RWTH AACHEN UNIVERSITY

Parallel Computer Architecture
- Basics -

Christian Terboven <terboven@rz.rwth-aachen.de>

19.03.2012 / Aachen, Germany

Stand: 15.03.2012

Version 2.3

Agenda

- Processor Microarchitecture
- Shared-Memory Parallel Systems
- Distributed-Memory Parallel Systems (Cluster)
- General Purpose Graphic Processing Units (GPGPUs)
- Summary and Conclusions

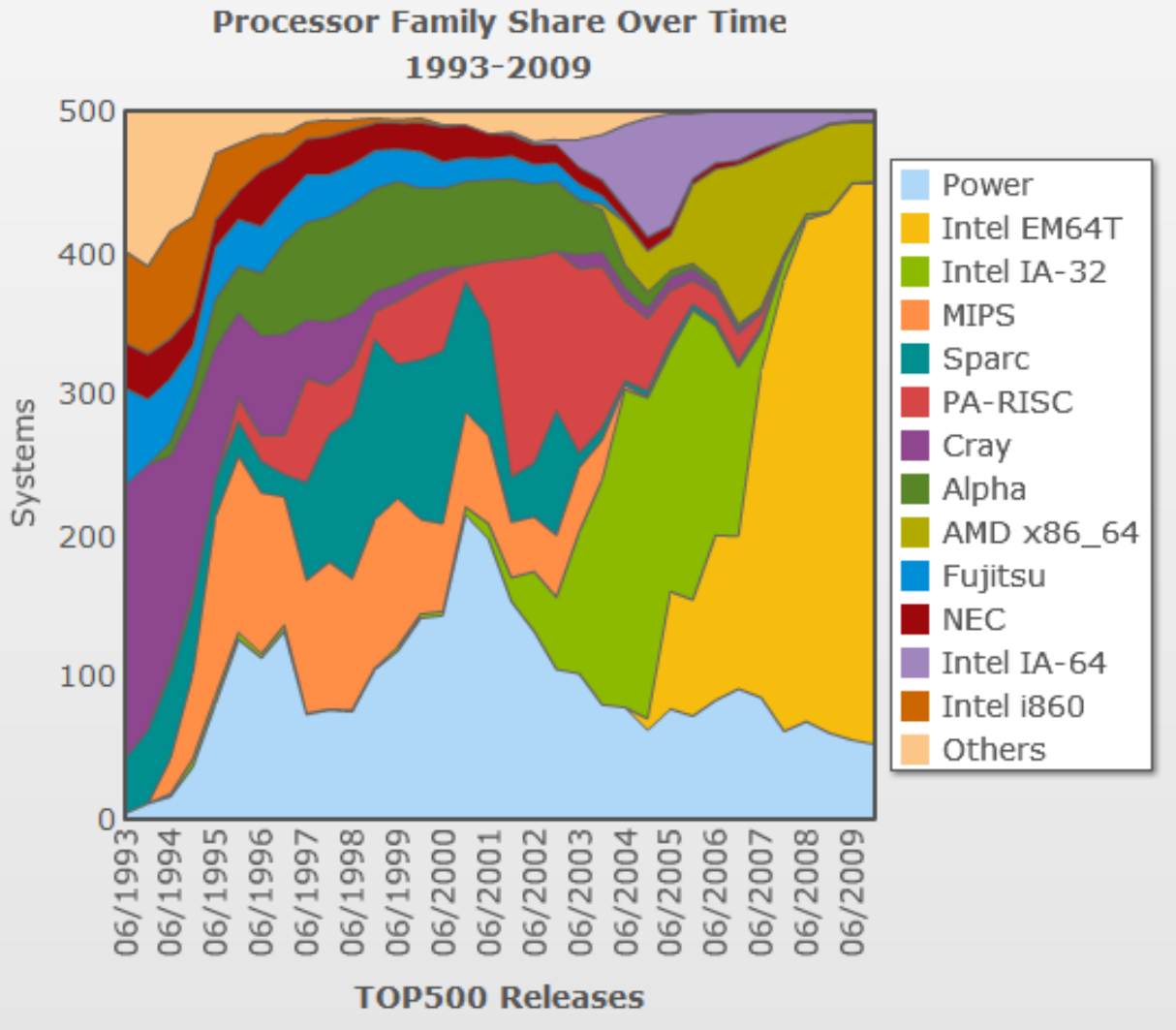
Processor Microarchitecture

- **FLOPS = Floating Point Operation per Second**
  - Megaflops = 10^6 FLOPS
  - Gigaflops = 10^9 FLOPS
  - Teraflops = 10^{12} FLOPS
  - Petaflops = 10^{15} FLOPS
- **Memory Bandwidth: Rate at which data can be read from or stored into a semiconductor memory by a processor.**
- **Memory Latency: Delay incurred when a processors accesses data inside the main memory (data not in the cache).**

LINPACK Benchmark

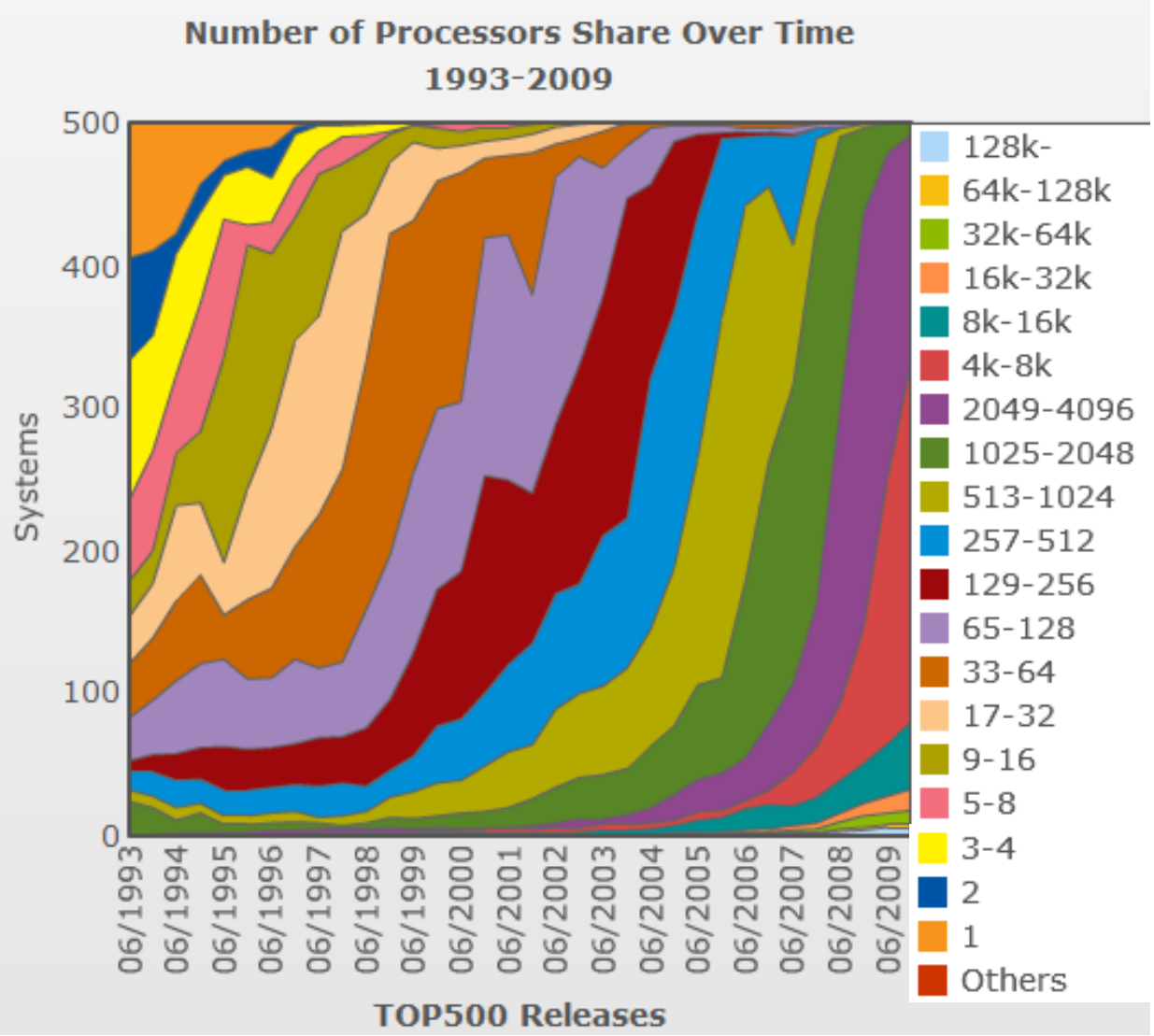
- **The theoretical *peak performance* is defined by the clock rate and cannot be achieved by a real application.**
- **The 500 fastest computer systems of the world are compared in the *Top500 list*:**
  - Updated in June at ISC (Germany) and November at SC (USA)
  - LINPACK benchmark (www.top500.org): Parallel solution of a dense (random) linear equation system, performance measured in FLOPS
  - Currently fastest system: *K computer* (Japan), 11.280.384 GFLOPS
    - Cluster of 18688 nodes with two six-core 2.6 GHz Opterons each
- **Example: Our Bull Cluster with about 1500 nodes**
  - Peak: 292.135,94 GFLOPS
  - Linpack: 219,8 TFLOPS (ranked 32 in June 2011)

Top500 List Statistics: Processor Family



X86 has evolved to the main architecture for HPC systems.

Top500 List Statistics: Number of Processors



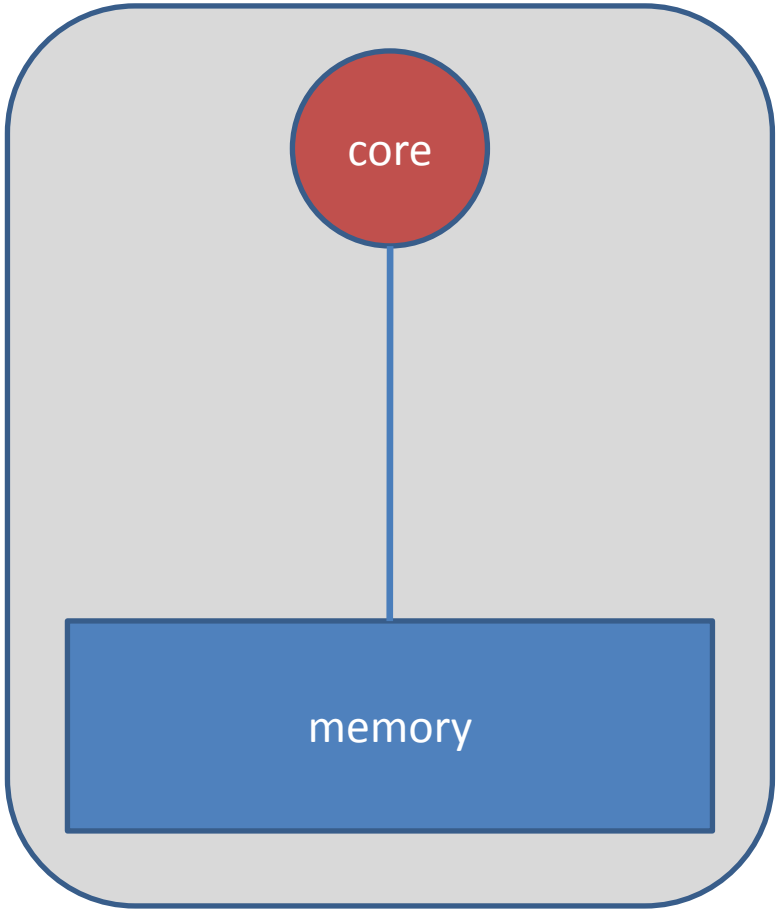
The number of processors per system is exploding.

