

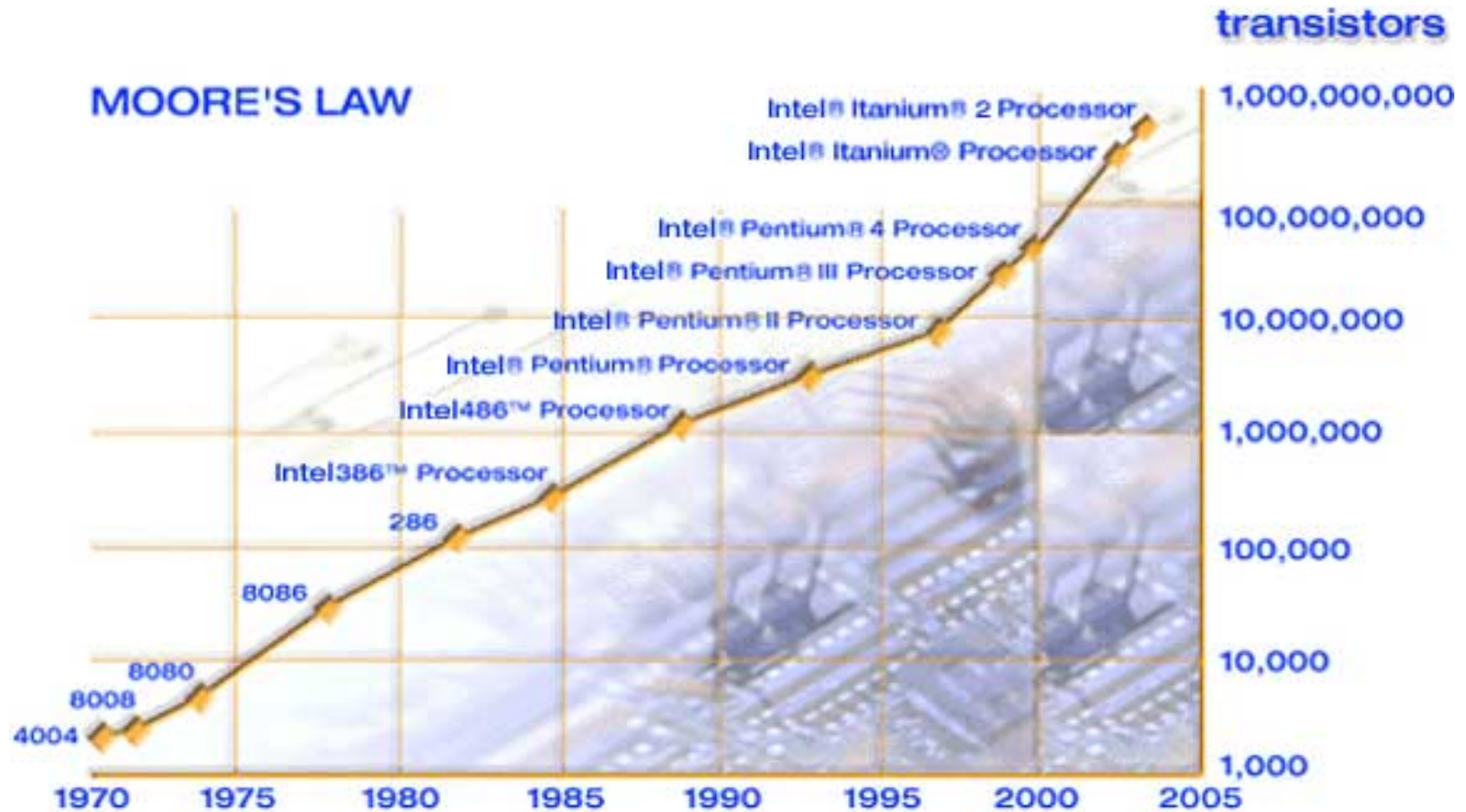
Computer Architecture: Multi-Core Processors: Why?

