Write-Optimization for File-System Metadata

Rob Johnson
VMware Research

ExxonMobil

"A user does ls -l on a directory with a million files"

Aaron Steichen

"Want fast find on trillions of files"

Chris Beecroft

"Flattening the namespace gives great performance but renames are expensive"

Dave Bonnie

File systems need to perform well on many metadata operations

	Query	Upda
Point	• `stat`	• atime upd • `creat` • `unlink`
Recursive	• `find` • `grep`	• `rm -rf` • recursive chmod

Write optimization

Full-path indexing

Custom data structures

`rename`

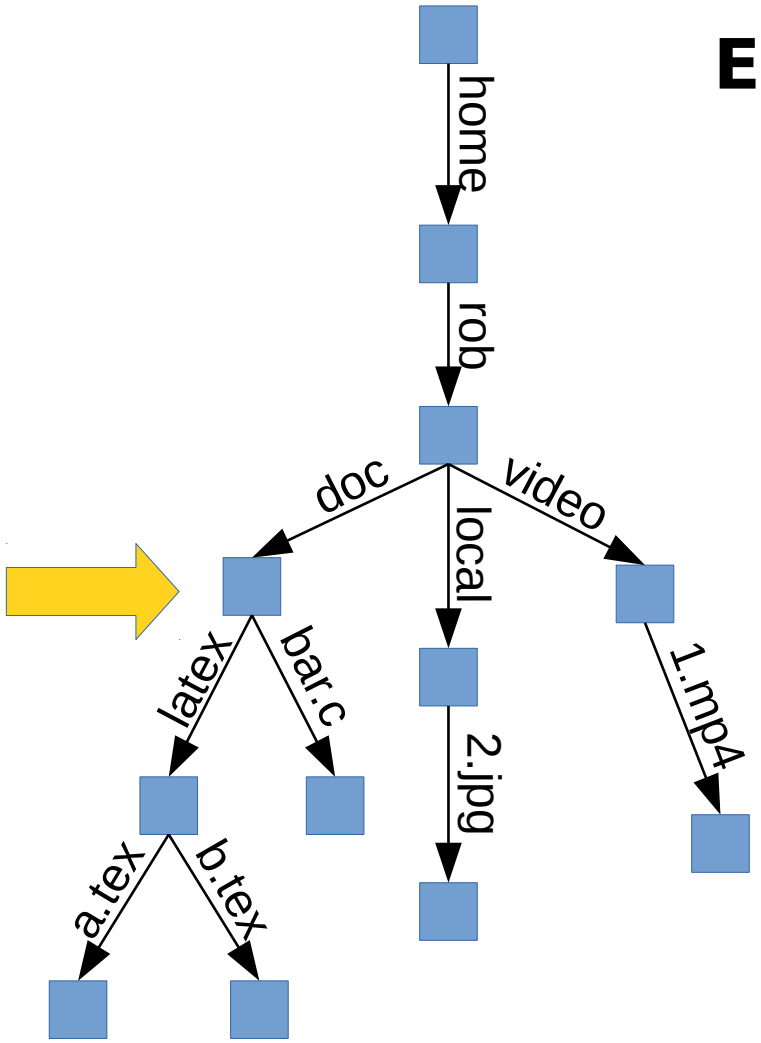
Full-path indexing

Full-path indexing in BetrFS

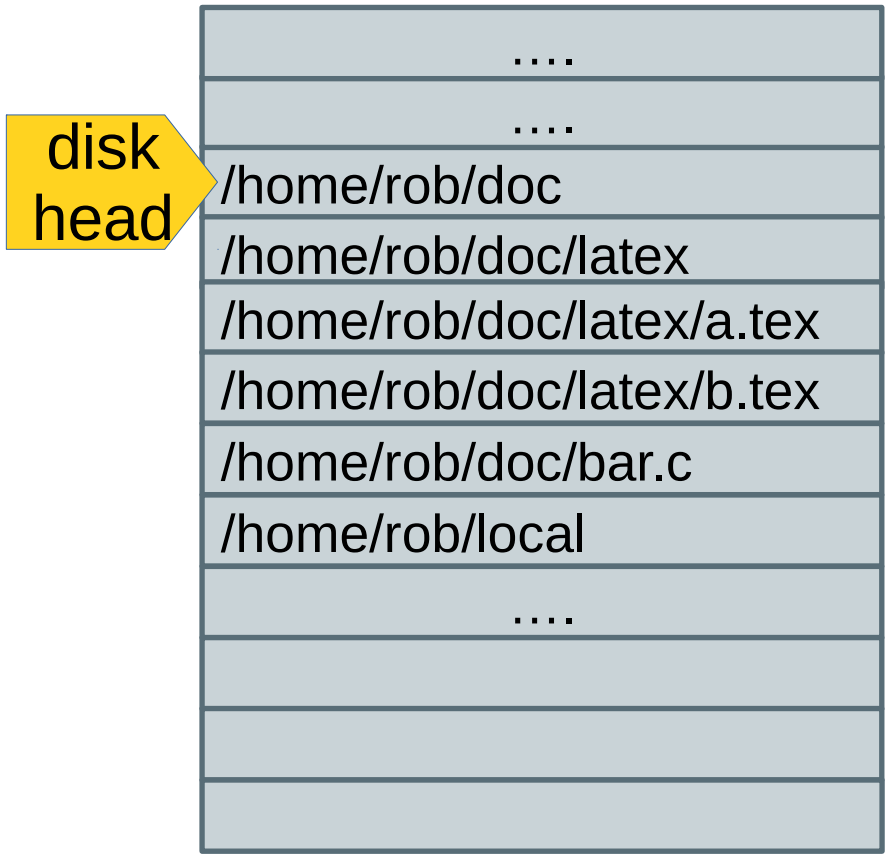
- BetrFS maintains two indexes
  - Metadata index: `path → struct stat`
  - Data index: `(path,blk#) → data[4096]`
- **Paths sorted in DFS order (i.e. full-path indexing)**
- Implications:
  - Data blocks laid out sequentially
  - **Directory scans → range queries**

