Programming Modern HPC Platforms

1: Task-based Parallel Programming for HPC
2: StarPU, A Unified Runtime for Heterogeneous Architectures

Olivier Aumage, Team STORM
Inria – LaBRI
olivier.aumage@inria.fr

1

Task-based Parallel Programming for HPC

1.1

Runtime Systems

Why use runtime systems for HPC applications?

Examples of Runtime Systems

Examples of Runtime Systems

Graphics

- DirectX, Direct3D (Microsoft Windows)
- OpenGL

Examples of Runtime Systems

Graphics

- DirectX, Direct3D (Microsoft Windows)
- OpenGL

Networking

- **MPI** (Message Passing Interface), Global Arrays
- GASPI / GPI-2
- GASNet, CCI
- Distributed Shared Memory systems
- SHMEM

Examples of Runtime Systems

Graphics

- DirectX, Direct3D (Microsoft Windows)
- OpenGL

Networking

- **MPI** (Message Passing Interface), Global Arrays
- GASPI / GPI-2
- GASNet, CCI
- Distributed Shared Memory systems
- SHMEM

I/O

- MPI-IO
- HDF5 libraries
- Database engines

Why use runtime systems for HPC applications?

Why use runtime systems for HPC applications?

The Role(s) of Runtime Systems:

- Portability

Why use runtime systems for HPC applications?

The Role(s) of Runtime Systems:

- Portability
- Control

Why use runtime systems for HPC applications?

The Role(s) of Runtime Systems:

- Portability
- Control
- Adaptiveness

Why use runtime systems for HPC applications?

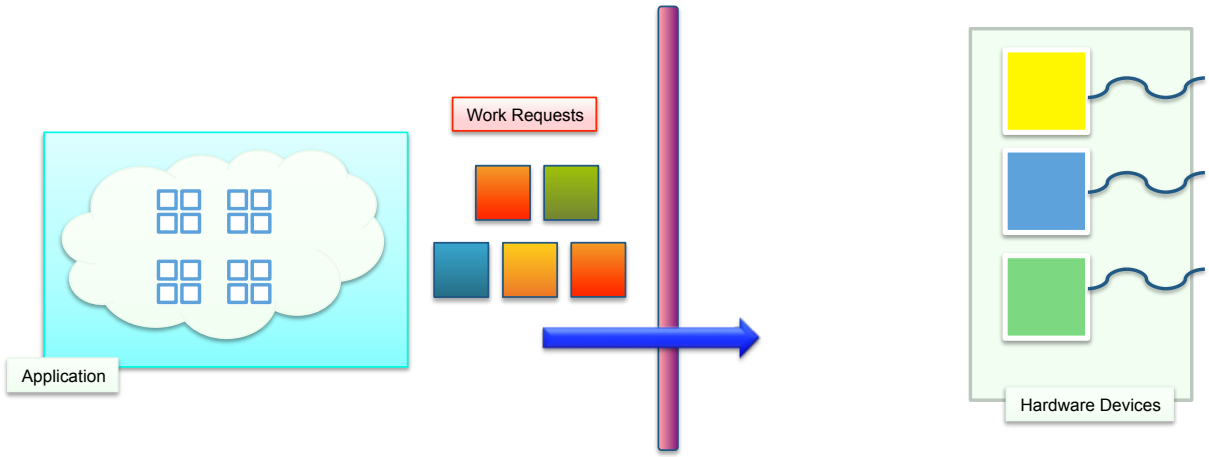
The Role(s) of Runtime Systems:

- Portability
- Control
- Adaptiveness
- Optimization

The Role(s) of Runtime Systems: Portability

The Role(s) of Runtime Systems: Portability

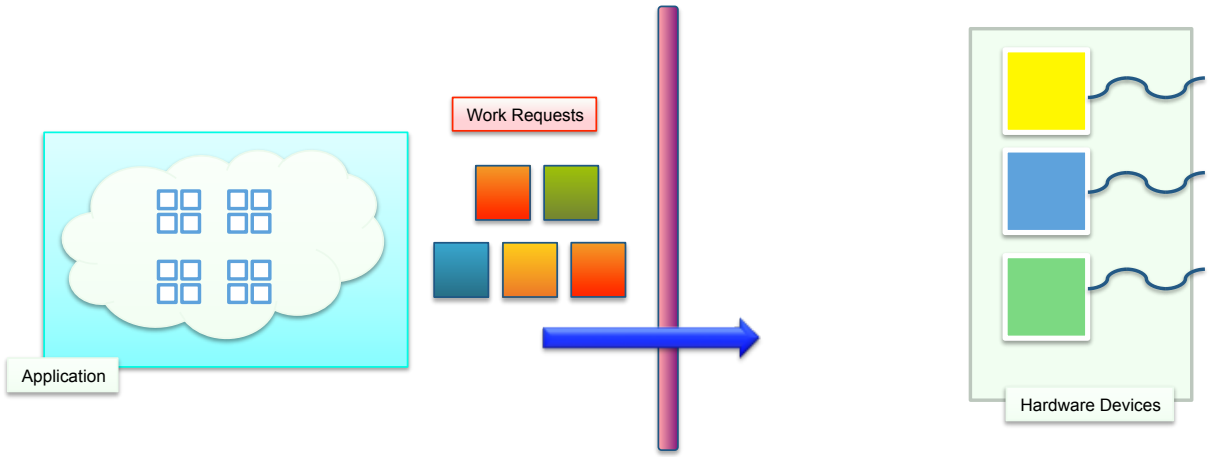
- Abstraction
  - Uniform front-end layer
  - Device-independent API
  - Targeted by applications



The Role(s) of Runtime Systems: Portability

- Abstraction
  - Uniform front-end layer
  - Device-independent API
  - Targeted by applications

- Drivers, plugins
  - Device-dependent backend layer
  - Targeted by vendors and/or device specialist

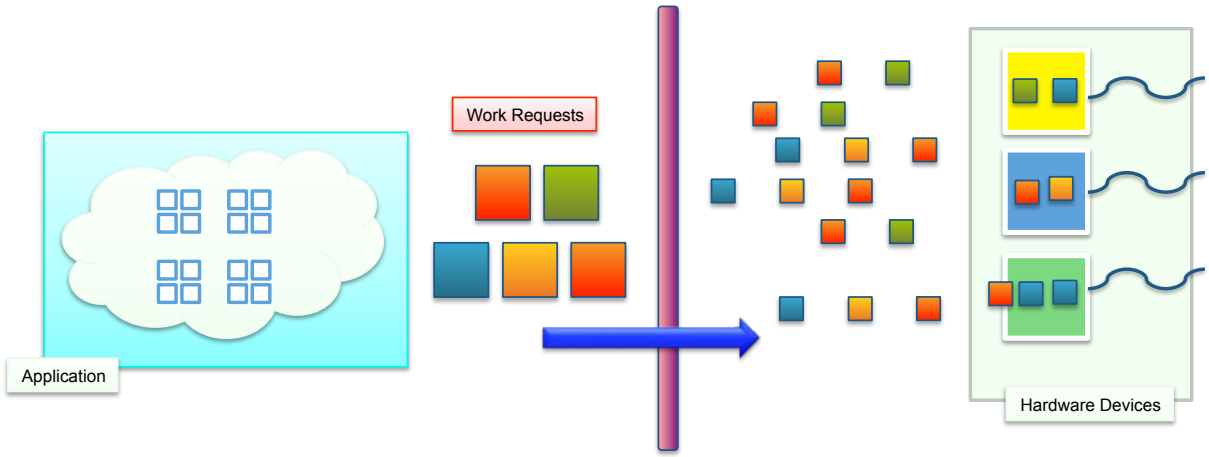


The Role(s) of Runtime Systems: Portability

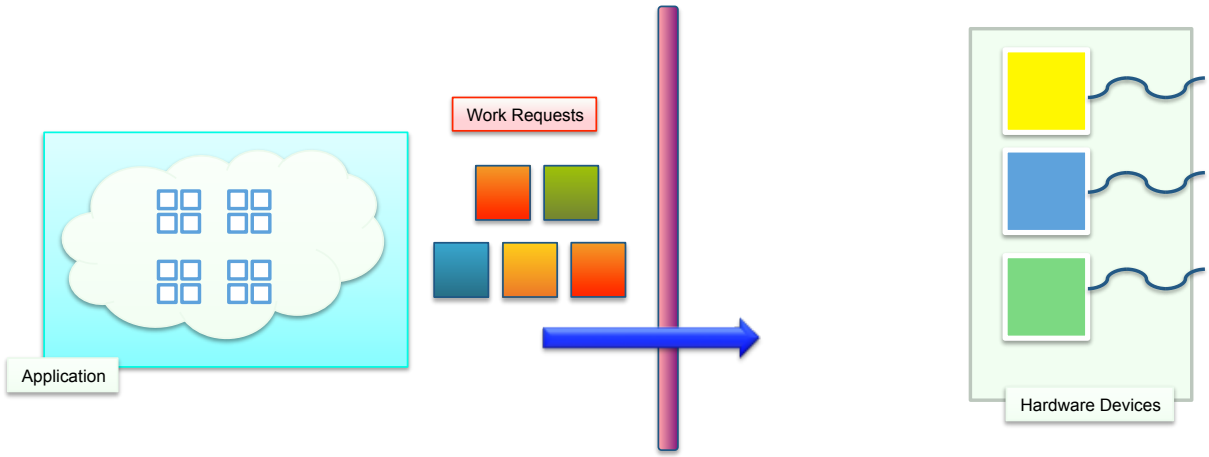
- Abstraction
  - Uniform front-end layer
  - Device-independent API
  - Targeted by applications

- Drivers, plugins
  - Device-dependent backend layer
  - Targeted by vendors and/or device specialist

- **Decoupling applications from device specific matters**

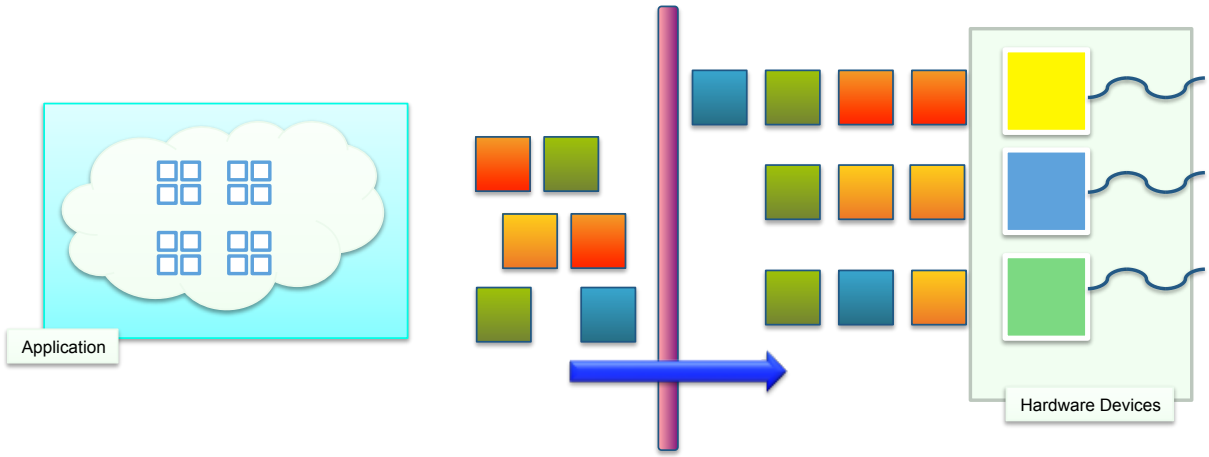


The Role(s) of Runtime Systems: Control



The Role(s) of Runtime Systems: Control

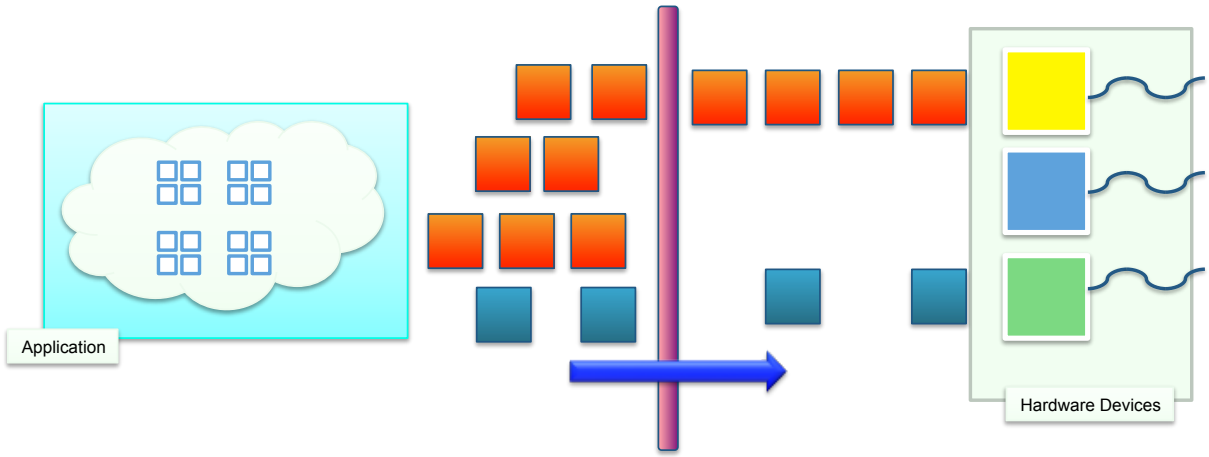
- Resource mapping
  - Deciding **which** hardware resource to use/not to use for some application workload
  - Spatial work mapping



The Role(s) of Runtime Systems: Control

- Resource mapping
  - Deciding **which** hardware resource to use/not to use for some application workload
  - Spatial work mapping

- Scheduling
  - Deciding **when** and in which order to perform some application workload
  - Temporal work mapping

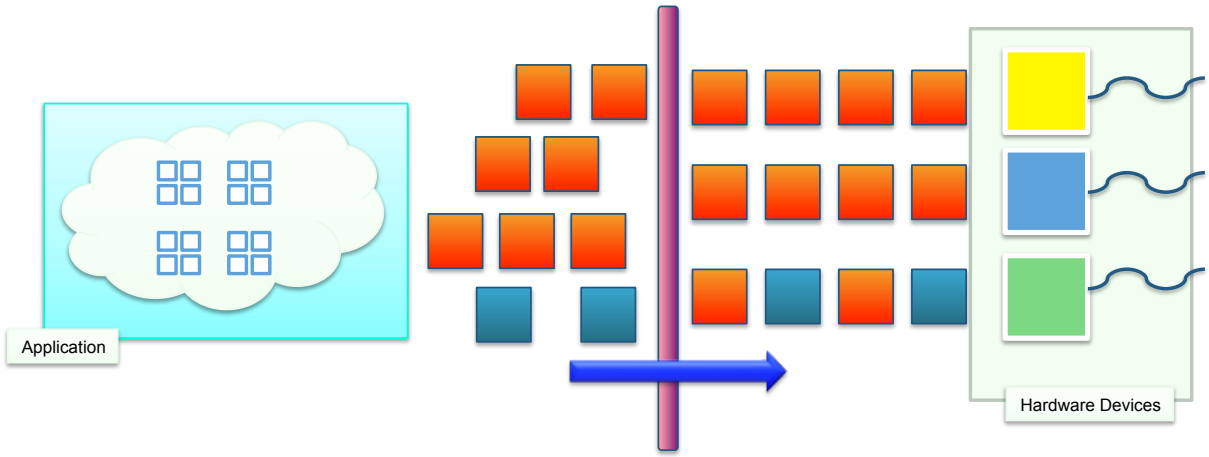


The Role(s) of Runtime Systems: Control

- Resource mapping
  - Deciding **which** hardware resource to use/not to use for some application workload
  - Spatial work mapping

- Scheduling
  - Deciding **when** and in which order to perform some application workload
  - Temporal work mapping

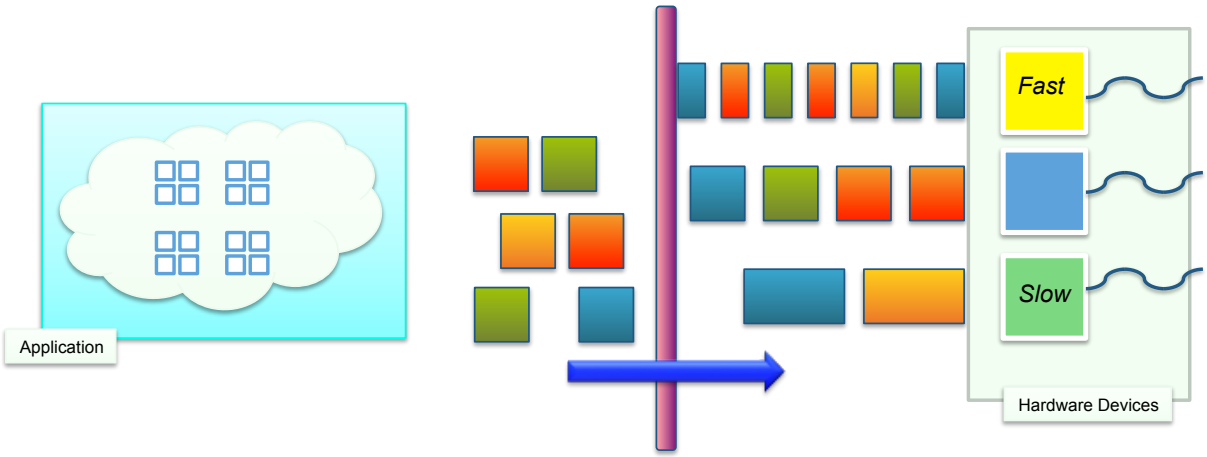
- **Plan application workload execution**



The Role(s) of Runtime Systems: Adaptiveness

The Role(s) of Runtime Systems: Adaptiveness

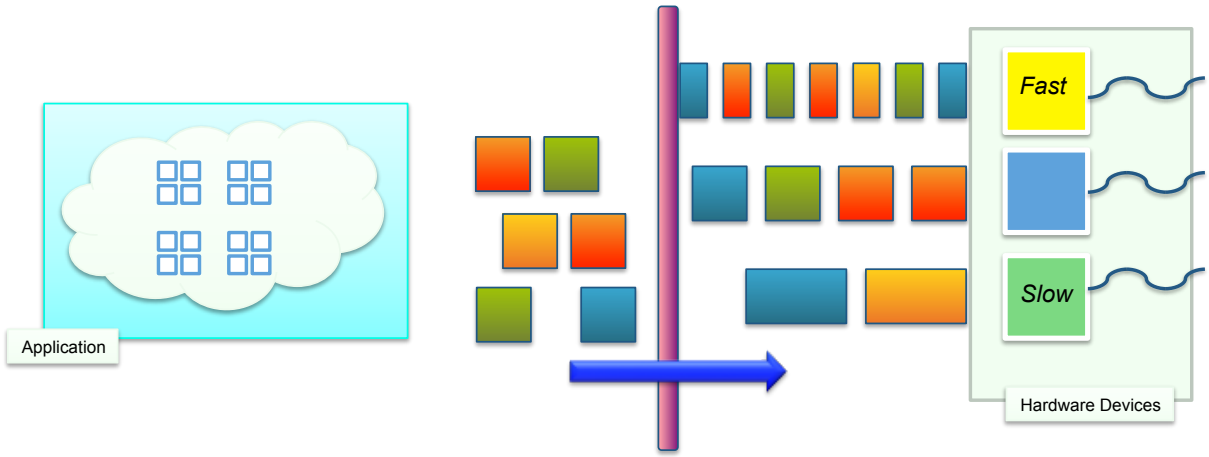
- Discovering, sampling, calibrating
  - Detecting **qualitative** hardware capabilities
  - Providing fallbacks, when possible
  - Detecting **quantitative** hardware capabilities



The Role(s) of Runtime Systems: Adaptiveness

- Discovering, sampling, calibrating
  - Detecting **qualitative** hardware capabilities
  - Providing fallbacks, when possible
  - Detecting **quantitative** hardware capabilities

- Monitoring, load balancing
  - Throttling workload feed
  - Reacting to hardware status changes

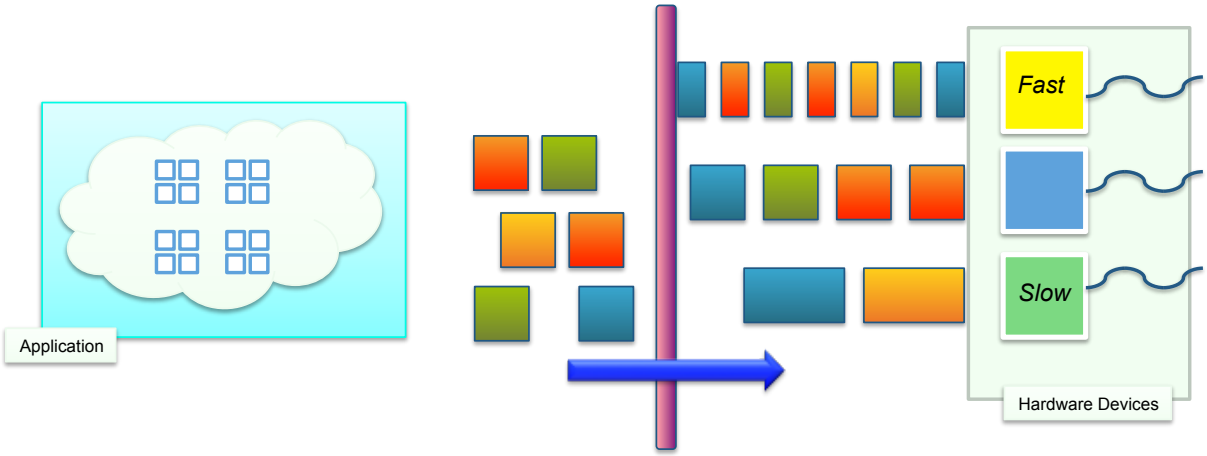


The Role(s) of Runtime Systems: Adaptiveness

- Discovering, sampling, calibrating
  - Detecting **qualitative** hardware capabilities
  - Providing fallbacks, when possible
  - Detecting **quantitative** hardware capabilities

- Monitoring, load balancing
  - Throttling workload feed
  - Reacting to hardware status changes

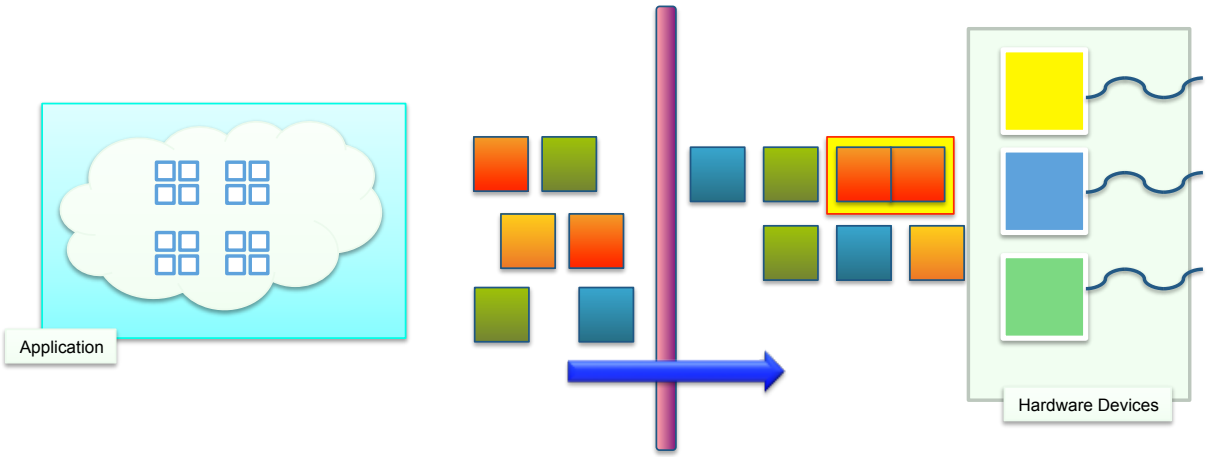
- **Cope with effective hardware aptitude and performance level**



The Role(s) of Runtime Systems: Optimization

The Role(s) of Runtime Systems: Optimization

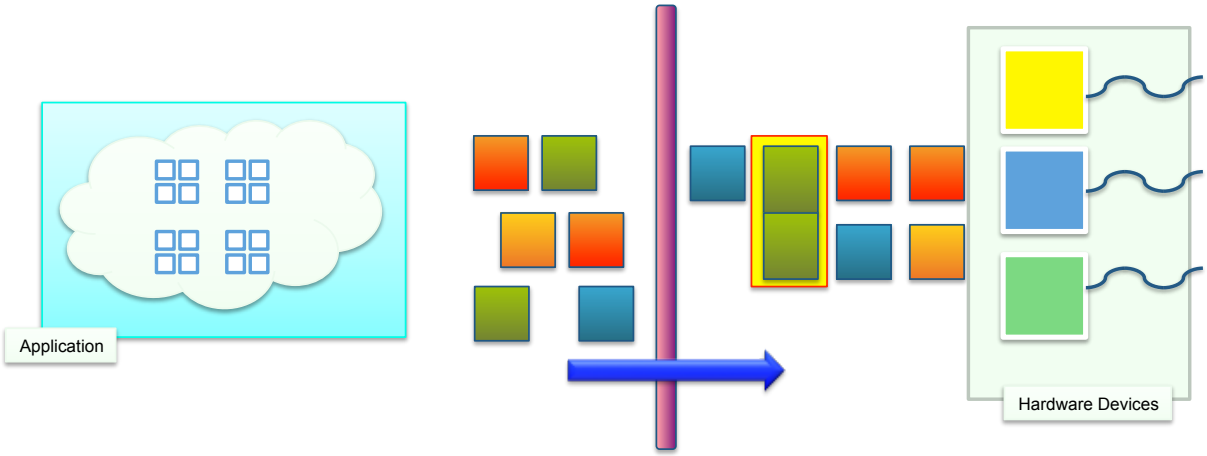
- Capitalize on workload look-ahead to bring performance-oriented added value
  - Requests aggregation
  - Resource locality
  - Computation offload
  - Computation/transfer overlap



The Role(s) of Runtime Systems: Optimization

- Capitalize on workload look-ahead to bring performance-oriented added value
  - Requests aggregation
  - Resource locality
  - Computation offload
  - Computation/transfer overlap

- Take advantage of the cross-cutting point of view of the runtime system
  - Perform global optimizations when possible

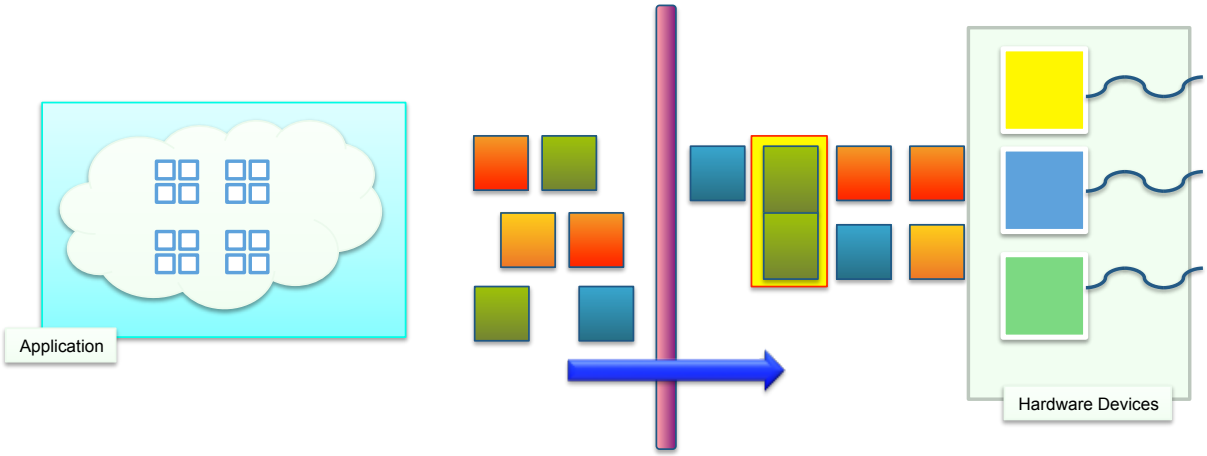


The Role(s) of Runtime Systems: Optimization

- Capitalize on workload look-ahead to bring performance-oriented added value
  - Requests aggregation
  - Resource locality
  - Computation offload
  - Computation/transfer overlap

- Take advantage of the cross-cutting point of view of the runtime system
  - Perform global optimizations when possible

- **Out-weight the cost of an extra, intermediate software layer**



1.2

Runtime Systems for Computing

Evolution of Computing Hardware

Rupture

- The “Frequency Wall”
  - Processing units cannot run anymore faster
- Looking for other sources of performance

Evolution of Computing Hardware

Rupture

- The "Frequency Wall"
  - Processing units cannot run anymore faster
- Looking for other sources of performance

Hardware Parallelism

- Multiply existing processing power
  - Have several processing units work together

Evolution of Computing Hardware

Rupture

- The "Frequency Wall"
  - Processing units cannot run anymore faster
- Looking for other sources of performance

Hardware Parallelism

- Multiply existing processing power
  - Have several processing units work together
- Not a new idea...
- ...but definitely the key performance factor now

Problematics

Unified computing runtime system for heterogeneous platforms

- Portability of performance
  - Abstraction
  - Adaptiveness
  - Execution Control
  - Optimization

Need a way to abstract application execution...

...into elementary, manageable objects

1.3

Abstracting Application Workload

Thread Scheduling

Reasoning on *Thread* objects?

Thread

- One instruction flow
  - Unbounded flow
  - Parallel activity
- One state/context per thread
  - Stack

Thread Scheduling

Reasoning on *Thread* objects?

Thread

- One instruction flow
  - Unbounded flow
  - Parallel activity
- One state/context per thread
  - Stack

- Examples
  - OpenMP parallel regions
  - libpthread
  - C++ threads

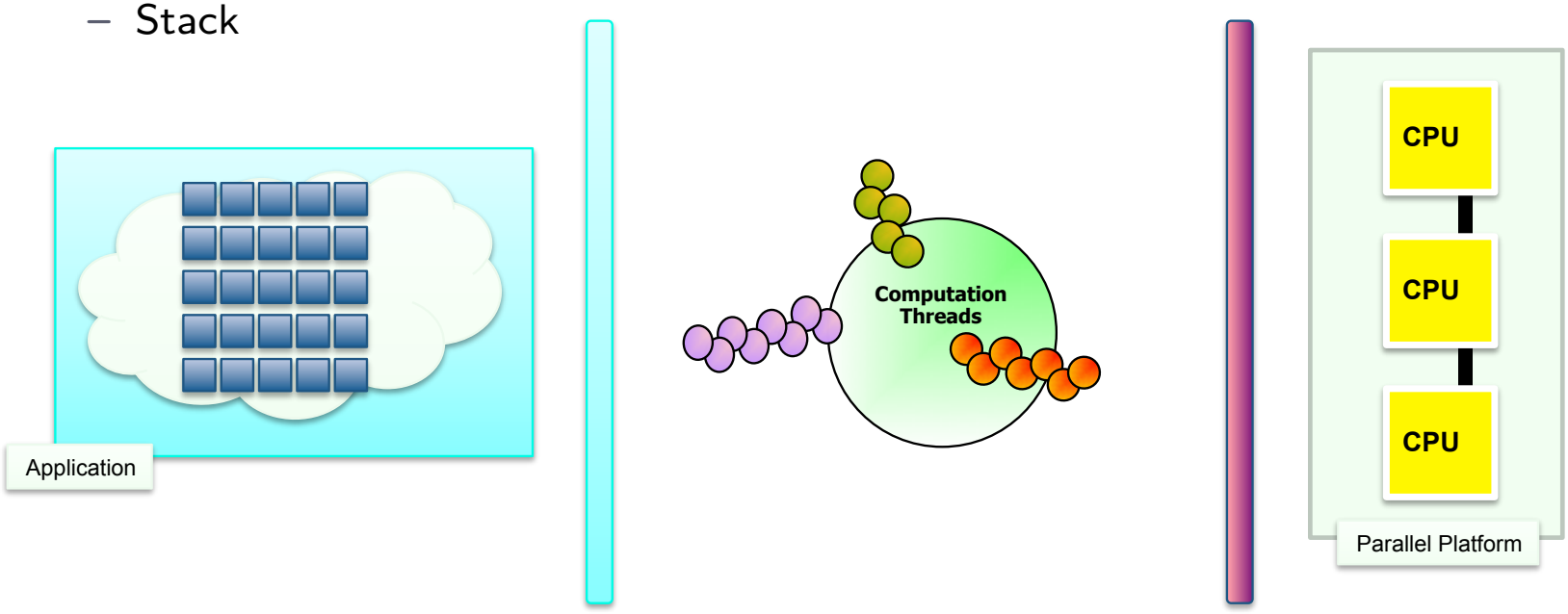
Thread Scheduling

Reasoning on *Thread* objects?

Thread

- One instruction flow
  - Unbounded flow
  - Parallel activity
- One state/context per thread
  - Stack

- Examples
  - OpenMP parallel regions
  - libpthread
  - C++ threads



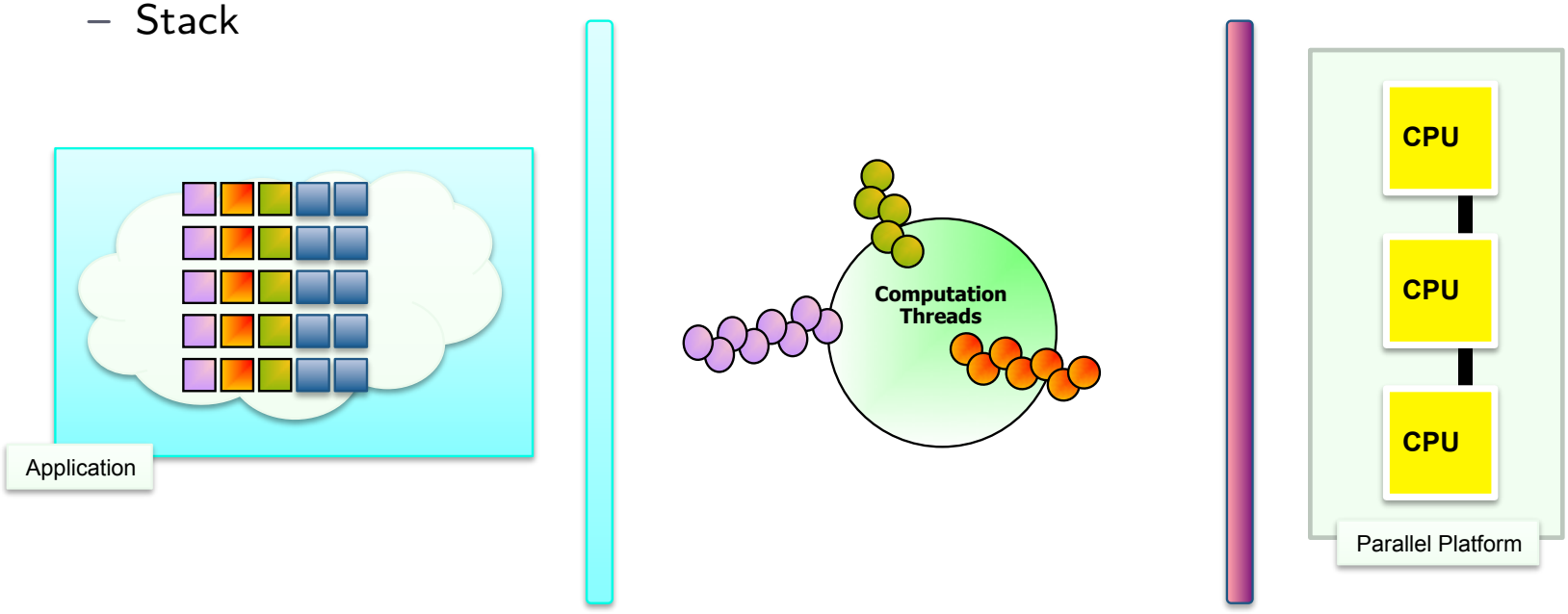
Thread Scheduling

Reasoning on *Thread* objects?

Thread

- One instruction flow
  - Unbounded flow
  - Parallel activity
- One state/context per thread
  - Stack

- Examples
  - OpenMP parallel regions
  - libpthread
  - C++ threads



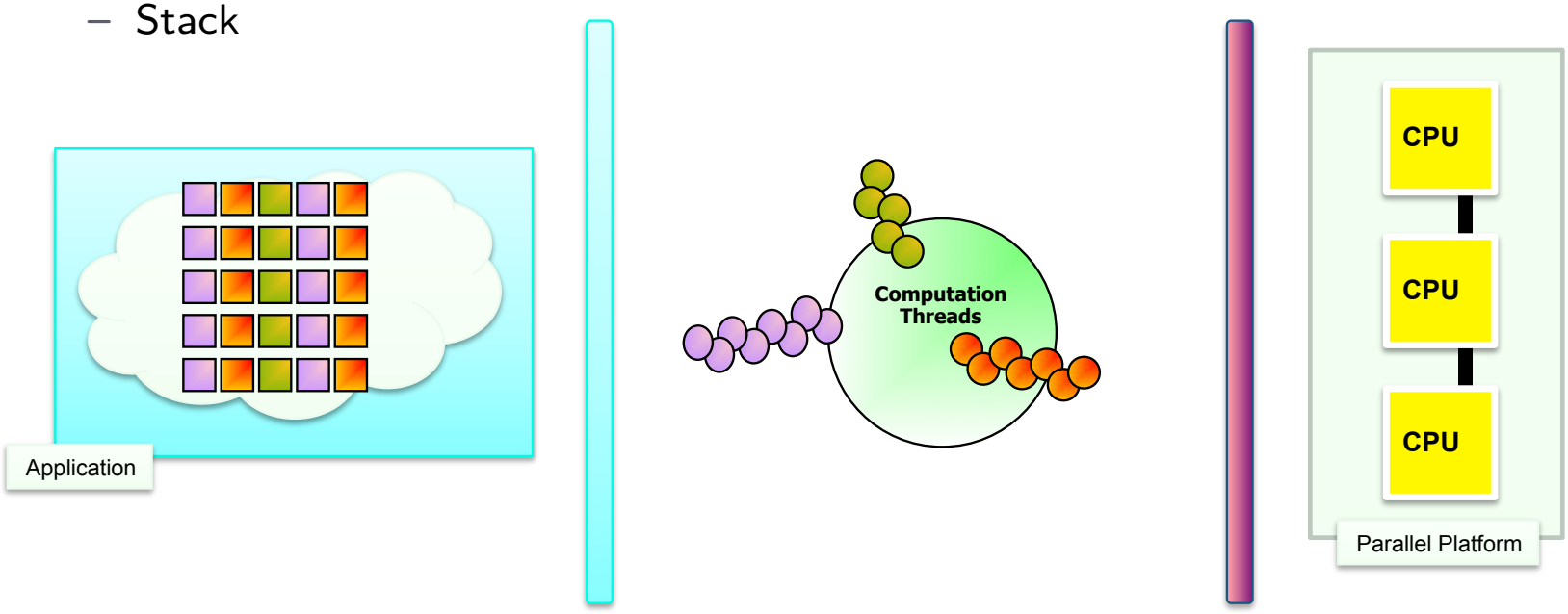
Thread Scheduling

Reasoning on *Thread* objects?

Thread

- One instruction flow
  - Unbounded flow
  - Parallel activity
- One state/context per thread
  - Stack

- Examples
  - OpenMP parallel regions
  - libpthread
  - C++ threads



Threads: Resources vs Needs

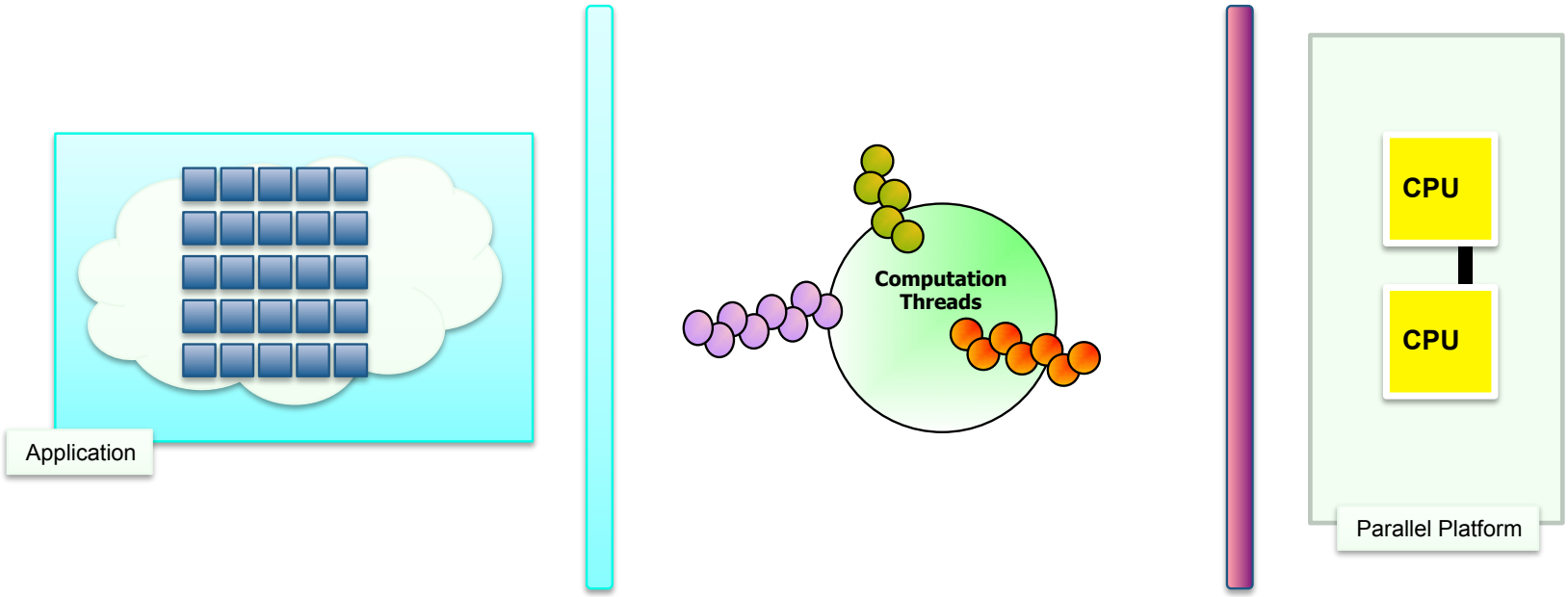
Lack of abstraction

- Threads express explicit resource request
- instead of application requirements

Threads: Resources vs Needs

Lack of abstraction

- Threads express explicit resource request
- instead of application requirements



Threads: Resources Miss-subscription

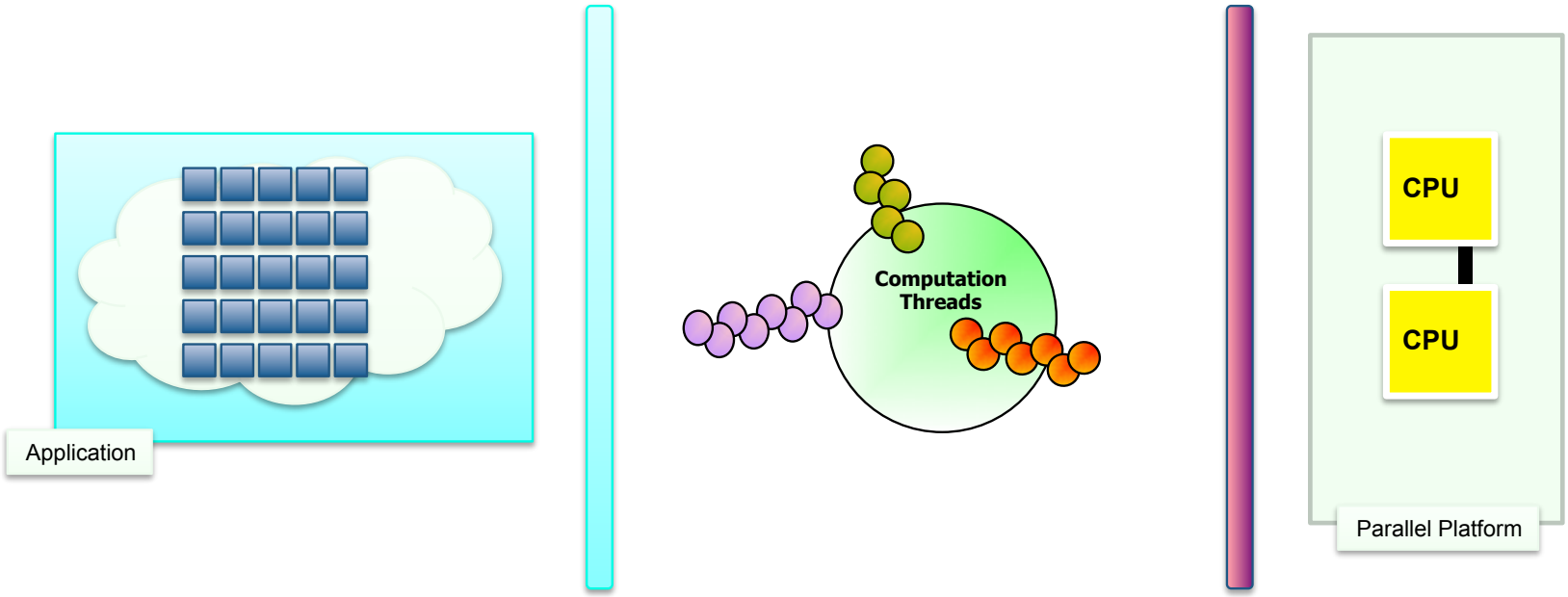
Software vs hardware mismatch

- Over-subscription
- Under-subscription
- Fixed number of threads

Threads: Resources Miss-subscription

Software vs hardware mismatch

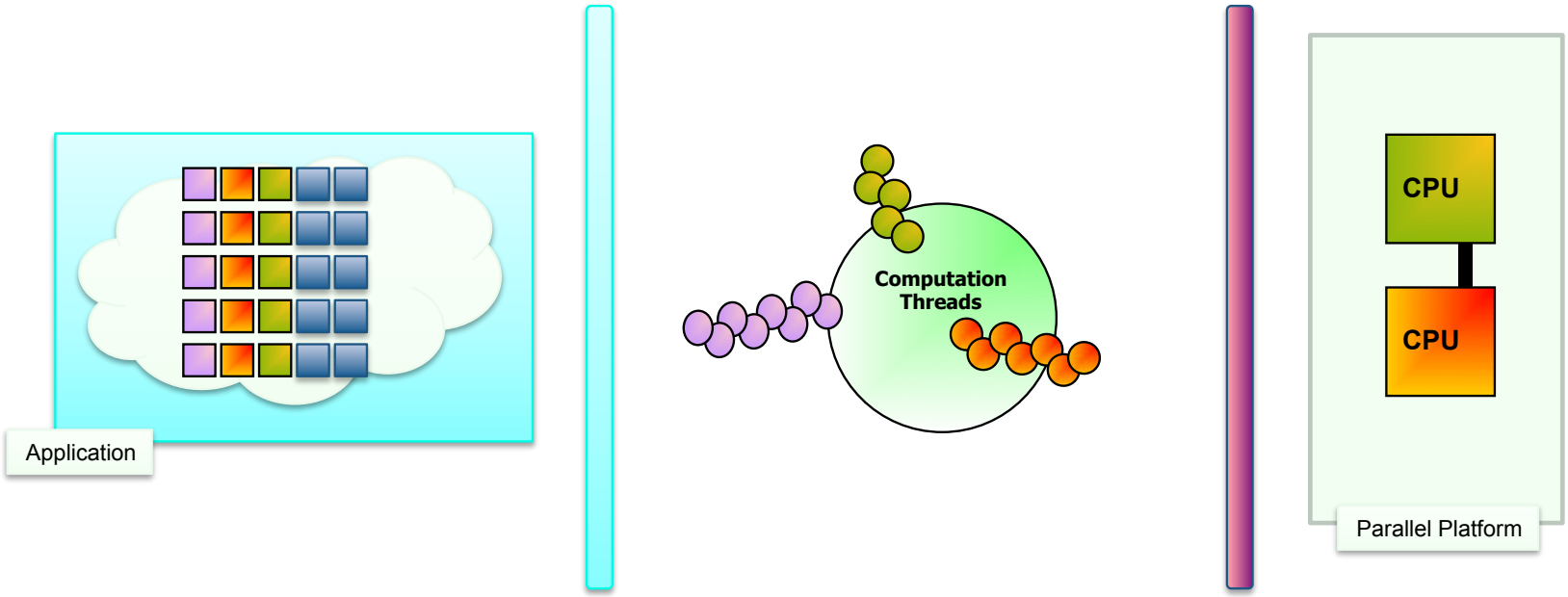
- Over-subscription
- Under-subscription
- Fixed number of threads



Threads: Resources Miss-subscription

Software vs hardware mismatch

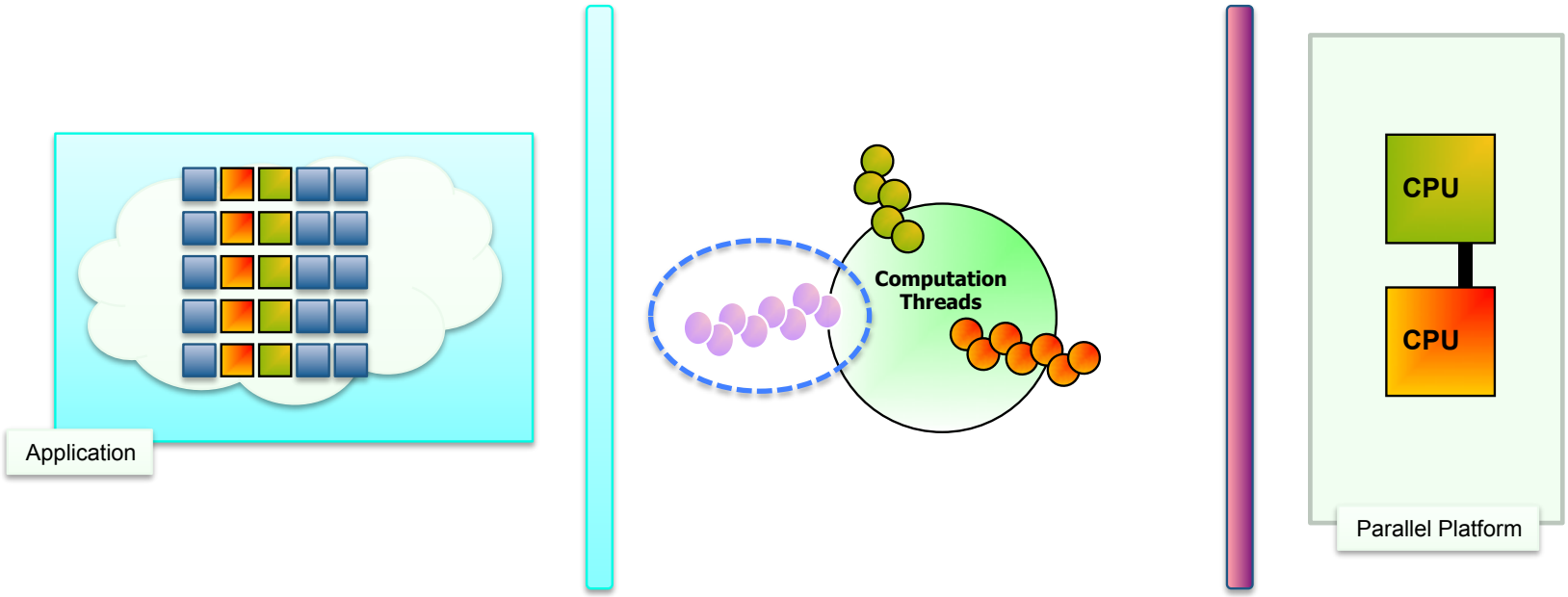
- Over-subscription
- Under-subscription
- Fixed number of threads



Threads: Resources Miss-subscription

Software vs hardware mismatch

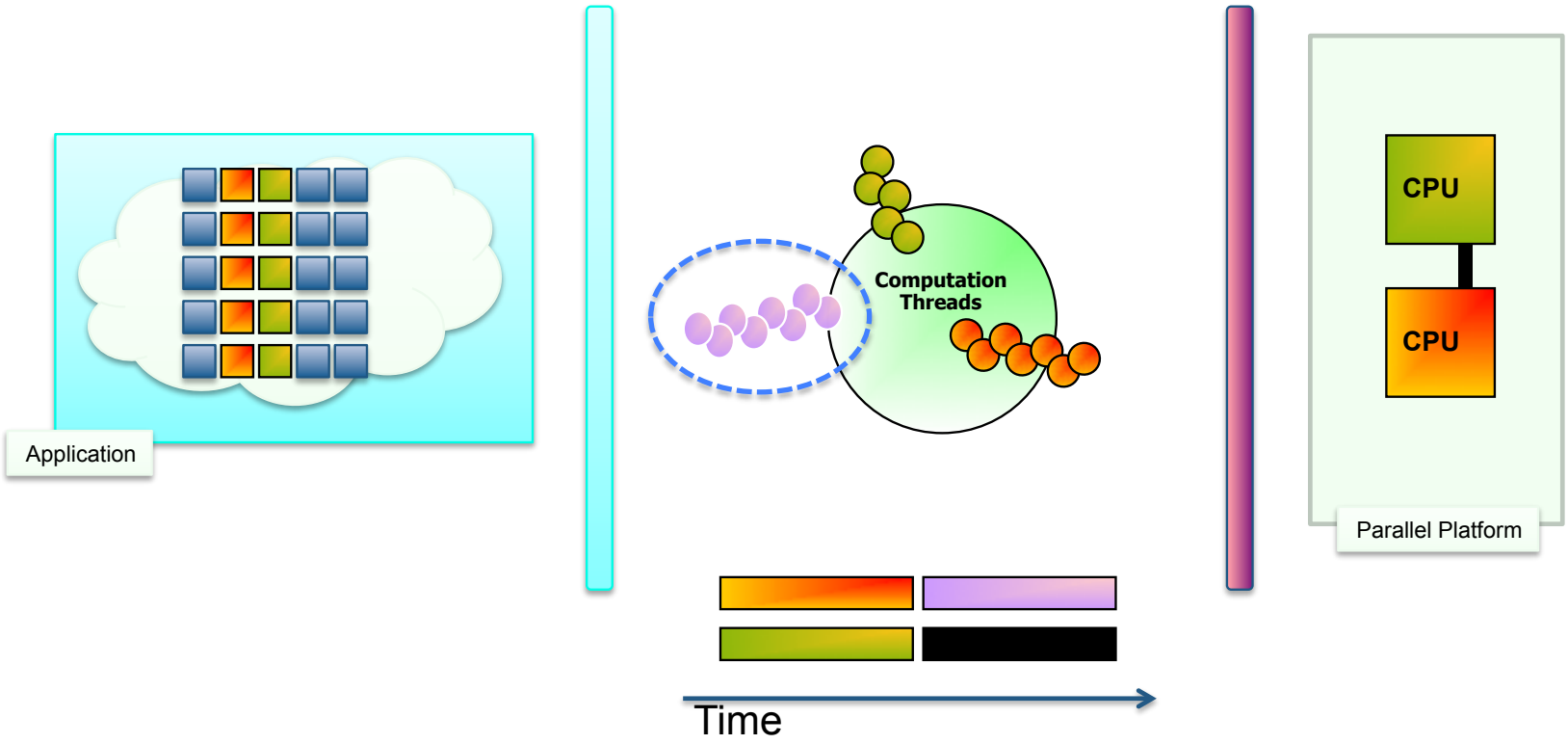
- Over-subscription
- Under-subscription
- Fixed number of threads



Threads: Resources Miss-subscription

Software vs hardware mismatch

- Over-subscription
- Under-subscription
- Fixed number of threads



Threads: Lack of Semantics

What does a thread really do?

- Resource usage?
- Inter-thread constraints
- Chaining constraints, ordering?

Planning Issues

- Unbounded computation
- System-controlled context switches

Consequences

- Heavy synchronizations: barriers
- User-managed fine-grain synchronizations: locks, mutexes
- Little to no help from runtime system

Threads: Load Balancing Issues

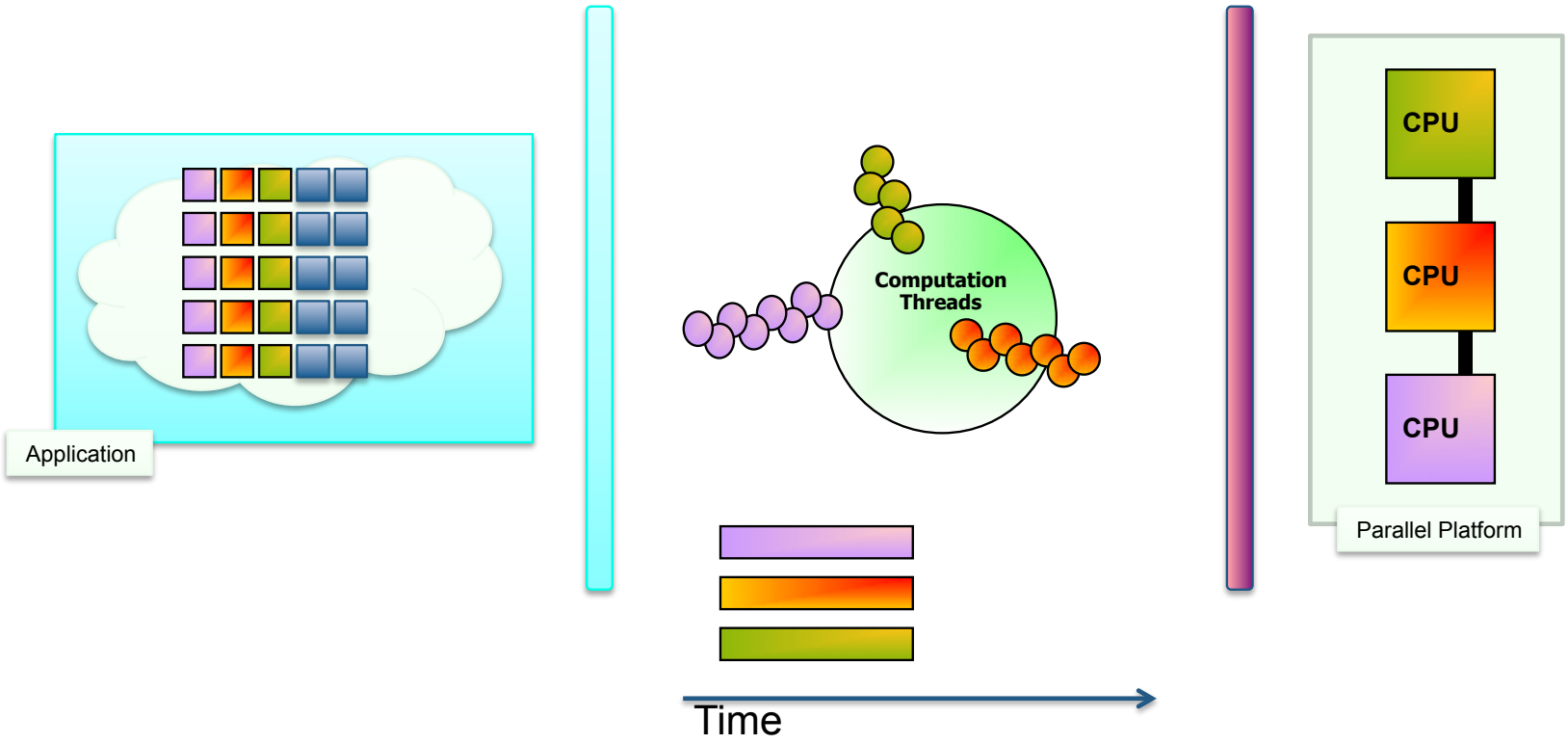
Keeping every hardware unit busy

- Irregular application, workload
- Uncontrolled synchronization shift
- Heterogeneous platforms: accelerators, GPU

Threads: Load Balancing Issues

Keeping every hardware unit busy

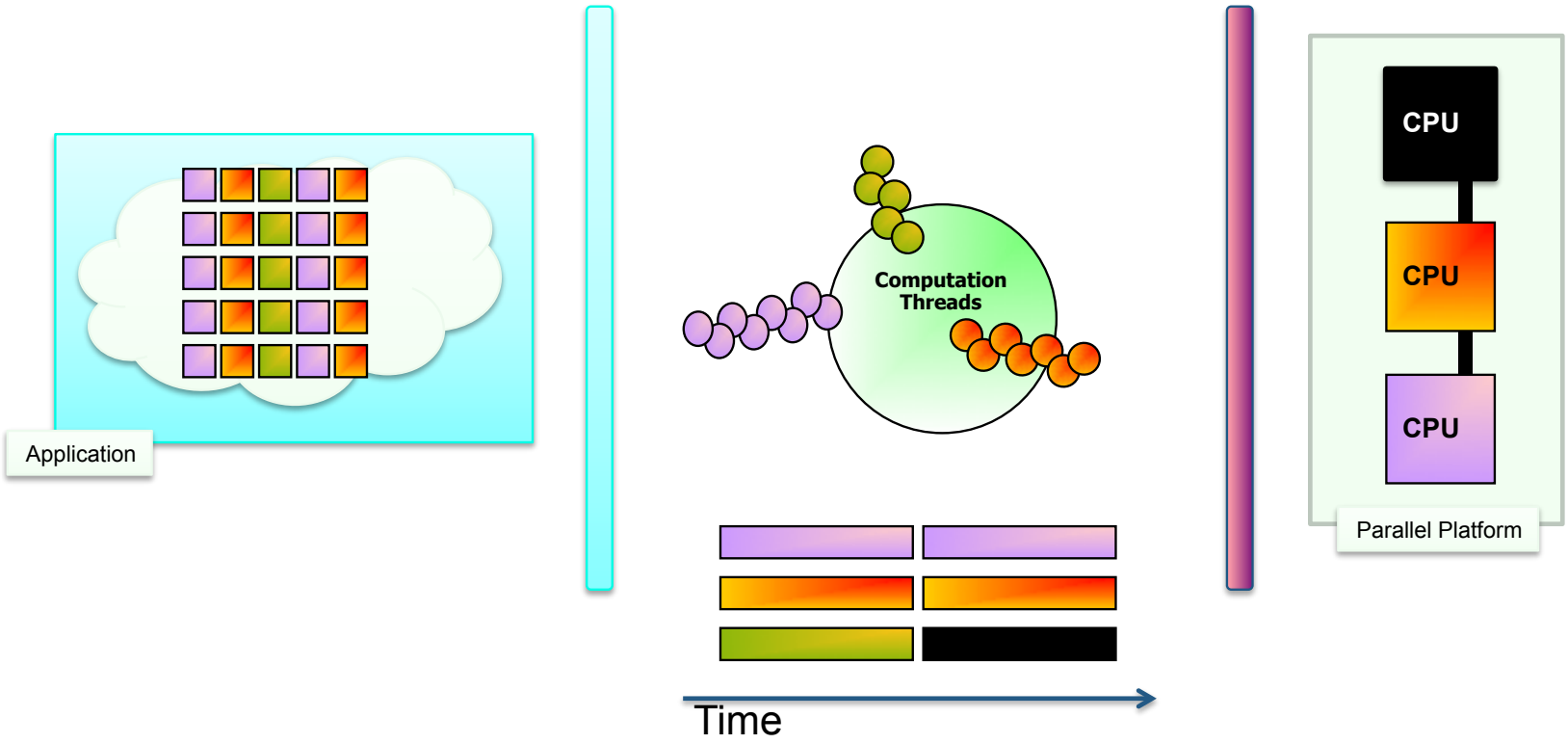
- Irregular application, workload
- Uncontrolled synchronization shift
- Heterogeneous platforms: accelerators, GPU



Threads: Load Balancing Issues

Keeping every hardware unit busy

- Irregular application, workload
- Uncontrolled synchronization shift
- Heterogeneous platforms: accelerators, GPU

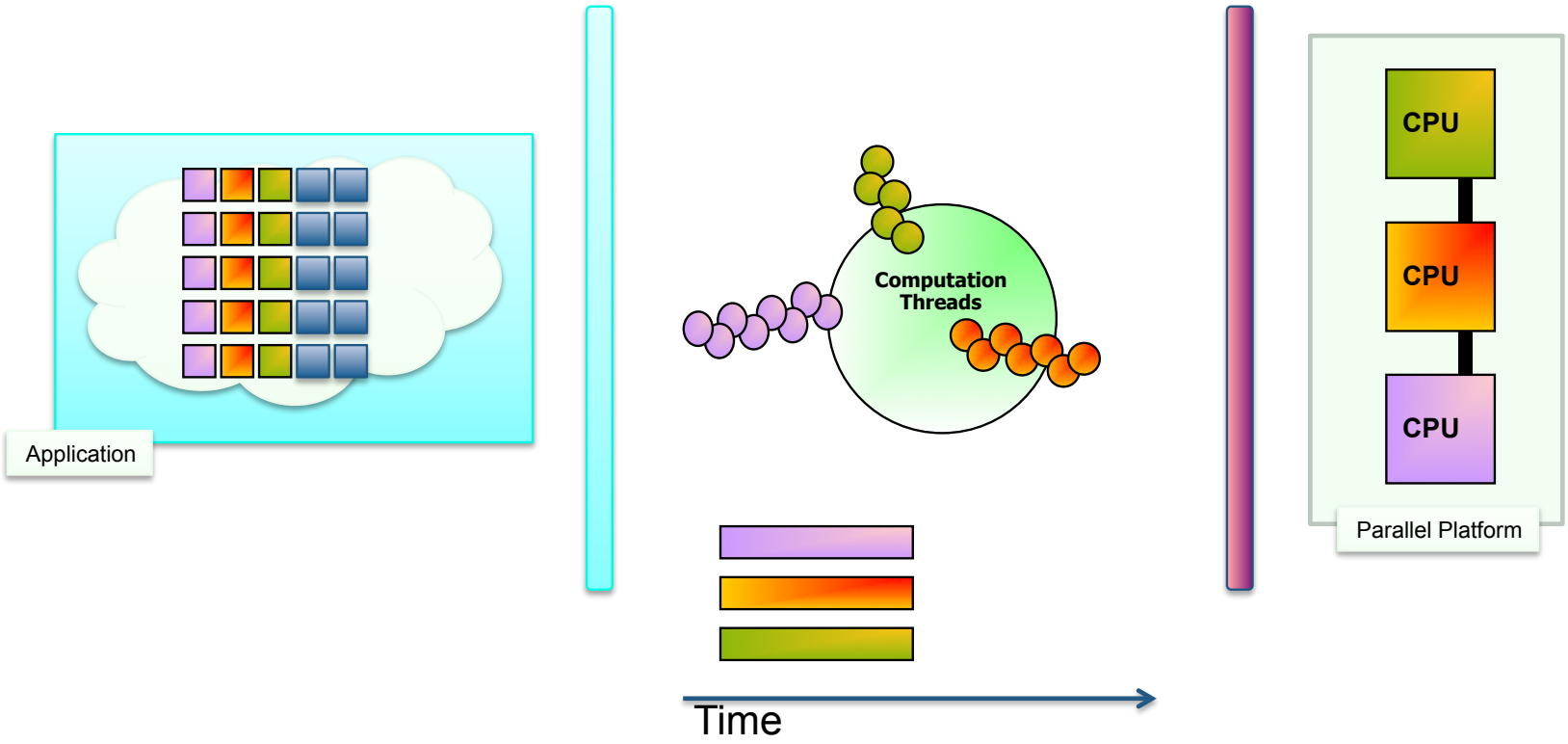


Threads: Networking and I/O Issues

- Computation/communication overlapping?
- Bulk I/O / network transfer mitigation?
- Thread-level idle time reduction?