Onur Mutlu & Seth Copen Goldstein

Carnegie Mellon University

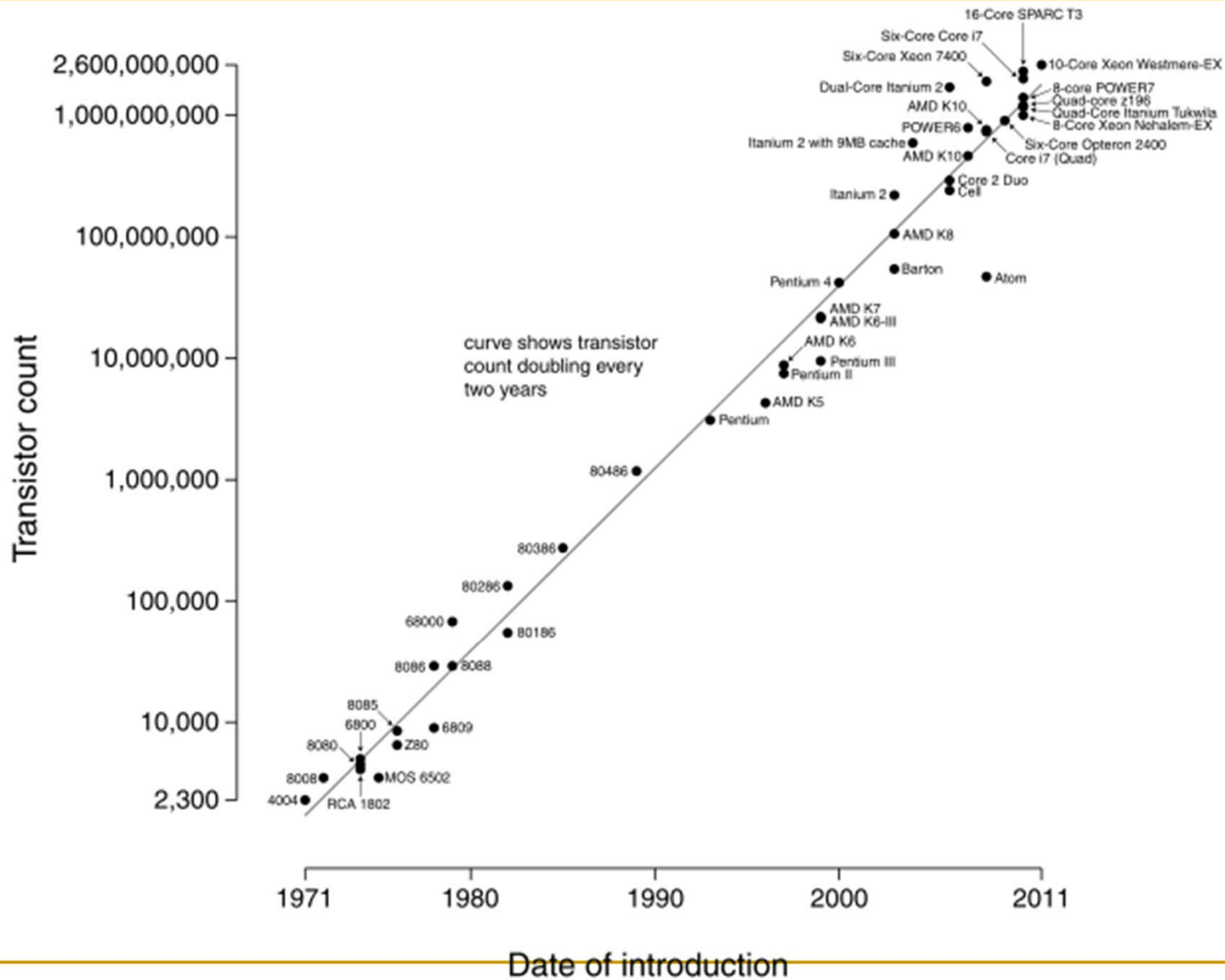
9/11/13

Moore's Law

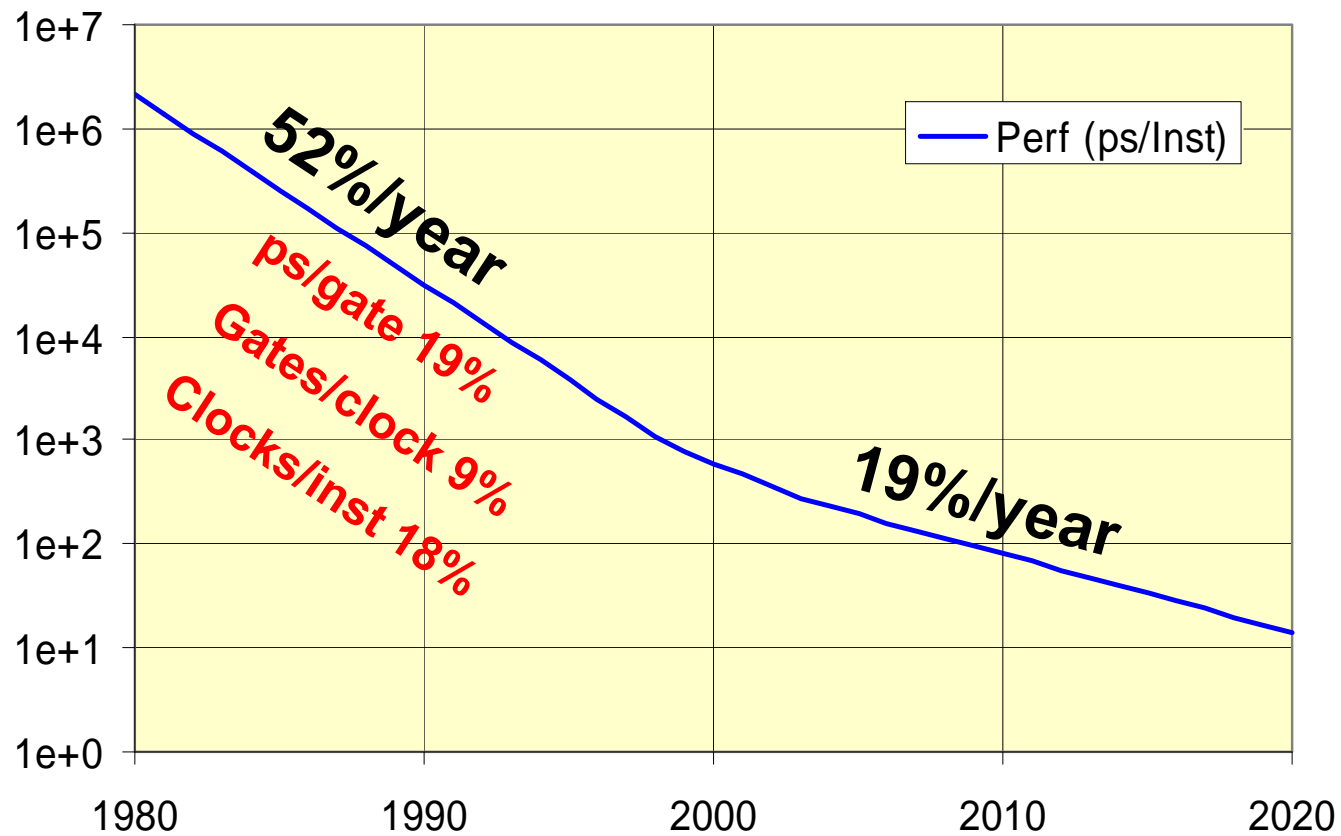


Moore, “Cramming more components onto integrated circuits,”
Electronics, 1965.

Microprocessor Transistor Counts 1971-2011 & Moore's Law



Conventional Processors Stop Scaling Performance by 50% each year



Bill Dally

Multi-Core

- **Idea:** Put multiple processors on the same die.
- Technology scaling (Moore's Law) enables more transistors to be placed on the same die area
- What else could you do with the die area you dedicate to multiple processors?
 - Have a bigger, more powerful core
 - Have larger caches in the memory hierarchy
 - Simultaneous multithreading
 - Integrate platform components on chip (e.g., network interface, memory controllers)
 - ...

Why Not a Better Single Core?

