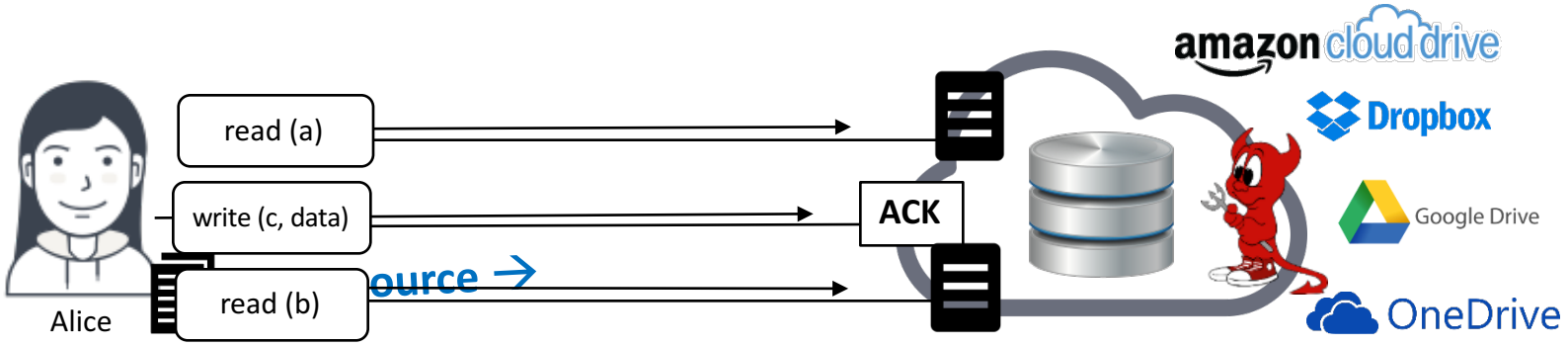
TaoStore
Overcoming Asynchronicity in Oblivious Data Storage

CETIN SAHIN, VICTOR ZAKHARY, AMR EL ABBADI, HUIJIA (RACHEL) LIN, STEFANO TESSARO

UNIVERSITY OF CALIFORNIA, SANTA BARBARA
IEEE SECURITY AND PRIVACY, 2016

Outsourced Private Data



Security Concerns?

Confidentiality of Data

Encryption

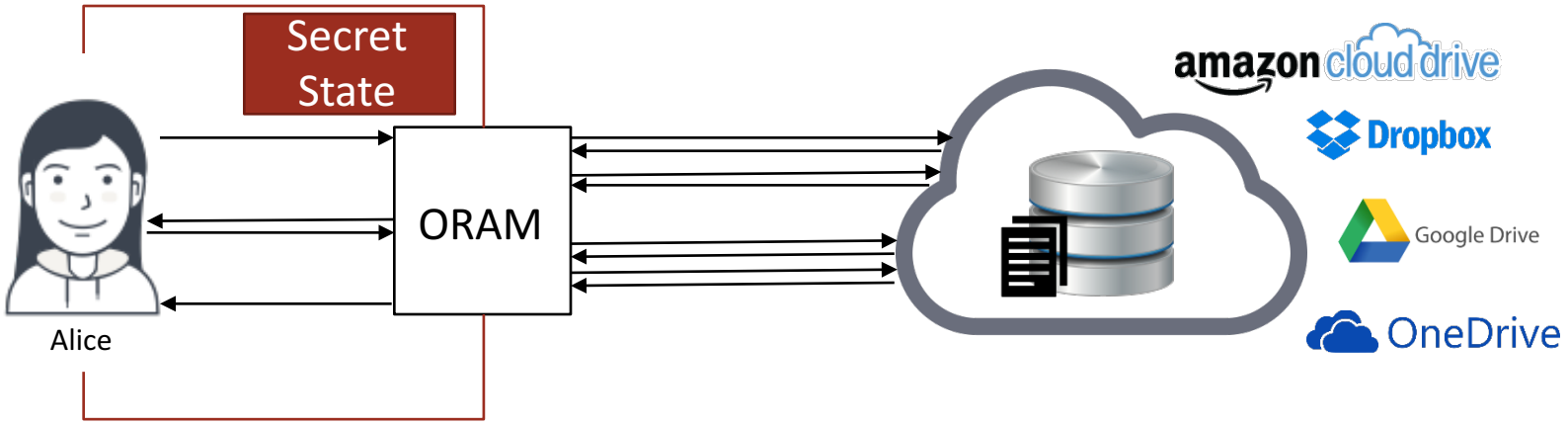
Is encryption enough?

Encryption alone is not enough!!!

Access patterns can leak sensitive information
[Islam et al. NDSS'12]

read(1), read(1) vs read(1), write(3)

Outsourced Private Data



Goal: Oblivious Access

Translate each logical access
to a sequence of random-looking accesses

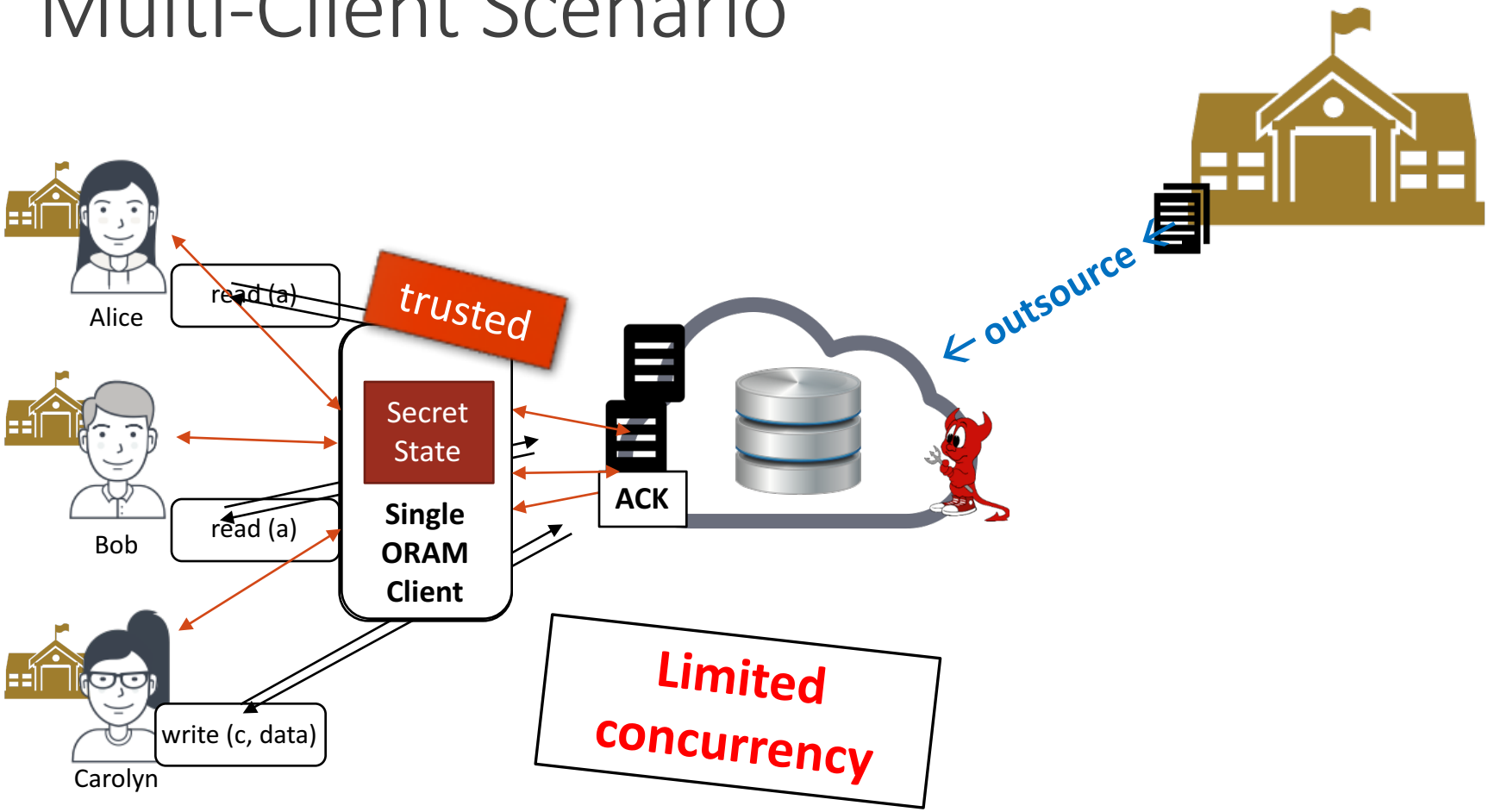
OBLIVIOUS RAM (ORAM)

