Google File System

goals

- monitoring, fault tolerance, auto-recovery (thousands of low-cost machines)
- focus on multi-GB files
- optimised for sequential reads and append writes (websites: seldom random writes & reads)
- handle *appends* efficiently
- co-design GFS and the applications

operations supported

classic operations

- create, read, write, delete, open, close

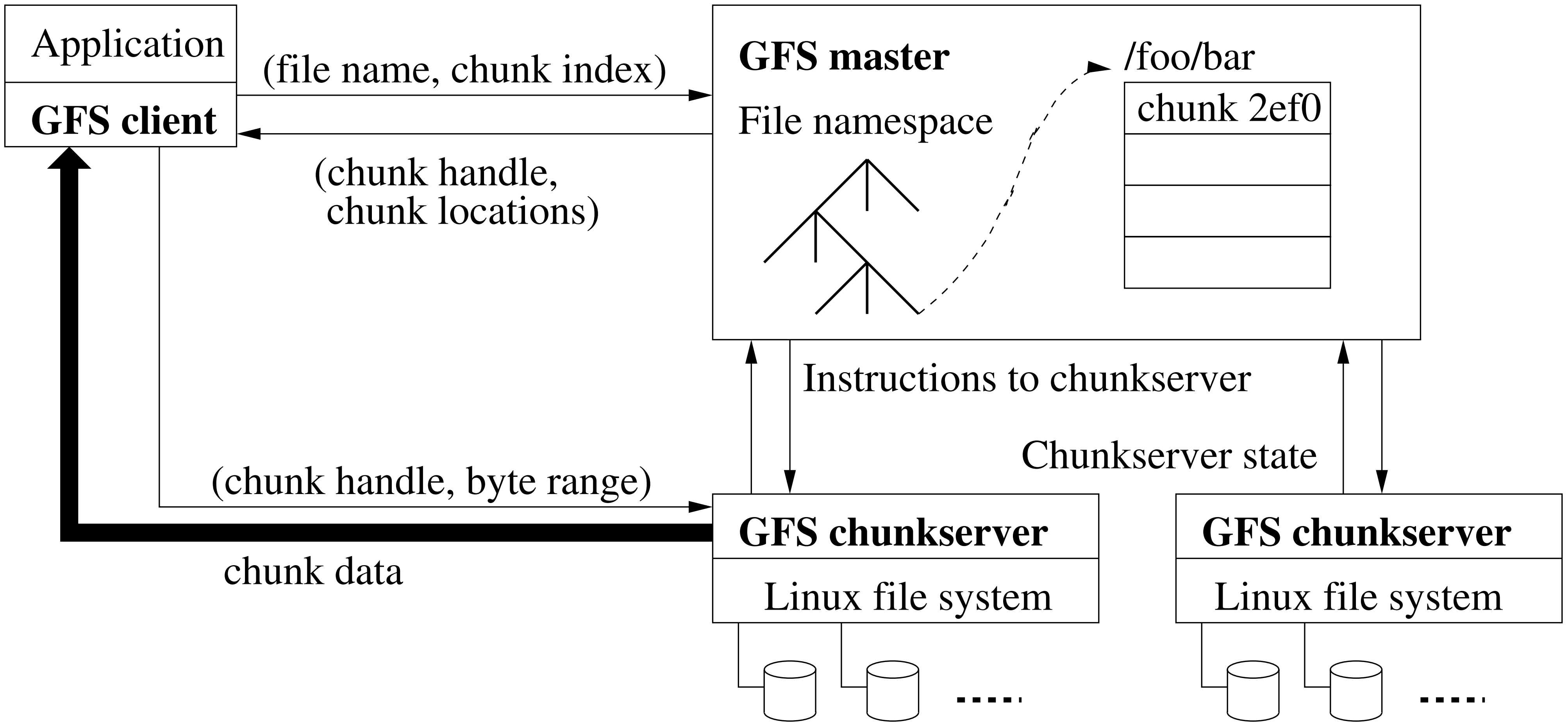
new operations

- snapshot—quick&low cost ‘picture’ of a file(dir)
- record append—multiple clients appending simultaneously, no sync required

terminology

- chunk—fixed-size piece of file
- chunk server—holds chunks
- master—coordinates chunk servers
- chunk handle—ID of a chunk (64 bit, globally unique)

cluster architecture



the master

- maintains all the metadata
- controls system-wide activities
    - collects chunks of a chunk server at startup (polls) and
        - generates in-memory mapping of files and chunk server pointers
    - chunk lease management (replication, (re)placement)
    - garbage collection of orphaned chunks
    - chunk migration
    - HeartBeat with chunk servers (collect state, check they’re ok)
- deals with *all* clients for metadata operations

