CS 61C: Great Ideas in Computer Architecture

Virtual Memory Problems, Warehouse-Scale Computers

Instructor: Alan Christopher

Agenda

- Virtual Memory Review
- Administrivia
- Virtual Memory Problems
  - Example Final Question
- Warehouse Scale Computers
- Request Level Parallelism

Paging Summary

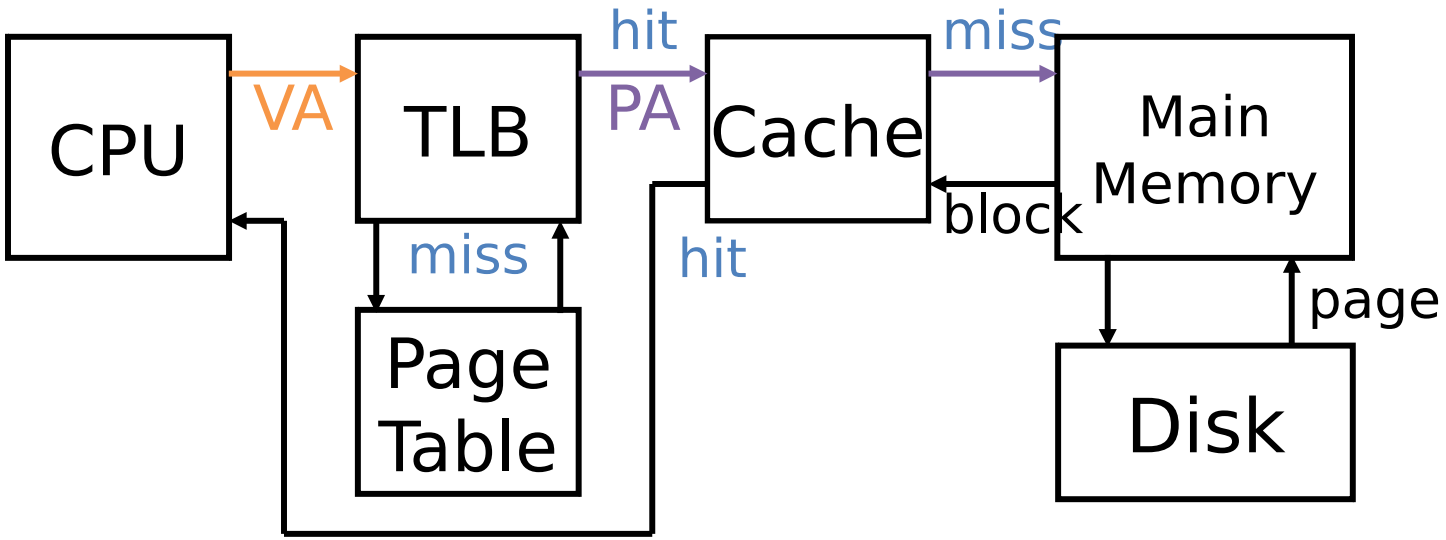
- Paging requires address *translation*
  - Can run programs larger than main memory
  - Hides variable machine configurations (RAM/HDD)
  - Solves fragmentation problem
- Address mappings stored in page tables in memory
  - Additional memory access mitigated with TLB
  - Check TLB, then Page Table (if necessary), then Cache

Page Table vs. TLB

- Data: PPN
- Return mapping to TLB
- Location: Memory
- Size: every VPN
- Management Bits
  - Valid, Access Rights, Dirty, (Ref)
- Invalidation
- Max valid entries set by size of PM

- Data: PPN
- Send PA to Cache
- Location: Separate HW
- Size: design choice
- Management Bits
  - Valid, Access Rights, (Ref), Dirty, TLB Tag
- Replacement
- All entries can be valid

Data Fetch (1/2)



Virtual Address To TLB:

Physical Address From TLB:

To Cache:

Data Fetch (2/2)

1) Check TLB (input: VPN, output: PPN)
   - *TLB Hit:* Fetch translation, return PPN
   - *TLB Miss:* Check page table (in memory)
     - *Page Table Hit:* Load page table entry into TLB
     - *Page Table Miss (Page Fault):* Fetch page from disk to memory, update corresponding page table entry, then load entry into TLB
2) Check cache (input: PPN, output: data)
   - *Cache Hit:* Return data value to processor
   - *Cache Miss:* Fetch data value from memory, store it in cache, return it to processor

Data Consistency

- Meaning of set Dirty Bit:
  - In $: cache block more updated than PM
  - In PT/TLB: PM page more updated than Disk
- If TLB is write-back, then *always* update PT entry on TLB entry replacement
- If page not present in PM, data cannot be in $
  - Need to invalidate corresponding entries in $ on page replacement

Question: Assume the page table entry in question is present in the TLB and we are using a uniprocessor system. Are the following statements TRUE or FALSE?

1) The valid bit for that page must be the same in the PT and TLB
2) The dirty bit for that page must be the same in the PT and TLB

	1	2
(B)	F	F
(G)	F	T
(P)	T	F
(Y)	T	T

Running Multiple Processes

- *Context switch:* Changing of internal state of processor (switching between processes)
  - Save register values (and PC) and change value in Page Table Base register
- Invalidate TLB entries (for different VAs!)
- Can leave cache entries (PAs) as long as corresponding pages stay in PM
- Different processes *can* share pages
  - Access Rights may differ

Virtual Memory Terminology

- Virtual Address (VA)
  - Virtual Memory (VM)
  - Virtual Page Number (VPN)
  - Page Offset (PO)
  - TLB Tag
  - TLB Index
- Physical Address (PA)
  - Physical Memory (PM)
  - Physical Page Number (PPN)
  - Page Offset (PO)
  - Tag, Index, Offset
- Page Table (PT) and Translation Lookaside Buffer (TLB)
  - Valid (V), Dirty (D), Ref (R), Access Rights (AR)
  - TLB Hit/Miss
  - PT Hit, Page Fault
  - TLB/PT entries
- OS Tasks:
  - Swap Space
  - Page Table Base Register
  - Context Switching

Agenda

- Virtual Memory Wrap-Up
- Administrivia
- Virtual Memory Problems
  - Example Final Question
- Warehouse Scale Computers
- Request Level Parallelism

Administrivia

- Project 3 (partners) due Sunday 8/10
- Final Review – Sat 8/09, 2-5pm in 2060 VLSB
- Final – Fri 8/15, 9am-12pm, 155 Dwinelle
  - MIPS Green Sheet provided again
  - Two two-sided handwritten cheat sheets
    - Can re-use your midterm cheat sheet!

