Grid Computing
introduction & illustration

T. Gautier
V. Danjean

SMAI 26 mai 2009

Facts

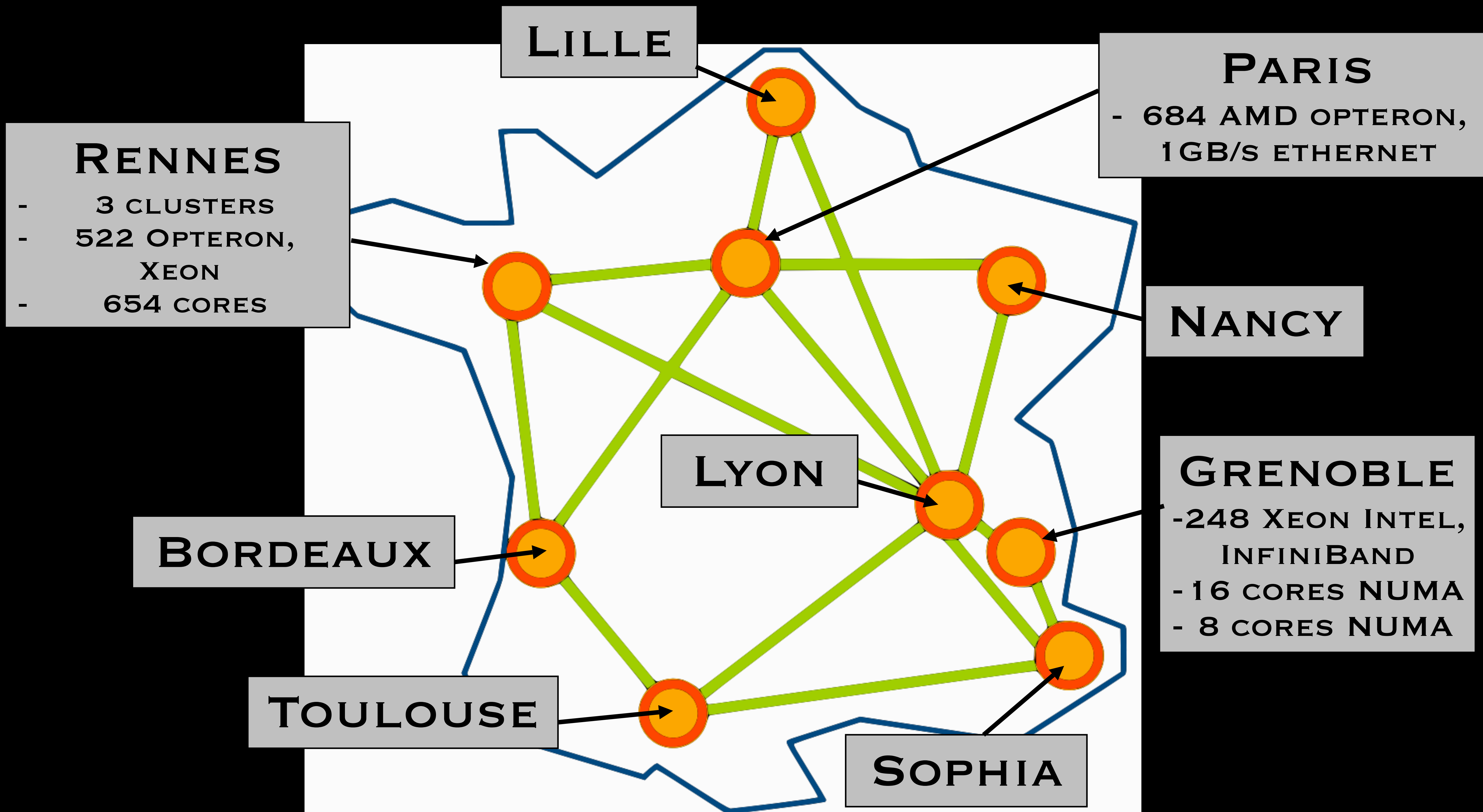
- No choice : parallelism is in any computer
  - MPSoC, Multicore, Manycore, Cluster, Grid
- Exact Solution to the Quadratic Assignment Problem
  - Combinatorial Problem: NUG30, 7 days, 650 processors
- Solving bigger CFD, CEM problems
  - 100GBytes of memory

