SPORC

Group Collaboration using Untrusted Cloud Resources

Ariel J. Feldman, William P. Zeller,
Michael J. Freedman, Edward W. Felten

PRINCETON UNIVERSITY

Cloud deployment: pro & con

For user-facing applications:

(e.g. word processing, calendaring, e-mail, IM)

Google docs

Google calendar

Microsoft Office Live

⋮

Cloud deployment is attractive

- Scalable, highly available, globally accessible
- Real-time collaboration

But, there's a price…

Must trust the cloud provider for confidentiality and integrity

SPORC goals

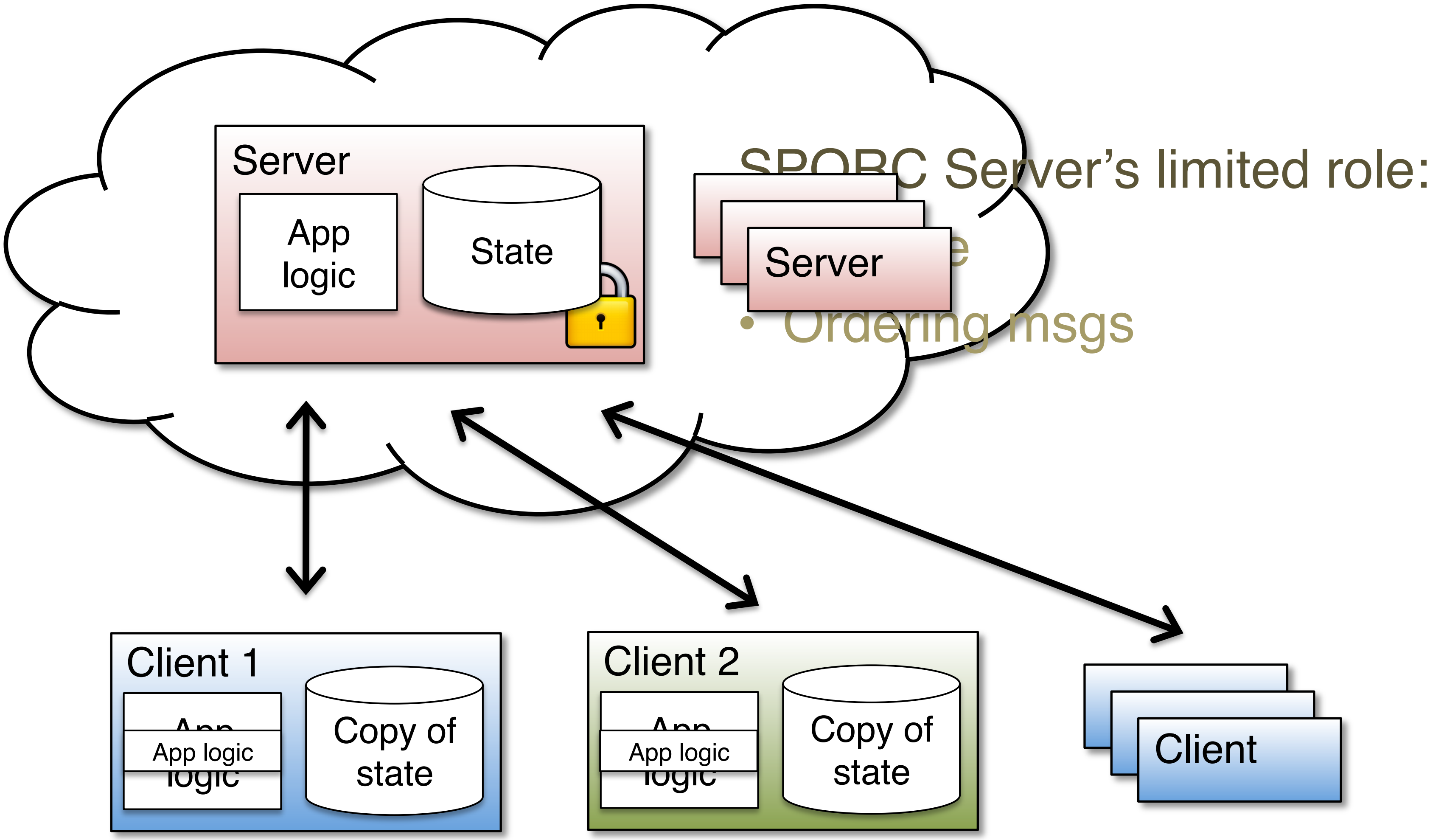
Practical cloud apps

- Flexible framework
- Real-time collaboration
- Work offline

Untrusted servers

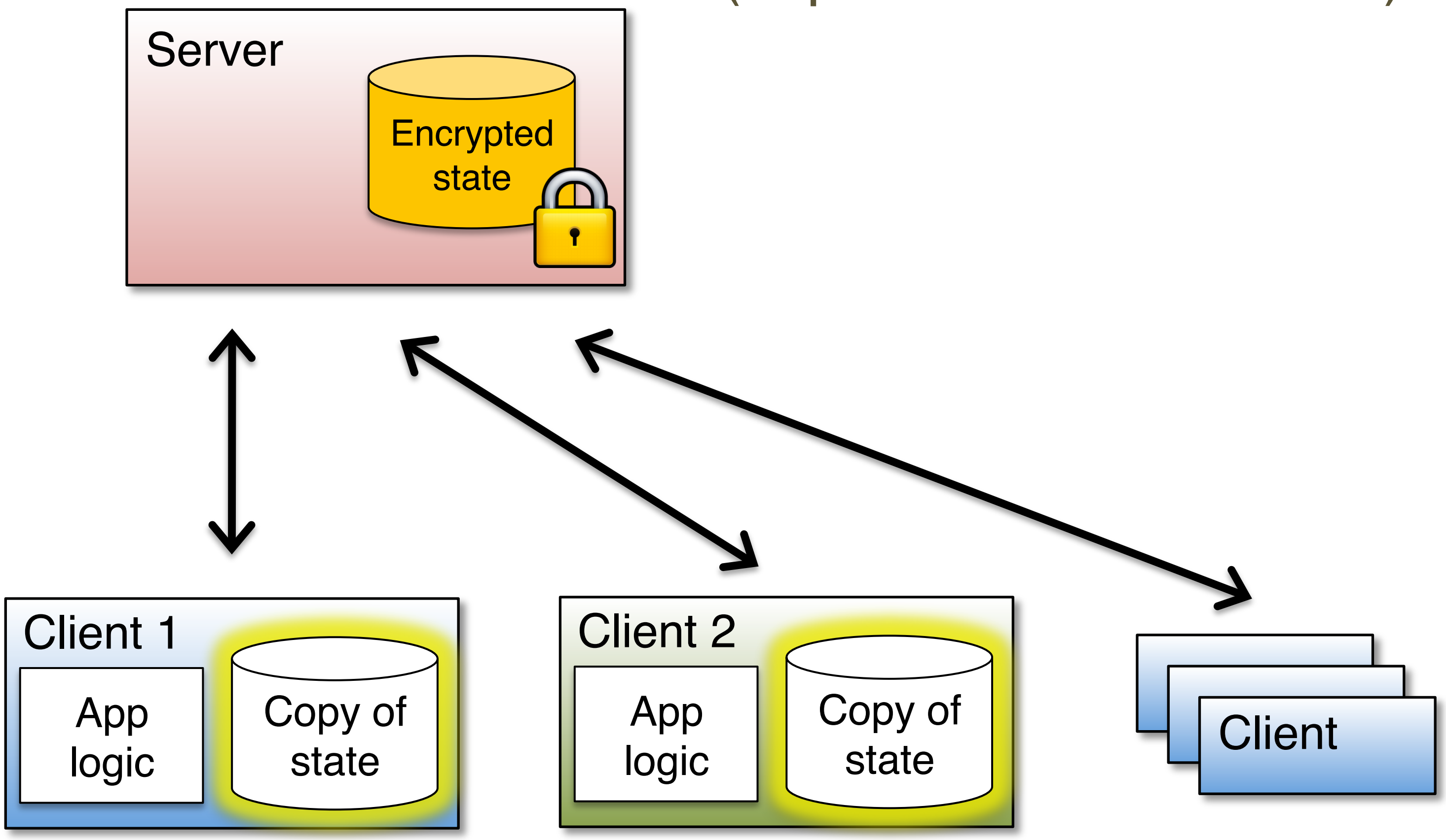
- Can't read user data
- Can't tamper with user data without risking detection
- Clients can recover from tampering

