Lecture 7
Thread Level Parallelism (1)

EEC 171 Parallel Architectures

John Owens
UC Davis

Credits

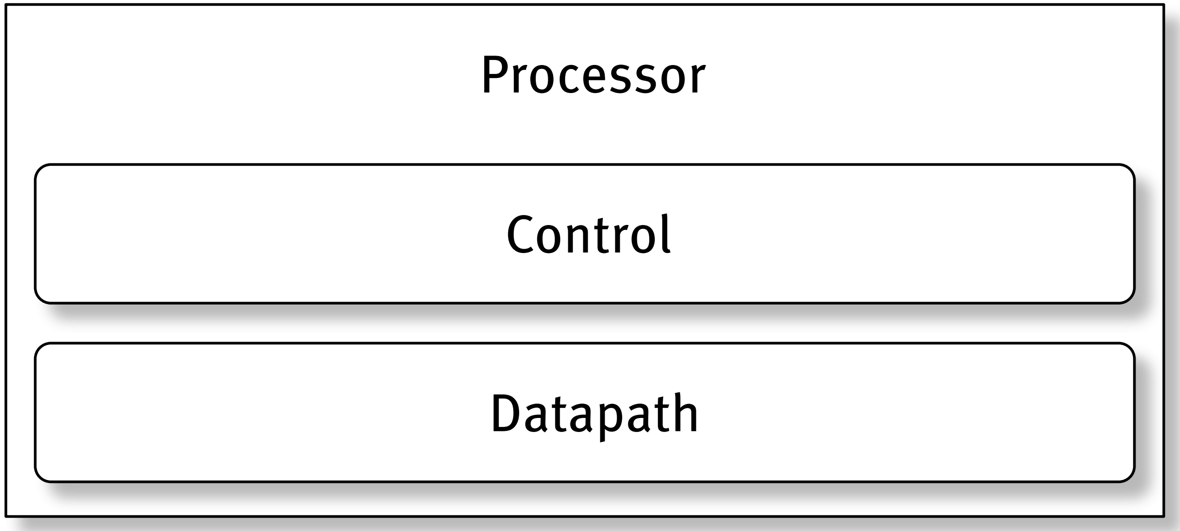
- © John Owens / UC Davis 2007–17.
- Thanks to many sources for slide material: Computer Organization and Design (Patterson & Hennessy) © 2005, Computer Architecture (Hennessy & Patterson) © 2007, Inside the Machine (Jon Stokes) © 2007, © Dan Connors / University of Colorado 2007, © Wen-Mei Hwu/David Kirk, University of Illinois 2007, © David Patterson / UCB 2003–7, © John Lazzaro / UCB 2006, © Mary Jane Irwin / Penn State 2005, © John Kubiatowicz / UCB 2002, © Krste Asinovic/Arvind / MIT 2002, © Morgan Kaufmann Publishers 1998.

Michael Abrash, 4/1/09

- To understand what Larrabee is, it helps to understand why Larrabee is. Intel has been making single cores faster for decades by increasing the clock speed, increasing cache size, and using the extra transistors each new process generation provides to boost the work done during each clock. That process certainly hasn't stopped, and will continue to be an essential feature of main system processors for the foreseeable future, but it's getting harder. This is partly because most of the low-hanging fruit has already been picked, and partly because processors are starting to run up against power budgets, and both out-of-order instruction execution and higher clock frequency are power-intensive.

  More recently, Intel has also been applying additional transistors in a different way – by adding more cores. This approach has the great advantage that, given software that can parallelize across many such cores, performance can scale nearly linearly as more and more cores get packed onto chips in the future.

What We Know



- What new techniques have we learned that make ...
    - ... control go fast?
    - ... datapath go fast?

Cook analogy

- We want to prepare food for several banquets, each of which requires many dinners.
- We have two positions we can fill:
    - The boss (control), who gets all the ingredients and tells the chef what to do
    - The chef (datapath), who does all the cooking
- ILP is analogous to:
    - One ultra-talented boss with many hands
    - One ultra-talented chef with many hands

Cook analogy

- We want to prepare food for several banquets, each of which requires many dinners.
- But one boss and one chef isn’t enough to do all our cooking.
- What are our options?

Chef scaling

- What’s the cheapest way to cook more?
- Is it easy or difficult to share (ingredients, cooked food, etc.) between chefs?
- Which method of scaling is most flexible?

Flynn’s Classification Scheme

- SISD – single instruction, single data stream
  - Uniprocessors
- SIMD – single instruction, multiple data streams
  - single control unit broadcasting operations to multiple datapaths
- MISD – multiple instruction, single data
  - no such machine (although some people put vector machines in this category)
- MIMD – multiple instructions, multiple data streams
  - aka multiprocessors (SMPs, MPPs, clusters, NOWs)

Performance beyond single thread ILP

- There can be much higher natural parallelism in some applications (e.g., database or scientific codes)
- Explicit Thread Level Parallelism or Data Level Parallelism
- Thread: process with own instructions and data
    - Thread may be a subpart of a parallel program (“thread”), or it may be an independent program (“process”)
    - Each thread has all the state (instructions, data, PC, register state, and so on) necessary to allow it to execute
    - Many kitchens, each with own boss and chef
- Data Level Parallelism: Perform identical operations on data, and lots of data
    - 1 kitchen, 1 boss, many chefs

“Granularity”

- Granularity: Size of chunks of work (“grains”) given to each processor/core

Continuum of Granularity

- “Coarse”
  - Each processor is more powerful
  - Usually fewer processors
  - Communication is more expensive between processors
  - Processors are more loosely coupled
  - Tend toward MIMD
- “Fine”
  - Each processor is less powerful
  - Usually more processors
  - Communication is cheaper between processors
  - Processors are more tightly coupled
  - Tend toward SIMD

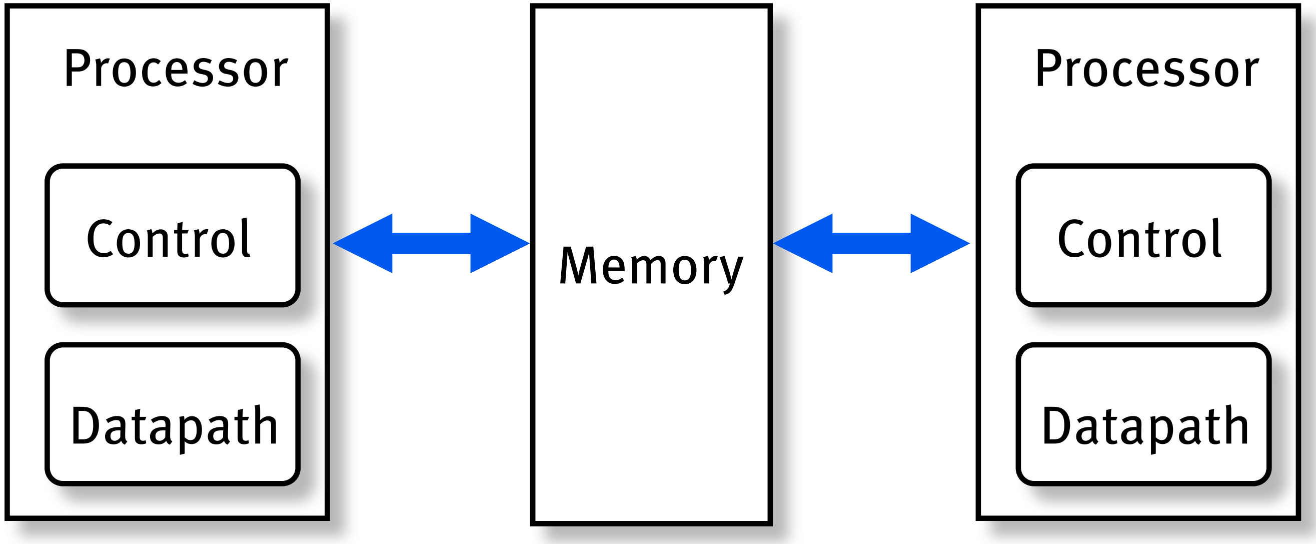
The Rest of the Class

- Next 3 weeks:
  - Thread-level parallelism. Coarse-grained parallelism. Multiprocessors, clusters, multicore.
- 3 weeks hence:
  - Data-level parallelism. Fine-grained parallelism. MMX, SSE. Vector & stream processors. GPUs.

Thread Level Parallelism

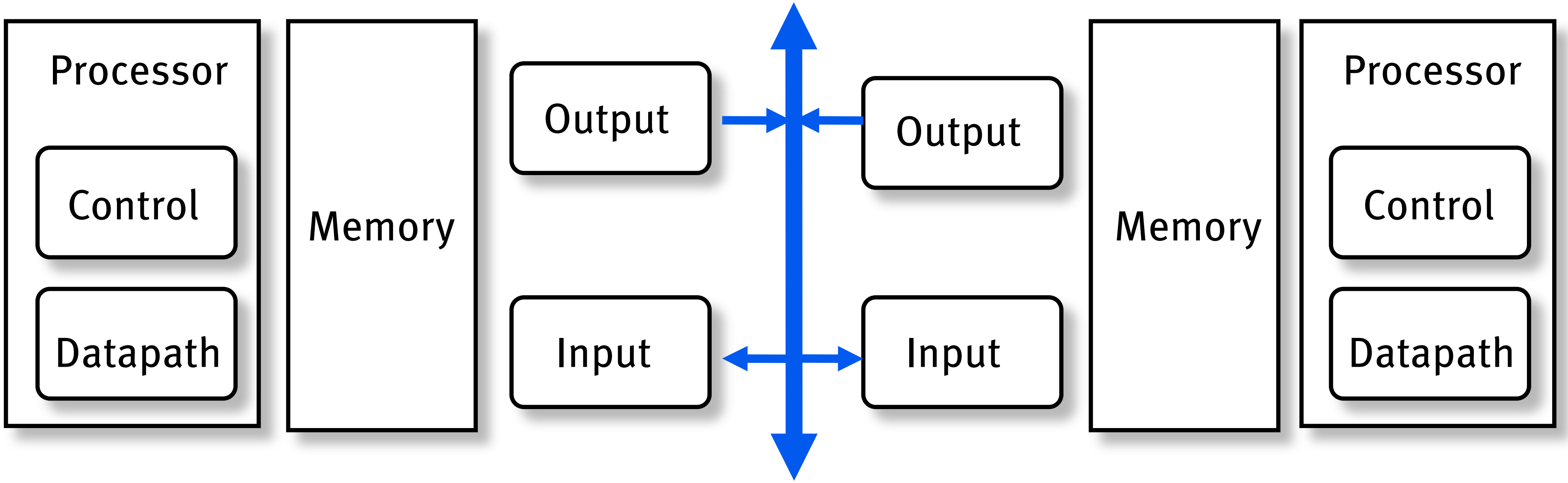
- ILP exploits ***implicit*** parallel operations within a loop or straight-line code segment
- TLP ***explicitly*** represented by the use of multiple threads of execution that are inherently parallel
  - ***You must rewrite your code to be thread-parallel:***
    - ***Divide work up into different chunks***
    - ***Communicate between chunks***
- Goal: Use multiple instruction streams to improve …
  - … throughput of computers that run many programs
  - … execution time of multi-threaded programs
- TLP could be more cost-effective to exploit than ILP

Organizing Many Processors



- Multiprocessor—multiple processors with a single shared address space
  - Symmetric multiprocessors: All memory is the same distance away from all processors (UMA = uniform memory access)
    - NUMA = nonuniform memory access. We will use this term.
  - Multicore & dual-socket processors are here.

