Lecture 18:
Multi-core and Multi-threading

Acknowledgement: Some slides were adopted from “Multi-core architectures” prepared by Prof. Barbic at USC and https://www.cise.ufl.edu/~mssz/.../CDA3101-L34-35-multicore-PrabhatMishra.ppt

Why Multi-core Architecture?

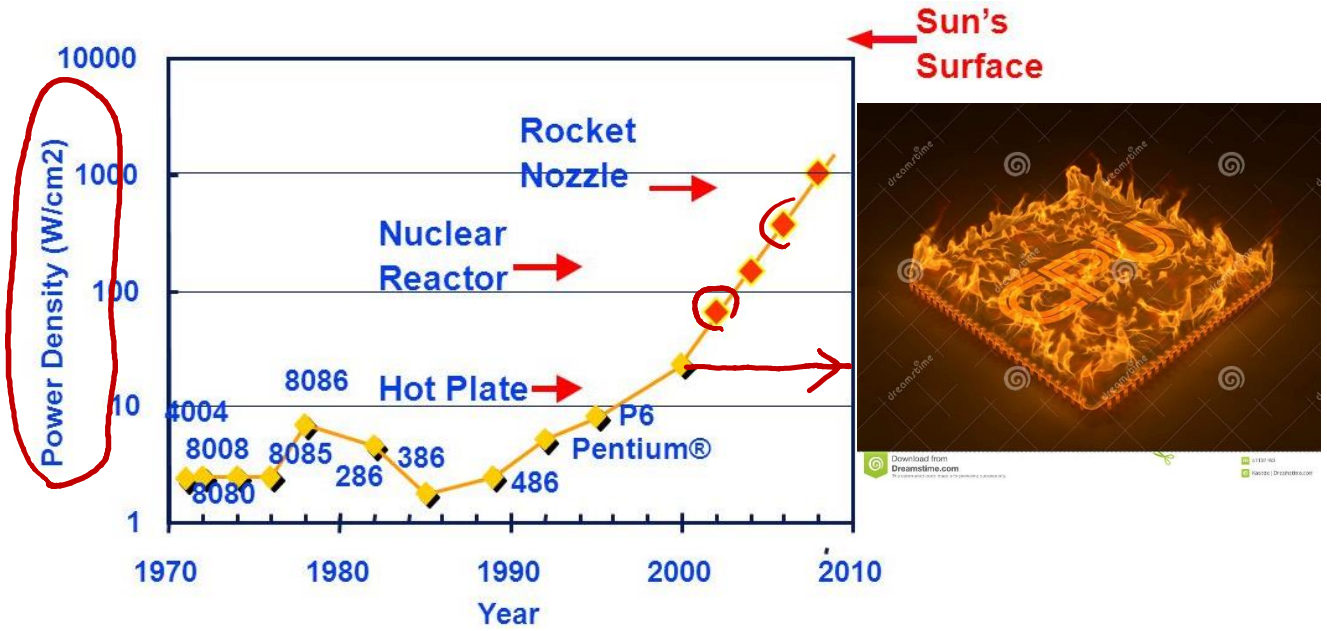
- Never ending story …
  - **Complex Applications**
  - **Faster Computation**
  - How far did we go with uniprocessors?

Why multi-core Architecture?

- difficult to make single-core clock frequencies even higher
  - ✓ hard to extract more ILP b/c larger instruction windows are very expensive, slow, power-hungry circuits

- deeply pipelined circuits:
  - ✓ heat problems
  - ✓ speed of light problems
  - ✓ difficult design and verification
  - ✓ large design teams necessary
  - ✓ server farms need expensive air-conditioning

Power and Thermal Limit



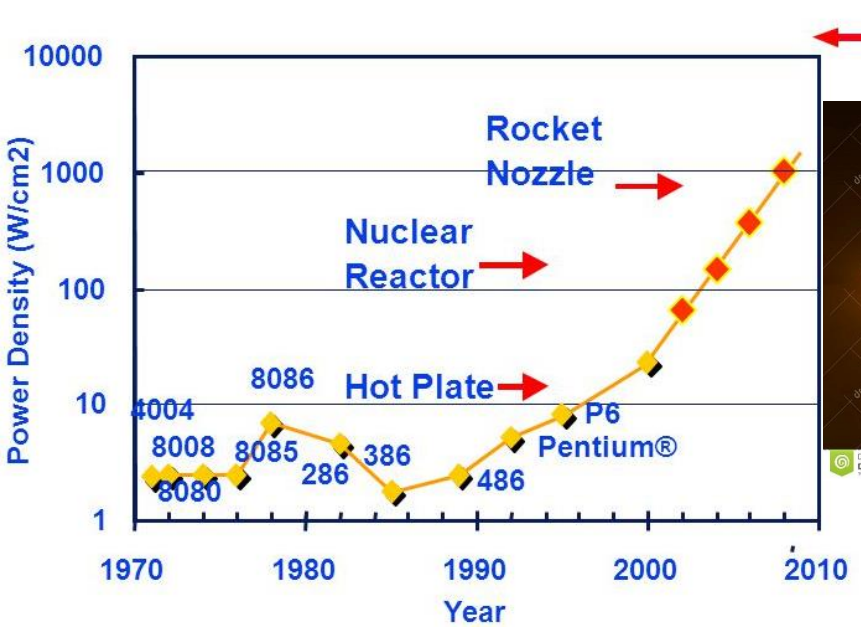
https://doi.ieeecomputersociety.org/cms/Computer.org/dl/mags/mi/2012/03/figures/mmi20120301221.gif
https://hsto.org/storage2/e68/410/9b5/e684109b586e5e9612cff3befdf36fbe.png

How to Reduce Power Consumption

- Multicore
  - One core with frequency 2 GHz
  - Two cores with 1 GHz frequency (each)
    - Same performance
    - Two 1 GHz cores require half power/energy
      - Power ➔ freq^3
      - 1GHz core needs one-fourth power compared to 2GHz core.

- New challenges – Performance
  - How to utilize the cores
  - It is difficult to find parallelism in programs to keep all these cores busy.

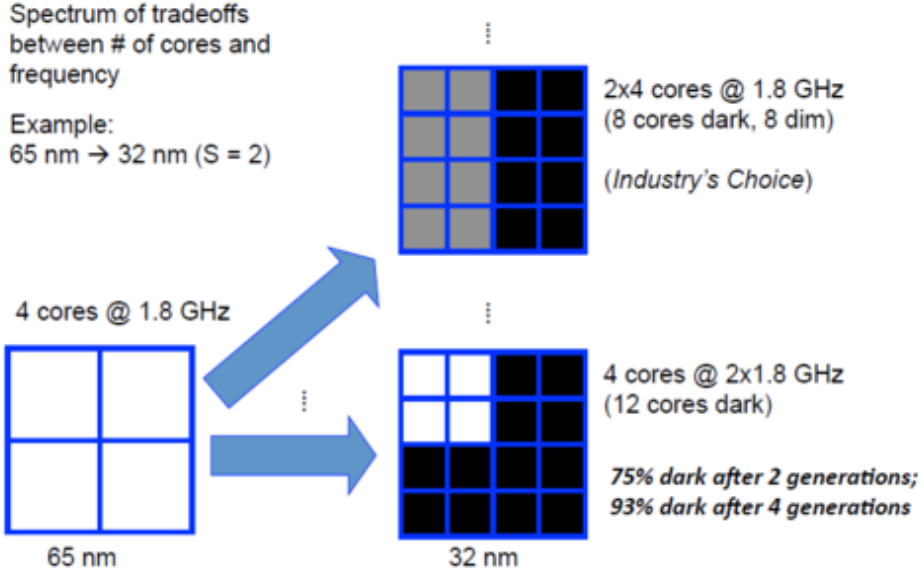
Power and Thermal Limit



T boosting

"dark silicon"

dim silicon



https://doi.ieeecomputersociety.org/cms/Computer.org/dl/mags/mi/2012/03/figures/mmi20120301221.gif
https://hsto.org/storage2/e68/410/9b5/e684109b586e5e9612cff3befdf36fbe.png

Why Multi-core Architecture?

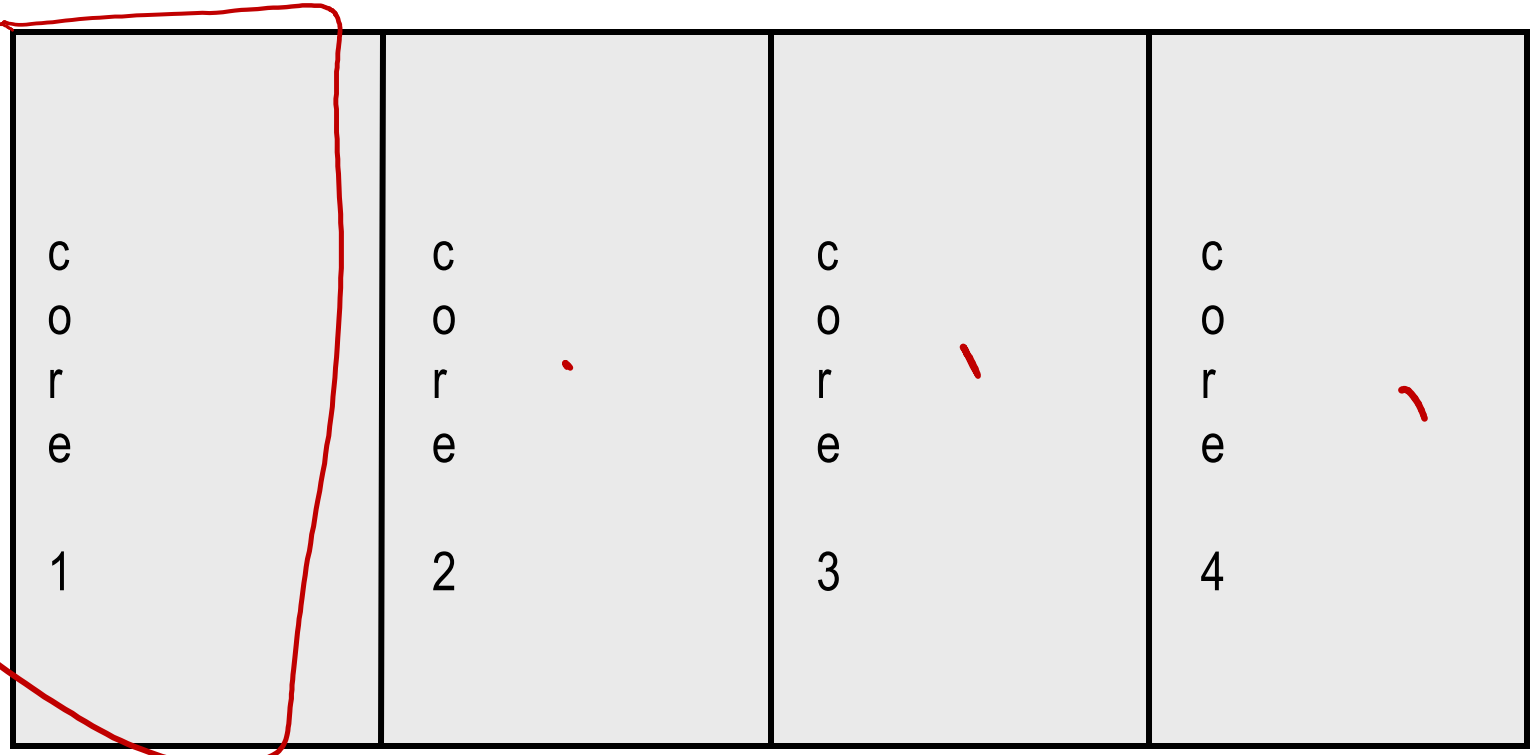
- Never ending story …
  - **Complex Applications**
  - **Faster Computation**
  - How far did we go with uniprocessors?
- **Parallel Processors** now play a major role
  - Logical way to improve performance
    - Connect multiple microprocessors
  - Not much left with ILP exploitation
  - Server and embedded software have parallelism
- Multiprocessor architectures will become increasingly attractive
  - Due to slowdown in advances of uniprocessors

