Approximate Indexing with BF-Trees*
A RUM access method

Manos Athanassoulis*
Harvard SEAS

Anastasia Ailamaki
EPFL

HARVARD UNIVERSITY

*work done while at EPFL

EPFL

Tree indexing

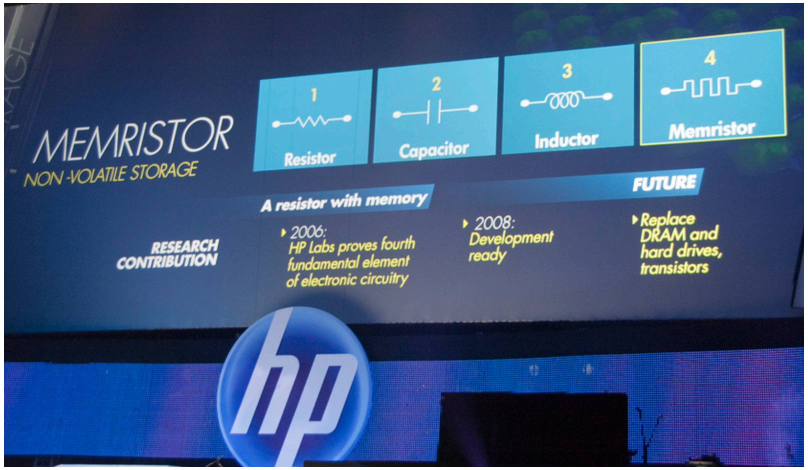
wide and short trees

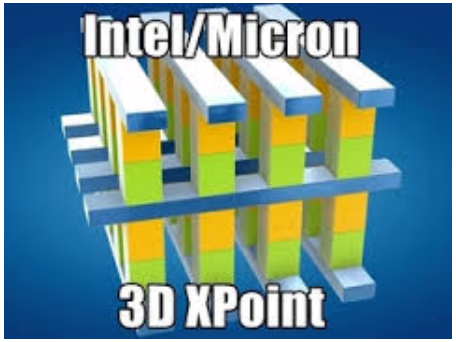
… which have large size in order to minimizing random accesses

This is not enough!

… is designed for disks

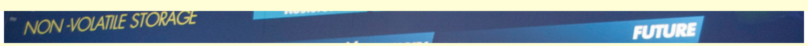
A brave new (storage) world

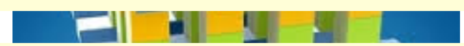




A brave new (storage) world

update cost varies

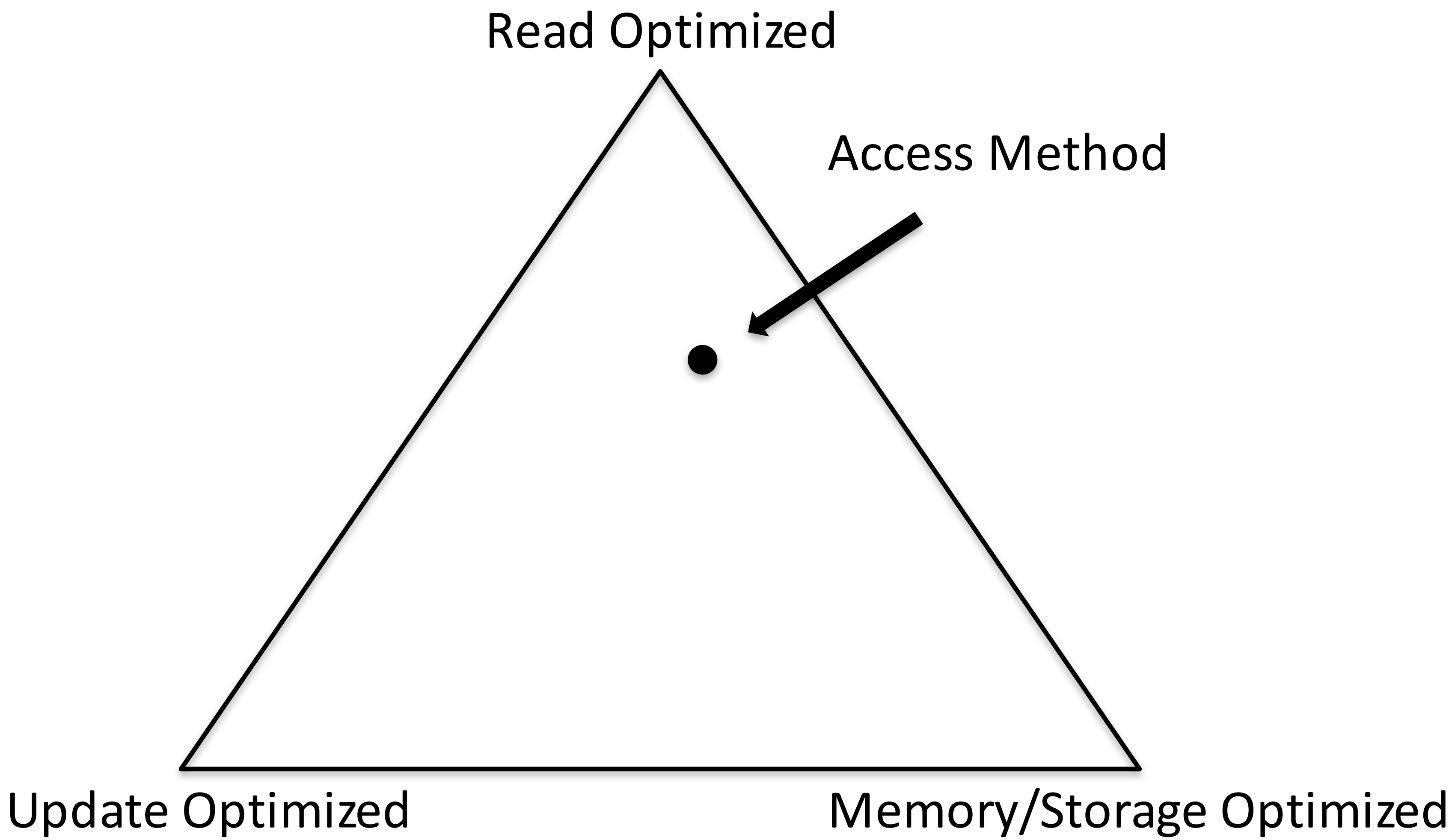




read performance varies

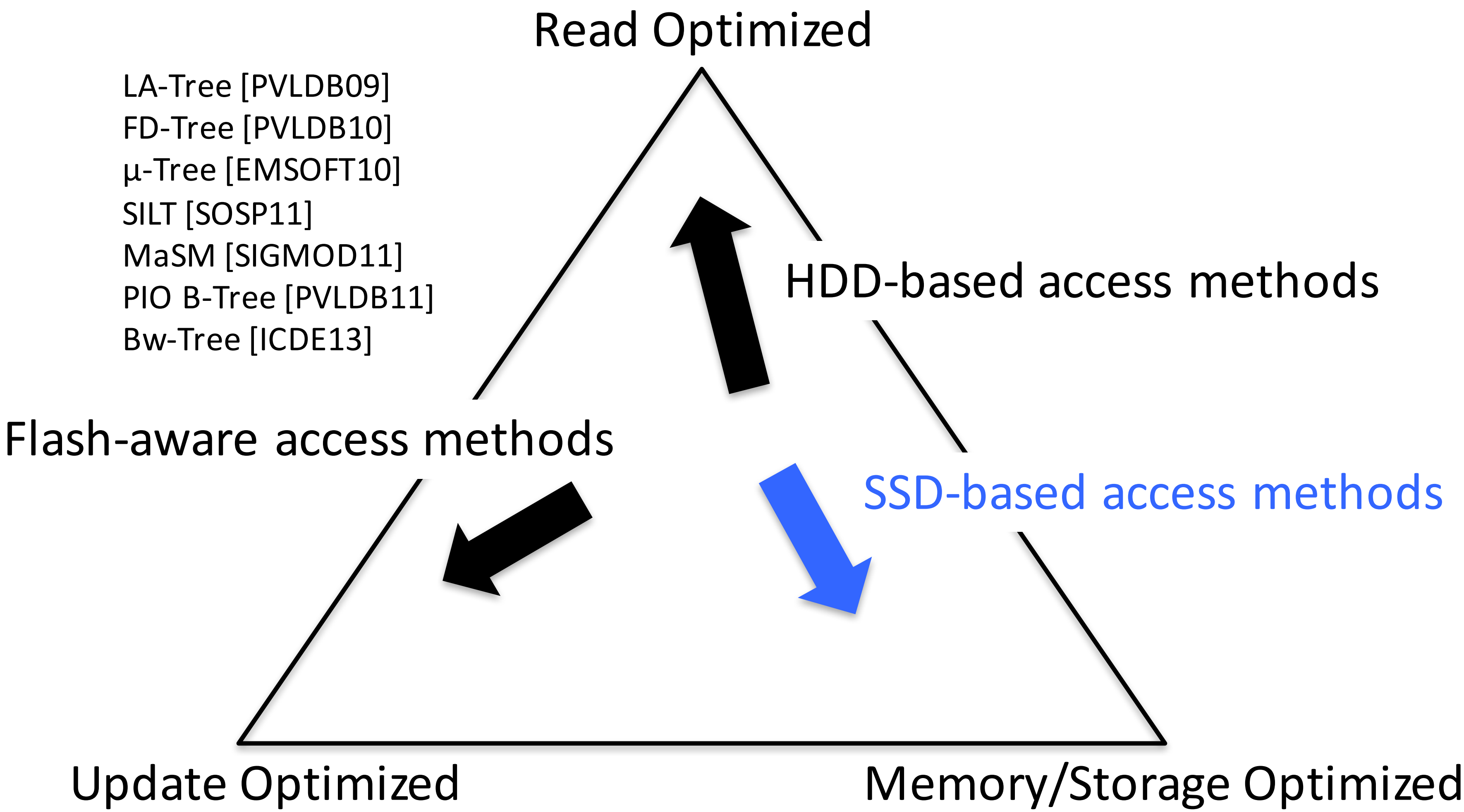
memory price varies

RUM: Read vs Update vs Memory

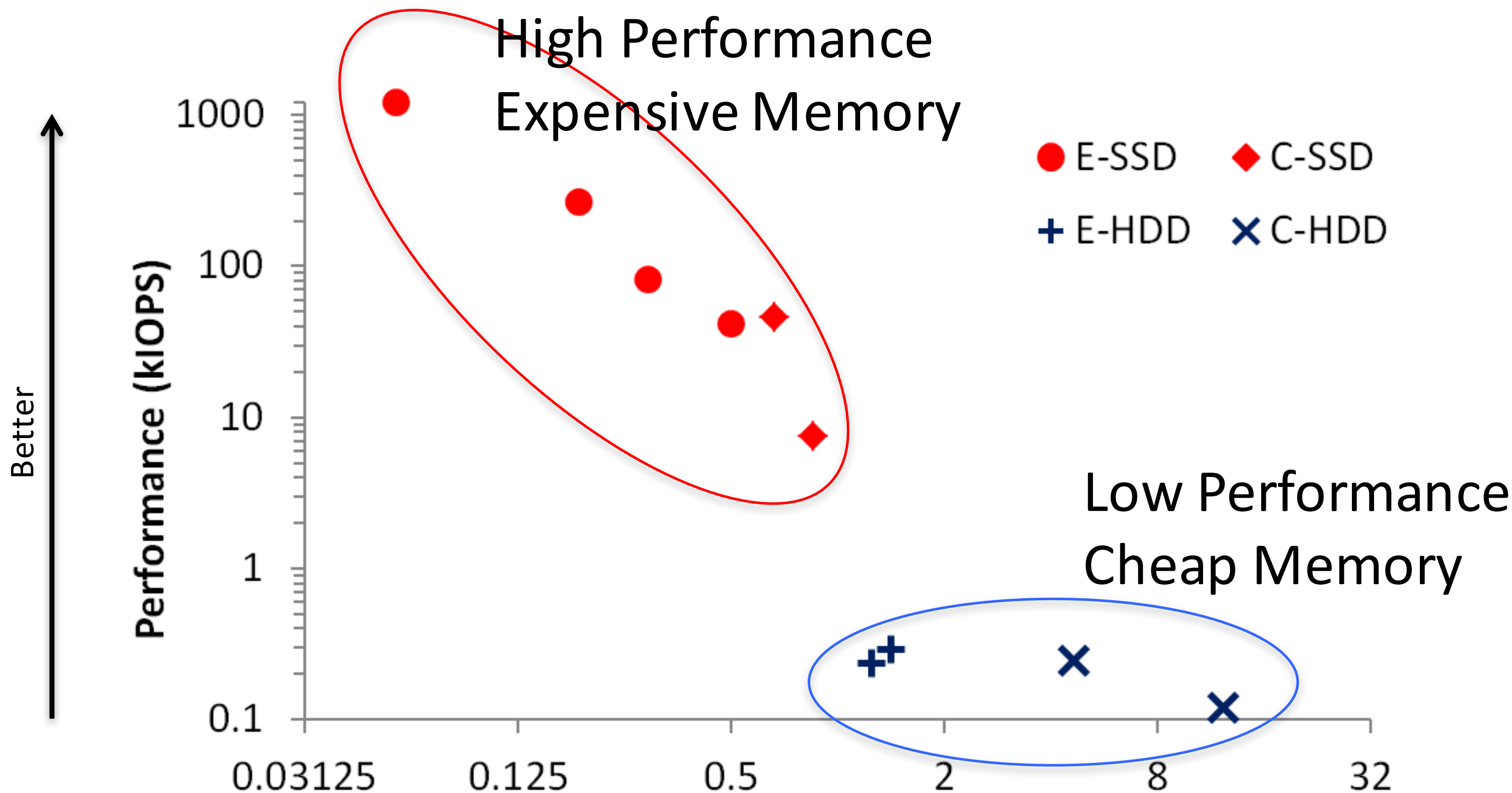


RUM: Read vs Update vs Memory

LA-Tree [PVLDB09]
FD-Tree [PVLDB10]
μ-Tree [EMSOFT10]
SILT [SOSP11]
MaSM [SIGMOD11]
PIO B-Tree [PVLDB11]
Bw-Tree [ICDE13]



