Introduction to Percona Server for MongoDB

Stéphane Combaudon
October 28th, 2015

Agenda

- What is PSMDB?
- Fractal Trees & PerconaFT
- LSM Trees & RocksDB
- Backups
- Migration to PSMDB

PERCONA

SERVER FOR MONGODB®

- Drop-in replacement for MongoDB 3.0 CE
  - Free and open-source
- Additional storage engines
  - PerconaFT and RocksDB
- Full online backup for PerconaFT & RocksDB
- External authentication plugin and audit plugin
  - Not available in MongoDB CE

Storage engines

- MMAP: B-Trees, no compression
- WiredTiger: B-Trees and compression
- PerconaFT: Fractal Trees and compression
- RocksDB: LSM Trees and compression

Which storage engine to choose?

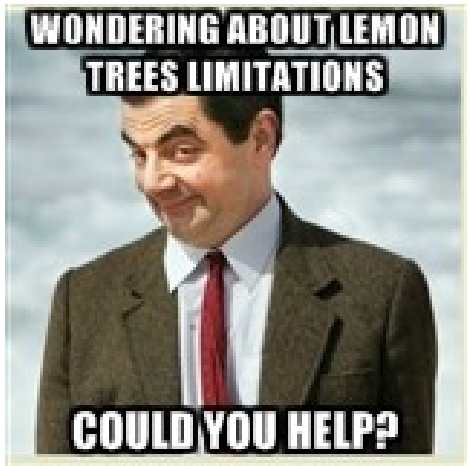
- MMAP
  - Default SE, not great performance but easier migration if you're coming from MongoDB
- WiredTiger
  - Good tradeoff between reads and writes
- RocksDB, PerconaFT
  - Write-optimized engines

Agenda

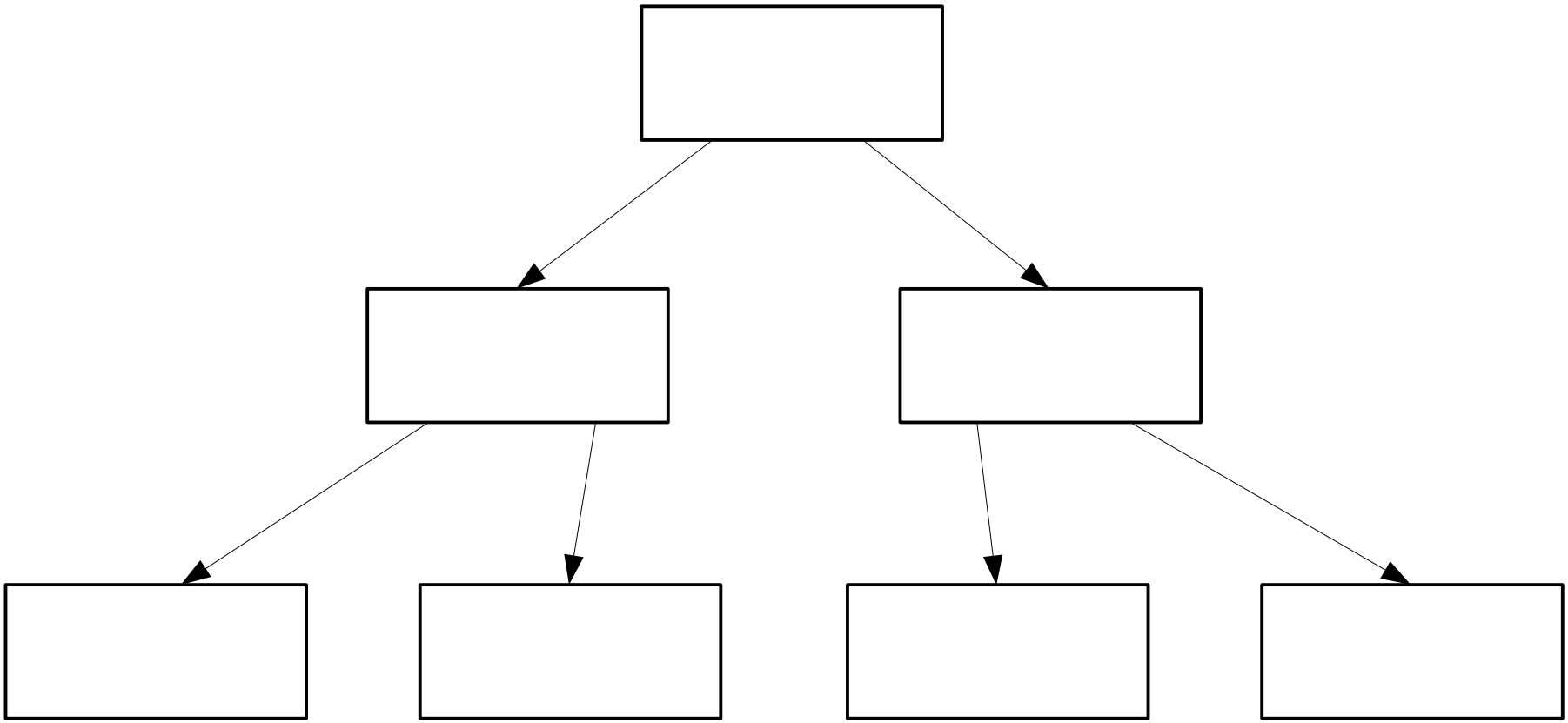
- What is PSMDB?
- Fractal Trees & PerconaFT
- LSM Trees & RocksDB
- Backups
- Migration to PSMDB

Back to B-Trees

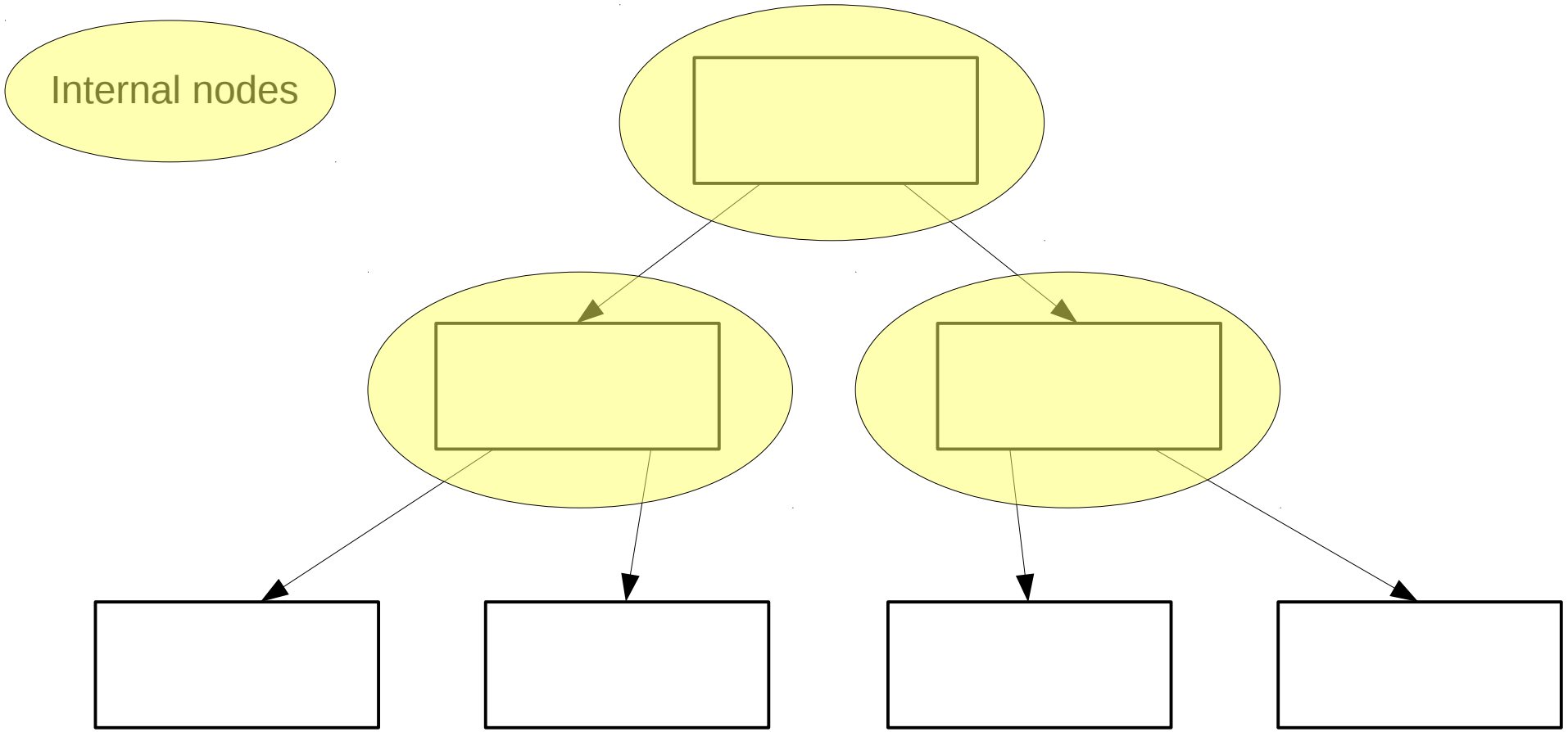
- Why are they so widely used?
  - Acceptable/good perf for most operations – reads and writes
- Main limitation: write perf when data is much larger than RAM



