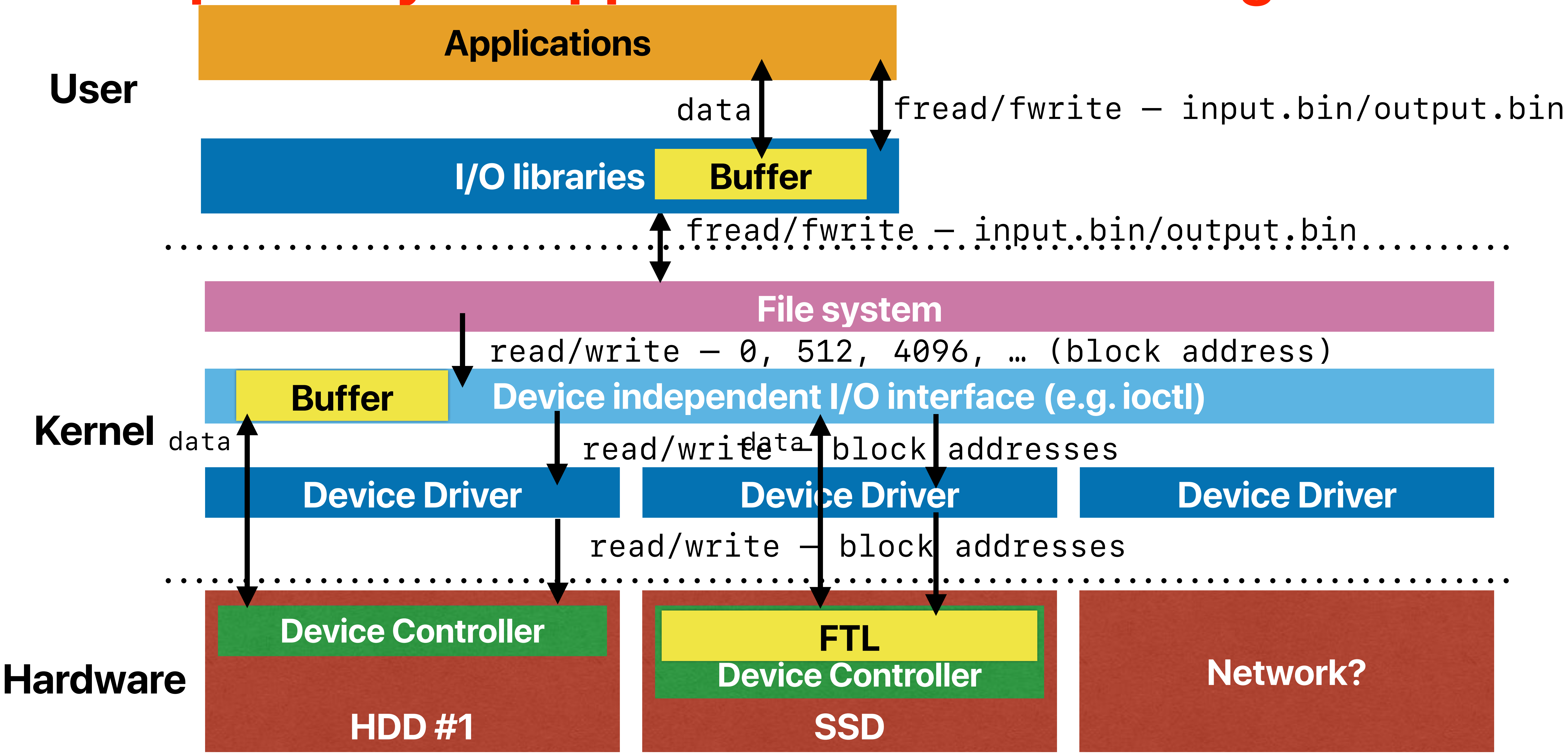
File systems over the network

Hung-Wei Tseng

Recap: How your application reaches storage device



User

Applications

data

fread/fwrite — input.bin/output.bin

I/O libraries

Buffer

fread/fwrite — input.bin/output.bin

Kernel

File system

read/write — 0, 512, 4096, … (block address)

Buffer

Device independent I/O interface (e.g. ioctl)

data

read/write — block addresses

data

Device Driver

Device Driver

Device Driver

read/write — block addresses

Hardware

Device Controller

HDD #1

FTL

Device Controller

SSD

Network?

Recap: File systems on a computer

- Unix File System
  - Hierarchical directory structure
  - File — metadata (inode) + data
  - Everything is files
- BSD Fast File System — optimize for reads
  - Cylinder group — Layout data carefully with device characteristics, replicated metadata
  - Larger block size & fragments to fix the drawback
  - A few other new features
- Sprite Log-structured File System — optimize for small random writes
  - Computers cache a lot — reads are no more the dominating traffic
  - Aggregates small writes into large sequential writes to the disk
  - Invalidate older copies to support recovery

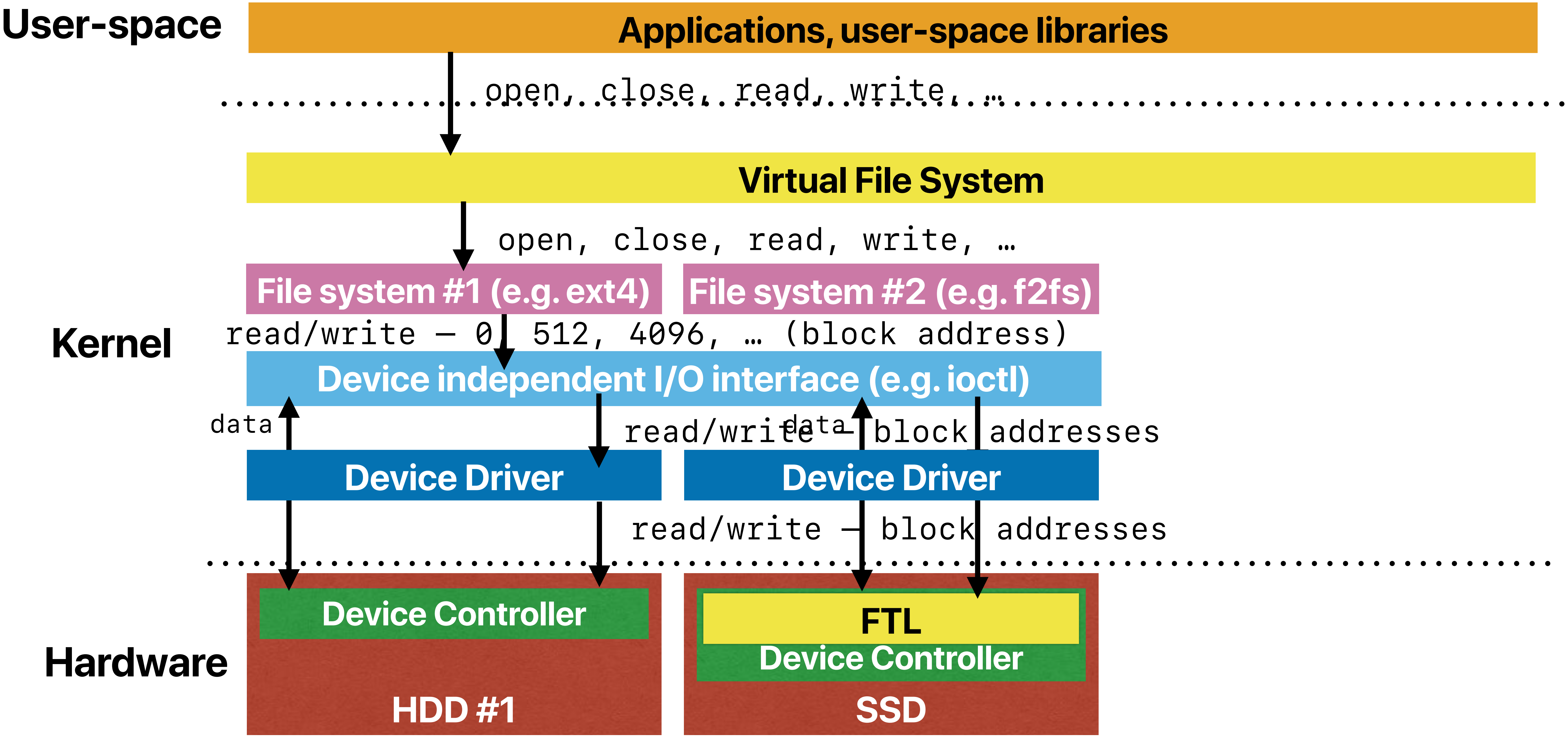
Recap: Extent file systems — ext2, ext3, ext4

