CSE544
Data Management

Lecture 14

LSM Trees

Outline

- Briefly discuss Learned Indexes

- LSM Trees

Learned Index Structures

- What are the arguments in favor of learned index structures?

- Why is an index a “model”?

- What does Neumann’s blog say?

Learned Index Structures

- What are the arguments in favor of learned index structures?
  - B+ trees, hash tables: distribution agnostic
  - GPU/TPU: efficient for regression model
- Why is an index a "model"?

- What does Neumann's blog say?

The Case for Learned Index Structures,
Kraska et al., SIGMOD'2018

Learned Index Structures

- What are the arguments in favor of learned index structures?
  - B+ trees, hash tables: distribution agnostic
  - GPU/TPU: efficient for regression model
- Why is an index a "model"?
  - Index maps key value to position
  - Regression model does the same
- What does Neumann's blog say?

The Case for Learned Index Structures,
Kraska et al., SIGMOD'2018

Learned Index Structures

- What are the arguments in favor of learned index structures?
  - B+ trees, hash tables: distribution agnostic
  - GPU/TPU: efficient for regression model
- Why is an index a "model"?
  - Index maps key value to position
  - Regression model does the same
- What does Neumann's blog say?
  - Use a *simple* regression model

Learned Index Structures

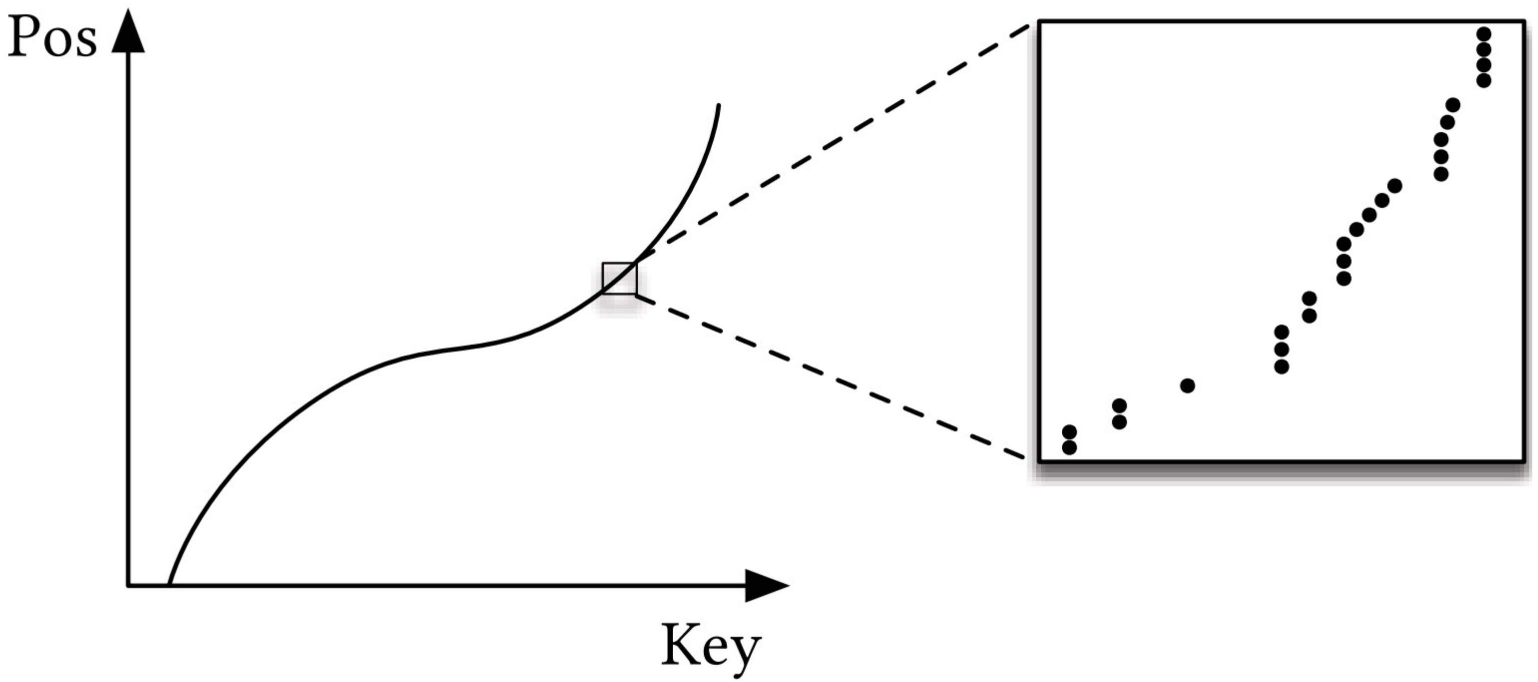


Figure 2: Indexes as CDFs

Discussion

(in class)

Outline

- Briefly discuss Learned Indexes

- LSM Trees

Slides based on
Monkey: Optimal Navigable Key-Value Store,
Dayan, Athanassoulis, Idreos,
SIGMOD'2017
Reading for Monday!!

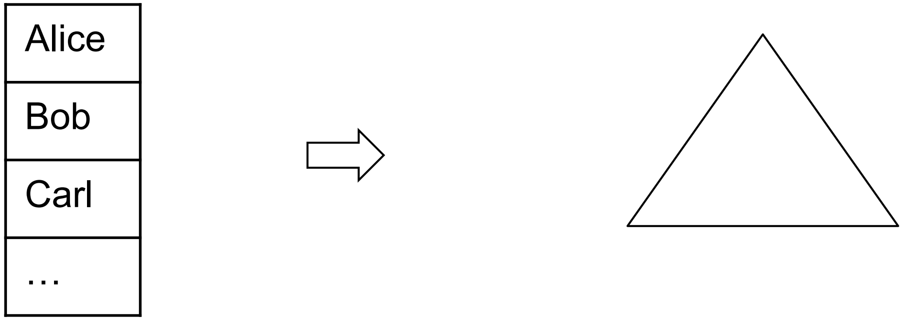
Motivation

- Sorted arrays = best for reads
- Unsorted log file = best for writes
- B+ trees = good for read, so-so for write

- LSM trees = optimize the writes
- Notice:
  - Primary (clustered) index only
  - Key/value stores, but also relational DBs

More Motivation

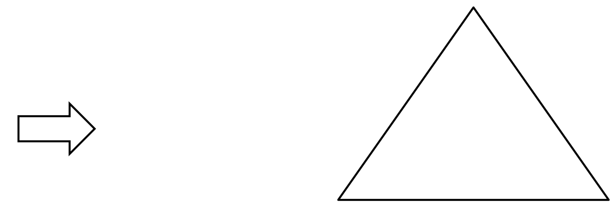
Index for one attribute:
Person.name



More Motivation

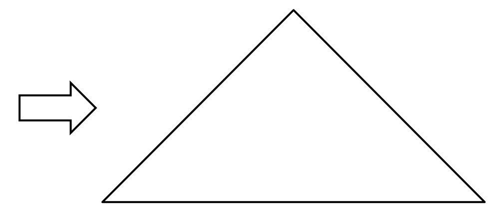
Index for one attribute:
Person.name

Alice
Bob
Carl
…



Index for entire table:
Person(name,age,city)

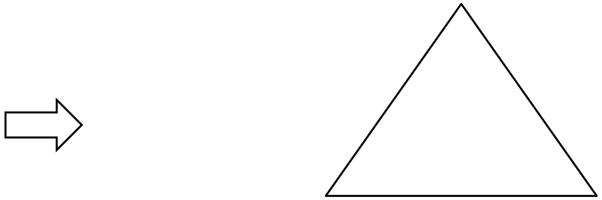
Alice	22	Seattle
Bob	53	Kent
Carl	37	Pasco
…		



More Motivation

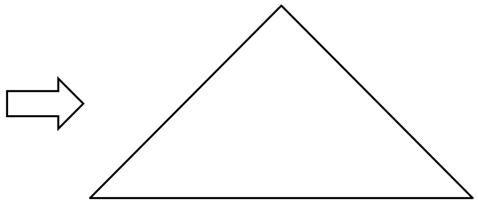
Index for one attribute:
Person.name

Alice
Bob
Carl
…



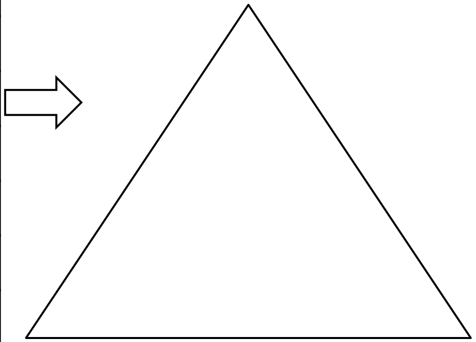
Index for entire table:
Person(name,age,city)

Alice	22	Seattle
Bob	53	Kent
Carl	37	Pasco
…		



Index for entire db:
Person, Dept, Project,…

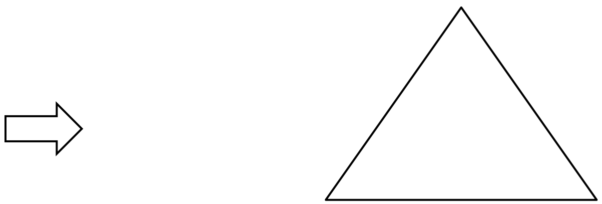
Person	Accounting	4th floor	
Person	Sales	2nd floor	
Person	…		
Dept	Alice	22	Seattle
Dept	Bob	53	Kent
Dept	Carl	37	Pasco
Project	Compiler	$55000	
Project	Database	$77000	
Project	…		