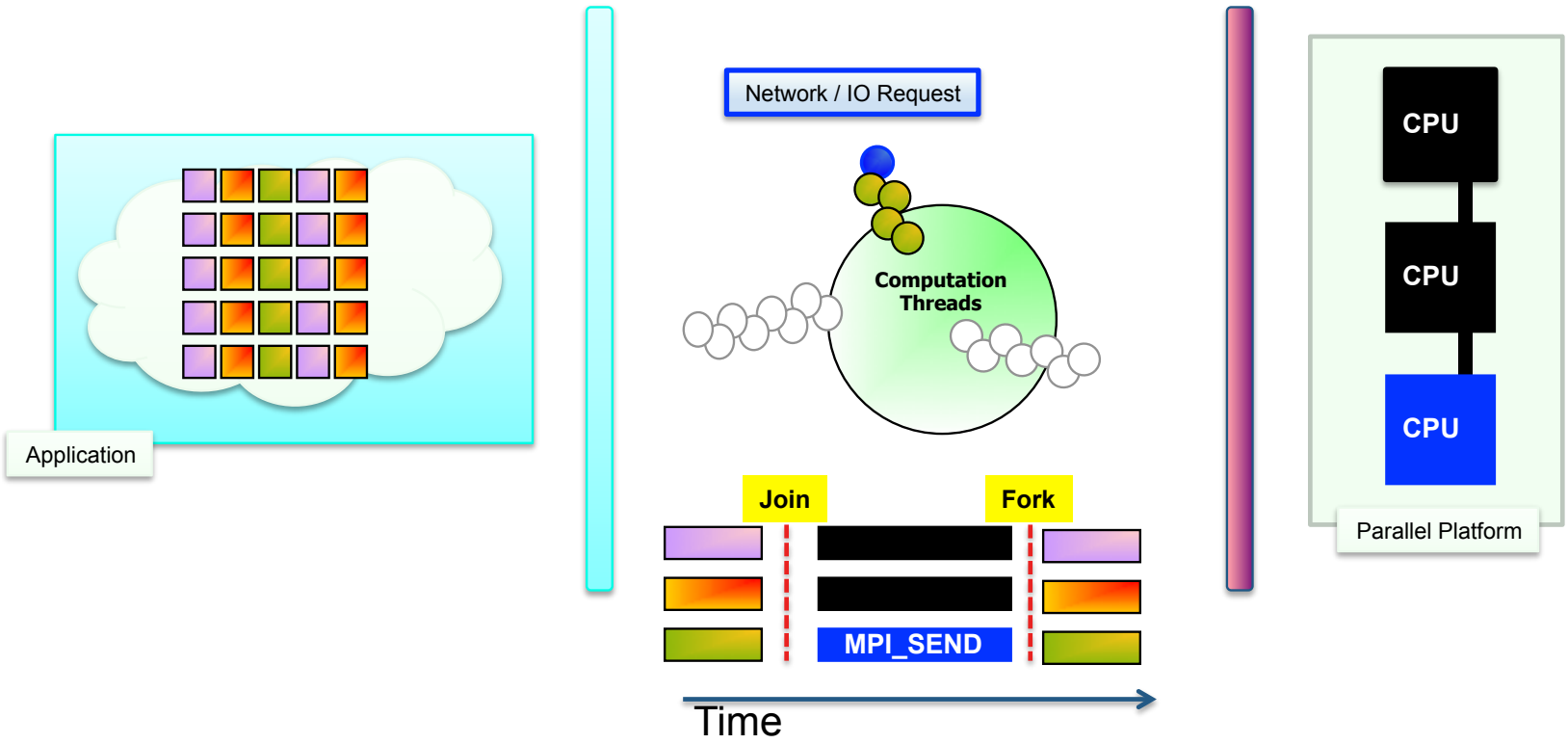
Threads: Networking and I/O Issues

- Computation/communication overlapping?
- Bulk I/O / network transfer mitigation?
- Thread-level idle time reduction?



Threads: Networking and I/O Issues

- Computation/communication overlapping?
- Bulk I/O / network transfer mitigation?
- Thread-level idle time reduction?



Threads: Outcome

Perhaps not the right semantics for end-user application development

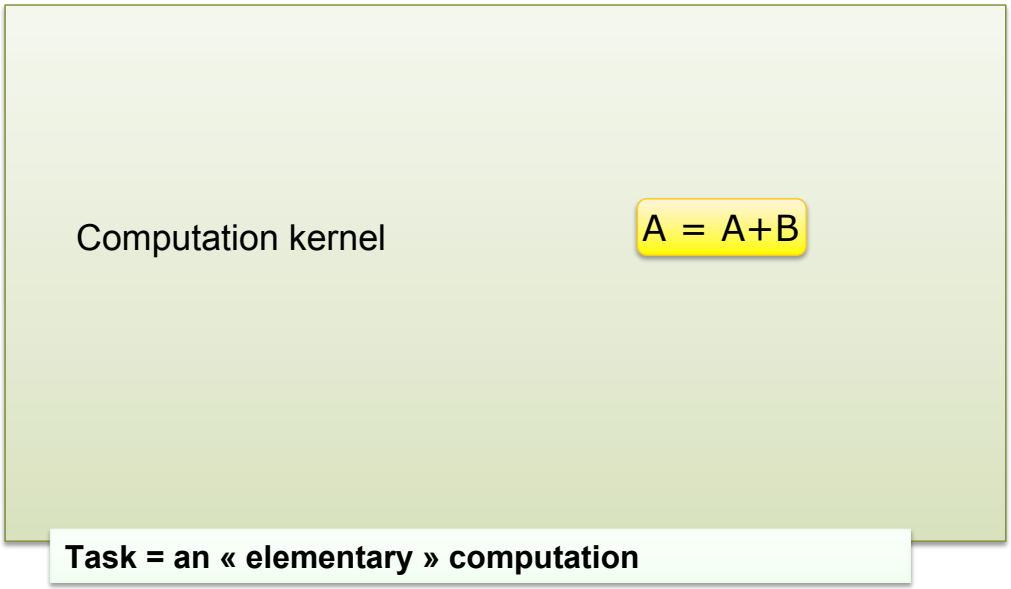
- Over-constrained concept for application programming
- Awkward object to manipulate at the runtime system level
- Not well suited to leverage theoretical scheduling results
  - Completion?
  - Other metrics?

Task Scheduling

Reasoning on *Task* objects

Common definition

- **Elementary computation**
  - Numerical kernel
  - BLAS call
  - ...
- → Potential parallel work

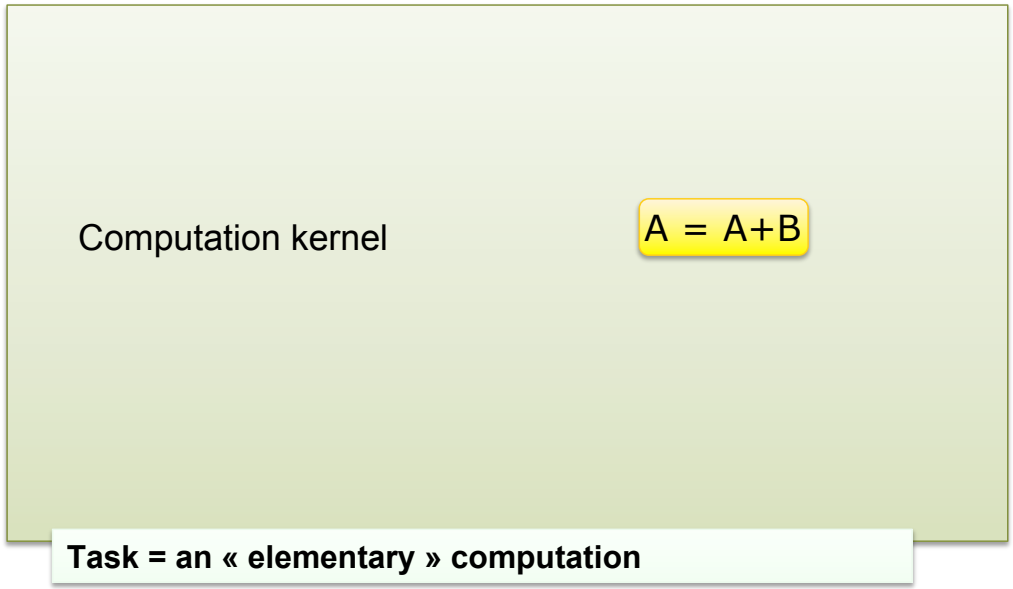


Task Scheduling

Reasoning on *Task* objects

Common definition

- **Elementary computation**
  - Numerical kernel
  - BLAS call
  - ...
- → Potential parallel work



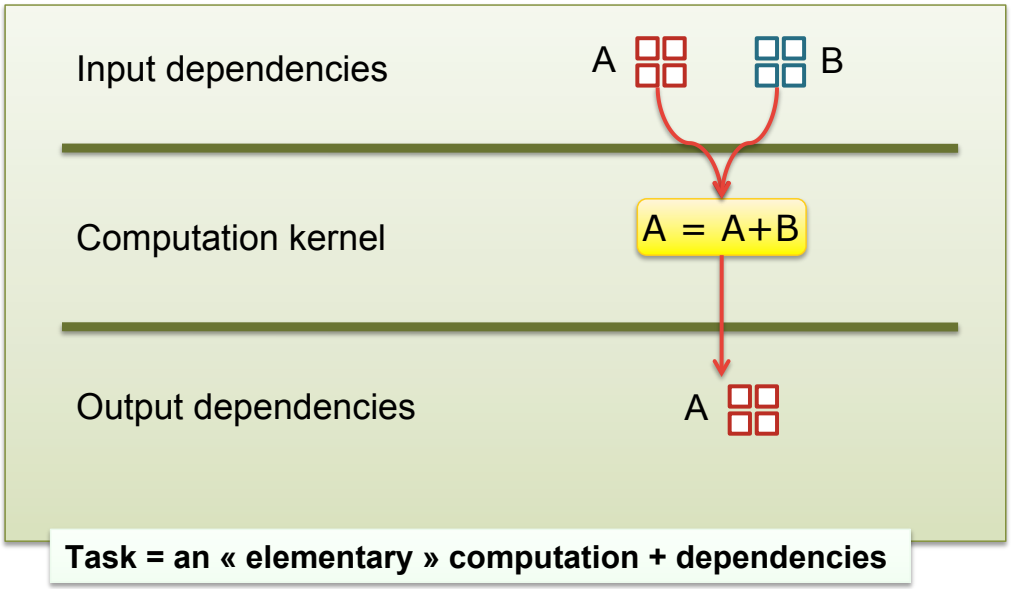
- **Shared (often fixed) pool of worker threads**
- → Decoupled engine, to realize a potentially parallel execution

Task Scheduling

Reasoning on *Task* objects

Common definition

- **Elementary computation**
  - Numerical kernel
  - BLAS call
  - ...
- → Potential parallel work



- **Shared (often fixed) pool of worker threads**
- → Decoupled engine, to realize a potentially parallel execution
- **Constraints** (with some programming models)
  - Input needed
  - Output produced
  - → Dependencies
  - *No side effect* (no hidden dependencies)
- → Degrees of Freedom in realizing the potential parallelism

Tasks: Resources vs Needs?

A task expresses **what** to do (e.g. which computation)

The runtime remains free to decide the amount of resources to execute a task

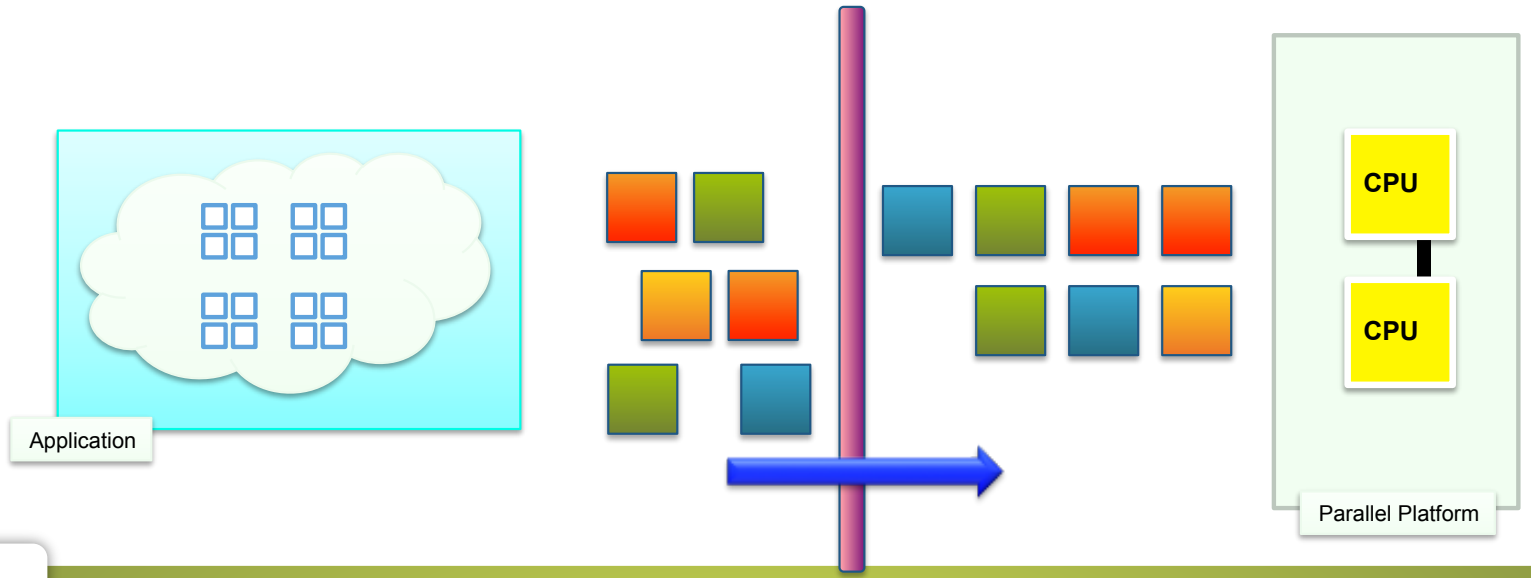
- Rationalize resource consumption
  - Thread and associated stack reused among several tasks
- Enforce separation of concerns
  - Management code brought out of the application
- Open the way to resource allocation optimization
  - Cross-cutting view of the application requirements

Tasks: Resources vs Needs?

A task expresses **what** to do (e.g. which computation)

The runtime remains free to decide the amount of resources to execute a task

- Rationalize resource consumption
  - Thread and associated stack reused among several tasks
- Enforce separation of concerns
  - Management code brought out of the application
- Open the way to resource allocation optimization
  - Cross-cutting view of the application requirements



Tasks: Resources Miss-subscription?

The runtime system may initialize a pool of worker threads according to the hardware capabilities

The application submit tasks independently to the runtime, independently of the hardware capabilities

- Tasks submitted by the application according to its natural algorithm
  - Abstraction with respect to hardware
- Workers allocated according to hardware resource, topology
  - Typically one thread per core or per hardware thread
- Operating system scheduler interference largely eliminated
  - No competition between worker threads

Tasks: Lack of Semantics?

A task expresses **what** to do (e.g. which computation), under **which** constraints.

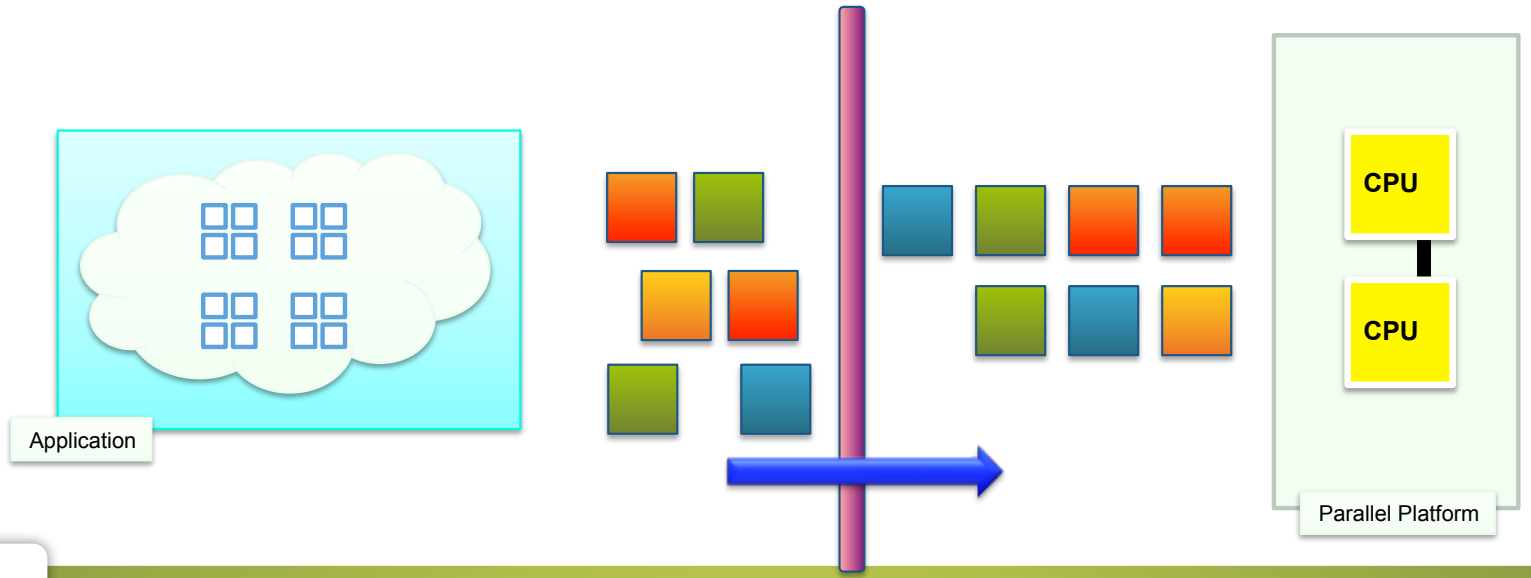
The runtime system can take advantage of this knowledge

Tasks: Lack of Semantics?

A task expresses **what** to do (e.g. which computation), under **which** constraints.

The runtime system can take advantage of this knowledge

- Optimize spatial resource usage
  - Decide which computing resource is best suited for a given task

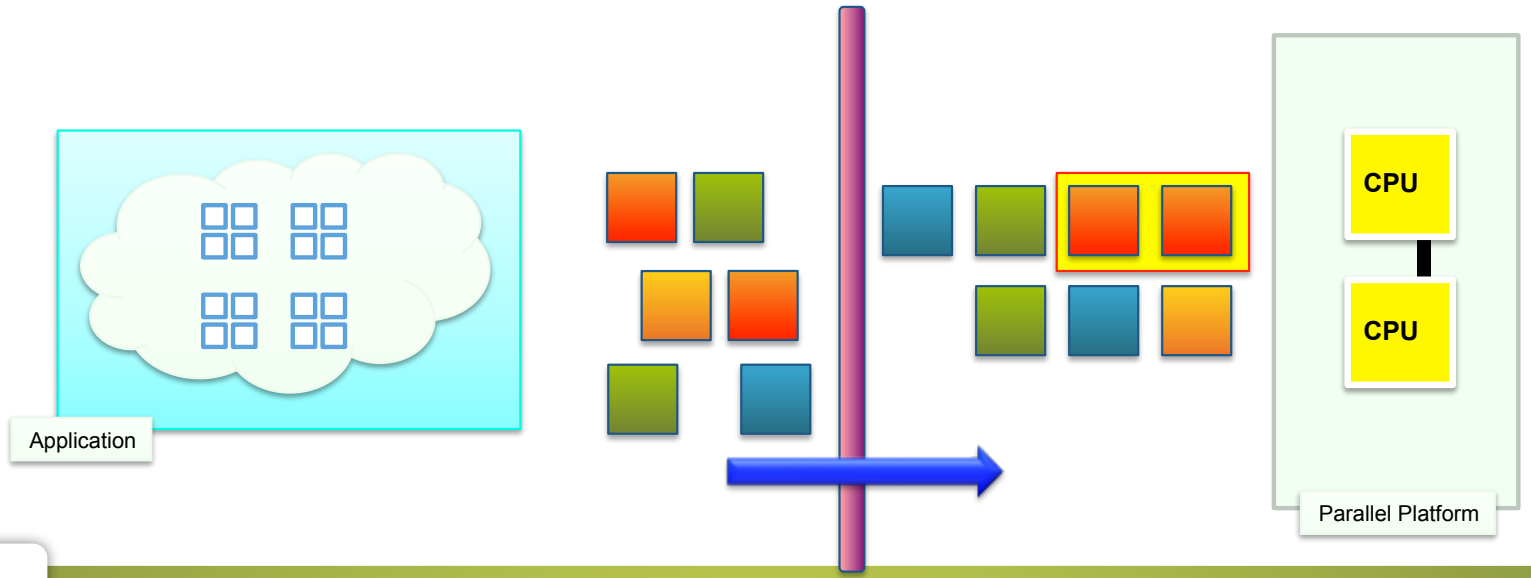


Tasks: Lack of Semantics?

A task expresses **what** to do (e.g. which computation), under **which** constraints.

The runtime system can take advantage of this knowledge

- Optimize spatial resource usage
  - Decide which computing resource is best suited for a given task
- Optimize temporal resource usage
  - Decide in which order to execute tasks

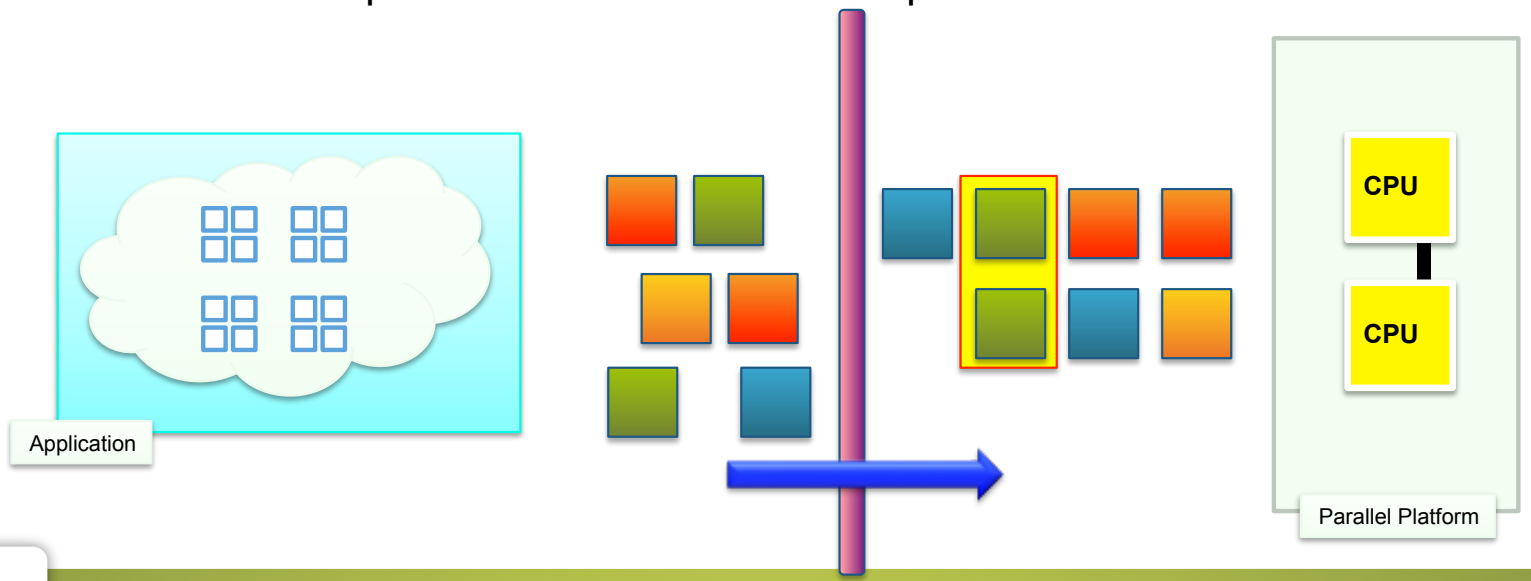


Tasks: Lack of Semantics?

A task expresses **what** to do (e.g. which computation), under **which** constraints.

The runtime system can take advantage of this knowledge

- Optimize spatial resource usage
  - Decide which computing resource is best suited for a given task
- Optimize temporal resource usage
  - Decide in which order to execute tasks
- Optimize concurrent resource usage
  - Decide which pairs of tasks to execute in parallel



Tasks: Lack of Semantics?

A task expresses **what** to do (e.g. which computation), under **which** constraints.

The runtime system can take advantage of this knowledge

- Optimize spatial resource usage
  - Decide which computing resource is best suited for a given task
- Optimize temporal resource usage
  - Decide in which order to execute tasks
- Optimize concurrent resource usage
  - Decide which pairs of tasks to execute in parallel
- No lock directly manipulated by the application

Tasks: Load Balancing Issues?

Tasks may transparently fill arising idle times as long as sufficient parallelism is available

The runtime system reacts to the situation observed at any time during the execution

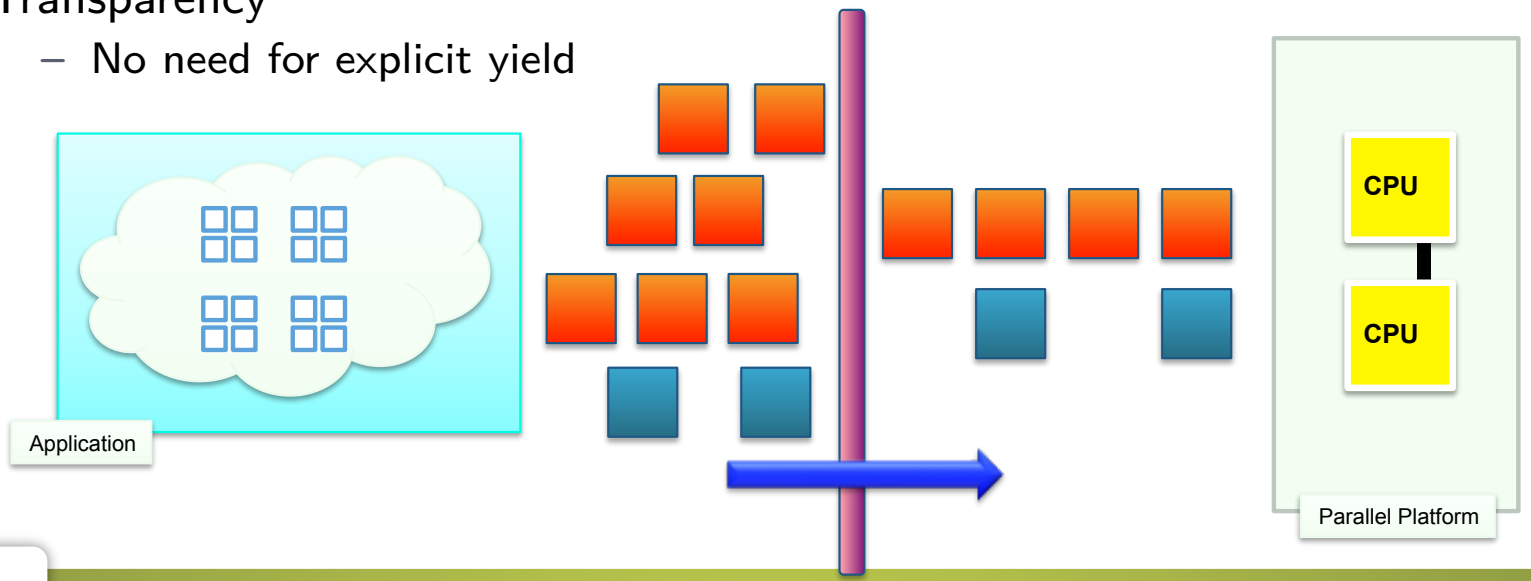
- Flexibility
  - No need for all tasks to have a uniform duration
  - Naturally opens the way to heterogeneous computations, accelerated offloads
- Transparency
  - No need for explicit yield

Tasks: Load Balancing Issues?

Tasks may transparently fill arising idle times as long as sufficient parallelism is available

The runtime system reacts to the situation observed at any time during the execution

- Flexibility
  - No need for all tasks to have a uniform duration
  - Naturally opens the way to heterogeneous computations, accelerated offloads
- Transparency
  - No need for explicit yield

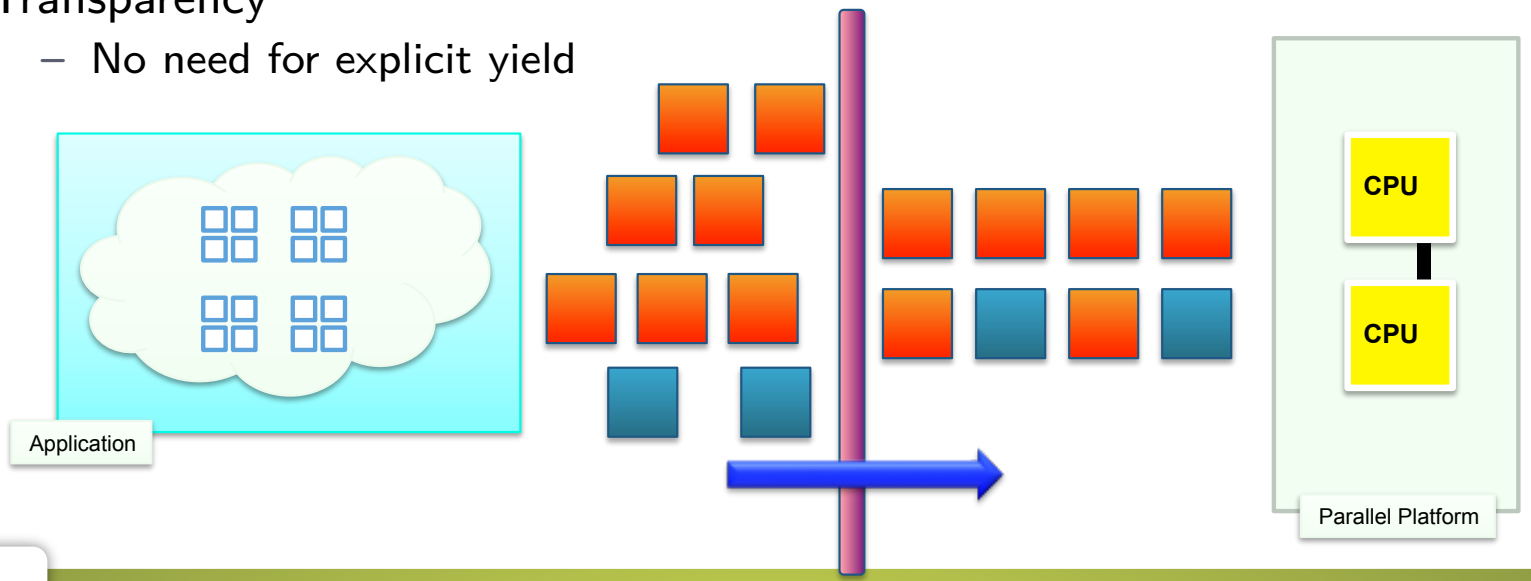


Tasks: Load Balancing Issues?

Tasks may transparently fill arising idle times as long as sufficient parallelism is available

The runtime system reacts to the situation observed at any time during the execution

- Flexibility
  - No need for all tasks to have a uniform duration
  - Naturally opens the way to heterogeneous computations, accelerated offloads
- Transparency
  - No need for explicit yield

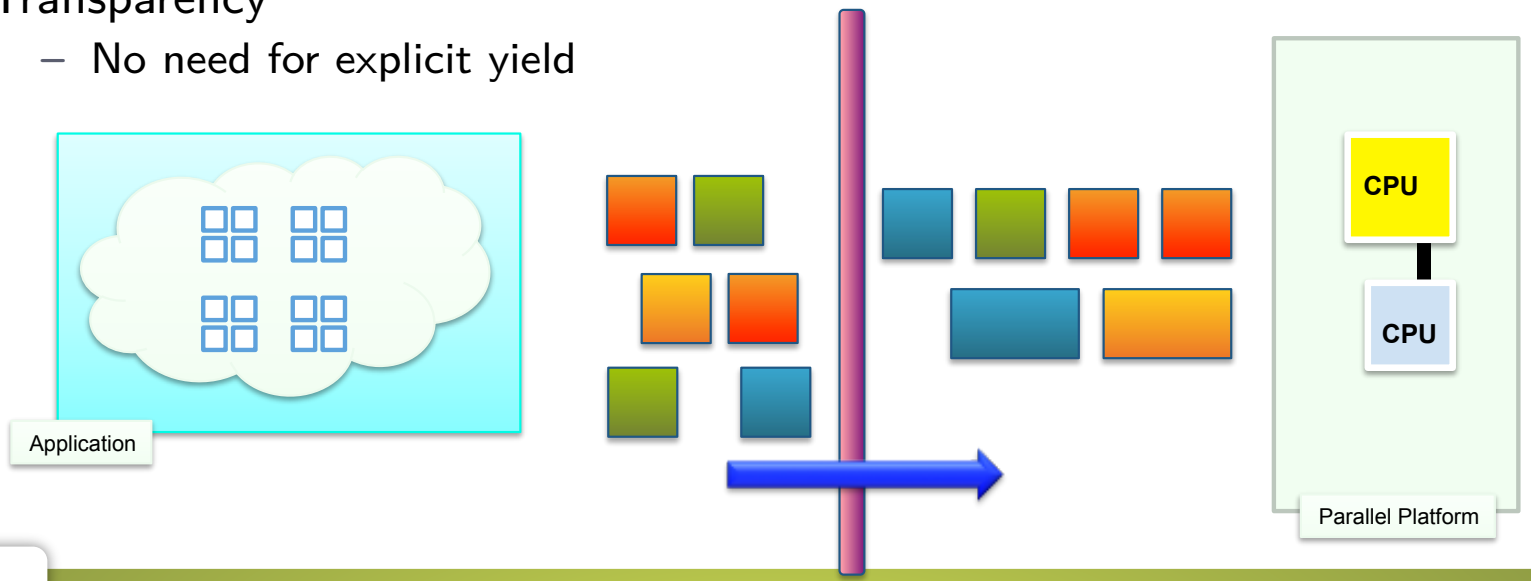


Tasks: Load Balancing Issues?

Tasks may transparently fill arising idle times as long as sufficient parallelism is available

The runtime system reacts to the situation observed at any time during the execution

- Flexibility
  - No need for all tasks to have a uniform duration
  - Naturally opens the way to heterogeneous computations, accelerated offloads
- Transparency
  - No need for explicit yield

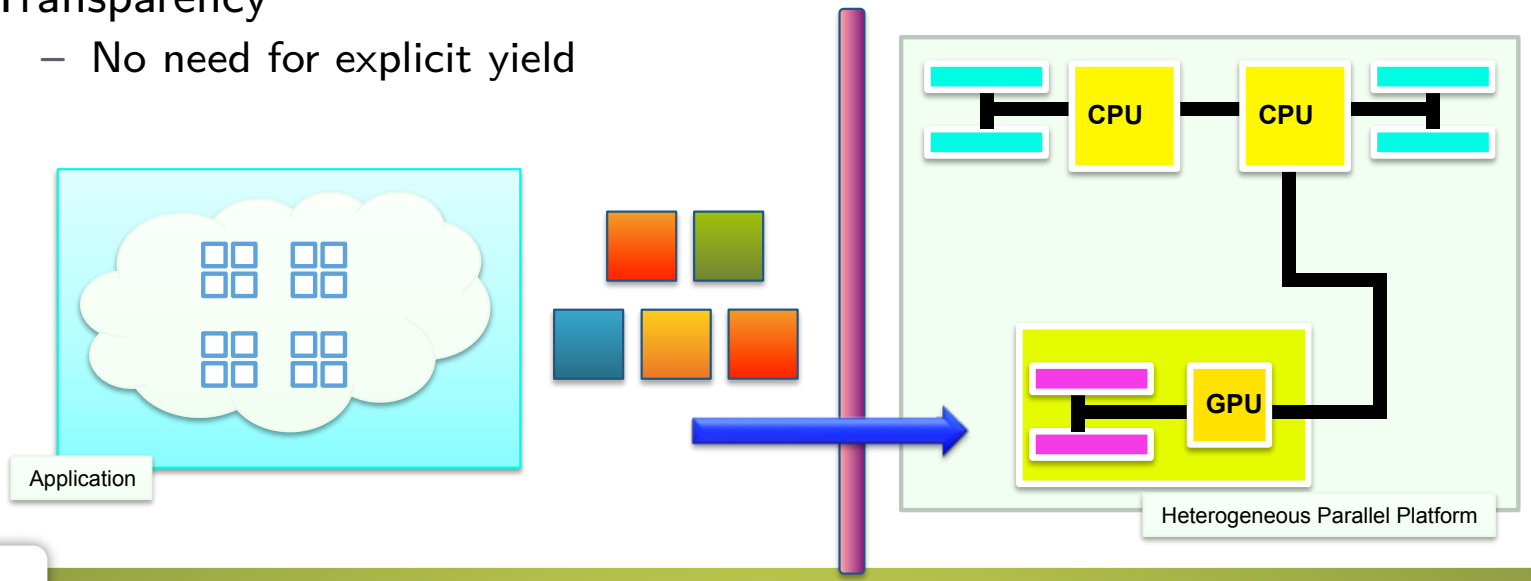


Tasks: Load Balancing Issues?

Tasks may transparently fill arising idle times as long as sufficient parallelism is available

The runtime system reacts to the situation observed at any time during the execution

- Flexibility
  - No need for all tasks to have a uniform duration
  - Naturally opens the way to heterogeneous computations, accelerated offloads
- Transparency
  - No need for explicit yield

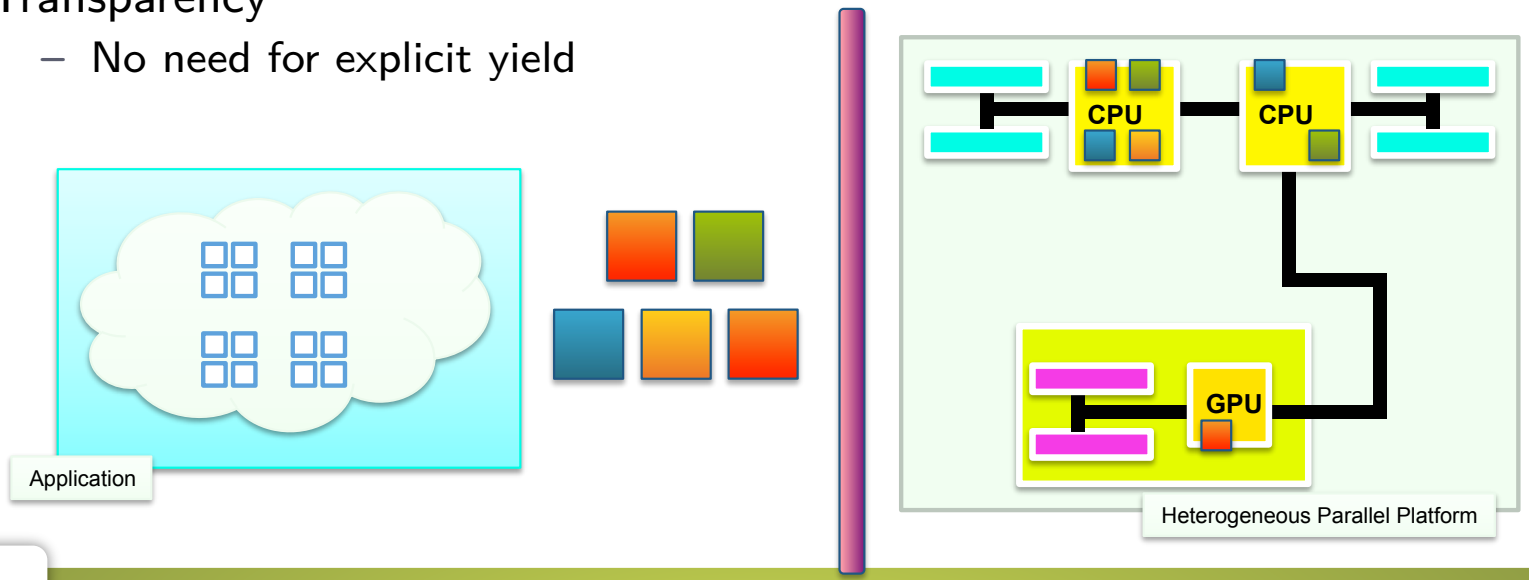


Tasks: Load Balancing Issues?

Tasks may transparently fill arising idle times as long as sufficient parallelism is available

The runtime system reacts to the situation observed at any time during the execution

- Flexibility
  - No need for all tasks to have a uniform duration
  - Naturally opens the way to heterogeneous computations, accelerated offloads
- Transparency
  - No need for explicit yield



Tasks: Networking and I/O Issues?

Potential 1-to-1 relationship between tasks and network/IO requests

- Network/IO request may start as soon as the task producing the data has been completed
- Tasks may be triggered as the result of network/IO requests completion
- Significant difference with fork-join models, MPI+X
  - Transparent interoperability
  - Avoid deferred network/IO requests until next join
  - Avoid custom network/IO requests management inside the application code

Tasks: Outcome

Task = Characterizable work

- **Well-defined**
  - Workload
  - Completion
  - Dependencies
  - **Similar to the pure function concept from Functional programming domain**
- **Suitable object for modelling**
  - Constraints
  - Degrees of freedom
  - **Large corpus of task scheduling theory**
- **Enforcing separation of concerns**
  - Application specialist
  - Kernel(s) specialist
  - Scheduling theoretician specialist
  - Runtime-system specialist

Programming Modern Platforms using Tasks

Expressing tasks?

- Divide and conquer: Cilk (recursive tasks)
- OpenMP 3.x and 4.x
- Dependencies compiler: PaRSEC (parameterized task graph)
- Sequential task flow: StarPU (directed acyclic task graph)

Programming Modern Platforms using Tasks

Expressing tasks?

- **Divide and conquer**: Cilk (recursive tasks)
- OpenMP 3.x and 4.x
- Dependencies compiler: PaRSEC (parameterized task graph)
- Sequential task flow: StarPU (directed acyclic task graph)

1.4

The Forerunner Task Model: Cilk

Cilk

The two "*all-time*" goals in parallel programming

- Programming parallel applications
  - **Easily**
- Running parallel applications
  - **Efficiently**

The **Cilk** language and framework played **an anticipative role** in reaching these goals for some classes of applications

Cilk

Cilk in a Few Words

- A programming environment
  - A **language** and compiler: keyword-based extension of C
  - An **execution** model and a run-time system

- Developed at the MIT
  - Supertech Research Group
  - *Charles E. Leiserson*'s team
  - Mid-90's

History

- Academic Era — `Cilk`
  - 1994: Cilk 1
  - 1998: **The Implementation of the Cilk-5 Multithreaded Language** paper by Matteo Frigo, Charles E. Leiserson, and Keith H. Randall, at PLDI'98
- Start-up Era — `Cilk++`
  - 2006: Launch of "Cilk Arts" company
  - 2008: Cilk++ version 1.0
- Intel Era — `Cilk Plus`
  - 2009: Intel acquires Cilk Arts
  - 2010: Intel Cilk Plus released as part of the Intel C++ Compiler
  - 2012: Release of the Cilk Plus support for the GNU GCC Compiler, implemented by Intel

Context

Middle of nineties

Hardware

- SMP: *Symmetric Multi-Processors*
- **Need for parallel programming models**

Context

Middle of nineties

Hardware

- SMP: *Symmetric Multi-Processors*
- **Need for parallel programming models**

Software

- Notion of threads: concurrent processing contexts within single process
- How to efficiently/easily express application parallelism using threads?

Context

Middle of nineties

Hardware

- SMP: *Symmetric Multi-Processors*
- **Need for parallel programming models**

Software

- Notion of threads: concurrent processing contexts within single process
- How to efficiently/easily express application parallelism using threads?

Program **Easily**?

- Parallel program quickly derived from sequential program
- Concurrency expressed safely (correctness, consistency)

Context

Middle of nineties

Hardware

- SMP: *Symmetric Multi-Processors*
- **Need for parallel programming models**

Software

- Notion of threads: concurrent processing contexts within single process
- How to efficiently/easily express application parallelism using threads?

Program **Easily**?

- Parallel program quickly derived from sequential program
- Concurrency expressed safely (correctness, consistency)

Run **Efficiently**?

- No over/under-subscription
- Load-balancing
- Low overhead

C Extension

Cilk is a faithful extension of C

- Keyword based language
  - Per opposition to pragma based languages (e.g. OpenMP)

C Extension

Cilk is a faithful extension of C

- Keyword based language
  - Per opposition to pragma based languages (e.g. OpenMP)
- Main keywords (original Cilk):
  - **cilk** : declaration of a potentially parallel routine
  - **spawn**: launch of a potentially parallel routine
  - **sync**: wait for completion of launched routines
  - **inlet** : special function to aggregate results (reduction)

```
1  #include <cilk.h>
2
3  cilk int fib (int n) {
4    if (n < 2) {
5      return n;
6    } else {
7      int x, y;
8      x = spawn fib (n-1);
9      y = spawn fib (n-2);
10     sync;
11     return x+y;
12   }
13 }
```

C Extension

Cilk is a faithful extension of C

- Keyword based language
  - Per opposition to pragma based languages (e.g. OpenMP)
- Main keywords (original Cilk):
  - **cilk** : declaration of a potentially parallel routine
  - **spawn**: launch of a potentially parallel routine
  - **sync**: wait for completion of launched routines
  - **inlet** : special function to aggregate results (reduction)
- A faithful extension of C
  - The C *elision* of a Cilk program is a valid C program

```
1
2
3       int fib (int n) {
4     if (n < 2) {
5       return n;
6     } else {
7       int x, y;
8       x =         fib (n-1);
9       y =         fib (n-2);
10
11      return x+y;
12    }
13  }
```

C Extension

Cilk is a faithful extension of C

- Keyword based language
  - Per opposition to pragma based languages (e.g. OpenMP)
- Main keywords (original Cilk):
  - **cilk** : declaration of a potentially parallel routine
  - **spawn**: launch of a potentially parallel routine
  - **sync**: wait for completion of launched routines
  - **inlet** : special function to aggregate results (reduction)
- A faithful extension of C
  - The C *elision* of a Cilk program is a valid C program

```
1  #include <cilk.h>
2
3  cilk int fib (int n) {
4    if (n < 2) {
5      return n;
6    } else {
7      int x, y;
8      x = spawn fib (n-1);
9      y = spawn fib (n-2);
10     sync;
11     return x+y;
12   }
13 }
```

Cilk Tasks

Notion of frame

- State of the current cilk function being executed
- Live local variables, function arguments
- "*Program Counter*" (PC)

Cilk Tasks

Notion of frame

- State of the current cilk function being executed
- Live local variables, function arguments
- "*Program Counter*" (PC)

A Frame is a Task

Cilk Task Lists

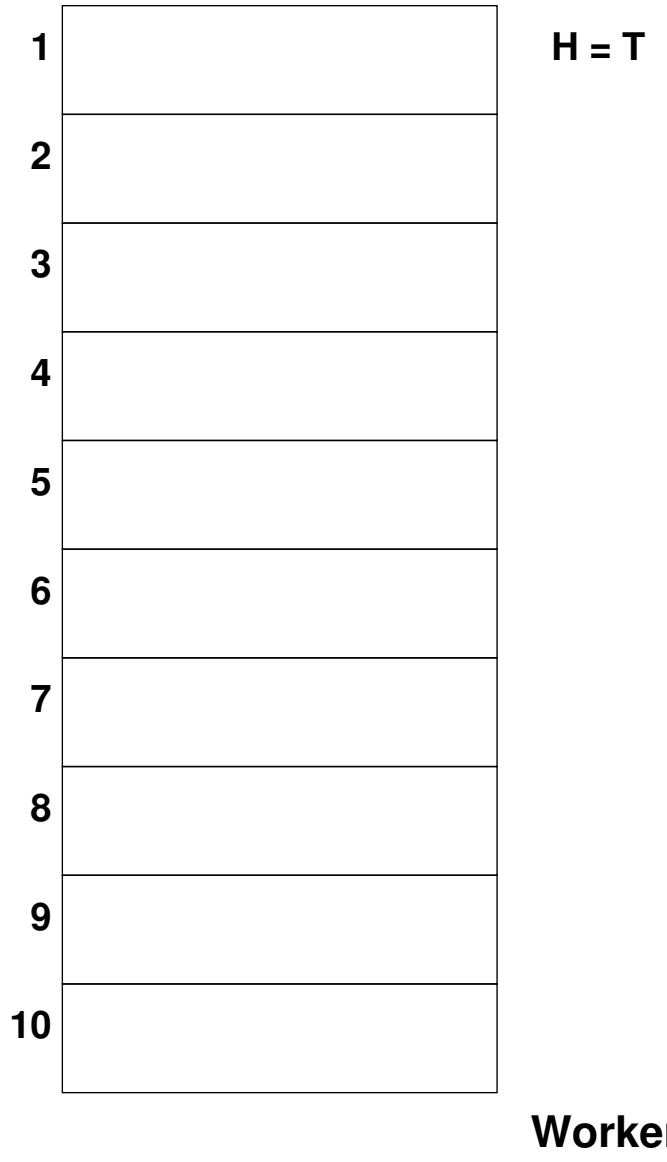
Notion of frame deque

- One task list per worker
- Implemented as a "*deque*" (doubly-ended queue of frames)
  - Head H
  - Tail T

Cilk Task Lists

Notion of frame deque

- One task list per worker
- Implemented as a "*deque*" (doubly-ended queue of frames)
  - Head H
  - Tail T

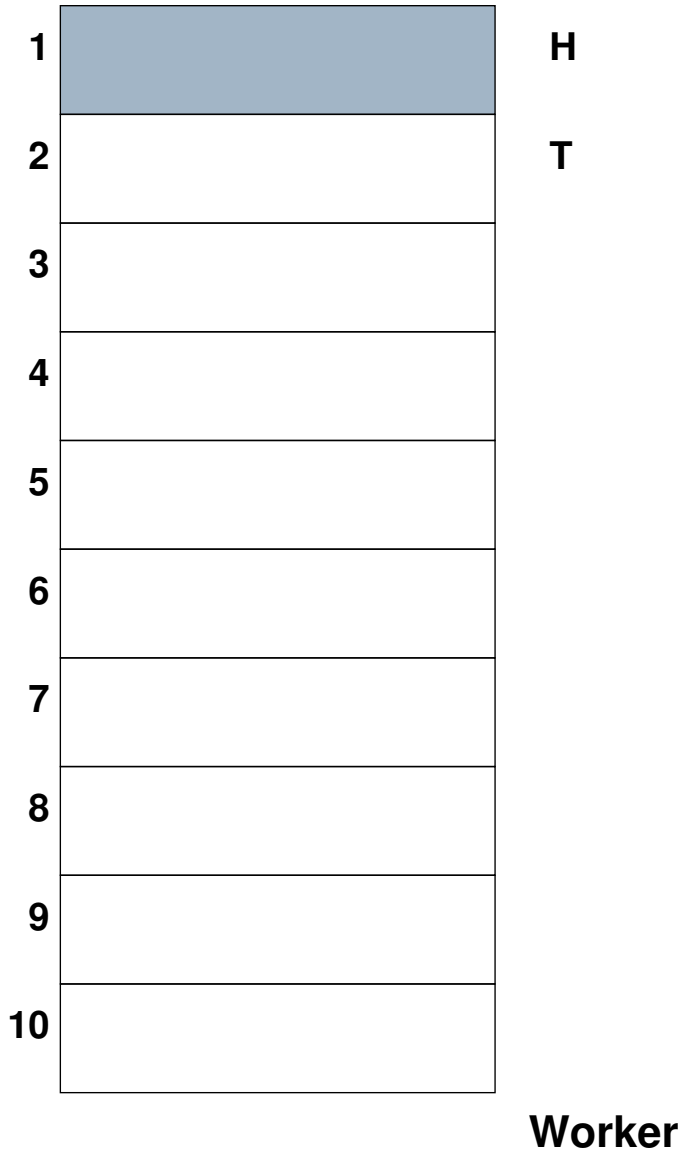


Cilk Task Lists

Notion of frame deque

- One task list per worker
- Implemented as a "*deque*" (doubly-ended queue of frames)
  - Head H
  - Tail T

A worker thread pushes/pops frames at the Tail side of its deque

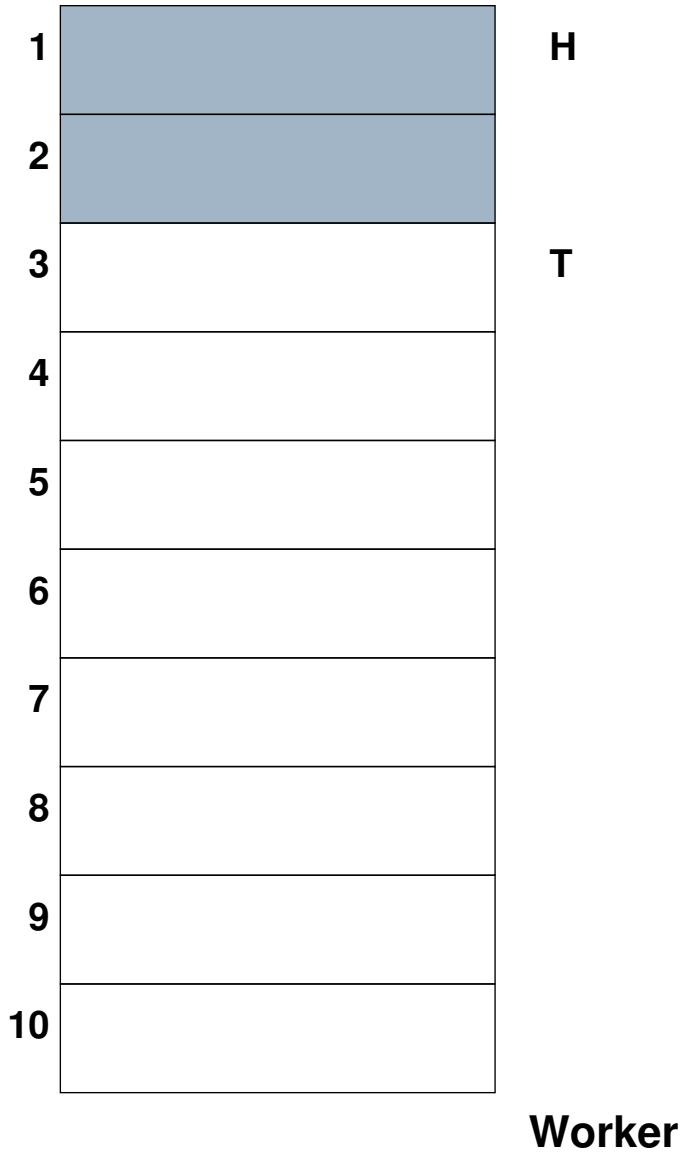


Cilk Task Lists

Notion of frame deque

- One task list per worker
- Implemented as a "*deque*" (doubly-ended queue of frames)
  - Head H
  - Tail T

A worker thread pushes/pops frames at the Tail side of its deque

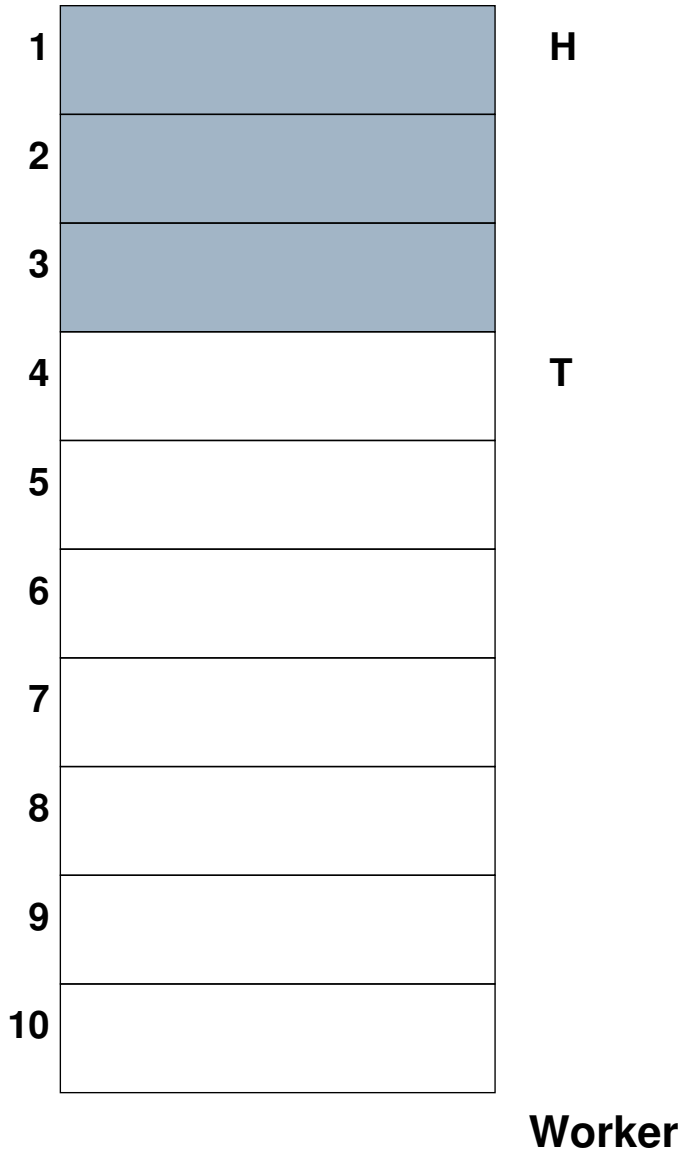


Cilk Task Lists

Notion of frame deque

- One task list per worker
- Implemented as a "*deque*" (doubly-ended queue of frames)
  - Head H
  - Tail T

A worker thread pushes/pops frames at the Tail side of its deque

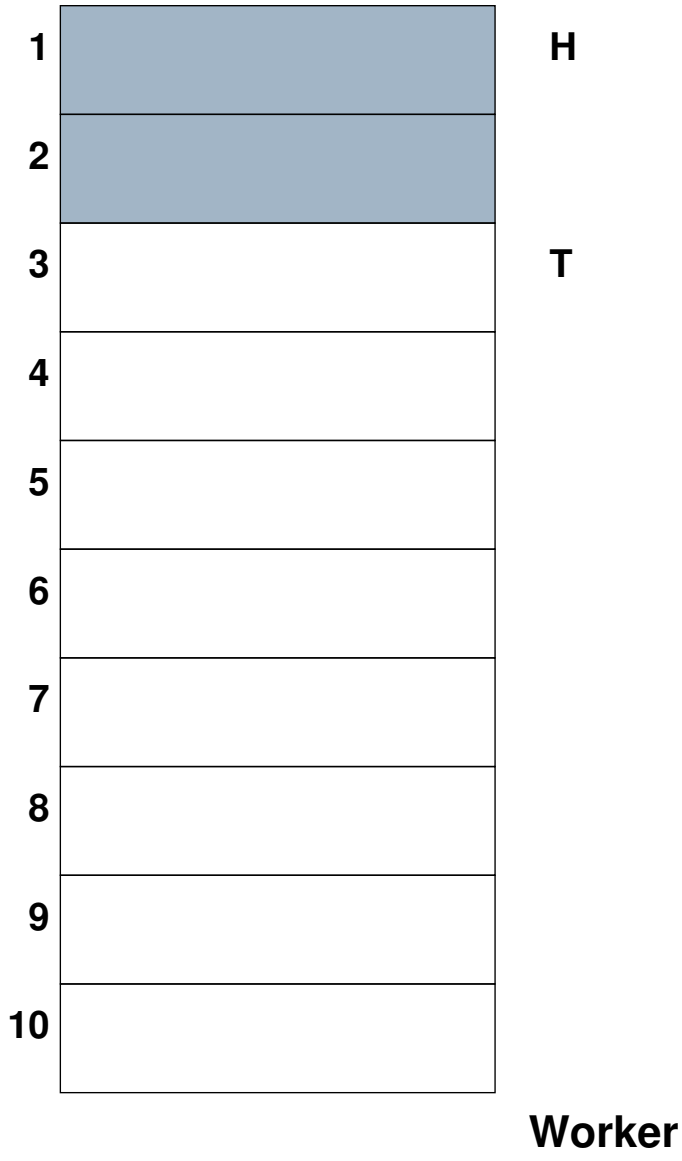


Cilk Task Lists

Notion of frame deque

- One task list per worker
- Implemented as a "*deque*" (doubly-ended queue of frames)
  - Head H
  - Tail T

A worker thread pushes/pops frames at the Tail side of its deque



Task Spawn

Upon a spawn, the worker...