Goldreich and Ostrovsky '96

More practical solutions: MG'11, DB'11, ES'11, EK'12, ES'12, PW'12, ES'13, CG'13, KC'13, KC'14, LR'15, TM'15 , SD'16, …

Single client

Multi-Client Scenario



PrivateFS [Williams et al. CCS'12]

ObliviStore [Stefanov et al. Oakland'13]

CURIOUS [Bindschaedler et al. CCS'15]

Concurrency

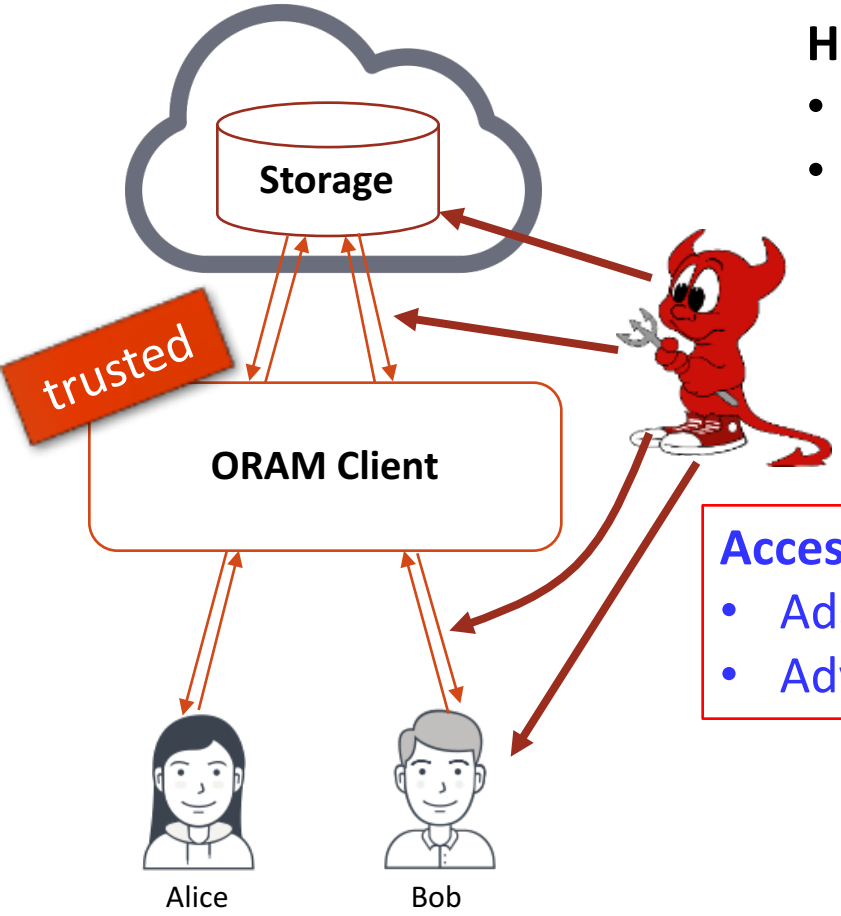
Asynchronicity

Contributions

A security model for asynchronous ORAM and attack

TaoStore: A new asynchronous and concurrent tree-based oblivious storage

Asynchronous ORAM – Threat Model



Honest-but-curious adversary

- Sees raw storage data
- Network communication

Asynchronous links

- Adversarially controlled schedule

Access operations

- Adaptively scheduled by adversary
- Adversary learns response timing

New

Important:

- Network Intruder
- Adaptive Scheduling
- Side Channel

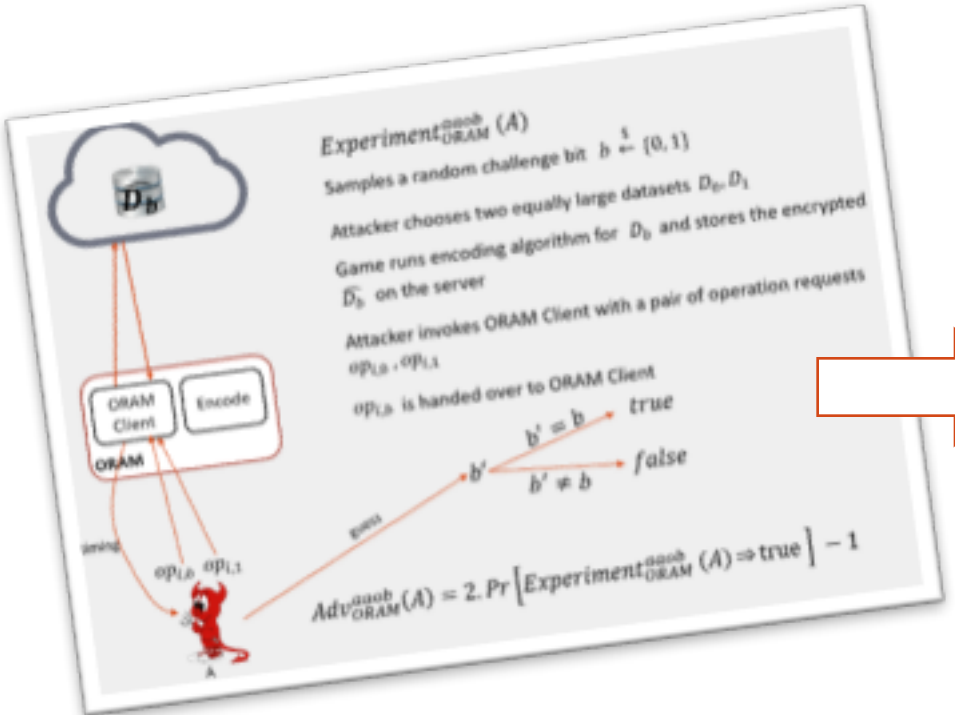
ObliviStore [Stefanov et al'13]
non-adaptive scheduling + no response time

Asynchronous ORAM - Security

We formalize obliviousness in this setting

Two timing-consistent executions should be indistinguishable in threat model

aaob-security: adaptive asynchronous obliviousness



Indistinguishability-based security definition.

See the paper!

Are existing systems aaob-secure?

ObliviStore is not secure [Bindschaedler et al.]