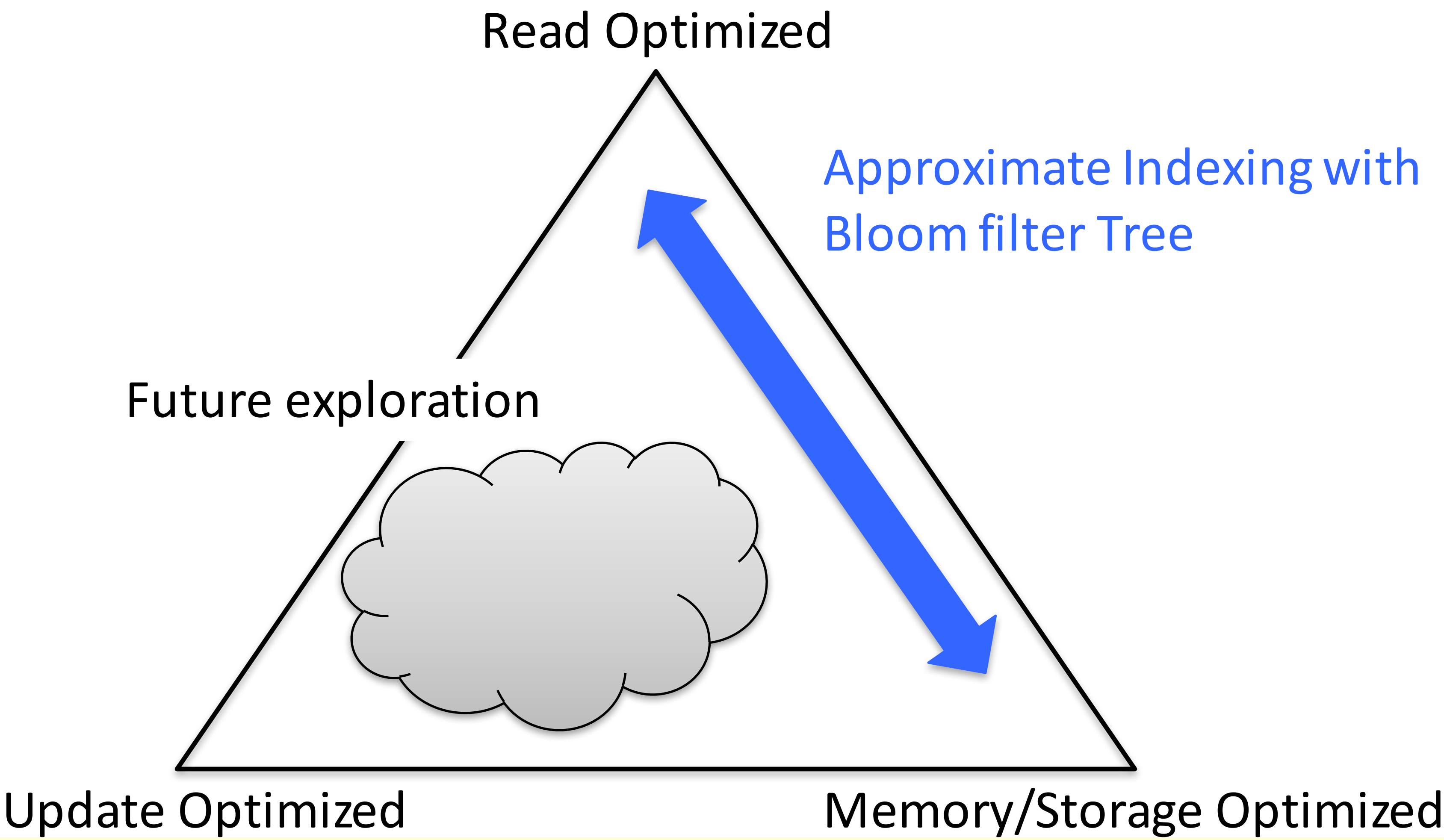
Memory Price vs. Reads



exchange more reads for lower size

rethink indexing!

RUM: Read vs Update vs Memory



let's see how BF-Tree works

Flash-aware indexing

focuses on internal node organization

lazy updates

immutable data

LA-Tree [PVLDB09]
FD-Tree [PVLDB10]
μ-Tree [EMSOFT10]
SILT [SOSP11]
MaSM [SIGMOD11]
PIO B-Tree [PVLDB11]
Bw-Tree [ICDE13]

what about solid-state storage *fast reads*?

Approximate Tree Indexing

Design choices

use *Bloom filters* for membership queries per page

tunable tree size

probabilistically tunable random reads

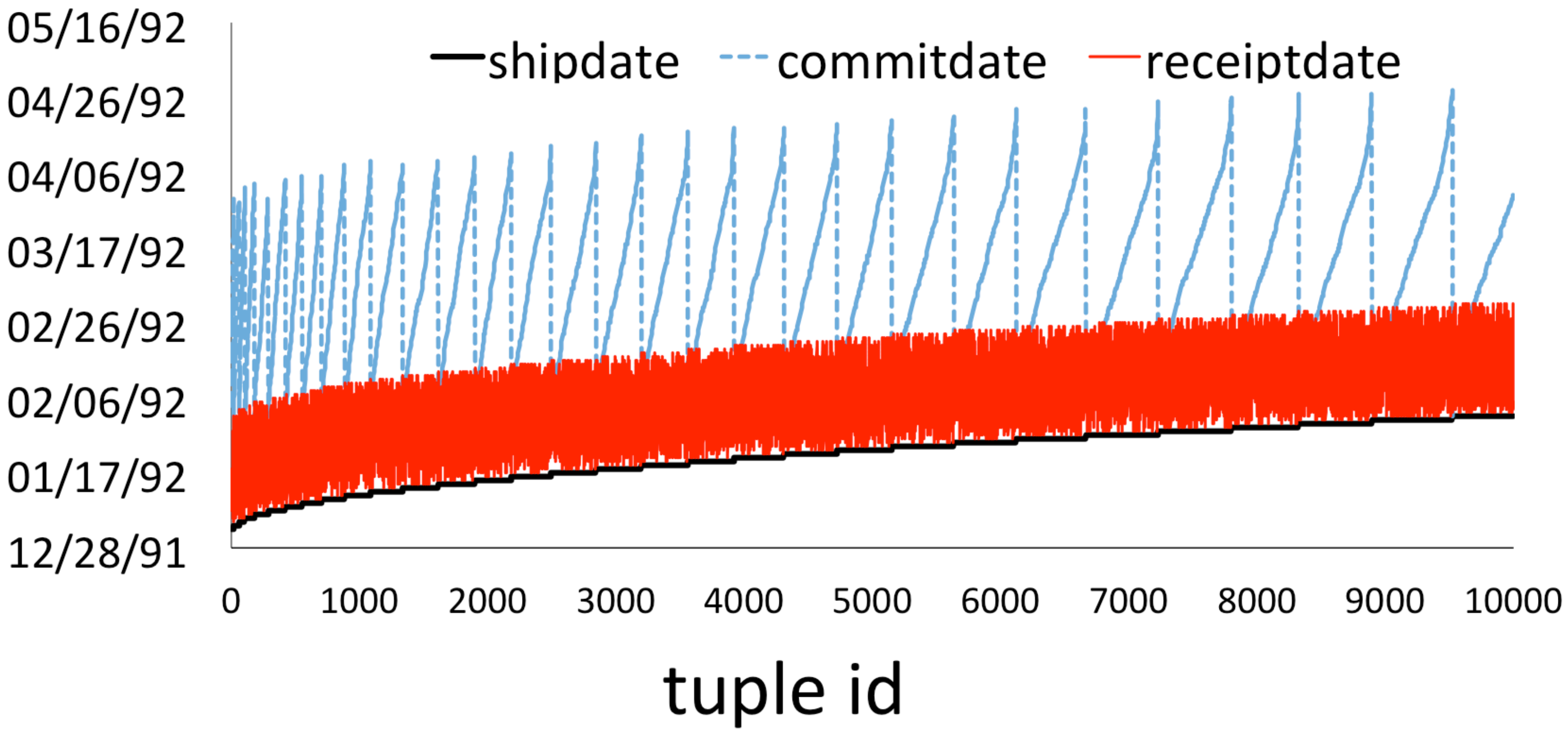
Caveat

works well for datasets with implicit clustering

how common is *implicit clustering*?

Implicit Clustering

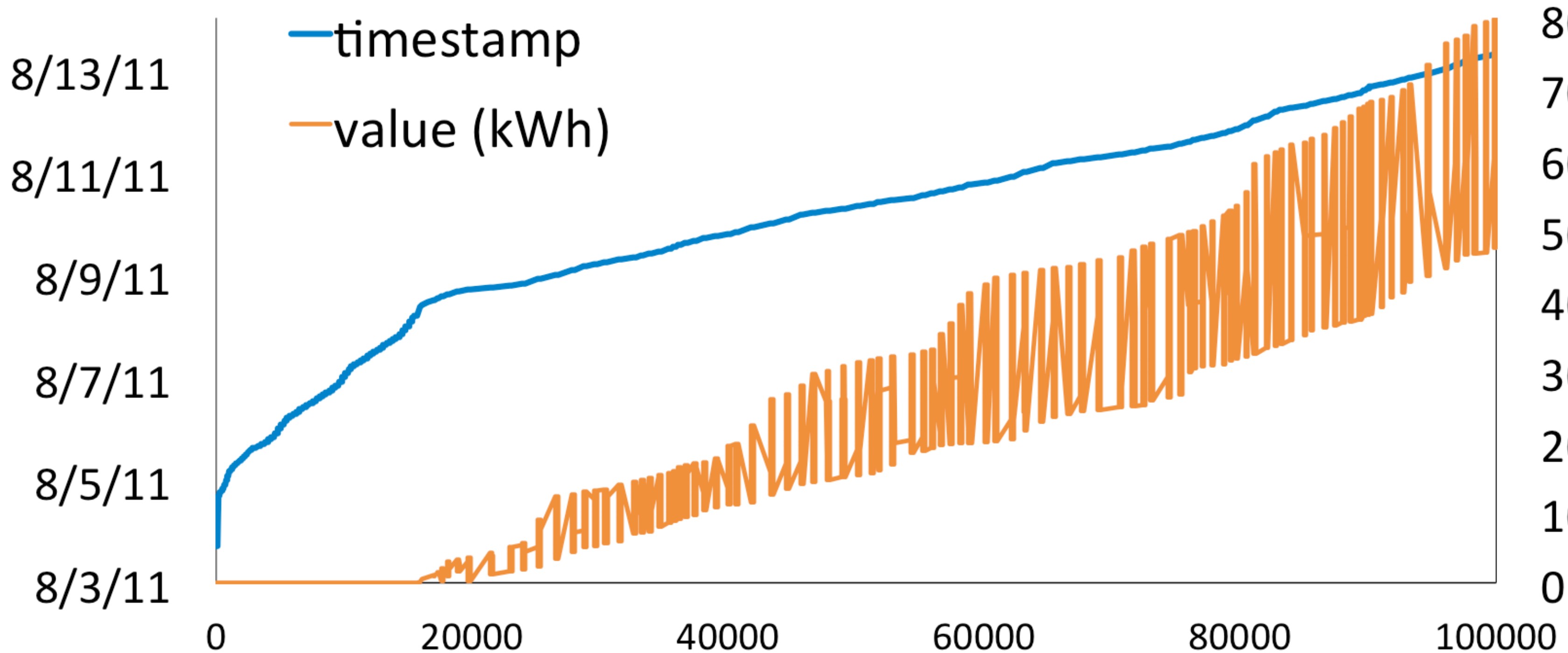
TPCH data: transaction dates



data organized based on creation time

Implicit Clustering

Electricity consumption – Smart Home Dataset (SHD)



how can BF-Tree index such data?