Full-path indexing yields fast directory scans

Example: grep -r “key” /home/rob/doc/



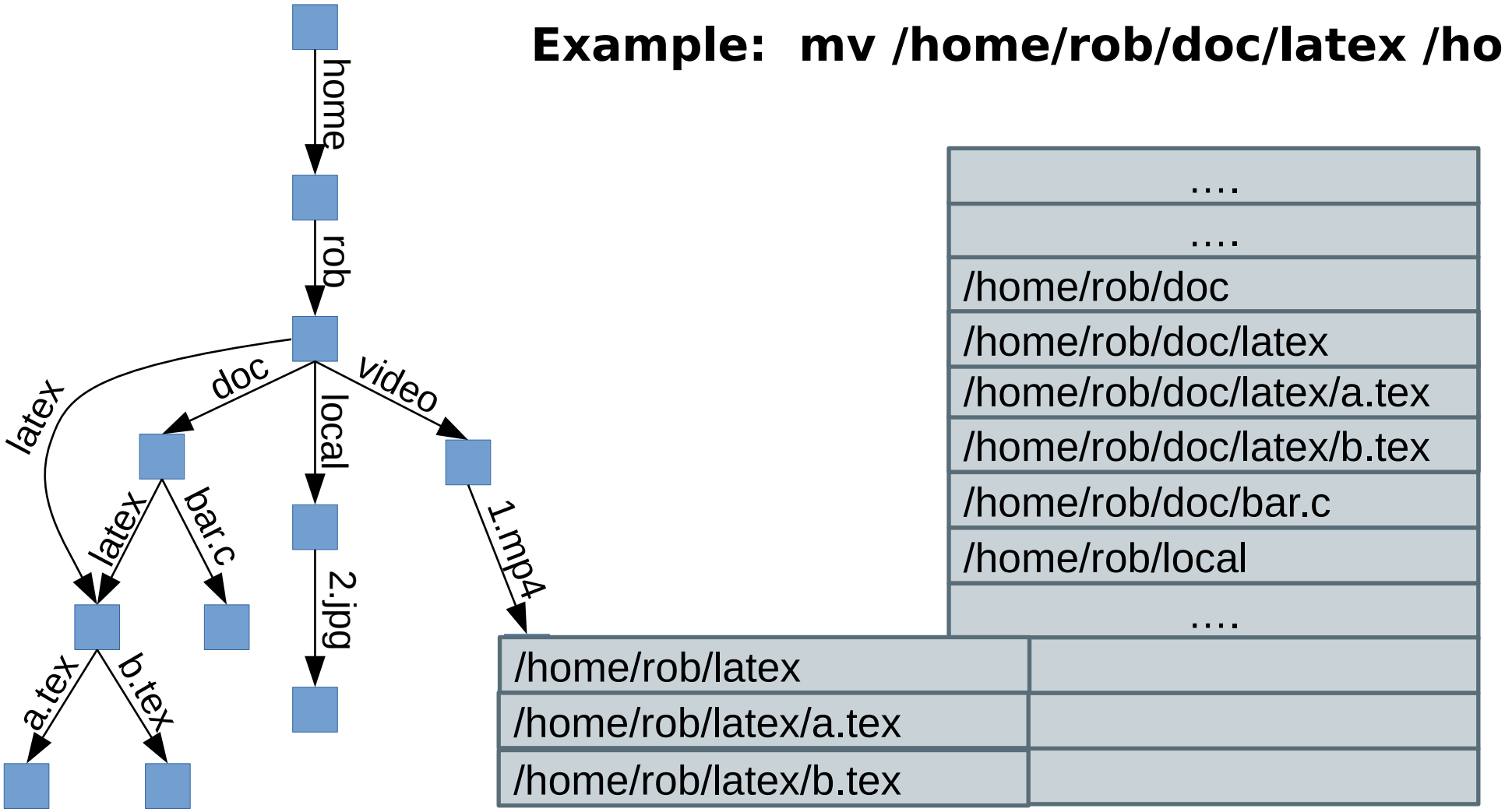
Directory Tree (logical)



Disk (physical)

Rename is expensive when using full-path indexing

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

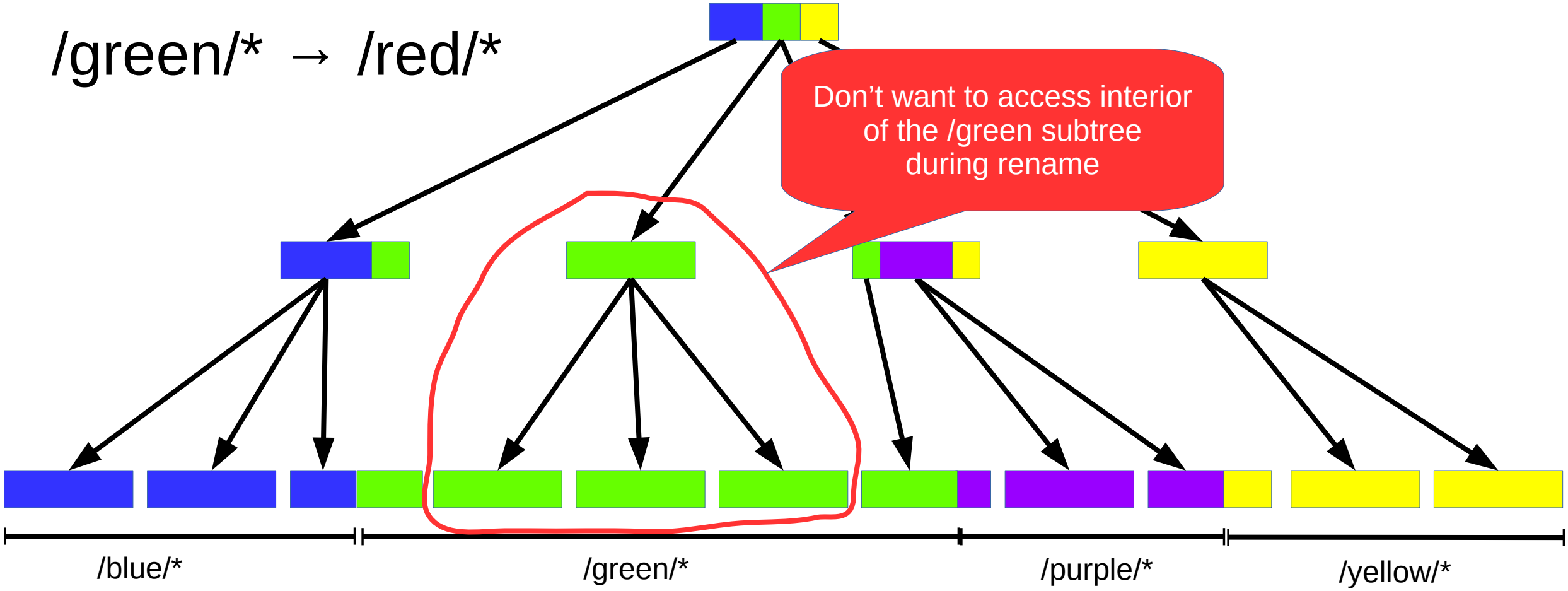


Directory Tree (logical)

Disk (physical)

