Dostoevsky: Better Space-Time Trade-Offs For LSM-Tree Based Key-Value Stores via **Adaptive Removal of Superfluous Merging**

DASlab @ Harvard SEAS

Niv Dayan &
Stratos Idreos

LSM-Tree

Key-Value Stores

LSM-Tree

Key-Value Stores

Time-Series Databases

Key-Value Stores

LSM-Tree

Relational Databases

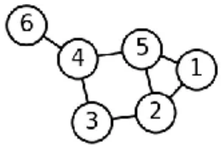
SQLite4

MyRocks

Time-Series Databases

quasardb

InfluxDB



Algorithm Design

RocksDB BigTable riak HBASE

DynamoDB accumulo cassandra levelDB

Key-Value Stores

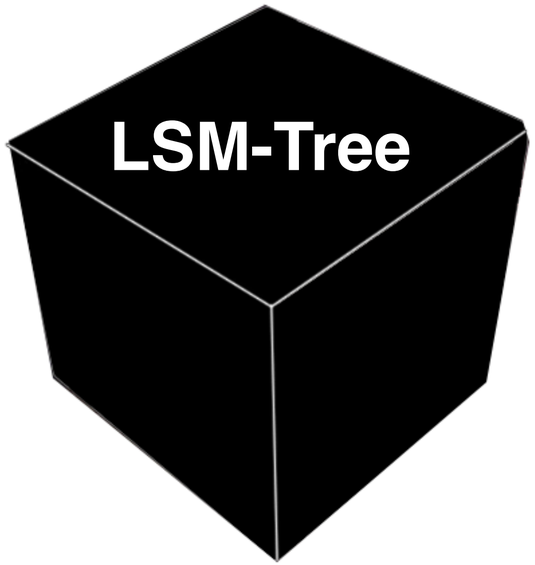
LSM-Tree

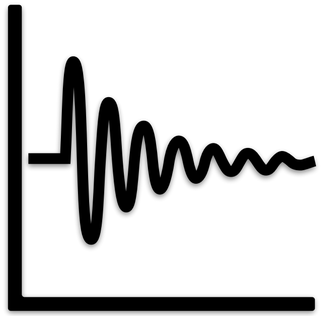
Relational Databases

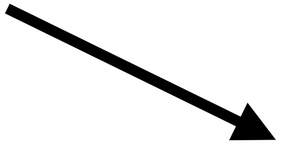
SQLite4 MyRocks

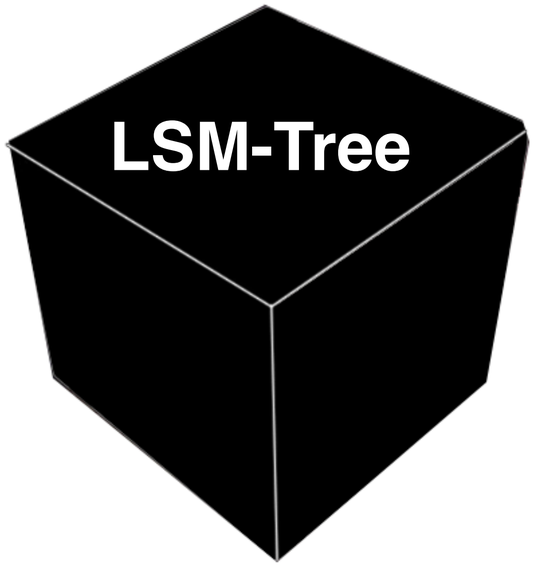
Time-Series Databases

quasardb InfluxDB

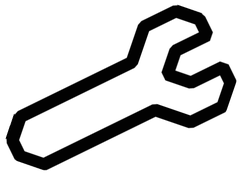

LSM-Tree

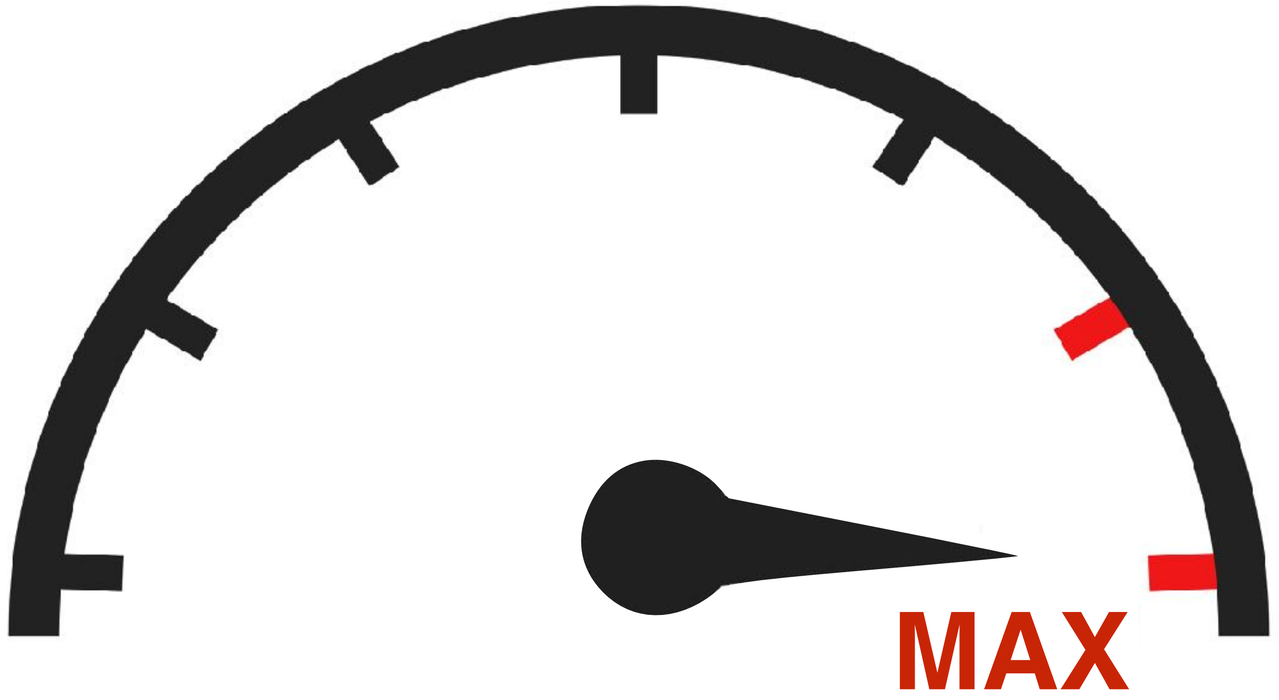

workload



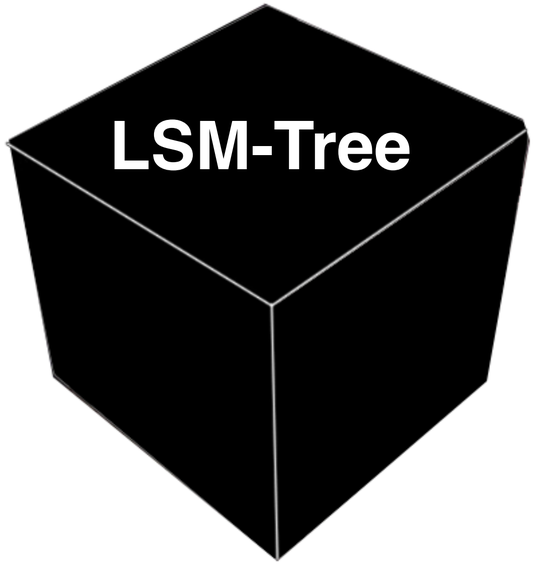

LSM-Tree


MAX

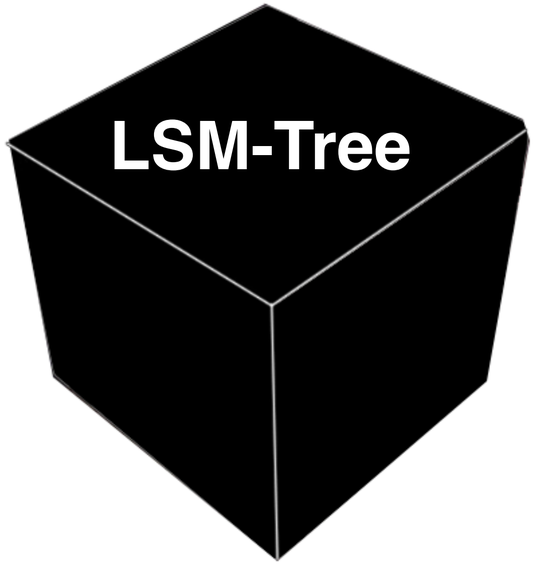



MAX


stuck here
MAX



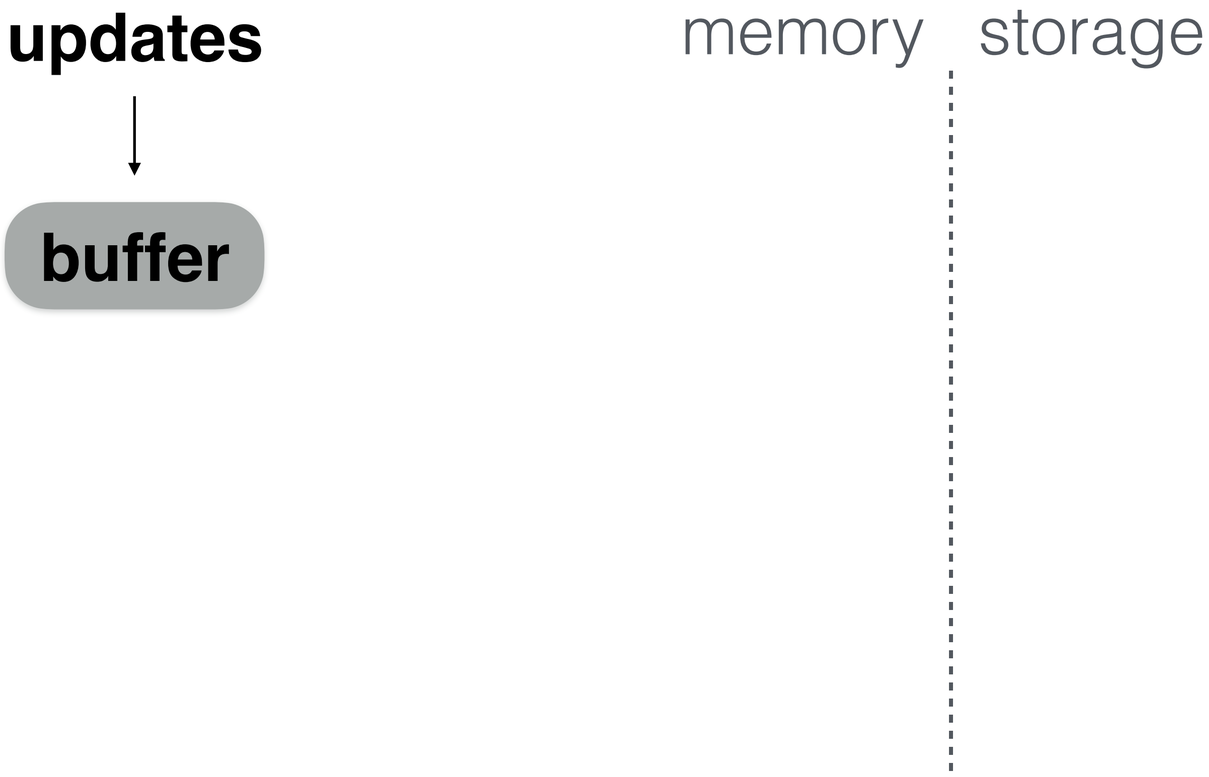
suboptimal design

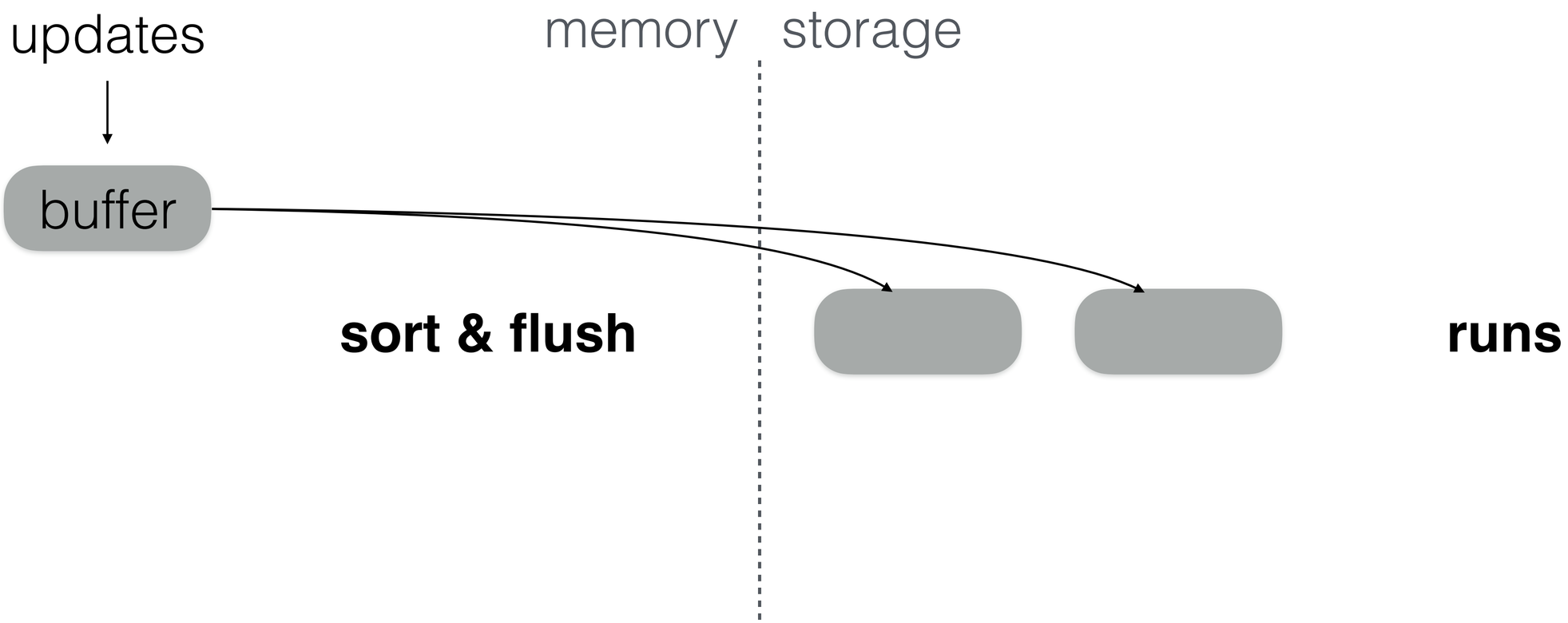


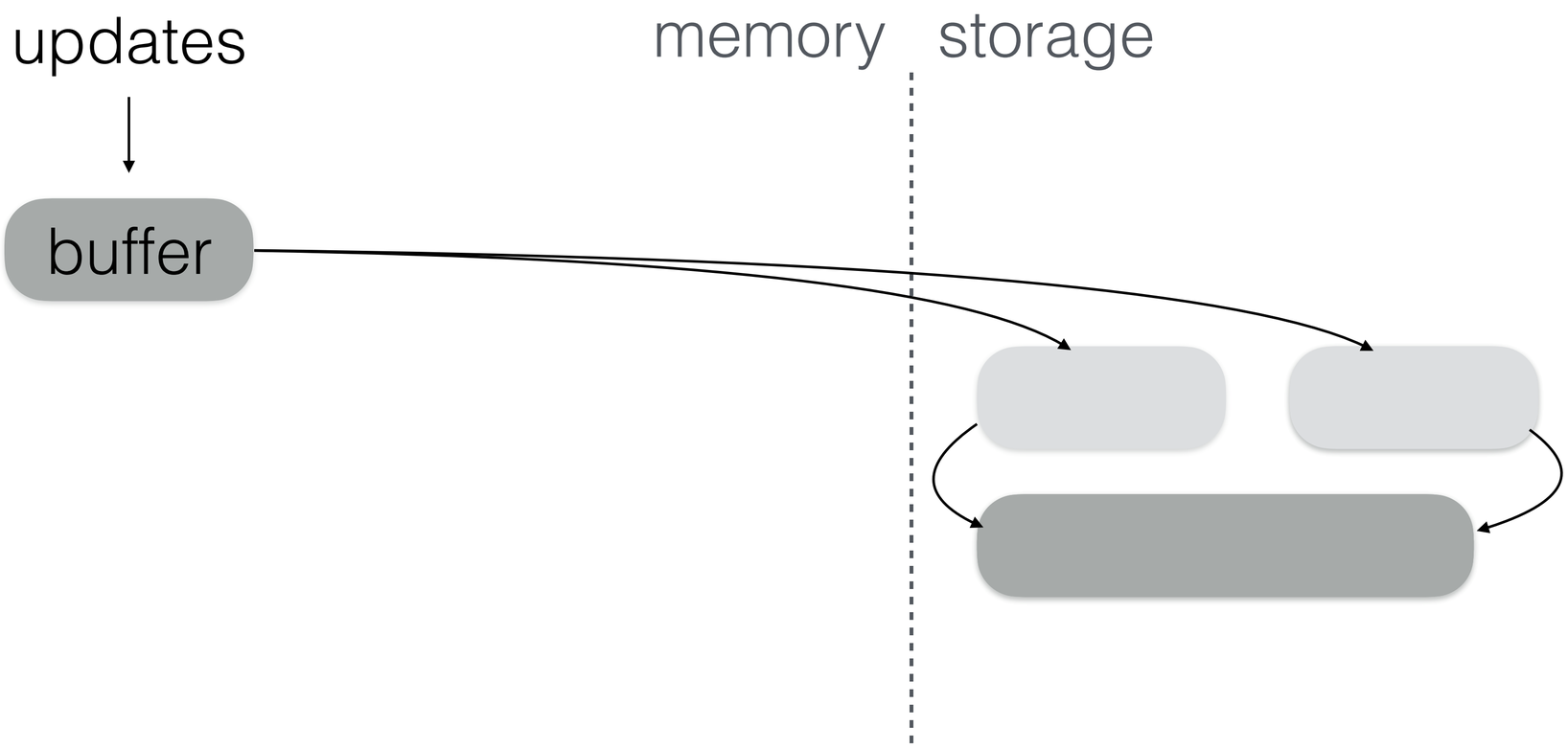
suboptimal design

why? how?