ILP versus TLP

- instruction-level parallelism (ILP)
  - ✓ parallelism at the machine-instruction level
  - ✓ the processor can re-order, pipeline instructions, split them into microinstructions, do aggressive branch prediction, etc.
  - ✓ instruction-level parallelism enabled rapid increases in processor speeds over the last 15 years
- thread-level parallelism (TLP)
  - ✓ parallelism on a more coarser scale
  - ✓ server can serve each client in a separate thread (web server, database server)
  - ✓ a computer game can do AI, graphics, and physics in three separate threads
  - ✓ single-core superscalar processors cannot fully exploit TLP
  - ✓ multi-core architectures are the next step in processor evolution: explicitly exploiting TLP

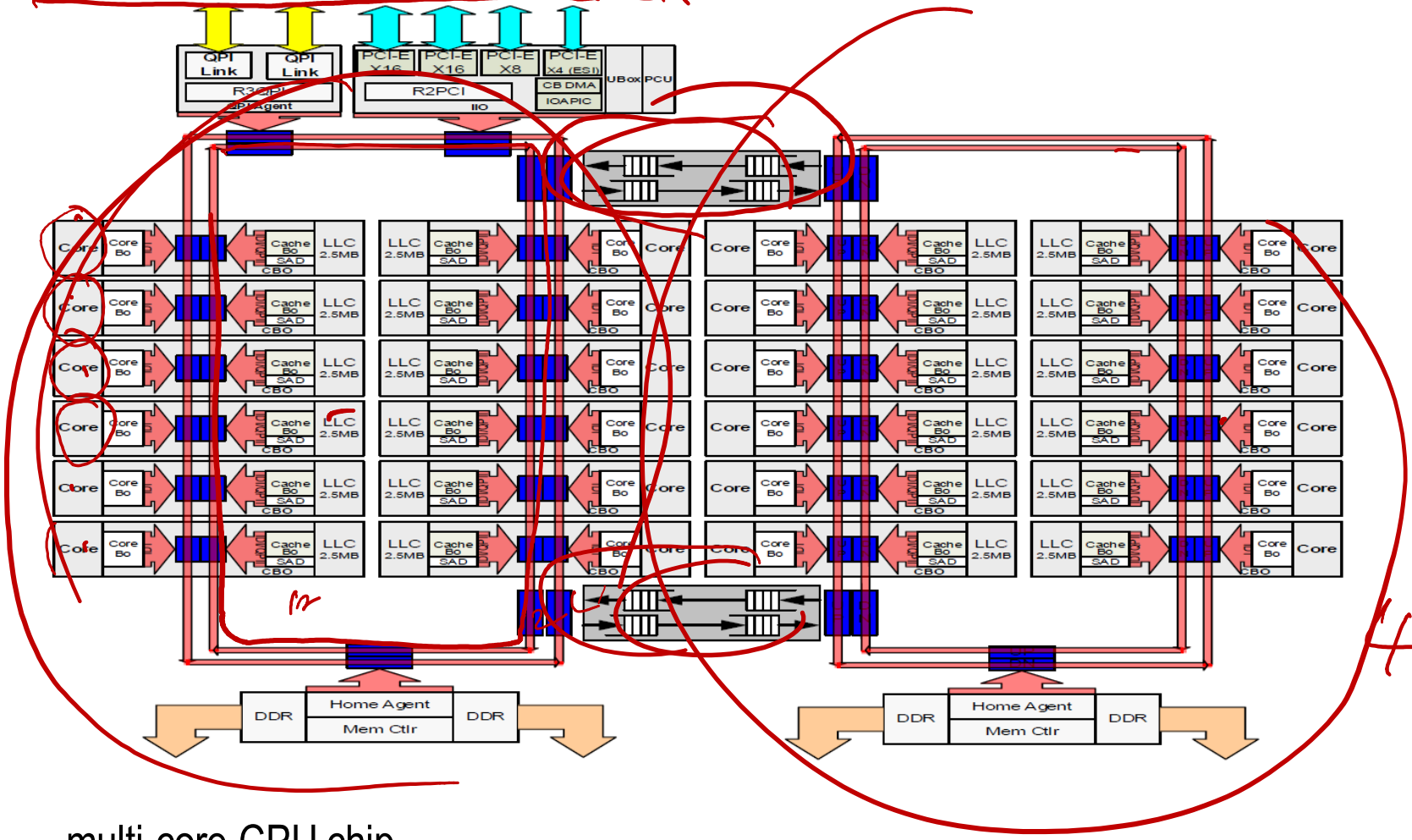
Multi-core CPU chip

- multiple cores fit on a single processor socket
  - ✓ also called CMP (Chip Multi-Processor)



Multi-core Architectures

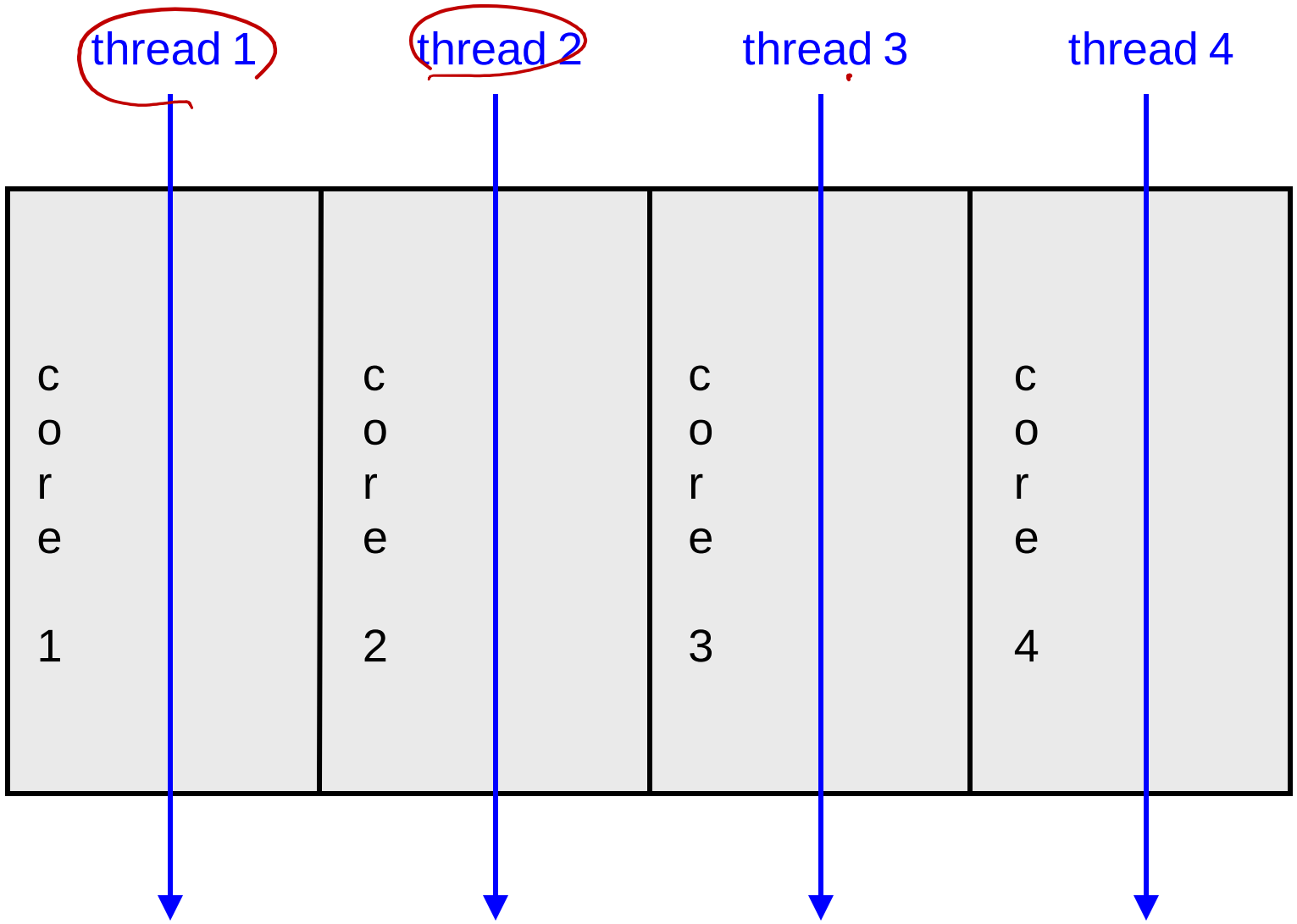
Intel® Xeon® Processor E5 v4 Product Family HCC



multi-core CPU chip

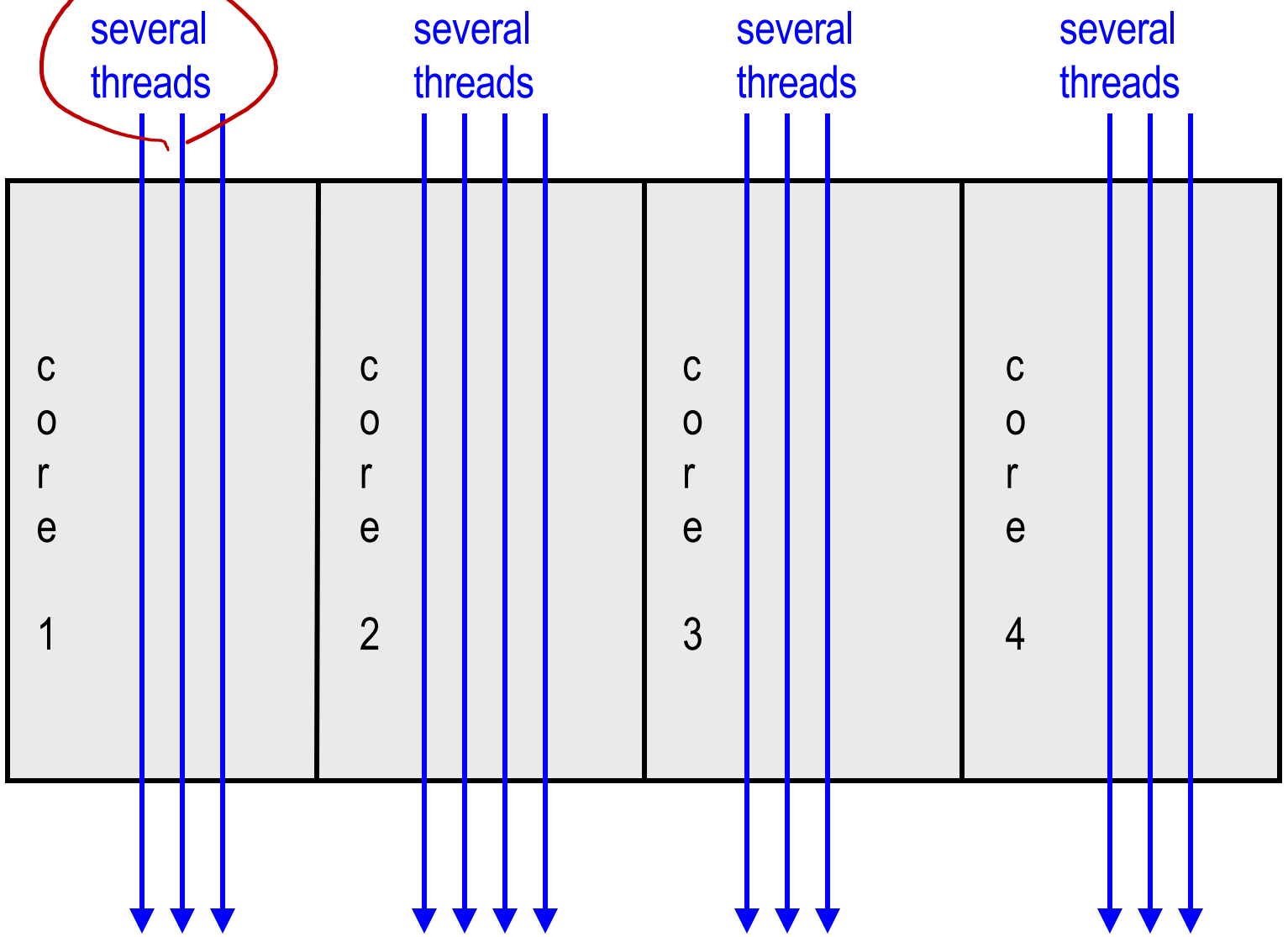
Cores Run in Parallel

- exploiting thread-level parallelism



Cores Run in Parallel

- threads can be time-sliced (just like on a uniprocessor) in each core



Interaction w/ OS

- OS perceives each core as a separate processor
- OS scheduler maps threads/processes to different cores
- most major OS support multi-core today
  - ✓ Windows, Linux, Mac OS X, …