- **Alternative: Bigger, more powerful single core**
 - Larger superscalar issue width, larger instruction window, more execution units, large trace caches, large branch predictors, etc

- + Improves single-thread performance transparently to programmer, compiler
- Very difficult to design (Scalable algorithms for improving single-thread performance elusive)
- Power hungry – many out-of-order execution structures consume significant power/area when scaled. Why?
- Diminishing returns on performance
- Does not significantly help memory-bound application performance (Scalable algorithms for this elusive)

Large Superscalar+OoO vs. Multi-Core

- Olukotun et al., “The Case for a Single-Chip Multiprocessor,” ASPLOS 1996.

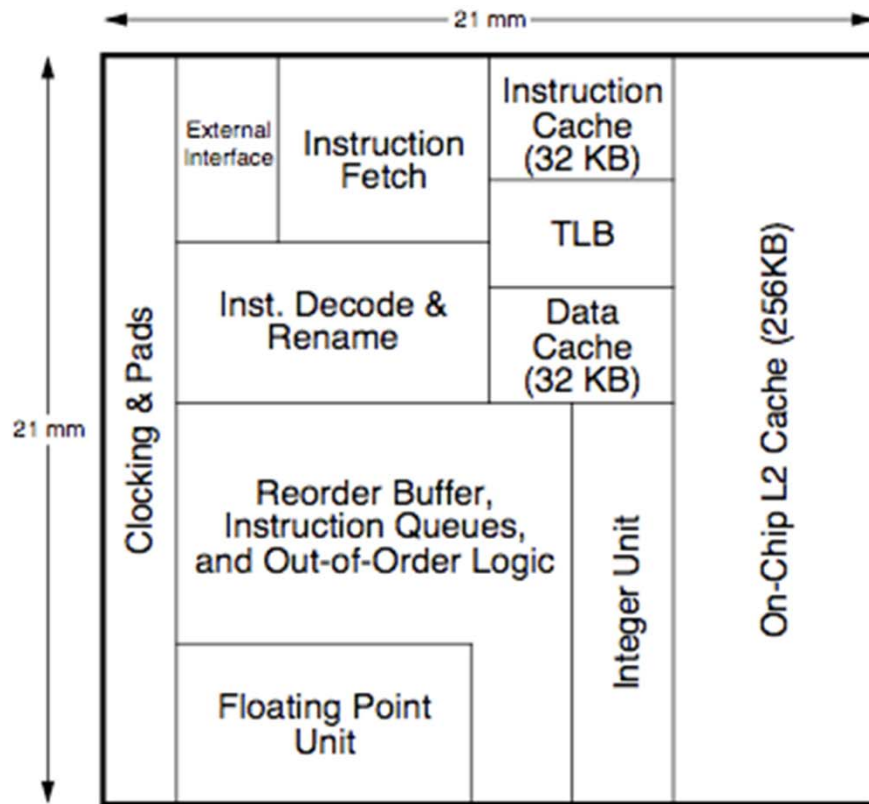


Figure 2. Floorplan for the six-issue dynamic superscalar microprocessor.