Organizing Many Processors

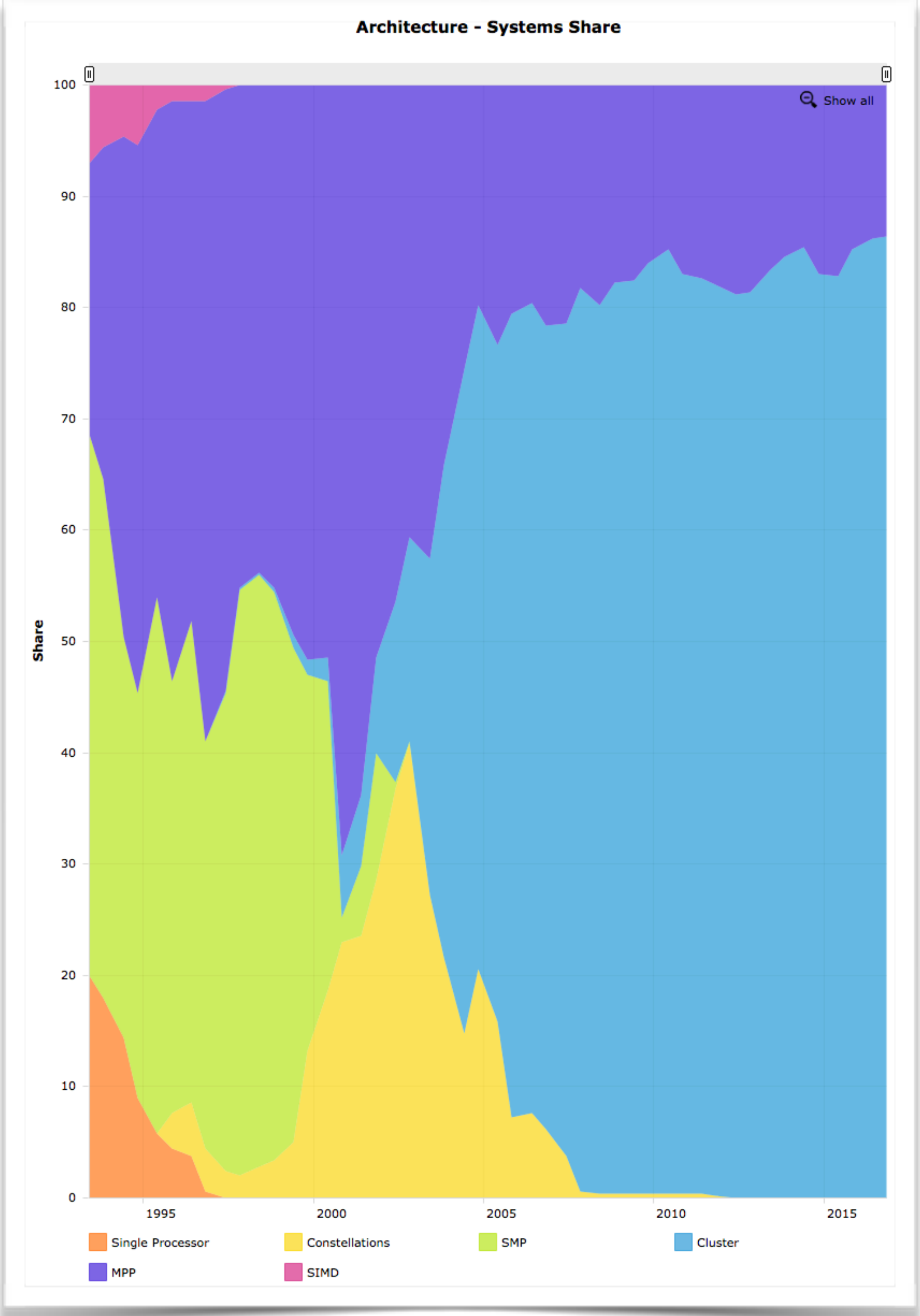


- Cluster—multiple computers (each with their own address space) connected over a local area network (LAN) functioning as a single system
  - “Constellation”: cluster of multiprocessors

Applications Needing “Supercomputing”

- Energy [plasma physics (simulating fusion reactions), geophysical (petroleum) exploration]
- DoE stockpile stewardship (to ensure the safety and reliability of the nation’s stockpile of nuclear weapons)
- Earth and climate (climate and weather prediction, earthquake, tsunami prediction and mitigation of risks)
- Transportation (improving vehicles’ airflow dynamics, fuel consumption, crashworthiness, noise reduction)
- Bioinformatics and computational biology (genomics, protein folding, designer drugs)
- Societal health and safety (pollution reduction, disaster planning, terrorist action detection)
- Financial (calculate options pricing, etc.)

Supercomputer Style Migration (Top500)



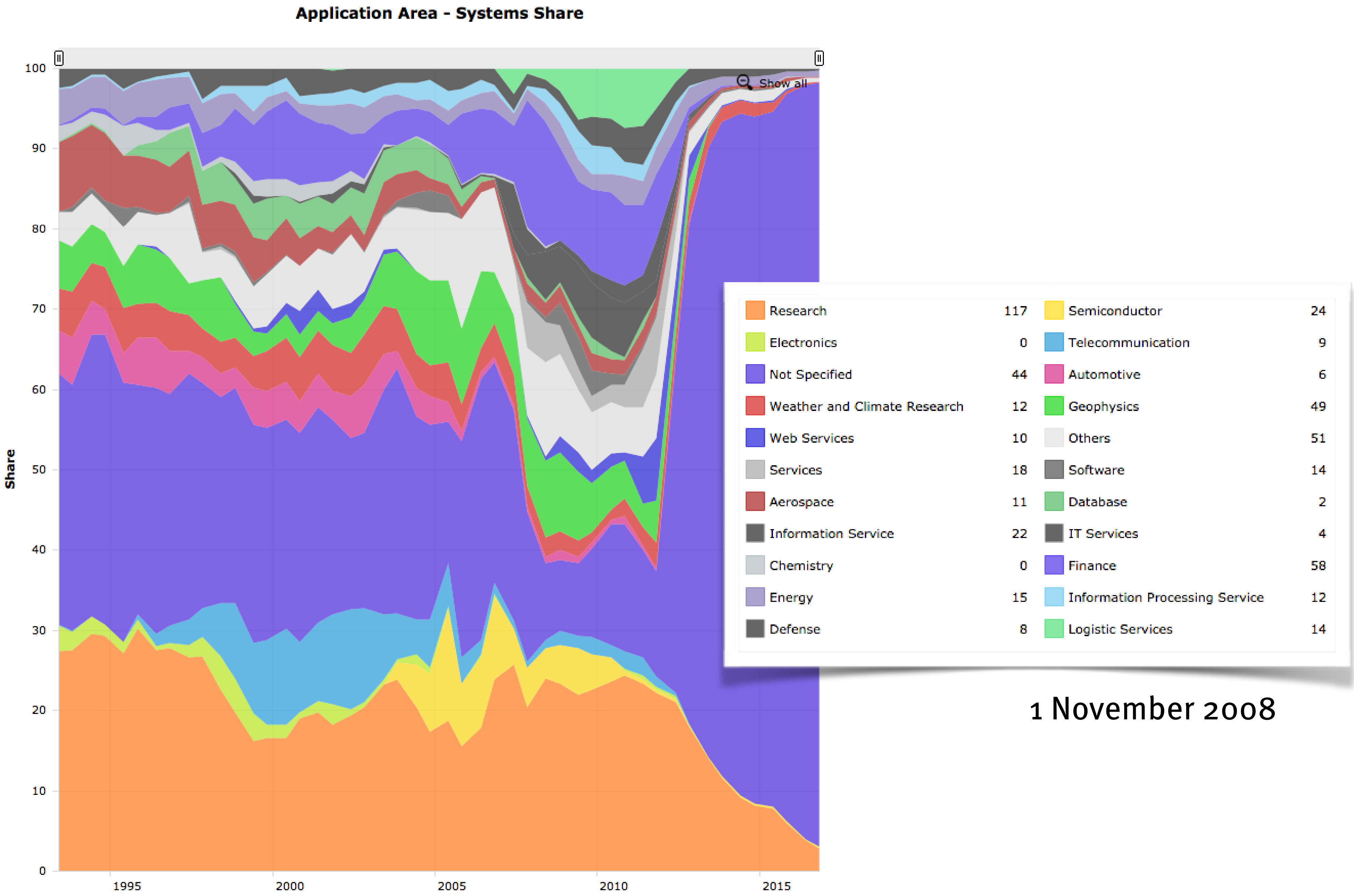
Cluster—whole computers interconnected using their I/O bus

Constellation—a cluster that uses an SMP multiprocessor as the building block

- In the last decade+ uniprocessor and SIMDs disappeared while clusters grew to 80%

http://www.top500.org/statistics/overtime/

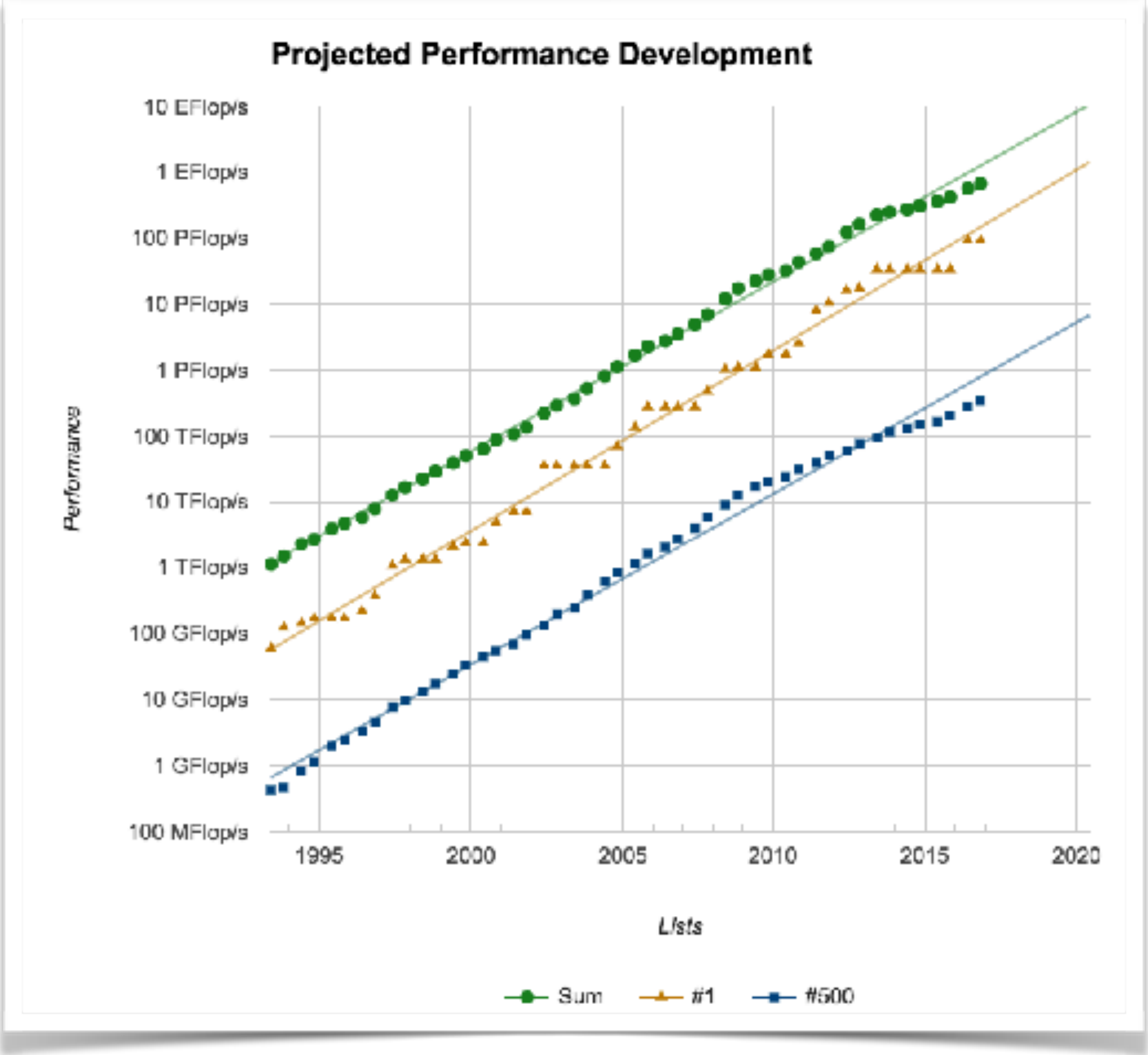
Top 500: Application Area



Research	117	Semiconductor	24
Electronics	0	Telecommunication	9
Not Specified	44	Automotive	6
Weather and Climate Research	12	Geophysics	49
Web Services	10	Others	51
Services	18	Software	14
Aerospace	11	Database	2
Information Service	22	IT Services	4
Chemistry	0	Finance	58
Energy	15	Information Processing Service	12
Defense	8	Logistic Services	14

