ObliviSync

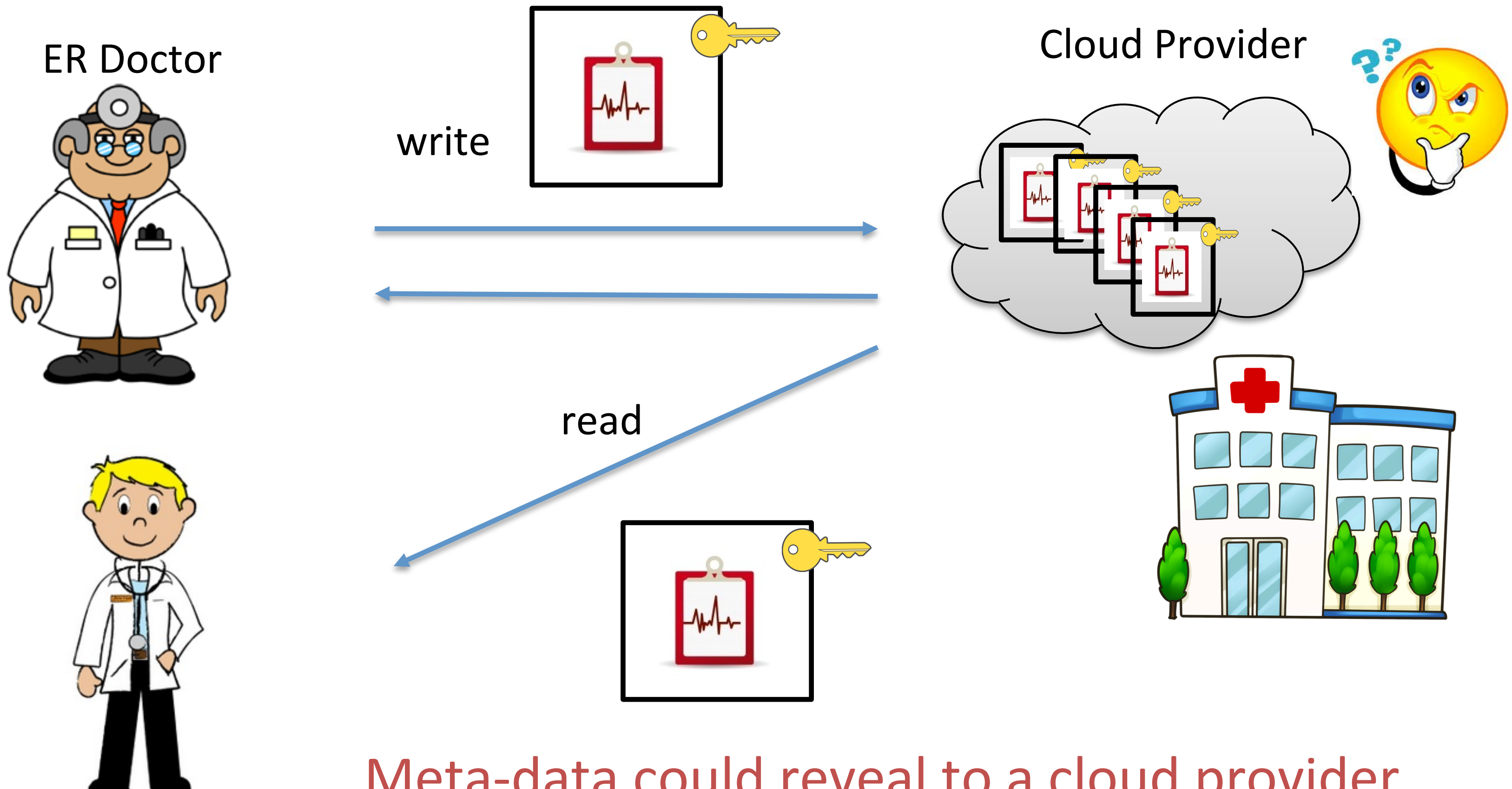
Practical Oblivious File Backup and Synchronization

Adam J Aviv   Seung Geol Choi   Travis Mayberry   Daniel S. Roche

United States Naval Academy
Annapolis, MD

Meta Data Protection

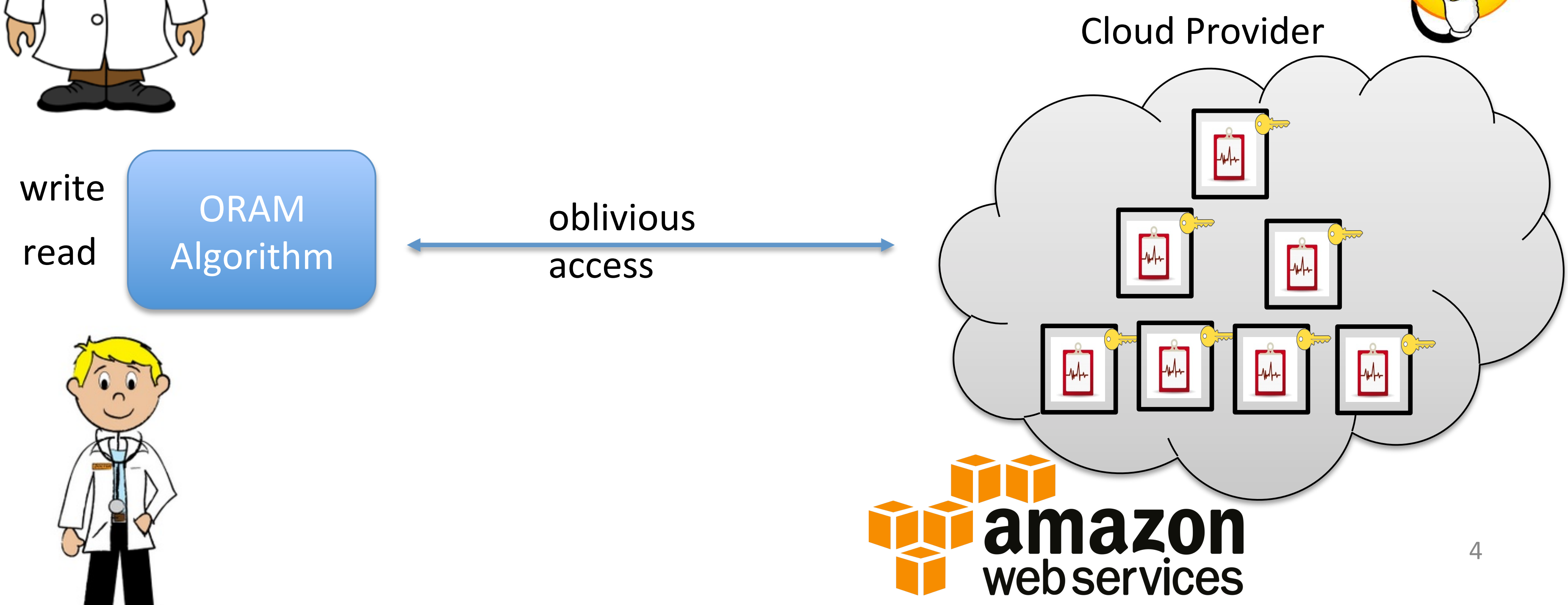
Meta Data Threat
e.g., Access Patterns



Meta-data could reveal to a cloud provider information about the patient, <u>even if the records are encrypted!</u>