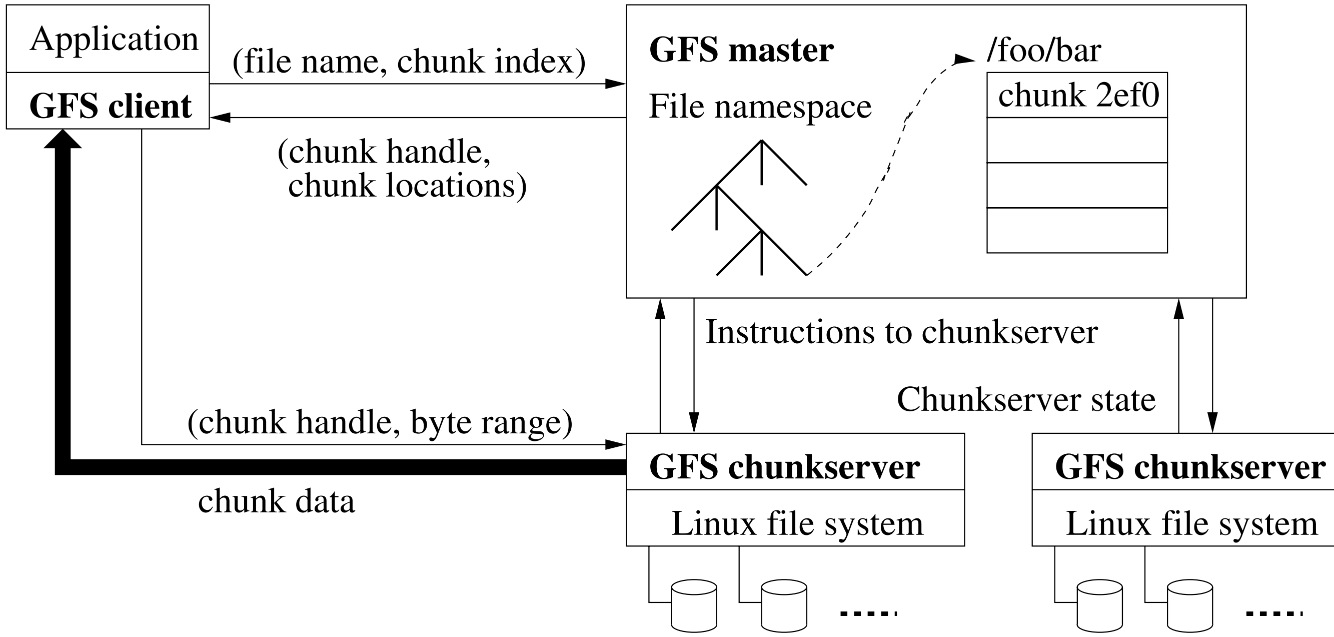
avoiding master bottleneck

clients

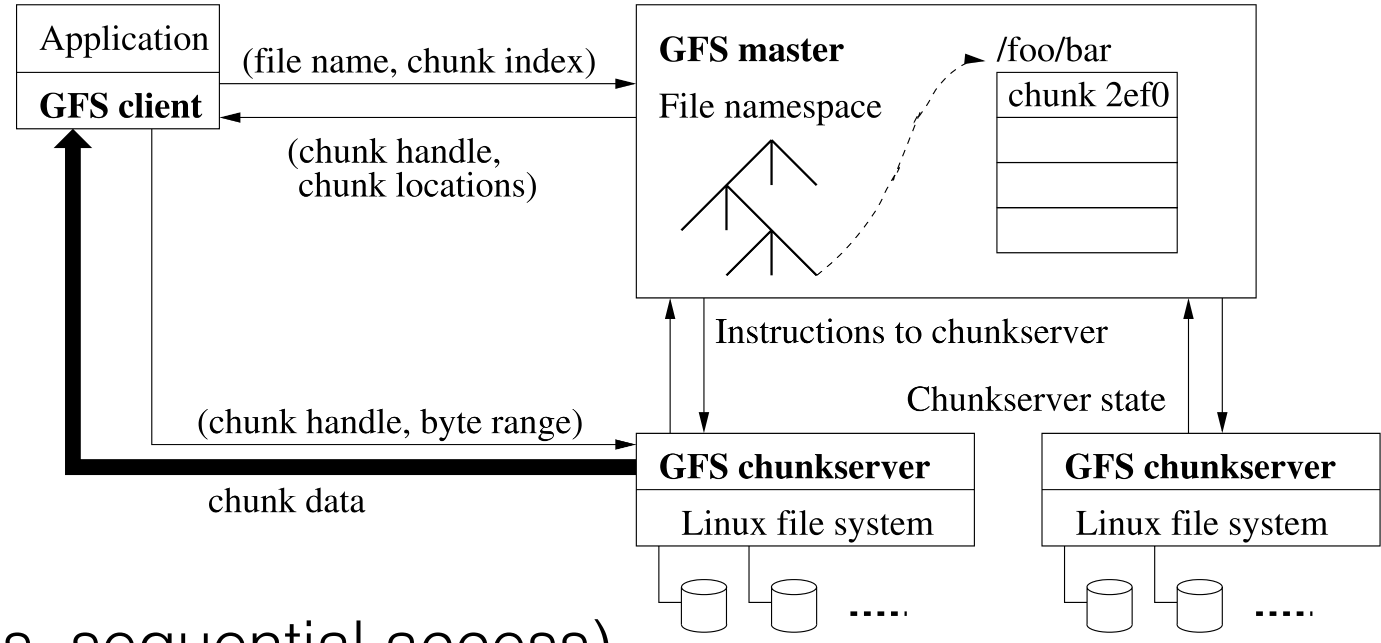
- get only ‘chunkserver pointers’ from master

- retrieve data directly from chunkservers (master just gives the directions to where…)

- cache the direction info for efficiency (no need to communicate with master for further reads of the same chunk)



chunks



properties in GFS:

- size = 64 MB; ID size = 64 bit
- plain linux file on server

advantages of 64MB chunks:

- reduce client-master interaction (large files, sequential access)
- reduce network overhead (successive ops on the same large chunk)
- reduce metadata size on master ==> in-memory metadata is possible

disadvantages of large chunks:

- internal fragmentation
- 1-chunk files turn chunk servers into hotspots (higher replication factor for small-files)

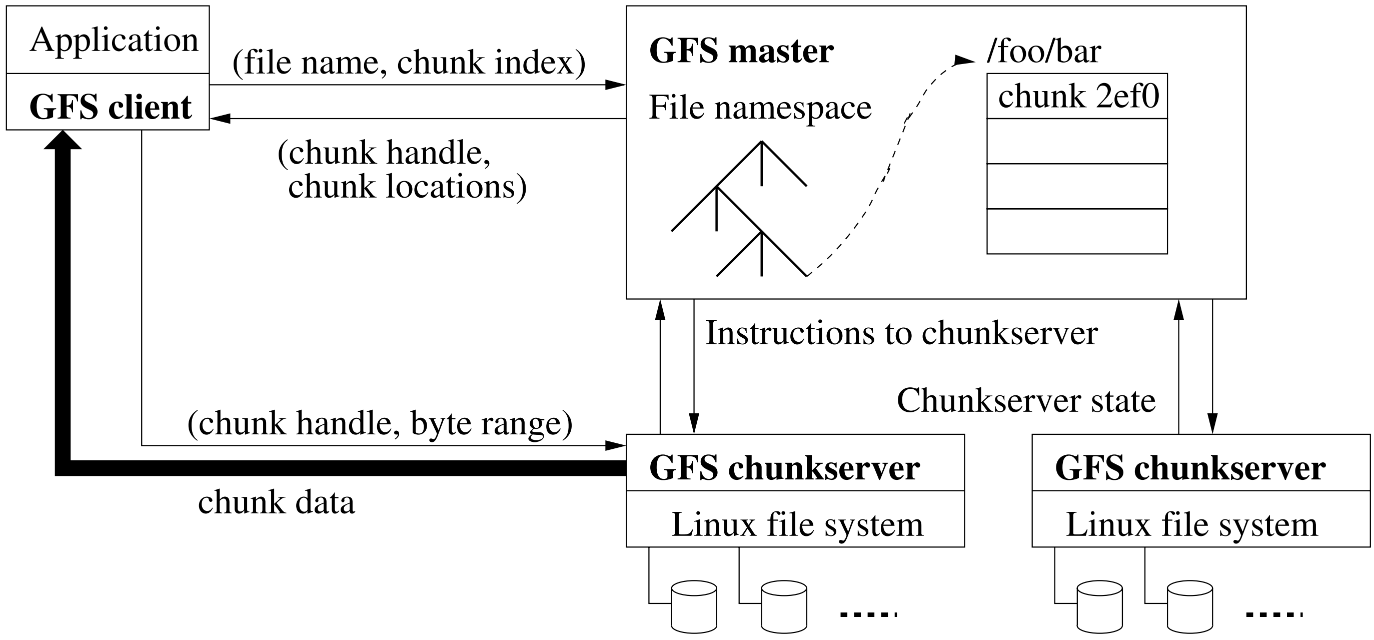
metadata

stuff kept in master's main memory only:

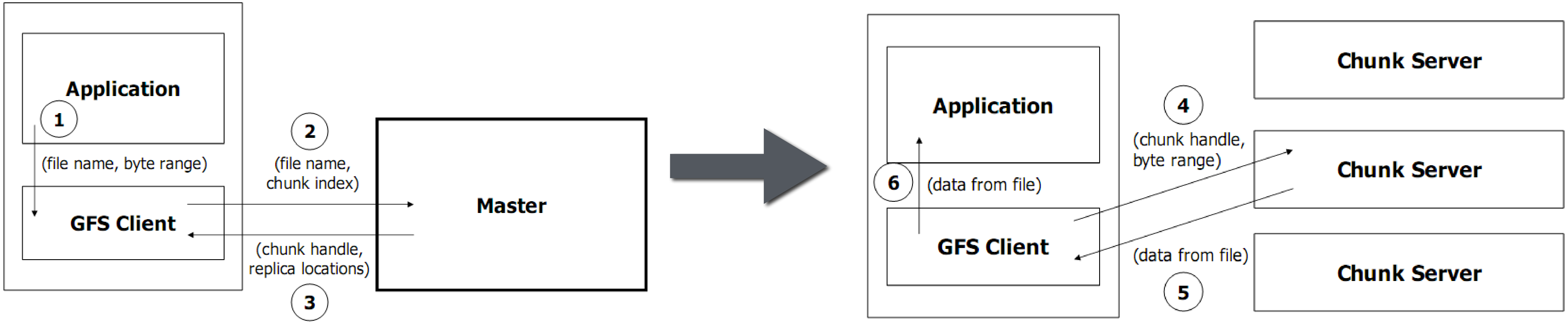
- namespace
- file <—> chunks mapping
- chunk location info

operation logs:

- stored reliably on master's disk
- replicated on multiple machines
- give logical timeline to operations on metadata
- necessary to re-build file-system state
- checkpoints to speed-up recovery



reading



Application

1

(file name, byte range)

GFS Client

2

(file name, chunk index)

Master

(chunk handle, replica locations)

3

Application

6

(data from file)

GFS Client

4

(chunk handle, byte range)

Chunk Server

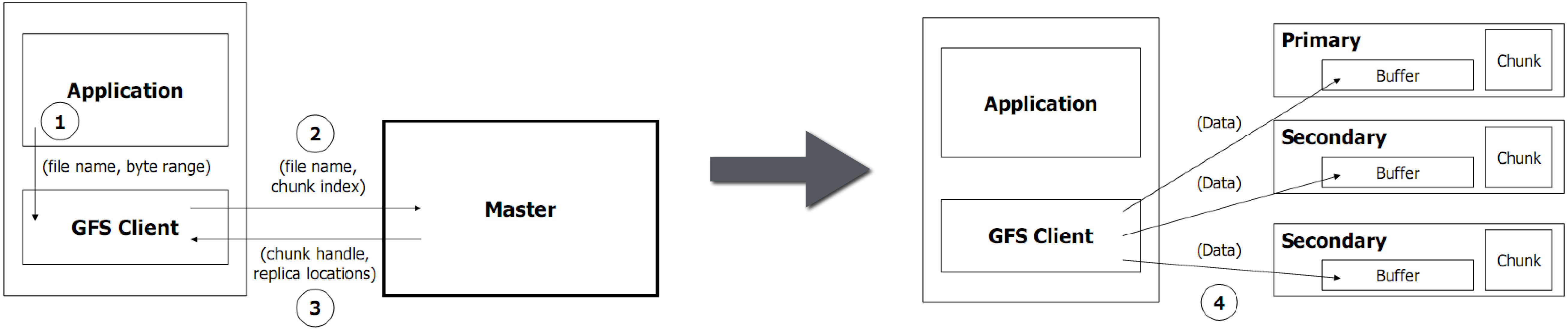
Chunk Server

Chunk Server

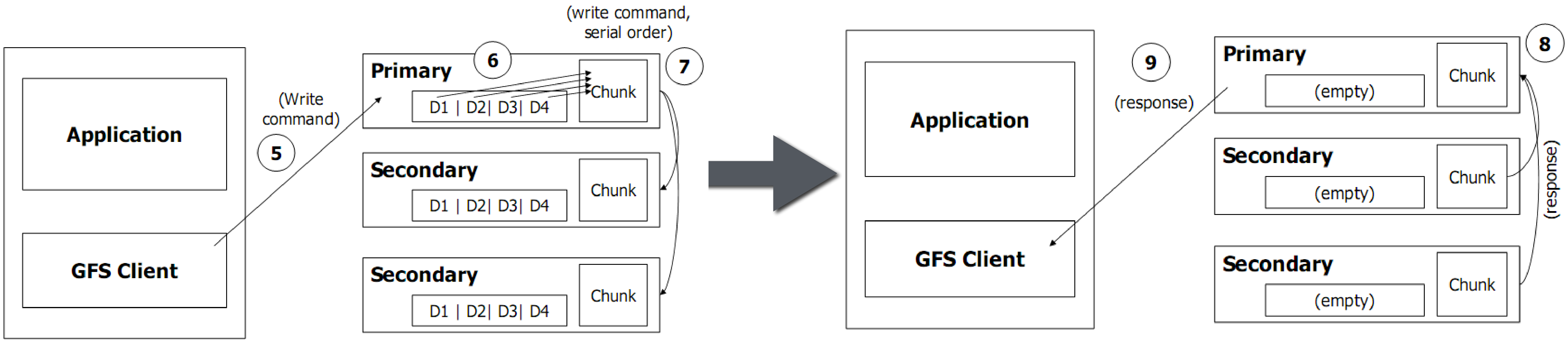
(data from file)