Properties of modern Microarchitectures

- **The program code (instructions) of the high level language (i.e. C/C++, Fortran) is translated into machine code by a compiler.**
- **The instructions are fetched from the main memory, decoded and executed in multiple steps (Pipelining). Conflicts are detected automatically and might lead to stall cycles.**
- **Modern (superscalar) processors are capable of executing multiple instructions in parallel (ILP = Instruction Level Parallelism).**
  - CPI = Clocks per Instruction, usually 0.5 to 2.0
  - Loops have the highest potential for efficient execution

Single Processor System (dying out) (1/2)

- **Processor**
  - Fetch program from memory
  - Execute program instructions
  - Load data from memory
  - Process data
  - Write results back to memory
- **Main Memory**
  - Store program
  - Store data
- **Input / Output is not covered here!**



Pipelining (1/2)

- ***Pipelining*: An implementation technique whereby multiple instructions are overlapped in execution (think of an assembly linefor automobiles).**
  - Throughput: Number of instructions per time interval
  - Speedup of pipelining: $\frac{\text{Time per instructions on unpipelined machine}}{\text{Number of pipeline stages}}$

- **Example: Assume any (RISC) instruction can be implemented in at most 5 clock cycles:**
  - Instruction fetch cycle (IF)
  - Instruction decode / register fetch cycle (ID)
  - Execution / effective address cycle (EX)
  - Memory access (MEM)
  - Write-back cycle (WB)

Pipelining (2/2)

- **Pipeline model of example architecture: On each clock cycle, another instruction is fetched and begins its 5-cycle exec.:**

Instruction Number	1	2	3	4	5	6	7	8	9
Instruction i	IF	ID	EX	MEM	WB				
Instruction i+1		IF	ID	EX	MEM	WB			
Instruction i+2			IF	ID	EX	MEM	WB		
Instruction i+3				IF	ID	EX	MEM	WB	
Instruction i+4					IF	ID	EX	MEM	WB

- **The major problems of pipelines:**
  - *Structural hazards*: Resource conflicts when hardware cannot support all possible combinations of overlapped instructions
  - *Data hazards*: Instruction depends on result of previous instr.
  - *Control hazards*: Branches or other interrupting events

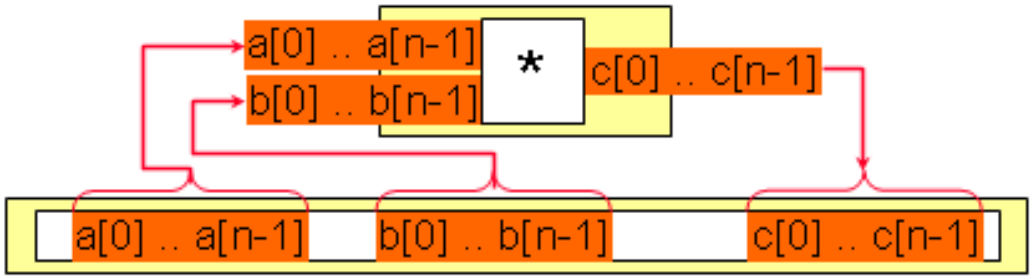
→ Any hazard leads to a pipeline stall.

Digression: Array Processor (Vector)

- SIMD = Single Instruction Multiple Data
- One instruction is applied to an array (vector) of operands
- Data is streamed from memory, saved in a vector register, operation is applied, data is streamed back to memory
  - No issues with caches
  - Extremly high memory bandwidth

Einfache Programmschleife:

```
double a[n], b[n],  c[n];
        :
for (i=0; i<n; i++)
  c[i] = b[i] * a[i];
        :
```

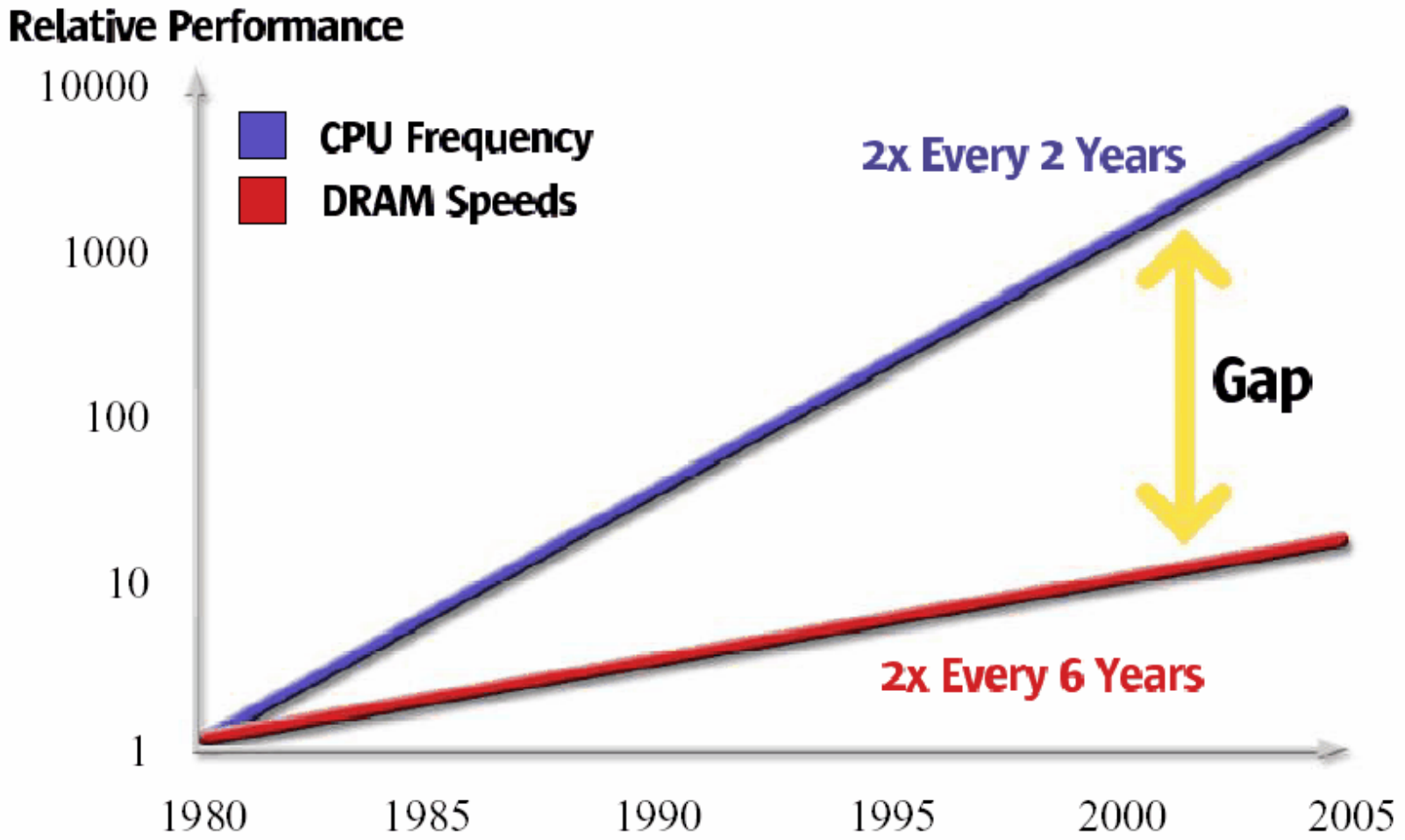


Vector Processor

Memory

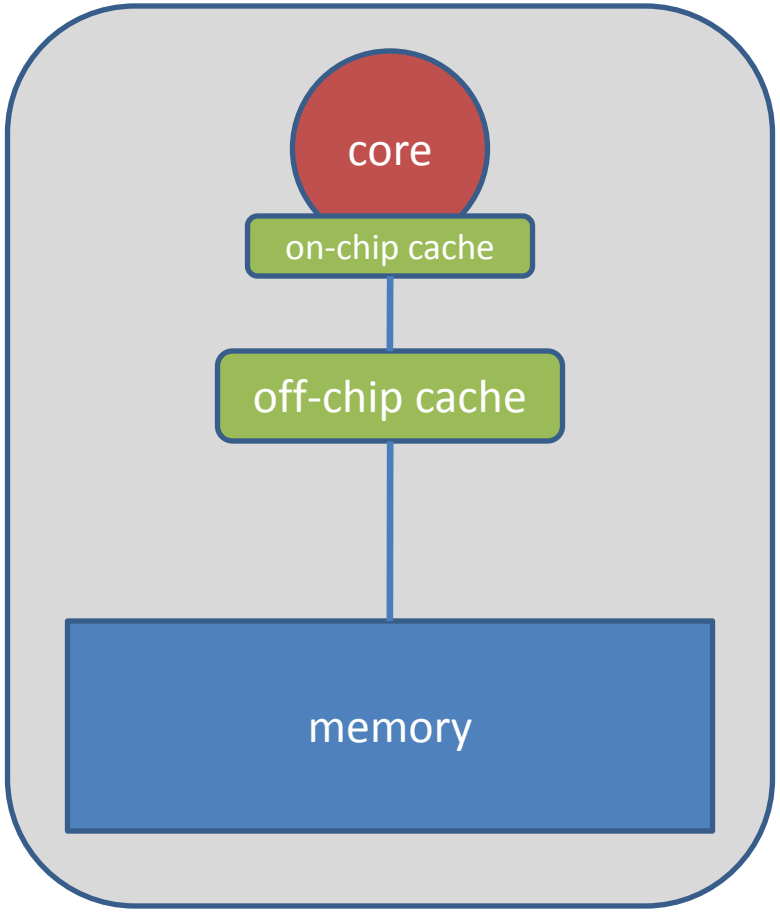
Memory Bottleneck

- There is a gap between core and memory performance.



Single Processor System (dying out) (2/2)

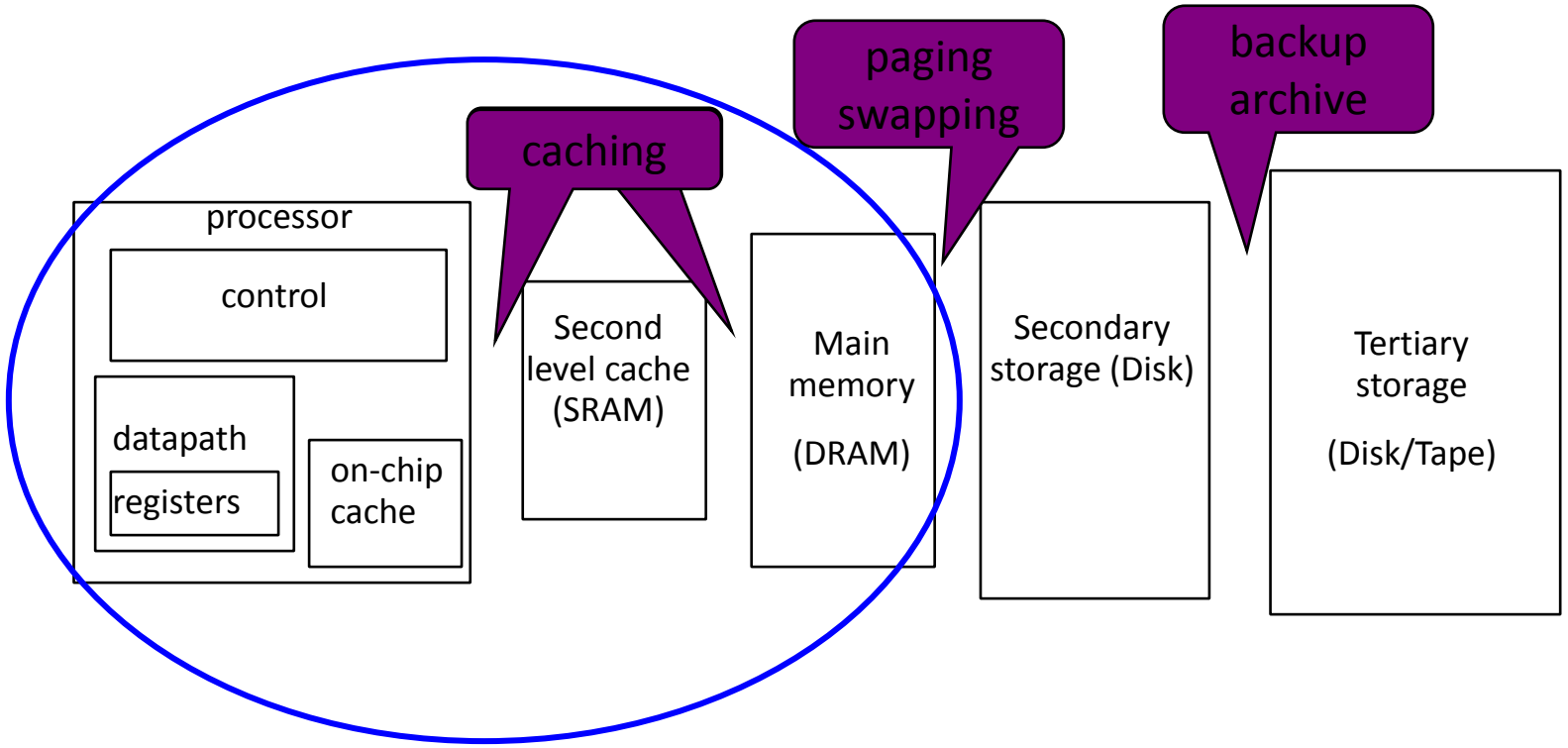
- **CPU is fast**
  - Order of 3.0 GHz
- **Caches:**
  - Fast, but expensive
  - Thus small, order of MB
- **Memory is slow**
  - Order of 0.3 GHz
  - Large, order of GB



- **A good utilization of caches is crucial for good performance of HPC applications!**

Memory Hierarchy

- **Since a large and fast memory is not feasible, the memory hierarchy has evolved and is getting deeper and deeper ...**