data values correlated with creation time

Bloom filter Trees

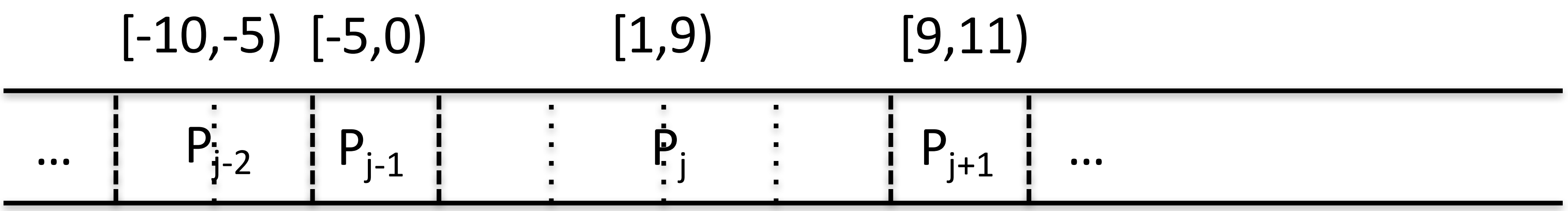
select desired tree size (aka BF per page size)

each partition

has about the same *unique* values

has a partition-wide *min* and *max*

has a (variable) number of physical pages



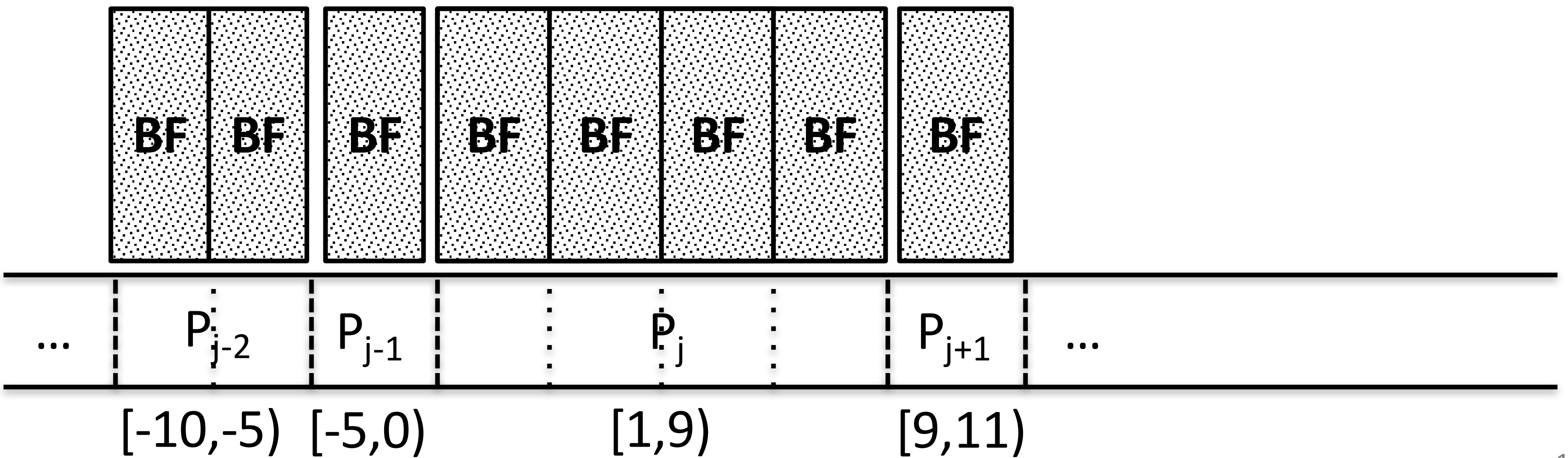
Bloom filter Trees

for every page of every partition

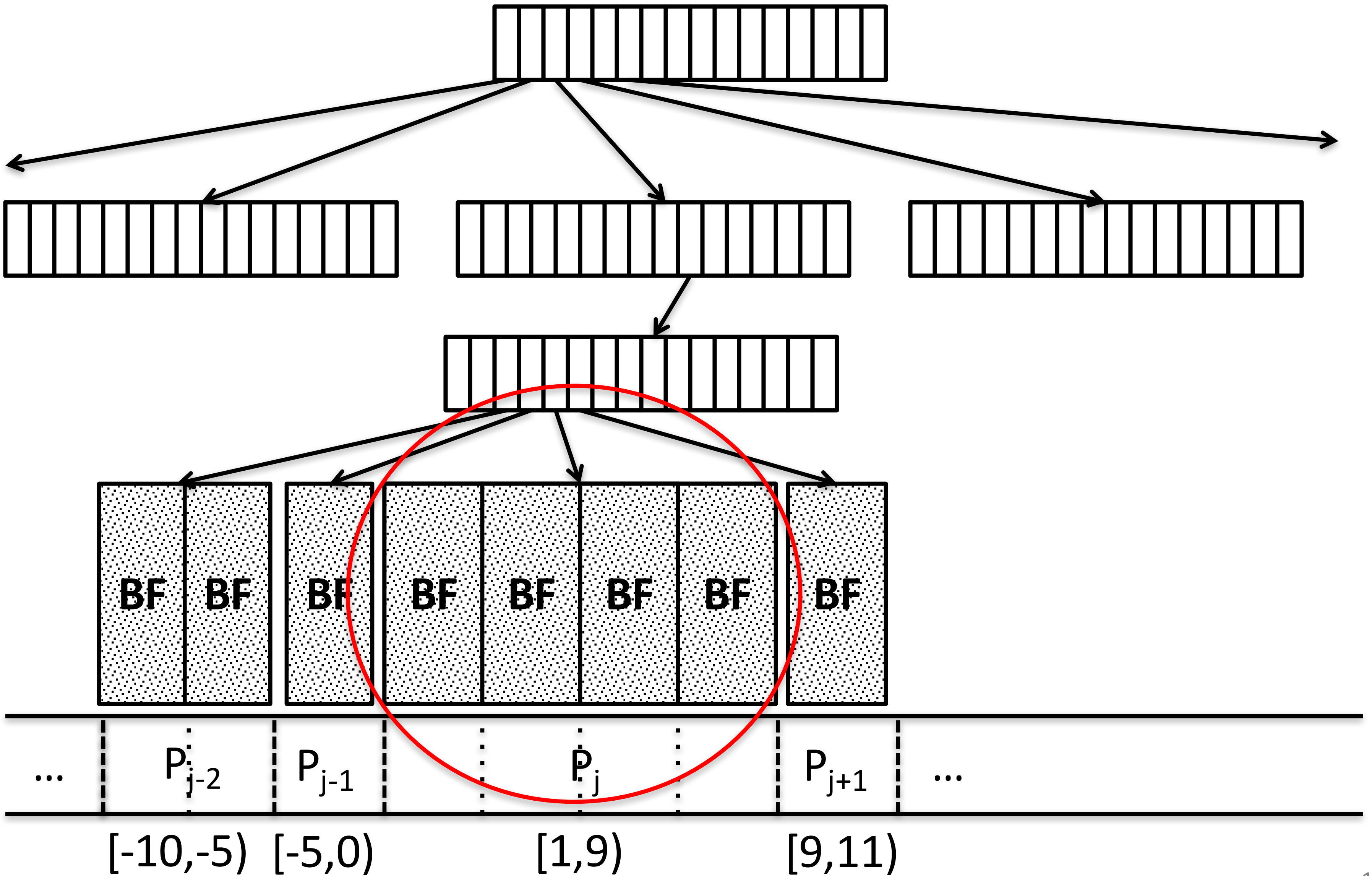
build BF with desired size (and, hence, *false positive*)

