Thread level parallelism: It’s time now !

André Seznec
IRISA/INRIA
CAPS team

Focus of high performance computer architecture

- Up to 1980
  - Mainframes
- Up to 1990
  - Supercomputers
- Till now:
  - General purpose microprocessors
- Coming:
  - Mobile computing, embedded computing

Uniprocessor architecture has driven performance progress so far

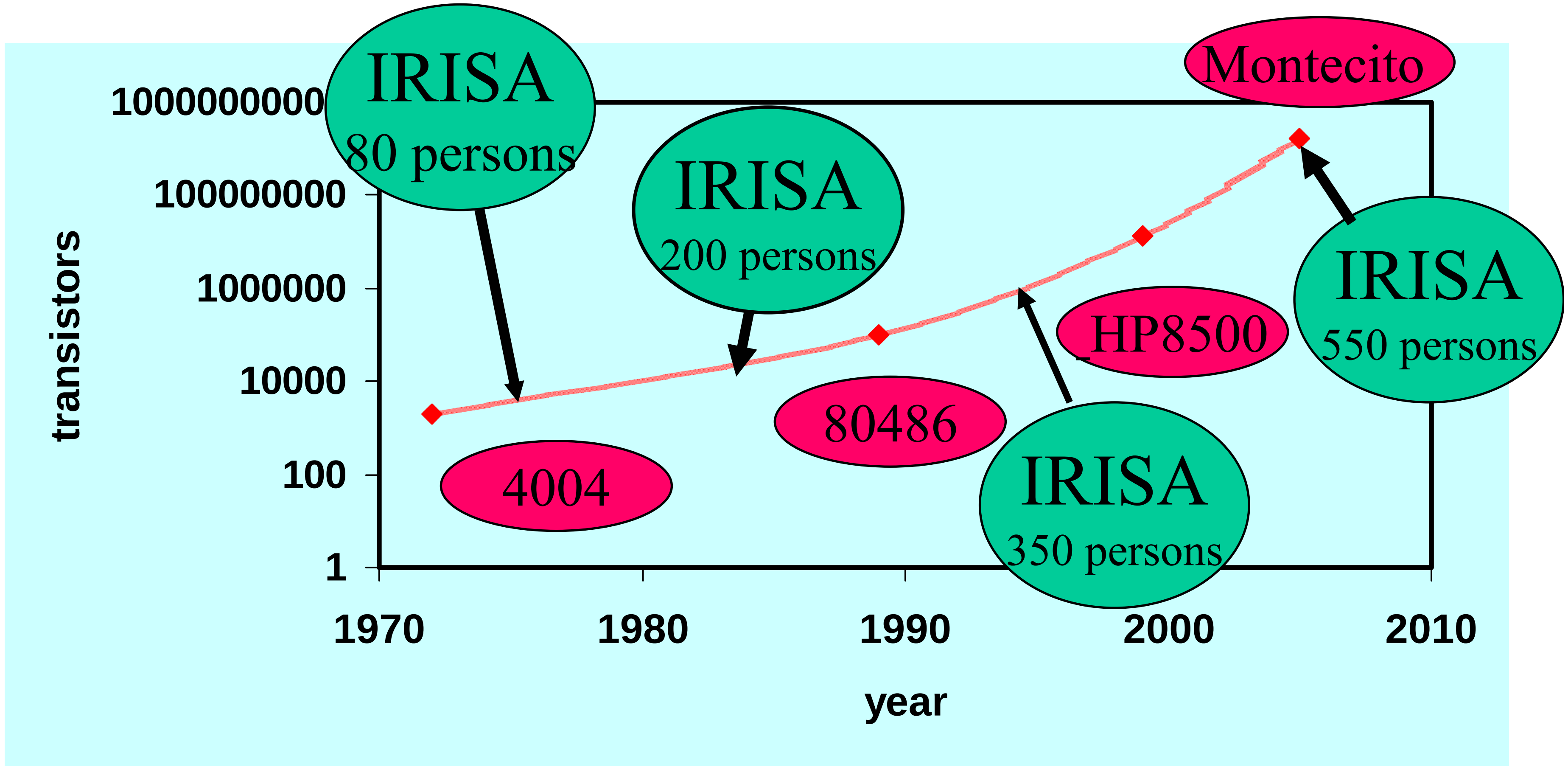
The famous "Moore's law" ☺

“Moore’s Law”: predict exponential growth

- Every 18 months:
  - ➔ The number of transistors on chip doubles
  - ➔ The performance of the processor doubles
  - ➔ The size of the main memory doubles