- ... suspends the parent function
- ... saves the state of the parent function in its frame
- ... pushes the parent frame on its task list

...before calling the spawned child function

Task Spawn

Upon a spawn, the worker...

- ... suspends the parent function
- ... saves the state of the parent function in its frame
- ... pushes the parent frame on its task list

...before calling the spawned child function

When the child function completes and returns, the worker...

- ... pops the parent frame
- ... restores the state of the parent function from its frame
- ... resumes the parent function

Task Spawn

Upon a spawn, the worker...

- ... suspends the parent function
- ... saves the state of the parent function in its frame
- ... pushes the parent frame on its task list

...before calling the spawned child function

When the child function completes and returns, the worker...

- ... pops the parent frame
- ... restores the state of the parent function from its frame
- ... resumes the parent function

This is more or less what regular functions do...

Task Spawn

Upon a spawn, the worker...

- ... suspends the parent function
- ... saves the state of the parent function in its frame
- ... pushes the parent frame on its task list

...before calling the spawned child function

When the child function completes and returns, the worker...

- ... pops the parent frame
- ... restores the state of the parent function from its frame
- ... resumes the parent function

This is more or less what regular functions do...

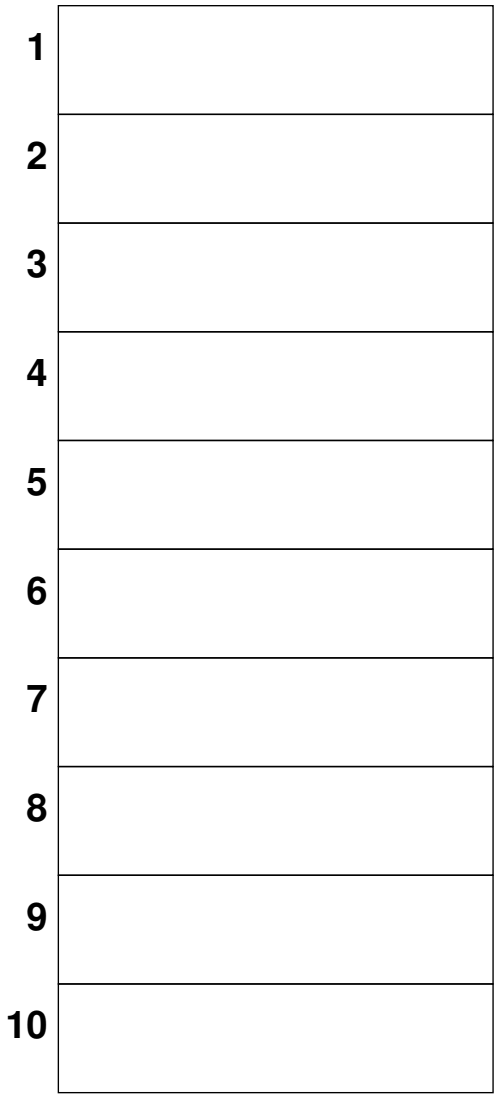
Where is the parallelism?

Example: Deque Management on Spawn

```
1  #include <cilk.h>
2
3  cilk int fib (int n) {
4    if (n < 2) {
5      return n;
6    } else {
7      int x, y;
8      x = spawn fib (n-1);
9      y = spawn fib (n-2);
10     sync;
11     return x+y;
12   }
13 }
```

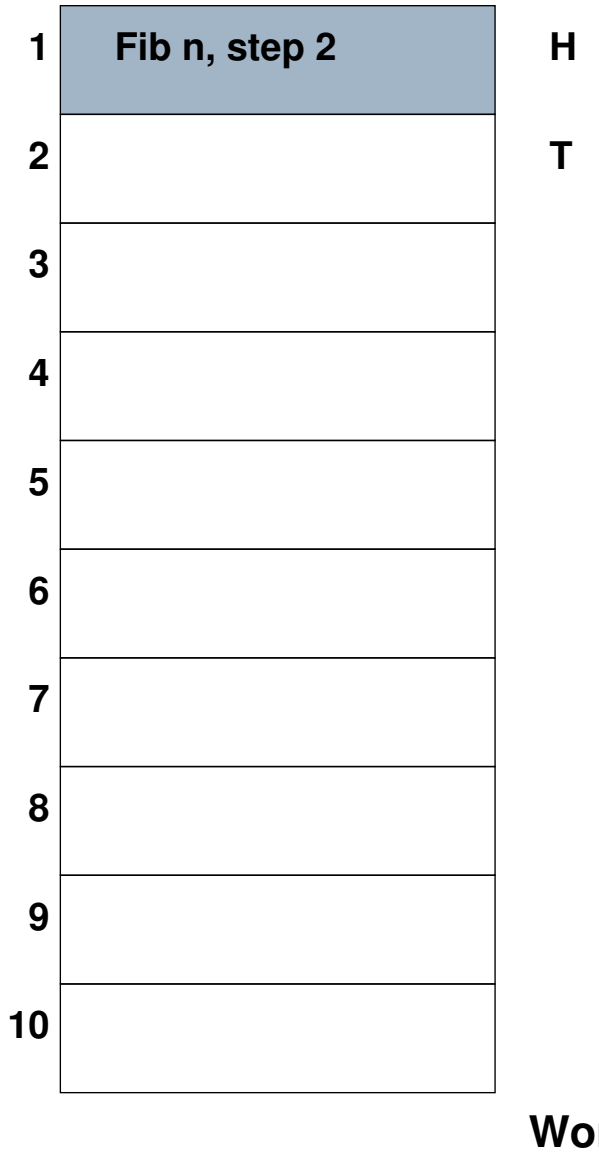
Example: Deque Management on Spawn

```
1  #include <cilk.h>
2
3  cilk int fib (int n) {
4    if (n < 2) {
5      return n;
6    } else {
7      int x, y;
8      x = spawn fib (n-1);
9      y = spawn fib (n-2);
10     sync;
11     return x+y;
12   }
13 }
```



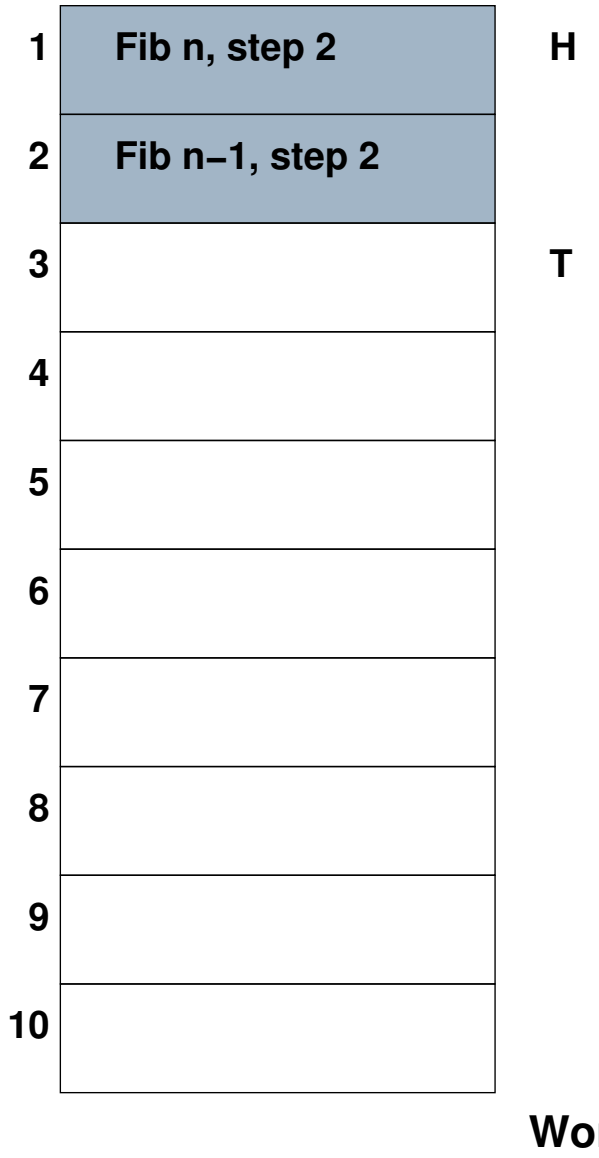
Example: Deque Management on Spawn

```
1  #include <cilk.h>
2
3  cilk int fib (int n) {
4    if (n < 2) {
5      return n;
6    } else {
7      int x, y;
8      x = spawn fib (n-1);
9      y = spawn fib (n-2);
10     sync;
11     return x+y;
12   }
13 }
```



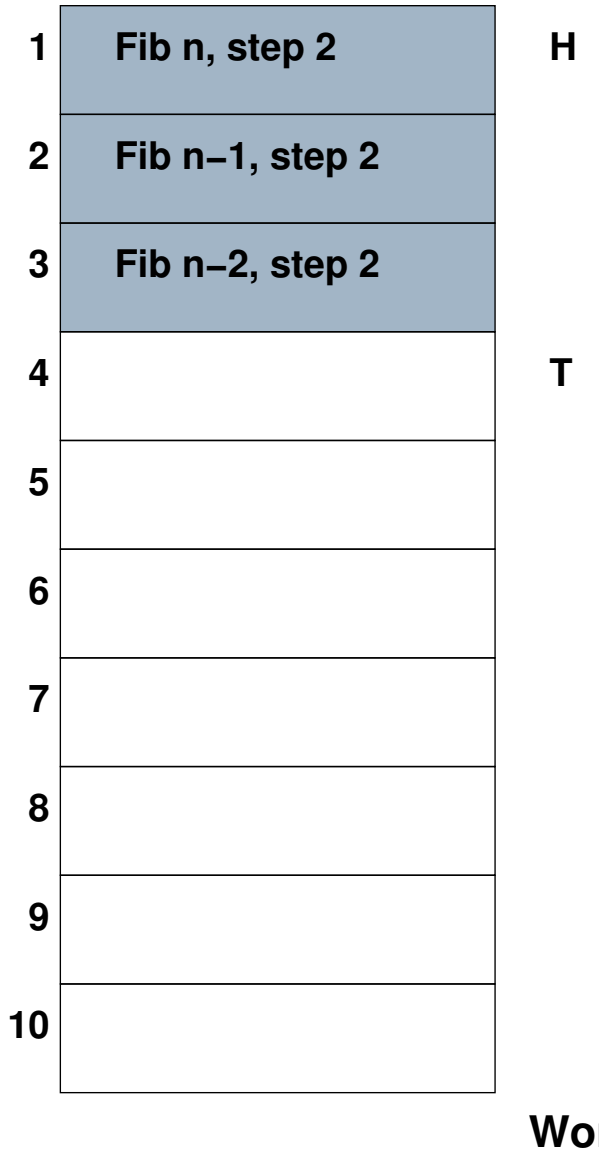
Example: Deque Management on Spawn

```
1  #include <cilk.h>
2
3  cilk int fib (int n) {
4    if (n < 2) {
5      return n;
6    } else {
7      int x, y;
8      x = spawn fib (n-1);
9      y = spawn fib (n-2);
10     sync;
11     return x+y;
12   }
13 }
```



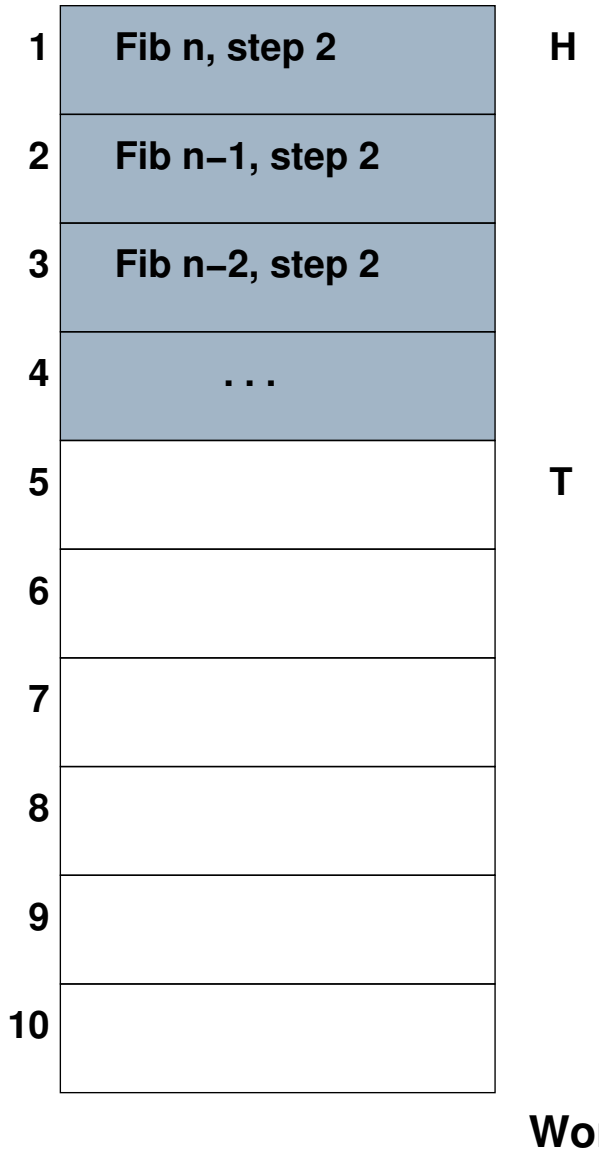
Example: Deque Management on Spawn

```
1  #include <cilk.h>
2
3  cilk int fib (int n) {
4    if (n < 2) {
5      return n;
6    } else {
7      int x, y;
8      x = spawn fib (n-1);
9      y = spawn fib (n-2);
10     sync;
11     return x+y;
12   }
13 }
```



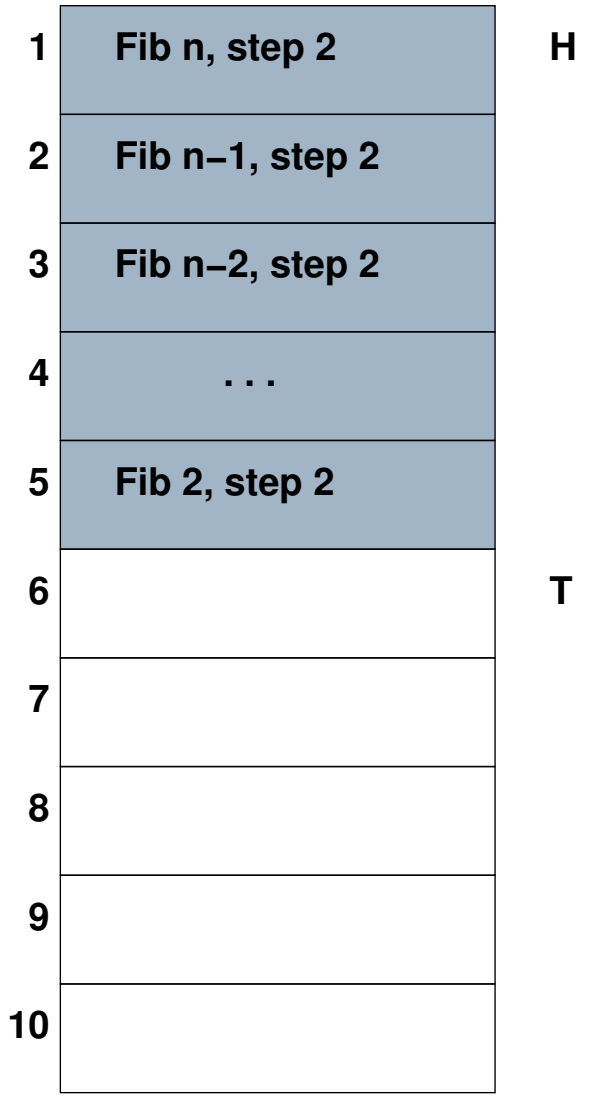
Example: Deque Management on Spawn

```
1  #include <cilk.h>
2
3  cilk int fib (int n) {
4    if (n < 2) {
5      return n;
6    } else {
7      int x, y;
8      x = spawn fib (n-1);
9      y = spawn fib (n-2);
10     sync;
11     return x+y;
12   }
13 }
```



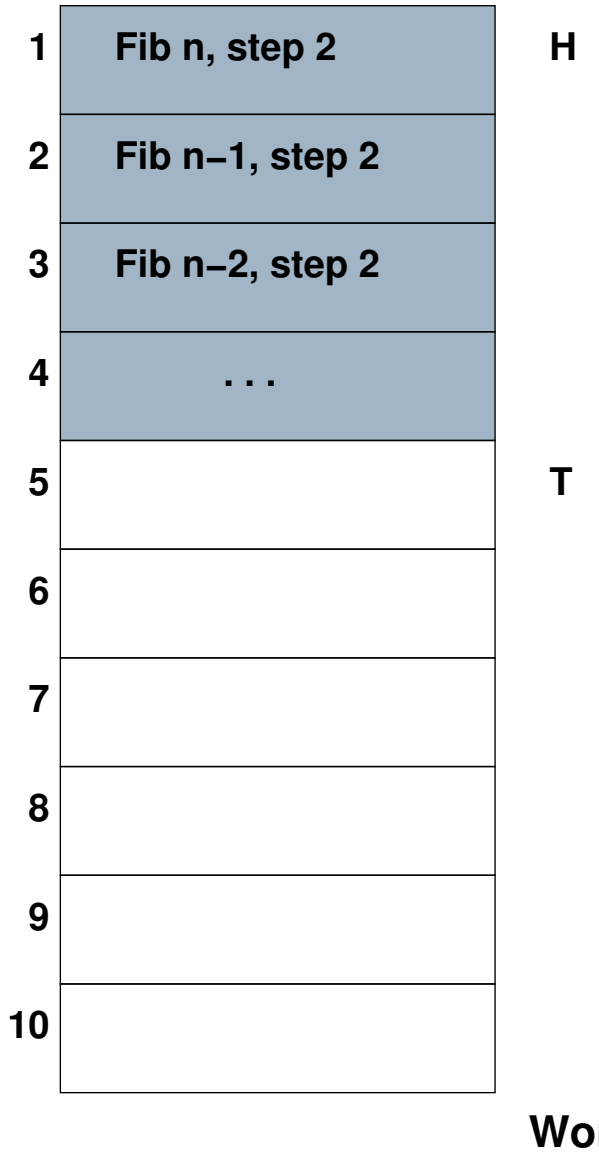
Example: Deque Management on Spawn

```
1  #include <cilk.h>
2
3  cilk int fib (int n) {
4    if (n < 2) {
5      return n;
6    } else {
7      int x, y;
8      x = spawn fib (n-1);
9      y = spawn fib (n-2);
10     sync;
11     return x+y;
12   }
13 }
```



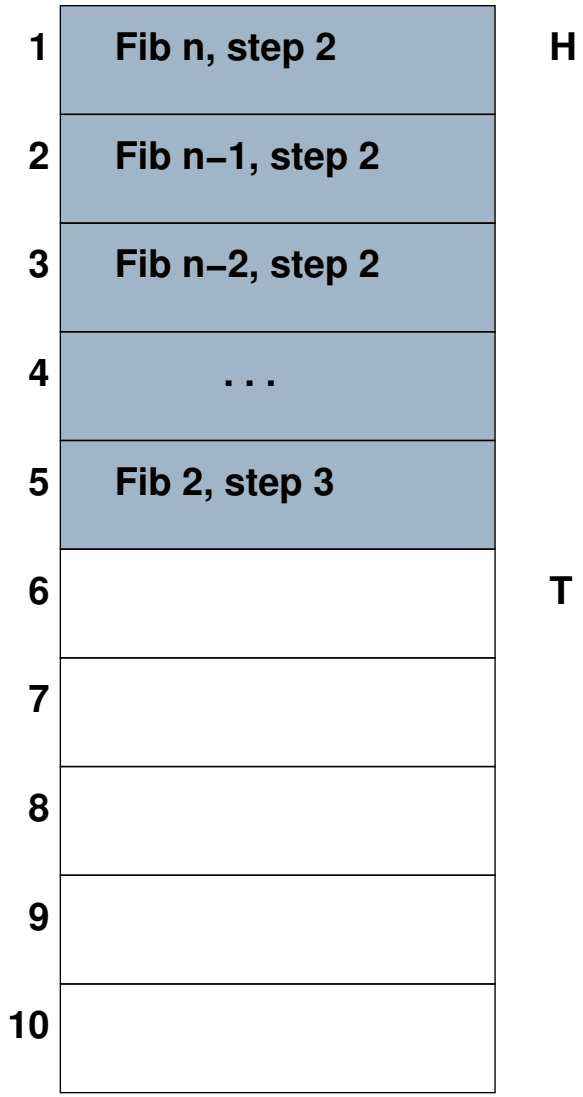
Example: Deque Management on Spawn

```
1  #include <cilk.h>
2
3  cilk int fib (int n) {
4    if (n < 2) {
5      return n;
6    } else {
7      int x, y;
8      x = spawn fib (n-1);
9      y = spawn fib (n-2);
10     sync;
11     return x+y;
12   }
13 }
```



Example: Deque Management on Spawn

```
1  #include <cilk.h>
2
3  cilk int fib (int n) {
4    if (n < 2) {
5      return n;
6    } else {
7      int x, y;
8      x = spawn fib (n-1);
9      y = spawn fib (n-2);
10     sync;
11     return x+y;
12   }
13 }
```



Parallelism

***Work Stealing* paradigm**

- **Idle** workers steal work...
- ... from other worker's queues

Parallelism

***Work Stealing* paradigm**

- **Idle** workers steal work...
- ... from other worker's queues

Work stolen as frame/task

- A thief resumes a suspended parent task...
- ... **while** a victim runs its child

Parallelism

***Work Stealing* paradigm**

- **Idle** workers steal work...
- ... from other worker's queues

Work stolen as frame/task

- A thief resumes a suspended parent task...
- ... **while** a victim runs its child

Load balancing: Idle workers steal from busy workers

Task Spawn [UPDATED]

Upon a spawn, the worker...

- ... suspends the parent function
- ... saves the state of the parent function in its frame
- ... pushes the parent frame on its task list

...before calling the spawned child function

Task Spawn [UPDATED]

Upon a spawn, the worker...

- ... suspends the parent function
- ... saves the state of the parent function in its frame
- ... pushes the parent frame on its task list

...before calling the spawned child function

When the child function completes and returns, the worker...

- **... attempts to pop the parent frame**
- **if it succeeds**, it...
  - ... restores the state of the parent function from its frame
  - ... resumes the parent function

Work Stealing Implementation

Task lists implementation...

- Doubly-ended queue
- Head H
- Tail T

Work Stealing Implementation

Task lists implementation...

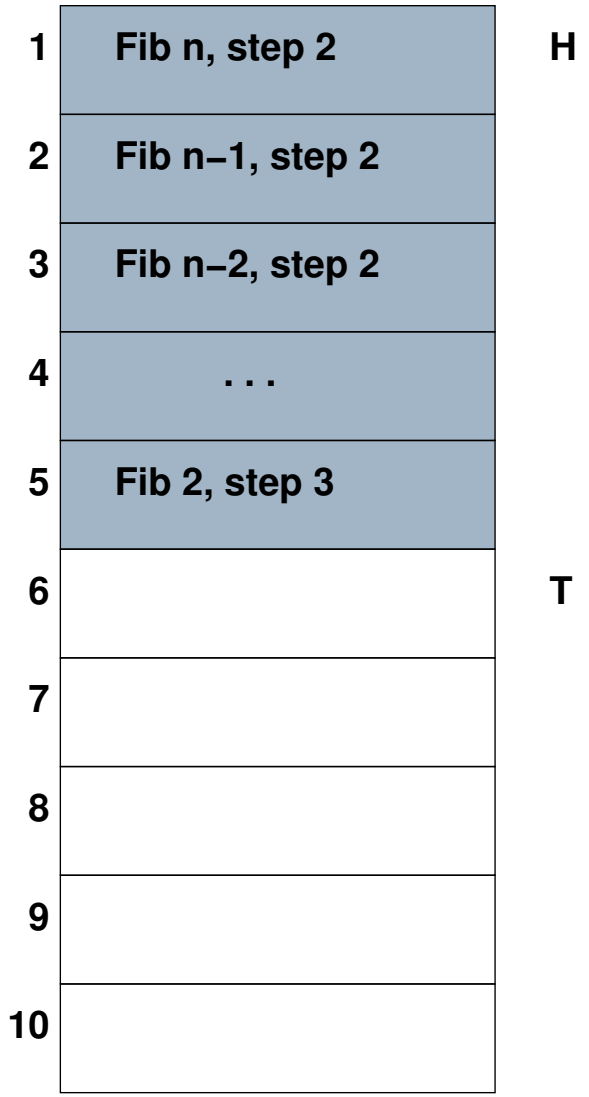
- Doubly-ended queue
- Head H
- Tail T

... with the following rules

- Workers push/pop work at the Tail side T of their own deque
- An idle worker (*thief*) steals work at the Head side H of another worker (*victim*) deque
- T >= H under normal conditions

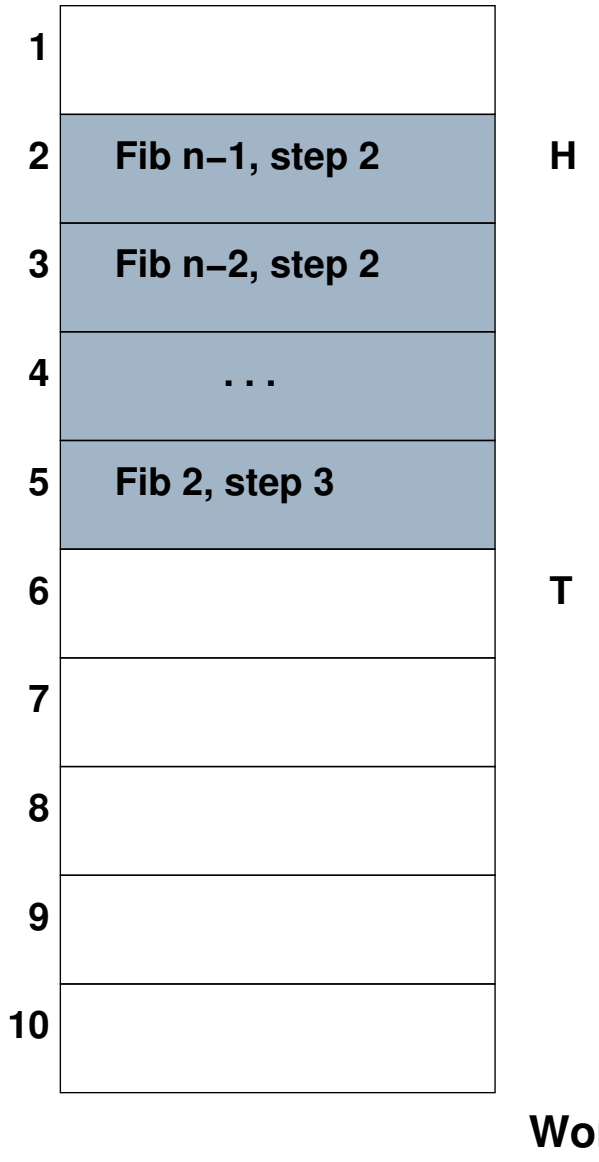
Example: Work Stealing

```
1  #include <cilk.h>
2
3  cilk int fib (int n) {
4    if (n < 2) {
5      return n;
6    } else {
7      int x, y;
8      x = spawn fib (n-1);
9      y = spawn fib (n-2);
10     sync;
11     return x+y;
12   }
13 }
```



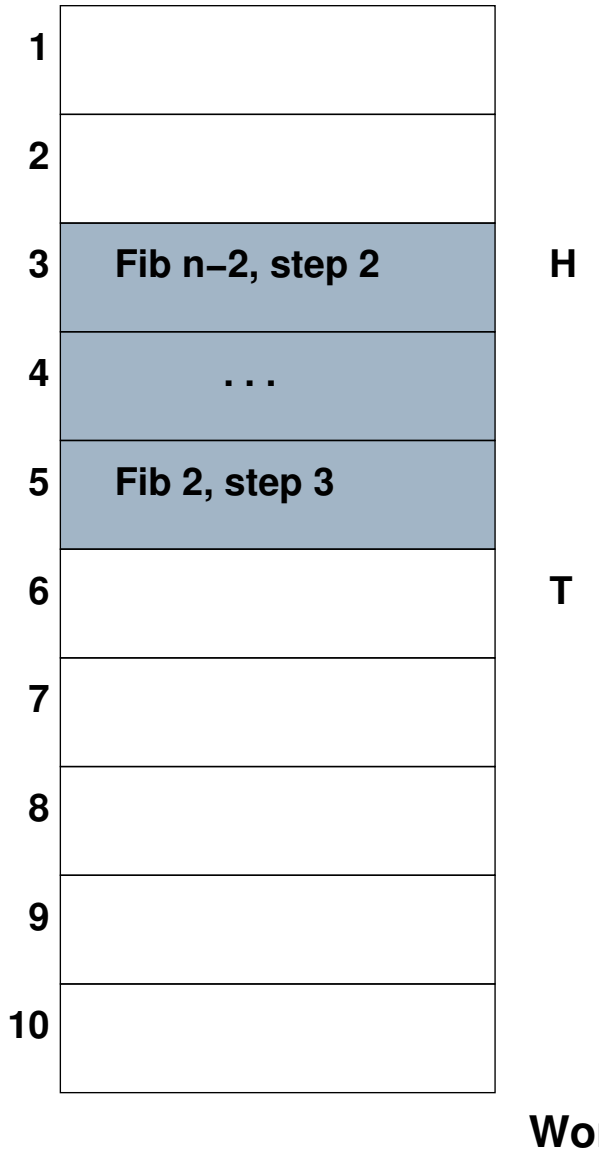
Example: Work Stealing

```
1  #include <cilk.h>
2
3  cilk int fib (int n) {
4    if (n < 2) {
5      return n;
6    } else {
7      int x, y;
8      x = spawn fib (n-1);
9      y = spawn fib (n-2);
10     sync;
11     return x+y;
12   }
13 }
```



Example: Work Stealing

```
1  #include <cilk.h>
2
3  cilk int fib (int n) {
4    if (n < 2) {
5      return n;
6    } else {
7      int x, y;
8      x = spawn fib (n-1);
9      y = spawn fib (n-2);
10     sync;
11     return x+y;
12   }
13 }
```



Cilk's Keywords Summary

- **cilk** : declaration of a potentially parallel routine
- **spawn**: launch of a potentially parallel routine
- **sync**: wait for completion of launched routines

Cilk's Keywords Summary

- **cilk** : declaration of a potentially parallel routine
- **spawn**: launch of a potentially parallel routine
- **sync**: wait for completion of launched routines
- **inlet** : special function to aggregate results (reduction)

Intel Cilk Plus

URL: http://www.cilkplus.org/

Changes

- Supports C and C++
- No need to declare Cilk functions

Intel Cilk Plus

URL: http://www.cilkplus.org/

Changes

- Supports C and C++
- No need to declare Cilk functions

Main keywords

- **cilk_spawn**: similar to original Cilk's **spawn**
- **cilk_sync**: similar to original Cilk's **sync**
- **cilk ::reducer** <...>
  - Template parameterized with a reduction op
  - Replacement for inlets
- **cilk_for** : parallel loop
- Fortran inspired *Array Notation*

Cilk Plus Parallel Loop

Work-Stealing Loop

- **cilk_for** keyword
- *Potentially* parallel loop
- Recursively divided range
- Work-stealing load balancing

```
1  int i;
2
3  for (i=0; i<N; i++) {
4    f(i);
5  }
6
7  /* ----- */
8
9  for (i=0; i<N; i++) {
10   cilk_spawn f(i);
11 }
12 cilk_sync;
13
14 /* ----- */
15
16 cilk_for (i=0; i<N; i++) {
17   f(i);
18 }
```

Other Cilk Plus Ports

Cilk Plus / GCC

- Integrated in GCC 4.9.2+
  - **Tasks**
  - **Array notation**
  - No cilk_for keyword yet
- Usage

```
1    g++ -fcilkplus -lcilkrts -o fib fib.cpp
```

Cilk Plus / LLVM

- URL: http://cilkplus.github.io/

Programming Modern Platforms using Tasks

Expressing tasks?

- Divide and conquer: Cilk (recursive tasks)
- **OpenMP 3.x and 4.x**
- Dependencies compiler: PaRSEC (parameterized task graph)
- Sequential task flow: StarPU (directed acyclic task graph)

1.5

Tasks for the Masses: OpenMP

OpenMP

Parallel programming with threads and tasks

- Consortium: OpenMP Architecture Review Board (ARB)
- C/C++ and Fortran annotations

History

- OpenMP 1.x (1997-98), OpenMP 2.x (2000-02)
  - Thread-based fork-join programming model design
- OpenMP 3.x (2008-11)
  - Independent tasks
- OpenMP 4.x (2013-15)
  - Task with dependencies
  - Accelerators / devices
- (OpenMP 5.x)
  - On-going work
  - Support for instrumenting tools (OMPT)

OpenMP Fork-Join Model

Thread-based parallel regions

```
1 {
2 #pragma omp parallel
3    {
4       printf("thread␣number␣%d␣of␣%d\n", omp_get_thread_num(),
              , omp_get_num_threads());
5    }
6 }
```

- Team of threads launched during parallel region
- Synchronizations using barriers, critical regions or locks

OpenMP Fork-Join Model

Thread-based parallel loops:

```
1 {
2   int i;
3
4 #pragma omp parallel for
5   for (i=0; i<n; i++) {
6     b[i] = (a[i] + a[i-1])/2.0;
7   }
8 }
```

- Team of threads launched during parallel region
- Parallel loop mapped on multiple threads
- Notion of *worksharing*

OpenMP 3.x Independent Tasks Support

Initial task support in OpenMP

- Inspired by Cilk
- Integrates tasks in the Fork-Join model
- Notion of implicit tasks
  - Each thread in a parallel region executes one implicit task
- Explicit tasks can be created by the pragma **omp task**
- Notion of *scheduling point*
  - *Pause / resume point for tasks*
  - *Recursive tasks*
  - *Task synchronization using pragma +omp taskwait, critical regions, locks*
  - *Barriers wait for all tasks created in the parallel region*

OpenMP 3.x Example: Independent Tasks

```
1  int item[N];
2
3  void g(int);
4
5  void f()
6  {
7  #pragma omp parallel
8      {
9  #pragma omp single
10         {
11             int i;
12             for (i=0; i<N; i++)
13 #pragma omp task untied
14                 g(item[i]);
15         }
16     }
17 }
```

OpenMP 3.x Example: Independent Tasks

```
1  void process_list_items(struct list_item * list)
2  {
3    #pragma omp parallel
4      {
5        #pragma omp single
6          {
7            struct list_item * p = list;
8            while (p)
9              {
10               #pragma omp task
11                 { /* p is firstprivate */
12                   process_item(p);
13                 }
14               p = p->next;
15             }
16         }
17     }
18 }
```

OpenMP 4.x Task Support

Extend the task model with data dependencies

- Inspired by BSC's OmpSs, Intel's task queues
- New keywords
  - in input data dependence
  - out output data dependence
  - inout input/output data dependence
- Data dependencies
  - Lock-less synchronization
  - Fine-grained synchronization

OpenMP 4.x Example: Task Dependencies

```
1  void f()
2  {
3      int a;
4
5  #pragma omp parallel
6  #pragma omp single
7      {
8  #pragma omp task shared(a) depend(out: a)
9          foo(&a);
10
11 #pragma omp task shared(a) depend(inout: a)
12         bar(&a);
13
14 #pragma omp task shared(a) depend(in: a)
15         bar(&a);
16     }
17 }
```

Programming Modern Platforms using Tasks

Expressing tasks?

- Divide and conquer: Cilk (recursive tasks)
- OpenMP 3.x and 4.x
- **Dependencies compiler: PaRSEC (parameterized task graph)**
- Sequential task flow: StarPU (directed acyclic task graph)

1.6

Parametric Task Graph model: PaRSEC

PaRSEC: Introduction and Principles

PaRSEC

- Developed at ICL Lab (UTK) and Univ. of Manchester
- Parallel + distributed platforms
- Compact representation of a graph of tasks

Parameterized task graph (PTG)

- Tasks and dependencies expressed in a specific language: JDF
- JDF source processed by a compiler
- Decentralized distributed execution
- Work-stealing load-balancing at node level

PaRSEC: basic JDF example

```
1  NB    [ type="int" ]
2
3  Task(k)
4  k = 0 .. NB
5  : taskdist( k )
6
7  RW  A <- (k == 0) ? NEW : A Task( k-1 )
8        -> (k < NB) ? A Task( k+1 )
9
10 BODY
11 {
12      int *Aint = (int*)A;
13
14      if ( k == 0 ) {
15          *Aint = 0;
16      } else {
17          *Aint += 1;
18      }
19      printf("A = %d\n", *Aint);
20 }
21 END
```

PaRSEC: more complex JDF example

```
1  potrf_zpotrf(k) [high_priority = on]
2
3  k = 0 .. descA->mt-1
4
5  :descA(k, k)
6
7  RW T <- (k == 0) ? descA(k, k) : T potrf_zherk(k-1, k)
8       -> T potrf_ztrsm(k+1..descA->mt-1, k)
9       -> descA(k, k)
10
11 ; (k >= (descA->mt - PRI_CHANGE)) ? (descA->mt - k) * (descA
       ->mt - k) * (descA->mt - k) : PRI_MAX
12
13 BODY [type=RECURSIVE]
14 { [...] }
15 END
16
17 BODY
18 { [...] }
19 END
```

PaRSEC: more complex JDF example (cont'd)

```
1  potrf_ztrsm(m, k) [high_priority = on]
2
3  // Execution space
4  m = 1 .. descA->mt-1
5  k = 0 .. m-1
6
7  // Parallel partitioning
8  : descA(m, k)
9
10 // Parameters
11 READ  T <- T potrf_zpotrf(k)
12 RW    C <- (k == 0) ? descA(m, k) : C potrf_zgemm(m, k, k-1)
13           -> A potrf_zherk(k, m)
14           -> A potrf_zgemm(m, k+1..m-1, k)
15           -> B potrf_zgemm(m+1..descA->mt-1, m, k)
16           -> descA(m, k)
17
18 ; (m >= (descA->mt - PRI_CHANGE)) ? (descA->mt - m) * (descA
       ->mt - m) * (descA->mt - m) + 3 * ((2 * descA->mt) - k -
       m - 1) * (m - k) : PRI_MAX
19
20 BODY [type=RECURSIVE]
```

Programming Modern Platforms using Tasks

Expressing tasks?

- Divide and conquer: Cilk (recursive tasks)
- OpenMP 3.x and 4.x
- Dependencies compiler: PaRSEC (parameterized task graph)

- **Sequential task flow**: StarPU (directed acyclic task graph)
  - **See second part**: Programming Modern Platforms with the StarPU Task-Based Runtime System

2

StarPU
A Unified Runtime for Heterogeneous Platforms

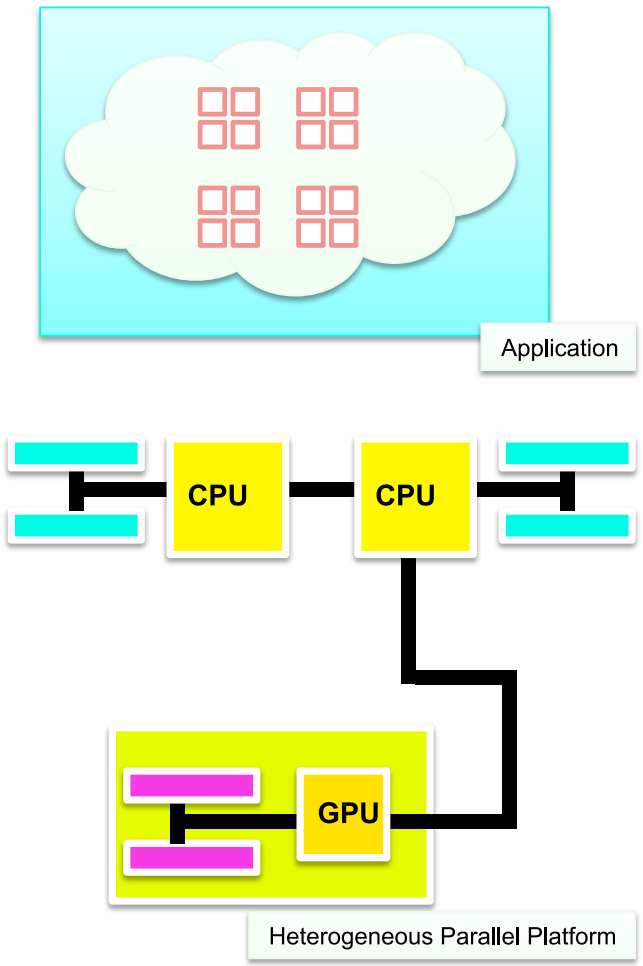
Heterogeneous Parallel Platforms

Heterogeneous Association

- General purpose processor
- Specialized accelerator

Generalization

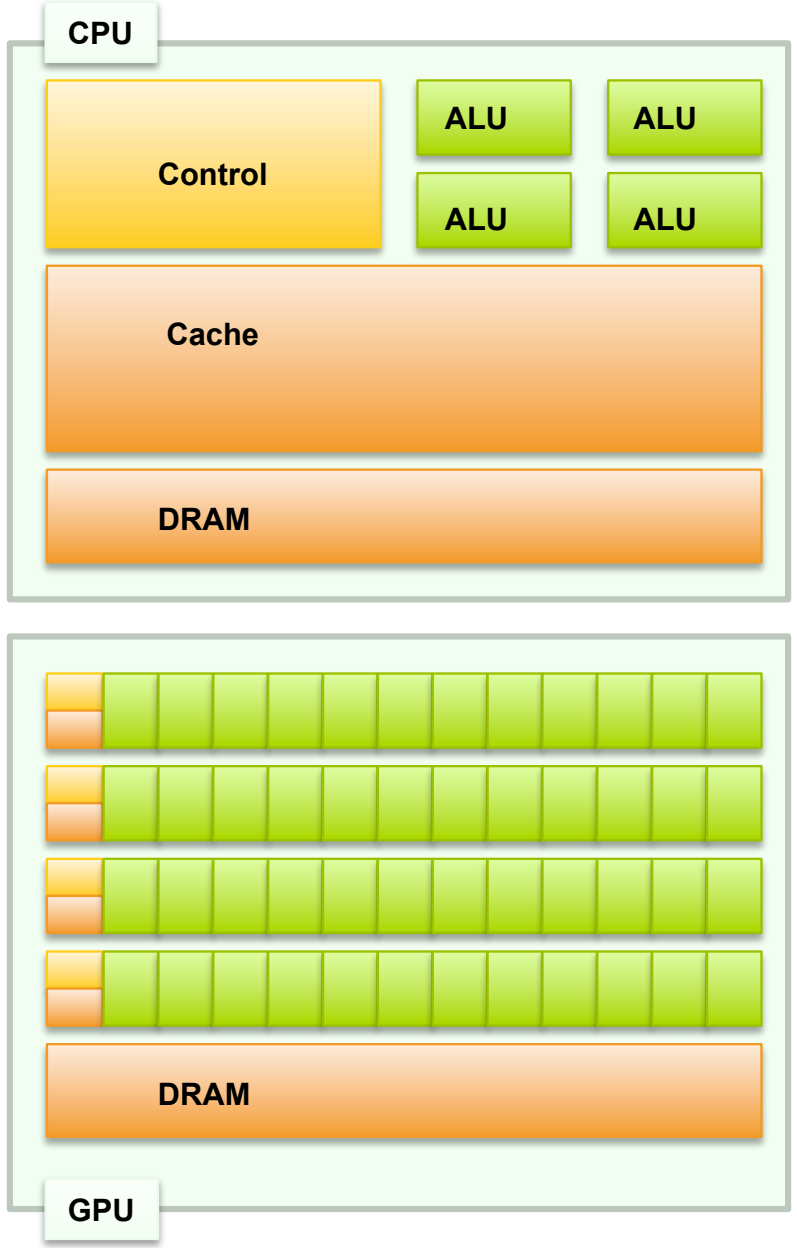
- Distributed cores, discrete accelerators
  - Standalone GPUs
  - Intel Xeon Phi (KNC)
- Integrated cores
  - Intel Skylake / Kaby Lake
  - Intel Xeon Phi (KNL)
  - AMD Fusion
  - nVidia Tegra, ARM big.LITTLE
- Combination of various units
  - Latency-optimized cores
  - Throughput-optimized cores
  - Energy-optimized cores



Example: CPU vs GPU Hardware

Multiple strategies for multiple purposes

- **CPU**
  - Strategy
    - Large caches
    - Large control
  - Purpose
    - Complex codes, branching
    - Complex memory access patterns
  - World Rally Championship car
- **GPU**
  - Strategy
    - Lot of computing power
    - Simplified control
  - Purpose
    - Regular data parallel codes
    - Simple memory access patterns
  - Formula One car



Accelerators

Special purpose computing devices
(or general purpose GPUs)

- (initially) a discrete expansion card
- Rationale: dye area trade-off

Accelerators

Special purpose computing devices
(or general purpose GPUs)

- (initially) a discrete expansion card
- Rationale: dye area trade-off

Single Instruction Multiple Threads (SIMT)

- A single control unit...
- ... for several computing units

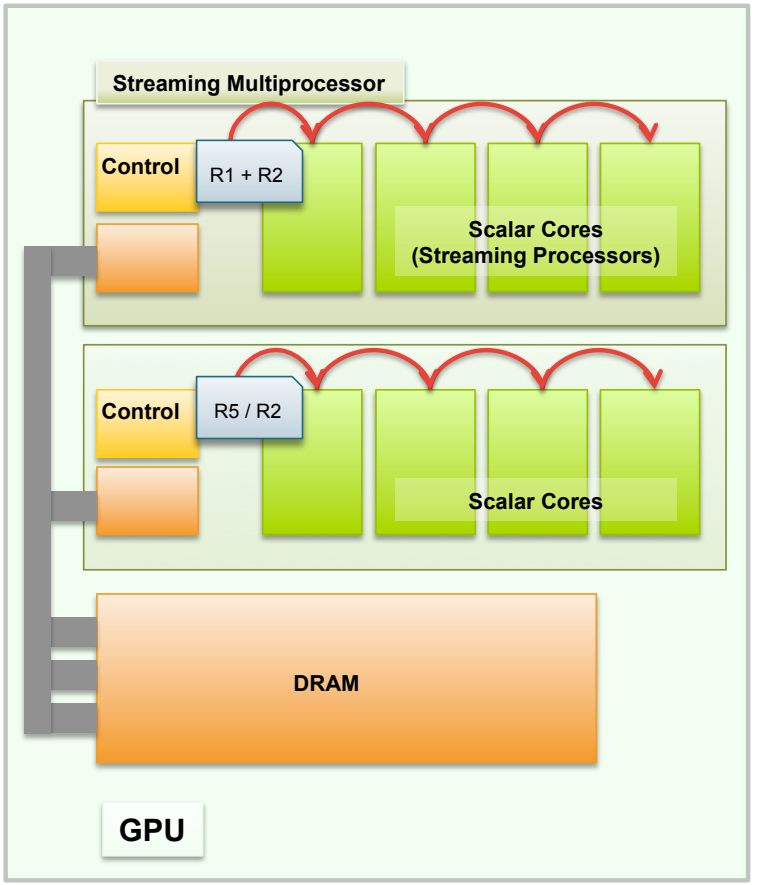
Accelerators

Special purpose computing devices
(or general purpose GPUs)

- (initially) a discrete expansion card
- Rationale: dye area trade-off

Single Instruction Multiple Threads (SIMT)

- A single control unit. . .
- . . . for several computing units



Accelerators

Special purpose computing devices
(or general purpose GPUs)

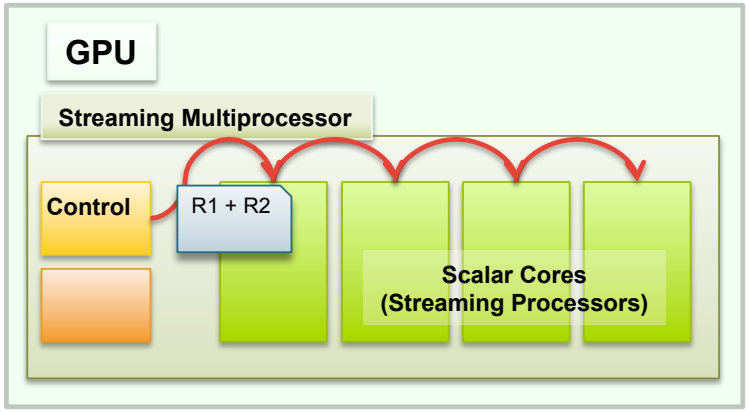
- (initially) a discrete expansion card
- Rationale: dye area trade-off

Single Instruction Multiple Threads (SIMT)

- A single control unit. . .
- . . . for several computing units

SIMT is distinct from SIMD

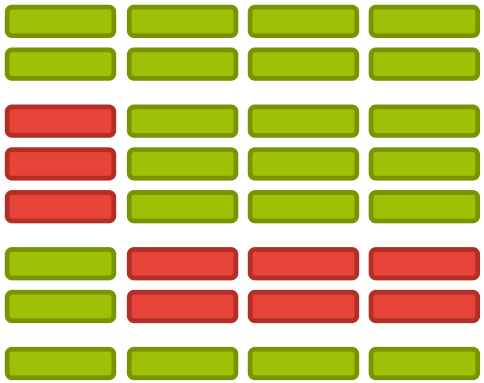
- Allows flows to diverge
- . . . but better avoid it!




```
...
if(cond){

  ...
  ...
  ...

} else {
  ...
  ...
}
...
```



StarPU Programming Model: Sequential Task Flow

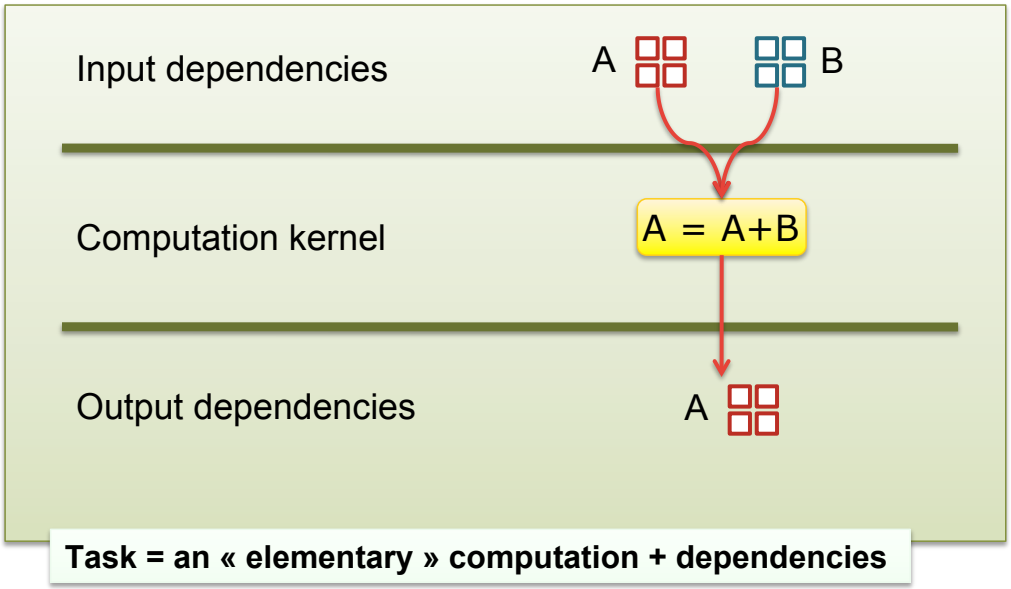
- Express parallelism...
- ... using the natural program flow

- **Submit** tasks in the sequential flow of the program...
- ... then let the runtime schedule the tasks asynchronously

Task Model

StarPU Tasks

- Elementary computation
  - Some kernel
- → Potential parallel work
- Constraints
  - Input needed
  - Output produced
  - → Dependencies
- → Degrees of Freedom in realizing the potential parallelism
- Specificities
  - Atomic tasks (non-interruptible)
  - Flat model (non-recursive)



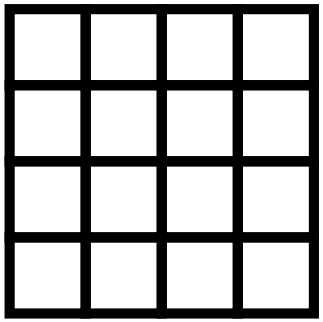
2.1

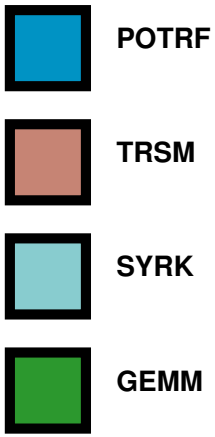
Let's *Taskify* some Linear Algebra Algorithm

Sequential Task Flow Graph Building

Example: Cholesky Decomposition

```
for (j = 0; j < N; j++) {
  POTRF (   A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (   A[i][j],   A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (   A[i][i],   A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (   A[i][k],
               A[i][j],   A[k][j]);
  }
}
```

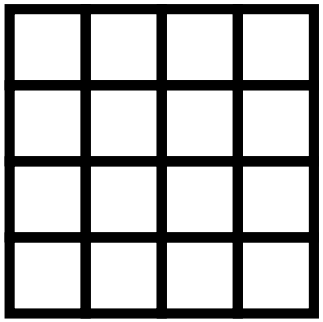


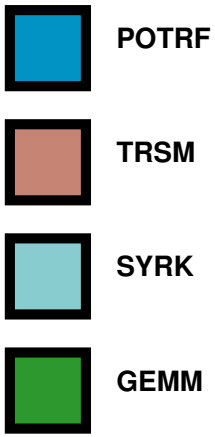


Sequential Task Flow Graph Building

Example: Cholesky Decomposition

```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
```

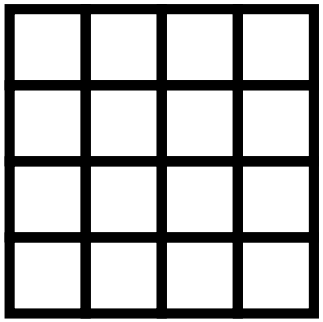


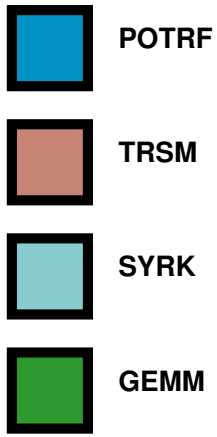


Sequential Task Flow Graph Building

Example: Cholesky Decomposition

```
for (j = 0; j < N; j++) {
  task_insert( POTRF (RW,A[j][j]) );
  for (i = j+1; i < N; i++)
    task_insert( TRSM (RW,A[i][j], R,A[j][j]) );
  for (i = j+1; i < N; i++) {
    task_insert( SYRK (RW,A[i][i], R,A[i][j]) );
    for (k = j+1; k < i; k++)
      task_insert( GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]) );
  }
}
wait_for_all();
```

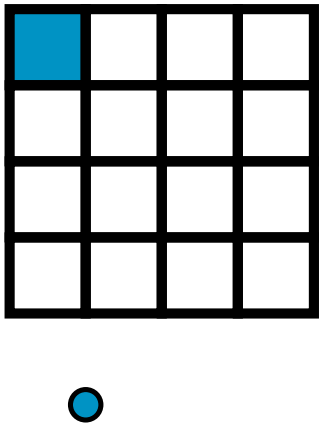




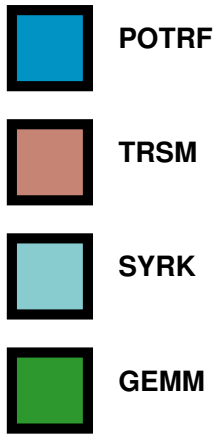
Sequential Task Flow Graph Building

Example: Cholesky Decomposition

```
for (j = 0; j < N; j++) {
  task_insert( POTRF (RW,A[j][j]) );
  for (i = j+1; i < N; i++)
    task_insert( TRSM (RW,A[i][j], R,A[j][j]) );
  for (i = j+1; i < N; i++) {
    task_insert( SYRK (RW,A[i][i], R,A[i][j]) );
    for (k = j+1; k < i; k++)
      task_insert( GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]) );
  }
}
wait_for_all();
```



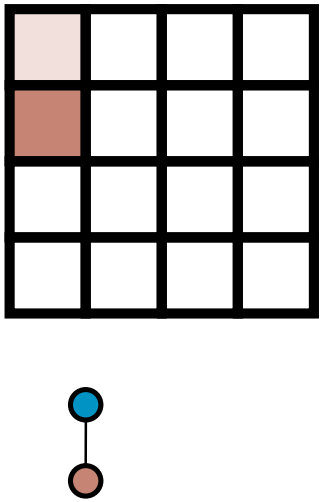
- Tasks are submitted asynchronously



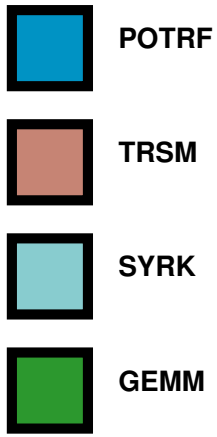
Sequential Task Flow Graph Building

Example: Cholesky Decomposition

```
for (j = 0; j < N; j++) {
  task_insert( POTRF (RW,A[j][j]) );
  for (i = j+1; i < N; i++)
    task_insert( TRSM (RW,A[i][j], R,A[j][j]) );
  for (i = j+1; i < N; i++) {
    task_insert( SYRK (RW,A[i][i], R,A[i][j]) );
    for (k = j+1; k < i; k++)
      task_insert( GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]) );
  }
}
wait_for_all();
```



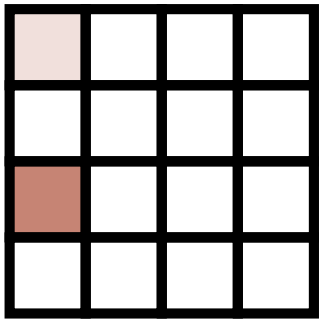
- Tasks are submitted asynchronously
- StarPU infers data dependences...