More Motivation

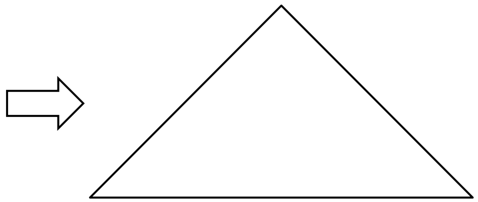
Index for one attribute:
Person.name

Alice
Bob
Carl
…



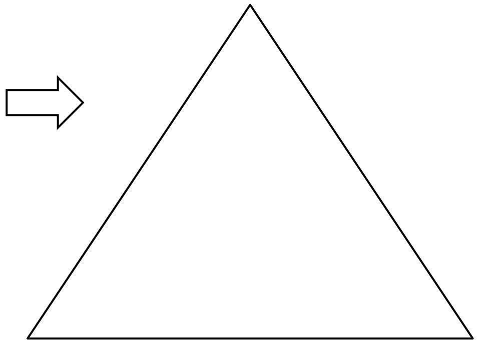
Index for entire table:
Person(name,age,city)

Alice	22	Seattle
Bob	53	Kent
Carl	37	Pasco
…		



Index for entire db:
Person, Dept, Project,…

Accounting	4th floor	
Sales	2nd floor	
…		
Alice	22	Seattle
Bob	53	Kent
Carl	37	Pasco
Compiler	$55000	
Database	$77000	
…		



E.g. MySQL
on RocksDB
using MyRocks

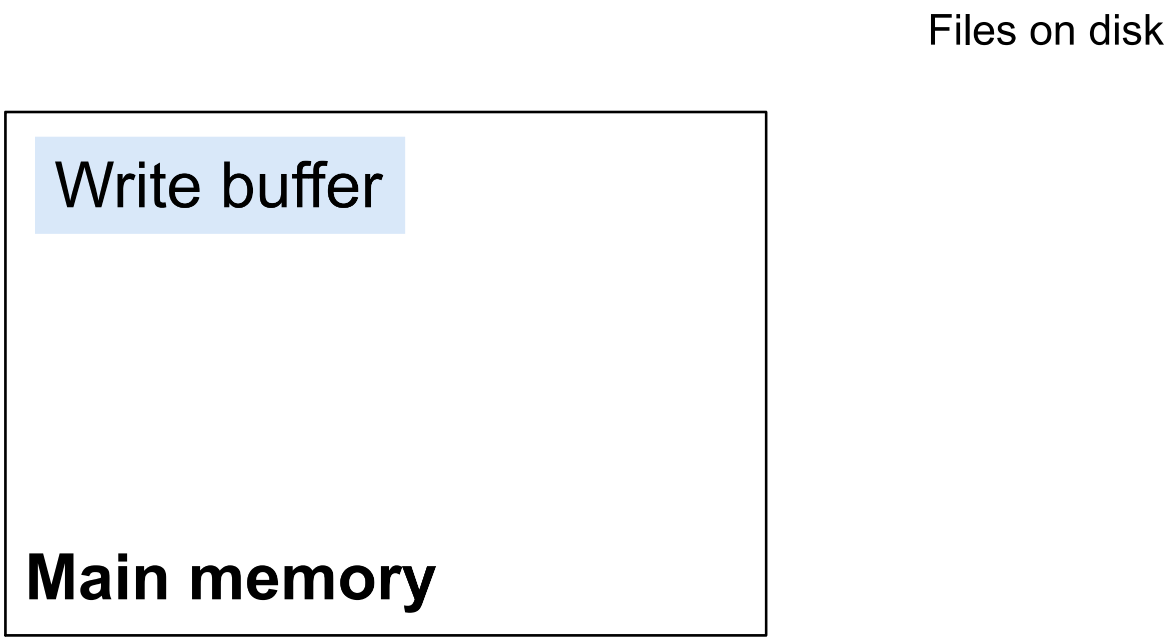
Three Main Ideas for Writes

1. Store writes in a buffer in main memory
   When full: spill to disk

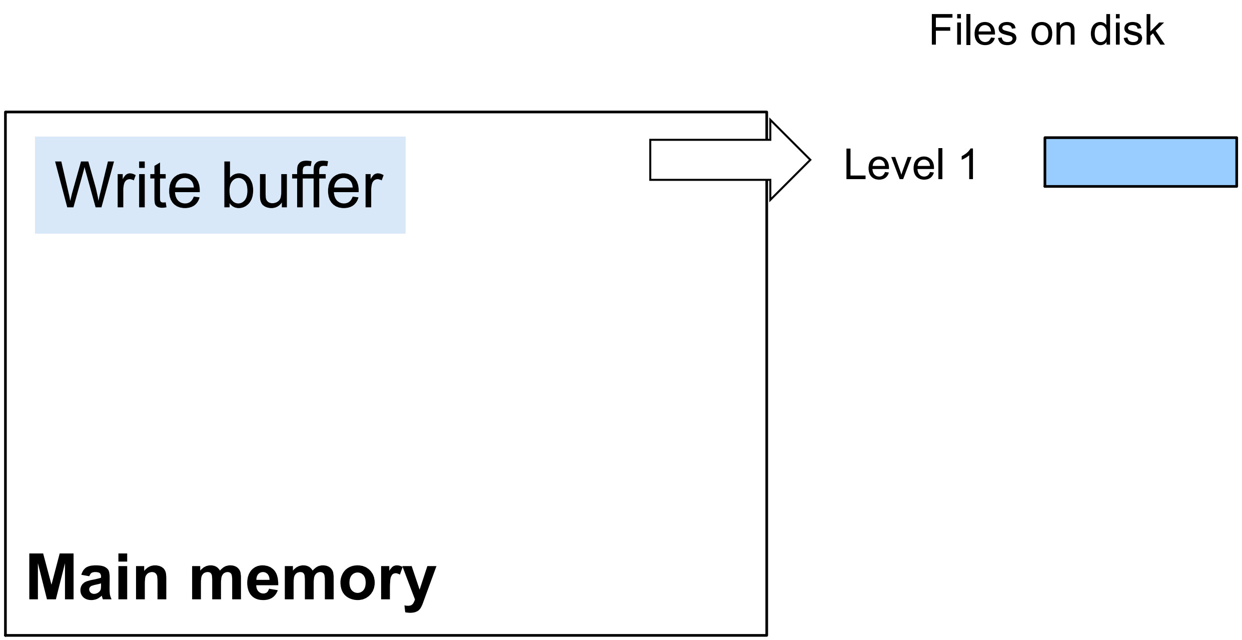
2. Spilling to disk (instead of a B+ tree):
   Sort and write to a sorted file.

