COMP 273
Introduction to Computer Systems

COMP 273

The Multi-Core CPU

Micro Architecture

Part 5

Prof. Joseph Vybihal

Announcements

- Multi-core not on midterm exam
- Assignment out after midterm
- Project out after midterm
  - Start thinking about teams of 3 or 4
  - Working alone is also possible... but not recommended
  - A team of 3 is recommended

Midterm Exam

- Monday, Oct 27, 2014
- 6pm – 8pm
- STBIO N2/2 and STBIO S3/3

- No class that day. Instead I will be holding office hours in MC323.

- Lecture slides 1 to 13 on the exam.
- Closed book exam

Midterm Exam

- Section 1: Multiple choice (10 to 15)
  - Theory, definitions, terms, math, what is this, syntax, data paths, combinatorial logic
- Section 2: Problems (3 to 4)
  - Memory/combinatorial logic, math, circuit interpretation, complete this circuit, machine instructions, floating point, data formats, micro instruction notation
- Section 3: Circuit design (1)

Conflict Midterm

- Friday, Oct 31st 2014
- MC103 from 10:00 to 12:00
- Only those with a valid conflict can attend.

- Other students who cannot attend the midterm exam and do not qualify for the Conflict Midterm can still skip the midterm exam and make their final exam worth more.

Readings

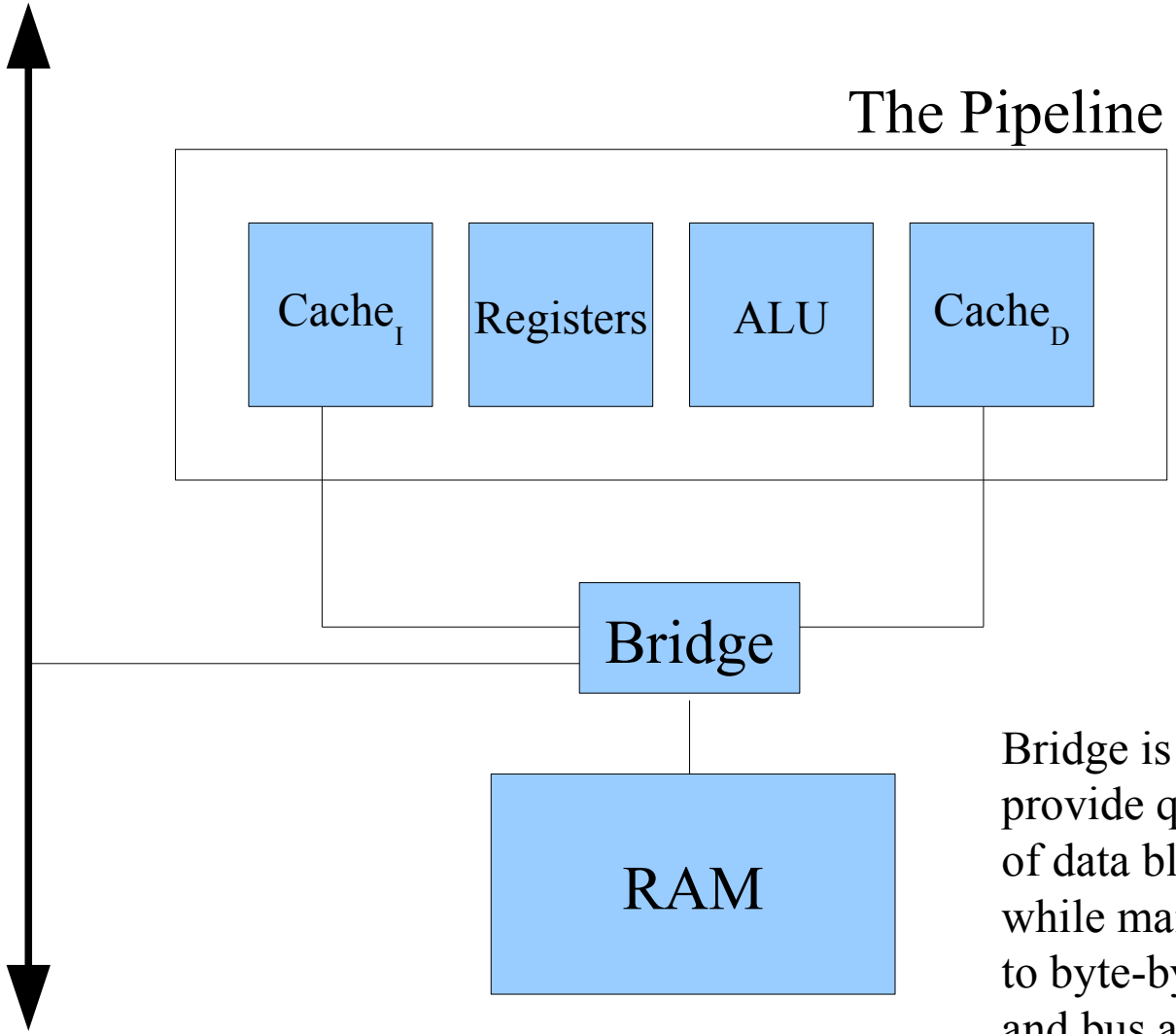
- Appendix E: A survey of RISC architectures
- Chapter 7:Multicores, multiprocessors, & clusters

About Multiprocessing computers

Types of Computers

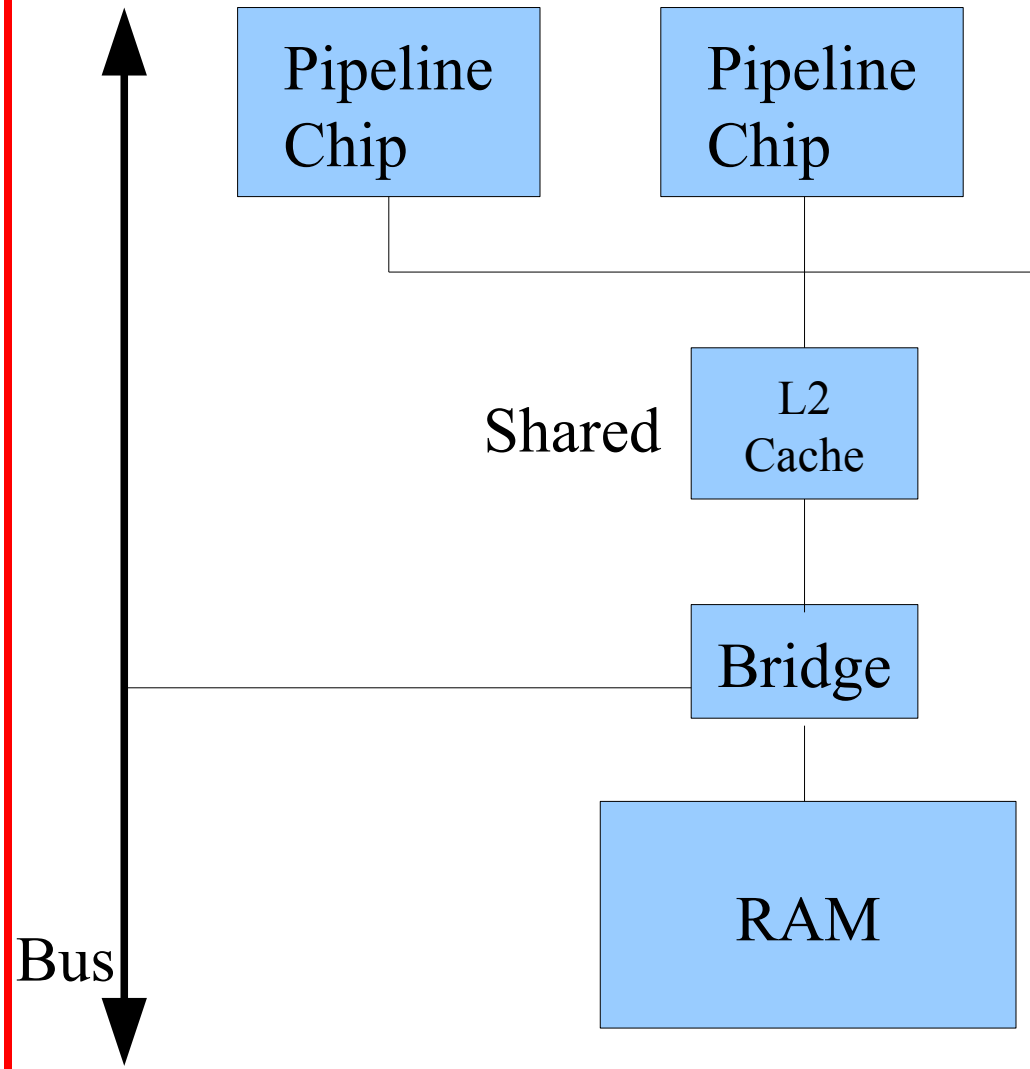
- Single CPU
  - A computer with a single CPU chip of any type (flat, pipeline,hyper threaded)
- Multi-CPU
  - A computer with more than one CPU chip of any type
- Single Core
  - Similar to single CPU chip, but optimized
- Multi-Core
  - A single CPU chip with multiple processors

Single CPU Computer



Bridge is designed to provide quick download of data blocks into cache, while maintaining access to byte-by-byte operations and bus access.

Multi-CPU Computer

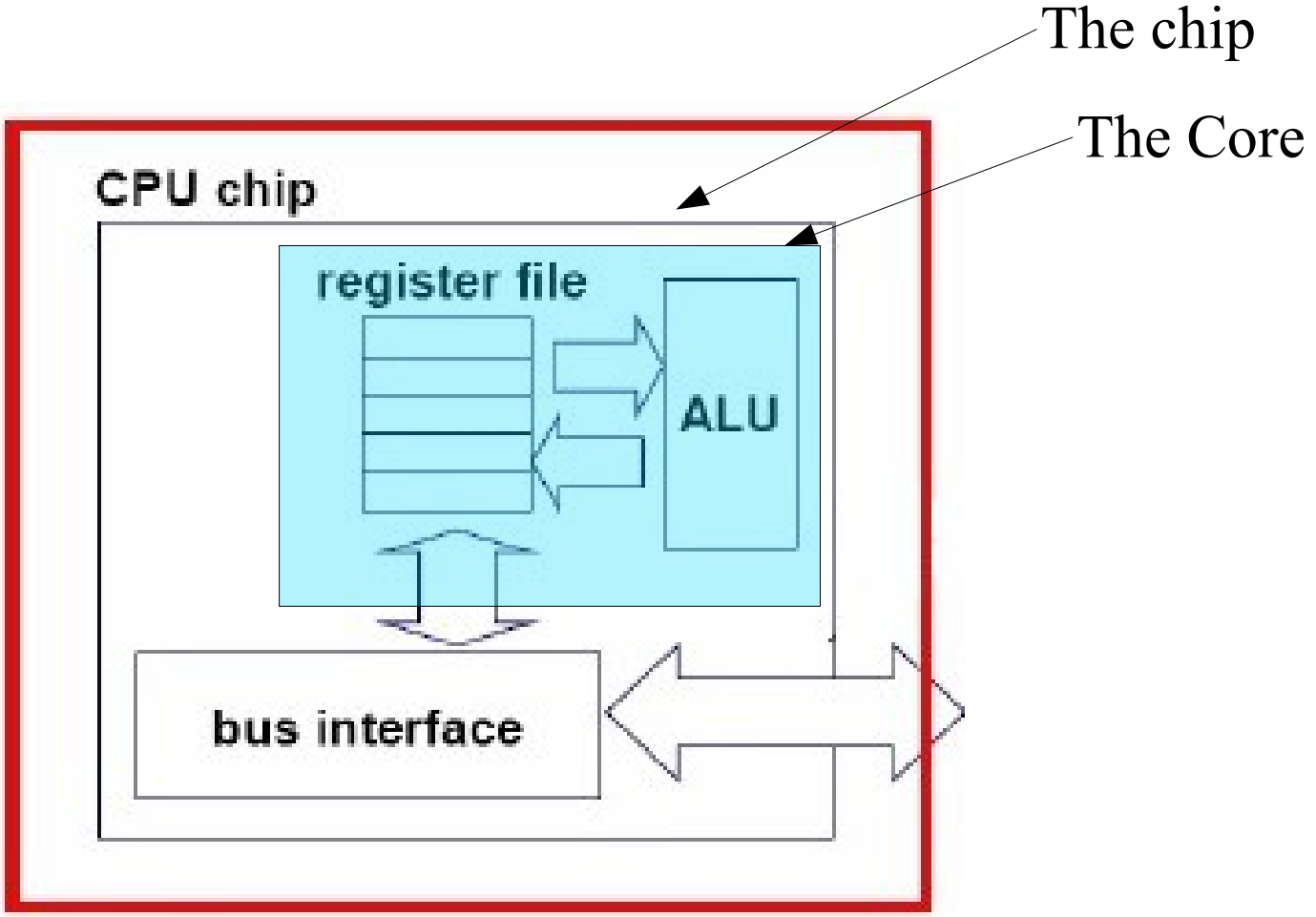


Each chip has it's Instruction and Data caches.

Each pipeline is a full standard CPU with all it's parts. This is obvious since they are individual chips.

A popular variation to multi CPU are PCs on a LAN working on a common problem.