build a B^+-Tree on top of all partitions

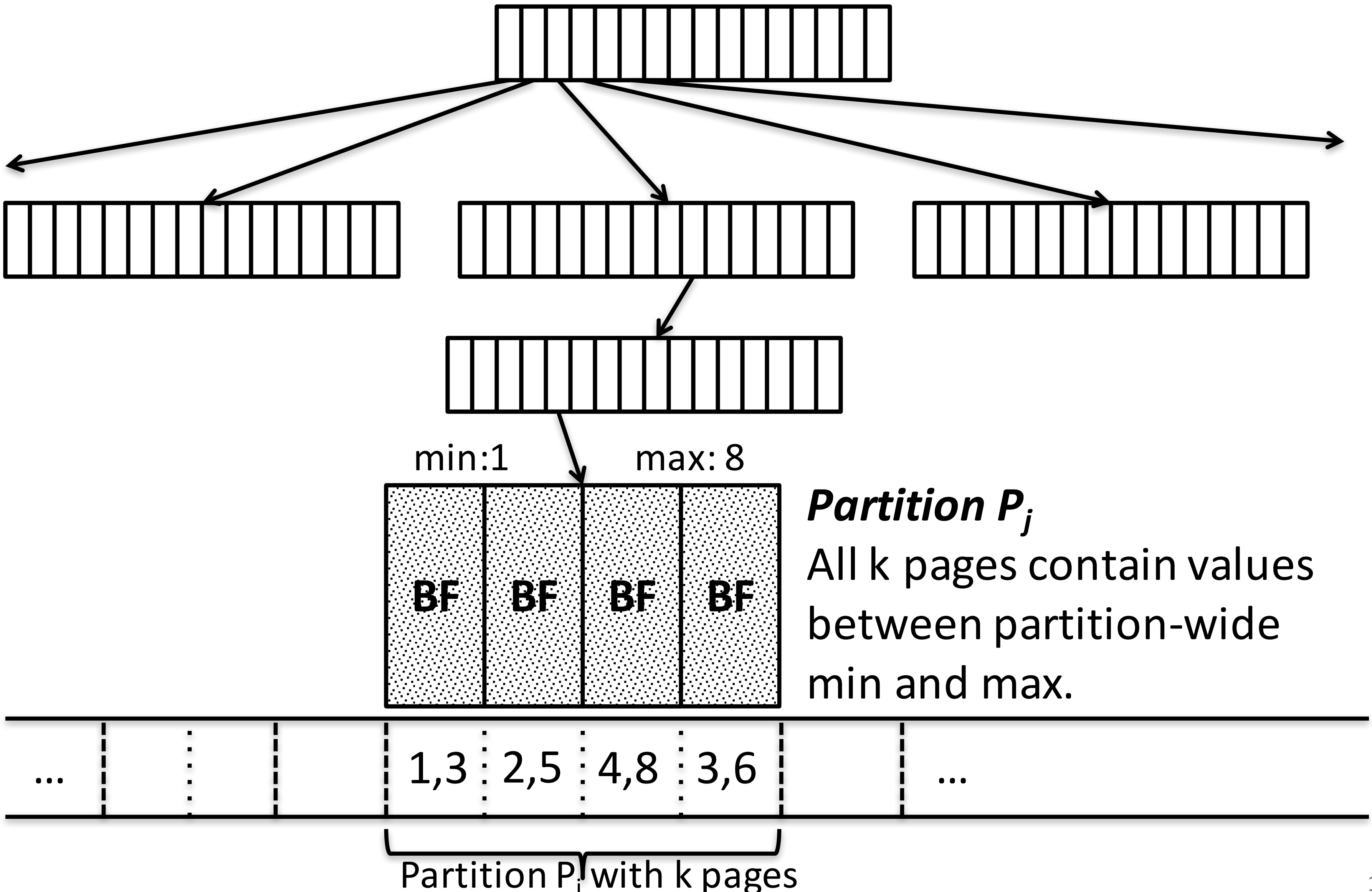
using the min/max as keys



Bloom filter Trees



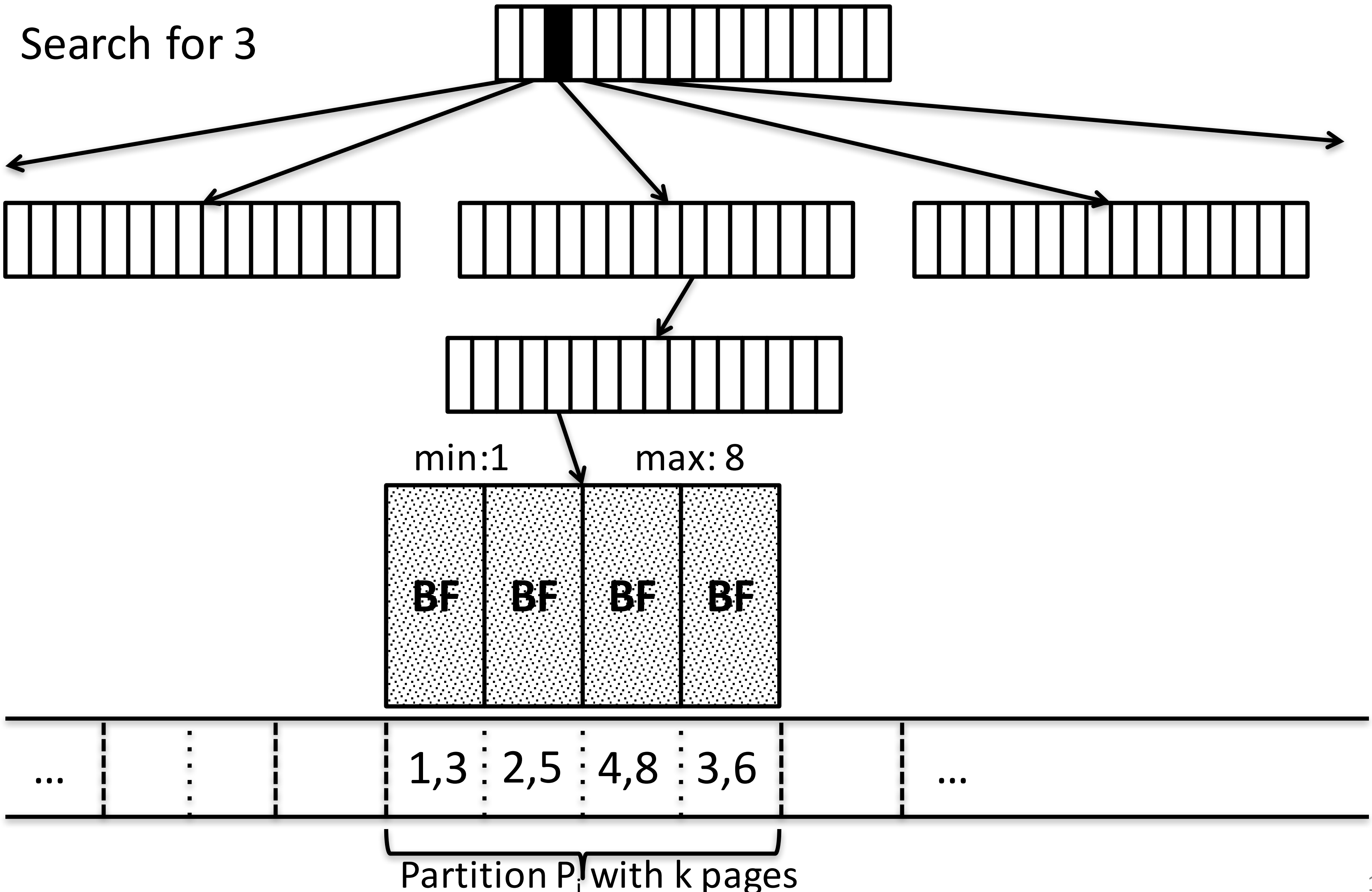
Bloom filter Trees



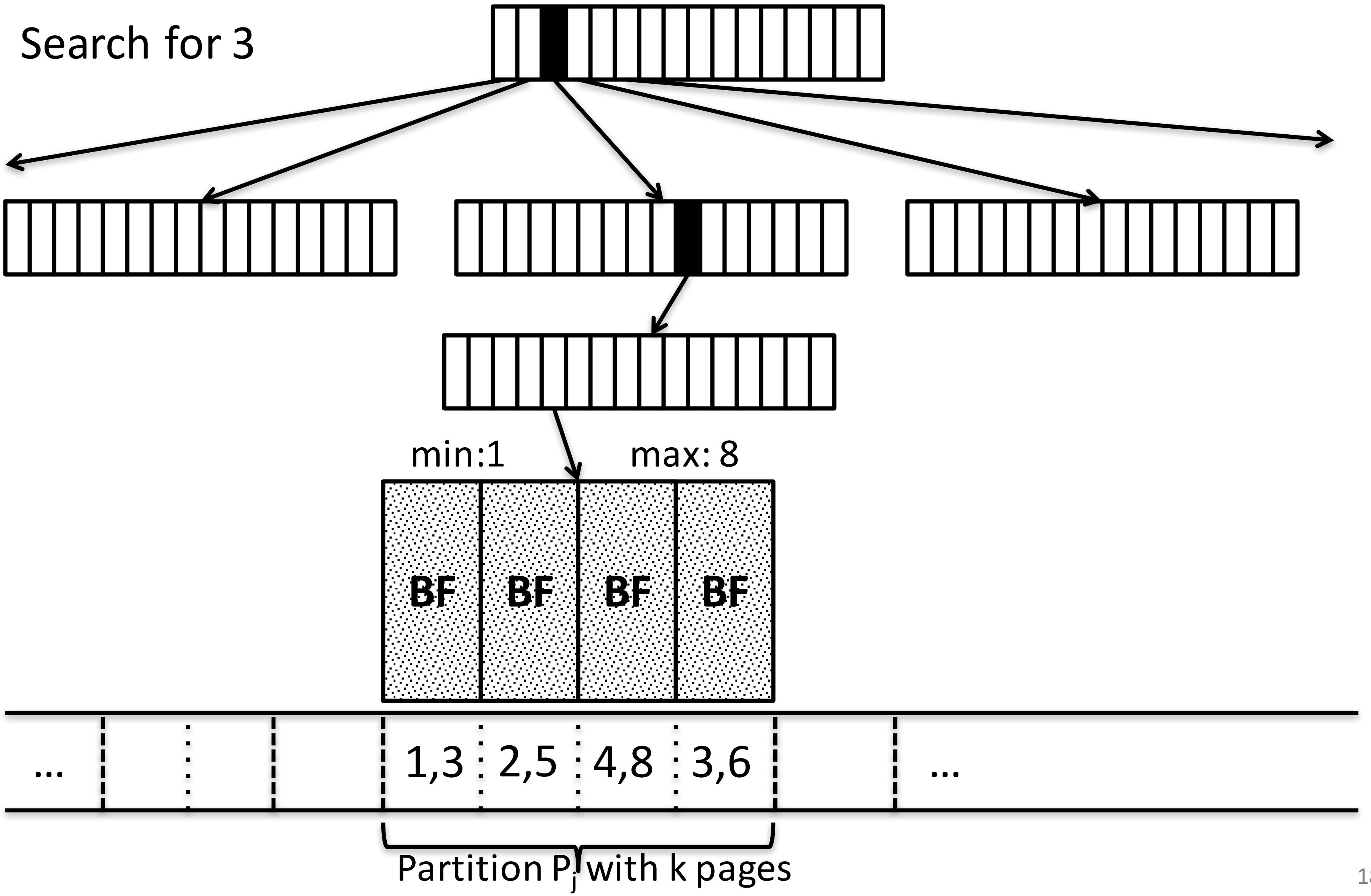
Partition P_j

All k pages contain values between partition-wide min and max.