LSM-Tree
Background

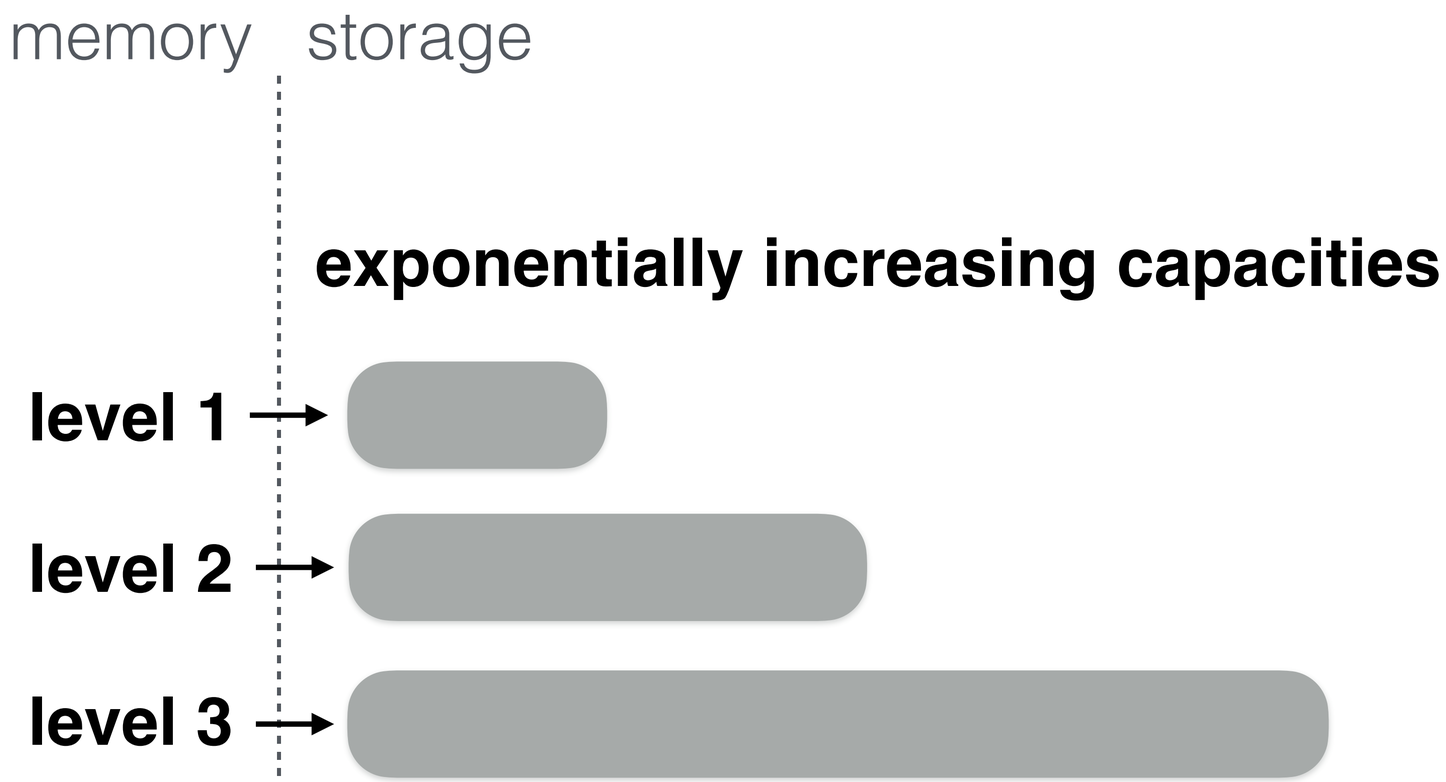

updates
buffer
memory
storage

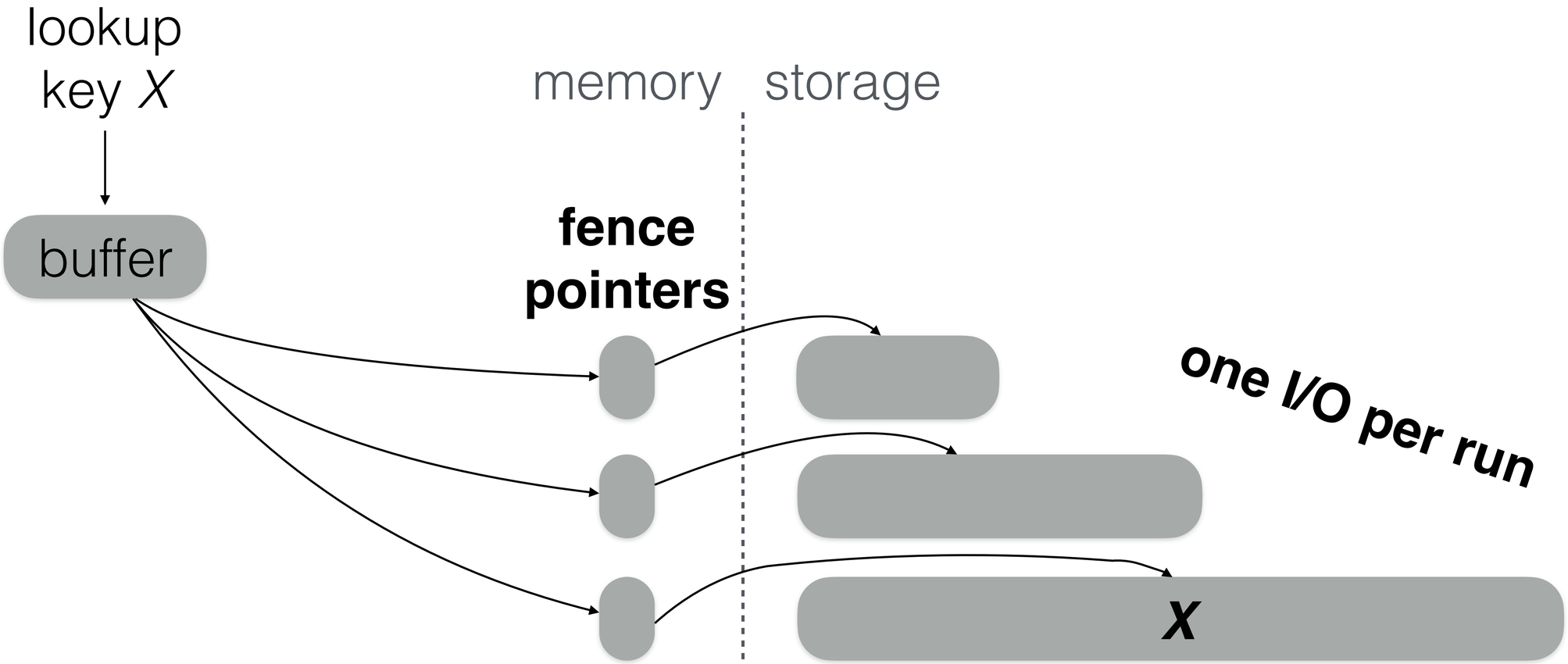

updates
memory
storage
buffer
sort & flush
runs

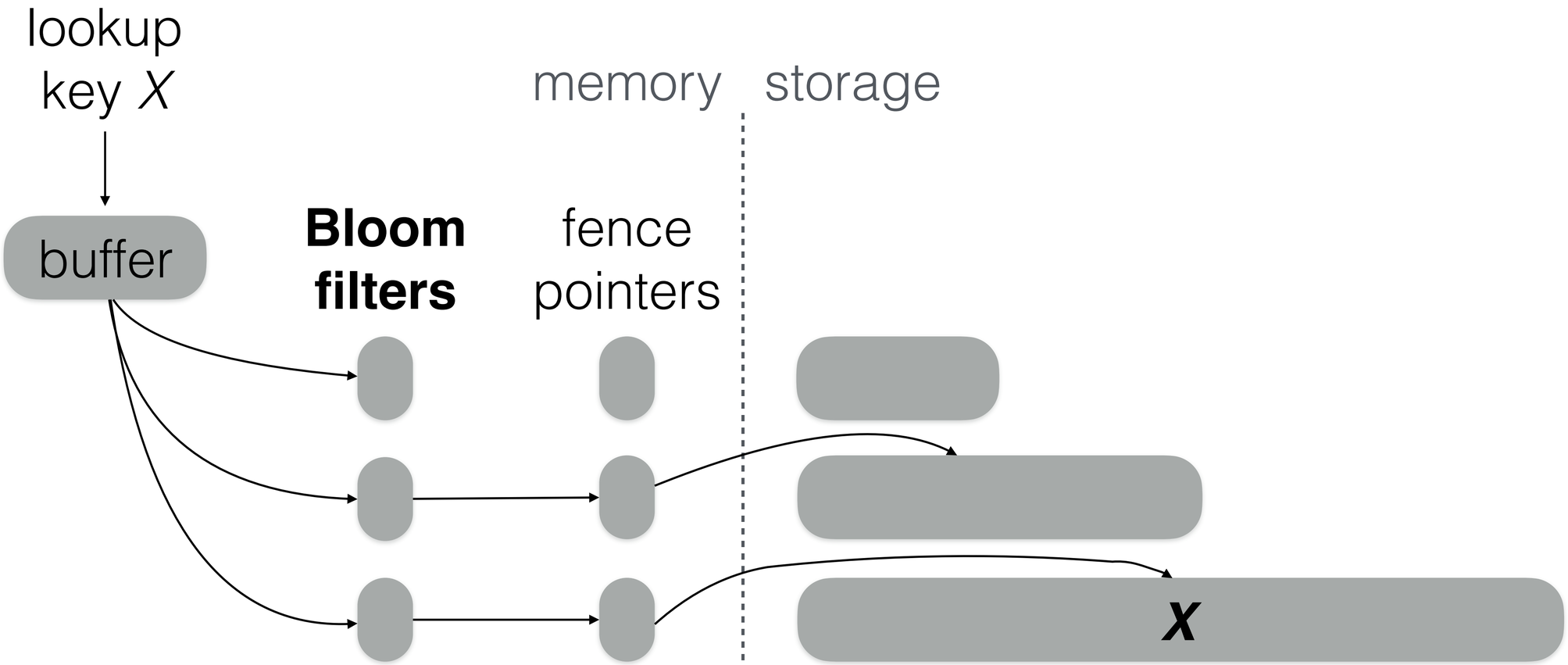


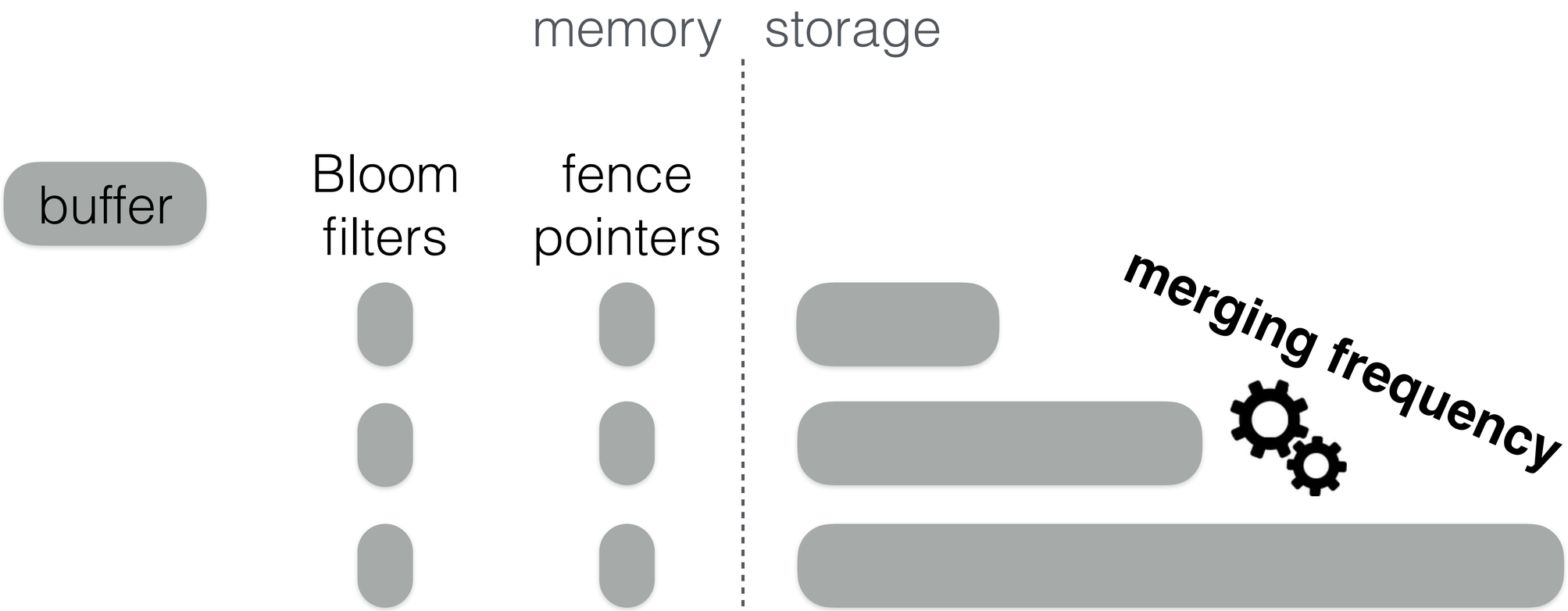
sort merge

buffer


memory
storage
exponentially increasing capacities
level 1
level 2
level 3


lookup
key X
buffer
memory
storage
fence
pointers
one I/O per run
X


lookup
key X
buffer
Bloom
filters
fence
pointers
memory
storage
X


memory
storage
buffer
Bloom filters
fence pointers
merging frequency

merging frequency

merge cost

lookup cost

merging frequency ↑

merge cost ↑

↓ lookup cost

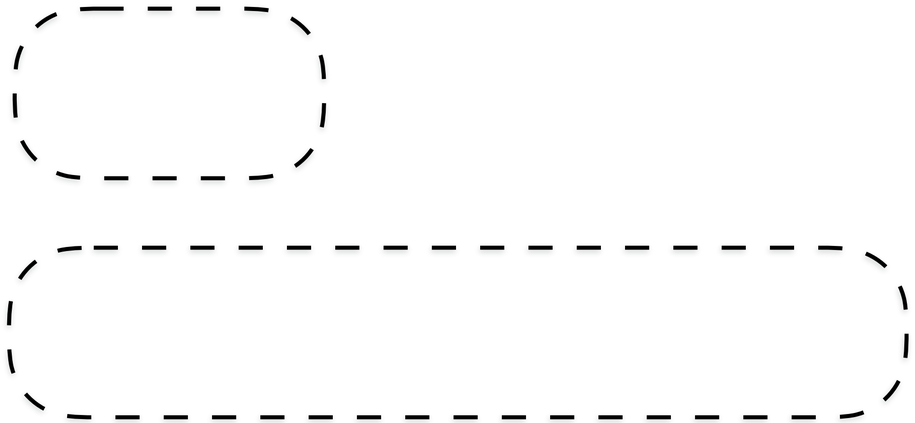
merging frequency

Tiering
merge-optimized

Leveling
lookup-optimized

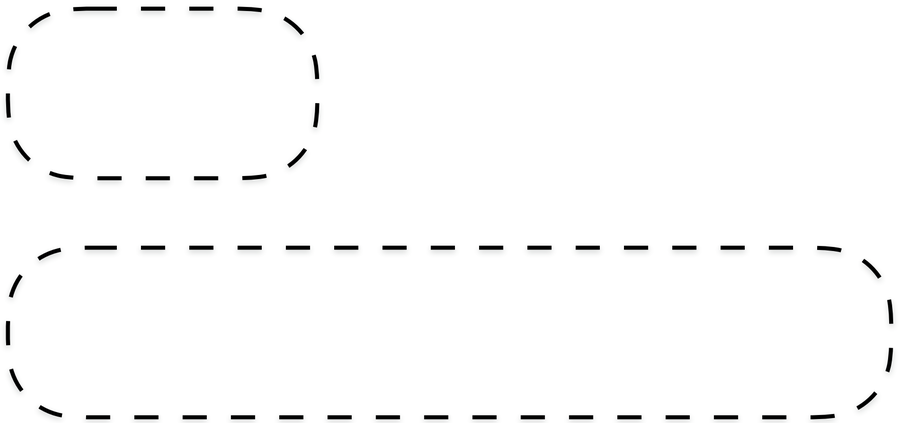
Tiering

merge-optimized



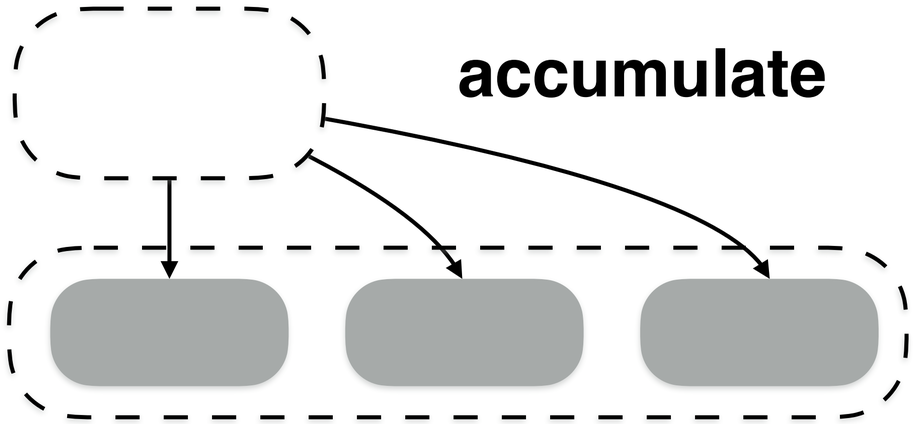
Leveling

lookup-optimized



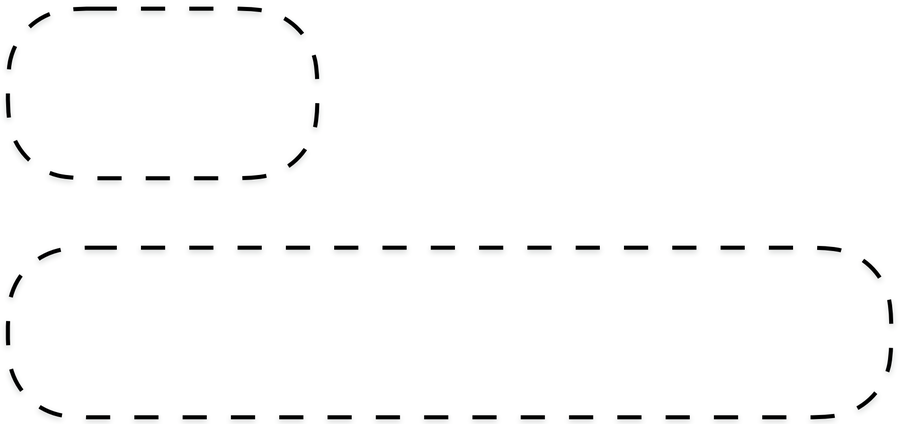
Tiering

merge-optimized



Leveling

lookup-optimized

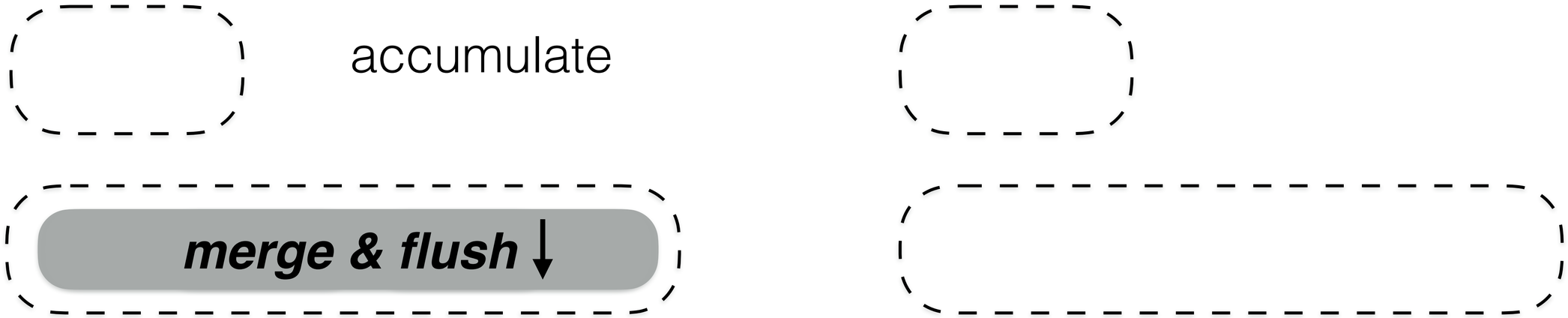


Tiering

merge-optimized

Leveling

lookup-optimized

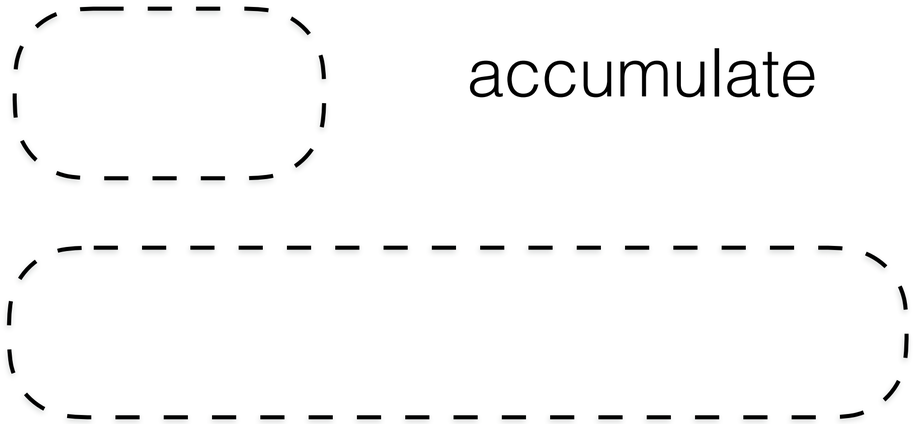


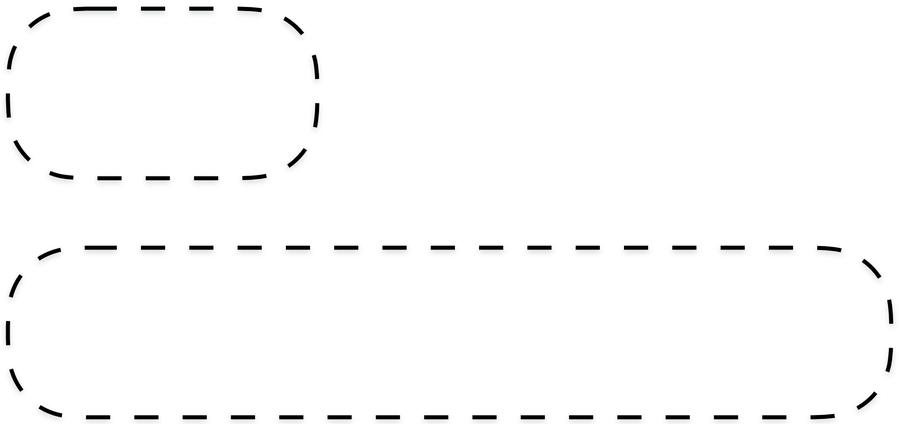
Tiering

merge-optimized

Leveling

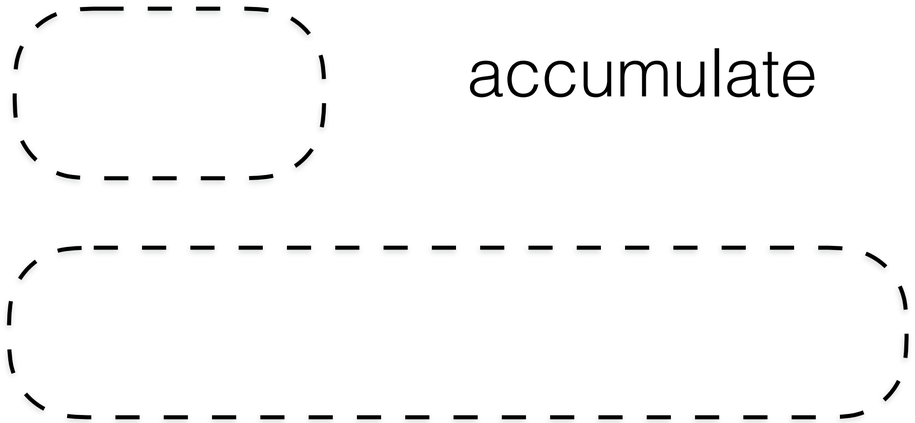
lookup-optimized





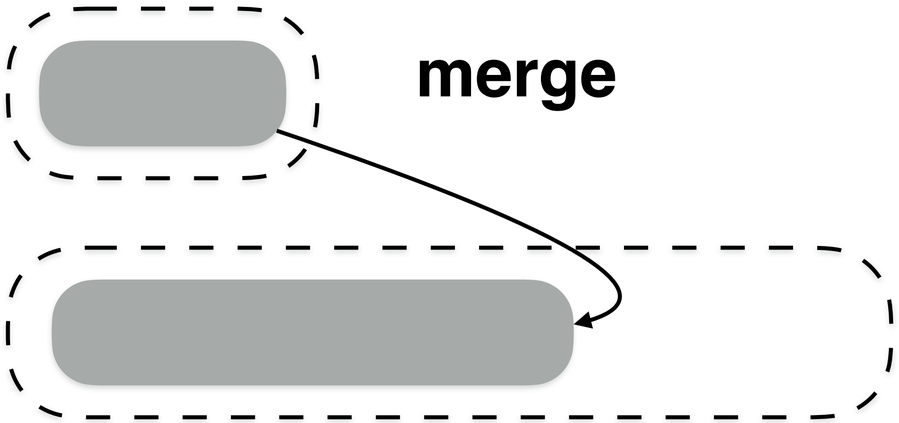
Tiering

merge-optimized



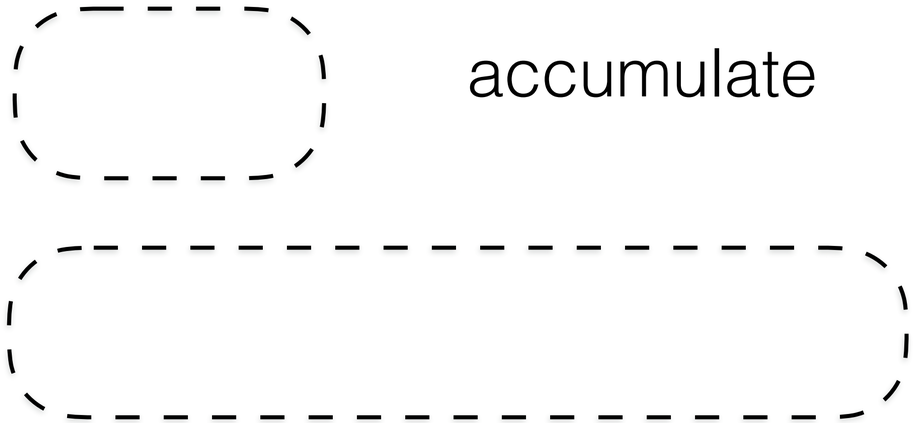
Leveling

lookup-optimized



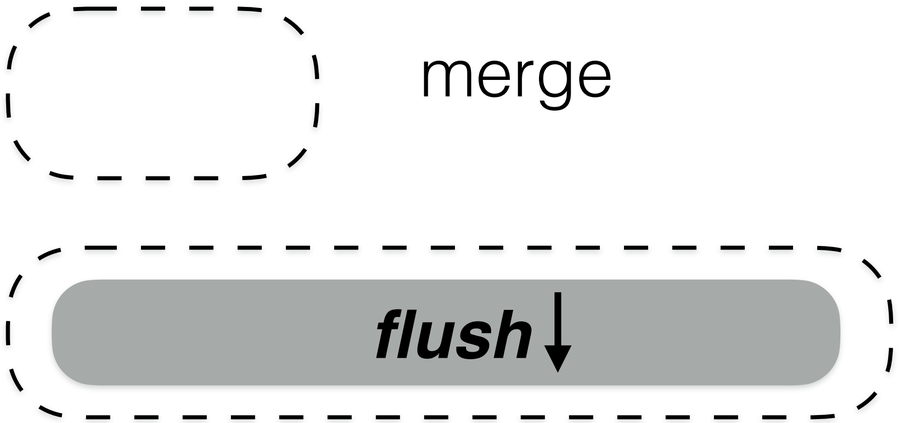
Tiering

merge-optimized



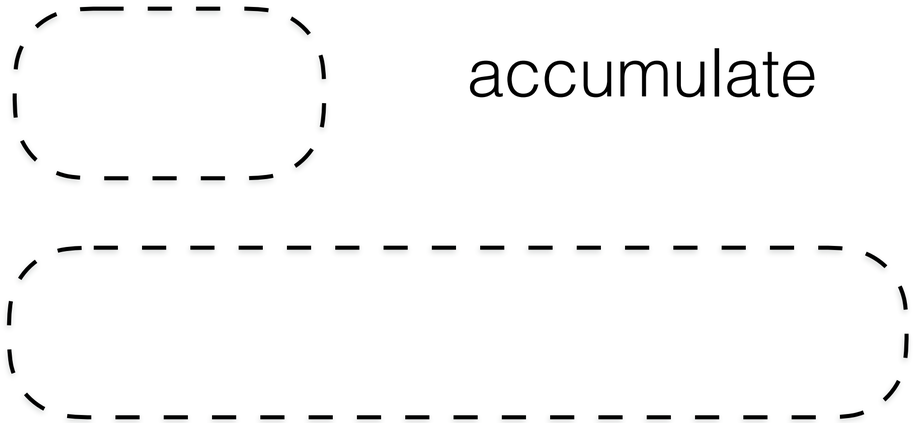
Leveling

lookup-optimized



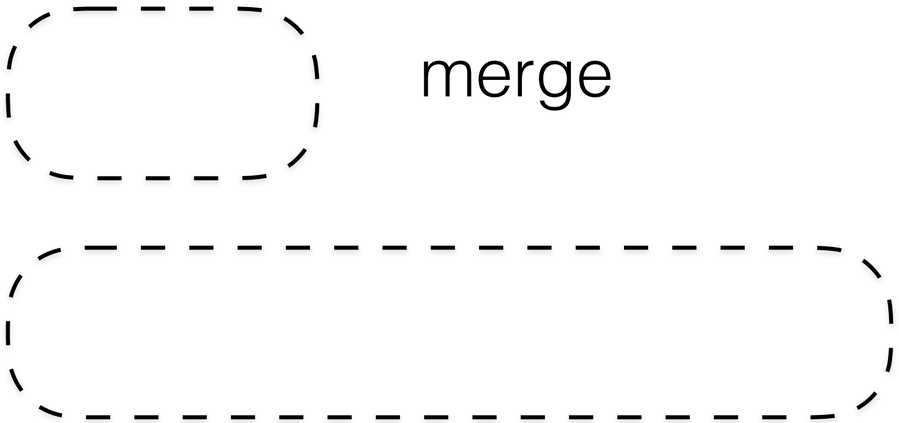
Tiering

merge-optimized