3. When too many sorted files:
   Merge them to a larger sorted file

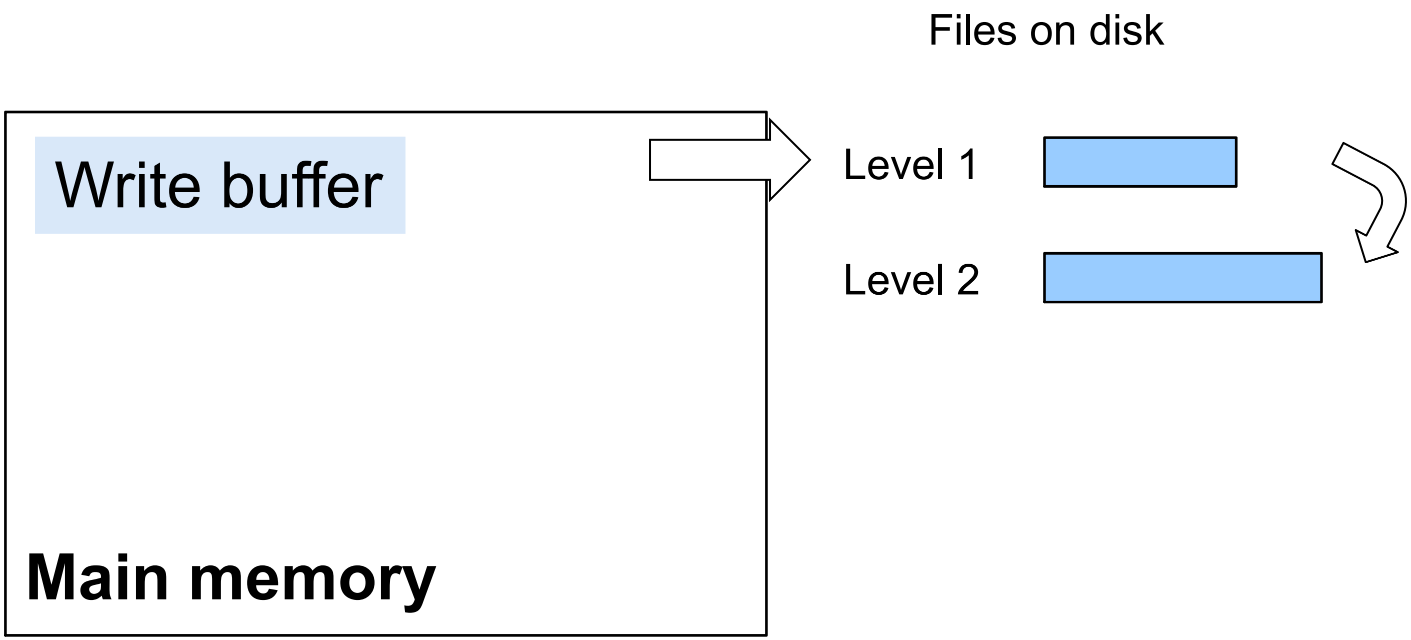
LSM Trees



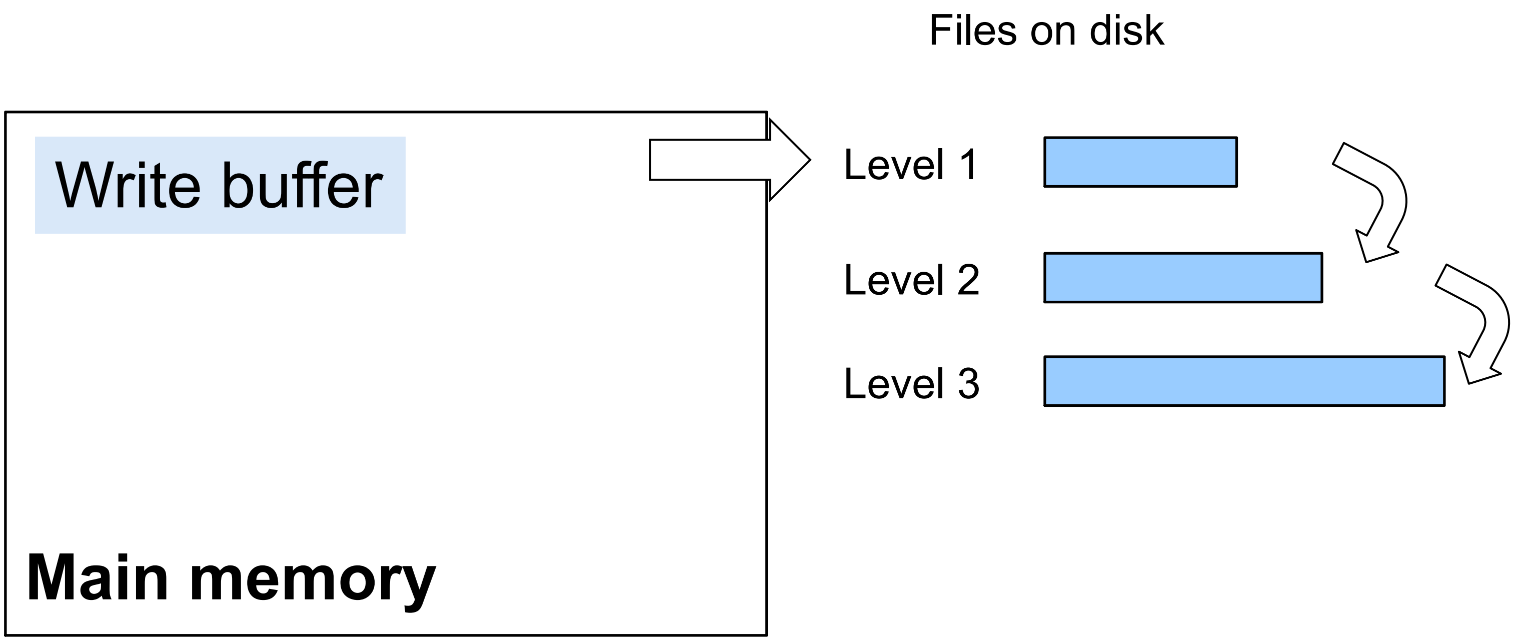
LSM Trees



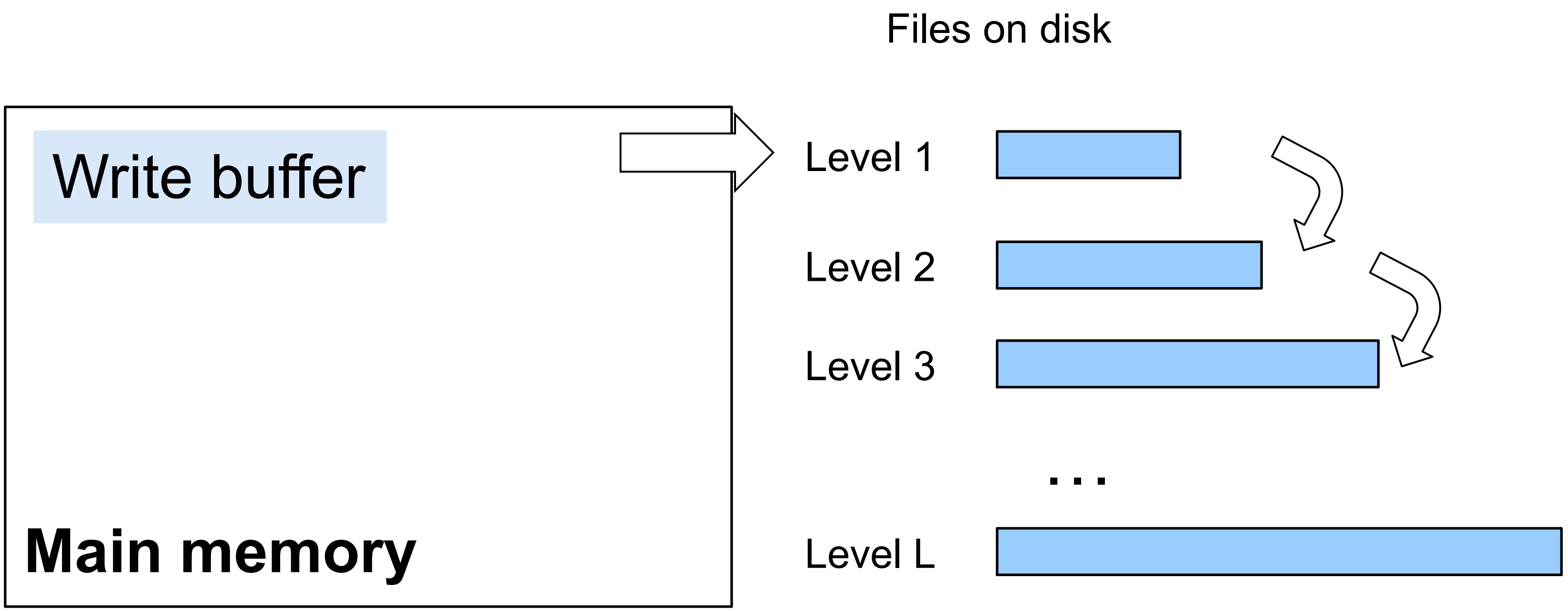
LSM Trees



LSM Trees

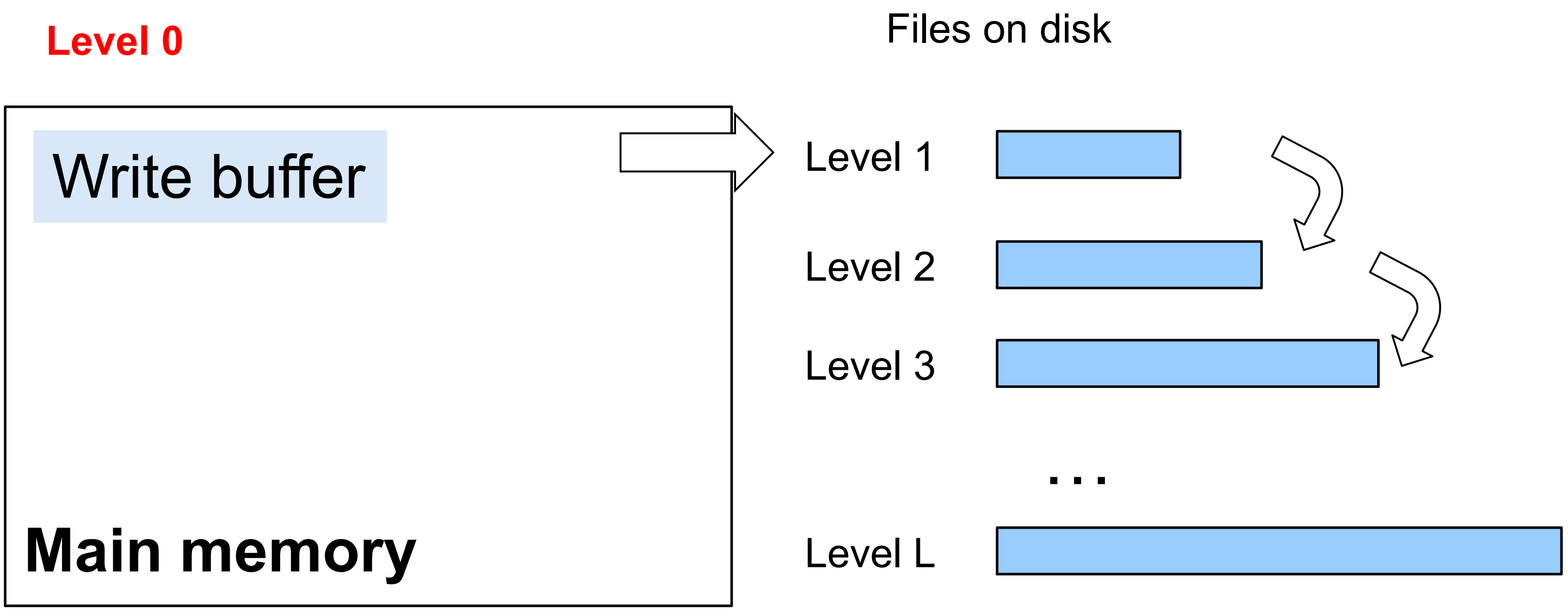

Files on disk
Write buffer
Level 1
Level 2
Level 3
Main memory

LSM Trees

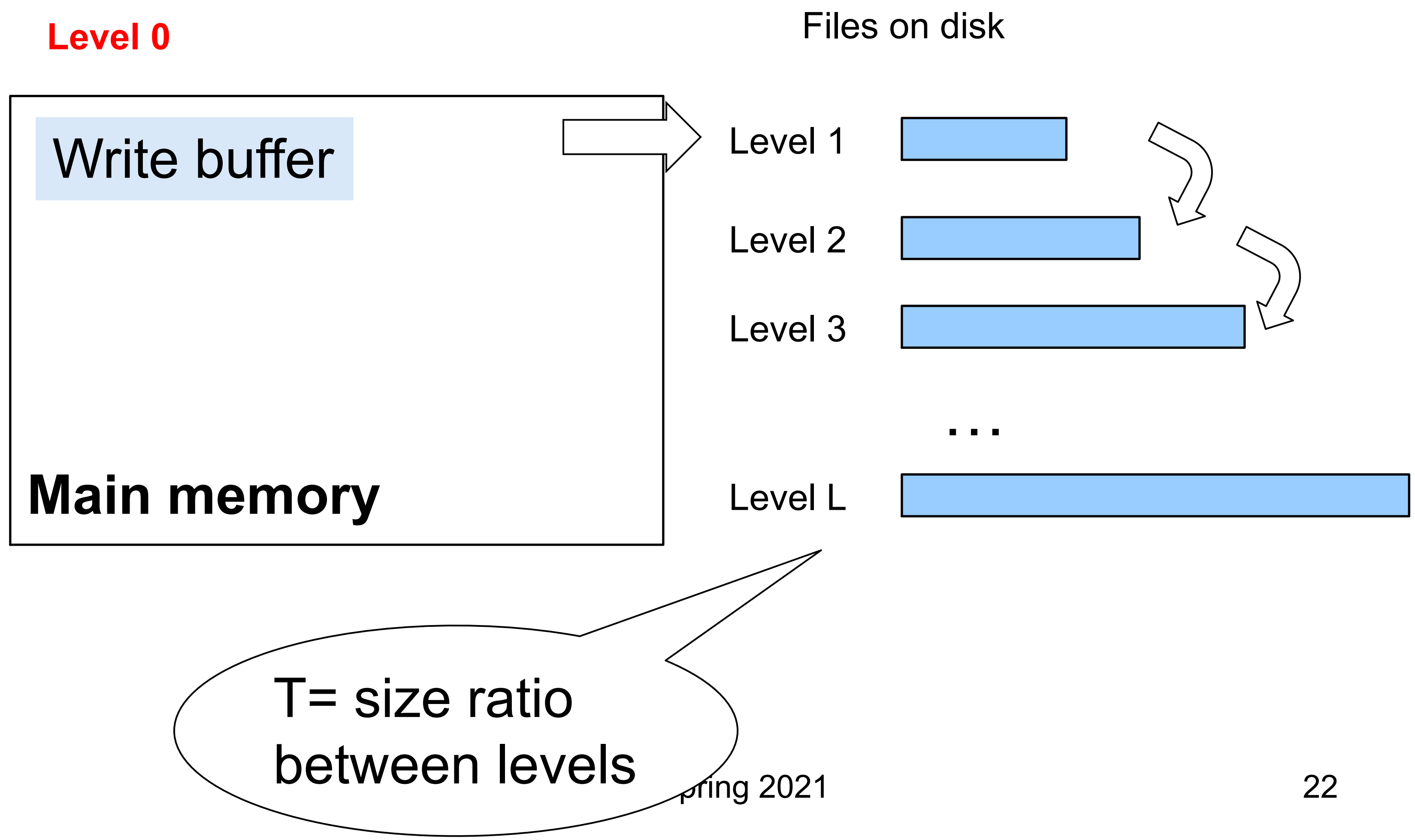


Files on disk
Write buffer
Main memory
Level 1
Level 2
Level 3
. . .
Level L

LSM Trees


Level 0
Files on disk
Write buffer
Level 1
Level 2
Level 3
...
Level L
Main memory

LSM Trees


Level 0
Files on disk
Write buffer
Level 1
Level 2
Level 3
...
Level L
Main memory
T= size ratio
between levels

Discussion

- Spilling to next level is a bulk operation; inserts a large number of values
- Better amortized cost than inserting those values one by one into a B+ tree
- Typically done by offline process

Read

- To read a key, we need to search it at all levels
- Cost is worse than B+ tree
- Three ideas to speedup reads (next)

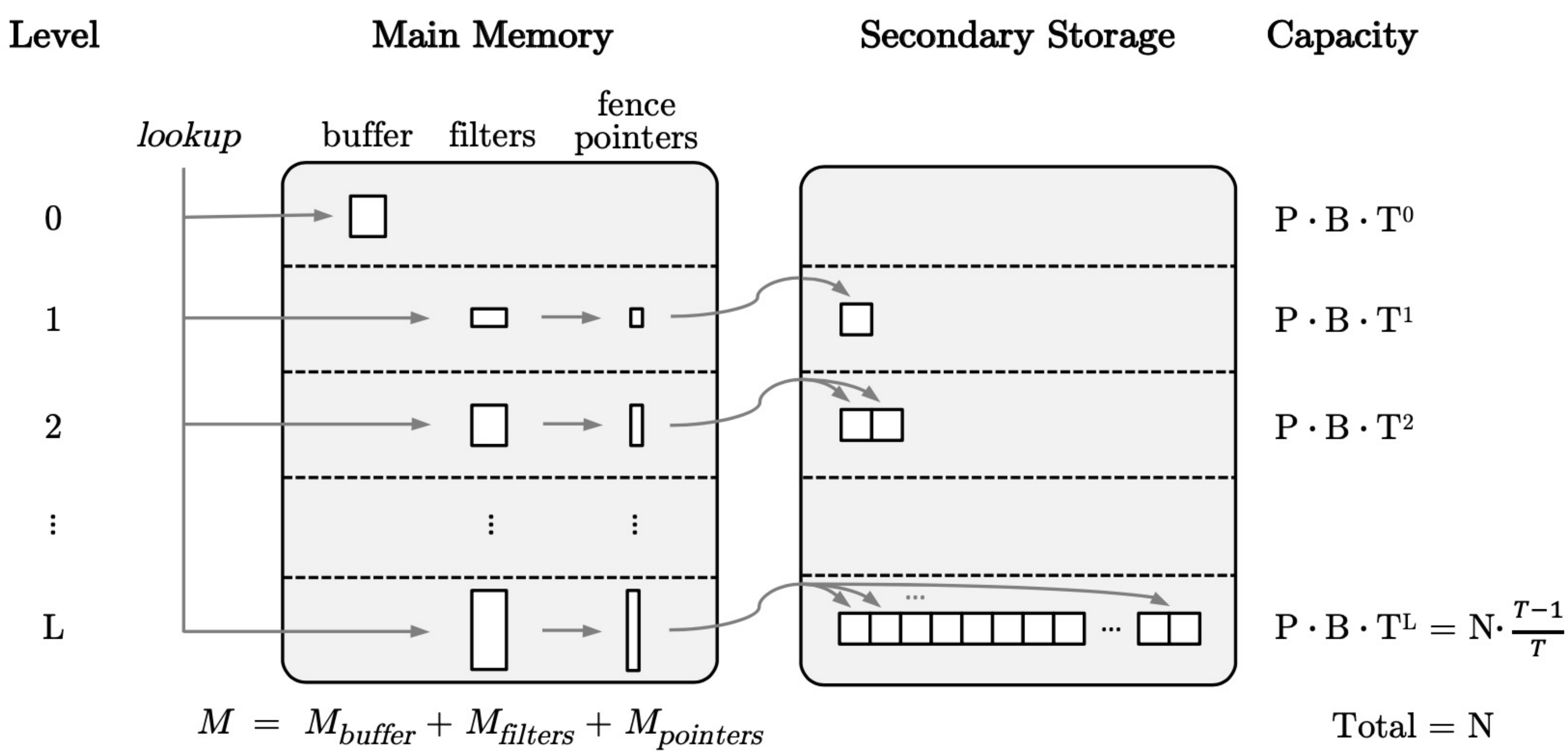
Three Main Ideas for Reads

1. Bloom filter for each level

2. Fenceposts in main memory for each level

3. Read single block for each level, do binary search

Reading



Level
Main Memory
Secondary Storage
Capacity
lookup
buffer
filters
fence pointers
0
1
2
⋮
L
$P \cdot B \cdot T^0$
$P \cdot B \cdot T^1$
$P \cdot B \cdot T^2$
$P \cdot B \cdot T^L = N \cdot \frac{T-1}{T}$
$M = M_{buffer} + M_{filters} + M_{pointers}$