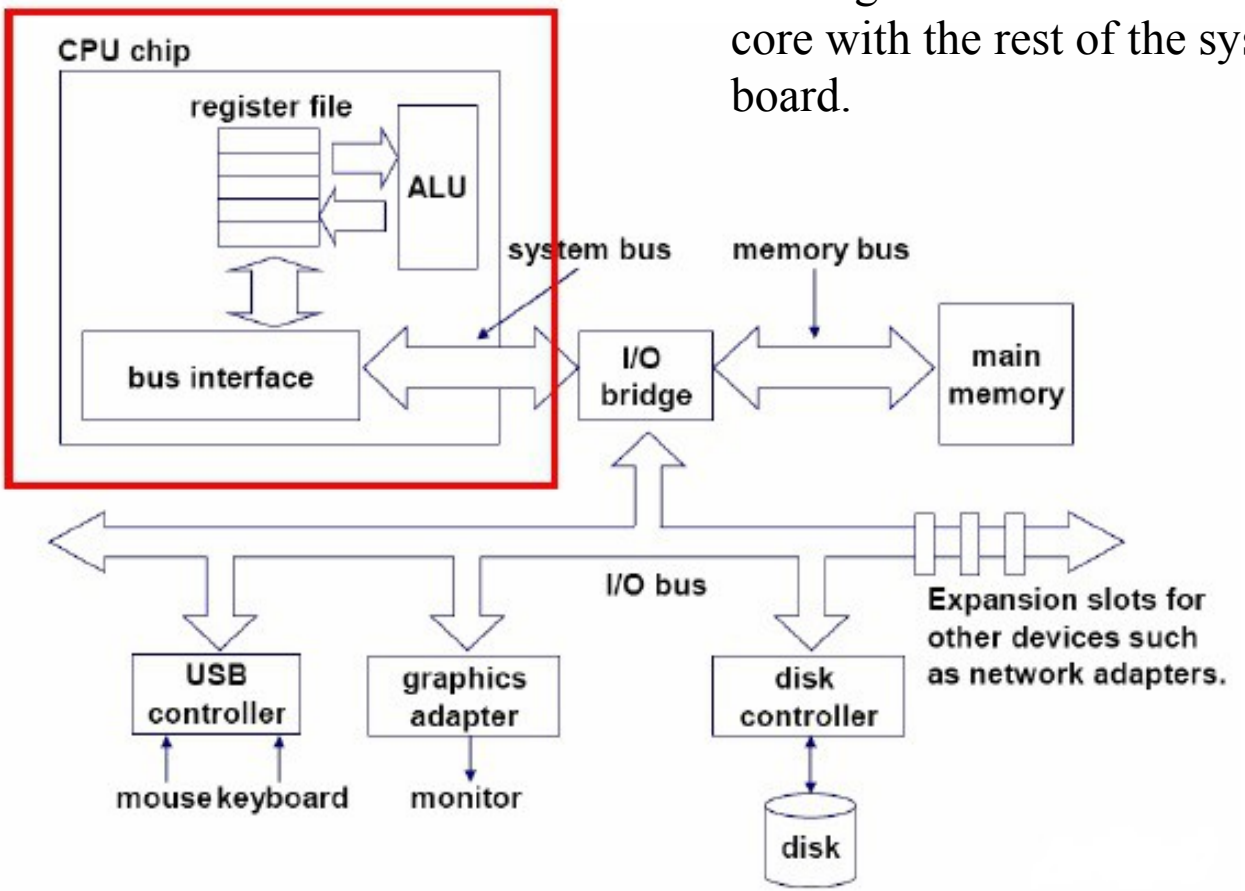
Single-core CPU Chip



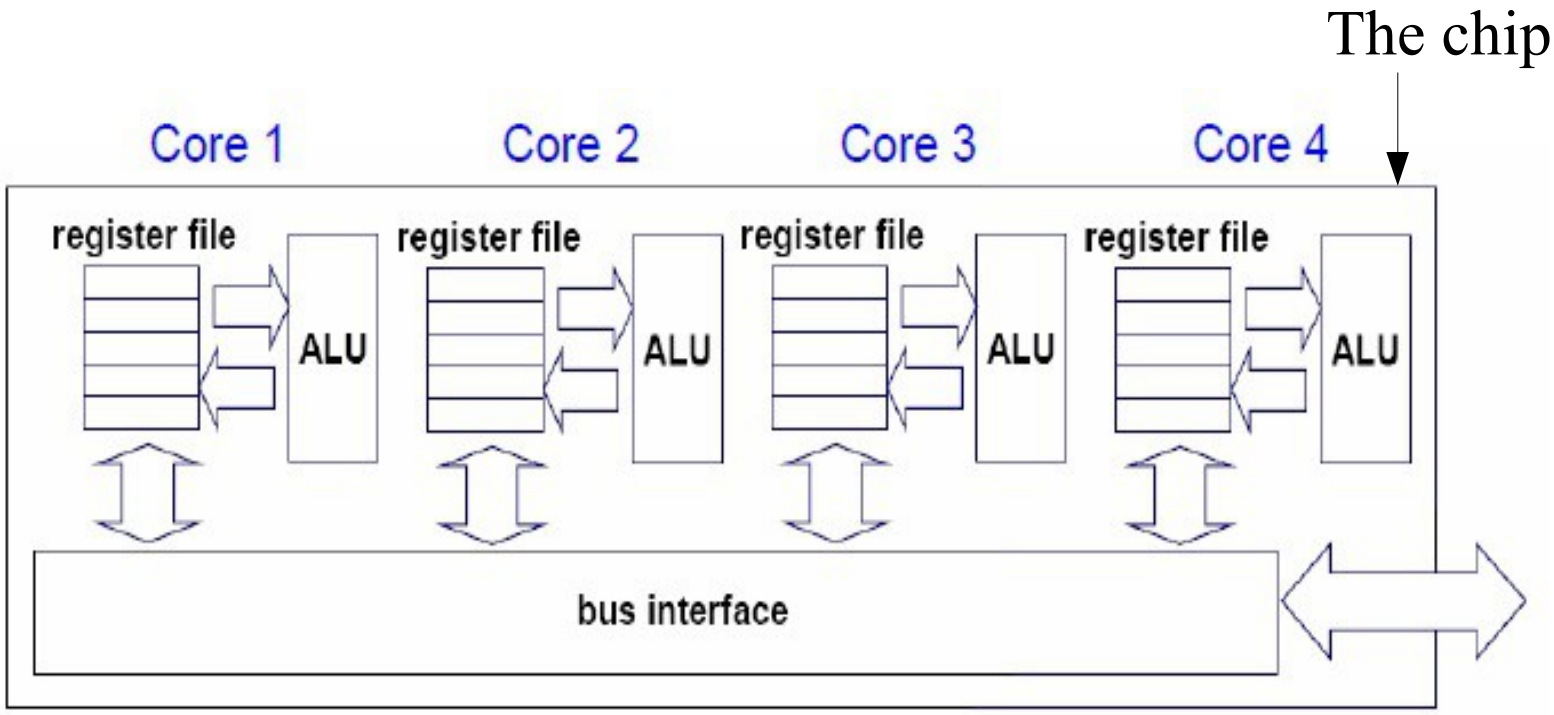
The bus interface often contains the cache, among other things.

Single-core Computer

A bridge exists to interconnect the core with the rest of the system board.



Multi-core Architecture



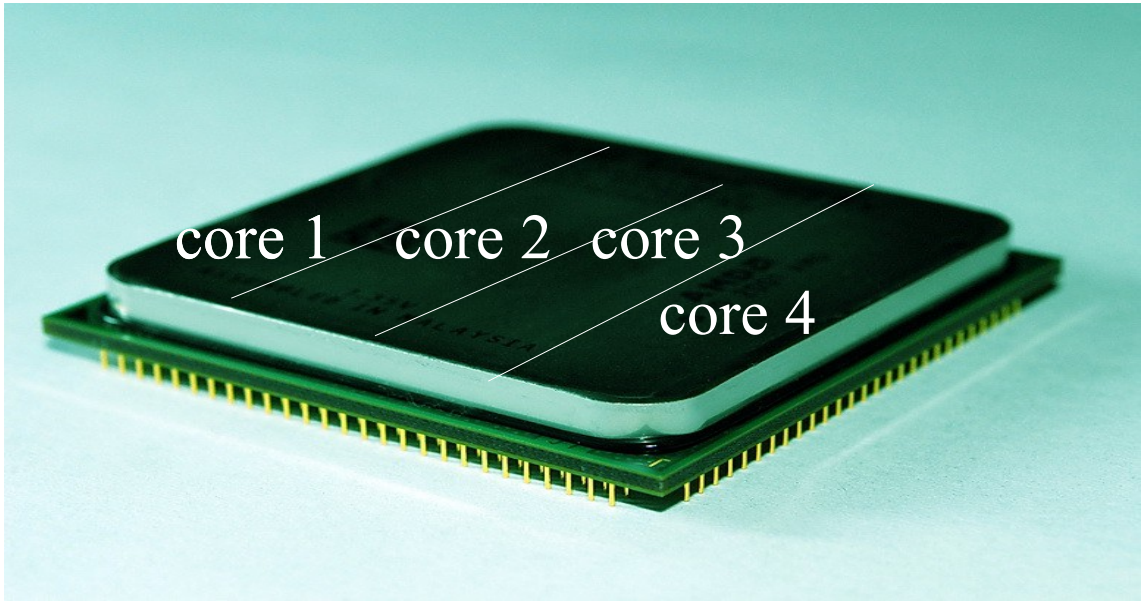
Multi-core CPU chip
Notice, in a Core, some of the pipeline is missing. (shared!)

Notice that the bus interface is shared. This permits the same data for all cores.
* Notice how this is different from multi-CPU.

The replication of multiple processor cores on a single die.

Multi-core CPU Chip

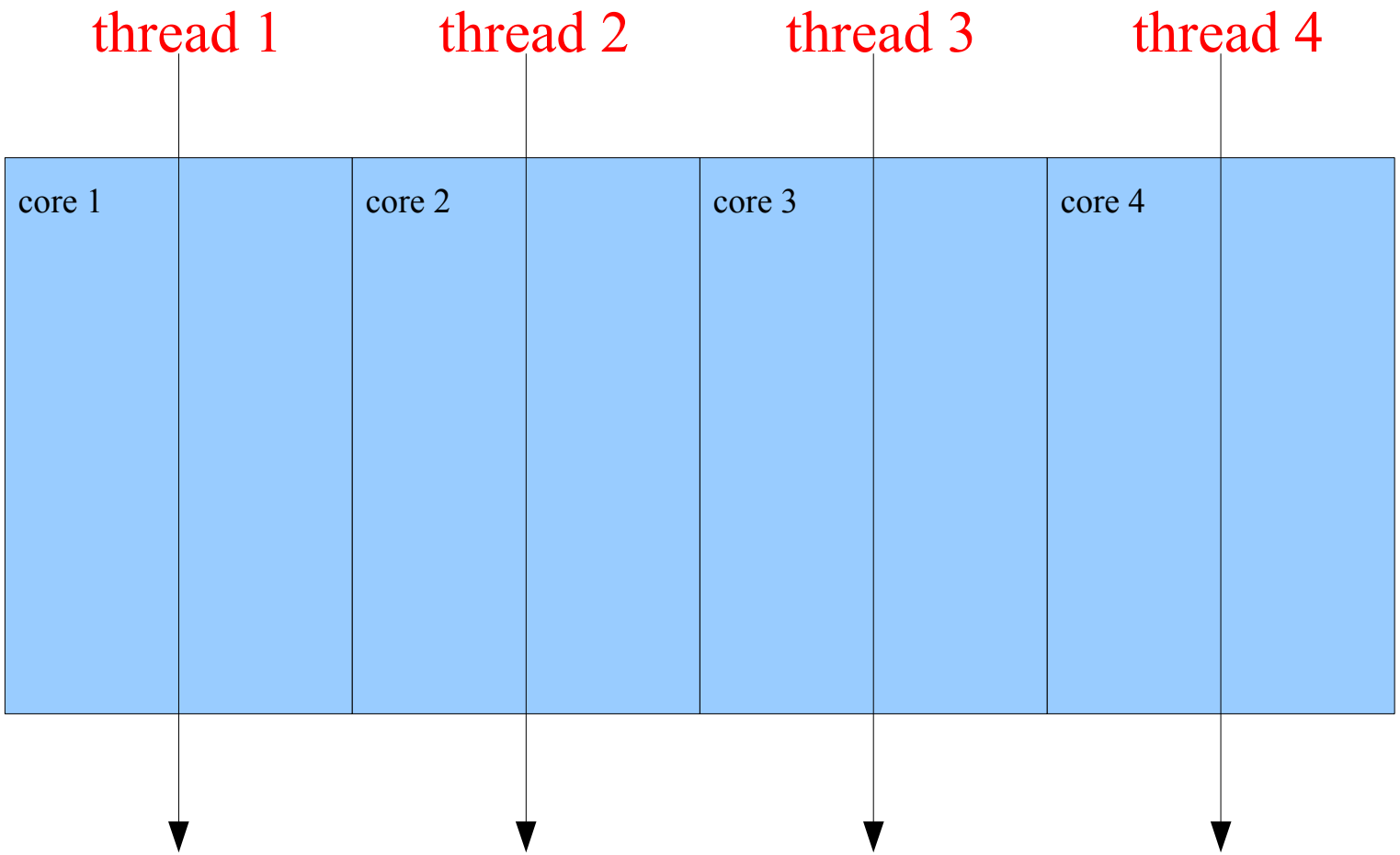
- The cores fit on a single processor socket
- Also called CMP (Chip Multi-Processor)



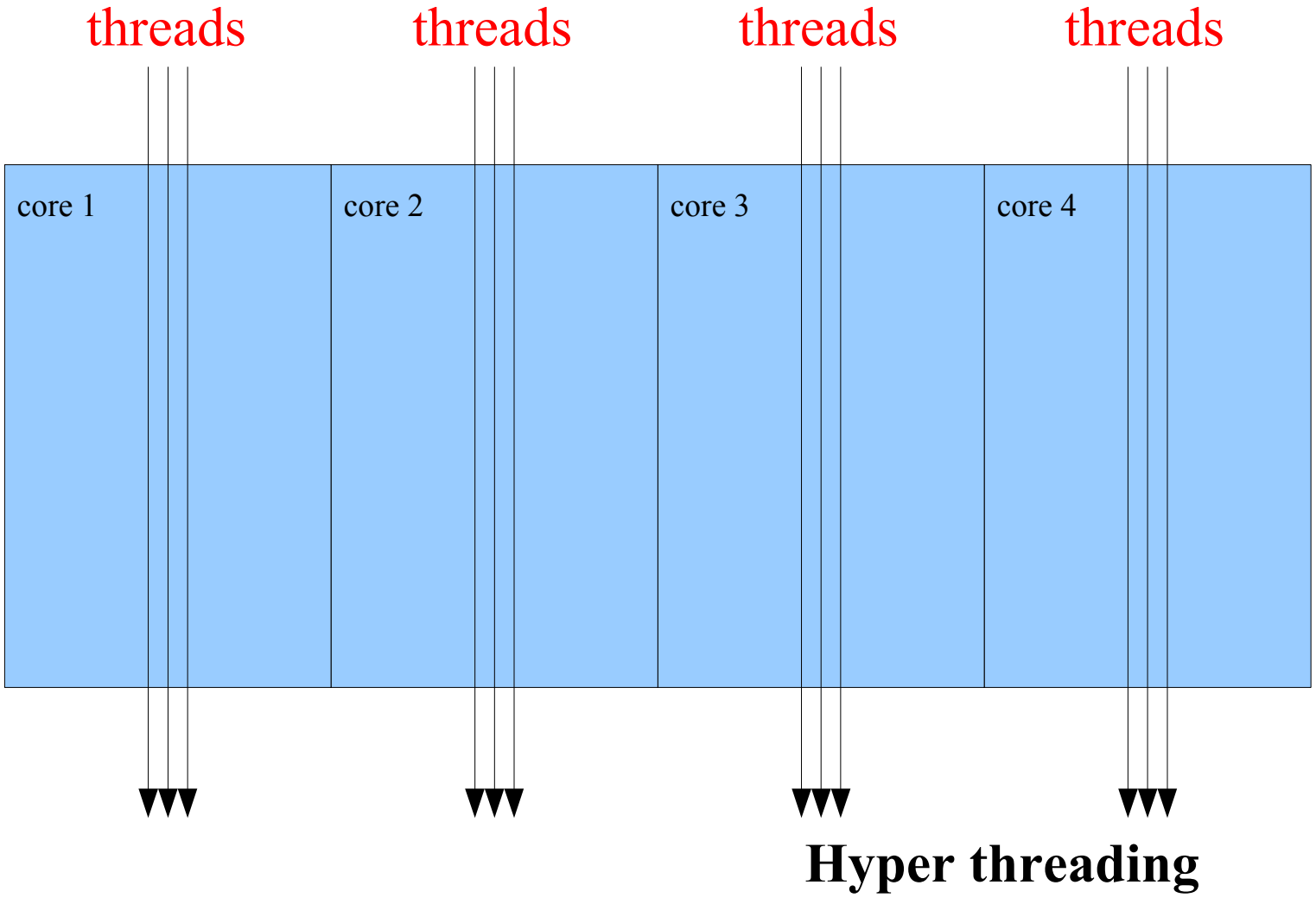
Programs, Processes & Threads

- A program is a compiled algorithm stored on disk.
  - It has OS loader & static components
- A process is the executing version of the program in RAM
  - It has static and dynamic components
- A thread are those instructions currently being executed by the CPU
  - One process may have many threads
  - Each thread is an instance of the process

The cores run in parallel



Time slicing within each core!