- Basically optimizations over FFS + Extent + Journaling (write-ahead logs)
- Extent — consecutive disk blocks
- A file in ext file systems — a list of extents
- Journal
  - Write-ahead logs — performs writes as in LFS
  - Apply the log to the target location when appropriate
- Block group
  - Modern H.D.Ds do not have the concept of "cylinders"
  - They label neighboring sectors with consecutive block addresses
  - Does not work for SSDs given the internal log-structured management of block addresses

Recap: flash SSDs, NVM-based SSDs

- Asymmetric read/write behavior/performance
- Wear-out faster than traditional magnetic disks
- Another layer of indirection is introduced
  - Intensify log-on-log issues
  - We need to revise the file system design

The introduction of virtual file system interface



User-space

Applications, user-space libraries

`open, close, read, write, …`

Virtual File System

`open, close, read, write, …`

Kernel

File system #1 (e.g. ext4)

File system #2 (e.g. f2fs)

`read/write — 0, 512, 4096, … (block address)`

Device independent I/O interface (e.g. ioctl)

`data`

`read/write — block addresses`

`data`

Device Driver

Device Driver

`read/write — block addresses`

Hardware

Device Controller

HDD #1

FTL

Device Controller

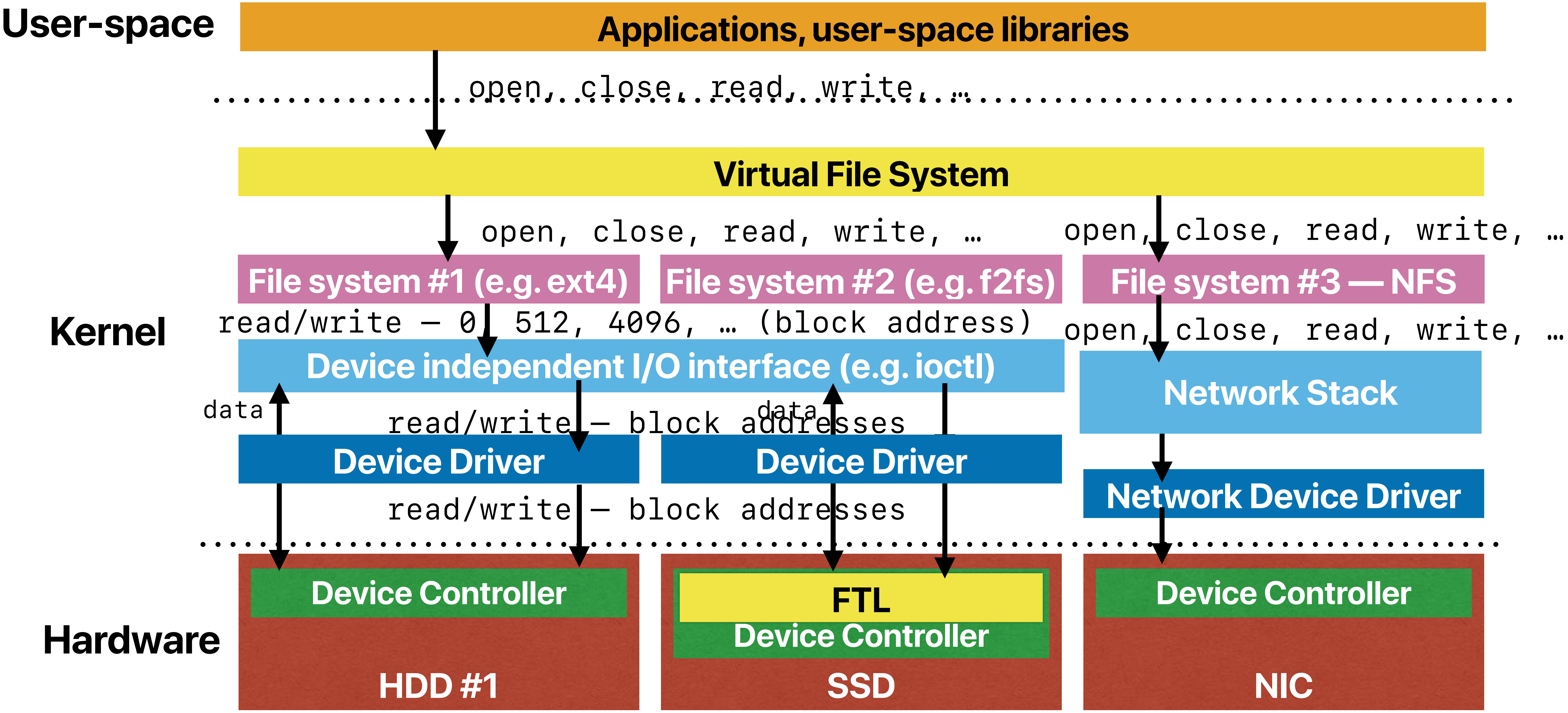
SSD

Outline

- NFS
- Google file system

Network File System

The introduction of virtual file system interface



User-space

Applications, user-space libraries

`open, close, read, write, …`

Virtual File System

`open, close, read, write, …`

`open, close, read, write, …`

File system #1 (e.g. ext4)

File system #2 (e.g. f2fs)

File system #3 — NFS

Kernel

`read/write — 0, 512, 4096, … (block address)`

`open, close, read, write, …`

Device independent I/O interface (e.g. ioctl)

