

Lecture 14: Virtualization and Caching

Principles of Computer Systems
Autumn 2019
Stanford University
Computer Science Department
Lecturer: Chris Gregg
Philip Levis



[PDF of this presentation](#)

Two of Seven

- Abstraction
- Modularity and Layering
- Naming and Name Resolution
- Caching
- Virtualization
- Concurrency
- Client-server request-and-response

Two of Seven

-
-
-
- Caching
- Virtualization
-
-

Two of Seven

-
-
-
- Caching
- **Virtualization**
-
-

Virtualization

- Through a layer of indirection, make one look like many or many look like one
 - Virtualizing the CPU (e.g., processes): one like many
 - Virtual machines: one like many
 - Virtual memory: one like many
 - RAID (Redundant Array of Inexpensive Disks): many like one
 - Logical volumes: one like many
 - Virtual private networks: one like many
- Decouples program from physical resources

Virtualization

- Through a layer of indirection, make one look like many or many look like one
 - Virtualizing the CPU (e.g., processes): one like many
 - Virtual machines: one like many
 - Virtual memory: one like many
 - **RAID (Redundant Array of Inexpensive Disks): many like one**
 - Logical volumes: one like many
 - Virtual private networks: one like many
- Decouples program from physical resources

RAID (Redundant Array of Inexpensive Disks)

- Disks have limited space: biggest disk today is ~15TB
- What if you need more than 15TB?
 - Could make bigger and bigger disks -- but cost is non-linear
- Use *virtualization*: put multiple physical disks together to look like one bigger virtual disk

EDITION: **US** ▼


ZDNet 🔍

VIDEOS 5G WINDOWS 10 CLOUD AI INNOVATION SECURITY MORE ▼ NEWSLETTERS ALL WRITERS

📄 **MUST READ:** Europe's cloud computing plan won't do much to scare the US giants

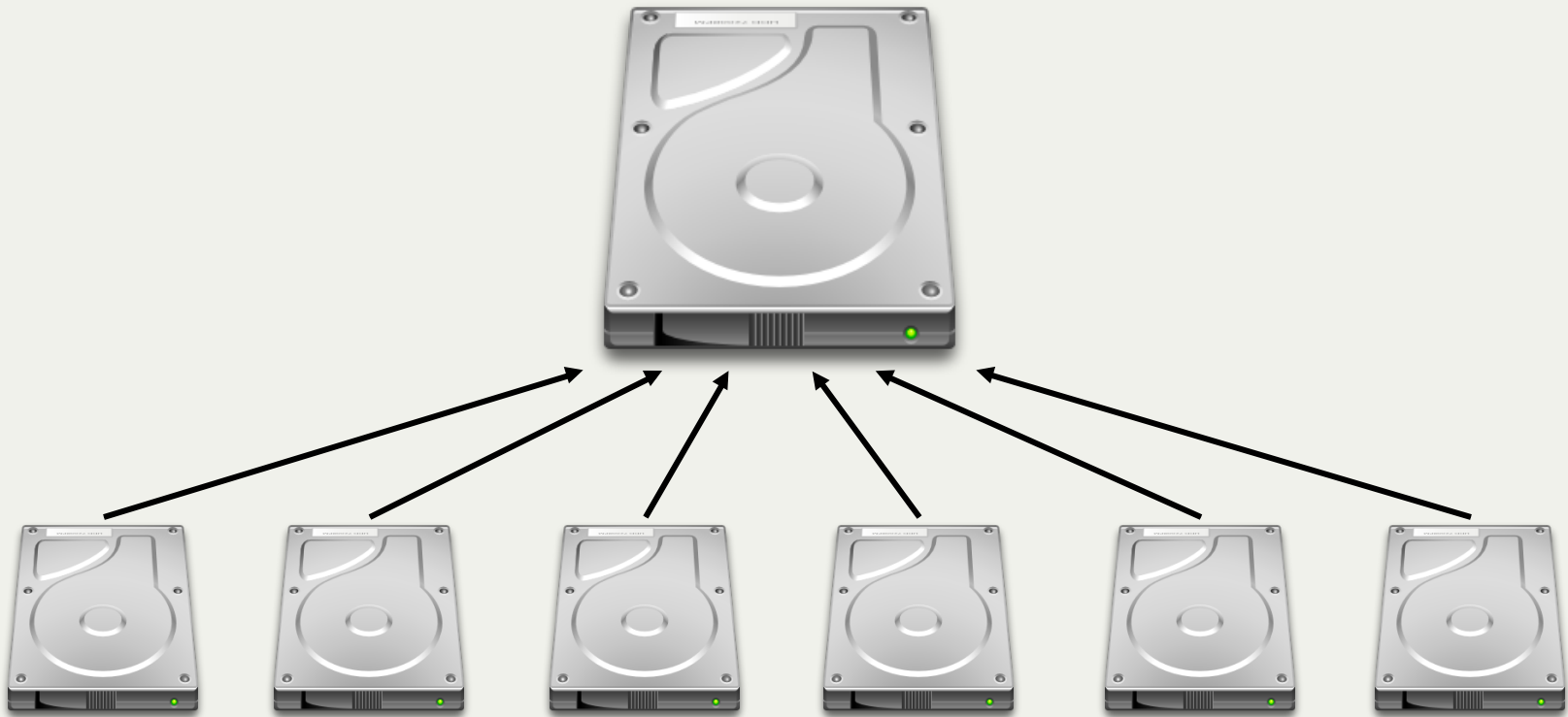
World's biggest hard drive: Meet Western Digital's 15TB monster

Western Digital packs another terabyte into its 3.5-inch hard disk drives.

 By [Liam Tung](#) | October 26, 2018 -- 12:49 GMT (05:49 PDT) | Topic: [Storage](#)

