Team 2

Version File System

Contents

- Introduction
- Metadata Storage
- Graphical Interface
  - Timeline Utility
  - History Analyzer
  - Disk Monitor
- Versioning Operations
  - Commit
  - Checkout/Branch
  - Revert
  - Delete
- Cleanup Version History
- Benchmarking

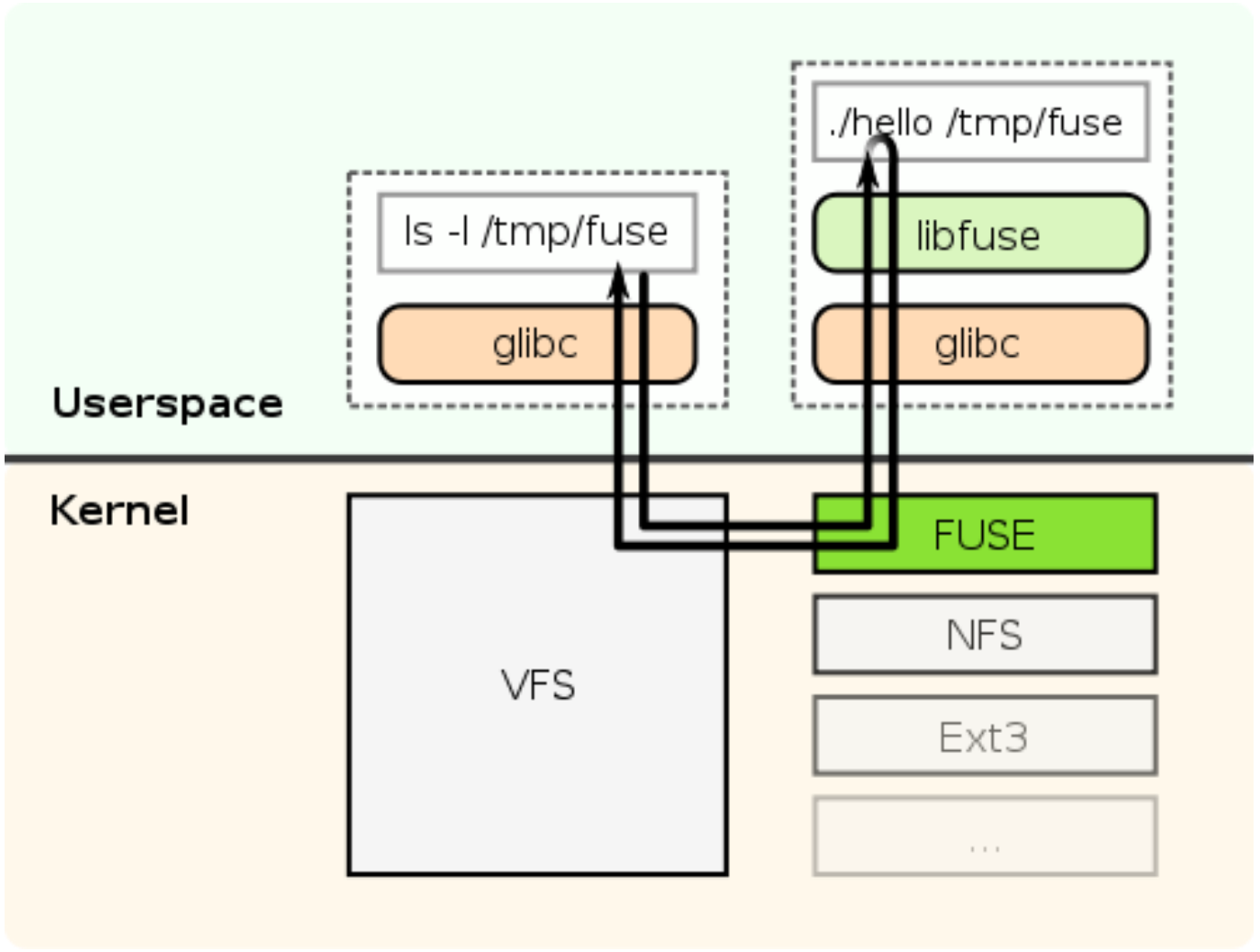
Introduction

- A **versioning file system** is any computer file system which allows a computer file to exist in several versions at the same time
- Kernel level interception of file operations delivers ubiquity
- Intuitive timeline GUI hooked to file explorer which makes it usable by laymen unfamiliar with command line
- Branching to give a complete versioning system functionality
- Better **multi user support** due to branching

External Integration

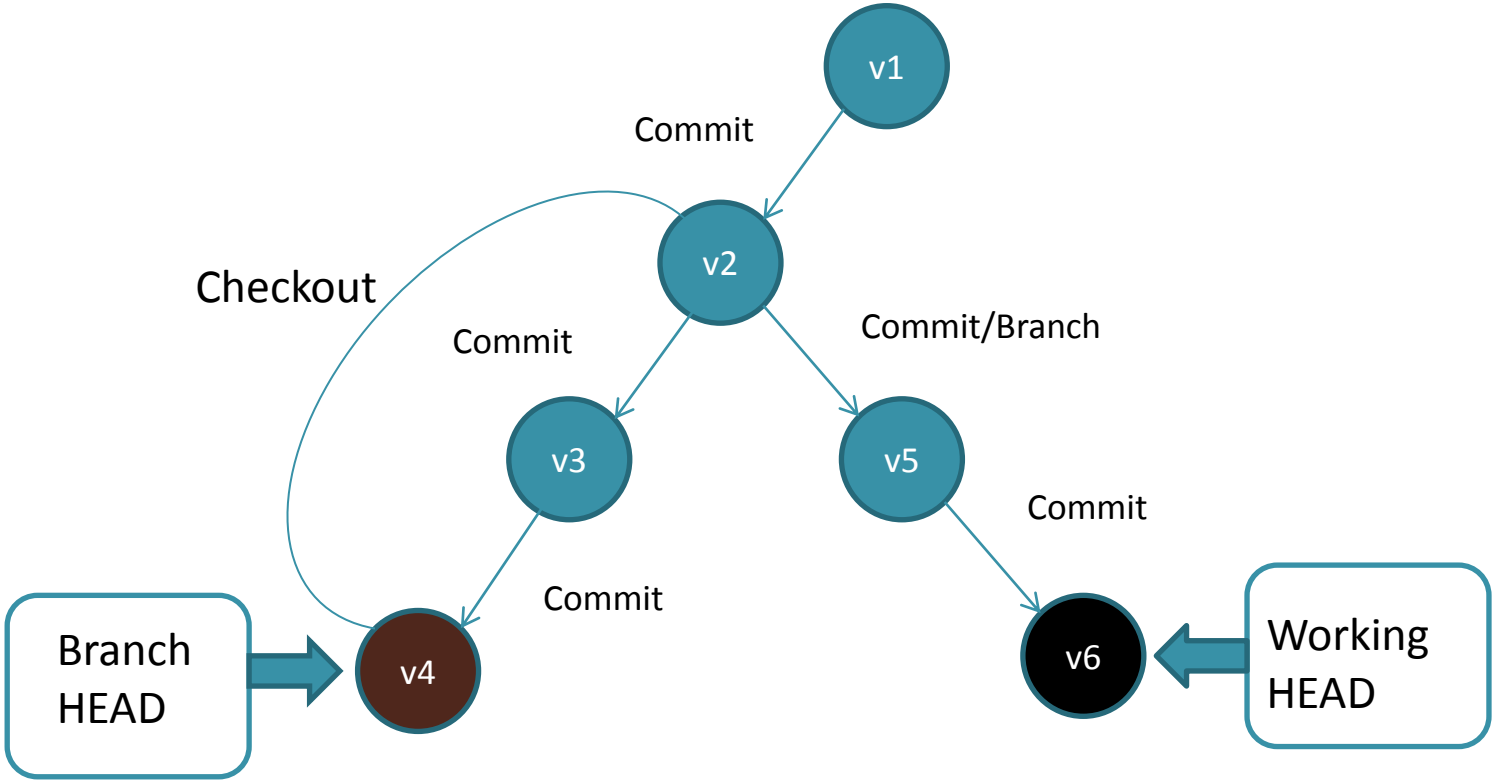
- File System in User Space – FUSE is used to intercept System Kernel calls
- Due to integration at Virtual File System, all higher level applications like Bash, File Explorers and Application Programs work inherently
- Versioning Metadata is “perfectly” invisible anywhere over Virtual File System
- Versioning operations like Snapshot reduce to copy command
- FUSE integrates with ext2, JFS, NTFS and even ISO

