BetrFS: A Right-Optimized Write-Optimized File System

Amogh Akshintala, Michael Bender, Kanchan Chandnani, Pooja Deo, Martin Farach-Colton, William Jannen, **Rob Johnson**, Zardosht Kasheff, Bradley C. Kuszmaul, Prashant Pandey, Donald E. Porter, Leif Walsh, Jun Yuan, Yang Zhan

Facebook, Farmingdale College, MIT & Oracle, Rutgers, Stony Brook, Two Sigma, UNC, Williams College

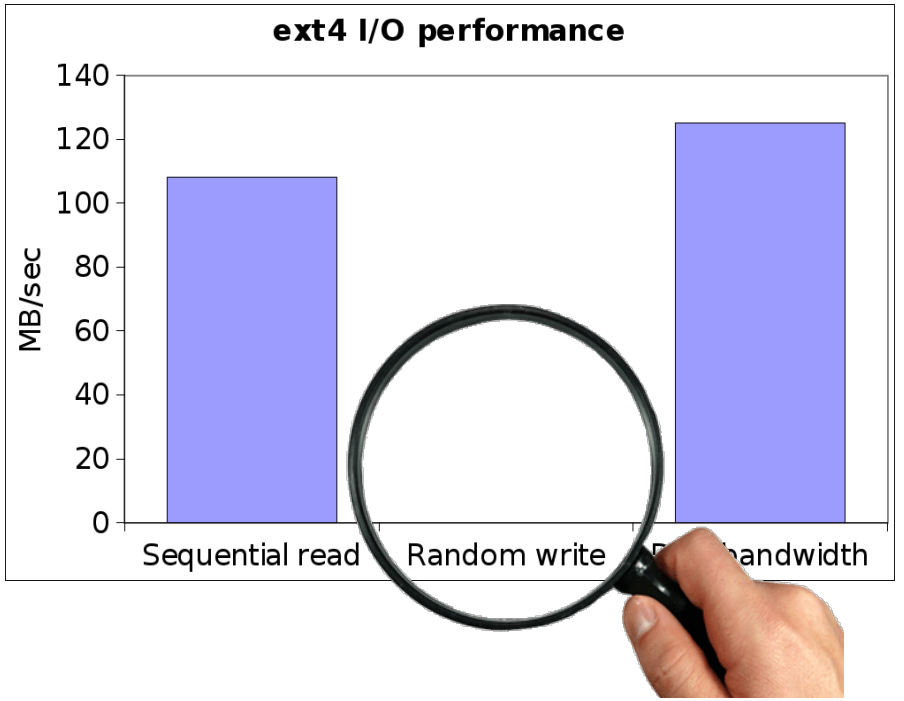
General-purpose file-systems strive to perform well on a wide variety of applications

- Sequential reads
- Sequential writes
- Random writes
- File/directory renames
- File deletes
- Recursive scans
- Metadata updates

Achieving good performance on all these operations is a long-standing challenge

- ✓ Sequential reads
- ✓ Sequential writes
- ✗ Random writes
- ✓ File/directory renames
- ✓ File deletes
- ✗ Recursive scans
- ✗ Metadata updates

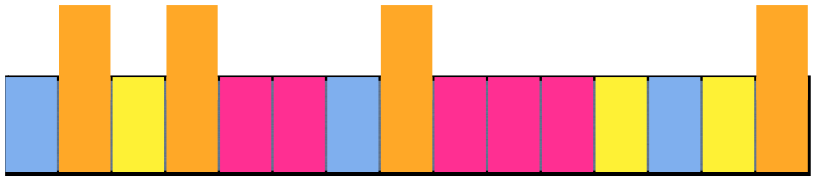
Example: ext4



Achieving good performance on all these operations is a long-standing challenge

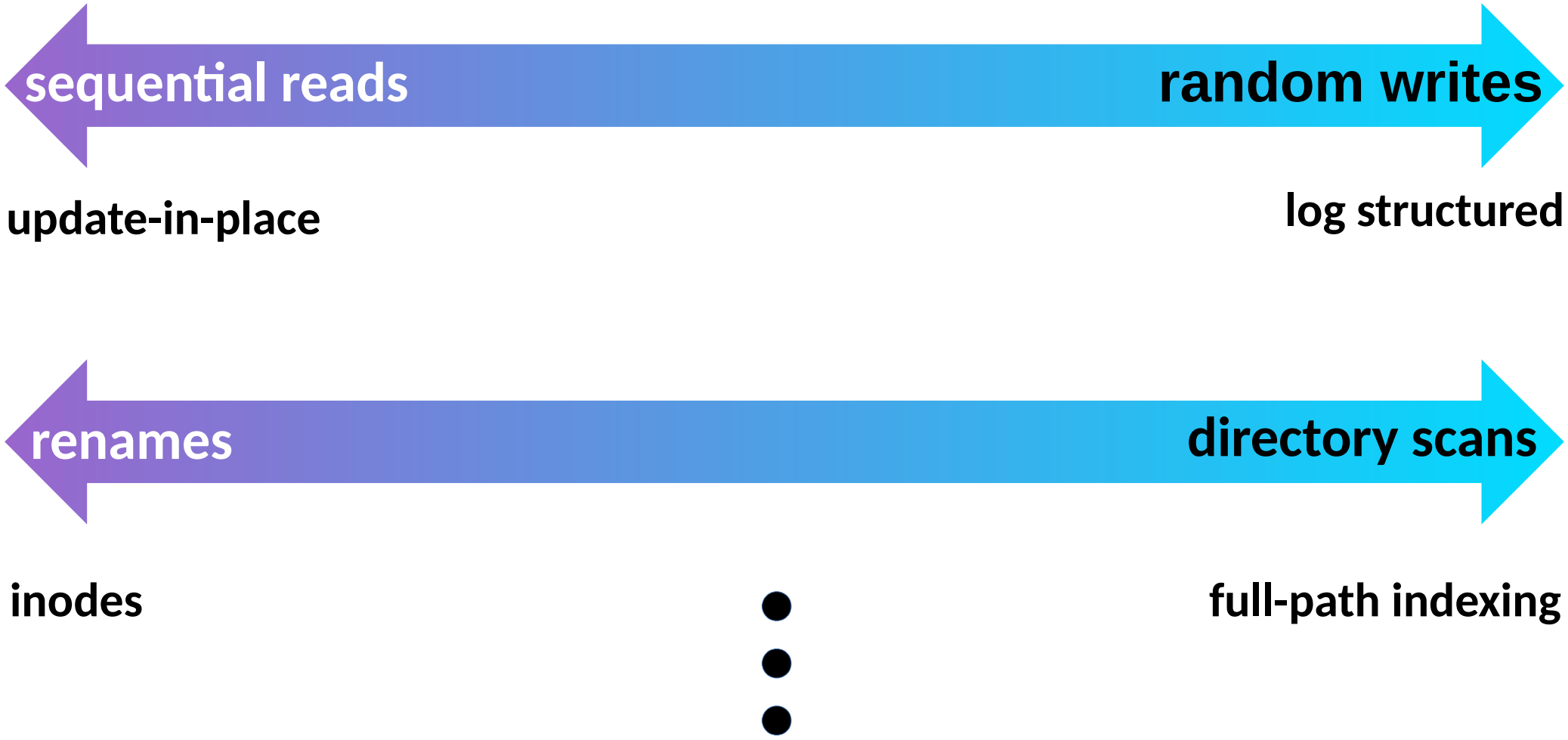
- ✗ Sequential reads
- ✓ Sequential writes
- ✓ Random writes
- ✓ File/directory renames
- ✓ File deletes
- ✗ Recursive scans
- ✓ Metadata updates

Example: log-based file systems

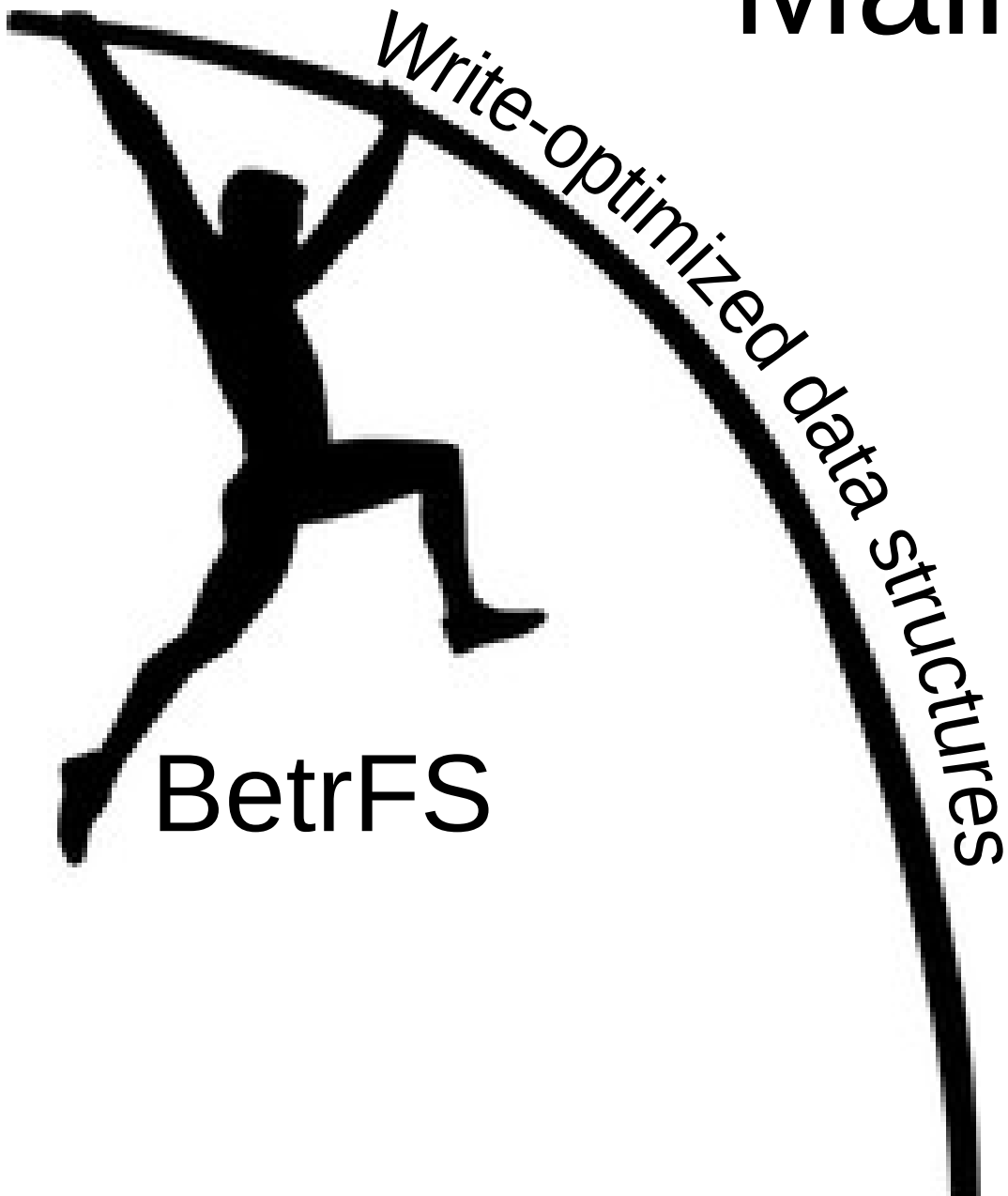


Logging updates is fast, but logged data can have little locality

Some operations seem to require a trade-off



Main idea of this talk



Trade-offs

Write-Optimized Data Structures (WODS)

- New class of data structure[illegible]ed in 90's
  - LSM trees [O'Neil, Cheng, Gawlick, & O'Neil '96]
  - Bε-trees [Brodal & Fagerberg '03]
  - COLAs [Bender, Farach-Colton, Fineman, Fogel, Kusz[illegible]]
  - xDicts [Brodal, Demaine, Fineman, Iacono, Langerman, & Munro '10]
- WODS perform inserts/updates/deletes orders-of-magnitude-faster than in a B-tree
  - WODS queries are asymptotically no slower than in a B-tree

BetrFS uses Bε-trees

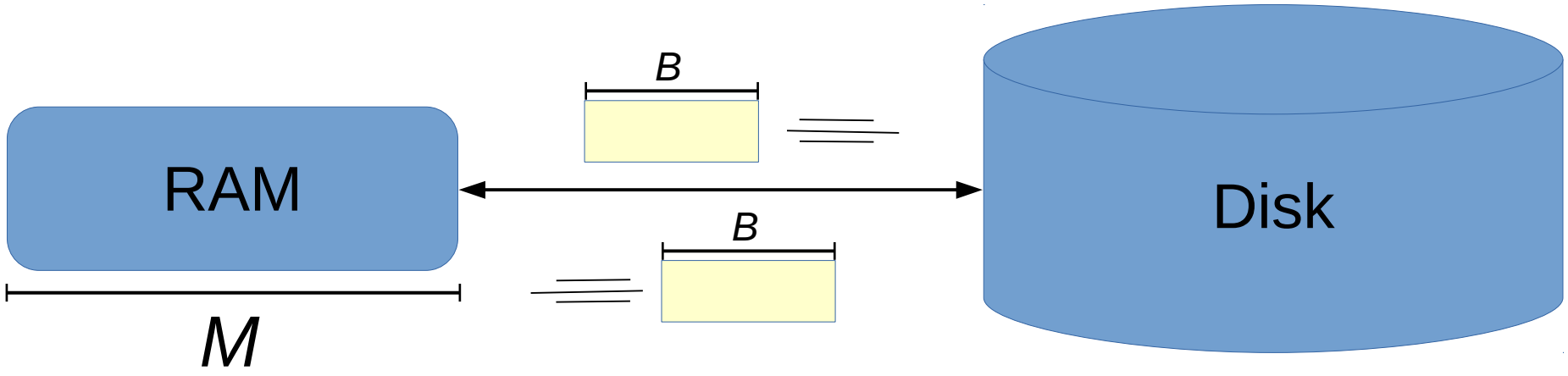
The Disk-Access Machine (DAM) model [Aggarwal & Vitter '88]

How computation works:

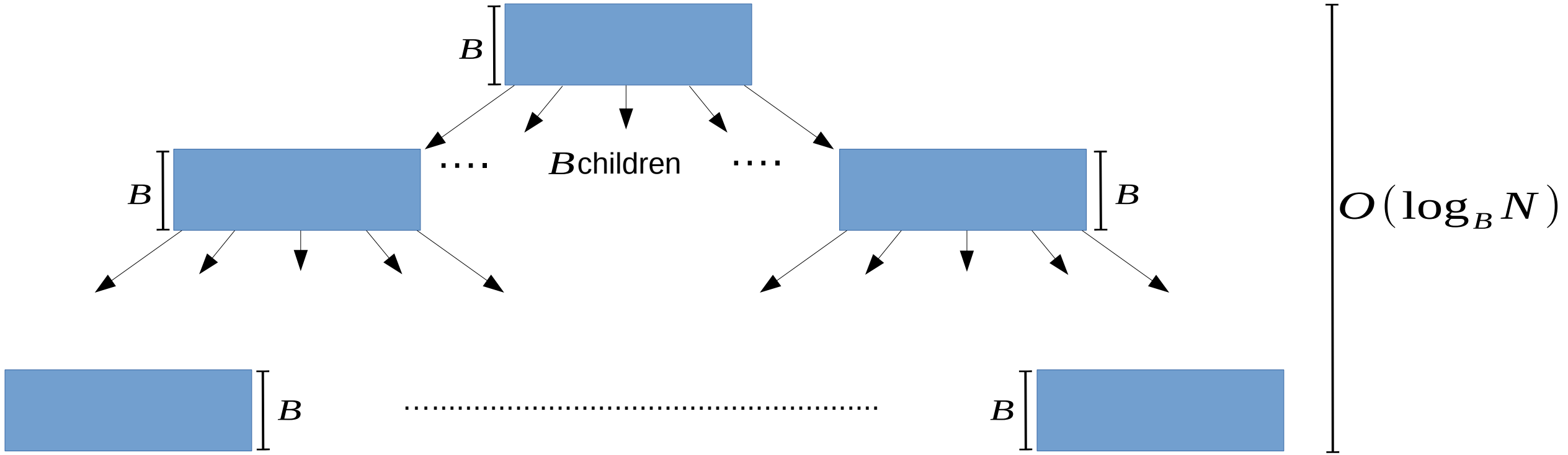
- Data is transferred in blocks between RAM and disk.
- The number of block transfers dominates the running time.