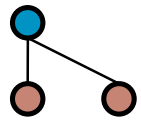


Sequential Task Flow Graph Building

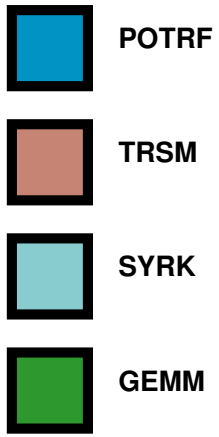
Example: Cholesky Decomposition

```
for (j = 0; j < N; j++) {
  task_insert( POTRF (RW,A[j][j]) );
  for (i = j+1; i < N; i++)
    task_insert( TRSM (RW,A[i][j], R,A[j][j]) );
  for (i = j+1; i < N; i++) {
    task_insert( SYRK (RW,A[i][i], R,A[i][j]) );
    for (k = j+1; k < i; k++)
      task_insert( GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]) );
  }
}
wait_for_all();
```





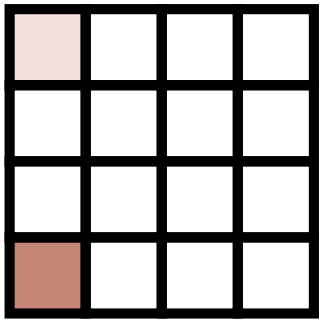
- Tasks are submitted asynchronously
- StarPU infers data dependences...
- ... and build a graph of tasks



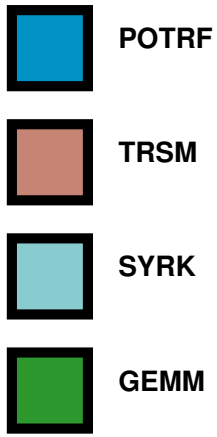
Sequential Task Flow Graph Building

Example: Cholesky Decomposition

```
for (j = 0; j < N; j++) {
  task_insert( POTRF (RW,A[j][j]) );
  for (i = j+1; i < N; i++)
    task_insert( TRSM (RW,A[i][j], R,A[j][j]) );
  for (i = j+1; i < N; i++) {
    task_insert( SYRK (RW,A[i][i], R,A[i][j]) );
    for (k = j+1; k < i; k++)
      task_insert( GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]) );
  }
}
wait_for_all();
```



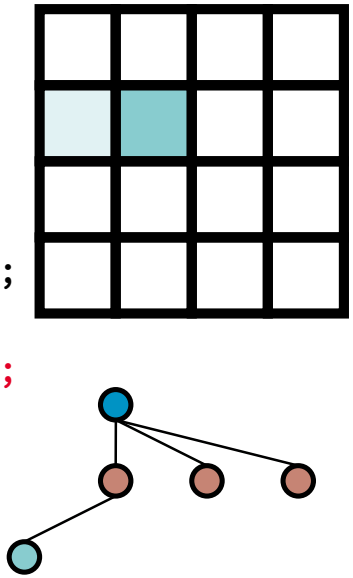
- Tasks are submitted asynchronously
- StarPU infers data dependences...
- ... and build a graph of tasks



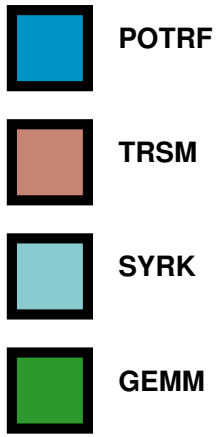
Sequential Task Flow Graph Building

Example: Cholesky Decomposition

```
for (j = 0; j < N; j++) {
  task_insert( POTRF (RW,A[j][j]) );
  for (i = j+1; i < N; i++)
    task_insert( TRSM (RW,A[i][j], R,A[j][j]) );
  for (i = j+1; i < N; i++) {
    task_insert( SYRK (RW,A[i][i], R,A[i][j]) );
    for (k = j+1; k < i; k++)
      task_insert( GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]) );
  }
}
wait_for_all();
```



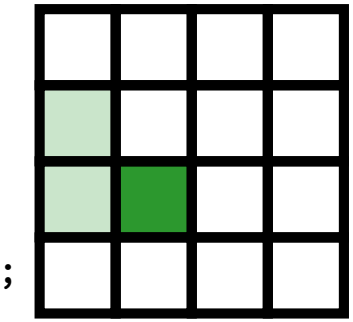
- Tasks are submitted asynchronously
- StarPU infers data dependences...
- ... and build a graph of tasks

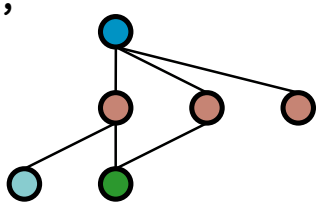


Sequential Task Flow Graph Building

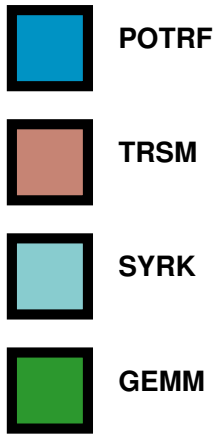
Example: Cholesky Decomposition

```
for (j = 0; j < N; j++) {
  task_insert( POTRF (RW,A[j][j]) );
  for (i = j+1; i < N; i++)
    task_insert( TRSM (RW,A[i][j], R,A[j][j]) );
  for (i = j+1; i < N; i++) {
    task_insert( SYRK (RW,A[i][i], R,A[i][j]) );
    for (k = j+1; k < i; k++)
      task_insert( GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]) );
  }
}
wait_for_all();
```





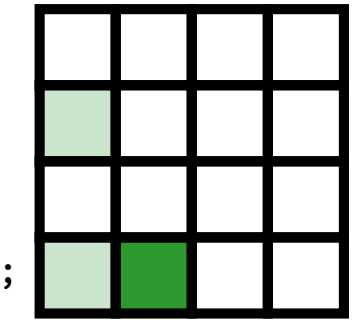
- Tasks are submitted asynchronously
- StarPU infers data dependences...
- ... and build a graph of tasks

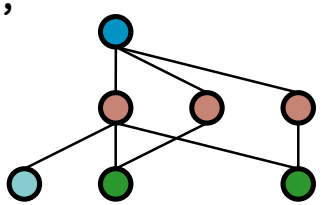


Sequential Task Flow Graph Building

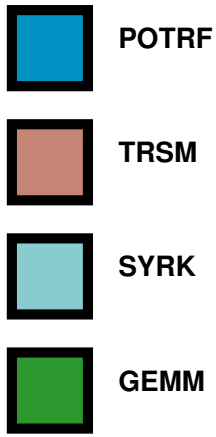
Example: Cholesky Decomposition

```
for (j = 0; j < N; j++) {
  task_insert( POTRF (RW,A[j][j]) );
  for (i = j+1; i < N; i++)
    task_insert( TRSM (RW,A[i][j], R,A[j][j]) );
  for (i = j+1; i < N; i++) {
    task_insert( SYRK (RW,A[i][i], R,A[i][j]) );
    for (k = j+1; k < i; k++)
      task_insert( GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]) );
  }
}
wait_for_all();
```





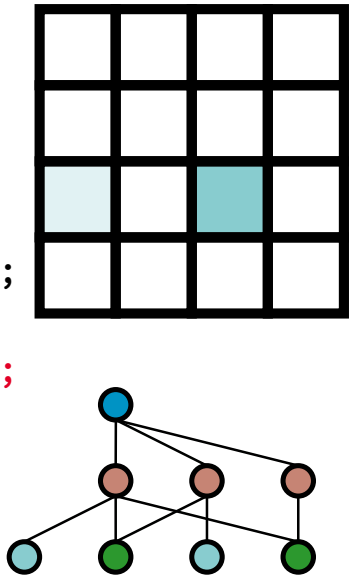
- Tasks are submitted asynchronously
- StarPU infers data dependences...
- ... and build a graph of tasks



Sequential Task Flow Graph Building

Example: Cholesky Decomposition

```
for (j = 0; j < N; j++) {
  task_insert( POTRF (RW,A[j][j]) );
  for (i = j+1; i < N; i++)
    task_insert( TRSM (RW,A[i][j], R,A[j][j]) );
  for (i = j+1; i < N; i++) {
    task_insert( SYRK (RW,A[i][i], R,A[i][j]) );
    for (k = j+1; k < i; k++)
      task_insert( GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]) );
  }
}
wait_for_all();
```

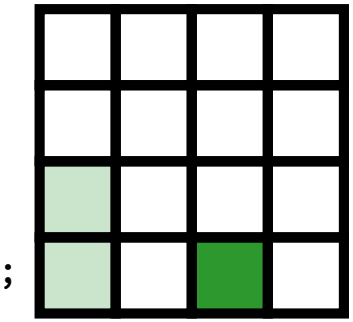


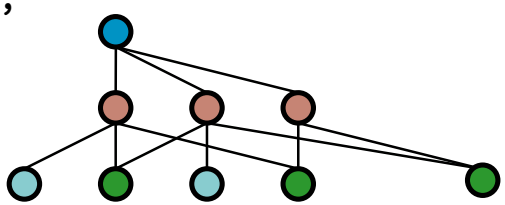
- Tasks are submitted asynchronously
- StarPU infers data dependences...
- ... and build a graph of tasks

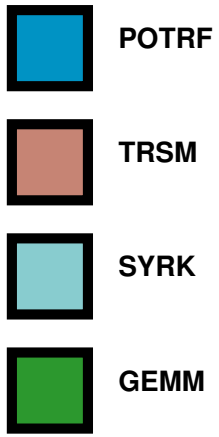
Sequential Task Flow Graph Building

Example: Cholesky Decomposition

```
for (j = 0; j < N; j++) {
  task_insert( POTRF (RW,A[j][j]) );
  for (i = j+1; i < N; i++)
    task_insert( TRSM (RW,A[i][j], R,A[j][j]) );
  for (i = j+1; i < N; i++) {
    task_insert( SYRK (RW,A[i][i], R,A[i][j]) );
    for (k = j+1; k < i; k++)
      task_insert( GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]) );
  }
}
wait_for_all();
```





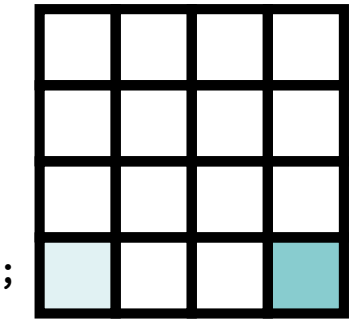


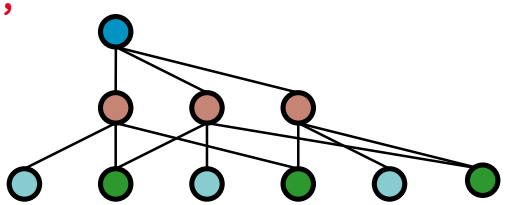
- Tasks are submitted asynchronously
- StarPU infers data dependences...
- ... and build a graph of tasks

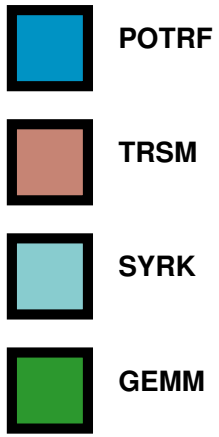
Sequential Task Flow Graph Building

Example: Cholesky Decomposition

```
for (j = 0; j < N; j++) {
  task_insert( POTRF (RW,A[j][j]) );
  for (i = j+1; i < N; i++)
    task_insert( TRSM (RW,A[i][j], R,A[j][j]) );
  for (i = j+1; i < N; i++) {
    task_insert( SYRK (RW,A[i][i], R,A[i][j]) );
    for (k = j+1; k < i; k++)
      task_insert( GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]) );
  }
}
wait_for_all();
```





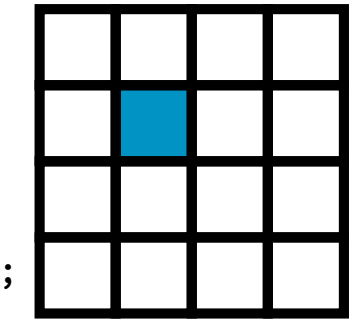


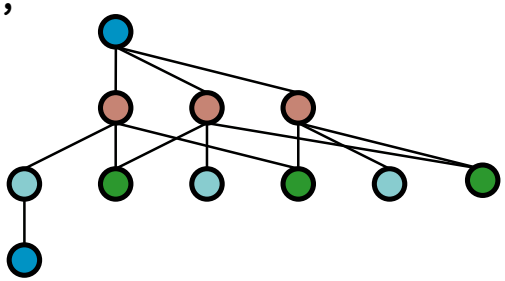
- Tasks are submitted asynchronously
- StarPU infers data dependences...
- ... and build a graph of tasks

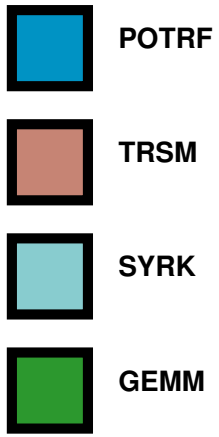
Sequential Task Flow Graph Building

Example: Cholesky Decomposition

```
for (j = 0; j < N; j++) {
  task_insert( POTRF (RW,A[j][j]) );
  for (i = j+1; i < N; i++)
    task_insert( TRSM (RW,A[i][j], R,A[j][j]) );
  for (i = j+1; i < N; i++) {
    task_insert( SYRK (RW,A[i][i], R,A[i][j]) );
    for (k = j+1; k < i; k++)
      task_insert( GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]) );
  }
}
wait_for_all();
```





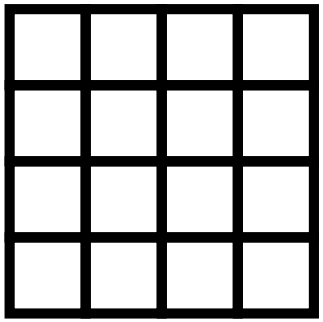


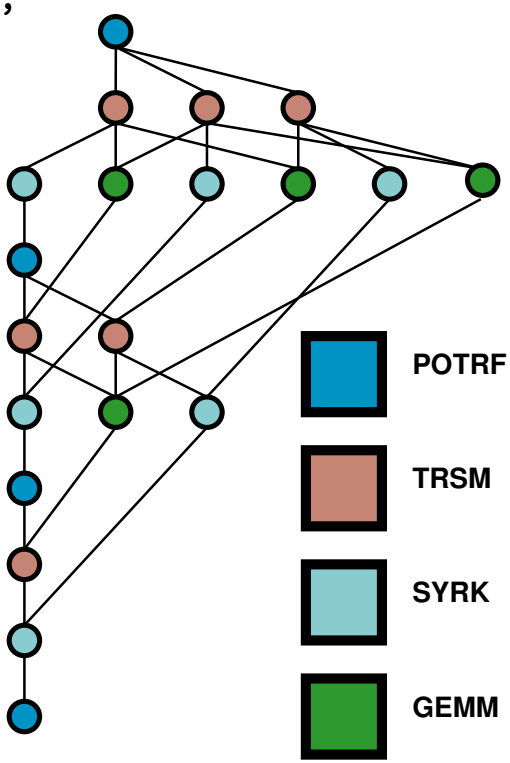
- Tasks are submitted asynchronously
- StarPU infers data dependences...
- ... and build a graph of tasks

Sequential Task Flow Graph Building

Example: Cholesky Decomposition

```
for (j = 0; j < N; j++) {
  task_insert( POTRF (RW,A[j][j]) );
  for (i = j+1; i < N; i++)
    task_insert( TRSM (RW,A[i][j], R,A[j][j]) );
  for (i = j+1; i < N; i++) {
    task_insert( SYRK (RW,A[i][i], R,A[i][j]) );
    for (k = j+1; k < i; k++)
      task_insert( GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]) );
  }
}
wait_for_all();
```





- Tasks are submitted asynchronously
- StarPU infers data dependences...
- ... and build a graph of tasks
- The graph of tasks is executed

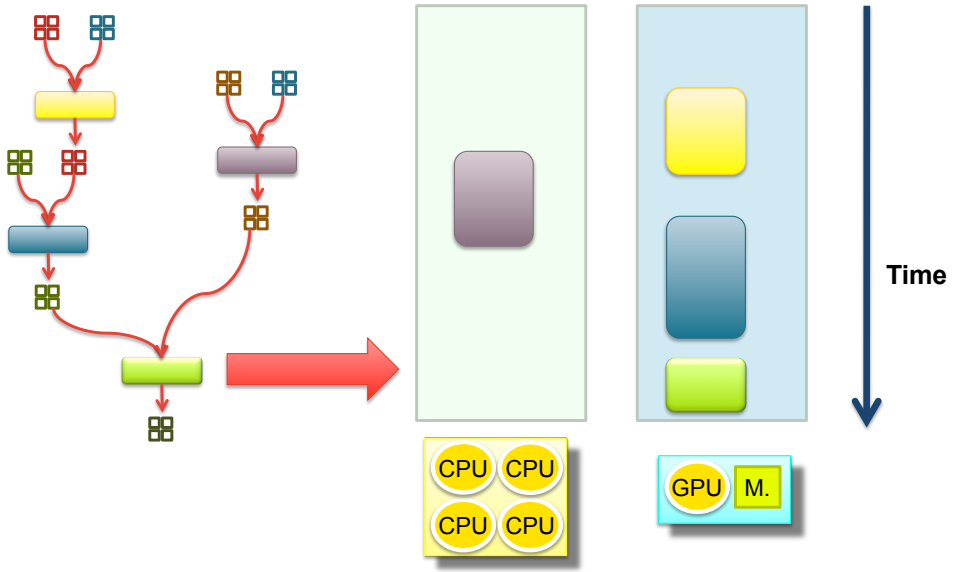
StarPU Execution Model: Task Scheduling

Mapping the graph of tasks (DAG) on the hardware

- Allocating computing resources
- Enforcing dependency constraints
- Handling data transfers

Adaptiveness

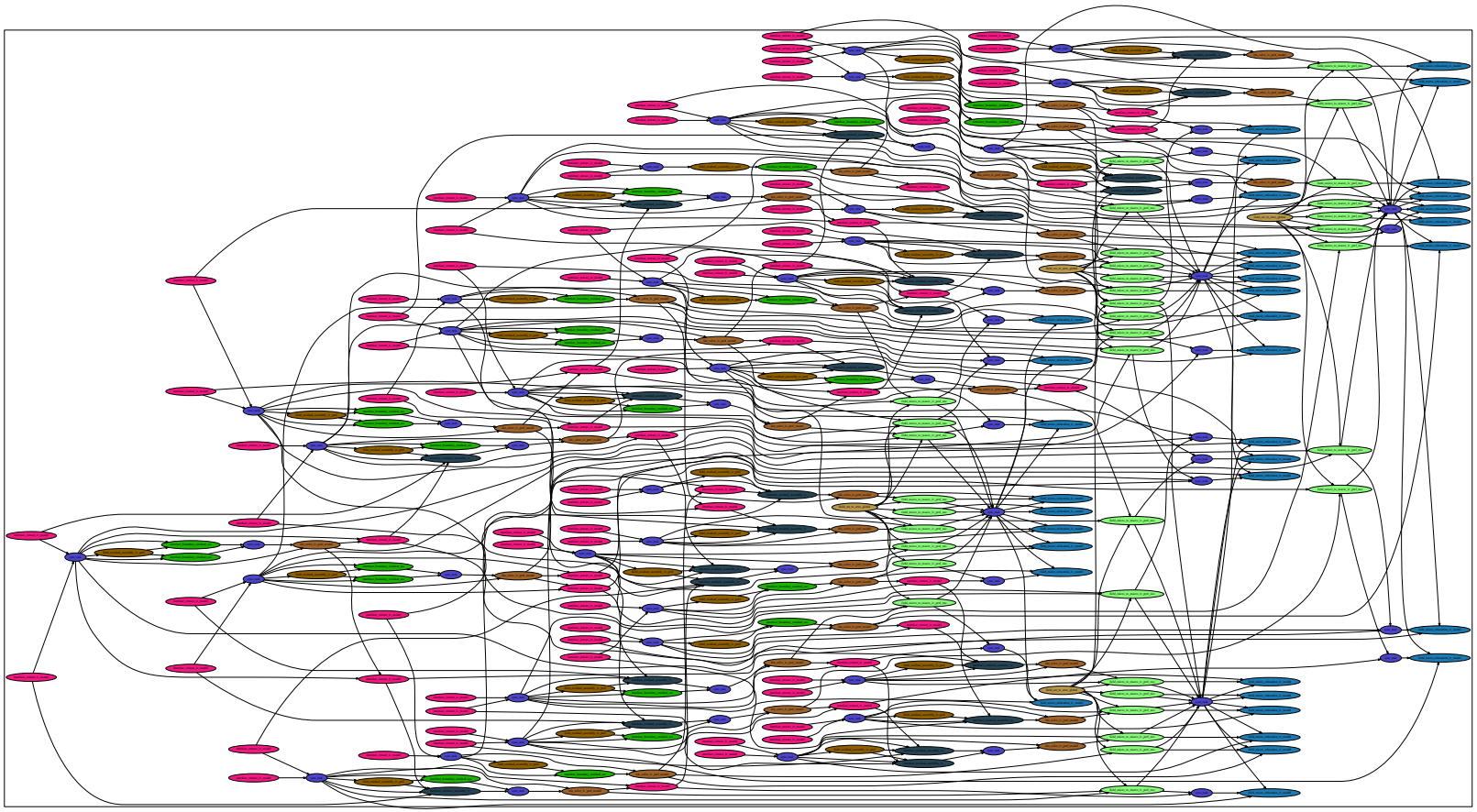
- A single DAG enables multiple schedulings
- A single DAG can be mapped on multiple platforms



Example: SCHNAPS, Implicit kinetic schemes

SCHNAPS Solver (Inria TONUS)

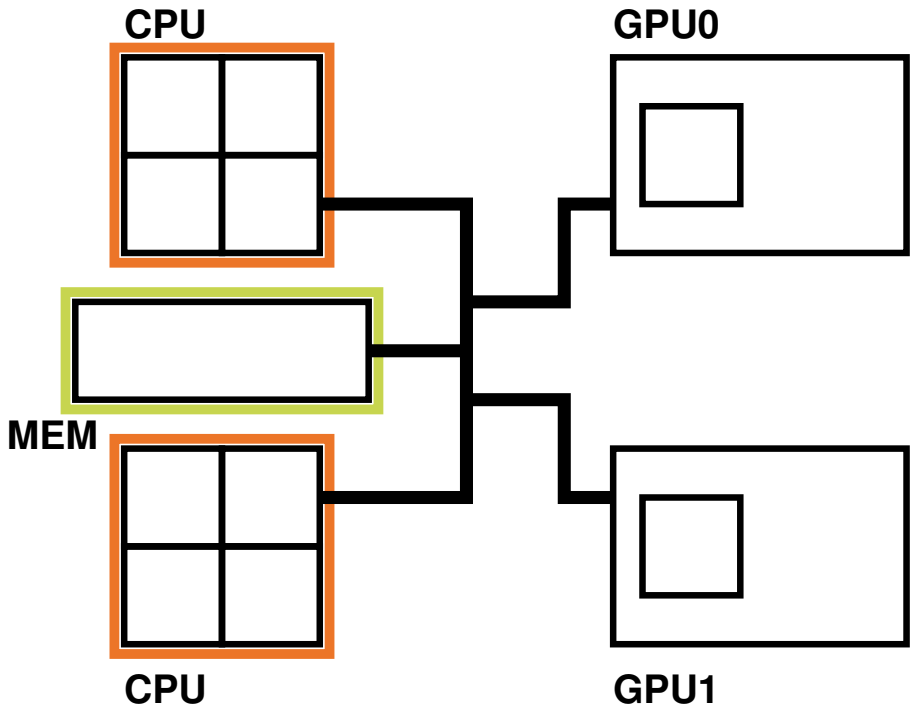
- Example of a task graph submitted to StarPU



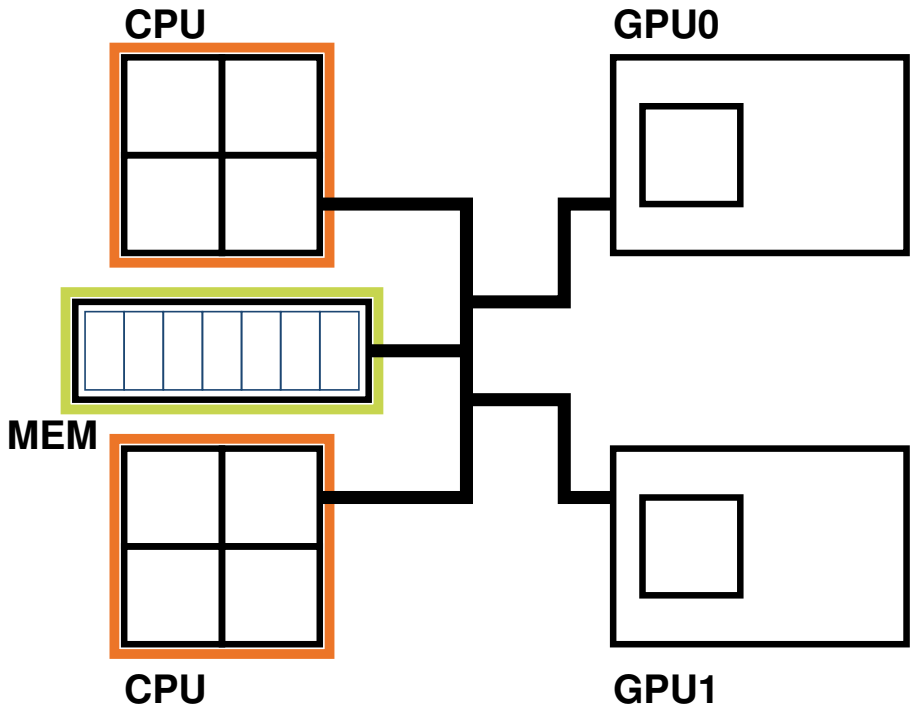
2.2

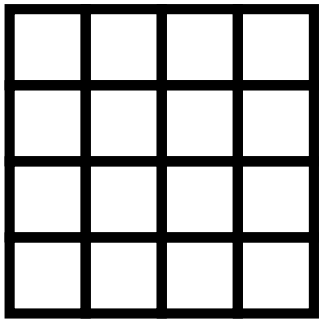
Now, Leverage an *Accelerated* Computing Node

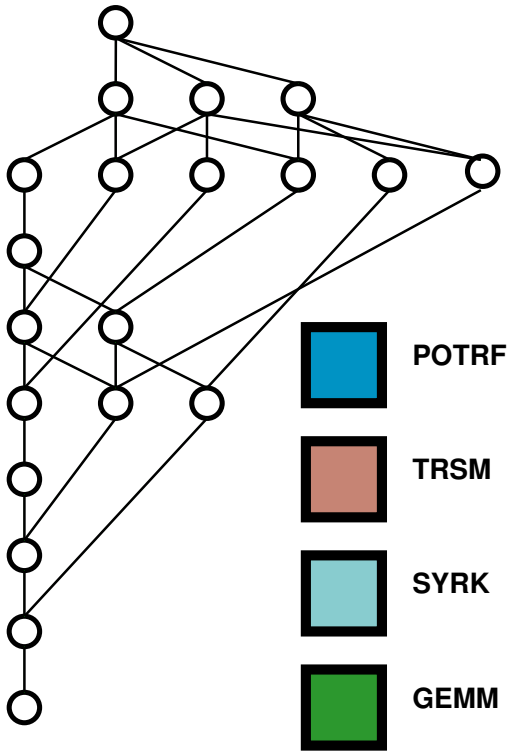
StarPU Heterogeneous Execution Model / Data Management



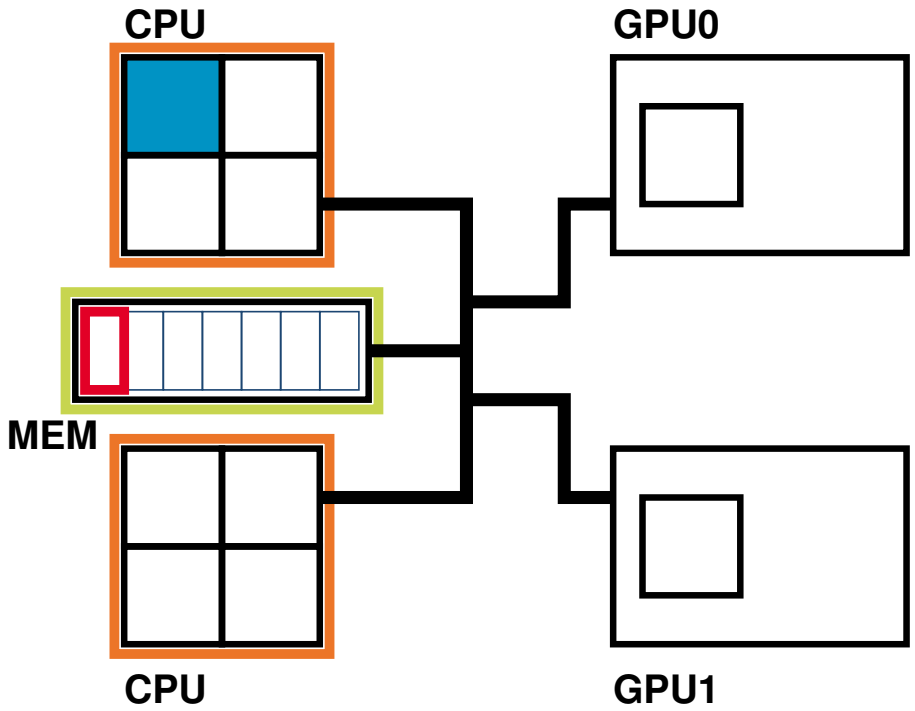
StarPU Heterogeneous Execution Model / Data Management

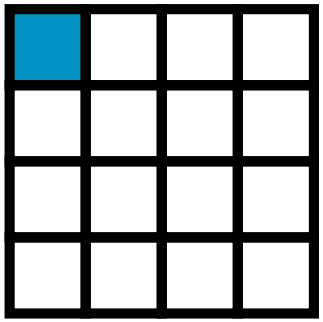


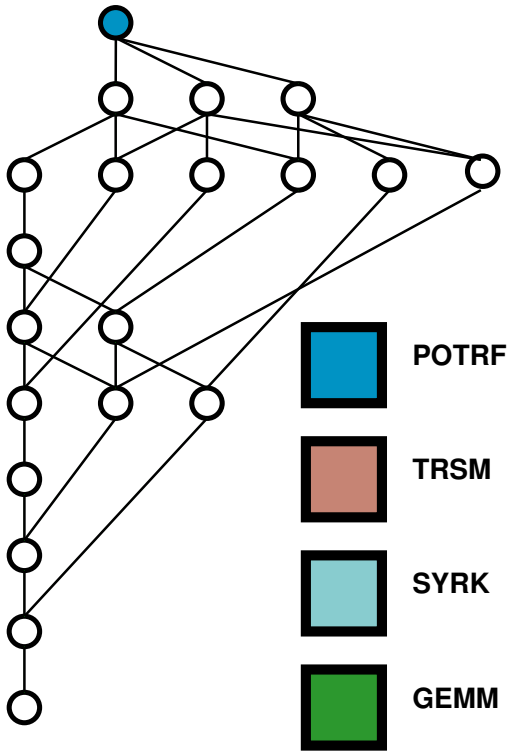




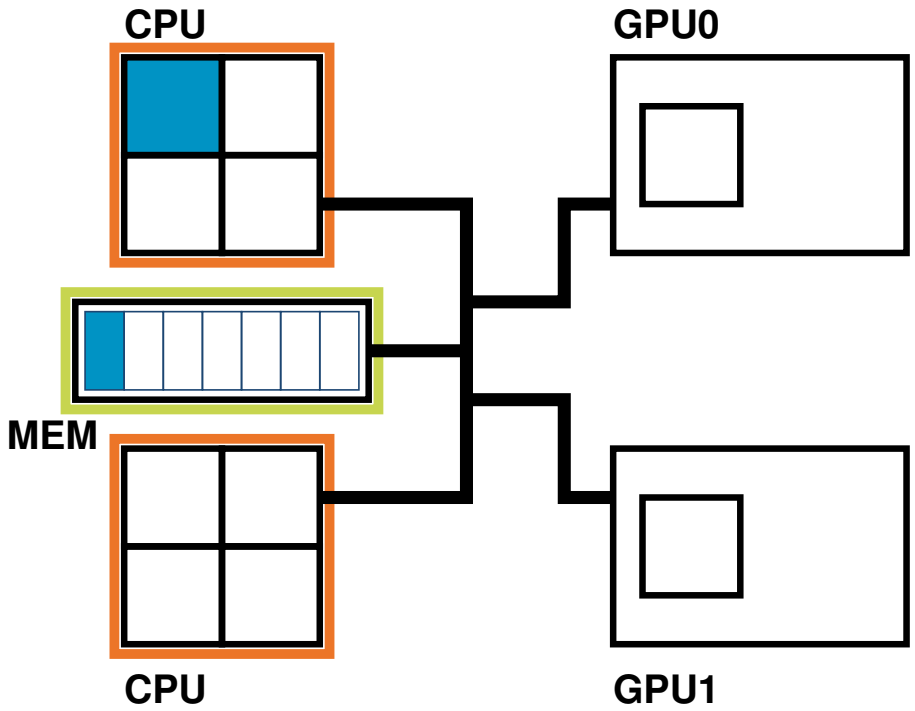
StarPU Heterogeneous Execution Model / Data Management

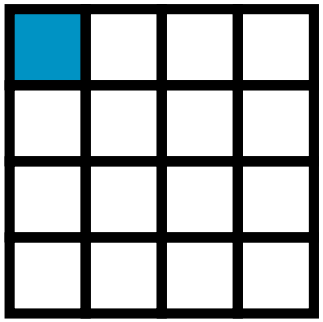


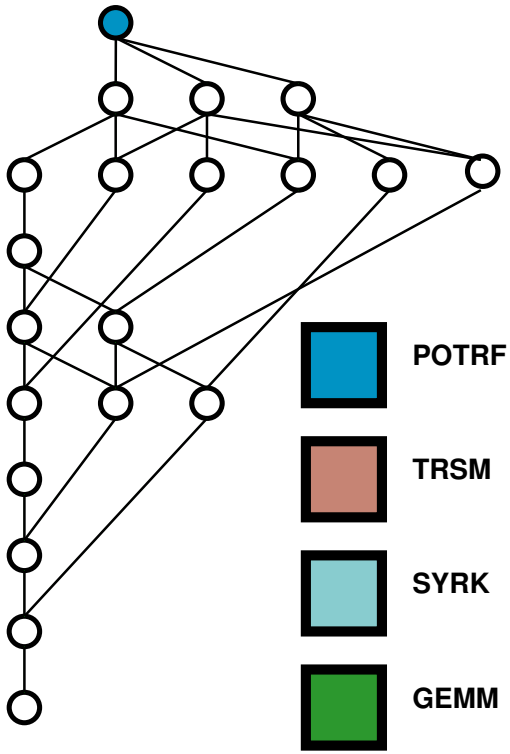




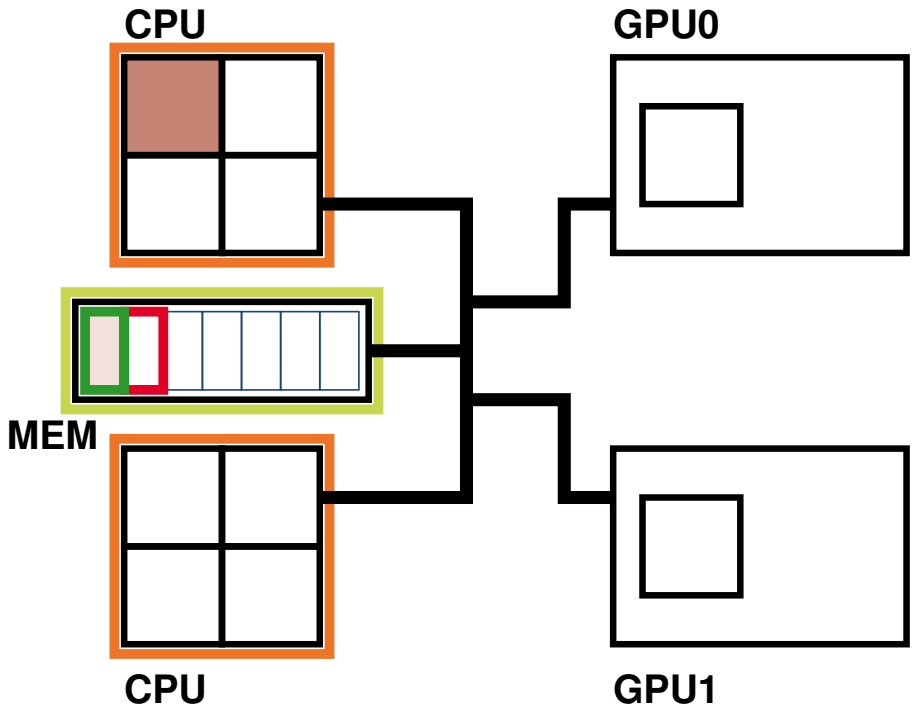
StarPU Heterogeneous Execution Model / Data Management

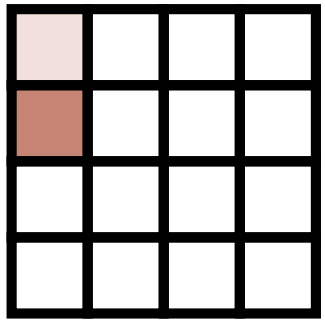


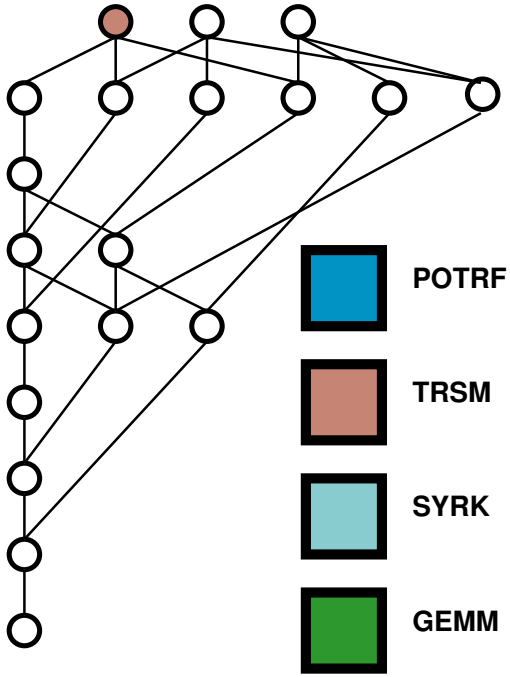




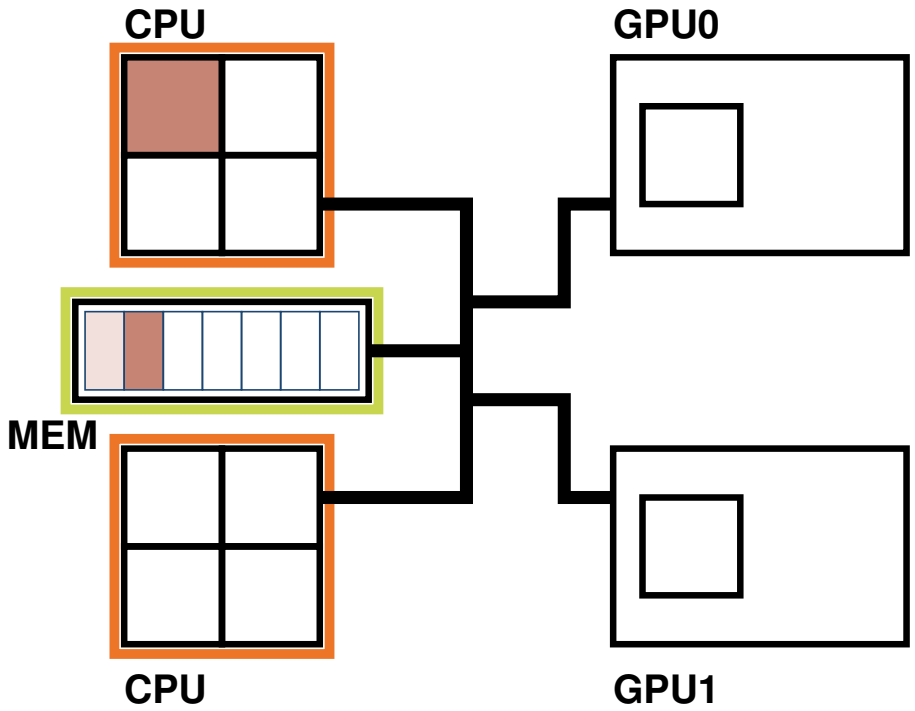
StarPU Heterogeneous Execution Model / Data Management

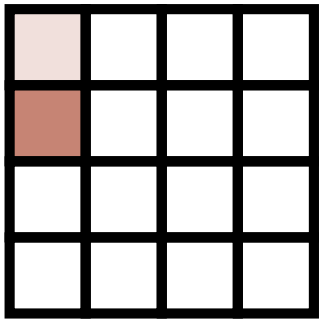


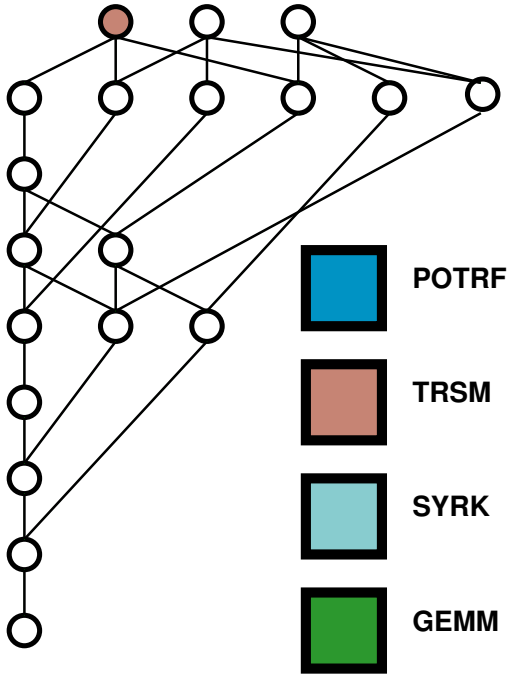




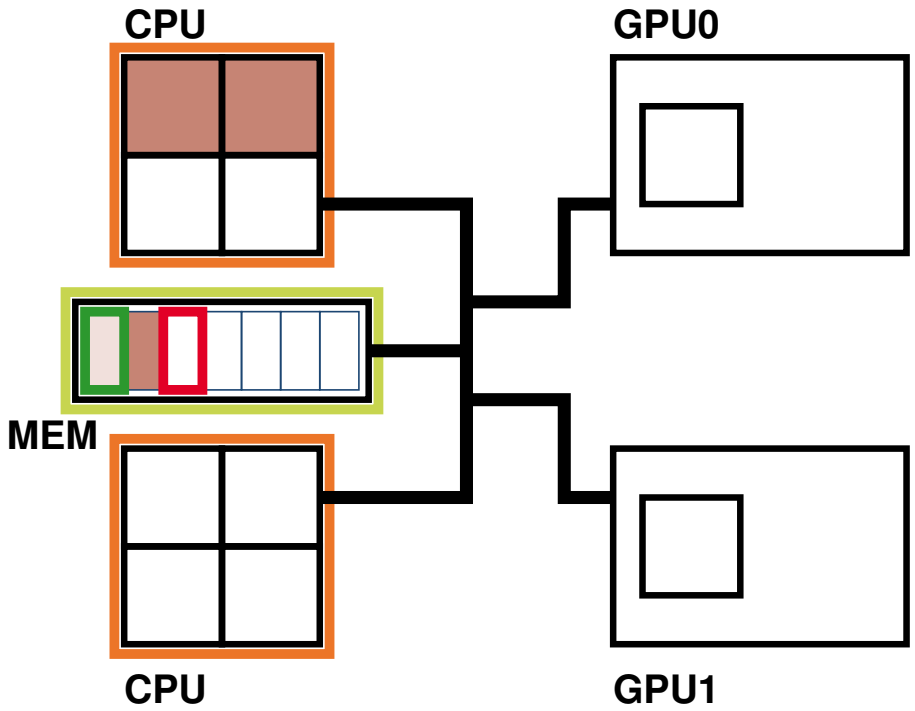
StarPU Heterogeneous Execution Model / Data Management

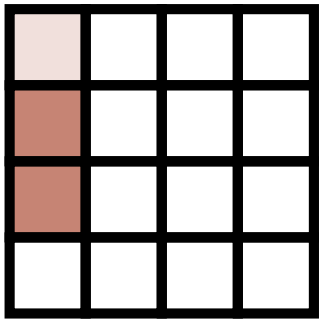


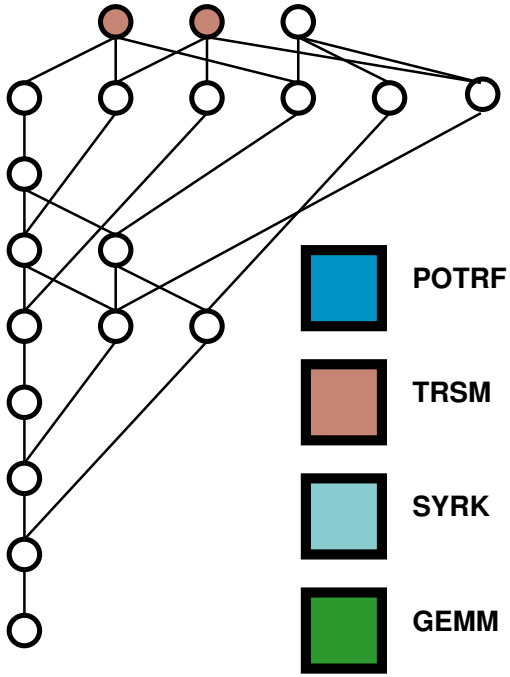




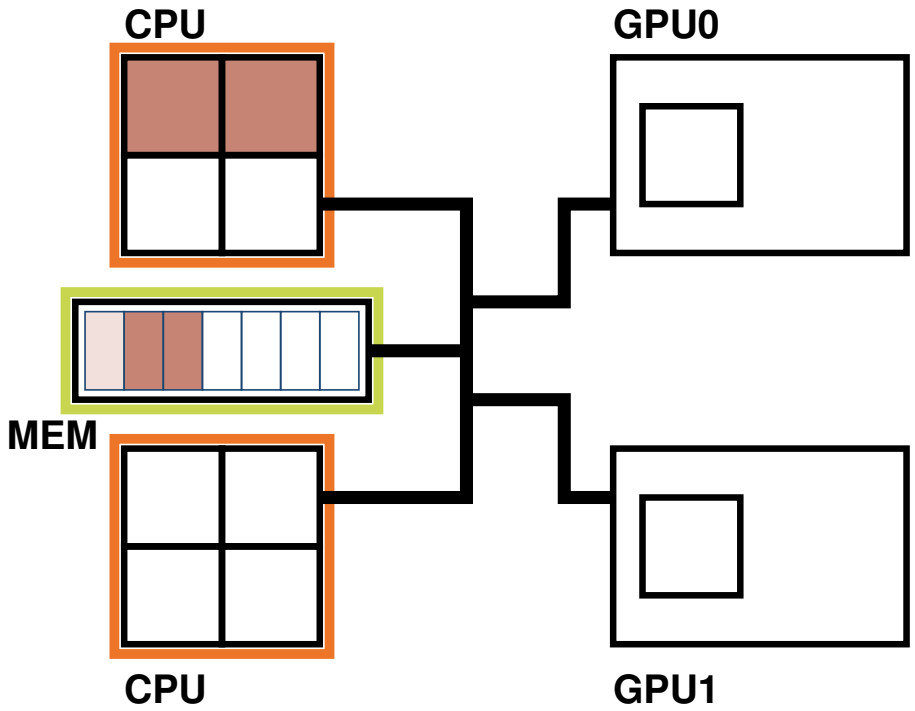
StarPU Heterogeneous Execution Model / Data Management

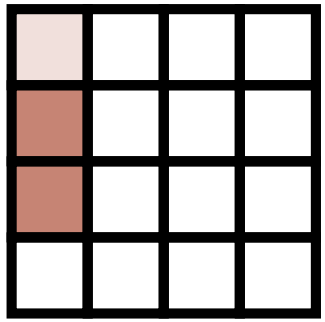


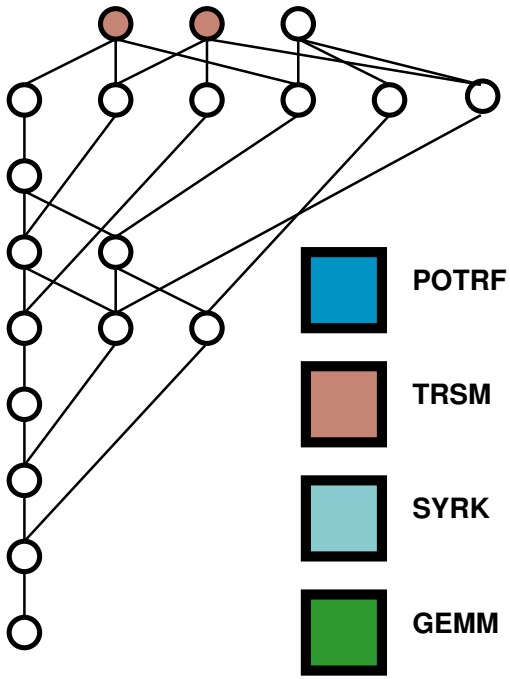




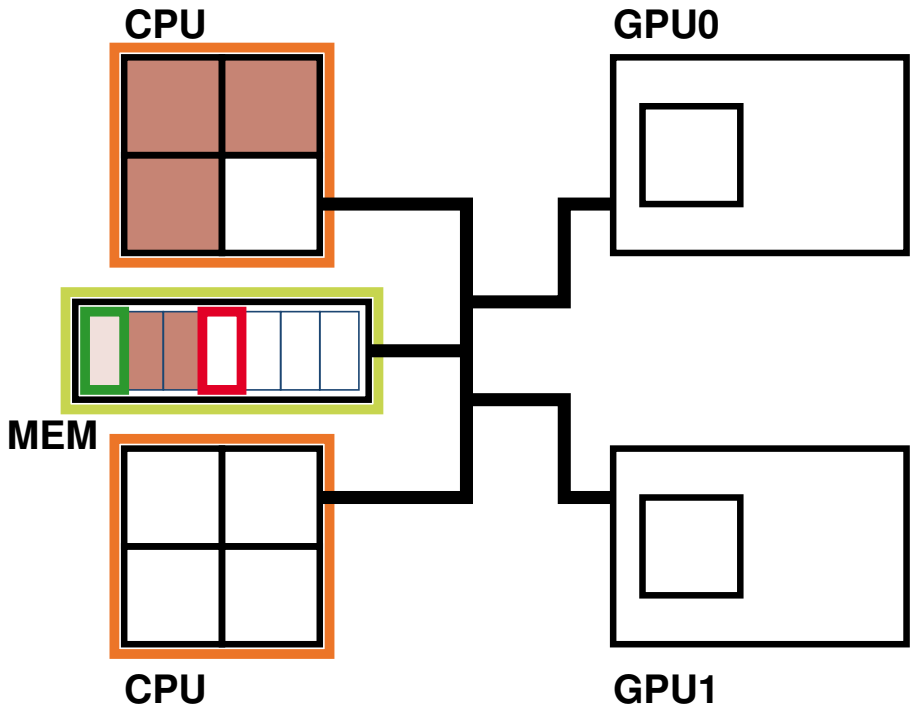
StarPU Heterogeneous Execution Model / Data Management

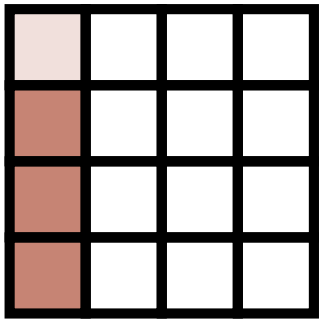


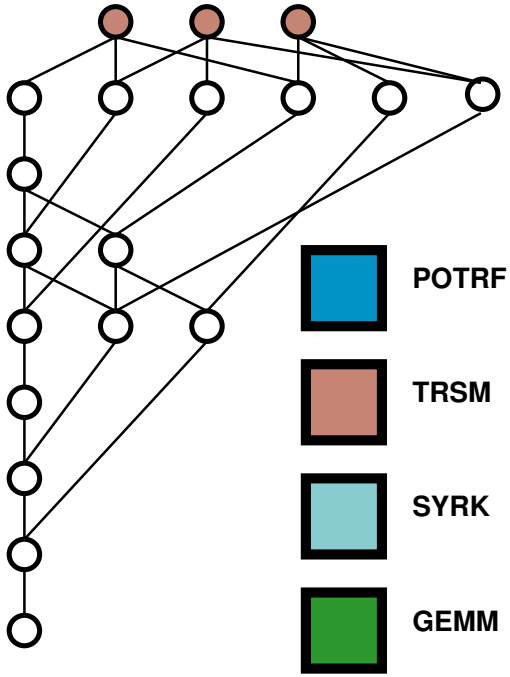




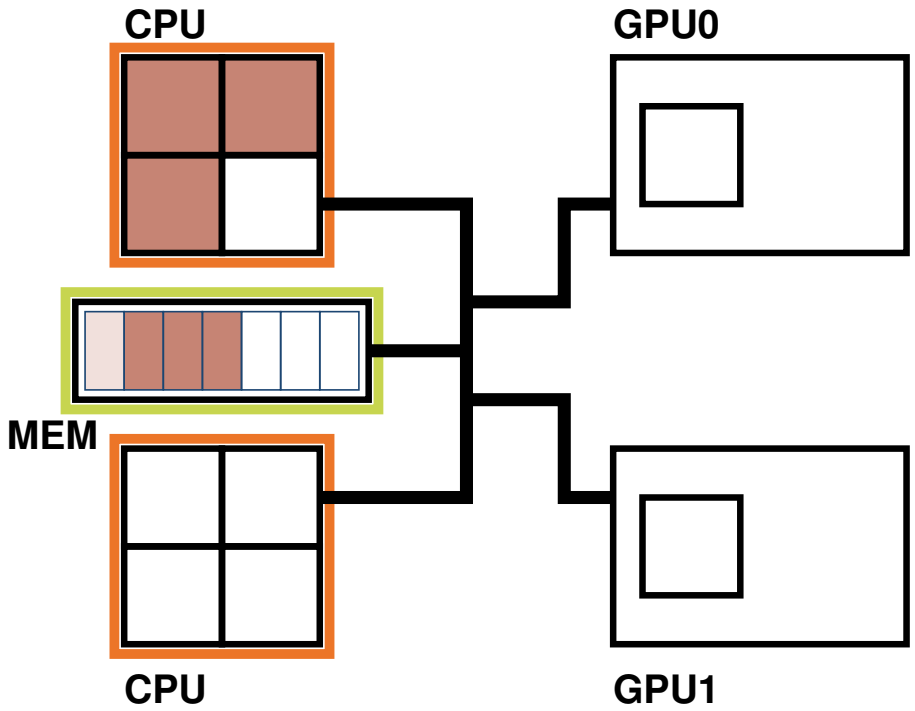
StarPU Heterogeneous Execution Model / Data Management

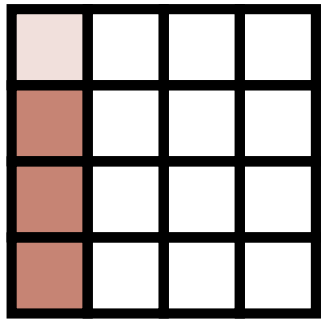


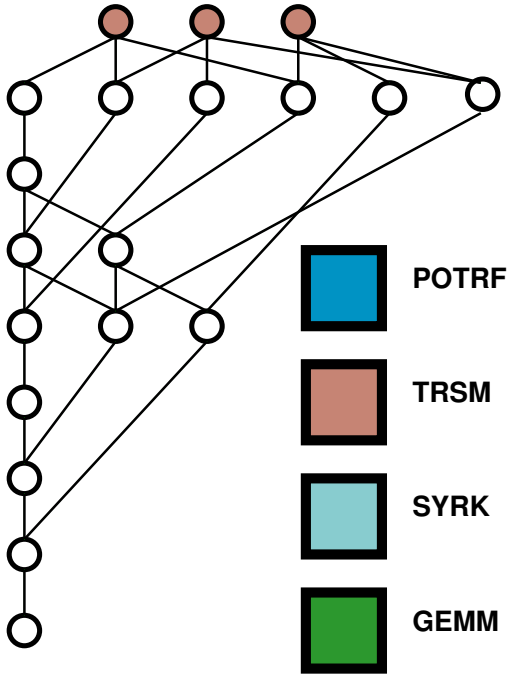




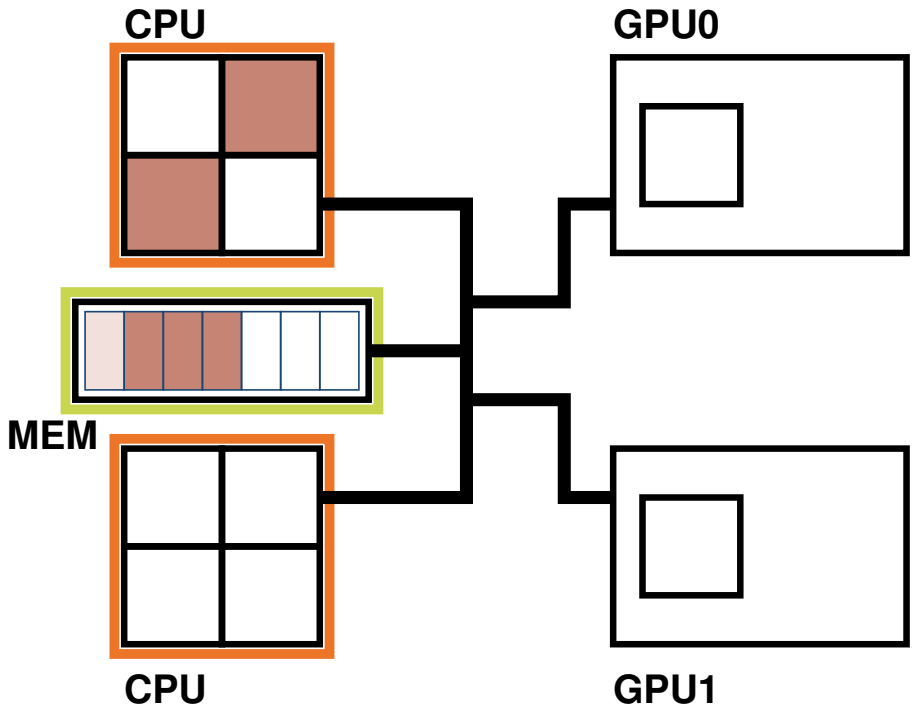
StarPU Heterogeneous Execution Model / Data Management

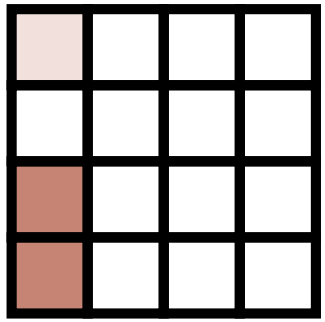


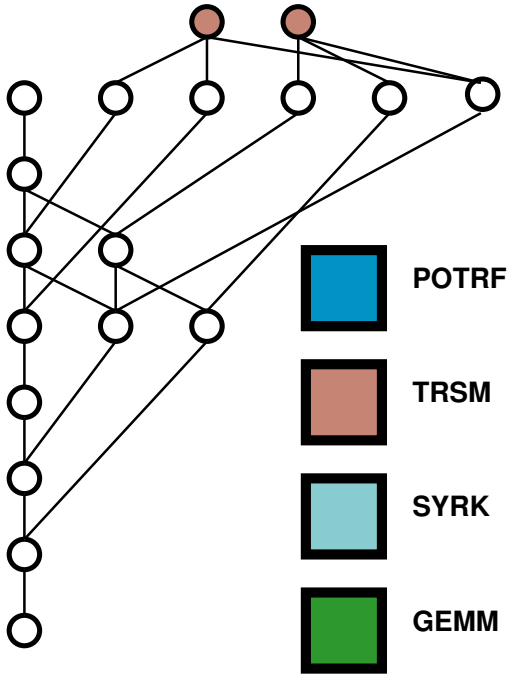




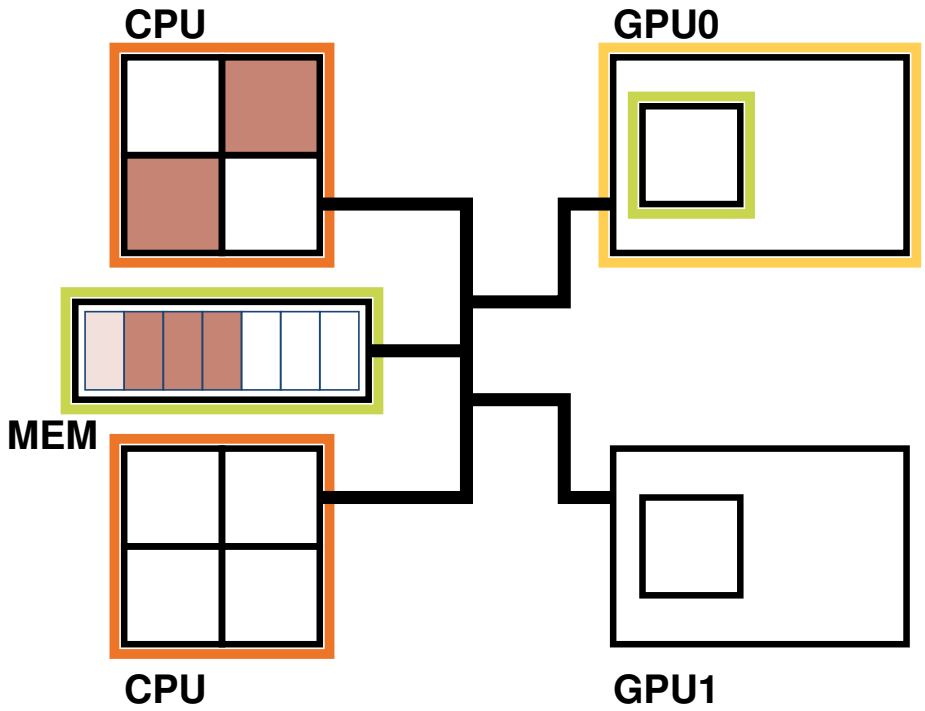
StarPU Heterogeneous Execution Model / Data Management