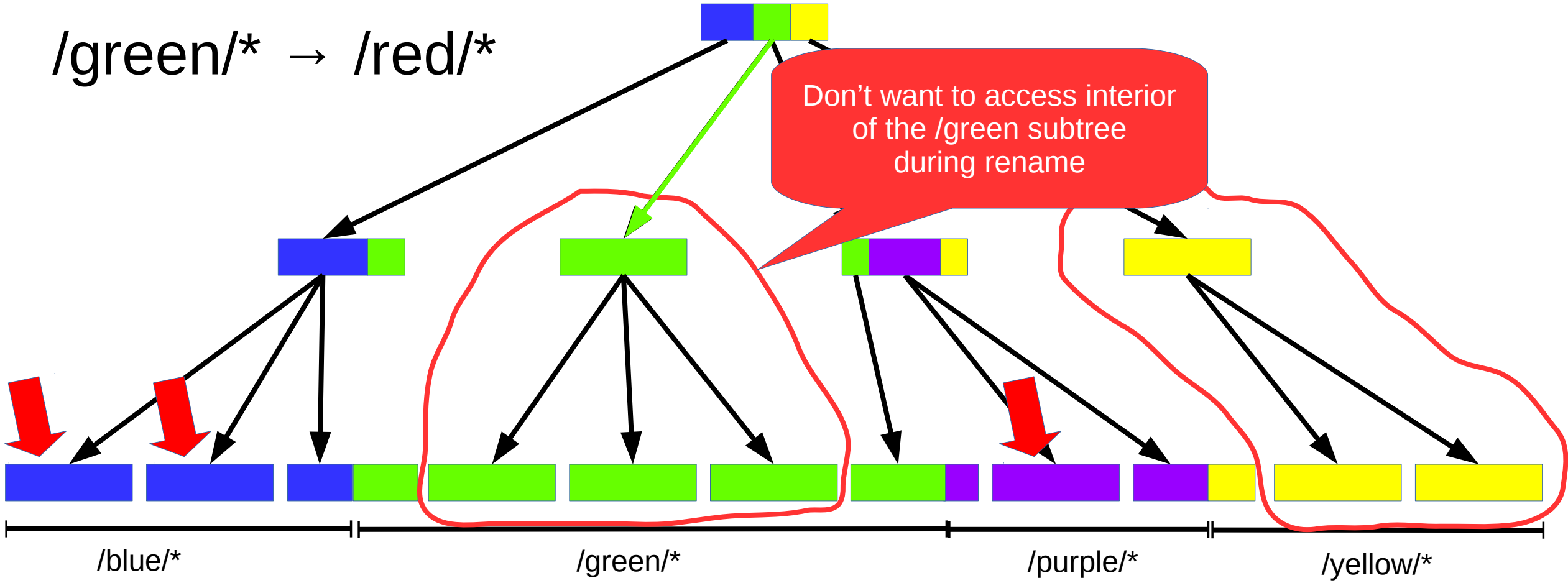
Rename in BetrFS

/green/* → /red/*



Lifted B$^{\varepsilon}$-trees

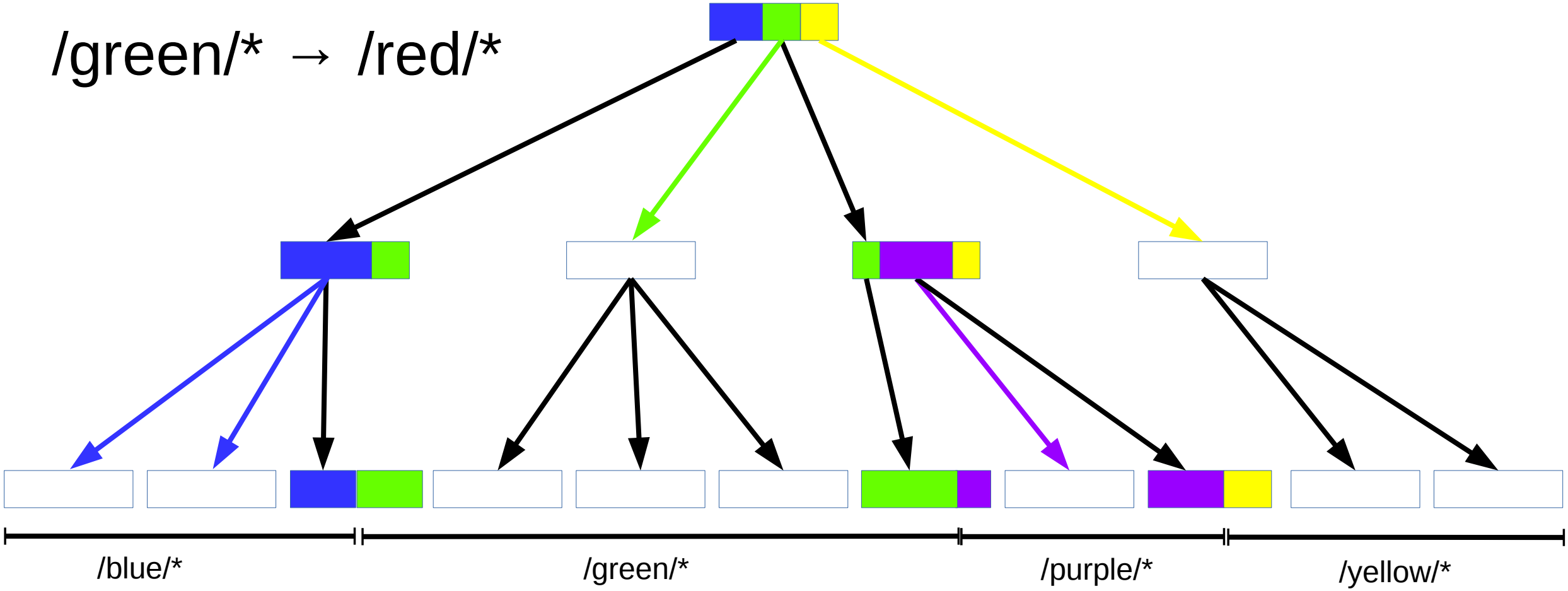
Idea: omit common prefixes from nodes and sub-trees