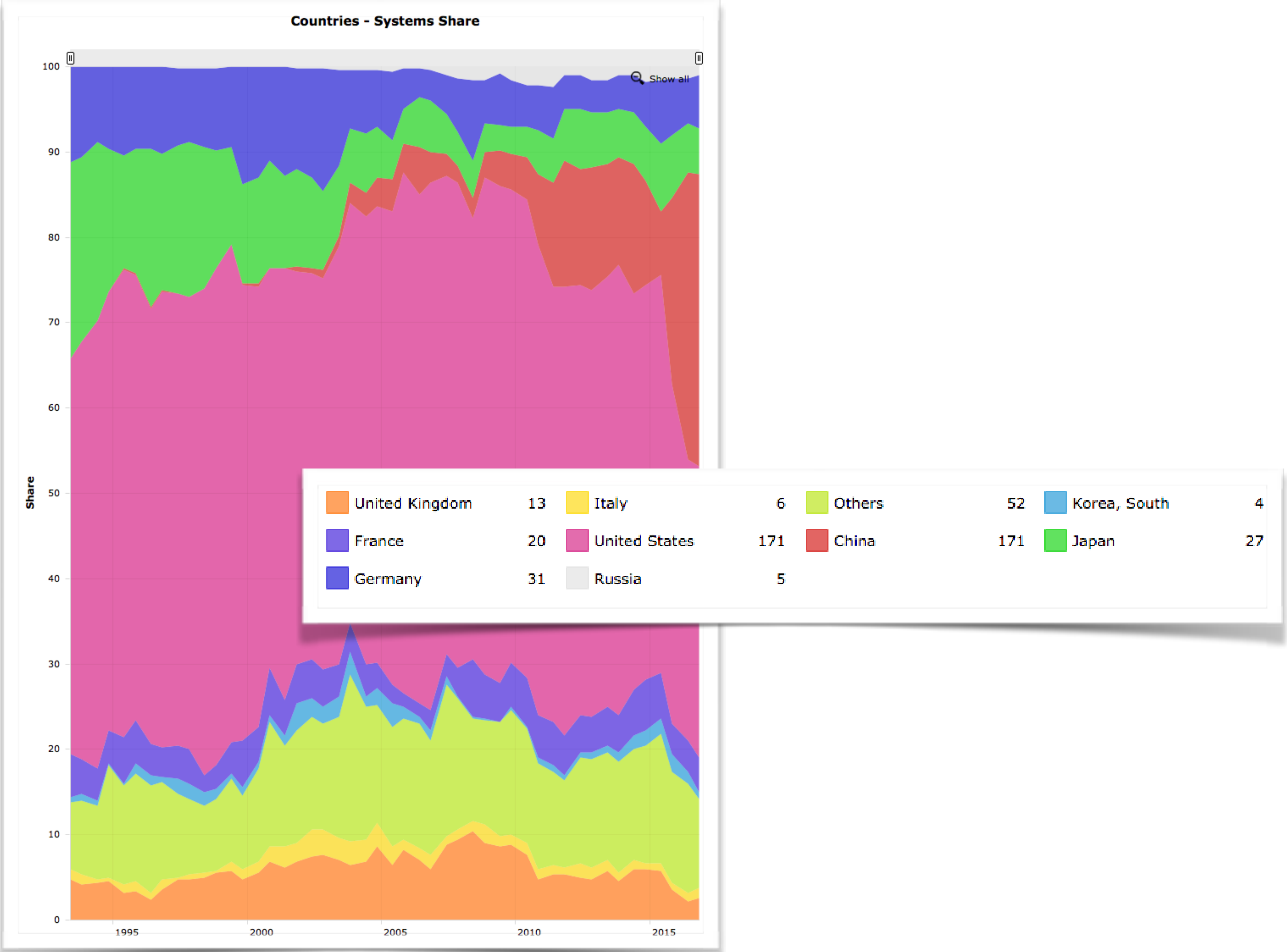
1 November 2008

Top 500: Historicals



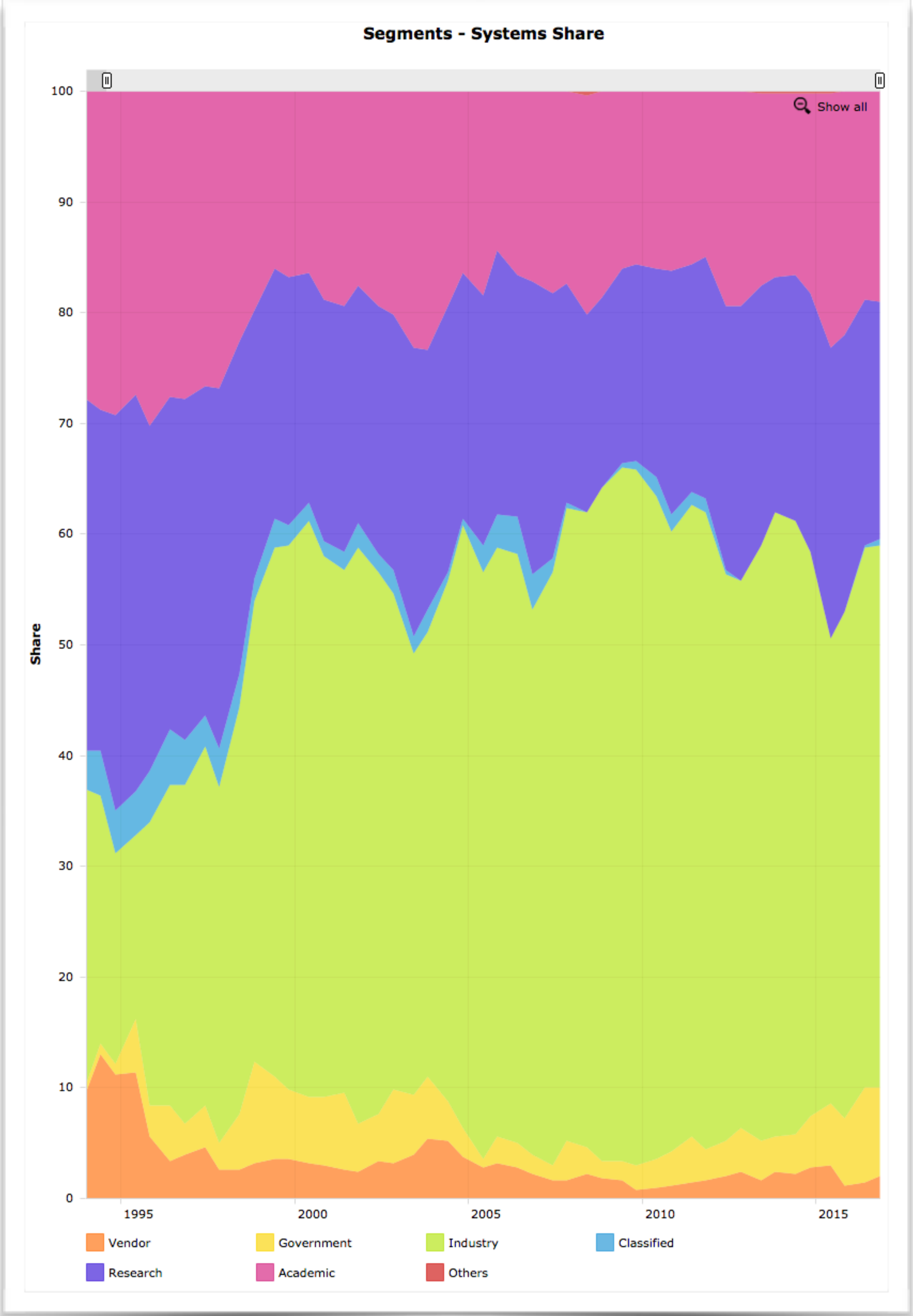
http://www.top500.org/statistics/perfdevel/

Top 500: Countries

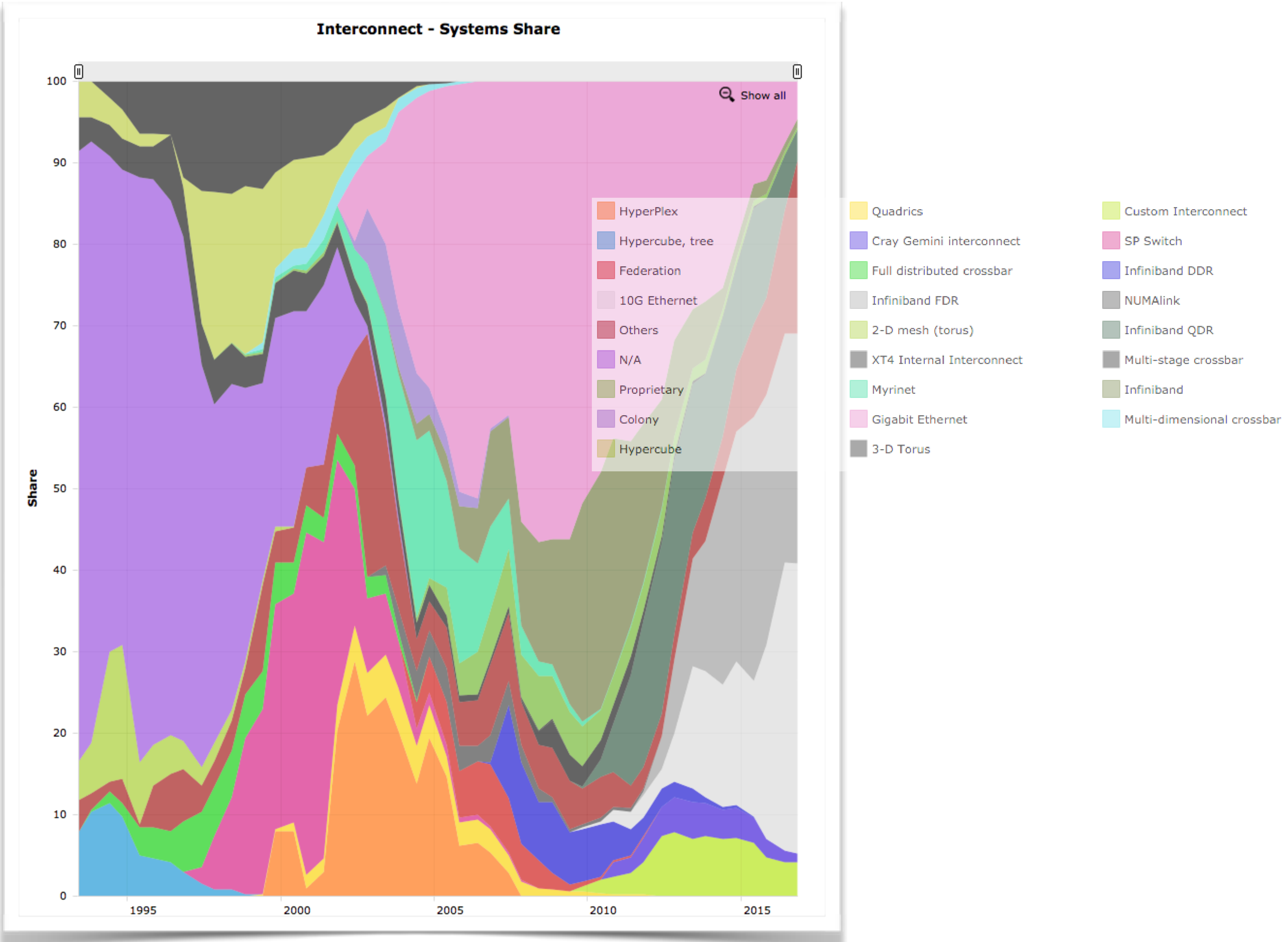


🟧 United Kingdom	13	🟨 Italy	6	🟩 Others	52	🟦 Korea, South	4
🟪 France	20	🟪 United States	171	🟥 China	171	🟩 Japan	27
🟦 Germany	31	⬜ Russia	5				