Total = N

Updates, Deletes

- Never!
- Instead, invalidate the record, and insert a new record if update

Next

- How do we optimize the main memory:
  - Write buffer
  - Bloom filters
  - Fence pointers
- Merge policy
  - Tiering or
  - Leveling

FPR= $e^{-\frac{m}{n}(\ln^2 2)}$, n= #items at given level

Optimizing Bloom Filters

Most memory used by Bloom filters

- Common practice:
  - Ensure the same FPR for all levels
  - FPR constant, space m increases by factor T

FPR= $e^{-\frac{m}{n}(\ln^2 2)}$, n= #items at given level

Optimizing Bloom Filters

Most memory used by Bloom filters

- Common practice:
  - Ensure the same FPR for all levels
  - FPR constant, space m increases by factor T
- Paper observes:
  - Cost per level is the same: reading 1 block
  - Space increases but benefit is constant!
  - Keep space constant, FPR increases by factor T

Merge Policy

- Tiering (write optimized)
  - Flush main memory buffer sorted to disk
  - Accumulate multiple sorted files/level
  - When more than T sorted files: merge them and add 1 file to the next level

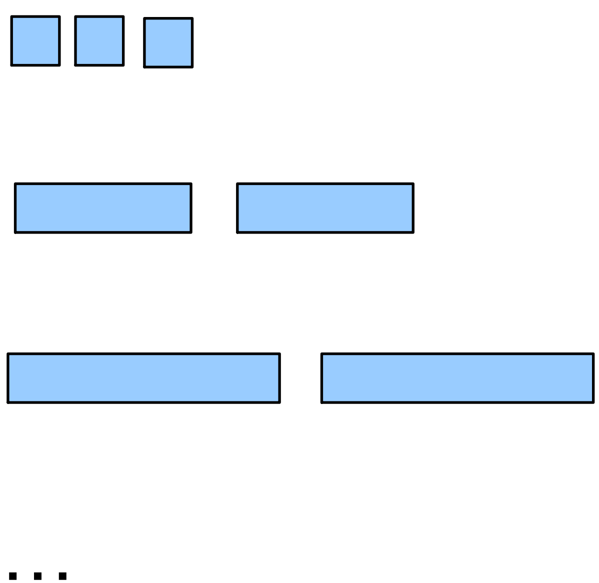
Merge Policy

- Tiering (write optimized)
  - Flush main memory buffer sorted to disk
  - Accumulate multiple sorted files/level
  - When more than T sorted files: merge them and add 1 file to the next level
- Leveling (read-optimized)
  - Merge-sort main memory with level 1
  - When a level becomes too large, move it to the next level by sorting