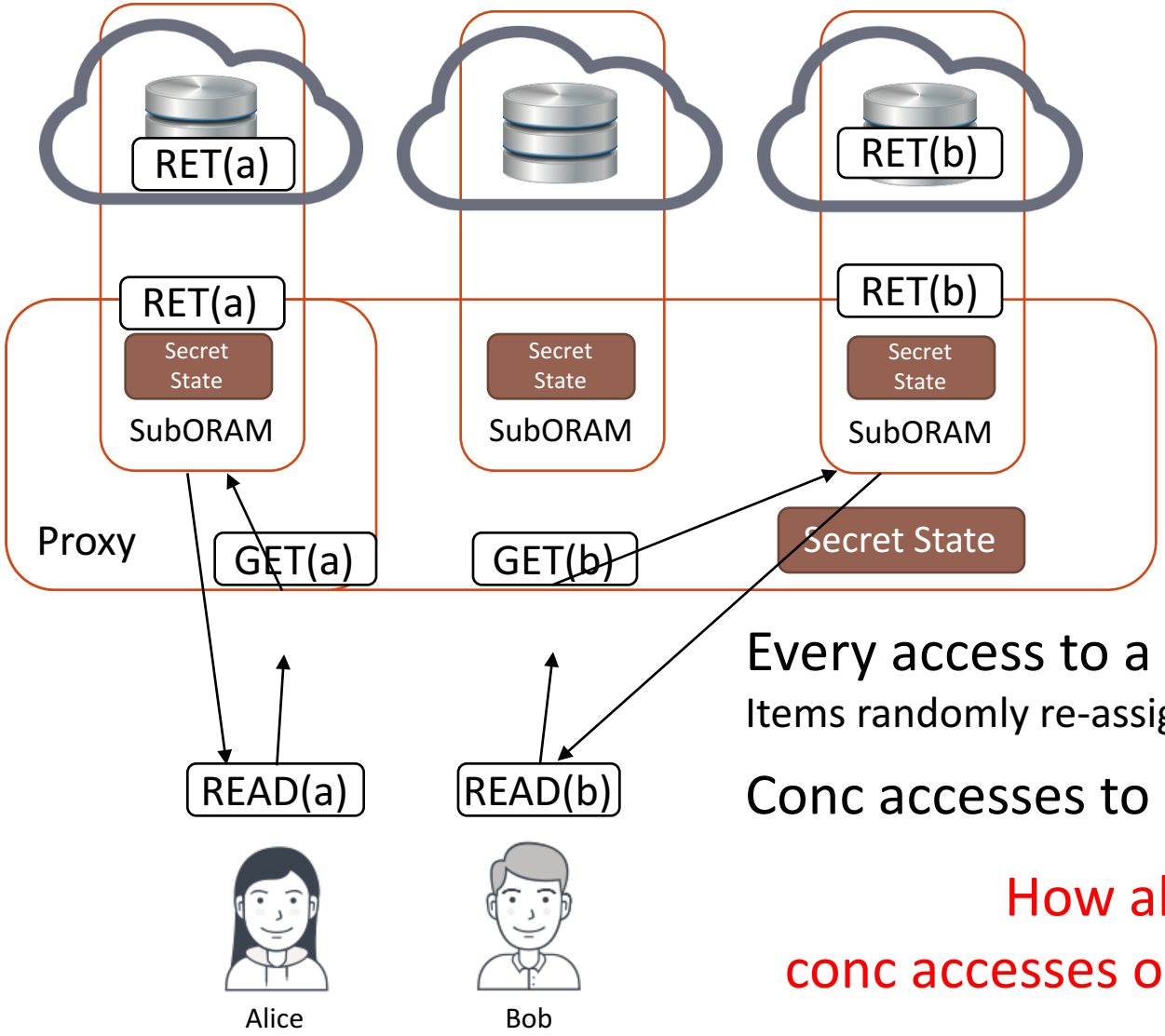
CURIOUS is secure in ObliviStore's threat model

We show

CURIOUS is not aaob-secure

Note: No claims are incorrect in CURIOUS

CURIOUS [Bindschaedler et al CCS'15]



Partition storage space

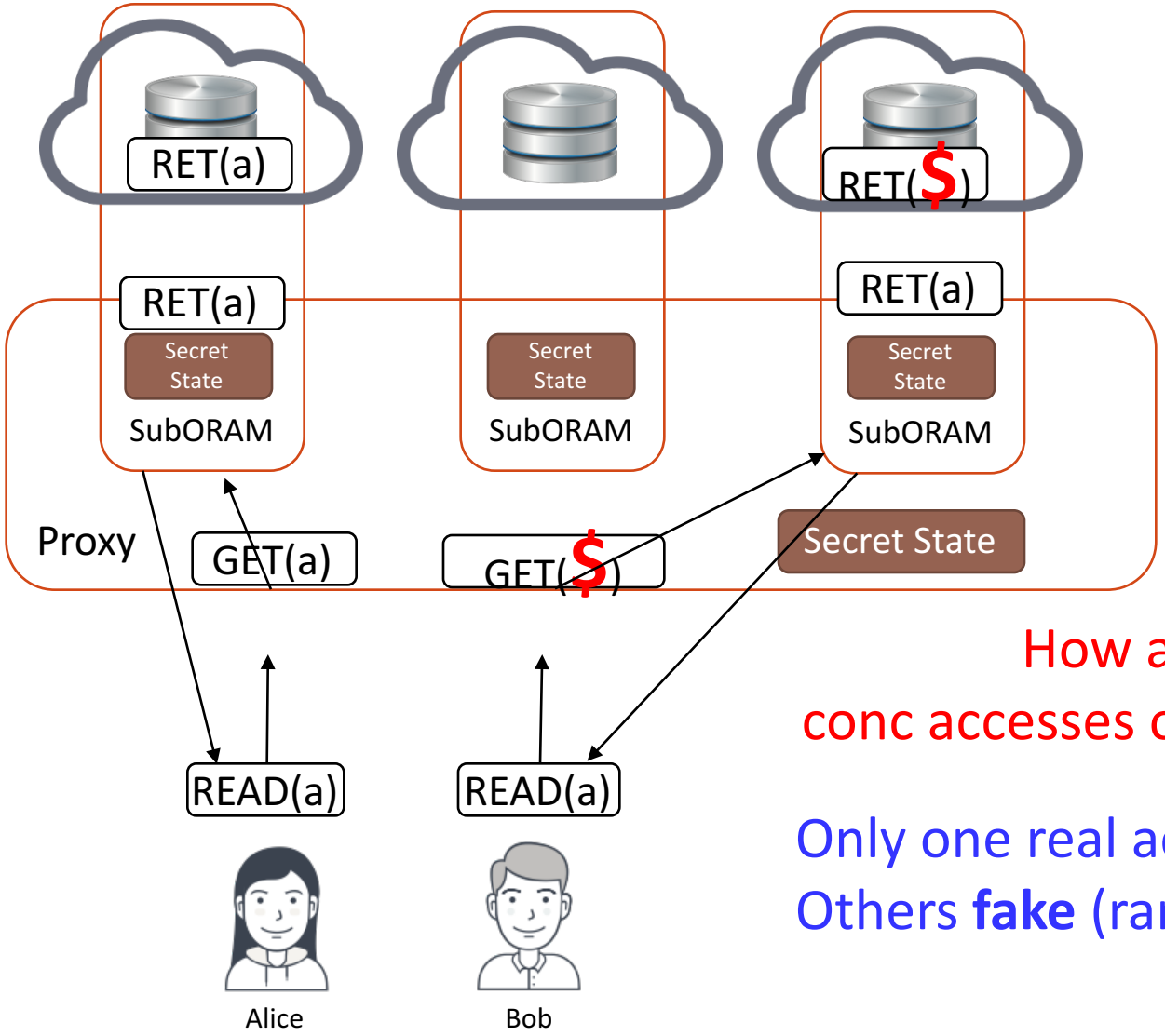
Every access to a random partition

Items randomly re-assigned after every access

Conc accesses to diff partitions

How about conc accesses on **same** item?

