18-600 Foundations of Computer Systems

Lecture 20: "Overview of Parallel Architectures"

John P. Shen & Zhiyi Yu

November 9, 2016

➢Recommended References:

- Chapter 11 of Shen and Lipasti.
- "Parallel Computer Organization and Design," by Michel Dubois, Murali Annavaram, Per Stenstrom, Chapters 5 and 7, 2012.

Electrical & Computer ENGINEERING

Carnegie Mellon University

18-600 Foundations of Computer Systems

Lecture 20: "Overview of Parallel Architectures"

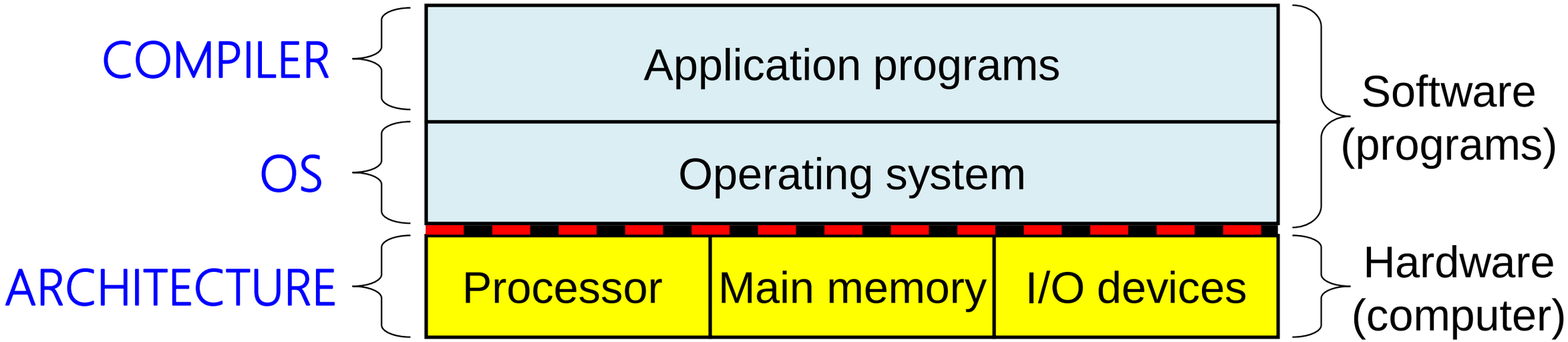
A. Multi-Core Processors (MCP)
   - Case for Multicore Processors
   - Multicore Processor Designs

B. Multi-Computer Clusters (MCC)
   - The Beowulf Cluster
   - The Google Cluster

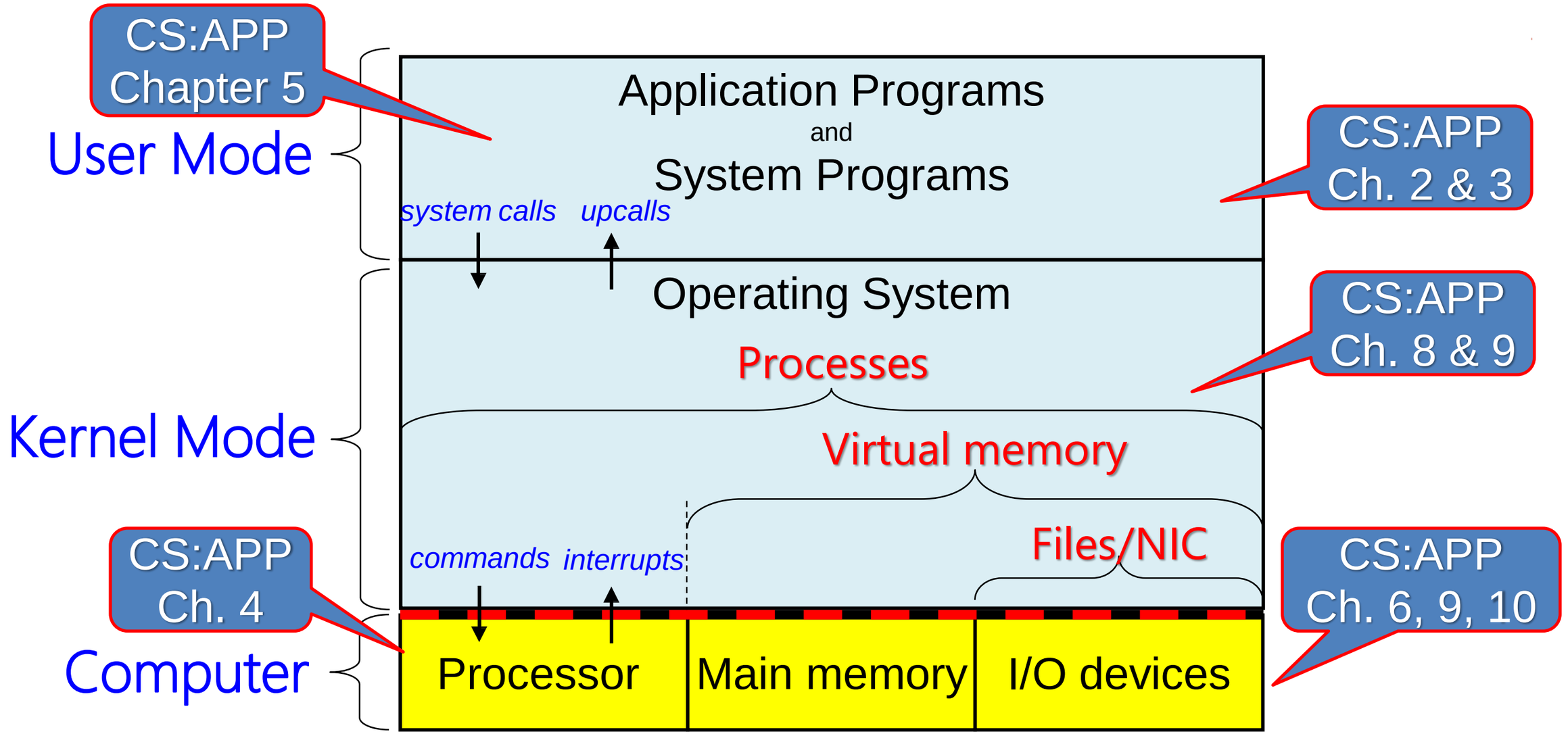
Electrical & Computer ENGINEERING

Anatomy of a Computer System: SW/HW

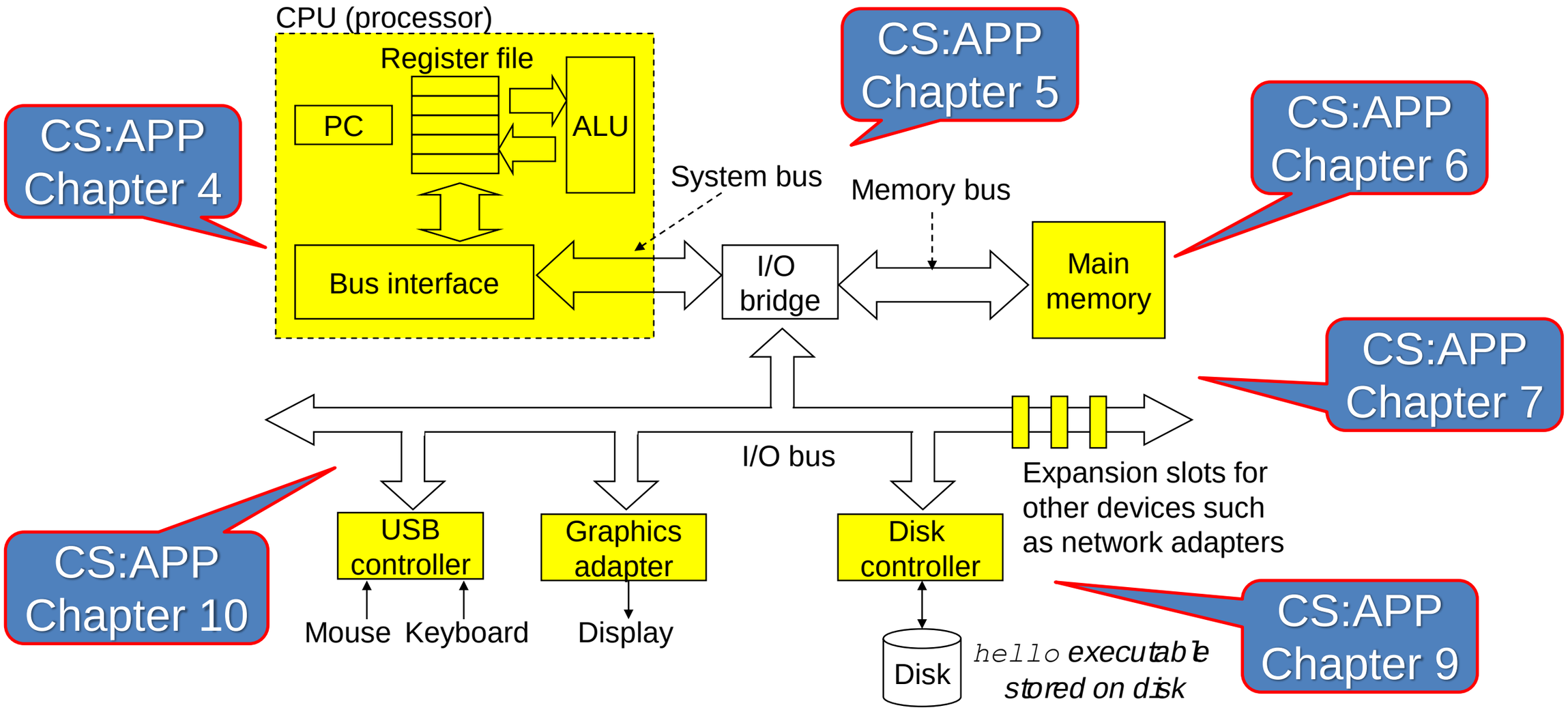
- What is a Computer System?
  - Software + Hardware
  - Programs + Computer ➔ [Application program + OS] + Computer
  - Programming Languages + Operating Systems + Computer Architecture