Hyper threading

The Operating System

- Each core is treated as a unique processor
- The OS schedules processes and threads to different cores
- All major operating systems support multi-core processing

The Operating System

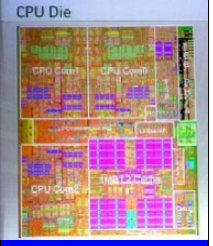
- Example:
  - One process with 2 threads on a quad-core
  - OS has choices:
    - Use 2 cores to run threads simultaneously
    - Use 1 core and task switch between threads
    - High priority code executing on all cores, therefore queue process and threads for later.

The Operating Systems

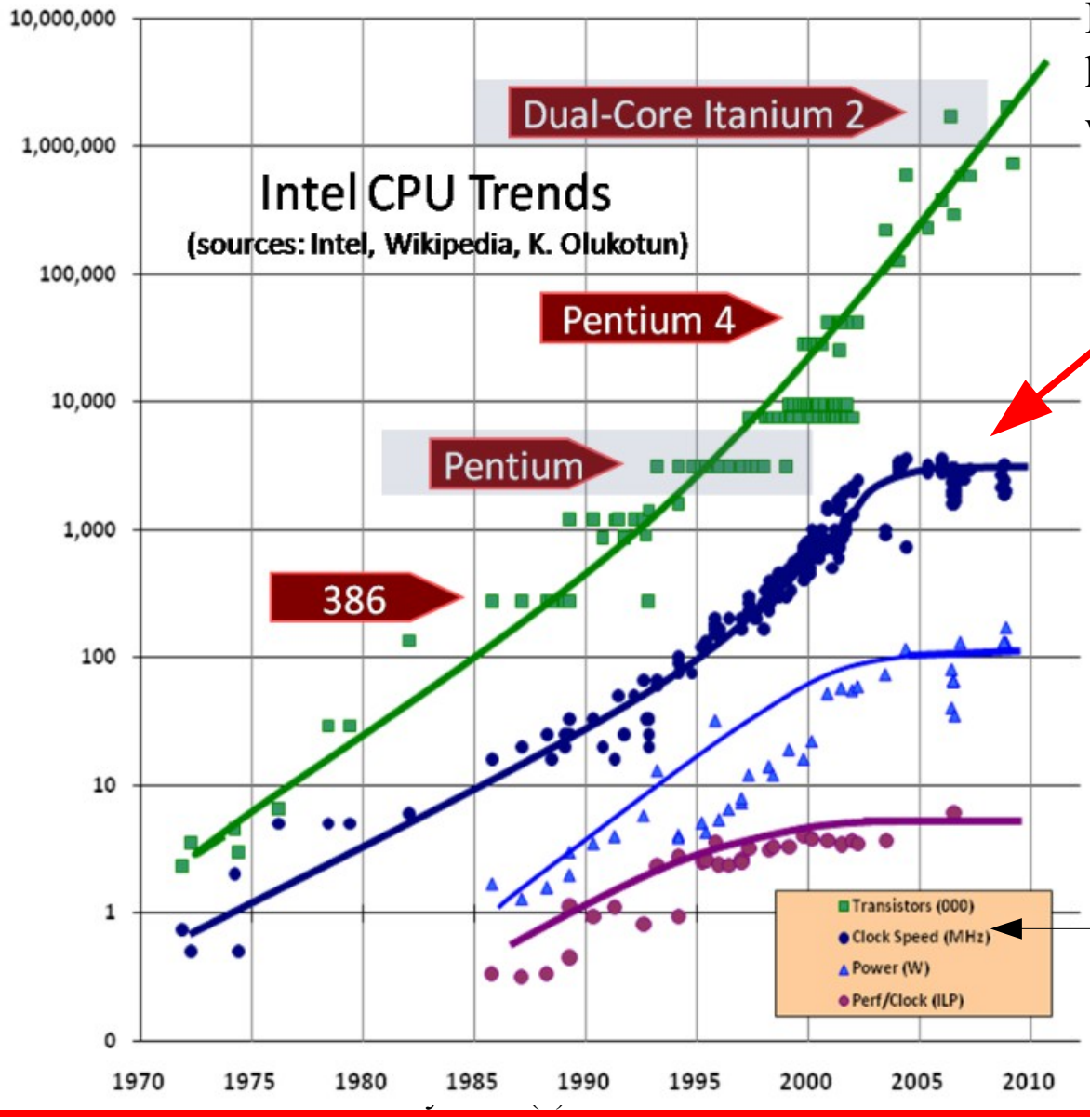
- Core management strategies
  - Special purpose core assignment
    - OS reserves cores for specific category of applications. For example: all OS code executes on core 1. Nothing else executes on core 1.
  - Dedicated application assignment
    - An application is attached to a core and always executes on that core. Browser assigned to core 2. But core is share-able with other processes.
  - Core pooled assignment
    - The cores are not dedicated to anyone. A queue manages all processes. The process at the head of the queue is given any available core.

OS Manages the Cores

- Simple management
  - Queue of processes waiting to execute
  - Each core assigned to a waiting process
  - After n micro-seconds release core, repeat
- Complex management
  - One process has multiple threads in queue waiting to execute
  - These need to share data, need to be placed on a core that can share data
  - Hardware to work out data access conflicts

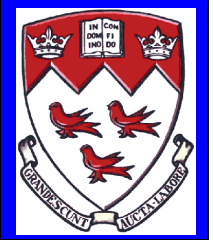


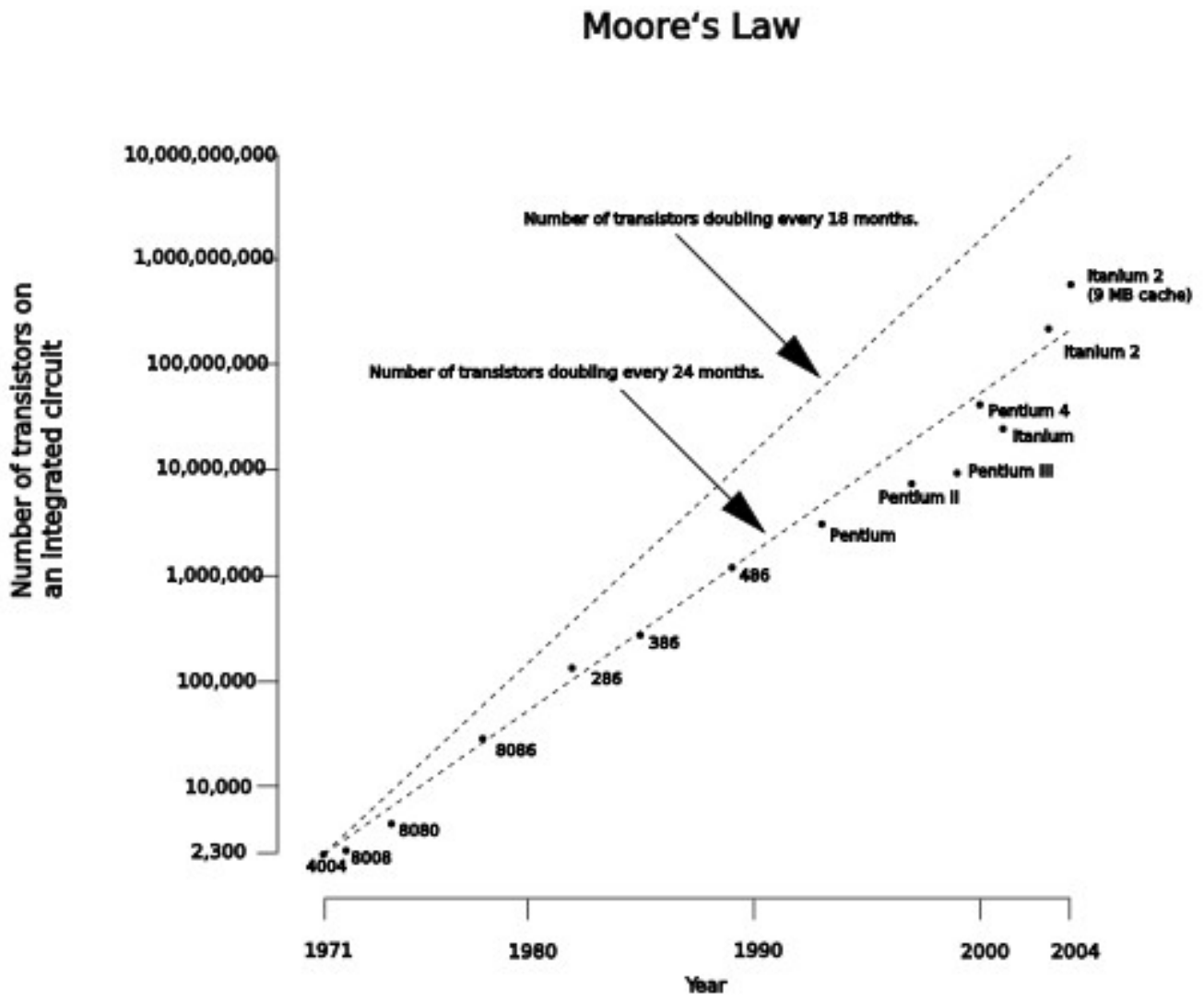
Why multi-core?



Multi-core helps keep work flowing

We can't make them faster !

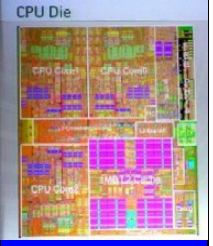




Not meeting our speed goals!

Multi-core Problem

- How unique is each thread?
  - 1 thread per program?
    - We behave this way: browser, compiler, OS all executing at the same time.
  - N threads per program?
    - Eg: browser with multiple windows...
    - If N threads then they must coordinate / synchronize
    - Managing this causes overhead
    - When N > 6 the effect of the overhead begins to overcome the speed improvement



Types of Parallelism

Instruction level parallelism

- The pipeline (single core parallelism)
  - Each stage of pipeline executes a different instruction
    - Through micro-instruction division
  - Re-ordering, branch prediction, etc.
- Used the last 15 years
  - And still going!

Thread-level parallelism (TLP)

- Multi-core strategy
  - Each core can be given a portion of a program to execute
    - Games: core 1 AI, core 2 graphics, core 3 physics, core 4 UI
    - Business: core 1 web server, core 2 database server, core 3-4 system processing
    - Personal computer: core 1 OS, core 2-4 for user programs
    - Science: core 1 OS, core 2 data acquisition device, core 3 data analysis device, core 4 other

The Future

- Single core super scalar processors cannot fully exploit TLP
- Multi-core architectures exploiting TLP is the next major stage in CPU design
  - TLP support at the core level
  - TLP support at the OS level
  - TLP support at the programming level

Programming Support Example

Dijkstra's primitive: PARBEGIN .. PAREND,
also known as COBEGIN .. COEND:

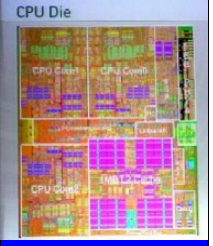
```
PARBEGIN      or:    COBEGIN
  S1;                    S1;
  S2;                    S2;
  ...                    ...
  Sn                     Sn
PAREND;                COEND;
```

Single Instruction Multiple Data

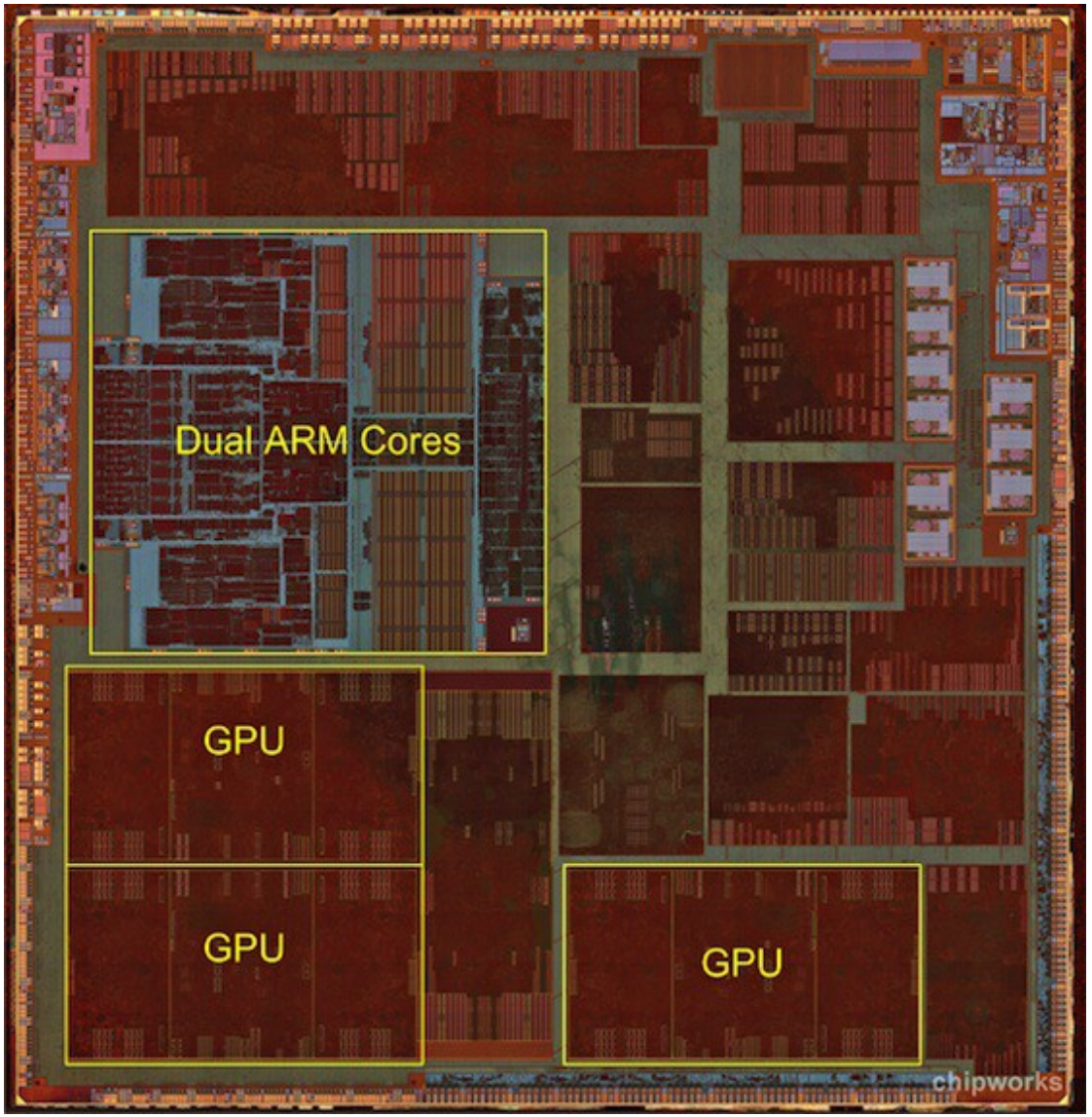
- SIMD
  - Example:
    - Modern GPU, graphics processing unit
  - Implementation:
    - Have a matrix of data
    - A single instruction is executed on all the cells in the matrix of data at the same time
    - Each matrix cell is actually a Core!