Agenda

- Virtual Memory Wrap-Up
- Administrivia
- Virtual Memory Problems
  - Example Final Question
- Warehouse Scale Computers
- Request Level Parallelism

Anatomy of a VM Question

- Almost identical to cache questions:
  - Address breakdown (VPN, PPN, TLB TIO)
  - For fixed parameters, analyze the performance of the TLB and/or PT for the given code/sequence
  - For fixed parameters, find best or worst case scenarios
  - For given code/sequence, how does changing VM parameters affect performance?
  - AMAT/CPI

VM Parameters

- Important VM parameters
  - Size of VM & PM, page size
  - TLB size/num entries, associativity, replacement policy
  - Solve for TLB TIO breakdown, VPN, PPN, width of TLB entry
- Initial state of TLB, PT, and PM?
  - Not always specified (best/worst case)

Access Patterns

- How many hits within a single page once it is loaded into PM?
- Will page still be in PM when you revisit its elements?
  - Do you have to account for context switching?
- Will there be a *protection fault* when you try to access the page?
- Are there special/edge cases to consider?
  - Usually edge of page boundary or edge of TLB reach boundary or edge of PM size

Example: (Sp13 Final F2)

- 32-bit VAs, 1 MiB pages, 512 MiB PM with LRU, fully associative TLB with 32 entries and LRU
- First thing! Solve for parameters:
  - PAs are $\log_2(512\ \text{Mi}) = 29$ bits
  - Page Offset = $\log_2(1\ \text{Mi}) = 20$ bits
  - VPN = 32 – 20 = 12 bits, PPN = 29 – 20 = 9 bits
  - TLB Reach = 32 * 1 MiB = 32 MiB

Example: (Sp13 Final F2)

32-bit VAs, 1 MiB pages, 512 MiB PM with LRU, fully associative TLB with 32 entries and LRU

a) How many entries does a page table contain?

Page table has 1 entry per page in *virtual* memory.

2^{32} B VM / 1 MiB pages = 2^{12} = 4 Ki-entries in PT

c) How wide is the page table base register?

PTBR holds the address of a page table, which sits in PM, so it must be at least as wide as a PA.

Already solved for PA = 29 bits

Example: (Sp13 Final F2)

32-bit VAs, 1 MiB pages, 512 MiB PM with LRU, fully associative TLB with 32 entries and LRU

```
int histogram[MAX_SCORE];
void update_hist(int *scores, int num_scores) {
for (int i = 0; i < num_scores; i++)
    histogram[scores[i]] += 1;
}
```

TWO array accesses

Potentially discontinuous accesses

Only this many iters

Read AND Write

Assume that only the code and the two arrays take up memory, ALL of the program's code fits in 1 page, the arrays are page-aligned, and this is the only process running.

Example: (Sp13 Final F2)

32-bit VAs, 1 MiB pages, 512 MiB PM with LRU, fully associative TLB with 32 entries and LRU

c) If `update_hist()` were called with `num_scores = 10`, how many page faults can occur in the worst-case scenario?

Worst case: TLB cold except for code page.

Only access 40 B (continuous) in `scores[]`, so fits in 1 page.

`histogram[]` accesses can jump, so worst case each in a different page.

1 page fault for `scores`, 10 for `histogram` = 11 page faults

Example: (Sp13 Final F2)

32-bit VAs, 1 MiB pages, 512 MiB PM with LRU, fully associative TLB with 32 entries and LRU

d) In the best-case scenario, what is the max `num_scores` before a TLB miss?

Best case: TLB starts with code page and pages of `scores[]` and `histogram[]`.

Best case: All accesses to `histogram[]` are to same page.

Index to `scores[]` increments, so assume rest of TLB (30 pages) filled with pages of `scores[]`.

30 pages holds 30*(1 MiB/4 B) = $30*2^{18}$ ints

Example: (Sp13 Final F2)

- 32-bit VAs, 1 MiB pages, 512 MiB PM with LRU, fully associative TLB with 32 entries and LRU

e) For a particular data set, you know the scores are clustered around 50 different values, but you still observe a high # of TLB misses during `update_hist()`. What pre-processing step could help reduce the # of TLB misses?

Data set is mostly repeats of a few scores.

Want to improve temporal locality of the data.

Sort the scores!

Questions?

Technology Break

Agenda

- Virtual Memory Wrap-Up
- Administrivia
- Virtual Memory Problems
  - Example Final Question
- Warehouse Scale Computers
- Request Level Parallelism

Why Cloud Computing Now?

- "The Web Space Race": Build-out of extremely large datacenters (10,000's of ***commodity*** PCs)
  - Build-out driven by growth in demand (more users)
  - Infrastructure software and Operational expertise
- Discovered economy of scale: 5-7x cheaper than provisioning a medium-sized (1000 servers) facility
- More pervasive broadband Internet so can access remote computers efficiently
- Commoditization of HW & SW
  - Standardized software stacks

Supercomputer for Hire

- One or many?
  - Supercomputer competition: 42nd place = $700/hr
  - Credit card → Can use 1000s of computers for cheaper!
- Ex: FarmVille on Amazon Web Services (AWS)
  - Prior biggest online game had 5M users
  - 4 days = 1M; 2 months = 10M; 9 months = 75M
  - What if had to build own datacenter?