FUSE Architecture



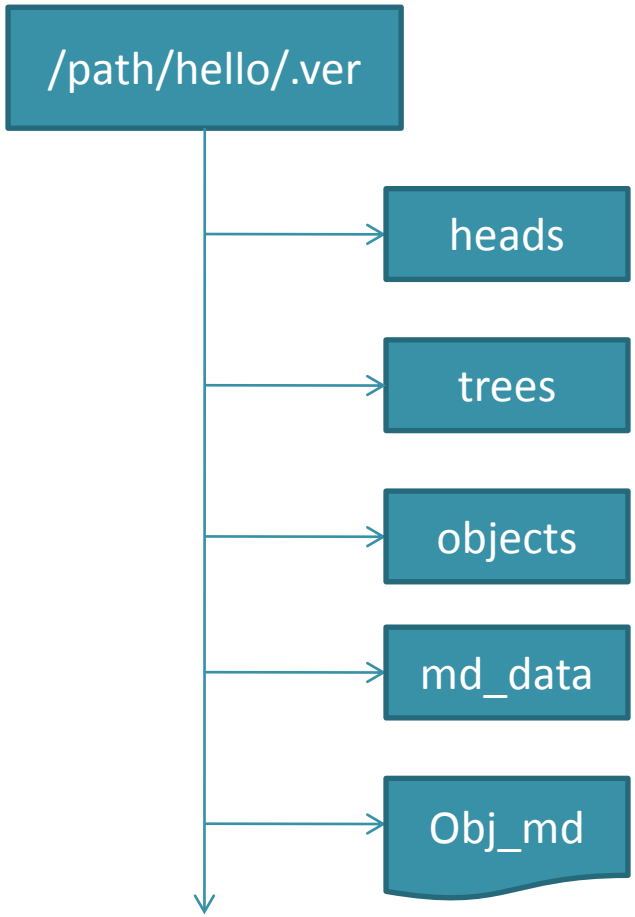
Representing History

- Versioning history is best represented as a tree. Let's see some of the ops.



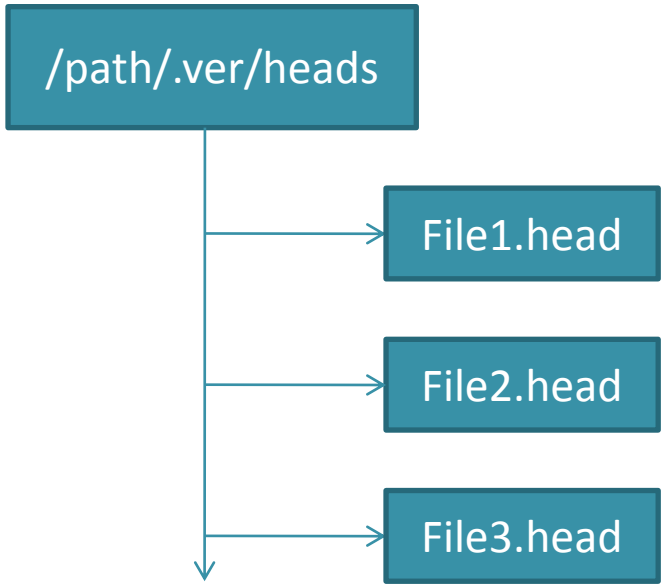
Metadata Storage

- Metadata is stored in each folder, marked as .ver
- .ver has 4 folders and a file
  - Heads – stores the information about Branch Heads
  - Trees – Information about File History
  - Objects – File diffs kept in compressed format
  - Md_data -- Overhead Profile
  - Obj_md – Metadata about objects folder

