Protecting Private Data in the Cloud: A Path Oblivious RAM Protocol

Nathan Wolfe and Ethan Zou
Mentors: Ling Ren and Xiangyao Yu
Fourth Annual MIT PRIMES Conference
May 18, 2014

Outline

1. **Background**
2. What is Oblivious RAM?
3. New Features
4. Evaluations
5. Future Work

Dropbox

Dropbox Security Problems

- Dropbox matches your files with other users' files to save space
  Encryption
- The federal government can compel Dropbox to release data
  Encryption
- Dropbox can see what files you change (access pattern)
  Oblivious RAM

ORAM: The Solution

- added layer of encryption on client’s end (1 & 2)
- obfuscation of access pattern and access type (3)

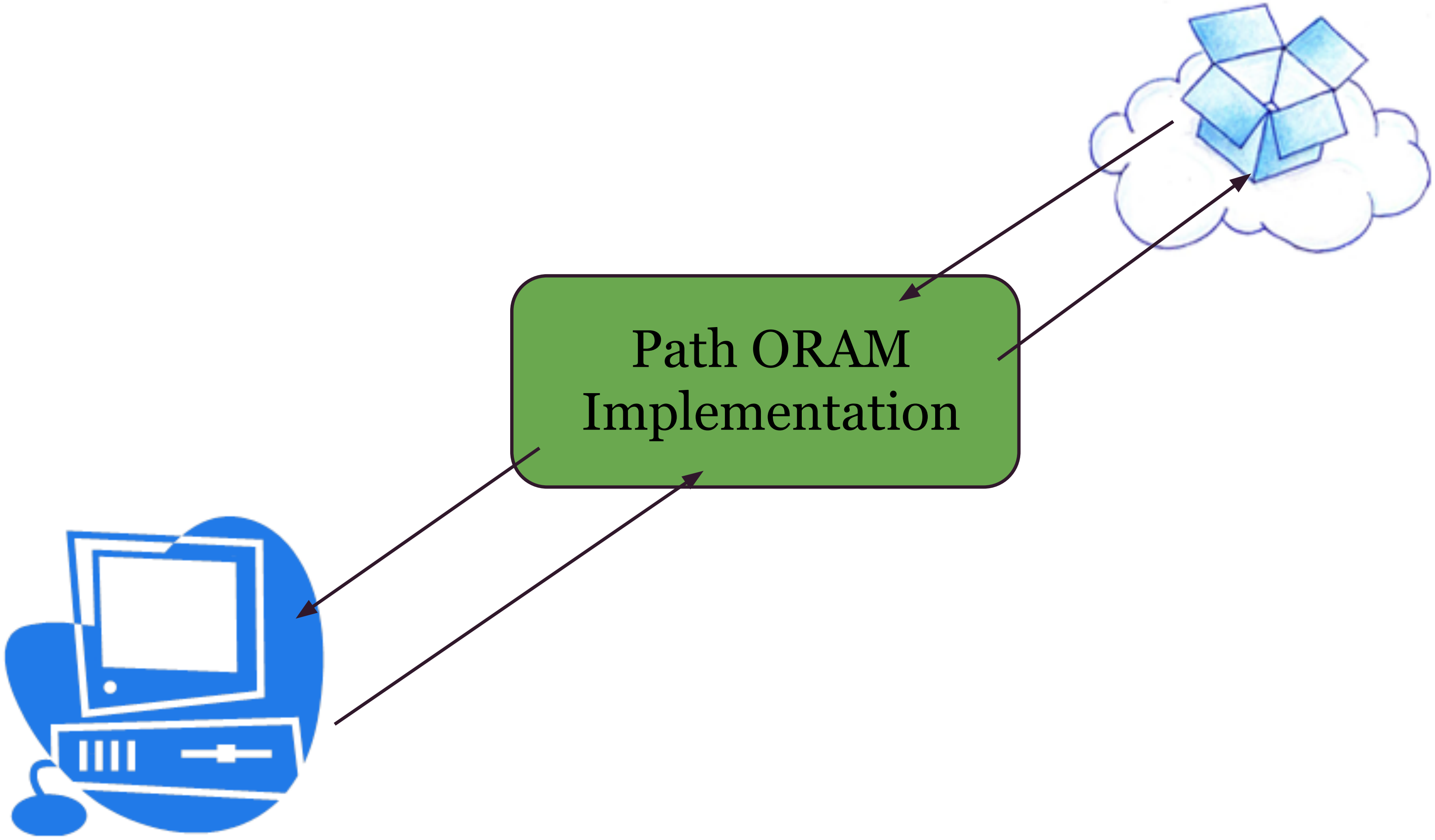
Outline

1. Background
2. **What is Oblivious RAM?**
3. New Features
4. Evaluations
5. Future Work

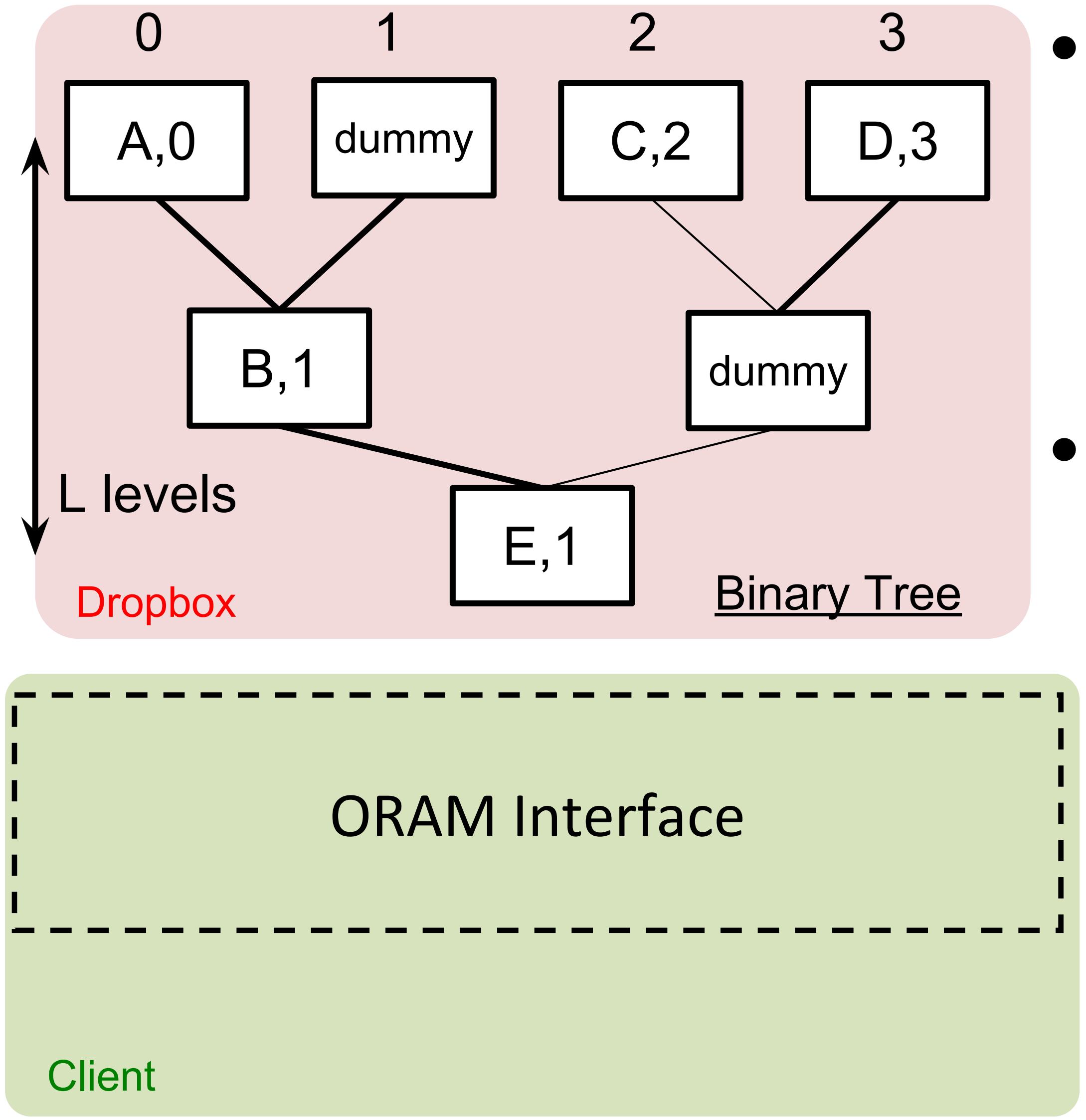
Oblivious RAM

- Naïve ORAM
    - Access all the data blocks for each memory access

Our Design: The Big Picture