Making servers untrusted

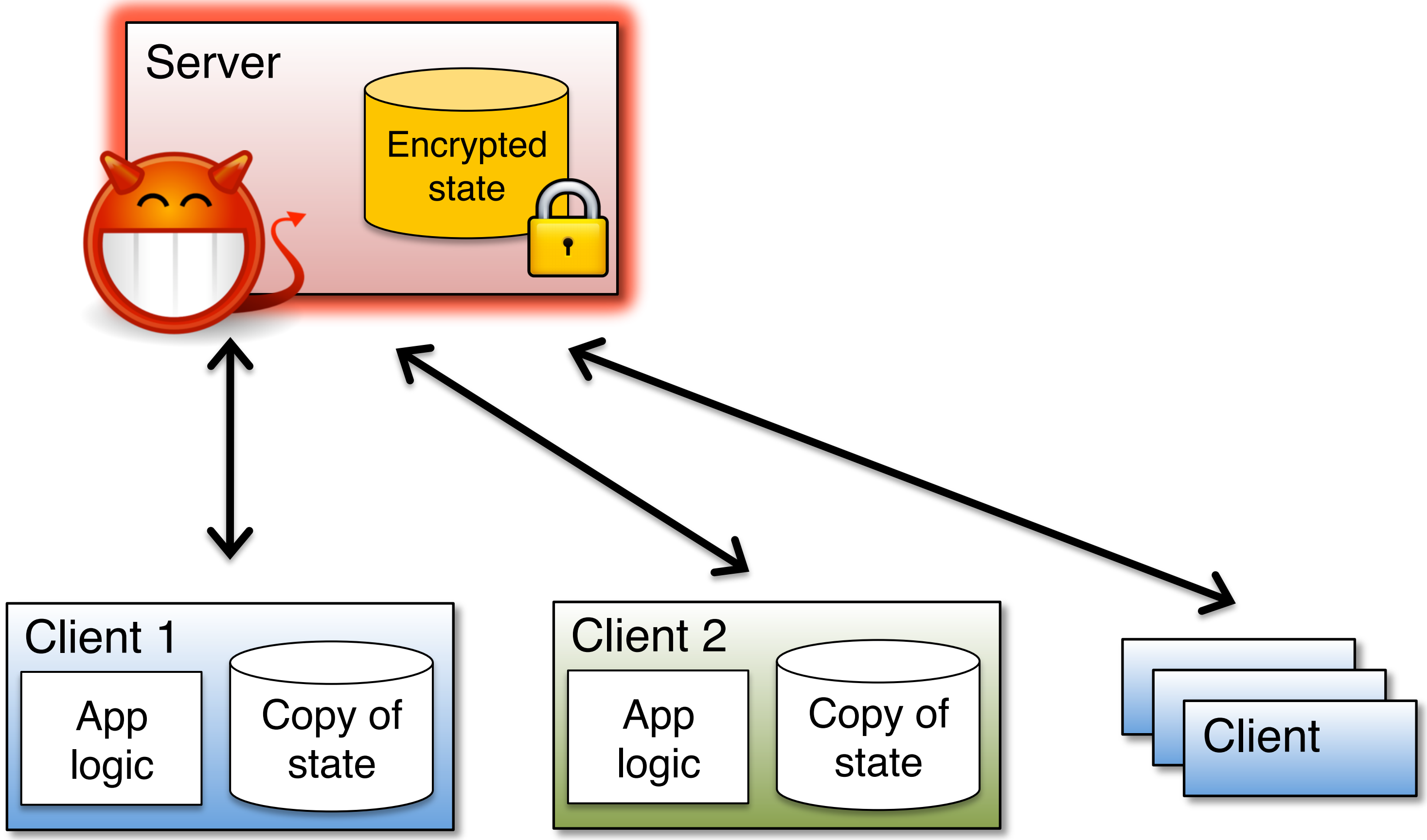


Problem #1: How do you keep clients' local copies consistent?

(esp. with offline access)



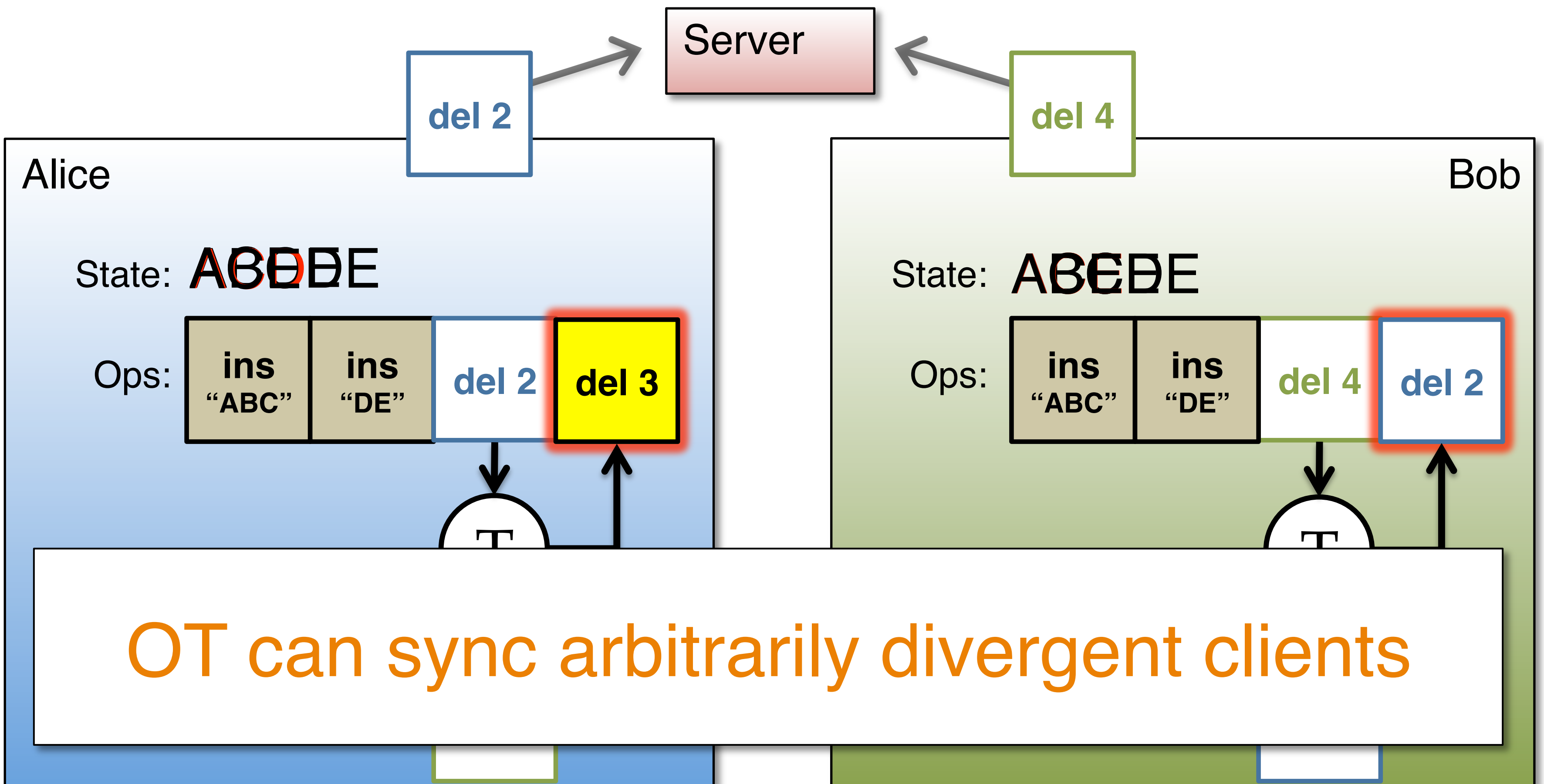
Problem #2: How do you deal with a malicious server?



Keeping clients in sync

Operational transformation (OT) [EG89]

(Used in Google Docs, EtherPad, etc.)