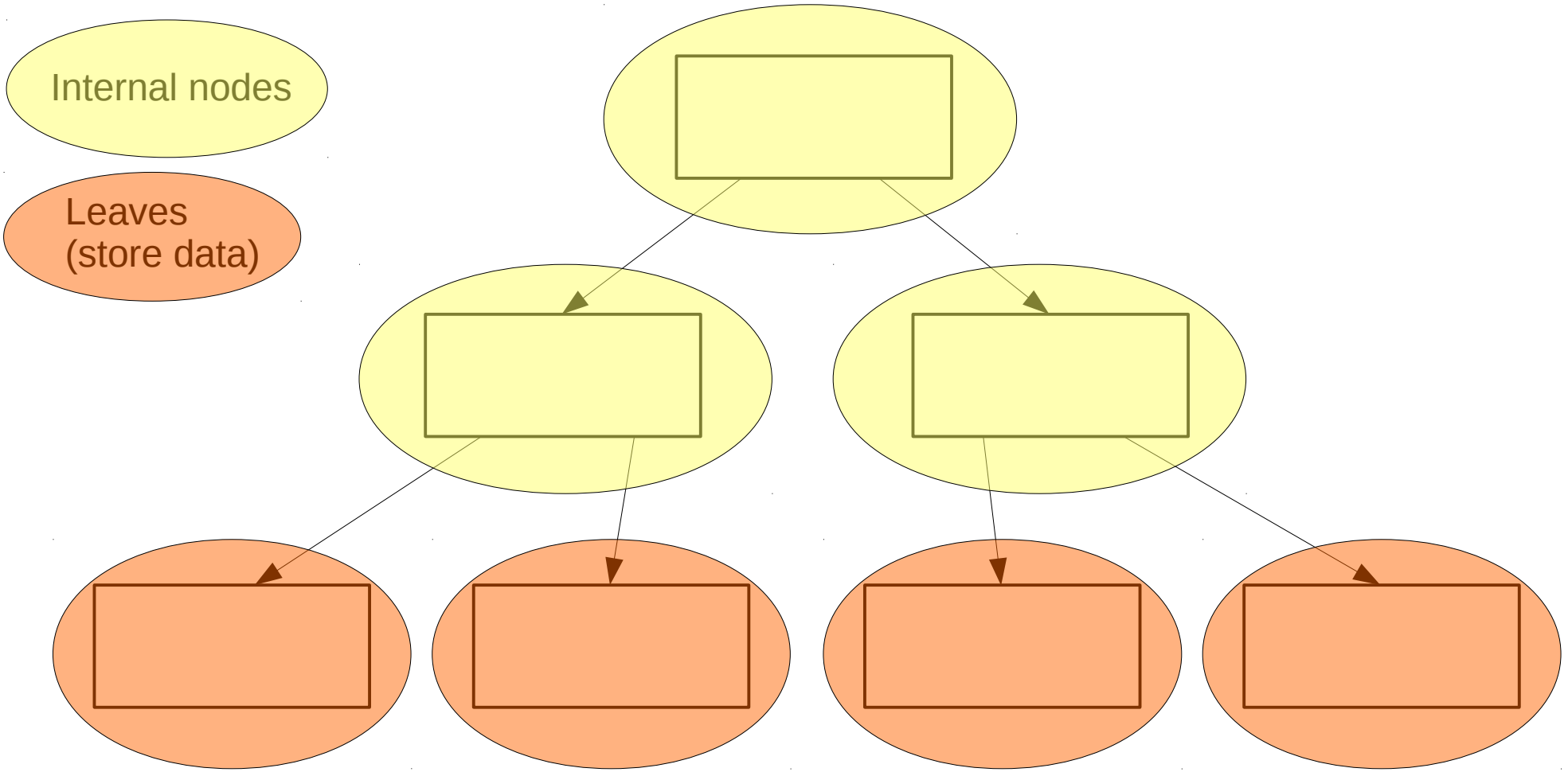
Simplified B-Tree



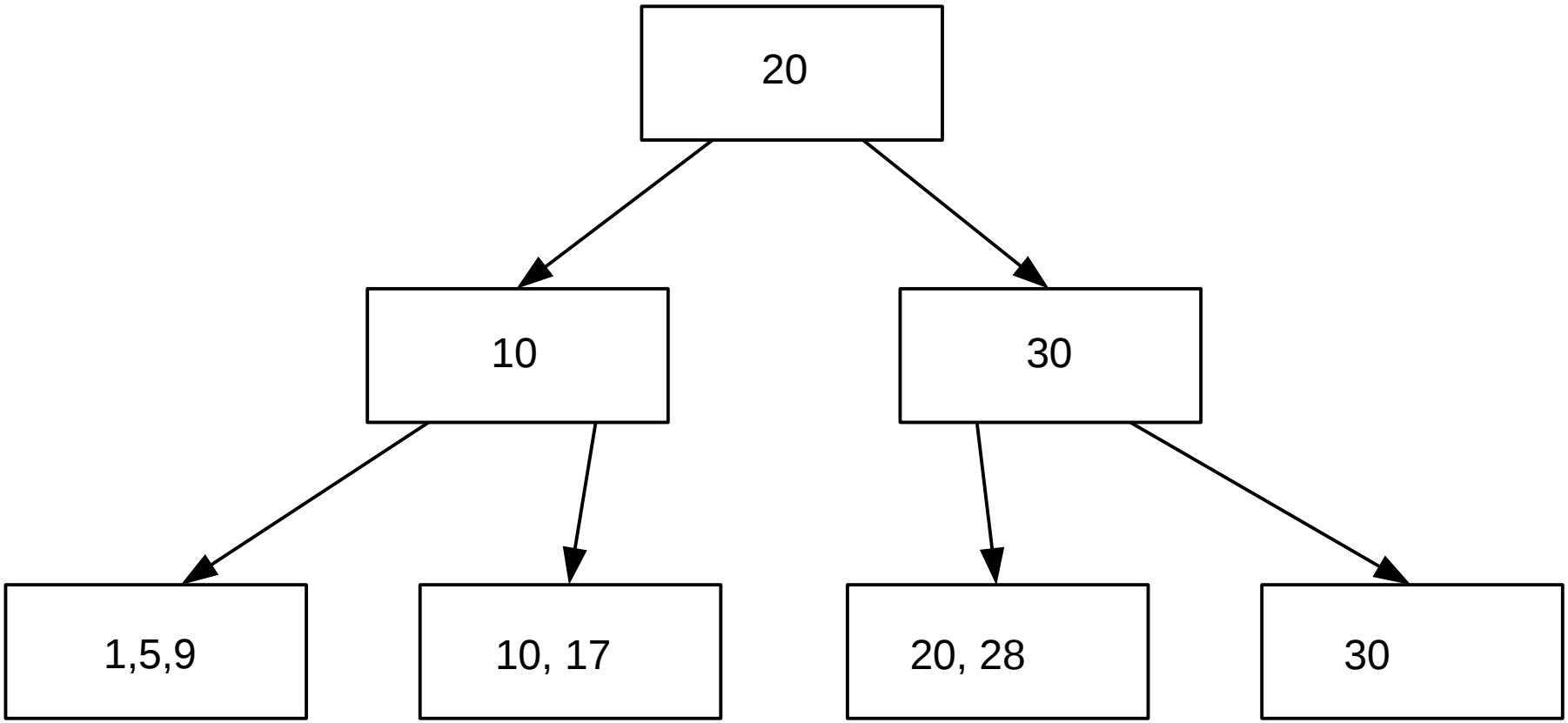
Simplified B-Tree



Simplified B-Tree

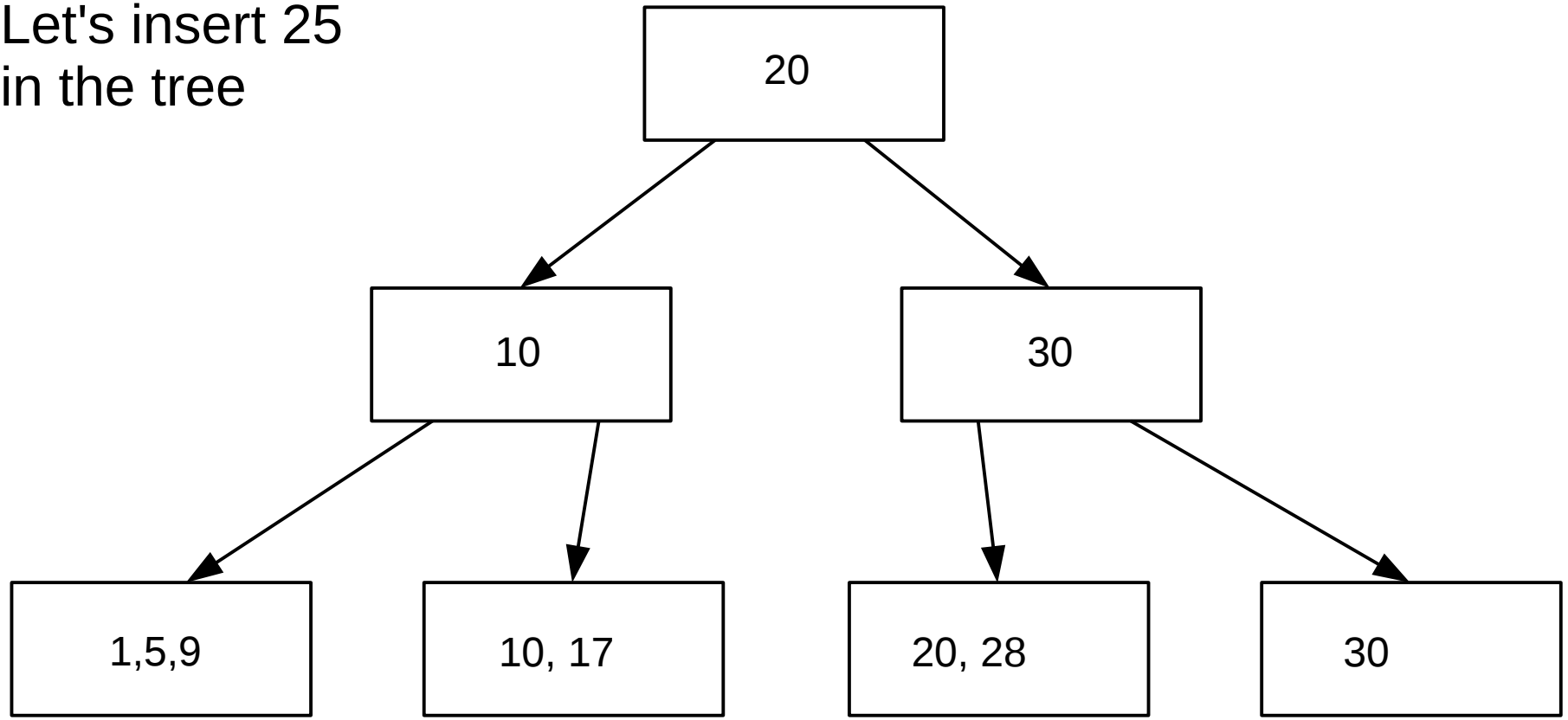


Insertion in a B-Tree



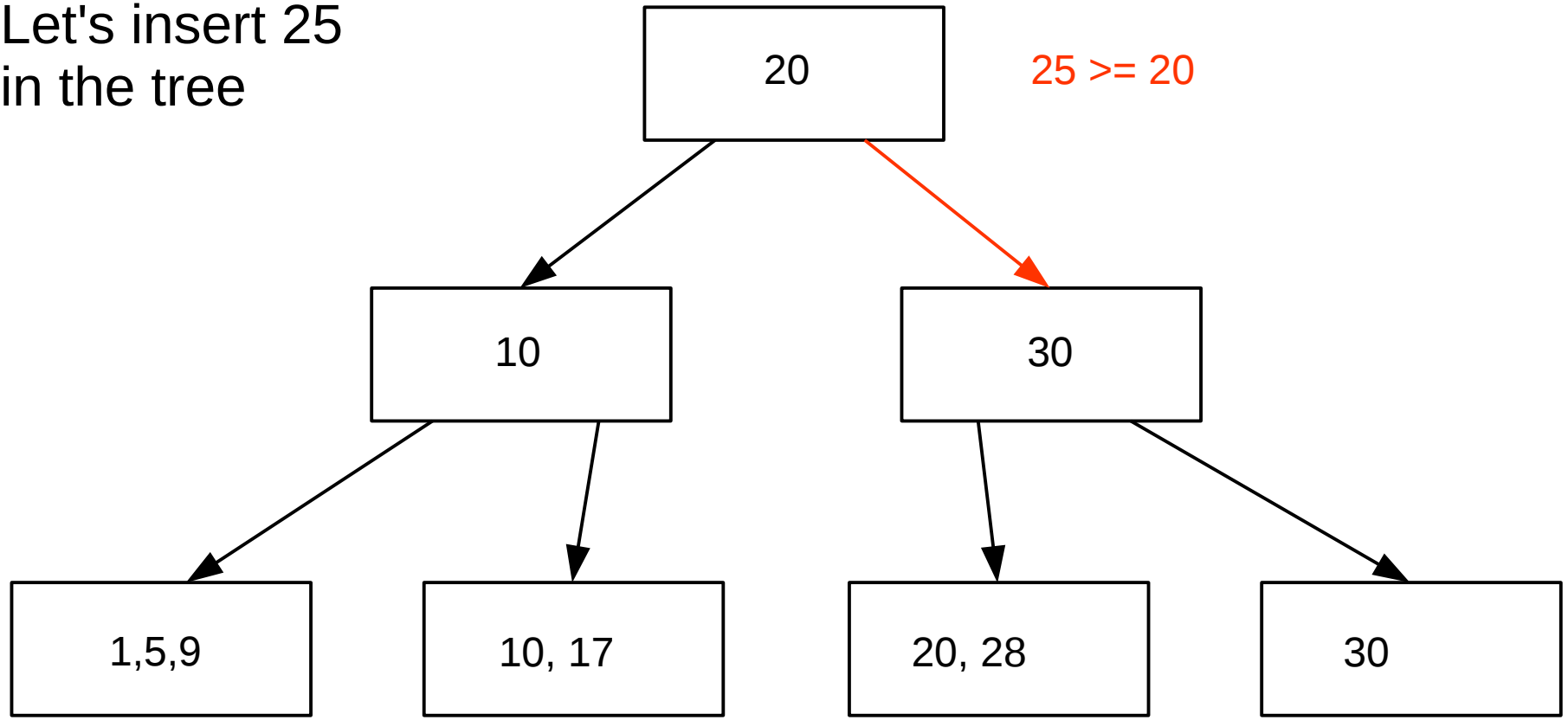
Insertion in a B-Tree

Let's insert 25
in the tree



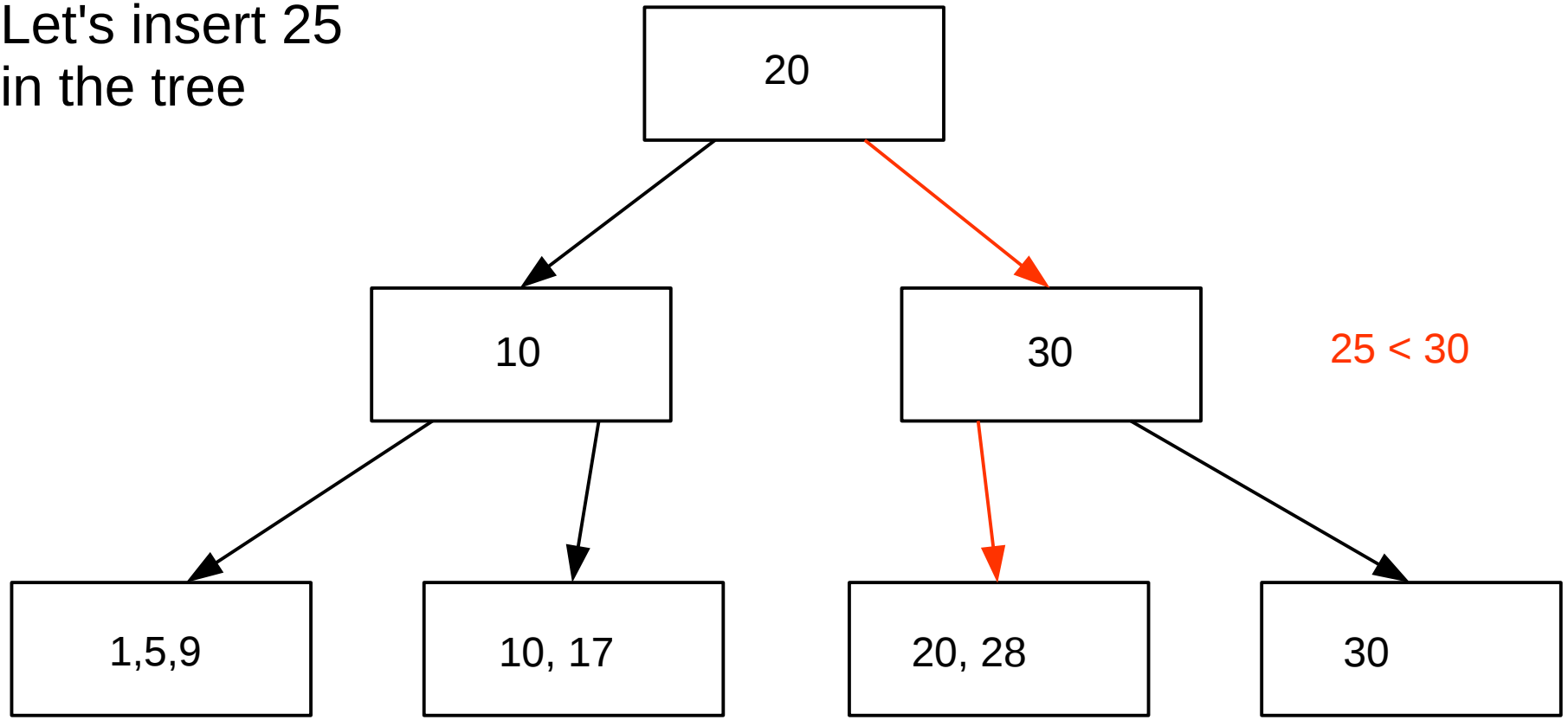
Insertion in a B-Tree

Let's insert 25
in the tree



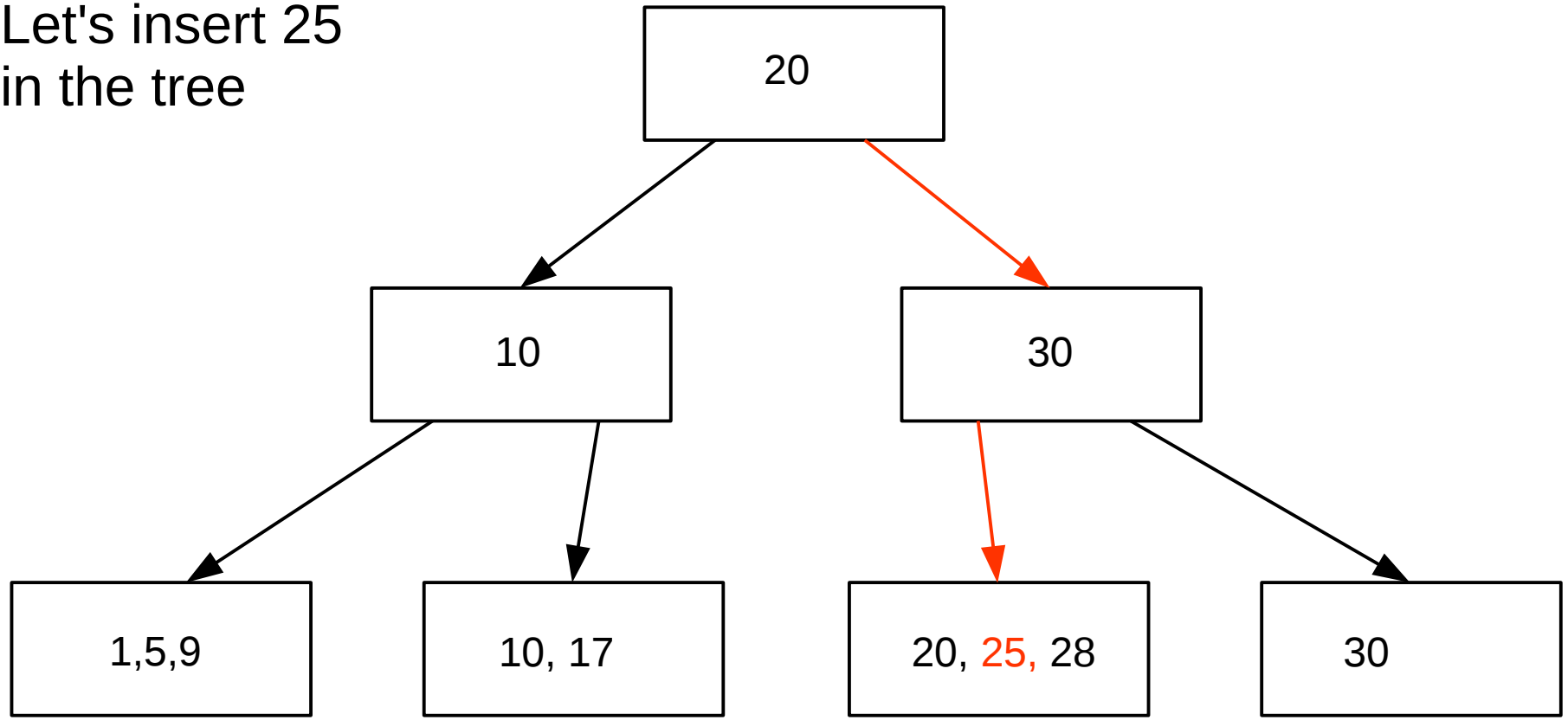