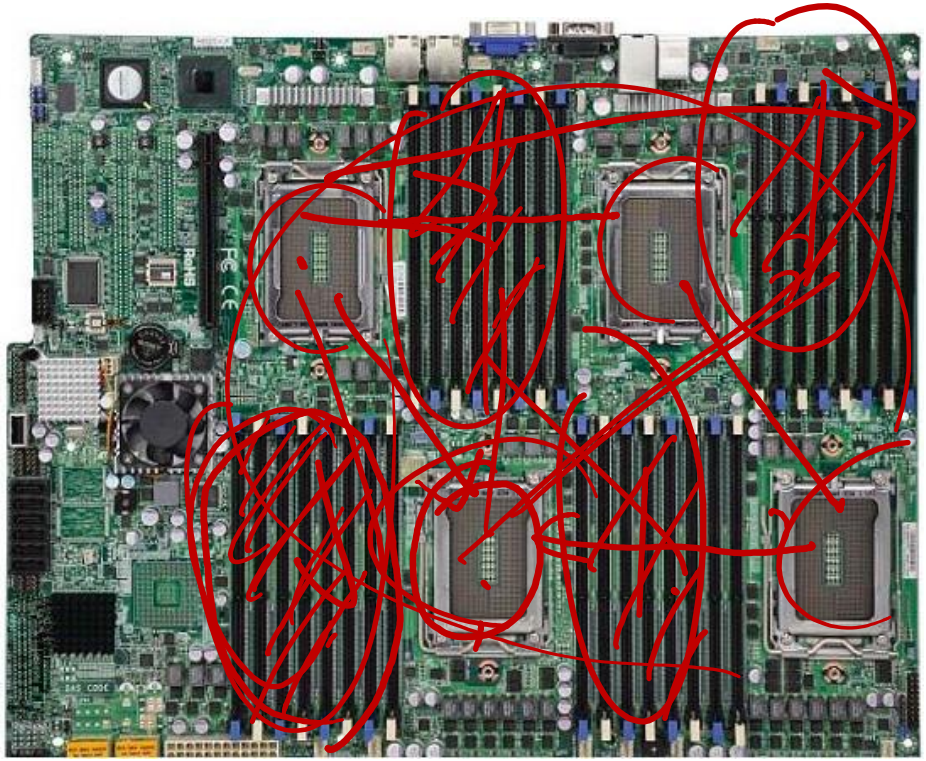
General Context: Multiprocessors

- multiprocessor is any computer with several processors
- SIMD
  - ✓ single instruction, multiple data
    - ○ modern graphics cards
- MIMD
  - ✓ multiple instructions, multiple data

Lemieux cluster, Pittsburgh supercomputing center

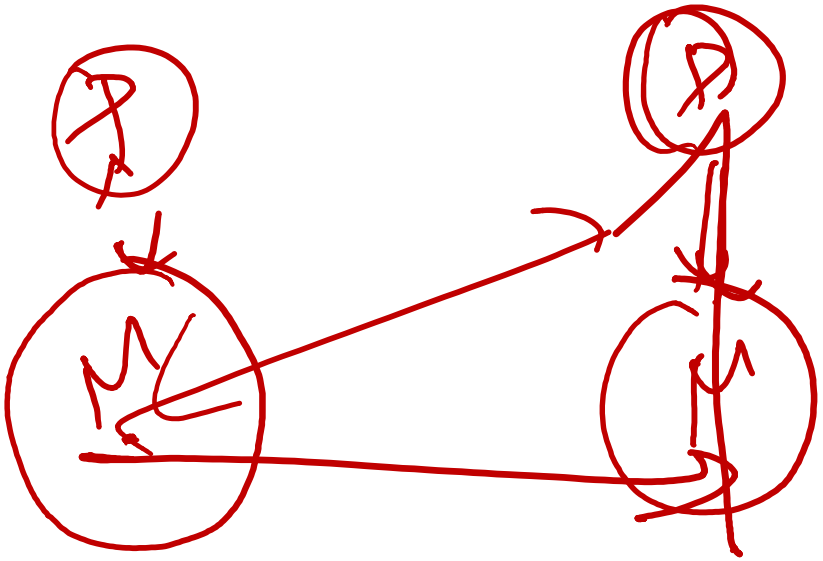
Multiprocessor

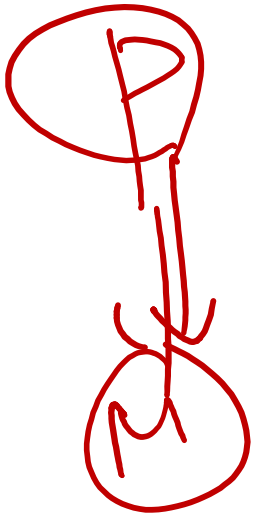
- 2—4 multi-core processors on a motherboard
  - ✓ connected by special short distance interconnect
    - ○ Intel QuickPath



Multiprocessor Memory Types

- shared memory
  - ✓ one (large) common shared memory for all processors
- distributed memory
  - ✓ each processor has its own (small) local memory, and its content is not replicated anywhere else



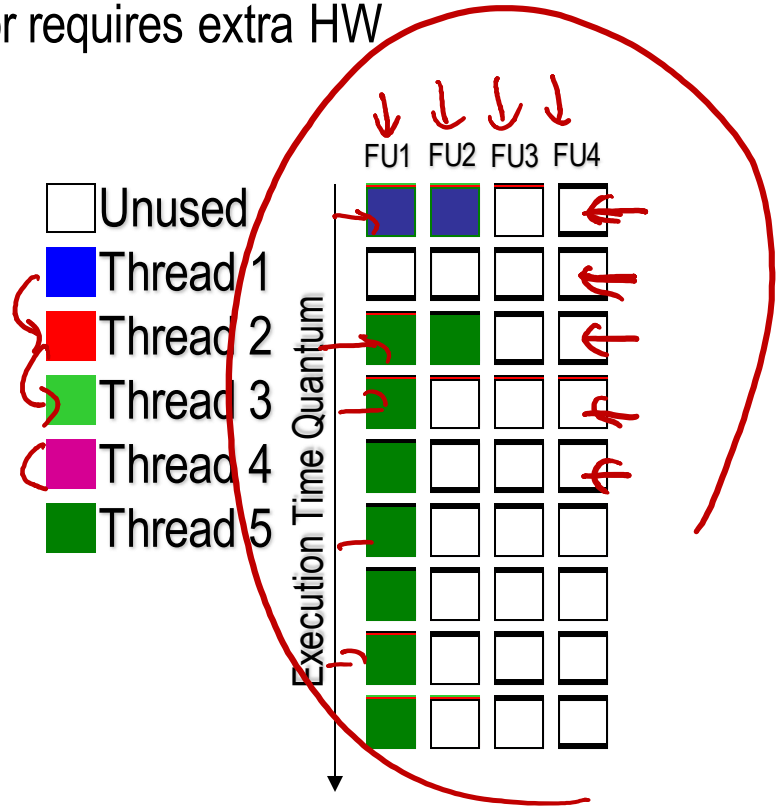


Multiprocessor versus Multi-core Processor

- multi-core processor
  - ✓ a special kind of a multiprocessor w/ all processors on the same chip
- multi-core processors are MIMD
  - ✓ different cores execute different threads (multiple instructions), operating on different parts of memory (multiple data)
- multi-core is a shared memory multiprocessor
  - ✓ all cores share the same memory

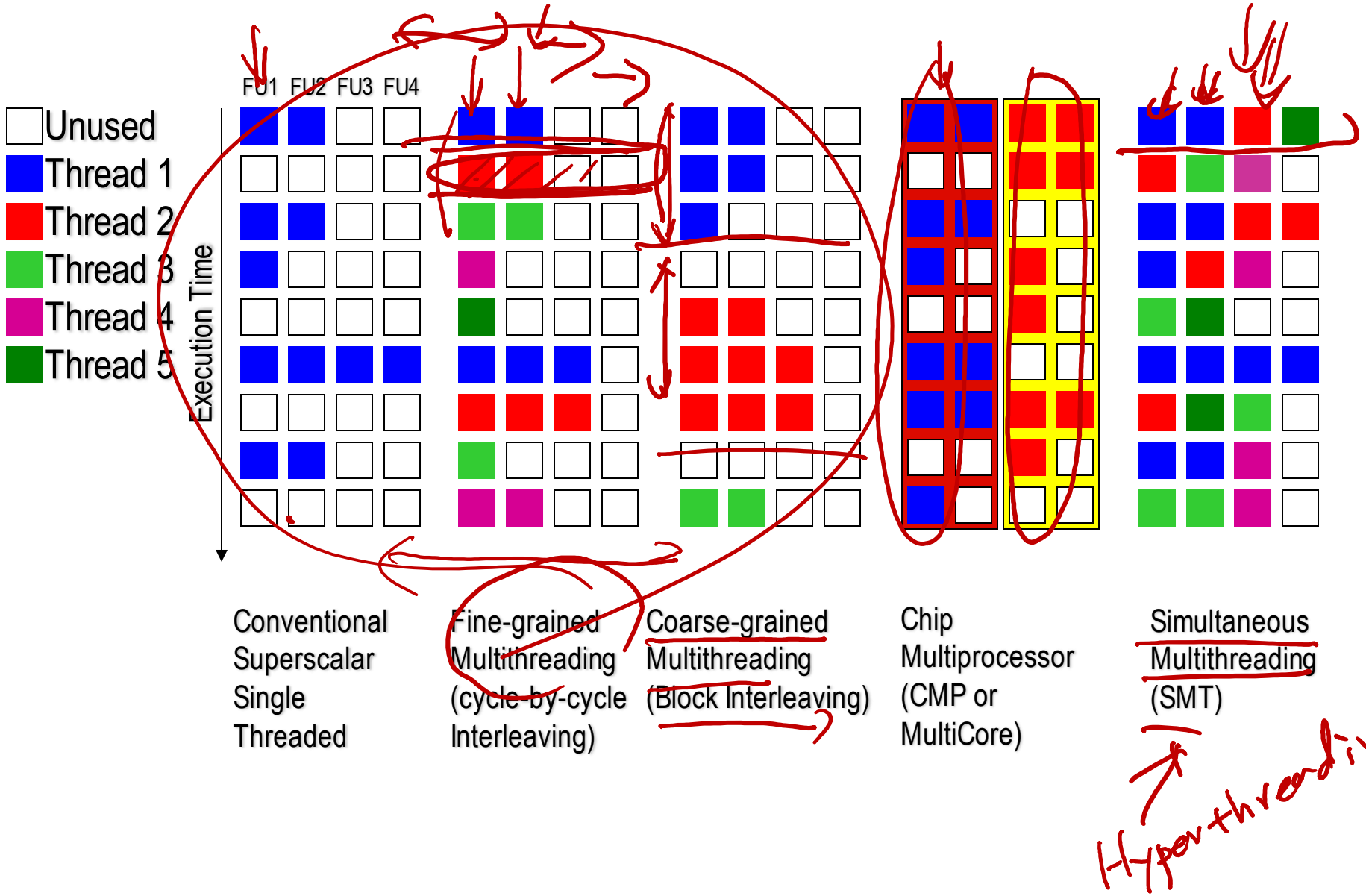
Multi-Tasking Paradigm

- virtual memory makes it easy
  - ✓ context switch could be expensive or requires extra HW