➡ Cluster & Grid computing

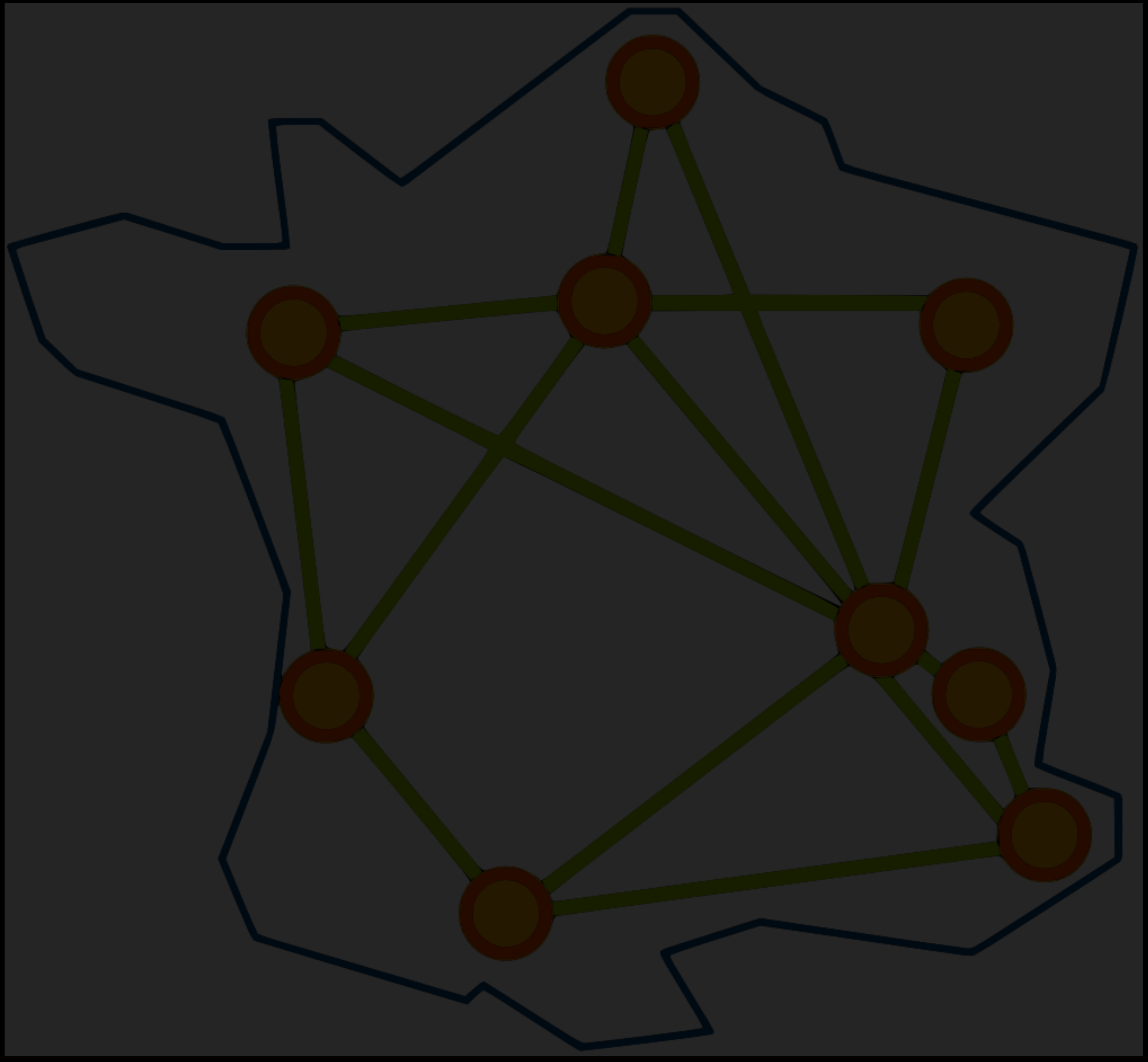
Outline

- Context
  - Parallel and distributed architecture
- Programming Challenges
  - Parallel algorithm
  - Scheduling & Communication
- Illustration with domain decomposition method
- Conclusions