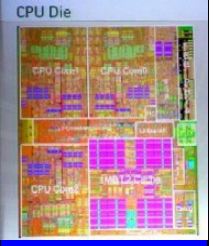
Multiple Instruction Multiple Data

- MIMD
  - Example:
    - Multi-pipeline CPU
  - Implementation:
    - Pipeline architecture per core
    - Share the same RAM
    - Each core has it own instruction and data cache
  - Conclusion
    - The core alone is single instruction/data
    - The chip as a whole in multiple instr/data

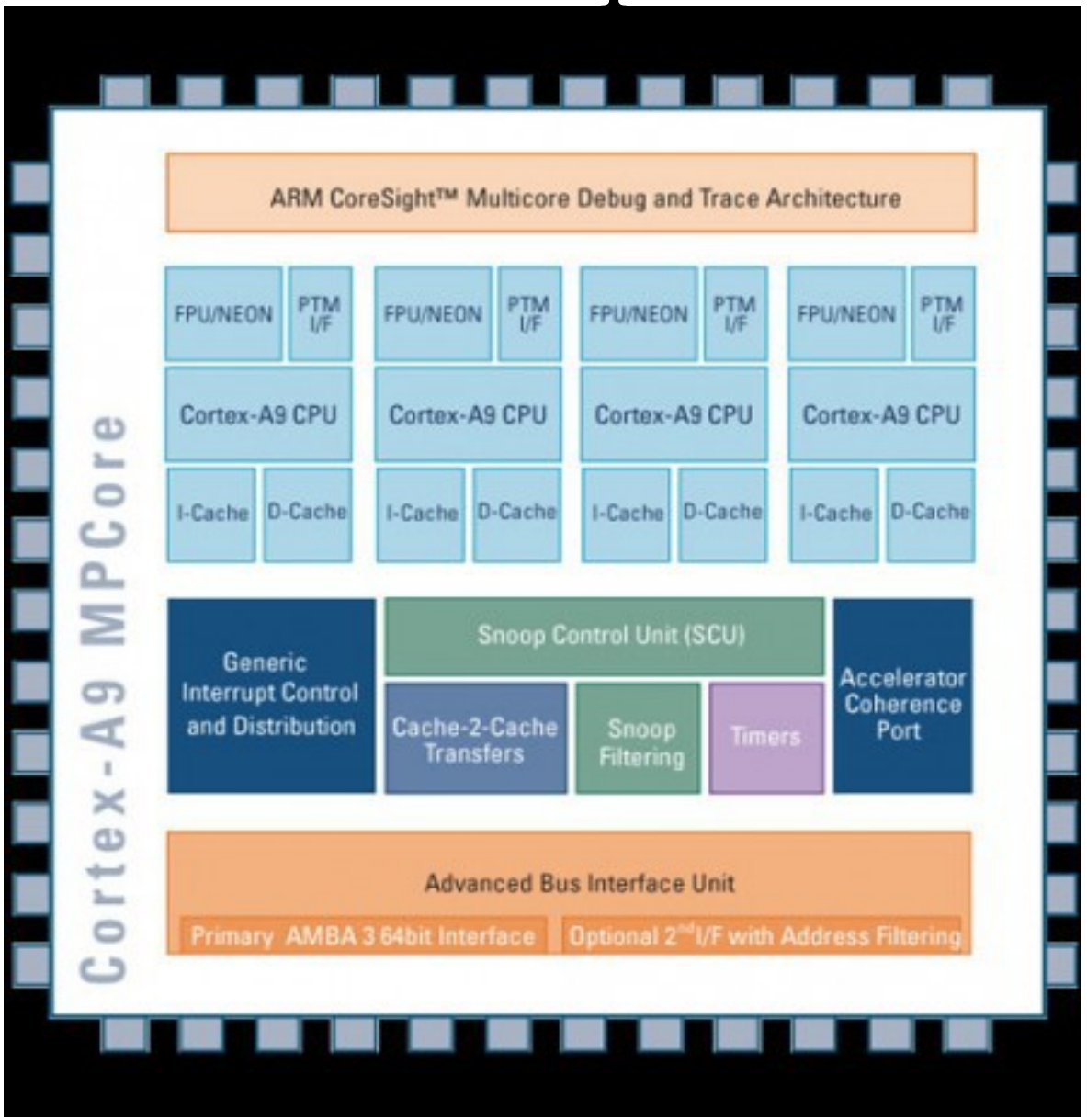


Example





Example 4 ARM Cores



Simultaneous multi-threading

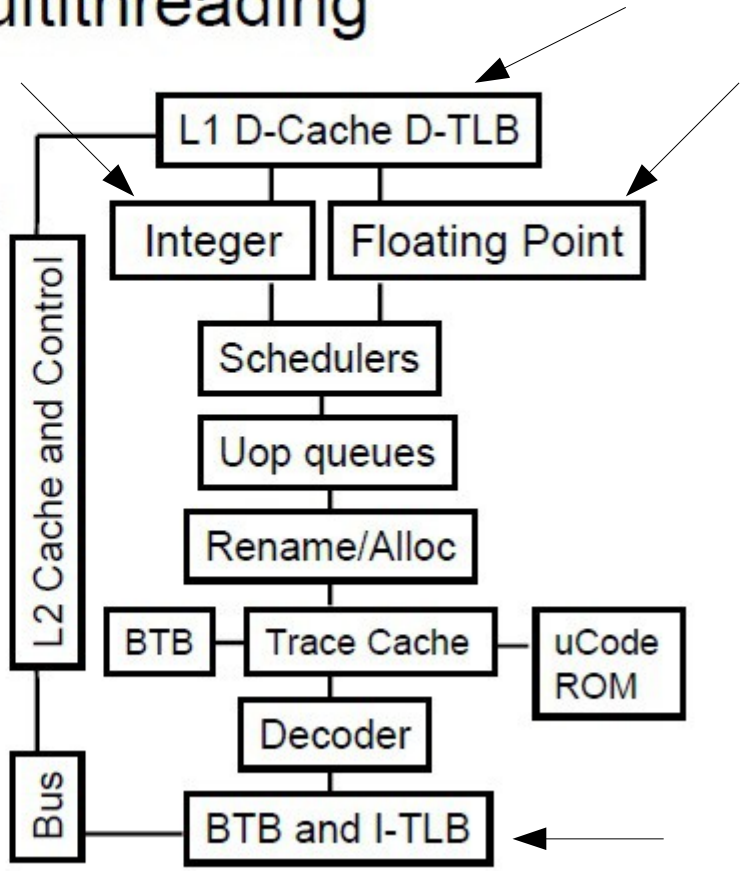
- SMT
  - Multiple independent threads executing simultaneously on the SAME core
    - "weaving multiple threads on the same core"
  - Example
    - A core's integer ALU queue is busy, but the fixed point ALU queue is empty. The process wanting the integer waits, the one wanting the fixed-point runs.

A technique complementary to multi-core:

Simultaneous multithreading

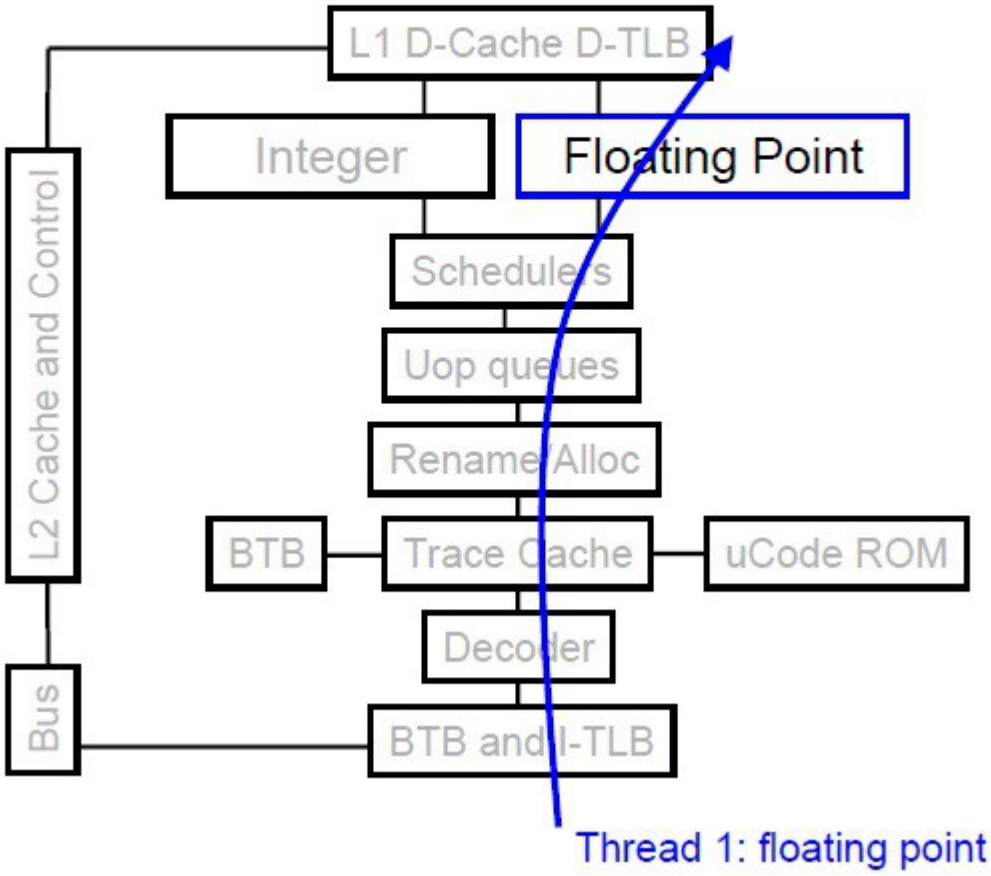
- Problem addressed: The processor pipeline can get stalled:
  - Waiting for the result of a long floating point (or integer) operation
  - Waiting for data to arrive from memory

Other execution units wait unused

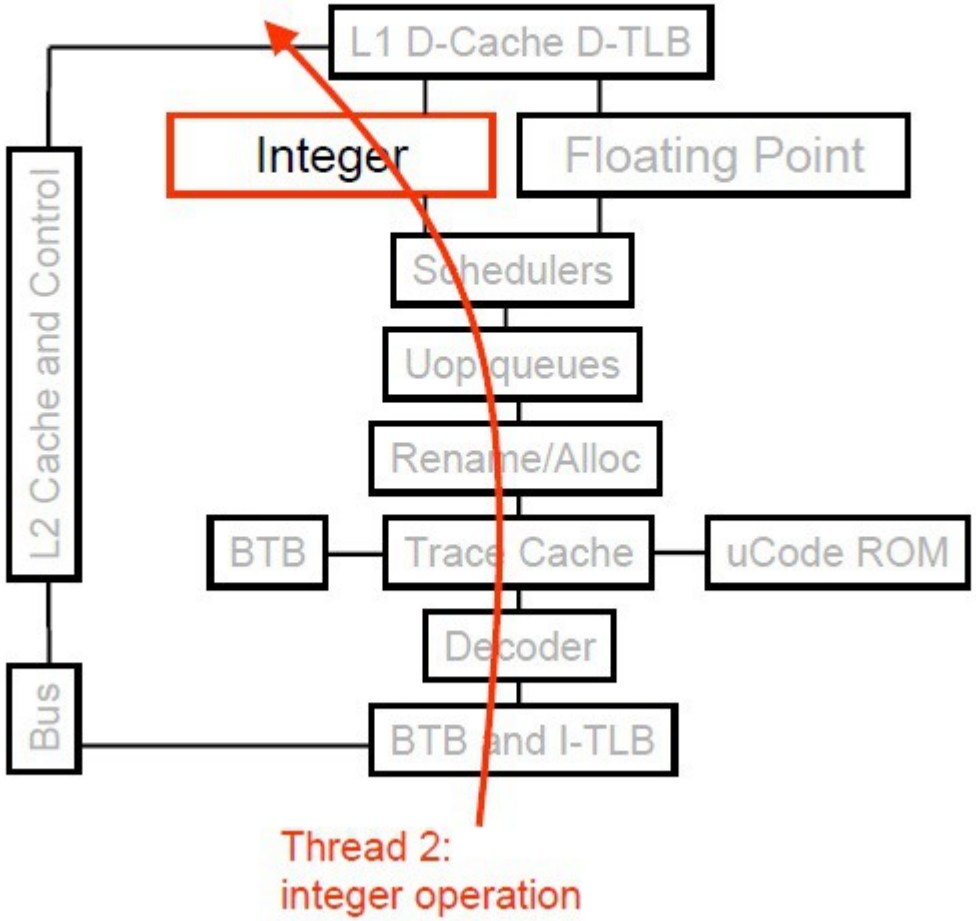


Source: Intel

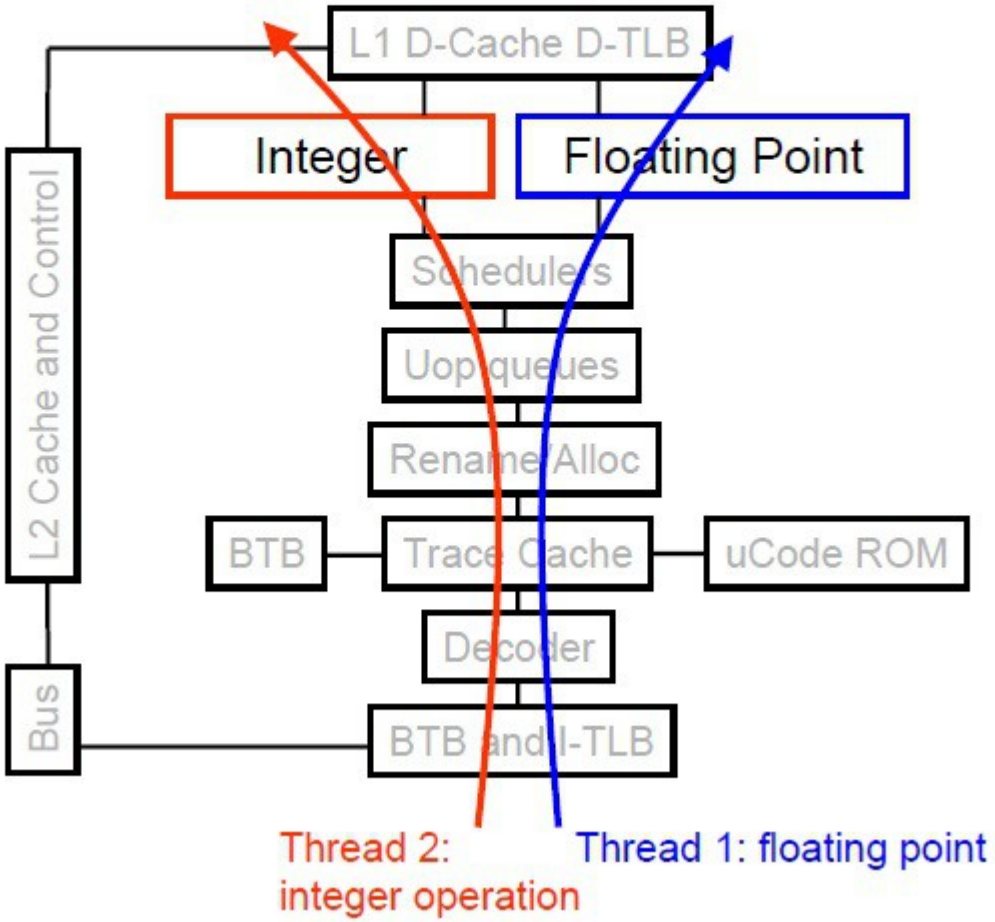
Without SMT, only a single thread can run at any given time



