Good morning!

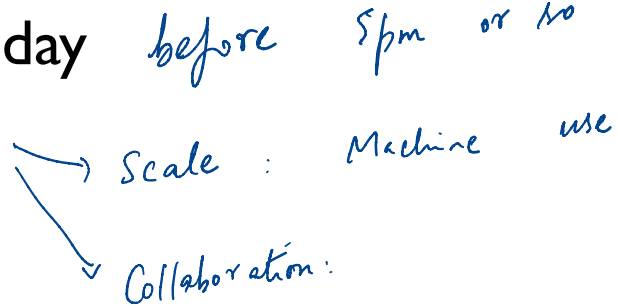
CS 744: GOOGLE FILE SYSTEM

Shivaram Venkataraman

Fall 2020

ANNOUNCEMENTS

- Assignment 1 out later today
- Group submission form
- Anybody on the waitlist?

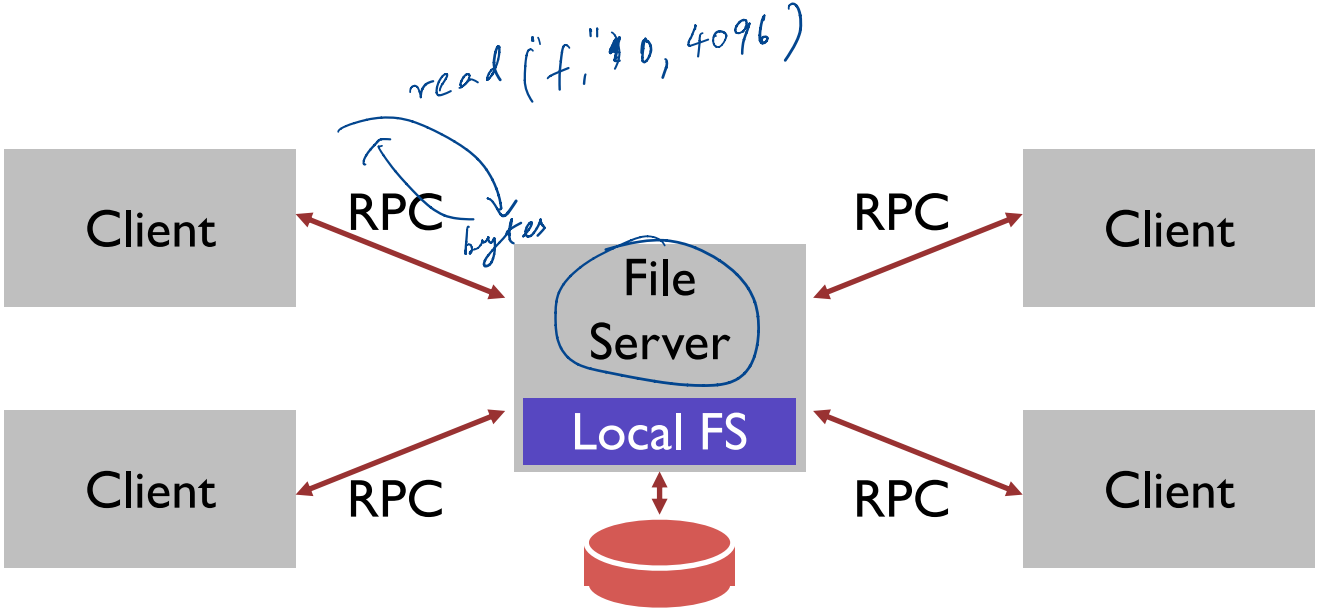


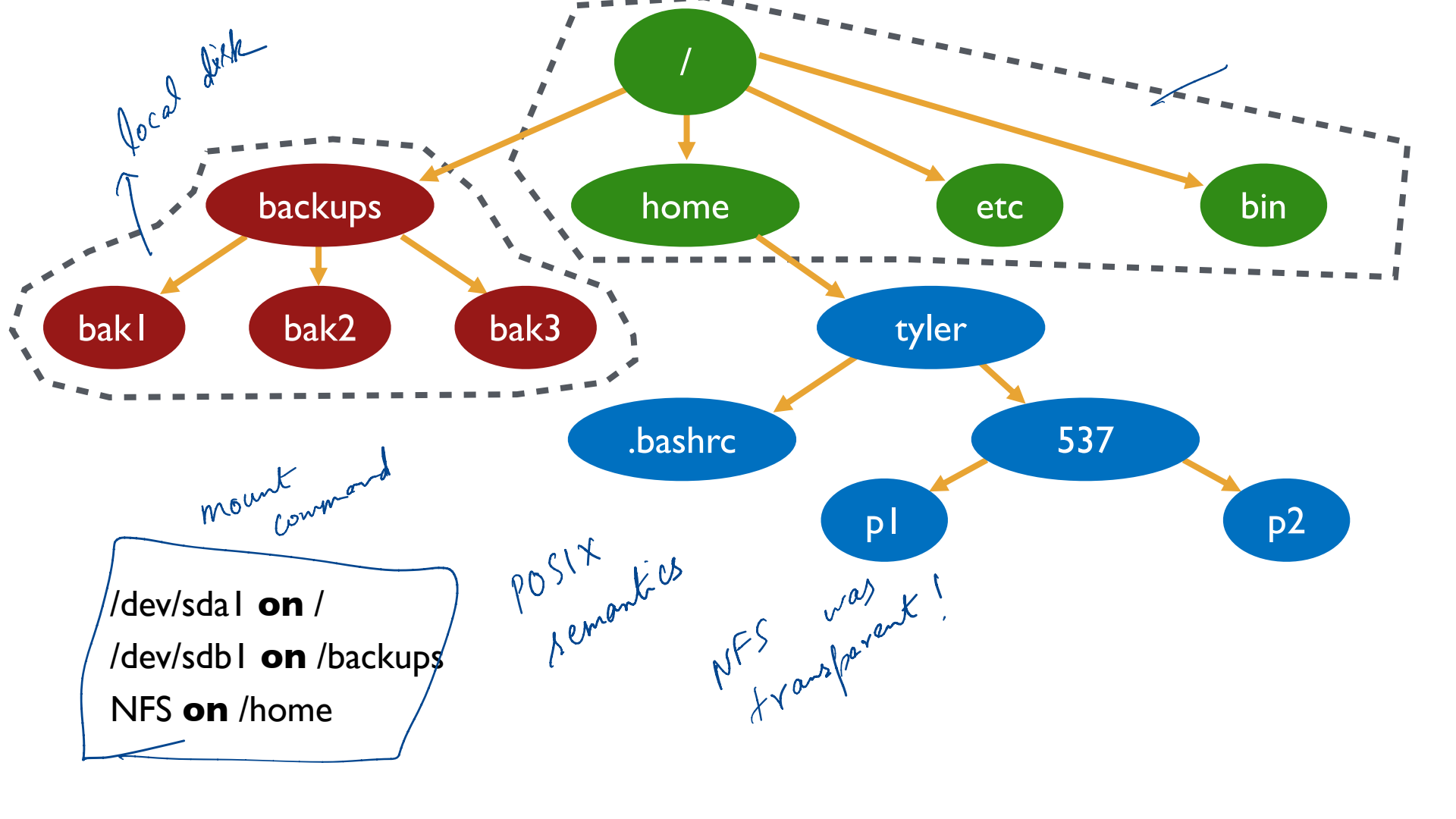
OUTLINE

1. Brief history
2. GFS
3. Discussion
4. What happened next?

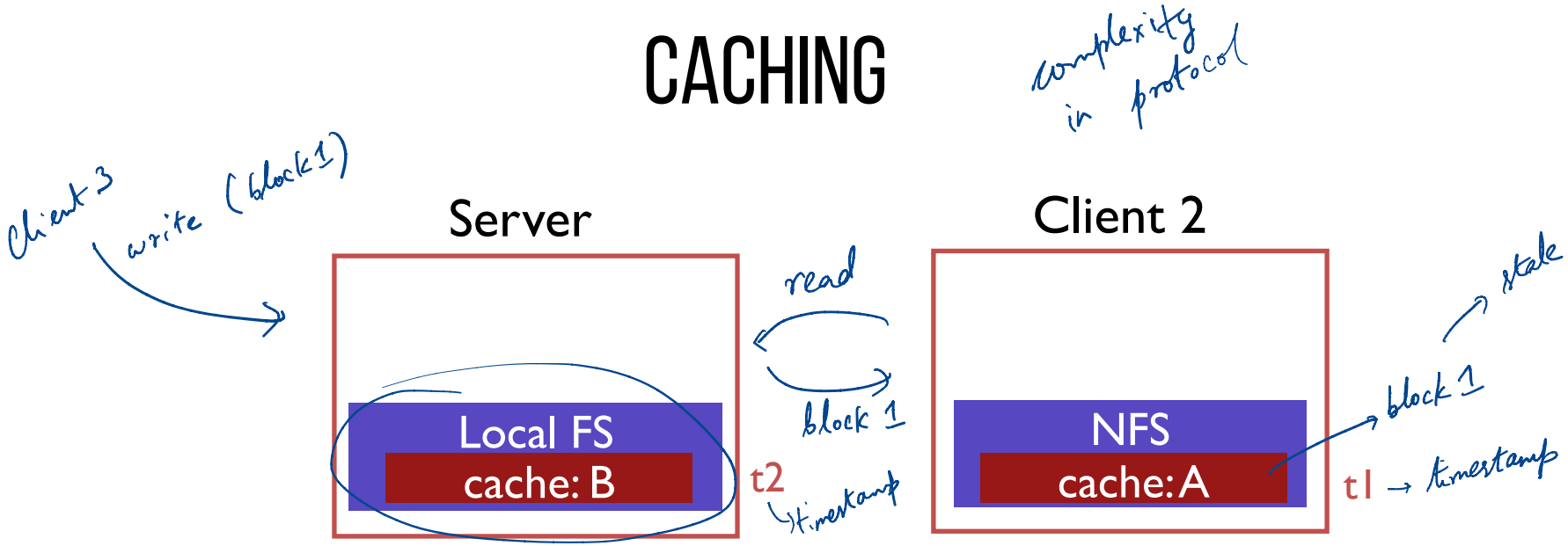
HISTORY OF DISTRIBUTED FILE SYSTEMS

SUN NFS → CS 537




local disk
/
backups
home
etc
bin
bak1
bak2
bak3
tyler
.bashrc
537
p1
p2
mount command
/dev/sda1 on /
/dev/sdb1 on /backups
NFS on /home
POSIX semantics
NFS was transparent!

CACHING



Client cache records time when data block was fetched (t1)

Before using data block, client does a STAT request to server

- get's last modified timestamp for this file (t2) (not block…)
- compare to cache timestamp
- refetch data block if changed since timestamp (t2 > t1)

