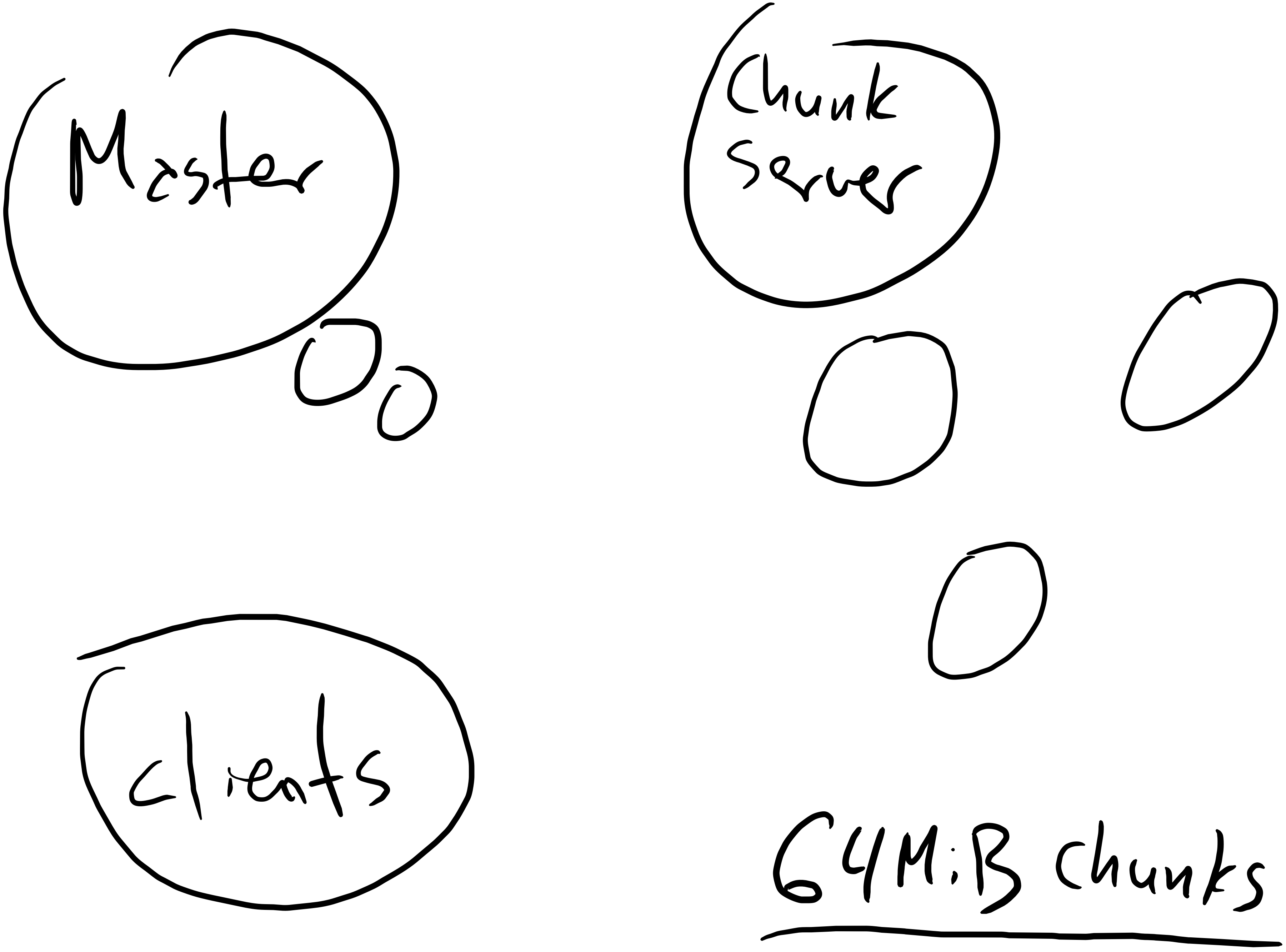
# CS244b - GFS

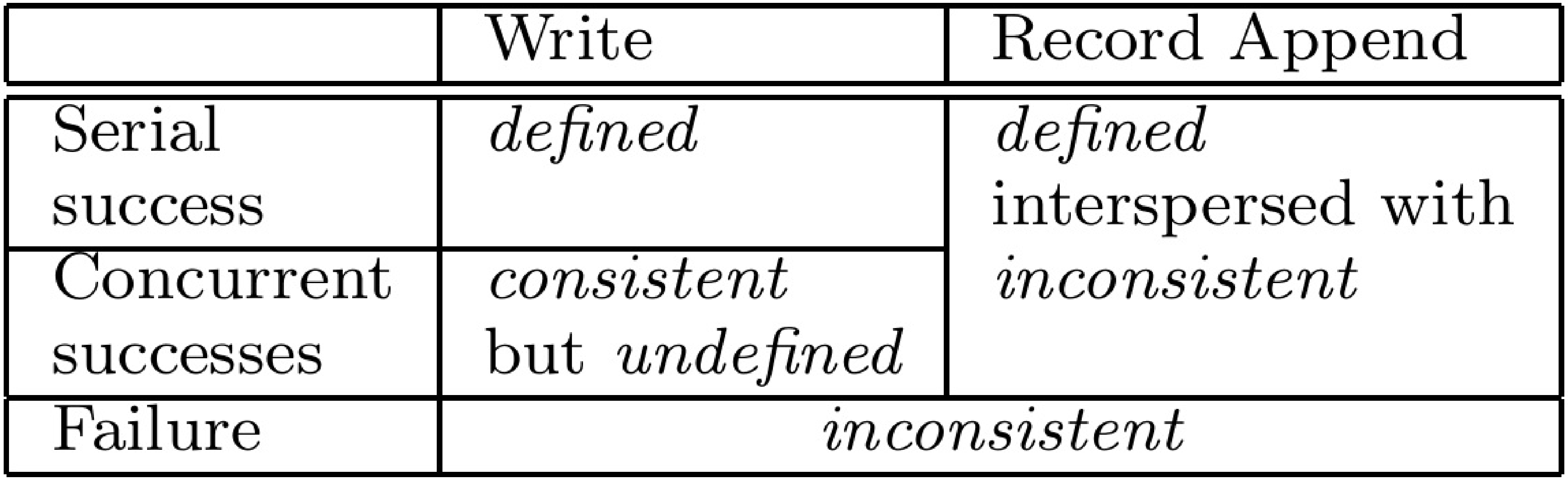
Learning Goals:

- Application/infrastructure co-design
- Restraint


Master
Chunk Server
clients
64MiB chunks

# API

- create
- snapshot
- read
- write
- append at least once
- FindMatchingFiles
- delete



| | Write | Record Append |
|---|---|---|
| Serial success | *defined* | *defined* interspersed with *inconsistent* |
| Concurrent successes | *consistent* but *undefined* | |
| Failure | *inconsistent* | |