Oblivious RAMs (ORAMs)

Threat Model:
Preventing the cloud provider from learning which files are accessed and when



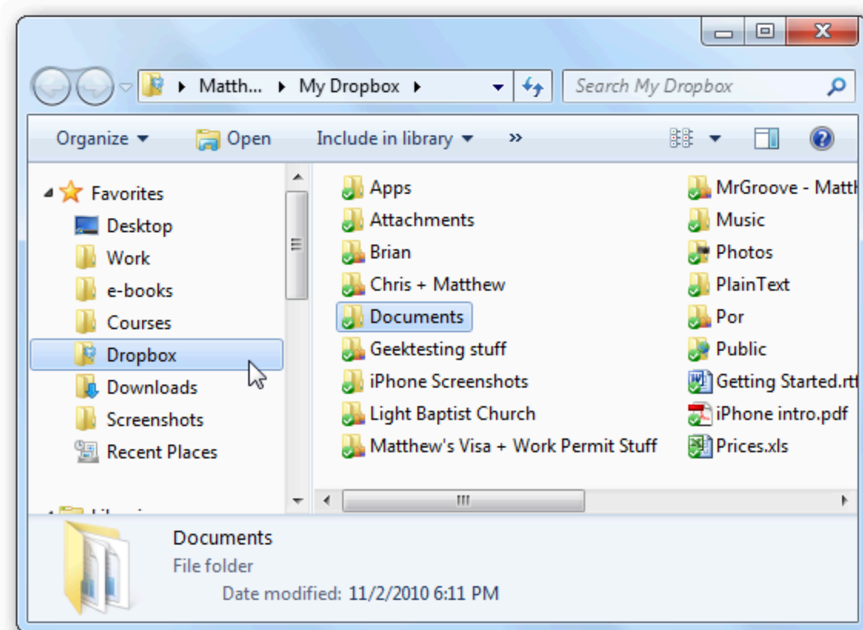
DropBox
Cloud Synchronization Setting

- Store a *local copy* of files across multiple computers

Reading is Oblivious (occurs locally)

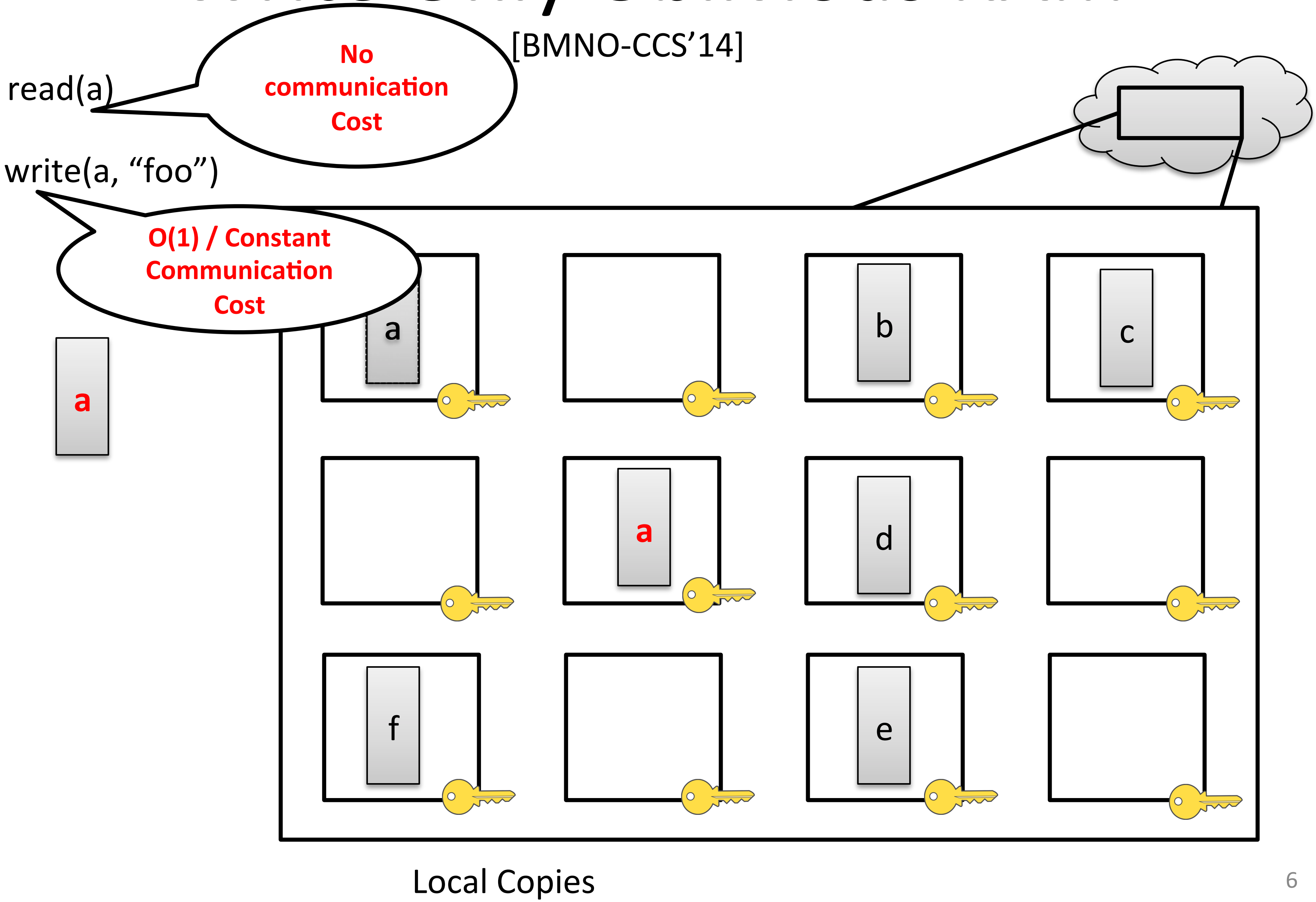
- *Synchronizes* writes to other clients' local copies

Writing needs protecting (revealed to cloud)

