Last lecture!

REVIEW, SUMMARY

Shivaram Venkataraman

CS 537, Spring 2020

ADMINISTRIVIA

Project 5 grades on the way. → Regrade requests by early next week.

Final Exam: → Monday !!

Everything up to including NFS. ← up to & including the previous lecture

May 4th, 10:05am-12:05pm

↳ Piazza

No discussion today!

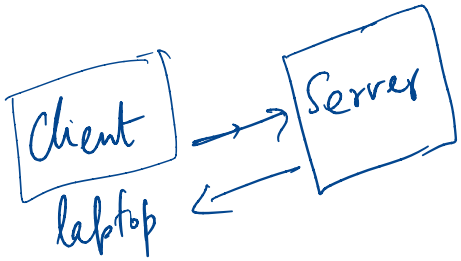
↳ Study / Practice for final

AGENDA / LEARNING OUTCOMES

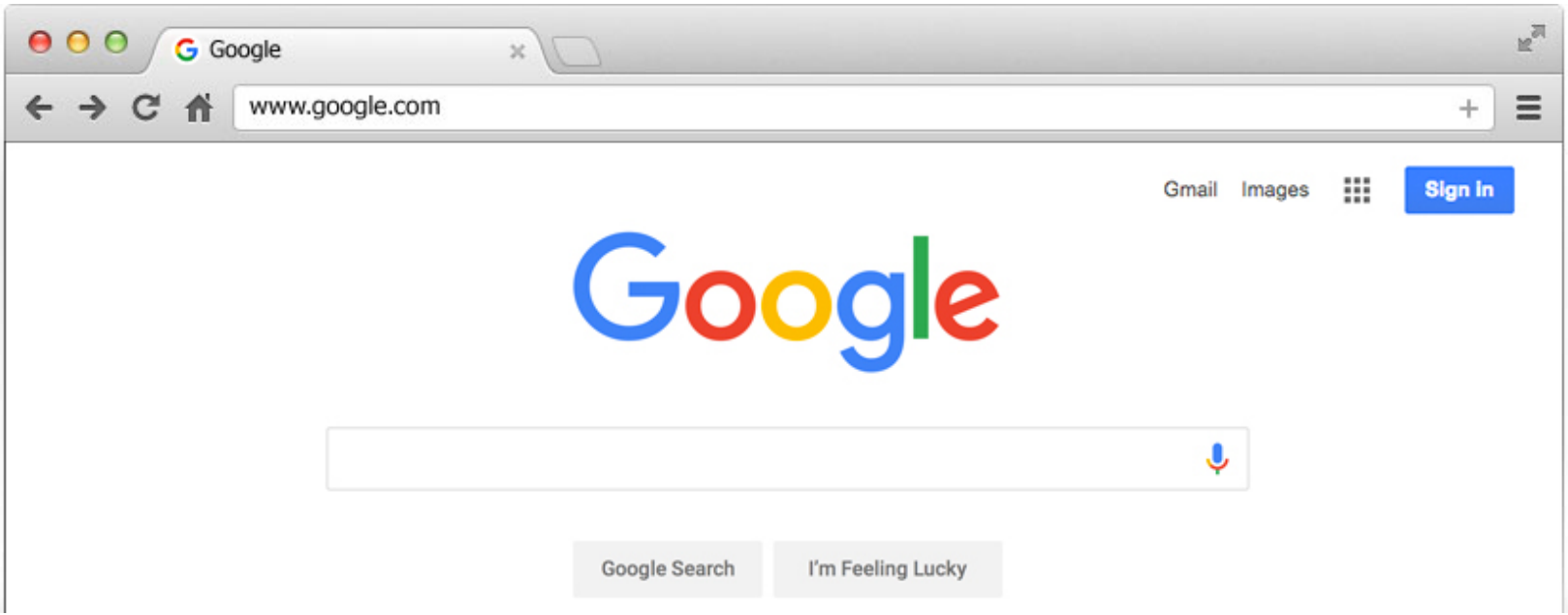
What are some alternate designs for networked filesystems? NFS → mid 80s

What is the role of OS in context of new trends like cloud computing?

