OS impact on performance

Sébastien Valat

PhD student

CEA, DAM, DIF, F-91297, Arpajon, France

Advisor : William Jalby

CEA supervisor : Marc Pérache

Plan

- **Remind goal of OS**
- **OS impact on performance**
- **Reproducibility**
- **Conclusion**

OS : between applications and hardware

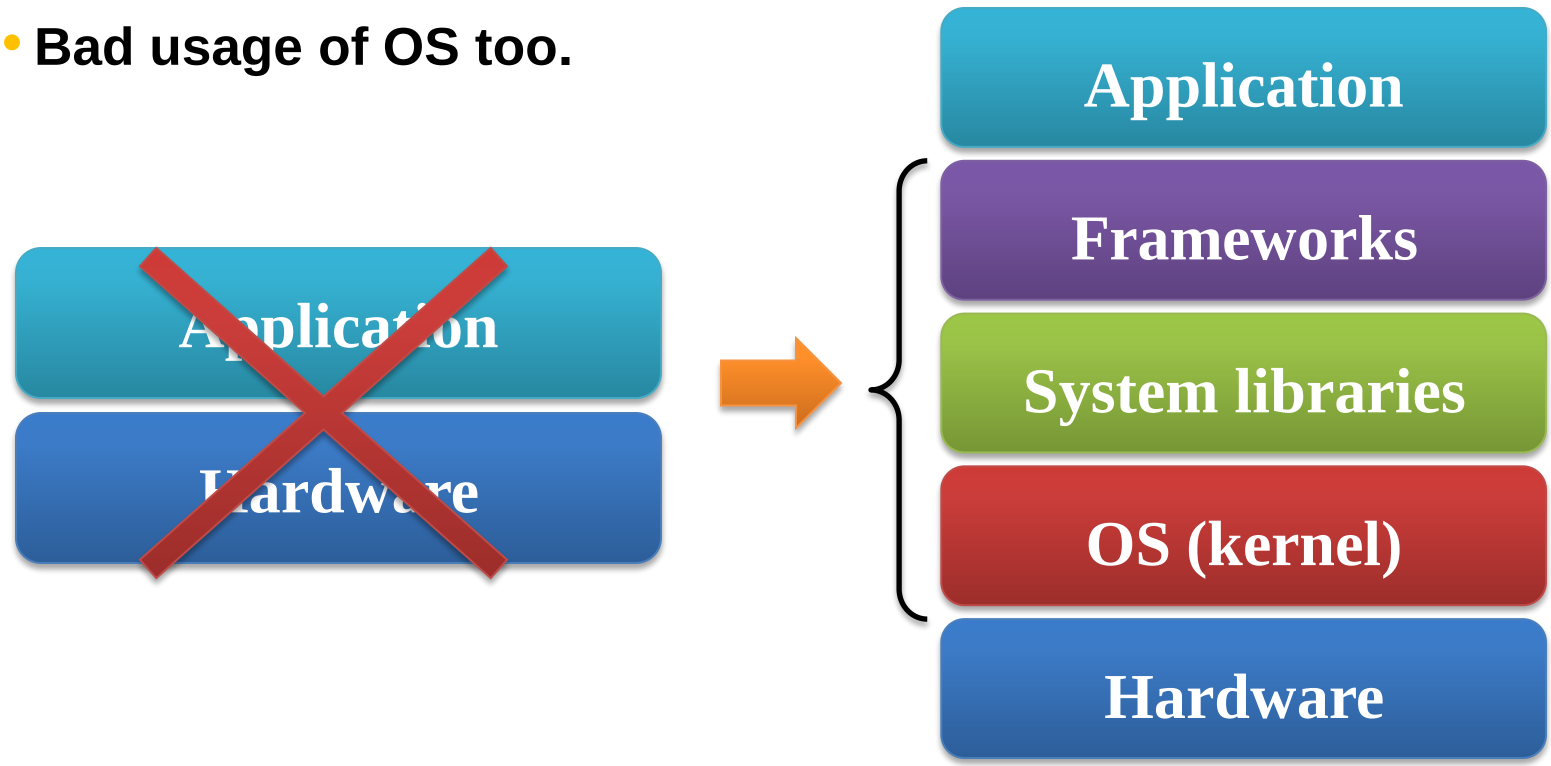
Hardware and software stack

- **To get performance we need to optimize interactions between all components.**
- **Non optimal hardware usage lead to slow down,**

Hardware and software stack

energie atomique • energies alternatives

- **To get performance we need to optimize interactions between all components.**
- **Non optimal hardware usage lead to slow down,**
- **We didn't be in direct contact to the hardware.**
- **Bad usage of OS too.**



What OS do…

- **Provide an abstract interface to interact with hardware**
- **Manage shared resources**
  - Memory
  - CPU time
  - Devices
  - File systems
  - Networks
- **Task isolation**
  - Security
  - Stability

