How to Copy Files

Yang Zhan[1,2], **Alex Conway[3,4]**, Ian Groombridge[5], Yizheng Jiao[1], Nirjhar Mukherjee[1], Michael A. Bender[6], Martín Farach-Colton[3], William Jannen[7], Rob Johnson[4], Donald E. Porter[1], Jun Yuan[5]

1 2 3 4 5 6 7

Copying is Ubiquitous and Important

cp -r

container instantiation

vmrun start

backup

Copying

Physical Copy

Copying

Physical Copy

High Latency

High Space Use

Existing Logical Copy Implementations

BTRFS

Leverages the underlying copy-on-write B-tree to implement `cp --reflink`

XFS

Uses an update-in-place B-tree but supports sharing data blocks with copy-on-write via `cp --reflink`

ZFS

Implements a limited version of copy-on-write copying via `zfs clone`

Copying

Physical Copy

Copy on Write

High Latency

High Space Use

Copying

Physical Copy

High Latency

High Space Use

Copy on Write

Low Latency

Better Space Use

High Fragmentation

Copying

Physical Copy

High Latency

High Space Use

Copy on Write

Low Latency

Better Space Use

High Fragmentation

Space Amplification and Fragmention

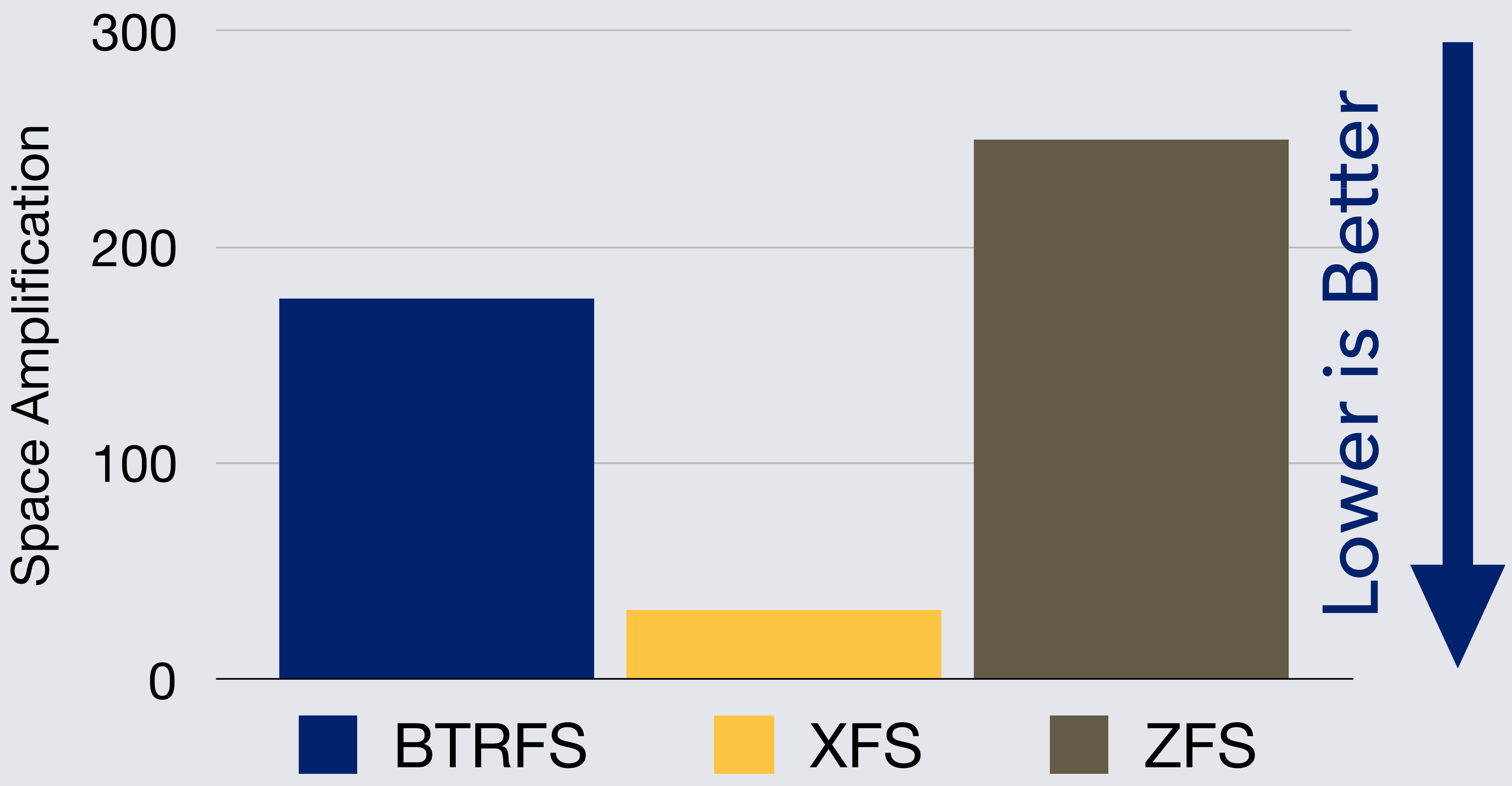
Dell Optiplex 790
4-core 3.40 GHz Intel Core i7 CPU
4GiB RAM
500GB 7200 RPM SATA disk

Init: 64 4MiB files with random data.
Each round: logically copy all files, then change 16B in each file (1KiB total)

Space Amplification and Fragmention

Dell Optiplex 790
4-core 3.40 GHz Intel Core i7 CPU
4GiB RAM
500GB 7200 RPM SATA disk

Init: 64 4MiB files with random data.
Each round: logically copy all files, then change 16B in each file (1KiB total)

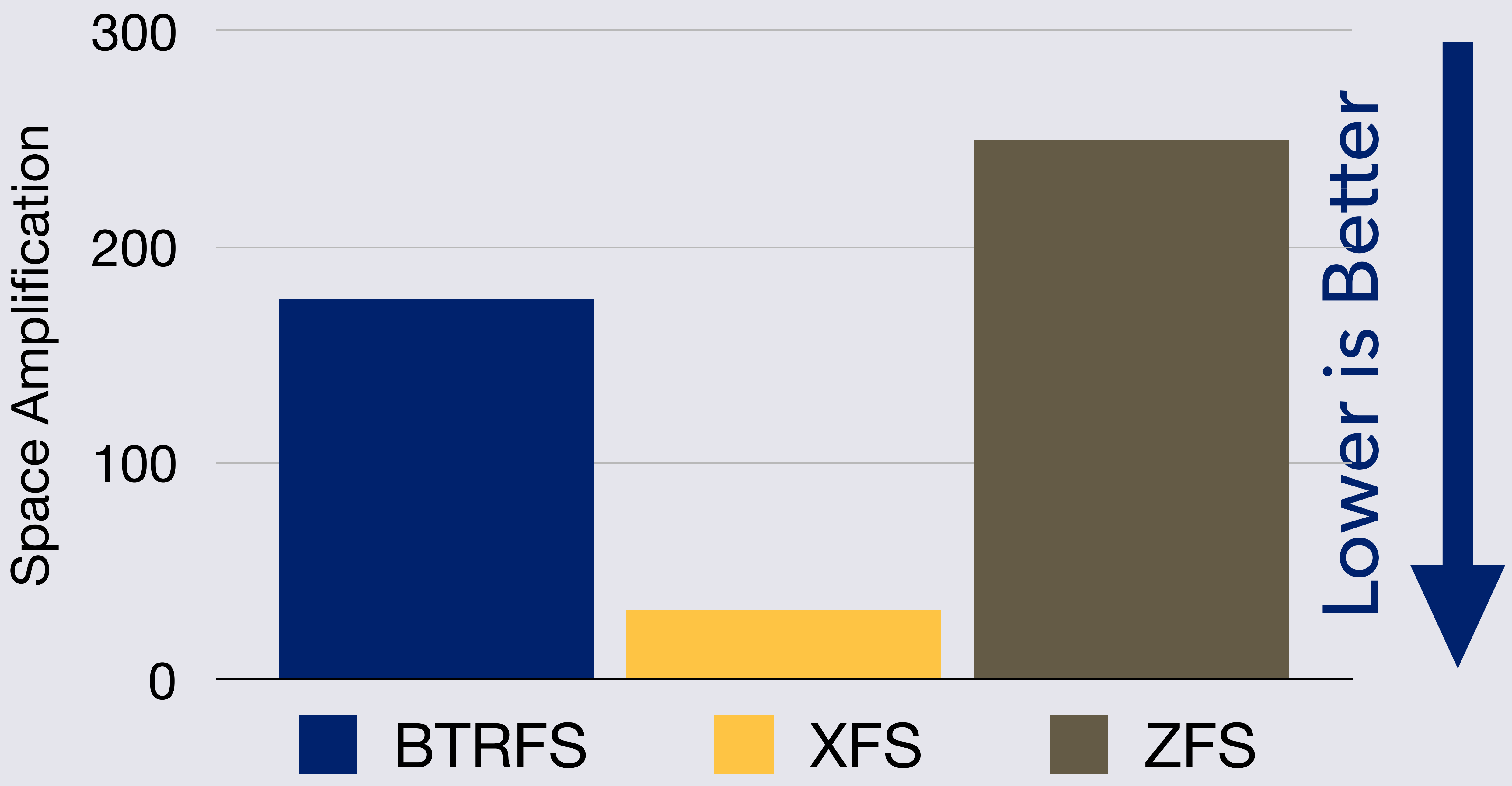


Space Amplification
additional file system size / added data (1KiB)

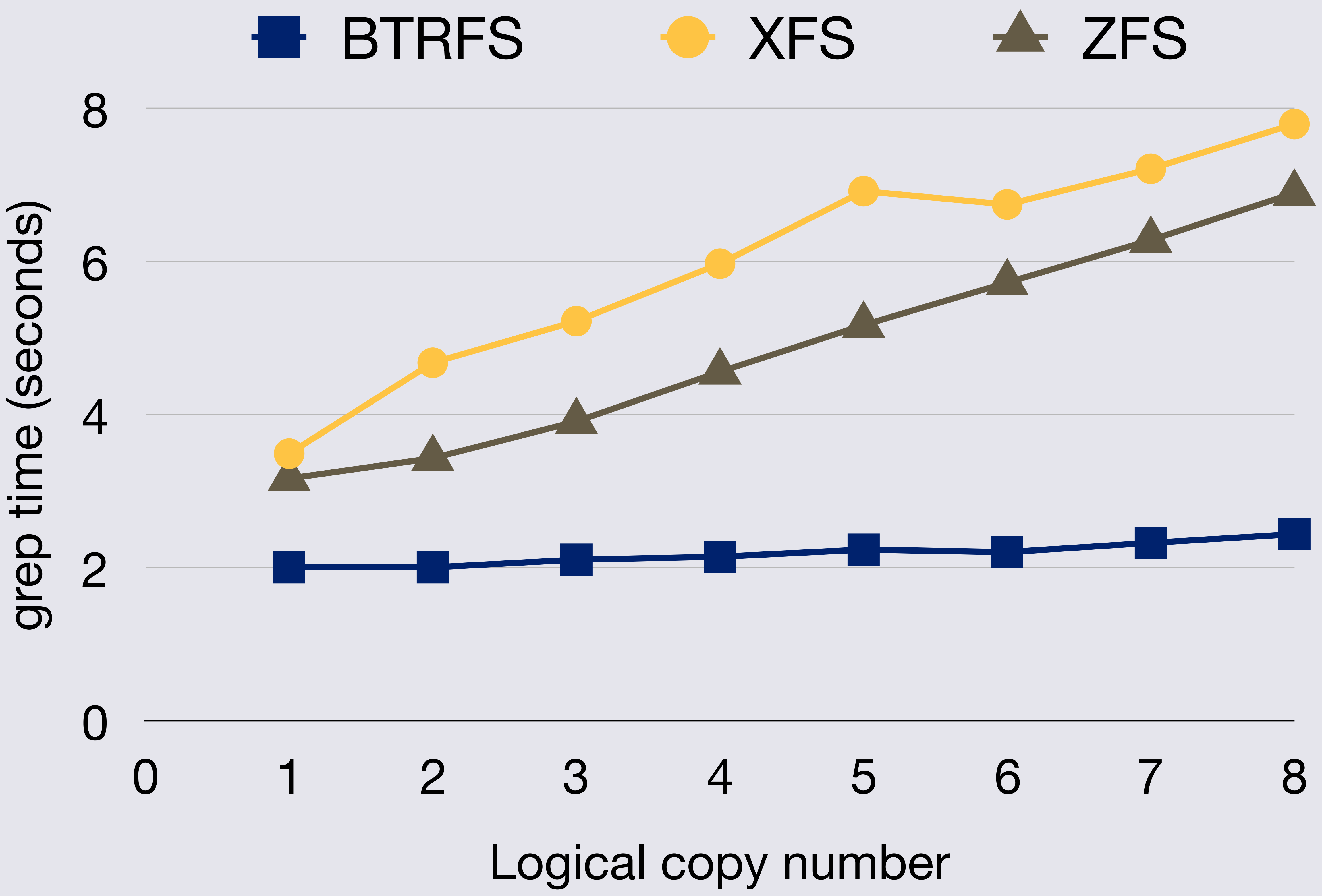
Space Amplification and Fragmention

Dell Optiplex 790
4-core 3.40 GHz Intel Core i7 CPU
4GiB RAM
500GB 7200 RPM SATA disk

Init: 64 4MiB files with random data.
Each round: logically copy all files, then change 16B in each file (1KiB total)



Space Amplification
additional file system size / added data (1KiB)



Fragmentation
measured by timing a grep over the latest copy

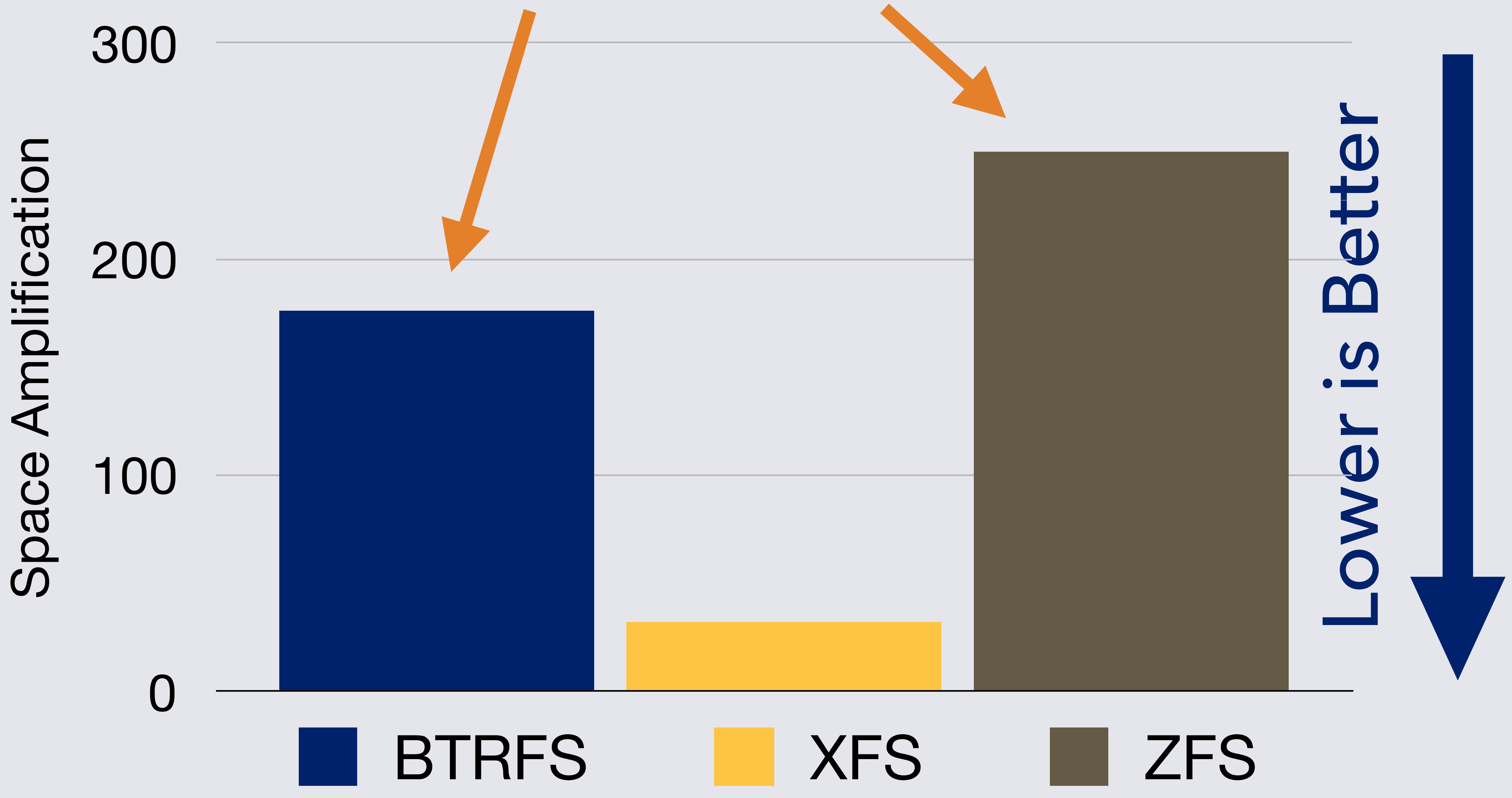
Space Amplification and Fragmention

Dell Optiplex 790
4-core 3.40 GHz Intel Core i7 CPU
4GiB RAM
500GB 7200 RPM SATA disk

Init... with random data.

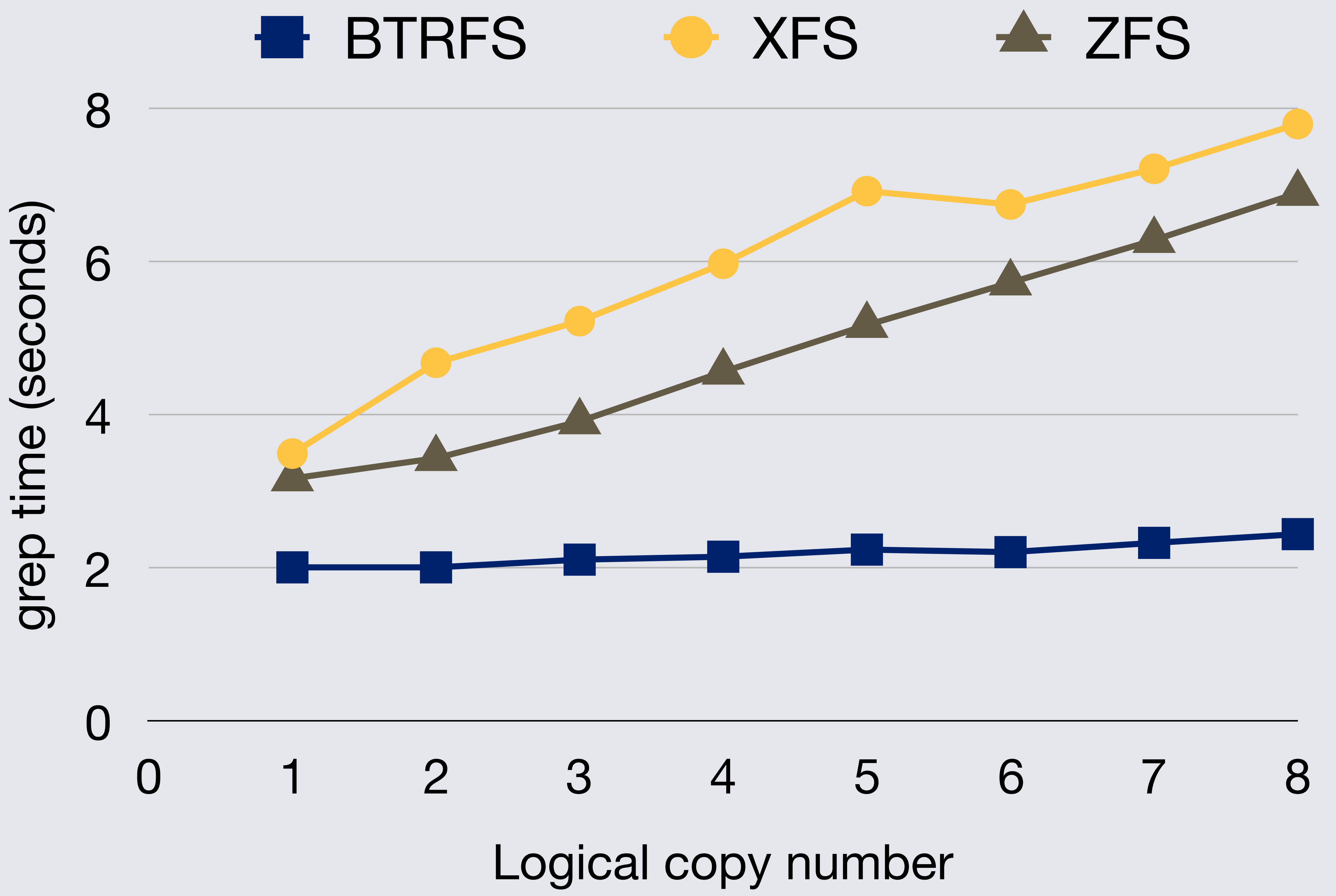
Ea... all files, then change 16B in each file (1KiB total)

Have large space amplification



Space Amplification

additional file system size / added data (1KiB)



Fragmentation

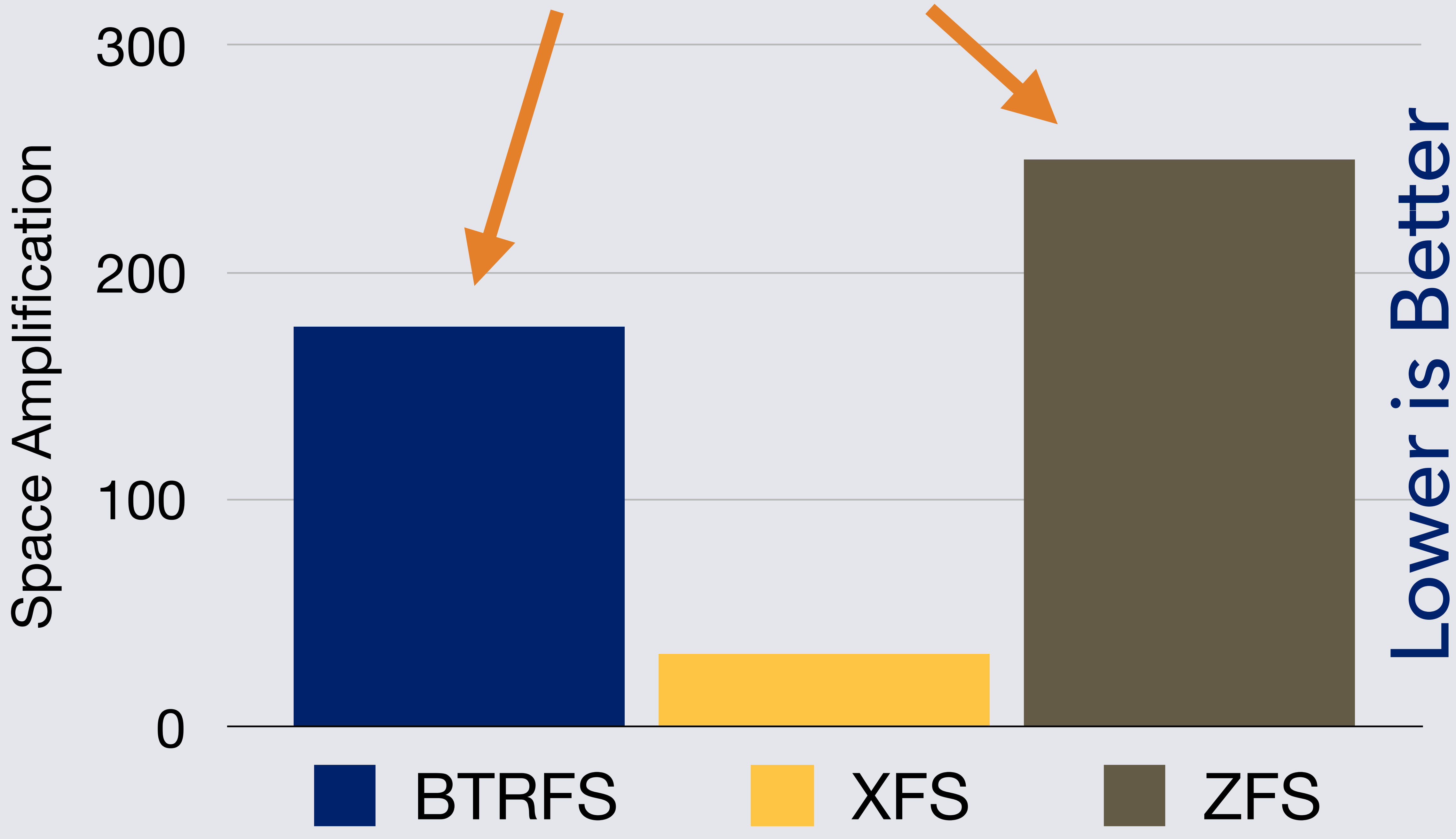
measured by timing a grep over the latest copy

Space Amplification and Fragmention

Dell Optiplex 790
4-core 3.40 GHz Intel Core i7 CPU
4GiB RAM
500GB 7200 RPM SATA disk

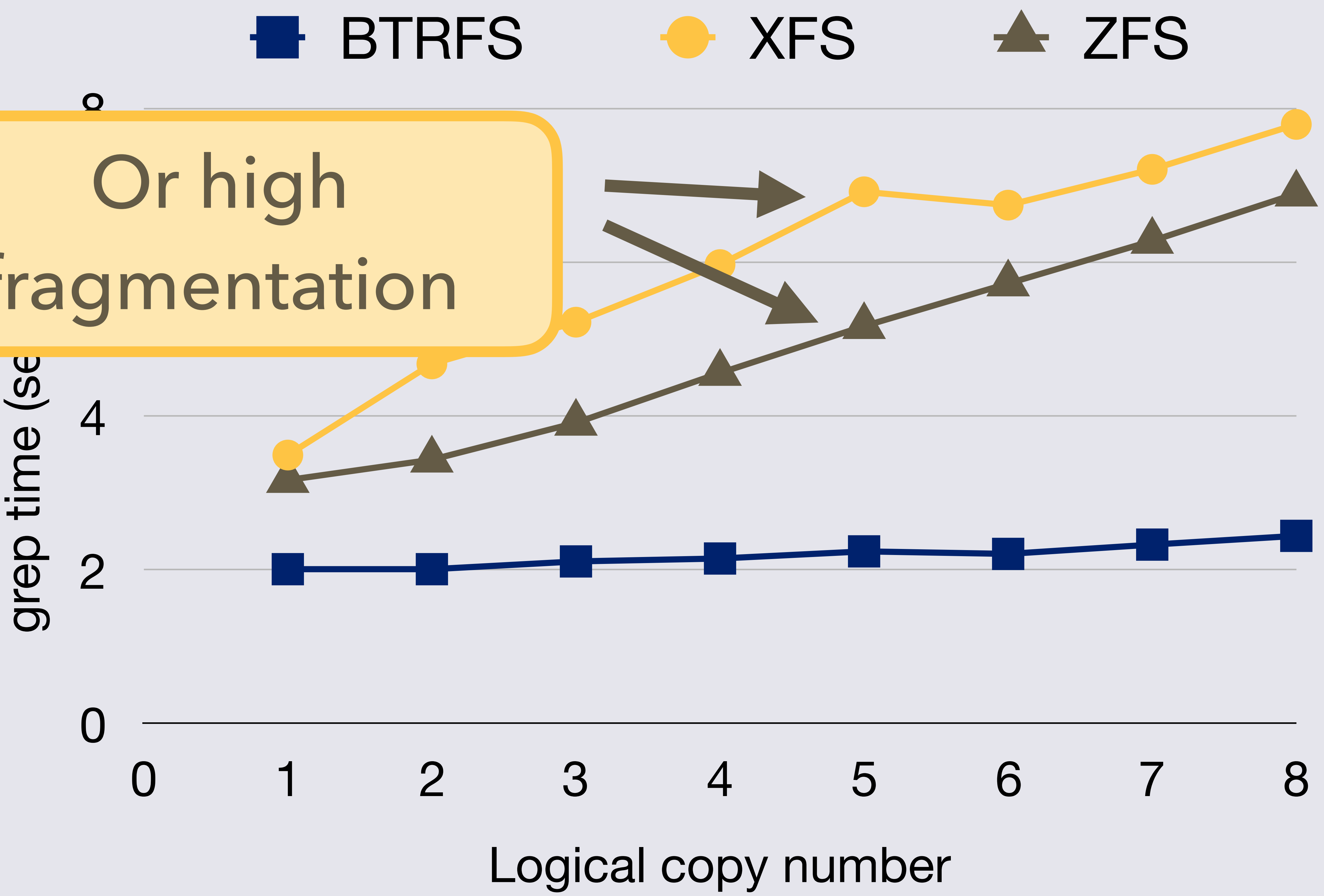
Init… with random data.
Ea… all files, then change 16B in each file (1KiB total)

Have large space amplification



Lower is Better

Or high fragmentation



Space Amplification
additional file system size / added data (1KiB)

Fragmentation
measured by timing a grep over the latest copy

Copy Performance Goals

Copy Performance Goals

Space efficient

Low latency

Specific to Copying

Copy Performance Goals

Space efficient

Low latency

Specific to Copying

Fast writes

Fast reads

General file system

Copy Performance Goals

Space efficient

Low latency

Specific to Copying

Locality

Fast writes

Fast reads

General file system

Contributions of this Paper

A high performance logical copy implementation…

Contributions of this Paper

A high performance logical copy implementation…

In BεtrFS, which leverages the properties of Copy-on-Write B^{ε}-trees

Contributions of this Paper

A high performance logical copy implementation…

In BεtrFS, which leverages the properties of Copy-on-Write B^{ε}-trees

Copy-specific	General file system
Space efficient	Fast writes
Low latency	Fast reads

Contributions of this Paper

A high performance logical copy implementation…

In BεtrFS, which leverages the properties of Copy-on-Write B^{ε}-trees

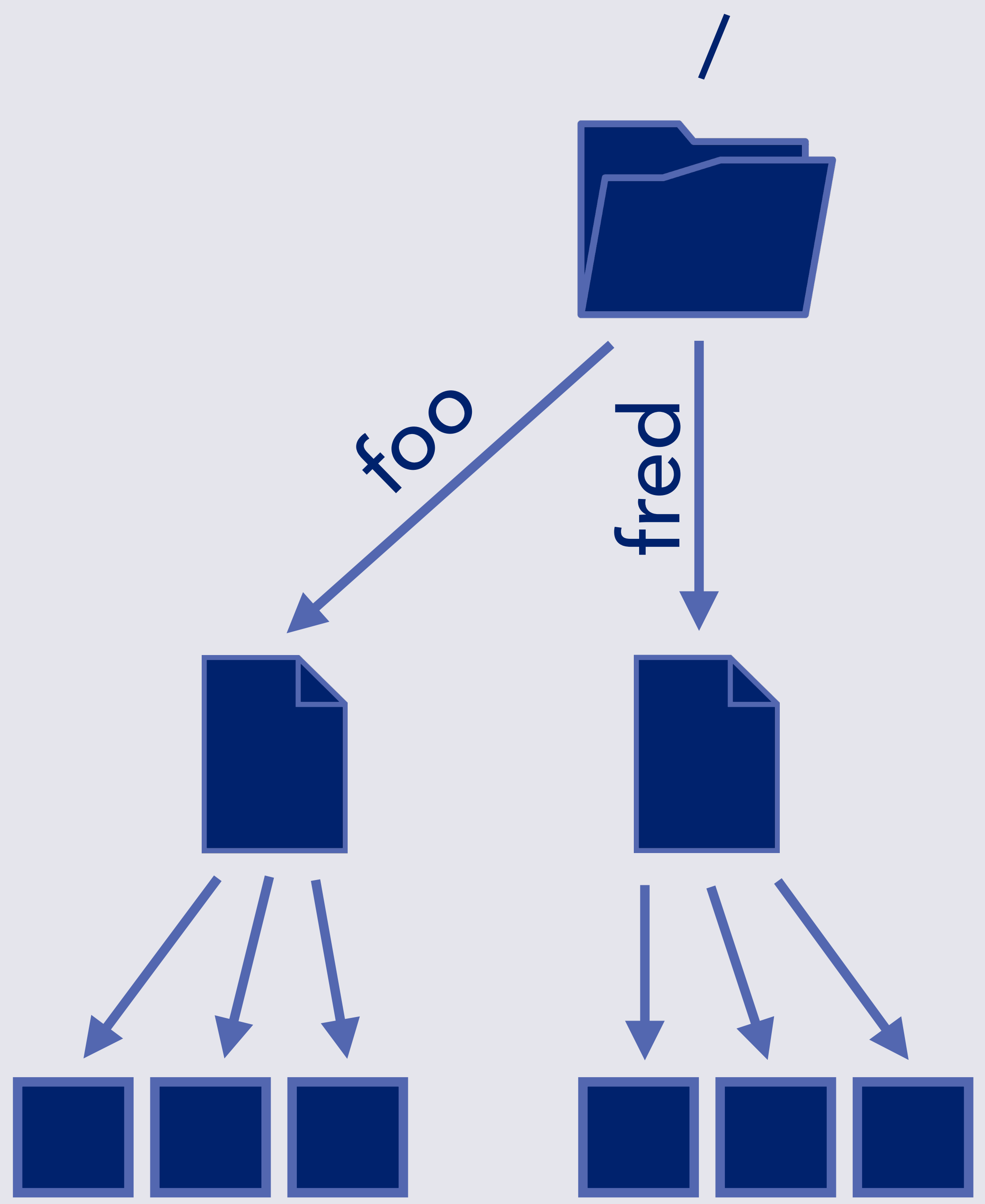
Copy-specific		General file system	
Space efficient	✓	Fast writes	✓
Low latency	✓	Fast reads	✓

What is the Challenge of Logical Copy?

Example: Logical copy in an inode file system

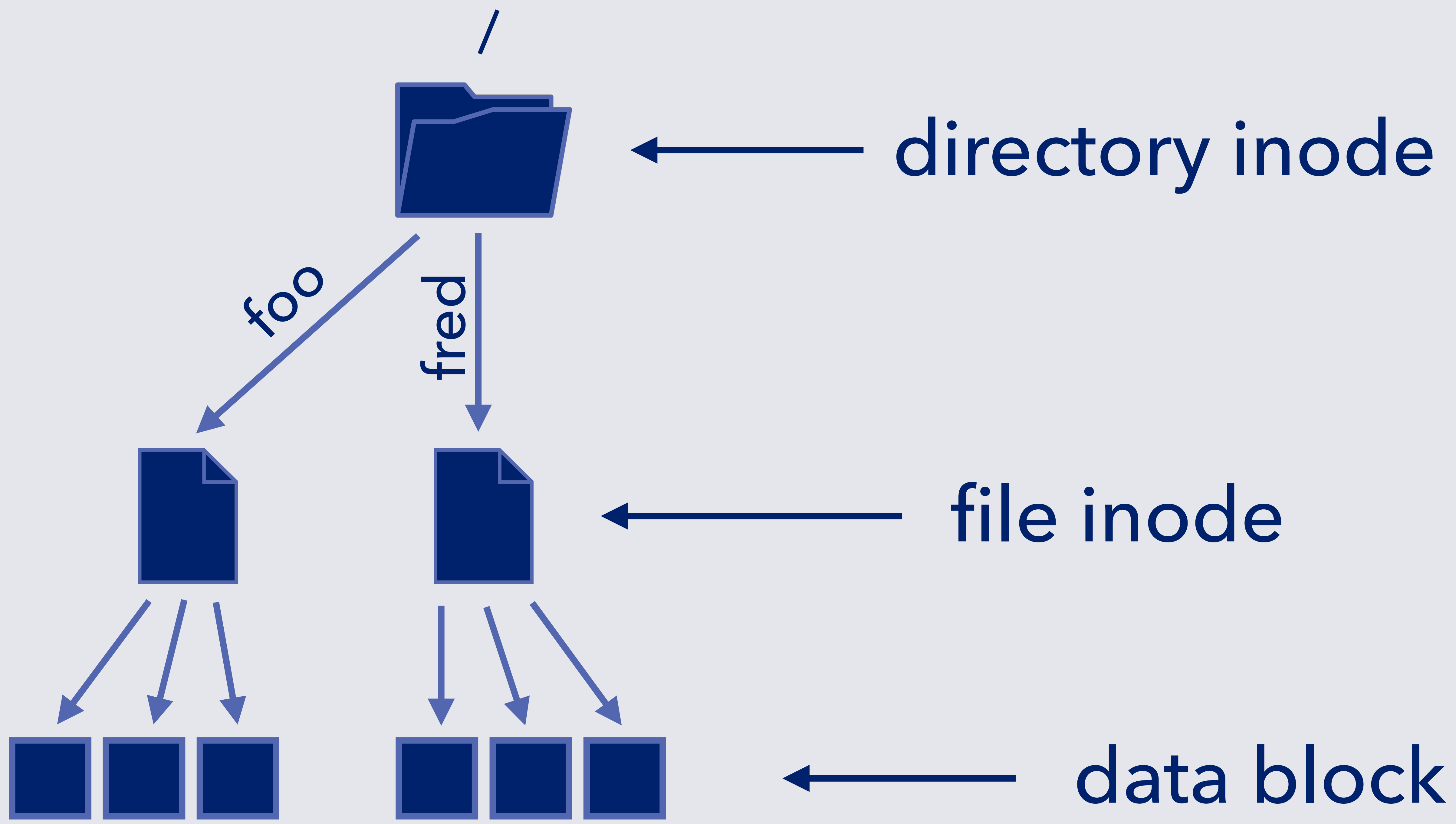
Logical Copy in an Inode File System

Copy /foo to /bar



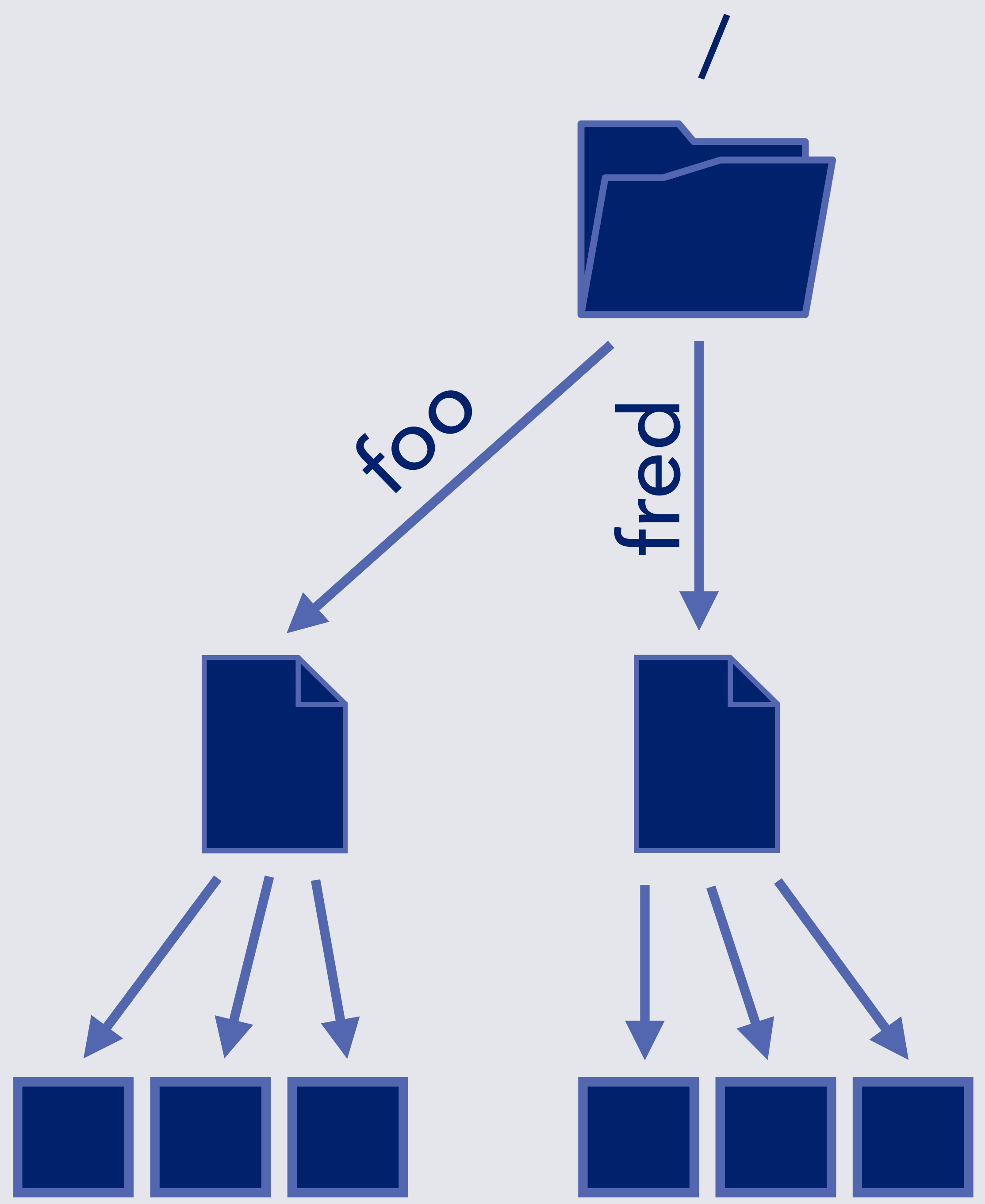
Logical Copy in an Inode File System

Copy /foo to /bar



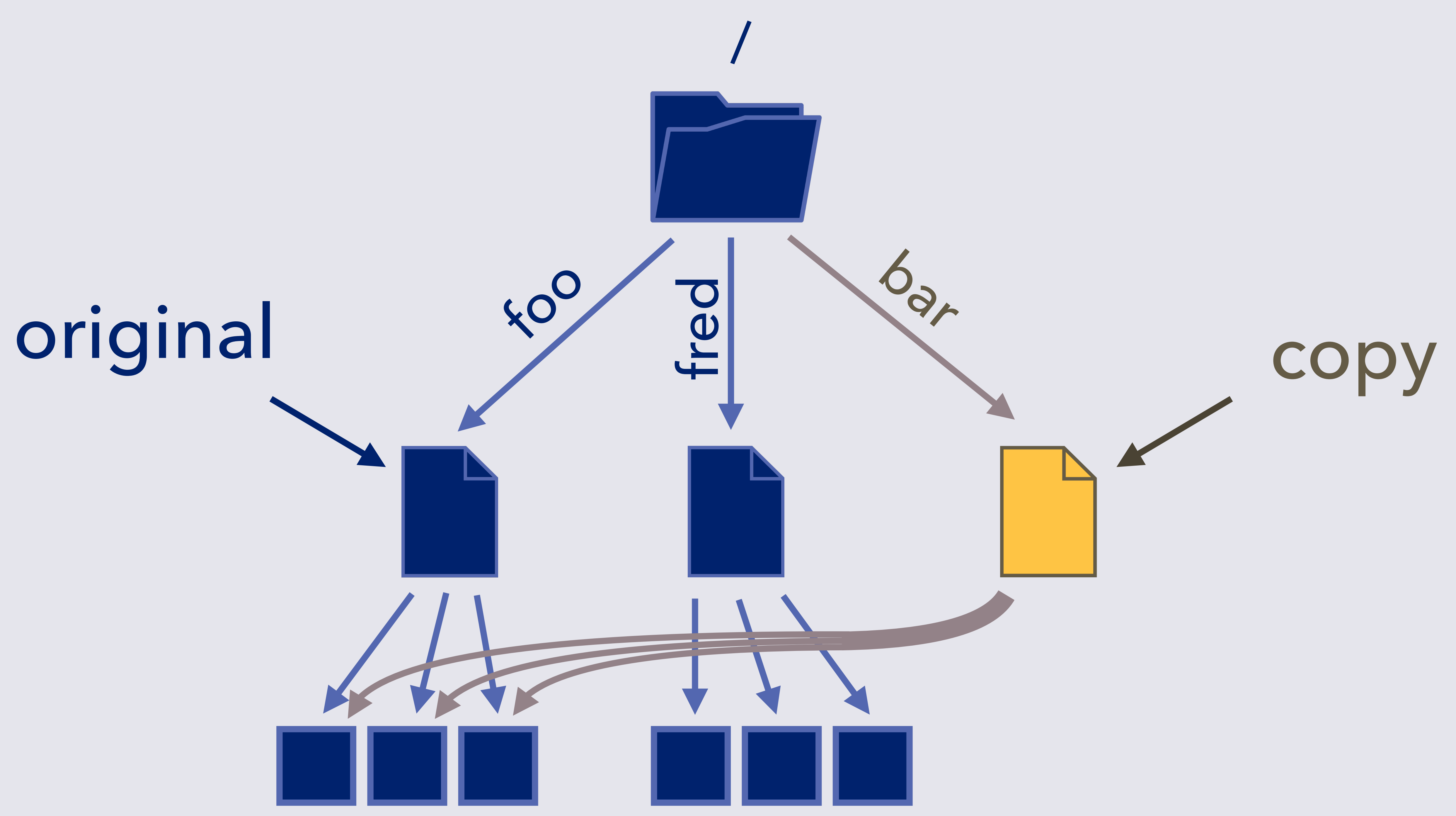
Logical Copy in an Inode File System

Copy /foo to /bar



Logical Copy in an Inode File System

Copy /foo to /bar



Logical Copy in an Inode File System

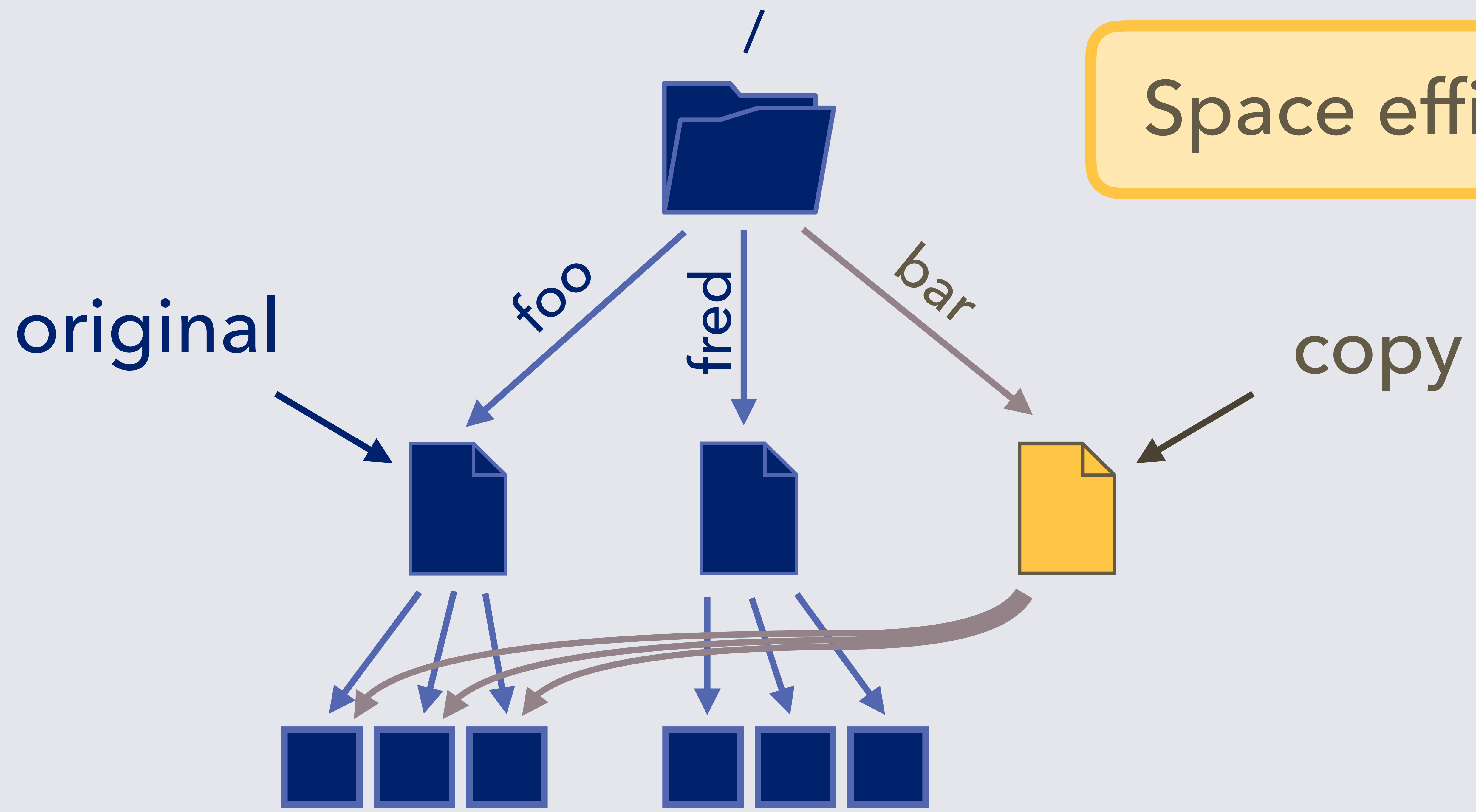
Copy /foo to /bar

Low latency ?

Space efficient ?

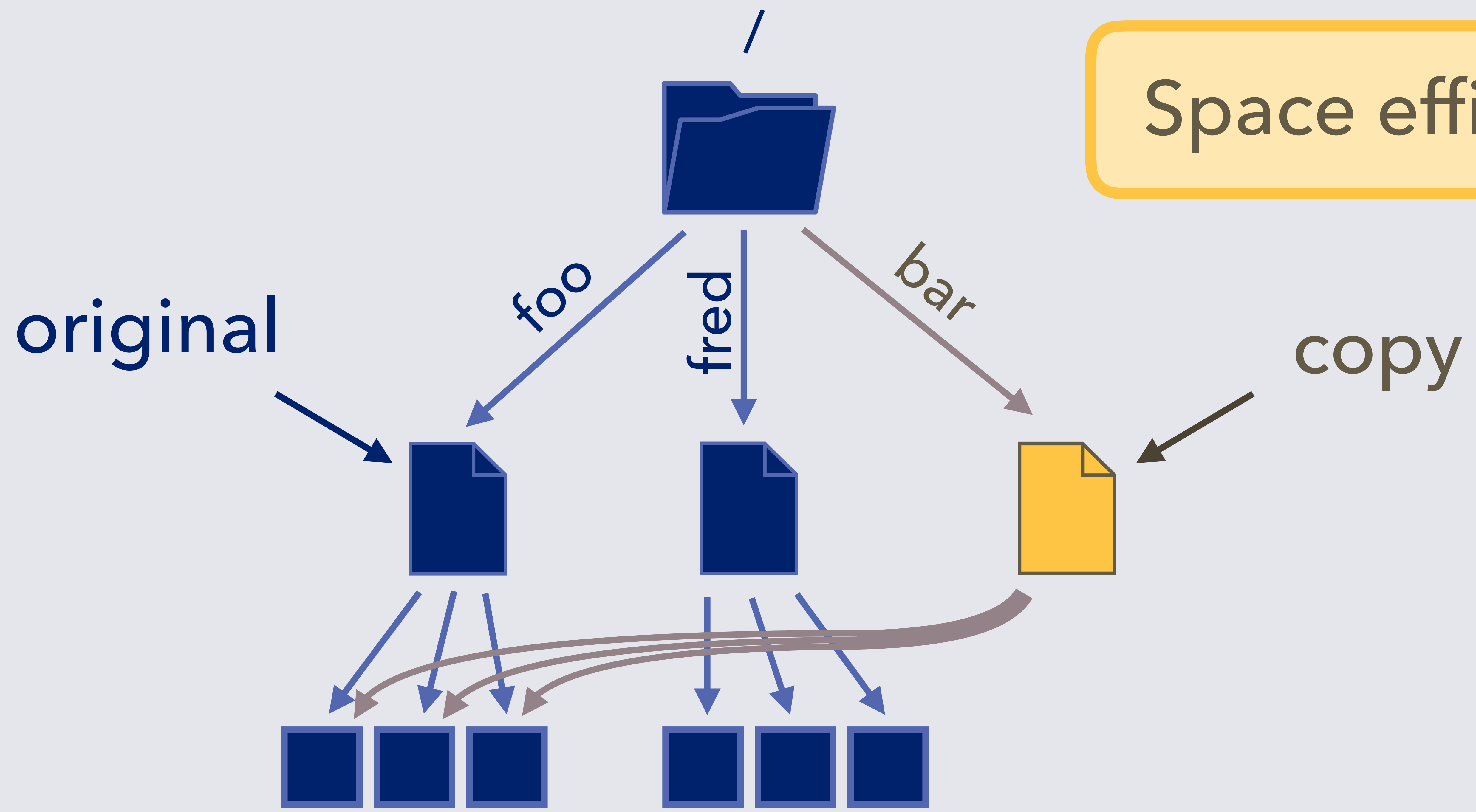
Fast Reads ?

Fast Writes ?



Logical Copy in an Inode File System

Copy /foo to /bar



Low latency ✓

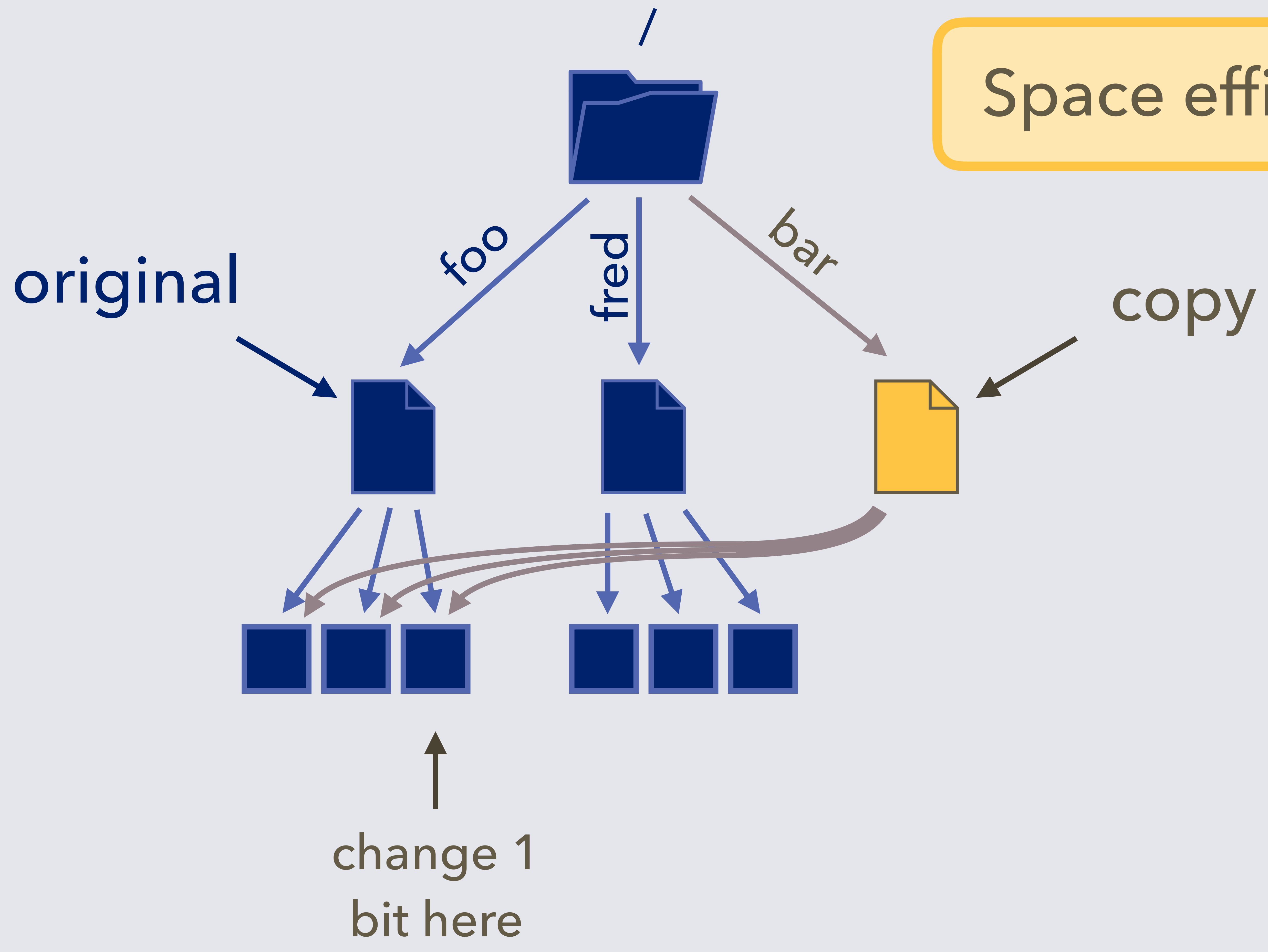
Space efficient ?

Fast Reads ?

Fast Writes ?

Logical Copy in an Inode File System

Copy /foo to /bar



Low latency ✓

Space efficient ?

Fast Reads ?

Fast Writes ?

Logical Copy in an Inode File System

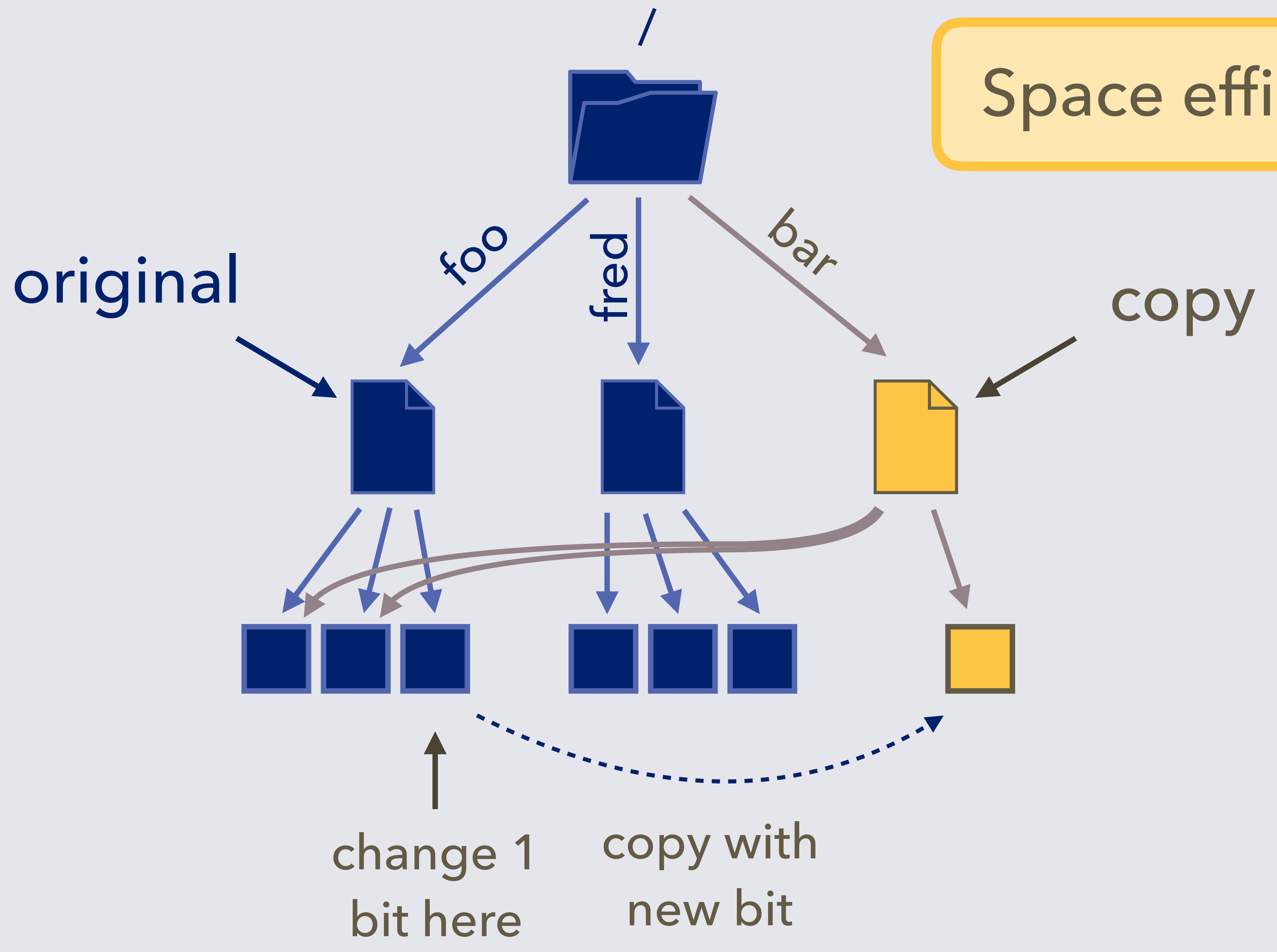
Copy /foo to /bar

Low latency ✓

Space efficient ?

Fast Reads ?

Fast Writes ?



Logical Copy in an Inode File System

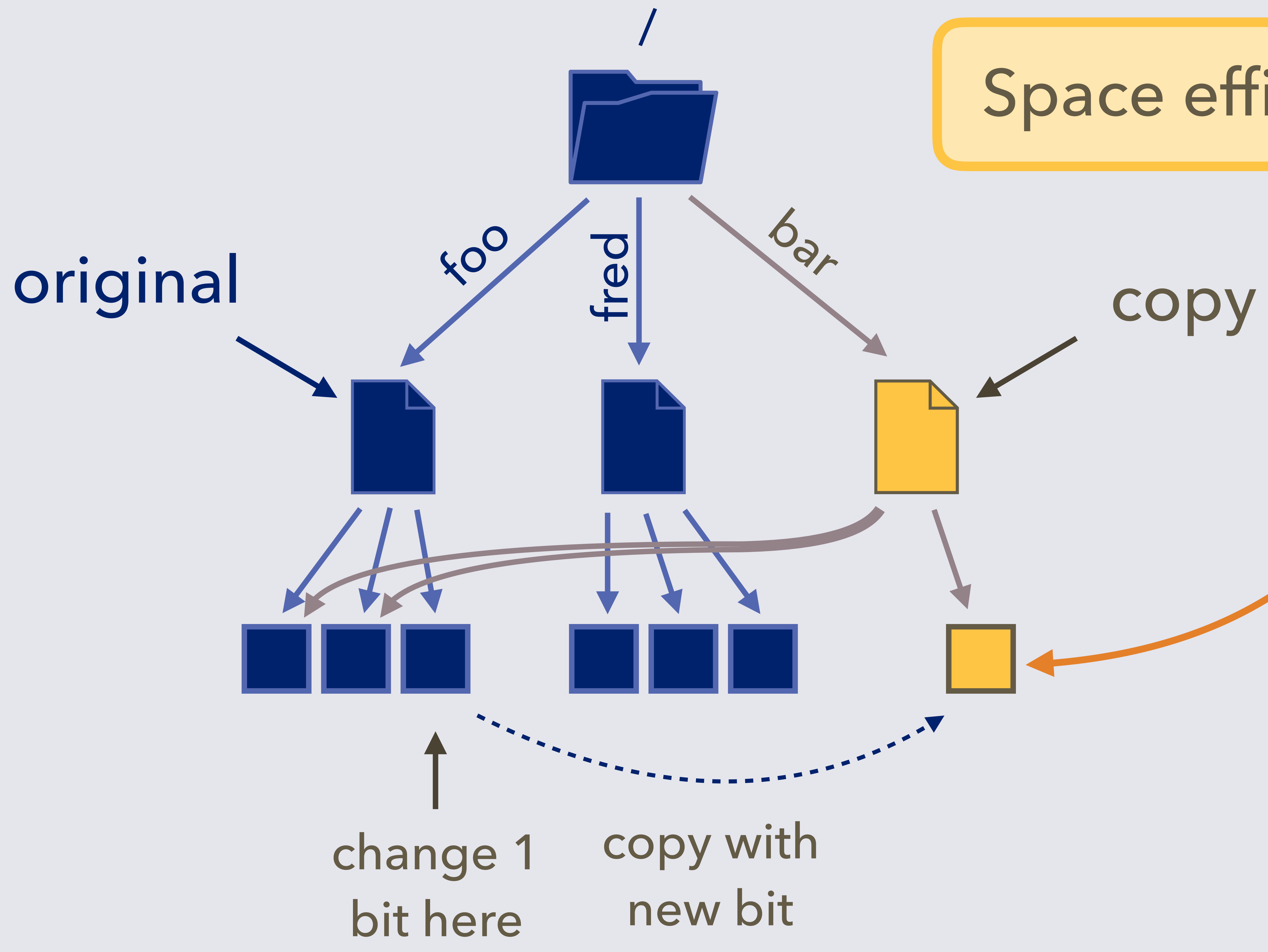
Copy /foo to /bar

Low latency ✓

Space efficient ?

Fast Reads ?

Fast Writes ?



Added a whole data block to change 1 bit

Logical Copy in an Inode File System

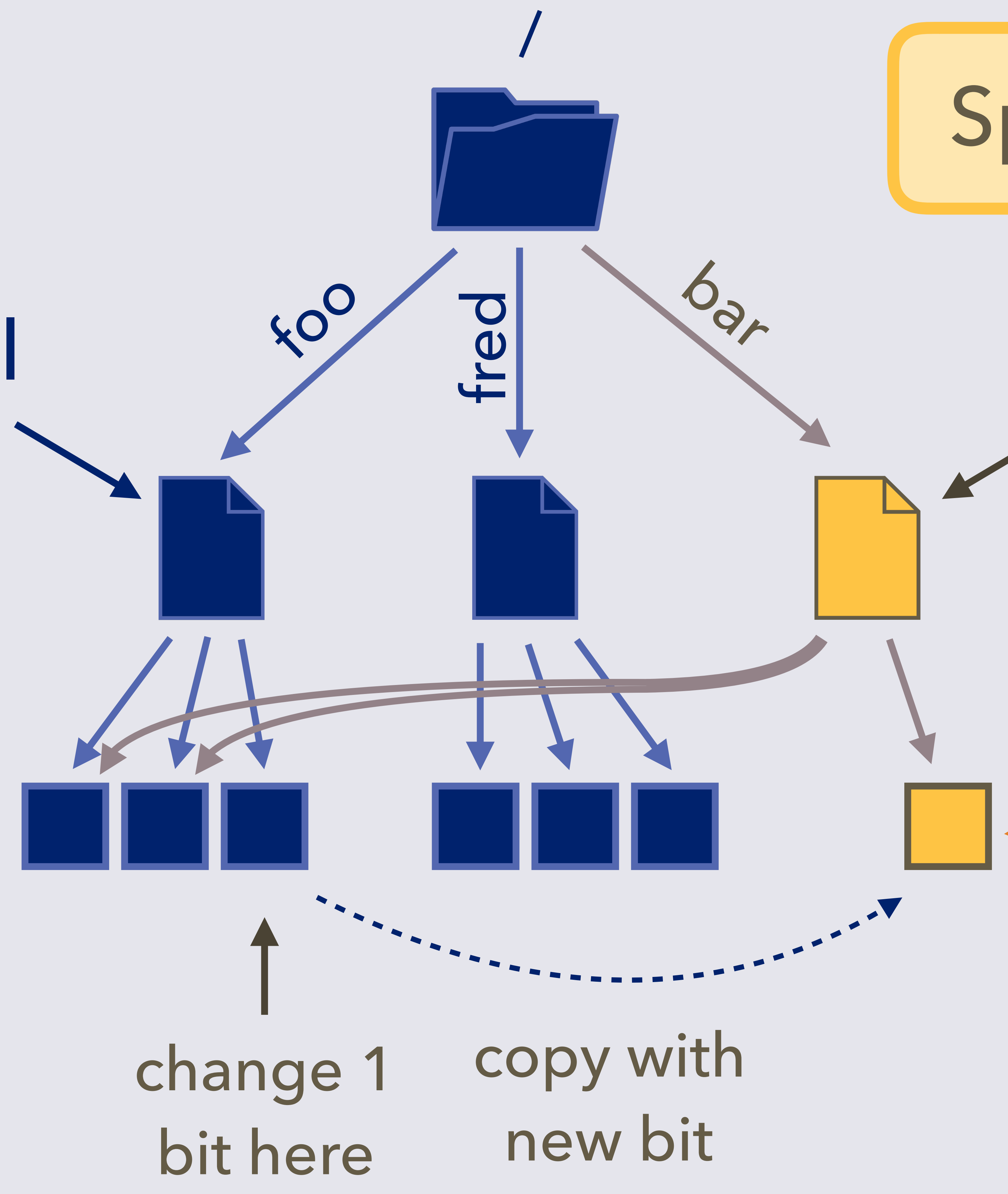
Copy /foo to /bar

Low latency ✓

Space efficient ?

Fast Reads ?

Fast Writes ?



Added a whole data block to change 1 bit

This is at least 4KiB and can be more

Logical Copy in an Inode File System

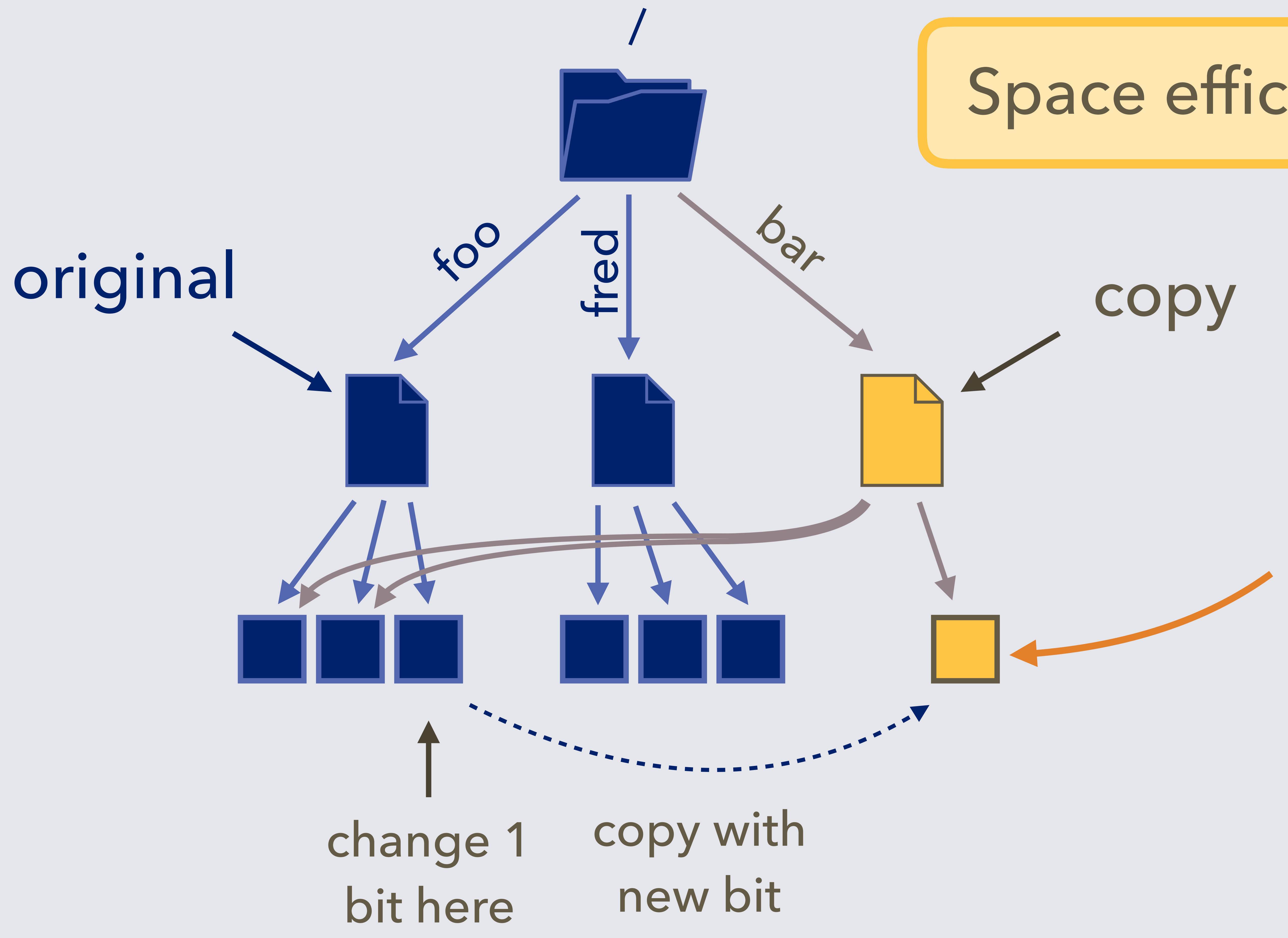
Copy /foo to /bar

Low latency ✓

Space efficient ✗

Fast Reads ?