Insertion in a B-Tree

Let's insert 25
in the tree

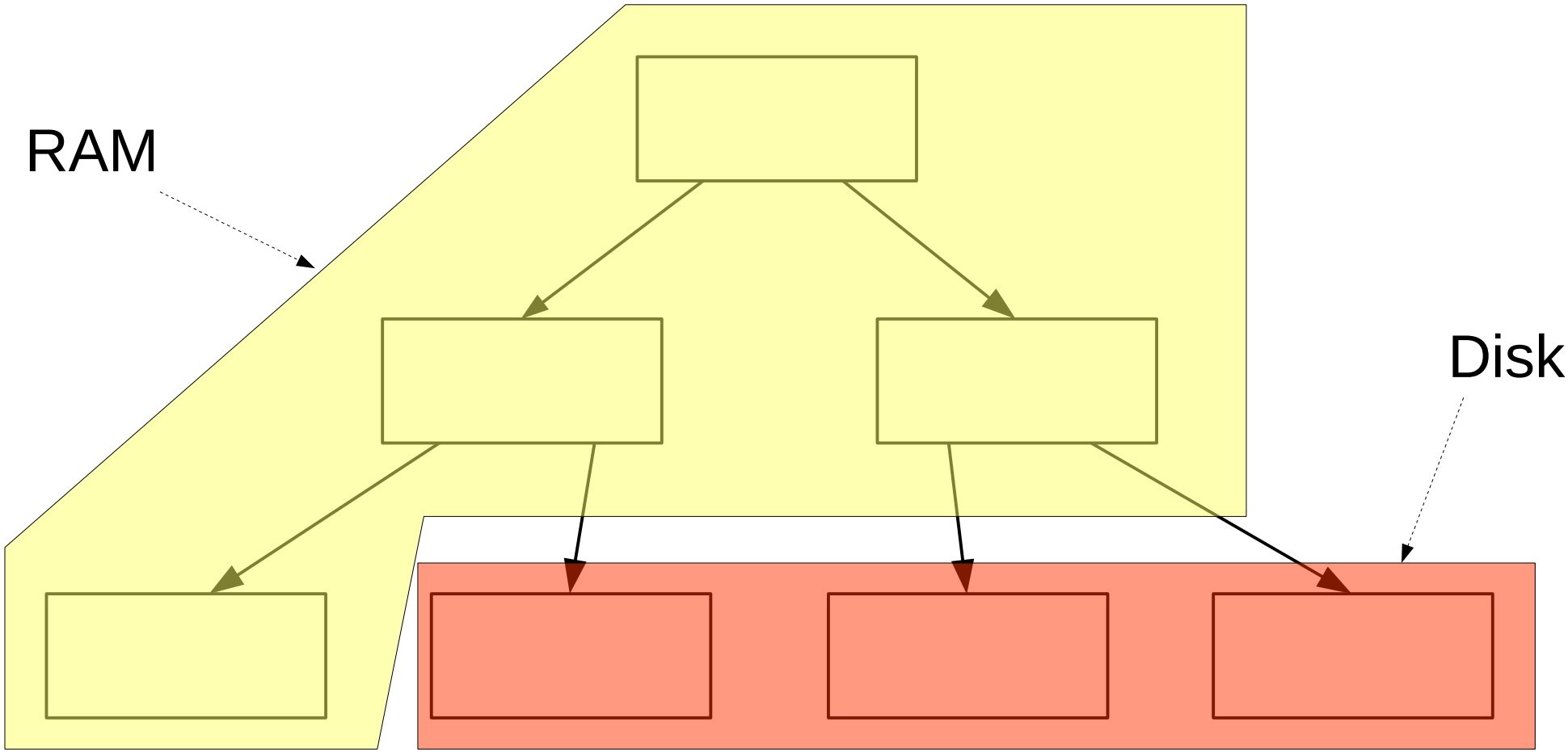


Insertion in a B-Tree

Let's insert 25
in the tree



RAM vs disk



- In general: internal nodes fit in RAM, leaves do not

Insertion in a B-Tree - Recap

- Find the leaf where the value has to be inserted
- Read the leaf ← an IO is needed here
- Update the leaf ← an IO is needed here

- Writing the leaf to disk can be delayed