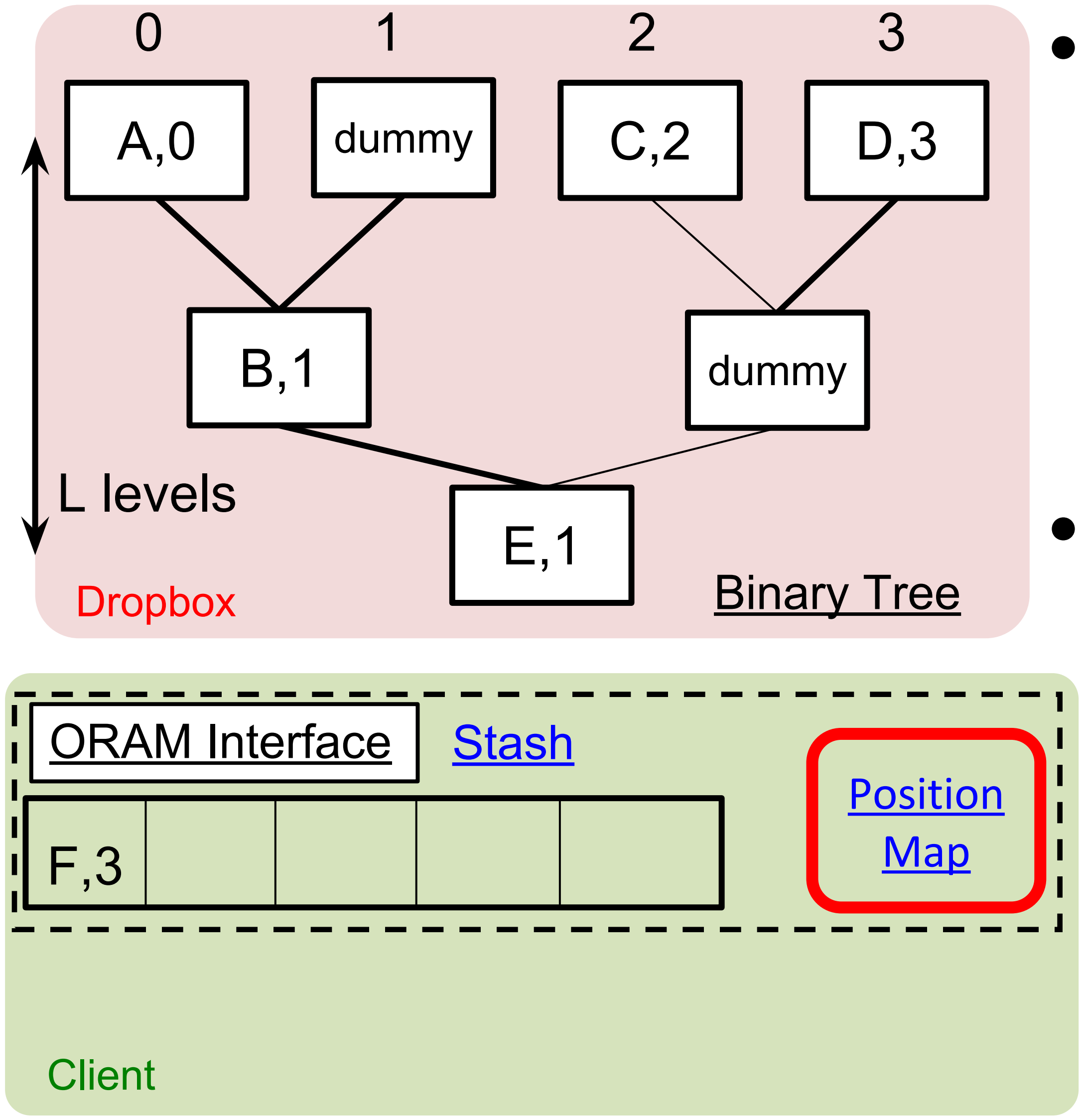


Path ORAM



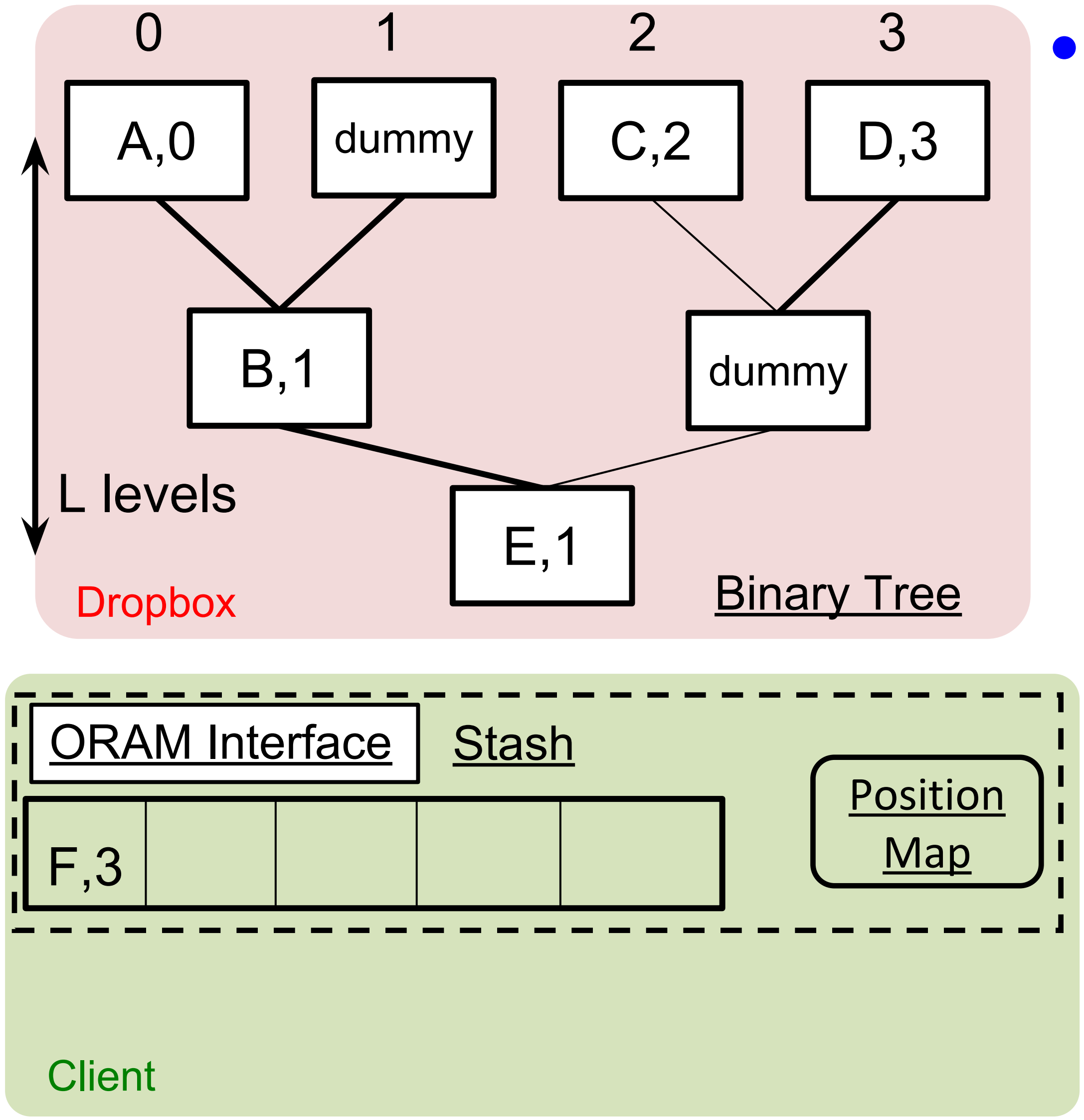
- Path ORAM is organized as a binary tree.
- Unoccupied nodes are filled with dummy blocks
  - Dummy and real blocks are indistinguishable after encryption

Path ORAM



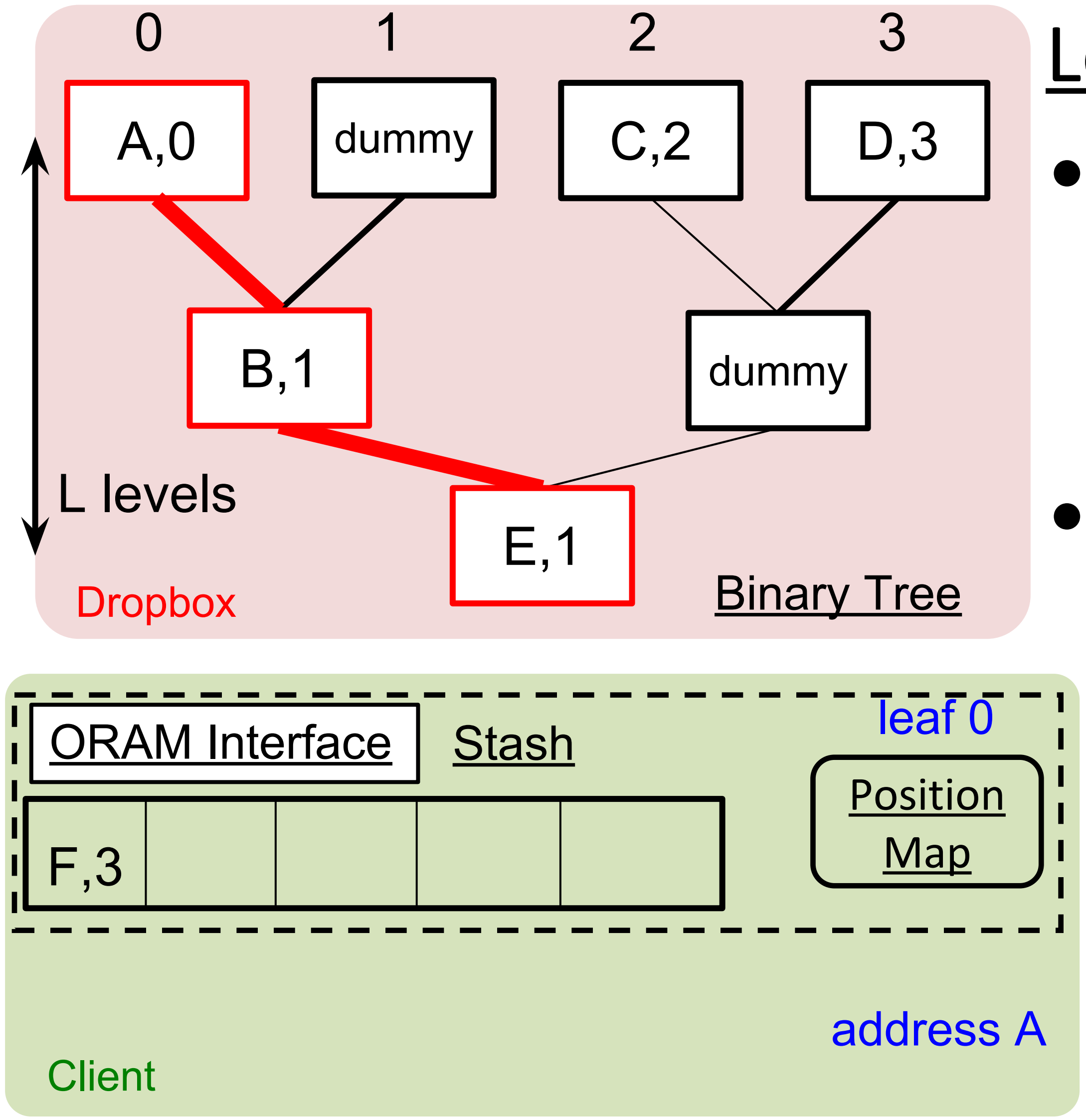
- Stash
  - A small list of data blocks
  - Background eviction prevents stash overflow
- Position Map
  - maps each program address to a ***random*** leaf