Goal: Minimize # of block transfers

- Performance bounds are parameterized by block size *B*, memory size *M*, data size *N*.



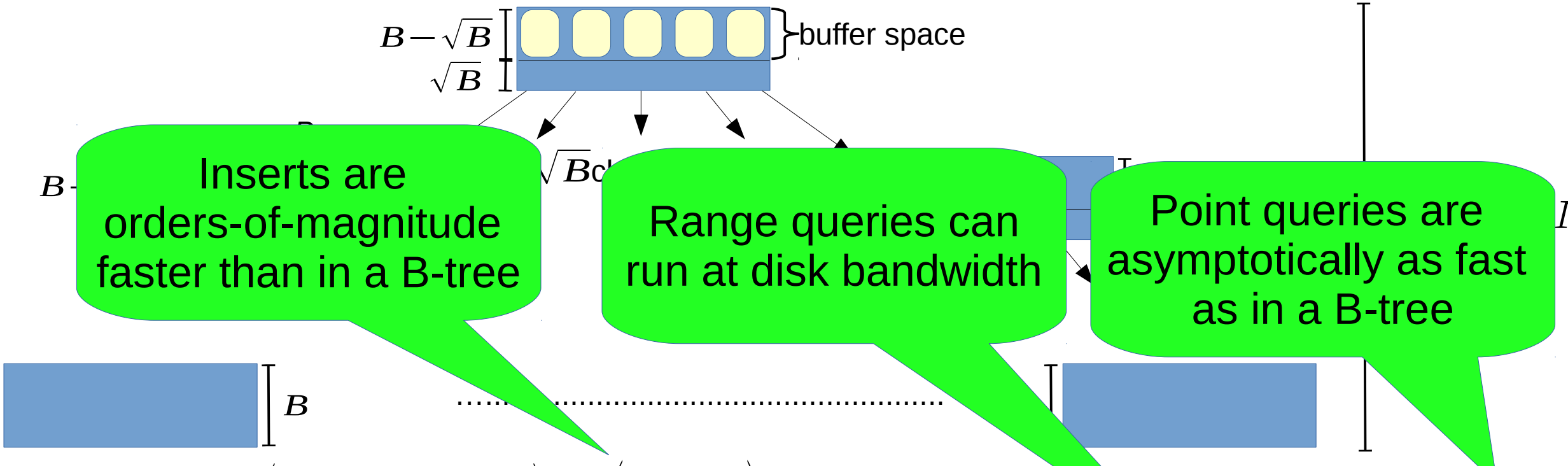
Example: B-trees



Insert cost: $O(\log_B N)$

Lookup cost: $O(\log_B N)$

B^ε-trees ($\varepsilon=1/2$)



Inserts are orders-of-magnitude faster than in a B-tree

Range queries can run at disk bandwidth

Point queries are asymptotically as fast as in a B-tree

Insert cost: $O\left(\frac{\sqrt{B}}{B-\sqrt{B}}\log_{\sqrt{B}} N\right)=O\left(\frac{\log_B N}{\sqrt{B}}\right)$

Lookup cost: $O(\log_{\sqrt{B}} N)=O(\log_B N)$

Range query cost: $O(\log_B N+k/B)$

The search-insert asymmetry

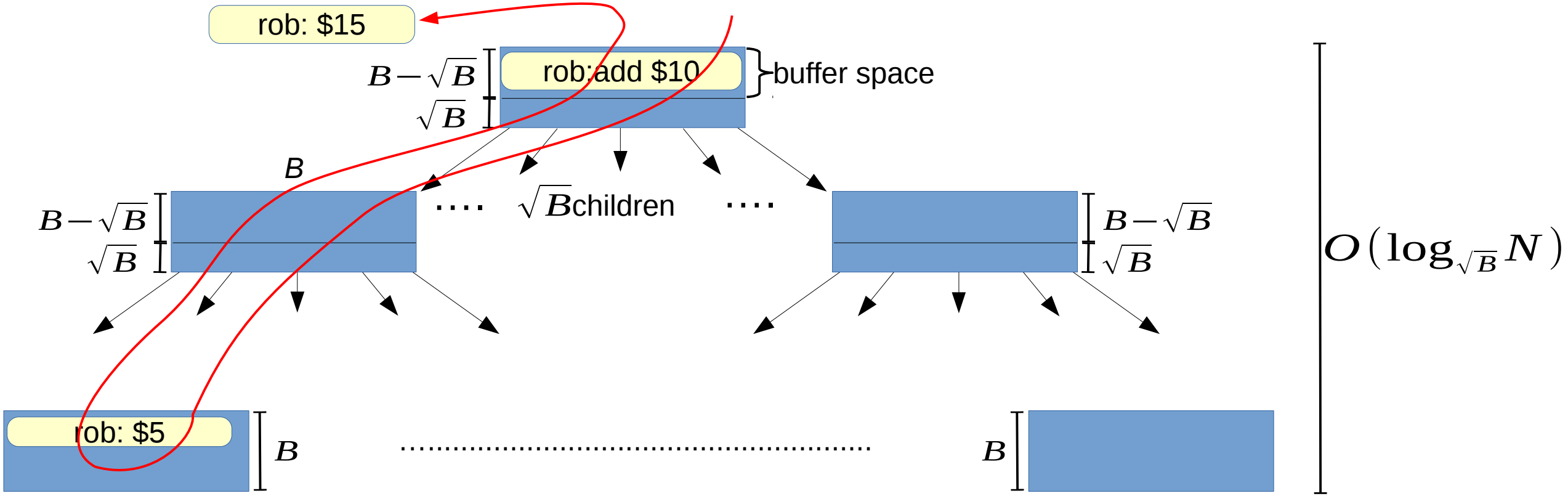
- Inserts are orders-of-magnitude faster than point queries

$$\text{Point query:} \quad O\left(\log_B N\right)$$

$$\text{Insert:} \quad O\left(\frac{\log_B N}{\sqrt{B}}\right)$$

- But many updates require querying the old value first
- e.g. “Add $10 to rob's account balance”
  - OldBalance = query(rob)
  - NewBalance = oldBalance + $10
  - insert(newBalance)

Upserts: read-modify-write as fast as an insert



B$^{\varepsilon}$-tree performance summary

Point query	$O\left(\log_B N\right)$	As fast as a B-tree
Insert/delete/upsert	$O\left(\frac{\log_B N}{\sqrt{B}}\right)$	Very fast (10K-100K per second)
Range query	$O\left(\log_B N + k/B\right)$	Near disk bandwidth

To get the best possible performance, we want to do

Inserts, deletes, upserts, and range queries, and avoid point queries.

The BetrFS schema (version 0.1)

- Maintain two separate B^{ε}-tree indexes:

  metadata index: `full path -> struct stat`
  data index: `(full path, blk#) -> data[4096]`

- Implications:
  - Fast directory scans
  - Data blocks are laid out sequentially

Mapping file-system operations to key-value operations

Operation	Implementation
read	range query
write	insert/upsert
metadata update	upsert
readdir	range query
mkdir/rmdir	insert/delete
unlink	*delete each block
rename	*delete then reinsert each bl

Fast atime updates

Fast directory traversals

Do not map onto single key-value operations

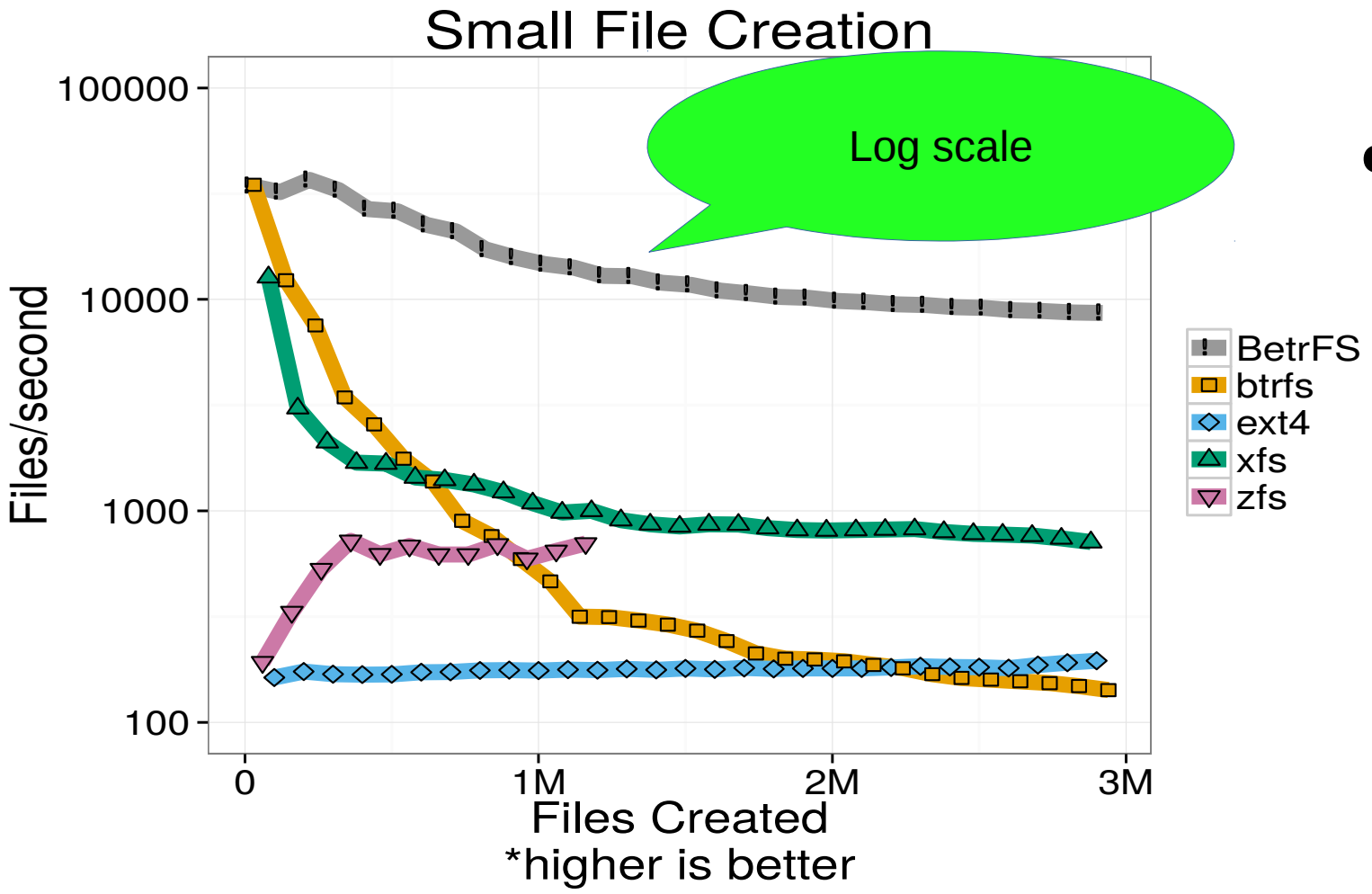
Small, random, unaligned writes are an order-of-magnitude faster

*lower is better

- 1 GiB file, random data
- 1,000 random 4-byte writes
- `fsync()` at end

BetrFS random writes benefit from B^{ε}-tree insertion performance

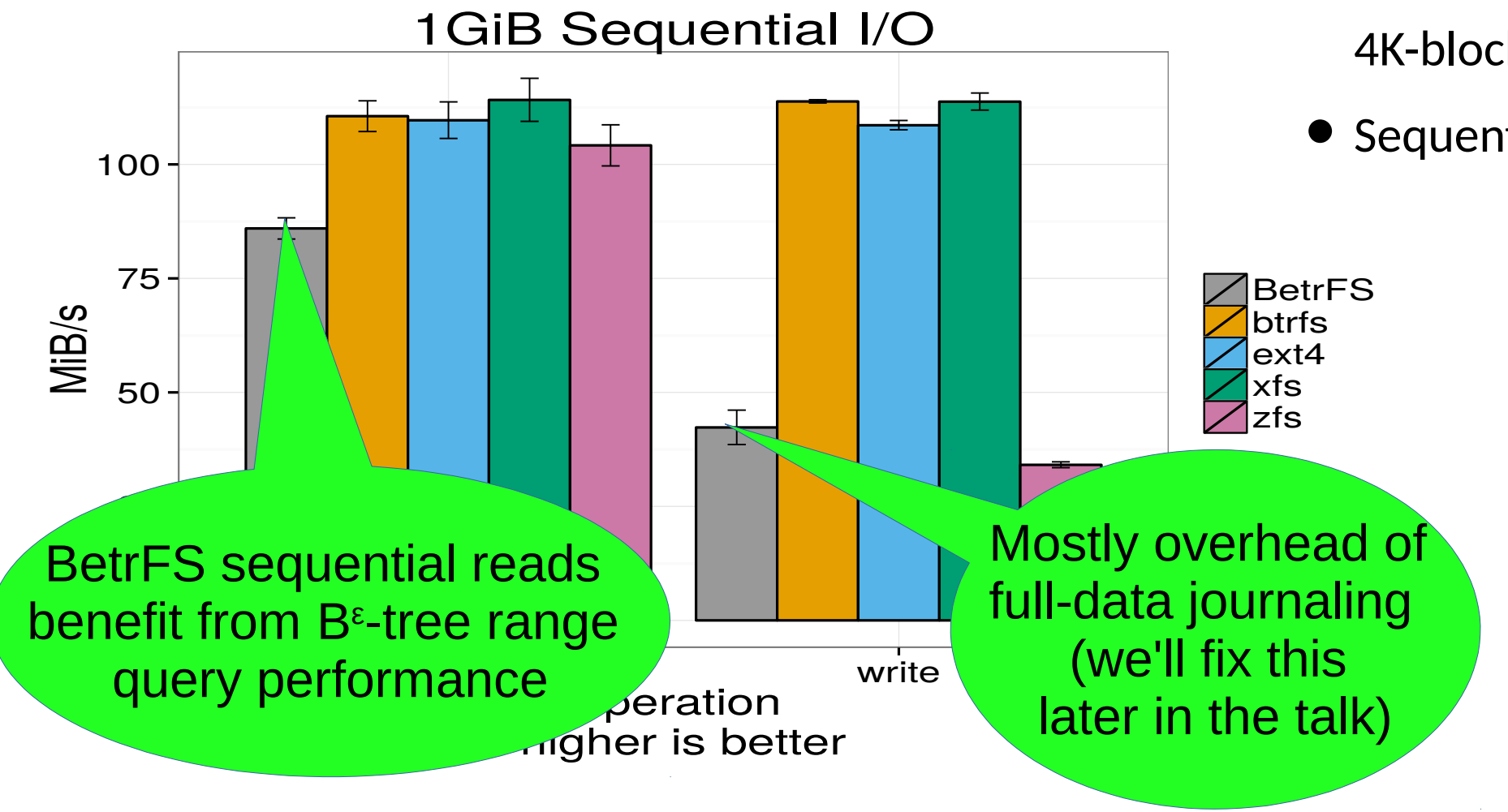
Small file creates are an order-of-magnitude faster