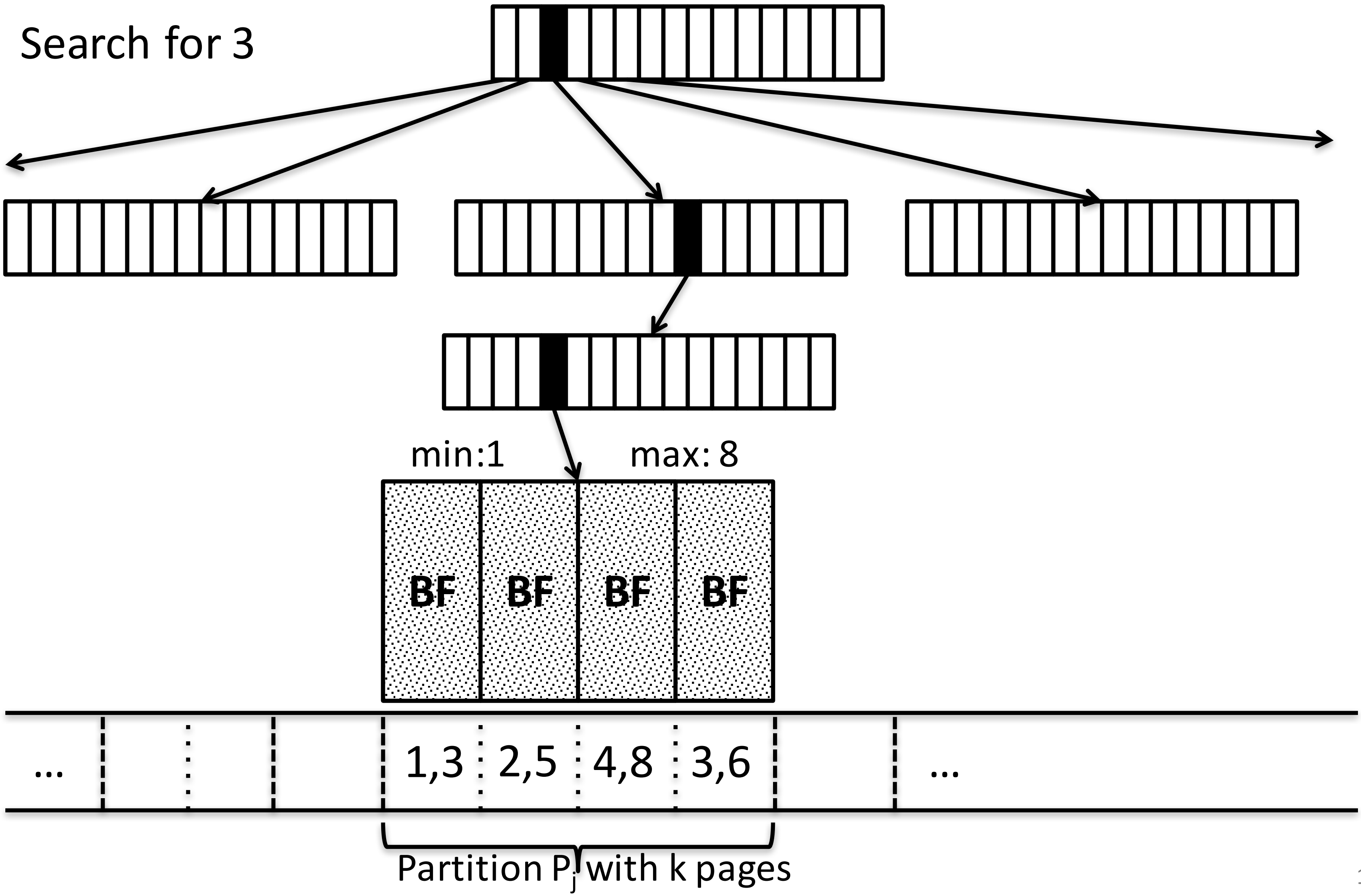
Bloom filter Trees



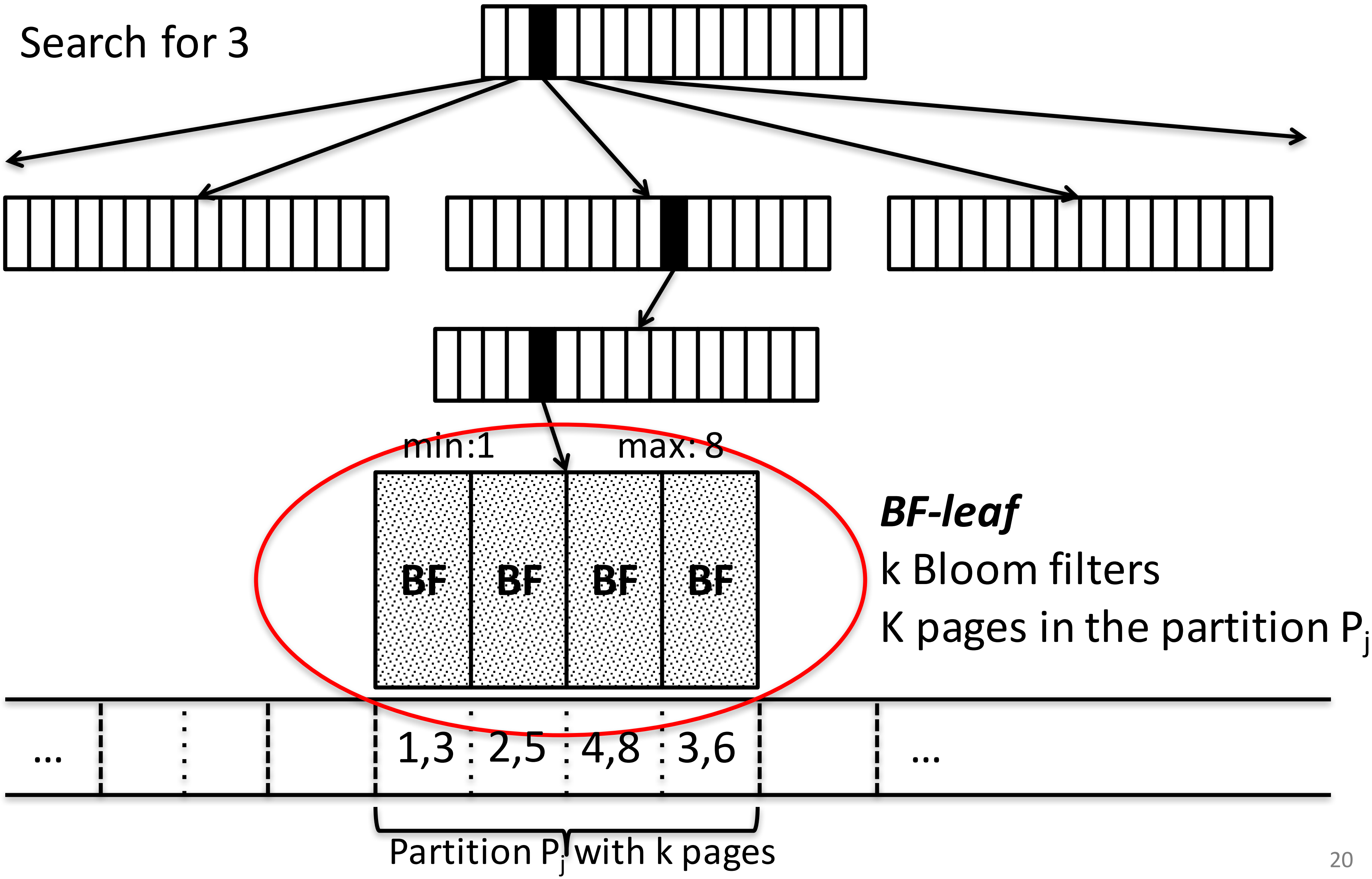
Bloom filter Trees



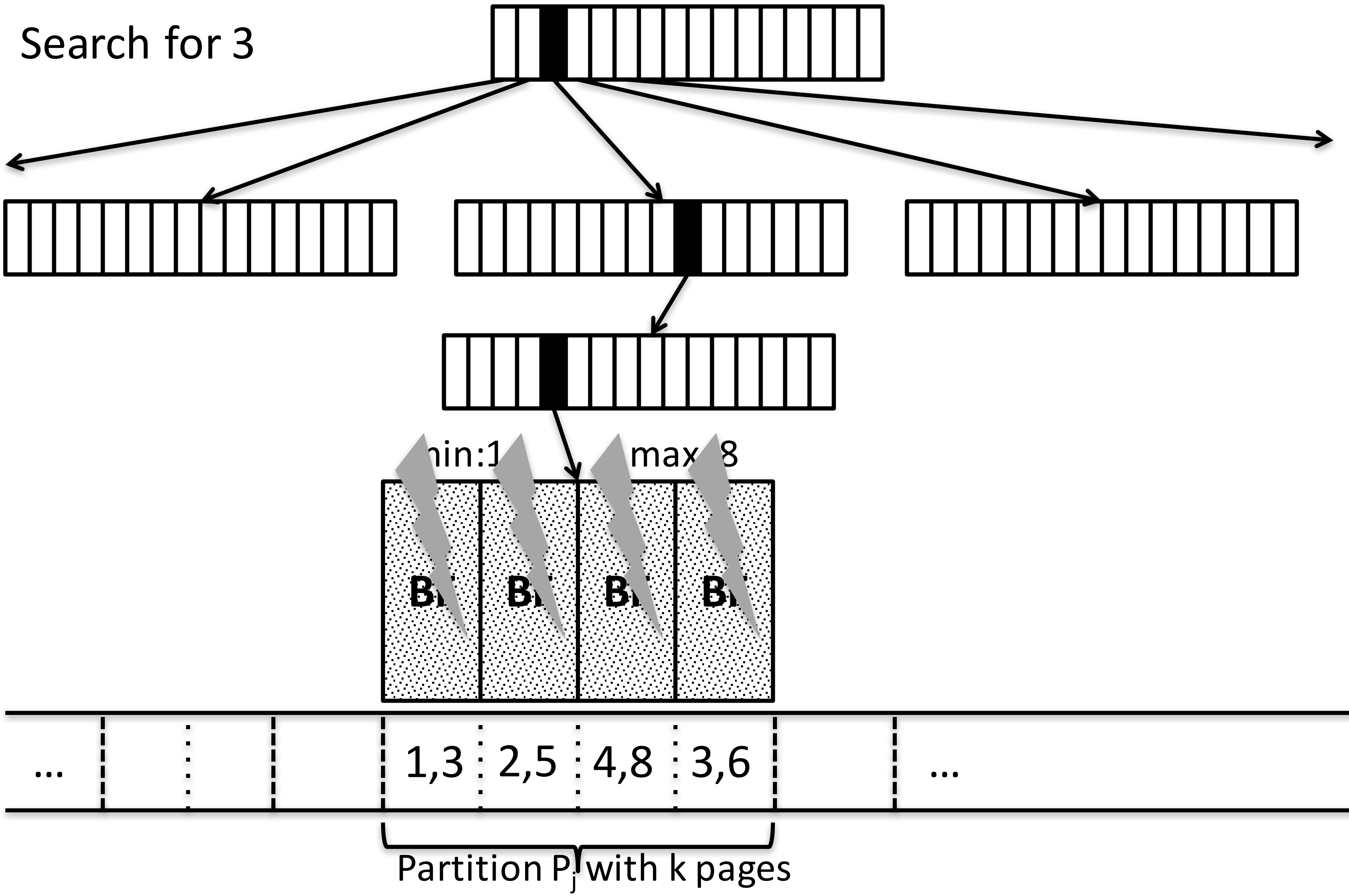
Bloom filter Trees



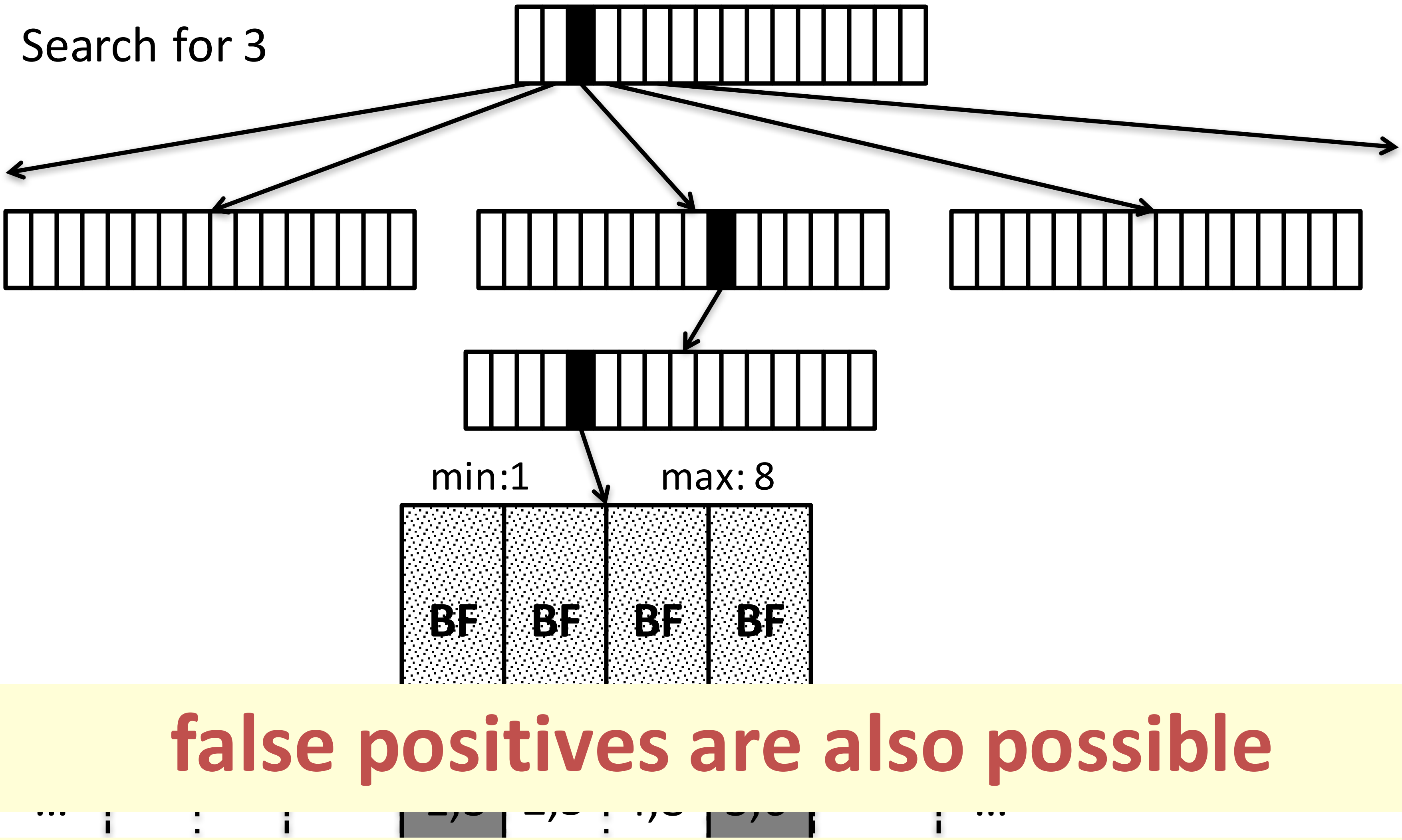
Bloom filter Trees



Bloom filter Trees



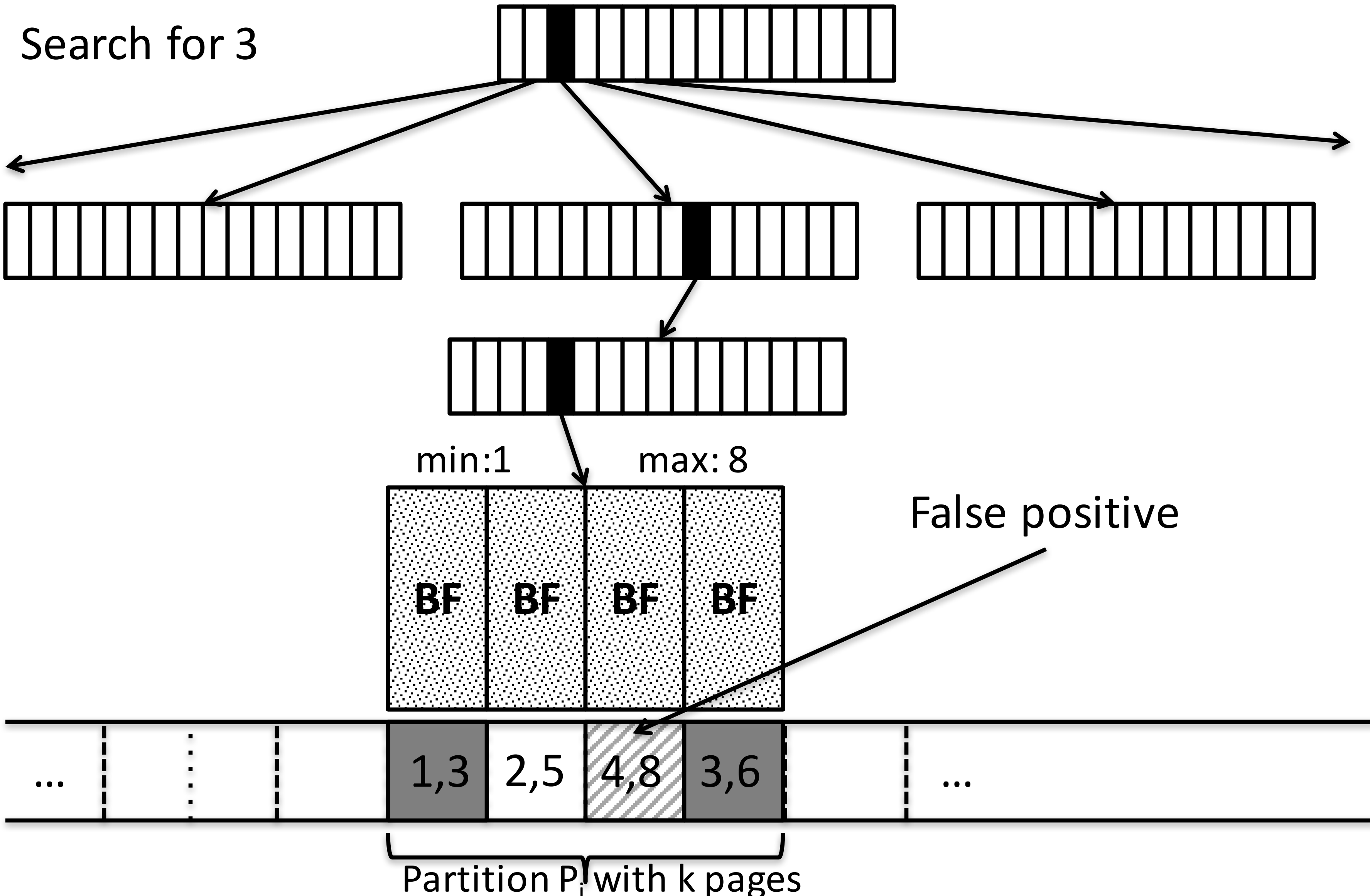
Bloom filter Trees



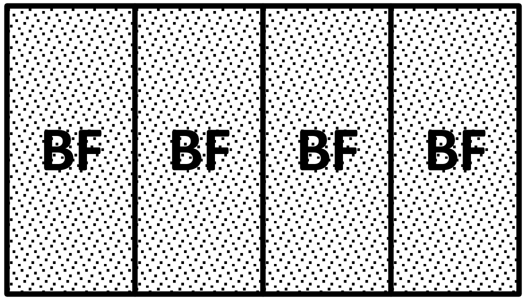
false positives are also possible

retrieve and search for desired value

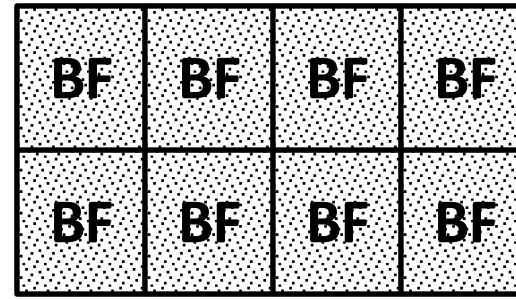
Bloom filter Trees



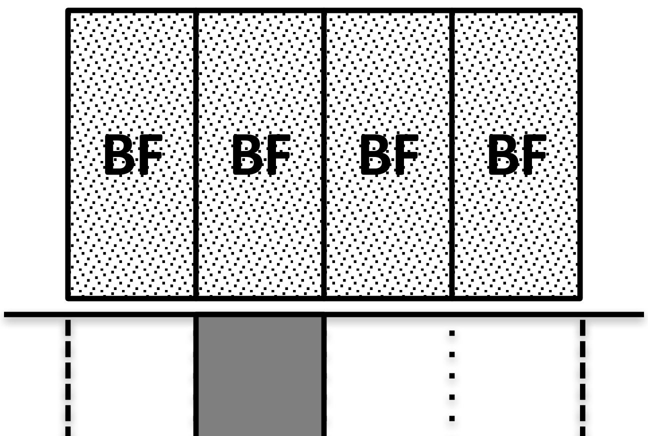
BF-Tree Design



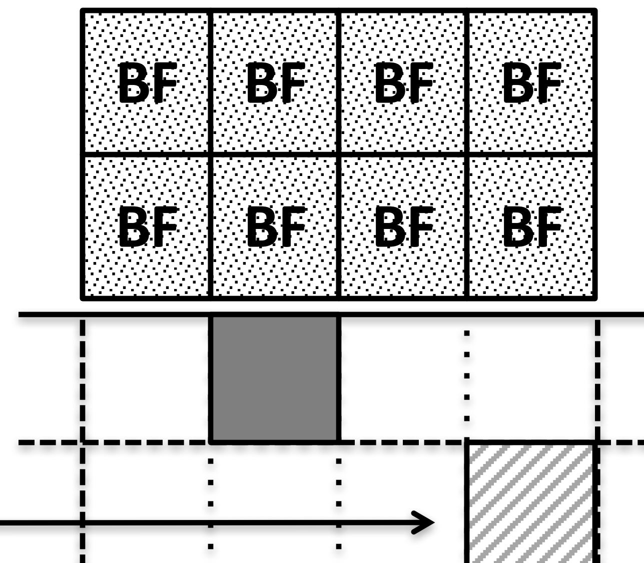