Path ORAM



- <span>Path ORAM invariant:</span> *If block a is mapped to leaf s, then a is stored*
  - *along the path from root to leaf s,* ***or***
  - *in the stash within the ORAM interface.*

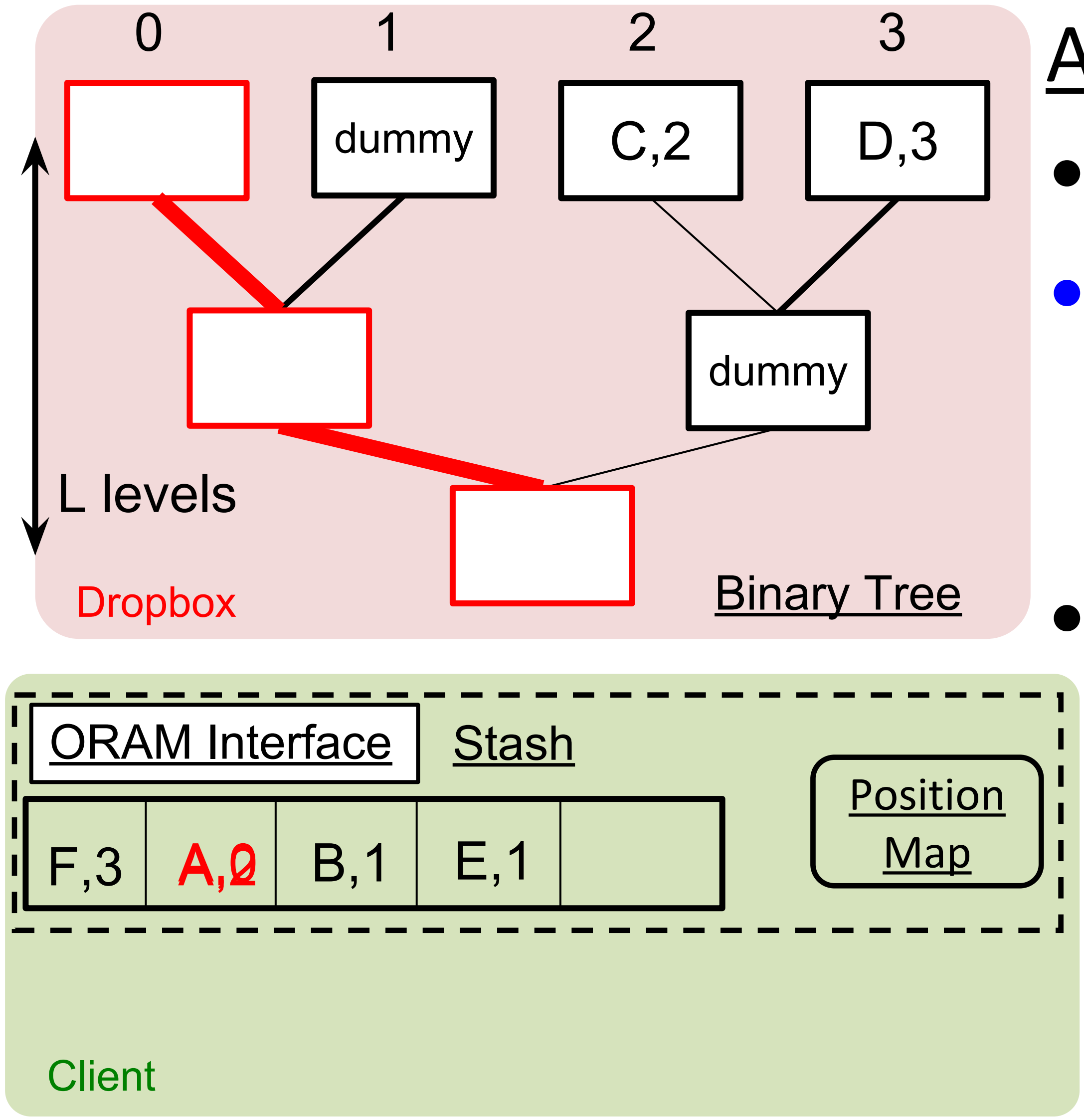
Path ORAM Example



Load Block A

- Lookup position map,
  s = PosMap(A)
- Load the path into stash

Path ORAM Example



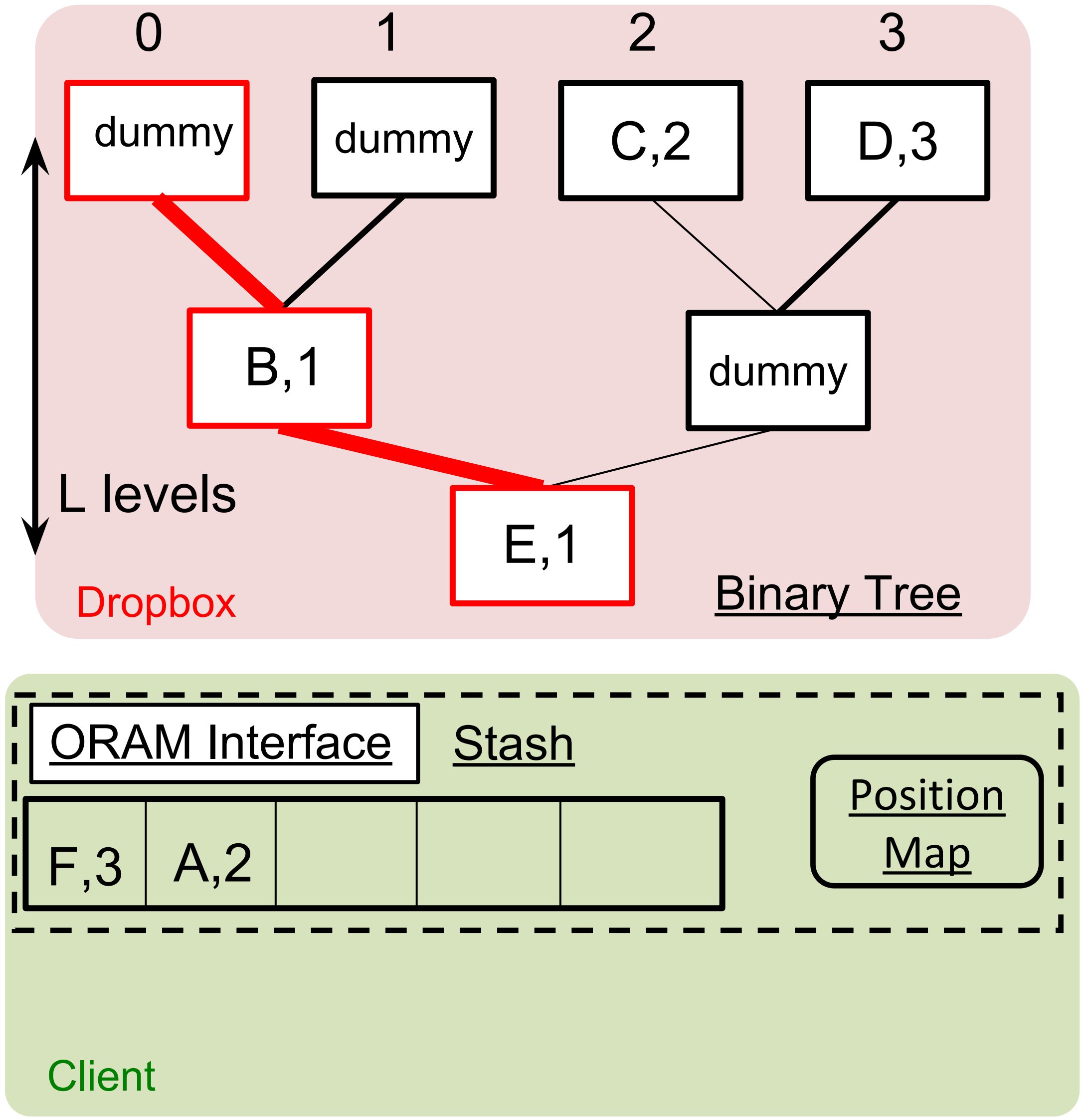
<u>Access and Remap</u>

- Read/Update Block A
- Remap A to a random leaf.

  PosMap(A) = rand()

- Each path ORAM access will access a random leaf

Path ORAM Example



Write Back

- Each block a_i in the stash is
  - written back to the tree, ***or***
  - stays in the stash