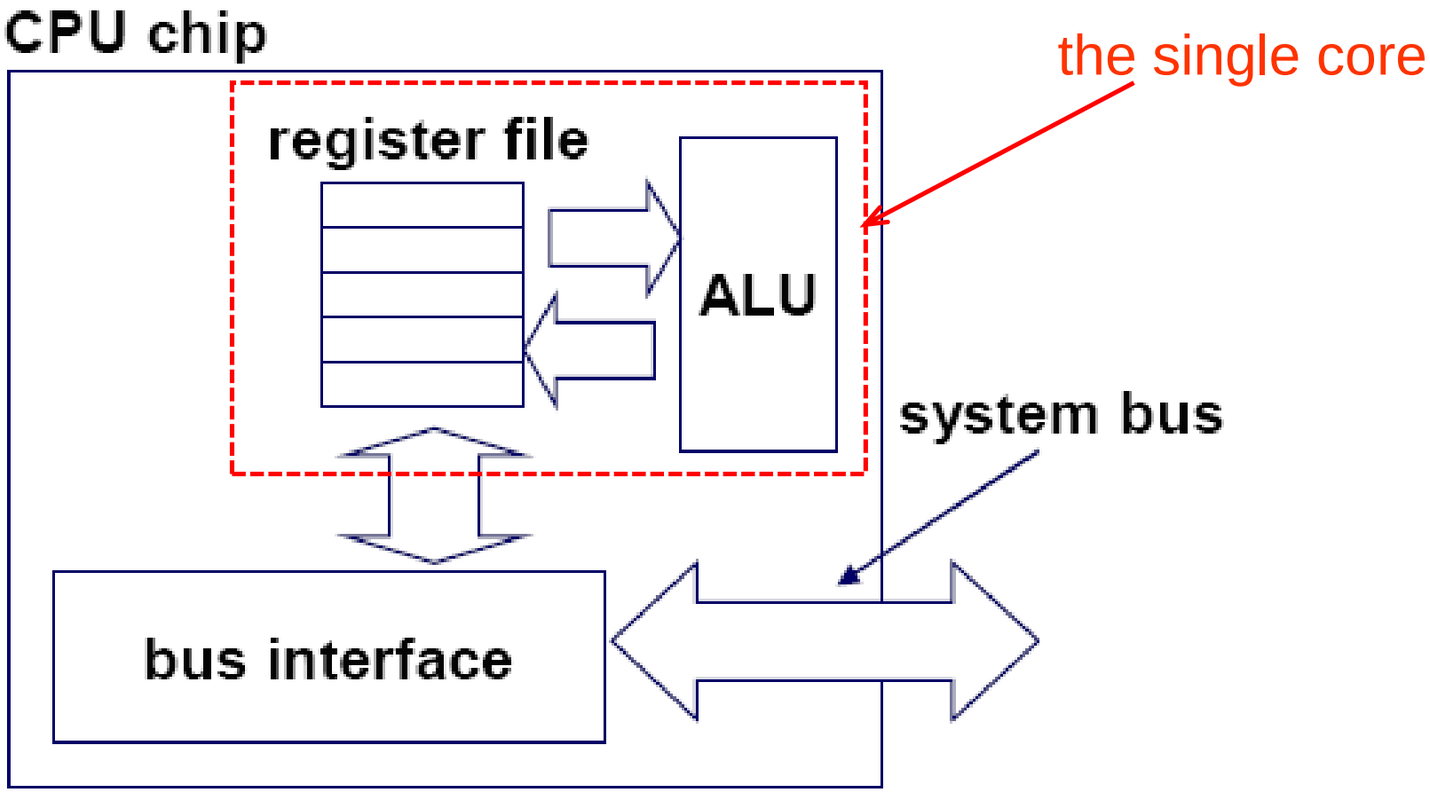
Anatomy of a Typical Computer System



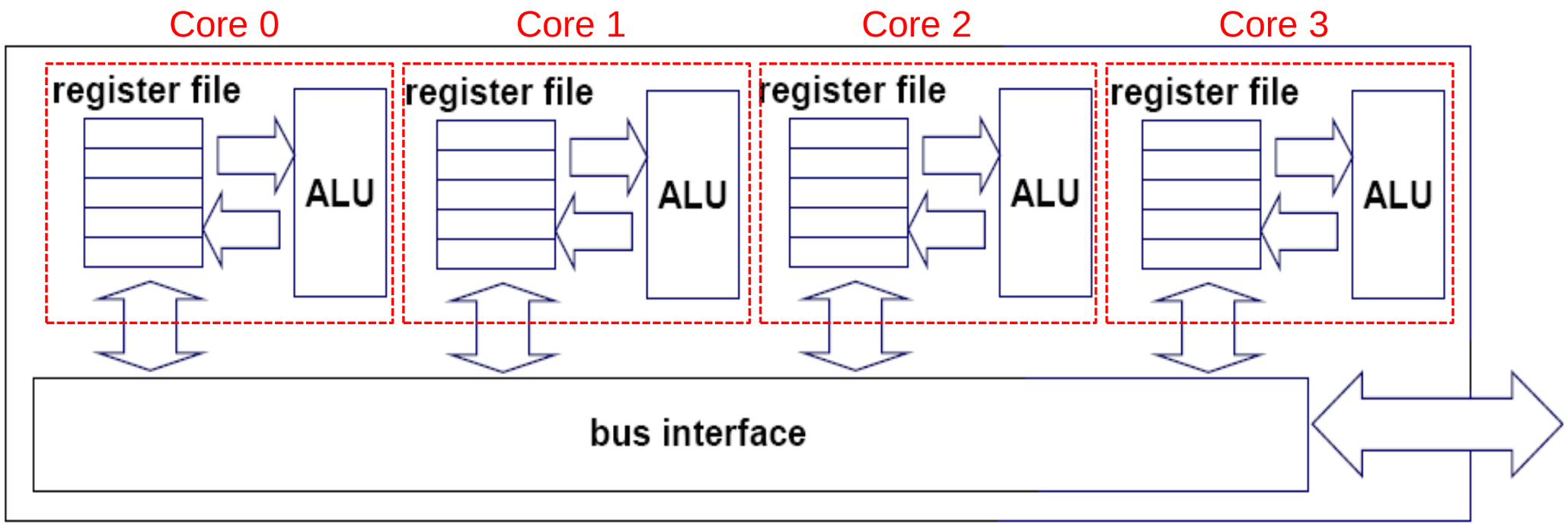
Typical Computer Hardware Organization



Single-core CPU Chip

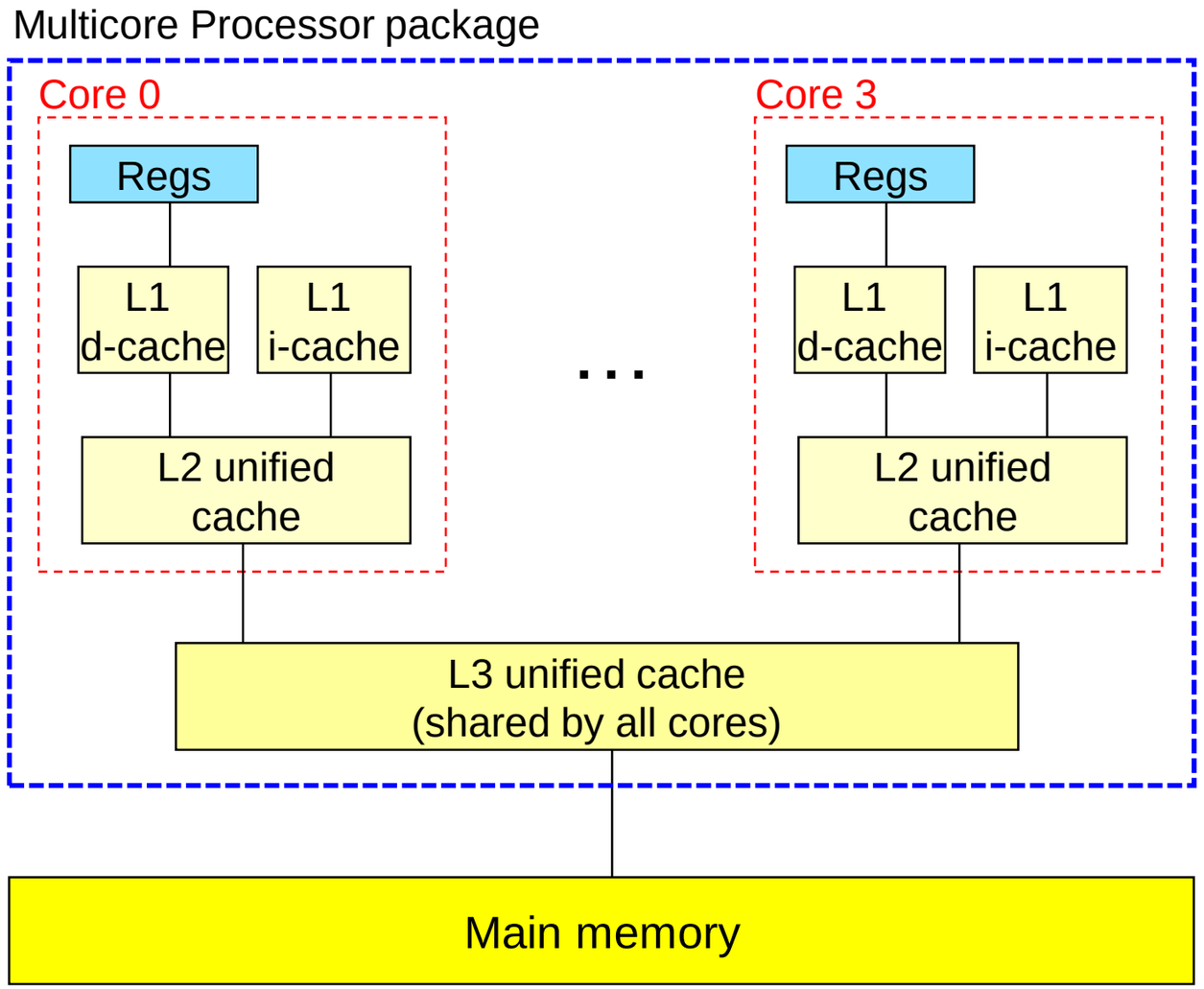


Multicore CPU Chip



Multi-core CPU chip

Multicore Processors or Chip Multiprocessors



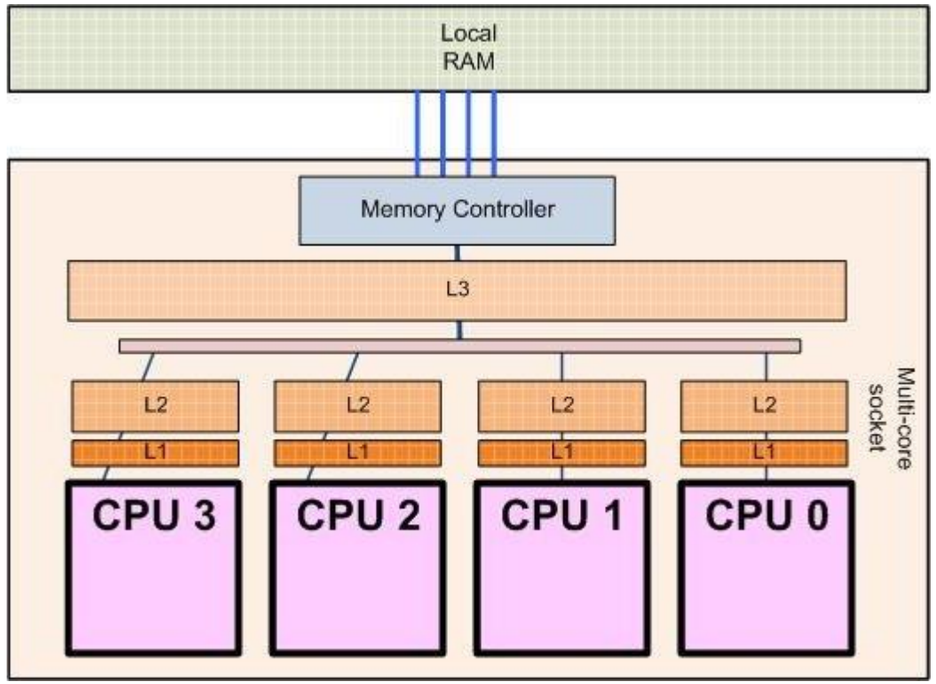
A. Multicore Processors (MCP)

MULTIPROCESSING

Shared Memory Multicore Processors (MCP) or Chip Multiprocessors (CMP)

CLUSTER COMPUTING

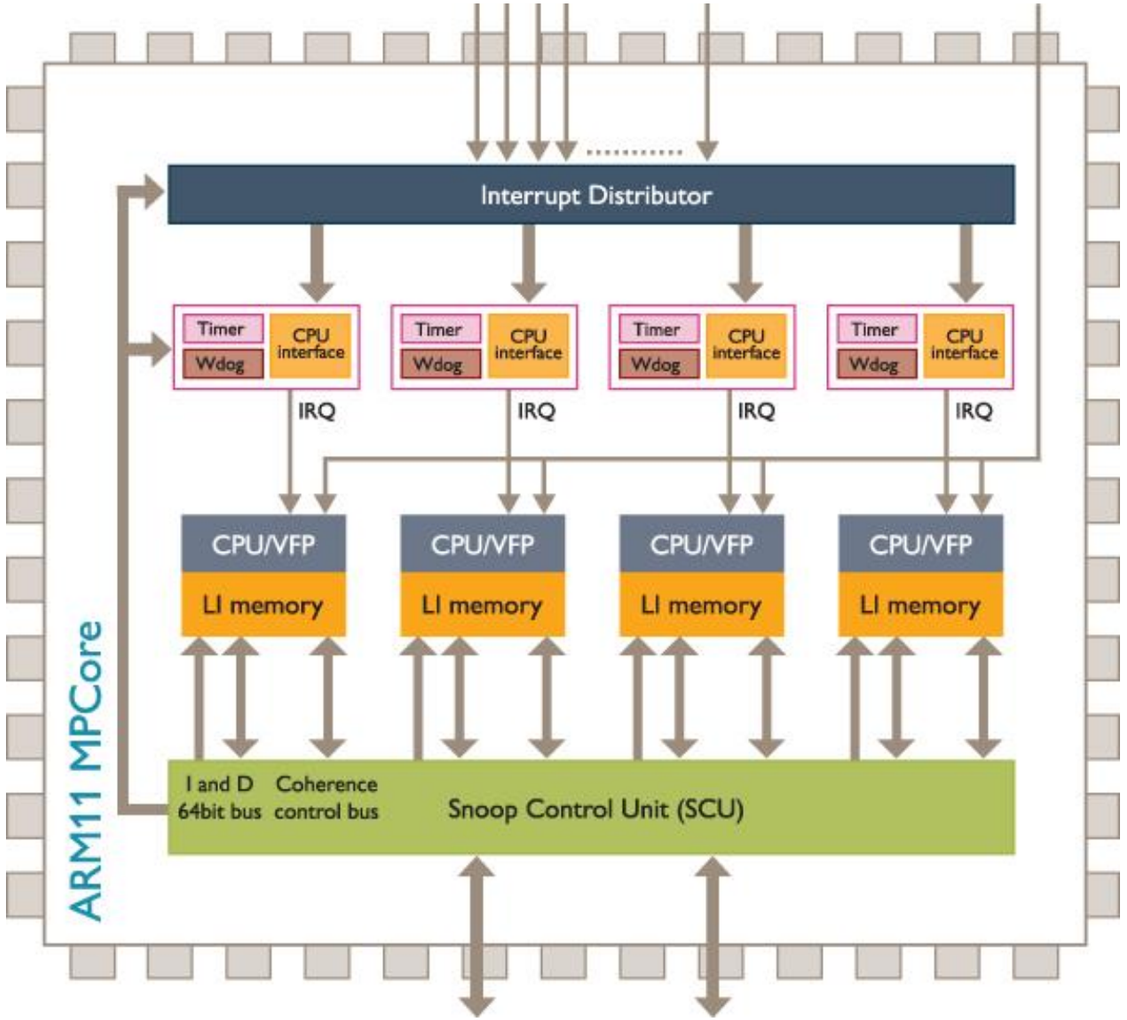
Shared File System and LAN Connected Multi-Computer Clusters (MCC)



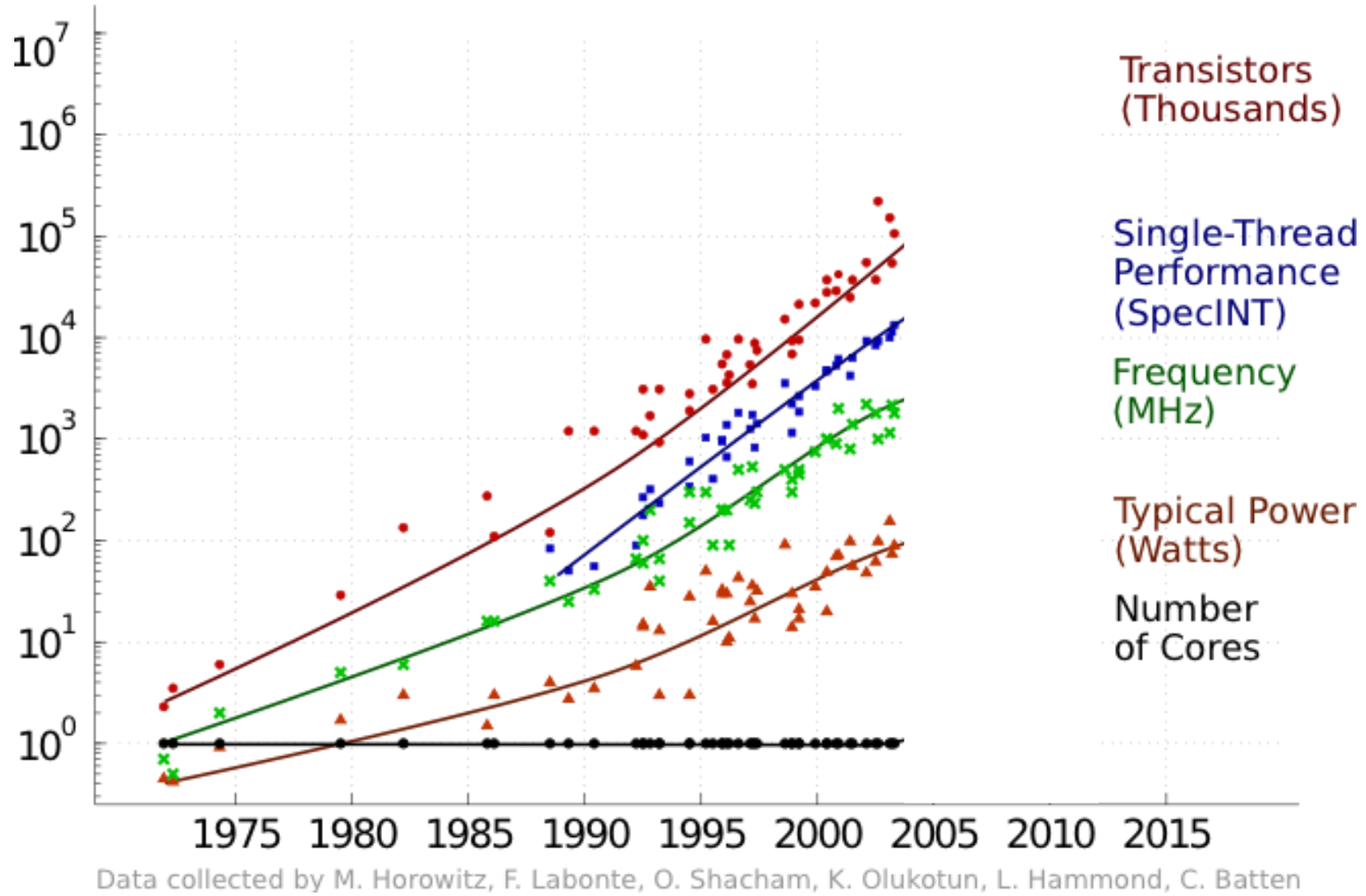
Electrical & Computer ENGINEERING

The Case for Multicore Processors

- Stalled Scaling of Single-Core Performance
- Expected continuation of Moore's Law
- Throughput Performance for Server Workloads



Processor Scaling Until ~2004



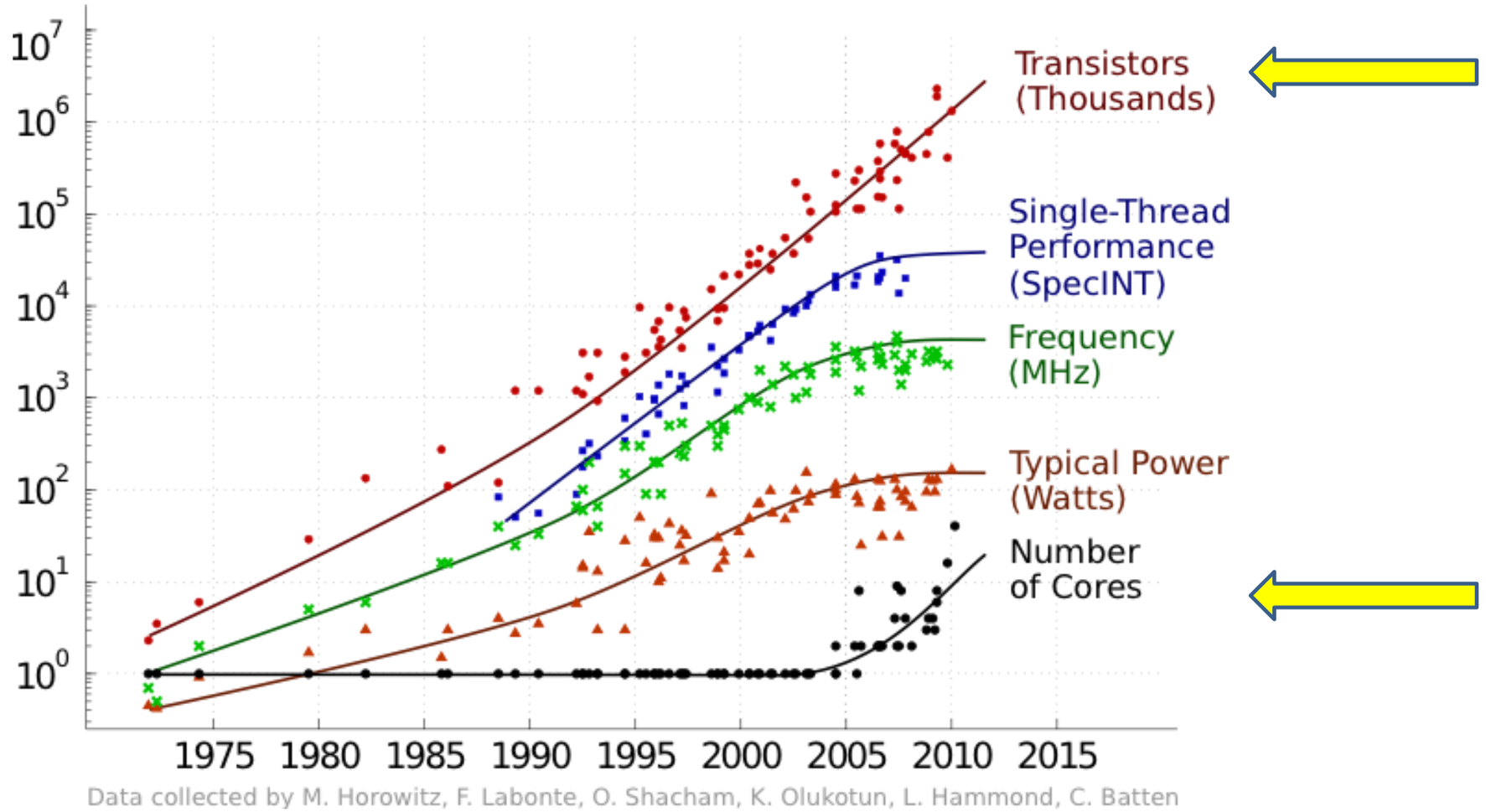
Processor Development Until ~2004

- Moore's Law: transistor count doubles every 18 months
  - Used to improve processor performance by 2x every 18 months
  - Single core, binary compatible to previous generations
- Contributors to performance improvements
  - More ILP through OOO superscalar techniques
    - Wider issue, better prediction, better scheduling, …
    - Better memory hierarchies, faster and larger
  - Clock frequency improvements

Problems with Single Core Performance

- Moore's Law is still doing well (for the foreseeable future…)
- The Power Wall
  - Power ≈ C * V_{dd}^2 * Freq
  - Cannot scale transistor count and frequency without reducing V_{dd}
  - Unfortunately, voltage scaling has essentially stalled
- The Complexity Wall
  - Designing and verifying increasingly large OOO cores is very expensive
    - 100s of engineers for 3-5 years
  - Caches are easier to design but can only help so much…