- Create 3 million files and write 200-bytes to each
- Balanced directory tree with fanout 128

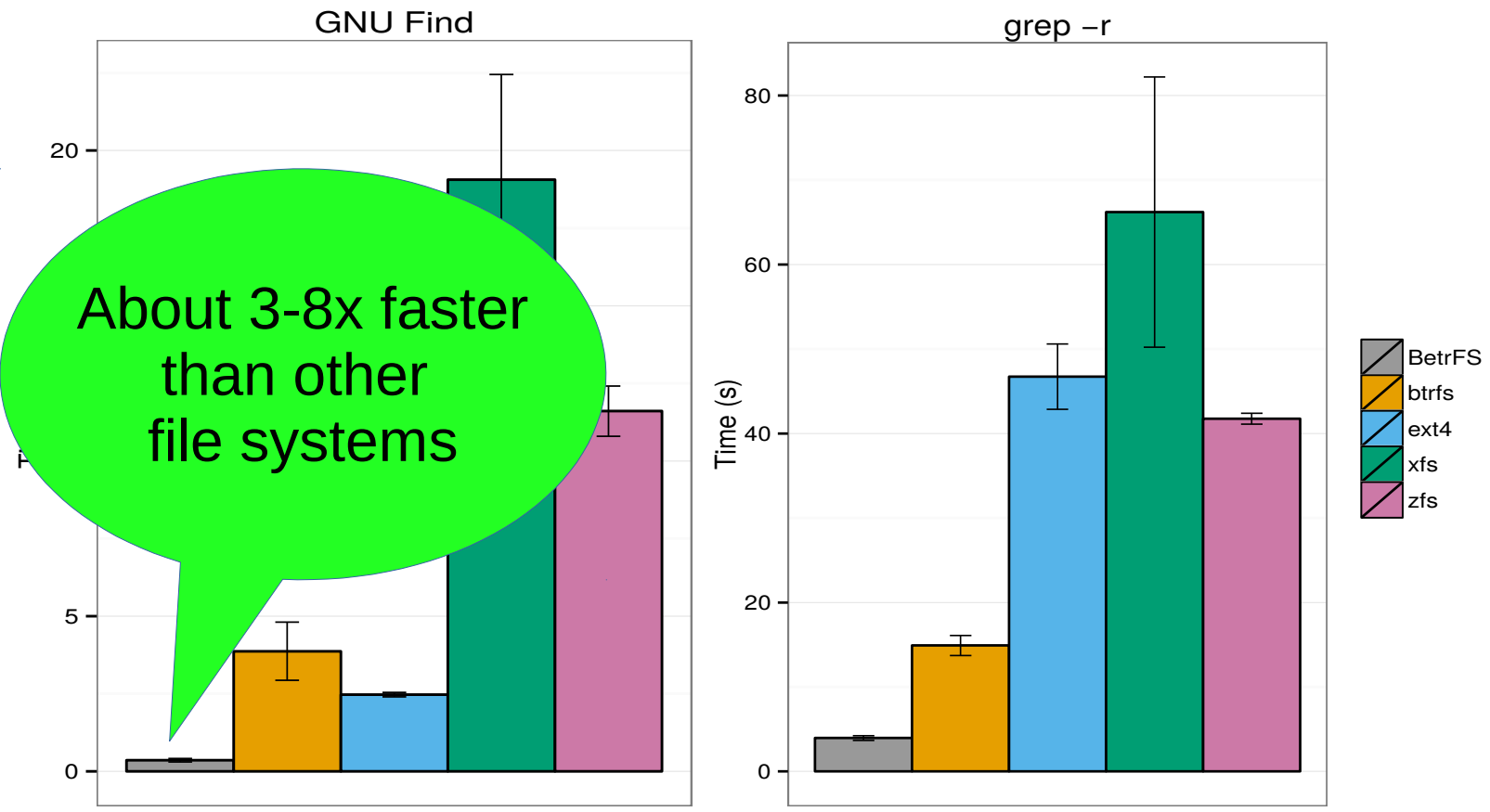
BetrFS file creates benefit from B^{ε}-tree insertion performance

Sequential I/O



- Write random data to file, 10 4K-blocks at a time
- Sequentially read data back

Recursive directory traversals

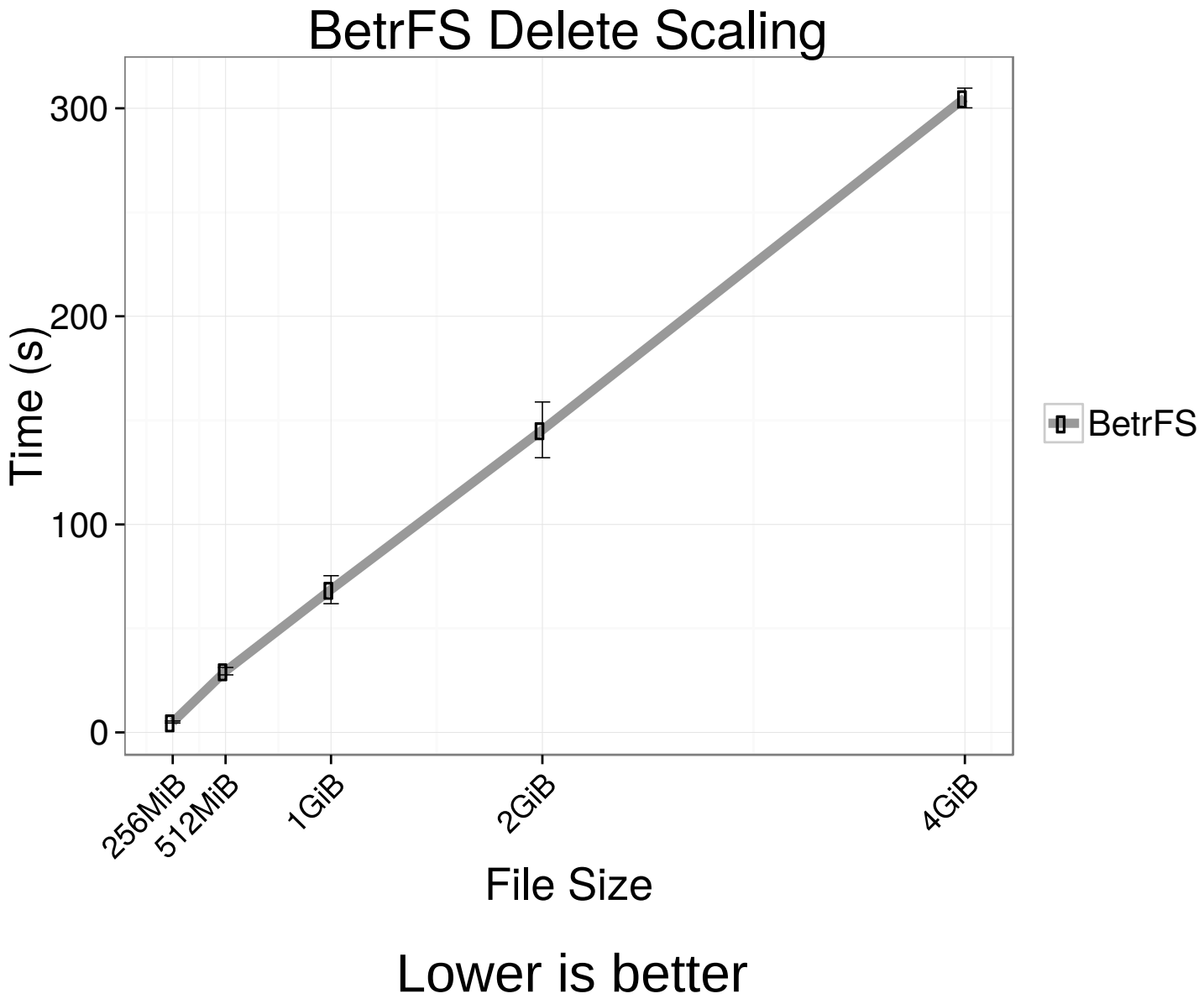


Lower is better

- Recursive scans from root of Linux 3.11.10 source
- GNU `find` scans file metadata
- `grep -r` scans file contents

BetrFS directory traversals benefit from B^{ε}-tree range-query performance

File deletion

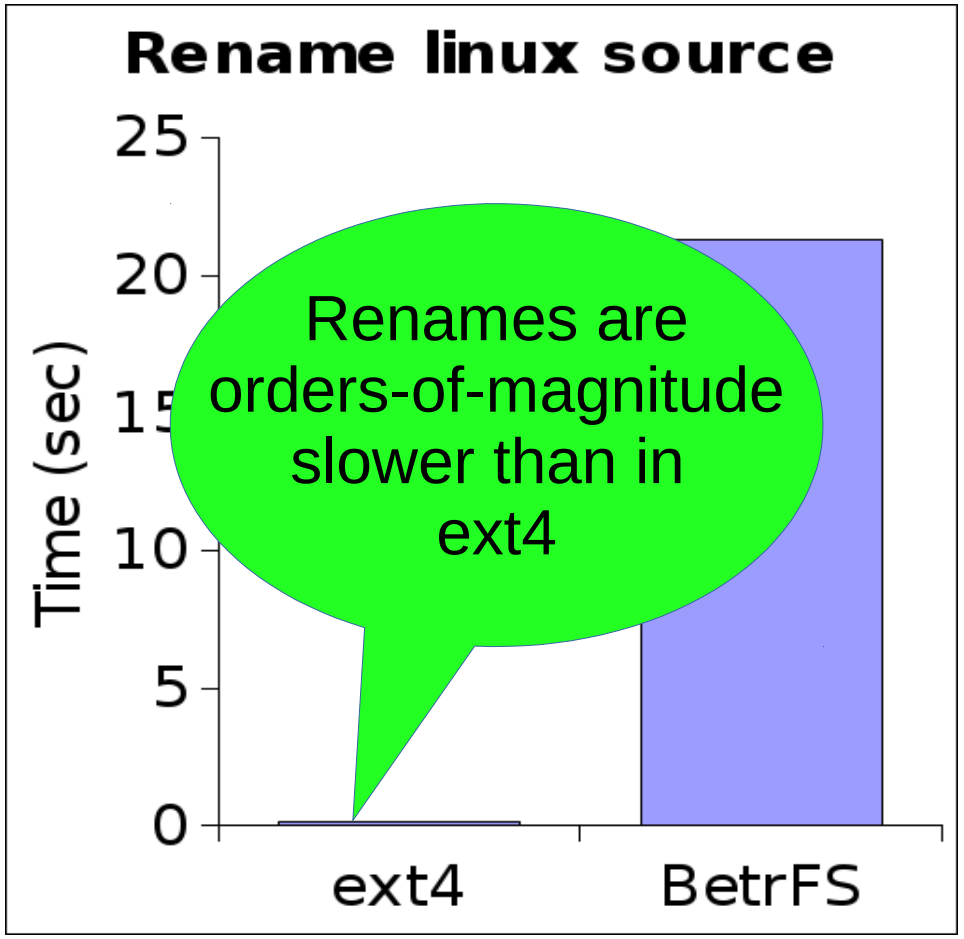


Lower is better

- Write random data to file, `fsync()` it
- Delete file

BetrFS deletes require O(n) key-value operations

Directory rename



Lower is better

BetrFS renames require O(n) key-value operations

BetrFS (version 0.1) performance summary

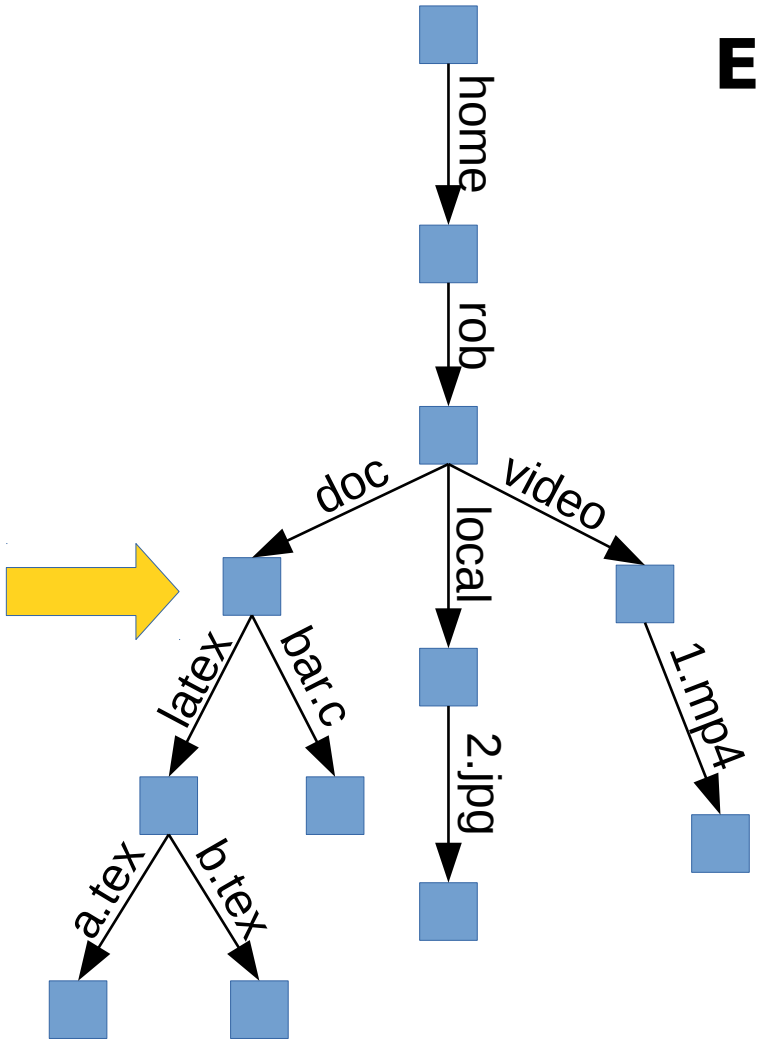
- ✓ Sequential reads
- ✗ Sequential writes
- ✓ Random writes
- ✗ File/directory renames
- ✗ File deletes
- ✓ Recursive scans
- ✓ Metadata updates