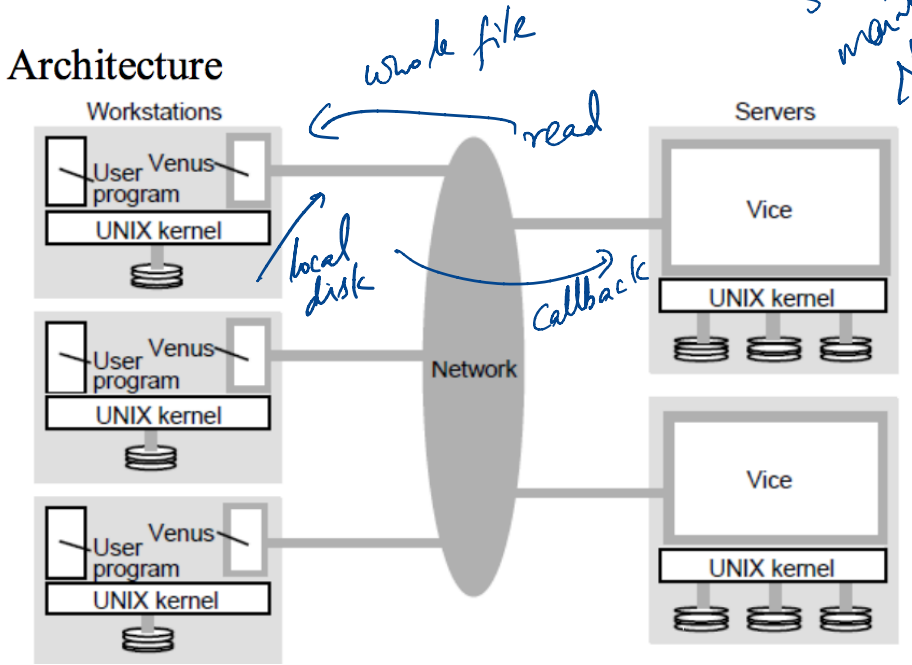
NFS ~ mid 80s

ANDREW FILE SYSTEM

- Design for scale

- Whole-file caching

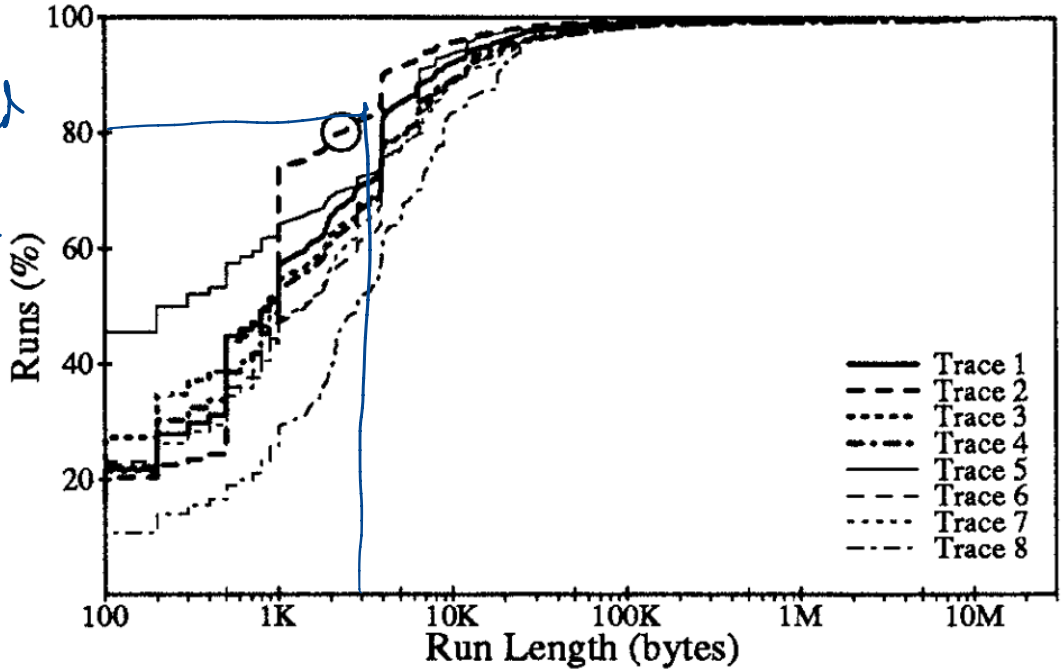
- Callbacks from server



WORKLOAD PATTERNS (1991)

majority of files requested was 1 4K

Workload patterns in lab setup



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

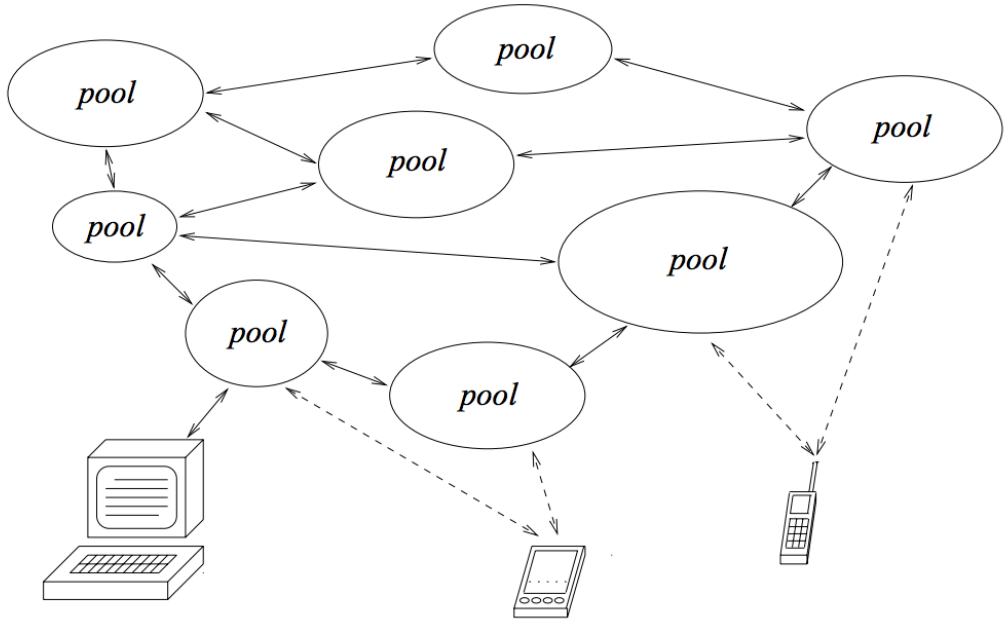
BitTorrent

OCEANSTORE/PAST ≃ late 90's / early 00

Wide area storage systems

Fully decentralized

Built on distributed hash tables (DHT)



GFS: WHY ?

Workloads
→ Files are large!
Access pattern: sequential write/read
Appends

Fault tolerance
→ Components that had frequent failures

Scalability
↳ number of concurrent writers

large scale

Components with failures

Files are huge !

GFS: WHY ?

Motivation

Applications are different

append

concurrent writers

GFS: WORKLOAD ASSUMPTIONS

"Modest" number of large files

Two kinds of reads: Large Streaming and small random

Writes: Many large, sequential writes. No random

High bandwidth more important than low latency

GFS: DESIGN

- Single Master for metadata

- Chunkservers for storing data

- No POSIX API !
- No Caches!

