

# Index Internals

Heikki Linnakangas / Pivotal

# Index Access Methods in PostgreSQL 9.5

- B-tree
- GiST
- GIN
- SP-GiST
- BRIN
  
- (Hash)

... but first, the **Heap**

# Heap

- Stores all tuples in table
- Unordered

Copenhagen  
Amsterdam  
Berlin  
Astana

Athens  
Baku  
Zagreb  
Andorra la Vella

Bern  
Helsinki  
Brussels  
Bucharest

Budapest  
Chişinău  
Ljubljana  
Dublin

Kiev  
Bratislava  
Lisbon  
Stockholm

# Heap

- Divided into 8k blocks

Blk 0

Copenhagen  
Amsterdam  
Berlin  
Astana

Blk 1

Athens  
Baku  
Zagreb  
Andorra la Vella

Blk 2

Bern  
Helsinki  
Brussels  
Bucharest

Blk 3

Budapest  
Chişinău  
Ljubljana  
Dublin

Blk 4

Kiev  
Bratislava  
Lisbon  
Stockholm

# TID: Physical location of heap tuple

Blk 0

0: Copenhagen  
1: Amsterdam  
2: Berlin  
3:  
4: Astana

Blk 1

0: Athens  
1:  
**2: Helsinki**  
3: Zagreb  
4: Andorra la Vella

*Example: Helsinki, Block 1, item 2 within block*

# Item pointer

*Example:* **Helsinki**, Block **1**, item **2** within block

(1, 2)

- Block number and position within page
- Uniquely identifies a tuple version

# Indexes in PostgreSQL

- Indexes store TIDs of heap tuples
  - except BRIN
- There is no visibility information in indexes
  - except for a simple “dead” flag, as an optimization
  - UPDATE inserts a new index tuple
  - Dead tuples are removed by VACUUM



B-tree

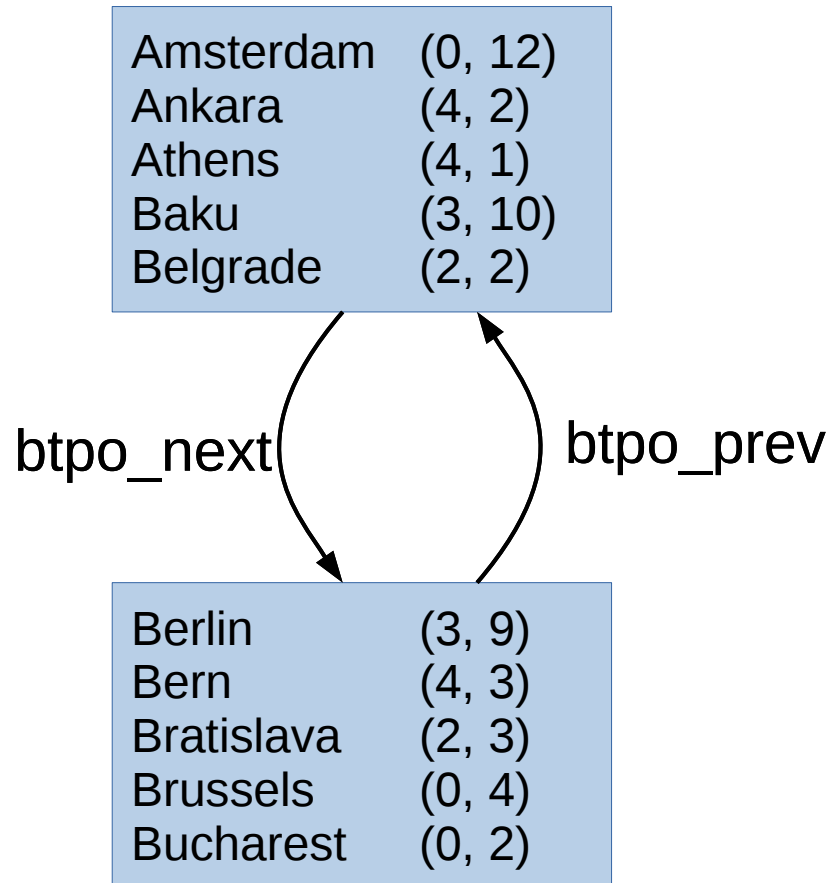
# Good old B-tree

- Default index type
- Tuples are stored on pages, ordered by key
- Tree, every branch has same depth

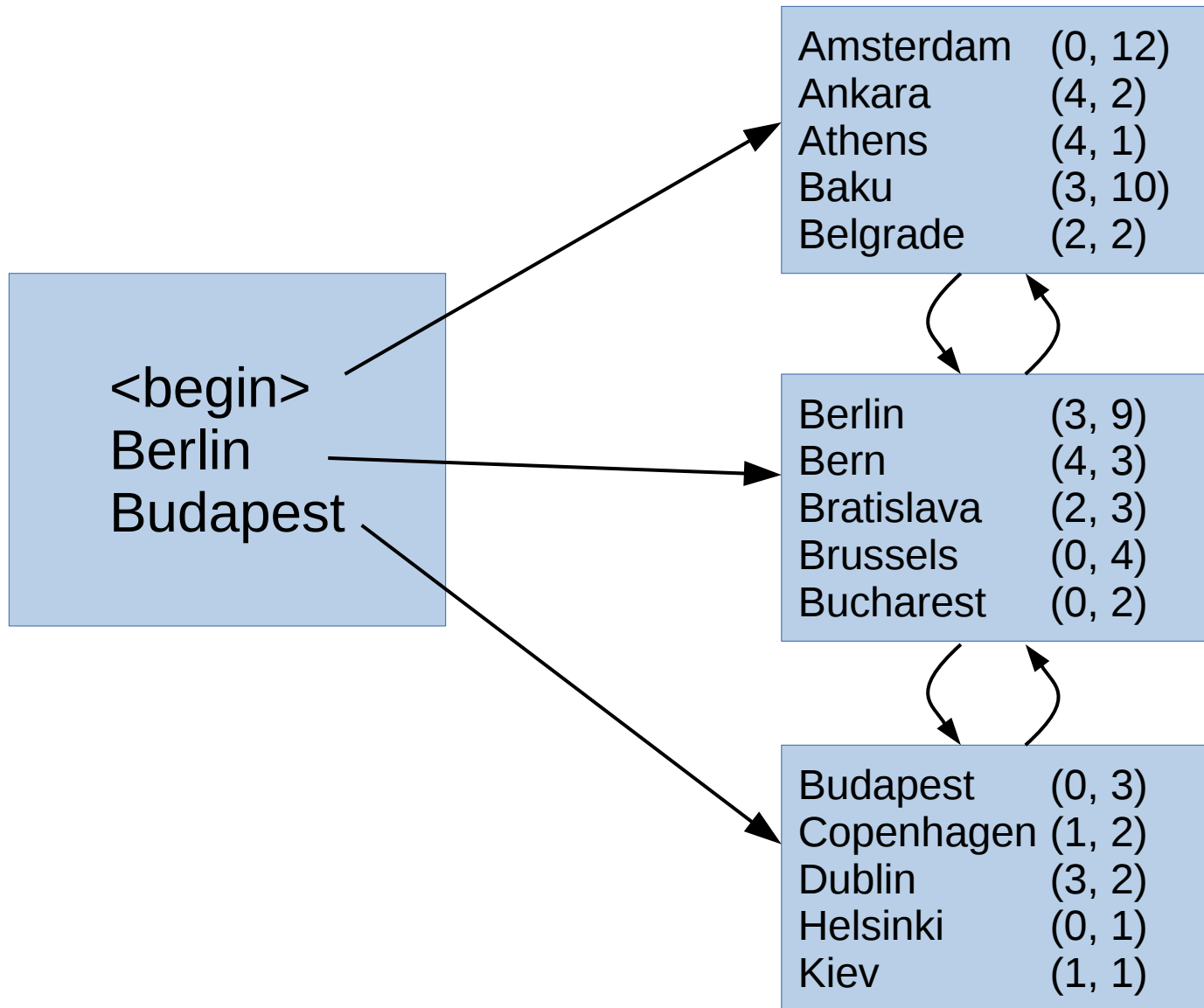
# B-tree, single page

Amsterdam	(0, 12)
Ankara	(4, 2)
Astana	(1, 9)
Athens	(4, 1)
Baku	(3, 10)
Belgrade	(2, 2)