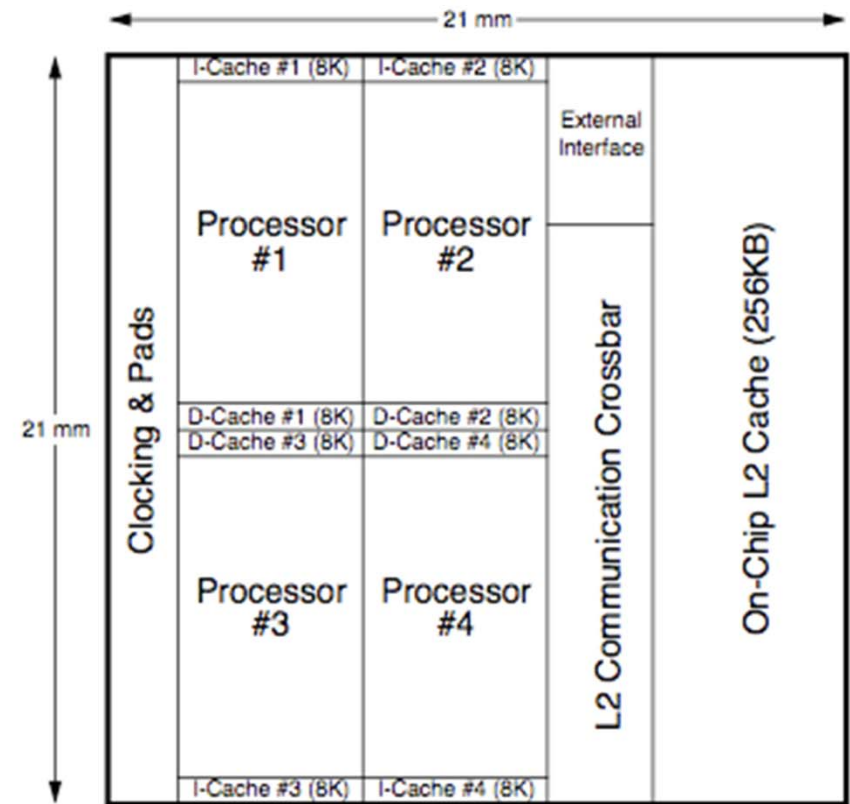


Figure 3. Floorplan for the four-way single-chip multiprocessor.

Multi-Core vs. Large Superscalar+OoO

■ Multi-core advantages

- + Simpler cores → more power efficient, lower complexity, easier to design and replicate, higher frequency (shorter wires, smaller structures)
- + Higher system throughput on multiprogrammed workloads → reduced context switches
- + Higher system performance in parallel applications

■ Multi-core disadvantages

- Requires parallel tasks/threads to improve performance (parallel programming)
- Resource sharing can reduce single-thread performance
- Shared hardware resources need to be managed
- Number of pins limits data supply for increased demand

Large Superscalar vs. Multi-Core

- Olukotun et al., “The Case for a Single-Chip Multiprocessor,” ASPLOS 1996.
- Technology push
 - **Instruction issue queue** size limits the cycle time of the superscalar, OoO processor → diminishing performance
 - Quadratic increase in complexity with issue width
 - **Large, multi-ported register files** to support large instruction windows and issue widths → reduced frequency or longer RF access, diminishing performance
- Application pull
 - Integer applications: little parallelism?
 - FP applications: abundant loop-level parallelism
 - Others (transaction proc., multiprogramming): CMP better fit

Comparison Points...

	6-way SS	4x2-way MP
# of CPUs	1	4
Degree superscalar	6	4 x 2
# of architectural registers	32int / 32fp	4 x 32int / 32fp
# of physical registers	160int / 160fp	4 x 40int / 40fp
# of integer functional units	3	4 x 1
# of floating pt. functional units	3	4 x 1
# of load/store ports	8 (one per bank)	4 x 1
BTB size	2048 entries	4 x 512 entries
Return stack size	32 entries	4 x 8 entries
Instruction issue queue size	128 entries	4 x 8 entries
I cache	32 KB, 2-way S. A.	4 x 8 KB, 2-way S. A.
D cache	32 KB, 2-way S. A.	4 x 8 KB, 2-way S. A.
L1 hit time	2 cycles (4 ns)	1 cycle (2 ns)
L1 cache interleaving	8 banks	N/A
Unified L2 cache	256 KB, 2-way S. A.	256 KB, 2-way S. A.
L2 hit time / L1 penalty	4 cycles (8 ns)	5 cycles (10 ns)
Memory latency / L2 penalty	50 cycles (100 ns)	50 cycles (100 ns)