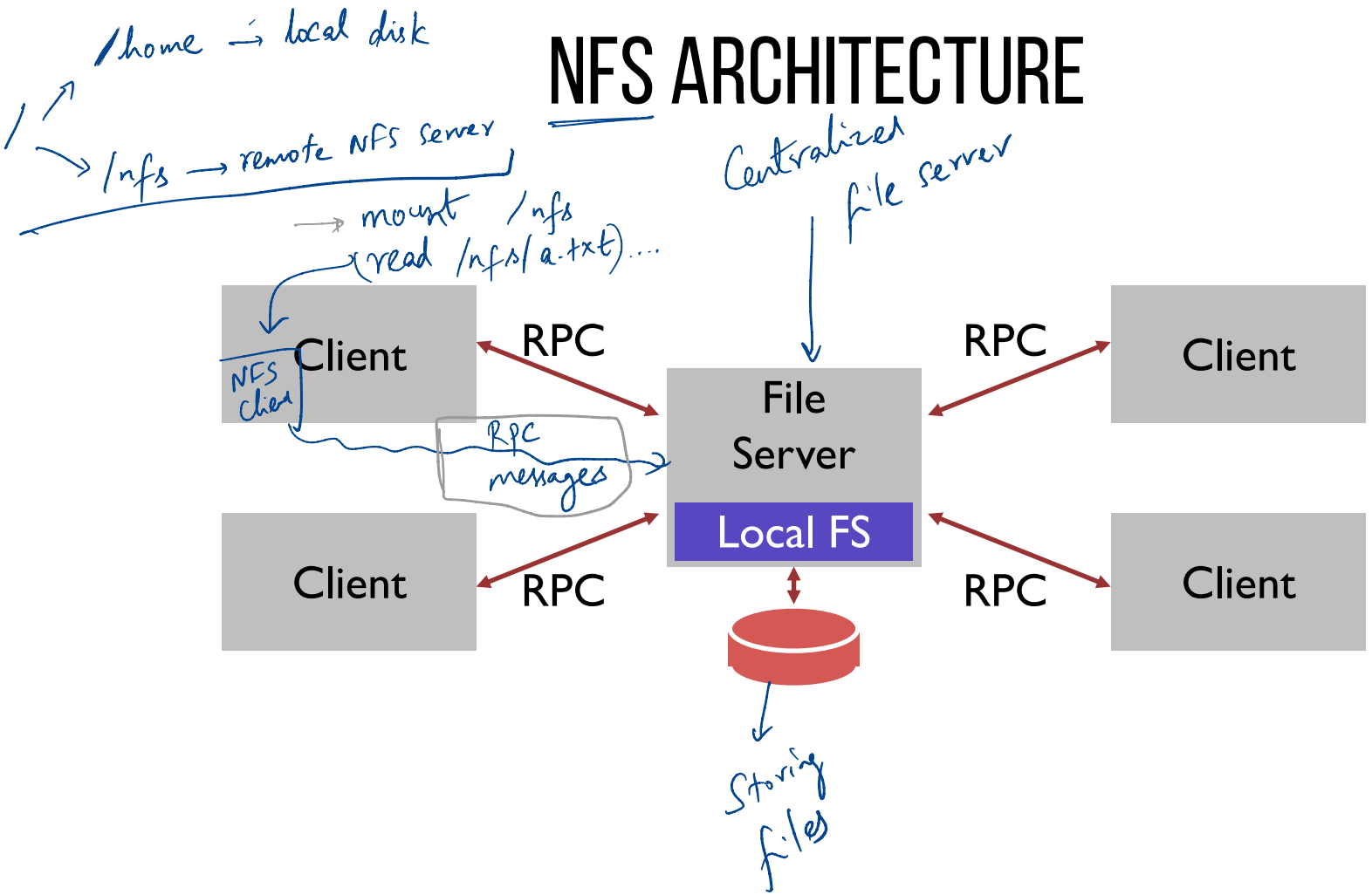
RECAP



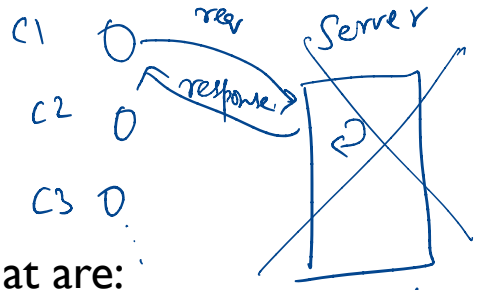
DISTRIBUTED SYSTEMS



NFS ARCHITECTURE



NFS SUMMARY



NFS handles client and server crashes very well; robust APIs that are:

- stateless: servers don't remember clients → File handle & client tracks offsets
- idempotent: doing things twice never hurts

Client retry requests to handle failure of server

Caching and write buffering is harder, especially with crashes

→ Update visibility → when do writes become visible

Problems: → Stale caches → when do caches get invalidated

- Consistency model is odd (client may not see updates until 3s after file closed)
- Scalability limitations as more clients call stat() on server

→ attribute cache duration

ALTERNATE DESIGN: ANDREW FILE SYSTEM (AFS)

Project at CMU

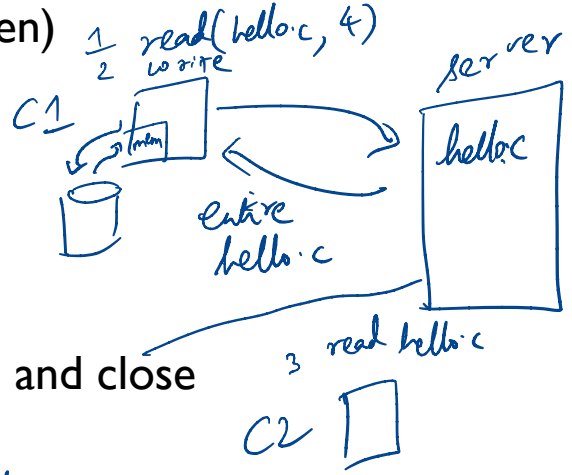
Still in use in our lab machines!!