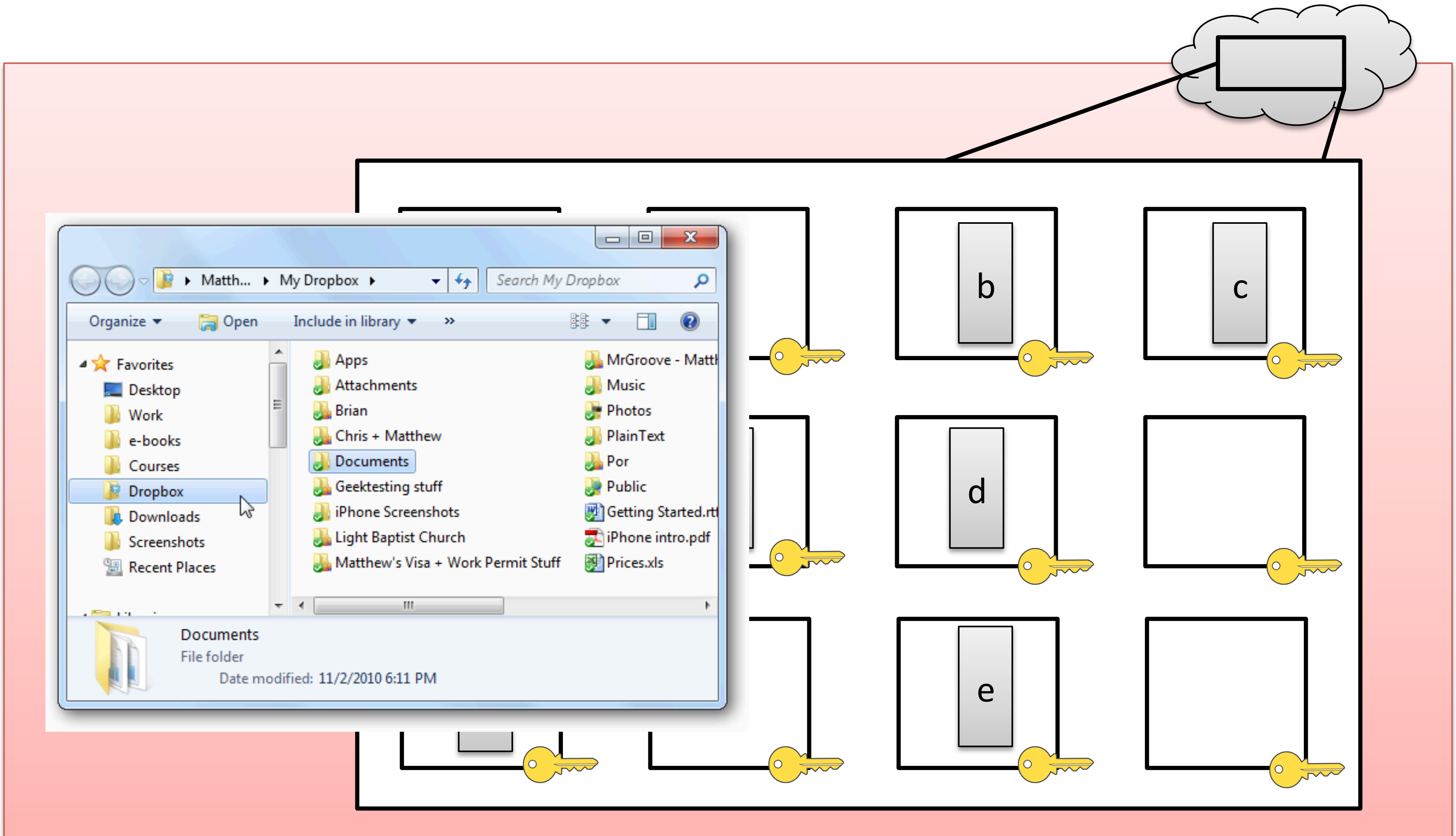


Write Only Oblivious RAM

[BMNO-CCS'14]



OblivSync


b
c
d
e
Matth... ▸ My Dropbox ▸
Search My Dropbox
Organize
Open
Include in library
Favorites
Desktop
Work
e-books
Courses
Dropbox
Downloads
Screenshots
Recent Places
Apps
Attachments
Brian
Chris + Matthew
Documents
Geektesting stuff
iPhone Screenshots
Light Baptist Church
Matthew's Visa + Work Permit Stuff
MrGroove - Matth
Music
Photos
PlainText
Por
Public
Getting Started.rtf
iPhone intro.pdf
Prices.xls
Documents
File folder
Date modified: 11/2/2010 6:11 PM

Our Contribution: ObliviSync

- *Adapting Write-Only ORAM with the Cloud Synchronization and Backup Model*

- Specifically model after DropBox like systems
  - Seamless file system integration
  - Seamless oblivious synchronization across clients

- Strong Security and Efficient Design
  - Write Oblivious and Timing Attack protection
  - Small overhead, 4x compared to non-private stores
  - Variable Size Files

- Realistic Implementation
  - Implemented using FUSE
  - Seamlessly works with Dropbox