5

writing

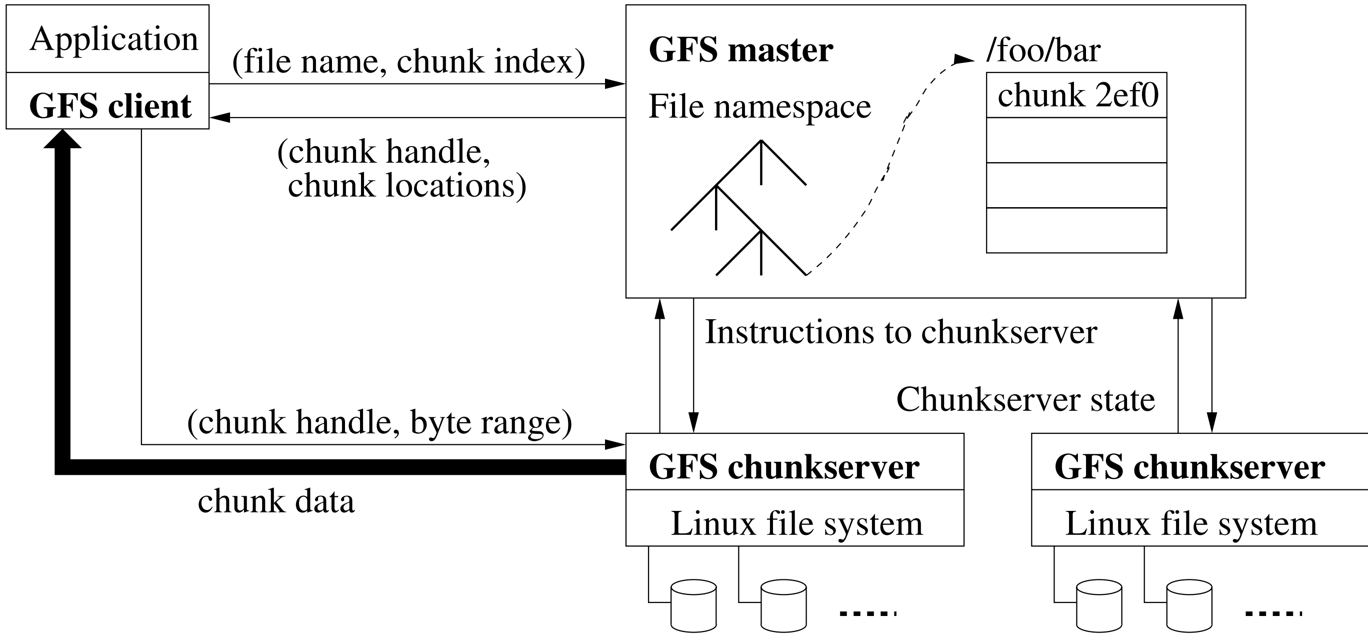

Application
1
(file name, byte range)
GFS Client
2
(file name,
chunk index)
Master
(chunk handle,
replica locations)
3
Application
GFS Client
(Data)
(Data)
(Data)
4
Primary
Buffer
Chunk
Secondary
Buffer
Chunk
Secondary
Buffer
Chunk

writing


Application
GFS Client
(Write command)
5
Primary
6
D1 | D2| D3| D4
Chunk
(write command, serial order)
7
Secondary
D1 | D2| D3| D4
Chunk
Secondary
D1 | D2| D3| D4
Chunk
Application
GFS Client
9
(response)
Primary
(empty)
Chunk
8
Secondary
(empty)
Chunk
Secondary
(empty)
Chunk
(response)

consistency

- atomic namespace mutations (master & op log)
- file region states: (un)defined, (un)consistent
- data mutations: writes or record appends

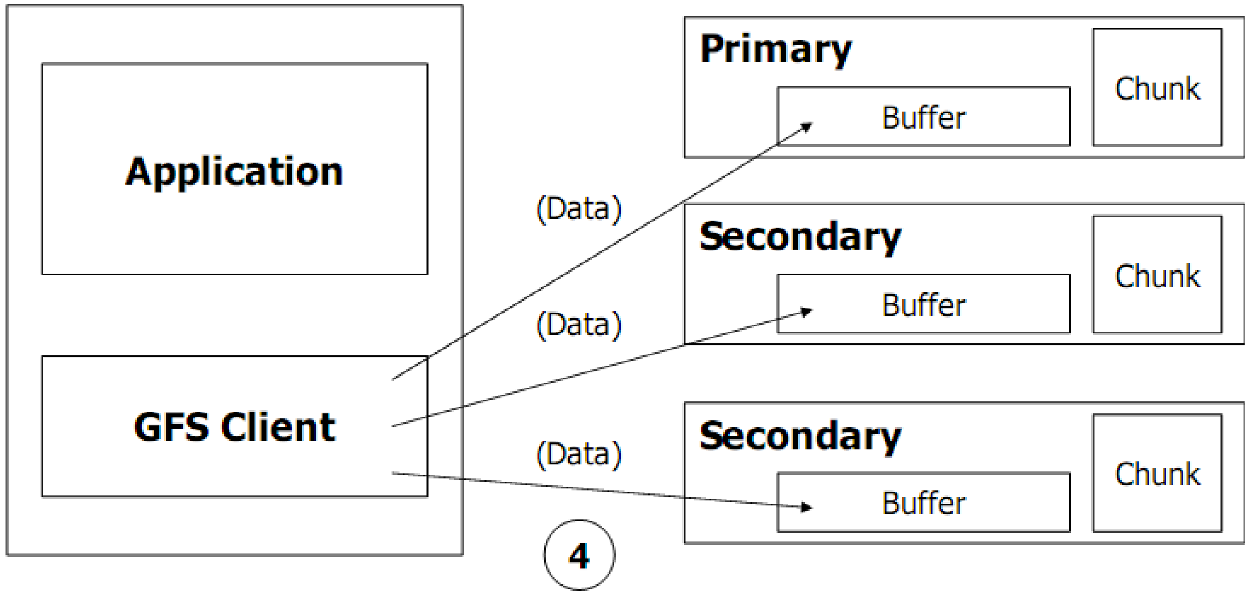


applications & consistency:

- write-on-create & append-only
- checkpointing (incremental on defined states)

