ViewBox
Integrating Local File Systems with Cloud Storage Services

Yupu Zhang[+], Charlotte Dragga[+*],
Andrea Arpaci-Dusseau[+], Remzi Arpaci-Dusseau[+]

[+]University of Wisconsin – Madison
[*]NetApp, Inc.

Personal Cloud Storage Services

- Exploding in popularity
  - Numerous providers: Dropbox, Google Drive, SkyDrive …
  - Large user base: Dropbox has more than 100 million users

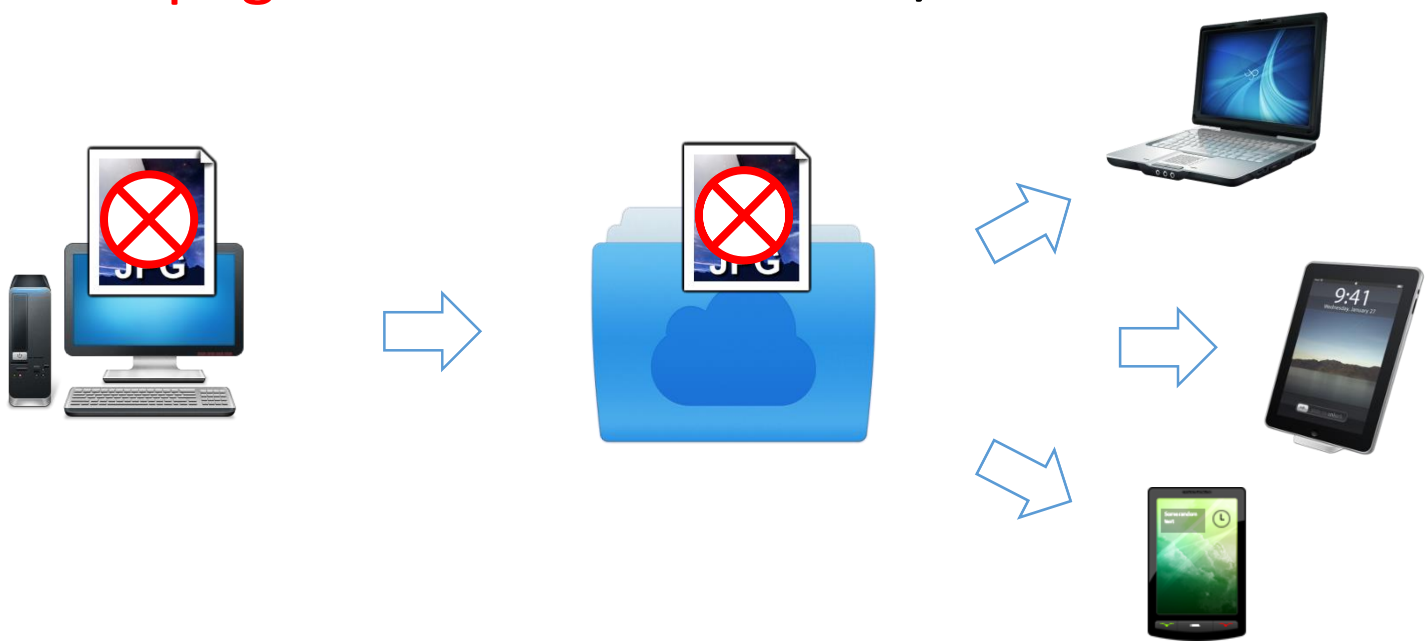
- Promising benefit
  - Reliable backup on the cloud
  - Automatic synchronization across clients/devices

There are so many copies…
My data must be safe...

Really?

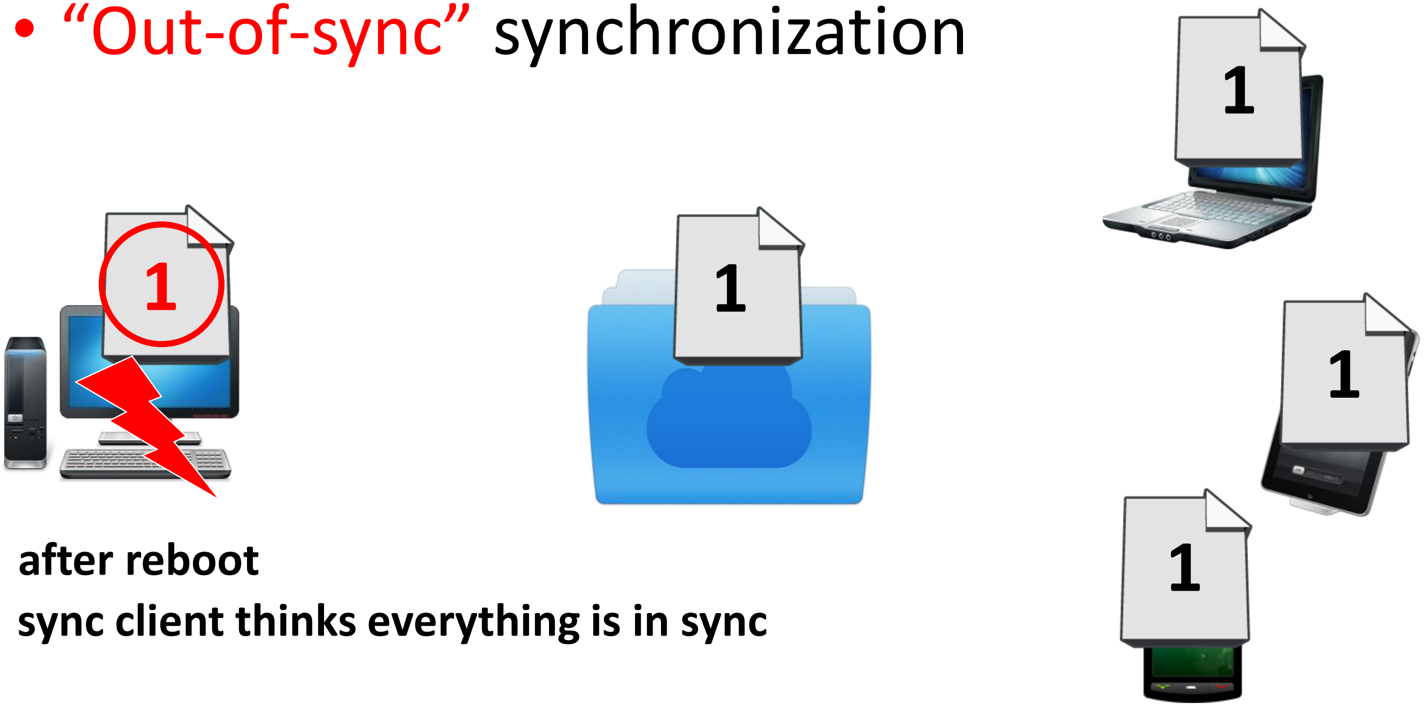
Is Your Data Really Safe?

- Data corruption
  - Uploaded from local machine to cloud
  - Propagated to other devices/clients



Is Your Data Really Safe?

- Crash inconsistency
  - Inconsistent data ends up everywhere
  - “Out-of-sync” synchronization



after reboot
sync client thinks everything is in sync

Is Your Data Really Safe?

- Causal inconsistency
  - Files are uploaded out of order
  - Cloud state does not match a valid FS state

Many copies do **NOT** make your data safe

Why? – File Systems

- Local file system is the weakest link
- Corruption and inconsistency are exposed

file system state ≠ correct state ✗

Why? – Sync Services

- Ad-hoc synchronization is harmful

- Sync client sees what regular application sees, but *not what file system sees*

cloud state ≠ file system state

✘

Can we achieve

cloud state = file system state = correct state

with existing systems?

Our solution: ViewBox

integrated file system and cloud storage

- Local detection + Cloud-aided recovery
  - Rely on strong local file system to detect problems
  - Utilize cloud data to recover from local failures

➡ file system state = correct state

- Orchestrated synchronization based on views
  - In-memory snapshots of valid file system state
  - *Sync client sees what file system sees*

➡ cloud state = file system state

Results

- ViewBox runs on top of existing systems
  - Enhance ext4 with data checksumming
  - Work with unmodified Dropbox and modified Seafile
- ViewBox provides better reliability
  - No global data pollution
  - Automatic recovery with cloud data
- ViewBox incurs minimal overhead
  - Less than 5% overhead for most workloads
  - Up to 30% reduction of synchronization time in some cases

Outline

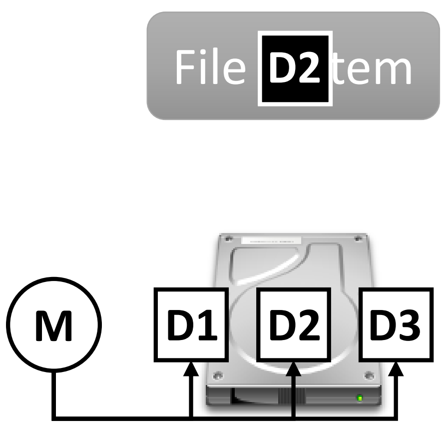
- Introduction
- Motivation
  - **Data Corruption**
  - Crash Inconsistency
  - Causal Inconsistency
- Design and Implementation
- Evaluation
- Conclusion

Experiment Setup

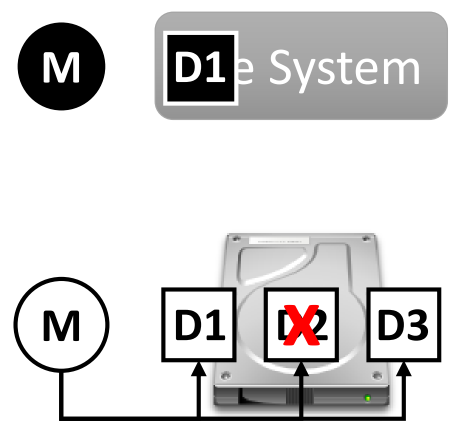
- File systems (on Linux)
  - ext4 w/ ordered journaling
  - ext4 w/ data journaling
  - ZFS
- Synchronization services
  - Dropbox
  - ownCloud
  - Seafile

Data Corruption – Method

- Inject corruption to a synchronized file on disk
- Perform various workloads
  - data writes
- Check if corruption is propagated



Regular Write



Fault Injection

Data Corruption – Results

L: local corruption G: global corruption D: detected R: recovered

FS	Service	Data Writes	Metadata Changes		
			mtime	ctime	atime
ext4	Dropbox	L G	L G	L G	L
	ownCloud	L G	L G	L	L
	Seafile	L G	L G	L G	L G

Corruption is uploaded even when there is no data change

Data Corruption – Results

L: local corruption G: global corruption D: detected R: recovered

FS	Service	Data Writes	Metadata Changes		
			mtime	ctime	atime
ZFS	**Dropbox**	D	D	D	L
	ownCloud	D	D	L	L
	Seafile	D	D	D	D