  - By writing to a sequential journal instead

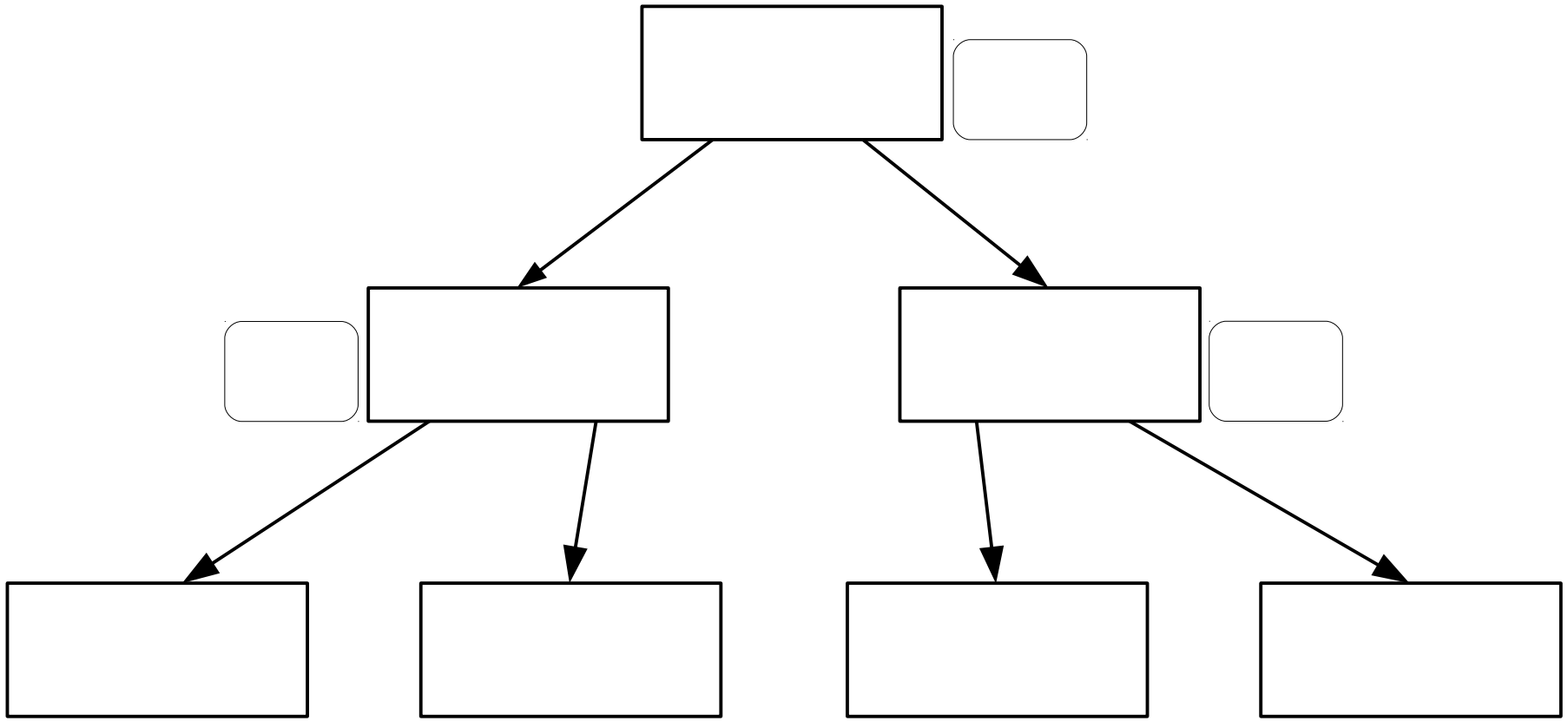
- Conclusion
  - At least one IO is needed for a B-Tree

Design rules for B-Trees

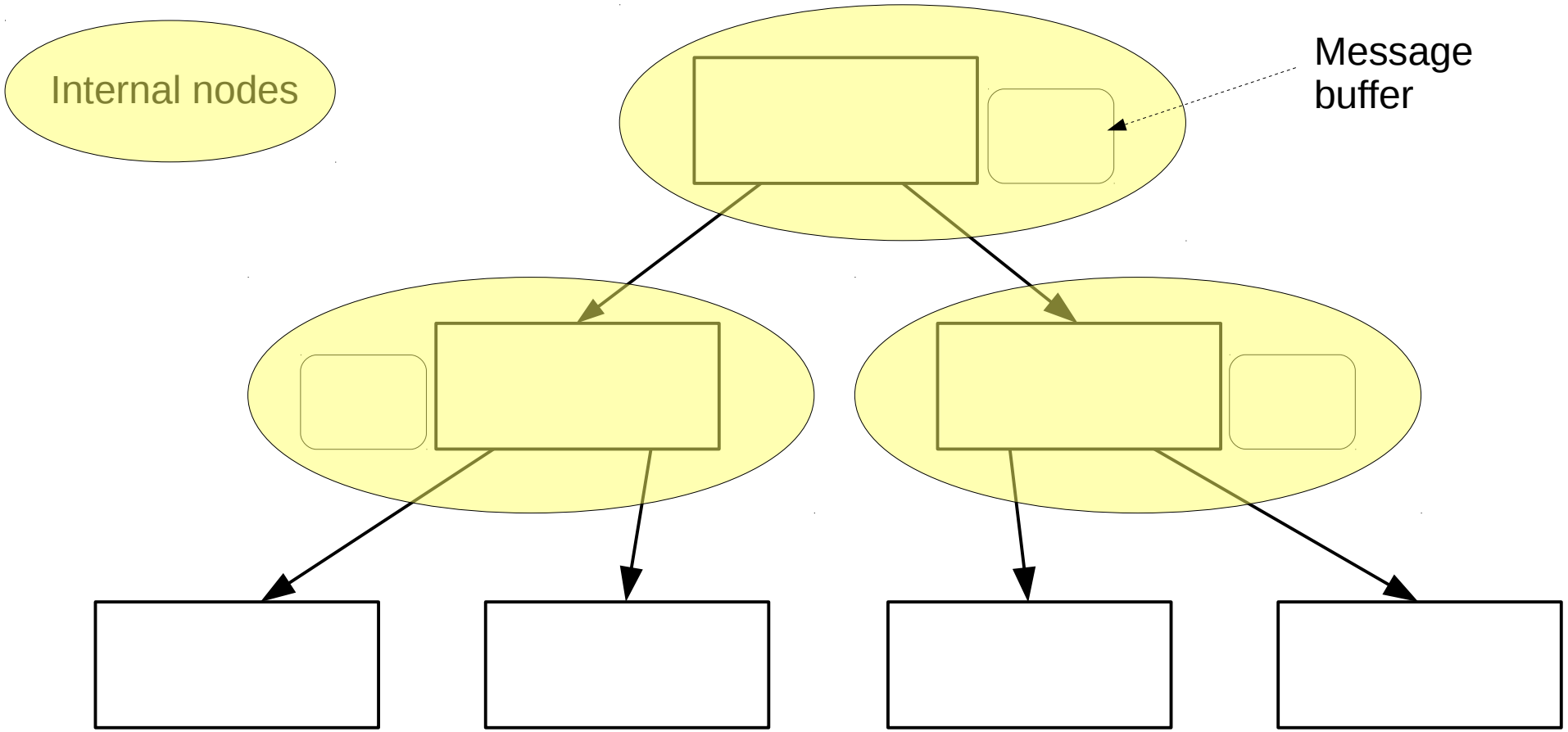
- Keep your indexes in memory
- Keep your working set in memory
- Have a right mostly insertion pattern

- All these rules come from B-Trees being IO bound when data is much larger than RAM