Leveling

lookup-optimized



Tiering

merge-optimized

Leveling

lookup-optimized

$\log_R(N)$

Tiering

merge-optimized

Leveling

lookup-optimized

size ratio

Tiering

merge-optimized

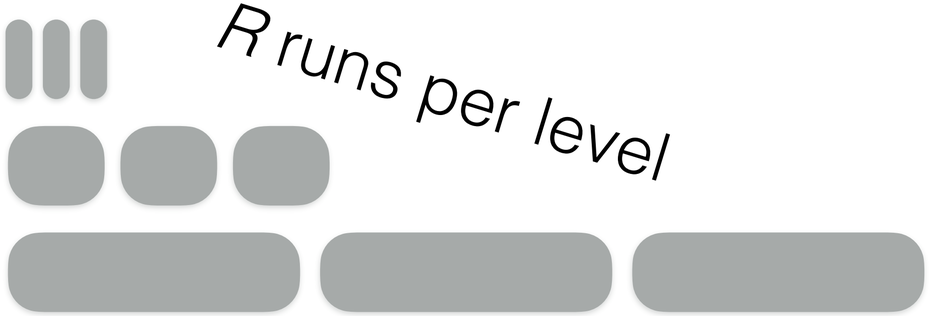
Leveling

lookup-optimized

size ratio

Tiering

merge-optimized



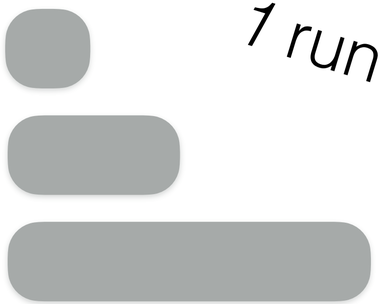
Leveling

lookup-optimized

size ratio *R*

Tiering

merge-optimized



1 run per level

Leveling

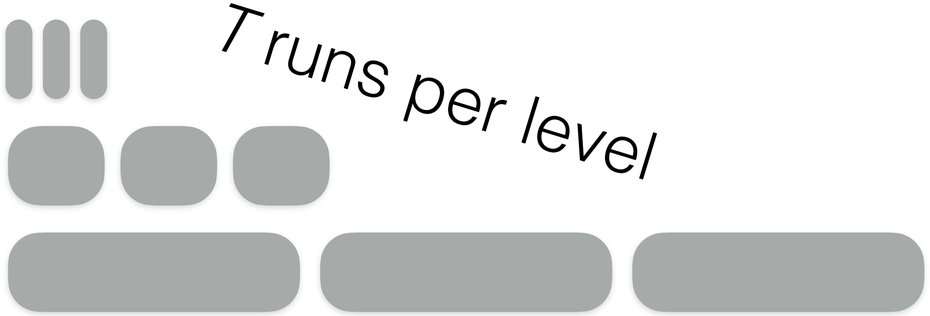
lookup-optimized

1 run per level

size ratio *R*

Tiering

merge-optimized



Leveling

lookup-optimized

size ratio *R*

Tiering

merge-optimized

Leveling

lookup-optimized

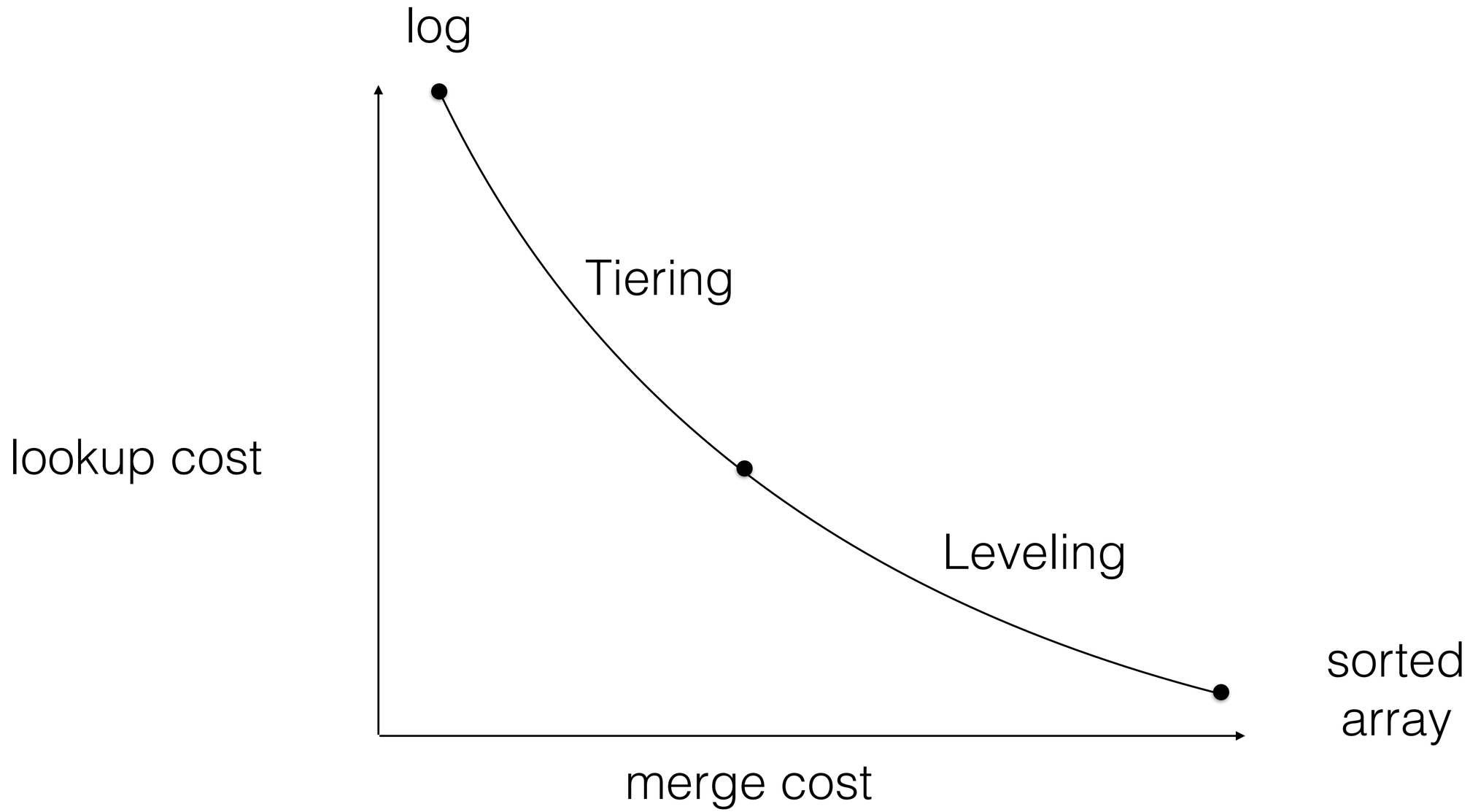
O(*N*) runs per level

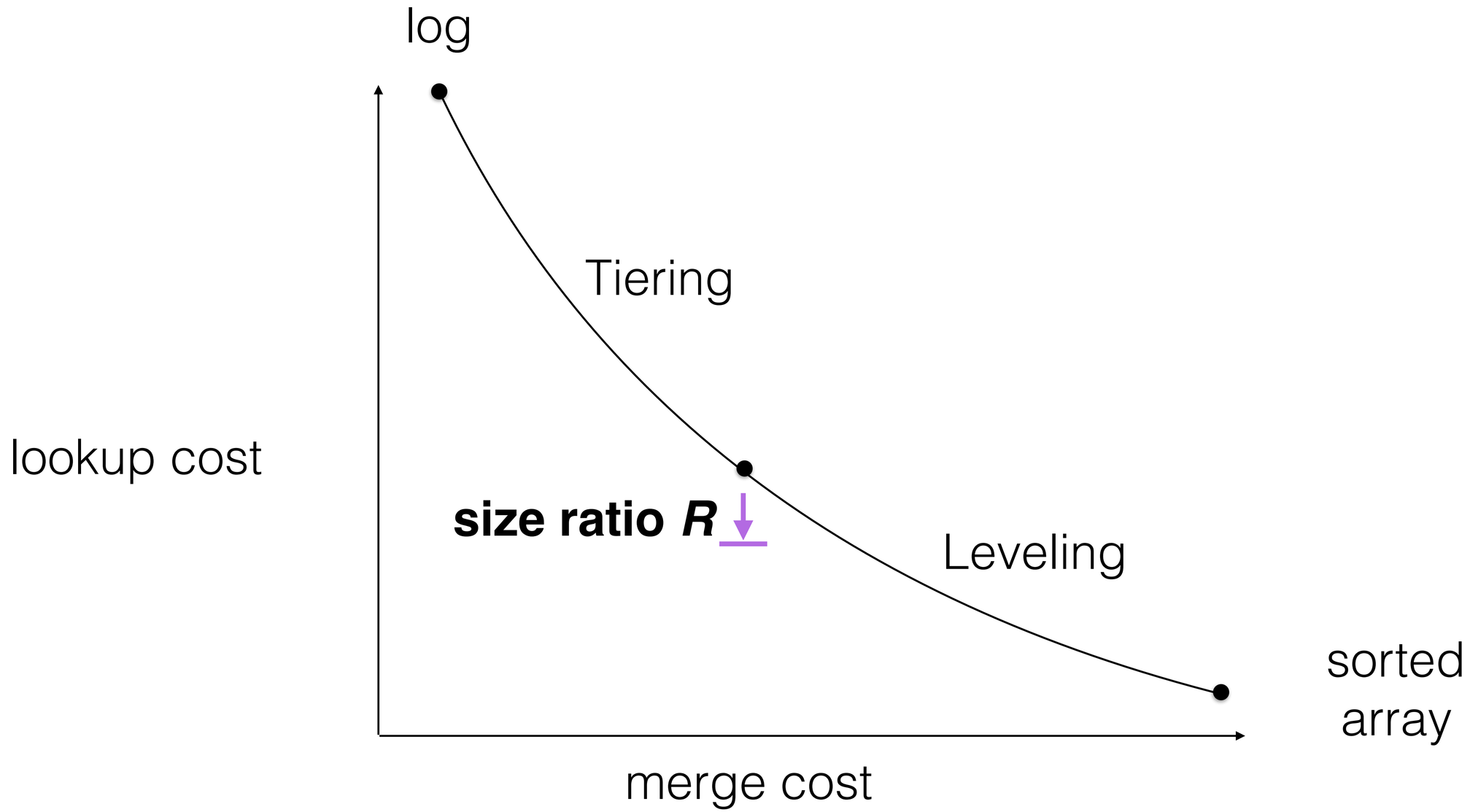
log

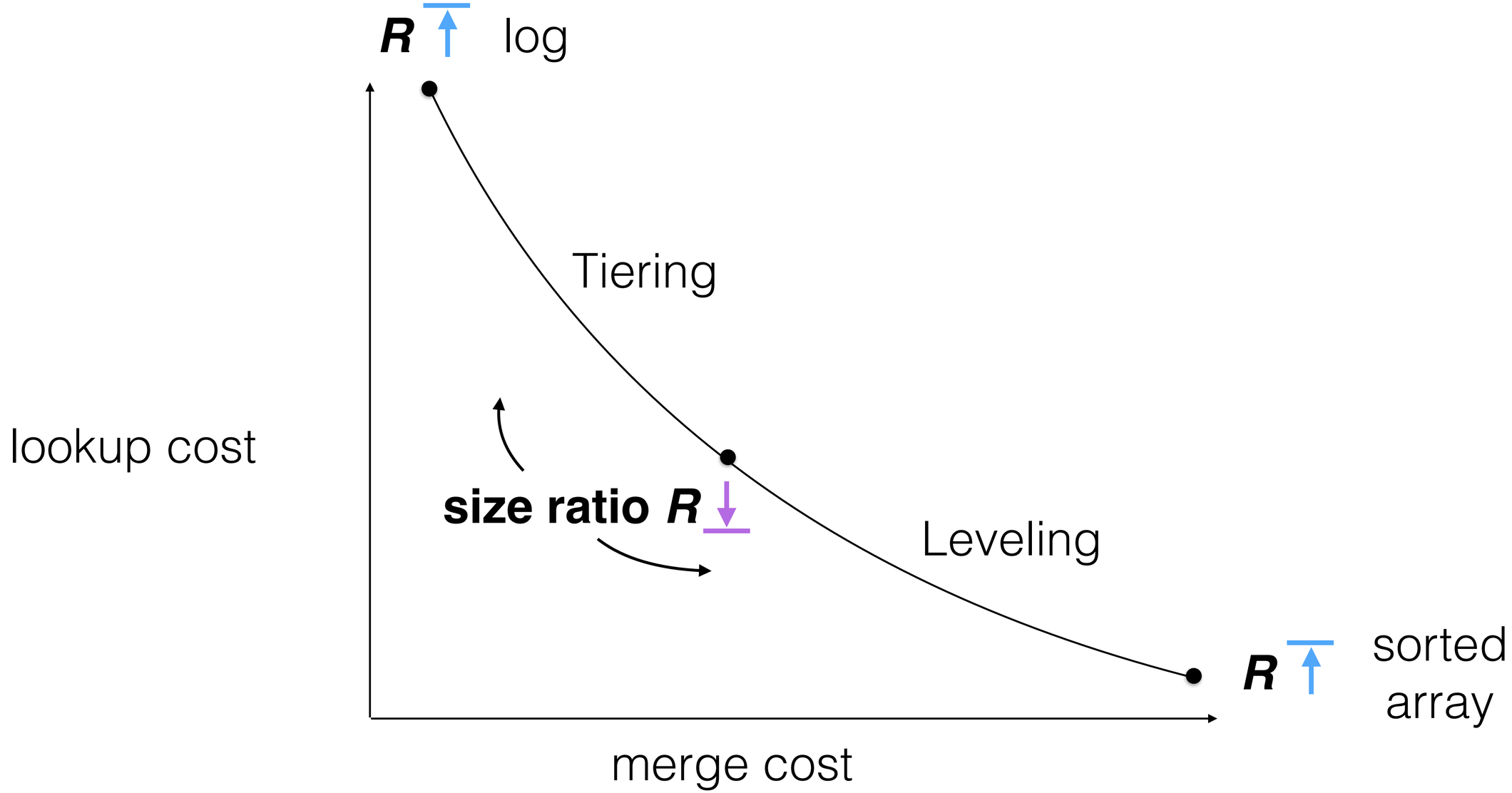
1 run per level

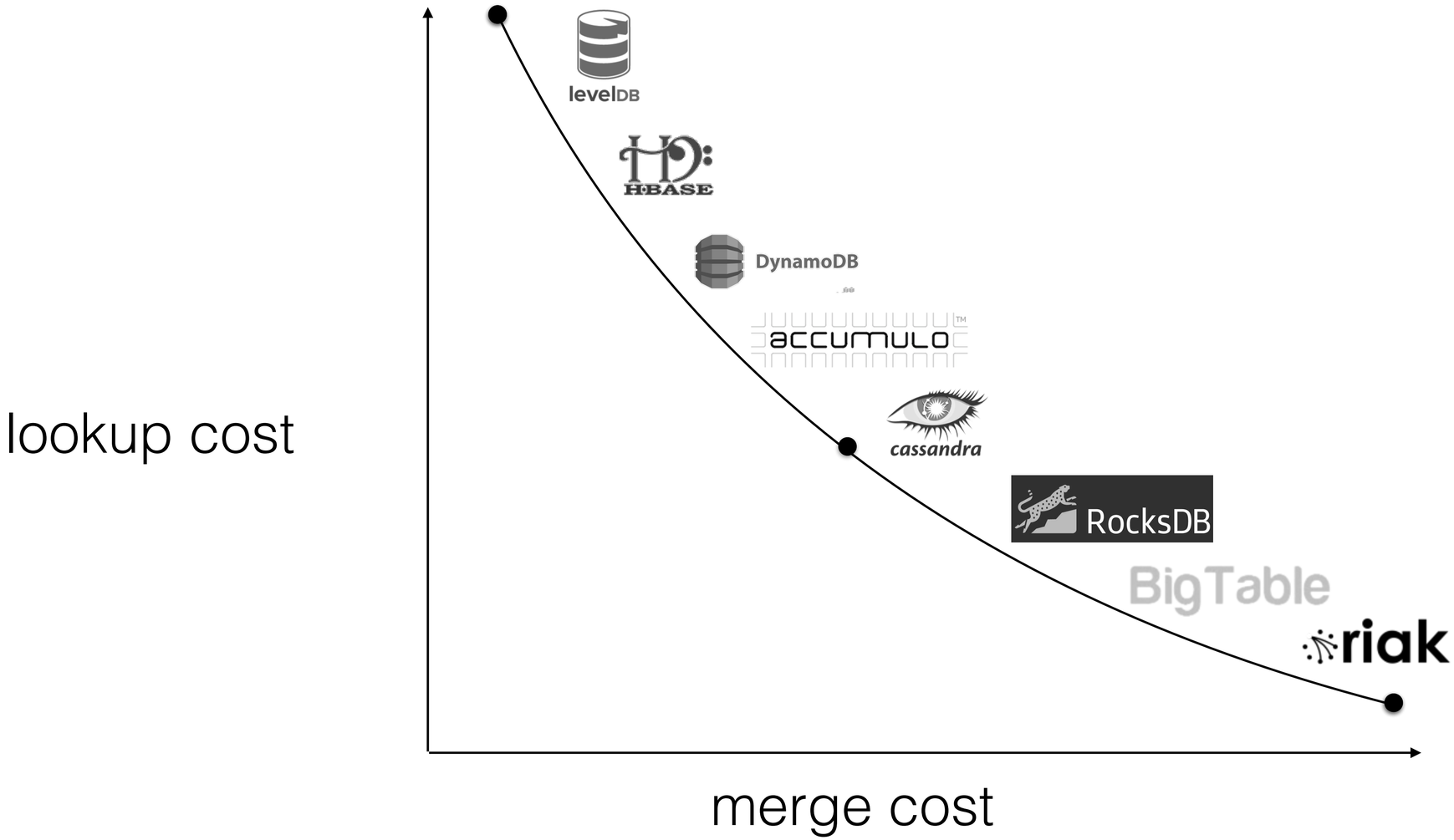
sorted array

size ratio *R*

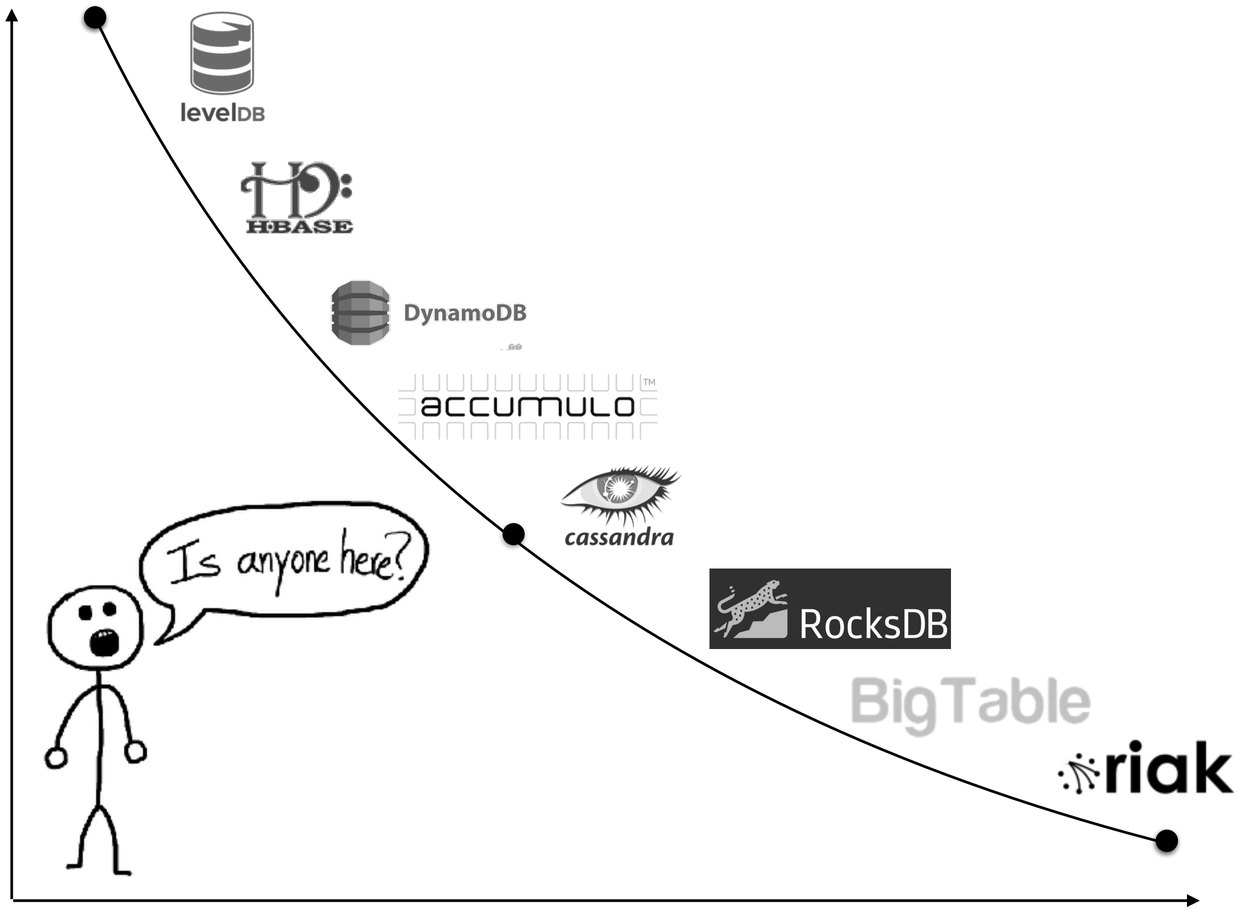

log
Tiering
lookup cost
Leveling
sorted
array
merge cost


log
Tiering
lookup cost
size ratio R
Leveling
sorted
array
merge cost

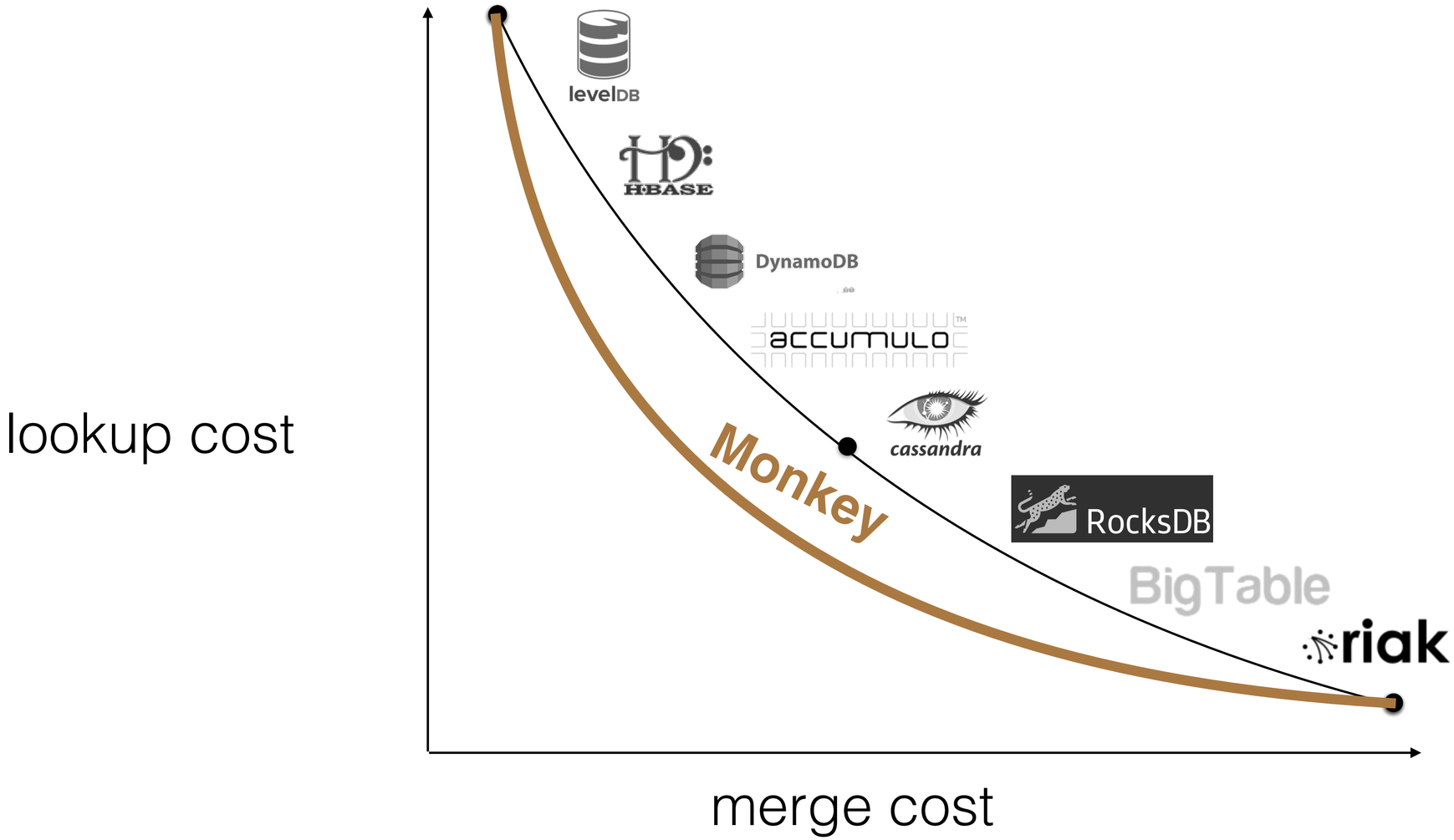

R
log
Tiering
lookup cost
size ratio R
Leveling
R
sorted
array
merge cost


lookup cost
levelDB
HBASE
DynamoDB
accumulo
cassandra
RocksDB
BigTable
riak
merge cost

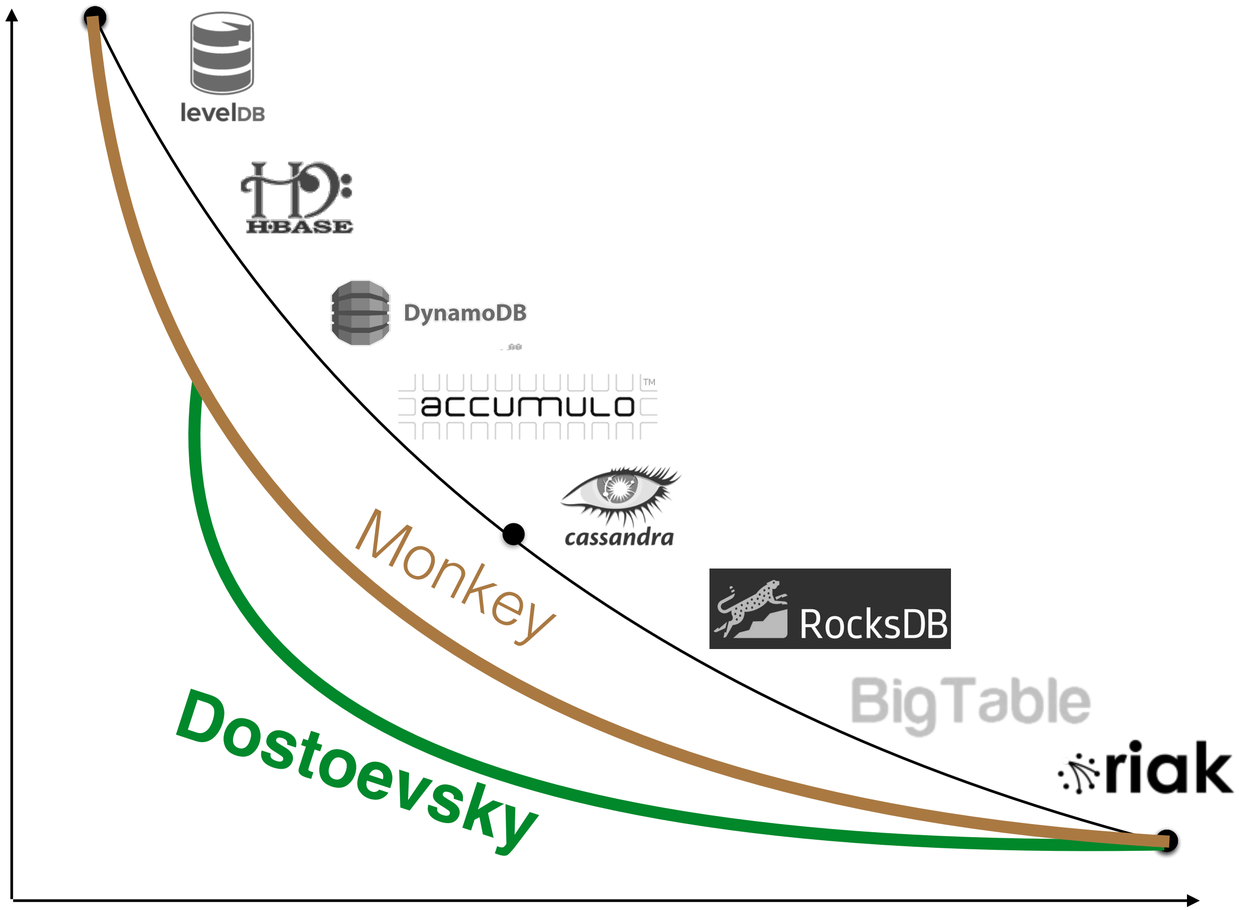
lookup cost



merge cost

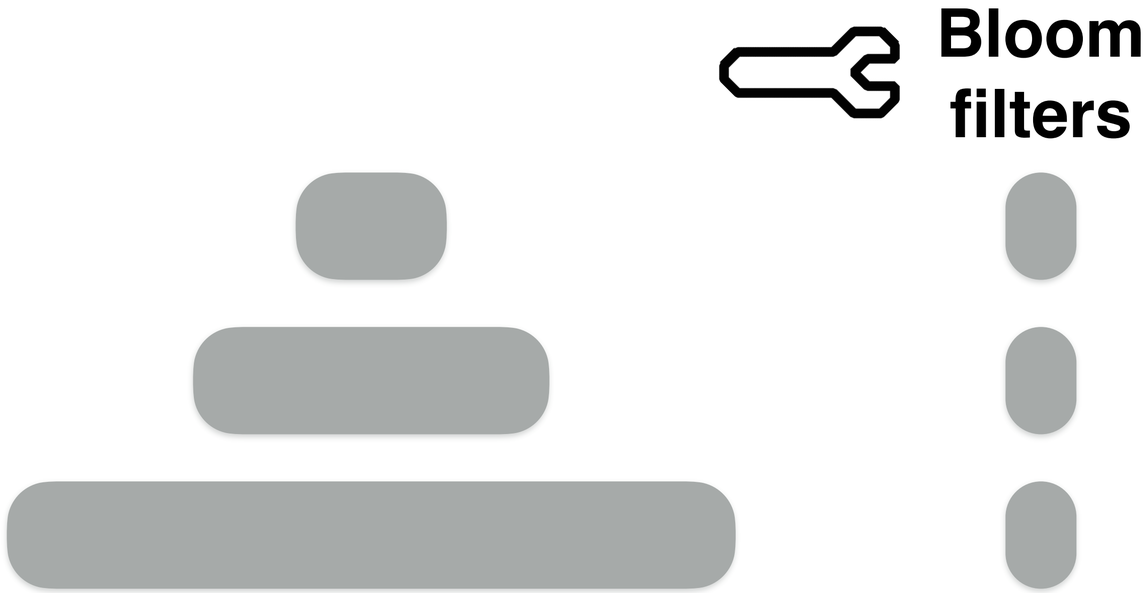

lookup cost
levelDB
HBASE
DynamoDB
accumulo
cassandra
Monkey
RocksDB
BigTable
riak
merge cost

lookup cost



merge cost

Monkey: Optimal Navigable Key-Value Store SIGMOD17



Monkey: Optimal Navigable Key-Value Store SIGMOD17

Bloom filters

false positive rate

O(p/R²)