Latency Dimension:	nsec	10 nsec	100 nsec	10 msec	10 sec
Size:	~32 KB	1-8 MB	1-100 GB	Tera-/Petabytes	Peta-/Exabytes

Visualizing the Memory Hierarchy (1/2)

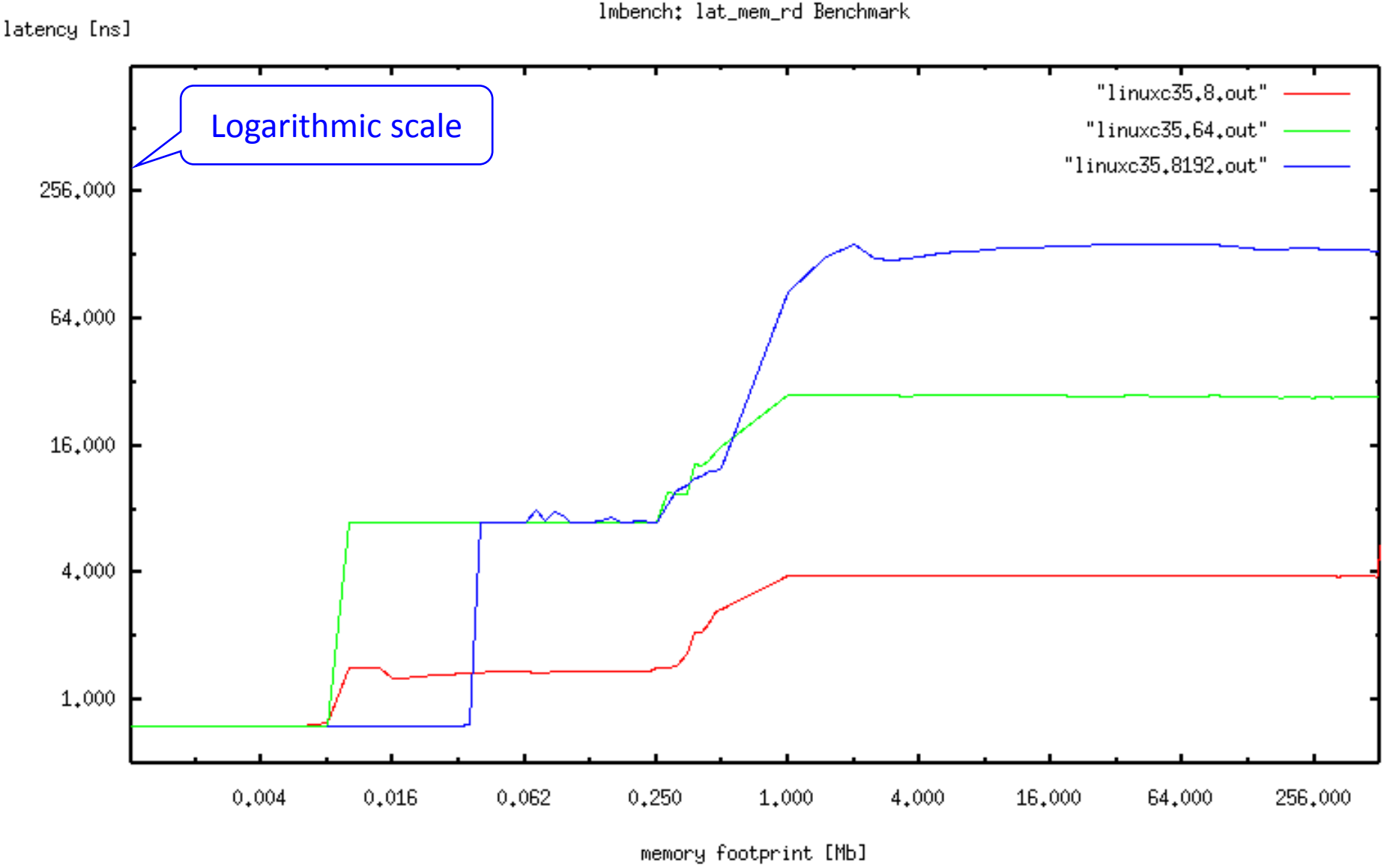
```
addr = (char *) alloc(512*1024*1024);
...
/* create list of pointers */
for ( ... ) {
        p = (char **) &addr[i];
        *p = &addr[i - stride];
}
```

```
/* pointer chasing */
while ( ... ) {
        p = (char **) *p;
        p = (char **) *p;
        p = (char **) *p;
        p = (char **) *p;
        p = (char **) *p;
        ...
}
```

- **In the memory range *addr* several addresses with distance *stride* are read again and again.**
  - Variation: size of *addr*
  - Variation: size of *stride*
  - Measurement: time -> Latency and Bandwidth

Visualizing the Memory Hierarchy (2/2)

- **Intel Pentium IV Xeon. 2.66 GHz. 533 MHz Frontsidebus**



Memory Model: C/C++ vs. Fortran

- **The order of multi-dimensional arrays (= matrices!) in C/C++ is different from the order in Fortran:**
  - C: int a[2][3]

a[0][0]	a[0][1]	a[0][2]	a[1][0]	a[1][1]	a[1][2]	…	**physical memory**

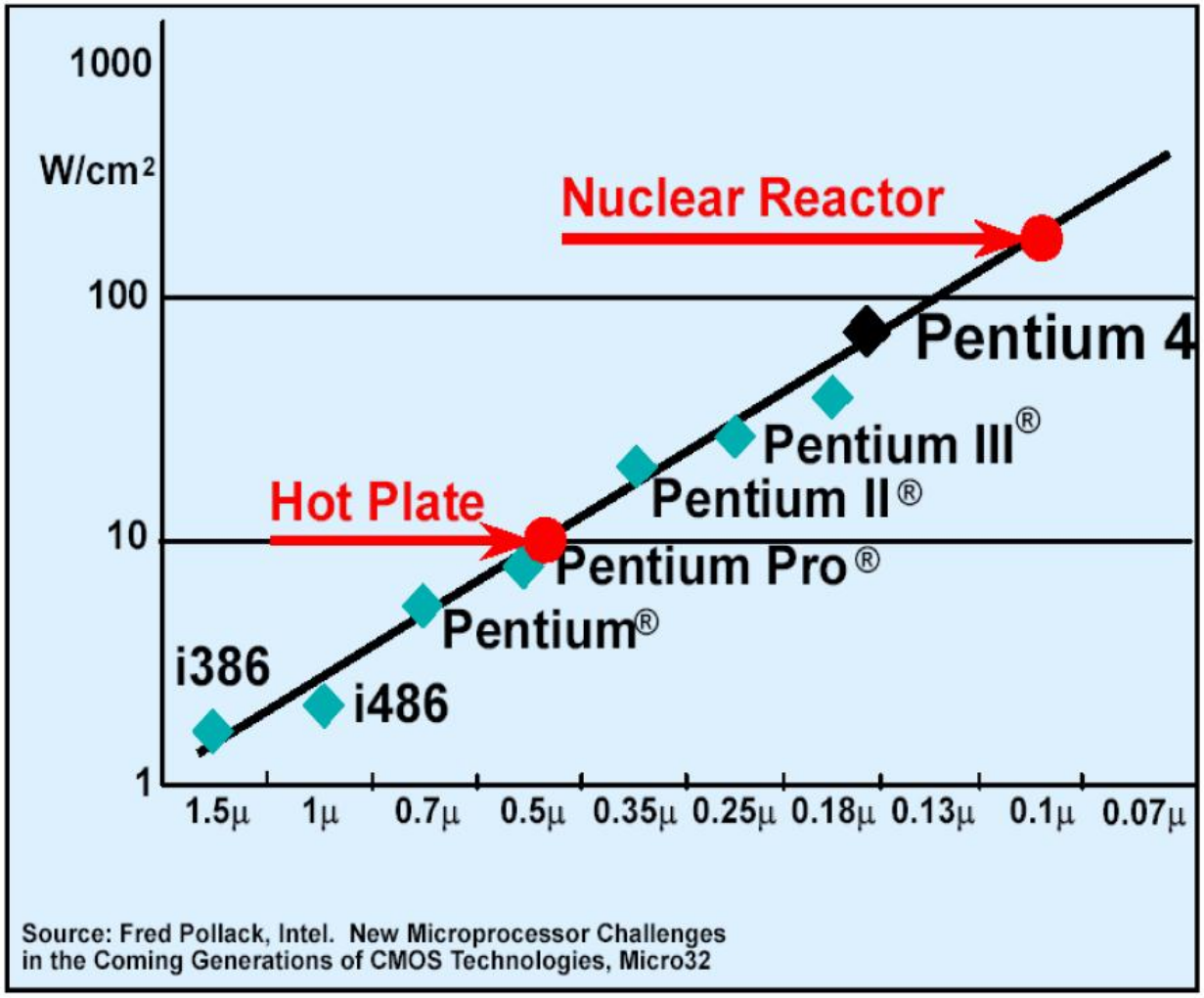
  - Fortran: INTEGER, DIMENSION(2, 3) :: A

a(1,1)	a(2,1)	a(1,2)	a(2,2)	a(1,3)	a(2,3)	…	**physical memory**

- **Thus, the following is equivalent:**
  - C: int i[4][3][2]
  - Fortran: INTEGER, DIMENSION(2, 3, 4) :: I
- **C: Increment in the *rightmost* loop index for next element in cache**
- **Fortran: Incr. in the *leftmost* loop index for next element in cache**

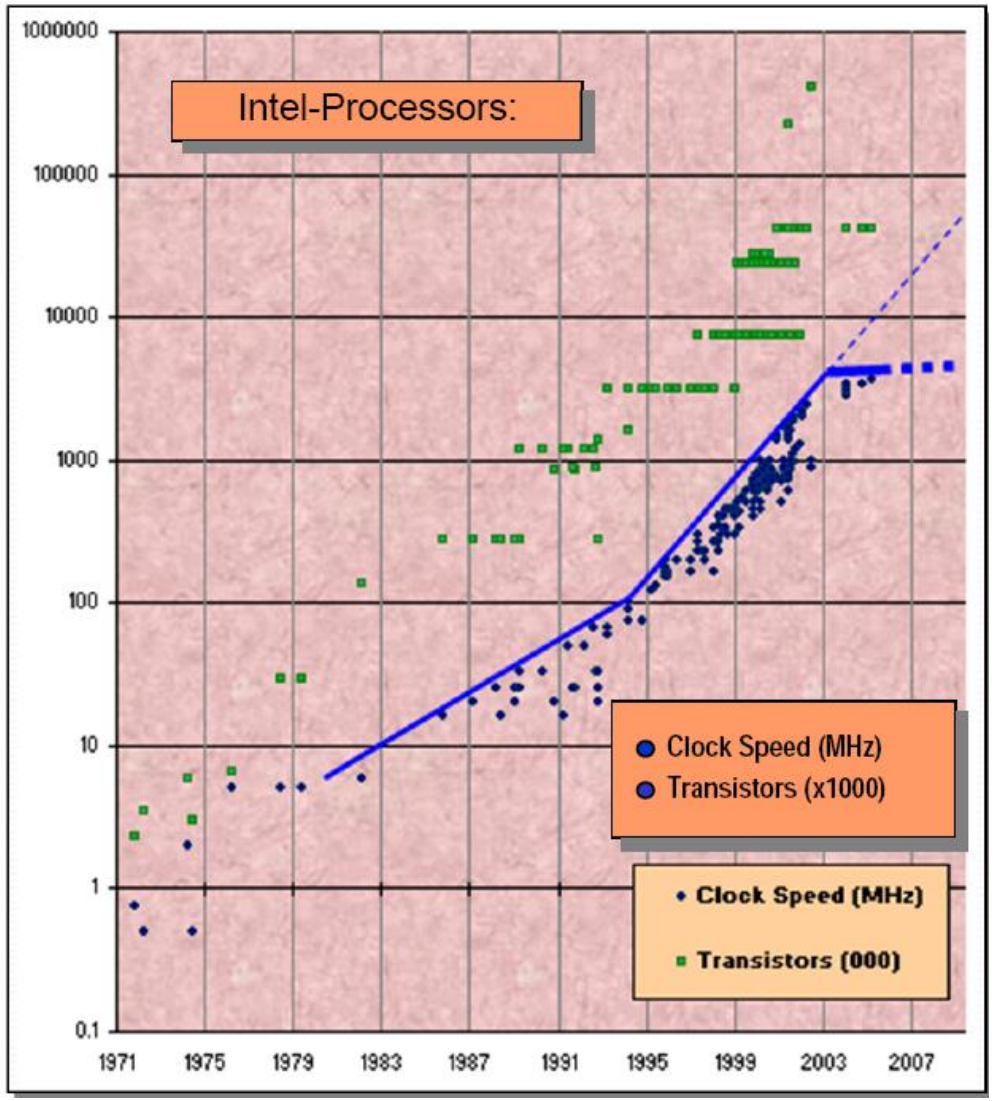
Why can‘t you buy a 4.0 GHz x86 CPU?