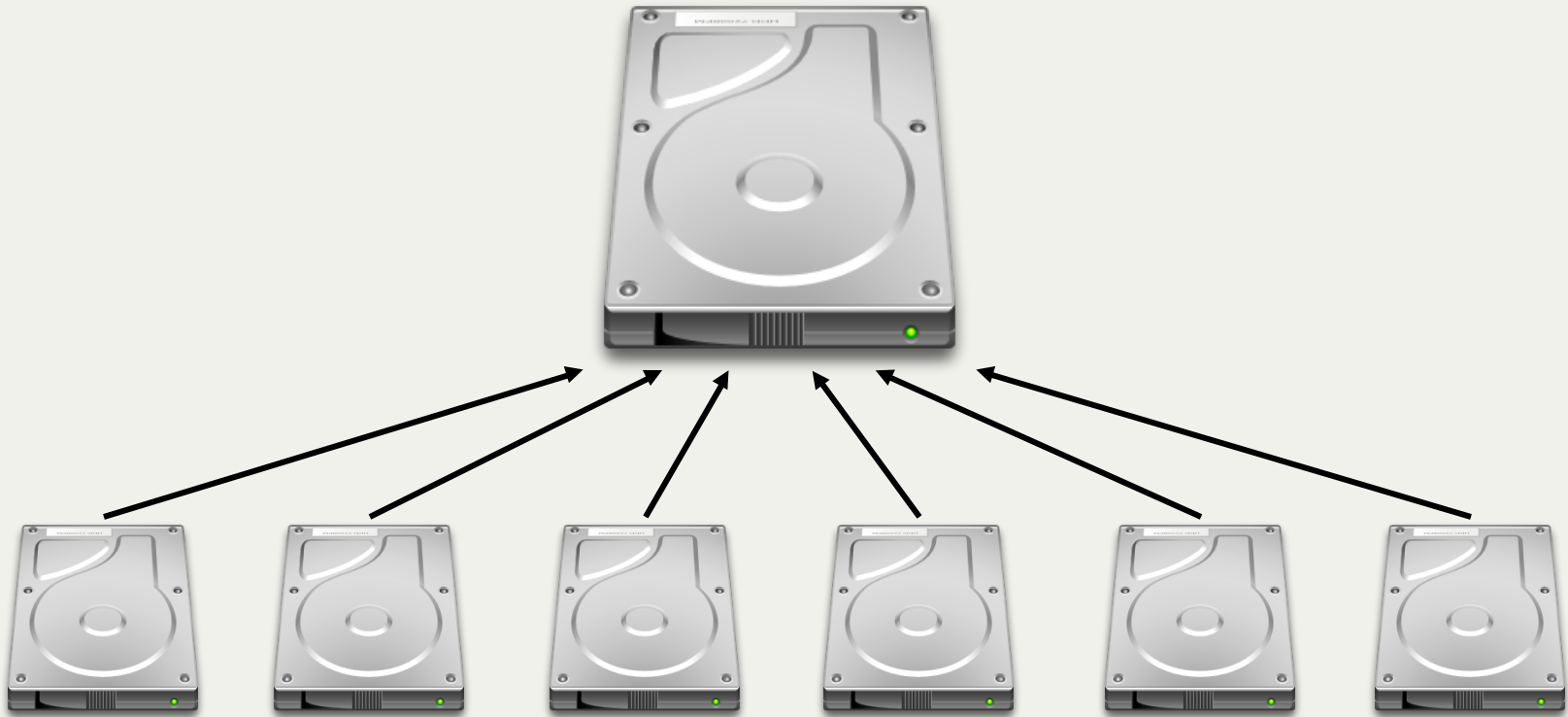
RAID (Redundant Array of Inexpensive Disks)

- Disks have limited space: biggest disk today is ~15TB
- What if you need more than 15TB?
 - Could make bigger and bigger disks -- but cost is non-linear
- Use *virtualization*: put multiple physical disks together to look like one bigger virtual disk



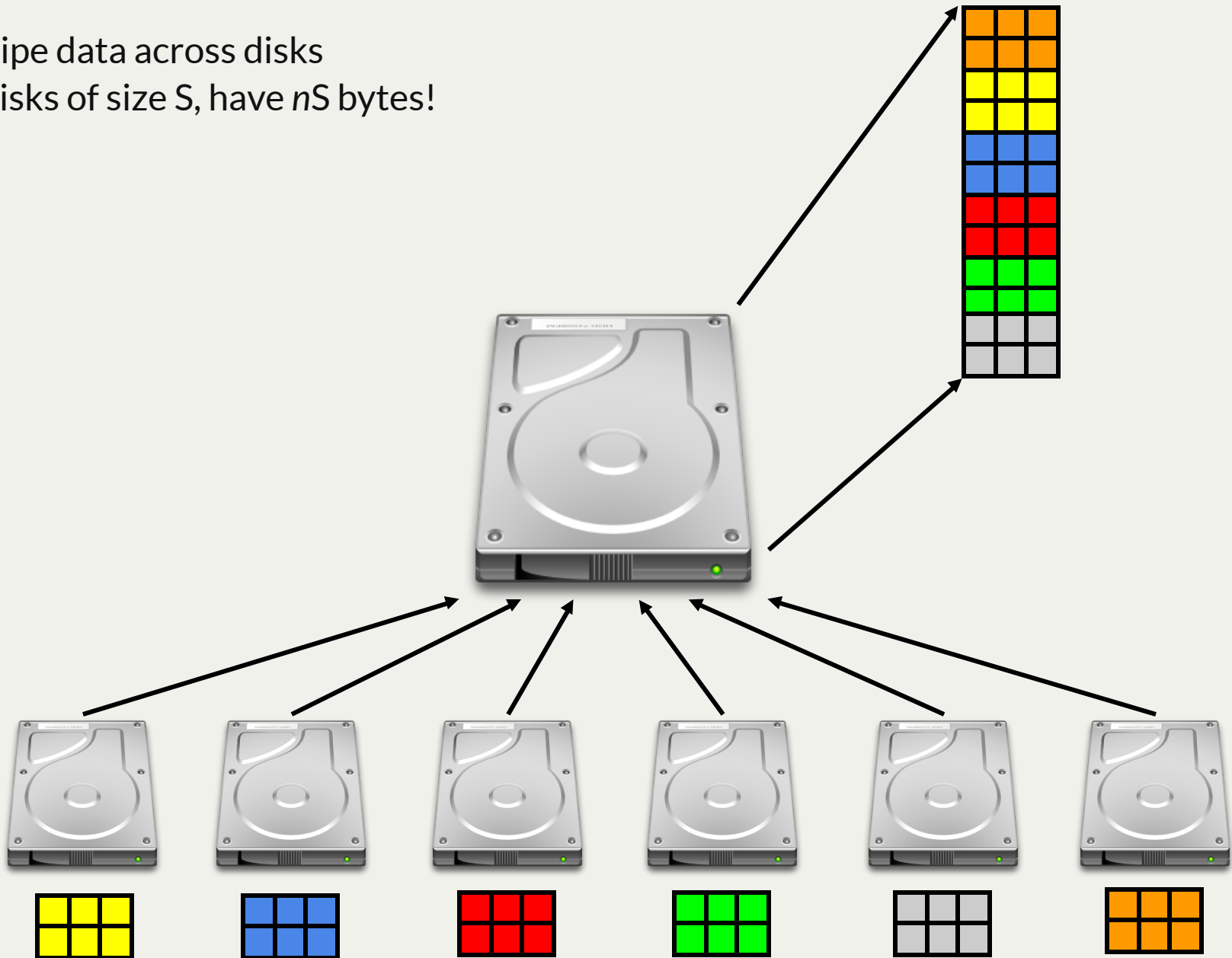
RAID: a lot of advantages

- Size: we can make arbitrarily large disks
- Speed: if we lay out data well, we can read from N disks in parallel, not just one
- Cost: N inexpensive disks is cheaper than one huge disk