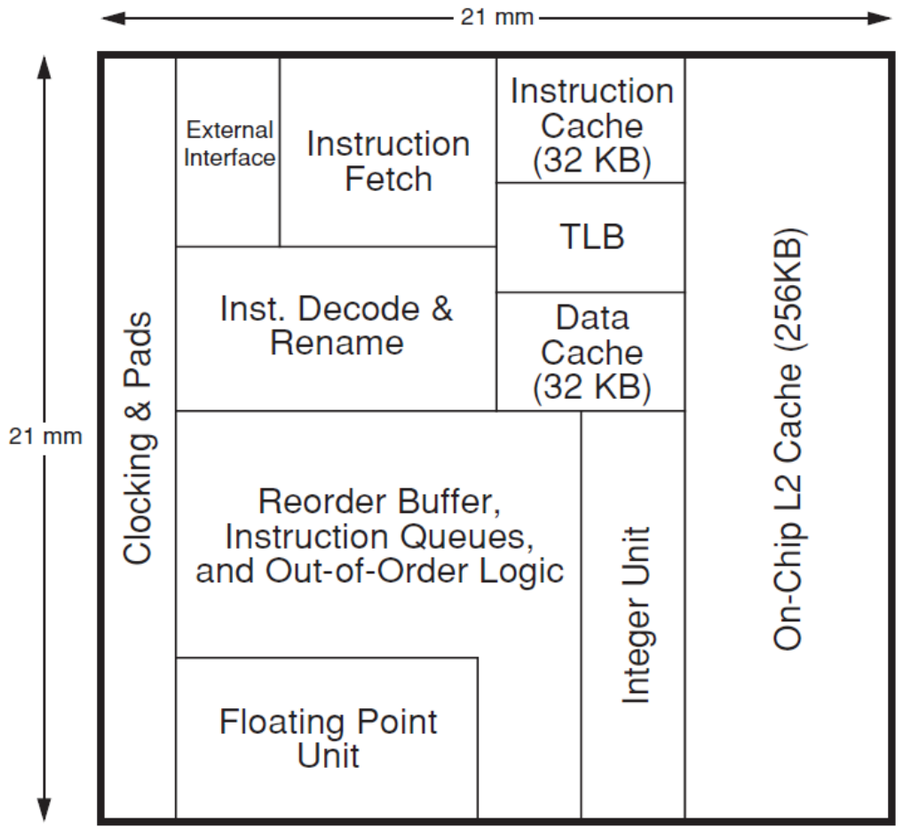
Processor Scaling Since ~2005

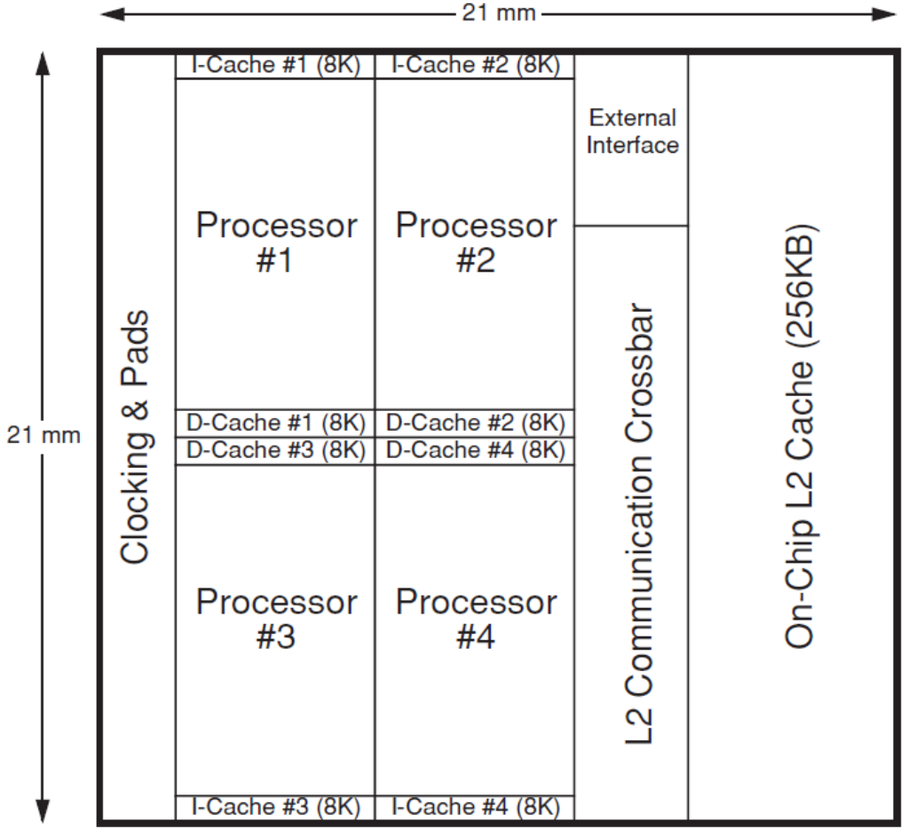


The Multicore Alternative

- Use Moore's law to place more cores per chip
  - Potentially 2x cores/chip with each CMOS generation
  - Without significantly compromising clock frequency
  - Known as Multi-Core Processors (MCP) or Chip Multiprocessors (CMP)
- The good news
  - Continued scaling of chip-level peak (throughput) performance
  - Mitigate the undesirable superscalar power scaling ("wrong side of the square law")
  - Facilitate design and verification, and product differentiation
- The bad news
  - Require multithreaded workloads: multiple programs or parallel programs
  - Require parallelizing single applications into parallel programs
  - Power is still an issue as transistors shrink due to leakage current

OOO Superscalar vs. Multicore Processor





IPC Scaling of Wide-Issue OOO Superscalar

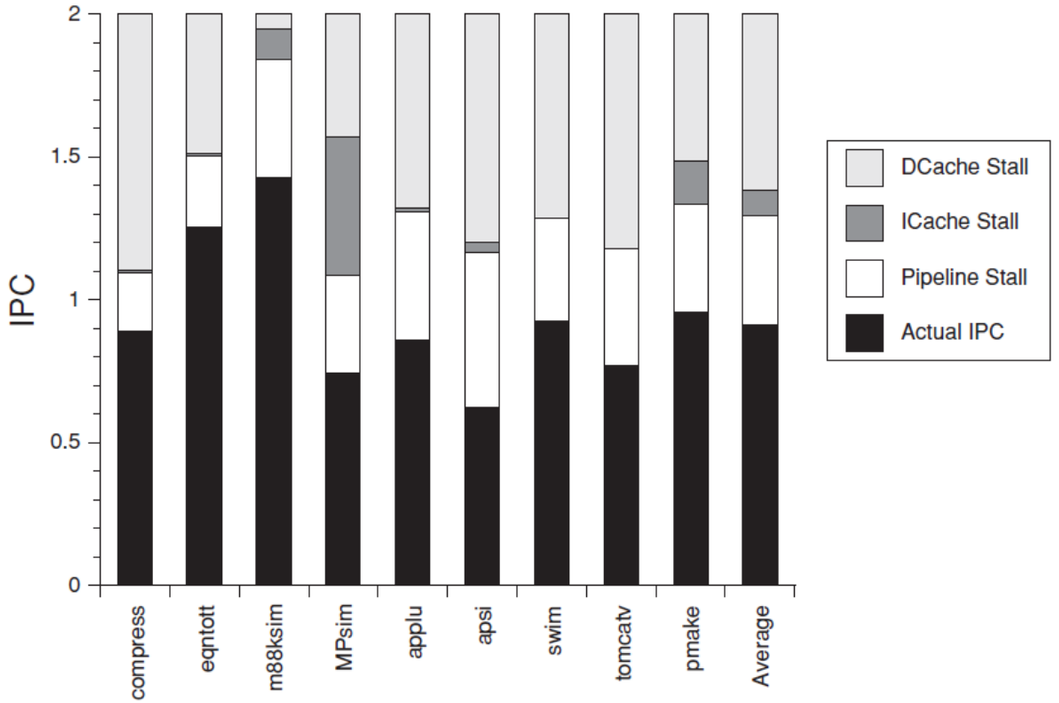


FIGURE 1.7: IPC breakdown for a single 2-issue processor.

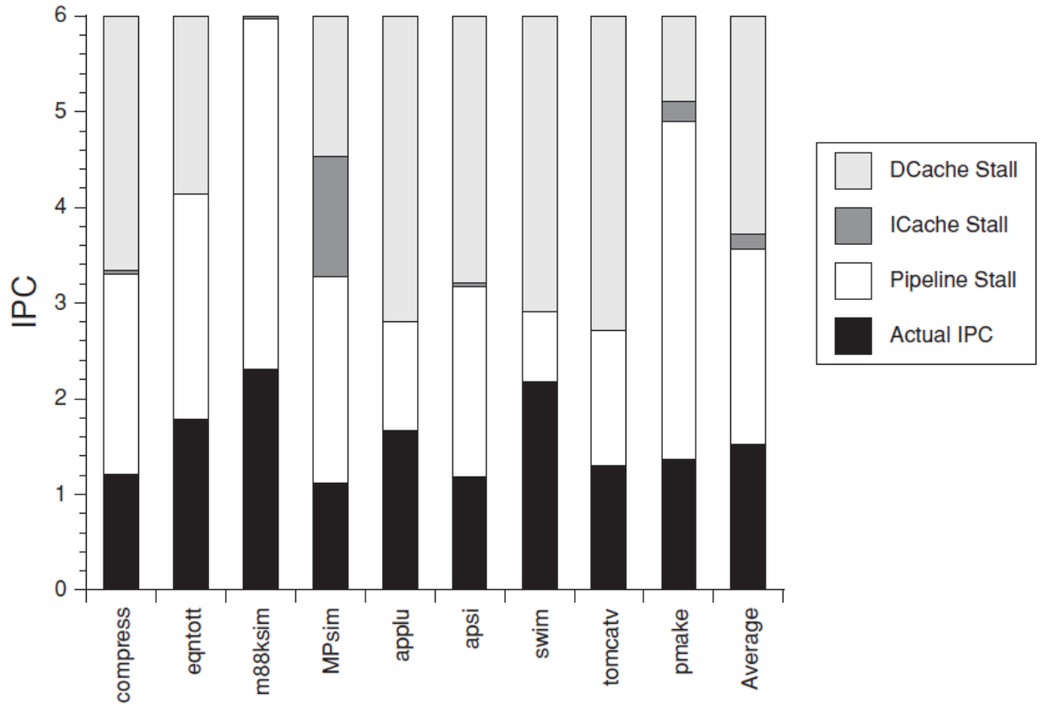
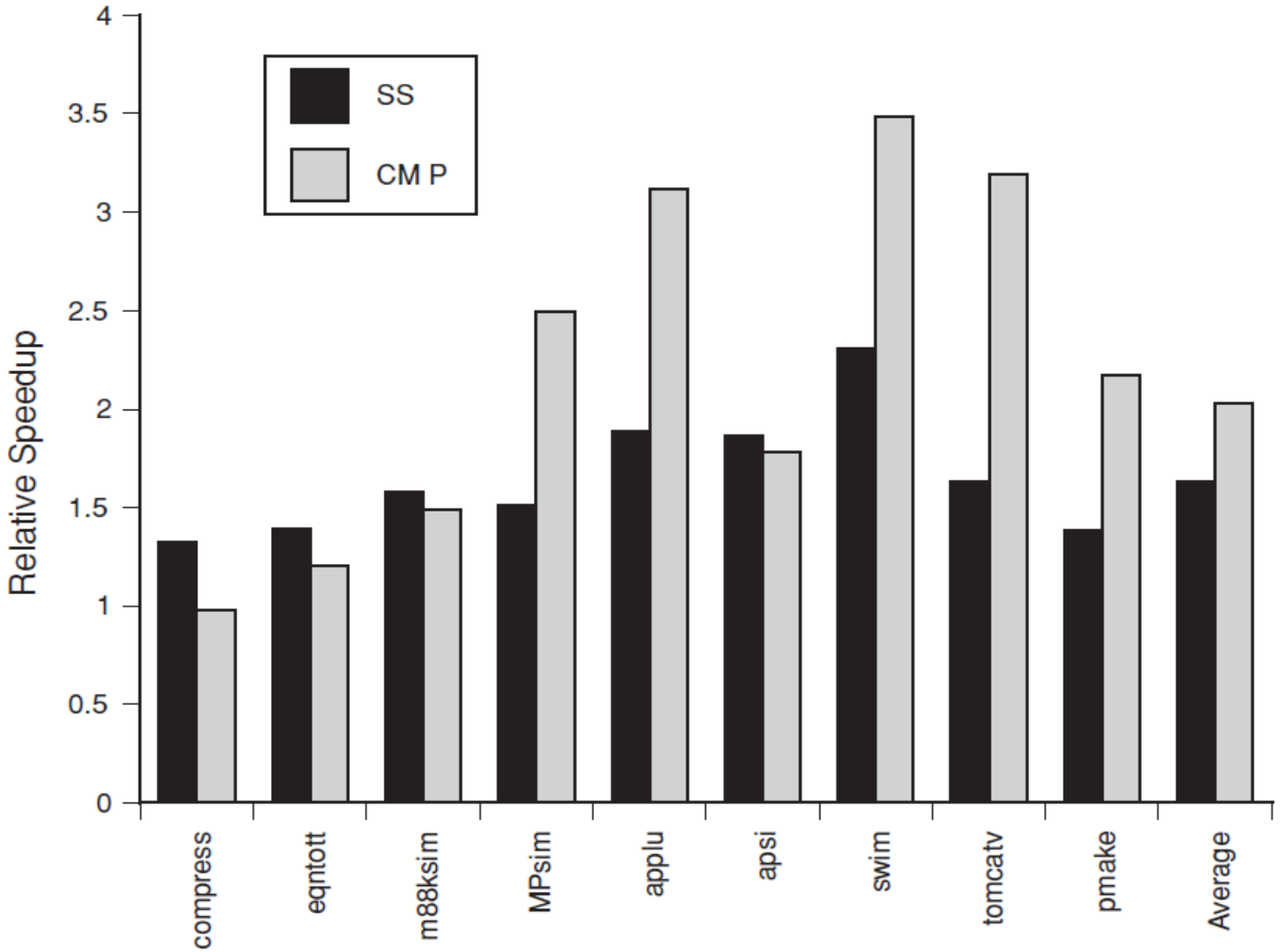
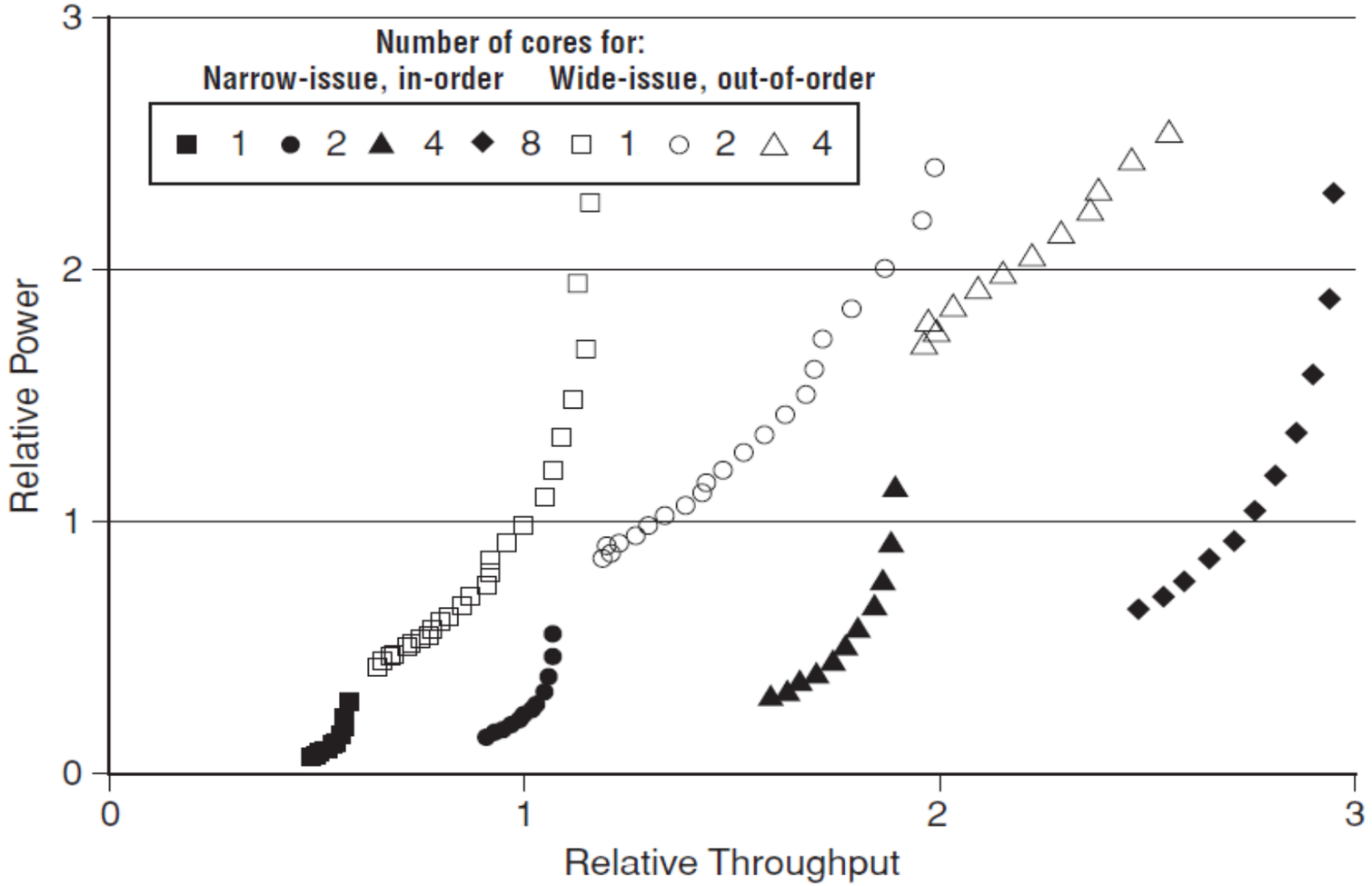


FIGURE 1.8: IPC breakdown for the 6-issue processor.

OOO Superscalar vs. Multicore Processor Speedups



Power Scaling for In-Order vs. OOO Cores



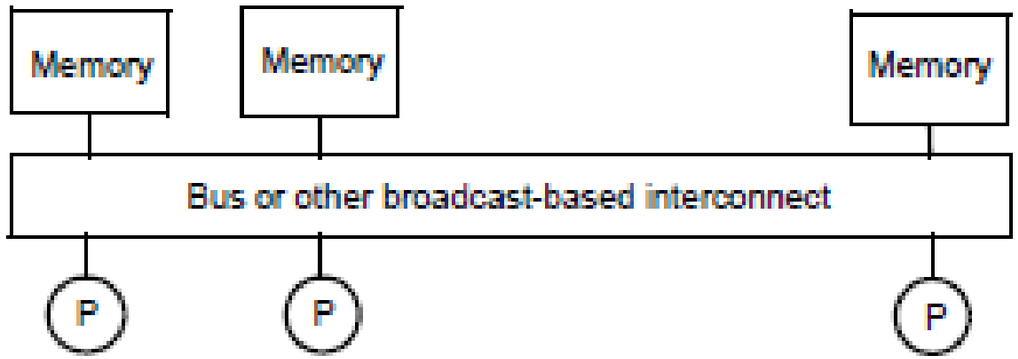
Multicore Processor Design Questions

- Type of cores
  - Big or small cores, e.g. few OOO cores Vs many simple cores
  - Same or different cores, e.g. homogeneous or heterogeneous
- Memory hierarchy
  - Which caching levels are shared and which are private
  - How to effectively share a cache
  - More on this in the next lecture
- On-chip interconnect
  - Bus vs. ring vs. scalable interconnect (e.g., mesh)
  - Flat vs. hierarchical organizations
- HW assists for parallel programming
  - HW support for fine-grain scheduling, transactions, etc.