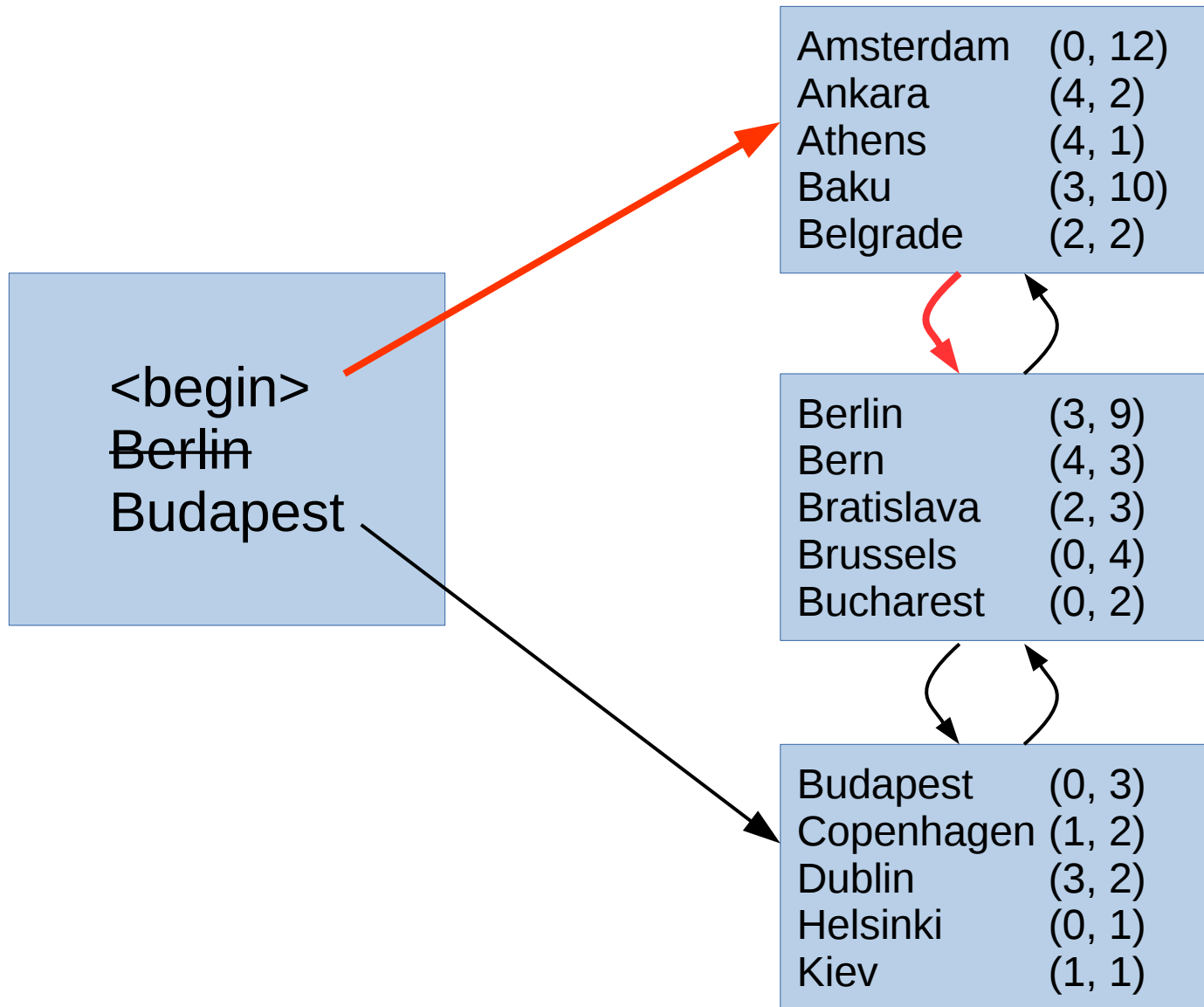
# B-tree, leaf level



# B-tree, two levels



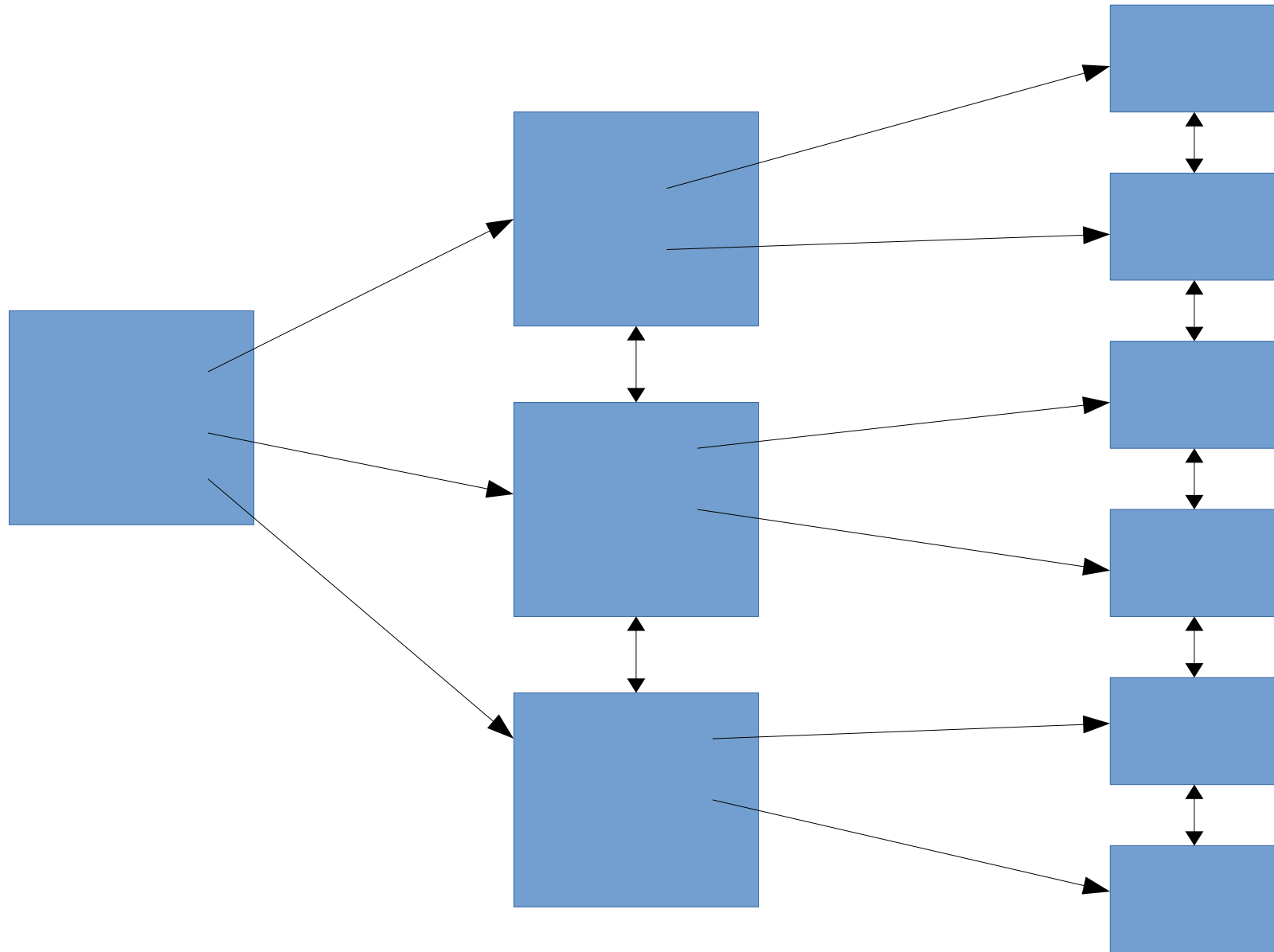
# B-tree that's missing nodes still works!



# B-tree details

- Lehman & Yao
- When a page becomes completely empty, it can be removed and recycled
- Half-empty pages are never merged
- Free Space Map to track unused pages

# B-tree, three levels

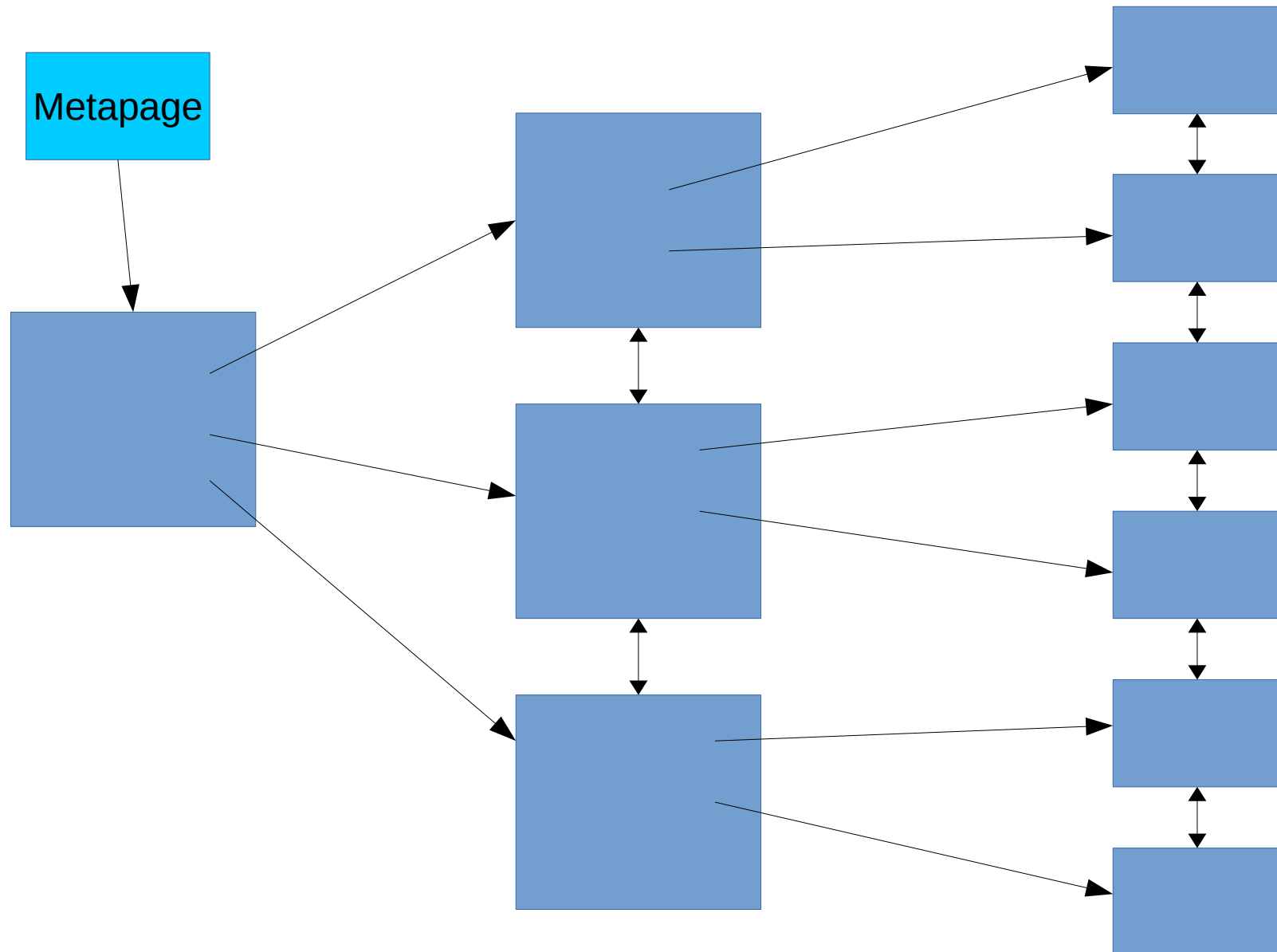




# Sidenote: Metapage

- Most index types in PostgreSQL has a metapage at block 0
  - All but GiST
- B-tree Metapage
  - Pointer to root page
  - Pointer to “fast root”

# Complete B-tree



# What can you do with a B-tree?

- Find key = X
- Find keys < X or > X
- ORDER BY
- LIKE 'foo%'

