SUNDR:

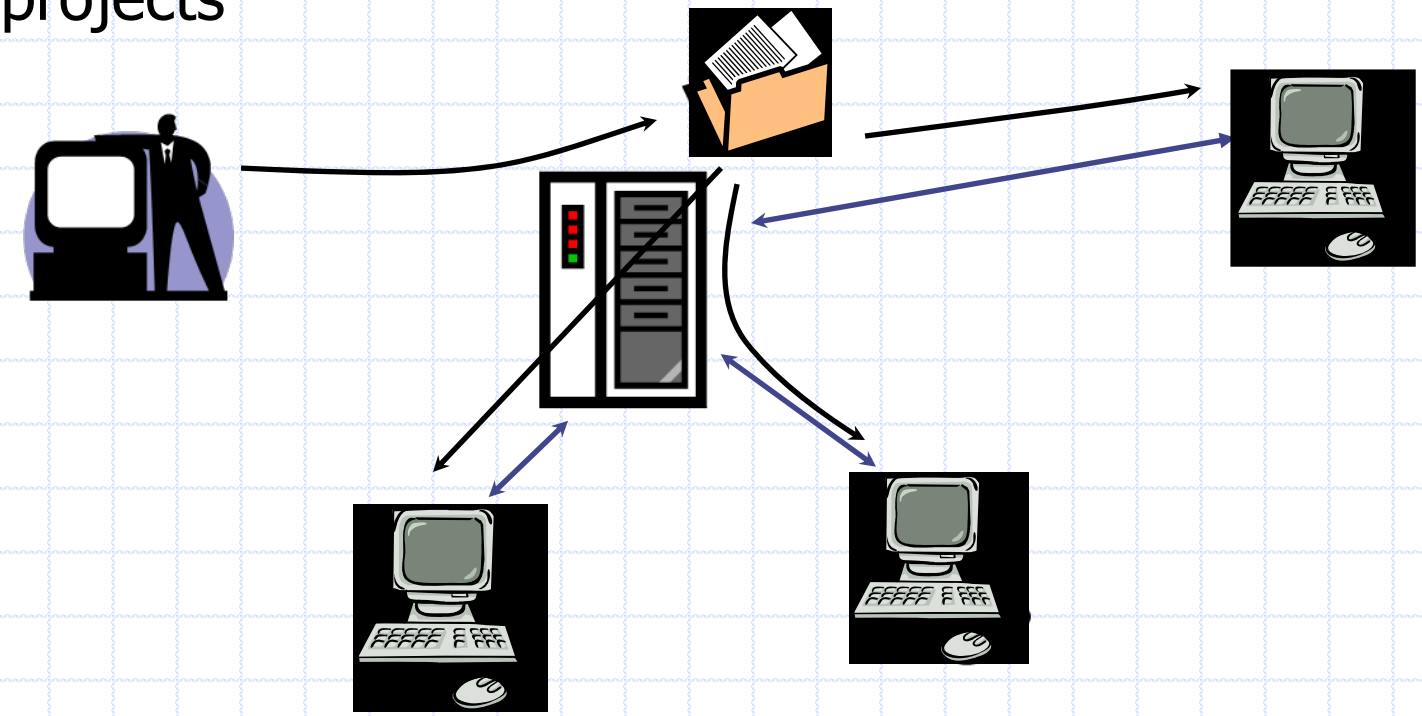
Secure Untrusted Data Repository

Jinyuan Li	NYU/Stanford
Maxwell Krohn	MIT
David Mazières	Stanford
Dennis Shasha	NYU

Stanford Computer Forum Security Workshop Mar 20, 2006

Motivation

- File system integrity is critical
  - sourceforge.net: 115,000+ projects, including kernel projects



Goal

- Prevent undetected tampering with your files!

Current approaches

- Trust system administrator to do a good job
  - Keep up with latest security patches
  - Restrict accesses as much as possible
  - …

Not always reliable

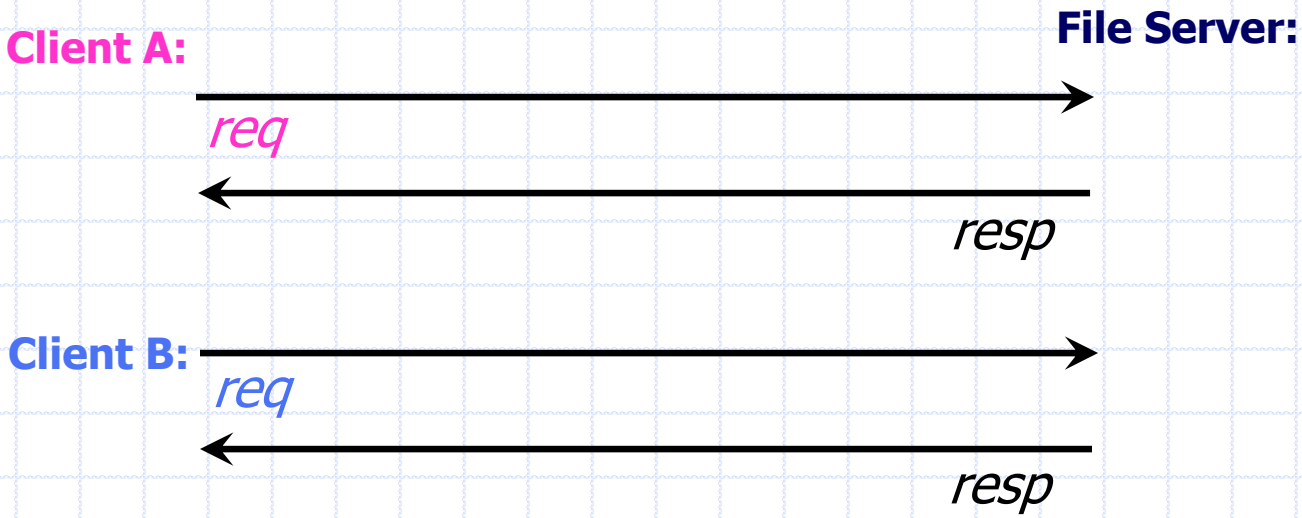
SUNDR's approach

- SUNDR is a network f/s designed for running on untrusted, or even compromised server
- Place trust in users authorized to modify particular files, not in server or admins maintaining server
- SUNDR properties:
  - Unauthorized operations will be immediately detected
  - If server drops operations, can be caught eventually