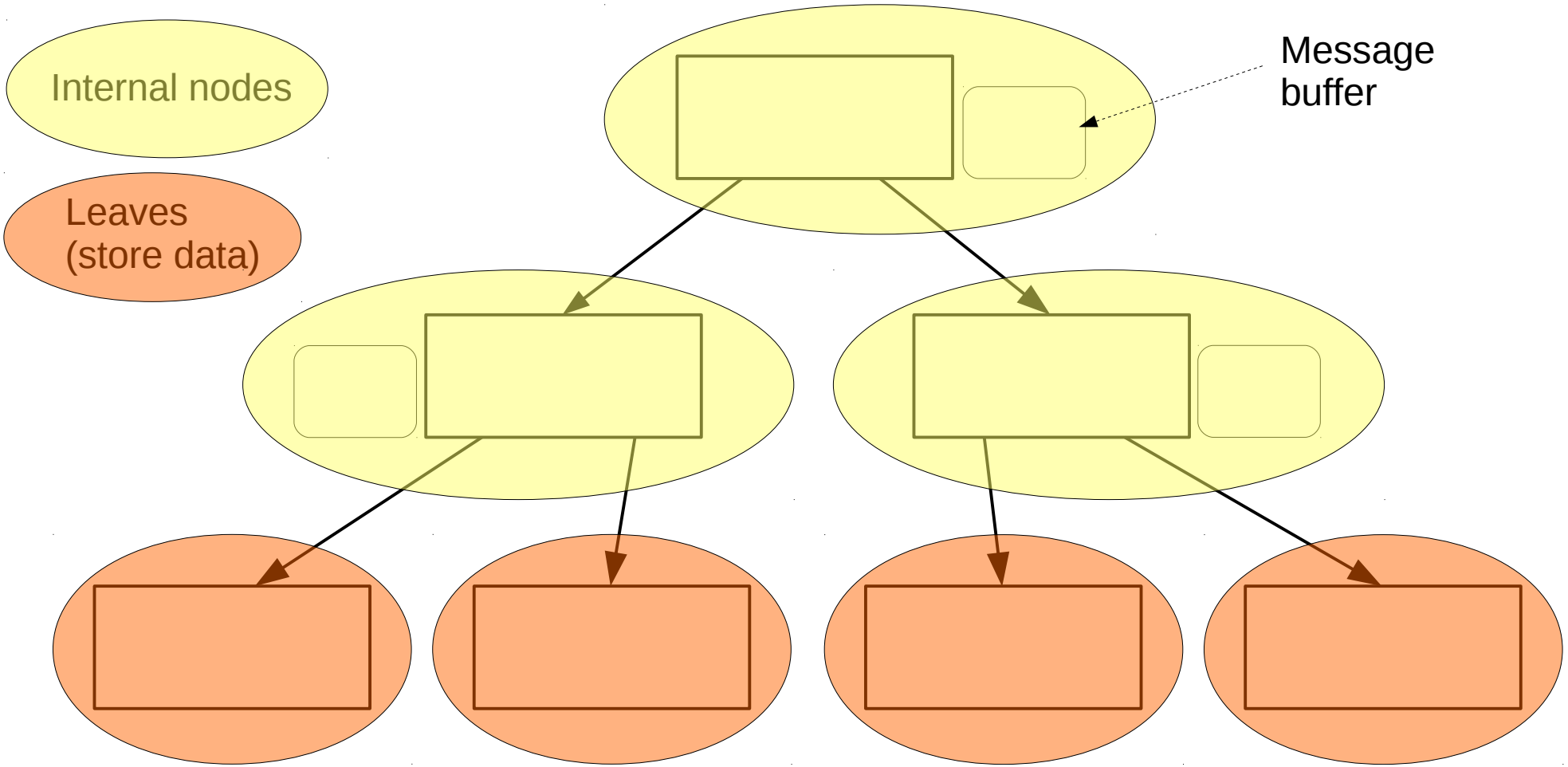
Overview of Fractal Trees



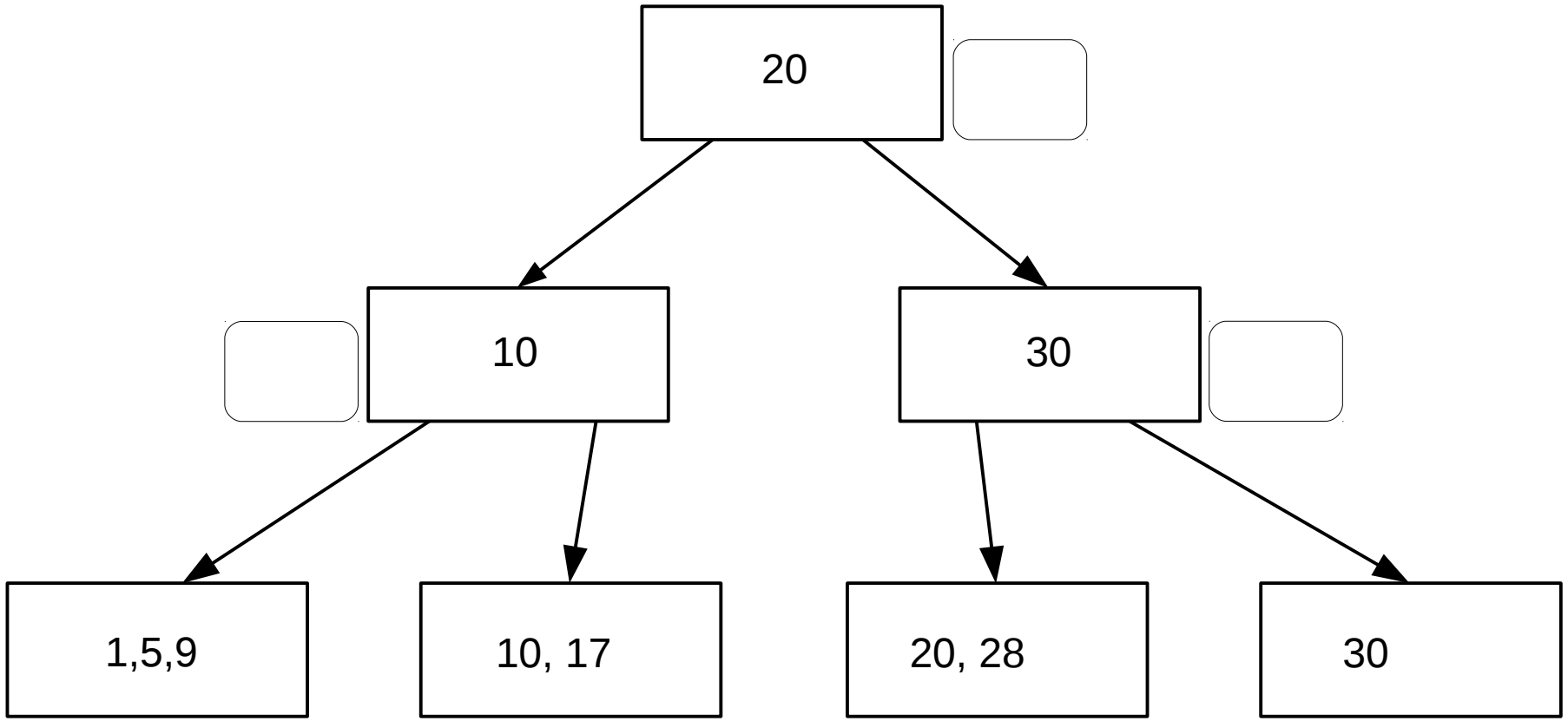
Overview of Fractal Trees



Overview of Fractal Trees

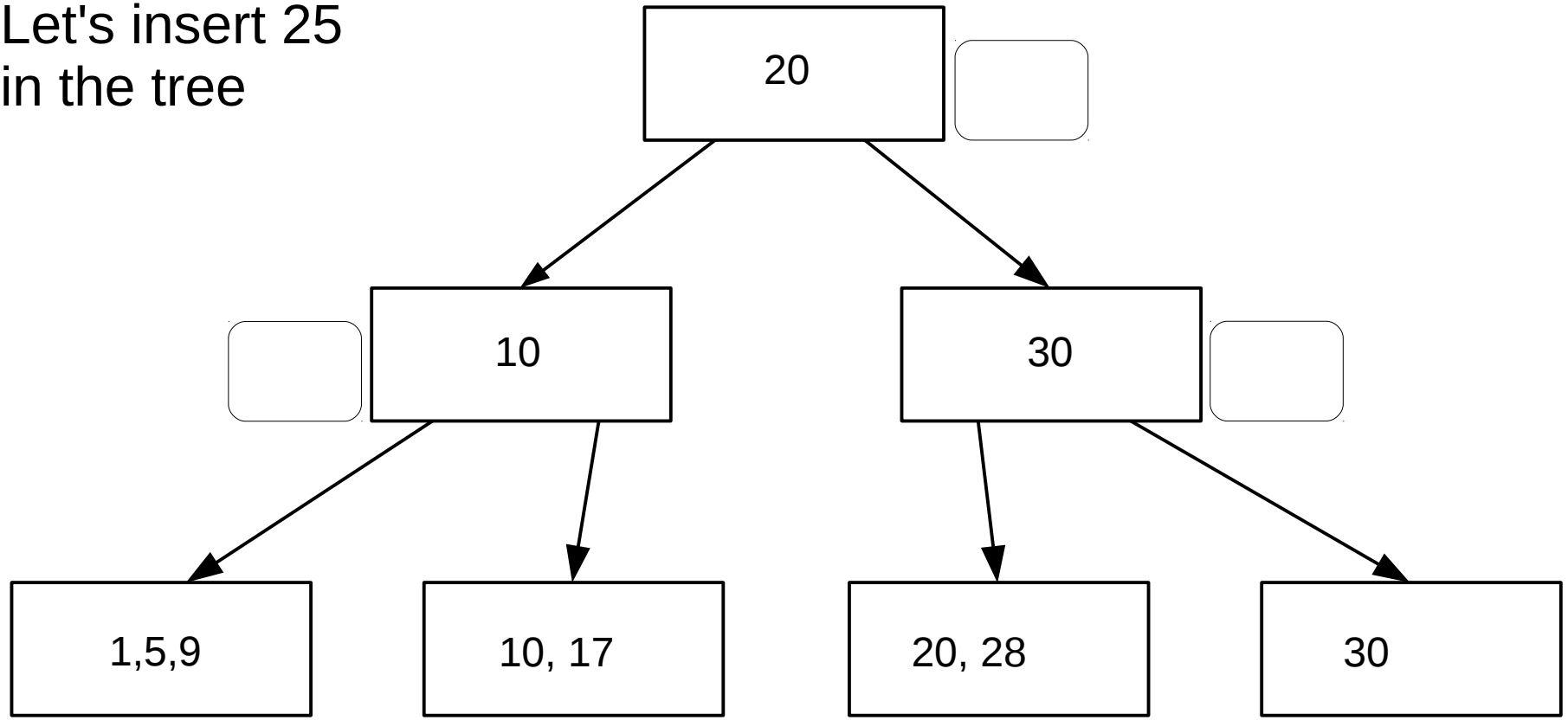


Insertion in a Fractal Tree - 1



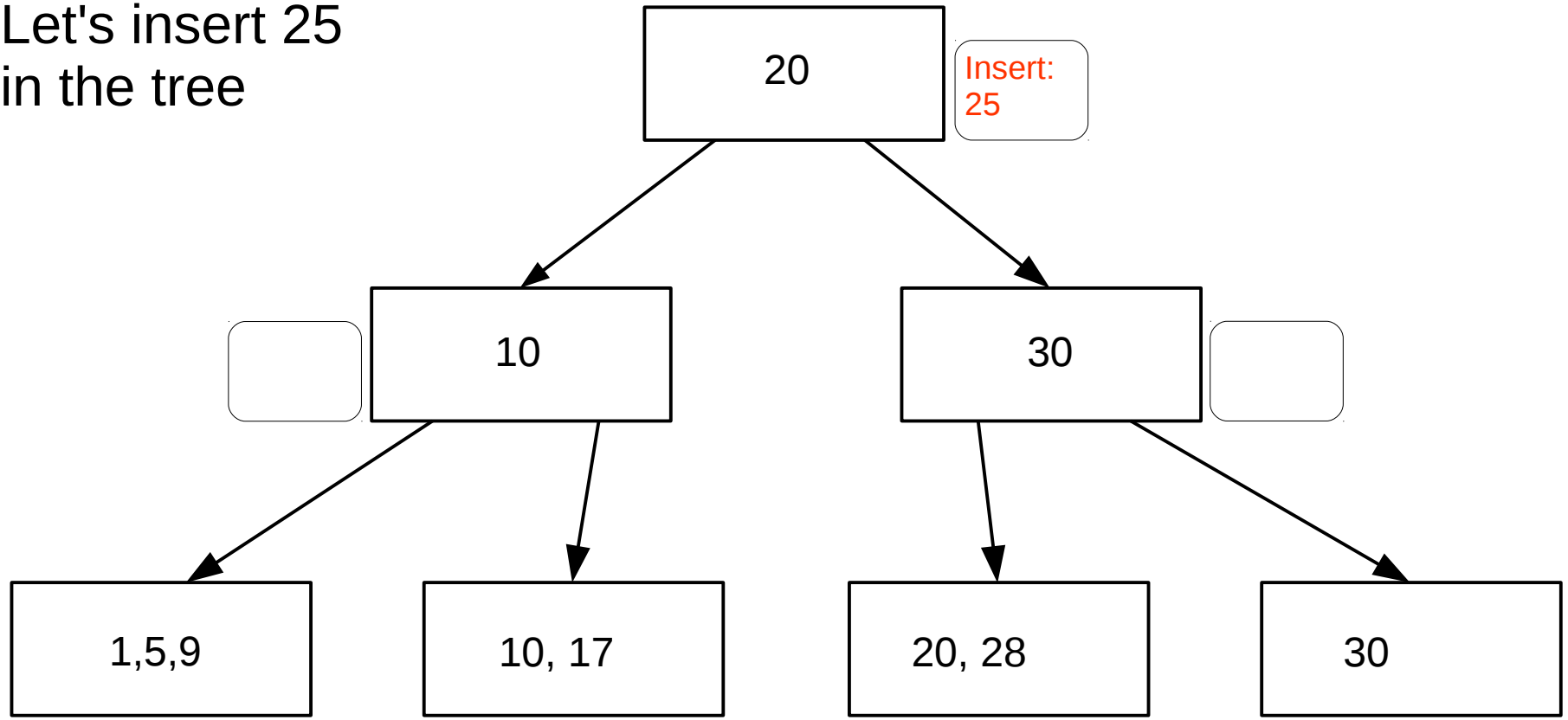