O(p/R)

O(p)

Monkey: Optimal Navigable Key-Value Store SIGMOD17

Bloom filters

false positive rate

$O(\mathbf{e}^{-x}/R^2)$

$O(\mathbf{e}^{-x}/R)$

$O(\mathbf{e}^{-x})$

Monkey: Optimal Navigable Key-Value Store SIGMOD17

Bloom filters

false positive rate

$O(e^{-x}/R^2)$

$O(e^{-x}/R)$

$O(e^{-x})$

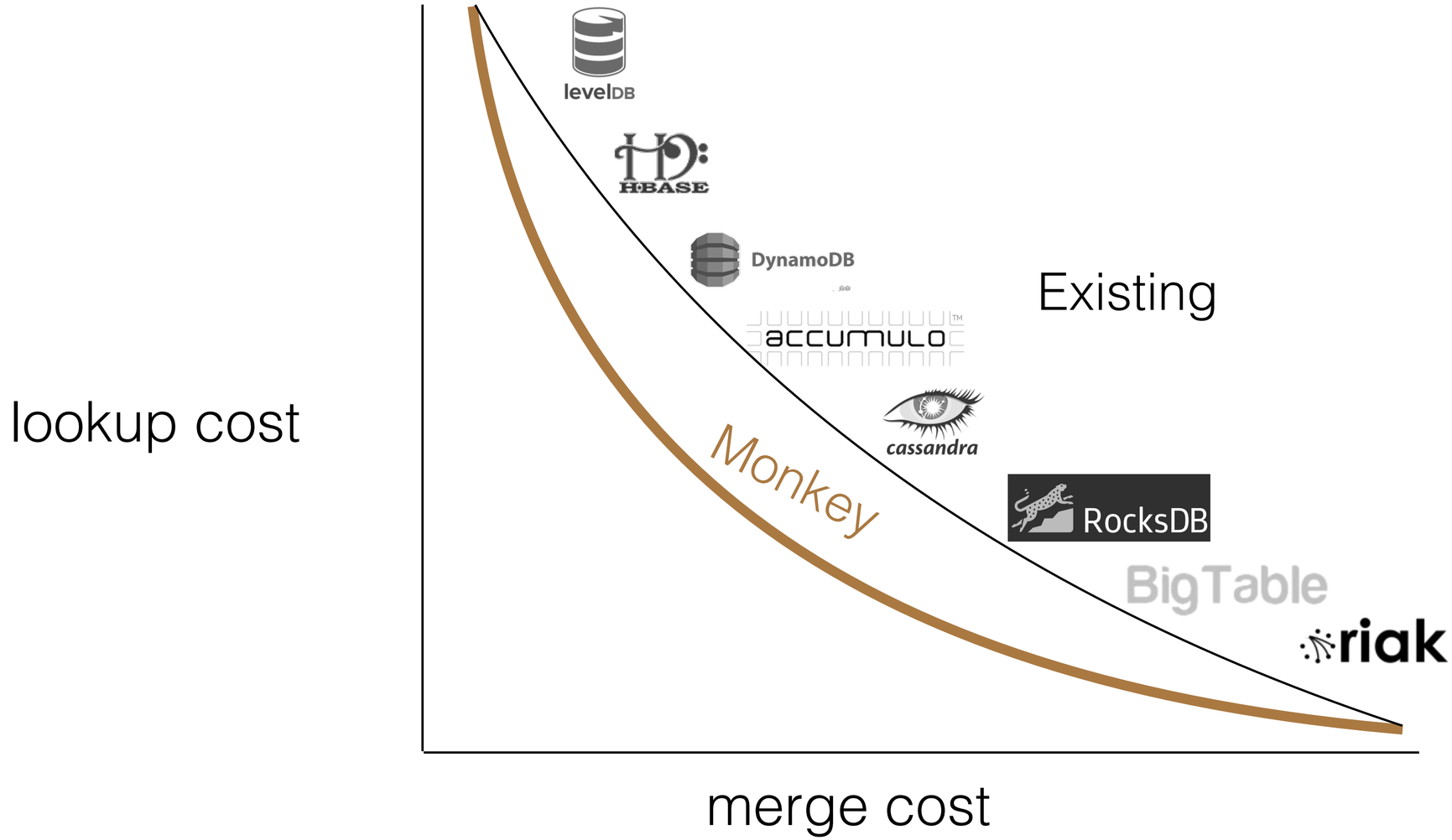
geometric progression

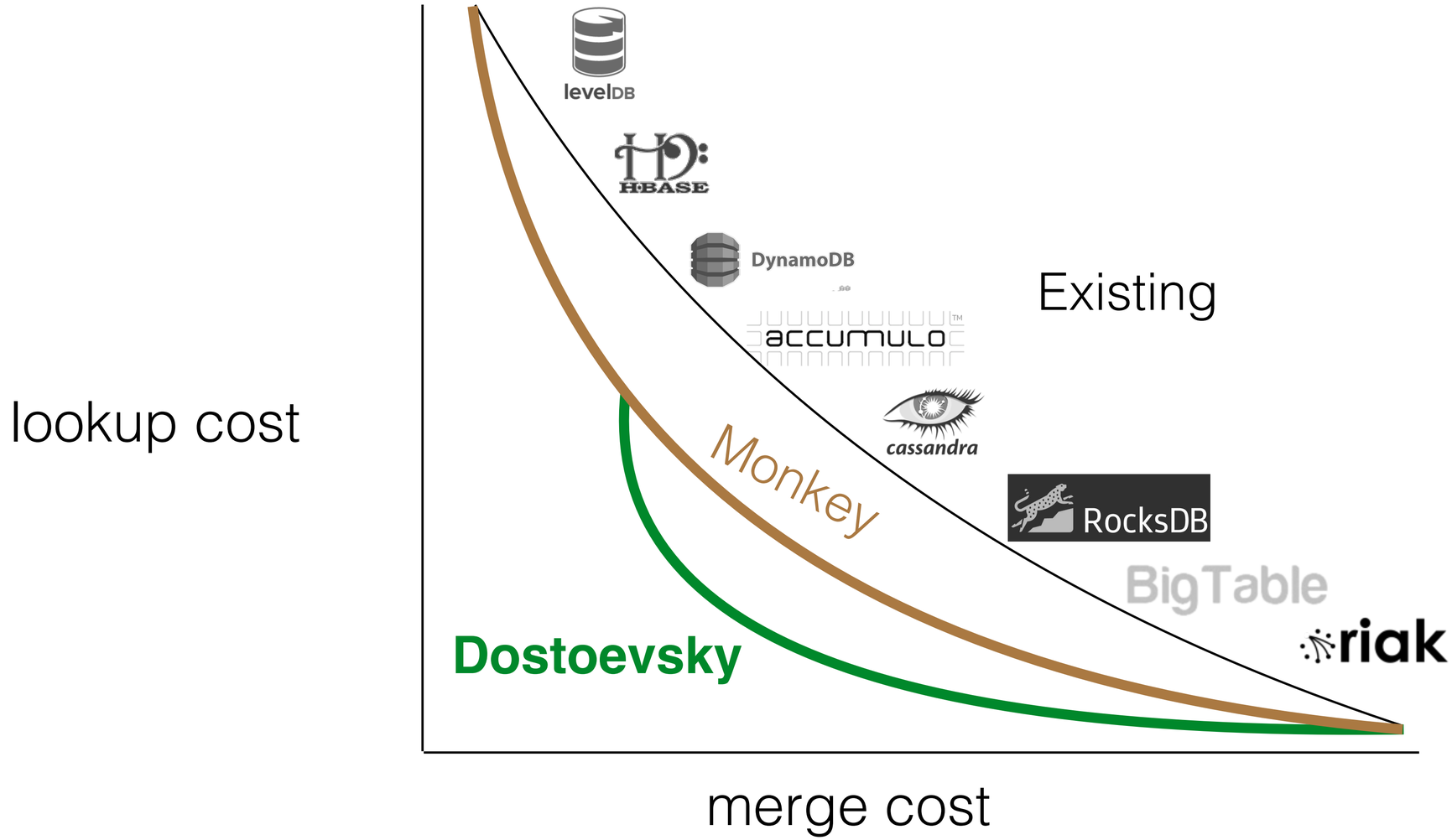
$= O(e^{-x})$

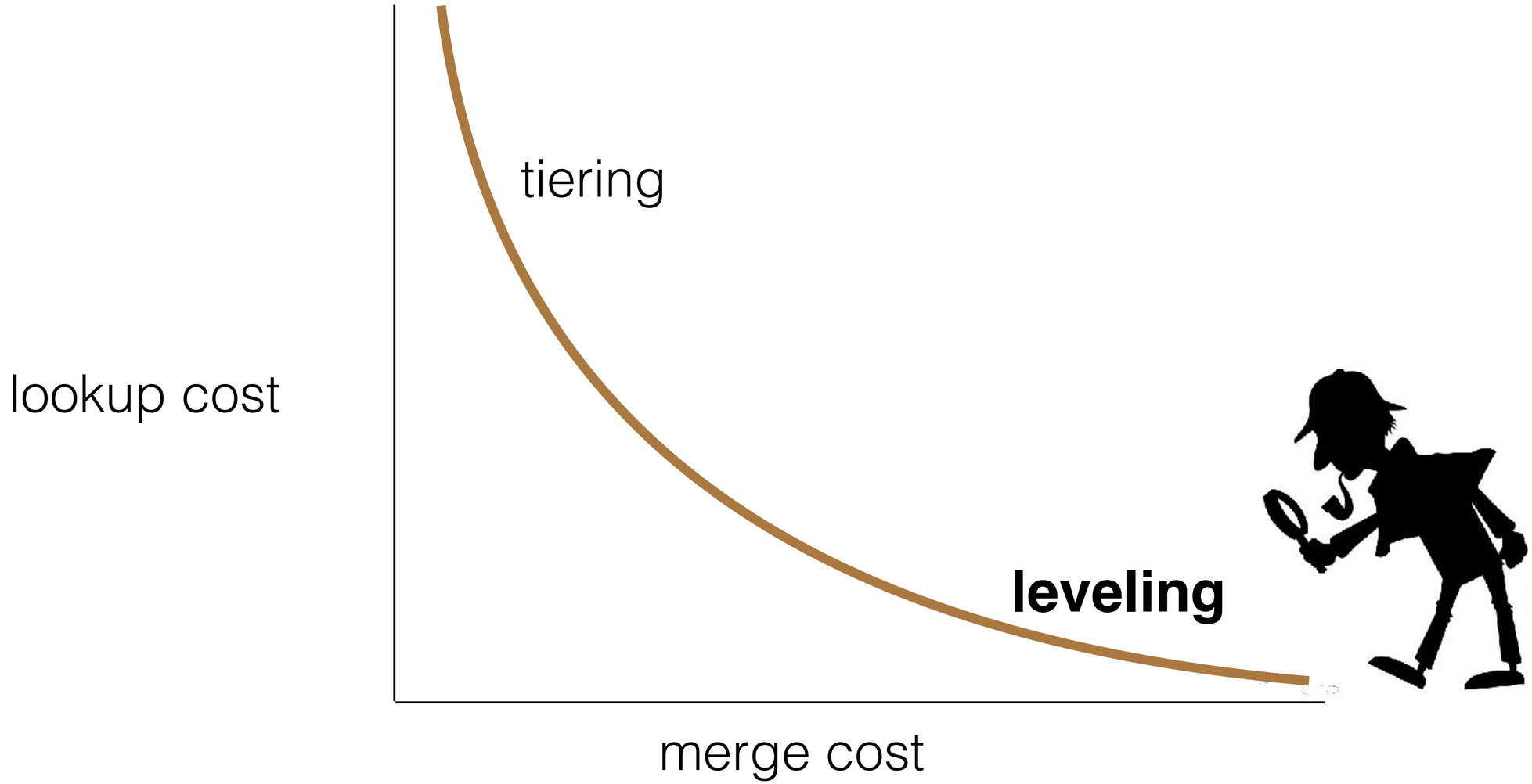
Monkey: Optimal Navigable Key-Value Store SIGMOD17

Bloom filters

$O(e^{-x} \cdot \log_R(N)) > O(e^{-x})$


lookup cost
levelDB
HBASE
DynamoDB
accumulo
cassandra
Existing
Monkey
RocksDB
BigTable
riak
merge cost


lookup cost
levelDB
HBASE
DynamoDB
accumulo
cassandra
Existing
Monkey
RocksDB
BigTable
riak
Dostoevsky
merge cost


lookup cost
tiering
leveling
merge cost

I/O overheads with leveling

point

long range

short range

merging

point

false positive rates

exponentially decreasing

$O(e^{-x/R^2})$

$O(e^{-x/R})$

$O(e^{-x})$

false positive rates

$O(e^{-x/R^2})$

$O(e^{-x/R})$

point ⟶ $O(e^{-x})$

point

largest level

$O(e^{-x})$

long range

short range

merging

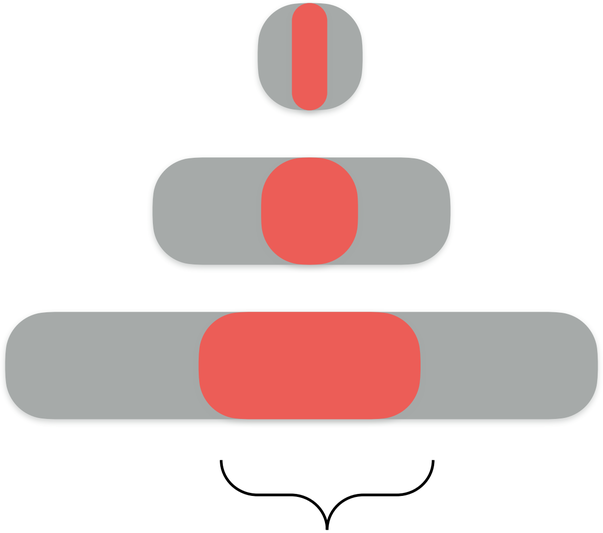
long range

target range

O(s/R^2)

O(s/R)

O(s)



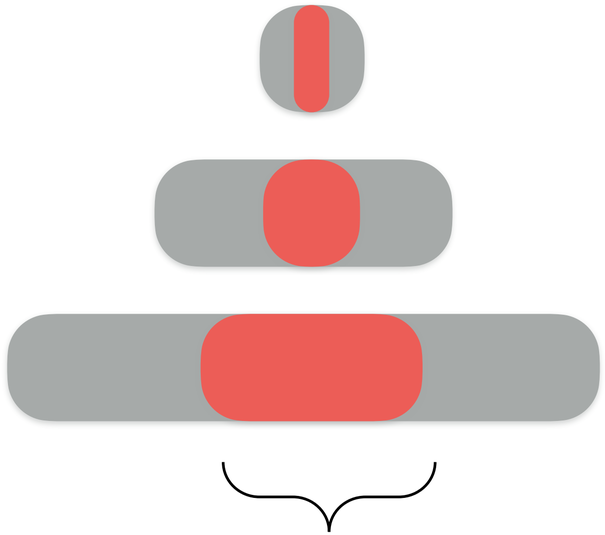
target key range

target range

$O(s/R^2)$

$O(s/R)$

$O(s)$



long range

largest level

target key range

point

largest level

$O(e^{-x})$

long range

largest level

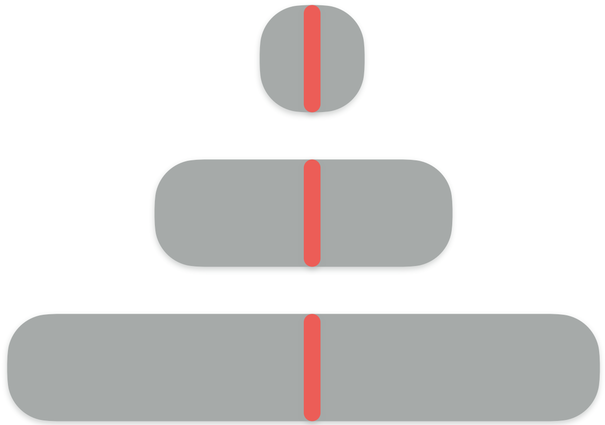
$O(s)$

short range

merging

short range

target range



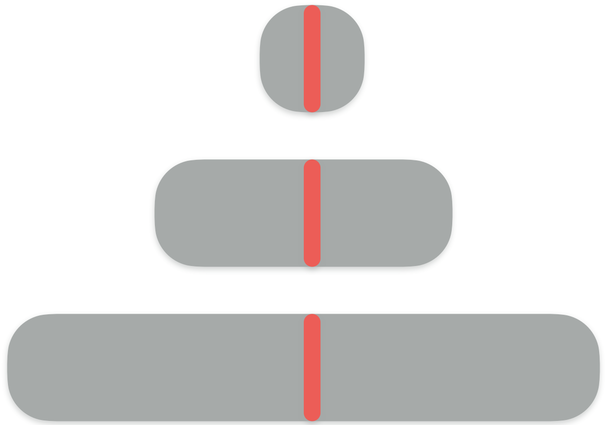
1

1

1

short range

target range



1

1

1

all levels

O(**log$_R$(*N*)**)

point

largest level

$O(e^{-x})$

long range

largest level

$O(s)$

short range

all levels

$O(\log_R(N))$

merging

merging

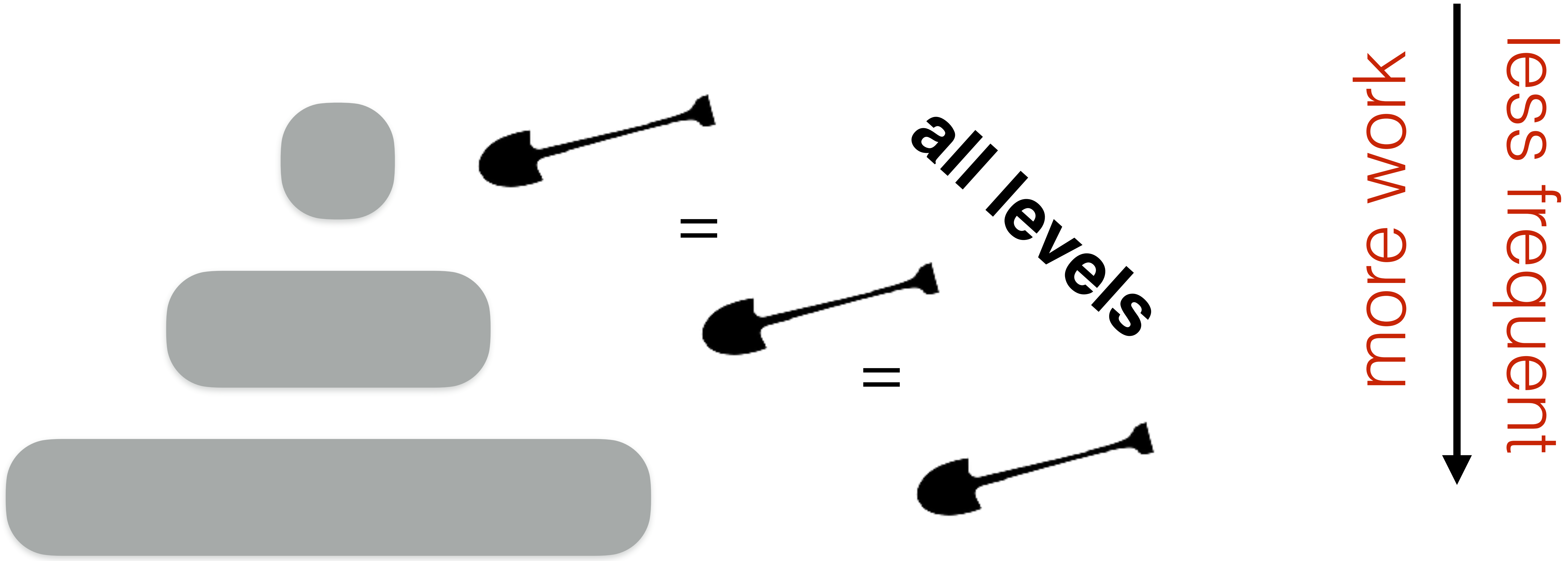
exponentially more work

merging

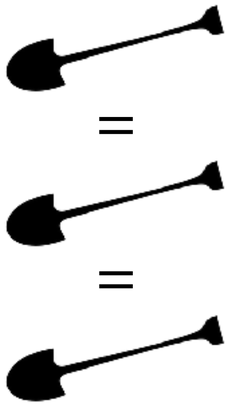
exponentially more work

exponentially less frequent

merging

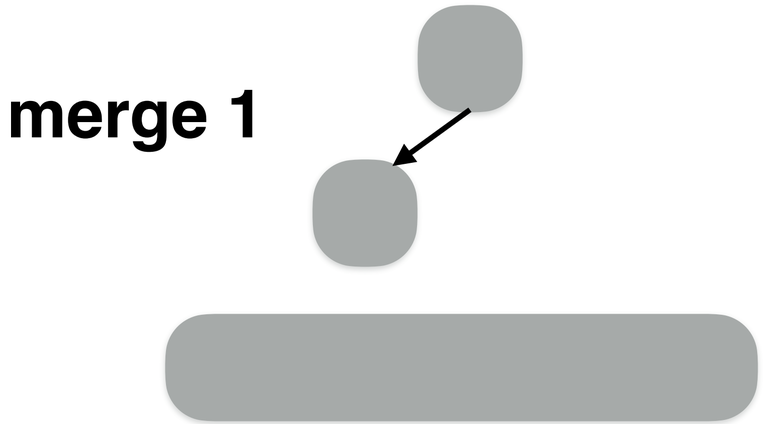


merging

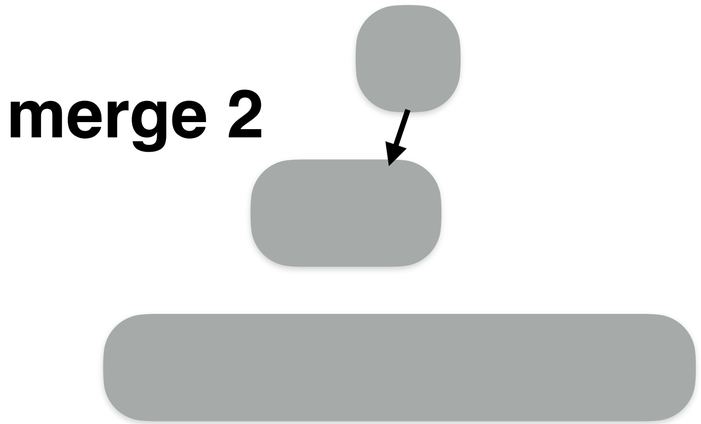


all levels

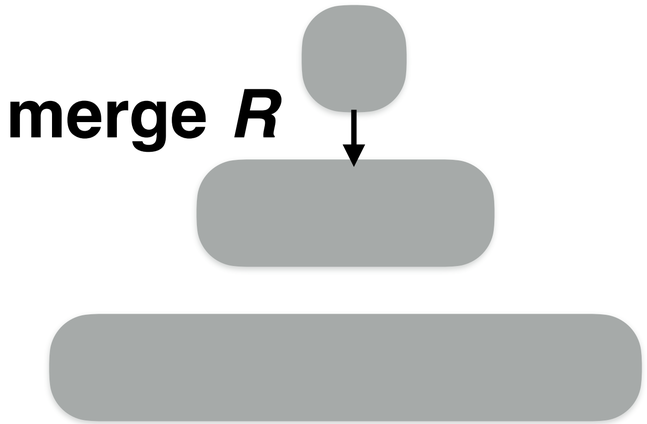
merging



merging



merging



merging

write-amplification

O(***R***)

O(***R***)

O(***R***)

merging

$O(R)$

$O(R)$

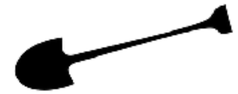
$O(R)$

$\mathbf{O(R \cdot \log_R(N))}$

point

long range

short range



merging

largest level

$O(e^{-x})$

=

$O(e^{-x}/R^2)$

+

$O(e^{-x}/R)$

+

$O(e^{-x})$

largest level

$O(s)$

=

$O(s/R^2)$

+

$O(s/R)$

+

$O(s)$

all levels

$O(\log_R(N))$

=

1

+

1

+

1

all levels

$O(R \cdot \log_R(N))$

=

$O(R)$

+

$O(R)$

+

$O(R)$

point

largest level

long range

largest level

merging

all levels

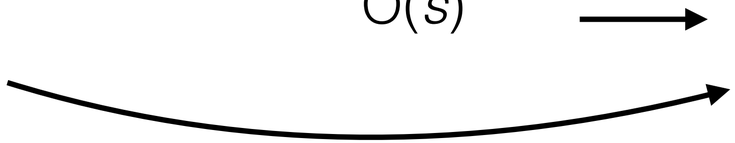
O(*R*)