Let's fix these problems

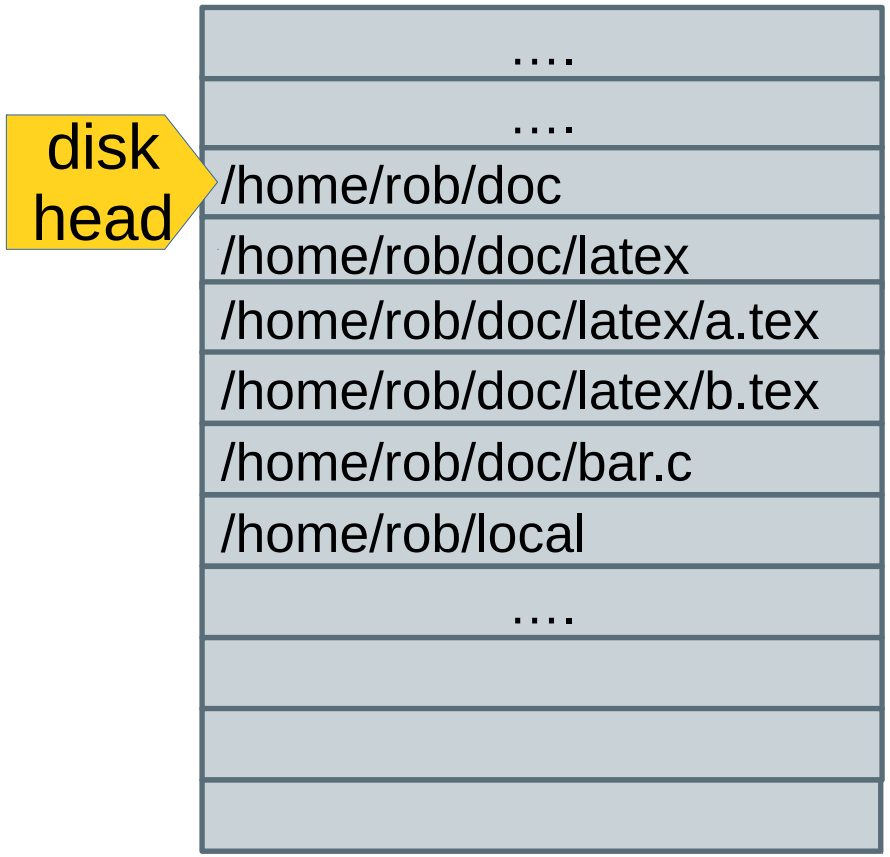
Accelerating rename without slowing down directory traversals

Full-path indexing yields fast directory scans

Example: grep -r “key” /home/rob/doc/



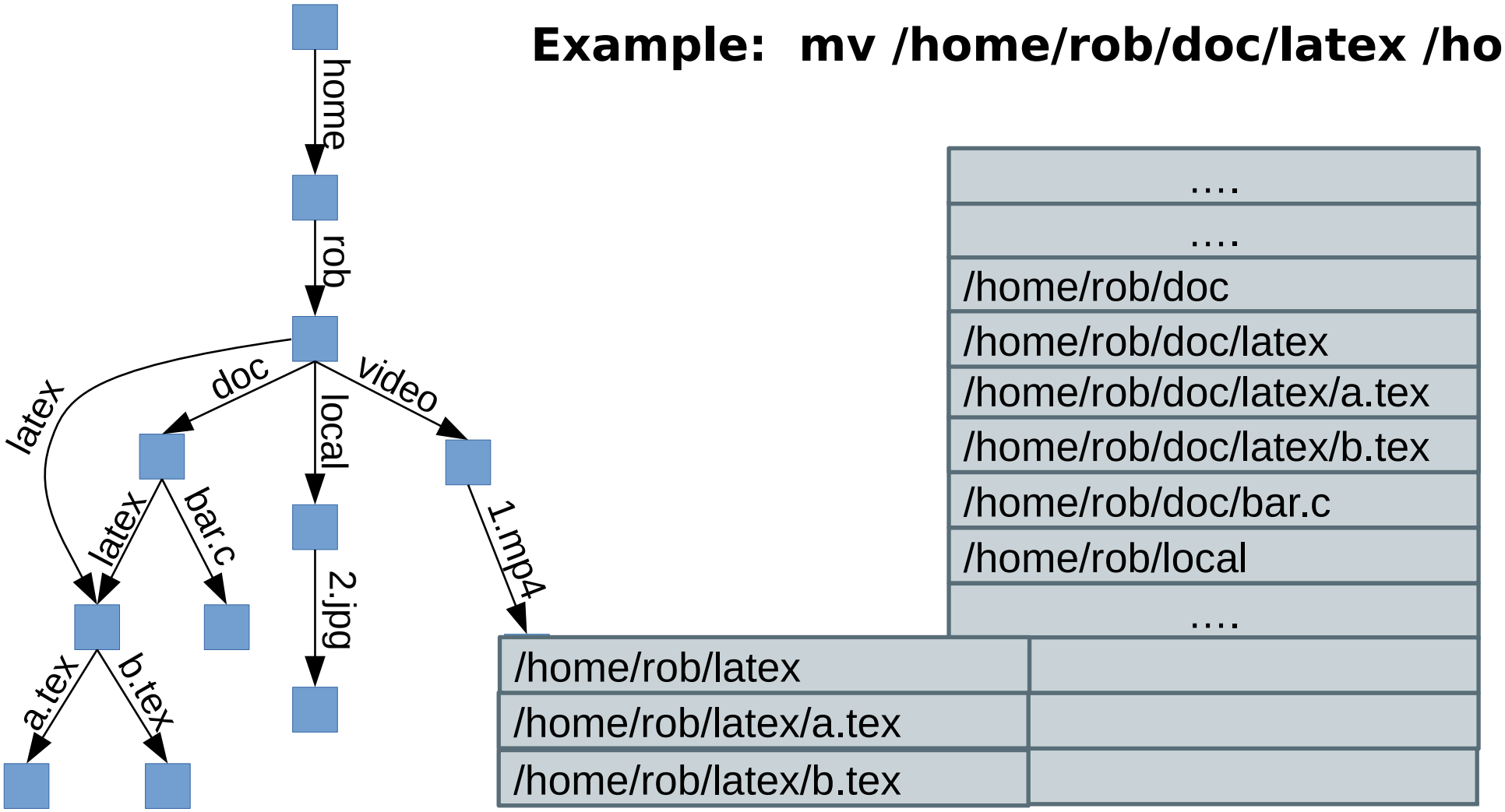
Directory Tree (logical)



Disk (physical)

Rename is expensive when using full-path indexing

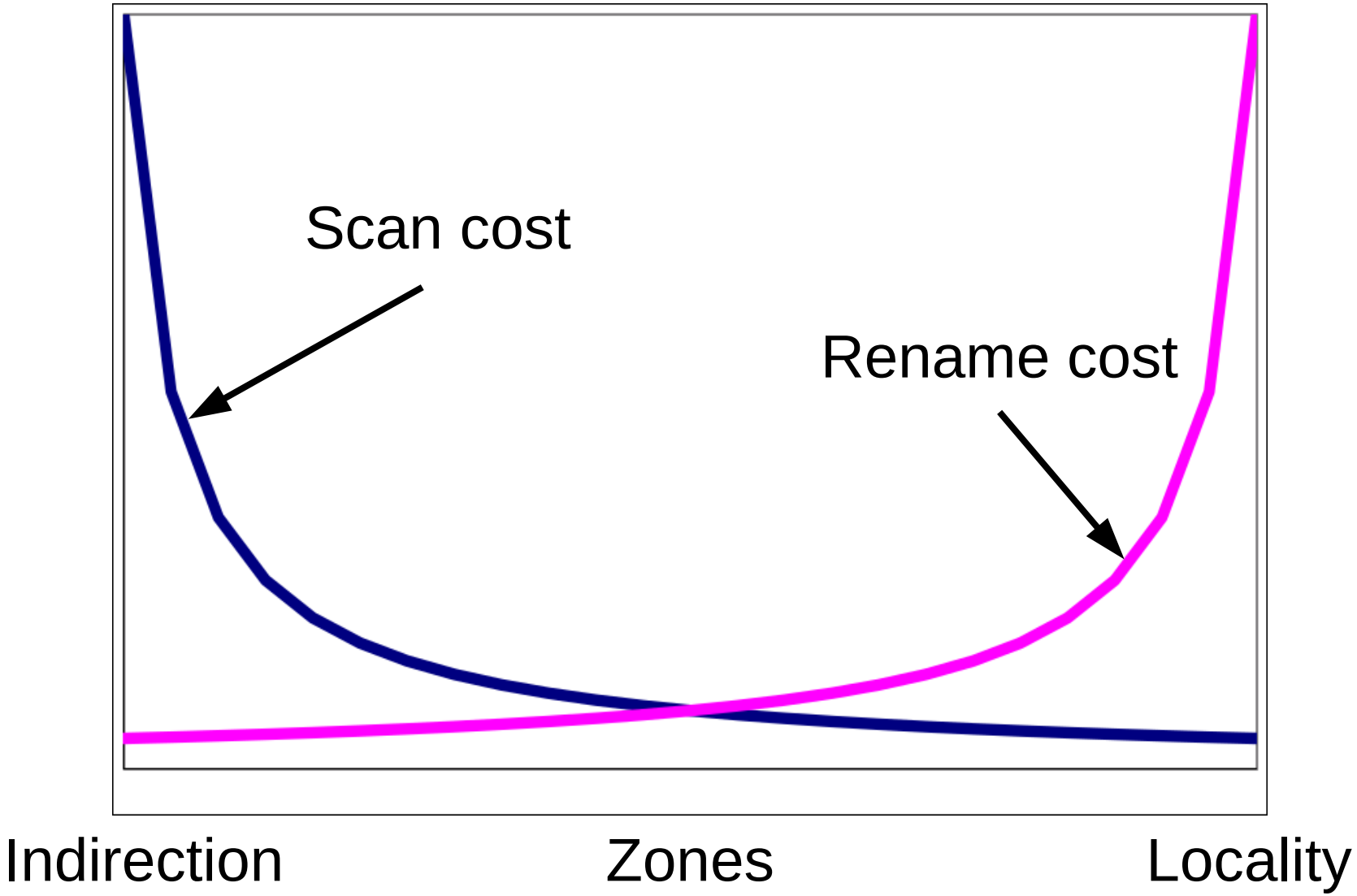
Example: mv /home/rob/doc/latex /home/rob



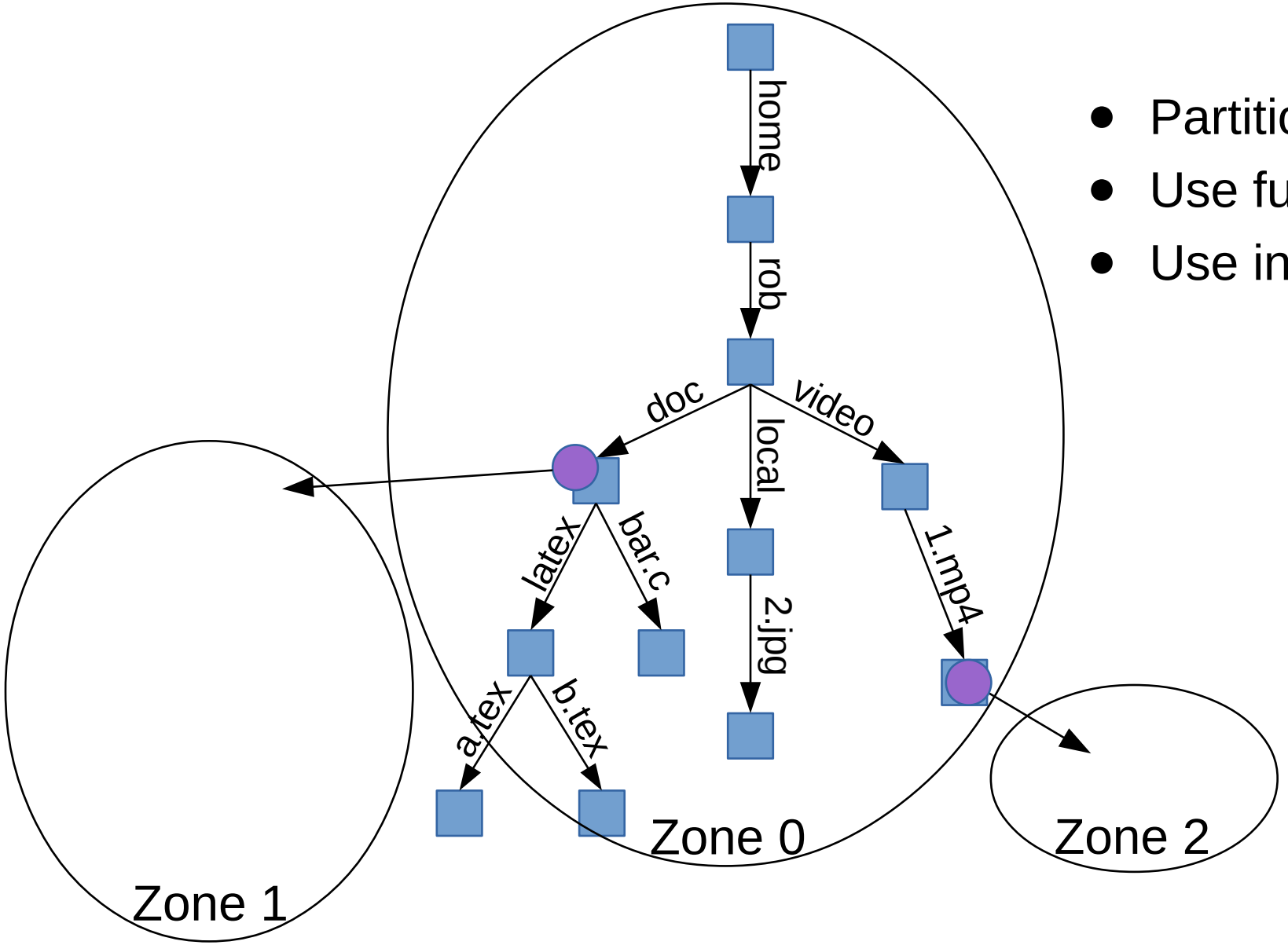
Directory Tree (logical)

Disk (physical)