Network Stack

`data`

`read/write — block addresses`

`data`

Device Driver

Device Driver

Network Device Driver

`read/write — block addresses`

Hardware

Device Controller

HDD #1

FTL

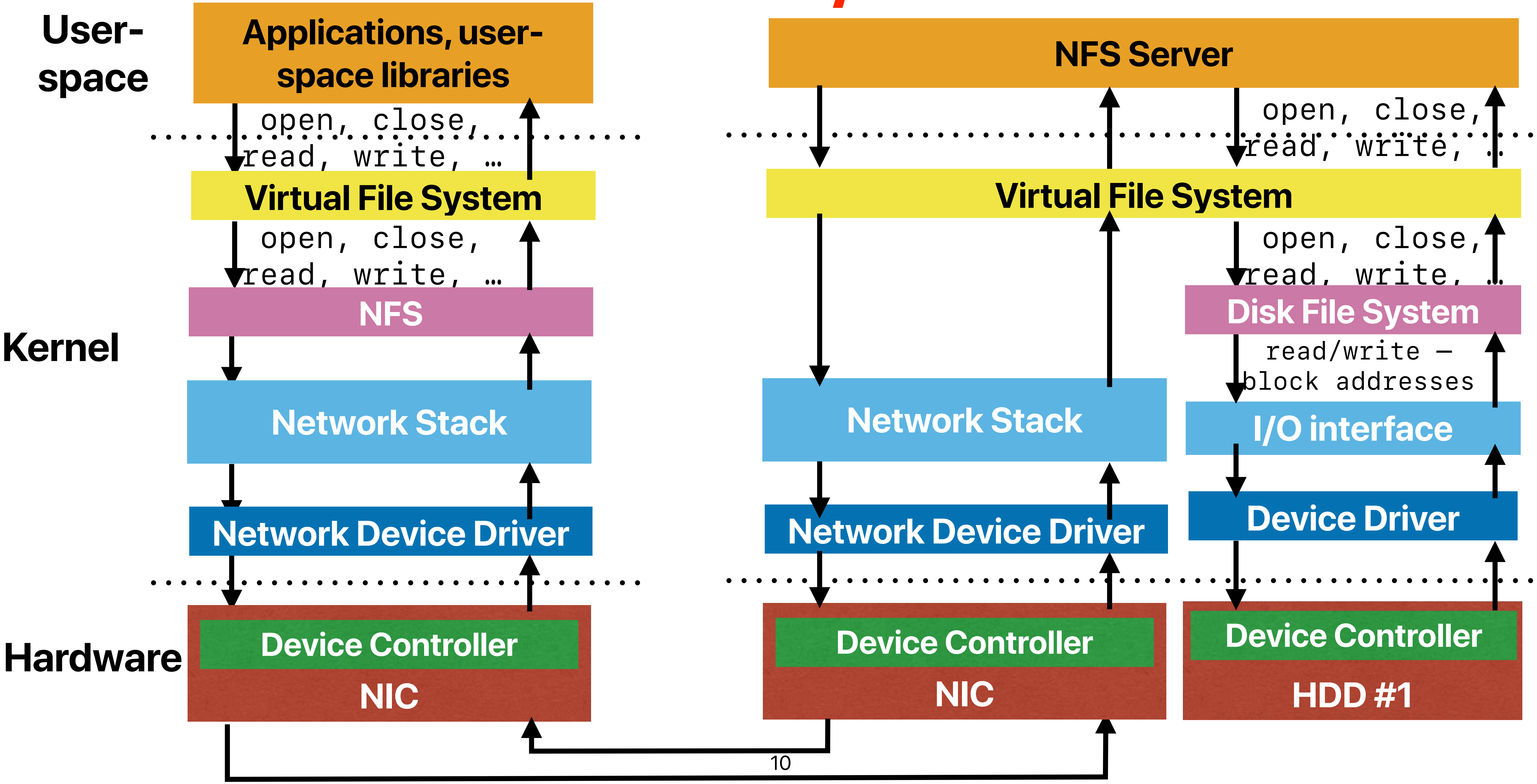
Device Controller

SSD

Device Controller

NIC

NFS Client/Server



User-space

Kernel

Hardware

Applications, user-space libraries

open, close, read, write, …

Virtual File System

open, close, read, write, …

NFS

Network Stack

Network Device Driver

Device Controller

NIC

NFS Server

open, close, read, write, …

Virtual File System

open, close, read, write, …

Disk File System

read/write – block addresses

I/O interface

Device Driver

Device Controller

HDD #1

Network Stack

Network Device Driver

Device Controller

NIC

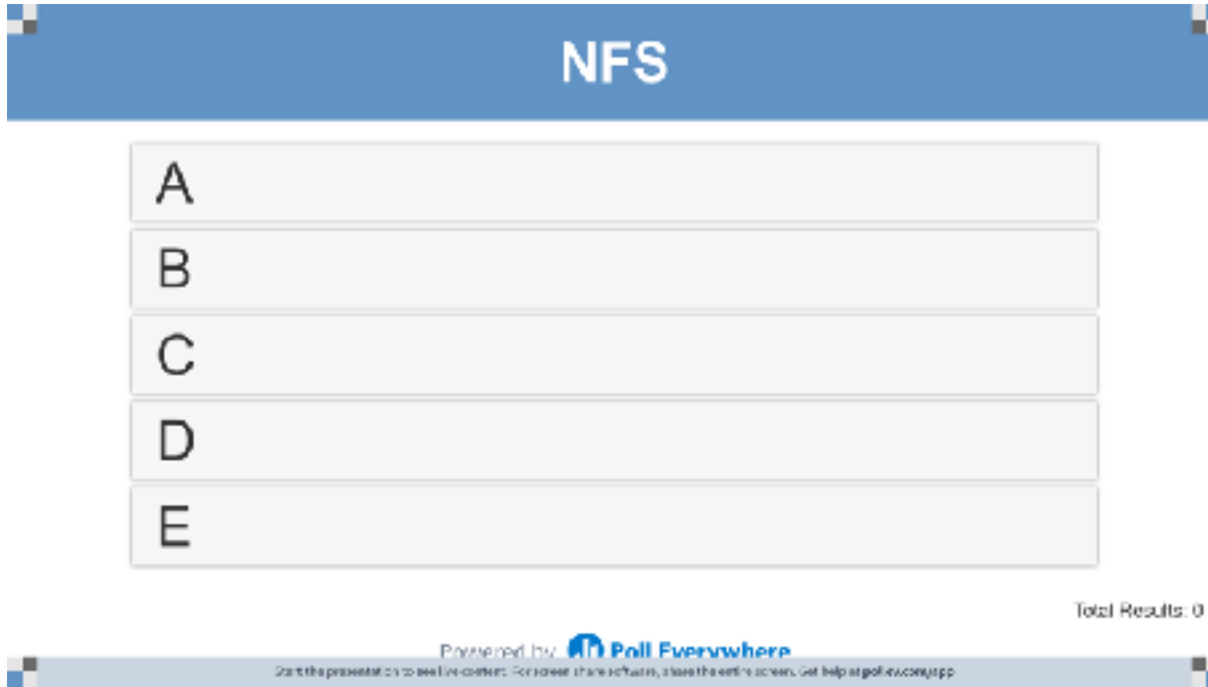
How does NFS handle a file?