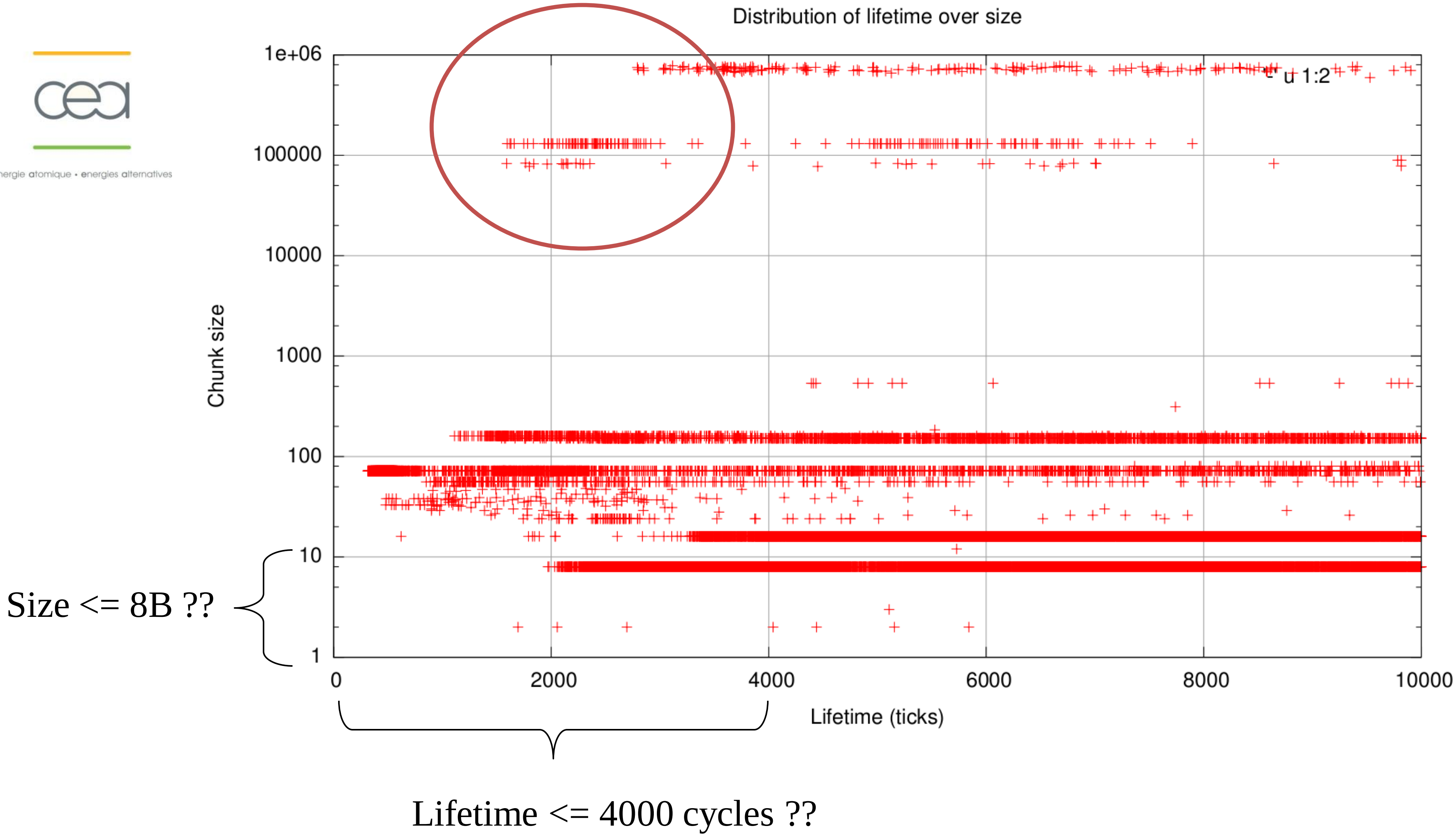
What it can’t do …

- **But it can’t prevent us from erroneous usage in term of performance.**

- **Some examples :**
  - Calling too many times IO functions.
  - Opening thousand files at same time.
  - Doing millions of memory allocation/free in loops.
  - Doing too much small allocations.

- **ASM code can be optimized, but OS can break benefits.**

Example : lifetime per size of a large C++ simulation



Complexity

- **Hardware become more complex :**
  - Multi-core / many-core
  - NUMA
  - Heterogeneous architecture
  - Multi-level caches, shared caches.
- **So the OS too**
- **But it can't mask all of it.**
- **Programmer need to be aware of this new complexity :**
  - Impact out of system call sections ?
  - Performance reproducibility ?

Do not neglect the OS

Bad component interactions

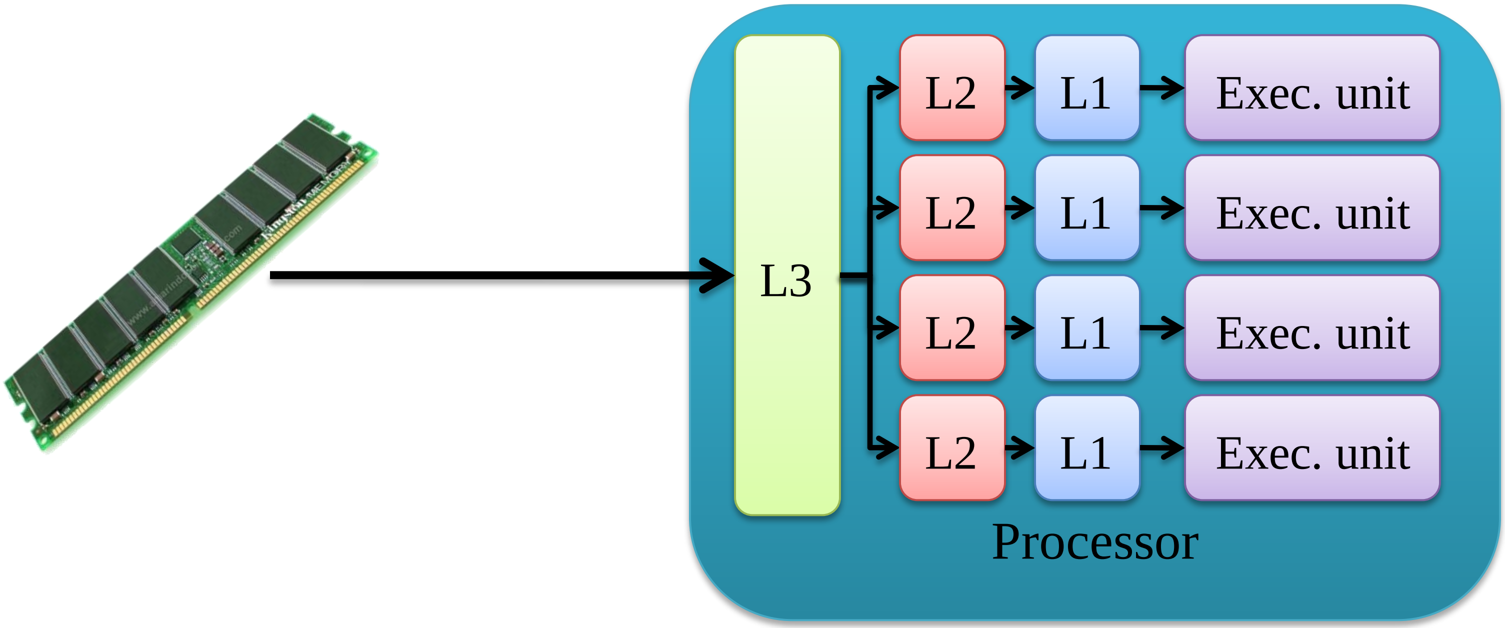
- **Some times we can found bad interactions between :**
  - Hardware
  - OS
  - Application

- **Even out of system calls.**

- **Three examples from memory :**
  - Multiple small allocations and caches
  - Cache leak
  - NUMA memory placement

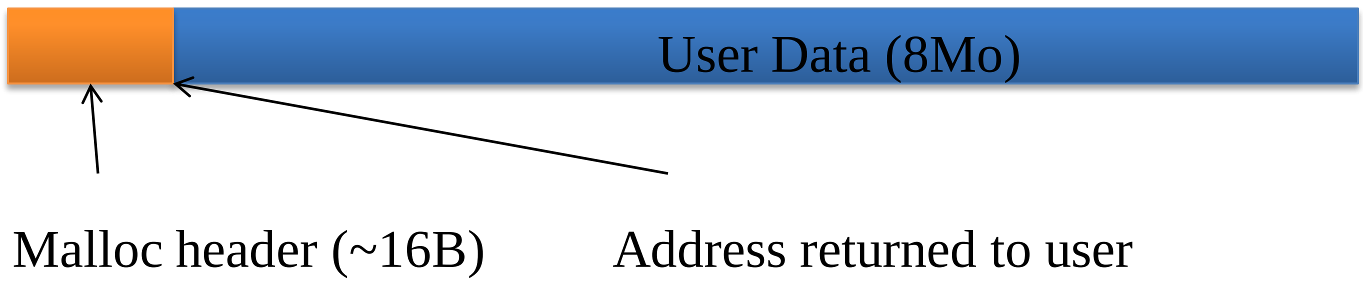
Hardware processor caches

- **Goal : hide memory access latencies**
- **Solution : add intermediate memory in processors**
- **Constraints : fast, but small.**
- **All exchanges are done with 64 bytes words**

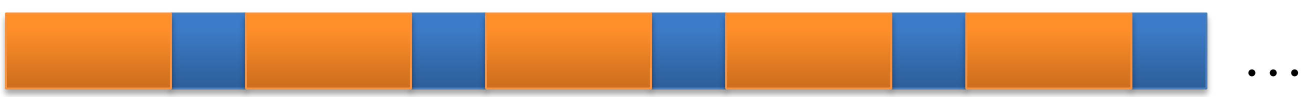


Malloc(8M) ≠ 1M * Malloc(8B)