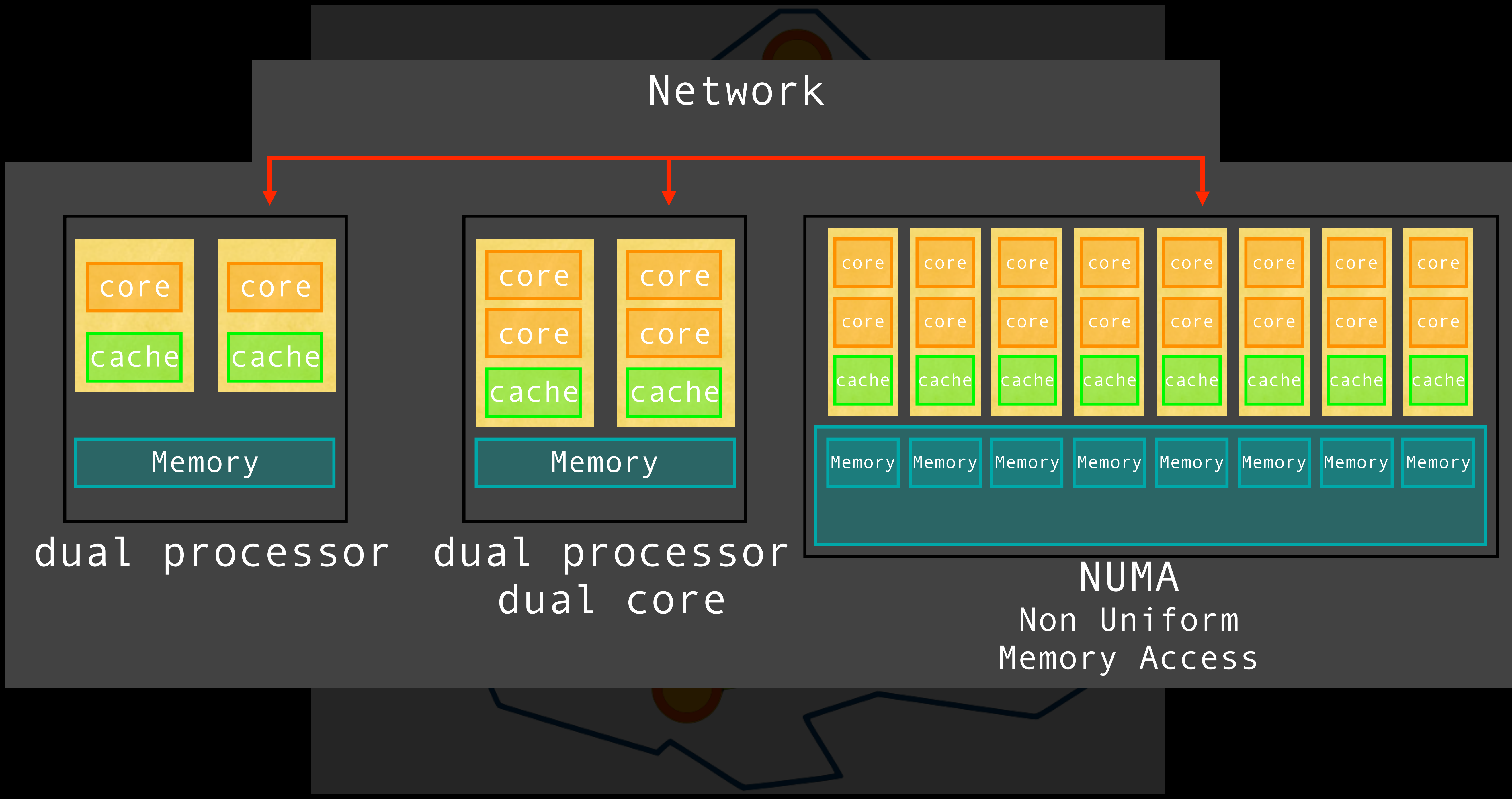
www.grid5000.fr



Grid5000



Grid5000


Network
core
core
cache
cache
Memory
dual processor
core
core
core
core
cache
cache
Memory
dual processor
dual core
core core core core core core core core
core core core core core core core core
cache cache cache cache cache cache cache cache
Memory Memory Memory Memory Memory Memory Memory Memory
NUMA
Non Uniform
Memory Access

Characterisics

- Cluster: set of homogeneous machines
  - homogeneous resources (memory, cpu)
    - Dual, Quad, Octo cores cpu, GBytes / machine
  - homogeneous & high performance network
    - Ethernet, Myrinet, InfiniBand
  - homogeneous administration domain
    - 1 user = 1 account, home directory mounted using NFS

Characterisics

- Grid: set of clusters
  - heterogeneous resources
    - CPU, memory, network speed
    - Each cluster has different number of machines
  - Network between clusters: high latency !
  - multiple administration domains
    - 1 user = multiple account
    - access to cluster through firewall
  - dependability problem
    - huge number of basic components

Programming Challenges

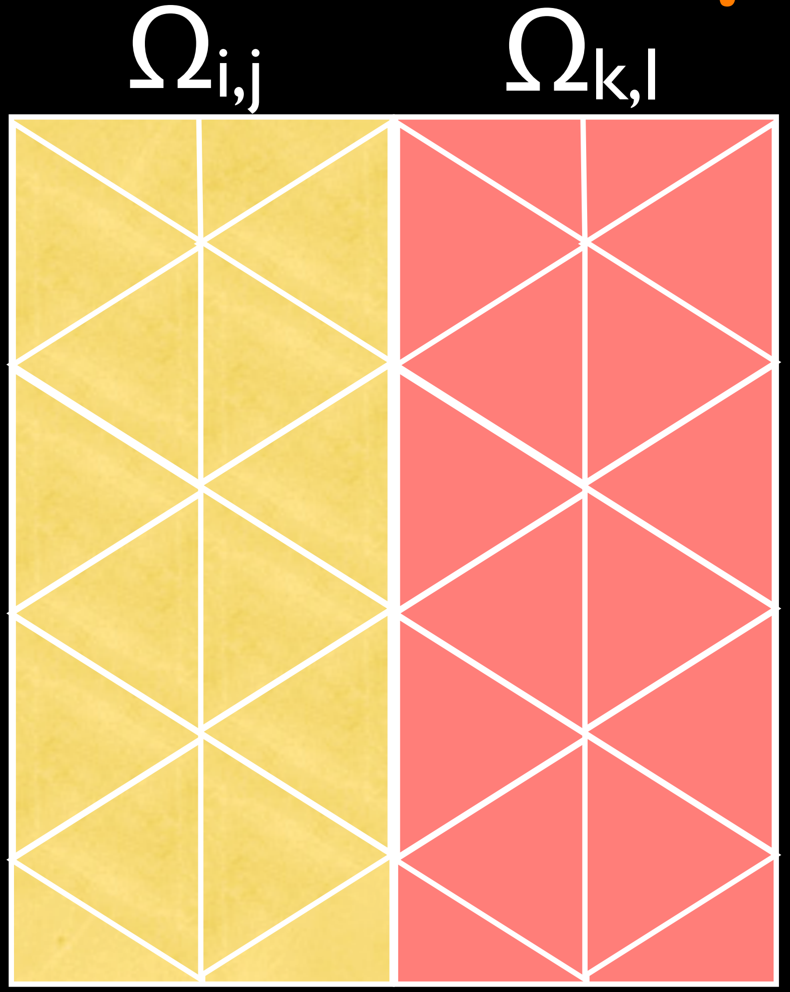
- Write once, run anyware
  - heterogeneity !
- Keypoints
  - parallel algorithm
  - scheduling
  - implementation
  - [fault tolerance]