- The client gives it’s file system a tuple to describe data
  - Volume: Identify which server contains the file — represented by the mount point in UNIX
  - inode: Where in the server
  - generation numer: version number of the file
- The local file system forwards the requests to the server
- The server response the client with file system attributes as local disks

https://www.pollev.com/hungweitseng close in 1:00

Number of network operations

- For a file /mnt/nfs/home/hungwei/foo.c , how many network sends/receives in total does NFS need to perform to fetch the actual file content in the worst case? (assume the file system is mounted to /mnt/nfs)
  A. 8
  B. 9
  C. 10
  D. 11
  E. 12



NFS

A

B

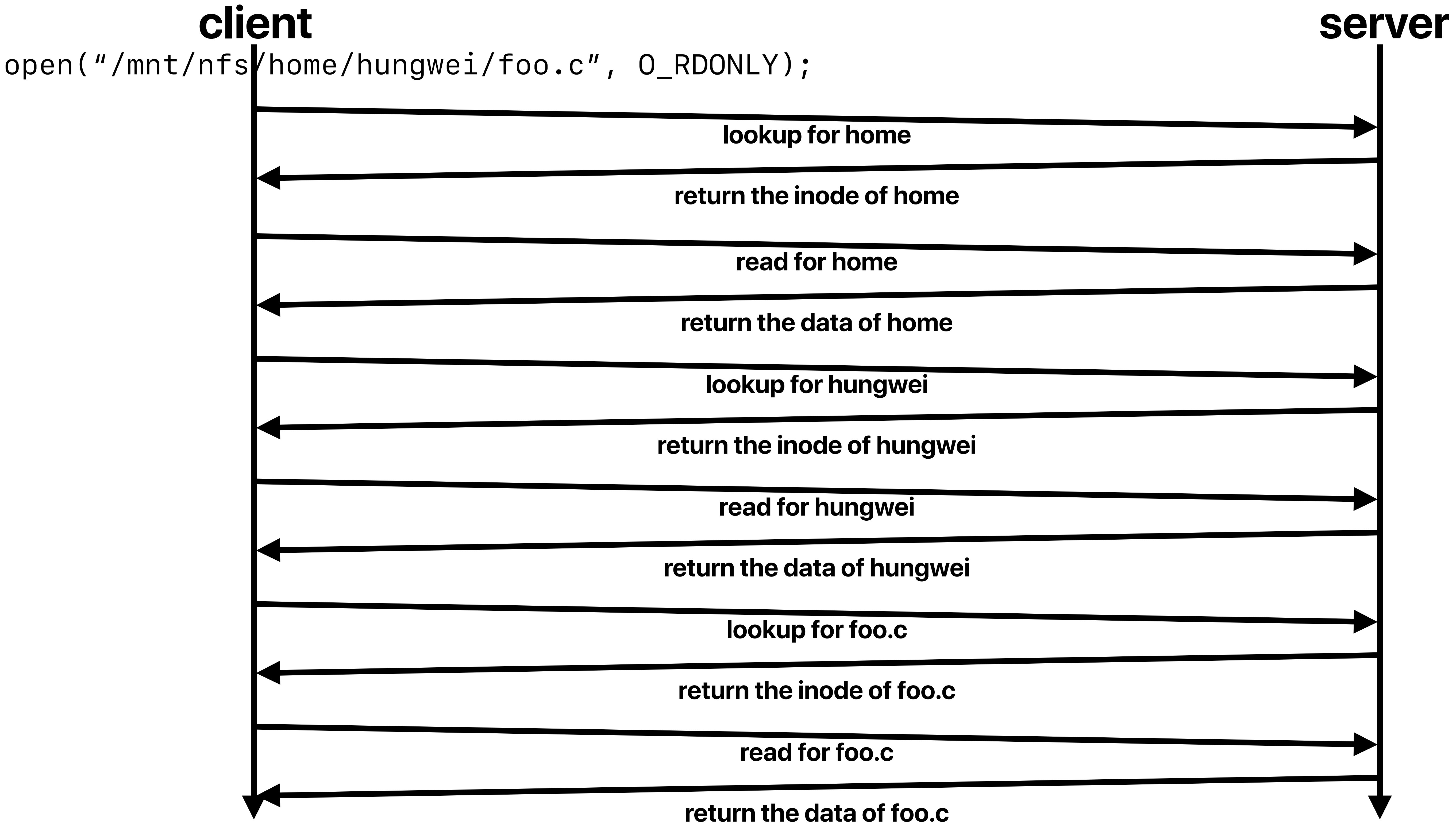
C

D

E

Total Results: 0

How open works with NFS



client

server

`open("/mnt/nfs/home/hungwei/foo.c", O_RDONLY);`

lookup for home

return the inode of home

read for home

return the data of home

lookup for hungwei

return the inode of hungwei

read for hungwei

return the data of hungwei

lookup for foo.c

return the inode of foo.c

read for foo.c

return the data of foo.c

Number of network operations

- For a file /mnt/nfs/home/hungwei/foo.c , how many network sends/receives in total does NFS need to perform to fetch the actual file content in the worst case? (assume the file system is mounted to /mnt/nfs)

  A. 8

  B. 9

  C. 10

  D. 11

  [✓ E. 12]

Caching

- NFS operations are expensive
  - Lots of network round-trips
  - NFS server is a user-space daemon
- With caching on the clients
  - Only the first reference needs network communication
  - Later requests can be satisfied in local memory

https://www.pollev.com/hungweitseng close in 1:00

Stateless NFS

- How many of the following statements fit the reason why NFS uses a stateless protocol, in which the protocol doesn't track any client state?