- **Allocators tend to add headers before the return segments :**



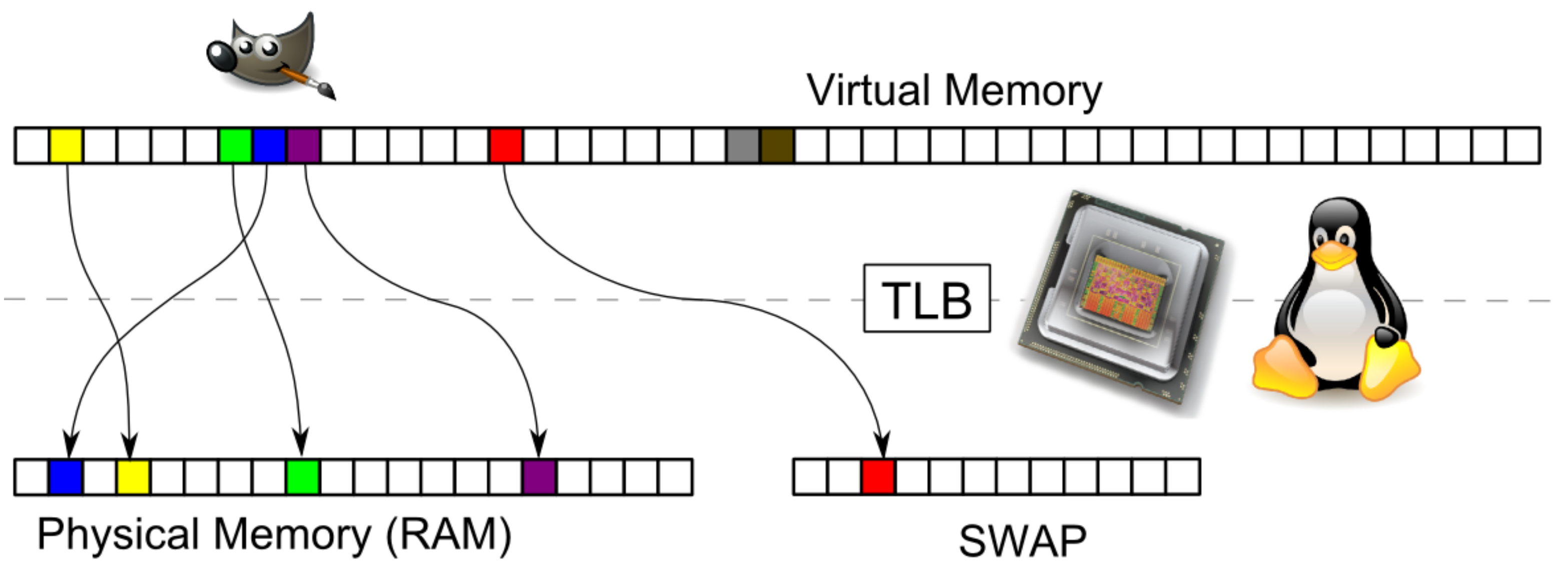
- **Total : 8Mo + 16B**
  - can fit in core i7 L3 cache.
- **1M * Malloc(8B) :**



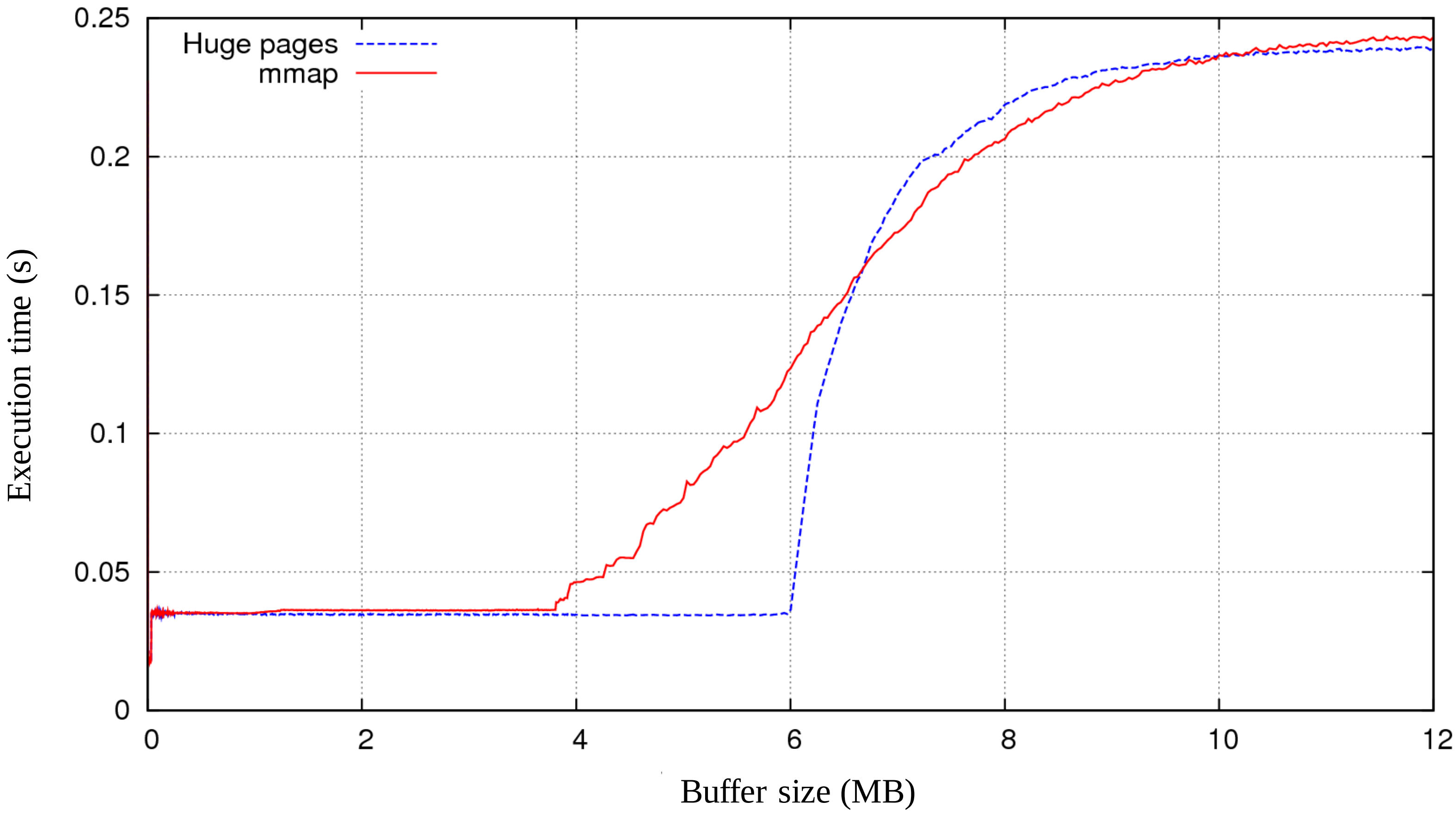
- **Total : 8Mo + 1M*16 = 24Mo**
  - Not fit anymore : 64B cache lines will fetch the header too.
- **Take care of the memory layout of your data for caches.**

OS : Memory paging

- **Physical + Virtual memory address space.**
  - Provide larger address space than physically
  - Isolate processes
  - Permit to support disk paging (swap).
- **Split the memory in blocs of 4 KB (pages) :**

