Summary

Siao-Ting Wang

Outline

- Summary of papers
  - Problems
  - Solutions
- Secure Untrusted Data Repository(SUNDR)

Problems

- Both papers deal with malicious attack.
    - Practical Byzantine Fault Tolerance
        - Malicious attacks and software errors cause faulty servers to exist. Users may get incorrect responses.
    - Needham-Schroeder, Kerberos
        - Not deal with software errors but intruders. Intruders may alter or copy parts of messages, replay messages, or emit false material. Users may get incorrect responses.

Solutions

- The two papers focus on two different aspects to solve attack problem.
    - Practical Byzantine Fault Tolerance
        - Assume some servers may be untrusted.
        - Fault Tolerance
            - Rely on a super-majority of votes to decide on correct computation.
            - Tolerates <=f failures using a RSM with 3f+1 replicas.
    - Needham-Schroeder, Kerberos
        - Assume servers are all trusted.
        - Combine authentication with Key Distribution Center to secure communication.
            - Provide a centralized authentication server to authenticate users to servers and servers to users.
            - Relies on conventional encryption, not public-key encryption.
            - Issue time limited tickets to authenticated users.

Challenge

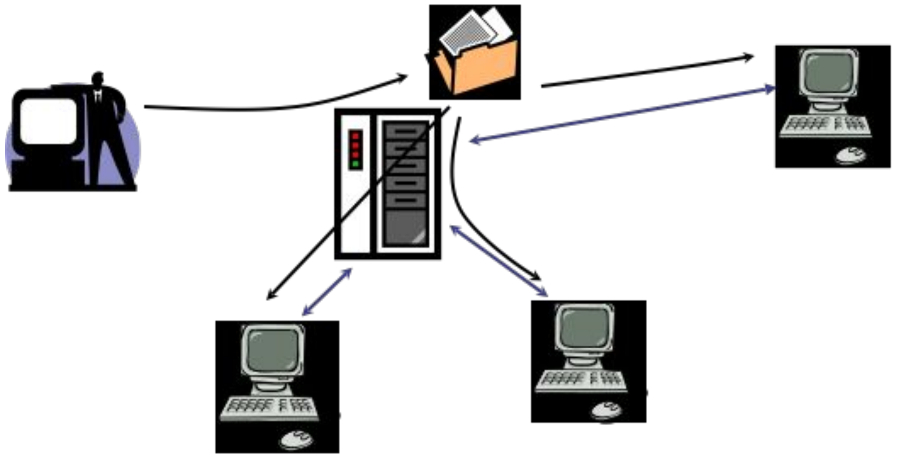
- However, there are some other security problems that we want to solve.
- What can we do when > ⅓ servers are bad?
- If we assume that some servers may be untrusted, can we detect the bad server?

Secure Untrusted Data Repository(SUNDR)

Jinyuan Li, Maxwell Krohn , David Mazieres, and Dennis Shasha
NYU Department of Computer Science

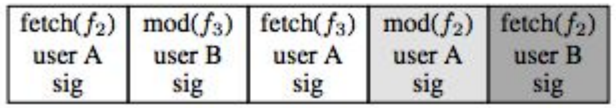
What is Secure Untrusted Data Repository(SUNDR)

- A (single server) network file system
- Handle potential Byzantine server behavior
- Design for running on untrusted servers
- Property
  - Unauthorized operations will be immediately detected
  - If server drops operations, can be caught eventually
- Ex: sourceforge.net: 90,000+ projects, including kernel projects



Strawman File System

- Represent file system calls as fetch/modify operations
  - Fetch: client downloads new data
  - Modify: client makes new change visible to others
- An authorized user that interacts the file system remembers their last operation and its signature. When that user comes back for another operation, it can verify the history of the file system up to and including its own last operation