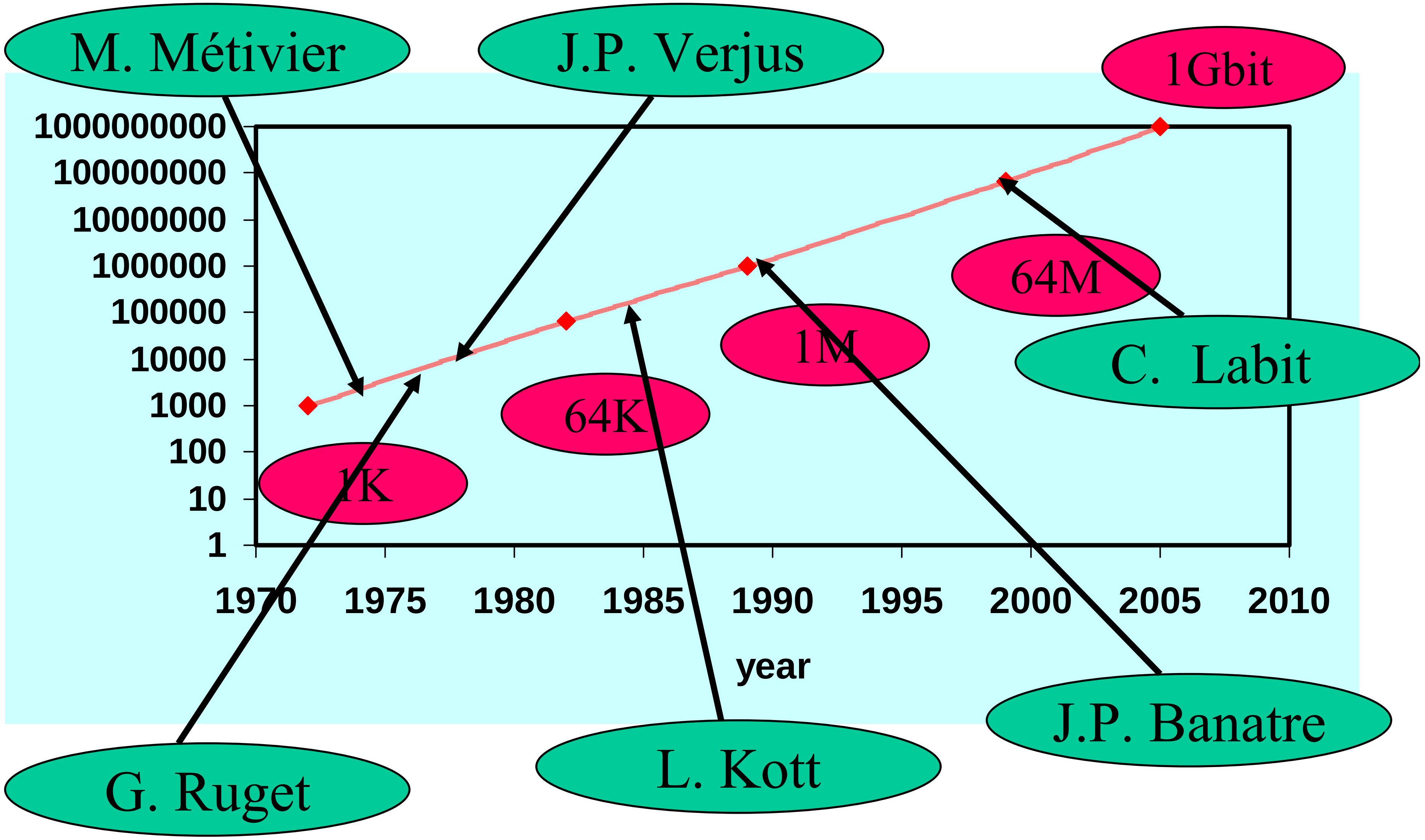
30 years IRISA = 1,000,000 x

Moore’s Law: transistors on a processor chip



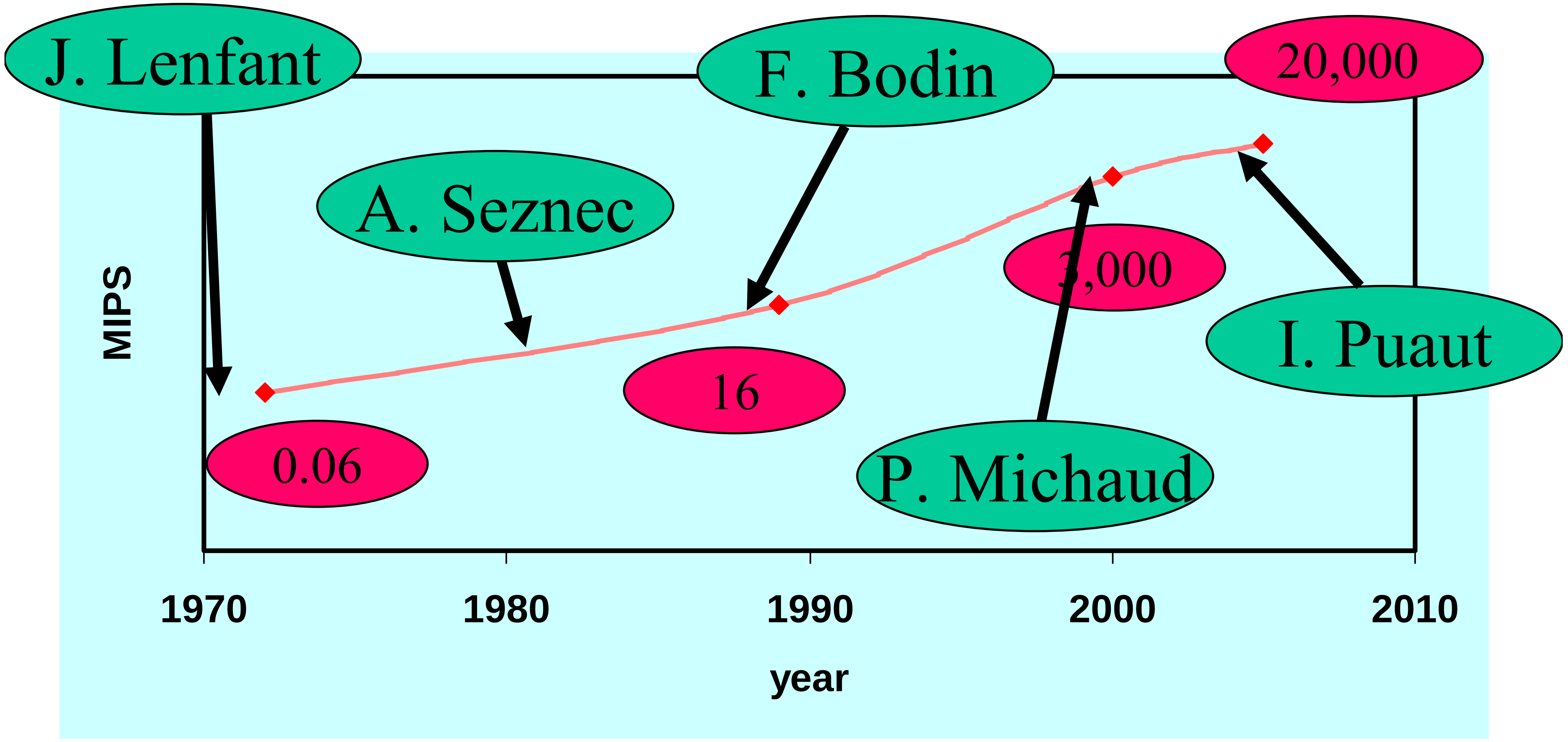
Moore’s Law: bits per DRAM chip

and the IRISA directors



Moore's Law: Performance

and the CAPS group ☺



And parallel machines, so far ..

- Parallel machines have been built from every processor generation:
  - ➔ Hardware coherent shared memory processors:
    - Dual processors board
    - Up to 8-processor server
    - Large (very expensive) hardware coherent NUMA architectures
  - ➔ Distributed memory systems:
    - Intel iPSC, Paragon
    - clusters, clusters of clusters ..

Parallel machines have not been mainstream so far

~~But it might change~~

But it will change

What has prevented parallel machines to prevail ?

- Economic issue:
  - ➔ Hardware cost grew superlinearly with the number of processors
  - ➔ Performance:
    - Never been able to use the last generation micropocessor:
  - ➔ Scalability issue:
    - Bus snooping does not scale well above 4-8 processors

- Parallel applications are missing:
  - ➔ Writing parallel applications requires thinking “parallel”
  - ➔ Automatic parallelization works on small segments

What has prevented parallel machines to prevail ? (2)

- We (the computer architects) were also guilty☺:
  - ➔ We just found how to use these transistors in a uniprocessor
- IC technology brought the transistors and the frequency
- We brought the performance ☺:
  - ➔ Compiler guys also helped a little bit ☺

Up to now, what was microarchitecture about ?

- Memory access time is 100 ns
- Program semantic is sequential
- Instruction life (fetch, decode,..,execute, ..,memory access,..) is 10-20 ns