Parallel algorithm

- 30 years of theoretical studies & experiments of parallel architectures
- What’s new ?
  - huge number of cores/CPU
  - fault tolerance
    - ABFT: Algorithm Based Fault Tolerant

Let's back to a problem

- **Domain decomposition for matrix vector product**

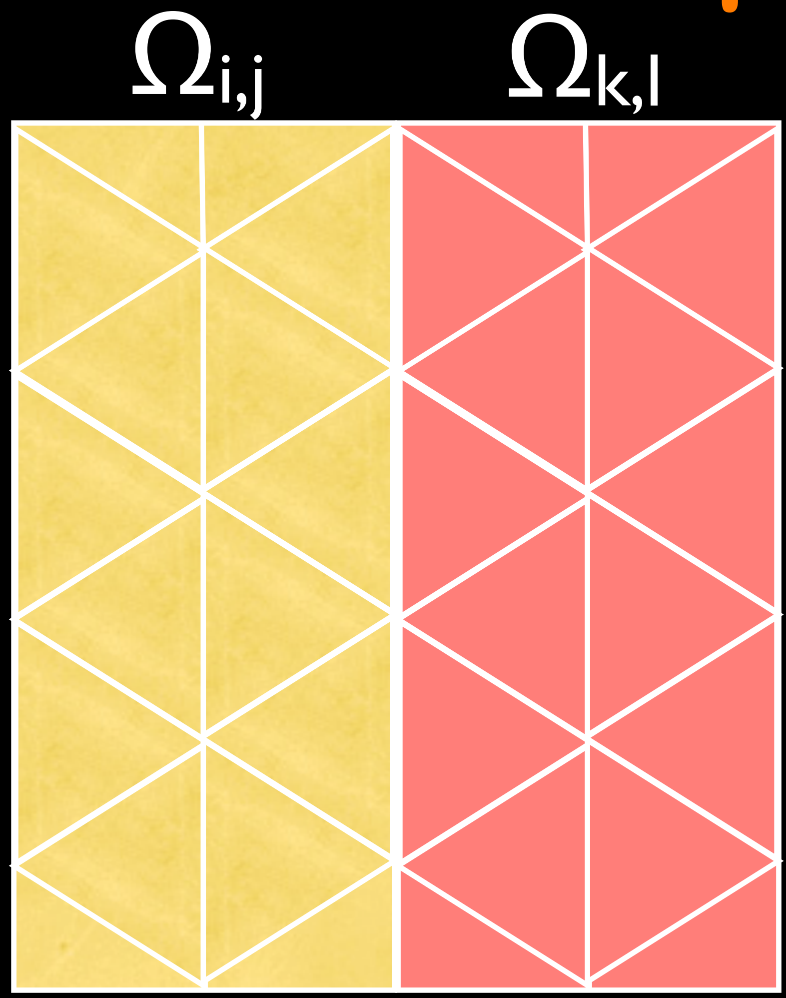


$$(Ax)_{i,j} = \begin{pmatrix} B_{i,j} & E_{i,j} \\ F_{i,j} & C_{i,j} \end{pmatrix} \begin{pmatrix} u_{i,j} \\ v_{i,j} \end{pmatrix} + \begin{pmatrix} 0 \\ \sum_{\Omega_{k,l} \in N_{i,j}} E_{i,j}^{k,l} v_{k,l} \end{pmatrix}$$

ANR DISCOGRID, coordinateur S. Lanteri

Let’s back to a problem

- **Domain decomposition for matrix vector product**



$$(Ax)_{i,j} = \begin{pmatrix} B_{i,j} & E_{i,j} \\ F_{i,j} & C_{i,j} \end{pmatrix} \begin{pmatrix} u_{i,j} \\ v_{i,j} \end{pmatrix} + \begin{pmatrix} 0 \\ \sum_{\Omega_{k,l} \in N_{i,j}} E_{i,j}^{k,l} v_{k,l} \end{pmatrix}$$