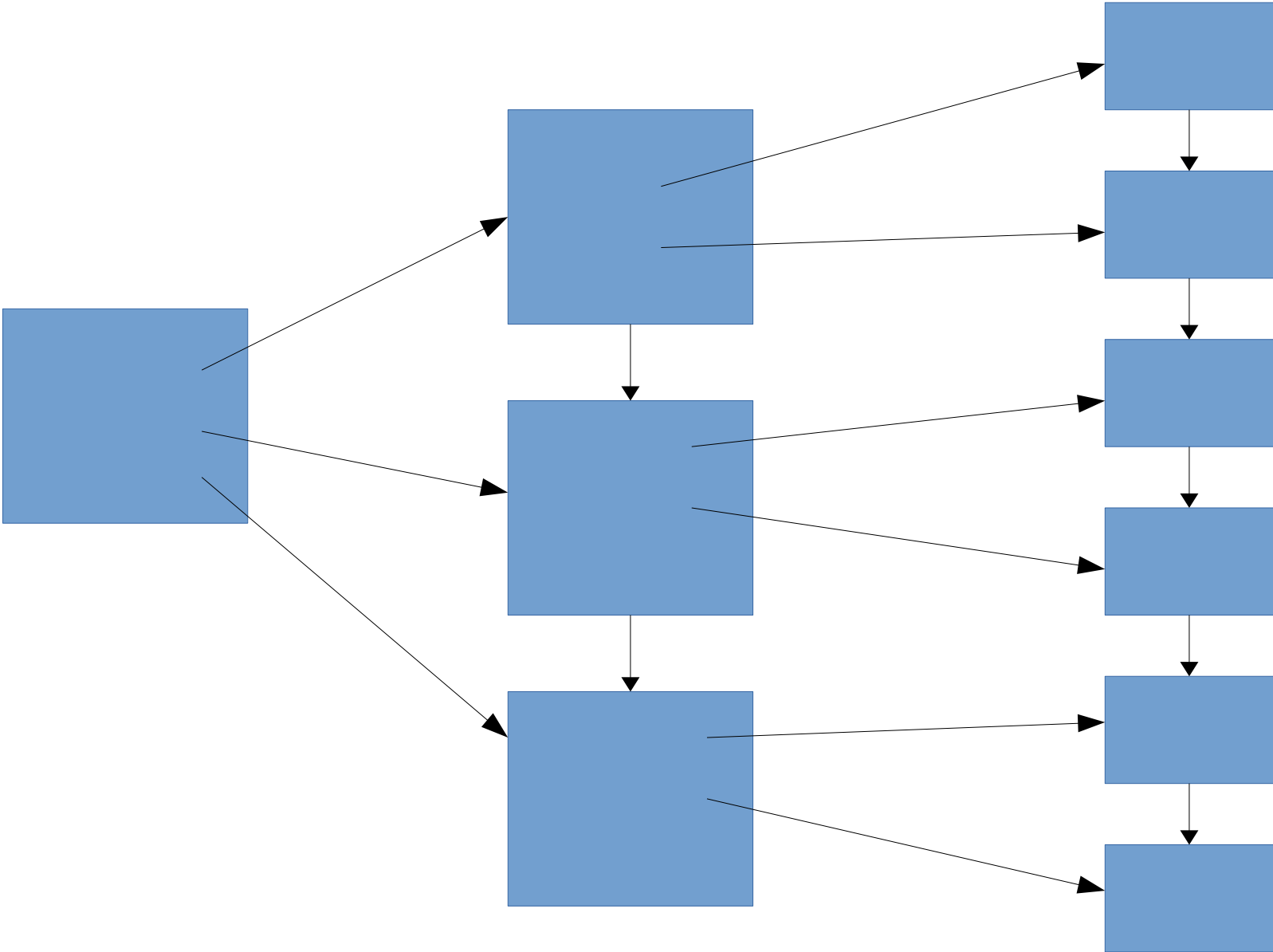
# GIN

= Generalized Inverted Index

# GIN

- Internal structure is basically just a B-tree
  - Optimized for storing a lot of duplicate keys
  - Duplicates are ordered by heap TID
- Interface supports indexing more than one key per indexed value
  - Full text search: “foo bar” → “foo”, “bar”
- Bitmap scans only

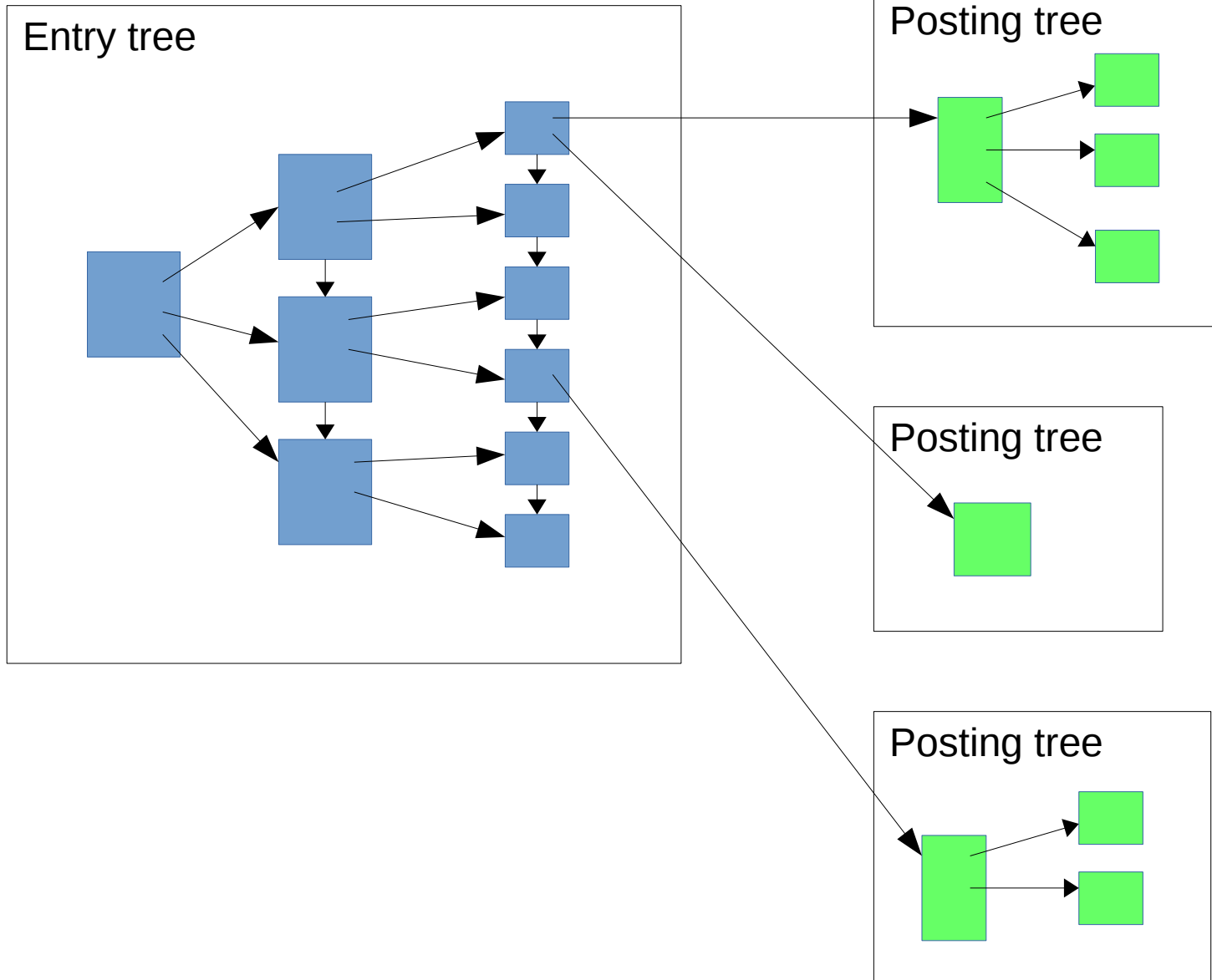
# GIN entry tree



# Three ways to store heap TIDs in entry item

- Single heap TID
  - trivial case, like normal B-tree
- Compressed list of heap TIDs
  - also known as a “posting list”
- Pointer (= blk #) to the root of posting tree
  - TIDs stored on a separate page or tree of pages, in TID order