Zoning: balancing indirection and locality



Zone: a subtree of the directory hierarchy

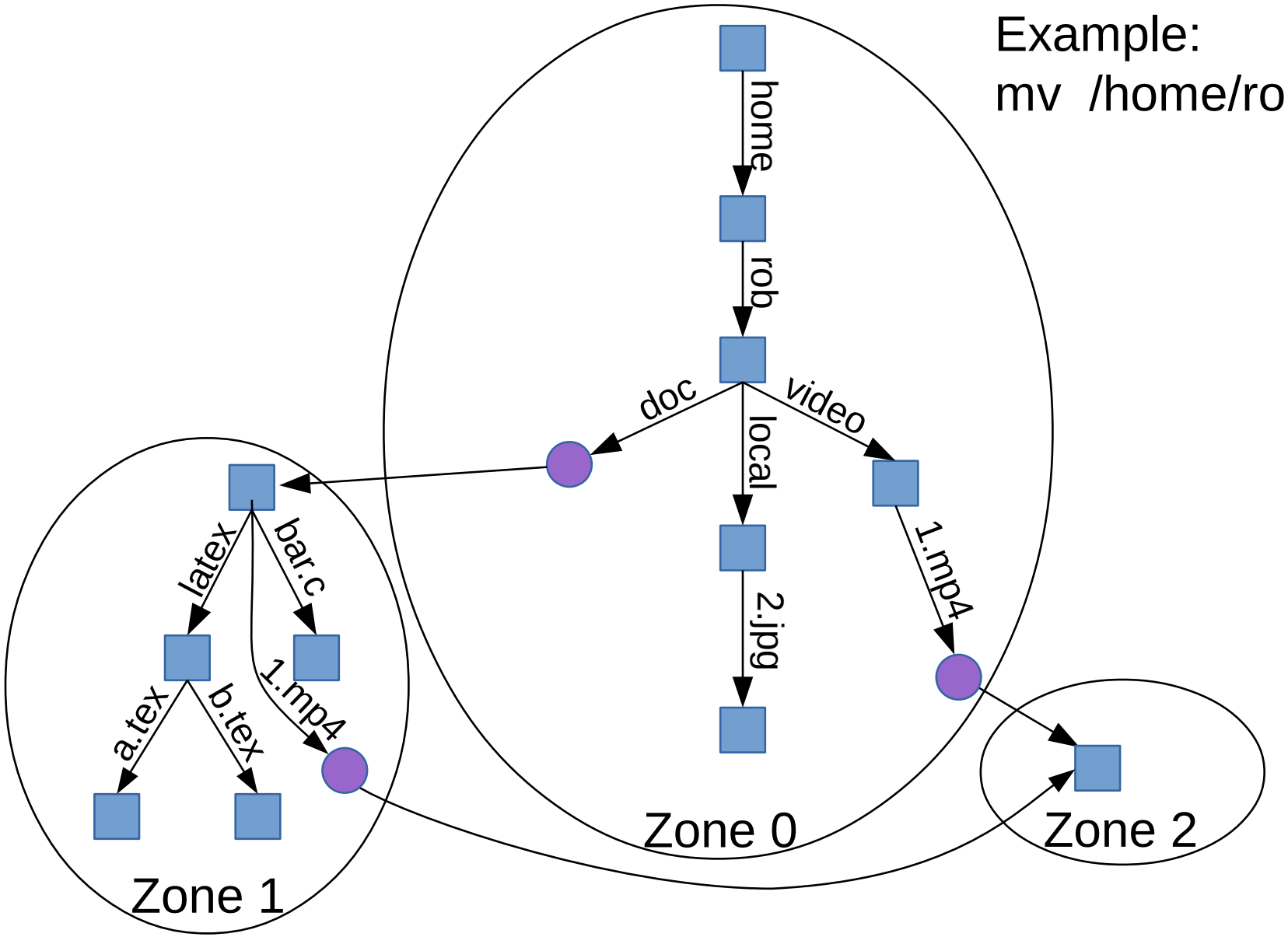


- Partition file system into zones
- Use full-path indexing within zones
- Use inodes between zones

Implication: Recursive directory scans only perform seeks when crossing zones

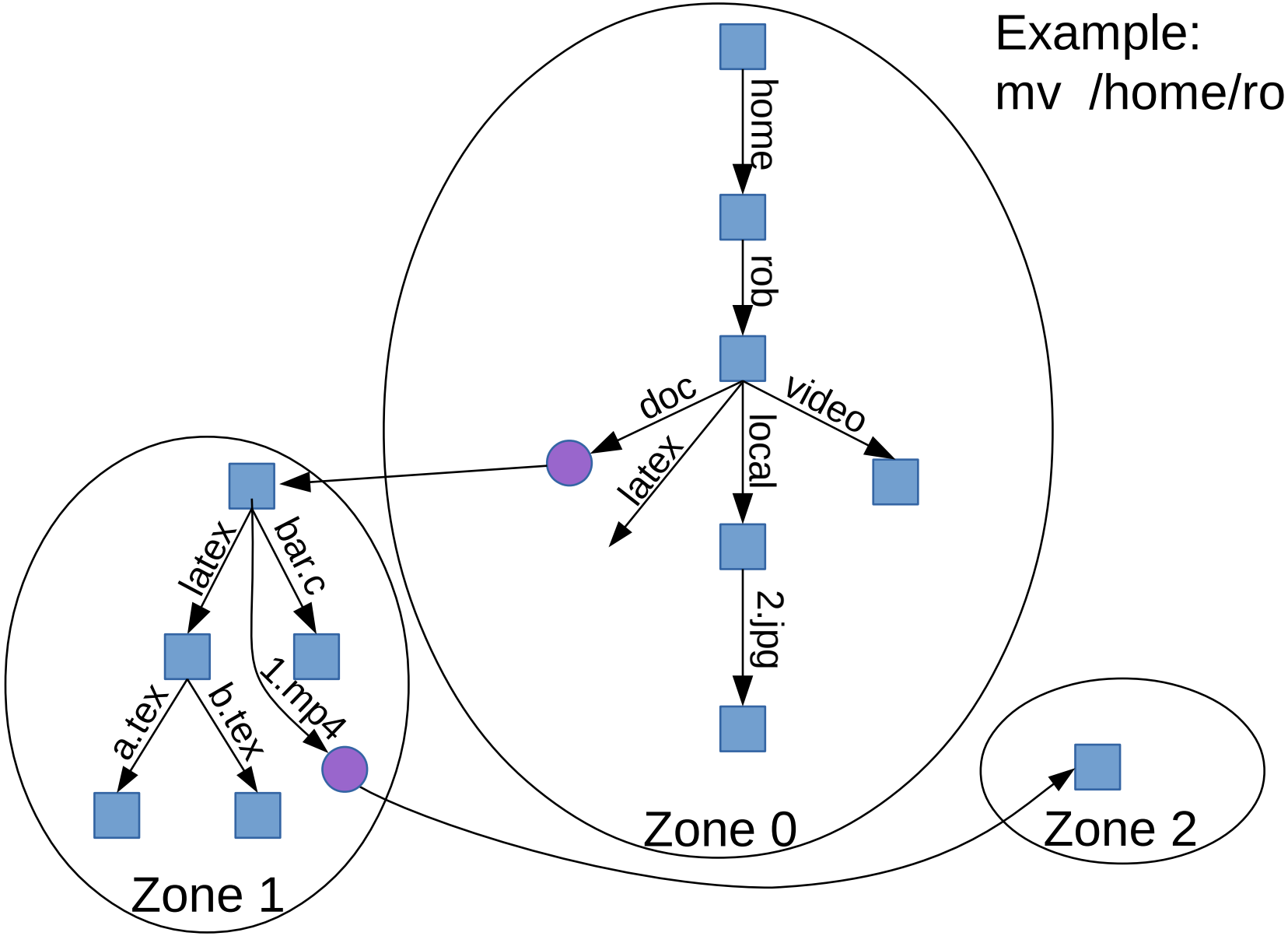
Moving the root of a zone is cheap

Example:
mv /home/rob/video/1.mp4 /home/rob/doc

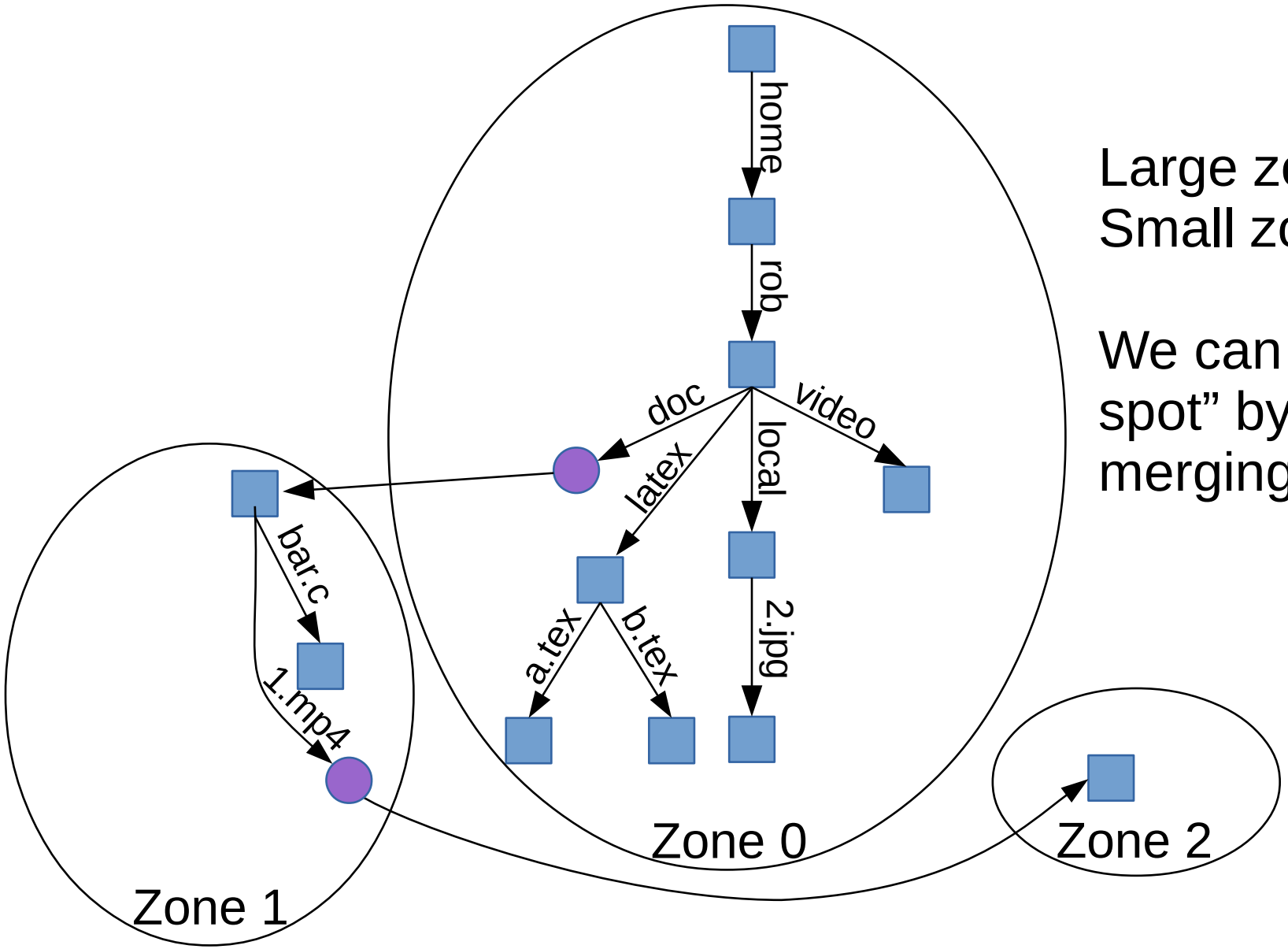


Renaming a subtree of a zone requires copying

Example:
mv /home/rob/doc/latex /home/rob/latex



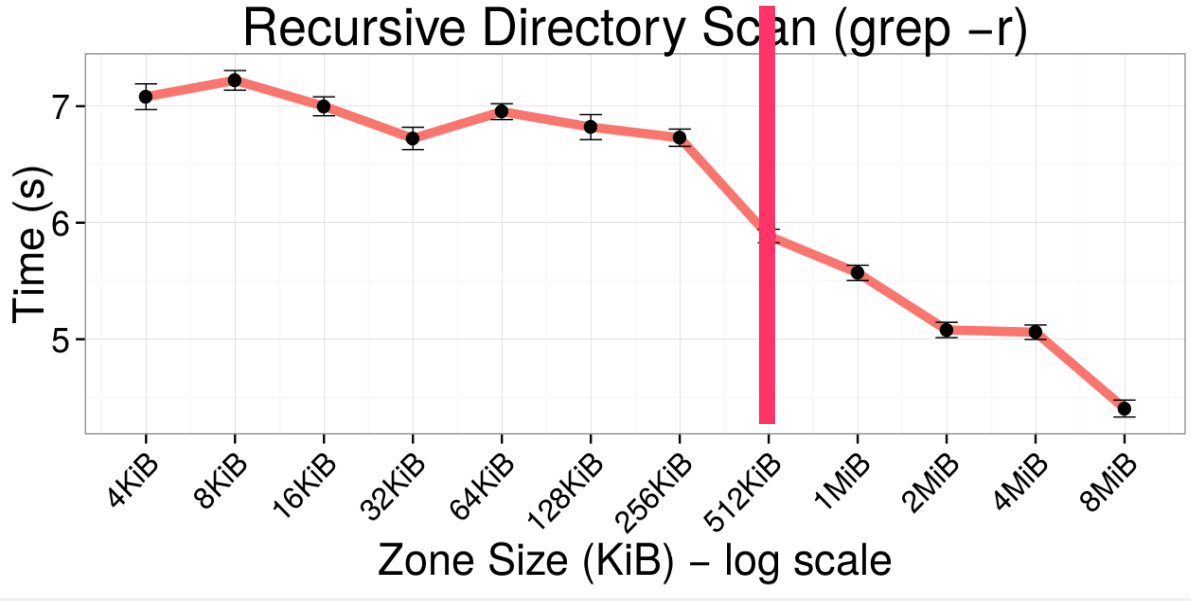
Managing zone sizes

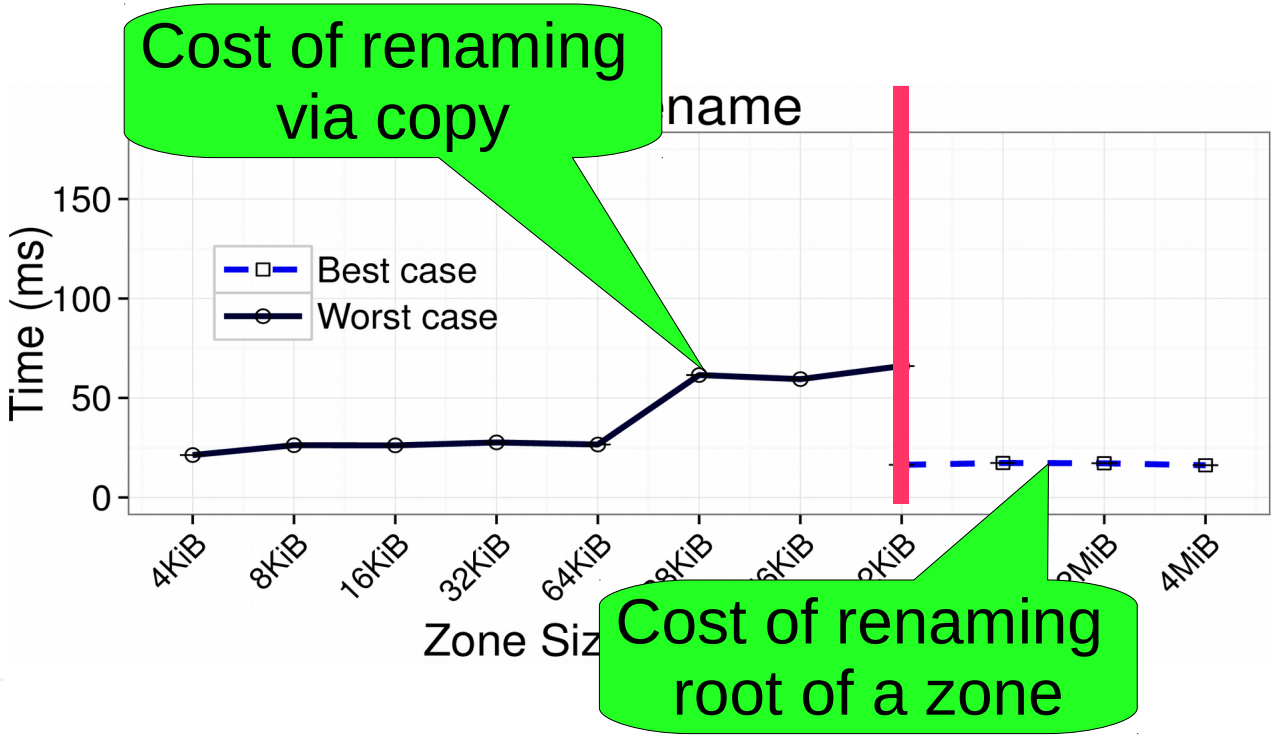


Large zones → fast directory scans
Small zones → fast renames

We can keep zone sizes in a “sweet spot” by splitting large zones and merging small zones