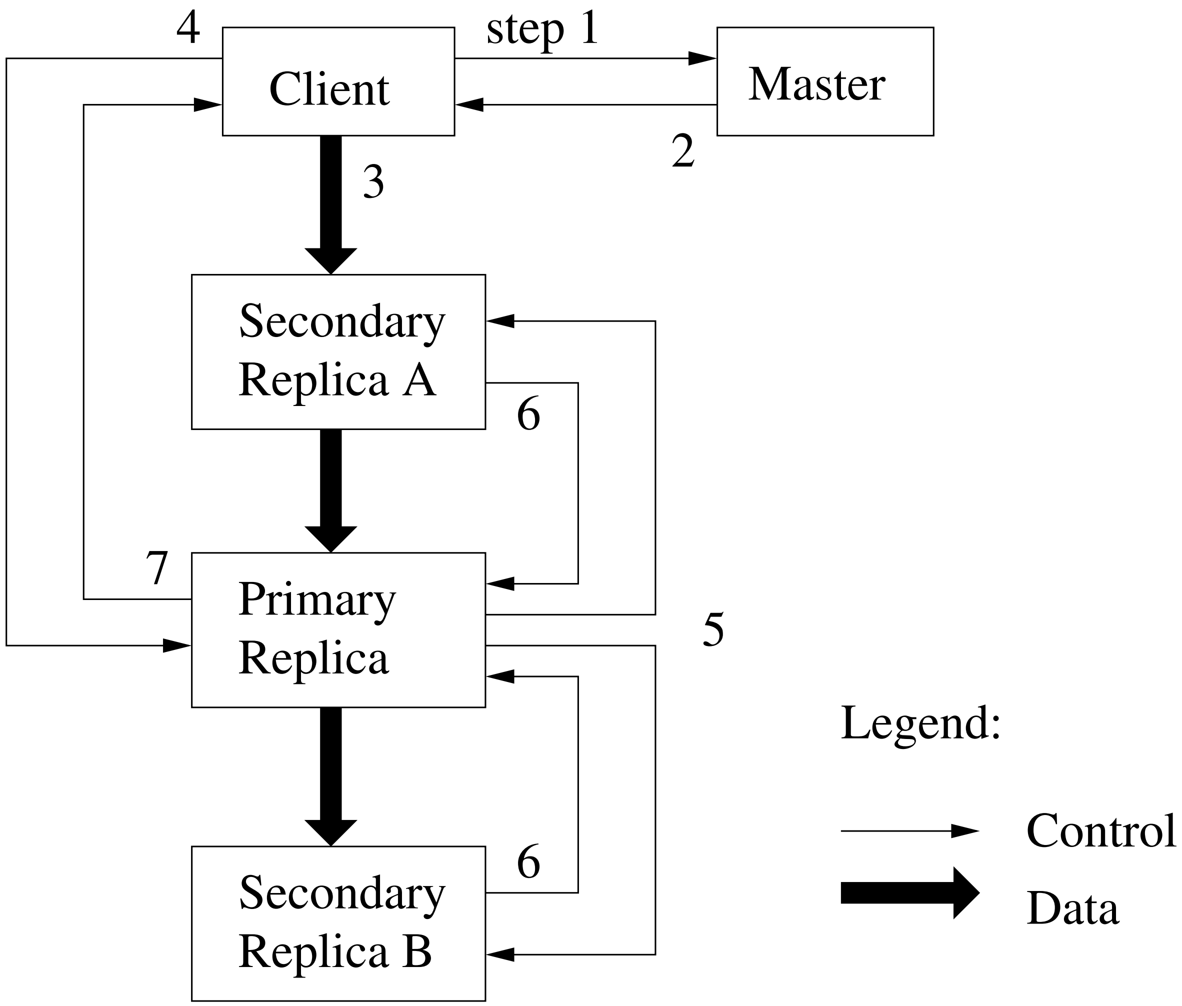
mutations & leases

- mutation performed on all replicas
- primary:
  - deciding mutation order
  - selected by the master (chunk lease)
- lease lasts 60 secs
- can be extended on request
- lease-messages piggybacked on heartbeat messages



write data flow

1. client asks for primary & replicas
2. master sends info, client caches it
3. client sends data to *all* replicas
4. client sends *write request* to primary
5. primary forwards mutation order + write request to secondaries
6. ack to primary about write complete
7. primary ack to client (errors too)