Fast Writes ?

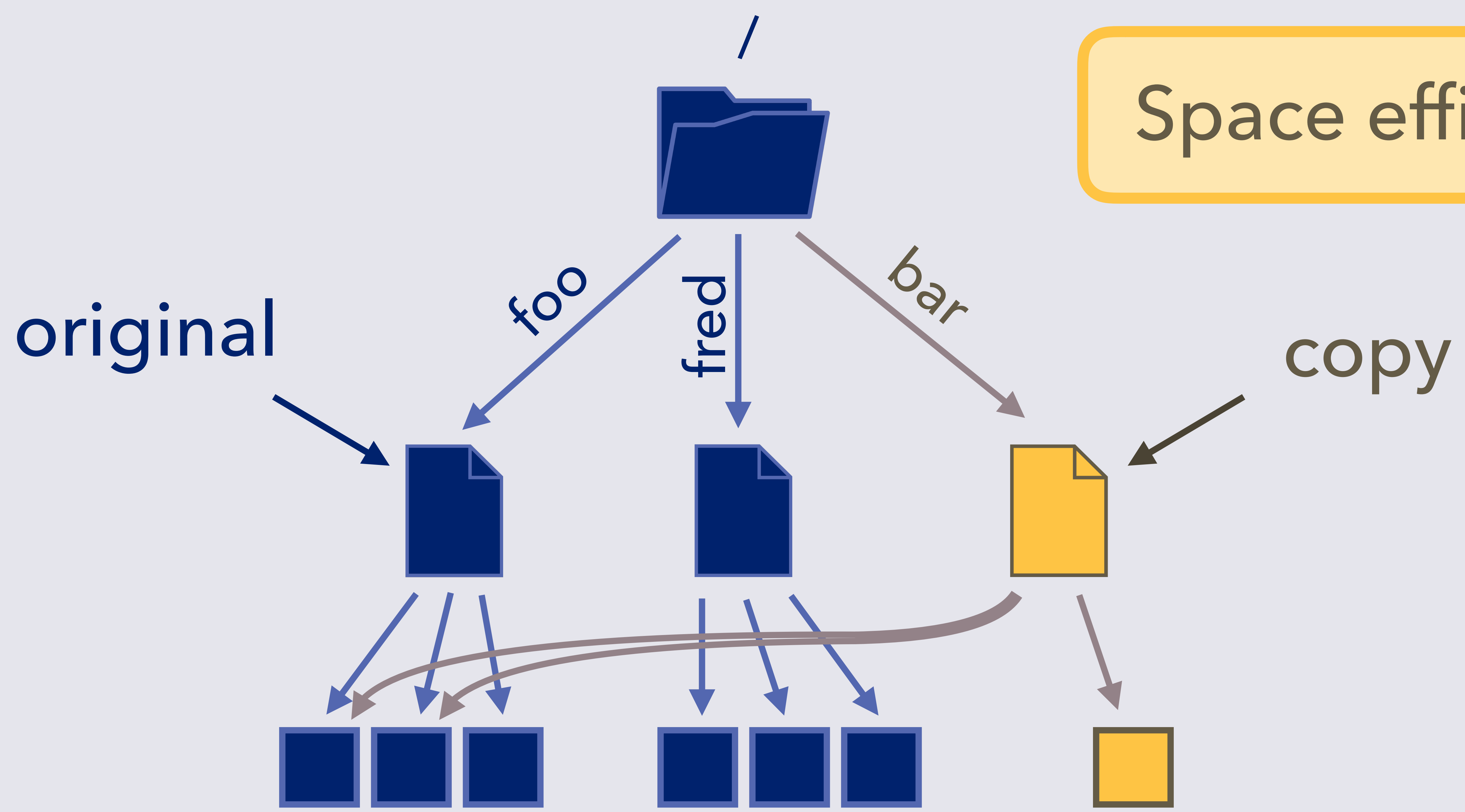


Added a whole data block to change 1 bit

This is at least 4KiB and can be more

Logical Copy in an Inode File System

Copy /foo to /bar



Low latency ✓

Space efficient ✗

Fast Reads ?

Fast Writes ?

Logical Copy in an Inode File System

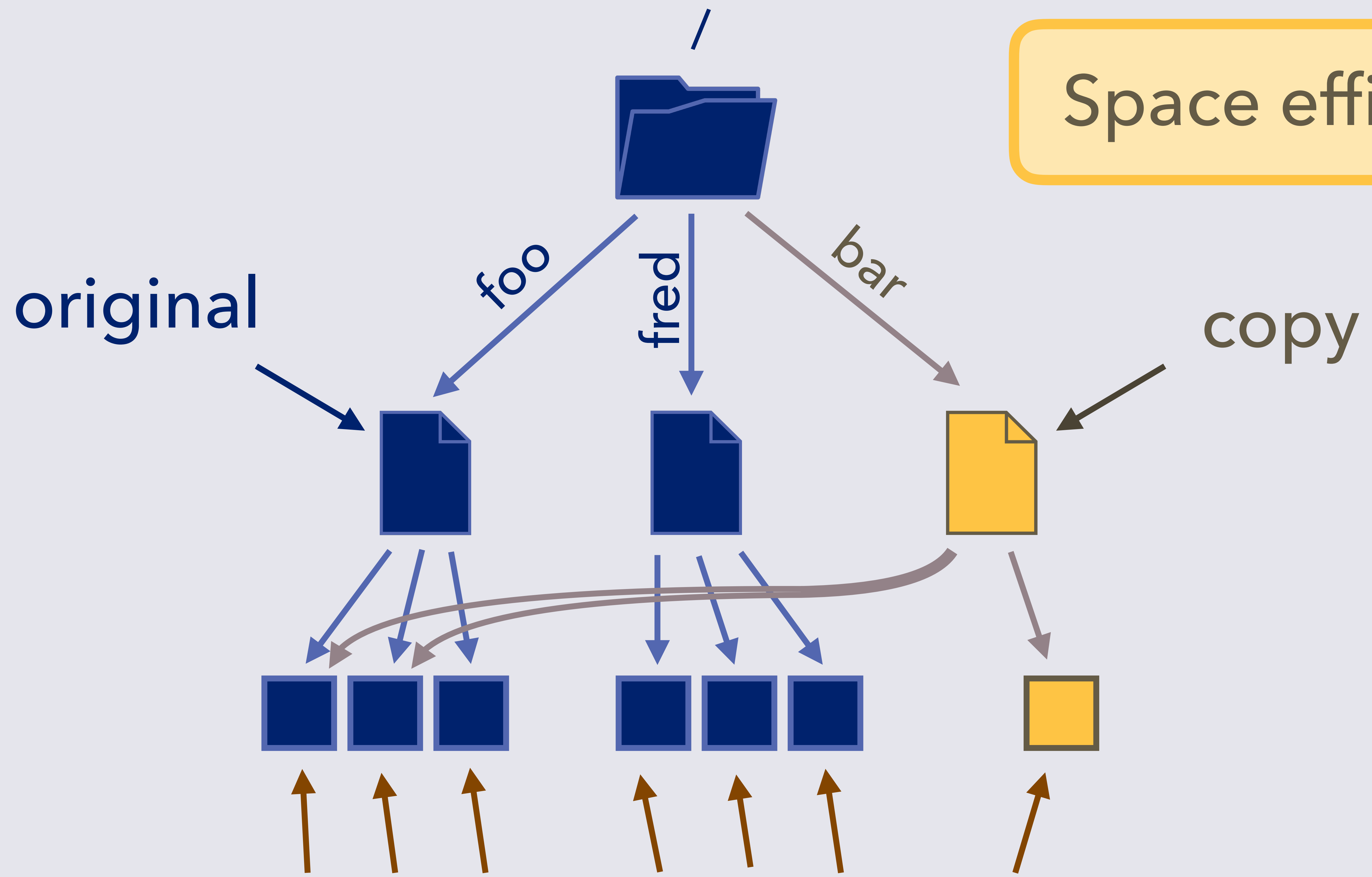
Copy /foo to /bar

Low latency ✓

Space efficient ✗

Fast Reads ?

Fast Writes ?



no locality guarantees between data blocks

Logical Copy in an Inode File System

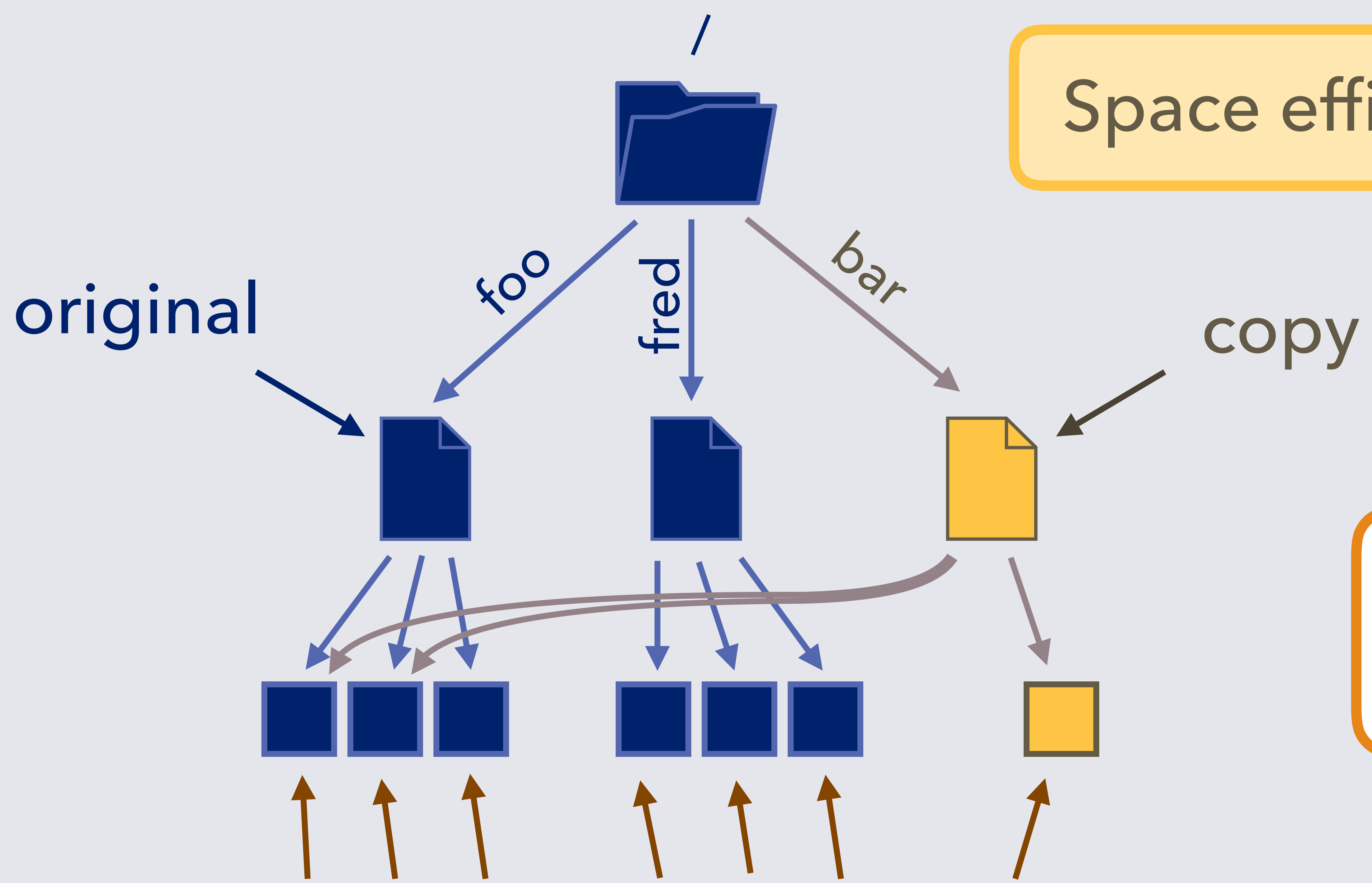
Copy /foo to /bar

Low latency ✓

Space efficient ✗

Fast Reads ?

Fast Writes ?



only have locality if the blocks are large

no locality guarantees between data blocks

Logical Copy in an Inode File System

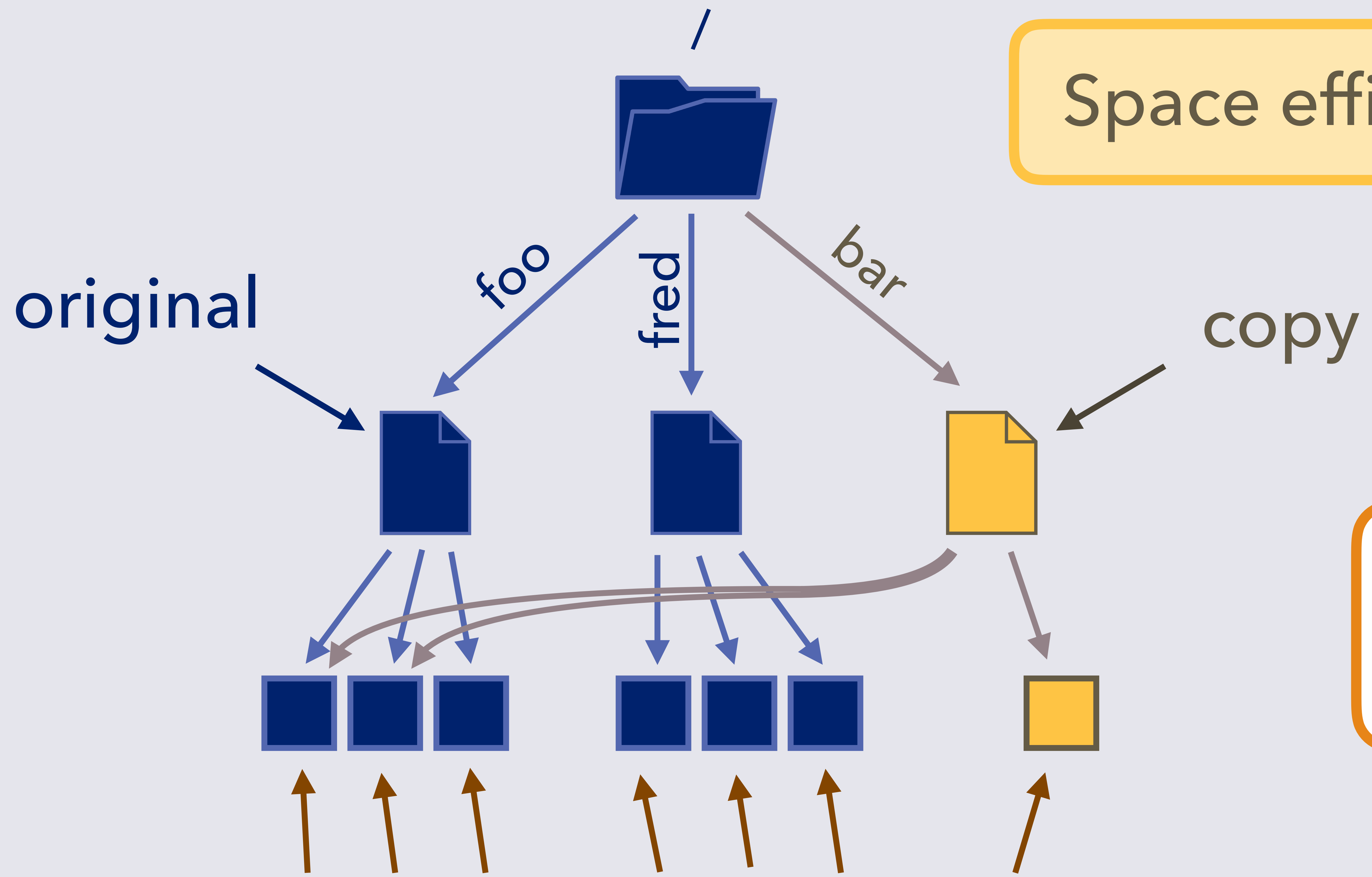
Copy /foo to /bar

Low latency ✓

Space efficient ✗

Fast Reads ?

Fast Writes ?



usually 4KiB ⇒ too small for locality

only have locality if the blocks are large

no locality guarantees between data blocks

Logical Copy in an Inode File System

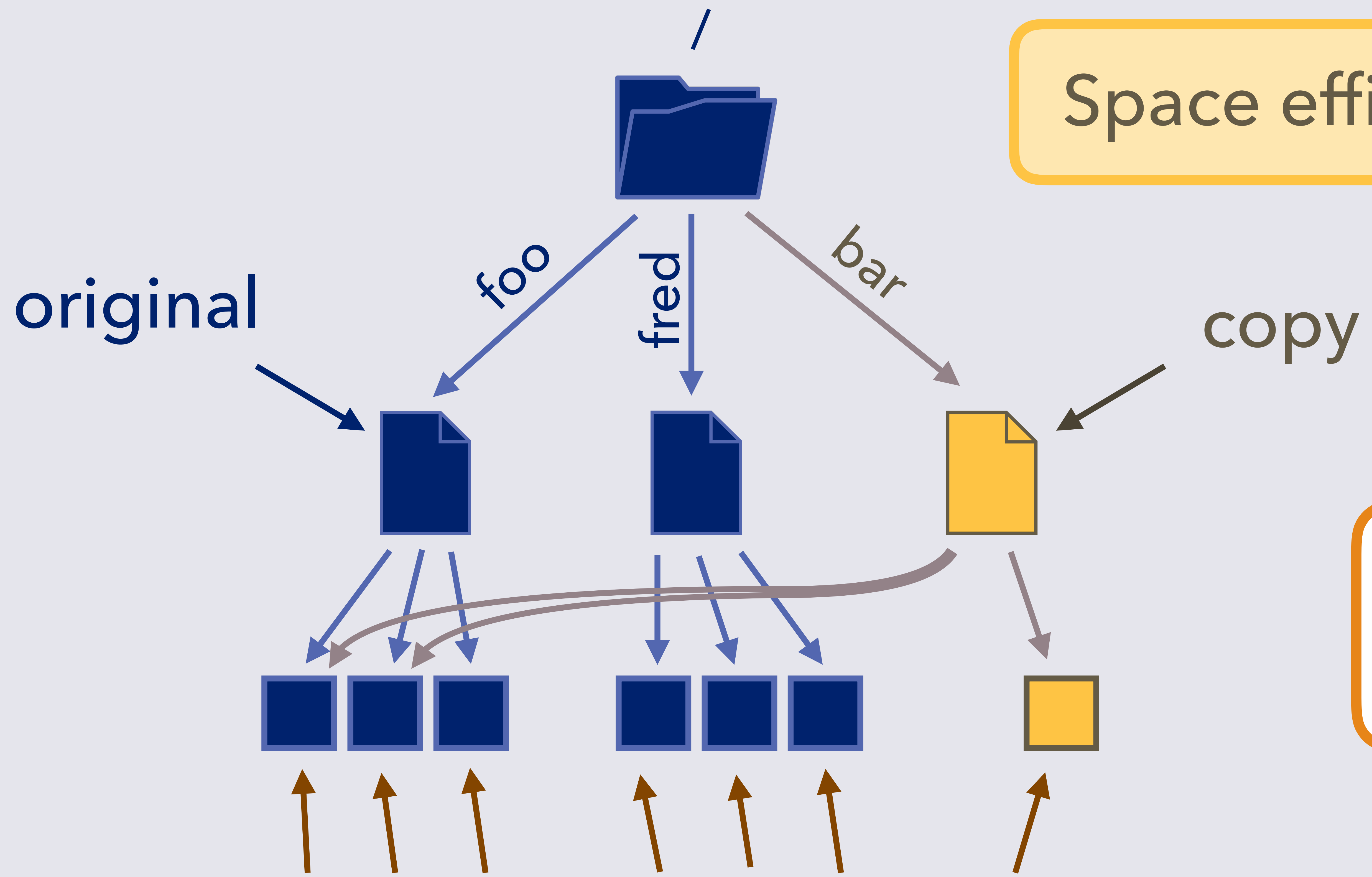
Copy /foo to /bar

Low latency ✓

Space efficient ✗

Fast Reads ✗

Fast Writes ✗



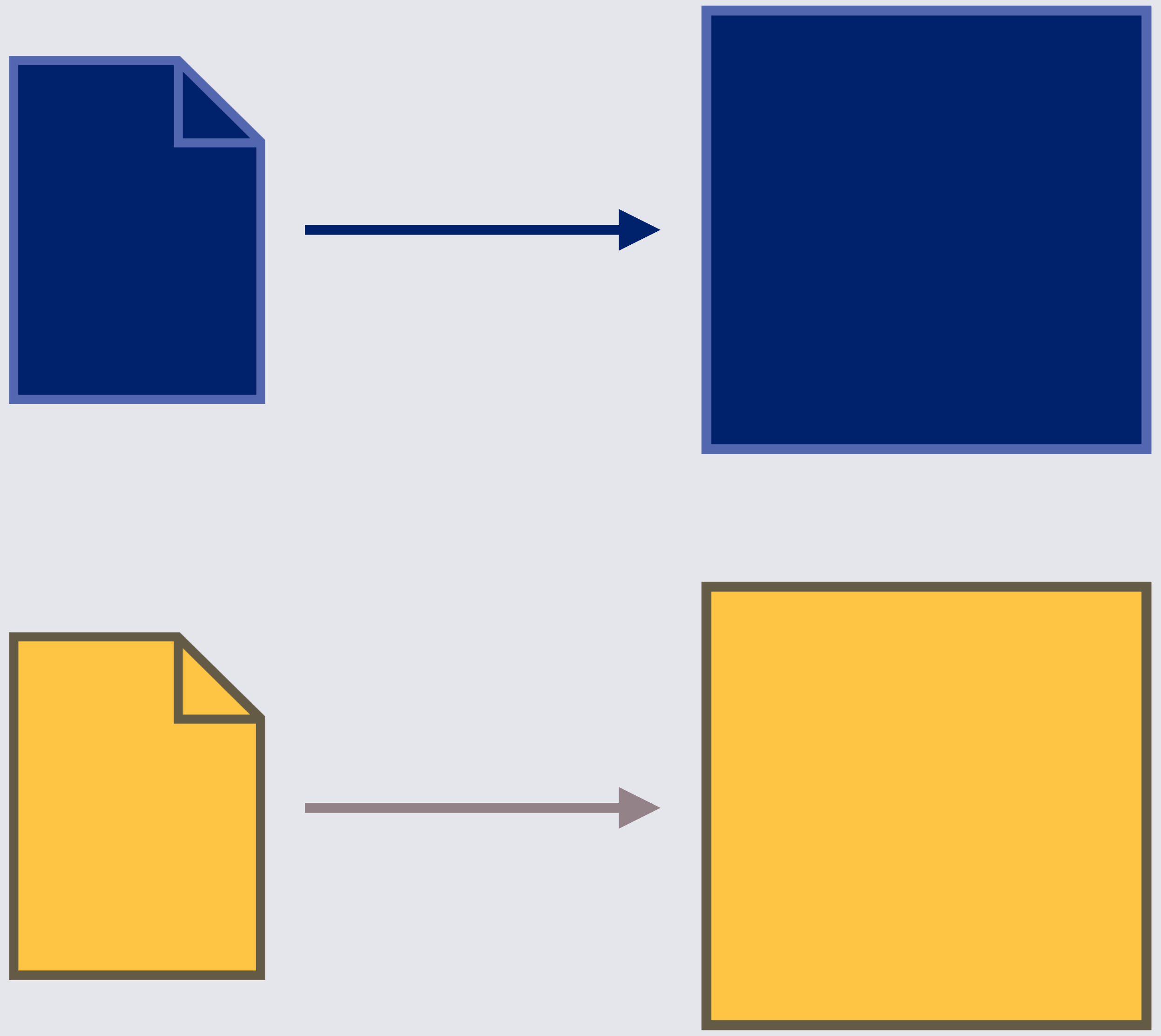
usually 4KiB ⇒ too small for locality

only have locality if the blocks are large

no locality guarantees between data blocks

Space-Locality Tradeoff

Larger blocks

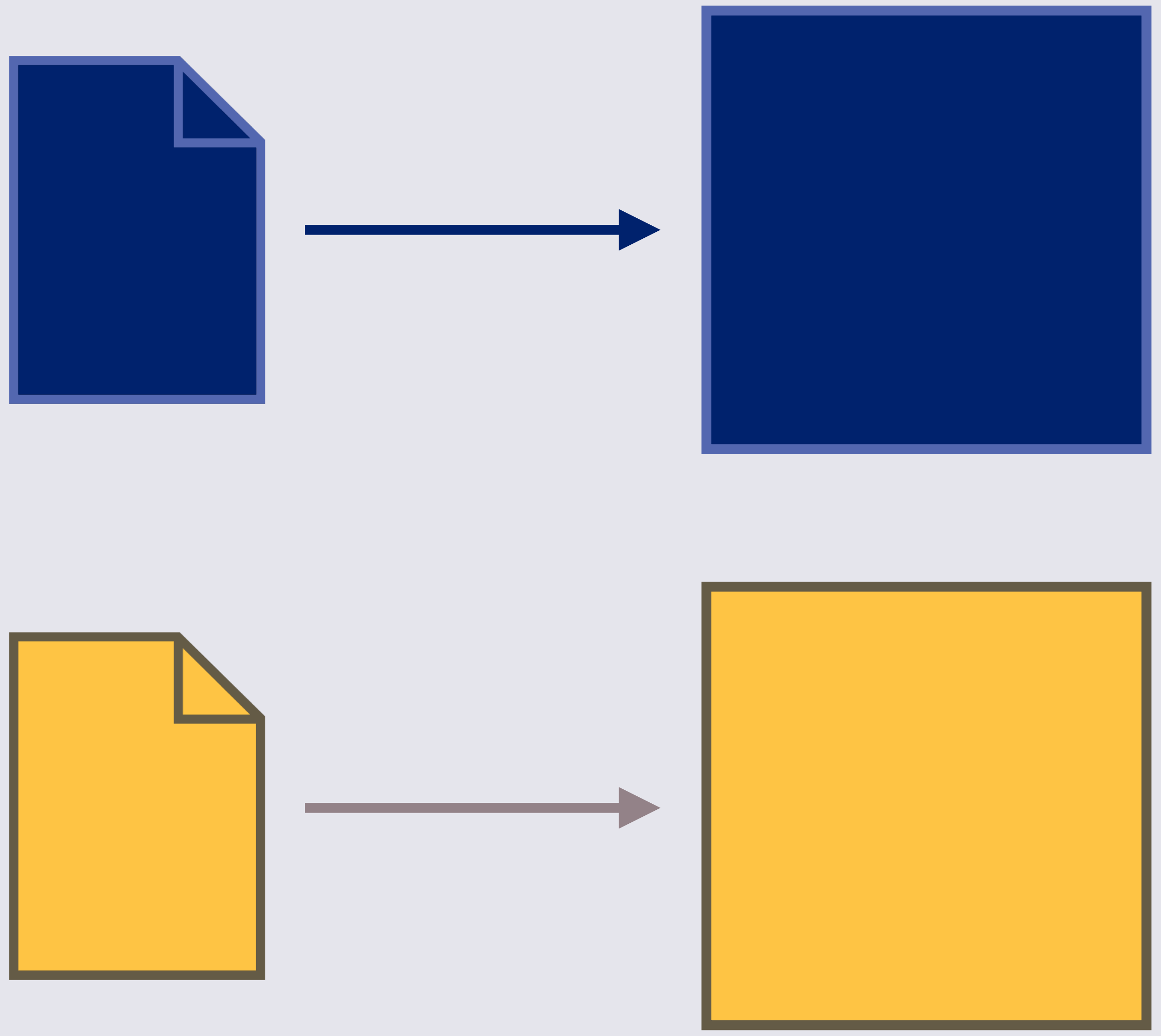


Better locality 😄

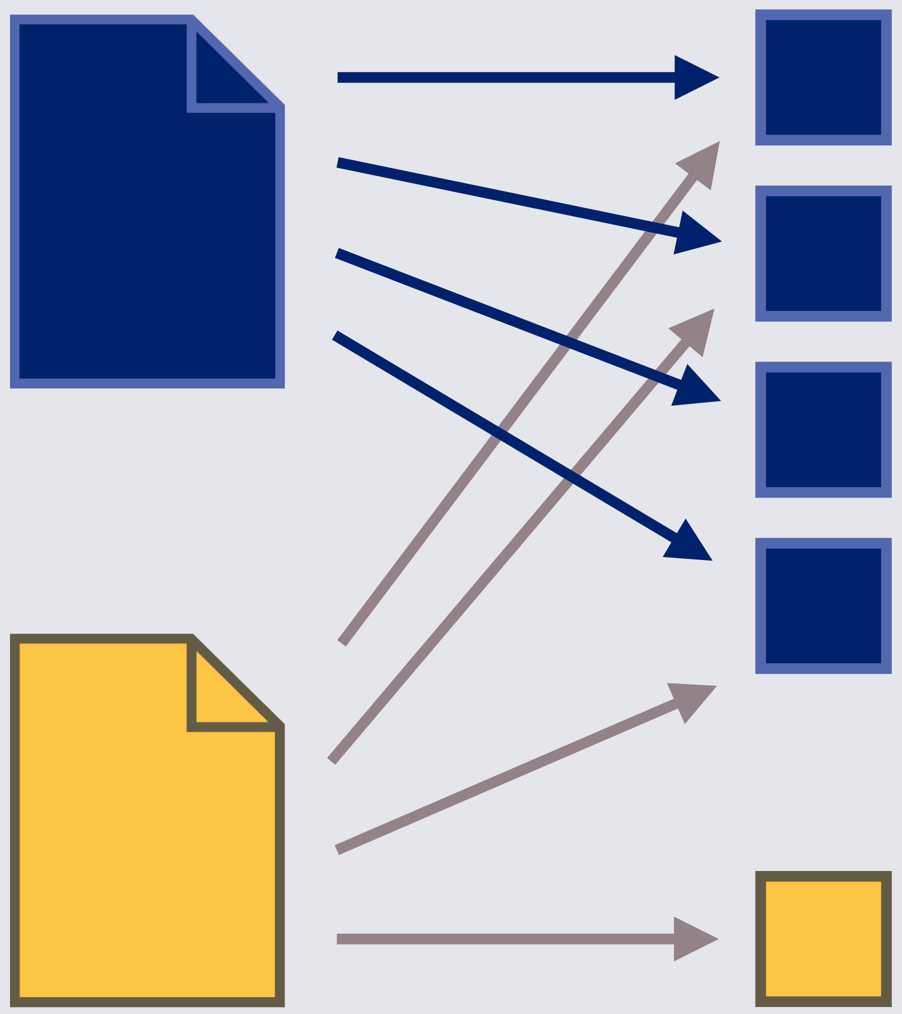
Worse space efficiency 😬

Space-Locality Tradeoff

Larger blocks



Smaller blocks



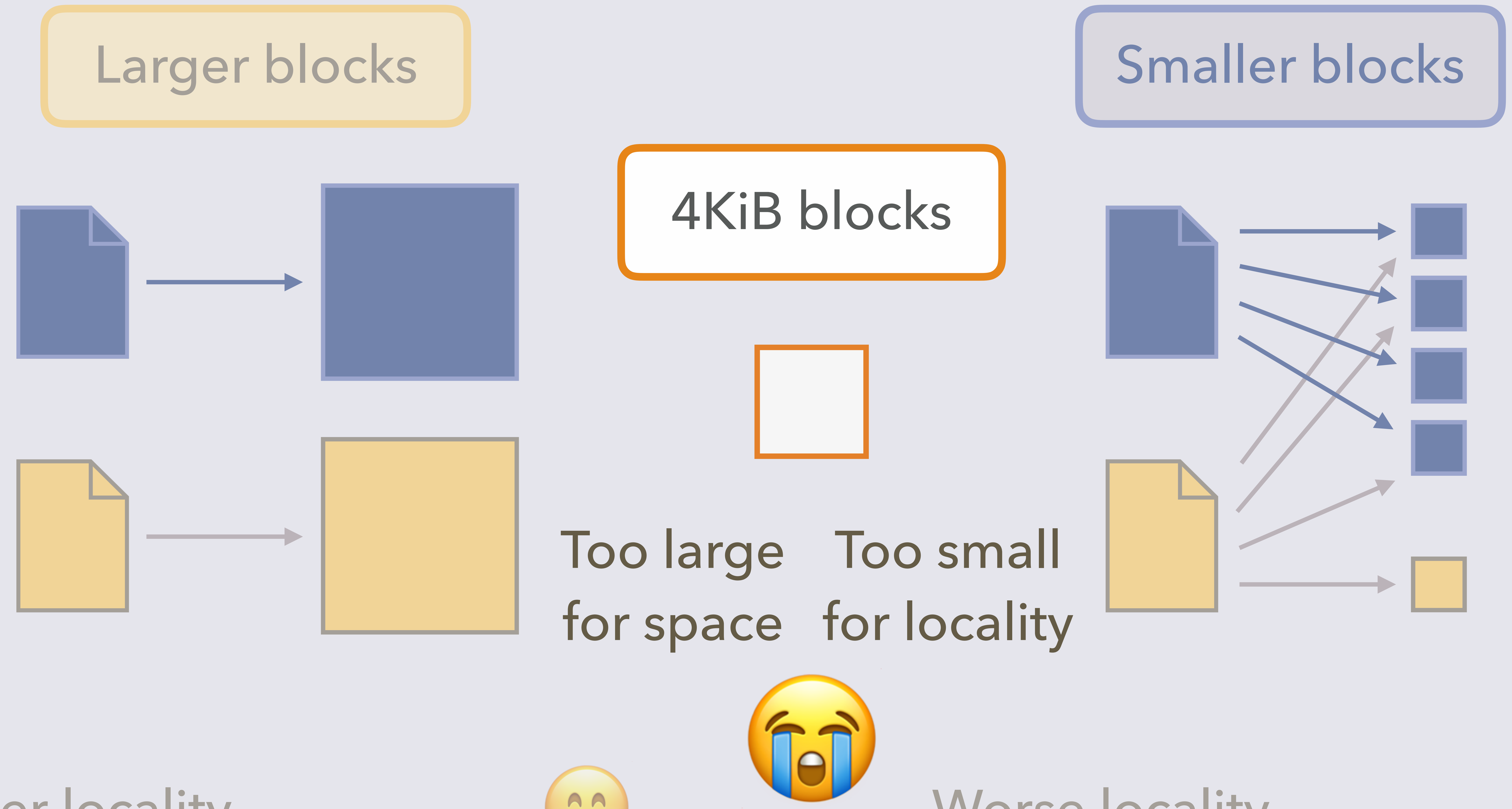
Better locality 😄

Worse space efficiency 😬

Worse locality 😬

Better space efficiency 😄

Space-Locality Tradeoff



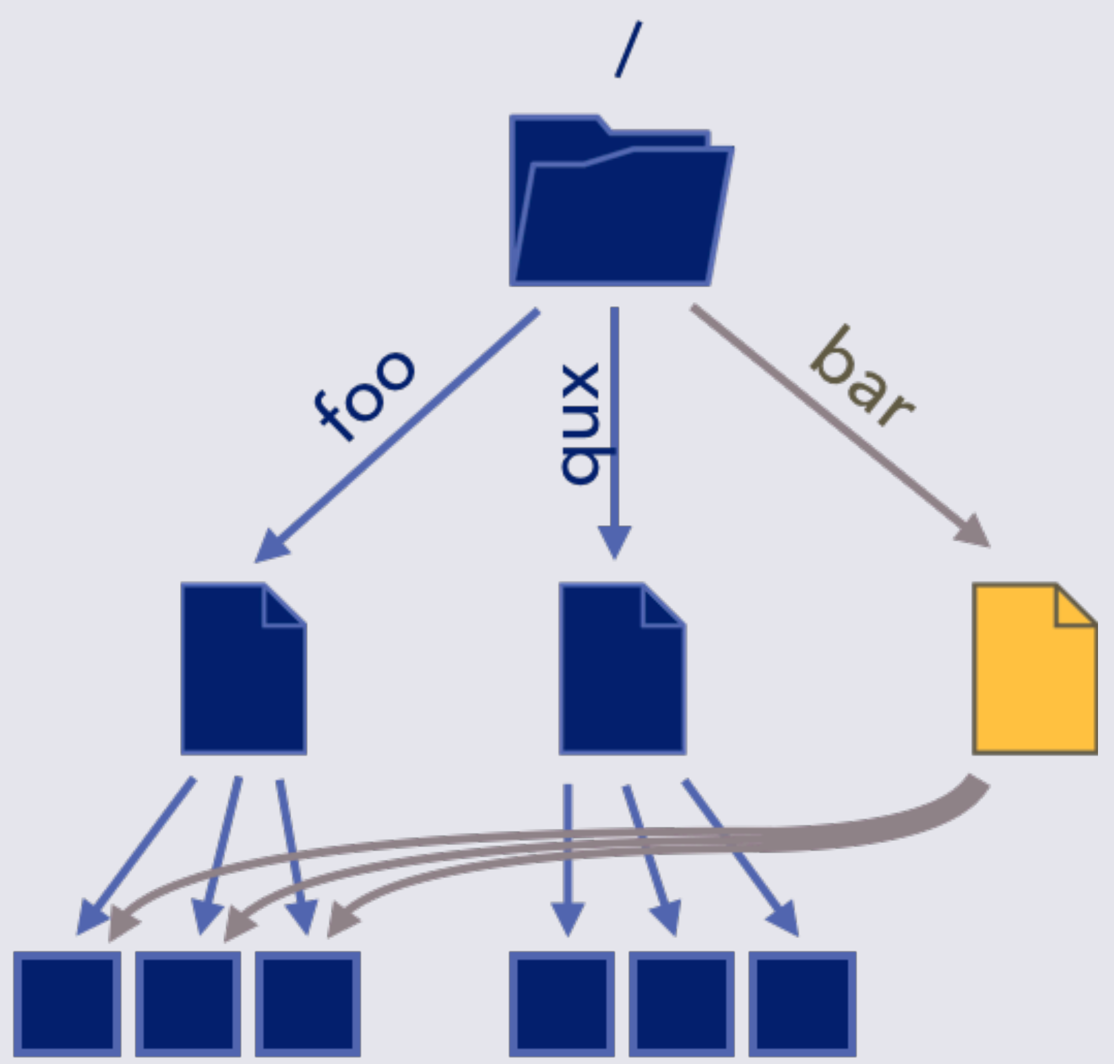
Better locality 😄

Worse space efficiency 😬

Worse locality 😬

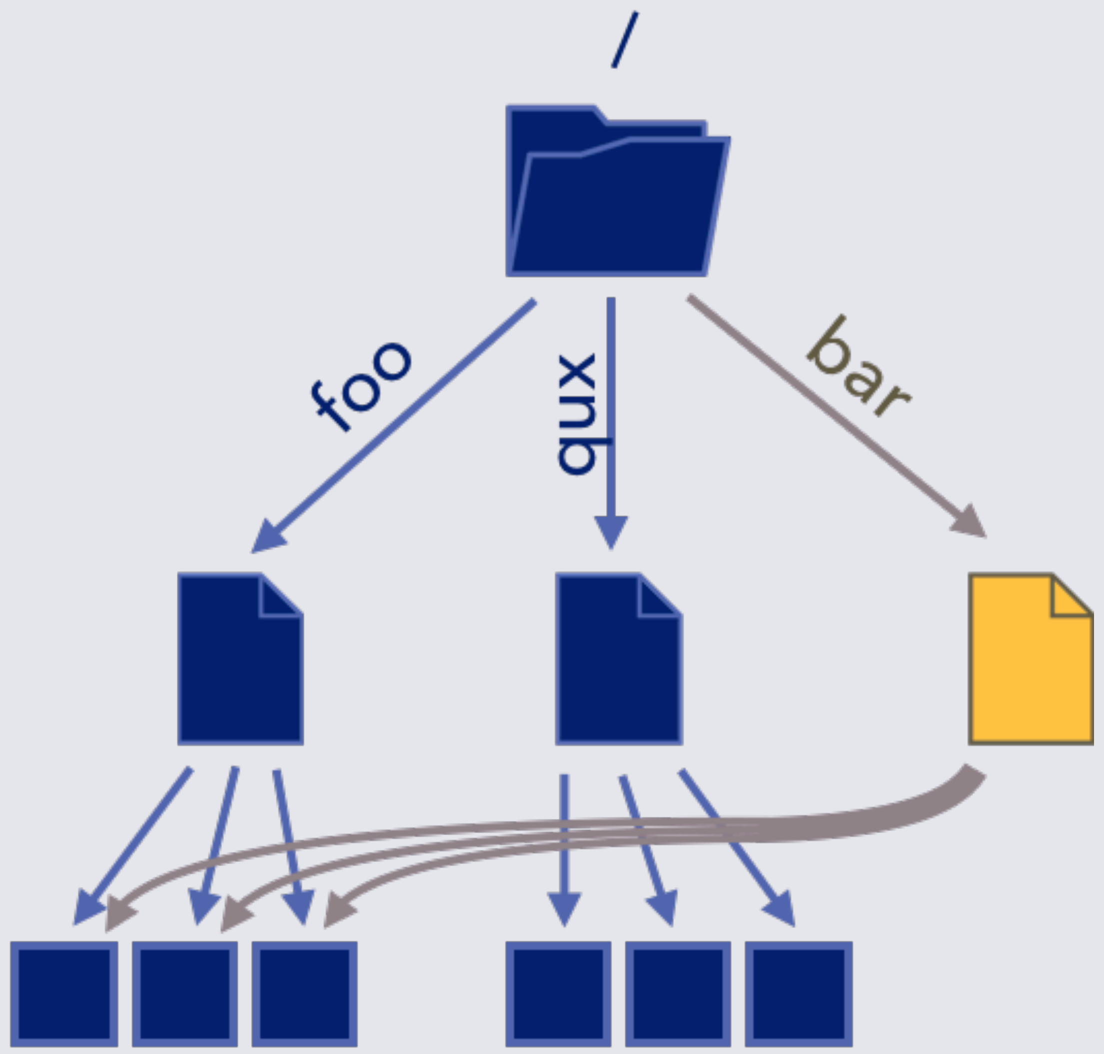
Better space efficiency 😄

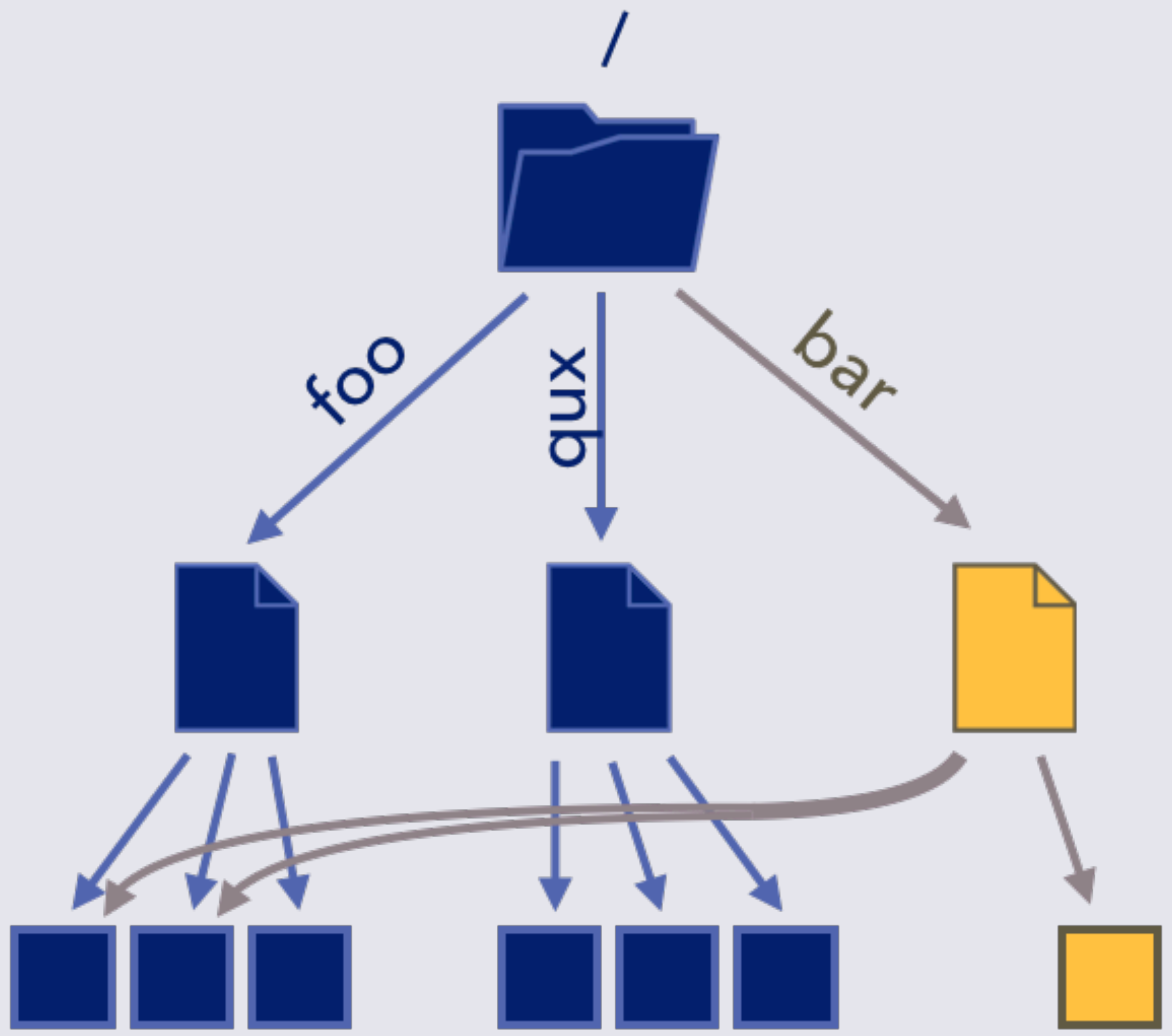
Inode Logical Copy Takeaway



Using a DAG to share data is great for latency

Inode Logical Copy Takeaway

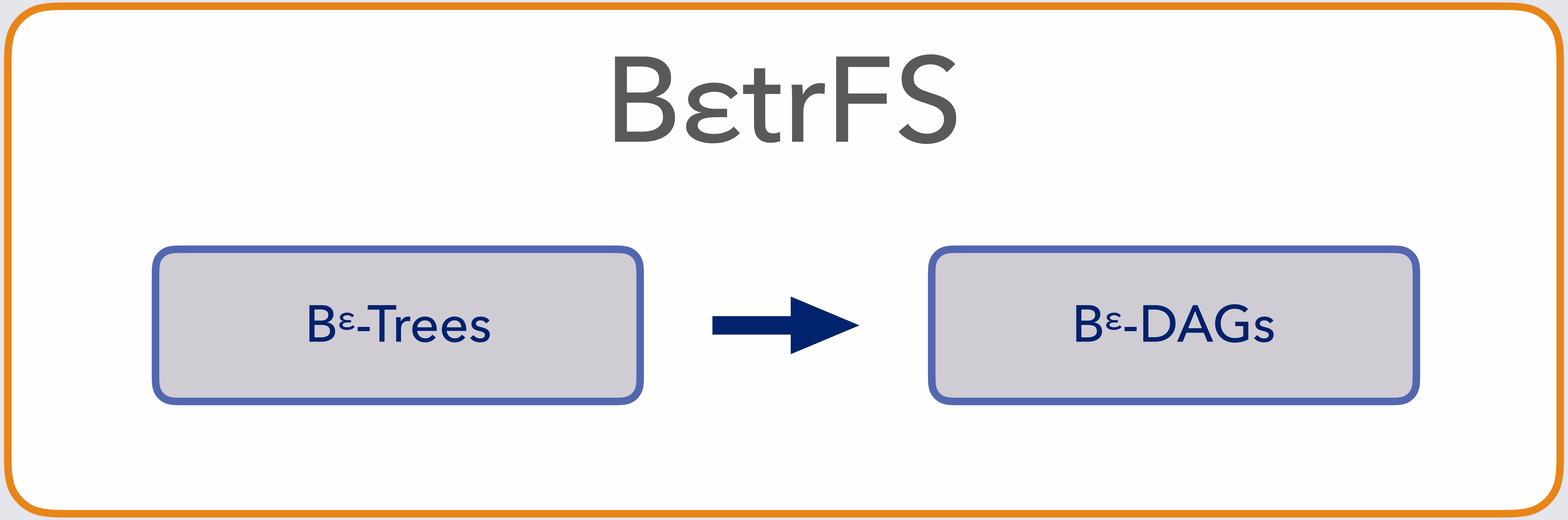




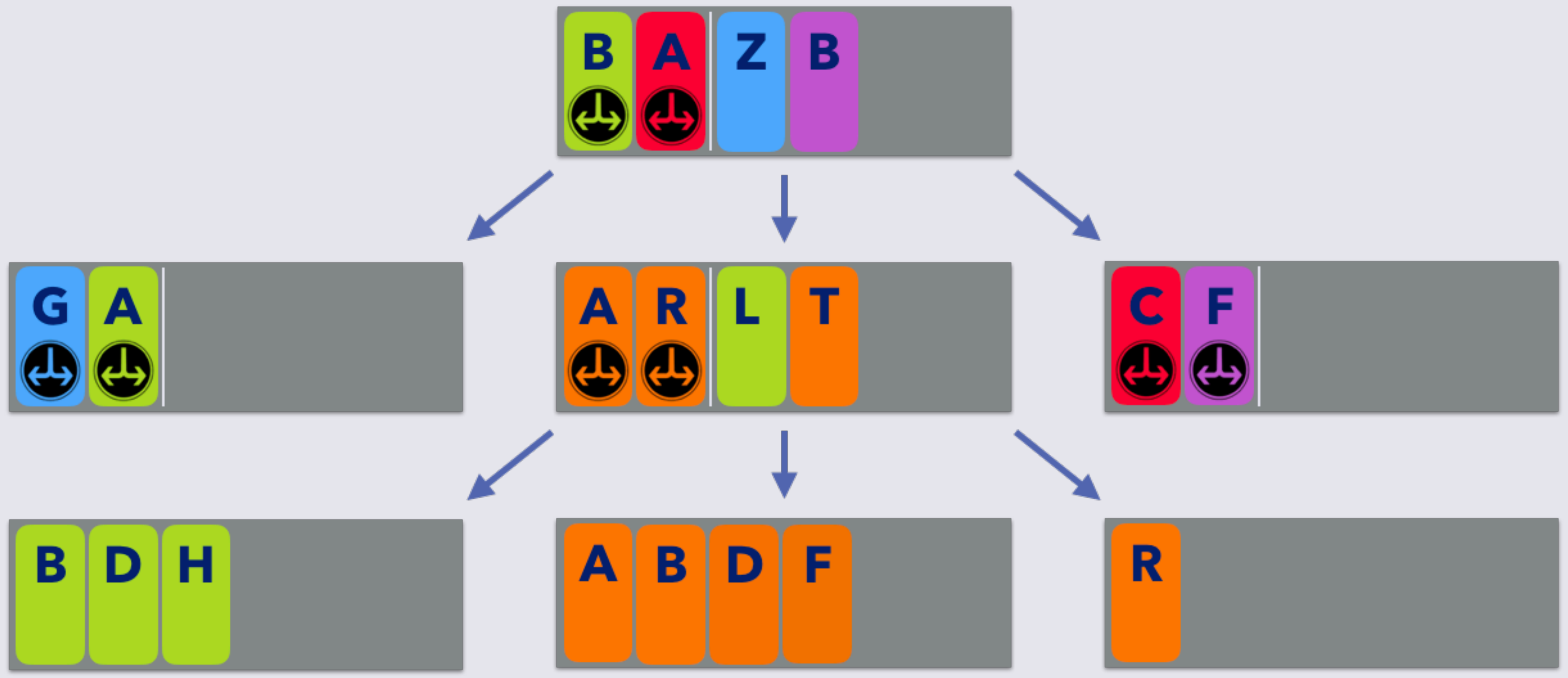
Using a DAG to share data is great for latency

Challenge:
Small writes break sharing

Our Solution: Bε-DAGs

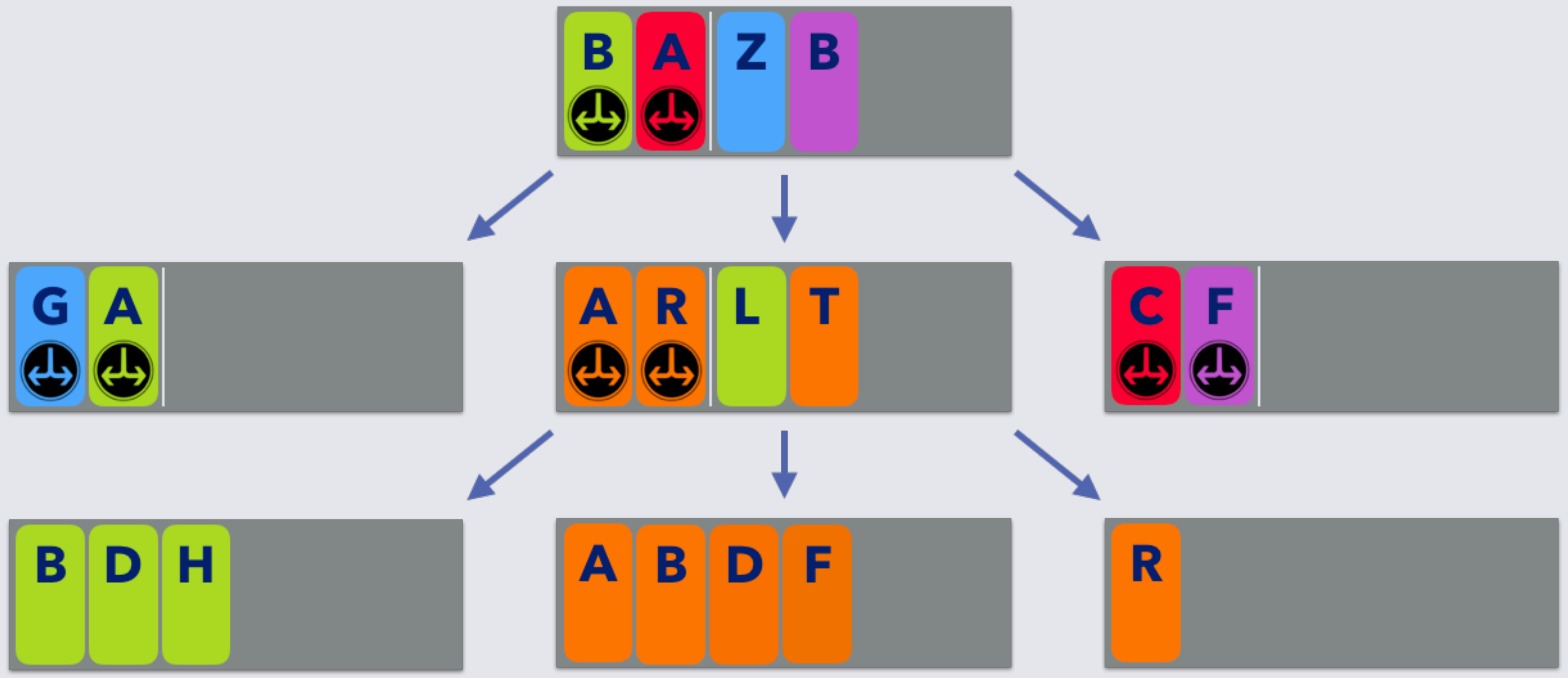


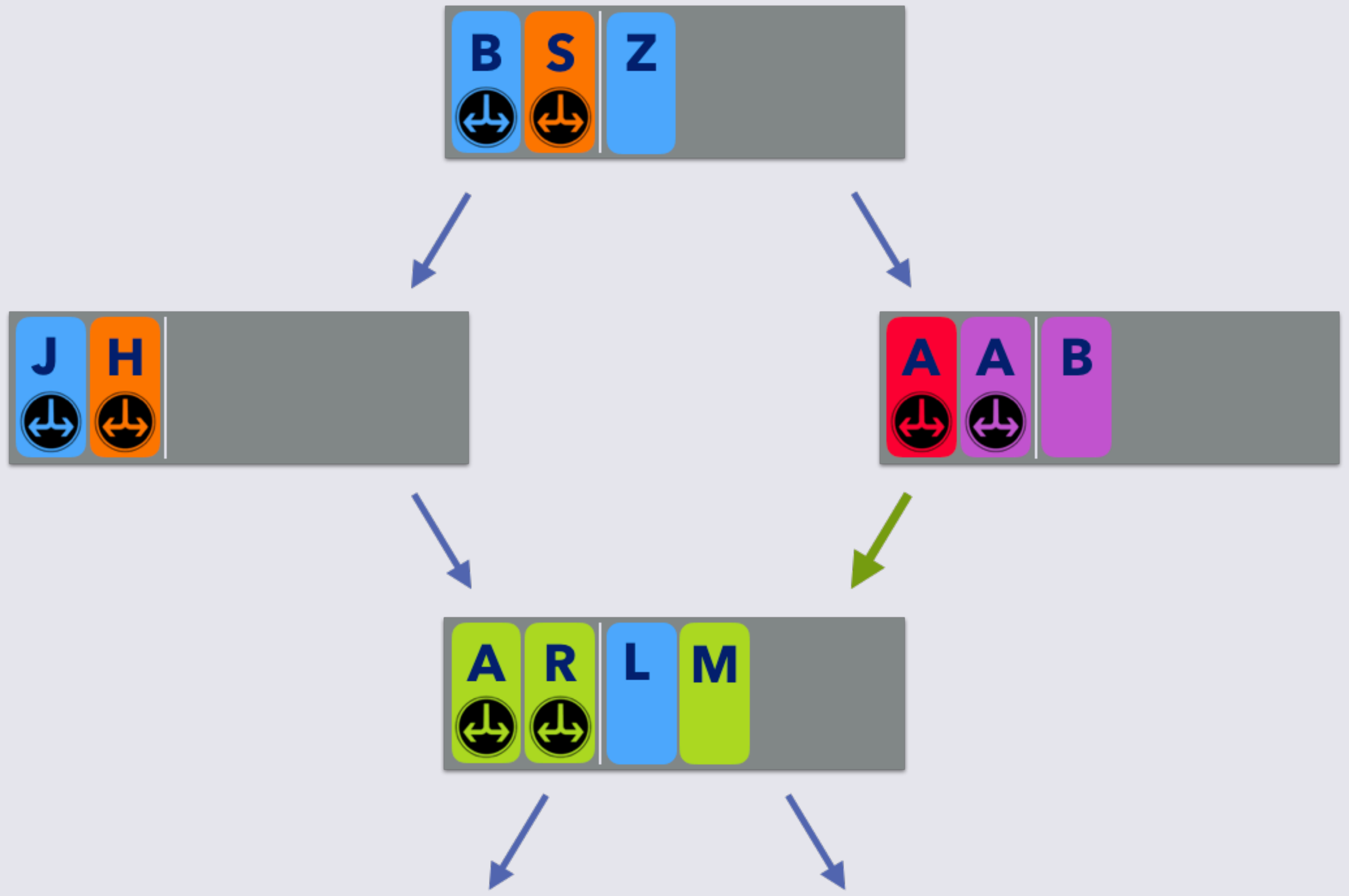
Our Solution: B$^\varepsilon$-DAGs



B$^\varepsilon$-trees have good locality and batch together small writes

Our Solution: B$^\varepsilon$-DAGs





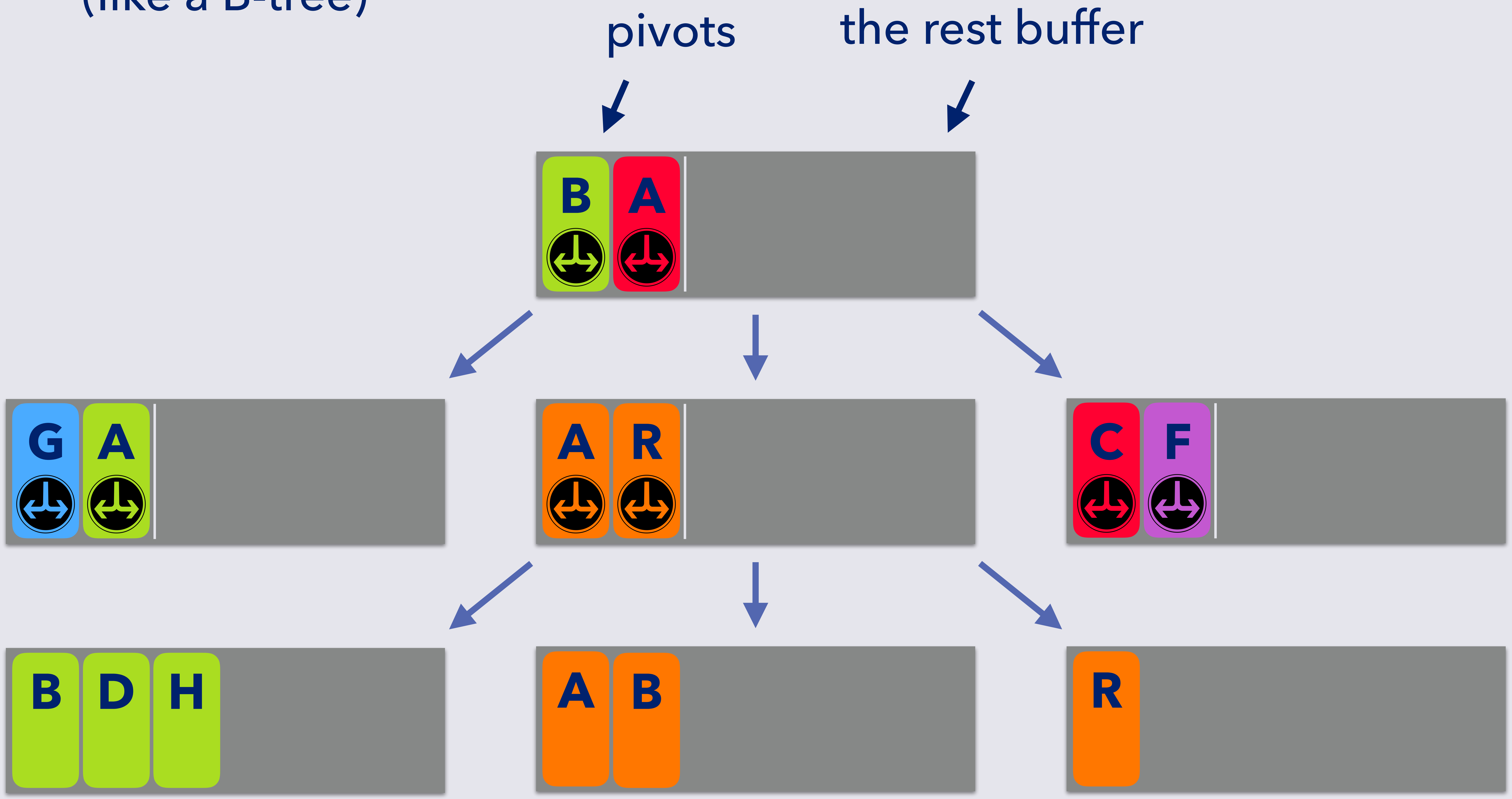
B$^\varepsilon$-trees have good locality and batch together small writes

In this paper, we turn B$^\varepsilon$-trees into B$^\varepsilon$-DAGs to share data between files

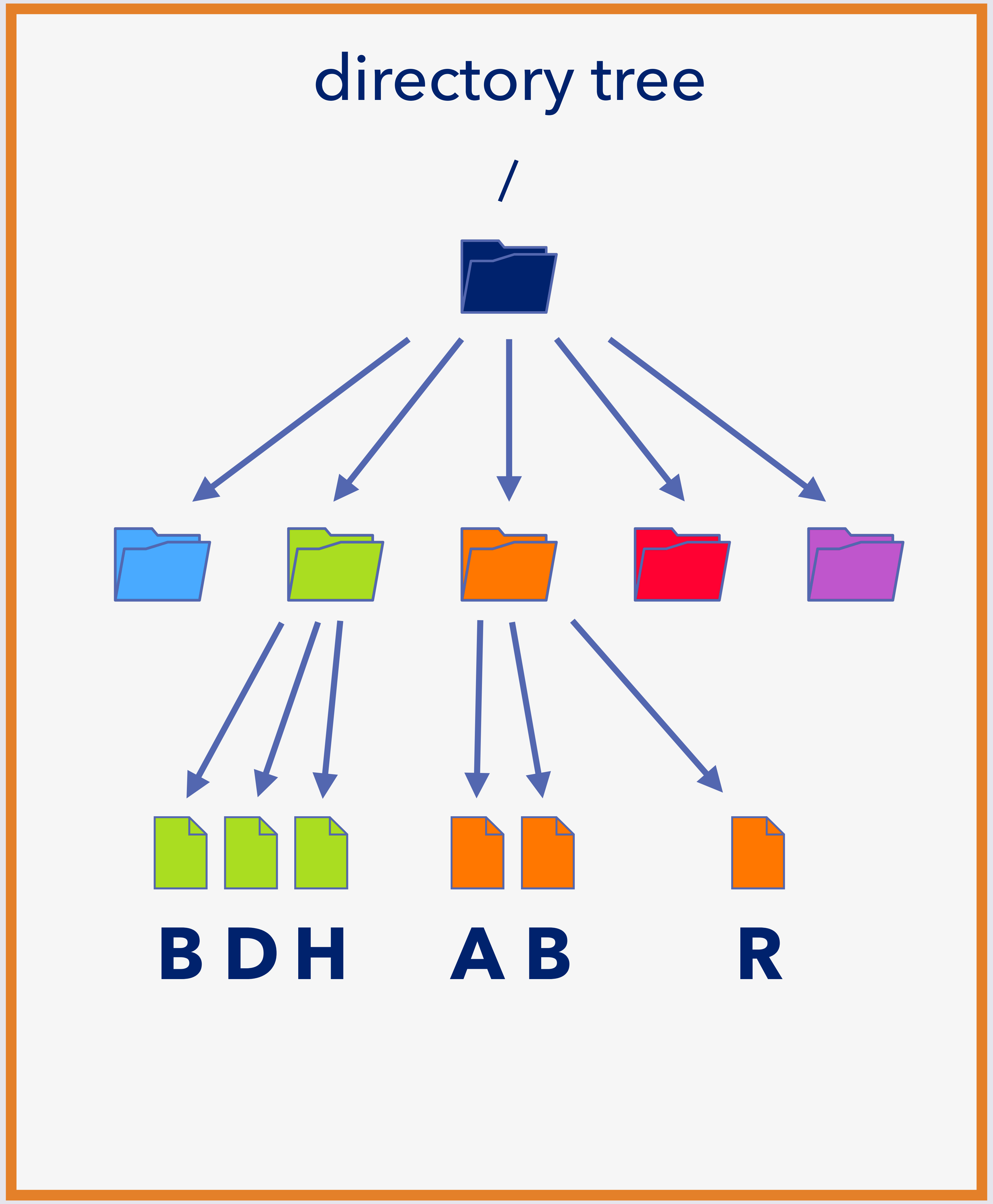
B^{ε}-Trees

B$^{\varepsilon}$-Trees

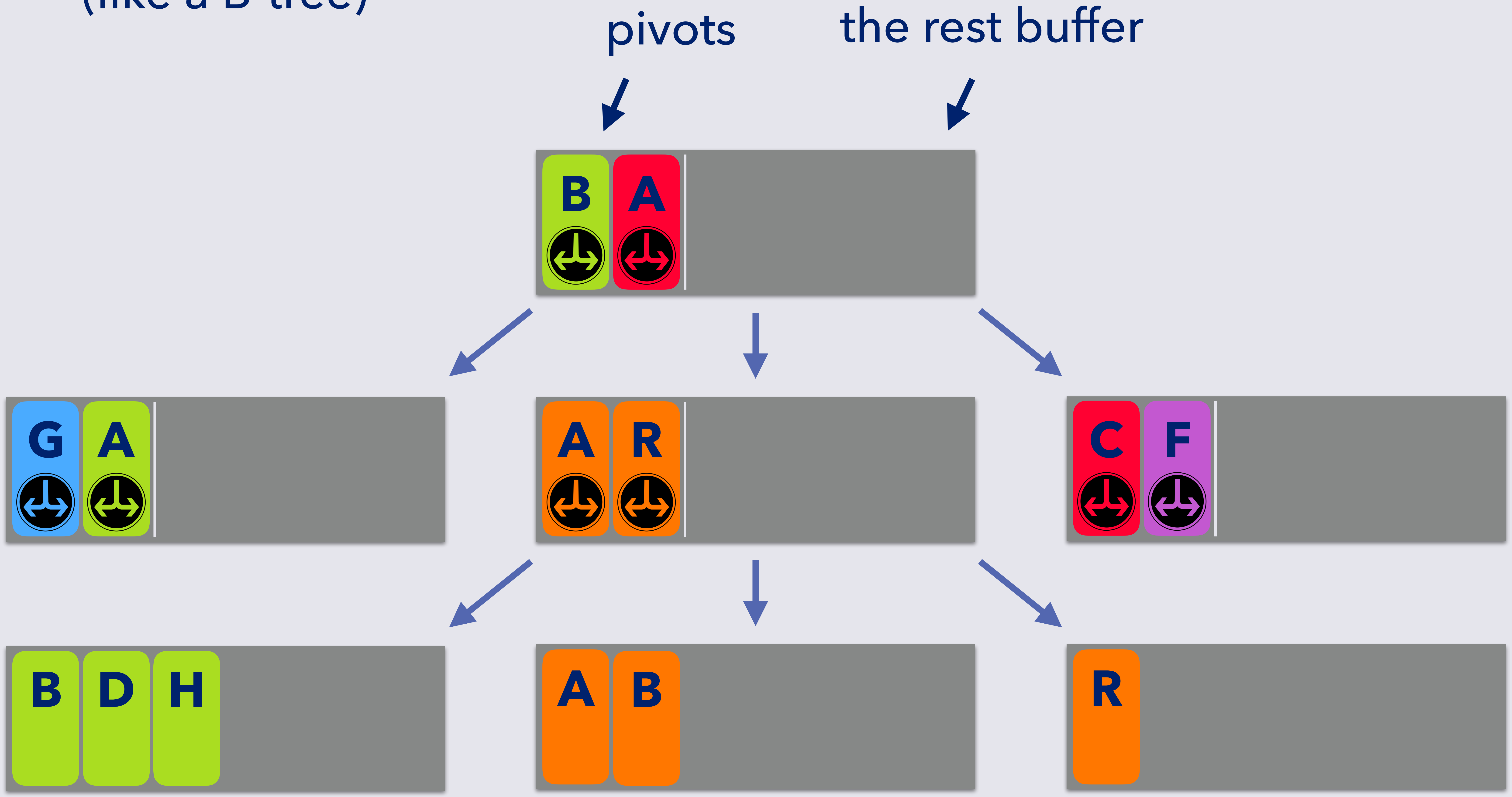
A B$^{\varepsilon}$-tree is a search tree
(like a B-tree)