CURIOUS [Bindschaedler et al CCS'15]

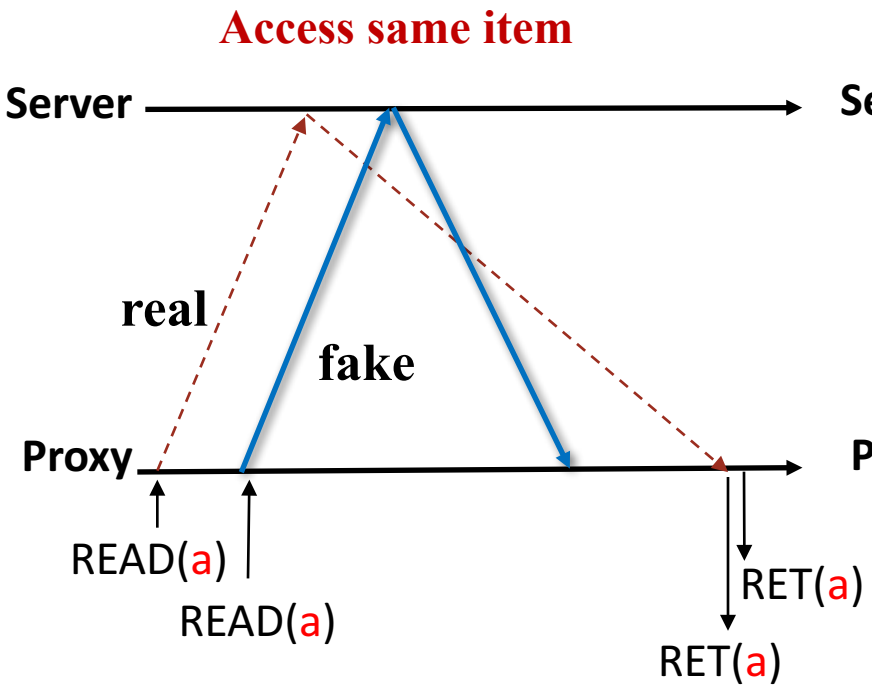


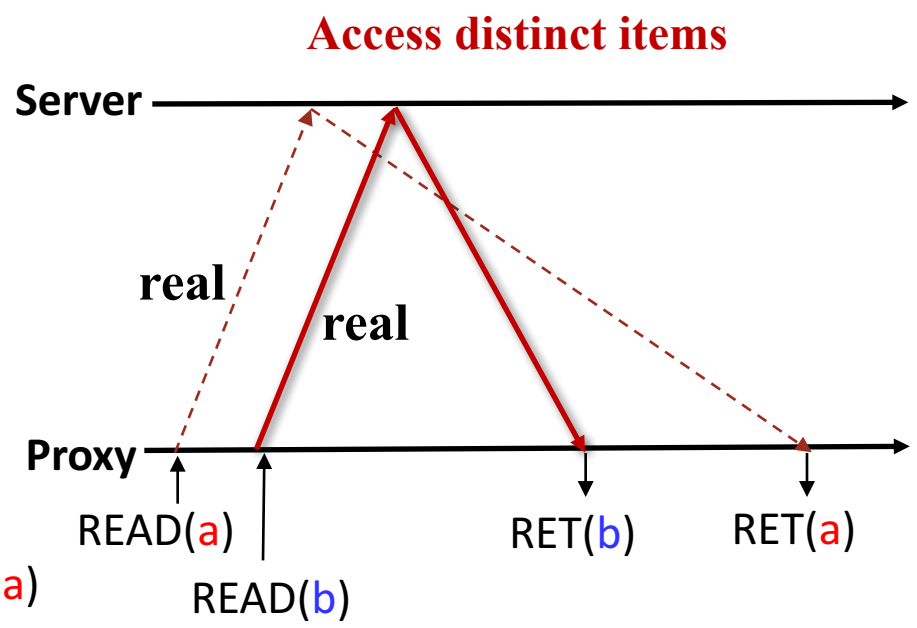
How about
conc accesses on **same** item?

Only one real access
Others **fake** (random) accesses

Attack Against CURIOUS

Reminder
Controls scheduling of messages + operations
Knows response timings





Attacker learns whether the accesses are on same item or not

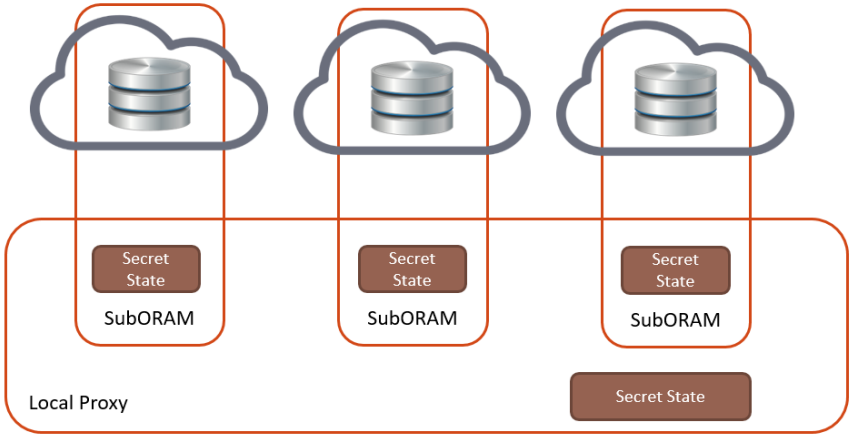
Fix? See later …

Contributions

A security model for asynchronous ORAM and attack

TaoStore: A new asynchronous and concurrent tree-based oblivious storage

CURIOUS: Modular, but two drawbacks



1. Security: Not aaob-secure

2. Efficiency: partitioning → concurrency
   (Underlying single-client ORAM not concurrent)

Wanted: Native concurrency!

Partitioning as simple add-on

Our solution – TaoStore

Tree-Based Asynchronous Oblivious Store

aaob-secure

Atomic semantics

Simple

Fully concurrent

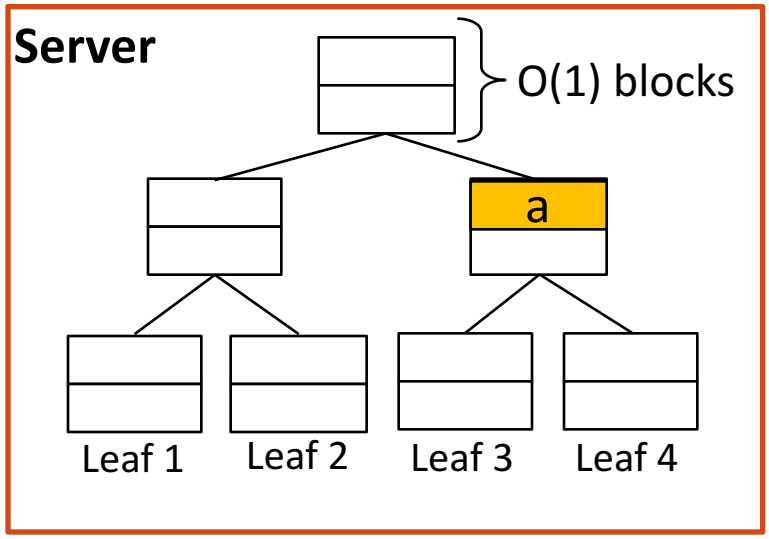
Enables easy partitioning

Many tree-based single-client ORAMs available: ES'11, ES'13, CG'13, KC'13, KC'14, XW'15, LR'15, SD'16, TM'15, …

Main challenge

How to make tree-based ORAM concurrent?