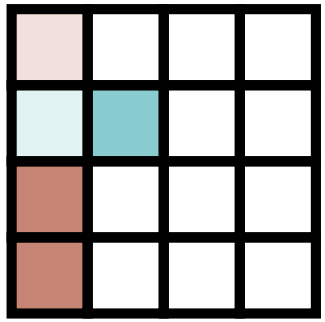


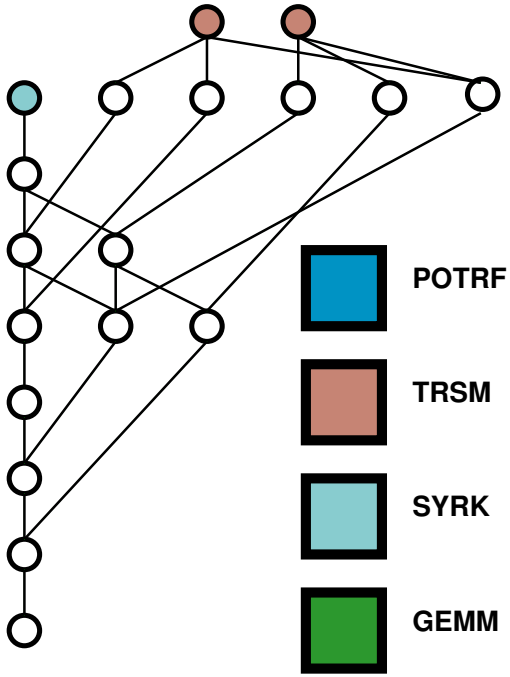




StarPU Heterogeneous Execution Model / Data Management

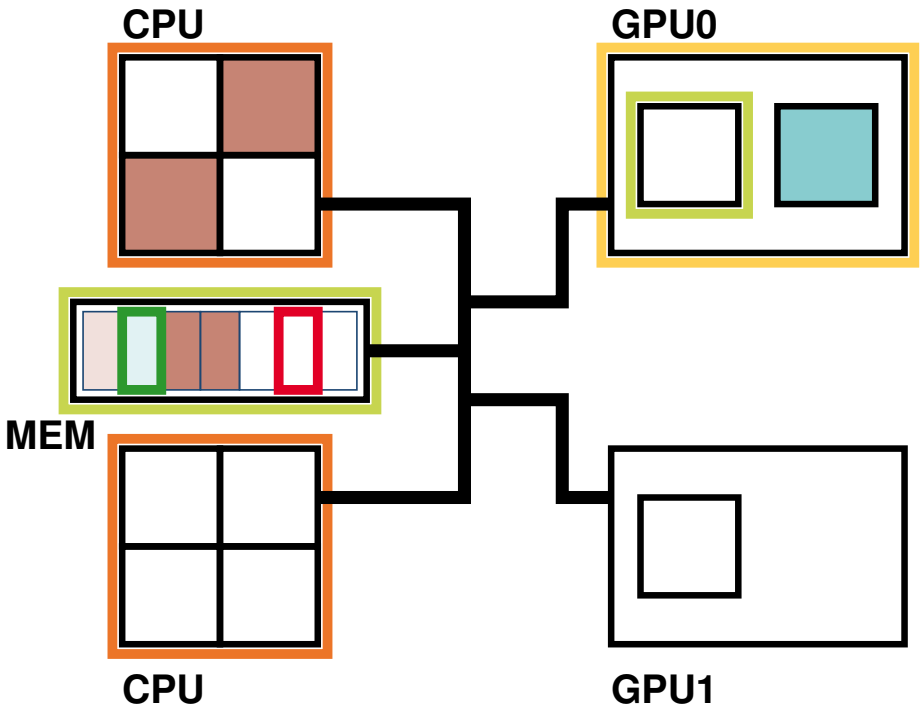


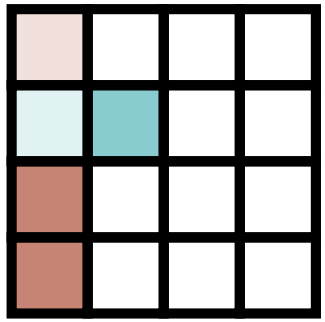


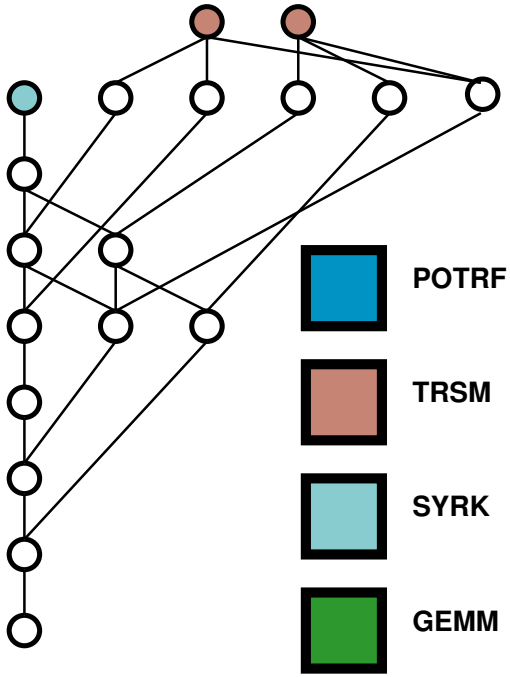


- **Dynamic accelerator selection**

StarPU Heterogeneous Execution Model / Data Management

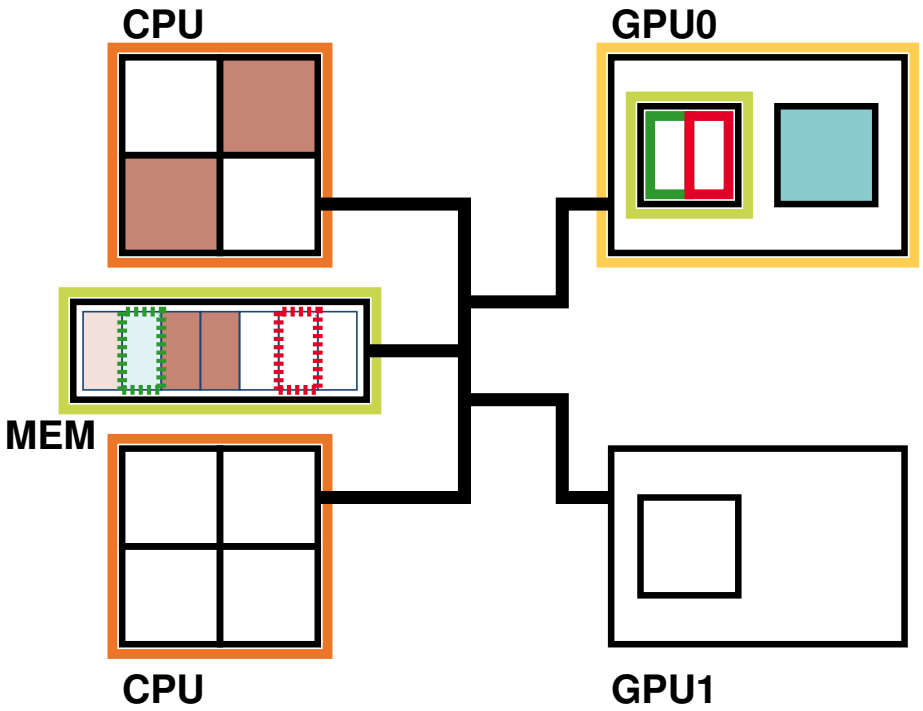


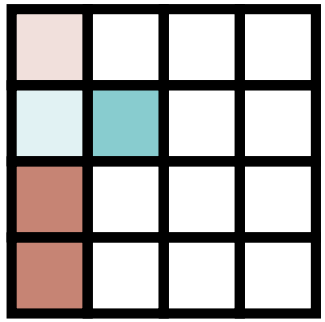


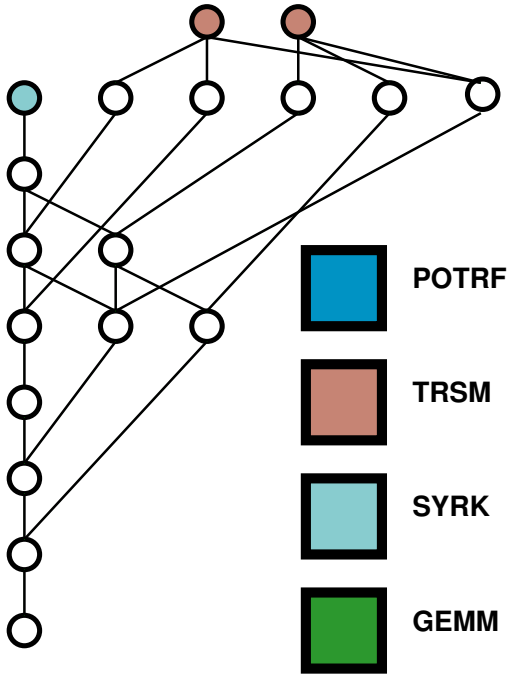


- Dynamic accelerator selection

StarPU Heterogeneous Execution Model / Data Management

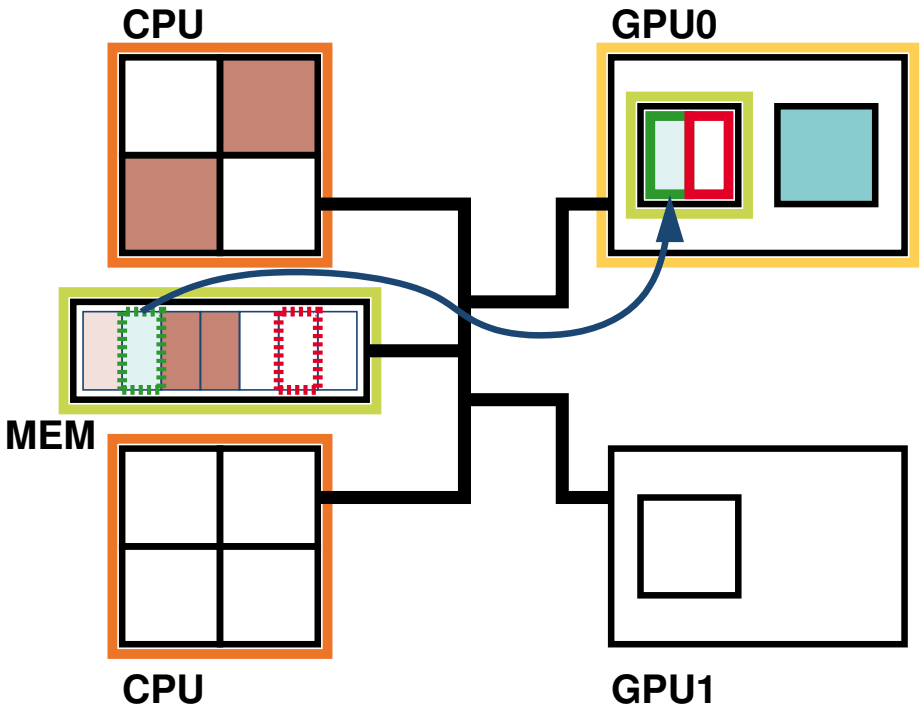


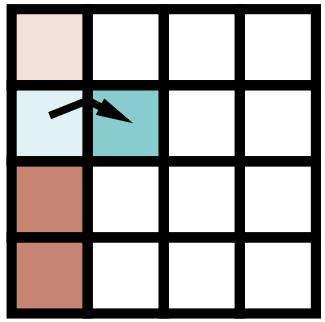


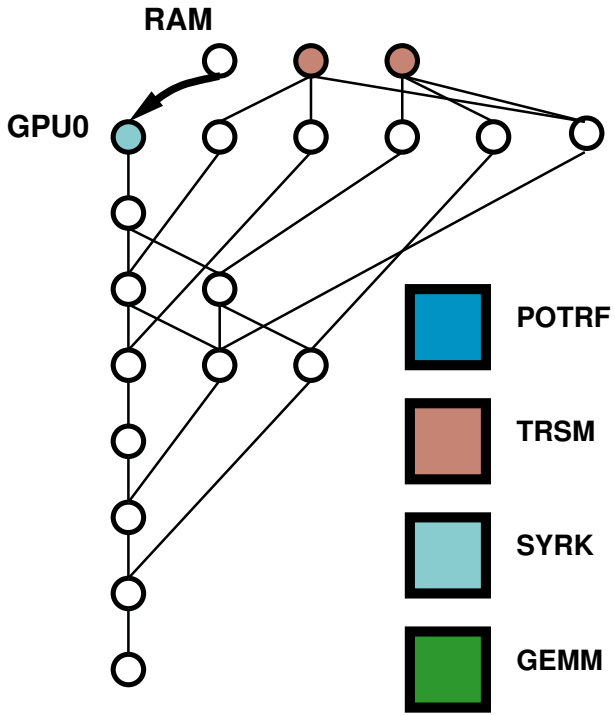


- Dynamic accelerator selection

StarPU Heterogeneous Execution Model / Data Management

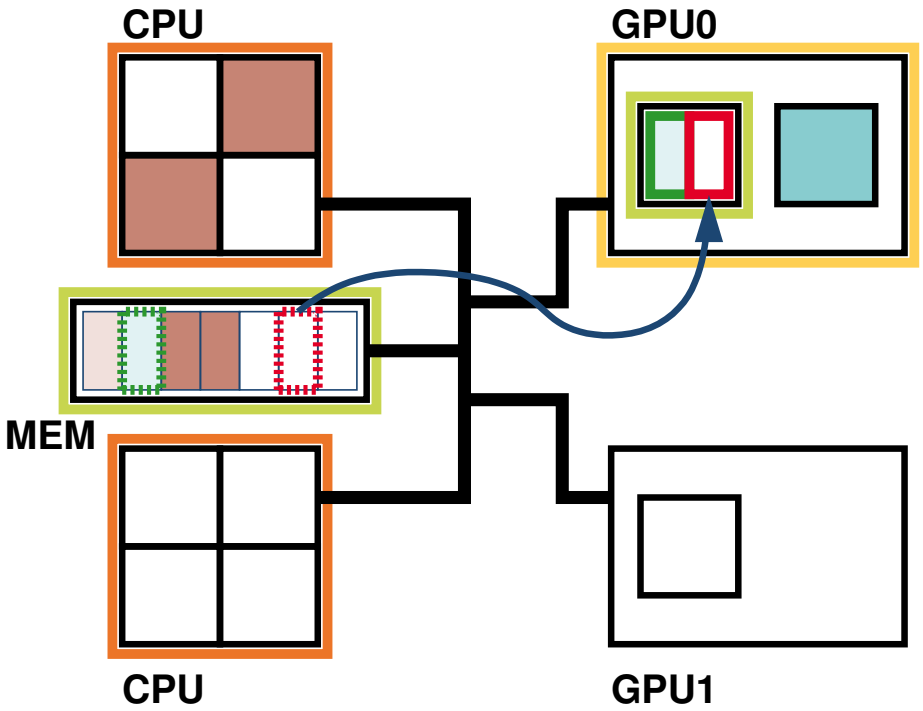


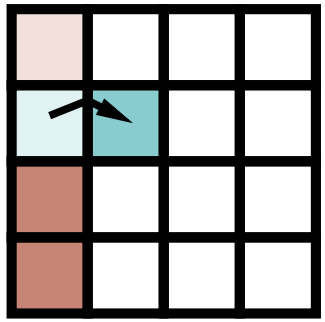


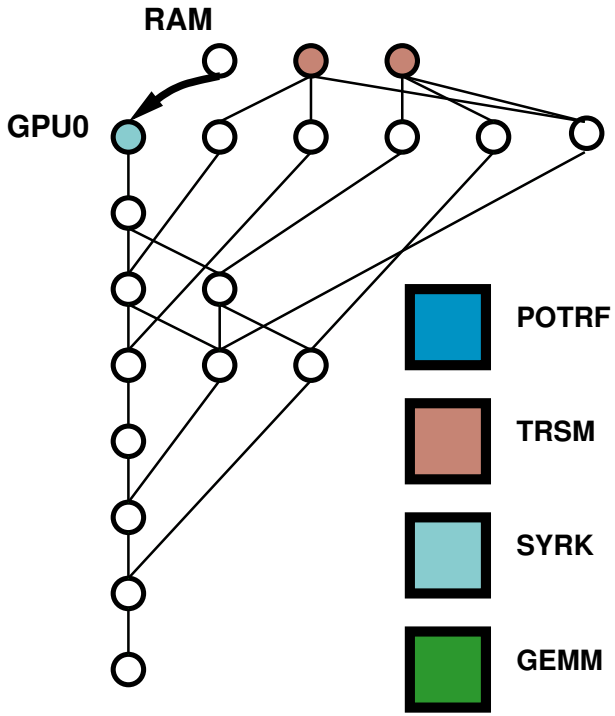


- Dynamic accelerator selection
- Inter-device dependencies

StarPU Heterogeneous Execution Model / Data Management

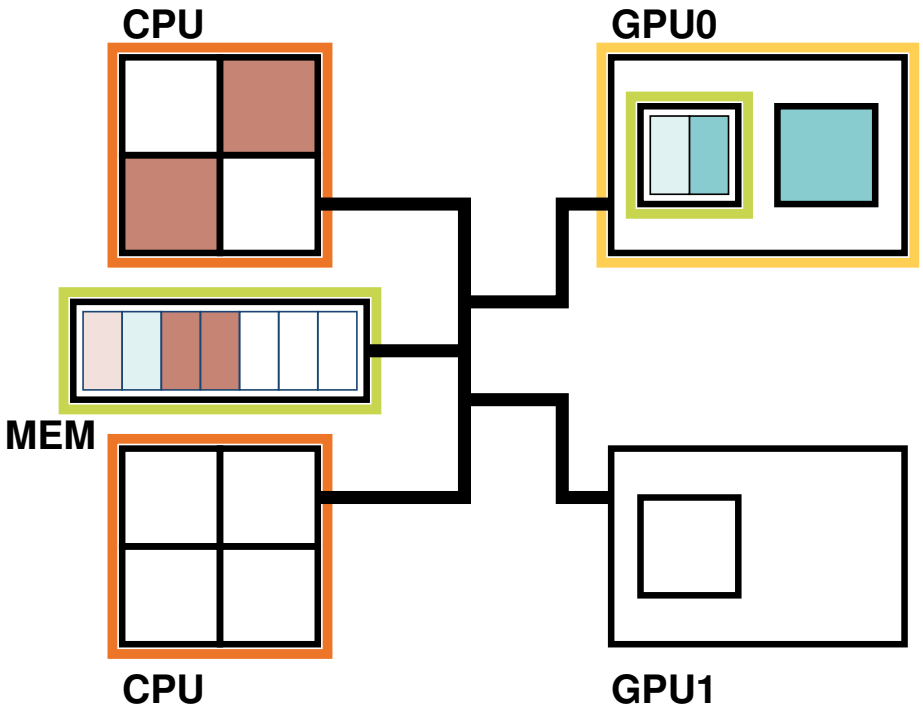


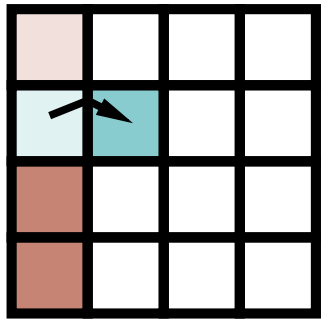


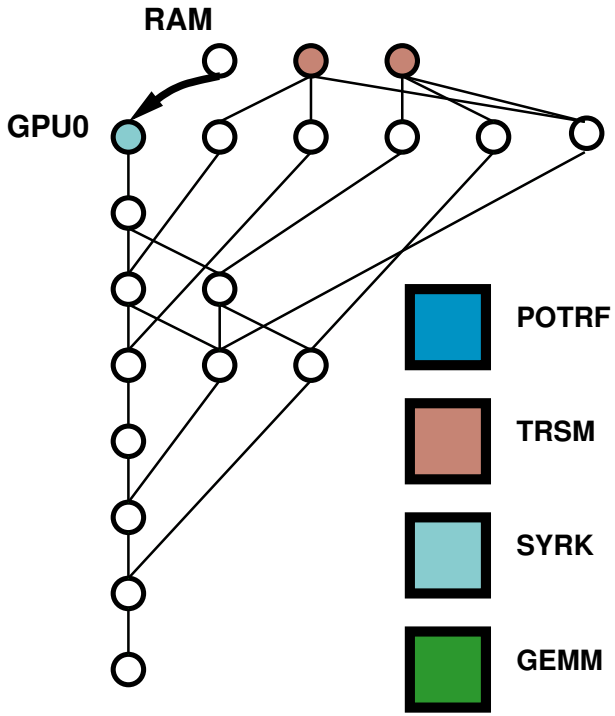


- **Dynamic accelerator selection**
- **Inter-device dependencies**
- Transparent data replicates
- Automatic data consistency management

StarPU Heterogeneous Execution Model / Data Management

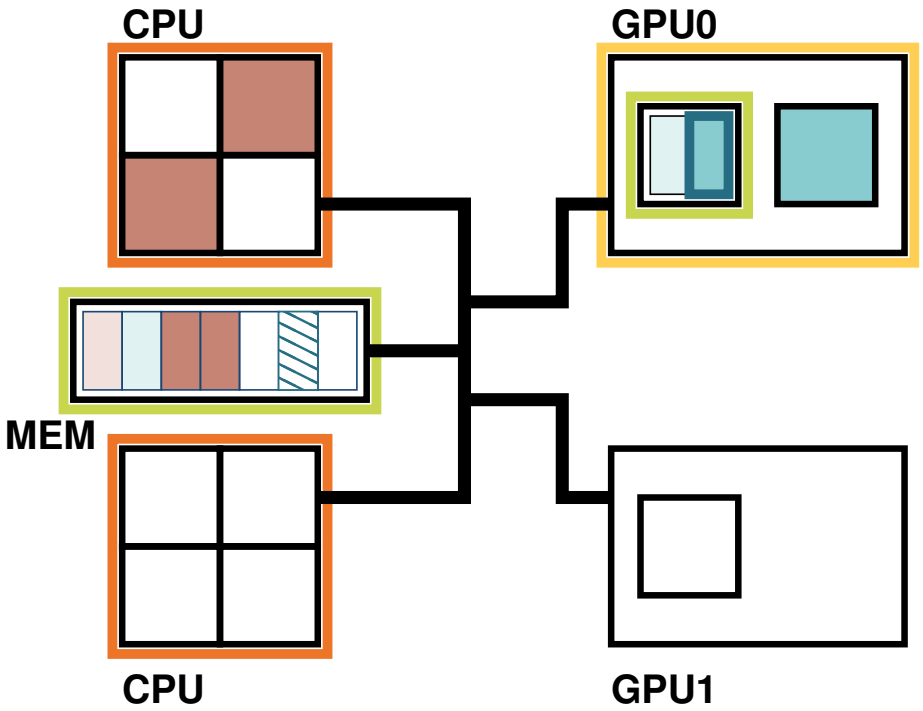


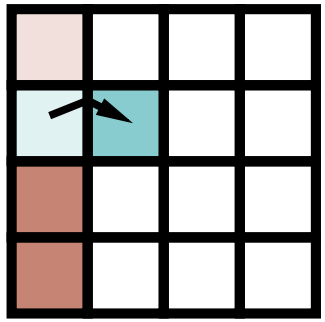


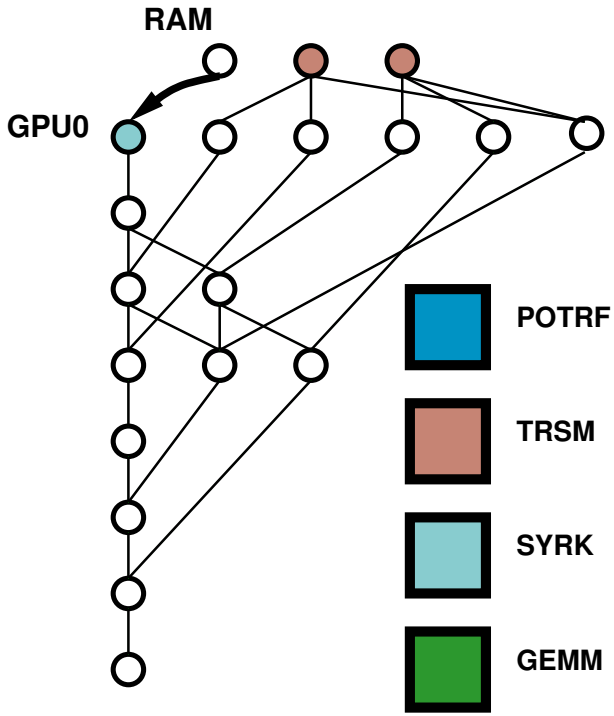


- **Dynamic accelerator selection**
- **Inter-device dependencies**
- Transparent data replicates
- Automatic data consistency management

StarPU Heterogeneous Execution Model / Data Management





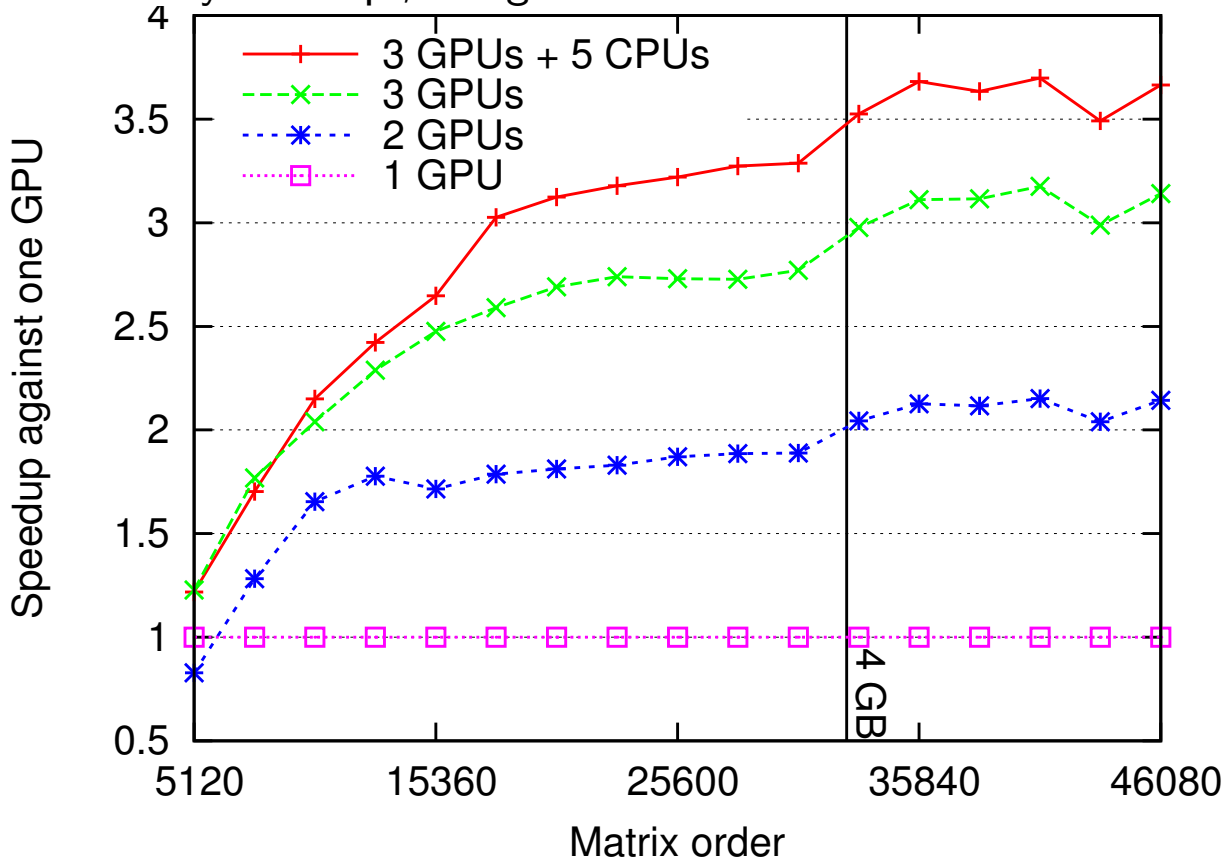


- **Dynamic accelerator selection**
- **Inter-device dependencies**
- Transparent data replicates
- Automatic data consistency management

Showcase with the Chameleon Linear Algebra Library

UTK, Inria HIEPACS, Inria RUNTIME

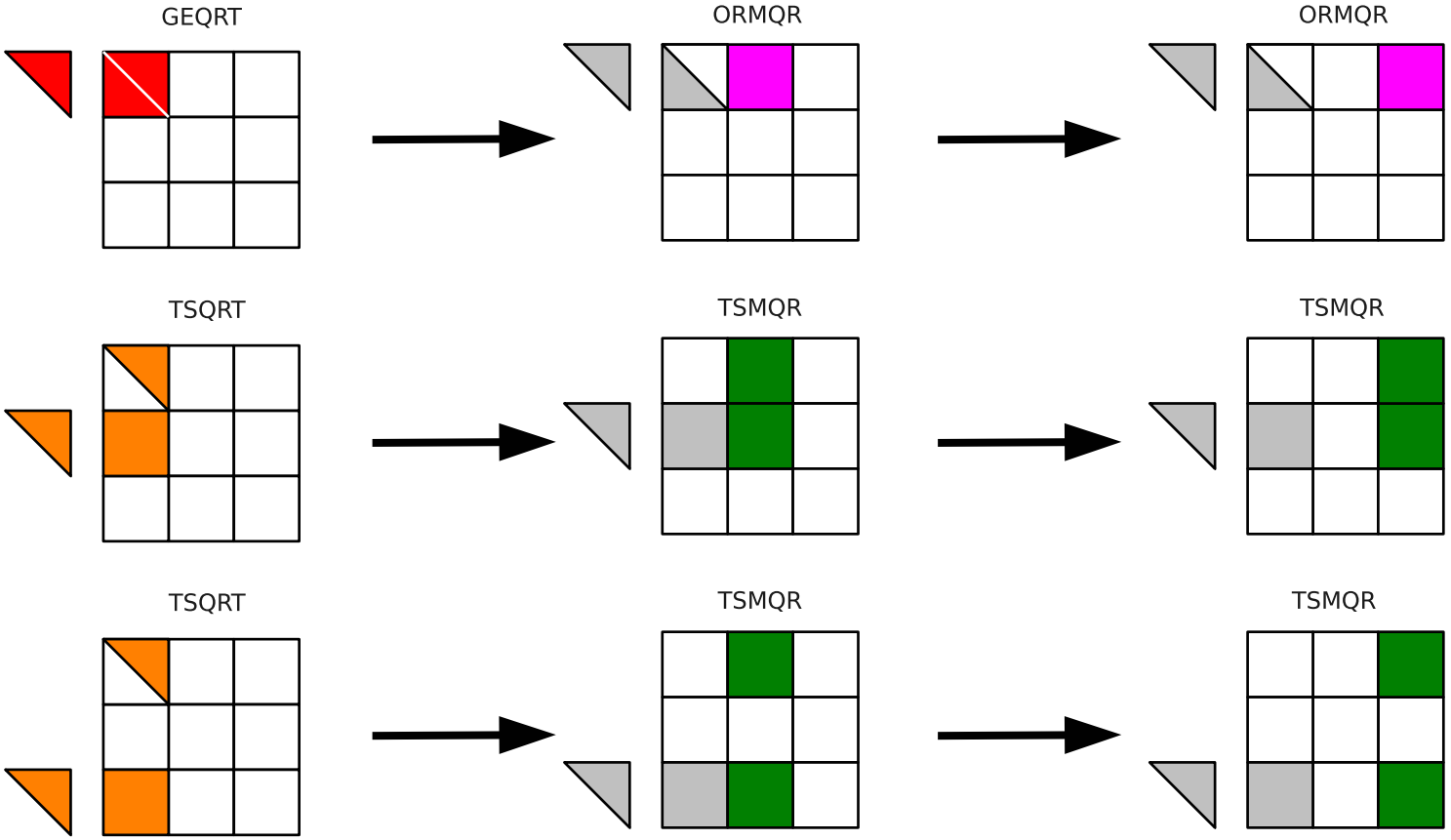
- Multi-GPU Cholesky decomp., using MAGMA GPU kernels



Showcase with the Chameleon Linear Algebra Library

UTK, Inria HIEPACS, Inria RUNTIME

- QR decomp. on 16 CPUs (AMD) + 4 GPUs (C1060) using MAGMA GPU kernels

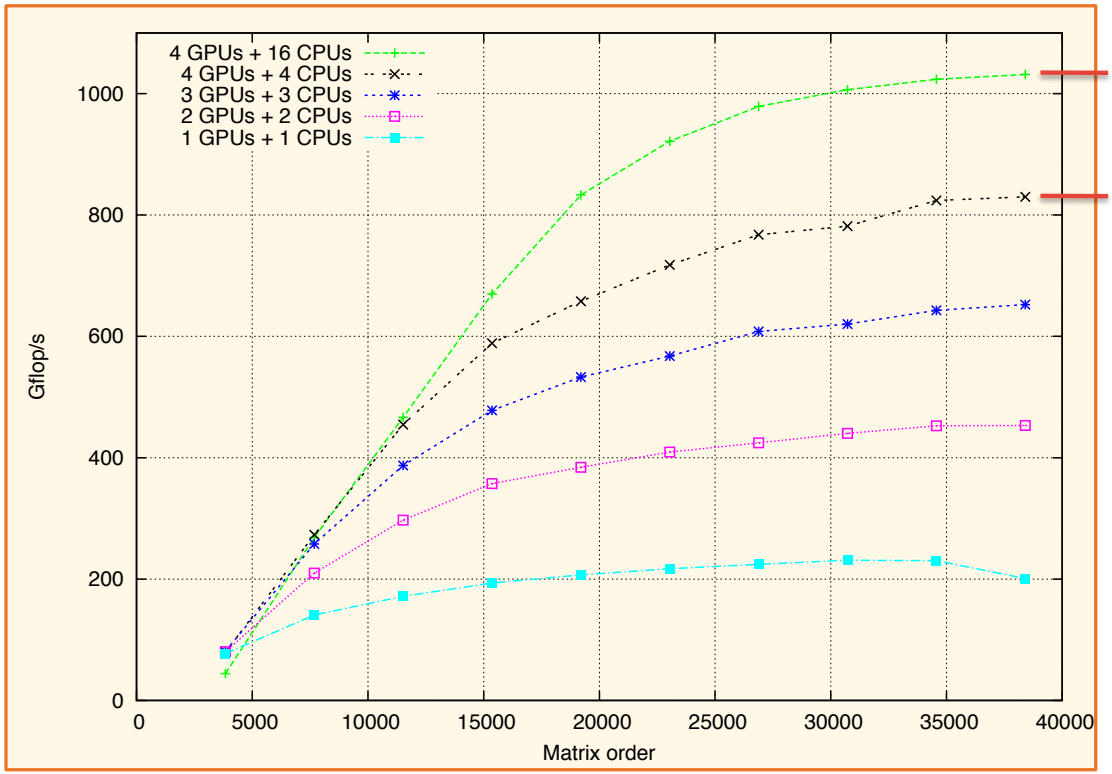


"E. Agullo, C. Augonnet, J. Dongarra, M. Faverge, H. Ltaief, et al. *QR Factorization on a Multicore Node Enhanced with Multiple GPU Accelerators*. 25th IEEE IPDPS, 2011."

Showcase with the Chameleon Linear Algebra Library

UTK, Inria HIEPACS, Inria RUNTIME

- QR decomp. on 16 CPUs (AMD) + 4 GPUs (C1060) using MAGMA GPU kernels



Measured increase:
+12 CPUs
~200 GFlops

Expected increase:
+12 CPUs
~150 Gflops

"E. Agullo, C. Augonnet, J. Dongarra, M. Faverge, H. Ltaief, et al. *QR Factorization on a Multicore Node Enhanced with Multiple GPU Accelerators*. 25th IEEE IPDPS, 2011."

Showcase with the Chameleon Linear Algebra Library

QR kernel properties

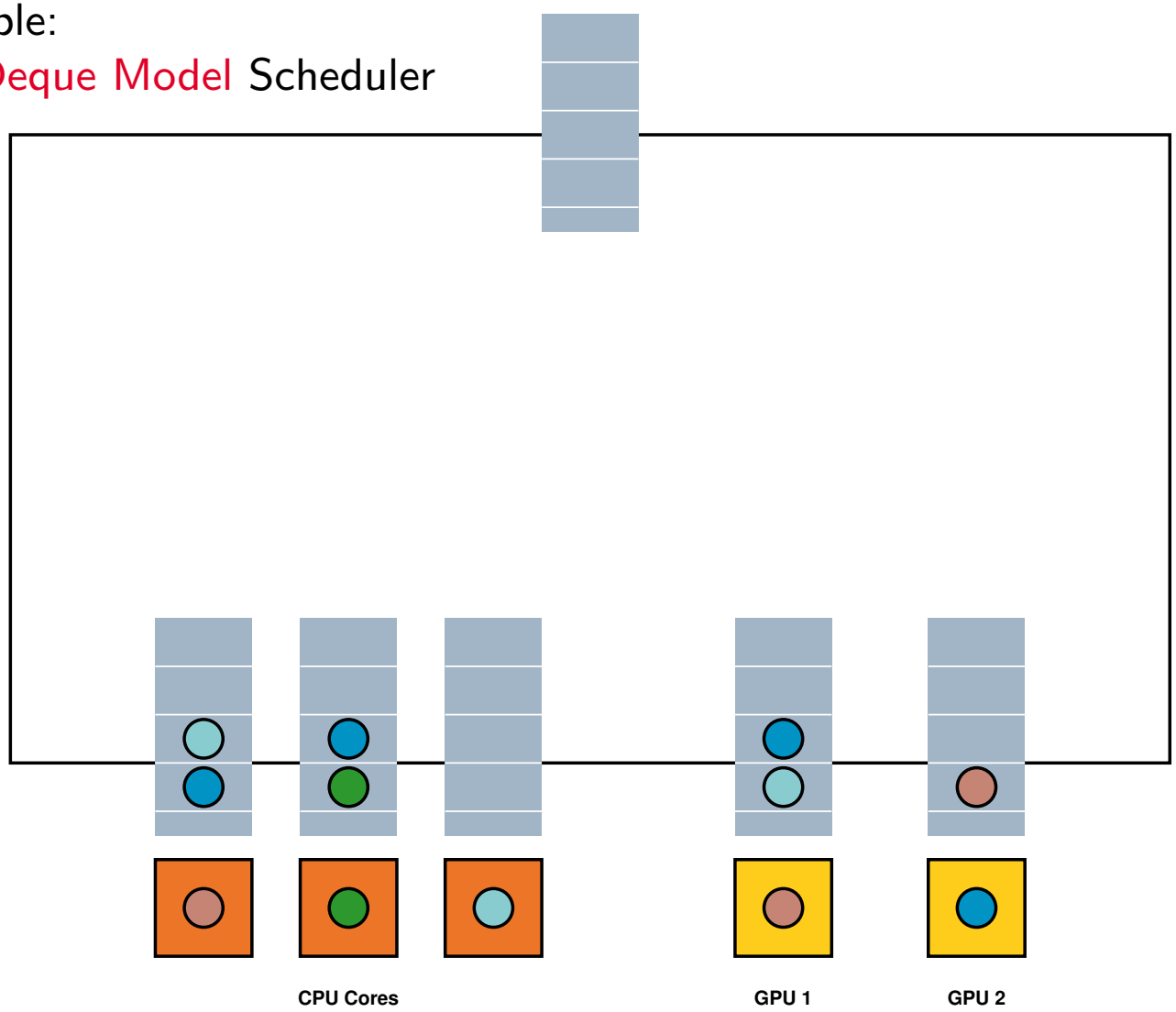
Kernel SGEQRT					
CPU:	9 GFlop/s	GPU:	30 GFlop/s	Speed-up:	3
Kernel STSQRT					
CPU:	12 GFlop/s	GPU:	37 GFlop/s	Speed-up:	3
Kernel SOMQRT					
CPU:	8.5 GFlop/s	GPU:	227 GFlop/s	Speed-up:	27
Kernel SSSMQ					
CPU:	10 GFlop/s	GPU:	285 GFlop/s	Speed-up:	28

Consequences

- Task distribution
  - SGEQRT: 20% Tasks on GPU
  - SSSMQ: 92% tasks on GPU
- Taking advantage of heterogeneity!
  - Only do what you are good for
  - Don't do what you are not good for

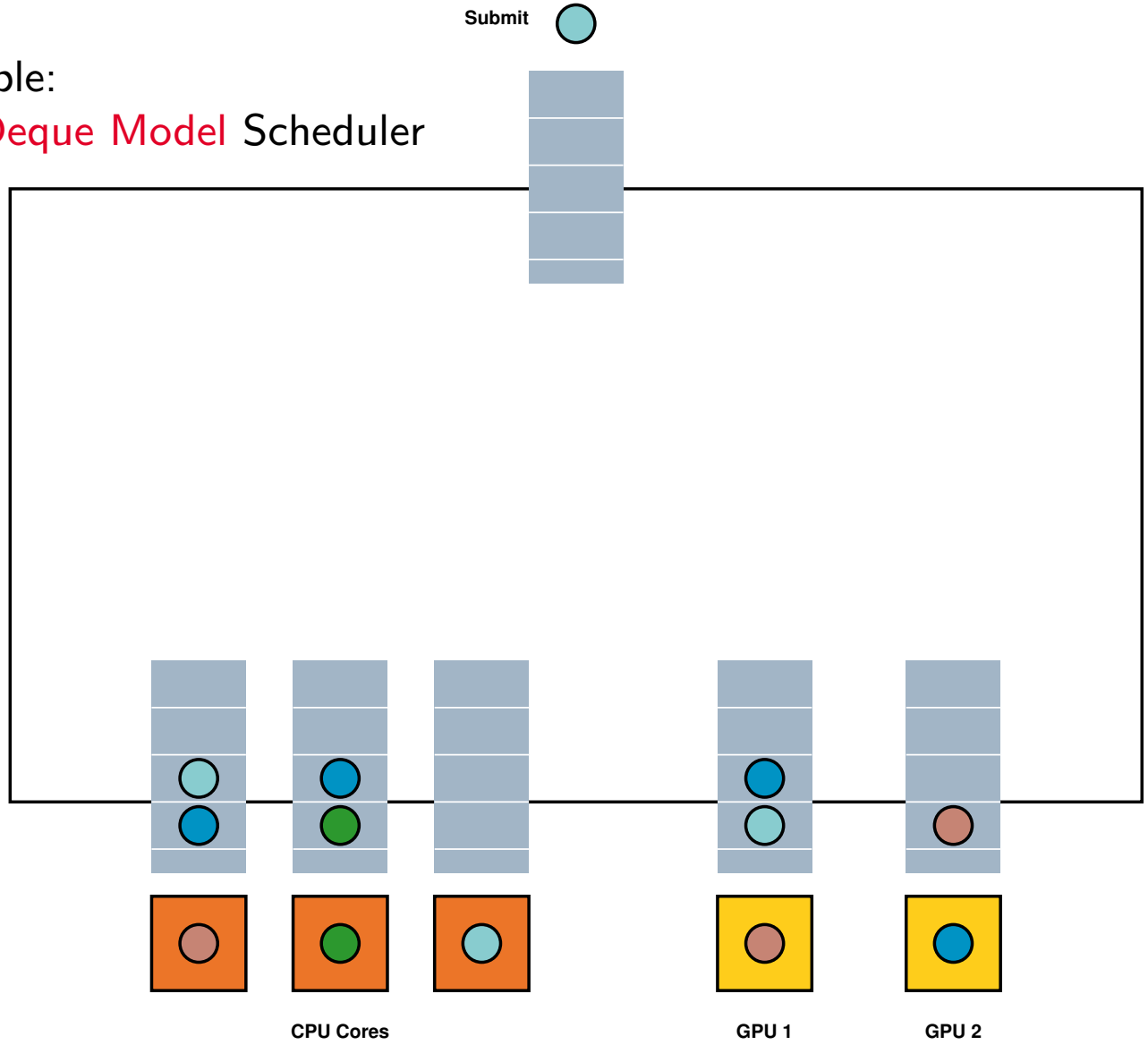
Task Mapping using Performance a Model

- Example:
  The Deque Model Scheduler



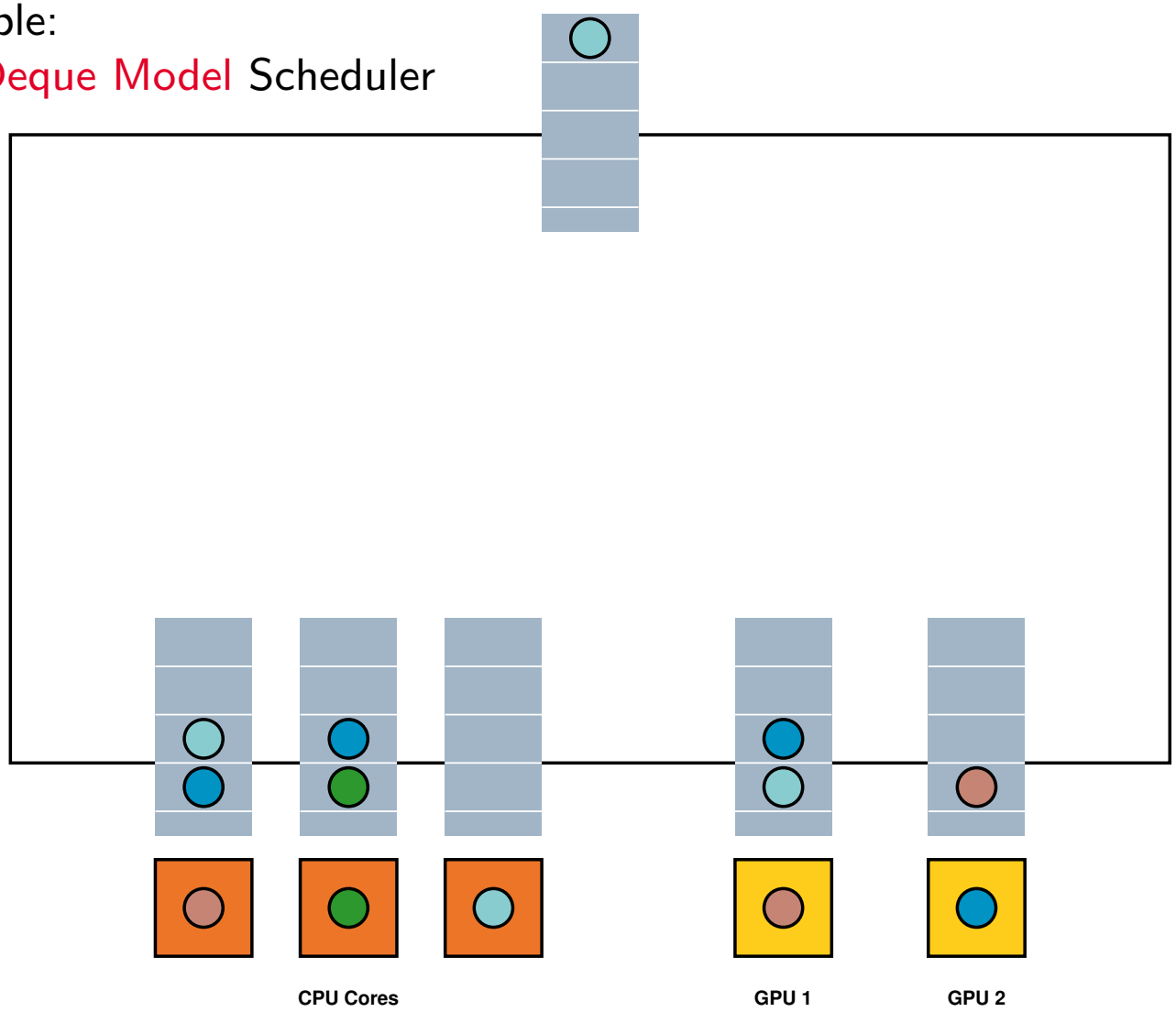
Task Mapping using Performance a Model

- Example:
  The Deque Model Scheduler



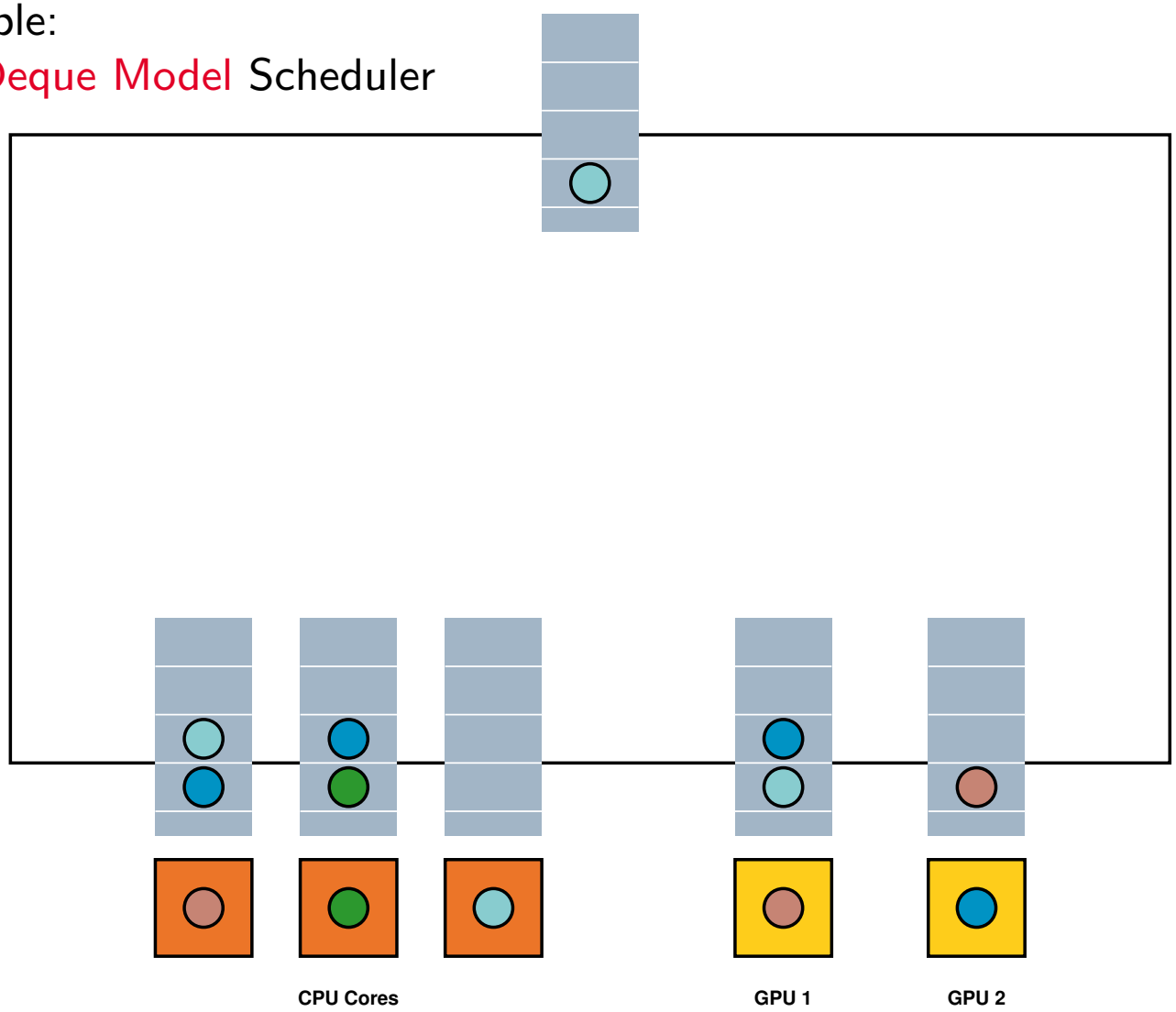
Task Mapping using Performance a Model

- Example:
  The Deque Model Scheduler



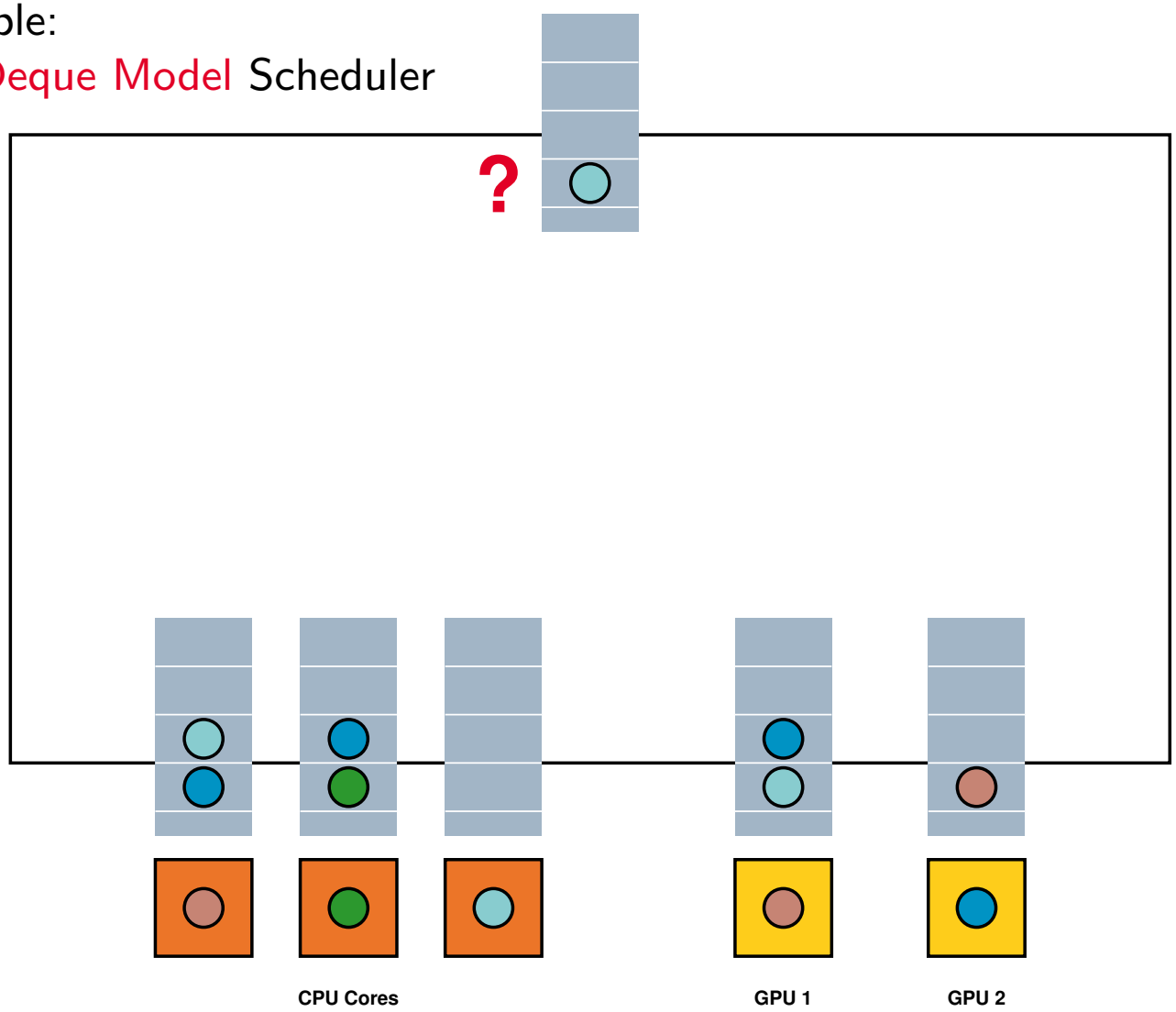
Task Mapping using Performance a Model

- Example:
  The Deque Model Scheduler



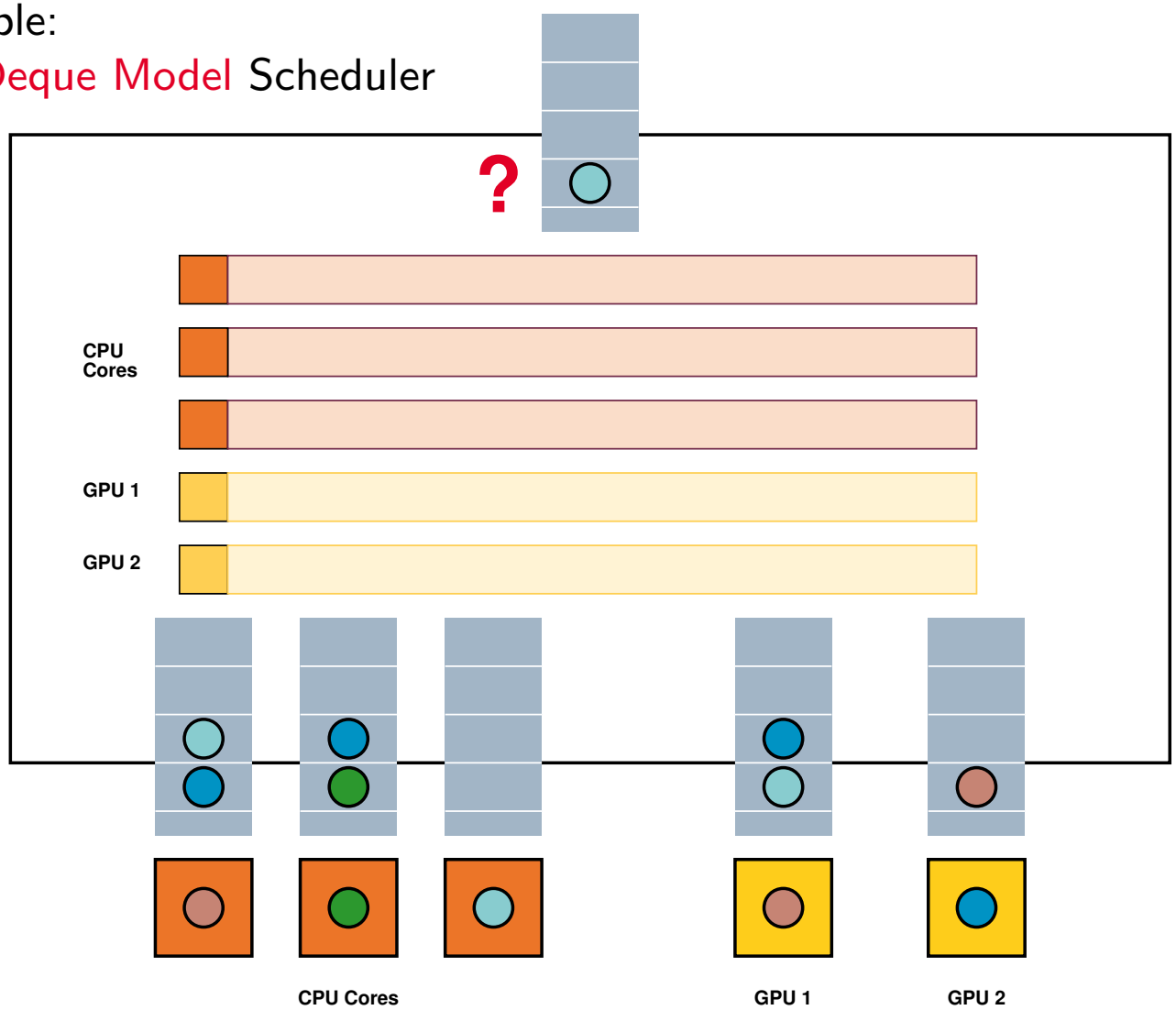
Task Mapping using Performance a Model

- Example:
  The Deque Model Scheduler



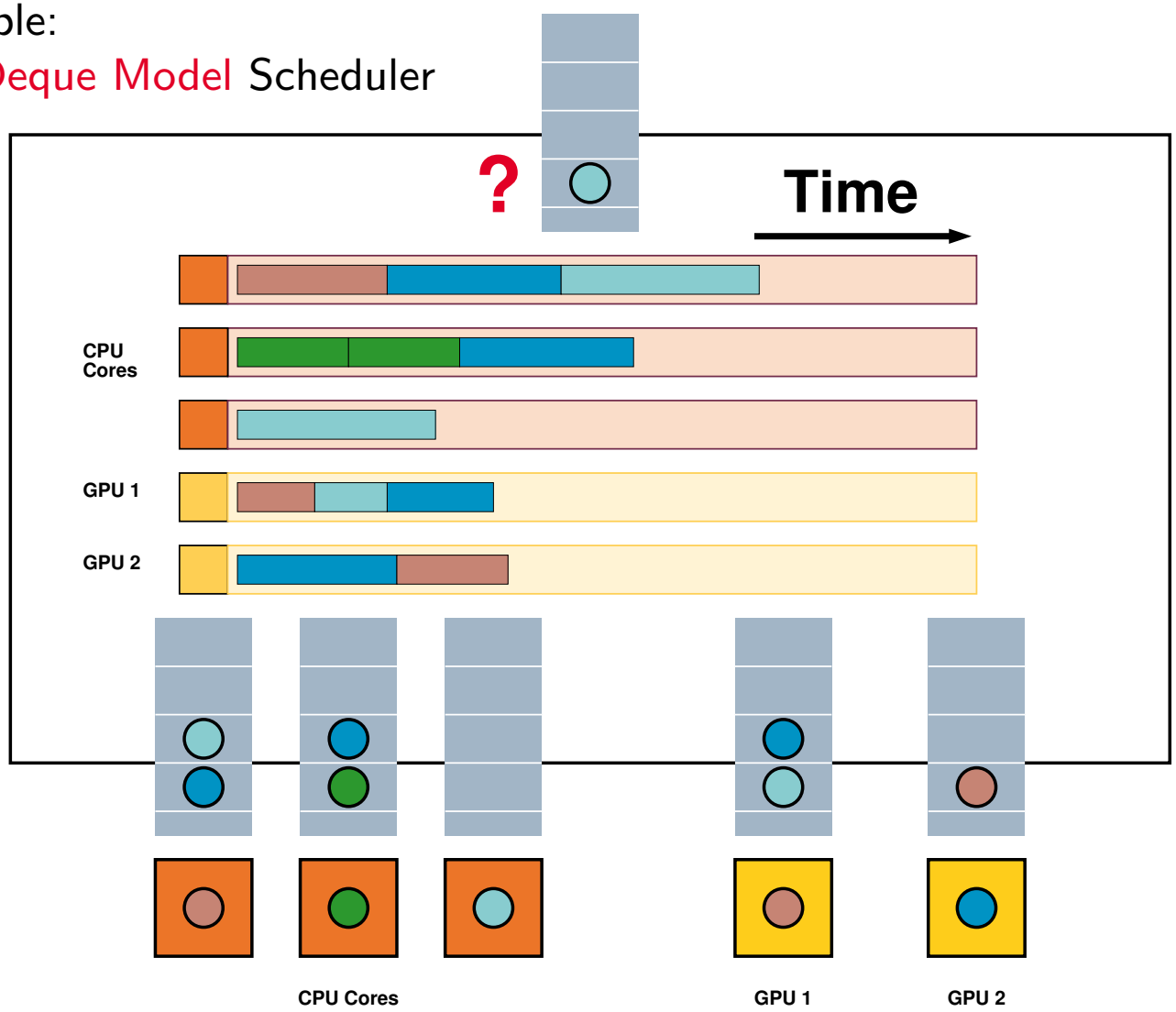
Task Mapping using Performance a Model

- Example:
  The Deque Model Scheduler



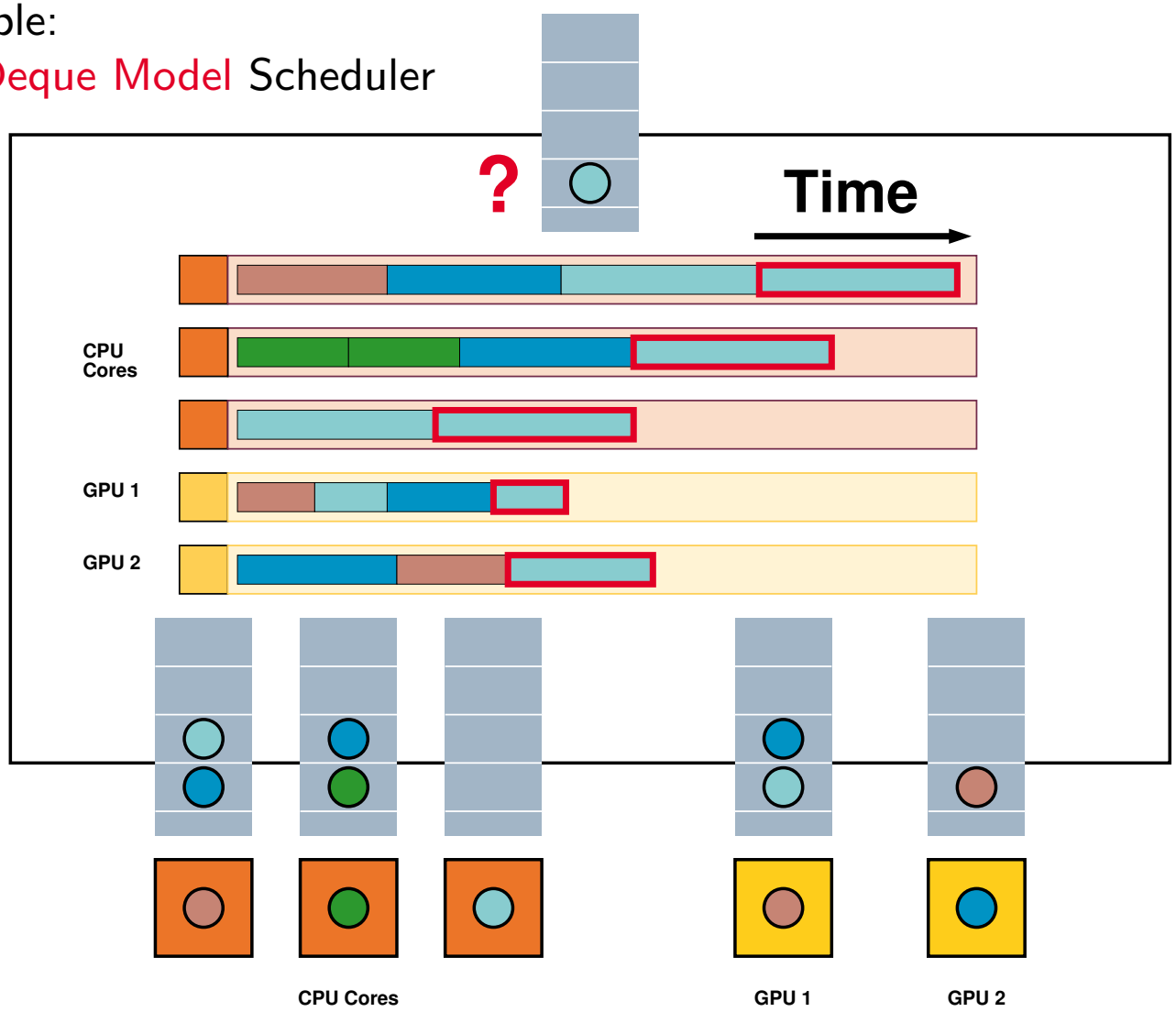
Task Mapping using Performance a Model

- Example:
  The Deque Model Scheduler



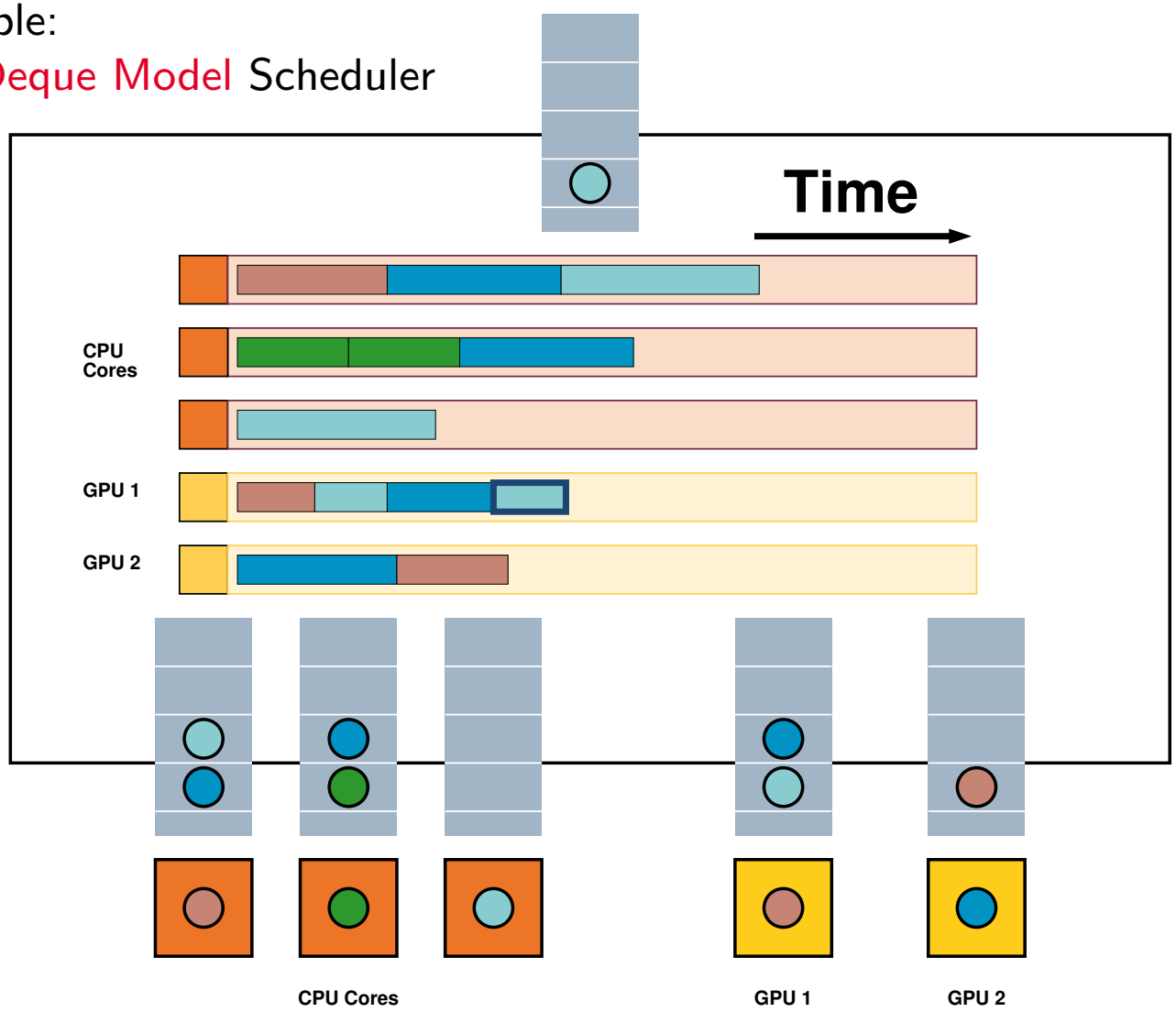
Task Mapping using Performance a Model

- Example:
  The Deque Model Scheduler



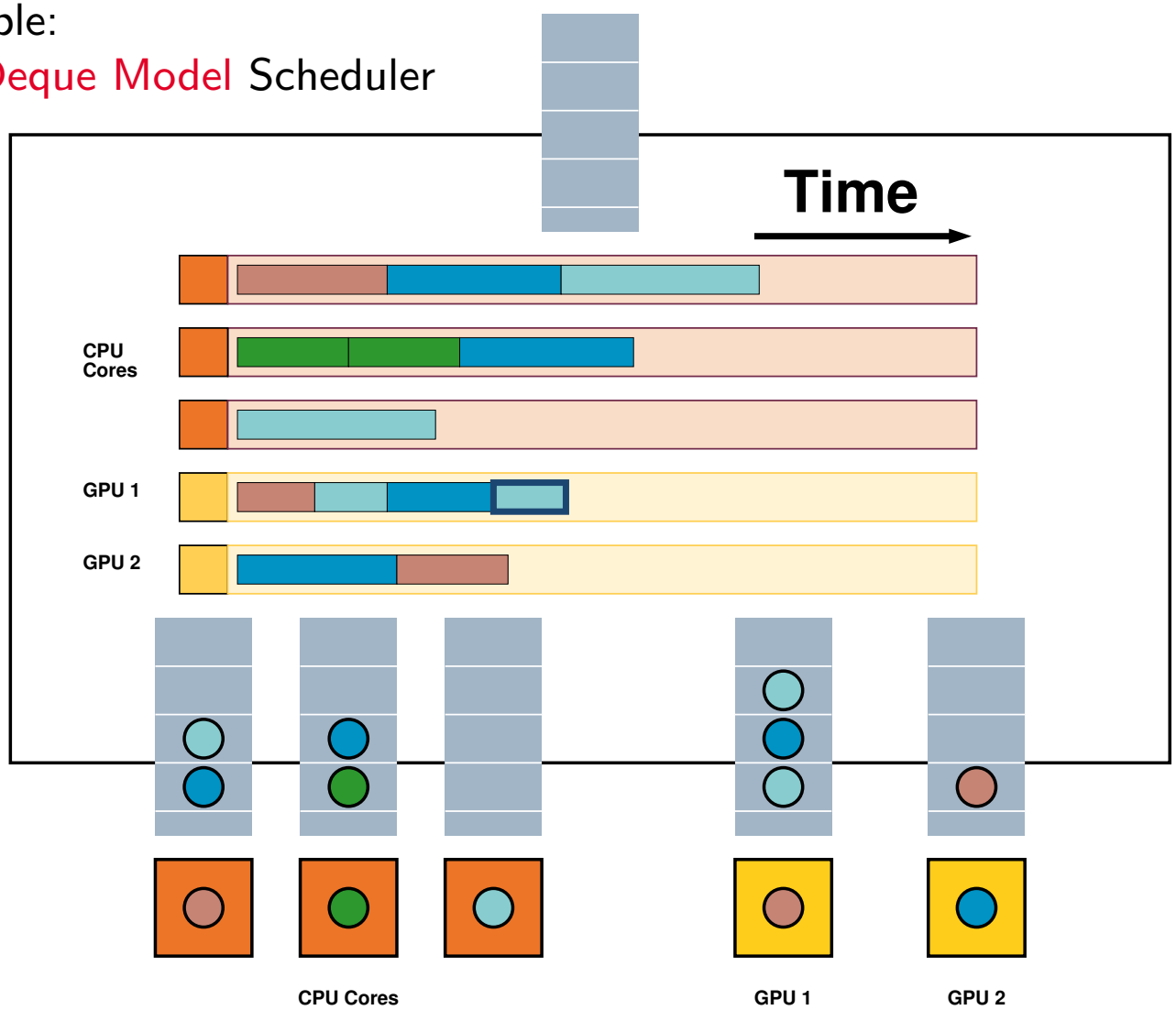
Task Mapping using Performance a Model

- Example:
  The Deque Model Scheduler



Task Mapping using Performance a Model

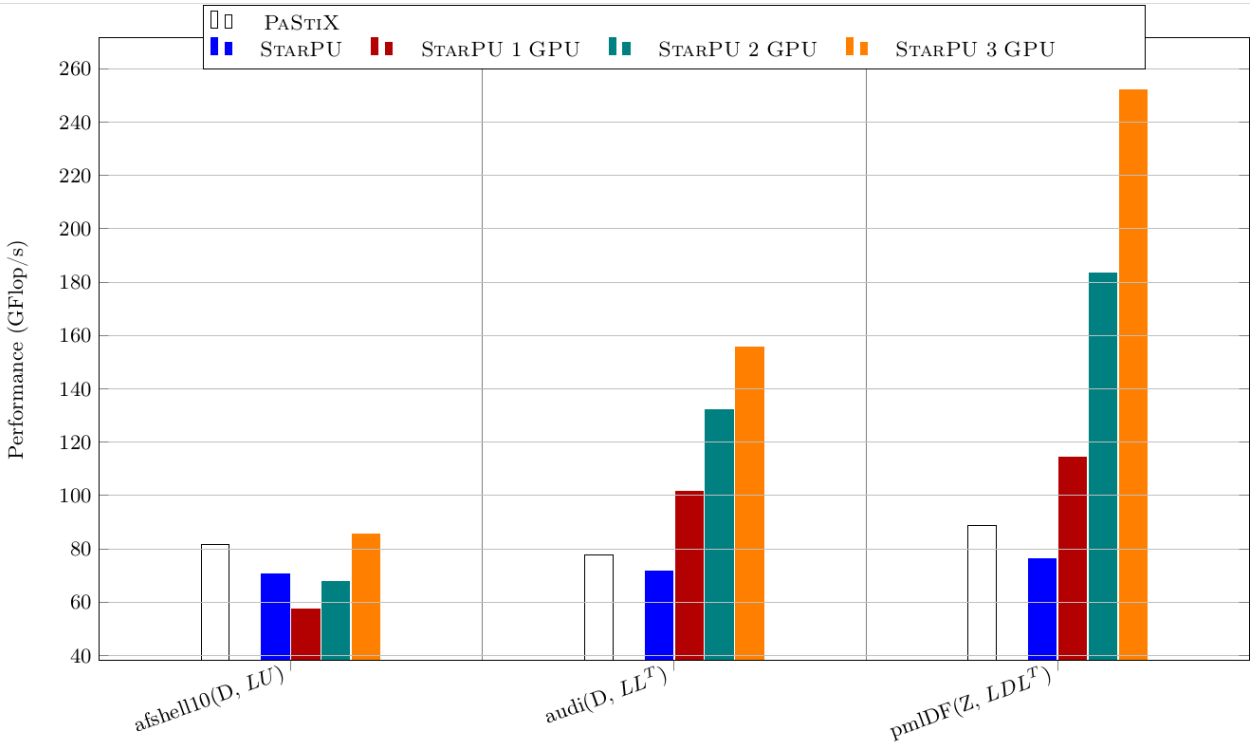
- Example:
  The Deque Model Scheduler



Example: PaSTIX Sparse Linear Algebra Solver

PaSTIX Solver (Inria HiePACS)

- Algorithm + GPU kernels
- 12 CPU cores (2 Xeon X5650)
- 3GPUs (3 Tesla M2070)



2.3

Now, *Scale* on Heterogeneous Clusters

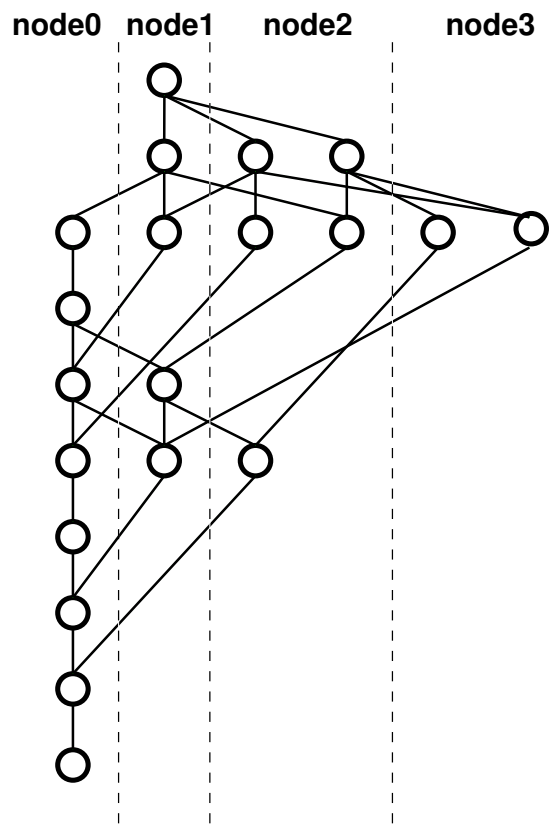
Distributed Support with StarPU

Sequential Task Flow Paradigm on Clusters

Each node unrolls the sequential task flow

Initial Data↔Node Mapping

- Provided by the application...