BFs have tunable size



Variable *false positive probability* (fpp)



$p_1 = 0.01\%$
If BF size is half
$p_2 = 1\%$



False positive

tunable size → variable performance

BF-Trees in action

Datasets

- *1GB synthetic* with 256b tuples and 8b keys
- *30GB TPCH (SF30)*
- *Smart Home Dataset (SHD)*

Workload

- Point queries (PK or TPCH date or energy level)

5 storage configurations (index/data)

- mem/SSD
- mem/HDD
- SSD/SSD
- SSD/HDD
- HDD/HDD

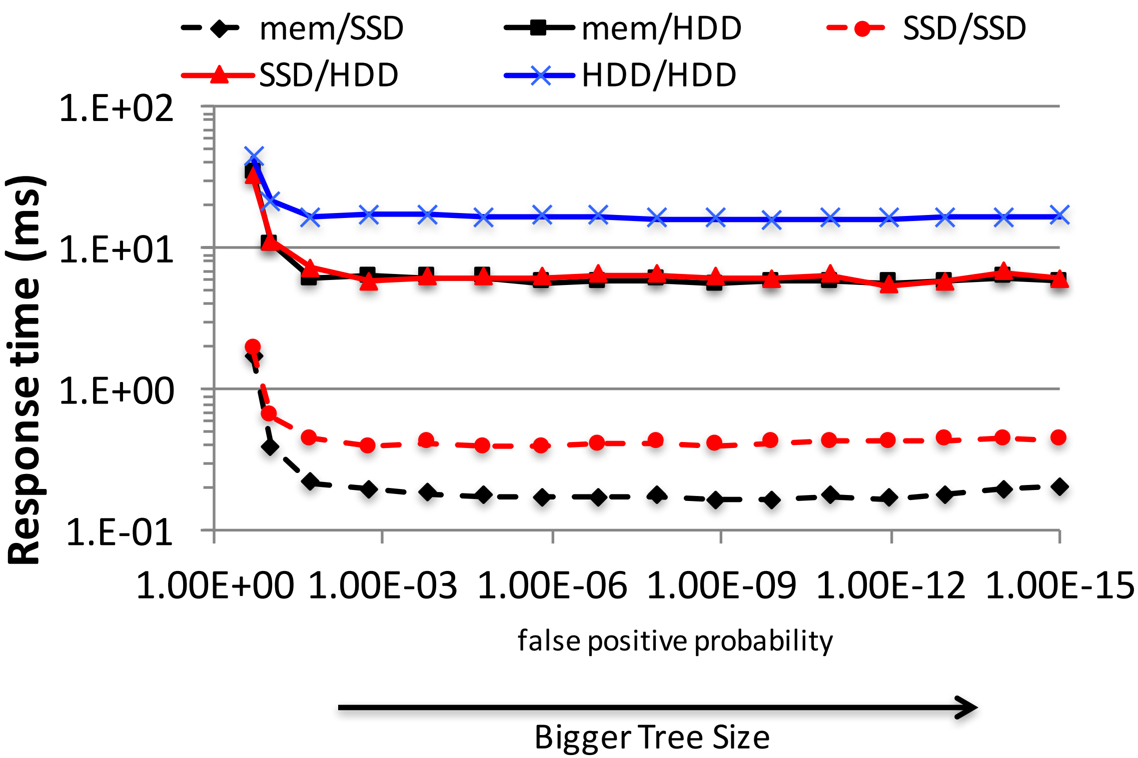
BF-Trees for PK

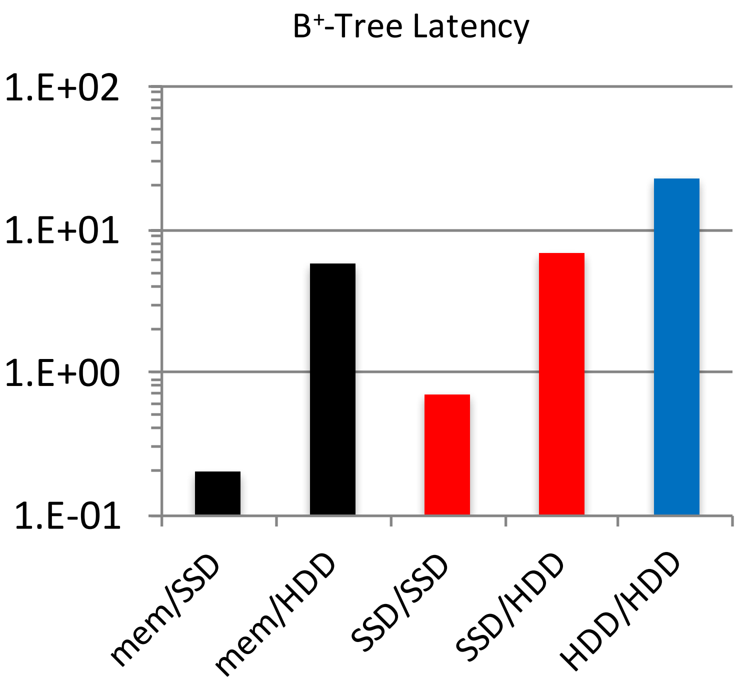
average index probe time for 1GB relation

