Lecture 07
Multicore Computation

Lecture based on notes from

John Mellor-Crummey
Department of Computer Science
Rice University & Jernej Barbic

Microprocessor Architecture (Mid 90's)

- Superscalar (SS) designs were the state of the art
  - —multiple instruction issue
  - —dynamic scheduling: h/w tracks dependencies between instr.
  - —speculative execution: look past predicted branches
  - —non-blocking caches: multiple outstanding memory ops
- Circuit density continues to double every 18 months
  - —provides raw material for more logic
  - —enables higher clock frequencies
- Apparent path to higher performance?
  - —wider instruction issue
  - —support for more speculation

This was thinking mid-90s.

"Flies in the Ointment"

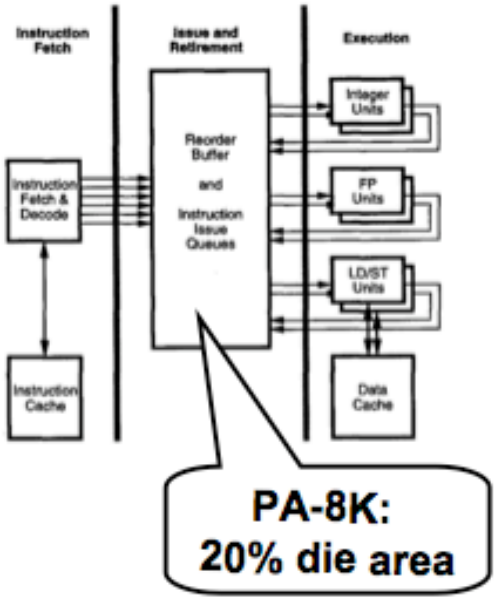
Claim: ⇑ issue width of SS will provide diminishing returns

Two factors

- **Fundamental circuit limitations**
- **Limited amount of instruction-level parallelism**

Superscalar Designs (e.g. R10K, PA-8K)

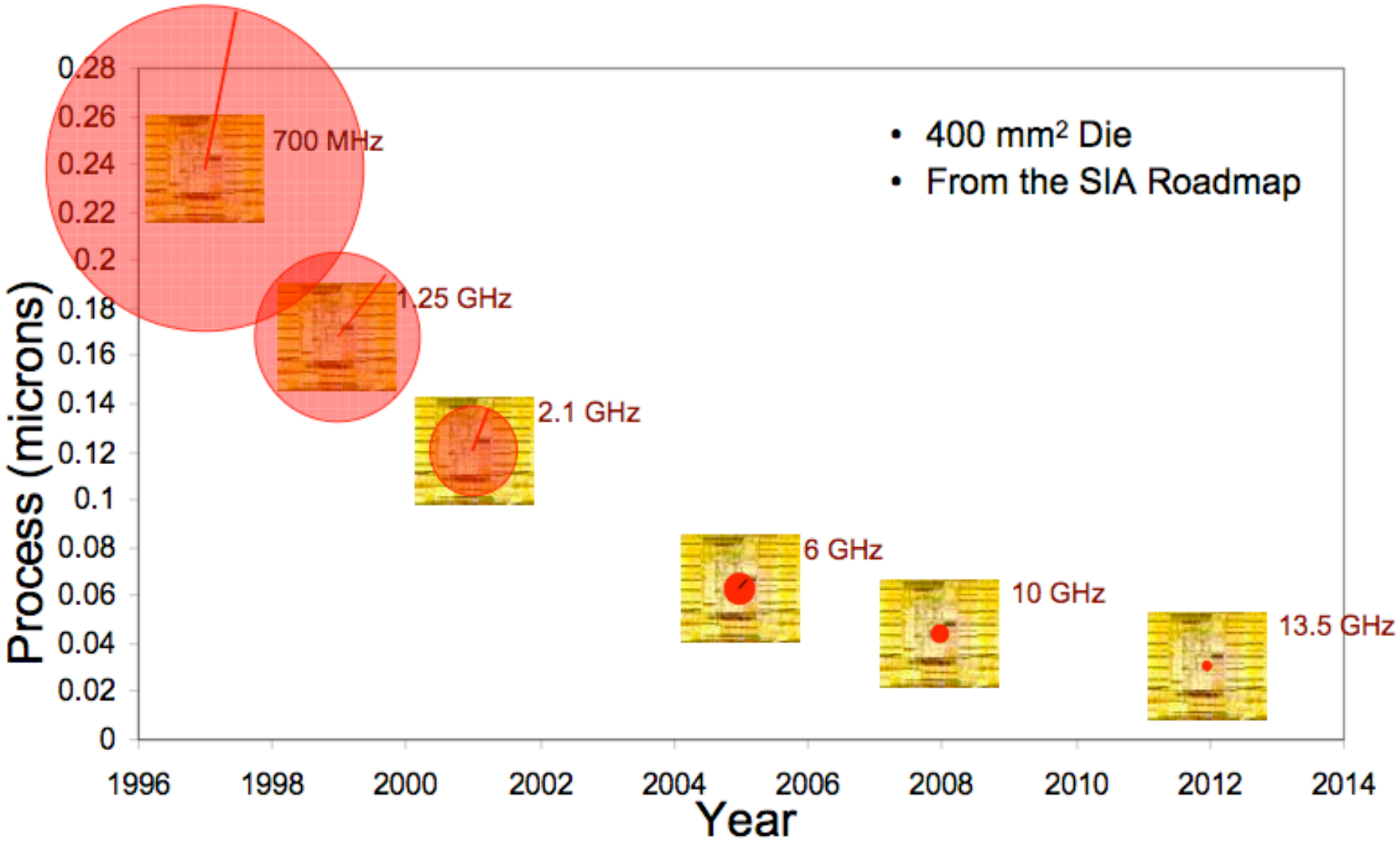
- 3 phases
  - —instruction fetch
  - —issue and retirement
  - —execution
- Circuit-level performance limiters?
  - —issue and retirement
    - – issue instruction when all operands ready
    - – register renaming: avoid artificial dependences
      - mapping table: (operands * issue width) ports
      - reorder buffer: # 1-bit comparators =
        - operands * issue width * $\log_2$(# reg) * issue Q length
    - – wider issue widths increase need for deeper issue Q's for sufficient ||-ism
      - claims:
        - quadratic increase in size of issue Q
        - long wires for bcast tags of issued instr. ⇒ ultimately limit cycle time
  - —execution phase: quadratic ⇑ register file, quadratic ⇑ bypass logic
    - – Farkas et al '96: 8-issue only 20% > 4-issue (reg file complexity limits perf))



Circuit complexity and interconnect delay limit practicality of support structures for larger issue width

The case for a single-chip multiprocessor, Olukotun et al., ASPLOS-VII, 1996.