Corruption is detected when it is read

No automatic recovery using cloud data

Data Corruption – Lessons

- Where do synchronization services fail?
  - Rely on file-level monitoring mechanism, e.g., inotify
  - Have to read whole file to identify changes
  - Cannot tell between legitimate changes and corruption

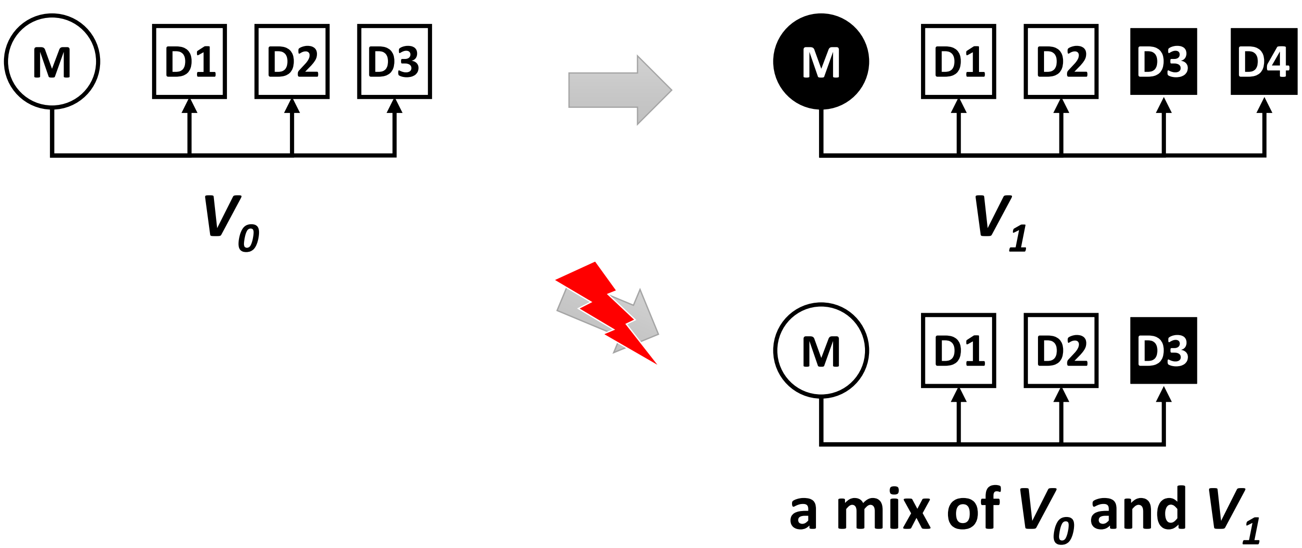
- Where do file systems fail?
  - Many file systems do not checksum data

Outline

- Introduction
- Motivation
  - Data Corruption
  - **Crash Inconsistency**
  - Causal Inconsistency
- Design and Implementation
- Evaluation
- Conclusion

Crash Inconsistency - Method

- A file is synchronized at V_0 on disk and cloud
- Update the file from V_0 to V_1
- Inject a crash and observe sync client's behavior



Crash Inconsistency – Results

YES: occurred   NO: did not occur   N/A: no result

Erratic behaviors

FS	Service	Upload Local Version	Download Cloud Version	Out of Sync
ext4 (ordered)	Dropbox	YES	NO	YES
	ownCloud	YES	YES	YES
	Seafile	N/A	N/A	N/A

Inconsistent local version gets uploaded

Fails to synchronize local changes

Crash Inconsistency – Results

YES: occurred NO: did not occur N/A: no result

FS	Service	Upload Local Version	Download Cloud Version	Out of Sync
ext4 (data) or ZFS	**Dropbox**	YES	NO	NO
	ownCloud	YES	YES	NO
	Seafile	YES	NO	NO

Local version is always consistent

May violate causal consistency

Crash Inconsistency – Lessons

- Where do synchronization services fail?
  - Depend on their own metadata tracking
  - Inconsistent with file system metadata upon crash

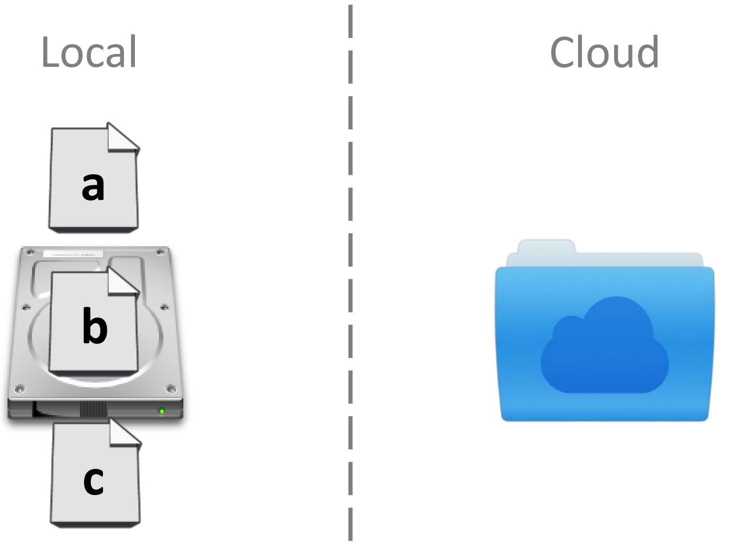
- Where do file systems fail?
  - Metadata journaling cannot provide data consistency

Outline

- Introduction
- Motivation
  - Data Corruption
  - Crash Inconsistency
  - **Causal Inconsistency**
- Design and Implementation
- Evaluation
- Conclusion

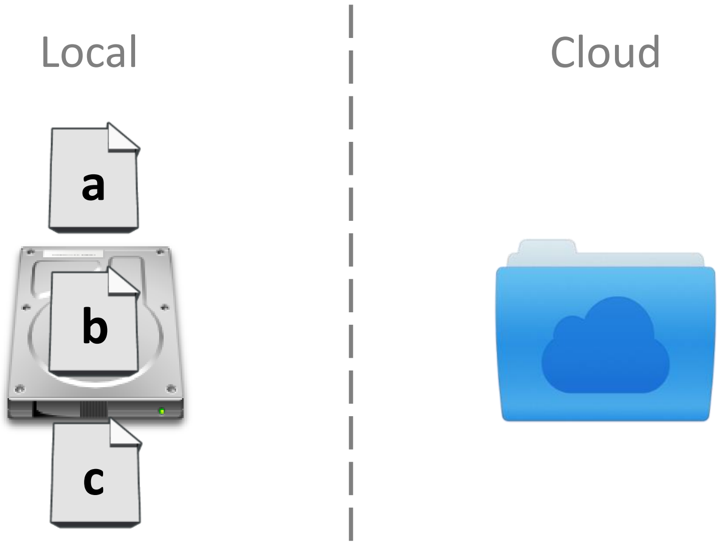
Causal Inconsistency – Method

- Write a series of files in a specified order
- See if these files are synchronized in correct order



Causal Inconsistency – Results

- The causal ordering can be violated in all three services on both ZFS and ext4



May not directly use data on cloud for recovery

Causal Inconsistency – Lessons

- Where do synchronization services fail?
  - Synchronize files out of order
- Where do file systems fail?
  - No efficient mechanism to provide a static and consistent view to sync services

Summary

- Both file systems and sync services are responsible for these failures
  - Many file systems lack strong reliability mechanisms

file system state ≠ correct state

  - What sync clients see is different from what local file systems see

cloud state ≠ file system state

cloud state ≠ file system state ≠ correct state

Summary (cont.)

File System	Corruption Detection	Crash Consistency	Causal Consistency	Recovery using Cloud
ext4 (metadata)	✘	✘	✘	✘
ext4 (data)	✘	✓	✘	✘
ZFS	✓	✓	✘	✘

- Not all problems can be avoided by switching to advanced file systems
- No automatic recovery with cloud data

Outline

- Introduction
- Motivation
- Design and Implementation
  - **ViewBox Overview**
  - Local Detection & Cloud-aided Recovery
  - View-based Synchronization
- Evaluation
- Conclusion

ViewBox Overview

- Local detection
  - No corruption/inconsistency is spread

ext4-cksum

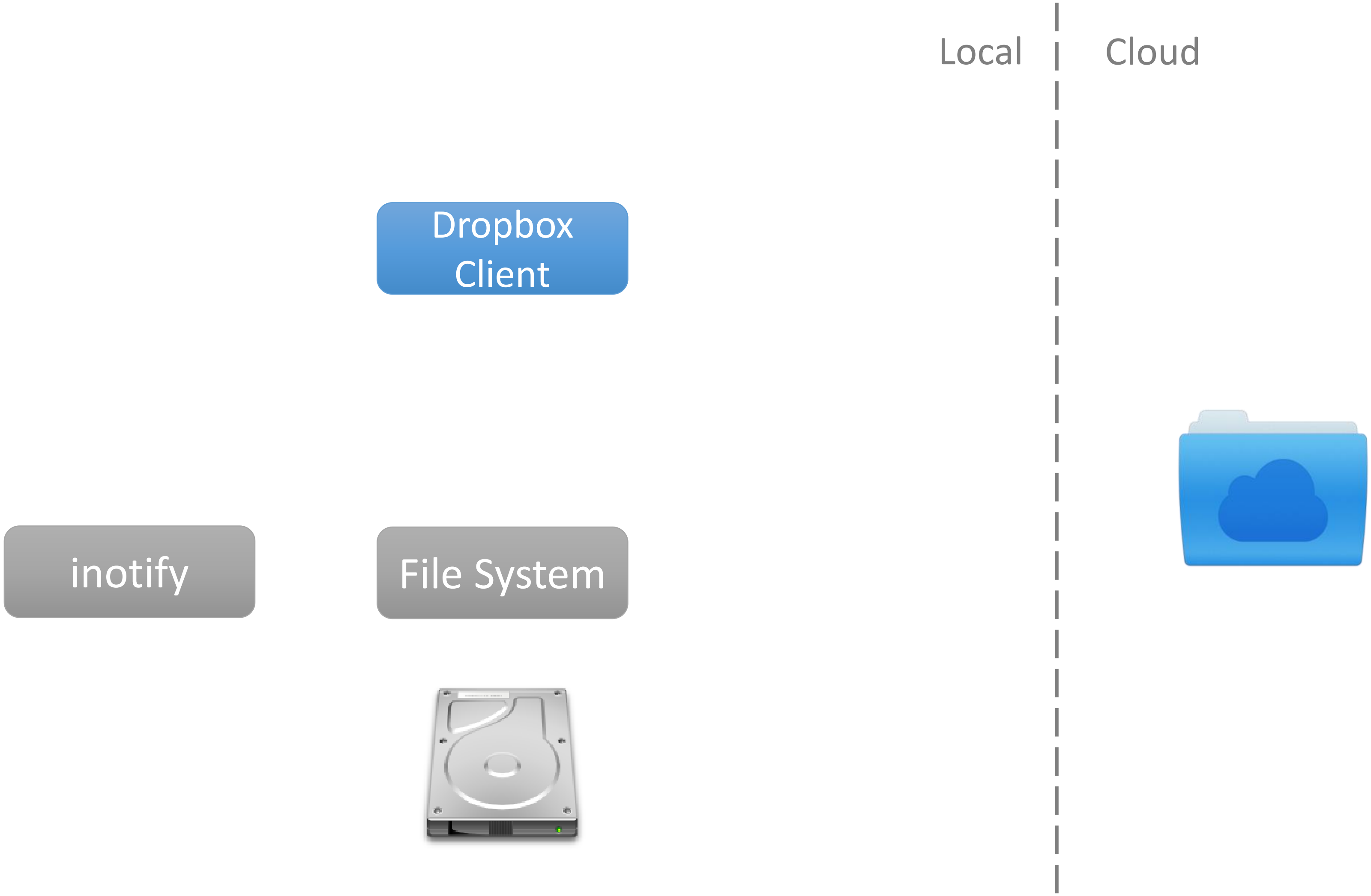
- Cloud-aided Recovery
  - Restore file system to correct state upon failure