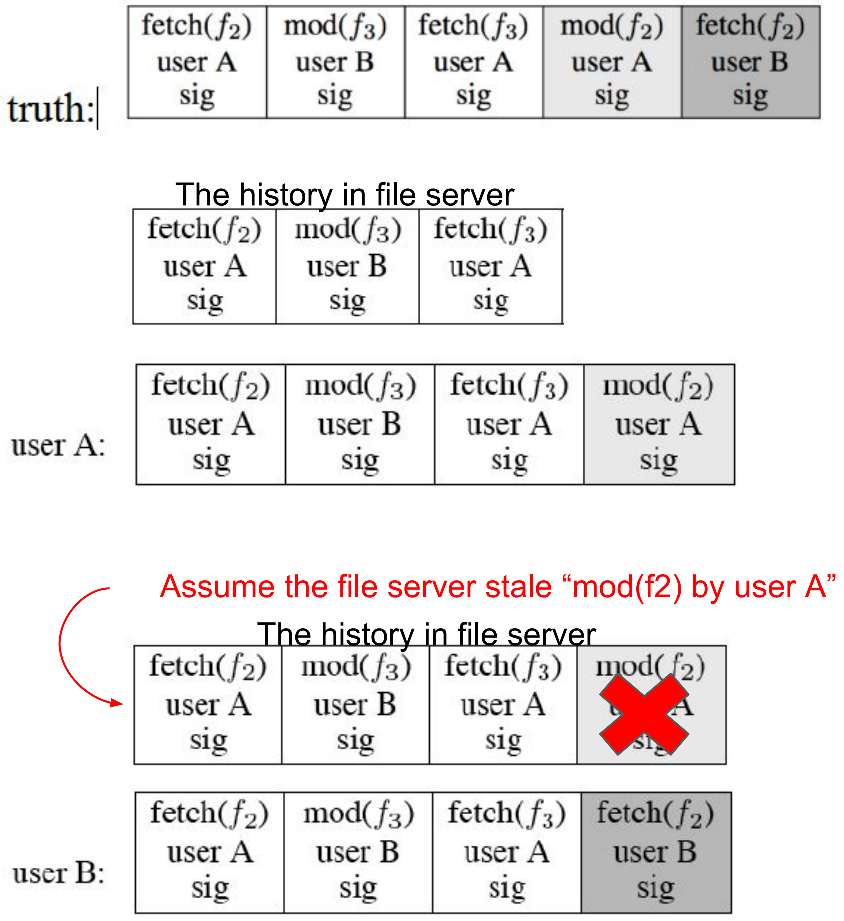


fetch(f_2)	mod(f_3)	fetch(f_3)	mod(f_2)	fetch(f_2)
user A	user B	user A	user A	user B
sig	sig	sig	sig	sig