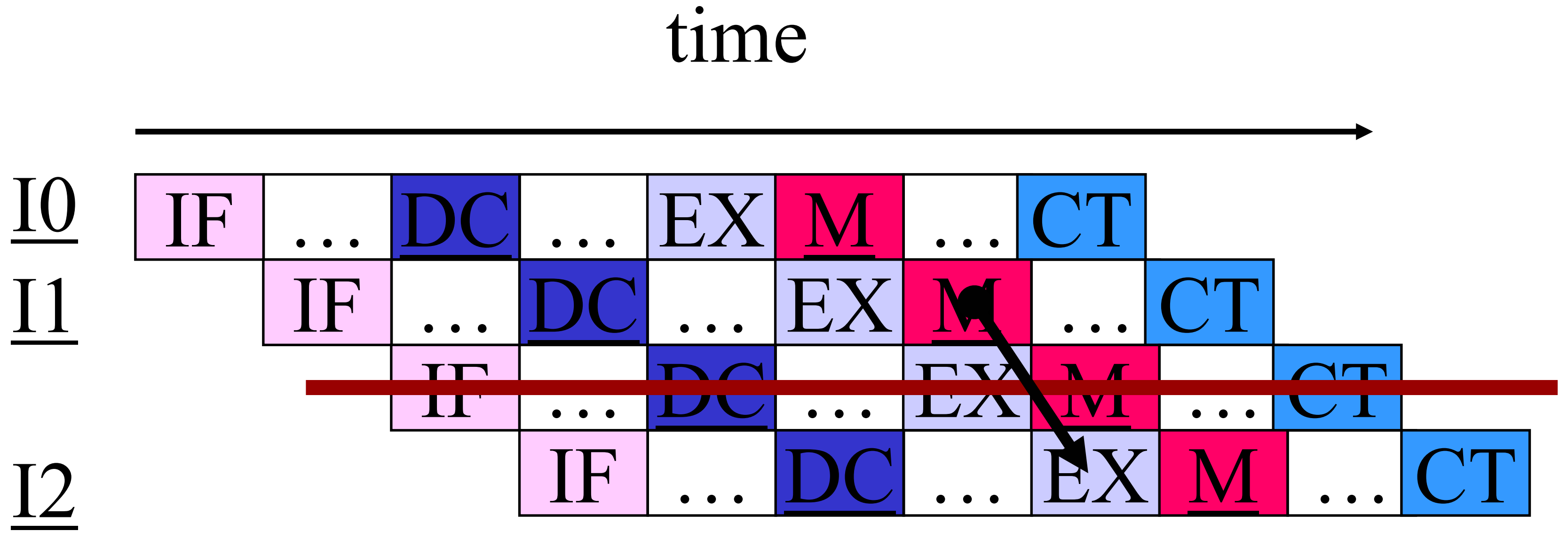
- **How can we use the transistors to achieve the highest performance as possible?**
  - ➔ **So far, up to 4 instructions every 0.3 ns**

The architect tool box for uniprocessor performance

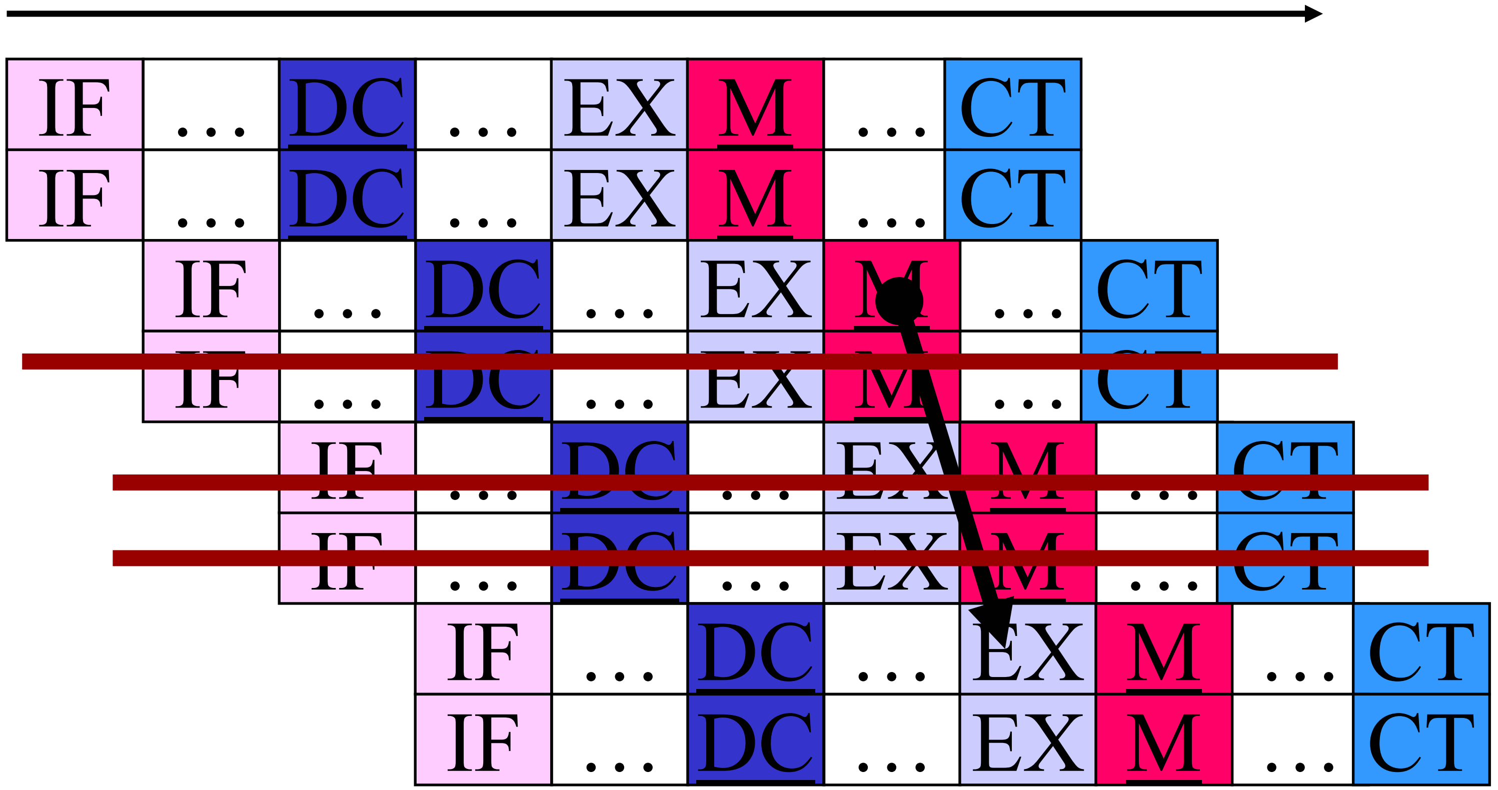
- Pipelining
- Instruction Level Parallelism
- Speculative execution
- Memory hierarchy

