# Backlog: Tracking Back  References in a Write-Anywhere File System

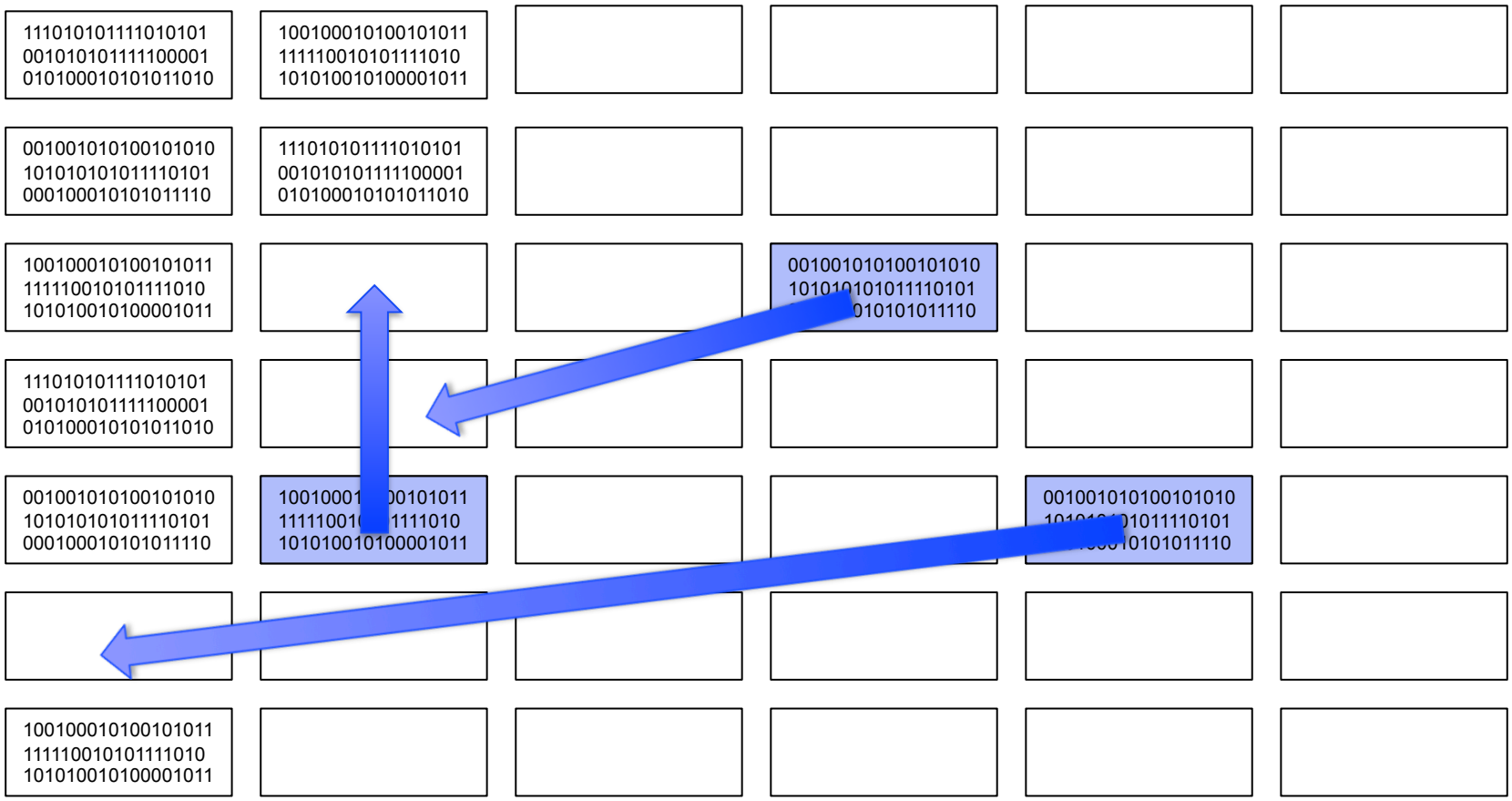

Peter Macko
Harvard University
Cambridge, MA

Margo Seltzer
Harvard University
Cambridge, MA

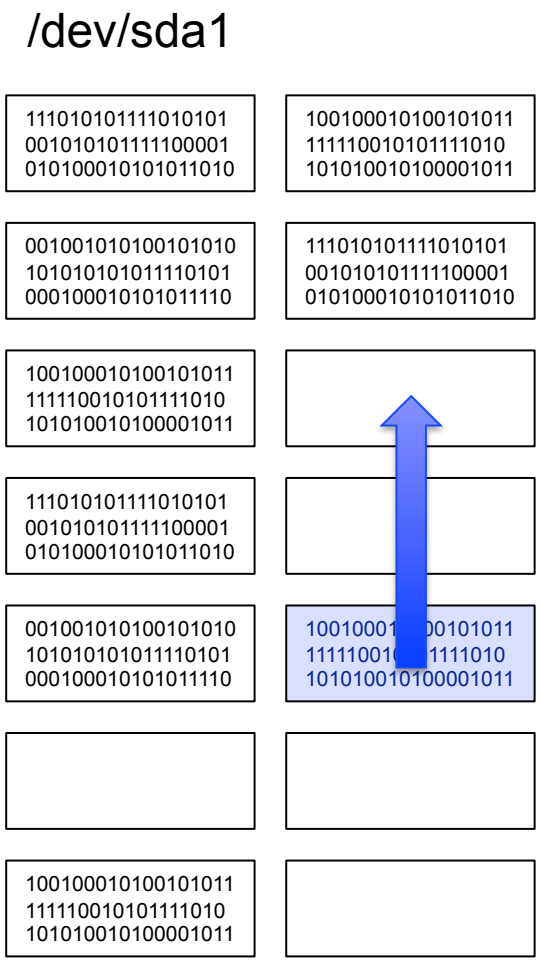
Keith Smith
NetApp, Inc.
Waltham, MA

# Challenge: Dynamically Reorganizing Data

/dev/sda1



# Challenge: Dynamically Reorganizing Data



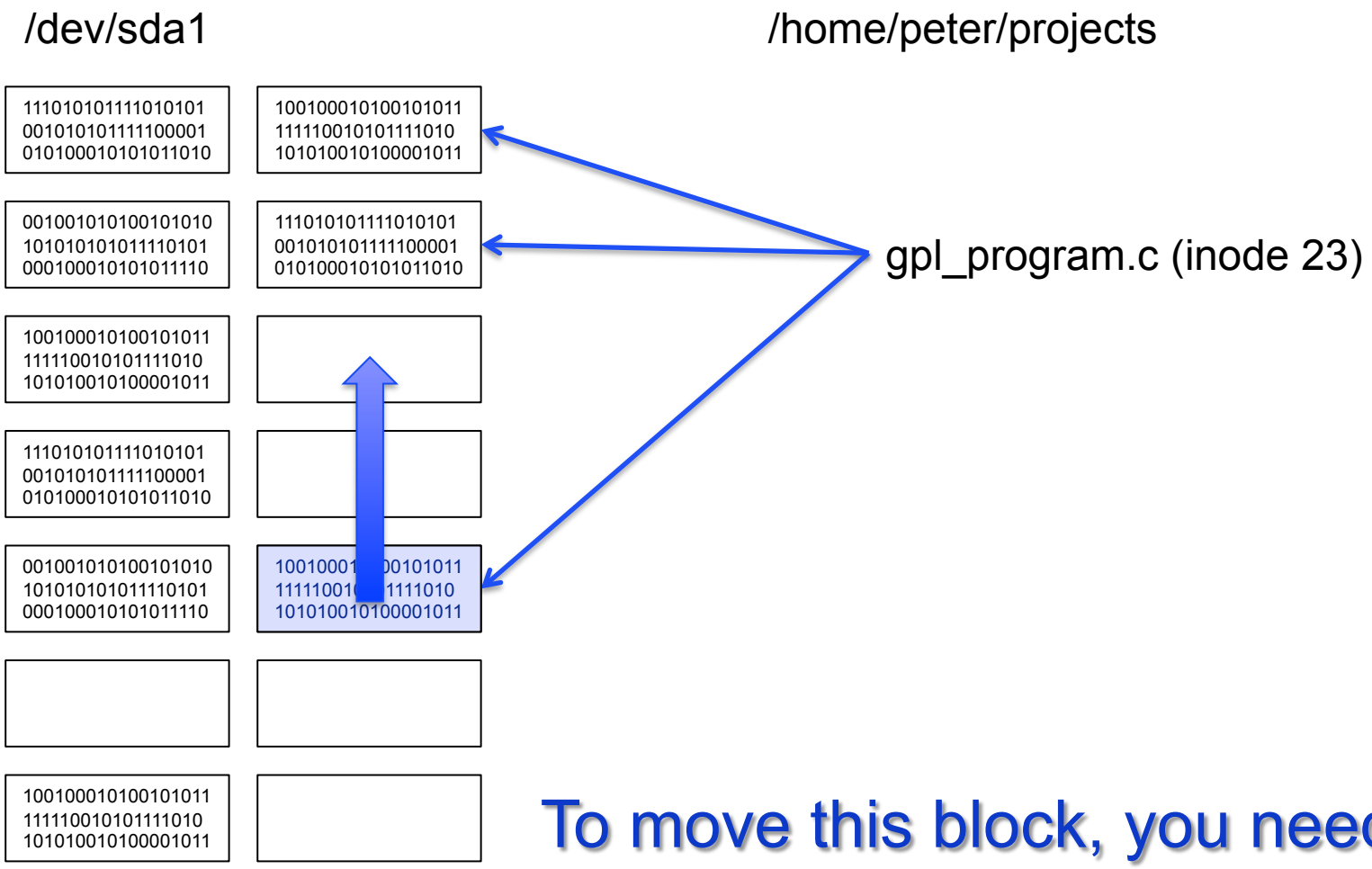

Reasons to move blocks:

- Consolidate free space
- Data migration
- Resize a partition
- Defragmentation

…

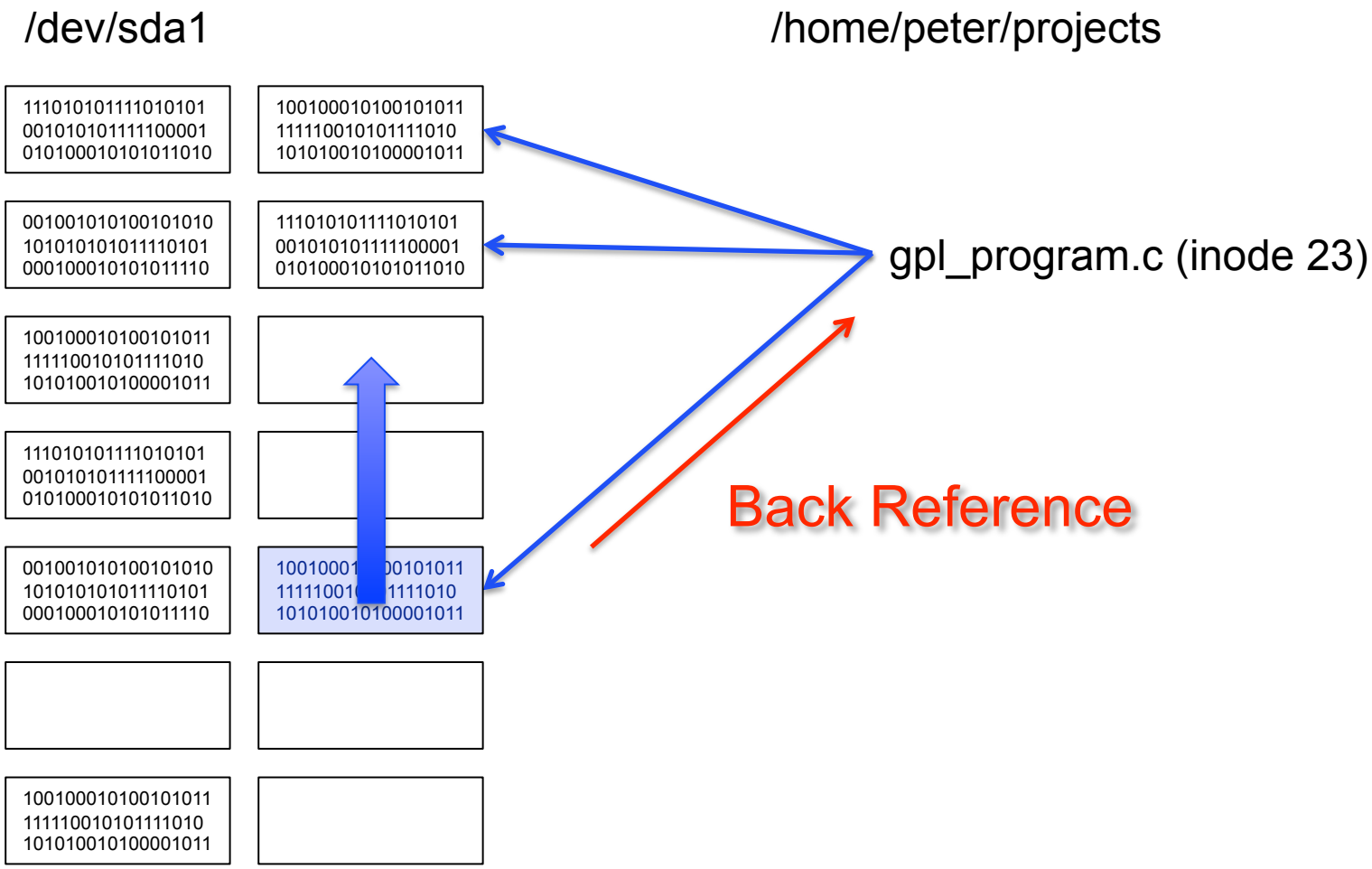
In many cases: You start with block numbers, not with the files

# Challenge: Dynamically Reorganizing Data




To move this block, you need to know all inodes that reference it

# Solution: Back References



# Outline

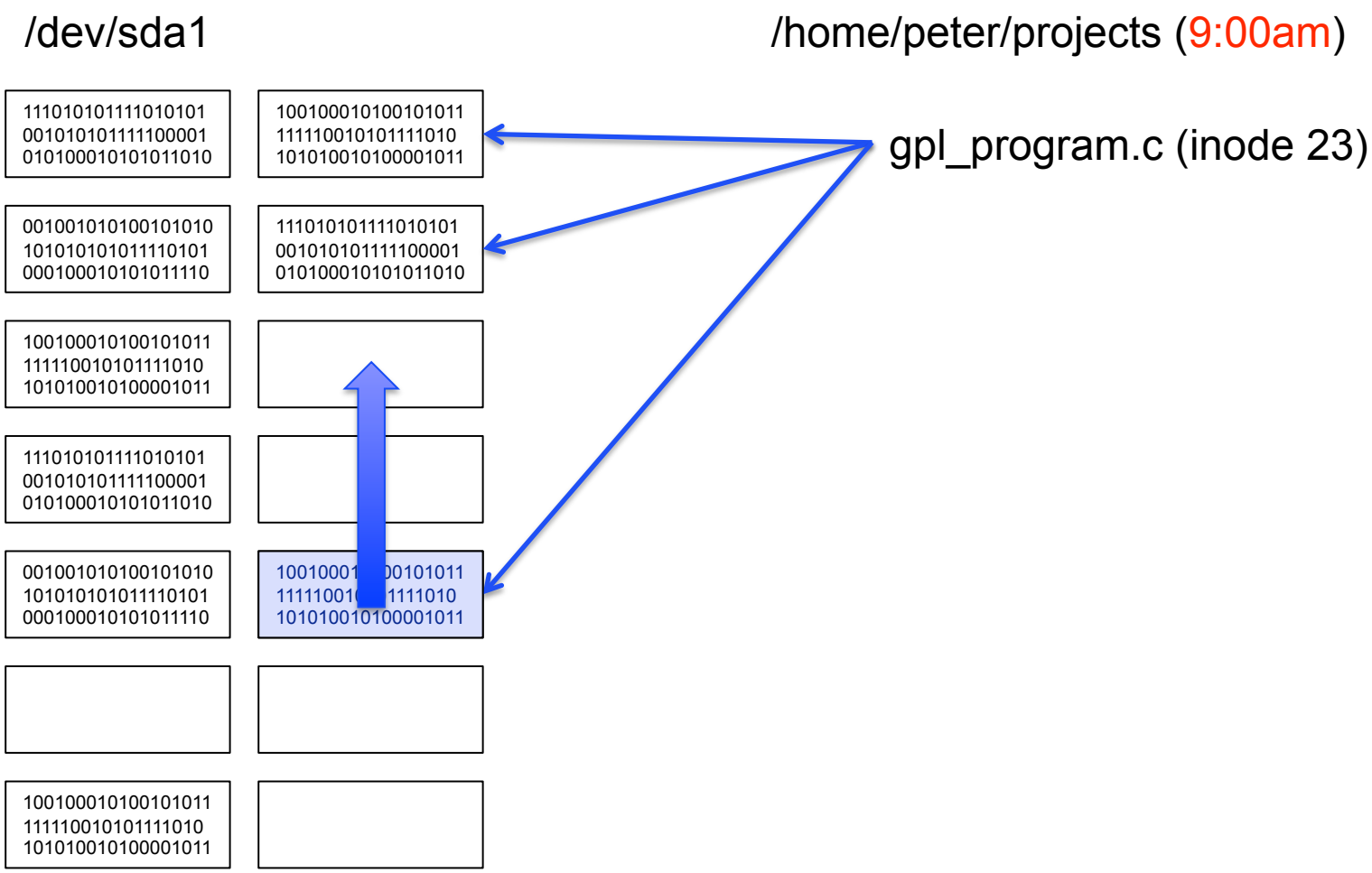
1. Introduction & Motivation
2. Challenges
3. Backlog Back References
4. Representing Writable Clones
5. Evaluation
6. Conclusion

# Challenges

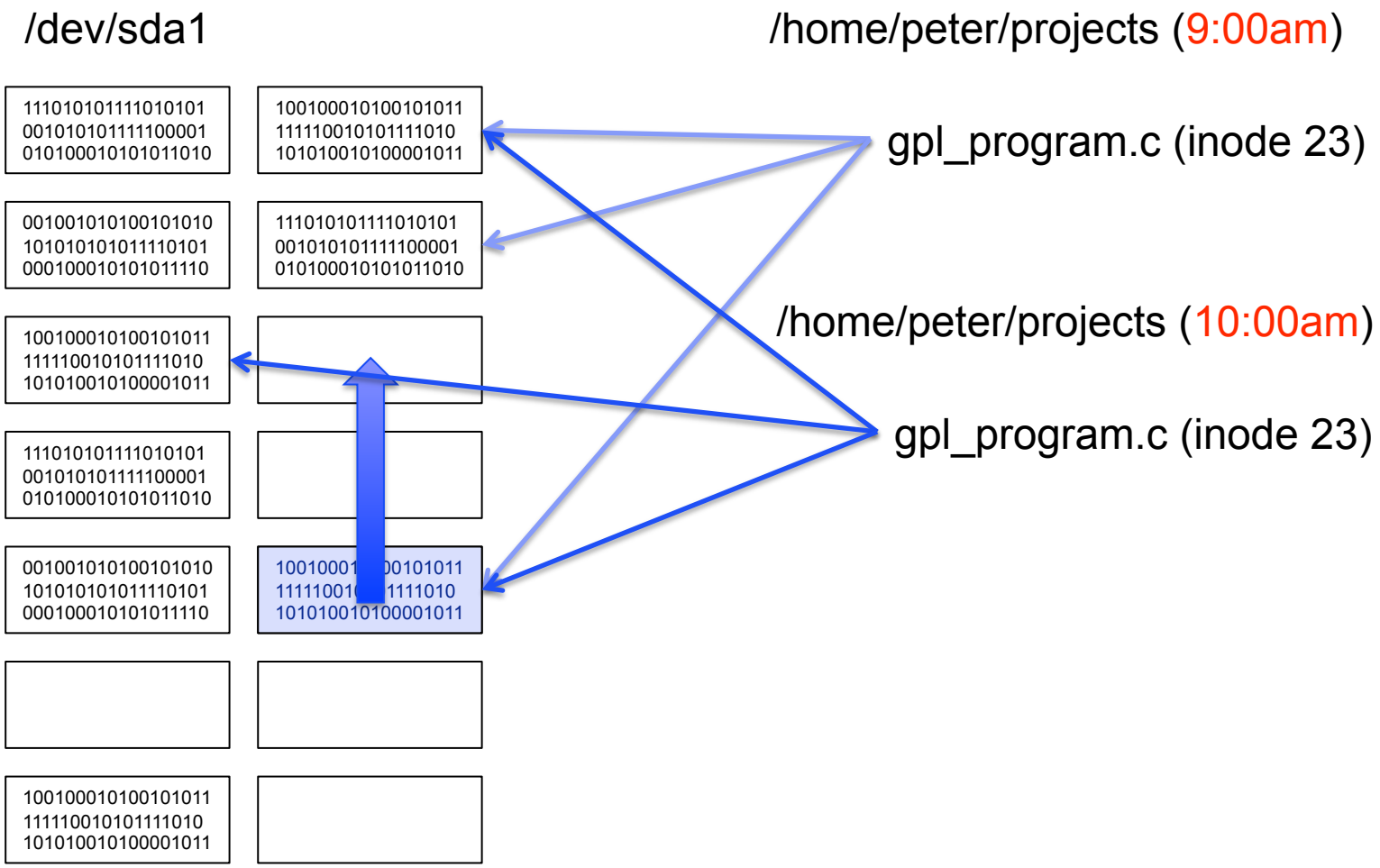
- Requirements
  - Low overhead
  - Performance does not decrease over time
  - Stable space overhead (not necessarily minimal)

- In the presence of:
  - Snapshots
  - Writable clones
  - Deduplication (block sharing)