Cloud Helper

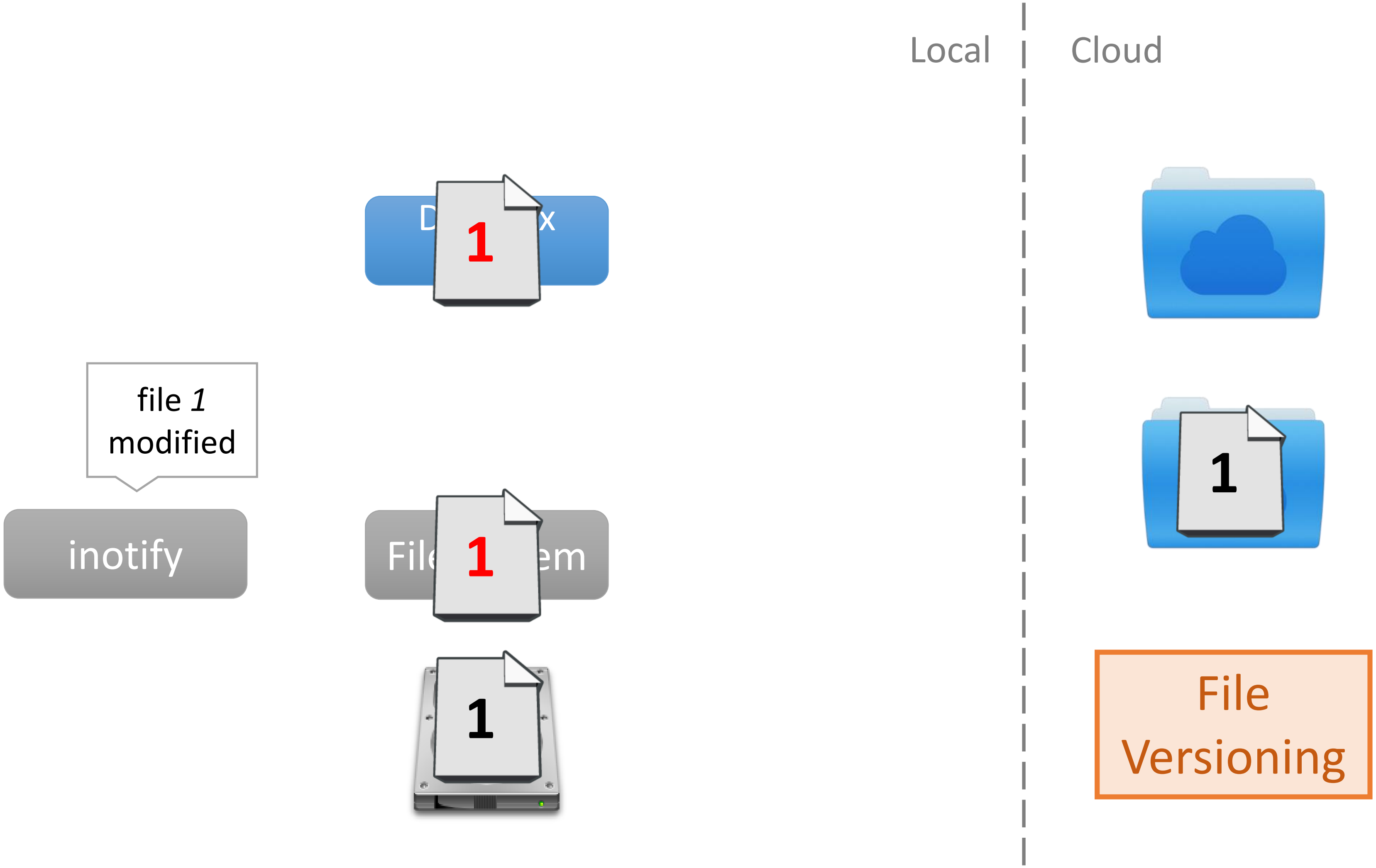
- View-based Synchronization
  - Present file system's view to sync service
  - Basis for consistency and correct recovery