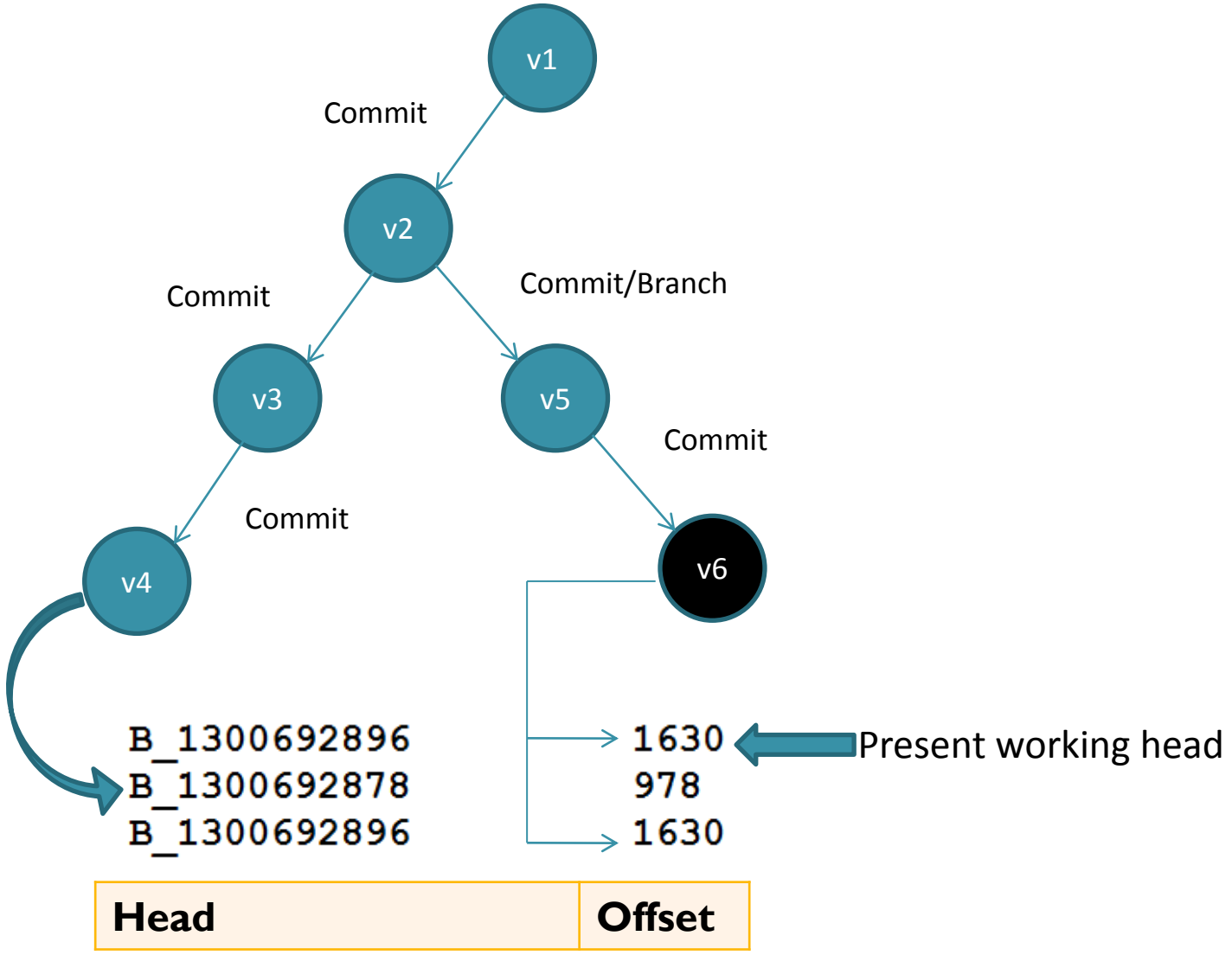


Metadata Storage: heads

- Heads folder contains per file information about the working heads.
- First line is the present working HEAD
- Rest line contain list of all possible HEADS

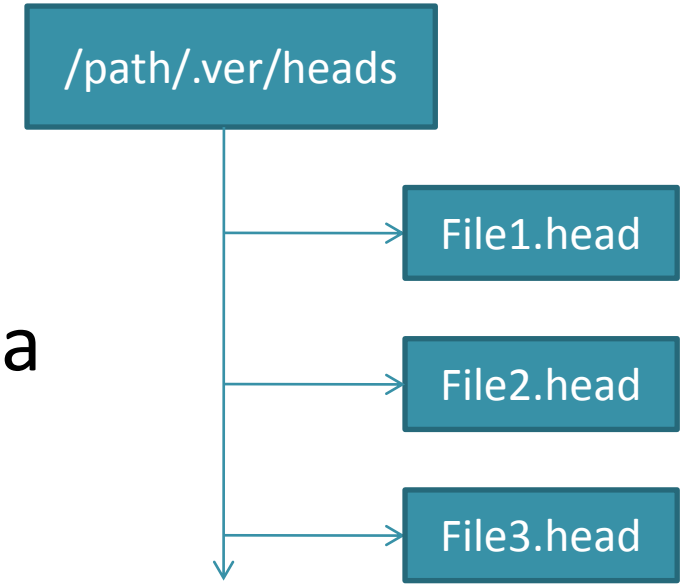


Metadata Storage: Heads

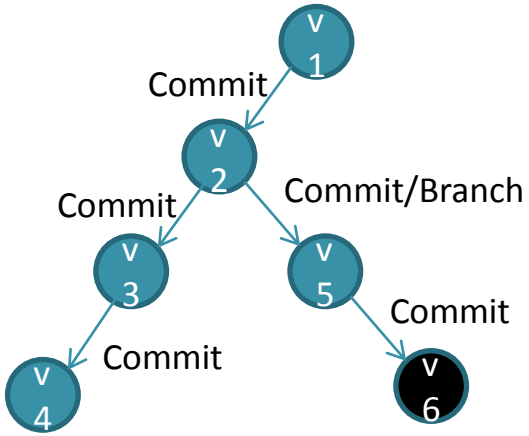


Metadata Storage: Trees

- Trees folder contains per file information of complete history info.
- Let's consider the following version tree of a file ….



Metadata Storage: Trees



/path/.ver/trees/file1.tree

V	Epoch	T	OBJ Hash	Tag	Diff	Parent
V1	1	1300692854	1	9af2f8218b150c351ad802c6f3d66abe	NULL	306	-1
V2	1	1300692860	0	8b4a4a203842447d6dd90e434b93e190	tag1	308	0
V3	1	1300692873	1	ce551a577e5076ecbe3d0df8510ebfd9	tag2	309	326
V4	1	1300692878	0	16d916f35360c2fa454c8bc12a05048e	NULL	0	652
V5	1	1300692893	1	050dcb71cf2ec481e4055679bcc01ea5	NULL	309	326
V6	1	1300692896	0	135c84c9b80e85777a0f294e9ff39916	NULL	0	1304

Metadata Storage: Objects/Obj_md

- Actual diff/files are kept in .ver/objects in compressed format.
- All junctions in the tree are complete files
- Other nodes are kept as Diffs.
- Let’s take a look at the sample version tree.

Metadata Storage: Objects/Obj_md

/path/.ver/trees/obj_md

Object Hash	Ref Count
9af2f8218b150c351ad802c6f3d66abe	2
8b4a4a203842447d6dd90e434b93e190	3
050dcb71cf2ec481e4055679bcc01ea5	2
ce551a577e5076ecbe3d0df8510ebfd9	1
16d916f35360c2fa454c8bc12a05048e	1
135c84c9b80e85777a0f294e9ff39916	1

Metadata: Overhead Profile

- .ver contains a folder md_data which contains metadata profile(excess storage used)
- For each version:
  - Timestamp
  - Filesize
  - Metadata Size
  - Ratio: Filesize/(Filesize+Metadata Size)
- This is used to render a metadata profile of a file in History Analyzer User Interface

Metadata: Overhead Profile

Epoch	File size	Metadata Size	Performance Ratio
1300692854	13	540	0.023508
1300692860	26	1174	0.021667
1300692878	78	2443	0.030940
1300692893	39	2820	0.013641
1300692896	52	3455	0.014827