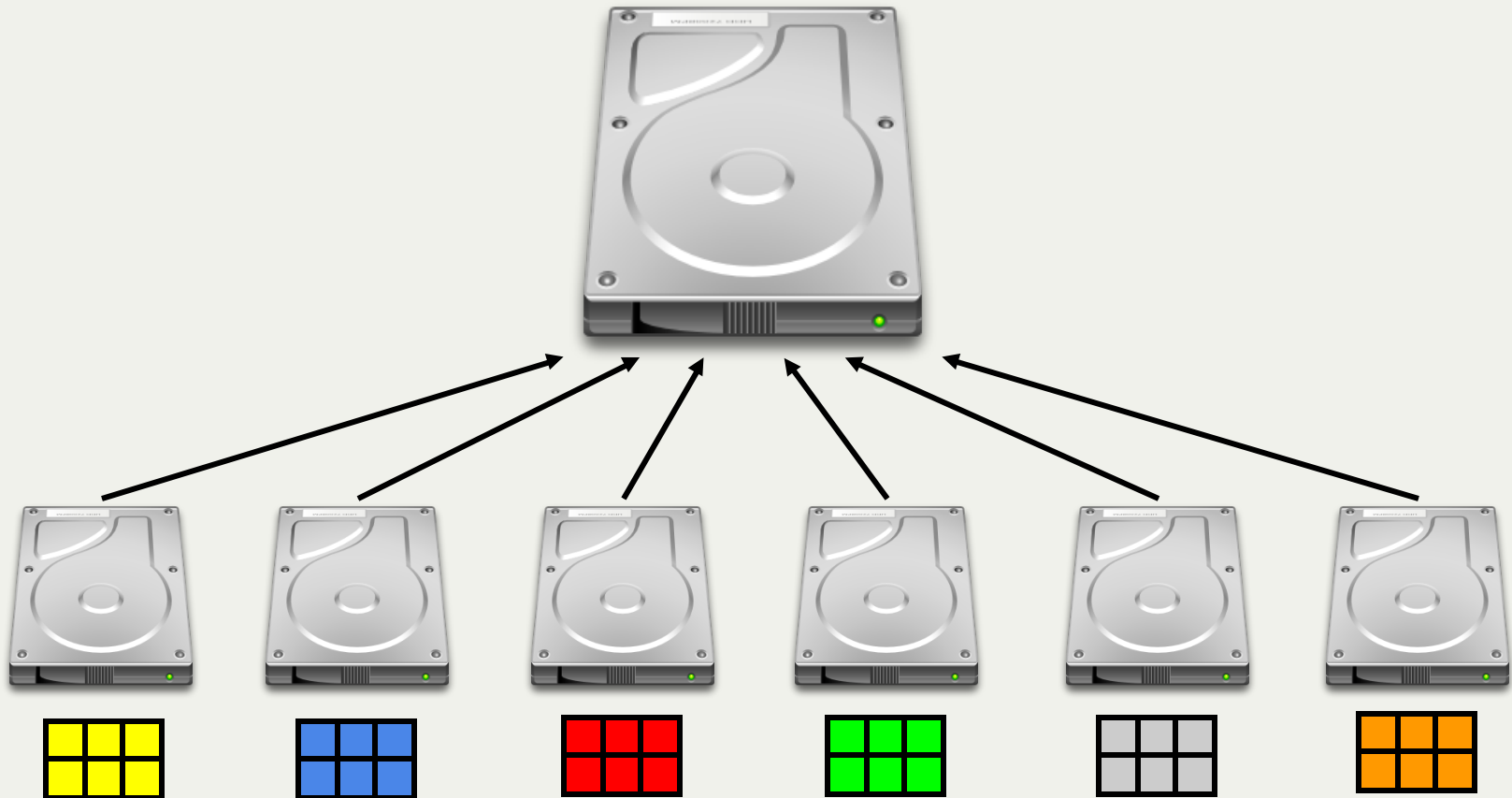
RAID 0

- Stripe data across disks
- n disks of size S , have nS bytes!



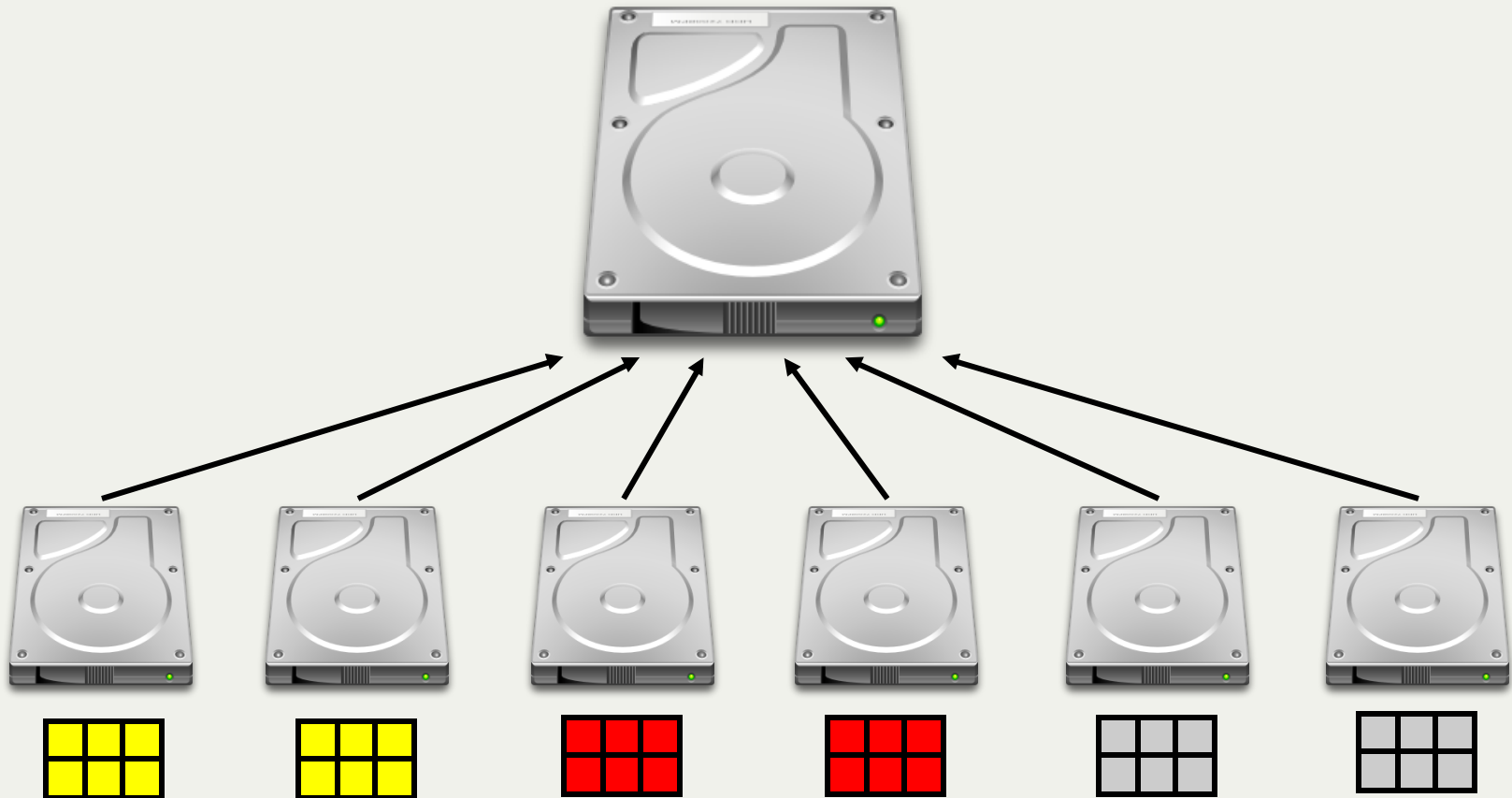
RAID 0 Problems

- If *one* disk fails, the entire RAID array fails
- Suppose each disk has a probability p of failing per month
- Probability each disk does not fail is $(1-p)$
- Probability all n disks do not fail is $(1-p)^n$
 - Suppose $p = 0.001$; if $n=20$, there's a 2% chance the RAID array will fail each month