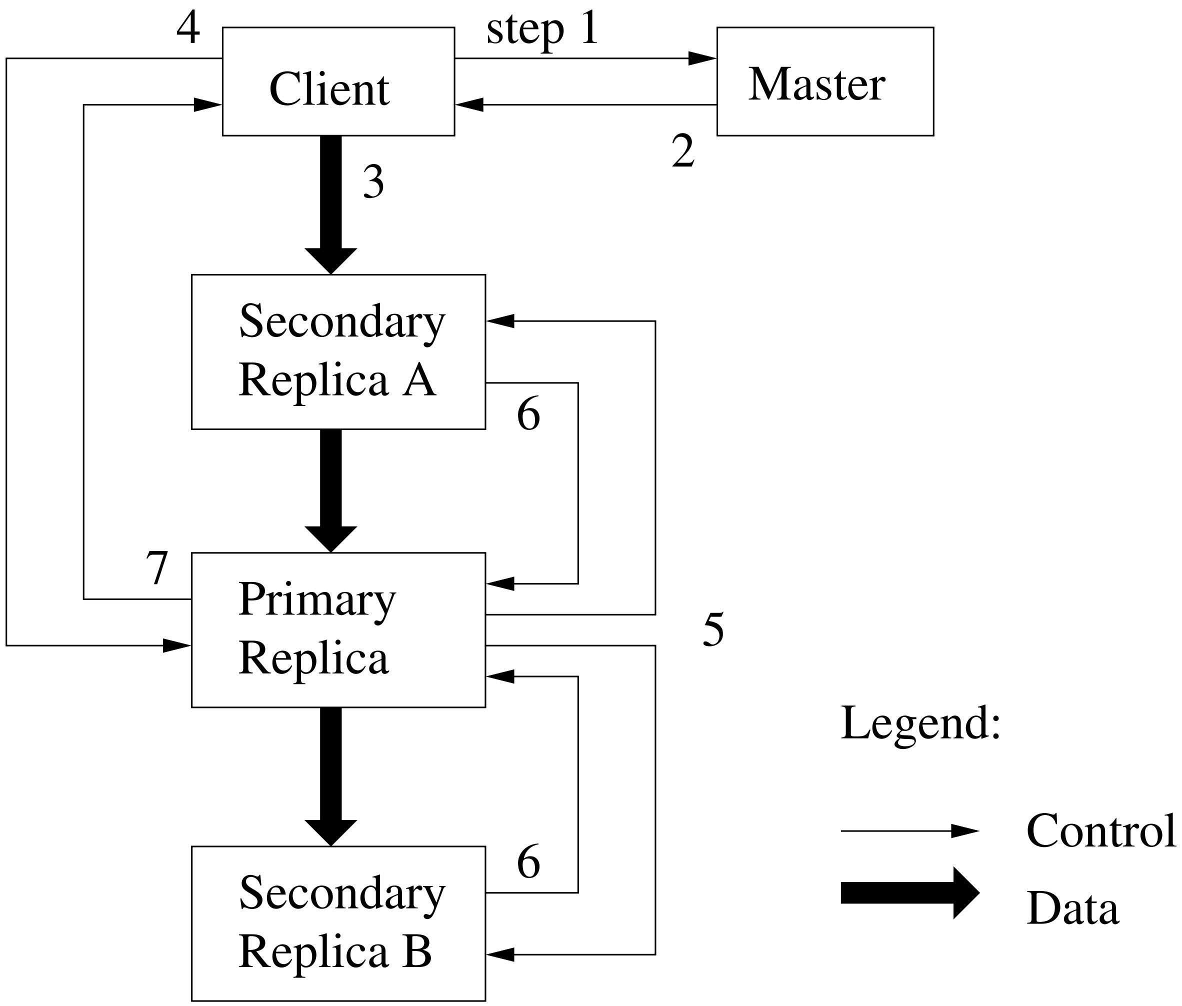
data vs control flow

two different flows for efficiency:

- control (through primary)
- data (chain of chunkservers)

data flow:

- next hop is the closest chunkserver
- closeness determined by IP
- data forwarded as they come

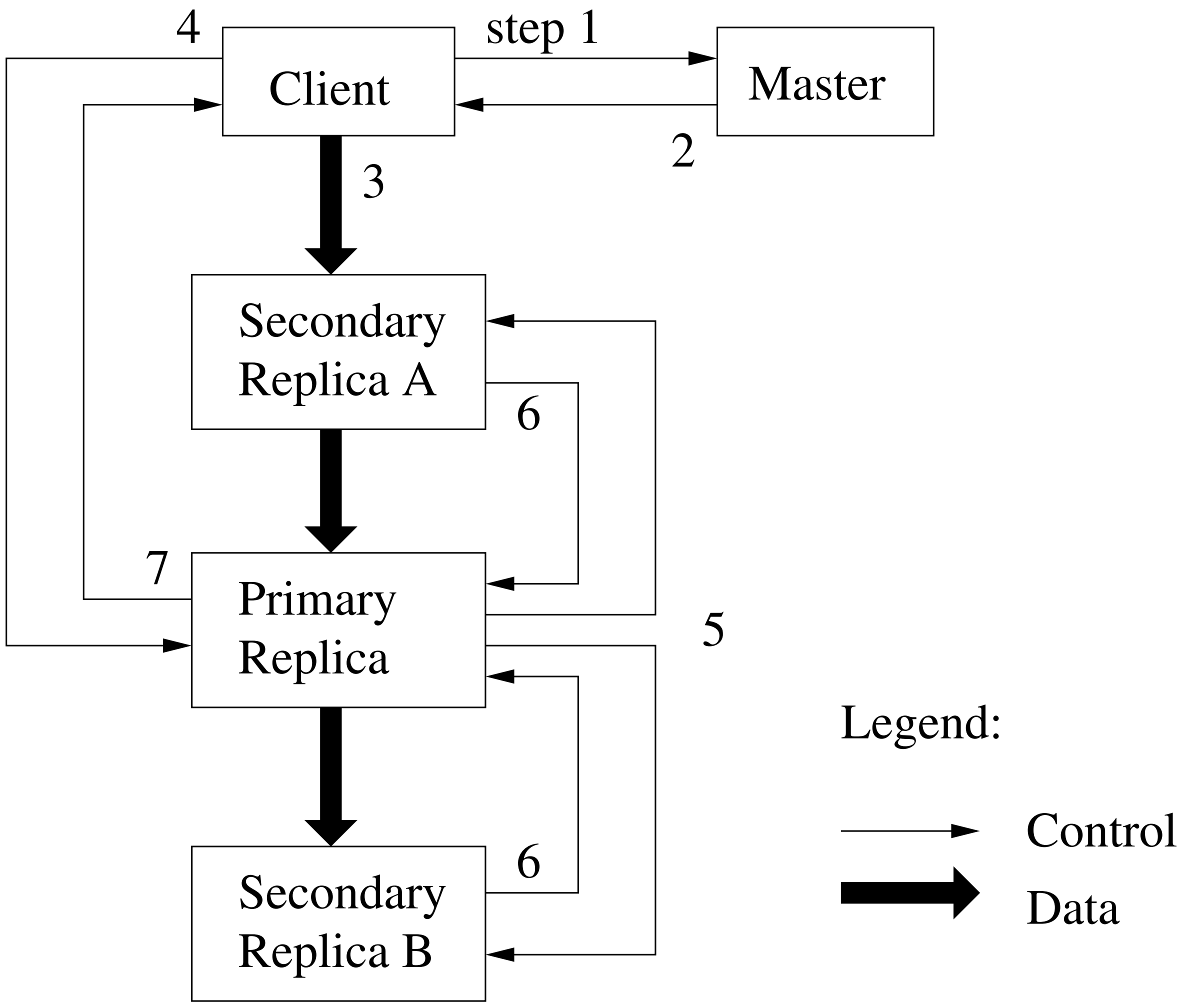


record appends

- the client specifies data only (no offset)
- GFS picks offset and sends it to client

data flow:

1. client sends data to replicas of last chunk
2. client sends request to primary
3. primary checks chunk availability & space (pads it if necessary, tells client to try with next chunk)
4. primary writes, tells replicas to write to the same offset, & acks the client