Extends STF Programming Model for Clusters

preserve the same code

```
for (j = 0; j < N; j++) {
  POTRF (RW, A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW, A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW, A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW, A[i][k],
        R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();
```

Extends STF Programming Model for Clusters

Almost preserve the same code

- MPI communicator

```
for (j = 0; j < N; j++) {
  POTRF (RW, A[j][j], MPI_COMM_WORLD);
  for (i = j+1; i < N; i++)
    TRSM (RW, A[i][j], R,A[j][j], MPI_COMM_WORLD);
  for (i = j+1; i < N; i++) {
    SYRK (RW, A[i][i], R,A[i][j], MPI_COMM_WORLD);
    for (k = j+1; k < i; k++)
      GEMM (RW, A[i][k],
        R,A[i][j], R,A[k][j], MPI_COMM_WORLD);
  }
}
task_wait_for_all();
```

Extends STF Programming Model for Clusters

Almost preserve the same code

- MPI communicator
- Mapping function

```
int getnode(int i, int j) { return((i%p)*q + j%q); }
```

```
for (j = 0; j < N; j++) {
  POTRF (RW, A[j][j], MPI_COMM_WORLD, getnode(j,j));
  for (i = j+1; i < N; i++)
    TRSM (RW, A[i][j], R,A[j][j], MPI_COMM_WORLD, getnode(i,j));
  for (i = j+1; i < N; i++) {
    SYRK (RW, A[i][i], R,A[i][j], MPI_COMM_WORLD, getnode(i,i));
    for (k = j+1; k < i; k++)
      GEMM (RW, A[i][k],
        R,A[i][j], R,A[k][j], MPI_COMM_WORLD, getnode(i,k));
  }
}
task_wait_for_all();
```

Extends STF Programming Model for Clusters

Almost preserve the same code

- MPI communicator
- Mapping function

```
int getnode(int i, int j) { return((i%p)*q + j%q); }
set_rank(A, getnode);

for (j = 0; j < N; j++) {
  POTRF (RW, A[j][j], MPI_COMM_WORLD);
  for (i = j+1; i < N; i++)
    TRSM (RW, A[i][j], R,A[j][j], MPI_COMM_WORLD);
  for (i = j+1; i < N; i++) {
    SYRK (RW, A[i][i], R,A[i][j], MPI_COMM_WORLD);
    for (k = j+1; k < i; k++)
      GEMM (RW, A[i][k],
        R,A[i][j], R,A[k][j], MPI_COMM_WORLD);
  }
}
task_wait_for_all();
```

Distributed Support with StarPU

Sequential Task Flow Paradigm on Clusters

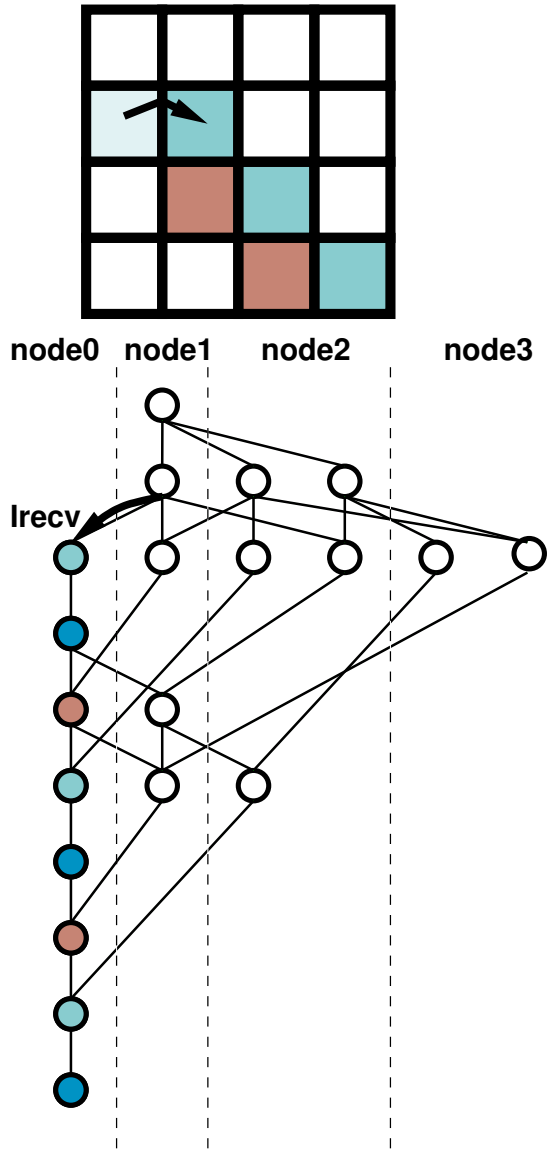
Each node unrolls the sequential task flow

Task↔Node Mapping

- Inferred from data location:
  - *Tasks move to data they modify*
- No global scheduling
- No synchronizations

Inter-node dependence management

- Inferred from the task graph edges
- Automatic **Isend** and **Irecv** calls



Distributed Support with StarPU

Sequential Task Flow Paradigm on Clusters

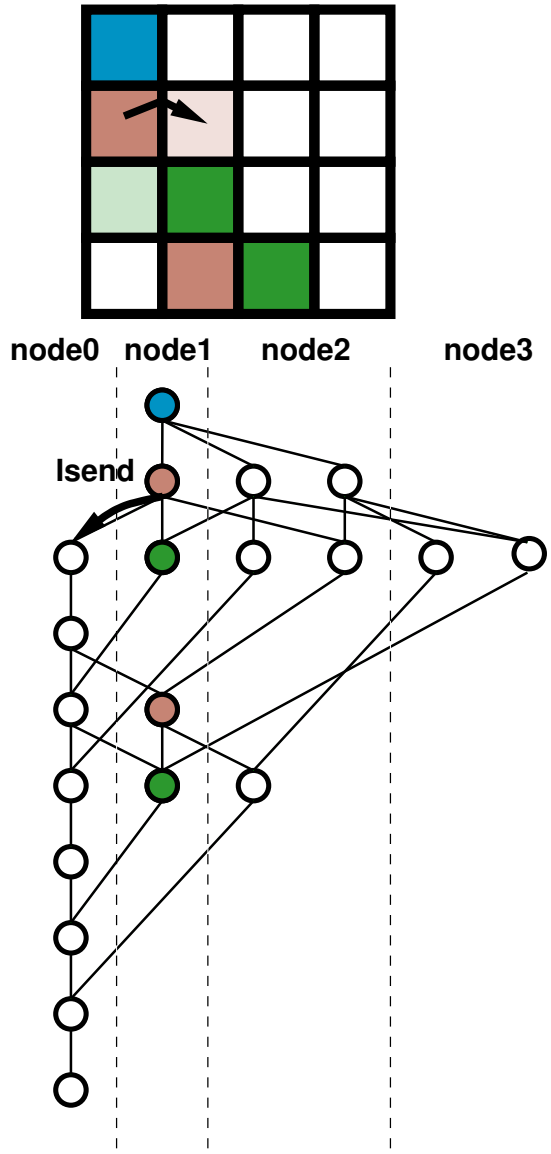
Each node unrolls the sequential task flow

Task↔Node Mapping

- Inferred from data location:
  - *Tasks move to data they modify*
- No global scheduling
- No synchronizations

Inter-node dependence management

- Inferred from the task graph edges
- Automatic **Isend** and **Irecv** calls



Distributed Scalability Study

Chameleon library (Inria HiePACS)

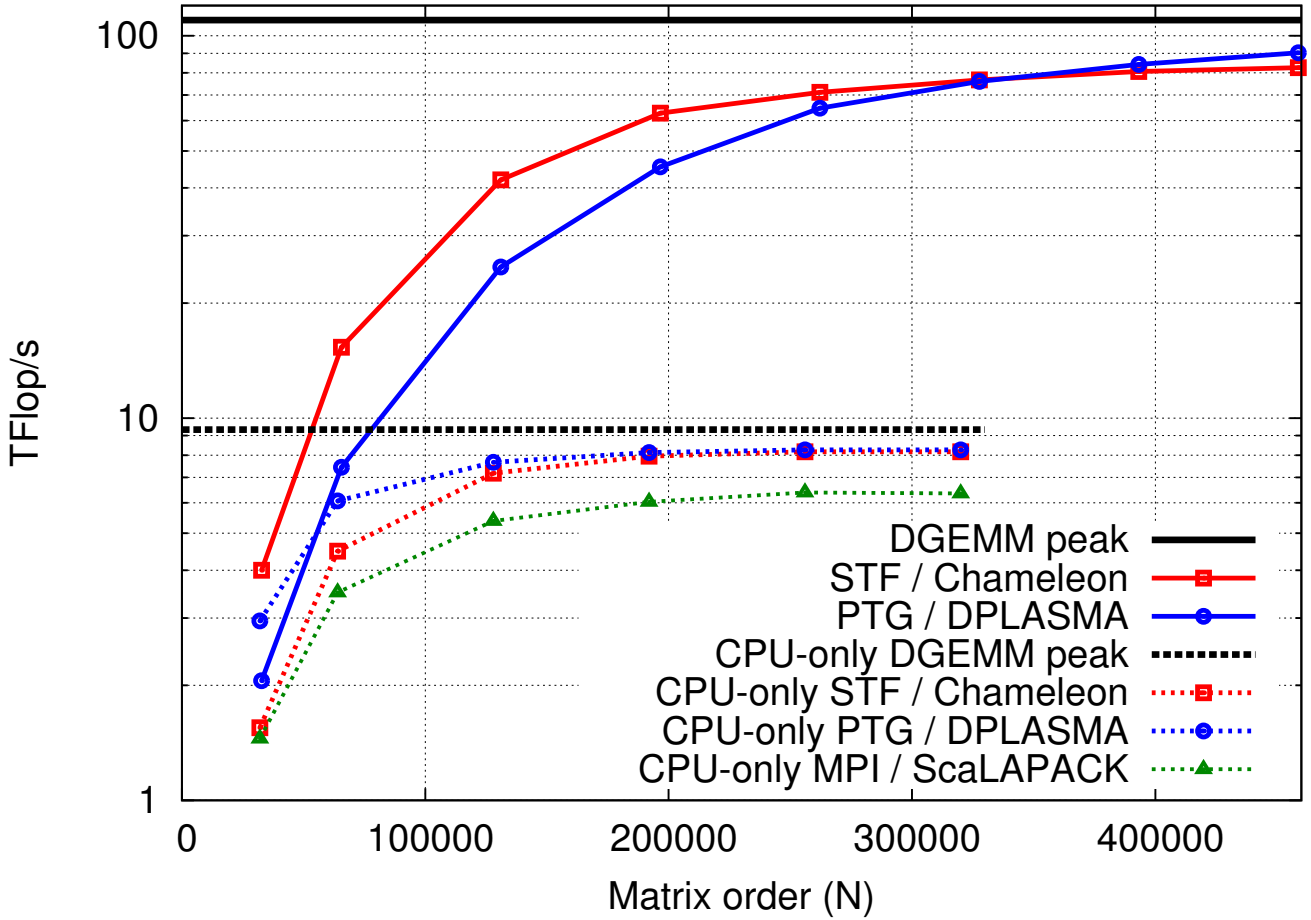
- Ph.D Marc Sergent (STORM+CEA CESTA)
- **3D electromagnetic** test case
- Complex double-precision distributed **dense Cholesky factorization**

- **Study available in Inria Research Report RR-8927**

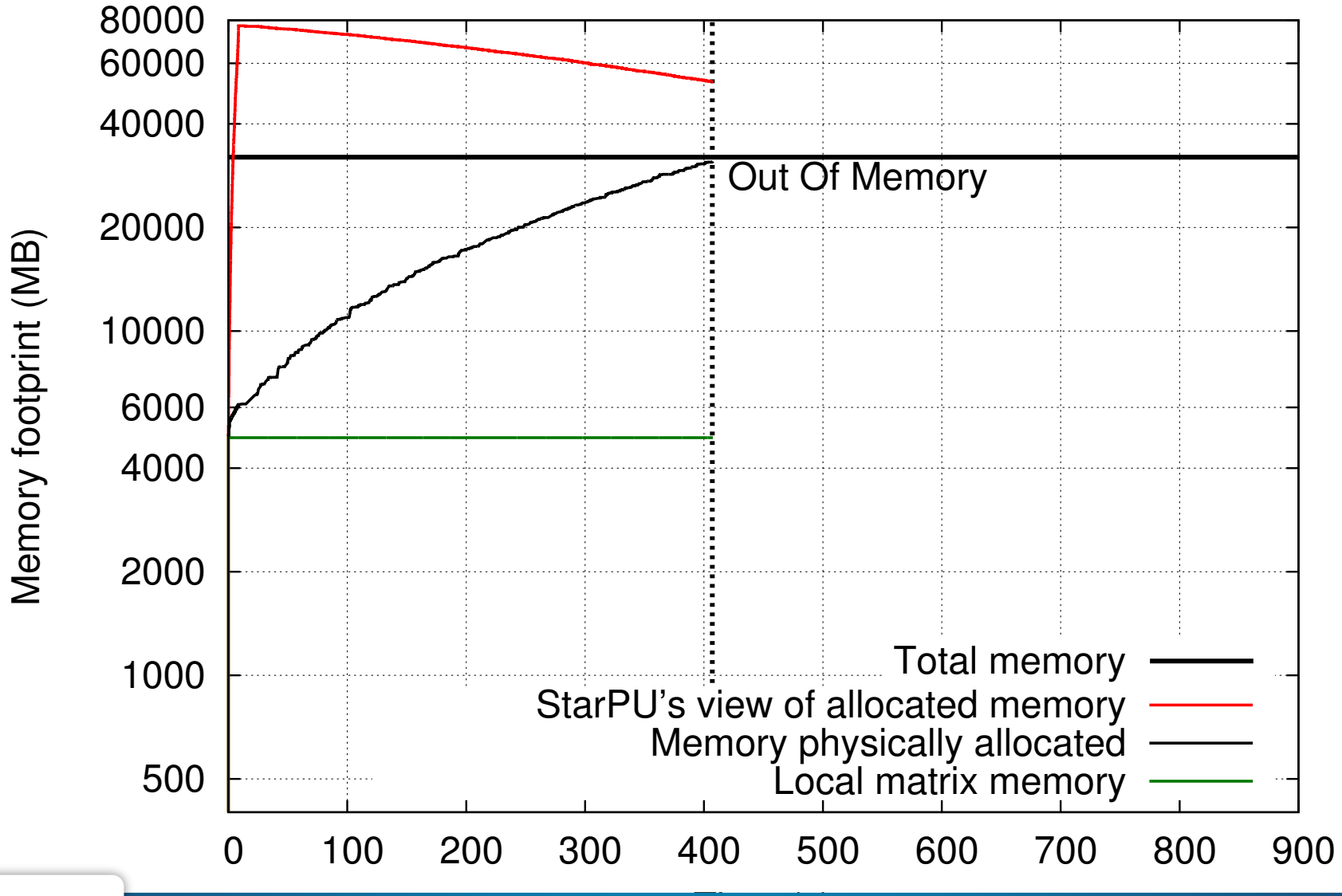
Distributed Scalability Study

Chameleon library (Inria HiePACS)

- Heterogeneous cluster: 1152 CPU cores+288 GPUs



Unbounded Task Submission Issue

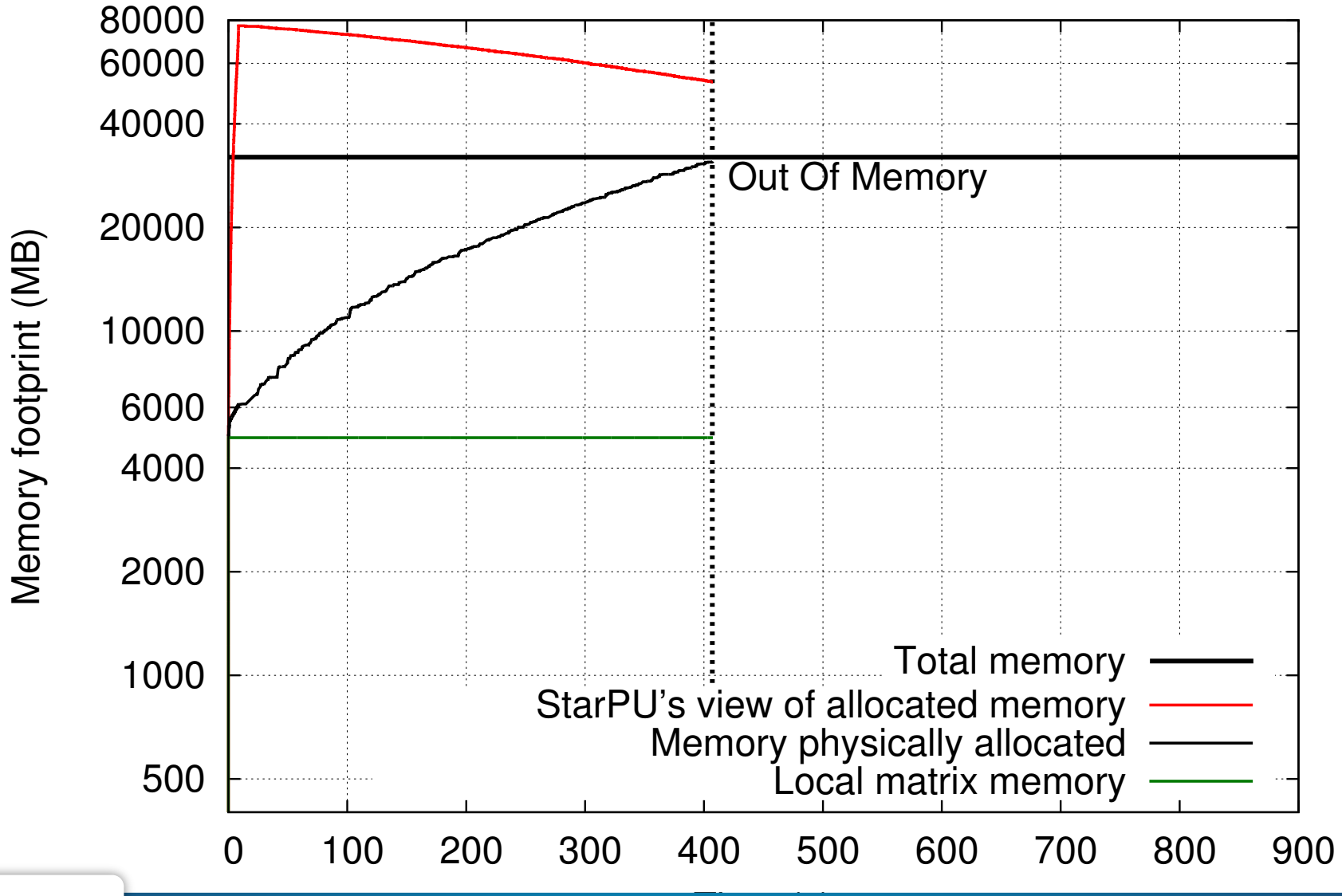


Lookahead Window on the Task Submission Side

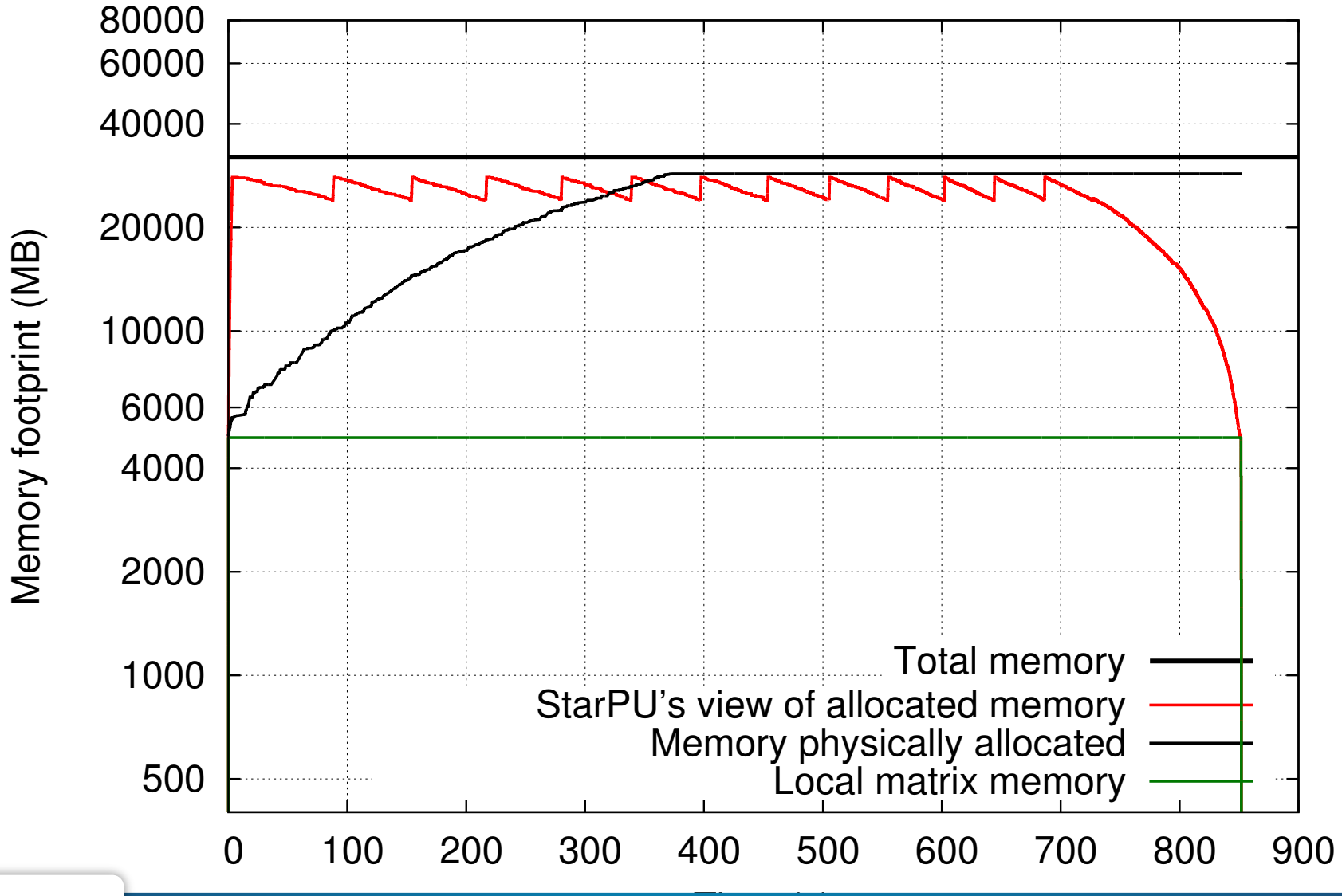
Control of the task submission flow

- **Memory tracking**
  - Account the memory subscription
- **Task submission throttling**
  - Blocking mechanism of the task submission flow
  - Allows the task submission to be controlled by an external criteria
- A **control policy** which uses the **memory tracking** to **throttle the task submission flow**

Memory Behaviour Without Memory Control



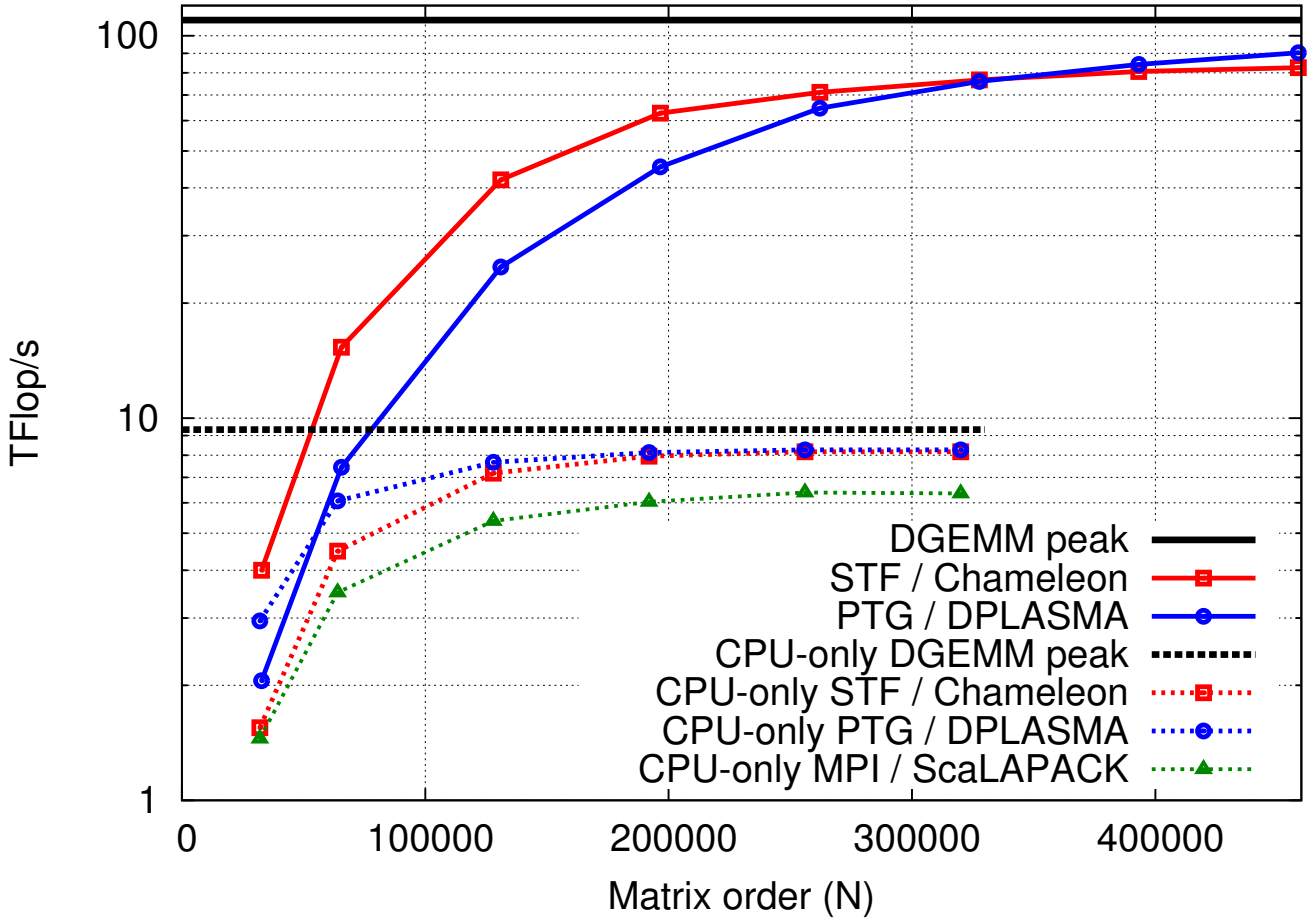
Memory Behaviour With Memory Control



Distributed Scalability Study Results

Chameleon library (Inria HiePACS): C2S@Exa Pole 1 ↔ Pole 3

- Heterogeneous cluster: 1152 CPU cores+288 GPUs



2.4

I/O and Out-of-Core Support

Input/Output Support

Integration with general StarPU's memory management layer

- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios

- Out-of-core / swap
- Mitigated startup load / solution output
- Building block for fault tolerance
  - Checkpointing

HiBOX Project (DGA RAPID)

- Imacs, Airbus

Input/Output Support

Integration with general StarPU's memory management layer

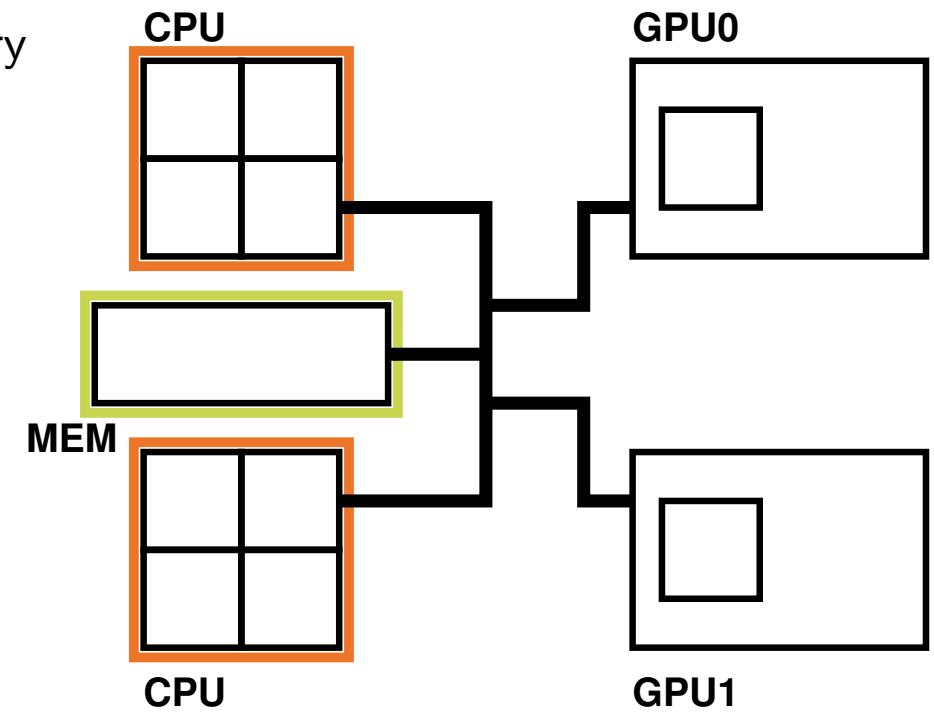
- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios

- Out-of-core / swap
- Mitigated startup load / solution output
- Building block for fault tolerance
  - Checkpointing

HiBOX Project (DGA RAPID)

- Imacs, Airbus



Input/Output Support

Integration with general StarPU's memory management layer

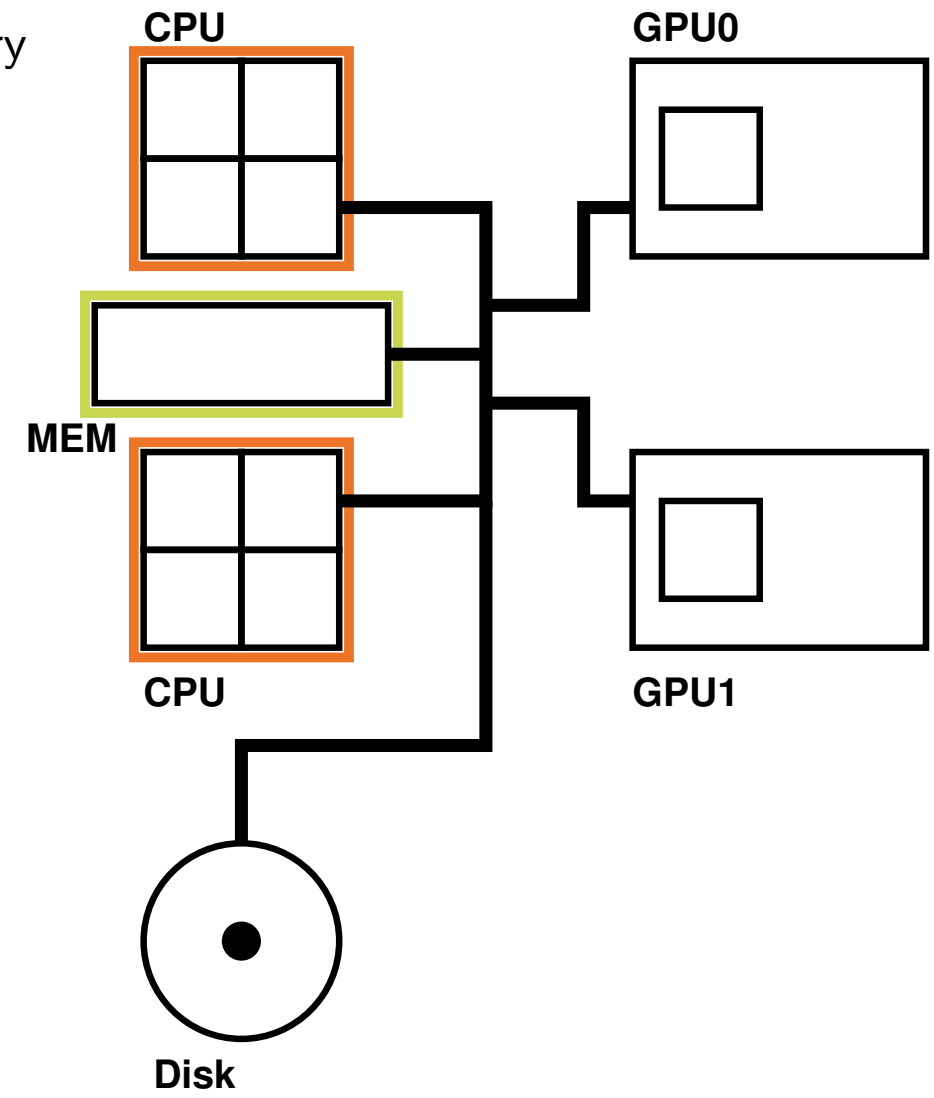
- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios

- Out-of-core / swap
- Mitigated startup load / solution output
- Building block for fault tolerance
  - Checkpointing

HiBOX Project (DGA RAPID)

- Imacs, Airbus



Input/Output Support

Integration with general StarPU's memory management layer

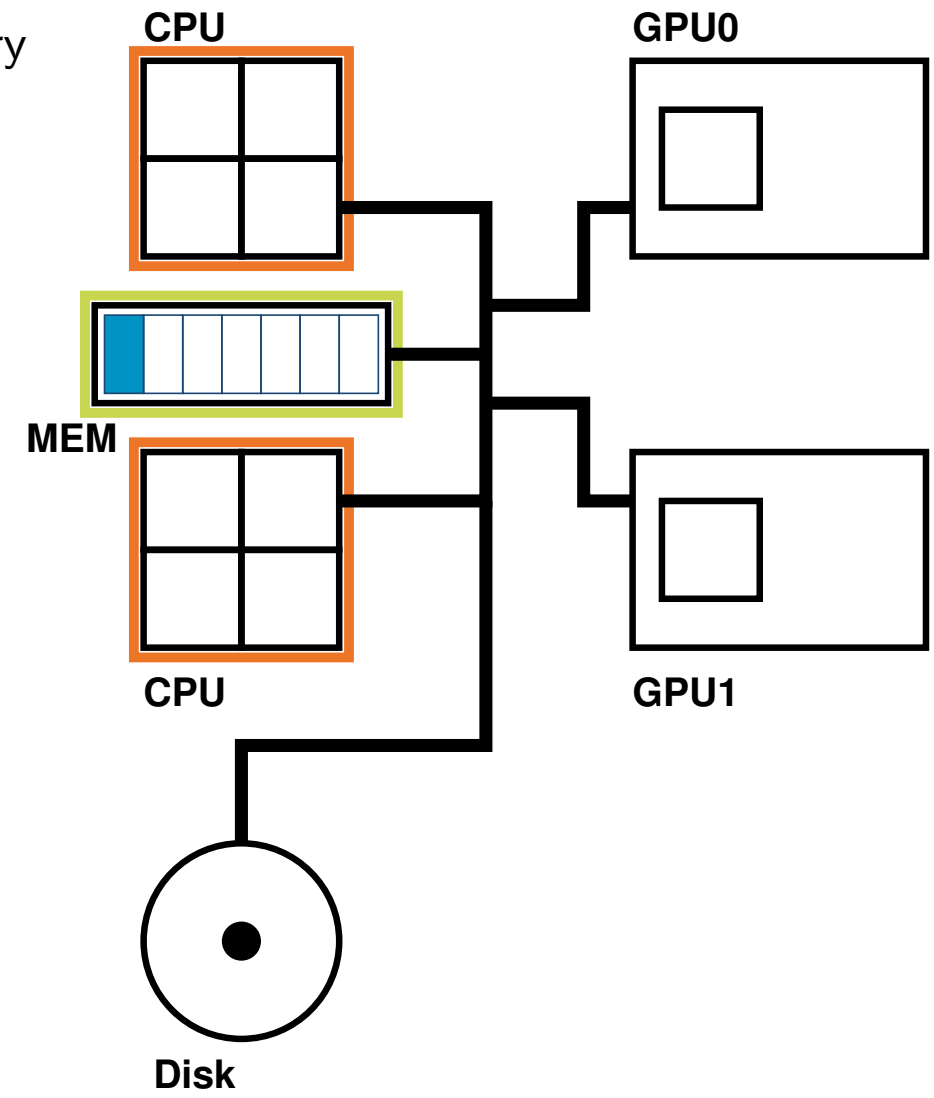
- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios

- Out-of-core / swap
- Mitigated startup load / solution output
- Building block for fault tolerance
  - Checkpointing

HiBOX Project (DGA RAPID)

- Imacs, Airbus



Input/Output Support

Integration with general StarPU's memory management layer

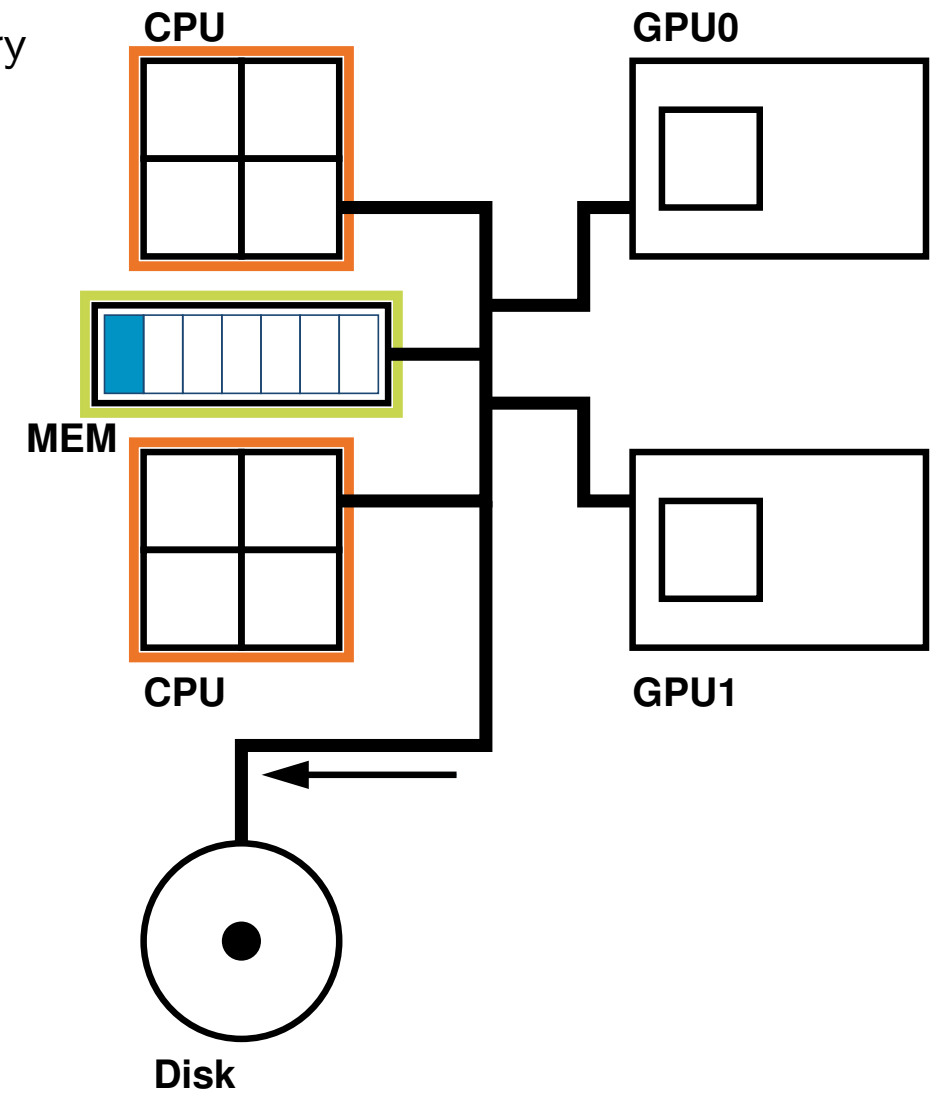
- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios

- Out-of-core / swap
- Mitigated startup load / solution output
- Building block for fault tolerance
  - Checkpointing

HiBOX Project (DGA RAPID)

- Imacs, Airbus



Input/Output Support

Integration with general StarPU's memory management layer

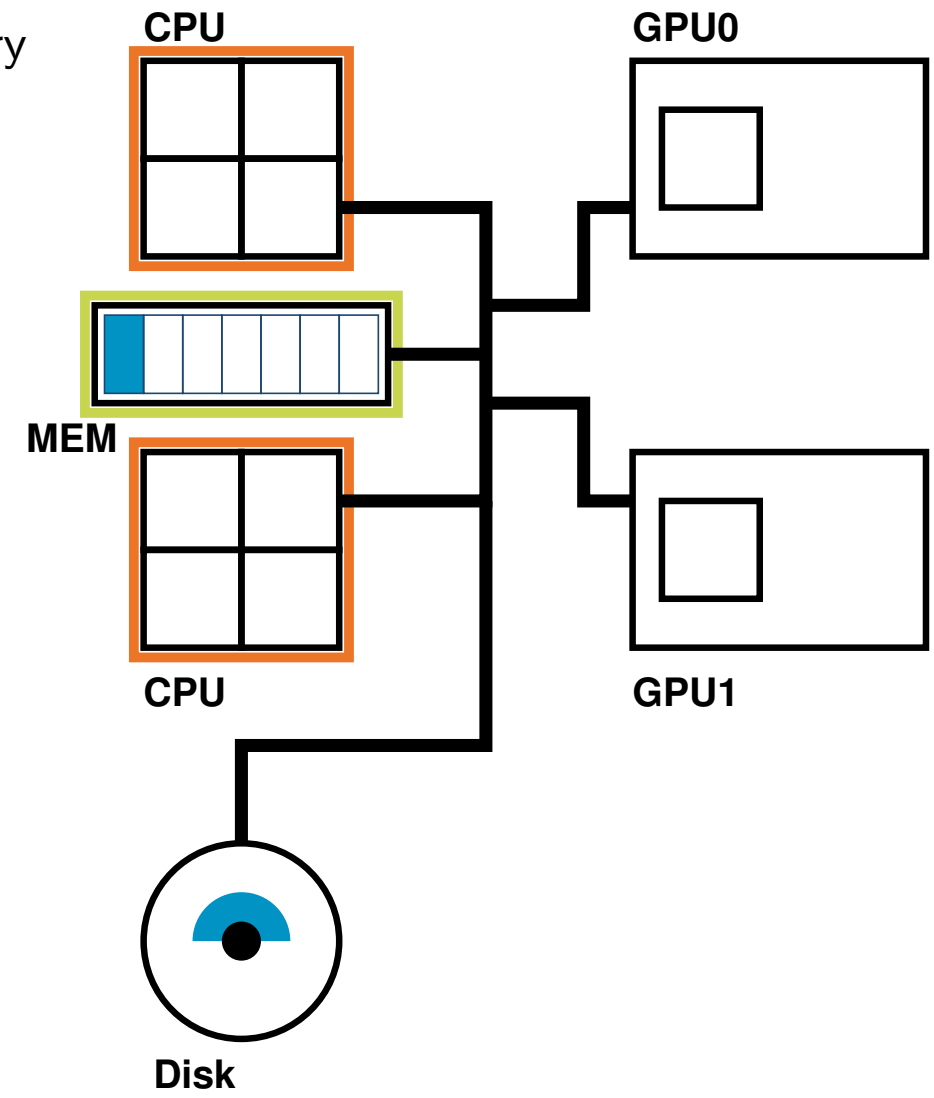
- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios

- Out-of-core / swap
- Mitigated startup load / solution output
- Building block for fault tolerance
  - Checkpointing

HiBOX Project (DGA RAPID)

- Imacs, Airbus



Input/Output Support

Integration with general StarPU's memory management layer

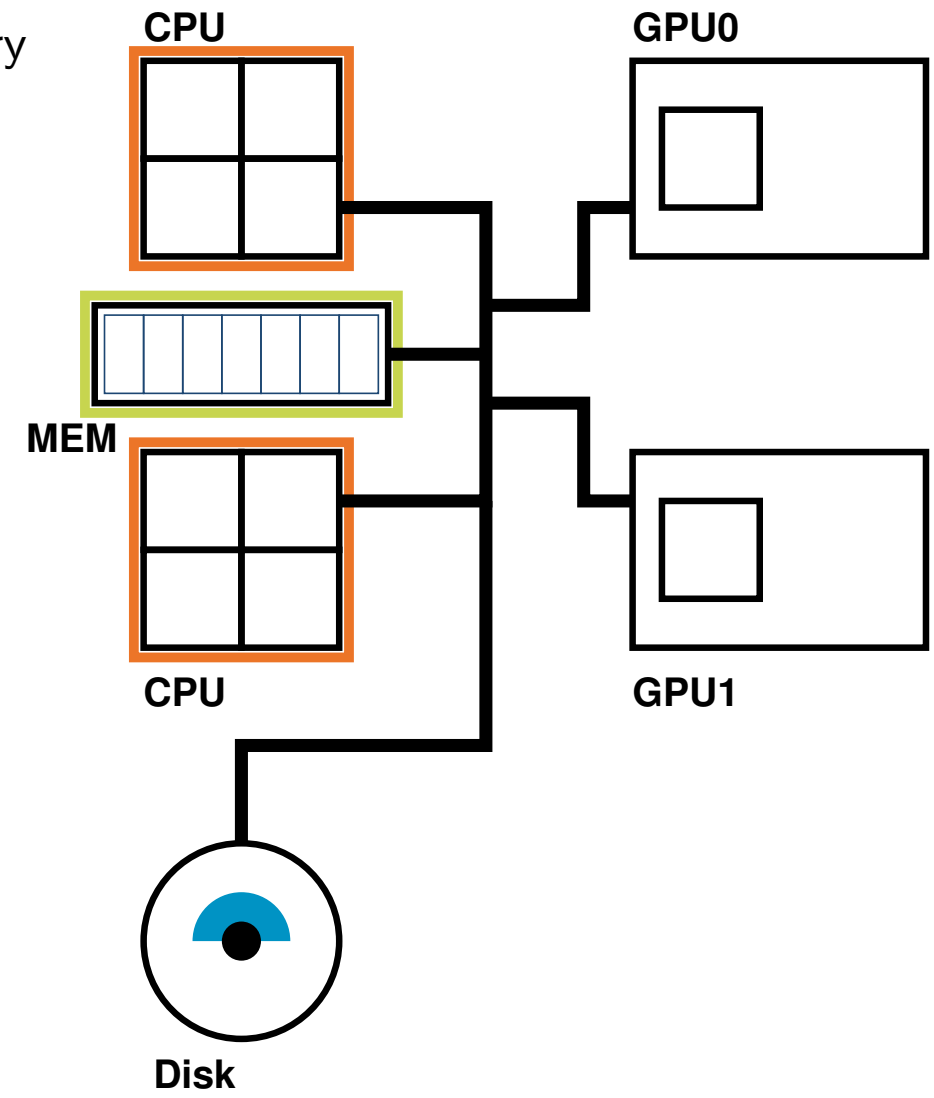
- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios

- Out-of-core / swap
- Mitigated startup load / solution output
- Building block for fault tolerance
  - Checkpointing

HiBOX Project (DGA RAPID)

- Imacs, Airbus



Input/Output Support

Integration with general StarPU's memory management layer

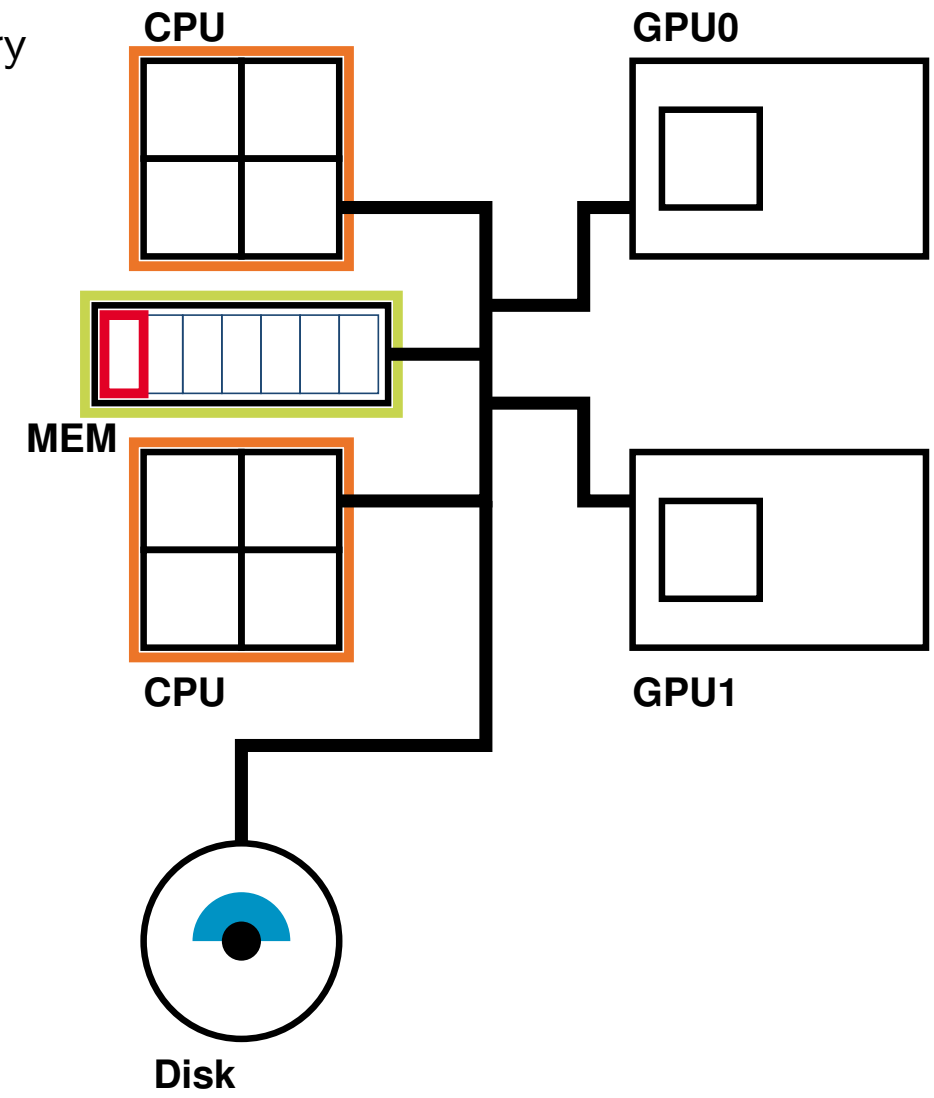
- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios

- Out-of-core / swap
- Mitigated startup load / solution output
- Building block for fault tolerance
  - Checkpointing

HiBOX Project (DGA RAPID)

- Imacs, Airbus



Input/Output Support

Integration with general StarPU's memory management layer

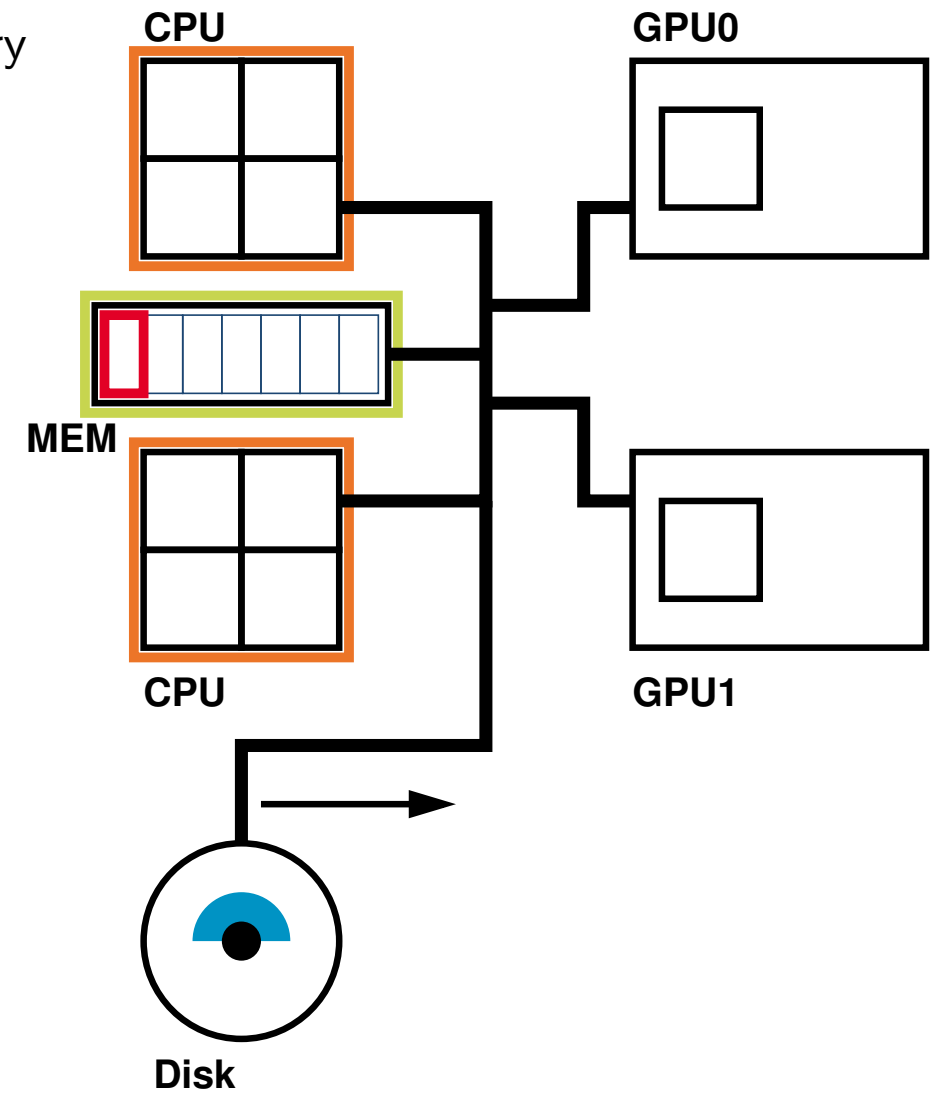
- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios

- Out-of-core / swap
- Mitigated startup load / solution output
- Building block for fault tolerance
  - Checkpointing

HiBOX Project (DGA RAPID)

- Imacs, Airbus



Input/Output Support

Integration with general StarPU's memory management layer

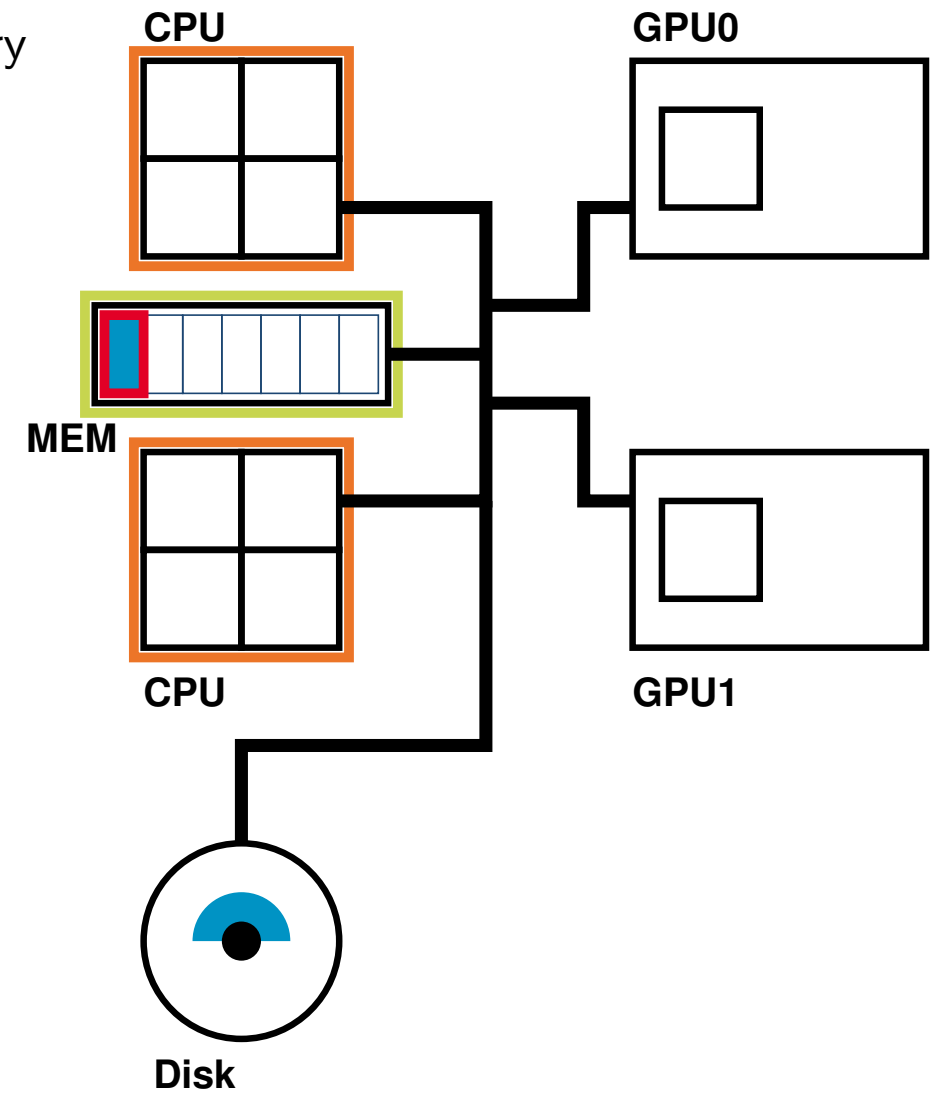
- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios

- Out-of-core / swap
- Mitigated startup load / solution output
- Building block for fault tolerance
  - Checkpointing

HiBOX Project (DGA RAPID)

- Imacs, Airbus



2.5

Programming with StarPU

Basic Example: Scaling a Vector

```
1 float factor = 3.14;
2 float vector[NX];
```

Basic Example: Scaling a Vector

```
1 float factor = 3.14;
2 float vector[NX];
3 starpu_data_handle_t vector_handle;
```

Basic Example: Scaling a Vector

```
1 float factor = 3.14;
2 float vector[NX];
3 starpu_data_handle_t vector_handle;
4
5 /* ... fill vector ... */
6
7 starpu_vector_data_register(&vector_handle, 0,
8                             (uintptr_t)vector, NX, sizeof(vector[0]))
                                ;
```

Basic Example: Scaling a Vector

```
1  float factor = 3.14;
2  float vector[NX];
3  starpu_data_handle_t vector_handle;
4
5  /* ... fill vector ... */
6
7  starpu_vector_data_register(&vector_handle, 0,
8                              (uintptr_t)vector, NX, sizeof(vector[0]))
                                   ;
9
10 starpu_task_insert(
11                     &scal_cl,
12                     STARPU_RW, vector_handle,
13                     STARPU_VALUE, &factor, sizeof(factor),
14                     0);
```

Basic Example: Scaling a Vector

```
1  float factor = 3.14;
2  float vector[NX];
3  starpu_data_handle_t vector_handle;
4
5  /* ... fill vector ... */
6
7  starpu_vector_data_register(&vector_handle, 0,
8                              (uintptr_t)vector, NX, sizeof(vector[0]))
                                   ;
9
10 starpu_task_insert(
11                    &scal_cl,
12                    STARPU_RW, vector_handle,
13                    STARPU_VALUE, &factor, sizeof(factor),
14                    0);
15
16 starpu_task_wait_for_all();
```

Basic Example: Scaling a Vector

```
1  float factor = 3.14;
2  float vector[NX];
3  starpu_data_handle_t vector_handle;
4
5  /* ... fill vector ... */
6
7  starpu_vector_data_register(&vector_handle, 0,
8                              (uintptr_t)vector, NX, sizeof(vector[0]))
                                   ;
9
10 starpu_task_insert(
11                    &scal_cl,
12                    STARPU_RW, vector_handle,
13                    STARPU_VALUE, &factor, sizeof(factor),
14                    0);
15
16 starpu_task_wait_for_all();
17 starpu_data_unregister(vector_handle);
18
19 /* ... display vector ... */
```

Terminology

- Codelet
- Task
- Data handle

Definition: A Codelet

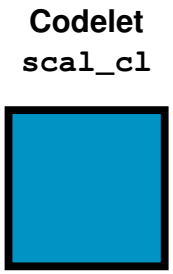
A **Codelet**...