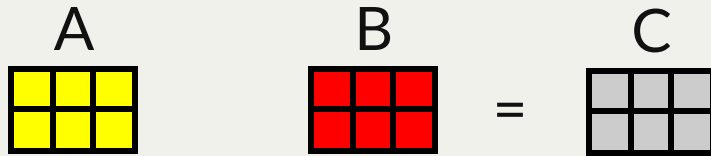
Redundant Array of Inexpensive Disks: RAID 1

- Key idea: arrange the data on the disks so the array can survive failures
- Simplest approach is mirroring, RAID 1
- Halves capacity, but still less expensive than a big disk
- Probability 2 replicas fail is $1-(1-p^2)^{n/2}$
 - If $p = 0.001$, if $n=20$, there's a .00001% chance the RAID array will fail each month



The Power of XOR

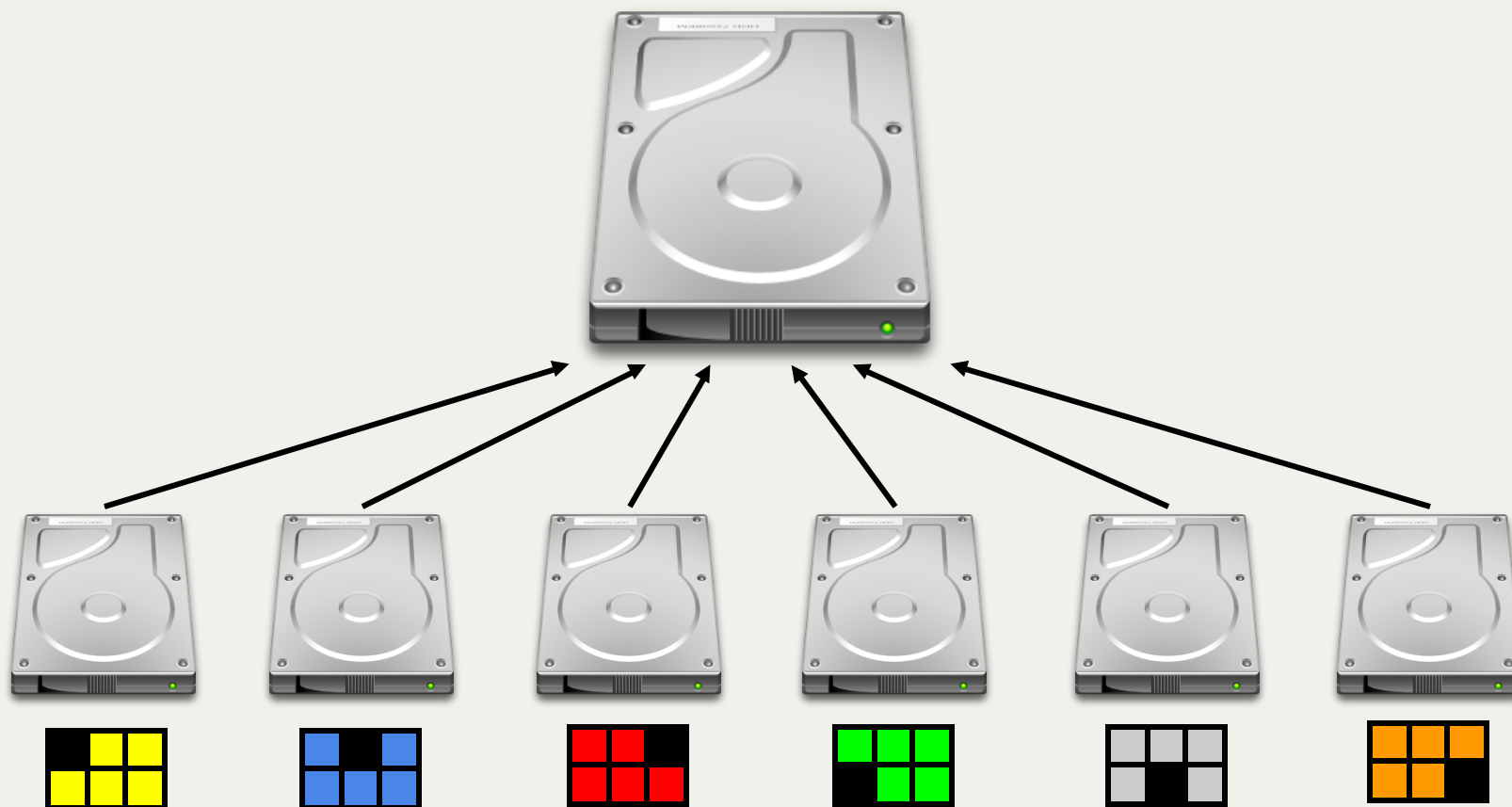
- There are better ways to have recovery data than simple replication
- Exclusive OR (XOR)
- Suppose we have two drives, A and B
- One extra drive C: $C = A \oplus B$
- If B fails, then you can recover B: $B = A \oplus C$