varying

false positive probability; storage configuration

Tuplesize: 256 bytes
Keysize: 8 bytes





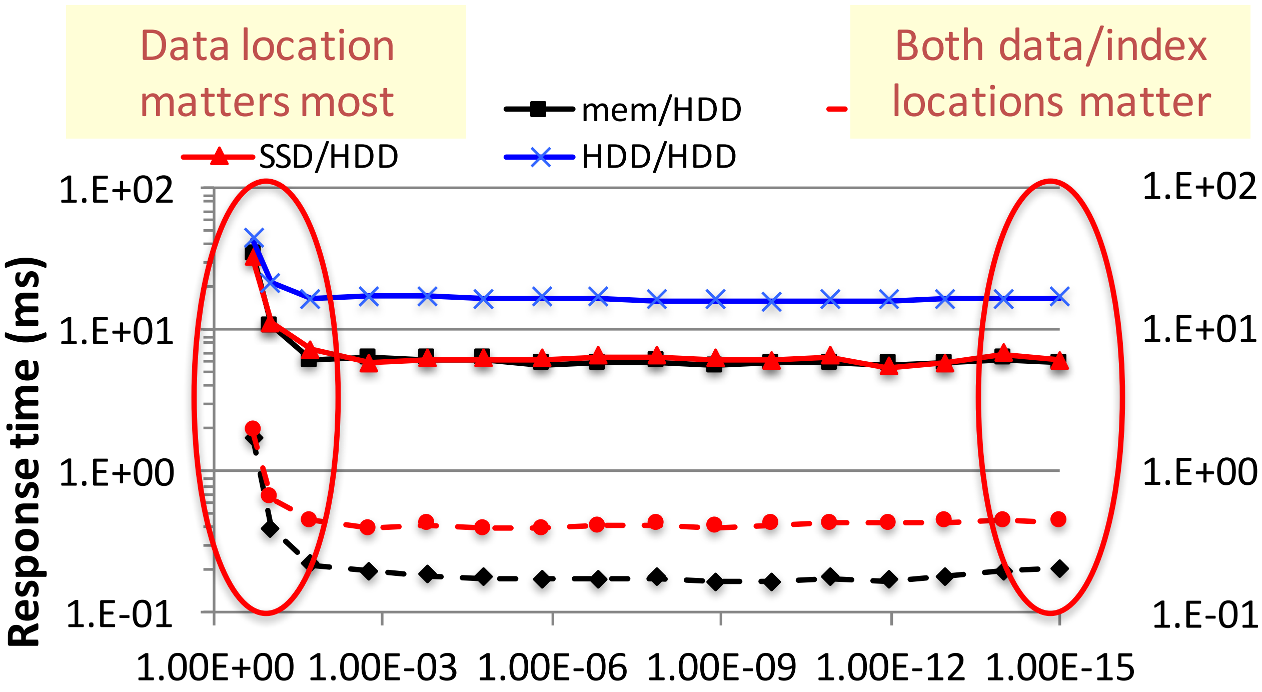
BF-Trees for PK

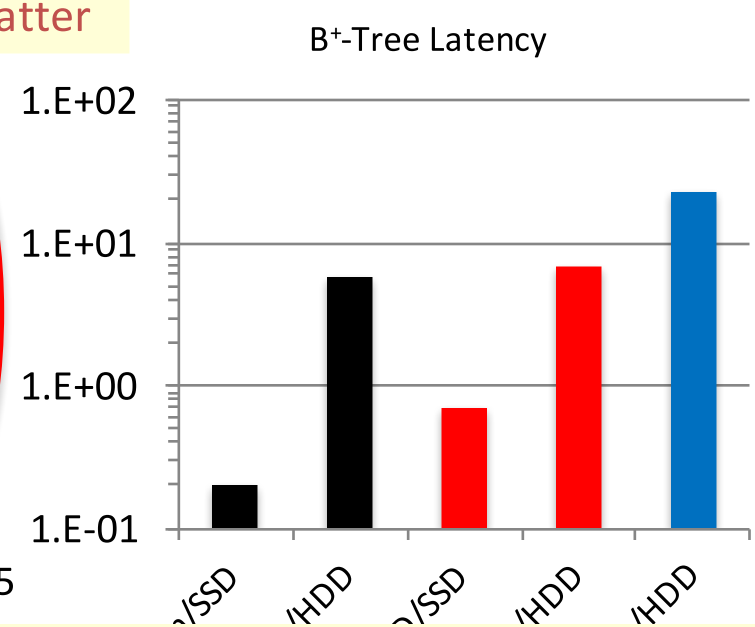
average index probe time for 1GB relation

varying

false positive probability; storage configuration

Tuplesize: 256 bytes
Keysize: 8 bytes





what about the tree size?

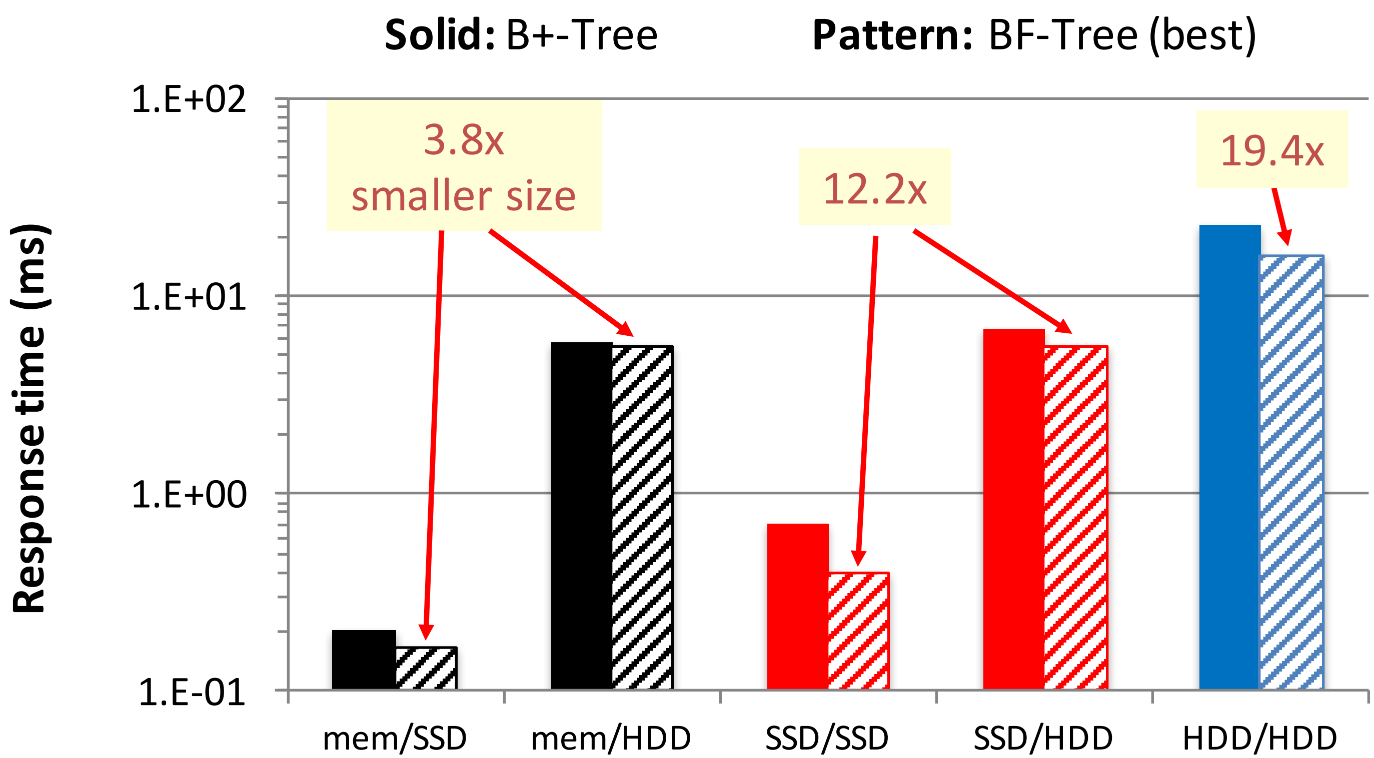
BF-Tree vs B^+-Tree: Size & Latency

average index probe time for 1GB relation

varying

- false positive probability; storage configuration

Tuplesize: 256 bytes
Keysize: 8 bytes



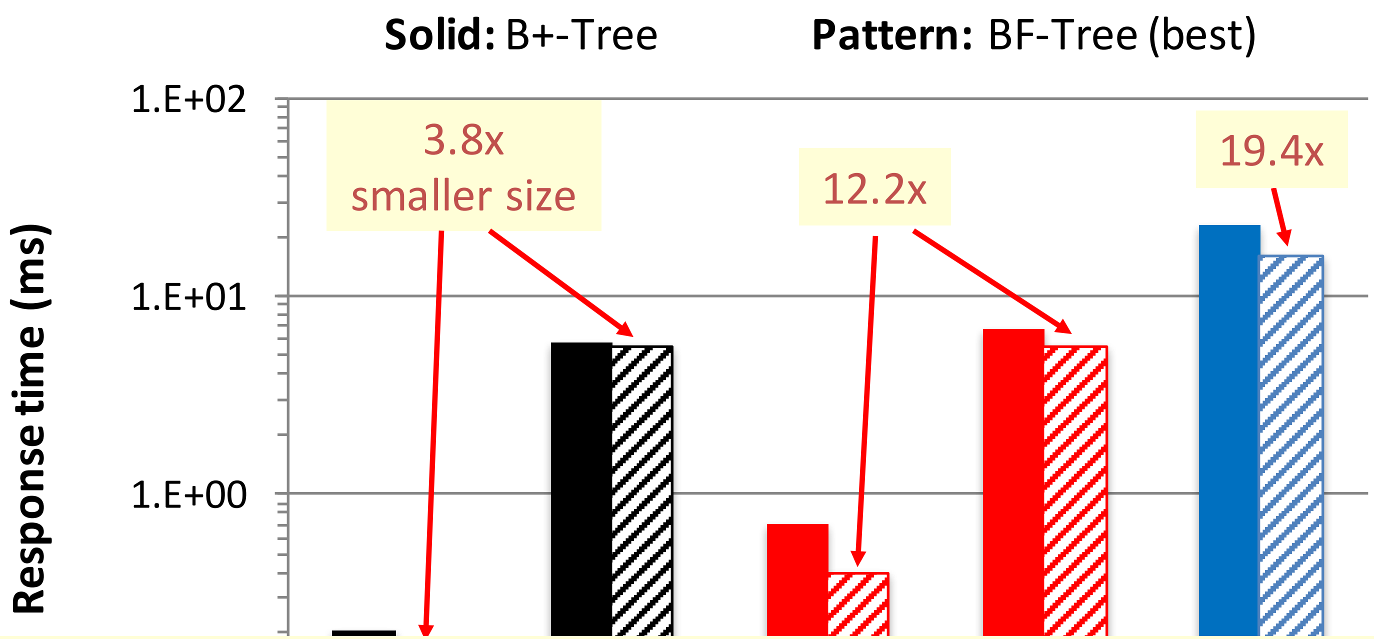
BF-Tree vs B^+-Tree: Size & Latency

average index probe time for 1GB relation

varying

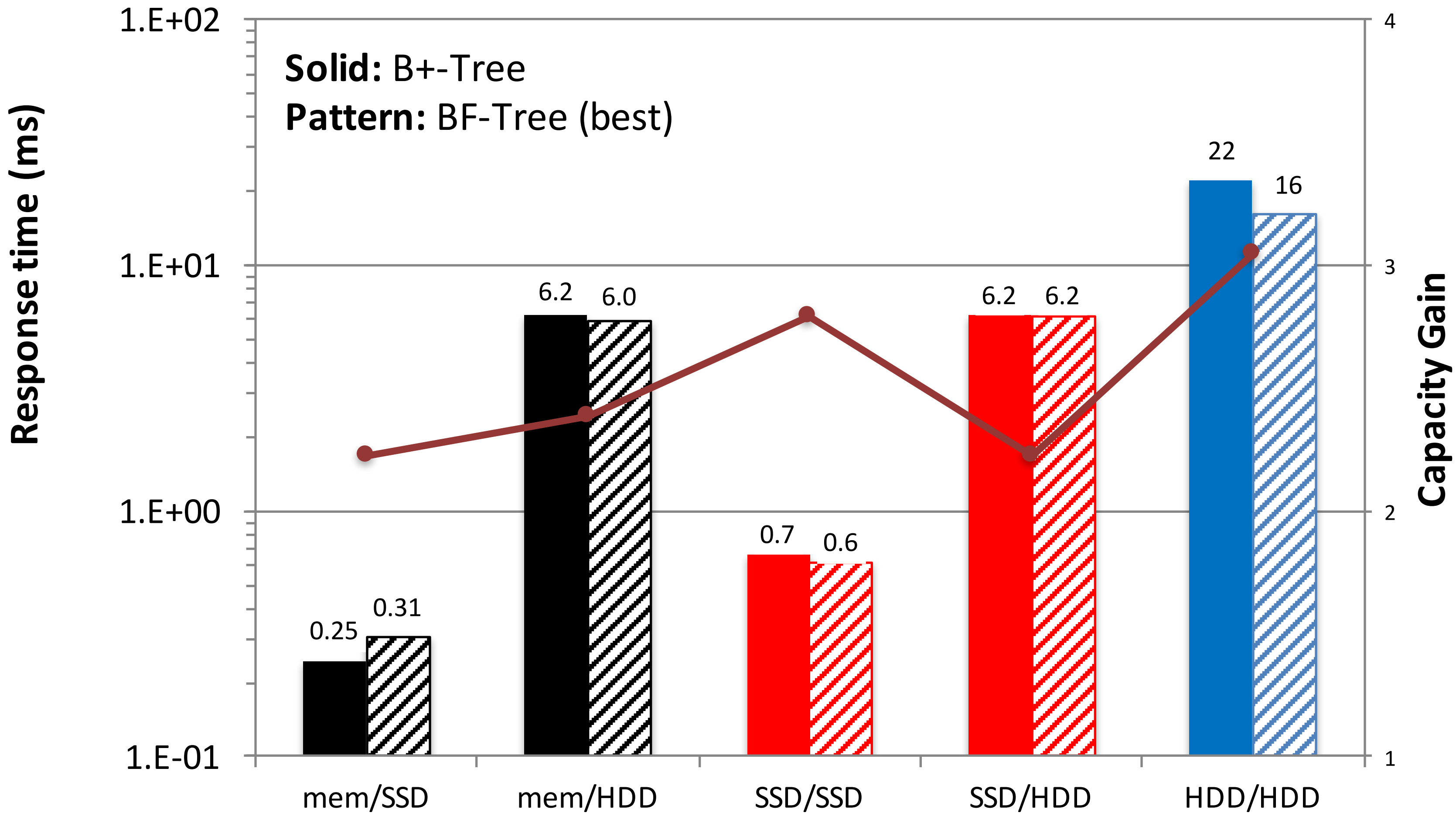
false positive probability; storage configuration