Outline

1. Background
2. What is Oblivious RAM?
3. **New Features**
4. Evaluations
5. Future Work

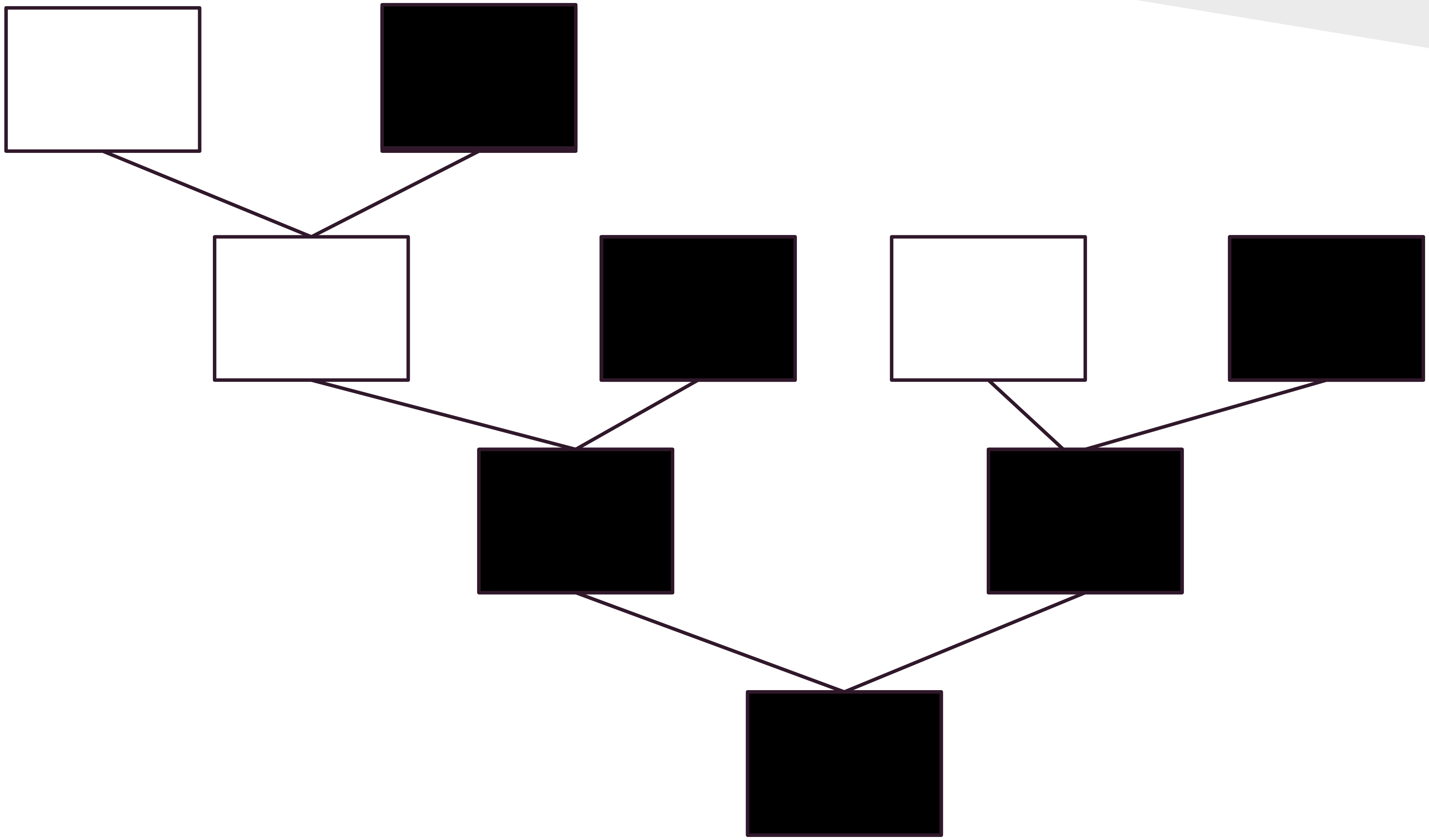
New Features

1. Dynamic Tree Growing/Shrinking
2. User File System
3. Multi-Computer

Dynamic Growing/Shrinking

- Dropbox
  - limited space, subscribe for additional
  - possibility to store unsecured files alongside
- Saves space when the tree is unnecessarily large
- Prevents overflow of the tree if too much data