The history in file server

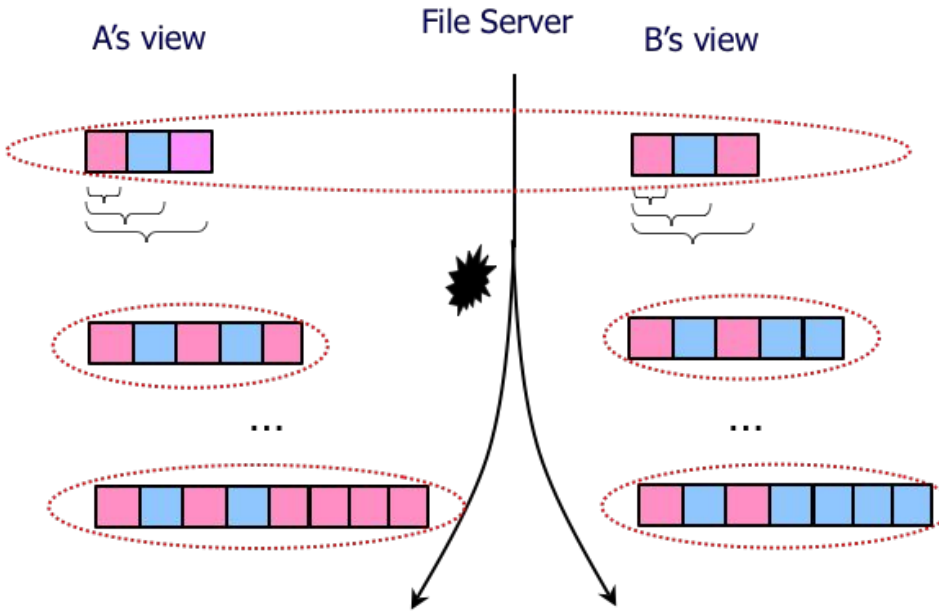
Strawman File System

- To fetch/modify a file, user:
  - Acquires global lock
  - Downloads entire history of filesystem
  - Validates each users' most recent signature
  - Checks that its most recent operation is in the downloaded history
  - Traverses history to construct FS, validates each operation is permissible
- Achieve fork consistency
- Problem
  - Unpractical and inefficient
  - Every access needs to check the entire history.


truth:
fetch(f_2) user A sig | mod(f_3) user B sig | fetch(f_3) user A sig | mod(f_2) user A sig | fetch(f_2) user B sig
The history in file server
fetch(f_2) user A sig | mod(f_3) user B sig | fetch(f_3) user A sig
user A:
fetch(f_2) user A sig | mod(f_3) user B sig | fetch(f_3) user A sig | mod(f_2) user A sig
Assume the file server stale "mod(f2) by user A"
The history in file server
fetch(f_2) user A sig | mod(f_3) user B sig | fetch(f_3) user A sig | mod(f_2) user A sig
user B:
fetch(f_2) user A sig | mod(f_3) user B sig | fetch(f_3) user A sig | fetch(f_2) user B sig

Fork Consistency

- If user A sees that user B fetches or modifies a file at time t, then this means that A saw all of B's prior changes as well
- Can be used as detections of server's misbehavior
  - Users periodically gossip to check violations

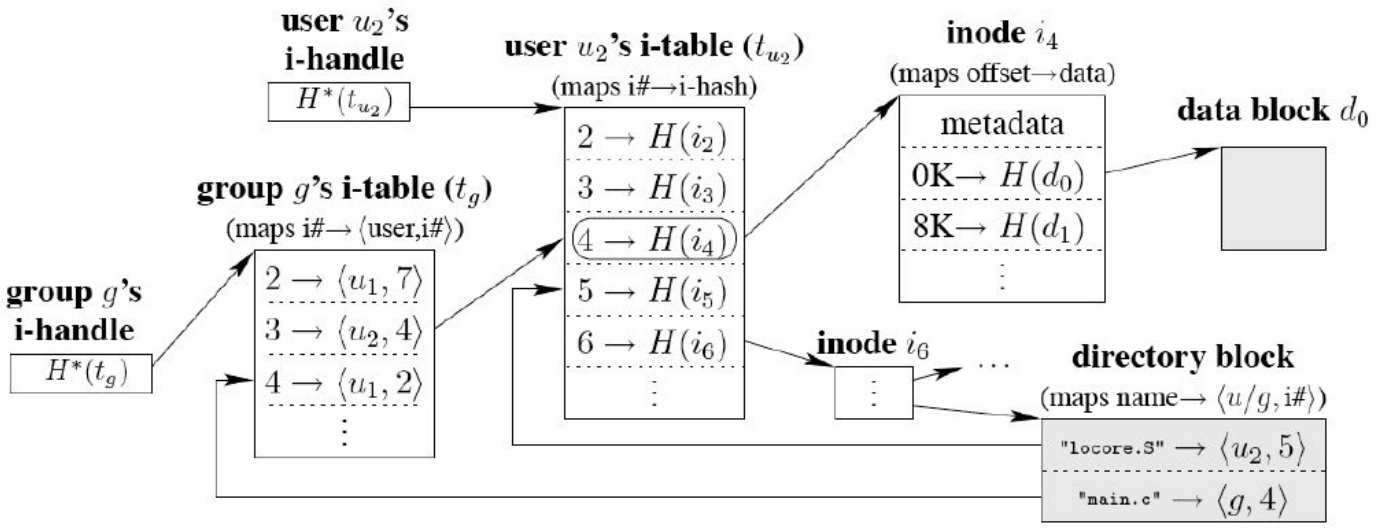


Secure Untrusted Data Repository(SUNDR)