March 2013 AWS Instances & Prices

Instance	Per Hour	Ratio to Small	Compute Units	Virtual Cores	Compute Unit/ Core	Memory (GiB)	Disk (GiB)	Address
Standard Small	$0.065	1.0	1.0	1	1.00	1.7	160	32 bit
Standard Large	$0.260	4.0	4.0	2	2.00	7.5	850	64 bit
Standard Extra Large	$0.520	8.0	8.0	4	2.00	15.0	1690	64 bit
High-Memory Extra Large	$0.460	5.9	6.5	2	3.25	17.1	420	64 bit
High-Memory Double Extra Large	$0.920	11.8	13.0	4	3.25	34.2	850	64 bit
High-Memory Quadruple Extra Large	$1.840	23.5	26.0	8	3.25	68.4	1690	64 bit
High-CPU Medium	$0.165	2.0	5.0	2	2.50	1.7	350	32 bit
High-CPU Extra Large	$0.660	8.0	20.0	8	2.50	7.0	1690	64 bit

- Closest computer in WSC example is Standard Extra Large
- At these low rates, Amazon EC2 can make money!
  - even if used only 50% of time

Warehouse Scale Computers

- Massive scale datacenters: 10,000 to 100,000 servers + networks to connect them together
  - Emphasize cost-efficiency
  - Attention to power: distribution and cooling
- (relatively) homogeneous hardware/software
- Offer very large applications (Internet services): search, social networking, video sharing
- Very highly available: < 1 hour down/year
  - Must cope with failures common at scale
- "…WSCs are no less worthy of the expertise of computer systems architects than any other class of machines" (Barroso and Hoelzle, 2009)

Design Goals of a WSC

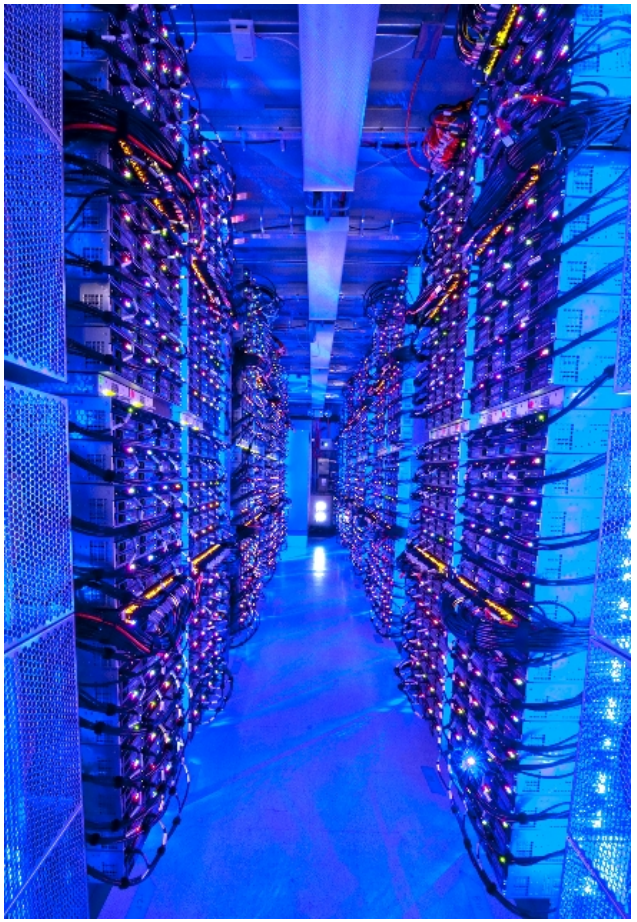
- Unique to Warehouse-scale
  - *Ample parallelism*:
    - Batch apps: large number independent data sets with independent processing.
  - *Scale and its Opportunities/Problems*
    - Relatively small number of WSC make design cost expensive and difficult to amortize
    - But price breaks are possible from purchases of very large numbers of commodity servers
    - Must also prepare for high component failures
  - *Operational Costs Count*:
    - Cost of equipment purchases << cost of ownership

Google’s Oregon WSC

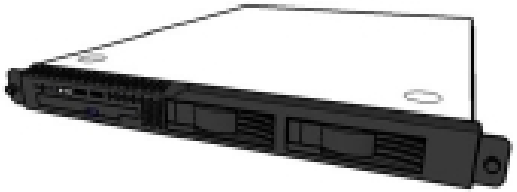
Containers in WSCs

Inside WSC

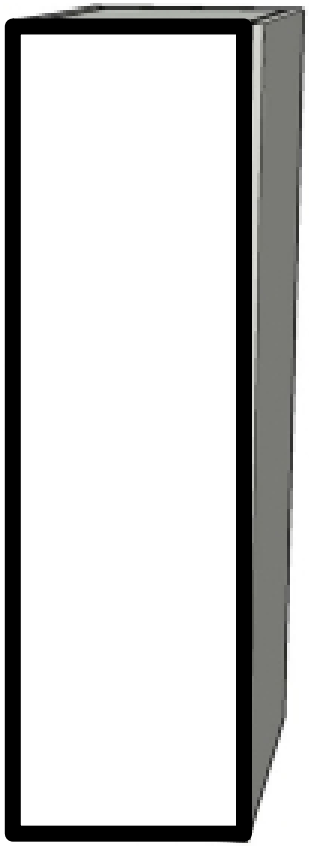
Inside Container



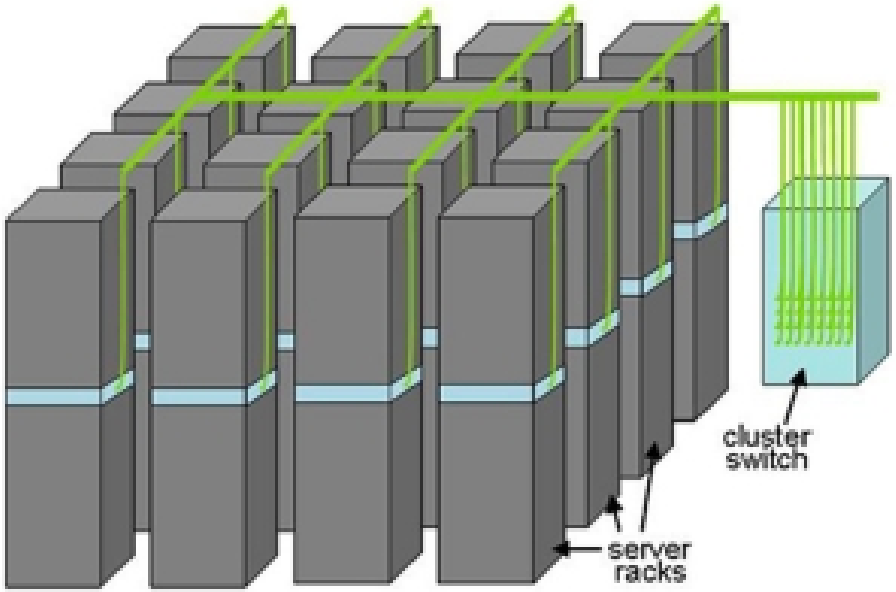
Equipment Inside a WSC



Server (in rack format):
1 ¾ inches high “1U”,
x 19 inches x 16-20
inches: 8 cores, 16 GB
DRAM, 4x1 TB disk