Range of a Wire in One Clock Cycle

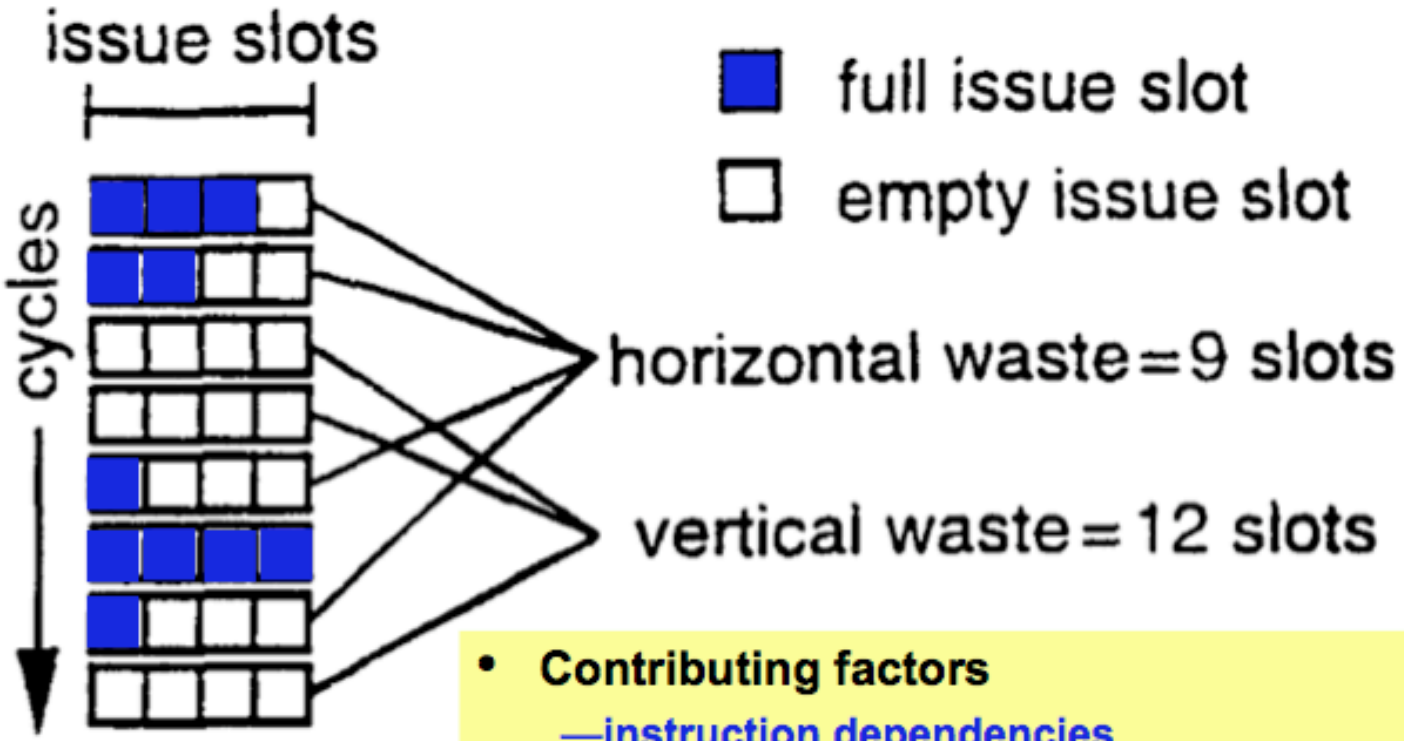


Circuit Technology Impact Summary

- Delays ⇑ as issue queues ⇑ and multi-port register files ⇑
- Increasing delays limit performance returns from wider issue

Instruction-level Parallelism Concerns

Issue Waste



- **Contributing factors**
  - —instruction dependencies
  - —long-latency operations within a thread

How do they contribute to waste?

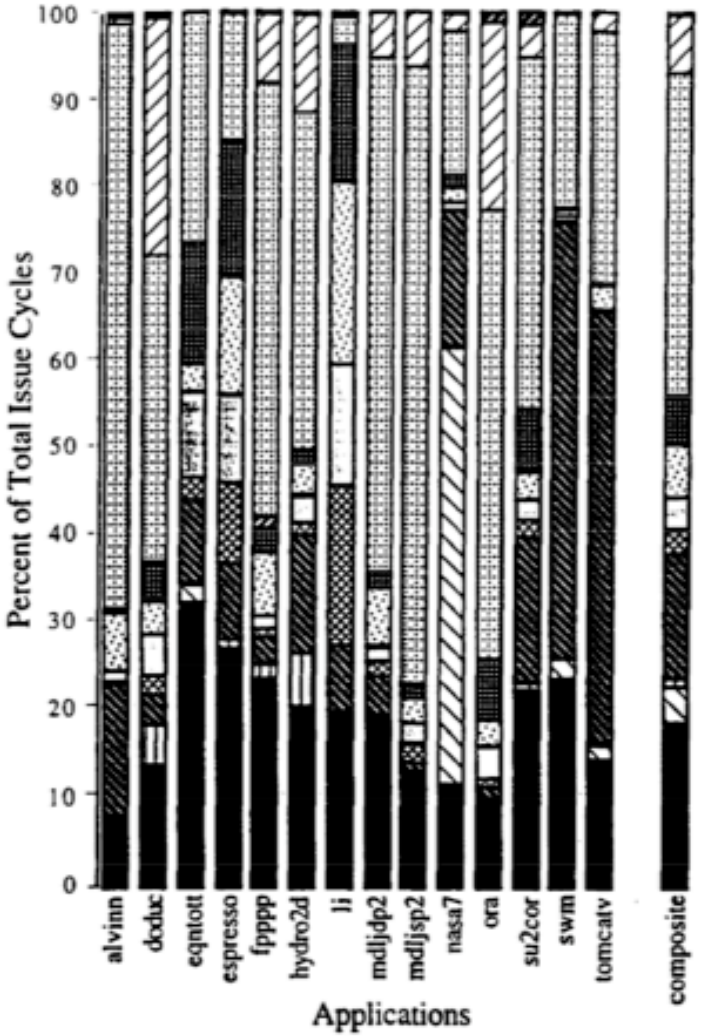
Sources of Wasted Issue Slots

- TLB miss
  - —larger TLB; h/w inst prefetching; h/w or s/w data prefetch
- I cache miss
  - —larger icache, more icache associativity; h/w prefetch
- D cache miss
  - —larger, more associative, prefetching, more dynamic execution
- Control hazard
  - —speculative execution; aggressive if-conversion
- Branch misprediction
  - —better prediction logic; lower mispredict penalty
- Load delays (L1 hits)
  - —shorten load latency; better scheduling
- Instruction delays
  - —better scheduling
- Memory conflict (multipleaccess to same location in a cycle)
  - —improved scheduling

How Much IPC is There?

- Approach: study applications and evaluate their characteristics
  - —assess quantity and character of parallelism present
- Are there any pitfalls to this approach?
- Is there any other approach?

Simulations of 8-issue Superscalar



Simultaneous multithreading: maximizing on-chip parallelism, Tullsen et. al. ISCA, 1995.

Applications: most of SPEC92

- On average < 1.5 IPC (19%)
- Dominant waste differs by appl.
- Short FP dependences: 37%
- 6 other causes ≥ 4.5%

Analysis of 8-issue Simulations

- No dominant cause of wasted cycles
- No dominant solution
  - —no single latency-tolerance technique likely to help dramatically
- Even if memory latencies eliminated, utilization < 40%
- Tullesen et. al. claim
  - —"instruction scheduling targets several important segments of the wasted issue bandwidth, but we expect that our compiler has already achieved most of the available gains in that regard"
- If specific latency hiding mechanisms limited
  - —need general latency hiding solution for increases in parallelism

Some important points

- Technology alone is not driving push to multi-core
  - What was state of the art - more issue, superscalar - provides diminishing performance returns b/c of program properties
- Still, performance gains possible with scaling

$$\frac{Instructions}{Program} \times \frac{Clock\ cycles}{Instruction} \times \frac{Seconds}{Clock\ Cycle} = \frac{Seconds}{Program} = CPU\ time$$

- If CCs/instruction performance gains tapped out + scaling performance inhibited (b/c of lower V_{dd}, lower clock rates), where does performance come from?

$$\frac{Instructions}{Program} \times \frac{Clock\ cycles}{Instruction} \times \frac{Seconds}{Clock\ Cycle} = \frac{Seconds}{Program} = CPU\ time$$

Some important points

- Performance must come from combination of parallelism + previously ignored HW optimizations
  - E.g. instead of getting 2x from technology, get 10% from A, 5% from B, etc.

What Should be Next?

- If not ⇑ superscalar issue width, then what?
- Alternatives
  - —single chip multiprocessor
  - —simultaneous multithreaded processor
- How should we decide?
- Best approach depends upon application characteristics

The Case for a Single-chip Multiprocessor

The Case for a Single Chip Multiprocessor

Two motivations

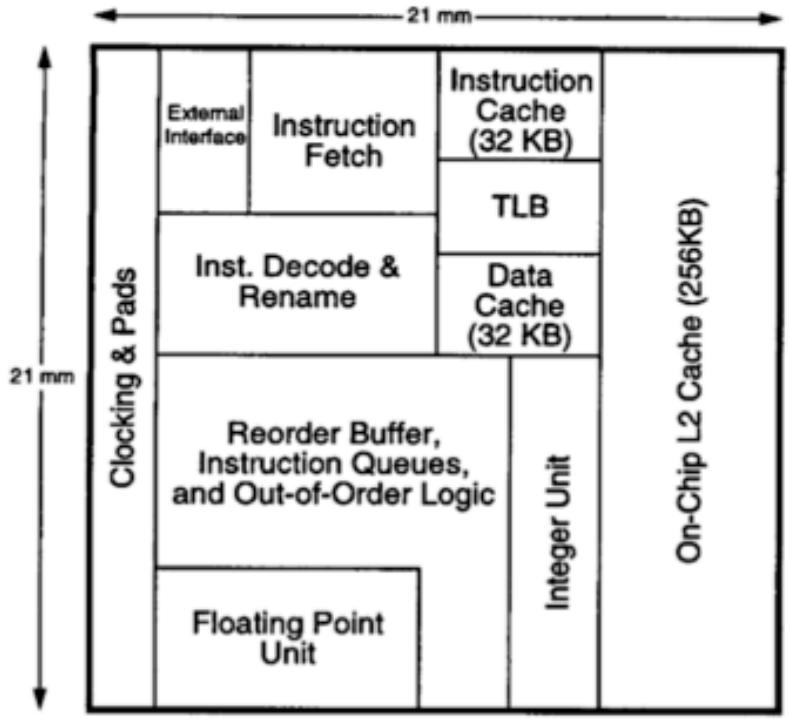
- "Technology push"
  - —delays ⇑ as issue queues lengthen and multi-port register files grow
  - —increasing delays limit performance returns from wider issue
- "Application pull"
  - —limited IPC is a problem