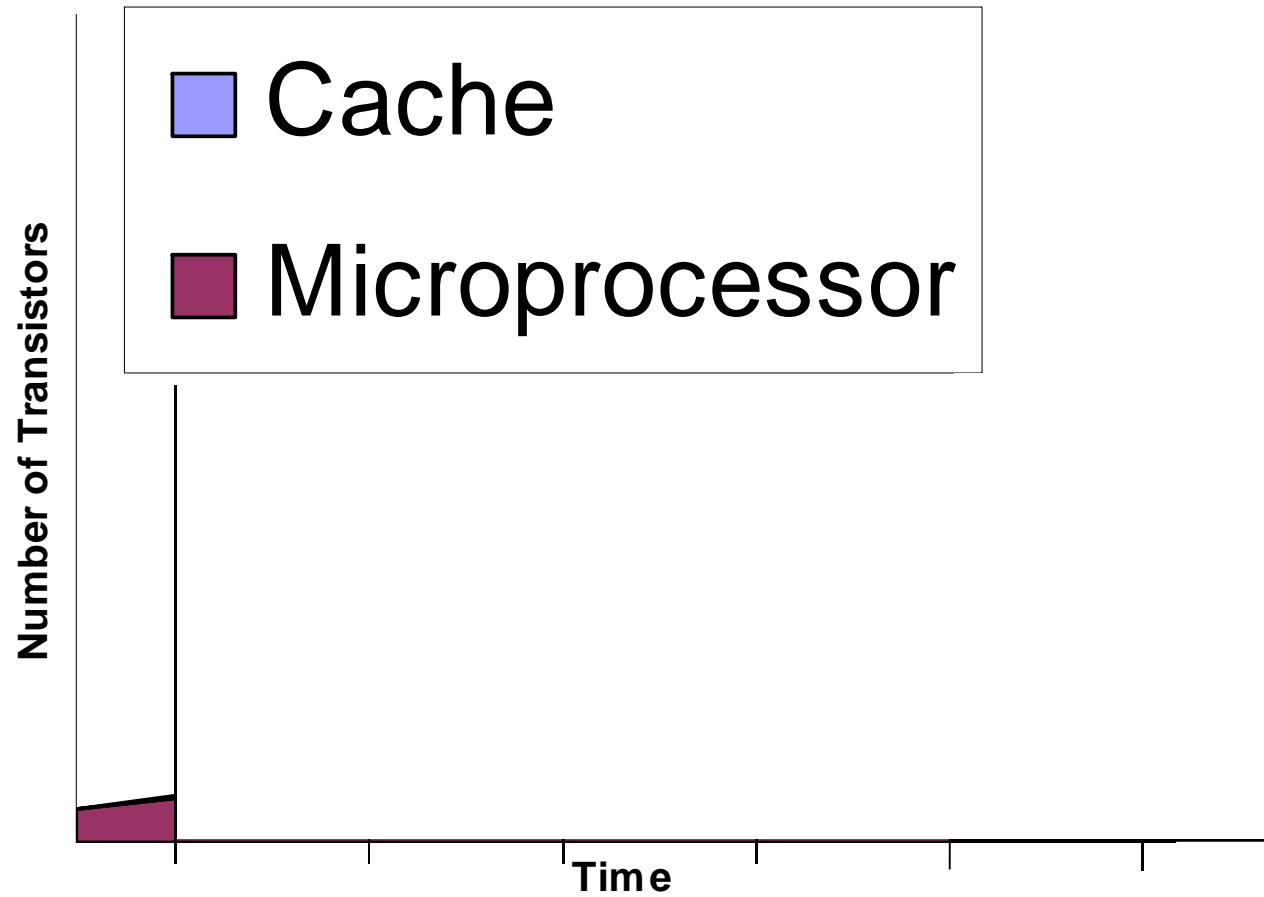
Table 1. Key characteristics of the two microarchitectures

Why Not bigger caches?

- Alternative: Bigger caches

- + Improves single-thread performance transparently to programmer, compiler
- + Simple to design
- Diminishing single-thread performance returns from cache size.
Why?
- Multiple levels complicate memory hierarchy

Cache vs. Core



Why Not Multithreading?

■ Alternative: (Simultaneous) Multithreading

- + Exploits thread-level parallelism (just like multi-core)
- + Good single-thread performance with SMT
- + No need to have an entire core for another thread
- + Parallel performance aided by tight sharing of caches

- Scalability is limited: need bigger register files, more function units, larger issue width (and associated costs) to have many threads → complex with many threads
- Parallel performance limited by shared fetch bandwidth
- Extensive resource sharing at the pipeline and memory system reduces both single-thread and parallel application performance

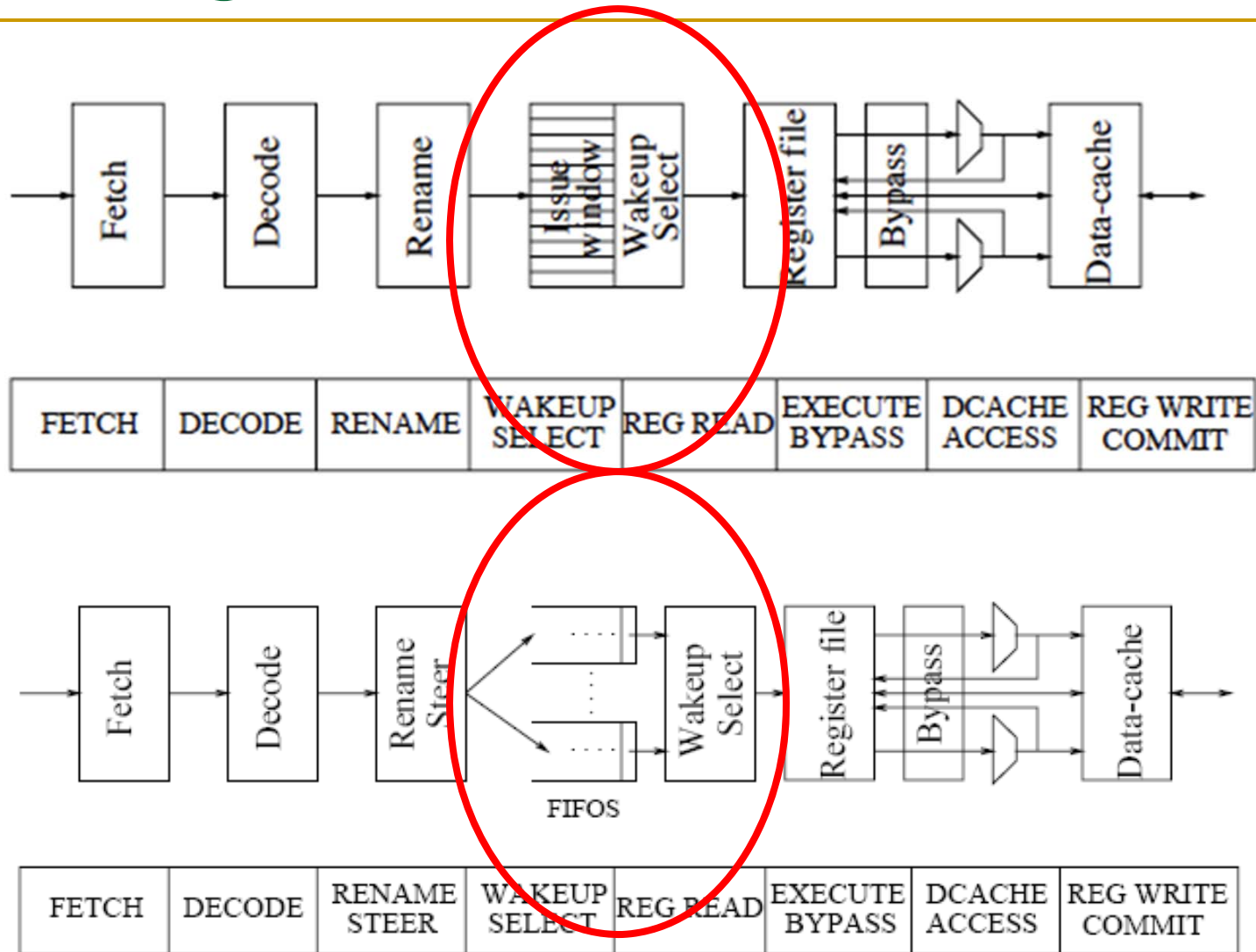
Why Not System on a Chip?