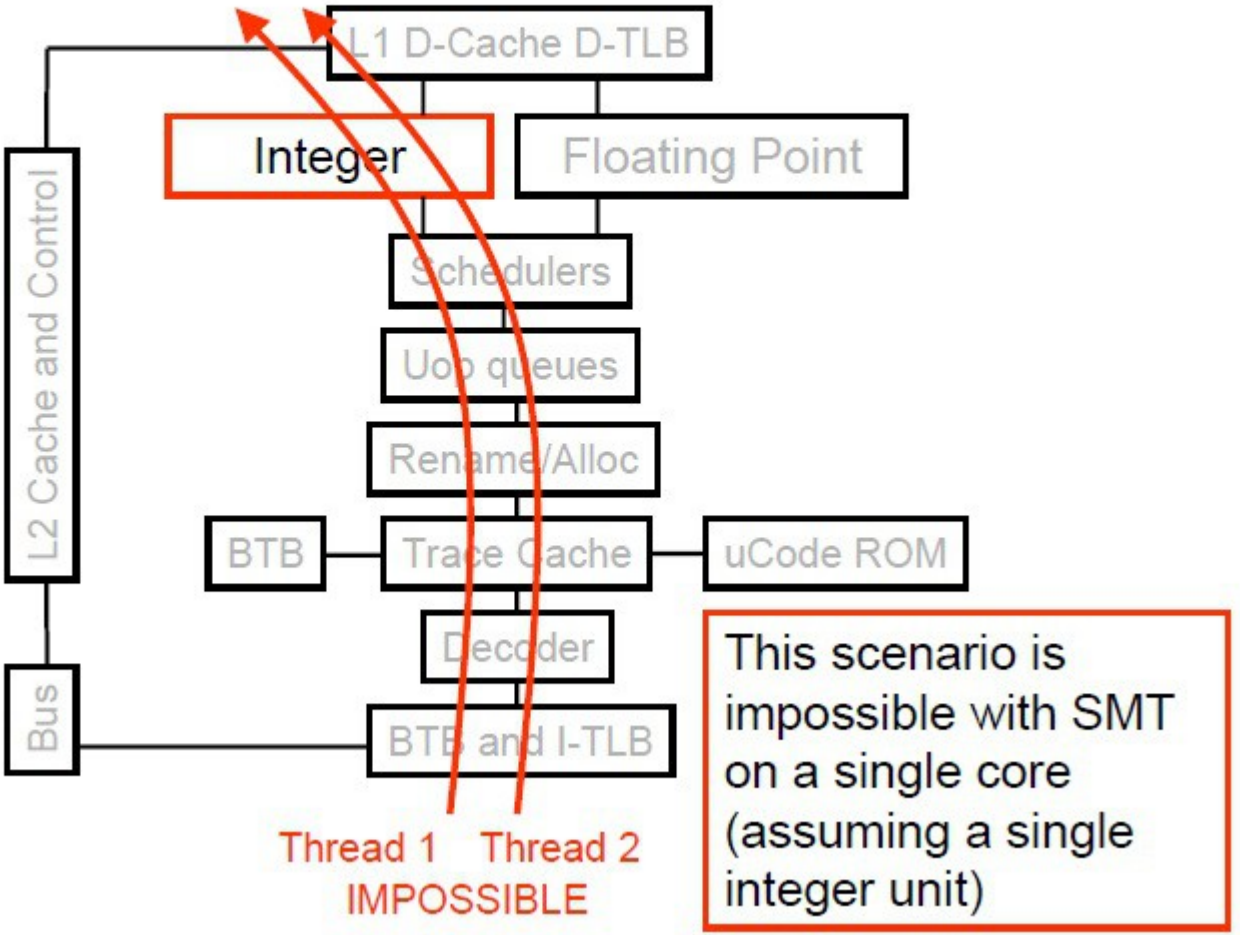
Without SMT, only a single thread can run at any given time



SMT processor: both threads can run concurrently



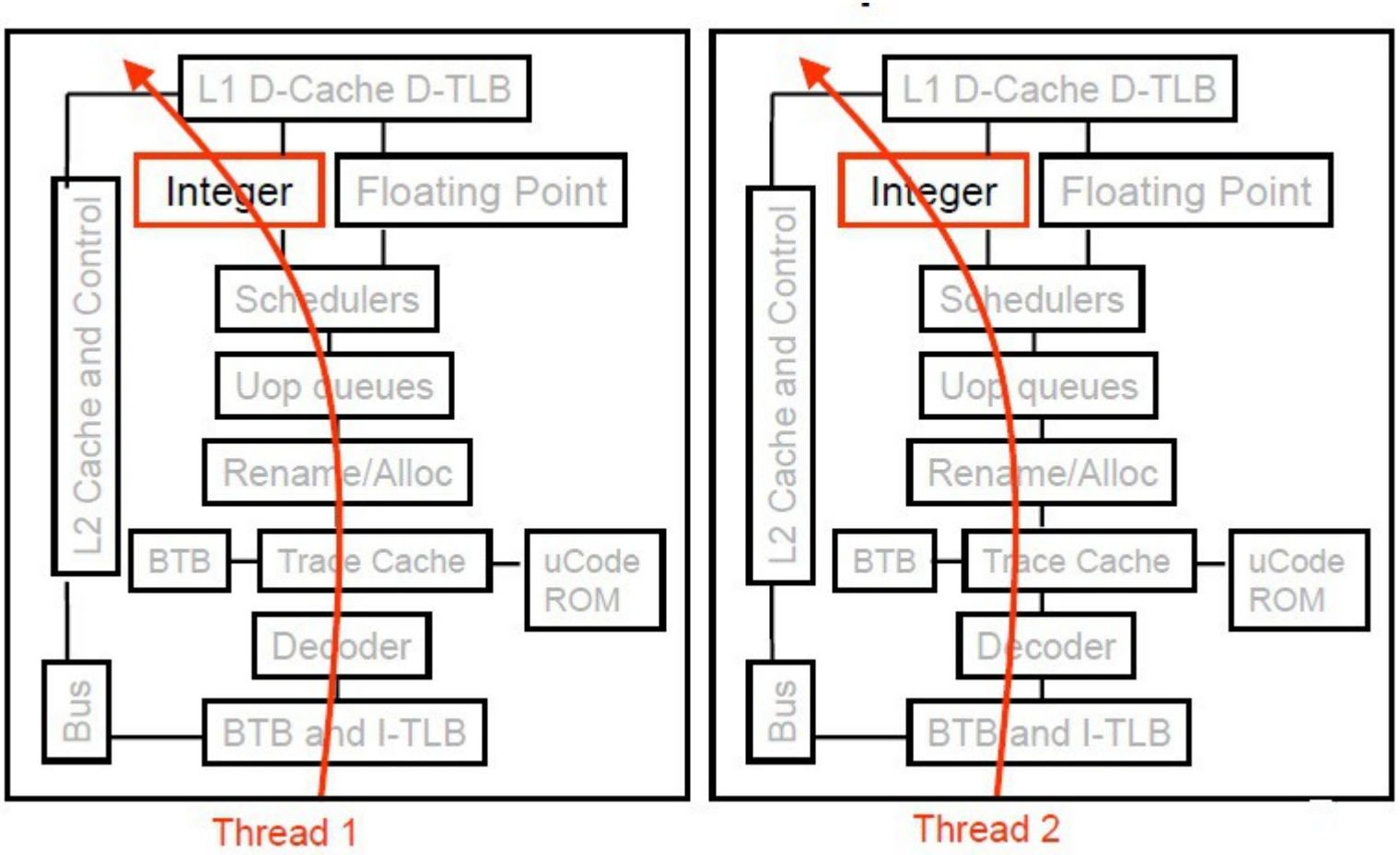
But: Can't simultaneously use the same functional unit



SMT not a "true" parallel processor

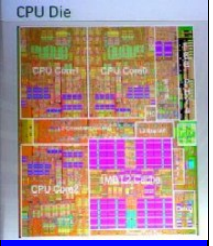
- > 30% threading
- OS sees virtual processor
- Multi-core has resources for each core
    - SMT does not

Combining SMT with Multi-Core

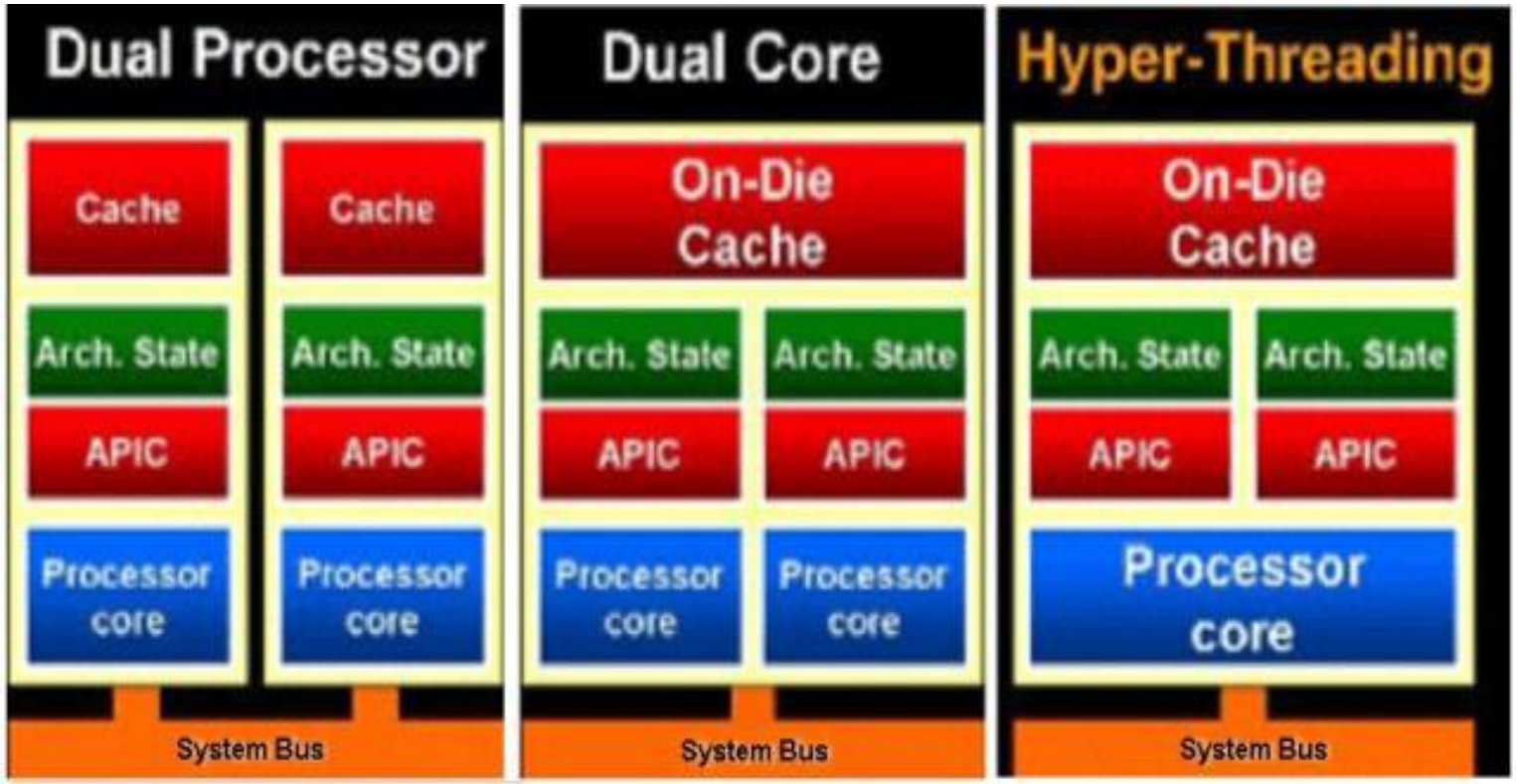


Combining SMT with Multi-Core

- Cores can be SMT enabled or not
- Different combinations:
  - Single-core, non-SMT
  - Single-core with SMT
  - Multi-core, non-SMT
  - Multi-core with SMT
- Number of SMT threads: 2, 4, 8
  - Intel calls them hyper-threads



Core Types Summary

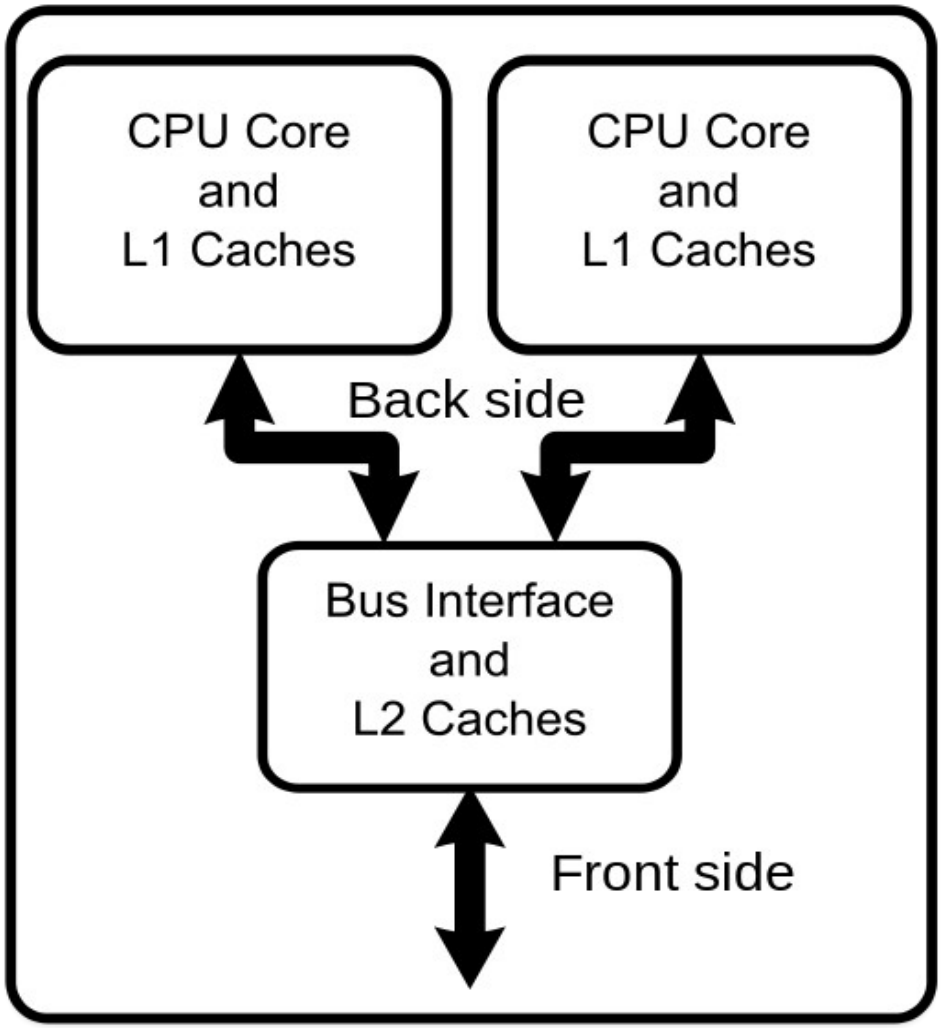


Memory Types

- Shared memory
  - One large common memory shared with all the processors
- Distributed memory
  - Each processor has its own small local memory. This memory is not replicated anywhere else
- Many computers implement both