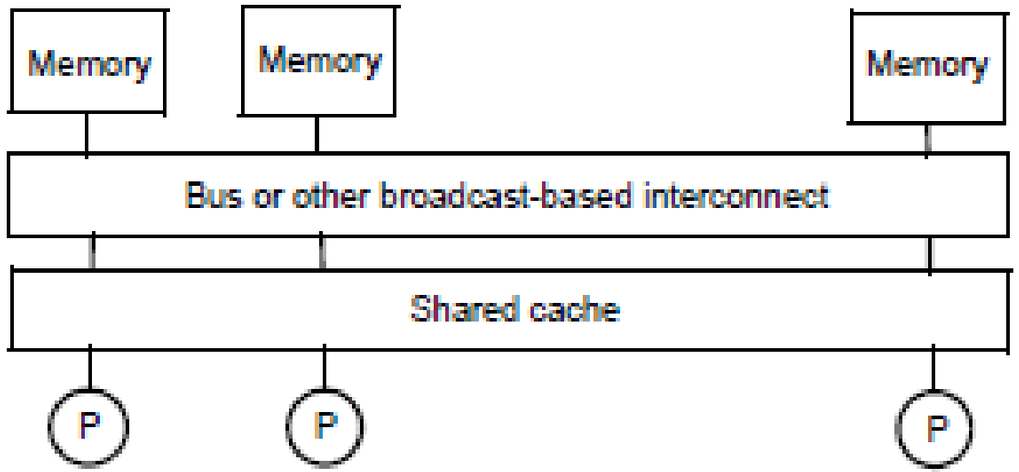
High-Level Design Issues

- Share caches?
  - yes: all designs that connect at L2 or L3
  - no: initial designs that connected at “bus”
- Coherence?
  - Private caches? Reuse existing MP/socket coherence
    - Optimize for on-chip sharing?
  - Shared caches?
    - Need new coherence protocol for on-chip caches
    - Often write-through L1 with back-invalidates for other caches

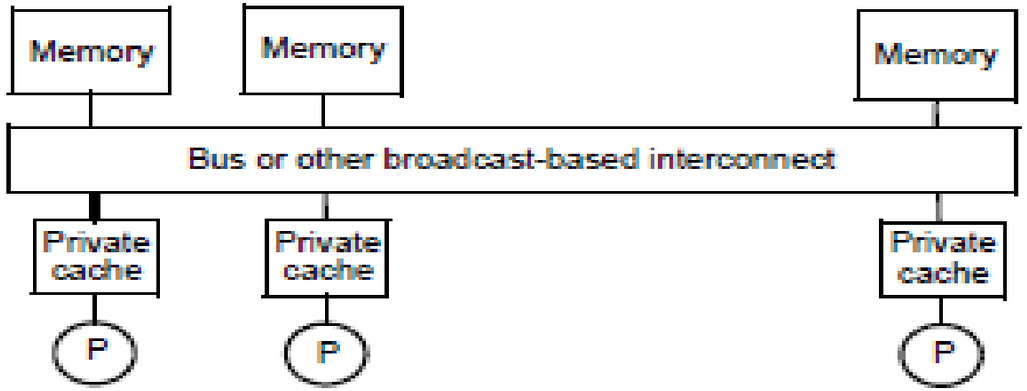
Cache Organization for Parallel Systems



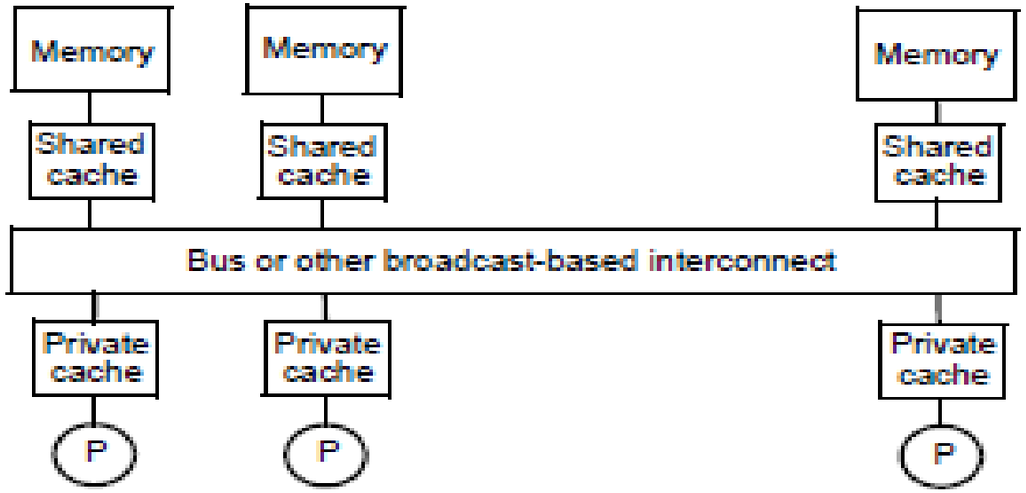
(a) Dance-hall multiprocessor architecture or SMP



(b) SMP with shared level 1 cache



(c) SMP with private caches

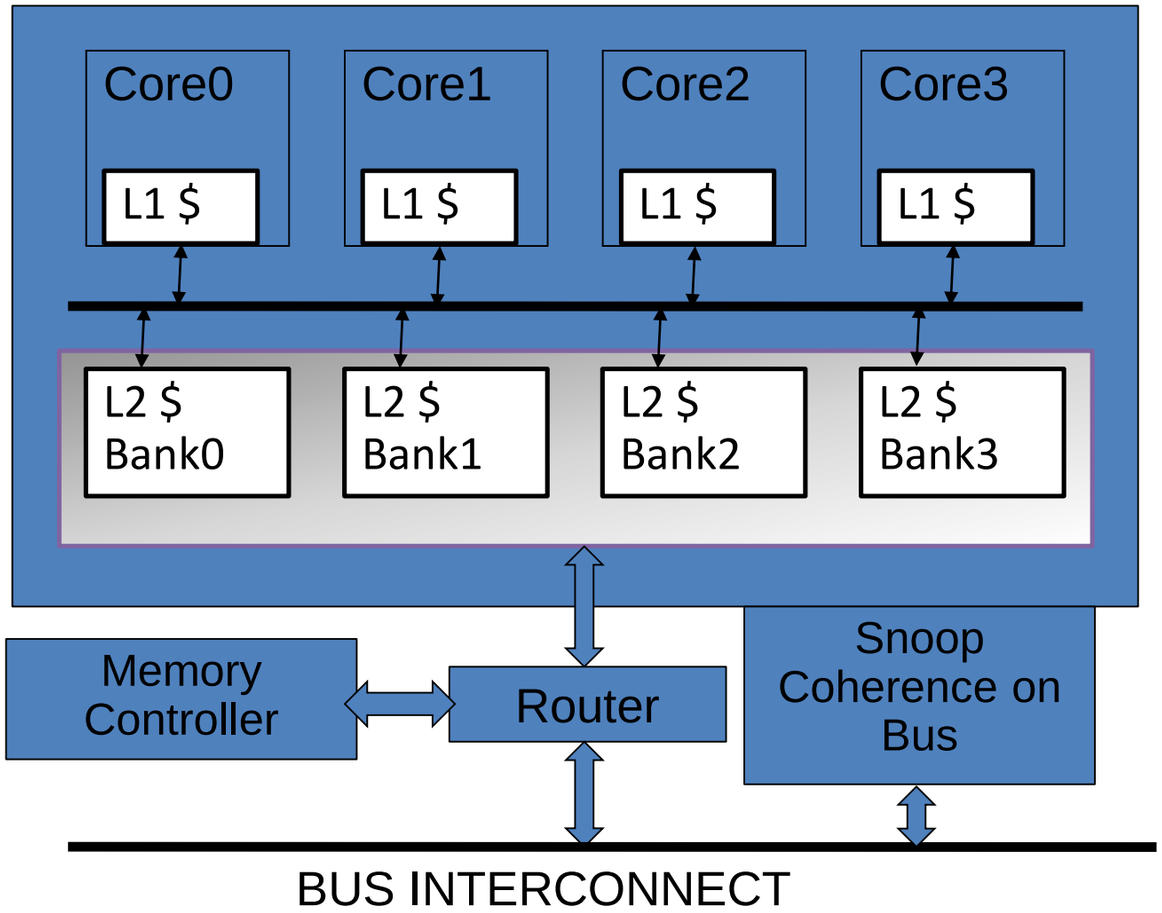


(d) SMP with private caches and shared Level 2 cache

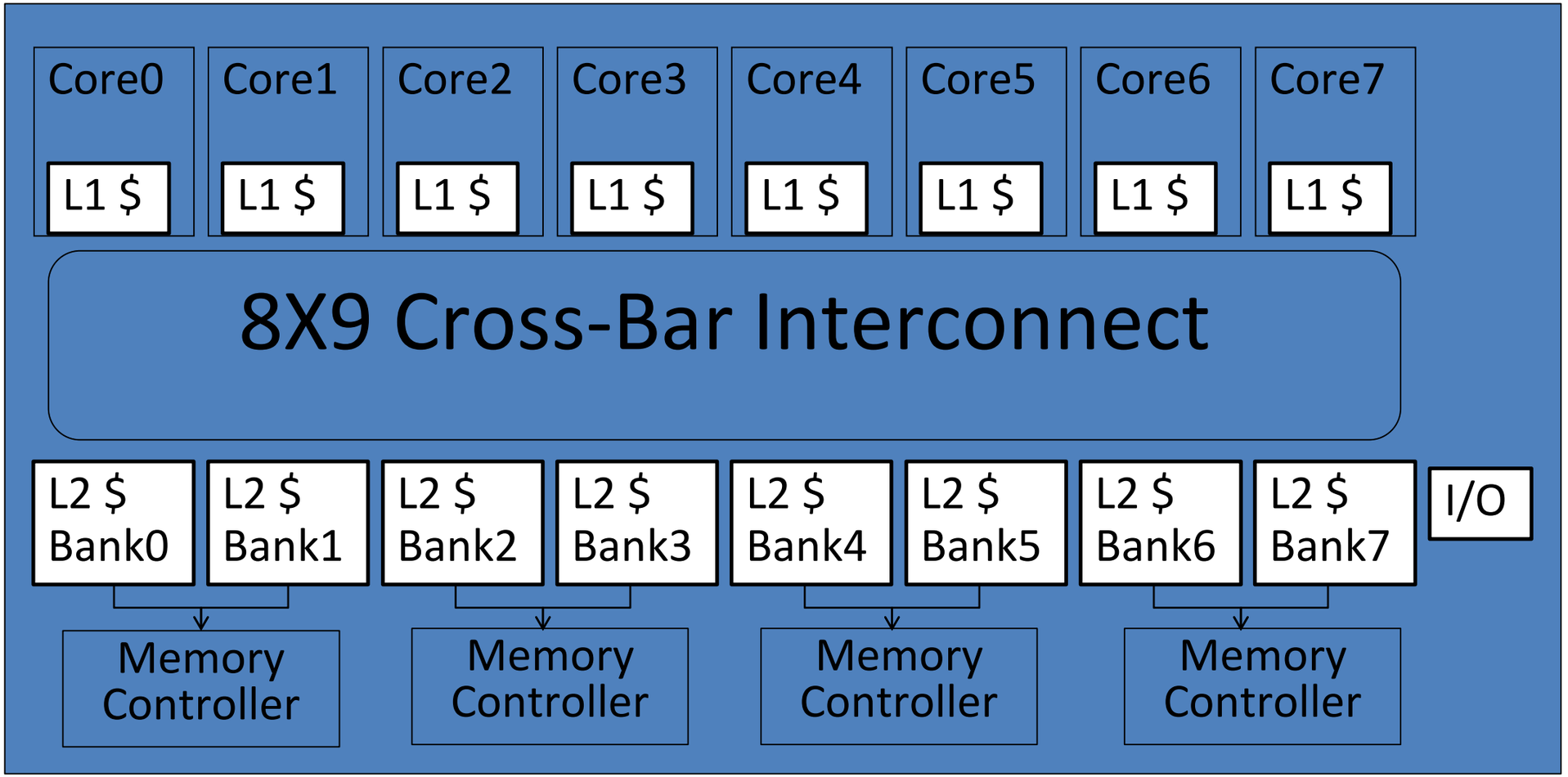
Multicore Interconnects

- Bus/Crossbar
- Point-to-point links, many possible topologies
  - 2D (suitable for planar realization)
    - Ring
    - Mesh
    - 2D torus
  - 3D - may become more interesting with 3D packaging (chip stacks)
    - Hypercube
    - 3D Mesh
    - 3D torus

Bus-Based MCPs (e.g. Pentium 4 Dual Core)



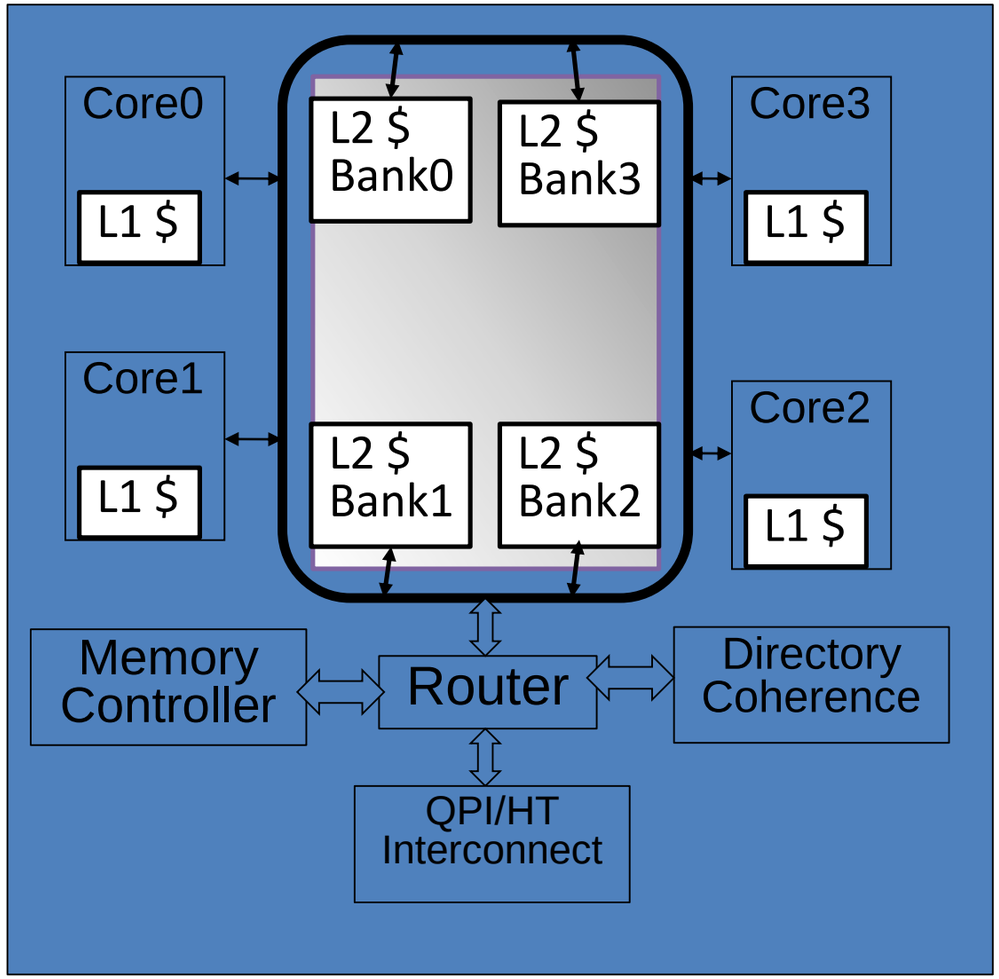
Crossbar-Based MCPs (e.g. IBM Power4/5/6/7)



On-Chip Bus/Crossbar Interconnects

- Used widely (Power4/5/6/7 Piranha, Niagara, etc.)
  - Assumed not scalable
  - Is this really true, given on-chip characteristics?
  - May scale "far enough": watch out for arguments at the limit
  - e.g. swizzle-switch makes x-bar scalable enough [UMich]
- Simple, straightforward, nice ordering properties
  - Wiring can be a nightmare (for crossbar)
  - Bus bandwidth is weak (even multiple busses)
  - Compare DEC Piranha 8-lane bus (32GB/s) to Power4 crossbar (100+GB/s)
  - Workload demands: commercial vs. scientific

Ring-Based MCPs



On-Chip Ring Interconnect

- Point-to-point ring interconnect
  - Simple, easy
  - Nice ordering properties (unidirectional)
  - Every request a broadcast (all nodes can snoop)
  - Scales poorly: *O(n)* latency, fixed bandwidth
- Optical ring (nanophotonic)
  - HP Labs Corona project [Vantrease review]
  - Latency is arguably *O(sqrt(n))*
    - Covert switching – broadcast not easy any more
  - Still fixed bandwidth (but lots of it)

On-Chip Mesh

- Widely assumed in academic literature
- Tilera (Wentzlaff reading), Intel 80-core prototype
- Not symmetric, so have to watch out for load imbalance on inner nodes/links
  - 2D torus: wraparound links to create symmetry
    - Not obviously planar
    - Can be laid out in 2D but longer wires, more intersecting links
- Latency, bandwidth scale well
- Lots of existing literature

MCPs with Heterogeneous Cores

- WORKLOADS HAVE DIFFERENT CHARACTERISTICS
  - LARGE NUMBER OF SMALL CORES (APPLICATIONS WITH HIGH THREAD COUNT)
  - SMALL NUMBER OF LARGE CORES (APPLICATIONS WITH SINGLE THREAD OR LIMITED THREAD COUNT)
  - MIX OF WORKLOADS
  - MOST PARALLEL APPLICATIONS HAVE PARALLEL AND SERIAL SECTIONS (AMDAHL LAW)
- HENCE, HETEROGENEITY
  - TEMPORAL: e.g., EPI THROTTLING
  - SPATIAL: EACH CORE CAN DIFFER EITHER IN PERFORMANCE OR FUNCTIONALITY
- PERFORMANCE ASYMMETRY
  - USING HOMOGENEOUS COREs AND DVFS, OR PROCESSOR WITH MIXED CORES
  - VARIABLE RESOURCES: e.g., ADAPT SIZE OF CACHE BY GATING OFF POWER TO CACHE BANKS (UP TO 50%)
  - SPECULATION CONTROL (LOW BRANCH PREDICTION CODE): THROTTLE THE NUMBER OF IN-FLIGHT INSTRUCTIONS (REDUCES ACTIVITY FACTOR)

MCPs with Heterogeneous Cores

- FUNCTIONAL ASYMMETRY
  - USE HETEROGENEOUS CORES
    - E.G., GP CORES, GRAPHICS PROCESSORS, CRYTOGRAPHY, VECTOR CORES, FLOATING-POINT CO-PROCESSORS
    - HETEROGENEOUS CORES MAY BE PROGRAMMED DIFFERENTLY
    - MECHANISMS MUST EXIST TO TRANSFER ACTIVITY FOR ONE CORE TO ANOTHER
    - FINE-GRAIN: IN THE CASE OF FLOATING POINT CO-PROCESSOR, USE ISA
    - COARSE GRAIN: TRANSFER THE COMPUTATION FROM ONE CORE TO ANOTHER USING APIs
  - EXAMPLES:
    - CORES WITH DIFFERENT ISAs
    - CORES WITH DIFFERENT CACHE SIZES, DIFFERENT ISSUE WIDTH, DIFFERENT BRANCH PREDICTORS
    - CORES WITH DIFFERENT MICRO-ARCHITECTURES (E.G., STATIC AND DYNAMIC)
    - DIFFERENT TYPES OF CORES (E.G., GP AND SIMD)
  - GOALS:
    - SAVE AREA (MORE CORES!)
    - SAVE POWER BY USING CORES WITH DIFFERENT POWER/PERFORMANCE CHARACTERISTICS FOR DIFFERENT PHASES OF EXECUTION