Starting point – Path ORAM [Stefanov et al CCS'13]



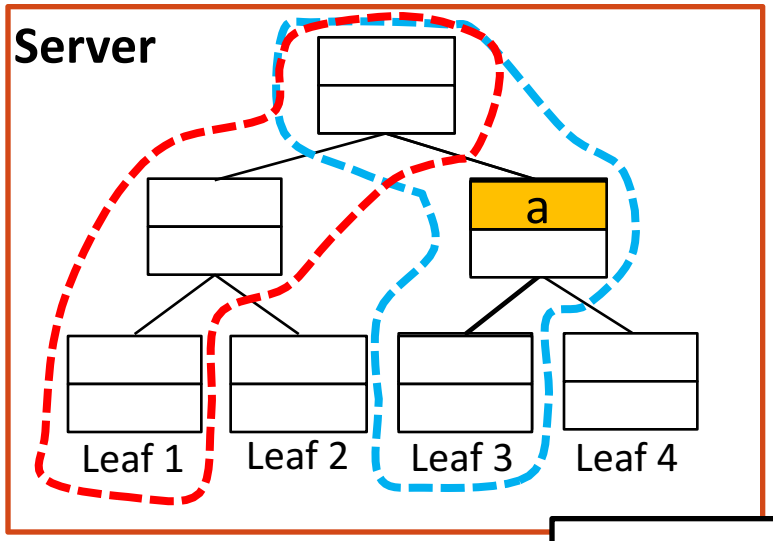
Storage is organized as a binary tree

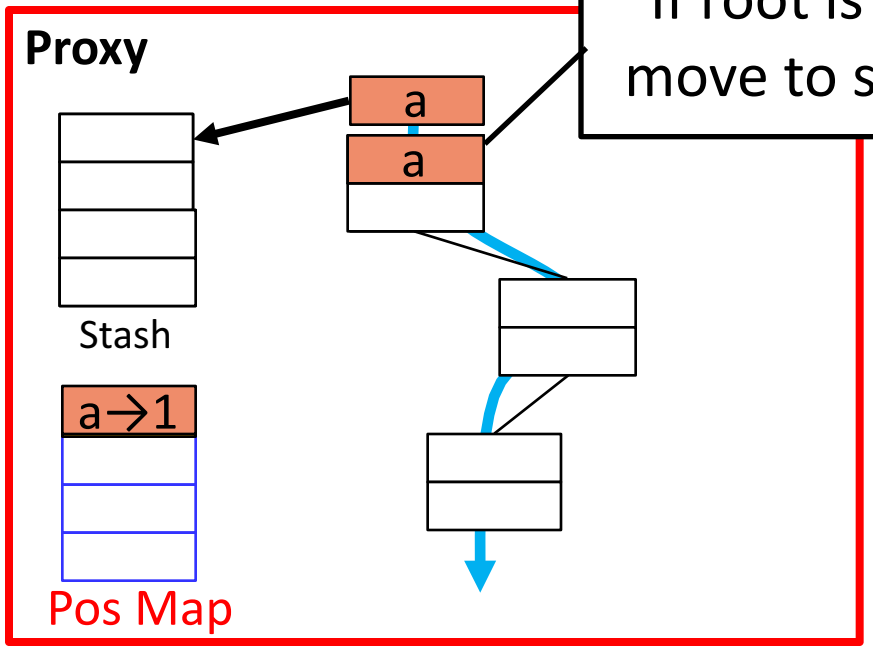
Every access to a random path

Items randomly re-assigned after every access

Possible to outsource position map recursively

Starting point – Path ORAM





If root is full move to stash

Read/Write block a

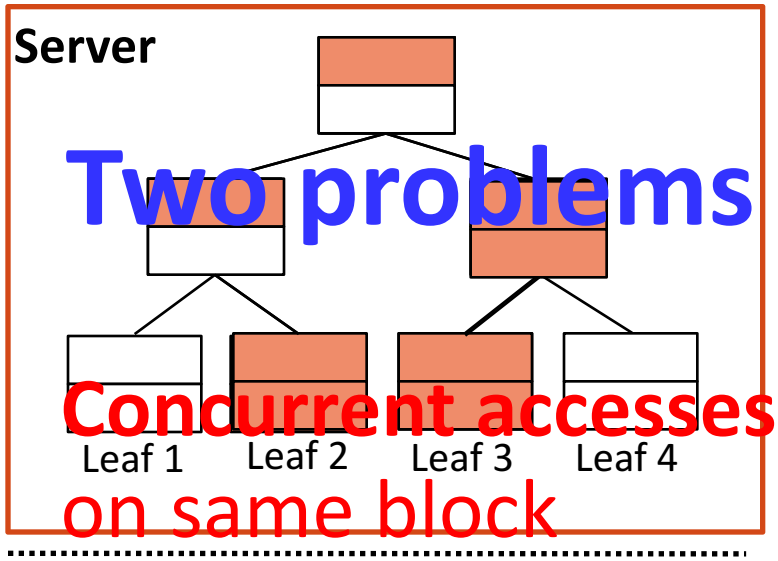
1) Read path
- Fetch associated path
- Read/Modify block
- Assign block to a new random path in position map

2) Flush

Push every block to the lowest non-full node that intersects with its assigned path (otherwise→stash)

3) Write-back
- Re-encrypt w/ fresh randomness

TaORAM – Basic Approach

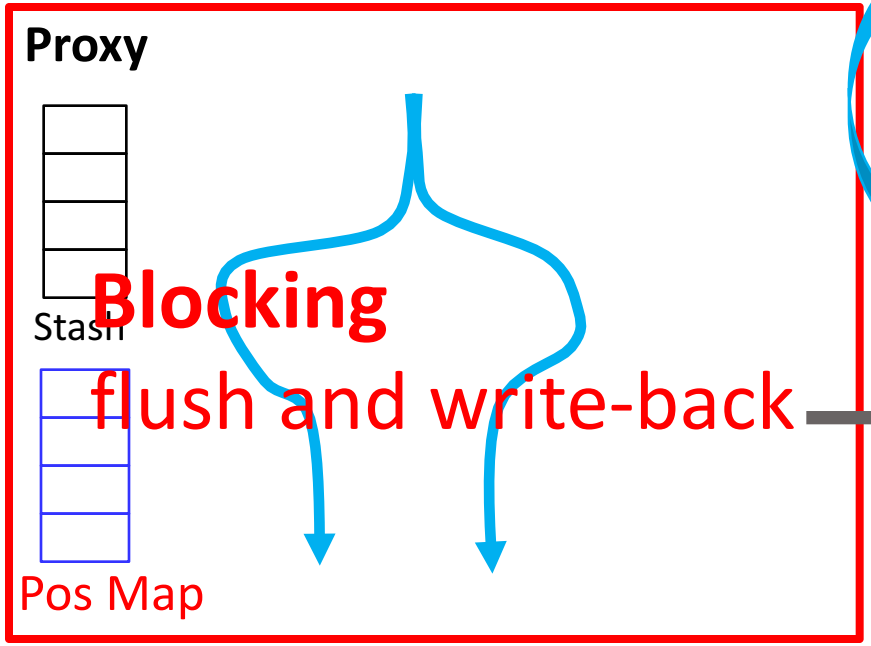


How to handle
≤ k concurrent requests?