Insertion in a Fractal Tree - 2

Let's insert 25
in the tree



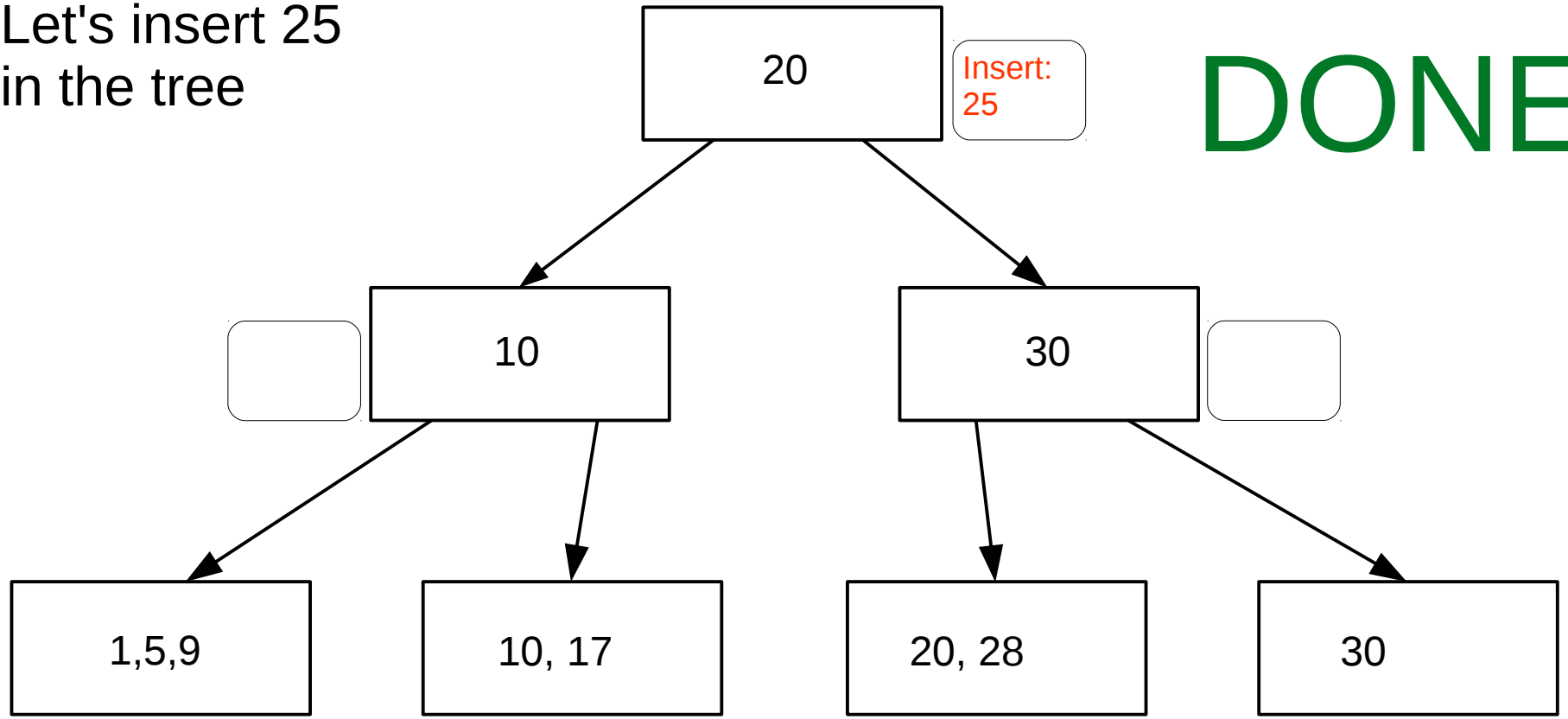
Insertion in a Fractal Tree - 3

Let's insert 25
in the tree



Insertion in a Fractal Tree - 3

Let's insert 25
in the tree

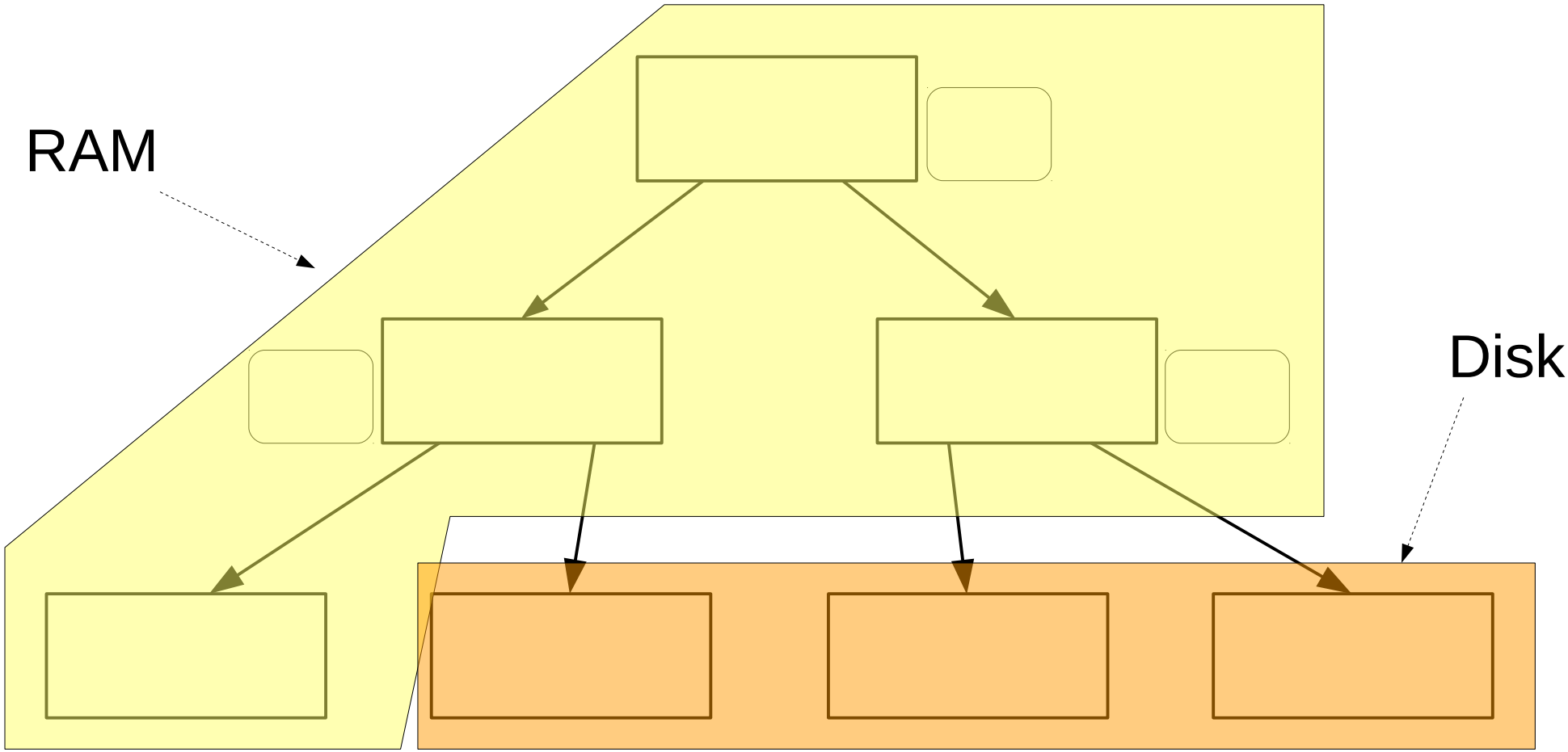


DONE!