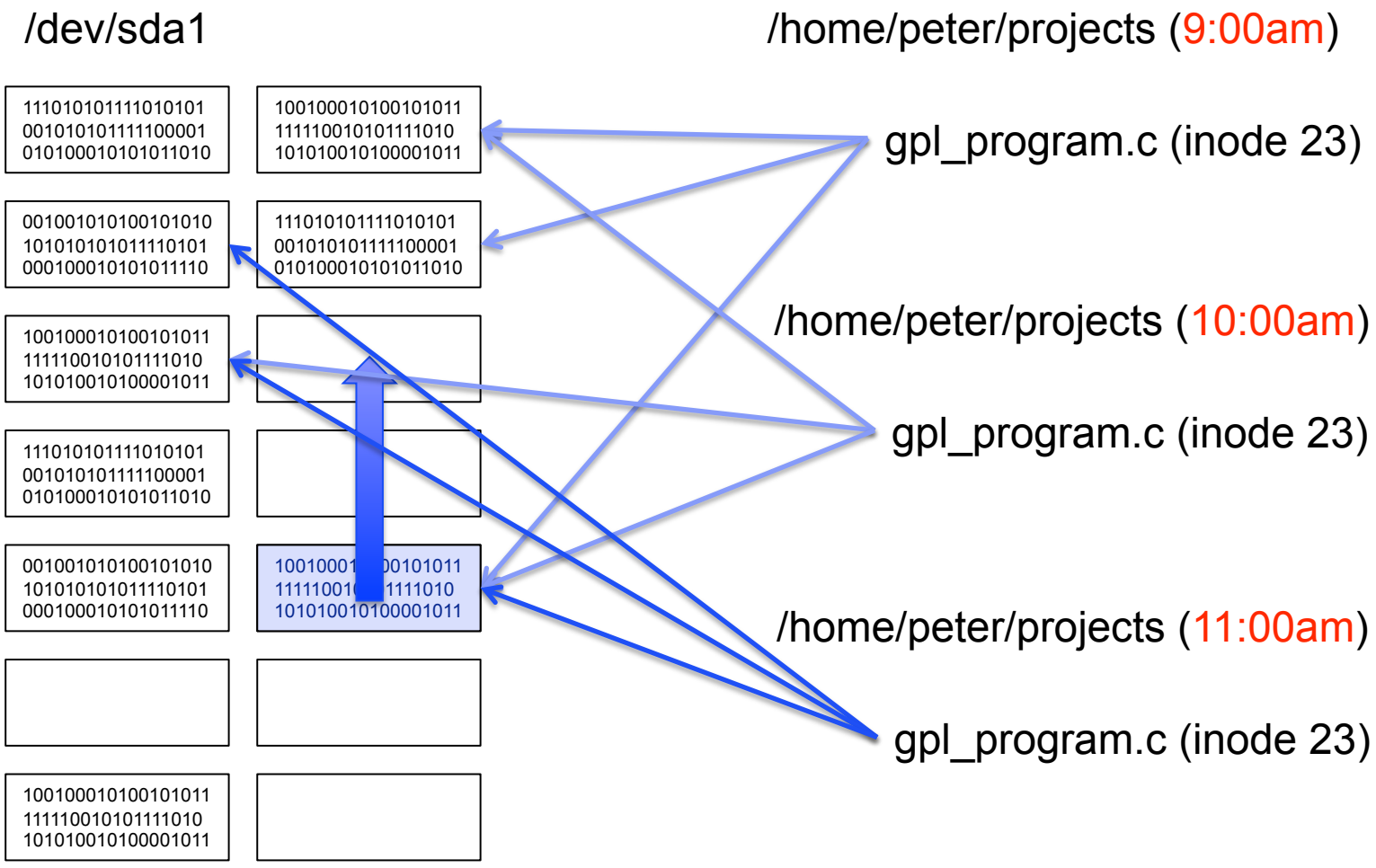
# Challenge: Read-Only Snapshots



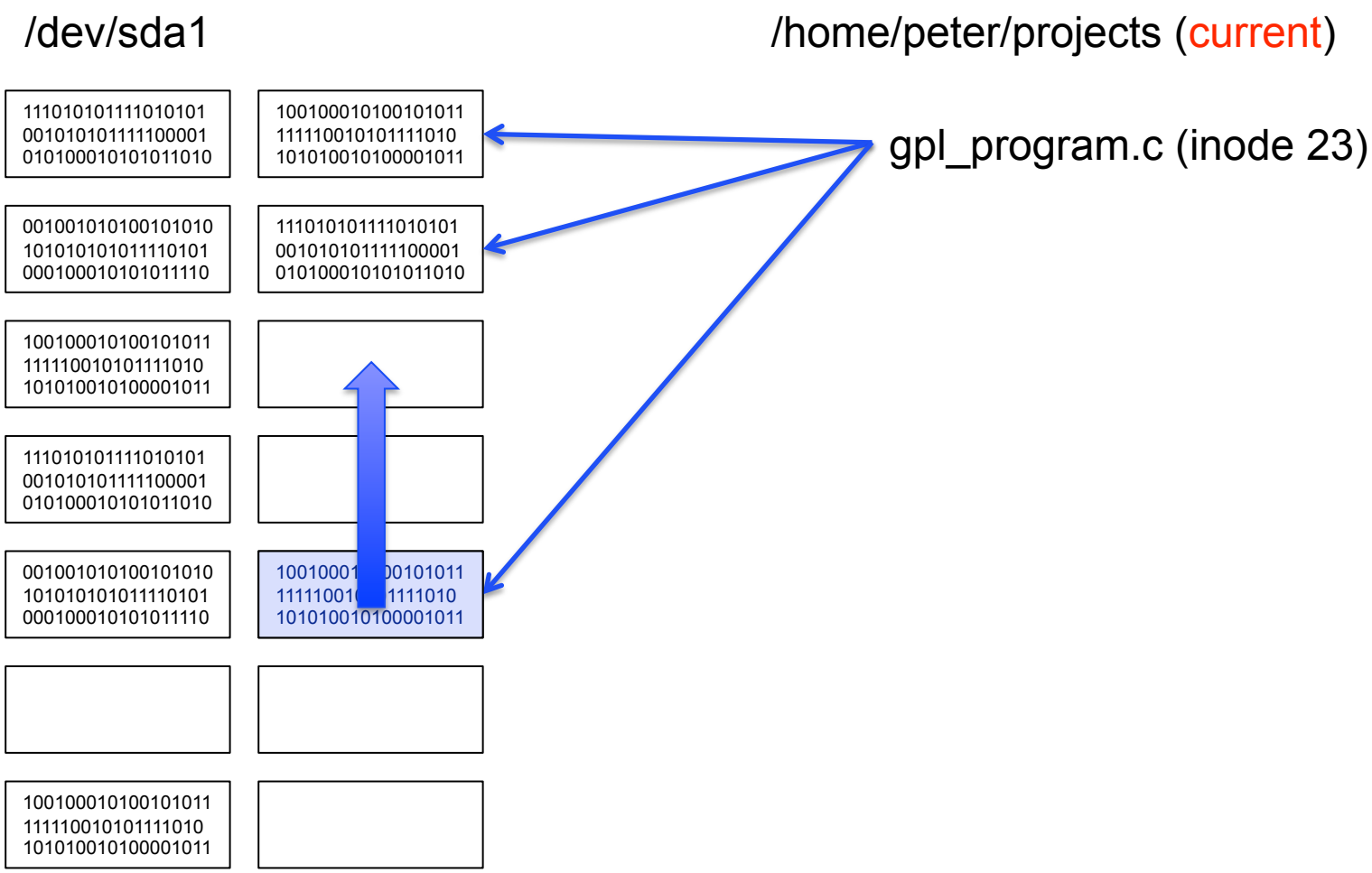
# Challenge: Read-Only Snapshots



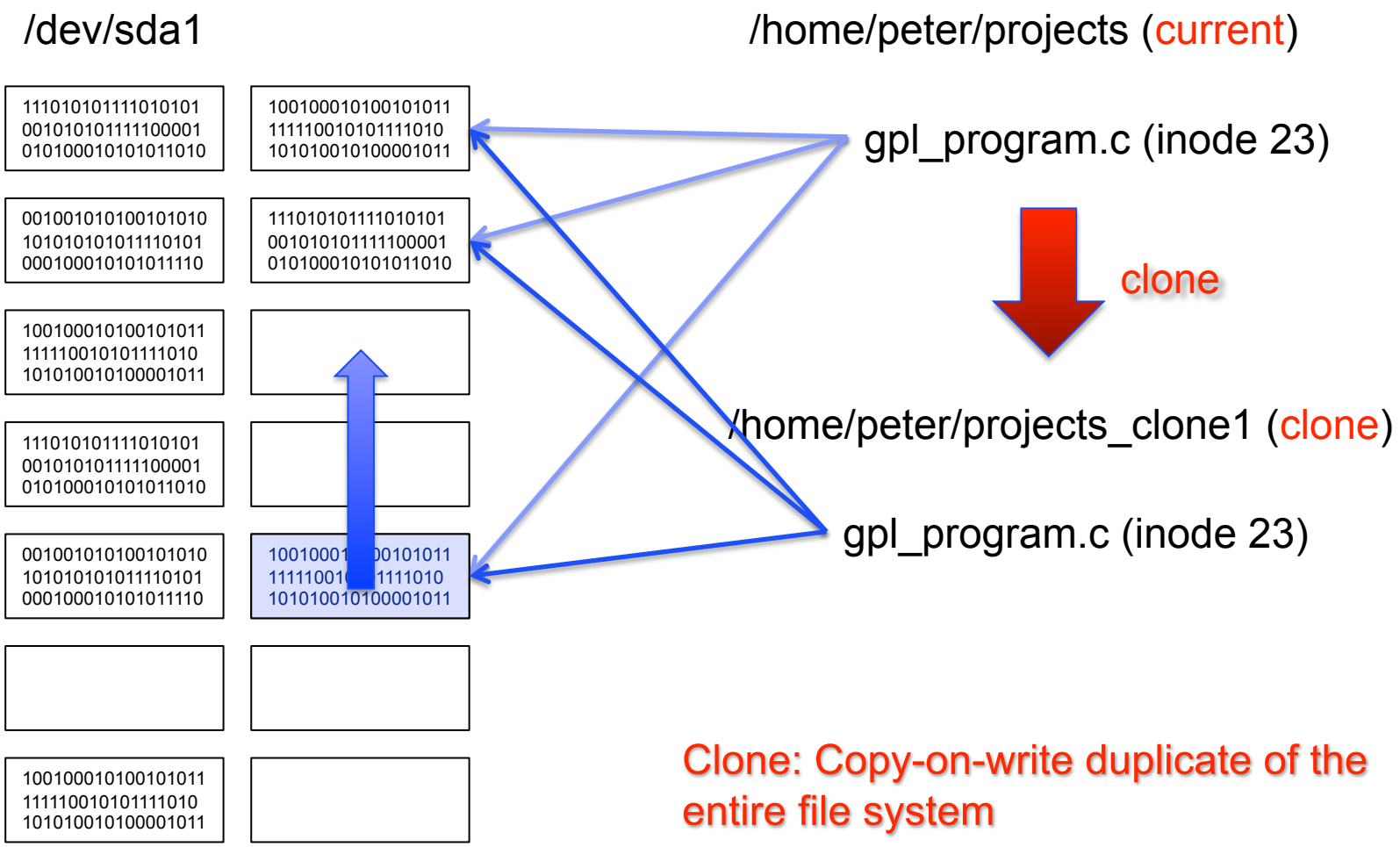
# Challenge: Read-Only Snapshots



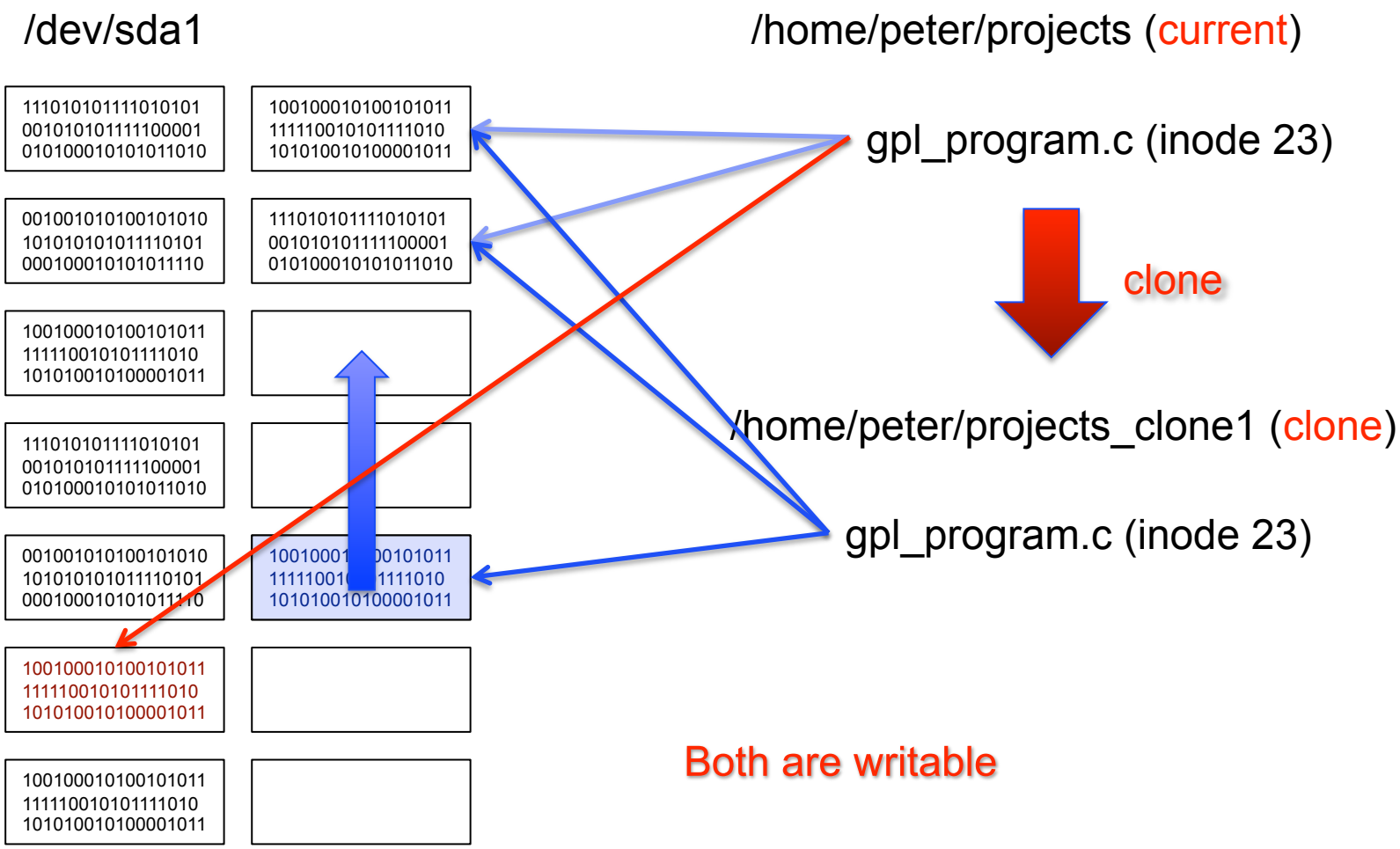
# Challenge: Writable Clones



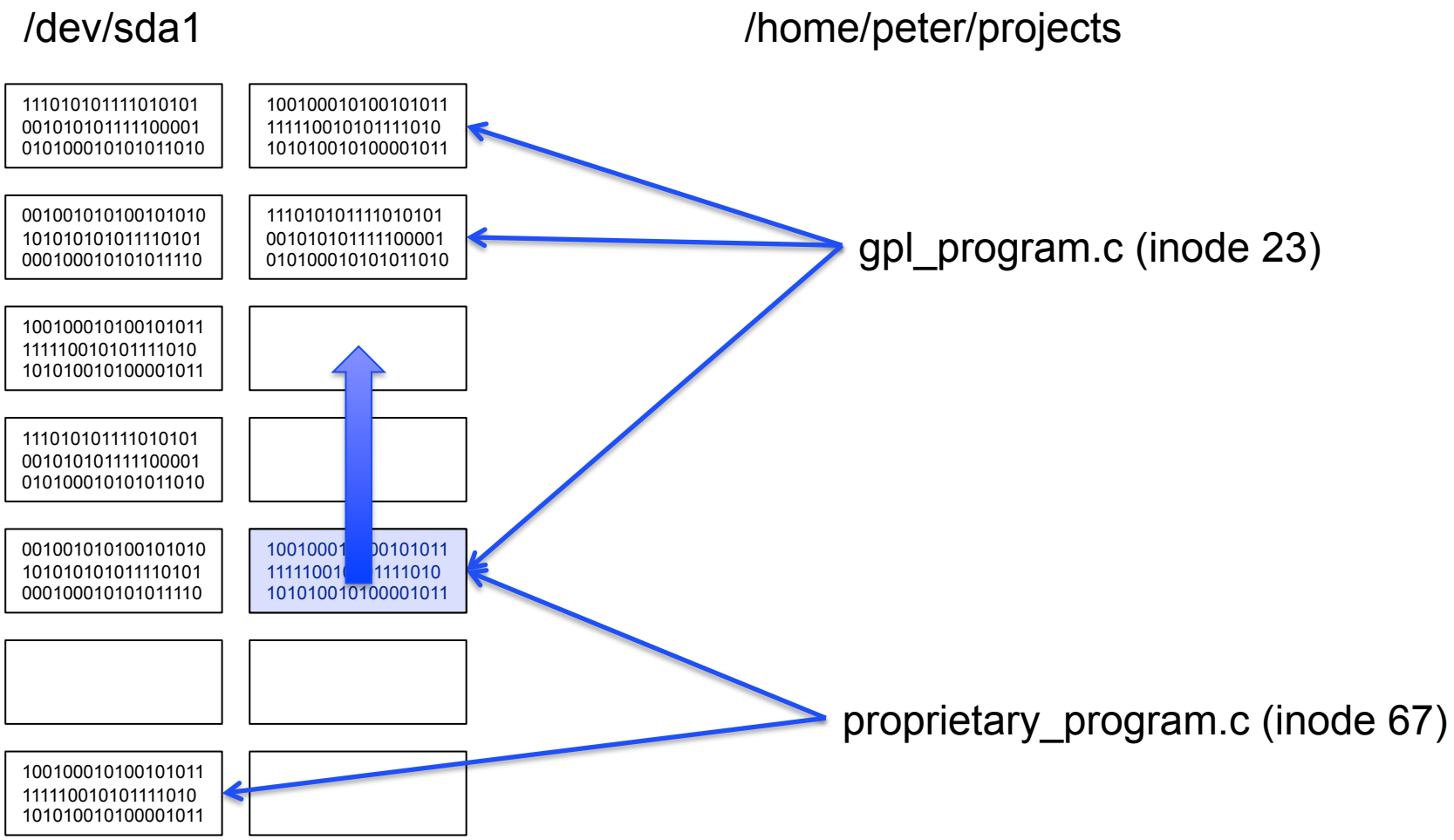
# Challenge: Writable Clones



# Challenge: Writable Clones



# Challenge: Deduplication (Block Sharing)



# Two Ways to Implement Back References



## BTRFS

- Very specific to btrfs
- Tightly-coupled with the extent allocator
- Theoretically the best you can do

## Backlog

- General approach
- Not specific to any particular file system
- Implemented in:
  - btrfs
  - ext3 (not in the paper)

# Backlog

- **Log-Structured Back References**
  - Perform all updates to back references using a single large sequential I/O
  - Avoid read-modify-write (to reduce seek)

# Outline

1. Introduction & Motivation
2. Challenges
3. Backlog Back References
4. Representing Writable Clones