A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

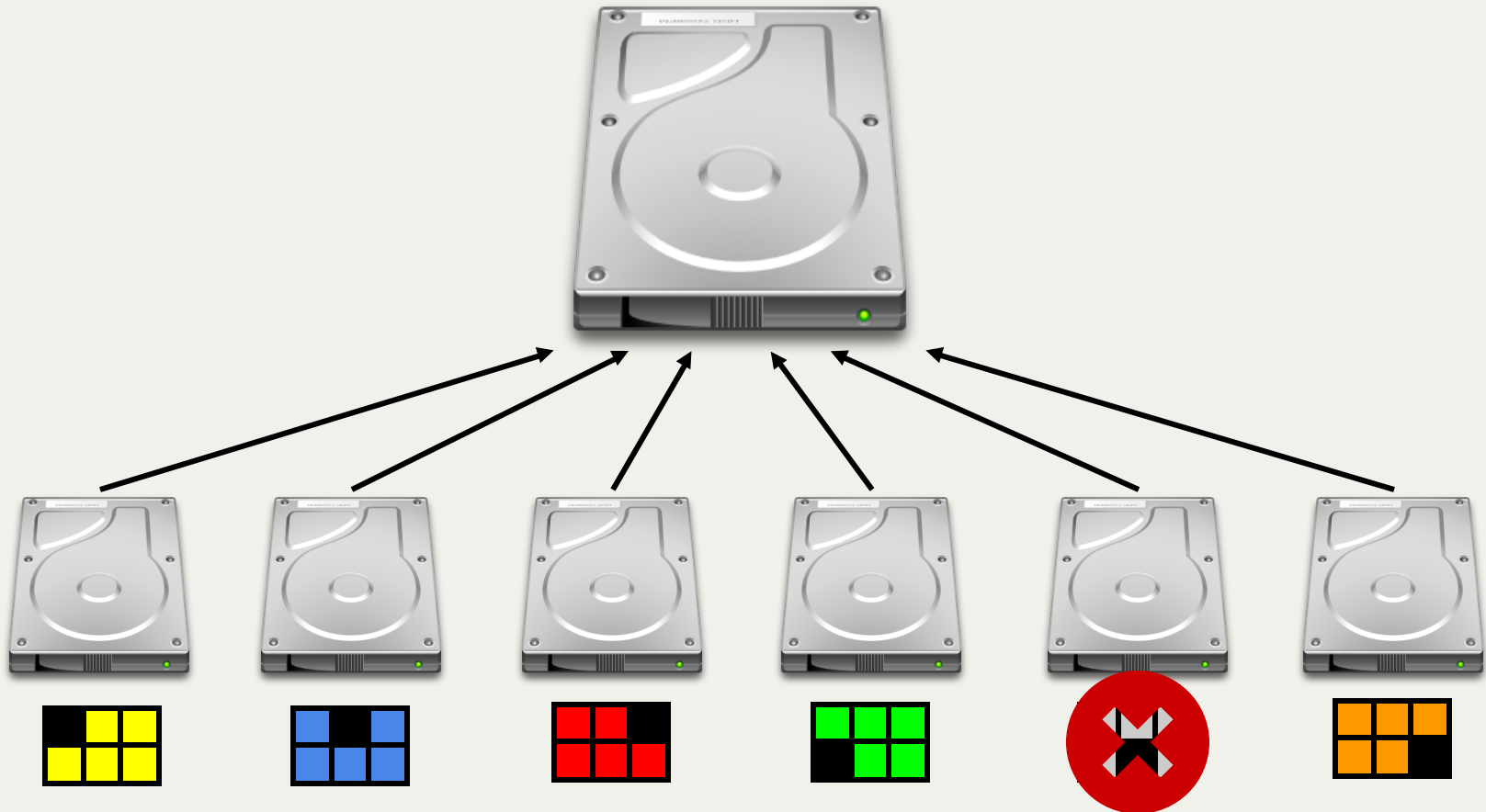
RAID 5: Resiliency With Less Cost

- RAID 5 stripes the data across disks, sets aside 1 disk worth of storage as *parity*
- Parity is the XOR of all of that sector on all of the other drives
 - Writes write two drives: data and parity; parity is spread: lose $1/n$ of storage
- Requires two drives to fail: $n=6$, $p=0.001$, failure ≈ 0.000015
 - If one drive fails, it can be recovered from the parity bits (just XOR other disks)



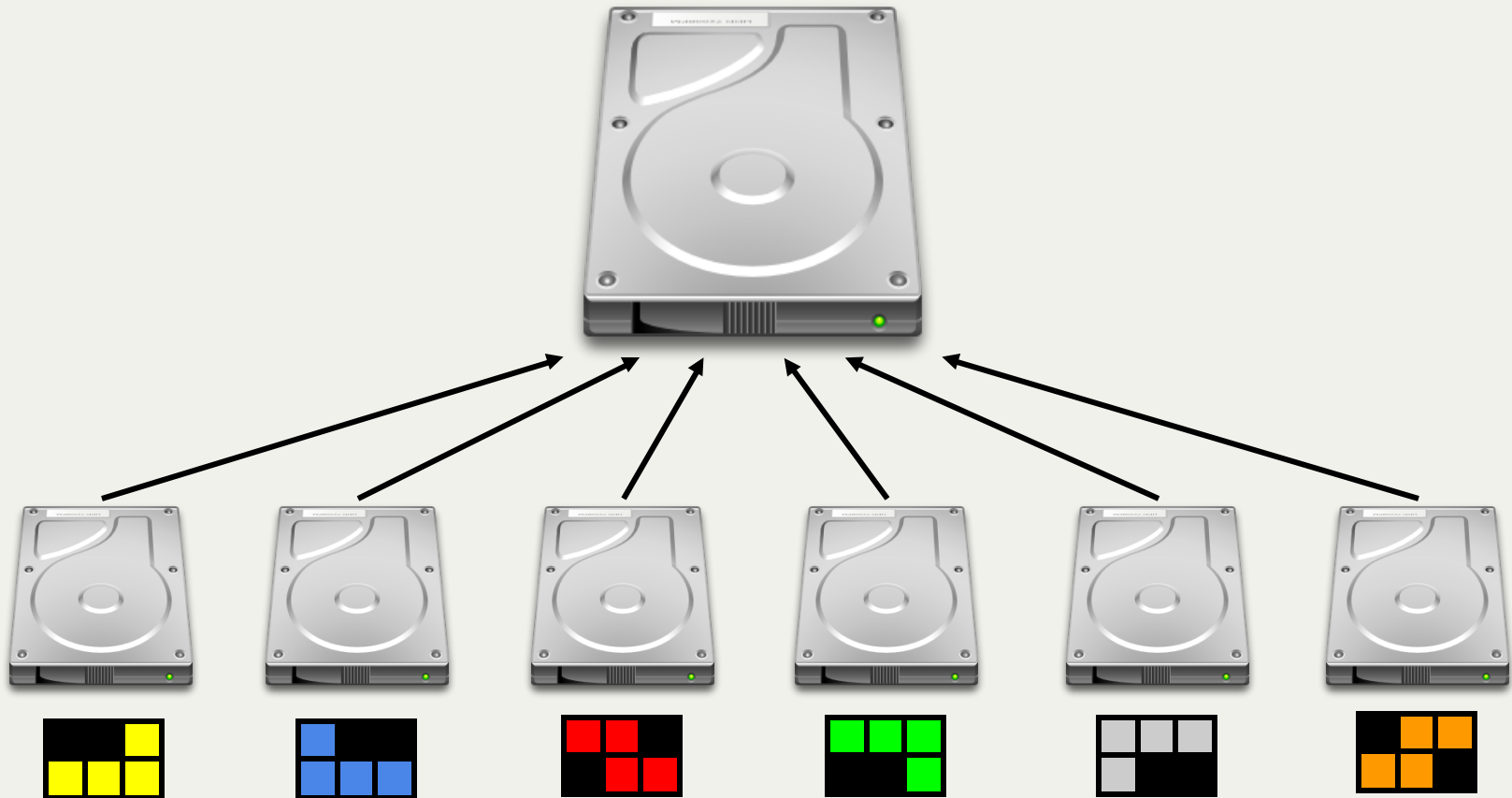
RAID 5: Resiliency With Less Cost

- Suppose we have 6 disks total, one parity disk
- We lose disk 4
- Question 1: Can we still service reads? If so, how does one read from disk 4?
- Question 2: Can we still service writes? If so, how does one write to disk 4?
- Question 3: How do we recover disk 4?