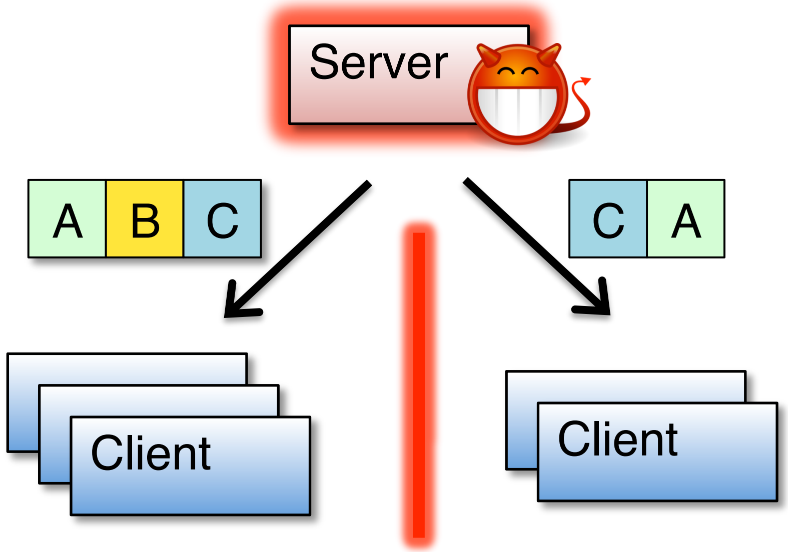
Dealing with a malicious server

Digital signatures aren't enough

Server can equivocate

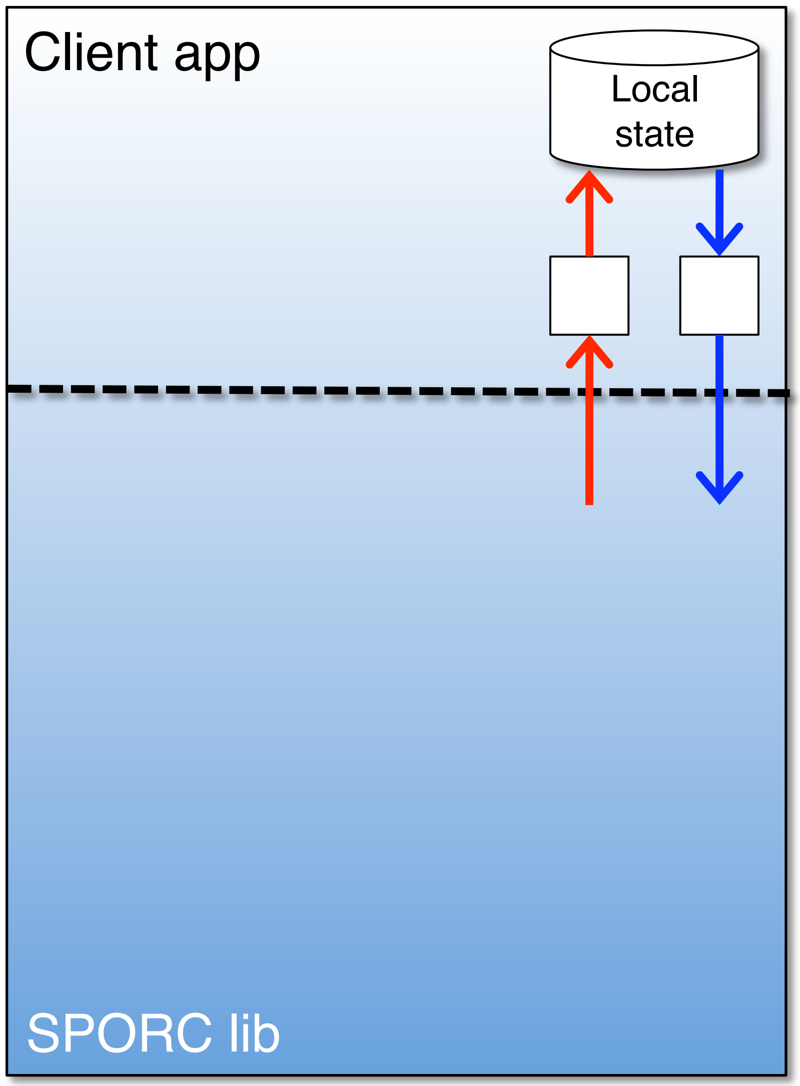
fork* consistency [LM07]

- Honest server: linearizability
- Malicious server: Alice and Bob detect equivocation after exchanging 2 messages
- Embed history hash in every message