Comparing Alternative Designs

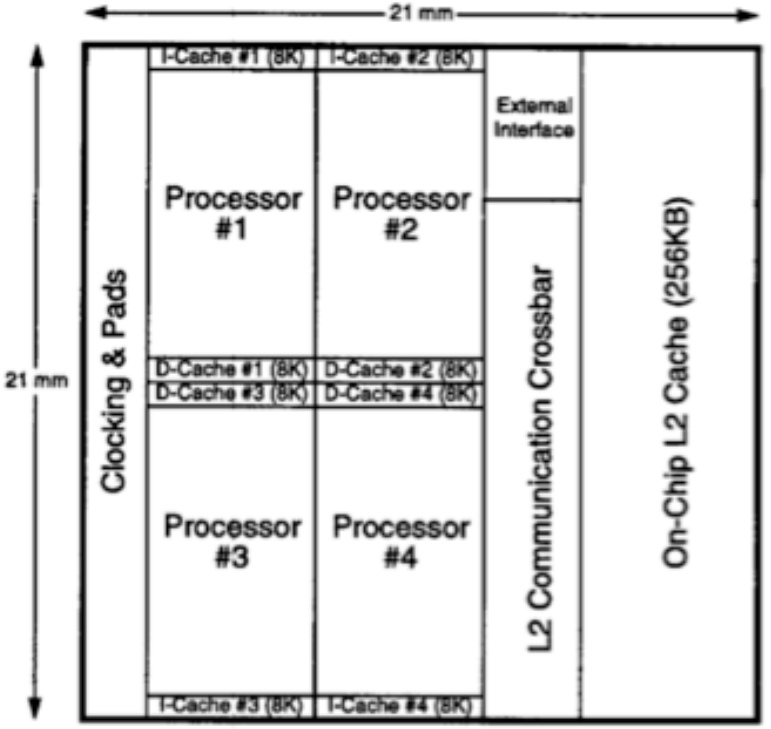
- Consider two microarchitectures
  - —6-way superscalar architecture
  - —4 x 2-way superscalar multiprocessor architecture

Floorplans: 6-issue SS vs. 4 x 2 CMP

6-issue superscalar



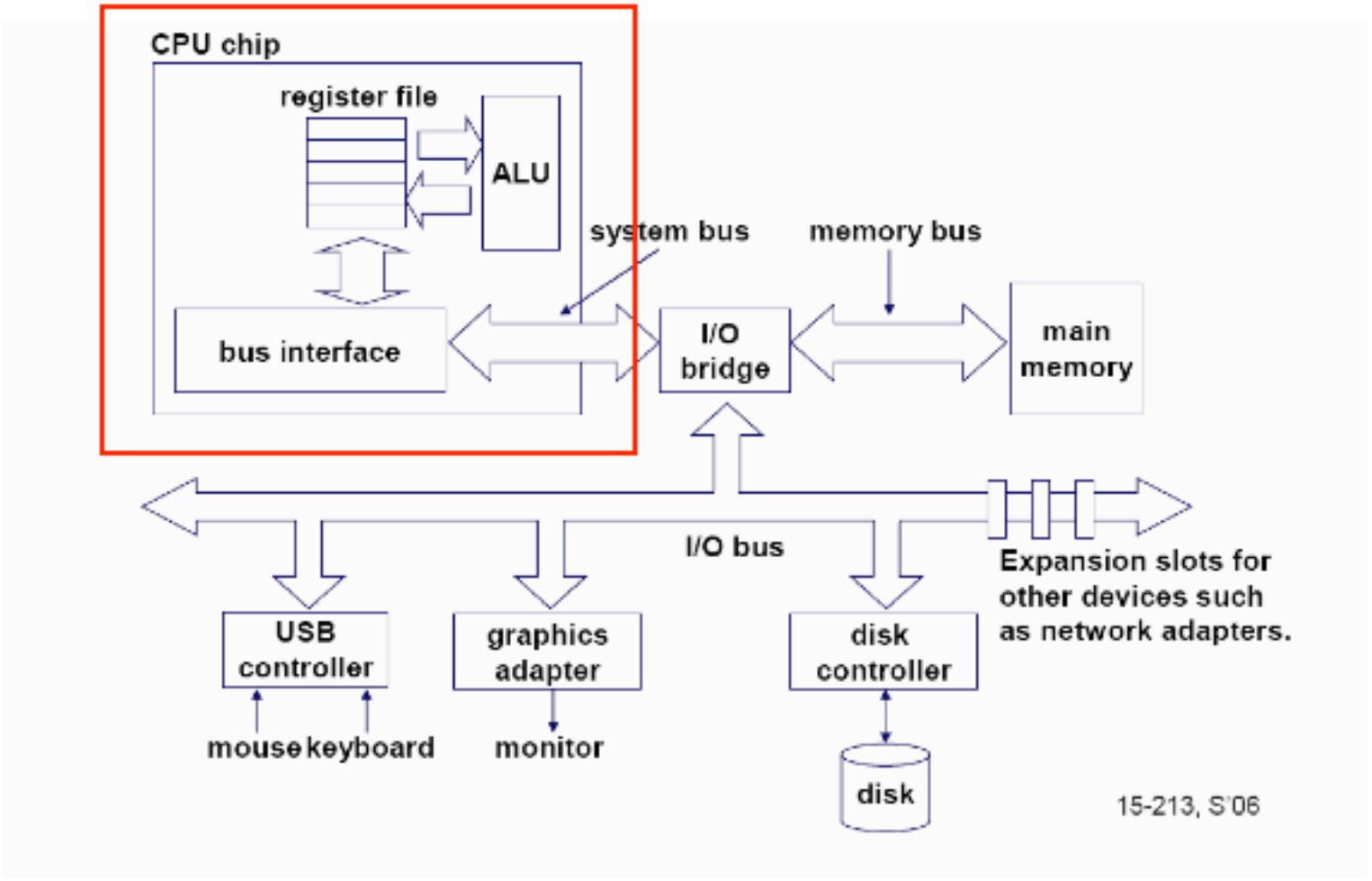
4 x 2 CMP



4 x 2 CMP differences

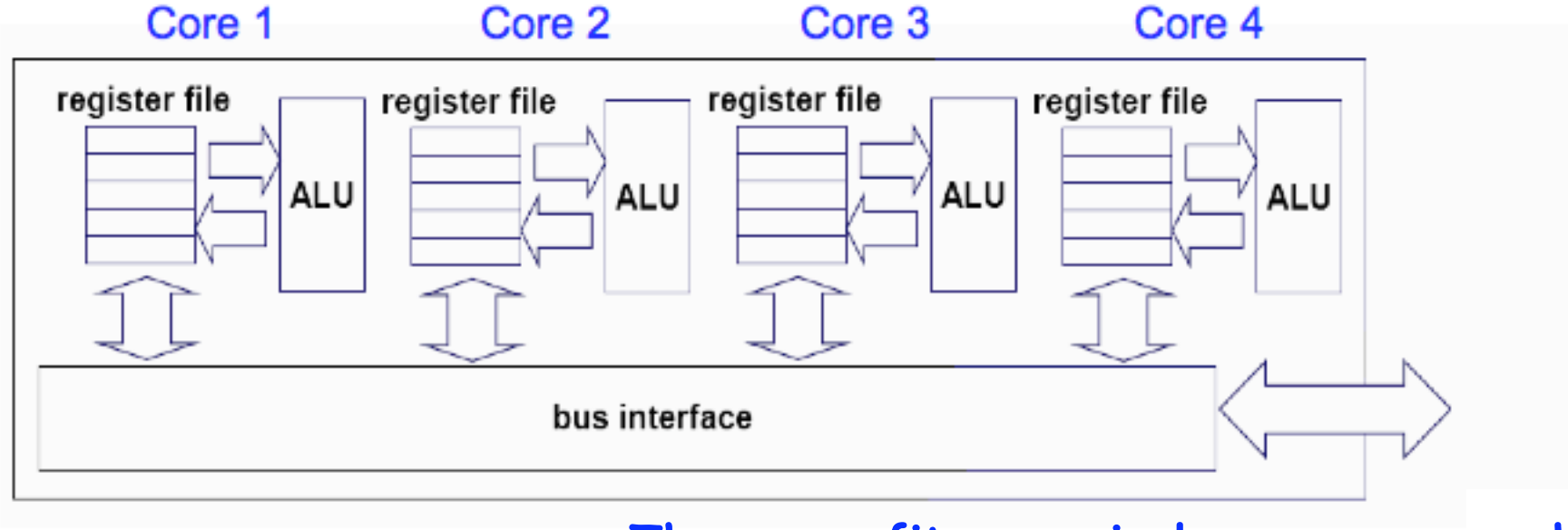
- —simpler issue logic
- —much simpler renaming logic
- —1/4 size branch prediction buffer
- —more execution units