Lifted B$^{\varepsilon}$-trees

Idea: omit common prefixes from nodes and sub-trees

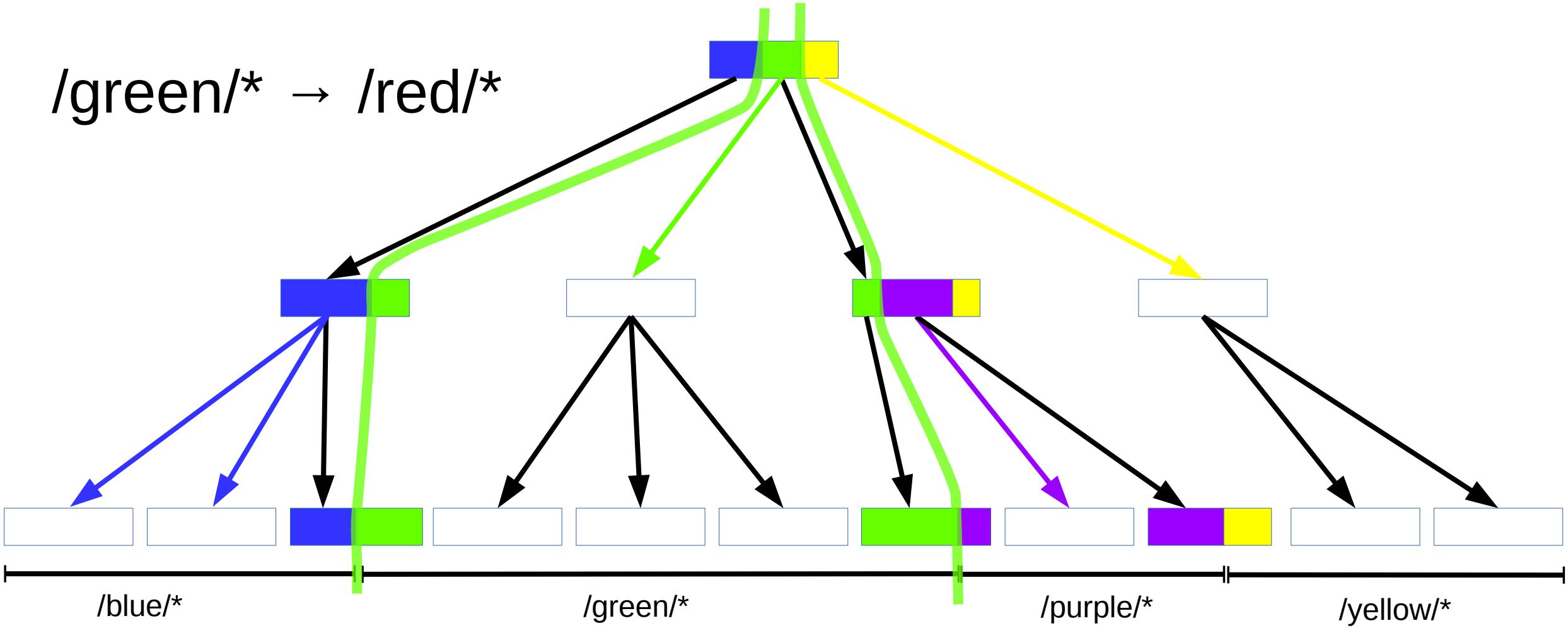
/green/* → /red/*



Range rename in lifted B$^{\varepsilon}$-trees

1.Slice out the /green subtree

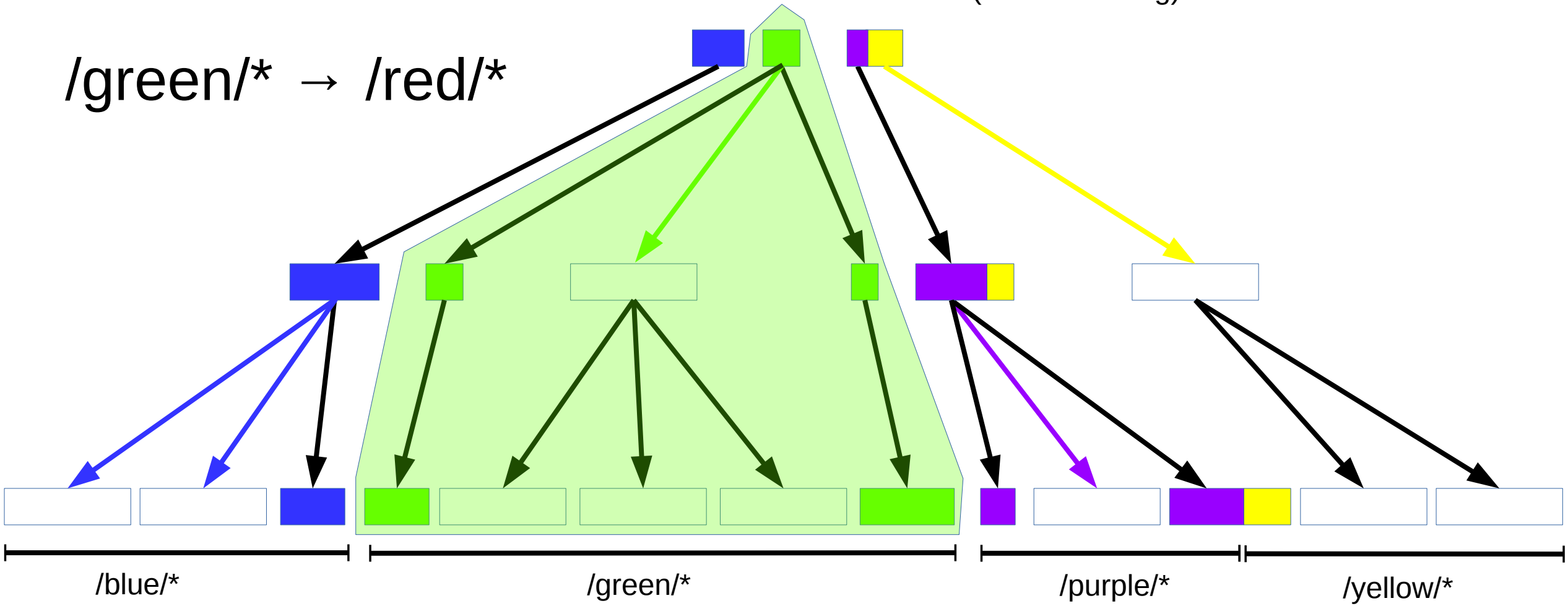
/green/* → /red/*



Range rename in lifted B$^{\varepsilon}$-trees

1. Slice out the /green subtree
   (maintain lifting)

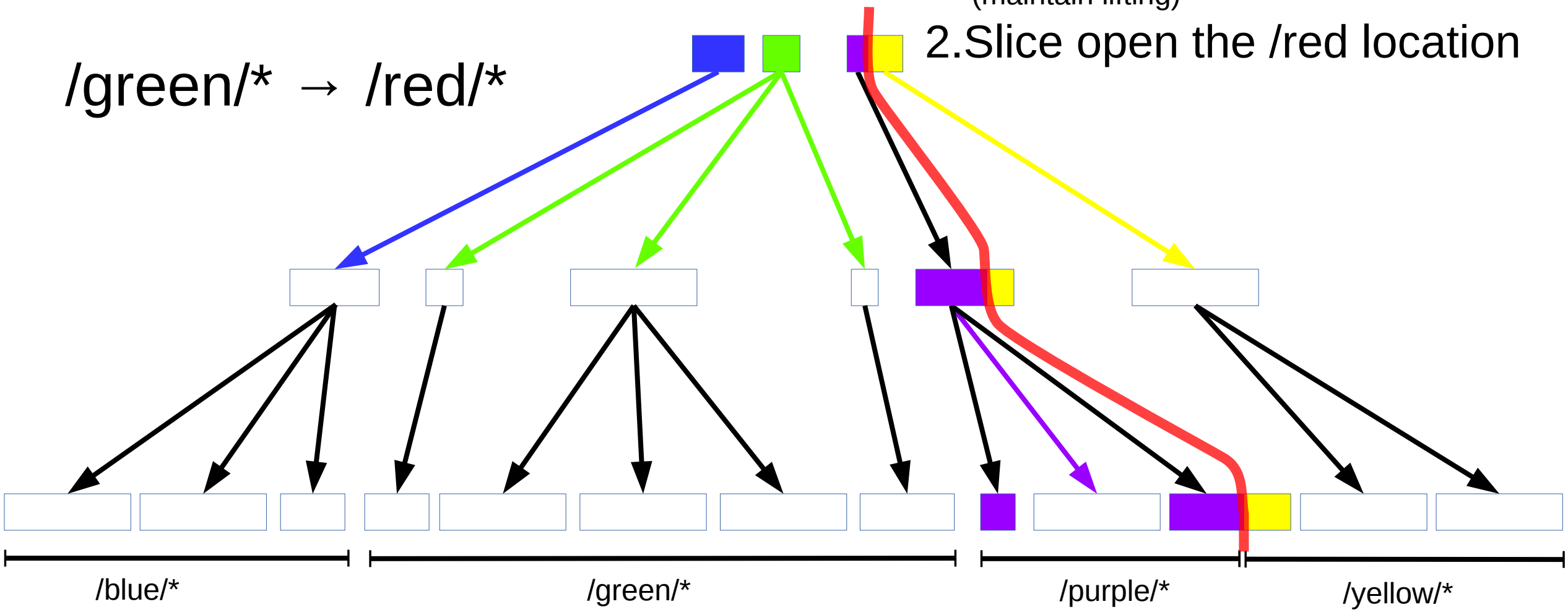
/green/* → /red/*



Range rename in lifted B$^{\varepsilon}$-trees

1. Slice out the /green subtree
   (maintain lifting)