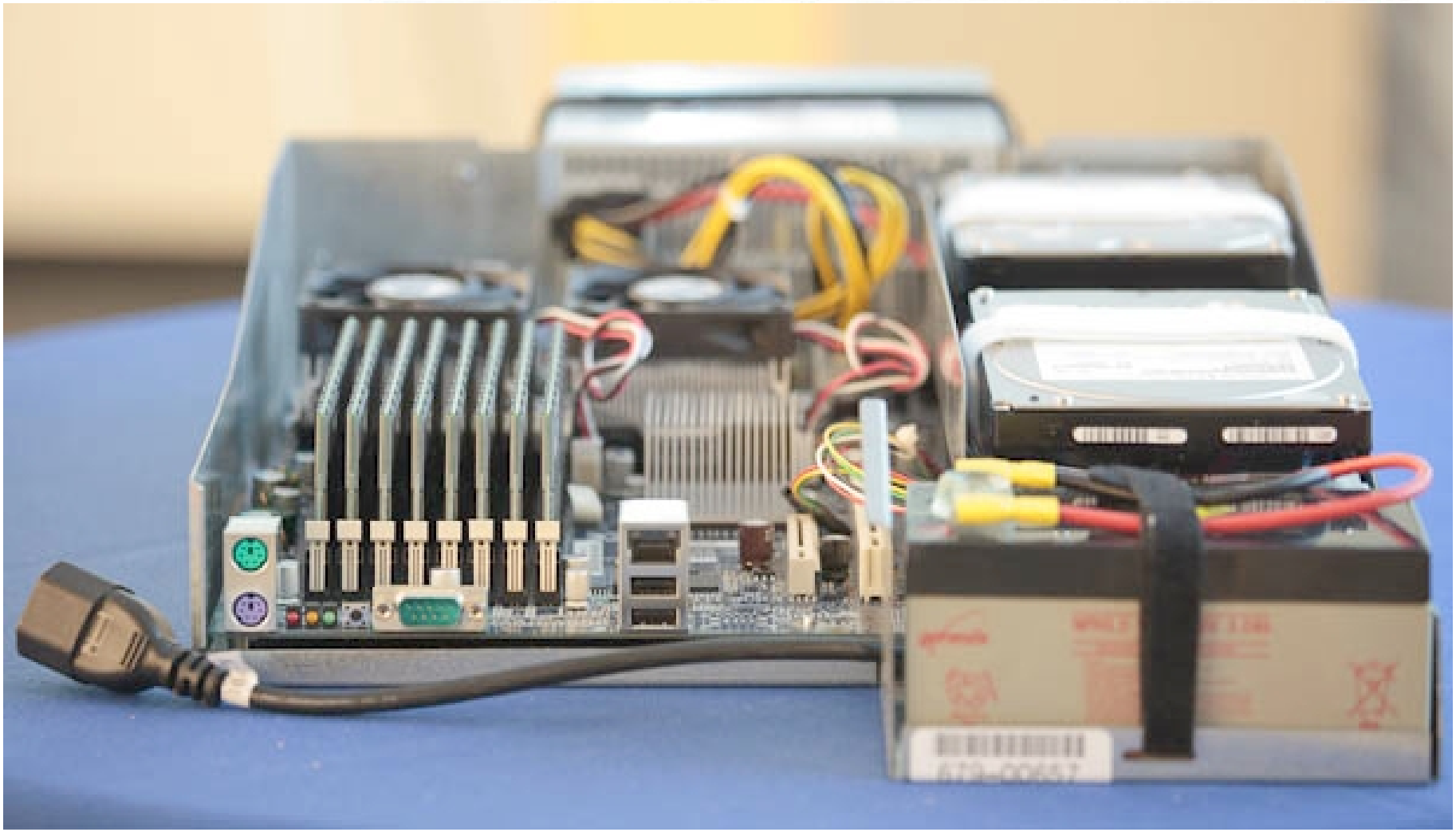
7 foot Rack: 40-80 servers + Ethernet local area network (1-10 Gbps) switch in middle (“rack switch”)



Array (aka cluster):
16-32 server racks + larger local area network switch (“array switch”) 10X faster => cost 100X: cost f(N^2)