Pipelining

- Just slice the instruction life in equal stages and launch concurrent execution:



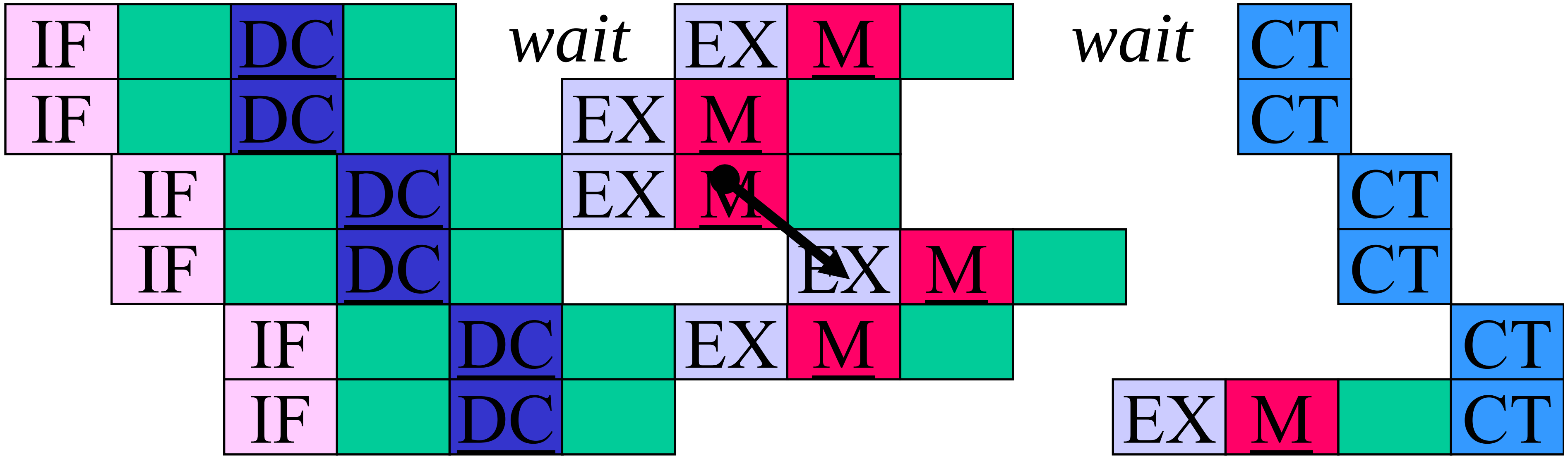
+ Instruction Level Parallelism



+ out-of-order execution

To optimize resource usage:

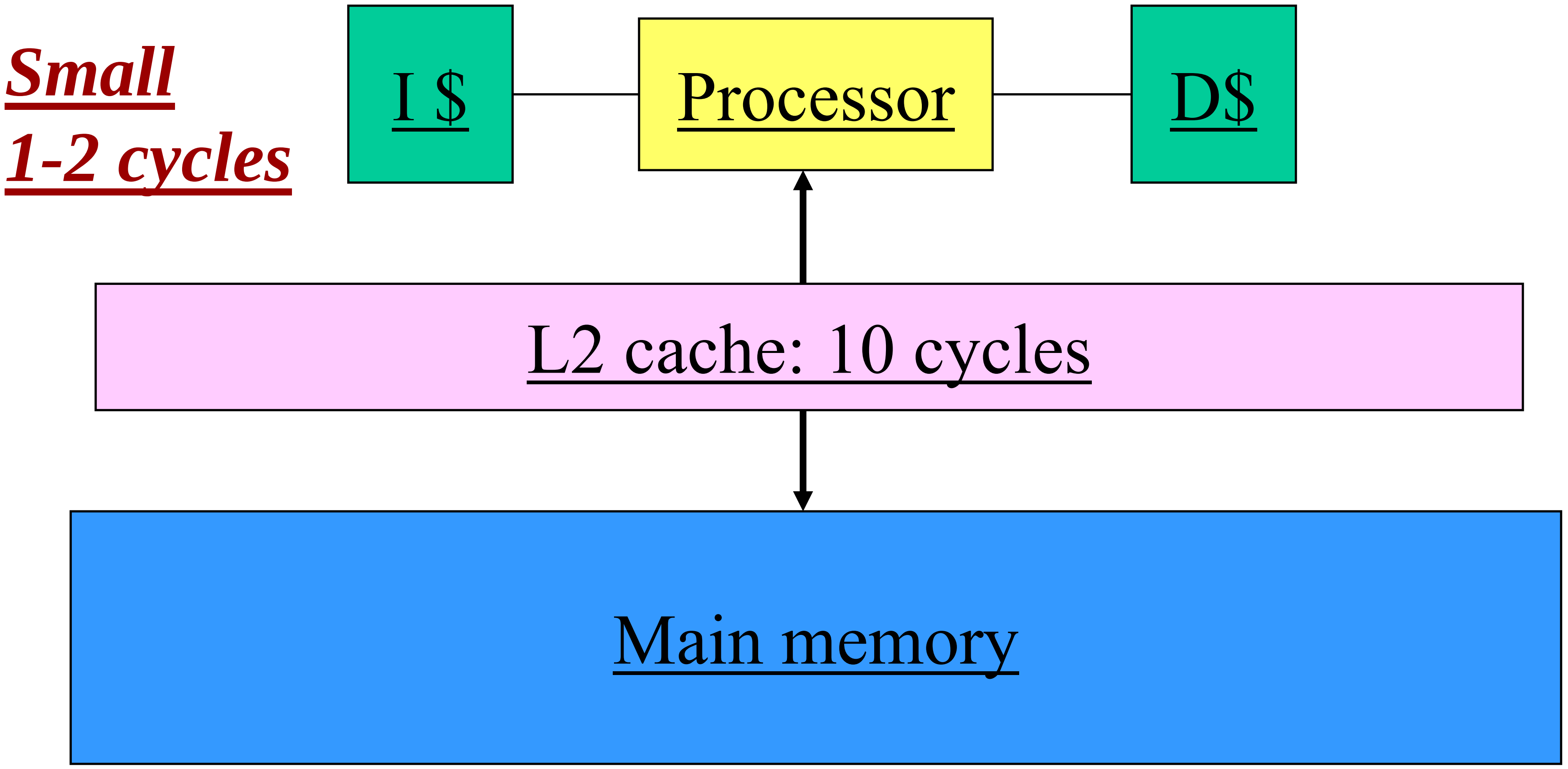
Executes as soon as operands are valid



+ speculative execution

- 10-15 % branches:
  - ➔ On Pentium 4: direction and target known at cycle 31 !!
- Predict and execute speculatively:
  - ➔ Validate at execution time
  - ➔ State-of-the-art predictors:
    - • ≈2 misprediction per 1000 instructions
- Also predict:
  - ➔ Memory (in)dependency
  - ➔ (limited) data value

+ memory hierarchy



+ prefetch

- On a miss on the L2 cache:
  - ➔stops for 300 cycles ! ?!

- Try to predict which memory block will be missing in the near future and bring it in the L2 cache

Can we continue to just throw transistors in uniprocessors ?

•Increasing inst. per cycles?
•Larger caches ?
•New prefetch mechanisms ?

One billion transistors now !!
The uniprocessor road seems over

- 16-32 way uniprocessor seems out of reach:
  - ➔ just not enough ILP
  - ➔ quadratic complexity on a few key components: register file, bypass, issue logic,..
  - ➔ to avoid temperature hot spots:
    - very long intra-CPU communications would be needed
  - ➔ 5-7 years to design a 4-way superscalar core:
    - How long to design a 16-way ?

One billion transistors: Process/Thread level parallelism.. it’s time now !

- Simultaneous multithreading:
  → TLP on a uniprocessor !