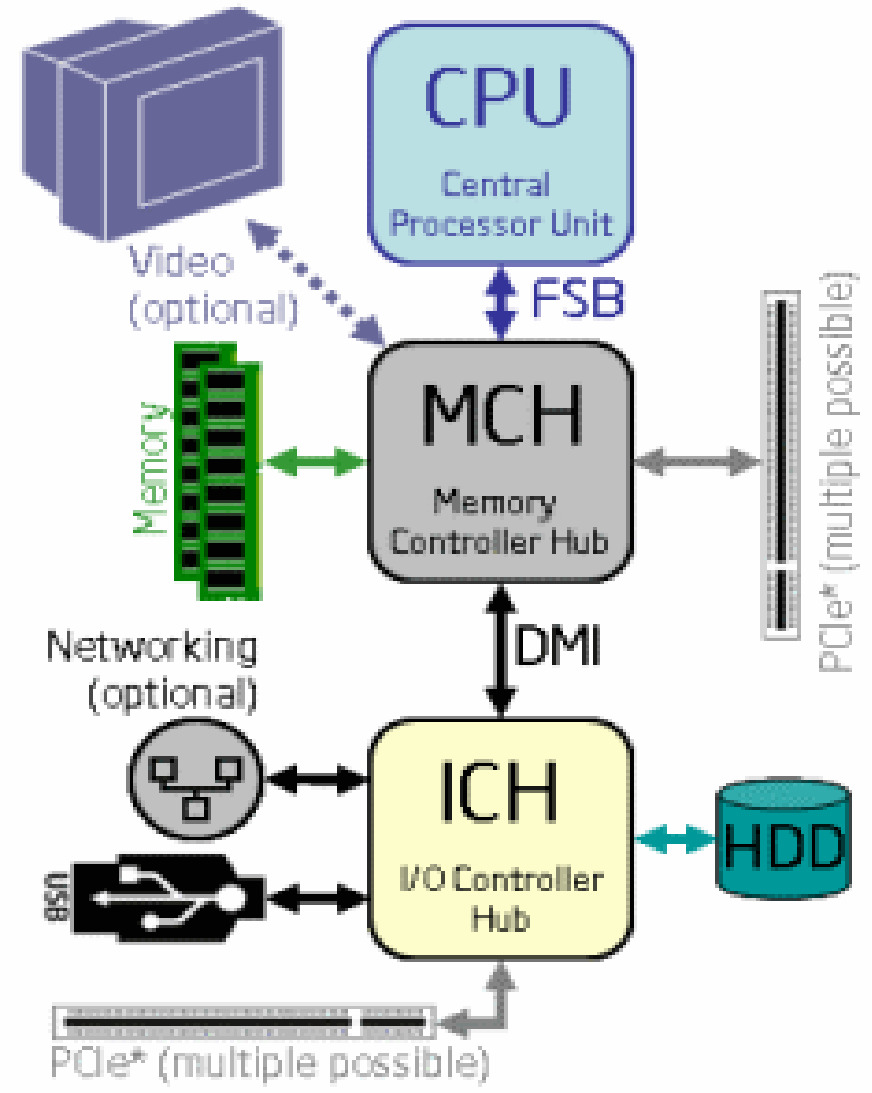

CPU Die

Example


CPU Core
and
L1 Caches
CPU Core
and
L1 Caches
Back side
Bus Interface
and
L2 Caches
Front side

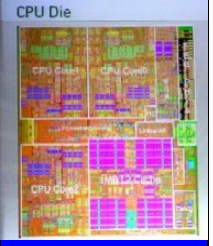
Example

Intel Core Due

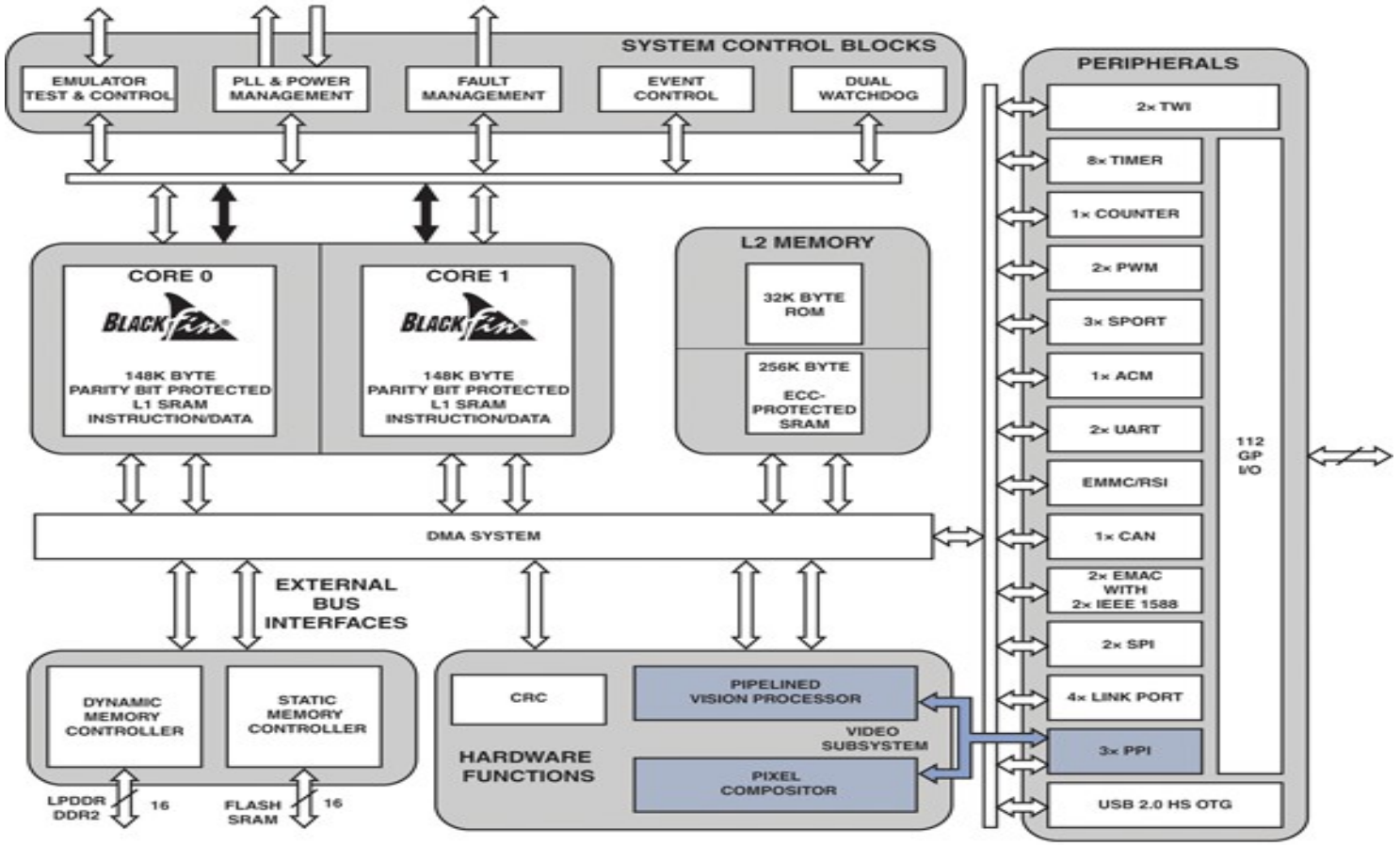


FSB – front side bus

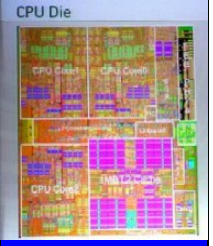
DMI – direct memory interface



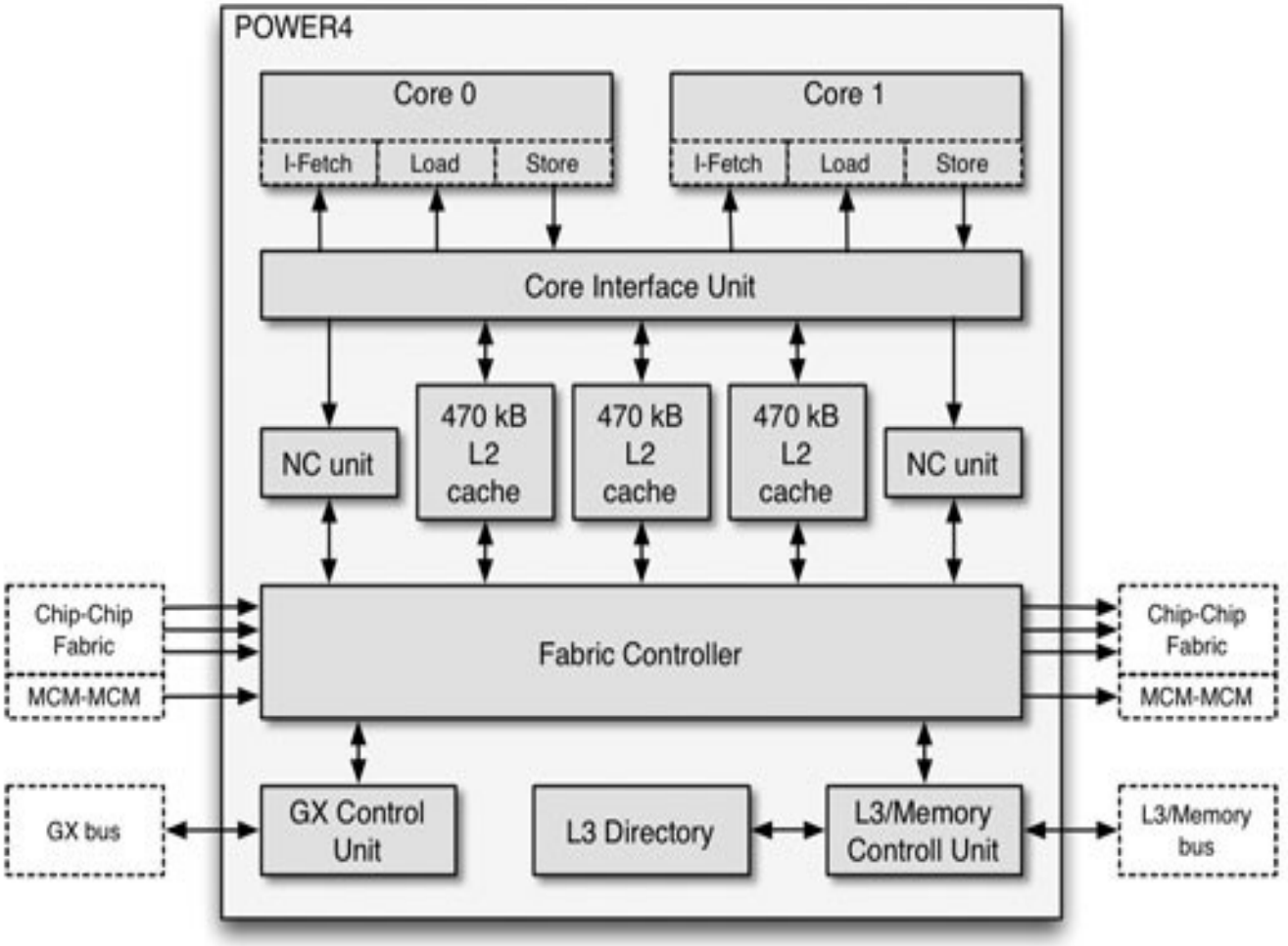
Blackfish Example



DMA – Direct Memory Access


CPU Die

Power PC 4

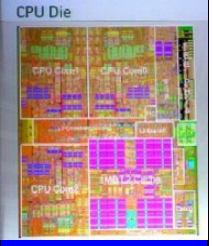

POWER4
Core 0
I-Fetch
Load
Store
Core 1
I-Fetch
Load
Store
Core Interface Unit
NC unit
470 kB L2 cache
470 kB L2 cache
470 kB L2 cache
NC unit
Chip-Chip Fabric
MCM-MCM
Fabric Controller
Chip-Chip Fabric
MCM-MCM
GX bus
GX Control Unit
L3 Directory
L3/Memory Controll Unit
L3/Memory bus

Memory Hierarchy

- Not shared:
  - Pipeline
  - L1 cache private
- Shared:
  - L2 cache (sometimes private)
  - L3 cache
  - Memory (RAM)
- Special case: -- simultaneous multi-threading --
  - All caches shared

Private vs. Shared Caches

- Private:
  - Closer to core so faster access
  - Less contention
- Shared:
  - Different cores can share the same cache
  - More space available for big instruction or big data programs