Server, Rack, Array

Google Server Internals



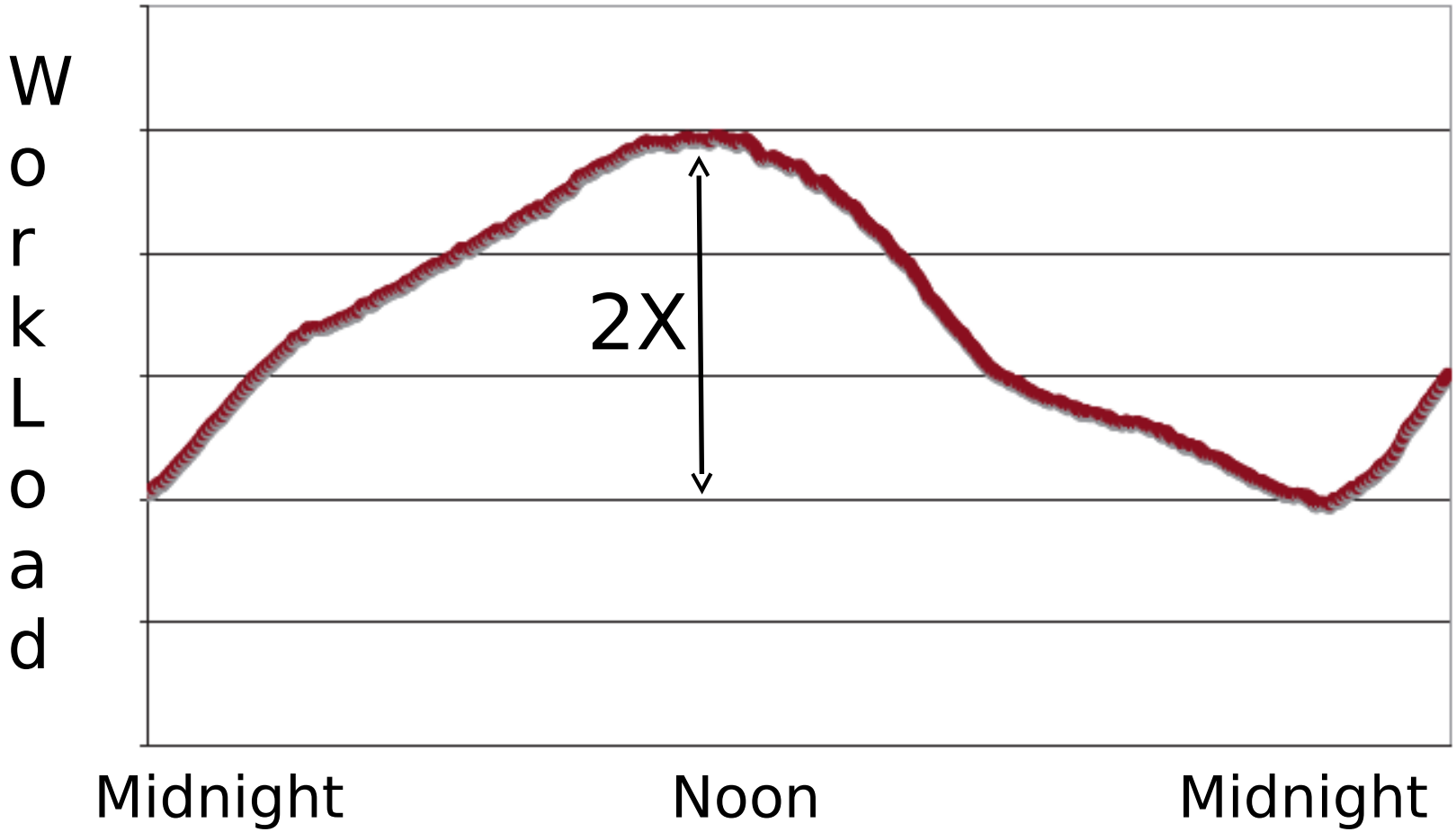
Coping with Performance in Array

Lower latency to DRAM in another server than local disk

Higher bandwidth to local disk than to DRAM in another server

	Local	Rack	Array
Racks	--	1	30
Servers	1	80	2400
Cores (Processors)	8	640	19,200
DRAM Capacity (GB)	16	1,280	38,400
Disk Capacity (GB)	4,000	320,000	9,600,000
DRAM Latency (microseconds)	0.1	100	300
Disk Latency (microseconds)	10,000	11,000	12,000
DRAM Bandwidth (MB/sec)	20,000	100	10
Disk Bandwidth (MB/sec)	200	100	10

Coping with Workload Variation

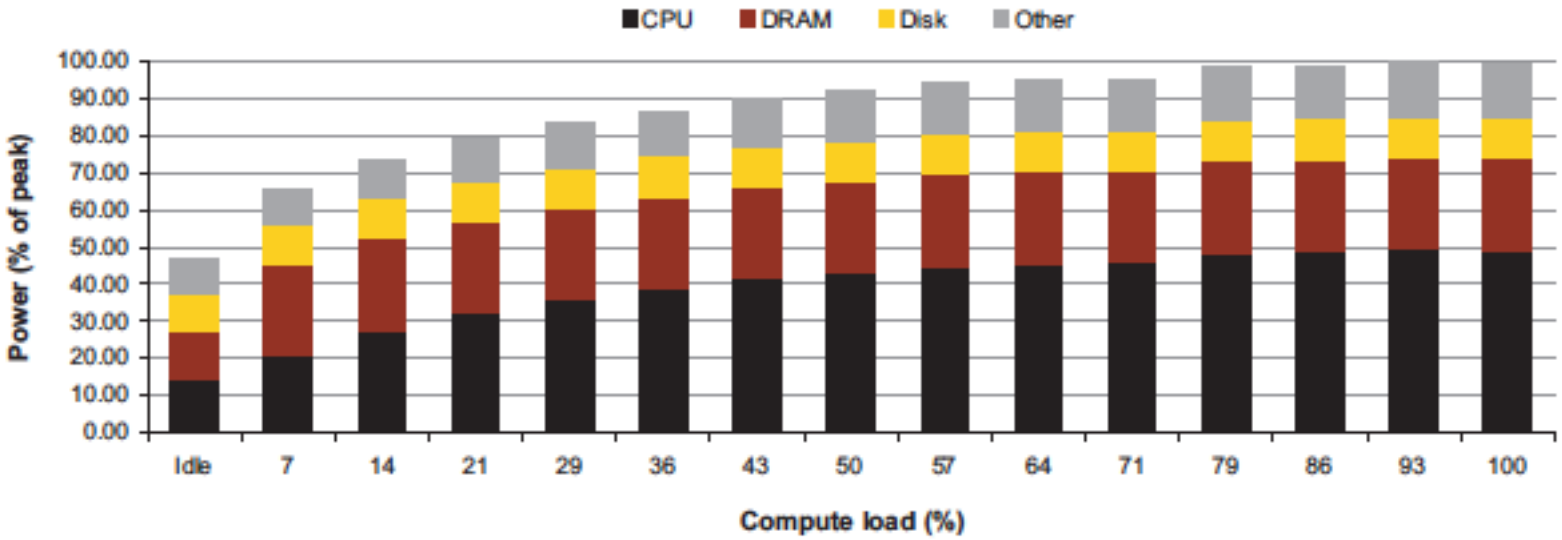


- Online service: Peak usage 2X off-peak

Impact of latency, bandwidth, failure, varying workload on WSC software?