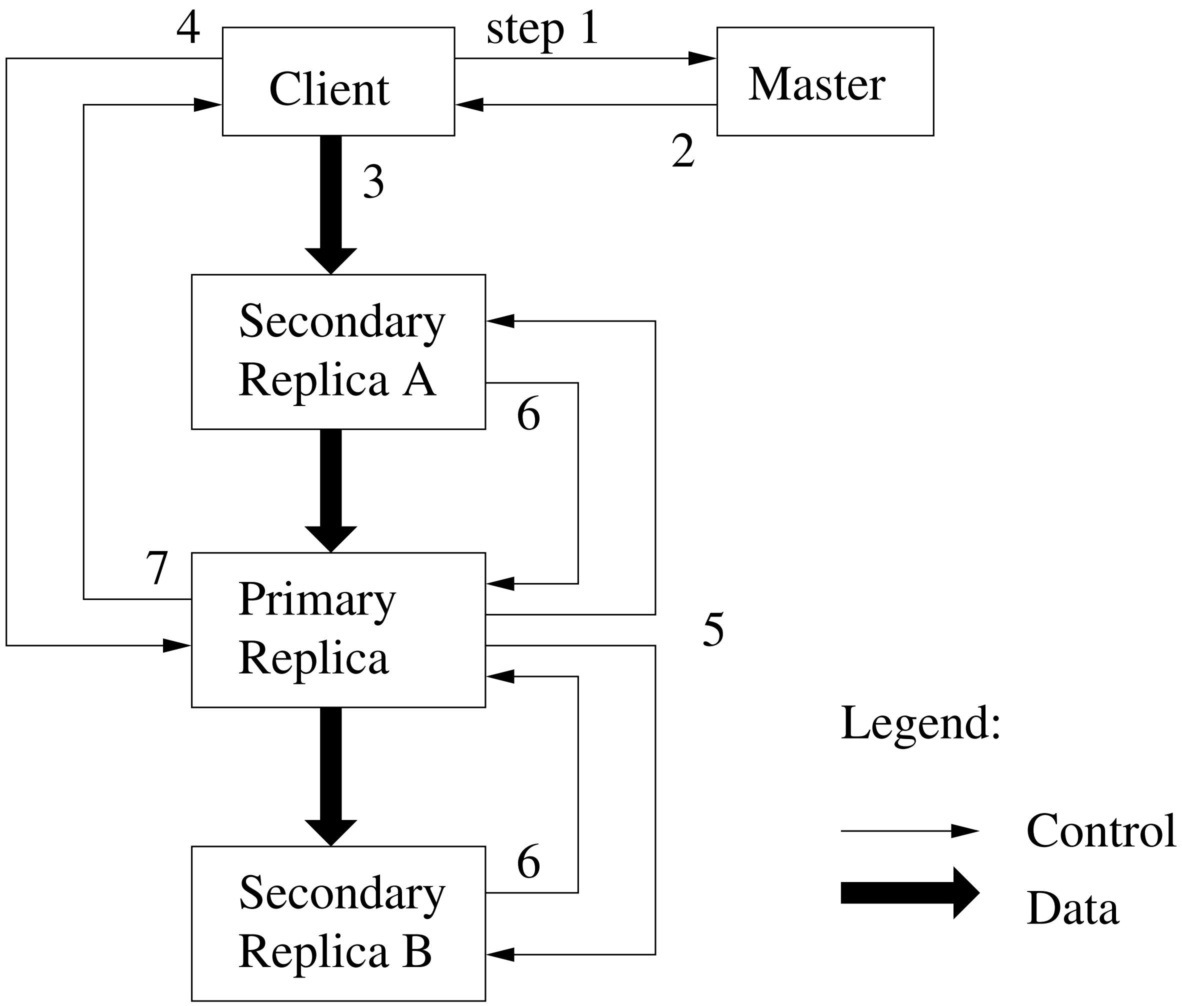
if record append fails

A. client retries

B. replicas could have different data in the chunk

C. client is ack-ed ok only if record written at same offset everywhere

D. the above regions are defined (and consistent)



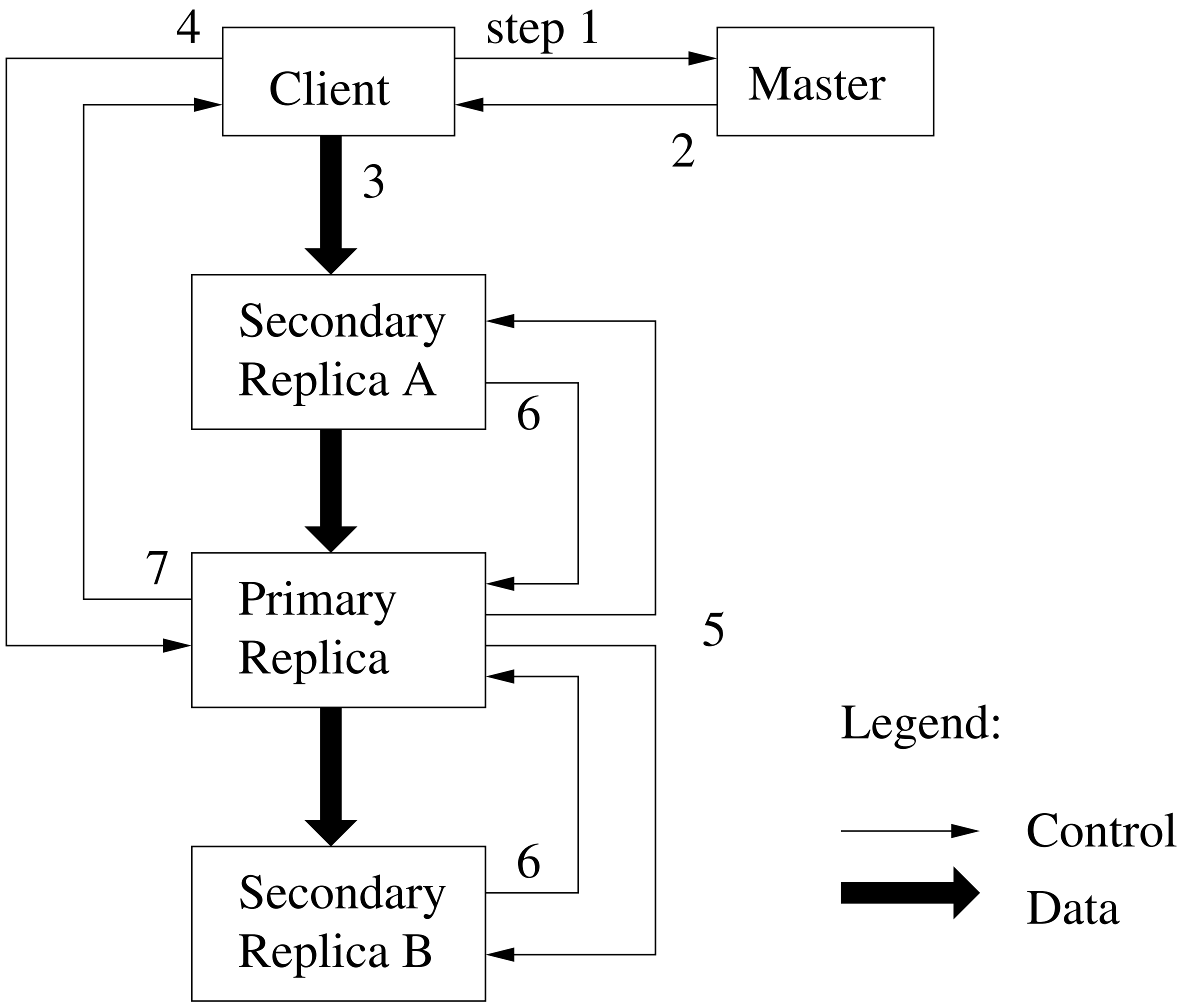
snapshot (of a file)

master duties

- duplicates the in-memory metadata for the file (reference count is now >1)
- revokes leases on chunk (why?)
- logs the op

actual copying (of chunk C)

