

CS 265

Stratos Idreos

BIG DATA SYSTEMS

NoSQL | Neural Networks | Image AI | LLMs | Data Science

Today + one more lecture on self-designing systems (Tuesday)

Then: 1 lecture on fast NN training through data movement optimization
(systems project)

One lecture on LLM+RAG

Then 3 lectures on end-to-end AI through reimagining storage

Then spring break

Then papers presentations

Systems projects should start coding by next week
Research projects should know projects in next two weeks

**sorted keys
columnar layout**

**sorted keys
columnar layout**

RULES



**sorted
search**

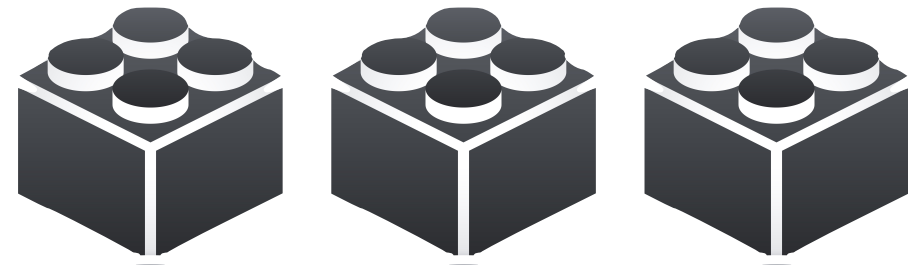
DEPENDS ON **HARDWARE ENGINEERING**

sorted keys
columnar layout

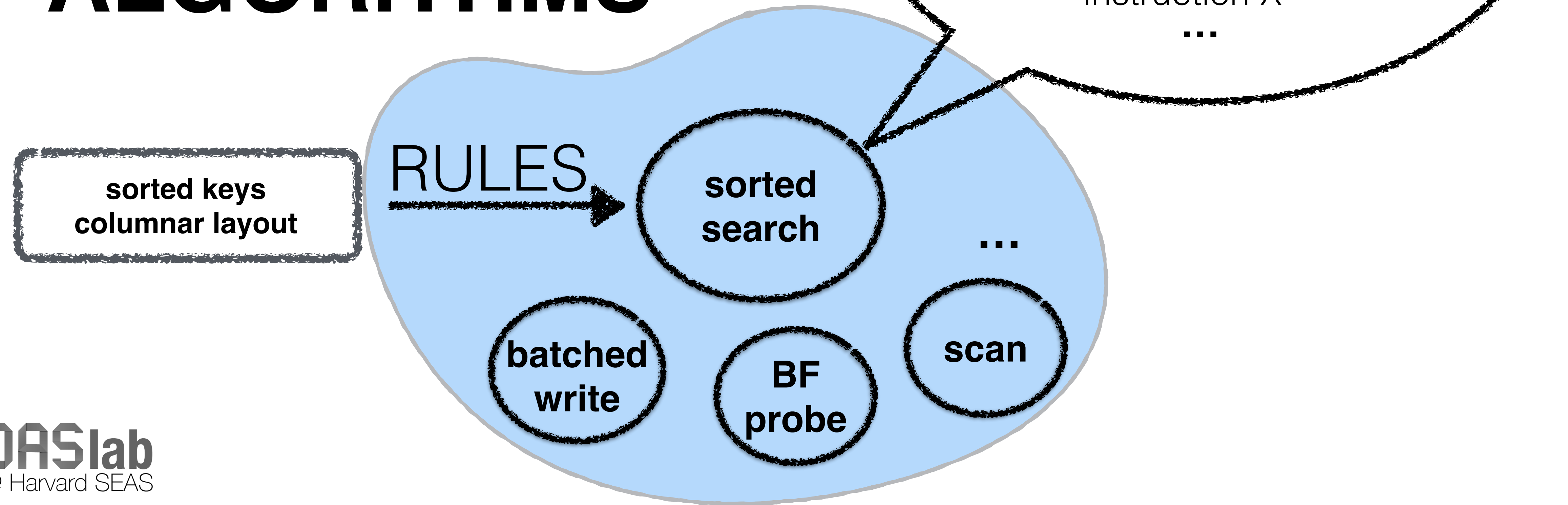
RULES →

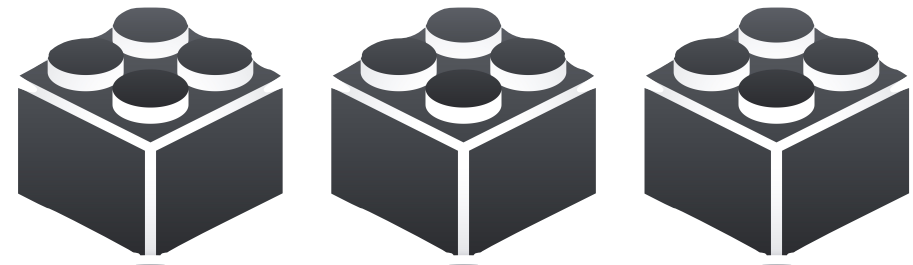
sorted
search

binary search1
binary search2
interpolation search1
interpolation search2
using new SIMD
instruction X
...



COMPONENTS OF KEY-VALUE ALGORITHMS





COMPONENTS OF KEY-VALUE ALGORITHMS

sorted keys
columnar layout

RULES →

sorted
search

batched
write

BF
probe

scan

...

binary search1
binary search2

code,
model

interpolation search1
interpolation search2

code,
model

using new SIMD
instruction X

code,
model

...