Single-core computer



Multi-core architectures

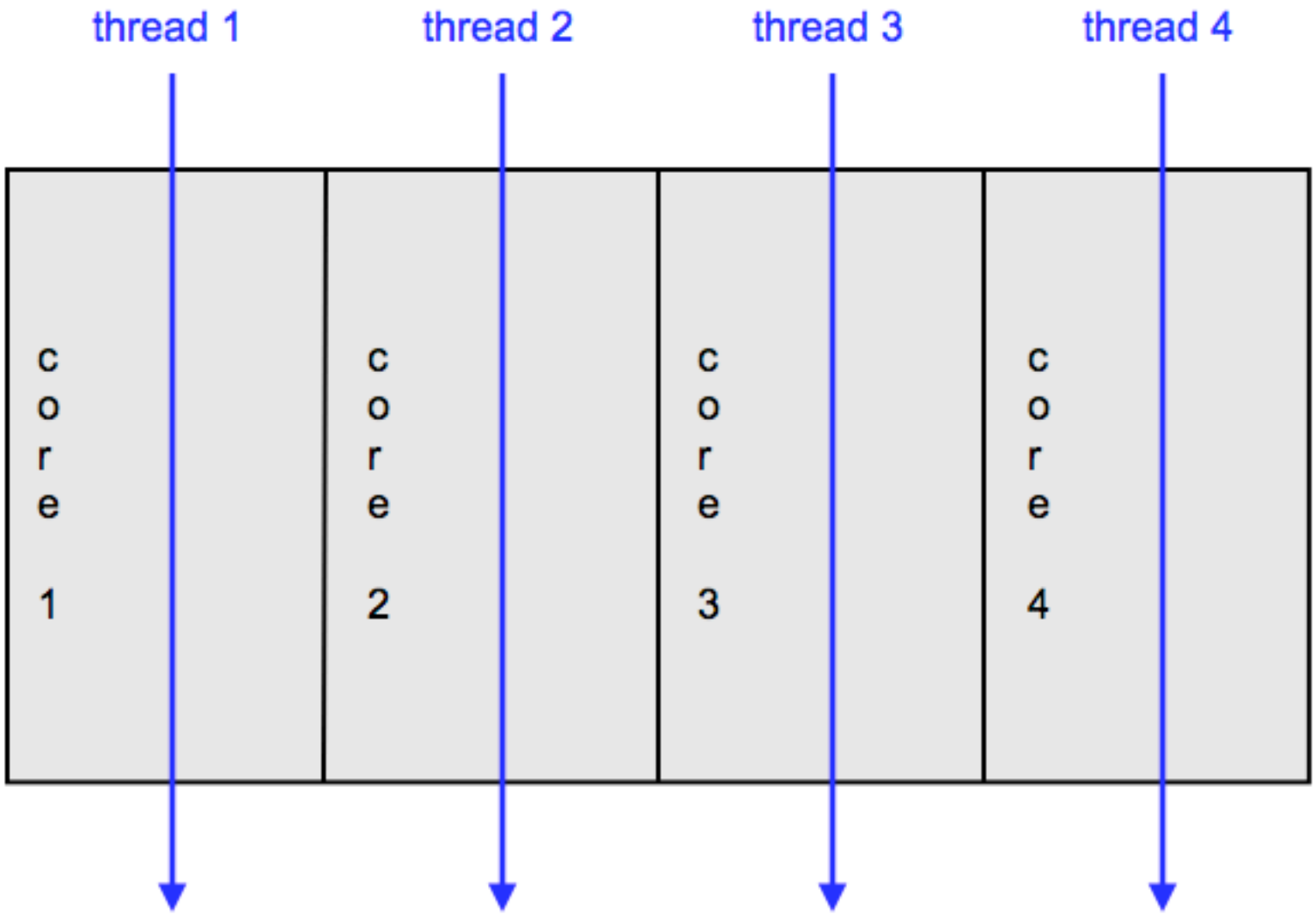
- This lecture is about a new trend in computer architecture:
  Replicate multiple processor cores on a single die.



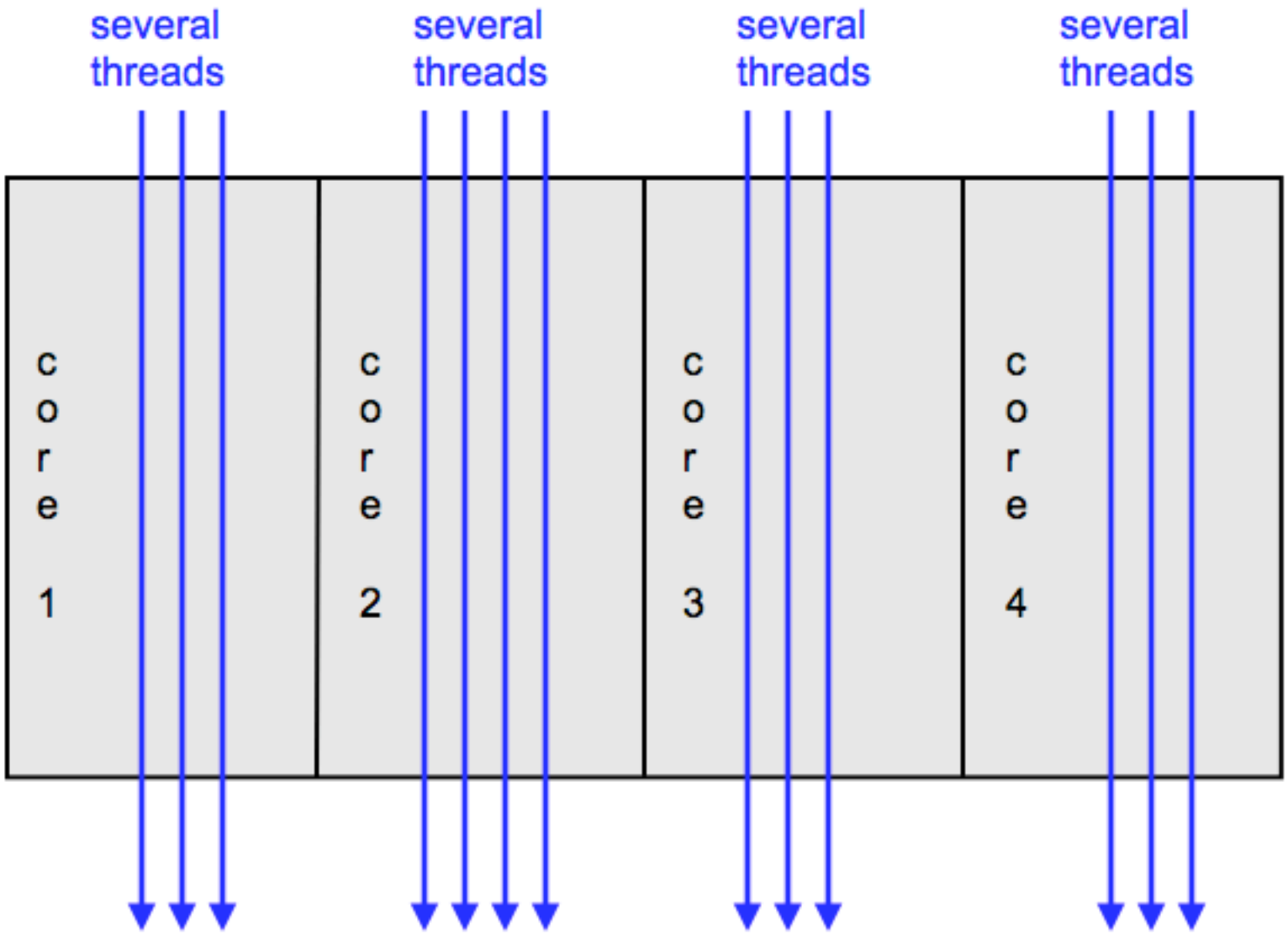
Multi-core CPU chip

The cores fit on a single processor socket
(also called CMP - chip multiprocessor)

The cores run in parallel



Within each core, threads are time-sliced (just like on a uniprocessor)



Back to case study…

Integer Benchmarks

- eqntott: translates logic equations into truth tables
  - —manually parallelized bit vector comparison routine (90% time)
- compress: compresses and uncompresses files in memory
  - —unmodified on both SS and SMT architectures
- m88ksim: simulates Motorola 88000 CPU
  - —manually parallelized into 3 threads using SUIF compiler
    - threads simulate different instructions in different phases
      parallelization analogous to the h/w pipelining it simulates
- MPSim: simulates a bus-based multiprocessor
  - —manually assign parts of model hierarchy to different threads
  - —4 threads: one for each simulated CPU

(standard benchmarks parallelized for comparison)

The case for a single-chip multiprocessor, Olukotun et al., ASPLOS-VII, 1996.