- Tricks for SUNDR
  - Store file system as a hash tree
    - No need to reconstruct FS image from log
  - Use version vector (in version structure) to totally order operations
    - No need to fetch entire history to check for misbehavior

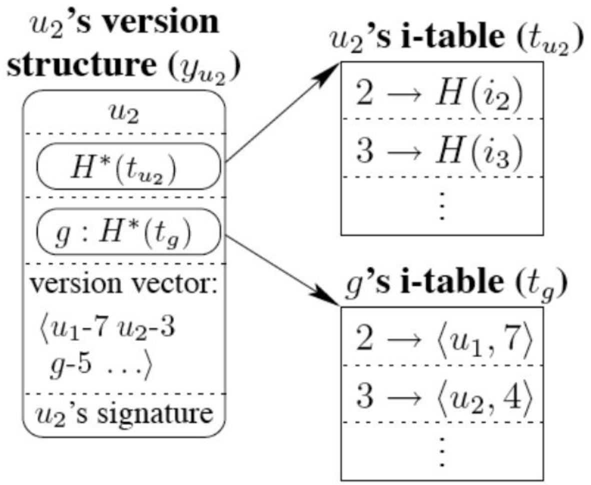
Hash Tree

- Store file system as a hash tree
  - Aggregate file state of user/group into single hash value using hash tree(i-handle)