O(*R*)

O(*R*)

O(e^{-x})

O(*s*)



DASlab
@ Harvard SEAS

point

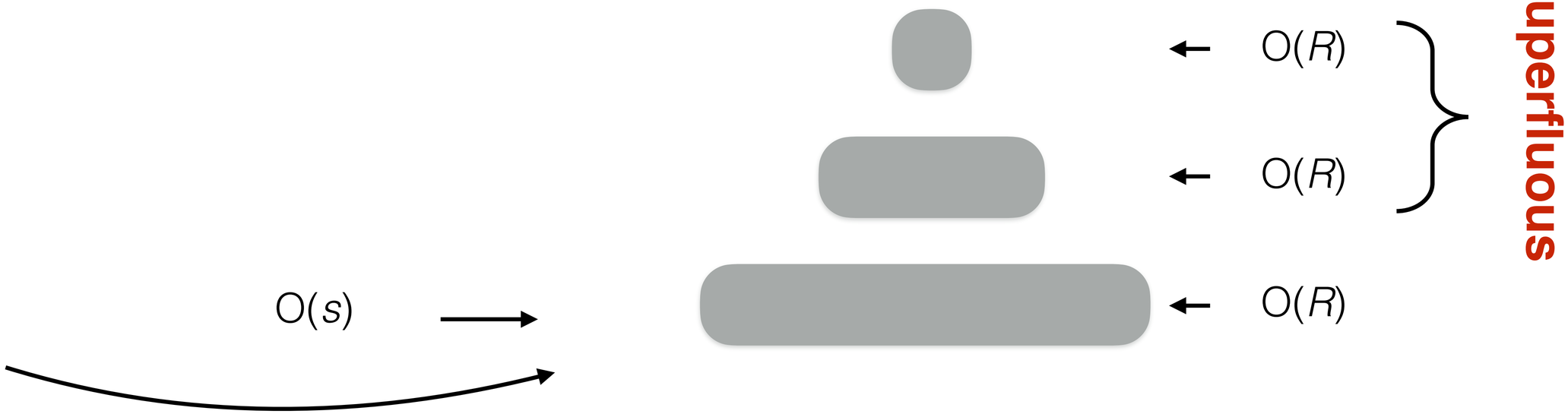
largest level

long range

largest level

merging

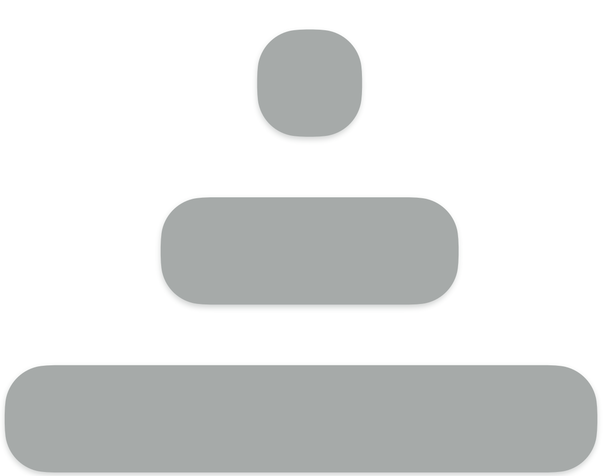
all levels

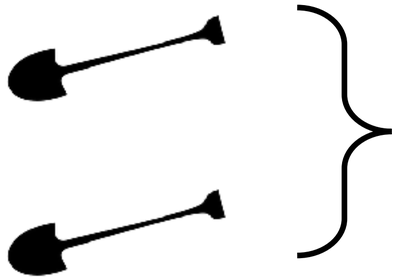


DASlab @ Harvard SEAS

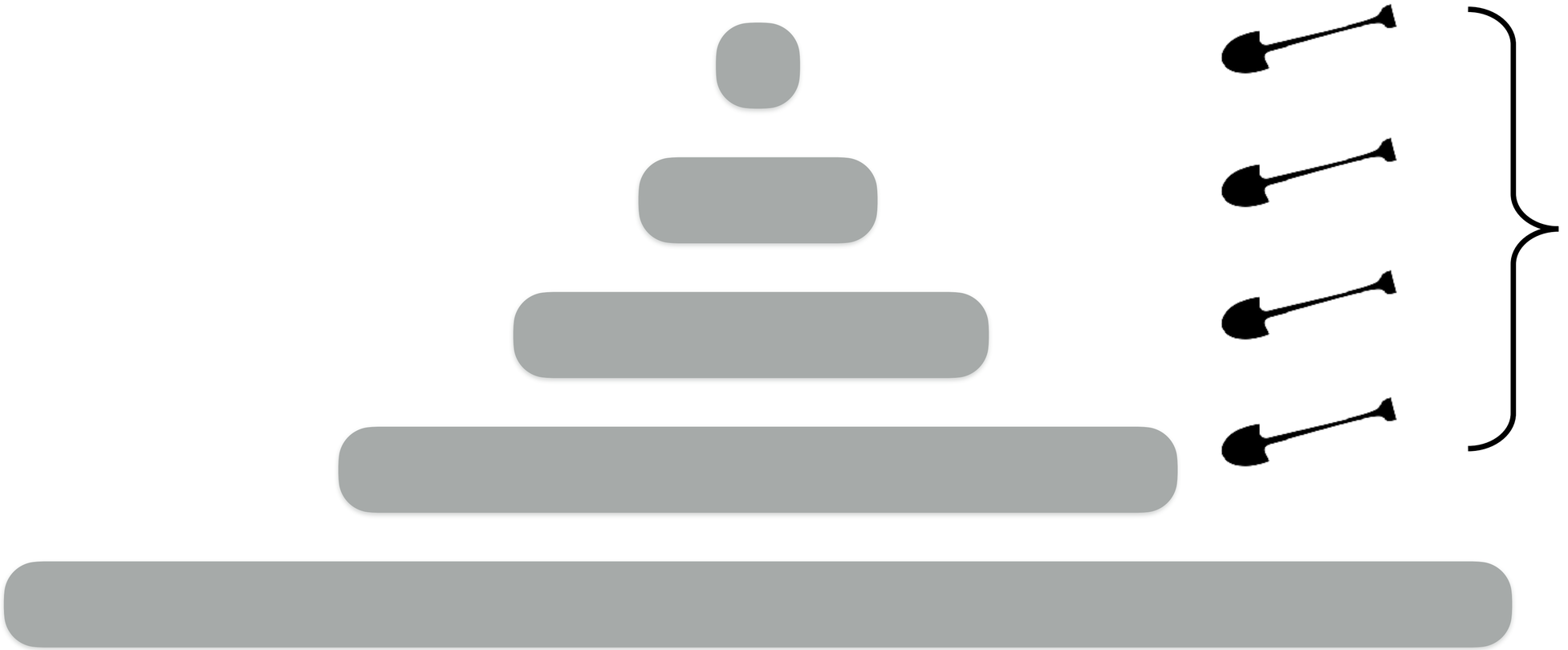
for point lookups and long range lookups

merging at **smaller levels** is **superfluous**





worse as data grows!



poor performance

poor performance

lower device lifetime (on SSD)

Dostoevsky

Dostoevsky: **S**pace-**T**ime **O**ptimized **Ev**olvable **S**calable **K**e**y**-Value Store

very write-optimized

Dostoevsky: **S**pace-**T**ime **O**ptimized **Ev**olvable **S**calable **K**e**y**-Value Store

Tiering
merge-optimized

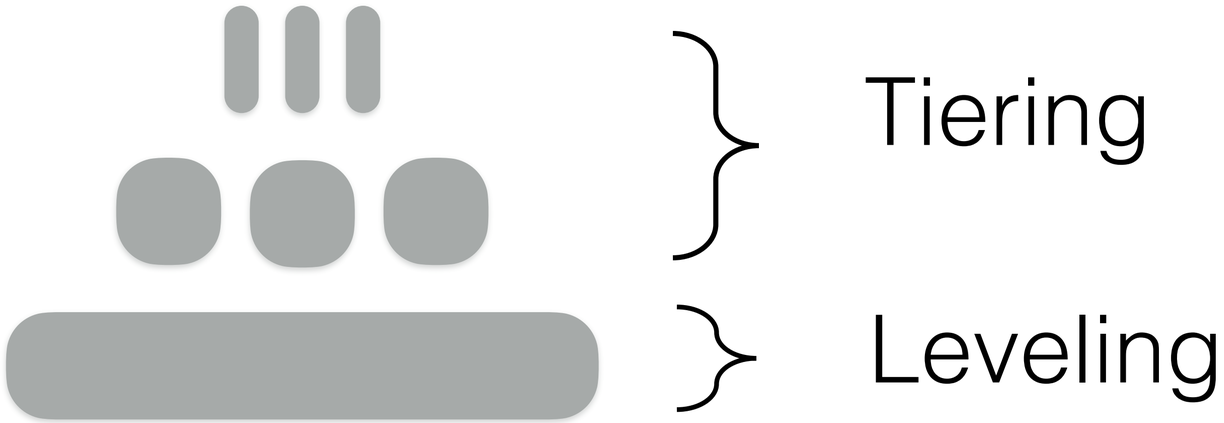
Leveling
lookup-optimized

Tiering
merge-optimized

Lazy Leveling
mixed-workloads

Leveling
lookup-optimized

Lazy Leveling



Lazy Leveling

merge when level fills

merge to have at most 1 run

point

long range

short range

merging

point

false positive rates

$O(e^{-x/R^3})$

$O(e^{-x/R^2})$

$O(e^{-x})$

point

false positive rates

exponentially decreasing

$O(e^{-x/R^3})$

$O(e^{-x/R^2})$

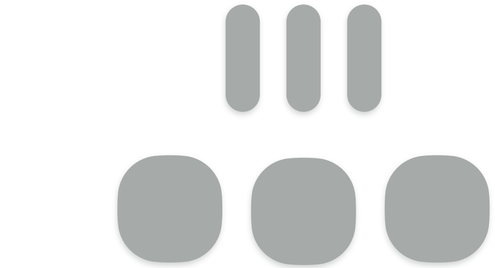
$O(e^{-x})$

false positive rates

$O(e^{-x}/R^3)$

$O(e^{-x}/R^2)$

point ⟶ $O(e^{-x})$



largest level

point

long range

short range

merging

$O(e^{-x})$

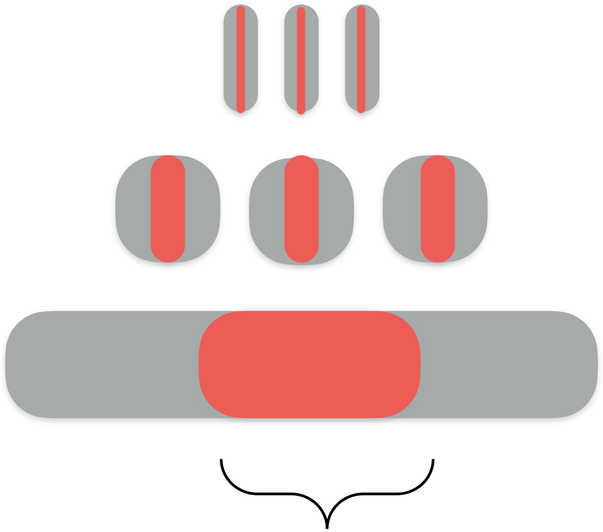
long range

target range

O(s/R^2)

O(s/R)

O(s)



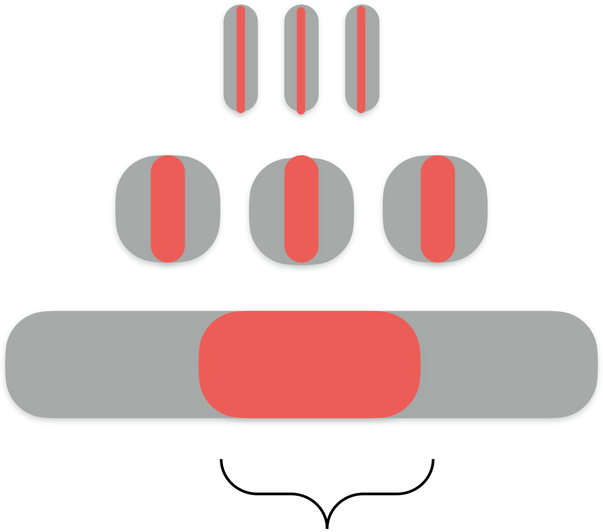
target key range

target range

$O(s/R^2)$

$O(s/R)$

$O(s)$



long range

largest level

target key range

point

long range

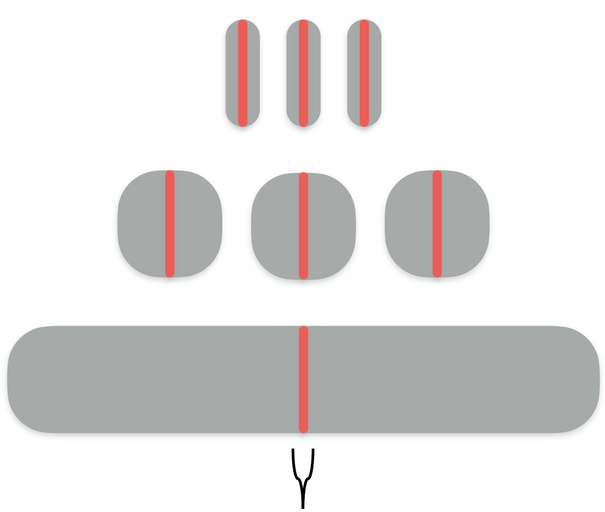
short range

merging

$O(e^{-x})$

$O(s)$

short range



O(***R***)

O(***R***)

1

O(**1+*R*·(log$_{R}$(*N*)-1)**)

target key range

point

long range

short range

merging

$O(e^{-x})$

$O(s)$

$O(1+R\cdot(\log_R(N)-1))$

merging

write-amplification

O(**1**)

O(**1**)

O(***R***)

merging

write-amplification

O(1)

O(1)

O(*R*)

O(***R*** + **log**$_{R}$(***N***))

point

long range

short range

merging

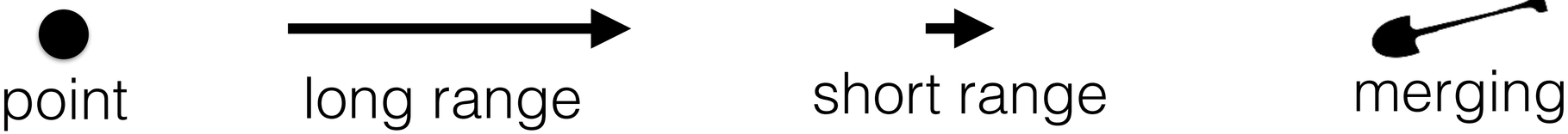
$O(e^{-x})$

$O(s)$

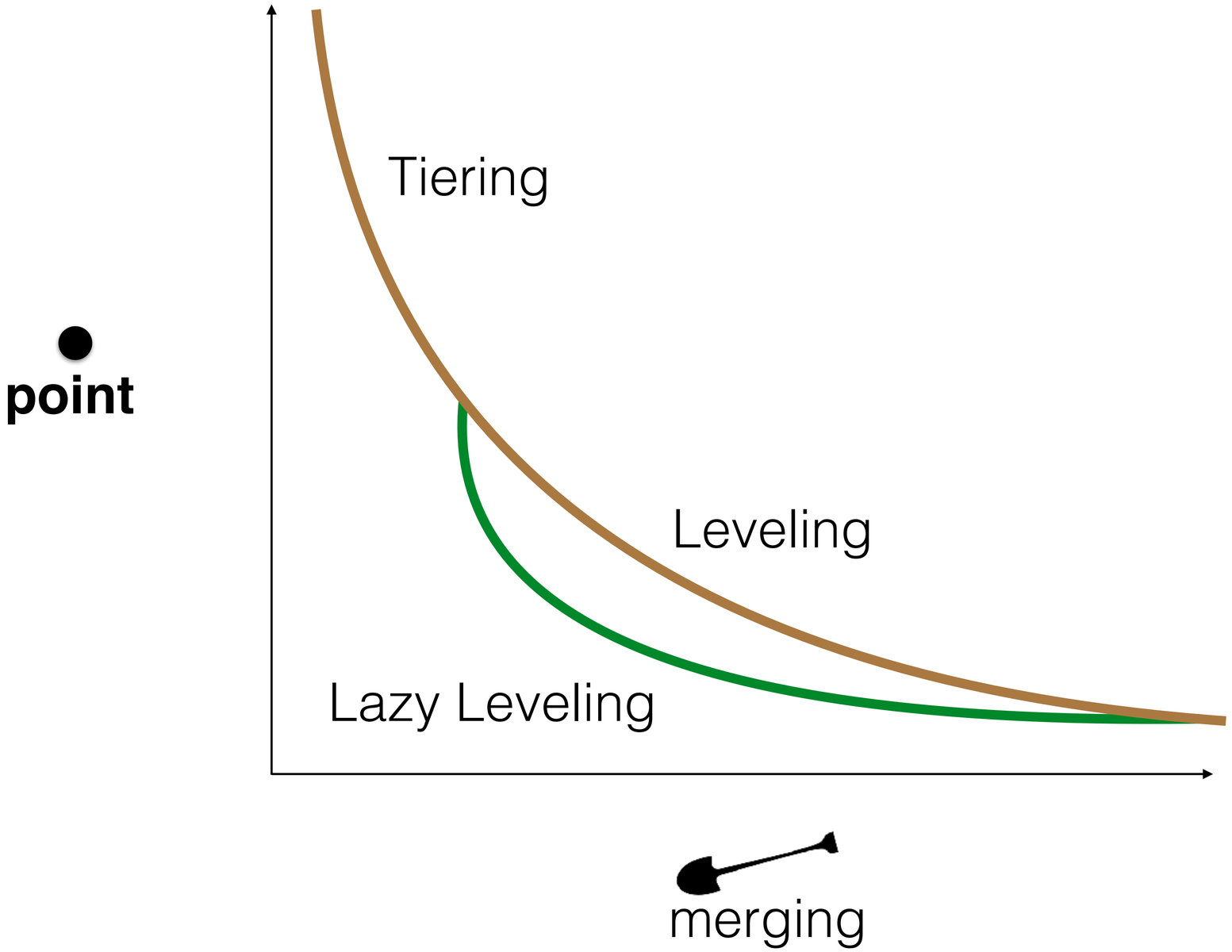
$O(1+R \cdot (\log_R(N)-1))$

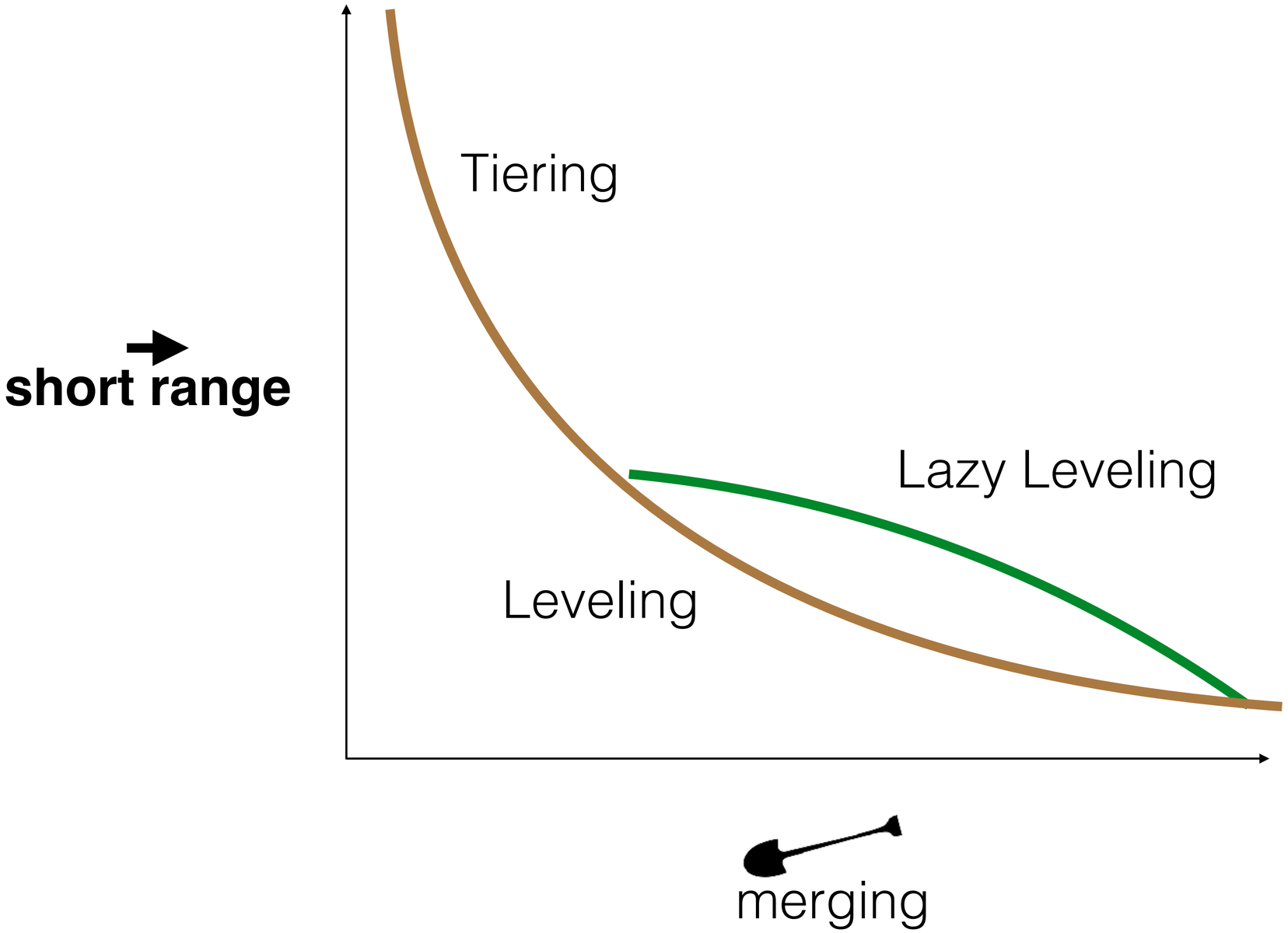
$O(R + \log_R(N))$

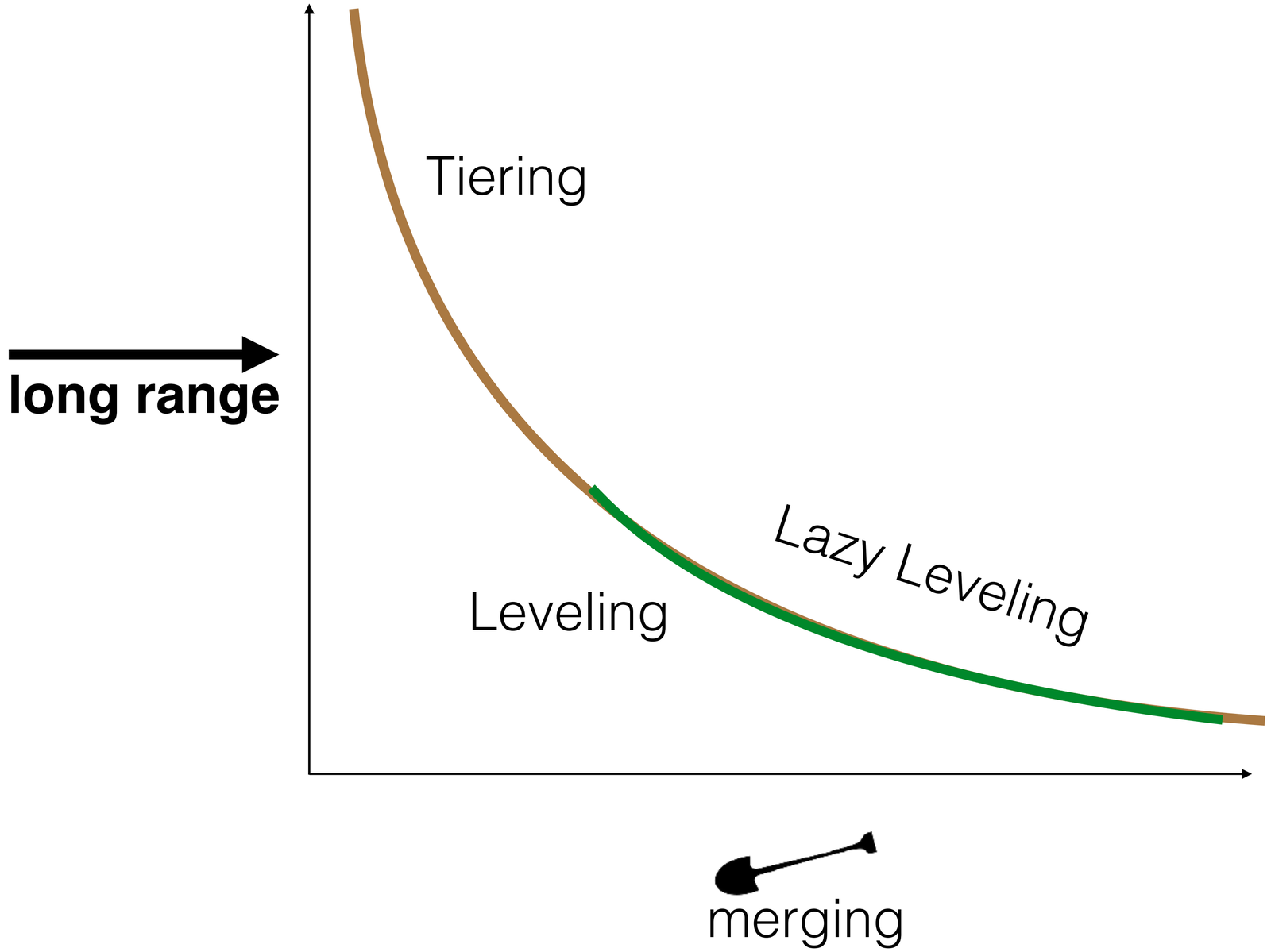
	point	long range	short range	merging
Lazy Leveling	$O(e^{-x})$	$O(s)$	$O(1+R \cdot (\log_R(N)-1))$	$O(R + \log_R(N))$
	=	=	∨	∧
Leveling	$O(e^{-x})$	$O(s)$	$O(\log_R(N))$	$O(R \cdot \log_R(N))$