- Alternative: Integrate platform components on chip instead
 - + Speeds up many system functions (e.g., network interface cards, Ethernet controller, memory controller, I/O controller)
 - Not all applications benefit (e.g., CPU intensive code sections)

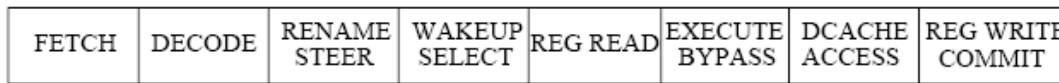
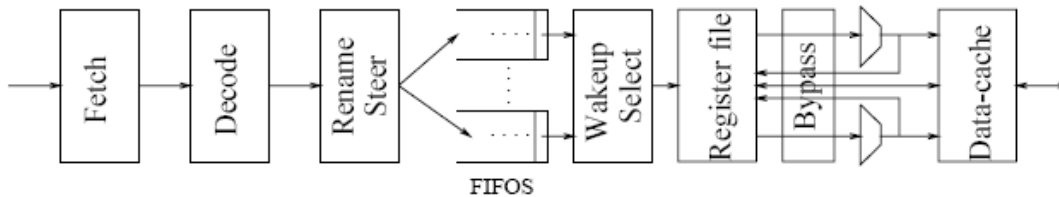
Why Not Clustering?

- **Alternative: More scalable superscalar, out-of-order engines**
 - Clustered superscalar processors (with multithreading)
 - + Simpler to design than superscalar, more scalable than simultaneous multithreading (less resource sharing)
 - + Can improve both single-thread and parallel application performance
 - Diminishing performance returns on single thread: Clustering reduces IPC performance compared to monolithic superscalar. Why?
 - Parallel performance limited by shared fetch bandwidth
 - Difficult to design

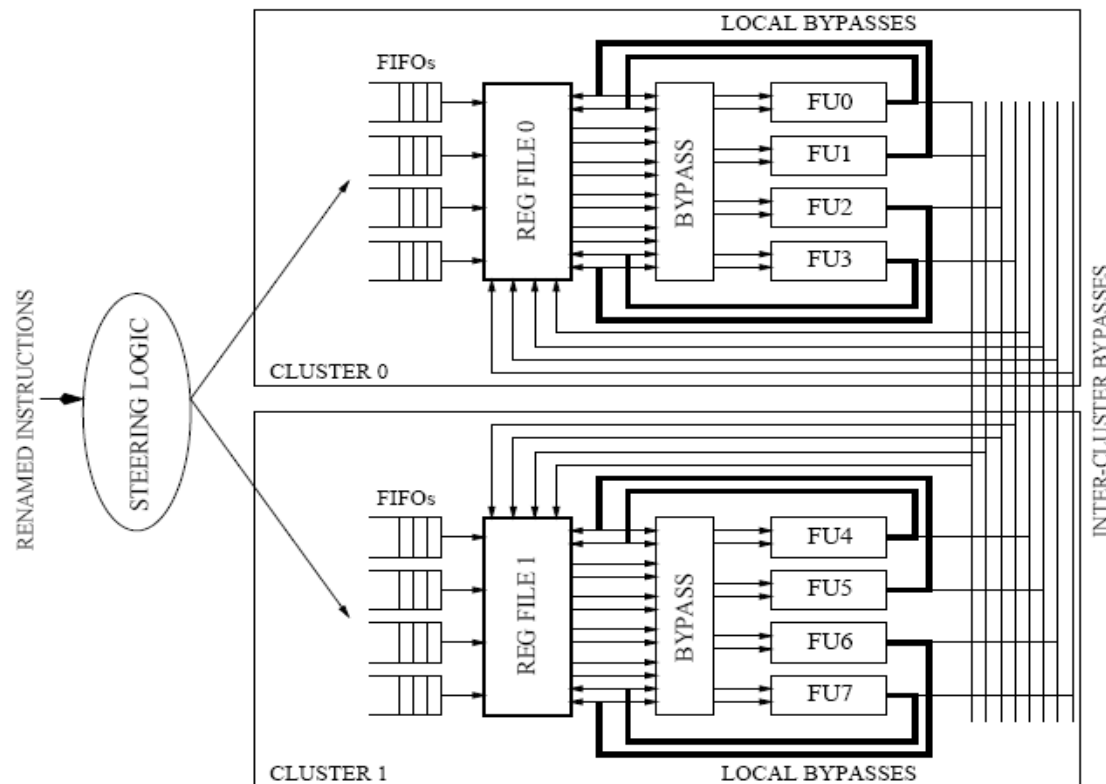
Clustering (I)