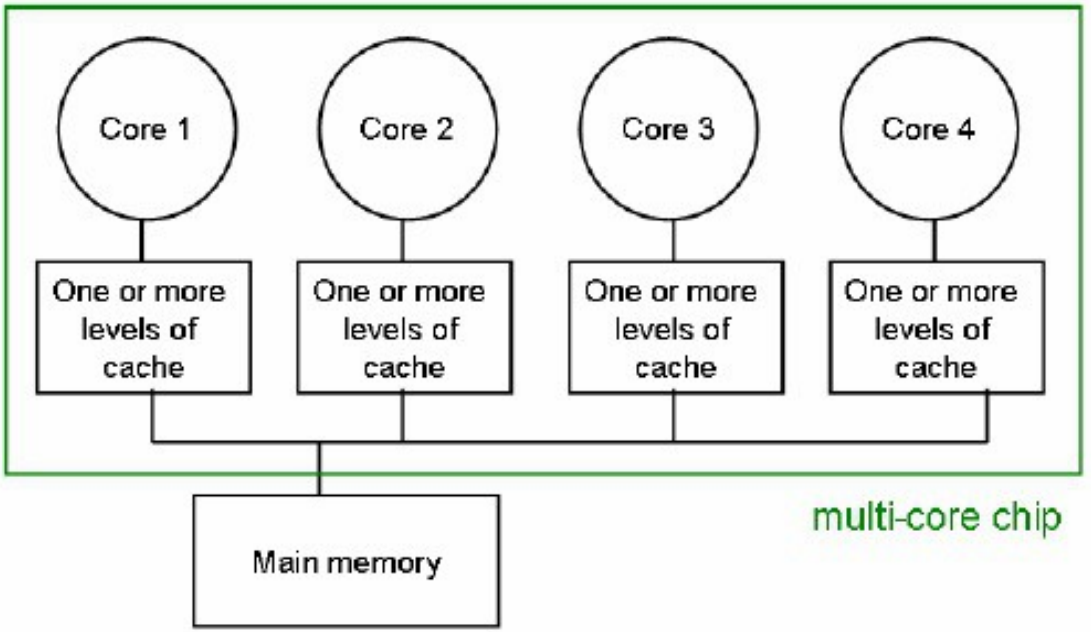
Problem #1

Contention

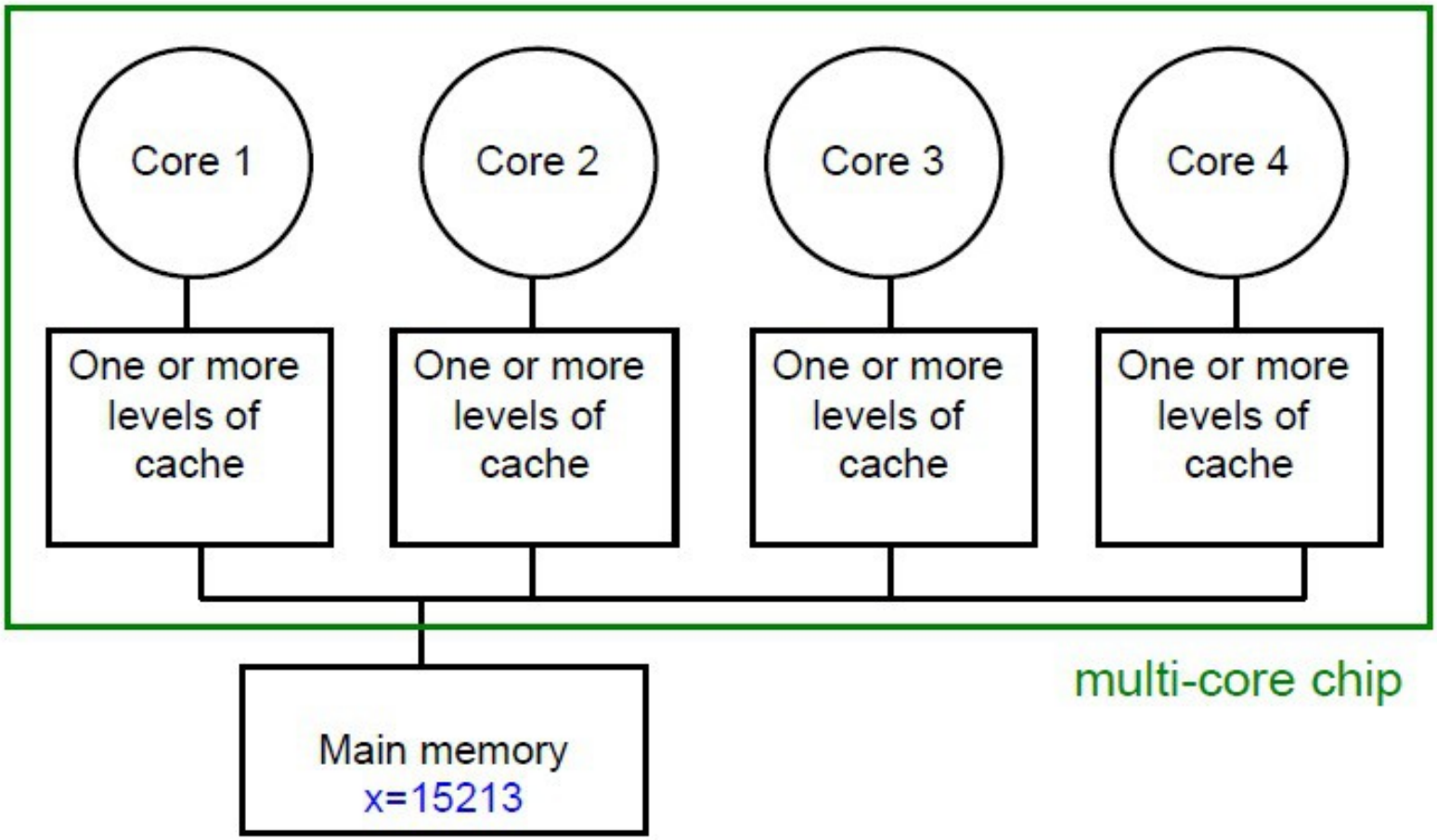
- When more than one core needs access to the same cache
  - One core must wait
    - Loose processing time
  - Need circuitry to handle competition
    - General CPU slowdown (small)

Cache coherence problem

- Since we have private caches:
  How to keep the data consistent across caches?
- Each core should perceive the memory as a monolithic array, shared by all the cores

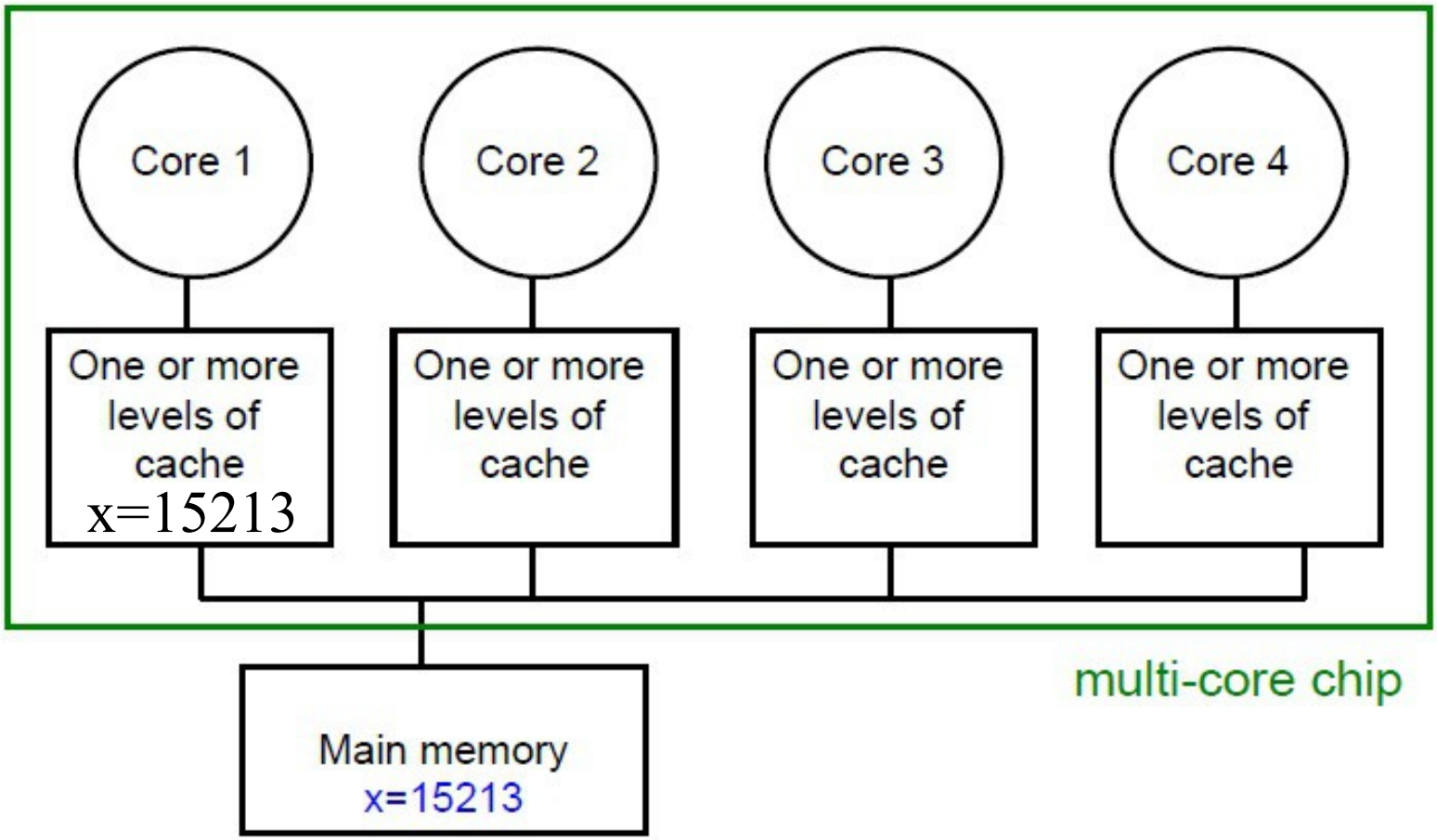


Suppose variable x initially contains 15213



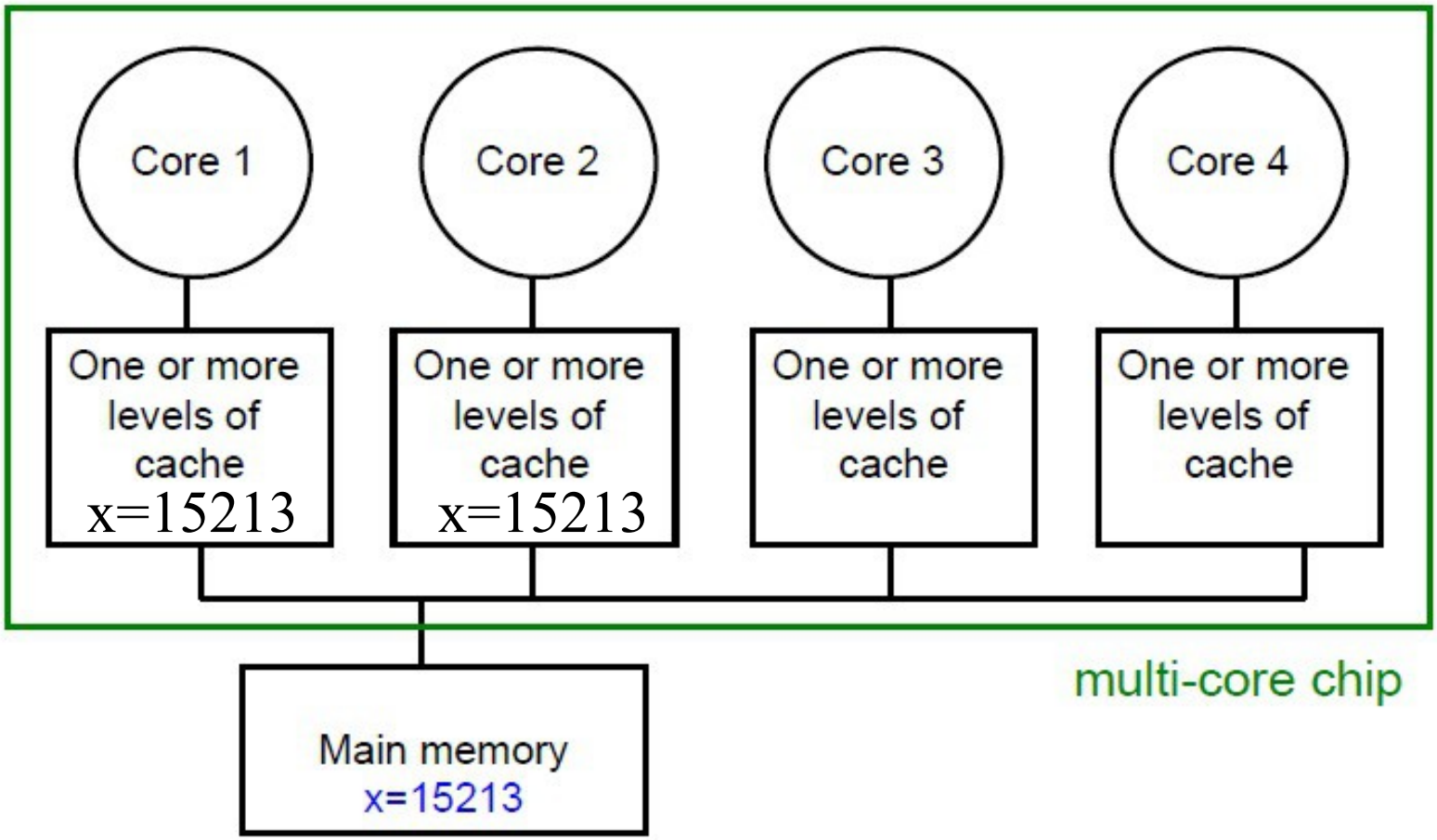

CPU Die

Core 1 read x

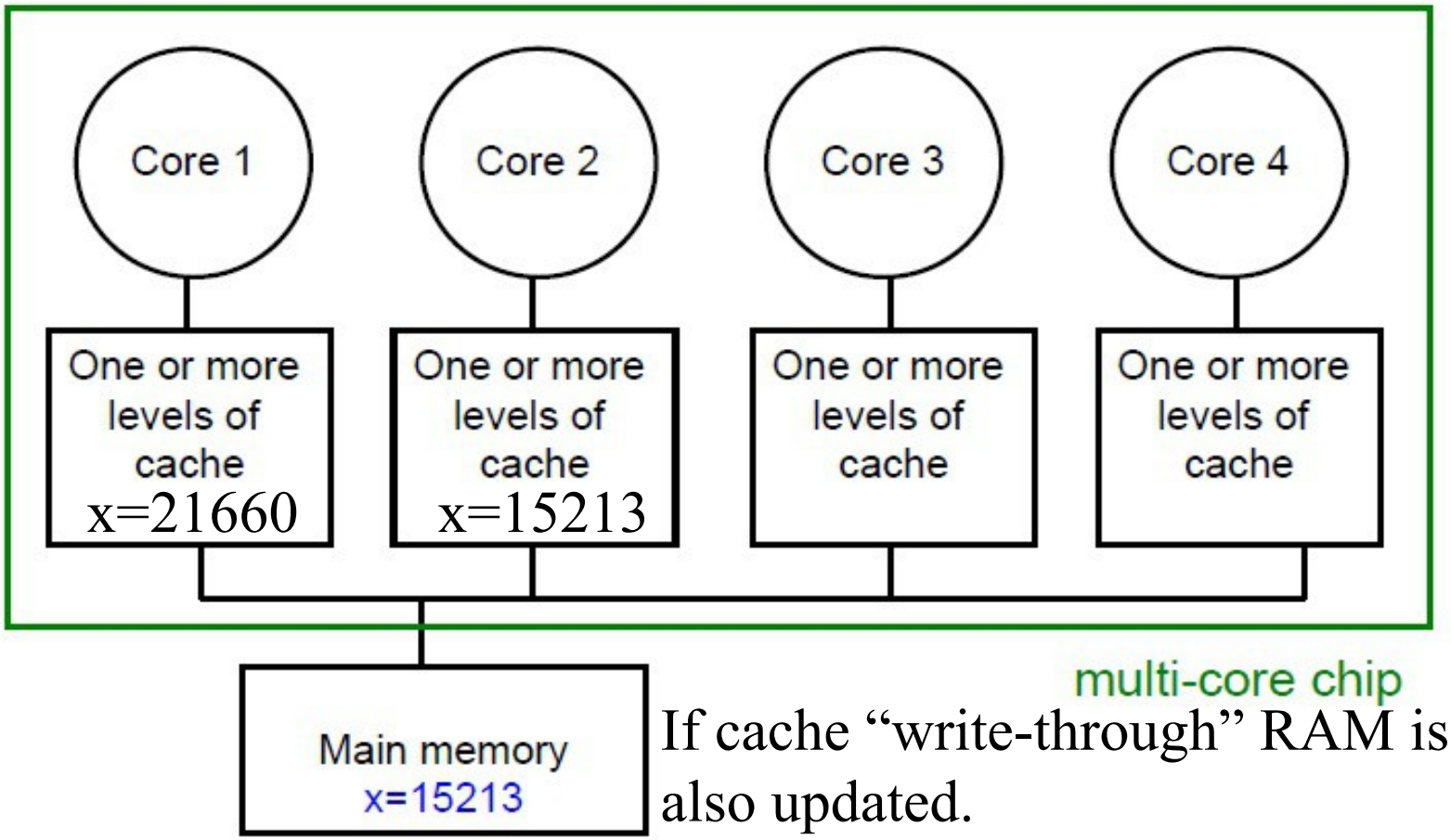

Core 1
Core 2
Core 3
Core 4
One or more levels of cache
x=15213
One or more levels of cache
One or more levels of cache
One or more levels of cache
multi-core chip
Main memory
x=15213