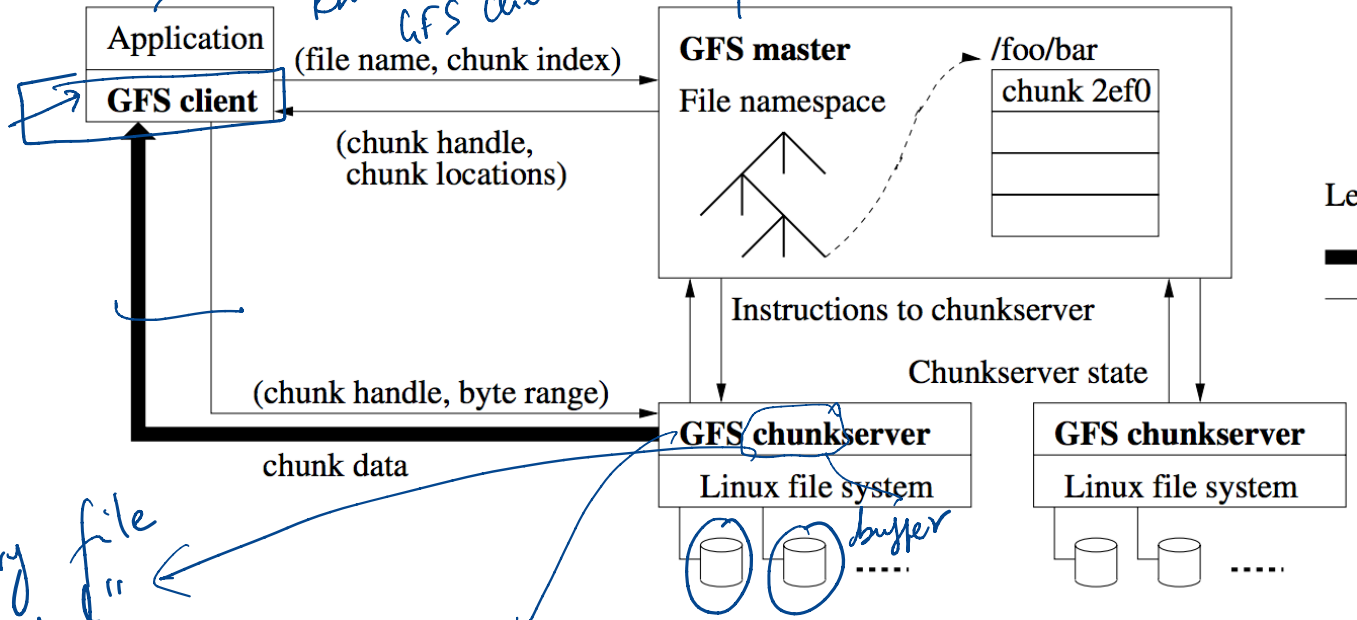


Figure 1: GFS Architecture

CHUNK SIZE TRADE-OFFS

Client → Master → smaller chunks → more metadata in reply

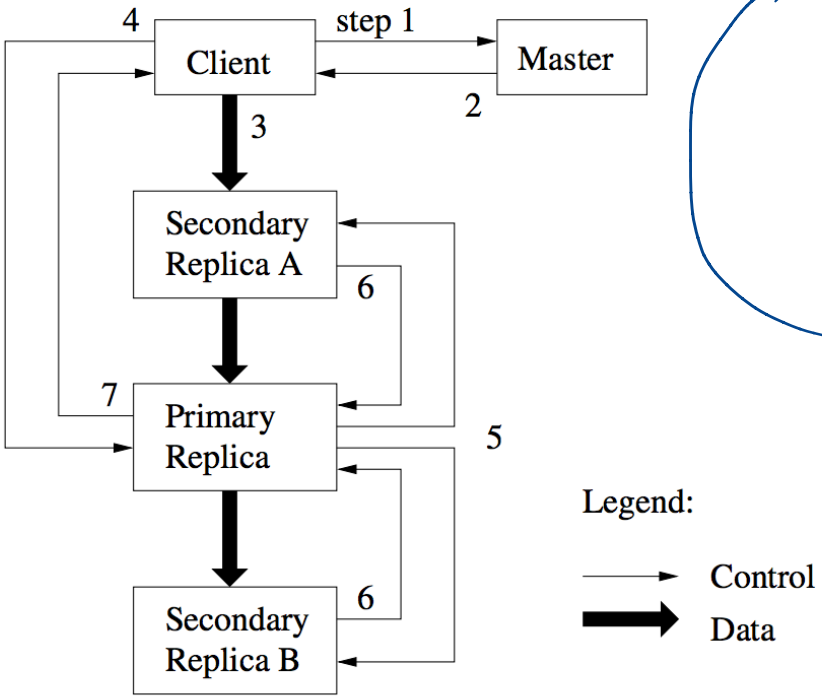
Client → Chunkserver → larger chunks → more hotspots / more requests to same chunk server

Metadata → larger chunks → less metadata

larger chunk → fragmentation?

≈ 64 MB in 2003

GFS: REPLICATION



- 3-way replication to handle faults
- Primary replica for each chunk
- Chain replication (consistency)
- Decouple data, control flow
- Dataflow: Pipelining, network-aware

RECORD APPENDS

Consistency model is tricky
↳ Applications are resistant!