Open - close semantics

WHOLE-FILE CACHING

Upon open, AFS client fetches whole file (even if huge), storing in local memory or disk

Upon close, client flushes file to server (if file was written)



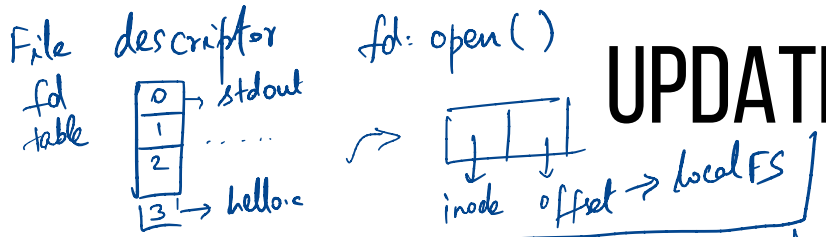
Convenient and intuitive semantics:

- AFS needs to do work only for open/close
- Reads/writes are local
- Use same version of file entire time between open and close

Pros → reads/writes incur no network traffic

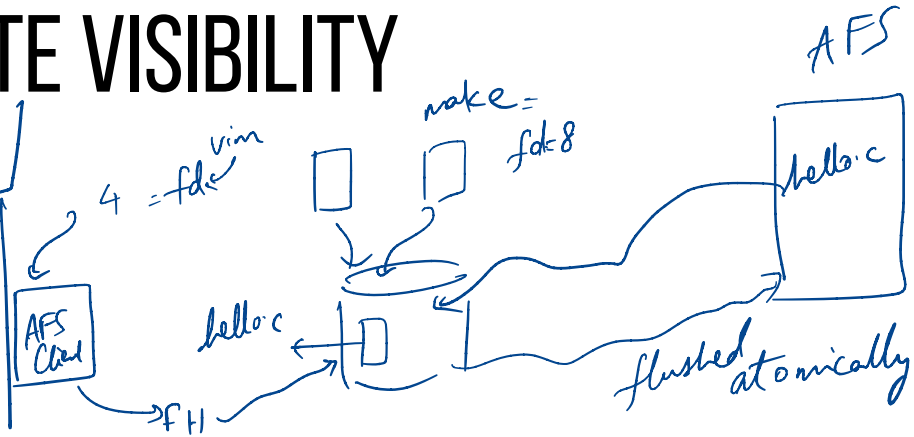
Cons → might need to fetch whole file even if you update just a few bytes.

UPDATE VISIBILITY



File handle which is internal to NFS/AFS

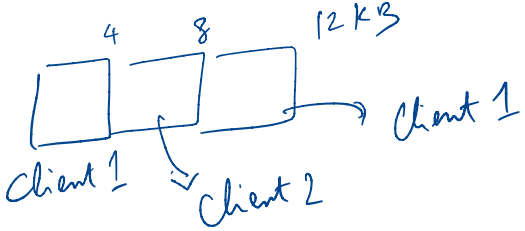
AFS solution: FD → FH mapping

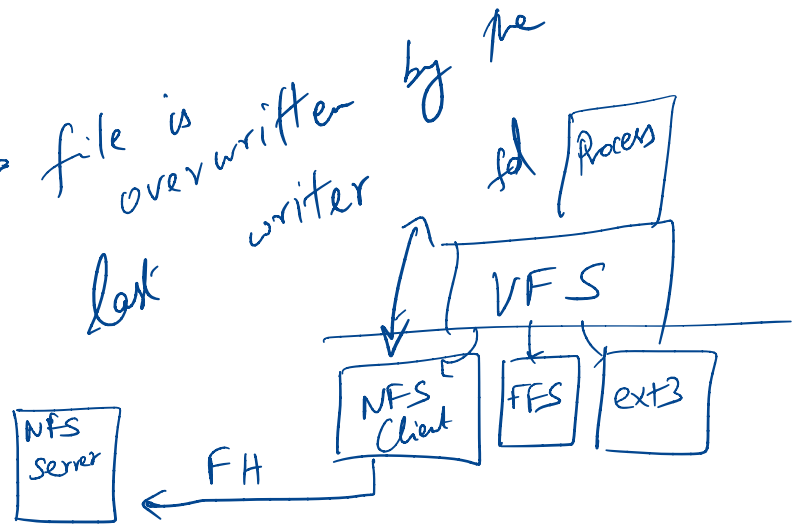


- also flush on close
- buffer whole files on local disk; update file on server atomically

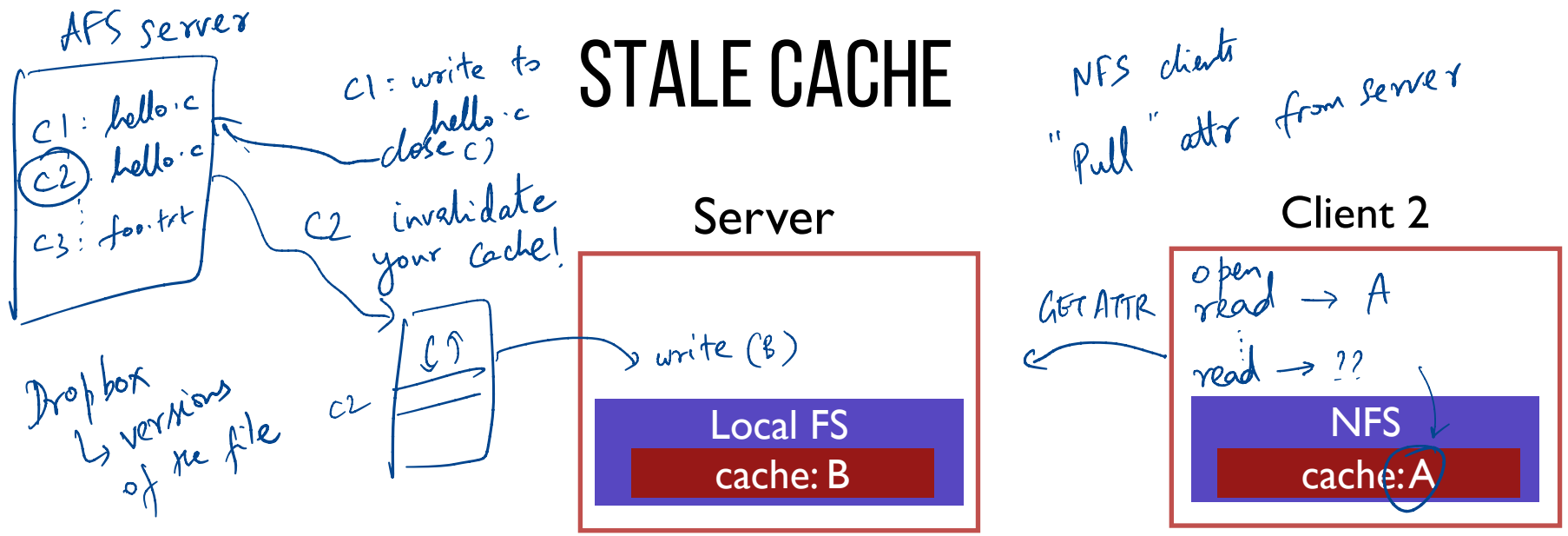
Concurrent writes?