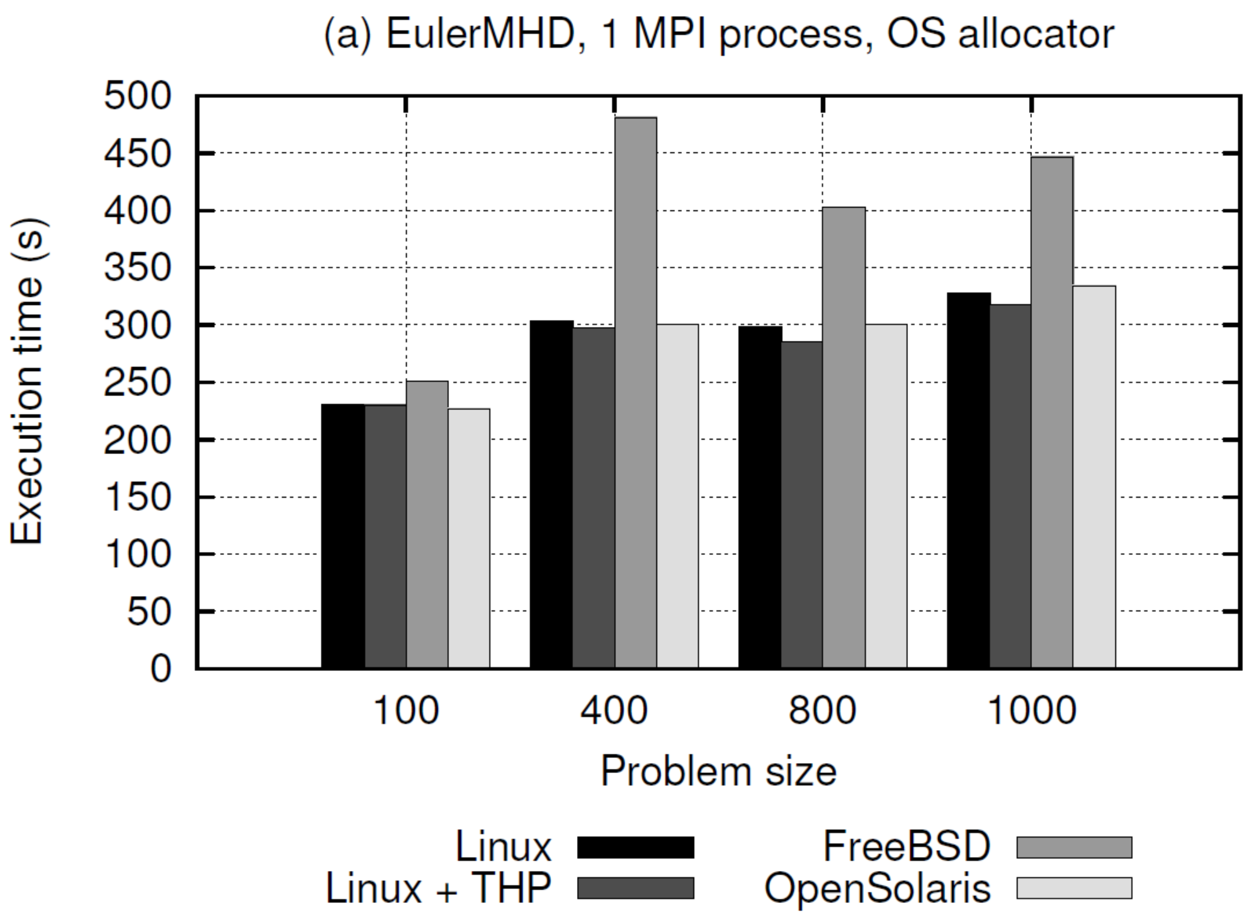


- **On Linux, due to random paging :**



Example of bad interaction between components

- **Bad interaction : application / malloc / OS paging**
- **OS policy can have large impact in some cases.**



Missing the major point

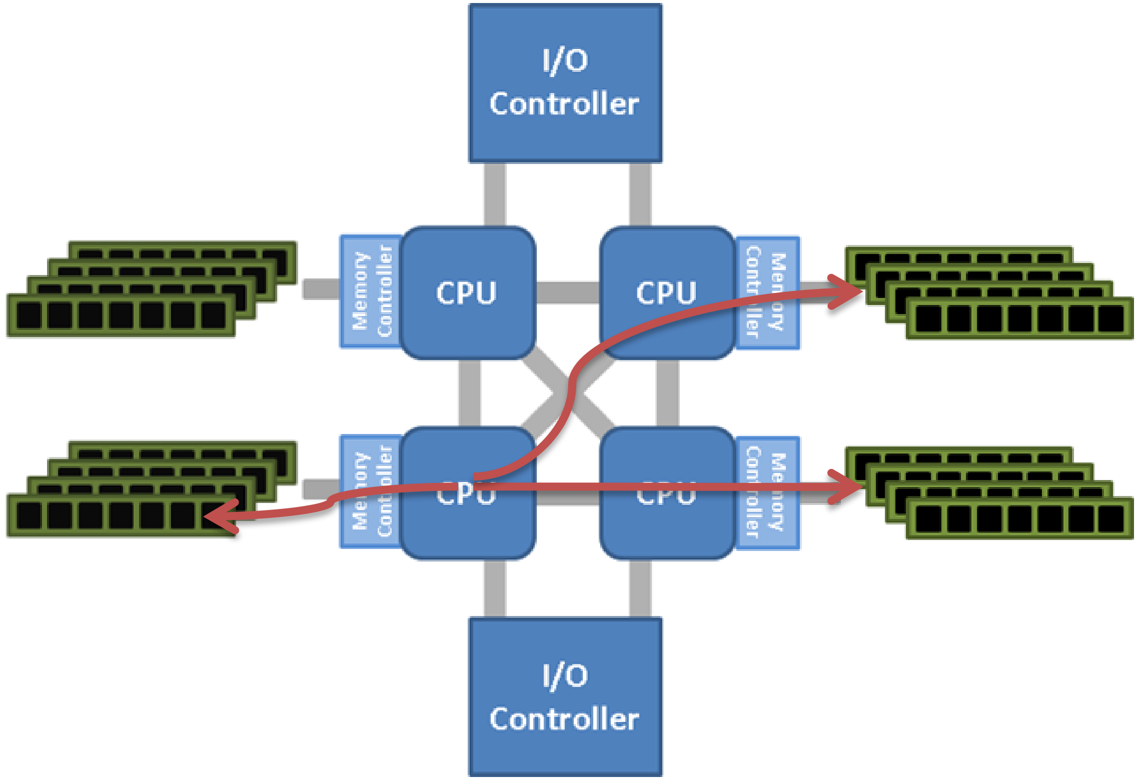
- **Optimizing a matrix vector product**
- **First option : optimize the code**
  - Some manual unrolling
  - Vectorization
  - ...
- **Second option : change the allocation pattern**
  - +16 on base address on one array.

(In cycles / iteration)	Original	Optimized
Default allocation	51.0	21.7
Padded allocation	21.8	21.3

- **In this case, the major issue is a bad system decision.**

NUMA allocations ?

- NUMA : Non Uniform Memory Access.
- Each processor has its own memory.
- Accessing to memory of other processors is slower.
- Where the OS allocate your data ?