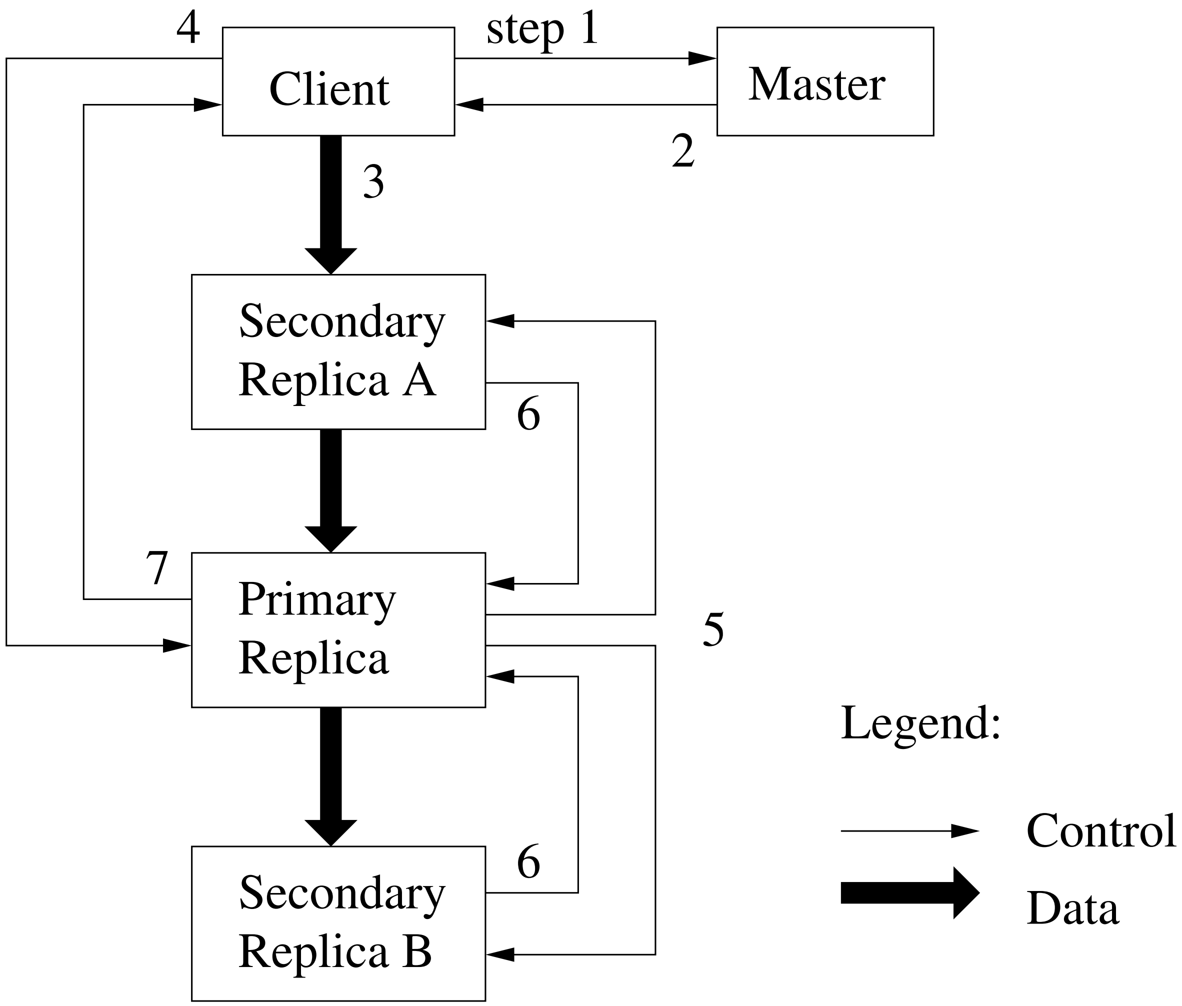
- when client needs to write on C
- reference count indicator
- triggers creation of copy C' (new chunk handle) on all replicas
- client will modify chunk C'



snapshot (of a file)

properties:

- copy-on-write (optimizes snapshots & disk usage)
- copy-on-same-replica (optimizes network bandwidth usage)



namespace & locking

- master performs many operations possibly in parallel
- namespace locks used to operate on files
- namespace tables: paths to metadata
- prefix compression (why?)
- every node read/write lock
- e.g. to deal with /d1/d2/…/dn/**leaf** will:
  - read-lock /**d1**, /d1/**d2**, …, /d1/../**dn**
  - read/write lock /d1/d2/…/dn/**leaf**

namespace & locking

properties:

- no directories concept, only files
- read-lock on dir name is sufficient for writing file
- concurrent mutations within same dir
  - read lock on dir name (prevents dir delete, snapshot, renamed)
  - write lock on file name
- locks acquired in consistent total order to prevent deadlock—level in the path & lexicograph

(re)placement

- distribute replicas over machines
- distribute replicas over racks
- new replicas on under-utilised chunk servers (equalize disk utilization)
- limit number of recent creations for a chunkserver
- replicate when nr of missing replicas is big (2 is better than 1)
- give priority to live files

deleting

- file renamed by master (name changes including delete timestamp)
- within 3 days can go back to normal
- after 3 days metadata is actually deleted
- orphan chunks (not reachable from files) are handled later on
    - heartbeat messages include list of chunk IDs
    - master sends back list of orphan chunks (not pointed by files in in-memory metadata)

fault-tolerance

high availability

- 3-way replication of chunks
- op logs: master + servers restartable in few secs (fast recovery)
- shadow masters

integrity

- checksum every 64K block

Thank you!