The message buffer

- If the message buffer is full
  - Messages are moved down one level
  - If next buffer is full, messages go down one level again
- Eventually messages are applied to the leaves
  - A background thread apply message to the leaves (checkpointing)

RAM vs disk



- Similar to B-Trees

Insertion in a Fractal Tree - Recap

- Update the appropriate message buffer
  - Flush it if full
- No synchronous IO is needed
  - Except for the journal of course (seq. write - cheap)
  - Same for a B-Tree
- An IO is only performed when flushing a buffer
  - A buffer stores many document
  - Much more efficient

Design rules for Fractal Trees

- ~~Keep your indexes in memory~~
- ~~Keep your working set in memory~~
- ~~Have a right mostly insertion pattern~~

- These good old engineering rules no longer apply with fractal trees
  - Because IO is no longer the bottleneck

Fractal Tree - Leaves

- Leaves are larger with a fractal tree
  - 4MB
  - Good for write performance and for compression
  - But bad for point queries

- To mitigate read performance degradation
  - A 4MB leaf is made of 64KB basement nodes, compressed individually
  - On reads, 64KB are read, not 4MB

Limitations of Fractal Tree

- Checkpointing is expensive
  - Performance graphs often show dips every 60s
  - We're working on improvements
- Workloads with many delete operations can be problematic
  - Documents are not deleted synchronously
  - Reads may initially fetch documents that are discarded when reading the message buffers

Reads: B-Trees vs Fractal Trees

- B-Tree
  - Read the appropriate leaf node: IO needed
- Fractal Tree
  - Read the appropriate leaf node: IO needed
  - + some work to merge the potential messages in the internal buffers
  - + nodes are larger
    - Good for range queries
    - Bad for point queries

Main configuration settings

- --PerconaFTEngineCacheSize
  - Cache size for data and indexes
  - Data in RAM is uncompressed
- --PerconaFTDirectio
  - Direct IO or filesystem cache?
  - If directio=off, FS cache stores compressed data
    - Can be beneficial in some scenarios

Main configuration settings - 2

- --PerconaFTCollectionPageSize
  - Size of the leaves for the clustered index (default: 4MB)
- --PerconaFTCollectionReadPageSize
  - Size of the basement nodes for the clustered index (default: 64KB)
- --PerconaFTCollectionCompression
  - Compression algorithm for the clustered index: none, zlib, lzma, quicklz

Main configuration settings - 3

- --PerconaFTIndexPageSize
  - Size of the leaves for the secondary indexes (default: 4MB)
- --PerconaFTIndexReadPageSize
  - Size of the basement nodes for the secondary indexes (default: 64KB)
- --PerconaFTCollectionCompression
  - Compression algorithm for the secondary indexes: none, zlib, lzma, quicklz

Agenda

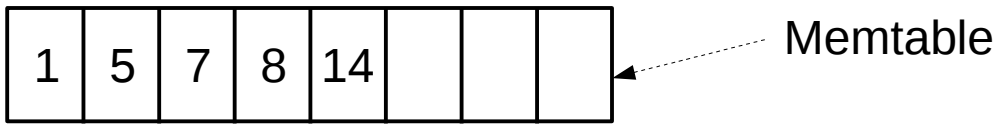
- What is PSMDB?
- Fractal Trees & PerconaFT
- LSM Trees & RocksDB
- Backups
- Migration to PSMDB

Overview of RocksDB

- Developed by Facebook
- Another write-optimized engine
  - Using LSM Trees
- http://rocksdb.org/

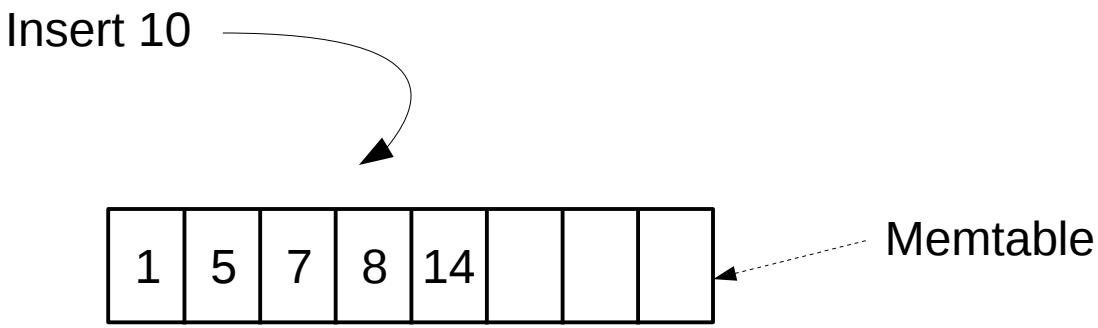
LSM Trees - 1

- New insertions are kept in memory in a sorted buffer (memtable)



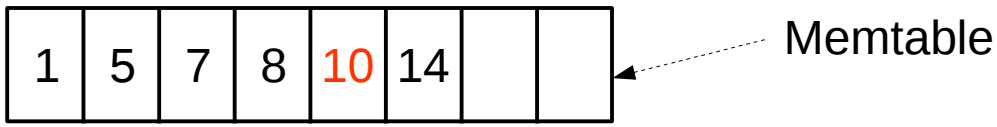
LSM Trees - 1

- New insertions are kept in memory in a sorted buffer (memtable)



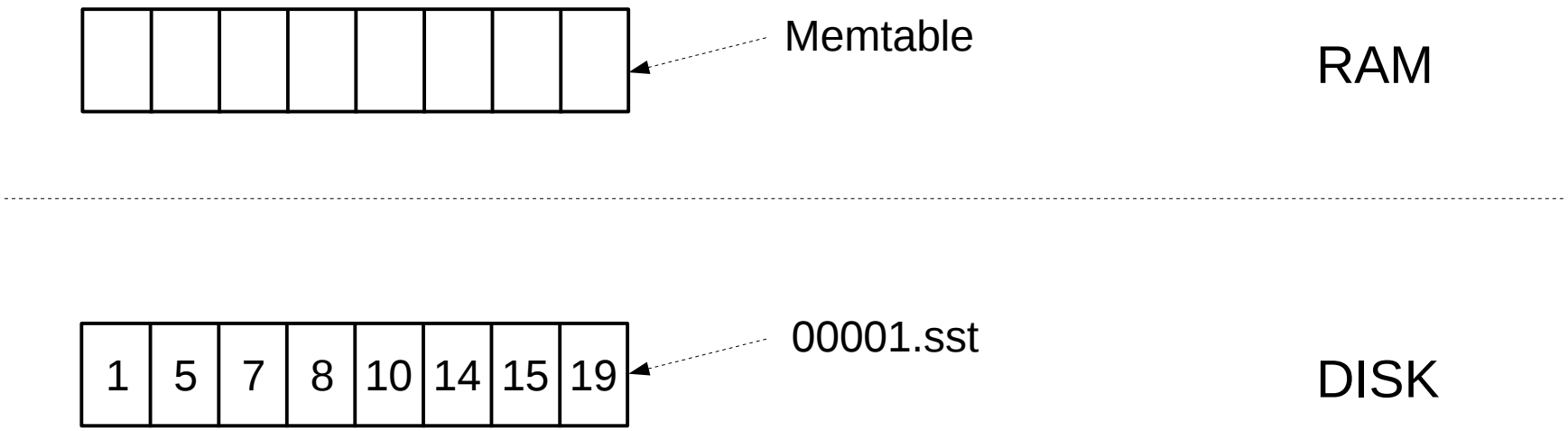
LSM Trees - 1

- New insertions are kept in memory in a sorted buffer (memtable)



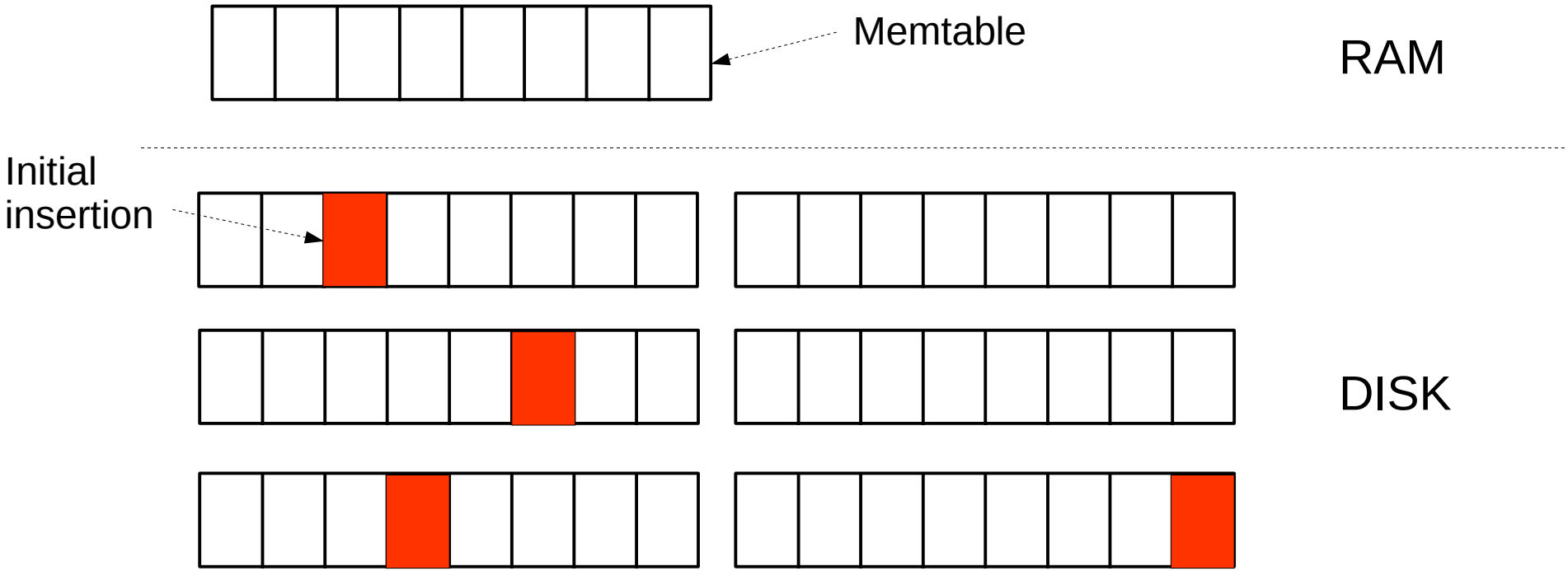
LSM Trees - 2

- When the memtable is full, it's flushed to disk
  - The corresponding .sst file is never updated



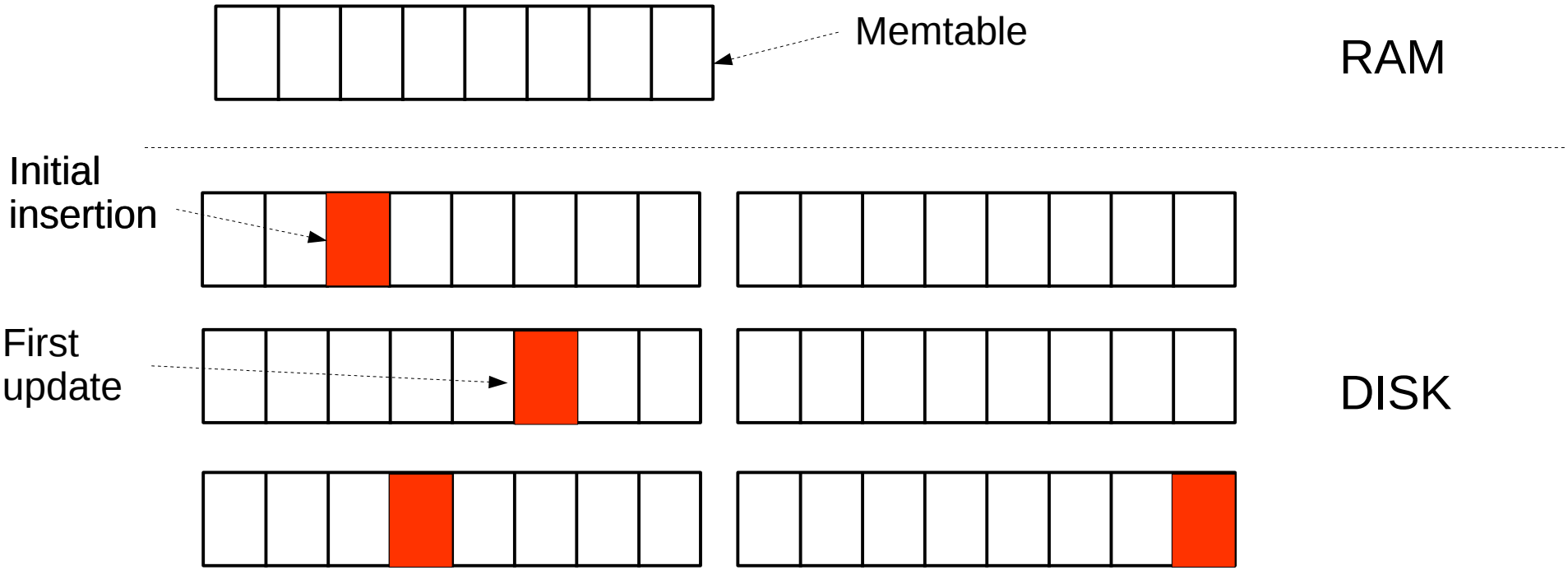
LSM Trees - 3

- After some time, we have many .sst files
  - Updates of the same record can be at multiple places