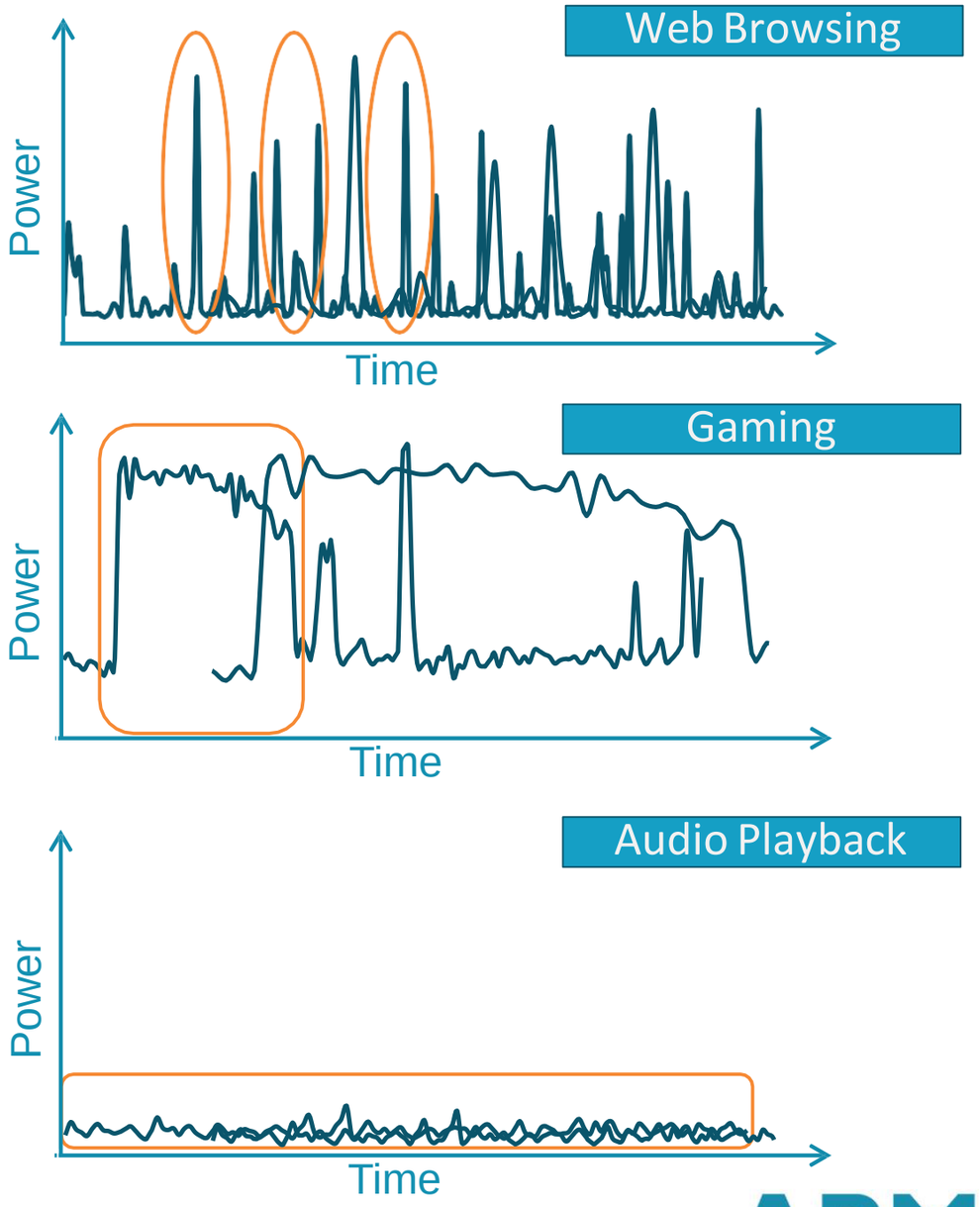
MCPs with Heterogeneous Cores

- DIFFERENT APPLICATIONS MAY HAVE BETTER PERFORMANCE/POWER CHARACTERISTICS ON SOME TYPES OF CORE (STATIC)
- SAME APPLICATION GOES THROUGH DIFFERENT PHASES THAT CAN USE DIFFERENT CORES MORE EFFICIENTLY (DYNAMIC)
  - EXECUTION MOVES FROM CORE TO CORE DYNAMICALLY
  - MOST INTERESTING CASE (DYNAMIC)
  - COST OF SWITCHING CORES (MUST BE INFREQUENT: SUCH AS O/S TIME-SLICE)
- ASSUME CORES WITH SAME ISA BUT DIFFERENT PERFORMANCE/ENERGY RATIO
  - NEED ABILITY TO TRACK PERFORMANCE AND ENERGY TO MAKE DECISIONS
  - GOAL: MINIMIZE DELAY-ENERGY PRODUCT
  - SAMPLE PERFORMANCE AND ENERGY SPENT PERIODICALLY
  - TO SAMPLE, RUN APPLICATION ON ONE OR MULTIPLE CORES IN SMALL INTERVALS

Mobile Application Workloads

- Mobile users spend a high amount of time on a range of mobile applications*:
  - 38% on web browsing and Facebook
  - 32% on gaming
  - 16% on audio, video and utility

- Common “building blocks” in workloads:
  - Short bursts of high intensity
  - Long periods of sustained high intensity
  - Low intensity



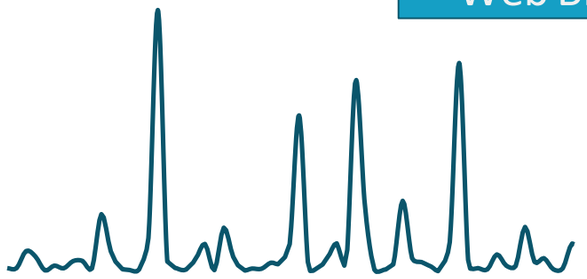
* Source: Flurry Analytics

Measured on a Quad Cortex-A7 Symmetric Multiprocessing platform

ARM

Mobile Application Workloads

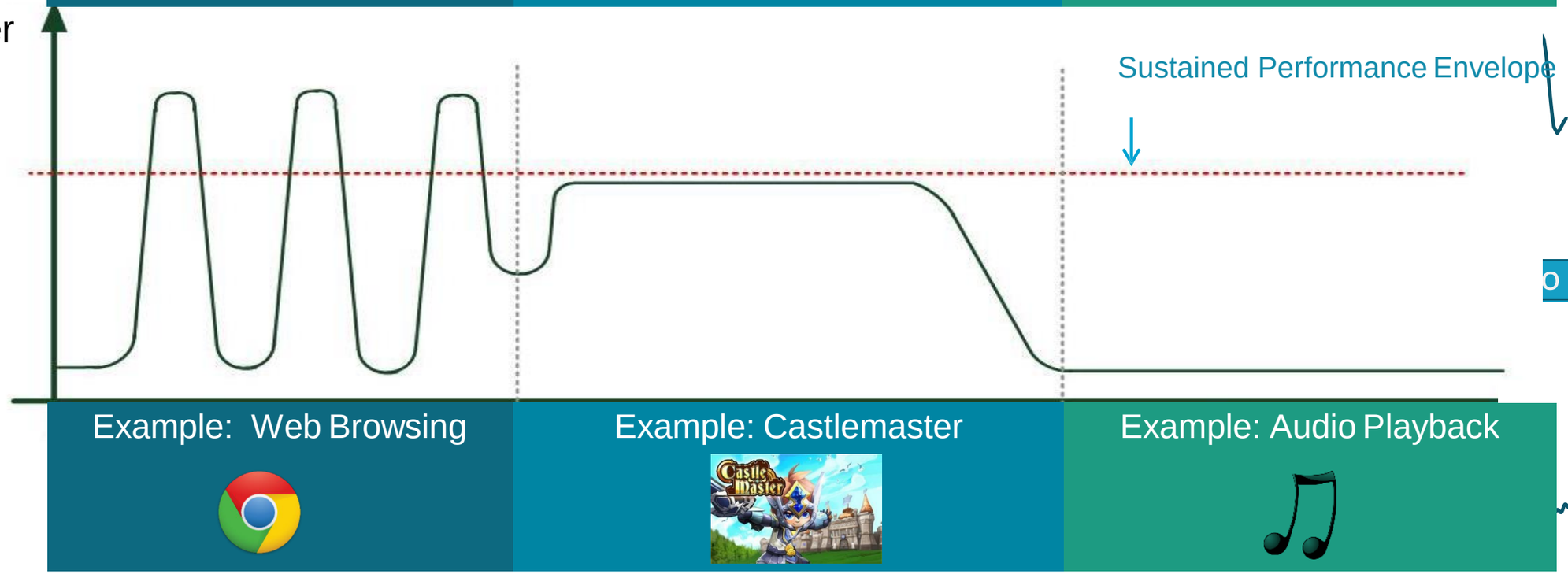
- Mobile users spend a high amount of time on a range of mobile applications:
  - 38% on Web Browsing and Facebook
  - 32%
- Co



Web Browsing

aming

o Playback

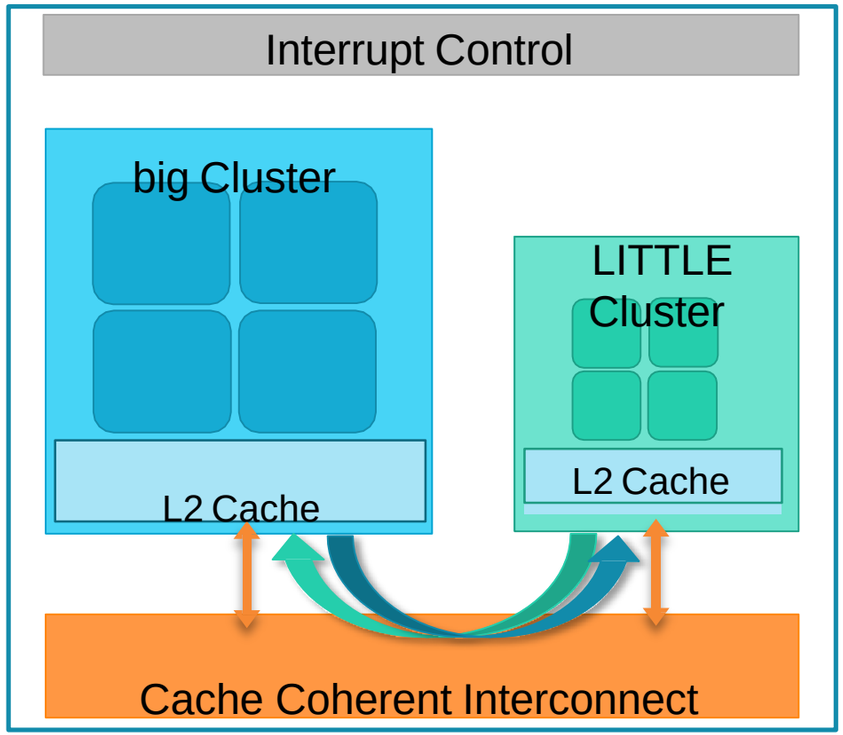


Measured on a Quad Cortex-A7 Symmetric Multiprocessing platform

ARM

big.LITTLE Technology

- Heterogeneous Computing
  - 2x higher performance vs. LITTLE only
  - Up to 75% CPU power savings vs. big only
- Architecturally Identical Processors
  - High-performance tuned big cores
  - Low-power tuned LITTLE cores
- Hardware Coherency
  - Cache Coherent Interconnect (CCI)
  - L1 and L2 snooping between clusters
- Seamless & Automatic Task Allocation



“Right Task on the Right Core”

Up to 40**% SOC power savings***

* Measured across a set of casual games and common use-cases on an ARM Partner 4xCortex-A15.4xCortex-A7 big.LITTLE device

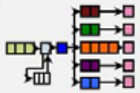
ARM

big.LITTLE Technology

LITTLE

Most energy-efficient applications processor from ARM

- Simple, in-order, 8 stage pipelines
- Performance better than mainstream, high-volume smartphones (Cortex-A8 and Cortex-A9)



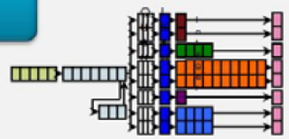
Cortex-A7

Cortex-A53

big

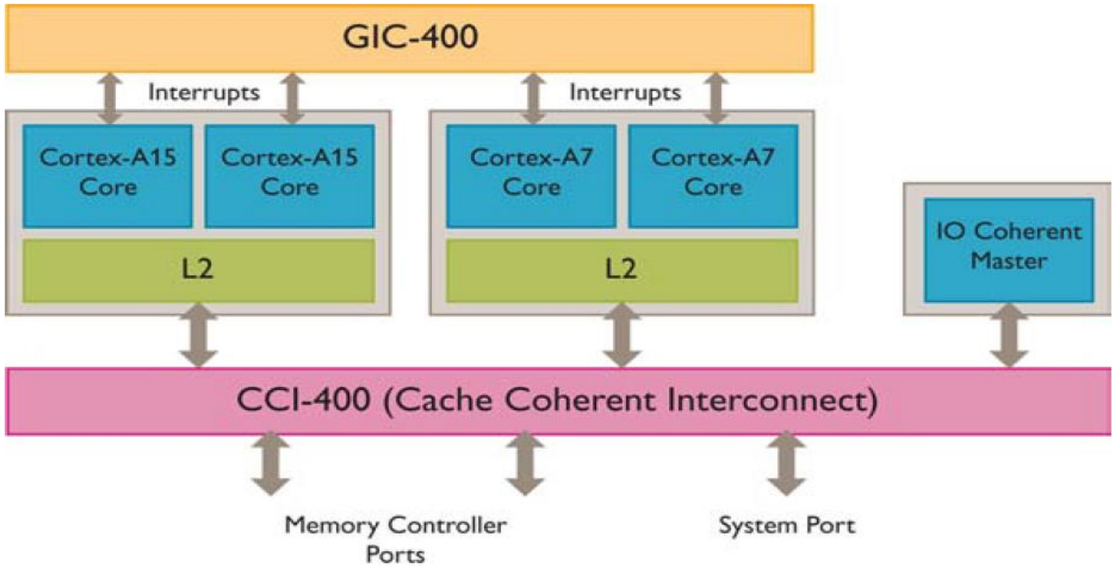
Highest performance in mobile power envelope

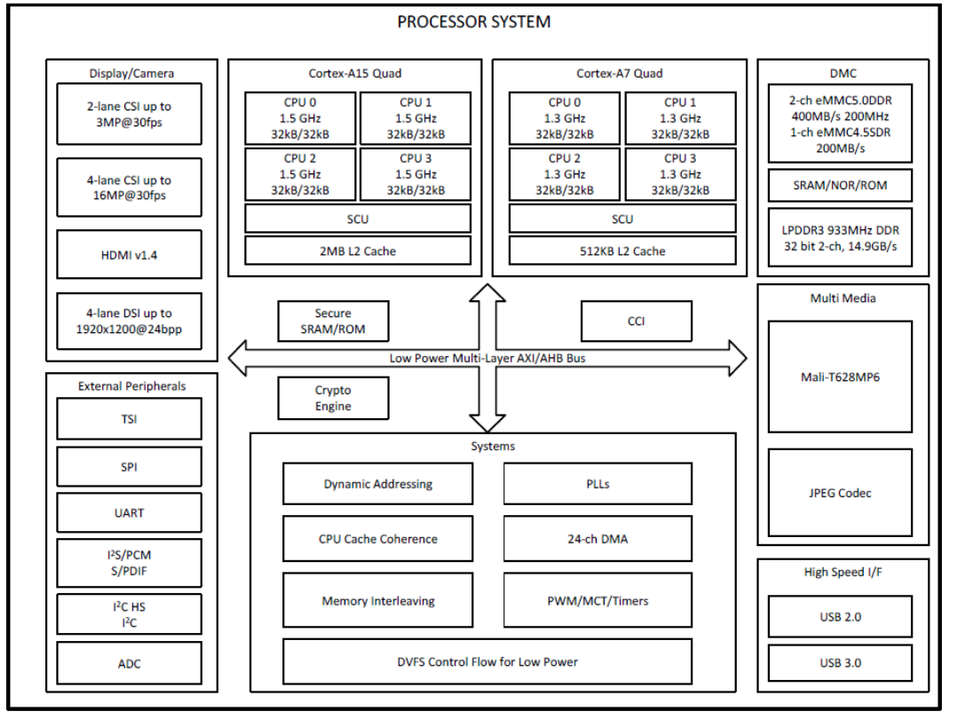
- Complex, out-of-order, multi-issue pipelines
- Up to 2x the performance of today's high-end smartphones



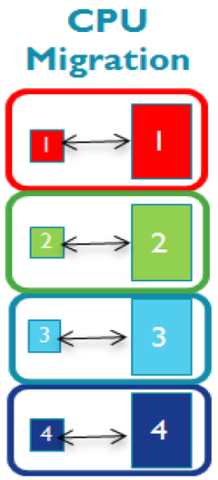
Cortex-A15

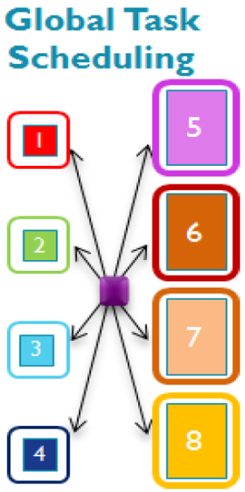
Cortex-A57





ARTIK 1020 Module Processor Sub System





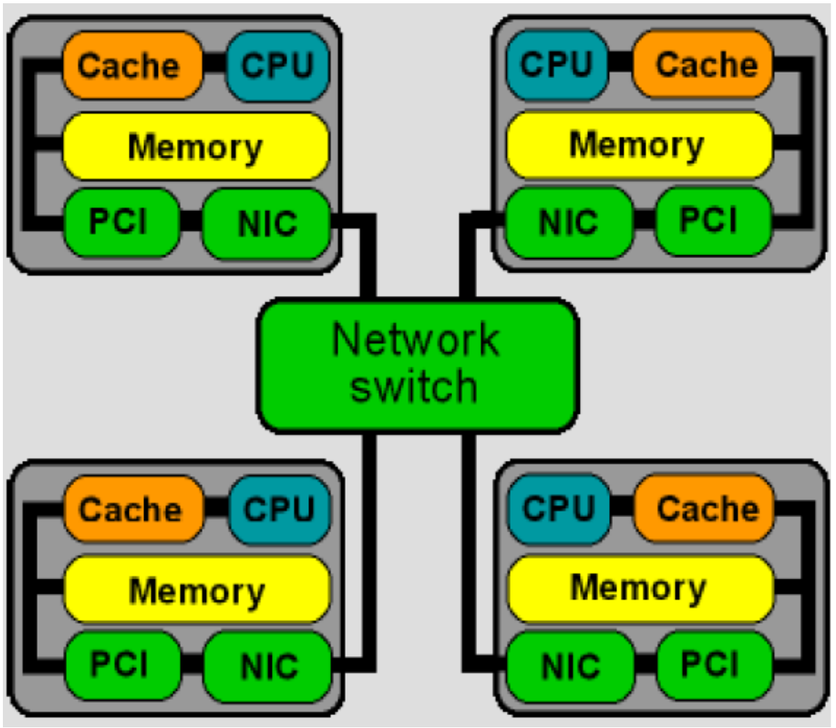
B. Multi-Computer Clusters (MCC)

MULTIPROCESSING

Shared Memory Multicore Processors (MCP) or Chip Multiprocessors (CMP)

CLUSTER COMPUTING

Shared File System and LAN Connected Multi-Computer Clusters (MCC)



Electrical & Computer ENGINEERING

[J.E. Smith, 2007, U. Wisc. ECE/CS 757]

What is a Multi-Computer Cluster?