STAGE 1

Process *k* operations

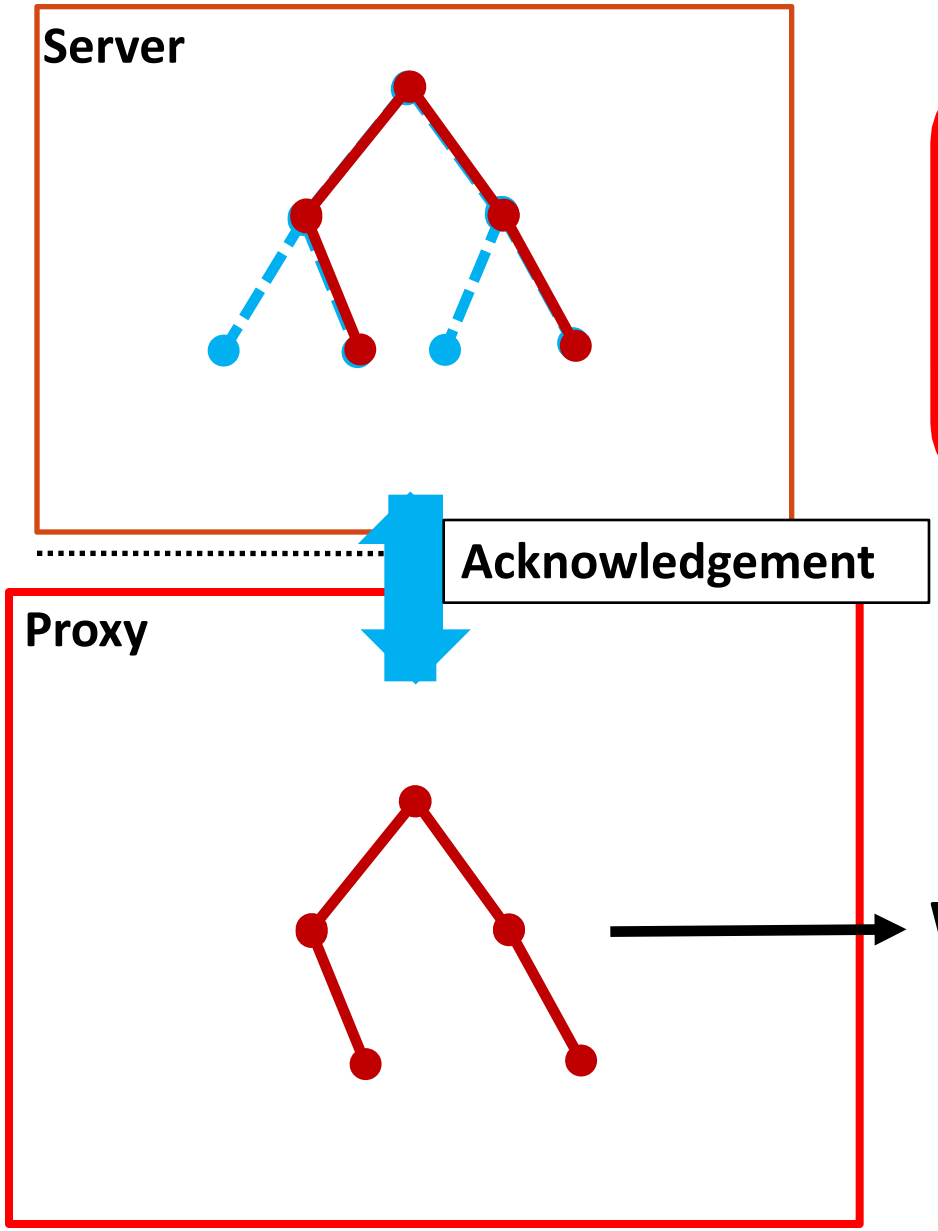
- Fetch corresponding *k* paths
- Form a *subtree* in proxy



STAGE 2

- Re-assign *k* items to new random paths
- Flush along the entire subtree and write-back

