[A]rchitecture et [O]ptimisation de [C]ode pour microprocesseur hautes performances

Parallel Architectures

Allen D. Malony

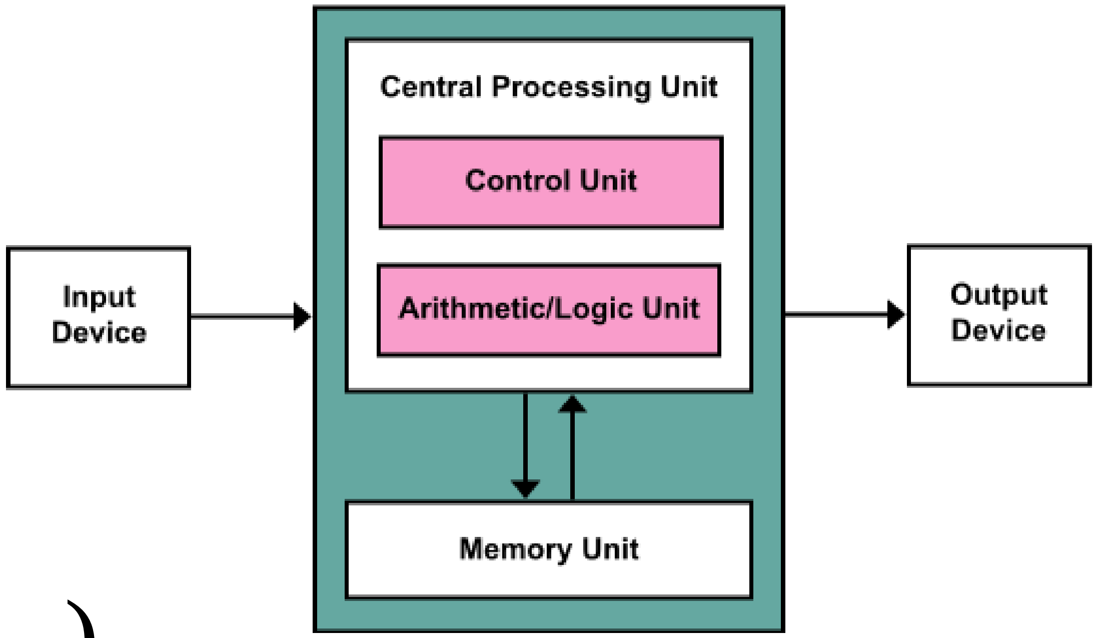
Department of Computer and Information Science
University of Oregon

Origin of Course Materials

- Patrick Carribault, CEA
    - AOC, 2016

Introduction

- Starting point for all of our discussions
  - von Neumann architecture
- Architectural mechanisms to improve core performance
  - Registers
  - Pipeline (bypass, forwarding, …)
  - Execution out of order
  - Branch prediction
  - Vector Units (SIMD)
  - Memory Hierarchy (cache, TLB)
  - Memory preload



Introduction (2)

- Code optimizations
  - Influence of architectural parameters
  - Importance of mastering micro-architectural mechanisms
  - Think about the link to implementation of an ISA
- Optimization model
  - Sequential
  - ILP / SIMD
  - Memory
- Research and product literature
  - Full of transformations / optimizations available!
  - Discuss order of complexity as well as edge effects

Introduction (3)

- How to go further in processor performance?
  - Focus on modern high-performance processors
  - Architecture of HPC computational nodes
  - Look at HPC clusters
- New mechanism to increase the performance of a computing core?

Outline

- Introduction
- Hyperthreading
- Multicore / Manycore
- Multiprocessos
- Architecture of HPC calculation node
- Conclusion

Hyperthreading

- Single core and single instruction stream
  - All previous optimizations take into account a single execution flow (single stream of instructions)
  - Development of several mechanisms to extract performance from this single instruction stream
    - branching, ILP, out-of-order, ...)
- Problem was finding enough independent instructions
- Idea
  - Avoiding stalls in the pipeline by scheduling several independent run-time flows (multiple instruction streams)
  - Need to have support for multiple “threads” where each thread is a separate stream of instructions
- This is called *hyperthreading*

Hyperthreading (2)

- Principle idea
  - Expose the idea of "thread" in the hardware
  - Execute several flows of instructions on single core
  - Several "logical" cores one physical core
- Threads execute (parallel) program instructions
  - Threads of instruction execution
  - Essentially a multicore vision from the OS perspective
- Hardware schedules "hyper threads" on a core
  - 2-way hyperthreading means there are 2 hyper threads
  - 4-way hyperthreading means there are 4 hyper threads
- Must add more hardware to CPU … what?

Hardware for Hyperthreading

- Each thread must have its own hardware state
  - Requires duplication of parts of the calculation core
- Hardware state
  - PC
  - registers
- Parts of the pipeline are duplicated
  - Mainly to handle instruction decode and control
- Major parts of the pipeline are shared
  - Functional units
  - TLB

Impact on the Application

- What are the impacts on the software part?
  - Still sharing a single core
  - Just trying to gain efficiencies with more instructions
- Hyperthreads are really executing software threads
  - Have to be careful about data
- Consider the cache / TLB
  - Hyperthreads touch the same pages and same data
- Consider registers
  - Registers are typically duplicated for performance
  - Makes it easier to compile
- Shared parts of the pipeline
  - Distribution on functional units between hyperthreads
- Hyperthreading can support parallel execution

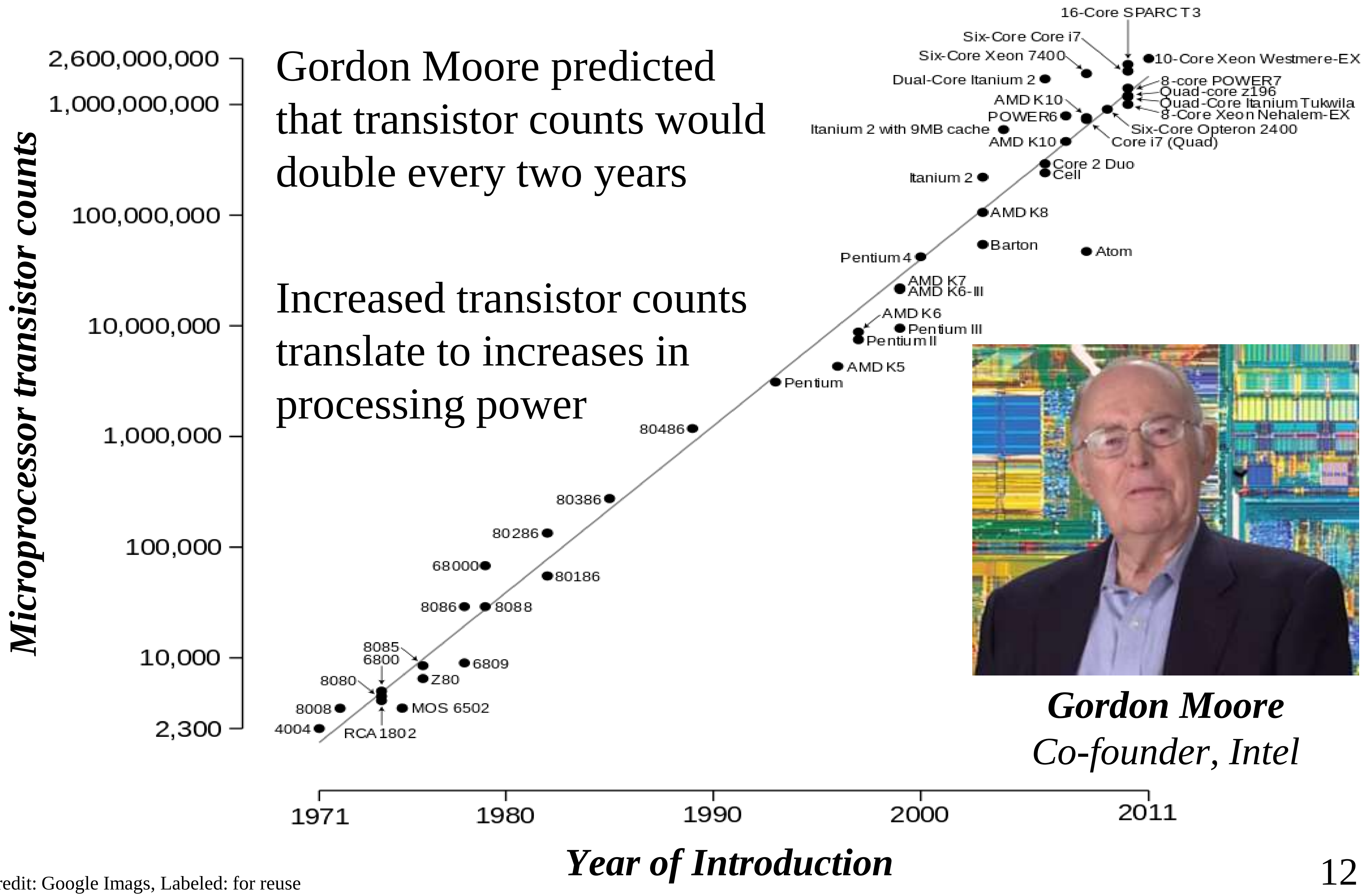
Hyperthreading Evaluation

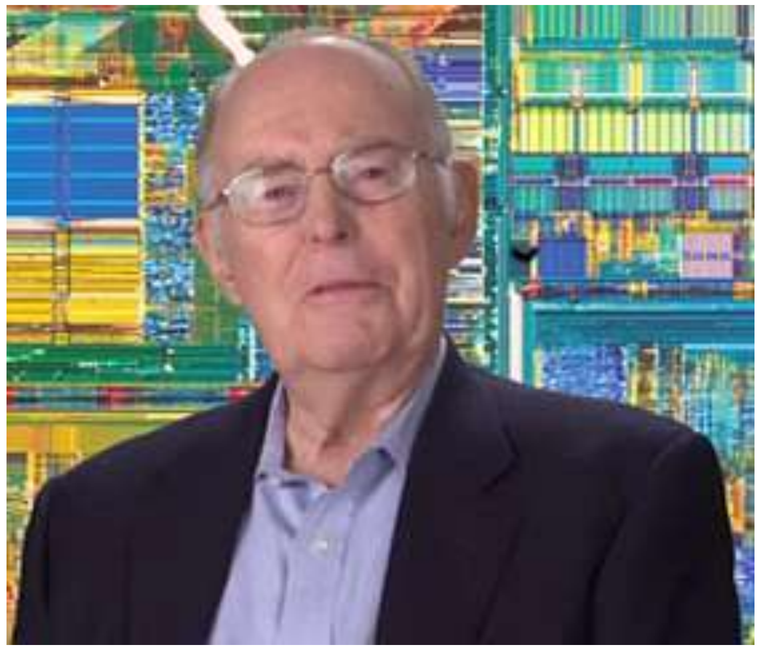
- Advantages
  - Automatic pipeline filling with natural parallelism of instructions
  - Better core utilization
  - Transparent mechanism for the user
- Disadvantages
  - Need to have multiple execution flows
  - Concept of multi-threaded code
  - Can come from a parallel program (does not have to)
- Limits
  - Does not achieve a performance factor of 2
  - At best, it can maxime a core’s full potential performance

Moore's Law

Gordon Moore predicted that transistor counts would double every two years

Increased transistor counts translate to increases in processing power

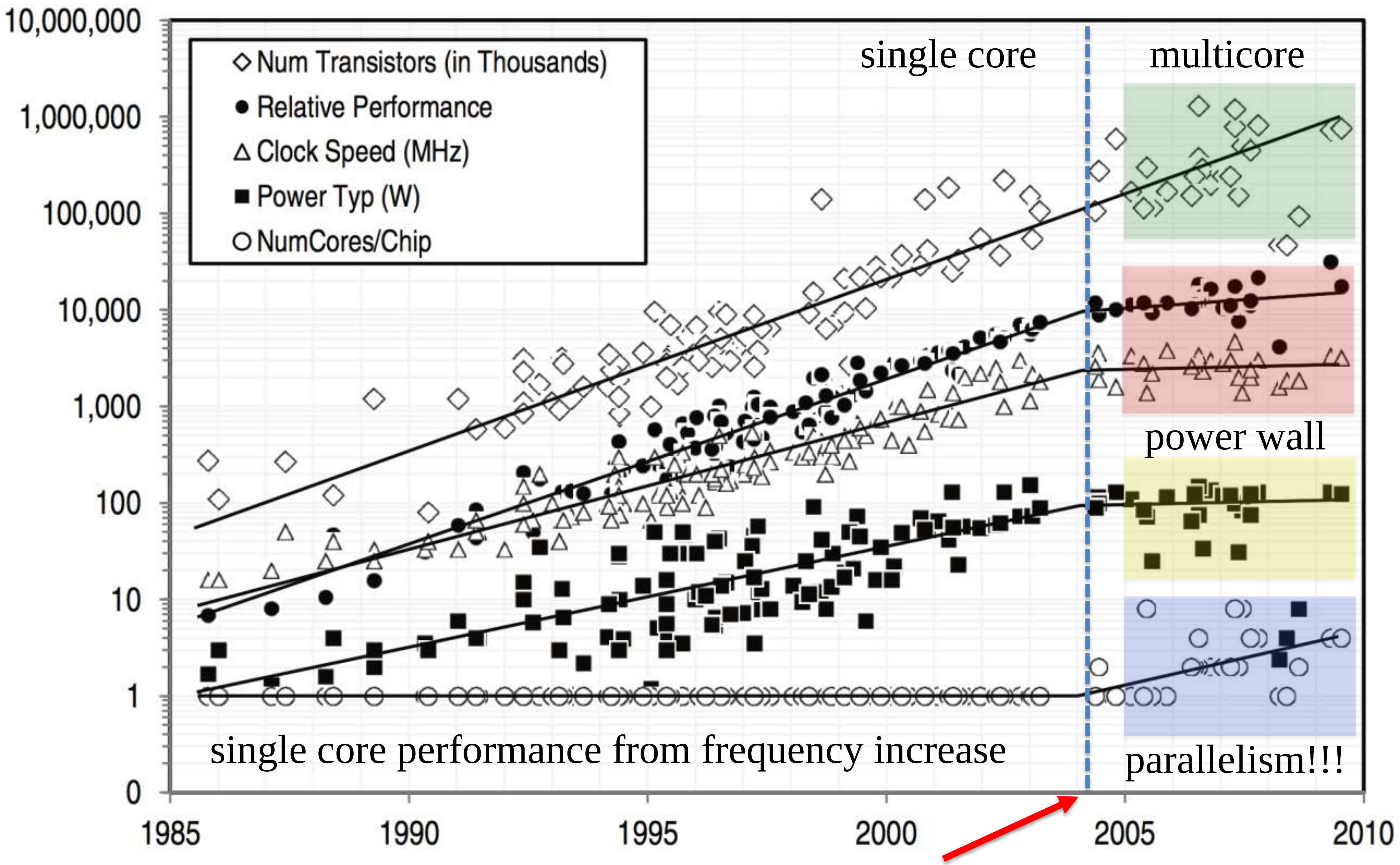




Gordon Moore
Co-founder, Intel

Credit: Google Imags, Labeled: for reuse

What is driving computer architecture?



hardware problem becomes a software problem

Credit: National Research Council, Report: The Future of Computing Performance: Game Over or Next Level?