- Last writer (i.e., last file closer) wins → file is overwritten by the last writer
- Never get mixed data on server





STALE CACHE



AFS solution: Tell clients when data is overwritten

- Server must remember which clients have this file open right now

When clients cache data, ask for "callback" from server if changes

- Clients can use data without checking all the time

Trade-offs

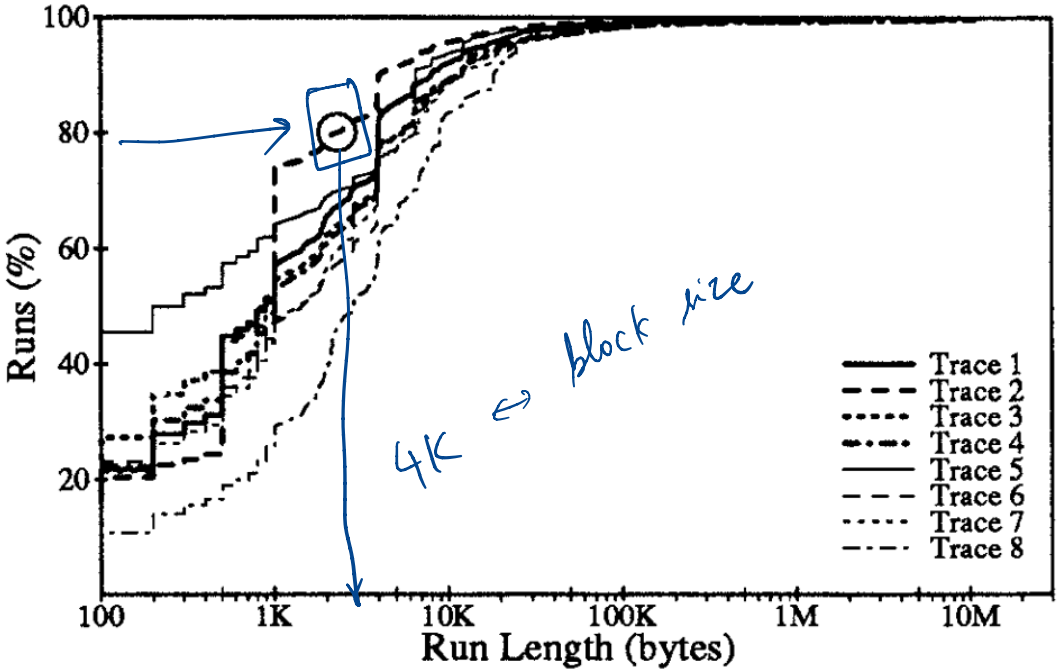
↓ reduces number of state

Server no longer stateless!

/home/shivaram

one user

WORKLOAD PATTERNS (1991)



Mary G. Baker, John H. Hartman, Michael D. Kupfer, Ken W. Shirriff, and John K. Ousterhout

↳ Madison

↳ LFS

WORKLOAD PATTERNS (1991)

university campus

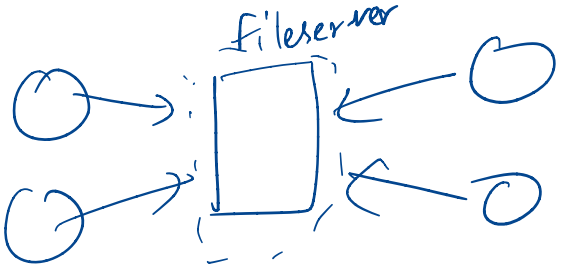
AFS design

File Usage	Type of Transfer	Accesses (%)		Bytes (%)	
Read-only	Whole-file	78	(64-91)	89	(46-96)
	Other sequential	19	(7-33)	5	(2-29)
	Random	3	(1-5)	7	(2-37)
Write-only	Whole-file	67	(50-79)	69	(56-76)
	Other sequential	29	(18-47)	19	(4-27)
	Random	4	(2-8)	11	(4-41)
Read/write	Whole-file	0	(0-0)	0	(0-0)
	Other sequential	0	(0-0)	0	(0-0)
	Random	100	(100-100)	100	(100-100)