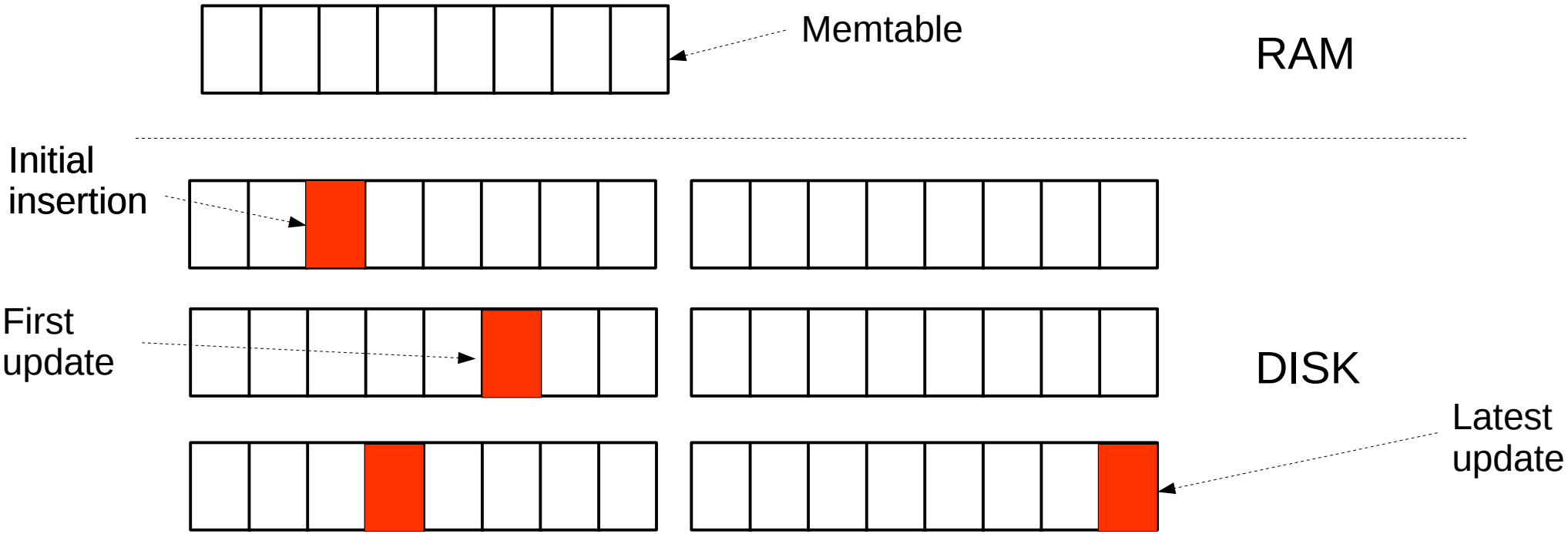
LSM Trees - 3

- After some time, we have many .sst files
  - Updates of the same record can be at multiple places



LSM Trees - 3

- After some time, we have many .sst files
  - Updates of the same record can be at multiple places

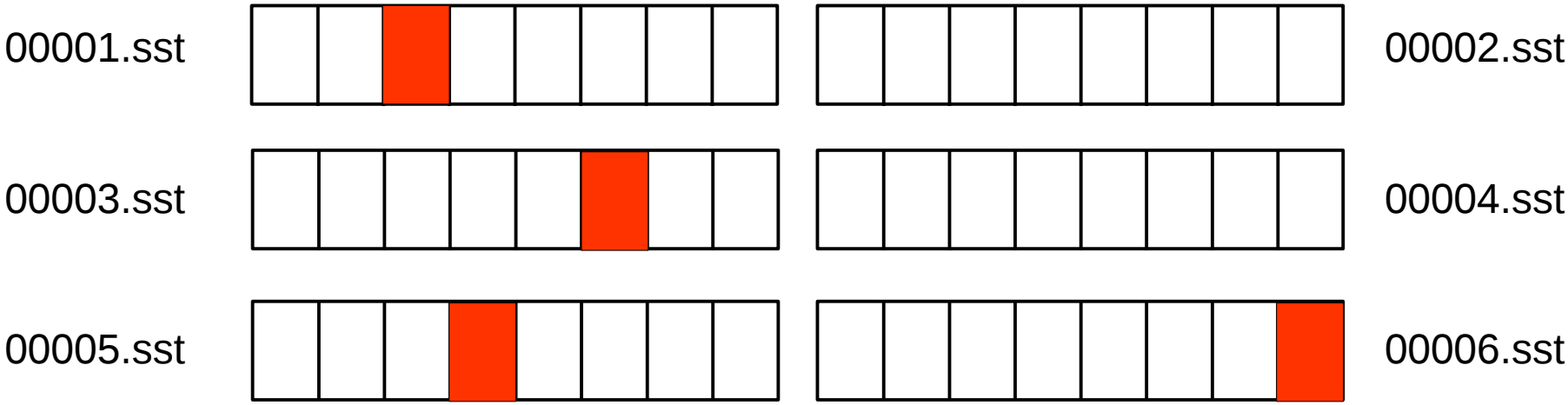


LSM Tree - 4

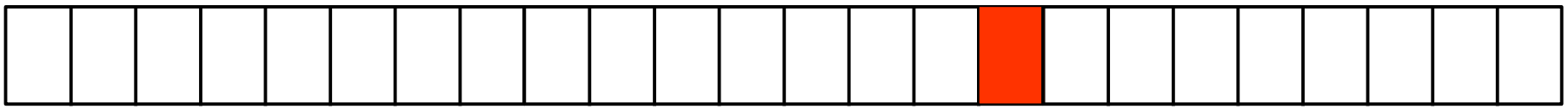
- Reading a record would then be expensive
  - Because we might need to read lots of .sst files
- So a periodic compaction is performed
  - .sst files are merged together

LSM Tree - 4

- Before compaction



- After compaction



LSM Tree - Recap

- Like for fractal trees, the idea is to minimize I/O for writes
  - The tradeoff is slower reads
  - Like for fractal trees, deletes can be problematic
- Read performance is increased with
  - Periodic compactions
  - Bloom filters: can efficiently tell if a record belongs to a .sst file

RocksDB – Main settings

- --rocksdbCacheSizeGB
  - Cache size for data and indexes
- --rocksdbCompression
  - Compression algorithm: none, snappy, zlib

Agenda

- What is PSMDB?
- Fractal Trees & PerconaFT
- LSM Trees & RocksDB
- Backups
- Migration to PSMDB

Good old backup methods

- Cold backup
  - Bring a replica down
  - Copy all files
  - Restart it
- Volume snapshot
  - LVM, EBS, etc
- Both can work with any storage engine

Online backup for PerconaFT

- The hot backup tool intercepts all system calls
  - Whenever a write is performed, it is mirrored in the backup directory

- Run this command

```
db.adminCommand({backupStart: “/path/to/backup/dir”})
```

Online backup for RocksDB

- Immutable files make backups easy

```
db.adminCommand({setParameter:1,
rocksdbBackup: “/path/to/backup/dir”})
```

- This command
  - Flushes the memtable
  - Creates hard links to .sst files

- You then simply need to copy the files in the backup directory

Agenda

- What is PSMDB?
- Fractal Trees & PerconaFT
- LSM Trees & RocksDB
- Backups
- Migration to PSMDB

Migration from MongoDB 3.0

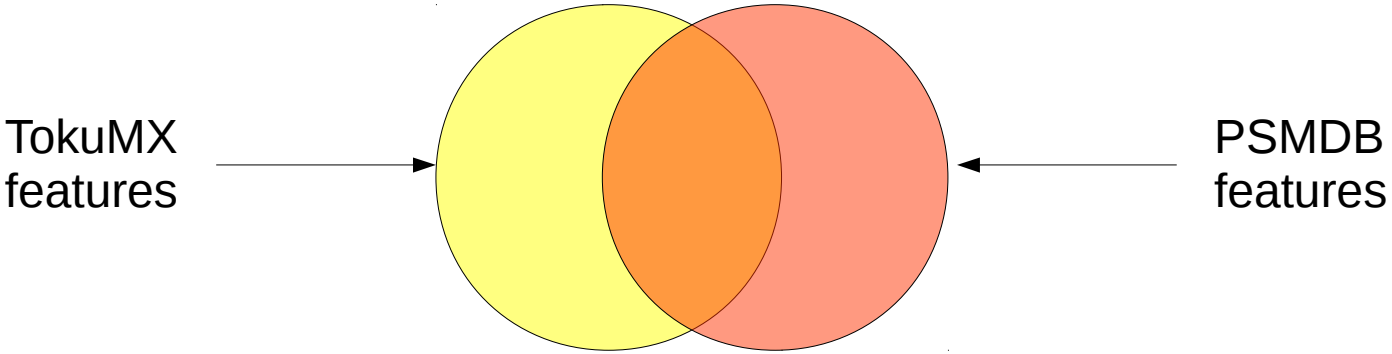
- Easiest scenario
  - PSMDB is a drop-in replacement
- Stop mongod / swap binaries / restart mongod
  - https://www.percona.com/doc/percona-server-for-mongodb/upgrading_guide_mongodb_psmdb.html
- A replica set can have MongoDB instances and PSMDB instances

Migration from MongoDB 2.6

- A replica set can have members running 2.6 and members running 3.0
- Upgrading a member is mostly swapping binaries
  - But there are changes between 2.6 and 3.0
  - Test carefully: perform may vary, queries may break

Migration from TokuMX

- TokuMX and PSMDB have a different feature set



- Yellow area: migration will be difficult
- Orange area: migration will be easier
- Some helper scripts :
  https://github.com/dbpercona/tokumx2_to_psmdb3_migration

Q&A

Thank you for attending!