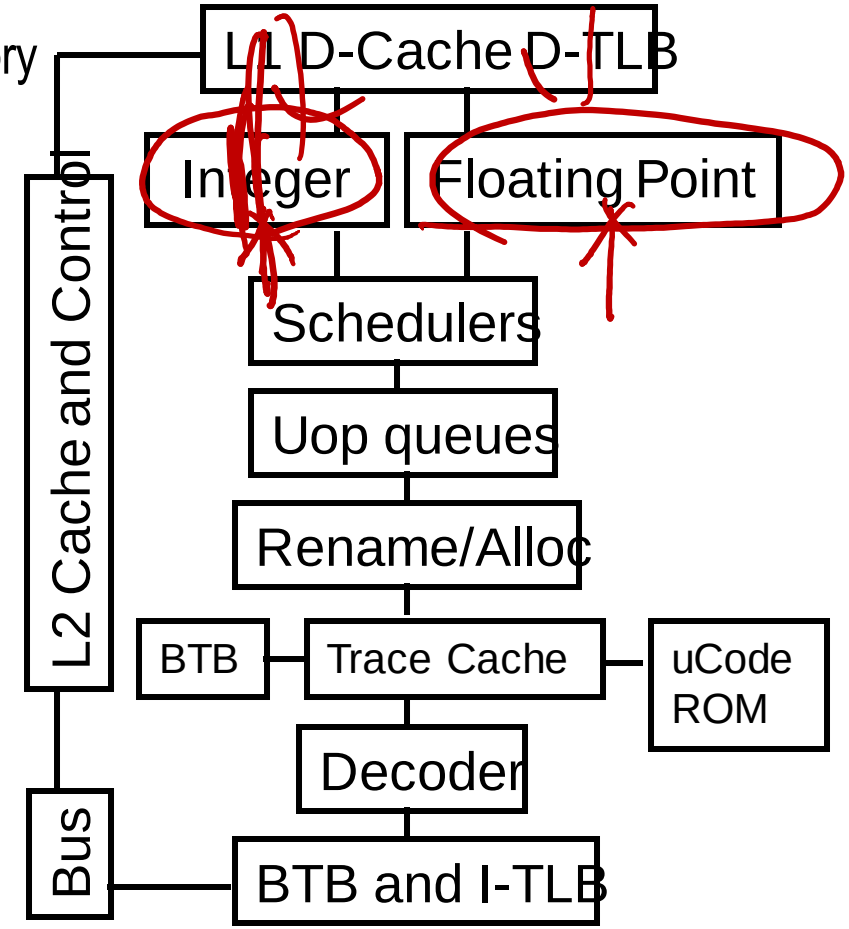
Conventional
Superscalar
Single
Threaded

Multi-Threading Paradigm



Simultaneous Multi-Threading (SMT)

- problem addressed
  - ✓ 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



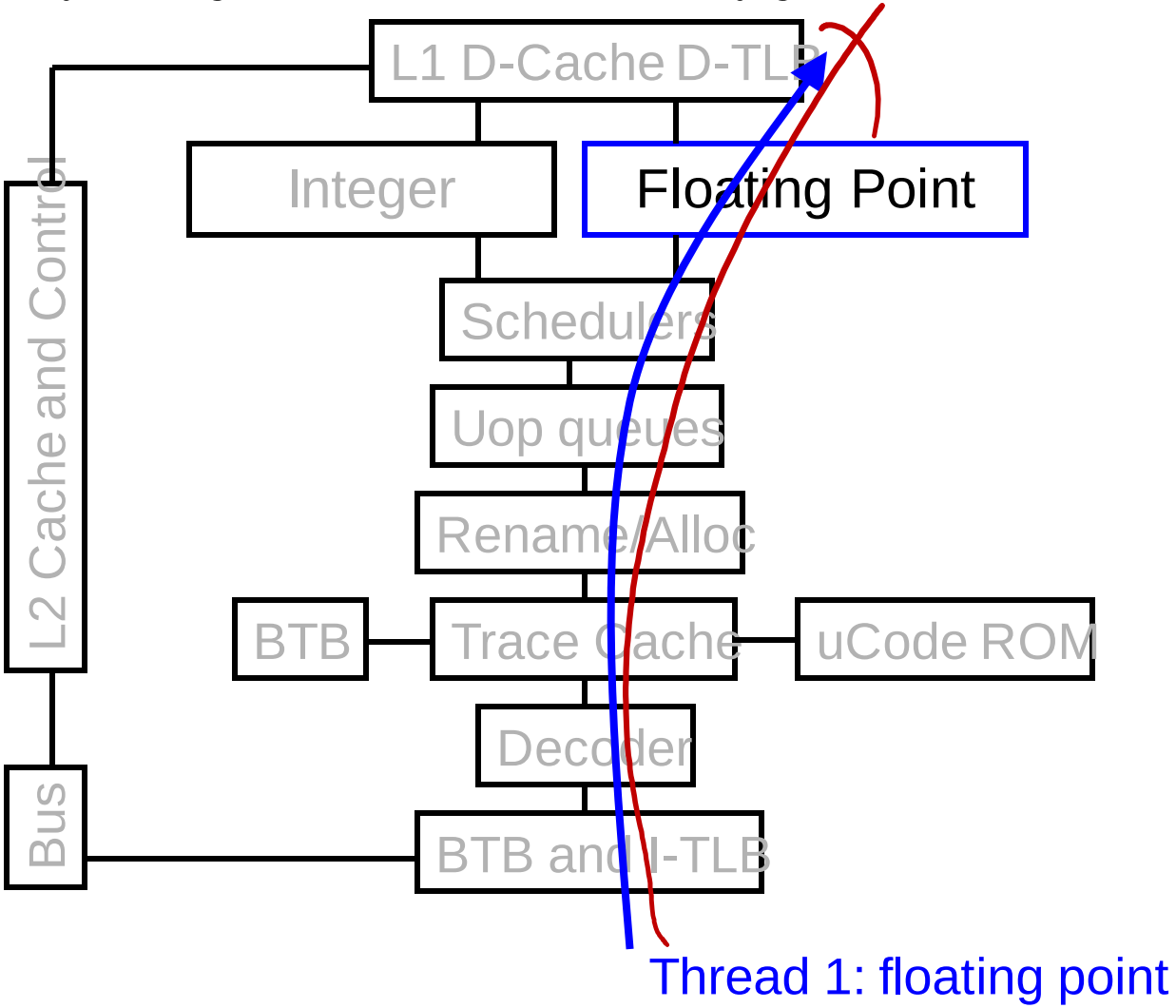
Source: Intel

Simultaneous Multi-Threading (SMT)

- permits multiple independent threads to execute SIMULTANEOUSLY on the SAME core
- weaving together multiple "threads" on the same core
- example: if one thread is waiting for a floating point operation to complete, another thread can use the integer units

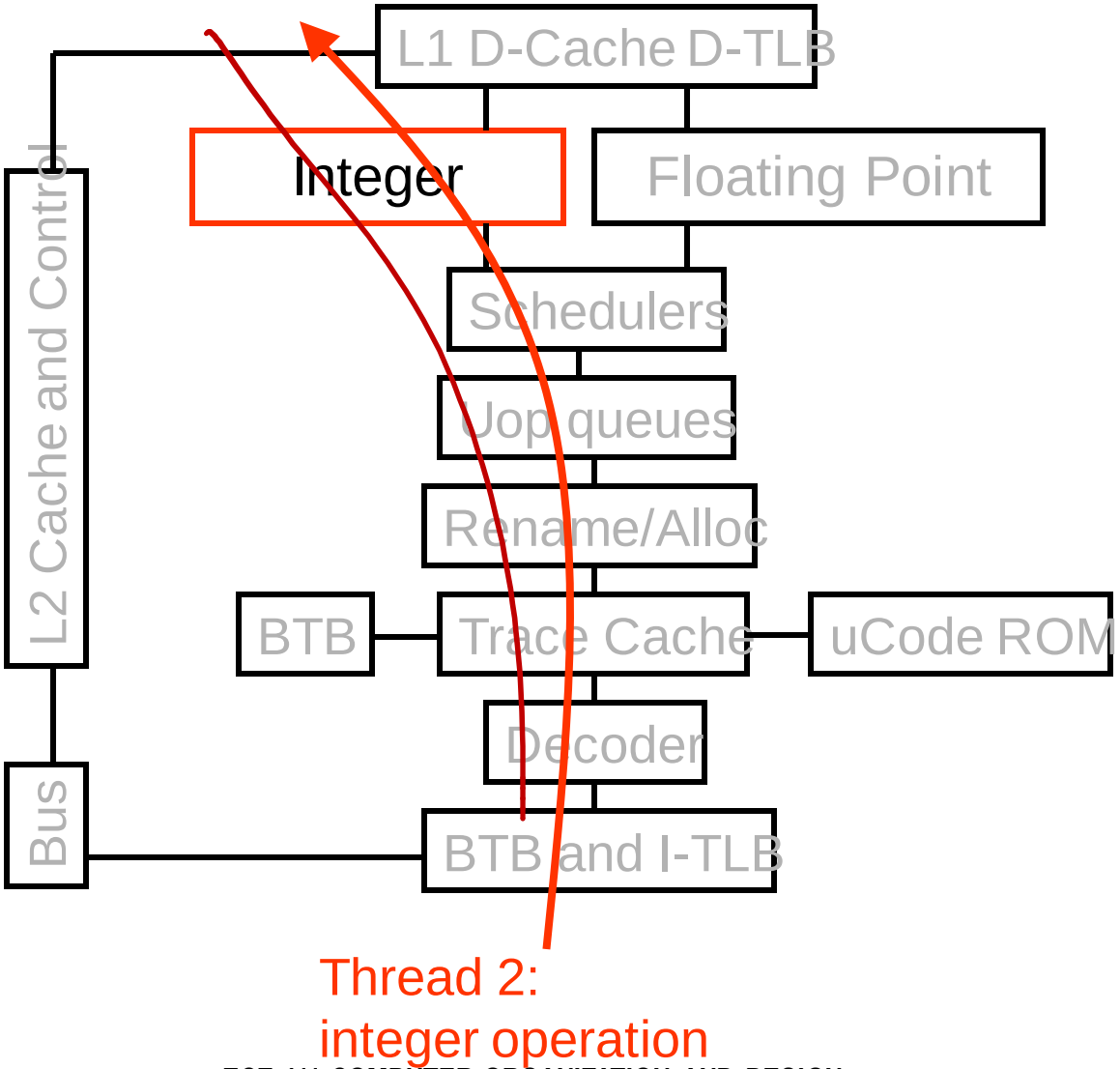
Simultaneous Multi-Threading (SMT)

- w/o SMT, only a single thread can run at any given time



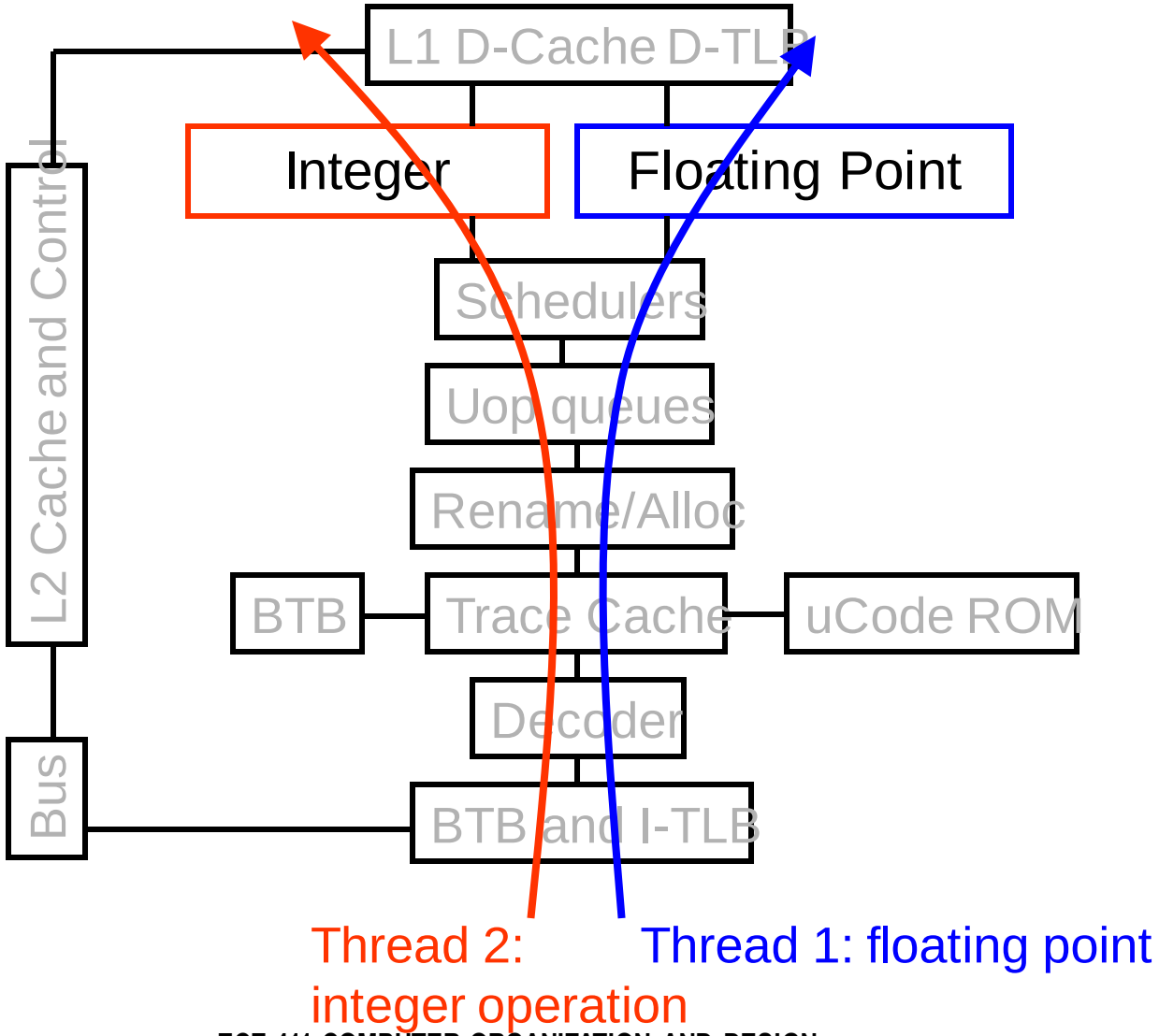
Simultaneous Multi-Threading (SMT)