  ① Simplify the system design for recovery after server crashes

  ② Simplify the client design for recovery after client crashes

  ③ Easier to guarantee file consistency

  ④ Improve the network latency

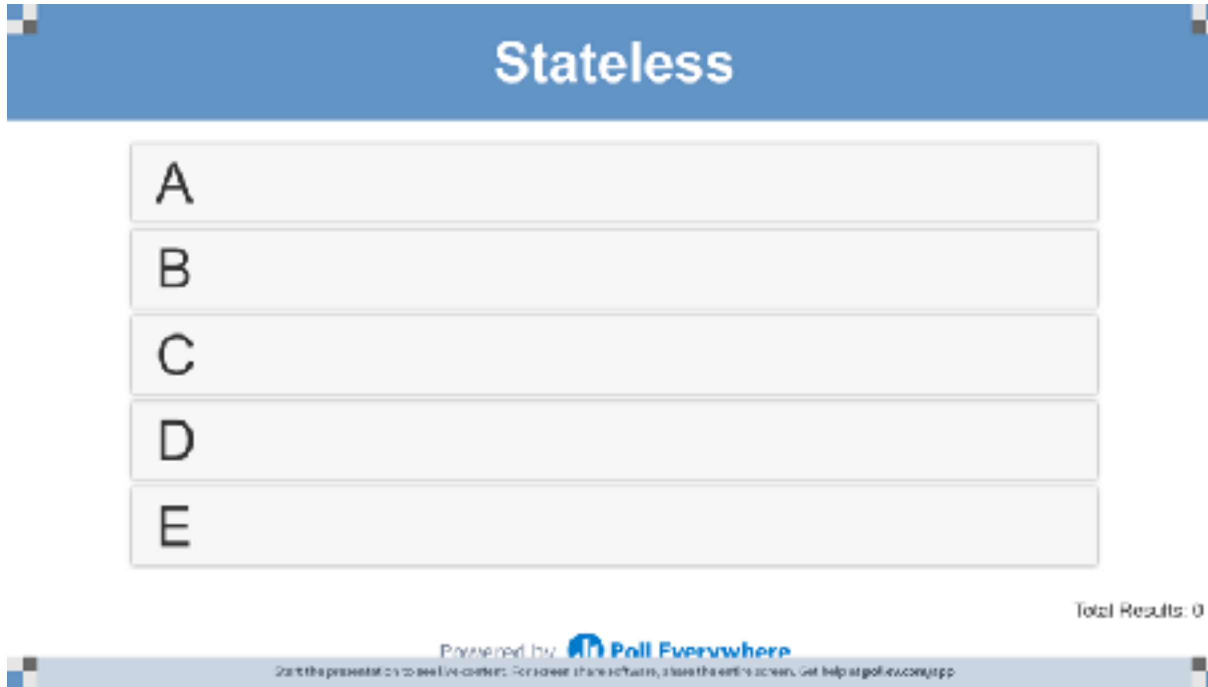
  A. 0

  B. 1

  C. 2

  D. 3

  E. 4



Stateless

A

B

C

D

E

Total Results: 0

Powered by Poll Everywhere

Stateless NFS

- How many of the following statements fit the reason why NFS uses a stateless protocol, in which the protocol doesn’t track any client state?
  - ✓ ① Simplify the system design for recovery after server crashes

    If using stateful protocol, FDs on all clients are lost
  - ✓ ② Simplify the client design for recovery after client crashes

    If using stateful protocol, server doesn’t know client crashes and consider the file is open still
  - ③ Easier to guarantee file consistency

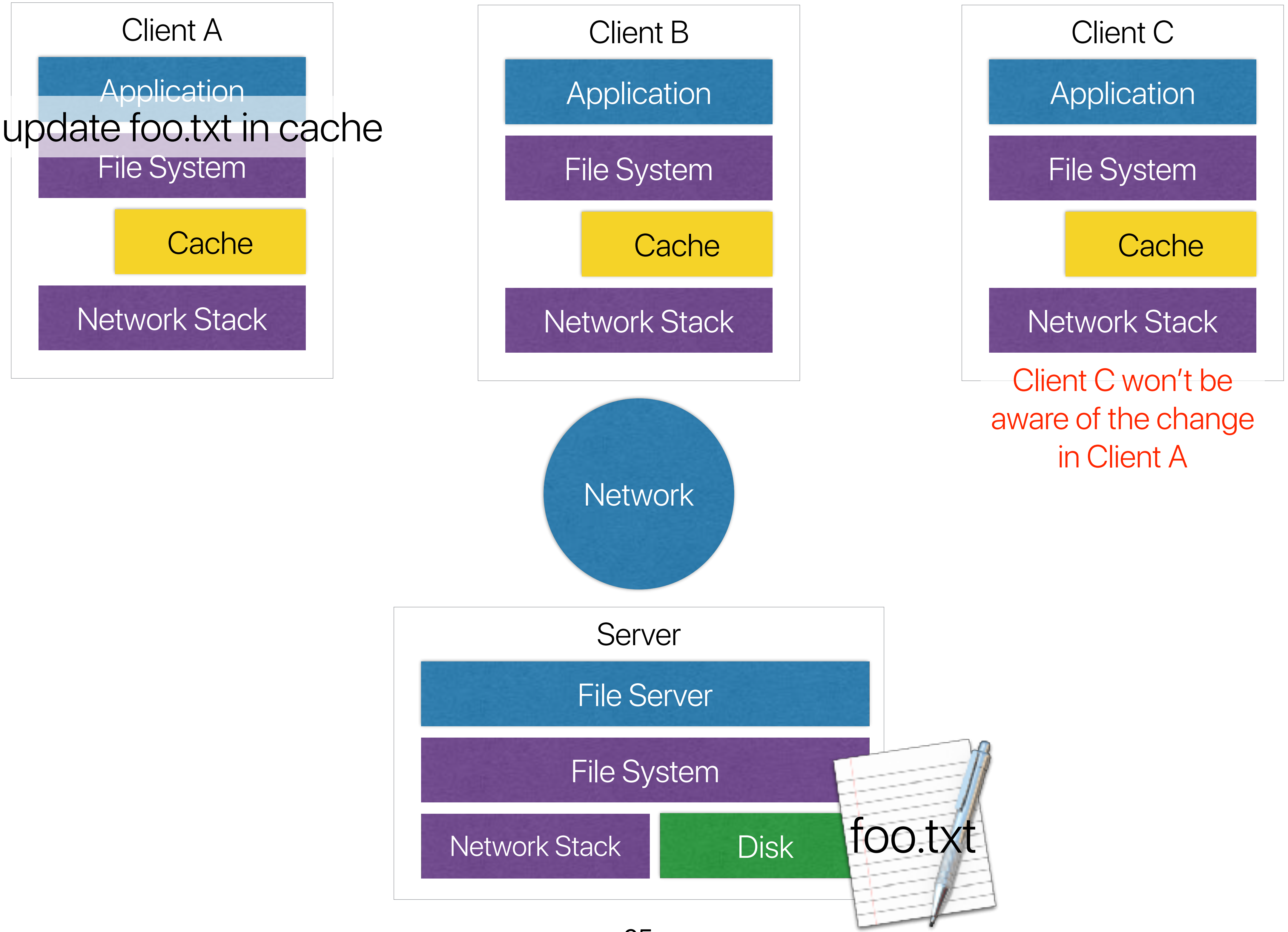
    The server has no knowledge about who has the file
  - ④ Improve the network latency

    Nothing to do with NFS
  - A. 0
  - B. 1
  - [✓ C. 2]
  - D. 3
  - E. 4

Idempotent operations

- Given the same input, always give the same output regardless how many times the operation is employed
- You only need to retry the same operation if it failed

Think about this



Client C won't be aware of the change in Client A

Solution

- Flush-on-close: flush all write buffer contents when close the file
  - Later open operations will get the latest content
- Force-getattr:
  - Open a file requires getattr from server to check timestamps
  - attribute cache to remedy the performance

The Google File System

Sanjay Ghemawat, Howard Gobioff, and
Shun-Tak Leung
Google

https://www.pollev.com/hungweitseng close in 1:00

GFS

- How many of the following fit the optimization goals for GFS?
  ① Optimize for storing small files
  ② Optimize for fast, modern storage devices
  ③ Optimize for random writes
  ④ Optimize for access latencies
  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