# GIN

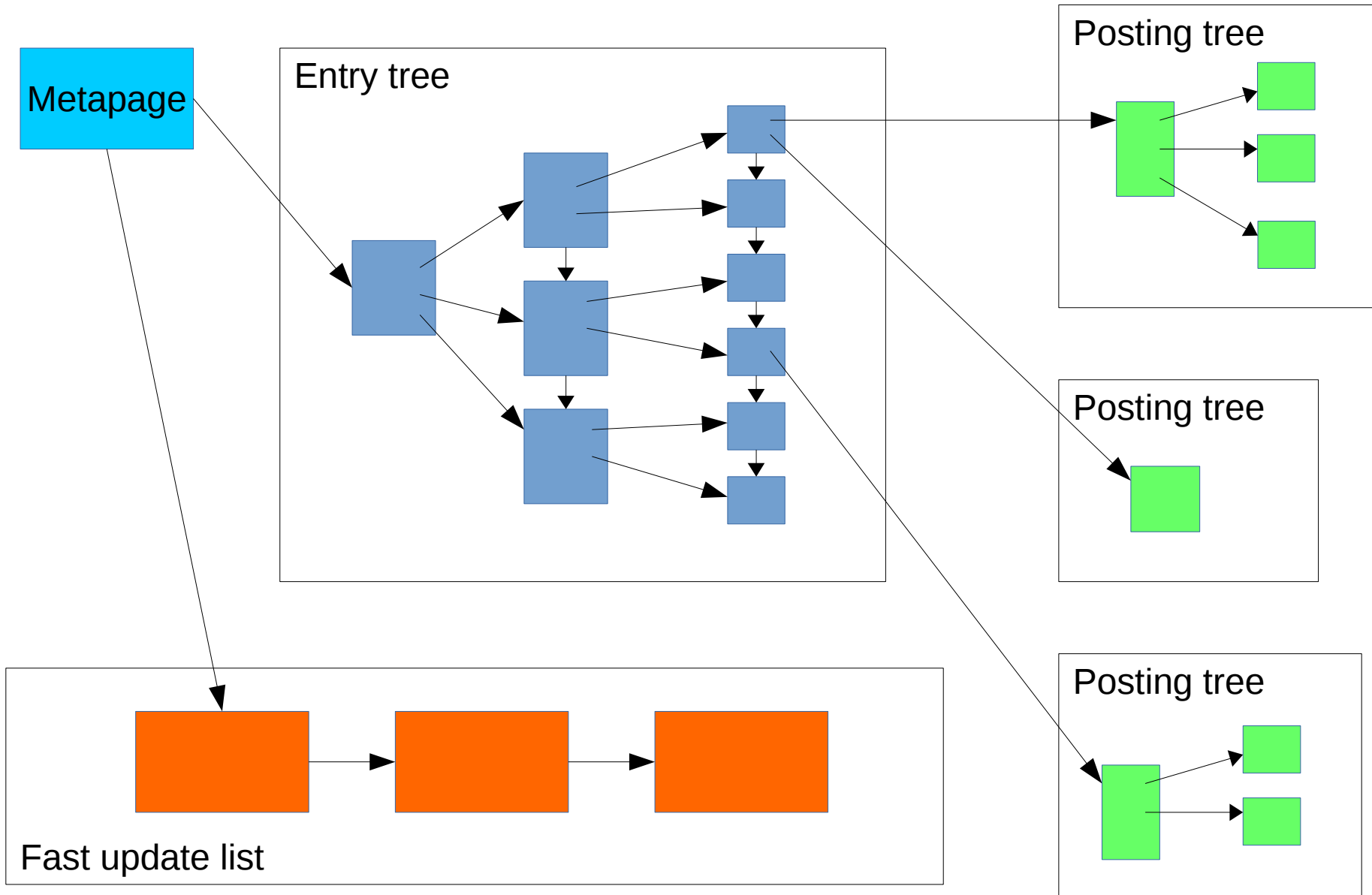




# GIN “fast updates”

- Insertions to GIN index go to a list of “fast updated” tuple.
  - Every search scans the list in addition to index
- Moved to index proper by VACUUM
  - Or by inserts, if grows too big
- Can be disabled with `FASTUPDATE = off` option

# Complete GIN



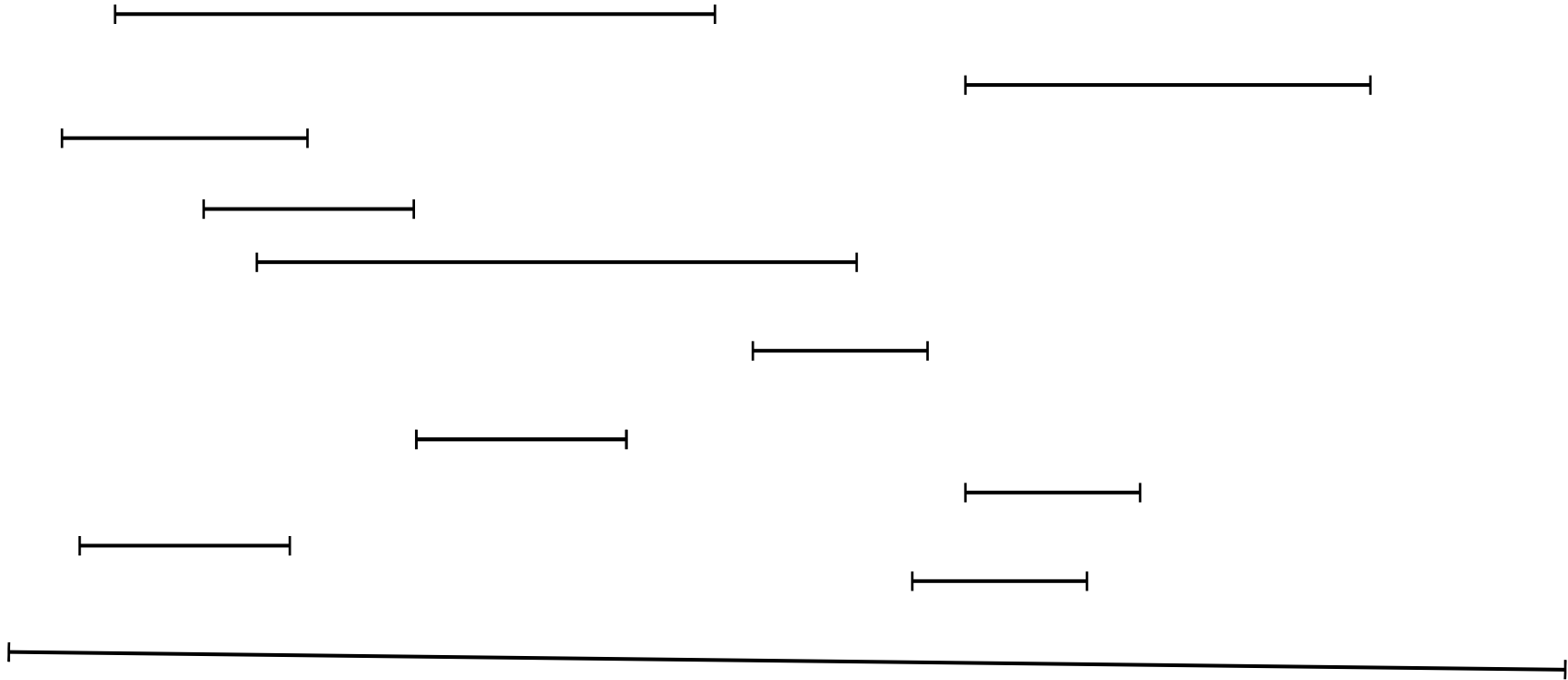
# GiST

= Generalized Search Tree

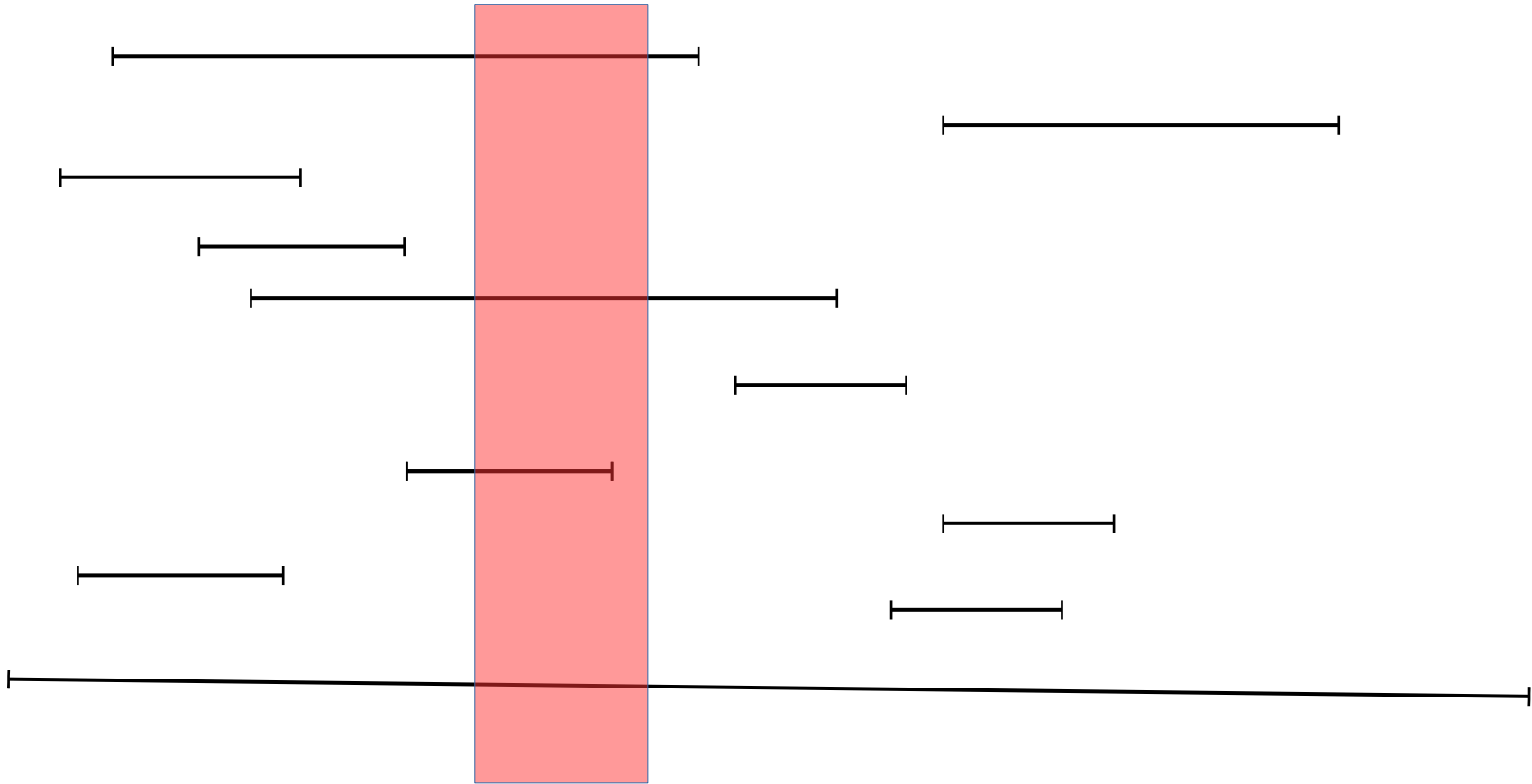
# GiST

- Tree-structure
- No order within pages
- Key ranges of pages can overlap
  - No single “correct” location for a particular tuple