Talk Outline

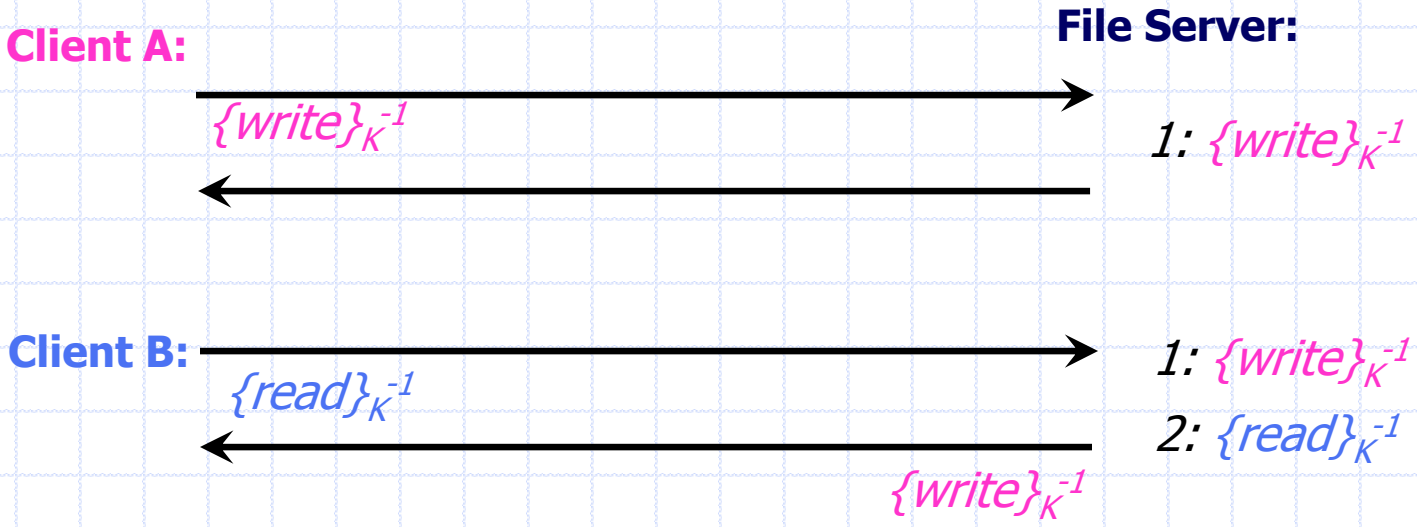
- Motivation
- Design
  - A strawman file system
  - SUNDR design
- Implementation
- Evaluation

Traditional file system model



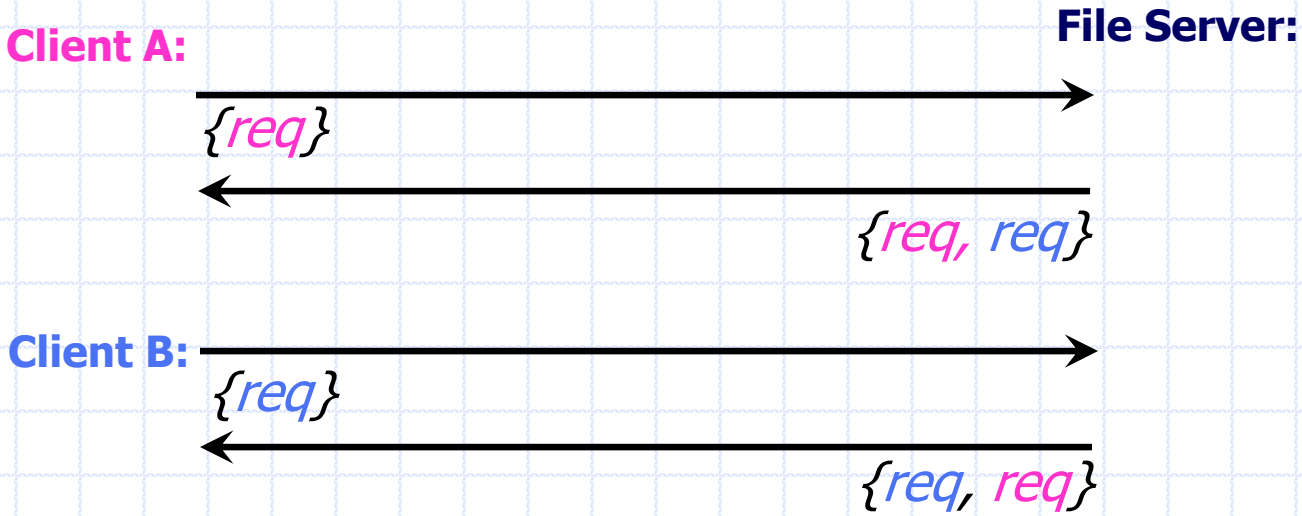
- Server can't prove the requests it has received and executed
  - Trust servers to execute the requests faithfully
  - Trust servers to return correct responses

SUNDR model



- Server does not execute anything
  - Server just stores signed requests from clients
  - Server replies the request with other signed requests
  - Client reconstructs the response by executing returned requests in order

Danger: Drop or reorder requests

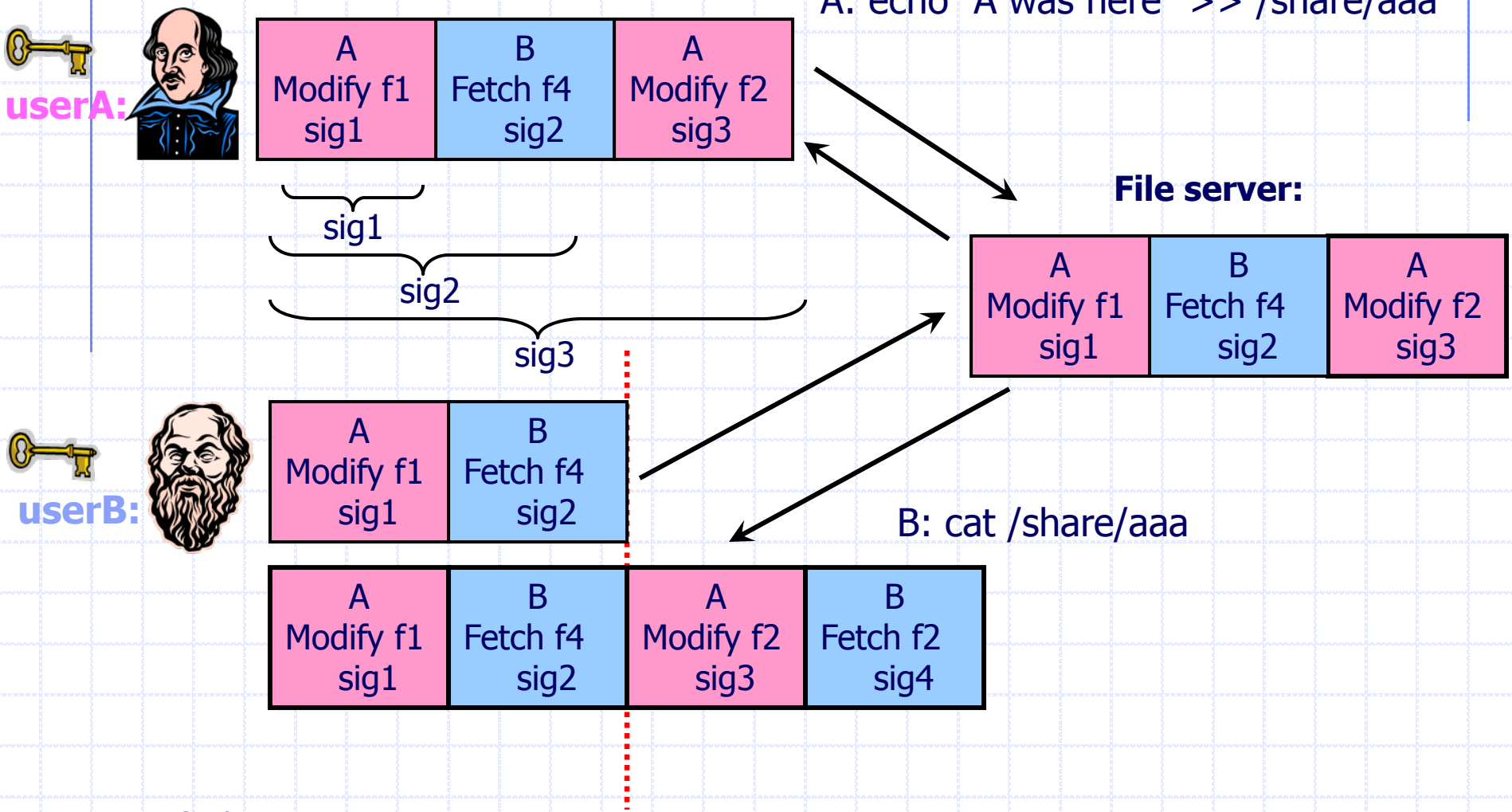


- Server can drop some requests
  - Back out critical security patches
- Or can show requests to clients in diff orders
  - Overwrite files with old version