5. Evaluation
6. Conclusion

# Conceptually

- Assume:
  - A write-anywhere file system
  - No support for writable clones (for now)

The time the block was **allocated** to the inode

**Back References**

| Block | Inode # | Offset | Time-Alloc | Time-Dealloc |
|---|---|---|---|---|

The time the block was **deallocated** from the inode (or ∞)

# Conceptually

- Assume:
  - A write-anywhere file system
  - No support for writable clones (for now)

The time the block was **allocated** to the inode

**Back References**

| Block | Inode # | Offset | Time-Alloc | Time-Dealloc |
|---|---|---|---|---|
| 2301 | 67 | 0 | 100 | ∞ |
| 2302 | 67 | 1 | 100 | 120 |

The time the block was **deallocated** from the inode (or ∞)

# The way we do it

- Keep logs for allocating and deallocating blocks (avoid read-modify-write):

**Operations:**

T=100 Allocate 2301

T=100 Allocate 2302

T=120 Free 2302

**Allocation Records**

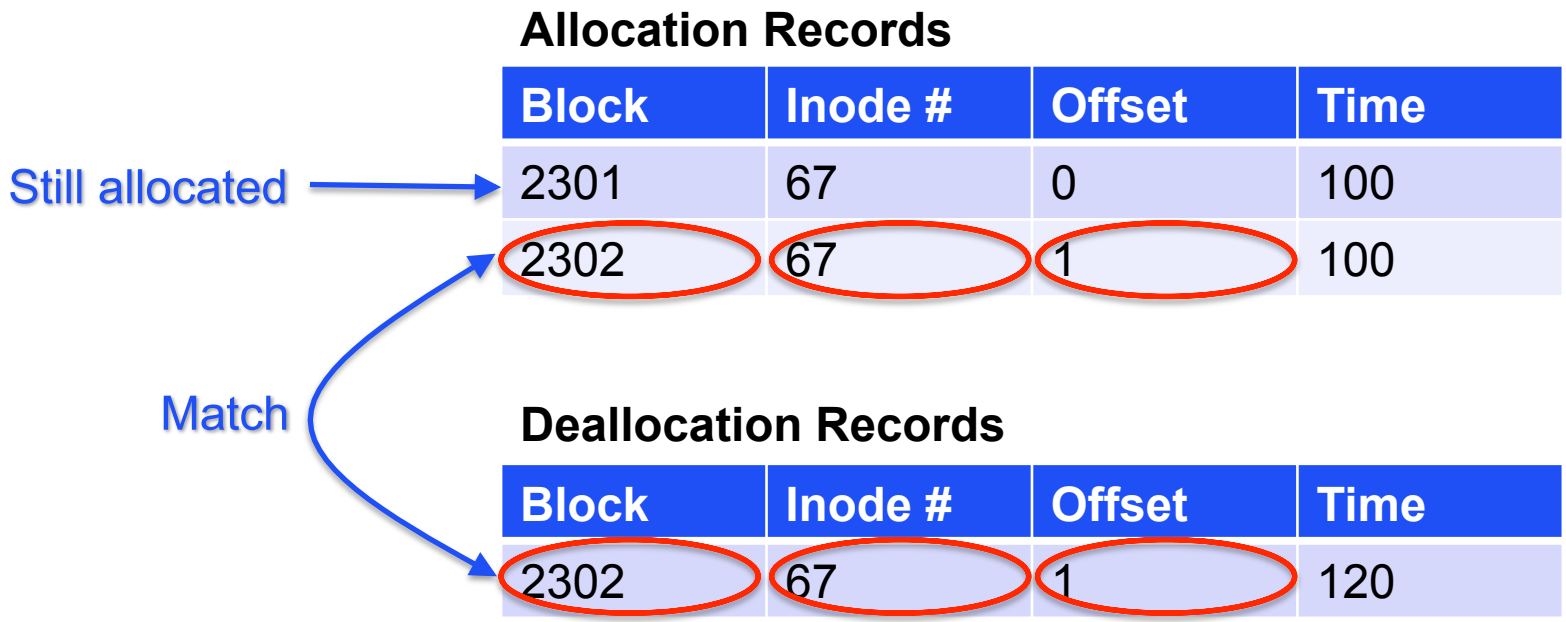
| Block | Inode # | Offset | Time |
|---|---|---|---|