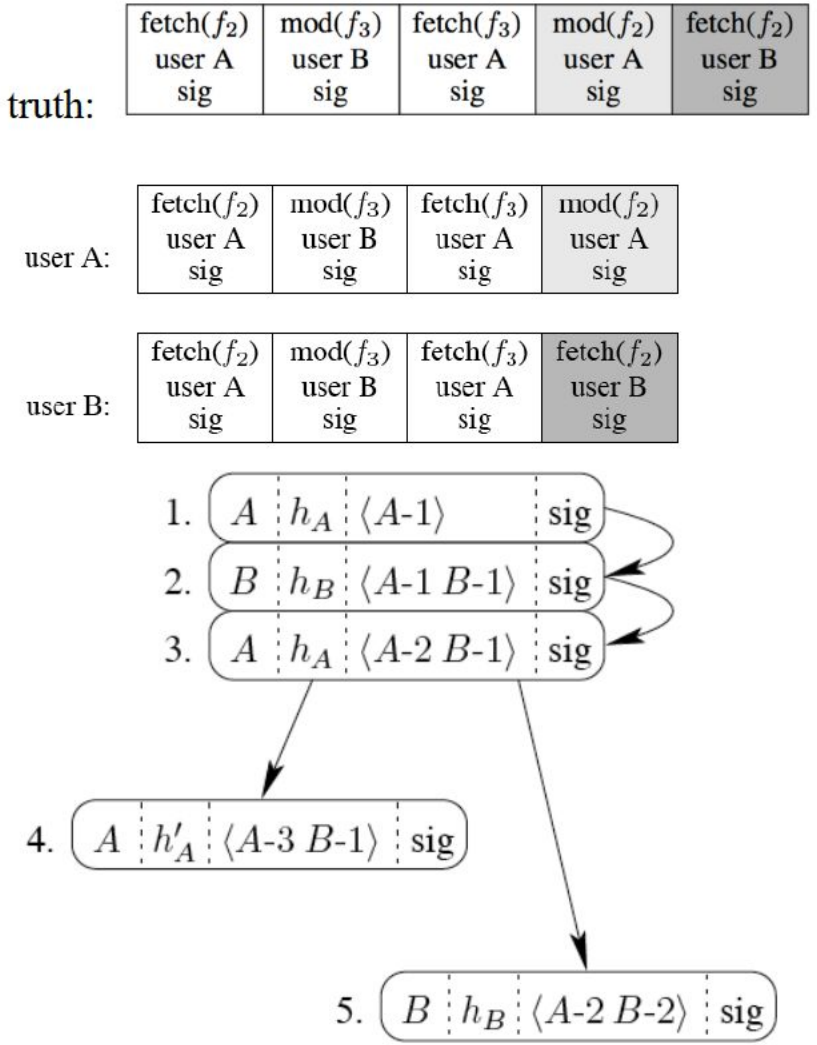
Version Vector

- Use version vector (in version structure) to totally order operations
  - Tie i-handles to each other using version vectors
  - Signatures are of user's ihandle and set of version vectors
  - Version vector: last known version vector for all files/groups in system
    - State of filesystem as it knew it


u_2's version structure (y_{u_2})
u_2
$H^*(t_{u_2})$
$g : H^*(t_g)$
version vector:
$\langle u_1\text{-}7\ u_2\text{-}3\ g\text{-}5\ \ldots\rangle$
u_2's signature
u_2's i-table (t_{u_2})
$2 \rightarrow H(i_2)$
$3 \rightarrow H(i_3)$
$\vdots$
g's i-table (t_g)
$2 \rightarrow \langle u_1, 7\rangle$
$3 \rightarrow \langle u_2, 4\rangle$
$\vdots$

Version Vector

- Server stores the (freshest) version vector
  - Version vector orders all operations
    - E.g. (0, 1, 2) ≤ (1, 1, 2)
- A client remembers the latest version vector given to it by the server
  - If new vector does not succeed old one, detect order violation!
    - E.g. (0,1,2) ≤ (0,2,1)



truth:

fetch(f_2) user A sig	mod(f_3) user B sig	fetch(f_3) user A sig	mod(f_2) user A sig	fetch(f_2) user B sig

user A:

fetch(f_2) user A sig	mod(f_3) user B sig	fetch(f_3) user A sig	mod(f_2) user A sig

user B:

fetch(f_2) user A sig	mod(f_3) user B sig	fetch(f_3) user A sig	fetch(f_2) user B sig

1. A h_A $\langle A\text{-}1 \rangle$ sig
2. B h_B $\langle A\text{-}1\ B\text{-}1 \rangle$ sig
3. A h_A $\langle A\text{-}2\ B\text{-}1 \rangle$ sig
4. A h'_A $\langle A\text{-}3\ B\text{-}1 \rangle$ sig
5. B h_B $\langle A\text{-}2\ B\text{-}2 \rangle$ sig