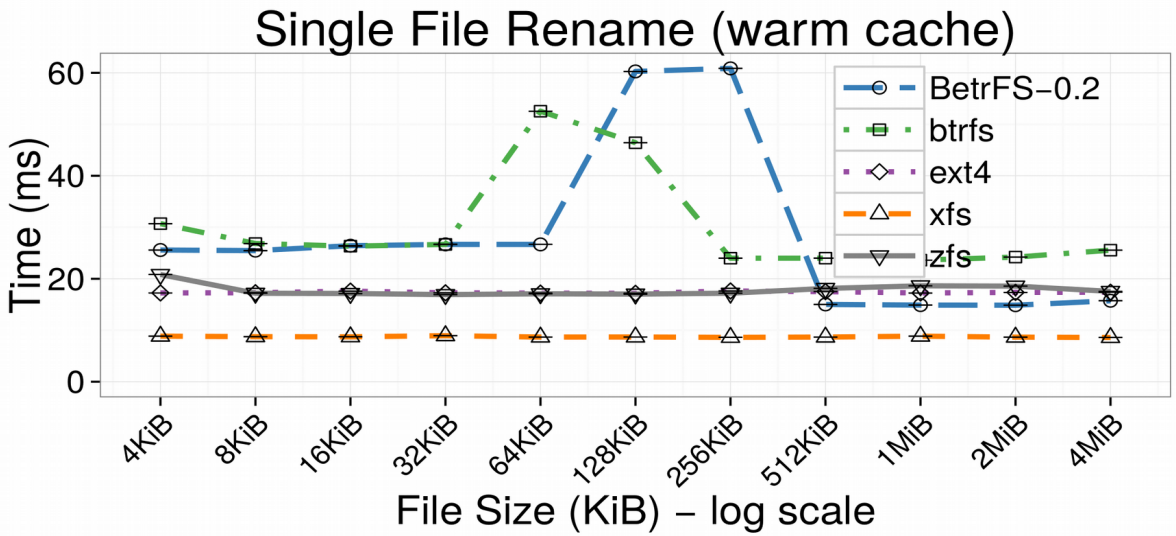
How big should zones be?



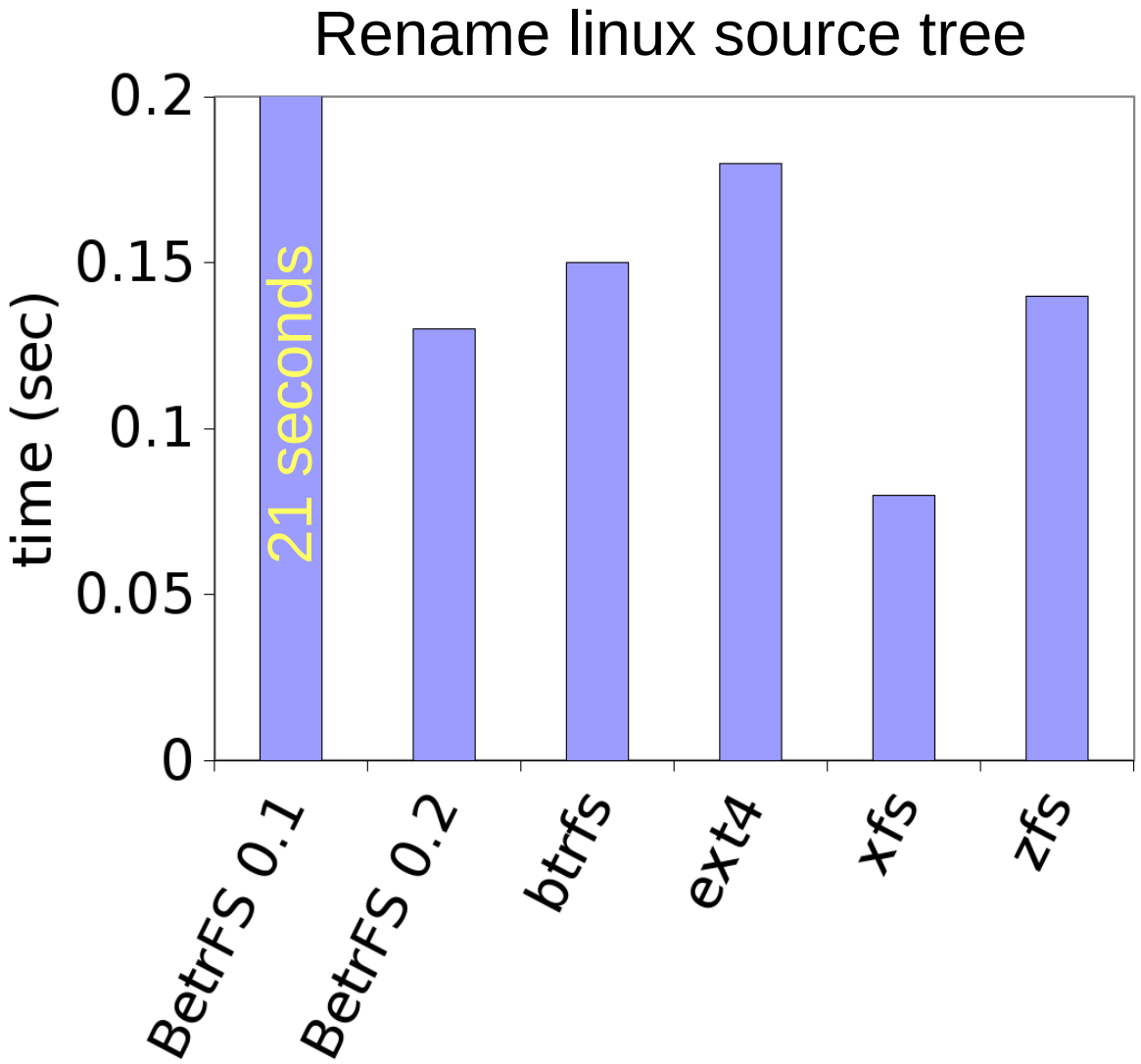


BetrFS-0.2 uses 512KB zones to balance rename and scan performance

BetrFS 0.2 rename performance

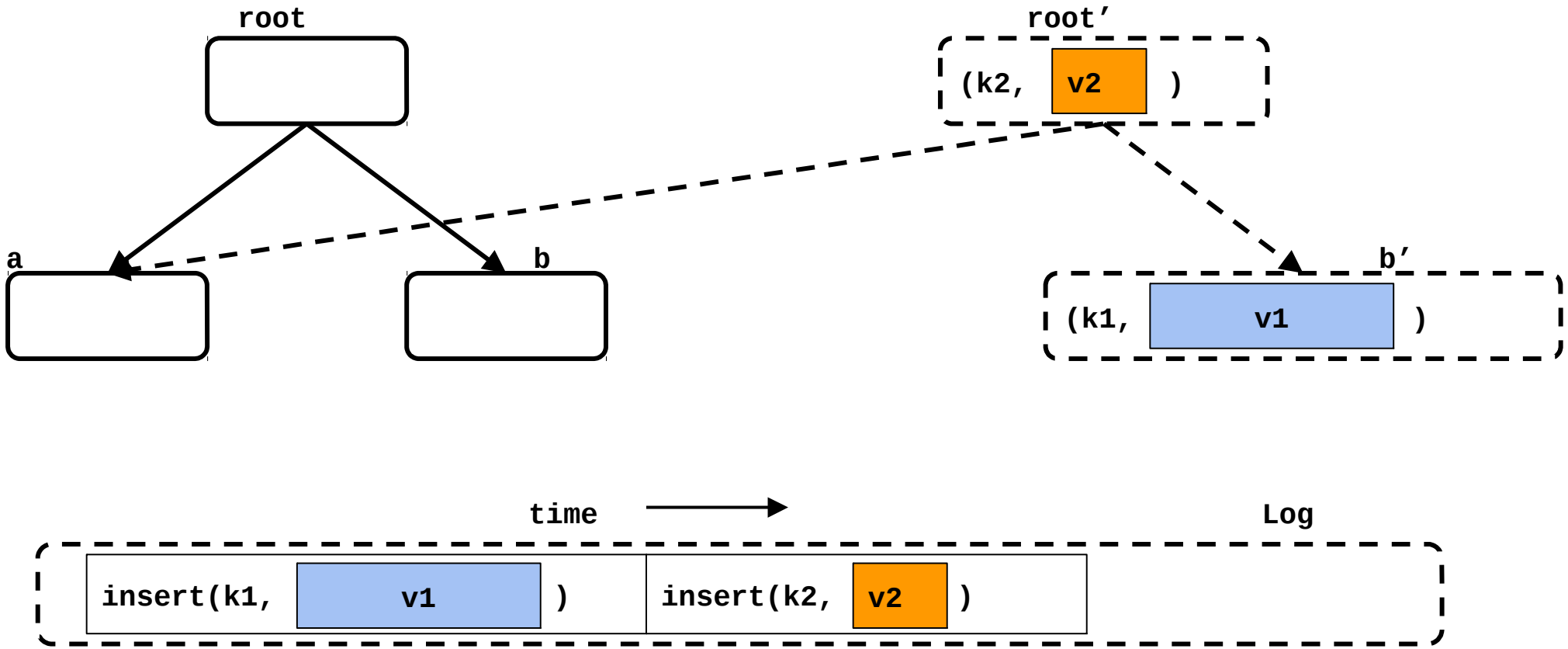


Rename performance is comparable to other file systems

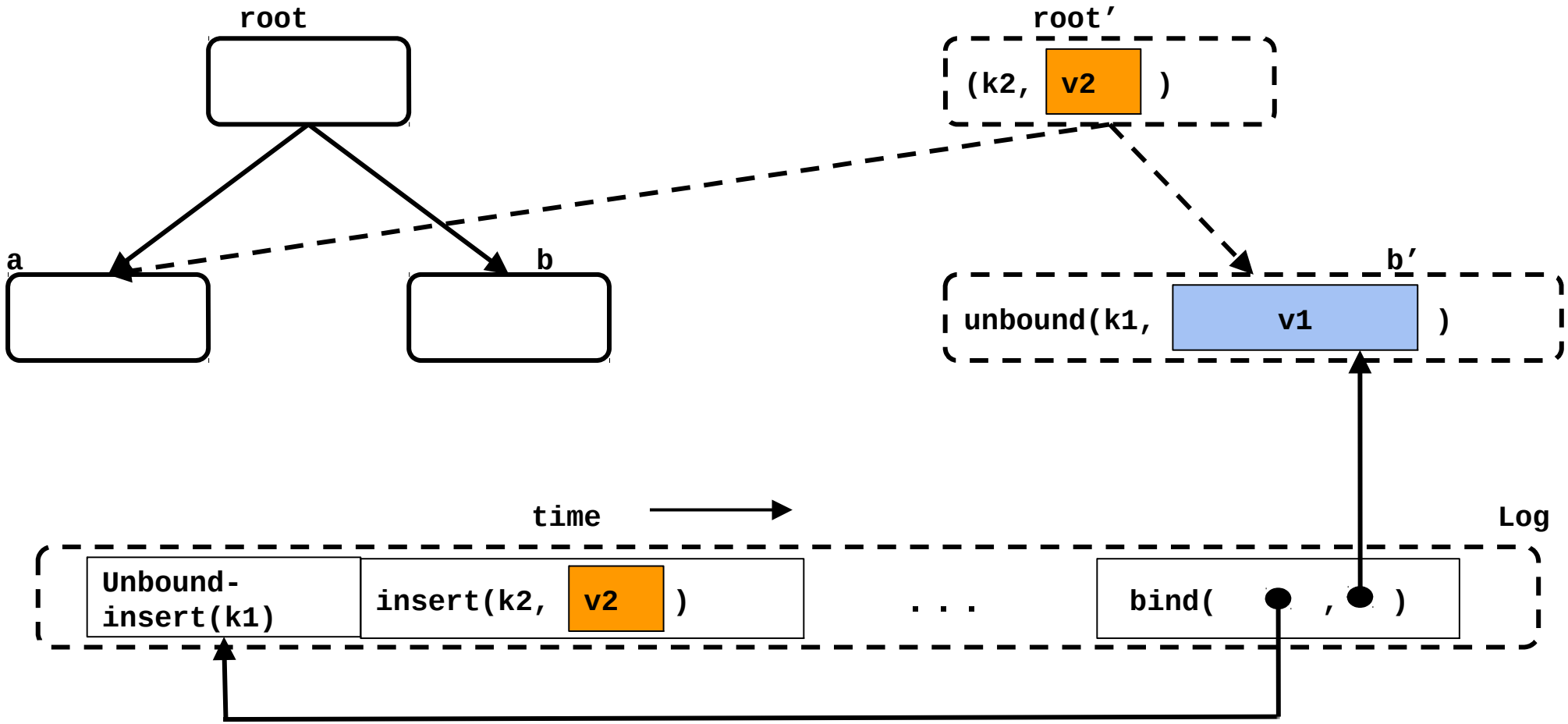


Performing sequential writes at disk bandwidth and with full data journaling semantics