**Table 1: File Region State After Mutation**

undef - data could be anything

inconsistent -

# Master State

File name → Metadata*, Chunks*, Locks

Chunk → Replicas, Version*, Reference Count*, Lease

Log*

Checkpoints*

# Chunk Server State

Chunks 64MiB

Version #, Checksum

serial num. leases

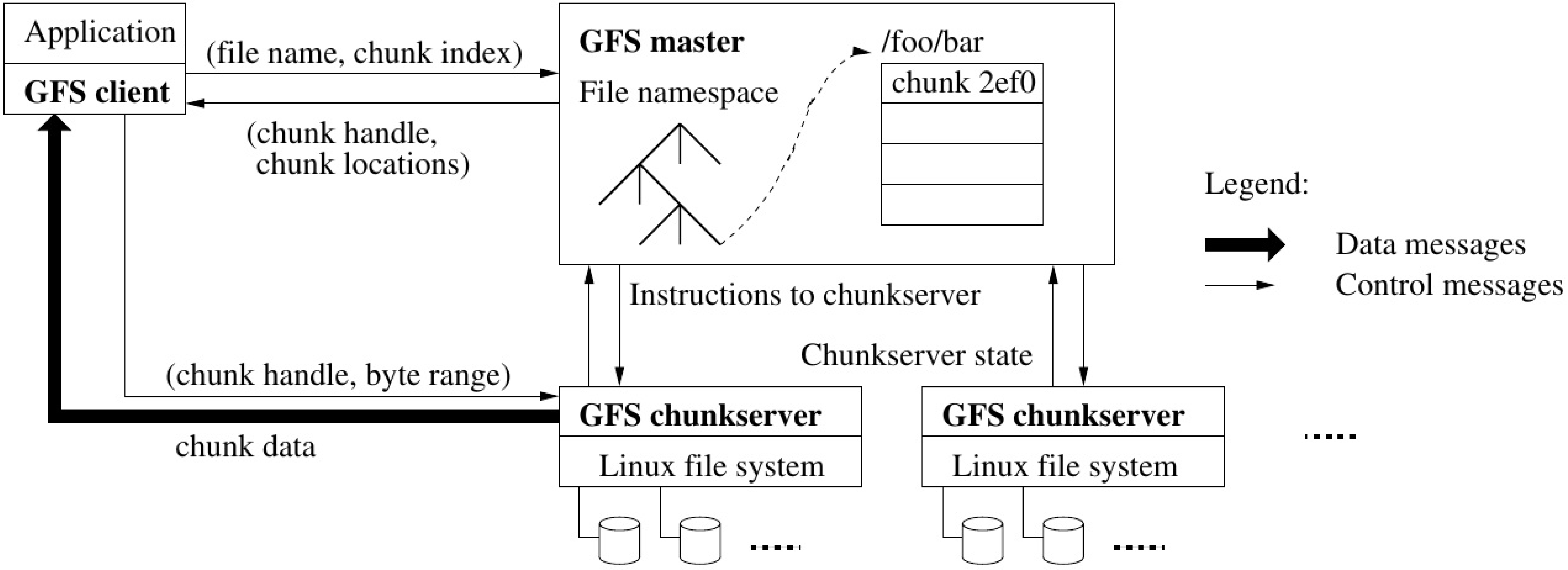
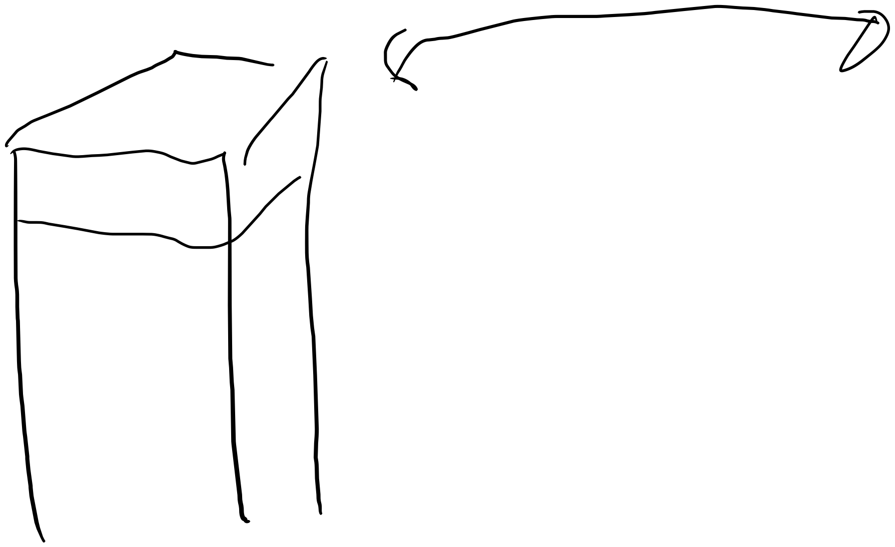