Write	Client specifies the offset
Record Append	GFS chooses offset

↳ Primary replica for the chunk

Consistency

- At-least once → because there might be failures
- Atomic → entire record appears together

no symlinks

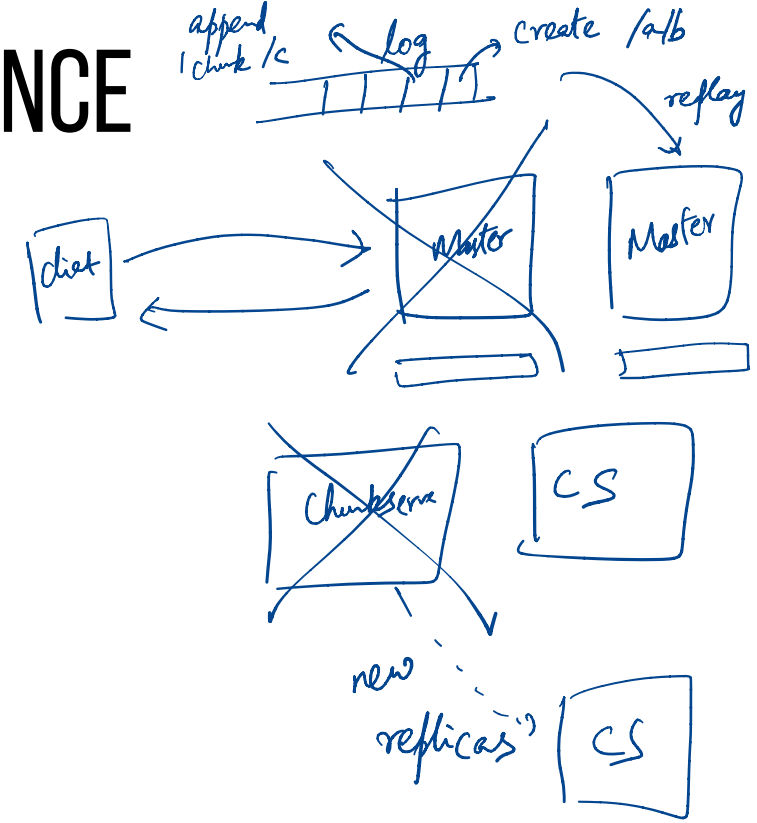
MASTER OPERATIONS

- No "directory" inode! Simplifies locking
- Replica placement considerations
  - not on same rack → failure
  - disk utilization
  - operations (write)
- Implementing deletes
  - lazy garbage collection

no data structure that tracks files in a directory

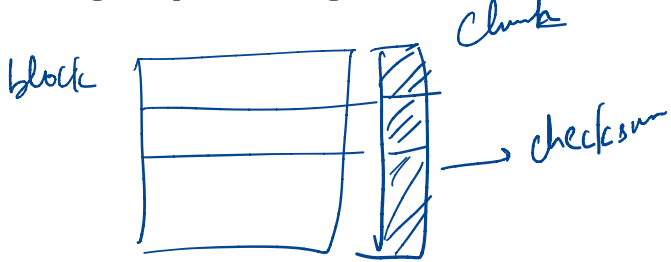
Key	Value
/a/b/c	<metadata>
/a/b	
/a	

FAULT TOLERANCE



- Chunk replication with 3 replicas
- Master
  - Replication of log, checkpoint
  - Shadow master

- Data integrity using checksum blocks



DISCUSSION

https://forms.gle/iUJh1MeVkKVRkt2X7

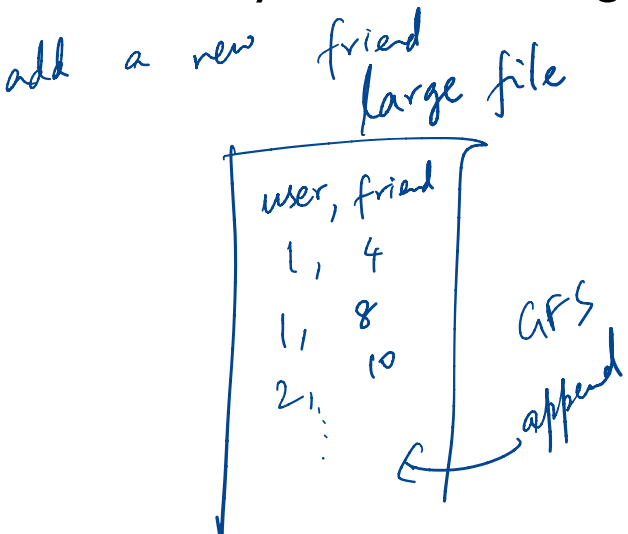
GFS SOCIAL NETWORK

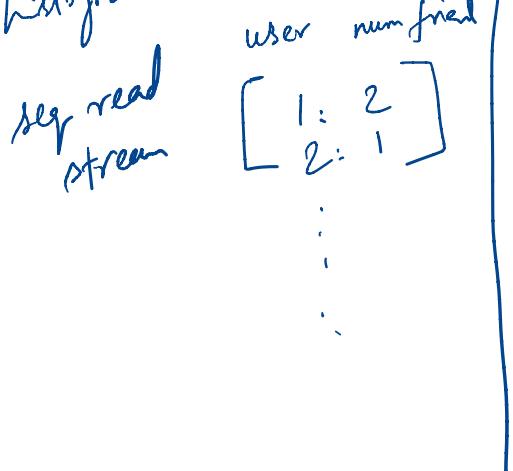
You are building a new social networking application. The operations you will need to perform are