Reed-Solomon Coding

- What if chances more than can fail becomes dangerous (thousands of drives)?
- Reed-Solomon coding: turn k data blocks into n , can recover from any $(n-k)$ failures
 - E.g., turn 223 data blocks into 255, can recover from any 32 failures
 - Used in CDs, DVDs, QR codes, Mars Rovers, and most cloud storage systems
- RAID 6: use Reed-Solomon to have two parity drives



RAID invented in 1988 (4 years after first Macintosh)

A Case for Redundant Arrays of Inexpensive Disks (RAID)

David A. Patterson, Garth Gibson, and Randy H. Katz

Computer Science Division
Department of Electrical Engineering and Computer Sciences
571 Evans Hall
University of California
Berkeley, CA 94720
(patt@cs.berkeley.edu)

Abstract Increasing performance of CPUs and memories will be squandered if not matched by a similar performance increase in I/O. While the capacity of Single Large Expensive Disks (SLED) has grown rapidly, the performance improvement of SLED has been modest. Redundant Arrays of Inexpensive Disks (RAID), based on the magnetic disk technology developed for personal computers, offers an attractive alternative to SLED, promising improvements of an order of magnitude in performance, reliability, power consumption, and scalability. This paper introduces five levels of RAID, giving their relative cost/performance, and compares RAID to an IBM 3380 and a Fujitsu Super Eagle.

One of the major trends in the development of magnetic disk technology is the growth in the maximum number of bits that can be stored per square inch, or the bits per inch in a track times the number of tracks per inch. Called MAD, for maximal areal density, the "First Law in Disk Density" predicts [Frank87]

$$MAD = 10^{(Year-1971)/10}$$

Magnetic disk technology has doubled capacity and halved price every three years, in line with the growth rate of semiconductor memory, and in practice between 1967 and 1979 the disk capacity of the average IBM data processing system more than kept up with its main memory [Stevens81].

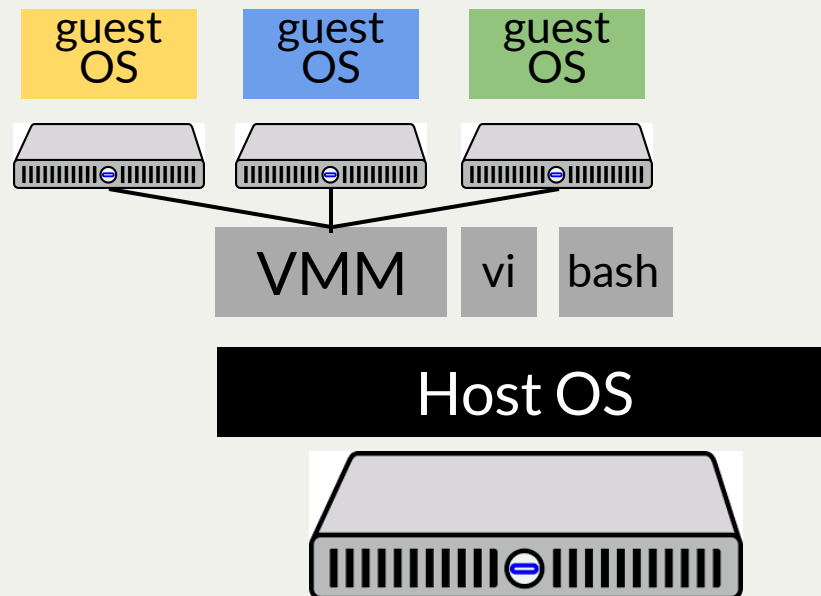
Described up to RAID 5 (also, RAID 2, RAID 3, RAID 4)

Virtualization

- Through a layer of indirection, make one look like many or many look like one
 - Virtualizing the CPU (e.g., processes): one like many
 - **Virtual machines: one like many**
 - Virtual memory: one like many
 - RAID (Redundant Array of Inexpensive Disks): many like one
 - Logical volumes: one like many
 - Virtual private networks: one like many
- Decouples program from physical resources

Virtual Machine

- Software that makes code (running in a process) think that it's running on raw hardware
- A *virtual machine monitor* runs in the *host operating system*
 - It loads and run disk images for *guest operating systems*
 - Operations in the guest operating system that are normally not allowed trap into the virtual machine monitor
 - Guest operating system tries to change page tables
 - Guest operating system tries to disable interrupts
 - Virtual machine monitor emulates the hardware

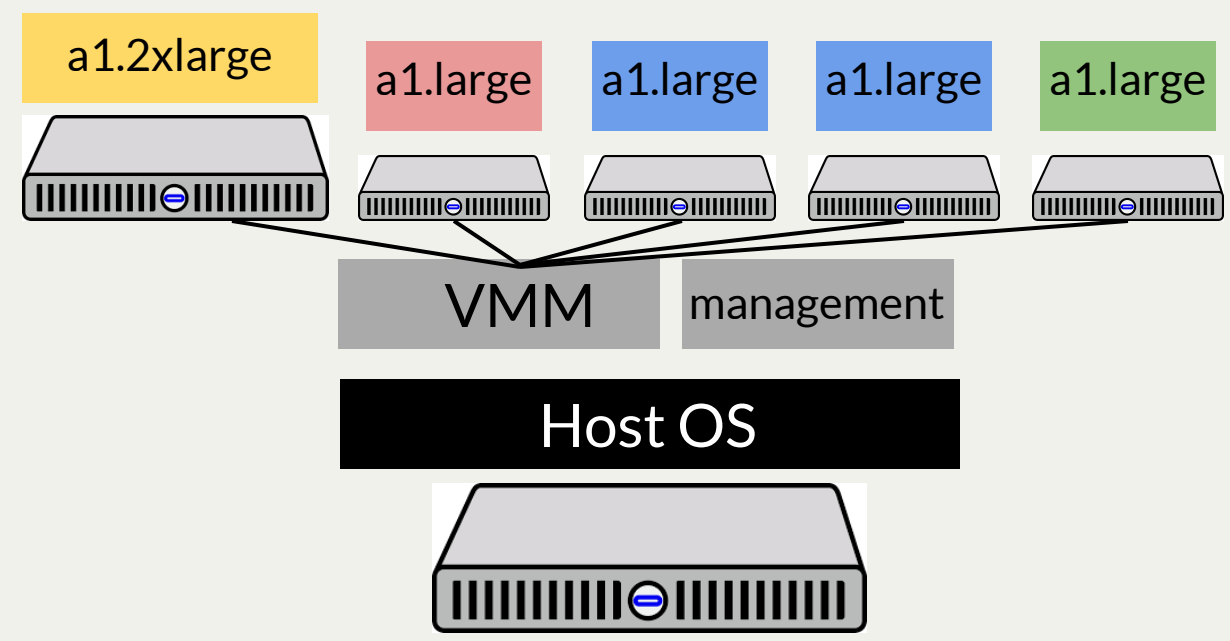


Amazon Elastic Compute Cloud (EC2)

- Amazon computing service: a virtual computer is called an *instance*
- Many different kinds of instance: general purpose, memory-optimized, compute-optimized, GPUs, etc.
- There's generally a full instance size, and you can have $1/2^n$ of it
 - Four a1.large is the same as one a1.2xlarge
 - Two a1.2x large is the same as one a1.4xlarge

	vCPU	ECU	Memory (GiB)	Instance Storage (GB)	Linux/UNIX Usage
General Purpose - Current Generation					
a1.medium	1	N/A	2 GiB	EBS Only	\$0.0255 per Hour
a1.large	2	N/A	4 GiB	EBS Only	\$0.051 per Hour
a1.xlarge	4	N/A	8 GiB	EBS Only	\$0.102 per Hour
a1.2xlarge	8	N/A	16 GiB	EBS Only	\$0.204 per Hour
a1.4xlarge	16	N/A	32 GiB	EBS Only	\$0.408 per Hour
a1.metal	16	N/A	32 GiB	EBS Only	\$0.408 per Hour