OCEANSTORE/PAST

~2001

Berkeley



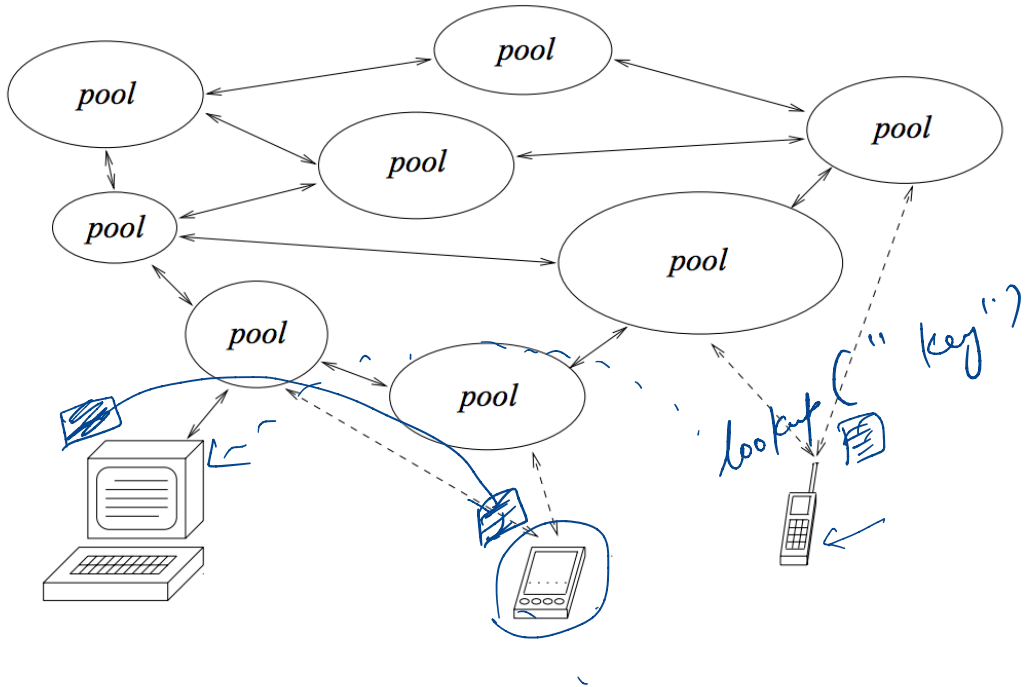
Wide area storage systems

P2P storage systems

Fully decentralized

Built on distributed hash tables (DHT)

File sharing
Bit torrent



OS/FILESYSTEMS FOR THE CLOUD?

FROM MID 2006

Rent virtual computers in the "Cloud" ~ 1$

On-demand machines, spot pricing

On-demand

AMAZON EC2 (2018)

Machine	Memory (GB)	Compute Units (ECU)	Local Storage (GB)	Cost / hour
t2.nano	0.5	1	0	$0.0058
r5d.24xlarge	~~244~~ 768	~~104~~ 96	4x900 NVMe	$6.912
x1.32xlarge	2 TB	4 * Xeon E7	3.4 TB (SSD)	$13.338
p3.16xlarge	488 GB	8 Nvidia Tesla V100 GPUs	0	$24.48

DATACENTER

multiple football fields

Google data centers in The Dulles, Oregon

DATACENTER EVOLUTION

Capacity:

~10000 machines

Bandwidth:

12-24 disks per node

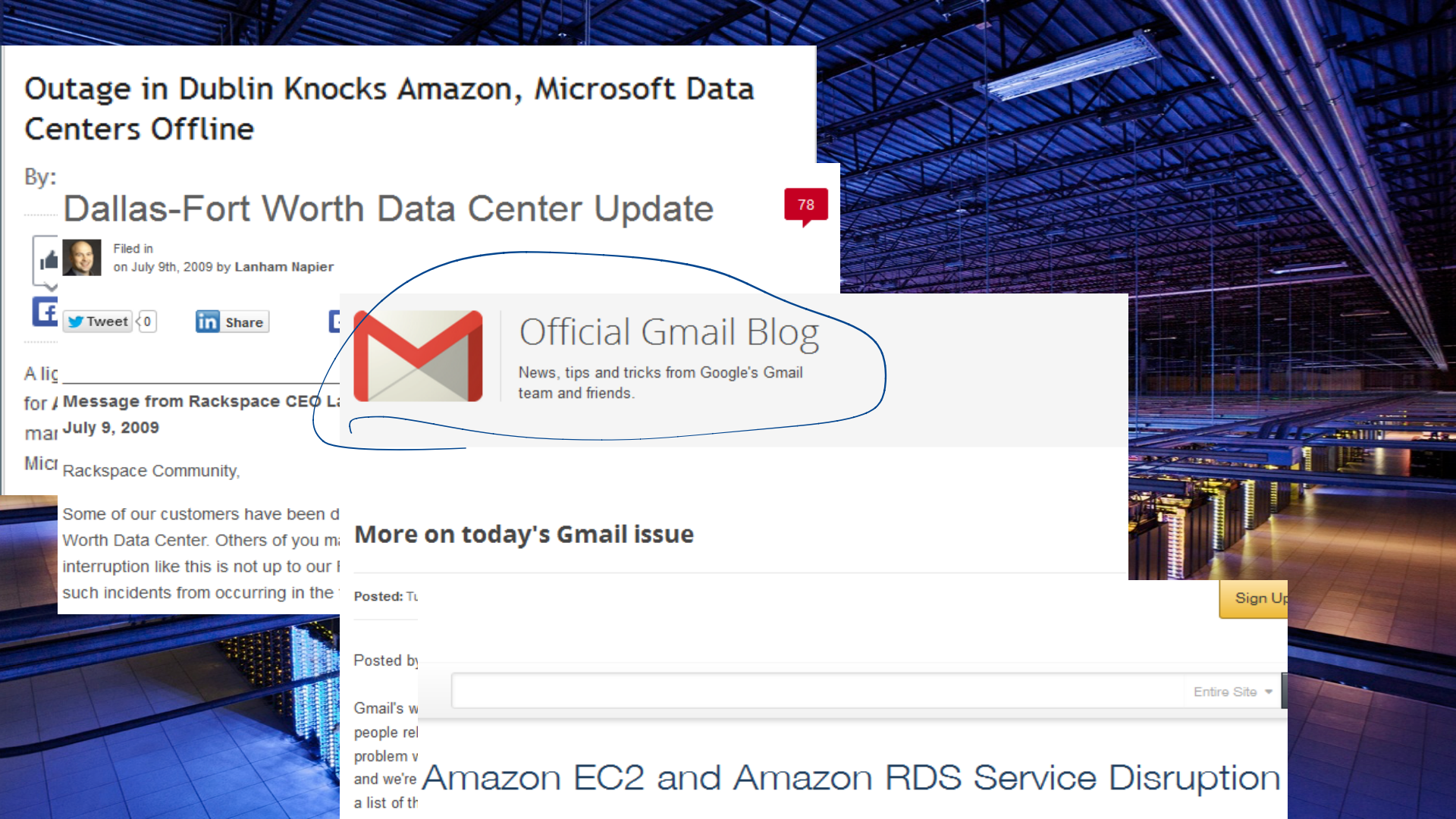
~1TB or 2TB ≃ 12 TB to 50 TB

Latency:

256GB RAM cache

or

2TB RAM



Outage in Dublin Knocks Amazon, Microsoft Data Centers Offline

By:

Dallas-Fort Worth Data Center Update

78

Filed in
on July 9th, 2009 by **Lanham Napier**

Tweet 0 Share

A lig
for A
mar
Micr

Message from Rackspace CEO La
July 9, 2009

Rackspace Community,

Some of our customers have been d
Worth Data Center. Others of you ma
interruption like this is not up to our h
such incidents from occurring in the

Official Gmail Blog

News, tips and tricks from Google's Gmail team and friends.

More on today's Gmail issue

Posted: Tu

Sign Up

Posted by

Gmail's w
people rel
problem v
and we're
a list of th

Entire Site

Amazon EC2 and Amazon RDS Service Disruption

The Joys of Real Hardware

Typical first year for a new cluster:

~0.5 overheating (power down most machines in <5 mins, ~1-2 days to recover)
~1 PDU failure (~500-1000 machines suddenly disappear, ~6 hours to come back)
~1 rack-move (plenty of warning, ~500-1000 machines powered down, ~6 hours)
~1 network rewiring (rolling ~5% of machines down over 2-day span)
~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
~5 racks go wonky (40-80 machines see 50% packetloss)
~8 network maintenances (4 might cause ~30-minute random connectivity losses)
~12 router reloads (takes out DNS and external vips for a couple minutes)
~3 router failures (have to immediately pull traffic for an hour)
~dozens of minor 30-second blips for dns
~1000 individual machine failures
~thousands of hard drive failures
slow disks, bad memory, misconfigured machines, flaky machines, etc.

Long distance links: wild dogs, sharks, dead horses, drunken hunters, etc.

Partial Failure is not rare but common

JEFF DEAN @ GOOGLE

FEEDBACK! ~41%

https://aefis.wisc.edu/

1. What was one idea or concept that you learnt in this course that you appreciated the most?

2. What are some future opportunities that you look forward to based on content from 537?

ALTERNATE DESIGN: GOOGLE FILE SYSTEM (GFS)

~ 2003

open source ~ 2006

GFS: WORKLOAD ASSUMPTIONS

- "Modest" number of large files (vs. large number of small files)
- Two kinds of reads: Large Streaming and small random
- Writes: Many large, sequential writes. No random
- High bandwidth more important than low latency

Building a web search index

MapReduce workloads

GFS: DESIGN

- Single Master for metadata

- Chunkservers for storing data

- No POSIX API !
- No Caches!