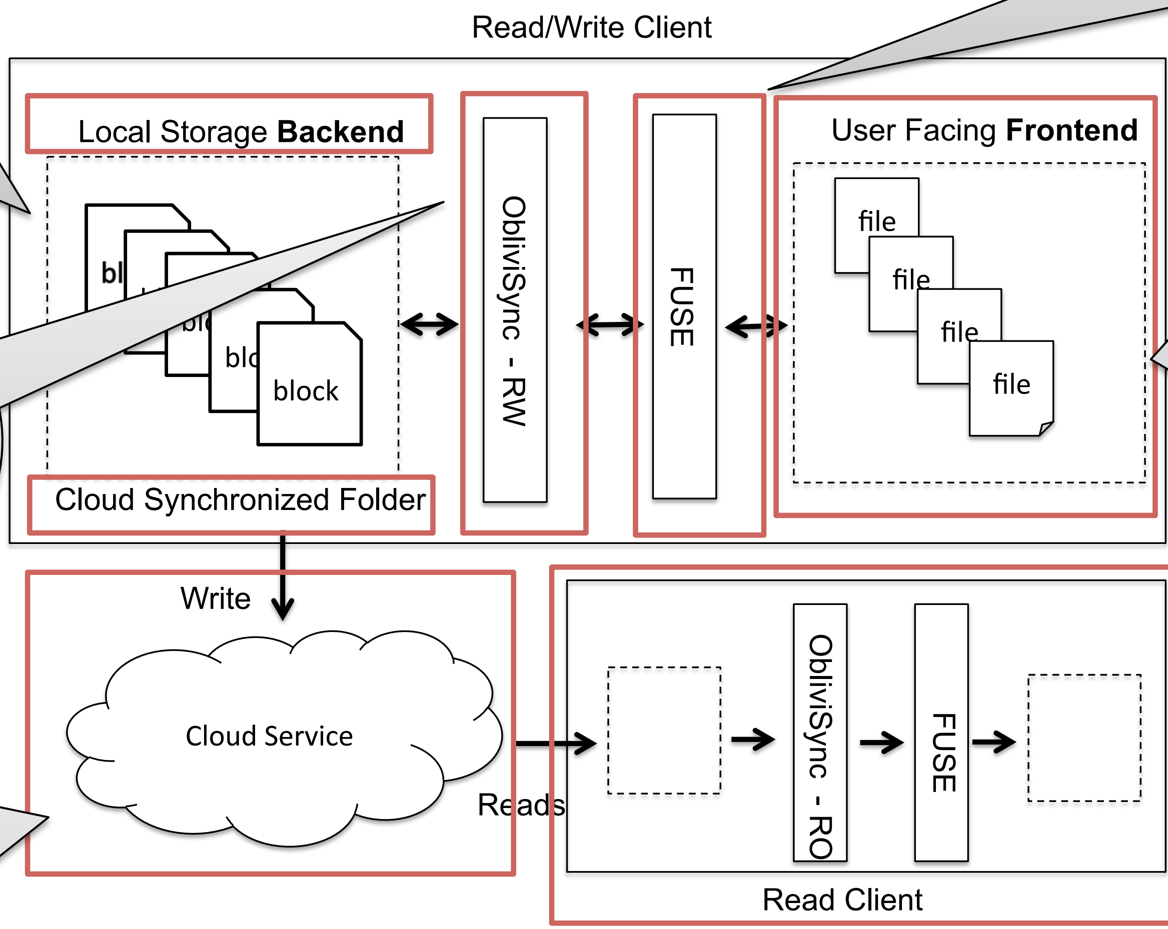
OBLIVISYNC DESIGN

ObliviSync Components

Backend is a collection of files for the write-only ORAM Stored in a synchronized folder

Encode a file system into the backend block that is efficient for Write-Only Oram

Updates to the backend are synchronized to other clients



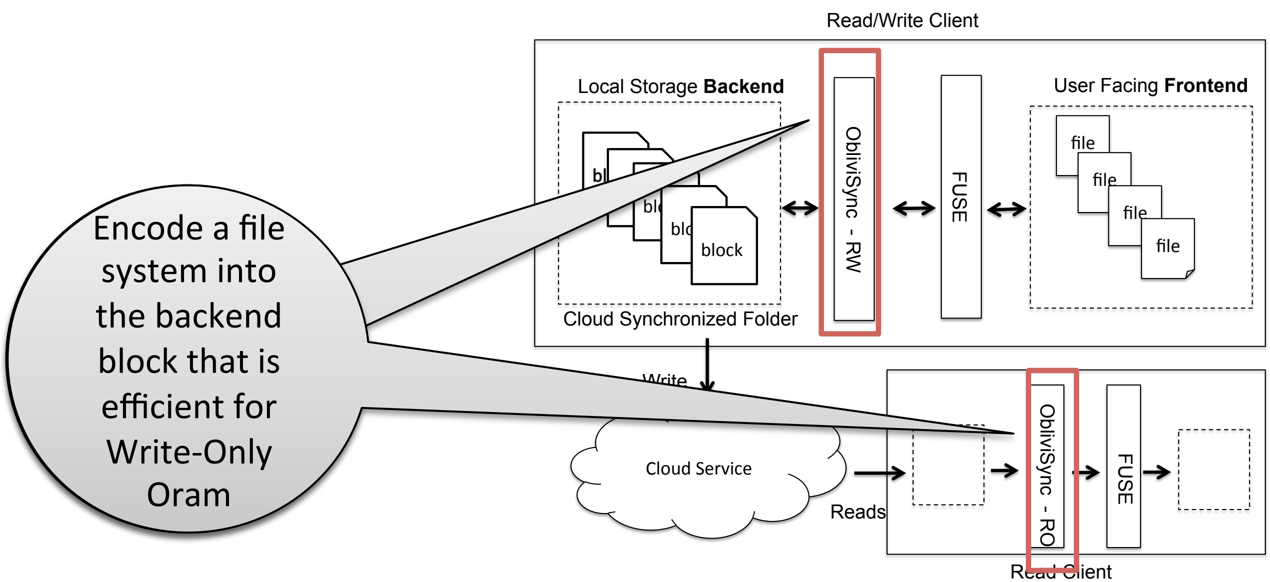
File system is "mounted" into the system using FUSE

User interaction occurs through normal file system calls

Read Client mounts the encoded file system with FUSE but only enables reading

Why embed a file system?

- Why not just treat the Write-Only ORAM as a block device?
  - <u>Efficiency and Security</u> of the system will be strongly dependent on avoiding unnecessary writes
  - Block devices may reveal access times and file sizes



ObliviSync Backend: ***TERMINOLOGY***

File-Id's: identifier of files stored with the embedded file system

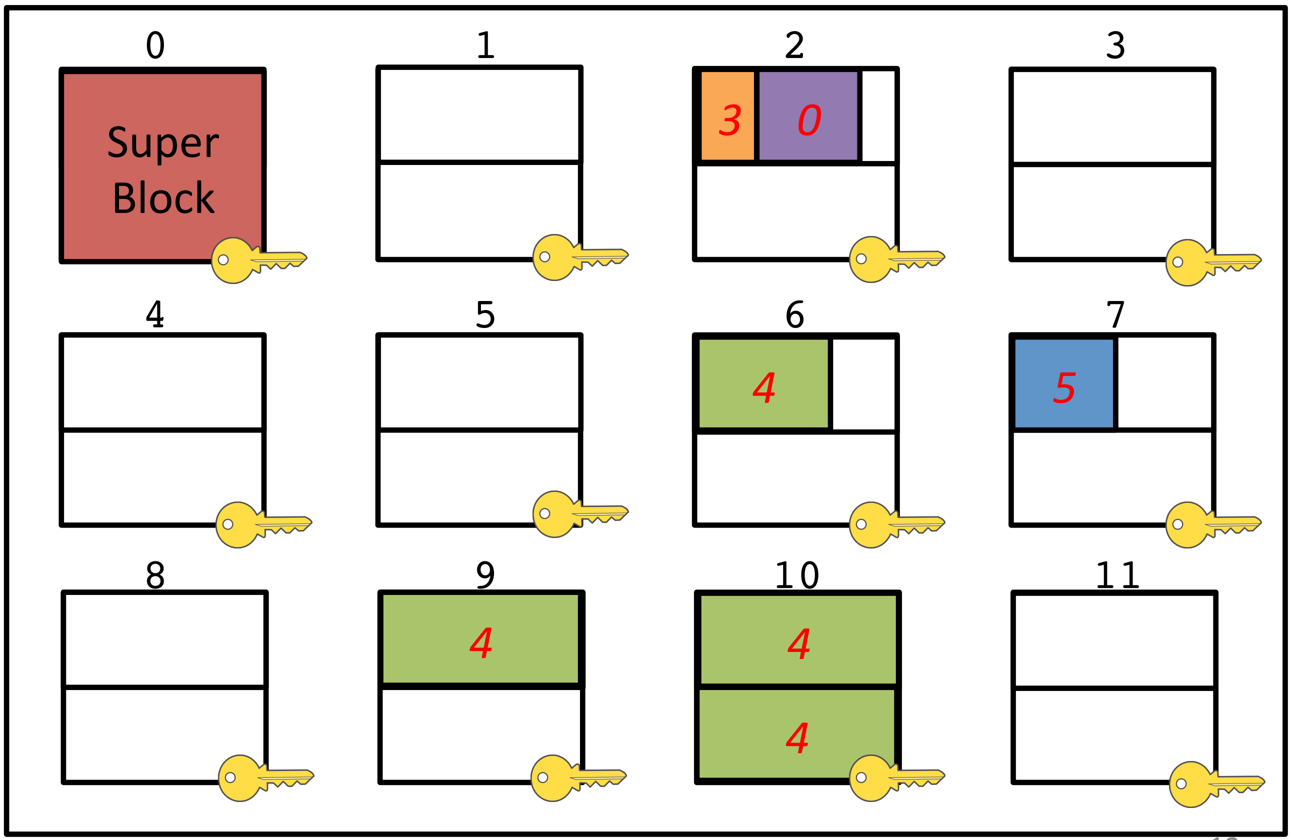
File-segments: Files are broken up to fit within blocks, can either be full or partial