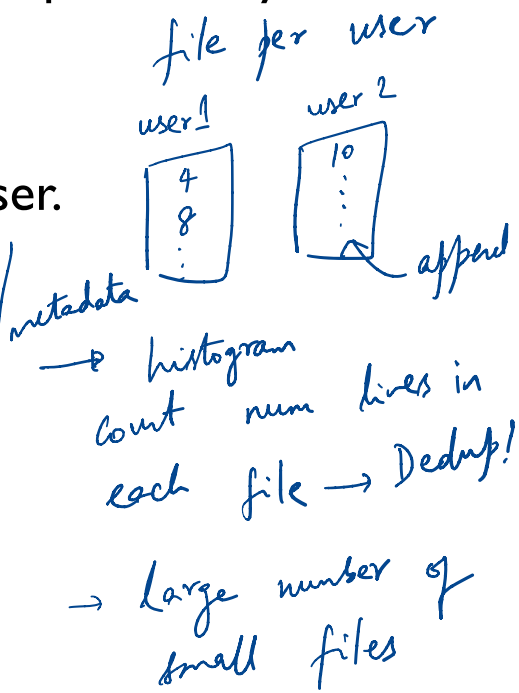
(a) add a new friend id for a given user

(b) generate a histogram of number of friends per user.

How will you do this using GFS as your storage system ?



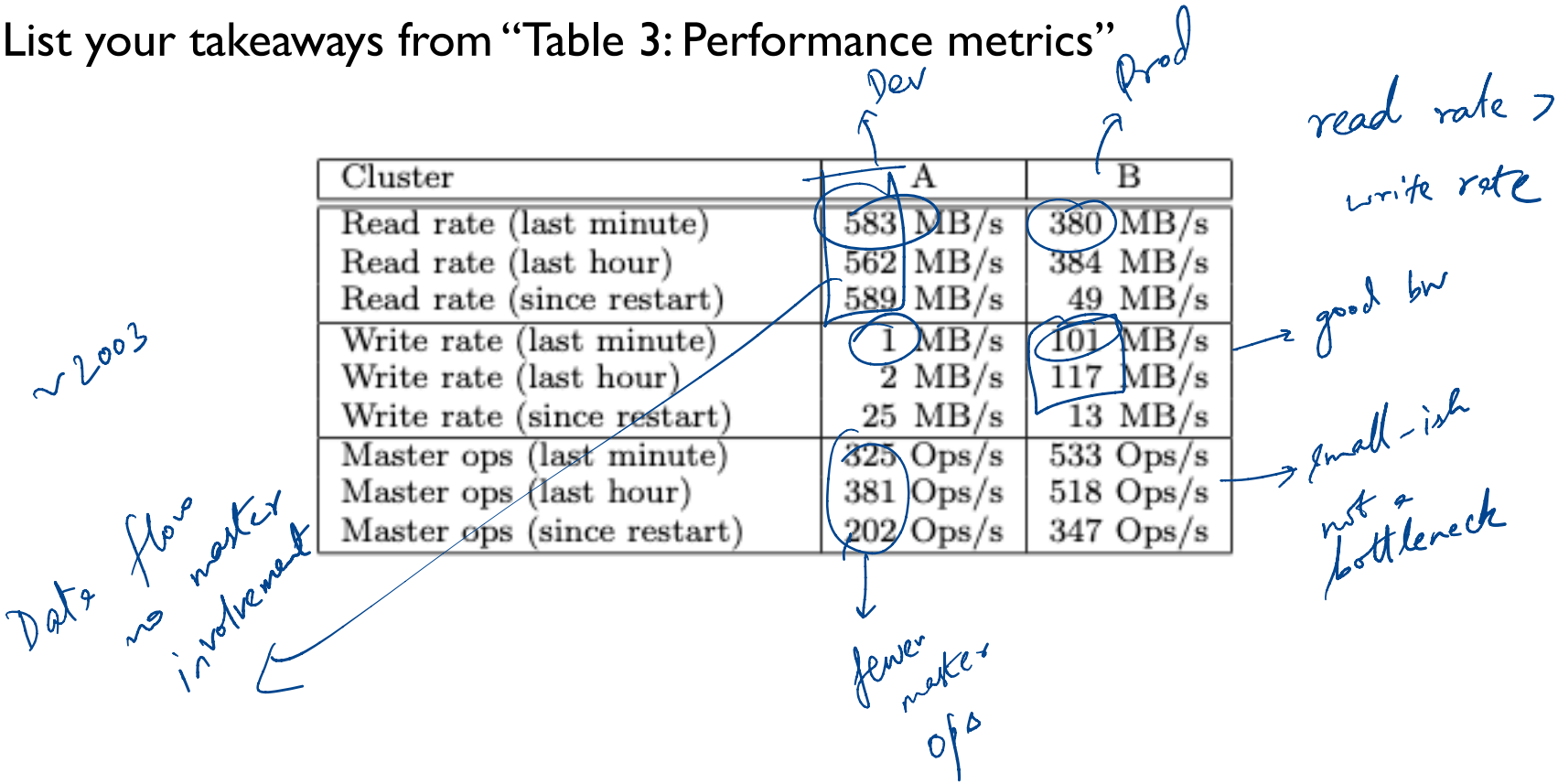




GFS EVAL

List your takeaways from "Table 3: Performance metrics"