LEARNING

SYNTHESIS FROM LEARNED MODELS

coding, modeling, generalized models, and a touch of ML

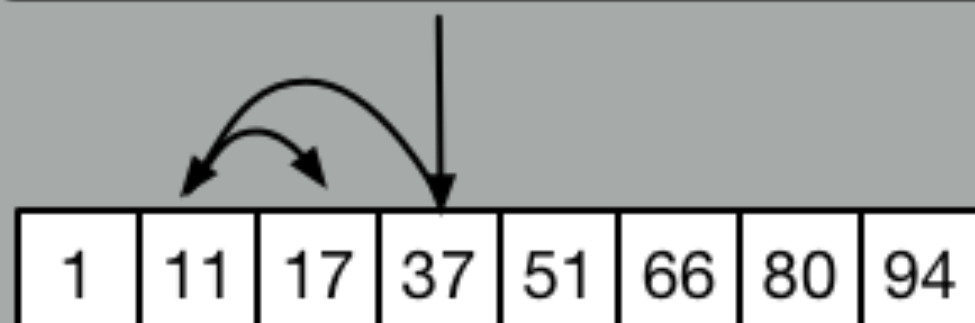


1. MINIMAL CODE

e.g., binary search

```
if (data[middle] < search_val) {  
    low = middle + 1;  
} else {  
    high = middle;  
}  
middle = (low + high)/2;
```

C++



SYNTHESIS FROM LEARNED MODELS

coding, modeling, generalized models, and a touch of ML



1. MINIMAL CODE

e.g., binary search

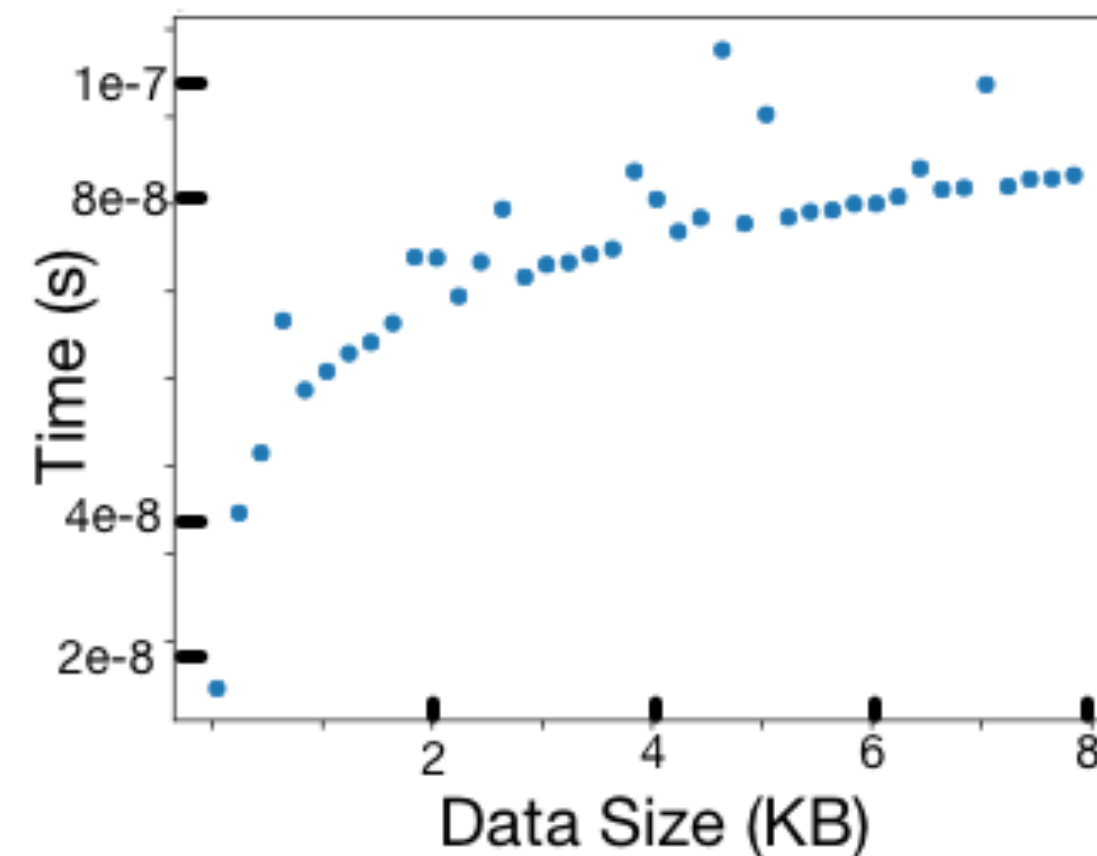
```
if (data[middle] < search_val) {  
    low = middle + 1;  
} else {  
    high = middle;  
}  
middle = (low + high)/2;
```



Run

1	11	17	37	51	66	80	94
---	----	----	----	----	----	----	----

2. BENCHMARK



SYNTHESIS FROM LEARNED MODELS

coding, modeling, generalized models, and a touch of ML



1. MINIMAL CODE

e.g., binary search

```
if (data[middle] < search_val) {  
    low = middle + 1;  
} else {  
    high = middle;  
}  
middle = (low + high)/2;
```