B$^{\varepsilon}$-Trees

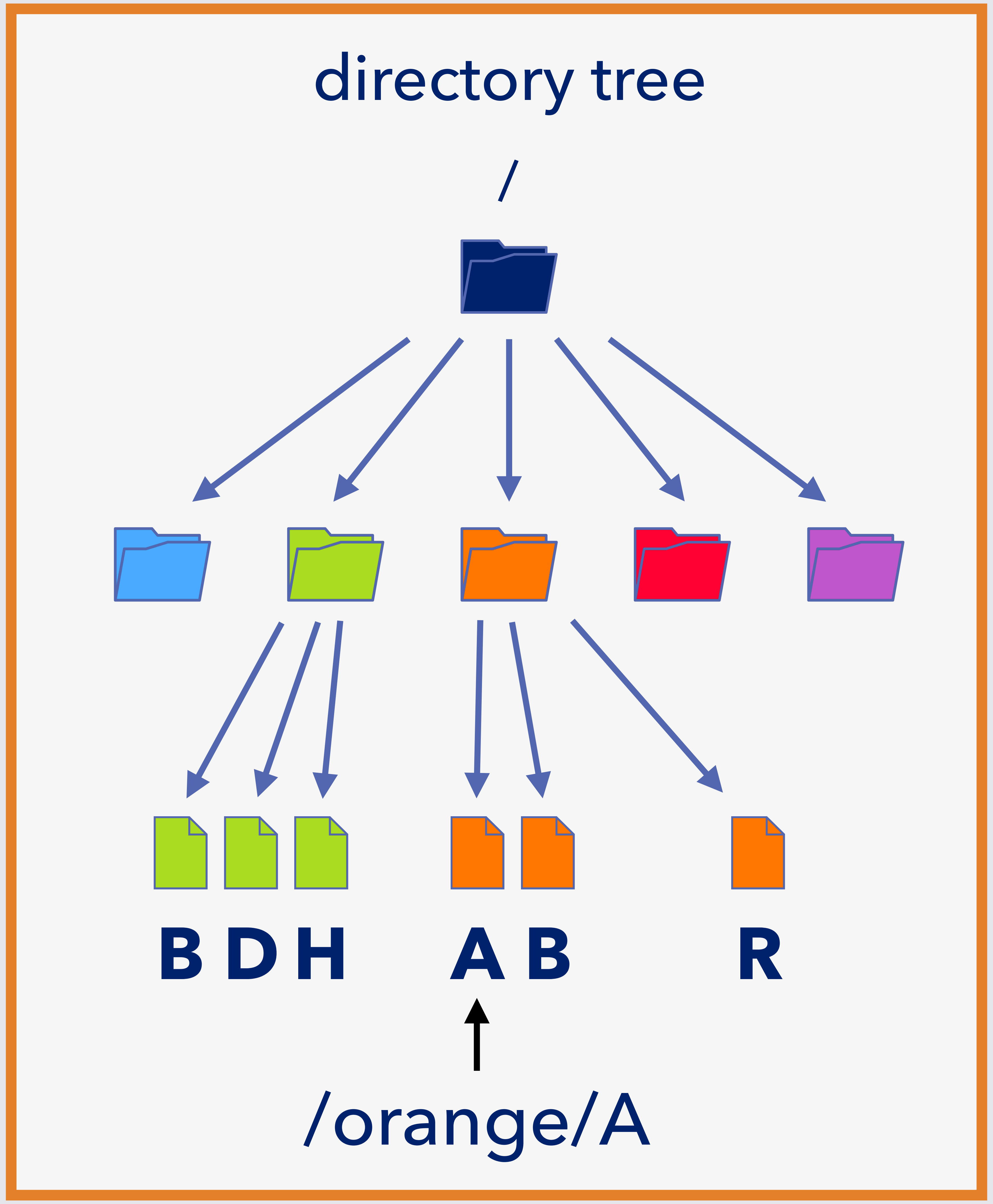


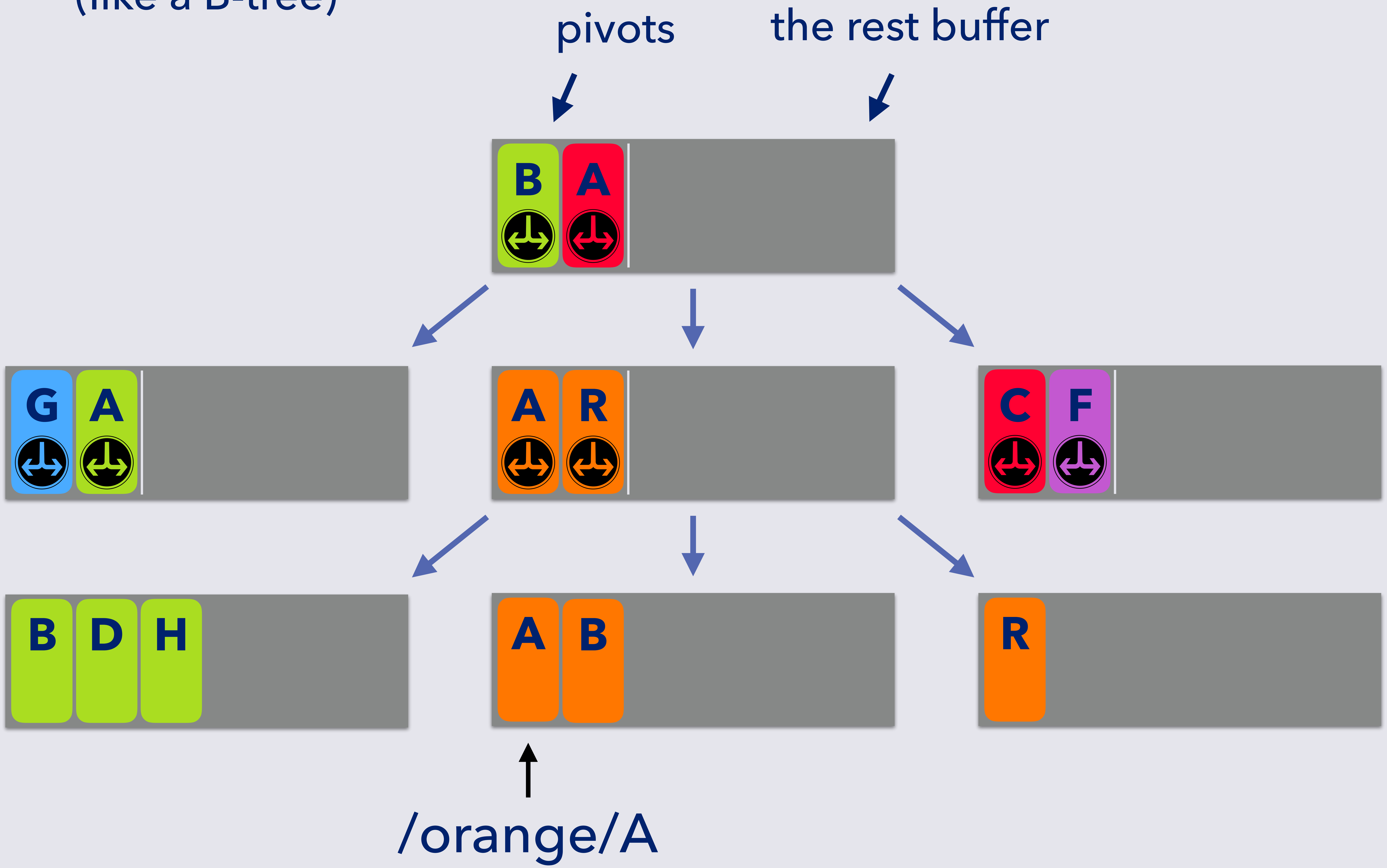
A B$^{\varepsilon}$-tree is a search tree
(like a B-tree)



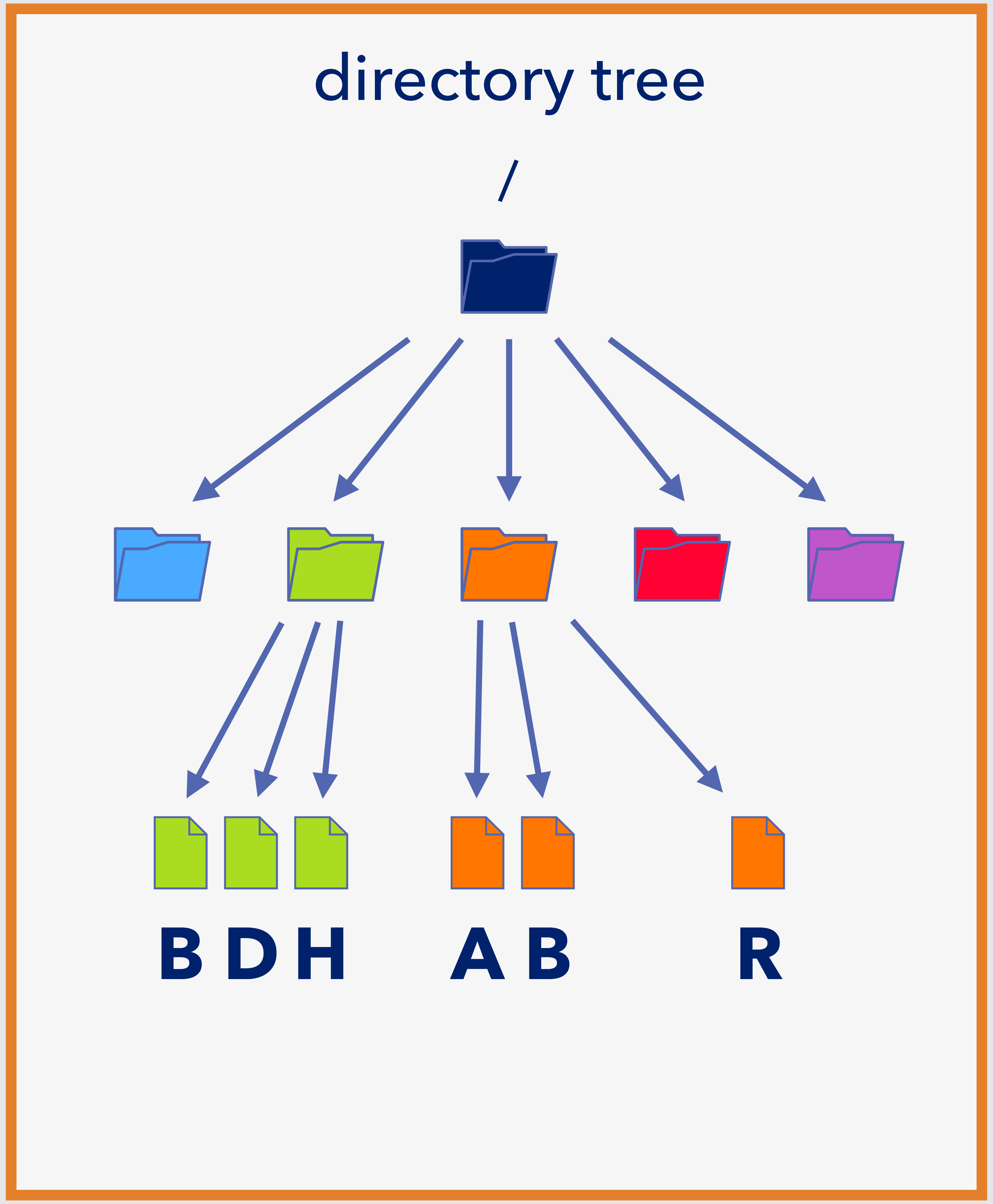
Bε-Trees

A Bε-tree is a search tree
(like a B-tree)





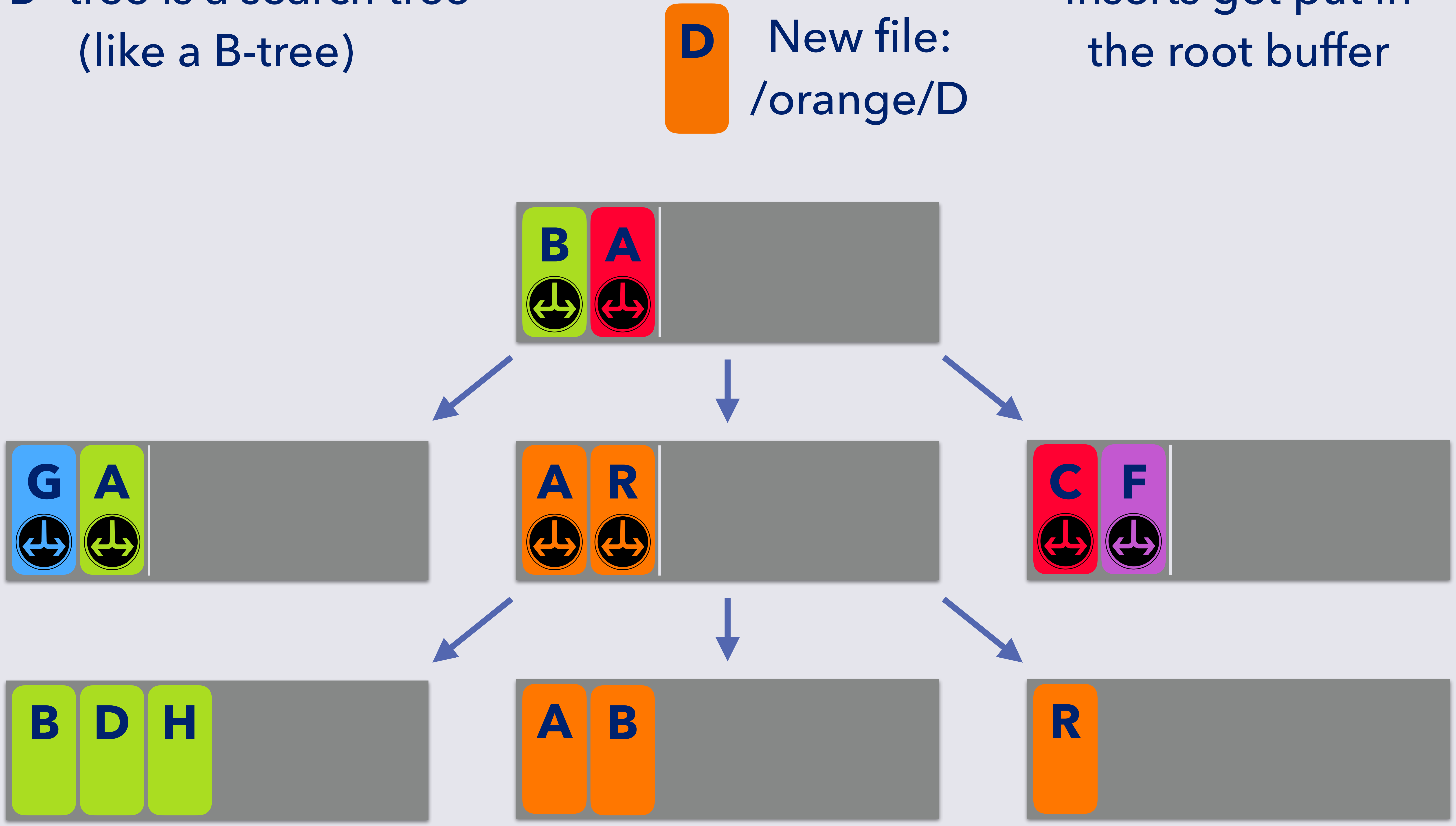
B$^\varepsilon$-Trees



A B$^\varepsilon$-tree is a search tree (like a B-tree)

New file: /orange/D

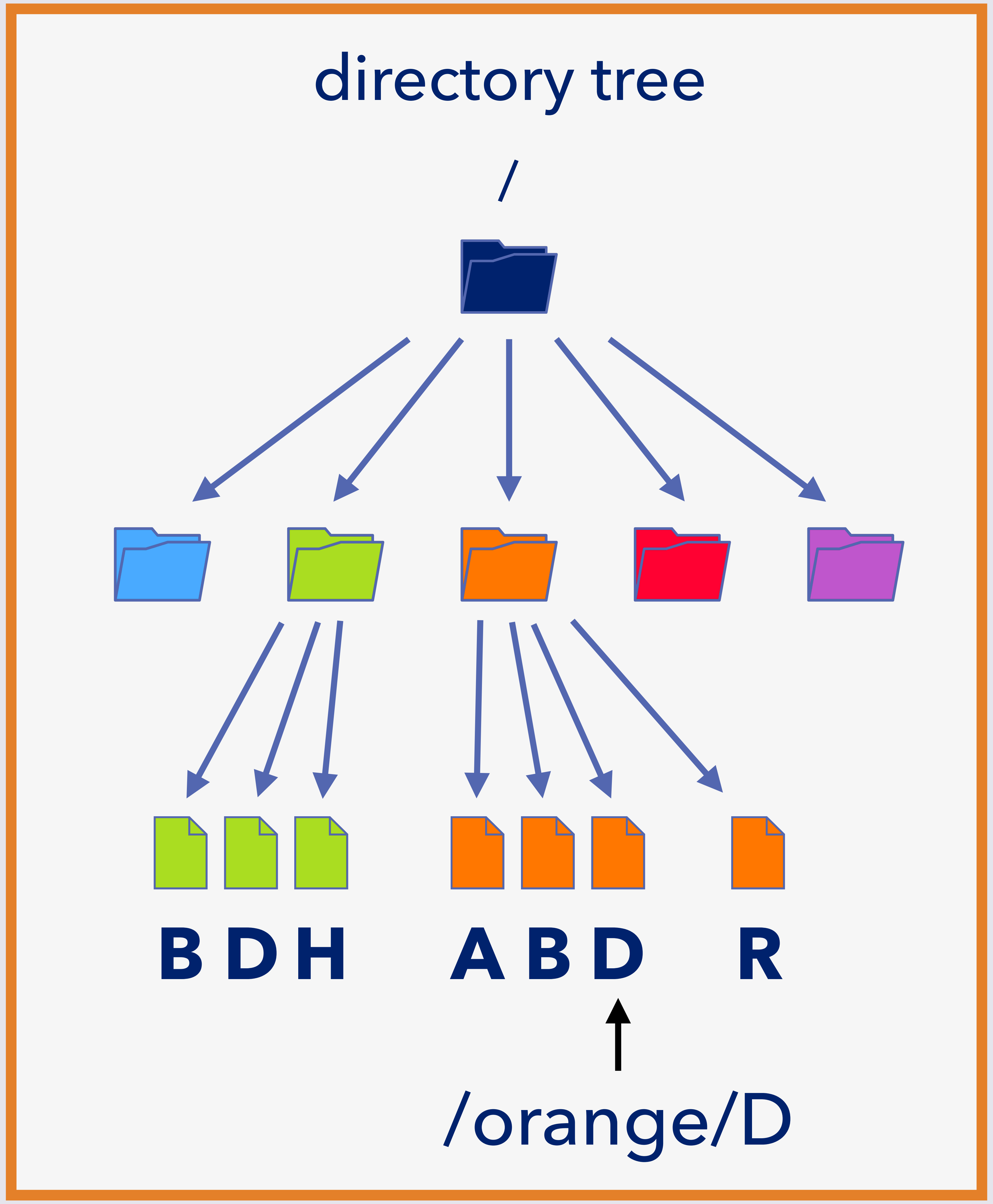
Inserts get put in the root buffer

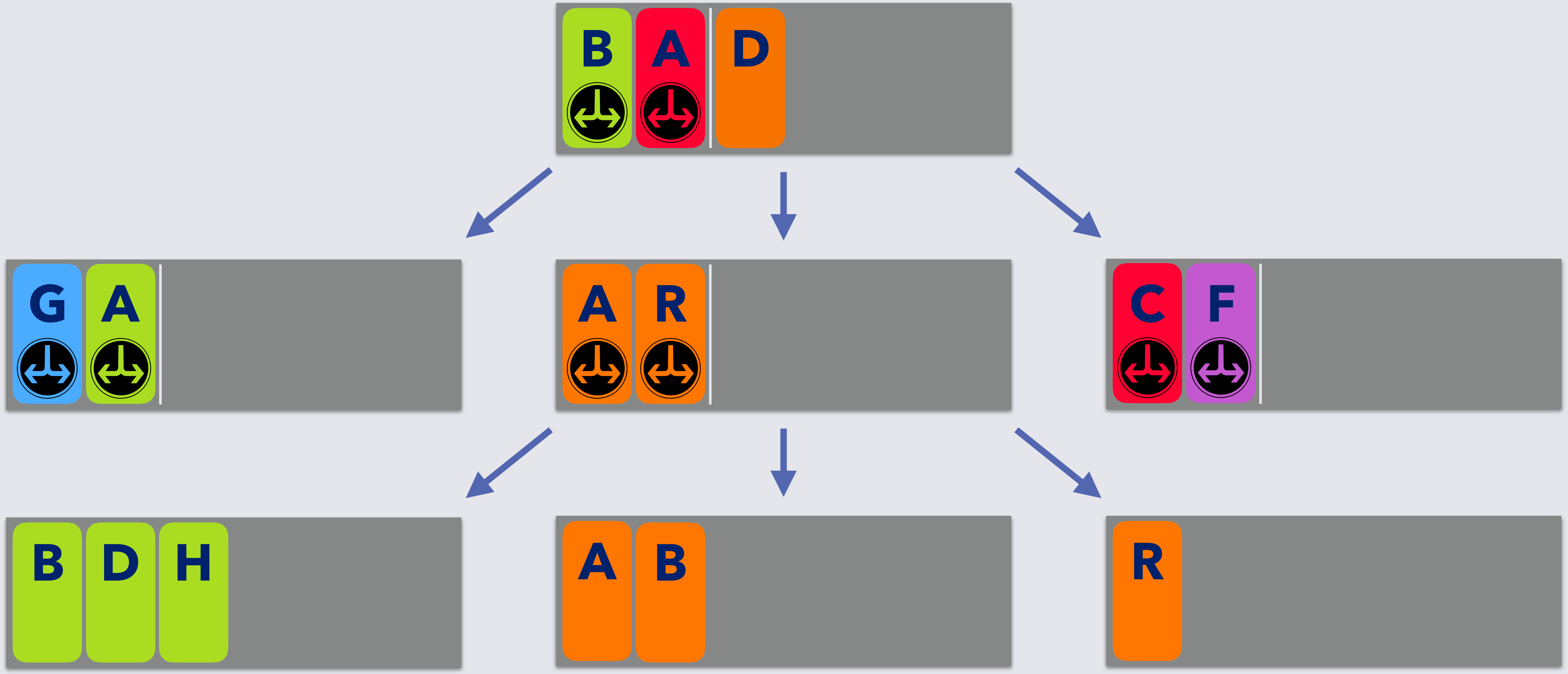


Bε-Trees

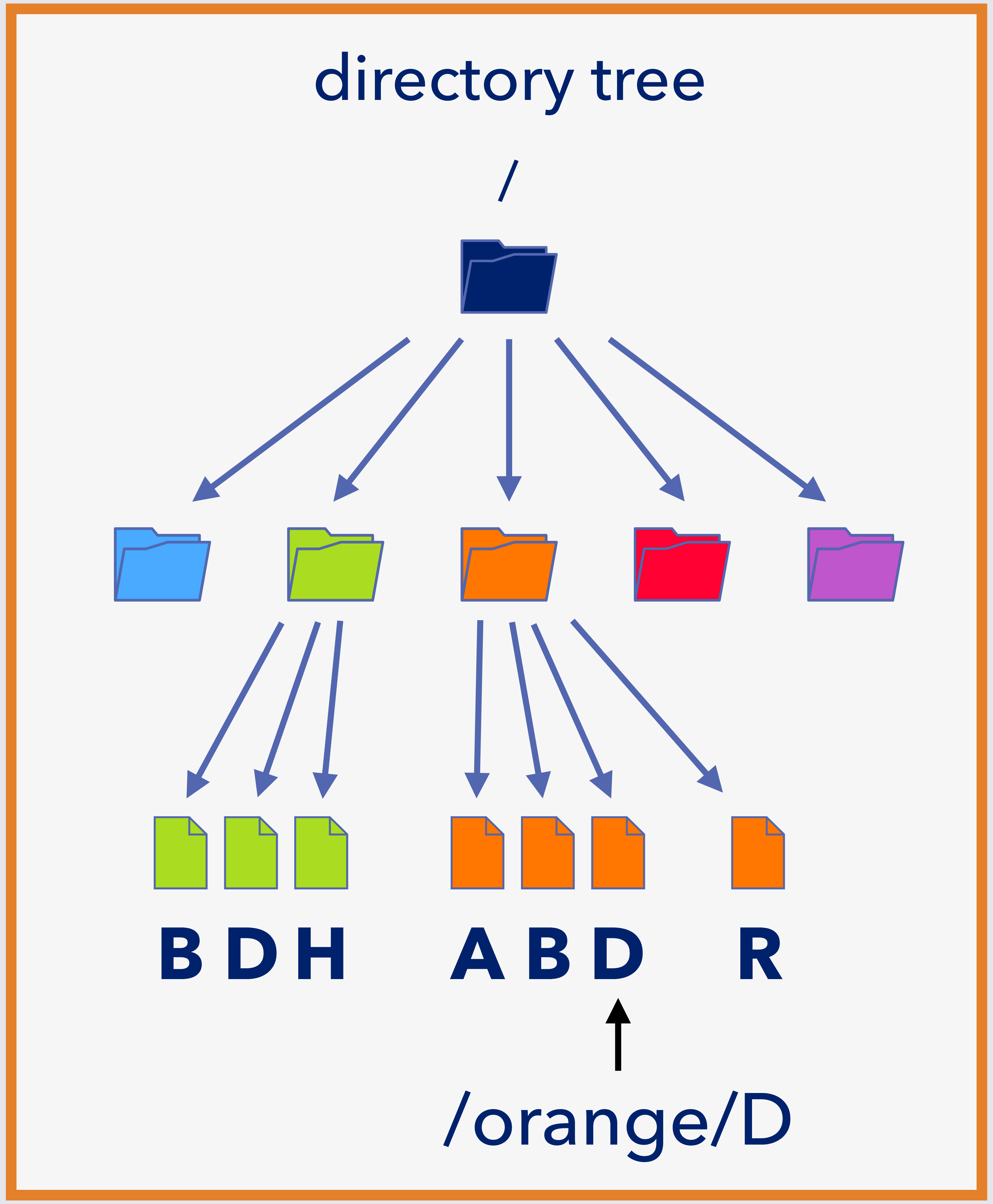
A Bε-tree is a search tree (like a B-tree)

Inserts get put in the root buffer

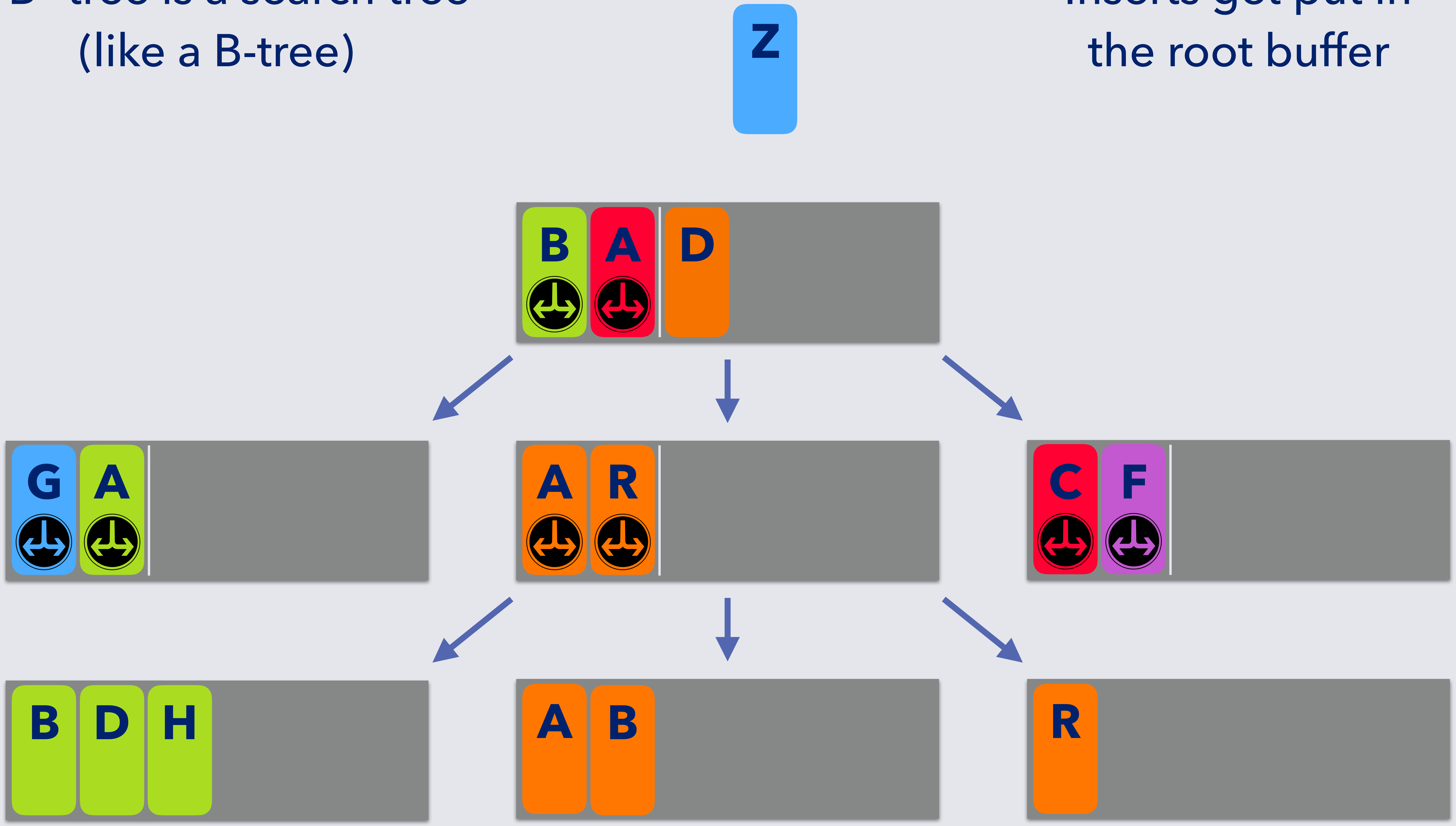




Bε-Trees



A Bε-tree is a search tree (like a B-tree)

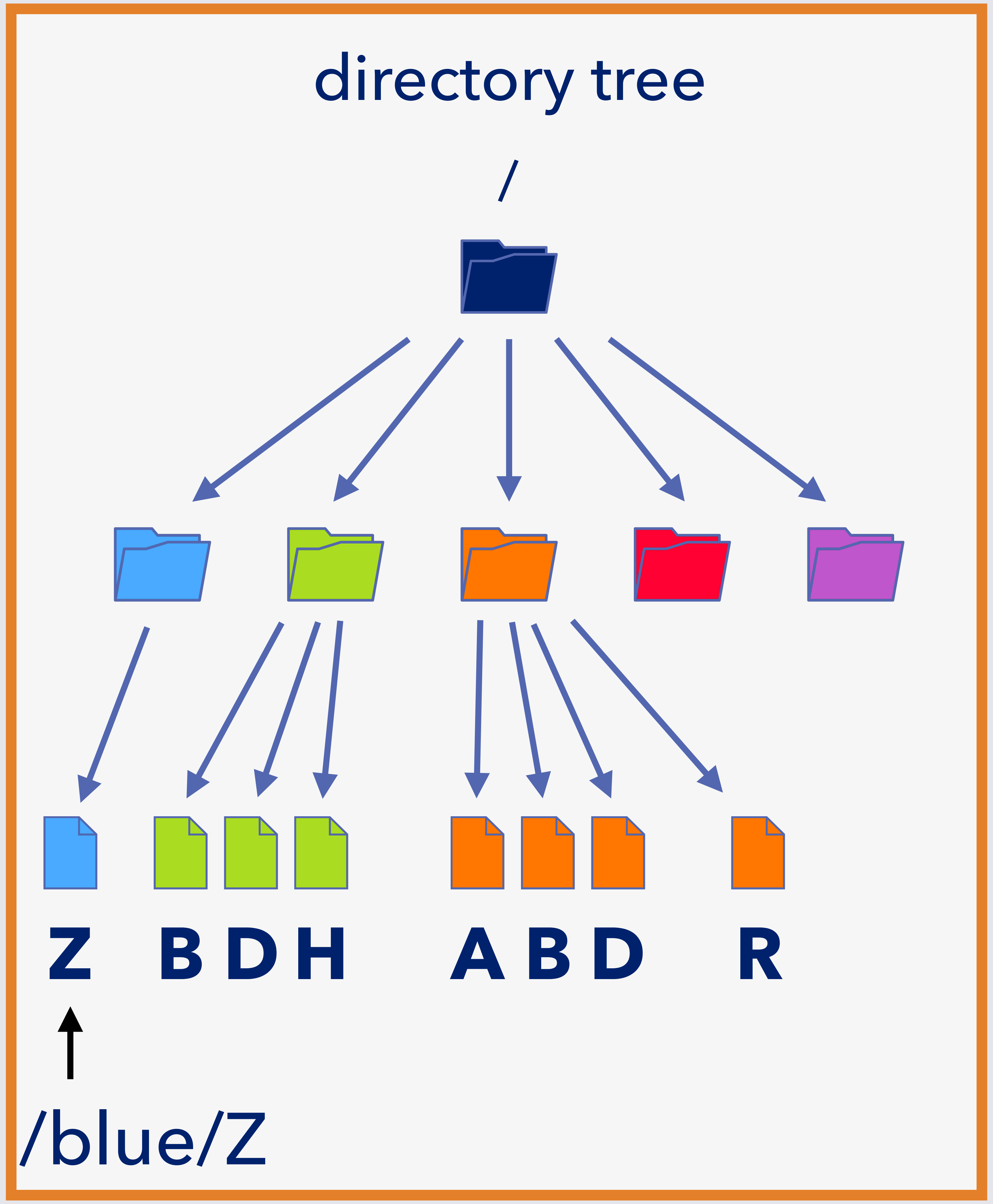


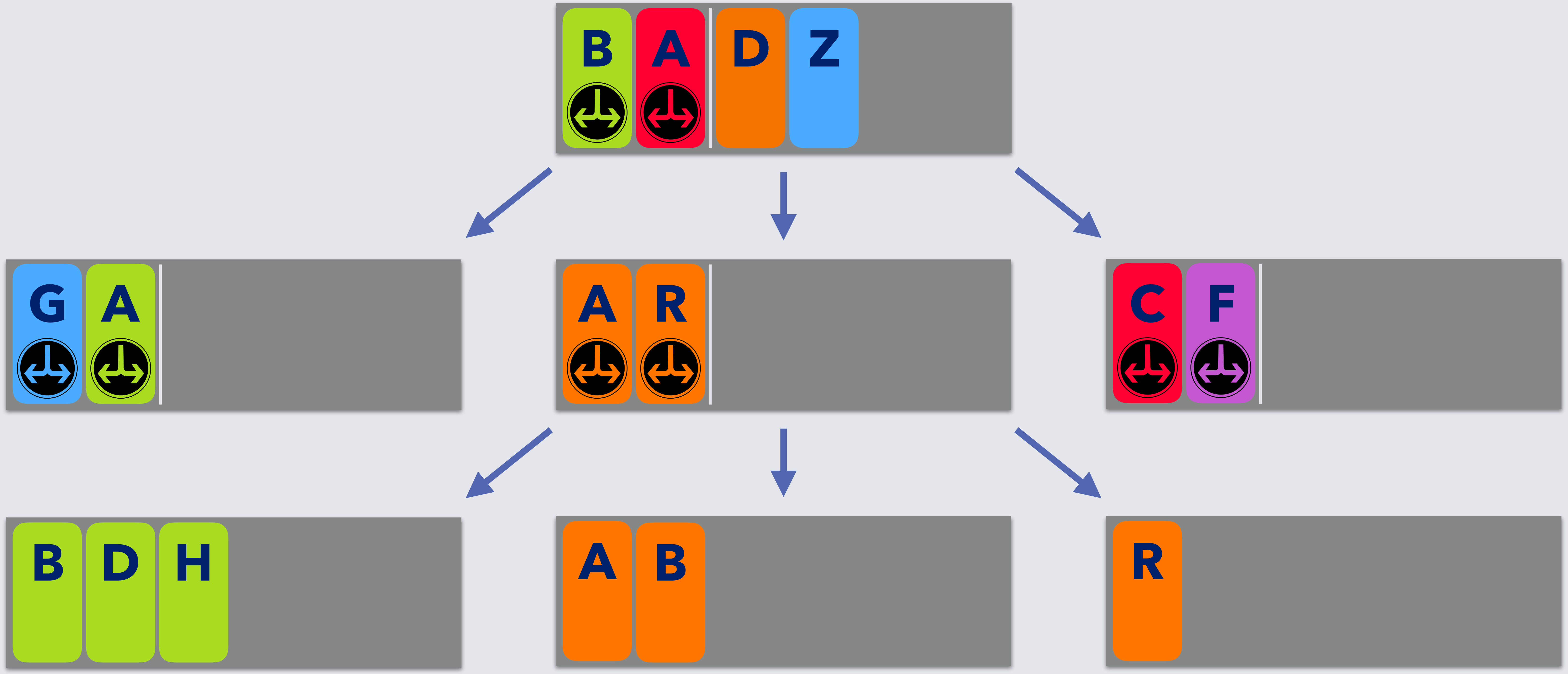
Inserts get put in the root buffer

B^ε-Trees

A B^ε-tree is a search tree (like a B-tree)

Inserts get put in the root buffer

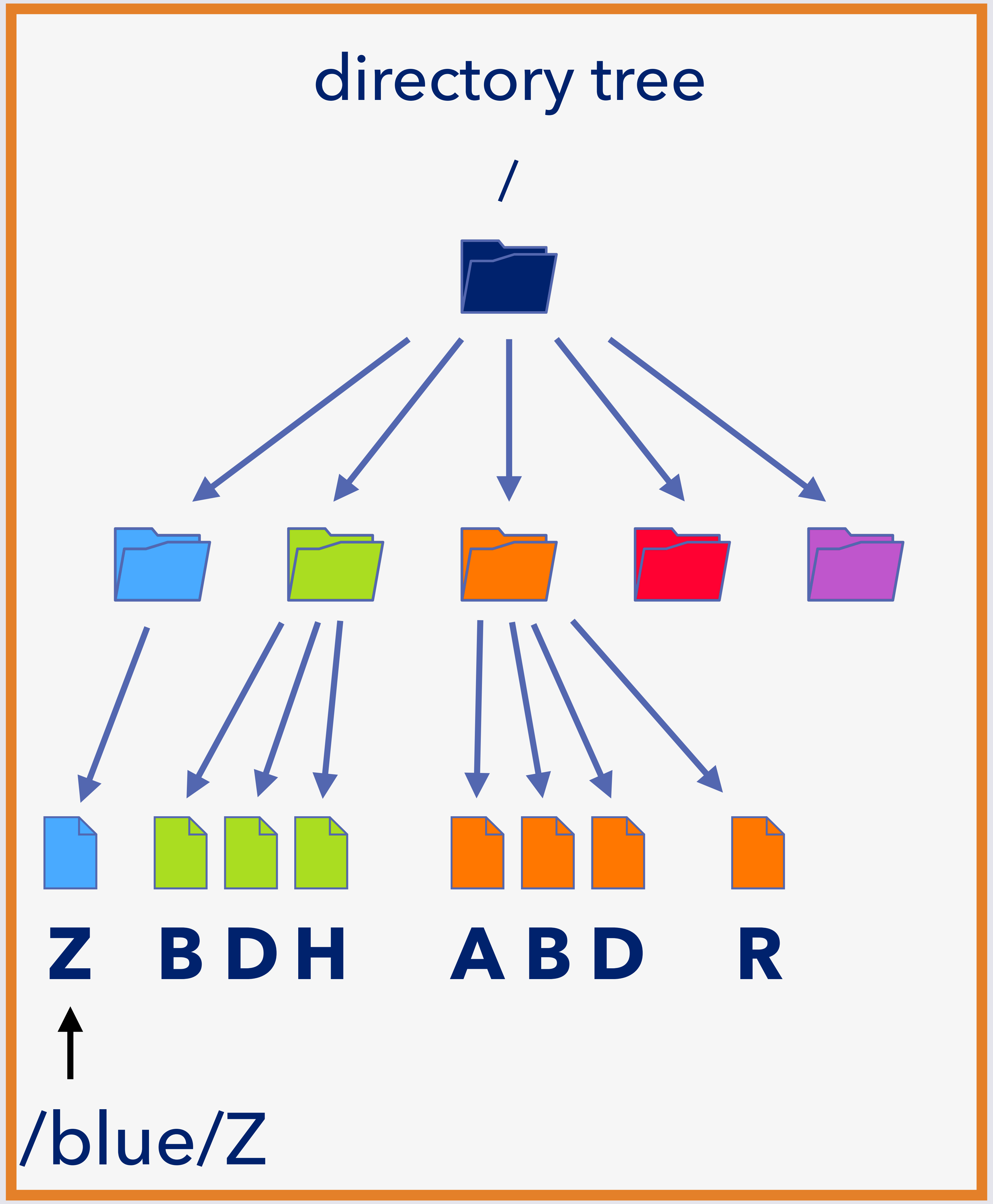


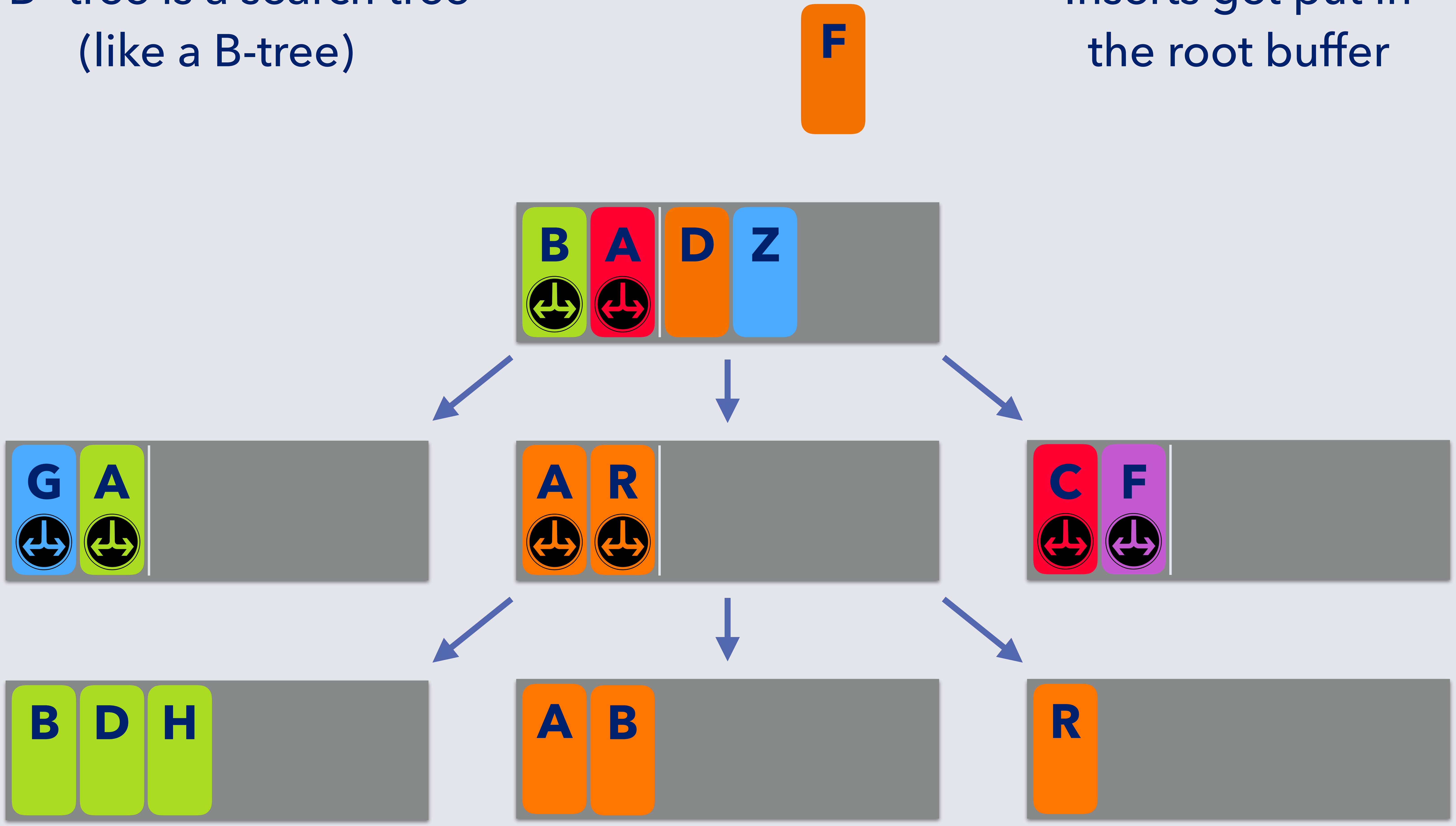


B$^{\varepsilon}$-Trees

A B$^{\varepsilon}$-tree is a search tree
(like a B-tree)

Inserts get put in
the root buffer

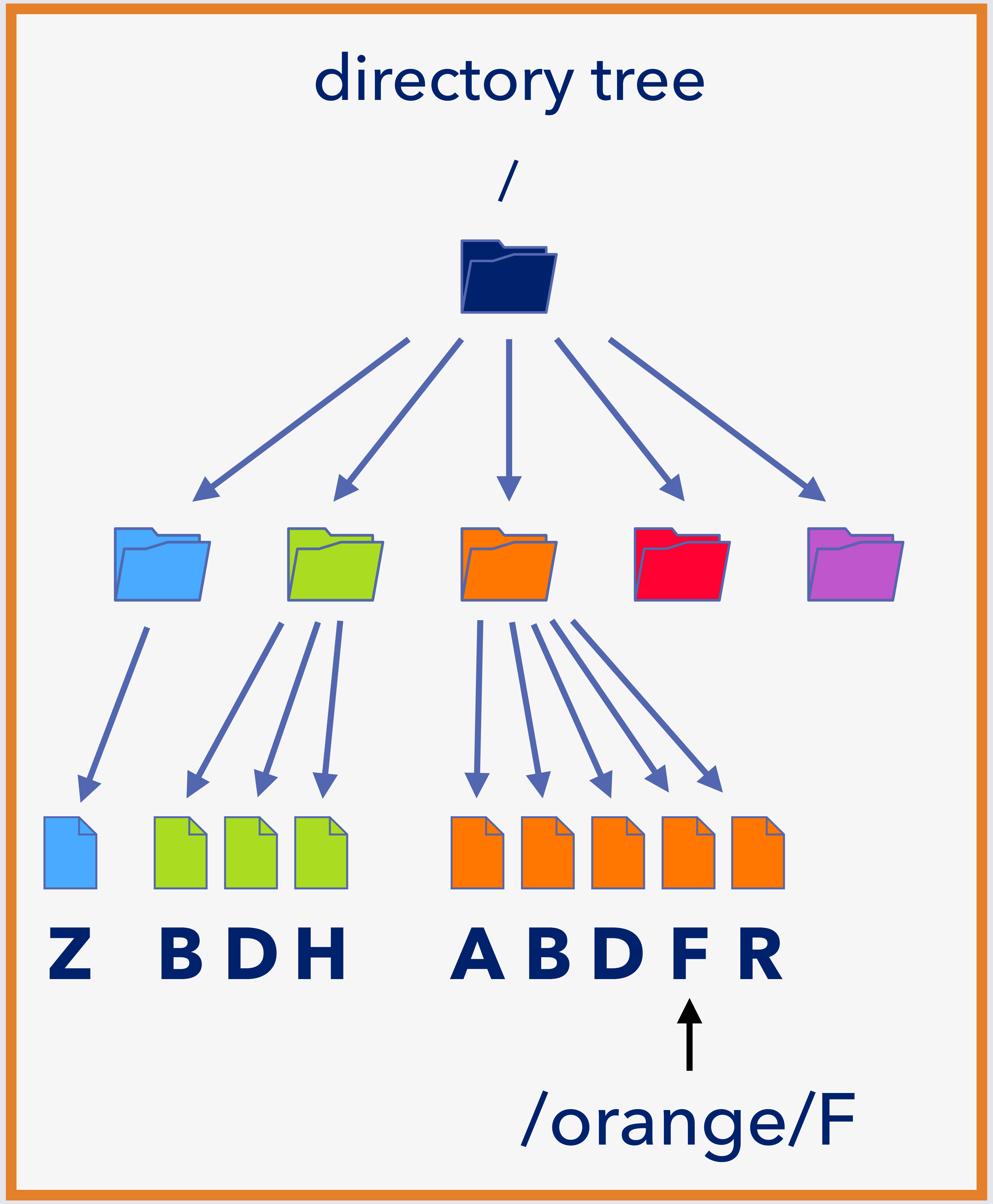


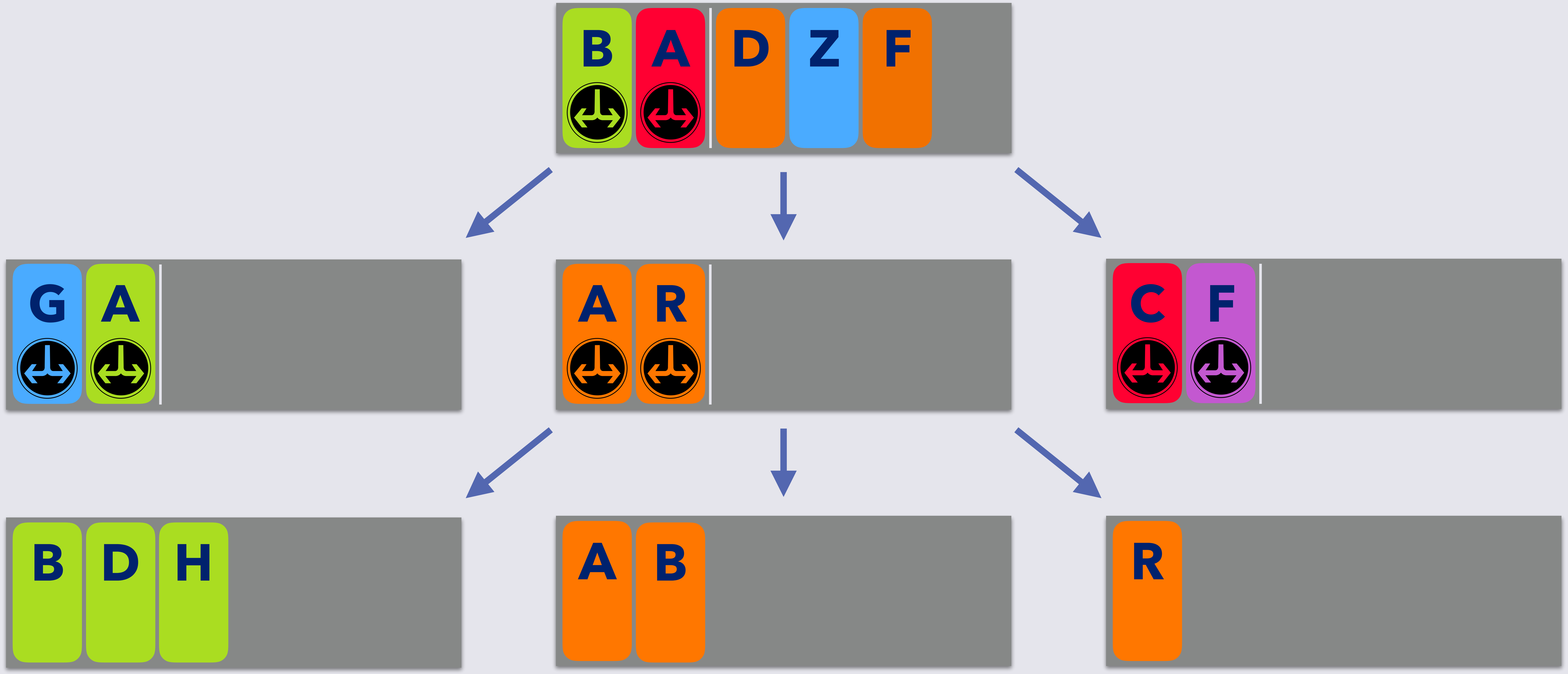


Bε-Trees

A Bε-tree is a search tree
(like a B-tree)

Inserts get put in
the root buffer



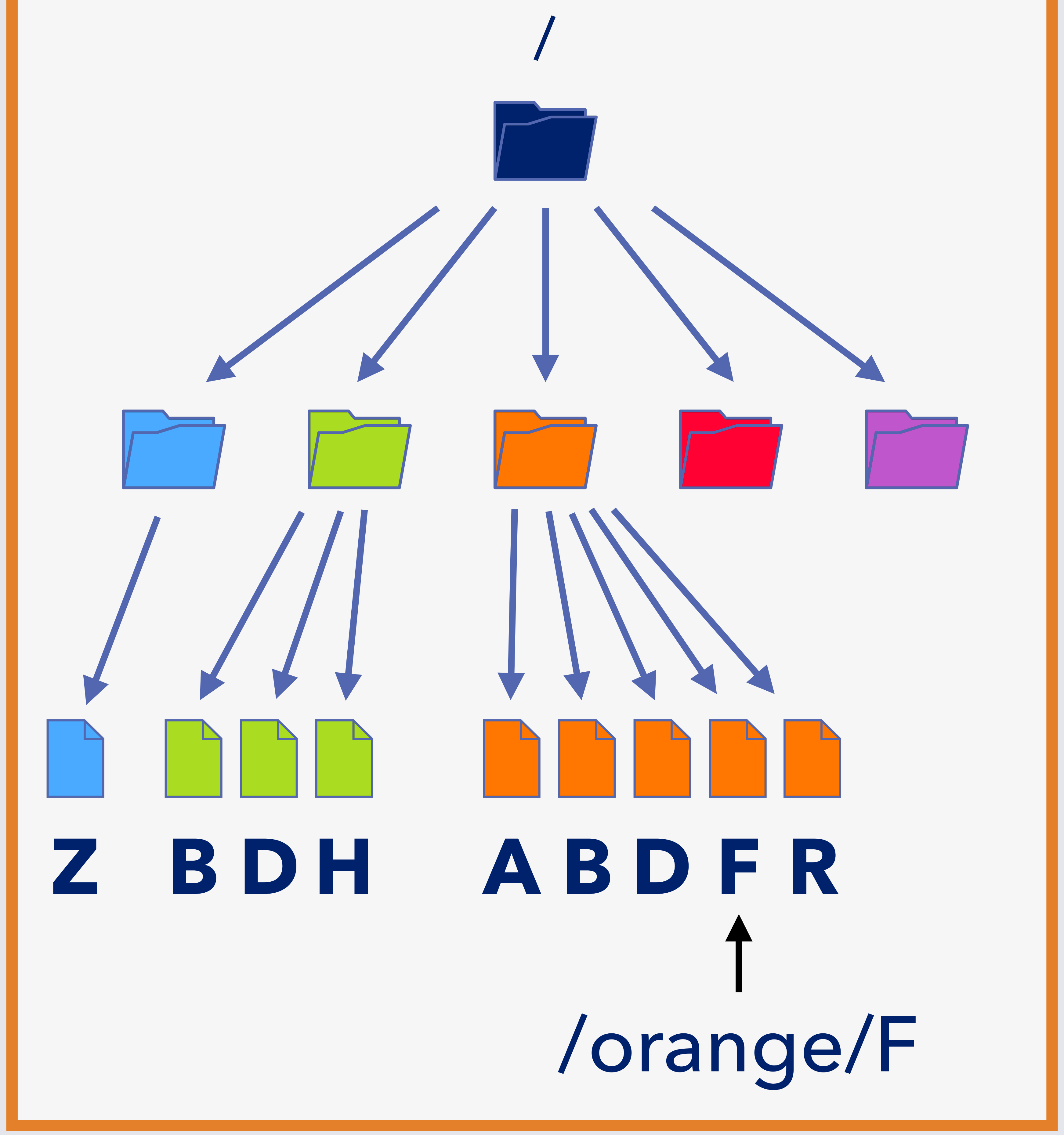


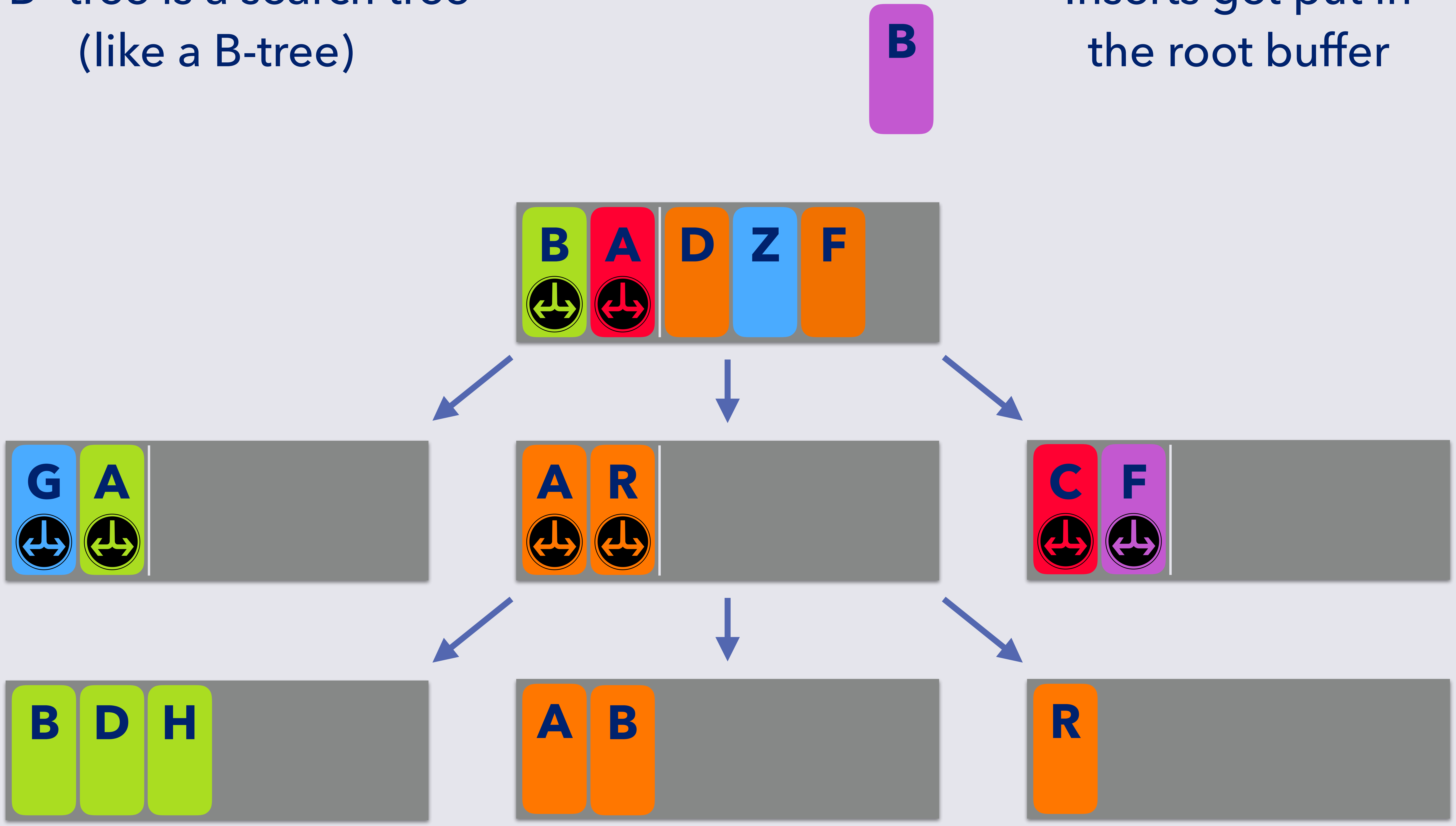
Bε-Trees

A Bε-tree is a search tree (like a B-tree)

Inserts get put in the root buffer

directory tree





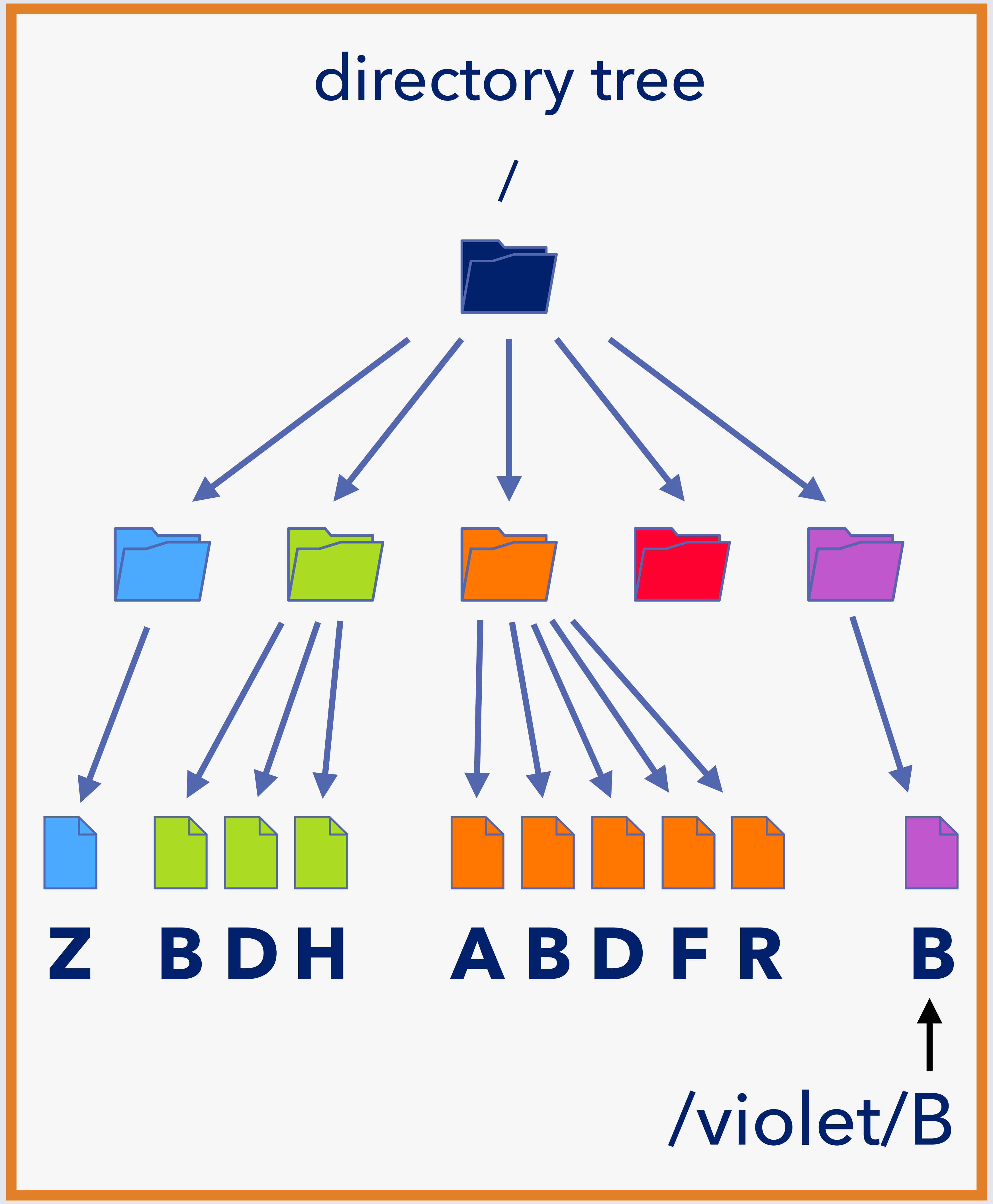
B^{ε}-Trees

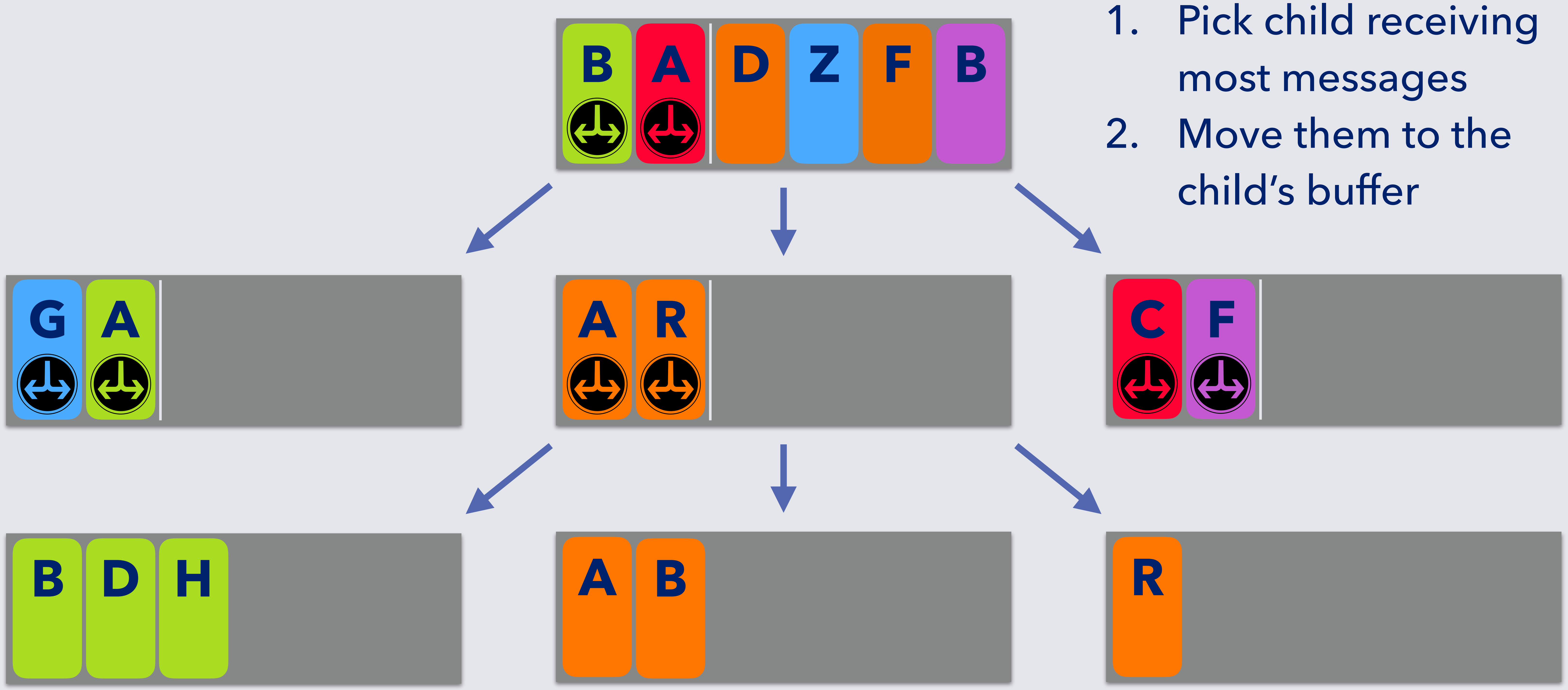
A B^{ε}-tree is a search tree (like a B-tree)

Inserts get put in the root buffer

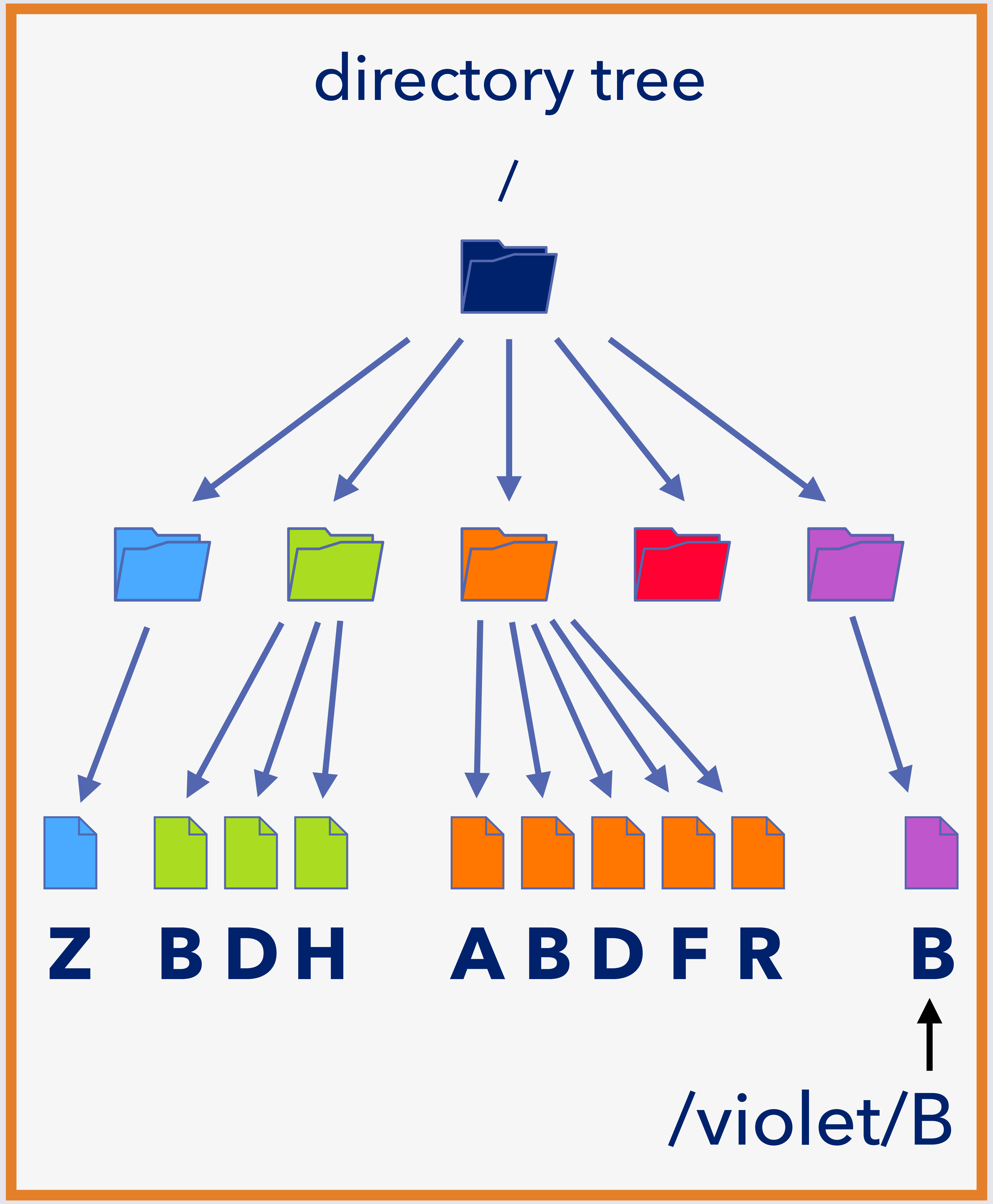
When a buffer is full:

1. Pick child receiving most messages
2. Move them to the child's buffer

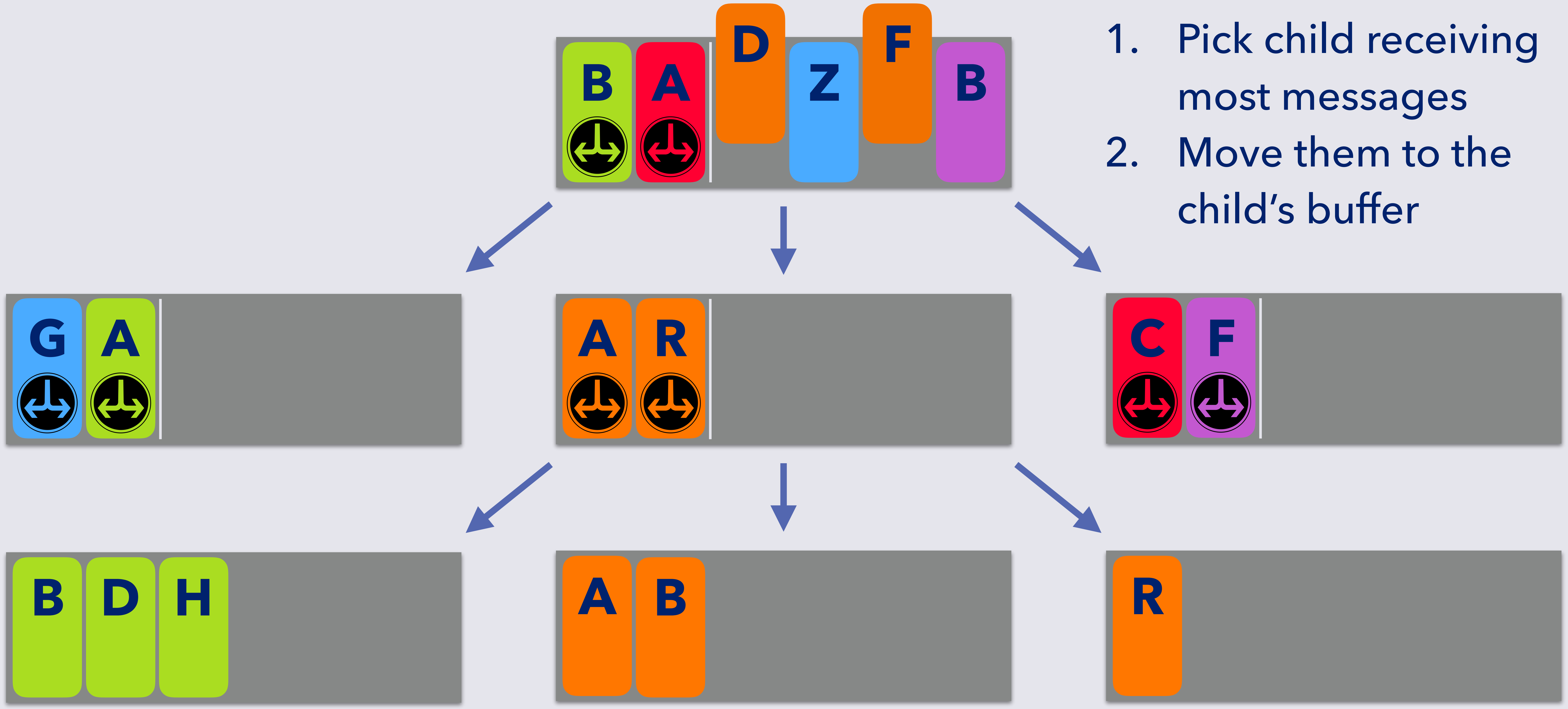




Bε-Trees



A Bε-tree is a search tree (like a B-tree)