Directory Entry: Root of file system, always have File-Id 0

Split-block: Each block in the backend is partitioned into two split-blocks

Block Id's: Identifier for a split-block in the backend

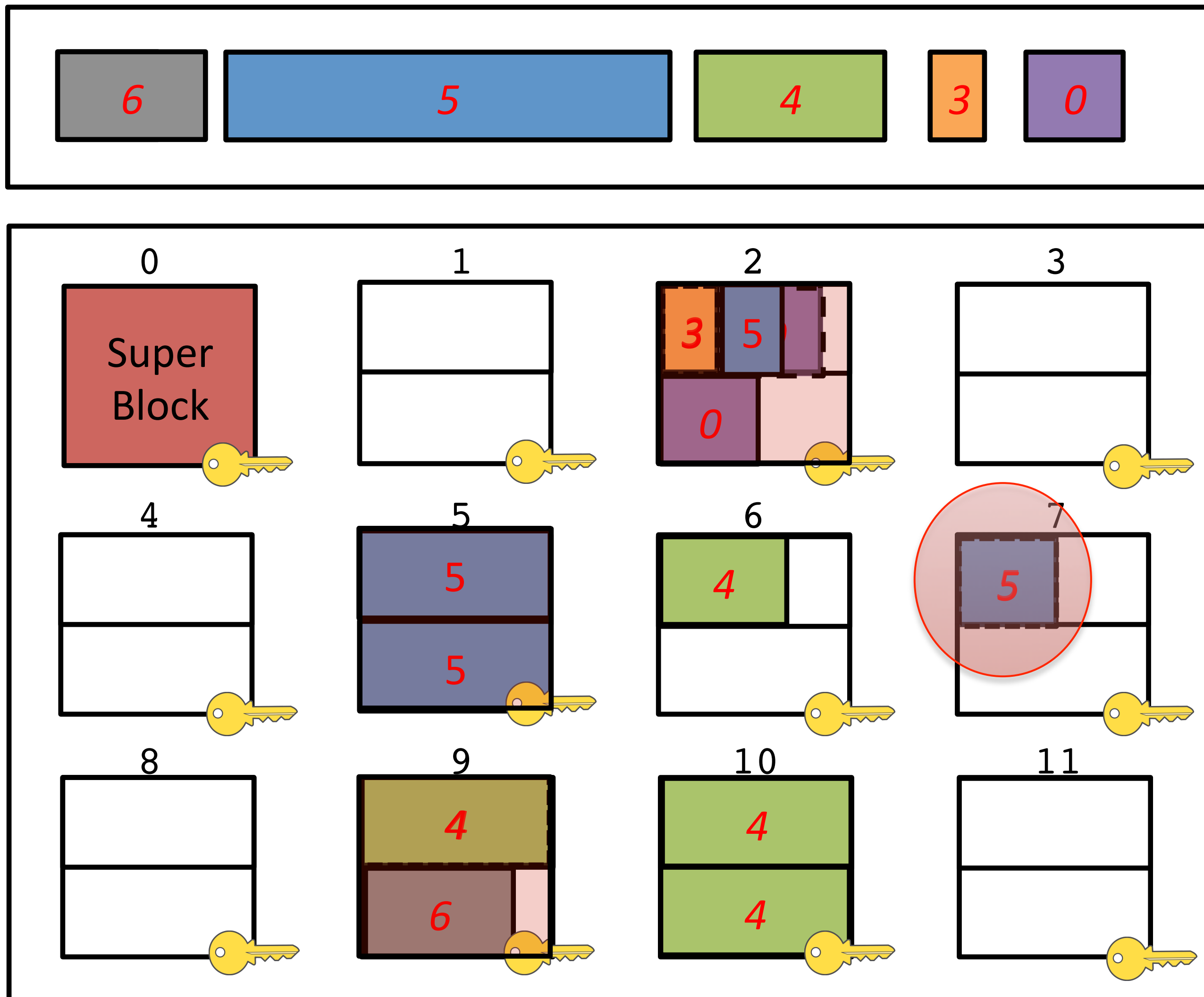
Superblock: Block with Block-Id 0 used to structural information for the embedded file system



Drip Rate = 3

Drip Time= 5 (s)

Synchronizing Buffer



Repacking Rules

- Existing file segments filling a full split block does not change location
- Existing file segments filling less than a full split block may only move to the other split block in the pair.

creat() -> 6

write(6)

write(6)

open(5, "a")

write(5)

close(5)

open(5)

read(5)

close(5)

Summary of Design Settings

- ***Specialize File System Embedded within a Write-only ORAM***
  - FUSE based user facing frontend for transparent user experience

- ***Synchronize to Cloud at Regular Intervals (epochs)***
  - Buffer writes and synchronize buffer via write-oblivious operations
  - Synchronize even when there is nothing in the buffer (*protection from timing attacks!*)

- ***Multiple Clients***
  - Allow only <u>one</u> reading and writing client
  - Can have any number of read-only clients receiving synchronizations