Example : NUMA allocation

- **Allocate A,B and C such as (SIZE = 128M) :**

```
double * A = malloc(SIZE);
double * B = malloc(SIZE);
double * C = malloc(SIZE);
```

- **Init to 0 :**

```
memset(A,0,SIZE);
memset(B,0,SIZE);
memset(C,0,SIZE);
```

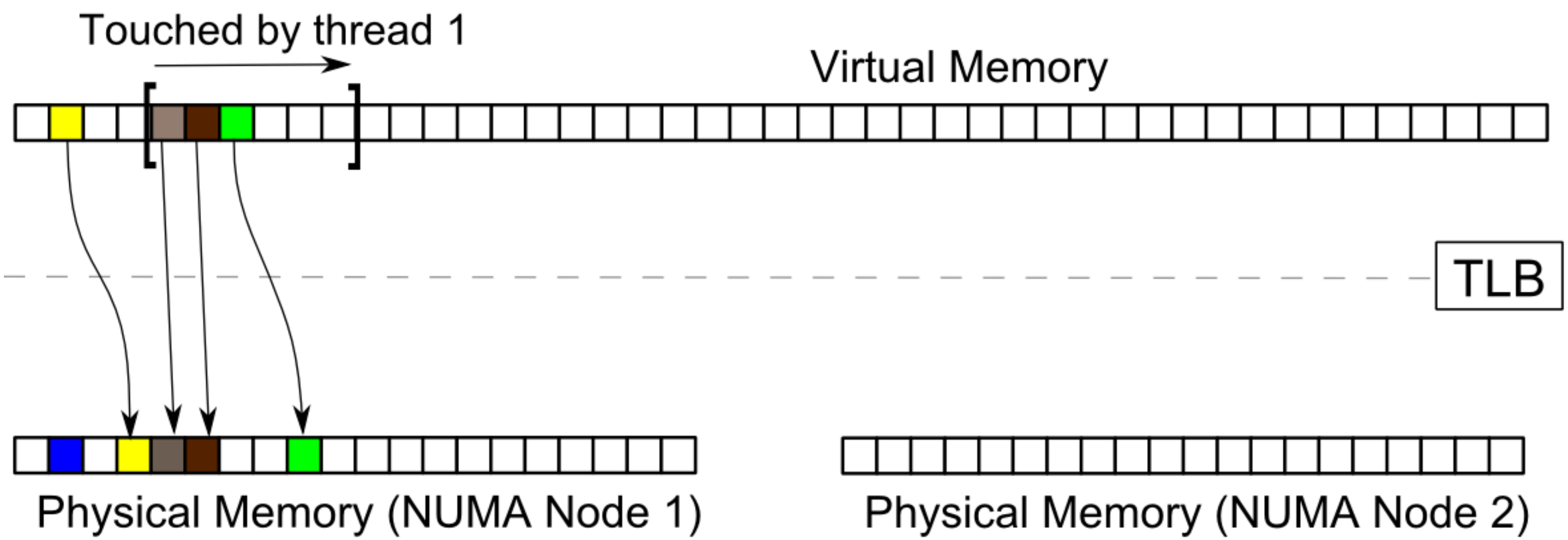
- **Measure execution time on 8 threads, 2 NUMA nodes :**

```
for ( rep = 0 ; rep < 500 ; rep++)
   #pragma omp parallel for private(i)
   for ( i = 0 ; i < SIZE / sizeof(double) ; i++)
       A[i] = B[i] + C[i];
```

- **What is the performance mistake ?**

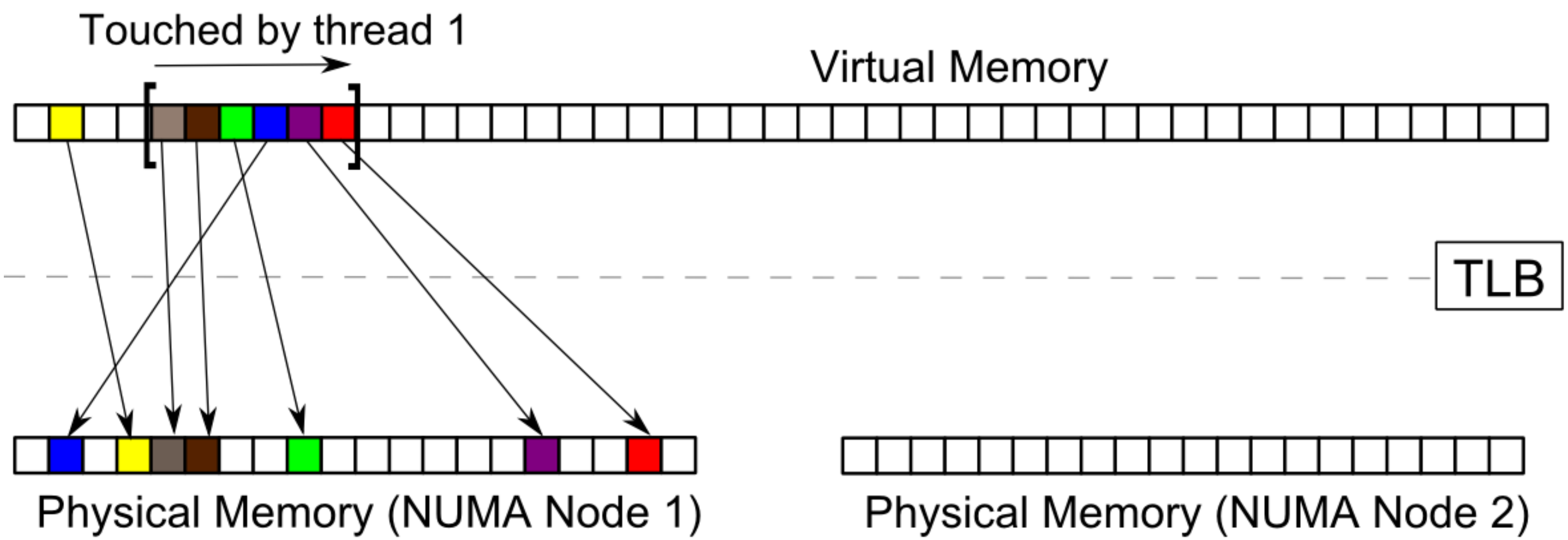
First touch mapping

- **Thread 1 (on NUMA node 1) do malloc**
- **Then call memset, so access to the memory**