- WSC Software must take care where it places data within an array to get good performance
- WSC Software must cope with failures gracefully
- WSC Software must scale up and down gracefully in response to varying demand
- More elaborate hierarchy of memories, failure tolerance, workload accommodation makes WSC software development more challenging than software for single computer

Power vs. Server Utilization



- Server power usage as load varies idle to 100%
- Uses ½ peak power when idle!
- Uses ⅔ peak power when 10% utilized! 90%@ 50%!
- Most servers in WSC utilized 10% to 50%
- Goal should be *Energy-Proportionality*:
  % peak load = % peak energy

Power Usage Effectiveness

- Overall WSC Energy Efficiency: amount of computational work performed divided by the total energy used in the process
- Power Usage Effectiveness (PUE): Total building power / IT equipment power
  - An power efficiency measure for WSC, *not* including efficiency of servers, networking gear
  - 1.0 = perfection

PUE in the Wild (2007)

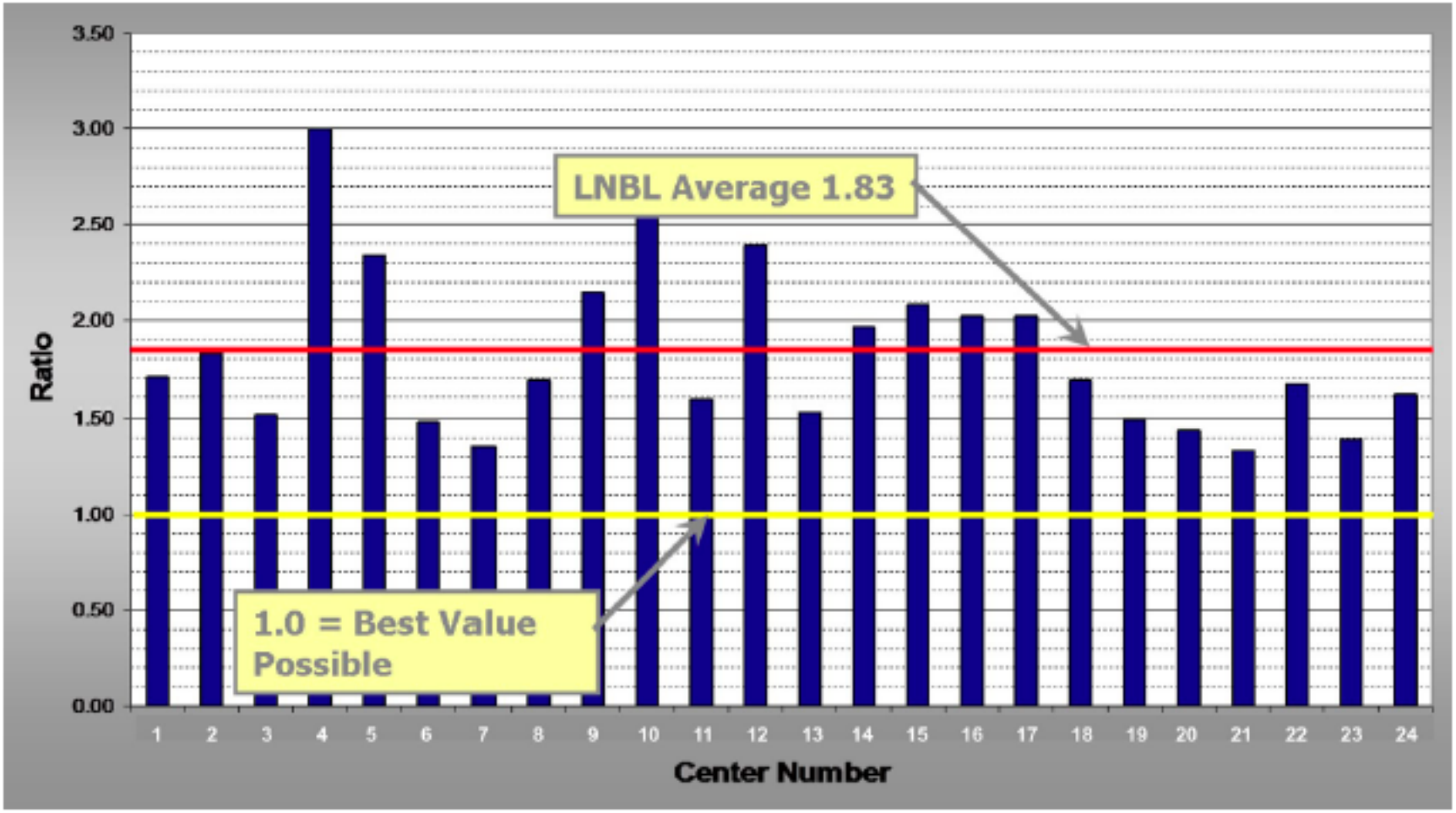


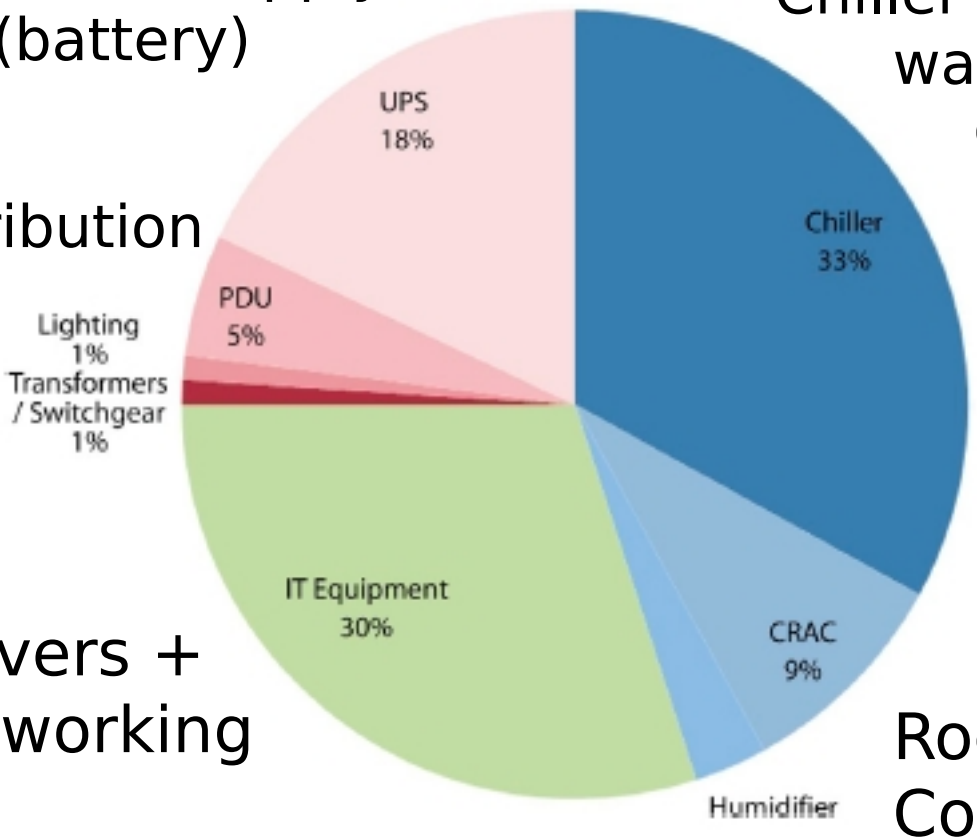