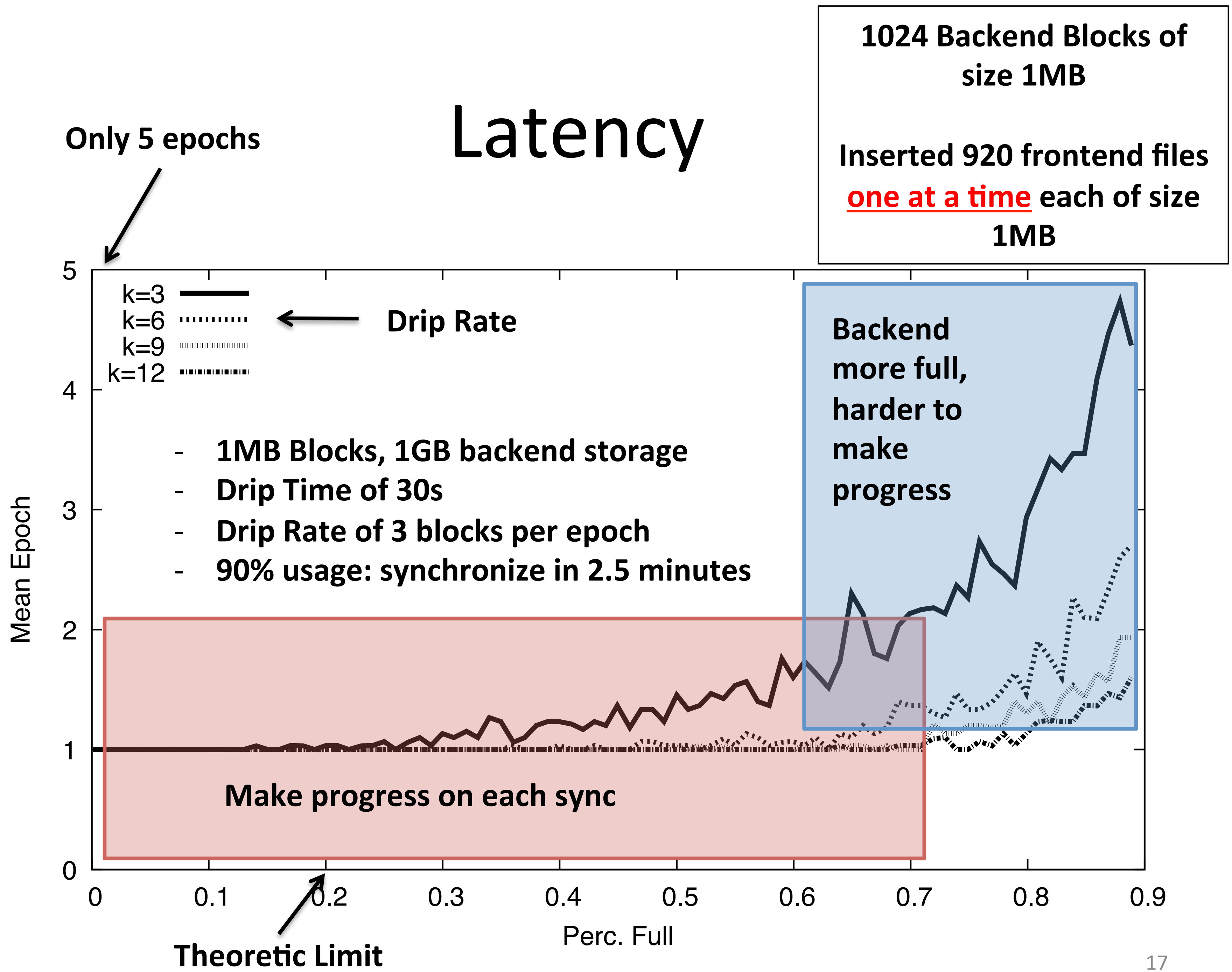
- ***Easily tuned to the right setting: <u>drip rate</u> and <u>drip time</u>***
  - to the <u>Cloud Storage Provider</u>: the size of the backend blocking
    - 4MB vs. 1MB vs. 4K blocks (Dropbox using 4MB backend)
  - to the <u>Application</u>: The amount and frequency of synchronization
    - Cloud File Syncs: Higher synchronization rate with lower amounts
    - Regular Backups: Lower synchronization rate with higher amounts

RESULTS

Experimental Results
Latency

- **Latency**
  – Insert a large number of files *one at a time*
  – *How long does it take for each of the files to sync?*
    - As there is less empty space to pack in files, should expect a decrease in performance

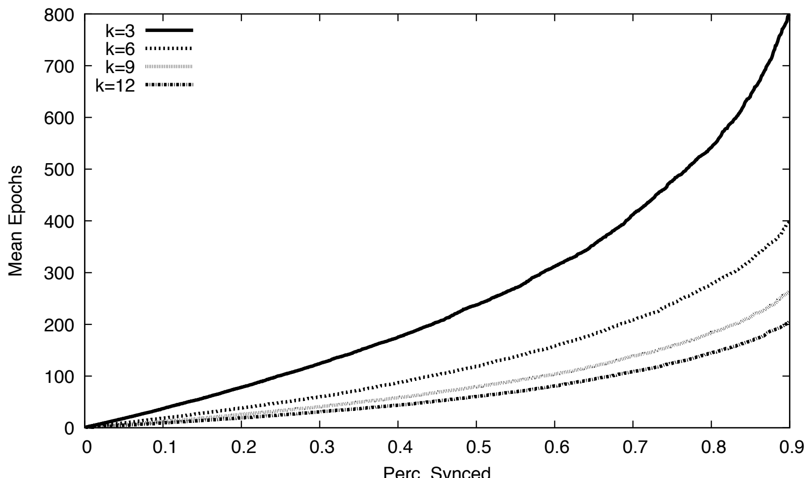
Latency



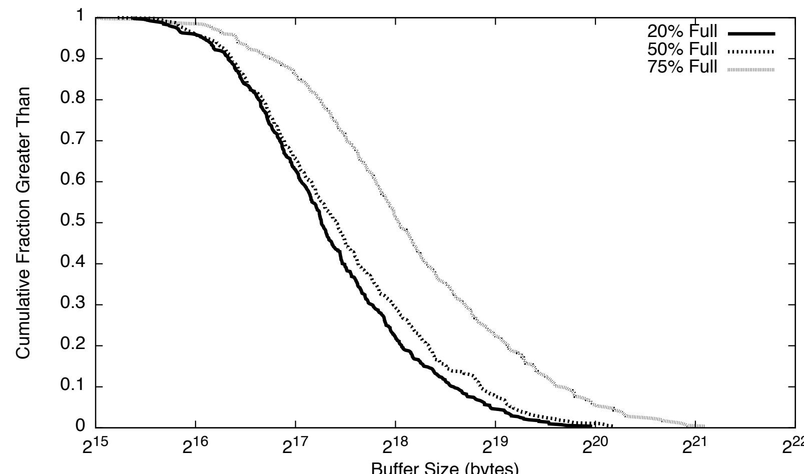
1024 Backend Blocks of size 1MB
Inserted 920 frontend files one at a time each of size 1MB
Only 5 epochs
k=3
k=6
k=9
k=12
Drip Rate
- 1MB Blocks, 1GB backend storage
- Drip Time of 30s
- Drip Rate of 3 blocks per epoch
- 90% usage: synchronize in 2.5 minutes
Backend more full, harder to make progress
Make progress on each sync
Mean Epoch
Perc. Full
Theoretic Limit
0
1
2
3
4
5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

More Results in the Paper!