- **Because that beast would get too hot!**



Fast clock cycles make processor chips more expensive, hotter and more power consuming.

Moore‘s Law still holds!



The number of transistors on a chip is still doubling every 24 months …

… but the clock speed is no longer increasing that fast!

Instead, we will see many more cores per chip!

Source: Herb Sutter

www.gotw.ca/publications/concurrency-ddj.htm

Chip Multi-Threading (CMT)

- **Traditional single-core processors can only process one thread at a time, spending a majority of time waiting for data from memory**

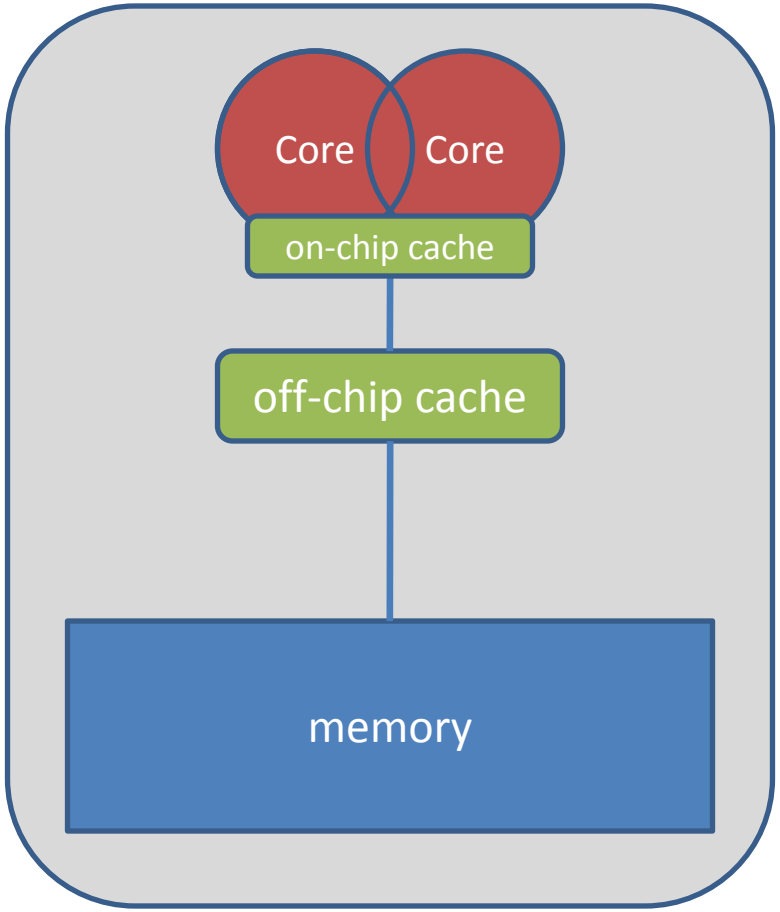
- **CMT refers to a processor‘s ability to process multiple software threads. Such capabilities can be implemented using a variety of methods, such as**
  - Having multiple cores on a single chip:

    Chip Multi-Processing (CMP)
  - Executing multiple threads on a single core:

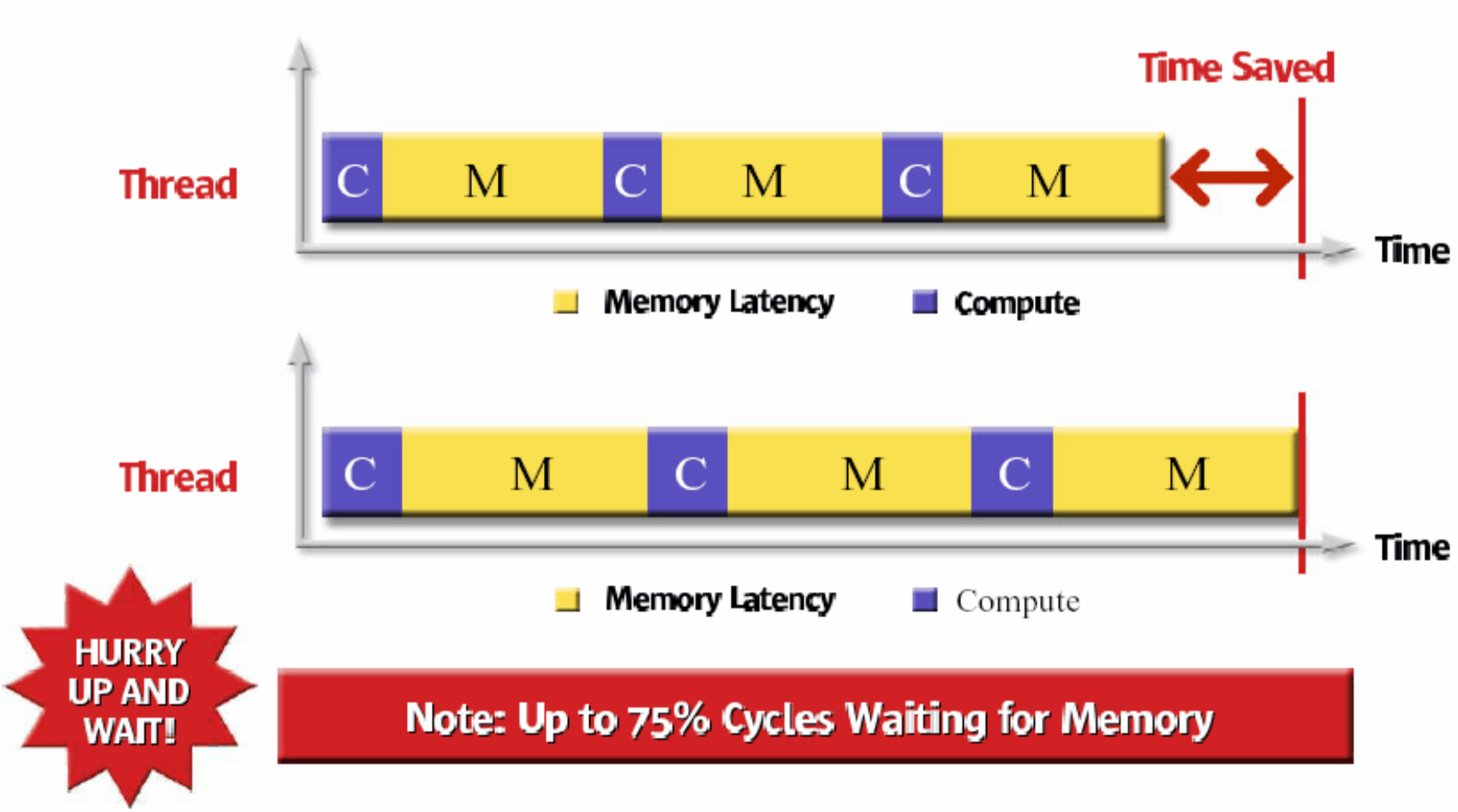
    Simultaneous Multi-Threading (SMT)
  - A combination of both CMP and SMT.

Dual-Core Processor System

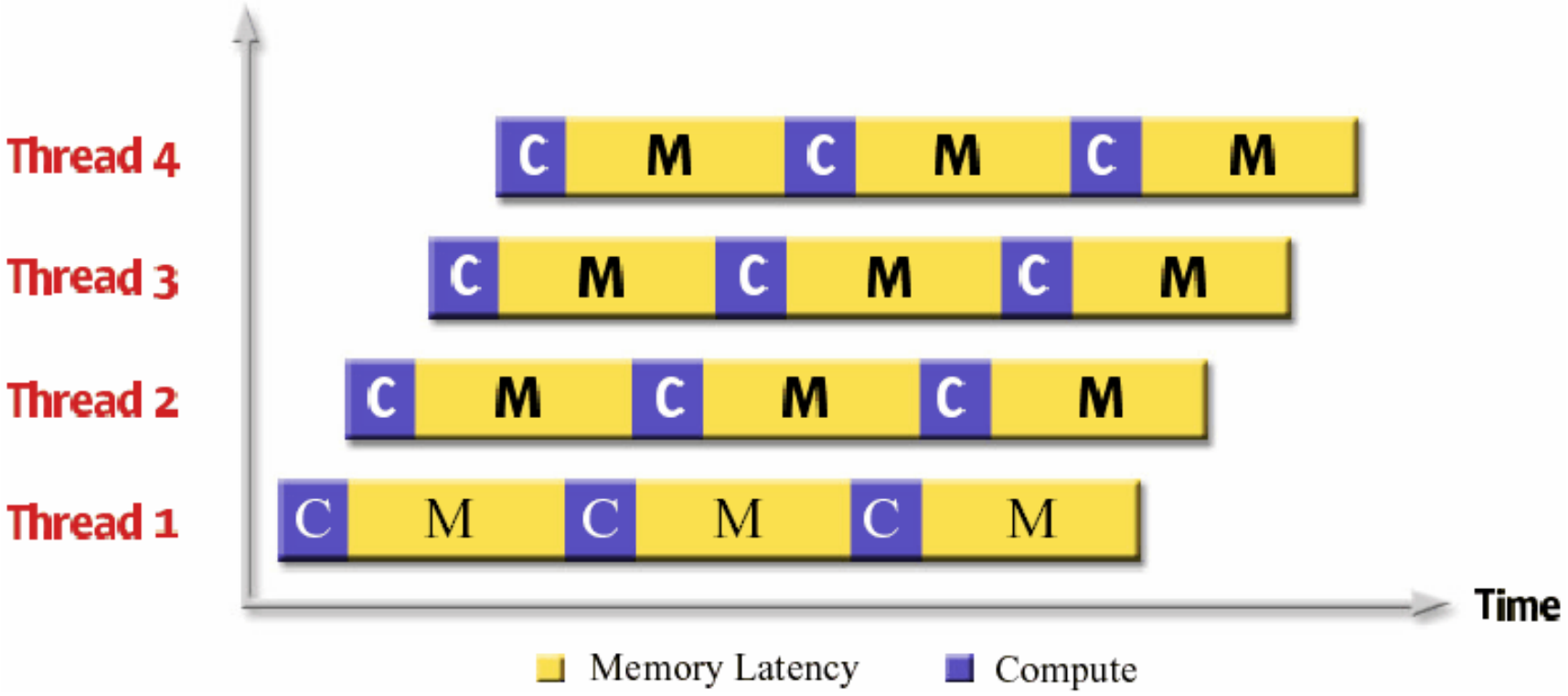
- Since 2005/2006 Intel and AMD are producing dual-core processors for the mass market!
- In 2006/2007 Intel and AMD introduced quad-core processors.
- → Any recently bought PC or laptop is a multi-core system already!