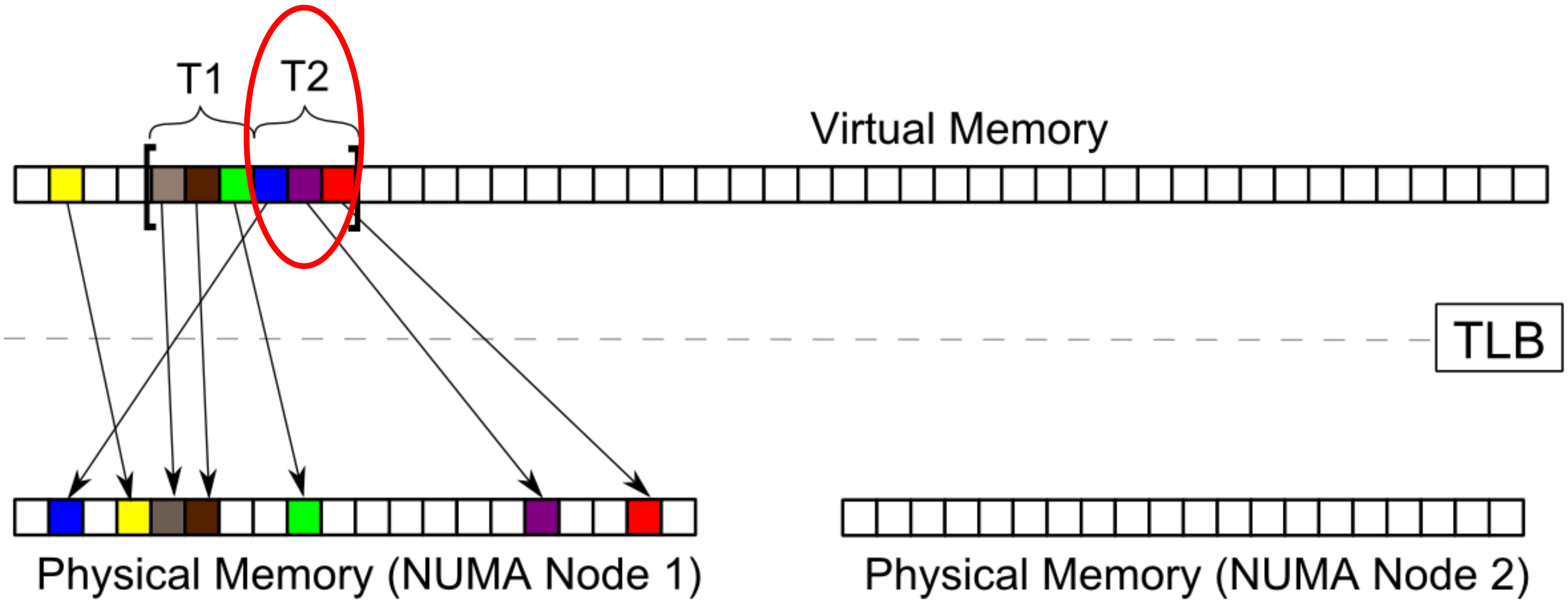
First touch mapping

- **Thread 1 (on NUMA node 1) do malloc**
- **Then call memset, so access to the memory**



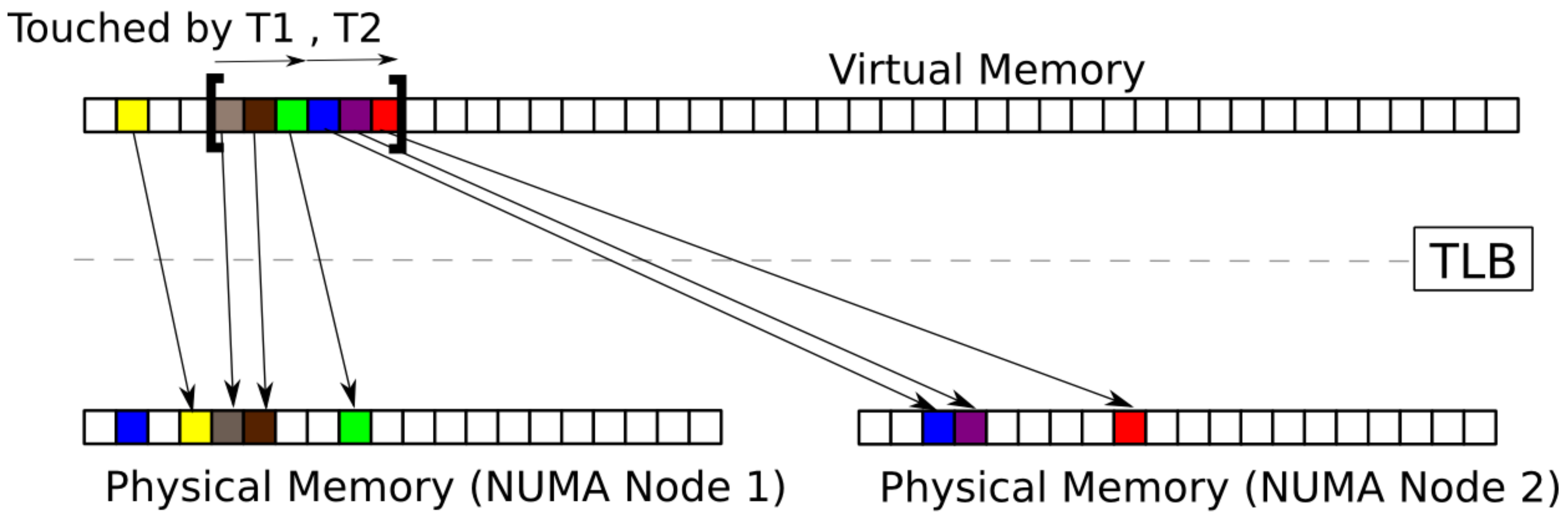
Compute

- **Thread 1 (T1) run on NUM Node 1**
- **Thread 2 (T2) run on NUM Node 2**
- **Problem : T2 access to memory located in NUMA node 1**



NUMA : Initialization with same access pattern

- **Initialize with same parallel access.**
- **It will ensure a better placement on NUMA nodes.**
- **Not always trivial if use different access pattern.**



Init method	Elapsed time (seconds)
Memset	35
#pragma omp for	15

NUMA allocation

- **Sequential programs :**
  - Mainly managed by the OS, clearly bounded by process.
  - User can request some enhancement : interleaving to get more bandwidth
- **In parallel programs, now exposed to programmers :**
  - The OS didn't know data-sets used by threads
  - Memory location is defined by first access (First touch)
  - Can force with some libraries : hwloc, libnuma…
  - Take care of consecutive small allocations : it can be on same page, but used on different NUMA nodes.

OS impact on performance

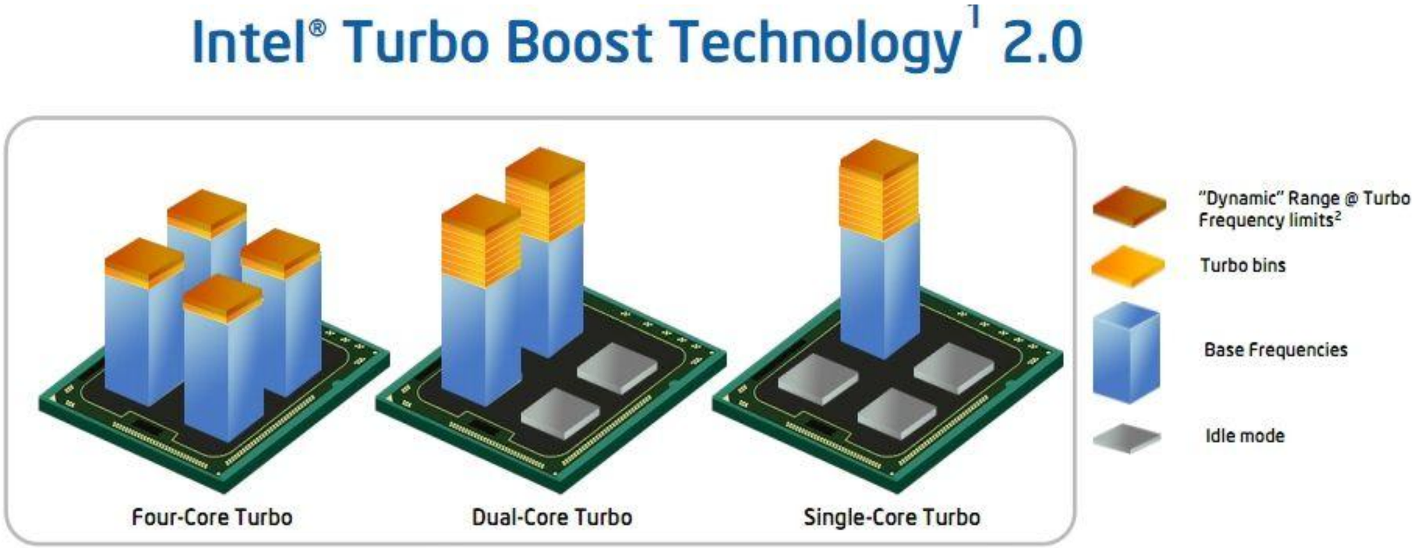
- **OS can have large impact on performance**
- **Event out of system call sections**
- **Sources of problems :**
  - Bad usage of system calls (eg. too many small allocations)
  - Limitation of OS policies (eg. cache leak)
  - Bad interaction between system libraries, OS and applications.
  - Don’t take care of what the OS do indirectly (eg. NUMA memory mappings).