Ideal File system semantics

- File system calls can be mapped to fetch/modify operations
  - Fetch – client validates cache, or downloads new data
  - Modify – client makes new change visible to others
- "Fetch-modify" consistency: A *fetch* reflects exactly the authorized *modifications* that happen before it
- Impossible without online trusted parties
  - Goal: Get as close to possible to "fetch-modify" consistency without online trusted parties

Strawman FS: Signed log approach



An ordering relation

A's latest log:

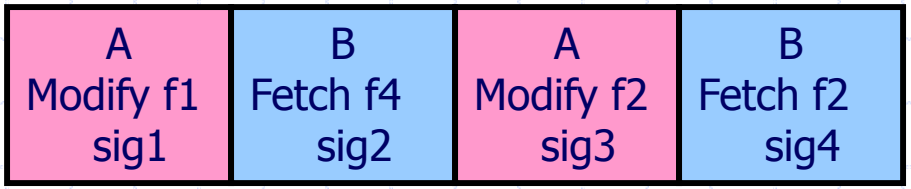
Log_A

We define ≤ relation:

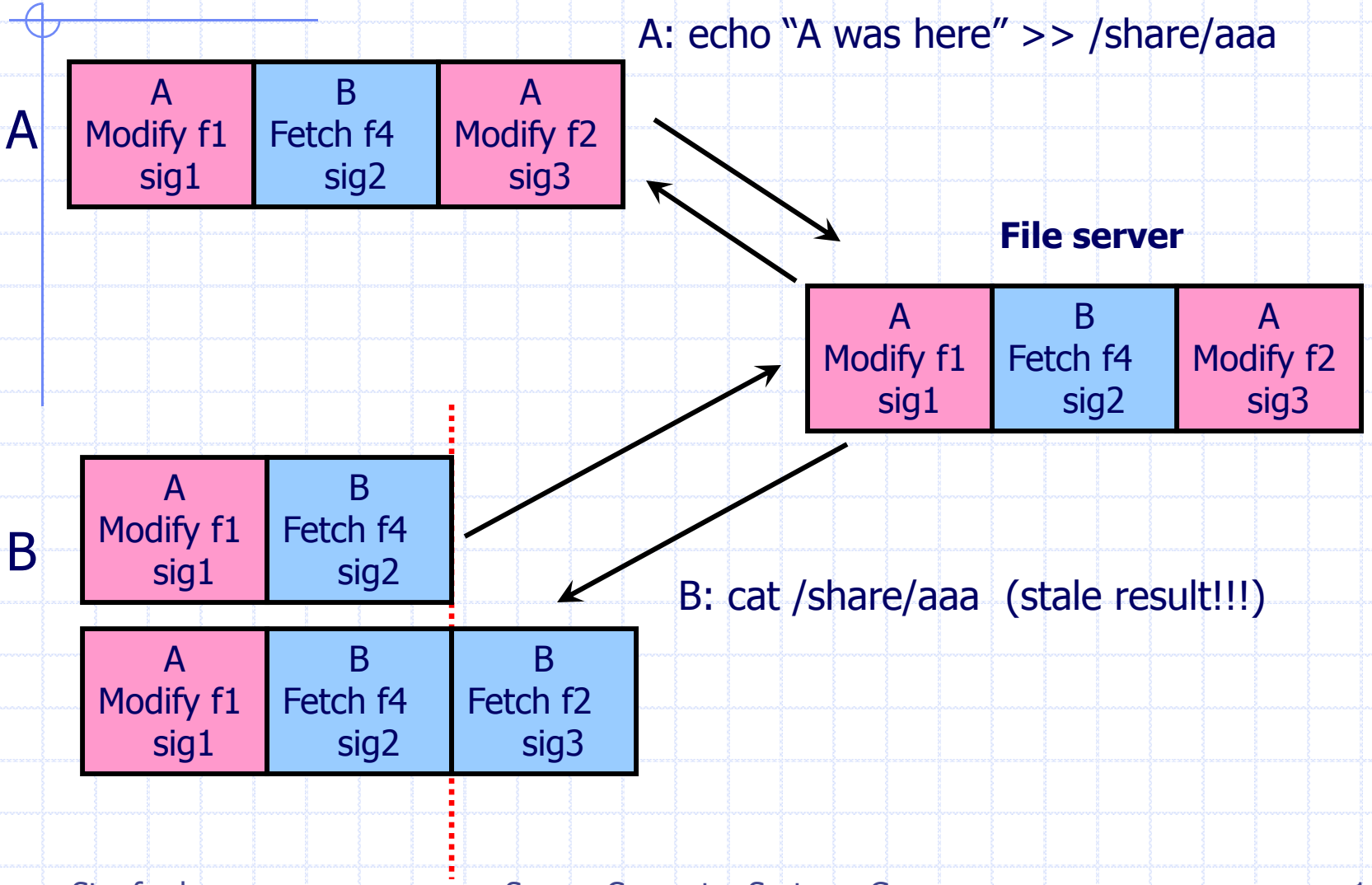
$Log_A \leq Log_B$ iff Log_A is prefix of Log_B

B's latest log:

Log_B

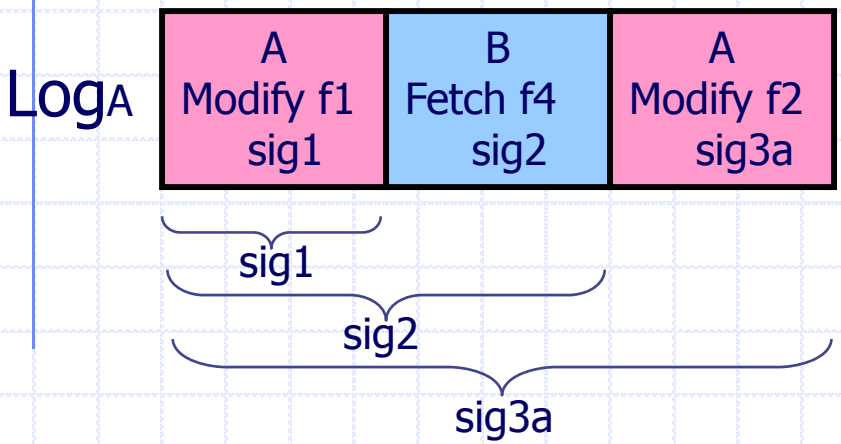


Detecting attacks by the server

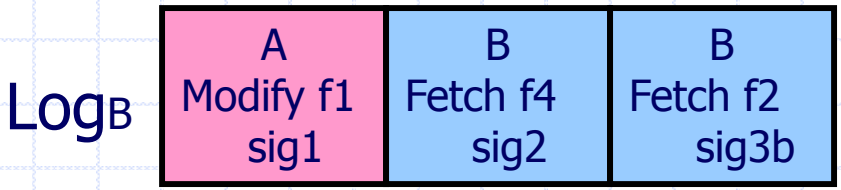


Detecting attacks by the server

A's latest log:



B's latest log:

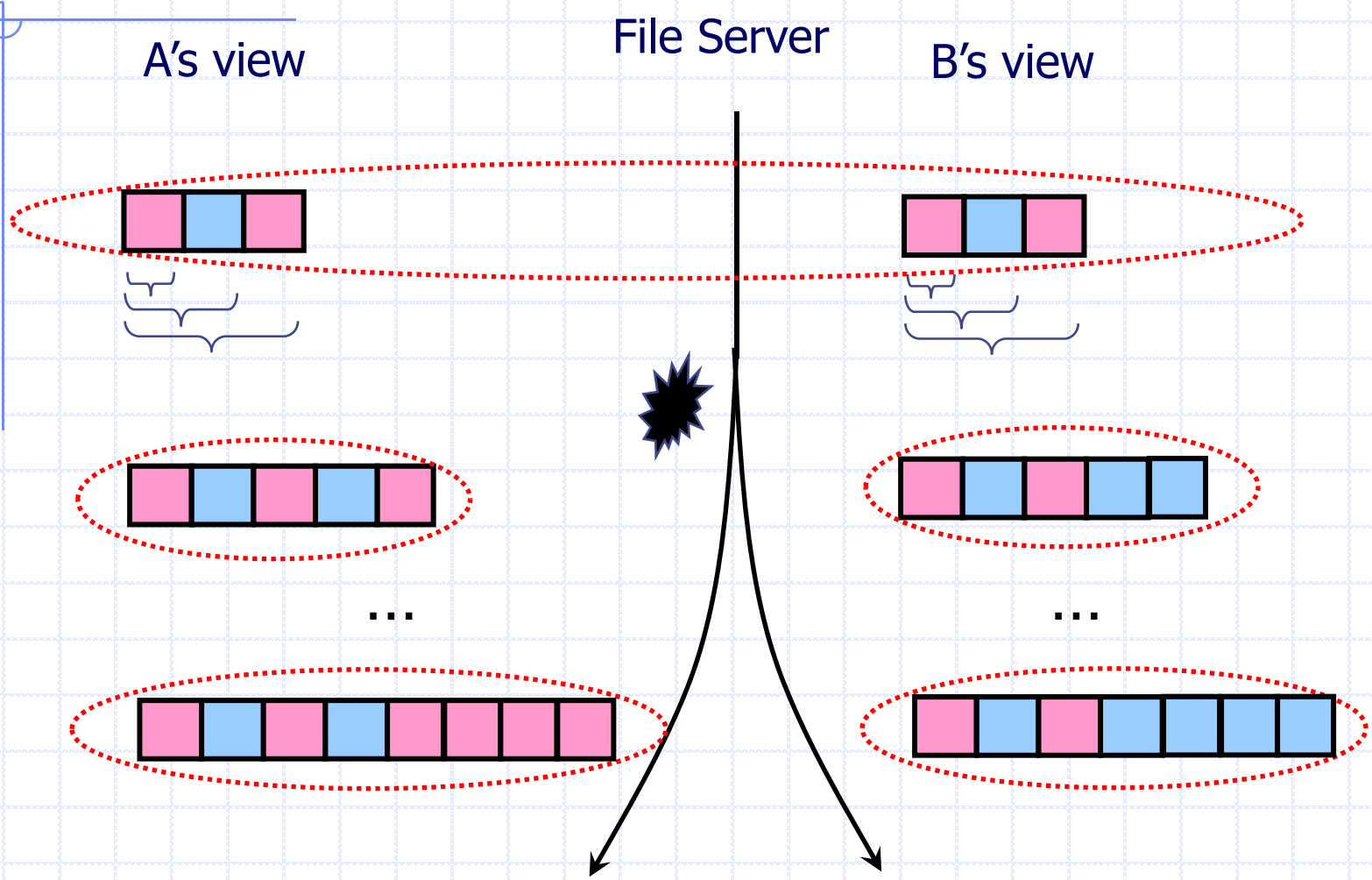


A's log and B's log can no longer be ordered:
$Log_A \not\leq Log_B$, $Log_B \not\leq Log_A$

Properties of Strawman FS

- ✘ High overhead, no concurrency
- ✓ A bad server can't make up operations users didn't perform
- ✓ A bad server can conceal users' operations from each other, however, it will be detected if users check with each other.
  - Call this property "fork consistency"

Fork Consistency: A tale of two worlds



Implications of fork consistency

- Closest possible consistency to "fetch-modify" without online trusted parties
- Can be leveraged with online trusted parties to detect violations of "fetch-modify" consistency
  - users periodically gossip to check violations
  - or deploy a trusted online "timestamp" box

Talk Outline

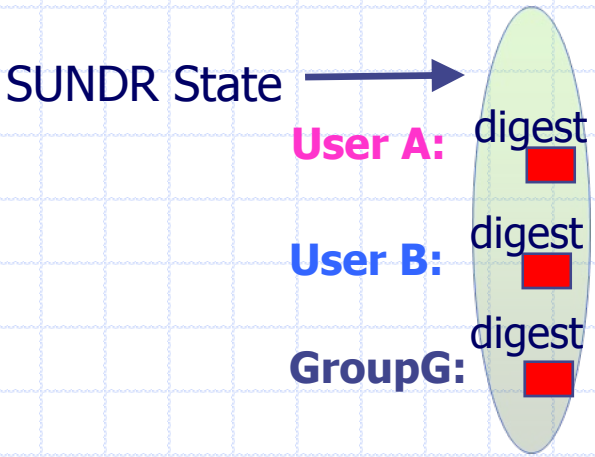
- Motivation
- Design
  - Strawman FS
  - SUNDR approach
- Implementation
- Evaluation

SUNDR Data Structure

- Part I
  - How to reduce each user's writable files to a hash value?
  - => given this value, we can fetch and verify any piece of data
- Part II
  - How to retrieve each other's latest hash value w/o trusted online parties?
  - => achieve fork consistency

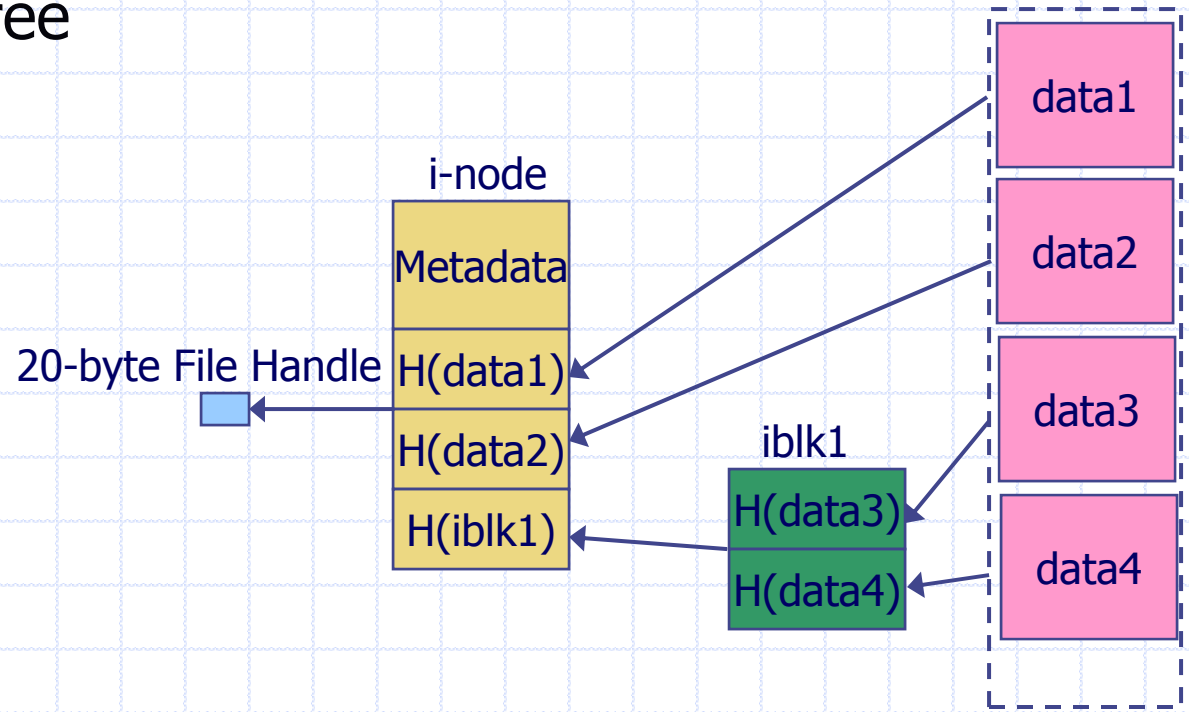
SUNDR data structures (Part I)