Clustering (II)



- Each scheduler is a FIFO
- + Simpler
- + Can have N FIFOs (OoO w.r.t. each other)
- + Reduces scheduling complexity
- More dispatch stalls

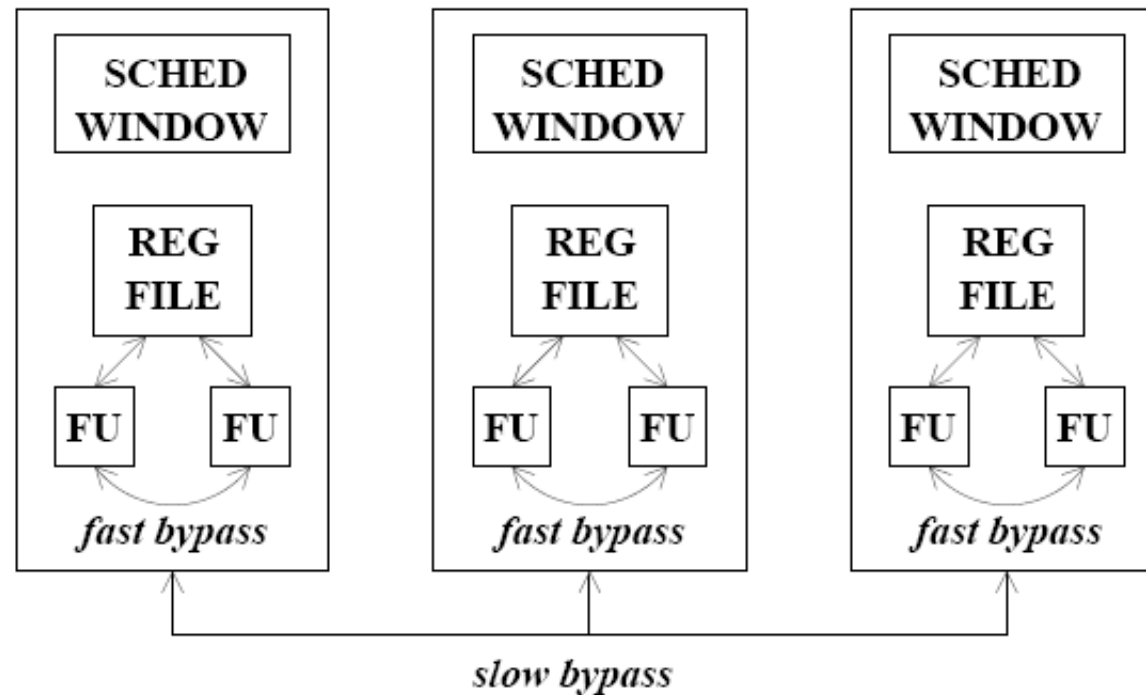


Inter-cluster bypass: Results produced by an FU in Cluster 0 is not individually forwarded to each FU in another cluster.

- Palacharla et al., “Complexity Effective Superscalar Processors,” ISCA 1997.

Clustering (III)

- Scheduling within each cluster can be out of order



Brown, “[Reducing Critical Path Execution Time by Breaking Critical Loops](#),” UT-Austin 2005.

Clustered Superscalar+OoO Processors

- **Clustering** (e.g., Alpha 21264 integer units)
 - Divide the scheduling window (and register file) into multiple clusters
 - Instructions steered into clusters (e.g. based on dependence)
 - Clusters schedule instructions out-of-order, within cluster scheduling can be in-order
 - Inter-cluster communication happens via register files (no full bypass)

- + Smaller scheduling windows, simpler wakeup algorithms
- + Fewer ports into register files
- + Faster within-cluster bypass

- Extra delay when instructions require across-cluster communication
- inherent difficulty of steering logic

Why Not Multi-Chip symmetric Multiproc?

- **Alternative: Traditional symmetric multiprocessors**
 - + Smaller die size (for the same processing core)
 - + More memory bandwidth (no pin bottleneck)
 - + Fewer shared resources → less contention between threads
 - Long latencies between cores (need to go off chip) → shared data accesses limit performance → parallel application scalability is limited
 - Worse resource efficiency due to less sharing → worse power/energy efficiency

Why Multi-Core?

- Other alternatives?
 - Dataflow?
 - VLIW?
 - Vector processors (SIMD)?
 - Streaming processors?
 - Integrating DRAM on chip?
 - Reconfigurable logic? (general purpose?)