Multicore Processors

- Decrease in line widths for IC manufacturing
  - What do you do with all the chip space available?
- For a single core
  - Add more registers
  - Add more functional units
  - Add more pipelines
  - Add hyperthreading support (several execution flows)
  - Increase cache size
- Why not add more cores?
  - Replicate the entire hardware
  - Give threads their own core
- Addition of multicore processors

Hardware Vision

- Duplication of everything!
- Keep it all on a single chip
- Let’s call this chip the processor
- We can then speak of multicore processors
- Each core is equivalent to what we called a CPU

Hardware Vision (2)

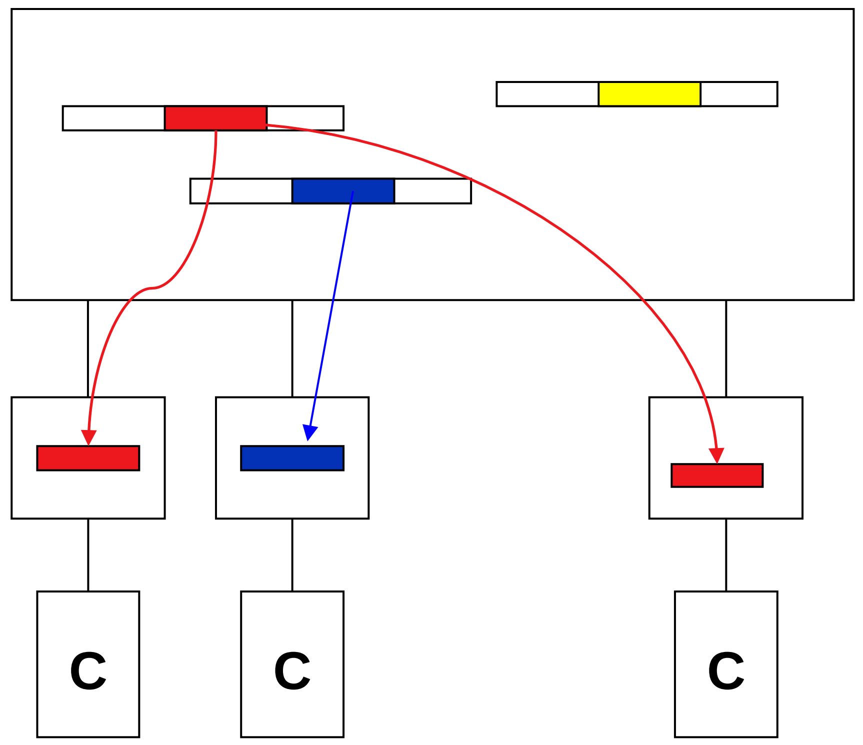
- There is sharing of some resources
- Memory (bandwidth)
  - Fully shared or non-shared bandwidth
- Cache hierarchy
  - Typically, each core has its own Level 1 cache
  - Level 2 cache can be private or shared
  - Last Level 3 cachce is shared
- Shared Cache Type
  - Real sharing of full cache
  - Pseudo sharing

Cache Coherency

- Unified vision of memory hierarchy
  - Single memory system
  - Physical address space
  - Main memory with faster cached levels
- Memory consistence
  - Main memory is where « true » values of data lives
  - If multiple cores are executing instructions that are accessing the same physical memory locations, it is possible for memory to become inconsistent
  - Why?
- Memory hierarchy (i.e., caches) copies data
  - Data can be modified in caches
  - Must maintain coherence of data

Cache Coherency (2)

- Principle problem
  - Memory is shared
  - Multiple copies of a data item present in multiple caches
  - A core attempts to modify the data in its private cache



- Need cache coherency protocols
  - MESI protocol (or derivative)
  - Adding a tag per cache line to record its status
  - Stateful automaton for the updating of this tag and the action to be performed
- Limitations
  - Contention
  - False Sharing

False Sharing

- Principle idea
  - Assume granularity of cache consistency is a cache line
  - No matter what part of this line a core wants to access / modify, the whole line is transferred to/from memory
  - But multiple cores may only access different data within a cache line
- We call this *false sharing*
- How to avoid it?
  - Ensure that data modified by different cores are on different cache lines
  - Requires more attention to have data is placed on different threads
- Example?

Software View

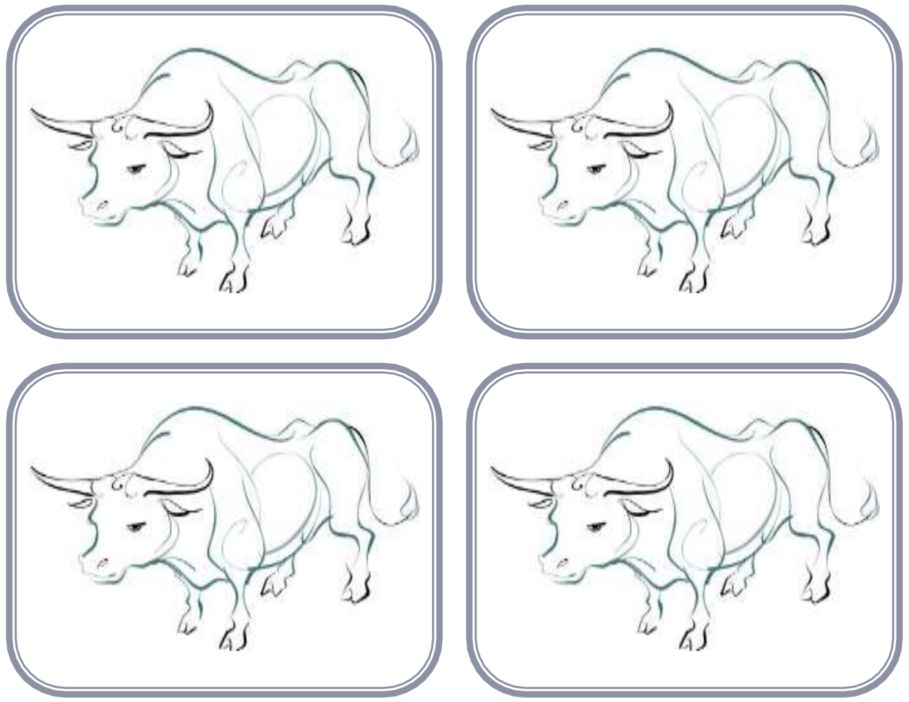
- Think of multiple processor cores as separate CPU that share physical memory
- OS sees several independent computing cores
- Different types of operation
  - Multiple independent processes
  - Multi-threaded process
    - threads from a single process run on the cores
- Shared cache
  - Sharing a memory resource between the cores
  - Optimization if are touching the same data
- Find out the specifications of a multicore processor
  - Use the HWLOC tool

Multicore Processor Evaluation

- Advantages
  - Good use of larger # transistors for multiple cores
  - Direct increase in performance potential
  - More flexible than hyperthreading
  - Scalable performance
  - Does not require increase in CPU frequency
  - Communication between cores is fast
- Disadvantages
  - Need to parallelize its application to get performance
  - Cache coherency needs support in hardware
  - Contention between cores can impact performance
- Limits
  - Cost in terms of transistors
  - No new mechanism to assist application

Manycore Processors

- Multicore processor can have up to 10s of cores
  - Most have 2, 4, 6, or 8
  - Some have 12, 16, 18, and so on
- A manycore processor has significantly higher number of core than a multicore processor
  - All cores still on a single chip