Tiering	$O(R \cdot e^{-x})$	$O(R \cdot s)$	$O(R \cdot \log_R(N))$	$O(\log_R(N))$
	V	V	V	Λ
Lazy Leveling	$O(e^{-x})$	$O(s)$	$O(1+R \cdot (\log_R(N)-1))$	$O(R + \log_R(N))$
Leveling	$O(e^{-x})$	$O(s)$	$O(\log_R(N))$	$O(R \cdot \log_R(N))$


point
Tiering
Leveling
Lazy Leveling
merging


short range
Tiering
Lazy Leveling
Leveling
merging


Tiering
long range
Lazy Leveling
Leveling
merging

Tiering

Lazy Leveling

Leveling

Tiering

updates 

Lazy Leveling

Leveling

Tiering

updates 

Lazy Leveling

Leveling

short range 

Tiering

updates 

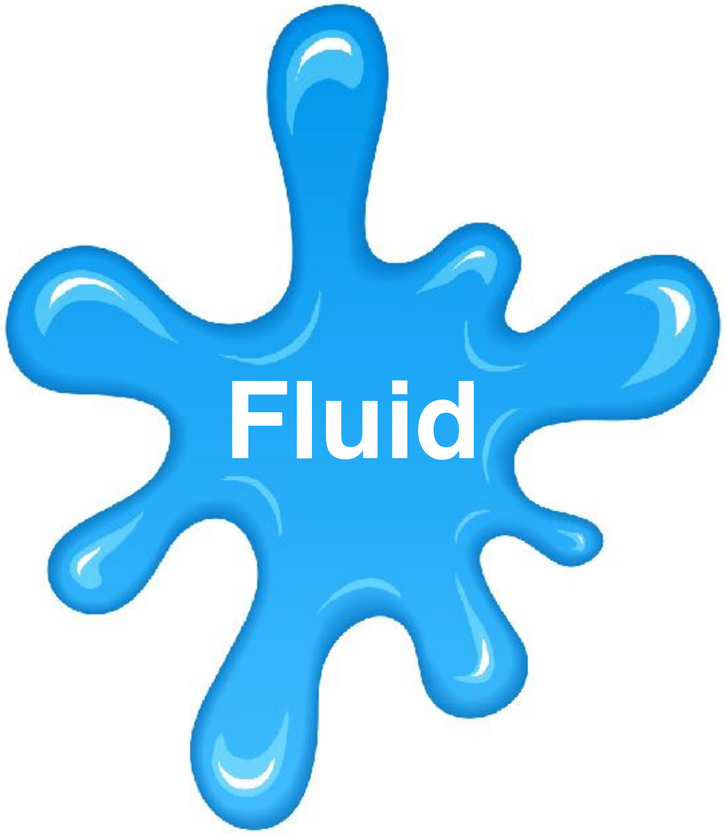
Lazy Leveling

updates & point 

Leveling

short range 

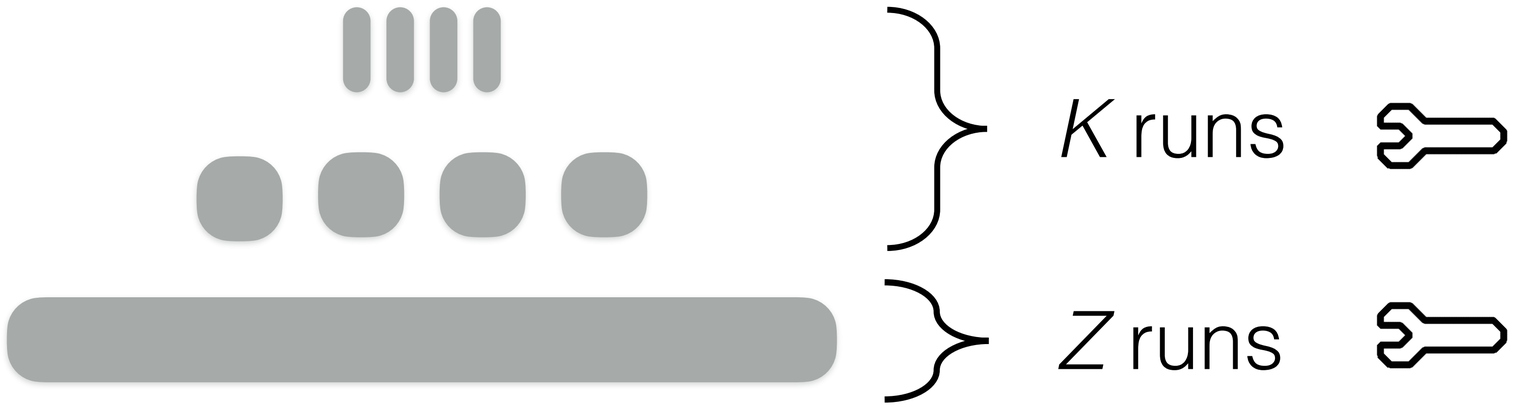
Lazy Leveling



Tiering

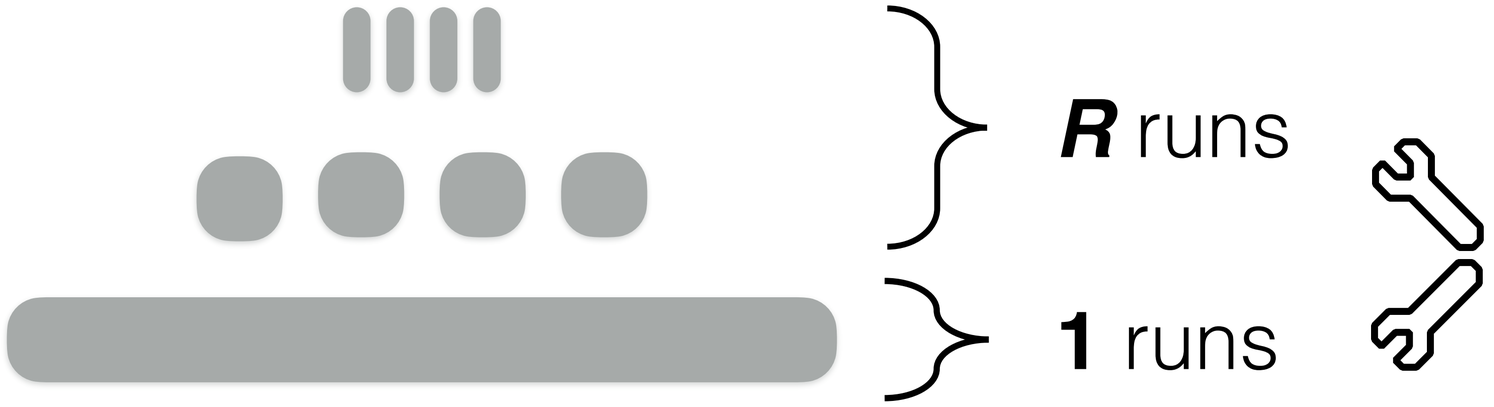
Leveling

Fluid LSM-Tree



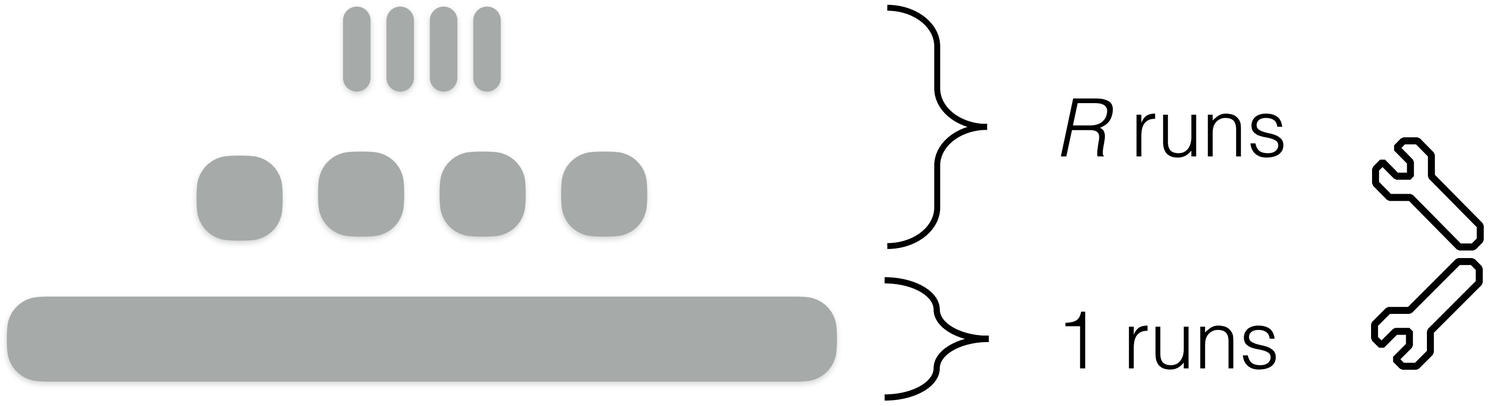
Fluid LSM-Tree

Lazy Leveling



point long range short range merging

Lazy Leveling



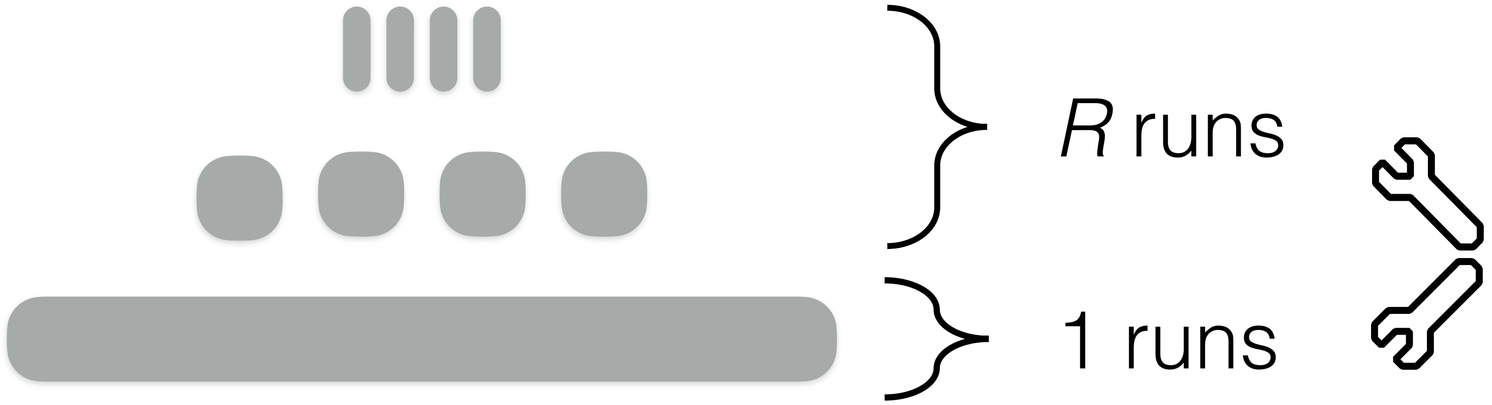
point

long range

optimize
short range

merging

Lazy Leveling



point

long range

optimize
short range

merging

Lazy Leveling

2 runs

1 runs

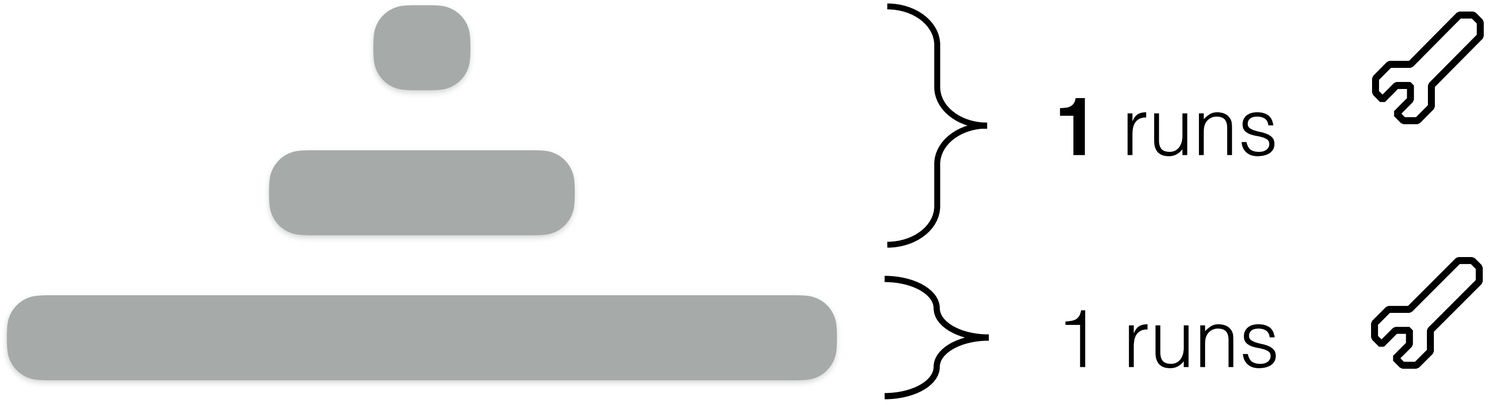
point

long range

optimize
short range

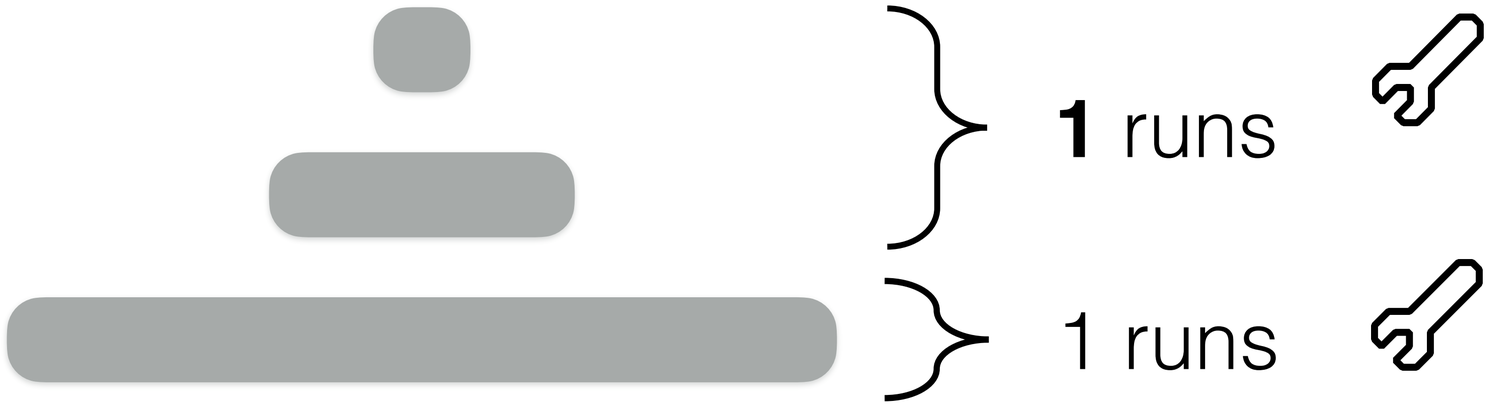
merging

Leveling



point long range short range optimize **merging**

Leveling



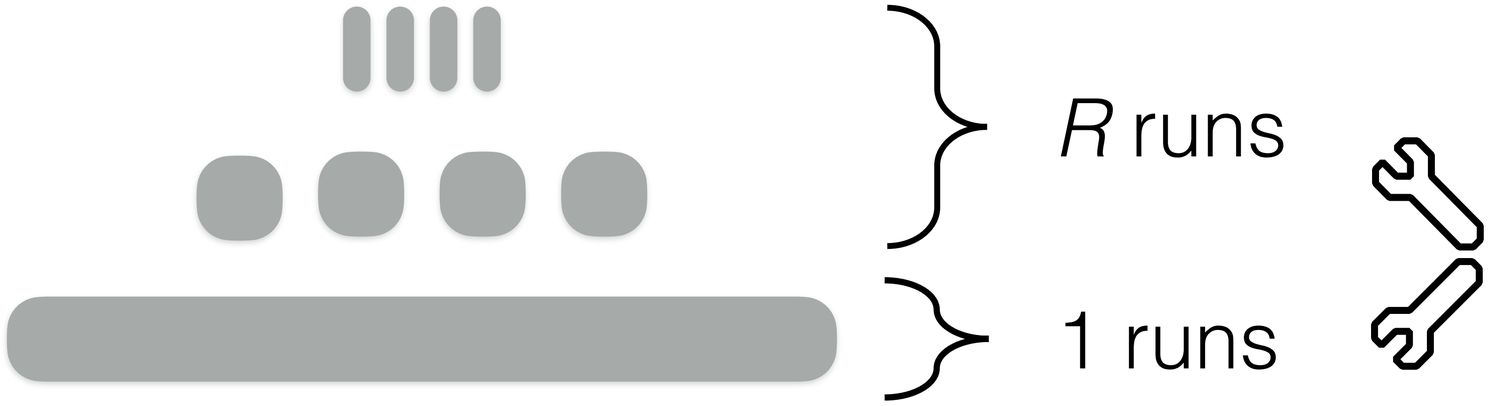
point

long range

short range

optimize
merging

Lazy Leveling



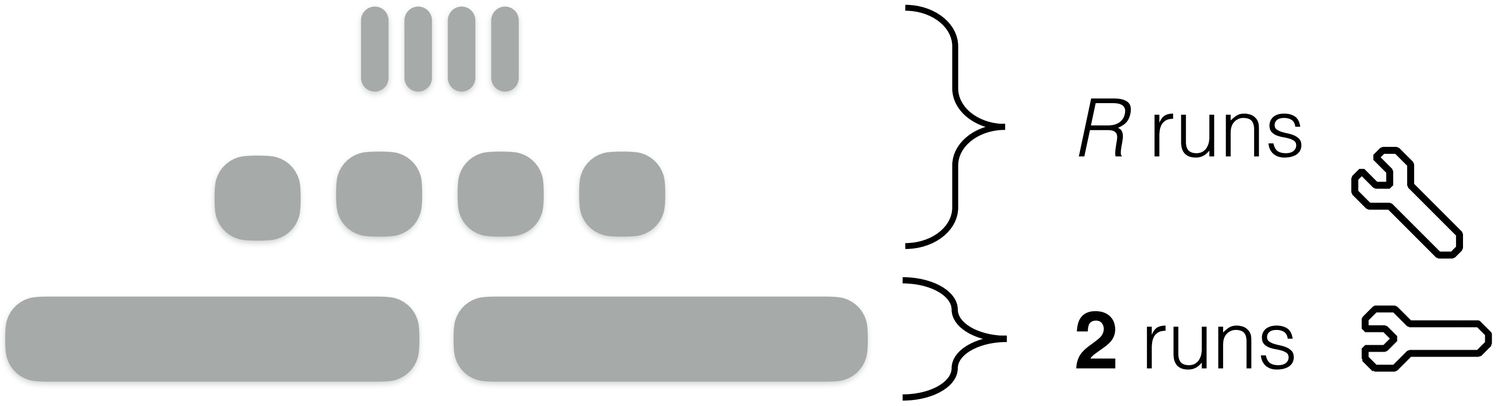
point

long range

short range

optimize
merging

Lazy Leveling



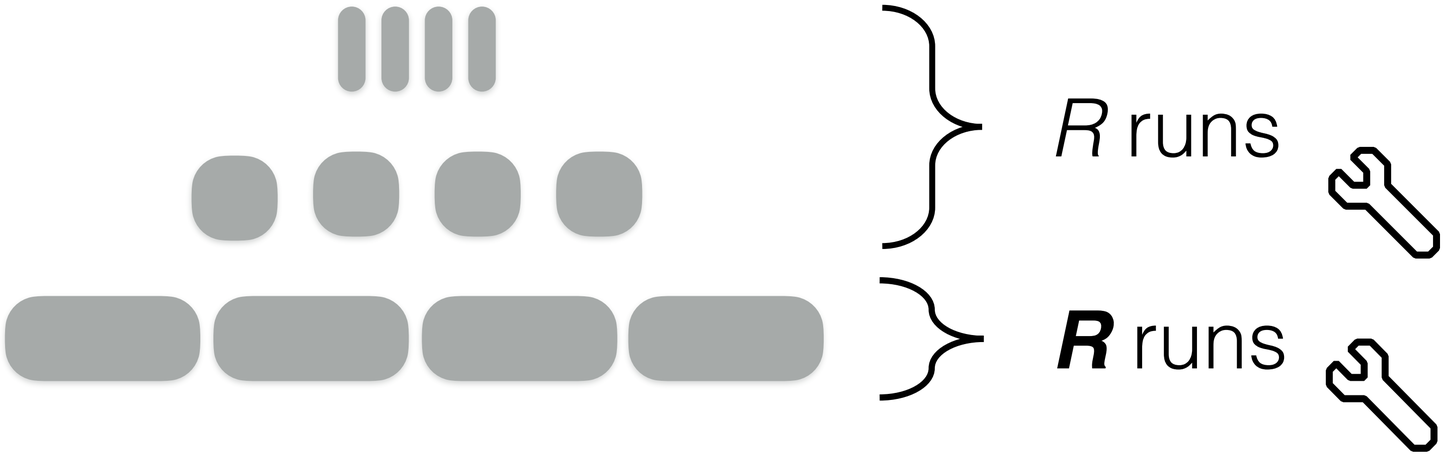
point

long range

short range

optimize
merging

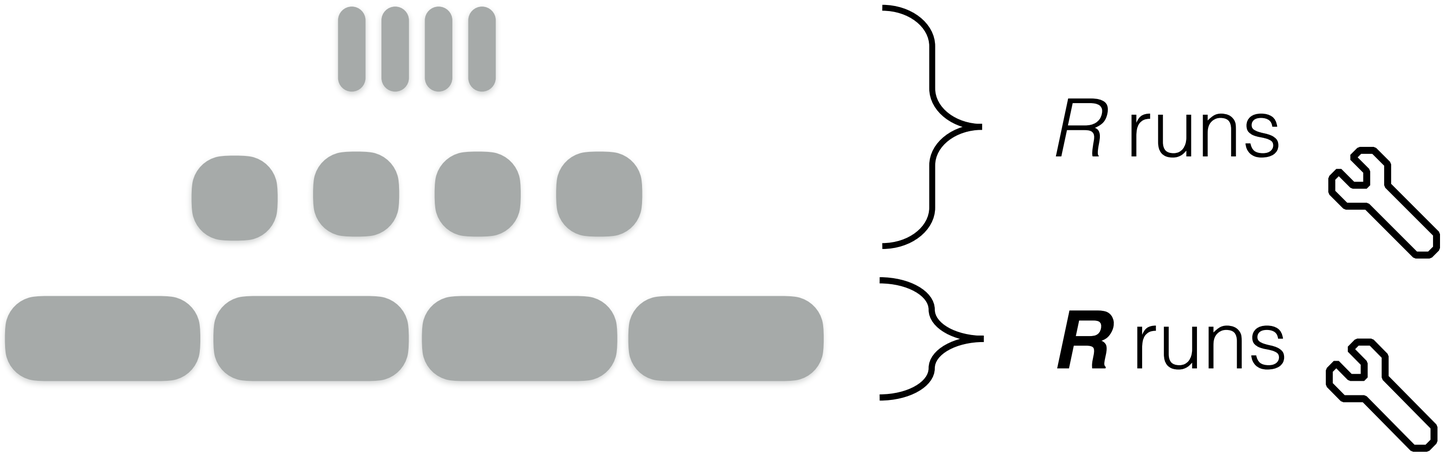
Tiering



optimize

point long range short range merging

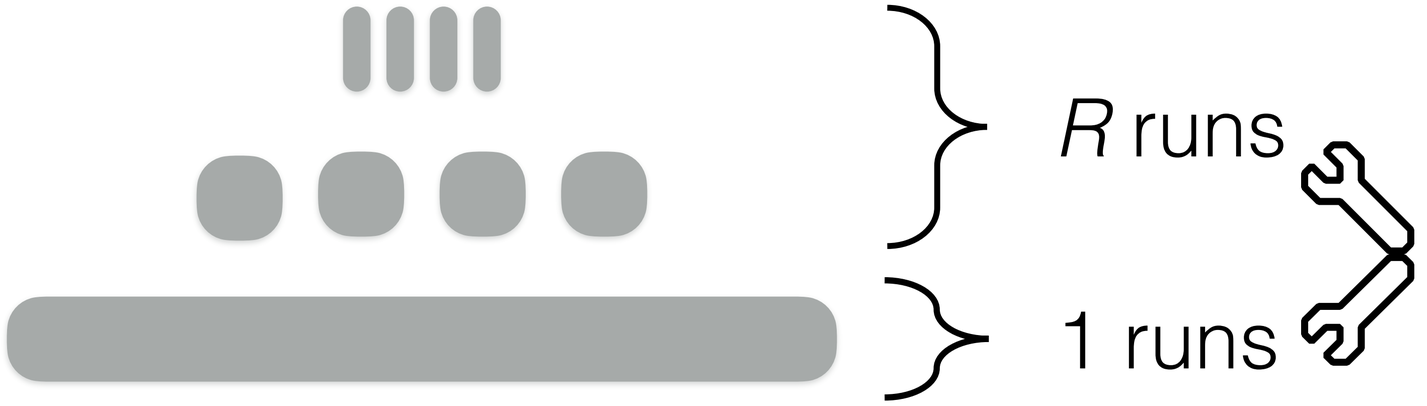
Tiering



optimize

point long range short range merging

Lazy Leveling



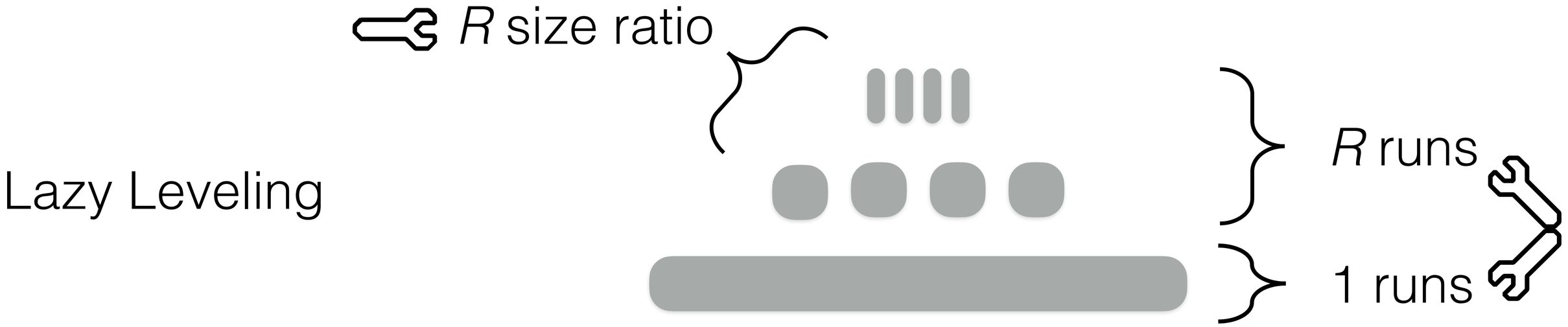
optimize

point

long range

short range

merging



optimize

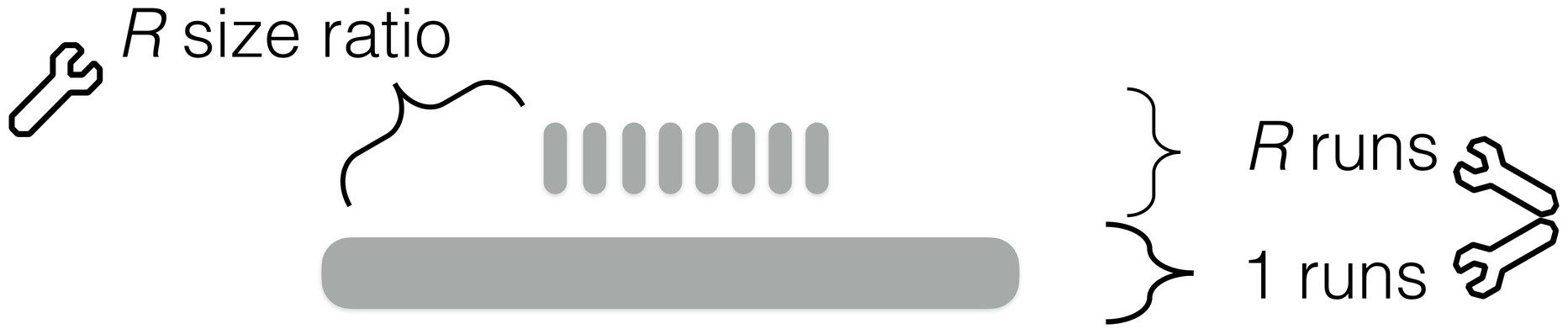
point

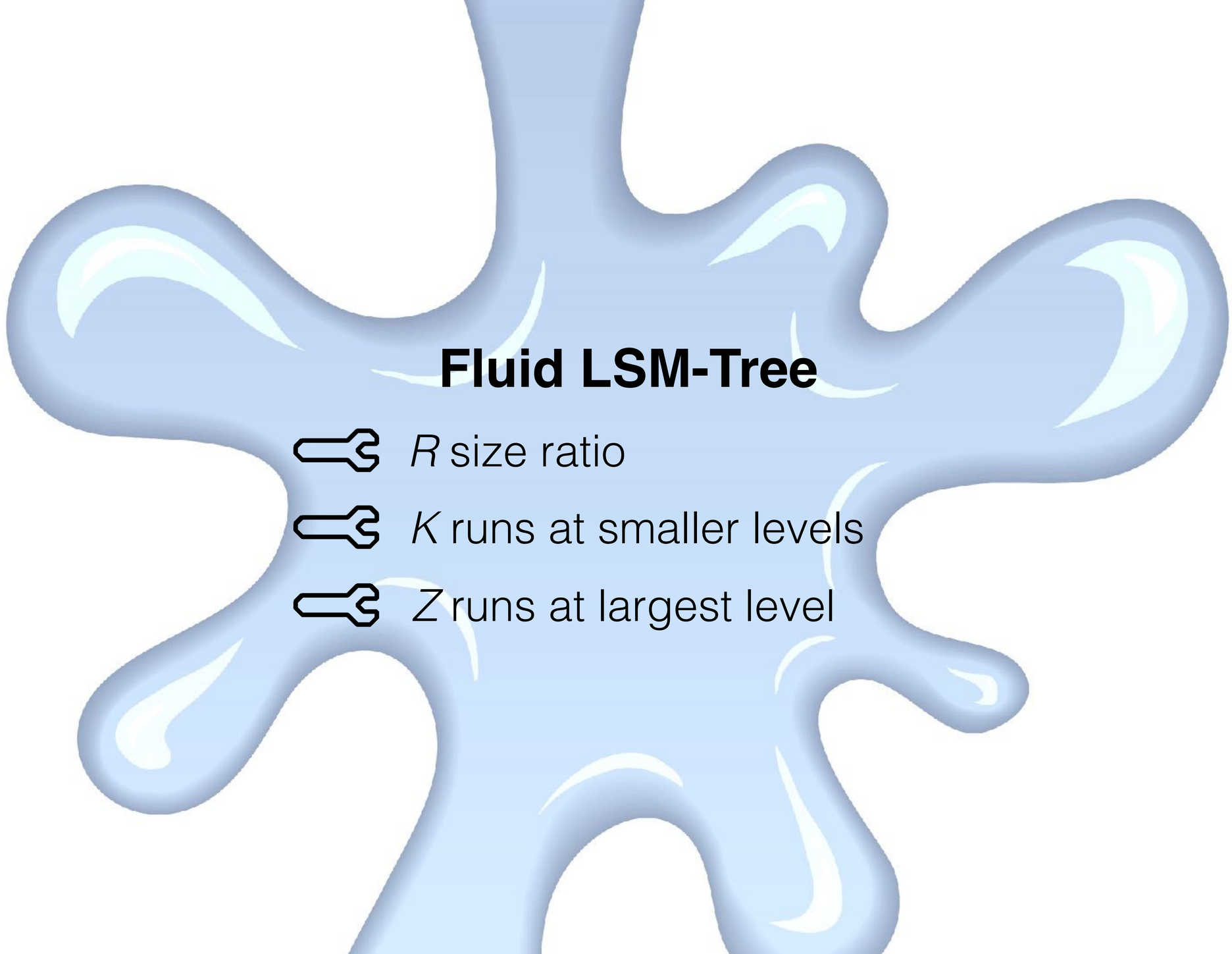
long range

short range

merging

Lazy Leveling

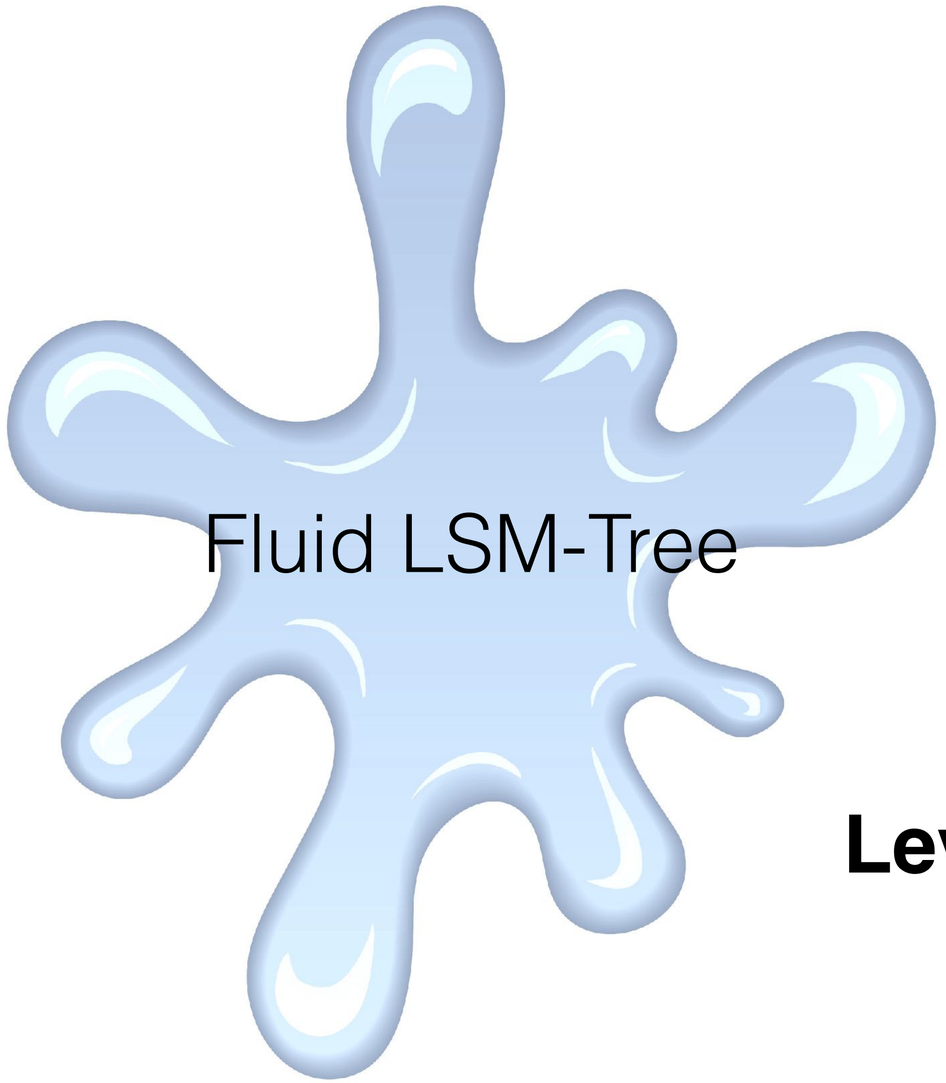




Fluid LSM-Tree

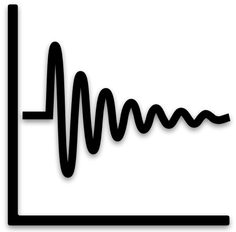
- *R* size ratio
- *K* runs at smaller levels
- *Z* runs at largest level