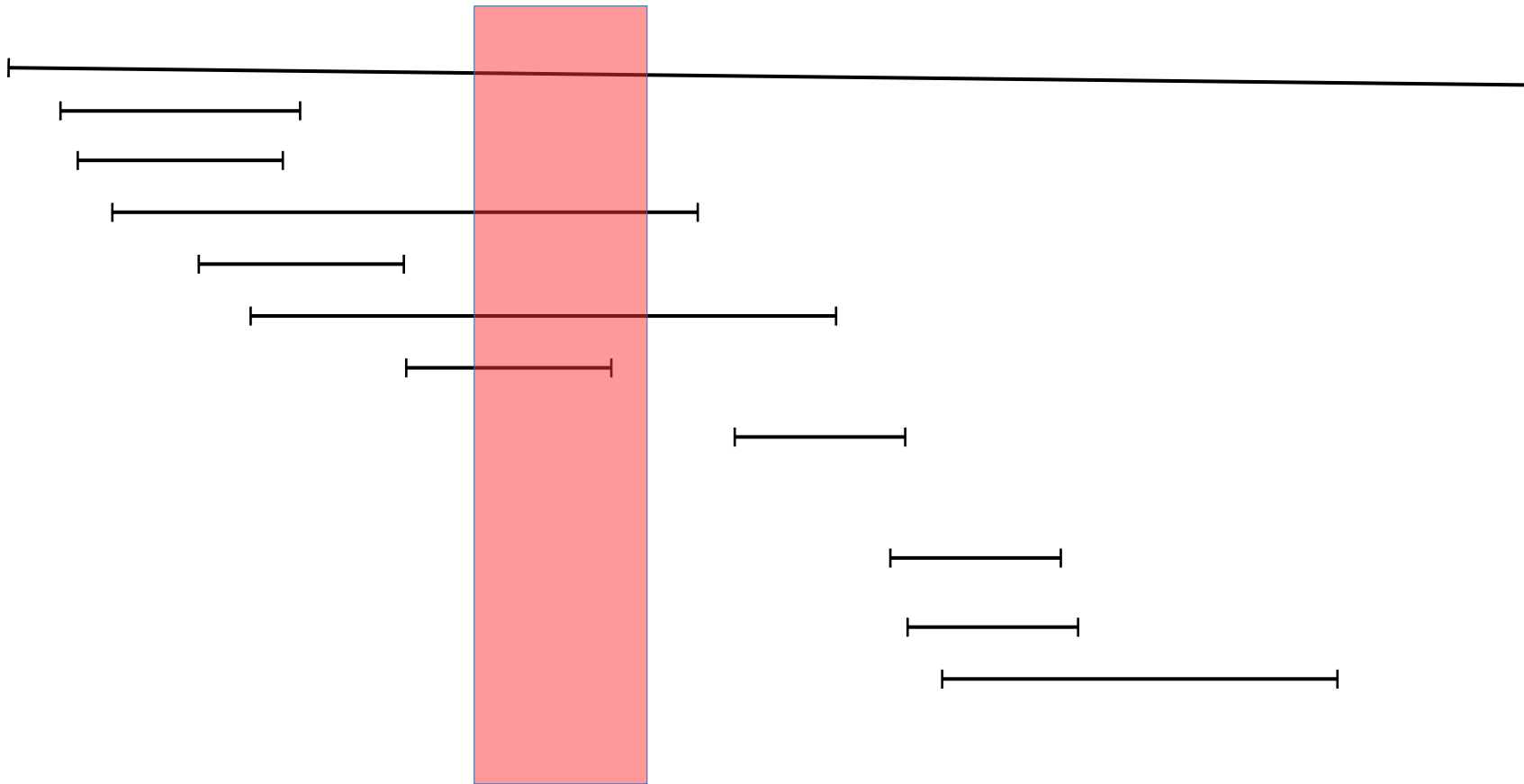
# Range types



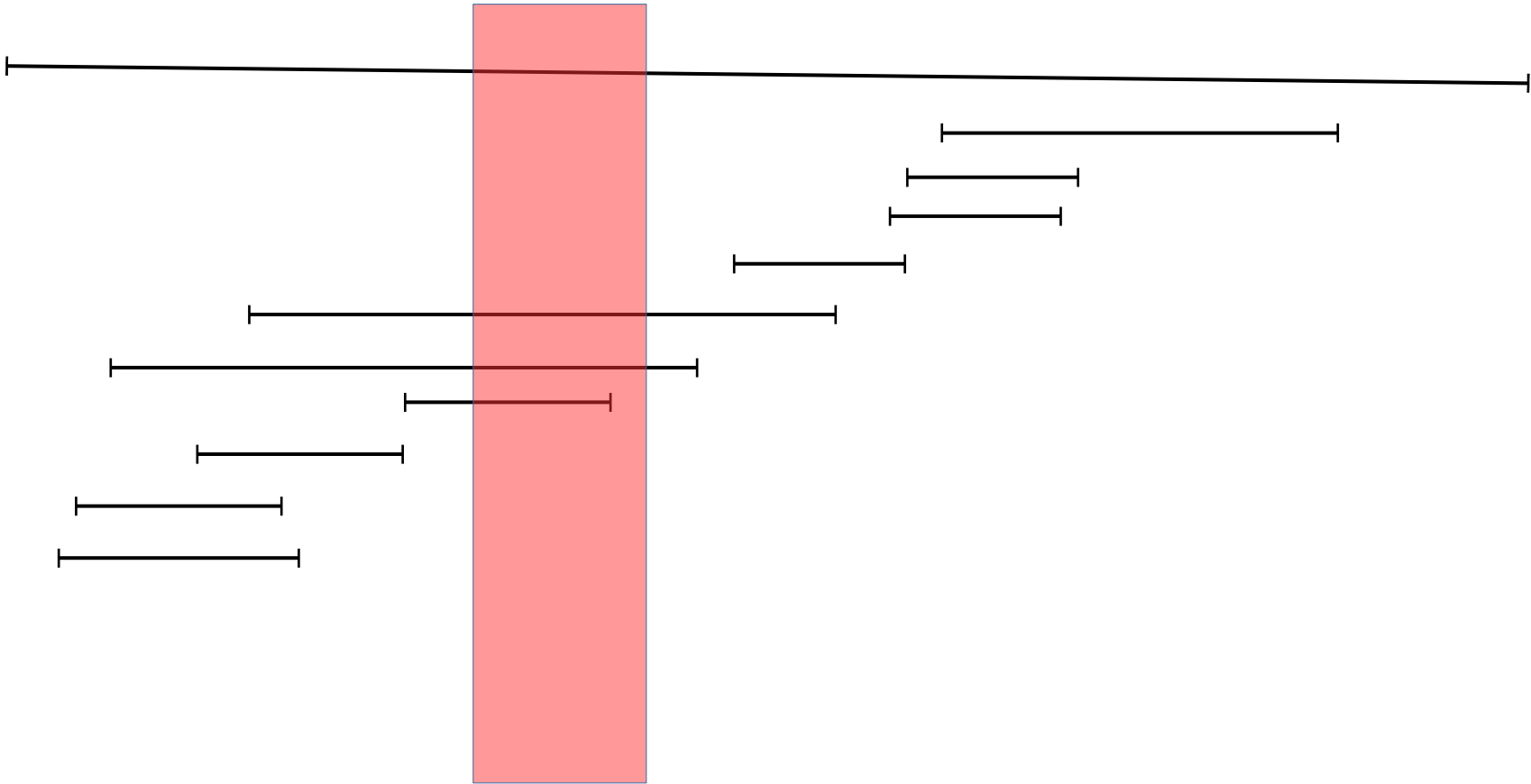
# Find ranges that overlap



# Sort by min

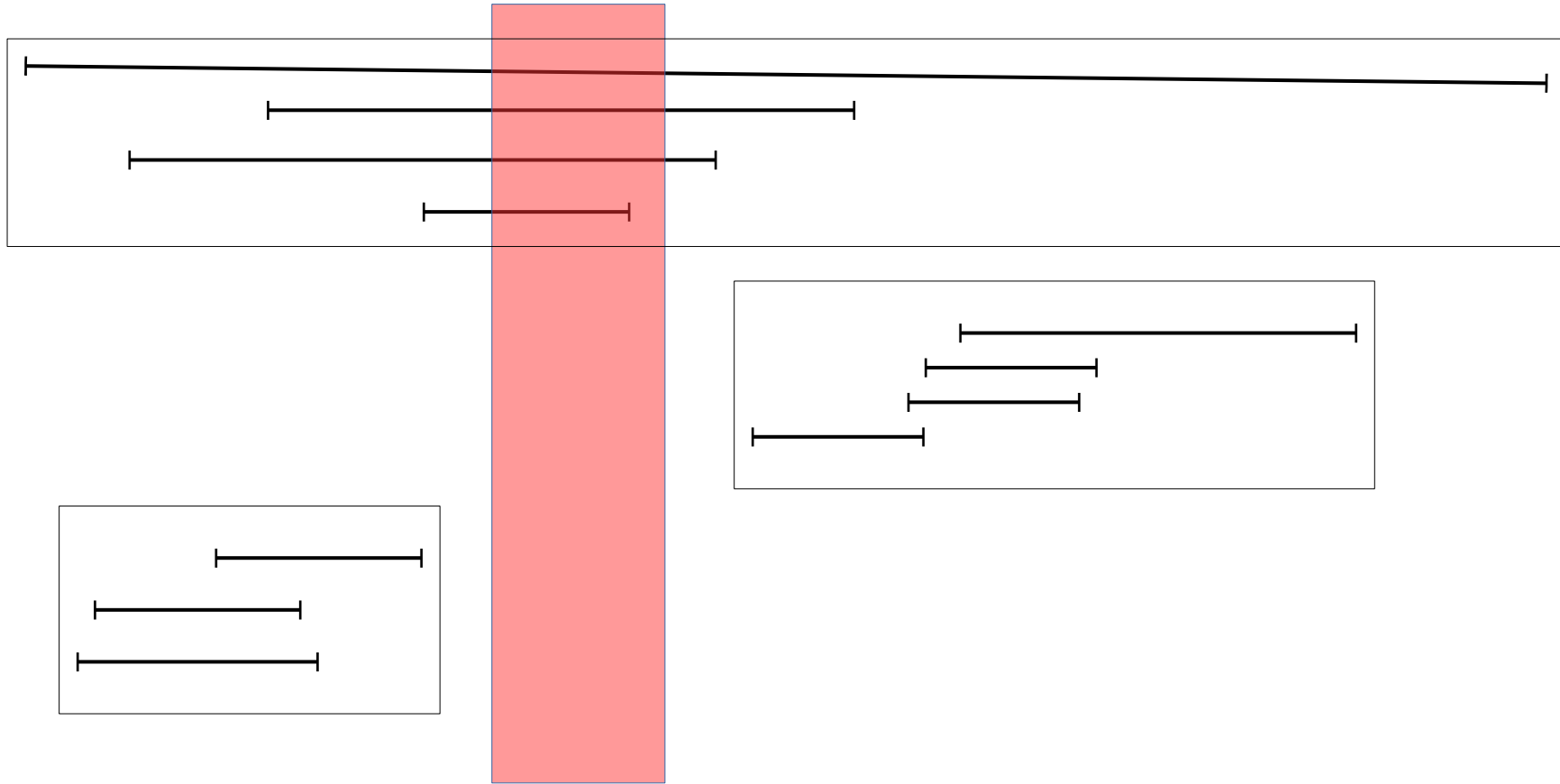


# Sort by max





# Group into clusters

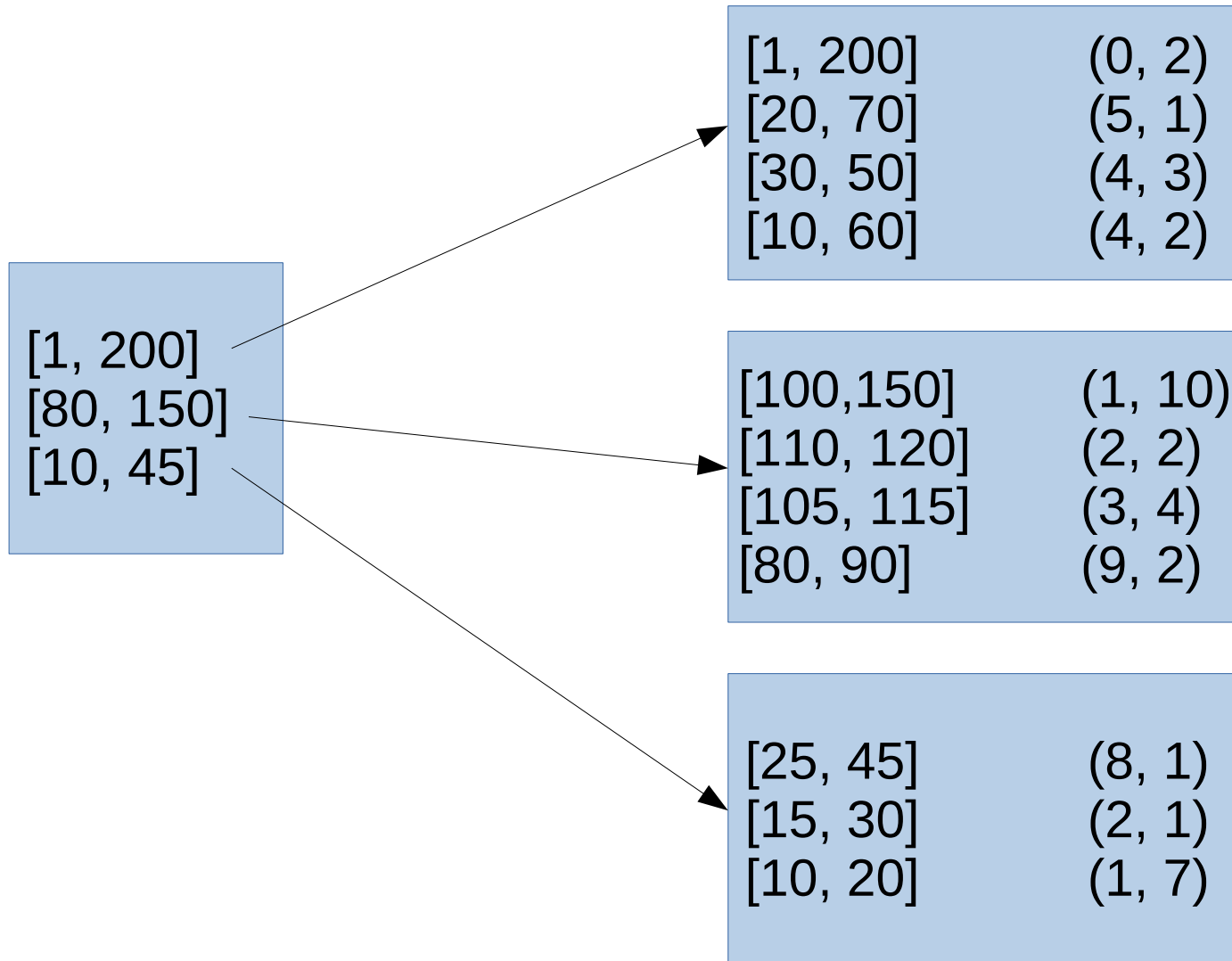


# GIST, single page

- Stores key + TID
- One index tuple per heap tuple
- Unordered

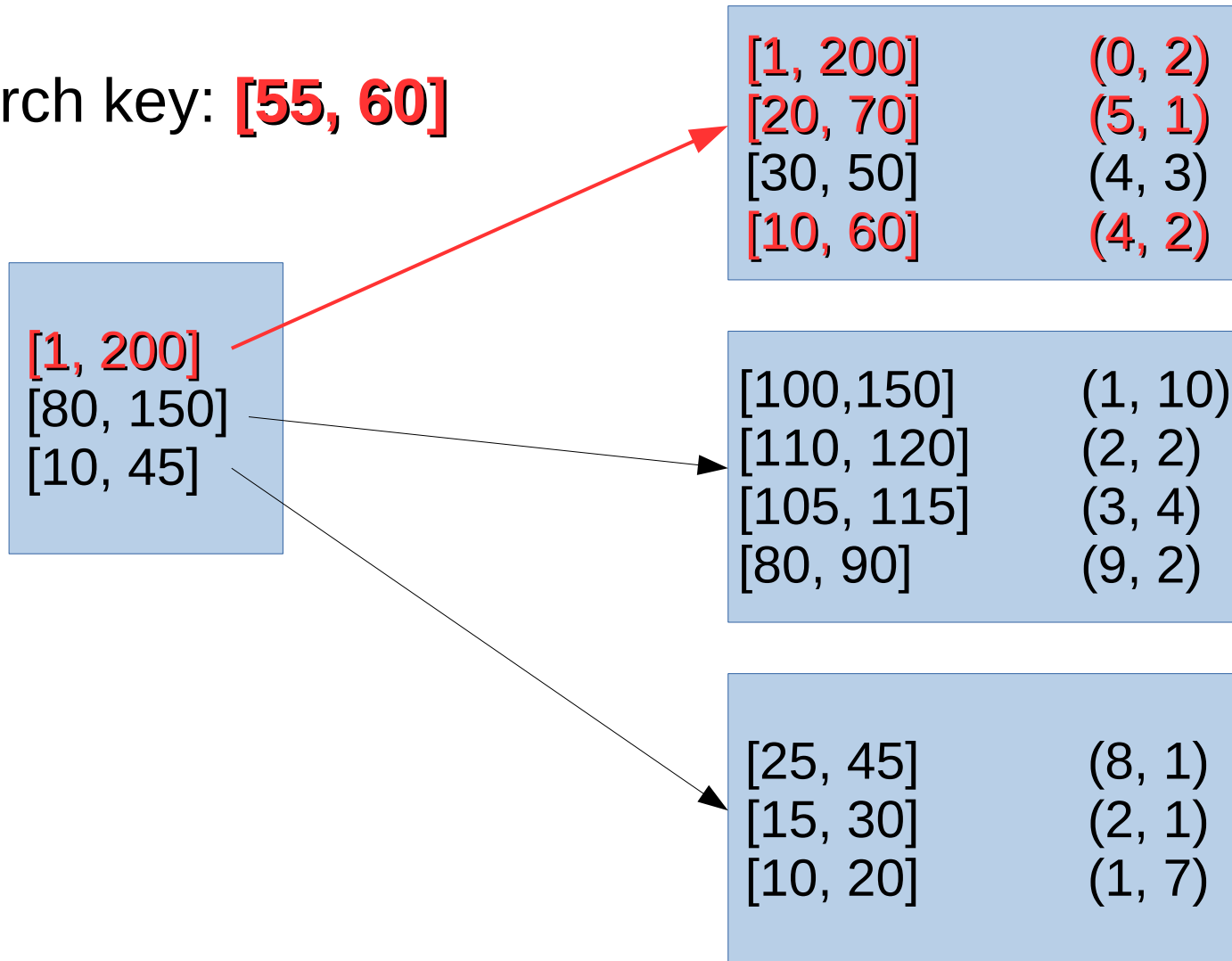
[100,150]	(1, 10)
[1, 200]	(0, 2)
[10, 60]	(4, 2)
[30, 50]	(4, 3)
[20, 70]	(5, 1)
[110, 120]	(2, 2)
[15, 30]	(2, 1)
[105, 115]	(3, 4)
[80, 90]	(9, 2)
[25, 45]	(8, 1)
[10, 20]	(1, 7)

# GIST, two levels



# GIST search

Search key: **[55, 60]**



# GiST

- Loose ordering
- Any key can legitimately be stored anywhere in the tree
  - As long as the keys in the upper levels are updated accordingly.
  - Performance goes out the window if you do that.
- Performance depends on how well the user-defined Picksplit and Choose functions can group keys

# What can you do with GiST?

- GIS stuff
- Find points within a bounding box
- Nearest Neighbor

# GiST, not only for geometries

- Contrib/intarray
- Full-text search
- Upper node “contains” everything below it
  - For points, a bounding box of all points below it
  - For intarray, the OR of all the nodes below it