Salient Features of Design

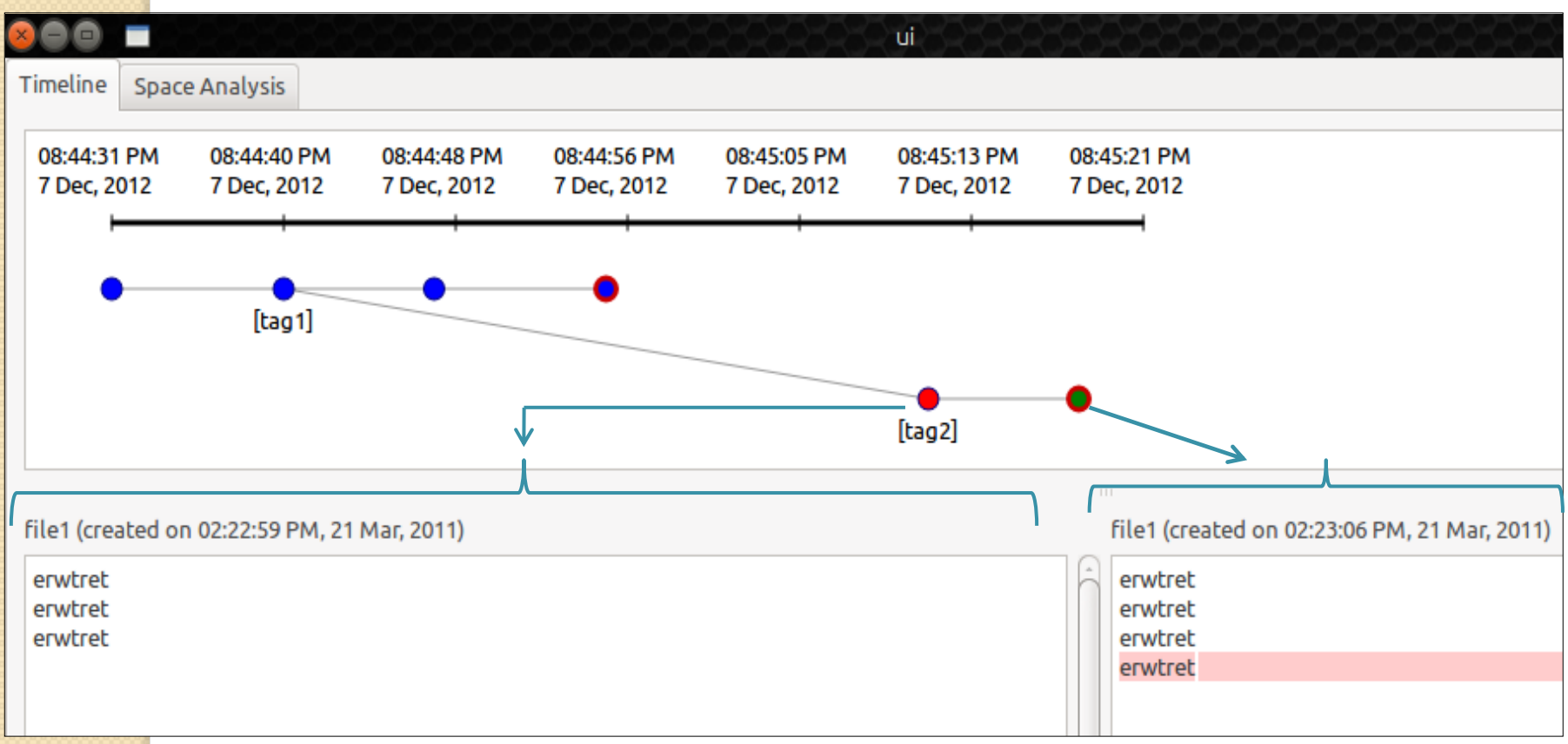
- MD5 Sum as object names
  - Avoids redundancy
- Tree can be traversed using offsets
  - No need to load the complete tree for traversal
- Distributed Metadata
  - Ensures smaller metadata for each file
  - Results in smaller lookup times and better time complexity

Graphical Utilities

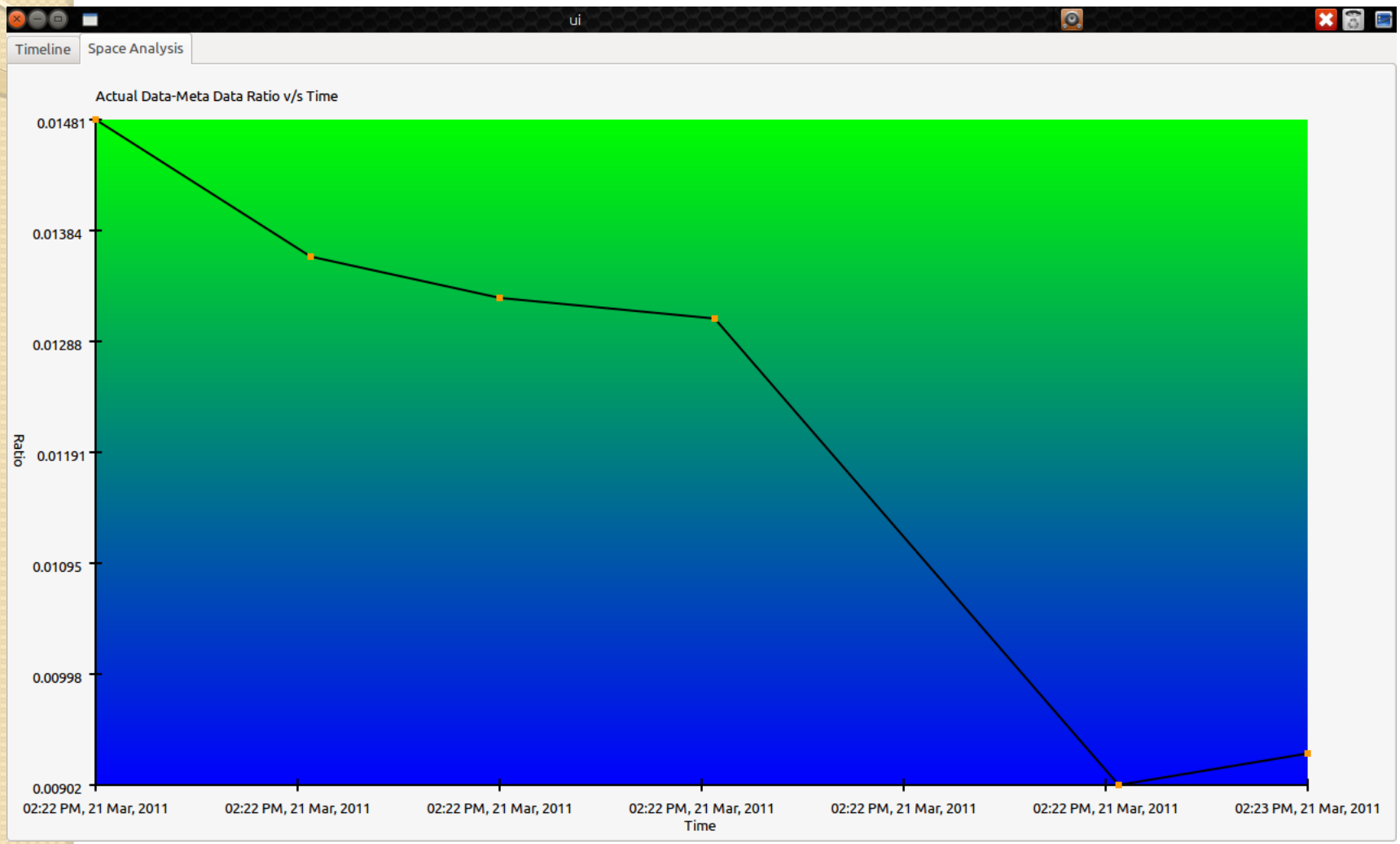
- A native Graphical Interface developed in QT and works across all platforms
- Users are more comfortable with a pictorial representation of versioning history
- TortoiseSVN and GitK are similar solutions
- Target users for our product are much larger than Code Versioning
- Strong need of simple and intuitive User Interface(away from Command Line)