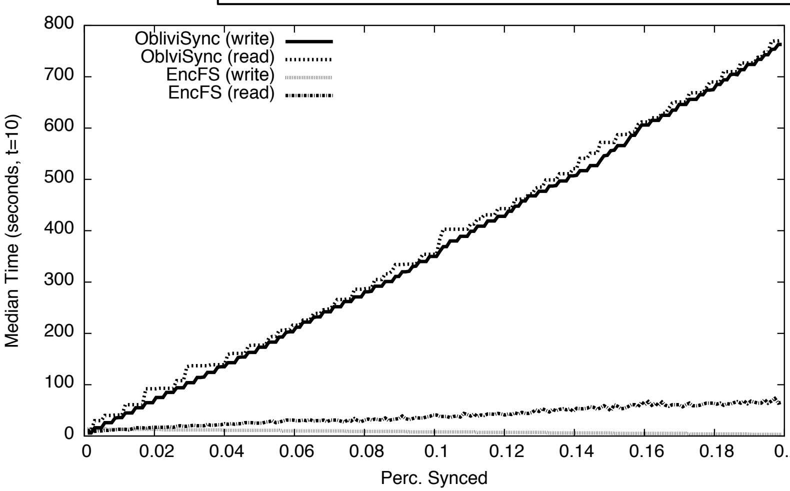
Throughput Measurements

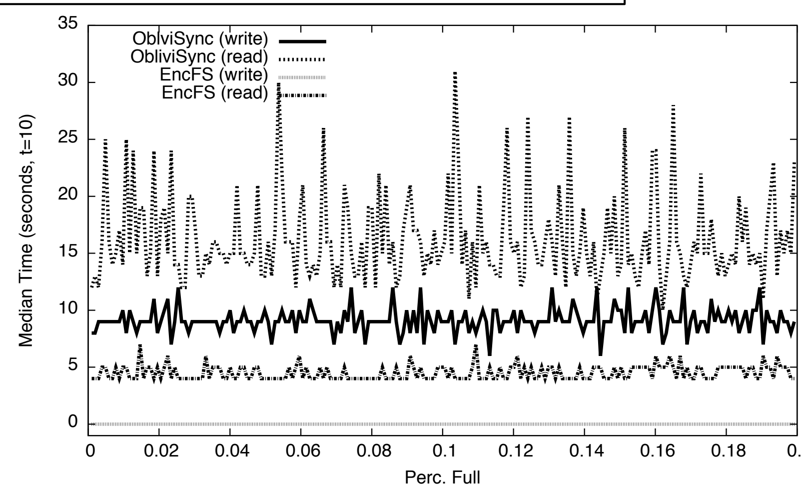


Realistic File Sizes



Comparison running on DropBox





Takeaways

- ***Oblivious Synchronization Services is <u>PRACTICAL</u>***
  - Reads are already Oblivious, need to protect writes
  - Leverage properties of the application
  - Small communication overhead: 4x

- ***ObliviSync***
  - Adapting Write-Only ORAM with a specialized Filed System
  - Handles variable size files
  - Is NOT susceptible to timing attacks
  - Tunable to the application
  - Implemented for a DropBox-like application that is transparent to the user

THANKS! Questions?

ObliviSync

Practical Oblivious File Backup and Synchronization

Adam J Aviv   Seung Geol Choi   Travis Mayberry   Daniel S. Roche

United States Naval Academy

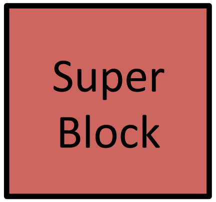
Annapolis, MD

Code Repository

https://github.com/oblivisync/oblivisync

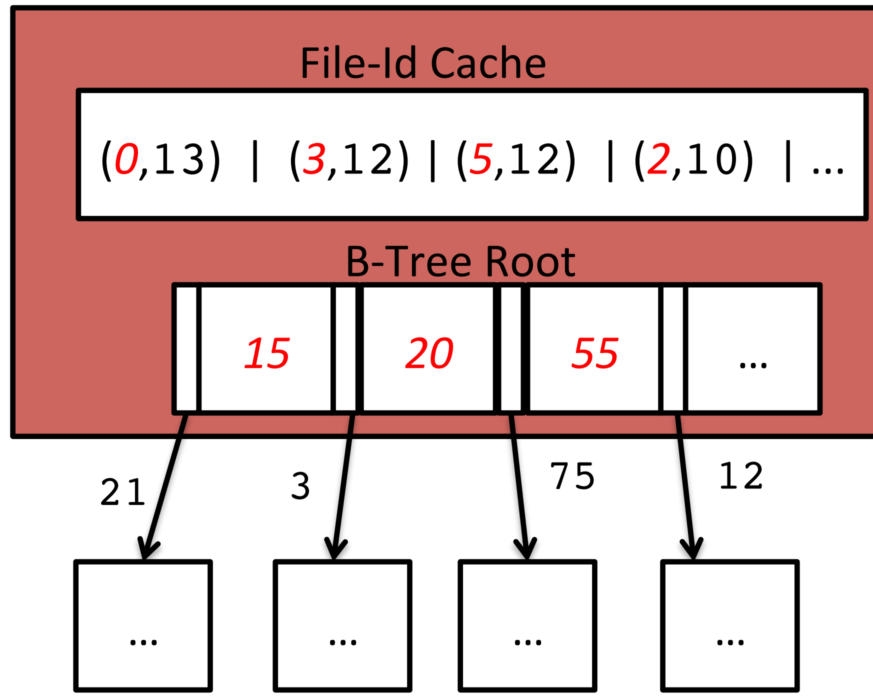
YouTube Video

https://youtu.be/-MYgtts_sO8



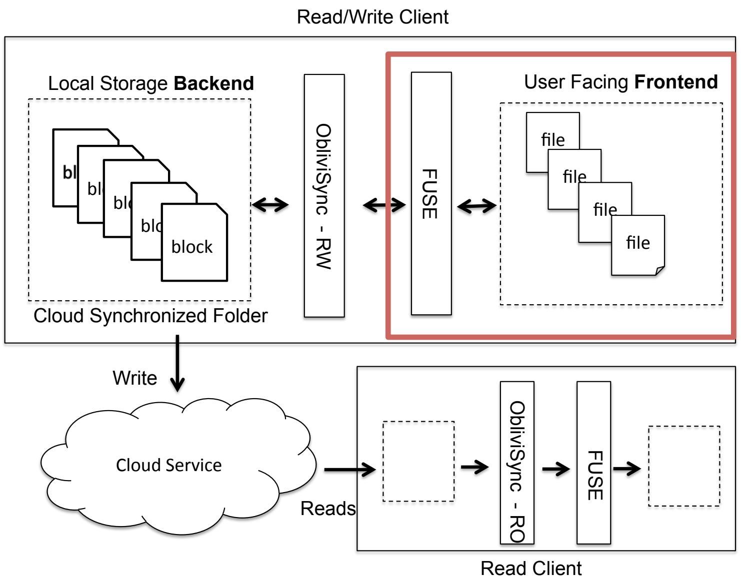
Superblock

- Mapping of File-Id to Block-Id
  - Directory entry maps filenames to File-Id's
  - Read (and written) on every access to the system
- Use a 2-level B-tree
  - B-Tree root is stored in the super block
  - Each leaf node is treated like a block in the system and referenced by its Block-Id
  - *With large blocks only need one level for most systems*
- Cache of recent mappings
  - Improves access time
  - All changes can occur within the super block without having to access leaf nodes