FP and Multiprogramming Benchmarks

Floating point applications (all parallelized with SUIF)

- applu: solves parabolic/eliptic PDEs
- apsi: computes temp, wind, velocity, and distrib. of pollutants
- swim: shallow water model with 1k x 1k grid
- tomcatv: generates mesh using Thompson solver

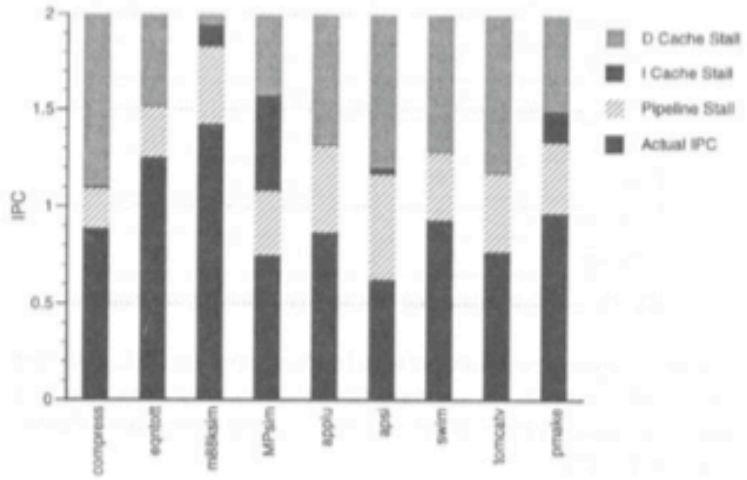
Multiprogramming application

- pmake: performs parallel make of gnuchess (C compiler)
  - —same application simulated on both architectures
  - —OS exploits extra PEs in MP architecture to run parallel compiles

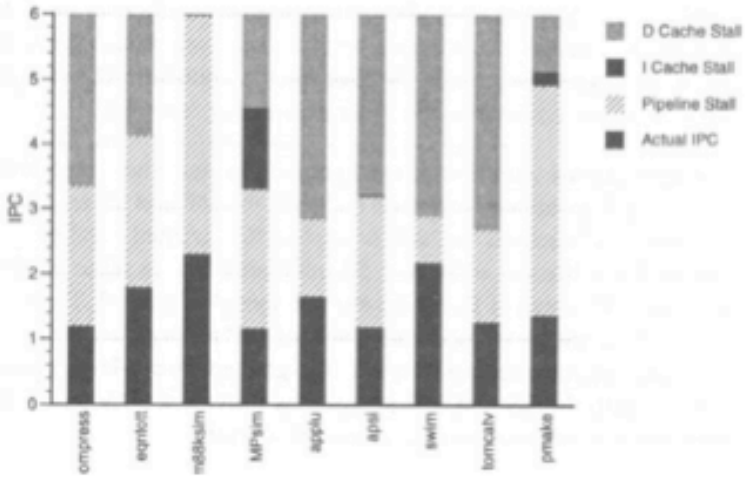
The case for a single-chip multiprocessor, Olukotun et al., ASPLOS-VII, 1996.

IPC Breakdown of Superscalar PEs

2-issue



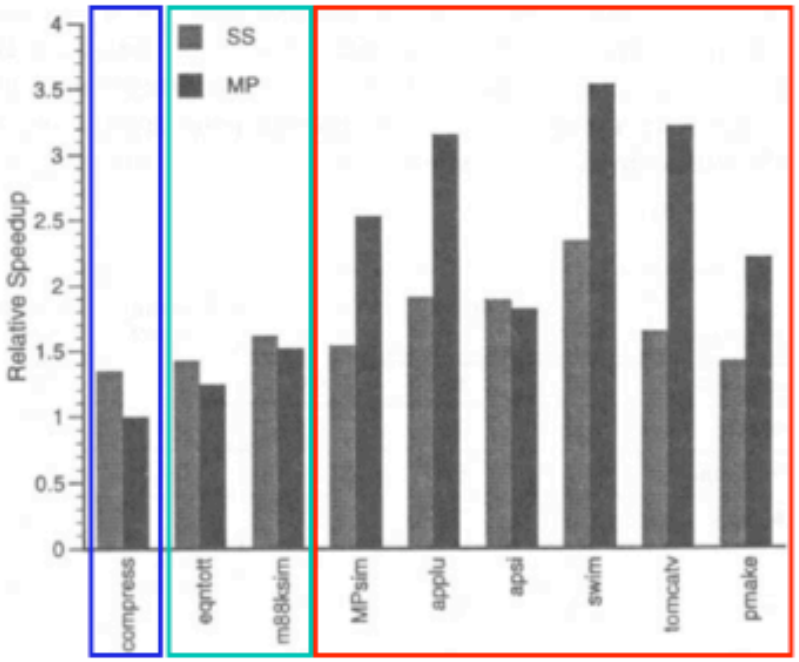
6-issue



The case for a single-chip multiprocessor, Olukotun et al., ASPLOS-VII, 1996.

- Large fraction of time due to dcache stalls
- 6-issue
  - — pipeline stalls increase: lack of IPC
  - — less icache stalls with larger icache
  - — FP appl have significant ILP, but dcache stalls consume > 50% IPC

Performance: 4 x 2 CMP vs. 6-issue SS



- **Non-parallelizable codes**
  - —wide superscalar architecture performs up to 30% better

- **Codes with fine-grain thread-level parallelism**
  - —wide superscalar architecture is at most 10% better w/ same clock frequency
  - —expect that simpler CPUs in CMP would support higher clock rates that would eliminate this difference

- **Codes with coarse-grain thread-level parallelism and multiprogramming workloads**
  - —CMP performs 50-100% better than wide superscalar

(If CPU time constant, performance comes from parallelism)

Take Aways

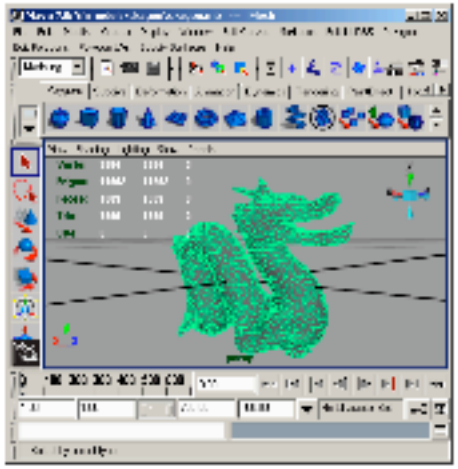
Why multi-core ?

- Difficult to make single-core clock frequencies even higher
- Deeply pipelined circuits:
  - heat problems
  - speed of light problems
  - difficult design and verification
  - large design teams necessary
  - server farms need expensive air-conditioning

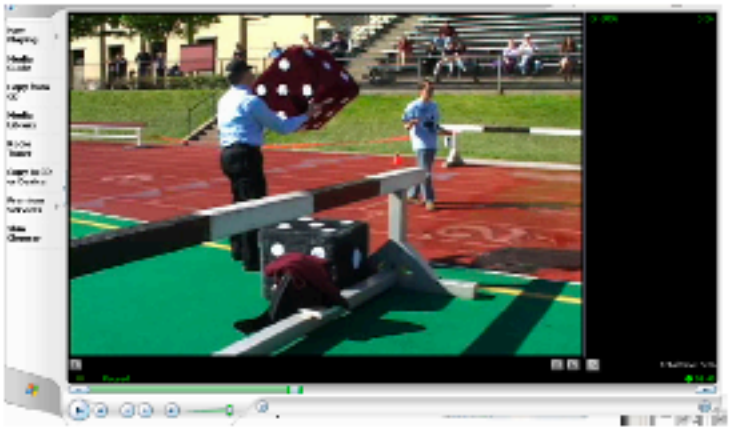
- Many new applications are multithreaded
- General trend in computer architecture (shift towards more parallelism)

What applications benefit from multi-core?