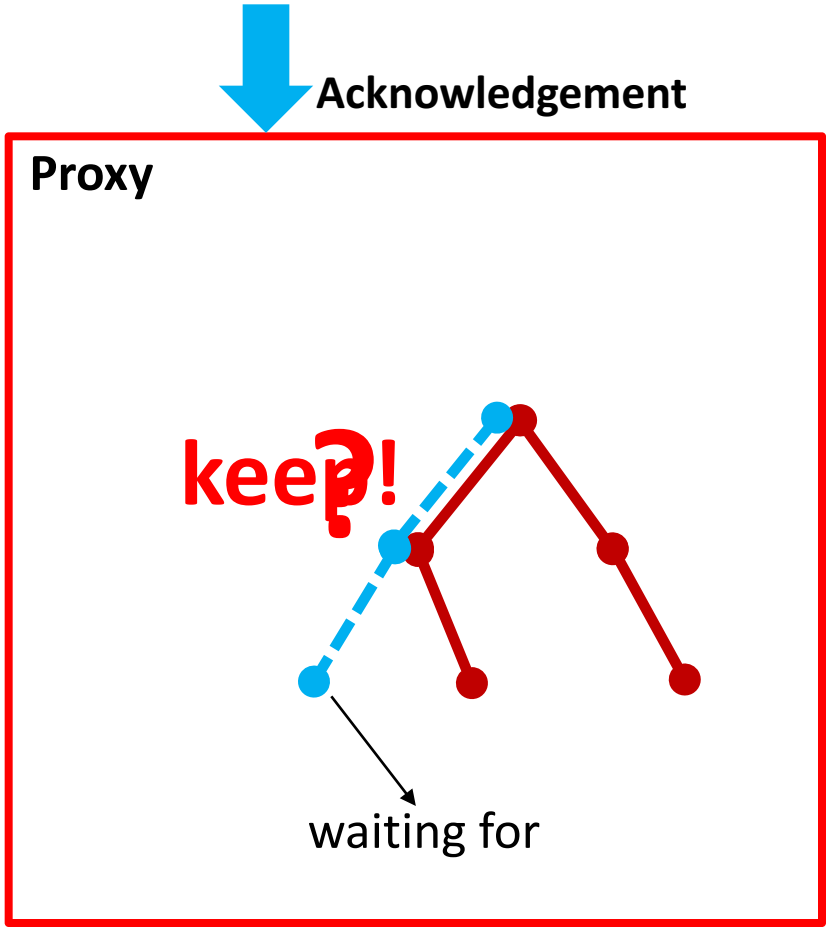
From Partial to Full Concurrency

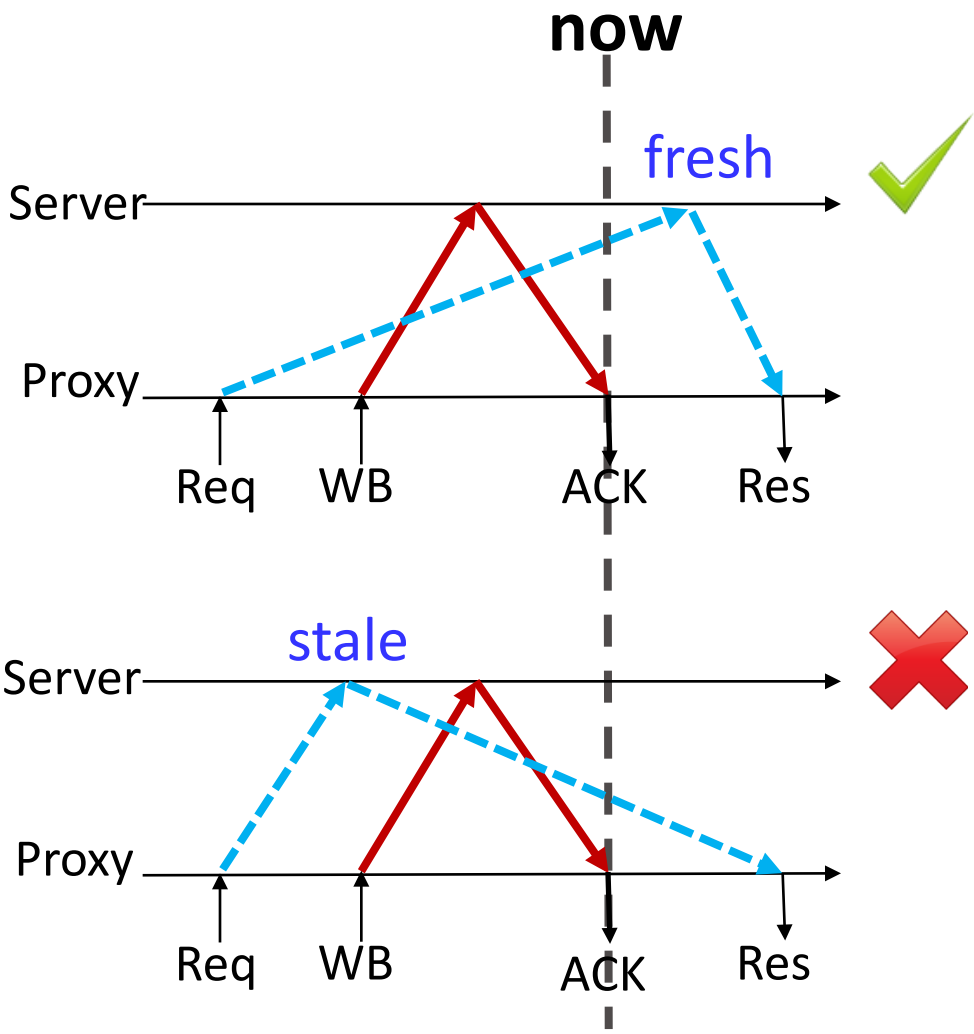


Non-blocking write-back
Continue processing operations while write-back is ongoing

What should we delete?

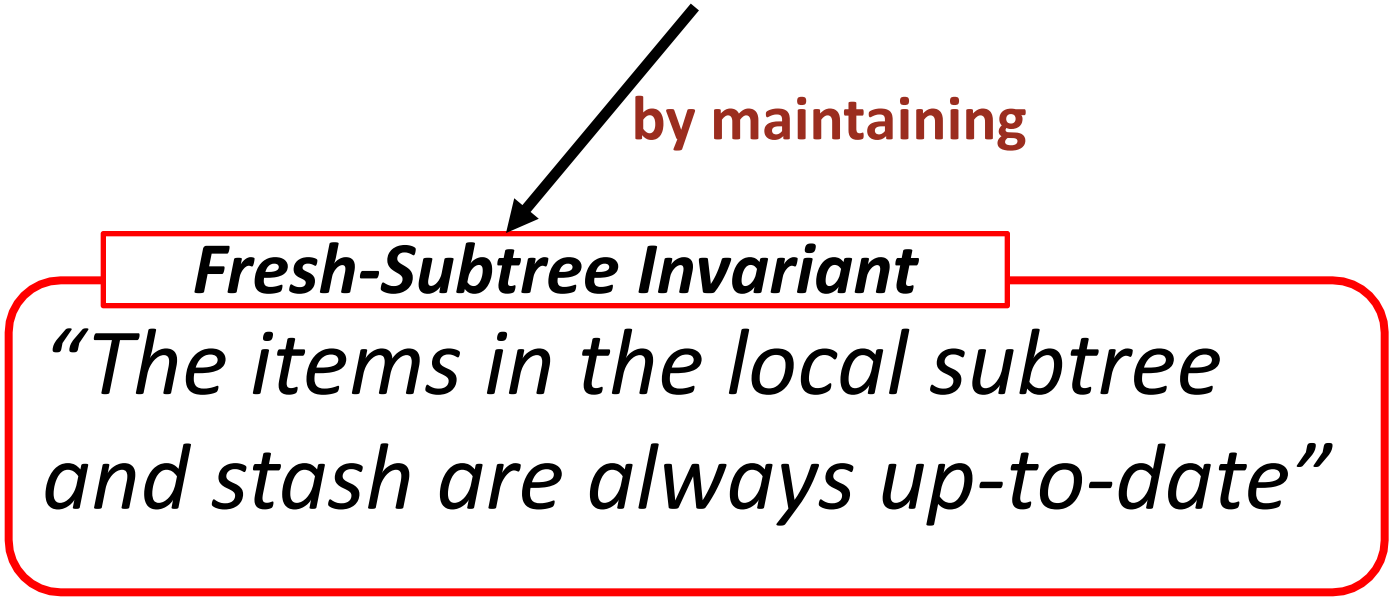
What should we delete?





+ more cases

TaoStore Achieves Full Concurrency



*See the correctness analysis in the paper.