- Memory is significantly slower than CPU

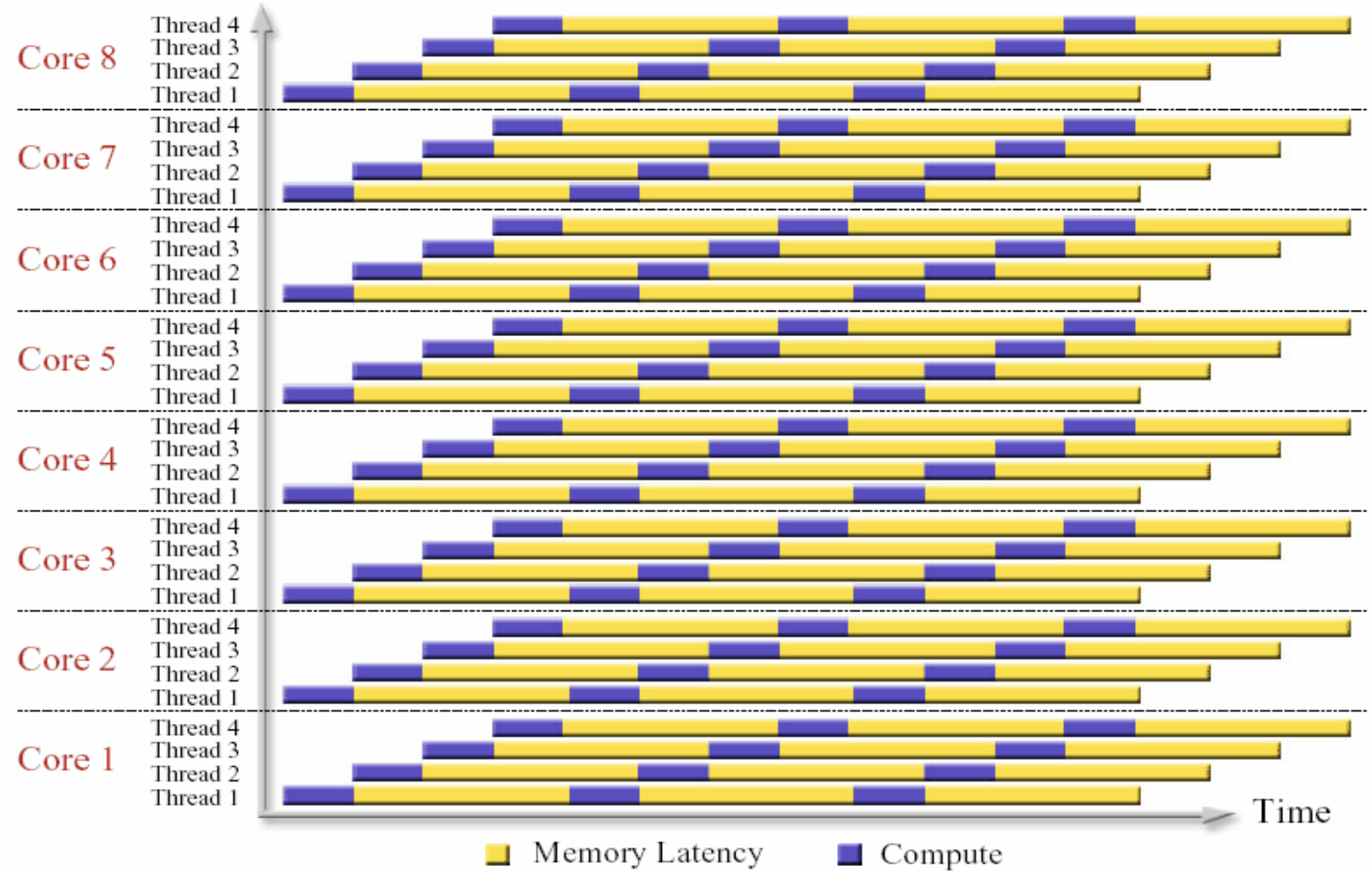


- **Each Core executes multiple threads simultaneously**
  - Typically there is one register set thread per thread
  - But compute units are shared



Today: Multiple Multi-Threaded Cores

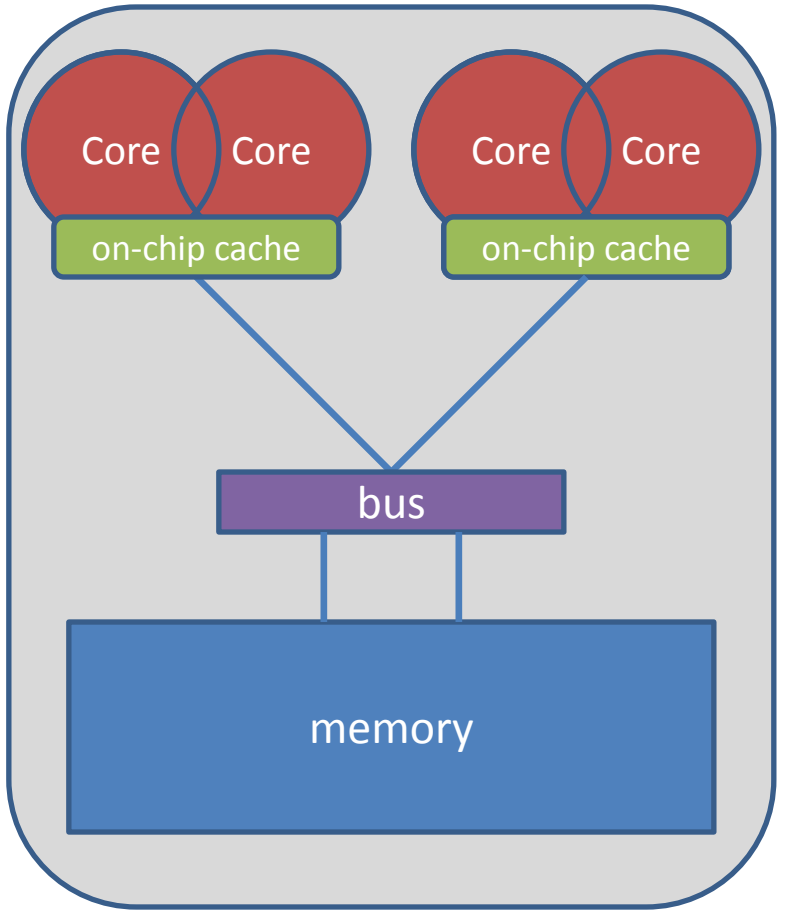
- **Combination of CMP and SMT at work:**



Shared-Memory Parallelism

Example for a SMP system

- **Dual-socket Intel Woodcrest (dual-core) system**
  - Two cores per chip, 3.0 GHz
  - Each chip has 4 MB of L2 cache on-chip, shared by both cores
  - No off-chip cache
  - Bus: Frontsidebus
- **SMP: Symmetric Multi Processor**
  - Memory access time is uniform on all cores
  - Limited scalabilty



Cache Coherence (cc)

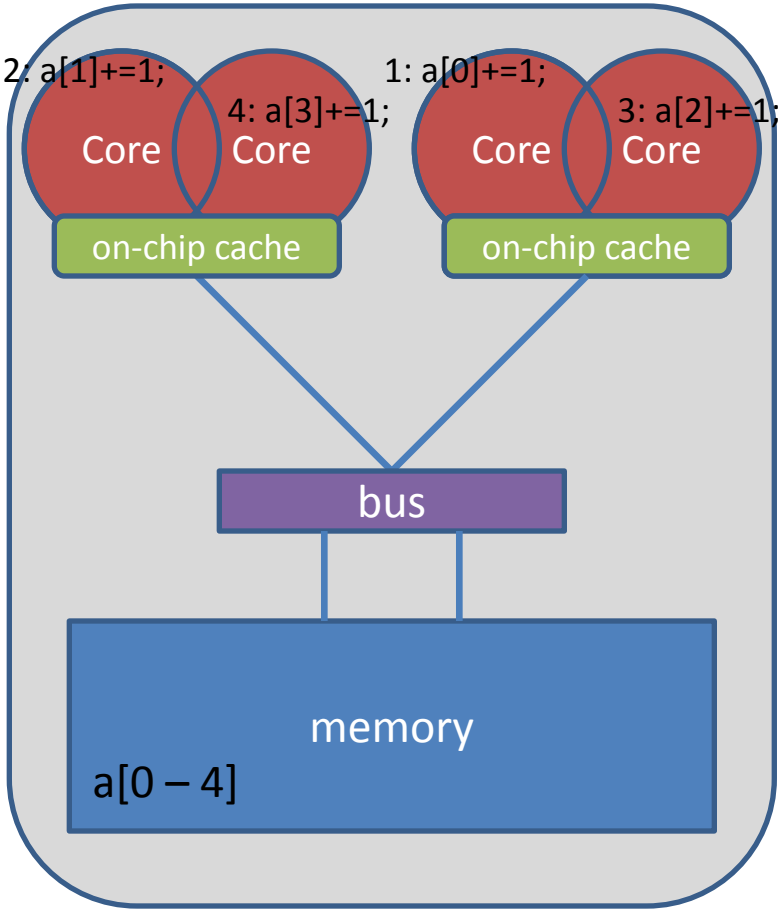
- **If there are multiple caches not shared by all cores in the system, the system takes care of the cache coherence.**
- **Example:**

```
int a[some_number]; //shared by all threads
thread 1: a[0] = 23;      thread 2: a[1] = 42;
--- thread + memory synchronization (barrier) ---
thread 1: x = a[1];       thread 2: y = a[0];
```

  - Elements of array `a` are stored in continuous memory range
  - Data is loaded into cache in 64 byte blocks (cache line)
  - Both `a[0]` and `a[1]` are stored in caches of thread 1 and 2
  - After synchronization point all threads need to have the same view of (shared) main memory

- **False Sharing: Parallel accesses to the same cache line may have a significant performance impact!**

- **False Sharing: Parallel accesses to the same cache line may have a significant performance impact!**



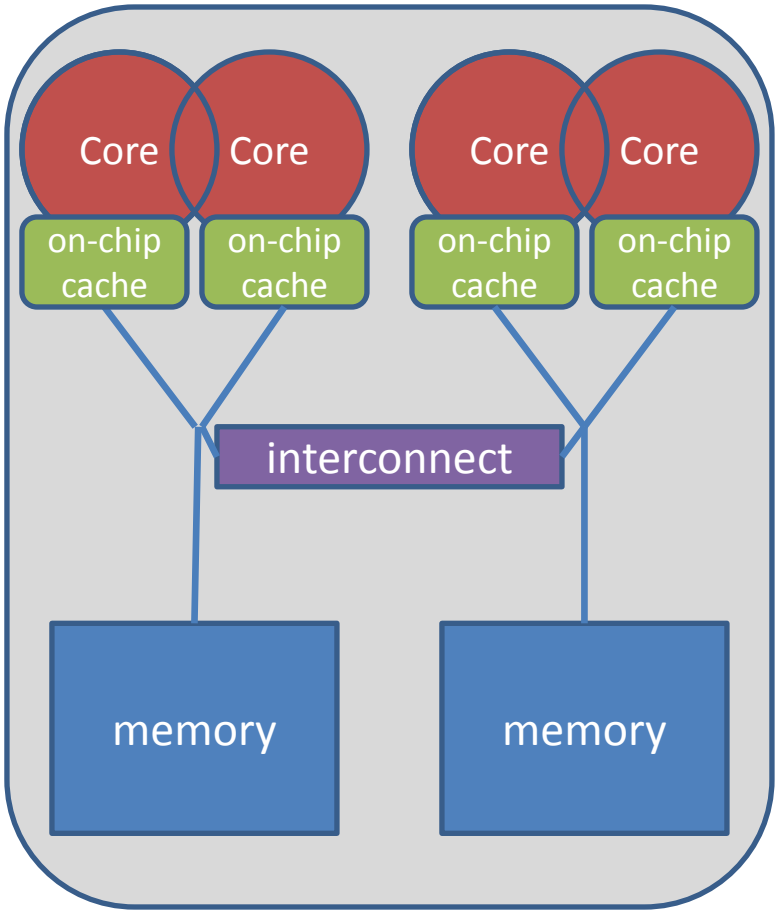
Caches are organized in lines of typically 64 bytes: integer array a[0-4] fits into one cache line.

Whenever one element of a cache line is updated, the whole cache line is Invalidated.

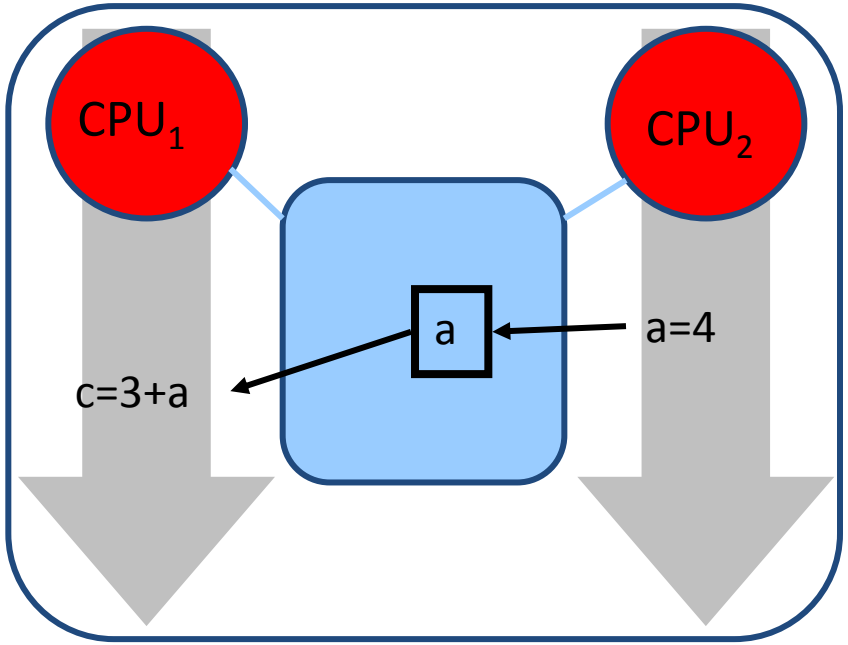
Local copies of a cache line have to be re-loaded from the main memory and the computation may have to be repeated.