Cluster	A	B
Read rate (last minute)	583 MB/s	380 MB/s
Read rate (last hour)	562 MB/s	384 MB/s
Read rate (since restart)	589 MB/s	49 MB/s
Write rate (last minute)	1 MB/s	101 MB/s
Write rate (last hour)	2 MB/s	117 MB/s
Write rate (since restart)	25 MB/s	13 MB/s
Master ops (last minute)	325 Ops/s	533 Ops/s
Master ops (last hour)	381 Ops/s	518 Ops/s
Master ops (since restart)	202 Ops/s	347 Ops/s



GFS SCALE

The evaluation (Table 2) shows clusters with up to 180 TB of data. What part of the design would need to change if we instead had 180 PB of data?

WHAT HAPPENED NEXT

Cluster-Level Storage @ Google

How we use *Colossus* to improve storage efficiency

Denis Serenyi
Senior Staff Software Engineer
dserenyi@google.com

Keynote at PDSW-DISCS 2017: 2nd Joint International Workshop On Parallel Data Storage & Data Intensive Scalable Computing Systems

GFS EVOLUTION

Motivation:

- GFS Master
  - One machine not large enough for large FS
  - Single bottleneck for metadata operations (data path offloaded)
  - Fault tolerant, but not HA

- Lack of predictable performance
  - No guarantees of latency
  - (GFS problems: one slow chunkserver -> slow writes)

GFS EVOLUTION

GFS master replaced by Colossus

Metadata stored in BigTable

Recursive structure ? If Metadata is ~1/10000 the size of data

100 PB data → 10 TB metadata

10TB metadata → 1GB metametadata

1GB metametadata → 100KB meta...

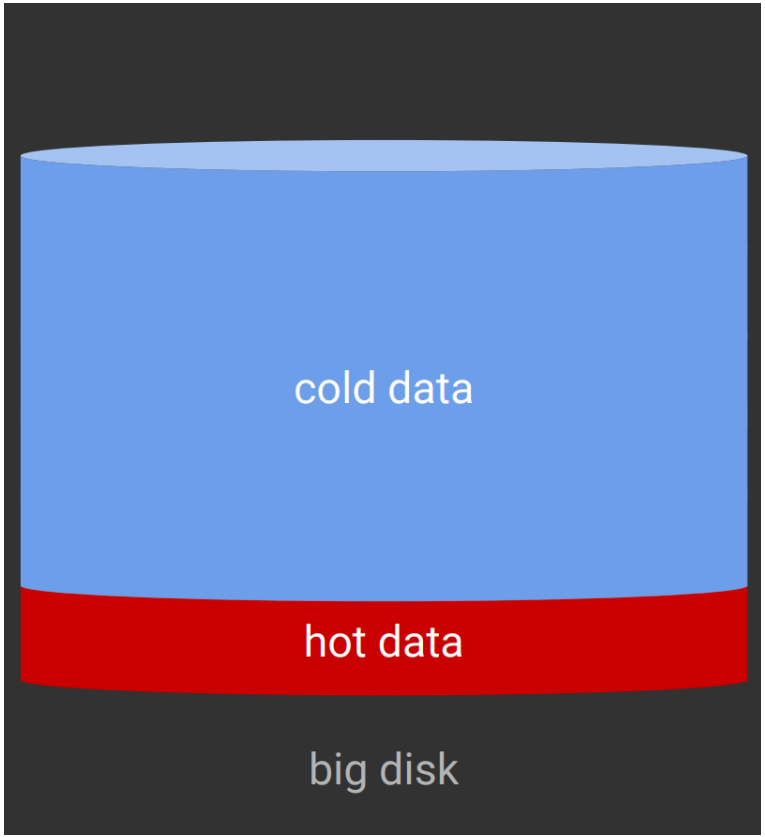
GFS EVOLUTION

Need for Efficient Storage

Rebalance old, cold data

Distributes newly written data evenly across disk

Manage both SSD and hard disks



HETEROGENEOUS STORAGE

F4: Facebook

Blob stores

Key Value Stores

NEXT STEPS

- Assignment 1 out tonight!
- Next week: MapReduce, Spark