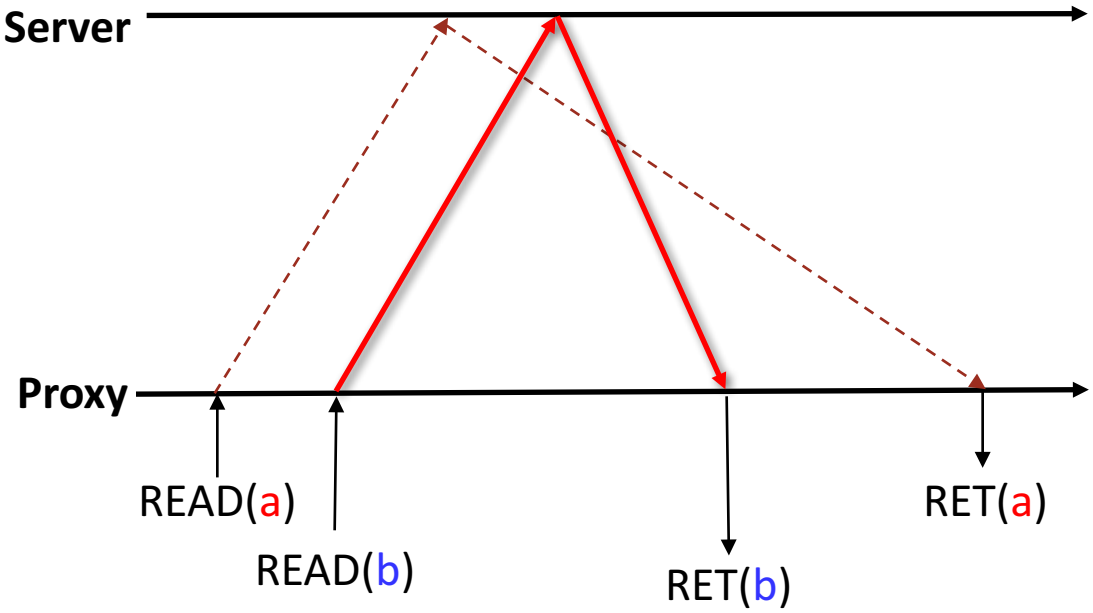
Concurrent accesses on same item

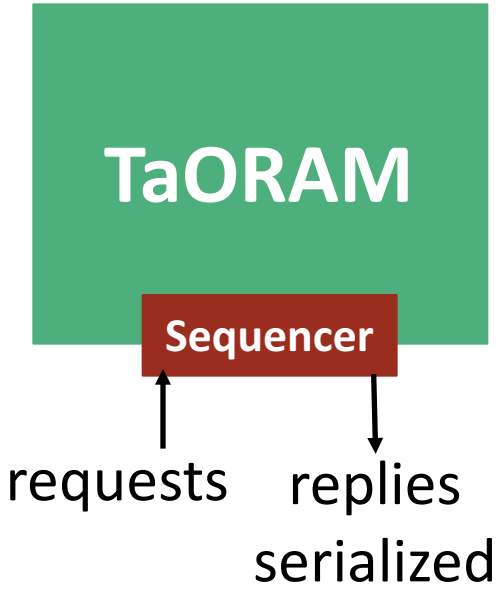
Use Fake Access (as in CURIOUS)

security problem

Our solution: Sequencer

Ensures logical requests replied in the same order they arrive

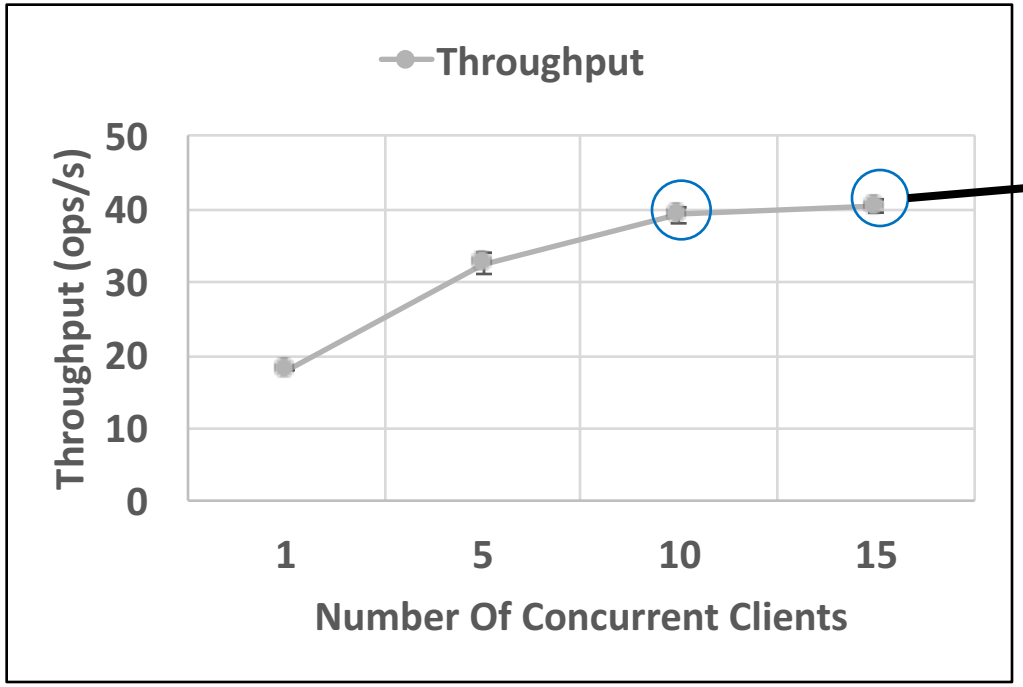




Generic solution: Also fixes CURIOUS

Cloud-based performance analysis

- Block Size: 4 KB - 1 GB dataset
- Proxy@UCSB (commodity workstation) + Storage Server: AWS EC2 (NorCal)
- Upstream/DownStream: 11 Mbytes/s. RTT: 12 ms
- Benchmark schedule: Adaptive requests

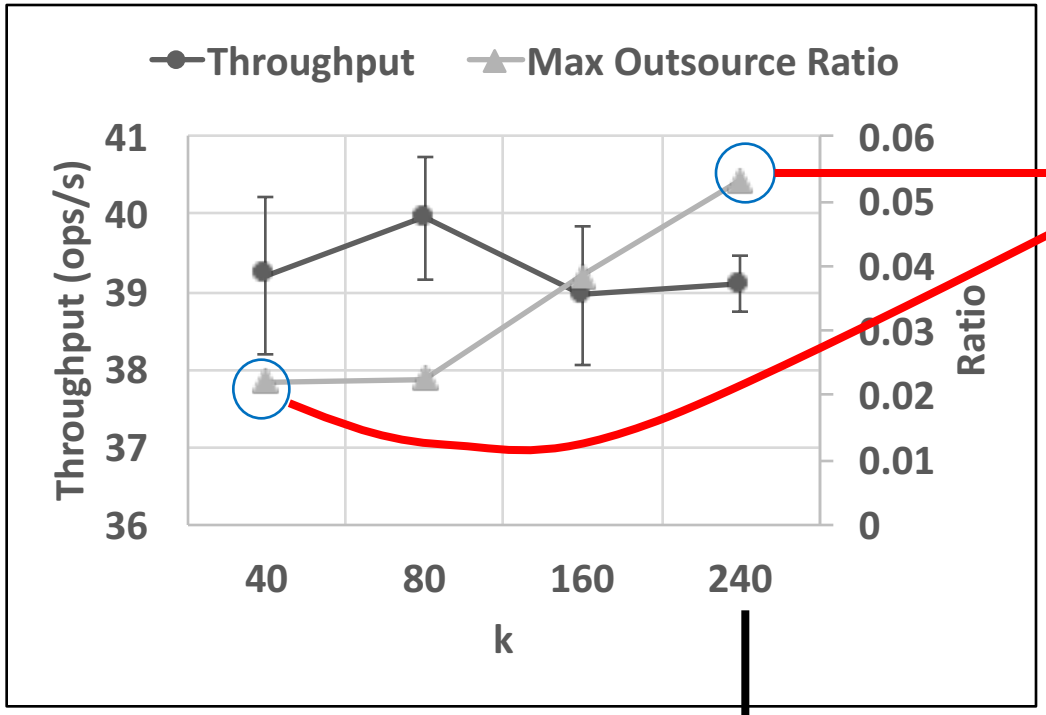


saturation due to bandwidth exhaustion

Bandwidth matters!

Cloud-based performance analysis

- Block Size: 4 KB - 1 GB dataset
- Proxy@UCSB (commodity workstation) + Storage Server: AWS EC2 (NorCal)
- Upstream/DownStream: 11 Mbytes/s. RTT: 12 ms
- Benchmark schedule: Adaptive requests



a low-memory utilization achieves similar performance

write-back in batches after k (240) path fetches

THANKS!

A security model for asynchronous ORAM and attack

TaoStore: A new asynchronous and concurrent tree-based oblivious storage