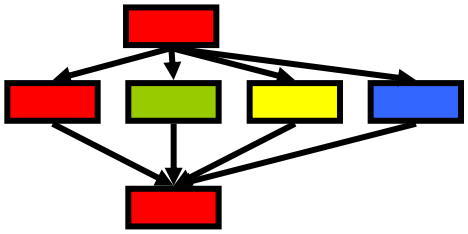
Example for a cc-NUMA system

- **Dual-socket AMD Opteron (dual-core) system**
  - Two cores per chip, 2.4 GHz
  - Each core has separate 1 MB of L2 cache on-chip
  - No off-chip cache
  - Interconnect: HyperTransport
- **cc-NUMA:**
  - Memory access time is non-uniform
  - Scalable (only if you do it right, as we will see)



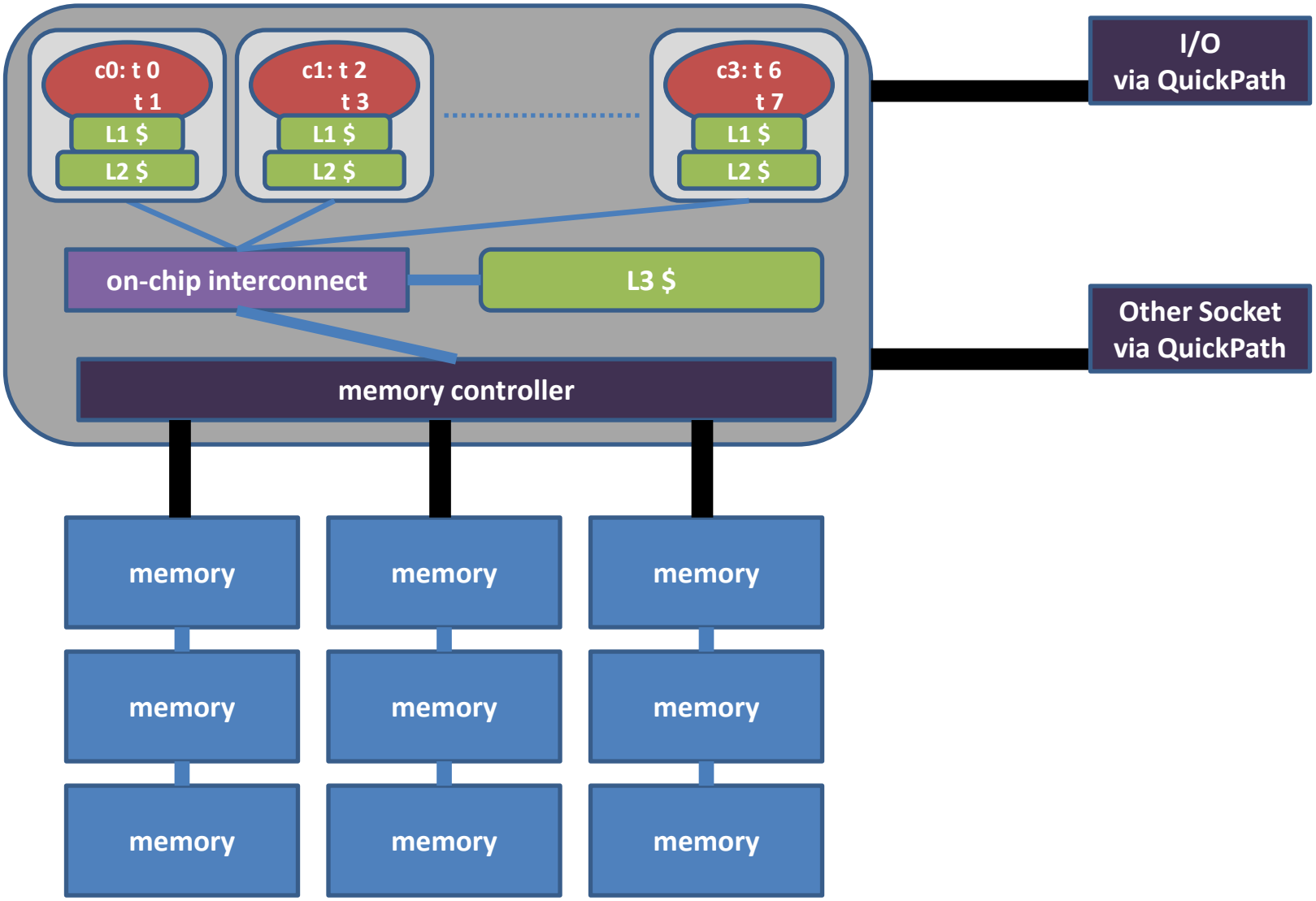
- **Memory can be accessed by several threads running on different cores in a multi-socket multi-core system:**





Look for tasks that can be executed simultaneously (task parallelism)

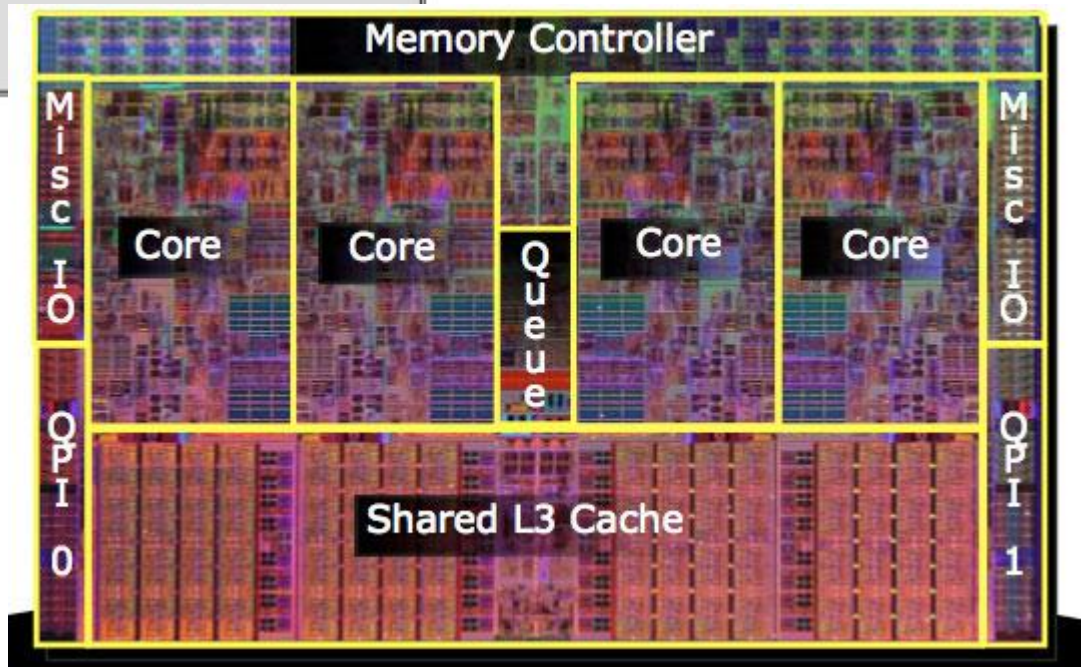
Intel‘s Nehalem (1/4)



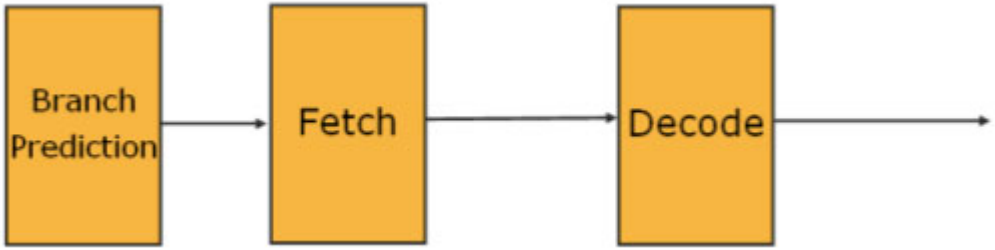
Intel‘s Nehalem (2/4)

- With each improvement in production technology the number of cores per chip increases.

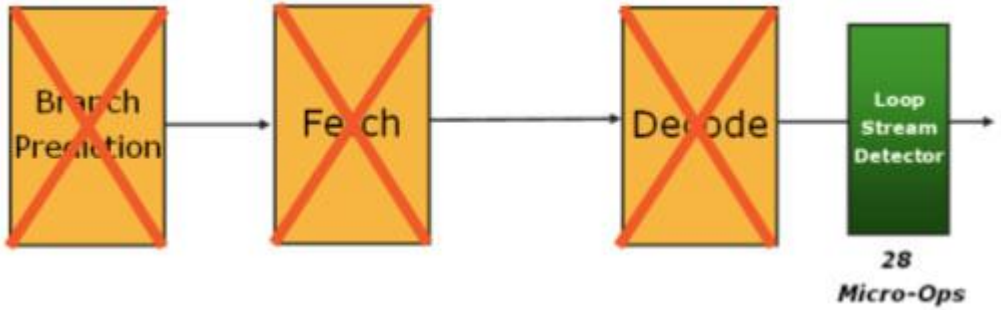
- Minor modification to cache hierarchy: L3 cache is shared by all cores, L1 and L2 caches are per core.



- **Memory efficiency is of high importance:**
  - Traditional prediction pipeline:



  - Loop Stream Detector (LSD) powers down branch prediction, fetch and decode hardware as soon as a loop is detected:



  - Up to 28 uOps can reside inside the LSD.
- **A core can be disabled completely as well!**

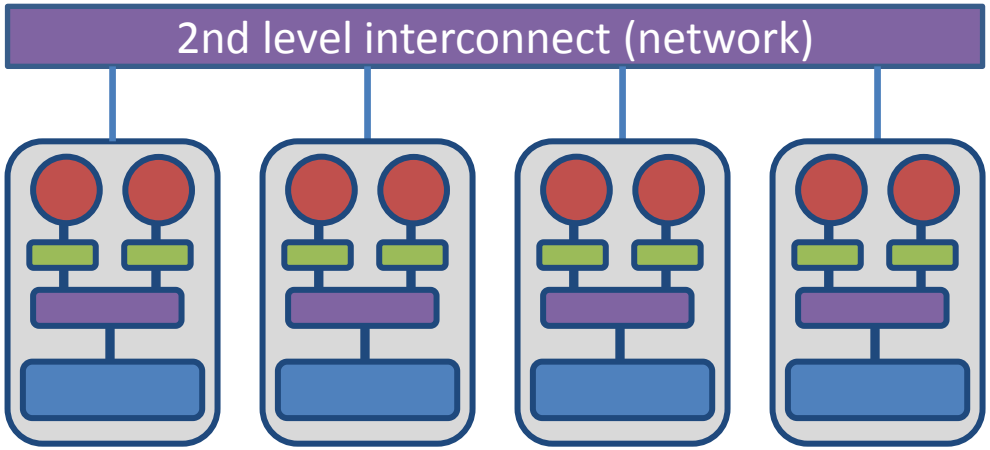
Intel‘s Nehalem (4/4)

- **Technology**
  - 45 nm manufacturing process
  - Integrated memory controller
    - Intel QuickPath Interconnect replaces FrontsideBus
    - Will offer cc-NUMA characteristics (see below)
  - Simultaneous Multi-Threading (SMT) = Hyper-Threading
  - Cache Hierarchy
    - 32 KB L1 instruction cache + 32 KB L1 data cache per core
    - 256 KB L2 cache per core
    - 2 or 3 MB L3 cache per core, but shared by all cores
  - Number of pipeline stages: Core microarchitecture has only 12 stages (compared to 30 in latest Netburst architecture)

Distributed-Memory Parallelelism

Example for a Cluster of SMP nodes

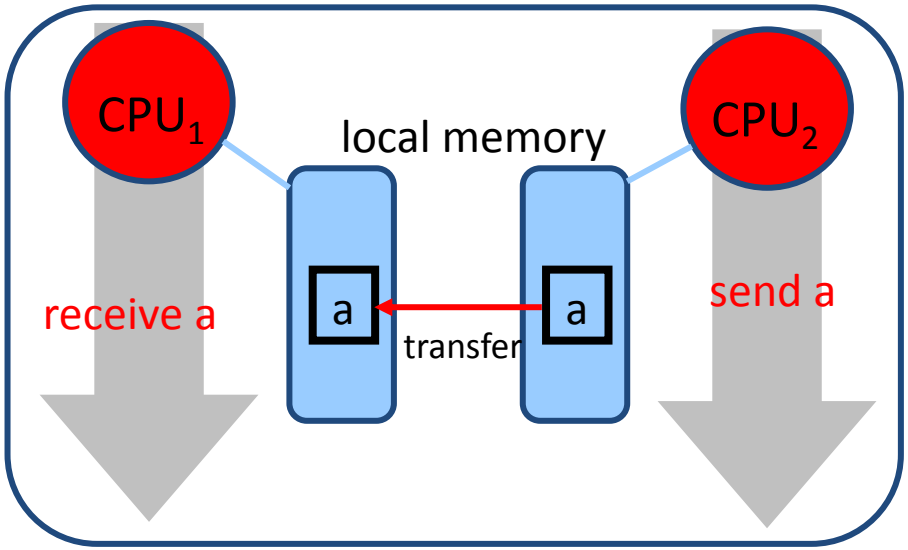
- **Second level interconnect (network) is not cache coherent**
  - Typically used in High Performane Computing: InfiniBand
    - Latency: <= 5 us
    - Bandwidth: >= 1200 MB/s
  - Also used: GigaBit Ethernet:
    - Latency: <= 60 us
    - Bandwidth: >= 100 MB/s