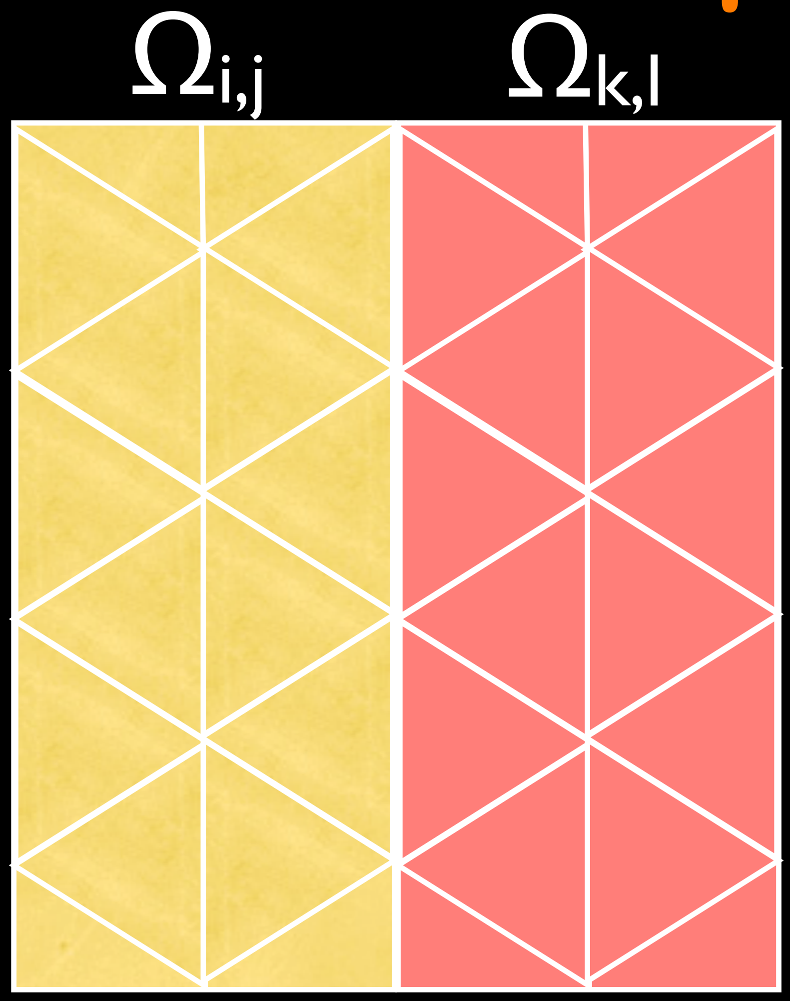
local internal values

local interface values

ANR DISCOGRID, coordinateur S. Lanteri

Let’s back to a problem

- **Domain decomposition for matrix vector product**



external interface values

$$(Ax)_{i,j} = \begin{pmatrix} B_{i,j} & E_{i,j} \\ F_{i,j} & C_{i,j} \end{pmatrix} \begin{pmatrix} u_{i,j} \\ v_{i,j} \end{pmatrix} + \begin{pmatrix} 0 \\ \sum_{\Omega_{k,l} \in N_{i,j}} E_{i,j}^{k,l} v_{k,l} \end{pmatrix}$$

local internal values

local interface values

ANR DISCOGRID, coordinateur S. Lanteri

Scheduling

- Once tasks and data are described by a parallel algorithm then:
  - For each task, compute where to execute it
  - For each data, compute where to store it
- Such that:
  - Completion time is minimize, ...
- NP-hard problem
  - Use algorithms to approximate the problem
  - Use heuristics, application dependent

Strict multi-threaded computations

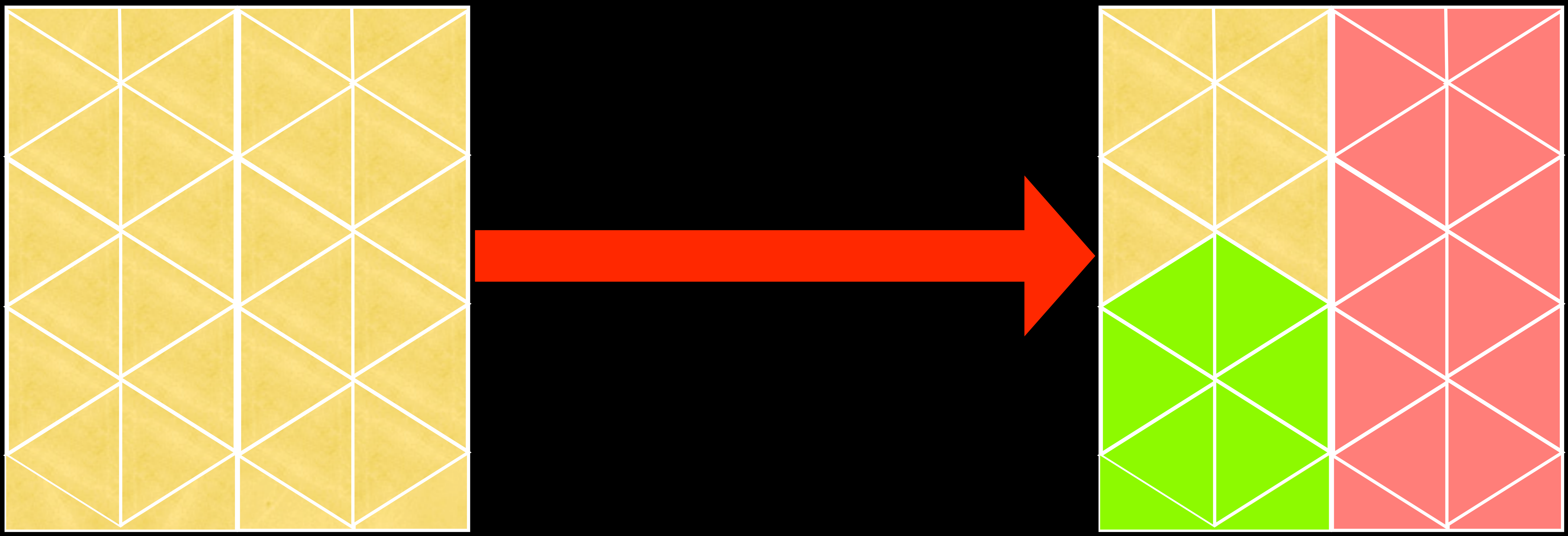
- **Notations**
  - T_s : Sequential work, time of sequential execution
  - D : Critical Path
  - P: the P processors
- **Properties**
  - with high probability, number of steals is