Timeline Interface

- Images speak a 1000 words!



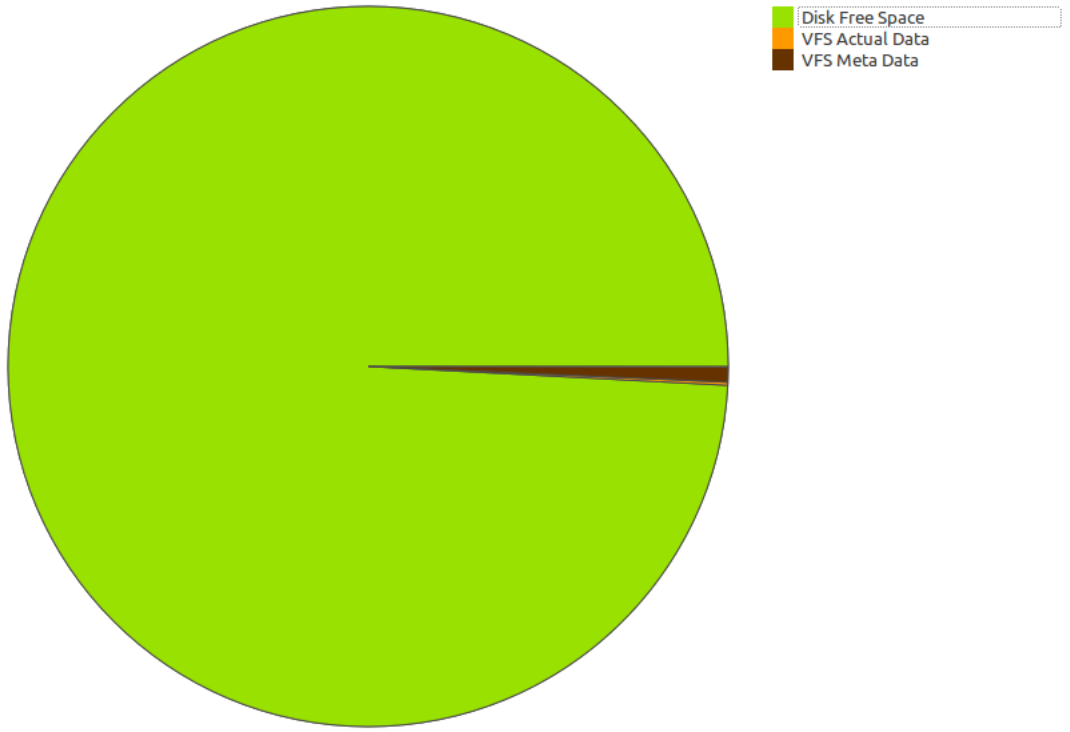
History Analyzer



Disk Monitor

File

	Label	Size in kB
1	Disk Free Space	1039928
2	VFS Actual Data	1532
3	VFS Meta Data	7116



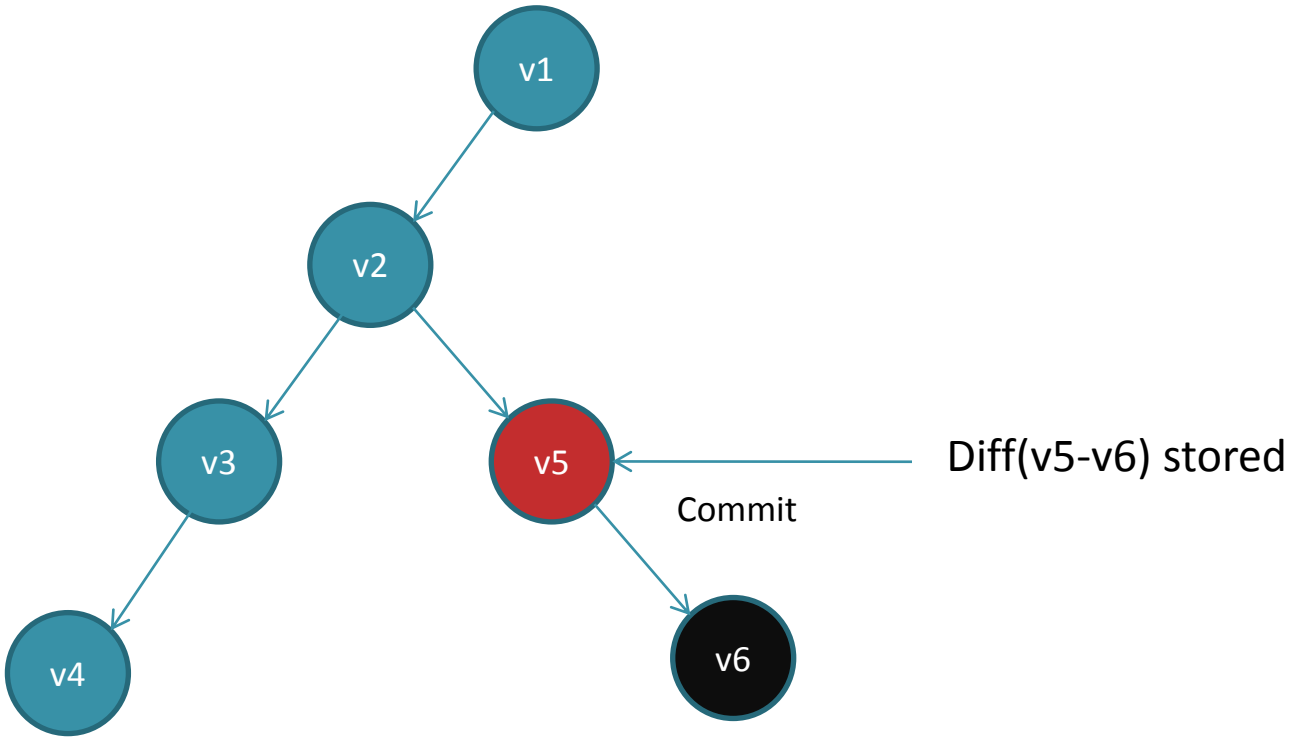
Versioning Operations

- Commit – make a new version on top of present branch
- Checkout – a previously stored file version
- Revert – to a previous saved version and delete not-required history