View Manager

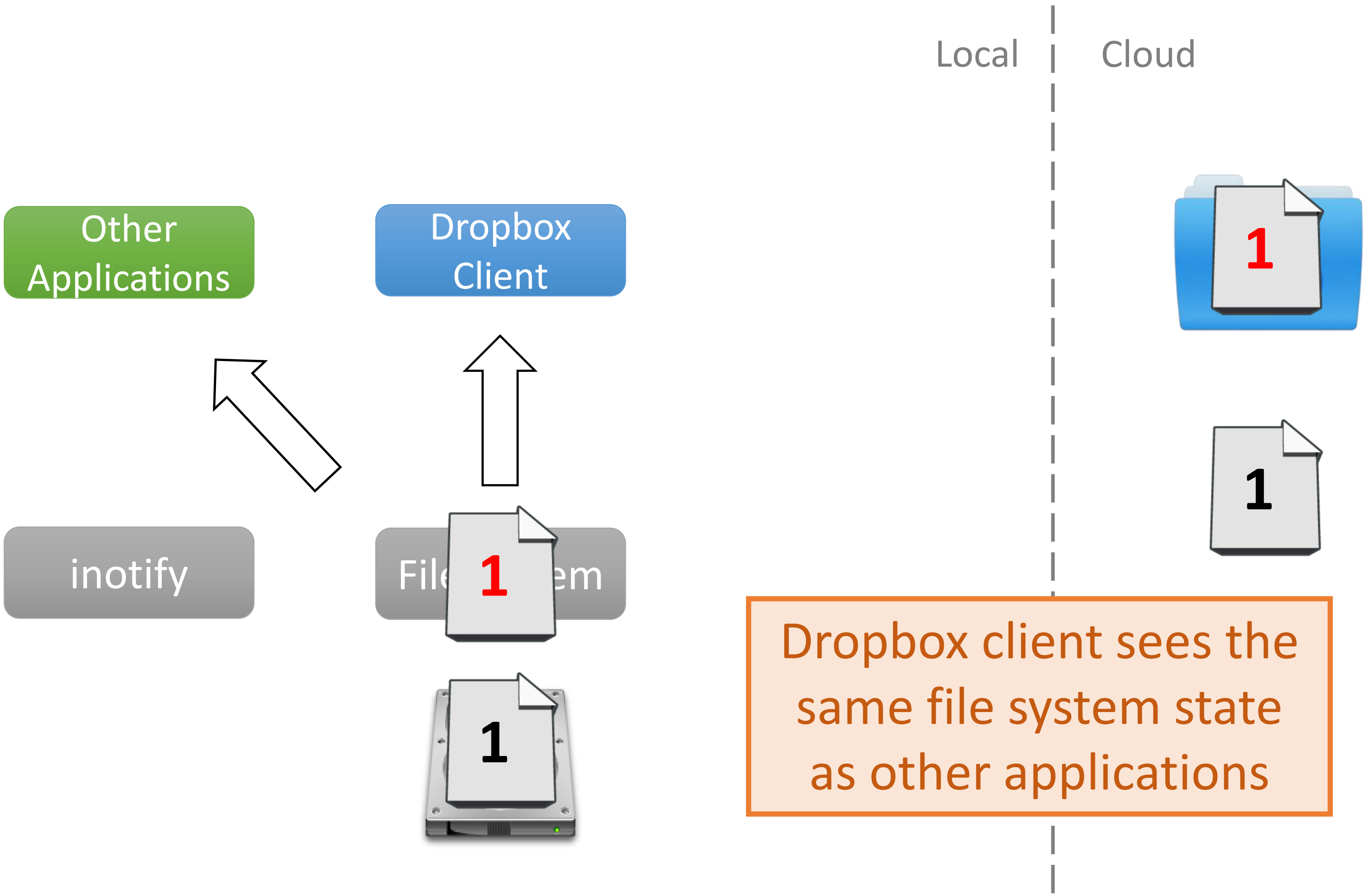
Dropbox Architecture



Dropbox Architecture

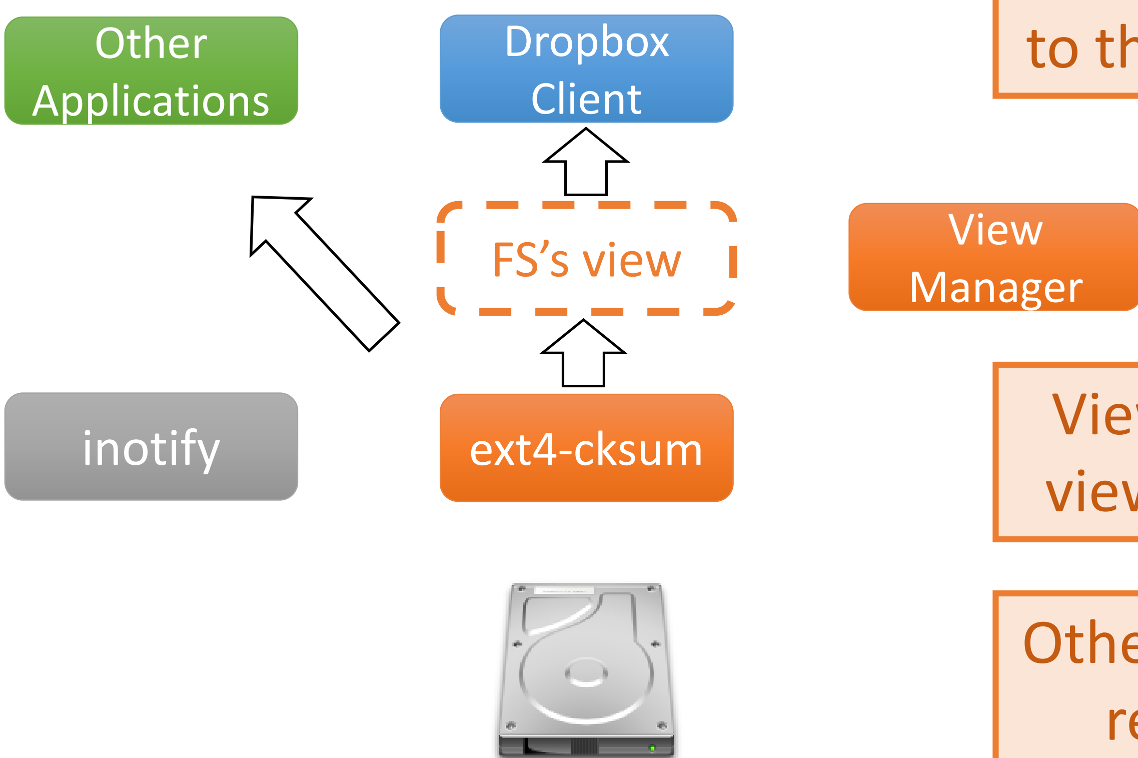


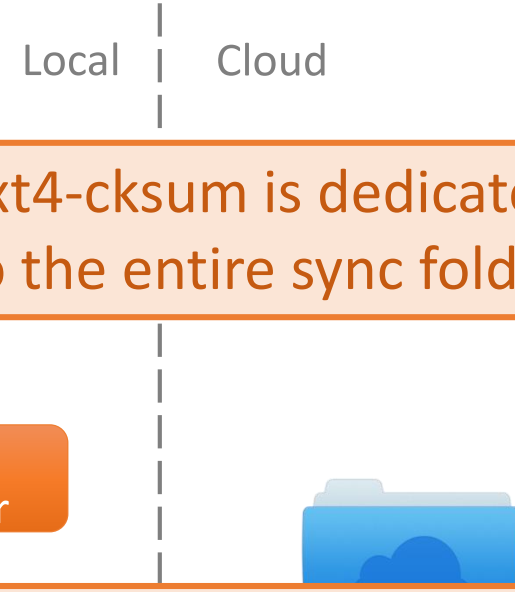
Dropbox Architecture



Dropbox client sees the same file system state as other applications

ViewBox Architecture



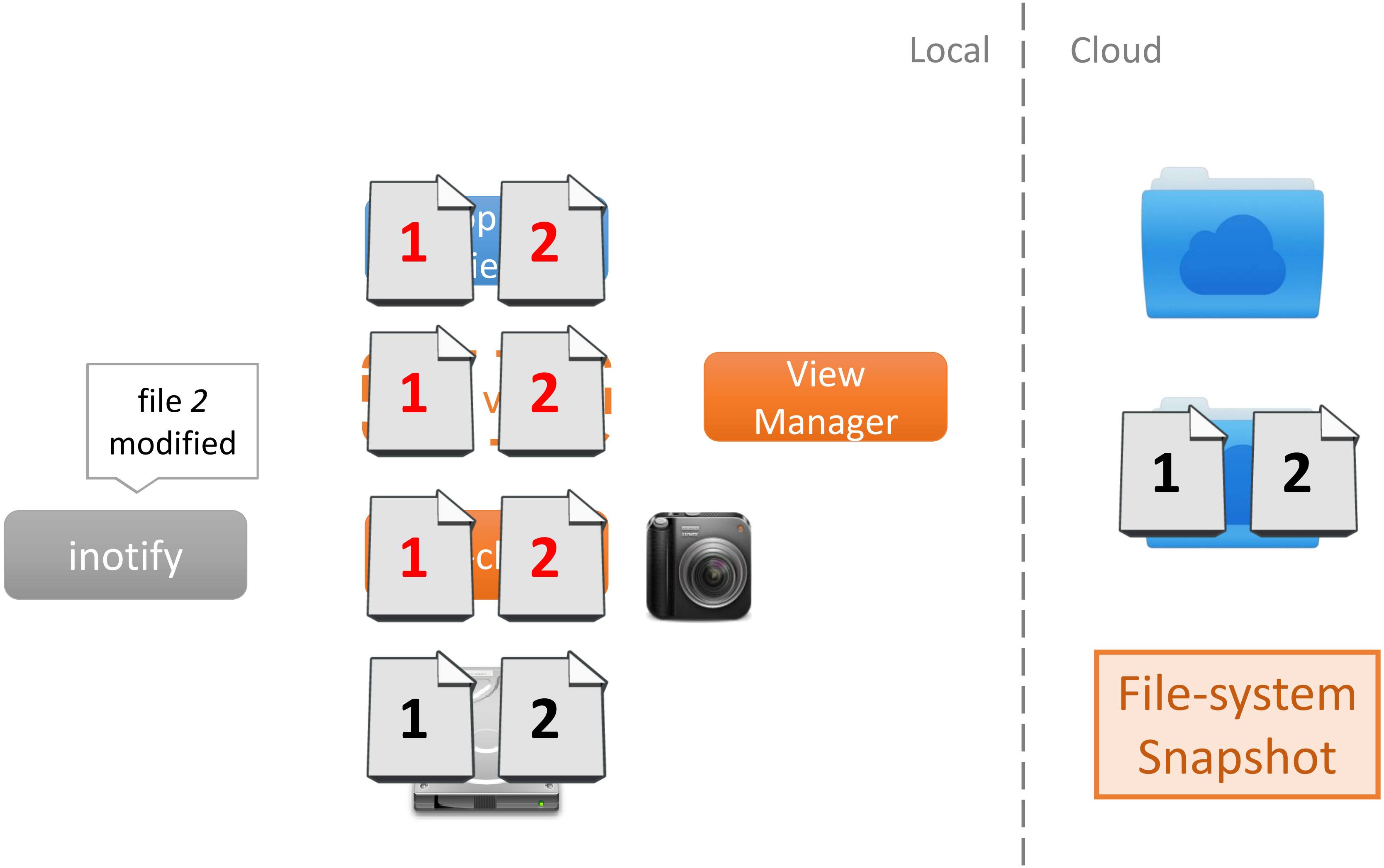


ext4-cksum is dedicated to the entire sync folder

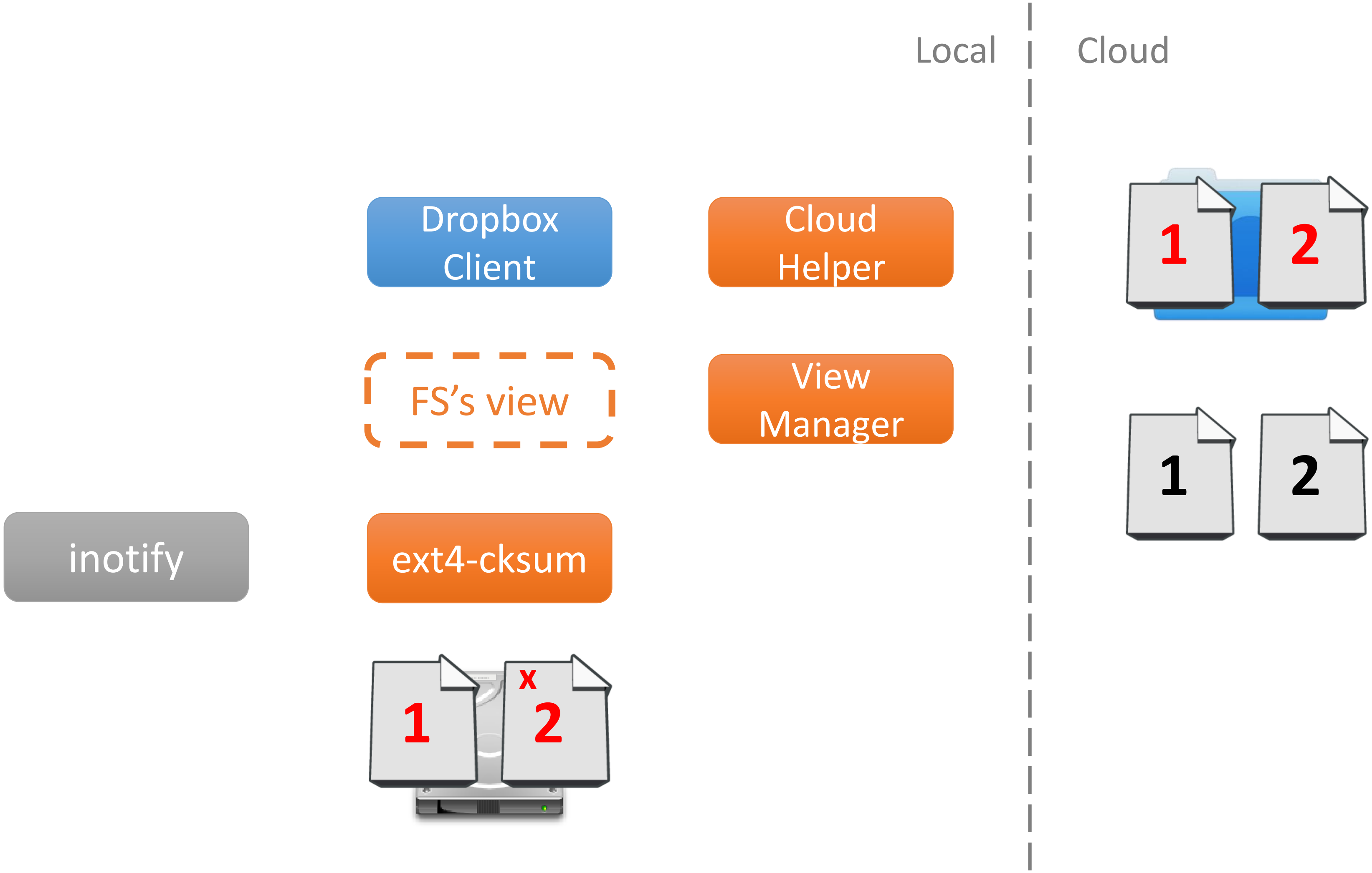
ViewBox presents FS's view to Dropbox client

Other applications' view remains the same

ViewBox Architecture



ViewBox Architecture



Outline

- Introduction
- Motivation
- Design and Implementation
  - ViewBox Overview
  - **Local Detection & Cloud-aided Recovery**
  - View-based Synchronization
- Evaluation
- Conclusion

ext4-cksum – Local Detection

Superblock	Group Descriptors	Block Bitmap	Inode Bitmap	Inode Table	Checksum Region	Data Blocks

- Checksum region
  - Pre-allocated space (~0.1% overhead)
    - 32-bit CRC checksum per 4KB block
    - 128KB checksum region for a 128MB block group
  - Each checksum maps to a data block in the block group
- Detect data corruption & inconsistency
  - More details in the paper