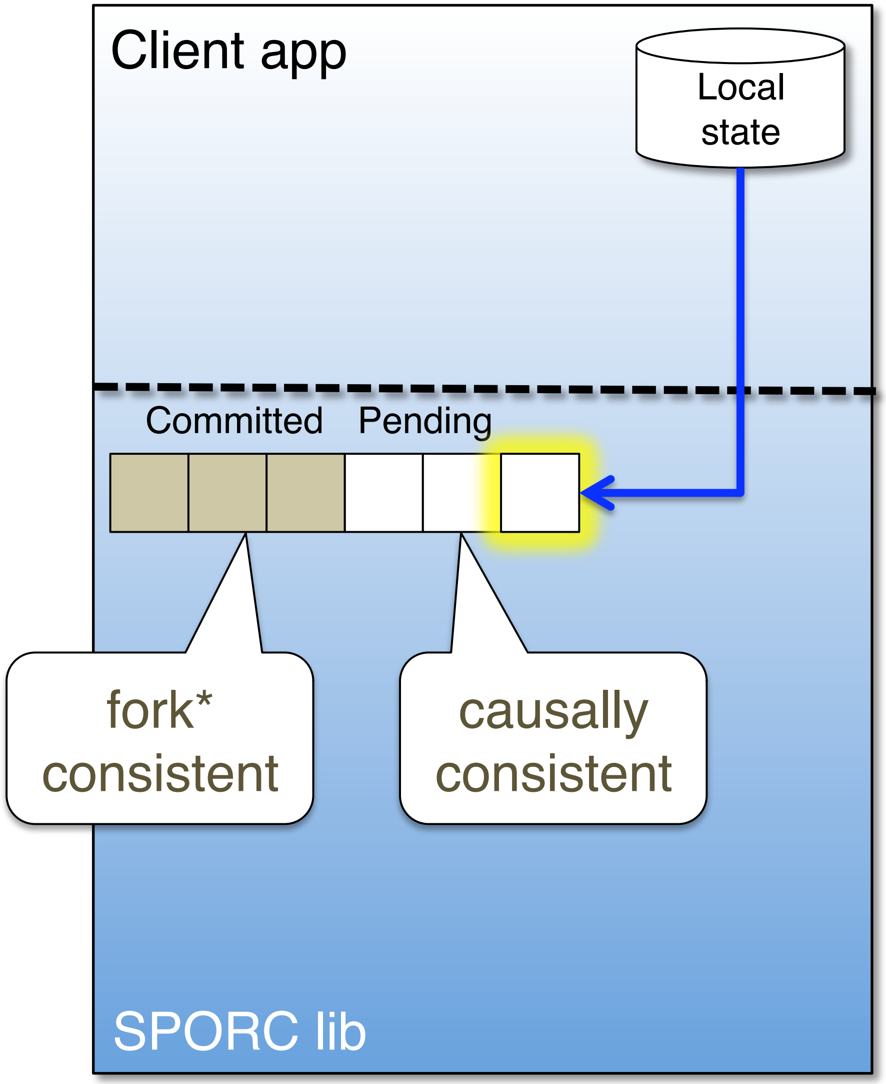


Server can still fork the clients, but can't unfork

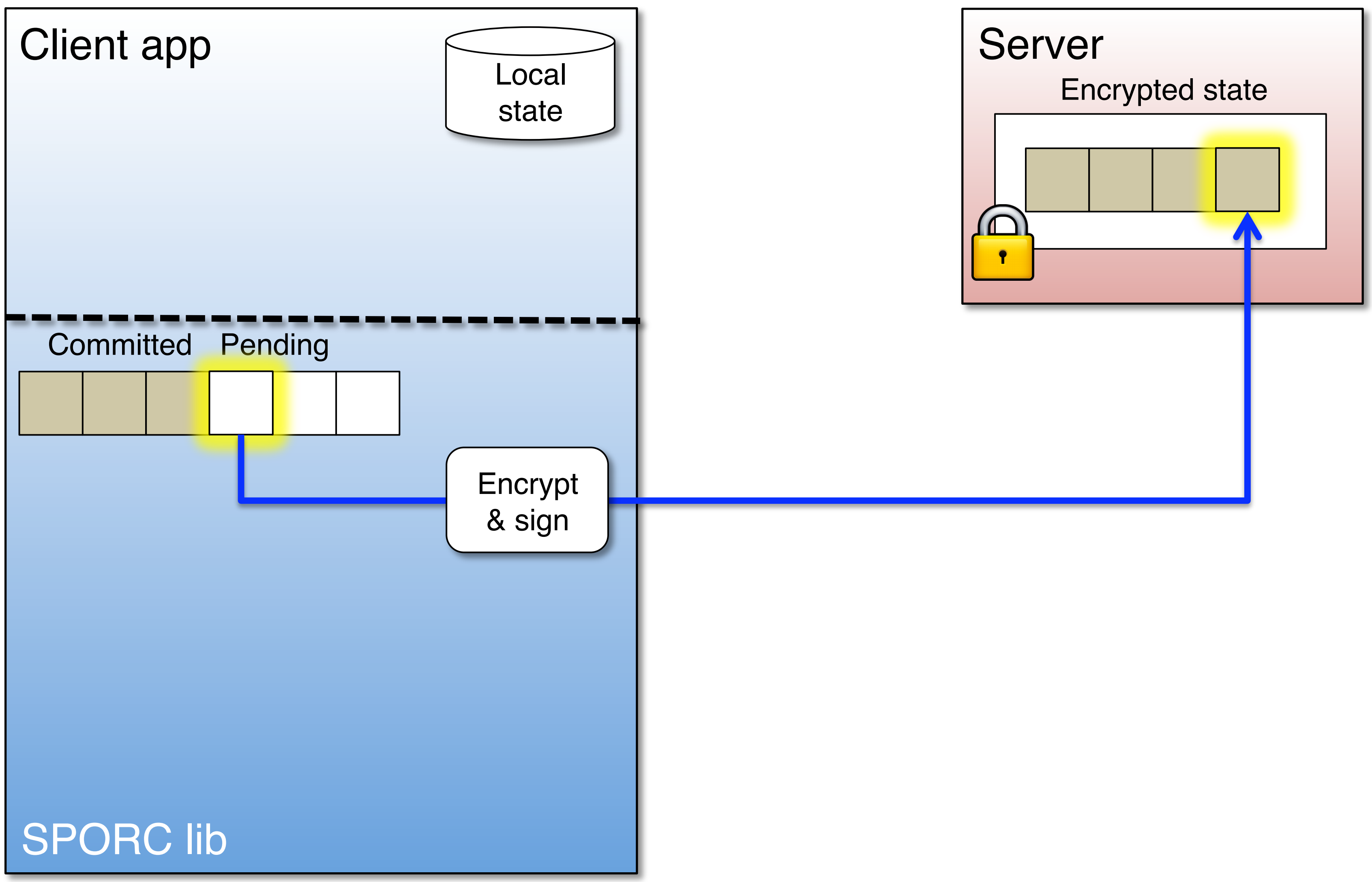
System design



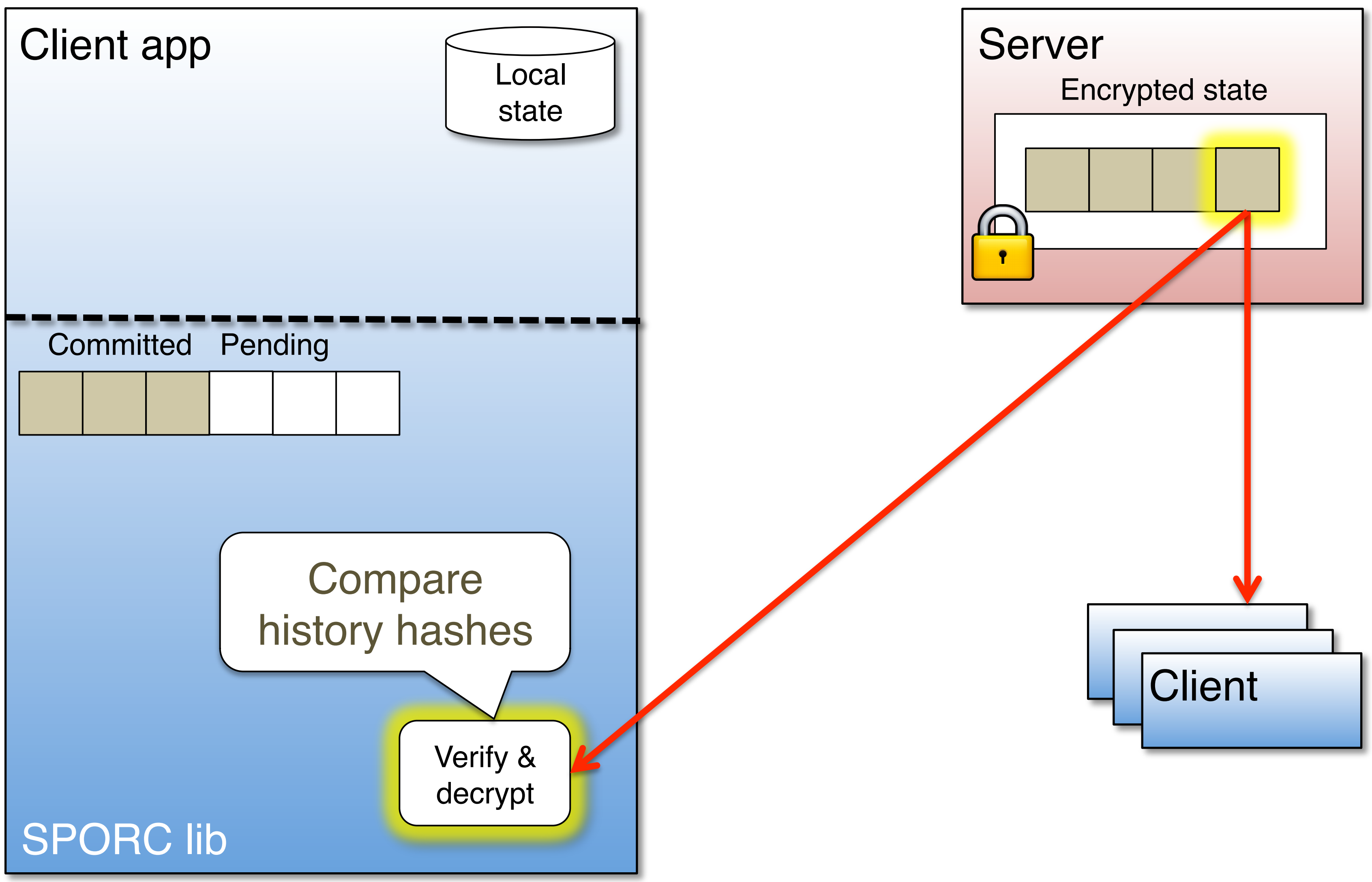
System design



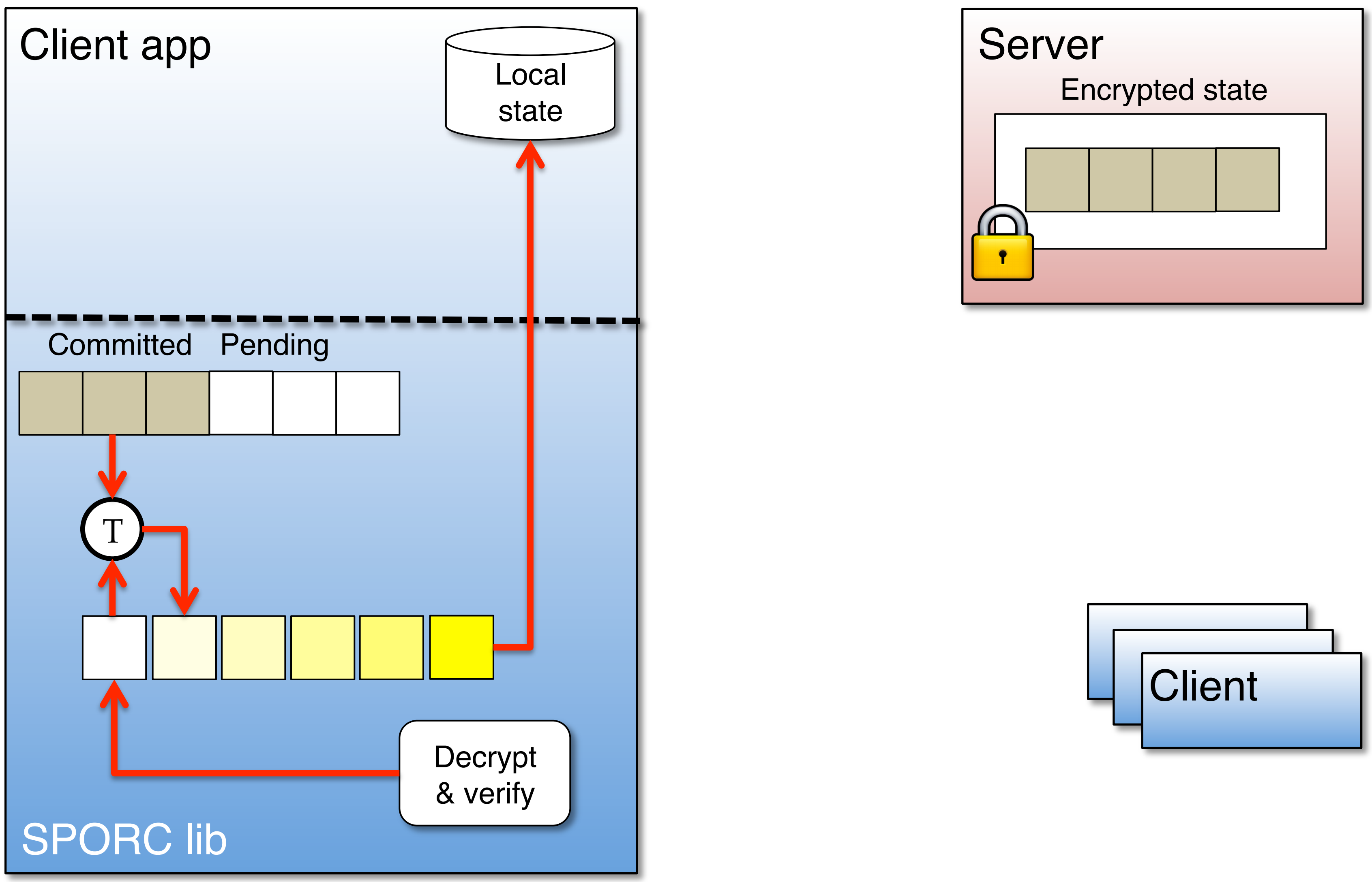
System design