**Deallocation Records**

| Block | Inode # | Offset | Time |
|---|---|---|---|

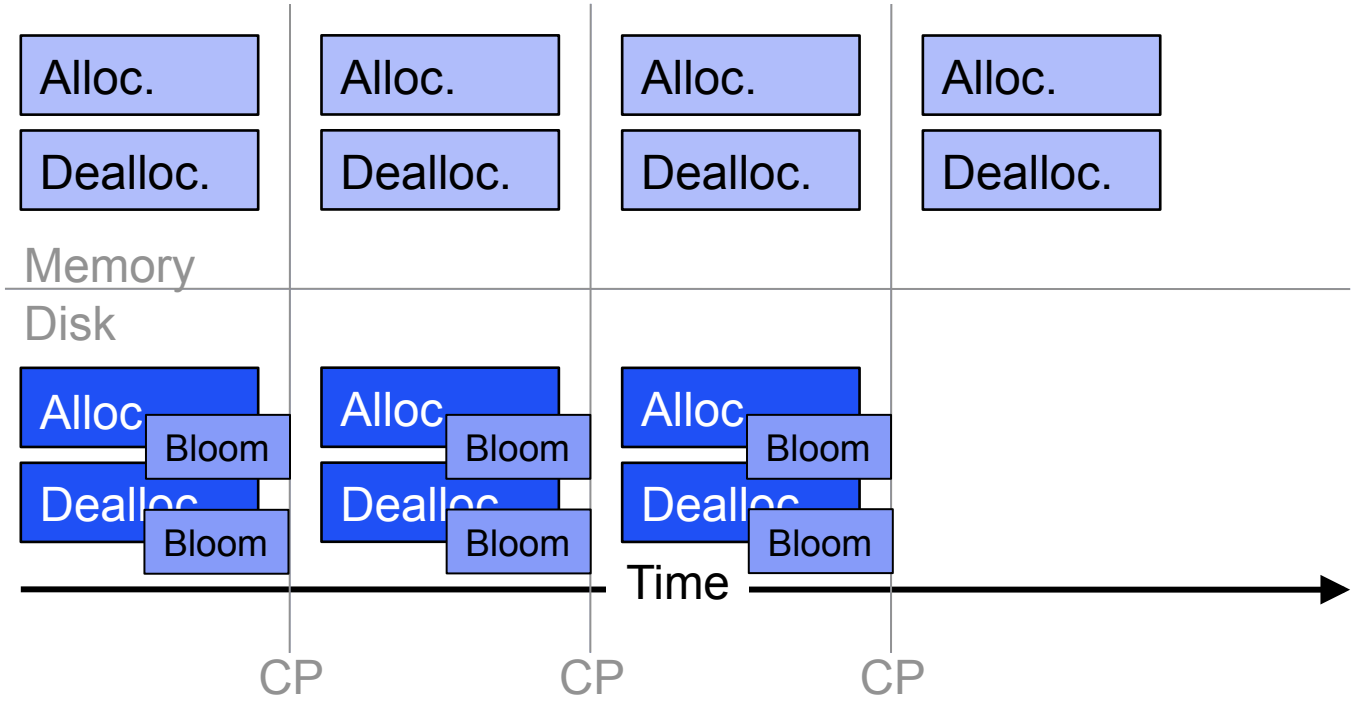
# The way we do it

- The conceptual view of back references can be efficiently reconstructed by comparing the two logs



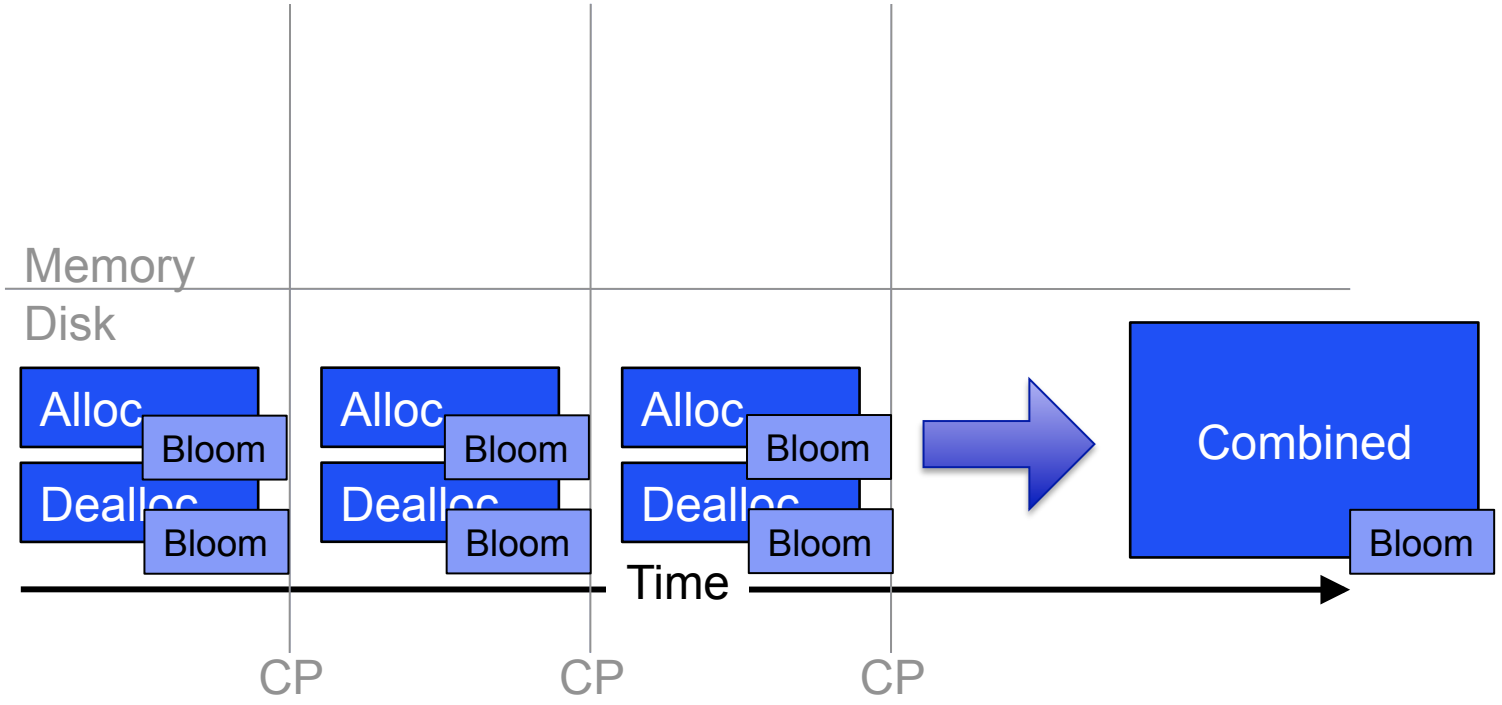
# Data Structures

- Build a B+Tree in memory, keyed by block no.
- Periodically flush to disk and start new tree
- Bloom filter per tree to speed up point queries



# Periodic Maintenance

- Merge the small trees into a single large tree
- Combine matching alloc. and dealloc. records
- Purge unneeded records

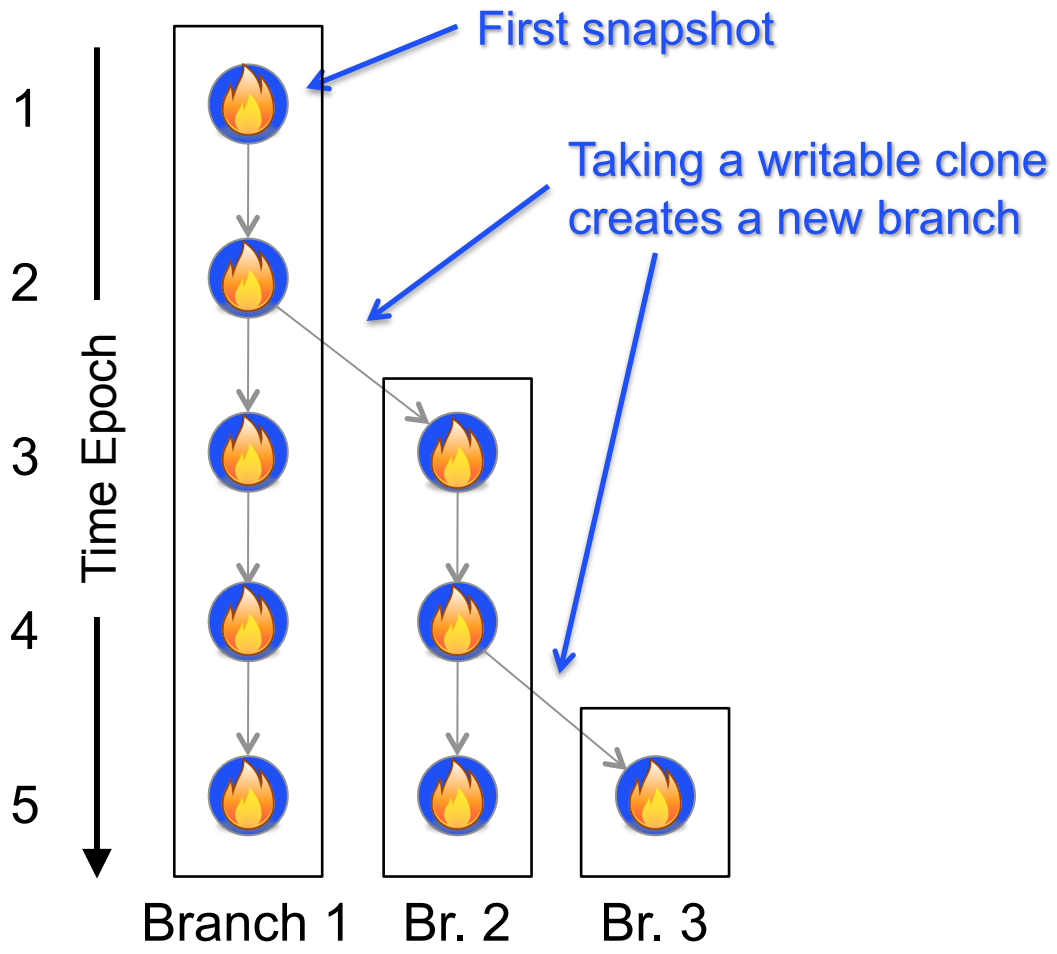


# Outline

1. Introduction & Motivation
2. Challenges
3. Backlog Back References
4. Representing Writable Clones
5. Evaluation
6. Conclusion

# Writable Clones

- Snapshots can be organized into a tree:



# Writable Clones

- Just add a Branch # field:

**Allocation Records**

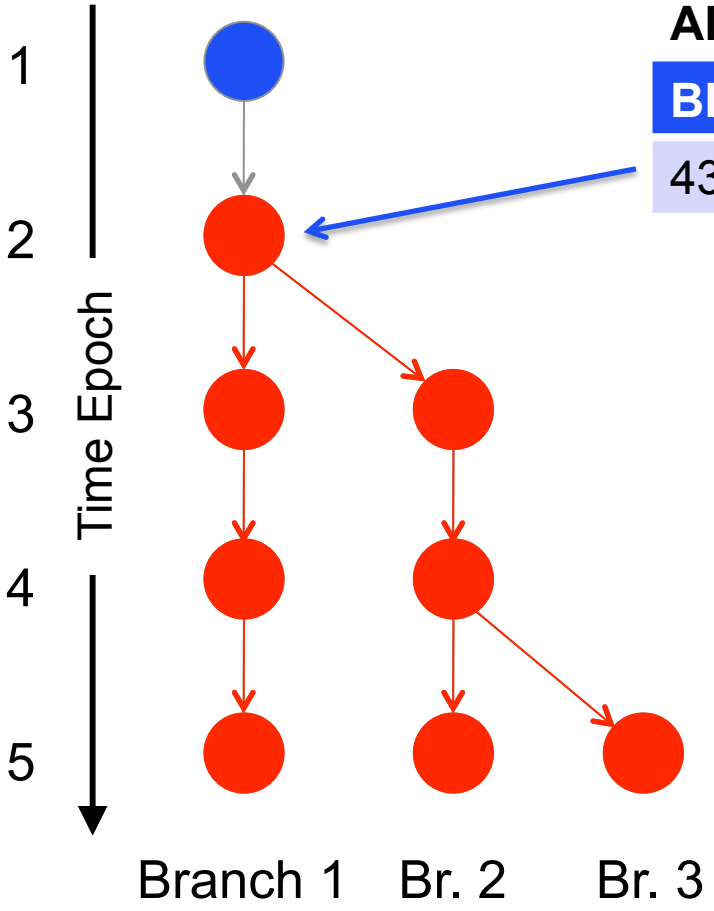
| Block | Inode # | Offset | Branch # | Time |
|---|---|---|---|---|
| 2301 | 67 | 0 | 1 | 100 |
| 2302 | 67 | 1 | 1 | 100 |

**Deallocation Records**

| Block | Inode # | Offset | Branch # | Time |
|---|---|---|---|---|
| 2302 | 67 | 1 | 1 | 120 |

# Writable Clones

- Structural Inheritance




**Allocation Records**

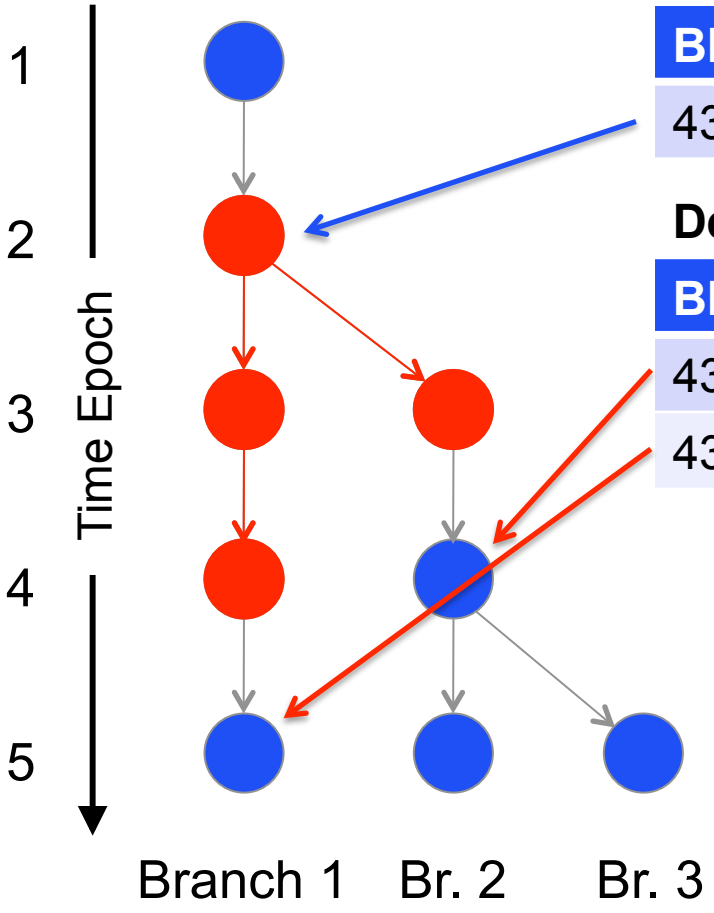
| Block | Inode # | Offset | Br. # | Time |
|---|---|---|---|---|
| 4381 | 39 | 0 | 1 | 2 |

In which snapshot is this block alive?

- Start at the snapshot identified by the record
- Just follow the arrows

# Writable Clones

- Structural Inheritance




**Allocation Records**

| Block | Inode # | Offset | Br. # | Time |
|---|---|---|---|---|
| 4381 | 39 | 0 | 1 | 2 |

**Deallocation Records**

| Block | Inode # | Offset | Br. # | Time |
|---|---|---|---|---|
| 4381 | 39 | 0 | 2 | 4 |
| 4381 | 39 | 0 | 1 | 5 |

In which snapshot is this block alive?

- Follow the arrows until you reach a deallocation record

# Outline

1. Introduction & Motivation
2. Challenges
3. Backlog Back References
4. Representing Writable Clones
5. Evaluation
6. Conclusion

# Evaluation: Key Metrics

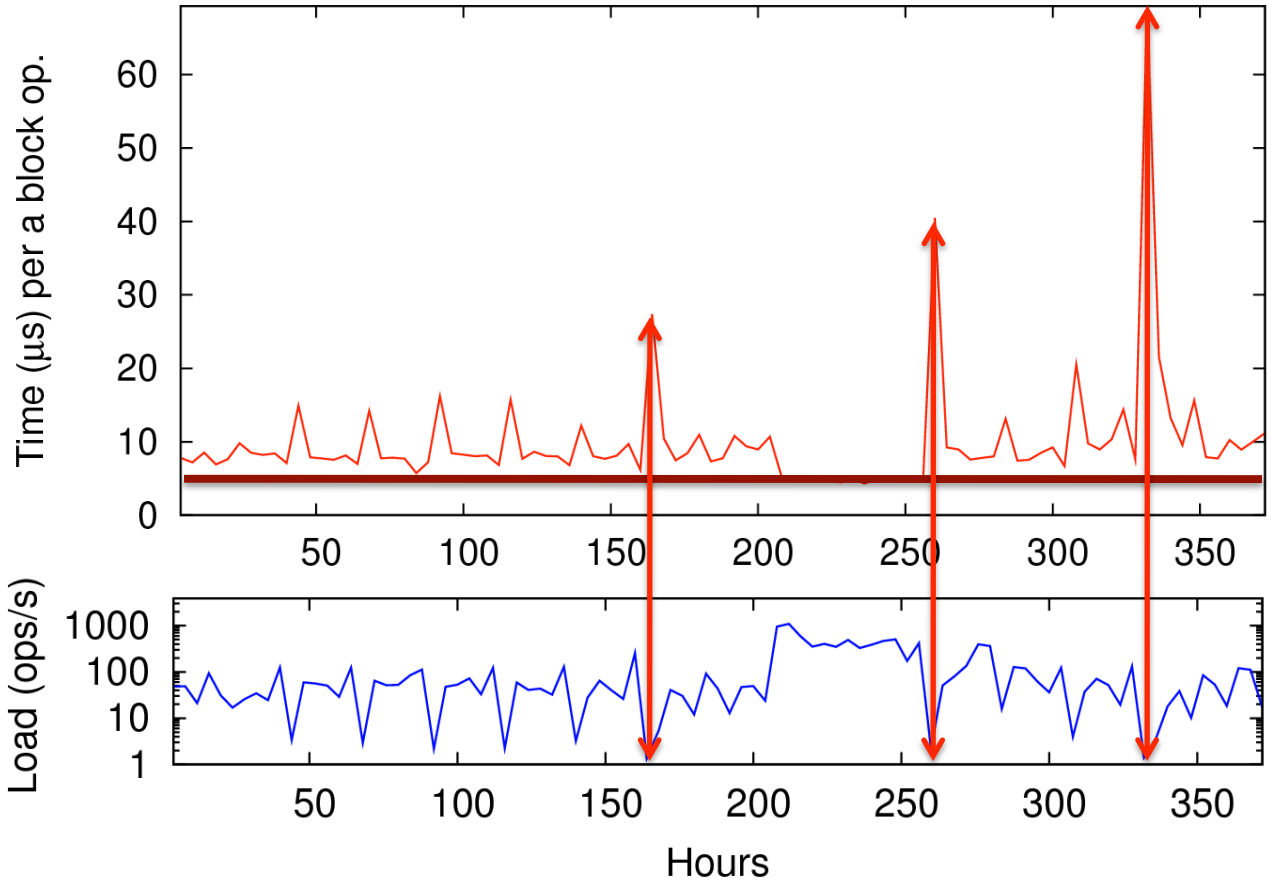
- Time overhead
- Space overhead
- Comparison with BTRFS-specific implementation

# Evaluation: Benchmarks

- File System Simulator
  - Synthetic Workloads
  - NFS Trace (EECS03, home directories)

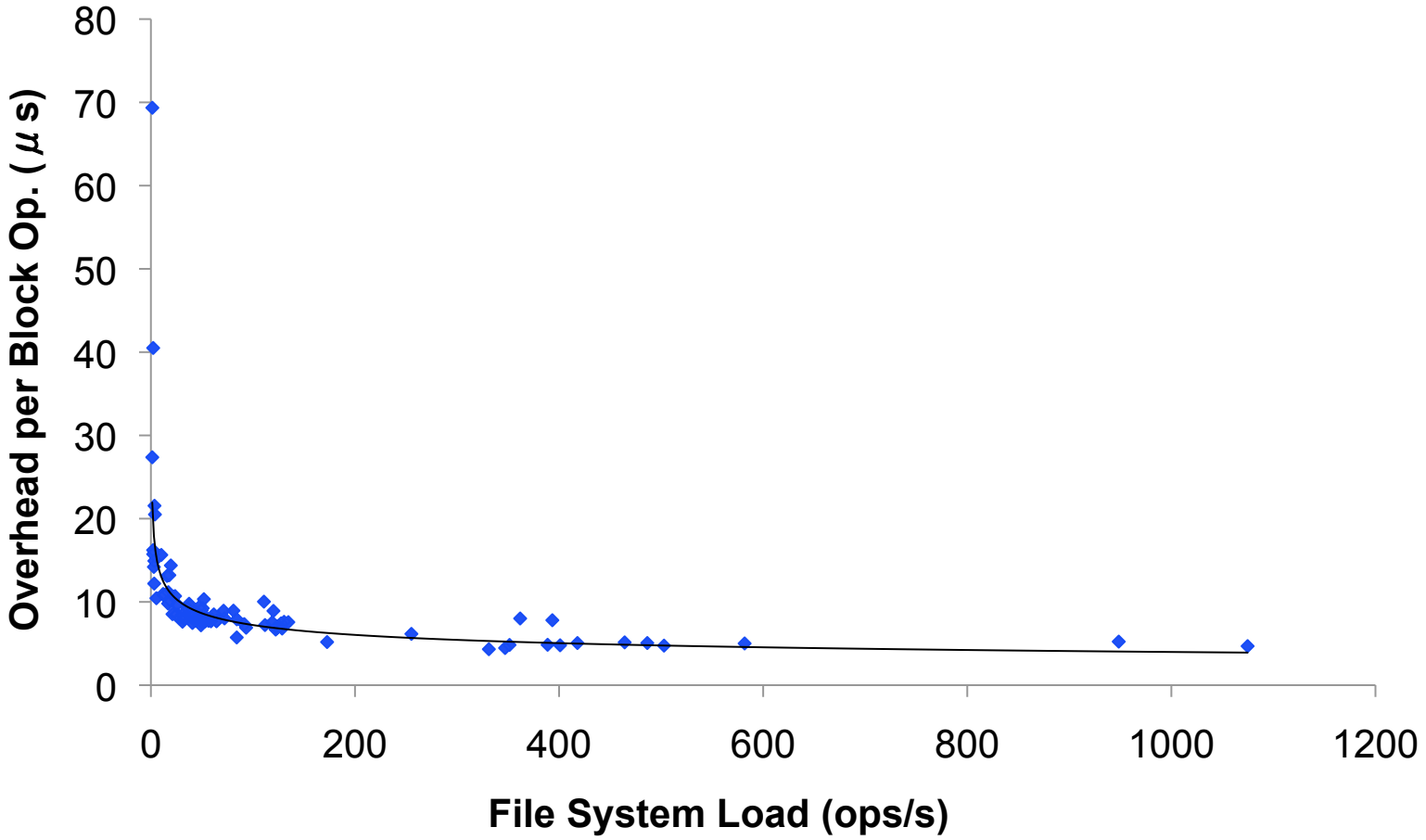
- Customized BTRFS (where we replaced its file-system-specific back references by our implementation)
  - Microbenchmarks
  - Application Benchmarks