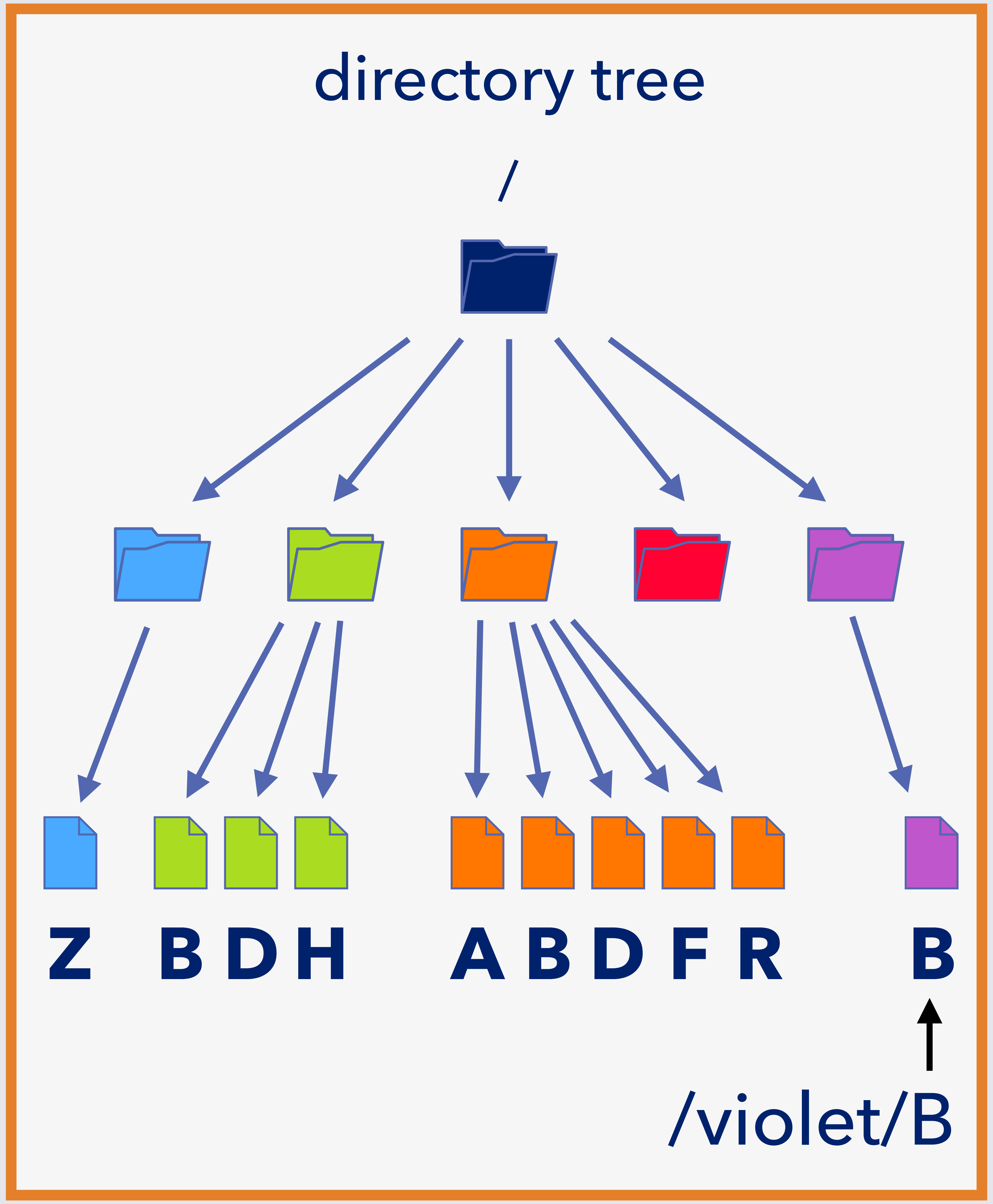


Inserts get put in the root buffer

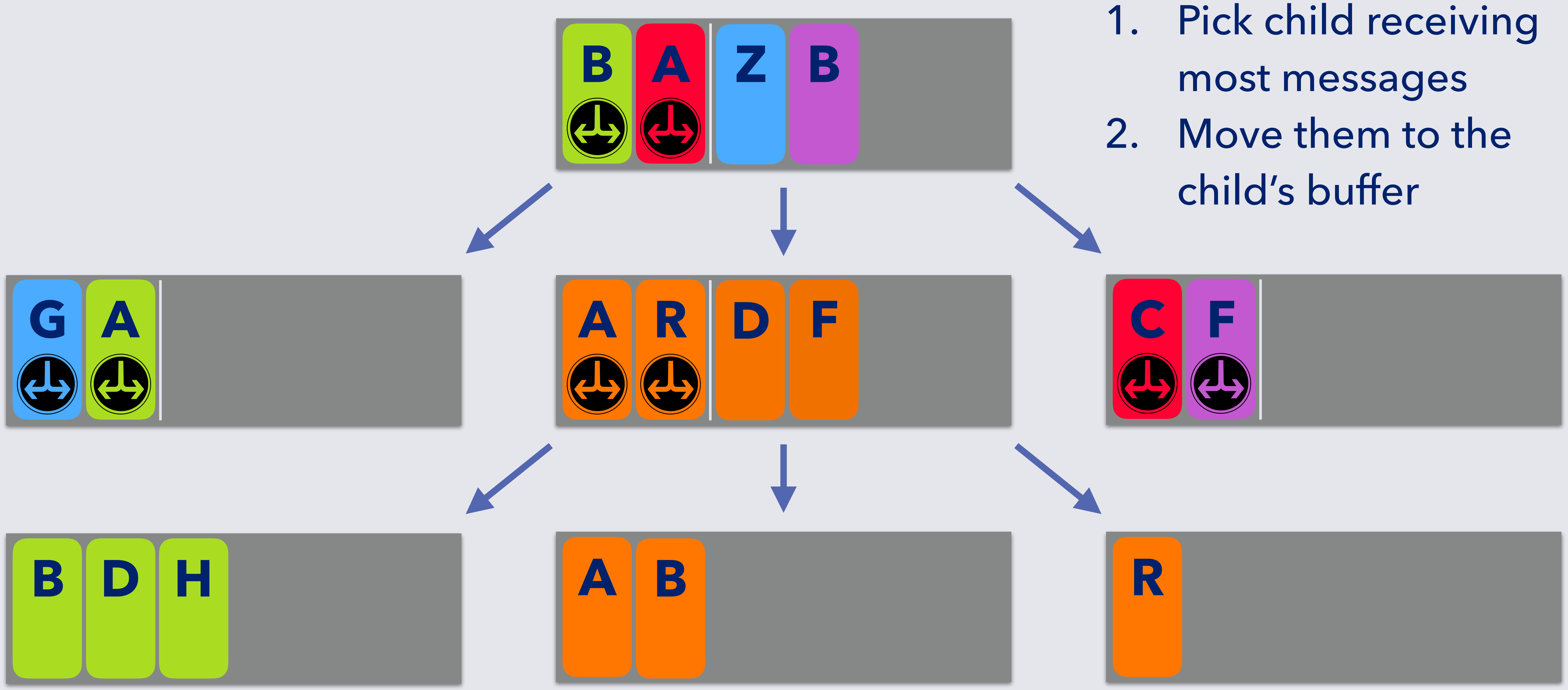
When a buffer is full:

1. Pick child receiving most messages
2. Move them to the child's buffer

B$^{\varepsilon}$-Trees



A B$^{\varepsilon}$-tree is a search tree (like a B-tree)

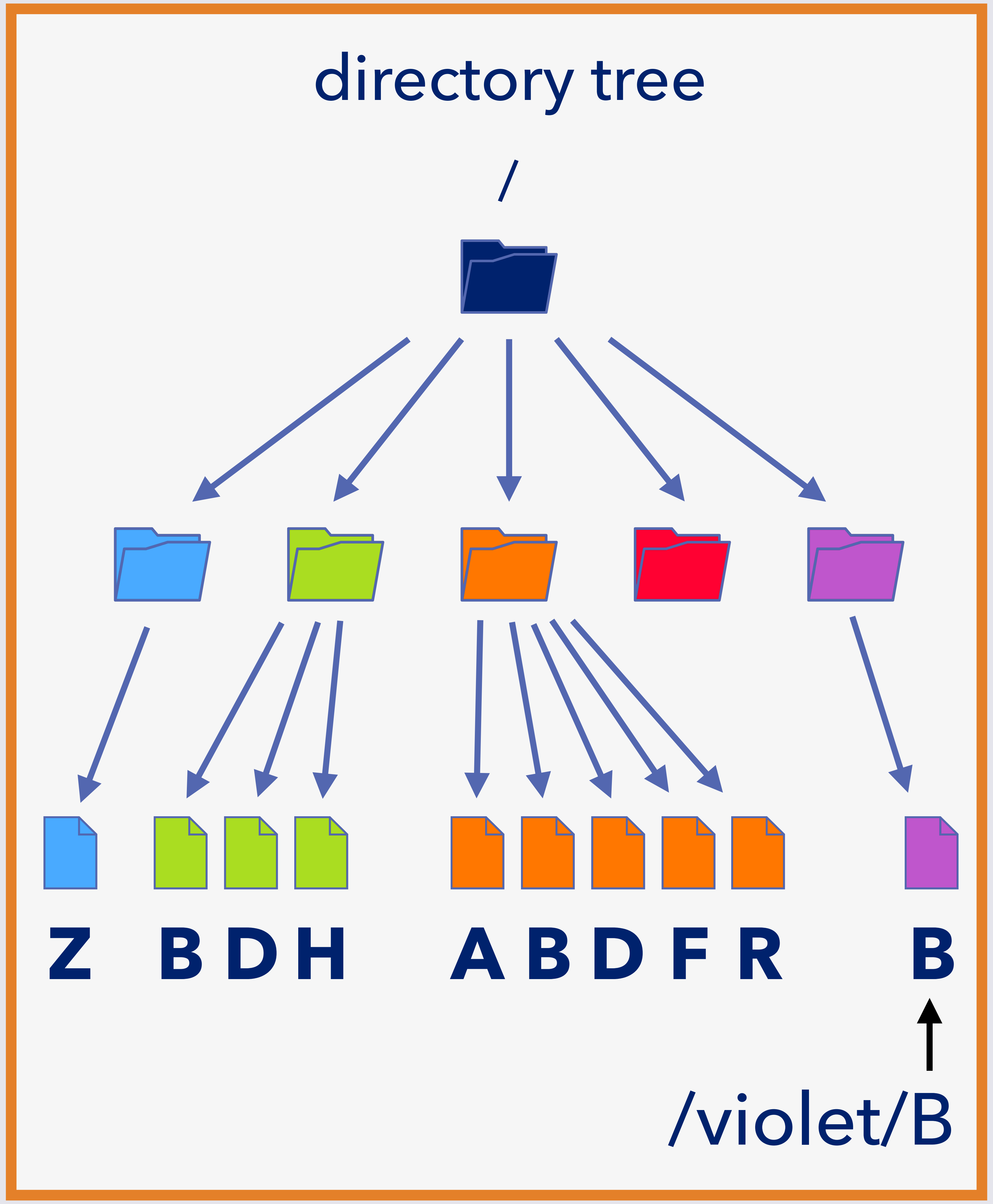


Inserts get put in the root buffer

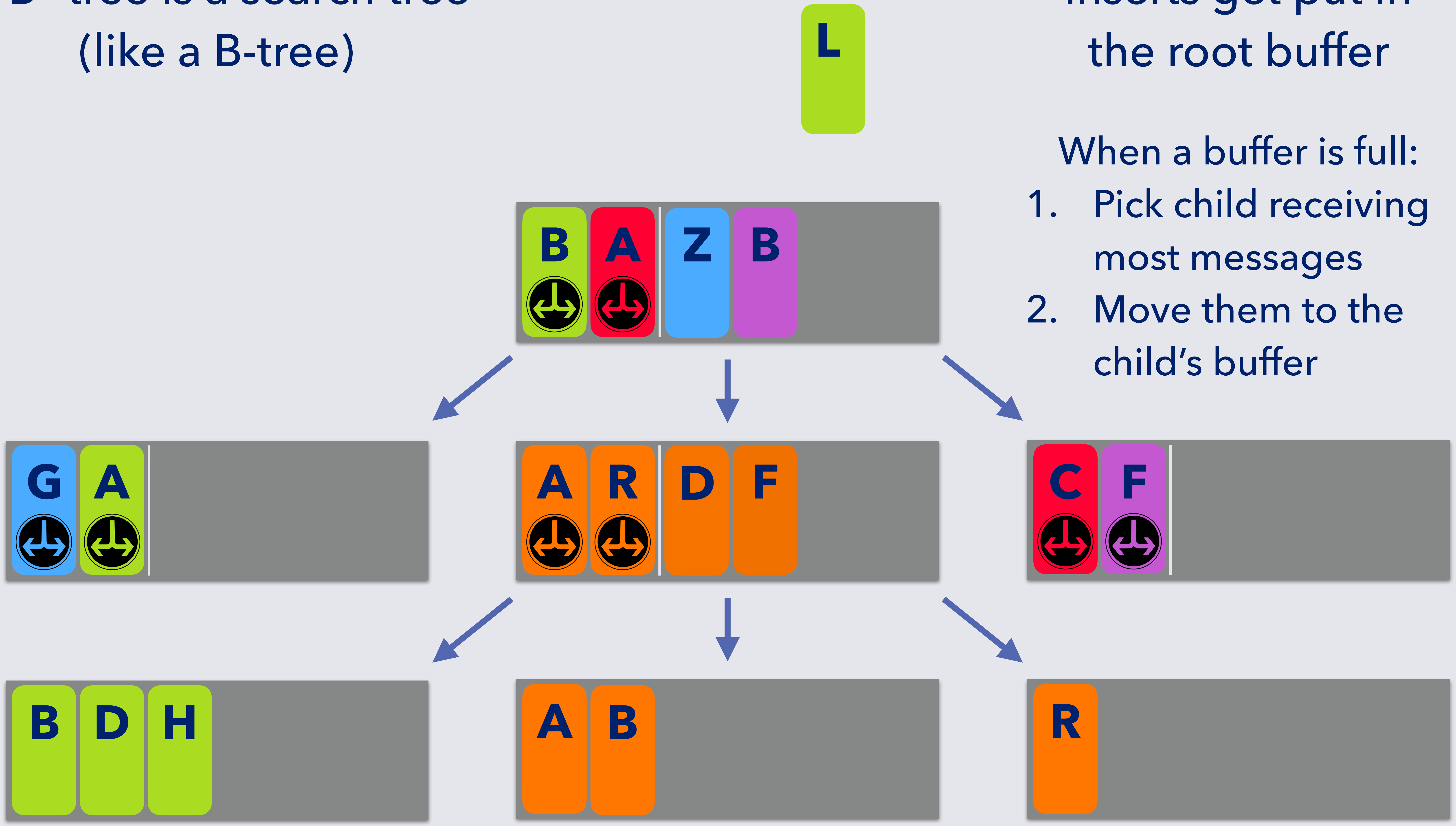
When a buffer is full:

1. Pick child receiving most messages
2. Move them to the child's buffer

Bε-Trees



A Bε-tree is a search tree
(like a B-tree)

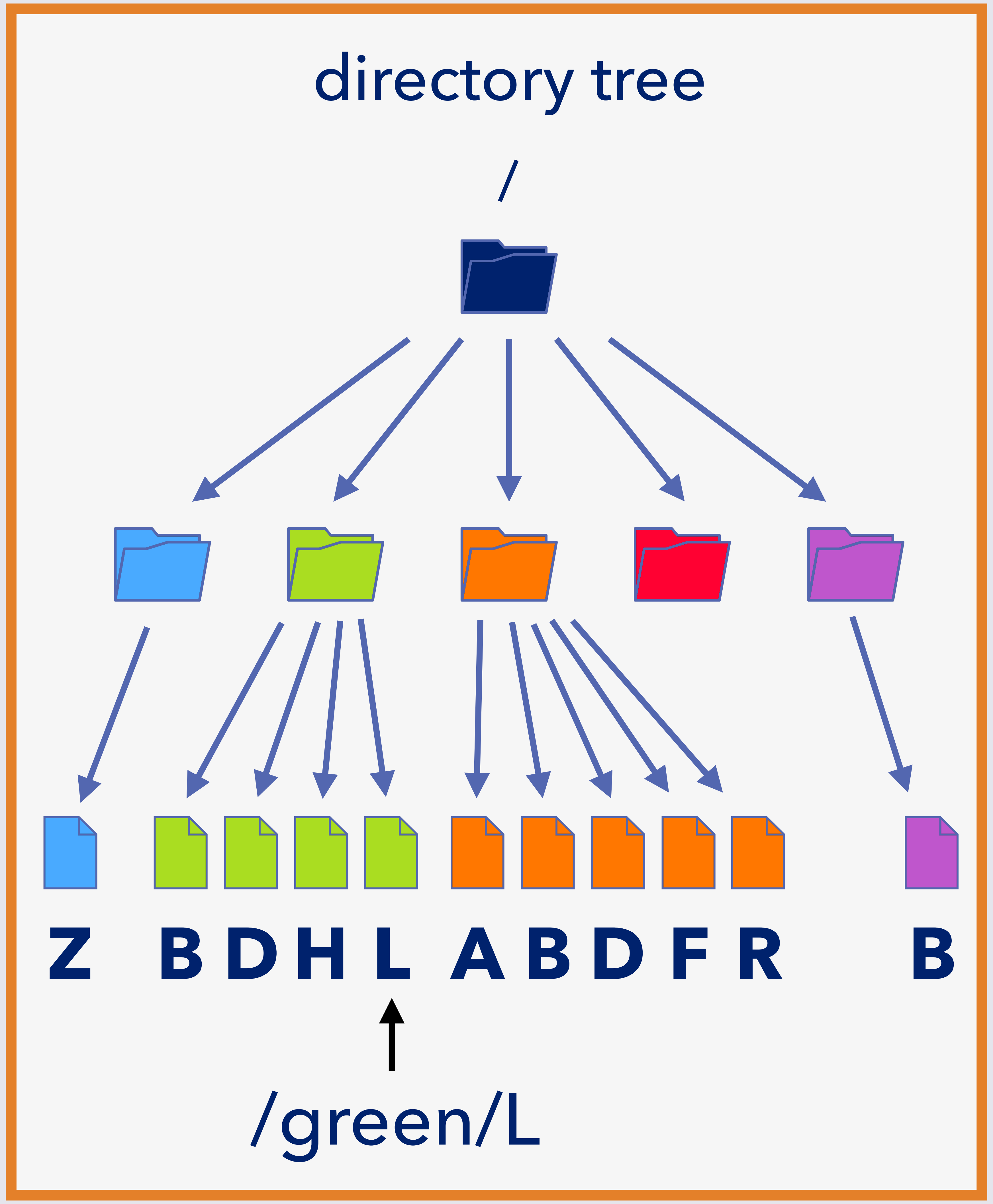


Inserts get put in the root buffer

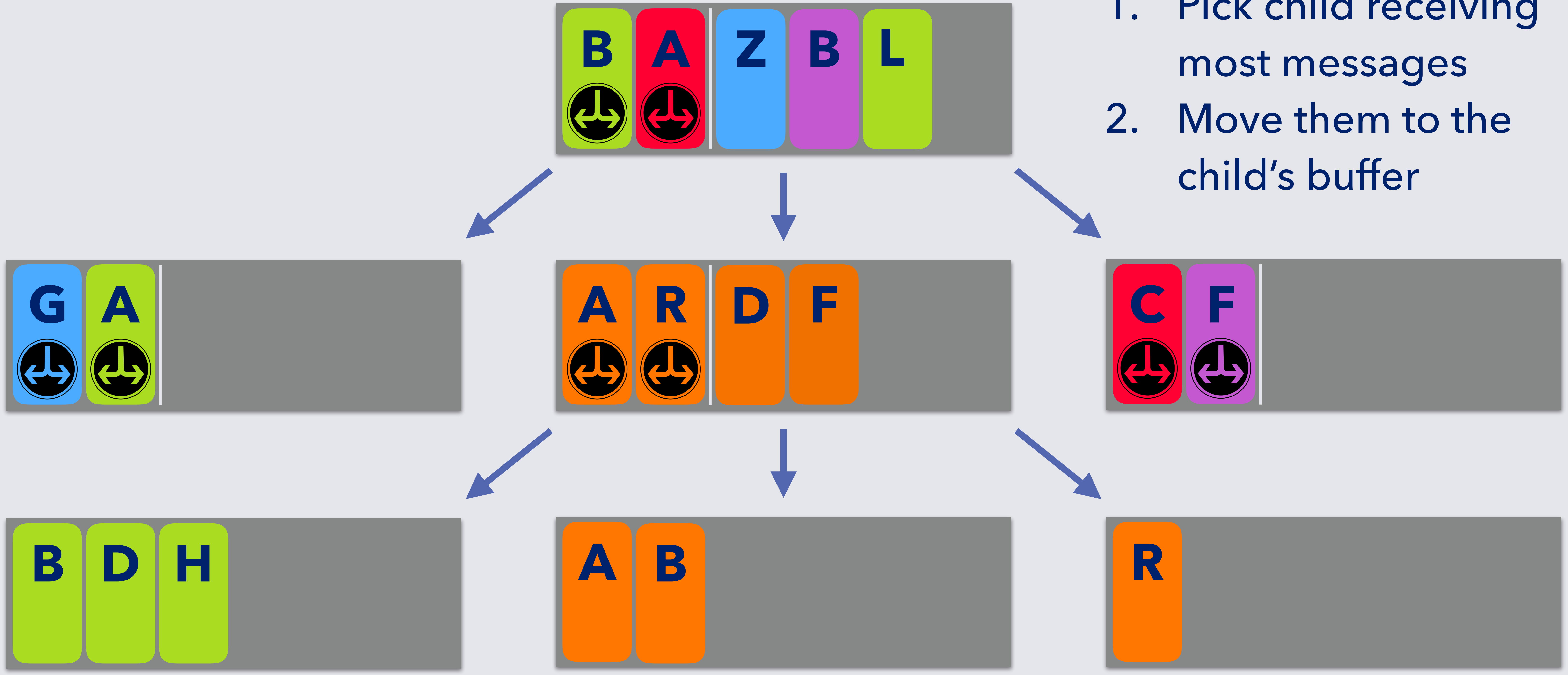
When a buffer is full:

1. Pick child receiving most messages
2. Move them to the child's buffer

Bε-Trees



A Bε-tree is a search tree (like a B-tree)

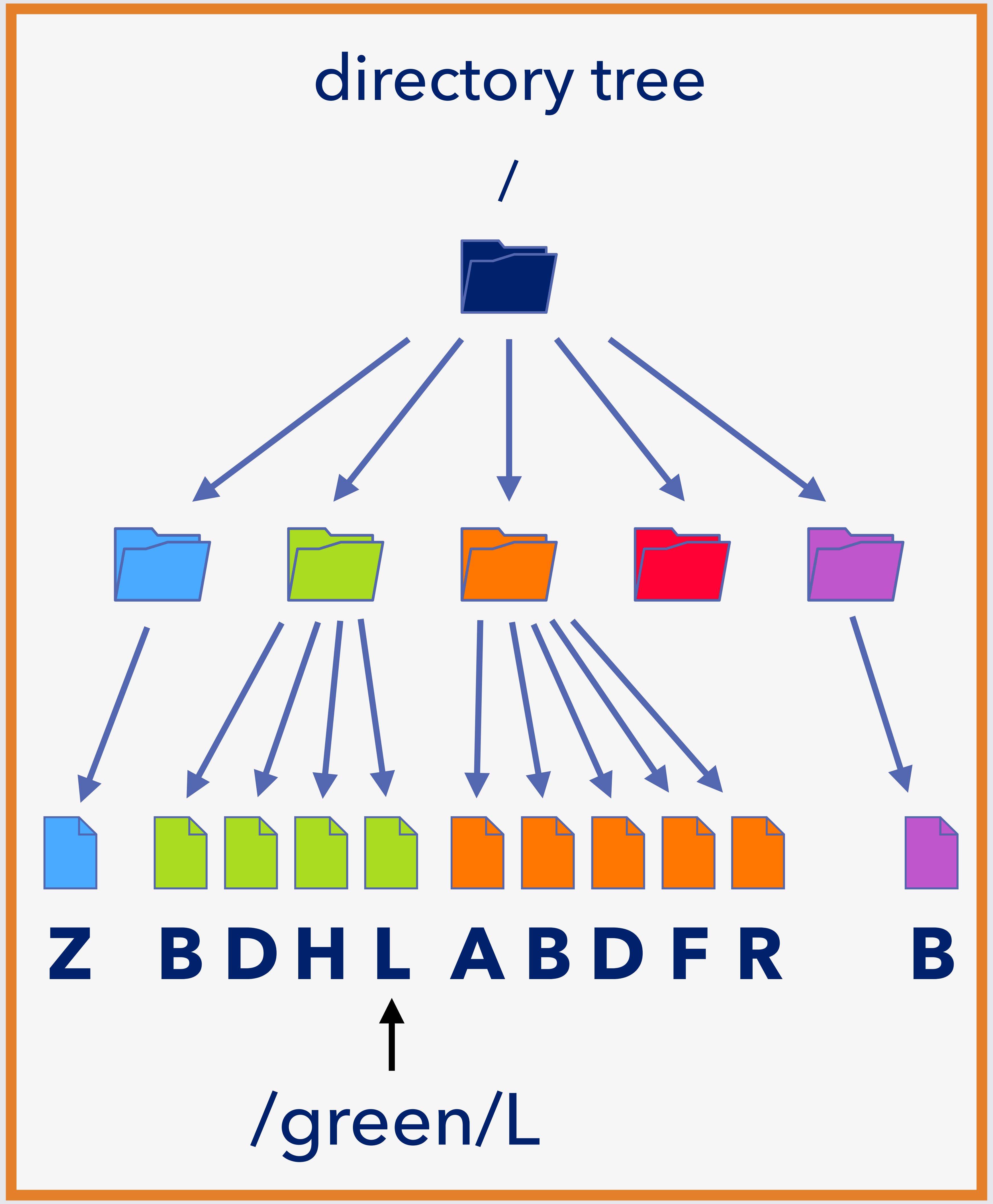


Inserts get put in the root buffer

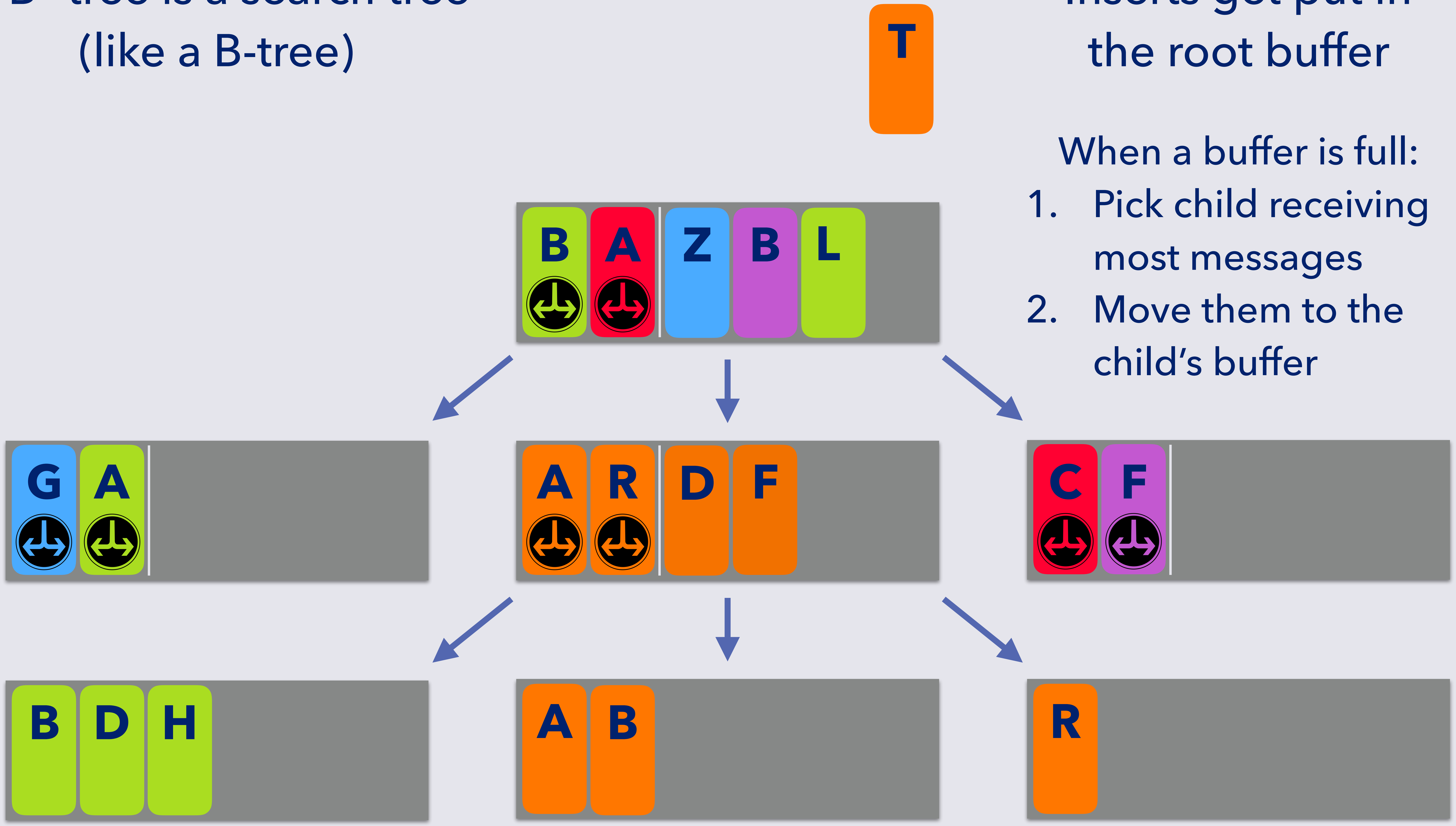
When a buffer is full:

1. Pick child receiving most messages
2. Move them to the child's buffer

Bε-Trees



A Bε-tree is a search tree
(like a B-tree)

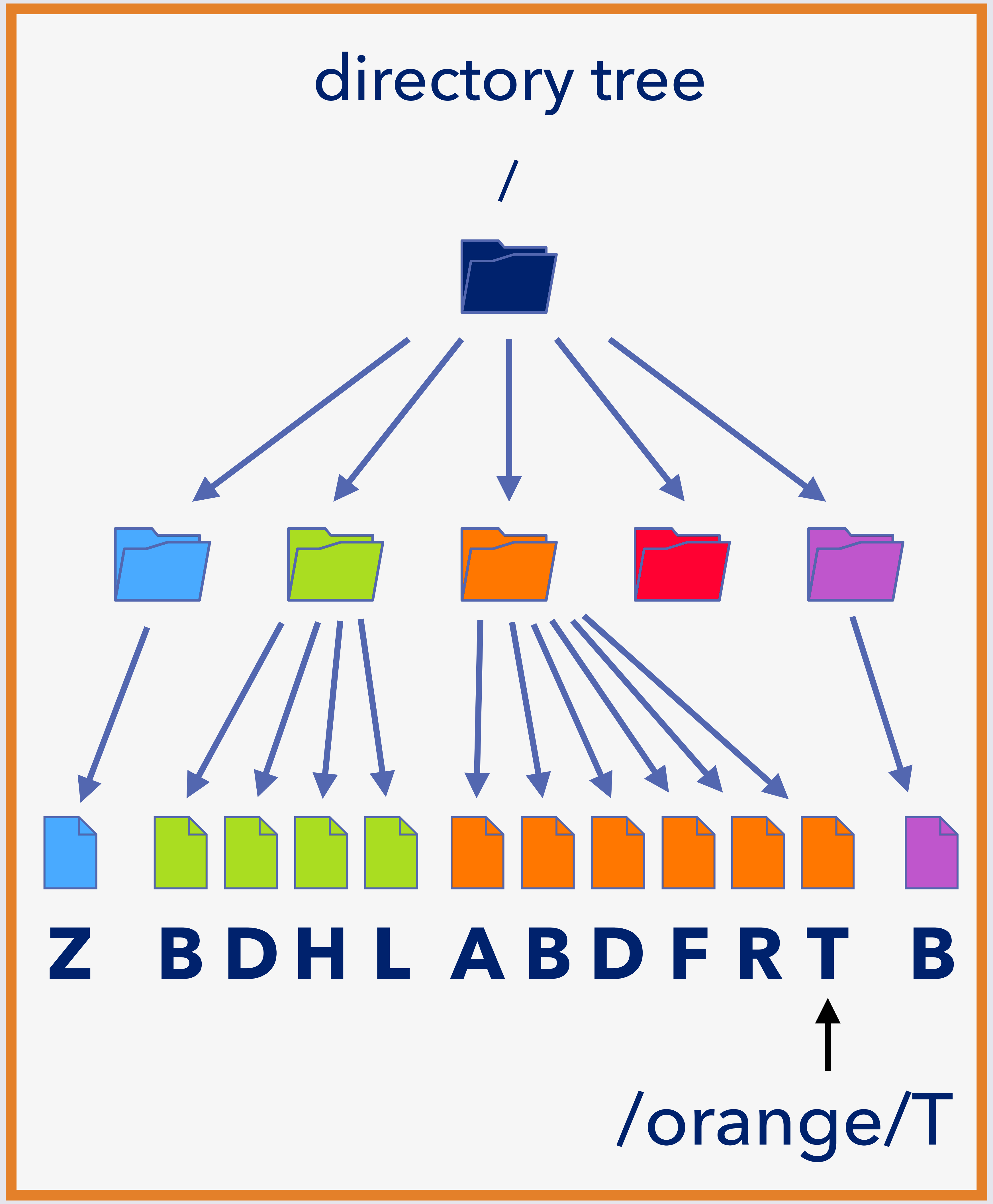


Inserts get put in the root buffer

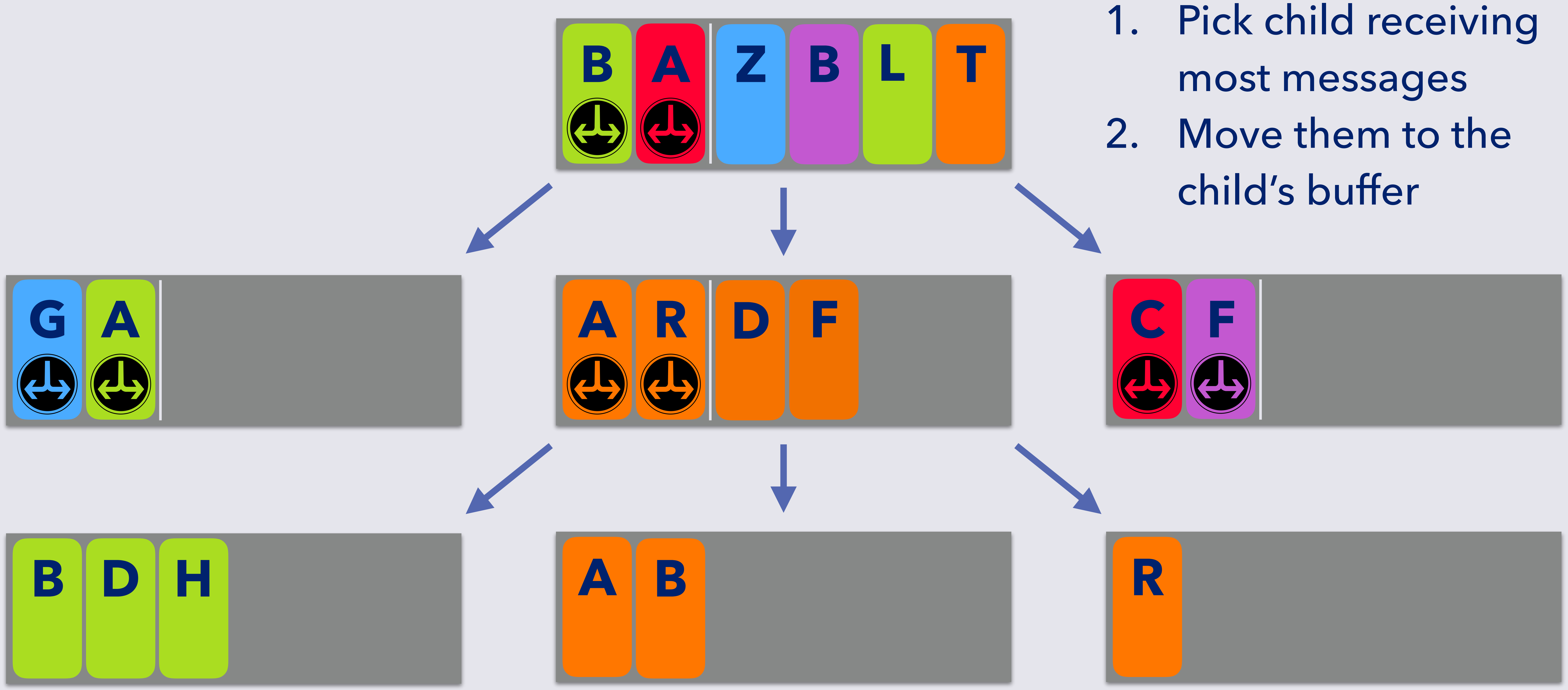
When a buffer is full:

1. Pick child receiving most messages
2. Move them to the child's buffer

B$^{\varepsilon}$-Trees



A B$^{\varepsilon}$-tree is a search tree
(like a B-tree)

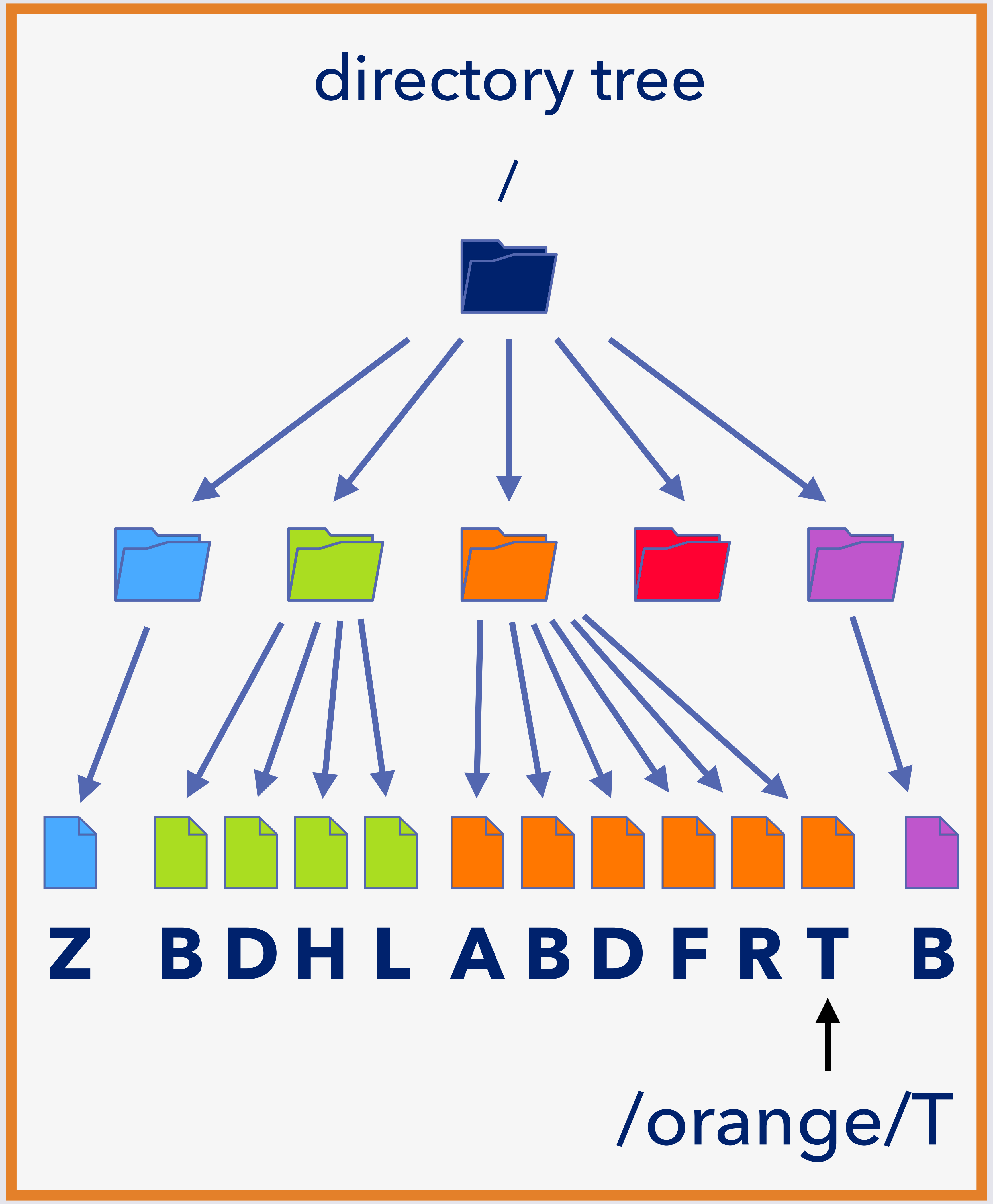


Inserts get put in
the root buffer

When a buffer is full:

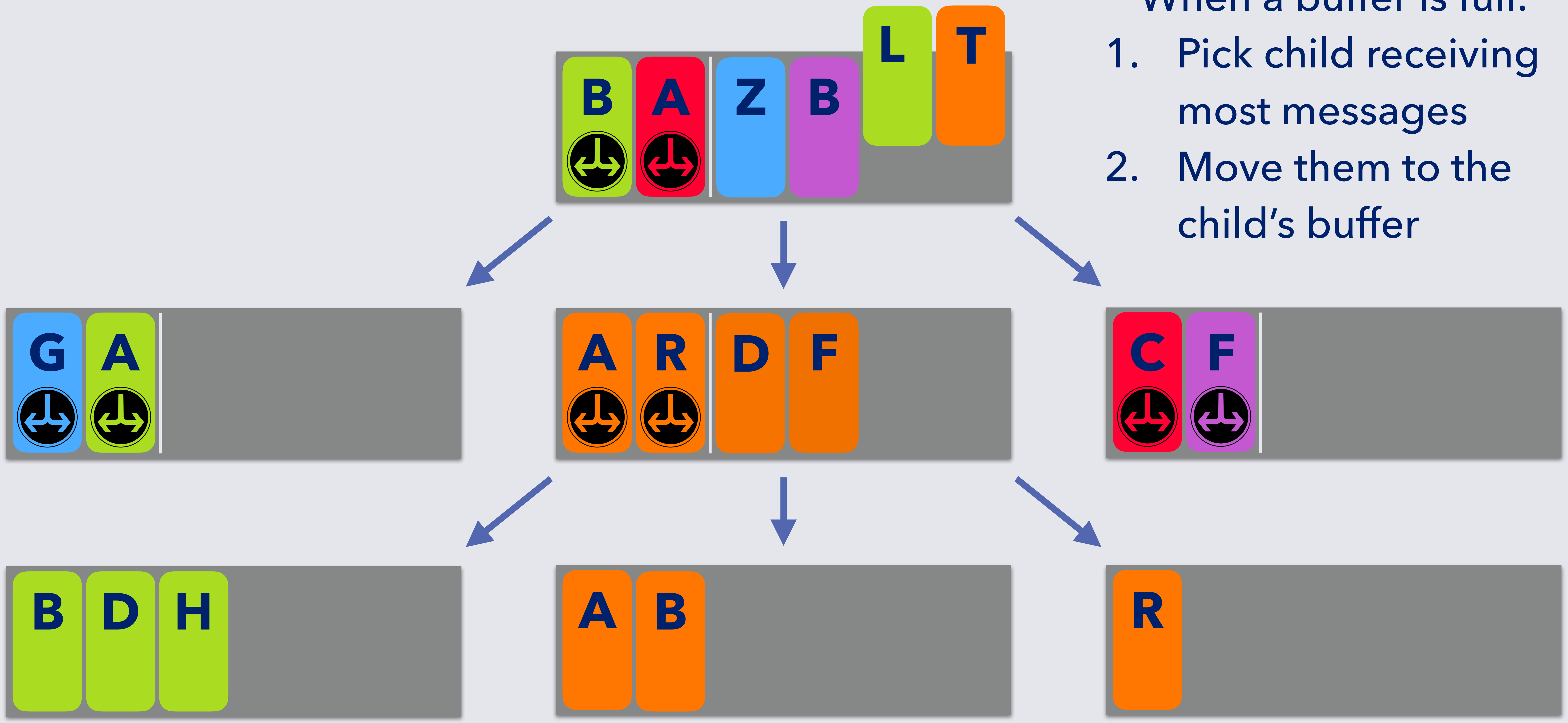
1. Pick child receiving most messages
2. Move them to the child's buffer

B$^{\varepsilon}$-Trees



A B$^{\varepsilon}$-tree is a search tree (like a B-tree)

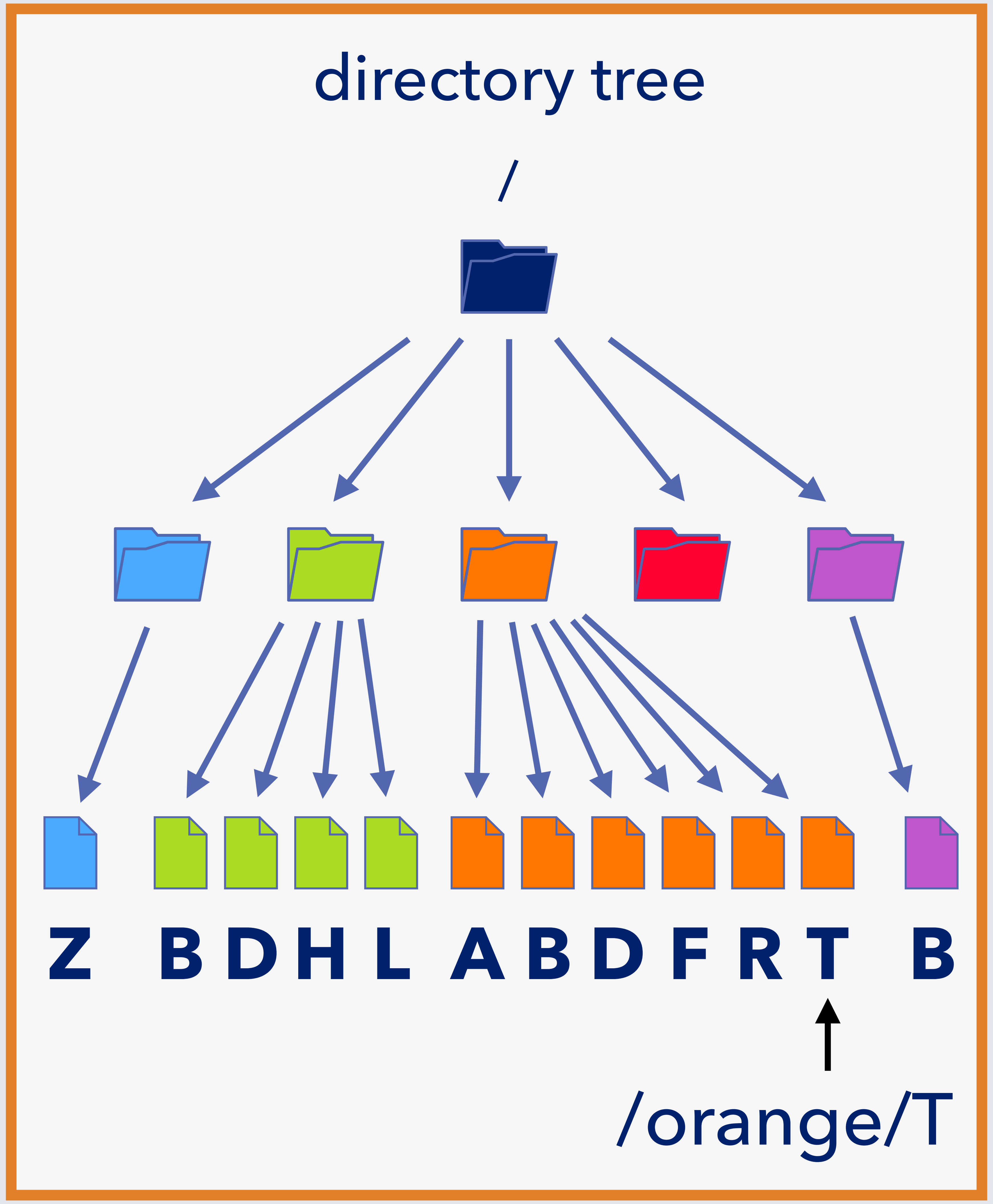
Inserts get put in the root buffer



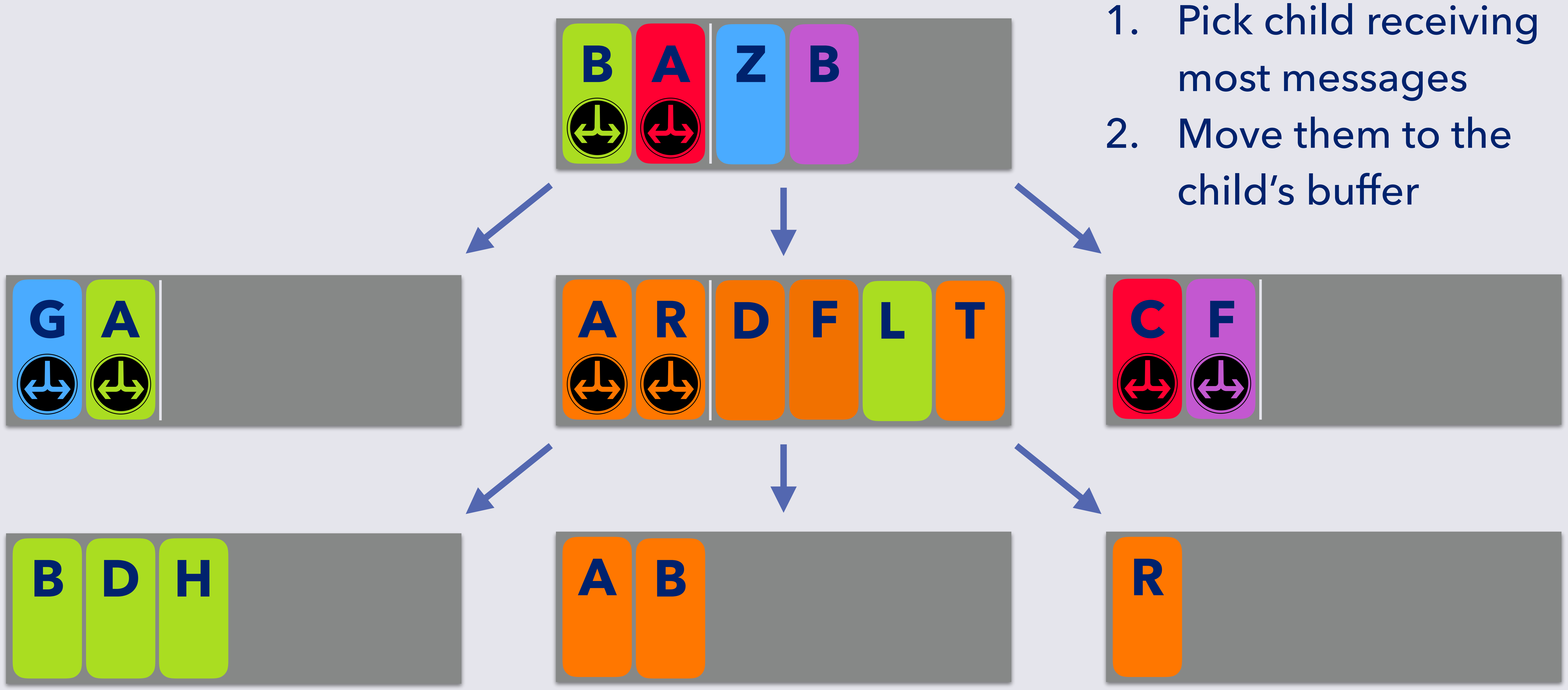
When a buffer is full:

1. Pick child receiving most messages
2. Move them to the child's buffer

B^{ε}-Trees



A B^{ε}-tree is a search tree
(like a B-tree)

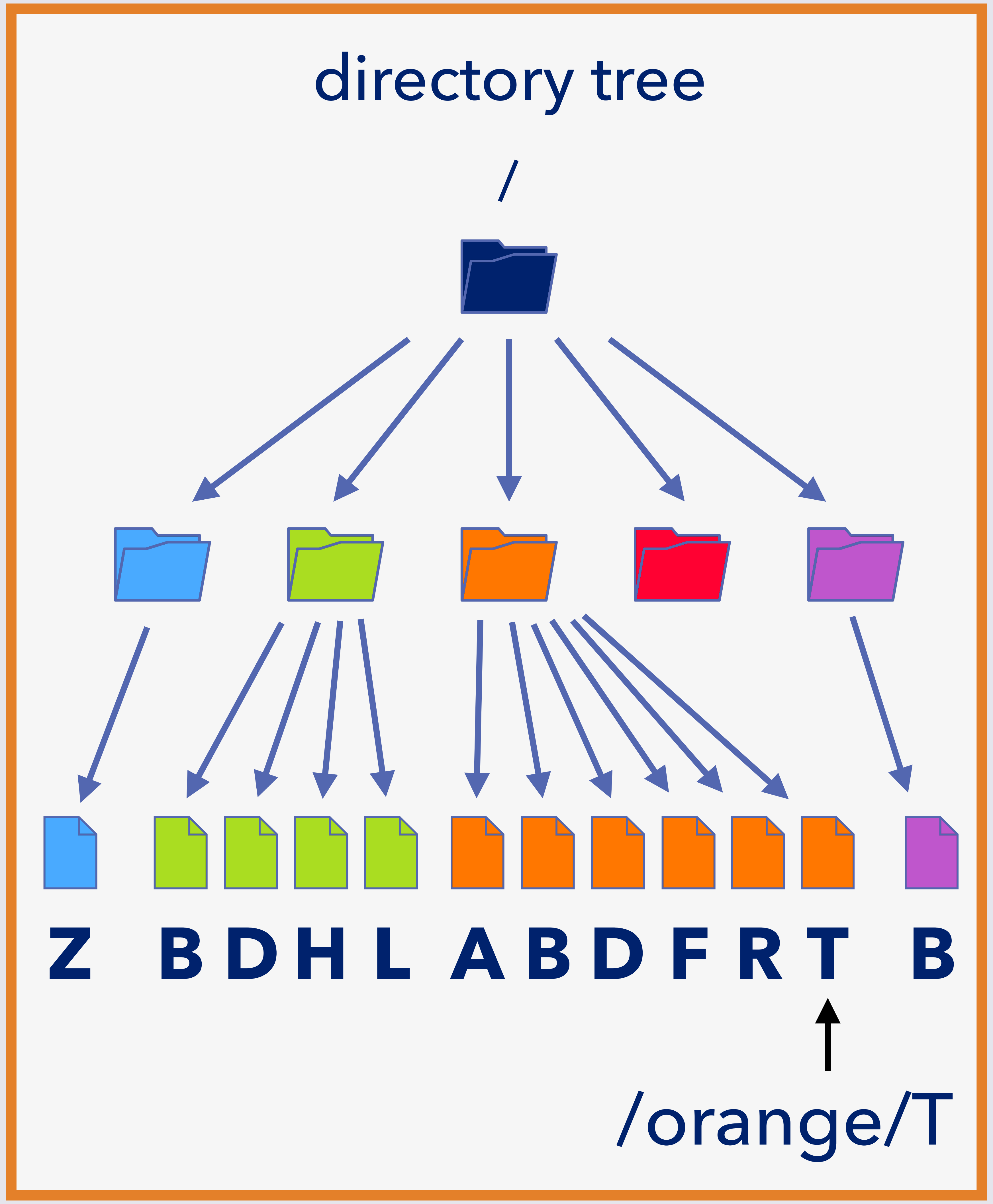


Inserts get put in the root buffer

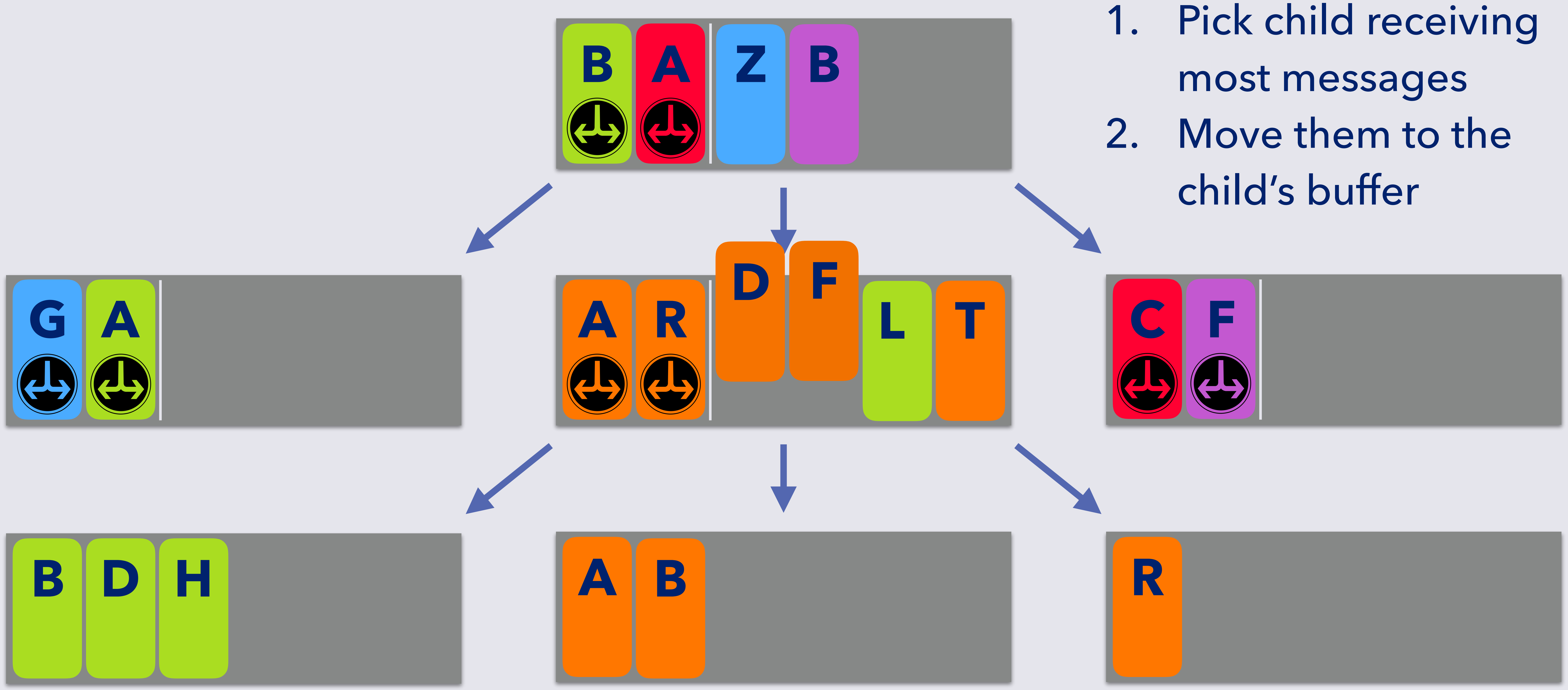
When a buffer is full:

1. Pick child receiving most messages
2. Move them to the child's buffer

B$^{\varepsilon}$-Trees



A B$^{\varepsilon}$-tree is a search tree (like a B-tree)

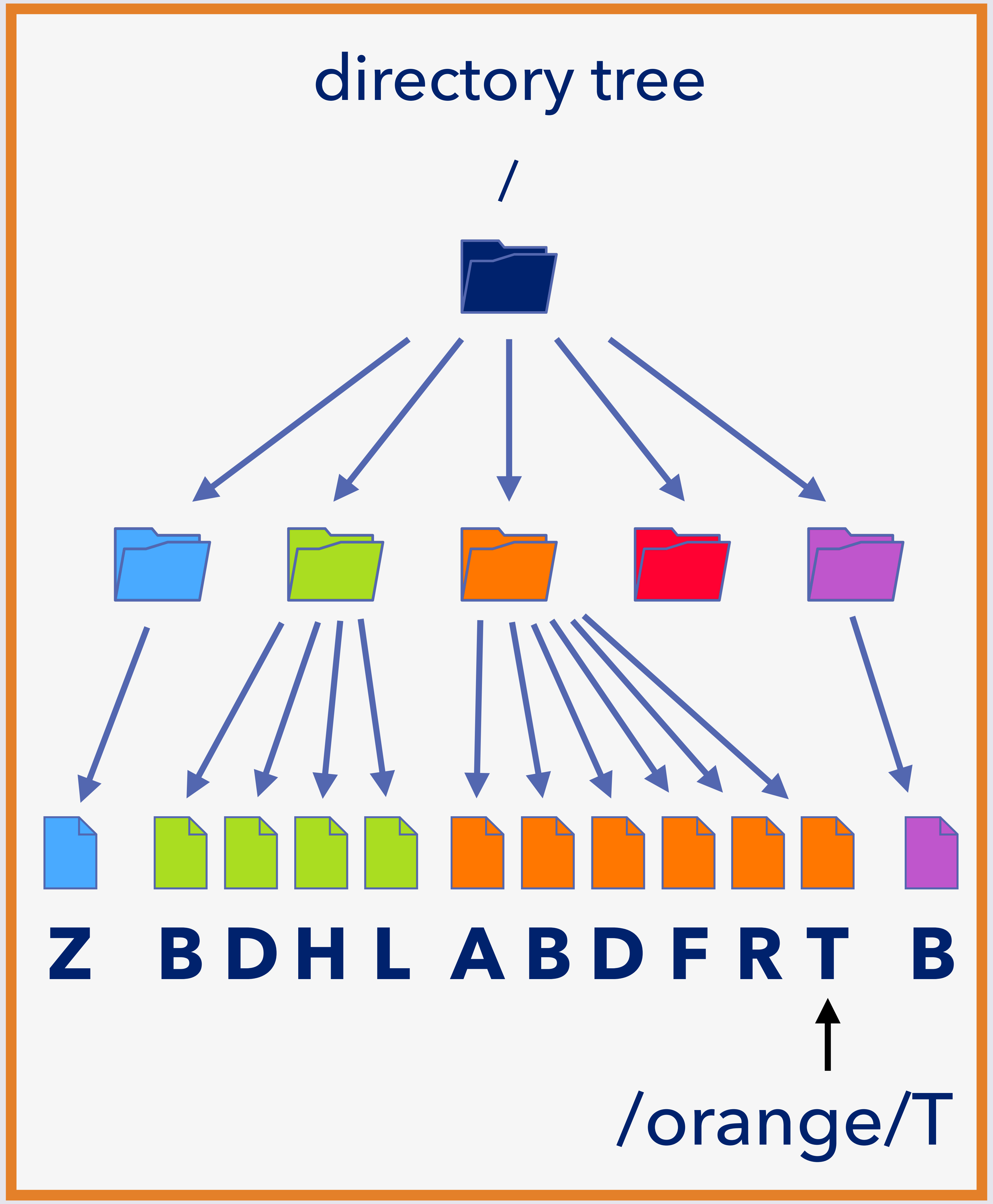


Inserts get put in the root buffer

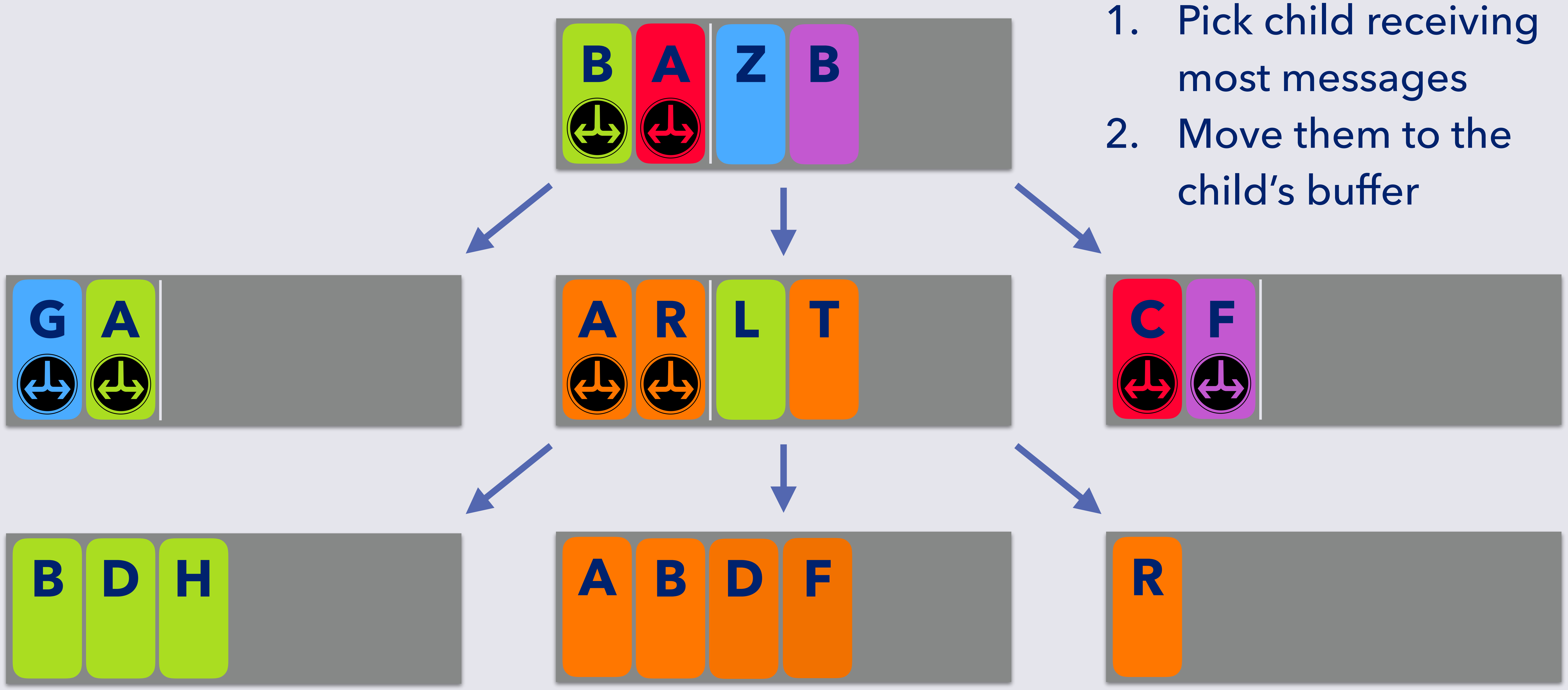
When a buffer is full:

1. Pick child receiving most messages
2. Move them to the child's buffer

B$^{\varepsilon}$-Trees



A B$^{\varepsilon}$-tree is a search tree (like a B-tree)