- ... relates an abstract computation kernel to its implementation(s)
- ... can be instantiated into one or more **tasks**
- ... defines characteristics common to a set of **task**s

Definition: A Codelet

A **Codelet**...

- ... relates an abstract computation kernel to its implementation(s)
- ... can be instantiated into one or more **tasks**
- ... defines characteristics common to a set of **task**s



Definition: A Codelet

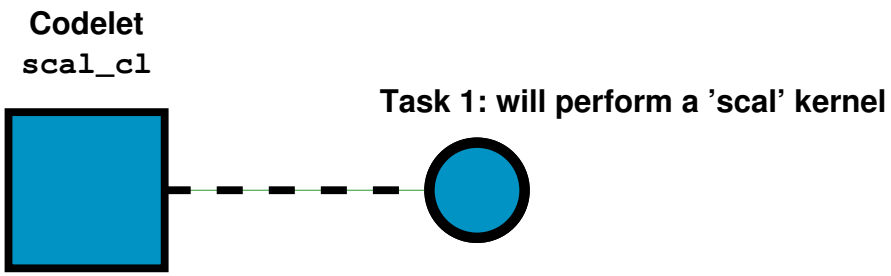
A **Codelet**...

- ... relates an abstract computation kernel to its implementation(s)
- ... can be instantiated into one or more **tasks**
- ... defines characteristics common to a set of **task**s

Definition: A Codelet

A **Codelet**...

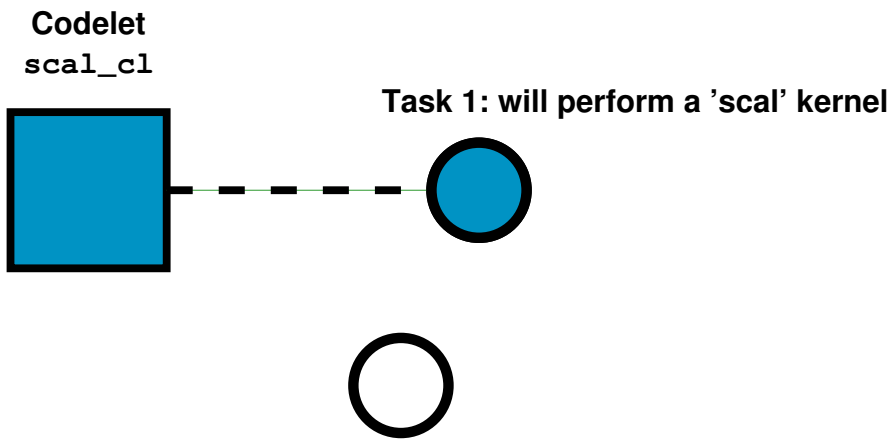
- ... relates an abstract computation kernel to its implementation(s)
- ... can be instantiated into one or more **tasks**
- ... defines characteristics common to a set of **task**s



Definition: A Codelet

A **Codelet**...

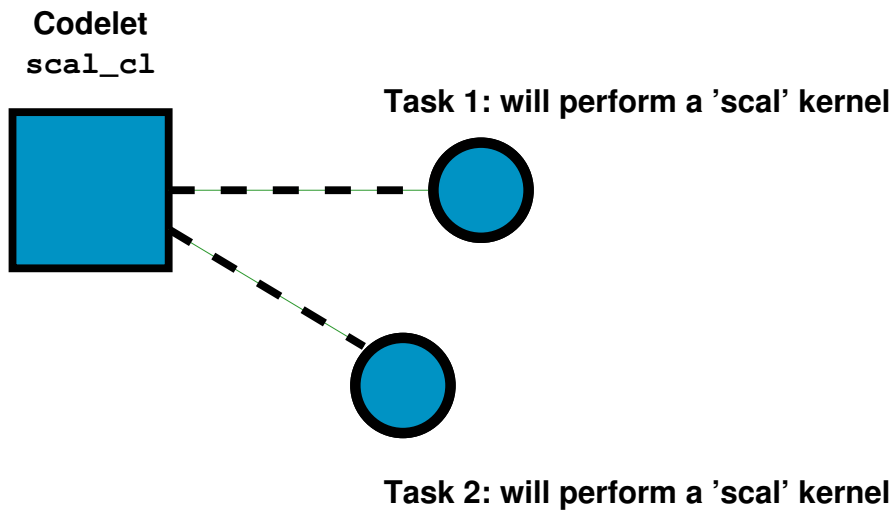
- ... relates an abstract computation kernel to its implementation(s)
- ... can be instantiated into one or more **tasks**
- ... defines characteristics common to a set of **task**s



Definition: A Codelet

A **Codelet**...

- ... relates an abstract computation kernel to its implementation(s)
- ... can be instantiated into one or more **tasks**
- ... defines characteristics common to a set of **task**s



Definition: A Task

A **Task**...

- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output

Definition: A Task

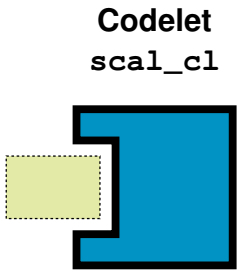
A **Task**...

- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output

Definition: A Task

A **Task**...

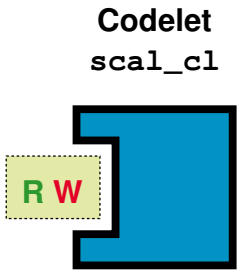
- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output



Definition: A Task

A **Task**...

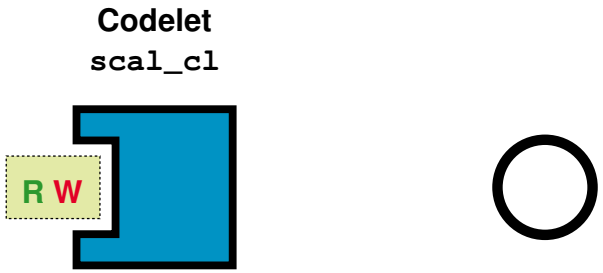
- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output



Definition: A Task

A **Task**...

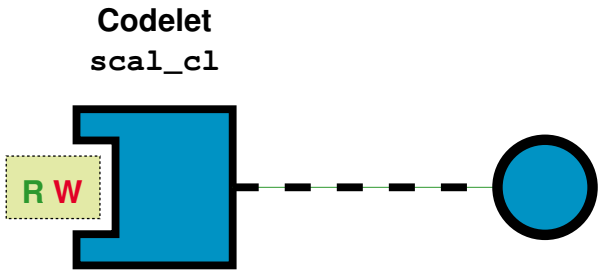
- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output



Definition: A Task

A **Task**...

- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output



Definition: A Task

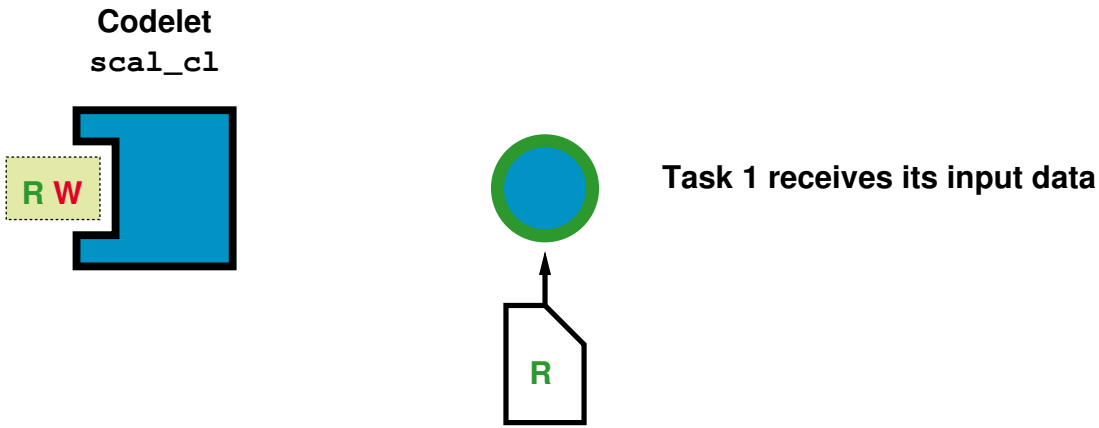
A **Task**...

- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output

Definition: A Task

A **Task**...

- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output



Definition: A Task

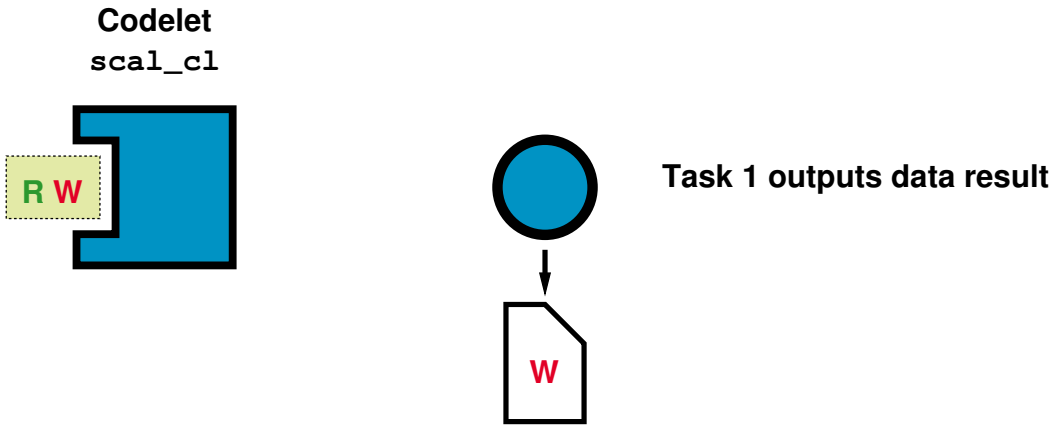
A **Task**...

- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output

Definition: A Task

A **Task**...

- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output



Definition: A Data Handle

A **Data Handle**...

- ... designates a piece of data managed by StarPU
- ... is typed (vector, matrix, etc.)
- ... can be passed as input/output for a **Task**

Elementary API

- Declaring a codelet
- Declaring and Managing Data
- Writing a Kernel Function
- Submitting a task
- Waiting for submitted tasks

Declaring a Codelet

Define a **struct** starpu_codelet

```
1 struct starpu_codelet scal_cl = {
2       ...
3 };
```

Declaring a Codelet

Define a **struct** **starpu_codelet**

- Plug the kernel function
  - Here: scal_cpu_func

```
1 struct starpu_codelet scal_cl = {
2       .cpu_func  = { scal_cpu_func, NULL },
3       ...
4 };
```

Declaring a Codelet

Define a **struct** **starpu_codelet**

- Plug the kernel function
  - Here: scal_cpu_func
- Declare the number of data pieces used by the kernel
  - Here: A single vector

```
1 struct starpu_codelet scal_cl = {
2       .cpu_func  = { scal_cpu_func, NULL },
3       .nbuffers = 1,
4       ...
5 };
```

Declaring a Codelet

Define a **struct** starpu_codelet

- Plug the kernel function
  - Here: scal_cpu_func
- Declare the number of data pieces used by the kernel
  - Here: A single vector
- Declare how the kernel accesses the piece of data
  - Here: The vector is scaled in-place, thus R/W

```
1 struct starpu_codelet scal_cl = {
2       .cpu_func  = { scal_cpu_func, NULL },
3       .nbuffers = 1,
4       .modes = { STARPU_RW },
5 };
```

Declaring and Managing Data

Put data under StarPU control

Declaring and Managing Data

Put data under StarPU control

- Initialize a piece of data

```
1 float vector[NX];
2 /* ... fill data ... */
```

Declaring and Managing Data

Put data under StarPU control

- Initialize a piece of data
- Register the piece of data and get a handle
  - The vector is now under StarPU control

```
1 float vector[NX];
2 /* ... fill data ... */
3
4 starpu_data_handle_t vector_handle;
5 starpu_vector_data_register(&vector_handle, 0,
6                             (uintptr_t)vector, NX, sizeof(vector[0]));
```

Declaring and Managing Data

Put data under StarPU control

- Initialize a piece of data
- Register the piece of data and get a handle
  - The vector is now under StarPU control
- Use data through the handle

```
1 float vector[NX];
2 /* ... fill data ... */
3
4 starpu_data_handle_t vector_handle;
5 starpu_vector_data_register(&vector_handle, 0,
6                             (uintptr_t)vector, NX, sizeof(vector[0]));
7
8 /* ... use the vector through the handle ... */
```

Declaring and Managing Data

Put data under StarPU control

- Initialize a piece of data
- Register the piece of data and get a handle
  - The vector is now under StarPU control
- Use data through the handle
- Unregister the piece of data
  - The handle is destroyed
  - The vector is now back under user control

```
1  float vector[NX];
2  /* ... fill data ... */
3
4  starpu_data_handle_t vector_handle;
5  starpu_vector_data_register(&vector_handle, 0,
6                              (uintptr_t)vector, NX, sizeof(vector[0]));
7
8  /* ... use the vector through the handle ... */
9
10 starpu_data_unregister(vector_handle);
```

Writing a Kernel Function

- Every kernel function has the same C prototype

```
1 void scal_cpu_func(void *buffers[], void *cl_arg) {
2     ...
3 }
```

Writing a Kernel Function

- Every kernel function has the same C prototype
- Retrieve the vector's handle

```
1 void scal_cpu_func(void *buffers[], void *cl_arg) {
2     struct starpu_vector_interface *vector_handle = buffers
          [0];
3
4     ...
5 }
```

Writing a Kernel Function

- Every kernel function has the same C prototype
- Retrieve the vector's handle
- Get vector's number of elements and base pointer

```
1 void scal_cpu_func(void *buffers[], void *cl_arg) {
2     struct starpu_vector_interface *vector_handle = buffers
          [0];
3
4     unsigned n    = STARPU_VECTOR_GET_NX(vector_handle);
5     float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
6
7     ...
8 }
```

Writing a Kernel Function

- Every kernel function has the same C prototype
- Retrieve the vector's handle
- Get vector's number of elements and base pointer
- Get the scaling factor as inline argument

```
1  void scal_cpu_func(void *buffers[], void *cl_arg) {
2      struct starpu_vector_interface *vector_handle = buffers
           [0];
3
4      unsigned n    = STARPU_VECTOR_GET_NX(vector_handle);
5      float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
6
7      float *ptr_factor = cl_arg;
8
9      ...
10 }
```

Writing a Kernel Function

- Every kernel function has the same C prototype
- Retrieve the vector's handle
- Get vector's number of elements and base pointer
- Get the scaling factor as inline argument
- Compute the vector scaling

```
1  void scal_cpu_func(void *buffers[], void *cl_arg) {
2      struct starpu_vector_interface *vector_handle = buffers
           [0];
3
4      unsigned n    = STARPU_VECTOR_GET_NX(vector_handle);
5      float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
6
7      float *ptr_factor = cl_arg;
8
9      unsigned i;
10     for (i = 0; i < n; i++)
11         vector[i] *= *ptr_factor;
12 }
```

Submitting a task

The **starpu_task_insert** call

- **Inserts** a task in the StarPU DAG

Submitting a task

The **starpu_task_insert** call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure

```
1 starpu_task_insert(&scal_cl
2                    ...);
```

Submitting a task

The starpu_task_insert call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure
- The StarPU-managed data

```
1 starpu_task_insert(&scal_cl,
2                     STARPU_RW, vector_handle,
3                     ...);
```

Submitting a task

The starpu_task_insert call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure
- The StarPU-managed data
- The small-size inline data

```
1 starpu_task_insert(&scal_cl,
2                     STARPU_RW, vector_handle,
3                     STARPU_VALUE, &factor, sizeof(factor),
4                     ...);
```

Submitting a task

The starpu_task_insert call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure
- The StarPU-managed data
- The small-size inline data
- 0 to mark the end of arguments

```
1 starpu_task_insert(&scal_cl,
2                     STARPU_RW, vector_handle,
3                     STARPU_VALUE, &factor, sizeof(factor),
4                     0);
```

Submitting a task

The **starpu_task_insert** call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure
- The StarPU-managed data
- The small-size inline data
- 0 to mark the end of arguments

Notes

- The task is submitted non-blockingly

Submitting a task

The **starpu_task_insert** call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure
- The StarPU-managed data
- The small-size inline data
- 0 to mark the end of arguments

Notes

- The task is submitted non-blockingly
- Dependencies are enforced with previously submitted tasks' data...

Submitting a task

The **starpu_task_insert** call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure
- The StarPU-managed data
- The small-size inline data
- 0 to mark the end of arguments

Notes

- The task is submitted non-blockingly
- Dependencies are enforced with previously submitted tasks' data...
- ... following the natural order of the program

Submitting a task

The **starpu_task_insert** call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure
- The StarPU-managed data
- The small-size inline data
- 0 to mark the end of arguments

Notes

- The task is submitted non-blockingly
- Dependencies are enforced with previously submitted tasks' data...
- ... following the natural order of the program
- This is the **Sequential Task Flow Paradigm**

Waiting for Submitted Task Completion

- Tasks are submitted non-blockingly

Waiting for Submitted Task Completion

- Tasks are submitted non-blockingly

```
1 /* non-blocking task submits */
2 starpu_task_insert(...);
3 starpu_task_insert(...);
4 starpu_task_insert(...);
5 ...
```

Waiting for Submitted Task Completion

- Tasks are submitted non-blockingly
- Wait for all submitted tasks to complete their work

```
1 /* non-blocking task submits */
2 starpu_task_insert(...);
3 starpu_task_insert(...);
4 starpu_task_insert(...);
5 ...
```

Waiting for Submitted Task Completion

- Tasks are submitted non-blockingly
- Wait for all submitted tasks to complete their work

```
1 /* non-blocking task submits */
2 starpu_task_insert(...);
3 starpu_task_insert(...);
4 starpu_task_insert(...);
5 ...
6
7 /* wait for all task submitted so far */
8 starpu_task_wait_for_all();
```

Basic Example: Scaling a Vector

```
1  float factor = 3.14;
2  float vector[NX];
3  starpu_data_handle_t vector_handle;
4
5  /* ... fill vector ... */
6
7  starpu_vector_data_register(&vector_handle, 0,
8                              (uintptr_t)vector, NX, sizeof(vector[0]))
                                   ;
9
10 starpu_task_insert(
11                   &scal_cl,
12                   STARPU_RW, vector_handle,
13                   STARPU_VALUE, &factor, sizeof(factor),
14                   0);
15
16 starpu_task_wait_for_all();
17 starpu_data_unregister(vector_handle);
18
19 /* ... display vector ... */
```

Heterogeneity: Device Kernels

Extending a codelet to handle heterogeneous platforms

Heterogeneity: Device Kernels

Extending a codelet to handle heterogeneous platforms

- Multiple kernel implementations for a CPU
  - SSE, AVX, ... optimized kernels

```
1 struct starpu_codelet scal_cl = {
2       .cpu_func  = { scal_cpu_func,
3                      scal_sse_cpu_func, scal_avx_cpu_func, NULL },
4       .nbuffers = 1,
5       .modes = { STARPU_RW },
6 };
```

Heterogeneity: Device Kernels

Extending a codelet to handle heterogeneous platforms

- Multiple kernel implementations for a CPU
  - SSE, AVX, ... optimized kernels
- Kernels implementations for accelerator devices
  - OpenCL, NVidia Cuda kernels

```
1 struct starpu_codelet scal_cl = {
2       .cpu_func  = { scal_cpu_func,
3                      scal_sse_cpu_func, scal_avx_cpu_func, NULL },
4       .opencl_func  = { scal_cpu_opencl, NULL },
5       .cuda_func  = { scal_cpu_cuda, NULL },
6       .nbuffers = 1,
7       .modes = { STARPU_RW },
8 };
```

Writing a Kernel Function for CUDA

Writing a Kernel Function for CUDA

```
1
2
3
4
5
6
7
8  extern "C" void scal_cuda_func(void *buffers[], void *cl_arg)
       {
9      struct starpu_vector_interface *vector_handle = buffers
           [0];
10     unsigned n    = STARPU_VECTOR_GET_NX(vector_handle);
11     float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
12     float *ptr_factor = cl_arg;
13
14     ...
15
16
17
18
19 }
```

Writing a Kernel Function for CUDA

```
1
2
3
4
5
6
7
8  extern "C" void scal_cuda_func(void *buffers[], void *cl_arg)
       {
9      struct starpu_vector_interface *vector_handle = buffers
           [0];
10     unsigned n    = STARPU_VECTOR_GET_NX(vector_handle);
11     float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
12     float *ptr_factor = cl_arg;
13
14     unsigned threads_per_block = 64;
15     unsigned nblocks = (n+threads_per_block-1)/
           threads_per_block;
16
17     ...
18
19 }
```

Writing a Kernel Function for CUDA

```
1
2
3
4
5
6
7
8  extern "C" void scal_cuda_func(void *buffers[], void *cl_arg)
       {
9      struct starpu_vector_interface *vector_handle = buffers
           [0];
10     unsigned n    = STARPU_VECTOR_GET_NX(vector_handle);
11     float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
12     float *ptr_factor = cl_arg;
13
14     unsigned threads_per_block = 64;
15     unsigned nblocks = (n+threads_per_block-1)/
           threads_per_block;
16
17     vector_mult_cuda<<<nblocks,threads_per_block,0,
18         starpu_cuda_get_local_stream()>>>(n,vector,*
               ptr_factor);
19 }
```

Writing a Kernel Function for CUDA

```
1  static __global__ void vector_mult_cuda(unsigned n,
2                                          float *vector, float factor
                                               ){
3      unsigned i = blockIdx.x*blockDim.x + threadIdx.x;
4
5      ...
6  }
7
8  extern "C" void scal_cuda_func(void *buffers[], void *cl_arg)
       {
9      struct starpu_vector_interface *vector_handle = buffers
           [0];
10     unsigned n      = STARPU_VECTOR_GET_NX(vector_handle);
11     float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
12     float *ptr_factor = cl_arg;
13
14     unsigned threads_per_block = 64;
15     unsigned nblocks = (n+threads_per_block-1)/
           threads_per_block;
16
17     vector_mult_cuda<<<nblocks,threads_per_block,0,
18         starpu_cuda_get_local_stream()>>>(n,vector,*
               ptr_factor);
19 }
```

Writing a Kernel Function for CUDA

```
1  static __global__ void vector_mult_cuda(unsigned n,
2                                          float *vector, float factor
                                               ){
3      unsigned i = blockIdx.x*blockDim.x + threadIdx.x;
4      if (i < n)
5          vector[i] *= factor;
6  }
7
8  extern "C" void scal_cuda_func(void *buffers[], void *cl_arg)
       {
9      struct starpu_vector_interface *vector_handle = buffers
           [0];
10     unsigned n      = STARPU_VECTOR_GET_NX(vector_handle);
11     float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
12     float *ptr_factor = cl_arg;
13
14     unsigned threads_per_block = 64;
15     unsigned nblocks = (n+threads_per_block-1)/
           threads_per_block;
16
17     vector_mult_cuda<<<nblocks,threads_per_block,0,
18         starpu_cuda_get_local_stream()>>>(n,vector,*
               ptr_factor);
19 }
```

StarPU Scheduling Policies

- No *one size fits all* policy
- Selectable scheduling policy
  - Predefined set of popular policies: eager, work-stealing, etc.

StarPU Scheduling Policies

- No *one size fits all* policy
- Selectable scheduling policy
  - Predefined set of popular policies: eager, work-stealing, etc.

Going beyond?

StarPU Scheduling Policies

- No *one size fits all* policy
- Selectable scheduling policy
  - Predefined set of popular policies: eager, work-stealing, etc.

Going beyond?

Scheduling is a decision process:

- Providing more input to the scheduler...
- ... can lead to better scheduling decisions

What kind of information?

- Relative importance of tasks
  - Priorities
- Cost of tasks
  - Codelet models
- Cost of transferring data
  - Bus calibration

Selecting a Scheduling Policy

- Use the `STARPU_SCHED` environment variable

Selecting a Scheduling Policy

- Use the STARPU_SCHED environment variable
- Example 1: selecting the prio scheduler

```
$ export STARPU_SCHED=prio
$ my_program
...
```

Selecting a Scheduling Policy

- Use the `STARPU_SCHED` environment variable
- Example 1: selecting the prio scheduler
- Example 2: selecting the dm scheduler

```
$ export STARPU_SCHED=prio
$ my_program
...
```

```
$ export STARPU_SCHED=dm
$ my_program
...
```

Selecting a Scheduling Policy

- Use the `STARPU_SCHED` environment variable
- Example 1: selecting the prio scheduler
- Example 2: selecting the dm scheduler
- Example 3: resetting to default scheduler eager

```
$ export STARPU_SCHED=prio
$ my_program
...
```

```
$ export STARPU_SCHED=dm
$ my_program
...
```

```
$ unset STARPU_SCHED
$ my_program
...
```

Selecting a Scheduling Policy

- Use the `STARPU_SCHED` environment variable
- Example 1: selecting the prio scheduler
- Example 2: selecting the dm scheduler
- Example 3: resetting to default scheduler eager
- No need to recompile the application

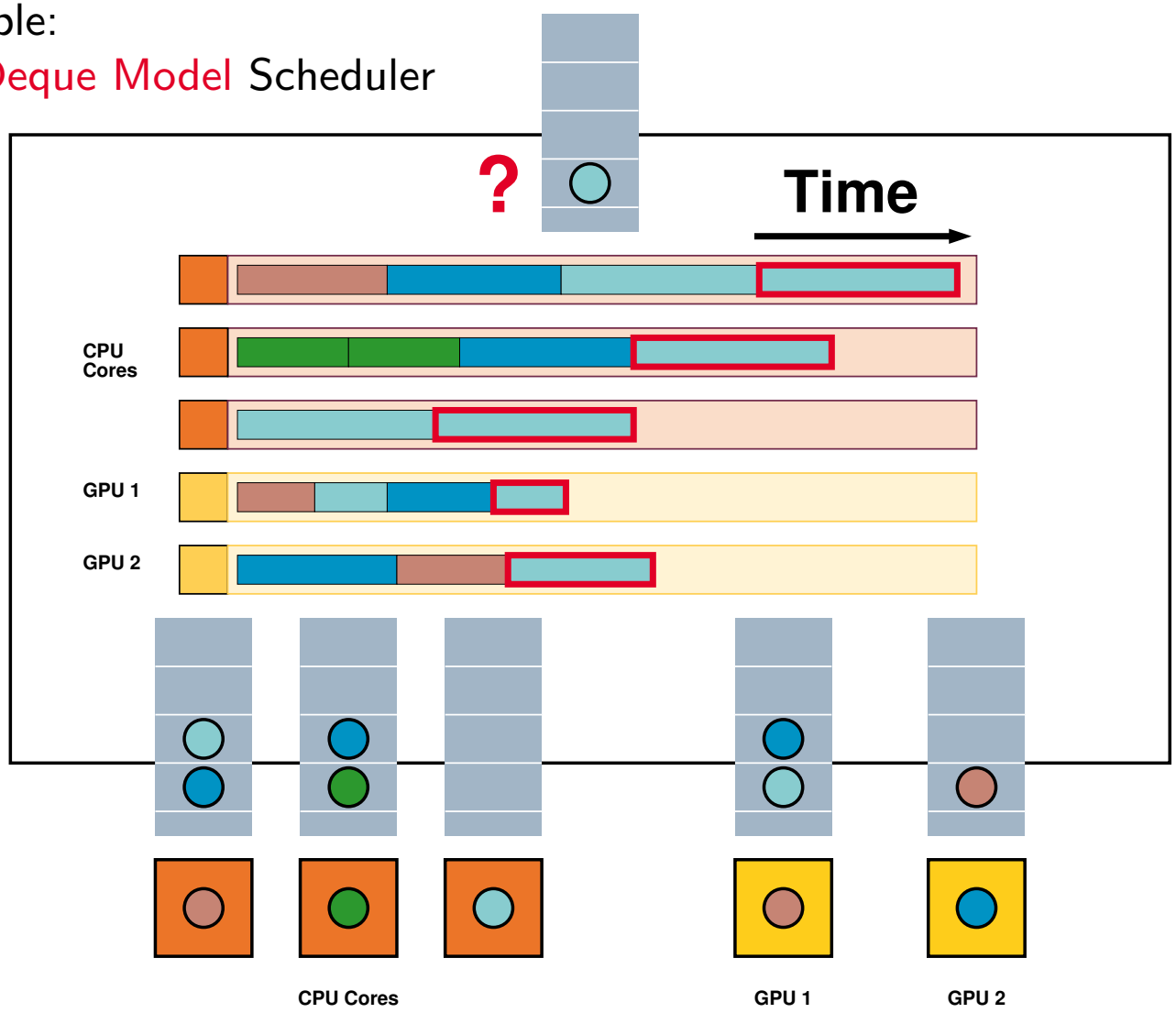
```
$ export STARPU_SCHED=prio
$ my_program
...
```

```
$ export STARPU_SCHED=dm
$ my_program
...
```

```
$ unset STARPU_SCHED
$ my_program
...
```

Task Mapping using a Performance Model

- Example:
  The Deque Model Scheduler



Task Mapping using a Performance Model

- Using codelet performance models
  - Kernel calibration on each available computing device
  - **Raw** history model of kernels' past execution times
  - **Refined** models using regression on kernels' execution times history
- Model parameter(s)
  - **Data size**
  - **User-defined** parameters

Data Transfer Cost Modelling for Improved Scheduling

Discrete accelerators

- CPU $\leftrightarrow$ GPU transfers
- Data transfer cost vs kernel offload benefit

Transfer cost modelling

- Bus calibration
  - Can differ even for identical devices
  - Platform's topology

Data-transfer aware scheduling

- Deque Model Data Aware (dmda) scheduling policy variants
- Tunable data transfer cost bias
  - locality
  - vs load balancing

Data Prefetching

Task states

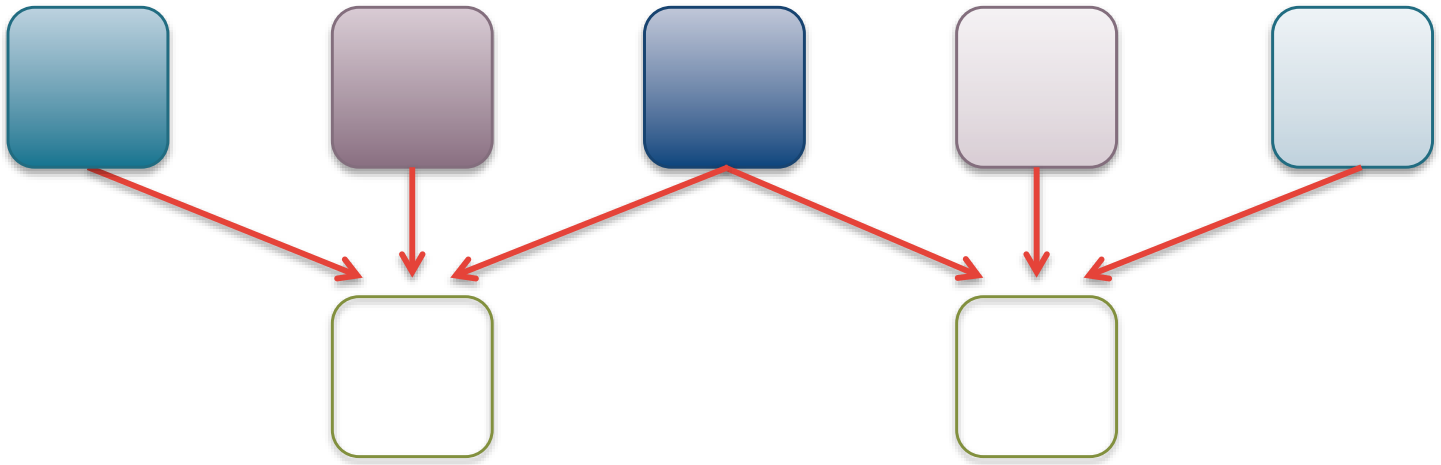
- Submitted
  - Task inserted by the application
- Ready
  - Task's dependencies resolved
- **Scheduled**
  - Task queued on a computing unit
- **Executing**
  - Task running on a computing unit

Anticipate on the **Scheduled** $\rightarrow$ **Executing** transition

- Prefetch triggered ASAP after **Scheduled** state
- Prefetch may also be triggered by the application

Unstructured data dependences

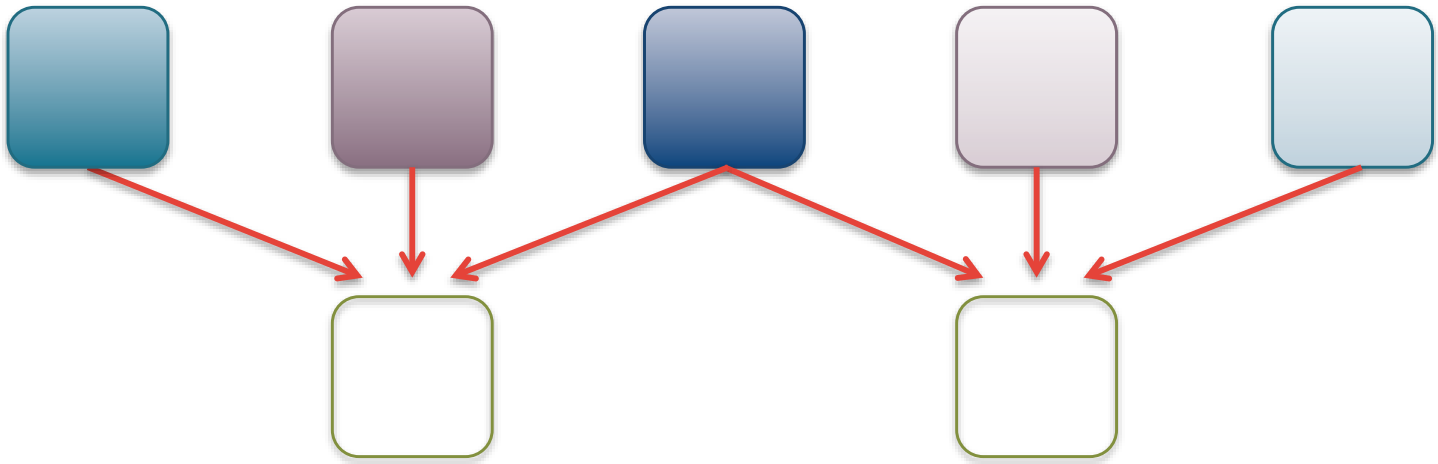
- Applications
  - N-body
  - Unstructured meshes
  - Multiple logically concurrent updates



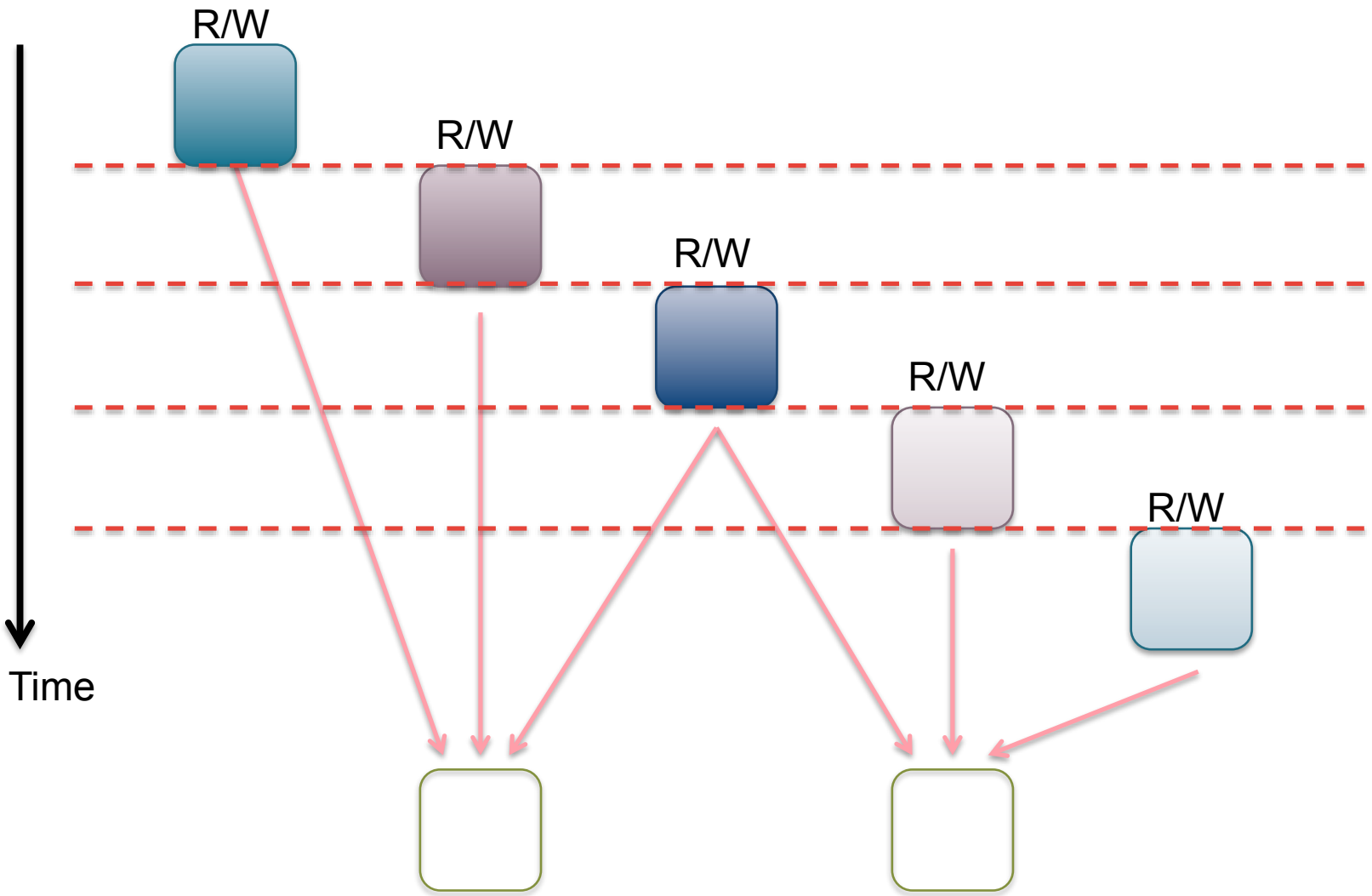
Unstructured data dependences

- Applications
  - N-body
  - Unstructured meshes
  - Multiple logically concurrent updates

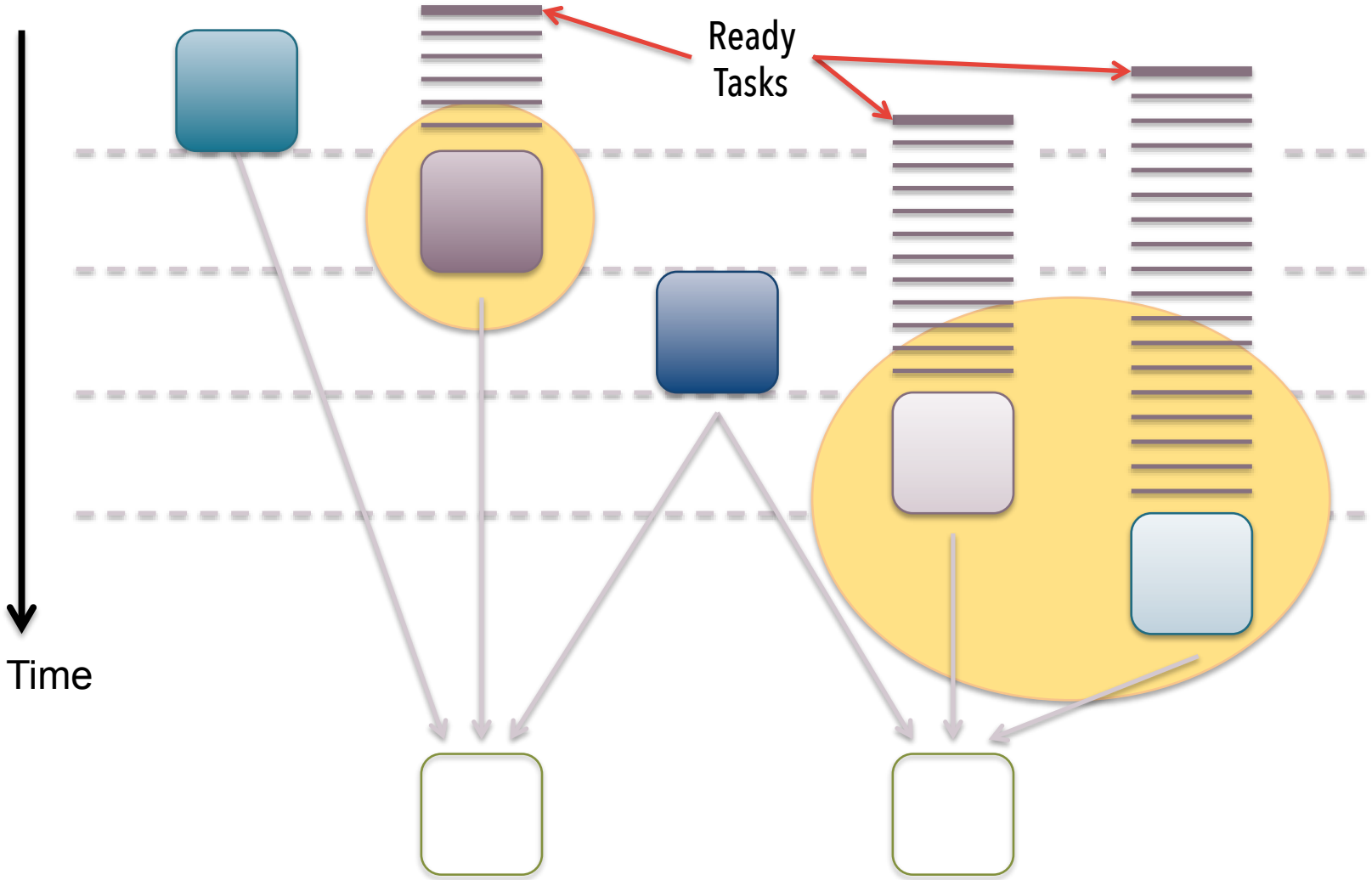
- Many tasks contributing to shared pieces of data (cells, particles)...
  - ... without natural, sequential order



Artificially ordered, sequential execution



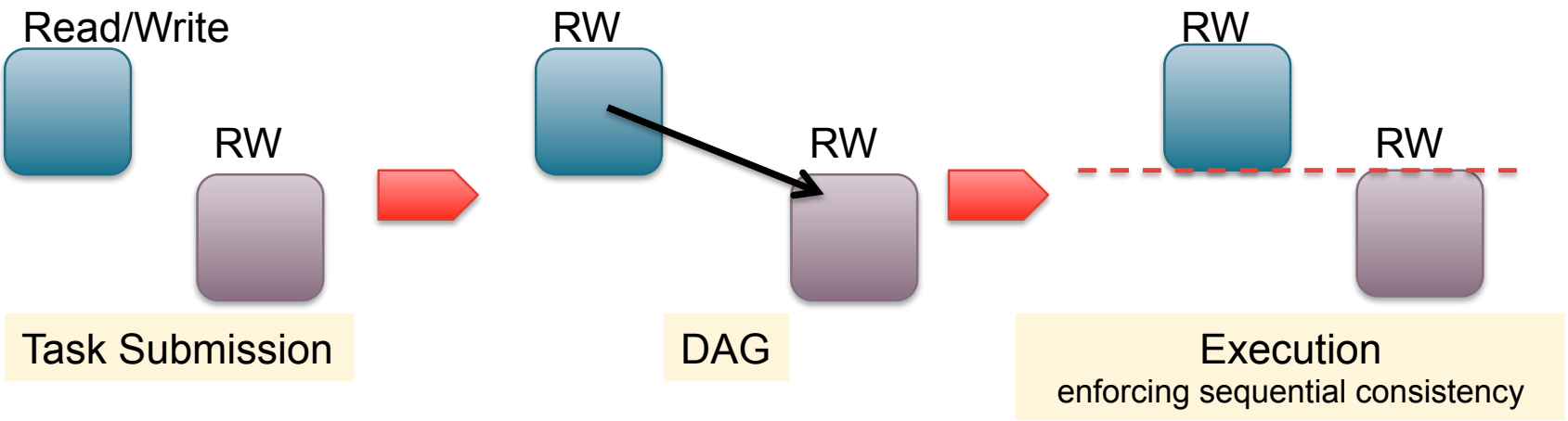
Loss of Parallelism



Relaxing Over-Constrained Multiple Updates

Issue with the dual purpose of usual Read/Write data dependence mode

- Mutual exclusion
  - Avoids two tasks modifying a piece of data concurrently
- Orderings
  - Enforces sequential consistency



Use StarPU's commutative dependence mode

Semantics

- Keep mutual exclusion role
- Relax ordering role

Rationale

- Integrate Element / Particle / Cell / Entity contributions ASAP
  - without artificial ordering, unwanted serialization
- Preserve all other StarPU functionalities (e.g.: heterogeneous scheduling, etc.)
- Straightforward use:

```
1 starpu_task_insert(&codelet,
2                     STARPU_R, fmic->wn_iv_handle[iv],
3                     STARPU_RW, fmac->wn_handle,
4                     ... );
```

Use StarPU's commutative dependence mode

Semantics

- Keep mutual exclusion role
- Relax ordering role

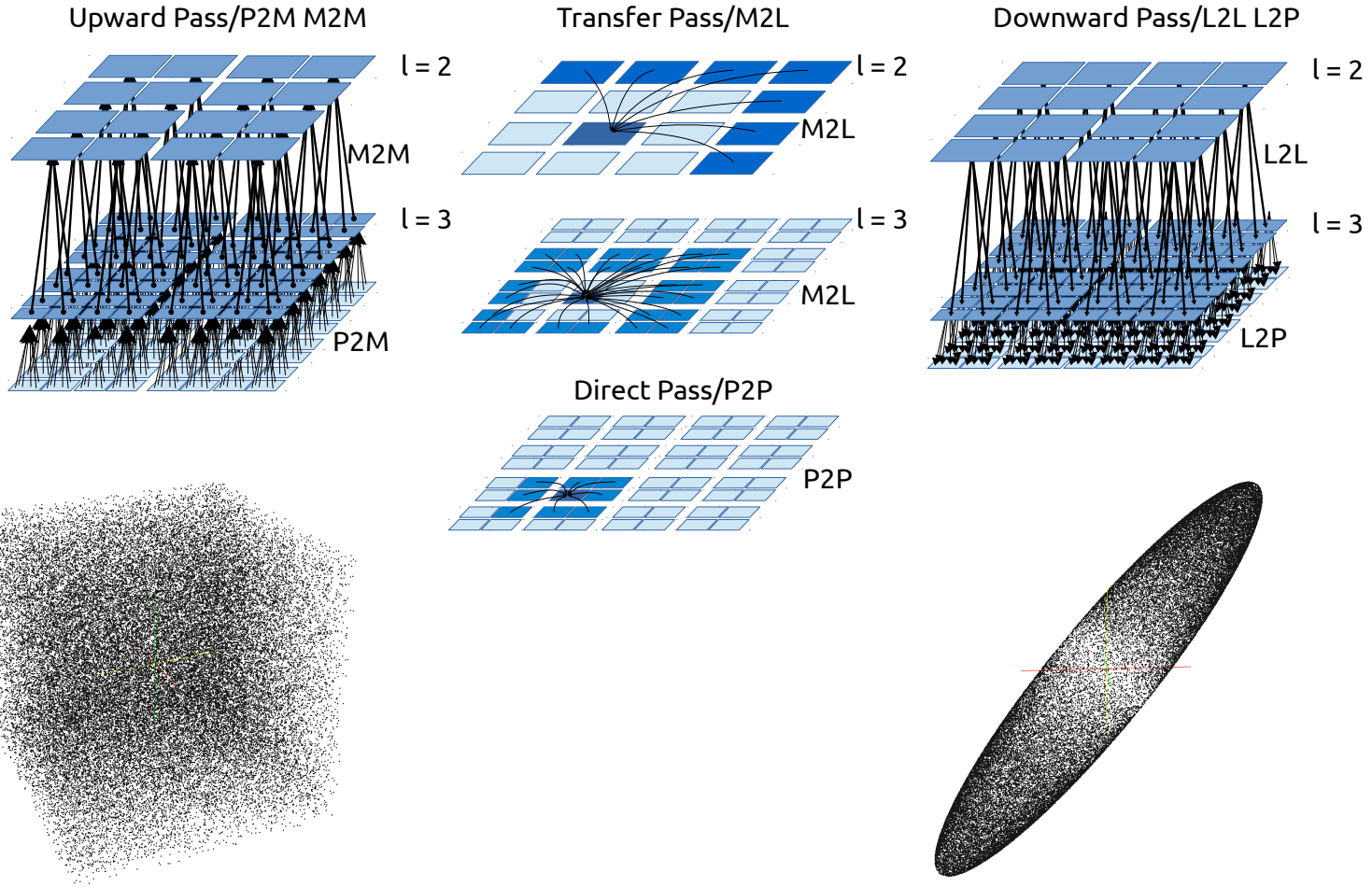
Rationale

- Integrate Element / Particle / Cell / Entity contributions ASAP
  - without artificial ordering, unwanted serialization
- Preserve all other StarPU functionalities (e.g.: heterogeneous scheduling, etc.)
- Straightforward use:

```
1 starpu_task_insert(&codelet,
2                     STARPU_R, fmic->wn_iv_handle[iv],
3                     STARPU_RW|STARPU_COMMUTE, fmac->wn_handle,
4                     ... );
```

Example: ScalFMM Fast Multipole Method

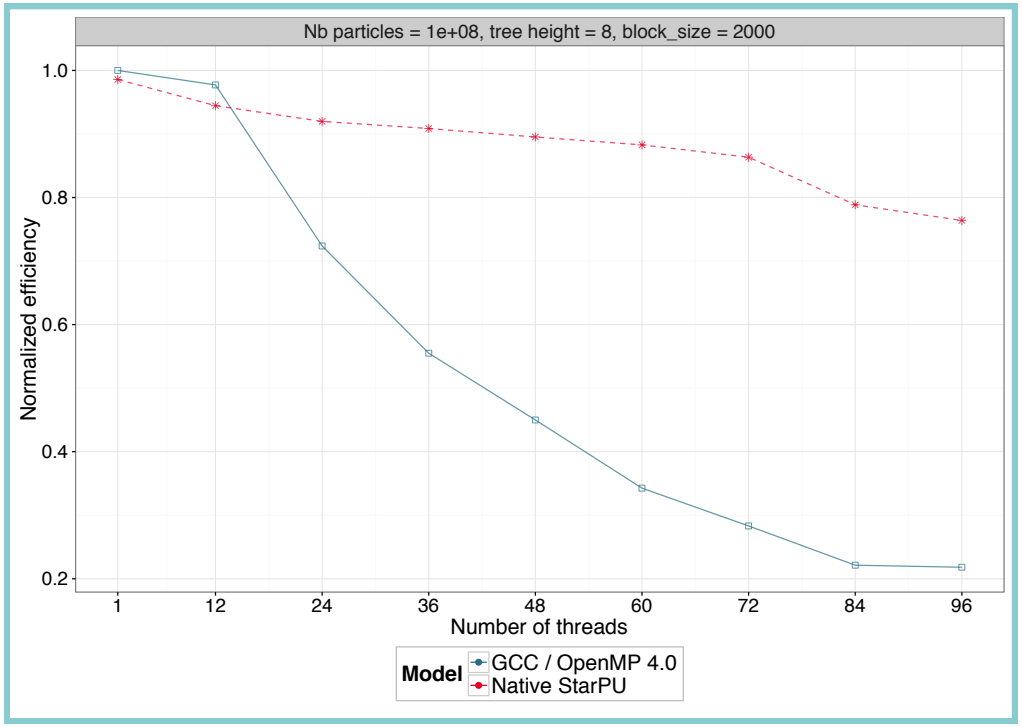
ScalFMM framework (Inria HiePACS)



Example: ScalFMM Fast Multipole Method

ScalFMM framework (Inria HiePACS)

- Test on 96-core homogeneous platform
- Comparative normalized efficiency
- GCC OpenMP 4
- Native StarPU
  - Base
  - Priorities + **Commutative dependencies**
- **Study available in Inria Research Report RR-8953**



2.6

High Level Programming Support

KStar OpenMP C/C++ Compiler

High level programming

- Source-to-source compiler
- Translate directives into runtime system API calls
  - **StarPU** Runtime System
  - Kaapi Runtime System (Inria Team AVALON, pole 4)
- OpenMP 3.1
  - Virtually full support
- OpenMP 4.0
  - Dependent tasks
  - Heterogeneous targets (on-going work)
- Based on LLVM/Clang

Available on:

- KStar project website – **http://kstar.gforge.inria.fr/**

Inria joined the OpenMP ARB standardisation consortium

OpenMP Example: Tasks

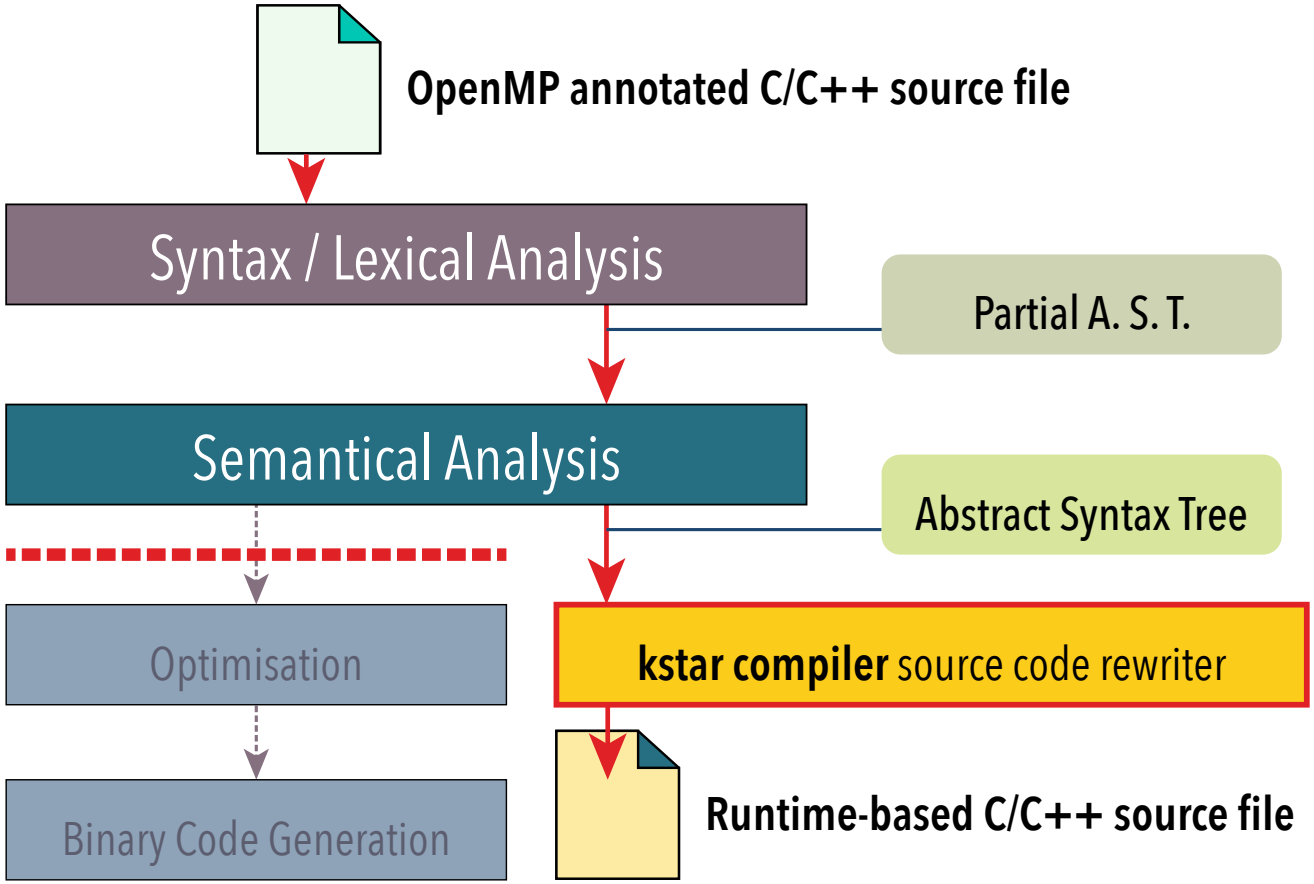
```
1  int item[N];
2
3  void g(int);
4
5  void f()
6  {
7  #pragma omp parallel
8      {
9  #pragma omp single
10         {
11             int i;
12             for (i=0; i<N; i++)
13 #pragma omp task untied
14                 g(item[i]);
15         }
16     }
17 }
```

OpenMP Example: Task Dependencies

```
1  void f()
2  {
3      int a;
4
5  #pragma omp parallel
6  #pragma omp single
7      {
8  #pragma omp task shared(a) depend(out: a)
9          foo(&a);
10
11 #pragma omp task shared(a) depend(in: a)
12         bar(&a);
13     }
14 }
```

KStar Compiler Architecture

OpenMP directive translations implemented as an AST Rewriter



Source code Translation

```c
void print(const char *msg) {
    printf("%s\n", msg);
}

int main( int argc, char** argv) {
    char mon_msg[] = "Hello !";
#pragma omp task
    print(mon_msg);
#pragma omp taskwait
    return 0;
}
```

```c
void print(const char *msg) {
    printf("%s\n", msg);
}

/* Generated arg struct */
struct __gen_argstruct {
  char [8] mon_msg;
};

int main( int argc, char** argv) {
    char mon_msg[] = "Hello !";
  /* Generated task spawn */
  omp_push_task(wrapper, args);
  /* Generated taskwait */
  omp_sched_sync();
  return 0;
}

/* Generated wrapper */
void wrapper(void *_k_arg) {
  // The captured stmts param
  __gen_argstruct *args = (__gen_argstruct
        *)_k_arg;
  char [8] mon_msg = args->mon_msg;
  // The captured stmts
  print(mon_msg);
}
/* ... */
```