# NFS Trace: Overhead over Time



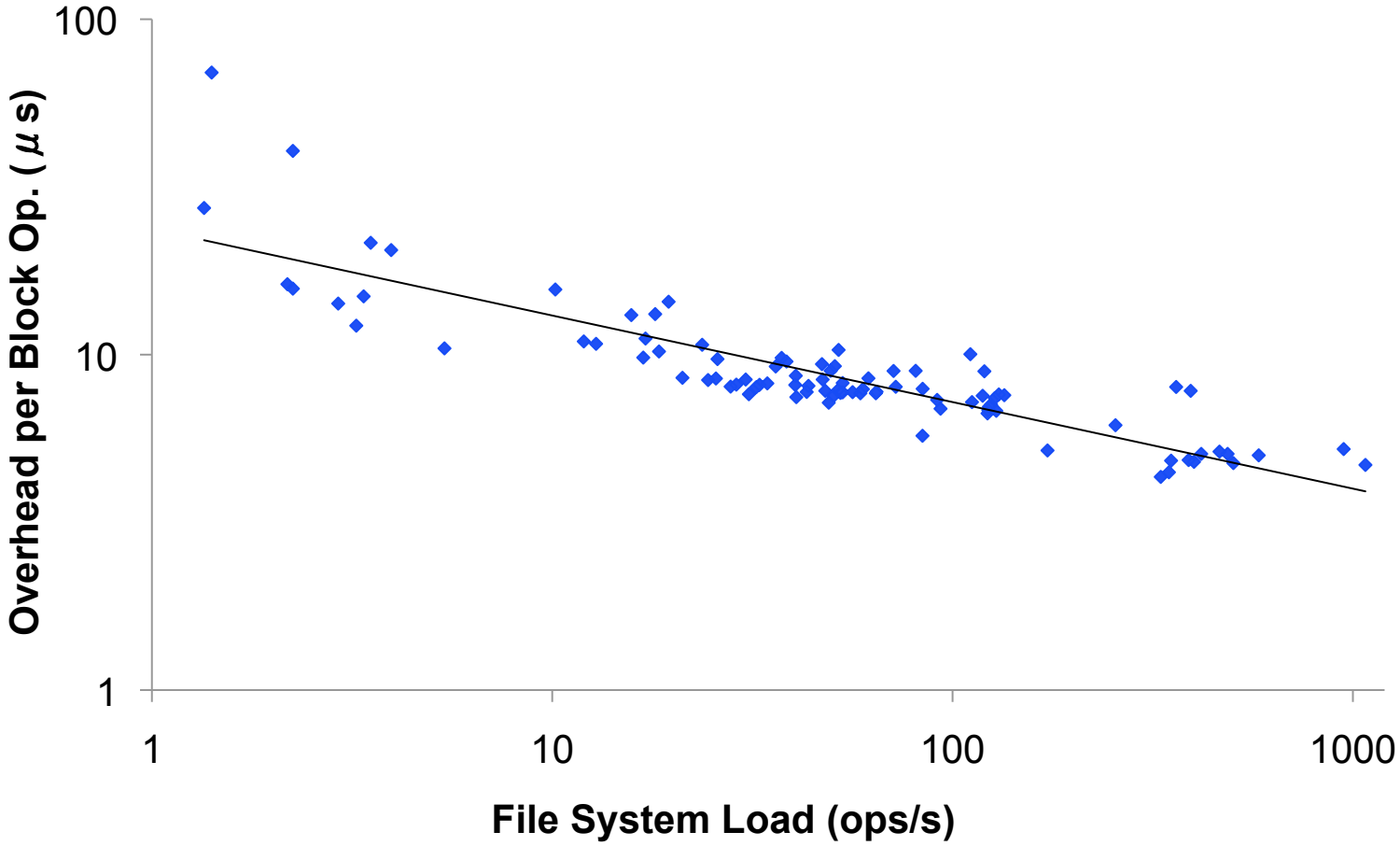

- No performance degradation over time
- Spikes correspond to the periods of low file system activity

# NFS Trace: Overhead vs. File System Load



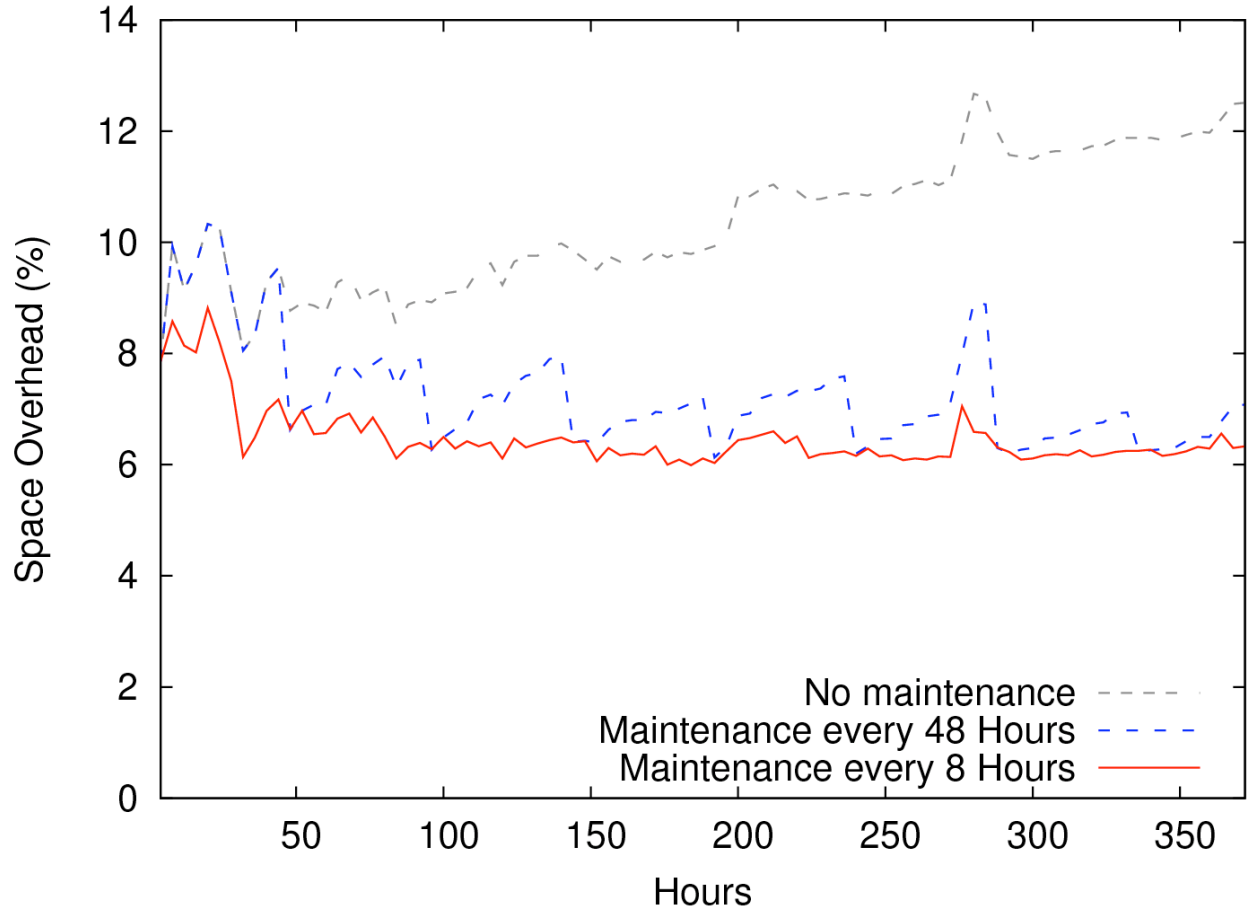

- Time overhead per block operation is inversely proportional to the file system load

# NFS Trace: Overhead vs. File System Load



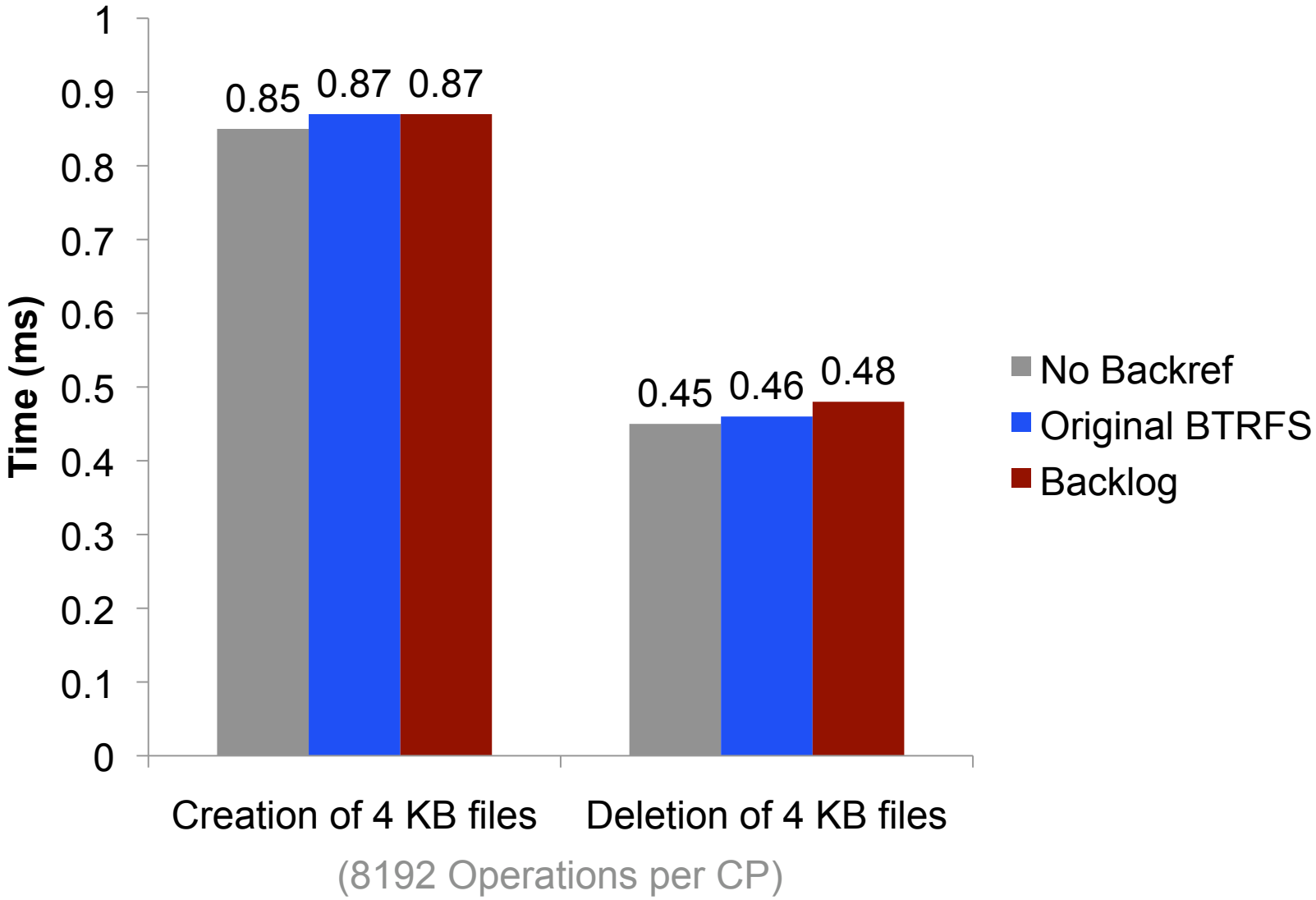

- Time overhead per block operation is inversely proportional to the file system load