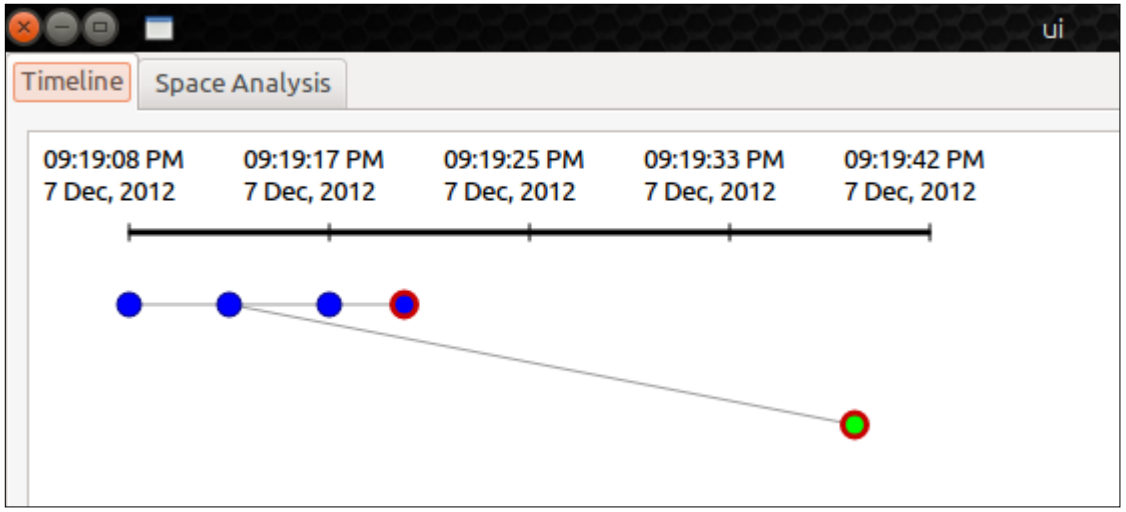
Commit

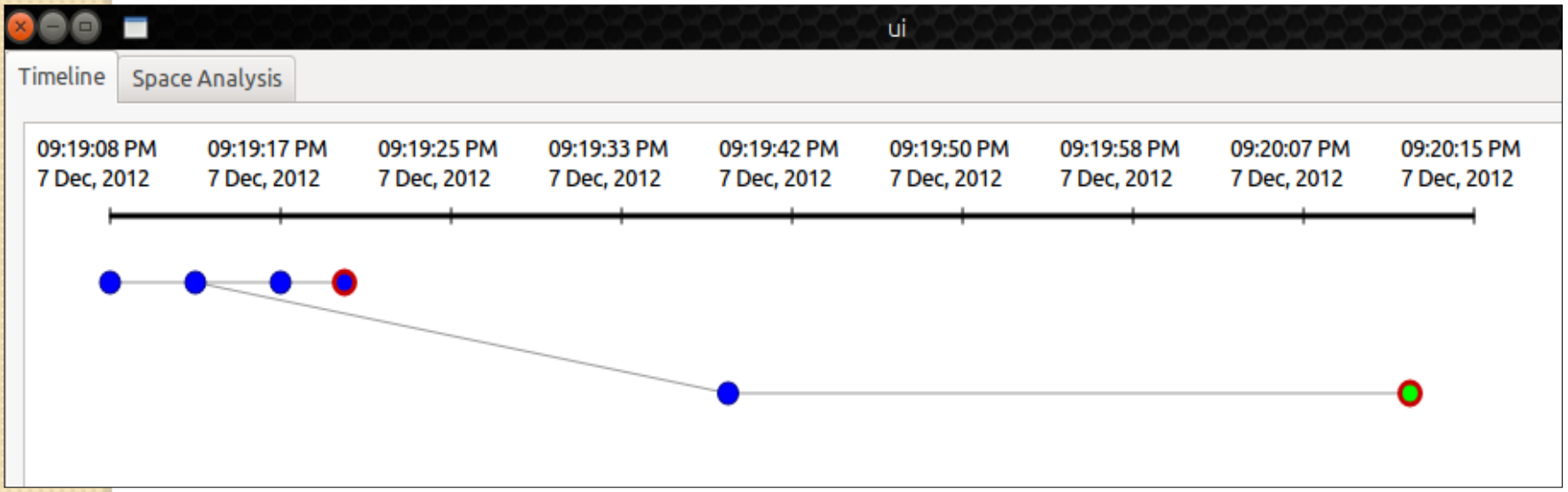
- While working on a current HEAD, on a write, new version is is committed
- Commit happens automatically on each save after a minimum time limit
- Algorithm:
  - Read the parent as the present working HEAD from file.head
  - Write to file.tree a new version with parent extracted above
  - Replace the present working HEAD with the new version created
  - Save the diff object to objects folder

Commit: Version tree



Version History: UI





Commit: file.tree

```
1 1300699535 1 904ef23e0b8aeabb95913ddaf9639c9a 299 -1
1 1300699539 0 d7c35fcf18bbfbdb59c67370217d6a69 302  0
1 1300699543 1 2769b2c6881d04061f028a6468d58d83 302  326
1 1300699546 0 cb5691204a1c47f66c979a6617652278 0    652
1 1300699564 0 2769b2c6881d04061f028a6468d58d83 0    326
```

```
1 1300699535 1 904ef23e0b8aeabb95913ddaf9639c9a 299 -1
1 1300699539 0 d7c35fcf18bbfbdb59c67370217d6a69 302  0
1 1300699543 1 2769b2c6881d04061f028a6468d58d83 302  326
1 1300699546 0 cb5691204a1c47f66c979a6617652278 0    652
1 1300699564 1 2769b2c6881d04061f028a6468d58d83 305  326
1 1300699596 0 1d471ce57084582816939f01ddd33c34 0    1304
```

Commit: Heads

```
B_1300699564     1304
B_1300699546     978
B_1300699564     1304
```

```
B_1300699596 1630
B_1300699546 978
B_1300699596 1630
```