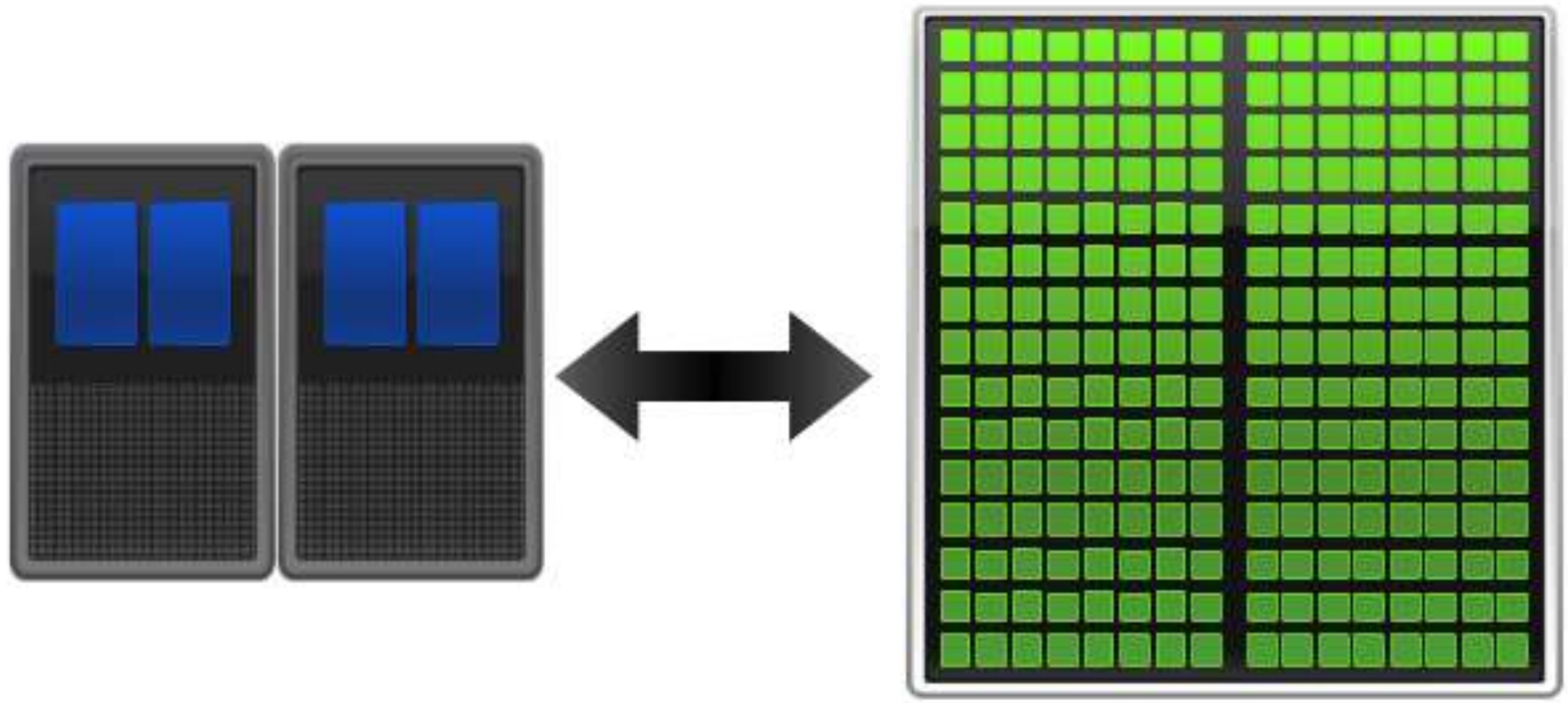

%&'()*+, - $

%53/ *+, - $

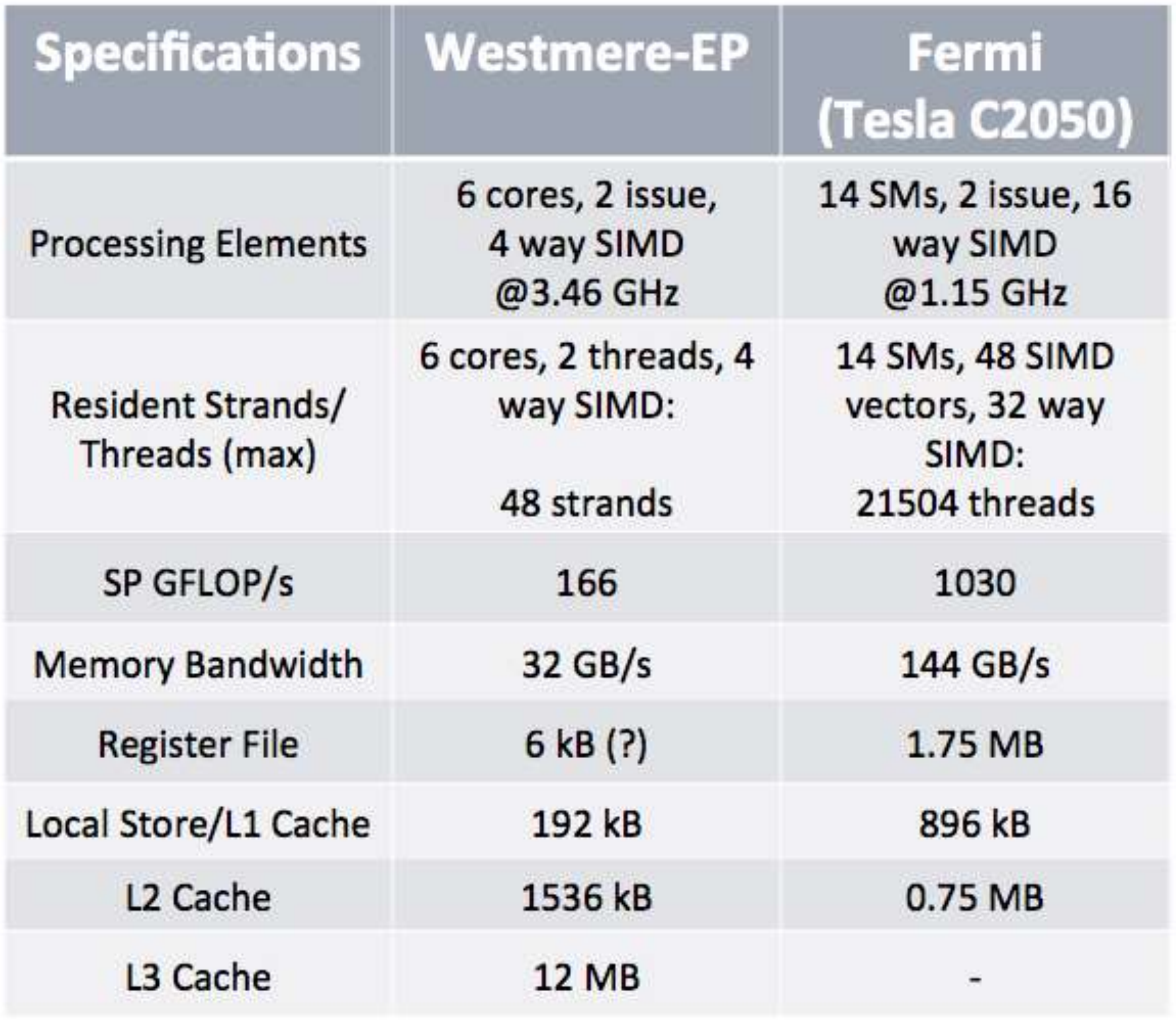
Manycore Processors (2)

- Compare multicore to manycore
  - Each multicore core is optimized for executing a single thread (more powerful pipeline)
  - Cores in a manycore are simpler, but there are more of them and they can achieve greater aggregate throughput
- There are 2 general types:
  - Accelerators (GPGPUs mainly)
  - General purpose (Intel Many Integrated Core (MIC))

GPU Accelerators

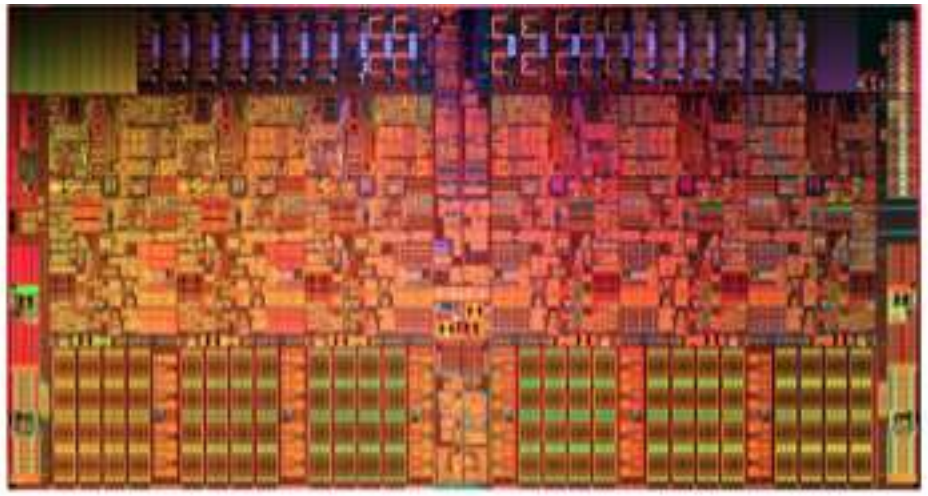


Multicore versus Manycore – Real Chips

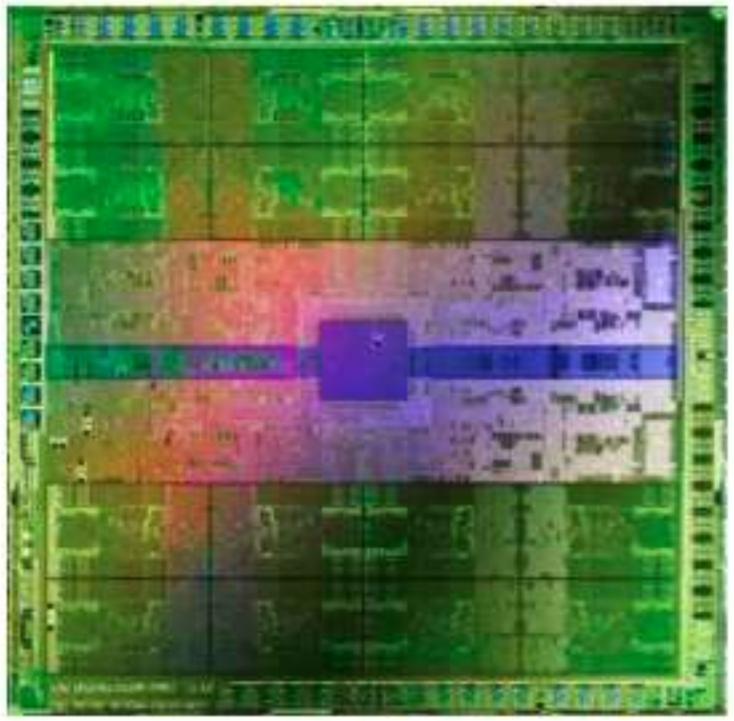


Specifications	Westmere-EP	Fermi (Tesla C2050)
Processing Elements	6 cores, 2 issue, 4 way SIMD @3.46 GHz	14 SMs, 2 issue, 16 way SIMD @1.15 GHz
Resident Strands/ Threads (max)	6 cores, 2 threads, 4 way SIMD: 48 strands	14 SMs, 48 SIMD vectors, 32 way SIMD: 21504 threads
SP GFLOP/s	166	1030
Memory Bandwidth	32 GB/s	144 GB/s
Register File	6 kB (?)	1.75 MB
Local Store/L1 Cache	192 kB	896 kB
L2 Cache	1536 kB	0.75 MB
L3 Cache	12 MB	-

# transistors & area:	1.2 B, 240 mm^2	3 B, 520 mm^2
thermal design power:	130 Watts	160+ Watts? (240 W/card)



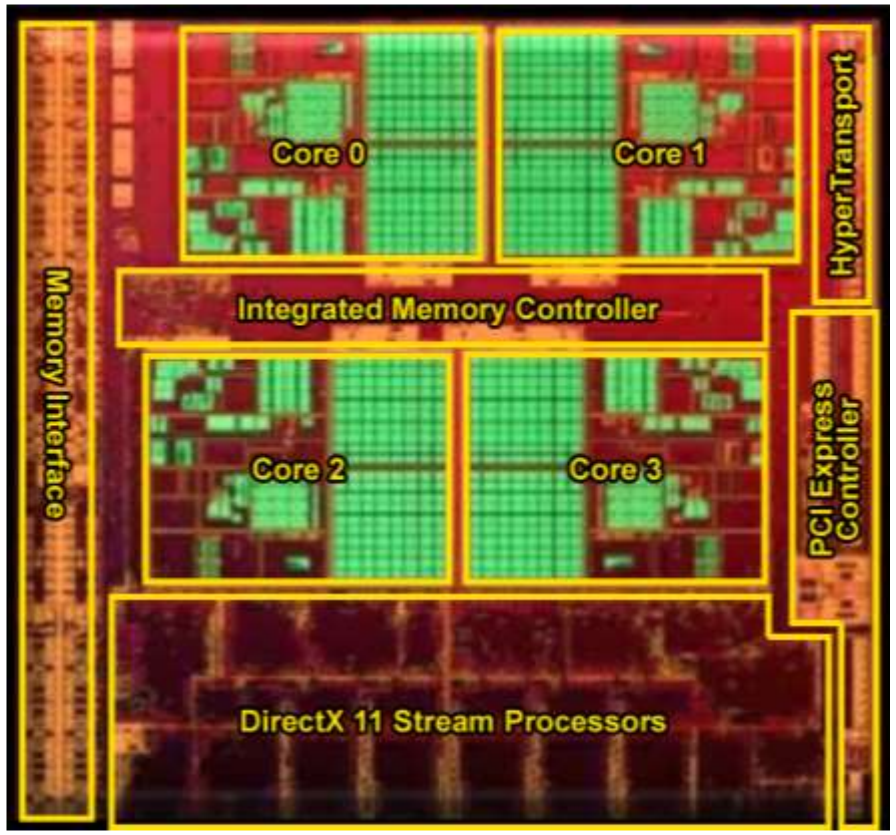
Westmere-EP (32nm)

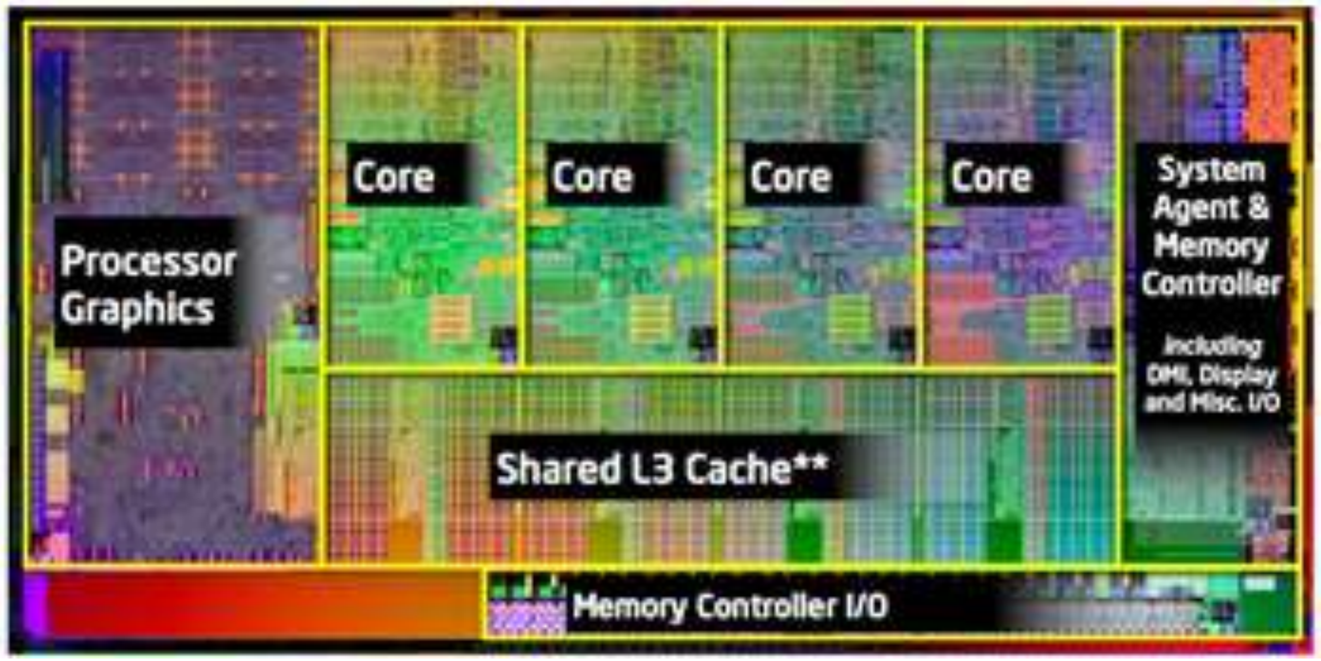


Fermi (40nm)

Integrated CPU and GPU

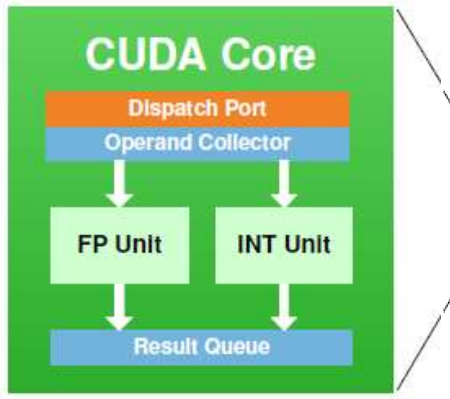
- AMD APU
  - x86 cores
  - Array of Radeon cores
  - Multimedia accelerators
  - Dual channel DDR3
- Intel Ivy Bridge and Haswell
  - x86 cores
  - Symmetric execution units