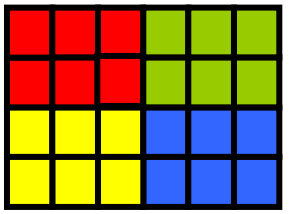


Latency: Time required to send a message of size zero (that is: time to setup the communication)

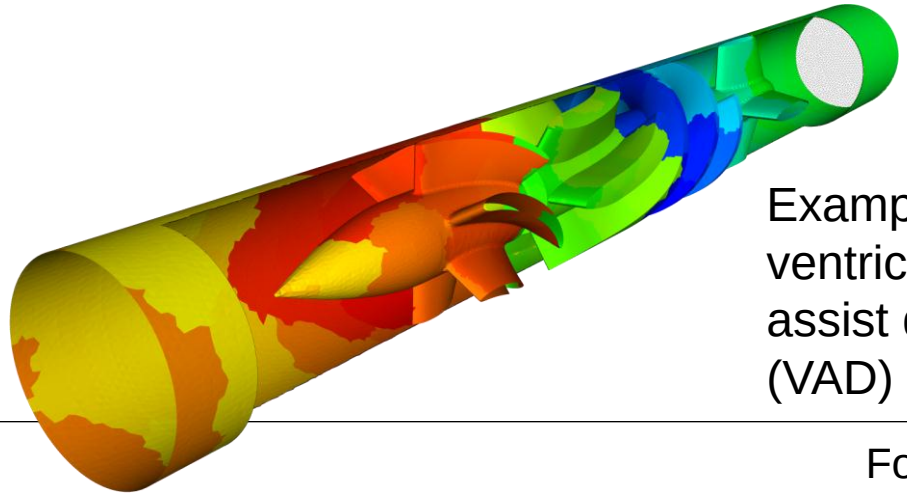
Bandwidth: Rate at which large messages (>= 2 MB) are transferred

- Each process has it‘s own distinct memory
- Communication via Message Passing



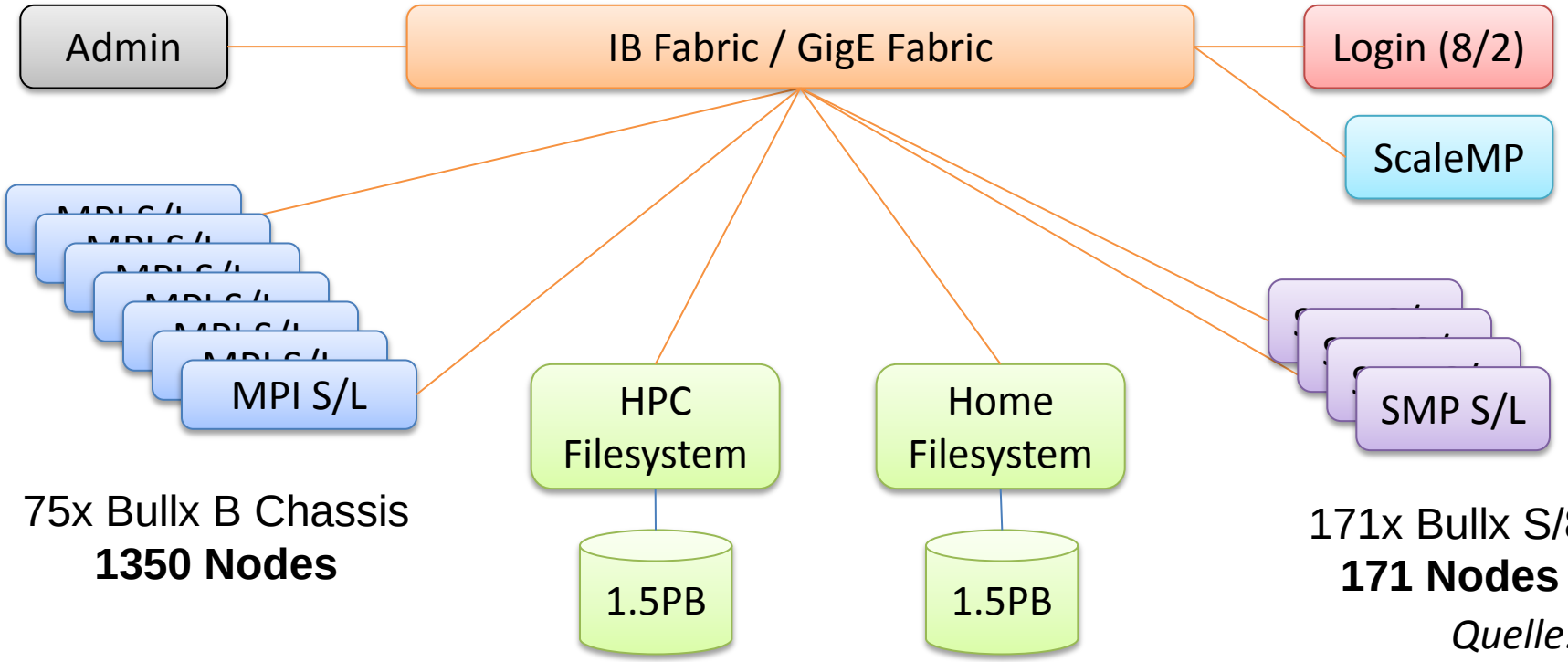


Decompose data into distinct chunks to be processed independently (data parallelism)



Example: ventricular assist device (VAD)

New HPC Cluster from Bull: Overview

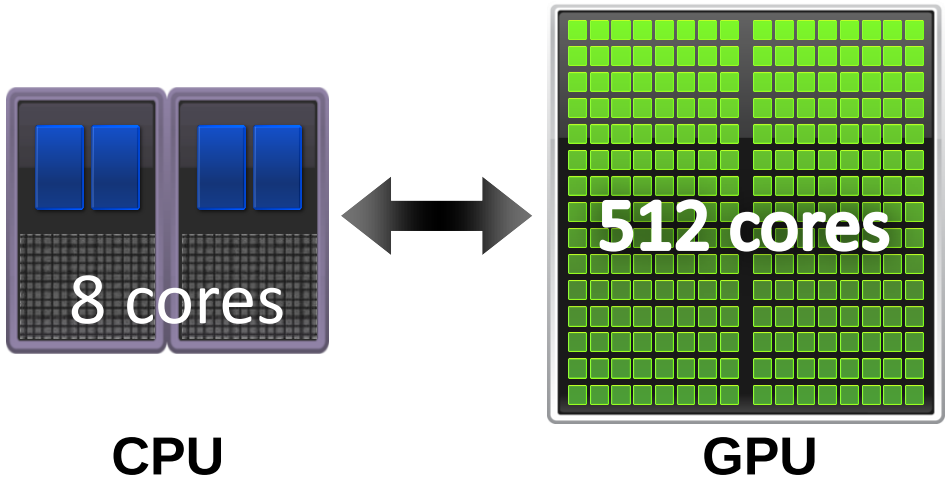


Quelle: Bull

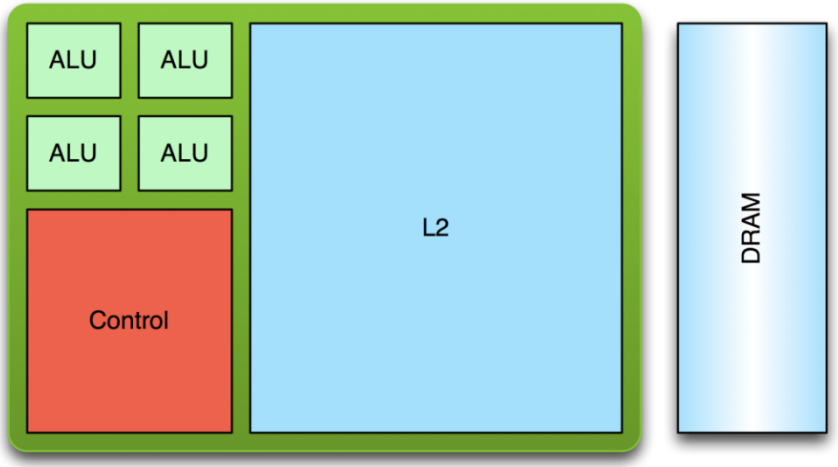
Group	# Nodes	Sum TFLOP	Sum Memory	# Procs per Node	Memory per Node
MPI-Small	1098	161	26 TB	2 x Westmere-EP (3,06 GHz, 6 Cores, 12 Threads)	24 GB
MPI-Large	252	37	24 TB		96 GB
SMP-Small	135	69	17 TB	8 x Nehalem-EX (2,0 GHz, 8 Cores, 16 Threads)	128 GB
SMP-Large	36	18	18 TB		512 GB
ScaleMP-vSMP	8 gekoppelt	4	4 TB		4 TB gekoppelt

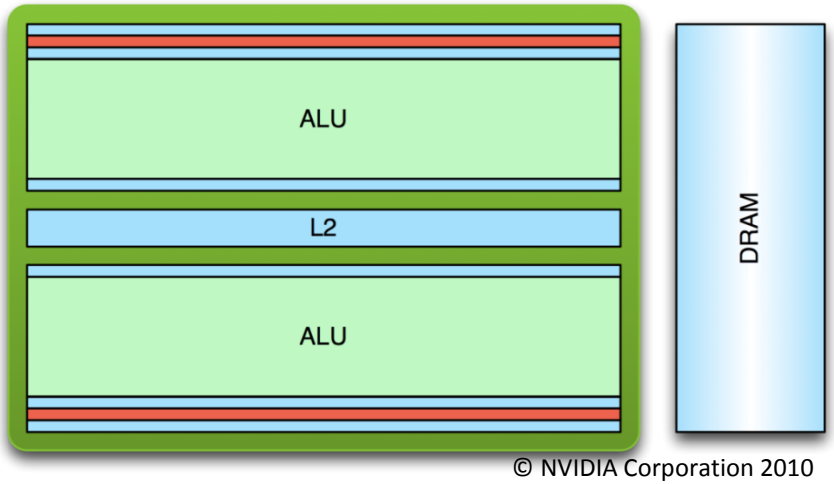
General Purpose Graphic Processing Units (GPGPUs)

- **GPGPUs = General Purpose Graphics Processing Units**
  - From fixed-function graphics pipeline to programmable processors for general purpose computations
- **Programming paradigms**
  - CUDA C/Fortran, OpenCL C, PGI Accelerator for C/Fortran,…
- **Main vendors**
  - NVIDIA, e.g. Quadro, Tesla, Fermi
  - AMD, e.g. FireStream, Radeon
- **„Massively parallel processors“**
- **„Manycore architecture“**



- **Similar # of transistors but different design**





CPU

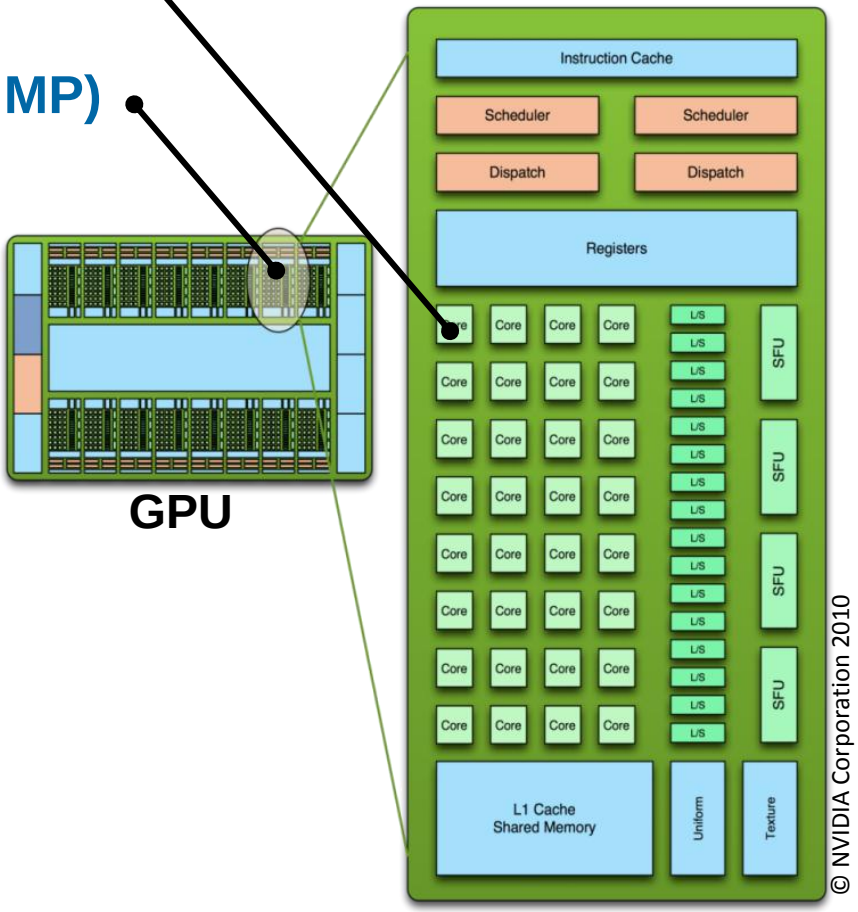
- Optimized for low-latency access to cached data sets
- Control logic for out-of-order and speculative execution