Reproducibility

Can we reproduce measurements ?

- **Getting more source of variations:**
  - Hardware / Software frequency scaling.
  - Threads placement.
  - NUMA memory management / scheduling.
  - Higher number of threads in interaction.
- **Need to take time to check what we measure and how.**
- **Codes must be as robust as possible face to this (in term of performance).**

- **OS can request frequency scaling (software)**
- **Now, Intel can increase frequency of some cores (Hardware)**



Example on

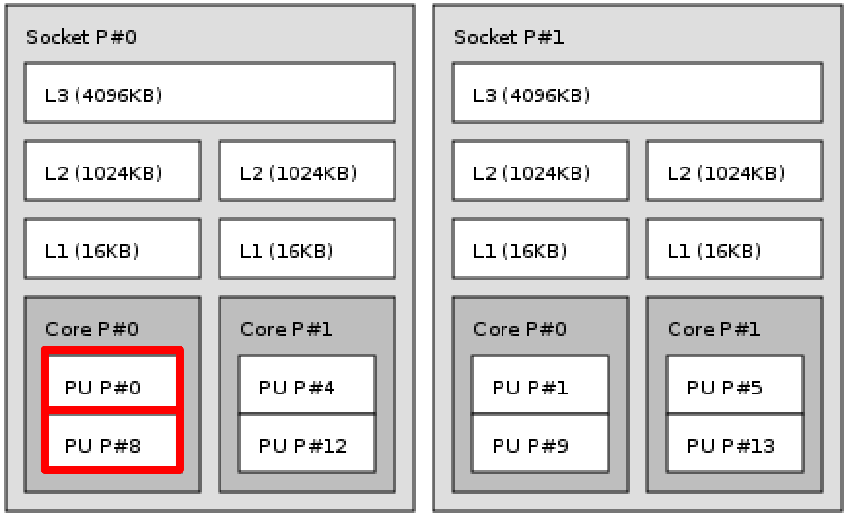
Sandy Bridge

Example 2 : Thread placement

- **Multiple thread can share resources :**
  - Shared hardware caches space.
  - Memory bandwidth.
  - Execution units (hyper-threads)
- **OS scheduling can change the sharing behavior :**
  - Is it better to share or not ? NUMA access ?
- **It depend on applications :**
  - Data sharing between threads
  - Bandwidth usage
  - Data set size
  - NUMA sensitivity

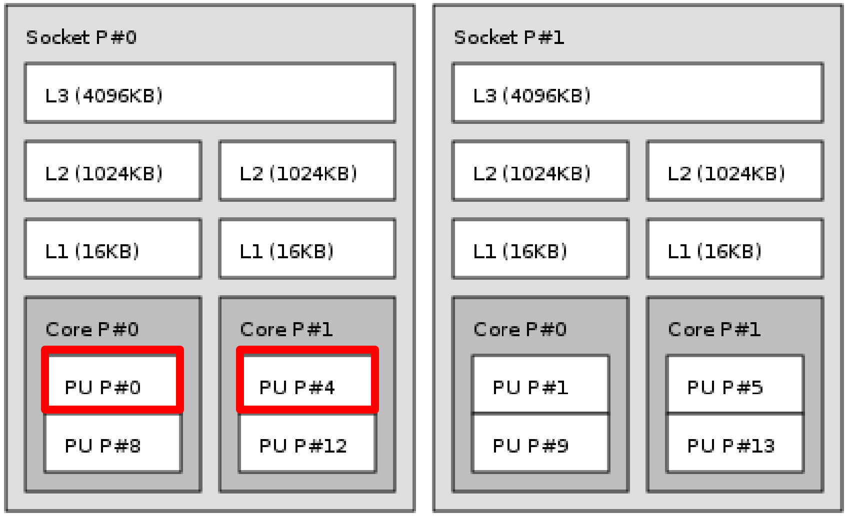
Example 2 : Thread placement of 2 threads

- **Example : 2 socket**
- **Three ways to map 2 threads :**
  1. All on core 0 (compact)



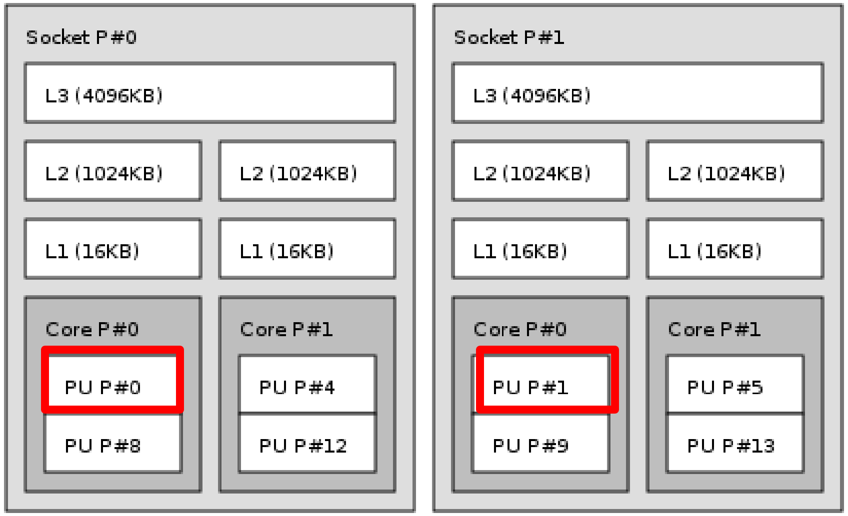
Example 2 : Thread placement of 2 threads

- **Example : 2 socket**
- **Three ways to map 2 threads :**
  1. All on core 0 (compact)
  2. One on each core on same socket