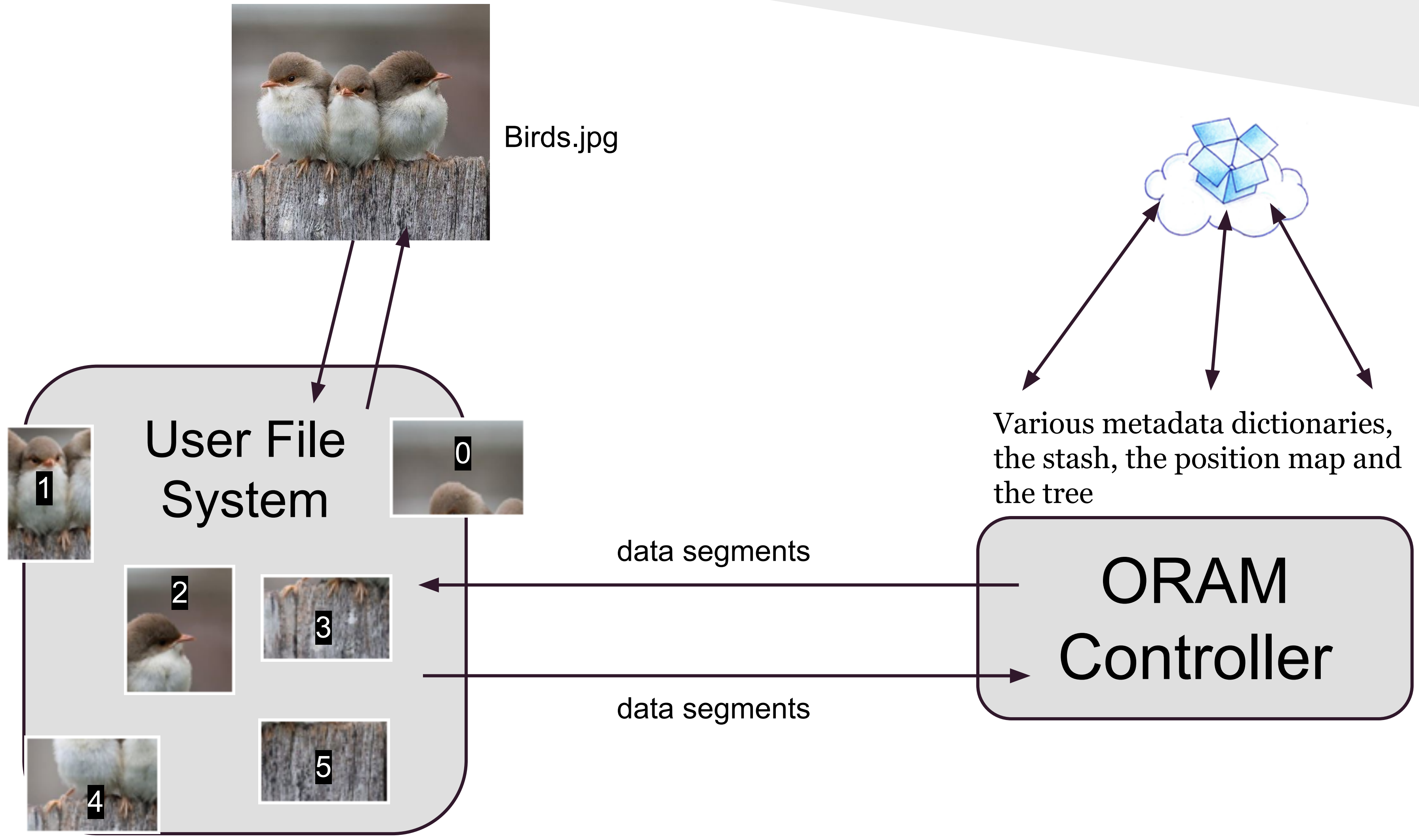
Dynamic Growing/Shrinking



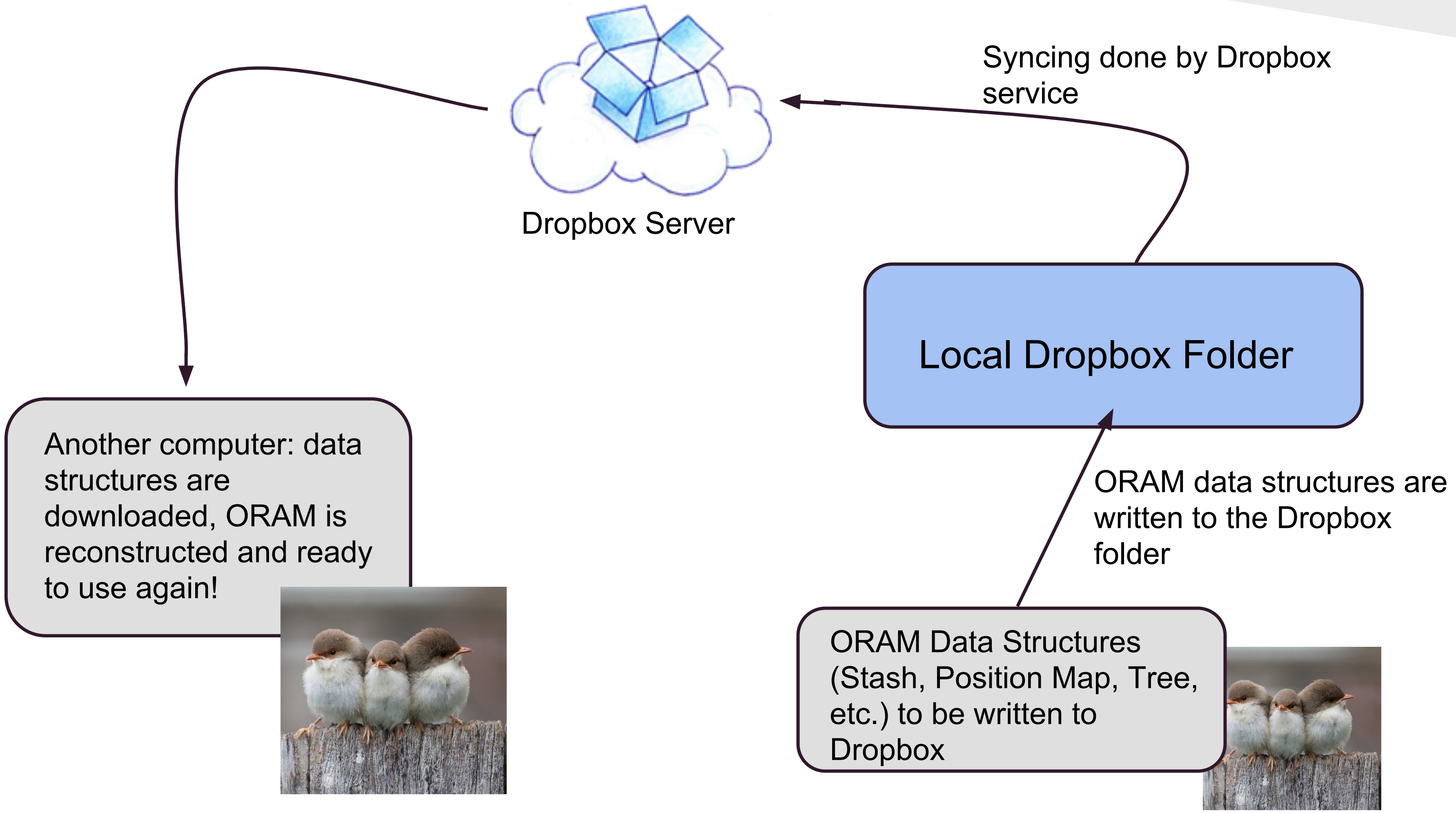
User File System

- allows writing of files of different sizes
- partitions files into manageable chunks and assigns each data segment with a unique segment ID
- writes/reads to and from the ORAM controller

User File System



Multi-Computer



Syncing done by Dropbox service

Dropbox Server

Local Dropbox Folder

ORAM data structures are written to the Dropbox folder

ORAM Data Structures (Stash, Position Map, Tree, etc.) to be written to Dropbox

Another computer: data structures are downloaded, ORAM is reconstructed and ready to use again!