GFS

A

B

C

D

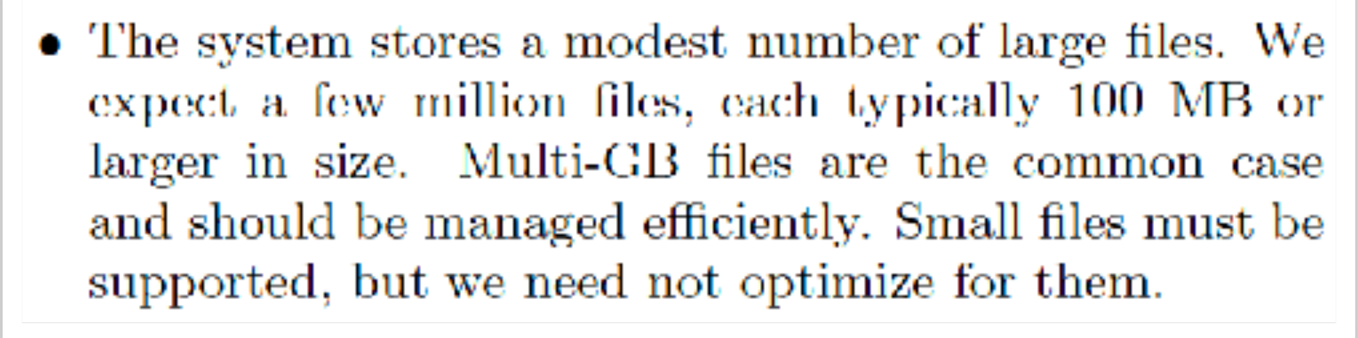
E

Total Results: 0

Powered by Poll Everywhere

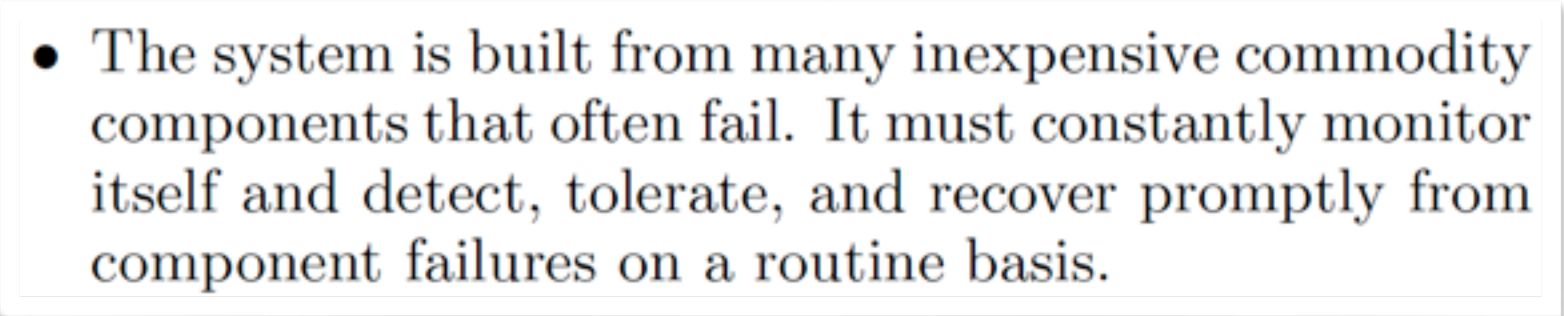
Start the presentation to see live content. For screen share software, share the entire screen. Get help at pollev.com/app

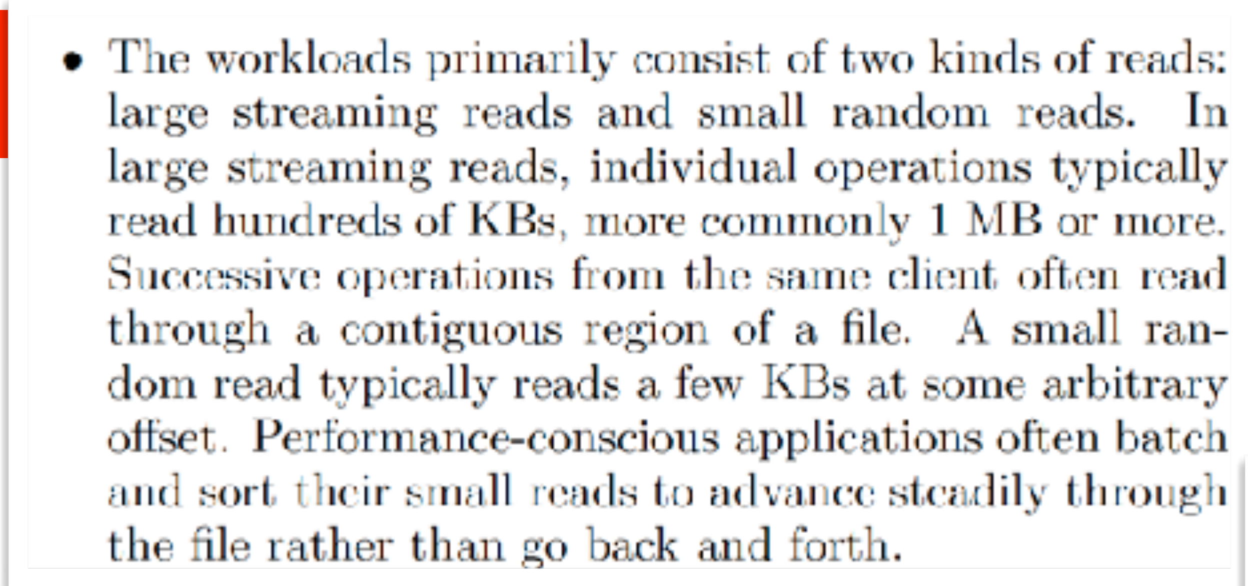
GFS

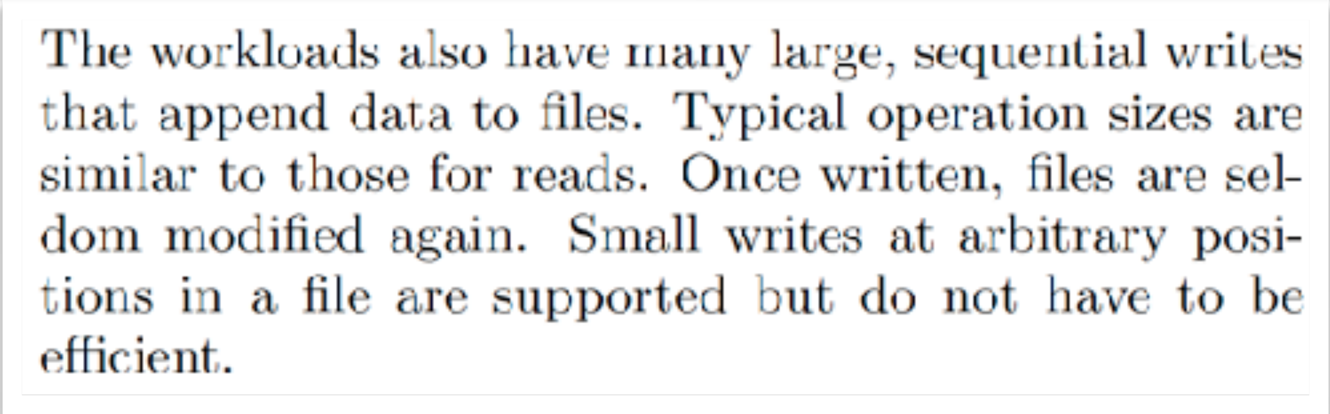

- The system stores a modest number of large files. We expect a few million files, each typically 100 MB or larger in size. Multi-GB files are the common case and should be managed efficiently. Small files must be supported, but we need not optimize for them.