**Figure 1: GFS Architecture**



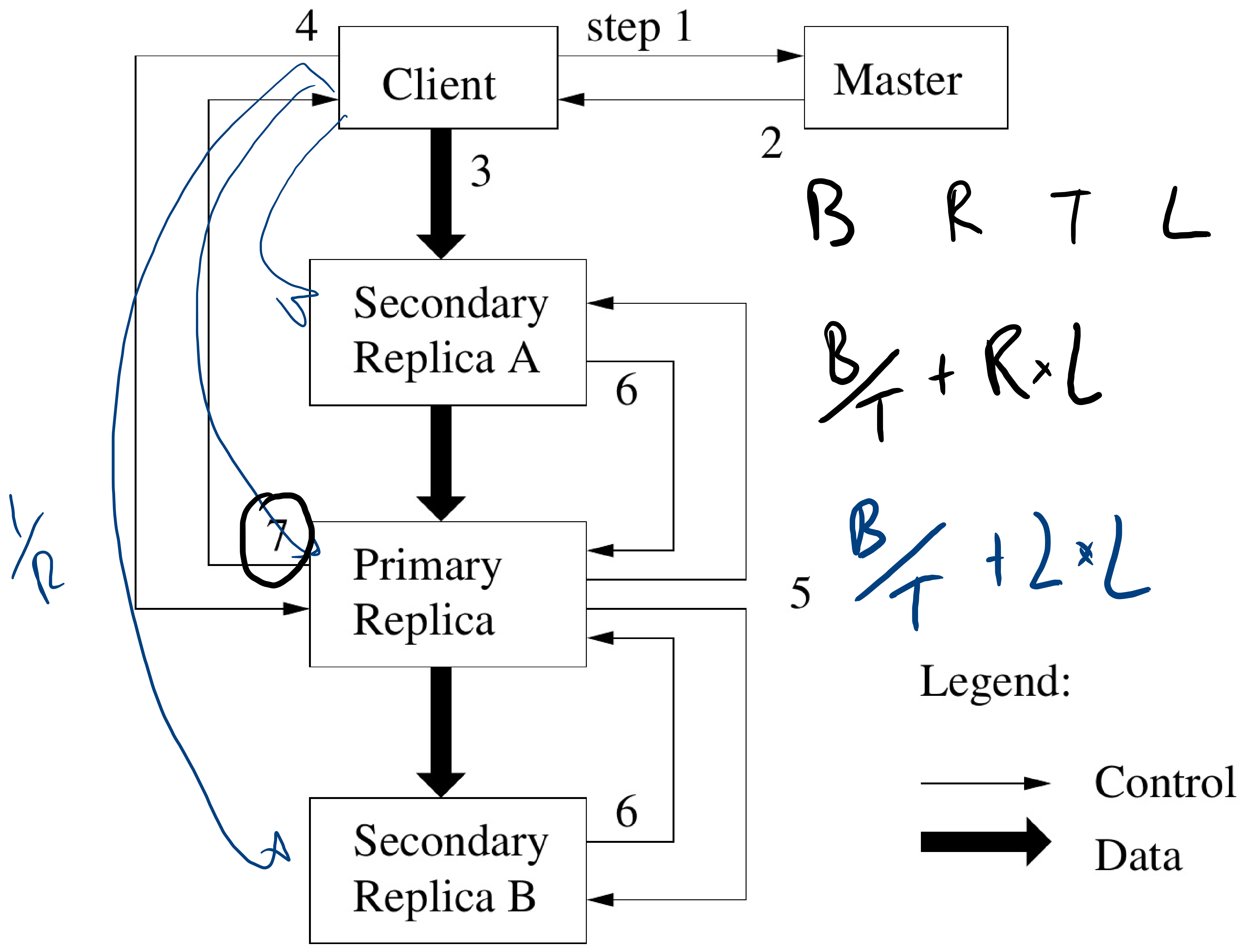


Figure 2: Write Control and Data Flow

$B \quad R \quad T \quad L$

$B/T + R \times L$

$B/T + 2 \times L$

$1/R$

full path → metadata, chunks, lock

$\underbrace{/\mathrm{d1}/\mathrm{d2}/\mathrm{d3}/}_{}\;.\;.\;.\;.$

Create $\underbrace{\underbrace{\underbrace{/\mathrm{home}}_{R}/\mathrm{user}}_{R}/\mathrm{foo}}_{W}$