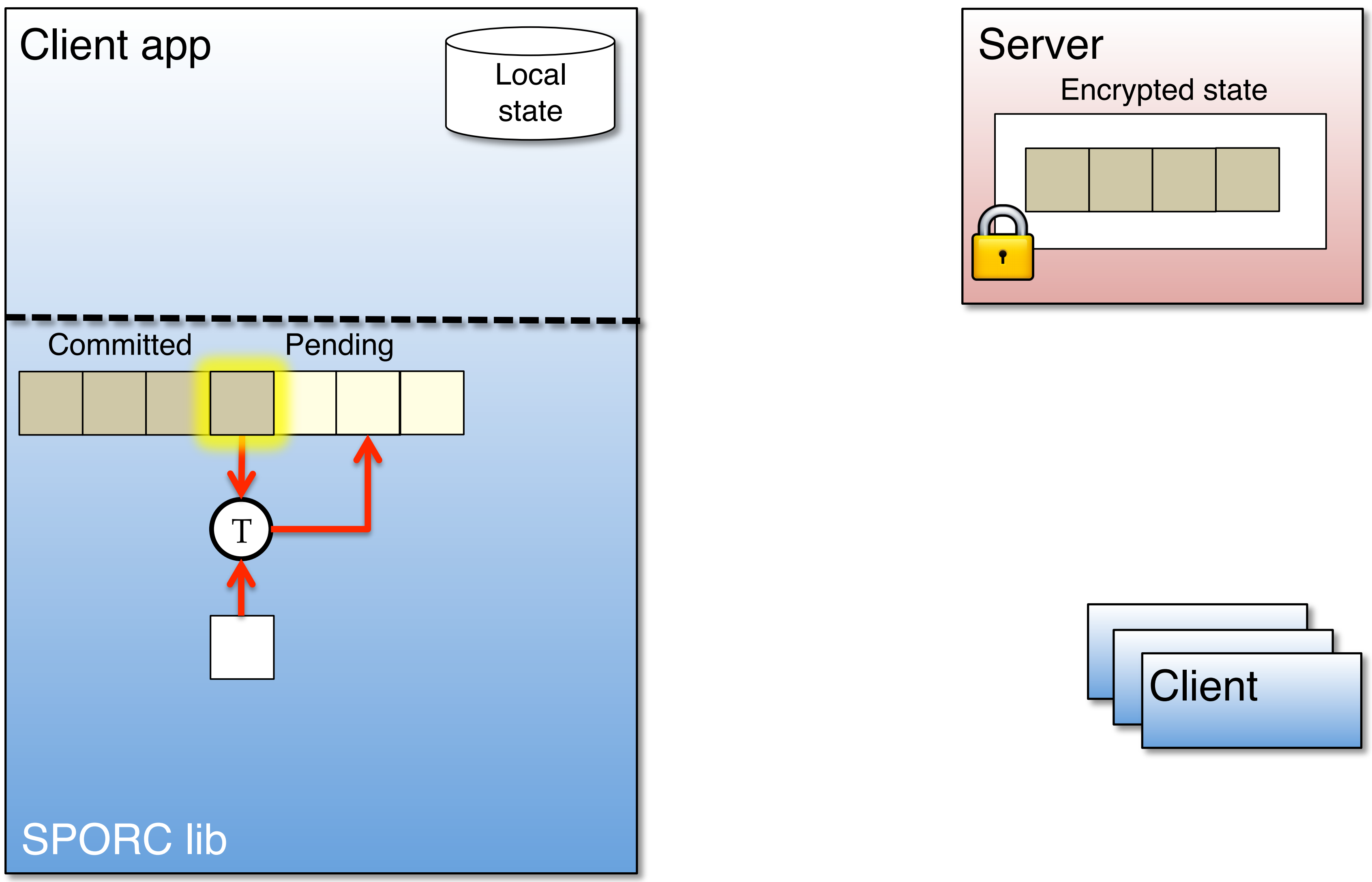
System design



System design



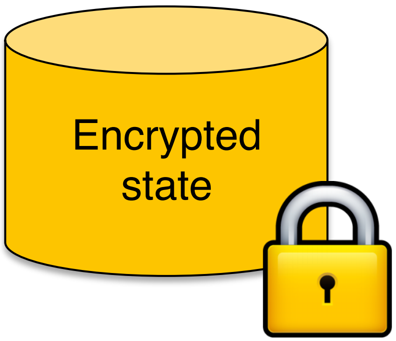
System design



Access control

Challenges

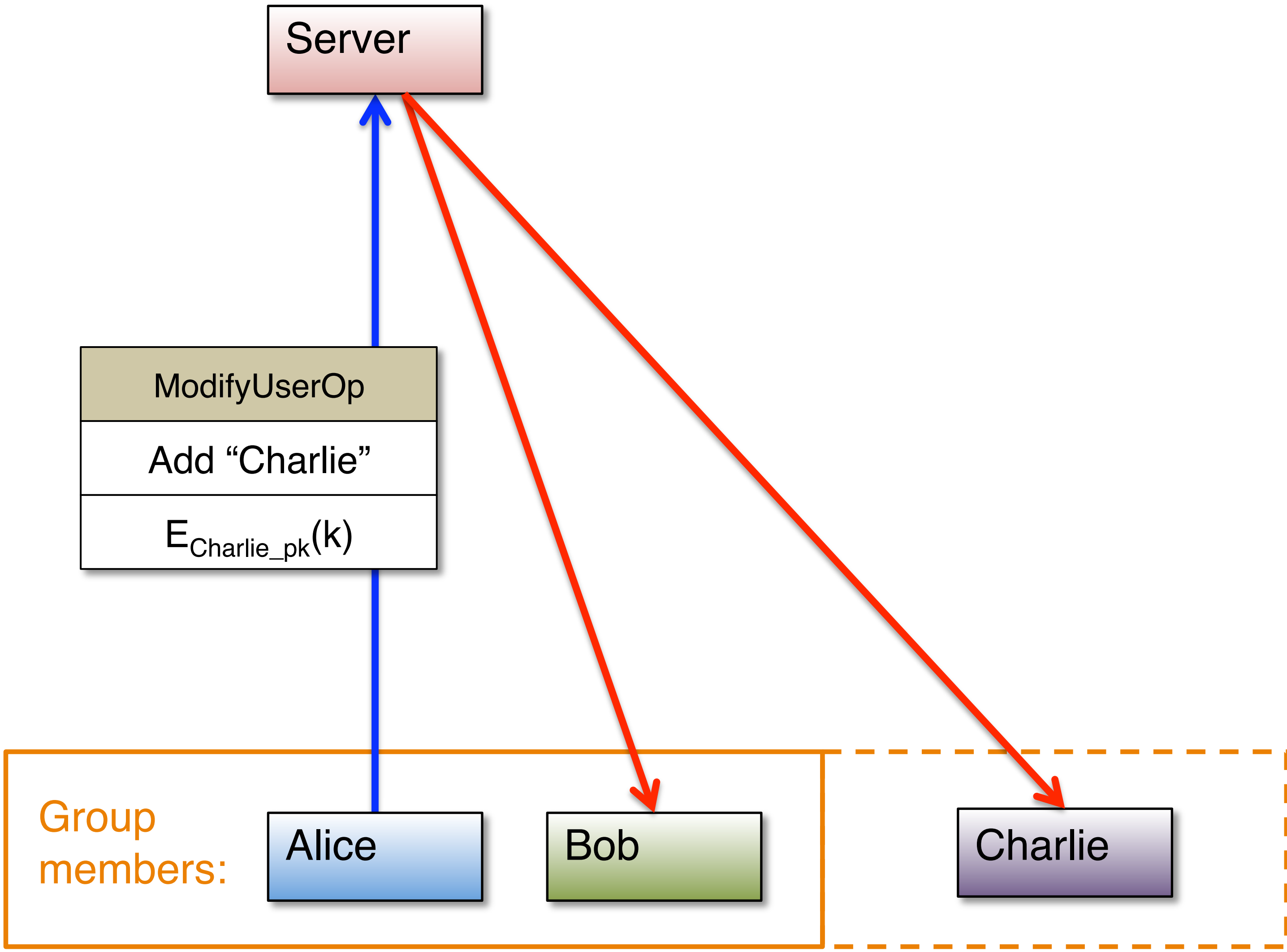
- Server can't do it — it's untrusted!
- Preserving causality
- Concurrency makes it harder