Tuplesize: 256 bytes
Keysize: 8 bytes

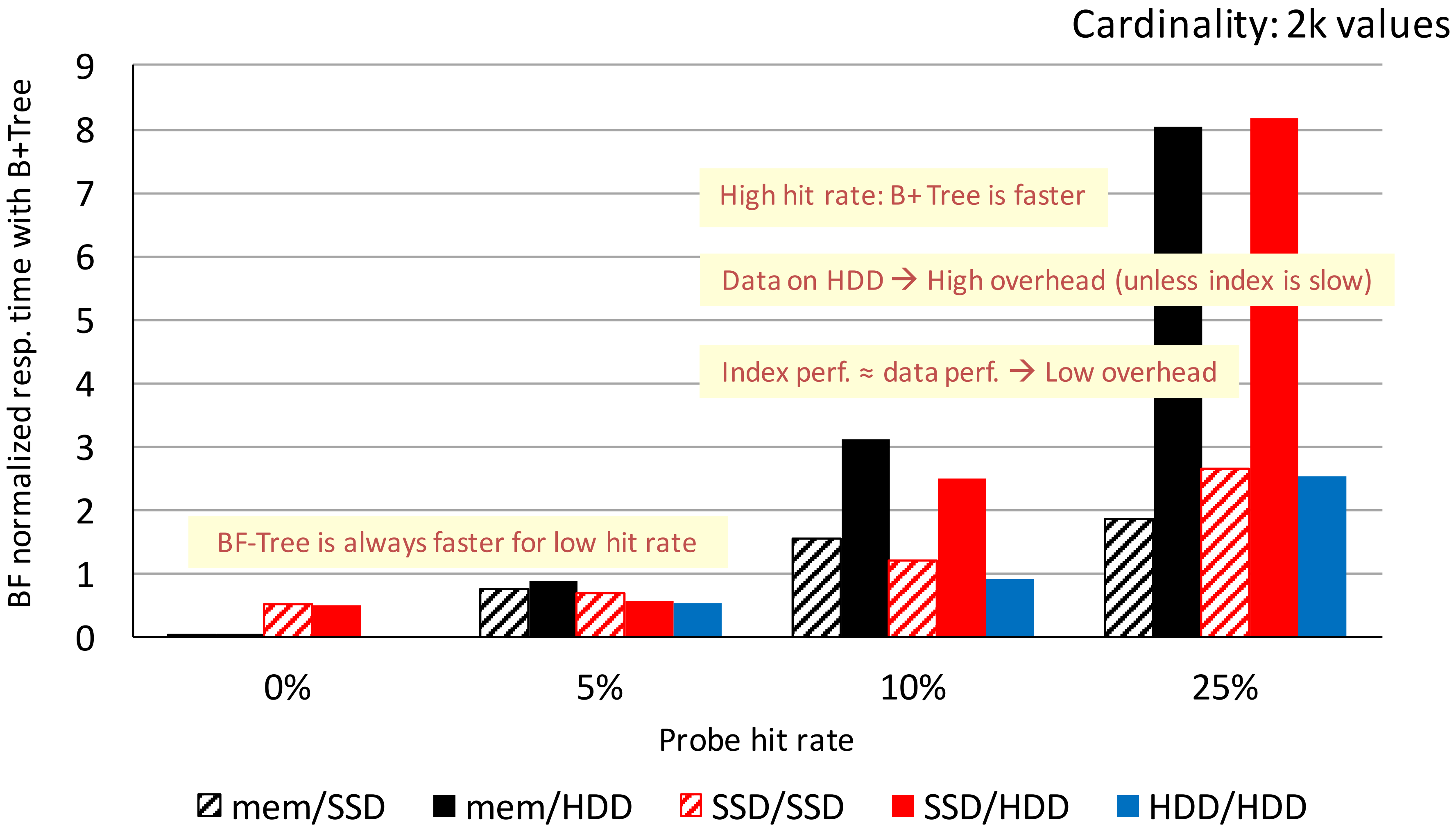


competitive performance with space savings

BF-Tree for SHD



TPCH point queries on date



Approximate Tree Indexing

tunable size → variable performance

competitive resp. time w/ 4-20x capacity savings

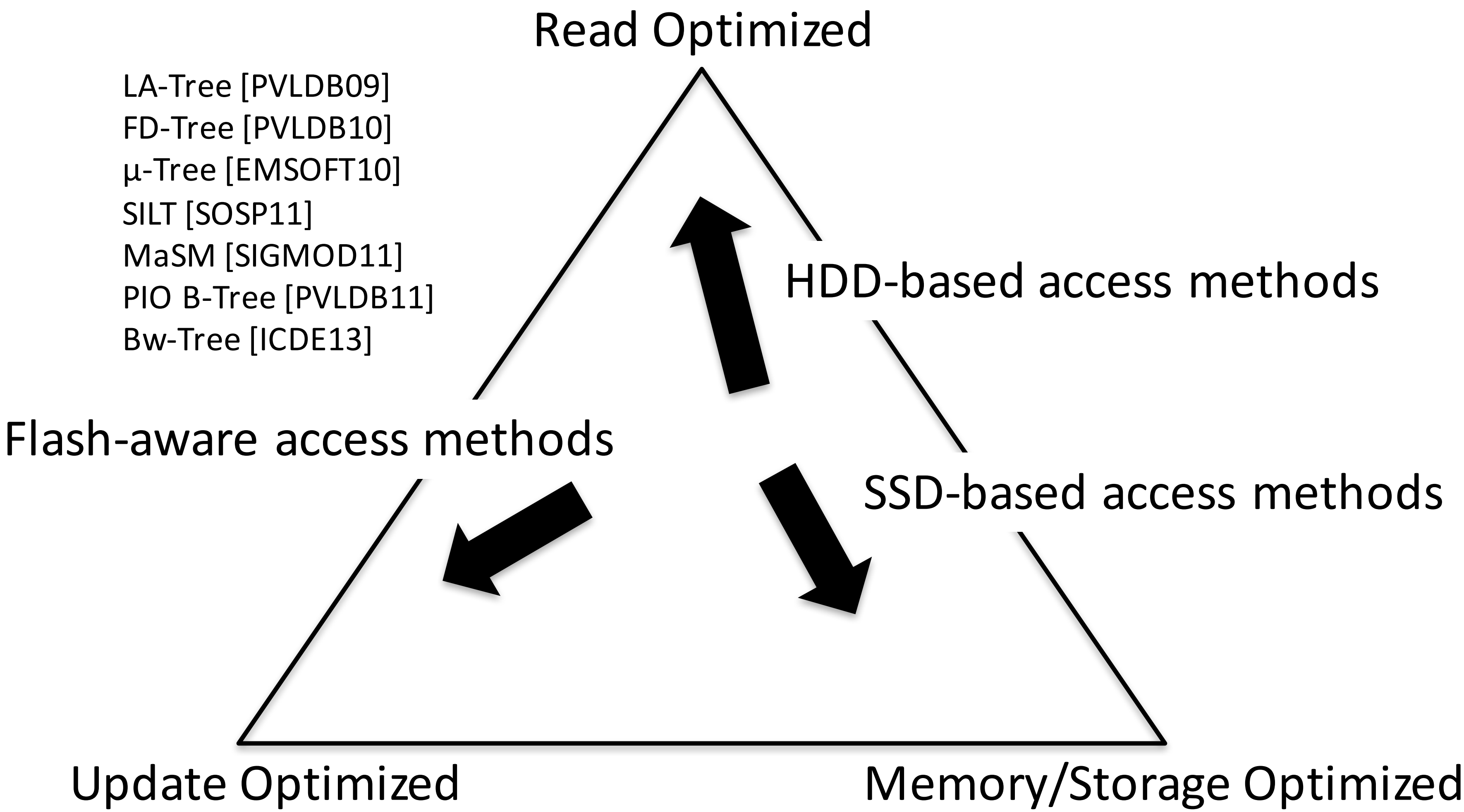
tailored for:

- datasets with *implicit clustering*
- workloads with low *hit rate*

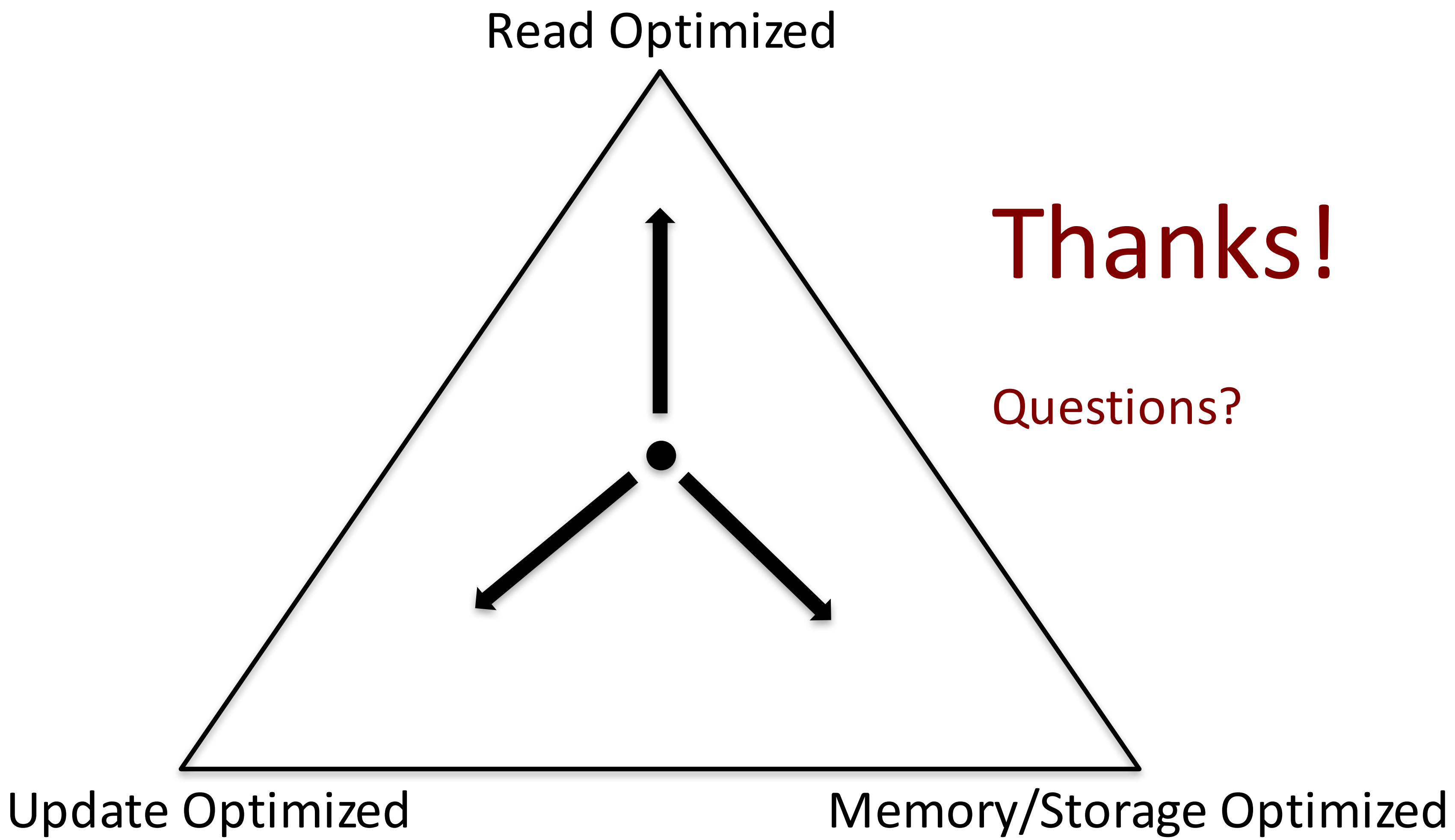
more details in paper:

- analytical modeling, range scans
- updates, datasets, more comparisons

RUM Tunable Indexing



RUM Tunable Indexing



Thanks!

Questions?

http://daslab.seas.harvard.edu/rum-conjecture/