Conclusion

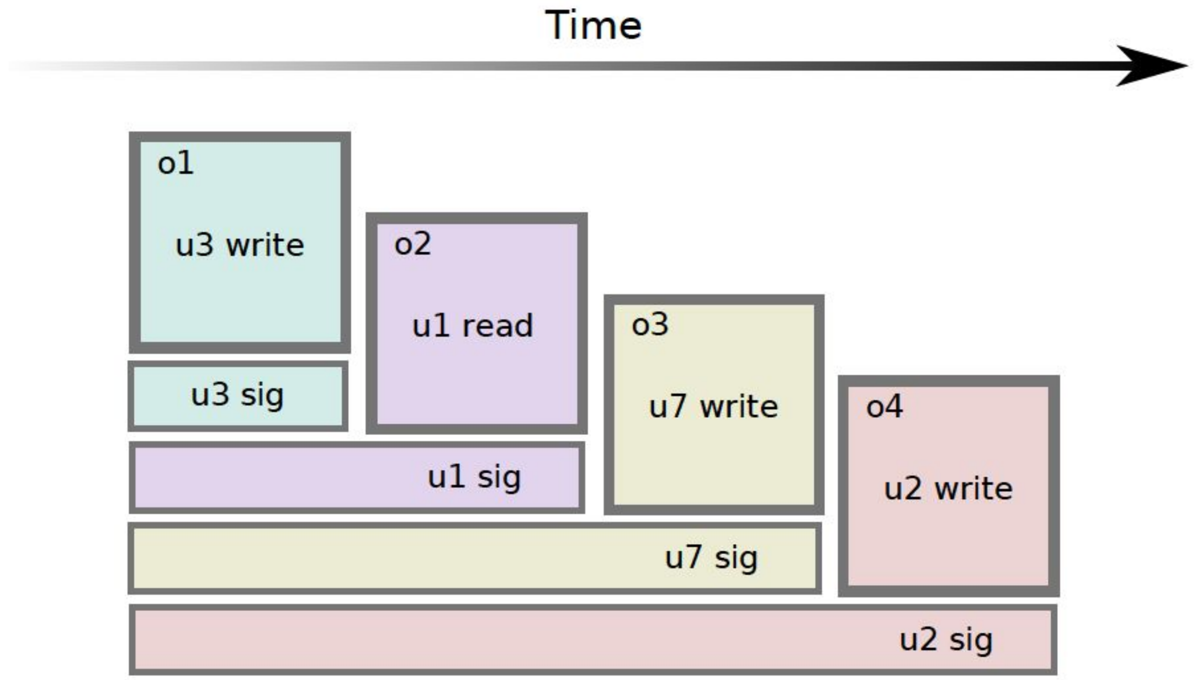
- Practical Byzantine Fault Tolerance
  - Rely on a super-majority of votes to make the system reliable.
- Needham-Schroeder, Kerberos
  - Combine authentication with Key Distribution Center to secure communication.
- SUNDR
  - Provide file system integrity with untrusted servers. Can detect bad servers