GPU

- Optimized for data-parallel, throughput computation
- Architecture tolerant of memory latency
- More transistors dedicated to computation

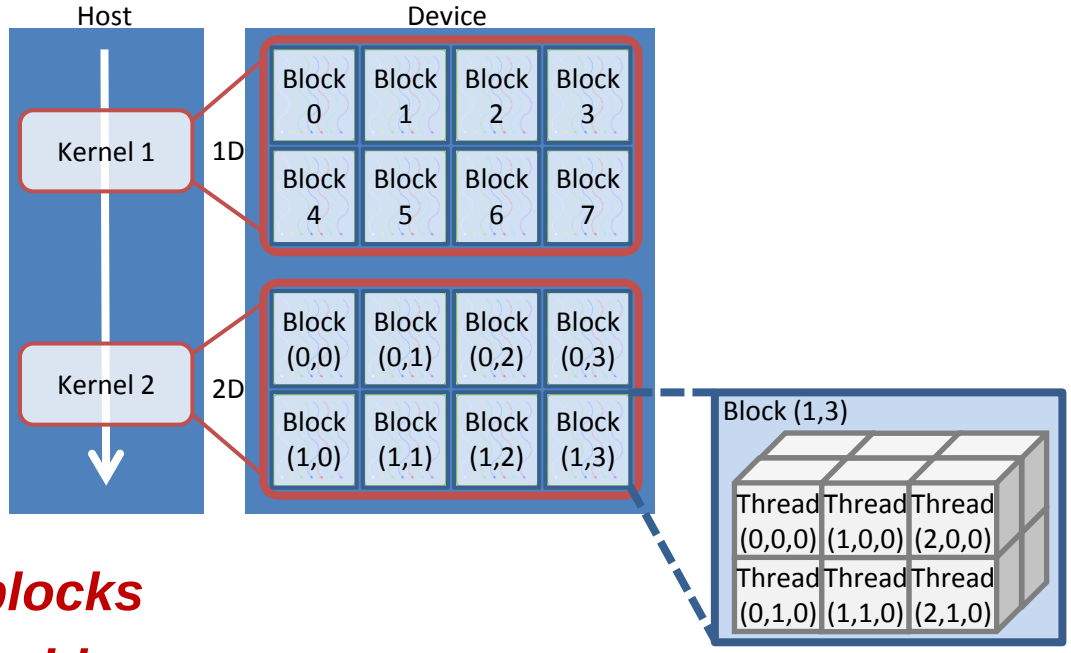
GPGPU architecture: NVIDIA‘s Fermi

- **3 billion transistors**
- **448 *Cores/ Streaming Processors* (SP)**
  - E.g. floating point and integer unit
- **14 *Streaming Multiprocessors* (SM, MP)**
  - 32 cores per MP
- **Memory hierarchy**
- **Processing flow**
  - Copy data from host to device
  - Execute kernel
  - Copy data from device to host



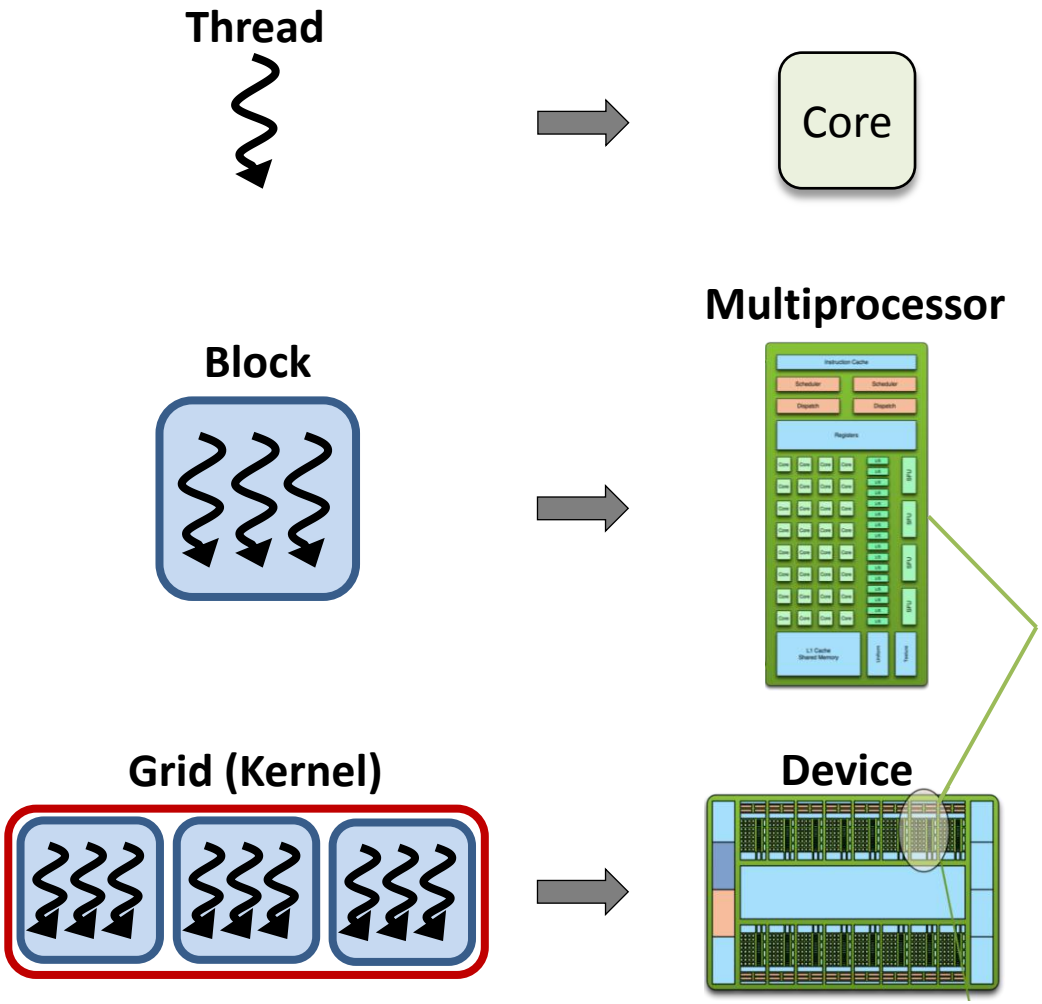
- **Parallel portion of application is executed as kernel on the device**
  - Kernel is executed as an array of threads
  - All threads execute the same code
- **GPU-Threads**
  - 1000s simultaneously
  - Lightweight,

    little creation overhead
  - Fast switching



- **Threads are grouped into *blocks***
- **Blocks are grouped into a *grid***

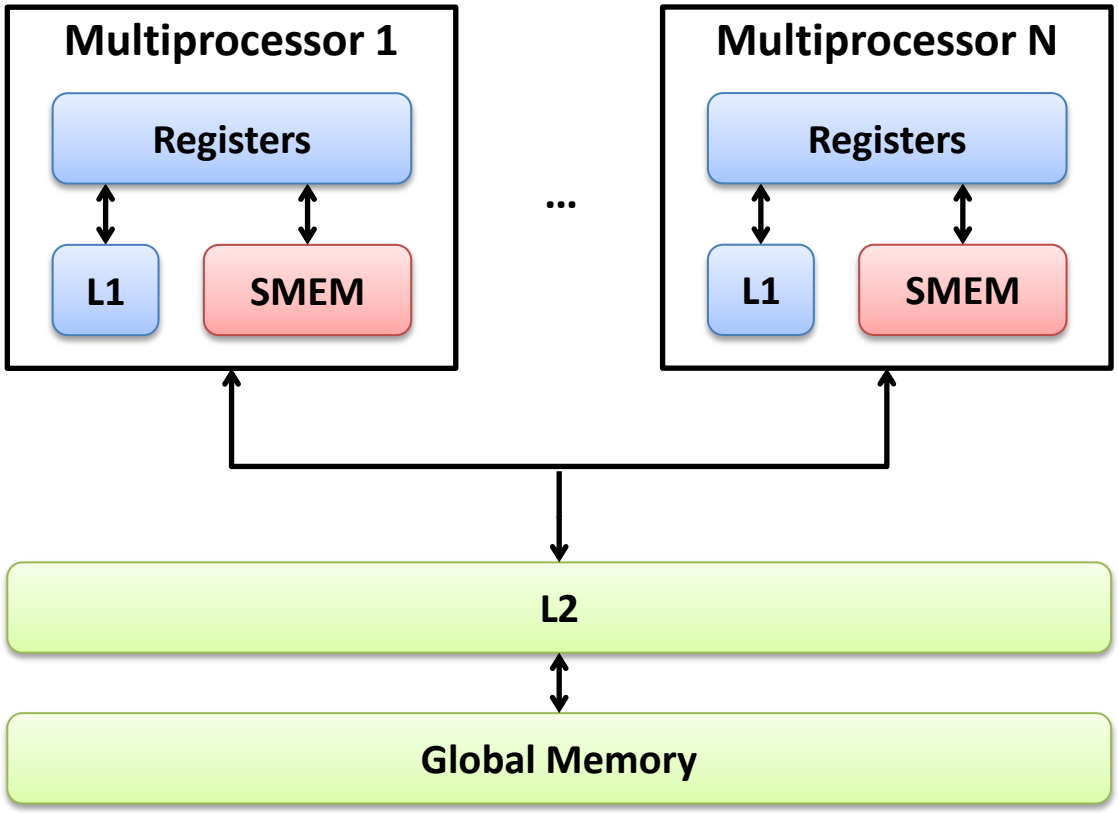
GPGPUs – Execution model



- Each thread is executed by a core
- Each block is executed on a multiprocessor
- Several concurrent blocks can reside on one MP, depending on memory requirements/ resources
- Each kernel is executed on the device

GPGPUs – Memory hierarchy (Fermi)

- **Thread**
  - *Registers*
- **Block**
  - *Shared* Mem
    Fermi: up to 48 KB; on-chip
- **Grid/ application**
  - *Global* Mem
    up to 6 GB; off-chip
- **Optimizing memory requests and access pattern is essential!**

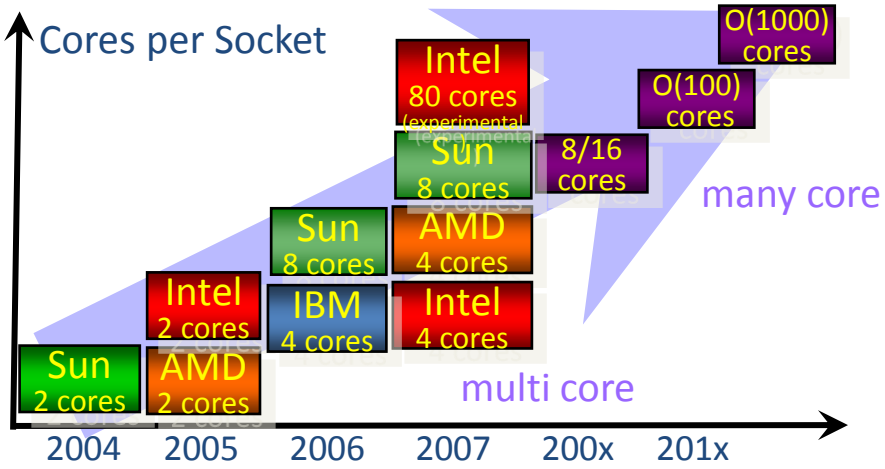


- **If massive data-parallelism → Lots of Flops achievable!**

Summary and Conclusions

Summary and Conclusion

- **With a growing number of cores per system, even the desktop or notebook has become a parallel computer.**
  - Only parallel programs will profit from new processors!
- **SMP boxes will be building blocks of large parallel systems.**
- **Memory hierarchies will grow further.**
- **CMP + SMT architecture is very promising for large, memory bound problems.**



- **Main problems: Power supply and Cooling:**
  - Woodcrest: 24 GFLOPS at ~100 W → 1 PFLOPS at ~4 MW!
- **BUT: Programming for 100,000 of cores is very hard!**

The End

Thank you for your attention.