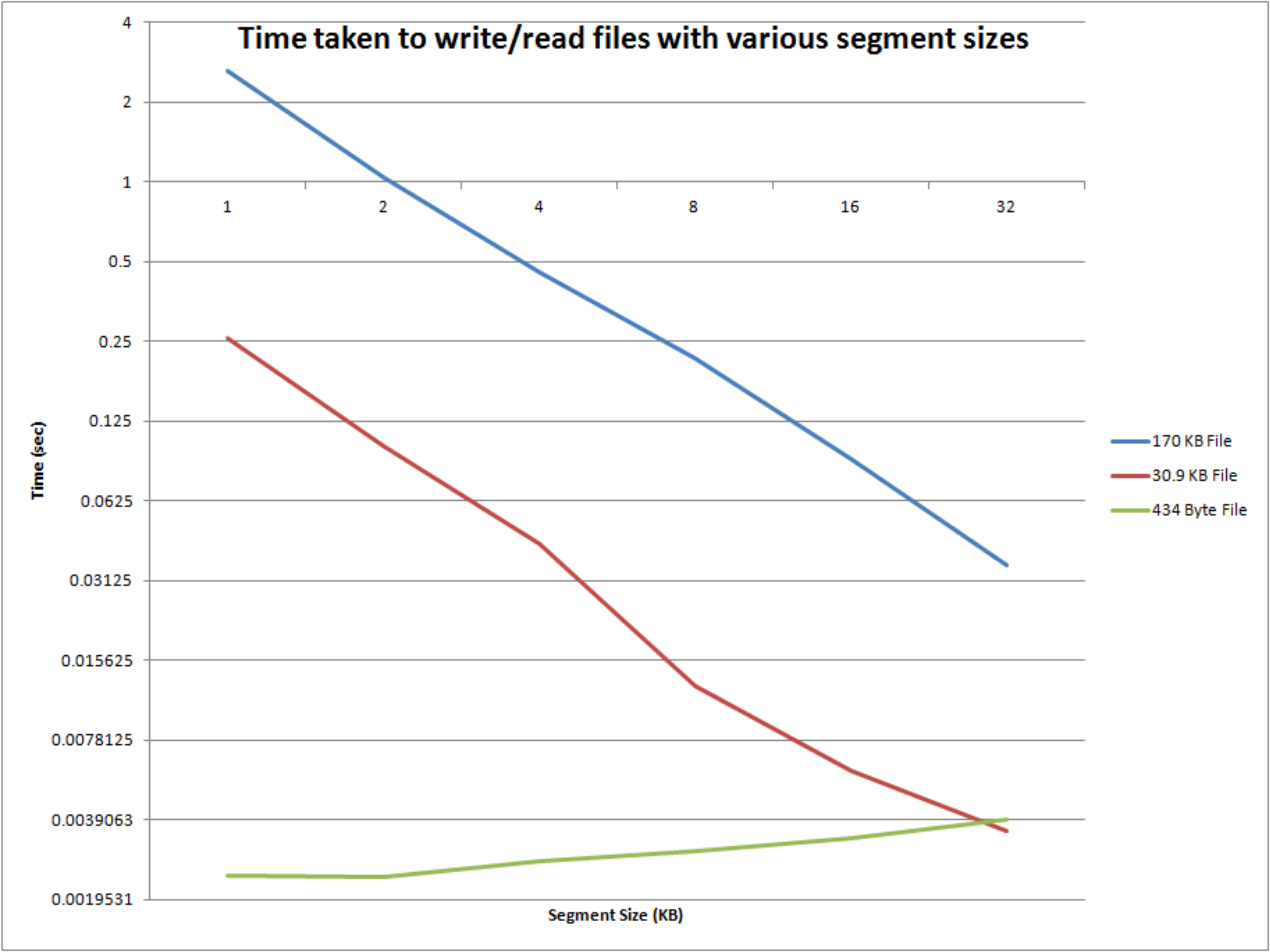
Outline

1. Background
2. What is Oblivious RAM?
3. New Features
4. **Evaluations**
5. Future Work

With ORAM vs Without it

File Type/Size	Without ORAM	With ORAM	With vs. Without
Photo (30.9 KB)	0.0009 sec (34,333 KB/sec)	0.045869 sec (674 KB/sec)	51x slower
PDF (170 KB)	0.0011576 sec (146,856 KB/sec)	0.5202 sec (327 KB/sec)	449x slower
Video (64.5 MB)	0.17809 sec (370,869 KB/sec)	2131.38207 sec (31 KB/sec)	11,964x slower

Parameters: z = 3, segment size = 4 KB


Time taken to write/read files with various segment sizes
Time (sec)
4
2
1
0.5
0.25
0.125
0.0625
0.03125
0.015625
0.0078125
0.0039063
0.0019531
1
2
4
8
16
32
Segment Size (KB)
170 KB File
30.9 KB File
434 Byte File

Outline

1. Background
2. What is Oblivious RAM?
3. New Features