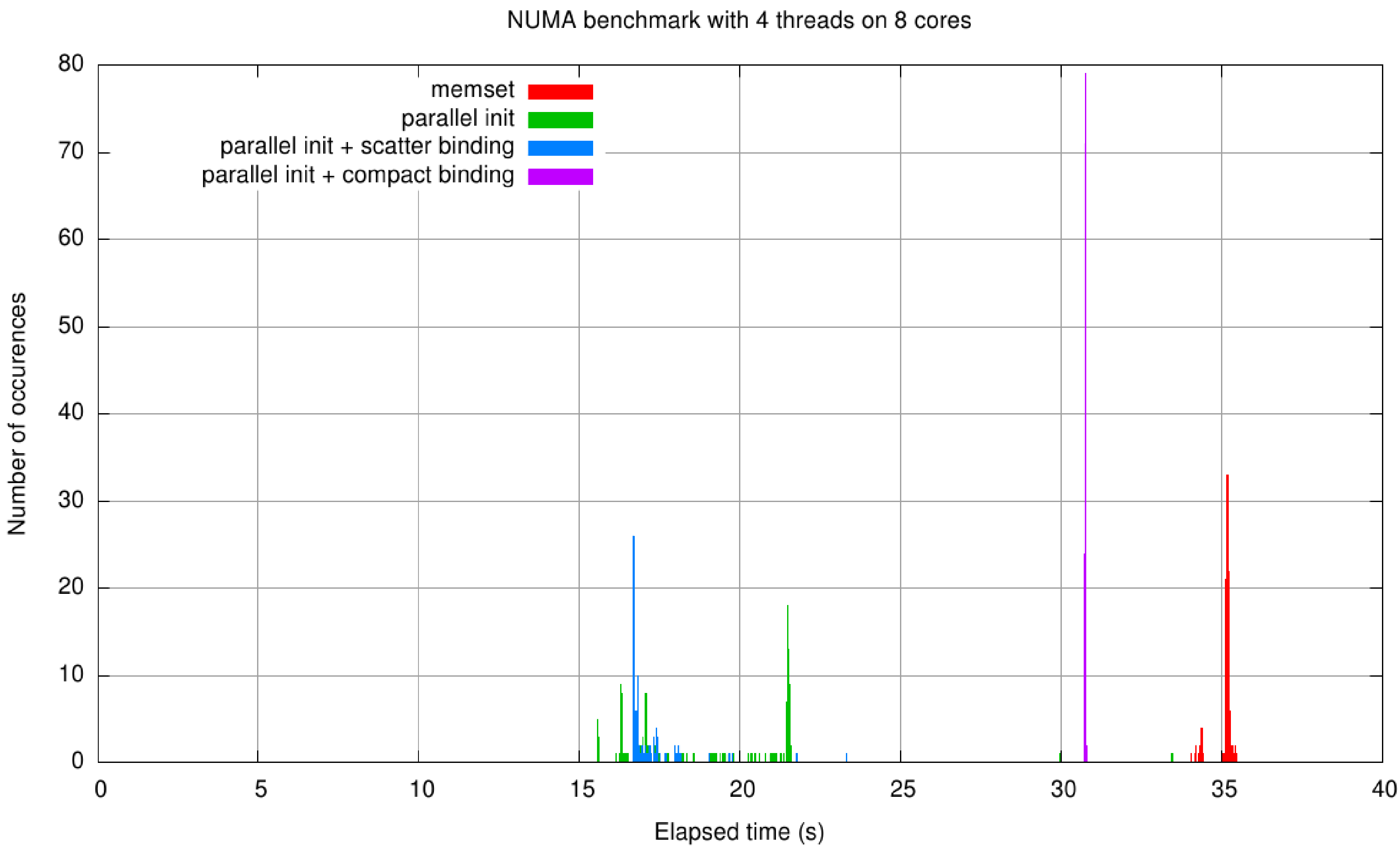


Example 2 : Thread placement of 2 threads

- **Example : 2 socket**
- **Three ways to map 2 threads :**
  1. All on core 0 (compact)
  2. One on each core on same socket
  3. One on each socket (scatter)



Observe NUMA bindings with time distributions



Example 2 : Thread placement

- **If impacted, take care of what you measure**
- **You can force placement.**
- **Intel OMP :**
  - Environment variable KMP_AFFINITY
- **API :**
  - hwloc
  - sched_set_affinity() (linux, system dependant)
- **Command line :**
  - hwloc-bind
  - numactl (linux)
  - schedtool (linux)
  - ….

Measurement methodology

- **When trying to understand what append :**
  - Ensure to know the status of parameters.
  - Fix them if you get in trouble to understand what append.
- **Repeat your measures to check reproducibility**
- **Evaluate variation amplitudes of your measures.**
  - If too large, find the source and adapt your approach
- **If use instrumentation, take care of overhead.**
  - Too large impact can change your code behavior.

Conclusion

Conclusion

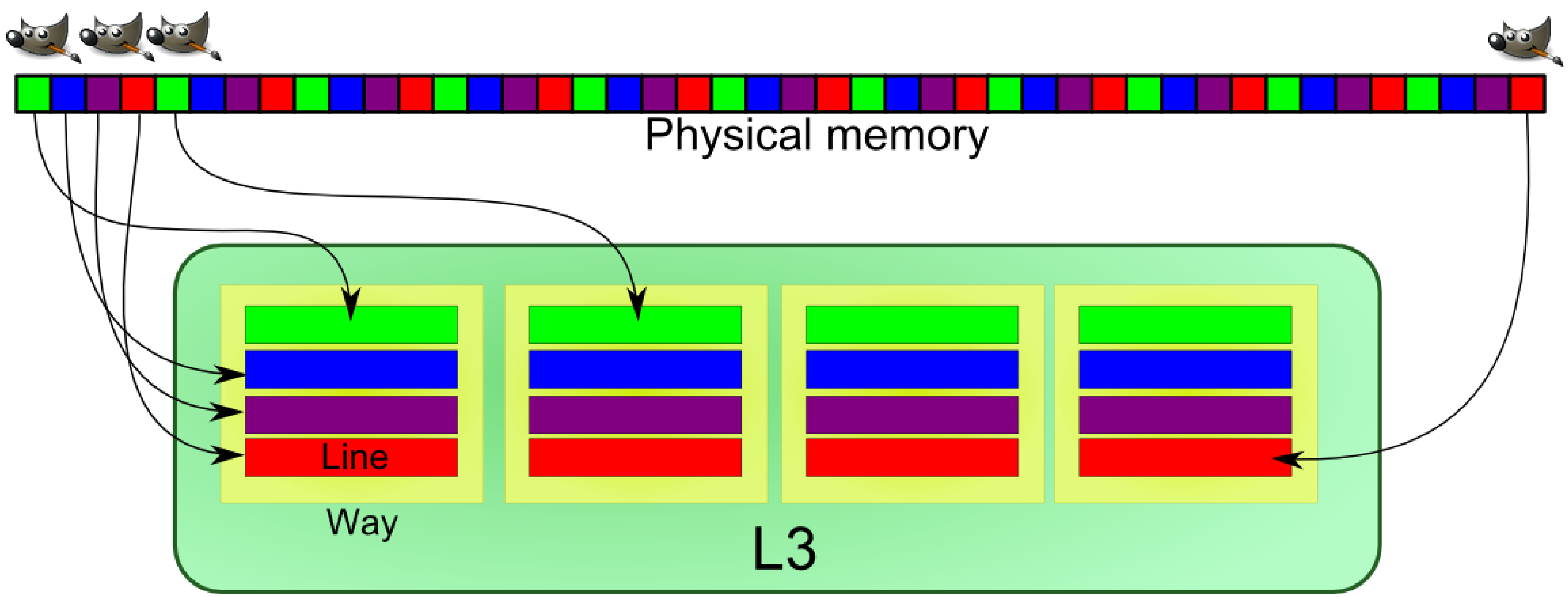
- **Optimization is a whole**
- **To get performance you need to efficiently use :**
  - Hardware
  - OS
  - Frameworks
  - Interaction between all
- **For analysis, take care of stability of you measurement**
  - Estimate variability
  - Check your parameters
  - Fix parameters if it penalized for understanding.

energie atomique • energies alternatives

BACKUP

Cache associativity

- **Problem : Need to find quickly data in the cache**
- **Solution :**
  - Each address has a unique location in the cache.
  - Replicate this schemes for flexibility : caches ways.



Cache associativity

- **Problem : Need to find quickly data in the cache**
- **Solution :**
  - Each address has a unique location in the cache.
  - Replicate this schemes for flexibility : caches ways.