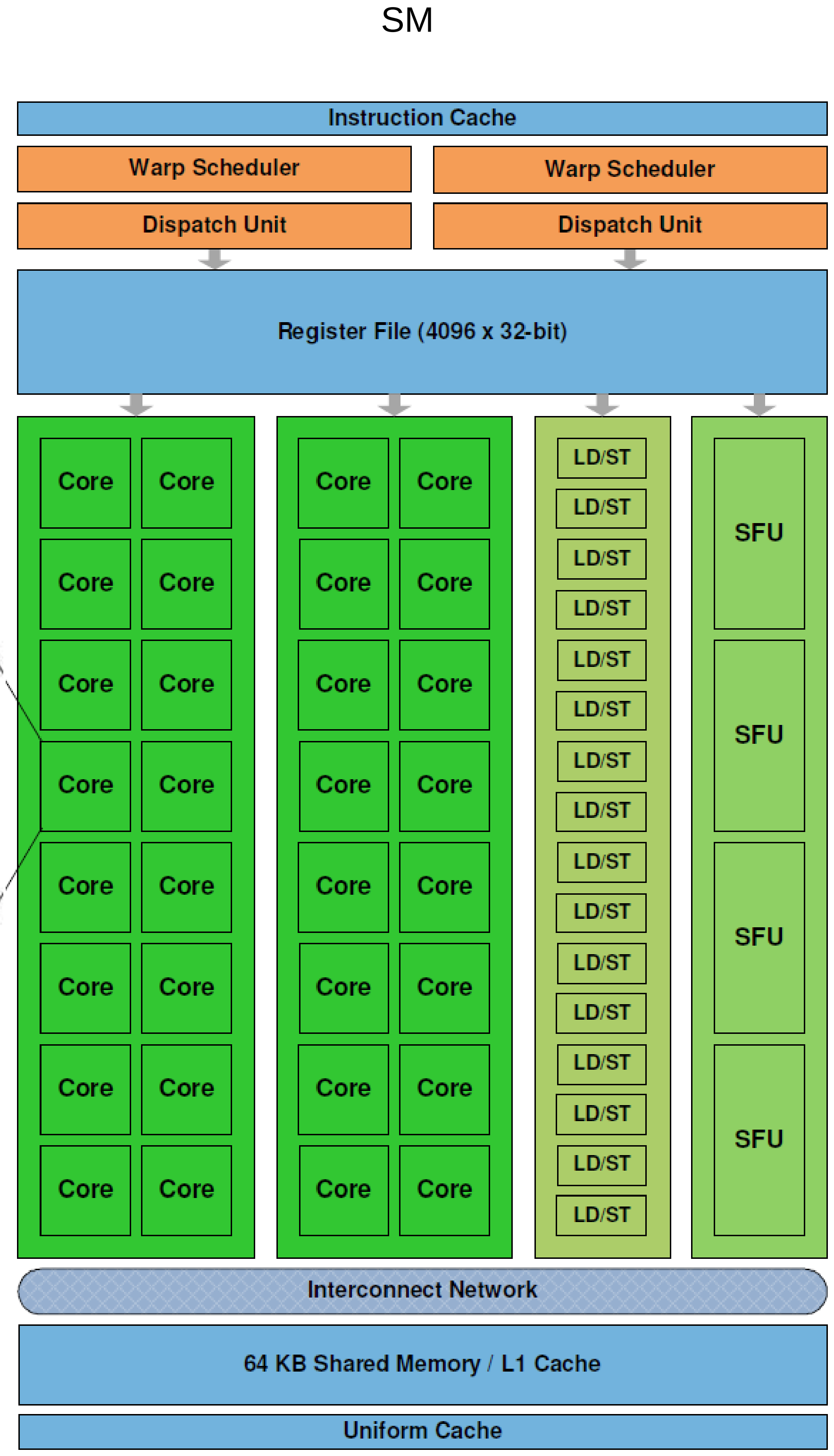




NVIDIA Fermi

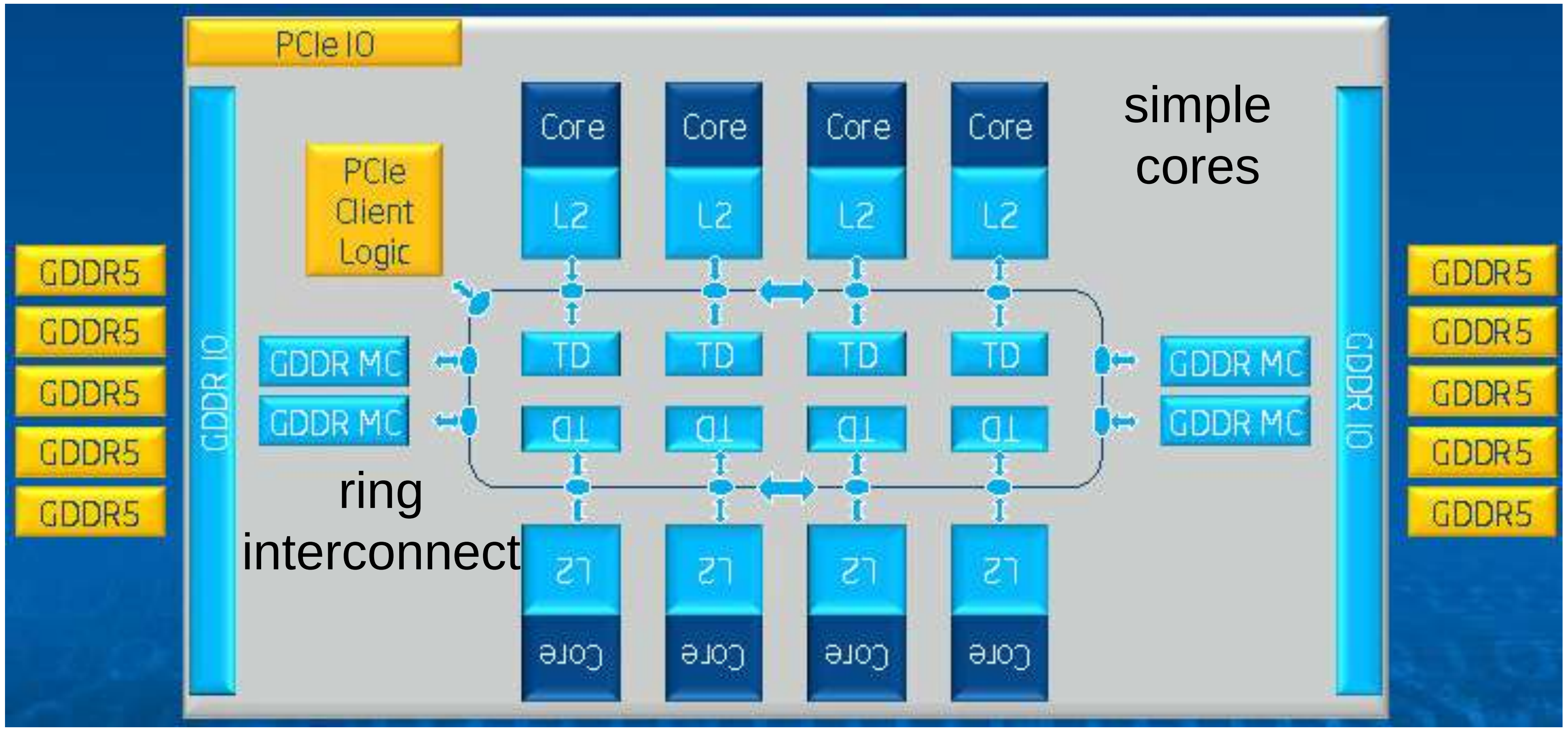
- 3B transistors in 40nm
- 512 CUDA Cores
  - New IEEE 754-2008 floating-point standard
    - FMA
    - 8× the peak double precision arithmetic performance over NVIDIA's last generation
  - 32 cores per SM, 21k threads per chip
- 384b GDDR5, 6 GB capacity
  - 178 GB/s memory BW
- C/M2090
  - 665 GigaFLOPS DP, 6GB
  - ECC Register files, L1/L2 caches, shared memory and DRAM





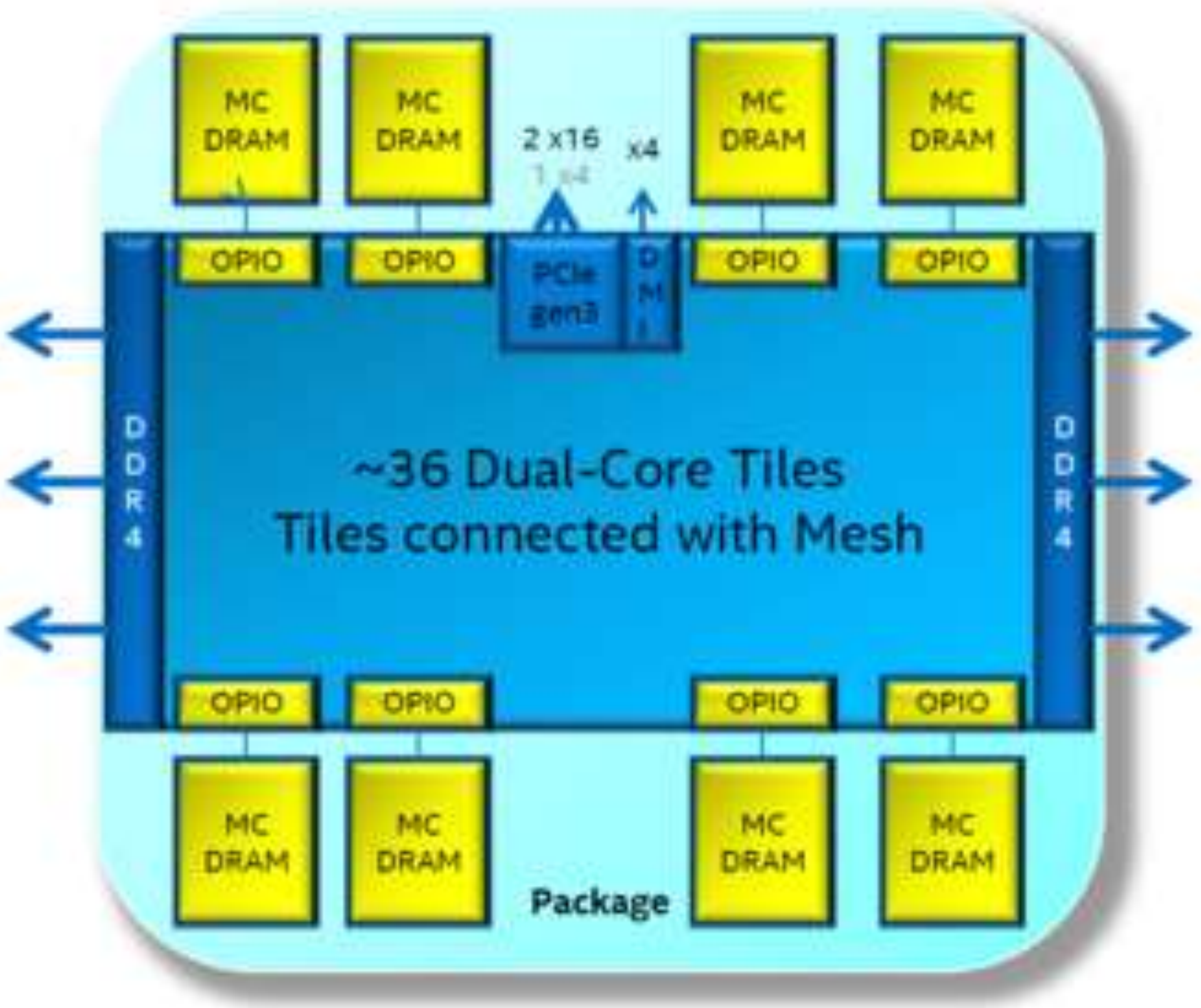
Intel Xeon Phi (Knights Corner)

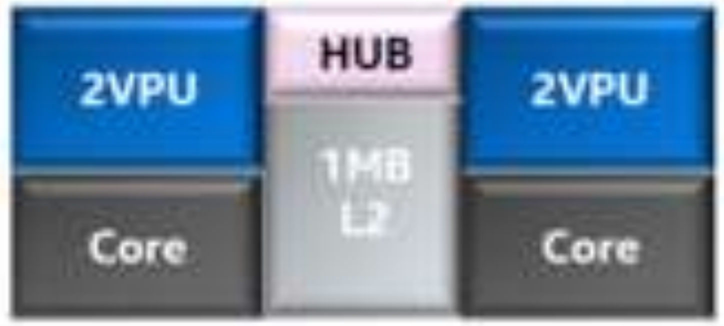
- Many Integrated Cores (MIC) architecture



Intel Xeon Phi (Knights Landing)

- ❑ Changed the interconnection architecture
- ❑ Beefed up the core





- Up to 72 new Intel® Architecture cores
- 36MB shared L2 cache
- Full Intel® Xeon™ processor ISA compatibility through Intel® Advanced Vector Extensions 2
- Extending Intel® Advanced Vector Extensions architecture to 512b (AVX-512)
- Based on Silvermont microarchitecture:
  - 4 threads/core
  - Dual 512b Vector units/core
- 6 channels of DDR4 2400 up to 384GB
- 36 lanes PCI Express® (PCIe®) Gen 3
- 8GB/16GB of extremely high bandwidth on package memory
- Up to 3x single thread performance improvement over prior gen [1,2]
- Up to 3x more power efficient than prior gen [1,2]