# NFS Trace: Space Overhead



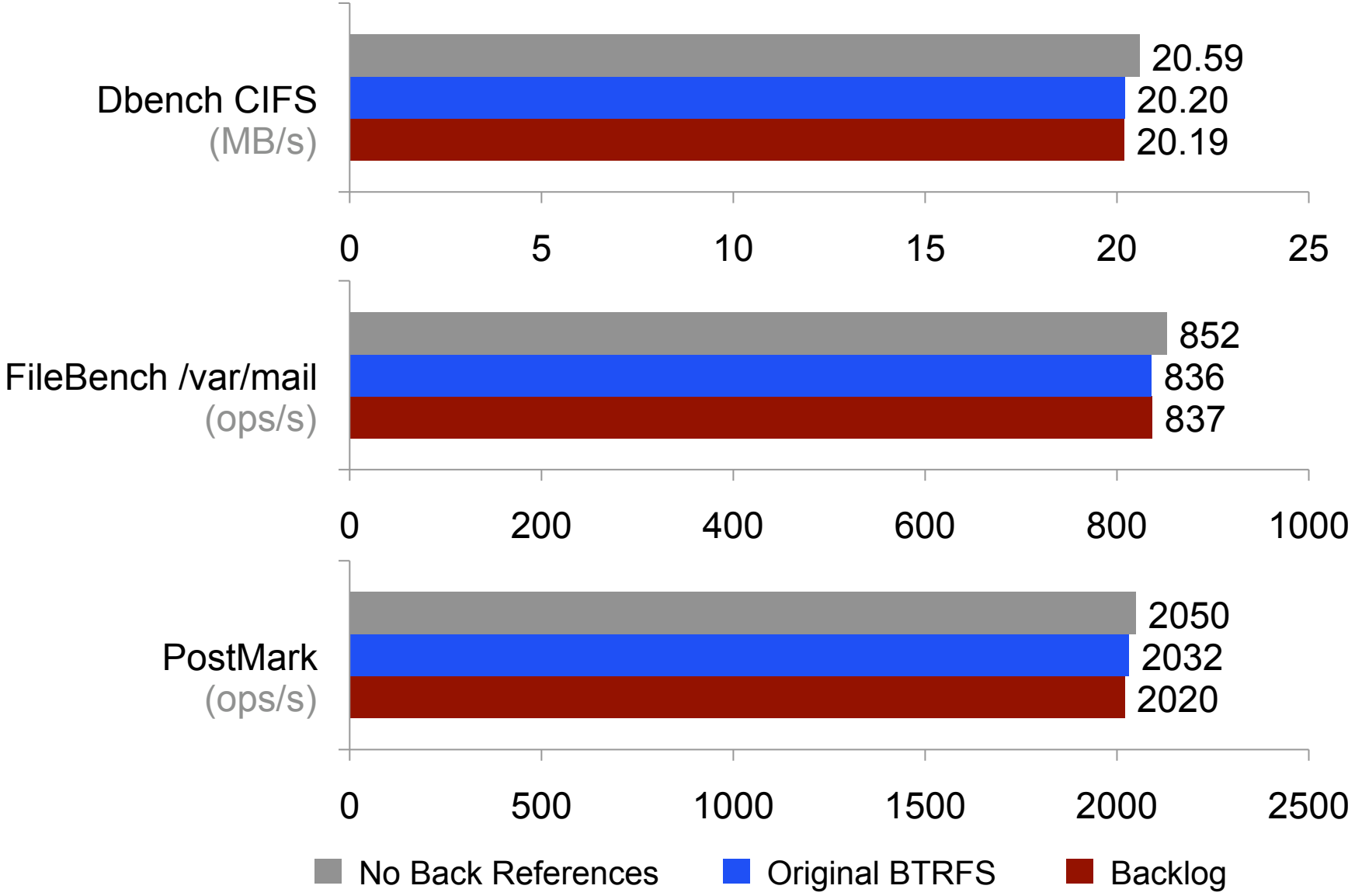

- Space overhead: 6.3%, stable over time
- Maintenance: at most 24.1 seconds

# BTRFS: Microbenchmarks




(8192 Operations per CP)

# BTRFS: Application Benchmarks



# Now that we have back references…

- Back references do not cost much
- What is the cost to use them?
  - Find all owners of a block
  - Find owners of all blocks in a region

# Query Performance

- Average query latency after maintenance:
  - Random queries: 5.07 ms per block
  - Sequential queries: 0.03 ms per block

Can read back references for a 100 MB block group (25,600 blocks) in 710 ms

- No back references – need to build inverted index from scratch:
  - 10 TB disk, 8 byte pointers → 20 GB of pointers
  - 1 ms per random I/O → 1½ hours

# Outline

1. Introduction & Motivation
2. Challenges
3. Backlog Back References
4. Representing Writable Clones
5. Evaluation
6. Conclusion

# Conclusions

- Back references are feasible
  - Minimal overhead
  - Fast query performance

- Backlog, a general approach
  - Write-optimized allocation and deallocation logs
  - Updates performed by large sequential writes
  - Using structural inheritance to represent writable clones

# Backlog: Tracking Back  References in a Write-Anywhere File System


Peter Macko
Harvard University
Cambridge, MA

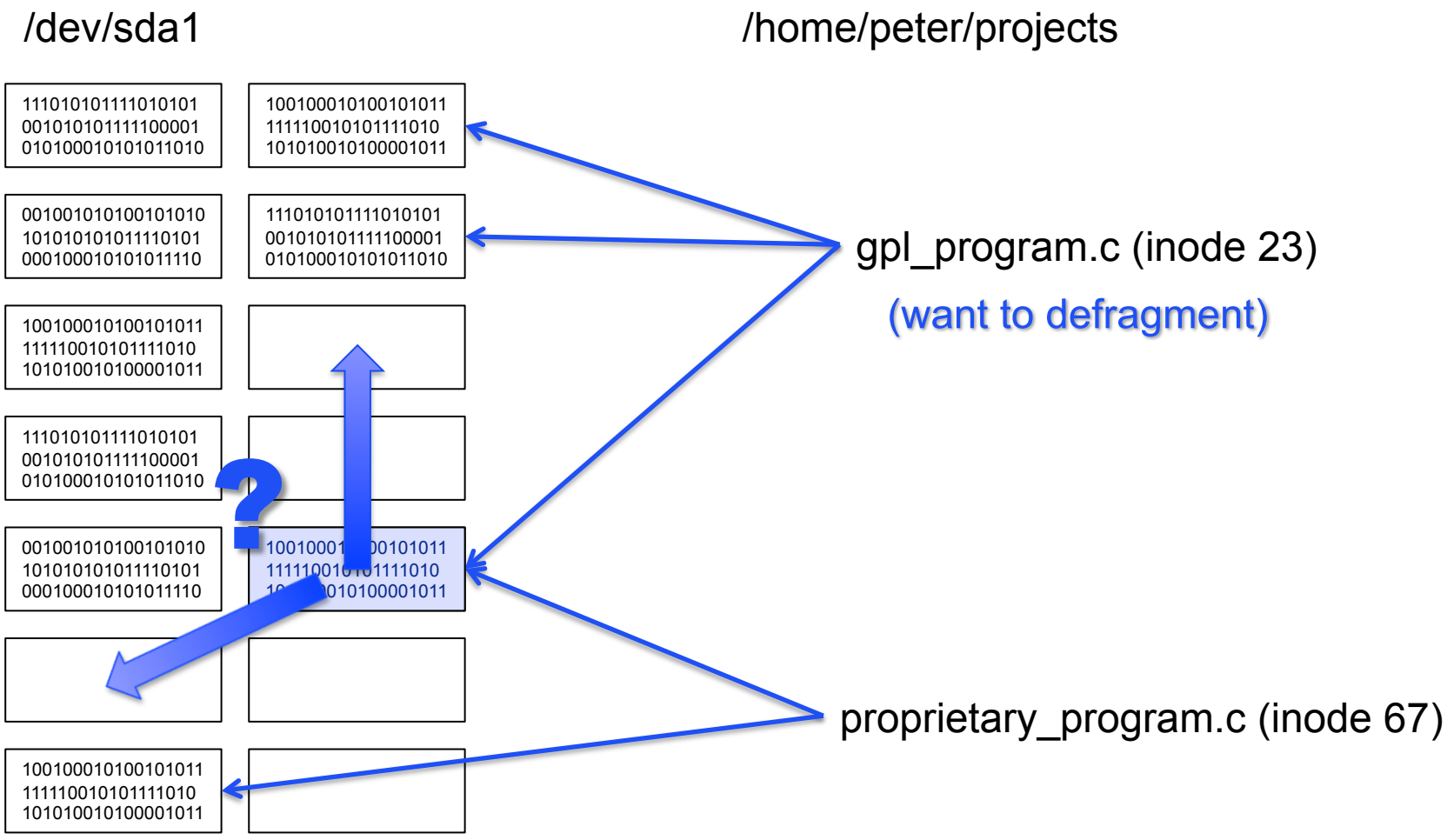
Margo Seltzer
Harvard University
Cambridge, MA

Keith Smith
NetApp, Inc.
Waltham, MA

{pmacko, margo}@eecs.harvard.edu, keith.smith@netapp.com

# BACKUP SLIDES

# Challenge: Context-Aware Defragmentation



# Extent-Based File Systems

- Just add a Length field!

**Allocation Records**

| Block | Length | Inode # | Offset | Branch # | Time |
|---|---|---|---|---|---|
| 2301 | 10 | 67 | 0 | 1 | 100 |
| 2310 | 8 | 67 | 10 | 1 | 100 |

**Deallocation Records**

| Block | Length | Inode # | Offset | Branch # | Time |
|---|---|---|---|---|---|
| 2310 | 8 | 67 | 10 | 1 | 120 |

Represents blocks 2310 – 2318

# Evaluation: Platform

- Simulator:
  - Intel Xeon 3.0 GHz
  - 10 GB of RAM
  - 60 MB/s disk throughput
- BTRFS:
  - Intel Pentium 4, 3.0 GHz
  - 512 MB of RAM
  - 50 MB/s disk throughput

# Query Performance

- Average query latency after maintenance:
  - Random queries: 5.07 ms per block
  - Sequential queries: 0.03 ms per block

Can read back references for a 100 MB block group (25,600 blocks) in 710 ms

- Query latency 12.5 mil. block writes (50 GB) since last maintenance:
  - Random queries: 11.7 ms per block
  - Sequential queries: 2.1 ms per block

# Keeping Track of Allocated Blocks

- Typical file system
  - Block allocation bitmap
- BTRFS
  - Each allocated extent is described by a variable-length record
  - Back references are stored as a part of the record