- w/o SMT, only a single thread can run at any given time



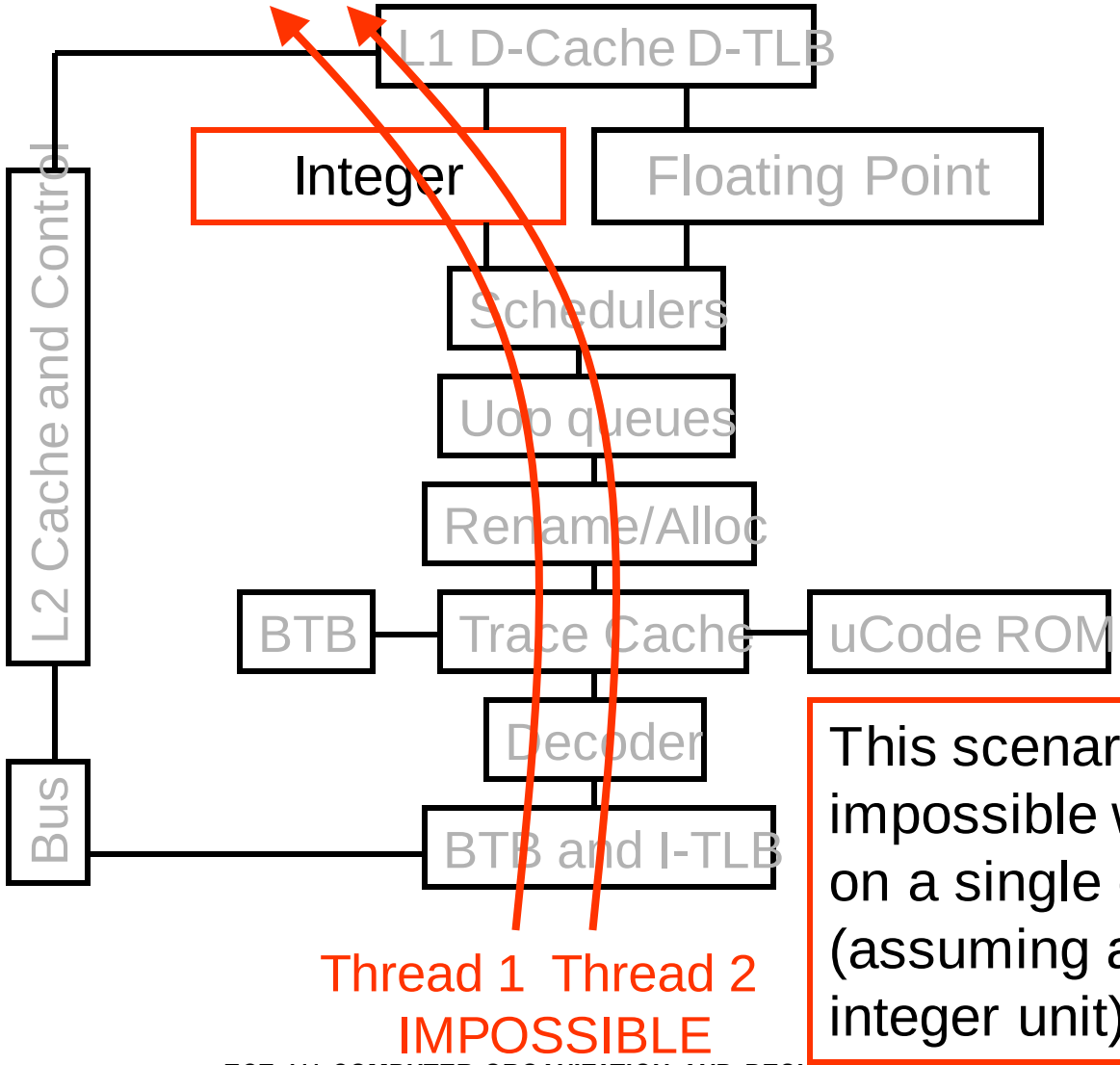
Simultaneous Multi-Threading (SMT)

- SMT processor: both threads can run concurrently



Simultaneous Multi-Threading (SMT)

- But SMT can't simultaneously use the same functional unit



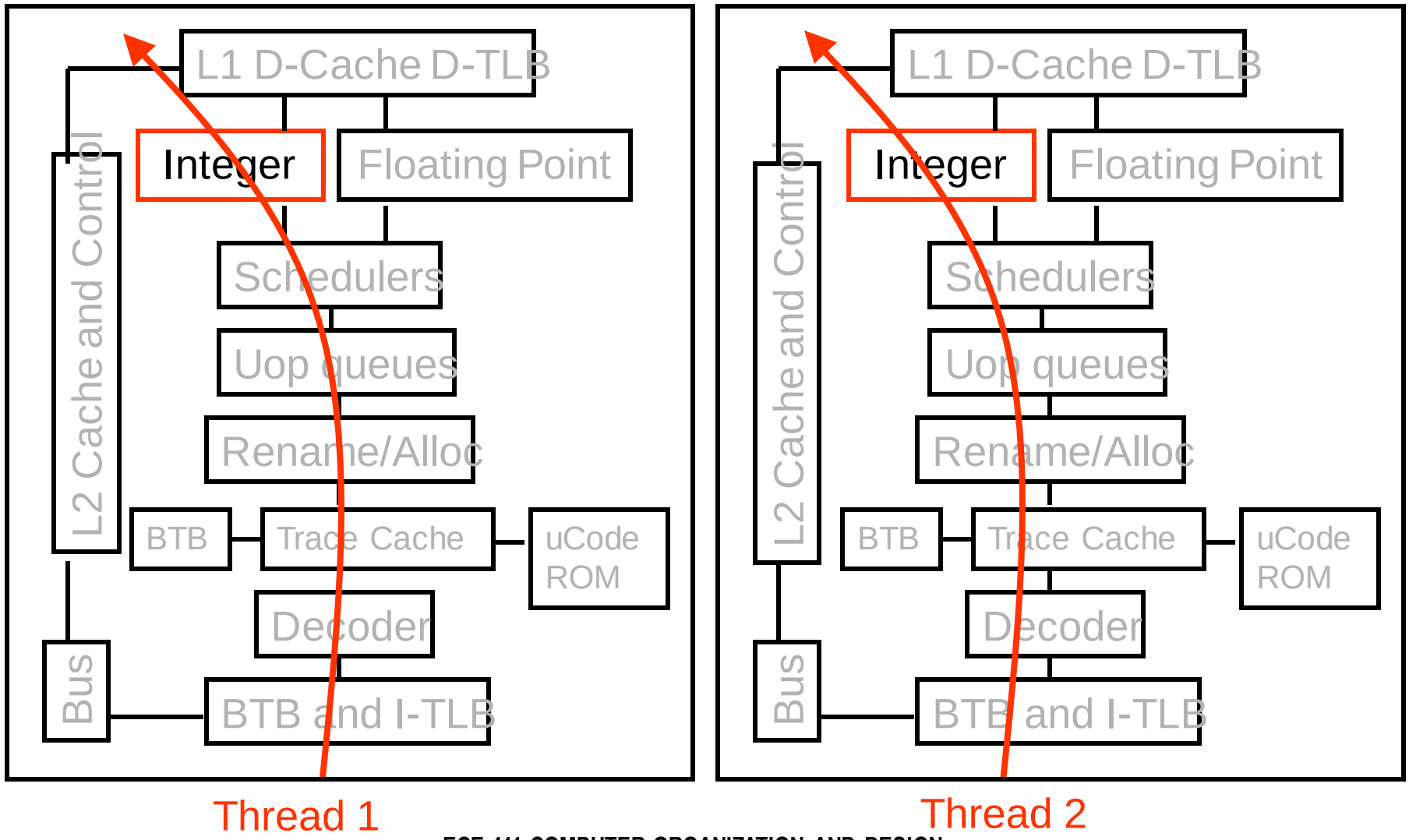
This scenario is impossible with SMT on a single core (assuming a single integer unit)

Simultaneous Multi-Threading (SMT)

- SMT not a "true" parallel processor
  - ✓ enables better threading (e.g. up to 30%)
  - ✓ OS and applications perceive each simultaneous thread as a separate "virtual processor"
  - ✓ the chip has only a single copy of each resource
  - ✓ compare to multi-core, each core has its own copy of resources