Cloud Helper – Cloud-aided Recovery

- A user-level daemon
  - Talks to local FS through ioctl
  - Communicates with the server through web API
- Upon data corruption
  - Fetches correct block from cloud
- After crash, two types of recovery
  - Recovers inconsistent files
  - Rolls back entire file system to the latest synced view

Outline

- Introduction
- Motivation
- Design and Implementation
  - ViewBox Overview
  - Local Detection & Cloud-aided Recovery
  - **View-based Synchronization**
- Evaluation
- Conclusion

View Manager – View-based Sync

- Create file system views
- Upload views to cloud through sync client

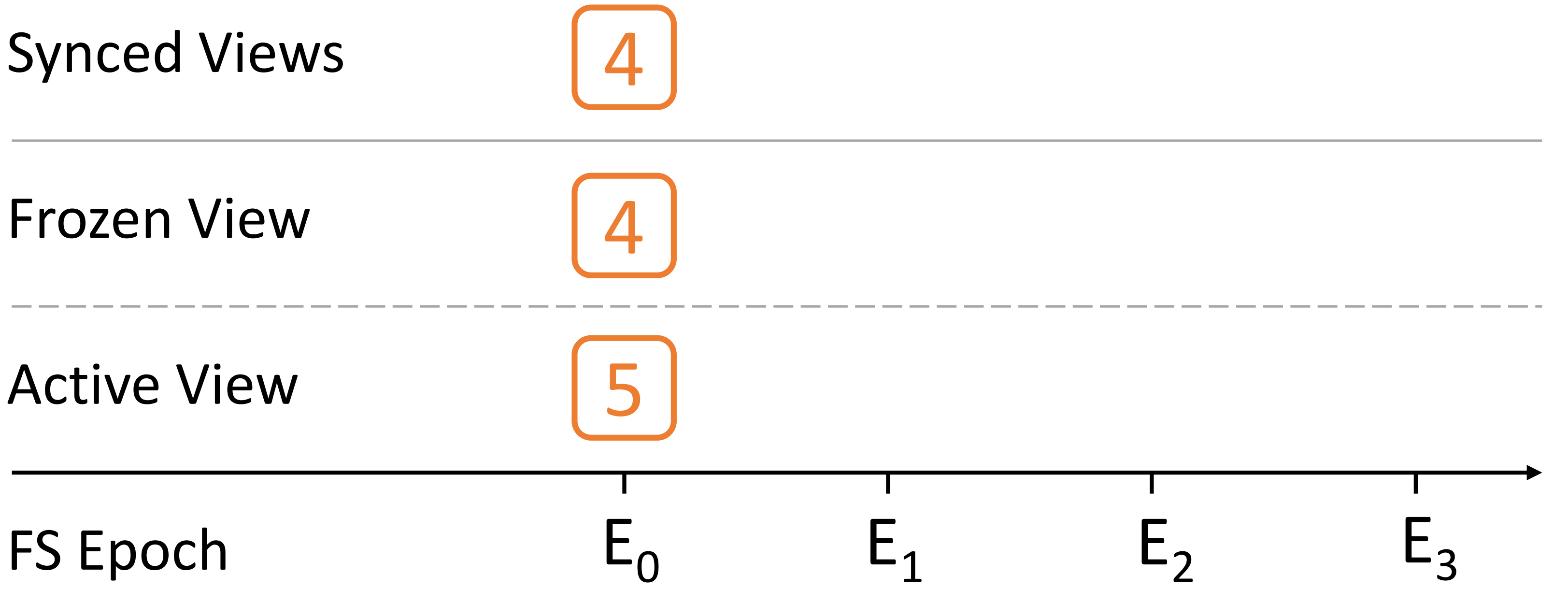
- Challenge 1 - How to provide consistency?
  - ext4-cksum still runs in ordered mode
  - Cloud journaling

- Challenge 2 - How to create views efficiently?
  - No support from ext4-cksum
  - Incremental snapshotting

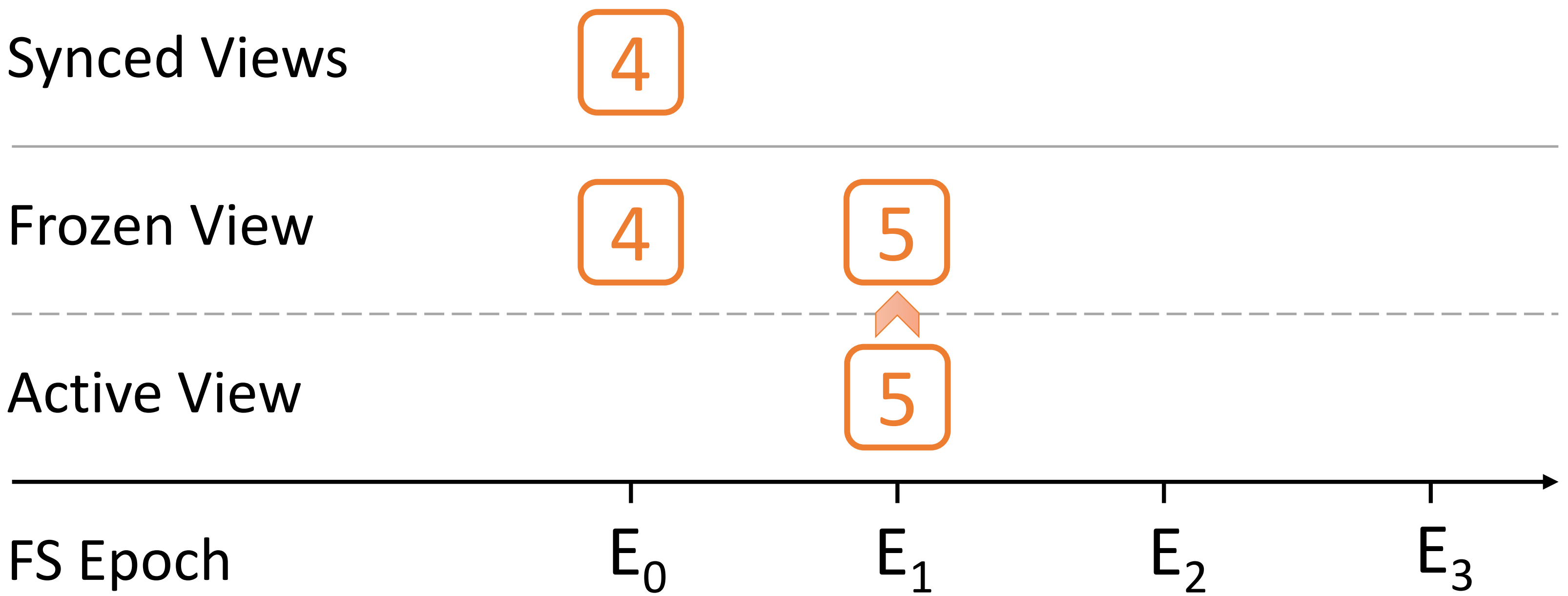
Challenge 1: How to Guarantee Consistency?

- Cloud journaling
  - Treat cloud storage as external journal
  - Synchronize local changes to cloud at FS epochs
    - i.e., when ext4-cksum performs a journal commit
- Three types of views
  - Active view (local) => Current FS state
  - Frozen view (local) => Last FS snapshot in memory
  - Synced views (on cloud) => Previously uploaded views
- Roll back to the latest synced view upon failure