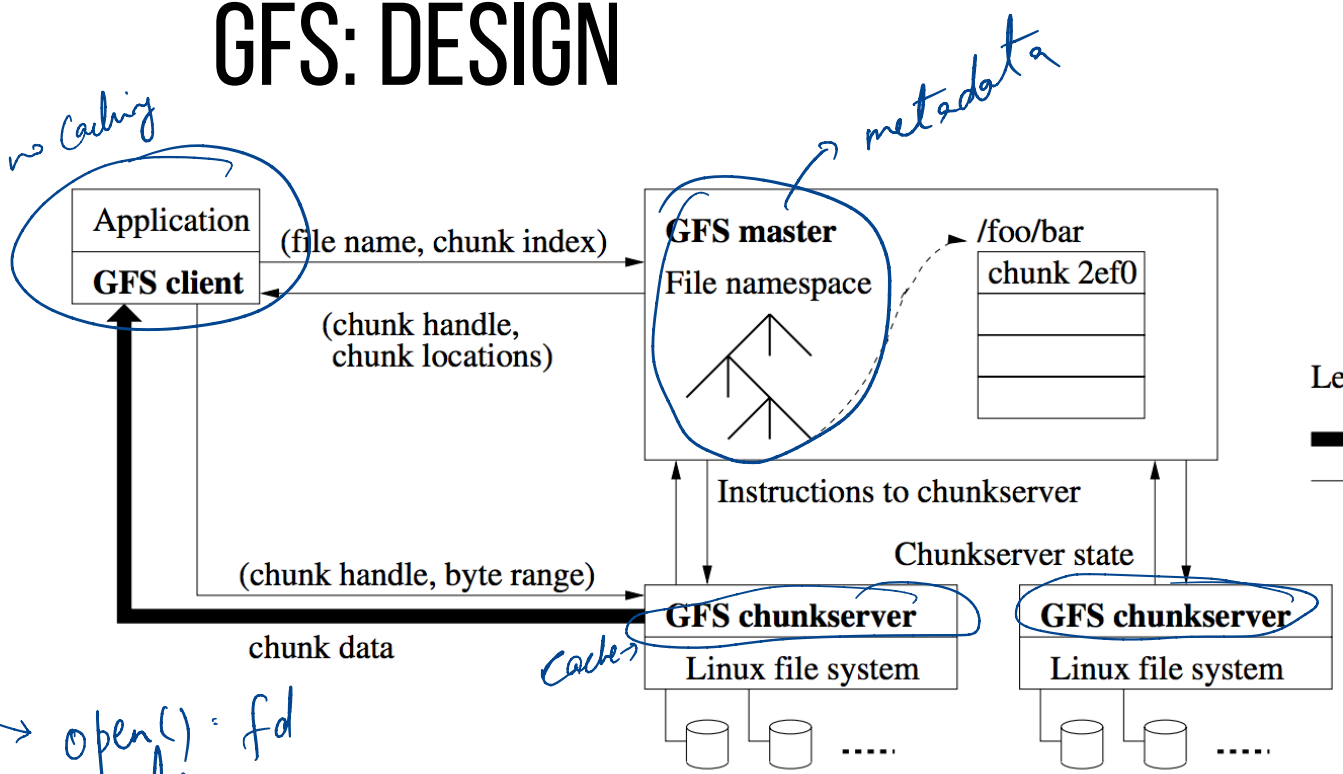


Figure 1: GFS Architecture

The Datacenter Needs an Operating System

~2010

Matei Zaharia, Benjamin Hindman, Andy Konwinski, Ali Ghodsi,
Anthony D. Joseph, Randy Katz, Scott Shenker, Ion Stoica
University of California, Berkeley

1 Introduction

Clusters of commodity servers have become a major computing platform, powering not only some of today's most popular consumer applications—Internet services such as search and social networks—but also a growing number of scientific and enterprise workloads [2]. This rise in cluster computing has even led some to declare that "the datacenter is the new computer" [16, 24]. However, the tools for managing and programming this new computer are still immature. This paper argues that, due to the growing diversity of cluster applications and users, the datacenter increasingly needs an operating system.[1]

and Pregel steps). However, this is currently difficult because applications are written independently, with no common interfaces for accessing resources and data.

In addition, clusters are serving increasing numbers of concurrent users, which require responsive time-sharing. For example, while MapReduce was initially used for a small set of batch jobs, organizations like Facebook are now using it to build data warehouses where hundreds of users run near-interactive ad-hoc queries [29].

Finally, programming and debugging cluster applications remains difficult even for experts, and is even more challenging for the growing number of non-expert users