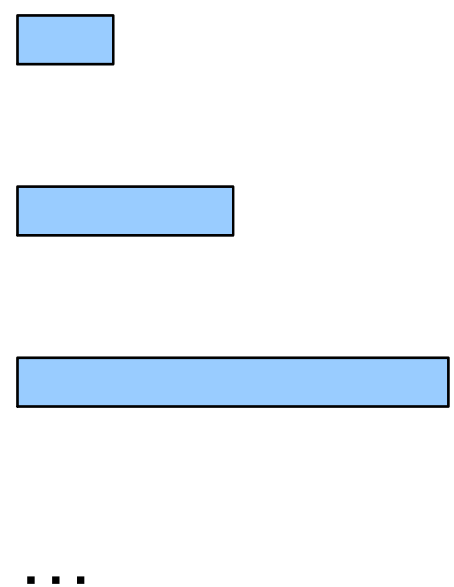
Size Ratio: T = 3

Merge Policies

Tiering



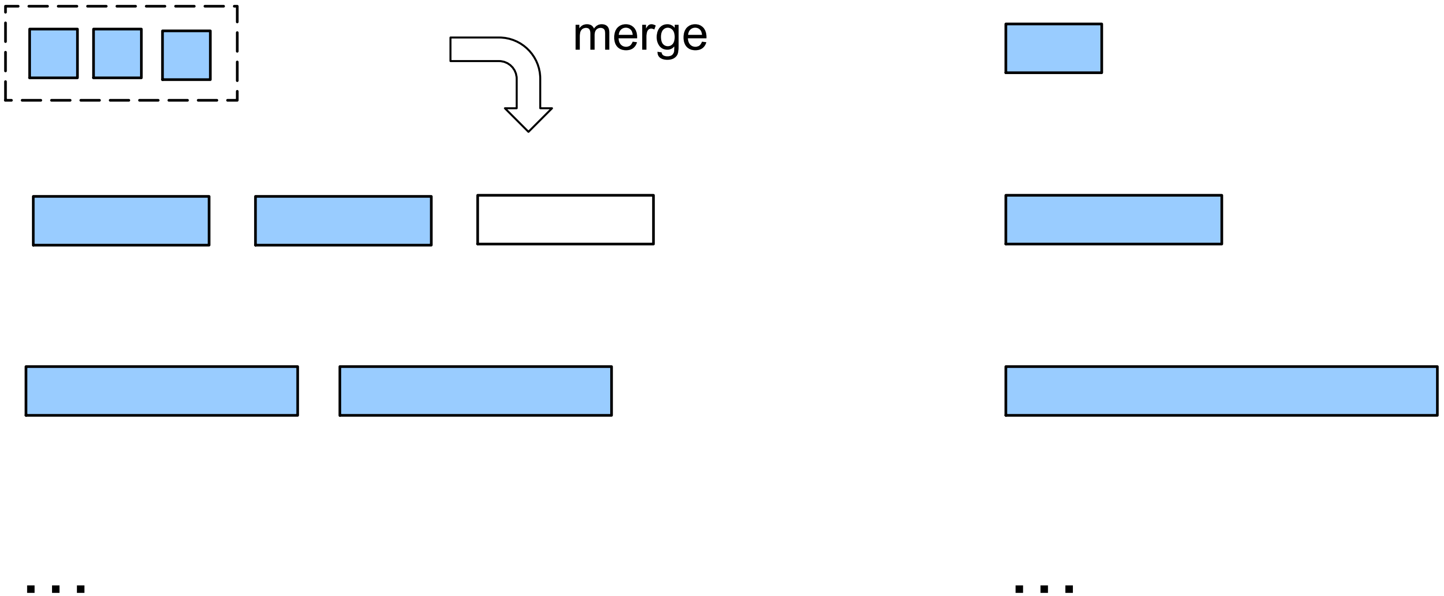
Leveling



Size Ratio: T = 3

Merge Policies

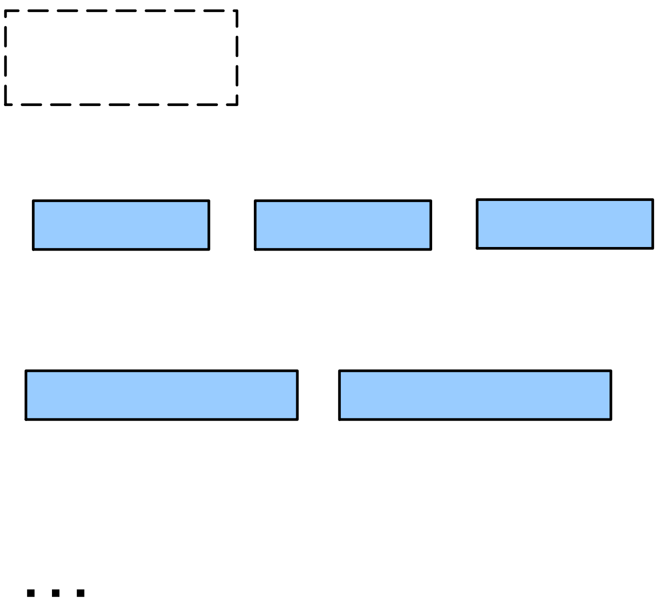
Tiering | Leveling



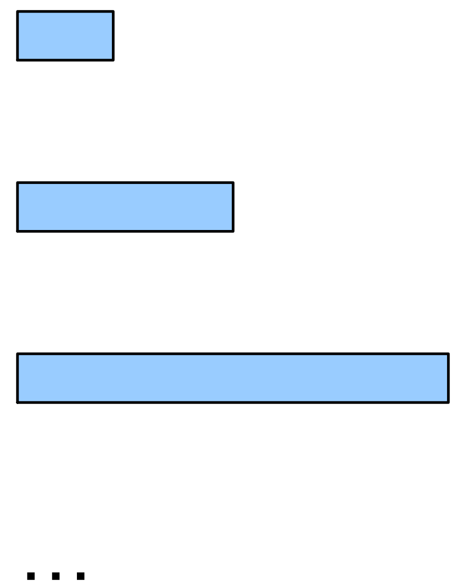
Size Ratio: T = 3

Merge Policies

Tiering

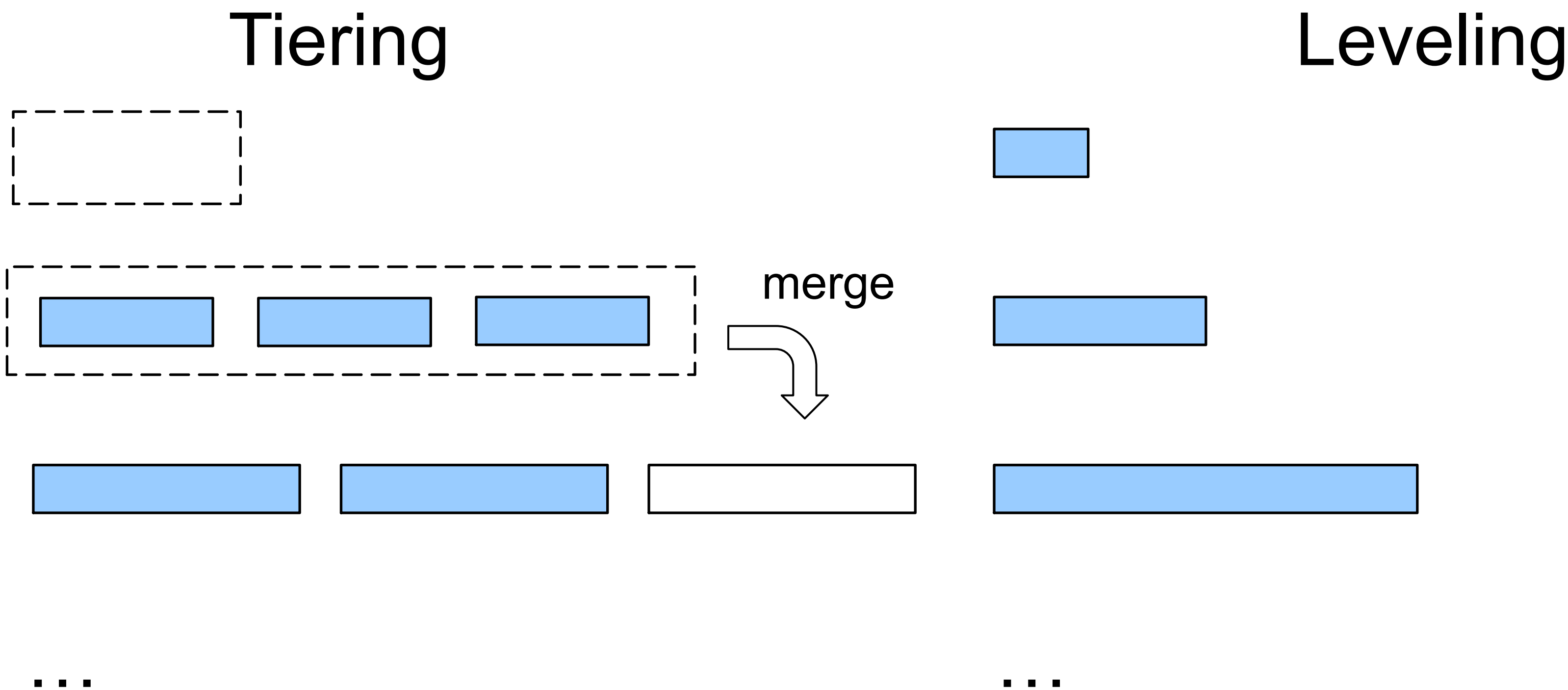


Leveling



Size Ratio: T = 3

Merge Policies

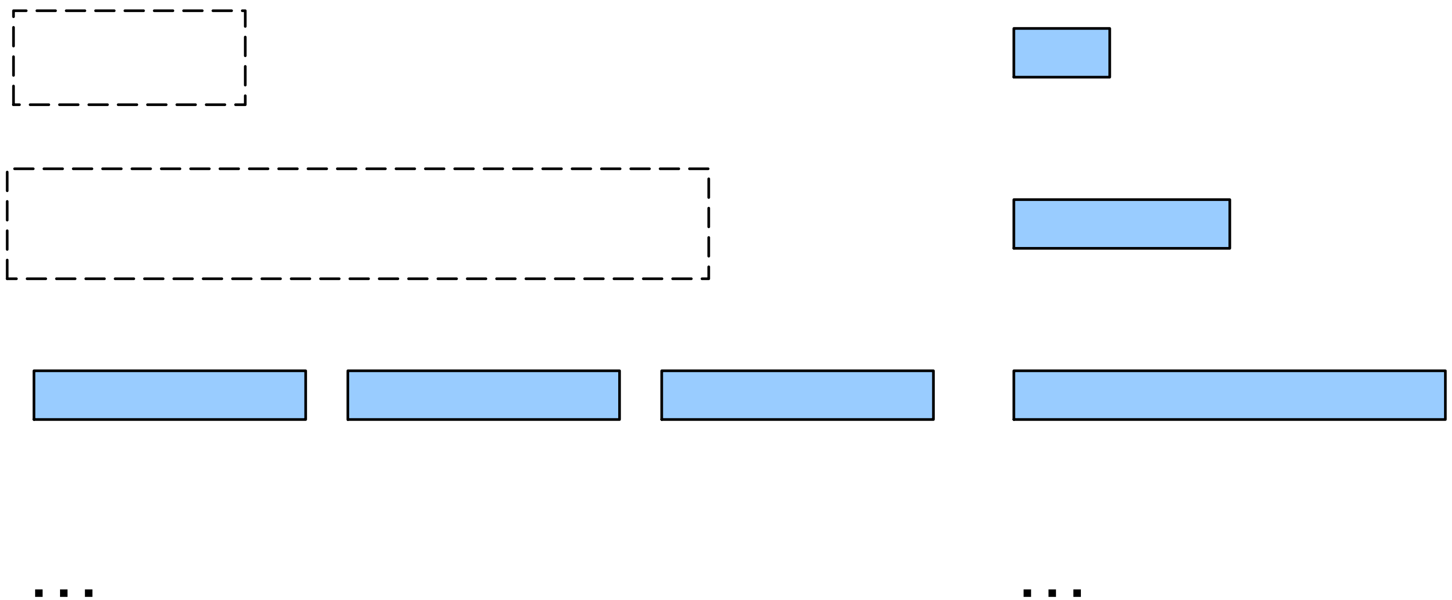


Size Ratio: T = 3

Merge Policies

Tiering

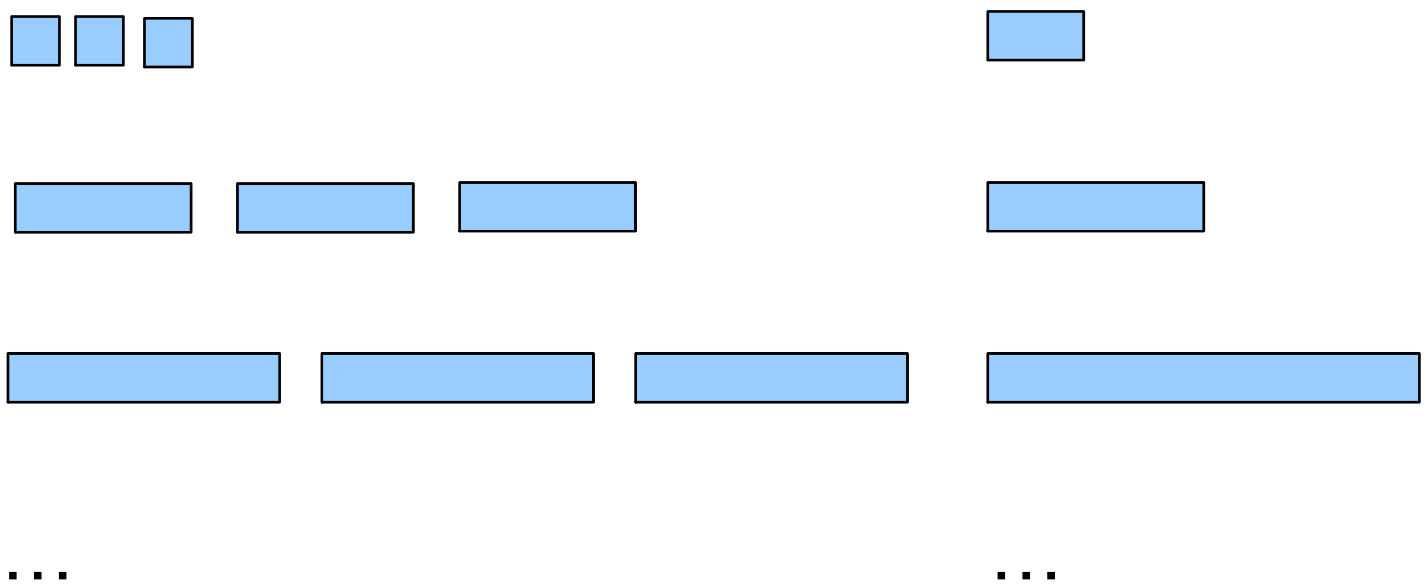