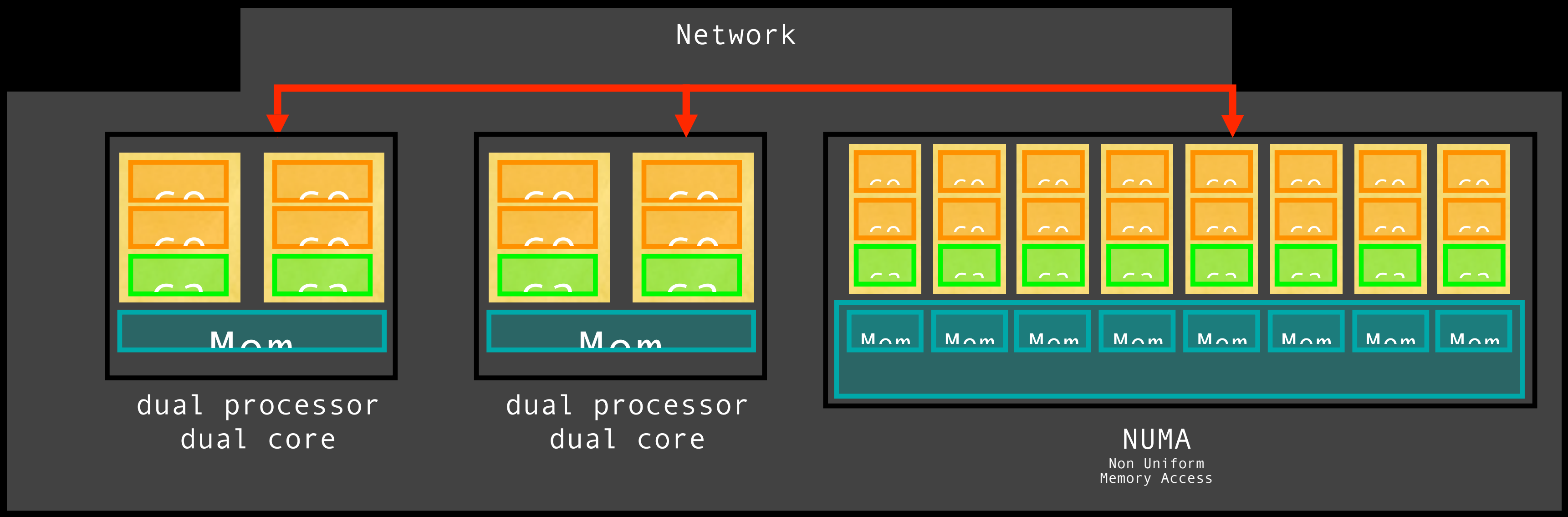
$$O(P \times D)$$

  - with high probability, execution time is

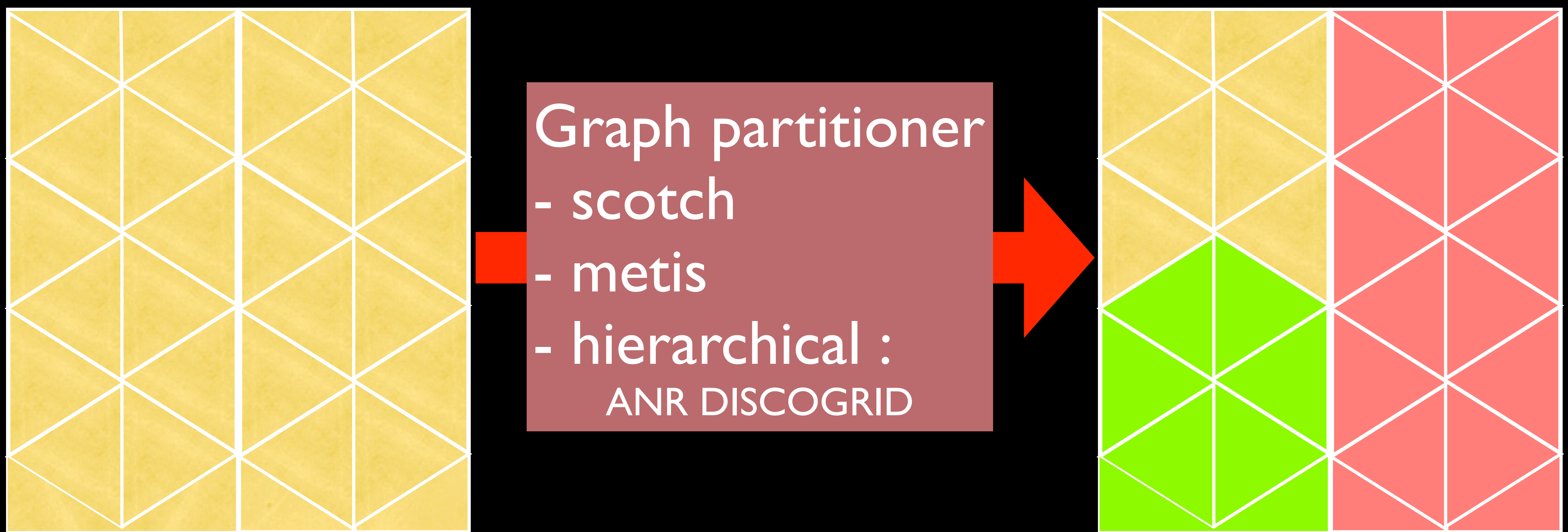
$$T_p \leq T_1 / P + O(D)$$

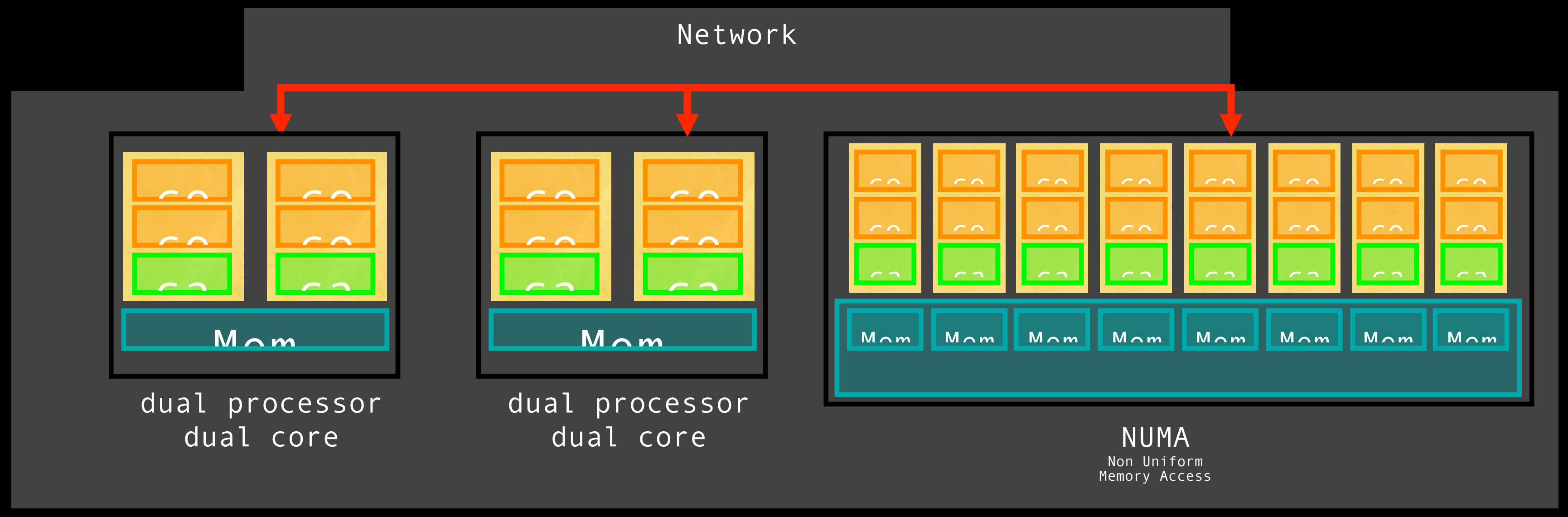
Domain decomposition





Domain decomposition





That's all ?

- Domain decomposition
  - Data = sub domain, mapped onto machines
  - Task : mapped using "owner compute rule"
- Program : one process per subdomain

```
while (error < epsilon)
{
    exchange interface
    do computation inside subdomain
    compute error
}
```

That's all ?

- Domain decomposition
  - Data = sub domain, mapped onto machines
  - Task : mapped using "owner compute rule"
- Program : one process per subdomain

```
while (error < epsilon)
{
    exchange interface
    do computation inside subdomain
    compute error
}
```

Communication between neighbors

That's all ?

- **Domain decomposition**
  - **Data = sub domain, mapped onto machines**
  - **Task : mapped using "owner compute rule"**
- **Program : one process per subdomain**

```
while (error < epsilon)
{
    exchange interface
    do computation inside subdomain
    compute error
}
```

Communication between neighbors

Global reduction communication

Improvement

- **Assume that emission & reception of message are concurrent with local computation**

```
while (error < epsilon)
{
    begin send message to my neighbors
    do internal computation
    wait until all messages have been received
    update internal computation
    compute error
}
```

Improvement

- **Assume that emission & reception of message are concurrent with local computation**

```
while (error < epsilon)
{
    begin send message to my neighbors
    do internal computation
    wait until all messages have been received
    update internal computation
    compute error
}
```

may overlap some delay of communication

How to program

- MPI, standard but low level API
  - scheduling and mapping should be coded
  - bad overlapping of communication by computation (at least in public domain implementation)
  - bad support multi-threaded computations
  - bad support for inter-cluster communication
- Research languages
  - UPC, Titanium, X10, Fortress
  - Our language: Athapascan (API) with Kaapi
    - AUTOMATIC SCHEDULING : http://moais.imag.fr