- Cluster = a set of computers connected with a network
  - *Cluster can be viewed by the user as a single system (or a "multi-computer" system)*
  - Typically commodity computers (PCs) connected by commodity LAN (Ethernet)
  - Each computer (cluster node) runs its own OS and has its own address space
  - Supports execution of parallel programs via message passing with a master node
  - Typically supported by a shared file system and some "clustering middleware"
- Advantages
  - Easy to build: early systems were customer built using commodity PCs and networks
  - Relatively inexpensive: can get significant throughput performance at very low cost
  - Wide range of systems: can vary from small personal clusters to supercomputers
  - *Leverage concurrent advancements in personal computers and local area networks*

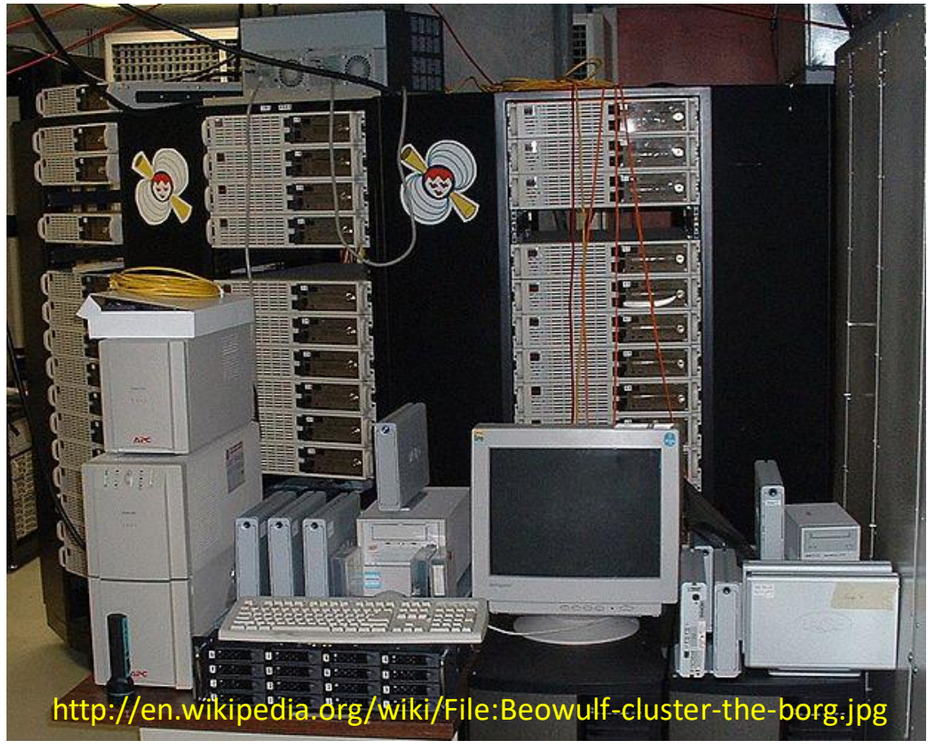
The Beowulf Cluster

[Thomas Sterling & Donald Becker, NASA, 1994]

"Beowulf is a multi-computer which can be used for parallel computations. It is a system with one server node, and one or more client nodes connected via Ethernet. Beowulf also uses commodity software like FreeBSD, Linux, PVM (Parallel Virtual Machine) and MPI (Message Passing Interface). Server node controls the cluster and serves files to the client nodes."

"If you have two networked computers which share at least the `/home` *file system via NFS, and trust each other to execute remote shell (*rsh*), then it could be argued that you have a simple, two node Beowulf machine."*

http://en.wikipedia.org/wiki/File:Beowulf.jpg

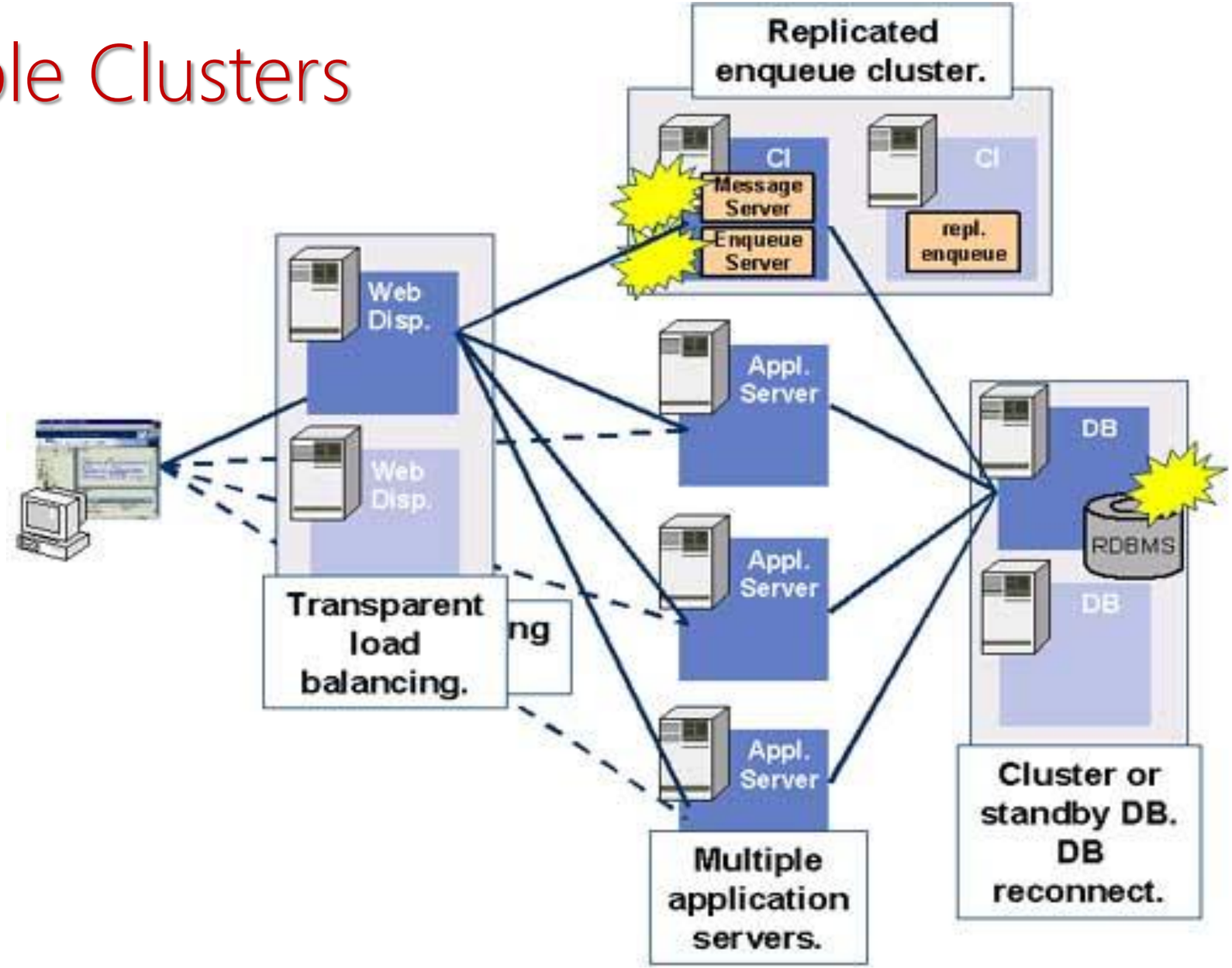


http://en.wikipedia.org/wiki/File:Beowulf-cluster-the-borg.jpg

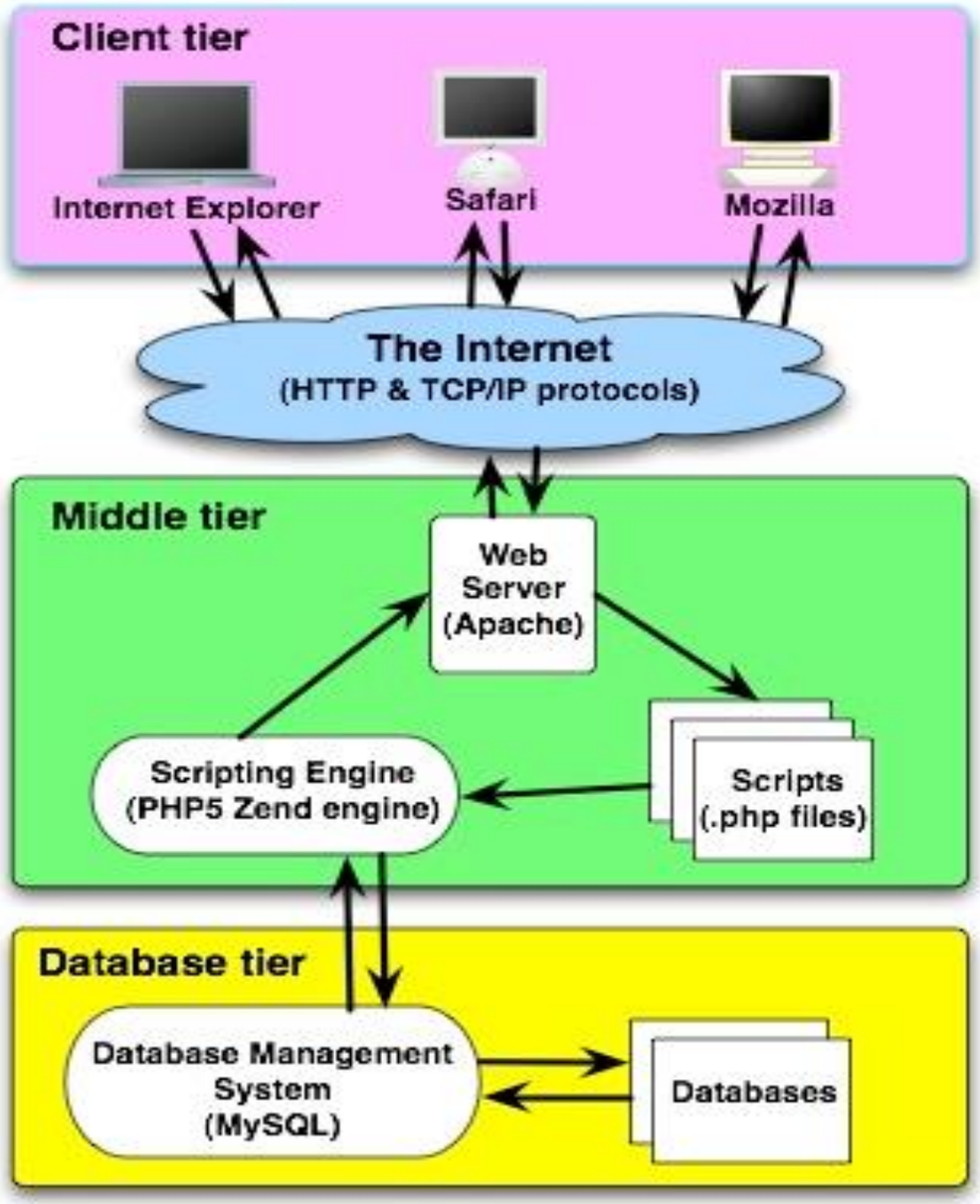
Beowulf Cluster Attributes

- Multi-Computer Architecture
  - Cluster Nodes: commodity PC's (each with its OS instance, memory address space, disk}
  - Cluster Interconnect: commodity LAN's, Ethernet, switches
  - Operating System: Unix, BSD, Linux (same OS running on each node)
  - Cluster Storage: local storage in each node, centralized shared storage
- Parallel Programming Model
  - "Clustering middleware:" a SW layer atop the nodes to orchestrate the nodes for users
  - Application programs do not see the cluster nodes, only interface with master node
  - PVM: (Oak Ridge NL) library installed on every node, runtime environment for message passing, task and resource management.
  - MPI: (ARPA, NSF) use TCP/IP and socket connection, now widely available, library to achieve high performance at low cost.

Example Clusters



Example Clusters



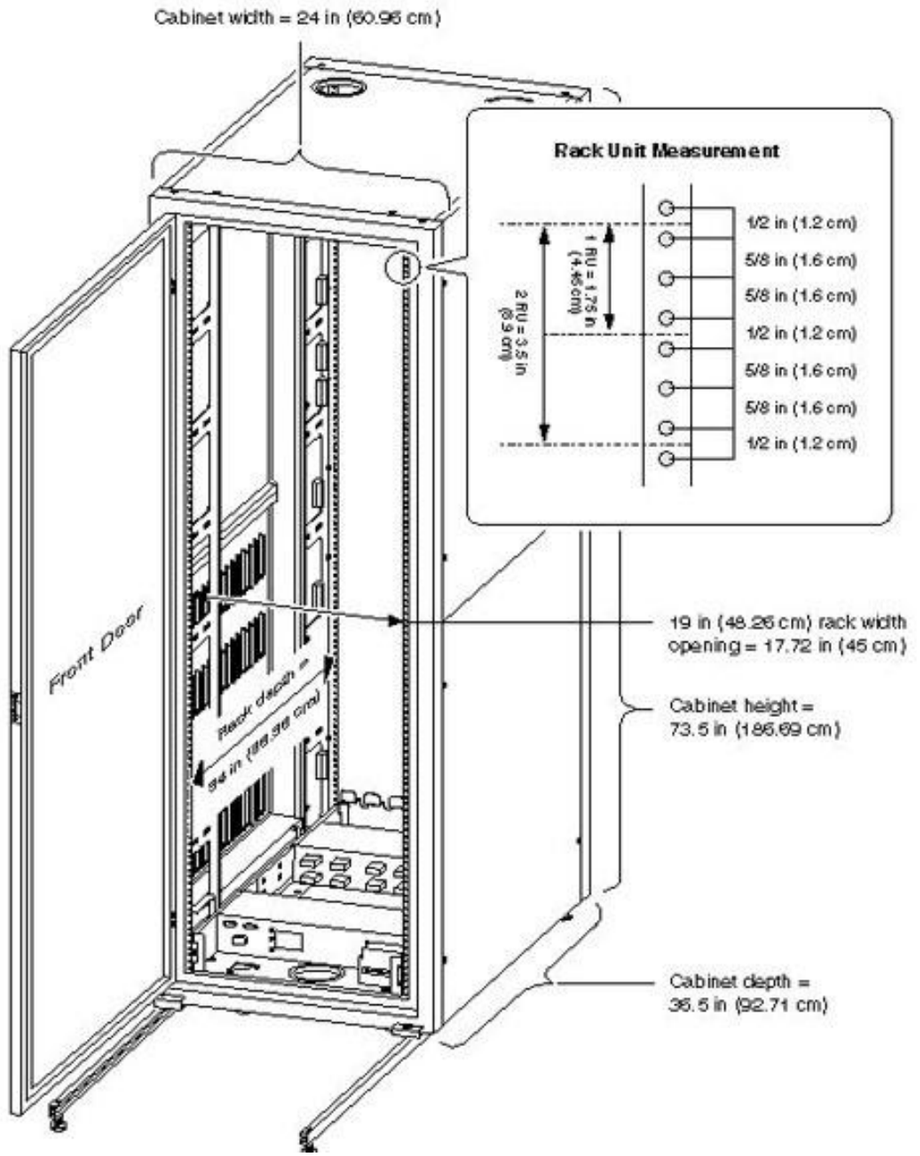
Physical Design

- Workstations (PCs) on a LAN

Physical Design

- Rack Mounted PC Boards

R-Cluster



Physical Design

- Blades
  - Shared power supply, cooling, etc.

45

Physical Design

- Sun's Project Black Box (Modular Datacenter)
  - "Portable computing" – up to 17 tons, 280 RU
  - BYOPS
  - No field-replaceable parts – why?