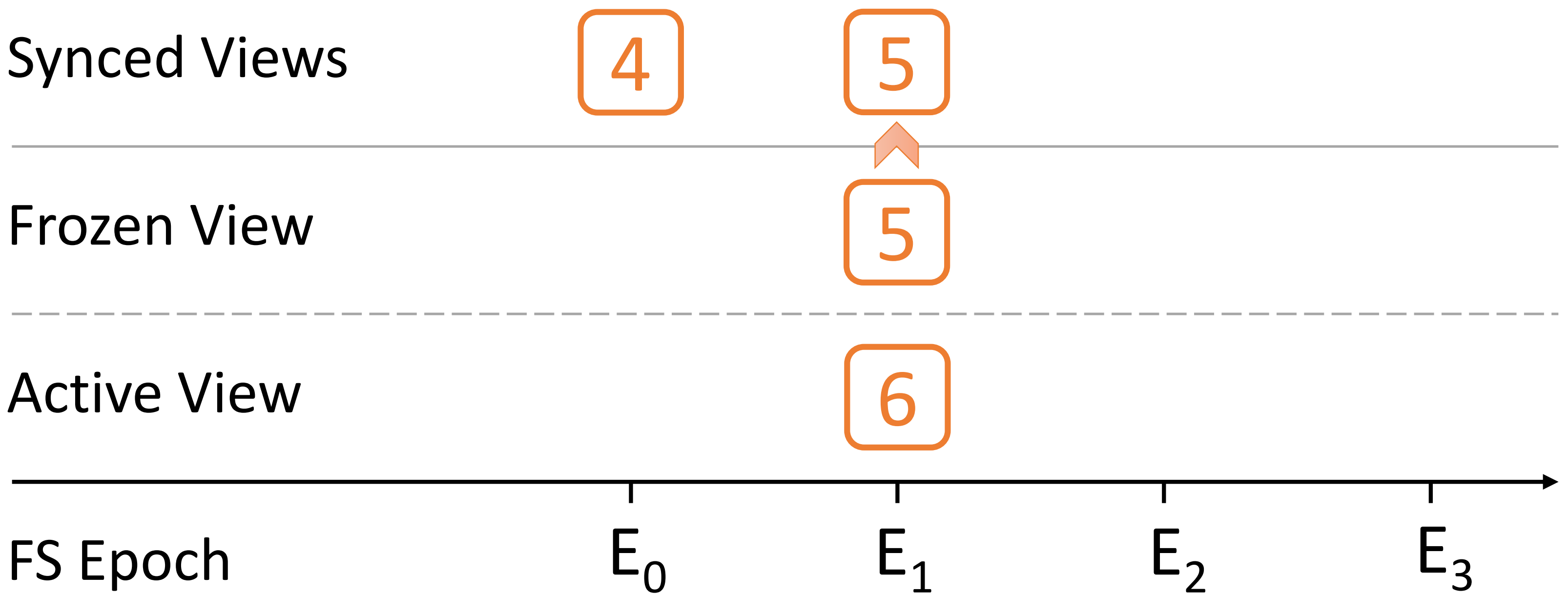
Cloud Journaling Example



Cloud Journaling Example

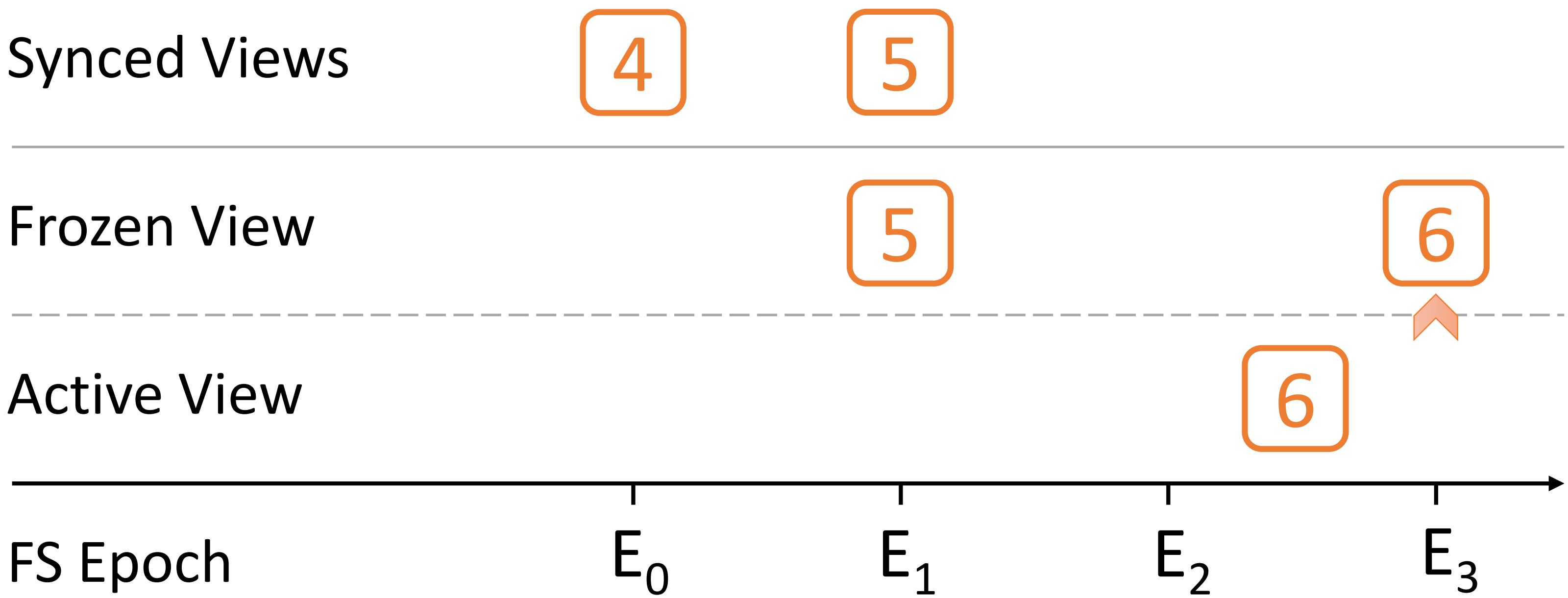


Cloud Journaling Example



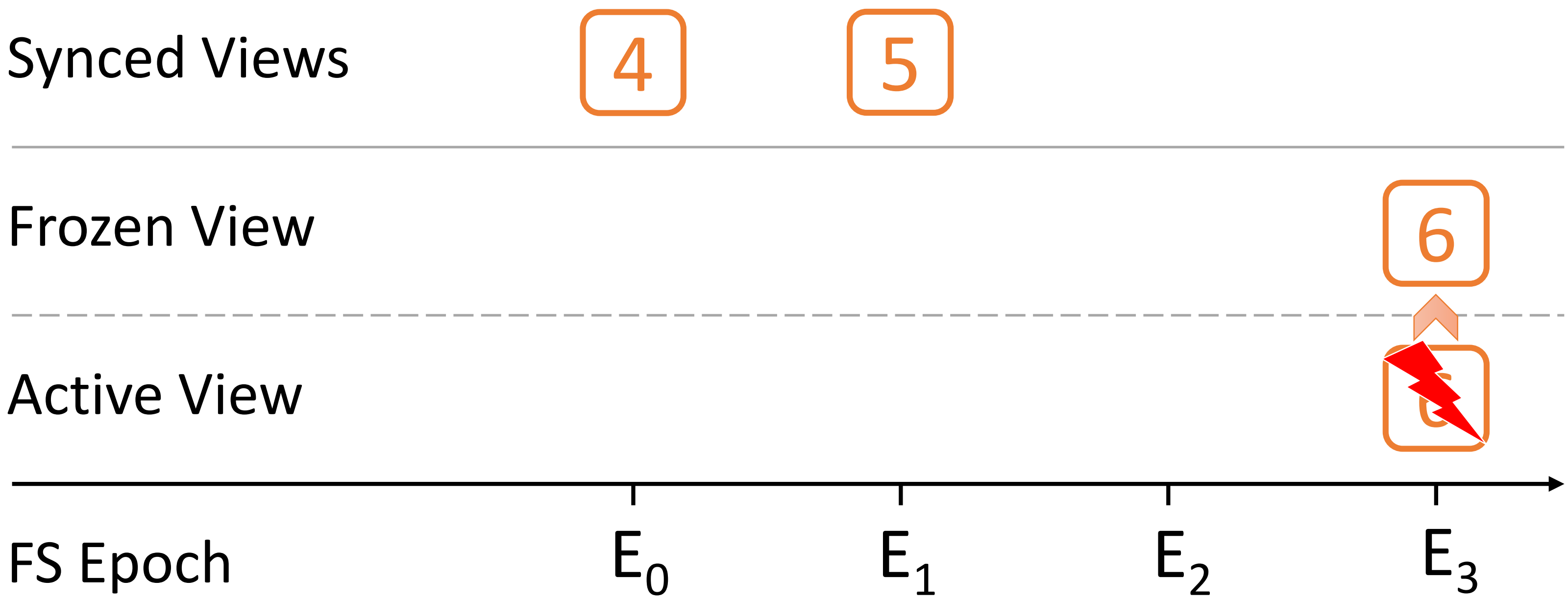
- Frozen view 5 has been uploaded completely
- Cannot freeze view 6 at this time

Cloud Journaling Example



- Create a new frozen view
  - after the previous frozen view is synchronized
  - and when FS reaches an epoch

Cloud Journaling Example



- Upon crash
  - Roll back to 5 from cloud

Server-side Changes

- Single-client scenario
  - Always one-direction synchronization (client to cloud)
  - No server-side changes are necessary
  - ViewBox + Dropbox (unmodified)

- Multi-client scenario
  - Server cannot propagate a partially-uploaded view
  - Client must handle conflicts carefully
  - ViewBox + Seafile (open-source, modified)

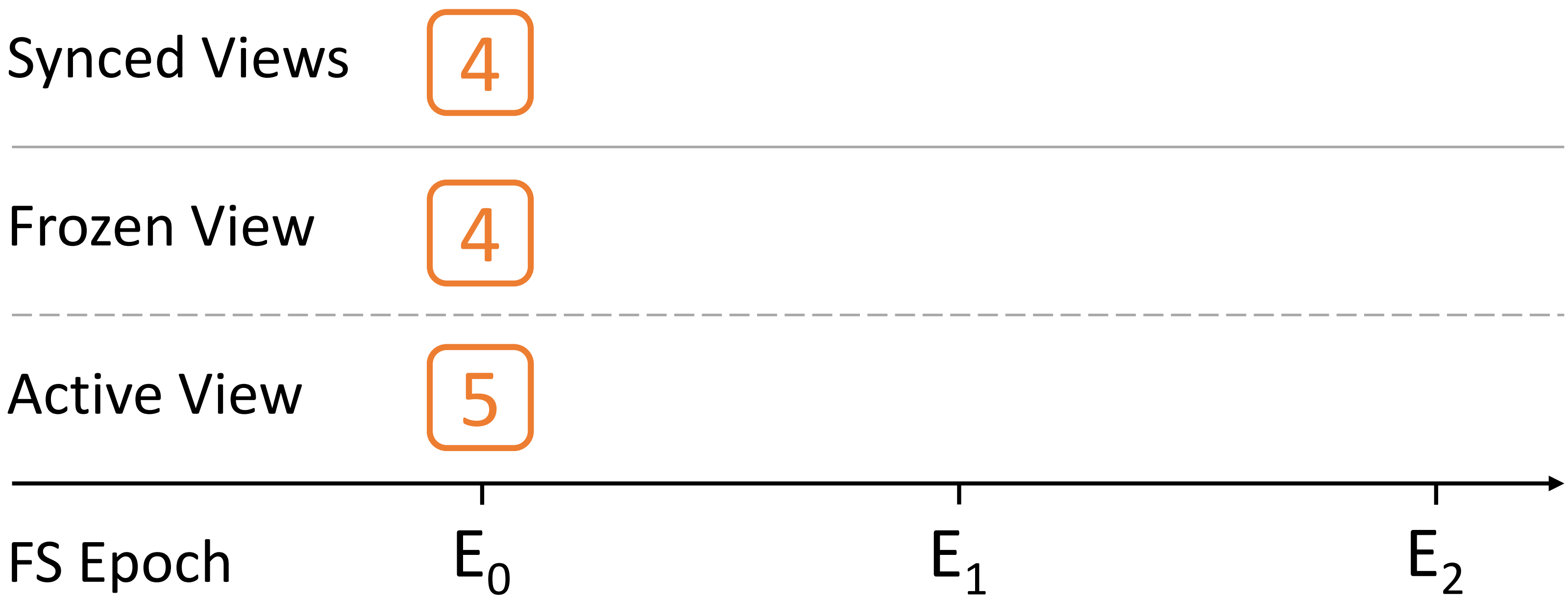
Challenge 2: How to Efficiently Freeze a View?