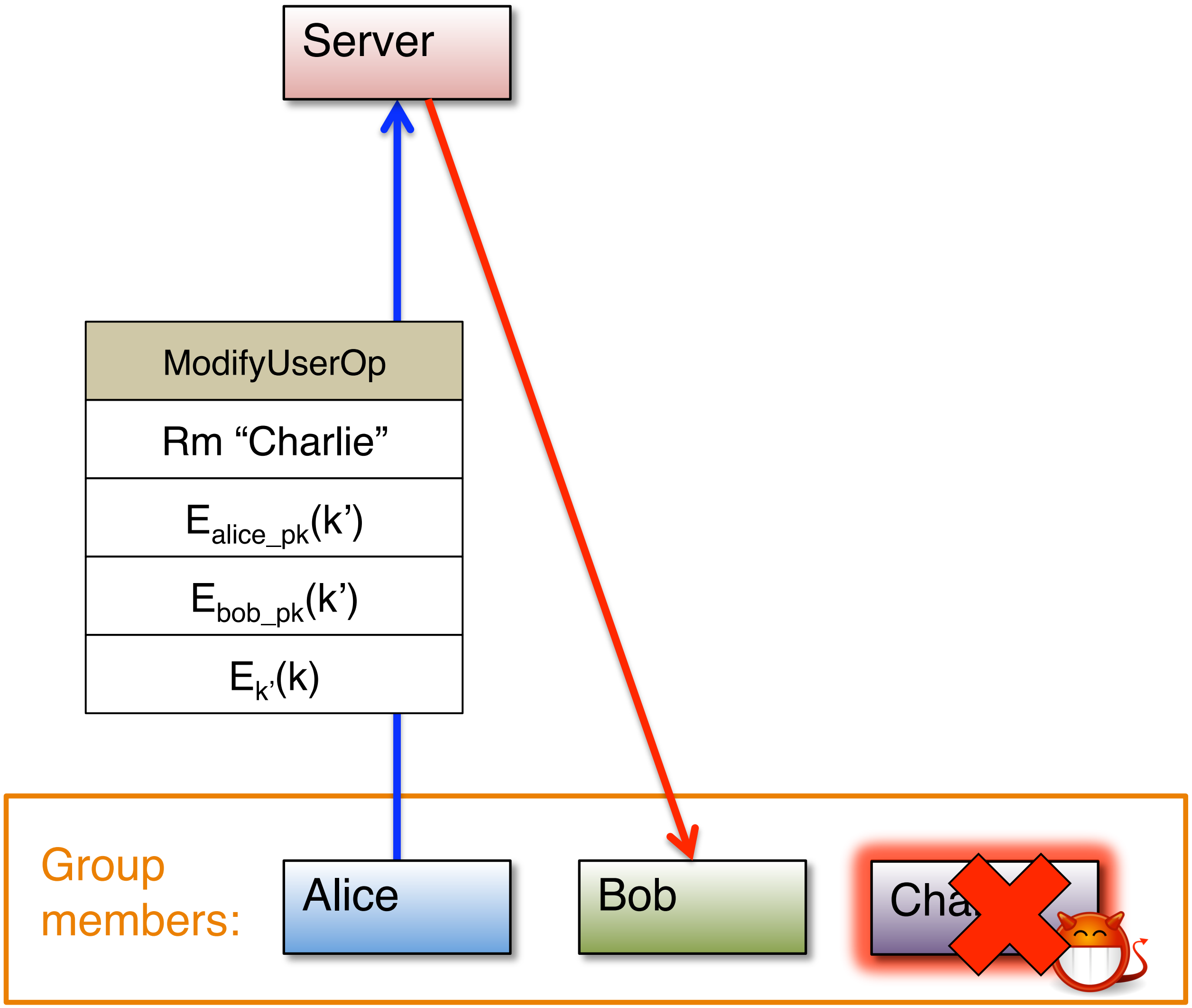
Solutions

- Ops encrypted with symmetric key shared by clients
- ACL changes are ops too
- Concurrent ACL changes handled with barriers