Leveling



Size Ratio: T = 3

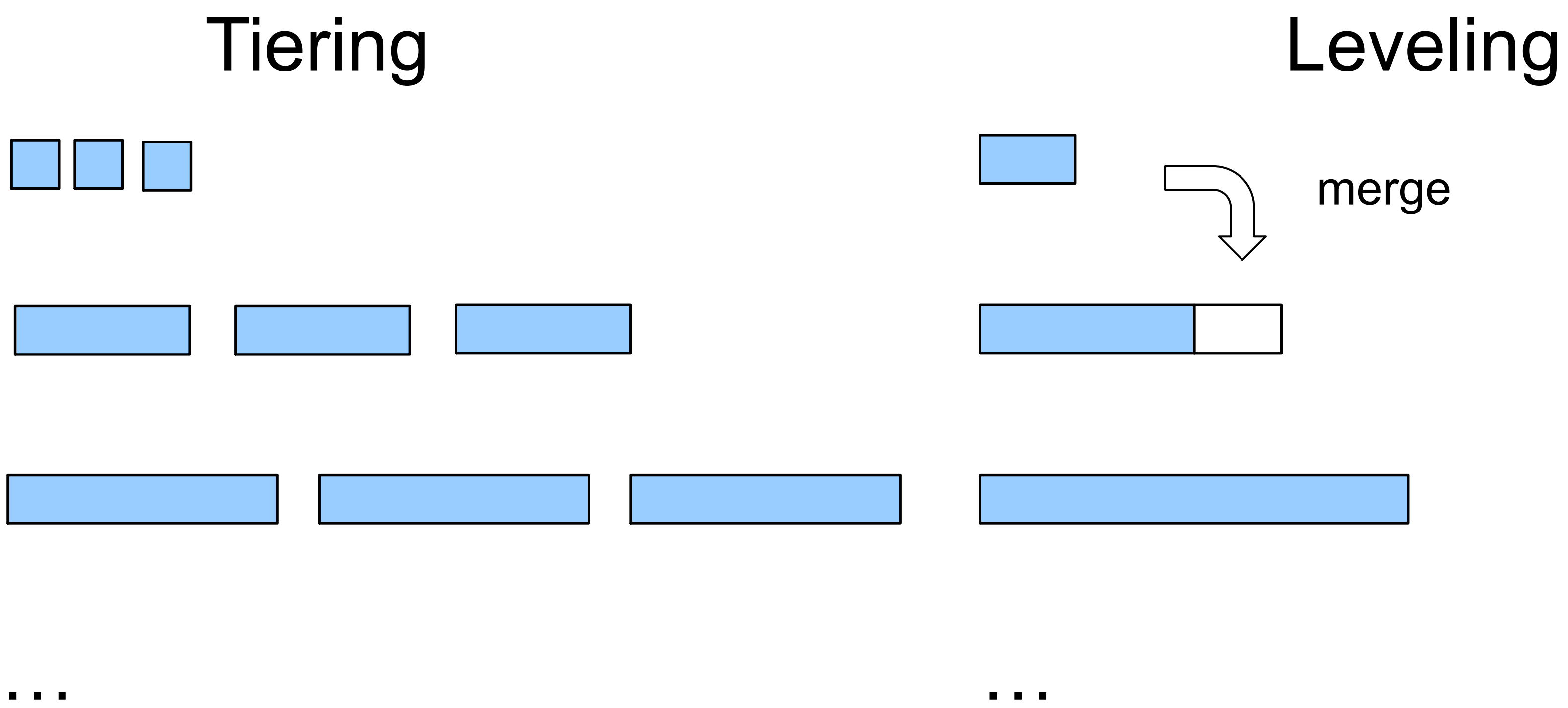
Merge Policies

Tiering | Leveling



Size Ratio: T = 3

Merge Policies

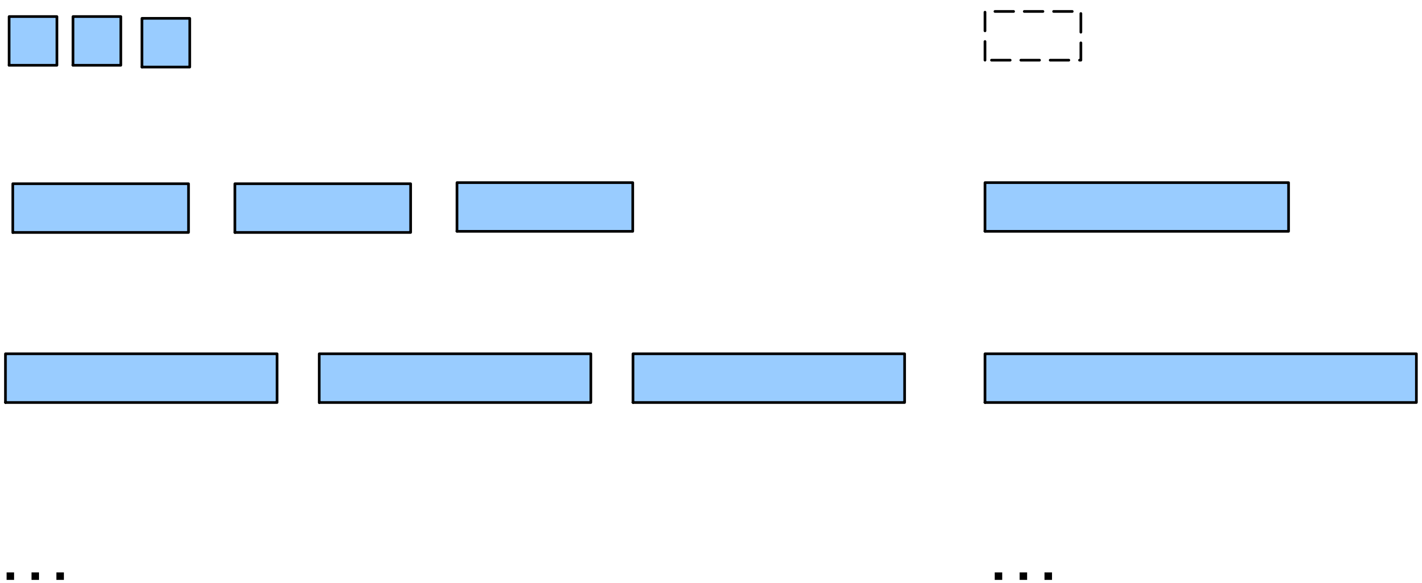


Size Ratio: T = 3

Merge Policies

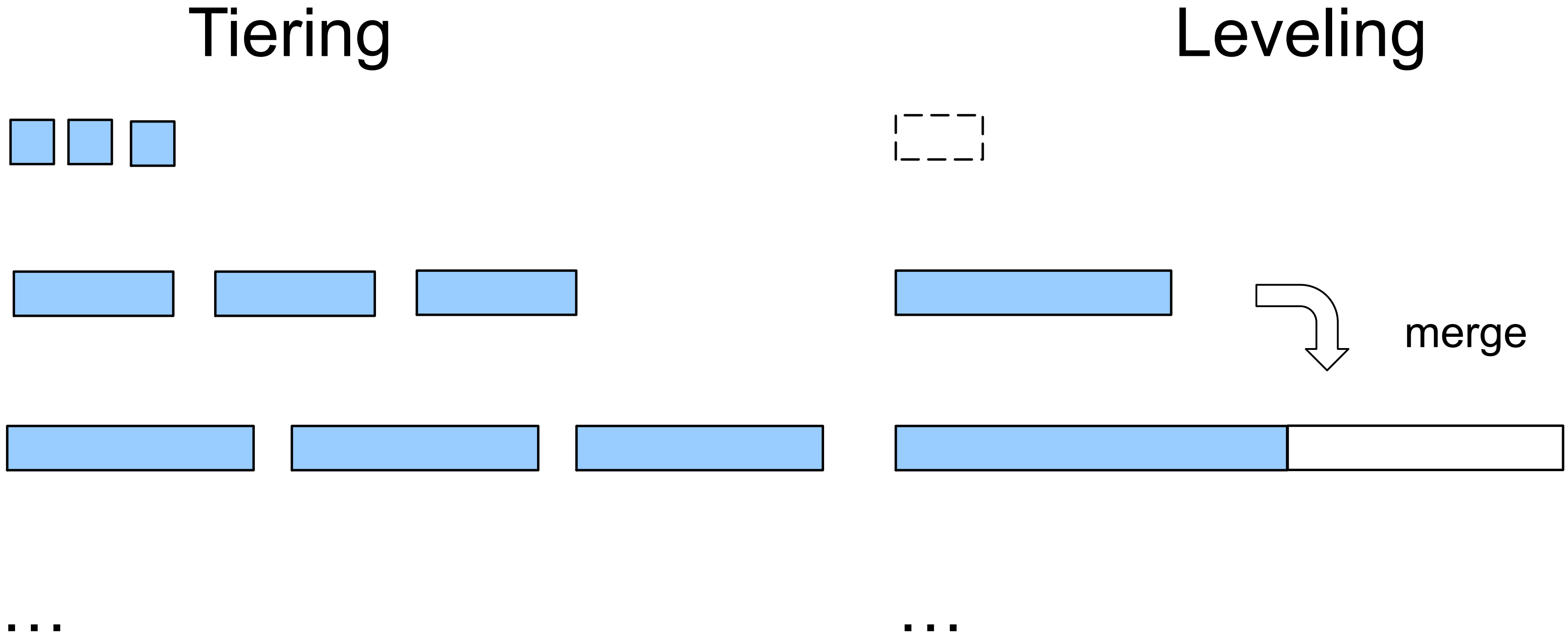
Tiering

Leveling



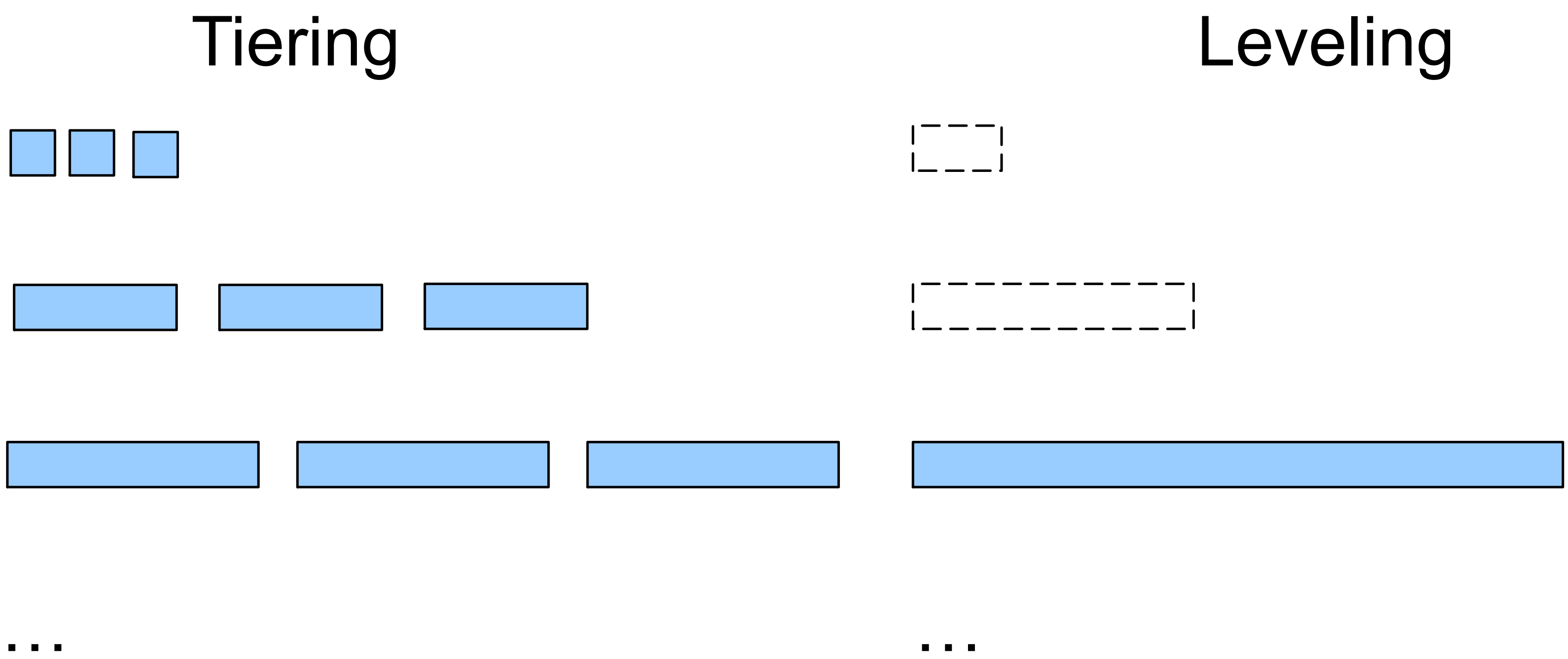
Size Ratio: T = 3

Merge Policies



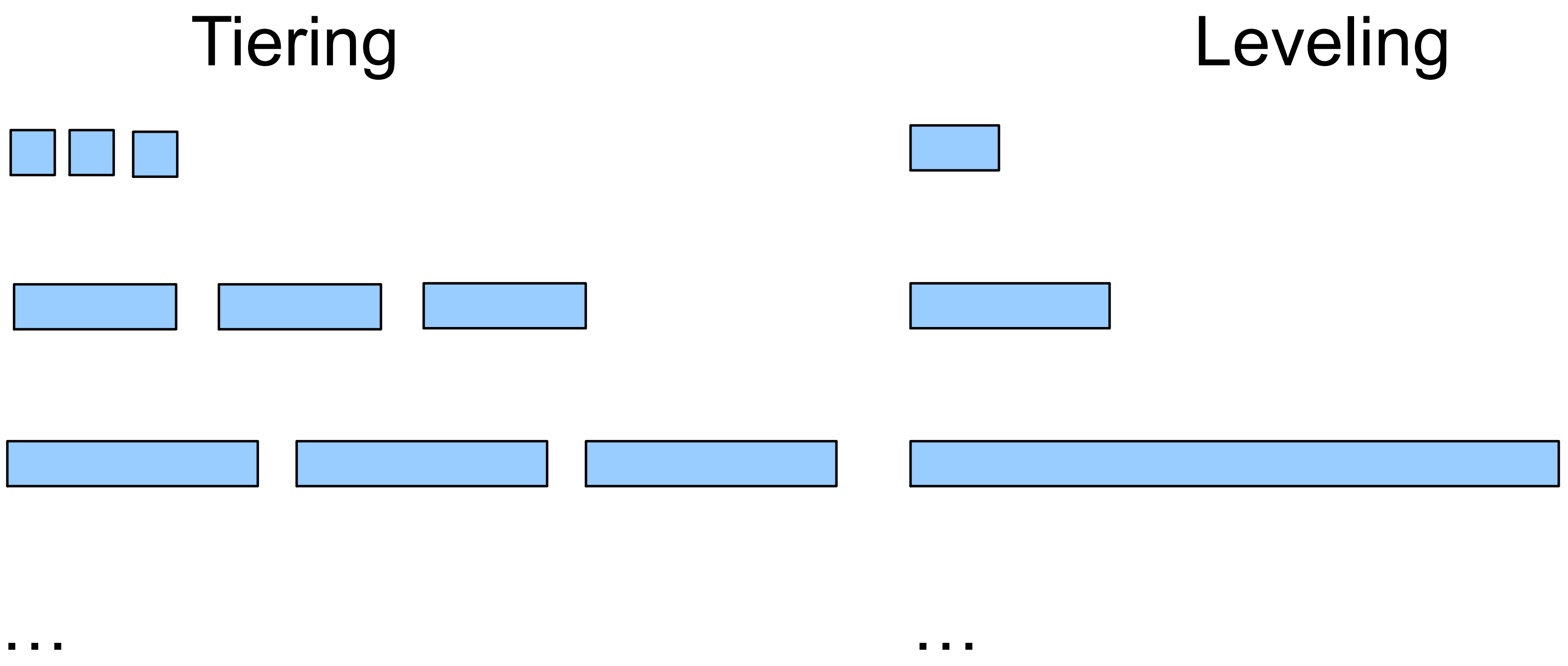
Size Ratio: T = 3

Merge Policies



Size Ratio: T = 3

Merge Policies



What happens when $T \rightarrow \infty$?

Merge Policies

Tiering

Leveling