- A frozen view is short-lived and kept only in memory

- Requirements
  - No changes to FS's on-disk structures
  - No delay to on-going FS operations
  - Minimal memory overhead

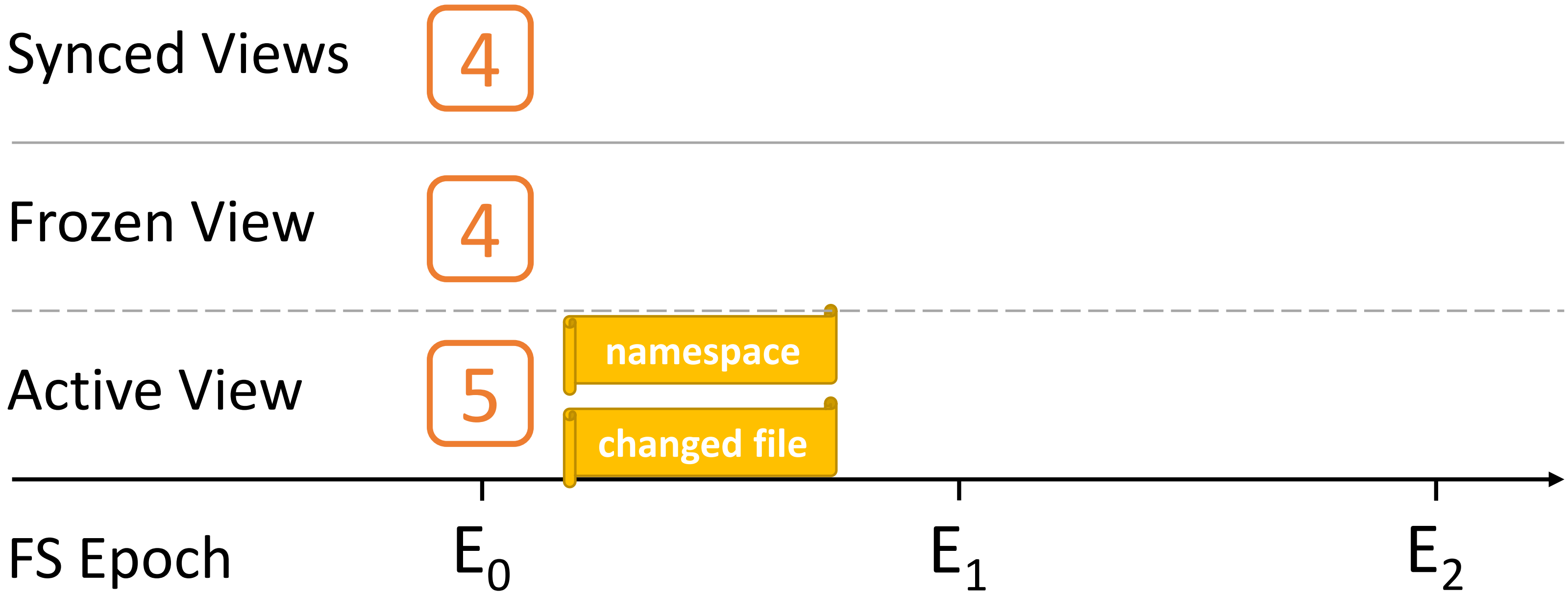
- Incremental snapshotting
  - Decouple namespace and data

Incremental Snapshotting Example



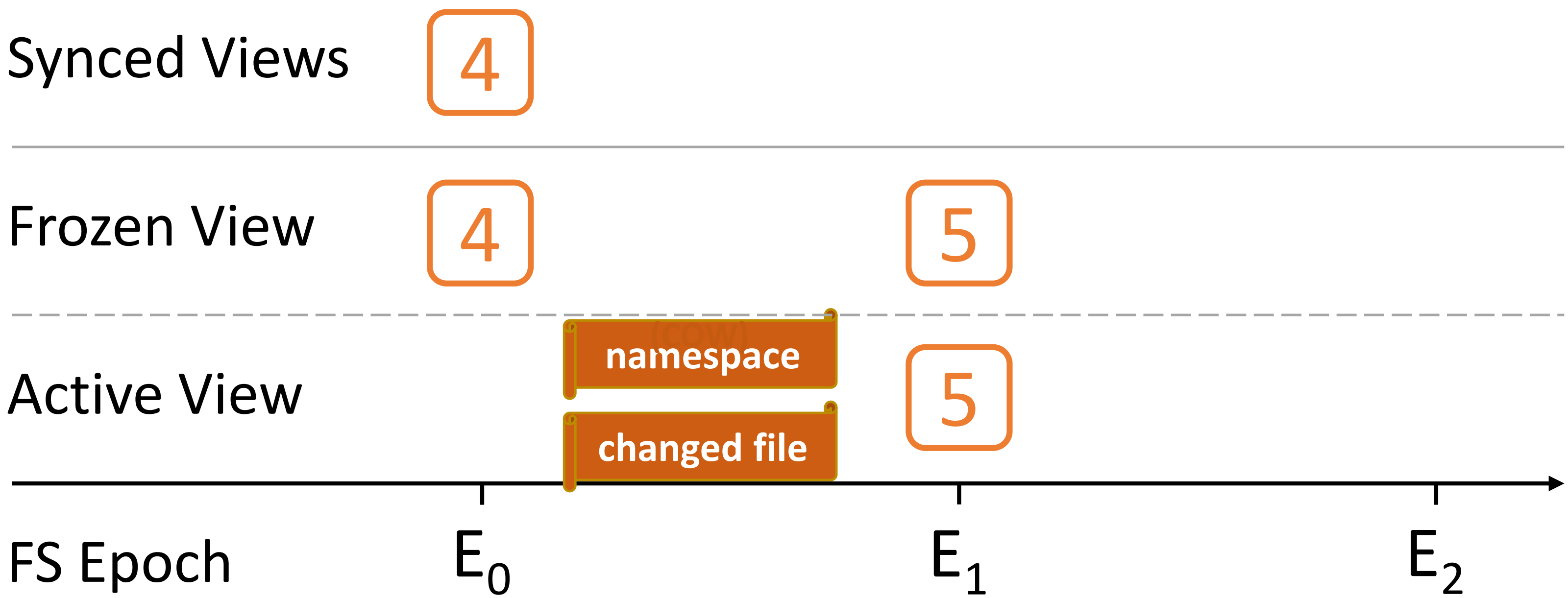
- Maintain last frozen view 4 in memory
  - Only namespace is preserved

Incremental Snapshotting Example



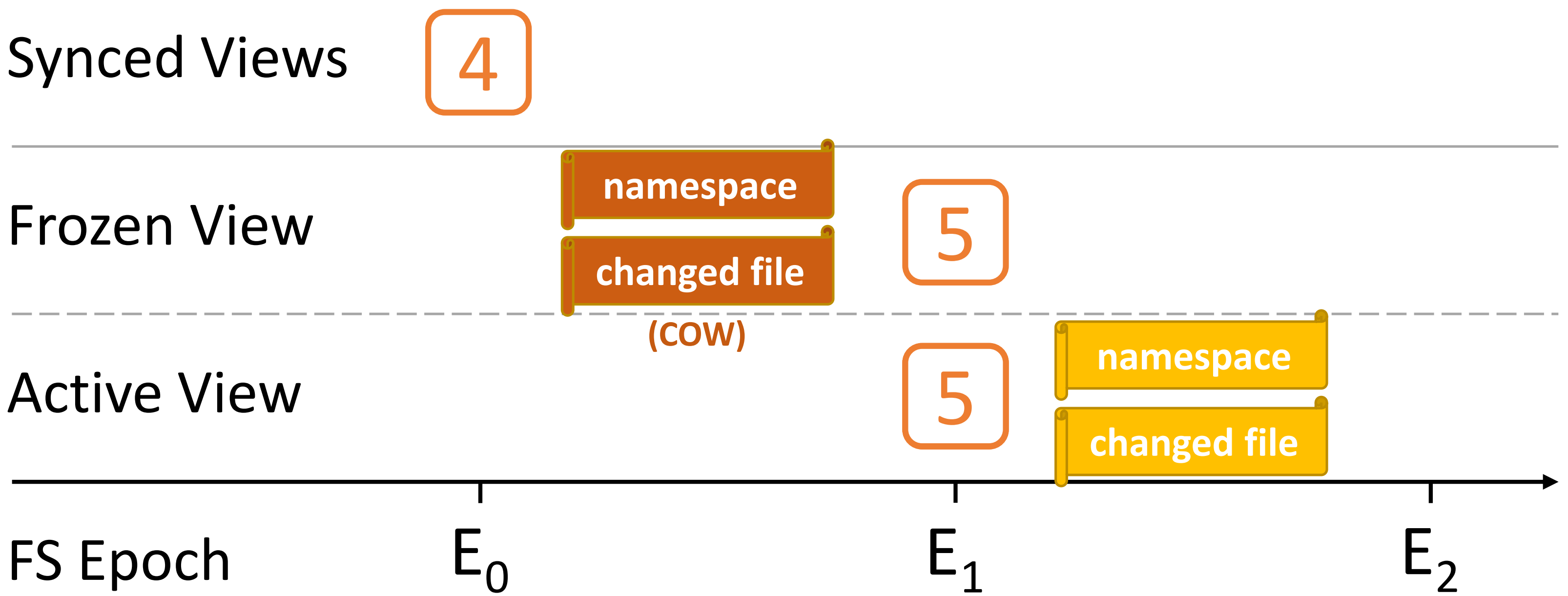
- Track updates in local FS through active view 5
  - Log namespace changes and data changes in memory

Incremental Snapshotting Example



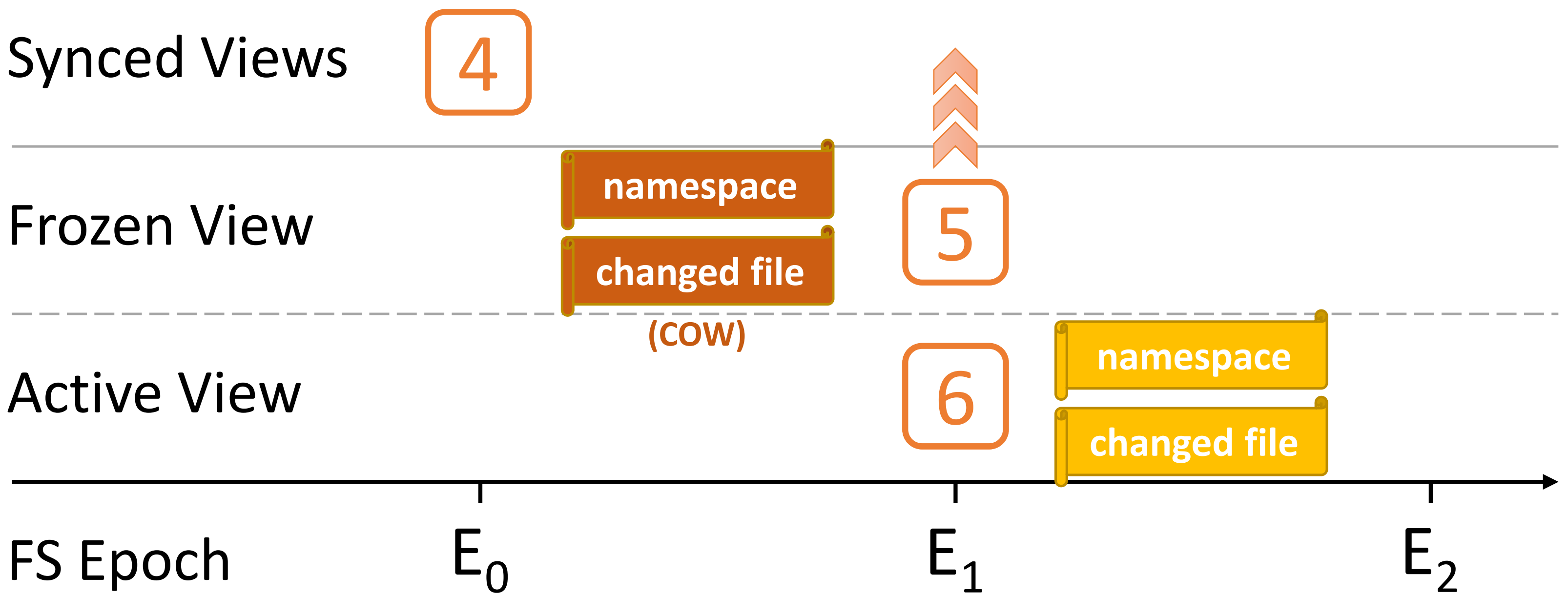
- Freeze current active view 5
  - Apply namespace changes to last frozen view 4
  - File data is still kept in local FS, but marked COW

Incremental Snapshotting Example



- At the same time, active view 6 starts immediately
  - On-going FS operations are not interrupted
  - COWed data is copied over to frozen view 5 if necessary

Incremental Snapshotting Example



- Upload frozen view 5
  - Re-generate inotify events
  - Trick sync client to upload changes from frozen view 5

Outline

- Introduction
- Motivation
- Design and Implementation
- **Evaluation**
- Conclusion

Evaluation

- Questions to answer
  - Can ViewBox offer integrity, consistency, and recoverability?
  - What is the overhead of ViewBox during user workloads?