Thank You

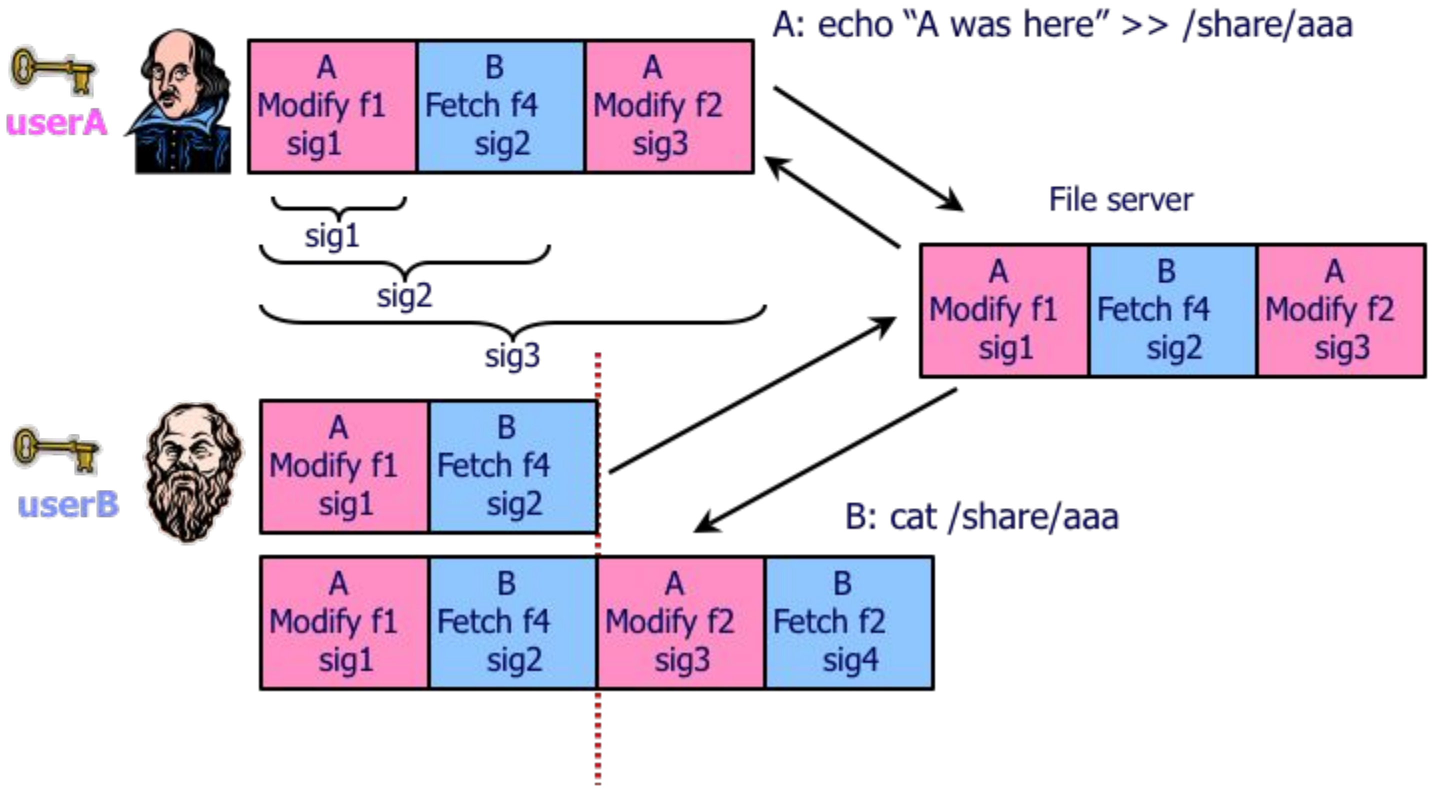
Reference

- Security
  - https://www.cse.buffalo.edu//~stevko/courses/cse486/spring15/lectures/34-security1.pdf
  - https://www.cse.buffalo.edu//~stevko/courses/cse486/spring15/lectures/35-security2.pdf
  - https://www.cs.cmu.edu/~dga/15-440/F10/lectures/sec-2.pdf
- https://courses.cs.washington.edu/courses/csep552/13sp/lectures/10/sundr.pdf
- http://w.scs.stanford.edu/jinyuan/sundr-osdi.ppt
-

The Signature Coverage

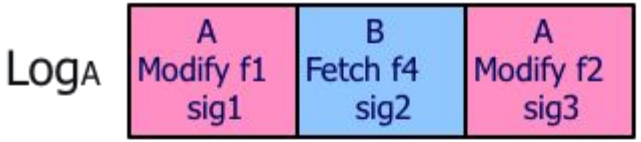


Fork Consistency



Fork Consistency

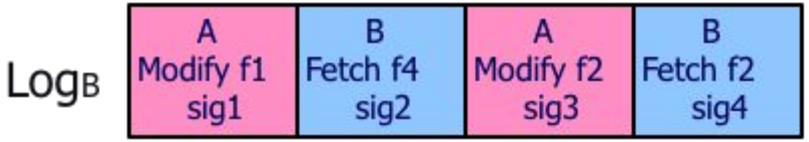
A's latest log:



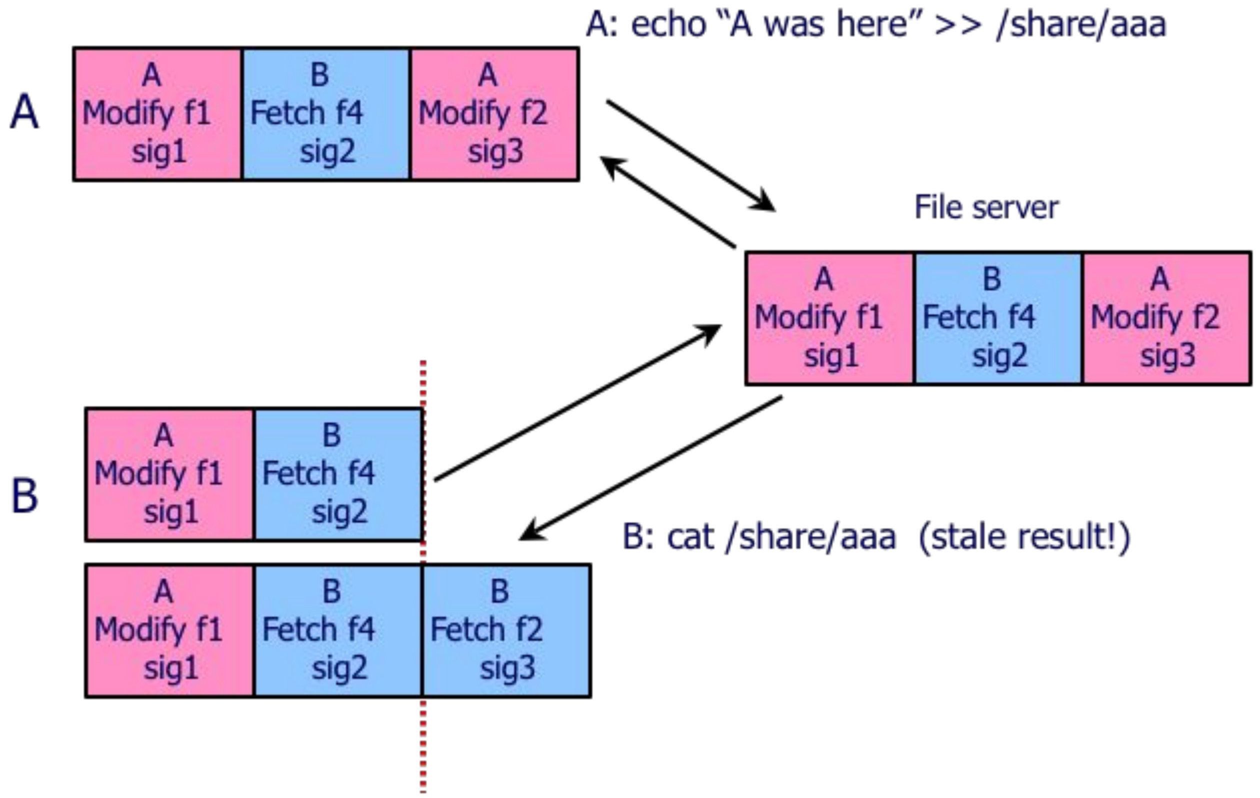
The total order:

Log_A ≤ Log_B iff Log_A is prefix of Log_B

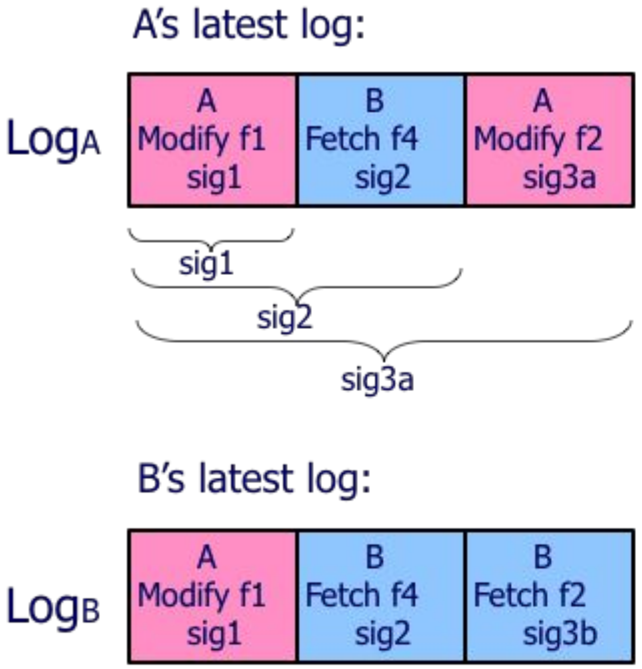
B's latest log:



Detecting attacks by the server



Detecting attacks by the server



A's log and B's log can no longer be ordered:
$\text{Log}_A \not\trianglelefteq \text{Log}_B$, $\text{Log}_B \not\trianglelefteq \text{Log}_A$