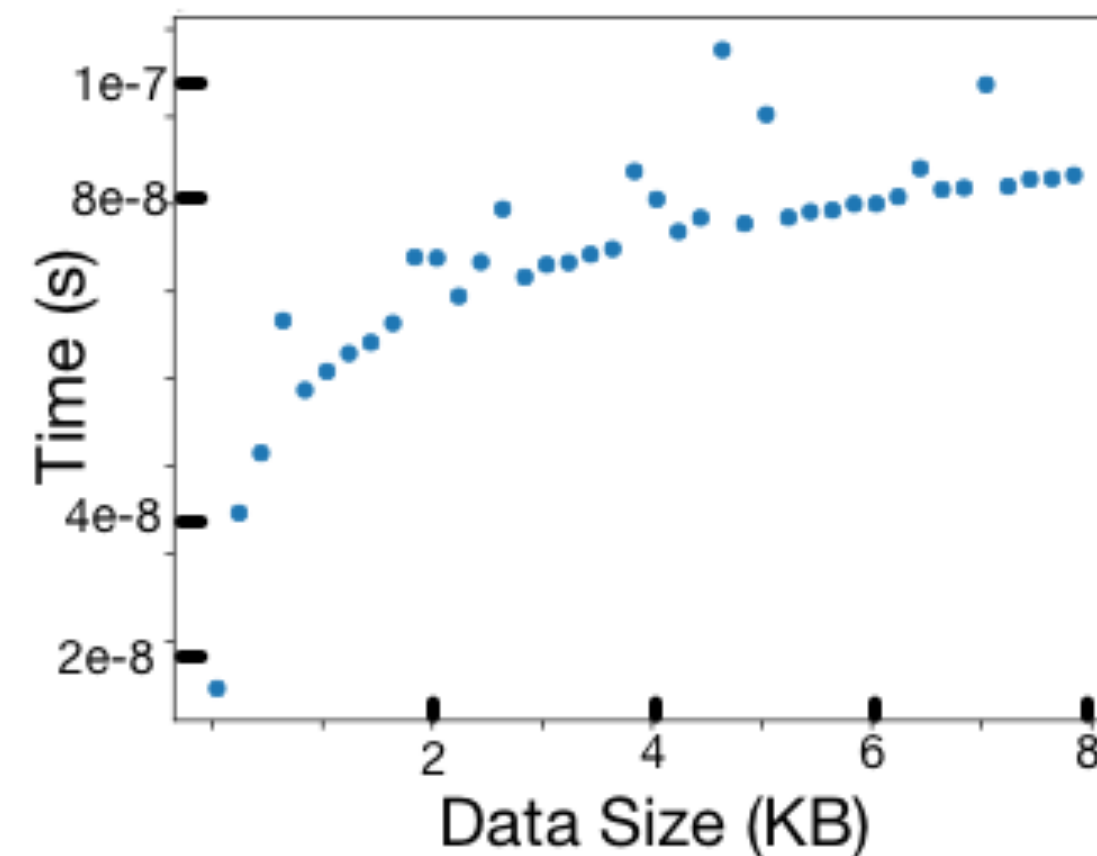
C++

1	11	17	37	51	66	80	94
---	----	----	----	----	----	----	----



Run

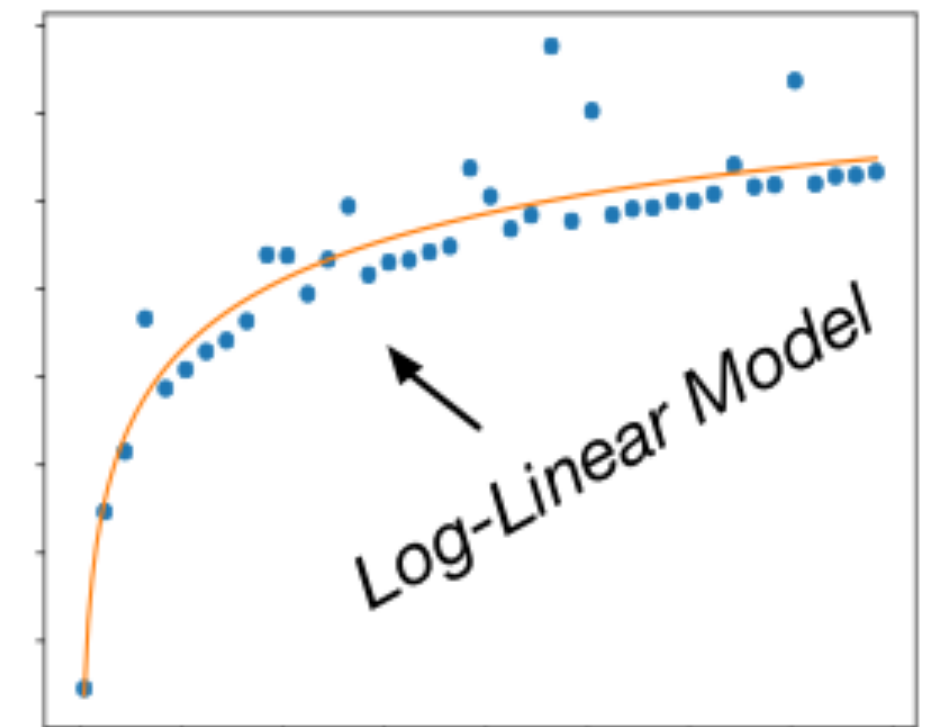
2. BENCHMARK



$f(x)$

Train

3. FIT MODEL



$$f(x) = ax + b \log x + c$$

SYNTHESIS FROM LEARNED MODELS

coding, modeling, generalized models, and a touch of ML



1. MINIMAL CODE

e.g., binary search

```
if (data[middle] < search_val) {  
    low = middle + 1;  
} else {  
    high = middle;  
}  
middle = (low + high)/2;
```