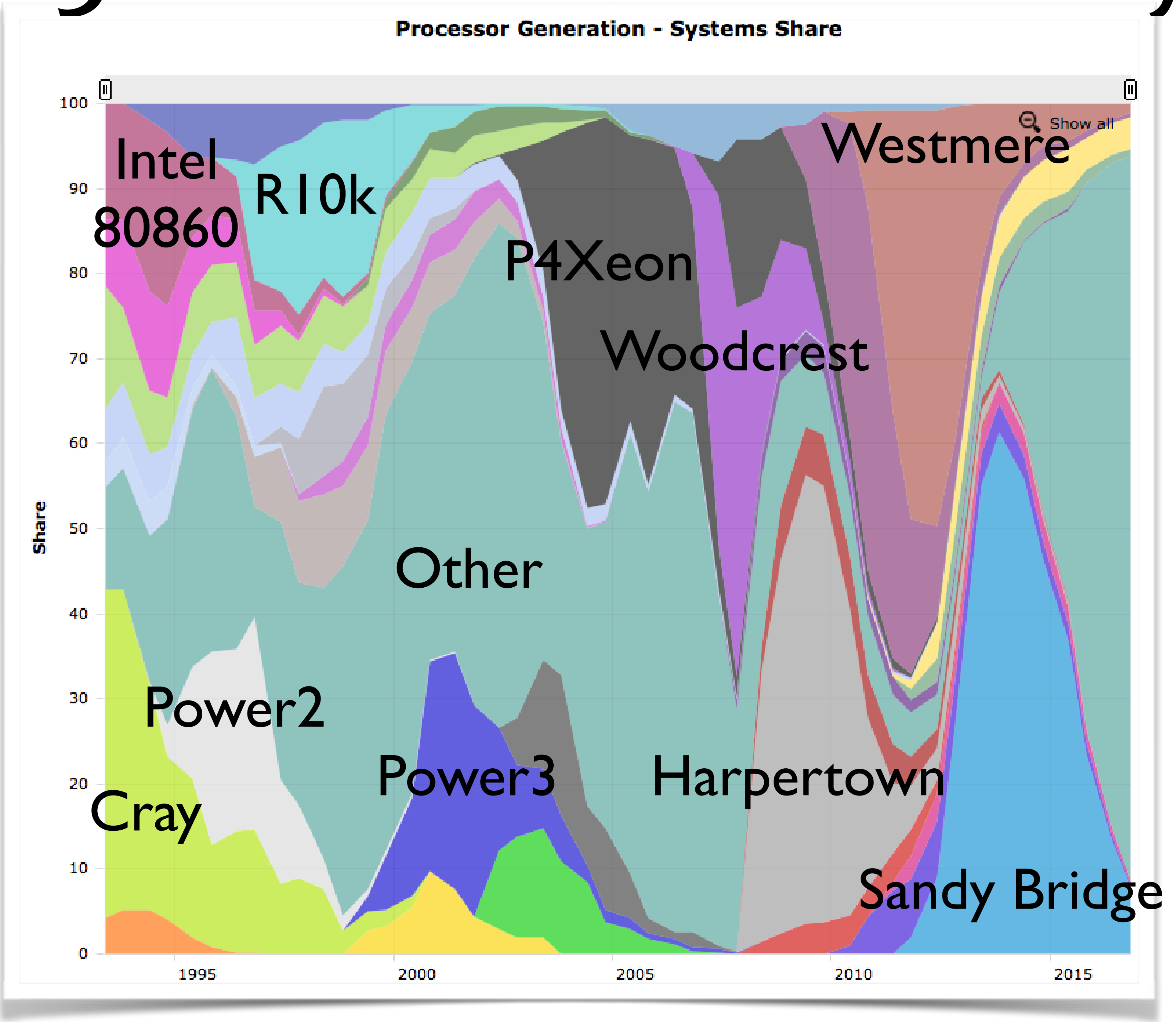
Top 500: Customers



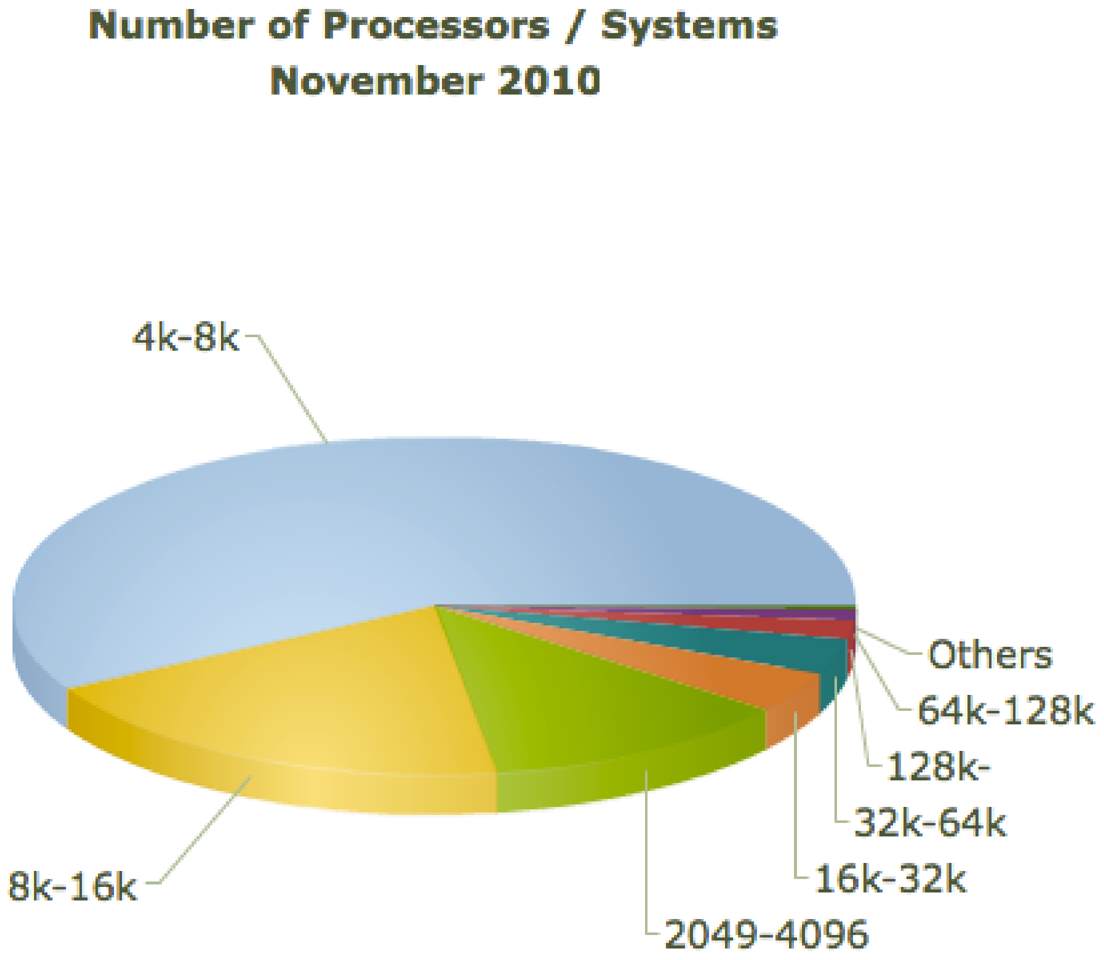
Top 500: Interconnect



Top 500: Processor Family

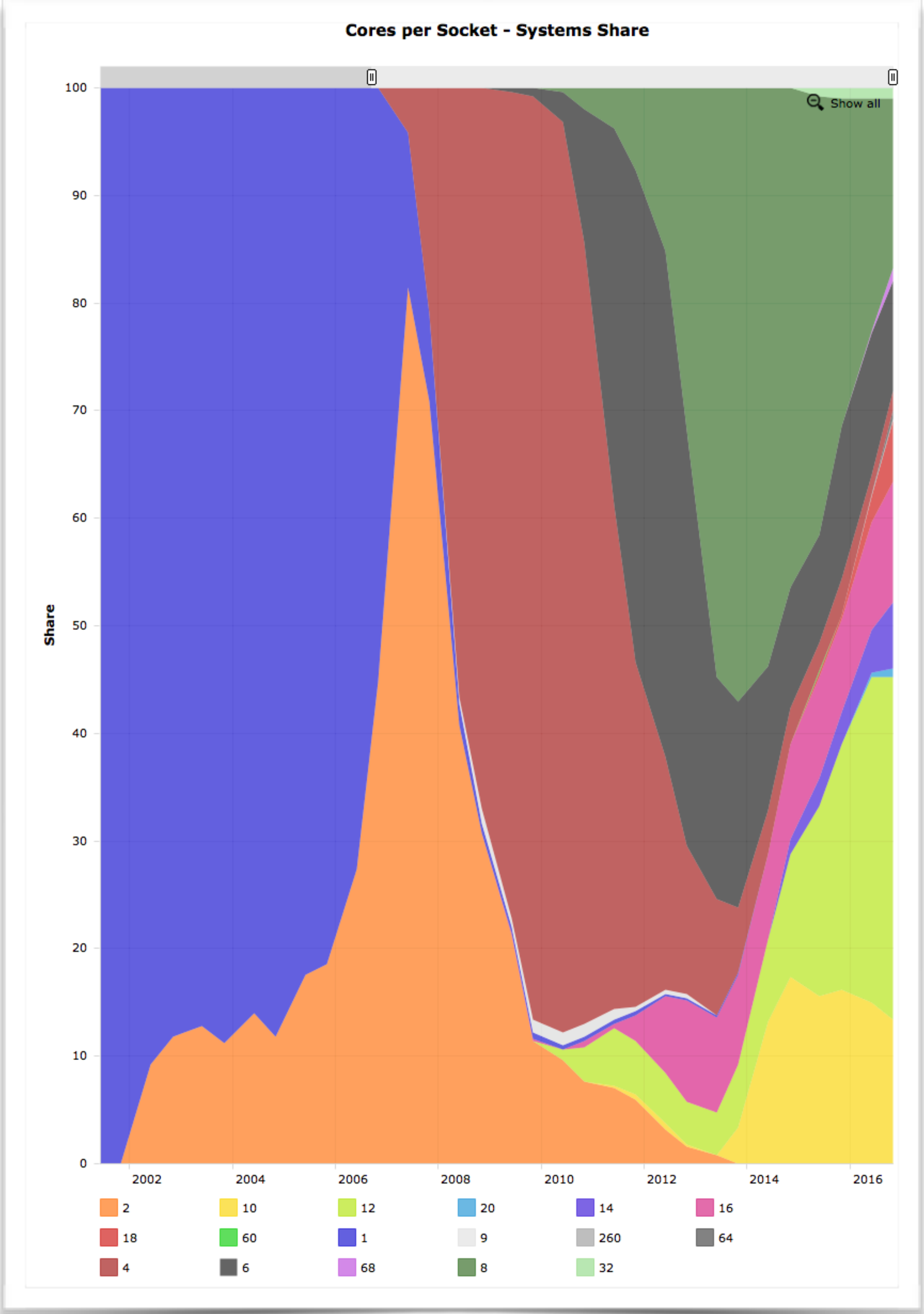


Top 500: Processor Count

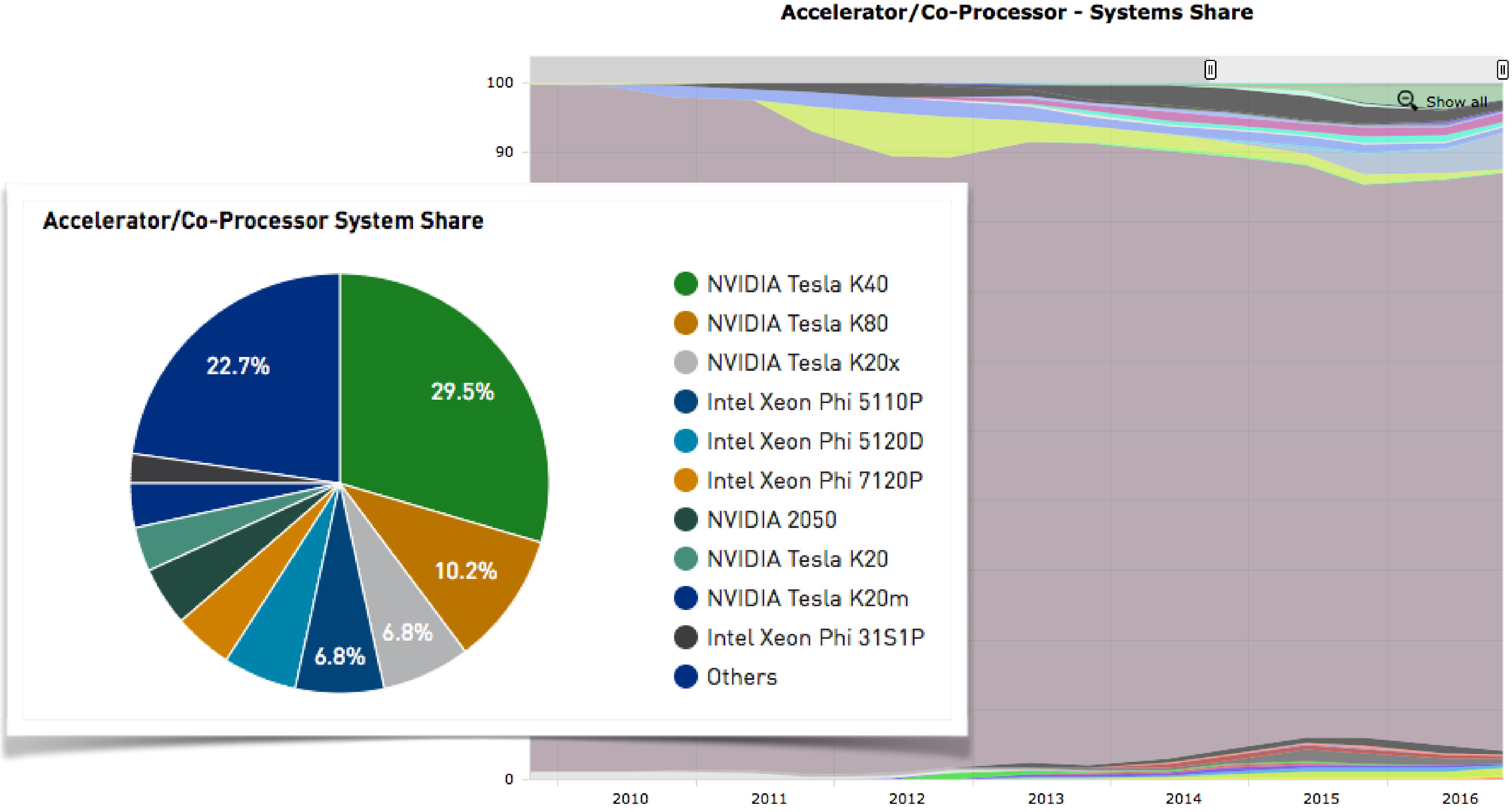


http://www.top500.org/charts/list/36/procclass

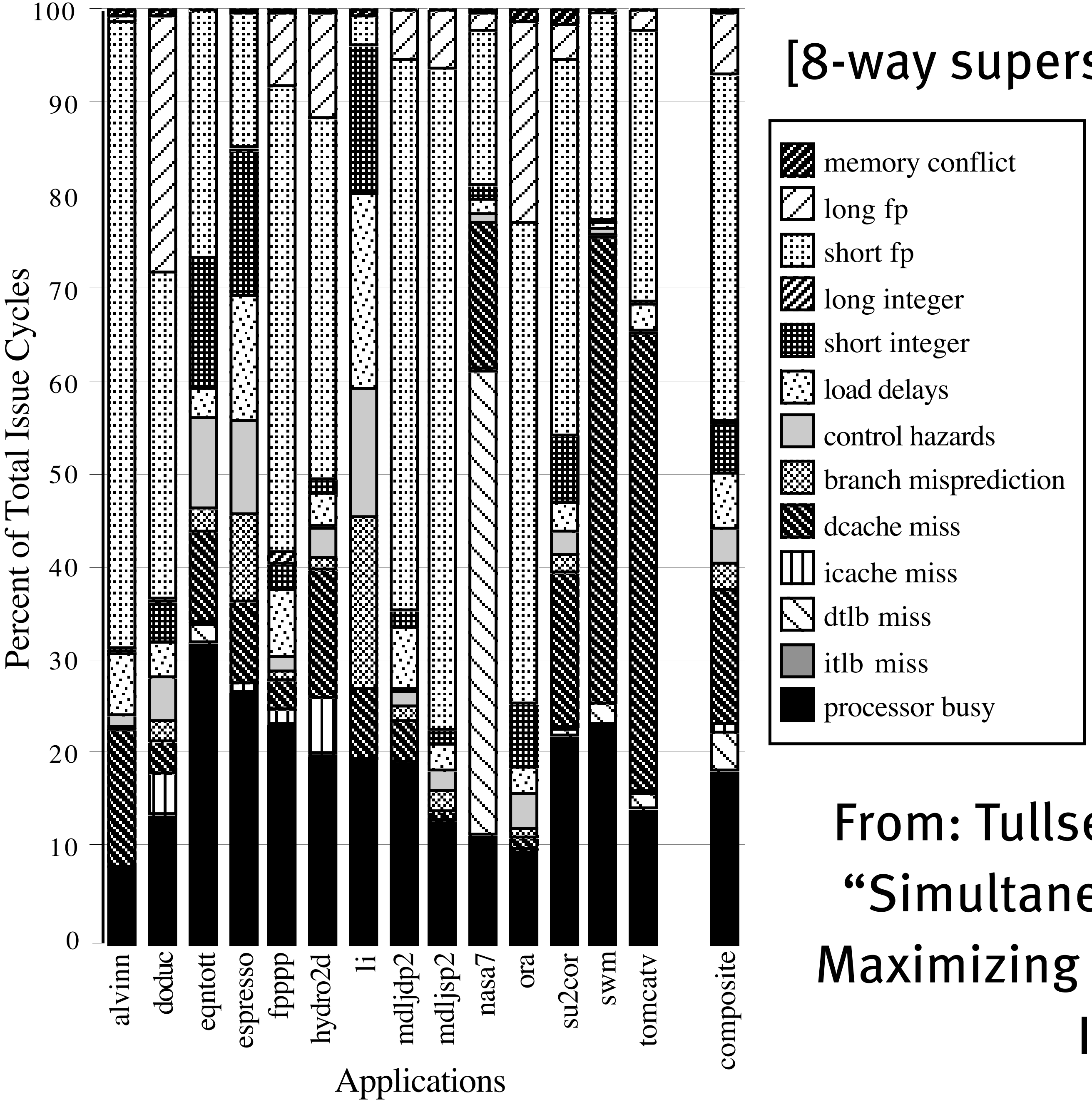
Top 500: Cores/Socket



Top 500: Accelerators



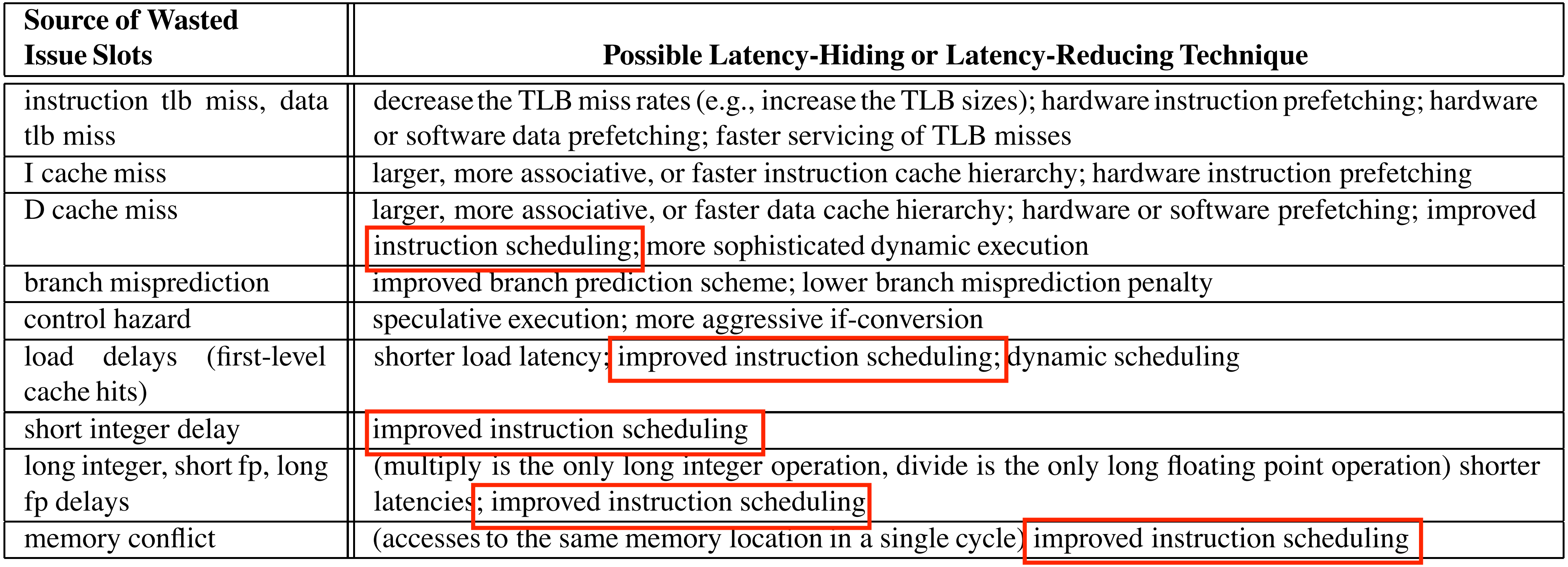
For most apps, most execution units lie idle



[8-way superscalar]

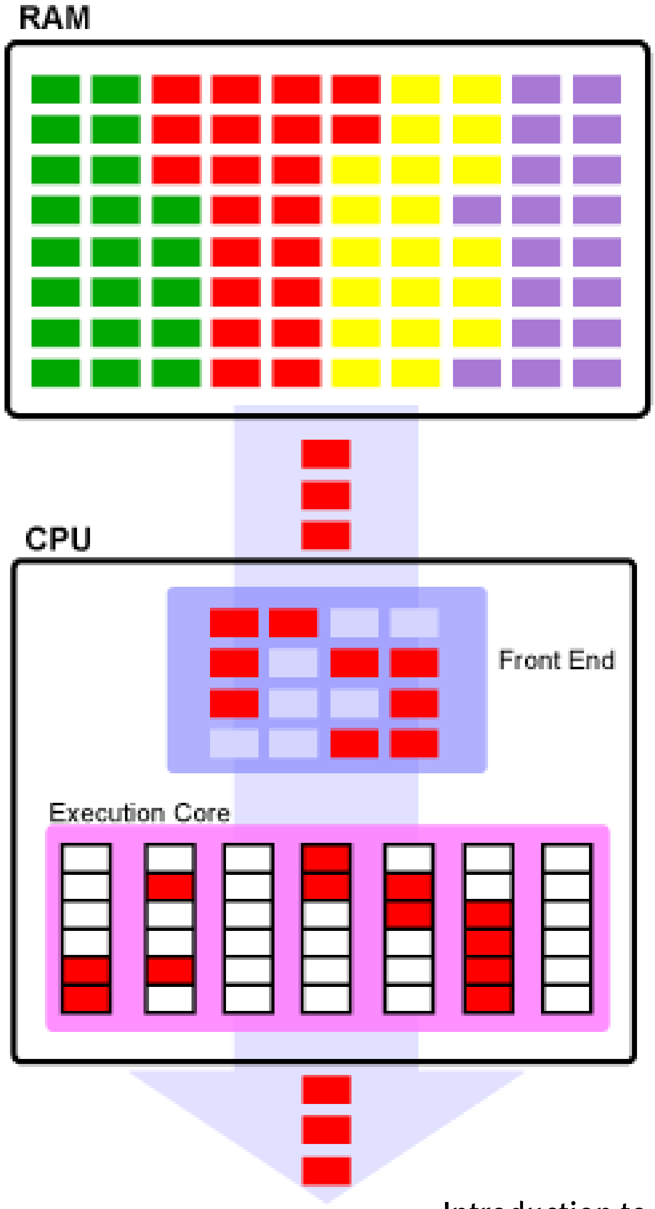
From: Tullsen, Eggers, and Levy, “Simultaneous Multithreading: Maximizing On-chip Parallelism”, ISCA 1995.

Source of Wasted Slots



Source of Wasted Issue Slots	**Possible Latency-Hiding or Latency-Reducing Technique**
instruction tlb miss, data tlb miss	decrease the TLB miss rates (e.g., increase the TLB sizes); hardware instruction prefetching; hardware or software data prefetching; faster servicing of TLB misses
I cache miss	larger, more associative, or faster instruction cache hierarchy; hardware instruction prefetching
D cache miss	larger, more associative, or faster data cache hierarchy; hardware or software prefetching; improved instruction scheduling; more sophisticated dynamic execution
branch misprediction	improved branch prediction scheme; lower branch misprediction penalty
control hazard	speculative execution; more aggressive if-conversion
load delays (first-level cache hits)	shorter load latency; improved instruction scheduling; dynamic scheduling
short integer delay	improved instruction scheduling
long integer, short fp, long fp delays	(multiply is the only long integer operation, divide is the only long floating point operation) shorter latencies; improved instruction scheduling
memory conflict	(accesses to the same memory location in a single cycle) improved instruction scheduling