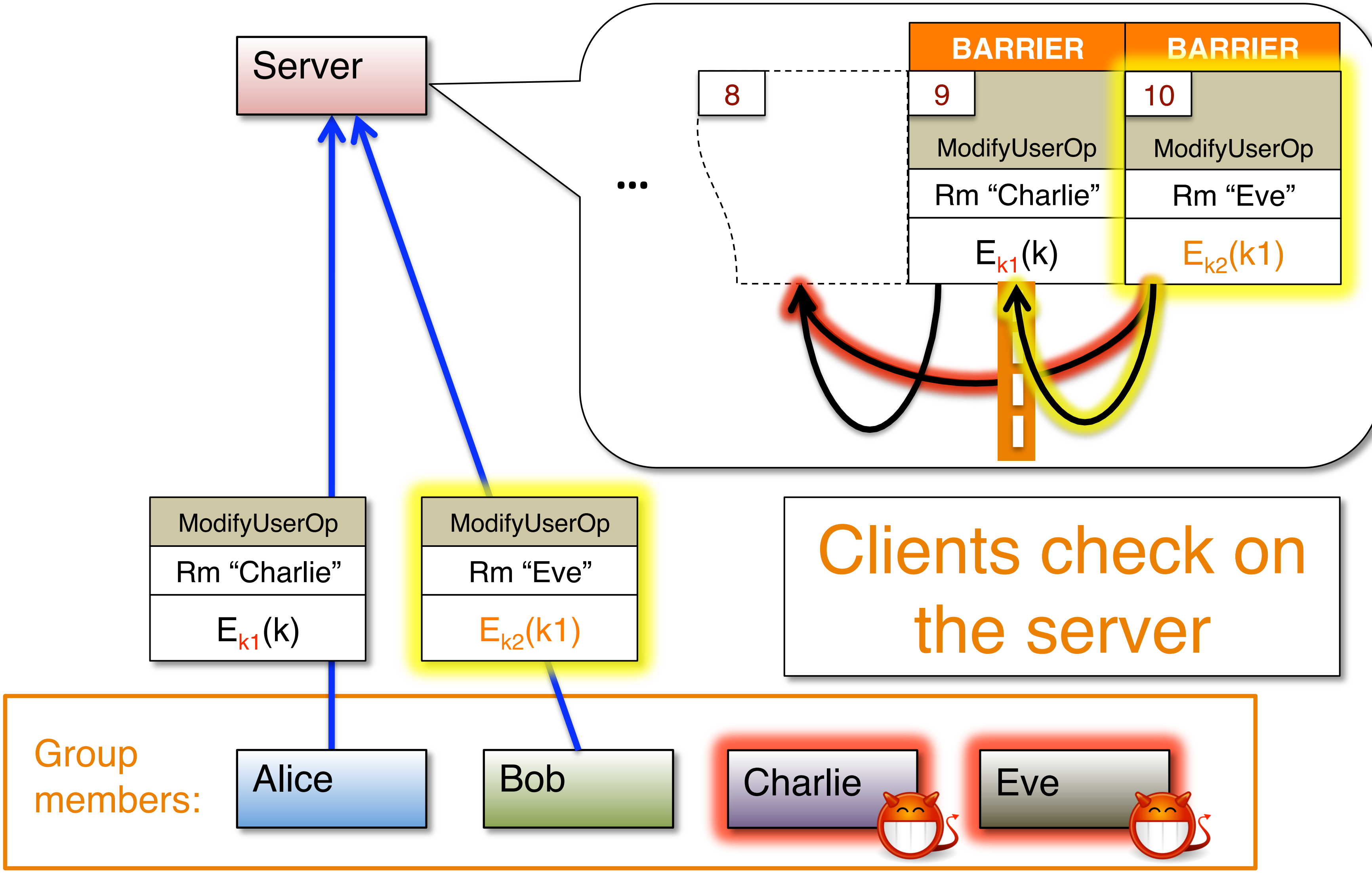
Adding a user



Removing a user

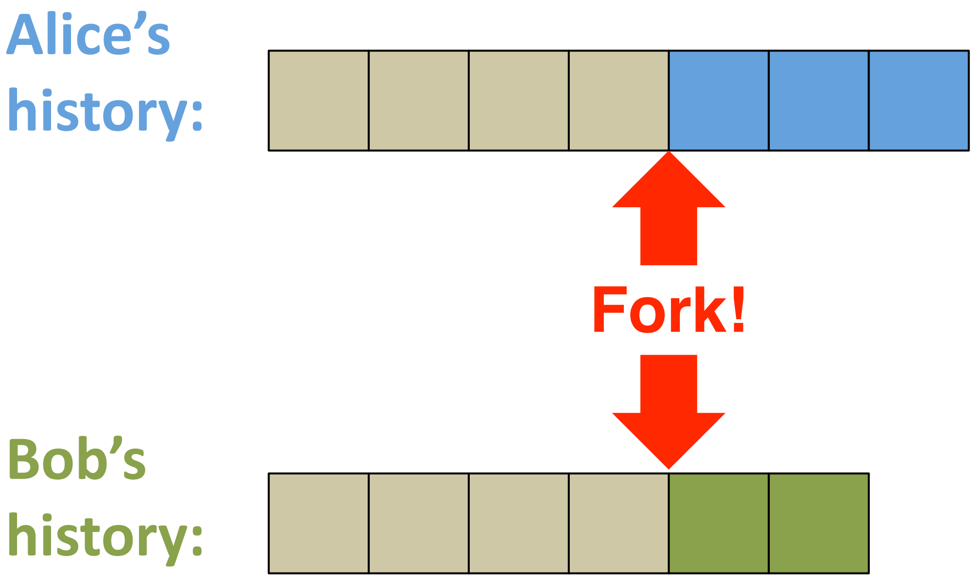


Barriers: dealing with concurrency



Clients check on the server

Recovering from a fork



Can use OT to resolve malicious forks too

Implementation

Client lib + generic server

App devs only need to define ops and provide a transformation function

Java CLI version + browser-based version (GWT)

Demo apps: key value store, browser-based collaborative text editor

Evaluation

Setup

- Tested Java CLI version
- 8-core 2.3 GHz AMD machines
  - 1 for server
  - 4 for clients (often >1 instance per machine)
- Gigabit LAN

Microbenchmarks

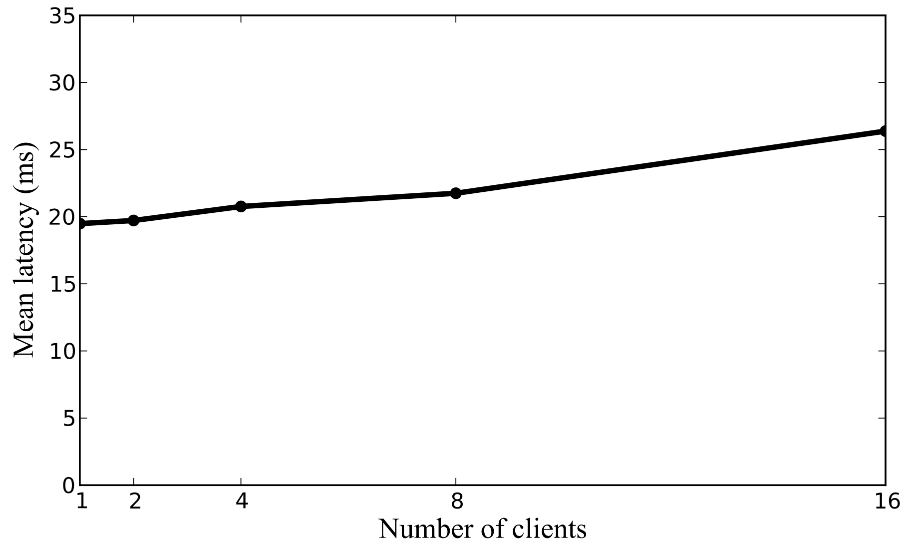
- Latency
- Server throughput
- Time-to-join (in paper)

Latency

(Text editor app)

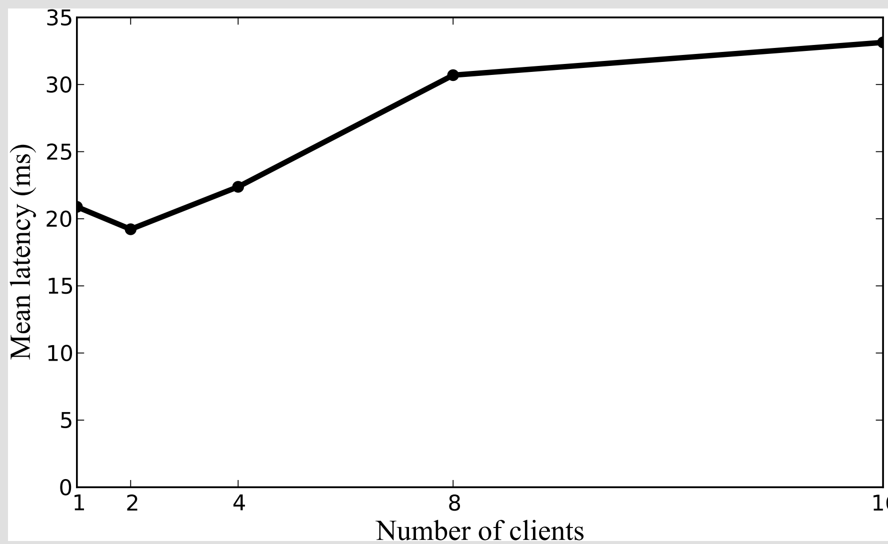
Low load

(1 client writer)



High load

(all clients are writers)

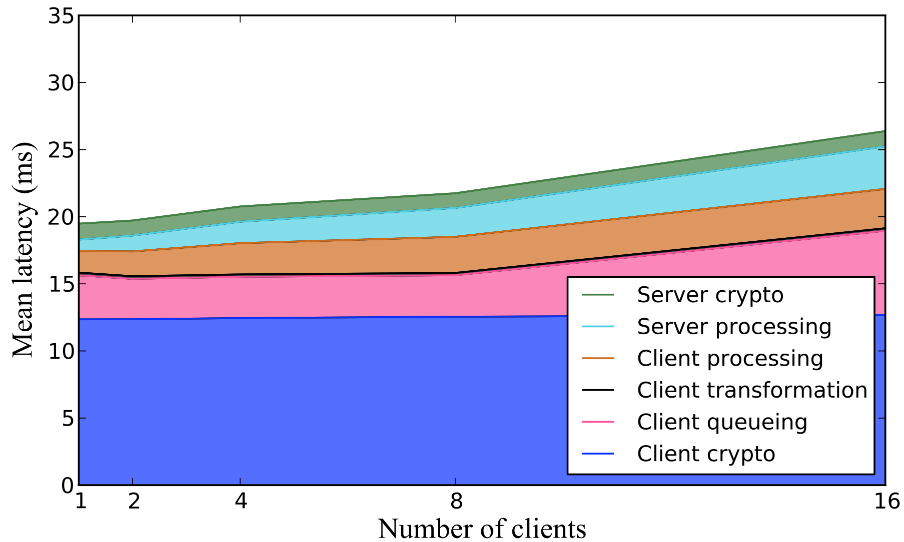


Latency

(Text editor app)

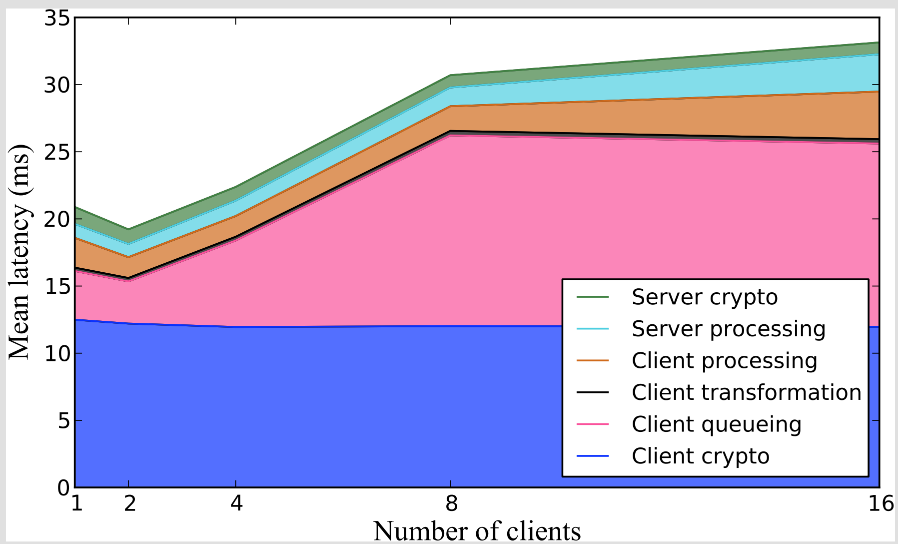
Low load

(1 client writer)