Commit: OBJ MD

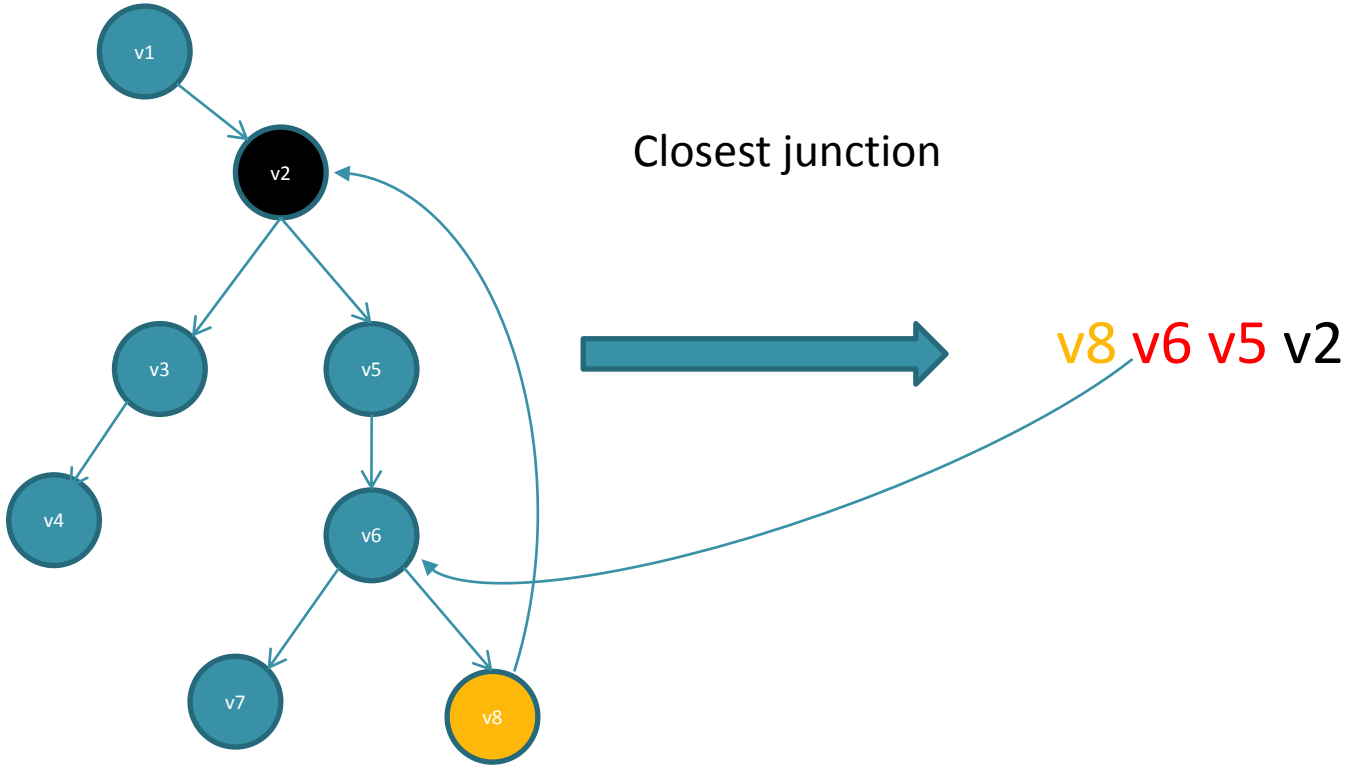
904ef23e0b8aeabb95913ddaf9639c9a 1
d7c35fcf18bbfbdb59c67370217d6a69 1
2769b2c6881d04061f028a6468d58d83 2
cb5691204a1c47f66c979a6617652278 1

904ef23e0b8aeabb95913ddaf9639c9a 1
d7c35fcf18bbfbdb59c67370217d6a69 1
2769b2c6881d04061f028a6468d58d83 2
cb5691204a1c47f66c979a6617652278 1
1d471ce5708458281693 9f01ddd33c34 1

Checkout

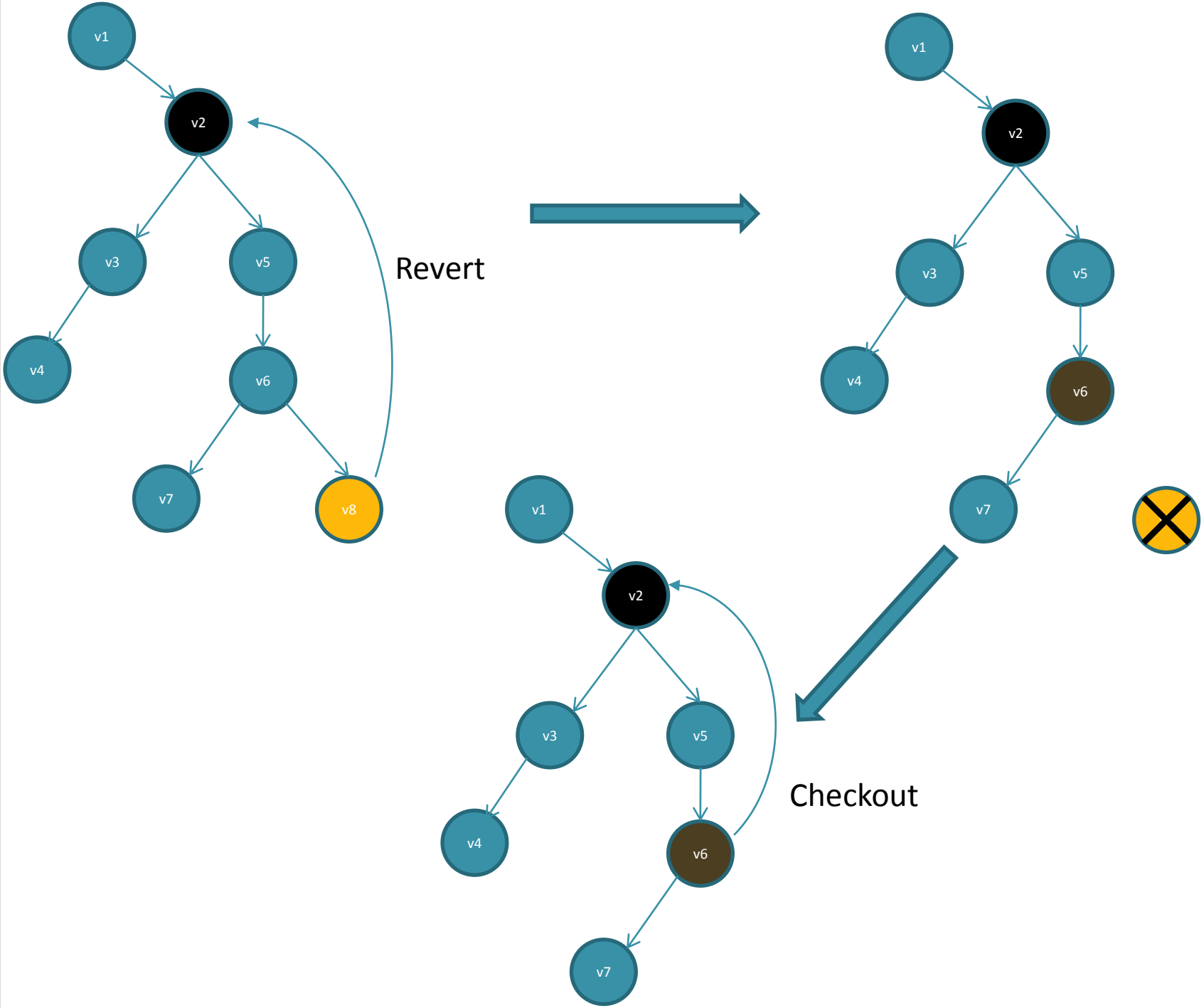
- With checkout you may see an old version
- Generally a checkout is followed by a branching
- As we save full file objects at all junctions, the algorithm for a checkout is as:
  - Lookup the HEAD offset in file.tree from file.head
  - Traverse from the present HEAD to checkout location
  - Check for closest junction
  - Rollback diffs to get the checkout version
- Bash command: lsver <version epoch>
- Interestingly for large branching situations, our algorithm will checkout very fast as it will roll back from nearest junction