Single-threaded CPU

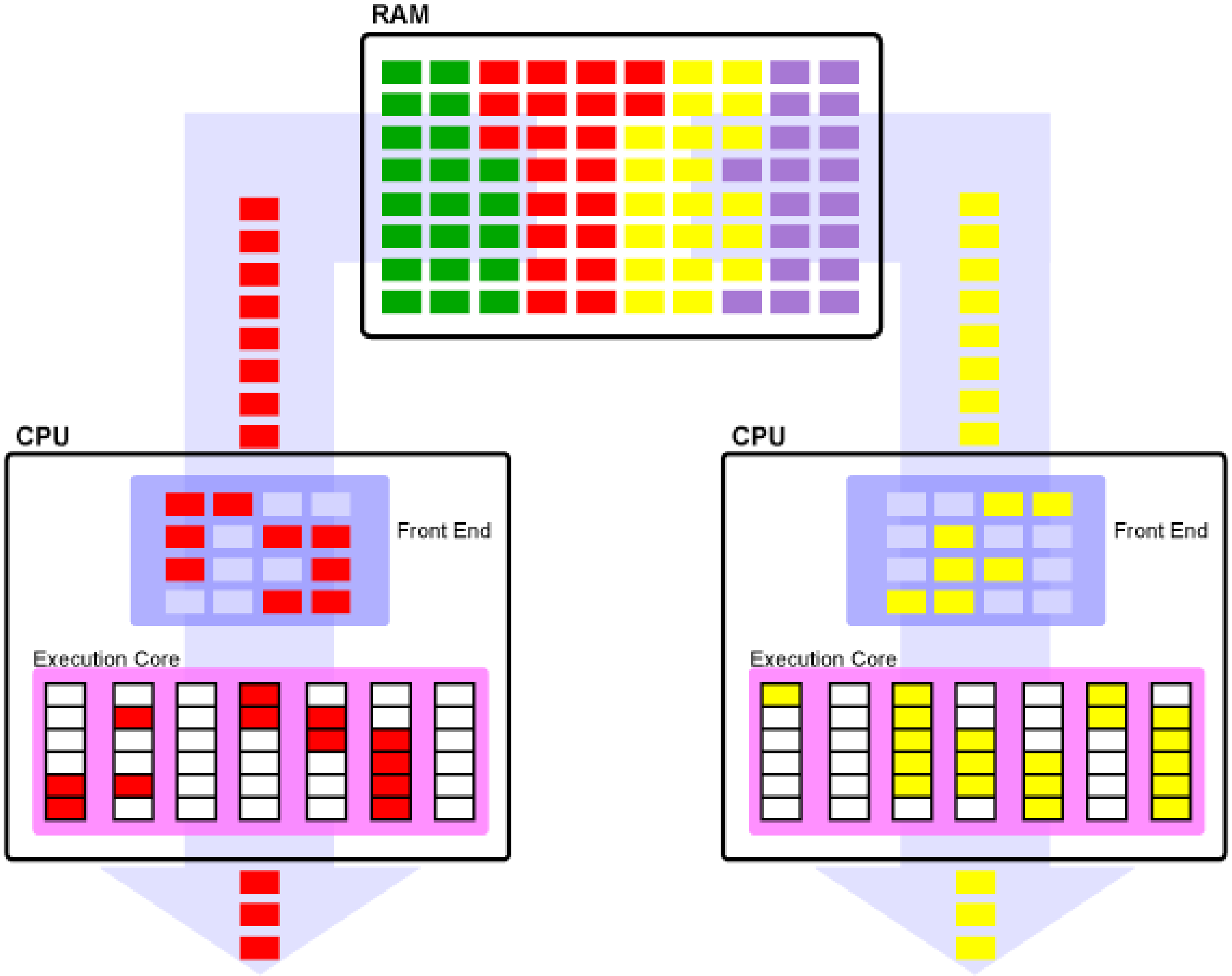


Introduction to Multithreading, Superthreading and Hyperthreading
By Jon Stokes
http://arstechnica.com/articles/paedia/cpu/hyperthreading.ars

We can add more CPUs …

- … and we’ll talk about this later in the class
- Note we have multiple CPUs reading out of the same instruction store
- If we define “efficient” as “filling a higher fraction of issue slots”:
  - Is this more efficient than having one CPU?

Symmetric Multiprocessing



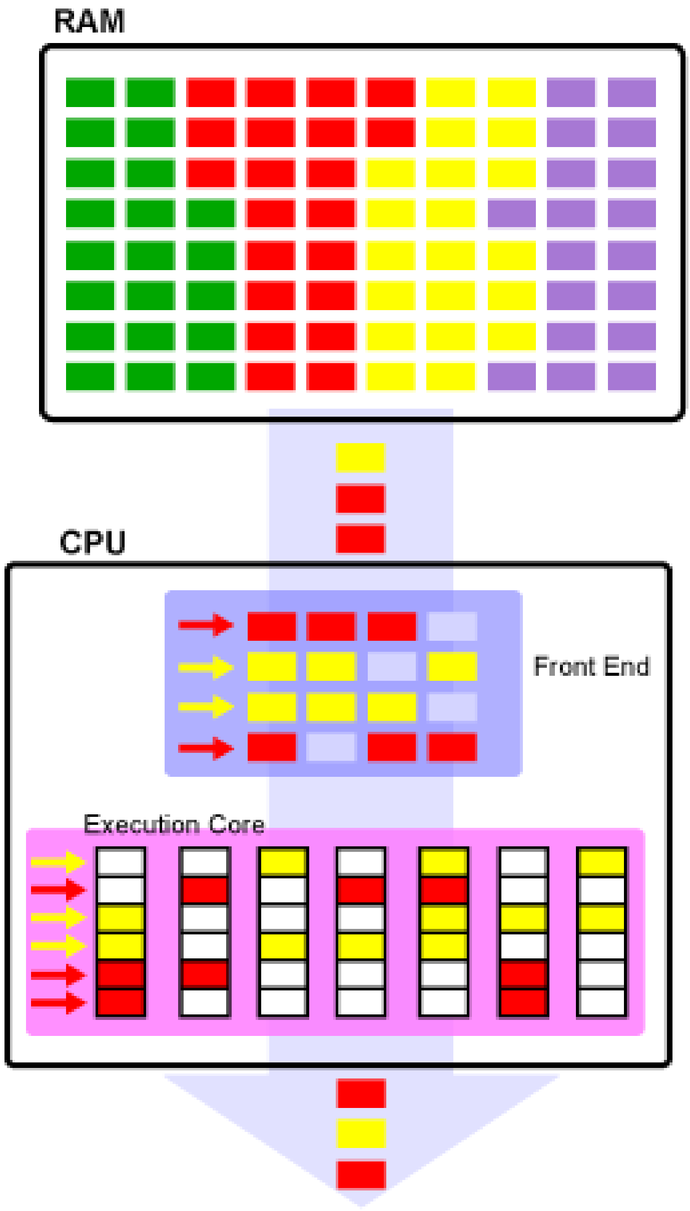
Conventional Multithreading

- How does a microprocessor run multiple processes / threads “at the same time”?
    - How does one program interact with another program?
- What is preemptive multitasking vs. cooperative multitasking?

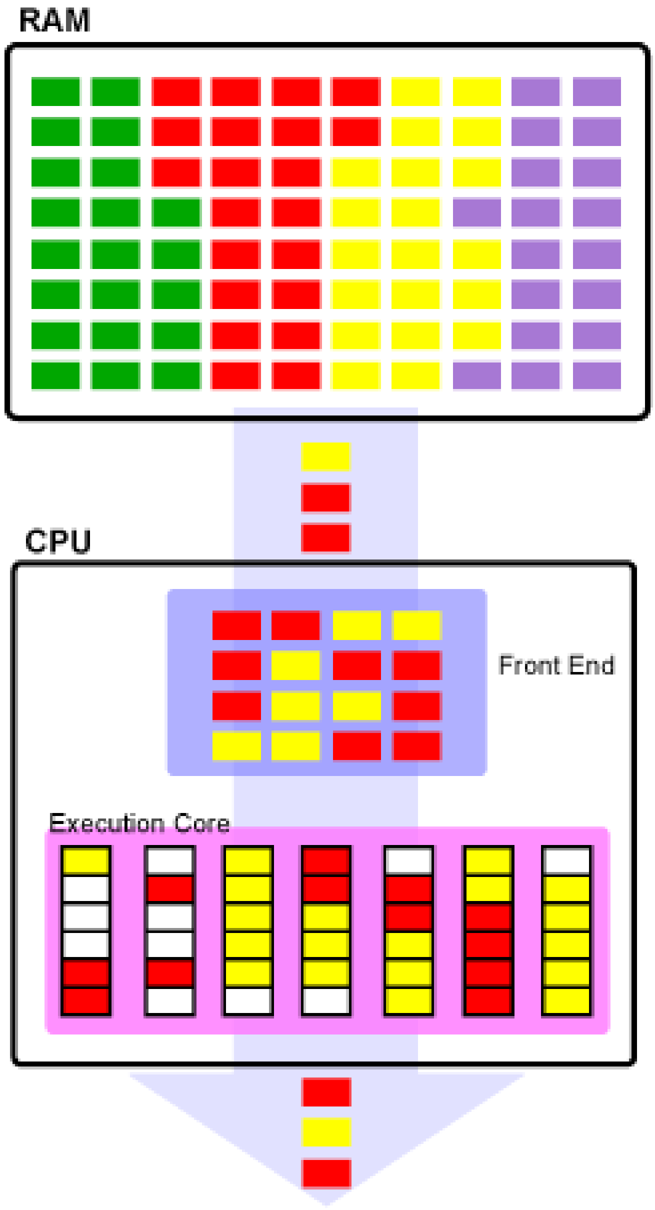
New Approach: Multithreaded Execution

- Multithreading: multiple threads to share the functional units of 1 processor via overlapping
  - processor must duplicate independent state of each thread e.g., a separate copy of register file, a separate PC, and for running independent programs, a separate page table
  - memory shared through the virtual memory mechanisms, which already support multiple processes
  - HW for fast thread switch; much faster than full process switch ≈ 100s to 1000s of clocks