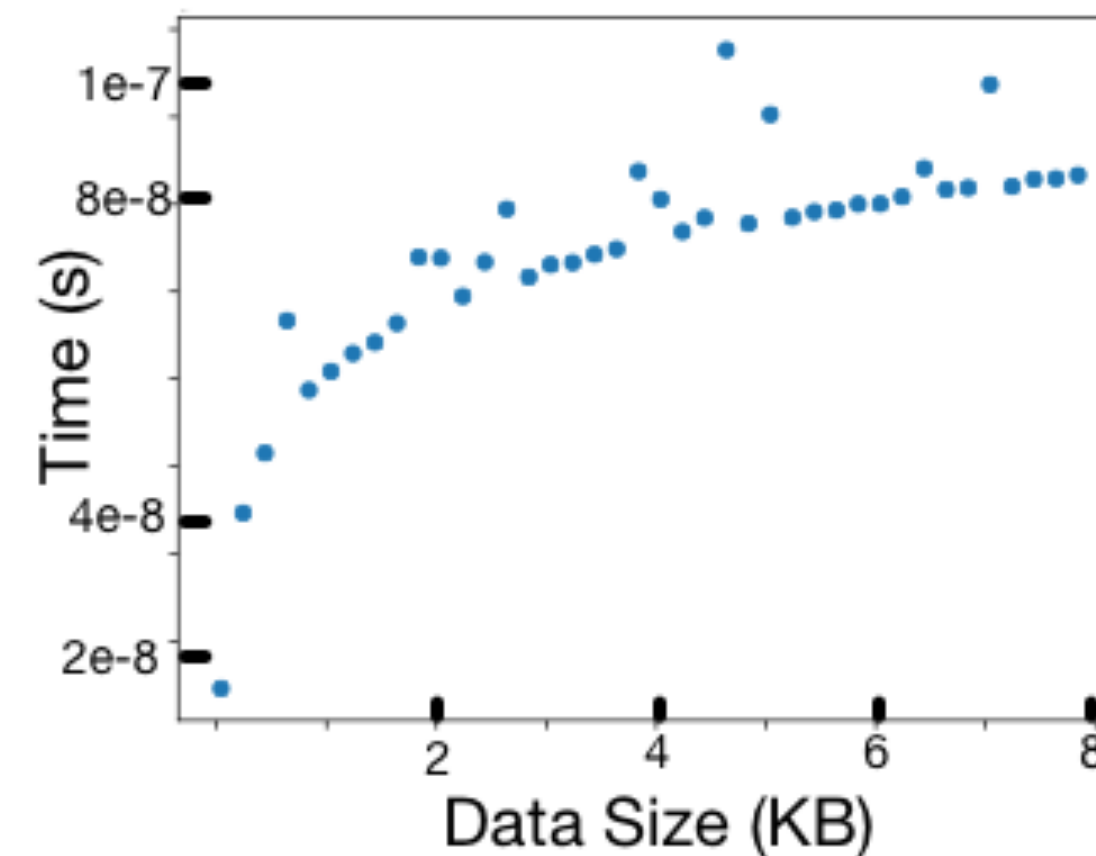
C++

1	11	17	37	51	66	80	94
---	----	----	----	----	----	----	----



Run

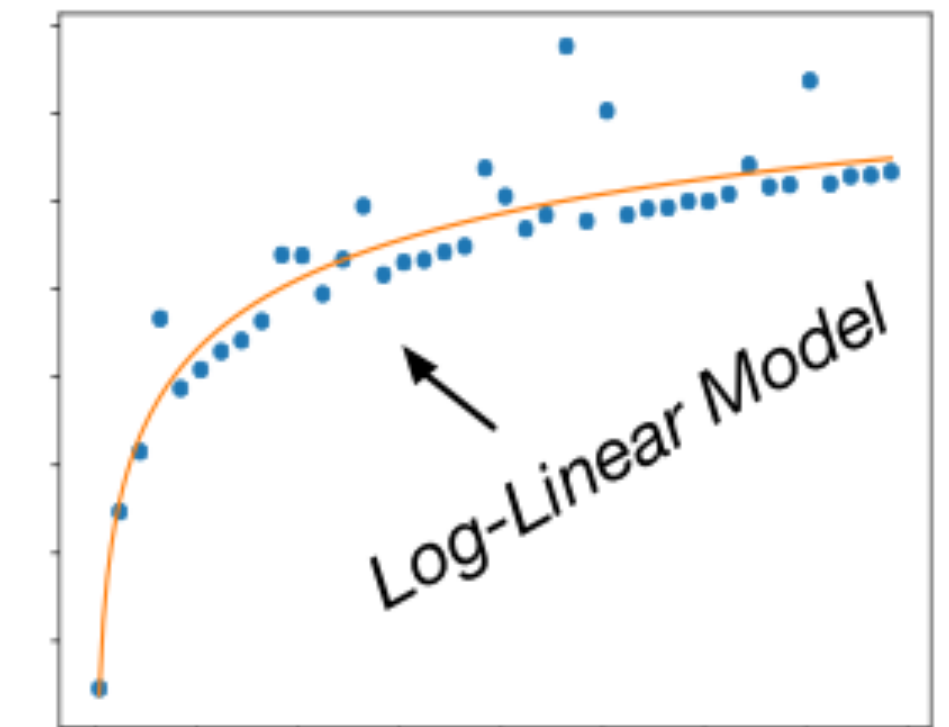