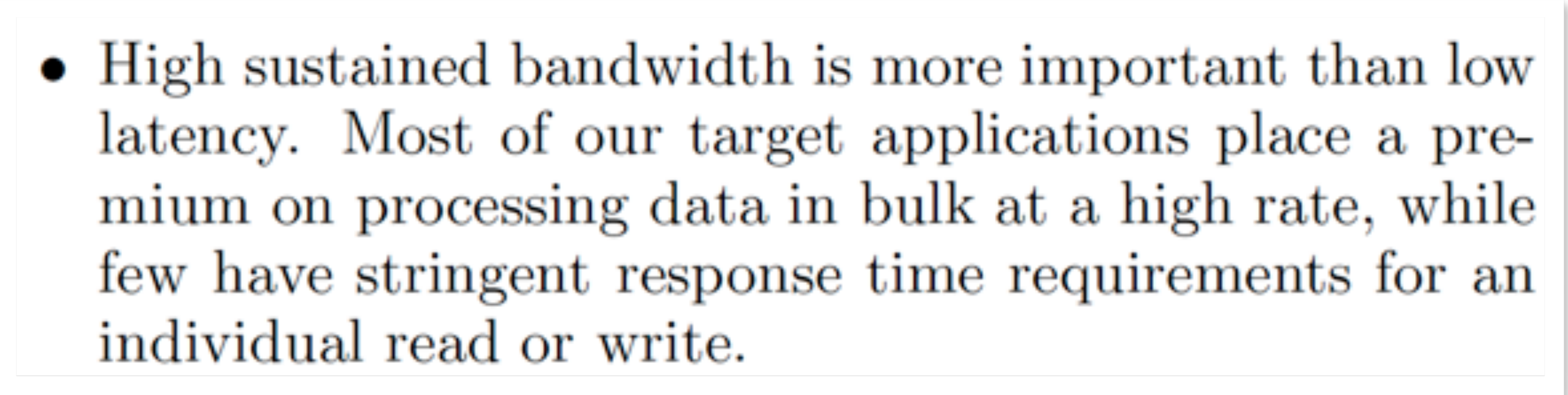
- How many of the following fit the optimization goals for GFS?
  - ☒ Optimize for storing small files
  - ☒ Optimize for fast, modern storage devices
  - ☒ Optimize for random writes
  - ☒ Optimize for access latencies

[✓ A. 0]
B. 1
C. 2
D. 3
E. 4


- The system is built from many inexpensive commodity components that often fail. It must constantly monitor itself and detect, tolerate, and recover promptly from component failures on a routine basis.


- The workloads primarily consist of two kinds of reads: large streaming reads and small random reads. In large streaming reads, individual operations typically read hundreds of KBs, more commonly 1 MB or more. Successive operations from the same client often read through a contiguous region of a file. A small random read typically reads a few KBs at some arbitrary offset. Performance-conscious applications often batch and sort their small reads to advance steadily through the file rather than go back and forth.


The workloads also have many large, sequential writes that append data to files. Typical operation sizes are similar to those for reads. Once written, files are seldom modified again. Small writes at arbitrary positions in a file are supported but do not have to be efficient.


- High sustained bandwidth is more important than low latency. Most of our target applications place a premium on processing data in bulk at a high rate, while few have stringent response time requirements for an individual read or write.

Why we care about GFS

- Conventional file systems do not fit the demand of data centers
- Workloads in data centers are different from conventional computers
  - Storage based on inexpensive disks that fail frequently
  - Many large files in contrast to small files for personal data
  - Primarily reading streams of data
  - Sequential writes appending to the end of existing files
  - Must support multiple concurrent operations
  - Bandwidth is more critical than latency

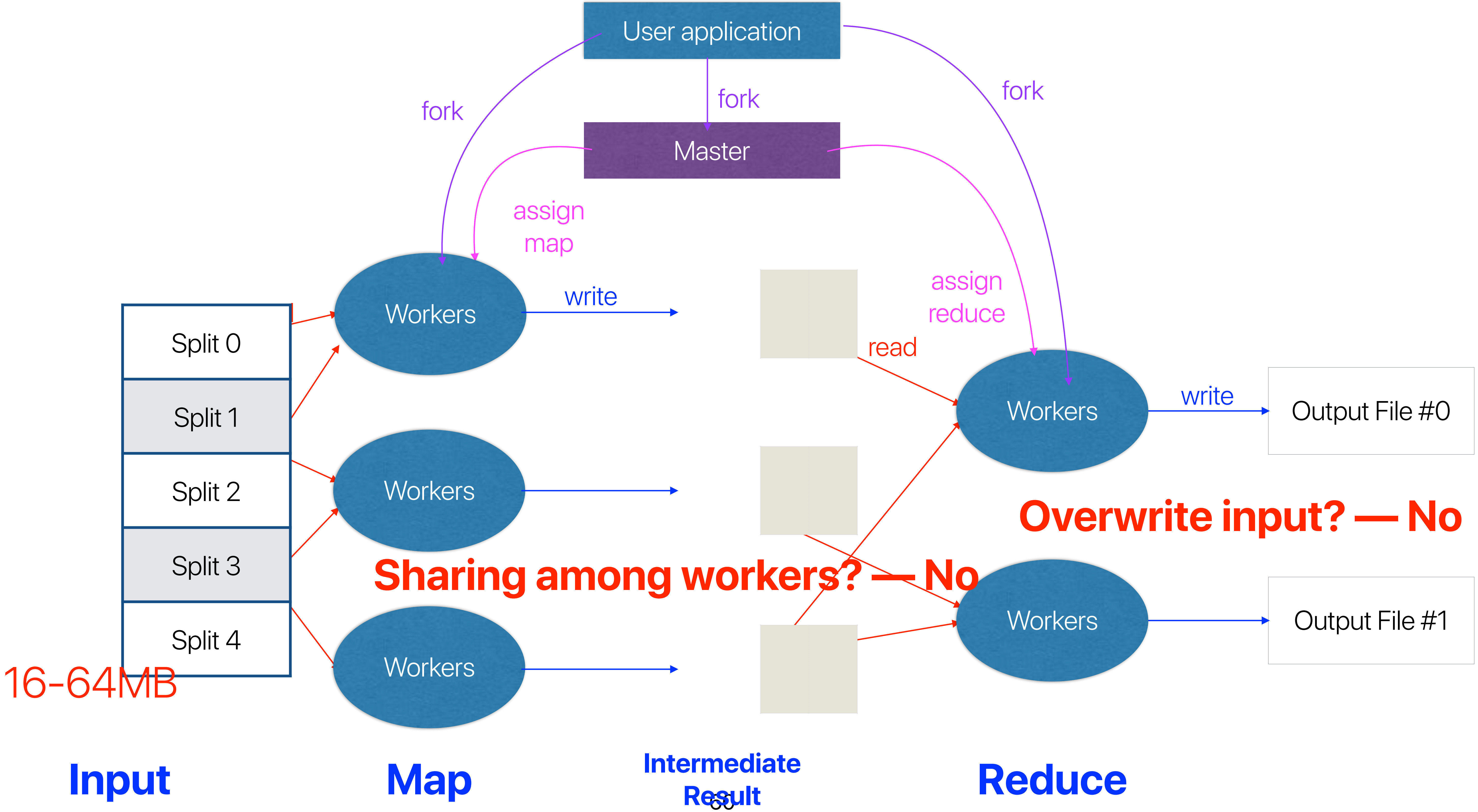
Data-center workloads for GFS

- Google Search (Web Search for a Planet: The Google Cluster Architecture, IEEE Micro, vol. 23, 2003)
- MapReduce (MapReduce: Simplified Data Processing on Large Clusters, OSDI 2004)
  - Large-scale machine learning problems
  - Extraction of user data for popular queries
  - Extraction of properties of web pages for new experiments and products
  - Large-scale graph computations
- BigTable (Bigtable: A Distributed Storage System for Structured Data, OSDI 2006)
  - Google analytics
  - Google earth
  - Personalized search