FIGURE 5.1: LBNL survey of the power usage efficiency of 24 datacenters, 2007 (Greenberg et al.)

High PUE: Where Does Power Go?

Uninterruptable Power Supply (battery)

Chiller cools warm water from Air Conditioner

Power Distribution Unit



Servers + Networking

Computer Room Air Conditioner

Google WSC A PUE: 1.24

1. Careful air flow handling
   - Don’t mix hot & cold; containers
2. Elevated cold aisle temperatures
   - Server reliability still OK if not too cool
3. Use of free cooling
   - Location climate, rivers
4. Per-server 12-V DC UPS
5. Measured vs. estimated PUE, publish PUE, and improve operation

Question: Which statement is TRUE about
Warehouse Scale Computers?

(B) Idling electronic equipment consumes almost no power

(G) Reducing lighting costs will *decrease* the Power Usage Effectiveness (PUE)

(P) Qatar (summer temps > 110°F) will one day be a hotbed for WSC housing

(Y) **Using cheaper components is more expensive because of the higher failure rate**

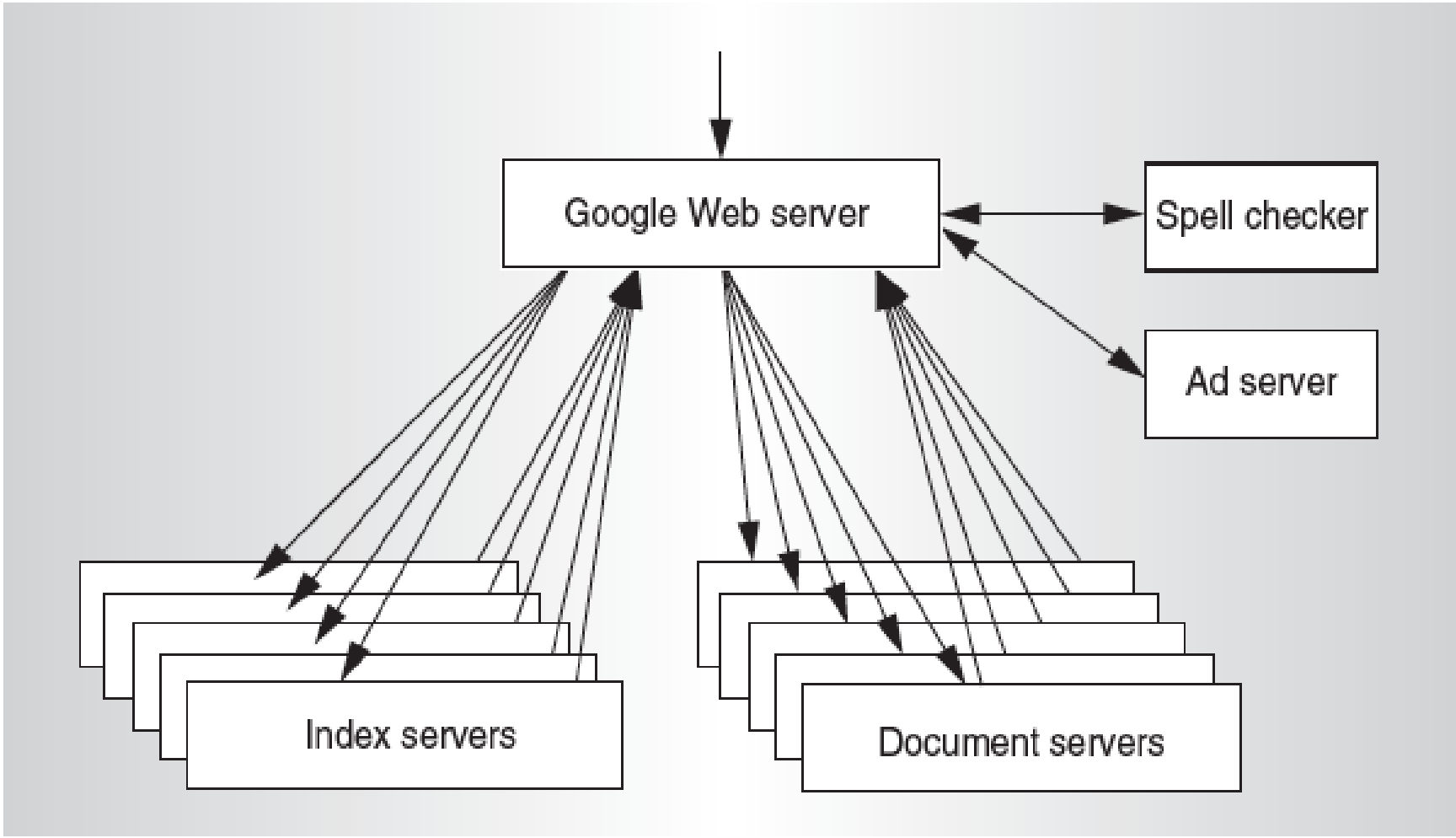
Agenda

- Virtual Memory Wrap-Up
- Administrivia
- Virtual Memory Problems
  - Example Final Question
- Warehouse Scale Computers
- Request Level Parallelism