2. Slice open the /red location

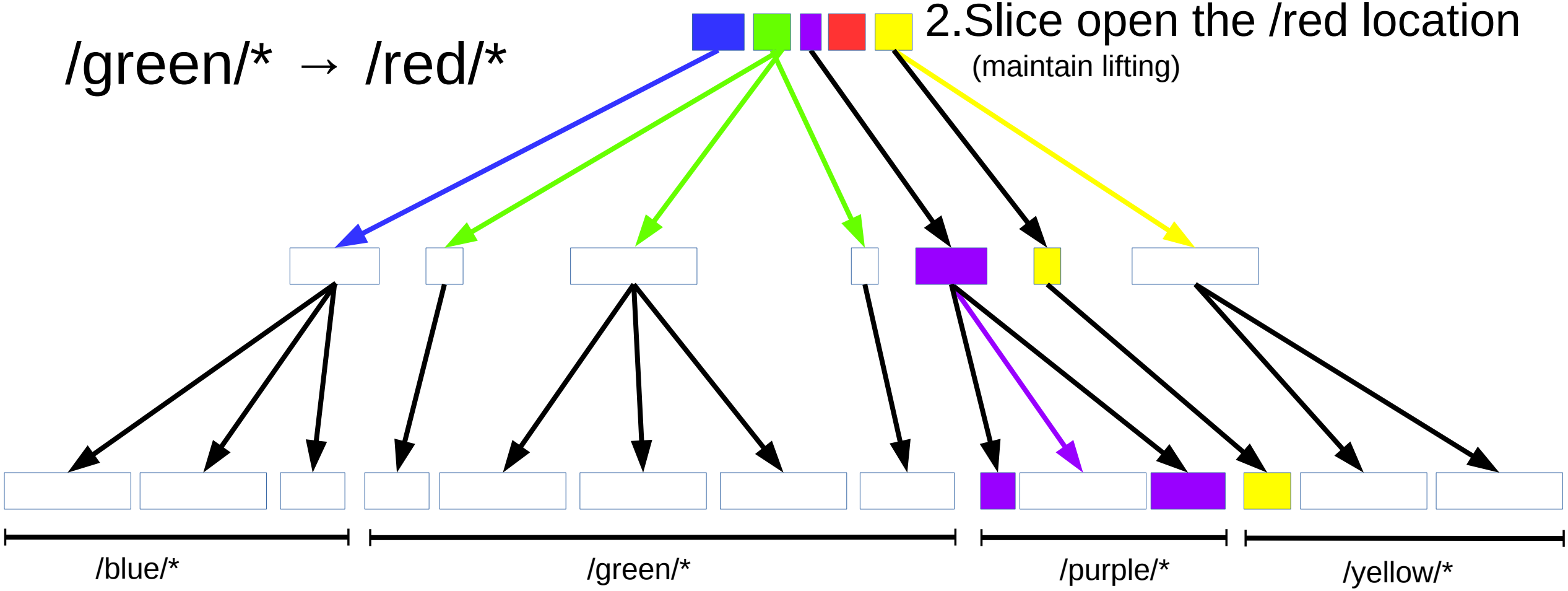
/green/* → /red/*



Range rename in lifted B$^{\varepsilon}$-trees

/green/* → /red/*

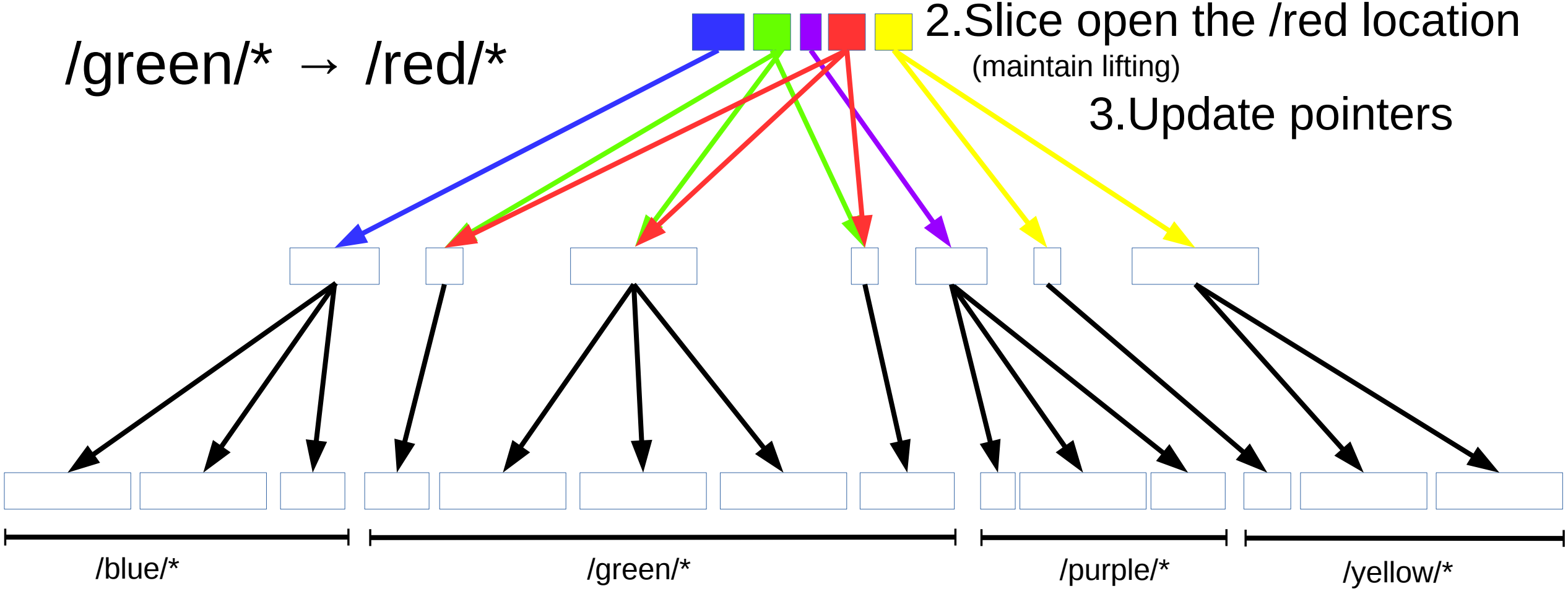
1. Slice out the /green subtree
   (maintain lifting)
2. Slice open the /red location
   (maintain lifting)



Range rename in lifted B$^{\varepsilon}$-trees

/green/* → /red/*

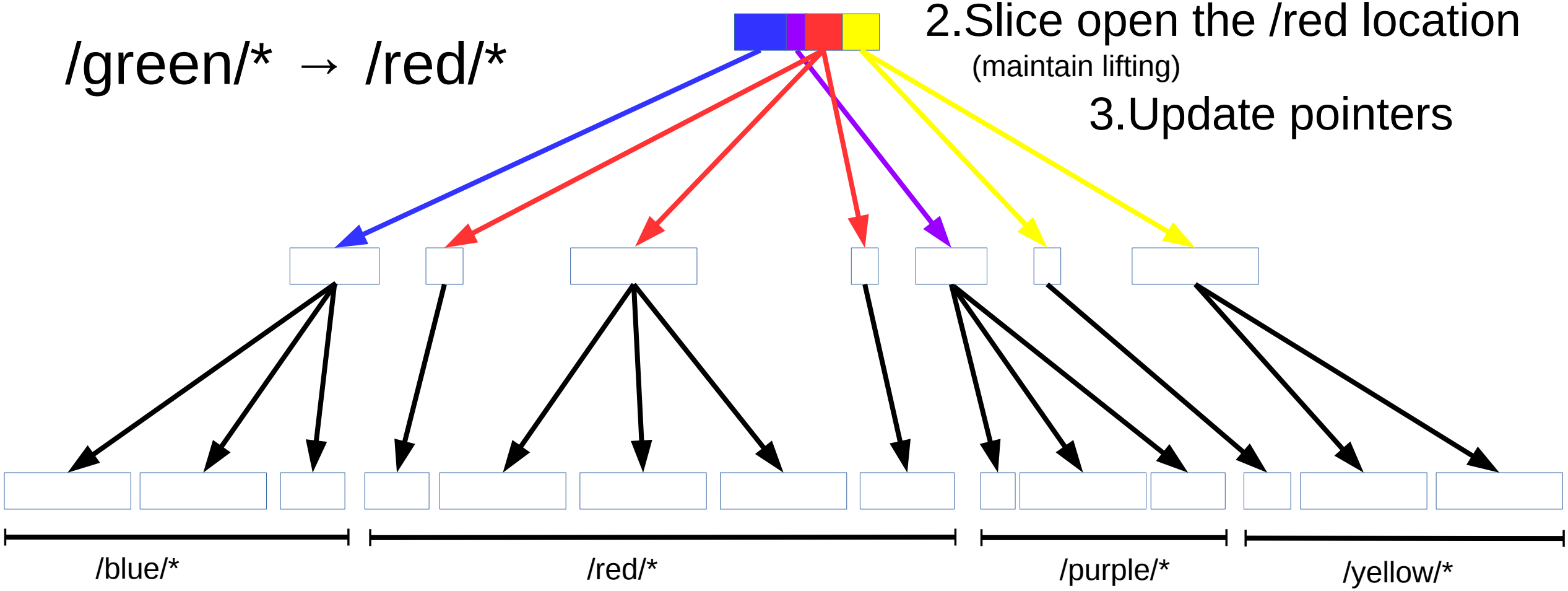
1. Slice out the /green subtree
   (maintain lifting)
2. Slice open the /red location
   (maintain lifting)
3. Update pointers