- Setup (for both server and client machines)
  - 3.3GHz Intel Quad Core CPU, 16 GB memory
  - 1TB Hitachi hard drive
  - Linux kernel 3.6.11 (64-bit), ~7000 LOC added/modified
  - Dropbox client 1.6.0
  - Seafile client and server 1.8.0

Reliability

- Data Corruption

L: Local corruption G: Global corruption
D: Detected R: Recovered

Service	Data Writes	Metadata Changes		
		mtime	ctime	atime
ViewBox w/Dropbox	D R	D R	D R	D R
ViewBox w/Seafile	D R	D R	D R	D R

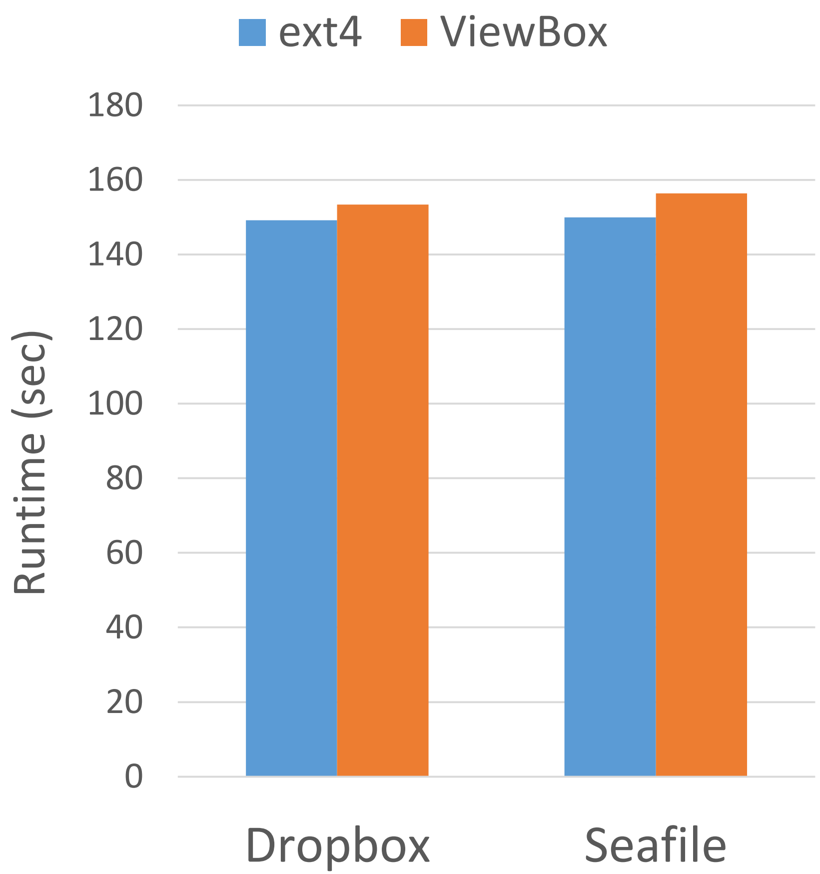
- Crash consistency

YES: occurred NO: did not occur

Service	Upload Local Ver.	Download Cloud Ver.	Out-of-sync (no sync)
ViewBox w/Dropbox	NO	YES	NO
ViewBox w/Seafile	NO	YES	NO

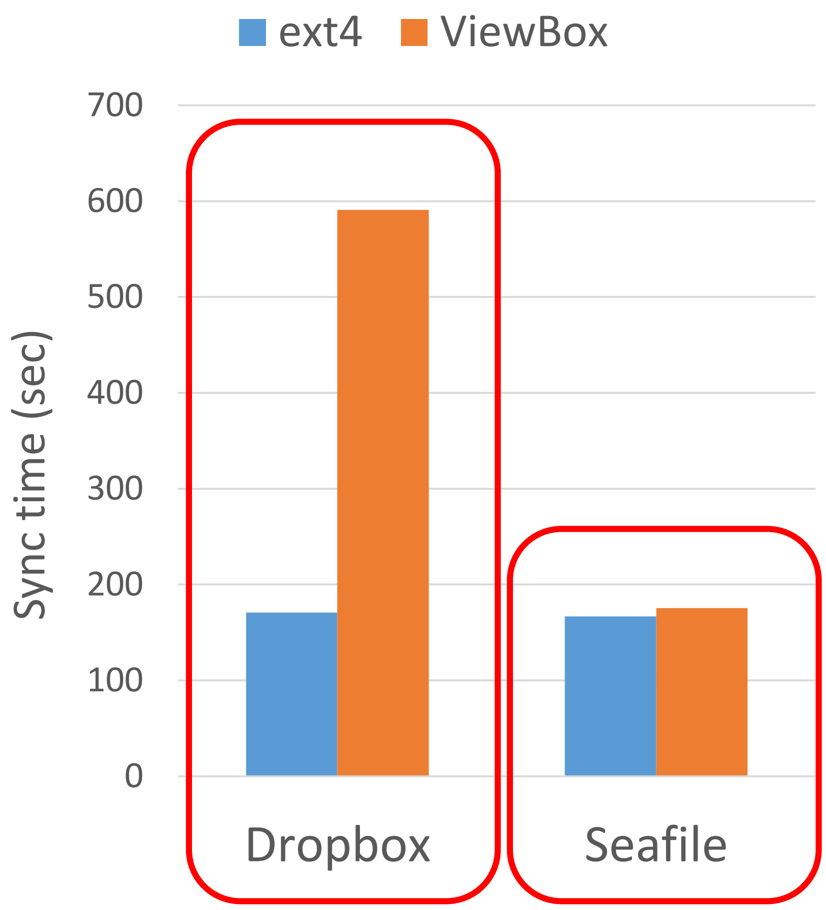
- Causal ordering is preserved

Performance - Photo Viewing



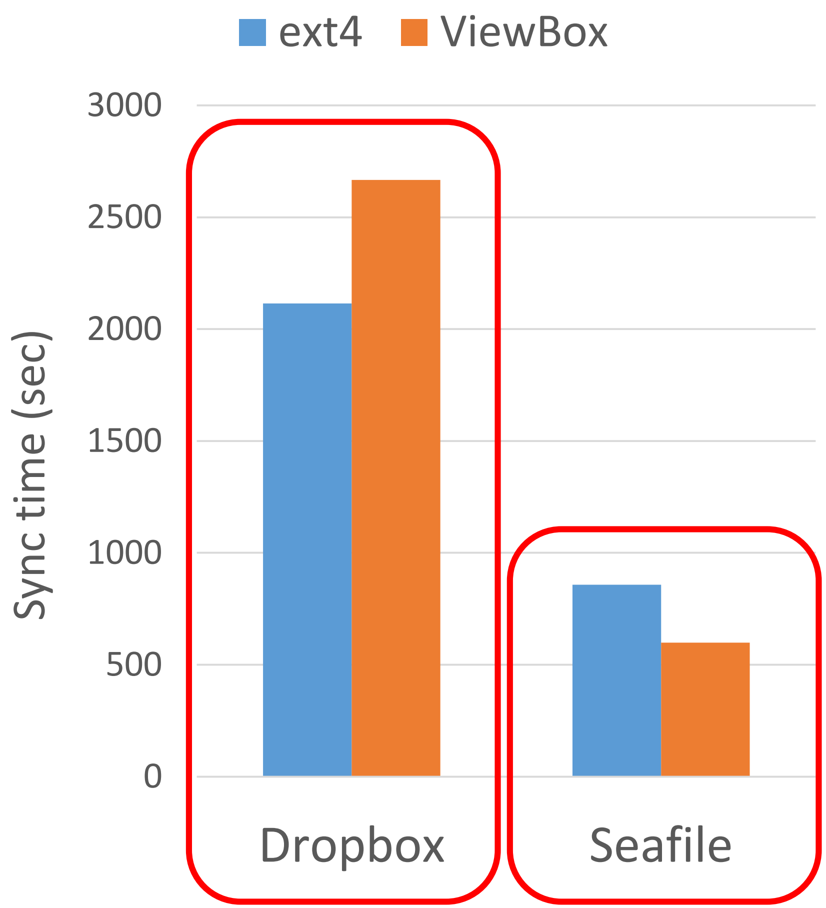
- iphoto_view from iBench [Harter2011]
  - Sequentially view 400 photos
  - Read-dominant
- Runtime
  - Time taken to finish the workload
  - ViewBox has <5% overhead
- Memory overhead
  - < 20MB

Performance - Photo Viewing



- Sync time
  - Time taken to finish synchronizing
- Huge increase in sync time with ViewBox + Dropbox
- View metadata for Dropbox
  - A list of {pathname, version number}
  - Remote walk ~1200 dirs (~1200 RTT) due to lack of proper server support
- View metadata for Seafile
  - Its internal commit ID

Performance - Photo Editing



- iphoto_edit from iBench [Harter2011]
  - Sequentially edit 400 photos
  - Reads:Writes = 7:3
- 30% reduction in sync time with ViewBox + Seafile
- Reduced interference from foreground update
  - Original Seafile may delay uploading
  - ViewBox keeps uploading changes from frozen views

Conclusion

- Problem: Cloud storage services and file systems fail to protect data

  cloud state ≠ file system state ≠ correct state

  - Many copies do NOT always make data safe

- Solution: ViewBox

  cloud state = file system state = correct state

  - Enhance local file systems with data checksumming
  - Present file system's view to sync service

- Tighter integration => more than reliability?

ViewBox: Integrating Local File Systems with Cloud Storage Services

Thanks! Questions?

Advanced Systems Lab (ADSL)

University of Wisconsin-Madison

http://www.cs.wisc.edu/adsl

Wisconsin Institute on Software-defined Datacenters in Madison

http://wisdom.cs.wisc.edu/