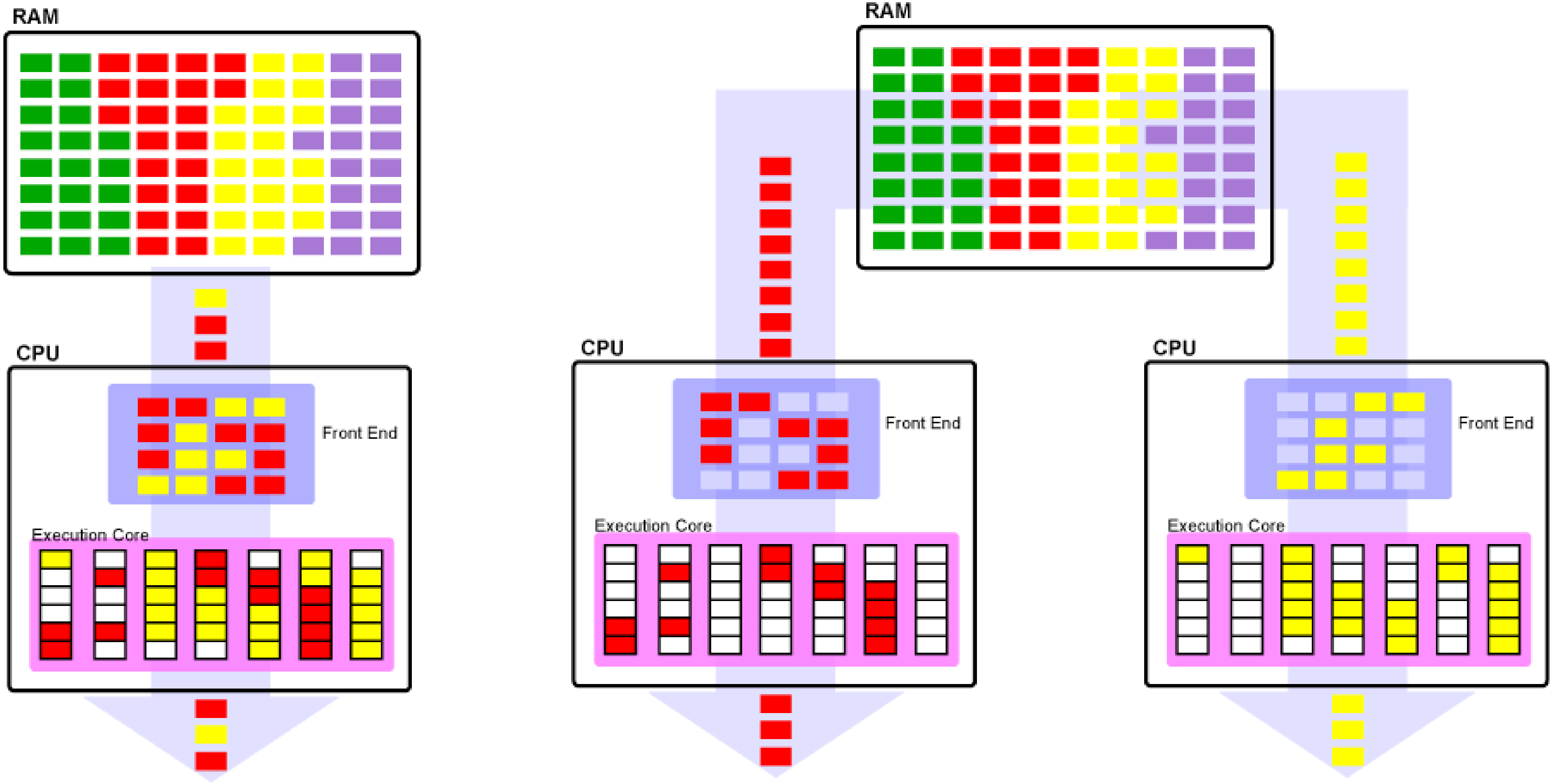
Superthreading



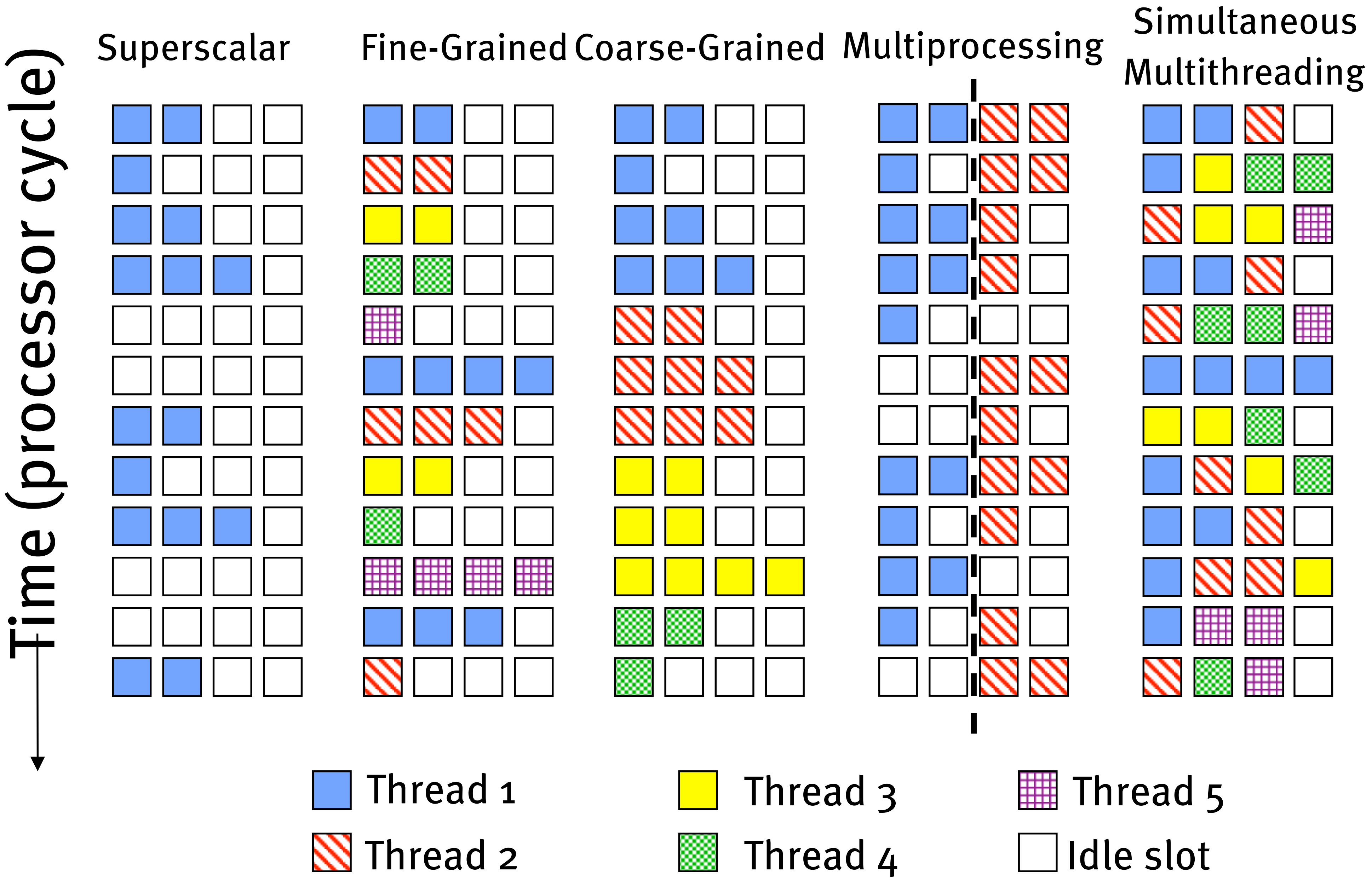
Simultaneous multithreading (SMT)



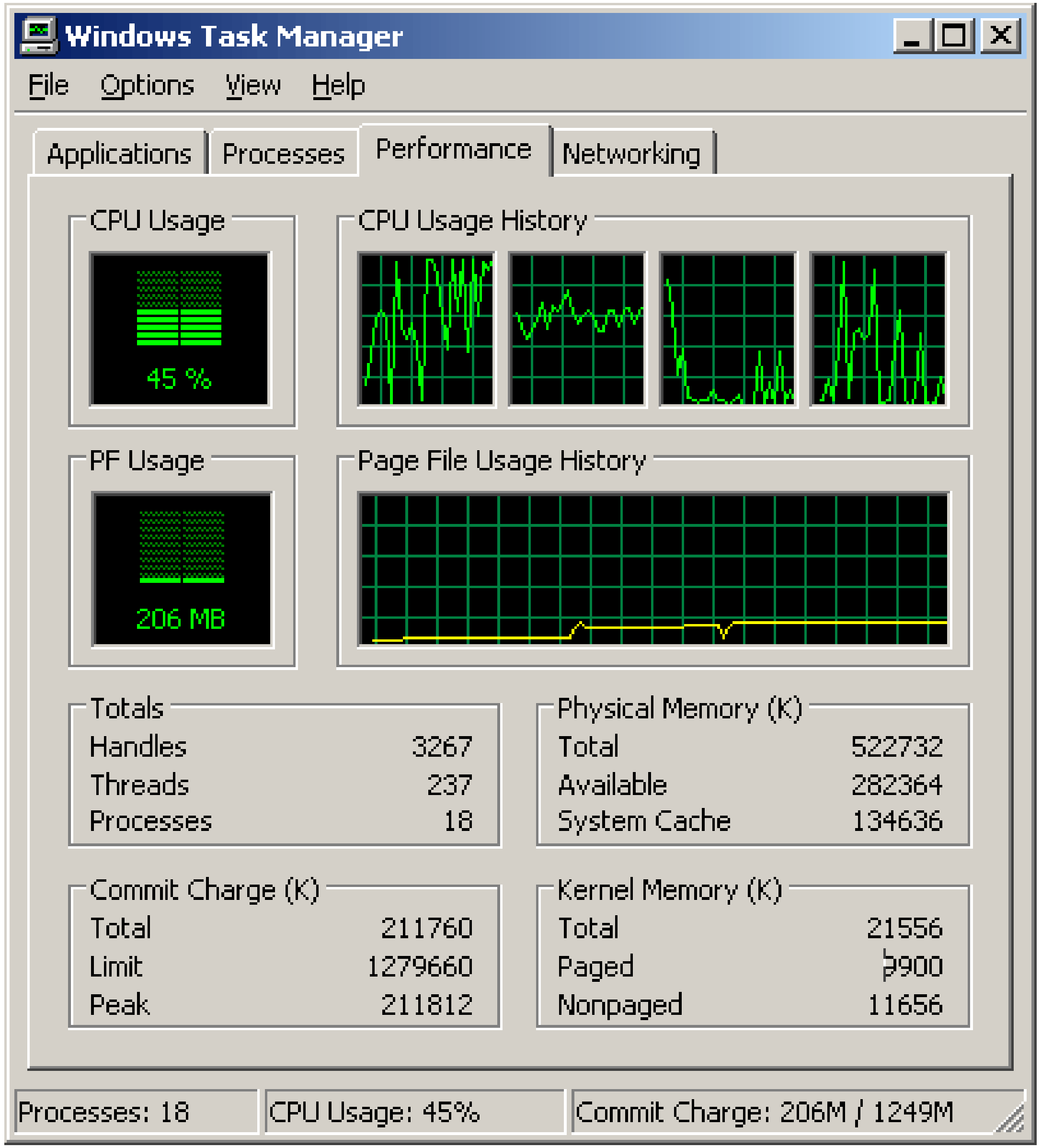
Simultaneous multithreading (SMT)



Multithreaded Categories



“Hyperthreading”



Totals	
Handles	3267
Threads	237
Processes	18

Physical Memory (K)	
Total	522732
Available	282364
System Cache	134636

Commit Charge (K)	
Total	211760
Limit	1279660
Peak	211812

Kernel Memory (K)	
Total	21556
Paged	9900
Nonpaged	11656

Processes: 18 CPU Usage: 45% Commit Charge: 206M / 1249M

http://www.2cpu.com/Hardware/ht_analysis/images/taskmanager.html

Multithreaded Execution

- When do we switch between threads?
  - Alternate instruction per thread (fine grain)
  - When a thread is stalled, perhaps for a cache miss, another thread can be executed (coarse grain)

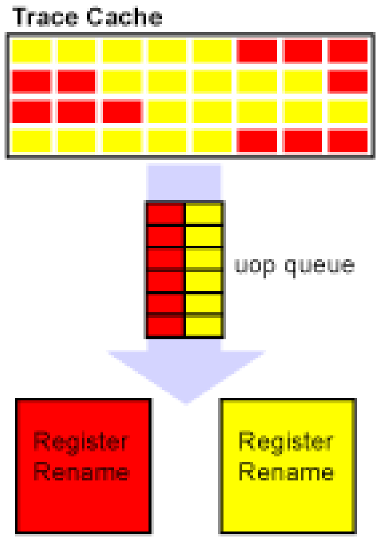
Fine-Grained Multithreading

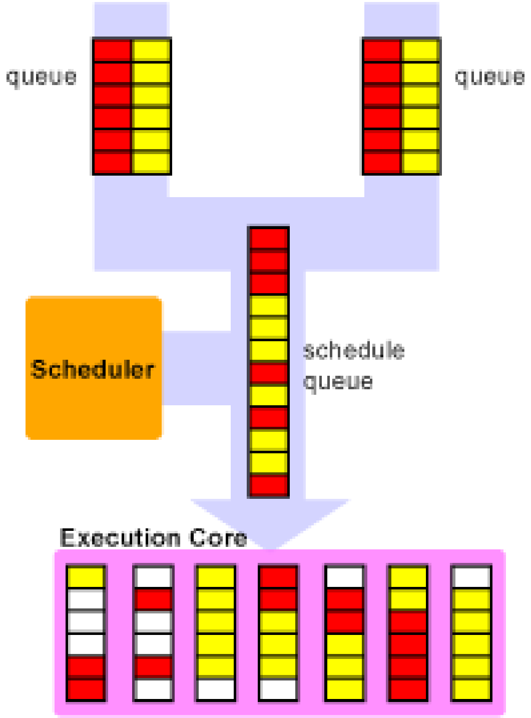
- Switches between threads on each instruction, causing the execution of multiple threads to be interleaved
- Usually done in a round-robin fashion, skipping any stalled threads
- CPU must be able to switch threads every clock
- Advantage is it can hide both short and long stalls, since instructions from other threads executed when one thread stalls
- Disadvantage is it slows down execution of individual threads, since a thread ready to execute without stalls will be delayed by instructions from other threads
- Used on Sun’s Niagara (will see later)

Coarse-Grained Multithreading

- Switches threads only on costly stalls, such as L2 cache misses
- Advantages
  - Relieves need to have very fast thread-switching
  - Doesn’t slow down thread, since instructions from other threads issued only when the thread encounters a costly stall
- Disadvantage is hard to overcome throughput losses from shorter stalls, due to pipeline start-up costs
  - Since CPU issues instructions from 1 thread, when a stall occurs, the pipeline must be emptied or frozen
  - New thread must fill pipeline before instructions can complete
- Because of this start-up overhead, coarse-grained multithreading is better for reducing penalty of high cost stalls, where pipeline refill ‹‹ stall time
- Used in IBM AS/400

Partitioning: Static vs. Dynamic





P4Xeon Microarchitecture

- Replicated
  - Register renaming logic
  - Instruction pointer, other architectural registers
  - ITLB
  - Return stack predictor
- Partitioned
  - Reorder buffers
  - Load/store buffers
  - Various queues: scheduling, uop, etc.
- Shared
  - Caches (trace, L1/L2/L3)
  - Microarchitectural registers
  - Execution units
- If configured as single-threaded, all resources go to one thread