Multiprocessors

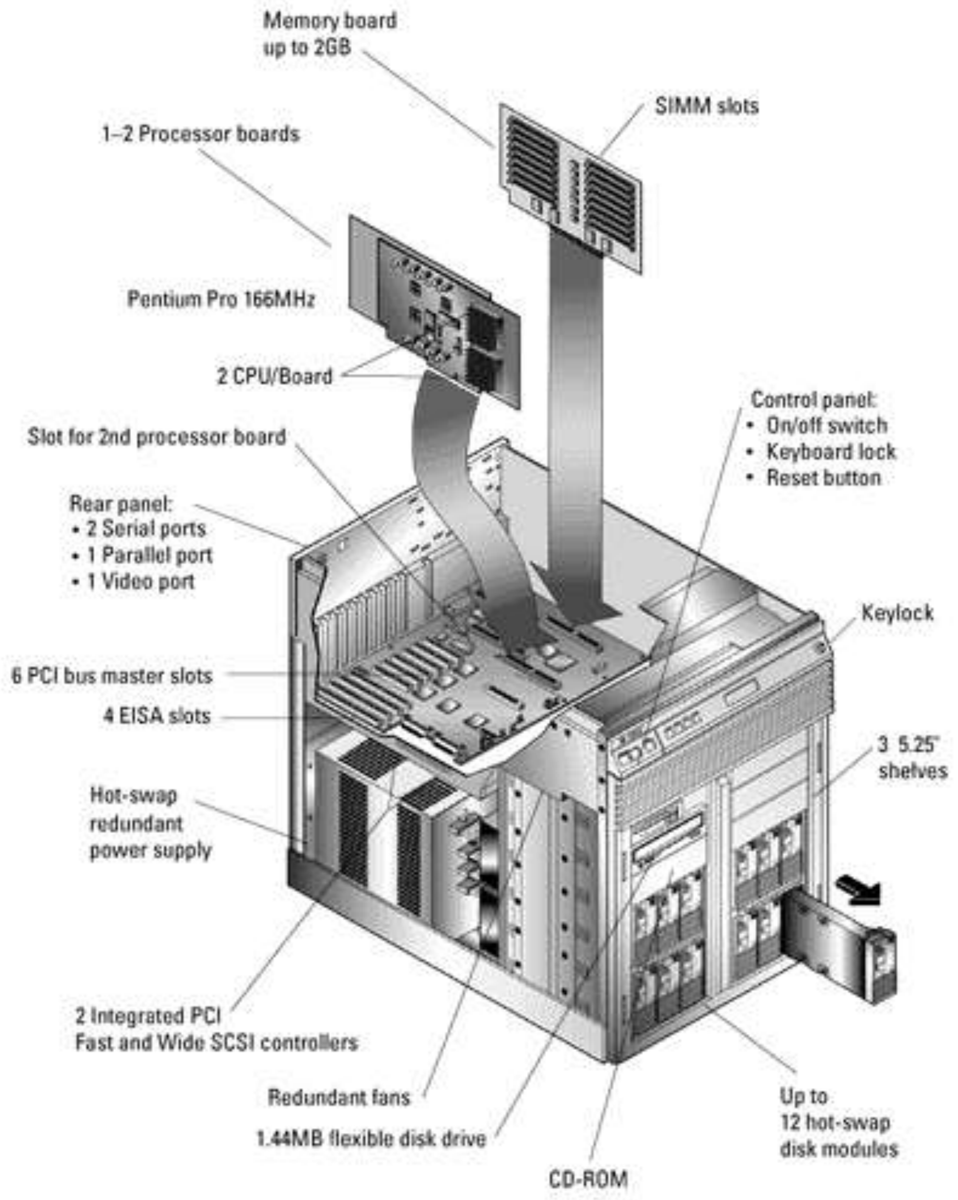
- Limits of multicore and manycore
  - Transistors on a Single Chip
- Increasing performance requires multiple processors
  - Multiple chips
- Think about how to organize these processors
  - Put multiple processors on a single motherboard
  - Provide shared memory hardware
- Hardware optimization (shared memory multiprocessors)
  - Memory hierachy is very important
  - Still have memory consisitency problem that requires hardware
  - Multicore memory hierarchy in each chip and across processors
  - Unified software vision of shared-memory multi-threading

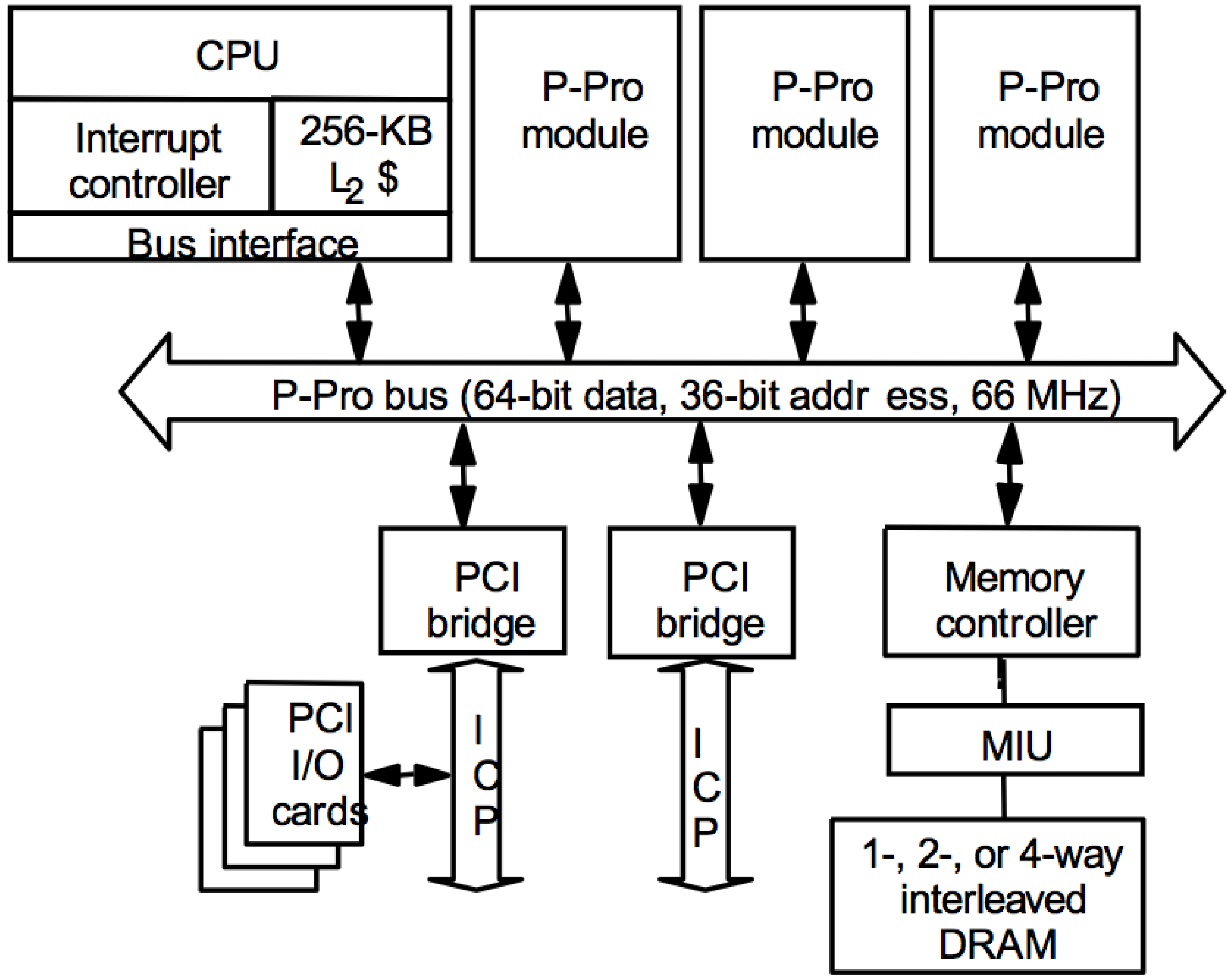
NUMA Node

- Memory within a processor is closer to the cores in that processor
- Accessing memory of another processor has higher latency
- Sharing memory requires more complex memory controller
  - Needs to handle non-uniform memory access
  - Needs to have a cache coherency protocol implemented across processors
- Hardware view
  - Each processor has its memory controller
  - Each processor accesses its own memory locally
  - Using a specific protocol to access the memory of a remote processor (on the same motherboard)
- Hardware has been developed for cache coherency management
- Hardware has been developed for fast data access
  - QPI or Hypertransport
- Again, optimize by better memory placement

Intel Pentium Pro Quad Processor

- All cache coherence and multiprocessing glue in processor module
- Highly integrated, targeted at high volume
- Low latency and bandwidth





Memory Allocation

- Understanding memory allocation policy
- Principle of *first touch*
  - Memory is allocated when a location is first accessed
  - Memory is allocated with respect to pages
    - granularity of a memory page on 4KB on standard x86 machine
- Allocation done with OS memory allocation routines
  - Does not actually touch memory
- When memory is accessed
  - Generating a signal (segmentation fault)
  - Check with the OS to make sure memory is allocated
  - Update table of pages
  - Re-Issue Memory Access

Memory Allocation (2)

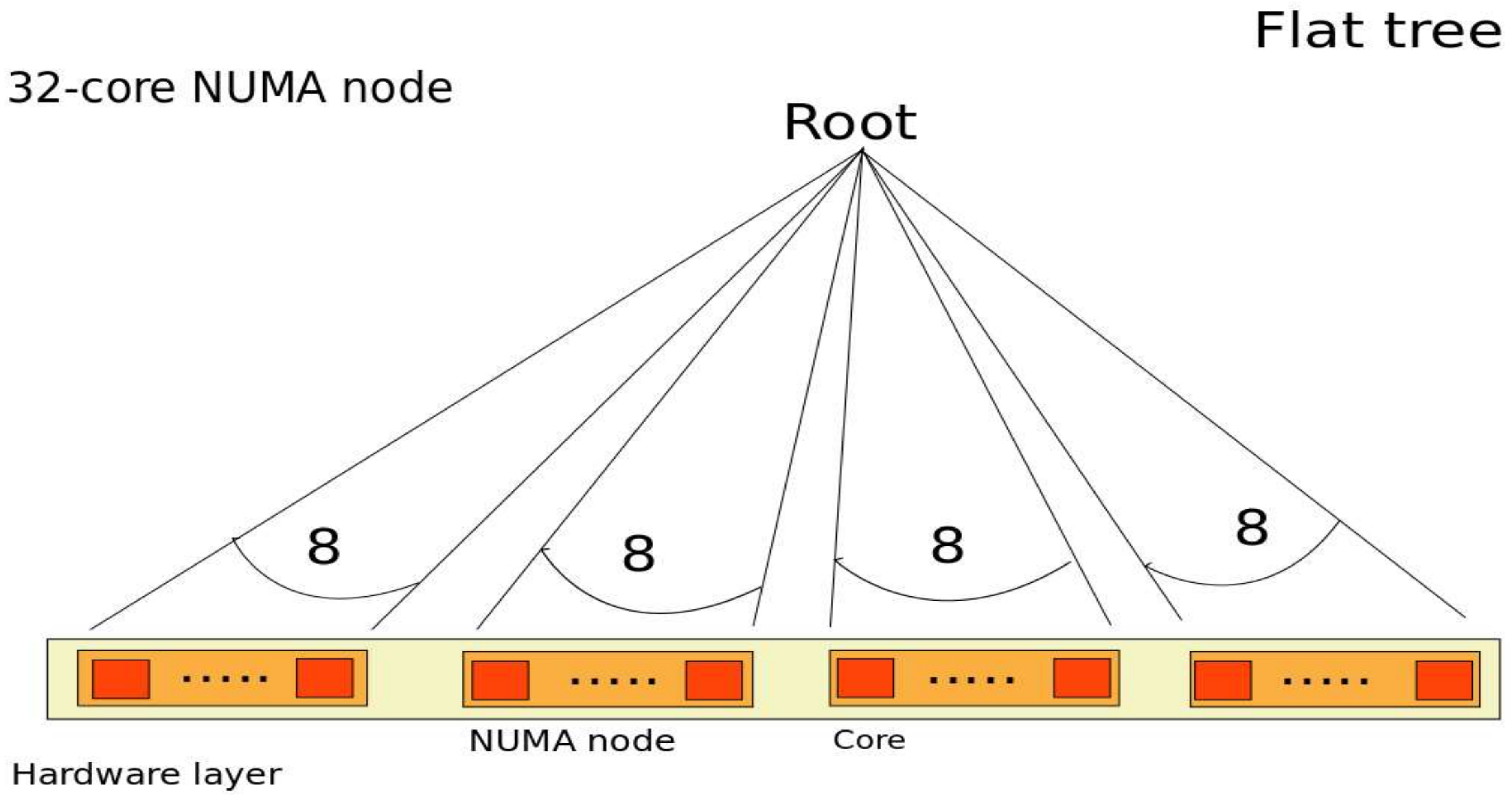
- When is first touch of memory done?
- Which thread touches memory first?
- What is the impact of NUMA?
- Consider the case with an OpenMP application

Synchronisation

- Consider the impact of NUMA on synchronization
- When threads share a variable
  - What happens in the memory system?
- All the cores will access this variable
  - Incurs NUMA latency + cache consistency updates
  - It can reduce performance
- Example of synchronization is a barrier
- Classical Barrier algorithms for shared memory use synchronization variables