- Database servers
- Web servers (Web commerce)
- Compilers
- Multimedia applications
- Scientific applications, CAD/CAM
- In general, applications with *Thread-level parallelism* (as opposed to instruction-level parallelism)



Each can run on its own core



More examples

- Editing a photo while recording a TV show through a digital video recorder
- Downloading software while running an anti-virus program
- “Anything that can be threaded today will map efficiently to multi-core”
- BUT: some applications difficult to parallelize

Multi-core flavors

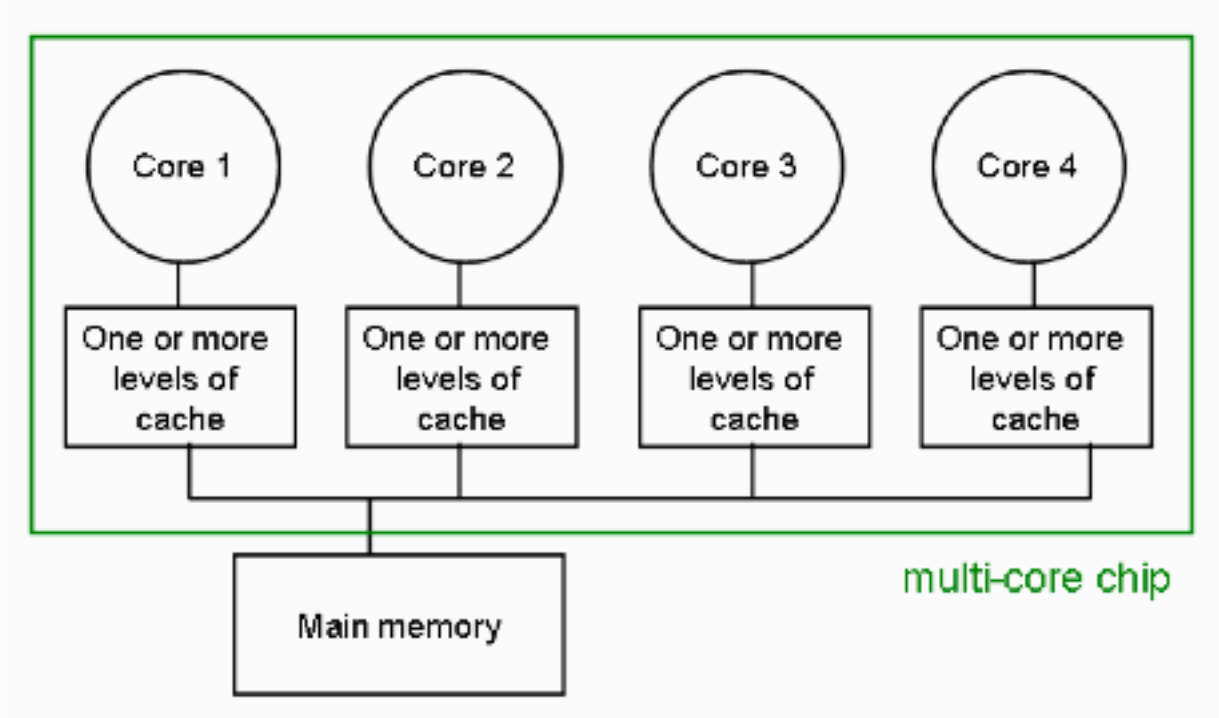
- Cores need not be the same
  - (If they are, we talk about symmetric core machines)
  - (If not, asymmetric)
    - Imagine FPGA + GP processor?

Other issues:

(Amdahl's Law and Parallelization)