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat
Google

MapReduce



User application

fork

fork

fork

Master

assign map

assign reduce

Split 0

Split 1

Split 2

Split 3

Split 4

16-64MB

Workers

Workers

Workers

write

read

Workers

Workers

write

Output File #0

Output File #1

Overwrite input? — No

Sharing among workers? — No

Input

Map

Intermediate Result

Reduce

Why we care about GFS

- Conventional file systems do not fit the demand of data centers
- Workloads in data centers are different from conventional computers
    - Storage based on inexpensive disks that fail frequently

      — MapReduce is fault tolerant
    - Many large files in contrast to small files for personal data

      — MapReduce aims at processing large amount of data once
    - Primarily reading streams of data — MapReduce reads chunks of large files
    - Sequential writes appending to the end of existing files

      — Output file keep growing as workers keep writing
    - Must support multiple concurrent operations

      —MapReduce has thousands of workers simultaneously
    - Bandwidth is more critical than latency

      —MapReduce only wants to finish tasks within "reasonable" amount of time

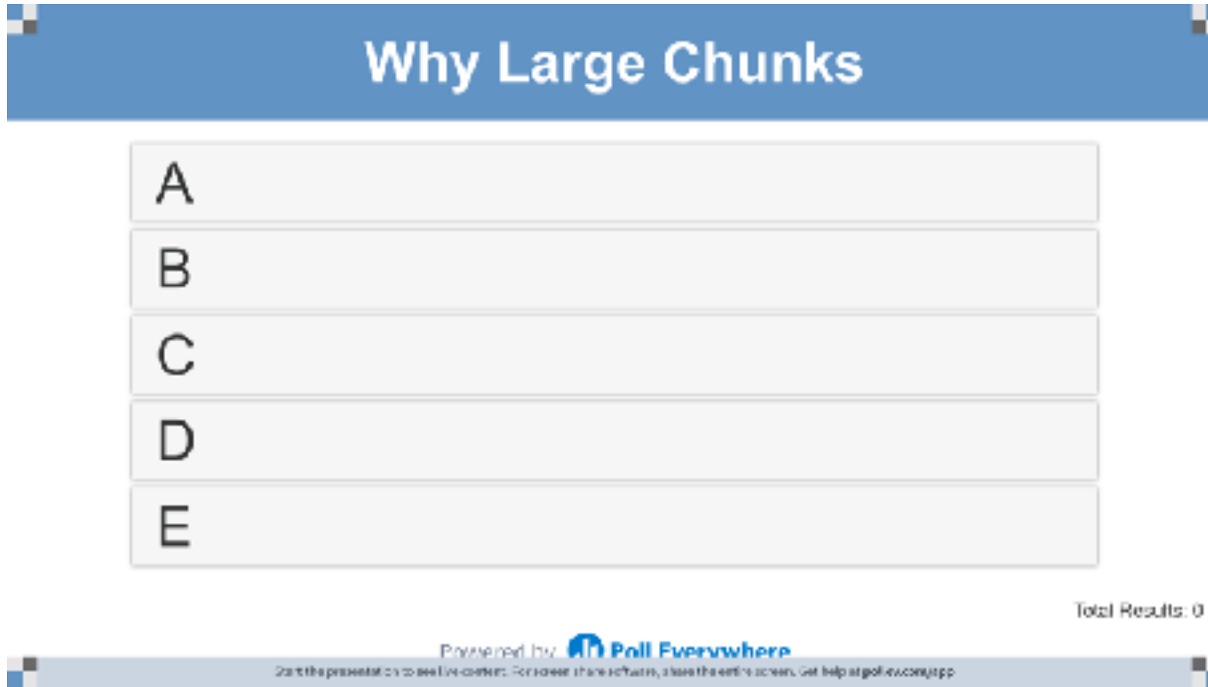
What GFS proposes?

- Maintaining the same interface
  - The same function calls
  - The same hierarchical directory/files
- Files are decomposed into large chunks (e.g. 64MB) with replicas
- Hierarchical namespace implemented with flat structure
- Master/chunkservers/clients

https://www.pollev.com/hungweitseng close in 1:00

Large Chunks

- How many of the following datacenter characteristics can large chunks help address?
  ① Storage based on inexpensive disks that fail frequently
  ② Many large files in contrast to small files for personal data
  ③ Primarily reading streams of data
  ④ Sequential writes appending to the end of existing files
  ⑤ Must support multiple concurrent operations
  ⑥ Bandwidth is more critical than latency
  A. 1
  B. 2
  C. 3
  D. 4
  E. 5



Why Large Chunks

A

B

C

D

E

Total Results: 0

Powered by Poll Everywhere

Large Chunks

- How many of the following datacenter characteristics can large chunks help address?

  ① Storage based on inexpensive disks that fail frequently
  [✓ ② Many large files in contrast to small files for personal data]
  [✓ ③ Primarily reading streams of data]
  [✓ ④ Sequential writes appending to the end of existing files]
  ⑤ Must support multiple concurrent operations
  [✓ ⑥ Bandwidth is more critical than latency]
  A. 1
  B. 2
  C. 3
  [✓ D. 4]
  E. 5

Latency Numbers Every Programmer Should Know

Operations	Latency (ns)	Latency (us)	Latency (ms)	
L1 cache reference	0.5 ns			~ 1 CPU cycle
Branch mispredict	5 ns			
L2 cache reference	7 ns			14x L1 cache
Mutex lock/unlock	25 ns			
Main memory reference	100 ns			20x L2 cache, 200x L1 cache
Compress 1K bytes with Zippy	3,000 ns	3 us		
Send 1K bytes over 1 Gbps network	10,000 ns	10 us		
Read 4K randomly from SSD*	150,000 ns	150 us		~1GB/sec SSD
Read 1 MB sequentially from memory	250,000 ns	250 us		
Round trip within same datacenter	500,000 ns	500 us		
Read 1 MB sequentially from SSD*	1,000,000 ns	1,000 us	1 ms	~1GB/sec SSD, 4X memory
Read 512B from disk	10,000,000 ns	10,000 us	10 ms	20x datacenter roundtrip
Read 1 MB sequentially from disk	20,000,000 ns	20,000 us	20 ms	80x memory, 20X SSD
Send packet CA-Netherlands-CA	150,000,000 ns	150,000 us	150 ms	

Announcement

- Second last Reading Quiz due next Tuesday
- Office hour
    - MTu 11a-12p, W 2p-3p & F 11a-12p
    - Use the office hour Zoom link, not the lecture one
- Project
    - Due 3/3
    - No late submission is allowed — to make time for grading and potential of regrading
    - Revision policy
        - fix your bugs and schedule a meeting with the TA within a week after grading
        - You have to answer several design questions
        - you can get 70% of the remaining grades if you passed

Computer
Science &
Engineering

202

つづく