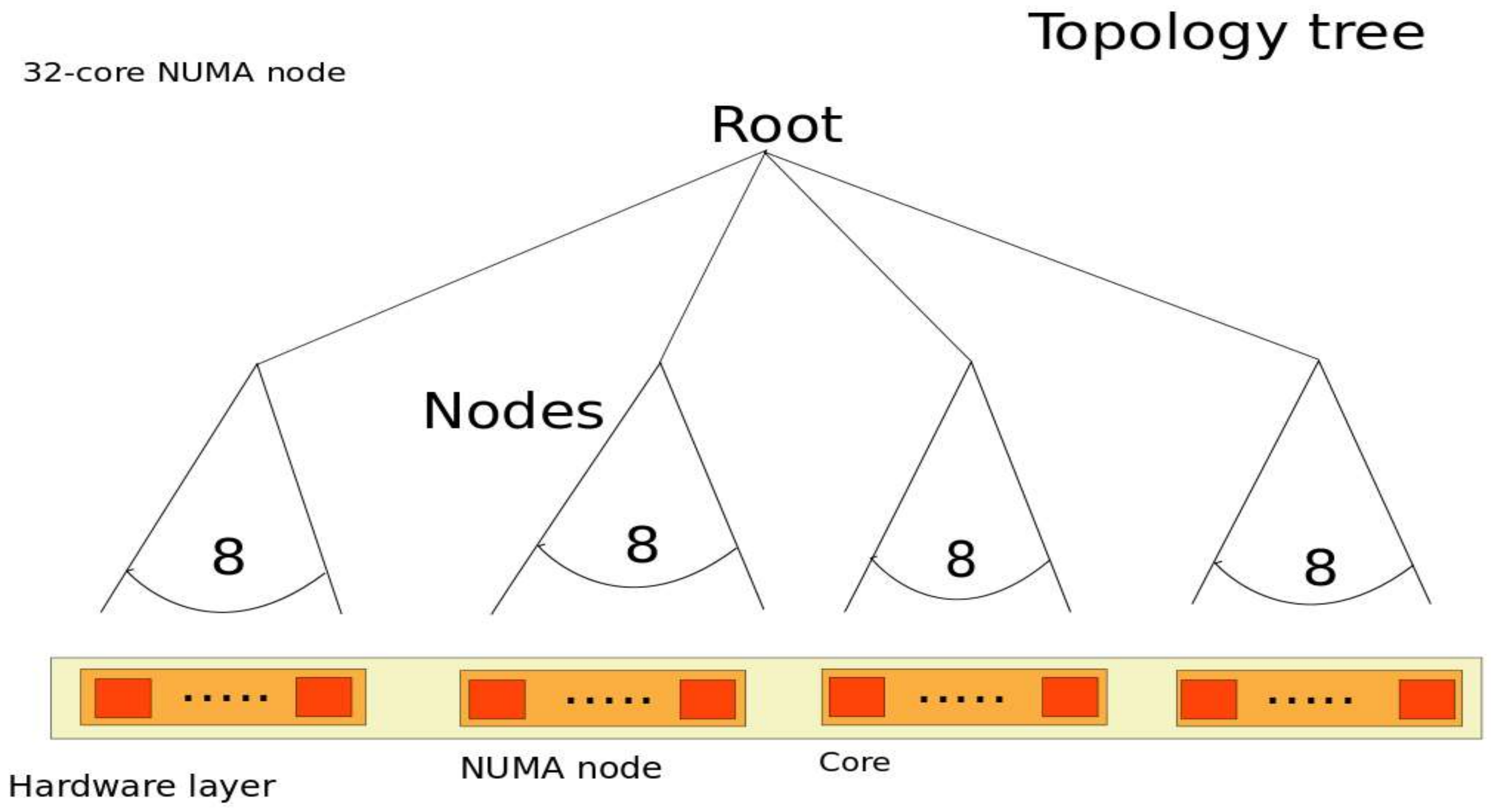
Barrier Based on a Flat Tree

- Simplest structure
- Use a single synchronization variable
- Fast for synchronize a few threads
- Has higher cost for a large number of threads



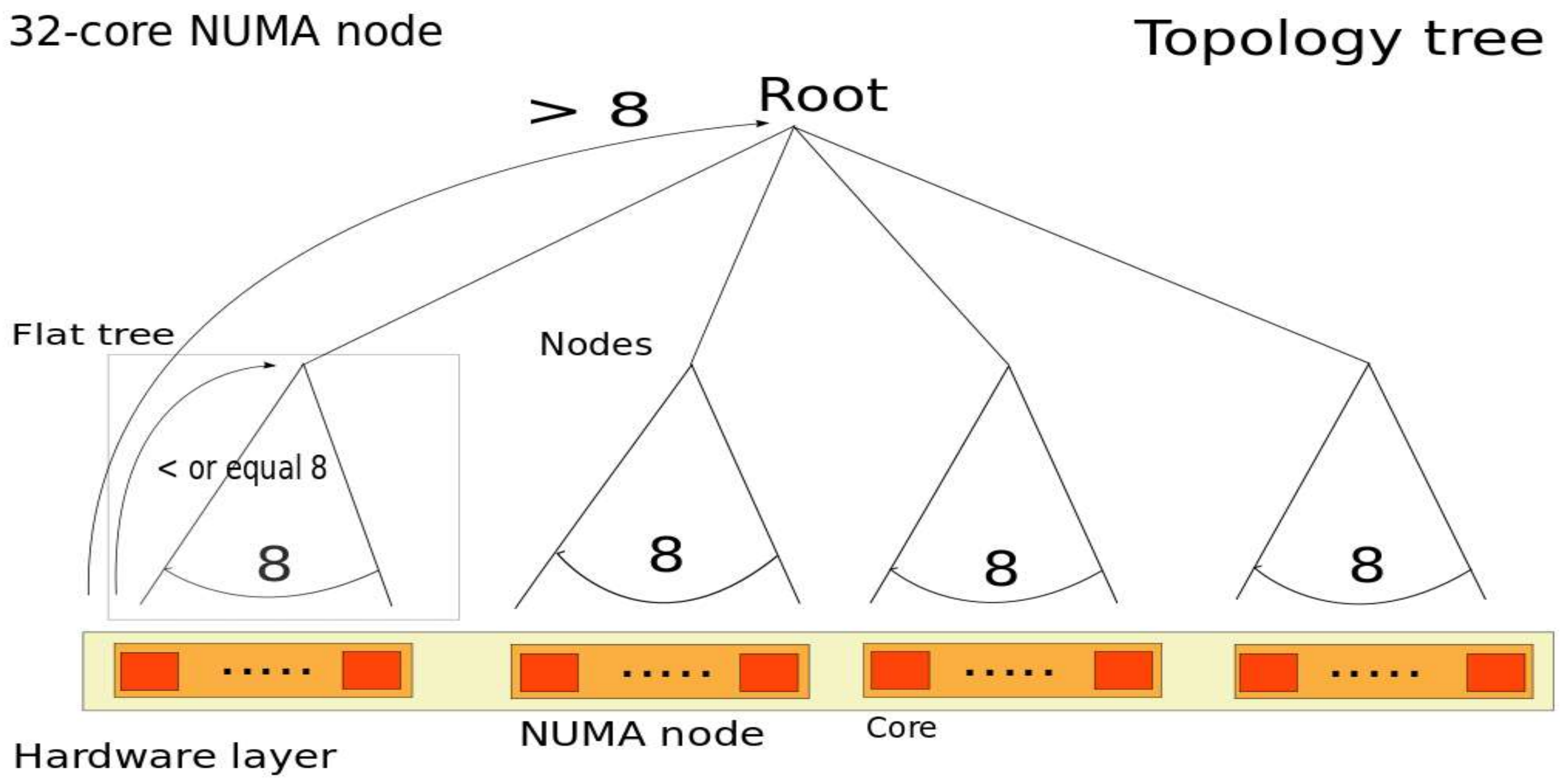
Barrier Approach for Hierarchical Memory

- We can take into account the organization of the multiprocesors node and hierarchical memory
- Implement a better synchronization scheme
  - Use a synchronization tree based on # cores / node
- More parallelism to synchronize a lot of threads
- Surplus for a small number of threads (height of the tree)



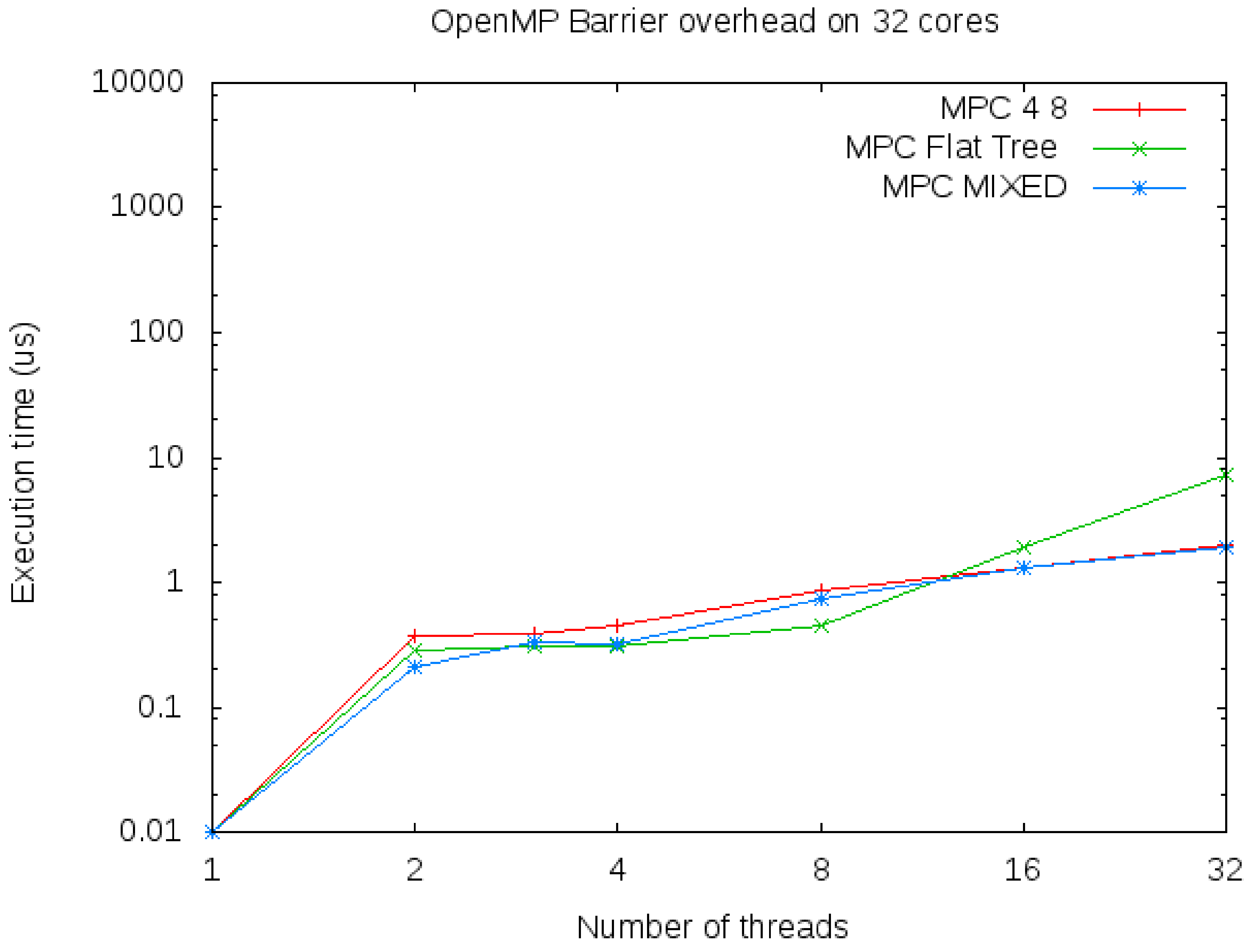
Make Barrier Tree Adaptive

- Idea is to dynamically choose the most suitable root
  - Isolation of a sub-tree regrouping all the threads to be synchronized
  - Depending on the number of threads, choose the most suitable sub-tree



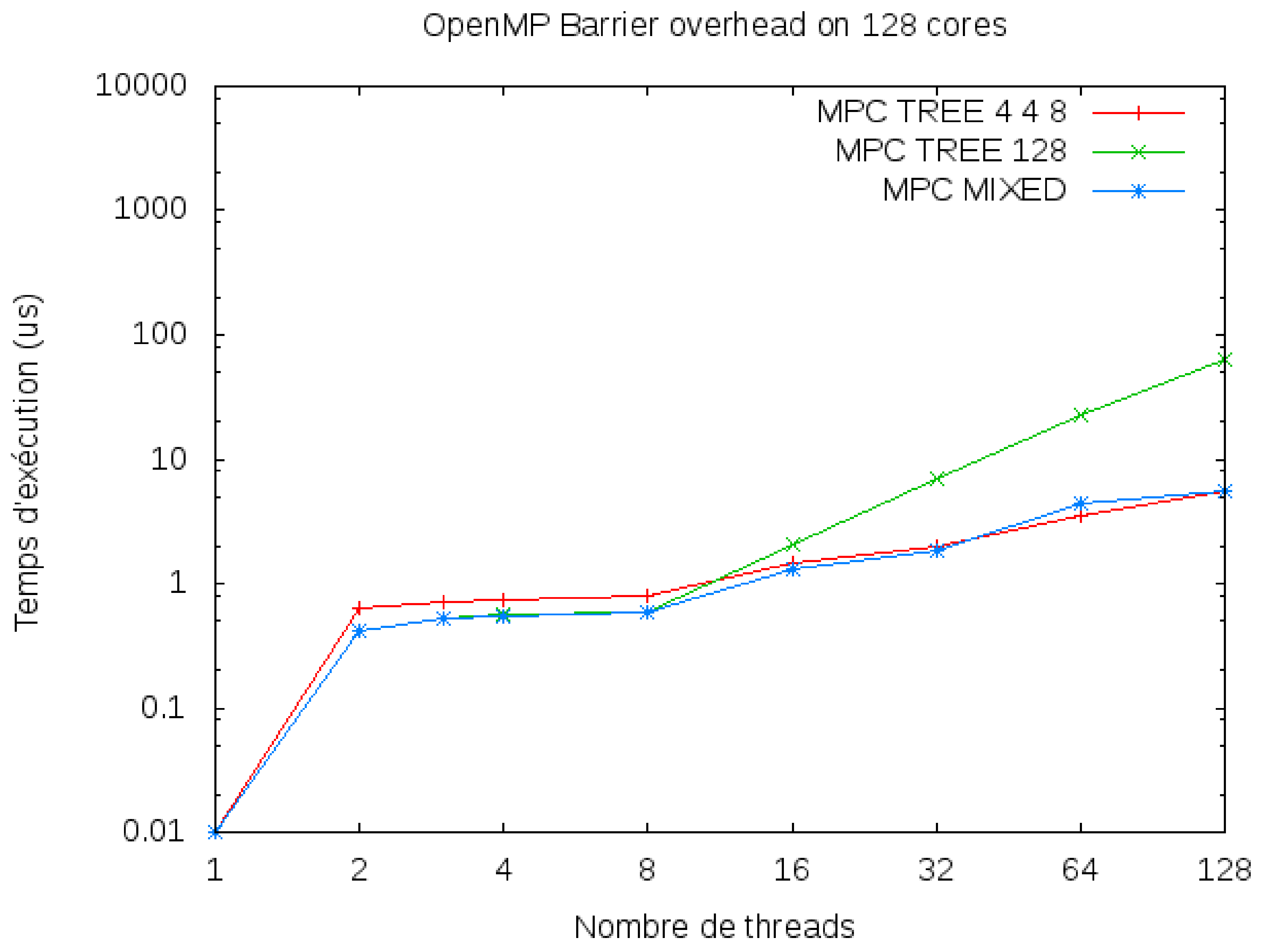
Performance

- OpenMP barrier on 32 cores
- MPC (Multi Processing Computing)