**SP-GiST**  
= Space-Partitioned GiST

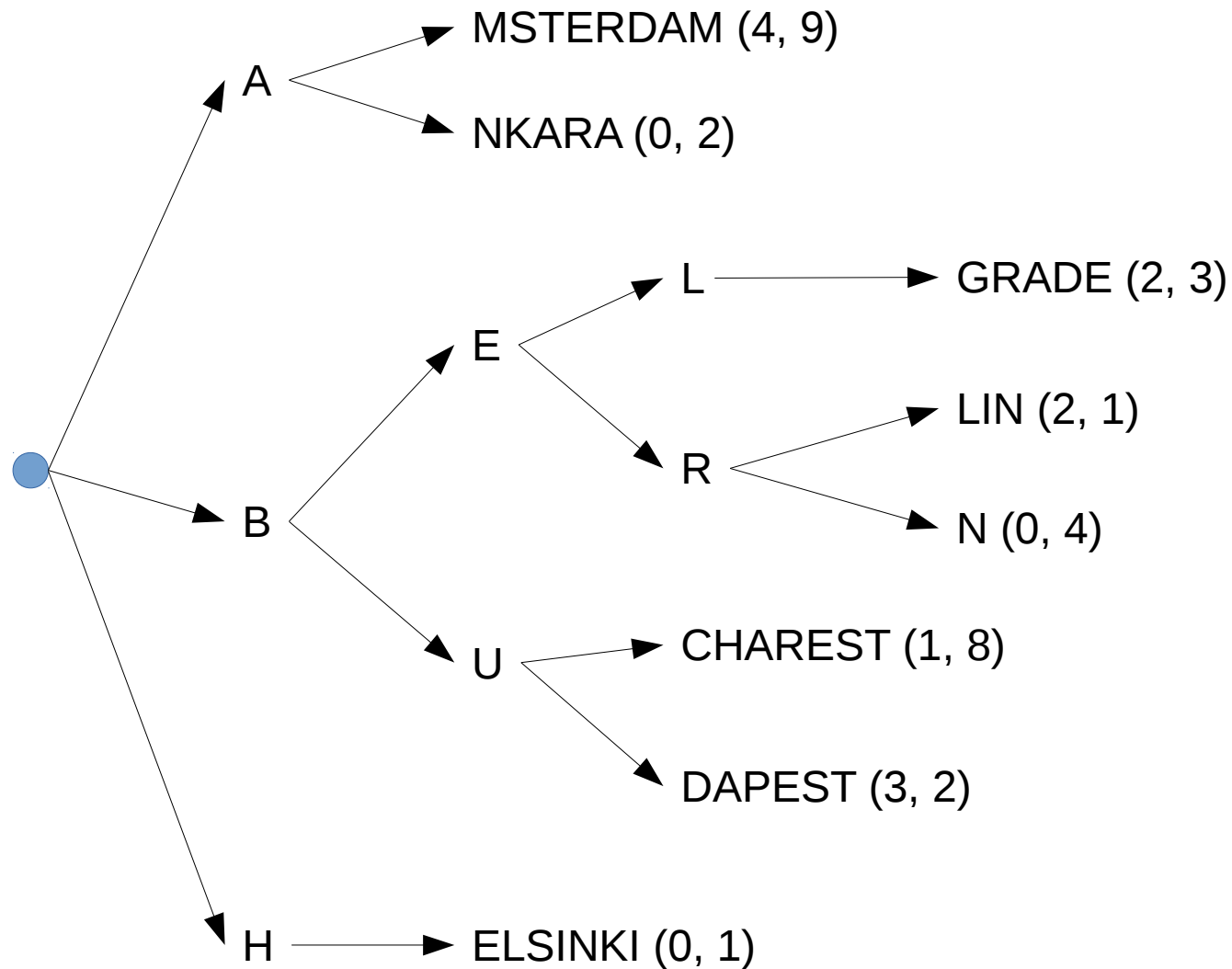


# SP-GiST

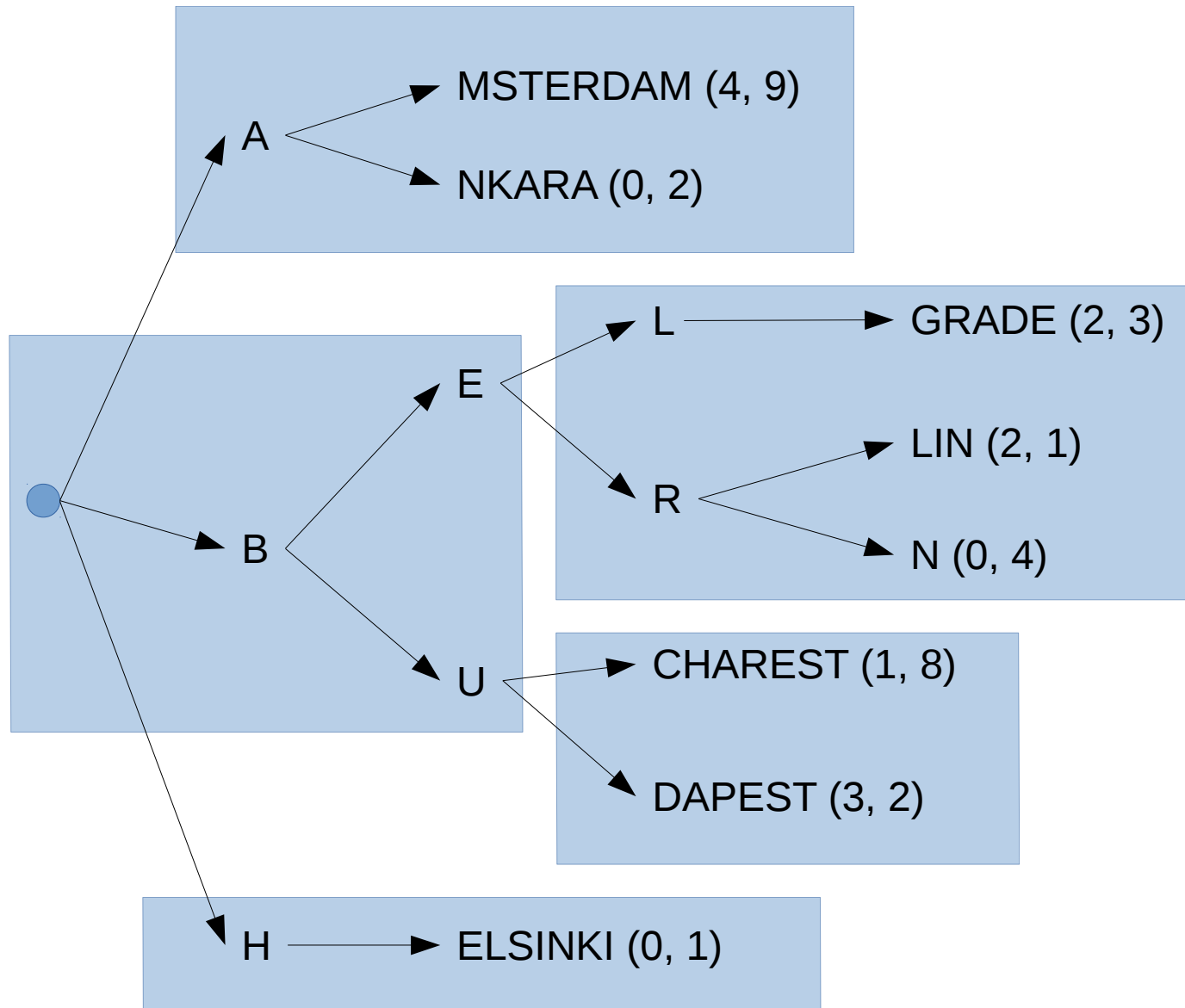
## *Space-Partitioned* GiST

- No overlap between nodes
- Quite different from GiST
- Variable depth
- Multiple nodes per physical page

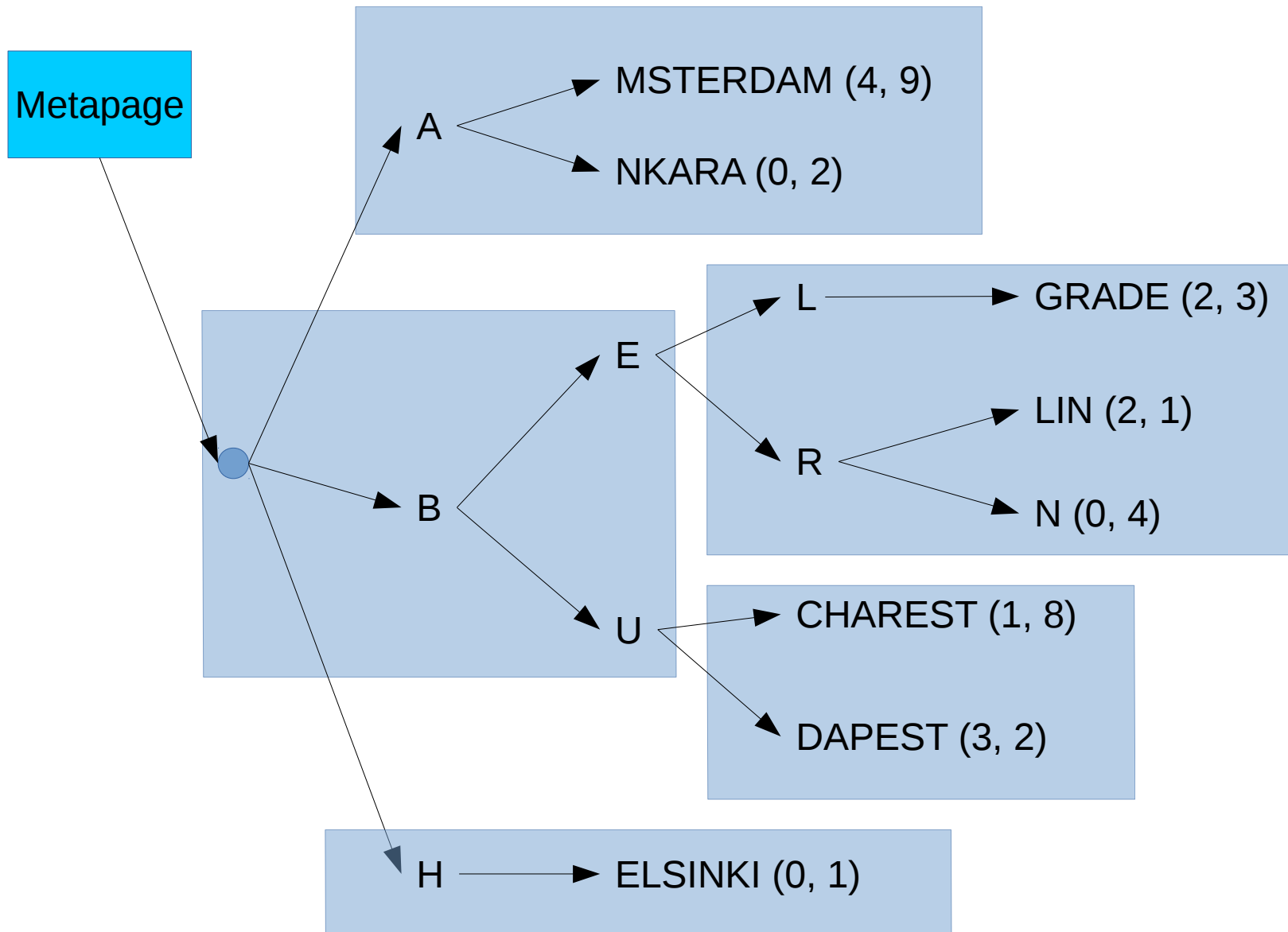
# SP-GiST example: Trie



# SP-GiST page layout



# SP-GiST page layout



# What can you do with it

- Kd-tree
  - Points only; shapes might overlap
- Prefix tree for text

# BRIN

= Block Range Index

# BRIN

- Not a tree
- Contains one entry per heap block (or range of heap blocks)
- **Very** compact
- Summary information for each block range

# Approximation #1

## BRIN Index

- 0: Amsterdam – Astana
- 1: Athens – Berlin
- 2: Bern – Bucharest
- 3: Budapest – Dublin
- 4: Helsinki – Ljubljana

## Heap

Amsterdam  
Andorra la Vella  
Ankara  
Astana

Athens  
Baku  
Belgrade  
Berlin

Bern  
Bratislava  
Brussels  
Bucharest

Budapest  
Chişinău  
Copenhagen  
Dublin

Helsinki  
Kiev  
Lisbon  
Ljubljana

...