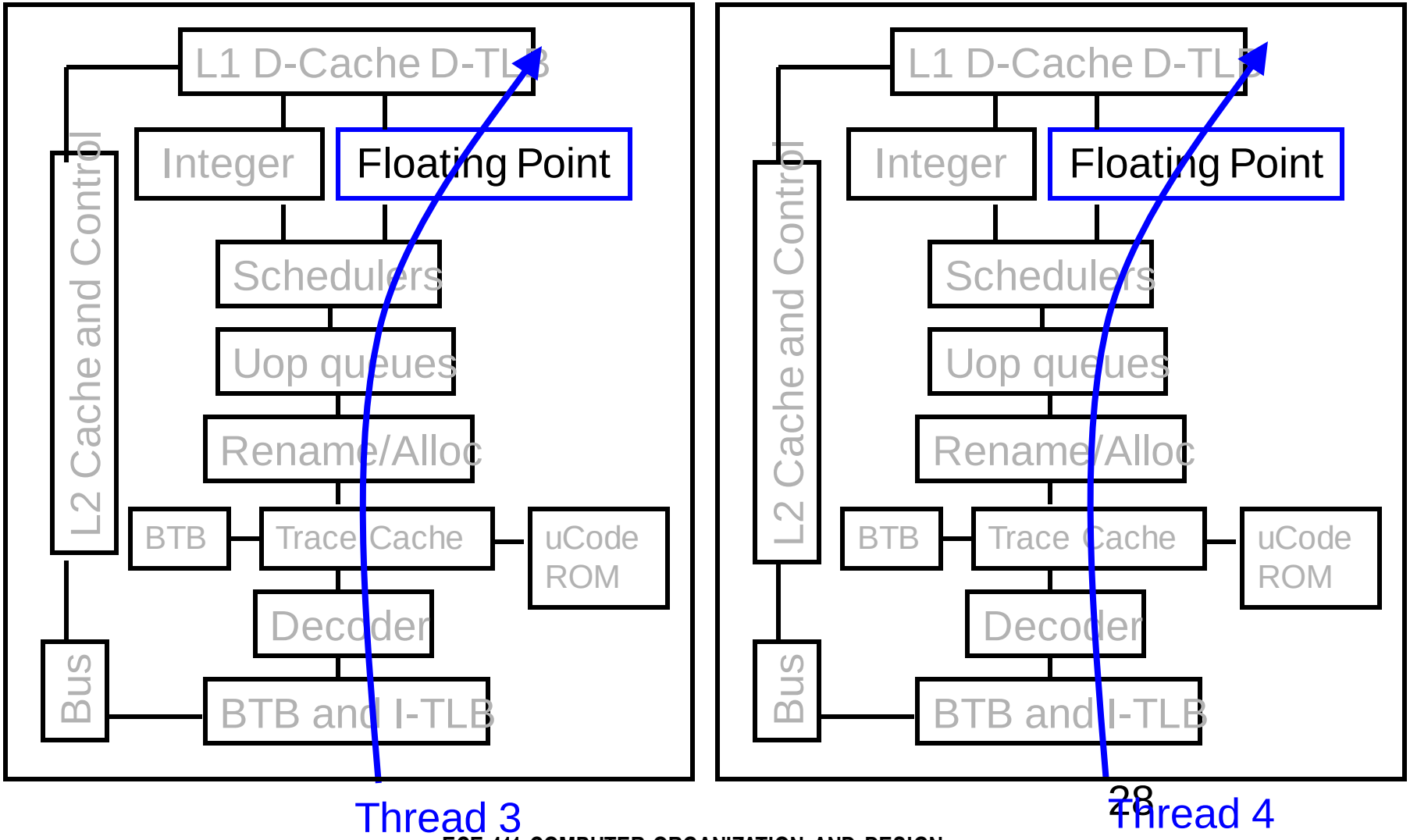
Multi-core

- threads can run on separate cores



Multi-core

- threads can run on separate cores

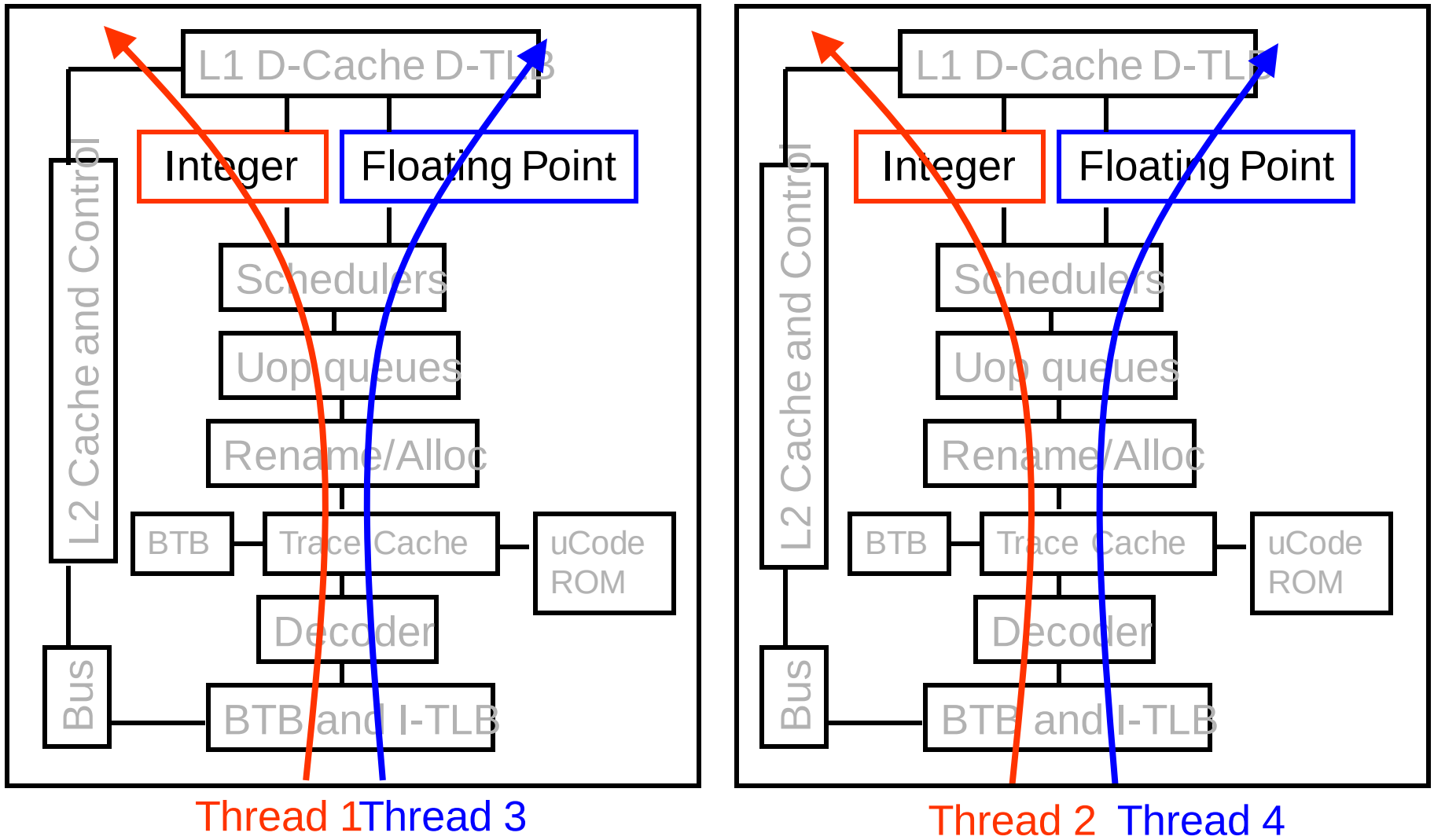


Combining Multi-core and SMT

- cores can be SMT-enabled (or not)
- different combinations:
  - ✓ single-core, non-SMT: standard uniprocessor
  - ✓ single-core, w/ SMT
  - ✓ multi-core, non-SMT
  - ✓ multi-core, w/ SMT: our fish machines
- number of SMT threads:
  - ✓ 2, 4, or sometimes 8 simultaneous threads
- Intel calls them "hyper-threads"

SMT Dual-core

- all four threads can run concurrently



Comparison: multi-core vs SMT

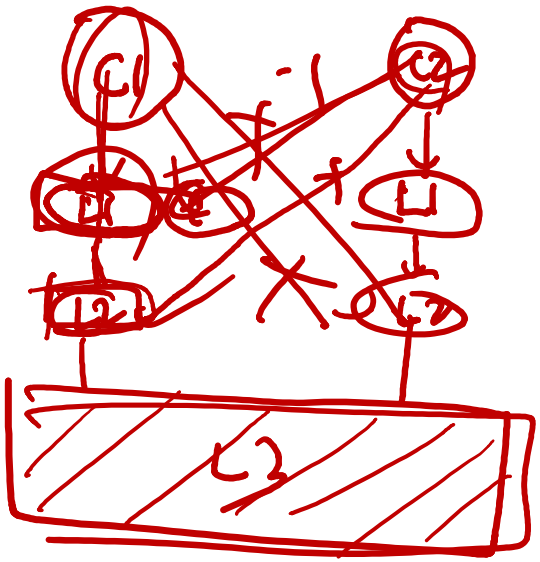
- advantages/disadvantages?

Comparison: multi-core vs SMT

- multi-core:
  - ✓ since there are several cores,
    each is smaller and not as powerful
    (but also easier to design and manufacture)
  - ✓ however, great with thread-level parallelism
- SMT
  - ✓ can have one large and fast superscalar core
  - ✓ great performance on a single thread
  - ✓ mostly still only exploits instruction-level parallelism

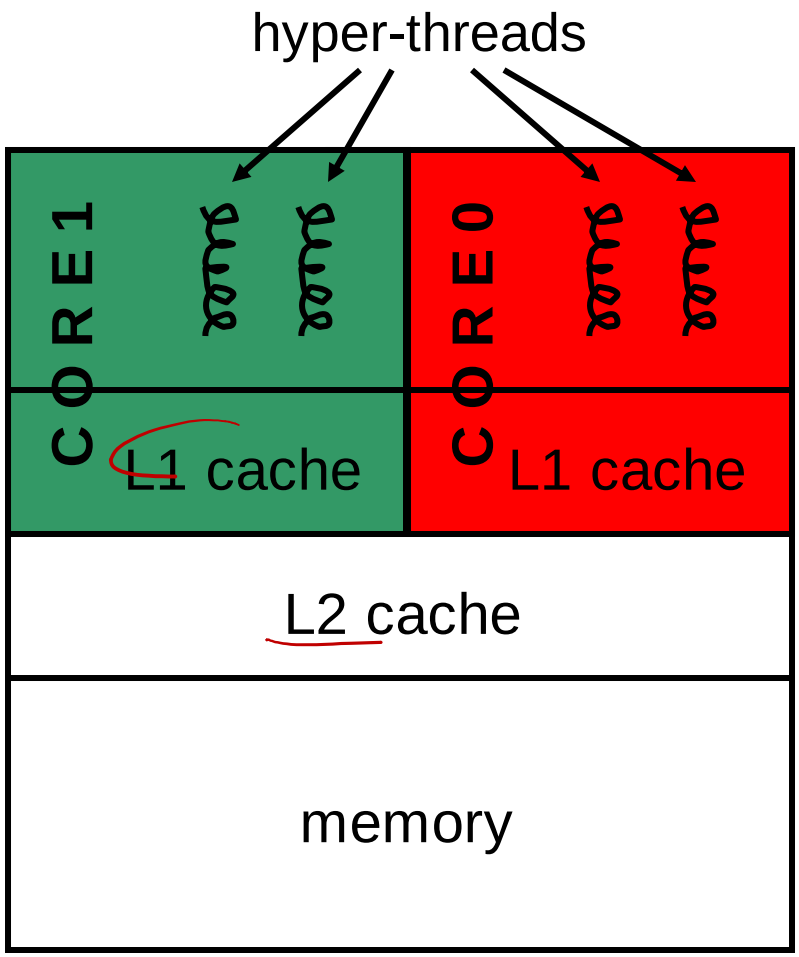
Memory Hierarchy

- If simultaneous multithreading only:
  - ✓ all caches shared
- multi-core chips:
  - ✓ L1 caches private
  - ✓ L2 caches private in some architectures and shared in others
- memory is always shared