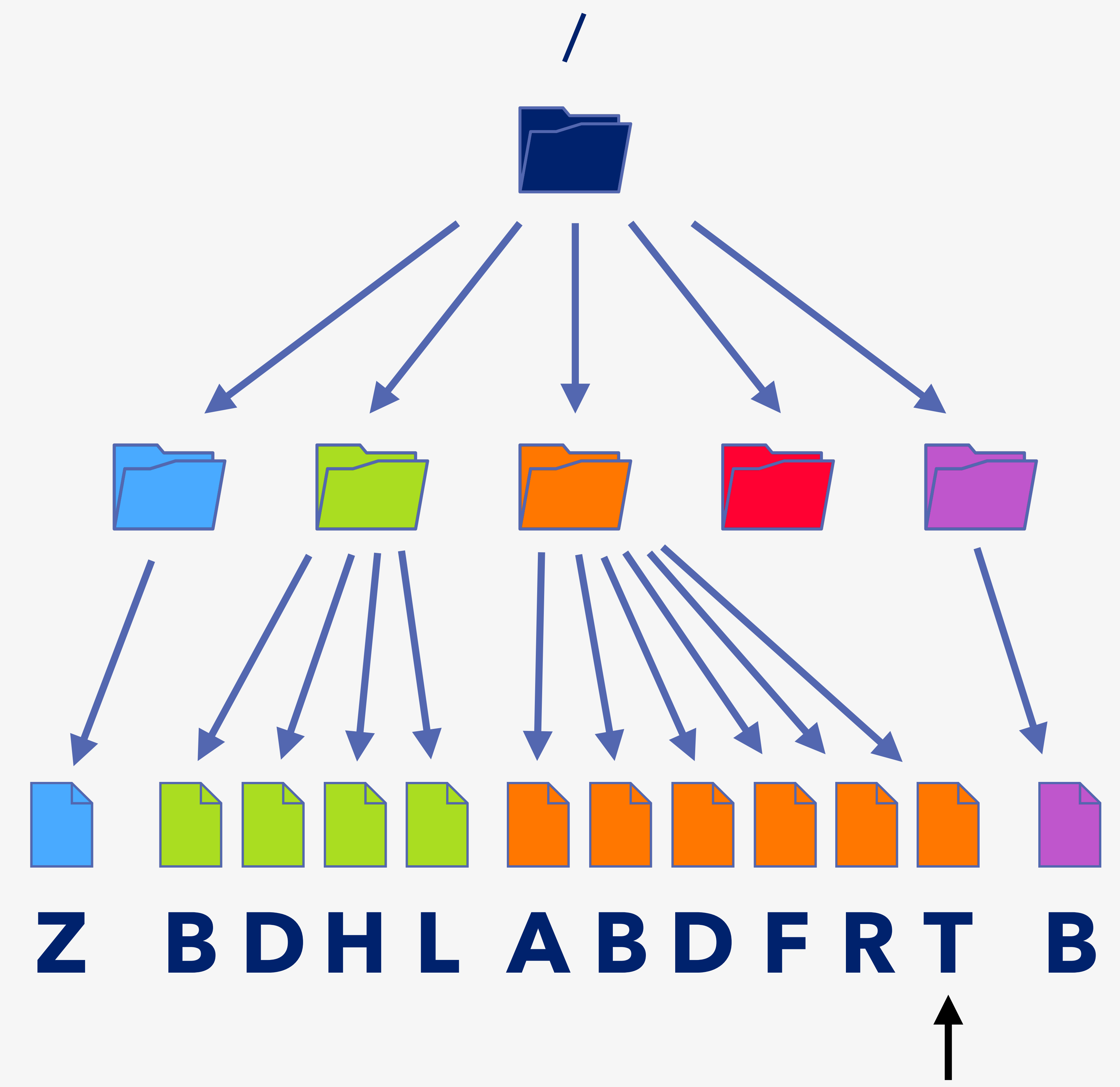


Inserts get put in the root buffer

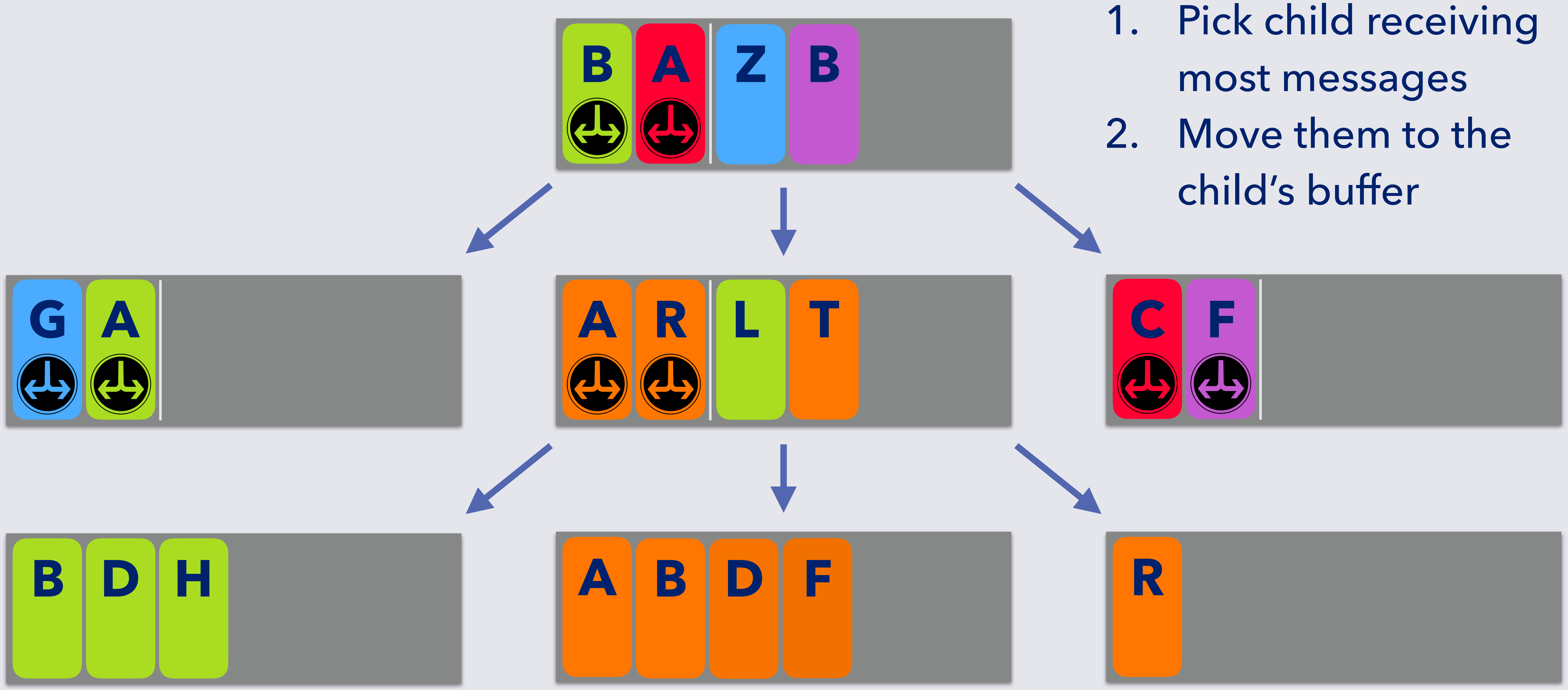
When a buffer is full:

1. Pick child receiving most messages
2. Move them to the child's buffer

Bε-Trees



A Bε-tree is a search tree (like a B-tree)



Inserts get put in the root buffer

When a buffer is full:

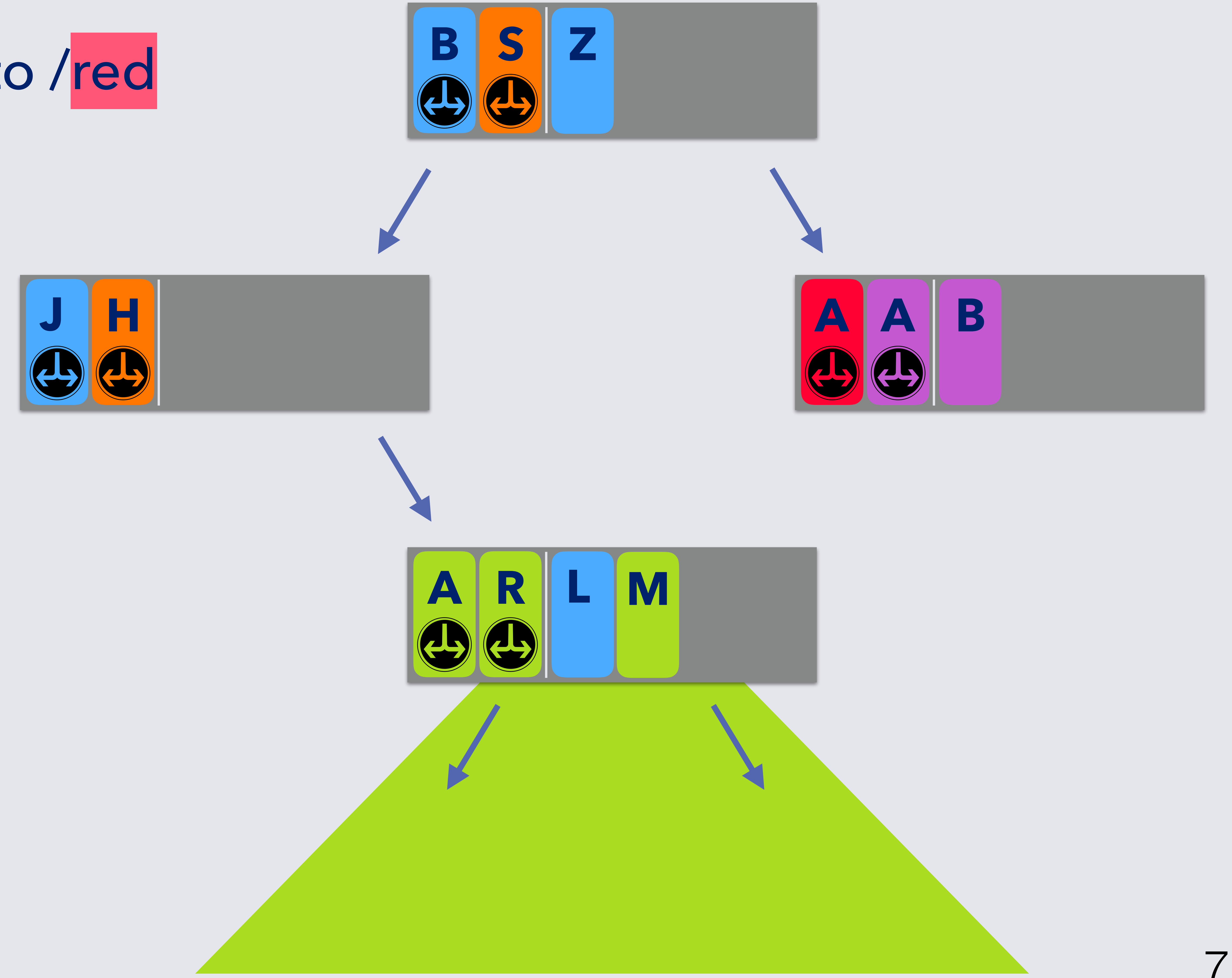
1. Pick child receiving most messages
2. Move them to the child's buffer

Key Insight: Each flush applies many small changes

Logical Copy with B$^{\varepsilon}$-DAGs

Logical Copy with Bε-DAGs

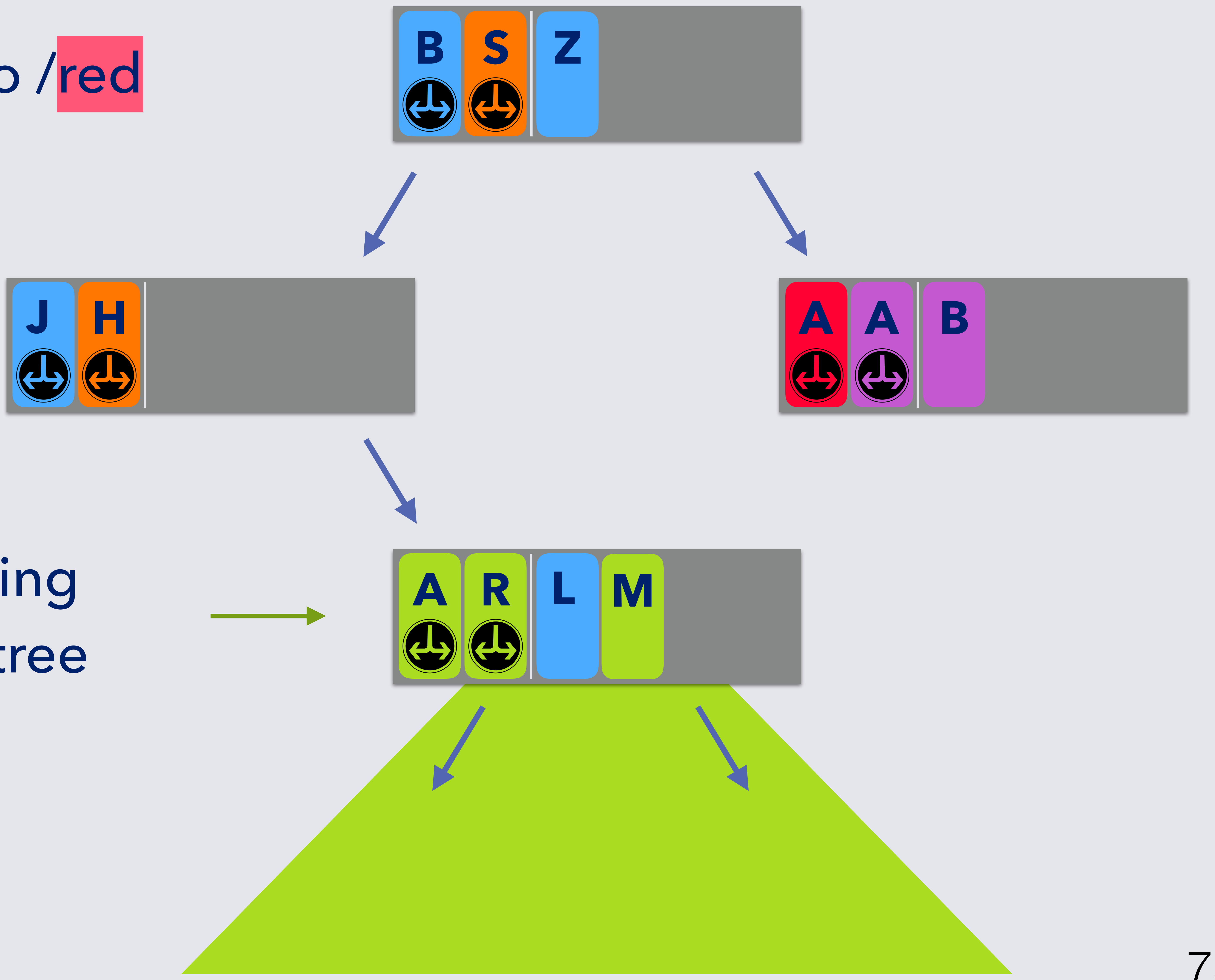
Copy /green to /red



Logical Copy with Bε-DAGs

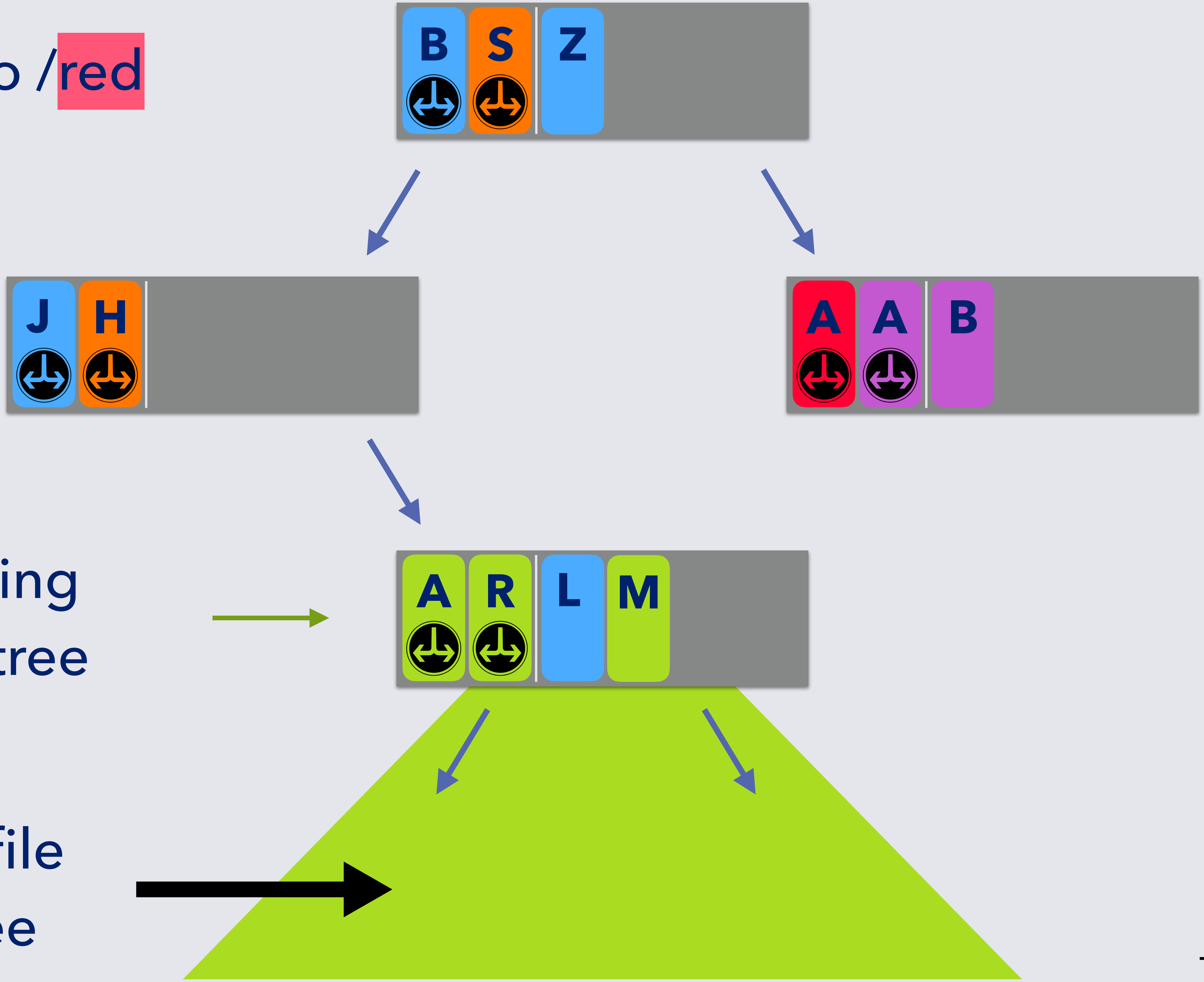
Copy /green to /red

Node covering /green/ subtree



Logical Copy with Bε-DAGs

Copy /green to /red



Node covering /green/ subtree

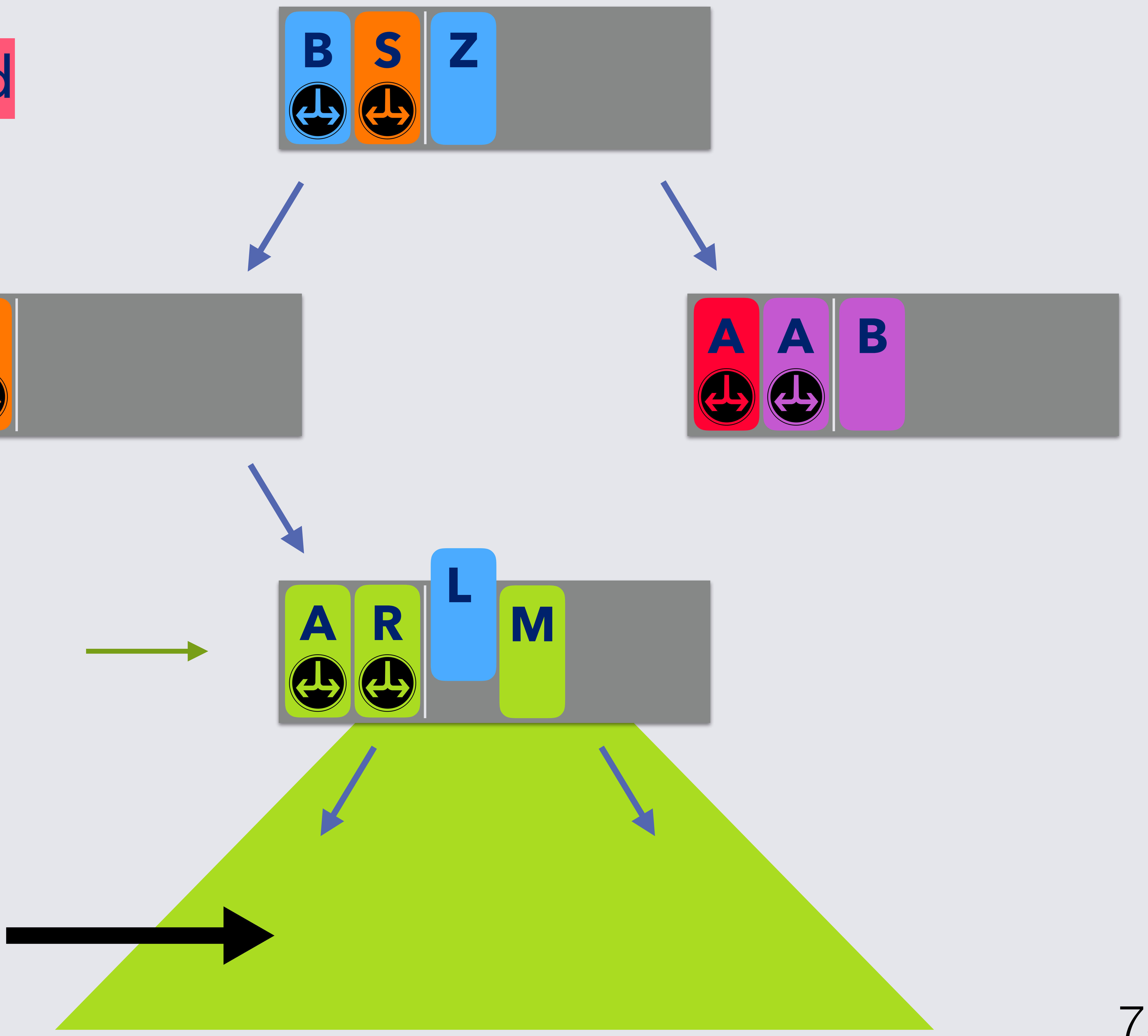
Every /green/* file is in this subtree

Logical Copy with Bε-DAGs

Copy /green to /red

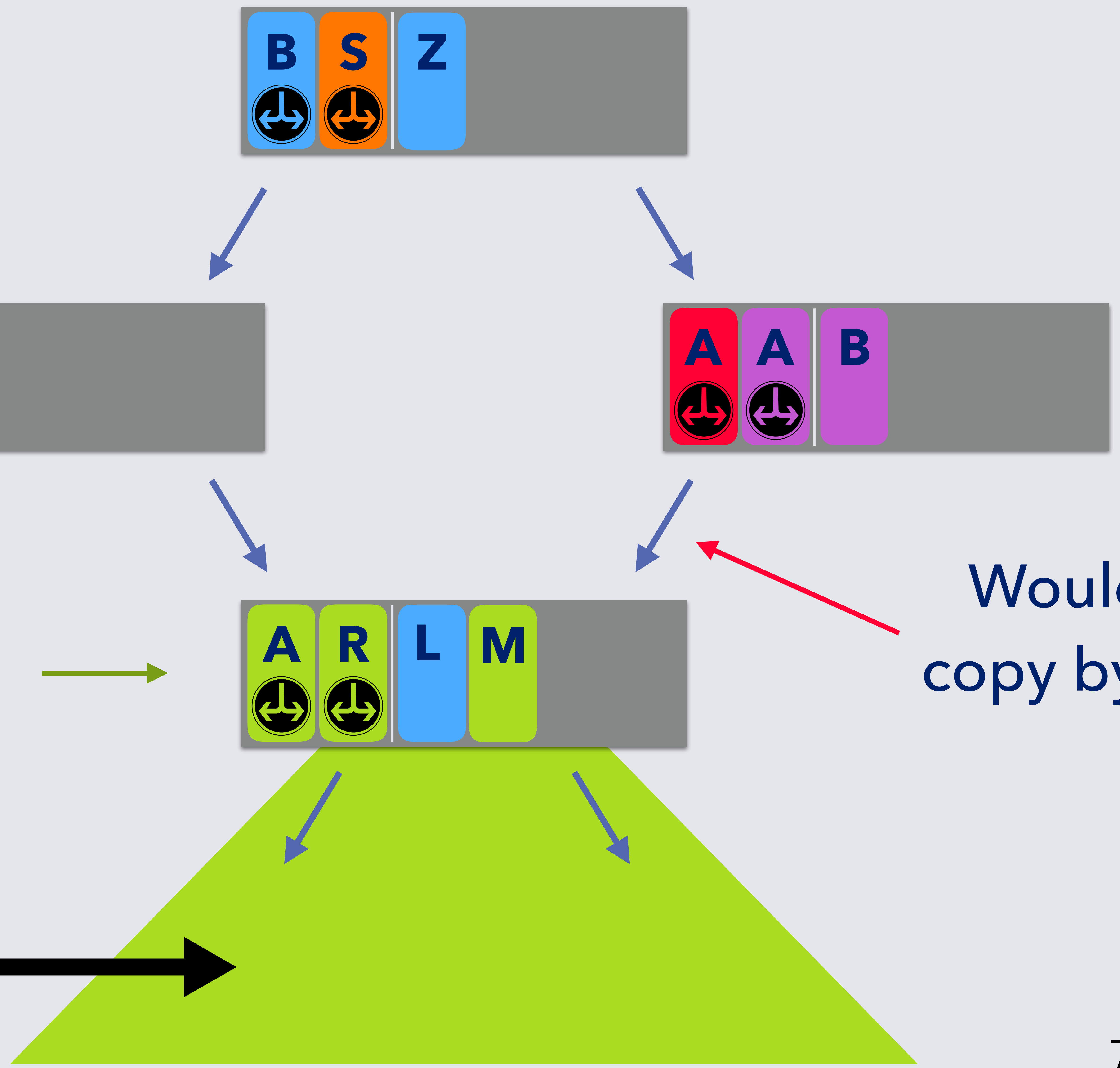
Node covering /green/ subtree

Every /green/* file is in this subtree



Logical Copy with Bε-DAGs

Copy /green to /red



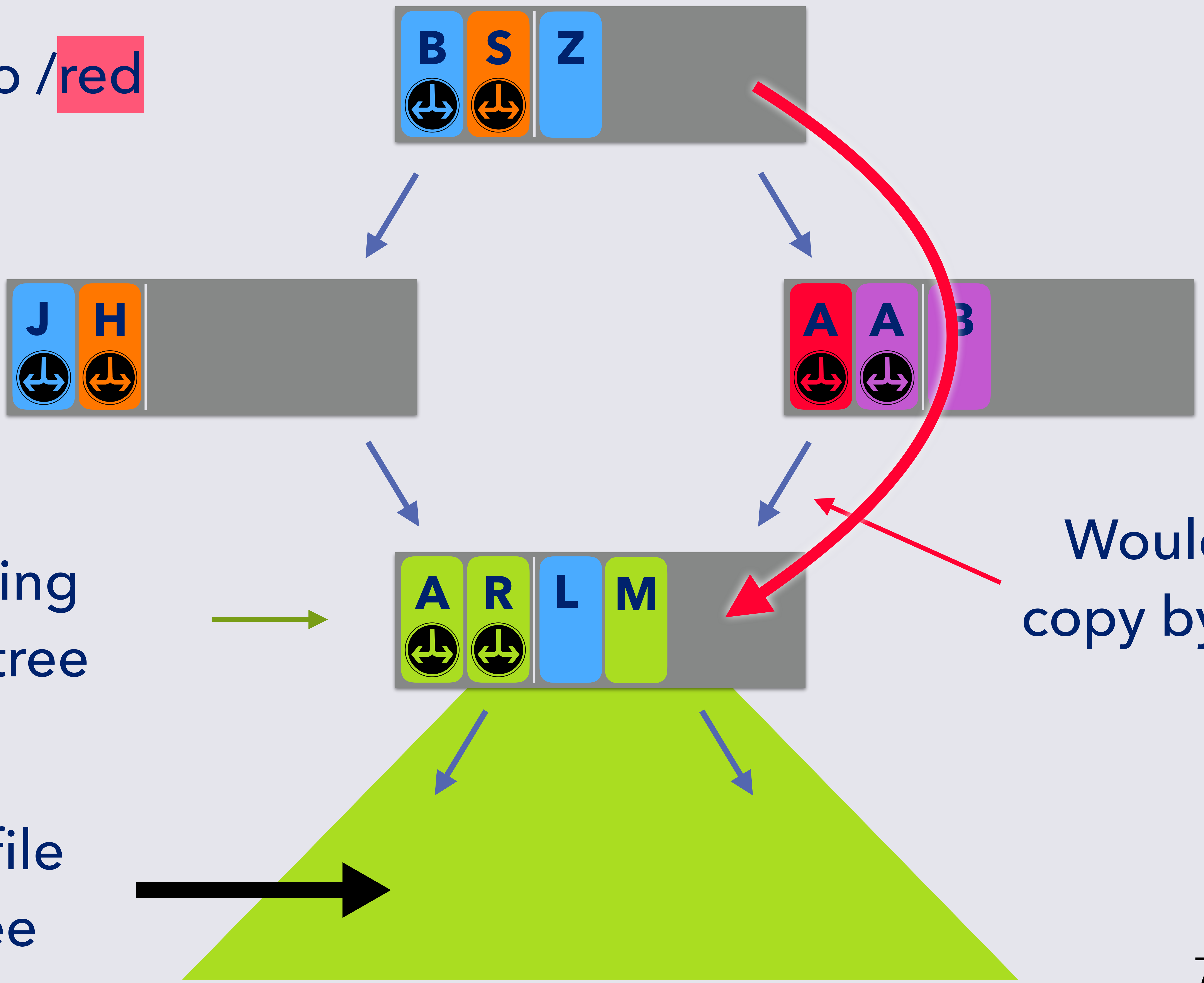
Node covering /green/ subtree

Would like to logically copy by adding this edge

Every /green/* file is in this subtree

Logical Copy with Bε-DAGs

Copy /green to /red



Node covering /green/ subtree

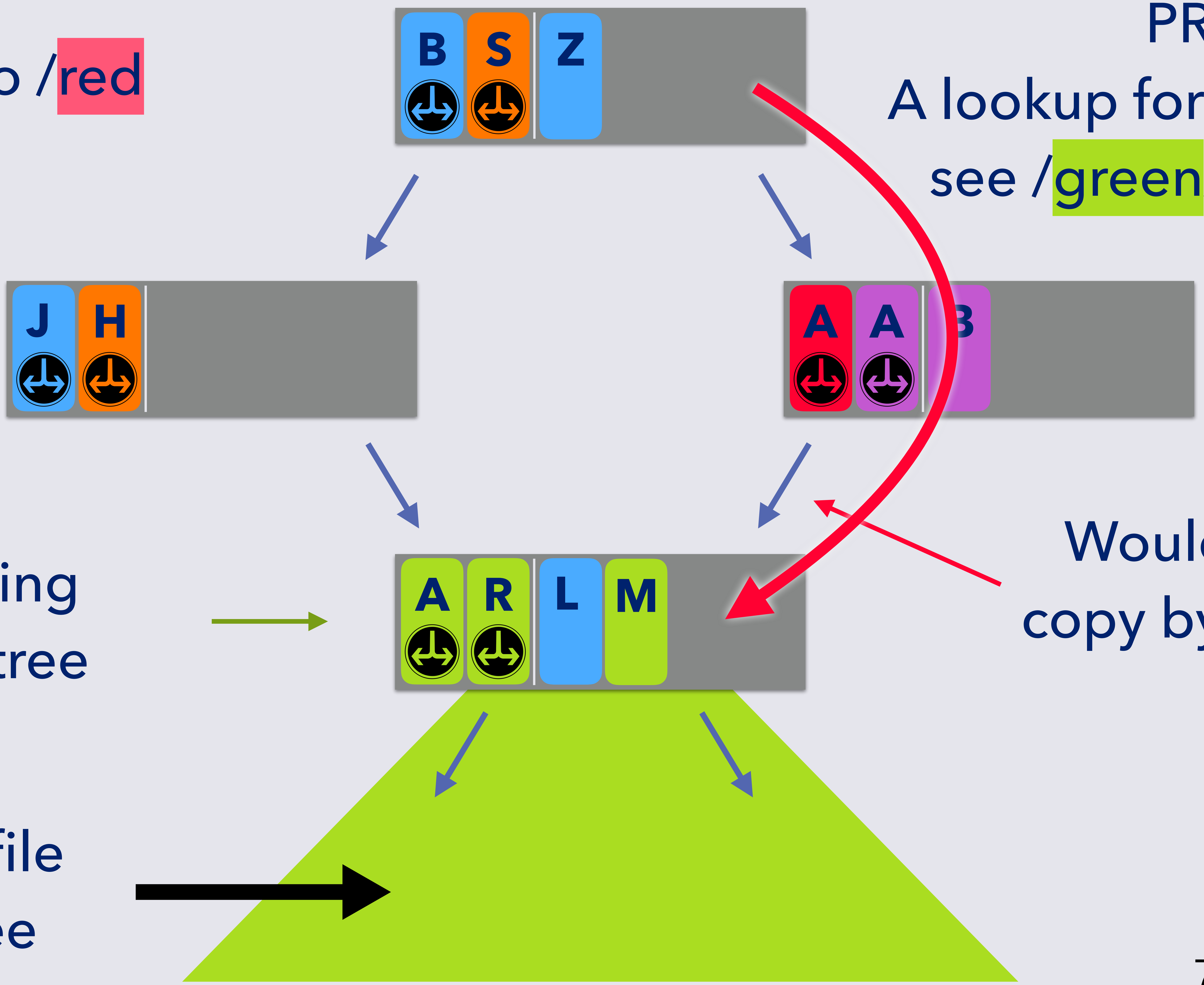
Every /green/* file is in this subtree

Would like to logically copy by adding this edge

Logical Copy with Bε-DAGs

Copy /green to /red

PROBLEM:
A lookup for /red/M is going to see /green/M in the subtree



Node covering /green/ subtree

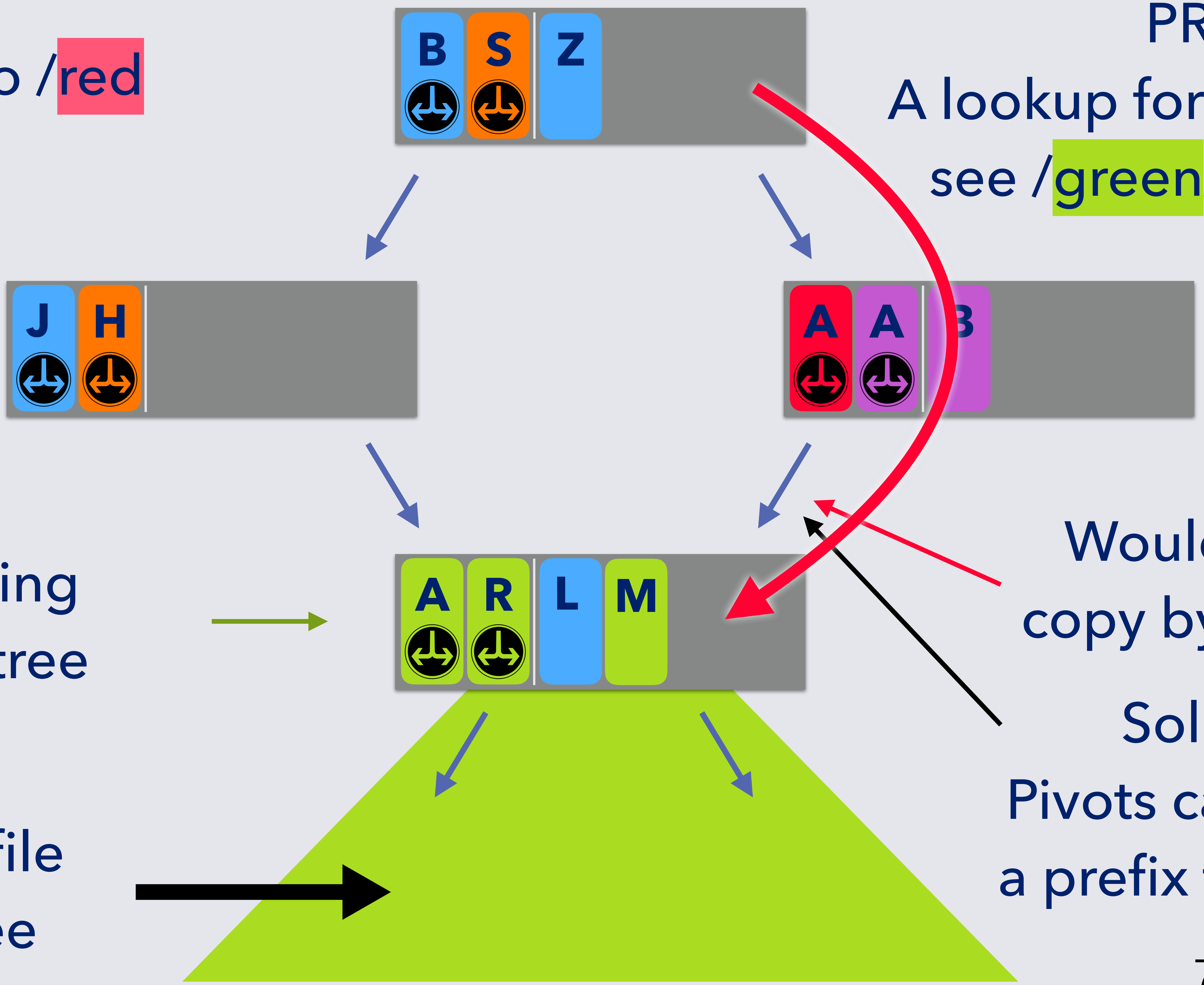
Would like to logically copy by adding this edge

Every /green/* file is in this subtree

Logical Copy with Bε-DAGs

Copy /green to /red

PROBLEM:
A lookup for /red/M is going to see /green/M in the subtree



Node covering /green/ subtree

Every /green/* file is in this subtree

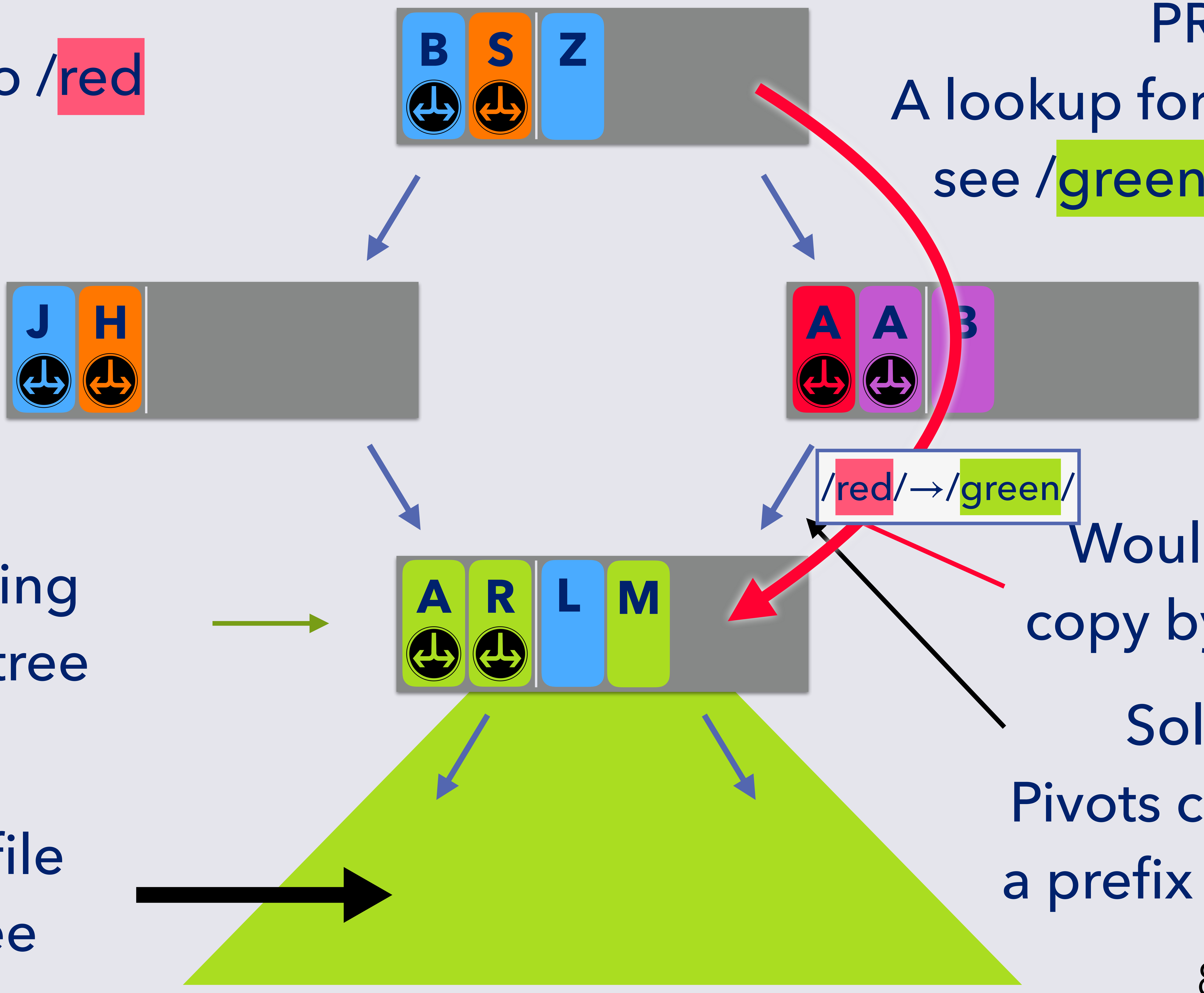
Would like to logically copy by adding this edge

Solution:
Pivots can include a prefix translation

Logical Copy with Bε-DAGs

Copy /green to /red

PROBLEM:
A lookup for /red/M is going to see /green/M in the subtree



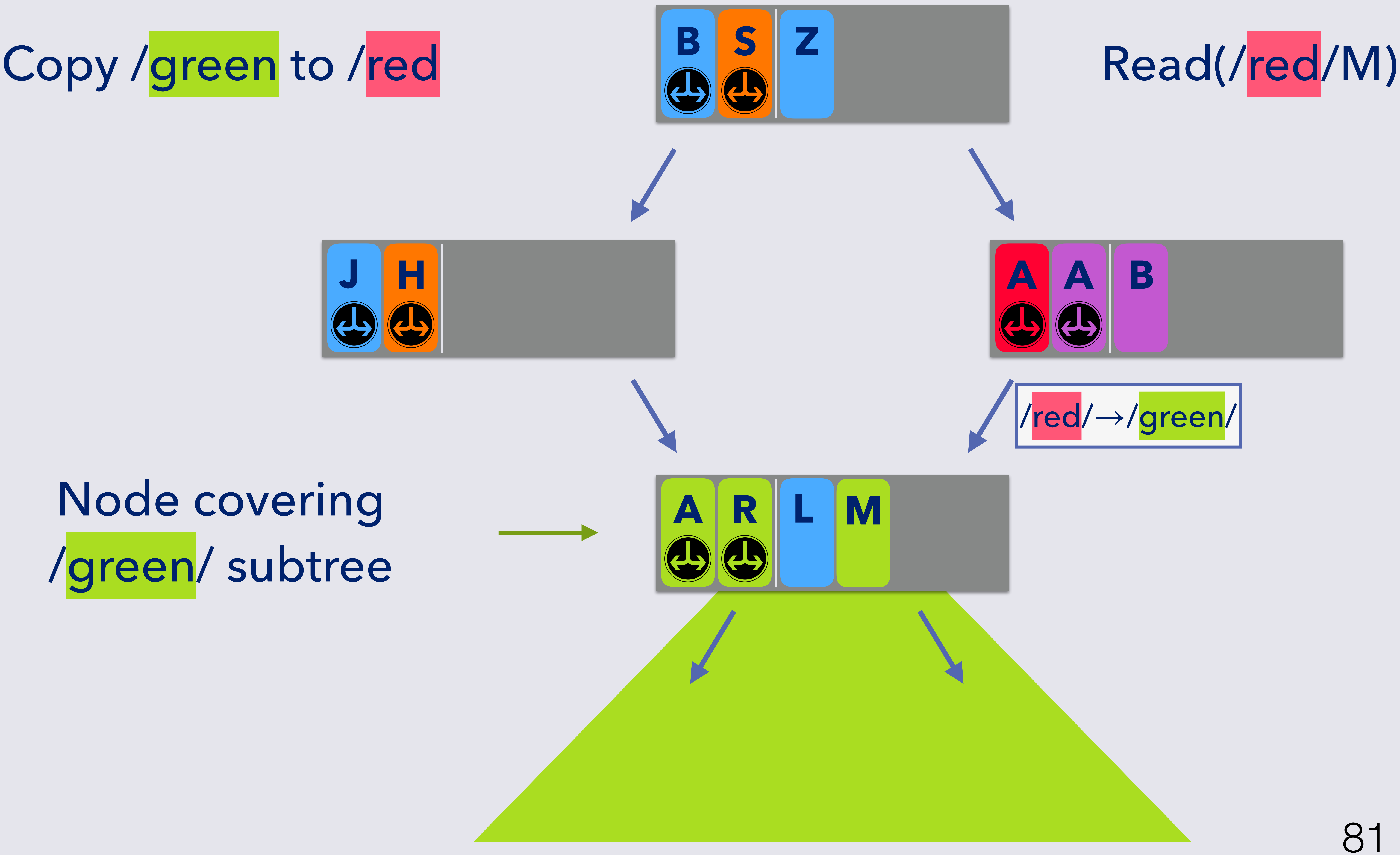
Node covering /green/ subtree

Would like to logically copy by adding this edge

Every /green/* file is in this subtree

Solution:
Pivots can include a prefix translation

Logical Copy with Bε-DAGs

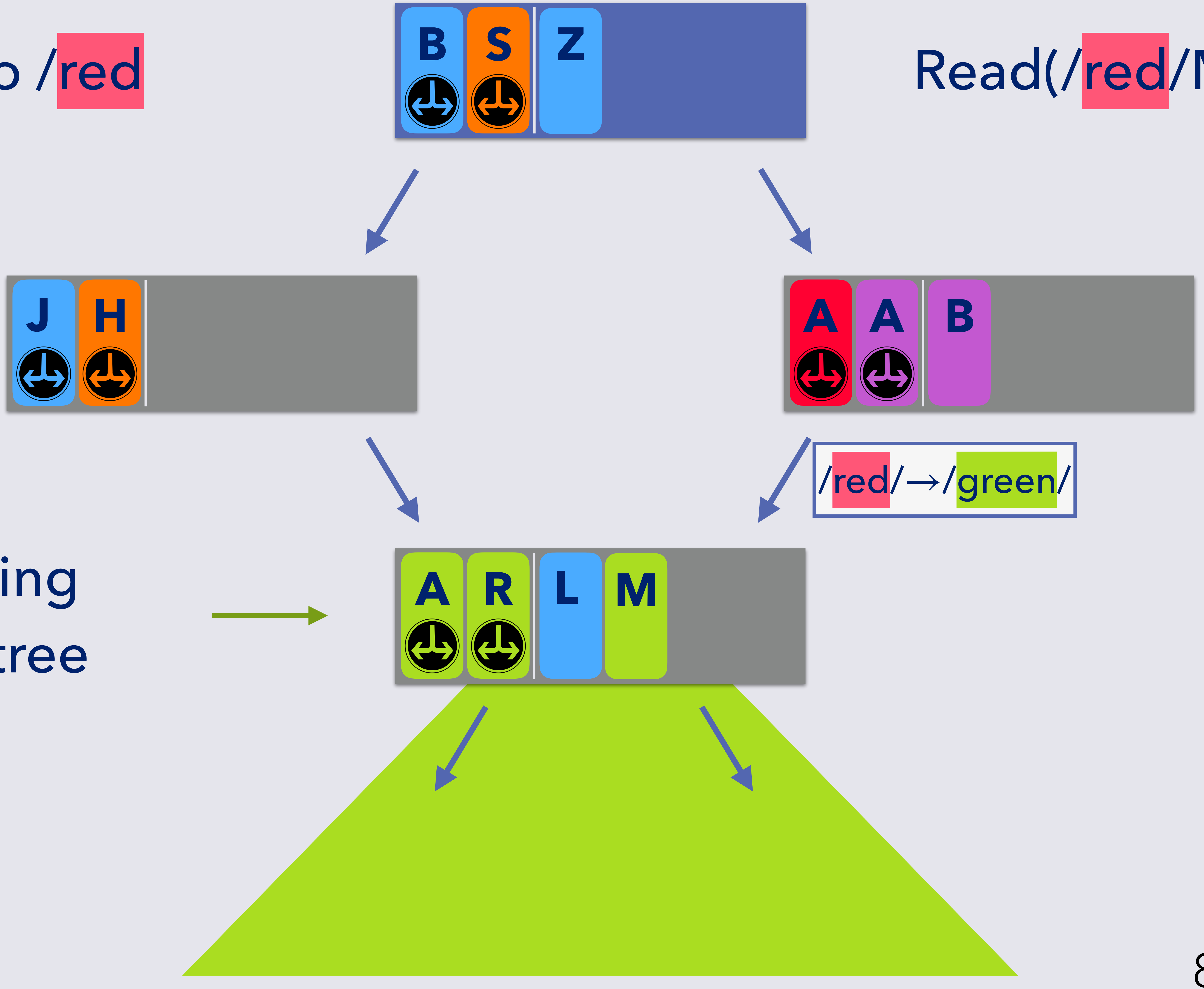


Logical Copy with Bε-DAGs

Copy /green to /red

Read(/red/M)

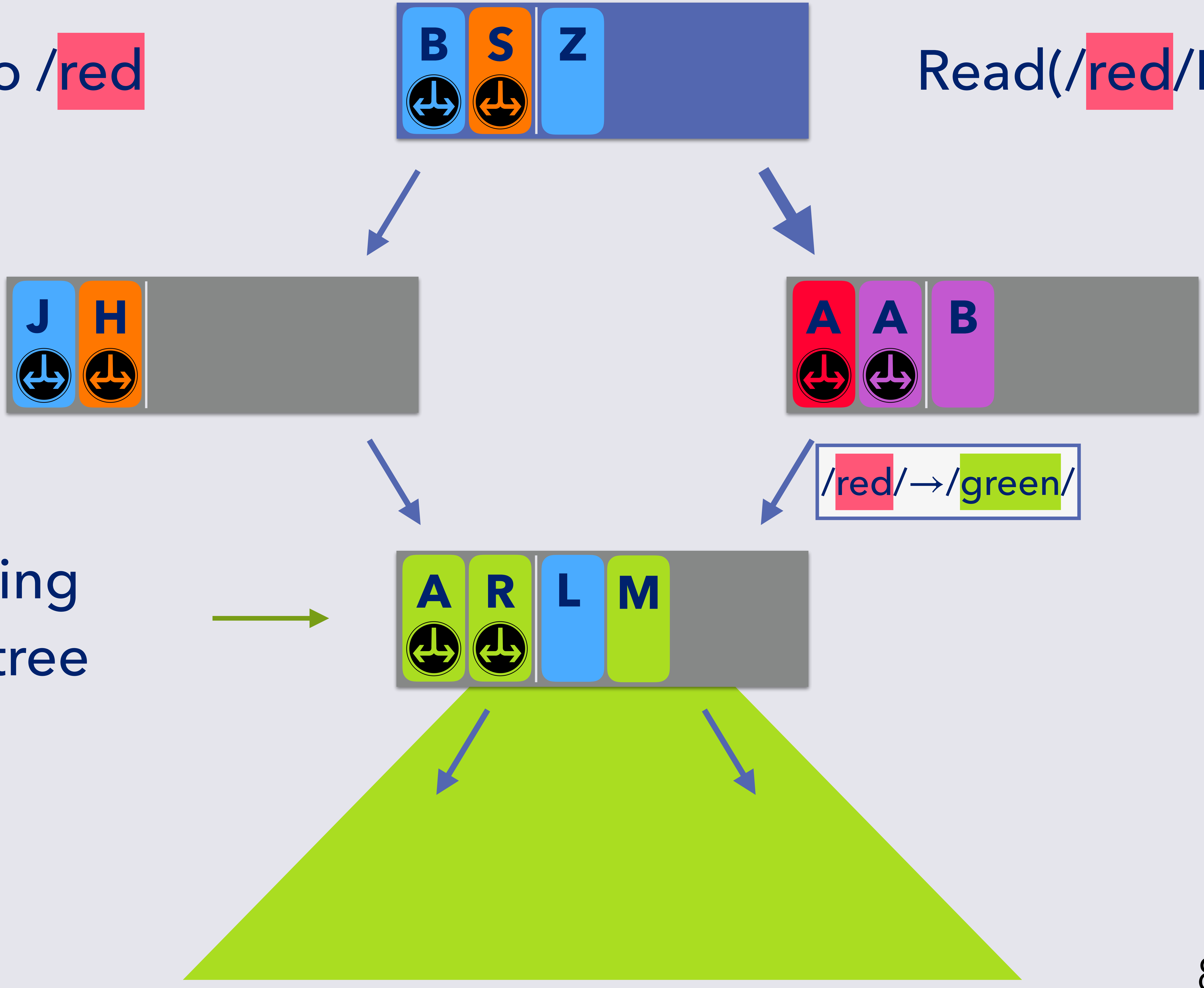
Node covering /green/ subtree



Logical Copy with Bε-DAGs

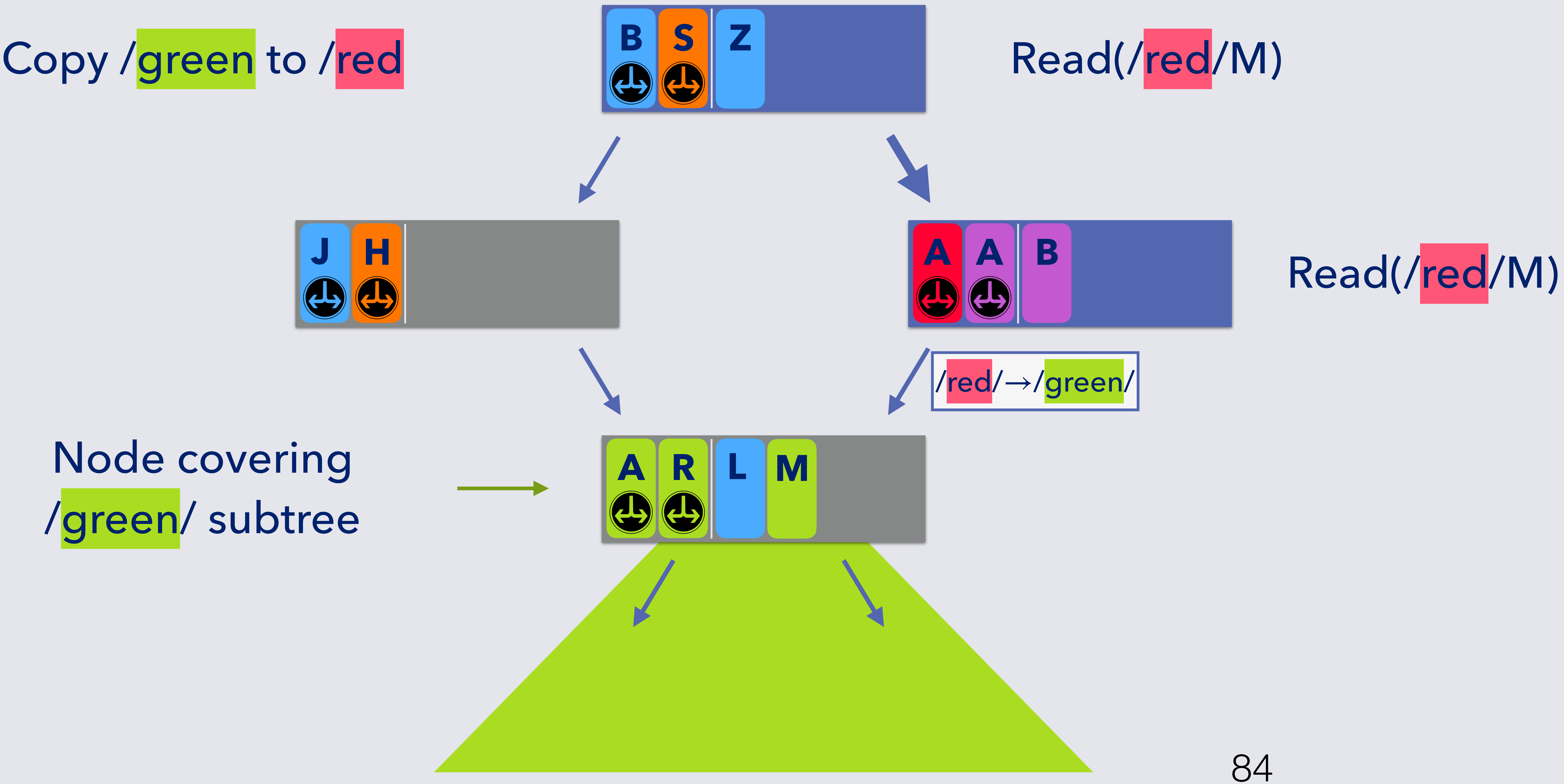
Copy /green to /red

Read(/red/M)