Range rename in lifted B$^{\varepsilon}$-trees

/green/* → /red/*

1. Slice out the /green subtree
   (maintain lifting)
2. Slice open the /red location
   (maintain lifting)
3. Update pointers



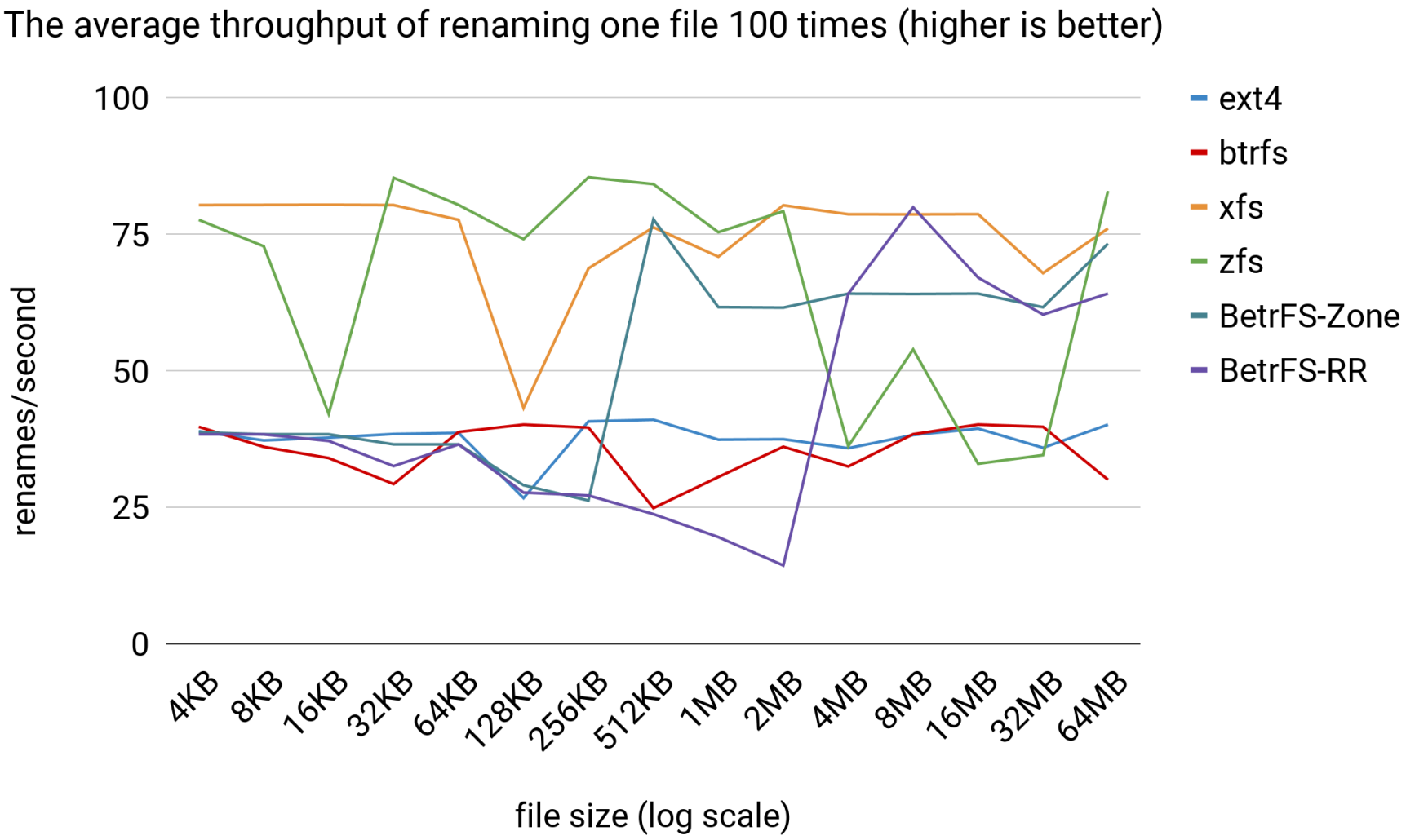
IO complexity of range rename

- Each slice touches a root-to-leaf path
- Maintaining lifting doesn't need to access any additional nodes