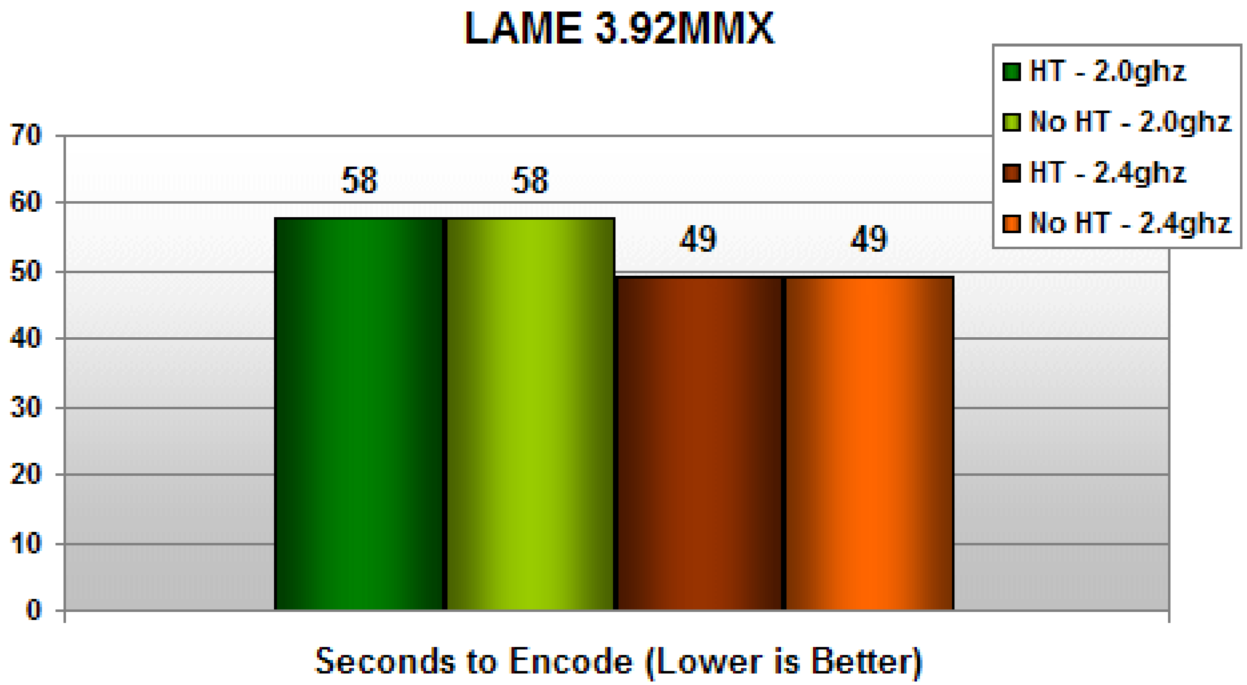
Design Challenges in SMT

- Since SMT makes sense only with fine-grained implementation, impact of fine-grained scheduling on single thread performance?
  - A preferred thread approach sacrifices neither throughput nor single-thread performance?
  - Unfortunately, with a preferred thread, the processor is likely to sacrifice some throughput, when preferred thread stalls
- Larger register file needed to hold multiple contexts
- Not affecting clock cycle time, especially in
  - Instruction issue—more candidate instructions need to be considered
  - Instruction completion—choosing which instructions to commit may be challenging
- Ensuring that cache and TLB conflicts generated by SMT do not degrade performance

Problems with SMT

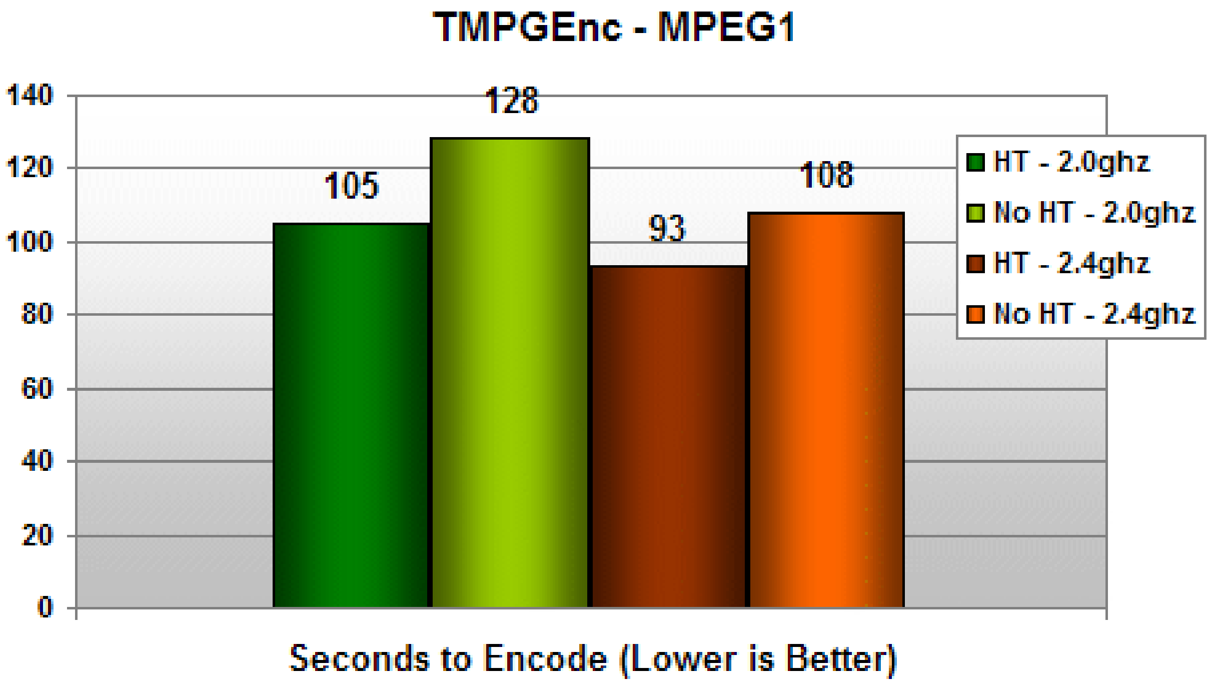
- One thread monopolizes resources
  - Example: One thread ties up FP unit with long-latency instruction, other thread tied up in scheduler
- Cache effects
  - Caches are unaware of SMT—can't make warring threads cooperate
  - If both warring threads access different memory and have cache conflicts, constant swapping

Hyperthreading Neutral!



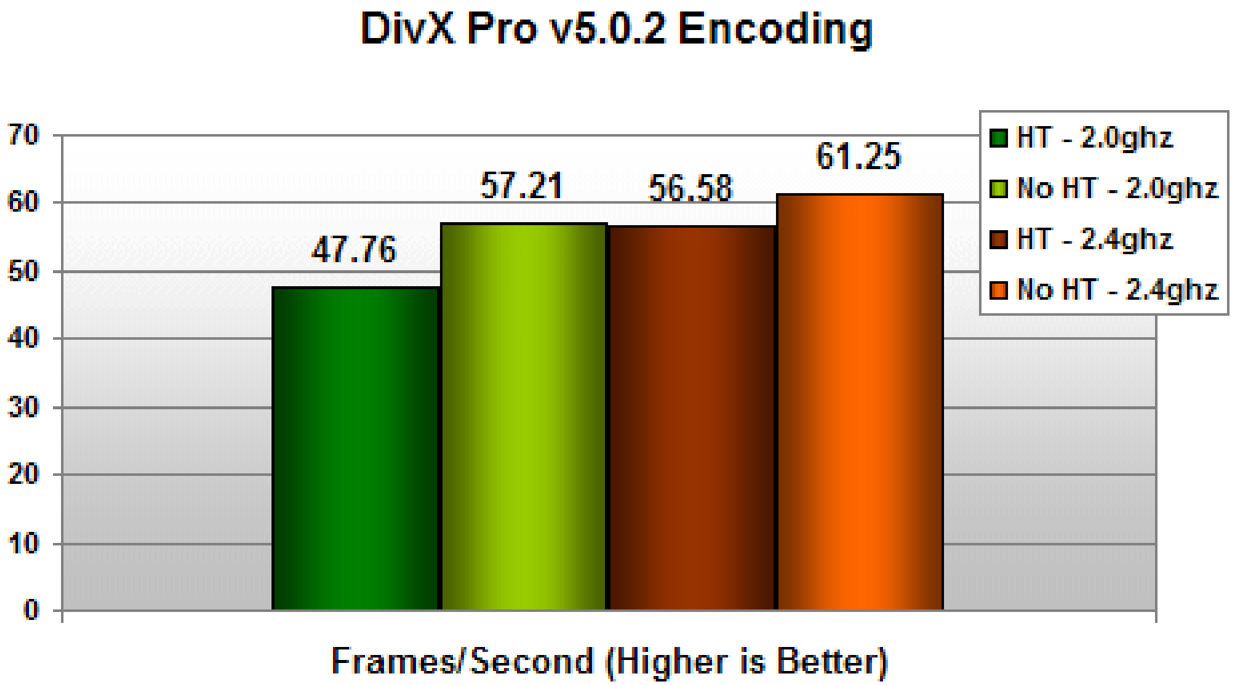
http://www.2cpu.com/articles/43_1.html

Hyperthreading Good!



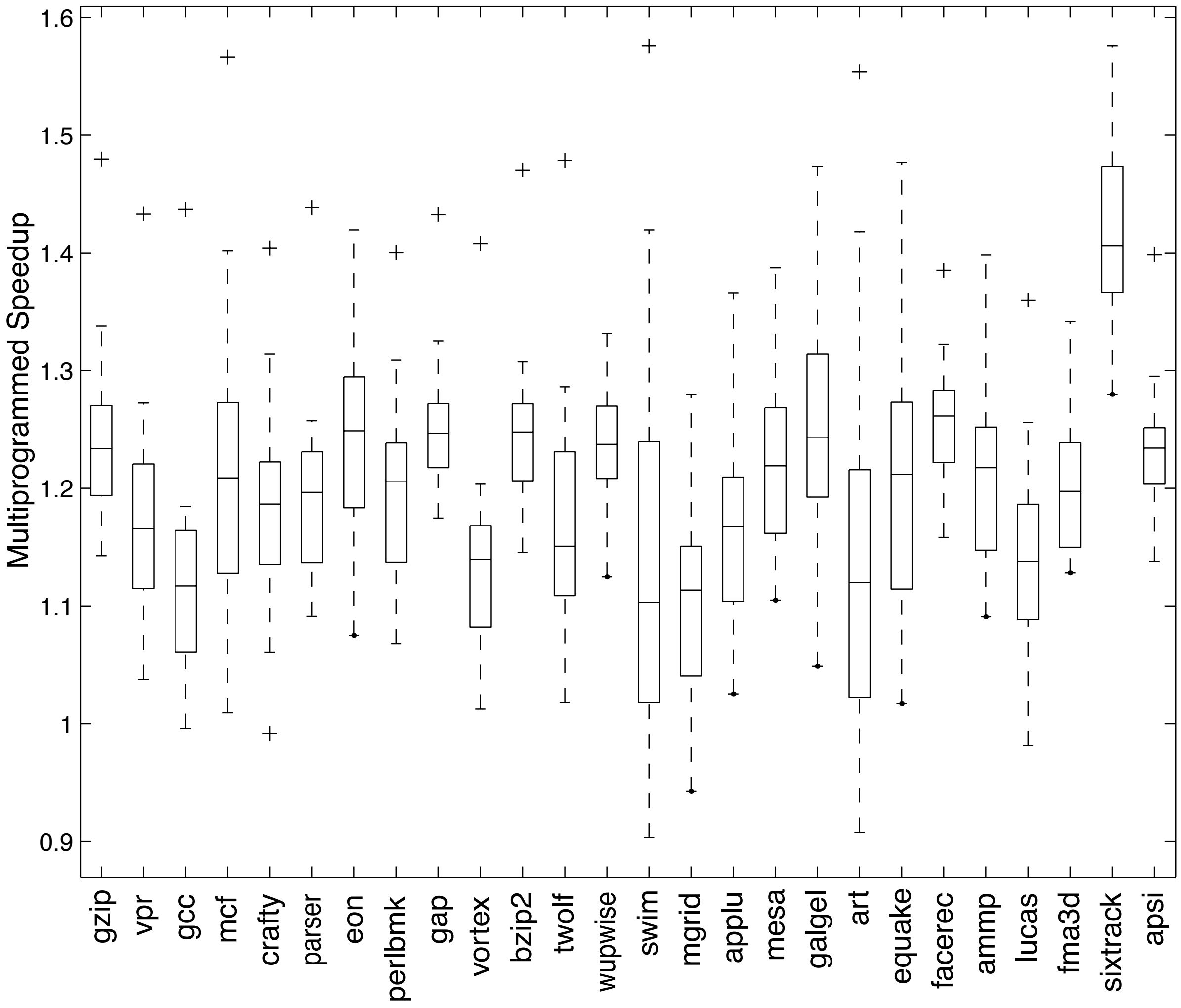
http://www.2cpu.com/articles/43_1.html

Hyperthreading Bad!



http://www.2cpu.com/articles/43_1.html

SPEC vs. SPEC (PACT '03)



- Avg. multithreaded speedup 1.20 (range 0.90–1.58)

"Initial Observations of the Simultaneous Multithreading Pentium 4 Processor", Nathan Tuck and Dean M. Tullsen (PACT '03)