BetrFS version 0.1 writes everything twice



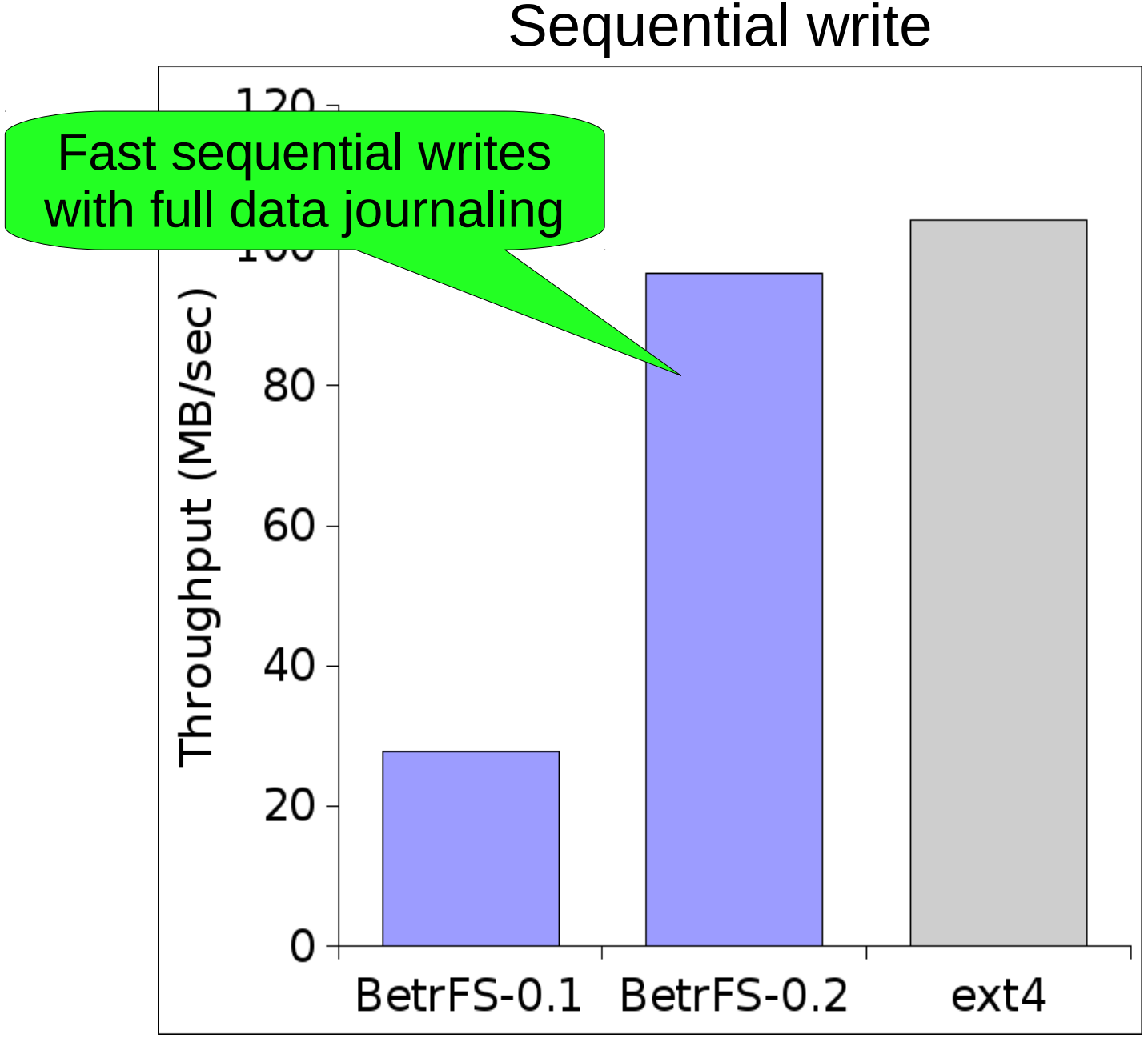
BetrFS version 0.2: late-binding journal



Why don't we use late-binding for small writes?

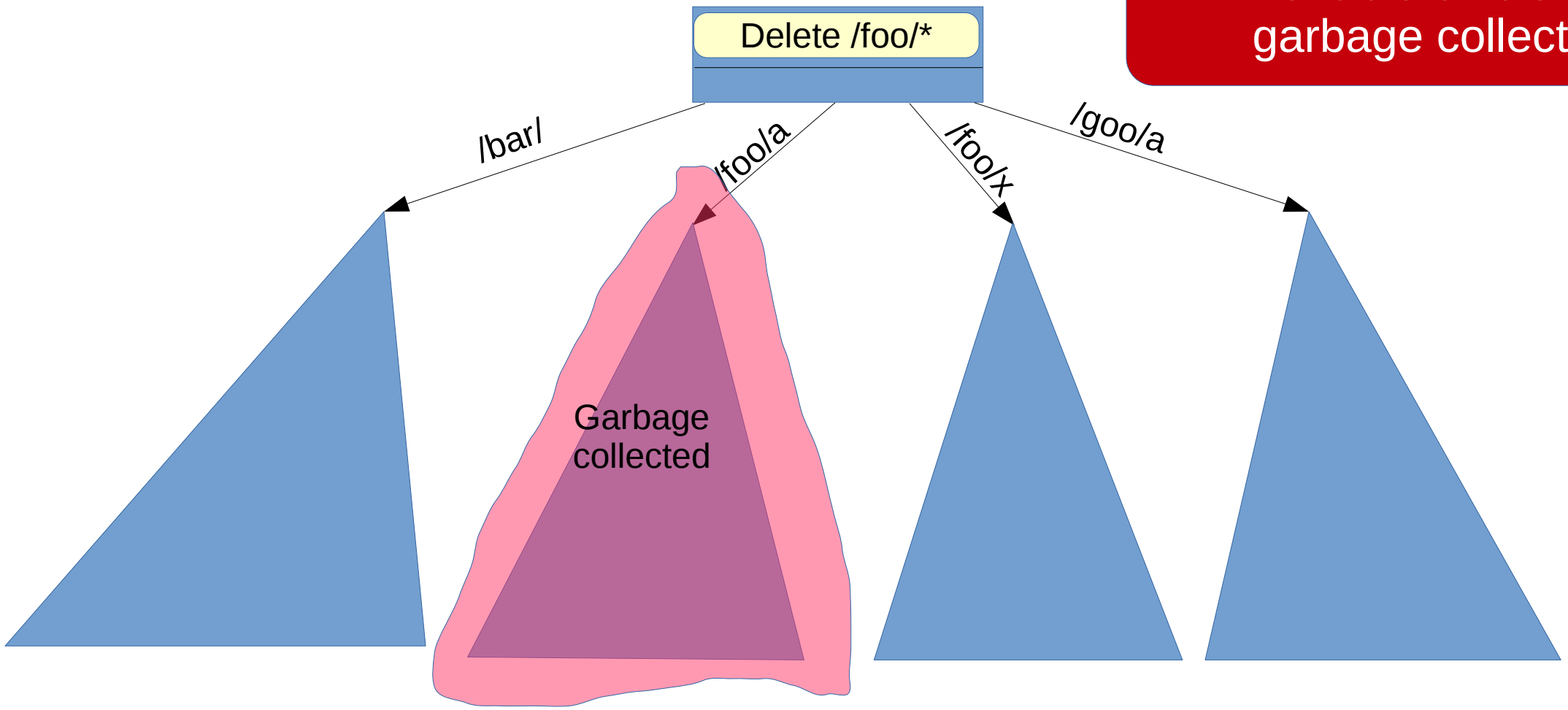
- Reason 1:
  - Late-binding requires writing out a large (e.g. 4MB) node
  - For small writes, this is huge write-amplification
  - It's more efficient to make small writes durable by simply logging them
- Reason 2:
  - Small, random writes get written to disk several times as they get flushed down the B^{ε}-tree
  - So writing them to the log is not a big extra cost

Late-binding journal: performance evaluation

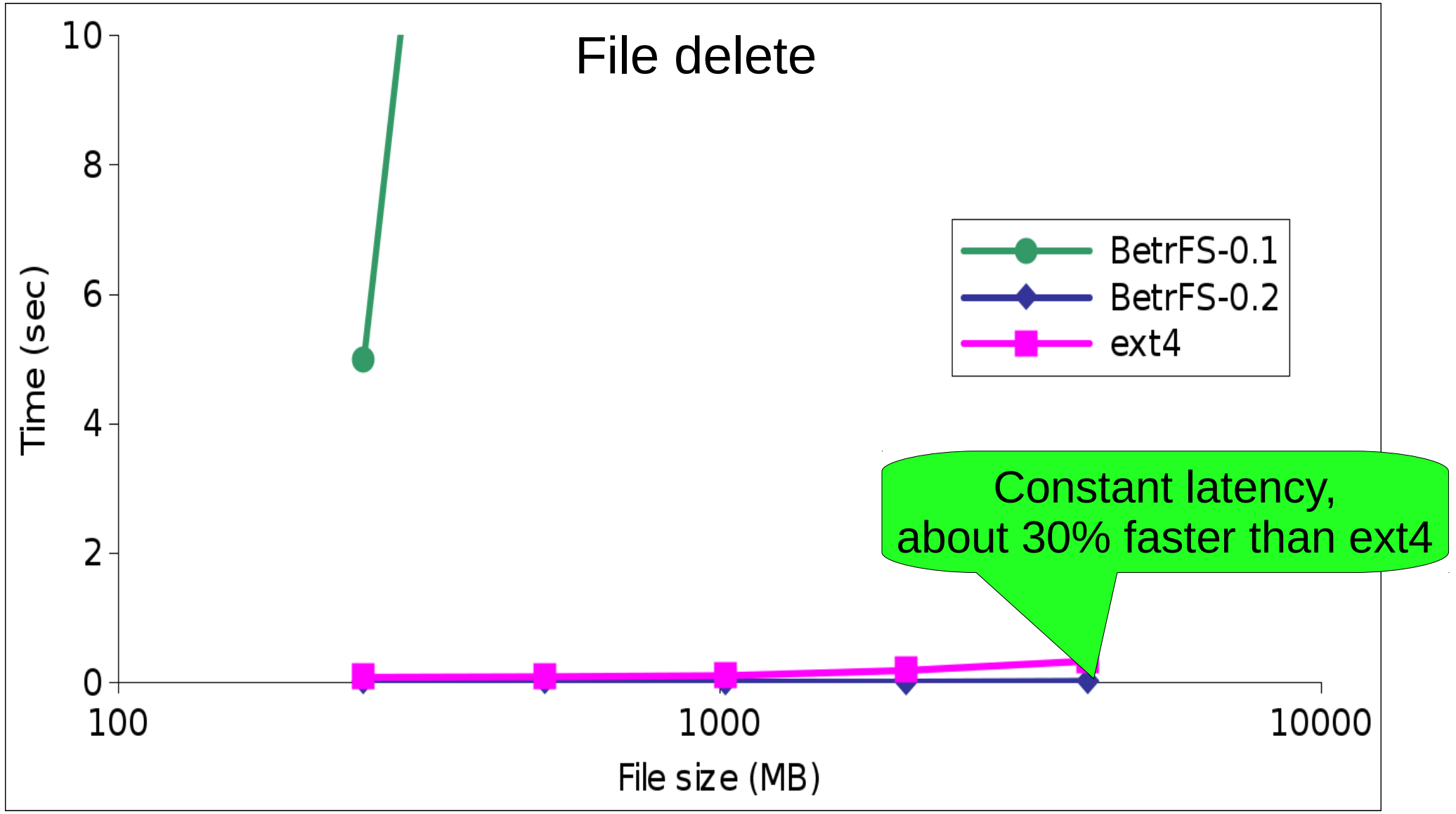


Rangecast delete

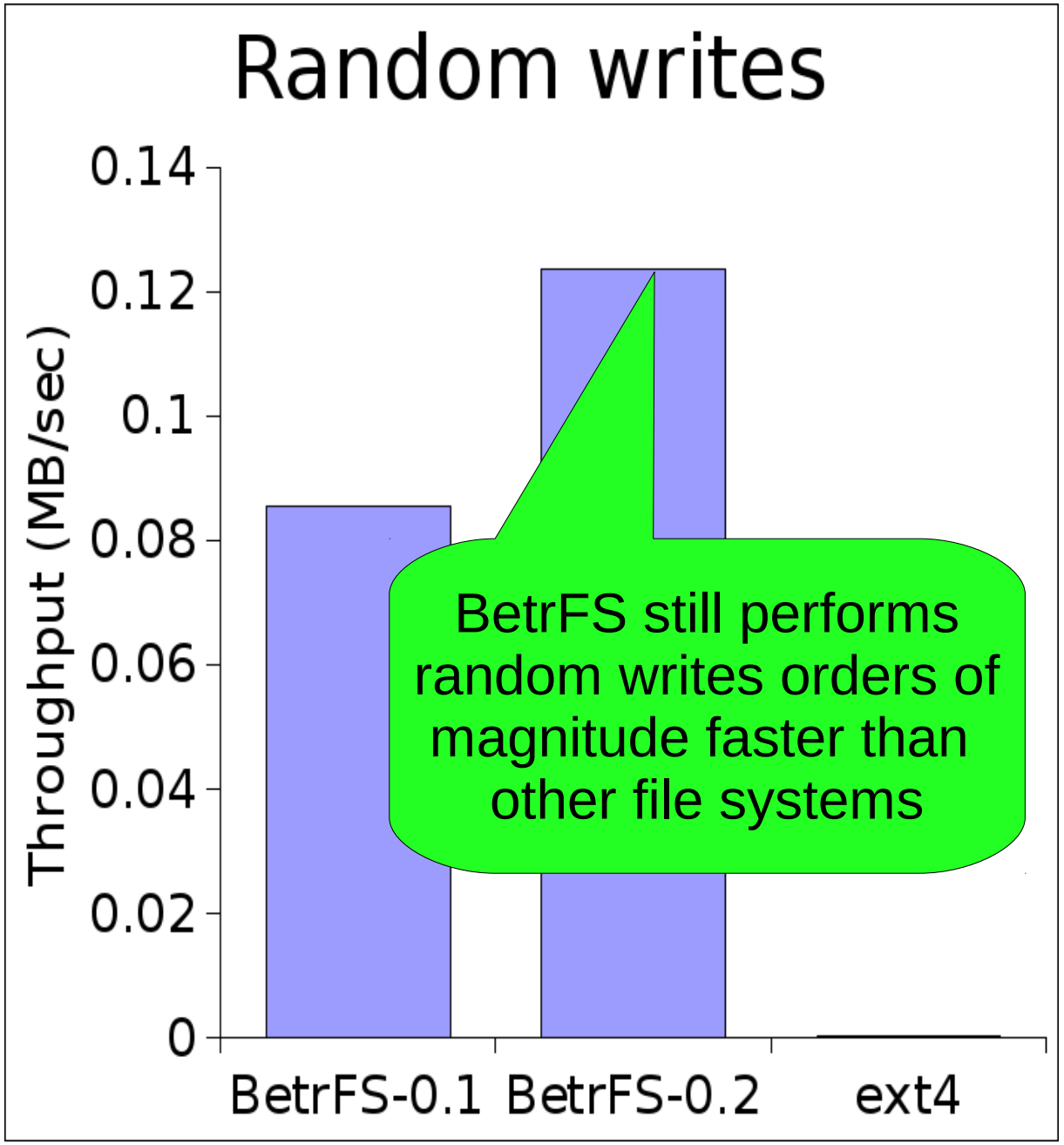
File deletions require inserting a single message and enable efficient garbage collection