- Each file is writable by one user or group
- Partition files by allowed writers
  - Hash each partition down to a 20-byte digest
- SUNDR FS state is the aggregation of all users' digests



Hash tree (1): File handle

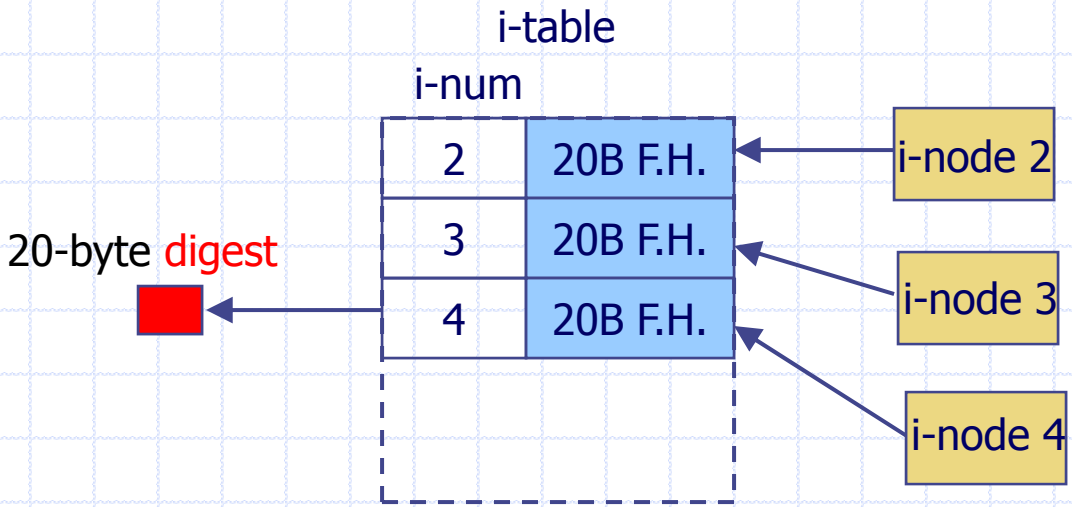
- Each file is hashed into a 20-byte value using a hash tree



- Blocks are stored and indexed by their content-hash
  - No trust needed on the server

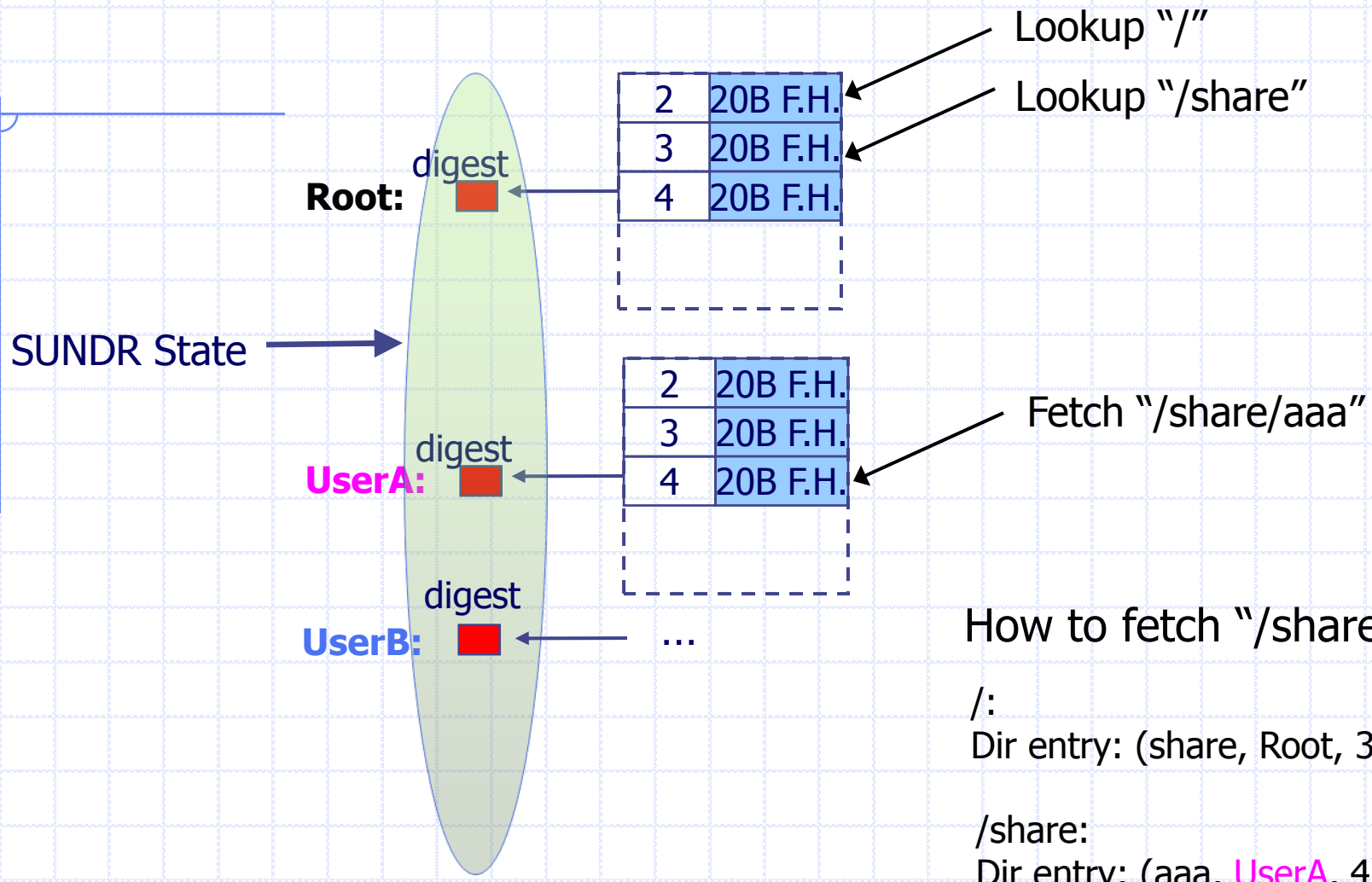
Hash tree (2): FS digest

- Hash all files writable by each user/group to a 20-byte digest



- From this digest, client can retrieve and verify any block of any file (SFSRO, CFS, Pond, …)

SUNDR FS



How to fetch "/share/aaa"?