2. BENCHMARK



$f(x)$

Train

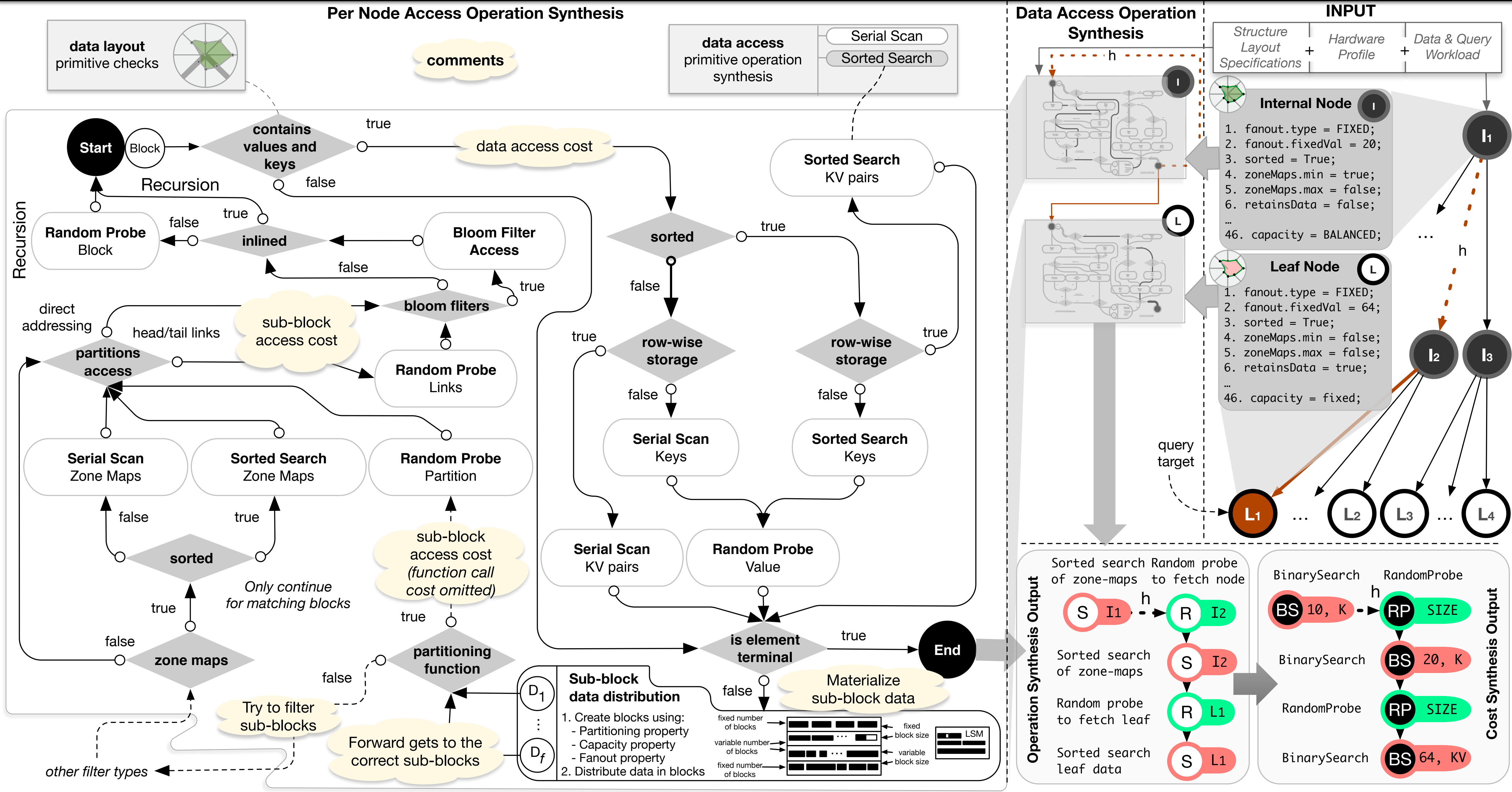
3. FIT MODEL



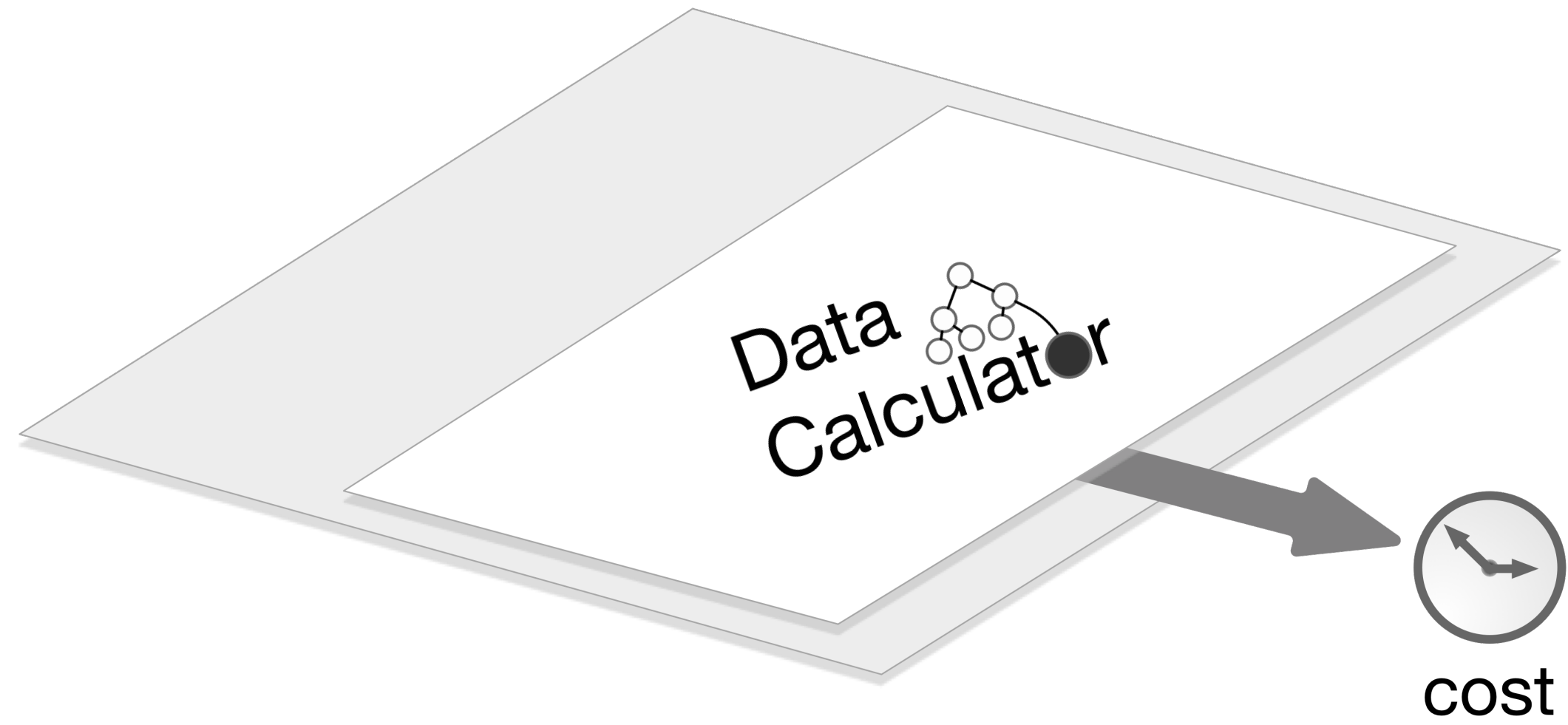
$$f(x) = ax + b \log x + c$$