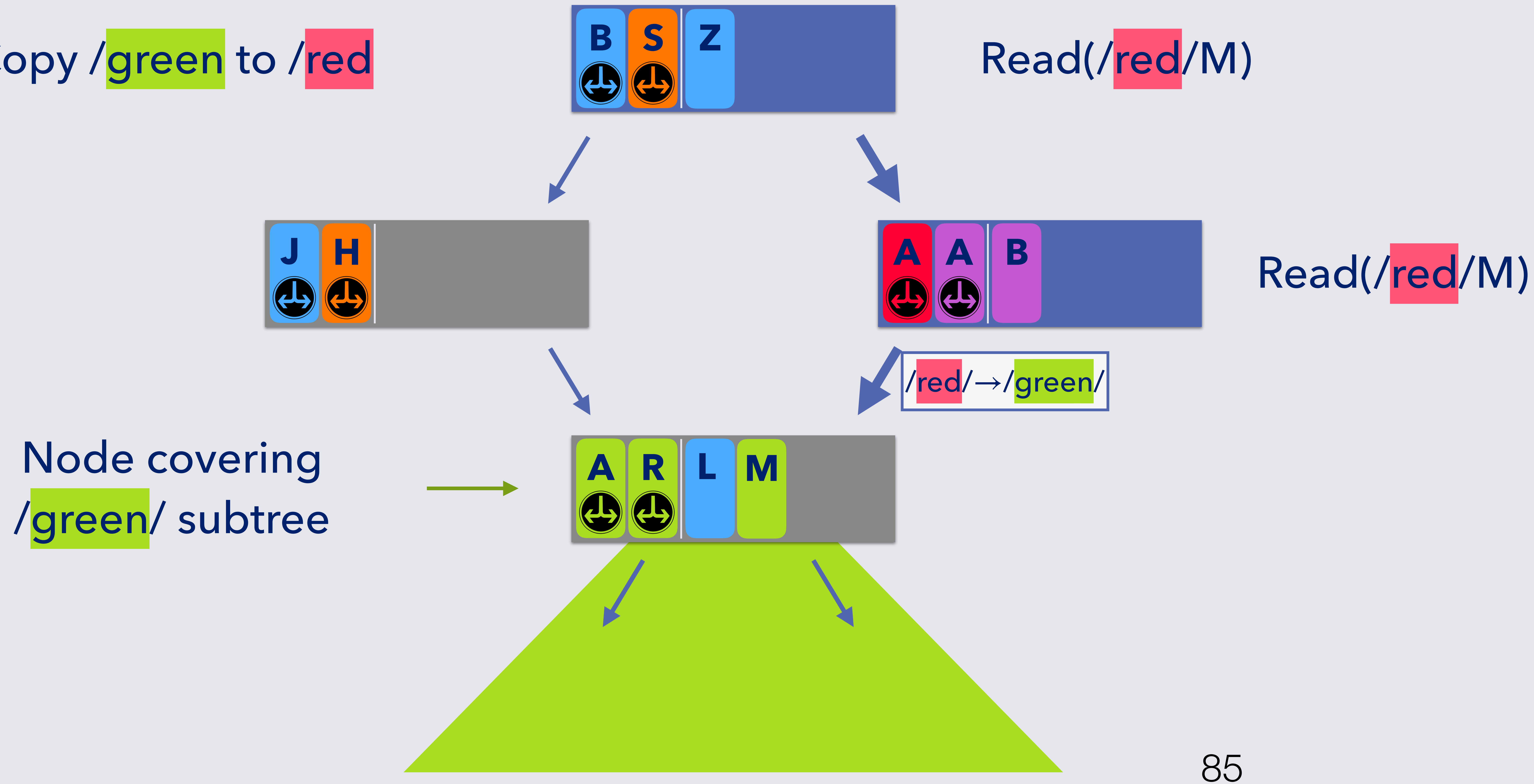
Node covering /green/ subtree

Logical Copy with Bε-DAGs



Logical Copy with Bε-DAGs

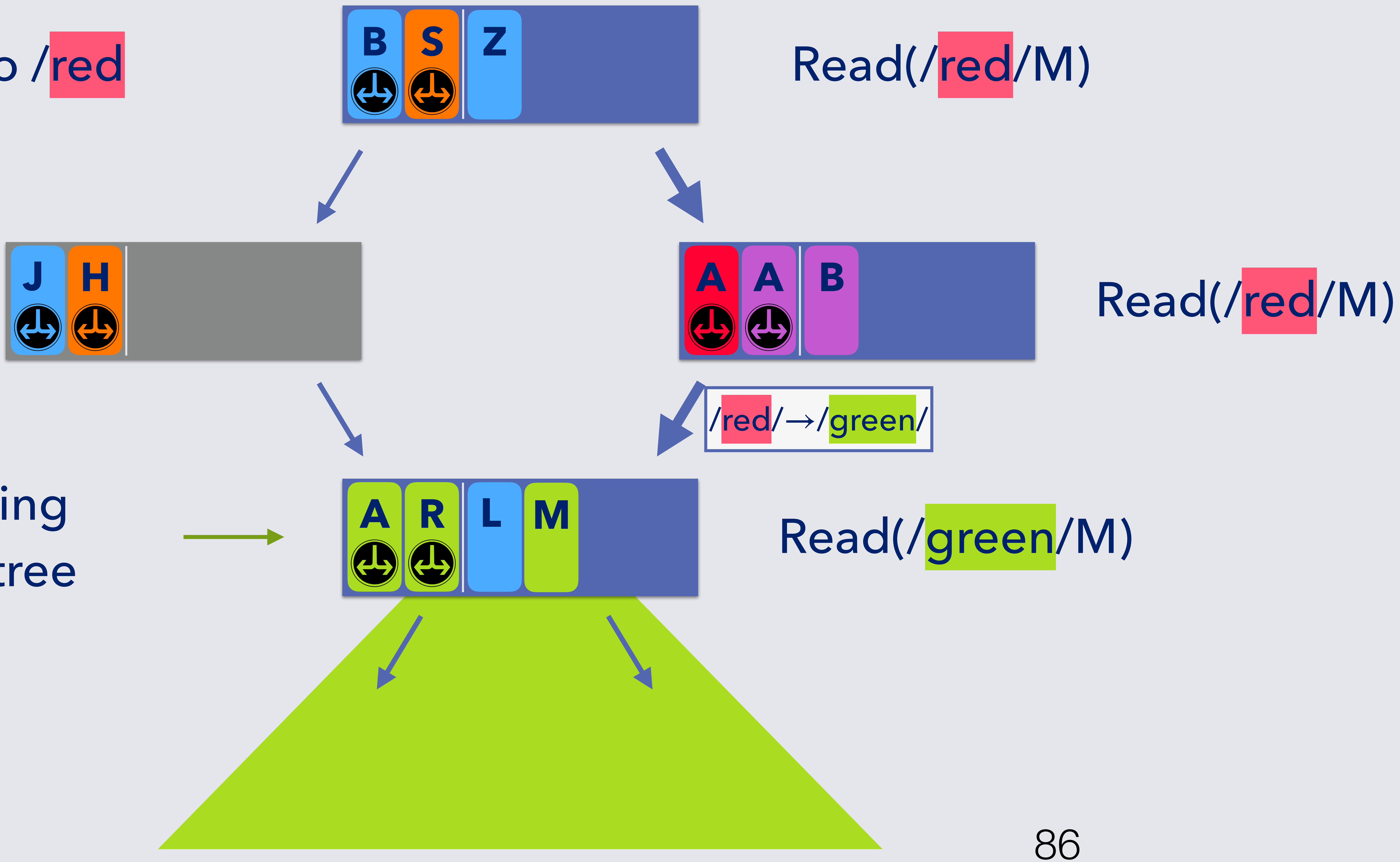
Copy /green to /red



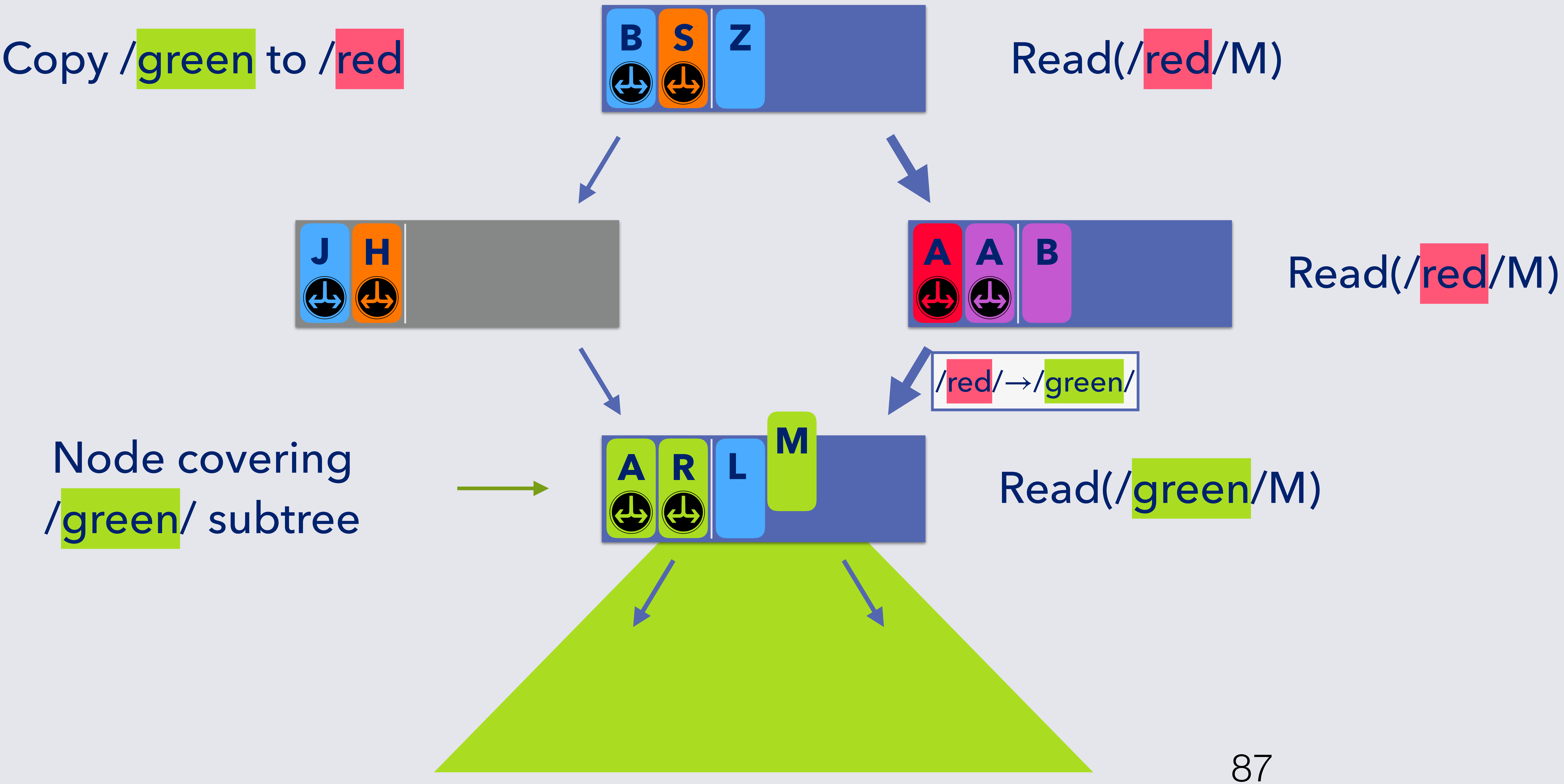
Logical Copy with Bε-DAGs

Copy /green to /red

Node covering /green/ subtree



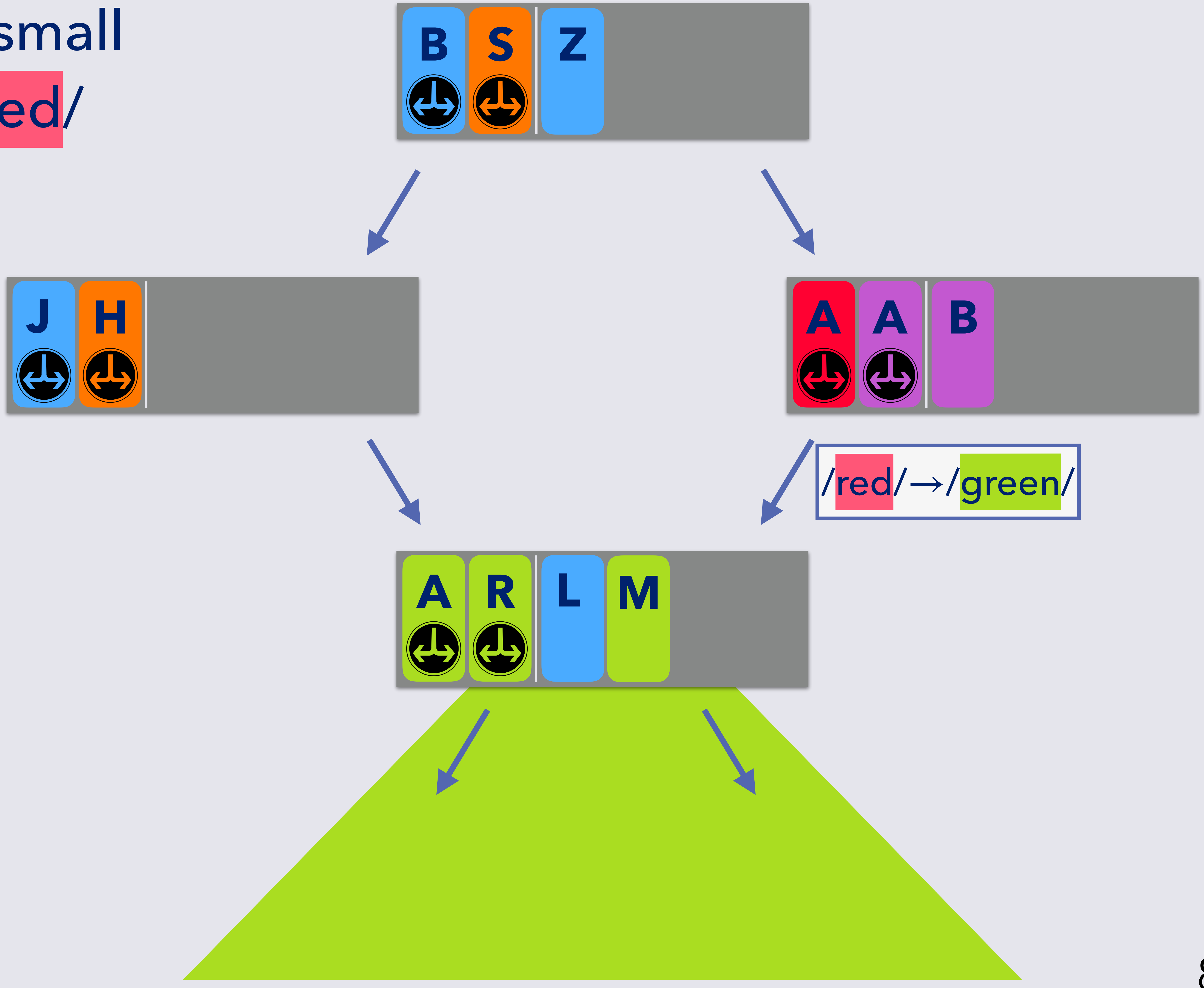
Logical Copy with Bε-DAGs



B^{ε}-DAGs and Small Writes

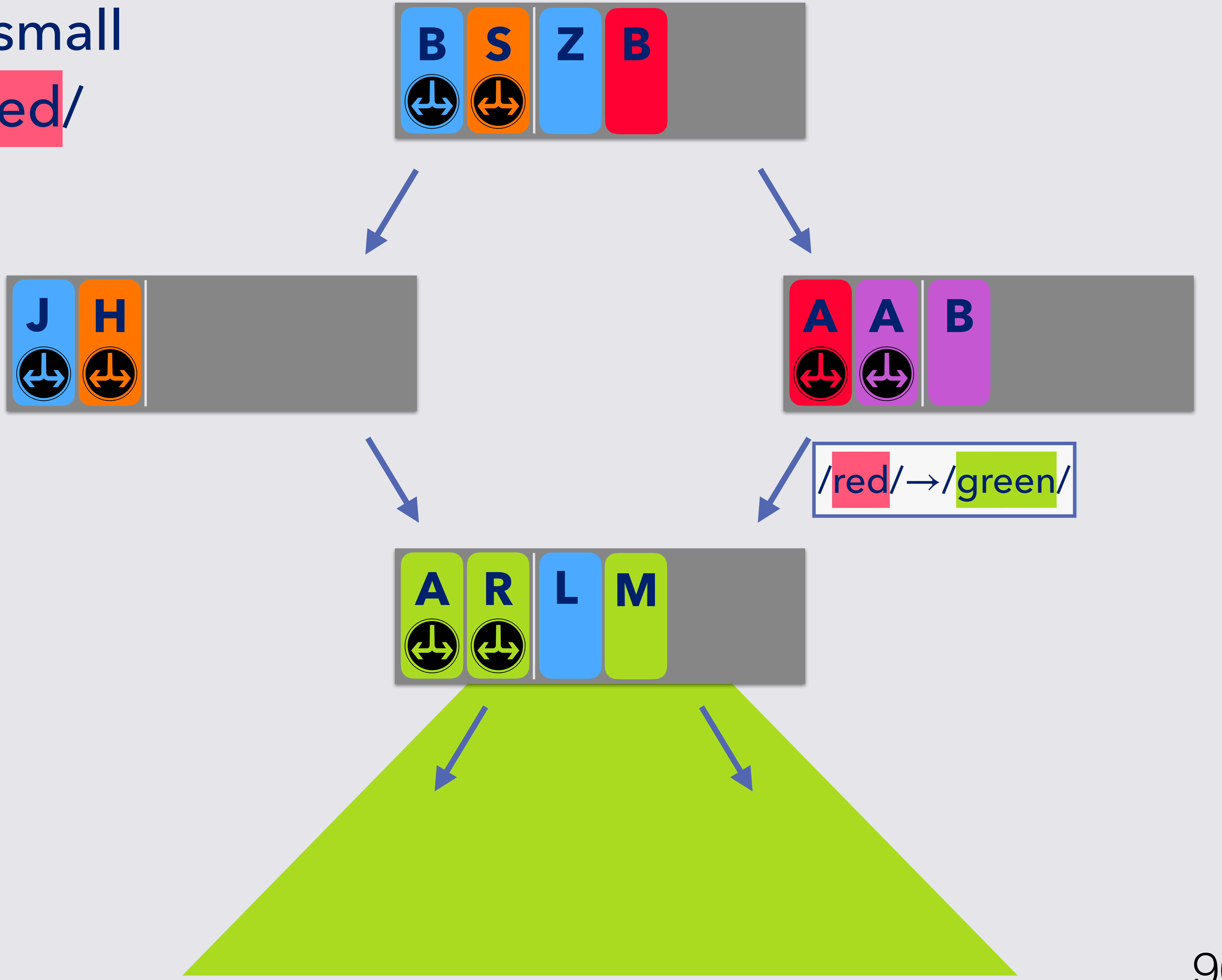
Bε-DAGs and Small Writes

Make some small writes to /red/



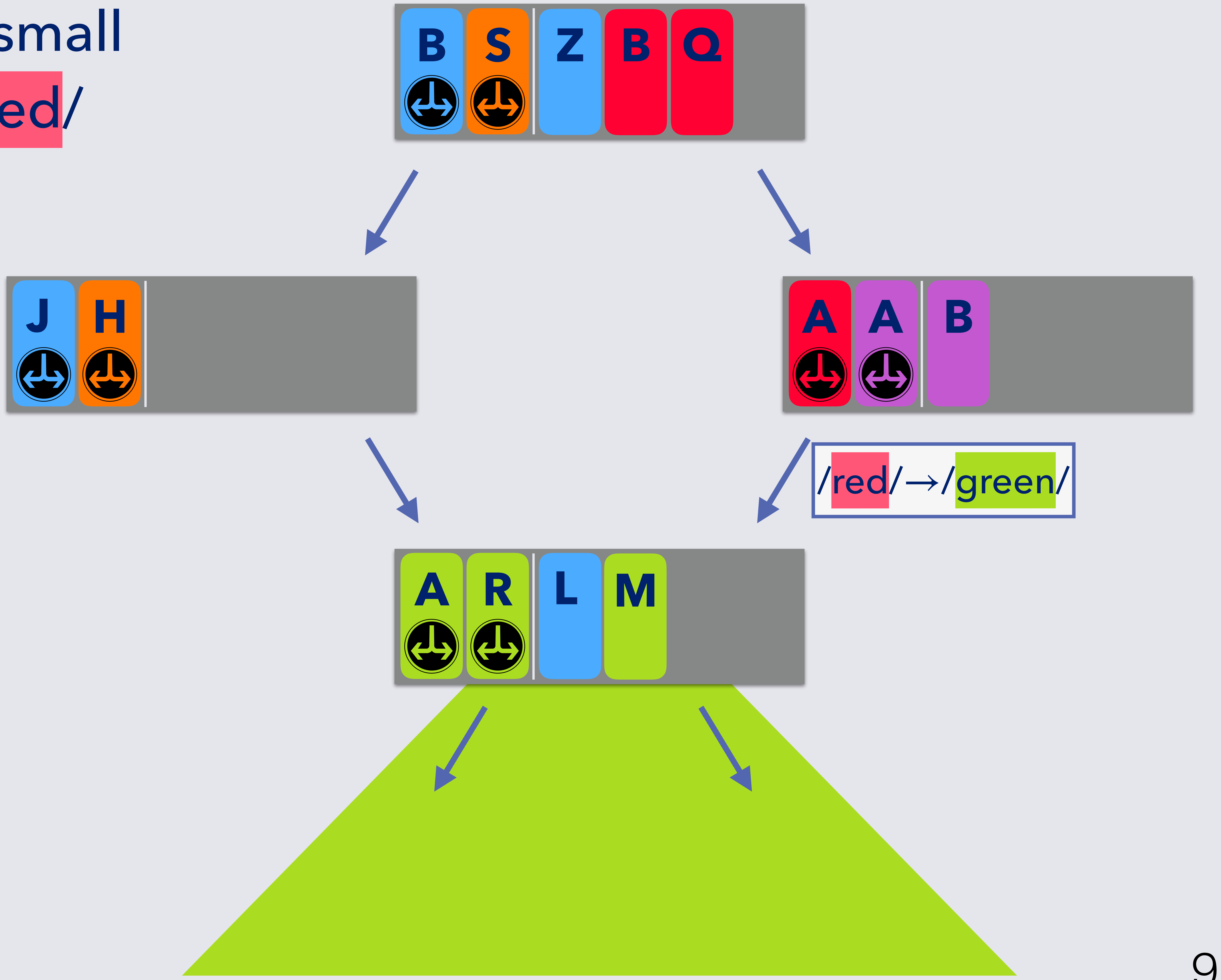
Bε-DAGs and Small Writes

Make some small writes to /red/



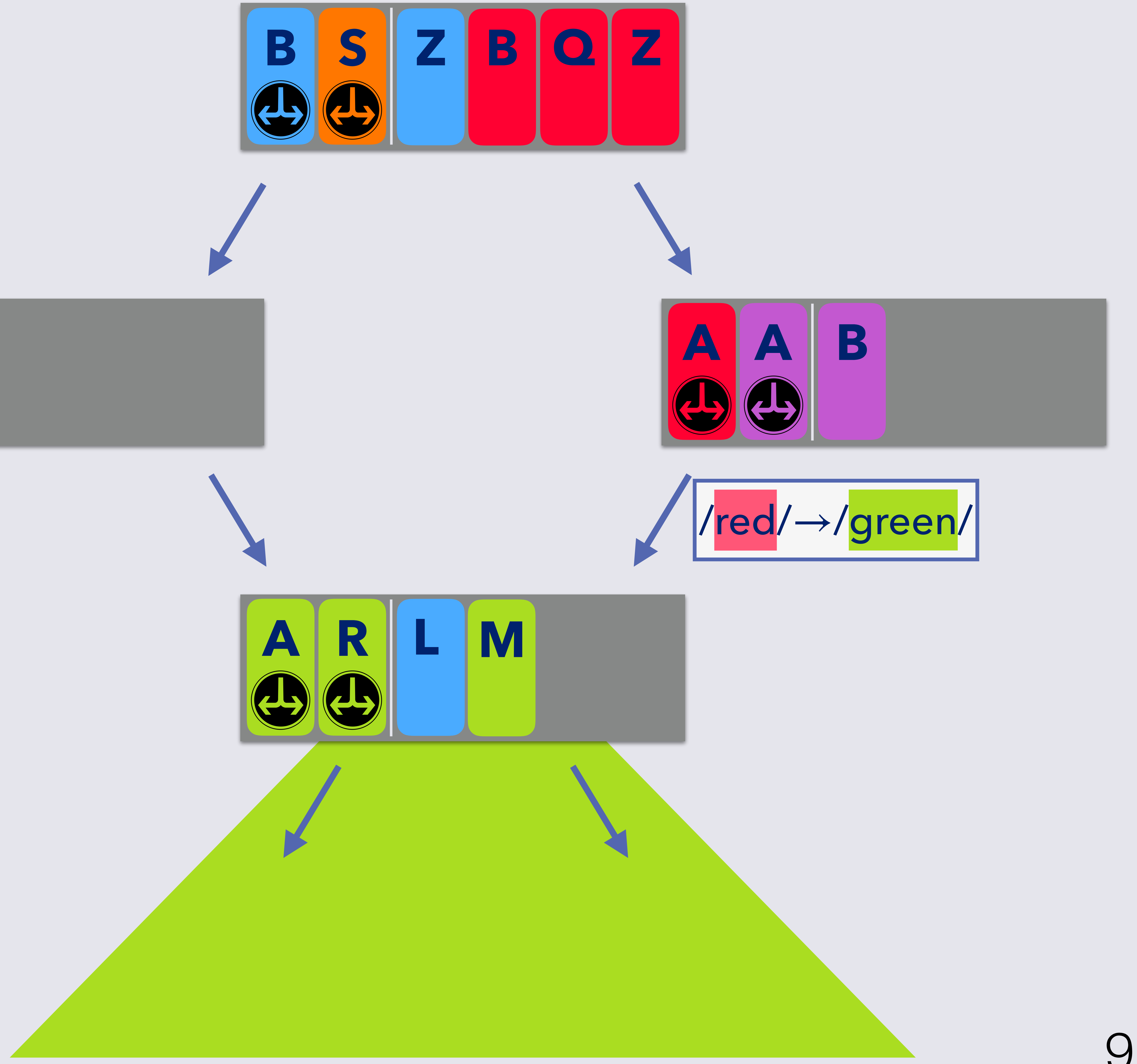
Bε-DAGs and Small Writes

Make some small writes to /red/



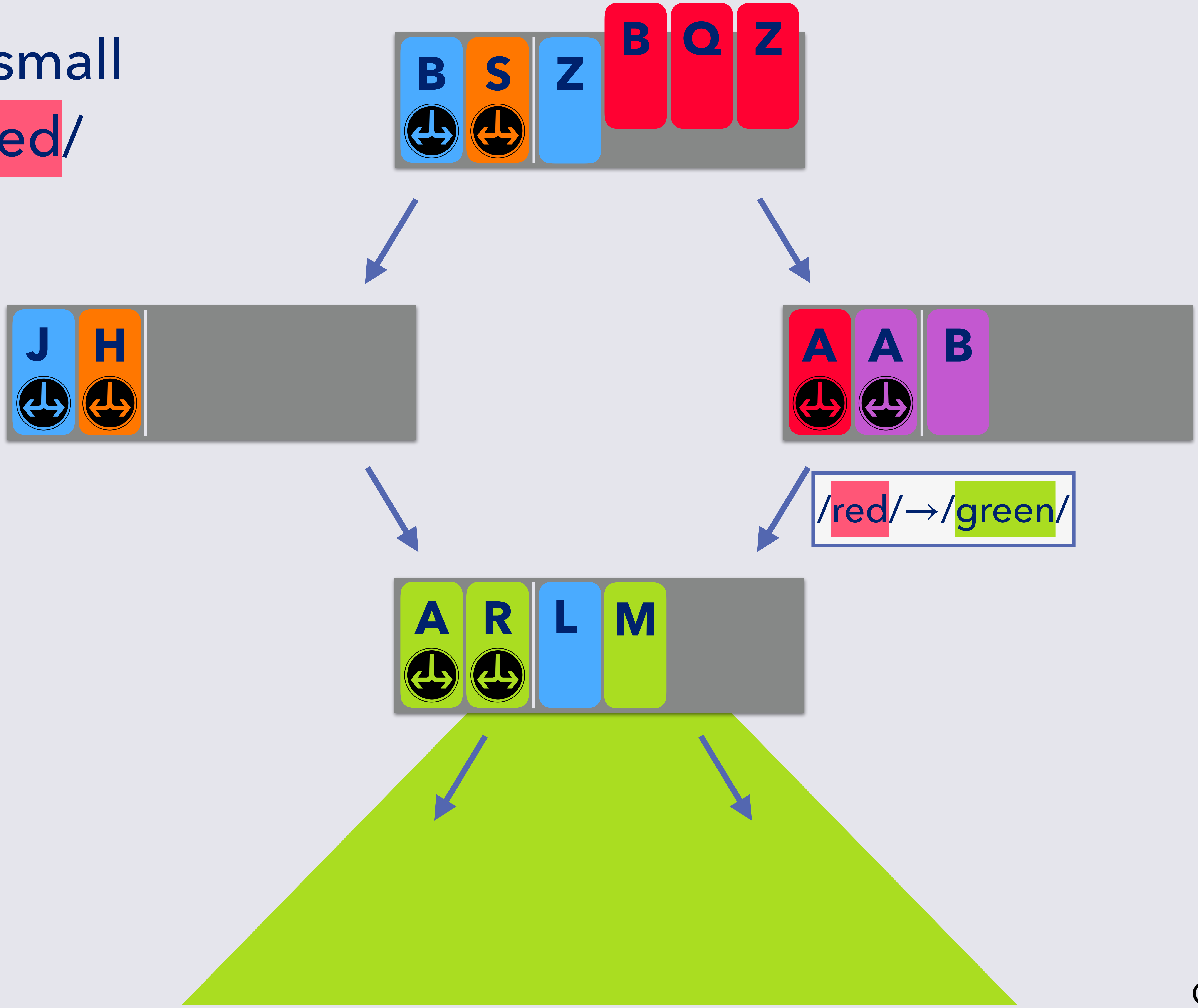
Bε-DAGs and Small Writes

Make some small writes to /red/



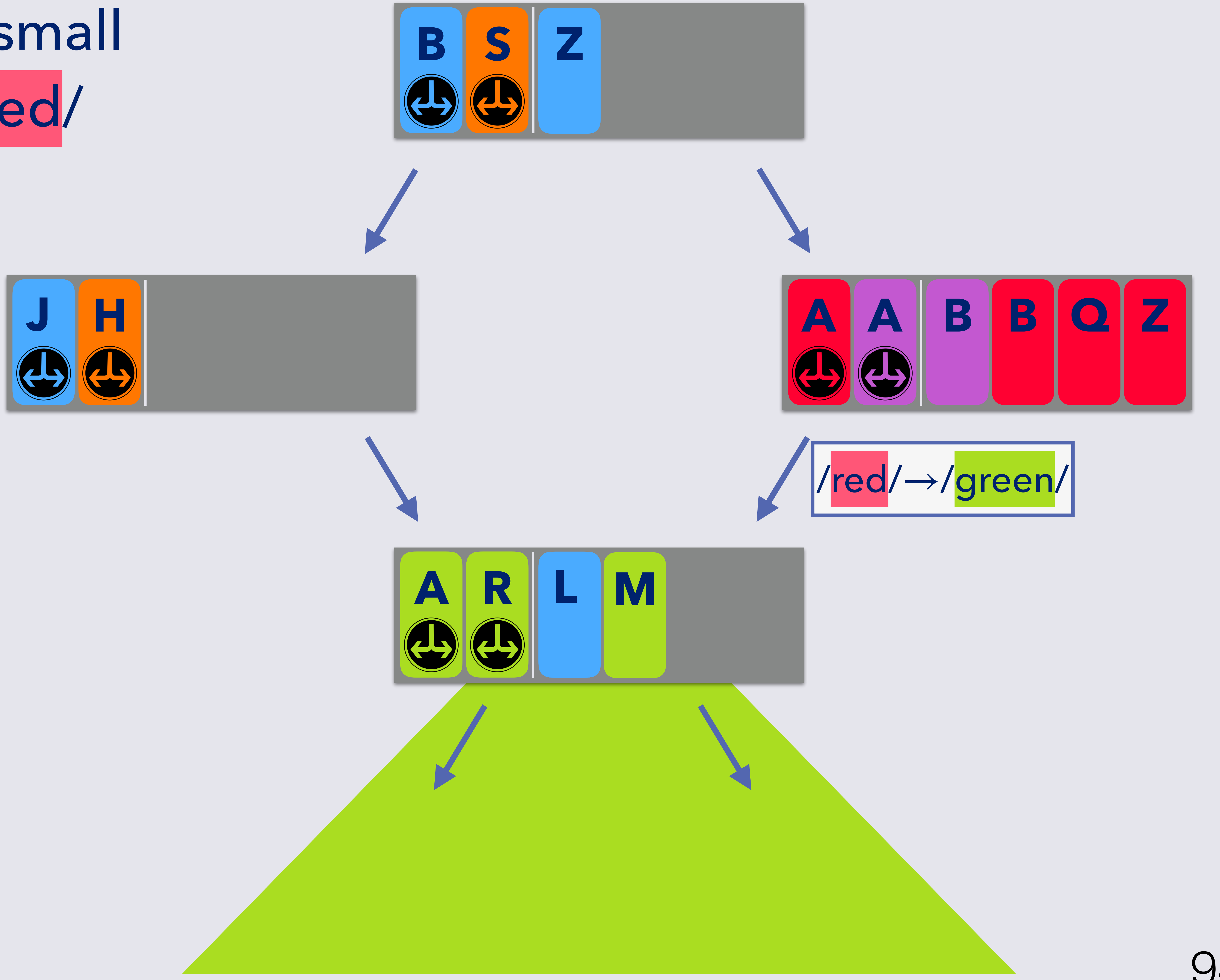
Bε-DAGs and Small Writes

Make some small writes to /red/



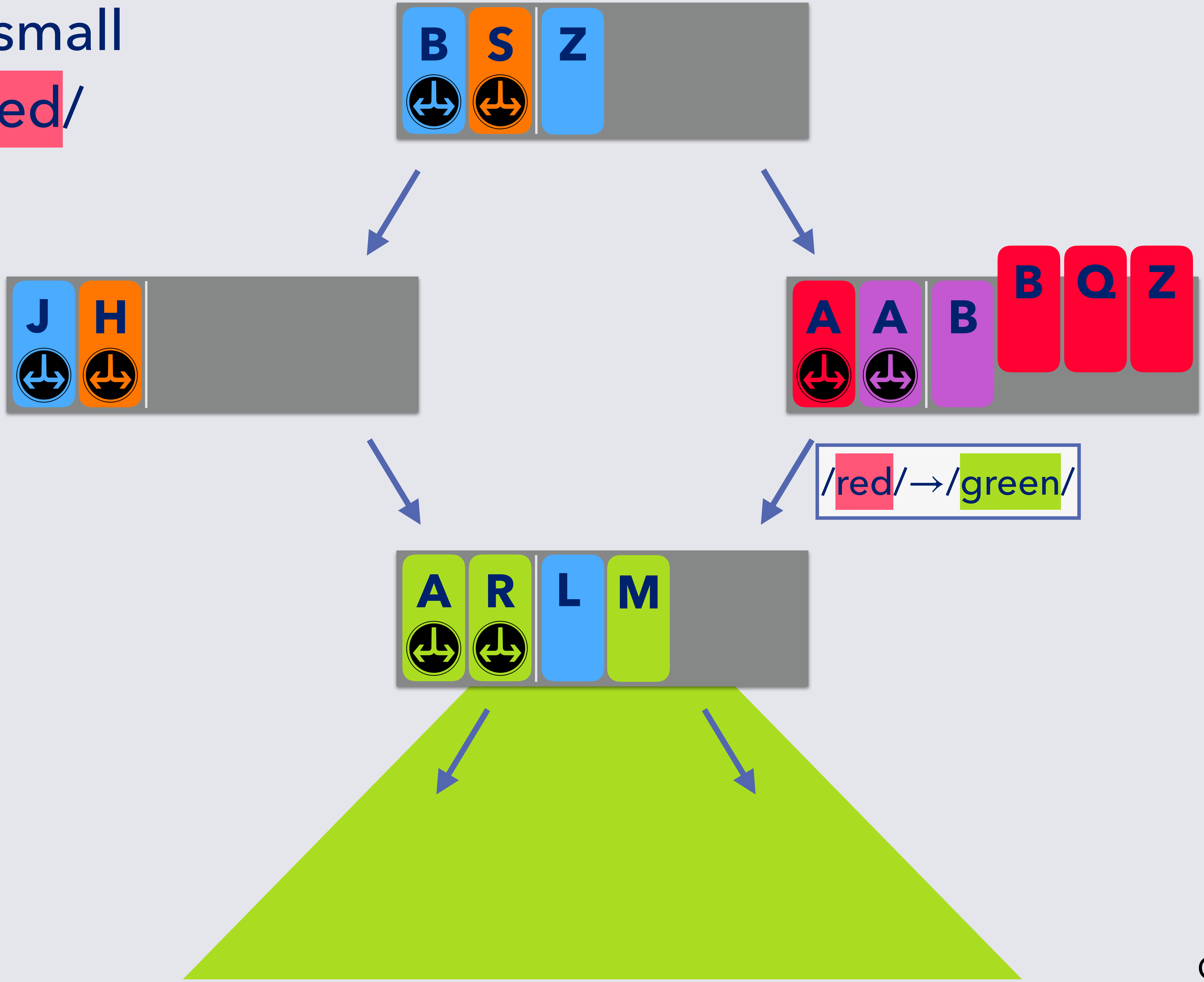
Bε-DAGs and Small Writes

Make some small writes to /red/



Bε-DAGs and Small Writes

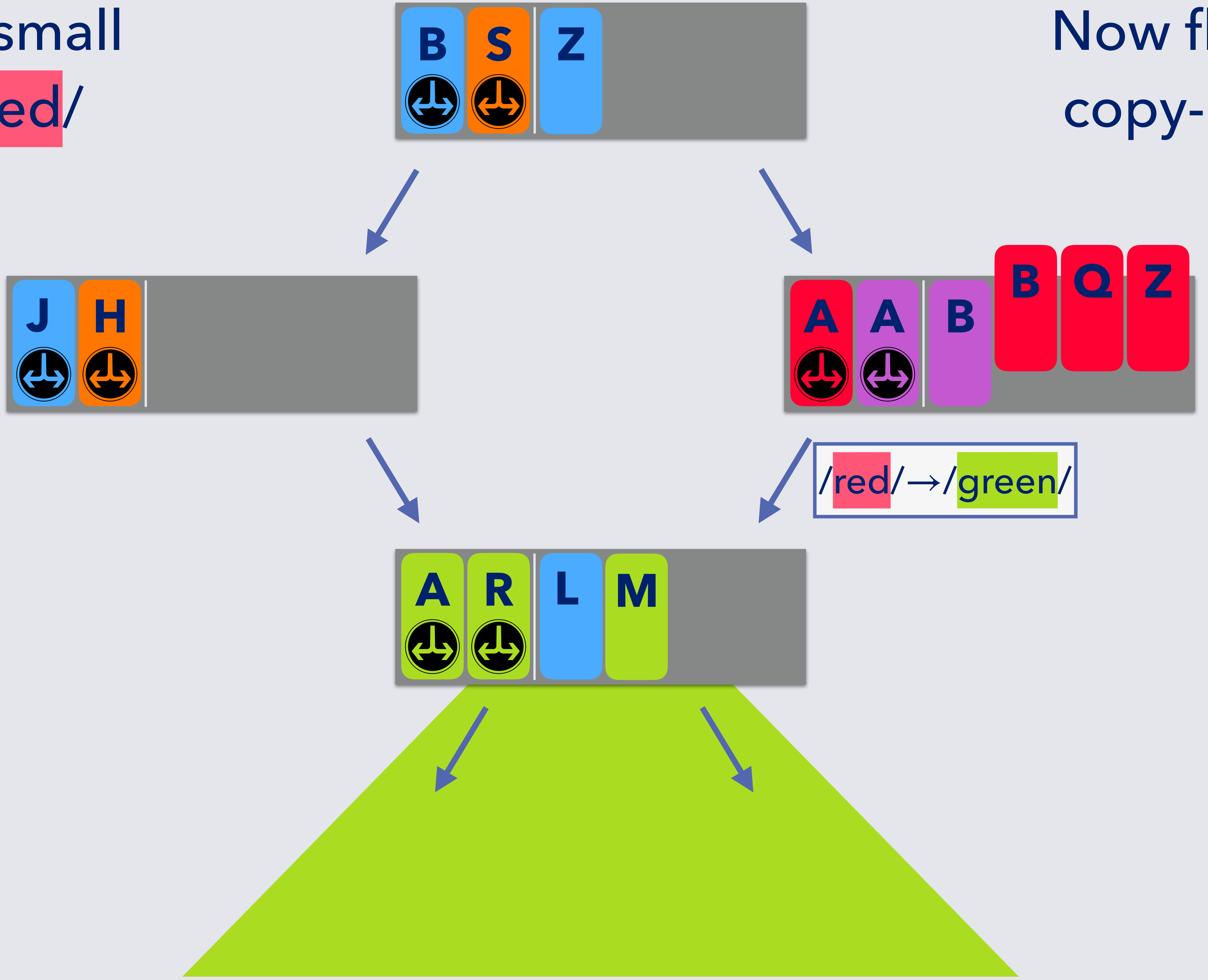
Make some small writes to /red/



Bε-DAGs and Small Writes

Make some small writes to /red/

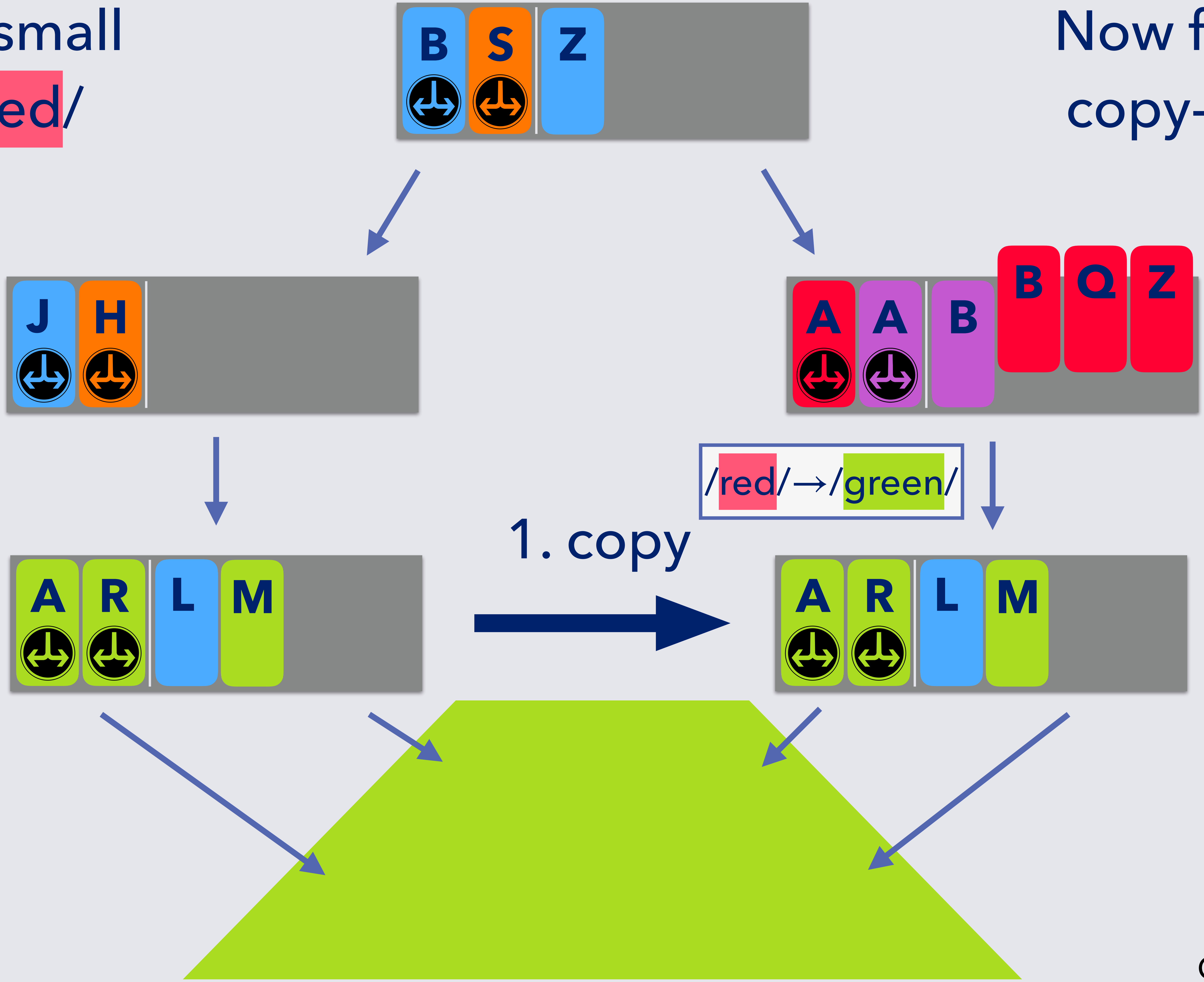
Now flush with copy-on-write



Bε-DAGs and Small Writes

Make some small writes to /red/

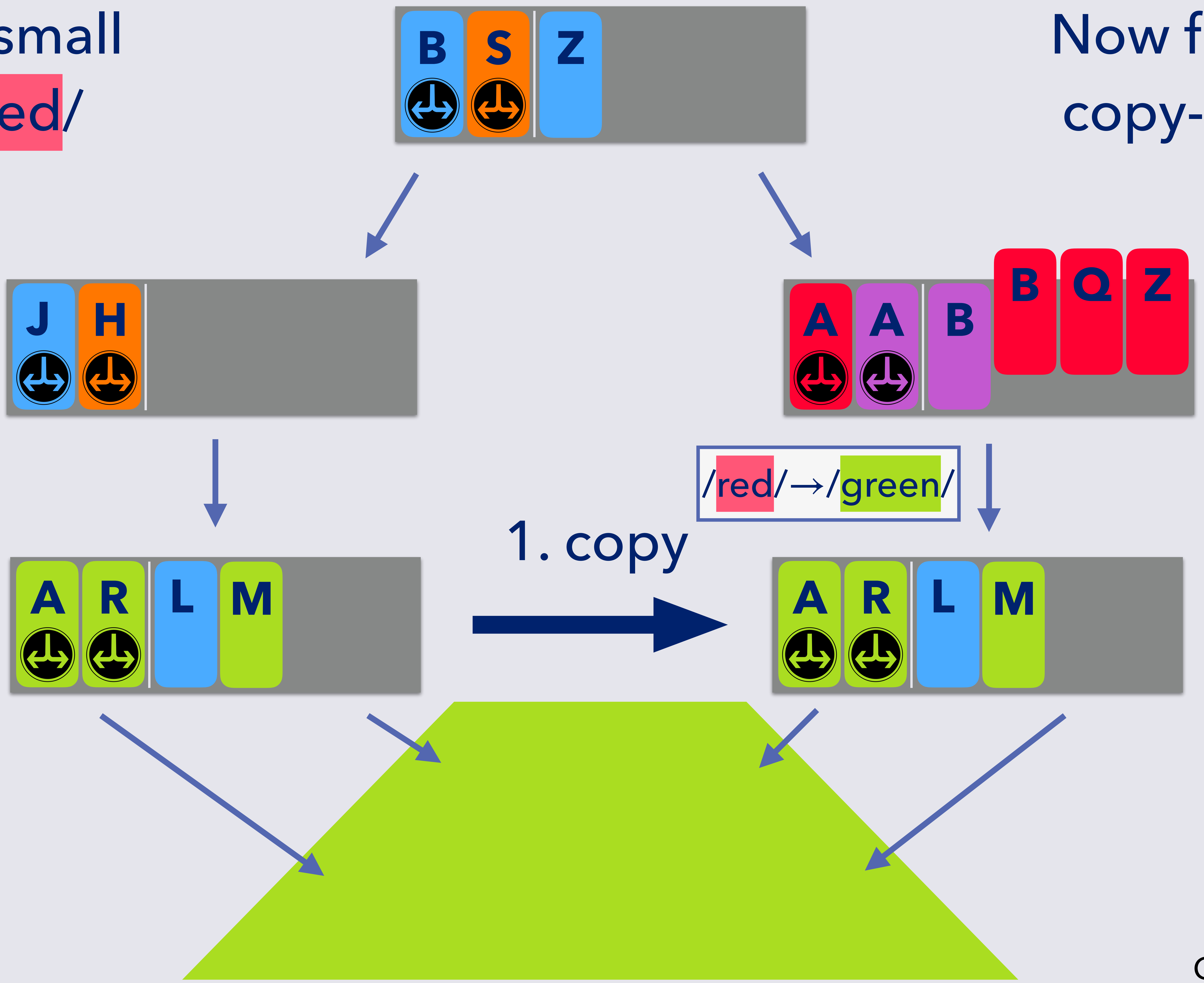
Now flush with copy-on-write



Bε-DAGs and Small Writes

Make some small writes to /red/

Now flush with copy-on-write

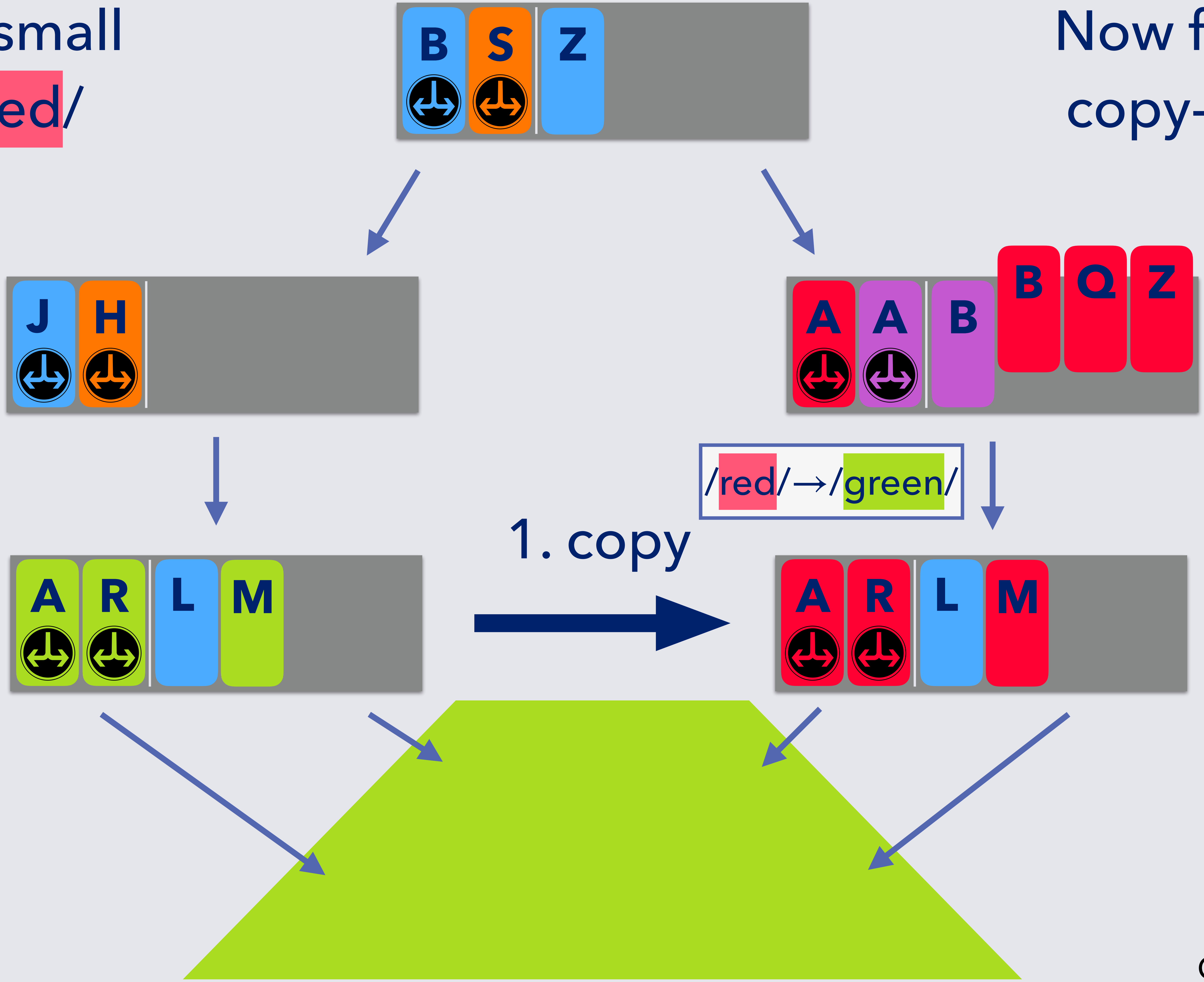


2. Translate /green/→ /red/

Bε-DAGs and Small Writes

Make some small writes to /red/

Now flush with copy-on-write

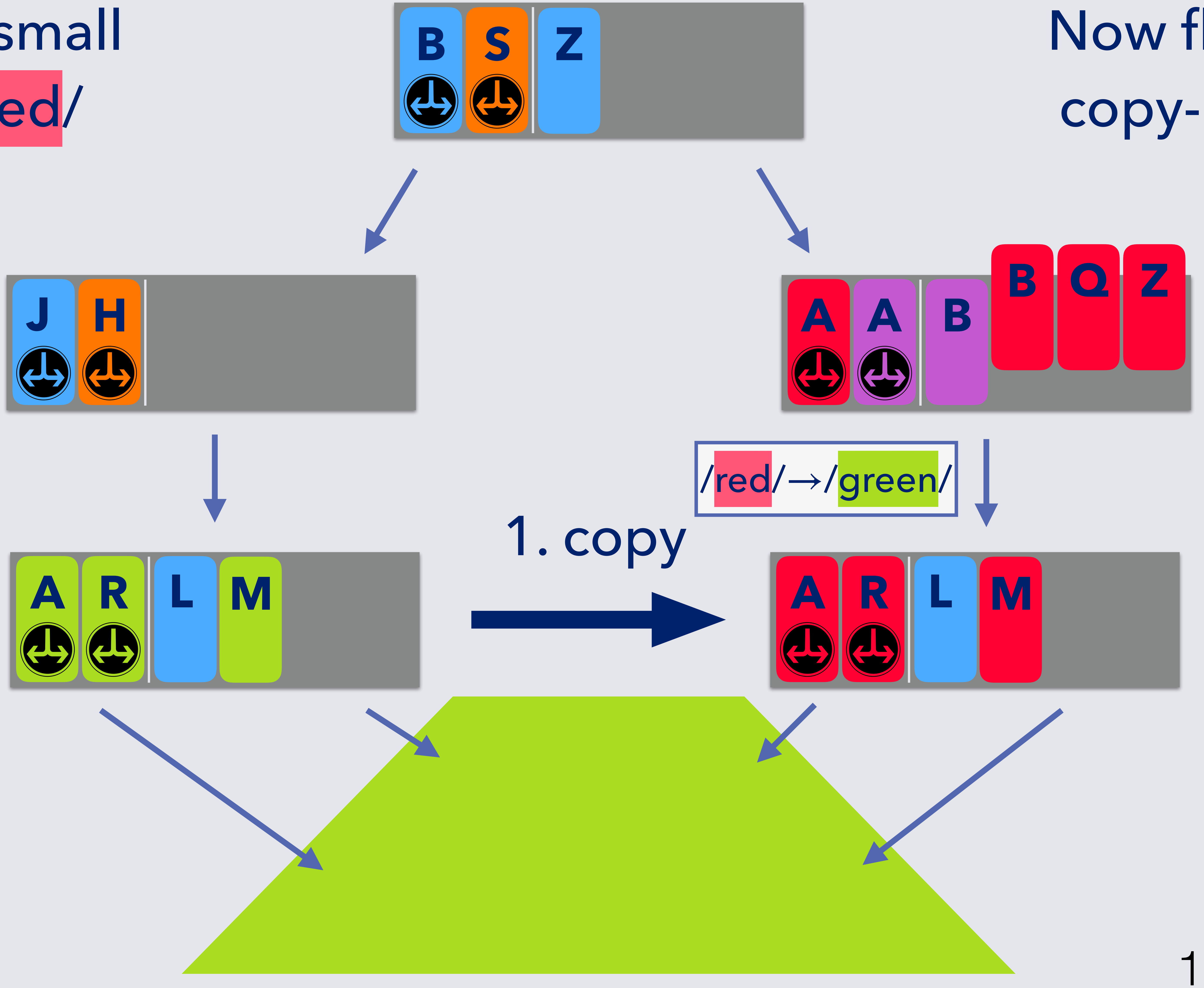


2. Translate /green/→ /red/

Bε-DAGs and Small Writes

Make some small writes to /red/

Now flush with copy-on-write



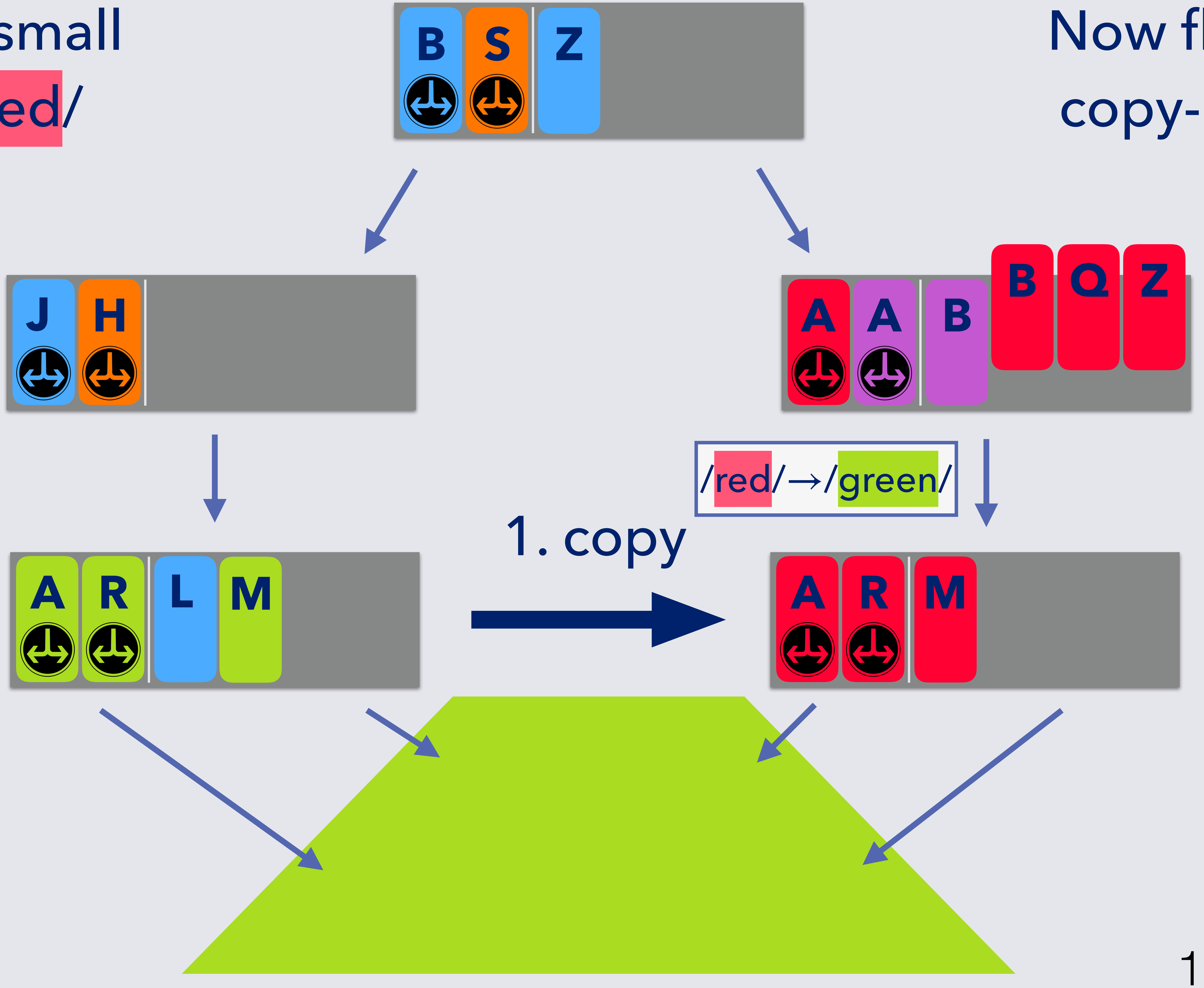
2. Translate /green/→ /red/

3. Delete unreachable data

Bε-DAGs and Small Writes

Make some small writes to /red/

Now flush with copy-on-write



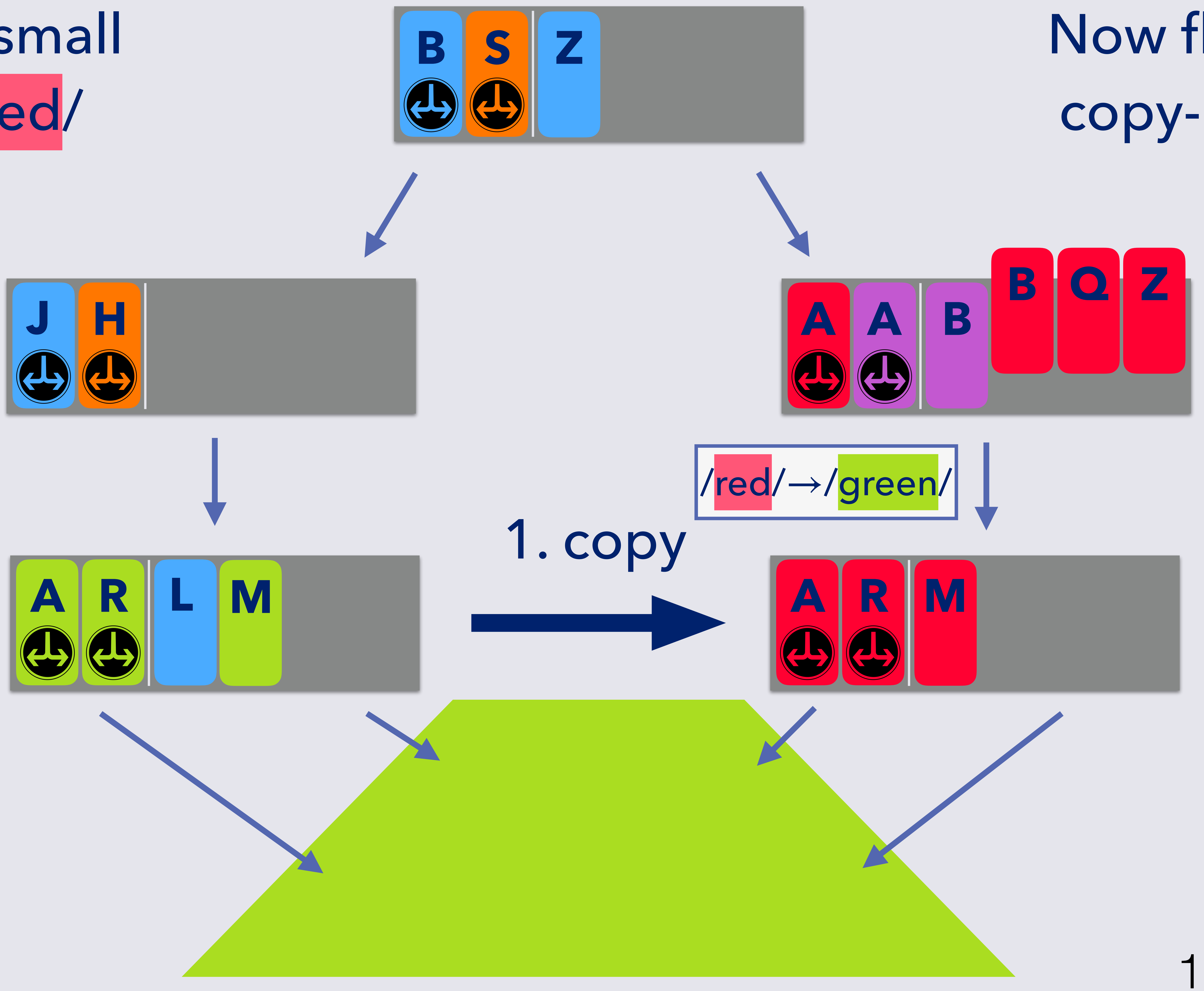
2. Translate /green/→ /red/

3. Delete unreachable data

Bε-DAGs and Small Writes

Make some small writes to /red/

Now flush with copy-on-write



2. Translate /green/→ /red/

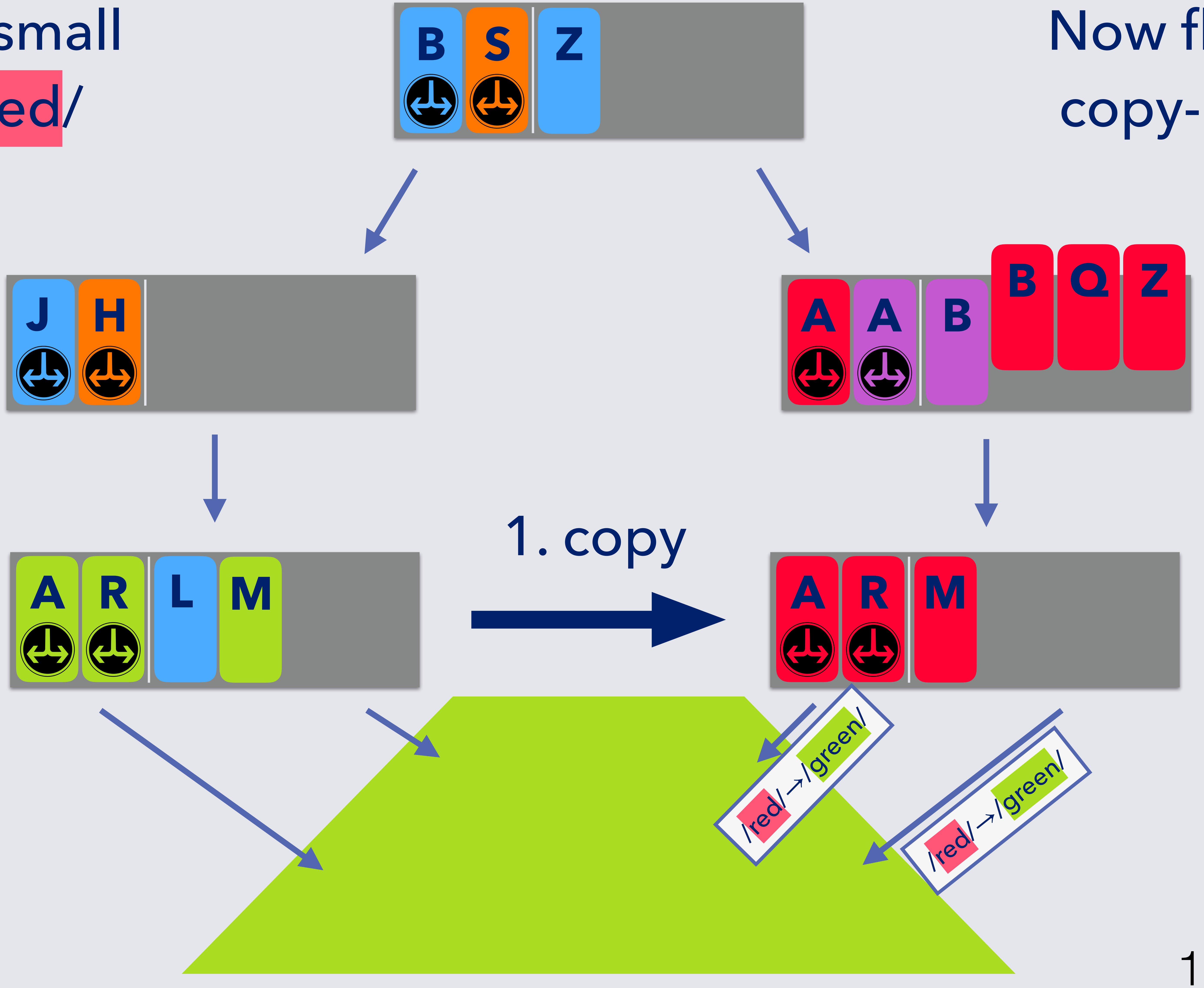
3. Delete unreachable data

4. Move pivot translation

Bε-DAGs and Small Writes

Make some small writes to /red/

Now flush with copy-on-write



2. Translate /green/→ /red/

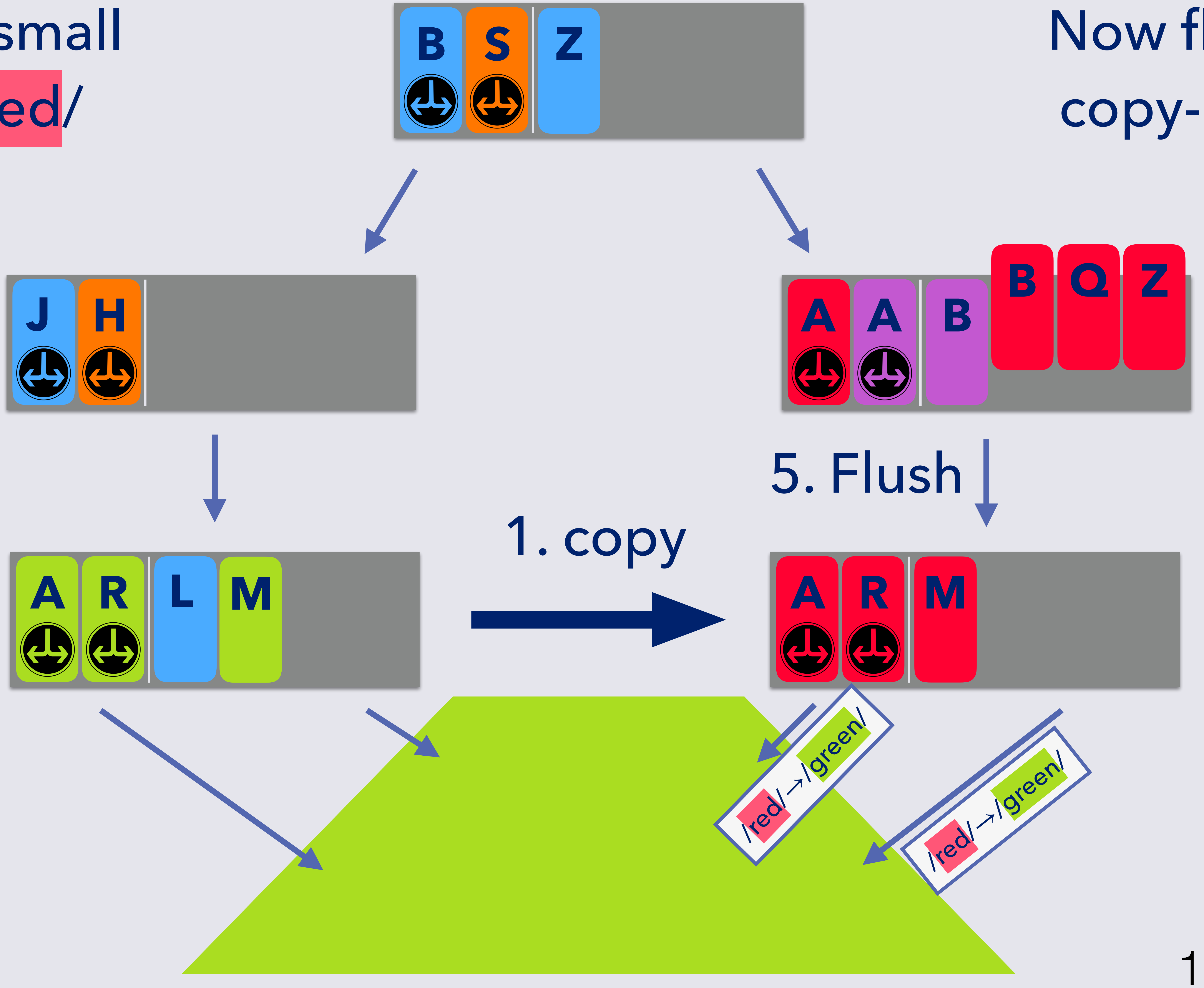
3. Delete unreachable data

4. Move pivot translation

Bε-DAGs and Small Writes

Make some small writes to /red/

Now flush with copy-on-write



2. Translate /green/→ /red/

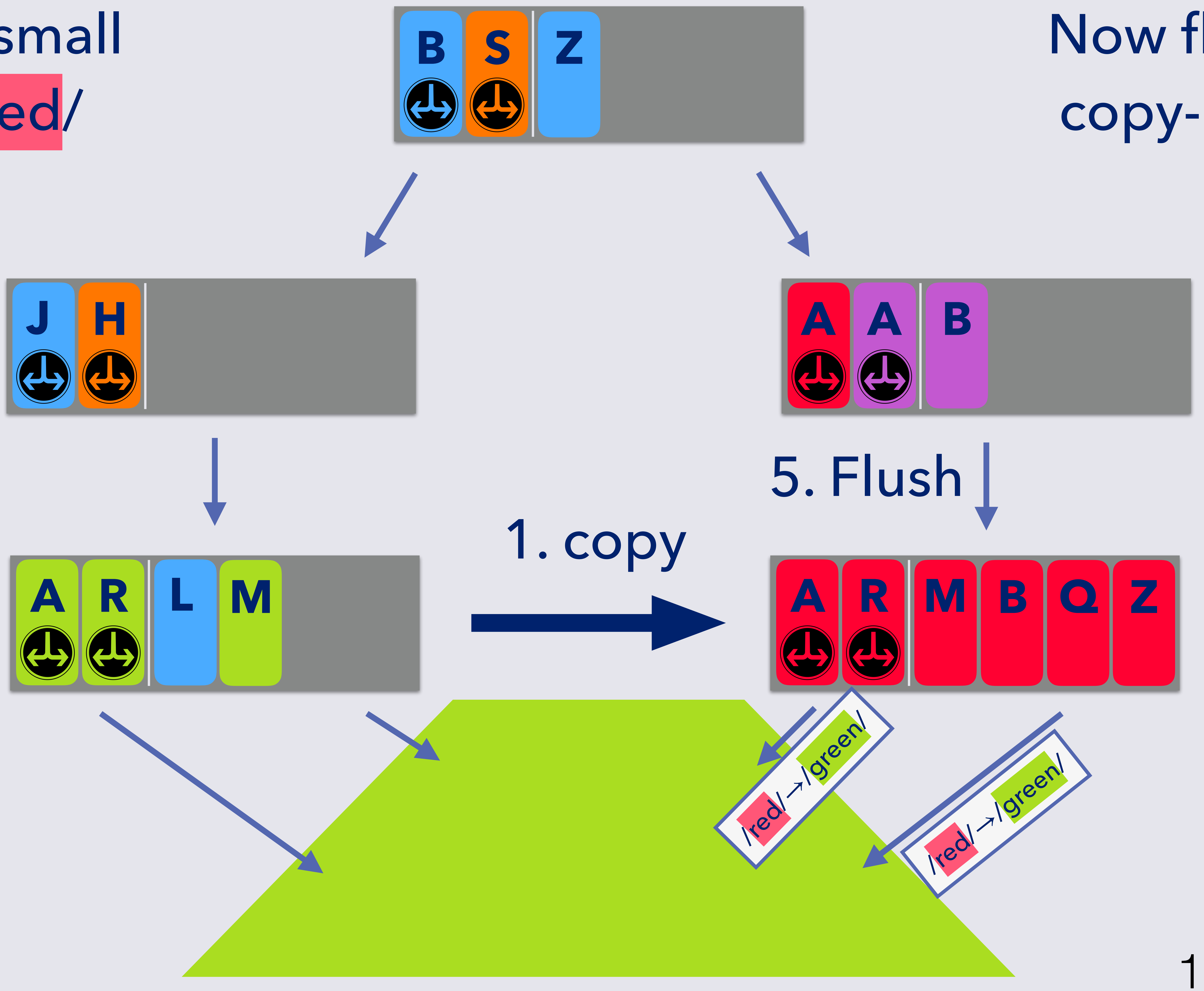
3. Delete unreachable data

4. Move pivot translation

Bε-DAGs and Small Writes

Make some small writes to /red/

Now flush with copy-on-write



2. Translate /green/→ /red/

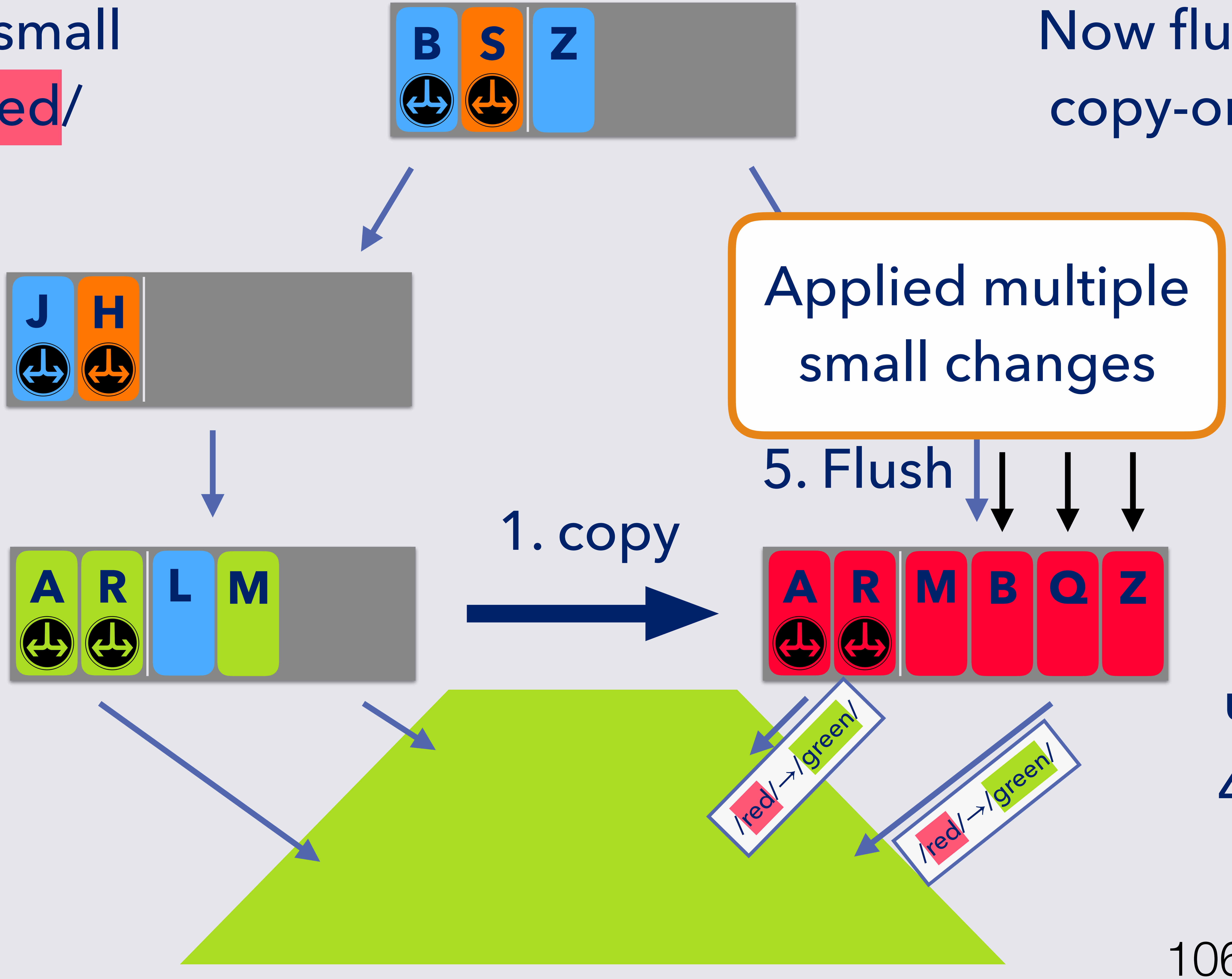
3. Delete unreachable data

4. Move pivot translation

Bε-DAGs and Small Writes

Make some small writes to /red/

Now flush with copy-on-write



2. Translate /green/→ /red/

3. Delete unreachable data

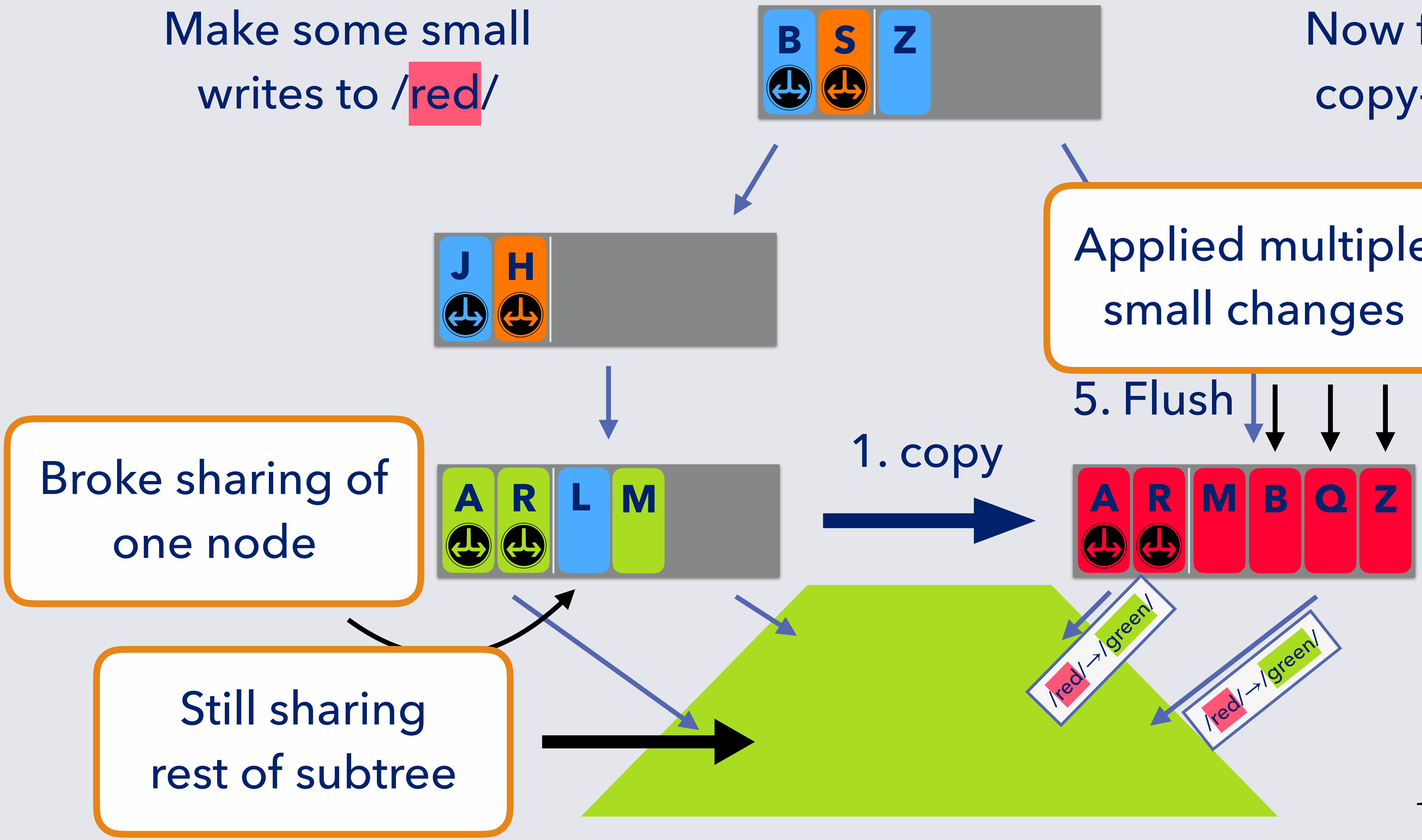
4. Move pivot translation

Bε-DAGs and Small Writes

Bε-DAGs and Small Writes

Make some small writes to /red/

Now flush with copy-on-write



2. Translate /green/→ /red/

3. Delete unreachable data

4. Move pivot translation

Bε-DAGs and Small Writes

Logical Copy with B^{ε}-DAGs

Performance Goals

Space efficient ?

Low latency ?

Copy-specific

Fast writes ?

Fast reads ?

General file system

Logical Copy with Bε-DAGs

Performance Goals

Space efficient ?

Low latency ?

Copy-specific

Bε-trees have large nodes

Fast writes ✓

⇒ Locality*

Fast reads ✓

General file system

**File Systems Fated for Senescence? Nonsense, Says Science!*, Conway et al, FAST 2017

Logical Copy with Bε-DAGs

Performance Goals

Copy-on-Abundant-Write

Space efficient ✓

Low latency ?