/:
Dir entry: (share, Root, 3)

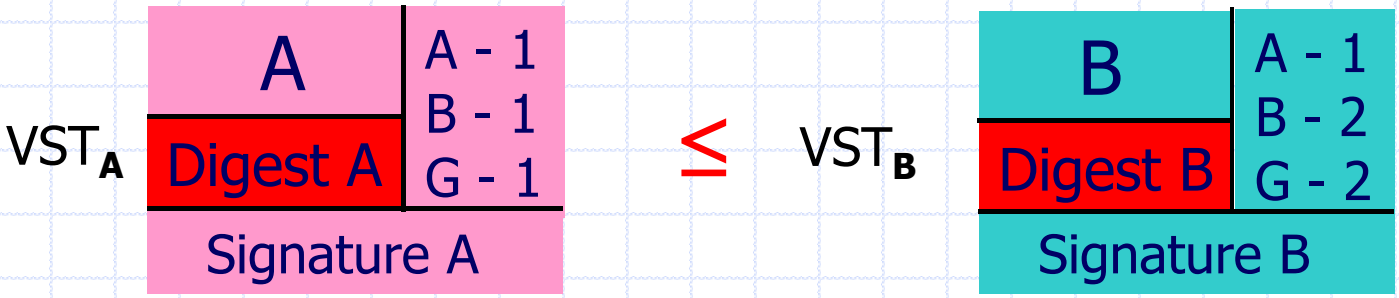
/share:
Dir entry: (aaa, UserA, 4)

SUNDR data structure (Part II)

- Want server to order users' fetch/modify operations w.r.t. users' digests
- Goal: Expose server's failure to order operations properly

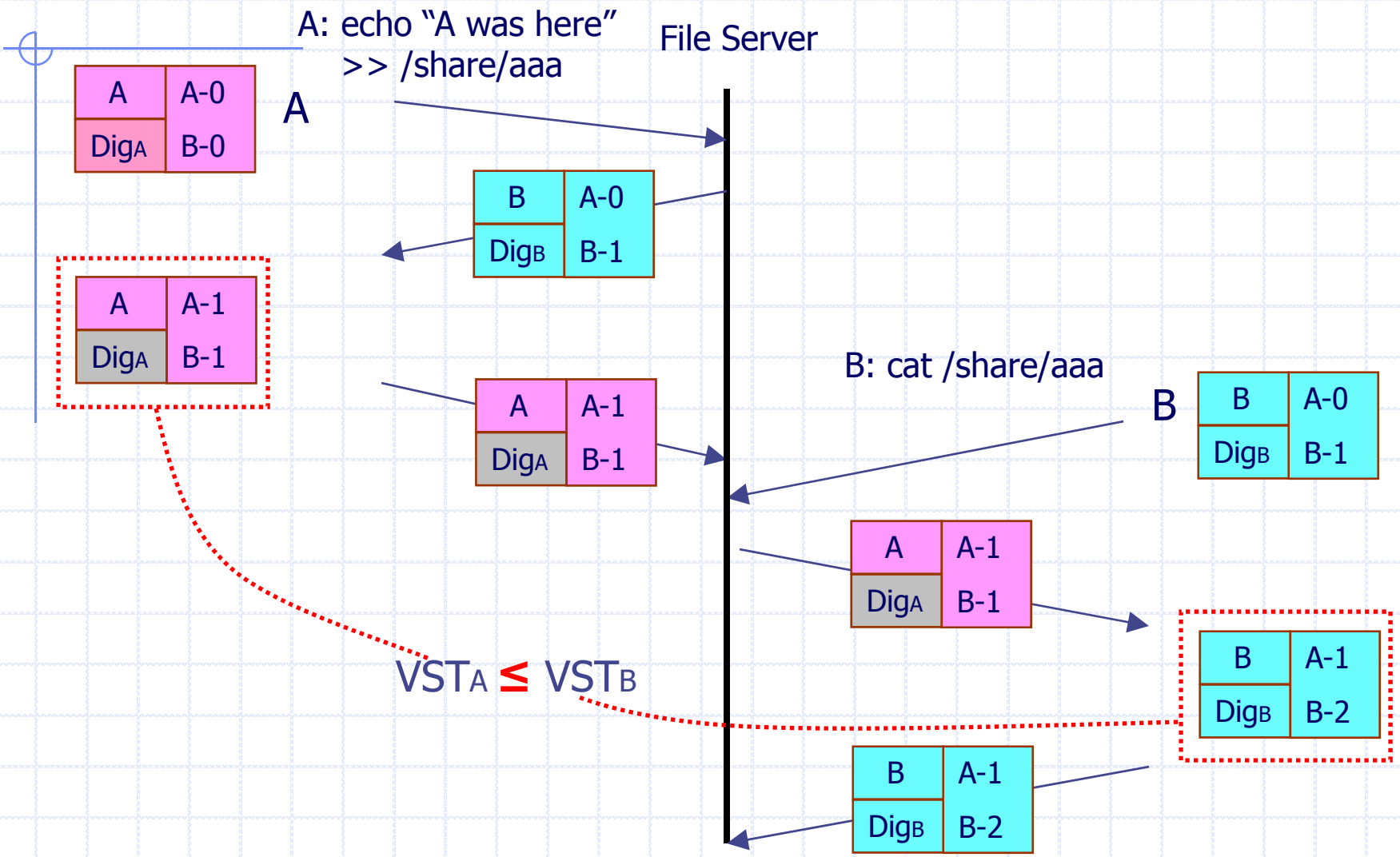
- Sign version vector along with digest
- Version vectors will expose ordering failures

Version structure (VST)