CPU Die

Core 2 read x

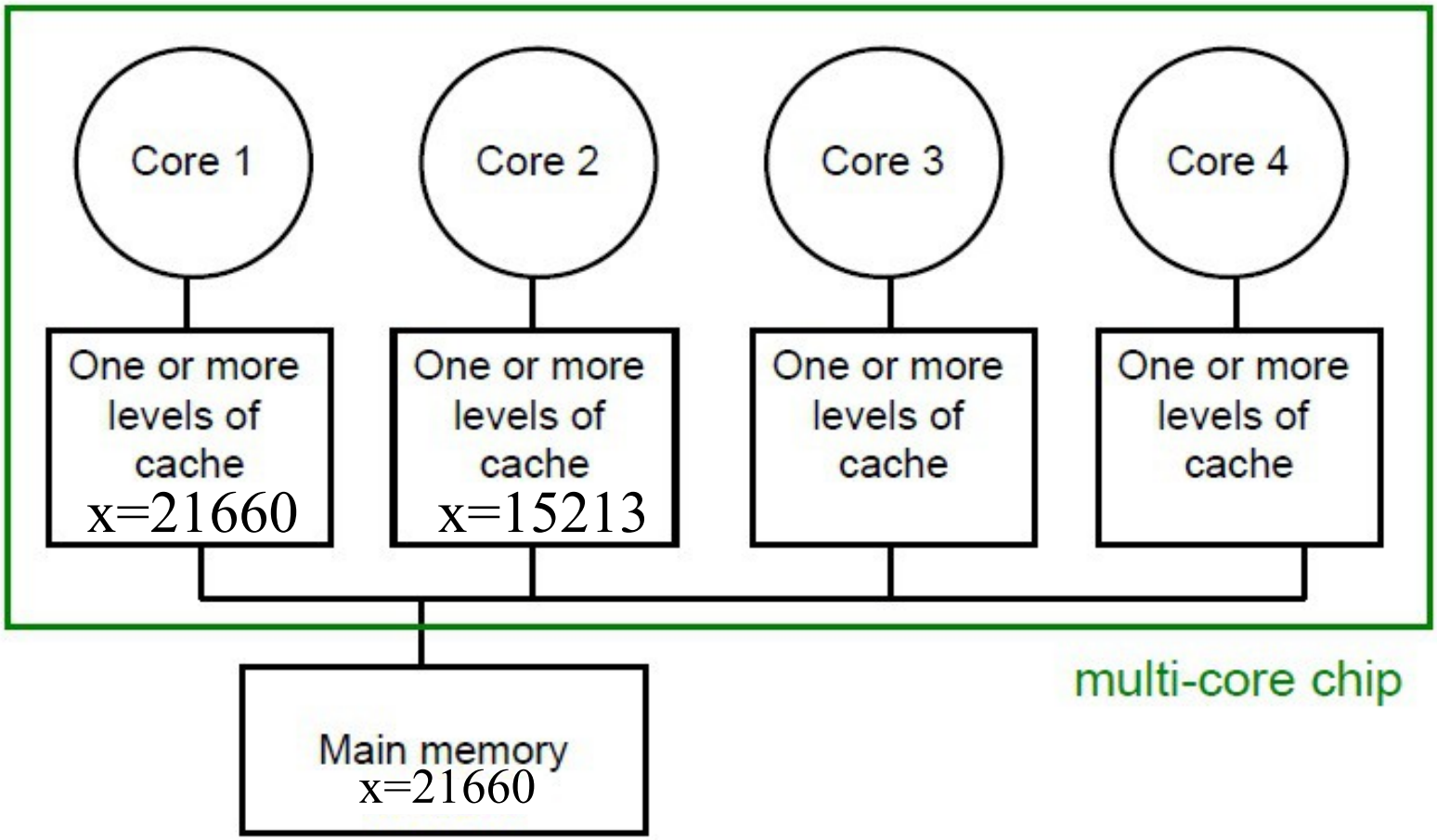

Core 1
Core 2
Core 3
Core 4
One or more levels of cache
x=15213
One or more levels of cache
x=15213
One or more levels of cache
One or more levels of cache
multi-core chip
Main memory
x=15213

Core 1 writes to x, setting it to 21660



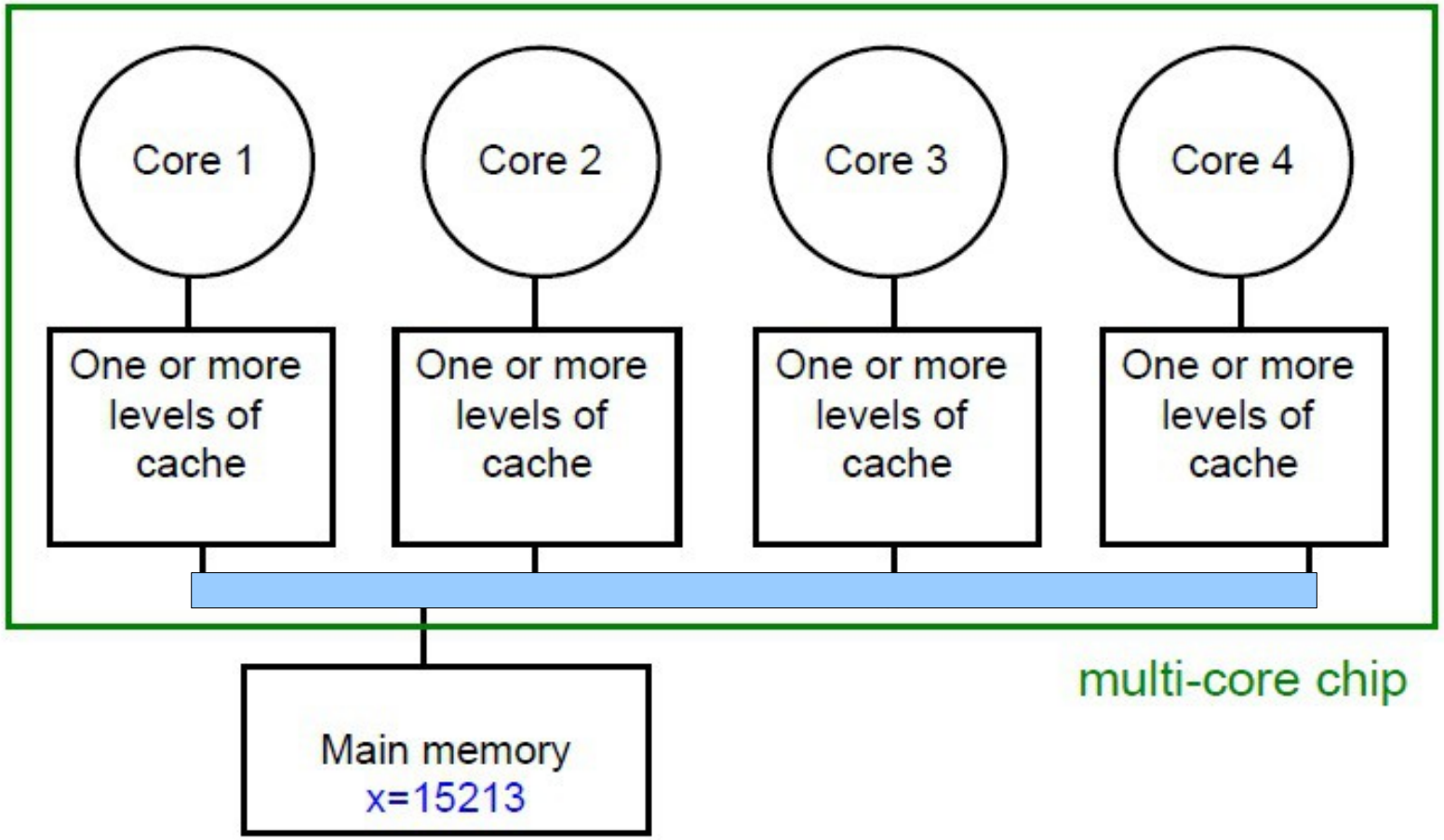
If cache "write-through" RAM is also updated.

Core 2 trys to read X and gets an old copy!



Solution: invalidation + snooping

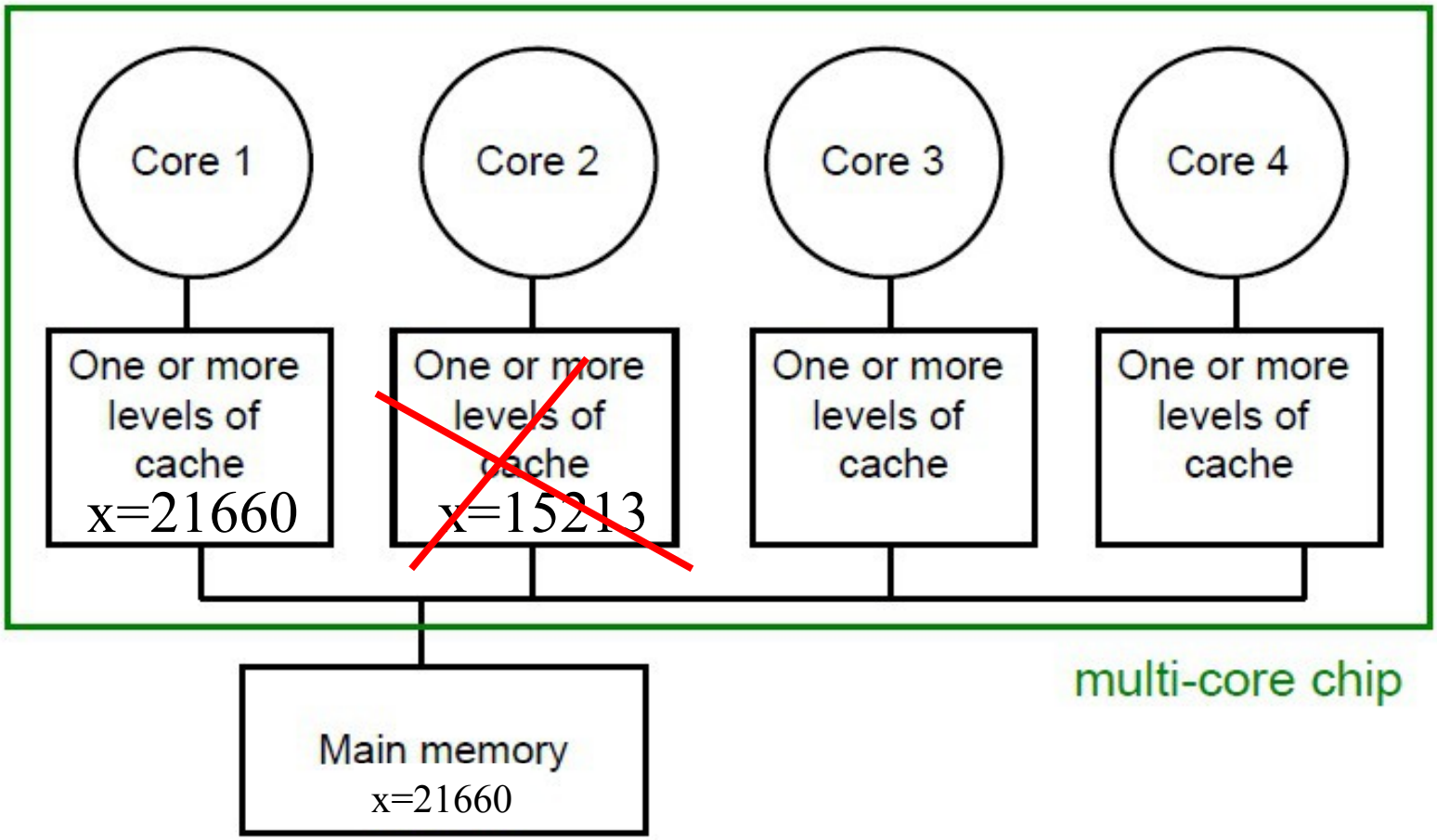
INTER-CORE BUS



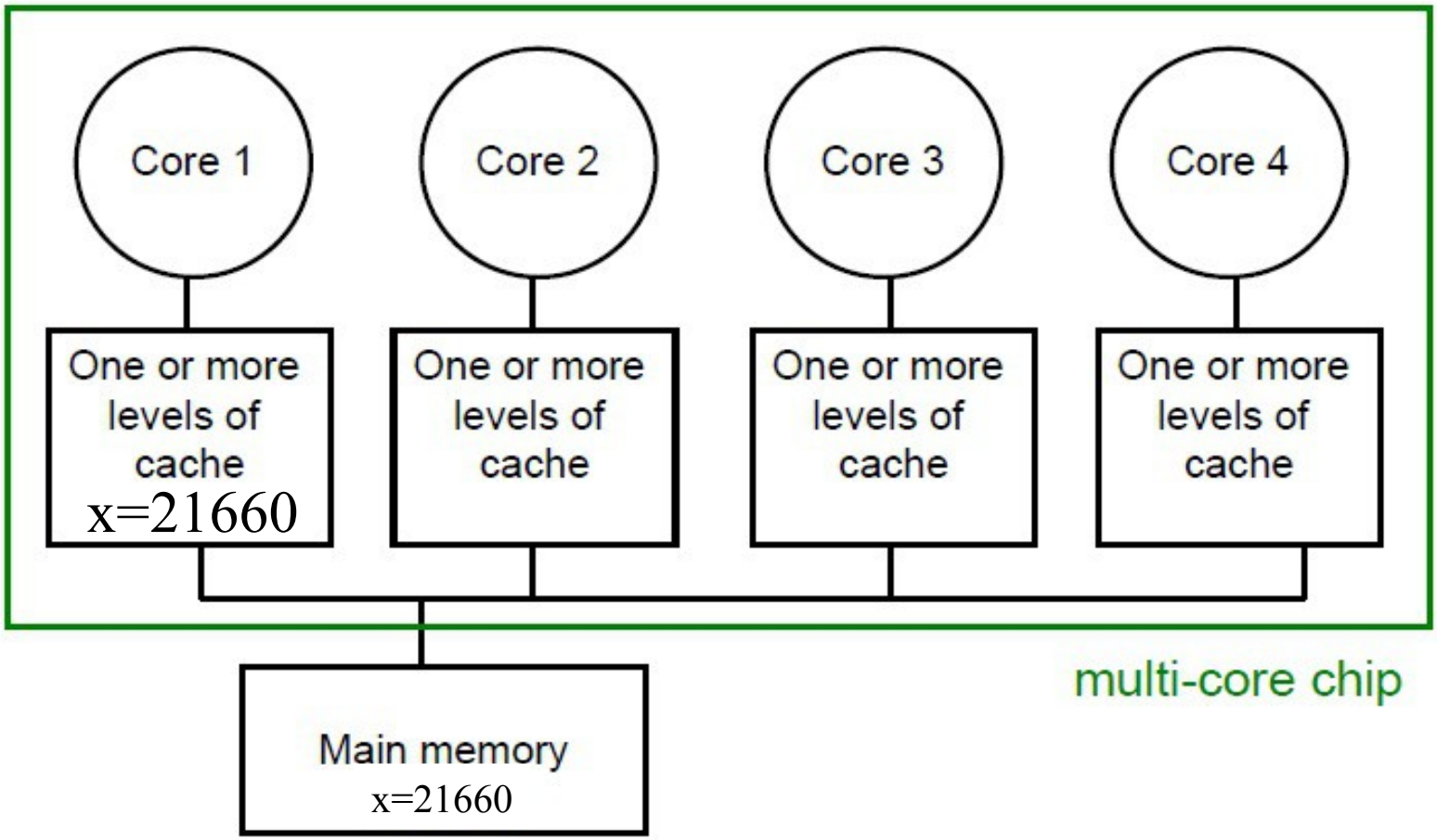
Algorithm

- Invalidation:
  - If a core writes to a cache, all other copies are flagged invalid.
- Snooping:
  - All cores monitor the bus connecting the cores for write operations

Core 1 writes to x, setting it to 21660



After invalidation: empty..



Core 2 reads and gets from memory