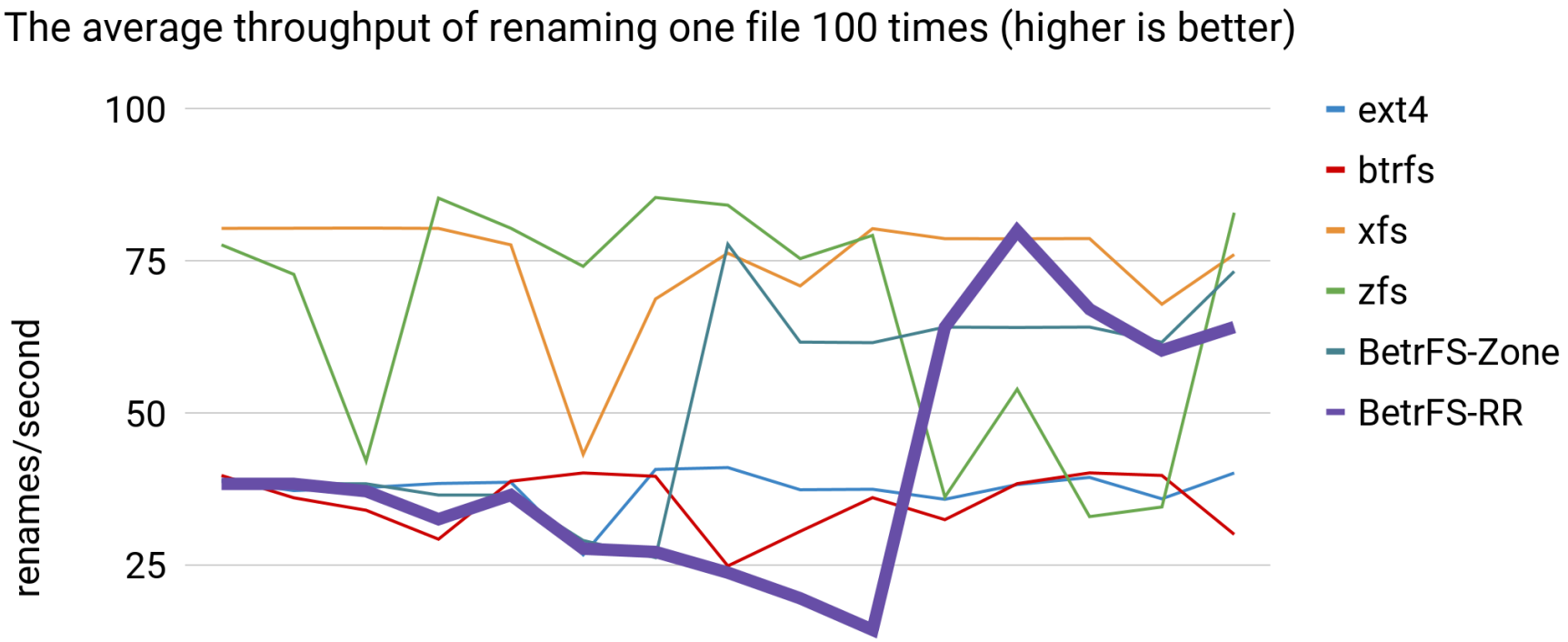
$$O(\text{tree height})$$

$$= O(\log_B N)$$

Rename Throughput

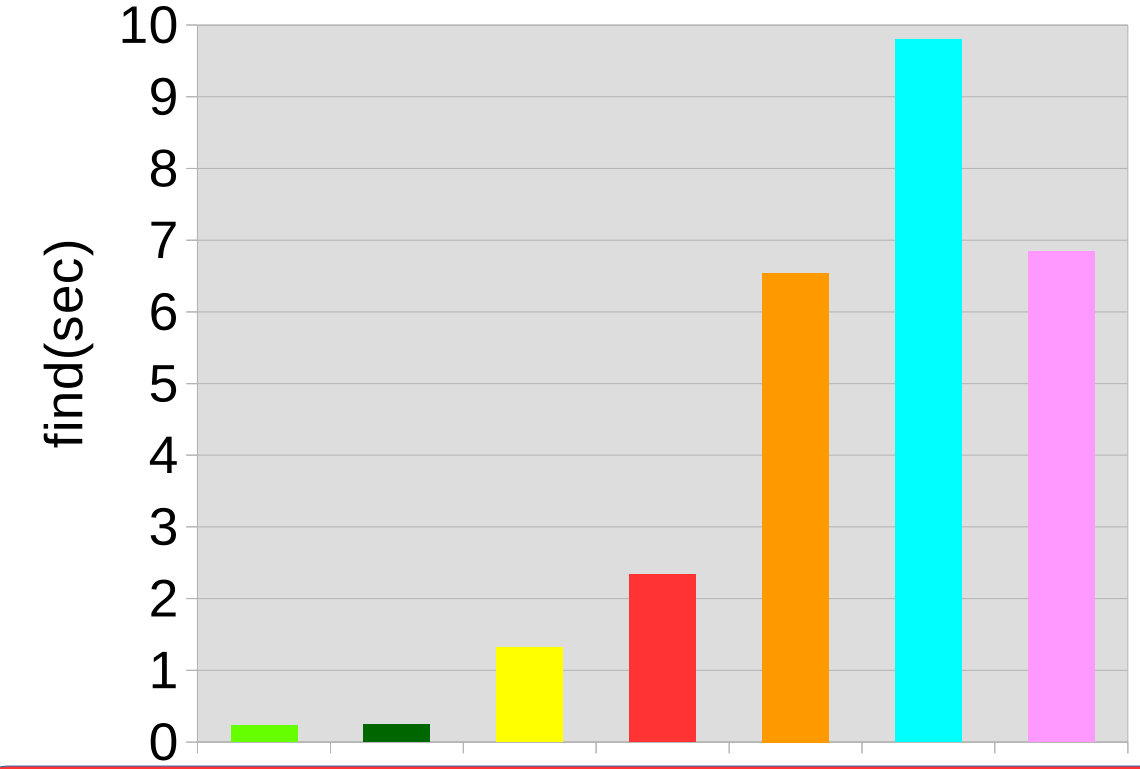


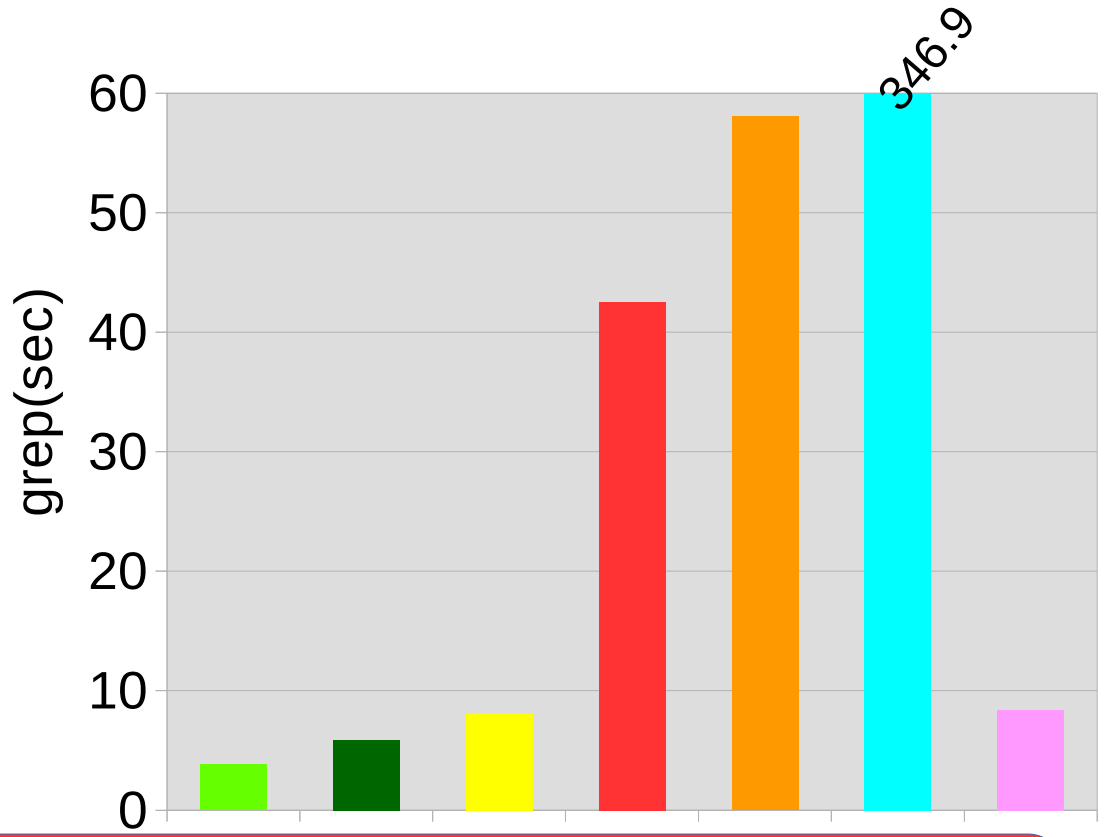
Rename Throughput



Rename can be as fast as inode-based file systems

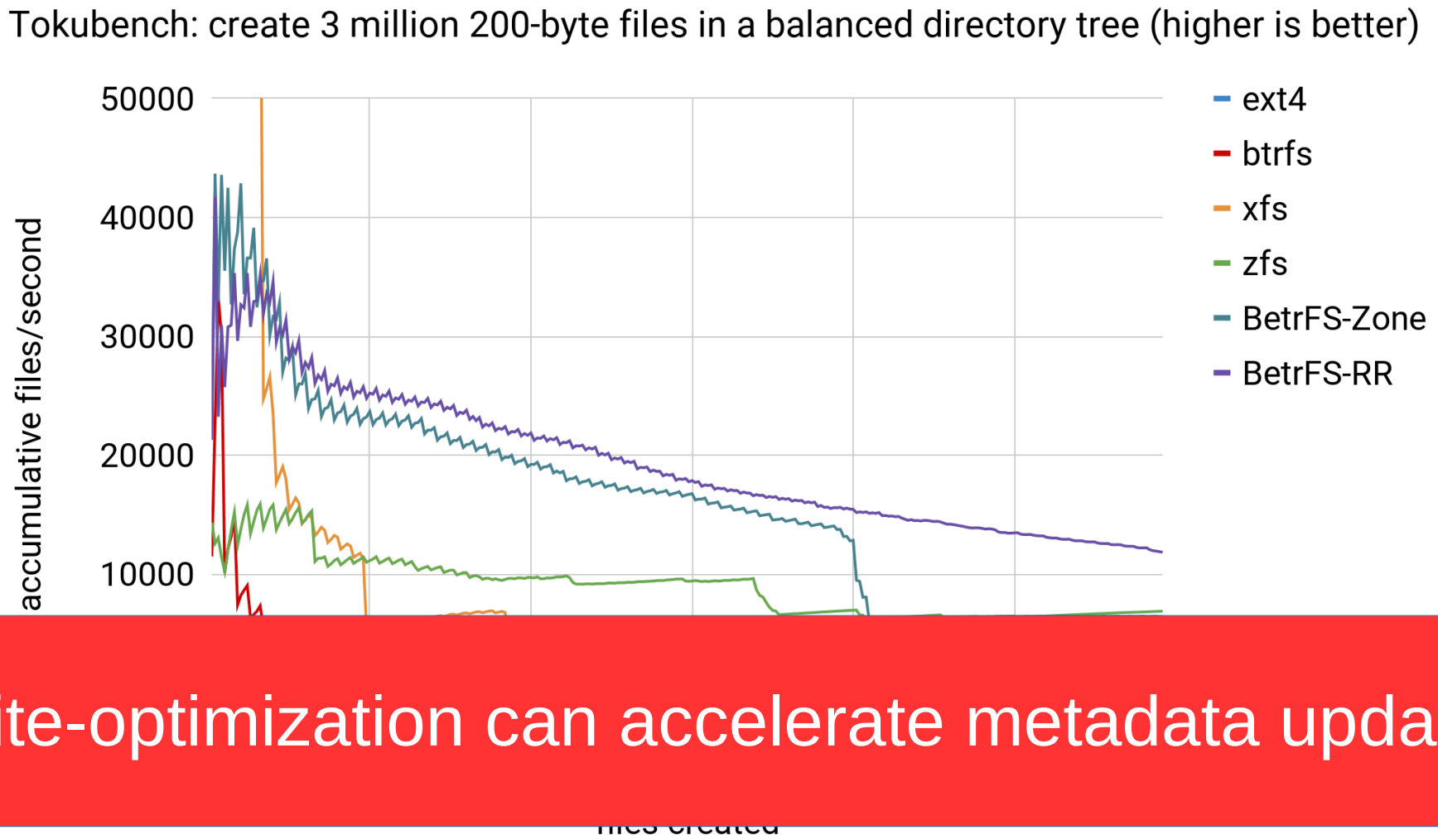
Recursive directory traversals





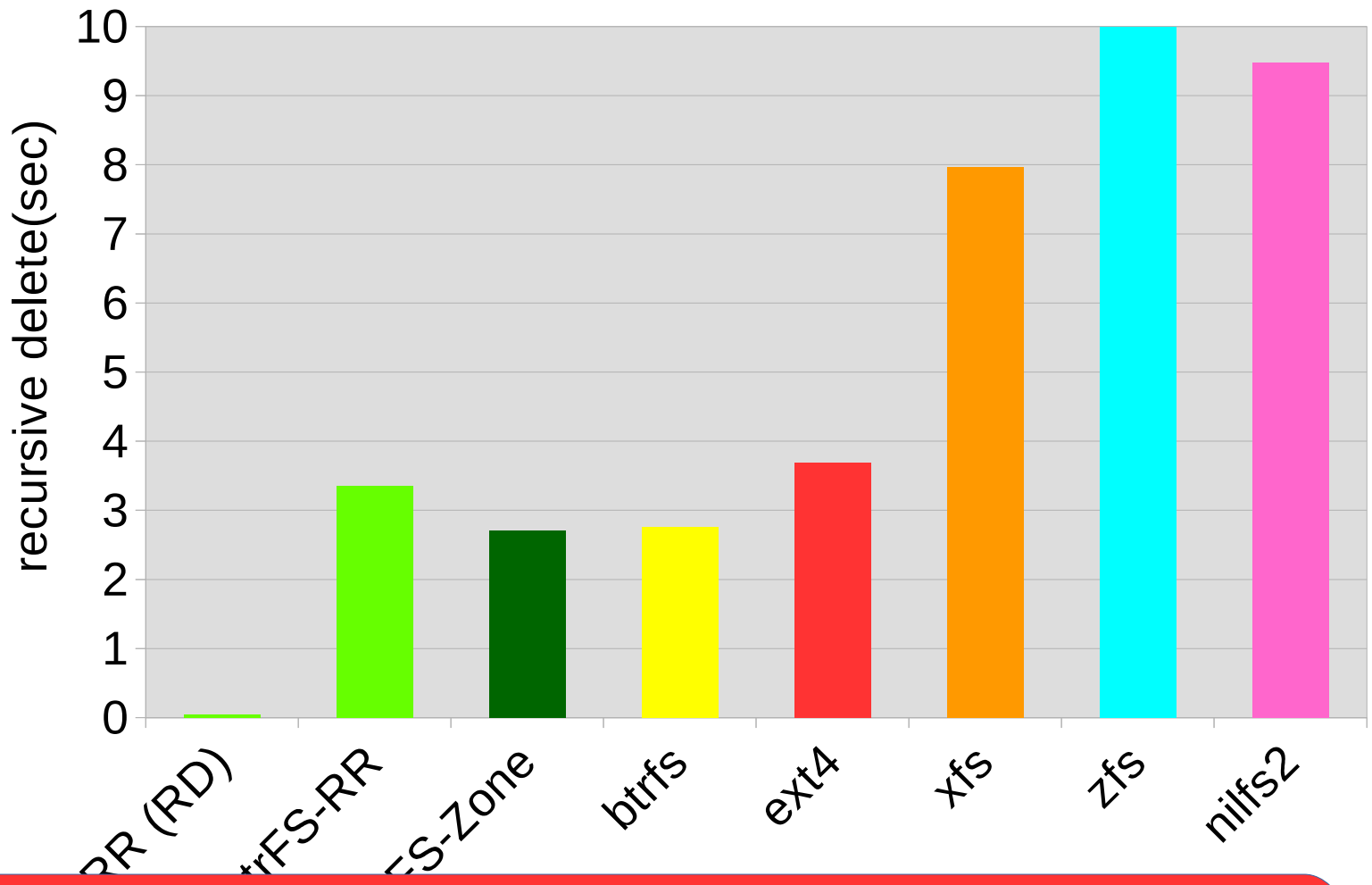
Full-path indexing can dramatically improve metadata scans

Tokubench



Write-optimization can accelerate metadata updates

Recursive delete



Full-path indexing can unleash new optimizations

Conclusion

- Write optimization can solve a lot of metadata problems
- But write optimization can require rethinking how we organize our metadata
- And we sometimes need to extend the underlying data structures to support our metadata needs

Code available at betrfs.org