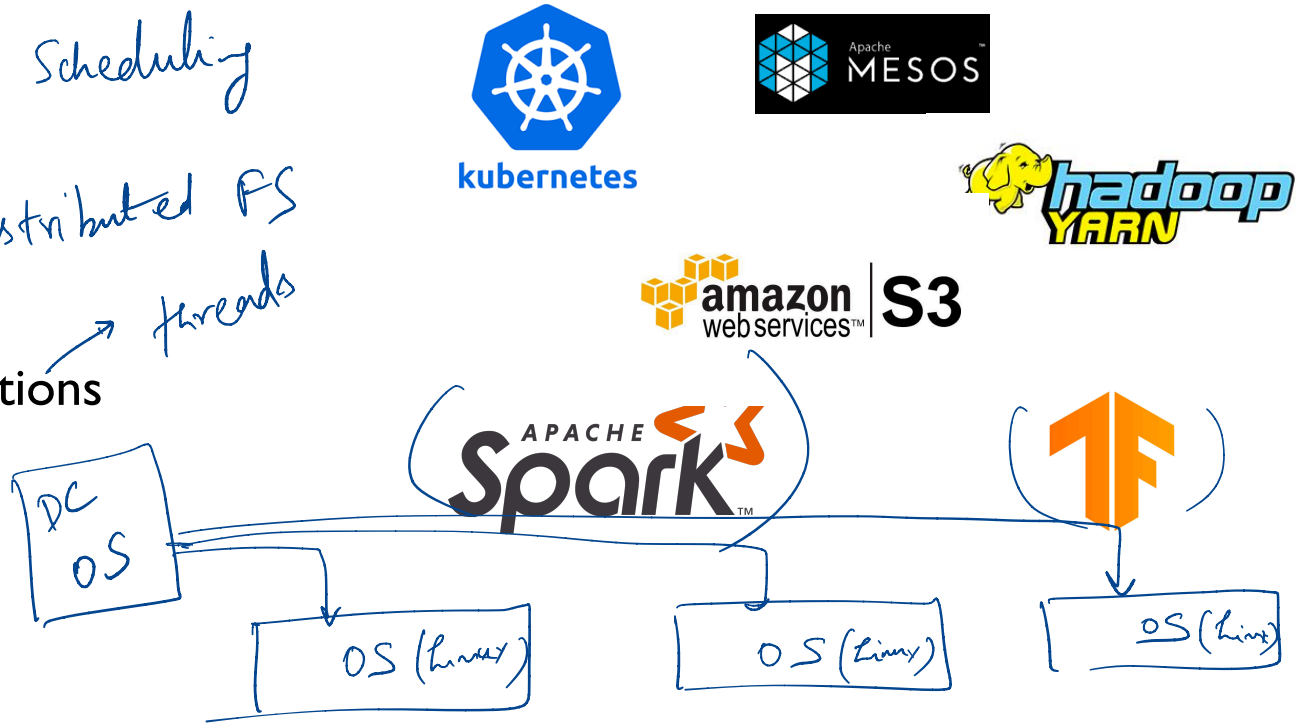
DATACENTER OPERATING SYSTEMS

Resource sharing → Scheduling

Data sharing → Distributed FS

Programming Abstractions → threads

Debugging



COURSE SUMMARY

OPERATING SYSTEMS: THREE EASY PIECES

Three conceptual pieces

1. Virtualization

2. Concurrency

3. Persistence

VIRTUALIZATION

Make each application believe it has each resource to itself

CPU and Memory

Abstraction: Process API, Address spaces → virtual memory

Mechanism:

- Limited direct execution, CPU scheduling
- Address translation (segmentation, paging, TLB)

Policy: MLFQ, LRU etc.

CONCURRENCY

Events occur simultaneously and may interact with one another

Need to

- Hide concurrency from independent processes
- Manage concurrency with interacting processes → 2 process / threads want to share data

Provide abstractions (locks, semaphores, condition variables etc.)

Correctness: mutual exclusion, ordering

Performance: scaling data structures, fairness

Common Bugs!

↳ Deadlocks!

PERSISTENCE

Managing devices: key role of OS!

- Hard disk drives → Device
  - Rotational, Seek, Transfer time
  - Disk scheduling: FIFO, SSTF, SCAN
- Filesystems API
  - File descriptors, Inodes → metadata
  - Directories
  - Hardlinks, softlinks

PERSISTENCE

Very simple FS

- Inodes, Bitmaps, Superblock, Data blocks

FFS

- Placement in groups, Allocation policy

LFS

- Write optimized, Garbage collection

Journaling, FSCK

NFS: Partial failures retry, cache consistency

LOOKING BACK, LOOKING FORWARD

Idea or concept you learnt

→ Concurrency is important!

future opportunities?

CS grad courses (744)

Security as a career

Kubernetes

Is OS research done?

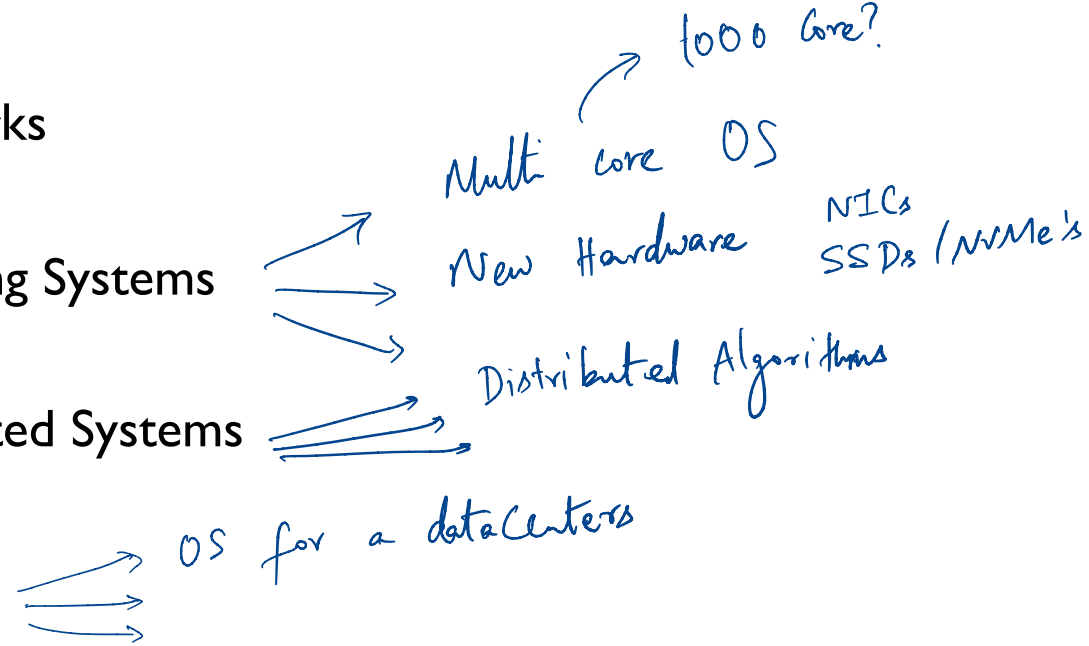
NEXT COURSES

CS 640: Computer Networks

CS 736: Advanced Operating Systems

CS 739: Advanced Distributed Systems

CS 744: Big Data Systems



THANK YOU!