Request-Level Parallelism (RLP)

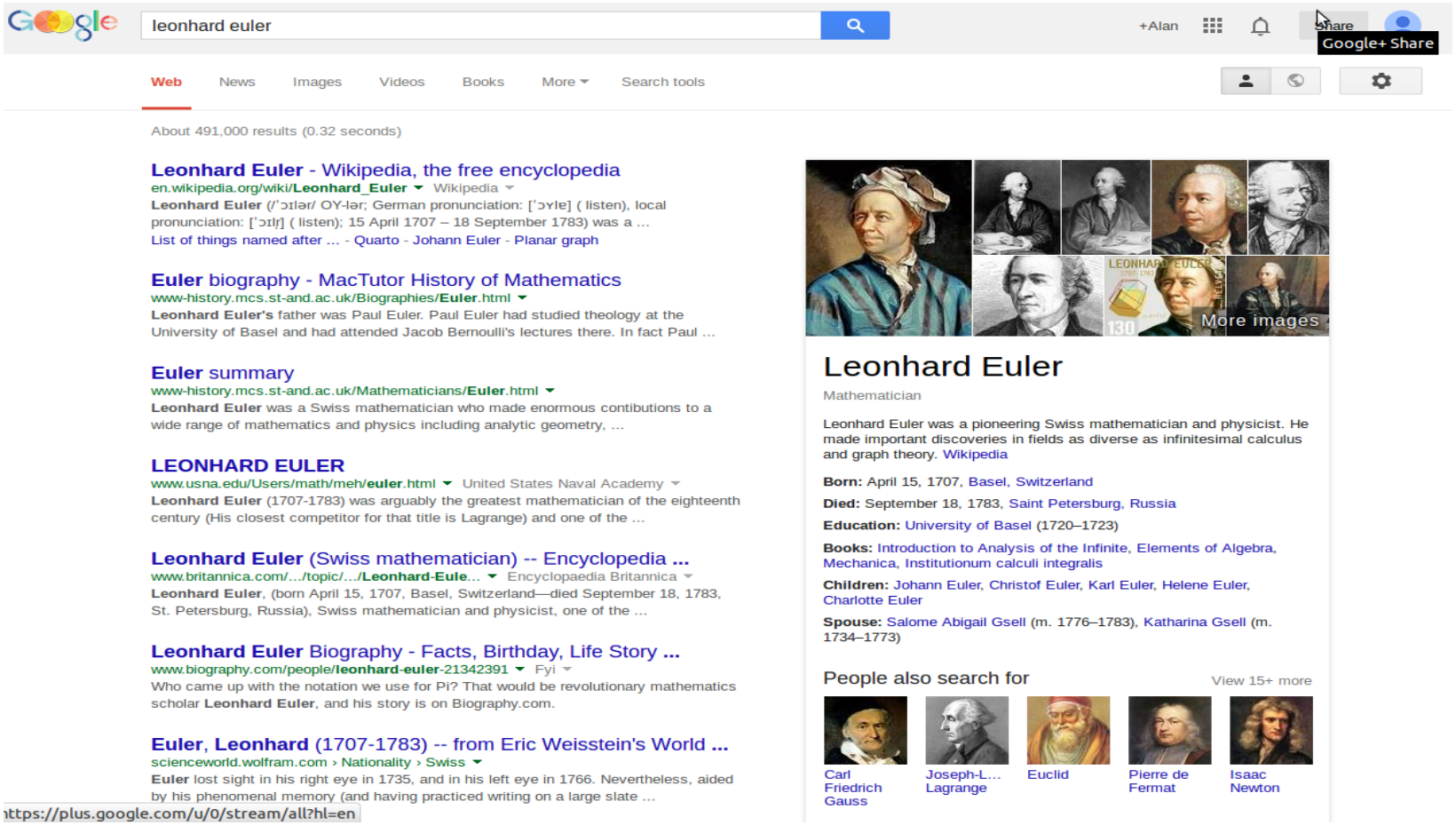
- Hundreds or thousands of requests per sec
  - Not your laptop or cell-phone, but popular Internet services like web search, social networking, …
  - Such requests are largely independent
    - Often involve read-mostly databases
    - Rarely involve strict read–write data sharing or synchronization across requests
- Computation easily partitioned within a request and across different requests

Google Query-Serving Architecture



Anatomy of a Web Search

- Google “Leonhard Euler”



Anatomy of a Web Search (1 of 3)

- Google “Leonhard Euler”
  - Direct request to “closest” Google Warehouse Scale Computer
  - Front-end load balancer directs request to one of many arrays (cluster of servers) within WSC
  - Within array, select one of many Google Web Servers (GWS) to handle the request and compose the response pages
  - GWS communicates with Index Servers to find documents that contain the search words, “Leonhard”, “Euler”, uses location of search as well
  - Return document list with associated relevance score

Anatomy of a Web Search (2 of 3)

- In parallel,
  - Ad system: run ad auction for bidders on search terms
  - Get images of various Leonhard Eulers
- Use docids (document IDs) to access indexed documents
- Compose the page
  - Result document extracts (with keyword in context) ordered by relevance score
  - Sponsored links (along the top) and advertisements (along the sides)

Anatomy of a Web Search (3 of 3)

- Implementation strategy
  - Randomly distribute the entries
  - Make many copies of data (a.k.a. “replicas”)
  - Load balance requests across replicas
- Redundant copies of indices and documents
  - Breaks up hot spots, e.g. “Guardians of the Galaxy”
  - Increases opportunities for request-level parallelism
  - Makes the system more tolerant of failures

Summary

- Warehouse Scale Computers
  - Supports many of the applications we have come to depend on
  - Software must cope with failure, load variation, and latency/bandwidth limitations
  - Hardware sensitive to cost and energy efficiency
- Request Level Parallelism
  - High request volume, each largely independent
  - Replication for better throughput, availability