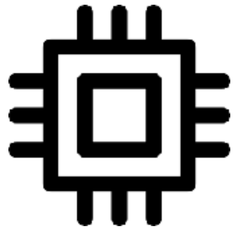
Lazy Leveling

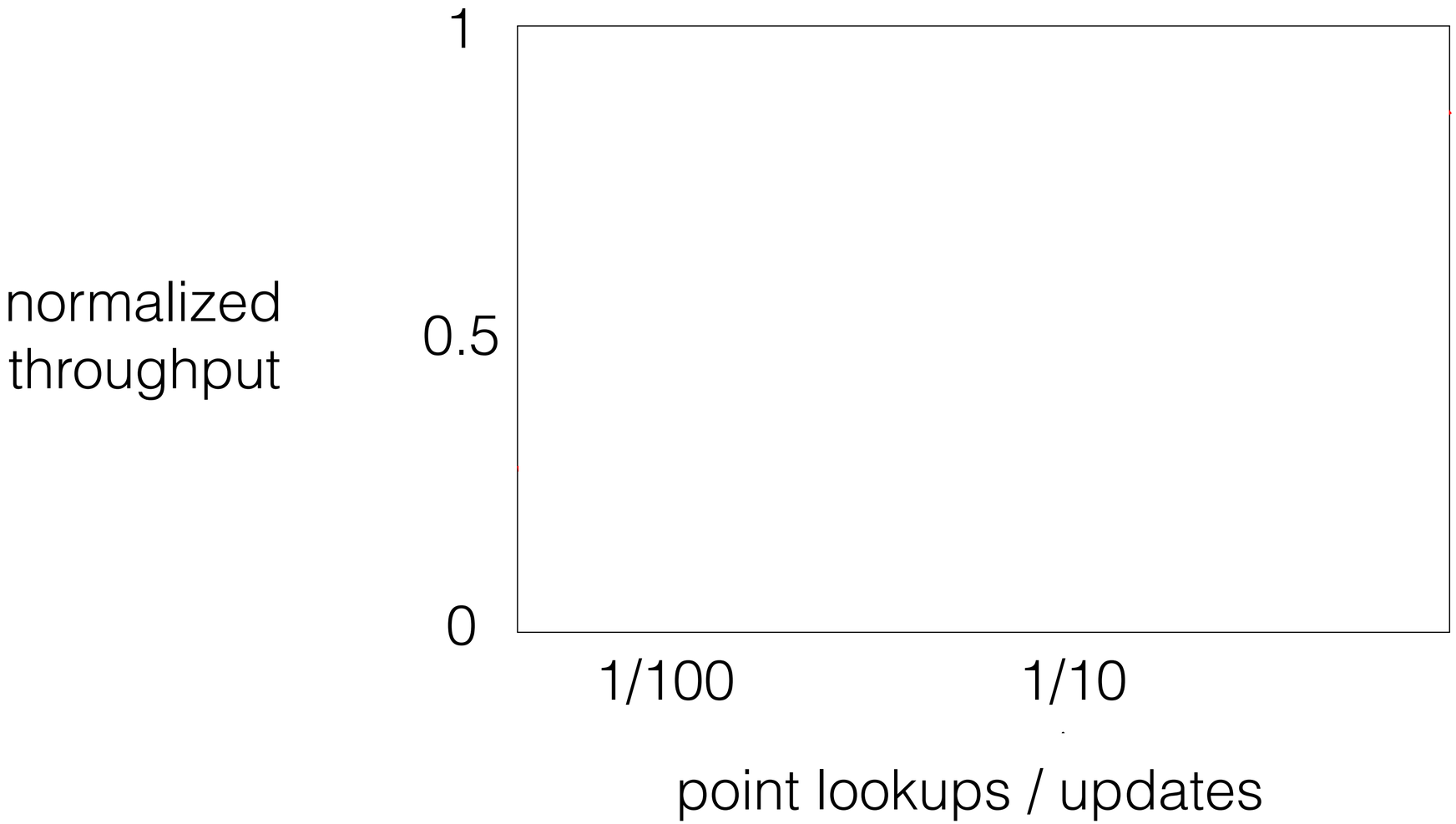


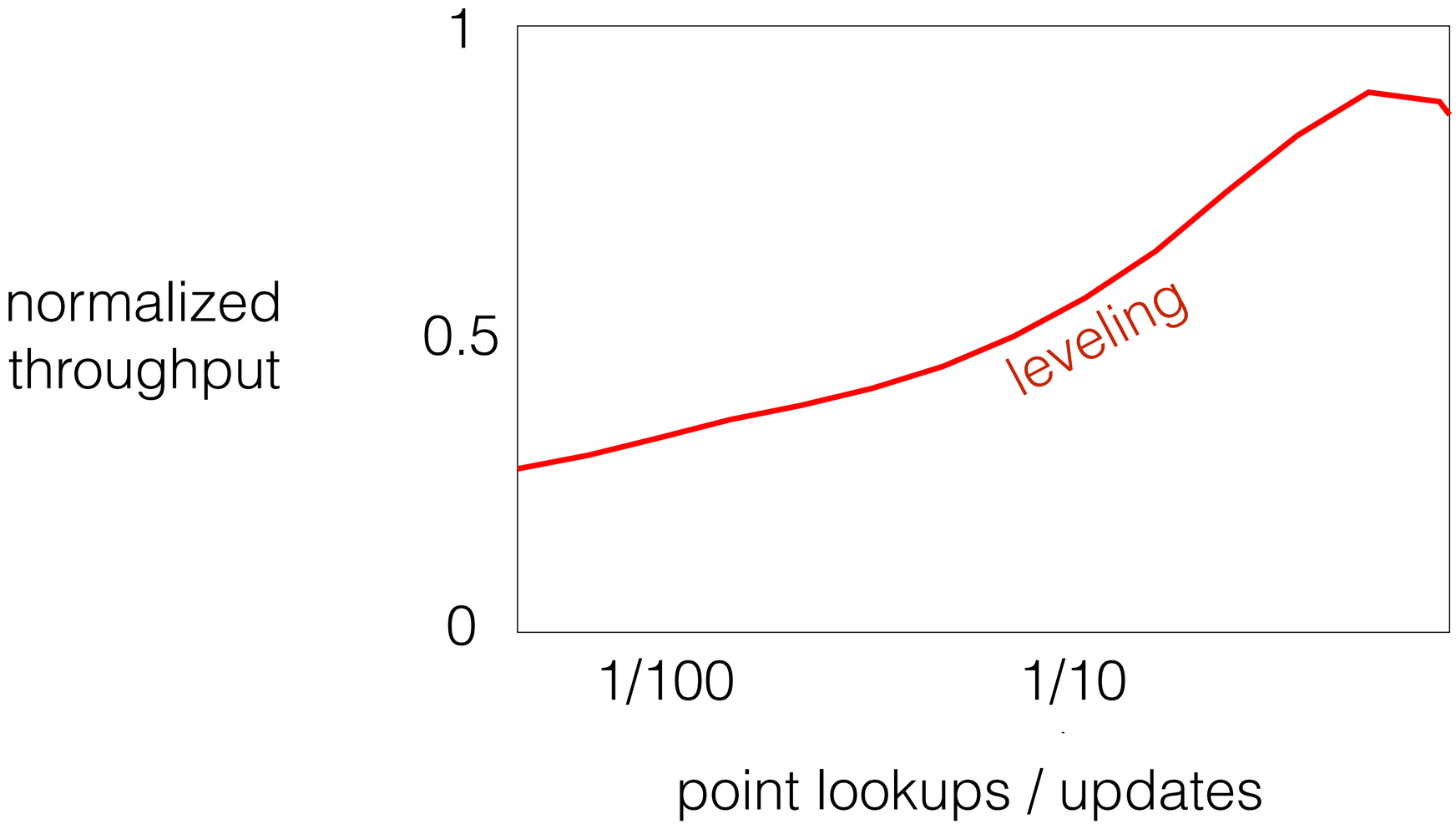
Tiering

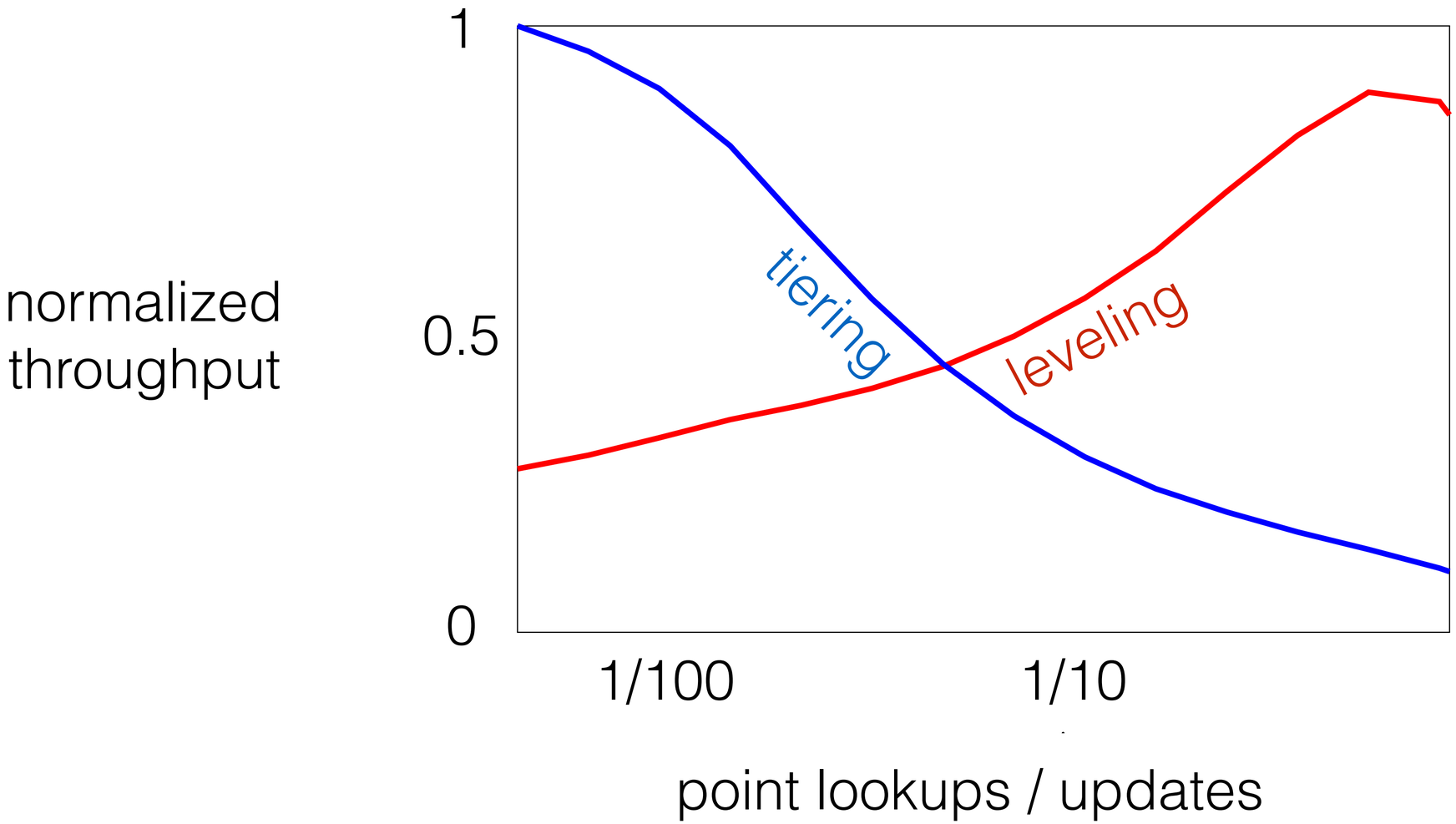
Leveling

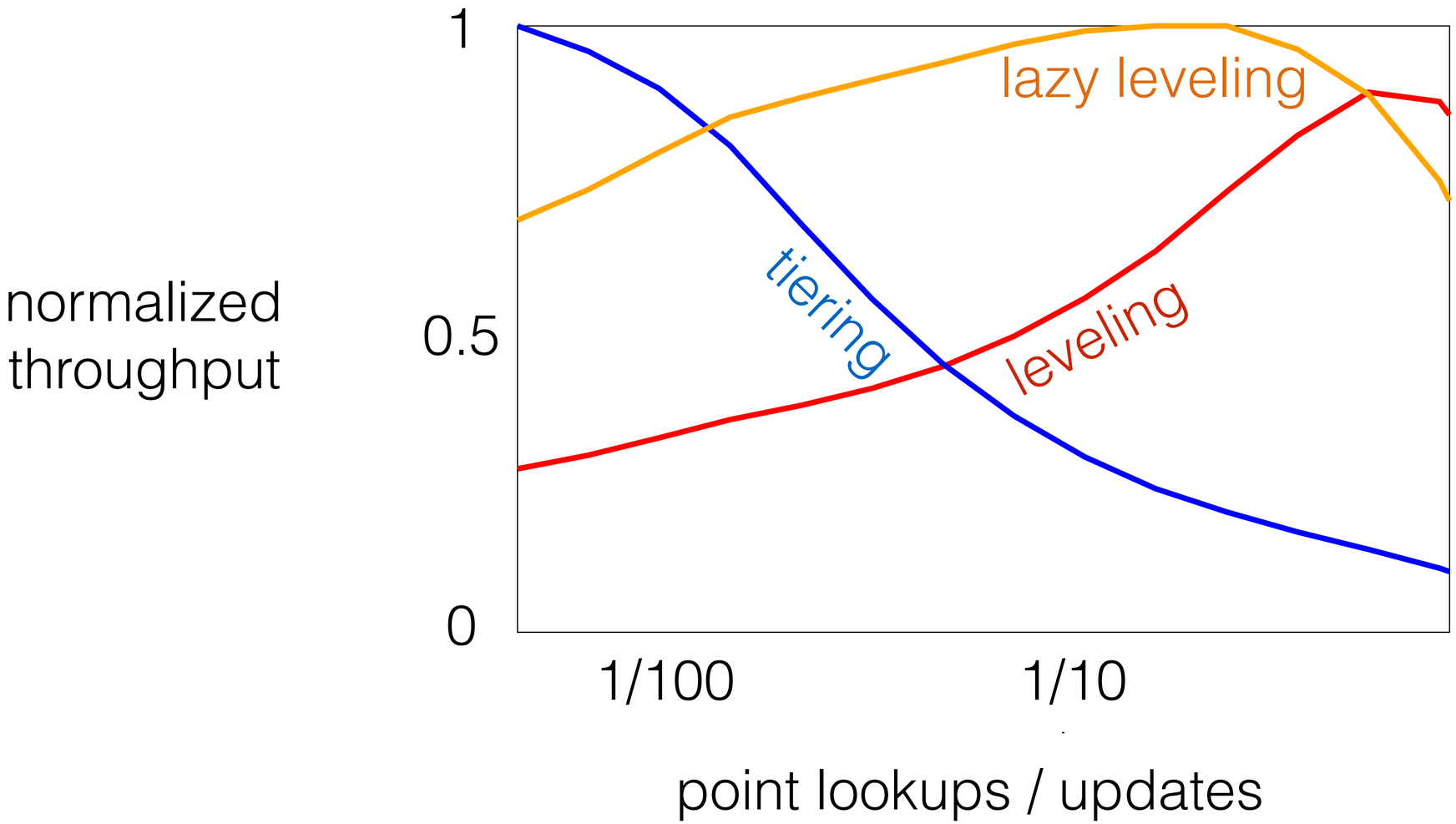


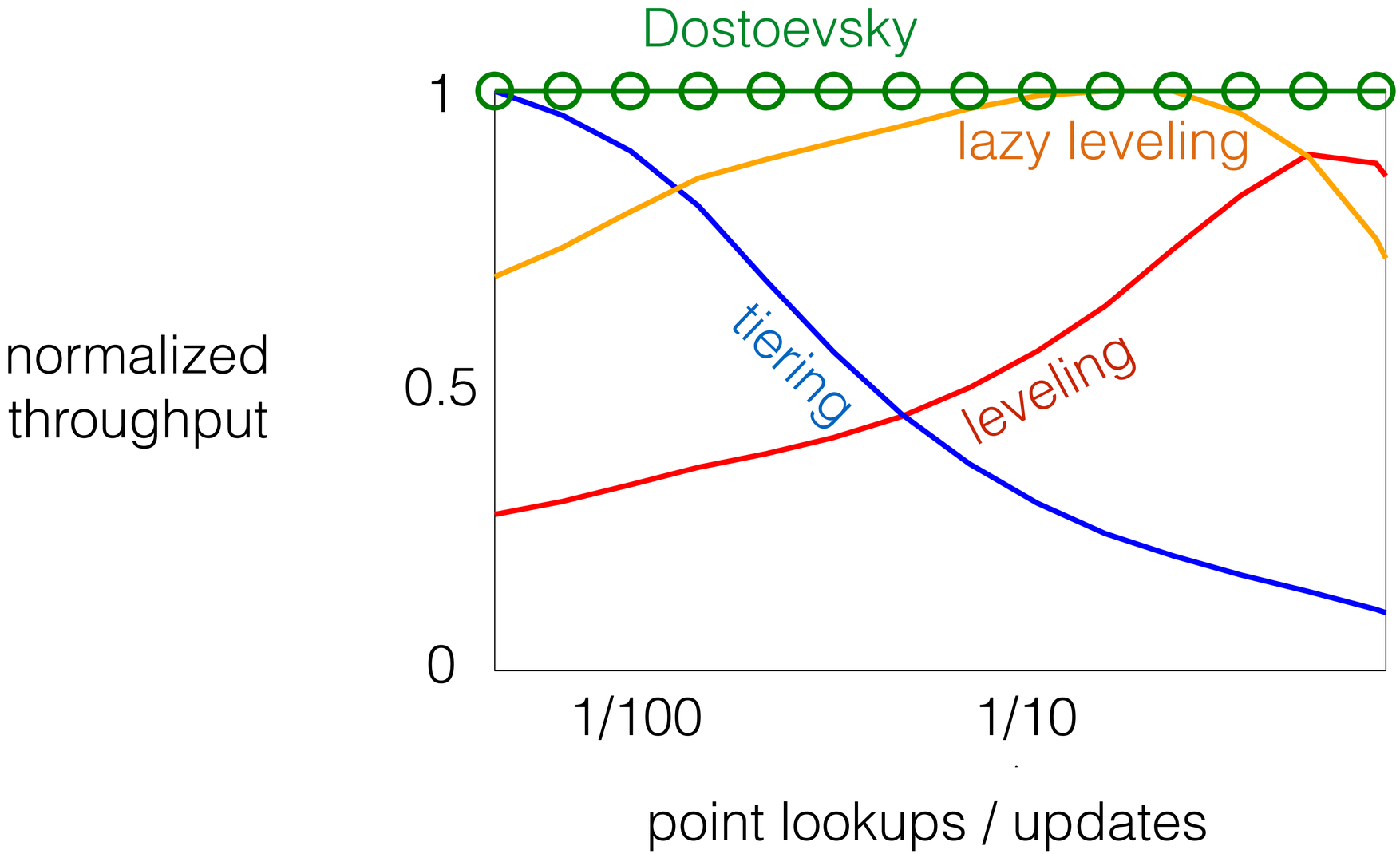


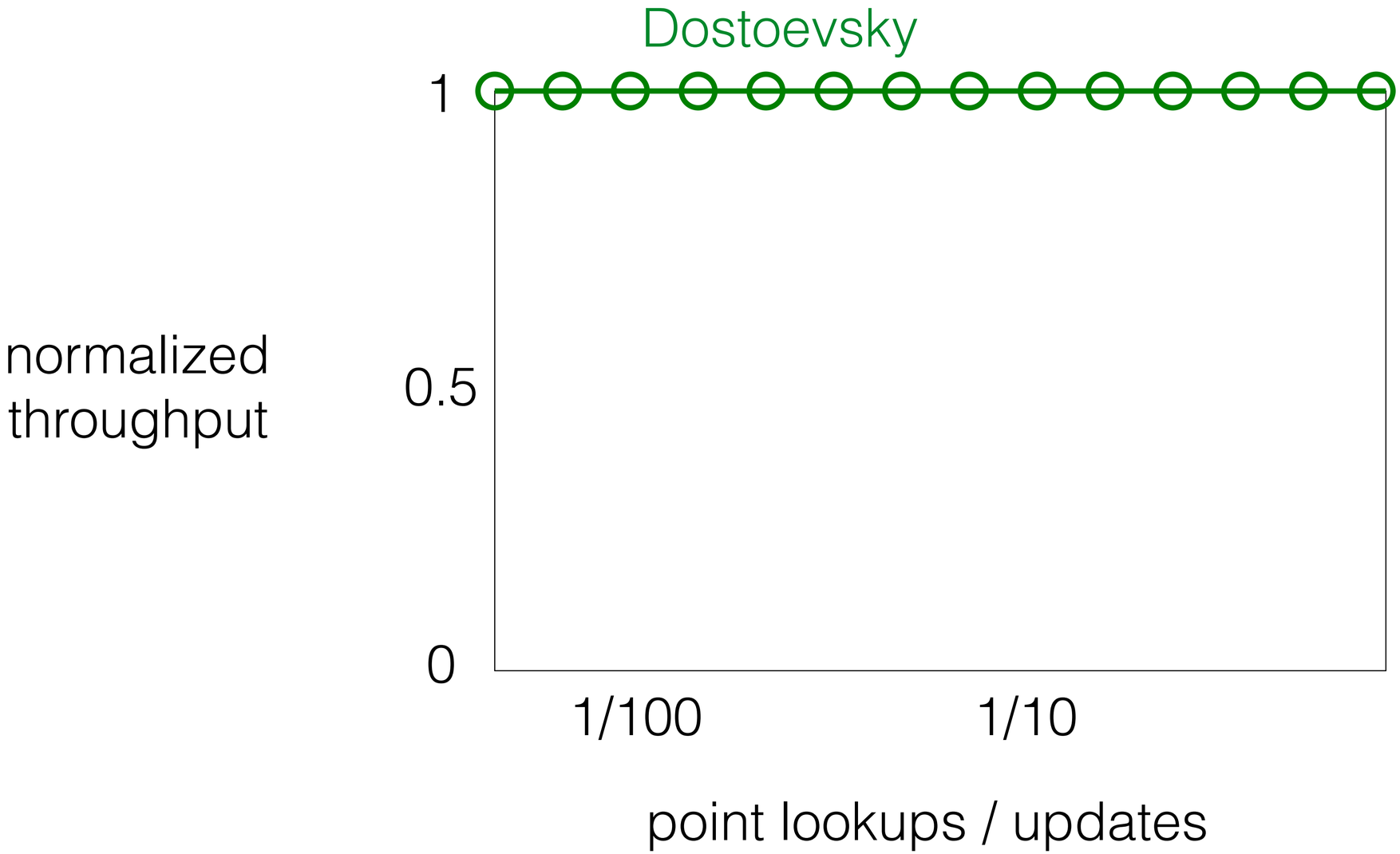

1
0.5
0
normalized
throughput
1/100
1/10
point lookups / updates

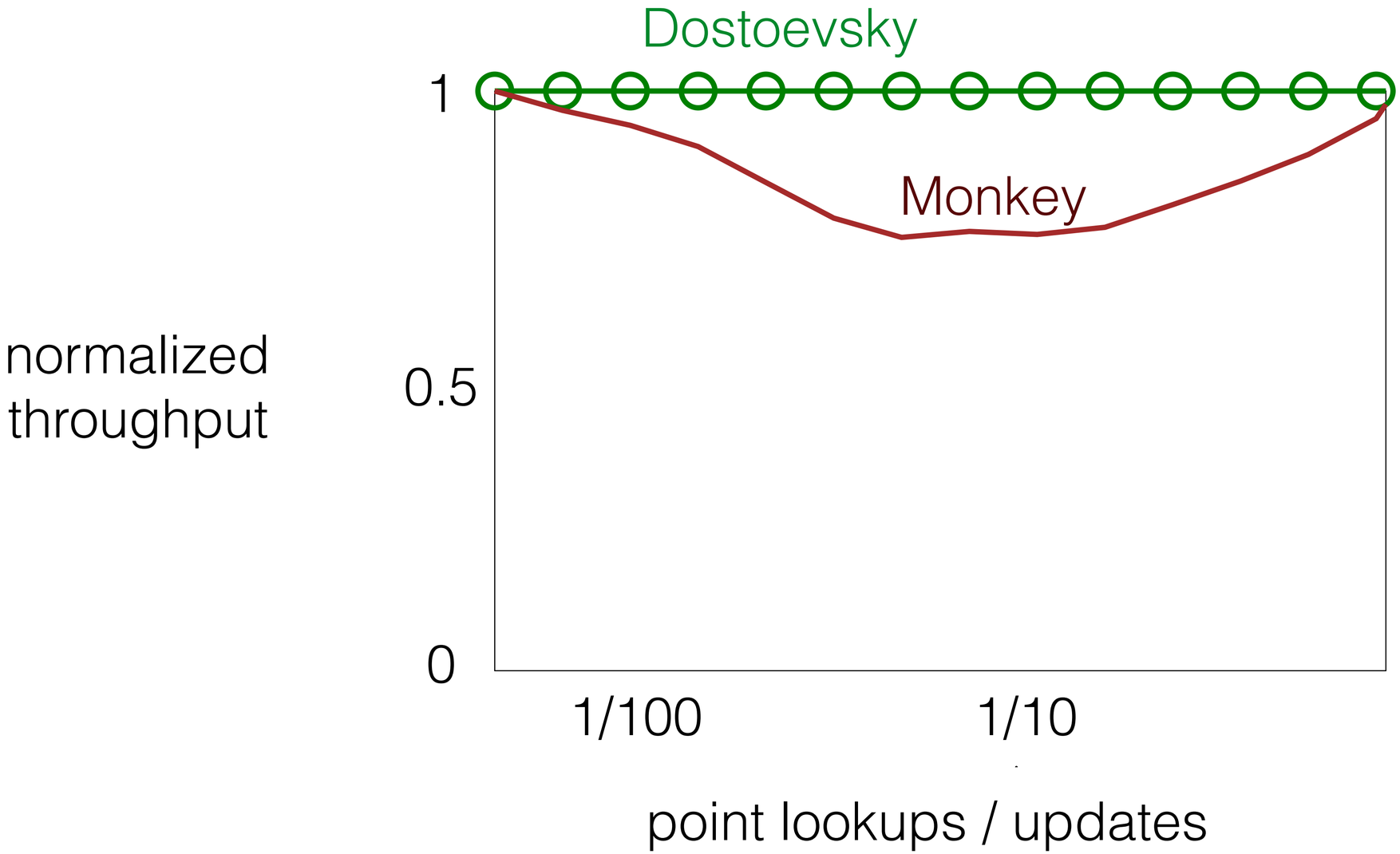

normalized
throughput
1
0.5
0
leveling
1/100
1/10
point lookups / updates

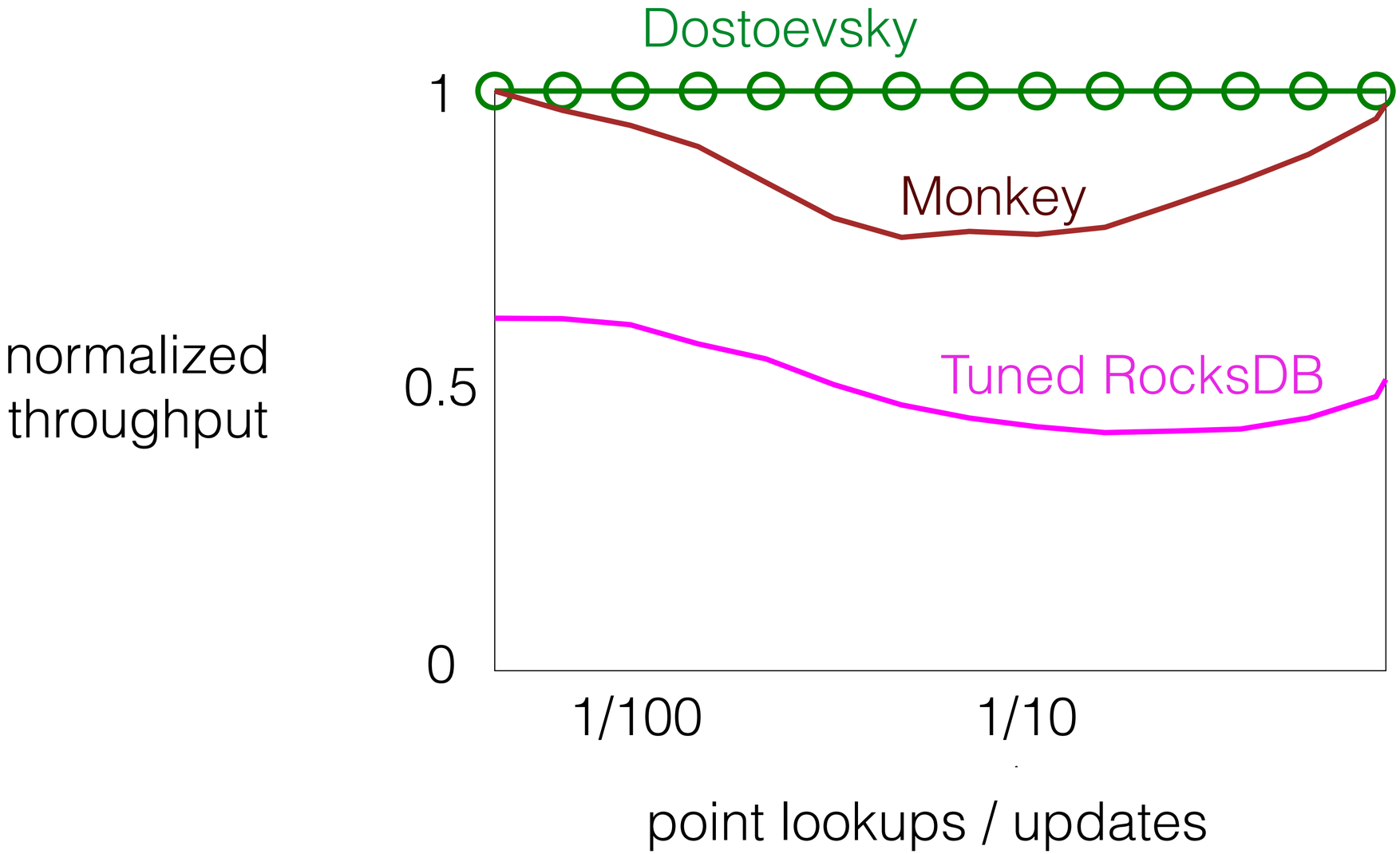

normalized
throughput
1
0.5
0
tiering
leveling
1/100
1/10
point lookups / updates

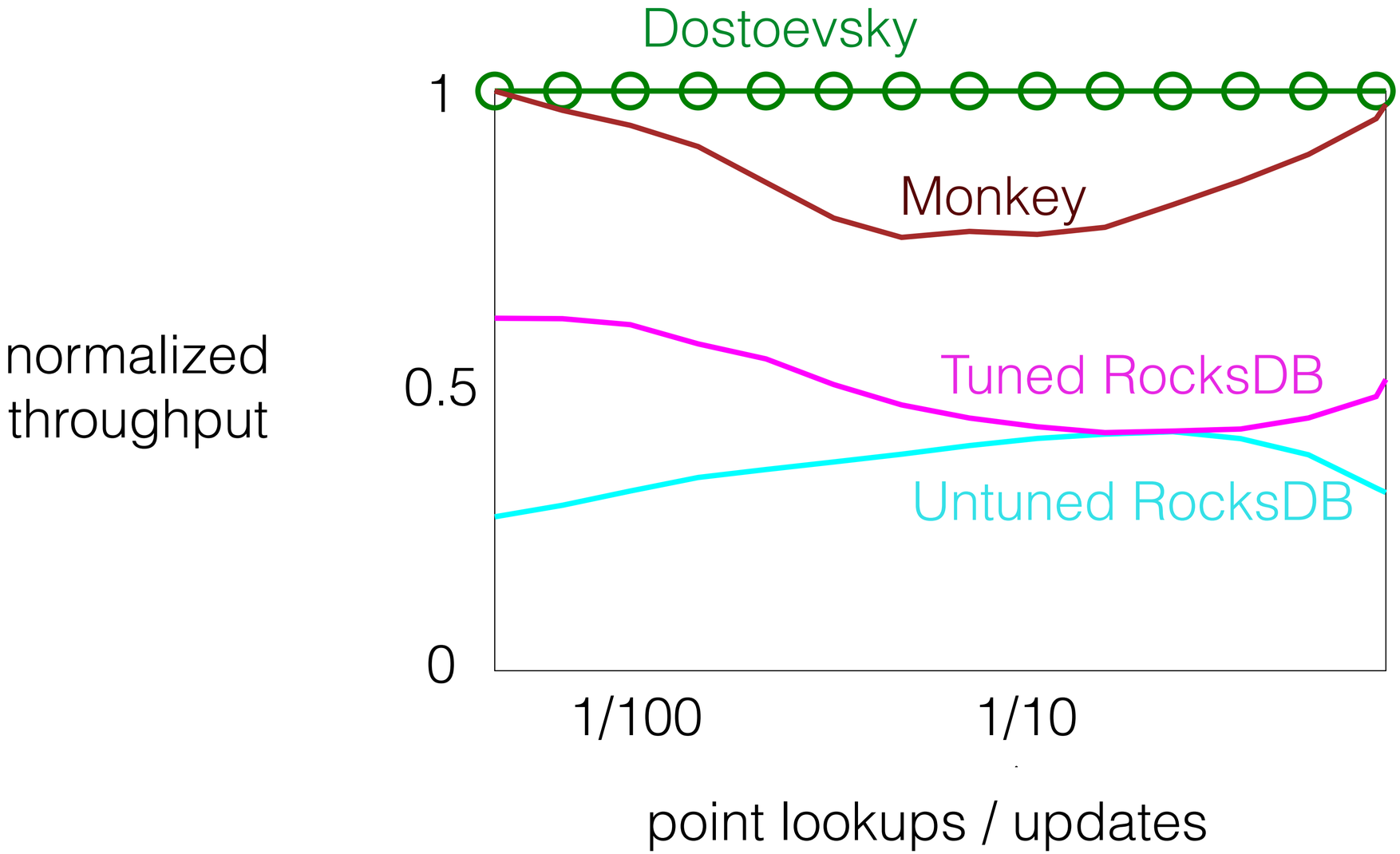

normalized
throughput
1
0.5
0
lazy leveling
tiering
leveling
1/100
1/10
point lookups / updates


Dostoevsky
1
lazy leveling
tiering
leveling
0.5
0
1/100
1/10
normalized
throughput
point lookups / updates