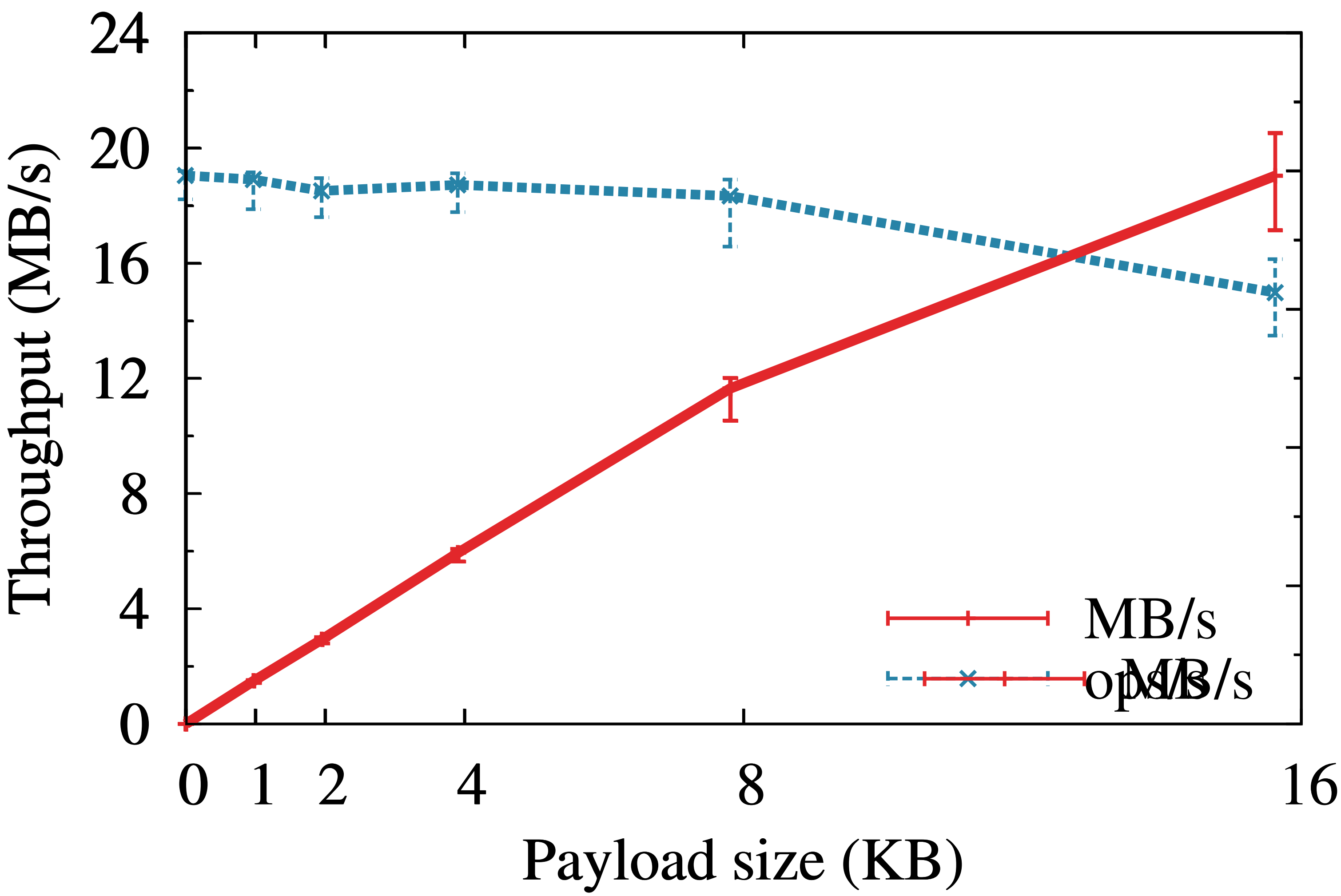


High load

(all clients are writers)



Server throughput



Conclusion

Practical cloud apps + untrusted servers

Operational transformation + fork* consistency

Dynamic access control and key distribution

Recovery from malicious forks

*

Thank you

Questions?

ajfeldma@cs.princeton.edu

*http://www.snowpeak.com/tableware/cutlery/titanium-original-spork-sct-004.html

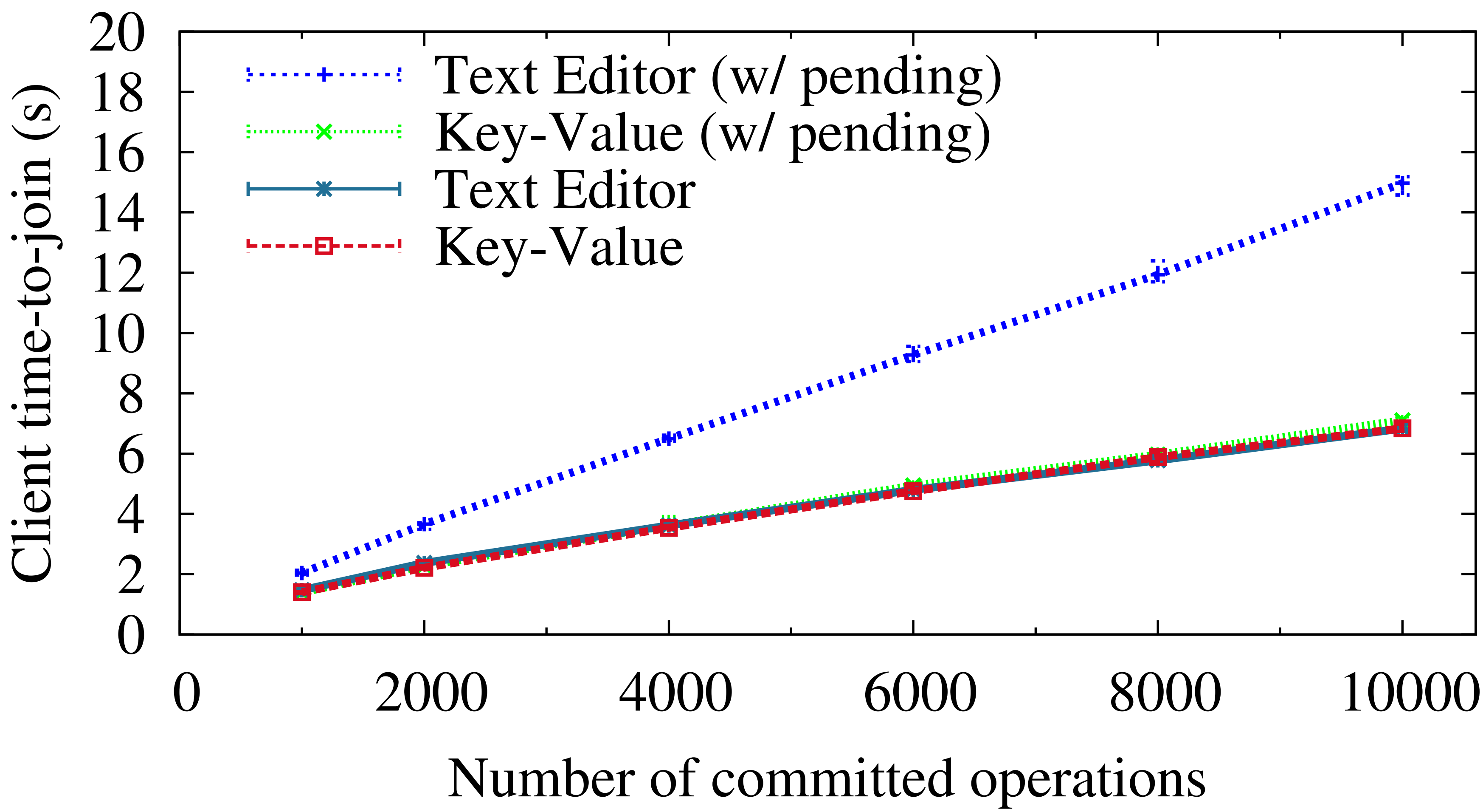
Comparison with Depot

	SPORC	Depot
Consistency with malicious servers	✔	✔
Consistency with malicious clients		✔
Fork recovery	✔	✔
Work offline	✔	✔
Dynamic access control	✔	
Confidentiality and key distribution	✔	

Depot exposes conflicts, but leaves it to the app to resolve them

Future work: SPORC + Depot? ;-)

Time-to-join



Text Editor (w/ pending)
Key-Value (w/ pending)
Text Editor
Key-Value
Client time-to-join (s)
0
2
4
6
8
10
12
14
16
18
20
0
2000
4000
6000
8000
10000
Number of committed operations