# Approximation #2

## BRIN Index

3: Budapest – Dublin  
0: Amsterdam – Astana  
2: Bern – Bucharest

4: Helsinki – Ljubljana  
1: Athens – Berlin

## Heap

Amsterdam  
Andorra la Vella  
Ankara  
Astana

Athens  
Baku  
Belgrade  
Berlin

Bern  
Bratislava  
Brussels  
Bucharest

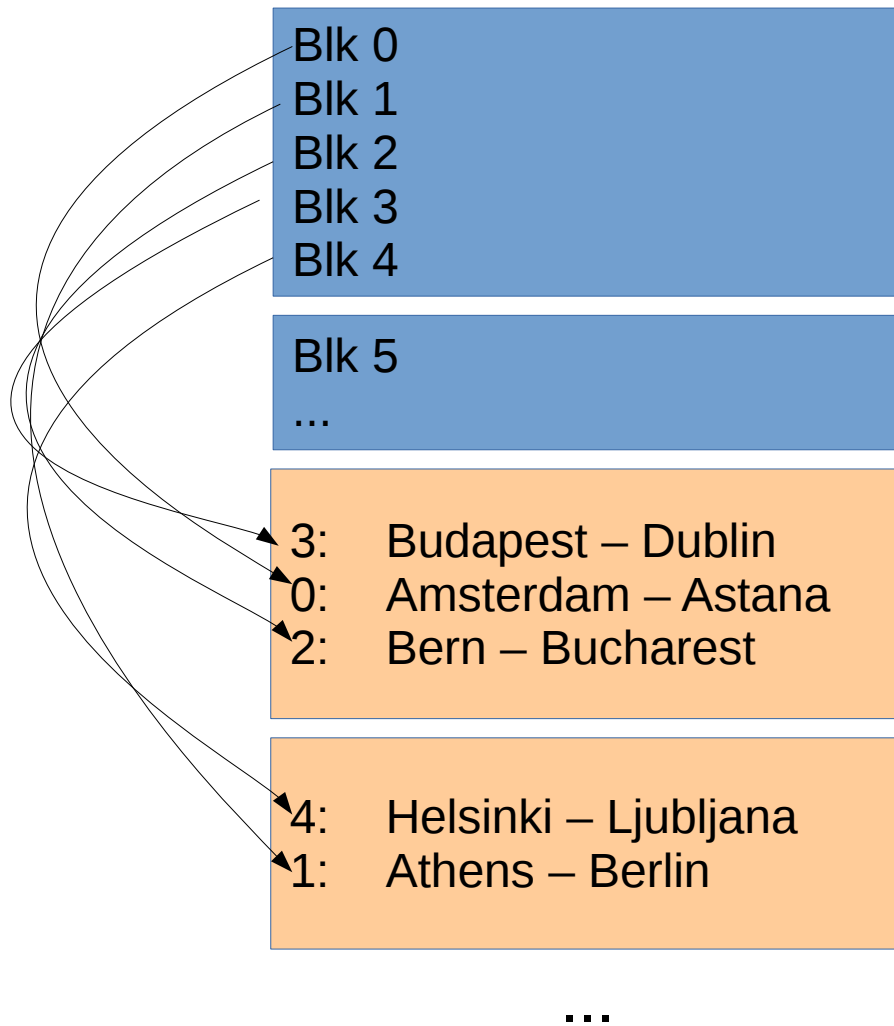
Budapest  
Chişinău  
Copenhagen  
Dublin

Helsinki  
Kiev  
Lisbon  
Ljubljana

...

# Approximation #3

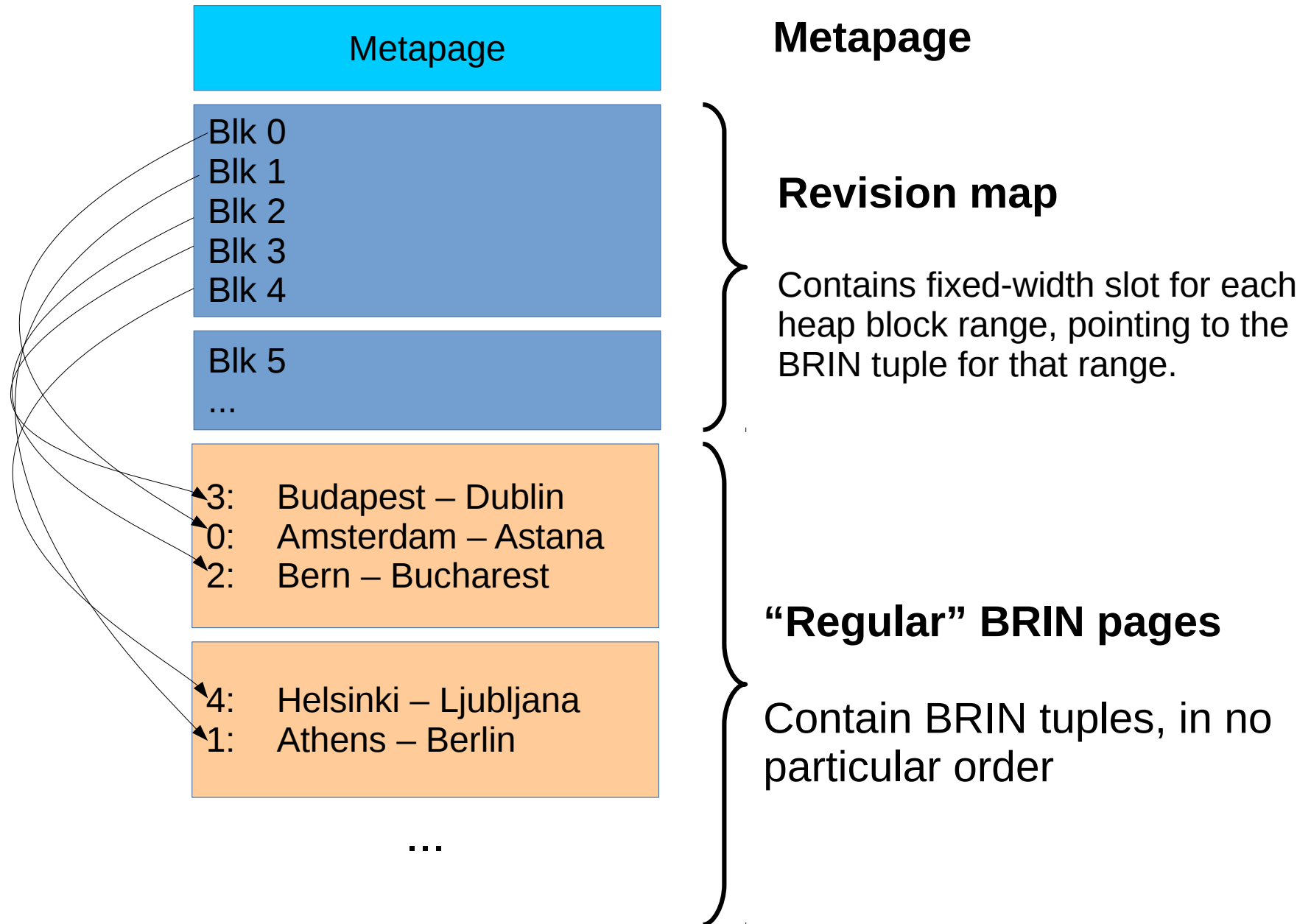
## BRIN Index



## Heap



# Complete BRIN



# BRIN: clustering is important!

- 0: Amsterdam – Astana
- 1: Athens – Berlin
- 2: Bern – Bucharest
- 3: Budapest – Dublin
- 4: Helsinki – Ljubljana

Amsterdam  
Andorra la Vella  
Ankara  
Astana

Athens  
Baku  
Belgrade  
Berlin

Bern  
Bratislava  
Brussels  
Bucharest

Budapest  
Chişinău  
Copenhagen  
Dublin

Helsinki  
Kiev  
Lisbon  
Ljubljana

# BRIN: clustering is important!

UPDATE cities SET name='Zagreb' WHERE ...

- 0: Amsterdam – Zagreb
- 1: Athens – Zagreb
- 2: Bern – Zagreb
- 3: Budapest – Zagreb
- 4: Helsinki – Zagreb

Amsterdam  
Zagreb  
Ankara  
Astana

Athens  
Baku  
Zagreb  
Berlin

Bern  
Zagreb  
Brussels  
Bucharest

Budapest  
Zagreb  
Copenhagen  
Dublin

Helsinki  
Kiev  
Zagreb  
Ljubljana

# What can you do with BRIN?

- Min-max for each block range
- Allows  $<$ ,  $=$ ,  $>$  searches
  - Much slower than B-tree lookups
  - Always scans the whole index (which is tiny though)
  - Always scans the whole heap page (range)
- Store bounding box for points, shapes
- Bloom filters

# What can you do with BRIN?

- Good for large tables with natural or accidental ordering
  - Tables loaded in primary key order
  - Timestamp columns
- A single out-of-order tuple in a page will “pollute” the index, and searches degenerate to full sequential scans.

# Summary

- B-tree
  - = < >
- GIN
  - B-tree on steroids
  - Stores duplicates efficiently
  - Multiple keys per heap tuple
- GiST
  - “containment” hierarchy
- Sp-GIST
  - Non-overlapping
- BRIN
  - Containment
  - For clustered data
  - Tiny index, slow searches