FOLDING ALGORITHMIC, ENGINEERING, AND H/W, PROPERTIES INTO THE COEFFICIENTS

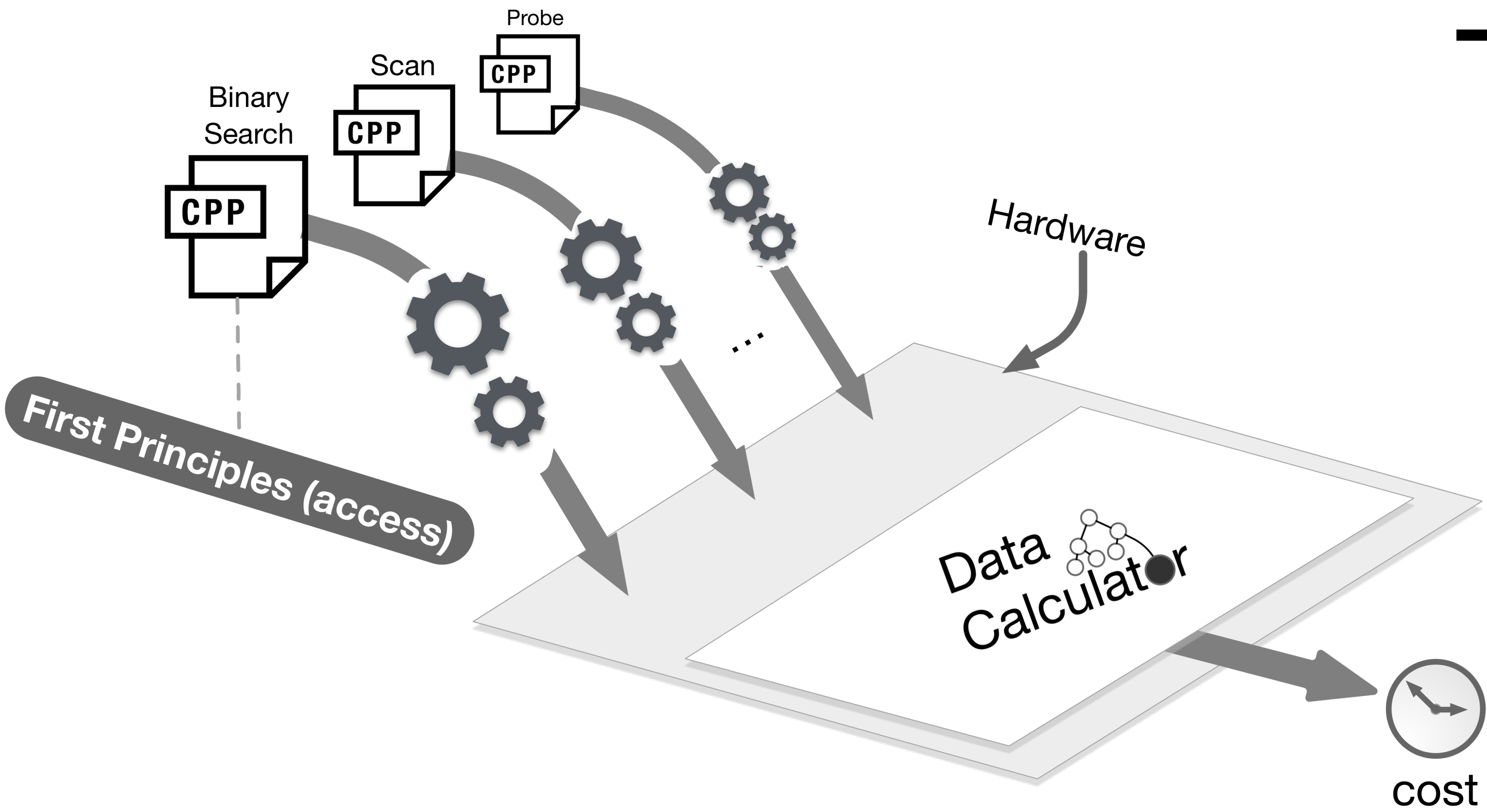
RULE/MODEL BASED SYSTEM SYNTHESIZES ALGORITHM AND COST