Then L = 1

What happens when $T \rightarrow \infty$?

Merge Policies

Tiering | Leveling

A log file!

Then L = 1

What happens when $T \to \infty$?

Merge Policies

Tiering

Leveling

A log file!

A sorted file!

Then L = 1

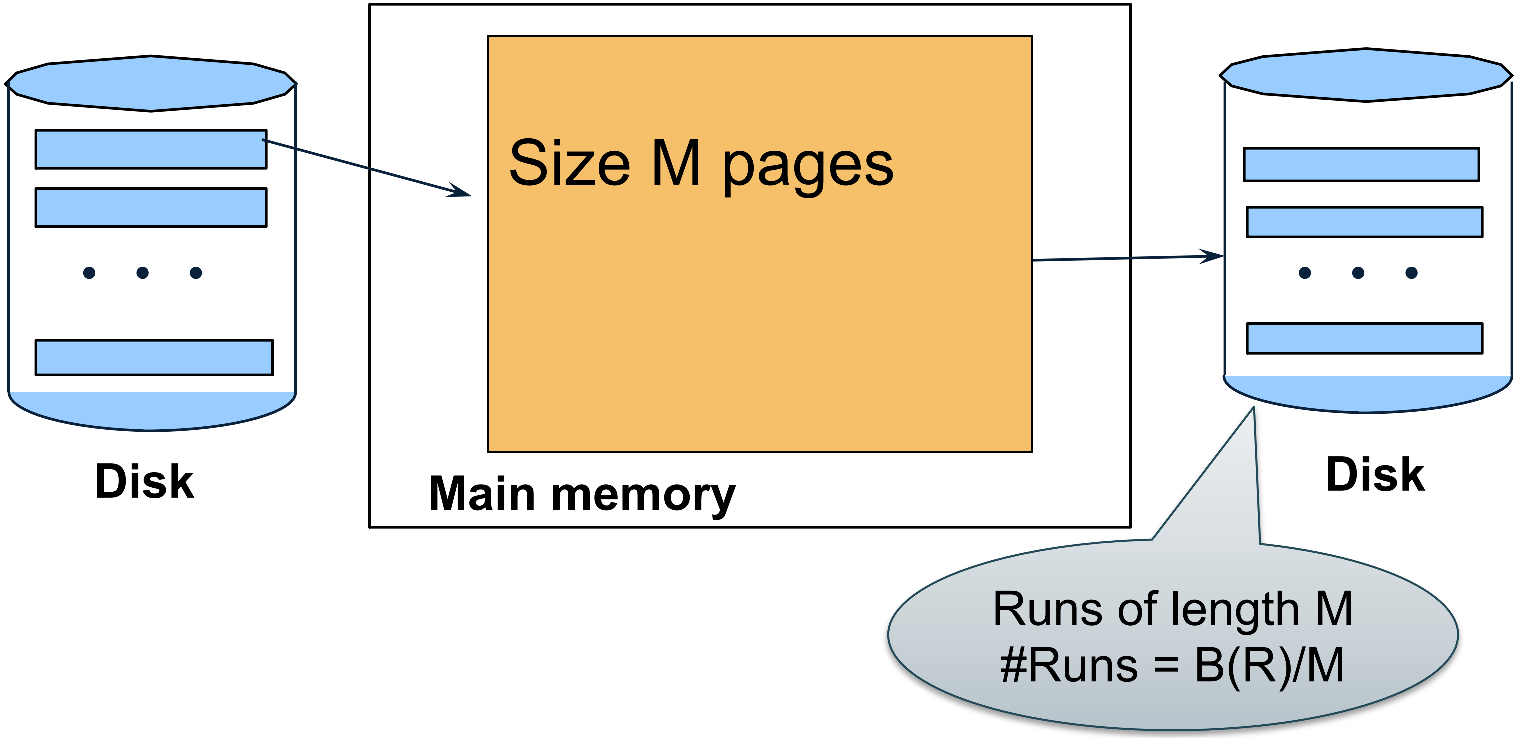
What happens when $T \rightarrow \infty$?

Recap: Merge-Sort

- Problem: Sort a file of size B with memory M
- Will discuss only 2-pass sorting, for when $B \leq M^2$

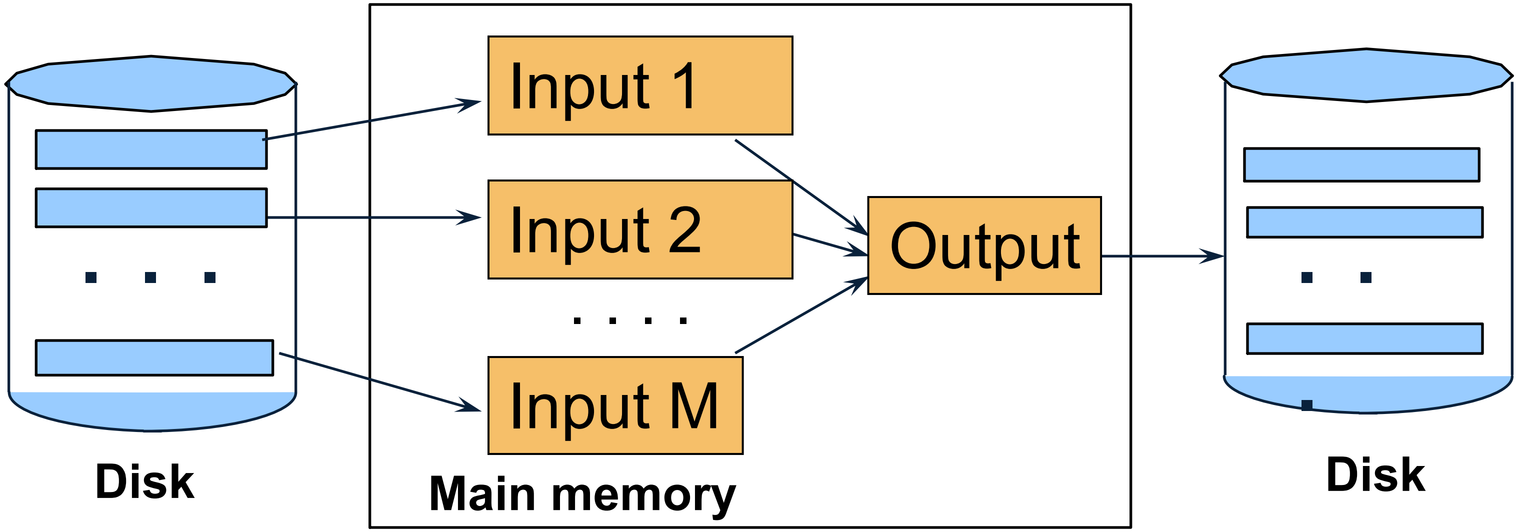
Merge-Sort: Step 1

- Phase one: load M pages in memory, sort



Merge-Sort: Step 2

- Merge M – 1 runs into a new run
- Result: runs of length M (M – 1) $\approx M^2$



Assuming $B \leq M^2$, we are done

Merge-Sort

- Cost:
  – Read+write+read = 3B(R)
  – Assumption: B(R) <= M^2

- Other considerations
  – In general, a lot of optimizations are possible

Summary

- LSM trees: optimized for write-intensive applications
- Three ideas for writes:
  - Memory buffer, spill to disk, multiple levels
- Three ideas for reads:
  - Bloom filters, fence posts, binary search
- When T is very large: log or sorted file