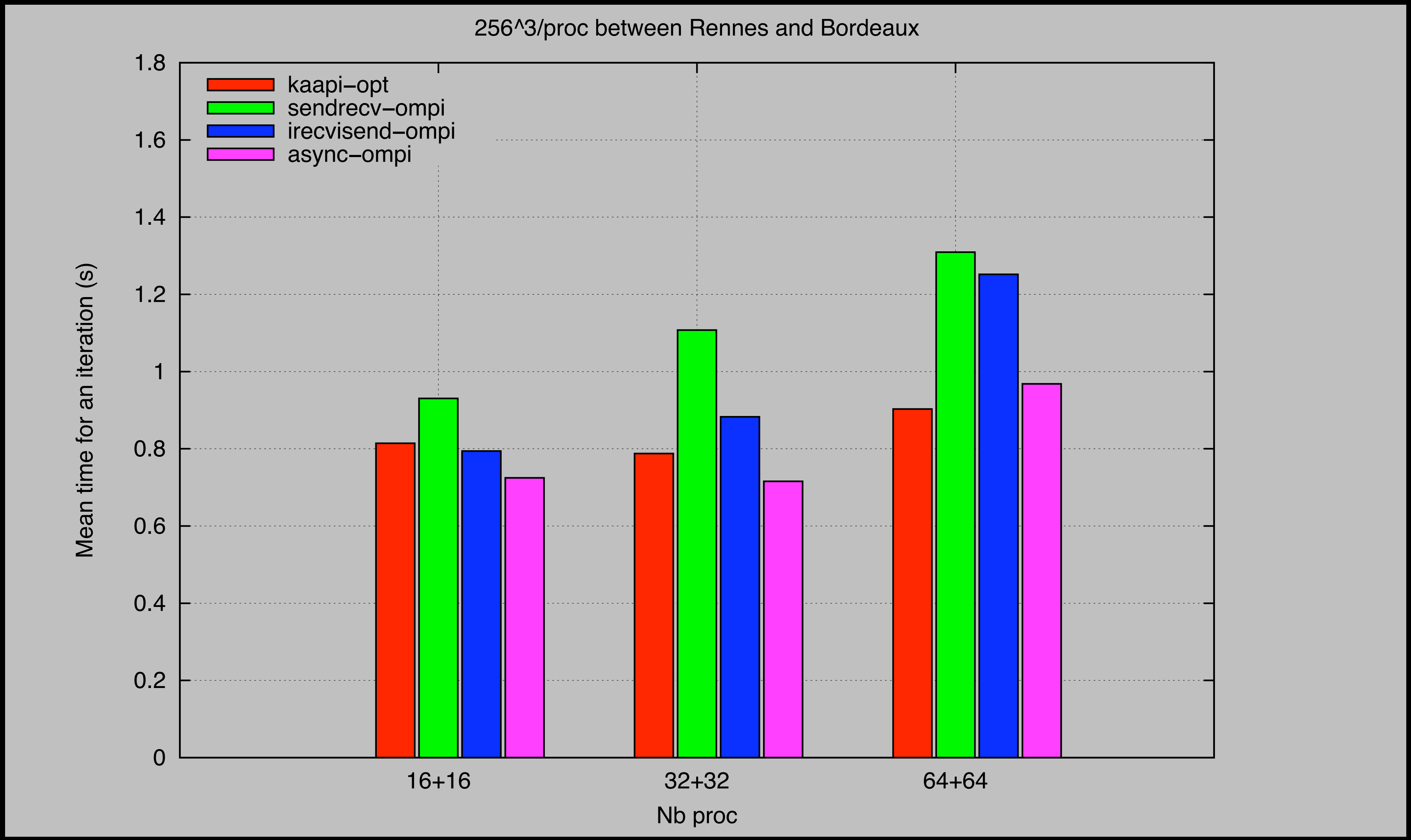
Experiments

- **Code**
  - Kaapi / C++ code versus Fortran MPI code
- **Platform**
  - Cluster : N processors on a cluster
  - Grid : N/4 processors per cluster, 4 clusters

D=256^3	# processors	Cluster (s)	Grid (s)	Overhead
KAAPI	1	0.49	0.49	-
	64	0.55	0.84	0,53
	128	0.65	0.91	0,4
MPI	1	0.44	0.44	-
	64	0.66	2.02	2,06
	128	0.68	1.57	1,31

Optimized Poisson 3D

- **Fortran code with non-blocking IO**
  - MPI_ISend, MPI_IRecv + MPI_Wait_all
  - Overlapping of communication by computation



Conclusion

- **SCHEDULING**
- **but also compilation, grid-reservation, parallel launching, runtime environment, firewall management, ...**
- **More references**

- Herlihy, M. and Shavit, N. The Art of Multiprocessor Programming, ISBN 0123705916. Morgan Kaufmann Publishing, 2008.
- **Foster I., Kesselman C. The GRID 2: Blueprint for a New Computing Infrastructure, ISBN 1558609334. Morgan Kauffman Publishing, 2 edition, 1999.**
- **Petascale Computing: Algorithms and Applications, D. Bader (Editor), ISBN 1584889098, Chapman & Hall/CRC, 2007**
- **Parallel Algorithms and Cluster Computing: Implementations, Algorithms and Applications. Karl Heinz Hoffmann (Editor), Arnd Meyer (Editor), ISBN 3540335390, Springer, 2006.**