Review: Multi-Core Alternatives

- Bigger, more powerful single core
- Bigger caches
- (Simultaneous) multithreading
- Integrate platform components on chip instead
- More scalable superscalar, out-of-order engines
- Traditional symmetric multiprocessors
- Dataflow?
- Vector processors (SIMD)?
- Integrating DRAM on chip?
- Reconfigurable logic? (general purpose?)
- Other alternatives?
- Your solution?

Why Multi-Core (Cynically)

- Huge investment and need ROI
- Have to offer some kind of upgrade path
- It is easy for the processor manufacturers

Why Multi-Core (Cynically)

- Huge investment and need ROI
- Have to offer some kind of upgrade path
- It is easy for the processor manufacturers

- But, Seriously:
- Some easy parallelism
 - Most general purpose machines run multiple tasks at a time
 - Some (very important) Apps have easy parallelism
- Power is a real issue
- Design complexity is very costly

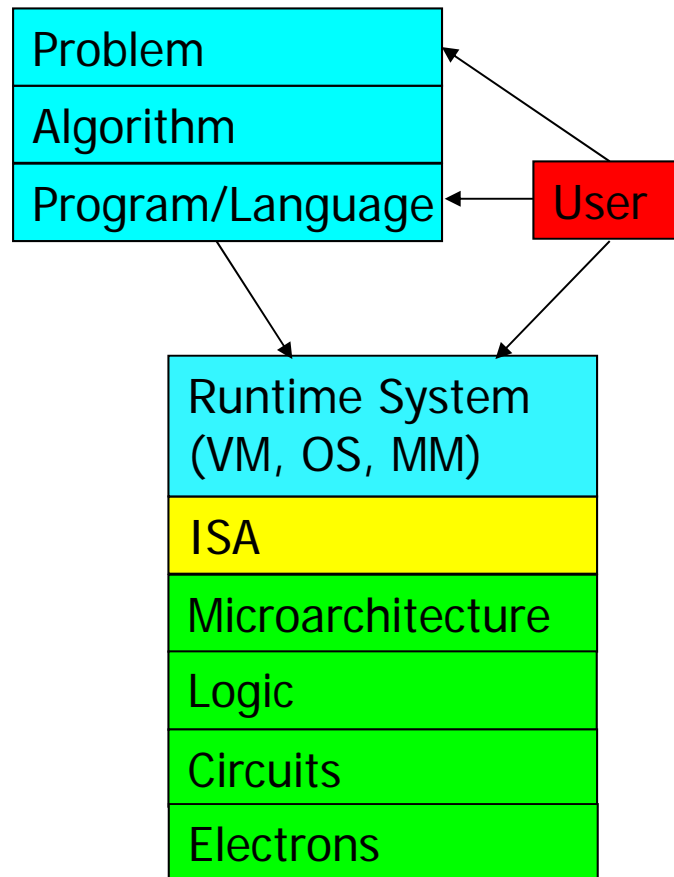
- Is it the right solution?

Computer Architecture Today (I)

- Today is a very exciting time to study computer architecture
- Industry is in a large paradigm shift (to multi-core and beyond) – many different potential system designs possible
- **Many difficult problems** *motivating* and *caused by* the shift
 - Power/energy constraints → multi-core?, accelerators?
 - Complexity of design → multi-core?
 - Difficulties in technology scaling → new technologies?
 - Memory wall/gap
 - Reliability wall/issues
 - Programmability wall/problem → single-core?
- No clear, definitive answers to these problems

Computer Architecture Today (II)

- These problems affect all parts of the computing stack – if we do not change the way we design systems



- No clear, definitive answers to these problems
-

Computer Architecture Today (III)

- You can revolutionize the way computers are built, if you understand both the hardware and the software (and change each accordingly)
- You can invent new paradigms for computation, communication, and storage
- Recommended book: Kuhn, "[The Structure of Scientific Revolutions](#)" (1962)
 - Pre-paradigm science: no clear consensus in the field
 - Normal science: dominant theory used to explain things (business as usual); exceptions considered anomalies
 - Revolutionary science: underlying assumptions re-examined

... but, first ...

- Let's understand the fundamentals...
- You can change the world only if you understand it well enough...
 - Especially the past and present dominant paradigms
 - And, their advantages and shortcomings -- tradeoffs

Computer Architecture: Multi-Core Processors: Why?

Prof. Onur Mutlu
Carnegie Mellon University

Backup slides

Referenced Readings

- Moore, “Cramming more components onto integrated circuits,” *Electronics*, 1965.
- Olukotun et al., “The Case for a Single-Chip Multiprocessor,” *ASPLOS* 1996.
- Tullsen et al., “Simultaneous Multithreading: Maximizing On-Chip Parallelism,” *ISCA* 1995.
- Kessler, “The Alpha 21264 Microprocessor,” *IEEE Micro* 1999.
- Brown, “Reducing Critical Path Execution Time by Breaking Critical Loops,” *UT-Austin* 2005.
- Palacharla et al., “Complexity Effective Superscalar Processors,” *ISCA* 1997.
- Kuhn, “The Structure of Scientific Revolutions,” 1962.

Related Videos

- Multi-Core Systems and Heterogeneity
 - <http://www.youtube.com/watch?v=LIDxT0hPI2U&list=PLVngZ7BemHHV6N0ejHhwOfLwTr8Q-UKXj&index=1>
 - <http://www.youtube.com/watch?v=Q0zyLVnzkrM&list=PLVngZ7BemHHV6N0ejHhwOfLwTr8Q-UKXj&index=2>