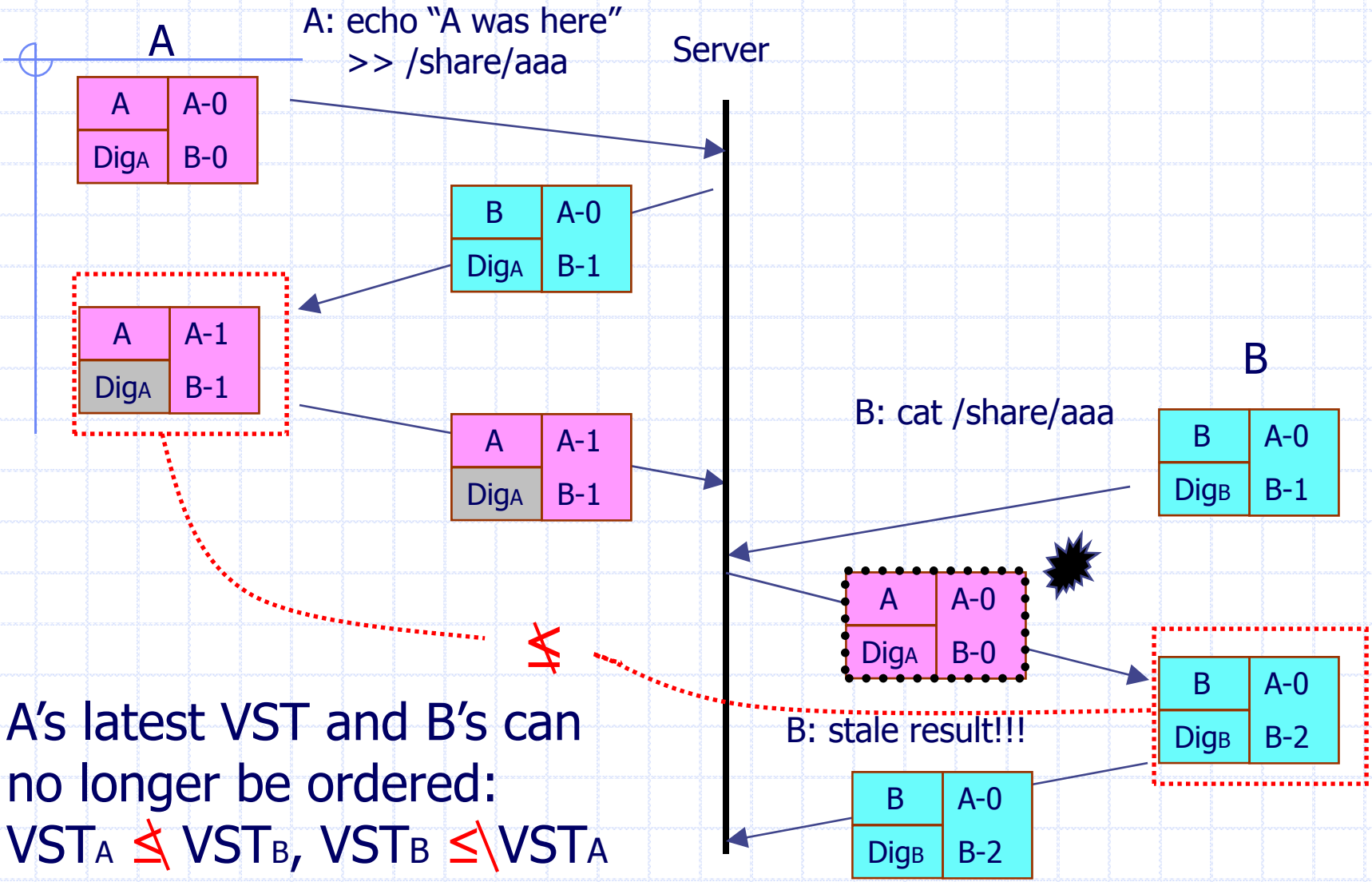


- Each user has its own version structure (VST)
- Server keeps latest VSTs of all users
- Clients fetch all other users' VSTs from server before each operation and cache them
- We order $VST_A \leq VST_B$ iff all the version numbers in VST_A are less than or equal in VST_B

Updating VST: An example



Detecting attacks

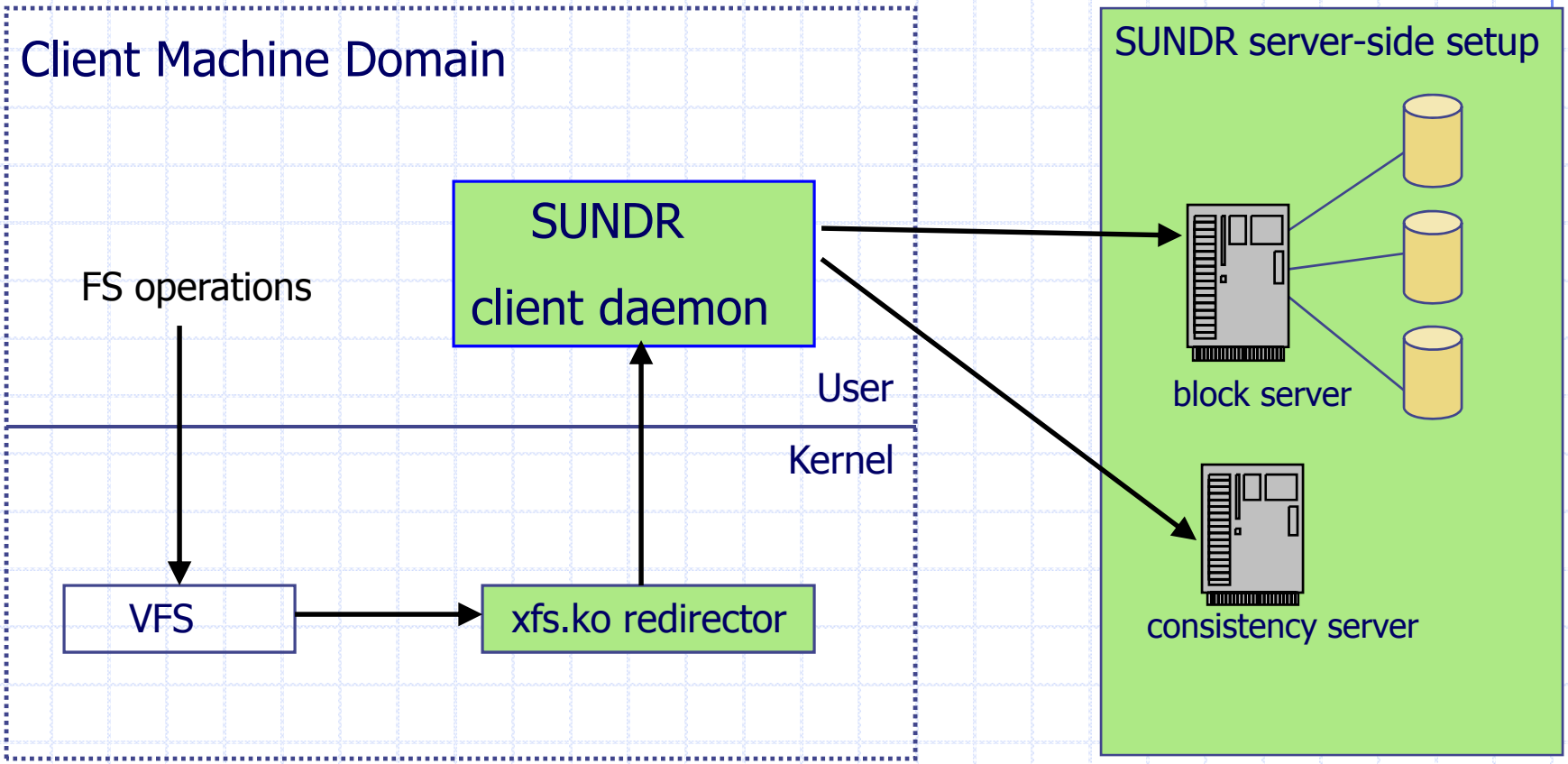


A's latest VST and B's can no longer be ordered:
$VST_A \not\leq VST_B$, $VST_B \not\leq VST_A$

Talk Outline

- Motivation
- Design
  - Straw-man FS
  - SUNDR approach
- Implementation
- Evaluation