locks

## Snapshots

$F_1 \rightarrow F_2$

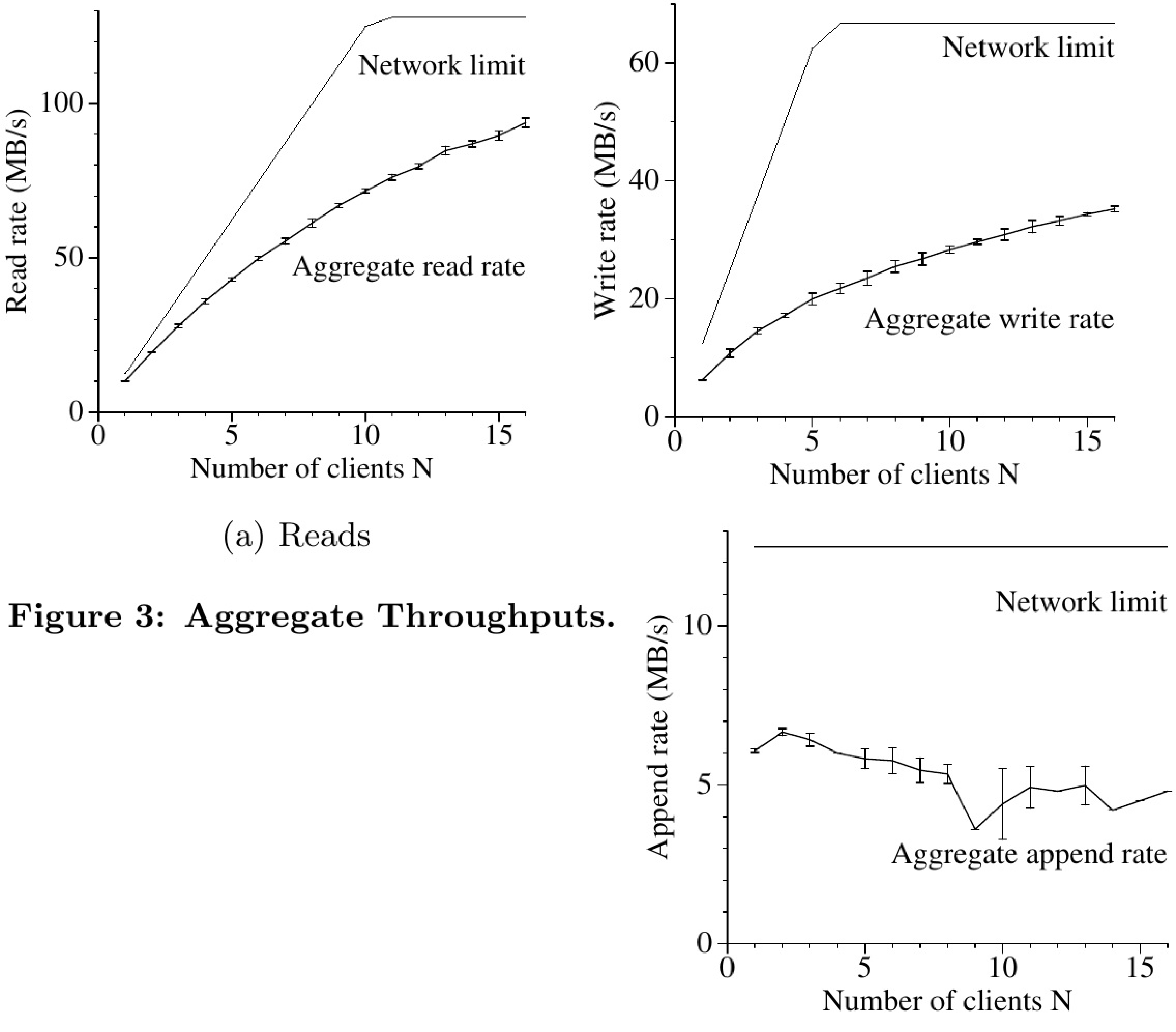
Revoke leases

Replicate filename mapping for $F_2$

Increase chunk ref counts.

## Copy on write

Tell chunk servers to make local copy

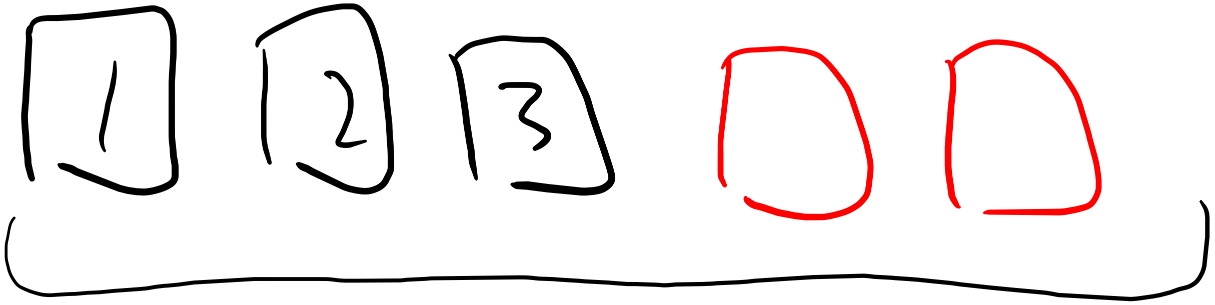


(a) Reads

(c) Record appends

**Figure 3: Aggregate Throughputs.**

incremental checksum



bottlenecks

Master

### 6.2.4 *Master Load*

Table 3 also shows that the rate of operations sent to the master was around 200 to 500 operations per second. The master can easily keep up with this rate, and therefore is not a bottleneck for these workloads.

In an earlier version of GFS, the master was occasionally a bottleneck for some workloads. It spent most of its time sequentially scanning through large directories (which contained hundreds of thousands of files) looking for particular files. We have since changed the master data structures to allow efficient binary searches through the namespace. It can now easily support many thousands of file accesses per second. If necessary, we could speed it up further by placing name lookup caches in front of the namespace data structures.