Dostoevsky
1
0.5
0
normalized
throughput
1/100
1/10
point lookups / updates


Dostoevsky
Monkey
1
0.5
0
normalized
throughput
1/100
1/10
point lookups / updates


Dostoevsky
1
Monkey
normalized
throughput
0.5
Tuned RocksDB
0
1/100
1/10
point lookups / updates


Dostoevsky
1
Monkey
normalized
throughput
0.5
Tuned RocksDB
Untuned RocksDB
0
1/100
1/10
point lookups / updates

Conclusion

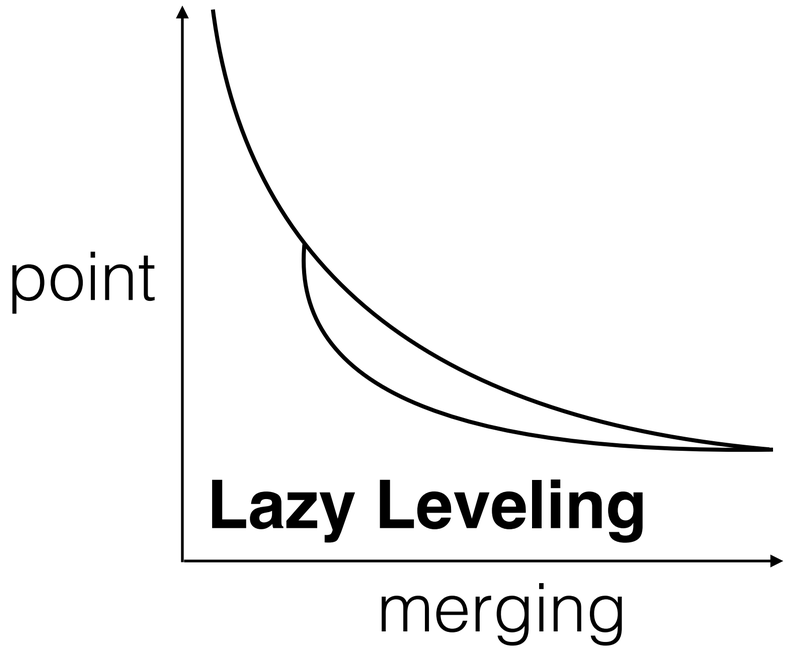
Conclusion

Lazy Leveling

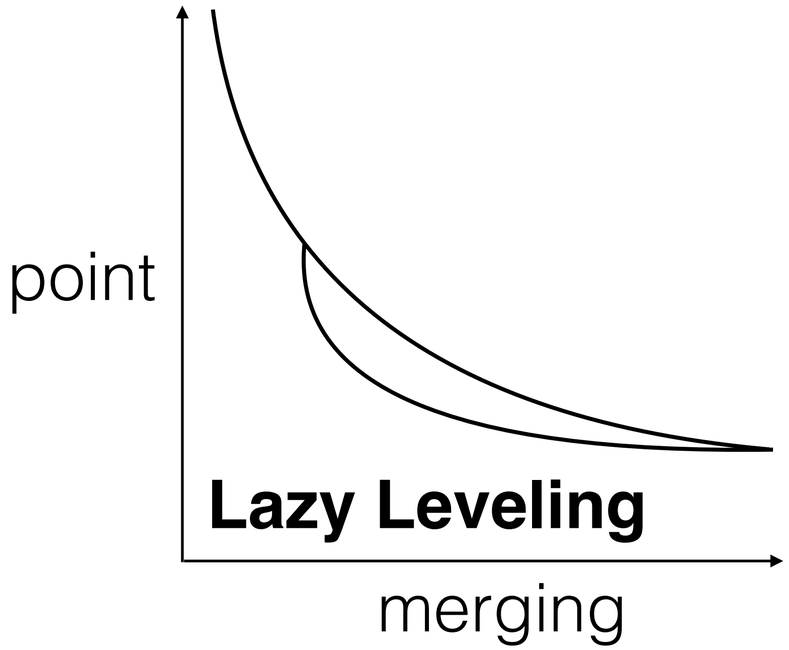
no merging

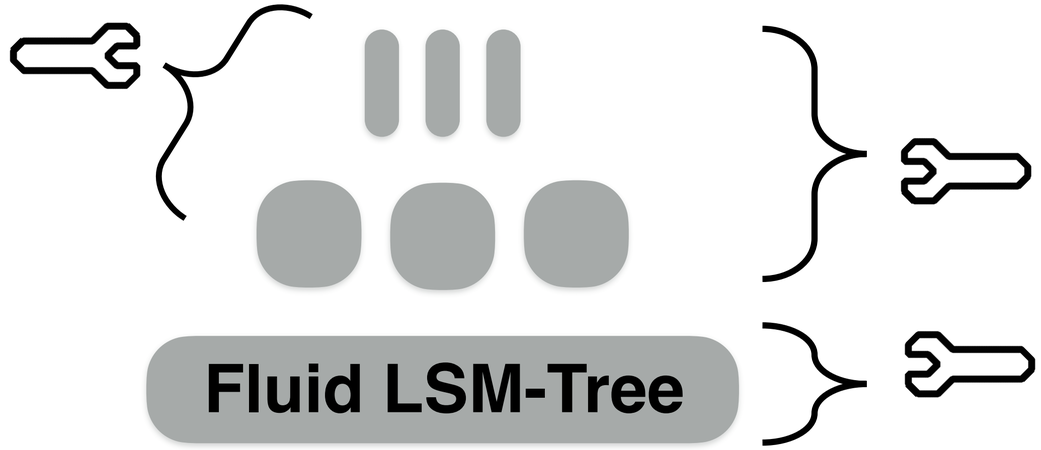
greedy merging

Conclusion

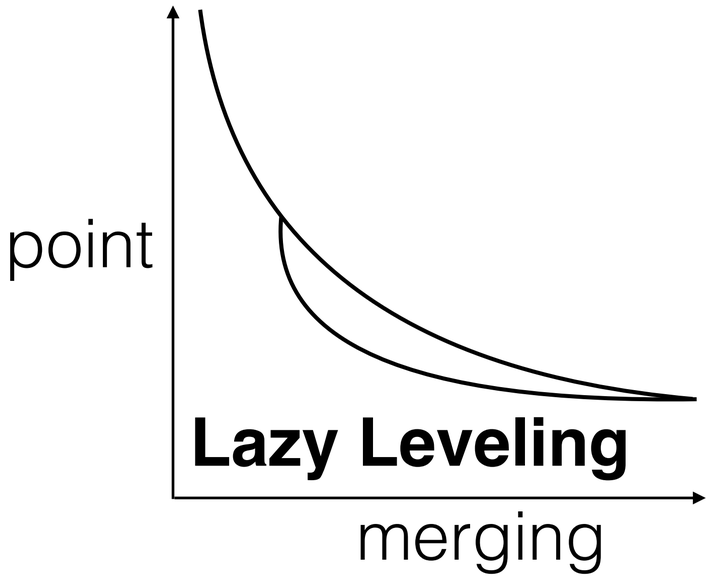


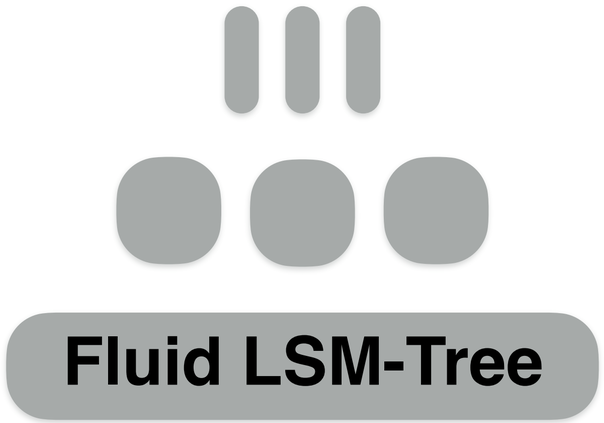
Conclusion





Conclusion





Dostoevsky

Dostoevsky: **S**pace-**T**ime **O**ptimized **Ev**olvable **S**calable **K**e**y**-Value Store

VERY WRITE-OPTIMIZED

Thanks!

CrimsonDB

A self-designing key-value store

www.crimsondb.org