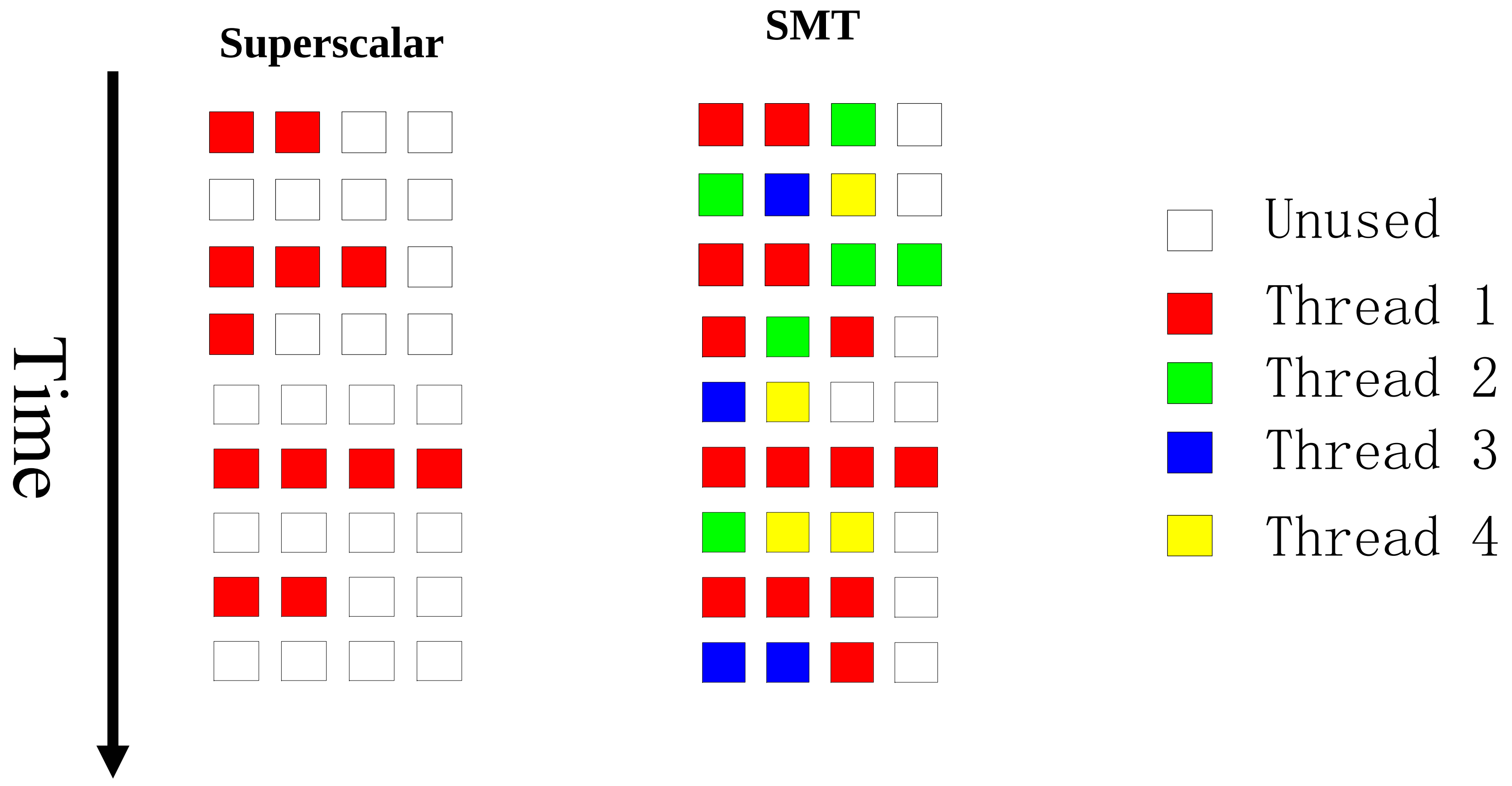
- Chip multiprocessor

Combine both !!

Simultaneous Multithreading (SMT): parallel processing on a uniprocessor

- functional units are underused on superscalar processors

- SMT:
  - ➔ Sharing the functional units on a superscalar processor between several process
- Advantages:
  - ➔ Single process can use all the resources
  - ➔ dynamic sharing of all structures on parallel/multiprocess workloads


Superscalar
SMT
Time
Unused
Thread 1
Thread 2
Thread 3
Thread 4

The Chip Multiprocessor

- Put a shared memory multiprocessor on a single die:
  - ➔Duplicate the processor, its L1 cache, may be L2,
  - ➔Keep the caches coherent
  - ➔Share the last level of the memory hierarchy (may be)
  - ➔Share the external interface (to memory and system)

General purpose Chip MultiProcessor (CMP): why it did not (really) appear before 2003

- Till 2003 better (economic) usage for transistors:
  - ➔ Single process performance is the most important
  - ➔ More complex superscalar implementation
  - ➔ More cache space:
    - Bring the L2 cache on-chip
    - Enlarge the L2 cache
    - Include a L3 cache (now)

Diminishing return !!

General Purpose CMP: why it should not still appear as mainstream

- No further (significant) benefit in complexifying single processors:
  - ➔ Logically we shoud use smaller and cheaper chips
  - ➔ or integrate the more functionalities on the same chip:
    - E.g. the graphic pipeline

- Very poor catalog of parallel applications:
  - ➔ Single processor is still mainstream
  - ➔ Parallel programming is the privilege (knowledge) of a few

General Purpose CMP:
why they appear as mainstream now !

The economic factor:

-The consumer user pays 1000-2000 euros for a PC

-The professional user pays 2000-3000 euros for a PC

A constant:

The processor represents 15-30 % of the PC price

Intel and AMD will not cut their share

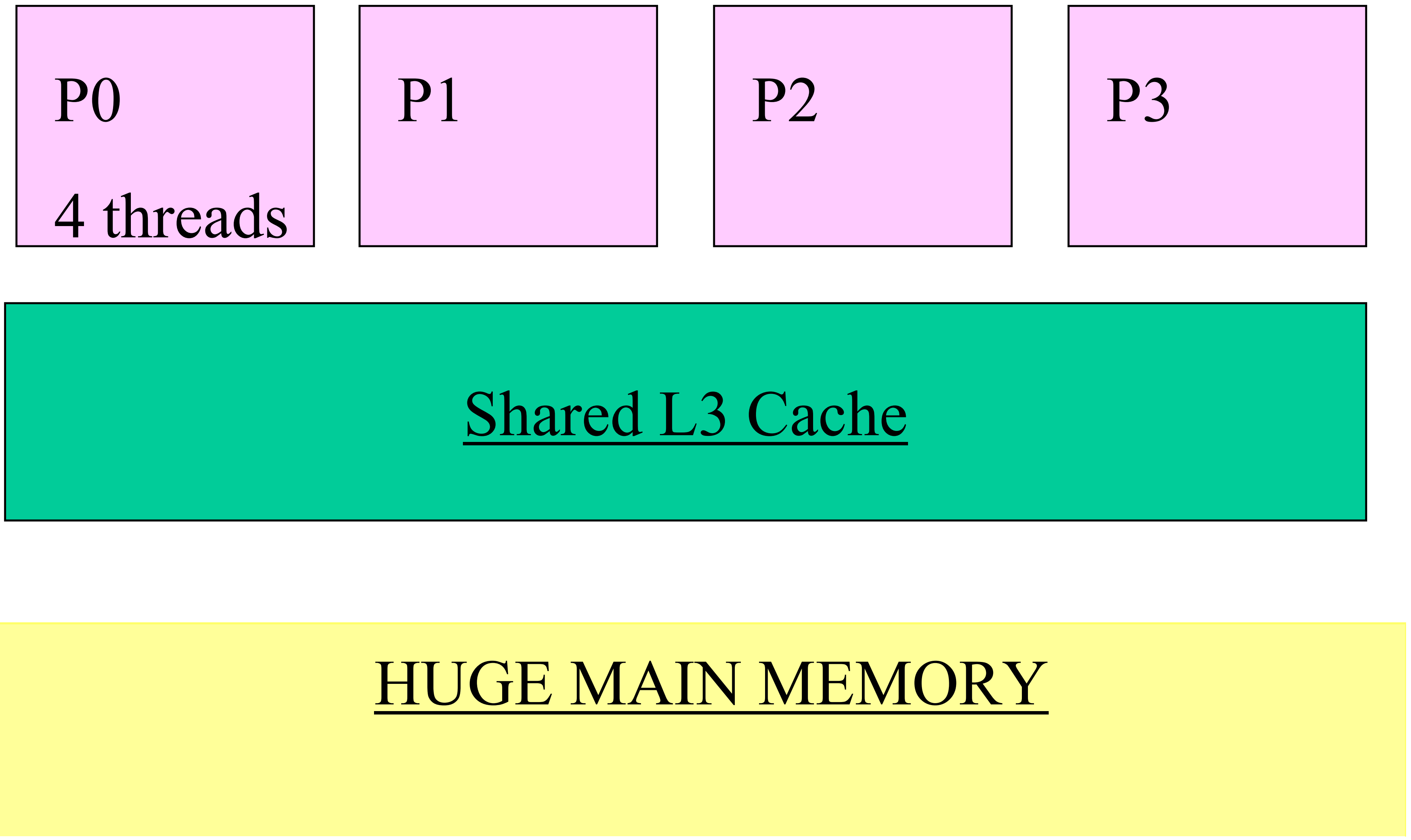
Chip multiprocessor: what is the situation (2005) ?

- PCs : Dual-core Pentium 4 and Amd64
- Servers:
  - Itanium Montecito: dual-core
  - IBM Power 5: dual-core
  - Sun Niagara: 8 processor CMP

General Purpose Multicore SMT: an industry reality Intel and IBM

- Intel Pentium 4 was developped as a 2-context SMT:
  - ➔ Coined as hyperthreading by Intel
  - ➔ Dual-core SMT ☺
- Intel Itanium Montecito: dual-core   2-context SMT
- IBM Power5 : dual-core 2-context SMT

a 4-core 4-thread SMT:
think about tuning for performance !



a 4-core 4-thread SMT: think about tuning for performance ! (2)

- Write the application with more than 16 threads
- Take in account first level memory hierarchy sharing !
- Take in account 2nd level memory hierarchy sharing !
- Optimize for instruction level parallelism

Hardware TLP is there !!

But where are the threads/processes ?

A unique opportunity for the software industry: hardware parallelism comes for free

Waiting for the threads (1)

- Generate threads to increase performance of single threads:
  - ➔ Speculative threads:
    - Predict threads at medium granularity
      - Either software or hardware
  - ➔ Helper threads:
    - Run ahead a speculative skeleton of the application to:
      - Avoid branch mispredictions
      - Prefetch data

Waiting for the threads (2)

- Hardware transcient faults are becoming a concern:
  - ➔ Runs twice the same thread on two cores and check integrity
- Security:
  - ➔ array bound checking is nearly for free on a out-of-order core

Waiting for the threads (3)

- Hardware clock frequency is limited by:
  - ➔ Power budget: every core running
  - ➔ Temperature hot-spots

- On single thread workload:
  - ➔ Increase clock frequency and migrate the process

Conclusion

- Hardware TLP is becoming mainstream for general-purpose computing
- Moderate degrees of hardware TLPs will be available for mid-term
- That is the first real opportunity for the whole software industry to go parallel !
  - ➔ But it might demand a new generation of application developpers !!