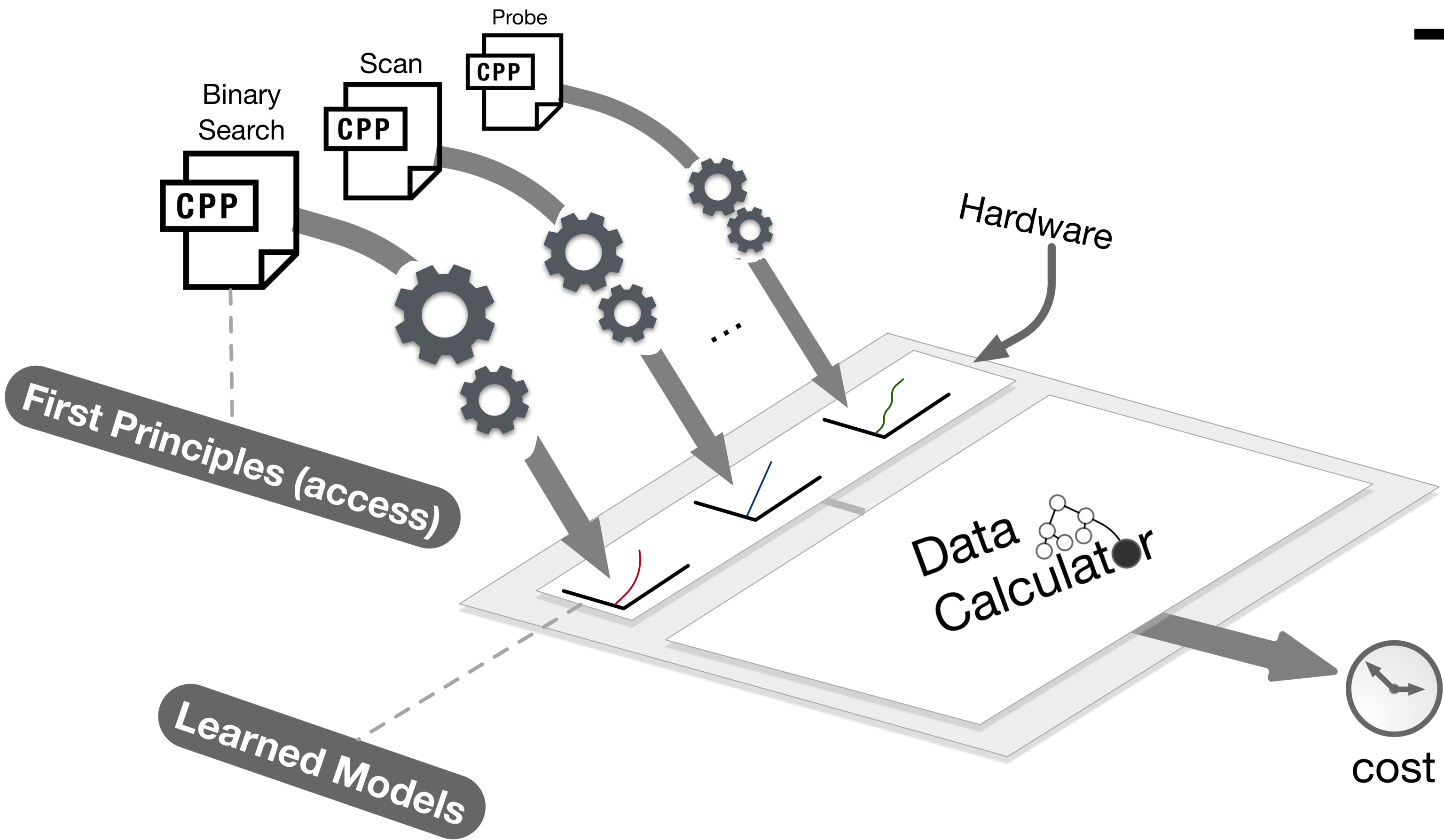
TRAINING



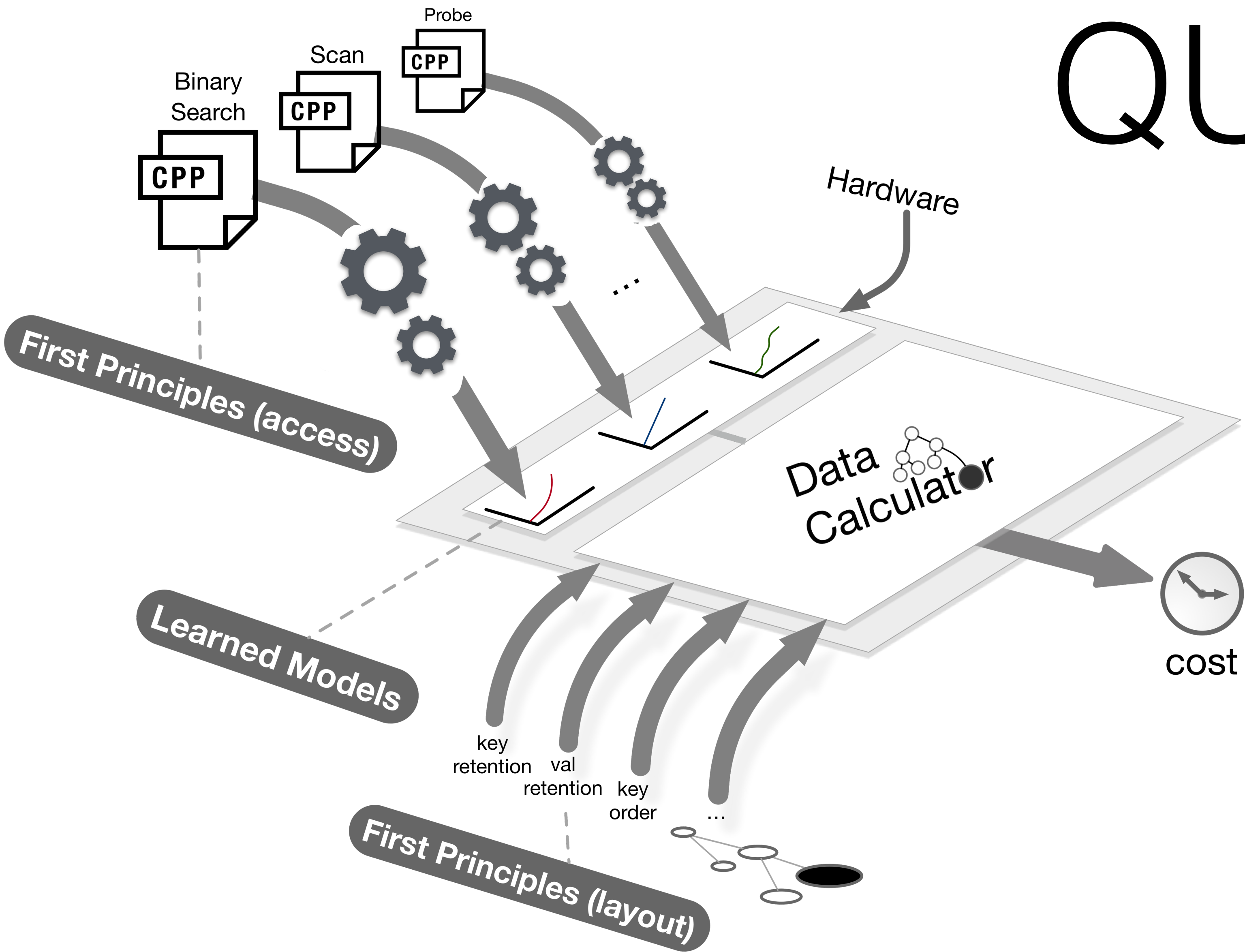
TRAINING