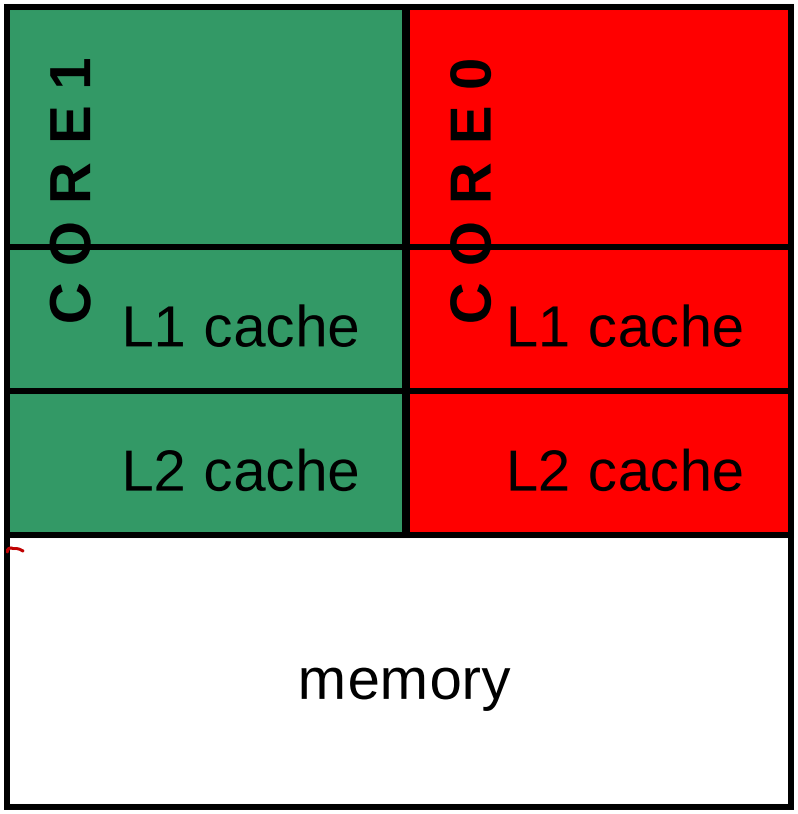


Dual-core Intel Xeon processors

- each core
  - ✓ hyper-threaded
- private L1 caches
- shared L2 caches

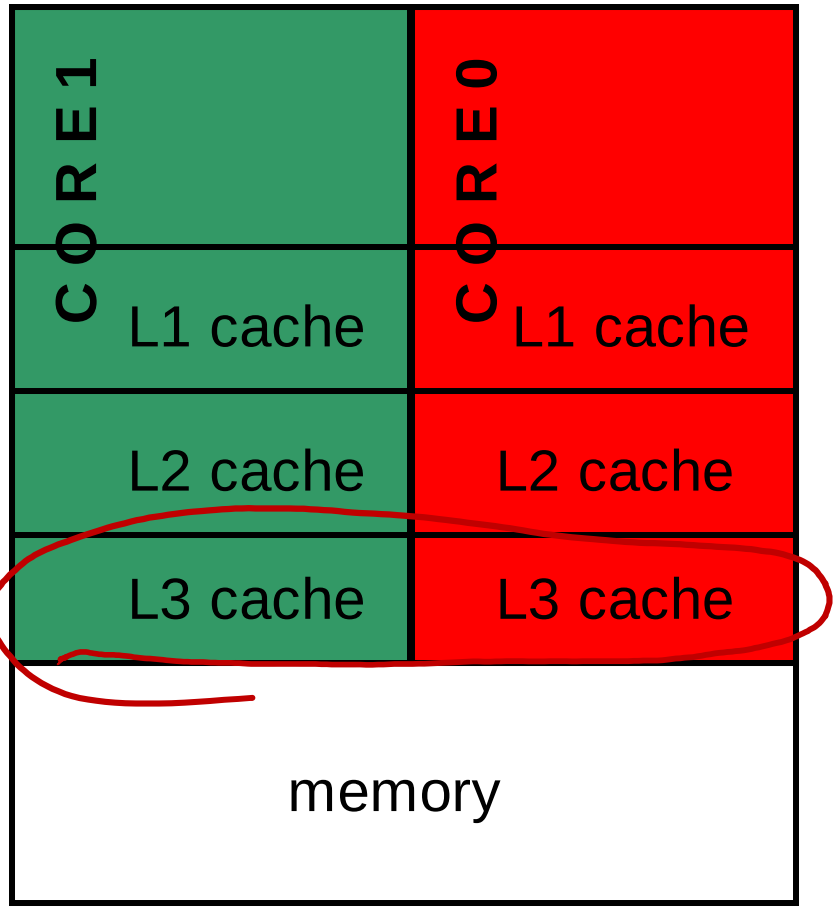


Designs w/ private L2 caches



both l1 and l2 are private

examples: amd opteron,
amd athlon, intel pentium d



a design with l3 caches

example: intel itanium 2

Private vs Shared Caches?

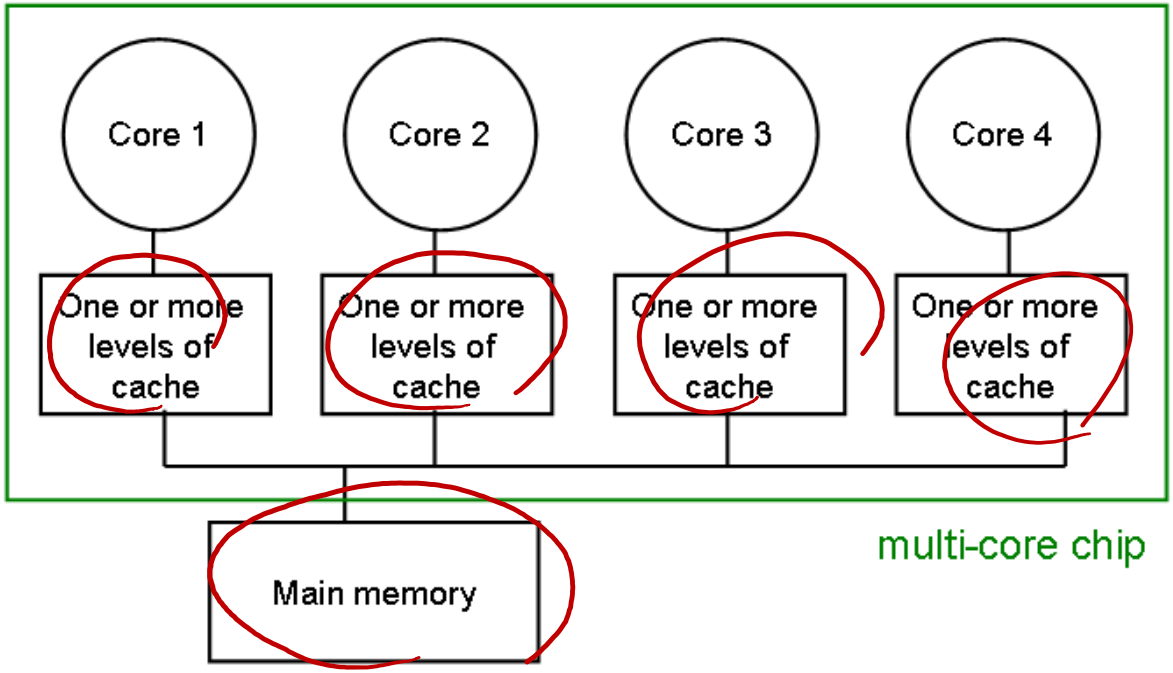
- advantages/disadvantages?

Private vs shared caches

- advantages of private:
  - ✓ they are closer to core, so faster access
  - ✓ reduces contention
- advantages of shared:
  - ✓ threads on different cores can share the same cache data
  - ✓ more cache space available if a single (or a few) high-performance thread runs on the system