Amazon EC2 Example



	vCPU	ECU	Memory (GiB)	Instance Storage (GB)	Linux/UNIX Usage
General Purpose - Current Generation					
a1.medium	1	N/A	2 GiB	EBS Only	\$0.0255 per Hour
a1.large	2	N/A	4 GiB	EBS Only	\$0.051 per Hour
a1.xlarge	4	N/A	8 GiB	EBS Only	\$0.102 per Hour
a1.2xlarge	8	N/A	16 GiB	EBS Only	\$0.204 per Hour
a1.4xlarge	16	N/A	32 GiB	EBS Only	\$0.408 per Hour
a1.metal	16	N/A	32 GiB	EBS Only	\$0.408 per Hour

Virtual Machine Advantages

- Move whole images anywhere: completely decouple all software from hardware
- Can replicate computer images: run more copies
 - If your service is overloaded, scale out by spinning up more instances
- Can arbitrarily start/stop/resume instances very quickly
 - Must faster than shutting down machines
- Complete software encapsulation
 - Common technique used in software tutorials: download this VM image and run it
 - Web hosting: one server can run 100 virtual machines, each one thinks it has a complete, independent computer to configure and use
- Complete software isolation
 - In theory, two VMs are completely isolated, can maybe only sense something due to timing (e.g., if they are sharing a CPU), more on this later
- Enabled us to have cloud computing
 - Original business case was the desktop! E.g., need to run Windows and Linux in parallel, don't want 2 machines.

Modern Virtual Machines Invented in 1997

Using the SimOS Machine Simulator to Study Complex Computer Systems

MENDEL ROSENBLUM, EDOUARD BUGNION, SCOTT DEVINE, and
STEPHEN A. HERROD

Computer Systems Laboratory, Stanford University

SimOS is an environment for studying the hardware and software of computer systems. SimOS simulates the hardware of a computer system in enough detail to boot a commercial operating system and run realistic workloads on top of it. This paper identifies two challenges that machine simulators such as SimOS must overcome in order to effectively analyze large complex workloads: handling long workload execution times and collecting data effectively. To study long-running workloads, SimOS includes multiple interchangeable simulation models for each hardware component. By selecting the appropriate combination of simulation models, the user can explicitly control the tradeoff between simulation speed and simulation detail. To handle the large amount of low-level data generated by the hardware simulation models, SimOS contains flexible annotation and event classification mechanisms that map the data back to concepts meaningful to the user. SimOS has been extensively used to study new computer hardware designs, to analyze application performance, and to study operating systems. We include two case studies that demonstrate how a low-level machine simulator such as SimOS can be used to study large and complex workloads.

Two of Seven

-
-
-
- **Caching**
- Virtualization
-
-

Latency Numbers Every Programmer Should Know

(Peter Norvig and Jeff Dean)

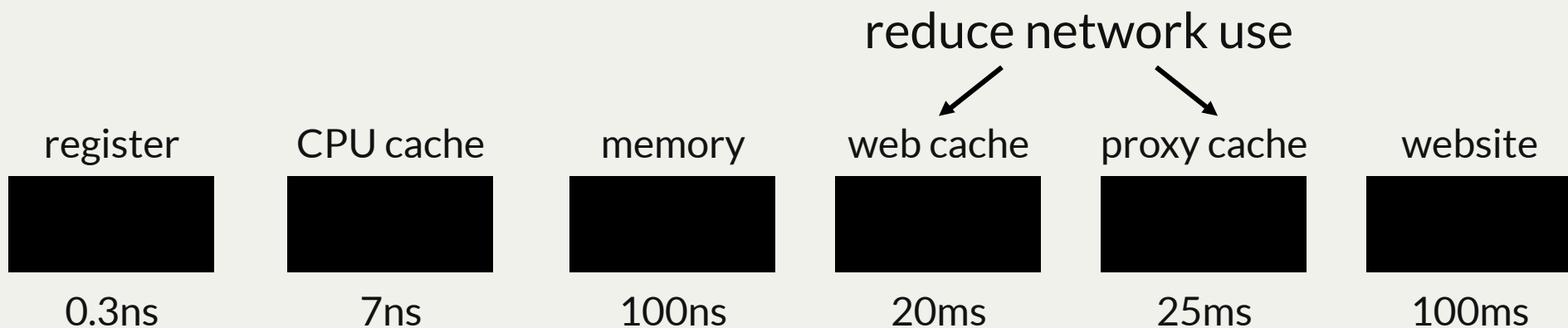
L1 cache reference	0.5ns		
Branch mispredict	5ns		
L2 cache reference	7ns		
Mutex lock/unlock	25ns		
Main memory reference	100ns		
Compress 1K with Zippy	3,000ns	3us	
Send 1K over 1Gbps network	10,000ns	10us	
Read 4K randomly from SSD	150,000ns	150us	
Read 1MB sequentially from RAM	250,000ns	250us	
Round trip within a datacenter	500,000ns	500us	
Read 1MB sequentially from SSD	1,000,000ns	1,000us	1ms
Hard disk seek	10,000,000ns	10,000us	10ms
Read 1MB sequentially from disk	20,000,000ns	20,000us	20ms
Send packet CA->Netherlands->CA	150,000,000ns	150,000us	150ms

Caching

- Performance optimization
- Keeping a copy of some data
 - Usually, closer to where the data is needed
 - Or, something that might be reused (don't recompute)
- Used *everywhere* in computer systems
 - Registers
 - Processor caches
 - File system buffer cache
 - DNS caching
 - memcached
 - Database page cache
 - Spark analytics framework
 - Web browser page/image cache
 - Phone email/SMS cache

Why Is Caching Useful

- There is a basic tradeoff in performance and size
- If you make it bigger, it's slower
 - Takes longer to get to (due to size)
 - Addressing it is more complex (more bits to switch on)
- Faster storage is more expensive
 - 16GB RAM: \$59.99
 - 1TB HDD: \$59.99
 - 4TB HDD: \$116.99
 - 4TB SSD: \$499.99
- Think about the places your web page might be stored...