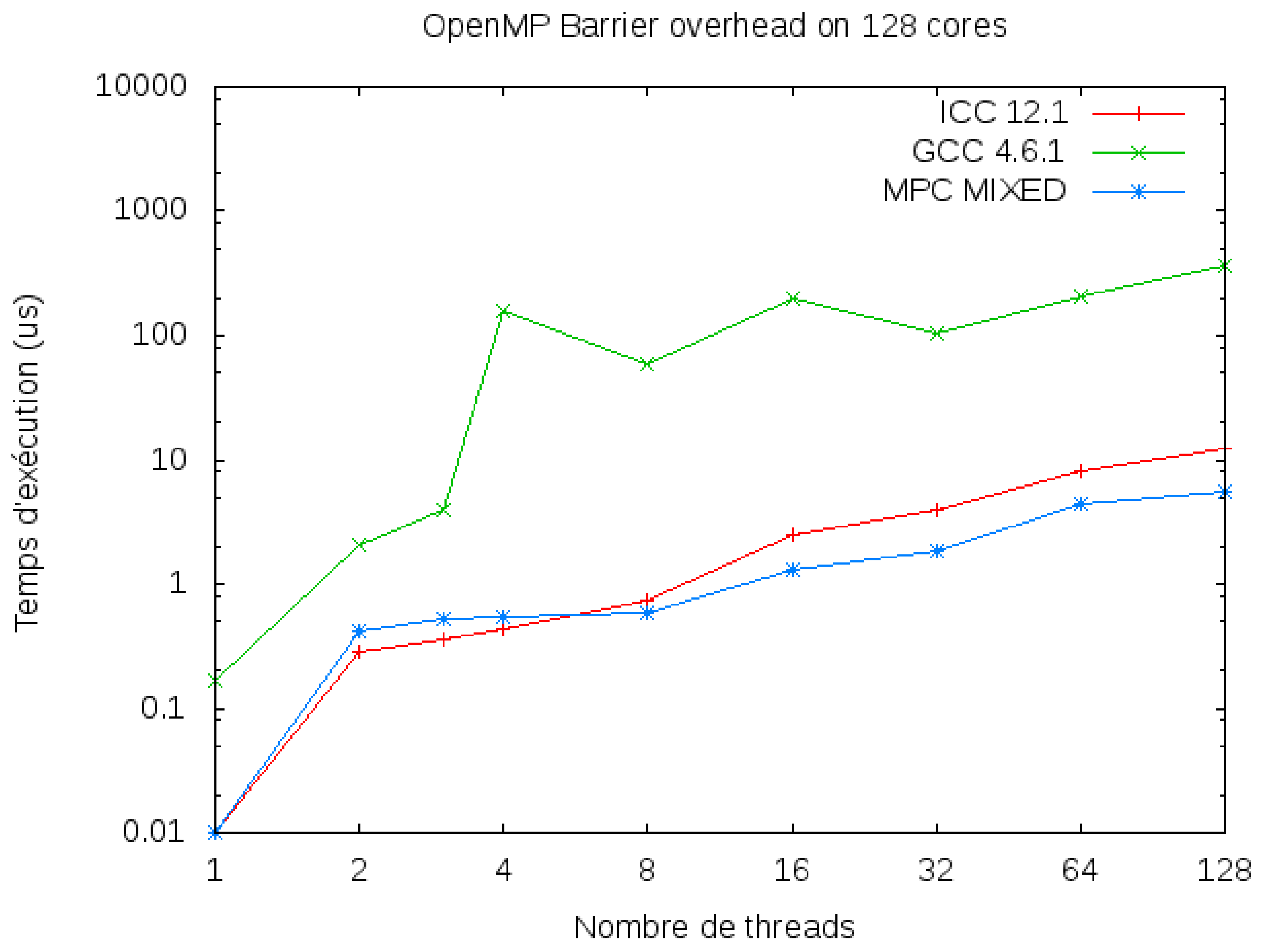
Performance (2)

- OpenMP barrier on 128 cores



Performance (3)

- OpenMP barrier on 128 cores
- Compare MPC with ICC and GCC



High-performance Calculation Node

- Very powerful computing node
- Group of a set of processors sharing memory
- Homogeneous nodes
  - Multiple processors
  - Each processor is multicore
  - Every core is superscalar with execution in disorder
- Heterogeneous nodes
  - Multiple processors
  - Combination of multicore and manycore
  - Processors share memory and/or connected through I/O

- Example: Tera 100, Curie

High-performance Calculation Node (2)

- Very powerful computing node
- Group of a set of processors sharing memory
- Homogeneous nodes
    - Multiple processors
    - Each processor is multicore
    - Every core is superscalar with execution in disorder
- Heterogeneous nodes
    - Multiple processors
    - Combination of multicore and manycore
    - Processors share memory and/or connected through I/O

Conclusion

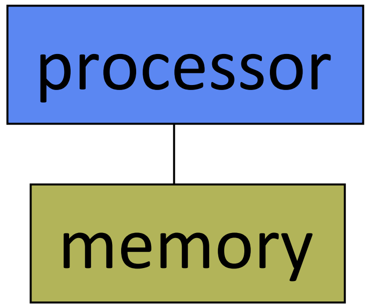
- Architectural evolution
  - Our starting point was the von Neumann model
- Addition of several mechanisms to increase the performance of a computer core
- Duplication of part of the core elements to allow multiple execution flows almost simultaneously (hyperthreading)
- Duplication of most core elements to improve performance (multicore / manycore)
- Grouping of several processors on the same calculation node with access to non-uniform memory (NUMA)

Conclusion (2)

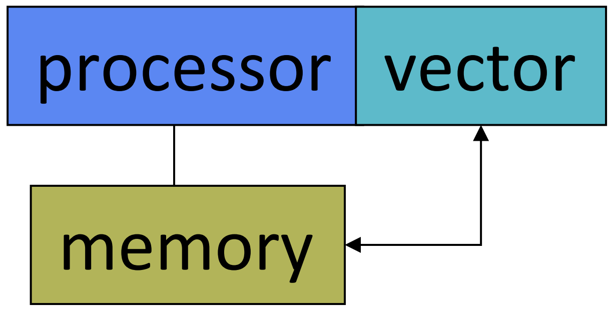
- Current / Future Trends
  - Simplification of computing cores
  - Duplication of functional and/or vector unit
  - Reduction of memory amount by core
- Impact on application development and optimization
  - Need to exploit more cores (hyper-parallelization)
  - Reduced memory consumption of the application
    - recalculate rather than access memory
    - avoid intrinsically consumer programming models
  - Sequential cost is increasingly prohibitive
  - Pay attention system calls
  - Increasingly costly synchronizations

Parallel Architecture Types

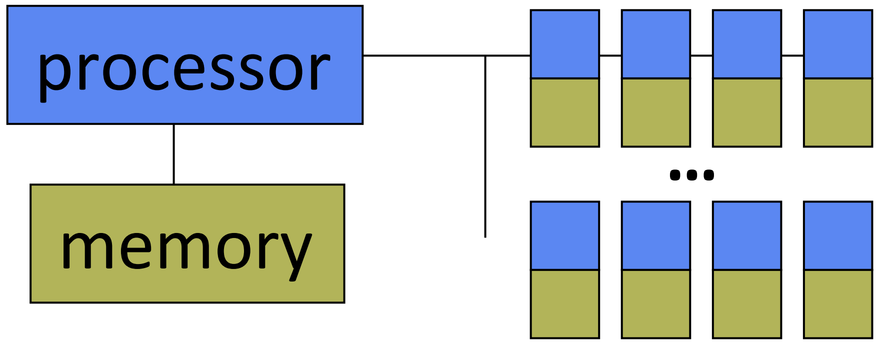
- Uniprocessor
  - Scalar processor



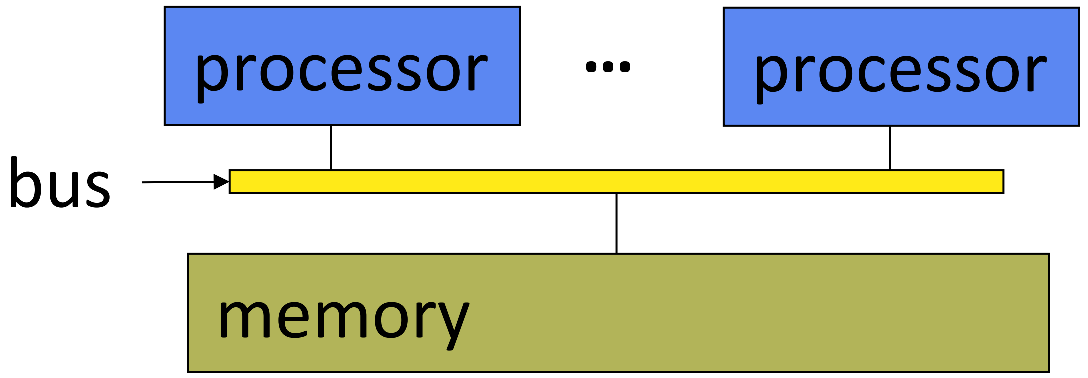
  - Vector processor



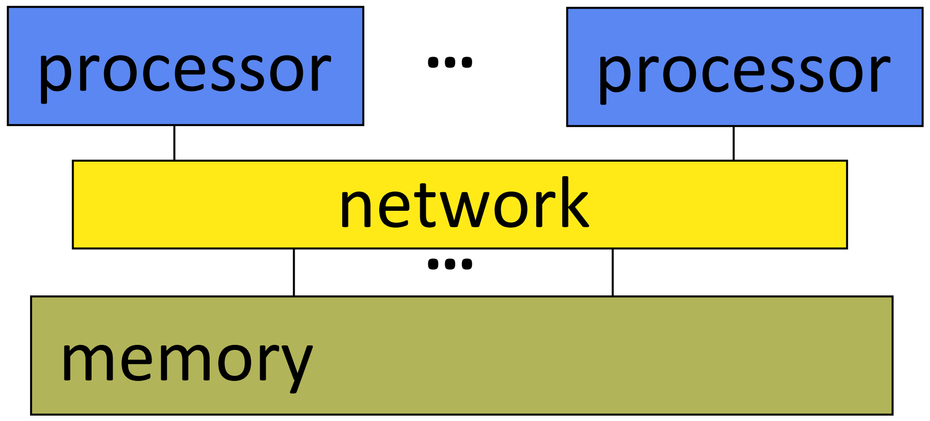
  - Single Instruction Multiple Data (SIMD)



- Shared Memory Multiprocessor (SMP)
  - Shared memory address space
  - Bus-based memory system