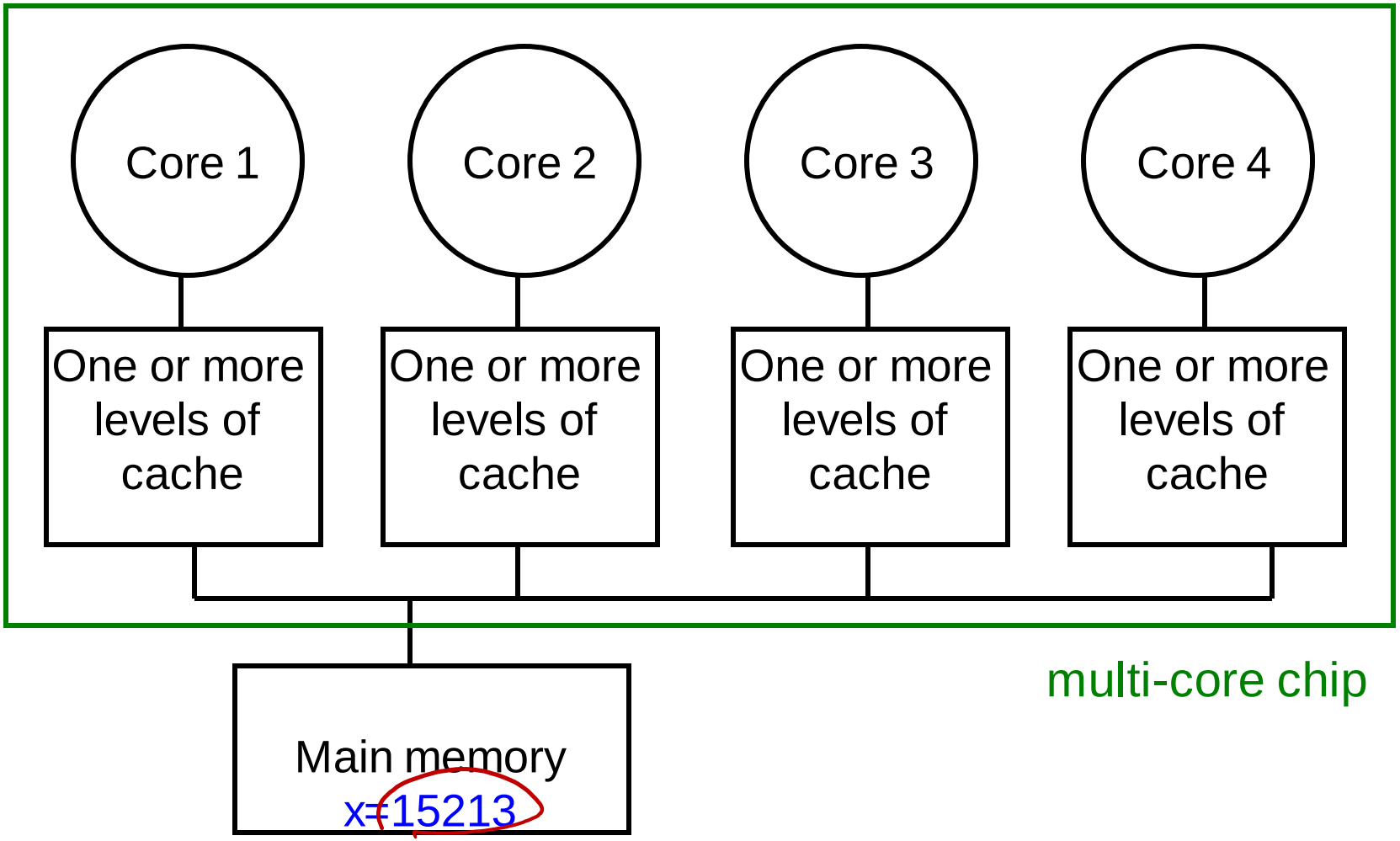
Cache Coherence Problem

- 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



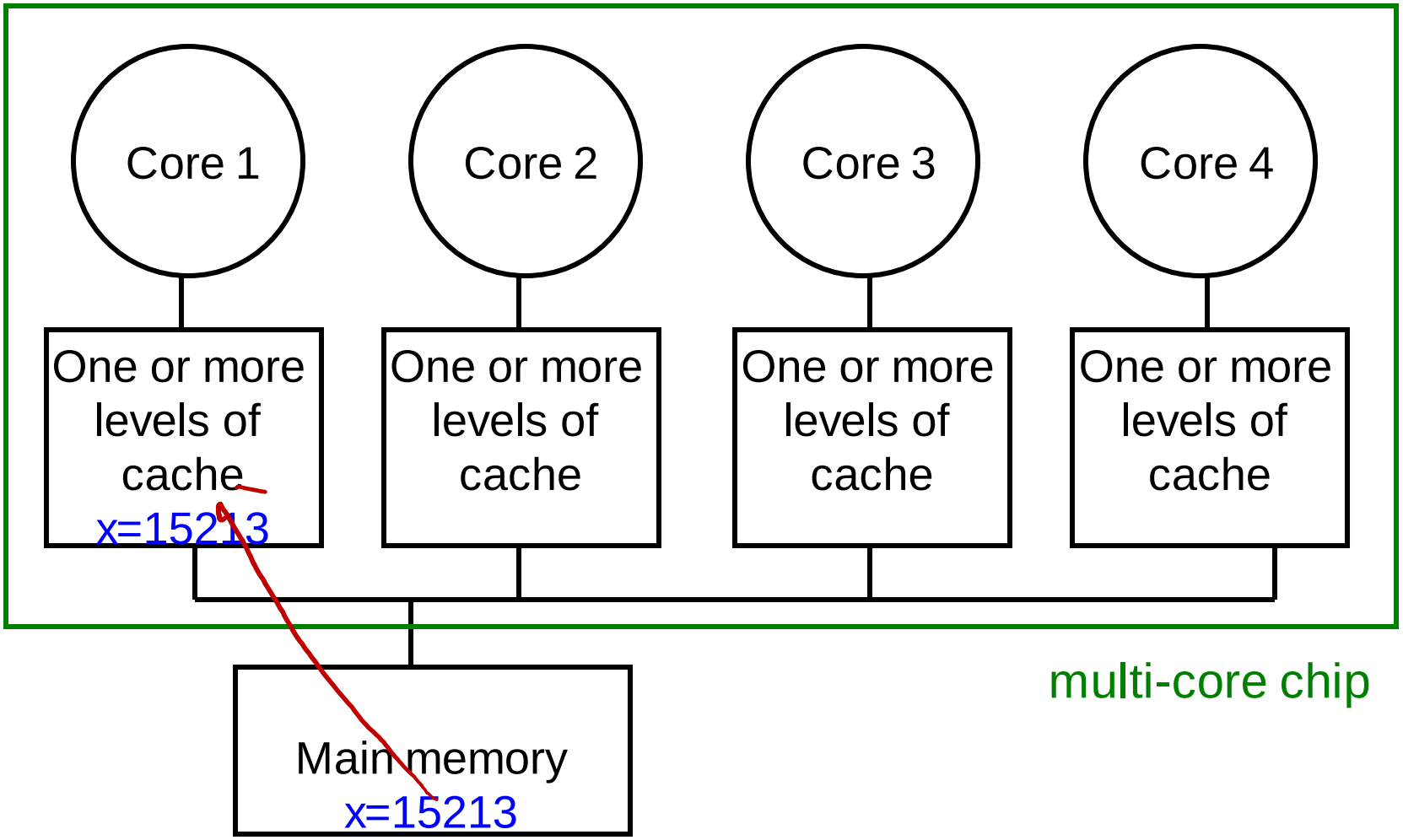
Cache Coherence Problem

- suppose variable x initially contains 15213



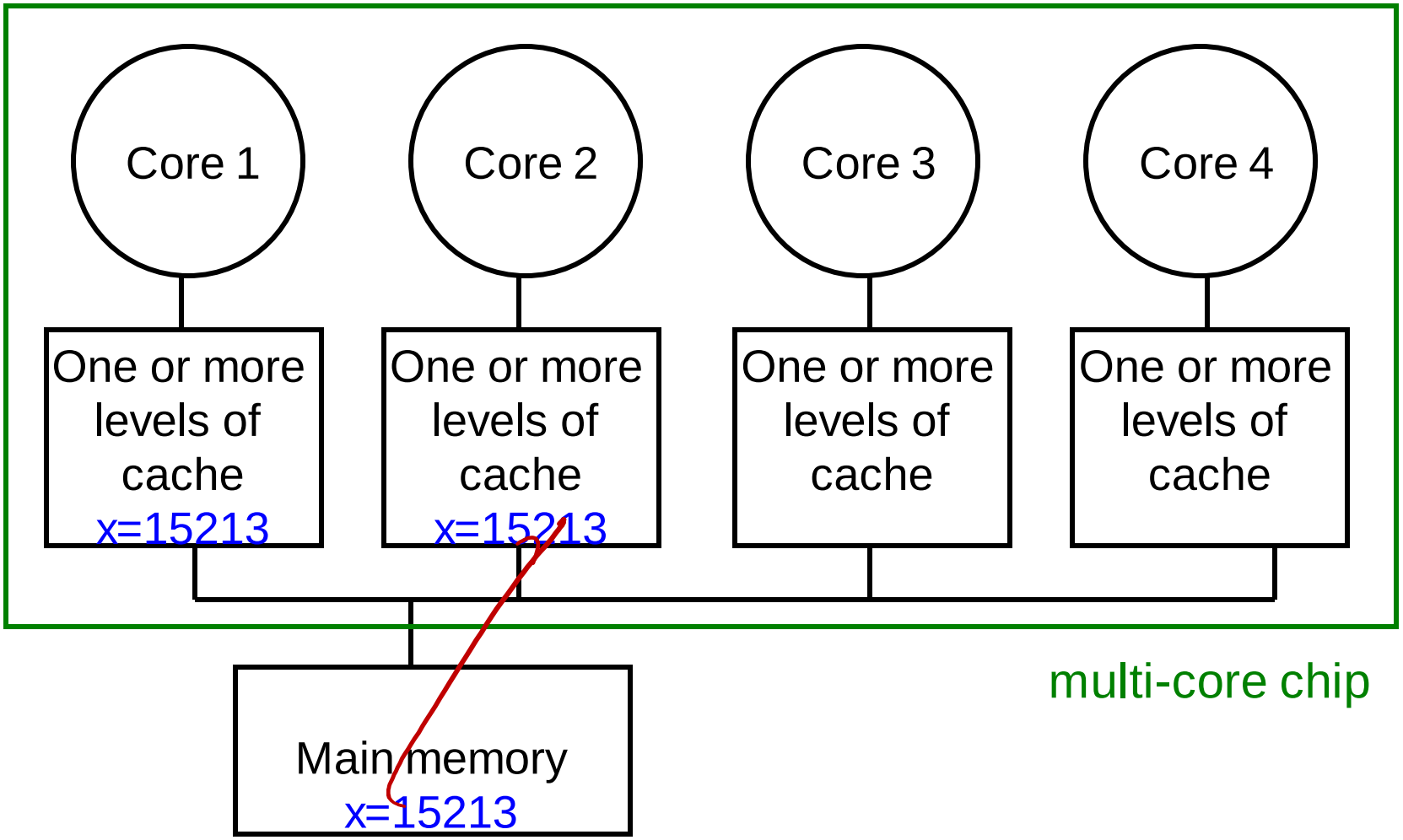
Cache Coherence Problem

- core 1 reads x



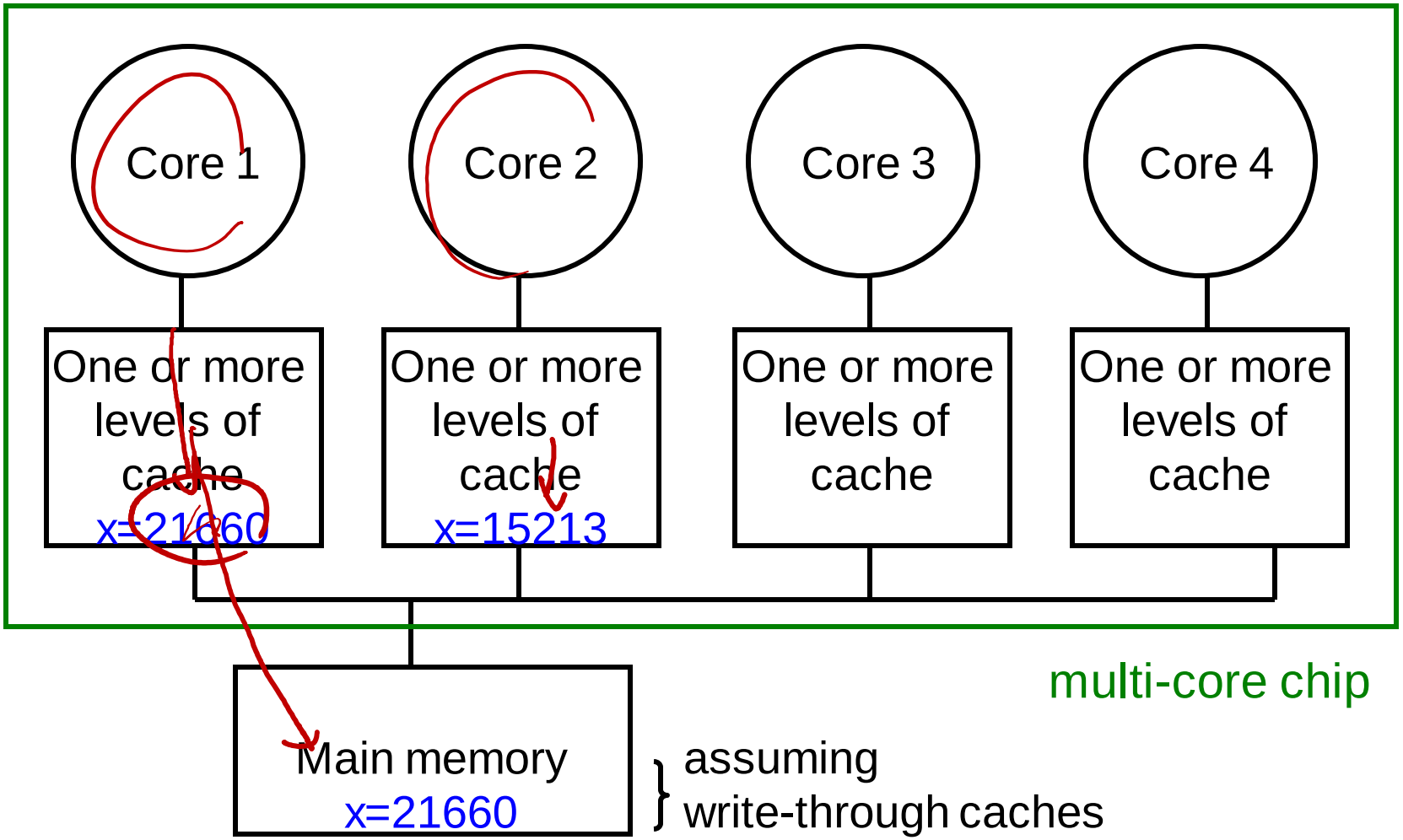
Cache Coherence Problem

- core 2 reads x



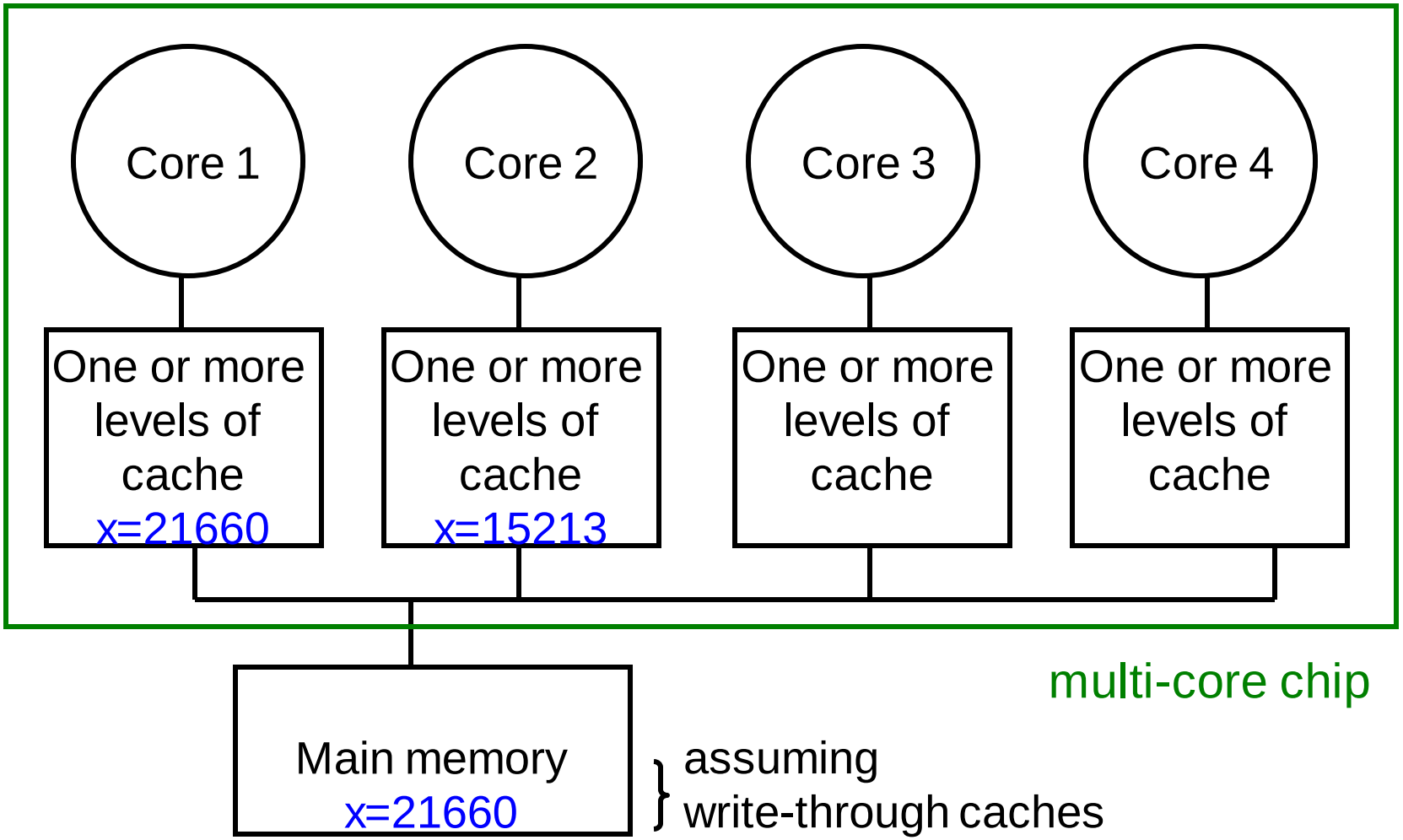
Cache Coherence Problem

- core 1 writes to x, setting it to 21660



Cache Coherence Problem

- core 2 attempts to read x… gets a stale copy



Solutions for Cache Coherence Problem

- general problem w/ multiprocessors, not limited just to multi-core
- many solution algorithms, coherence protocols, etc.
  - ✓ a simple solution: invalidation-based protocol with snooping
  - ✓ MSI, MESI (Modified, Exclusive, Shared, Invalid)

Announcement

- next lecture: continuing multi-core and multi-threading
  - ✓ Ch. 6.4 – 6.5 (HP1)
- MP assignment
  - ✓ MP3 checkpoint 4 due on 4/15 5pm