4. Evaluations
5. **Future Work**

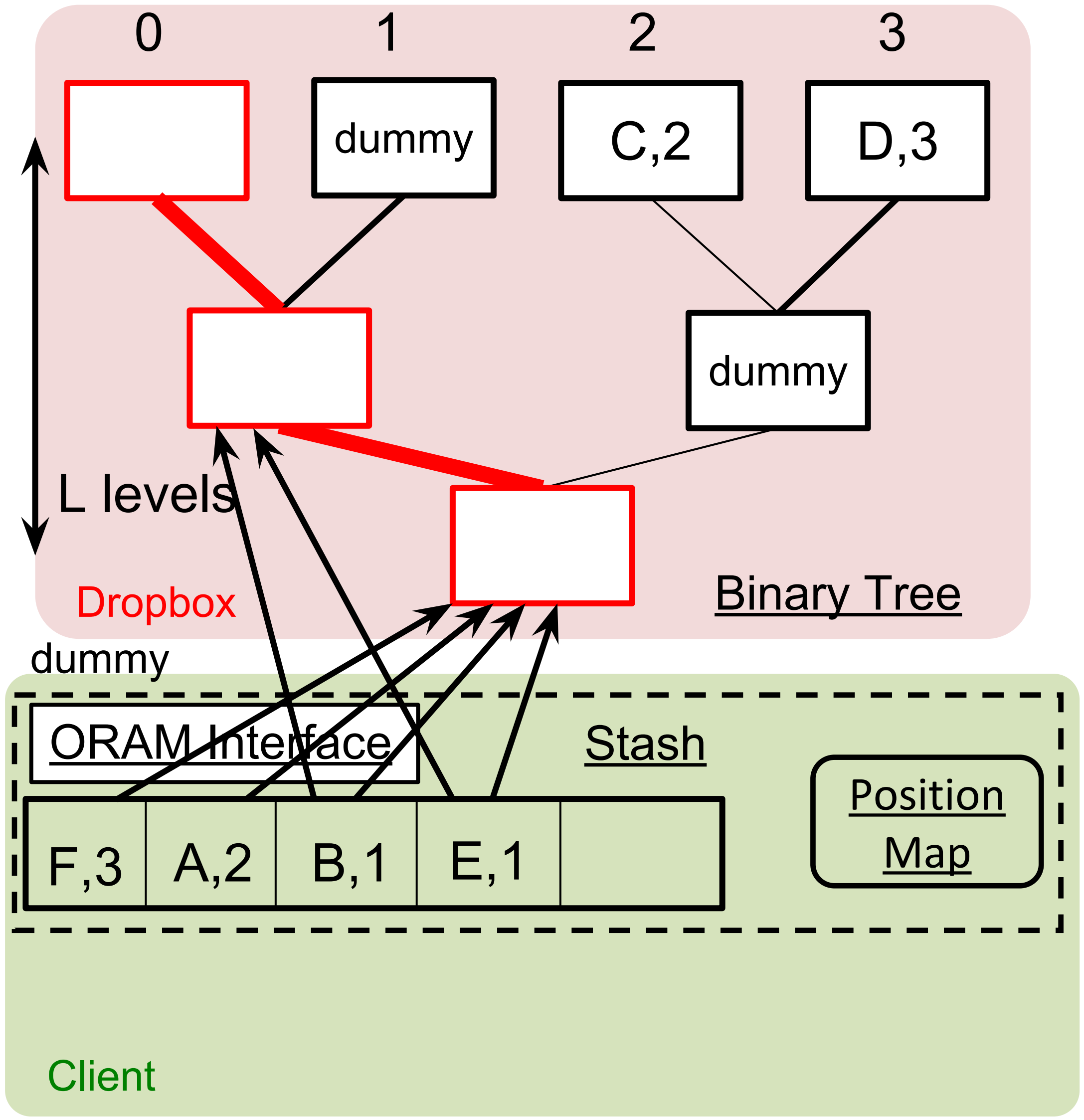
Future Research

- Software package
- Crash recovery
- User interface
    - graphics
    - directories
- Optimizations
    - hybrid ORAM
    - dynamic segment size
    - multi-block accesses

Acknowledgements

- Our mentors, Ling and Xiangyao, for their guidance and insight
- Professor Srini Devadas for suggesting the project and encouraging us along the way
- MIT PRIMES for making this research possible
- Our parents for their continuous support (and for transportation :))

Stash Eviction



Write Back

- Each block a_i in the stash is
  - written back to the common subpath of the accessed path and *PosMap(a_i)* as high as possible, ***or***
  - stays in the stash
- overhead = $2 \times Z \times L$