Copy-specific

Bε-trees have large nodes

Fast writes ✓

⇒ Locality*

Fast reads ✓

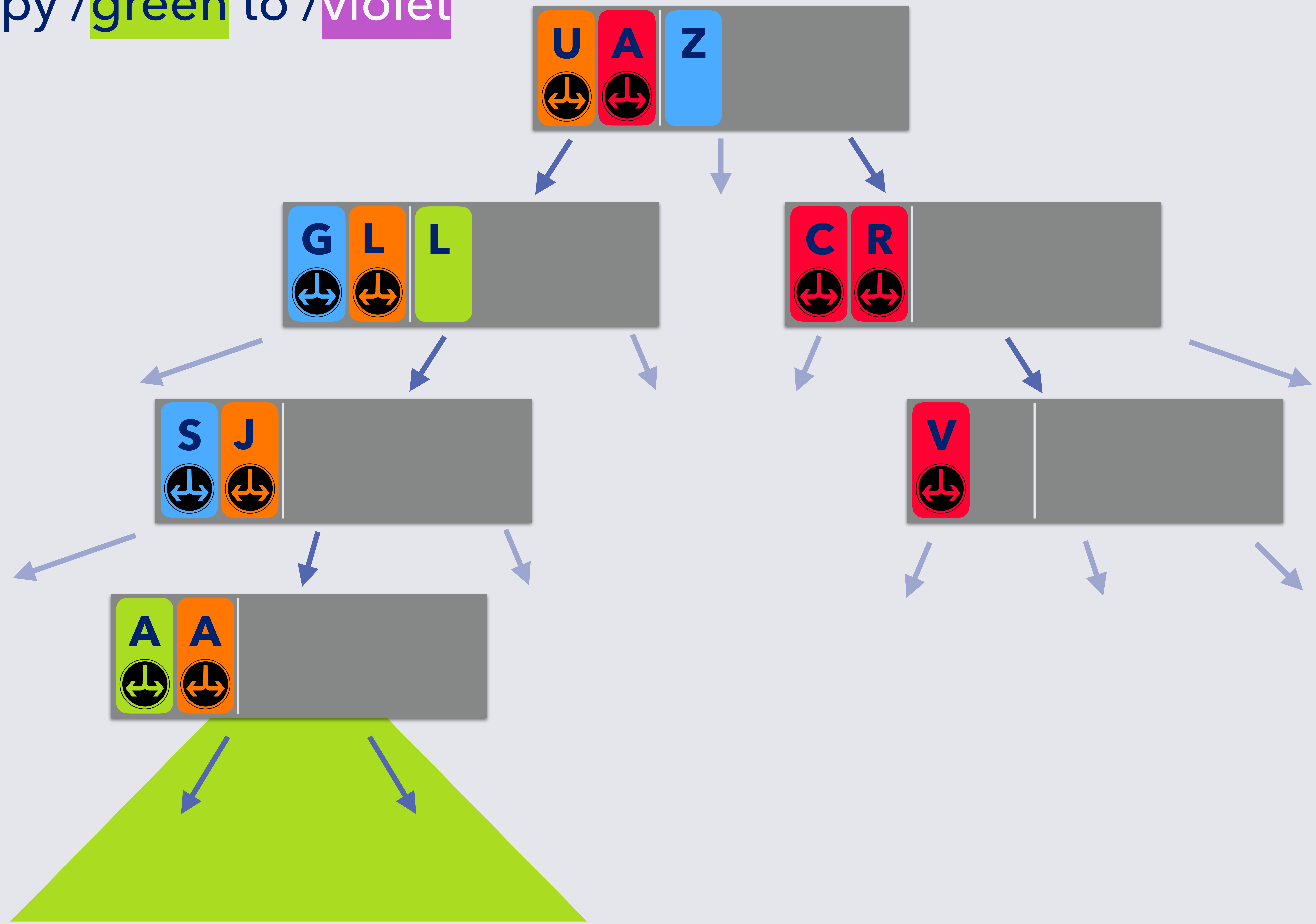
General file system

**File Systems Fated for Senescence? Nonsense, Says Science!*, Conway et al, FAST 2017

Reducing Copy Latency in B^{ε}-DAGs

Reducing Copy Latency in Bε-DAGs

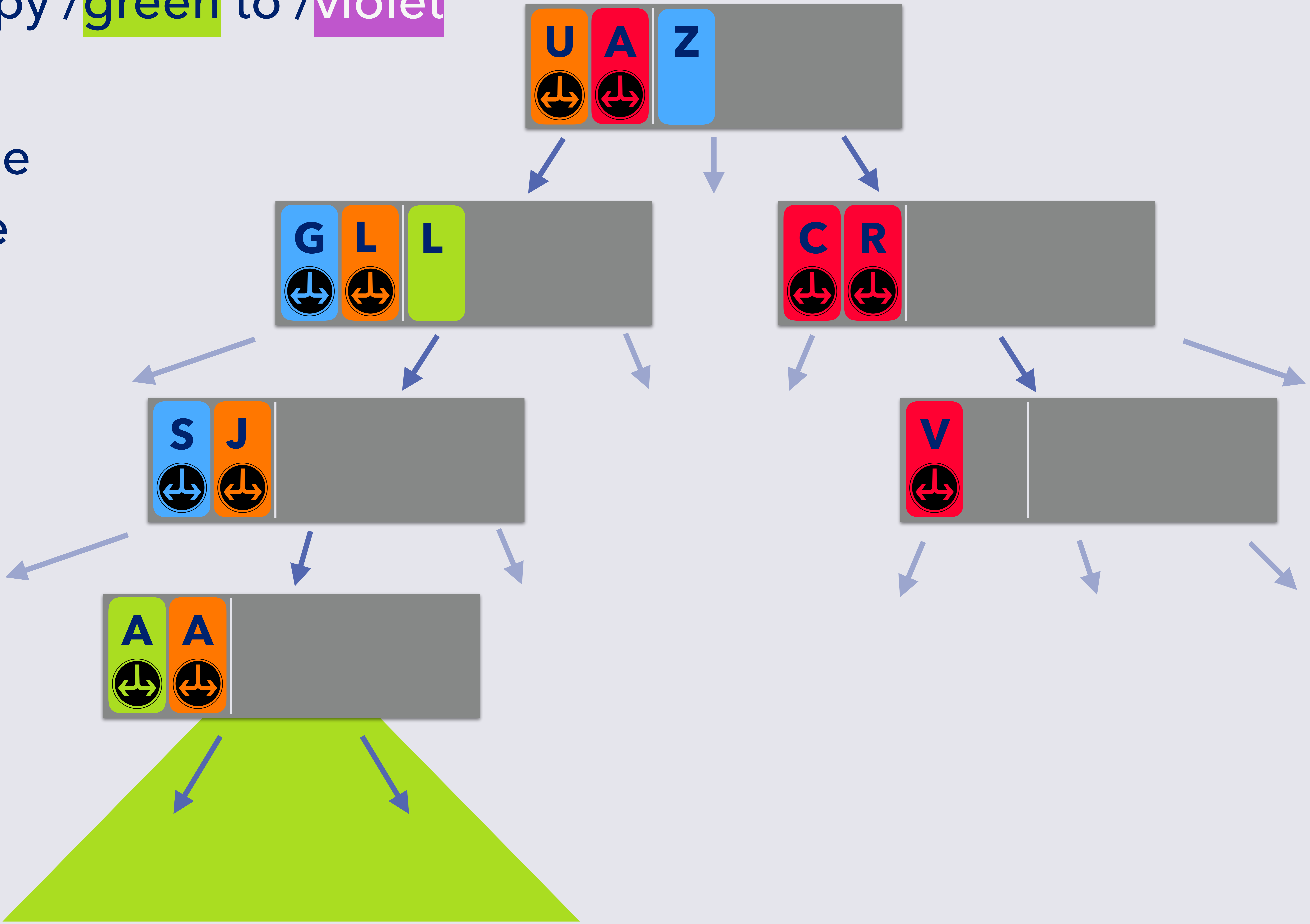
Copy /green to /violet



Reducing Copy Latency in B$^{\varepsilon}$-DAGs

Copy /green to /violet

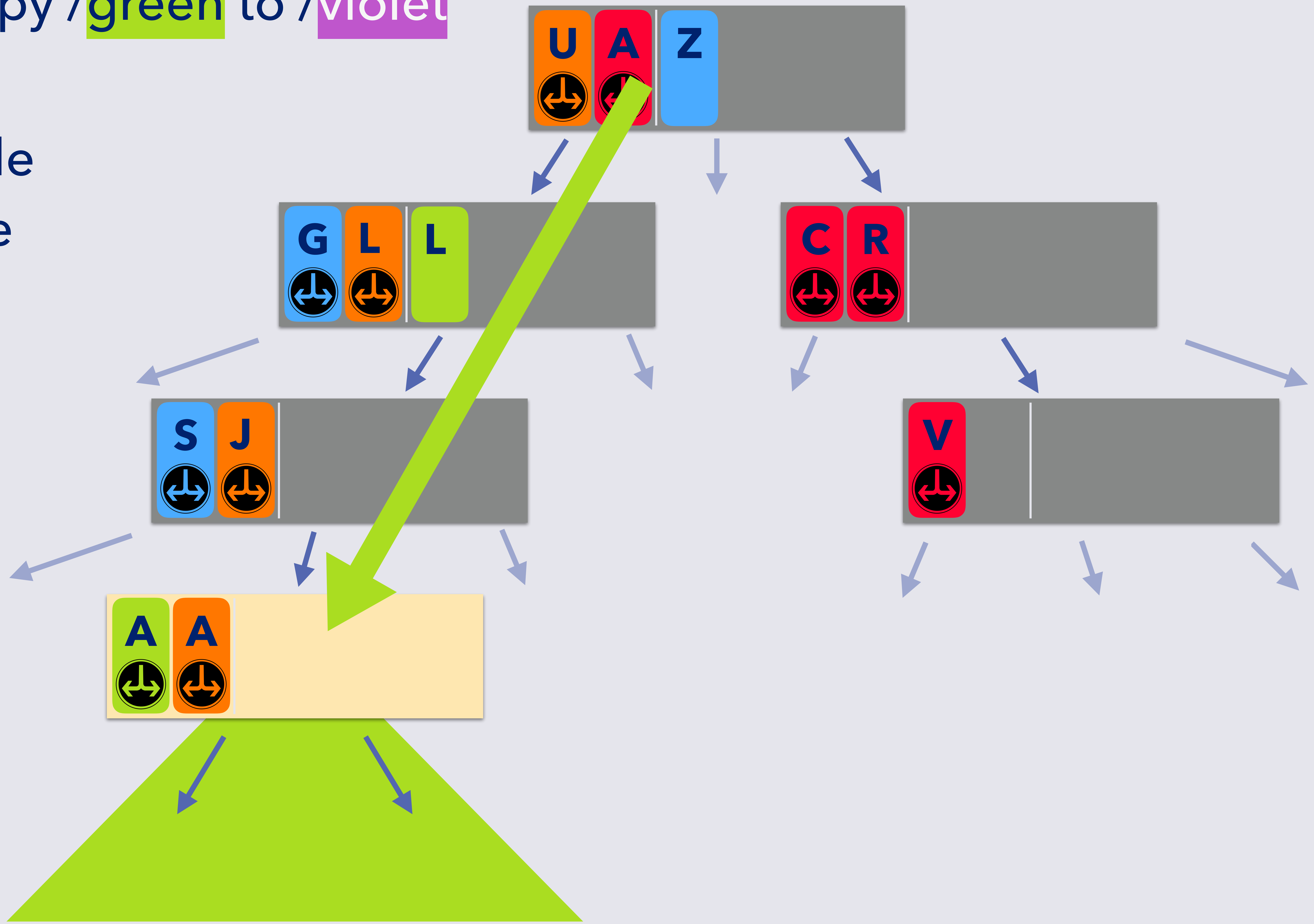
1. Flush messages to node covering /green subtree



Reducing Copy Latency in Bε-DAGs

Copy /green to /violet

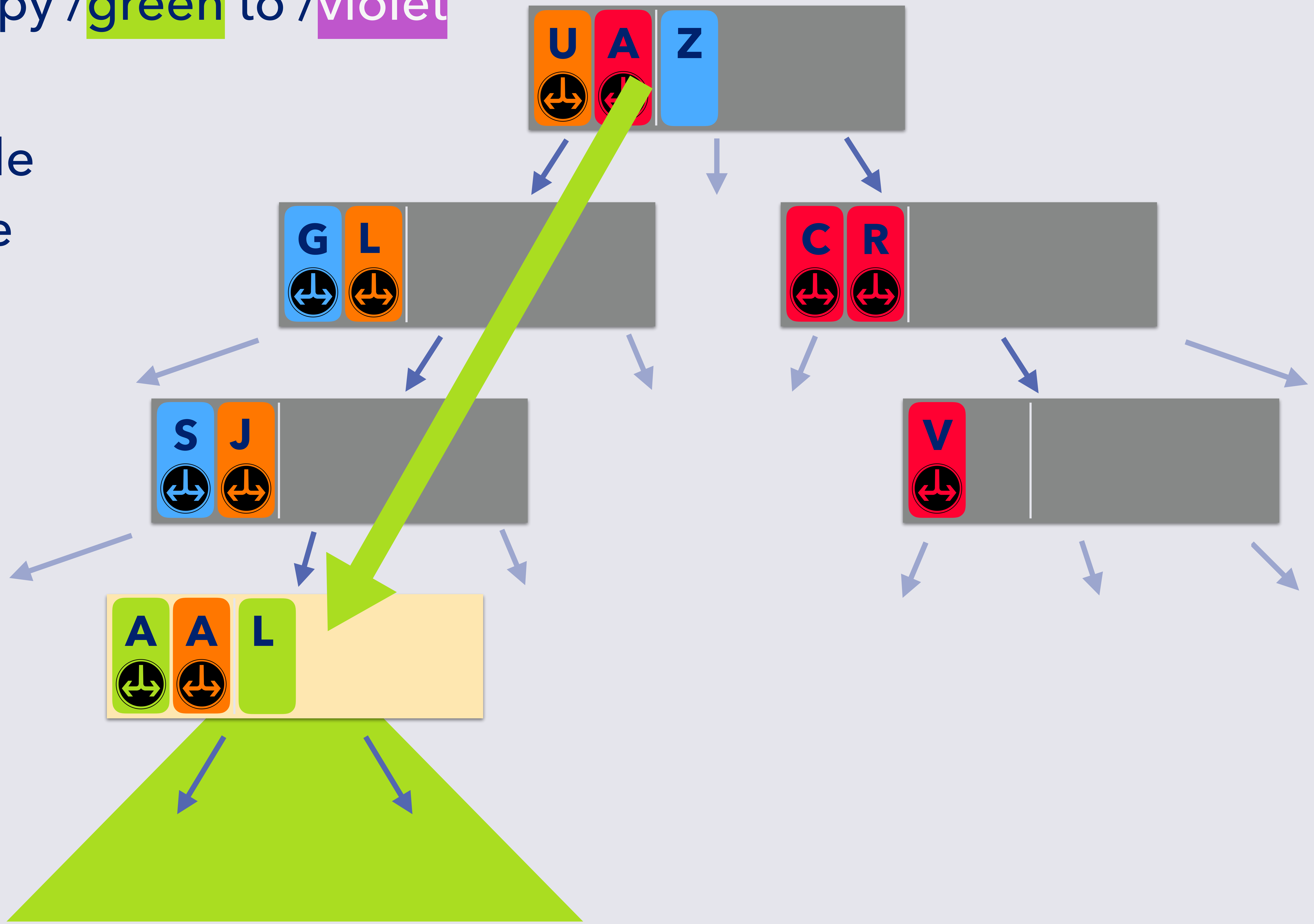
1. Flush messages to node covering /green subtree



Reducing Copy Latency in Bε-DAGs

Copy /green to /violet

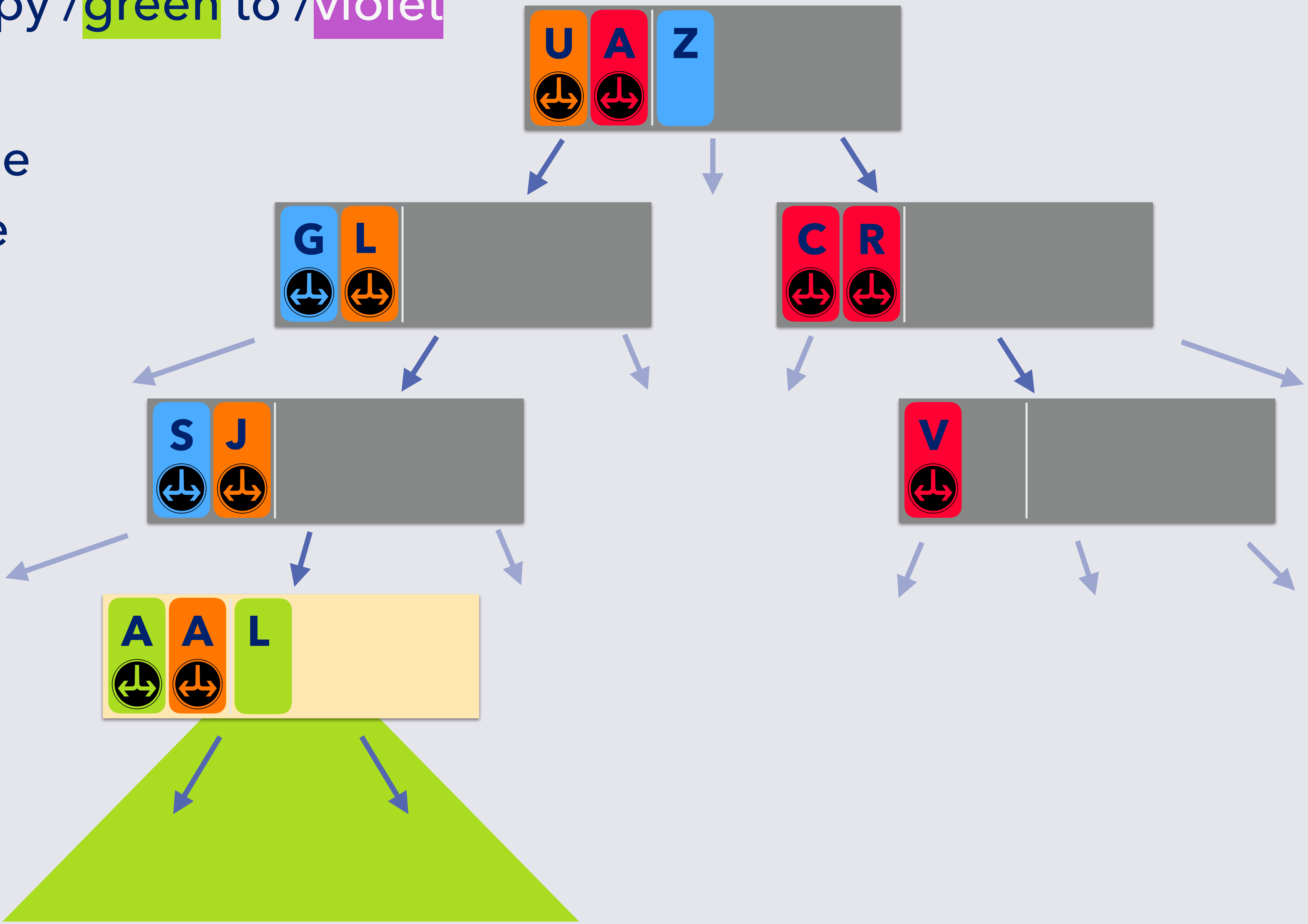
1. Flush messages to node covering /green subtree



Reducing Copy Latency in Bε-DAGs

Copy /green to /violet

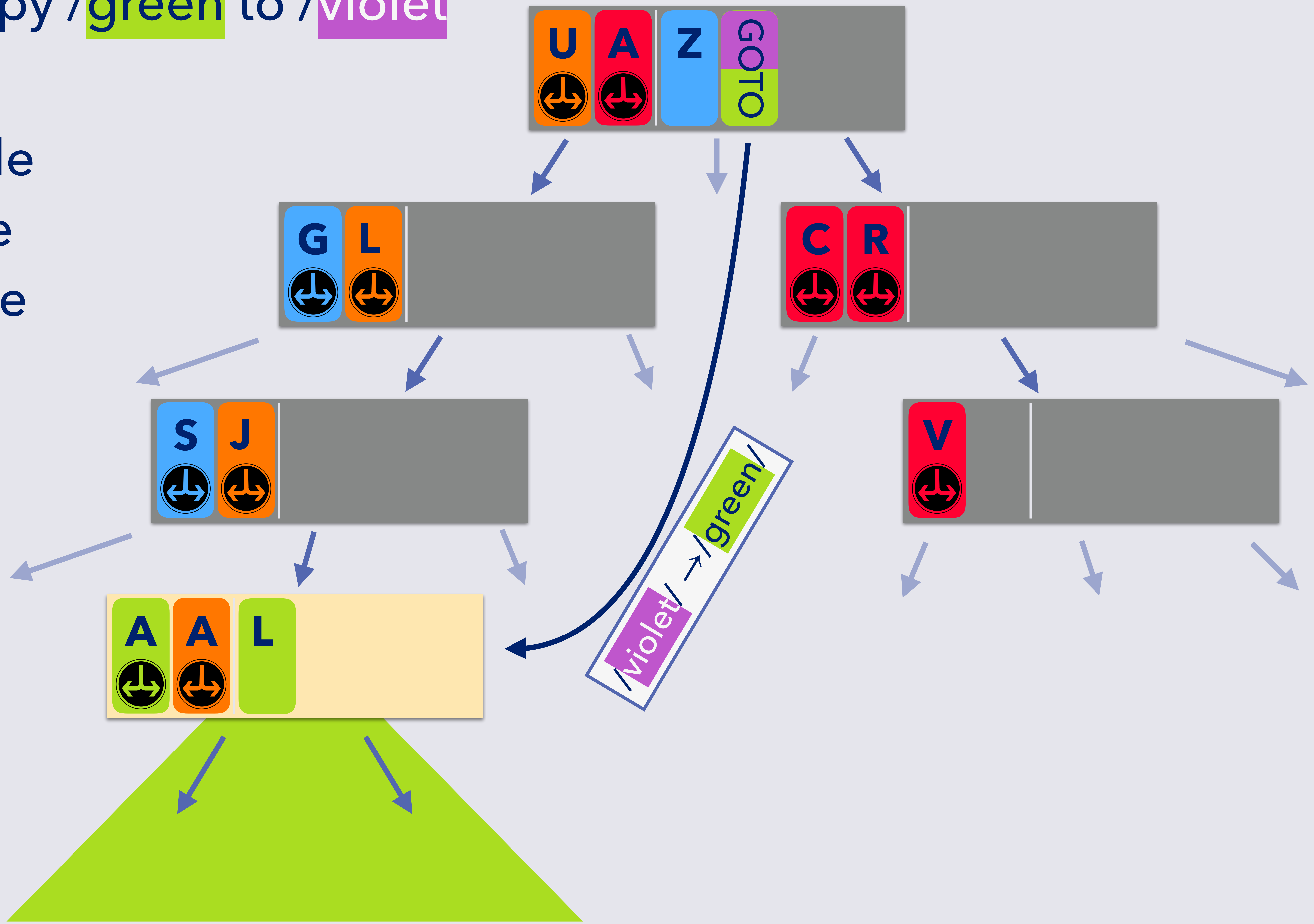
1. Flush messages to node covering /green subtree



Reducing Copy Latency in Bε-DAGs

Copy /green to /violet

1. Flush messages to node covering /green subtree
2. Insert a GOTO message



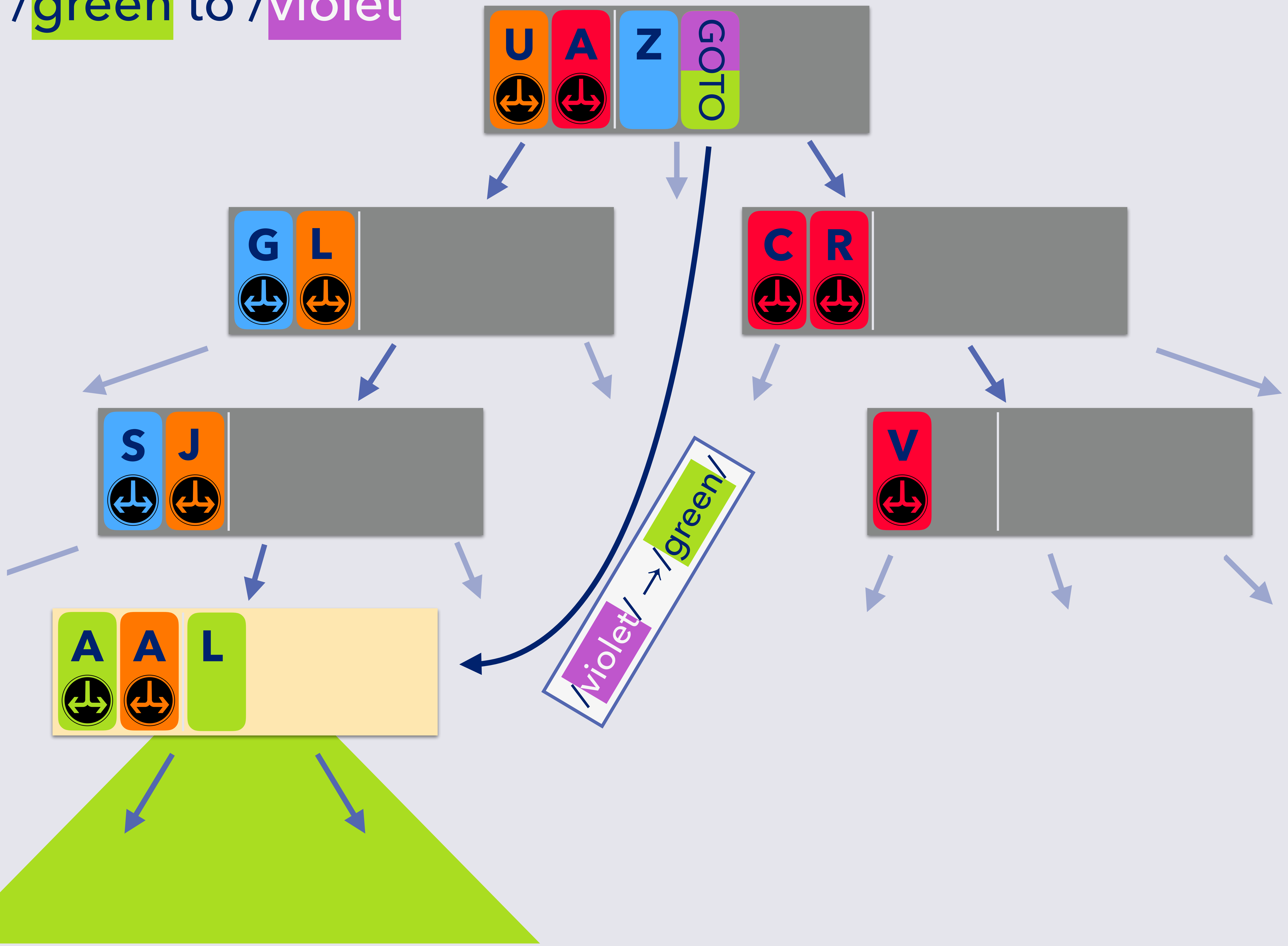
Reducing Copy Latency in Bε-DAGs

Copy /green to /violet

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

GOTO

A GOTO message changes the structure of the tree



Reducing Copy Latency in Bε-DAGs

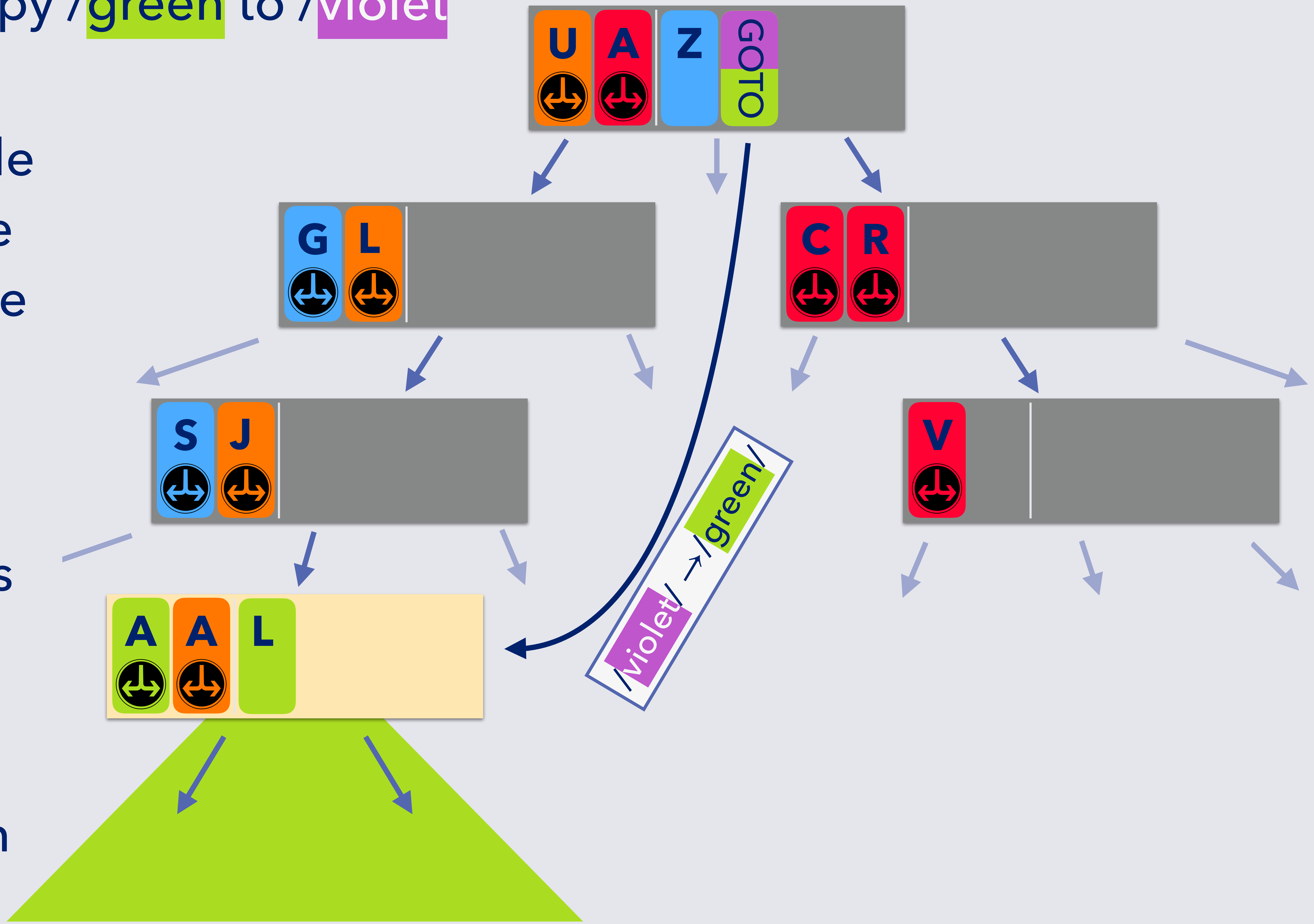
Copy /green to /violet

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

GOTO

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation



Reducing Copy Latency in Bε-DAGs

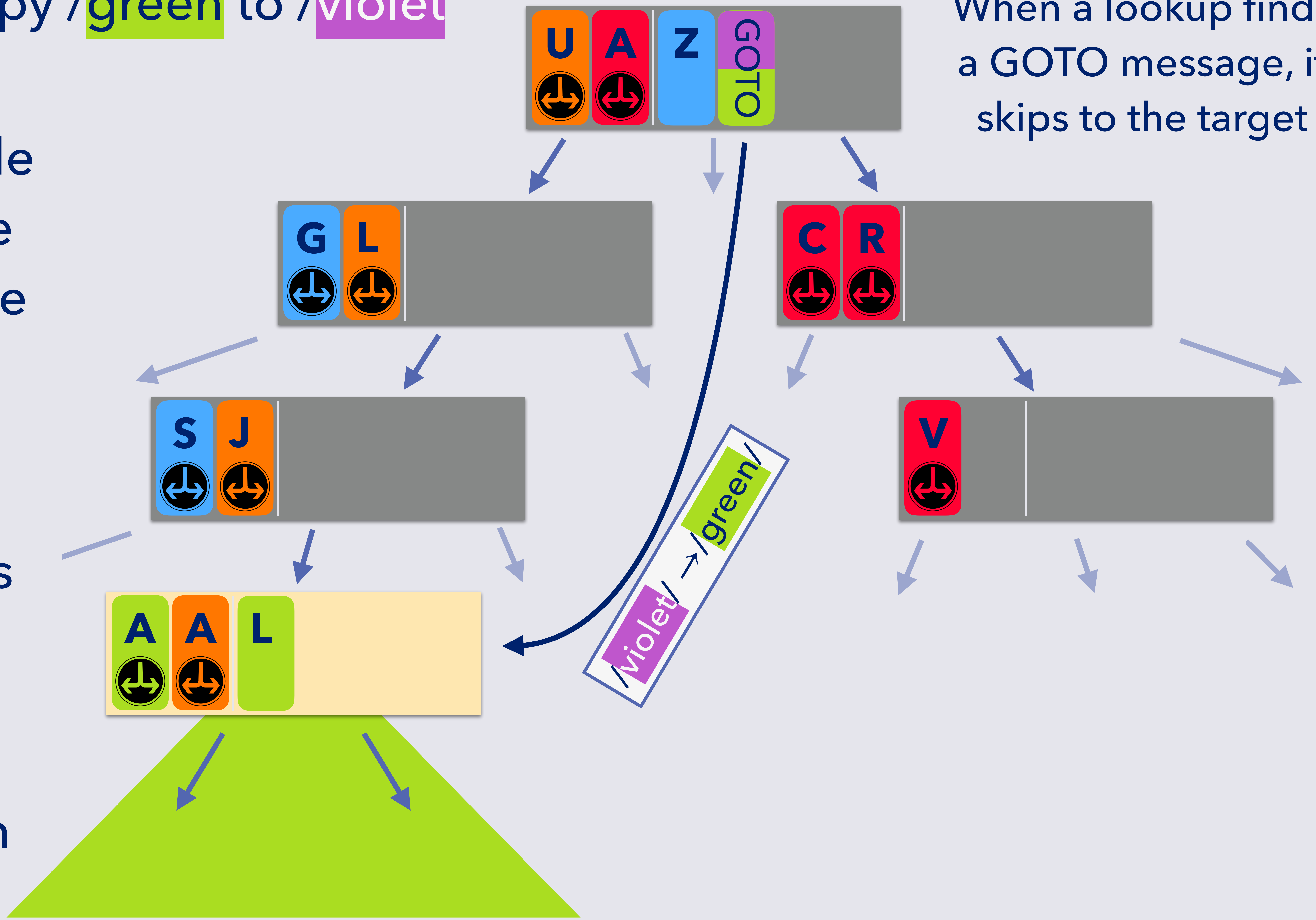
Copy /green to /violet

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

GOTO

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation



When a lookup finds a GOTO message, it skips to the target

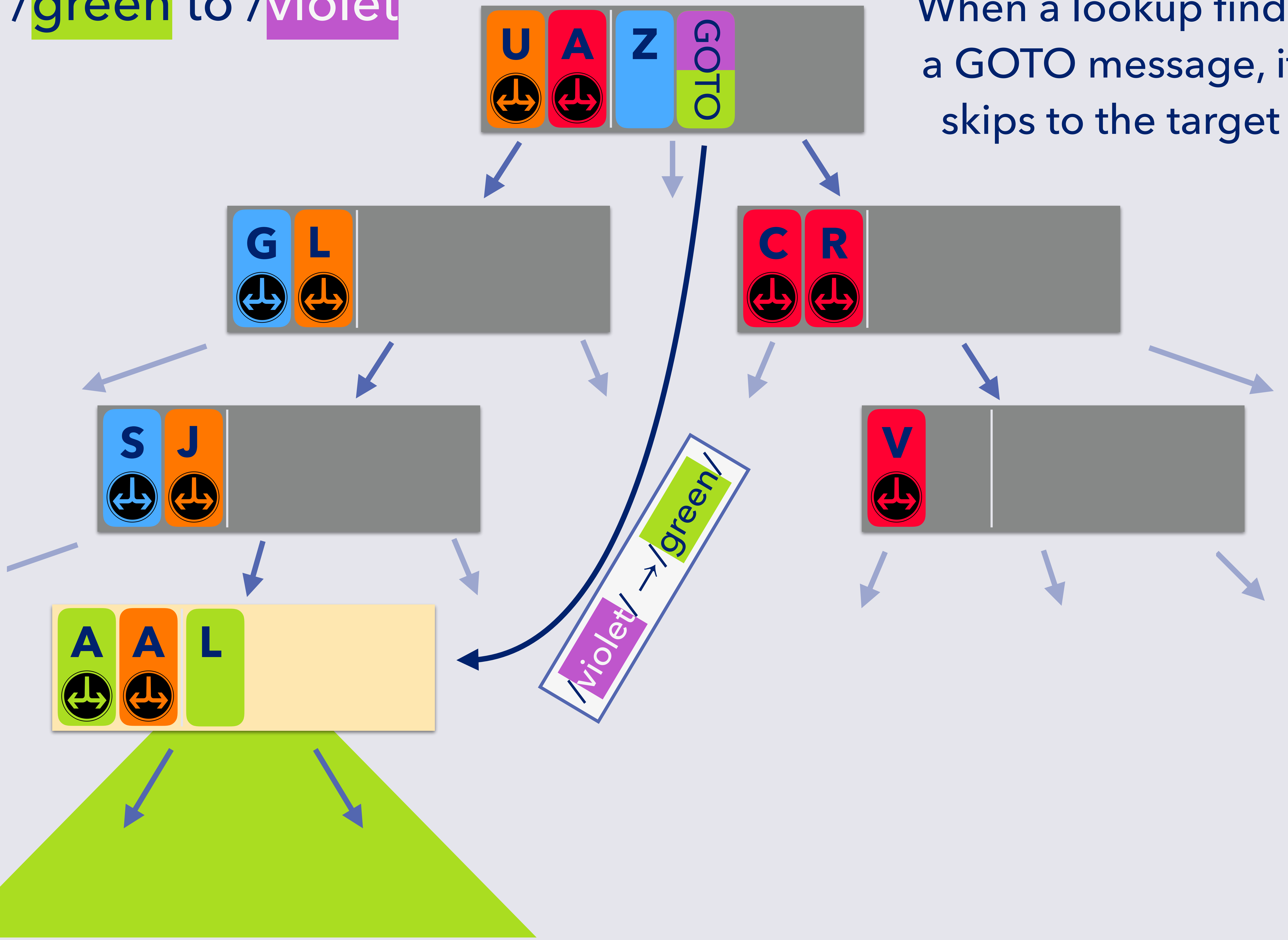
Reducing Copy Latency in B$^{\varepsilon}$-DAGs

Read(/violet/H)

Copy /green to /violet

When a lookup finds a GOTO message, it skips to the target

1. Flush messages to node covering /green subtree
2. Insert a GOTO message



GOTO

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation

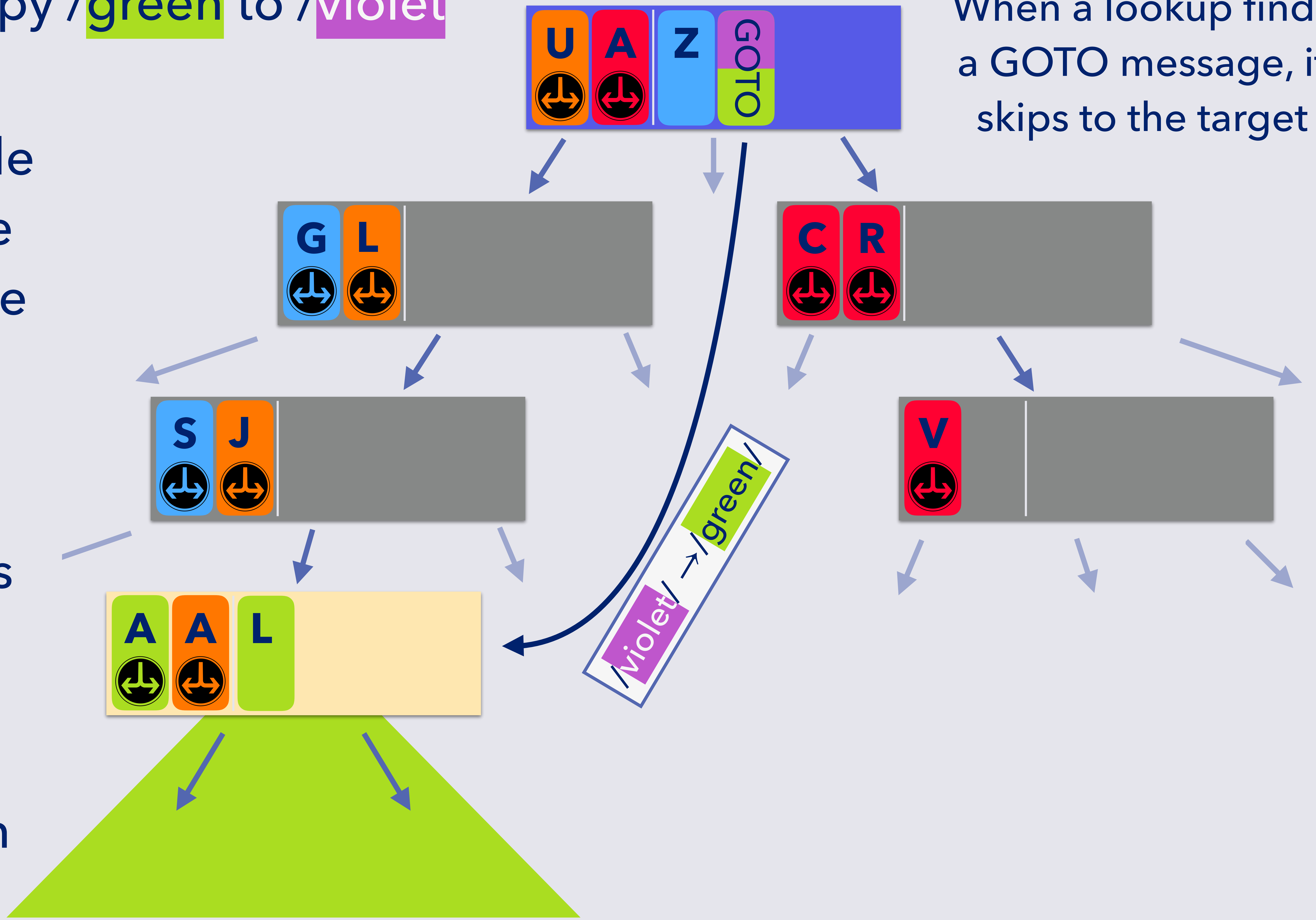
Reducing Copy Latency in B$^{\varepsilon}$-DA

Read(/violet/H)

Copy /green to /violet

When a lookup finds a GOTO message, it skips to the target

1. Flush messages to node covering /green subtree
2. Insert a GOTO message



GOTO

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation

Reducing Copy Latency in B$^{\varepsilon}$-DA

Copy /green to /violet

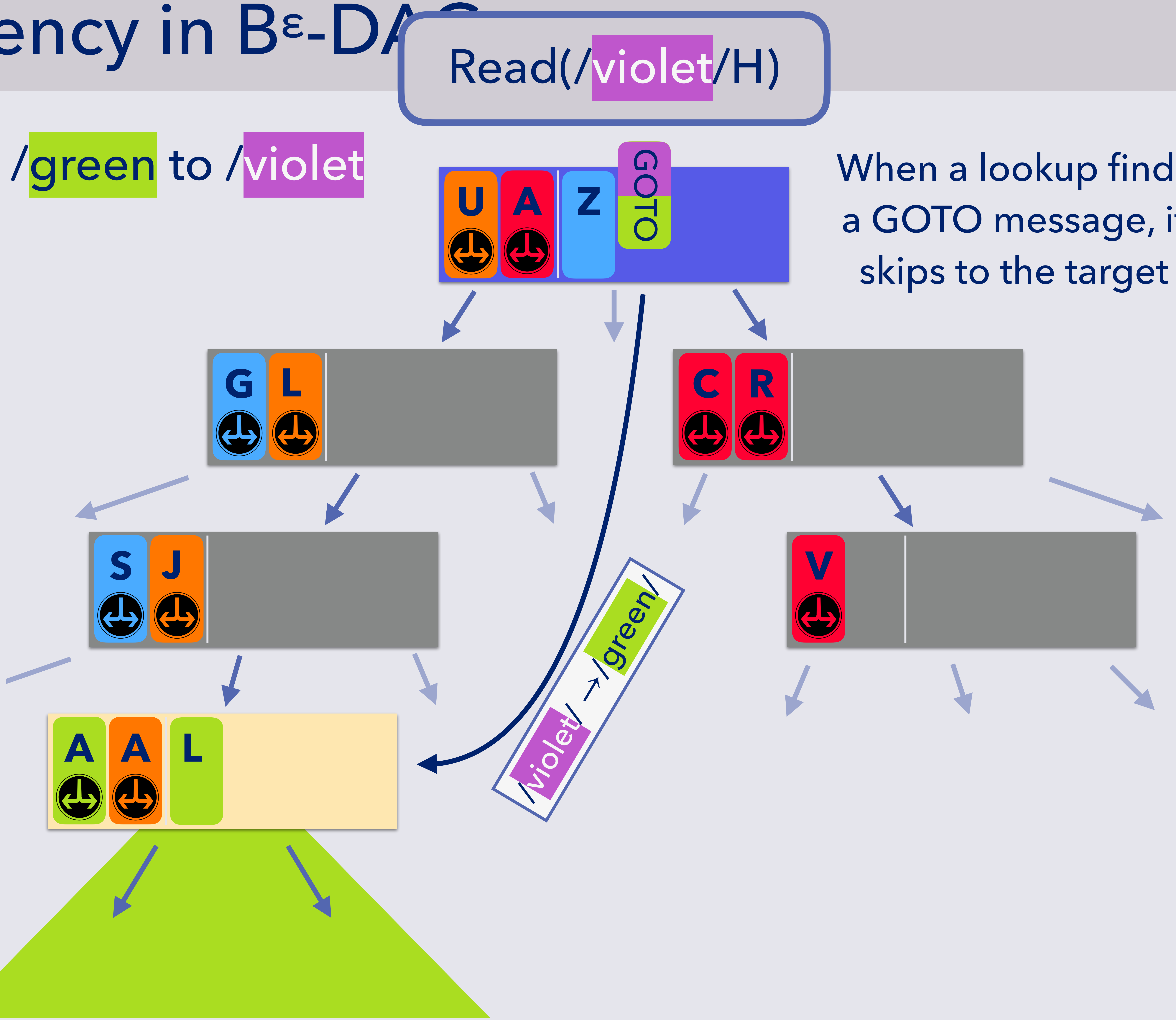
1. Flush messages to node covering /green subtree
2. Insert a GOTO message

GOTO

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation

When a lookup finds a GOTO message, it skips to the target

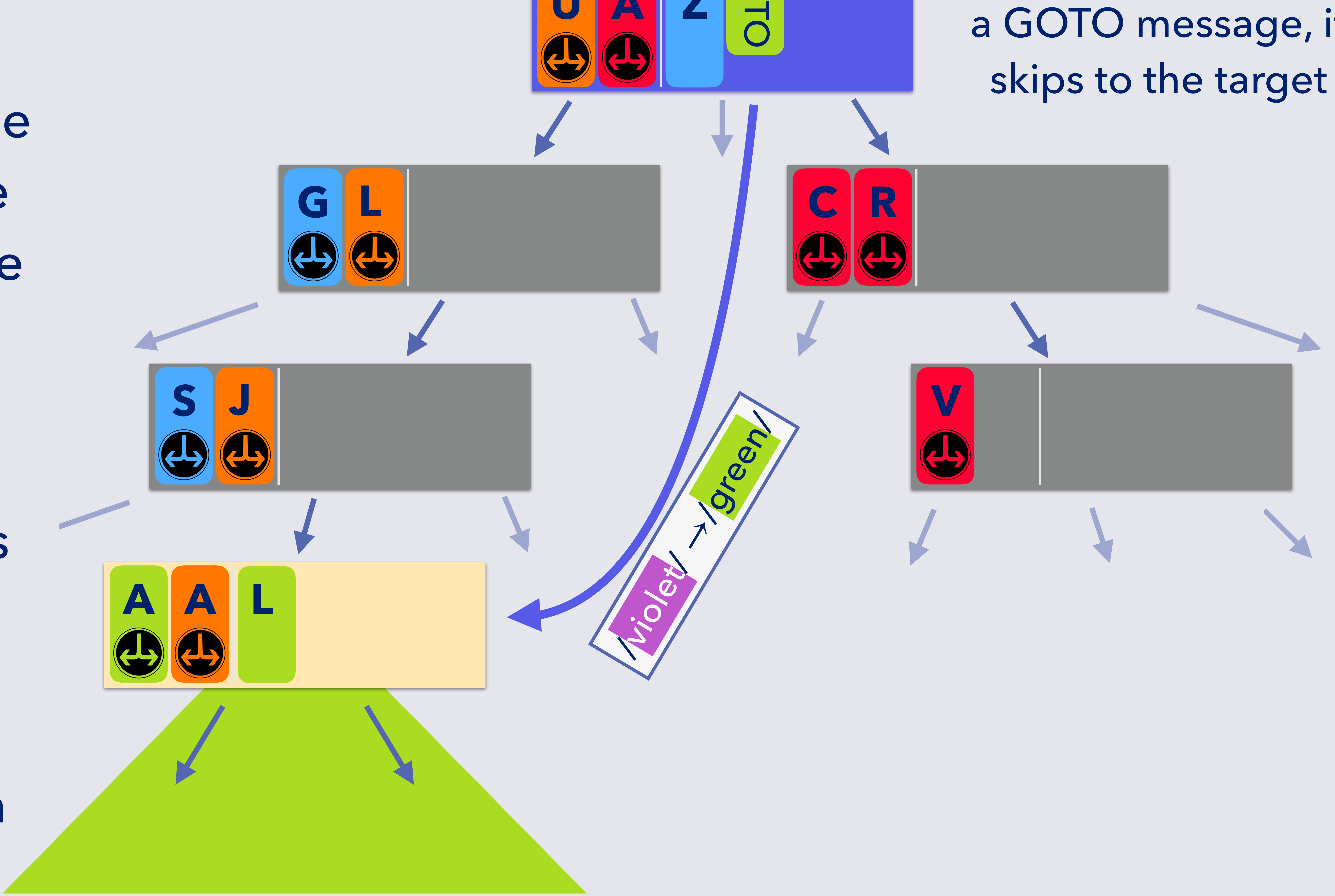


Reducing Copy Latency in B$^{\varepsilon}$-DAG

Read(/violet/H)

Copy /green to /violet

When a lookup finds a GOTO message, it skips to the target



1. Flush messages to node covering /green subtree
2. Insert a GOTO message

GOTO

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation

Reducing Copy Latency in Bε-DAGs

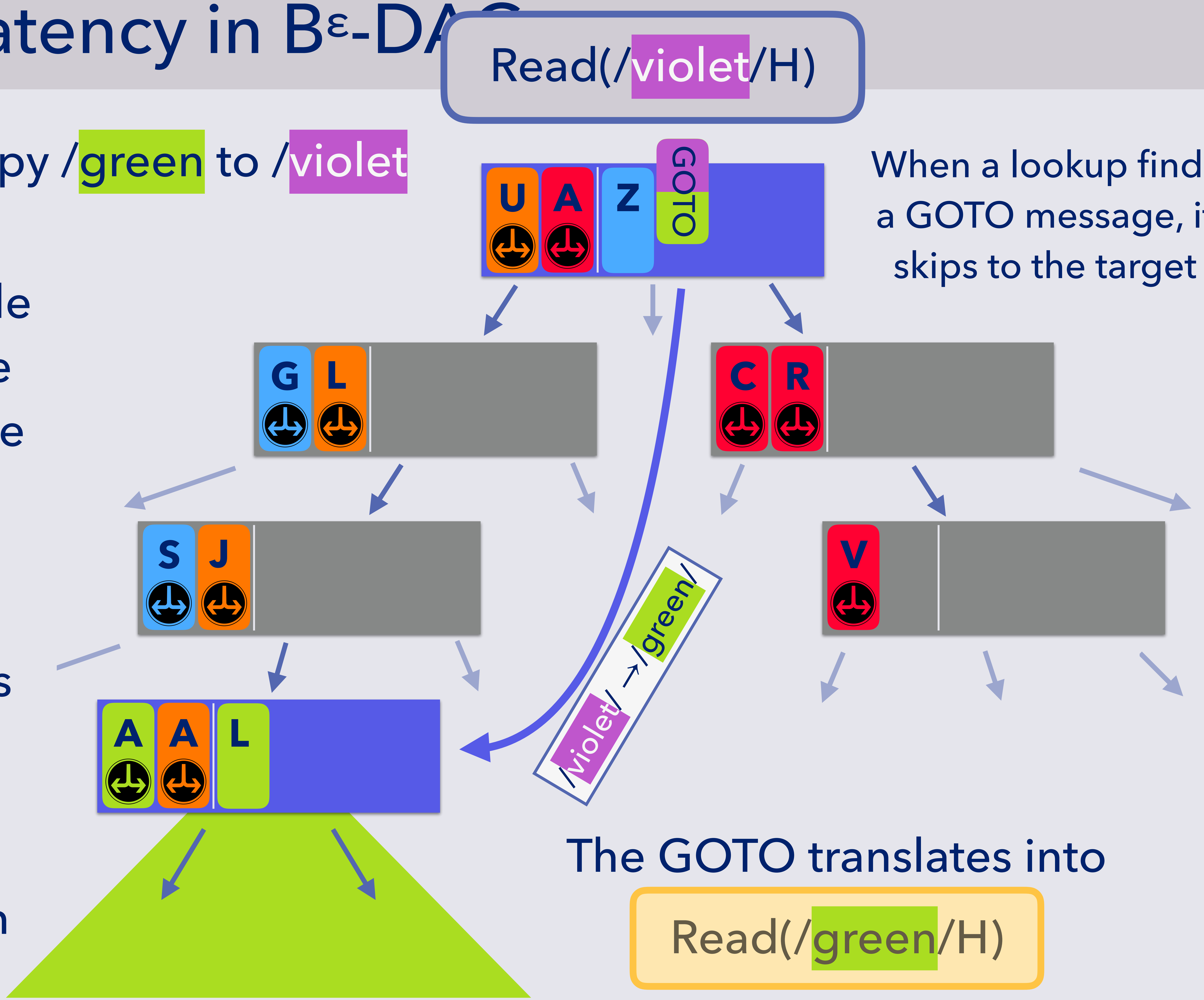
Copy /green to /violet

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

GOTO

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation



When a lookup finds a GOTO message, it skips to the target

The GOTO translates into

Read(/green/H)

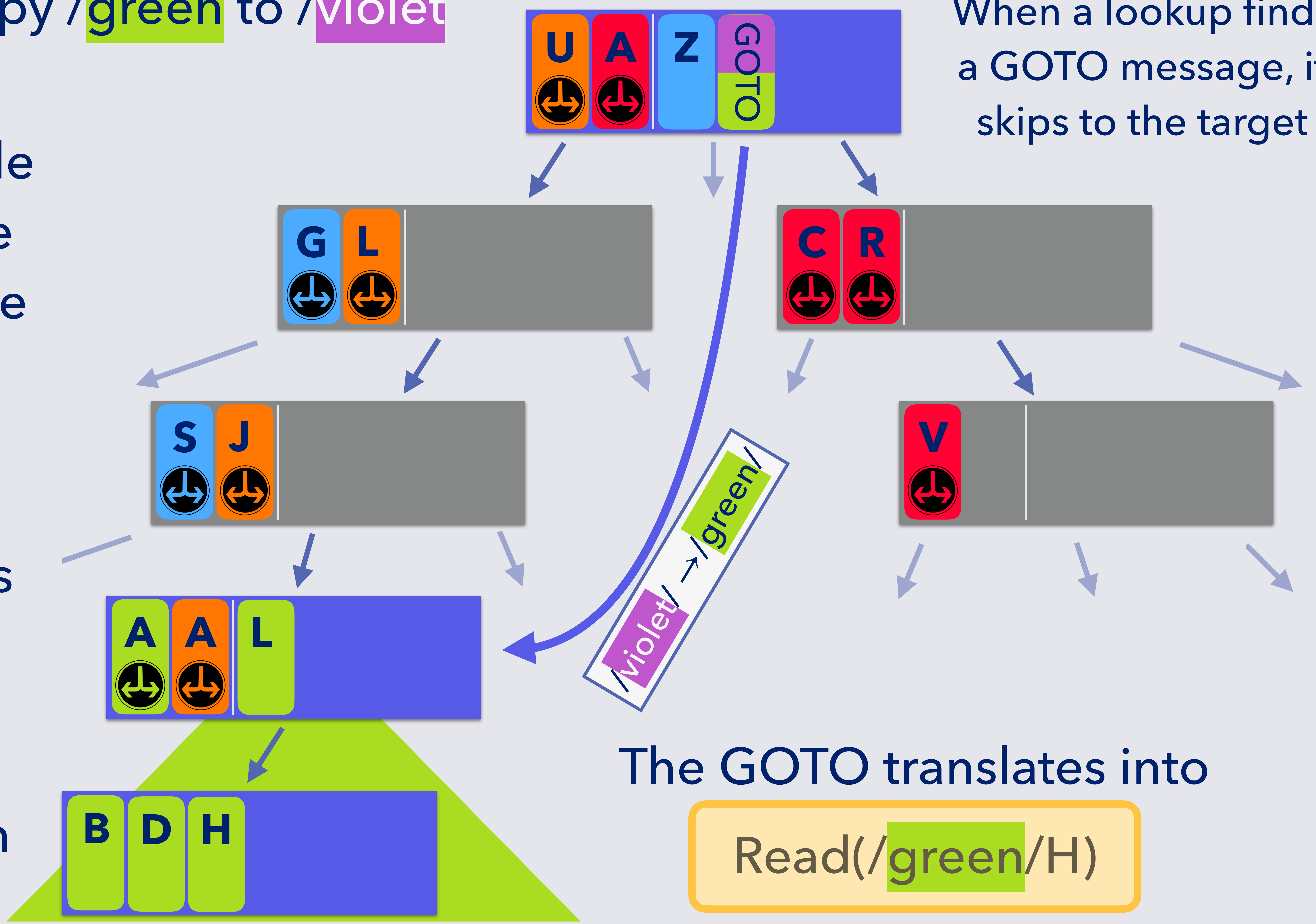
Reducing Copy Latency in B^{ε}-DAG

Read(/violet/H)

Copy /green to /violet

When a lookup finds a GOTO message, it skips to the target

1. Flush messages to node covering /green subtree
2. Insert a GOTO message



GOTO

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation

The GOTO translates into

Read(/green/H)

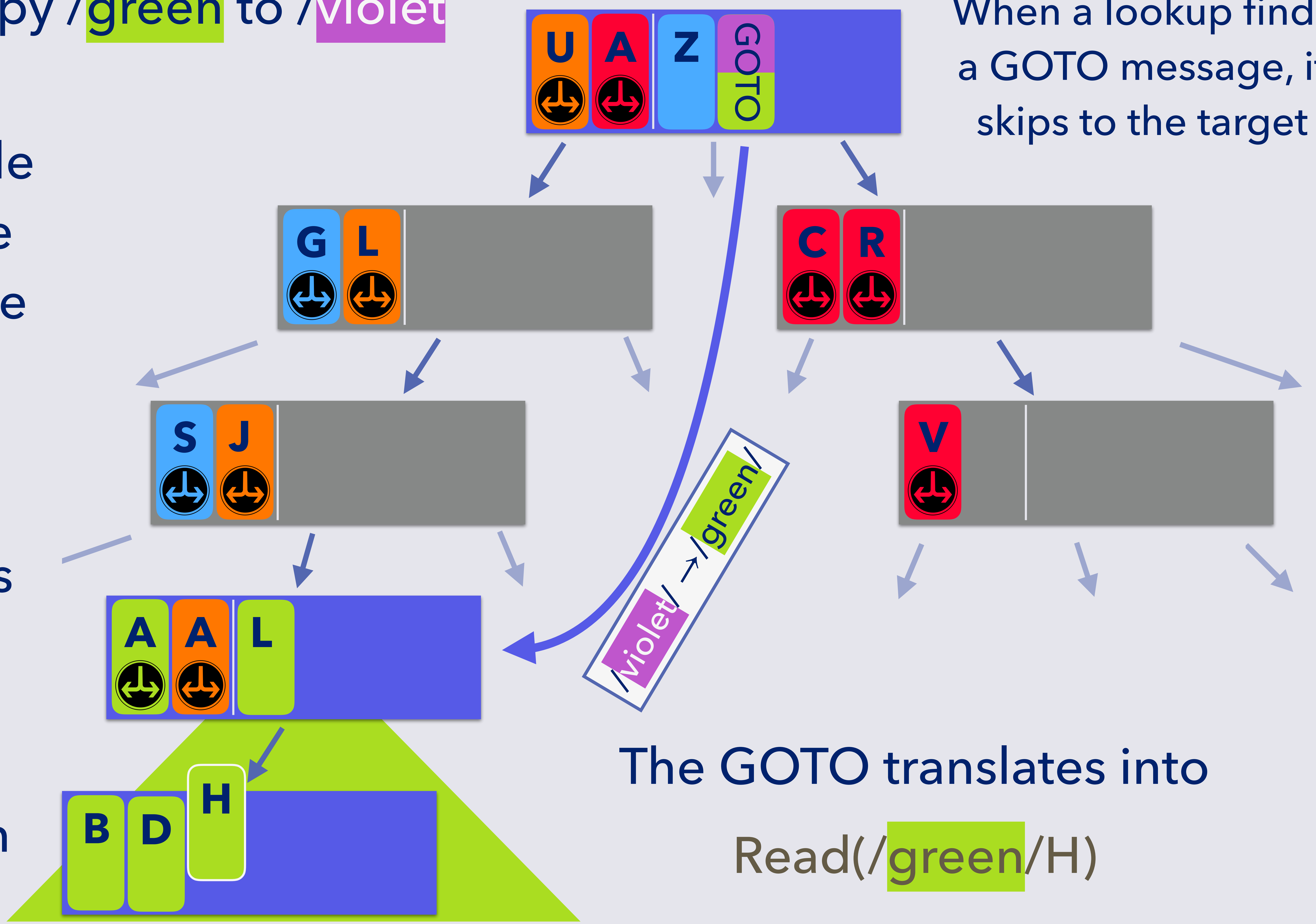
Reducing Copy Latency in Bε-DA

Read(/violet/H)

Copy /green to /violet

When a lookup finds a GOTO message, it skips to the target

1. Flush messages to node covering /green subtree
2. Insert a GOTO message



A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation

The GOTO translates into

Read(/green/H)

Reducing Copy Latency in Bε-DAGs

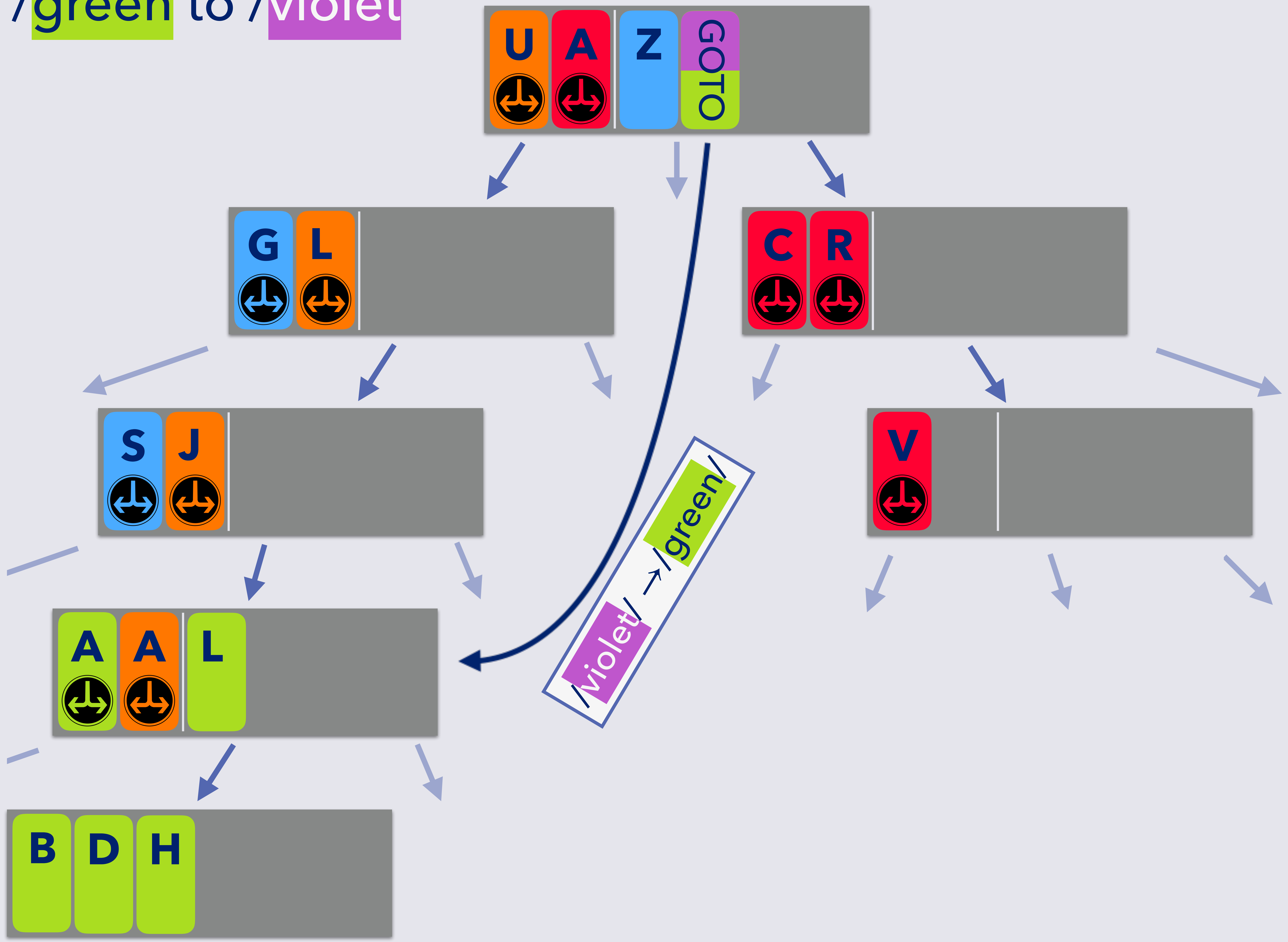
Copy /green to /violet

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

GOTO

A GOTO message changes the structure of the tree

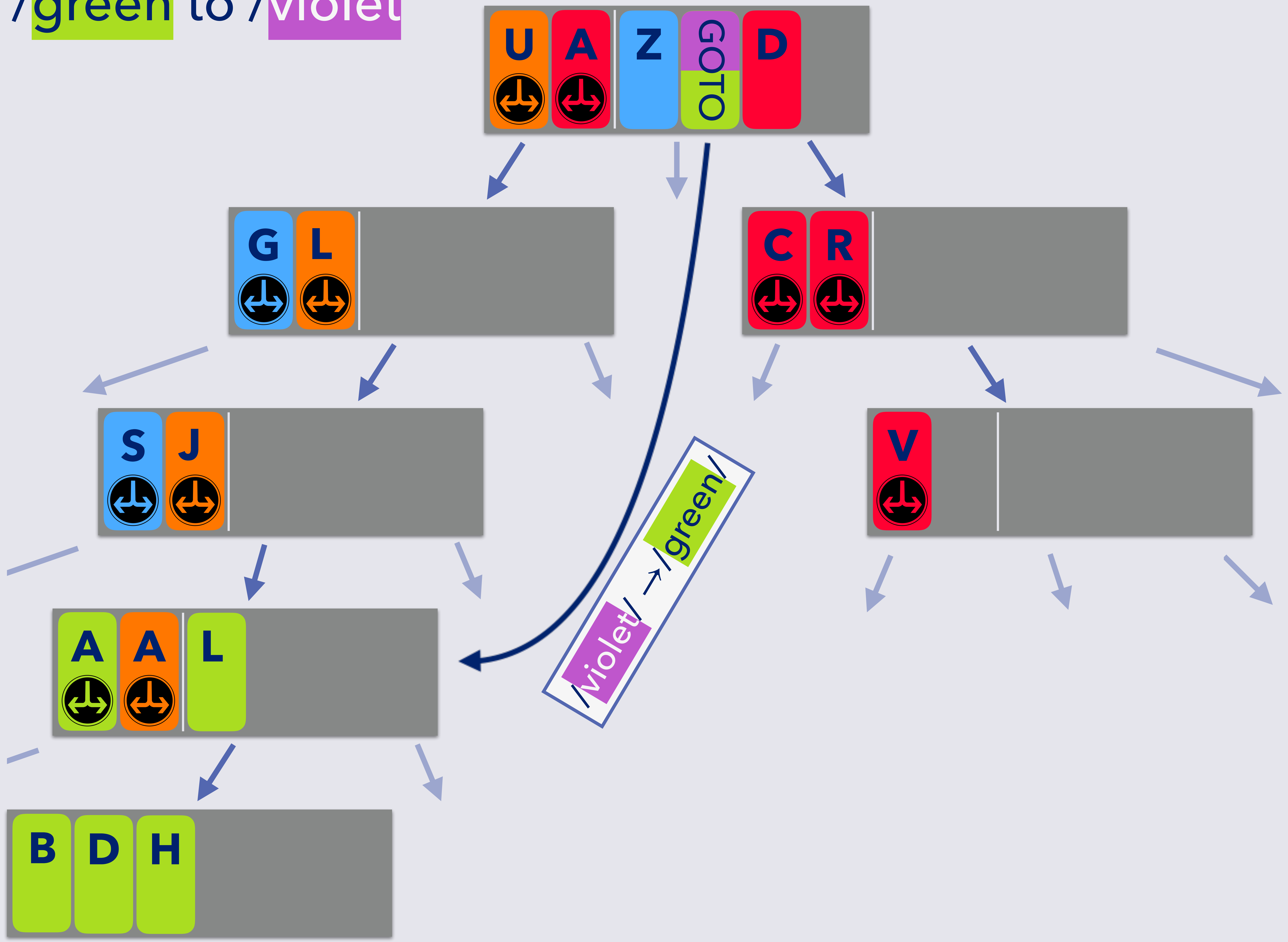
It functions as a pivot including prefix translation



Reducing Copy Latency in Bε-DAGs

Copy /green to /violet

1. Flush messages to node covering /green subtree
2. Insert a GOTO message



GOTO

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation

Reducing Copy Latency in Bε-DAGs

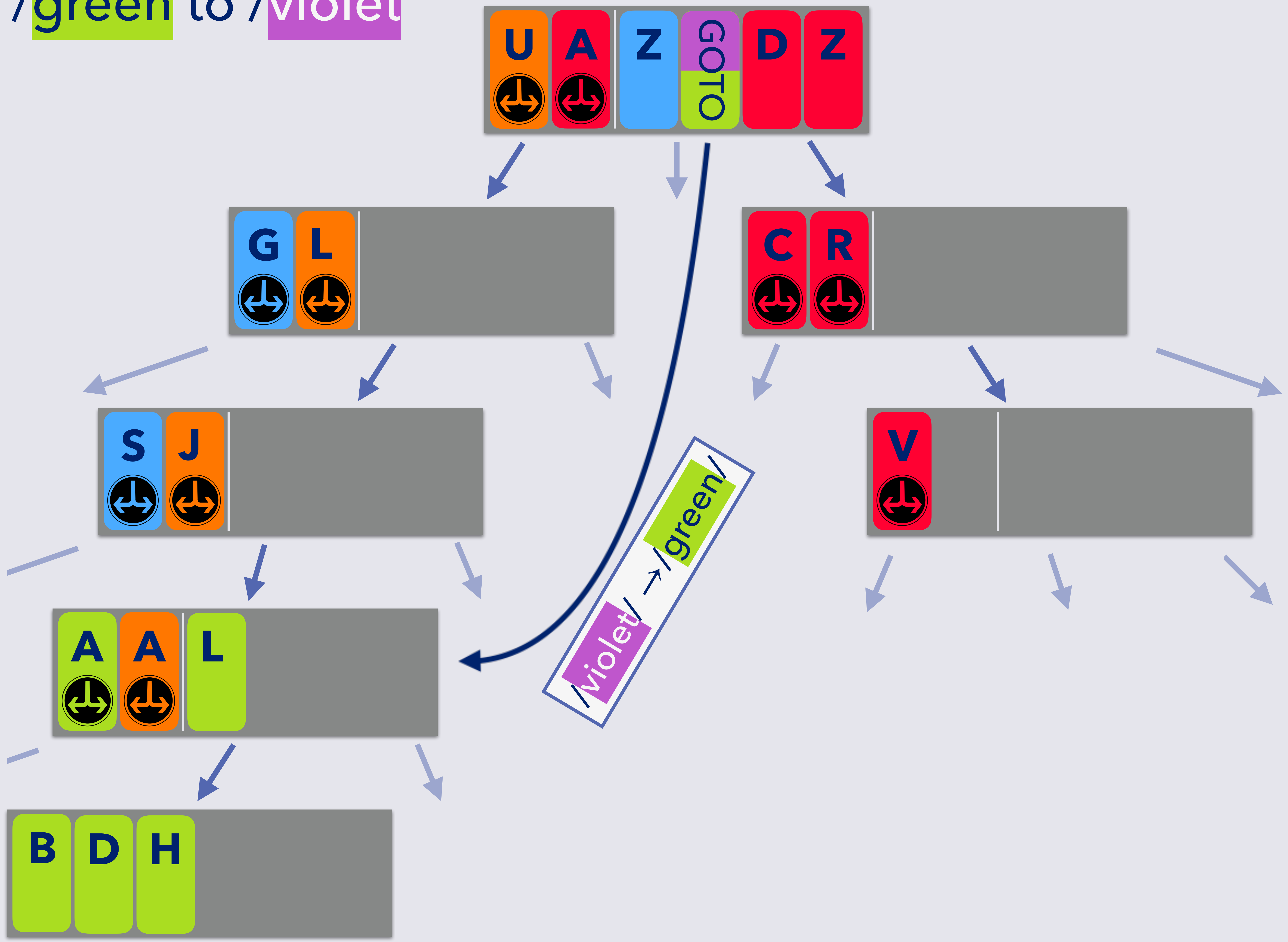
Copy /green to /violet

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

GOTO

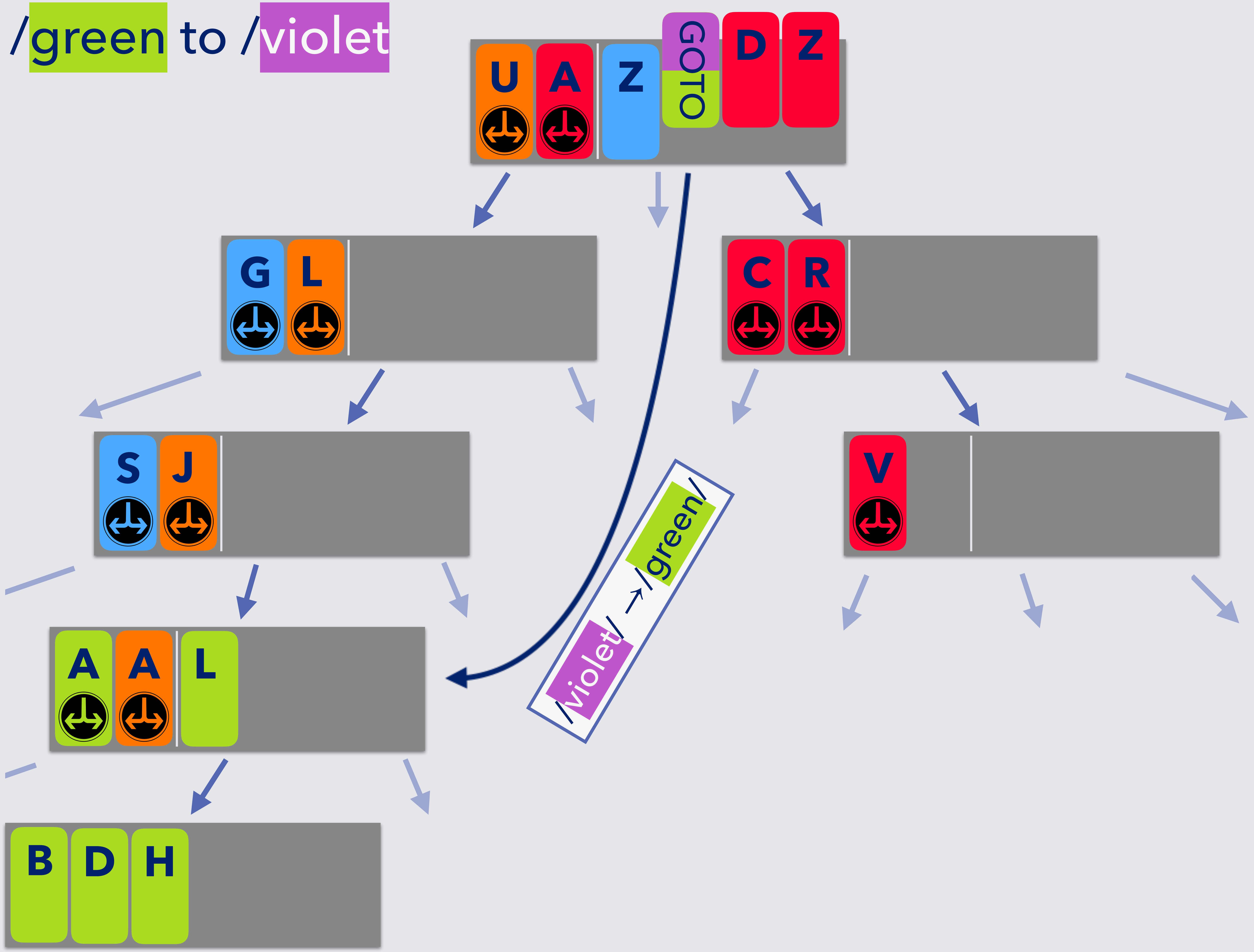
A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation



Reducing Copy Latency in Bε-DAGs

Copy /green to /violet



1. Flush messages to node covering /green subtree
2. Insert a GOTO message

GOTO

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation

Reducing Copy Latency in B^ε-DAGs

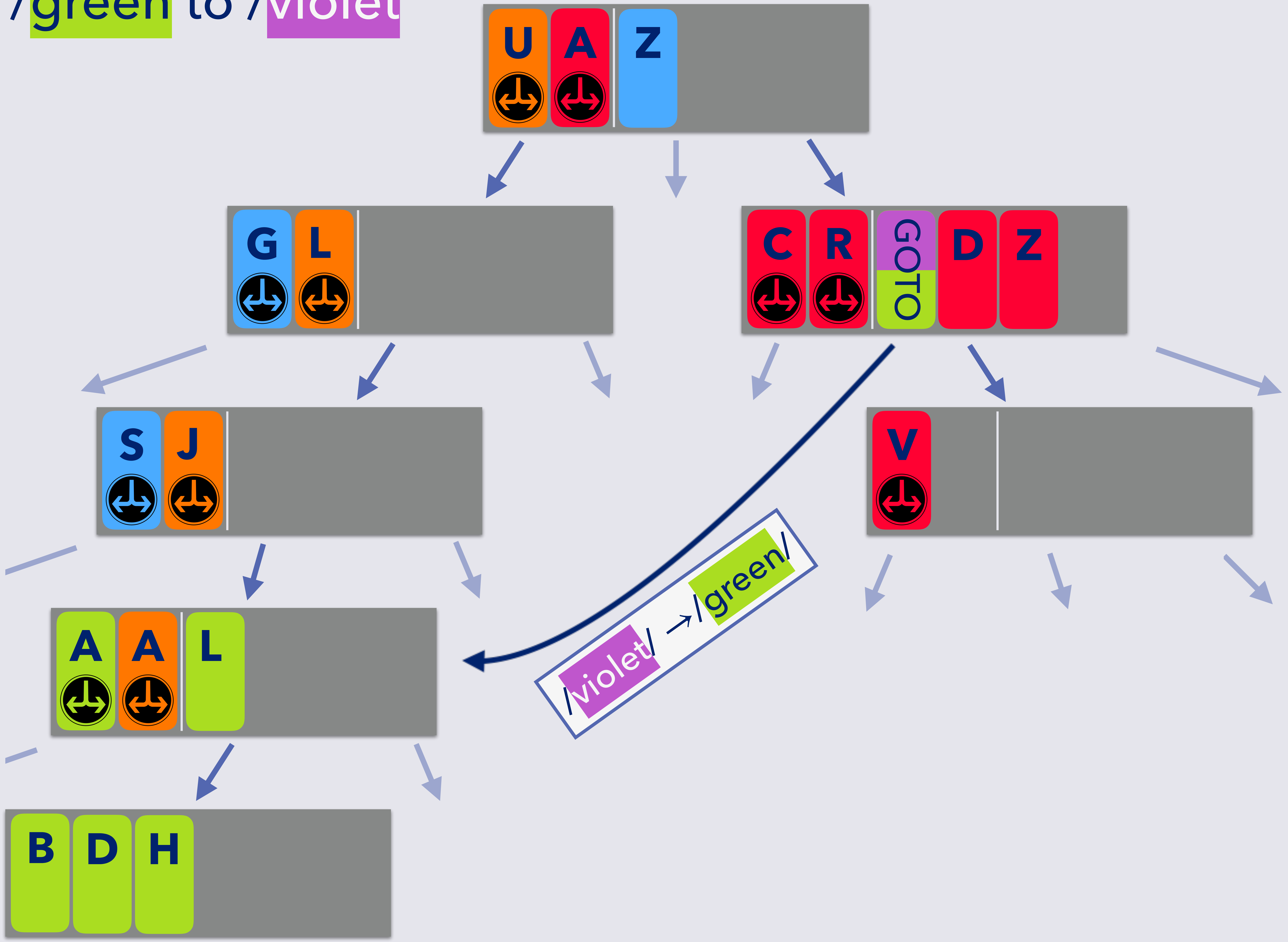
Copy /green to /violet

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

GOTO

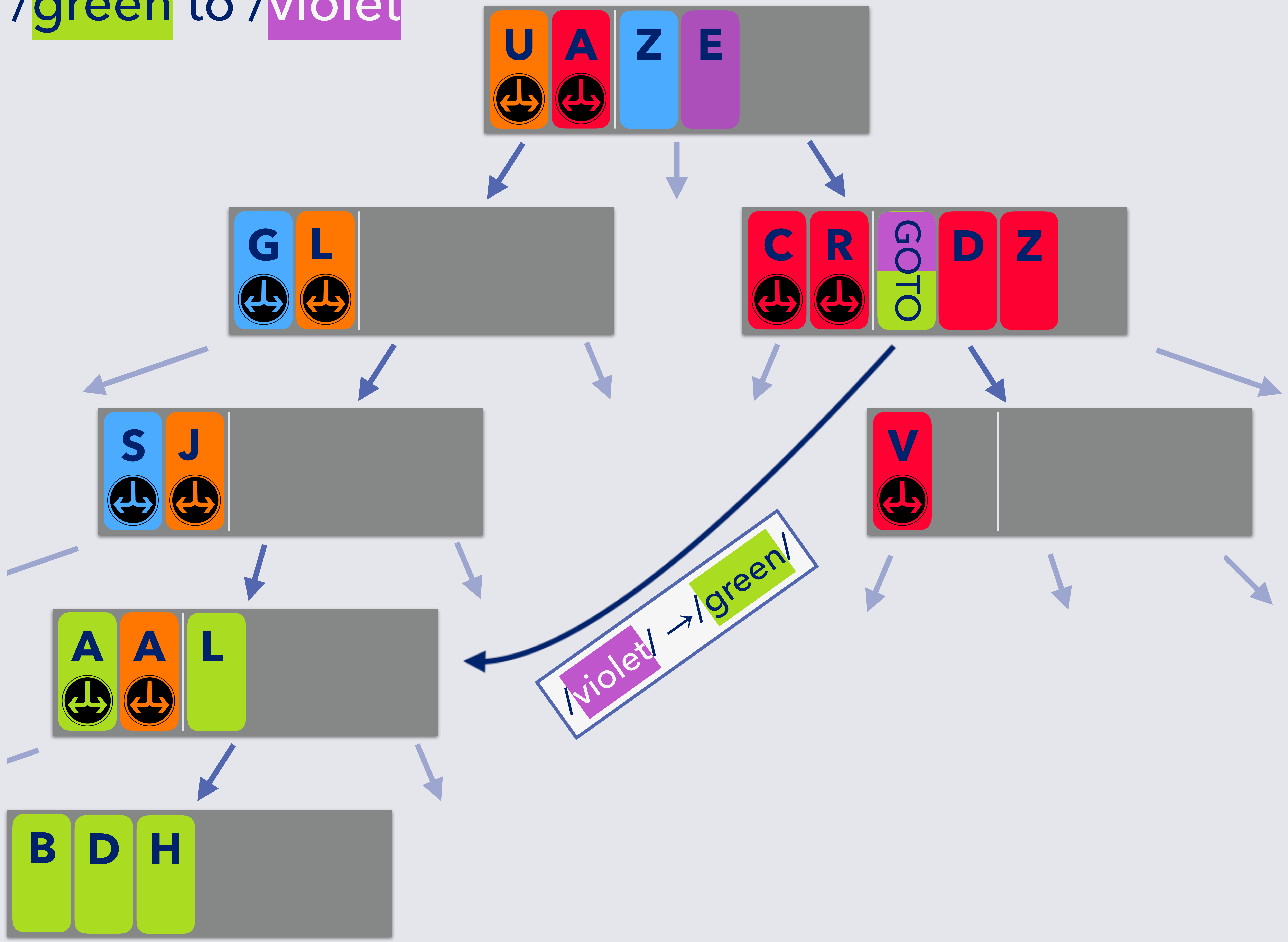
A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation



Reducing Copy Latency in Bε-DAGs

Copy /green to /violet



1. Flush messages to node covering /green subtree
2. Insert a GOTO message

GOTO

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation

Reducing Copy Latency in Bε-DAGs

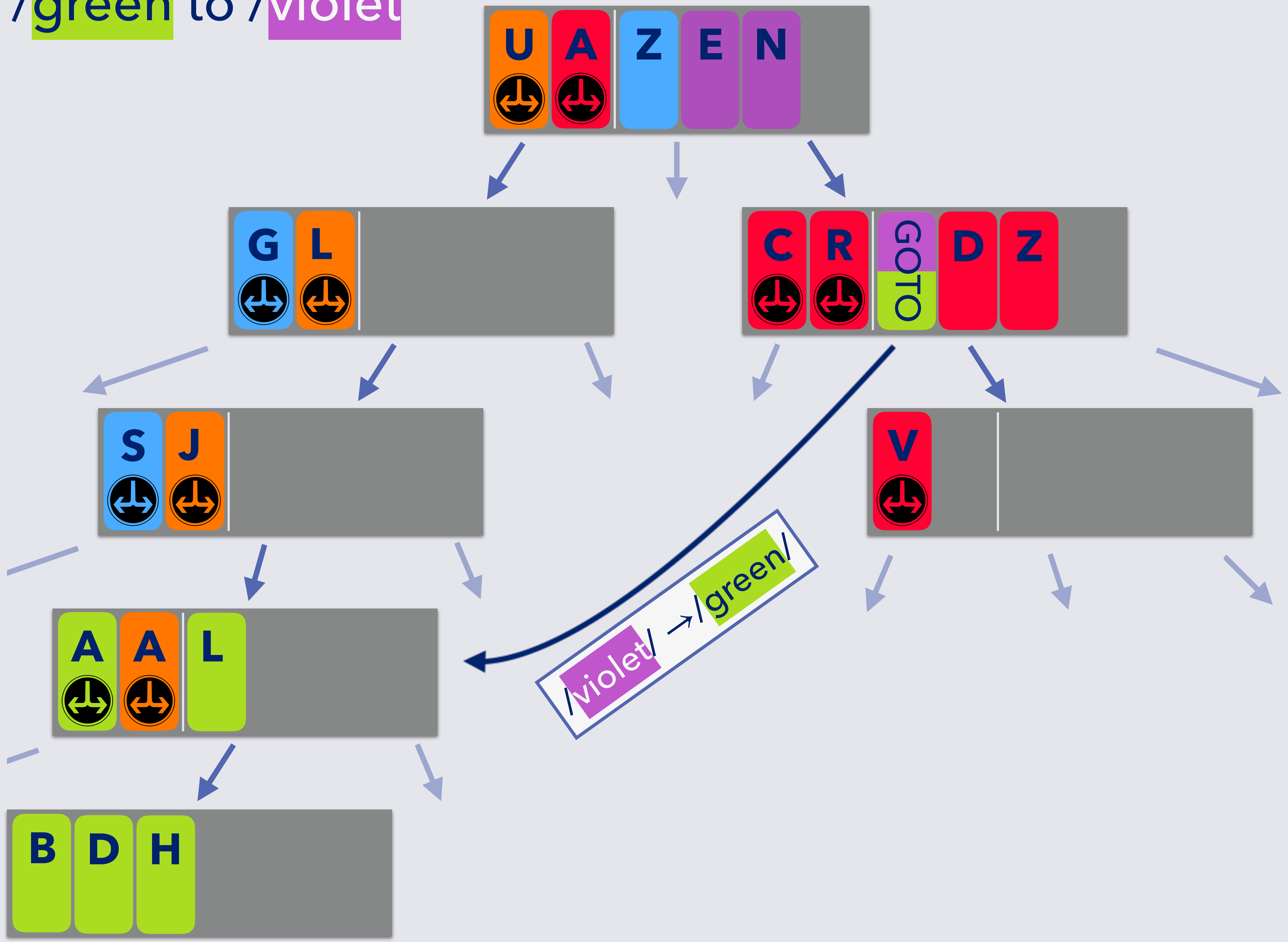
Copy /green to /violet

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

GOTO

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation



Reducing Copy Latency in B$^{\varepsilon}$-DAGs

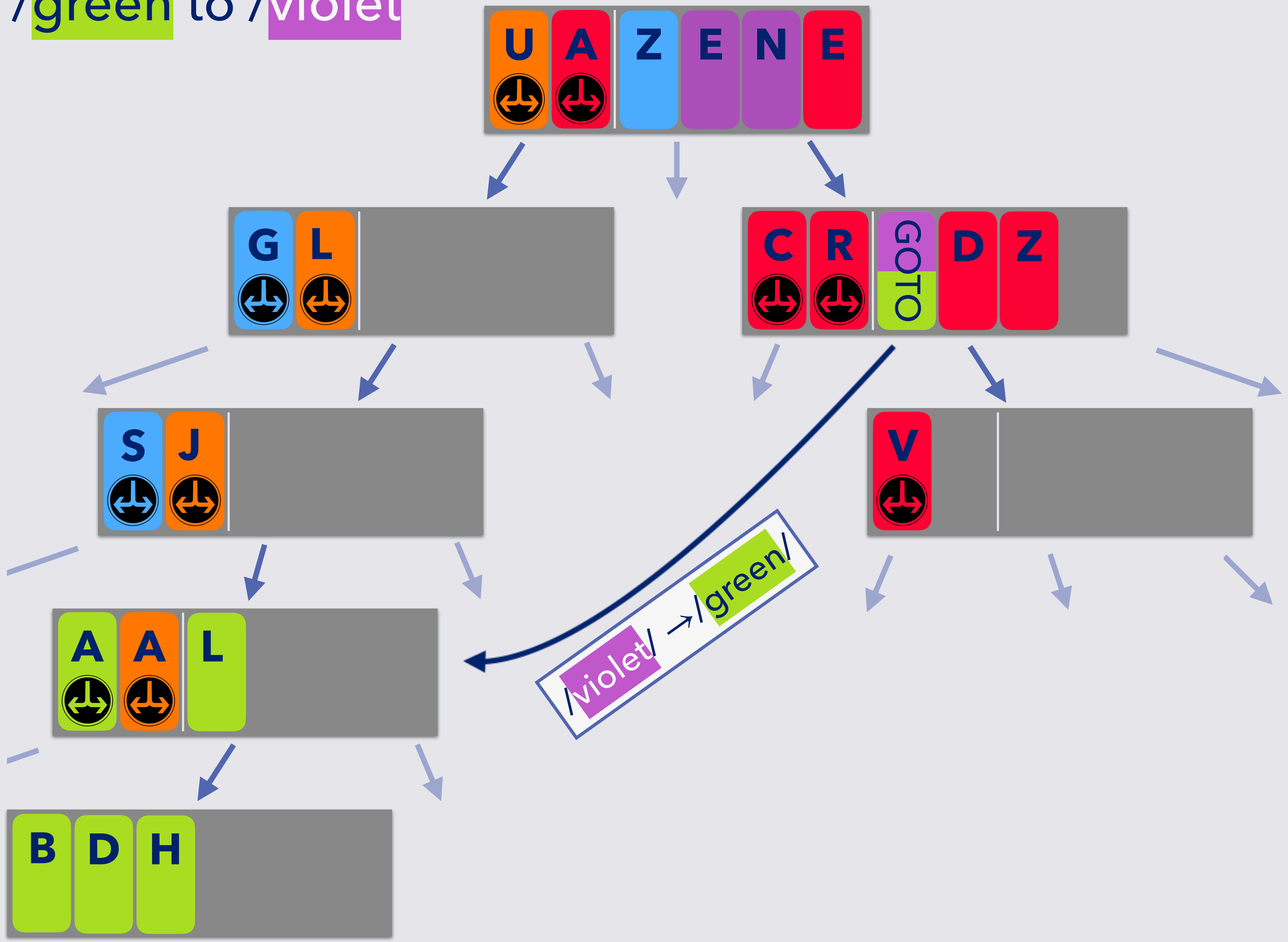
Copy /green to /violet

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

GOTO

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation



Reducing Copy Latency in Bε-DAGs

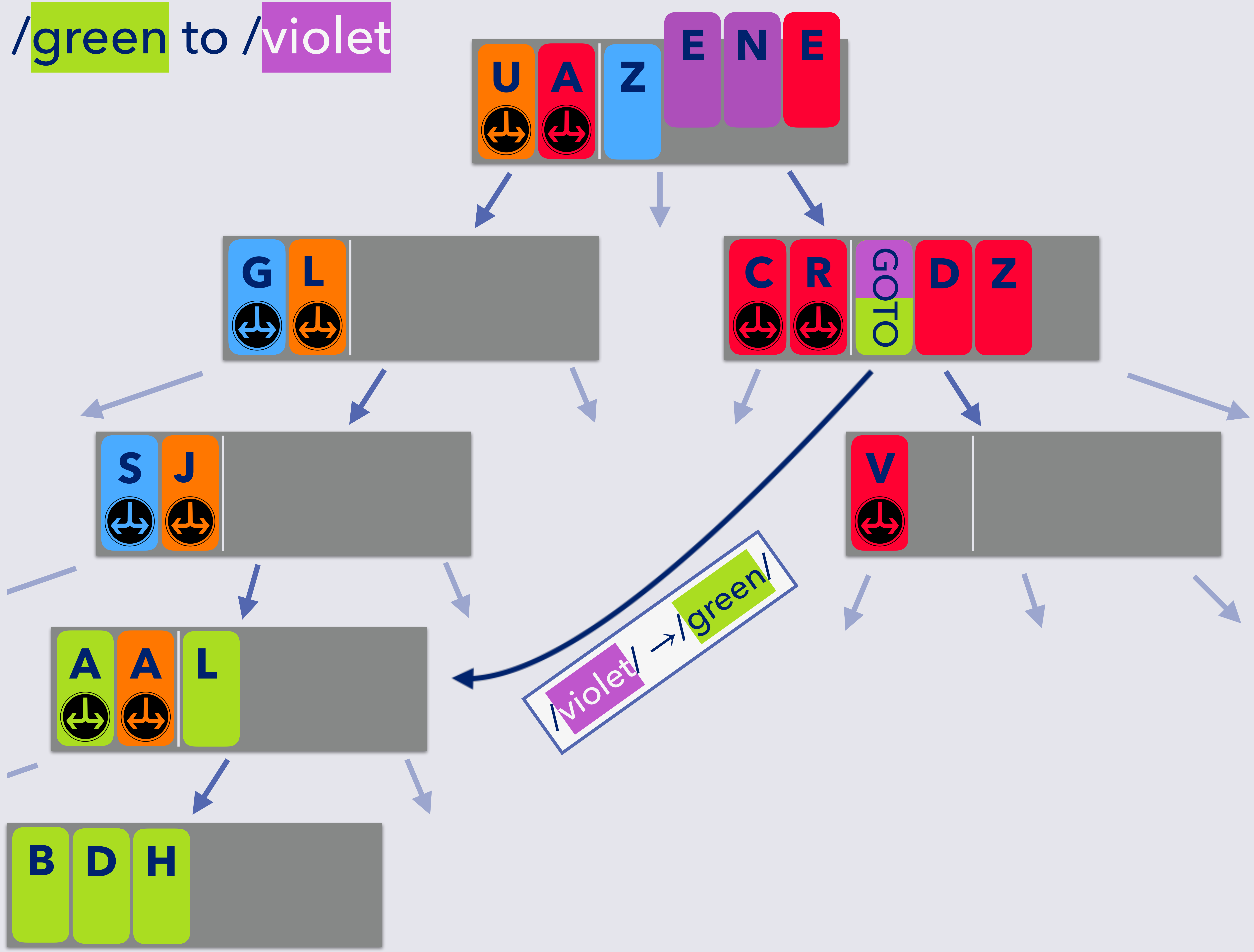
Copy /green to /violet

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

GOTO

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation



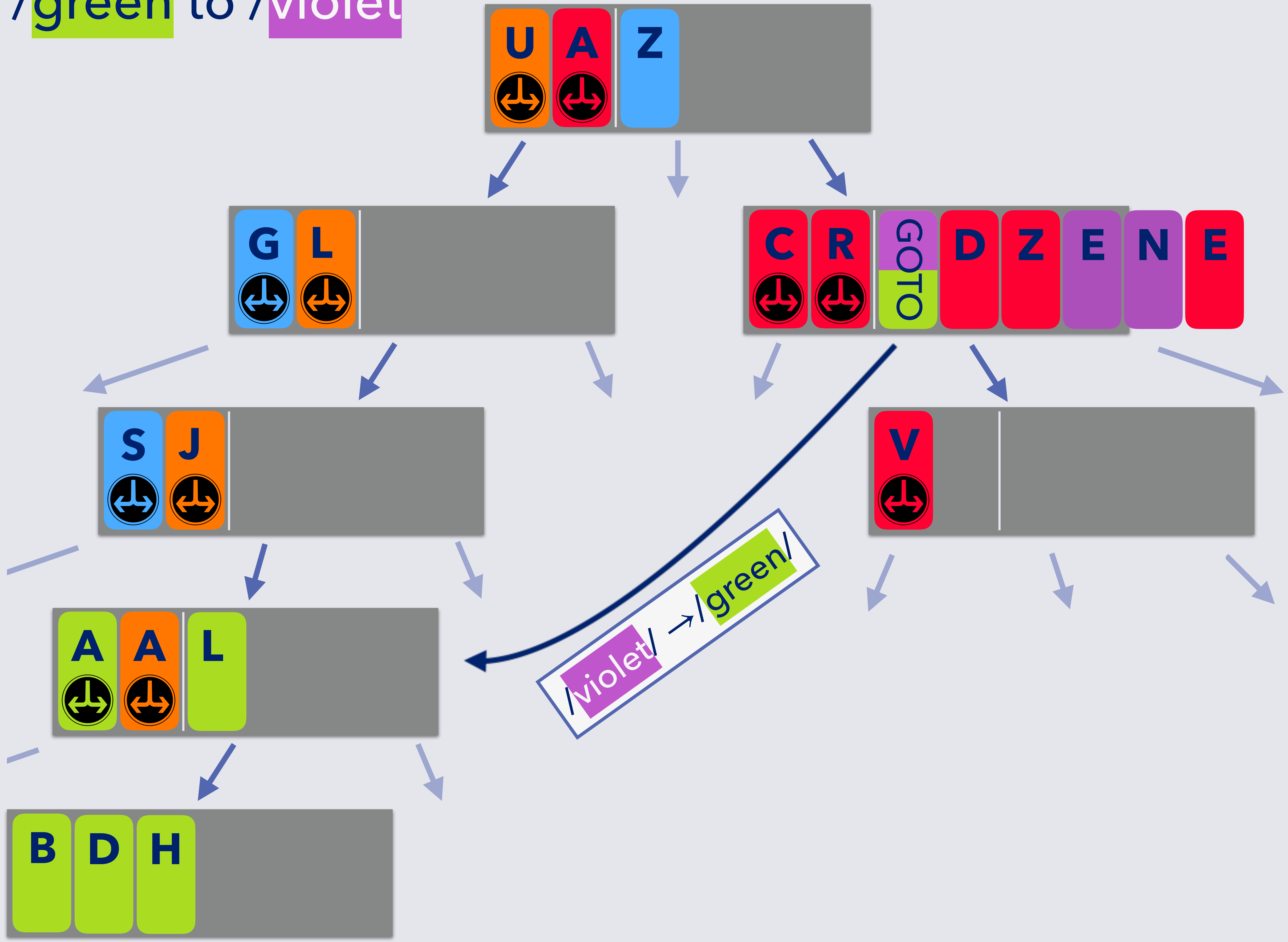
Reducing Copy Latency in Bε-DAGs

Copy /green to /violet

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation



Reducing Copy Latency in Bε-DAGs

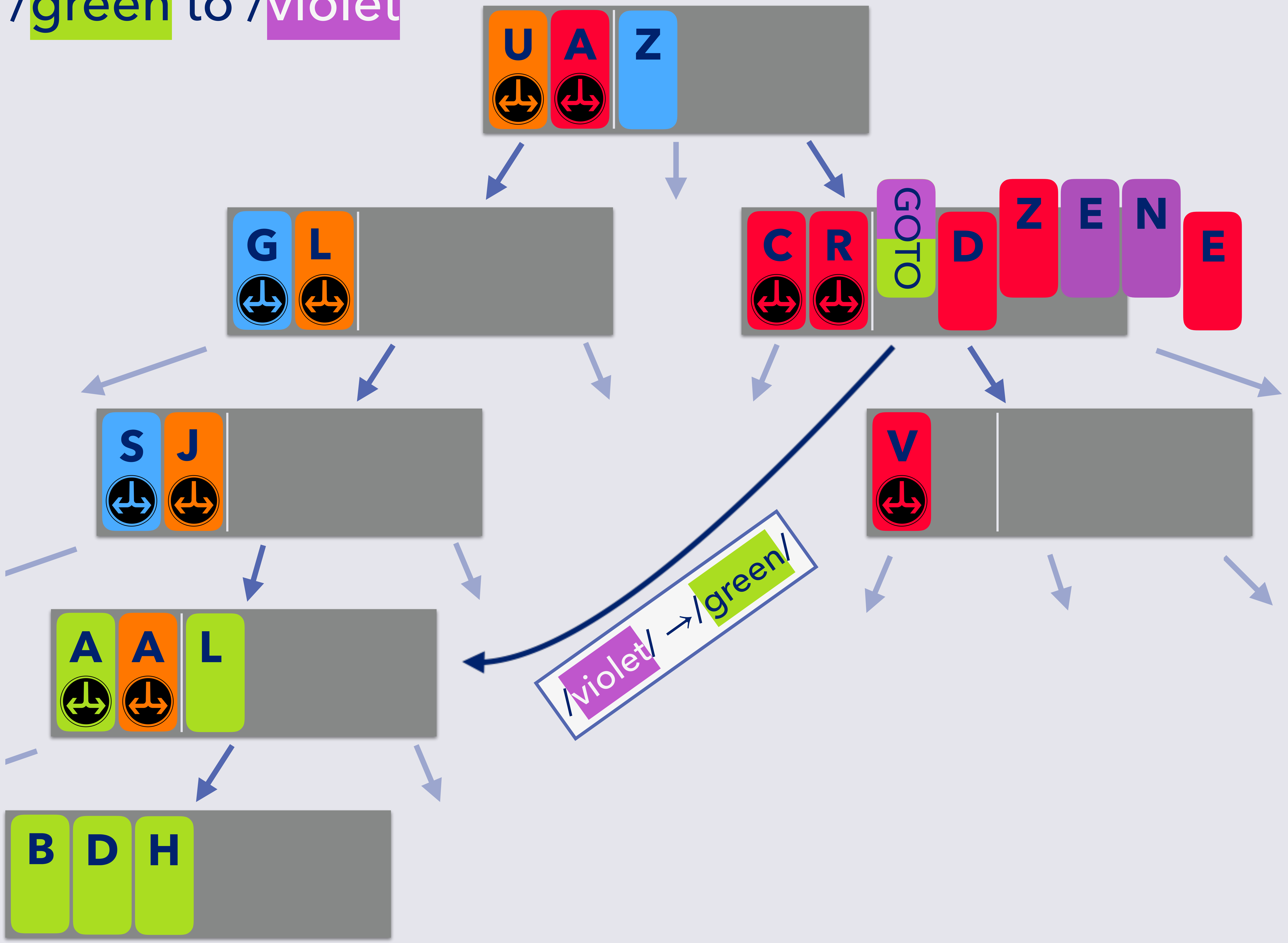
Copy /green to /violet

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

GOTO

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation



Reducing Copy Latency in Bε-DAGs

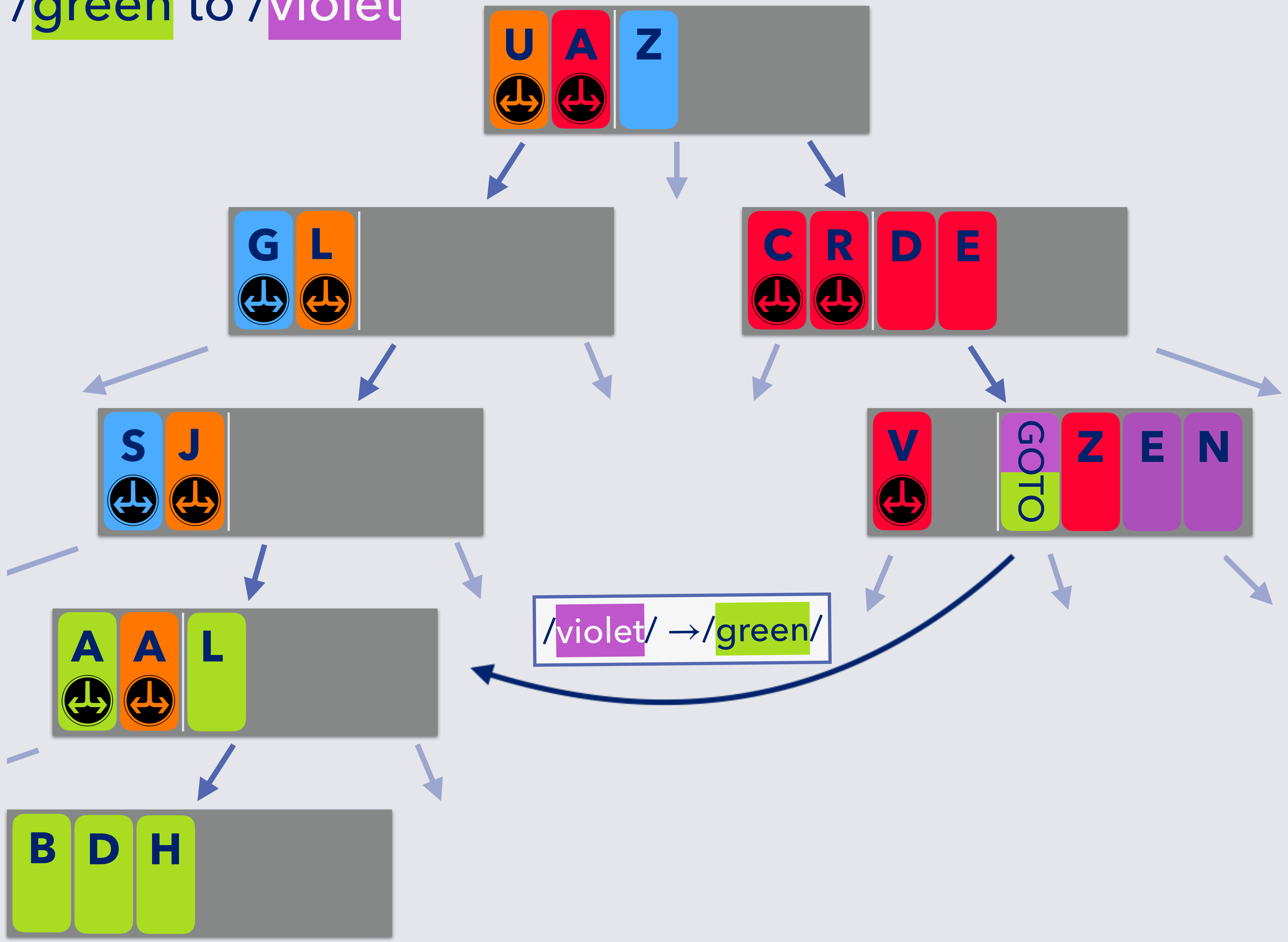
Copy /green to /violet

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

GOTO

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation



Reducing Copy Latency in Bε-DAGs

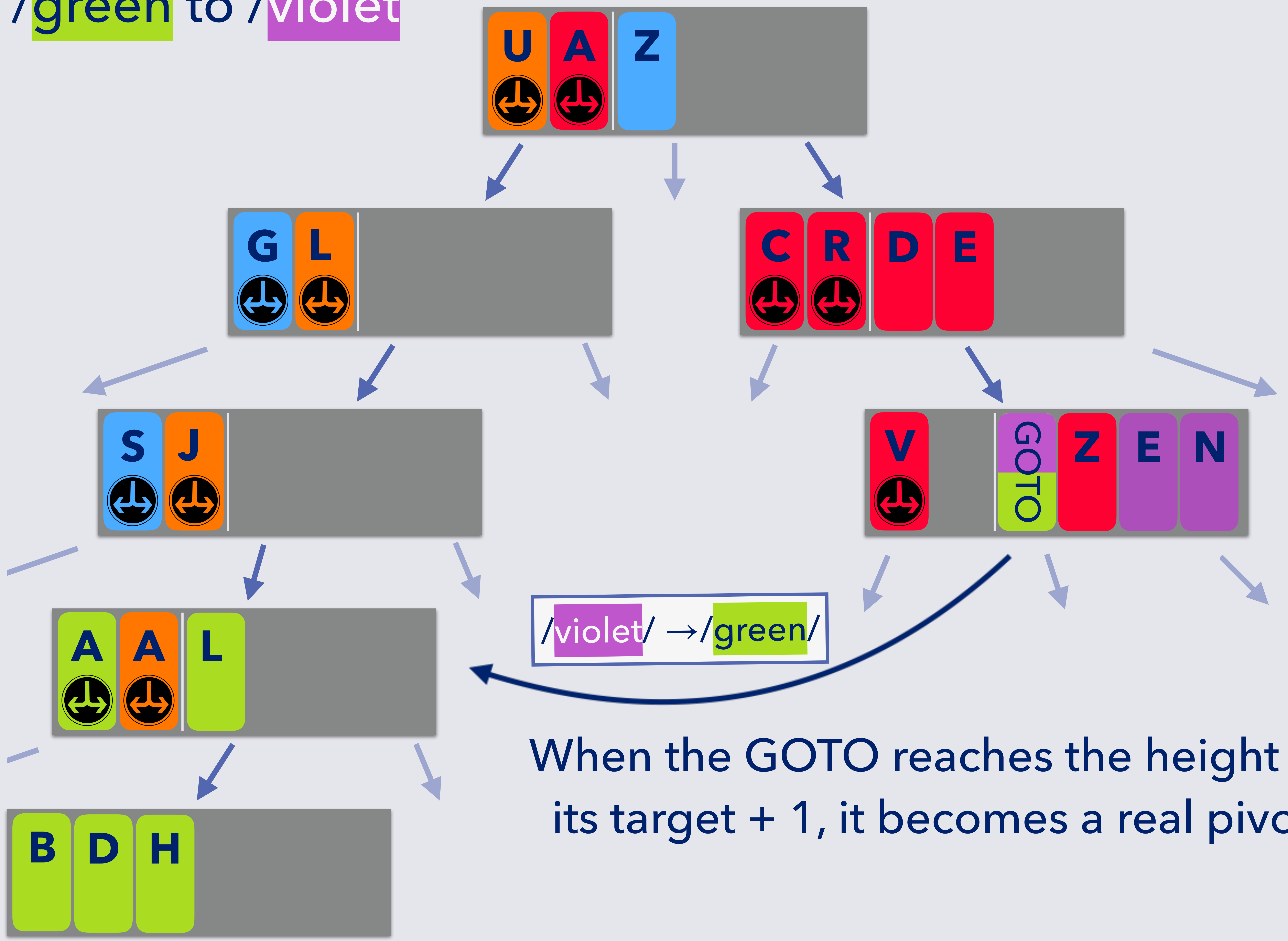
Copy /green to /violet

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

GOTO

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation



When the GOTO reaches the height of its target + 1, it becomes a real pivot

Reducing Copy Latency in Bε-DAGs

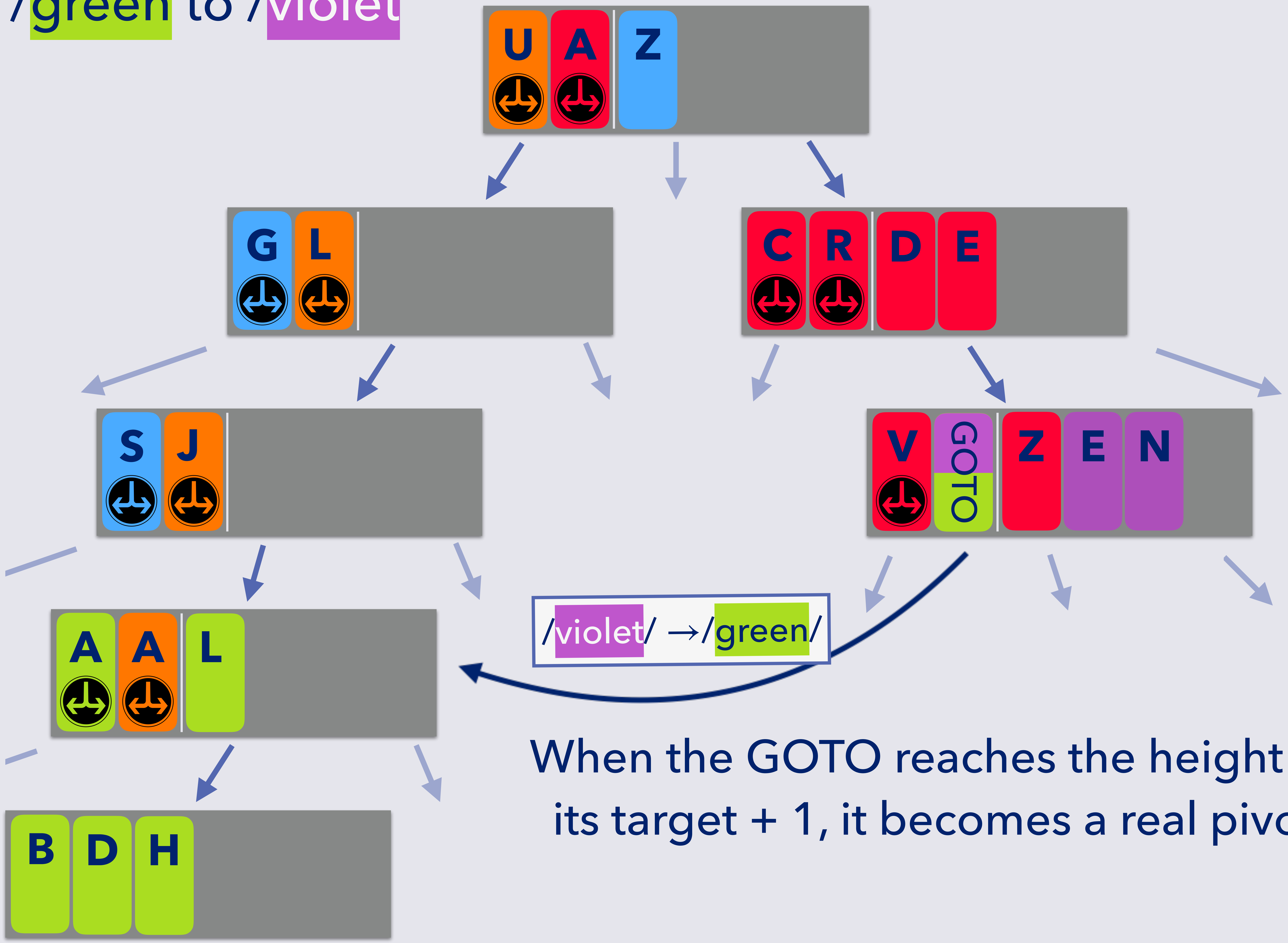
Copy /green to /violet

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

GOTO

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation



When the GOTO reaches the height of its target + 1, it becomes a real pivot

Reducing Copy Latency in Bε-DAGs

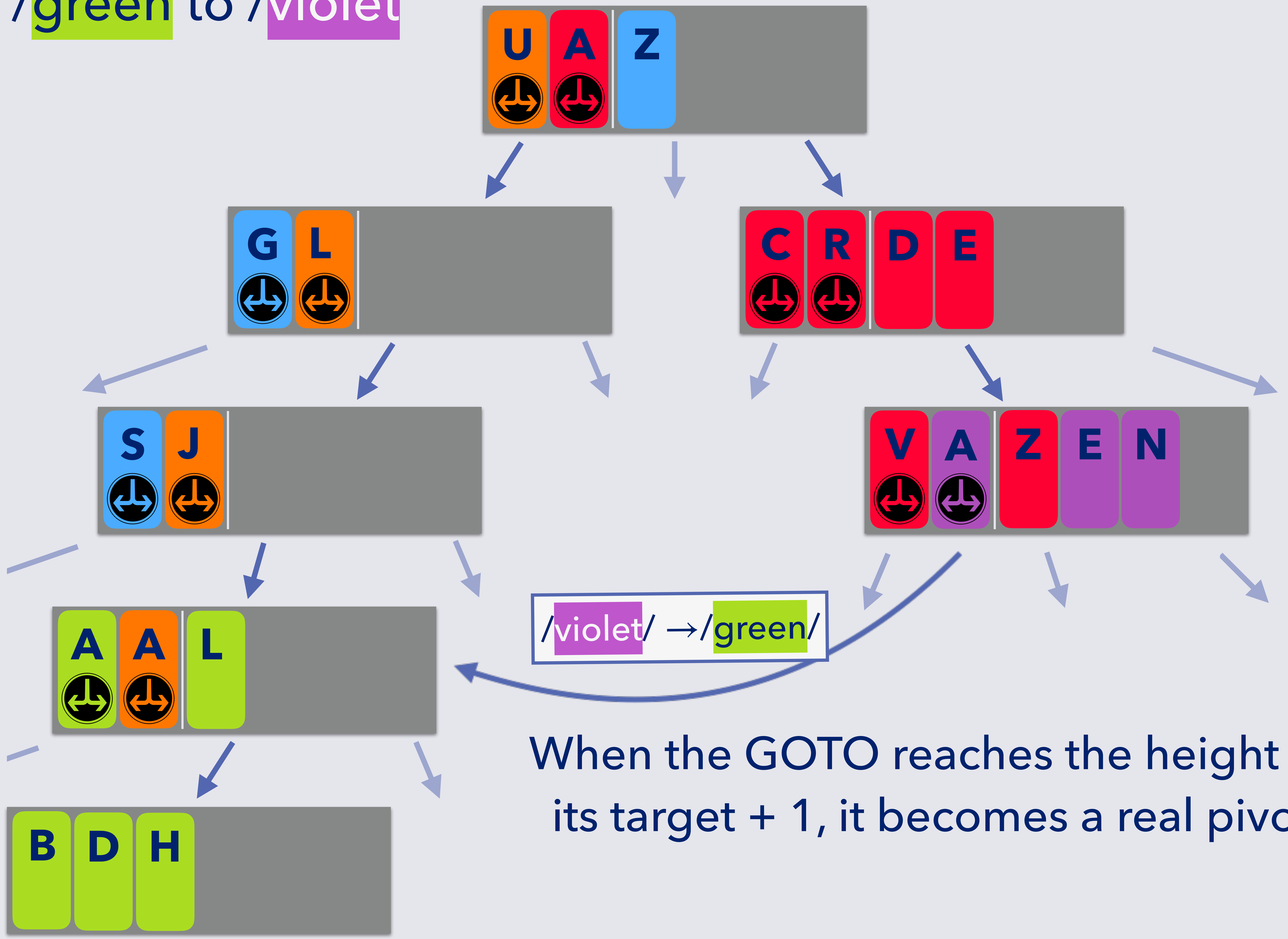
Copy /green to /violet

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

GOTO

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation



When the GOTO reaches the height of its target + 1, it becomes a real pivot

Logical Copy with Bε-DAGs

Performance Goals

Copy-on-Abundant-Write

Space efficient ✓

Low latency ?

Copy-specific

Bε-trees have large nodes

Fast writes ✓

⇒ Locality*

Fast reads ✓

General file system

**File Systems Fated for Senescence? Nonsense, Says Science!*, Conway et al, FAST 2017

Logical Copy with Bε-DAGs

Performance Goals

Copy-on-Abundant-Write

Space efficient ✓

GOTO messages

Low latency ✓

Copy-specific

Bε-trees have large nodes

Fast writes ✓

⇒ Locality*

Fast reads ✓

General file system

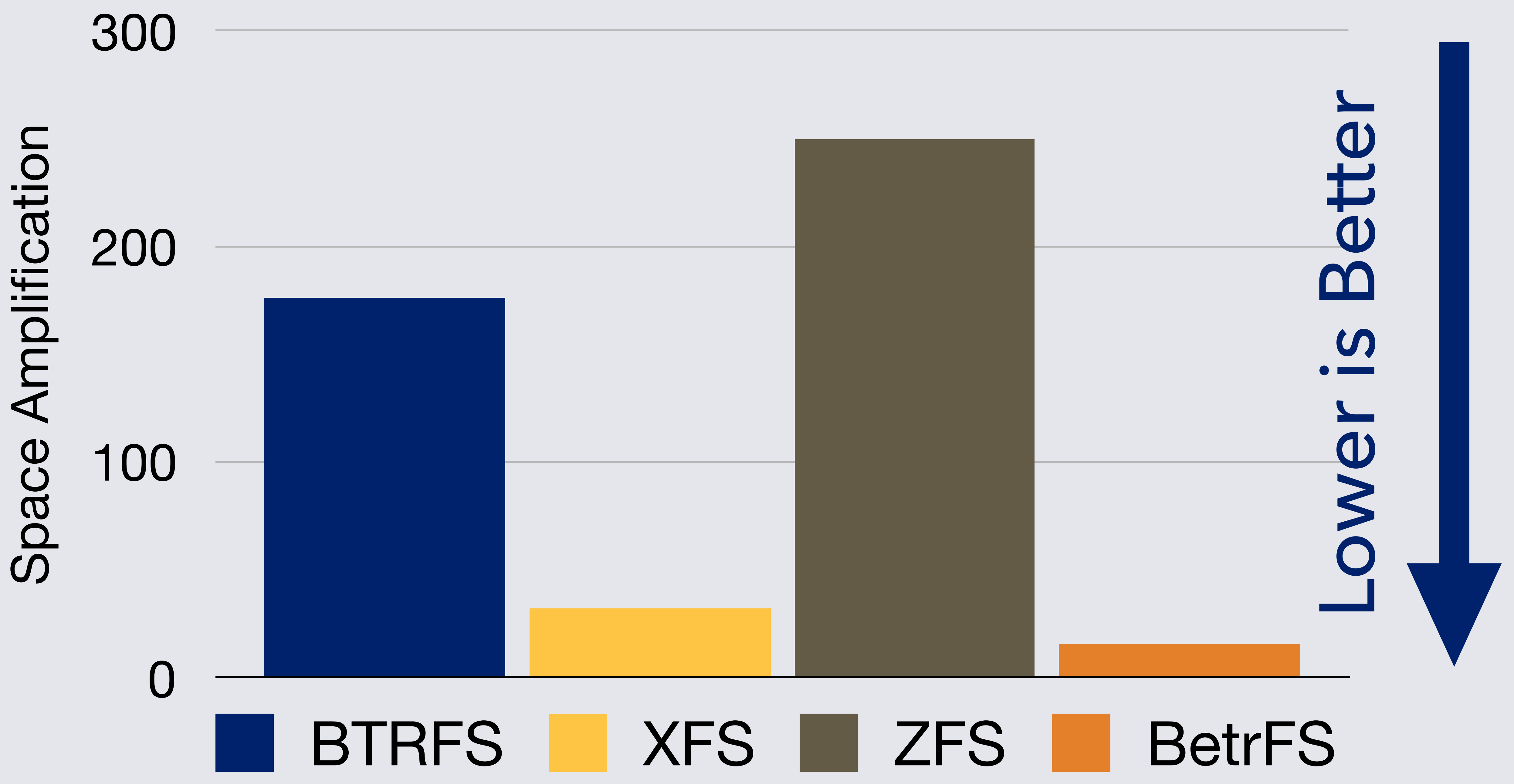
**File Systems Fated for Senescence? Nonsense, Says Science!*, Conway et al, FAST 2017

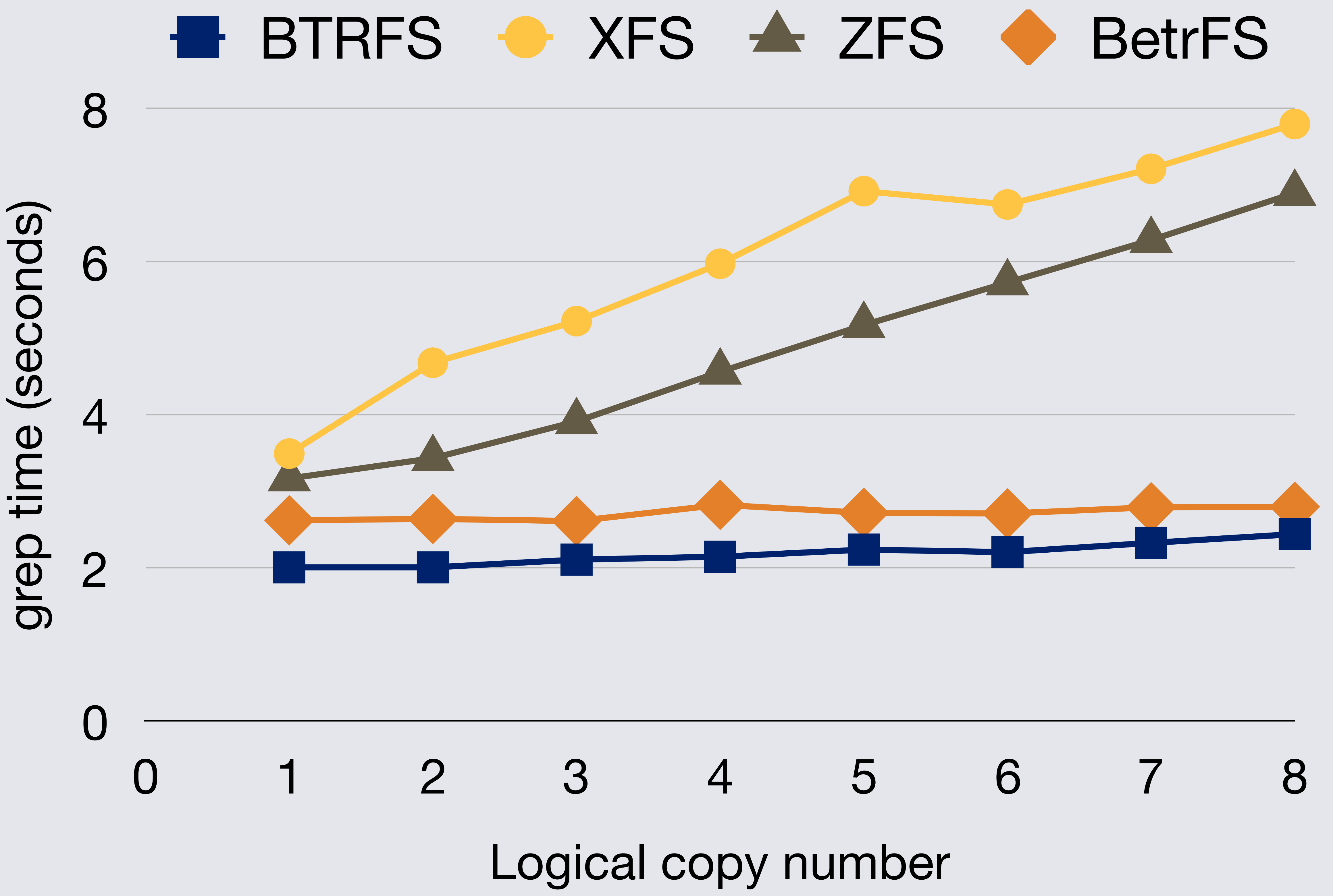
Evaluation

Space Amplification and Fragmention

Dell Optiplex 790
4-core 3.40 GHz Intel Core i7 CPU
4GiB RAM
500GB 7200 RPM SATA disk

Init: 64 4MiB files with random data.
Each round: logically copy all files, then change 16B in each file (1KiB total)





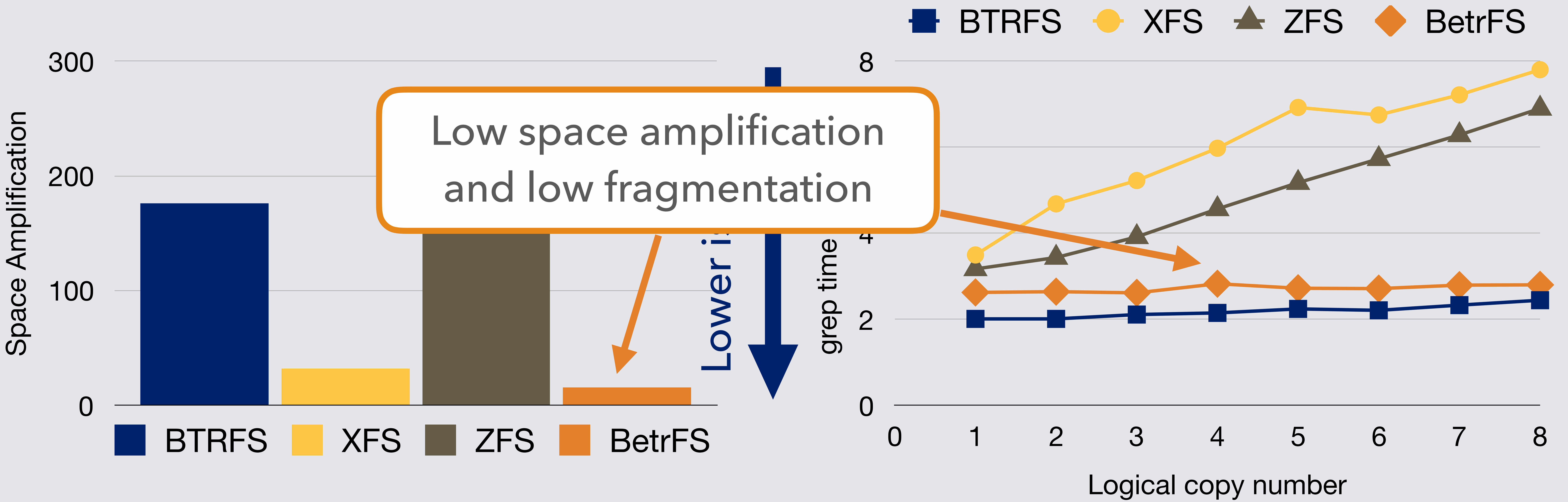
Space Amplification
additional file system size / added data (1KiB)

Fragmentation
measured by timing a grep over the file system

Space Amplification and Fragmention

Dell Optiplex 790
4-core 3.40 GHz Intel Core i7 CPU
4GiB RAM
500GB 7200 RPM SATA disk

Init: 64 4MiB files with random data.
Each round: logically copy all files, then change 16B in each file (1KiB total)



Space Amplification
additional file system size / added data (1KiB)

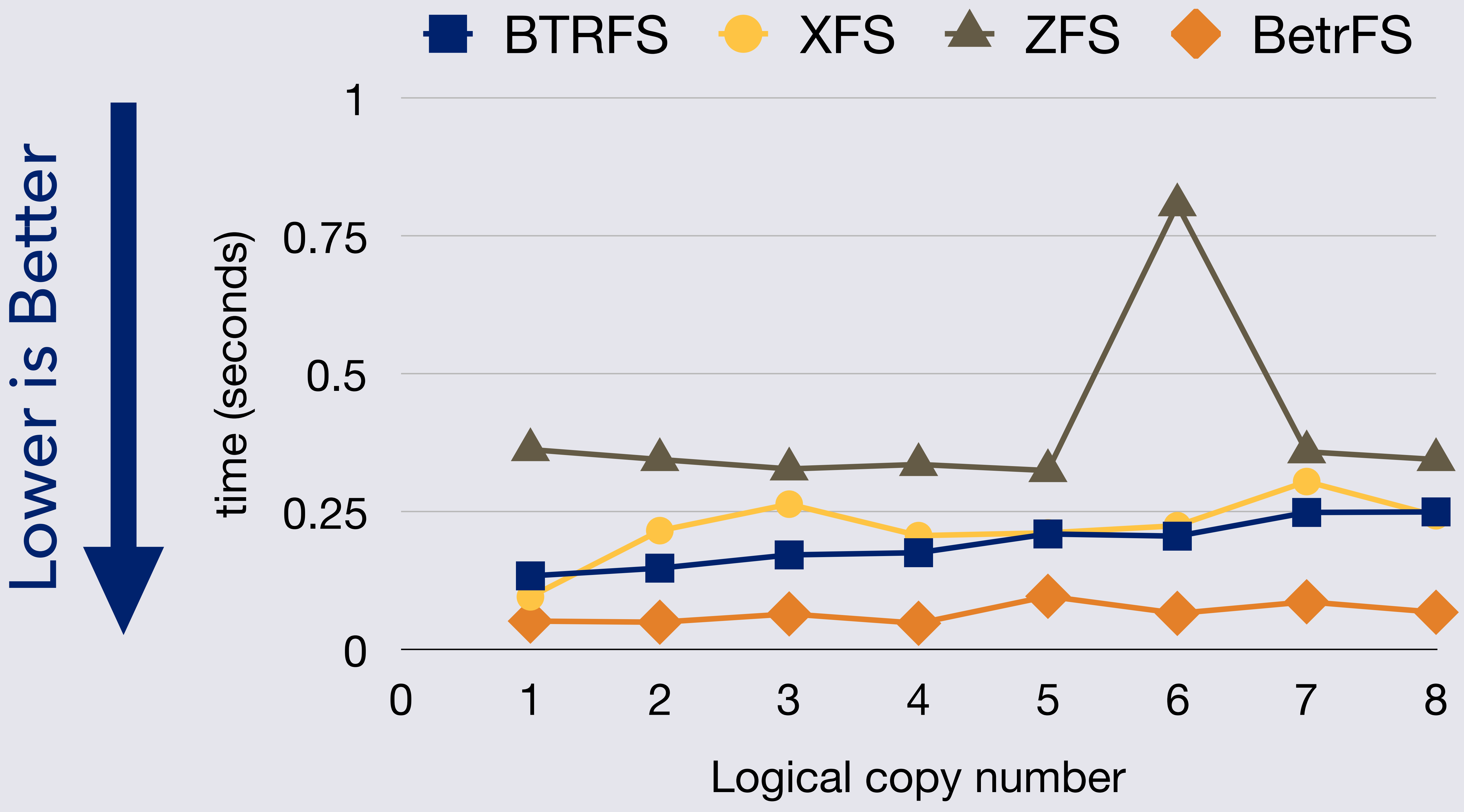
Fragmentation
measured by timing a grep over the file system

Copy Latency

Dell Optiplex 790
4-core 3.40 GHz Intel Core i7 CPU
4GiB RAM
500GB 7200 RPM SATA disk

Init: 64 4MiB files with random data.

Each round: logically copy all files, then change 16B in each file (1KiB total)



Copy Latency

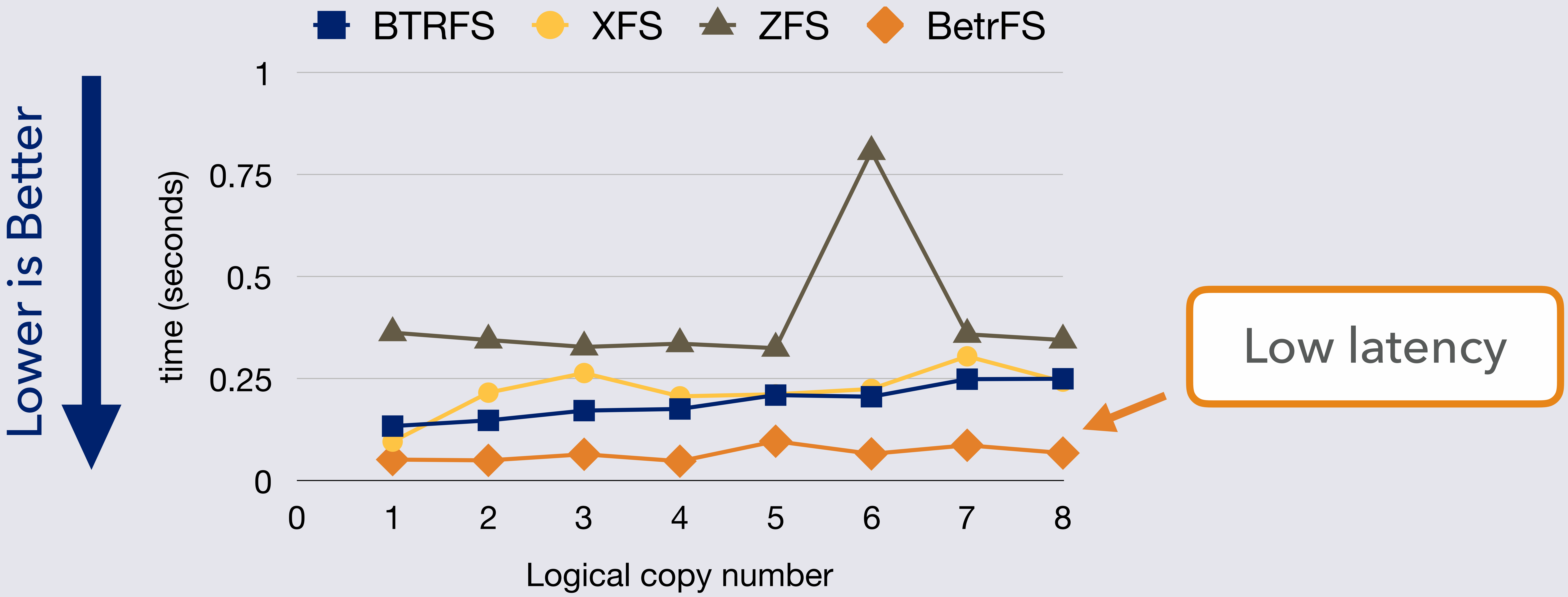
time to perform the copy

Copy Latency

Dell Optiplex 790
4-core 3.40 GHz Intel Core i7 CPU
4GiB RAM
500GB 7200 RPM SATA disk

Init: 64 4MiB files with random data.

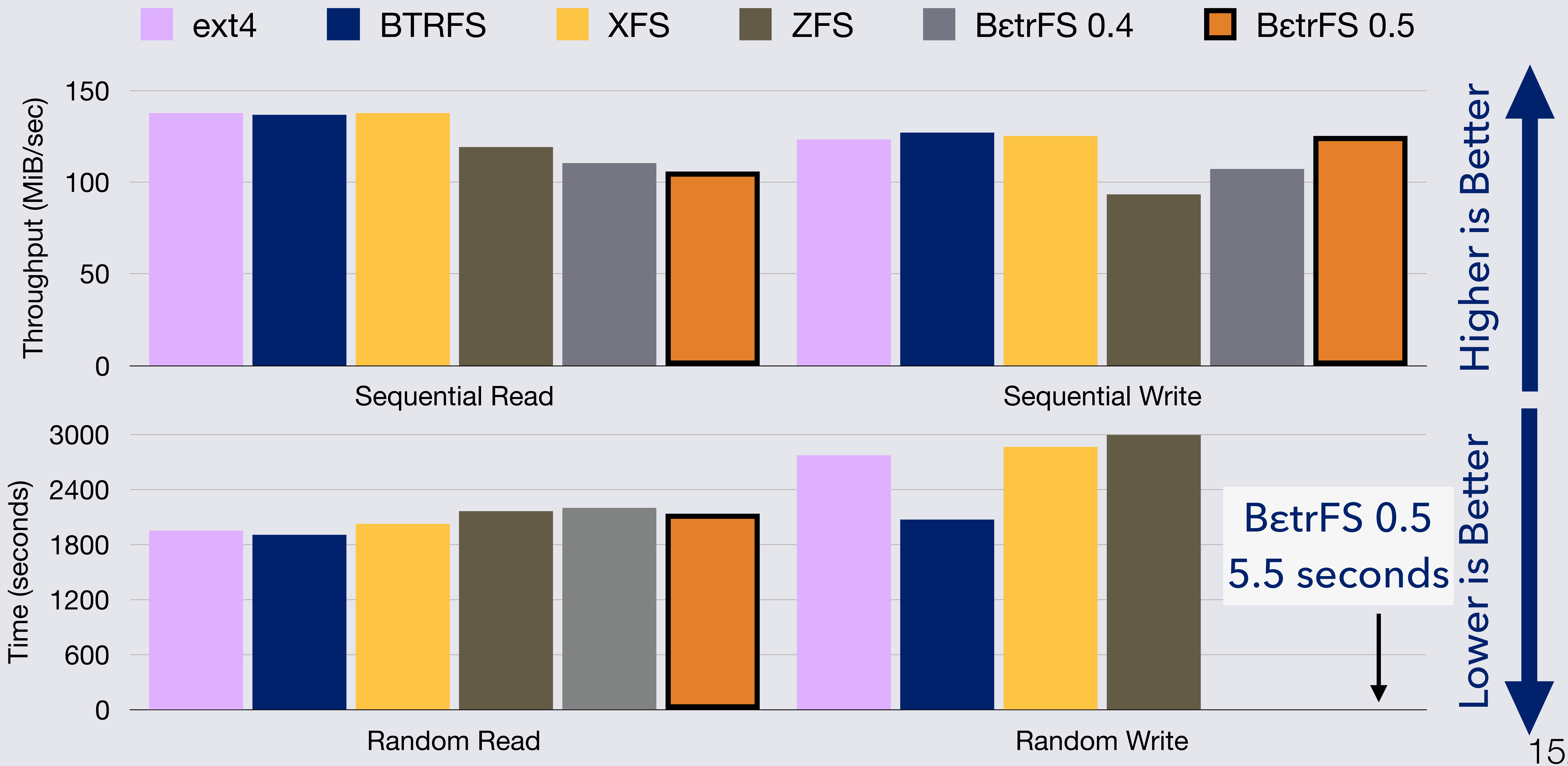
Each round: logically copy all files, then change 16B in each file (1KiB total)



Copy Latency

time to perform the copy

General File System Microbenchmarks



Application Benchmarks

Technical Conclusions

B$^{\varepsilon}$-DAGs transform copy-on-write into copy-on-abundant-write

- Gives strong bounds on space amplification
- Preserves locality, even with small writes
- Exploits B$^{\varepsilon}$-tree's batching and flushing

B$^{\varepsilon}$-DAGs preserve the fast reads and writes of B$^{\varepsilon}$-trees

- Preserve logarithmic tree height and query cost
- Preserve asymptotic costs of inserts and updates

Copies are fast and cheap

- GOTO messages enable low-latency DAG mutations
- Total work of copies is O(tree height)

Evaluation Conclusions

BetrFS with B$^\varepsilon$-DAGs has strong copy performance in practice

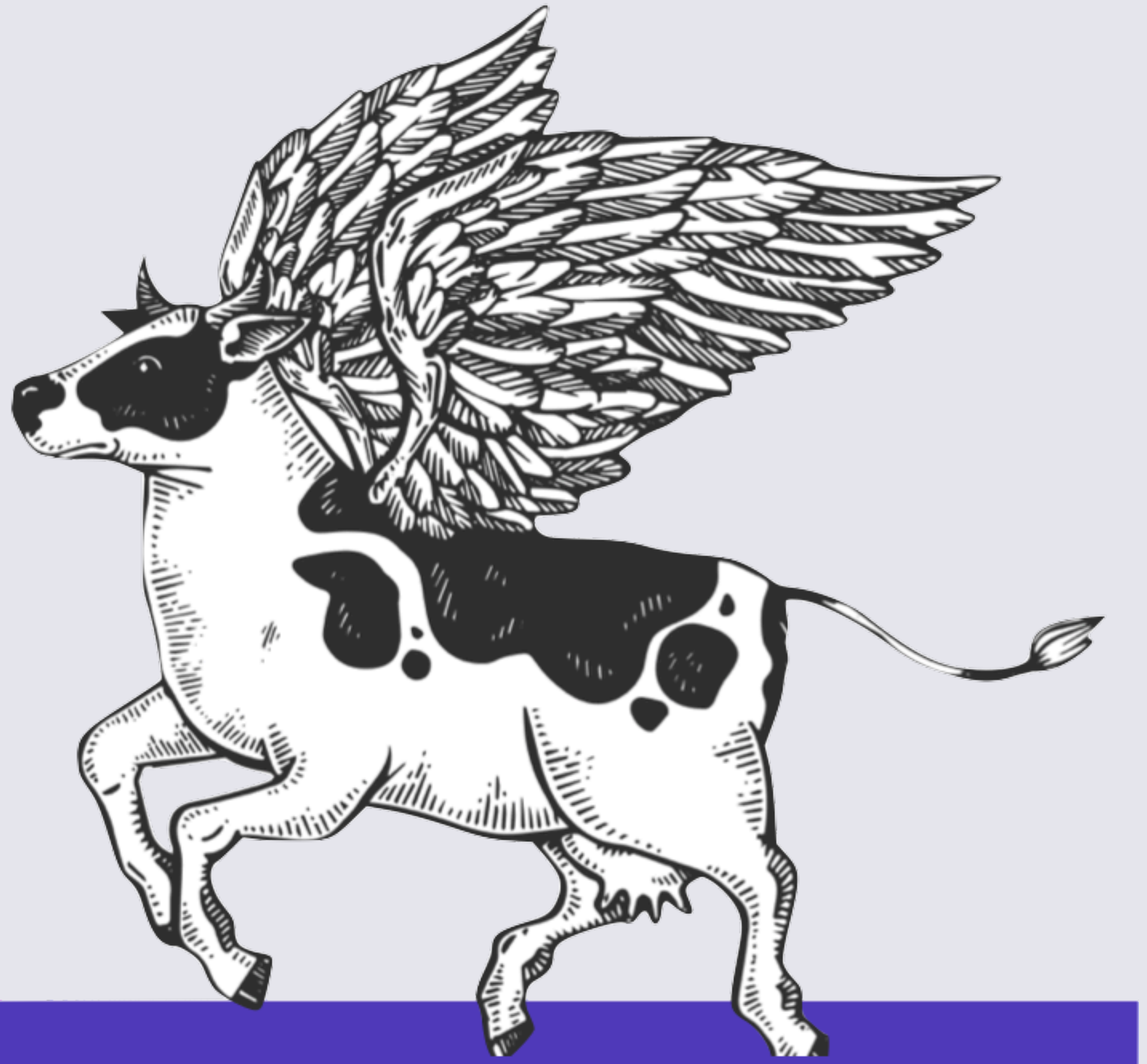
- Low space amplification
- Low latency copying
- Good locality

B$^\varepsilon$-DAGs preserve BetrFS's performance gains on other operations

- Fast random writes
- Good sequential I/O throughput
- No aging
- Strong across-the-board application benchmark performance

Thank you!

The CAW Awakens



BεtrFS Episode V: Attack of the Clones

O'Really? *betrfs.org*