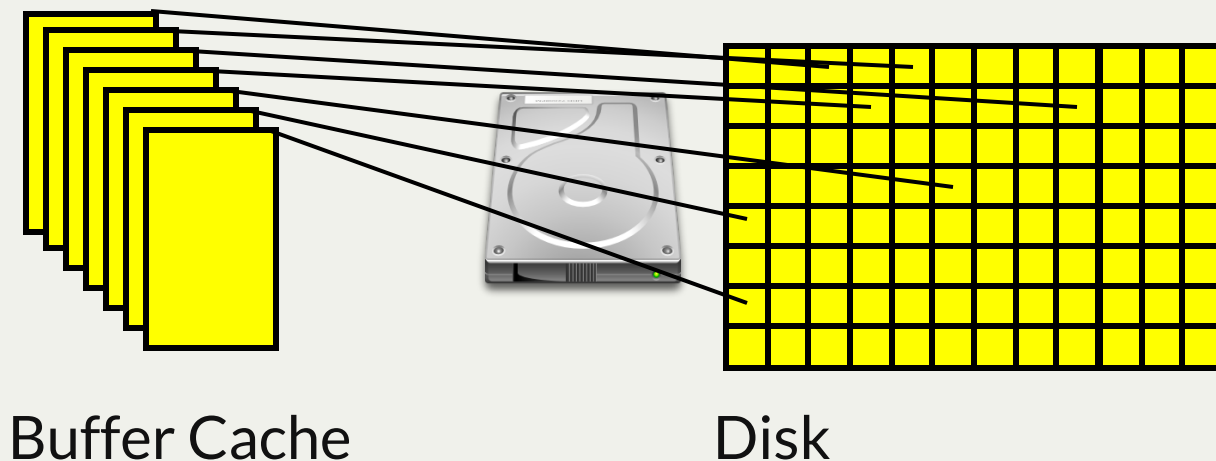


Caching

- Performance optimization
- Keeping a copy of some data
 - Usually, closer to where the data is needed
 - Or, something that might be reused (don't recompute)
- Used *everywhere* in computer systems
 - Registers
 - Processor caches
 - **File system buffer cache**
 - DNS caching
 - memcached
 - Database page cache
 - Spark analytics framework
 - Web browser page/image cache
 - Phone email/SMS cache

File System Buffer Cache

- The operating system maintains a *buffer cache* of disk blocks that have been brought into memory
- When you read or write a file, you read or write to a buffer cache entry
 - If that block was not in RAM, the OS brings it into RAM, then does the operation
- A write marks a buffer cache entry as *dirty*
- Dirty entries are asynchronously written back to disk
 - Can be forced with `fsync(2)`
- Buffer cache absorbs both reads and writes, prevents them from hitting disk (100,000x performance difference)



File System Buffer Cache Integration with mmap(2)

- Recall that a process memory space is divided into segments
- Some segments are mmaped files (e.g., your program, libraries)
- The buffer cache is what sits behind this
- If memory is low, start deleting buffer cache entries
 - If the entry is clean, just reclaim the memory
 - If it's dirty, write it back to disk
- Others are anonymous -- zeroed out memory for heap, stack, etc.
 - Backed by *swap*, a region of disk for storing program state when memory is scarce
- Why does sometimes a process take a while to respond after being idle?

Address	Kbytes	Mode	Offset	Device	Mapping
000055bde4835000	8	r-x--	0000000000000000	008:00008	gedit
000055bde4a36000	4	r----	00000000000001000	008:00008	gedit
000055bde4a37000	4	rw---	00000000000002000	008:00008	gedit
000055bde5d32000	13944	rw---	0000000000000000	000:00000	[anon]
00007fc910000000	132	rw---	0000000000000000	000:00000	[anon]
00007fc910021000	65404	-----	0000000000000000	000:00000	[anon]
00007fc918000000	896	rw---	0000000000000000	000:00000	[anon]
00007fc9180e0000	64640	-----	0000000000000000	000:00000	[anon]
00007fc91c750000	204	r----	0000000000000000	008:00008	UbuntuMono-R.ttf
00007fc91c783000	644	r-x--	0000000000000000	008:00008	libaspell.so.15.2.0
00007fc91c824000	2048	-----	00000000000a1000	008:00008	libaspell.so.15.2.0
00007fc91ca24000	20	r----	00000000000a1000	008:00008	libaspell.so.15.2.0

Caching

- Performance optimization
- Keeping a copy of some data
 - Usually, closer to where the data is needed
 - Or, something that might be reused (don't recompute)
- Used *everywhere* in computer systems
 - Registers
 - Processor caches
 - File system buffer cache
 - **DNS caching**
 - memcached
 - Database page cache
 - Spark analytics framework
 - Web browser page/image cache
 - Phone email/SMS cache

Domain Name System (DNS)

- Every computer on the Internet has an IP address
 - This is a 32-bit number, written as 4 8-bit values
 - stanford.edu: 171.67.215.200
 - Often, many computers share a single address, but let's not worry about that for now
- Network communication is in terms of these addresses
 - You can't send a web request to `www.stanford.edu`; you **can** send request its IP address
- The addresses have some structure
 - Stanford controls the block of 65,536 addresses starting with 171.67
 - Stanford has 5 such blocks (called a /16 because the first 16 bits are specified)
- The Domain Name System (DNS) maps names like `www.stanford.edu` to IP addresses
 - It's a network service run on some servers
 - Uses a special message format, called a query: you ask a DNS *resolver* to answer the query, it goes out and asks other servers around the Internet and returns the result to you
- Every answer has a *time-to-live* field: how long is this answer valid?
 - Resolvers cache the answer for at most that long: if another query comes in, answer from the cache rather than going out over the network
 - Some heavily-used, shared machines (e.g., myth) run their own resolver cache as well
- When you get an IP address by associated with a network, you're given an IP address to use to query DNS with

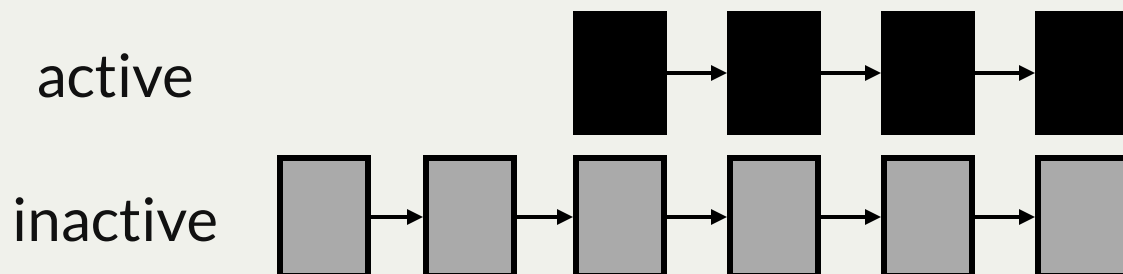
Domain Name System (DNS) Example With dig

Domain Name System (DNS) Naming

- Example of naming and name resolution
- Turn a structured, human readable name into an IP address
- Look it up in reverse order: `www.stanford.edu`
 - Ask root servers: "whom can I ask about `.edu`?"
 - Ask `.edu` servers: "whom can I ask about `stanford.edu`?"
 - Ask `stanford.edu` server: "What's the IP address of `www.stanford.edu`?"

Questions with Caching

- What do you do when the cache is full?
- Cache *eviction policy*
- Buffer cache
 - Optimal policy knows what will be accessed in the future, doesn't evict those
 - Let's approximate: least recently used (LRU)
 - Keeping track of exact LRU is expensive, let's approximate
 - Keep two FIFO queues, active and inactive
 - If a page in the inactive queue is accessed, put it into the active queue
 - If you need to evict, evict from head (oldest) of inactive queue
 - If active queue is too long, move some pages to inactive queue



- DNS
 - Keep record until its TTL expires

Systems Principles

- Abstraction
 - Modularity and Layering
 - Naming and Name Resolution
 - Caching
 - Virtualization
 - Concurrency
 - Client-server request-and-response
-
- These principles come up again and again in computer systems
 - As we start to dig into networking, we're going to see
 - abstraction
 - layering
 - naming and name resolution (DNS!)
 - caching
 - concurrency
 - client-server