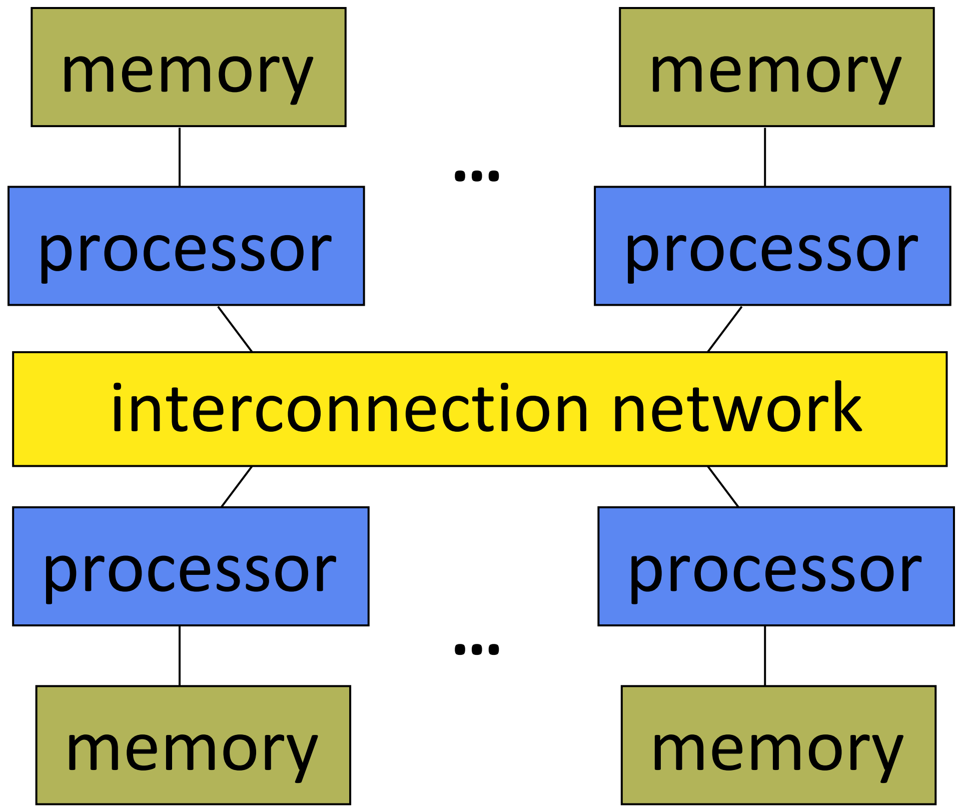


  - Interconnection network



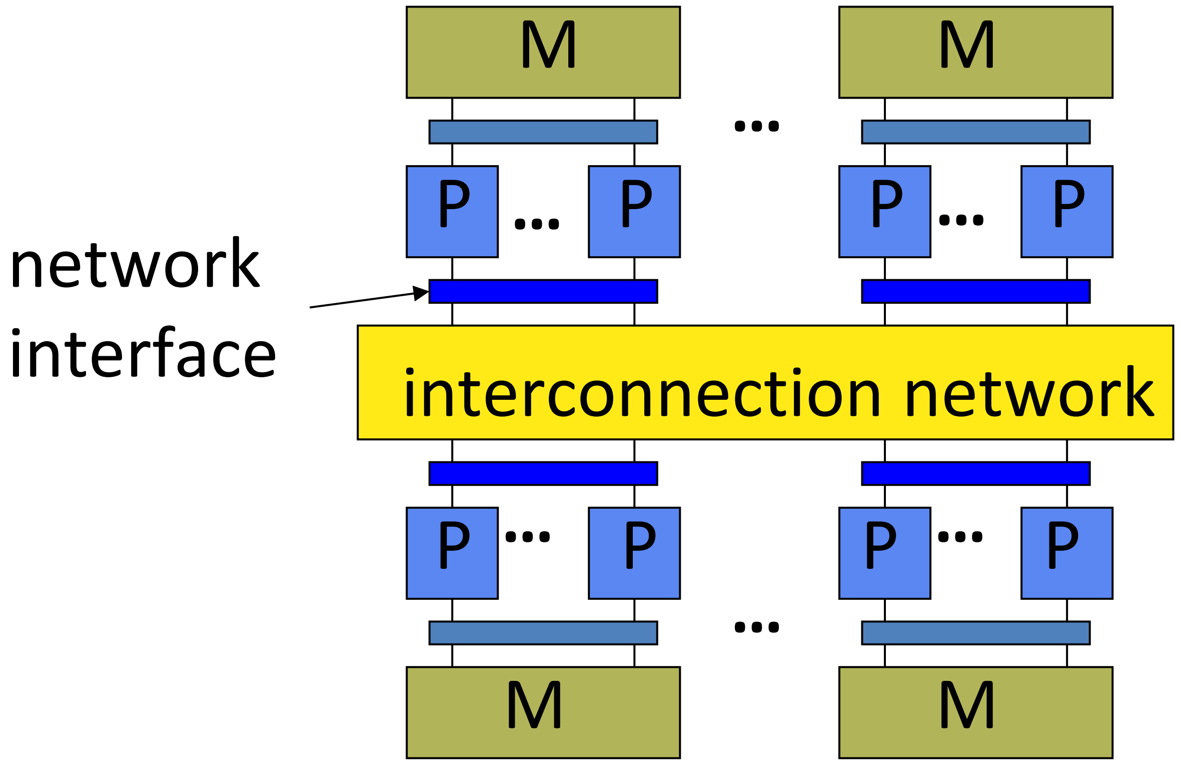
Parallel Architecture Types (2)

- Distributed Memory Multiprocessor
  - Message passing between nodes



  - Massively Parallel Processor (MPP)
    - many, many processors

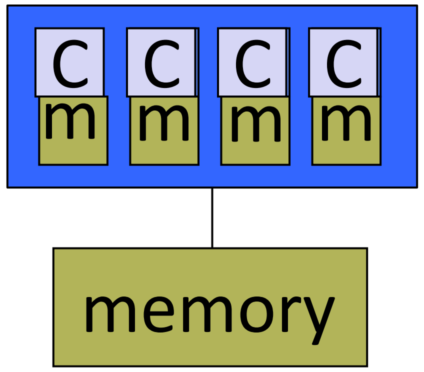
- Cluster of SMPs
  - Shared memory addressing within SMP node
  - Message passing between SMP nodes



  - Can also be regarded as MPP if processor number is large

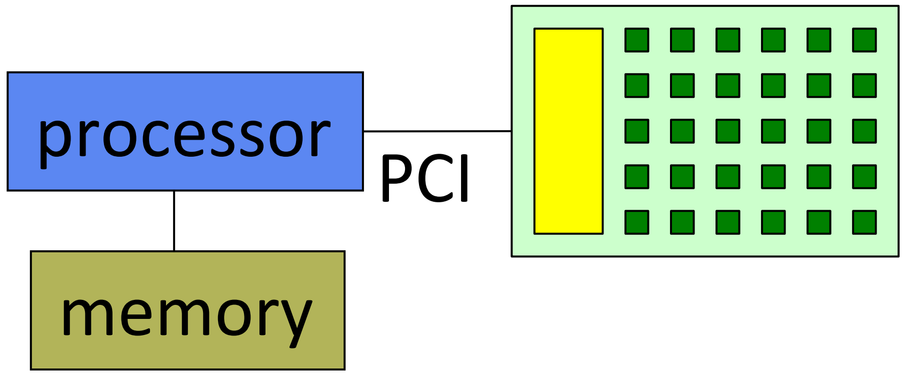
Parallel Architecture Types (3)

- Multicore
  - Multicore processor

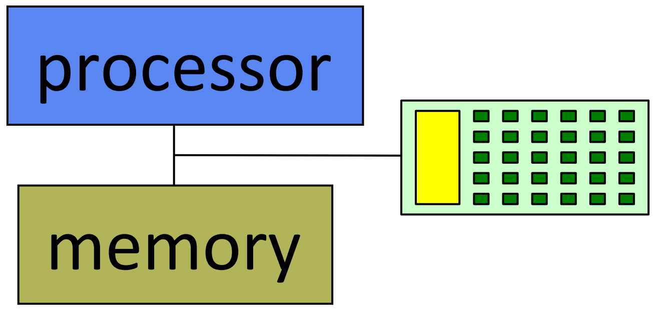


cores can be hardware multithreaded (hyperthread)

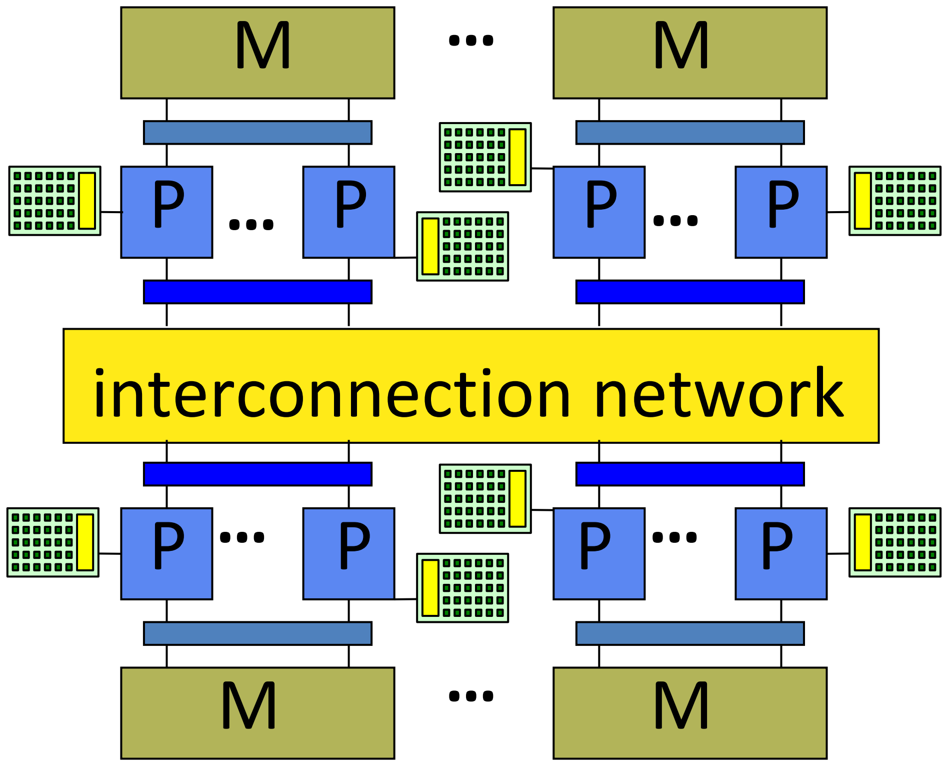
  - Manycore (accelerator)



  - "Fused" multicore + manycore

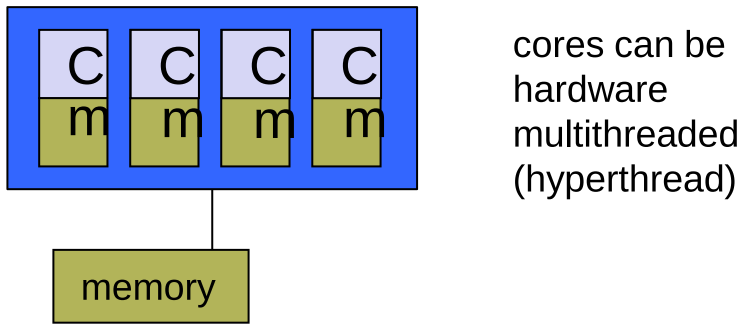


- Multicore SMP+accelerator Cluster
  - Shared memory addressing within SMP node
  - Message passing between SMP nodes
  - Accelerators attached

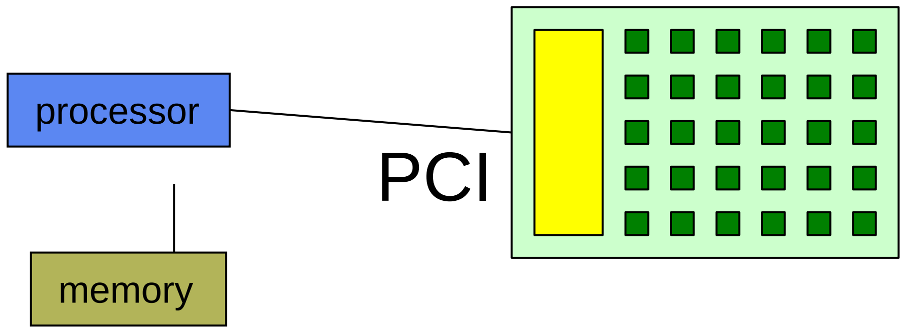


Parallel Architecture Types (3)

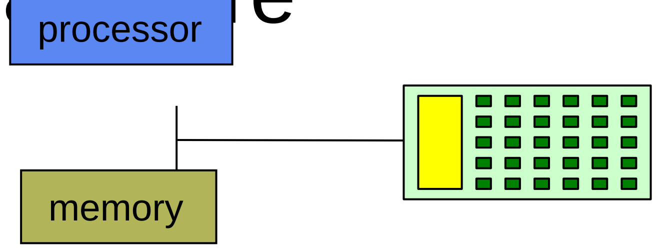
- Multicore
  - Multicore processor



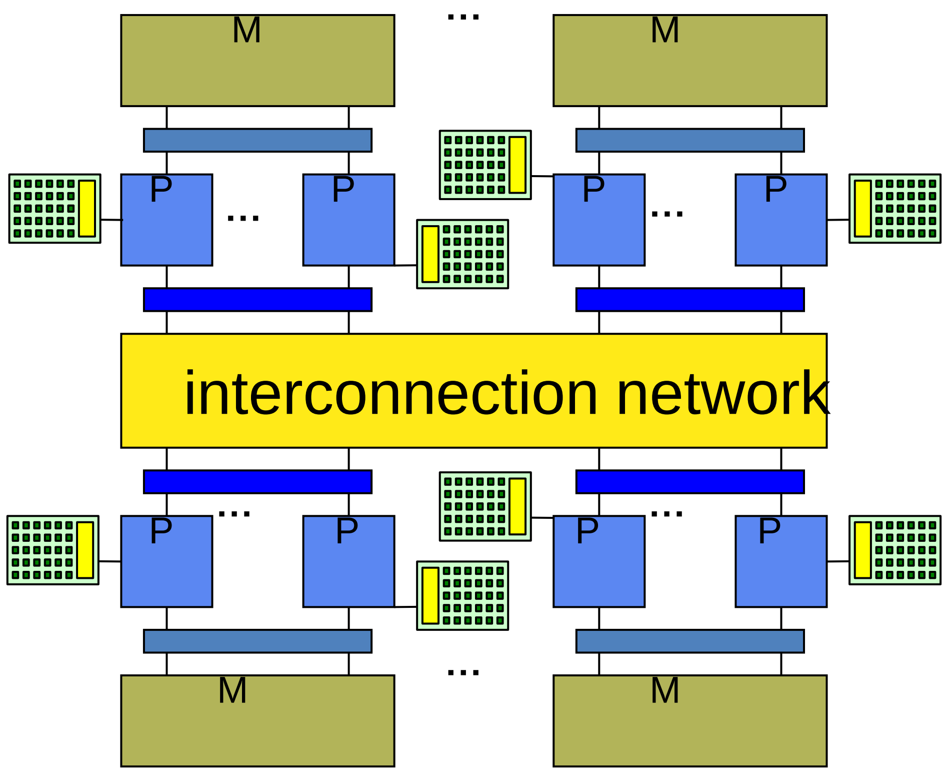
  - Manycore (accelerator)



  - "Fused" multicore + manycore



- Multicore SMP+accelerator Cluster
  - Shared memory addressing within SMP node
  - Message passing between SMP nodes
  - Accelerators attached


M
...
M
P
...
P
P
...
P
interconnection network
P
...
P
P
...
P
...
M
M