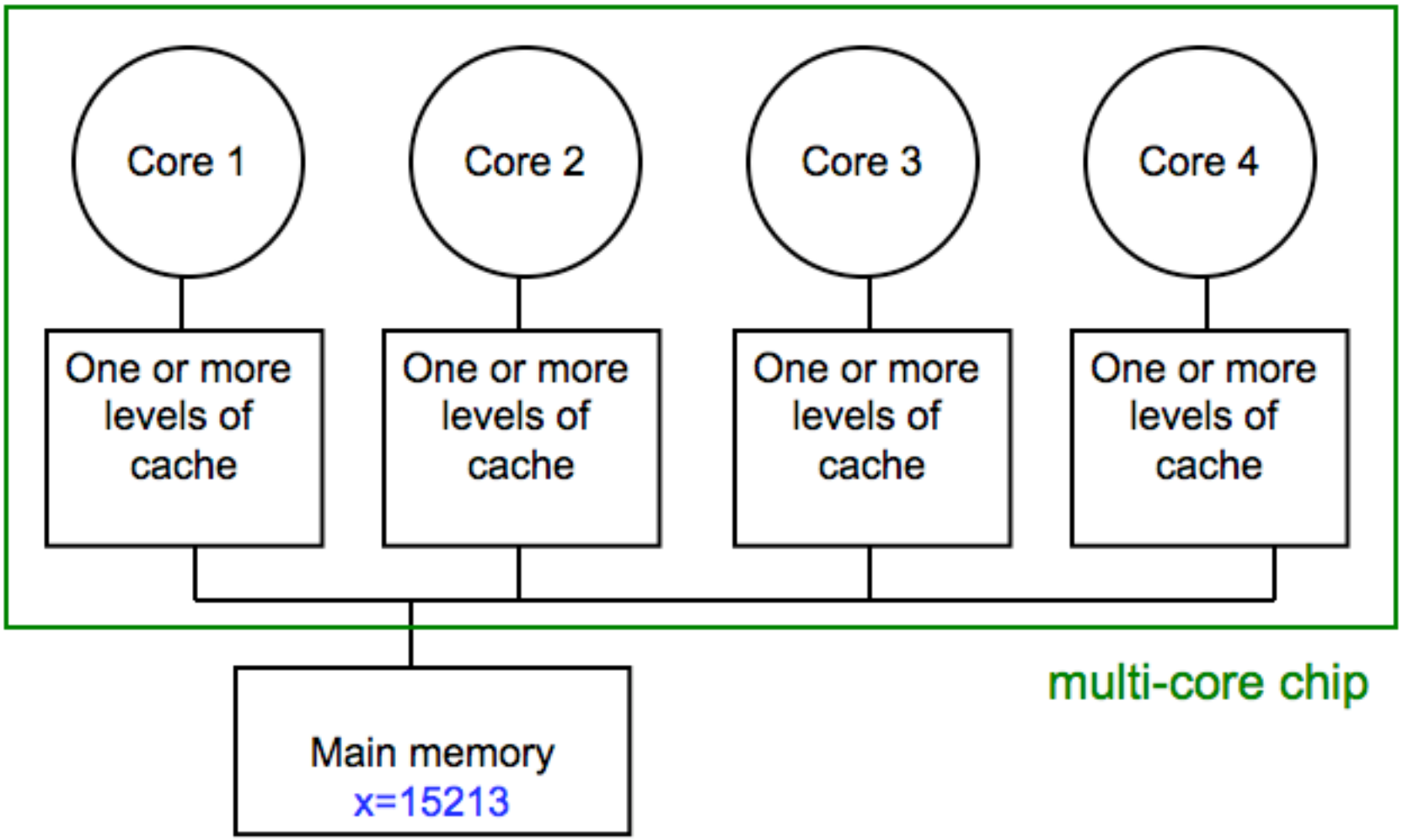
The cache coherence problem

- 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



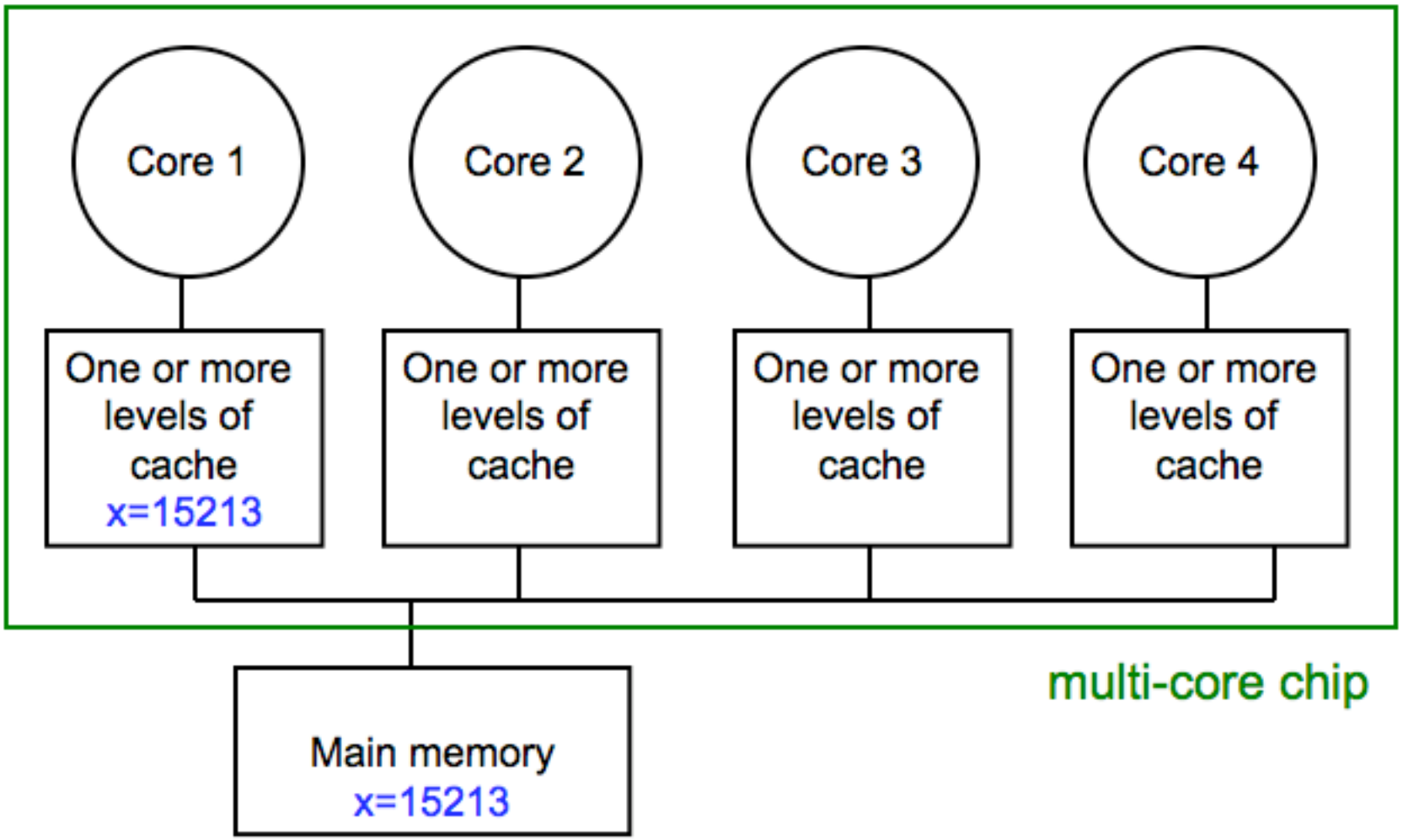
The cache coherence problem

Suppose variable x initially contains 15213



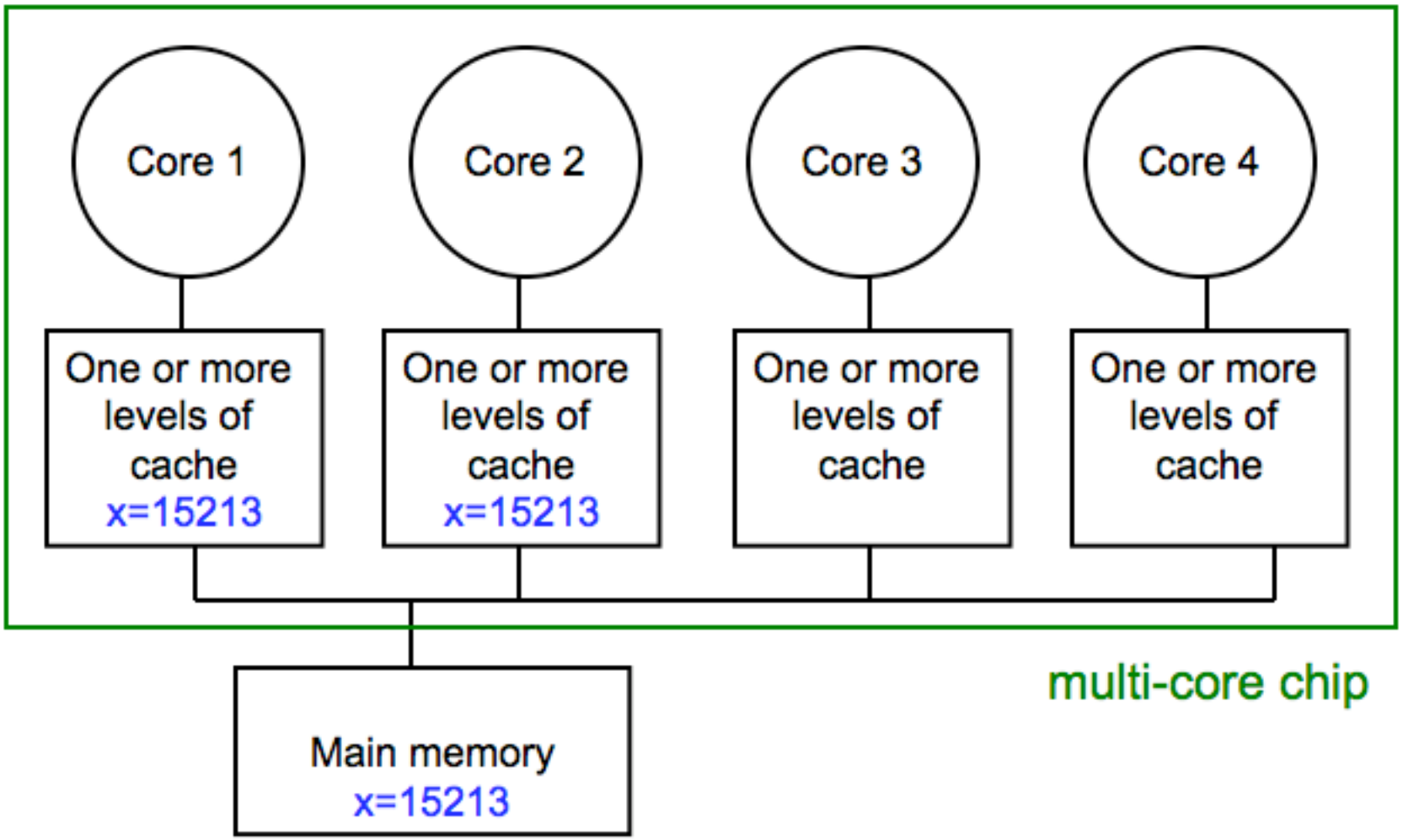
The cache coherence problem

Core 1 reads x



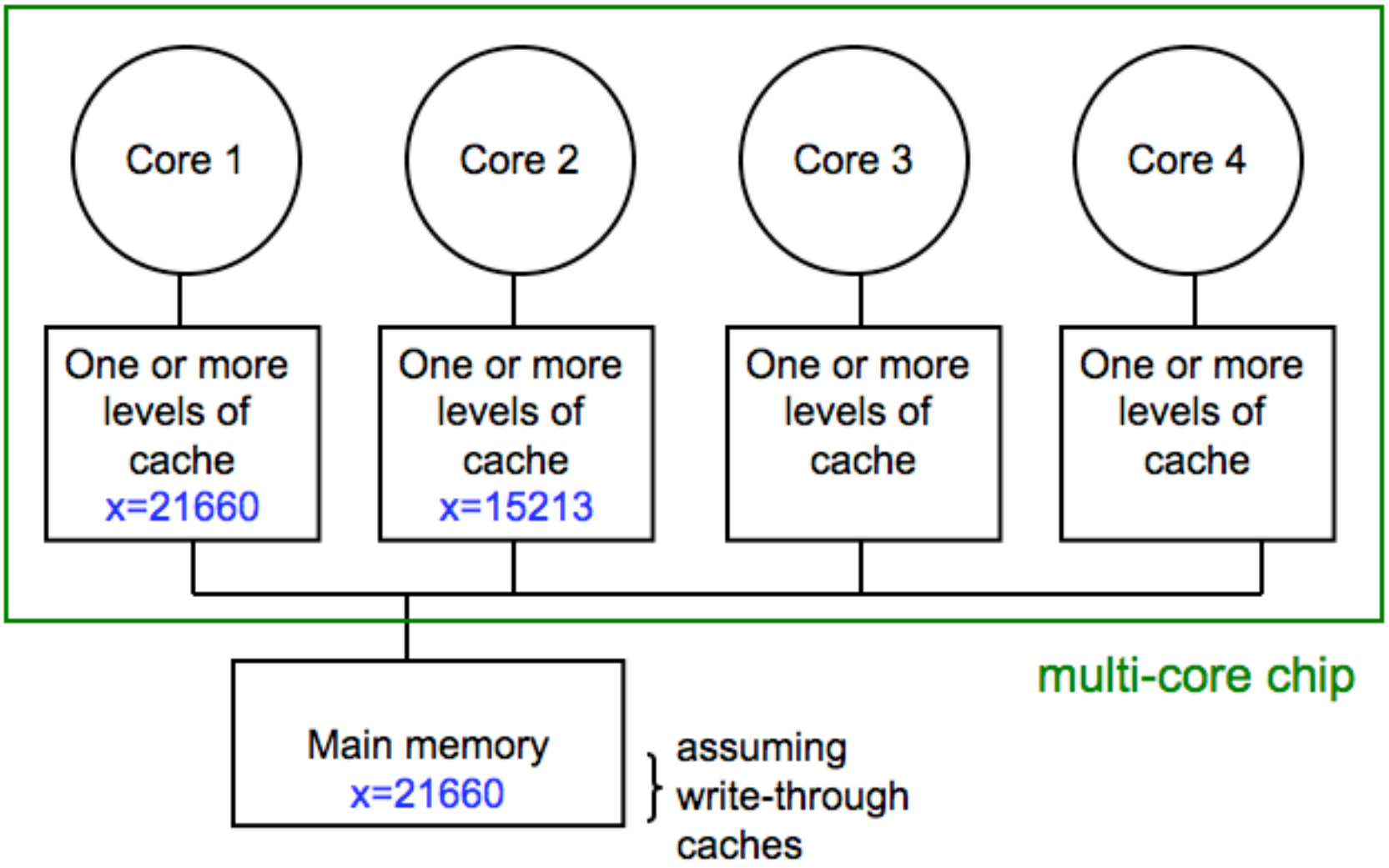
The cache coherence problem

Core 2 reads x



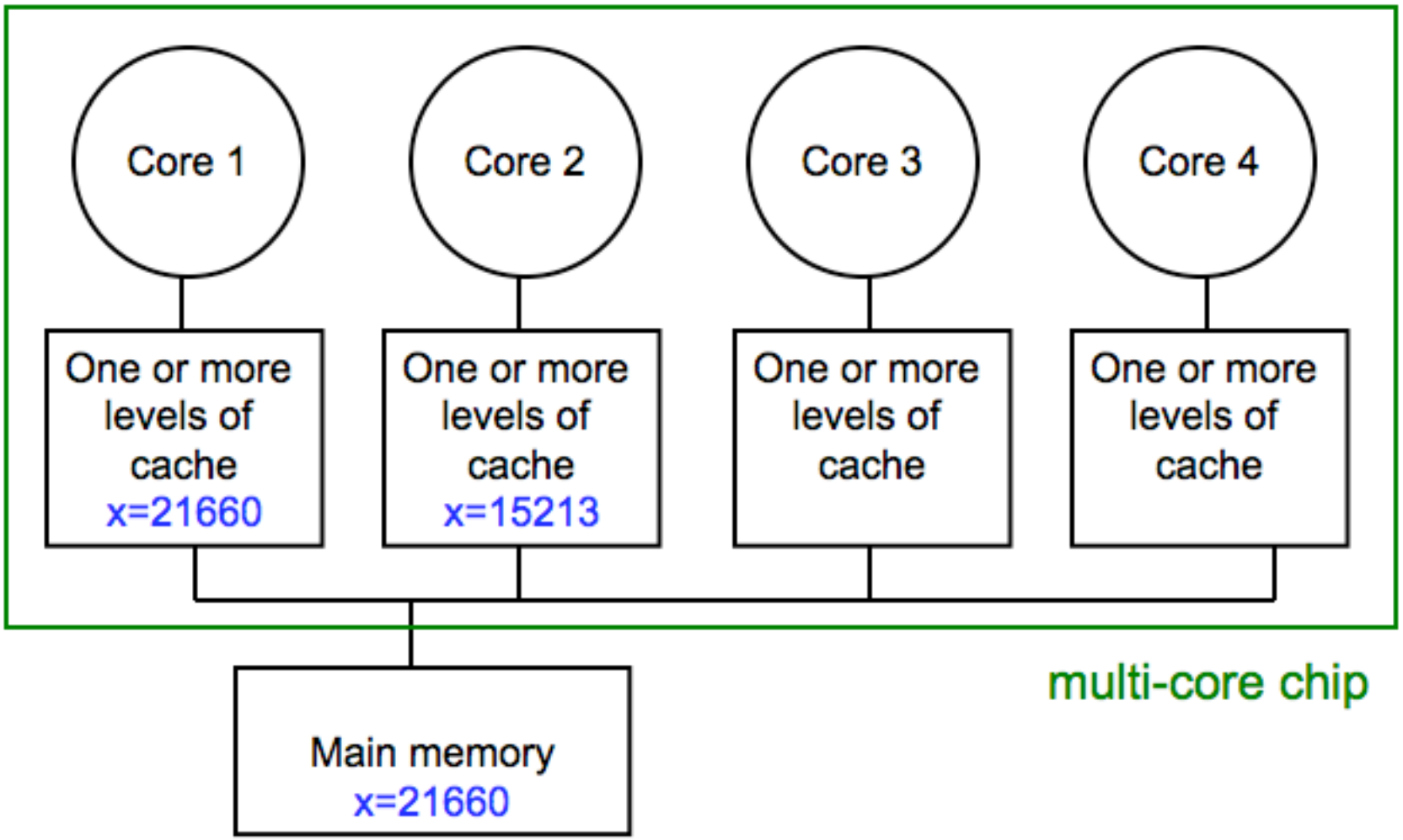
The cache coherence problem

Core 1 writes to x, setting it to 21660



The cache coherence problem

Core 2 attempts to read x… gets a stale copy



Solutions for cache coherence