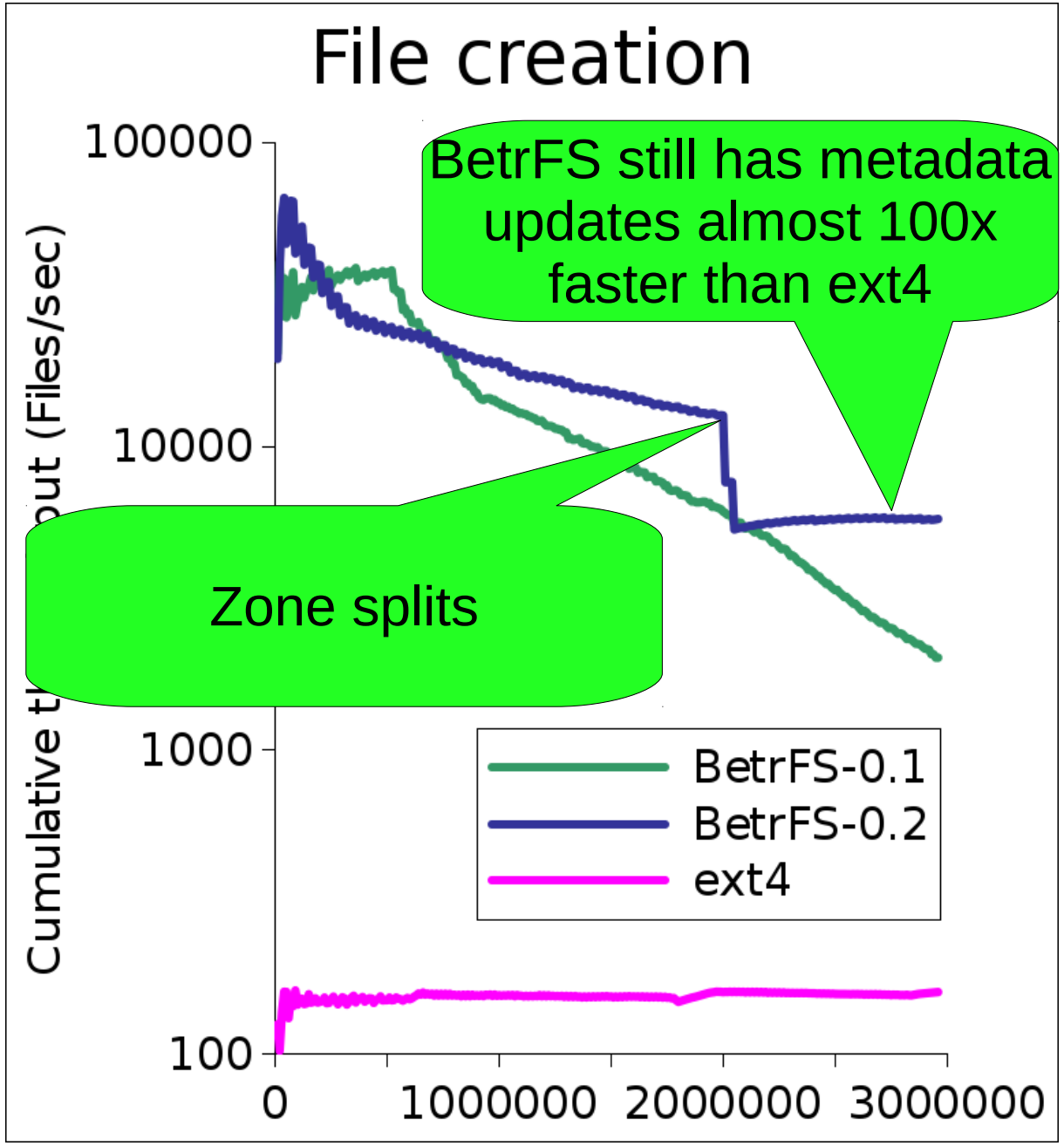


Rangecast delete performance evaluation



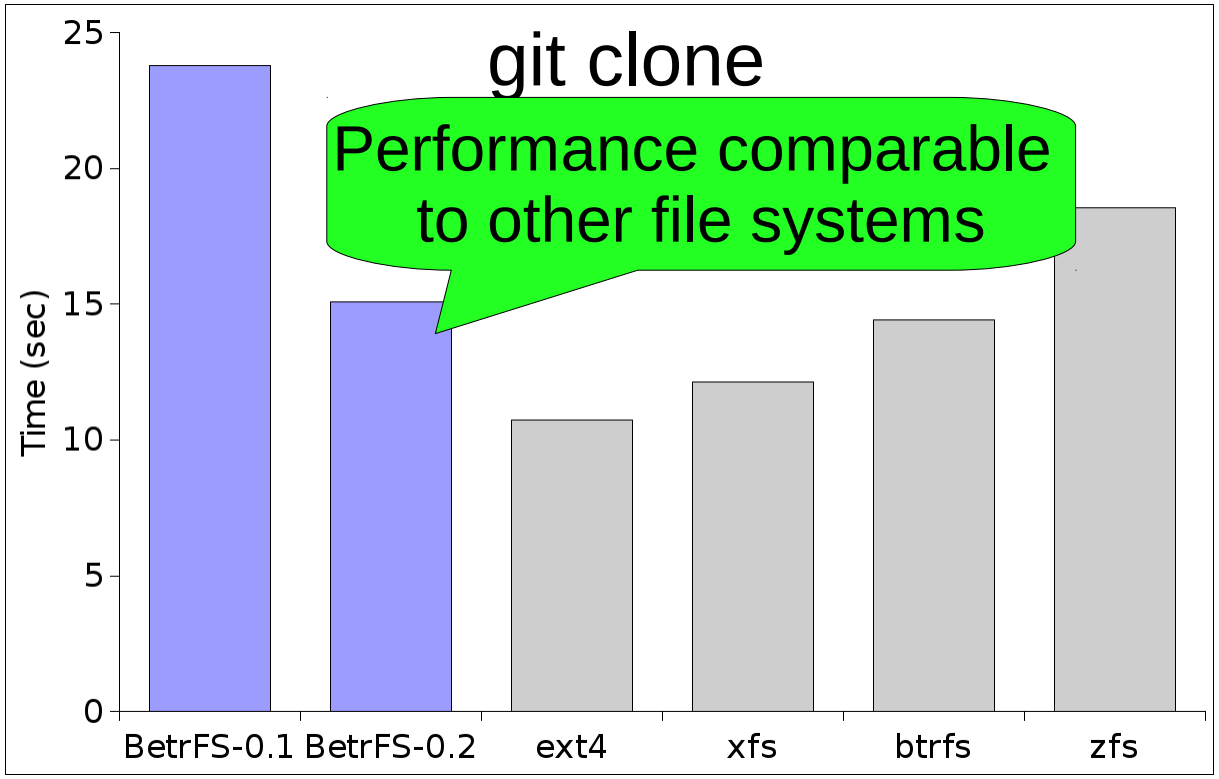
Is BetrFS still fast at other operations?

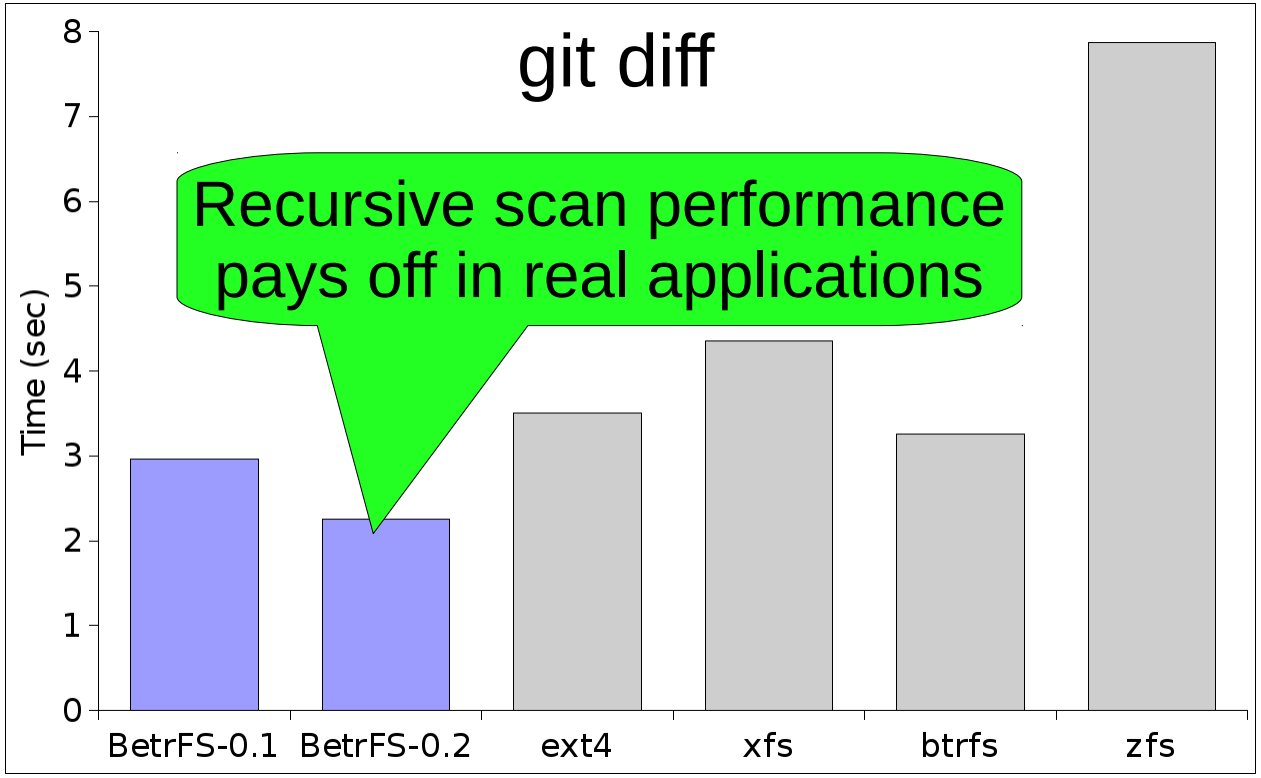

Random writes
Throughput (MB/sec)
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
BetrFS-0.1
BetrFS-0.2
ext4
BetrFS still performs random writes orders of magnitude faster than other file systems


File creation
Cumulative t
100000
10000
1000
100
0
1000000
2000000
3000000
BetrFS still has metadata updates almost 100x faster than ext4
Zone splits
BetrFS-0.1
BetrFS-0.2
ext4

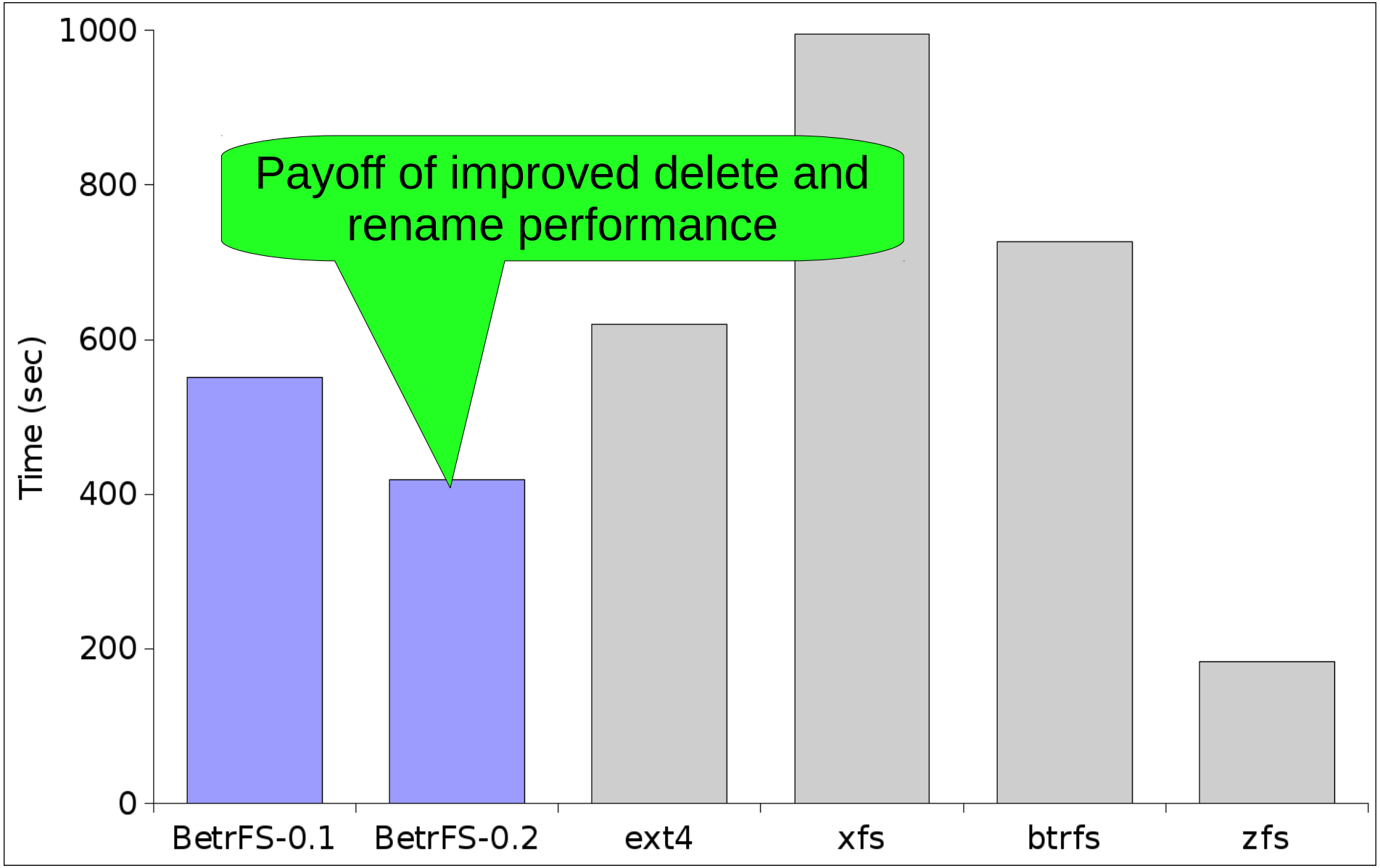
What about real application performance?

Macrobenchmark: git





Macrobenchmark: dovecot imap maildir workload



BetrFS (version 0.2) performance summary

- Sequential reads
- Sequential writes
- Random writes
- File/directory renames
- File deletes
- Recursive scans
- Metadata updates

Conclusion

- Write-optimized data structures can enable us to overcome long-standing file-system design trade-offs
- Write-optimized file systems can offer across-the-board top-of-the-line performance
- Write-optimization creates a need/opportunity to revisit many file system design issues
- **Code available at**

betrfs.org

SSD performance preview