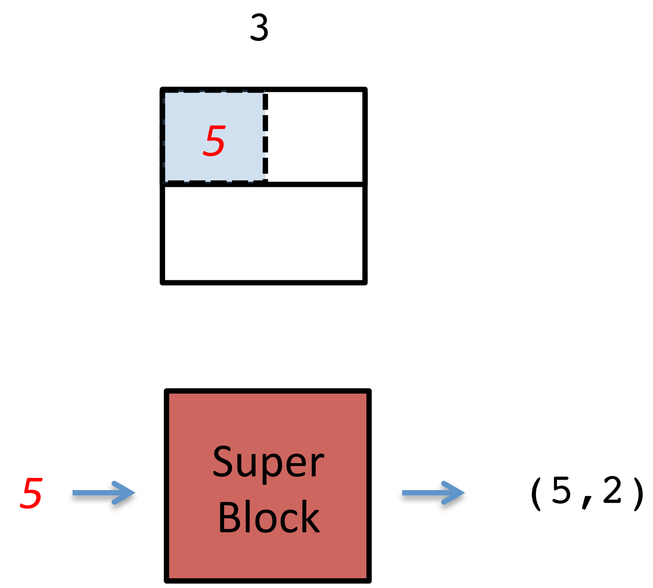
FUSE

- File System in User Space
  - A process intercepts all I/O system calls
- FUSE mounts the embedded file system such that it appears like any other directory to the user
- FUSE client also maintains the directory entry and is aware of the underlying ObliviSync System for efficiency



Detecting Stale Data

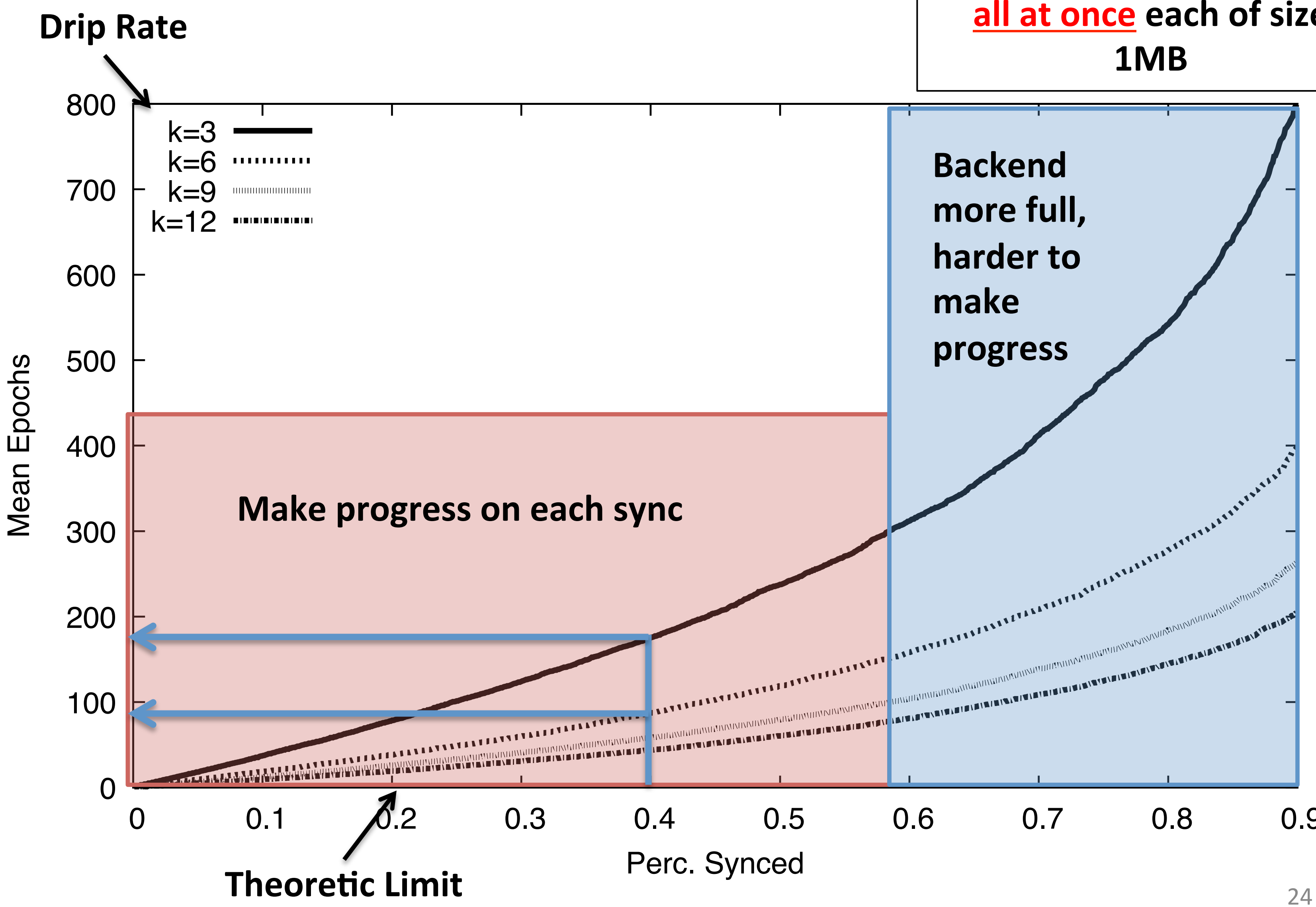
- How do we recognize if data is stale?
  - Perform a lookup in the superblock for the File-Id
  - If Block-Id is not listed it must be stale



Throughput

1024 Backend Blocks of size 1MB

Inserted 920 frontend files all at once each of size 1MB



How long does it take to clear the buffer?

> **Theorem 1.** *For a running* ObliviSync-RW *client with parameters* B, N, k *as above, let* m *be the total size (in bytes) of all non-stale data currently stored in the backend, and let* s *be the total size (in bytes) of pending write operations in the buffer, and suppose that* $m + s \leq NB/4$.
>
> *Then the expected number of sync operations until the buffer is entirely cleared is at most* $4s/(Bk)$.
>
> *Moreover, the probability that the buffer is not entirely cleared after at least* $\frac{4s}{Bk} + 18r$ *sync operations is at most* $\exp(-r)$.

- **A Buffer of size *s* will clear after O(*s*/(*Bk*)) operations**
  - *B*: Size of two split block, one backend storage file
  - *k*: is the drip rate, the number of size B files synced per epoch
- **Large percentage of backend blocks that should be empty**
  - 20% capacity or 80% empty for fast clearance
- ***Does not* depend on the distribution of file sizes**