46

Applications

- Commercial
  - Large shared databases
  - Largely independent threads
  - “Architecture” often means software architecture
  - May use higher performance storage area network (SAN)
- Scientific
  - Inexpensive high performance computing
  - Based on message passing
  - Also PGAS (partitioned global address space); software shared memory
  - May use higher performance node-to-node network
  - Where HPC clusters end and MPPs begin isn’t always clear

Software Considerations

- Throughput Parallelism
  - As in many commercial servers
  - Distributed OS message passing
  - VAXcluster early example
- True Parallelism
  - As in many scientific/engineering applications
  - Use programming model and user-level API
- Programming Models
  - Message-Passing
    - Commonly used
  - Shared memory
    - Virtual Shared Memory, Software Distributed Shared Memory
    - PGAS – software abstraction, runtime invokes remote DMA
- Of course, a real system can do both throughput and true parallelism

Computer Cluster – Summary

- Reasons for clusters
  - Performance – horizontal scaling
  - Cost: commodity h/w, commodity s/w
  - Redundancy/fault tolerance
- Hardware Challenges
  - Cost, commodity components
  - Power, cooling, energy proportionality
  - Reliability, usability, maintainability
- Software Challenges
  - Programming model
  - Applications with suitable characteristics
  - Load balancing

Google Cluster

[Luiz André Barroso , Jeffrey Dean , Urs Hölzle, 2003]

Web Search For a Planet: The Google Cluster Architecture

Luiz Andre Barroso, Jeffrey Dean, Urs Holzle
Google

Google Cluster Architecture

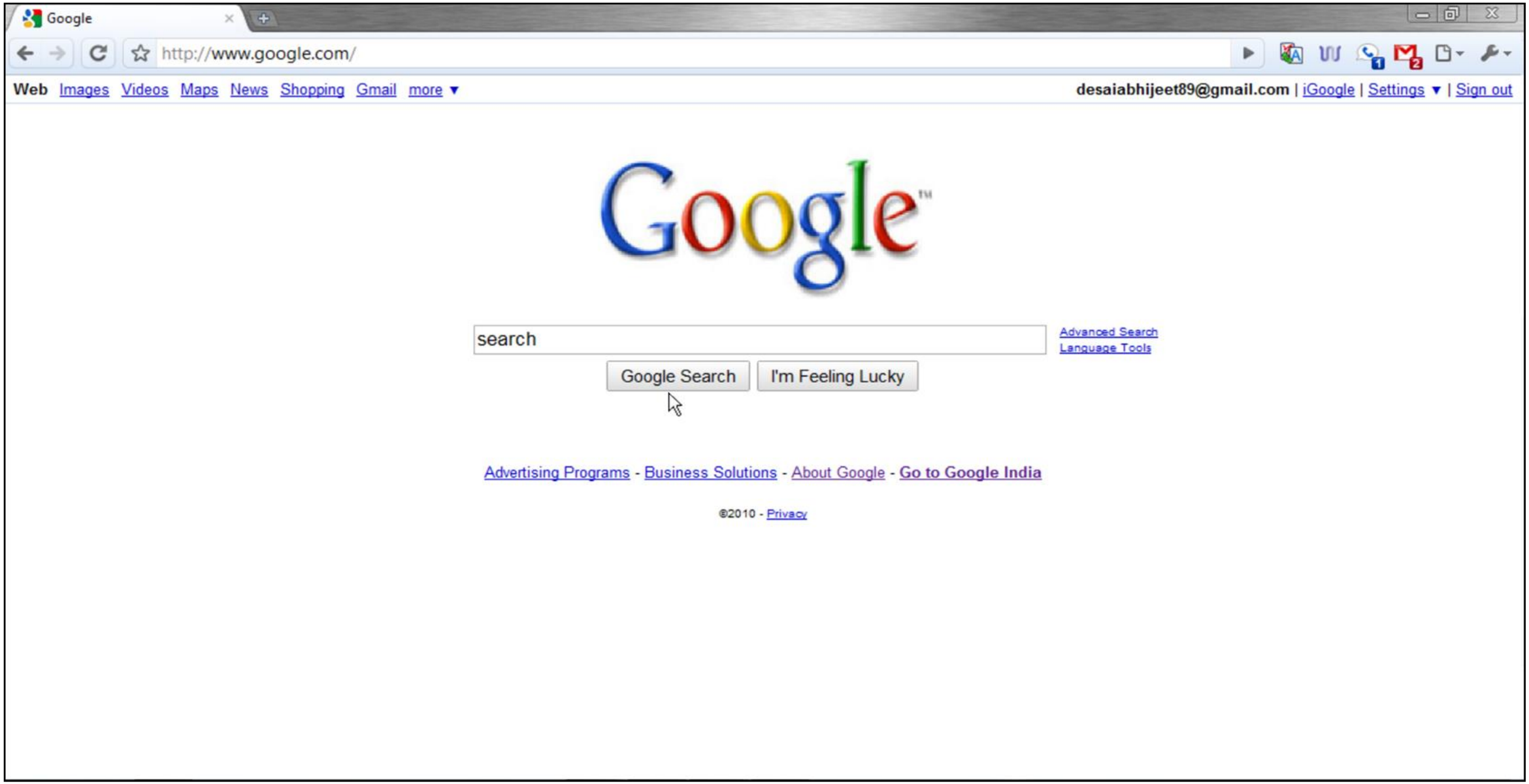
Web Search for a Planet and much more….

Abhijeet Desai
desaiabhijeet89@gmail.com

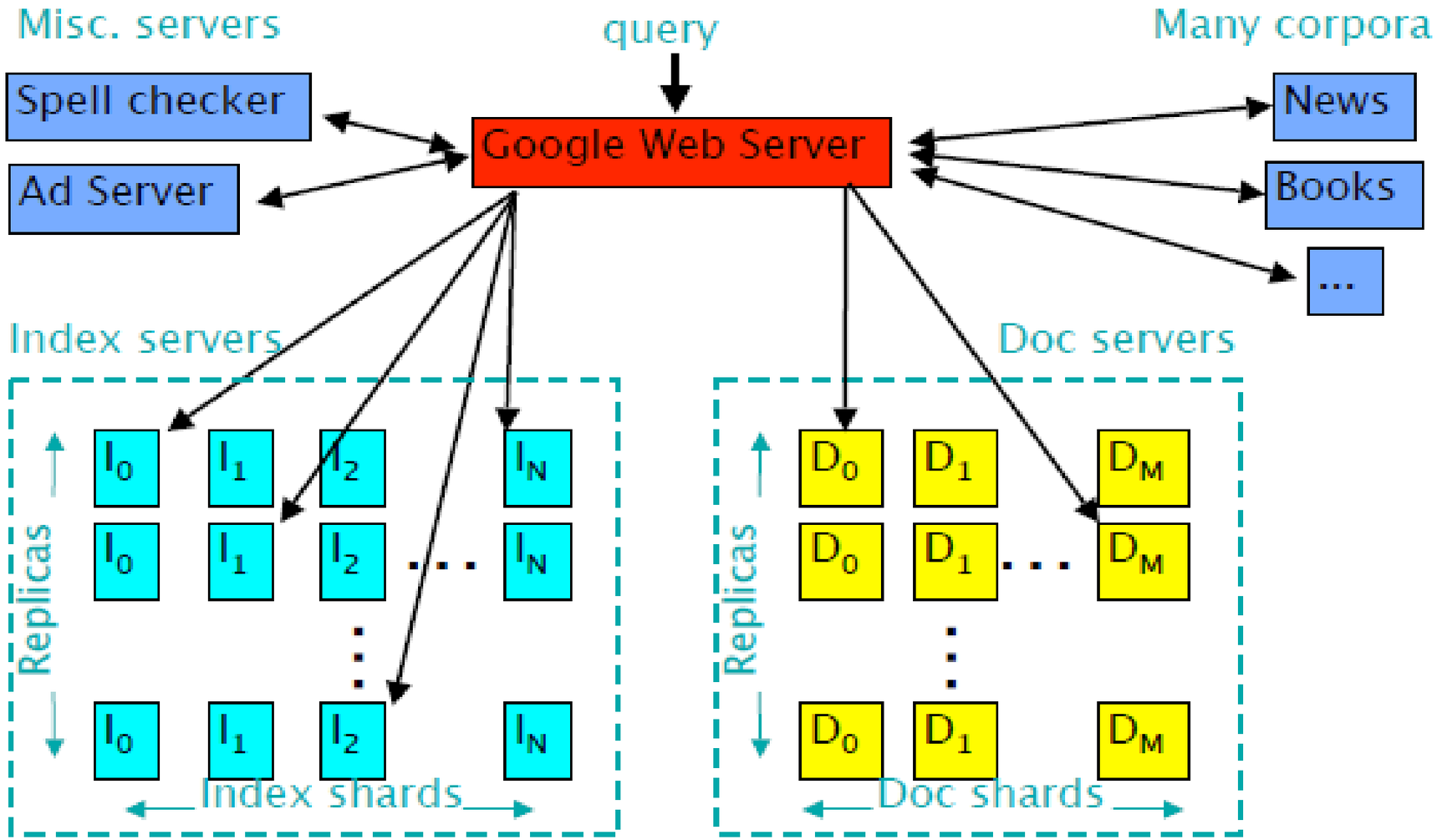
Electrical & Computer ENGINEERING

Design Principles of Google Clusters

- Software level reliability
  - No fault-tolerant hardware features; e.g.
    - redundant power supplies
    - A redundant array of inexpensive disks (RAID)
- Use replication
  - for better request throughput and availability
- Price/performance beats peak performance
  - CPUs giving the best performance per unit price
  - Not the CPUs with best absolute performance
- Using commodity PCs
  - reduces the cost of computation


Google
http://www.google.com/
Web Images Videos Maps News Shopping Gmail more ▼
desaiabhijeet89@gmail.com | iGoogle | Settings ▼ | Sign out
Google
search
Advanced Search
Language Tools
Google Search
I'm Feeling Lucky
Advertising Programs - Business Solutions - About Google - Go to Google India
©2010 - Privacy

Google Query Serving Infrastructure



Elapsed time: 0.25s, machines involved: 1000s+

Application Summary: Serving a Google Query

- Geographically distributed clusters
  - Each with many thousands of machines
- First perform Domain Name System (DNS) lookup
  - Maps request to a nearby cluster
- Send HTTP request to selected cluster
  - Request serviced locally w/in that cluster
- Clusters consist of Google Web Servers (GWSes)
  - Hardware-based load balancer distributes load among GWSes

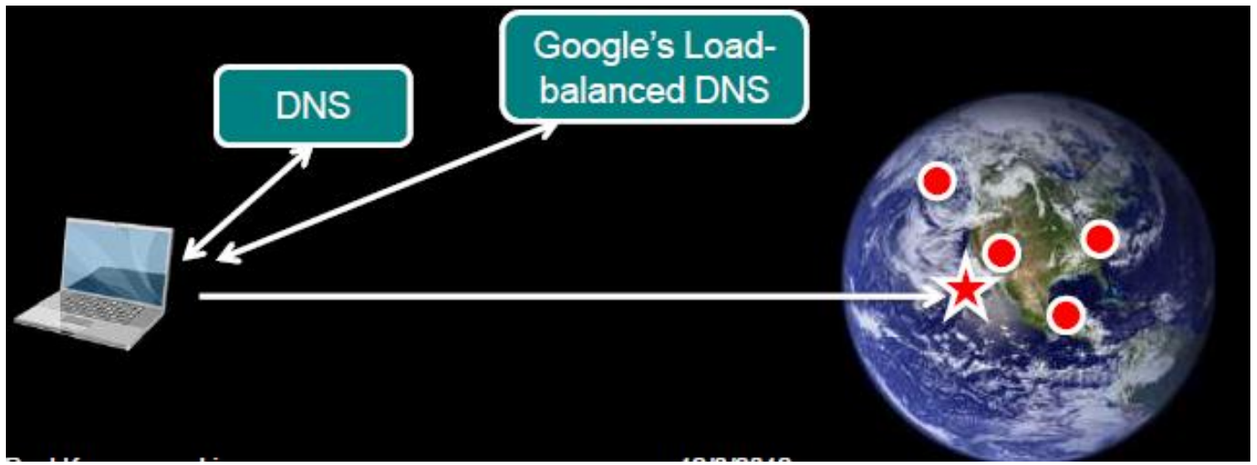
Query Execution

- Phase 1:
  - Index servers consult inverted index that maps query word to list of documents
  - Intersect hit lists and compute relevance score for each doc (*secret sauce*)
  - Results in ordered list of document ids (*docids*)
- Both documents and inverted index consume terabytes of data
- Index partitioned into "shards", shards are replicated
  - Index search becomes highly parallel; multiple servers per shard load balanced requests
  - Replicated shards add to parallelism and fault tolerance
- Phase 2:
  - Start w/ docids and determine title, resource locator, document summary
- Done by document servers
  - Partition documents into shards
  - Replicate on multiple servers
  - Route requests through load balancers

Step 1 – DNS

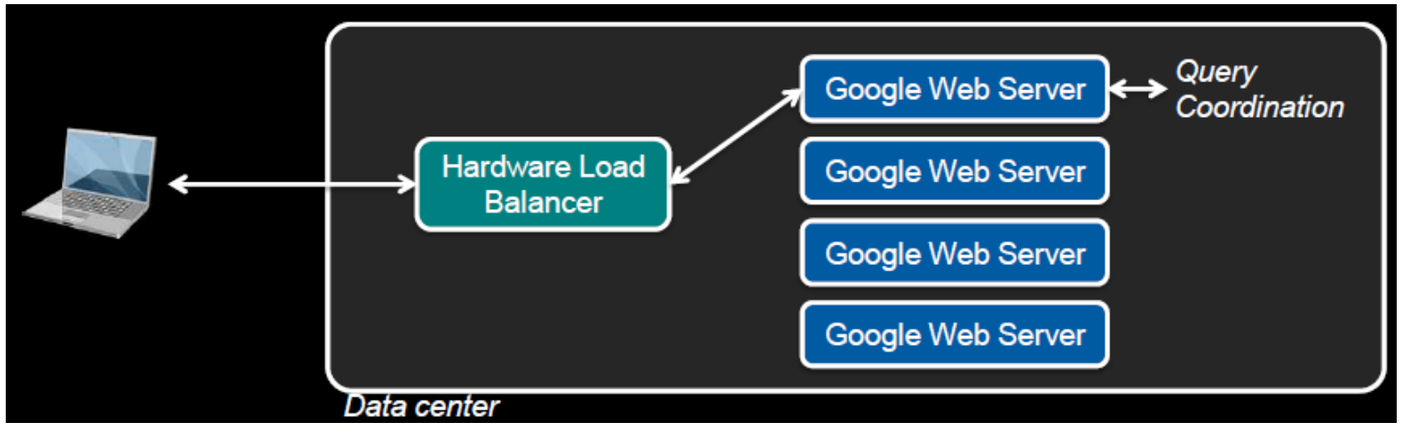
[Paul Krzyzanowski, 2012, Rutgers]

- User's browser must map google.com to an IP address
- "google.com" comprises
  - Multiple clusters distributed worldwide
  - Each cluster contains thousands of machines
- DNS-based load balancing
  - Select cluster by taking user's geographic proximity into account
  - Load balance across clusters
  - [similar to Akamai's approach]



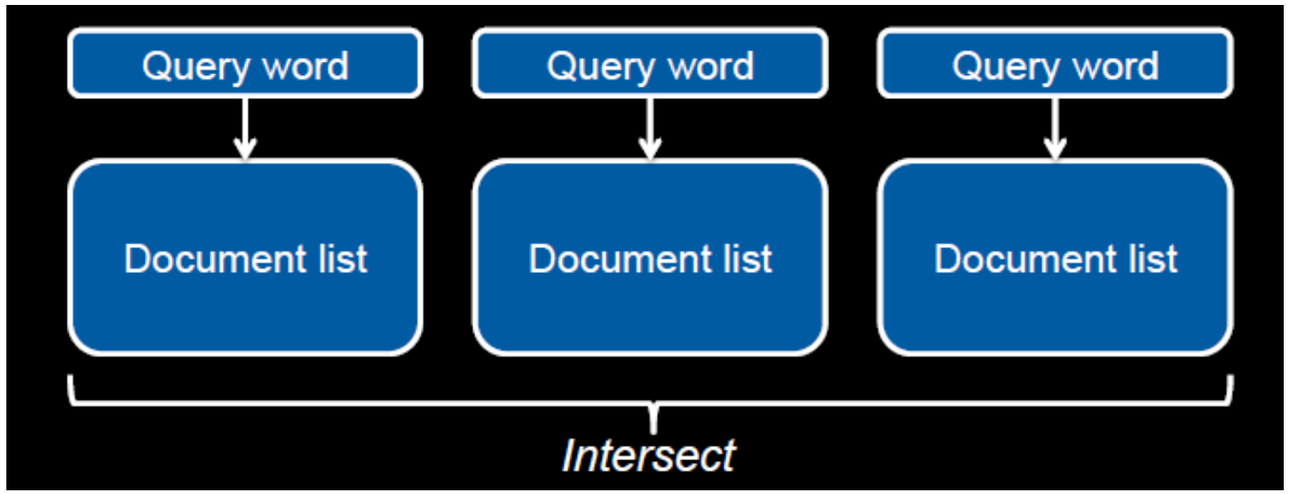
Step 2 – Send HTTP Request

- IP address corresponds to a load balancer within a cluster
- Load balancer
  - Monitors the set of Google Web Servers (GWS)
  - Performs local load balancing of requests among available servers
- GWS machine receives the query
  - Coordinates the execution of the query
  - Formats results into an HTML response to the user



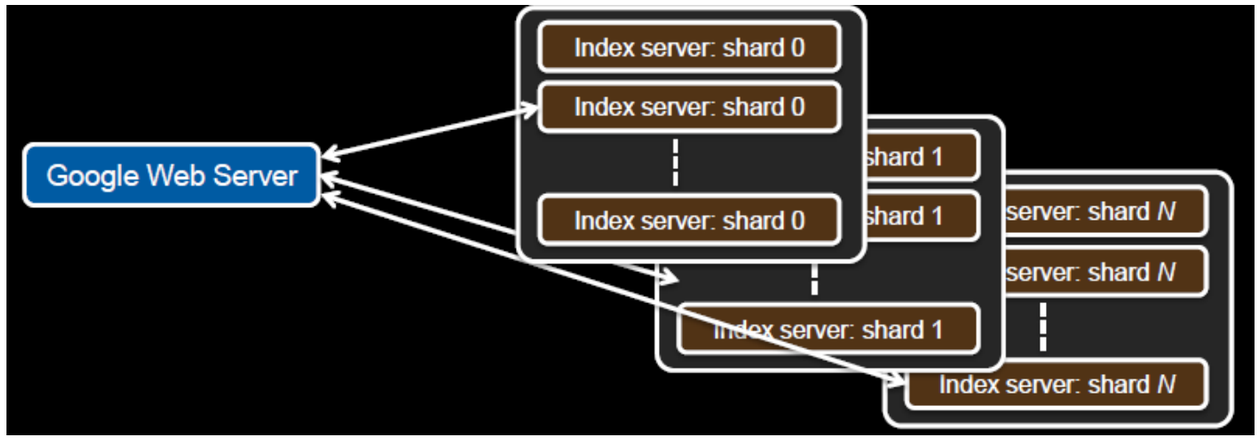
Step 3 – Find Document via Inverted Index

- Index Servers
  - Map each query word → {list of documents} (hit list)
    - Inverted index generated from web crawlers → MapReduce
  - Intersect the hit lists of each per-word query
    - Compute relevance score for each document
    - Determine set of documents
    - Sort by relevance score



Parallelizing the Inverted Index

- Inverted index is 10s of terabytes
- Search is parallelized
  - Index is divided into *index shards*
    - Each index shard is built from a randomly chosen subset of documents
    - Pool of machines serves requests for each shard
    - Pools are load balanced
  - Query goes to one machine per pool responsible for a shard
- Final result is ordered list of document identifiers (*docids*)