- This is a general problem with multiprocessors, not limited just to multi-core
- There exist many solution algorithms, coherence protocols, etc.

- A simple solution: *invalidation*-based protocol with *snooping*

Other issues: Core-to-core communication

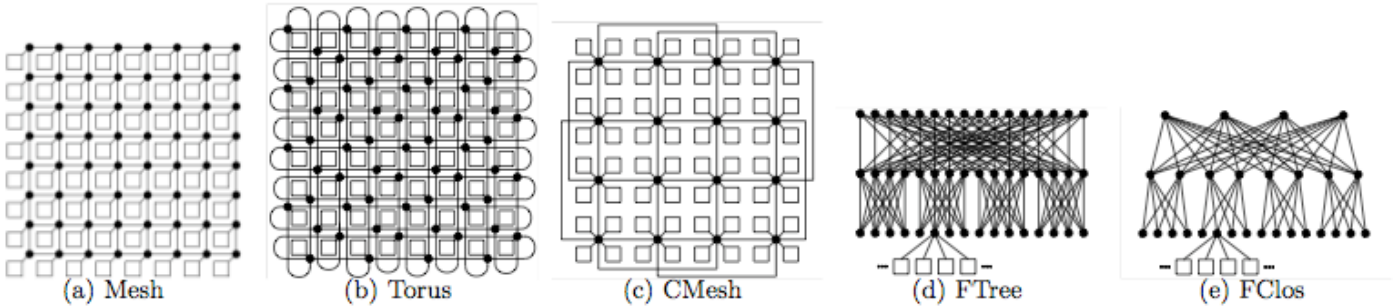


Figure 8: Network Topologies

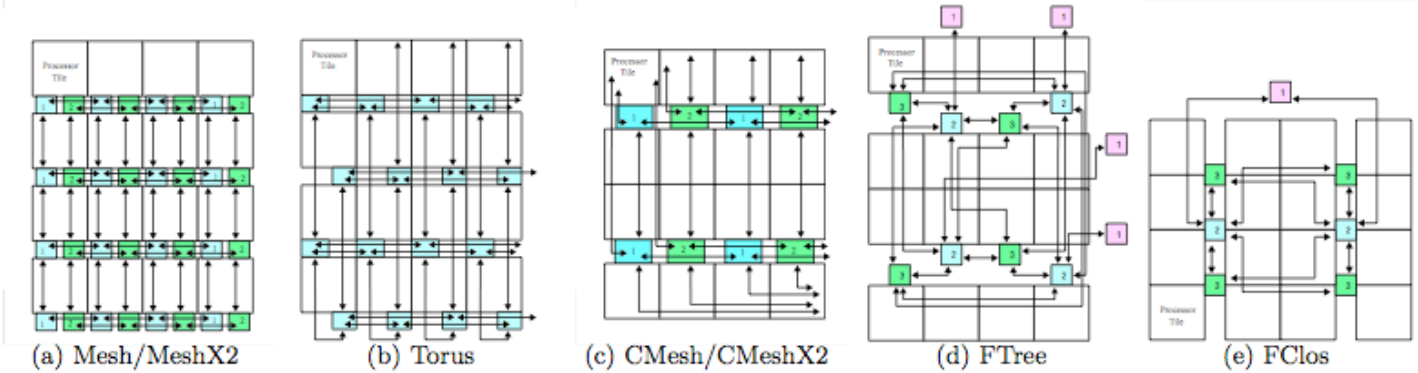


Figure 9: Placement of Routers used to Estimate Area (Lower Left Quadrant)

Must factor in communication costs in processing time too…

Back to Processor-Memory Wall (still need to *feed* cores)

(Peter Kogge will discuss on Monday)

(Not only a problem for multi-core)