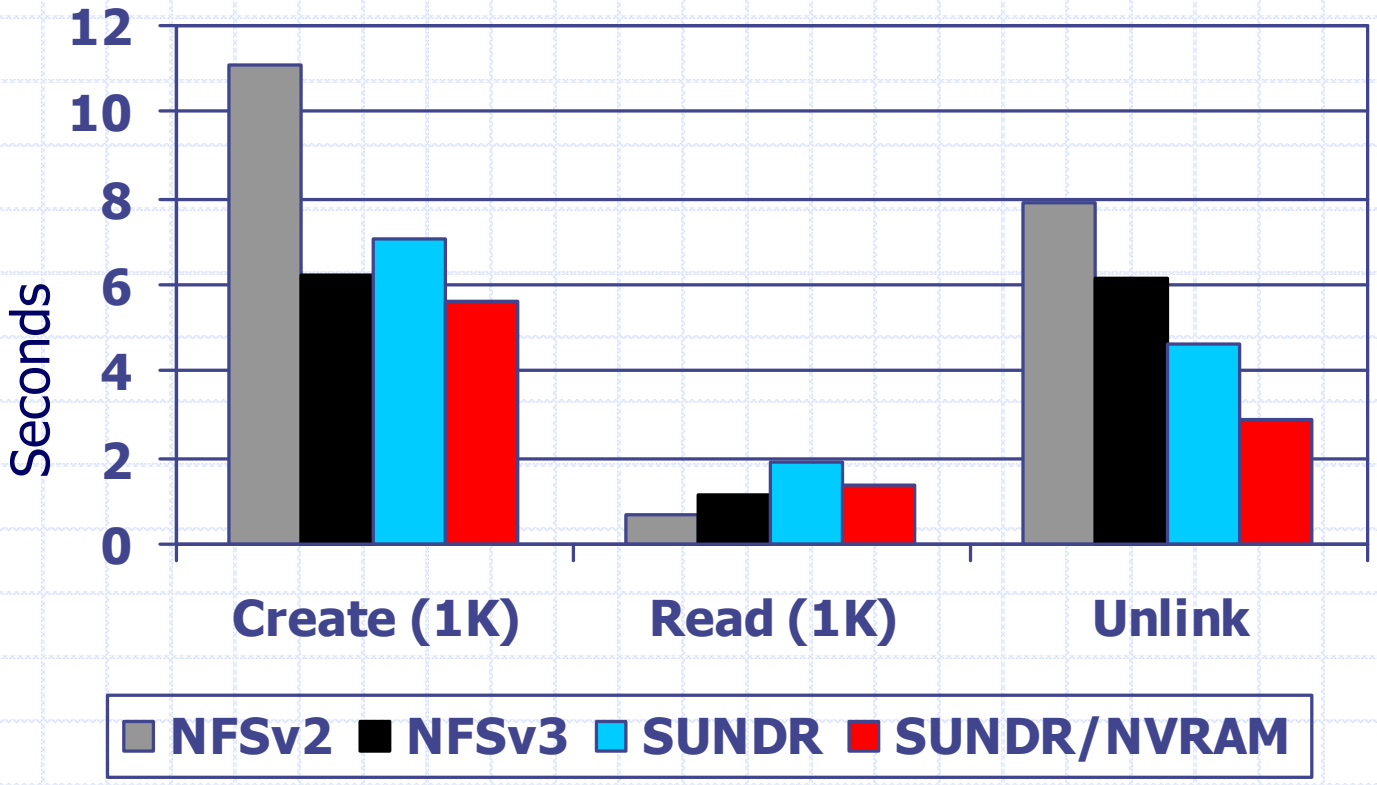
SUNDR Implementation



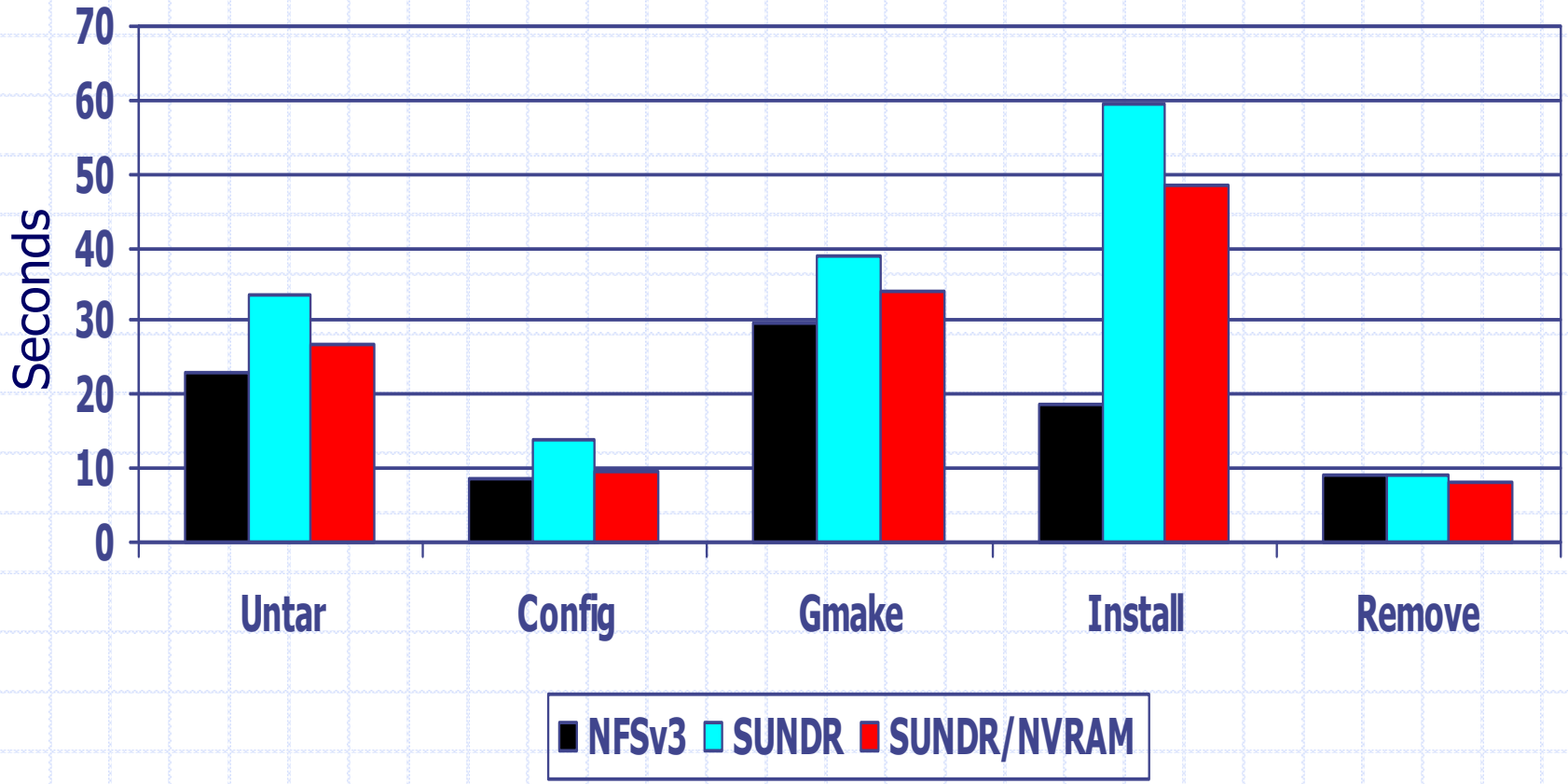
Evaluation

- Running on FreeBSD 4.9
  - PentiumIV 3G, 3G RAM, 100Mbps LAN
- Two configurations:
  - SUNDR : write updates to disk synchronously
  - SUNDR/NVRAM : simulates effects of NVRAM
- Esign cryptographic overhead
  - Sign: 155us
  - Verify: 100us

LFS small file benchmark



Emacs installation performance



Conclusion

- SUNDR provides file system integrity with untrusted servers
  - Users detect unauthorized operations immediately
  - Users can detect consistency violations eventually
- Yes, SUNDR is a practical file system
  - performance is close to NFS