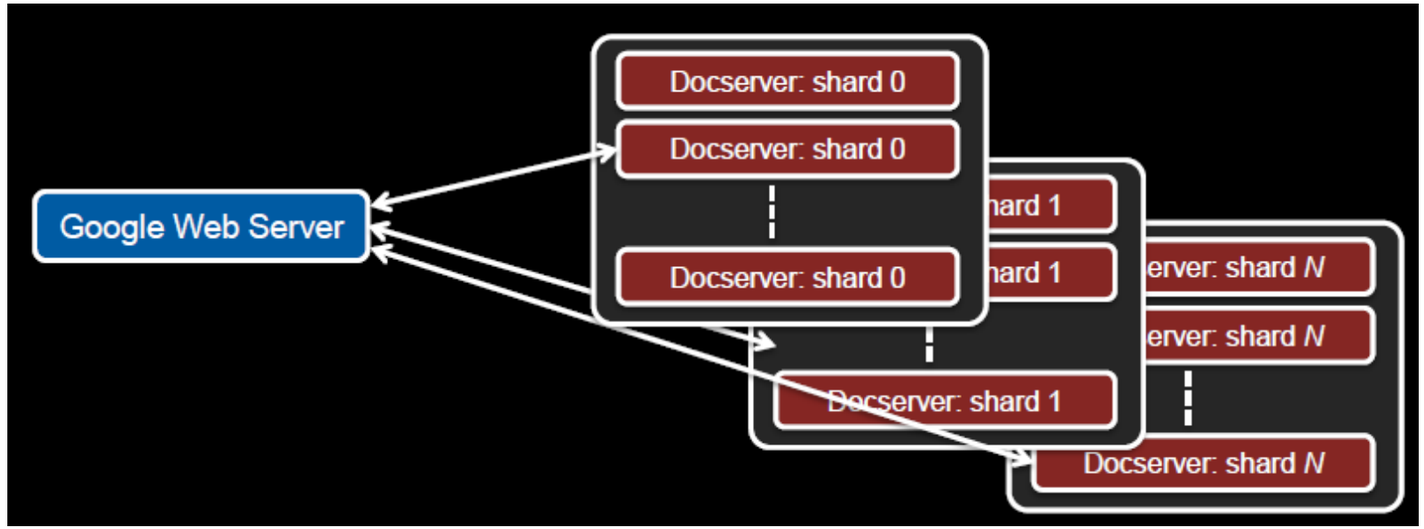


Step 4 – Get Title and URL for Each docid

- For each docid, the GWS looks up
  - Page title
  - URL
  - Relevant text: document summary specific to the query
- Handled by document servers (docservers)

Parallelizing Document Lookup

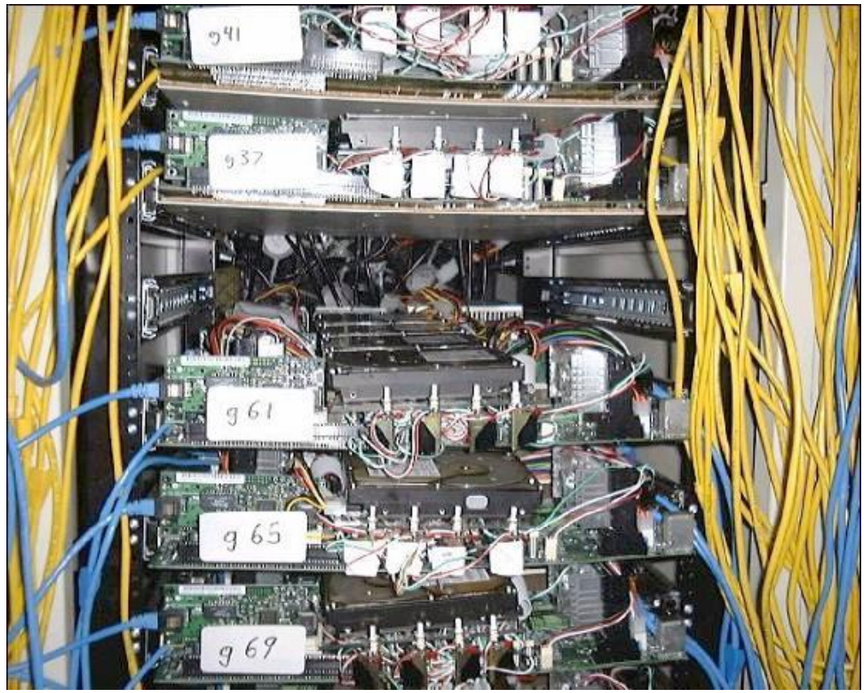
- Like index lookup, document lookup is partitioned & parallelized
- Documents distributed into smaller shards
  - Each shard = subset of documents
- Pool of load-balanced servers responsible for processing each shard
- Together, document servers access a cached copy of the entire web!



Google Clusters Through the Years

"Google" Circa 1997 (google.stanford.edu)

Google (circa 1999)



Google Clusters Through the Years

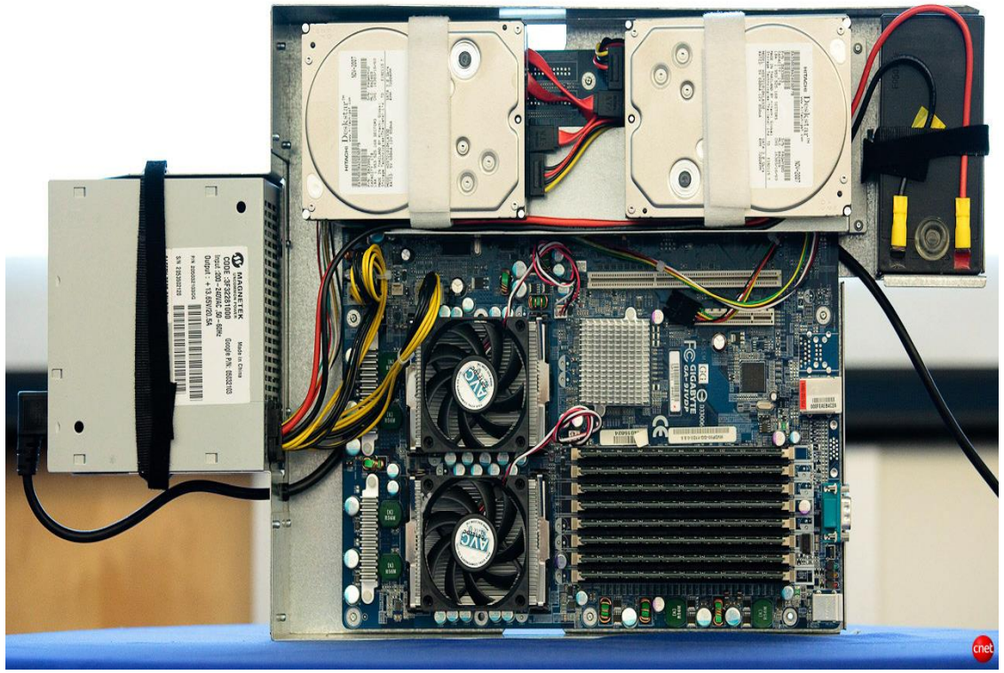
Google Data Center (Circa 2000)

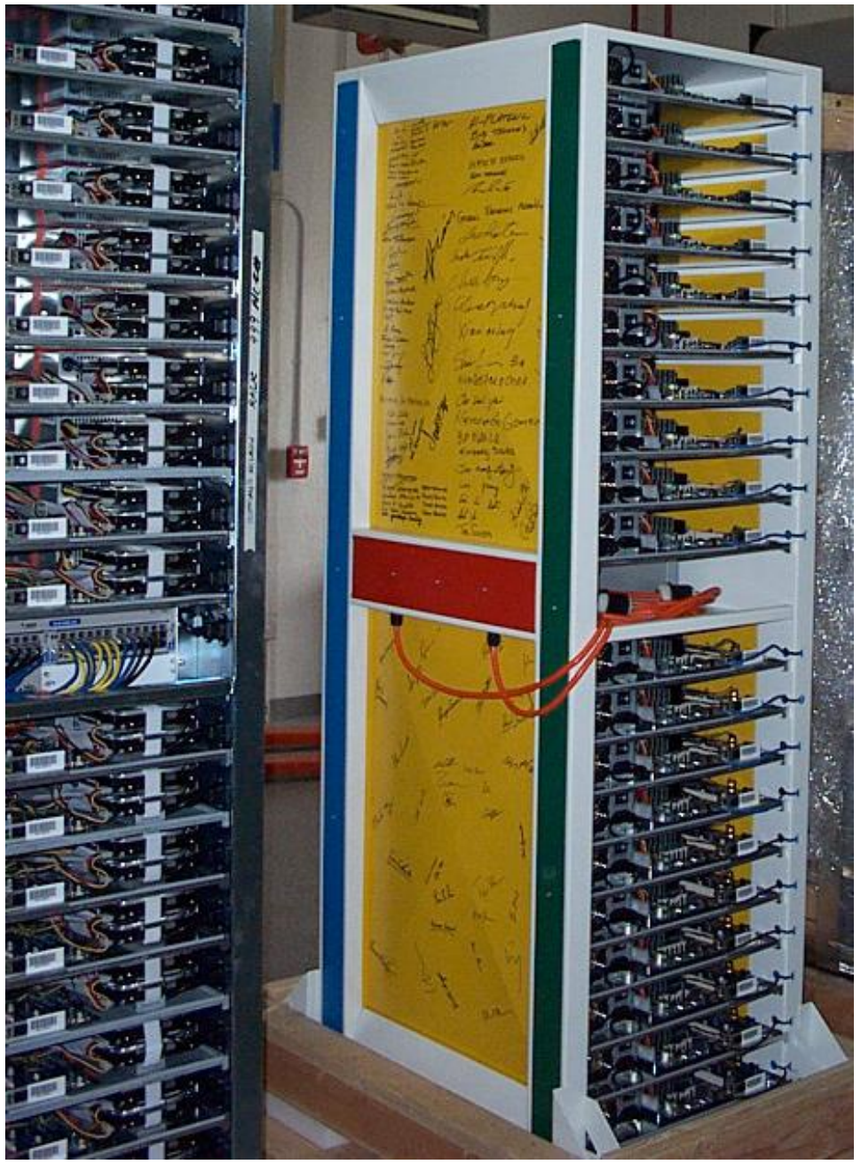
Google (new data center 2001)

3 days later

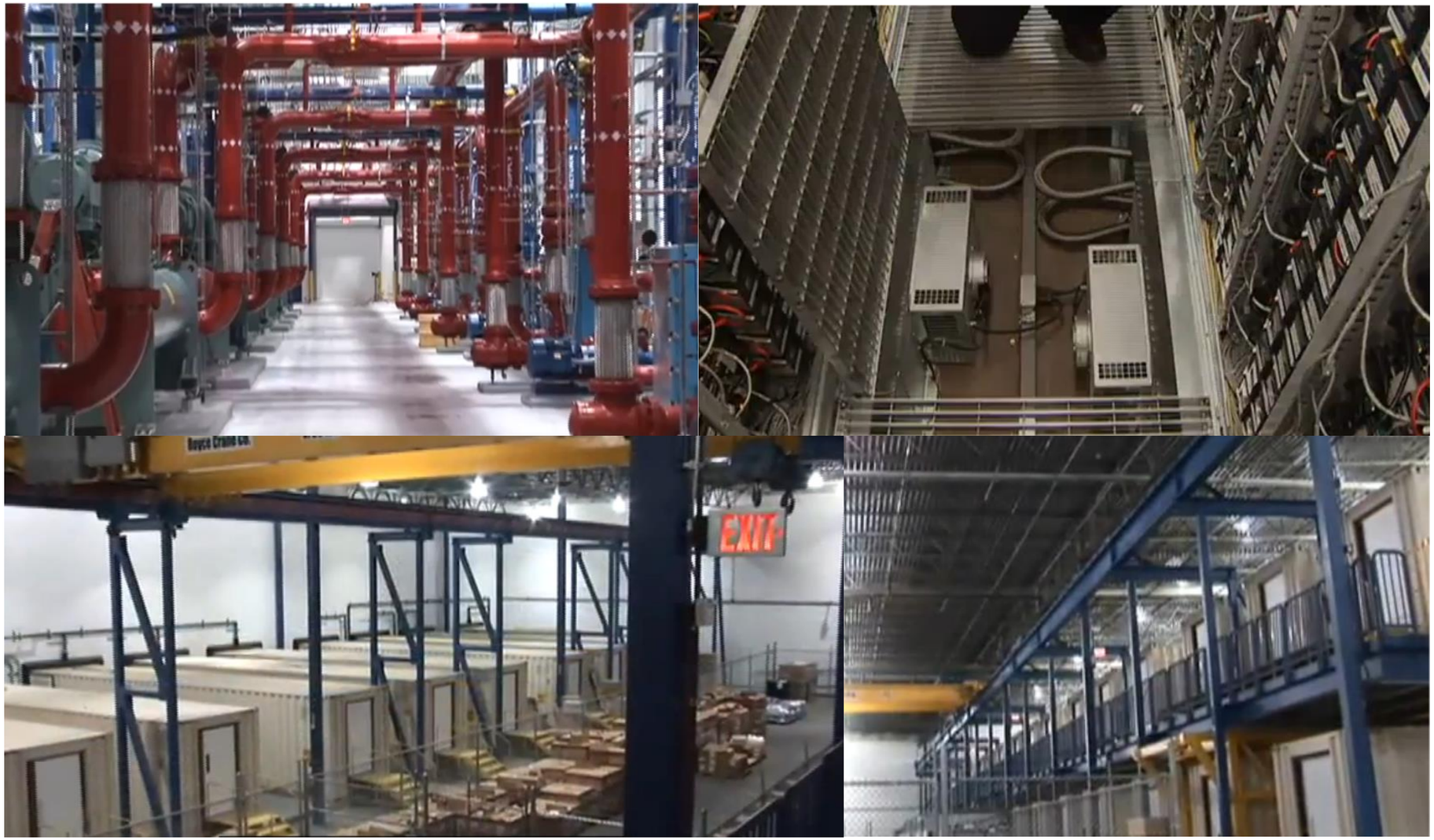
Recent Design

- In-house rack design
- PC-class motherboards
- Low-end storage and networking hardware
- Linux
- + in-house software





Container Datacenter



Container Datacenter

Comparison: Custom Built vs. High-end Servers

	Typical x86 - based server	Custom built x86 - based server	
PROCESSORS	8 2-GHz Xeon CPUs	176 2-GHz Xeon CPUs	22x
RAM	64 Gbytes of RAM	176 Gbytes of RAM	3x
DISK SPACE	8 Tbytes of disk space	7 Tbytes of disk space	-1 TB
PRICE	$758,000	$278,000	$480,000

The Google "Cloud"

[NYT: June 8, 2006]

Google Cluster – Conclusions

- For a large scale web service system like Google
  - Design the algorithm which can be easily parallelized
  - Design the architecture using replication to achieve distributed computing/storage and fault tolerance
  - Be aware of the power problem which significantly restricts the use of parallelism

- Several key pieces of infrastructure for search systems:
  - GFS
  - MapReduce
  - BigTable

Google Cluster – References

1. Luiz André Barroso , Jeffrey Dean , Urs Hölzle, Web Search for a Planet: The Google Cluster Architecture, IEEE Micro, v.23 n.2, p.22-28, March 2003 [doi>10.1109/MM.2003.1196112]
2. S. Brin and L. Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine," Proc. Seventh World Wide Web Conf. (WWW7), International World Wide Web Conference Committee (IW3C2), 1998, pp. 107-117.
3. "TPC Benchmark C Full Disclosure Report for IBM eserver xSeries 440 using Microsoft SQL Server 2000 Enterprise Edition and Microsoft Windows .NET Datacenter Server 2003, TPC-C Version 5.0," http://www.tpc.org/results/FDR/TPCC/ibm.x4408way.c5.fdr.02110801.pdf.
4. D. Marr et al., "Hyper-Threading Technology Architecture and Microarchitecture: A Hypertext History," Intel Technology J., vol. 6, issue 1, Feb. 2002.
5. L. Hammond, B. Nayfeh, and K. Olukotun, "A Single-Chip Multiprocessor," Computer, vol. 30, no. 9, Sept. 1997, pp. 79-85.
6. L.A. Barroso et al., "Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing," Proc. 27th ACM Int'l Symp. Computer Architecture, ACM Press, 2000, pp. 282-293.
7. L.A. Barroso, K. Gharachorloo, and E. Bugnion, "Memory System Characterization of Commercial Workloads," Proc. 25th ACM Int'l Symp. Computer Architecture, ACM Press, 1998, pp. 3-14.

Change to Caffeine

[Paul Krzyzanowski, 2012, Rutgers]

- In 2010, Google remodeled its search infrastructure
- Old system
  - Based on MapReduce (on GFS) to generate index files
  - Batch process: next phase of MapReduce cannot start until first is complete
    - Web crawling → MapReduce → propagation
  - Initially, Google updated its index every 4 months. Around 2000, it reindexed and propagated changes every month
    - Process took about 10 days
    - Users hitting different servers might get different results
- New system, named *Caffeine*
  - Fully incremental system: Based on BigTable running on GFS2
  - Support indexing many more documents: ~100 petabytes
  - High degree of interactivity: web crawlers can update tables dynamically
  - Analyze web continuously in small chunks
    - Identify pages that are likely to change frequently
  - BTW, MapReduce is not dead. Caffeine uses it in some places, as do lots of other services.

GFS to GFS2

[Paul Krzyzanowski, 2012, Rutgers]

- GFS was designed with MapReduce in mind
  - But found lots of other applications
  - Designed for batch-oriented operations
- Problems
  - Single *master node* in charge of chunkservers
  - All info (metadata) about files is stored in the master's memory – limits total number of files
  - Problems when storage grew to tens of petabytes (10^{12} bytes)
  - Automatic failover added (but still takes 10 seconds)
  - Designed for high throughput but delivers high latency: master can become a bottleneck
  - Delays due to recovering from a failed replica chunkserver delay the client
- GFS2
  - Distributed masters
  - Support smaller files: chunks go from 64 MB to 1 MB
  - Designed specifically for BigTable (does not make GFS obsolete)

More References

[Paul Krzyzanowski, 2012, Rutgers]

- *Web Search for a Planet: The Google Cluster Architecture*
  Luiz André Barroso, Jeffrey Dean, Urs Hölzle Google, Inc.
  research.google.com/archive/googlecluster.html

- *Our new search index: Caffeine*
  - The Official Google Blog
  - http://googleblog.blogspot.com/2010/06/our-new-search-index-caffeine.html
- *GFS: Evolution on Fast-forward*
  - Marshall Kirk McKusick, Sean Qunlan
  - Association for Computing Machinery, August 2009
  - http://queue.acm.org/detail.cfm?id=1594206
- *Google search index splits with MapReduce*
  - Cade Metz
  - The Register, September 2010
  - http://Sept/2009/08/12/google_file_system_part_deux/
- *Google File System II: Dawn of the Multiplying Master Nodes*
  - Cade Metz
  - The Register, August 2009
  - http://www.theregister.co.uk/2010/09/09/google_caffeine_explained/
- *Exclusive: How Google's Algorithm Rules the Web*
  - Steven Levy
  - Wired Magazine, March 2010
  - http://www.wired.com/magazine/2010/02/ff_google_algorithm/all/1

18-600 Foundations of Computer Systems

Lecture 21: "Multicore Cache Coherence"

John P. Shen & Zhiyi Yu

November 14, 2016

Next Time ...

- Recommended References:
  - "Chip Multiprocessor Architecture: Techniques to Improve Throughput and Latency" by Kunle Olukotun, Lance Hammond, and James Laudon in Synthesis Lectures on Computer Architecture, Morgan & Claypool, 2007.
  - "Parallel Computer Organization and Design," by Michel Dubois, Murali Annavaram, Per Stenstrom, Chapters 5 and 7, 2012.

Electrical & Computer ENGINEERING