Checkout: Version Tree



Revert

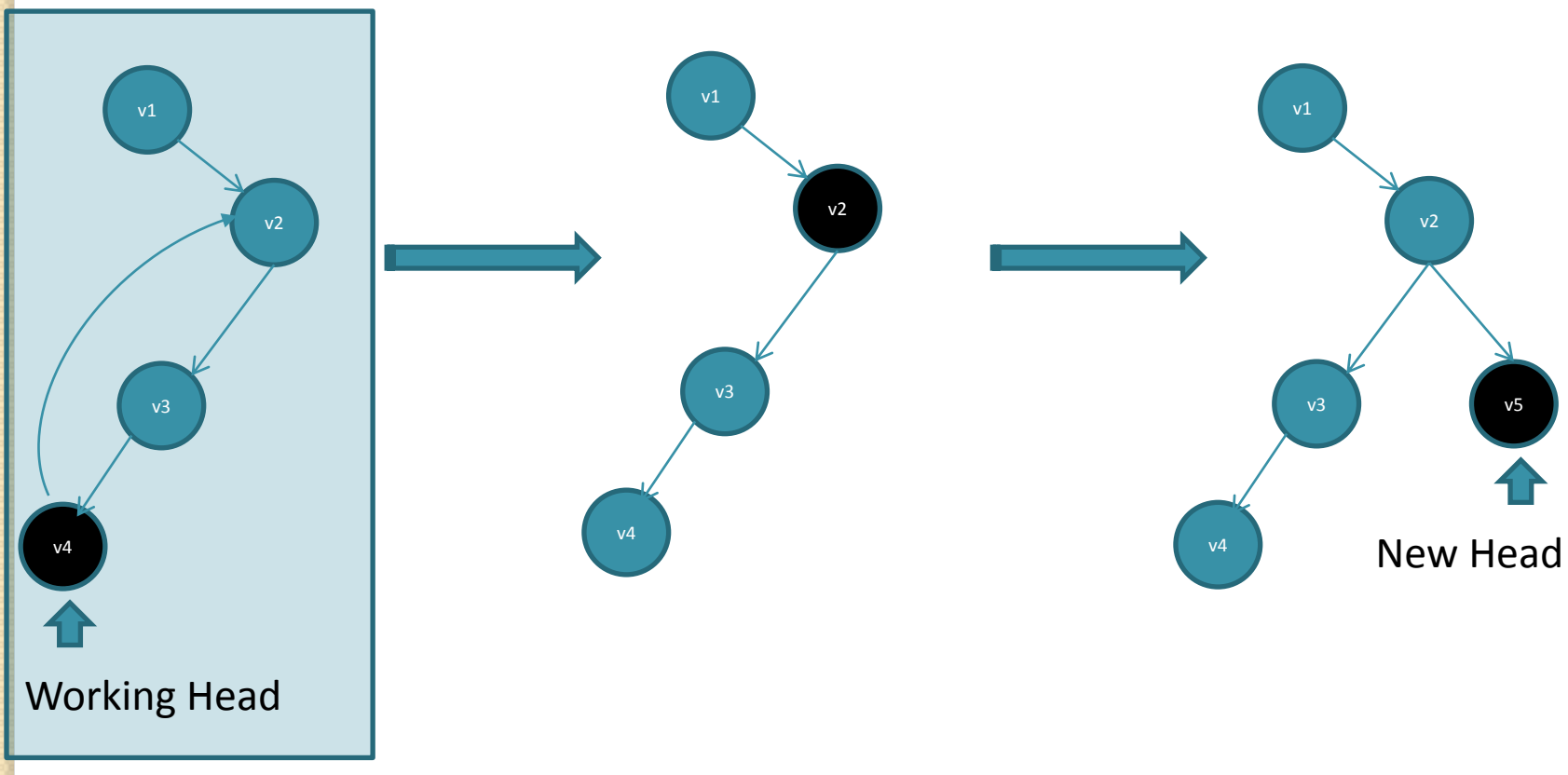
- With this operation one may delete history and roll back to a previous version in my working branch
- Algorithm
  - Extract working HEAD offset in tree
  - Traverse from the present HEAD to revert location
  - Delete all objects until the first junction
  - Do a checkout from the first junction to the revert location
- Bash Command: revert <version epoch>


v1
v2
v3
v5
v4
v6
v7
v8
Revert
v1
v2
v3
v5
v4
v6
v7
v1
v2
v3
v5
v4
v6
v7
Checkout

Branch

- It is an implicit combination of a checkout AND revert
- A typical workflow is:
  - Check out to an old version
  - Commit a new version
- Let’s see a representation

Branch: Checkout and Commit



Cleanup Version History

- A user may store various versions, without really using the history
- Essential to cleanup not-so-essential versioning history
- Measures possible:
  - Delete very old versions
  - Remove very small diffs
  - What about time
- Also how much to clean ?
- Measure used:
  - #diff / Δt, where Δt is version time difference

Cleanup Algorithm

- Version Importance: Vi = #diff / Δt
- Disk Efficiency(E): Filesize/(Filesize + Meta)
- Algorithm:
  - Input: file, threshold
  - do while E<threshold
  - Create for the file versions Vi and sort them
  - Merge/Delete version of least Vi and recalculate E
- Note:
  - We do not cleanup junctions and tagged versions because they inflict importance

Performance Analysis

- Performance depends on:
  - No. of versions
  - Diff b/w versions [diff size]
  - Degree of branching
- General user behaviors
  - Less and long branches
  - Small Diffs between consecutive writes
- We optimize and show our performance for the general user behavior
  - Small files (0-250 KB) , 5 % branching, 10 % diff, 50 nodes
  - Large files (1-5 MB) , 5 % branching, 10 % diff, 25 nodes

Use Scenarios

	Small Files[50]	**Large Files[25]**
Optimal Metadata Size	296 KB	3.4 MB
Final Version Size	456KB	4.944 MB
Metadata	1300KB	18.704MB
Checkout Junction	0.550 sec	0.315 sec
Checkout Non Junction	0.678 sec	1.468 sec
Metadata After Cleanup	673KB	10.21MB