KStar OpenMP Extensions

Experiments with new capabilities, constructs

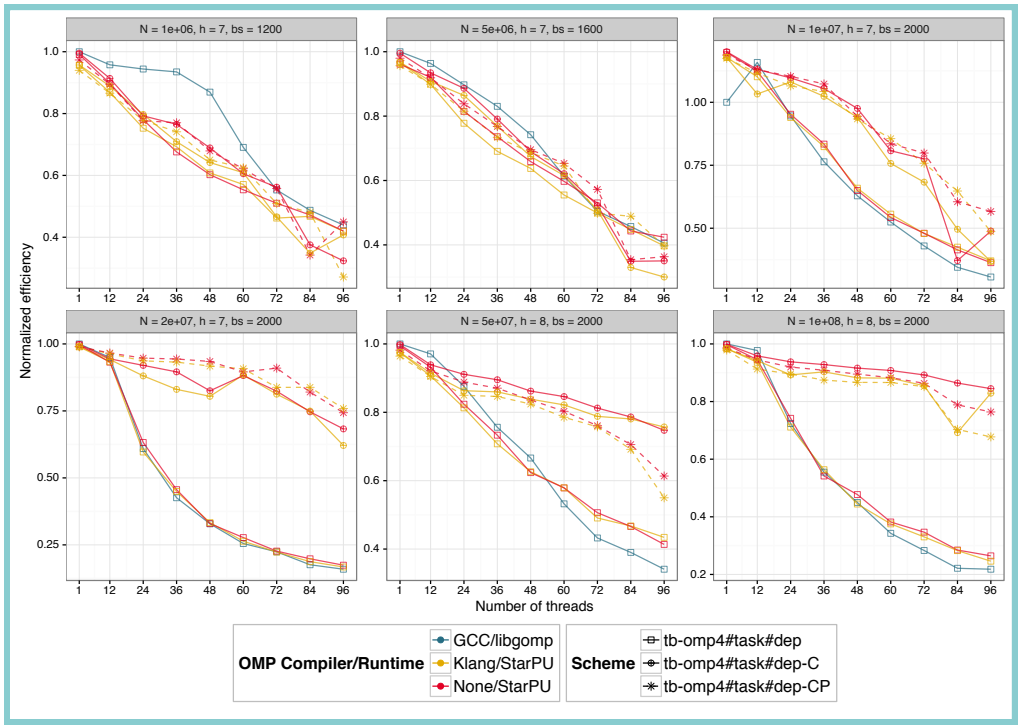
Benefiting from OpenMP's productivity...
... while leveraging advanced runtime system capabilities

- Task priorities (now included in OpenMP)
- Concurrent write
- Commutative dependencies
- Selectable task scheduling algorithm
- Heterogeneous task scheduling
  - Dynamic task implementation selection

Example: ScalFMM Fast Multipole Method + KStar

ScalFMM framework (Inria HiePACS)

- Test on 96-core homogeneous platform
- Comparative normalized efficiency
- GCC OpenMP 4
- KStar OpenMP 4 + StarPU
  - Base
  - Priorities
  - **Commutative dependencies**
- Native StarPU
  - Base
  - Priorities
  - Commutative dependencies



- **Study available in Inria Research Report RR-8953**

Feedback mechanisms

Online Tools

- Statistics
- Visual debugging

Offline Tools

- Trace-based analysis

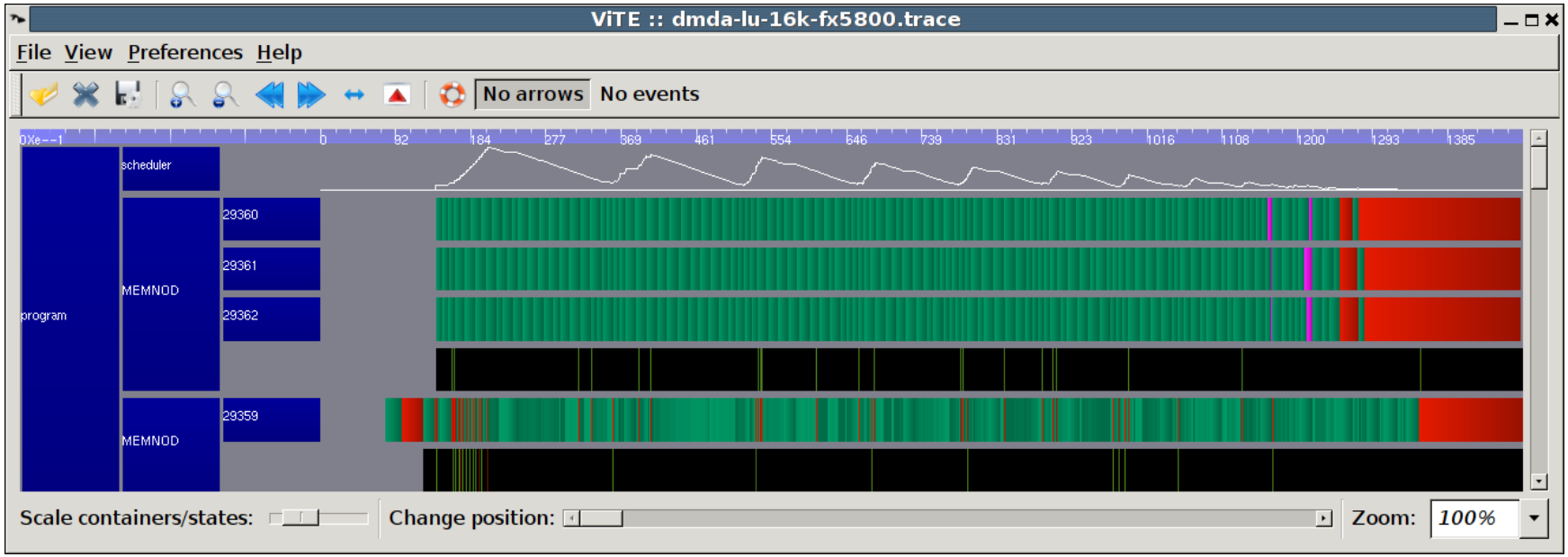
Offline Trace-Based Feedback

- FxT trace collection
- Trace analysis and display
  - ViTE Gantt
  - Graphviz DAG
  - R plots

Offline Feedback – Trace Analysis

Automatically generated

- Dependency graph (DAG)
- Activity diagramm (GANTT)
  - Visualize with ViTE



Offline Feedback – Kernel Model

Display the codelet performance models recorded by StarPU

- Command-line tool starpu_perfmodel_display
- History-based models
- Regression-based models

Offline Feedback – Kernel Model

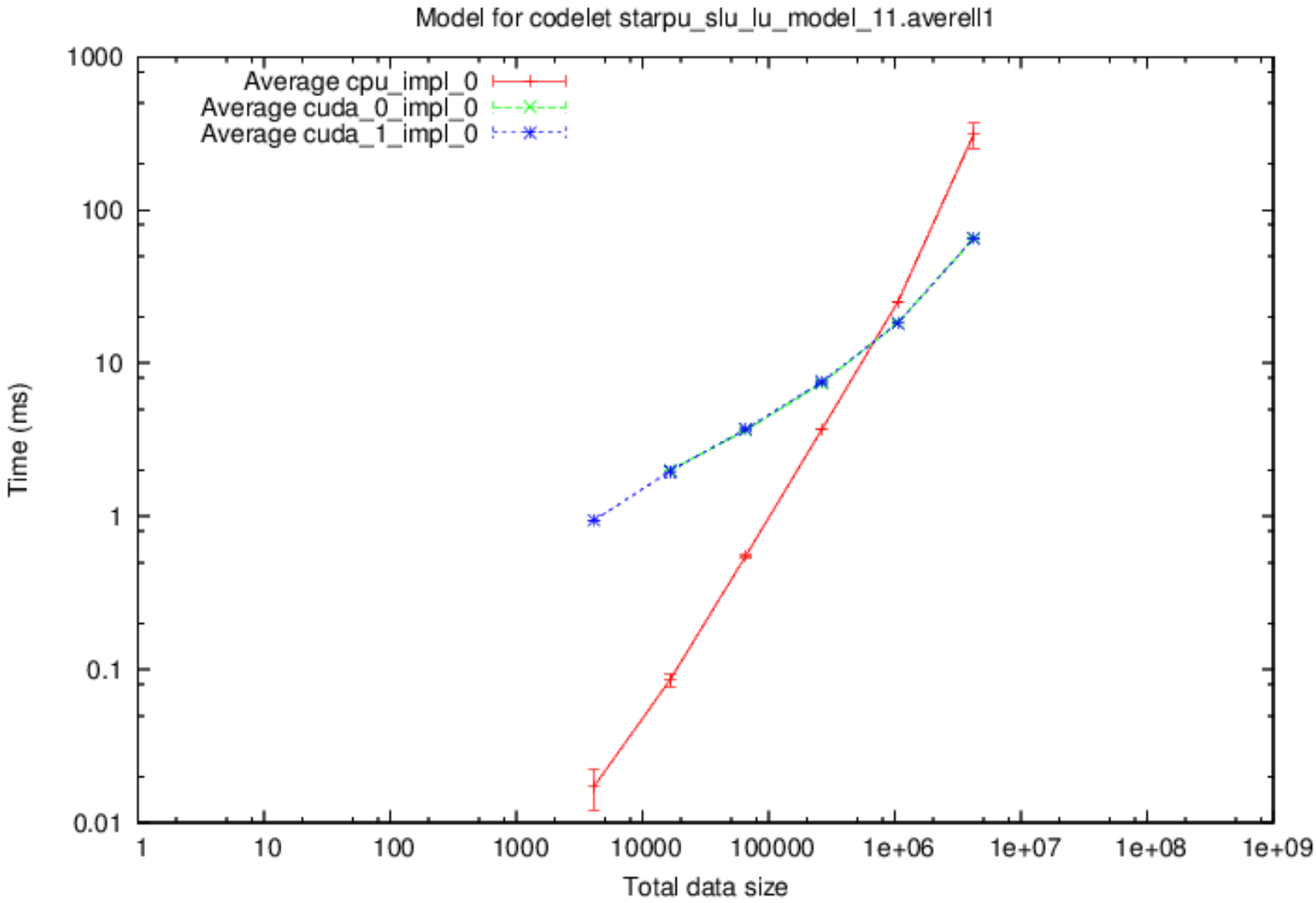
Display the codelet performance models recorded by StarPU

- Command-line tool starpu_perfmodel_display
- History-based models
- Regression-based models

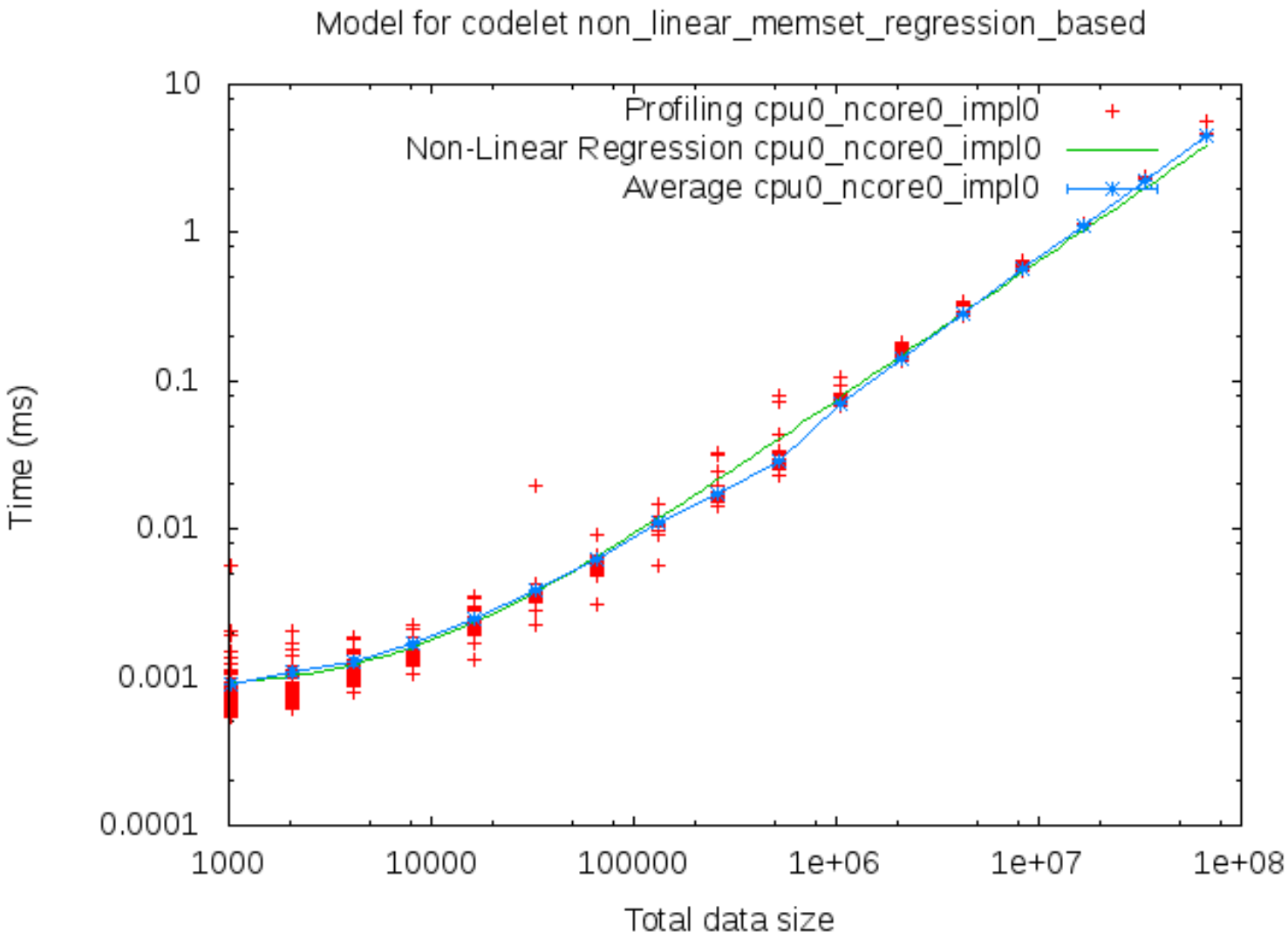
```
$ starpu_perfmodel_display -s starpu_slu_lu_model_11

performance model for cpu0_parallel1_impl0
# hash         size         mean (us)         stddev (us)       n
aa6d4ef7       4194304      3.055501e+05      5.804822e+04      48
```

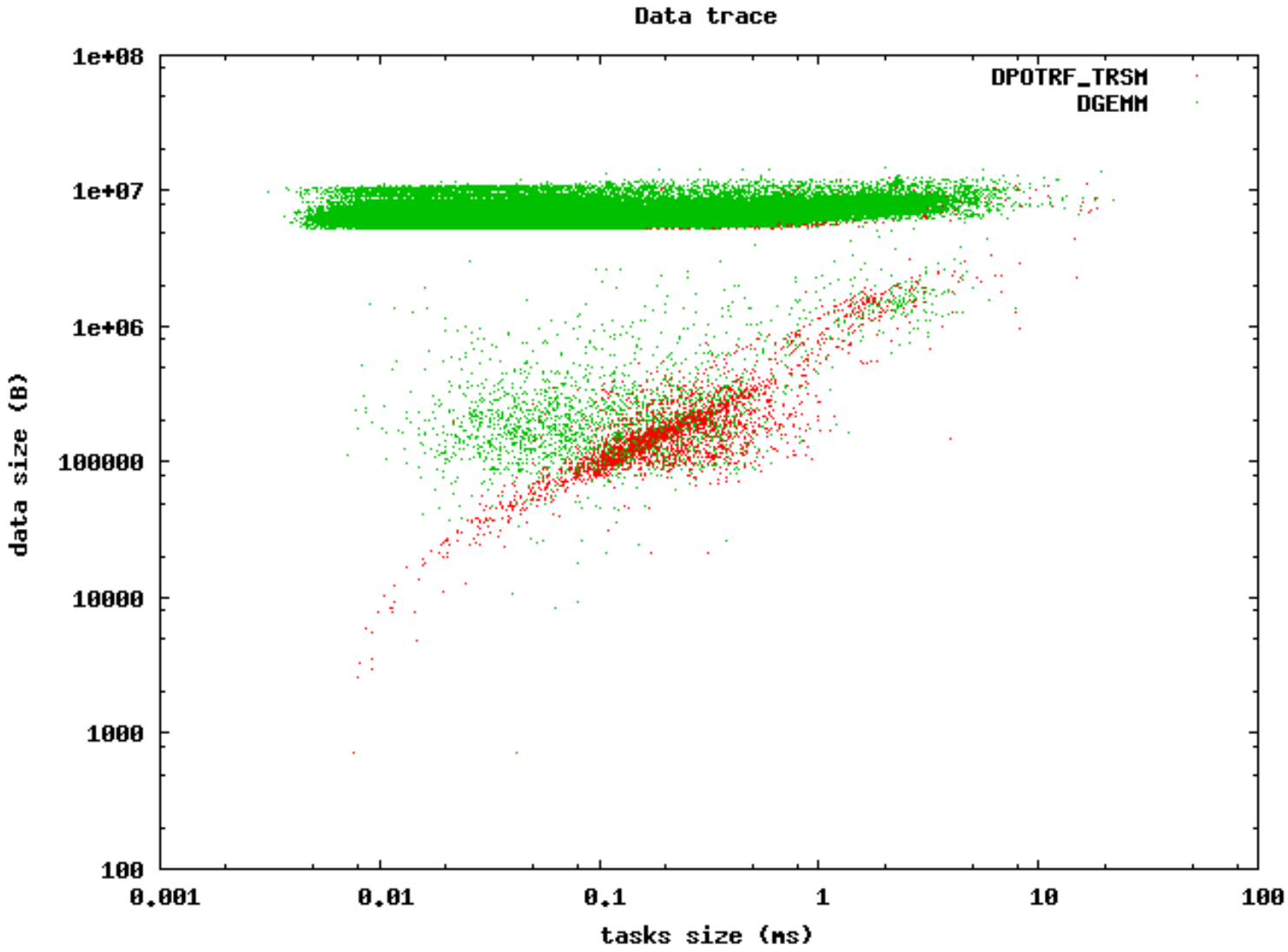
Offline Feedback – Kernel Model Characteristics



Offline Feedback – Kernel Model Regression Fitness

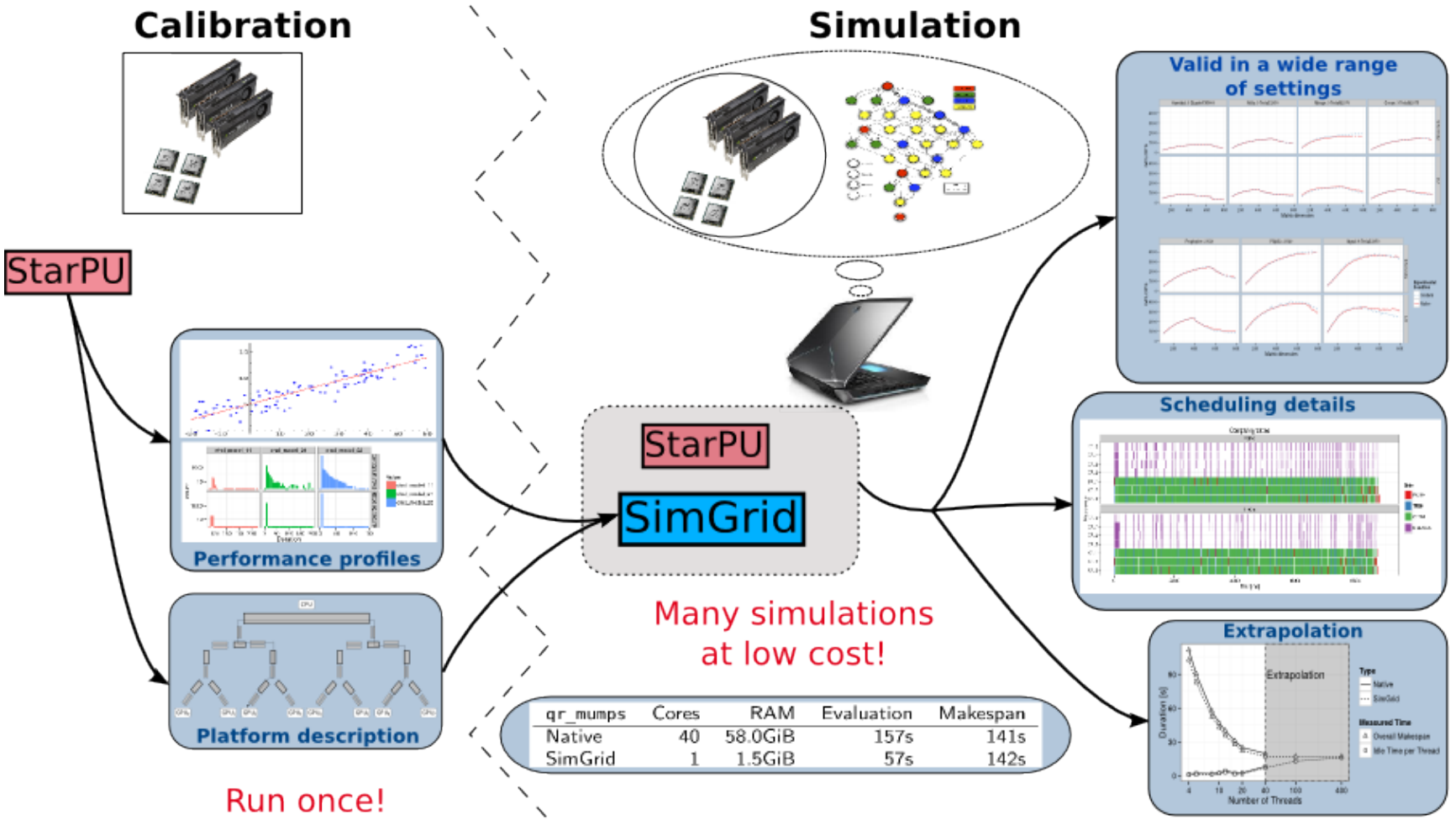


Offline Feedback – Synthetic Kernels' Behaviour



StarPU-SimGrid in a nutshell

PhD L. Stanisic



qr_mumps	Cores	RAM	Evaluation	Makespan
Native	40	58.0GiB	157s	141s
SimGrid	1	1.5GiB	57s	142s

Analysis and Simulation with StarPU + SimGrid

Ph.D Suraj Kumar
Inria STORM, Inria POLARIS, Inria HiePACS, Inria RealOpt

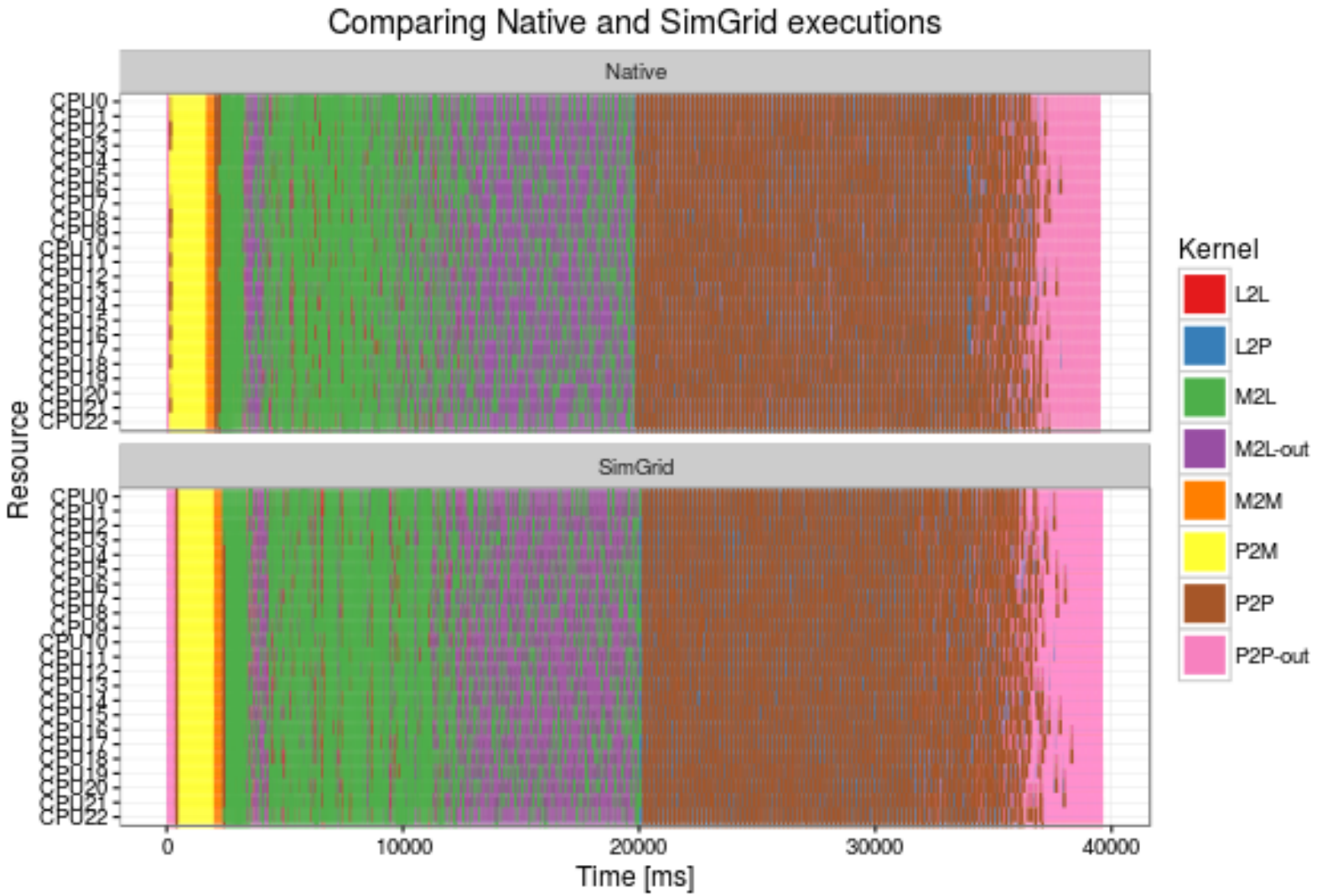
Scheduling tasks without executing kernels

- Builds on the SimGrid simulation environment
- Enables simulating large-scale scenarios
  - Large data sets
  - Large simulated hardware plaform
- Relies on **real** performance models...
- ...collected by StarPU on a real machine

Quickly exploring

- Enables fast experiments when designing **new scheduling algorithms**
- Enables fast experiments when designing **new platforms**

ScalFMM Simulation with StarPU/SimGrid (L. Stanisic)



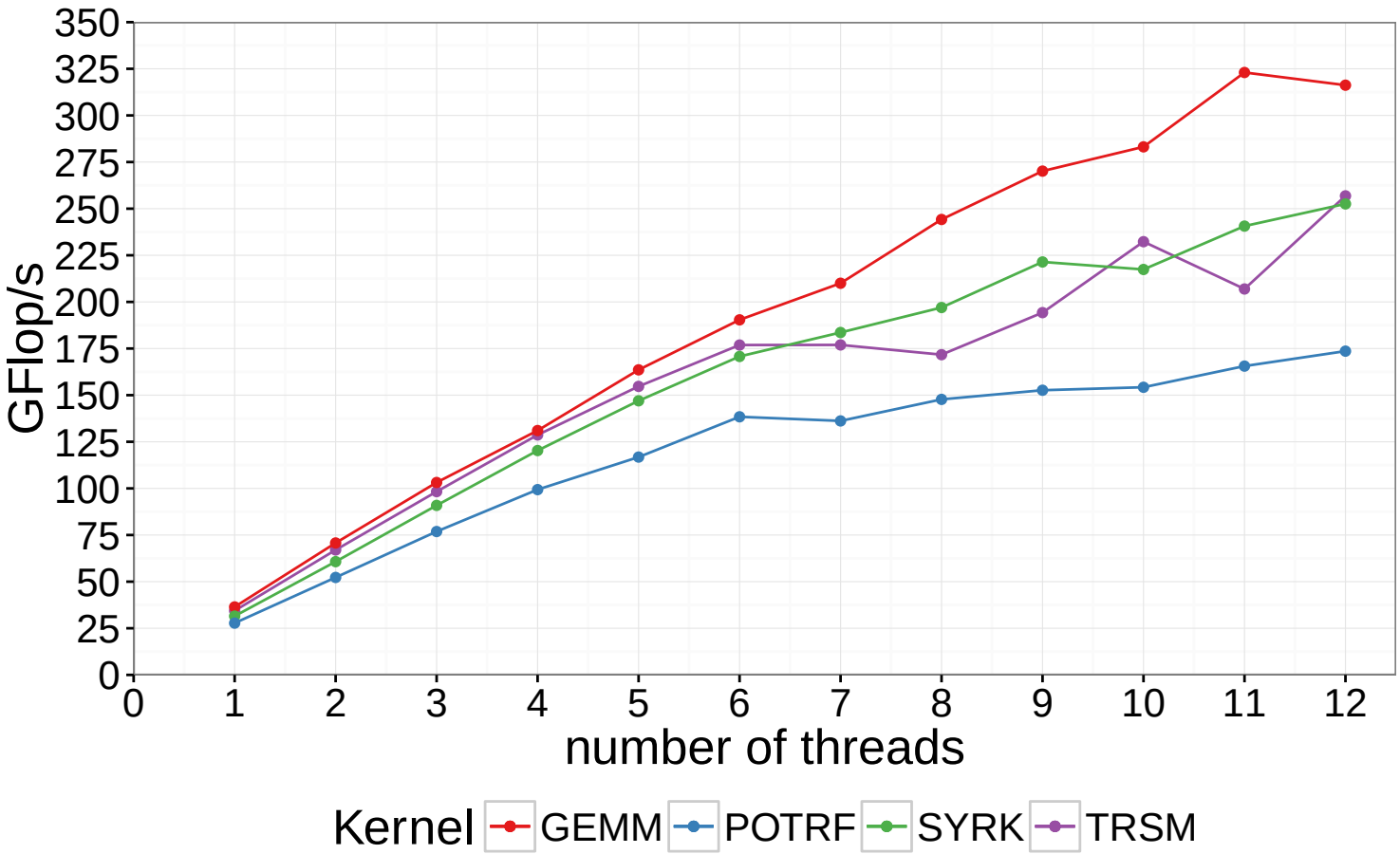
2.7

Going-further

Multicore CPUs: Parallel Tasks

Multicore CPUs: Parallel Tasks (T. Cojean)

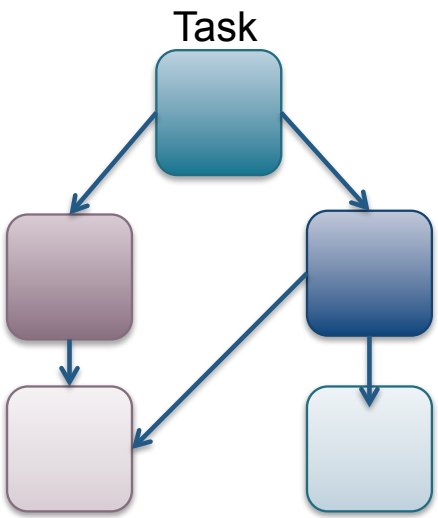
Kernel sweet spots: example with Cholesky factorization kernels
(1x Xeon E5-2680v3 2.5GHz 12 cores)



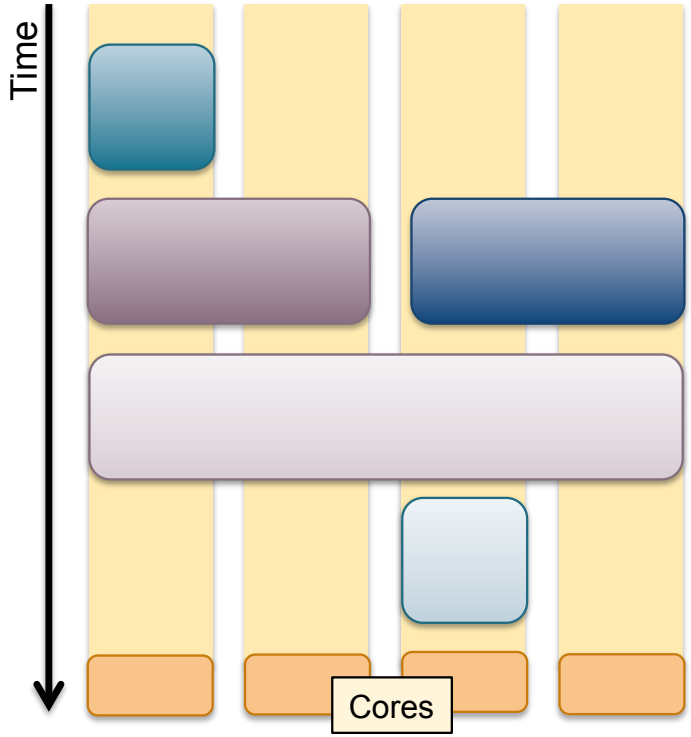
Multicore CPUs: Parallel Tasks

Rationale

- Run parallel kernels on multiple CPU cores
- Address CPU/GPU computing power imbalance
- Address nested-runtime interoperability



Task DAG



Execution

Multicore CPUs: Parallel Tasks

Rationale

- Run parallel kernels on multiple CPU cores
- Address CPU/GPU computing power imbalance
- Address nested-runtime interoperability

Reduce computing power imbalance between CPU and GPU

- Big kernel for GPU
- Small kernel for a single CPU core
- Run “bigger” kernel on several CPU cores

Multicore CPUs: Parallel Tasks

Rationale

- Run parallel kernels on multiple CPU cores
- Address CPU/GPU computing power imbalance
- Address nested-runtime interoperability

Reduce computing power imbalance between CPU and GPU

- Big kernel for GPU
- Small kernel for a single CPU core
- Run "bigger" kernel on several CPU cores

Make use of existing parallel kernels/codes

- Interoperability
- Libraries: BLAS, FFT, ...
- OpenMP code

Multicore CPUs – Technical details

Two flavors of **parallel tasks**

Multicore CPUs – Technical details

Two flavors of **parallel tasks**

Fork-mode

- StarPU provides threads on the participating cores

Multicore CPUs – Technical details

Two flavors of **parallel tasks**

Fork-mode

- StarPU provides threads on the participating cores

SPMD-mode

- StarPU launches the task on a single core
- . . . and let the task create its own threads
  - Black-box mode

Multicore CPUs – Technical details

Two flavors of **parallel tasks**

Fork-mode

- StarPU provides threads on the participating cores

SPMD-mode

- StarPU launches the task on a single core
- . . . and let the task create its own threads
  - Black-box mode

Locality enforcement in NUMA context

- Combined worker threads

Composing Multiple Codes

Rationale

Composing Multiple Codes

Rationale

- Sharing computing resources. . .

Composing Multiple Codes

Rationale

- Sharing computing resources...
- ... among multiple DAGs

Composing Multiple Codes

Rationale

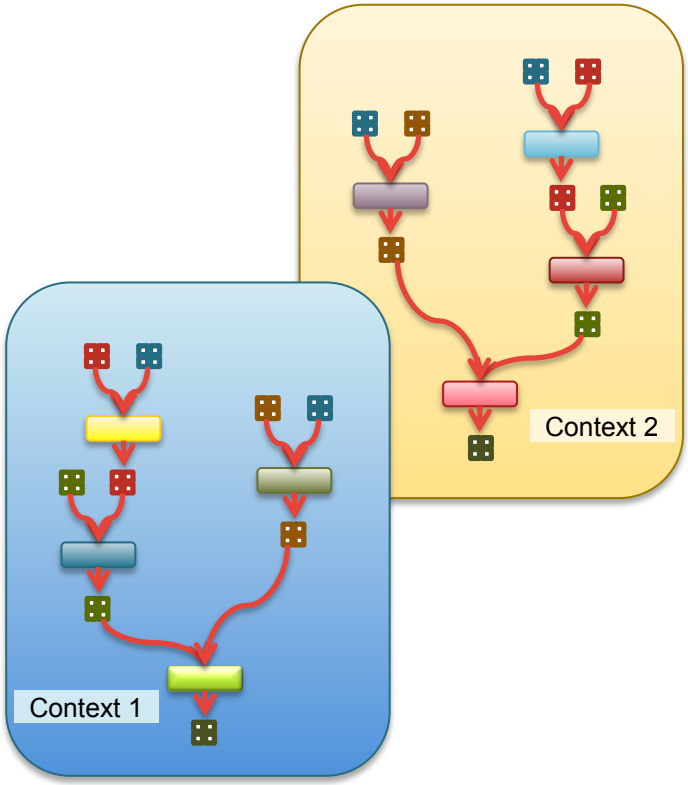
- Sharing computing resources...
- ... among multiple DAGs
- ... simultaneously

Composing Multiple Codes

Rationale

- Sharing computing resources...
- ... among multiple DAGs
- ... simultaneously

Scheduling Contexts



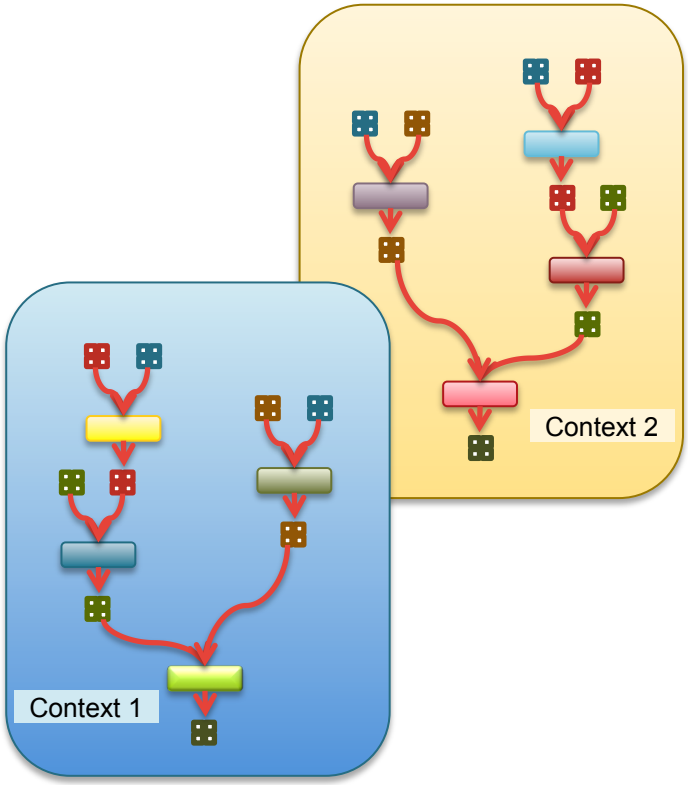
Composing Multiple Codes

Rationale

- Sharing computing resources...
- ... among multiple DAGs
- ... simultaneously

Scheduling Contexts

- Map DAGs on subsets of computing units



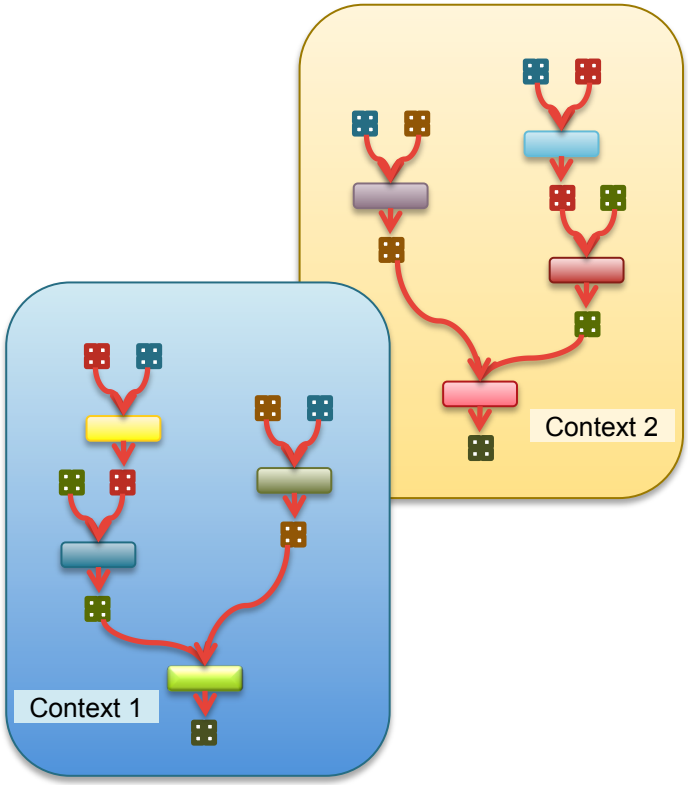
Composing Multiple Codes

Rationale

- Sharing computing resources...
- ... among multiple DAGs
- ... simultaneously

Scheduling Contexts

- Map DAGs on subsets of computing units
- Isolate competing kernels or library calls
  - OpenMP kernel, Intel MKL, etc.



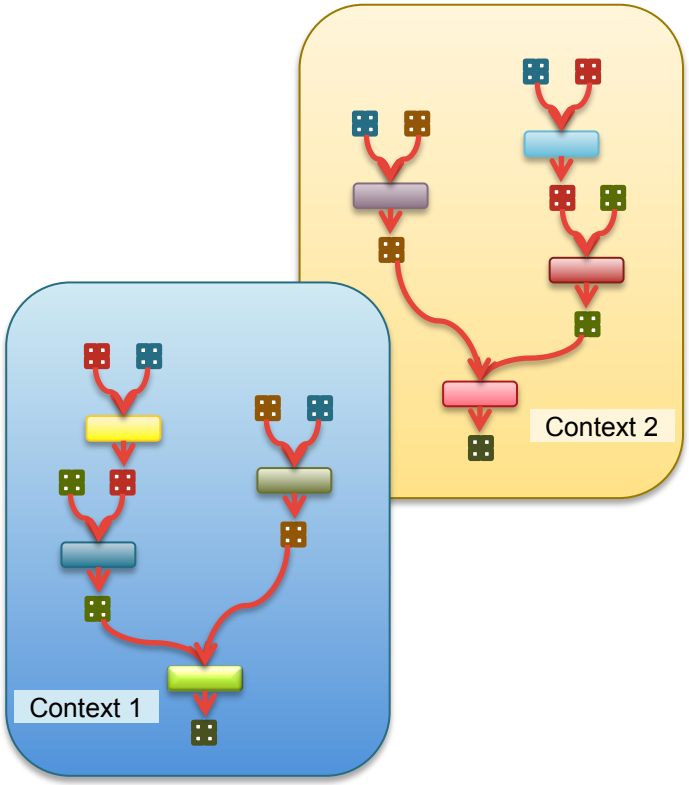
Composing Multiple Codes

Rationale

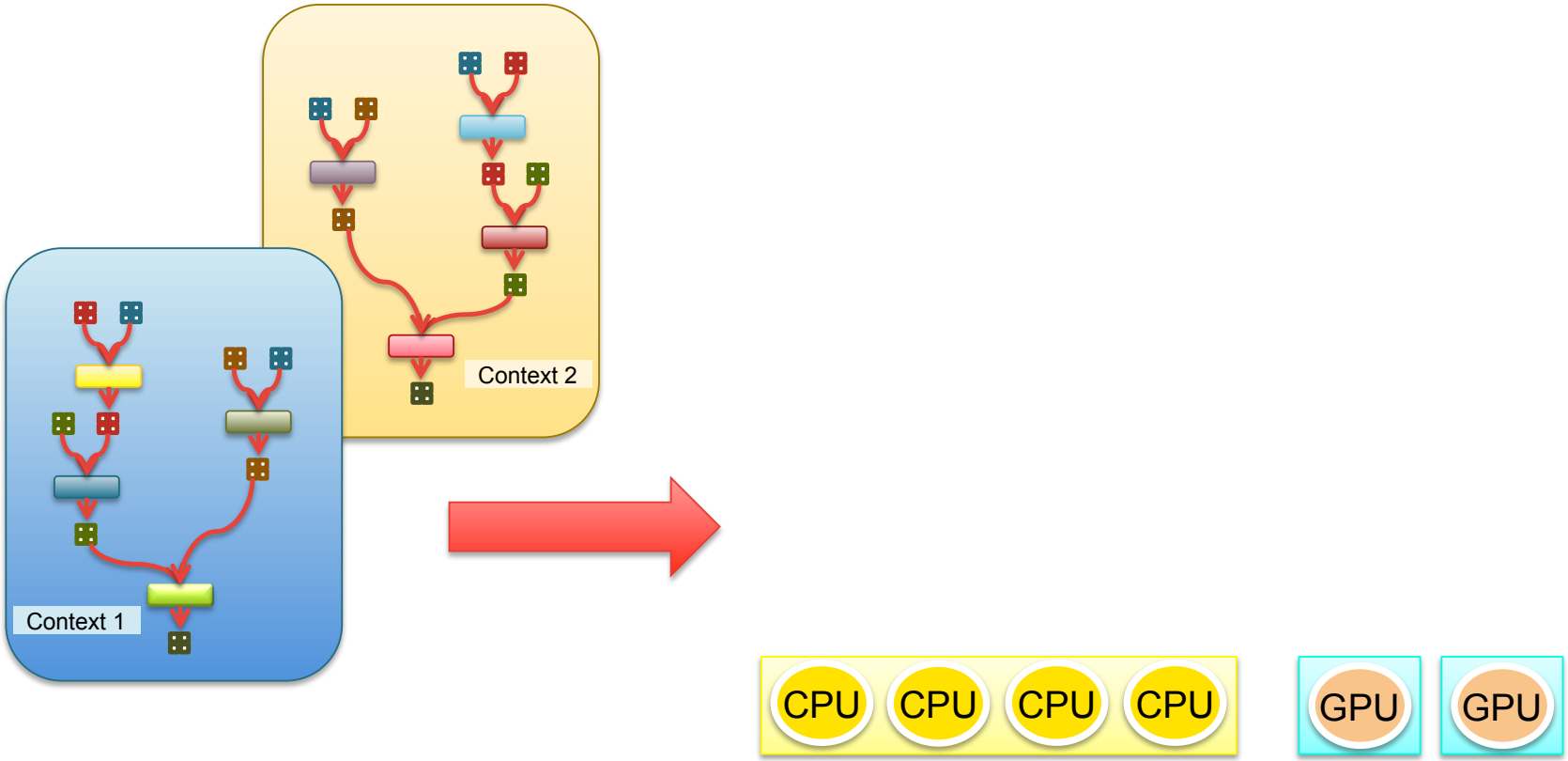
- Sharing computing resources...
- ... among multiple DAGs
- ... simultaneously

Scheduling Contexts

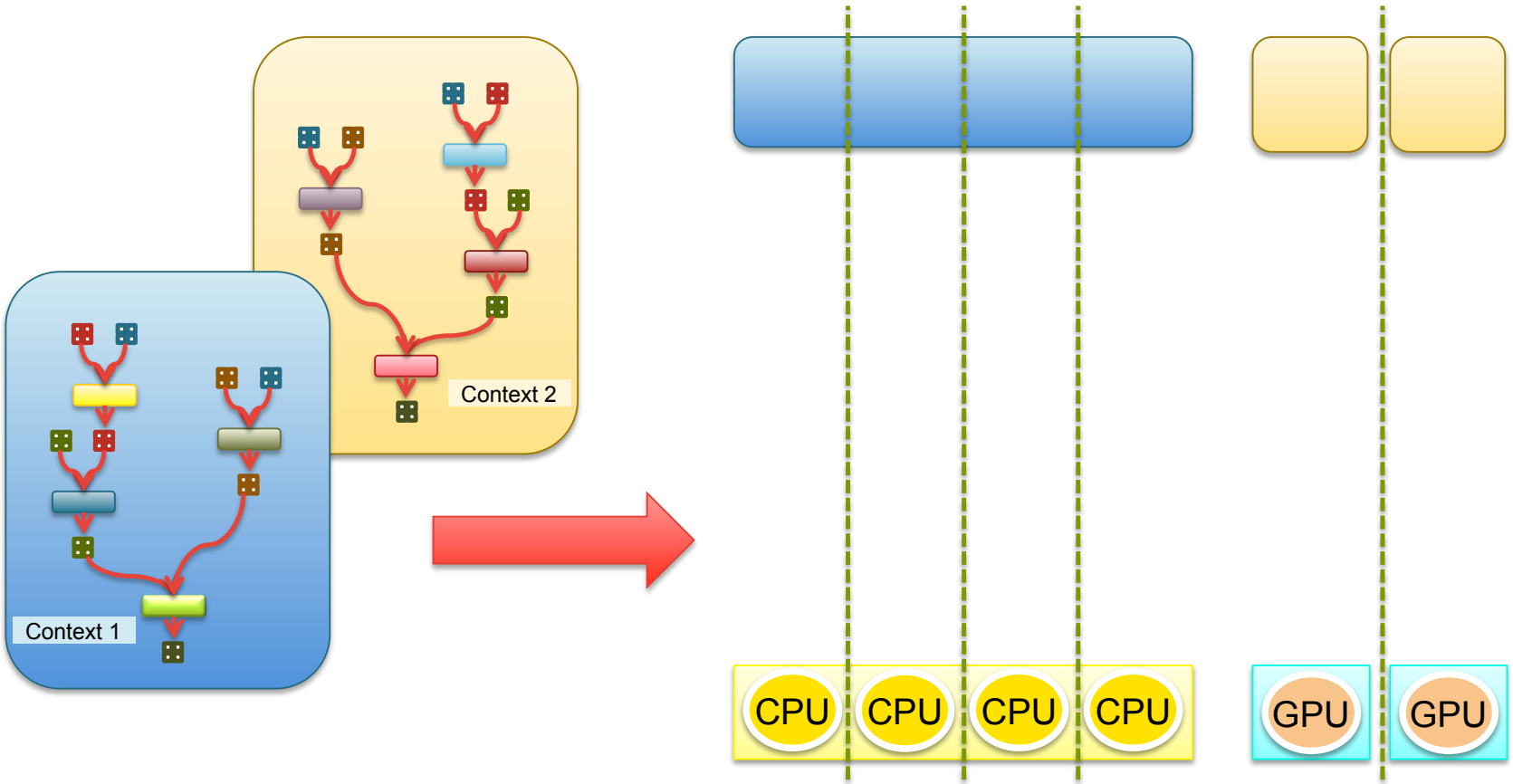
- Map DAGs on subsets of computing units
- Isolate competing kernels or library calls
  - OpenMP kernel, Intel MKL, etc.
- Select scheduling policy per context



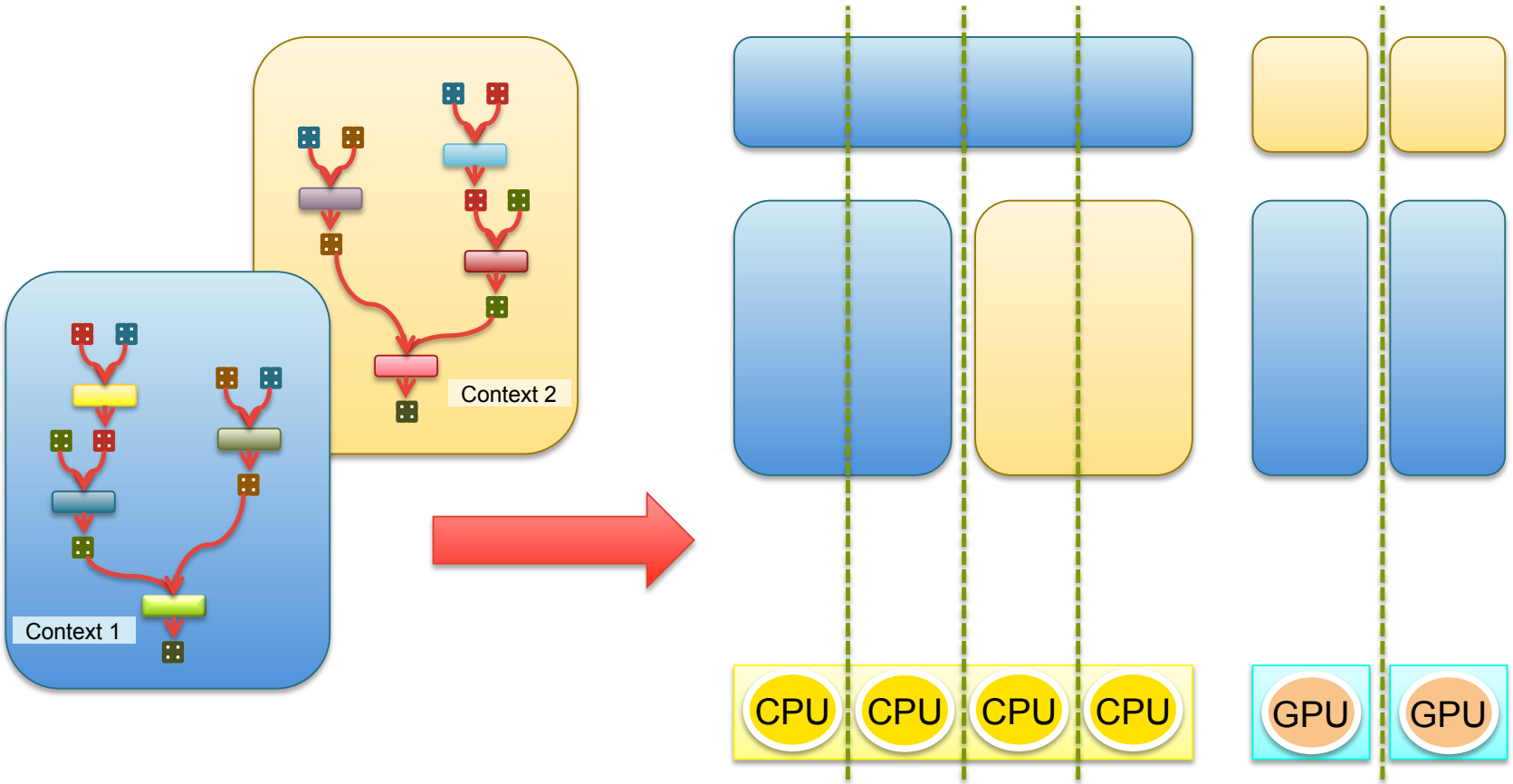
Contexts: Dynamic Resource Management



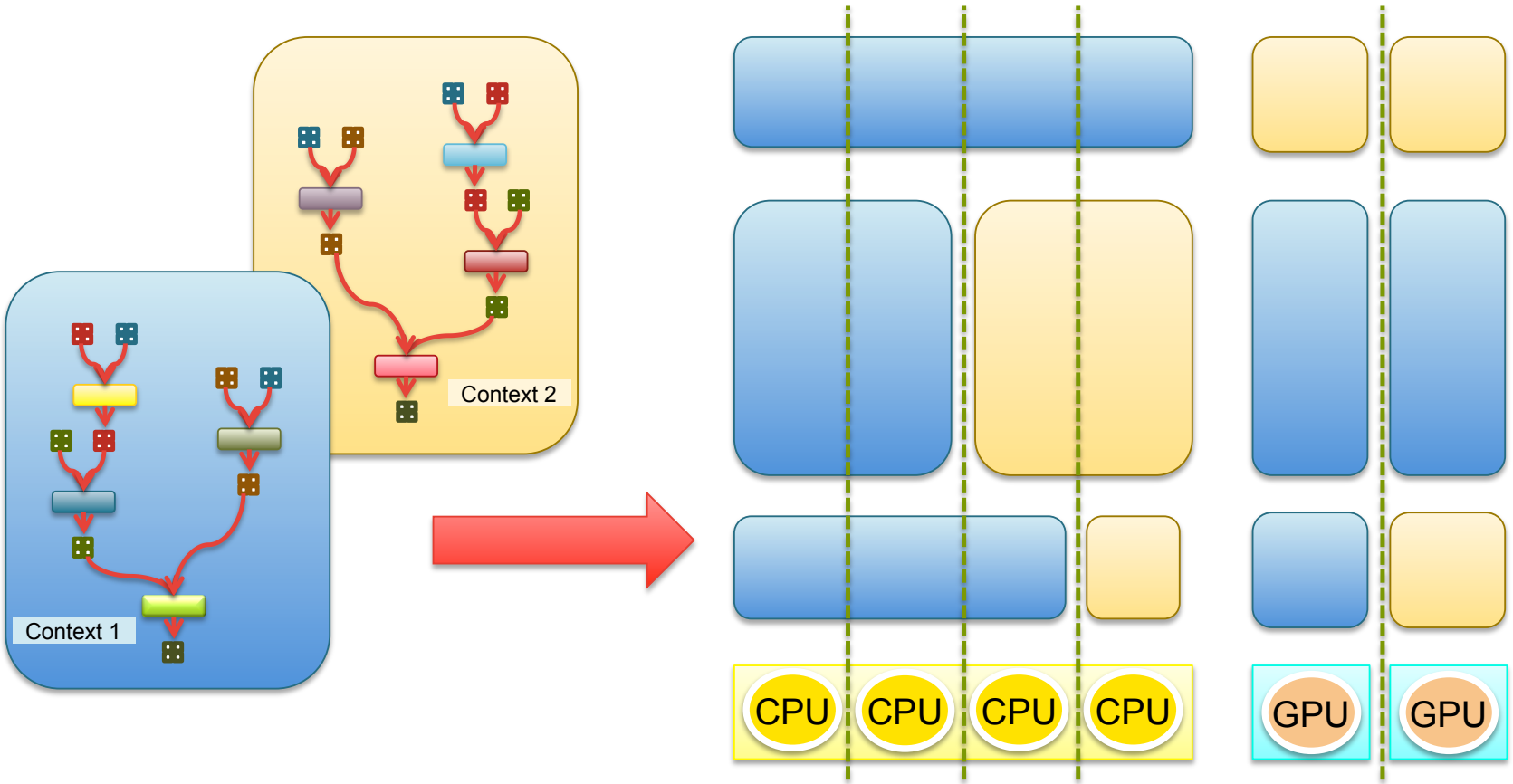
Contexts: Dynamic Resource Management



Contexts: Dynamic Resource Management



Contexts: Dynamic Resource Management



Going Even Further: Interoperability

Going Even Further: Interoperability

How to Make Runtimes, Libs Cooperate?

How to Make Runtimes, Libs Cooperate?

- Project INTERTWinE (EU H2020, 3-years, 2015-2018)
  - Task-based runtimes: StarPU, OmpSs, PaRSEC, OpenMP
  - Networking APIs: MPI, GASPI
  - Libraries: Plasma, DPlasma
  - Applications

How to Make Runtimes, Libs Cooperate?

- Project INTERTWinE (EU H2020, 3-years, 2015-2018)
  - Task-based runtimes: StarPU, OmpSs, PaRSEC, OpenMP
  - Networking APIs: MPI, GASPI
  - Libraries: Plasma, DPlasma
  - Applications
- Cooperative resource allocation and management
  - Cores
  - Accelerators
  - Memory
  - Pinned memory segments
  - ...

www.intertwine-project.eu

2.8

A Linear Algebra Solver Stack On a Task-Based Runtime System

MORSE

Matrices Over Runtime Systems @ Exascale

INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE

Inria
INVENTEURS DU MONDE NUMÉRIQUE

KAUST

University of Colorado Denver

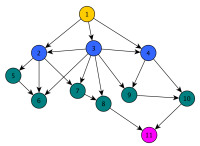
Linear algebra

$$AX = B$$

Sequential-Task-Flow

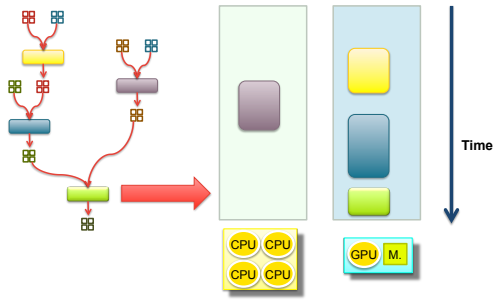
```
for (j = 0; j < N; j++)
  Task (A[j]);
```

Direct Acyclic Graph



→

Runtime systems



→

Heterogeneous platforms




Availability

MORSE associated team website:
https://www.inria.fr/en/associate-team/morse

2.9

Wrap-Up

Wrap-up: The Sequential Task Flow Programming Model

Benefits

- Automatic dependencies computation
  - Correctness
  - Productivity

Wrap-up: The Sequential Task Flow Programming Model

Benefits

- Automatic dependencies computation
  - Correctness
  - Productivity
- Unmodified sequential algorithm
  - Sequential validation/debugging
  - Long term perennialty

Wrap-up: The Sequential Task Flow Programming Model

Benefits

- Automatic dependencies computation
  - Correctness
  - Productivity
- Unmodified sequential algorithm
  - Sequential validation/debugging
  - Long term perennialty
- No forward dependencies
  - Deadlock-free by construction

Wrap-up: The Sequential Task Flow Programming Model

Benefits

- Automatic dependencies computation
  - Correctness
  - Productivity
- Unmodified sequential algorithm
  - Sequential validation/debugging
  - Long term perennialty
- No forward dependencies
  - Deadlock-free by construction
- Separation of concerns
  - Application algorithm
    - End user scientist, application-specific expert
  - Fine-tuned, hardware dependent kernels
    - Numerical methods expert and/or tool, (e.g. BLAS lib writer, BOAST kernel)
  - State-of-the-art scheduling algorithms
    - Parallel processing theory expert
  - Run-time execution management (StarPU)
    - Parallel processing practical expert

Wrap up: StarPU Status

API

- native C/C++
- native Fortran
- OpenMP C/C++

Operating System

- Linux
- MacOS
- MS Windows

Accelerators Supported

- nVidia CUDA
- OpenCL
- Intel Xeon Phi (KNC)

New Platforms

- IBM POWER8 / OpenPower
- Intel Xeon Phi (KNL)

Availability

- StarPU project website – **http://starpu.gforge.inria.fr/**
- LGPL License

Wrap up: Partnerships

- Industrial Partnerships
  - Airbus Group, CEA, Total SA, ONERA, IMACS
- **EU H2020 INTERTWinE**
  - Runtime systems interoperability
- MORSE Associated Team: Inria/UTK/UCD/Kaust
  - Linear Algebra
- Inria IPL C2S@Exa
  - Federation/integration of Inria's HPC Software
- Inria ADT KStar
  - OpenMP source-to-source compiler
- DGA RAPID Hi-BOX
  - FMM toolbox on top of StarPU
- PIA ELCI
  - Component models for task-based environments
- ANR SOLHAR
  - Sparse Linear Algebra
- ANR SONGS
  - SimGrid simulation

StarPU Overview

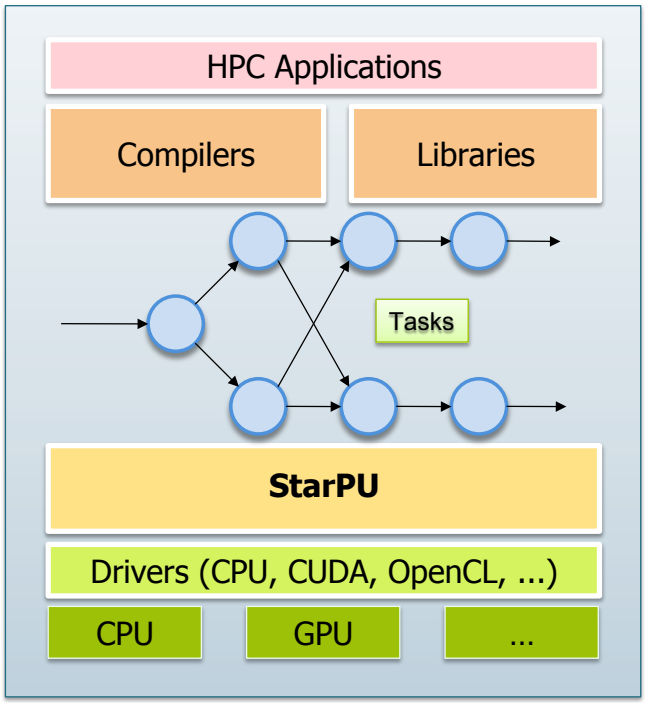
A Unified Runtime System for Heterogeneous Multicore Architectures

Programming Model: **Sequential Task Flow**
Execution Model: **Scheduler + DSM**

Key combination for:

- Portability
- Control
- Adaptiveness
- Optimization

Portability of Performance



Thanks for your attention.

StarPU runtime system

Web Site: http://starpu.gforge.inria.fr/

LGPL License

KStar OpenMP compiler

Web Site: http://kstar.gforge.inria.fr/

CeCILL-C License

Open to external contributors

Thanks for your attention.

StarPU runtime system

Web Site: http://starpu.gforge.inria.fr/

LGPL License

KStar OpenMP compiler

Web Site: http://kstar.gforge.inria.fr/

CeCILL-C License

Open to external contributors