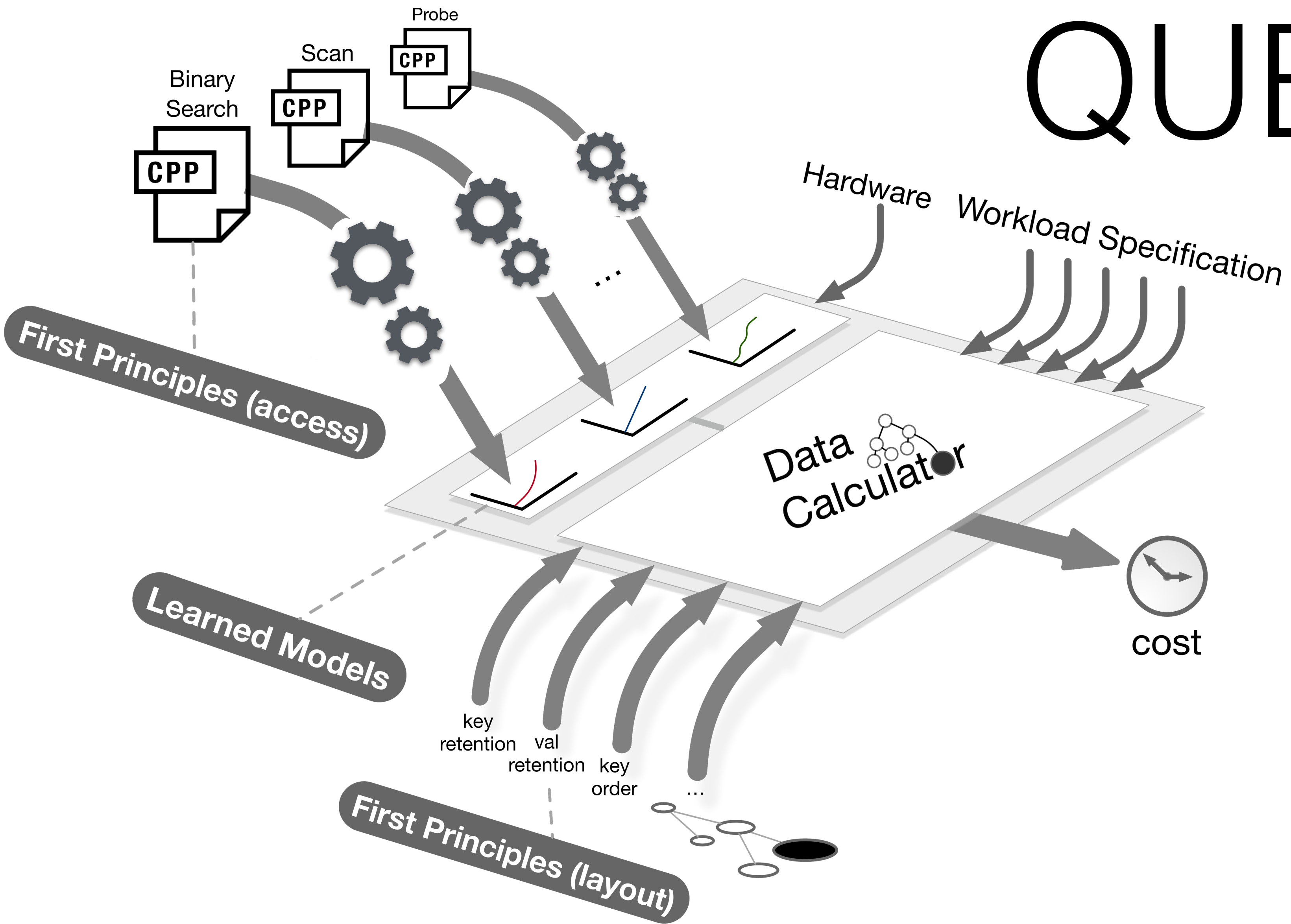
TRAINING



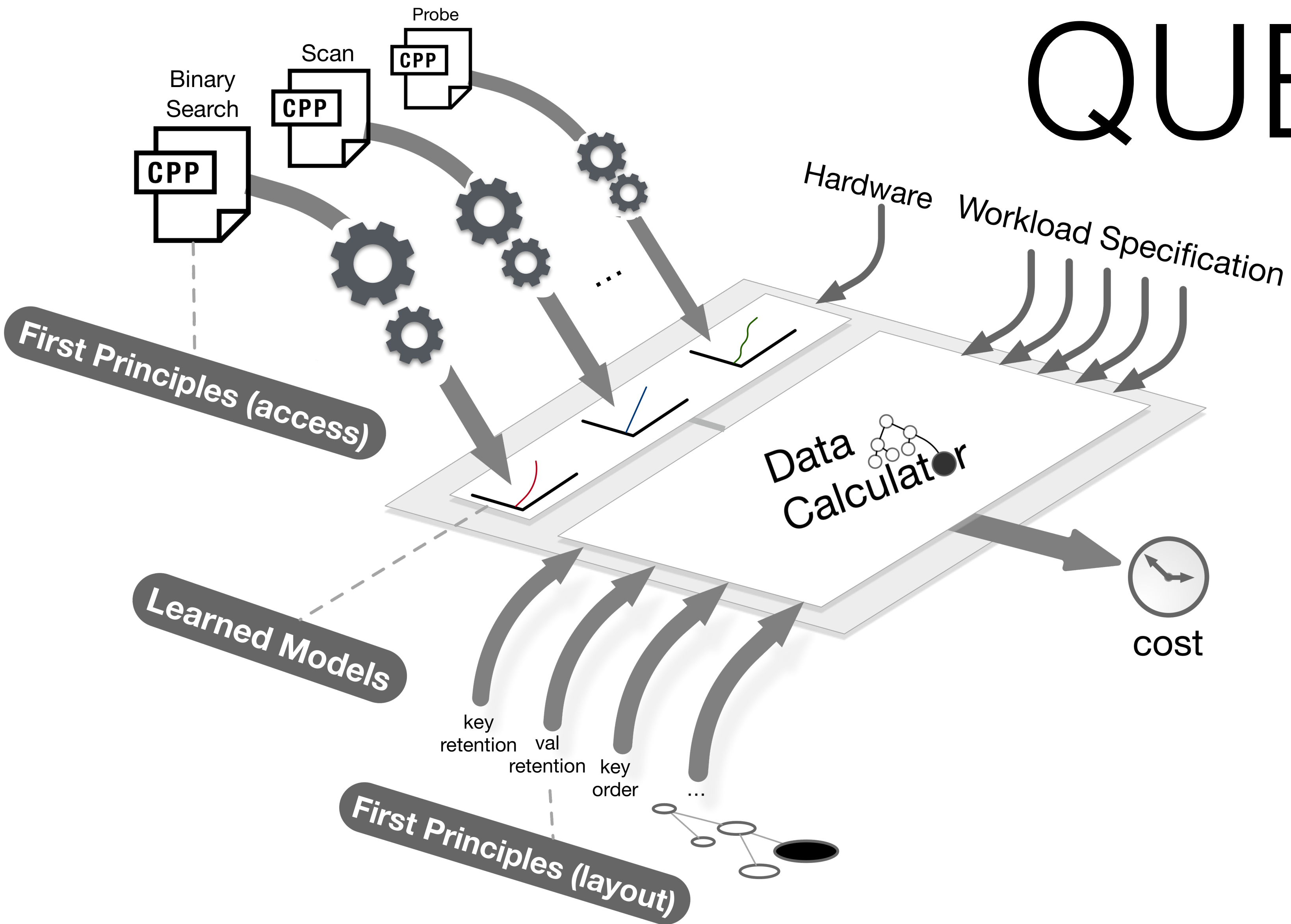
QUERYRYING



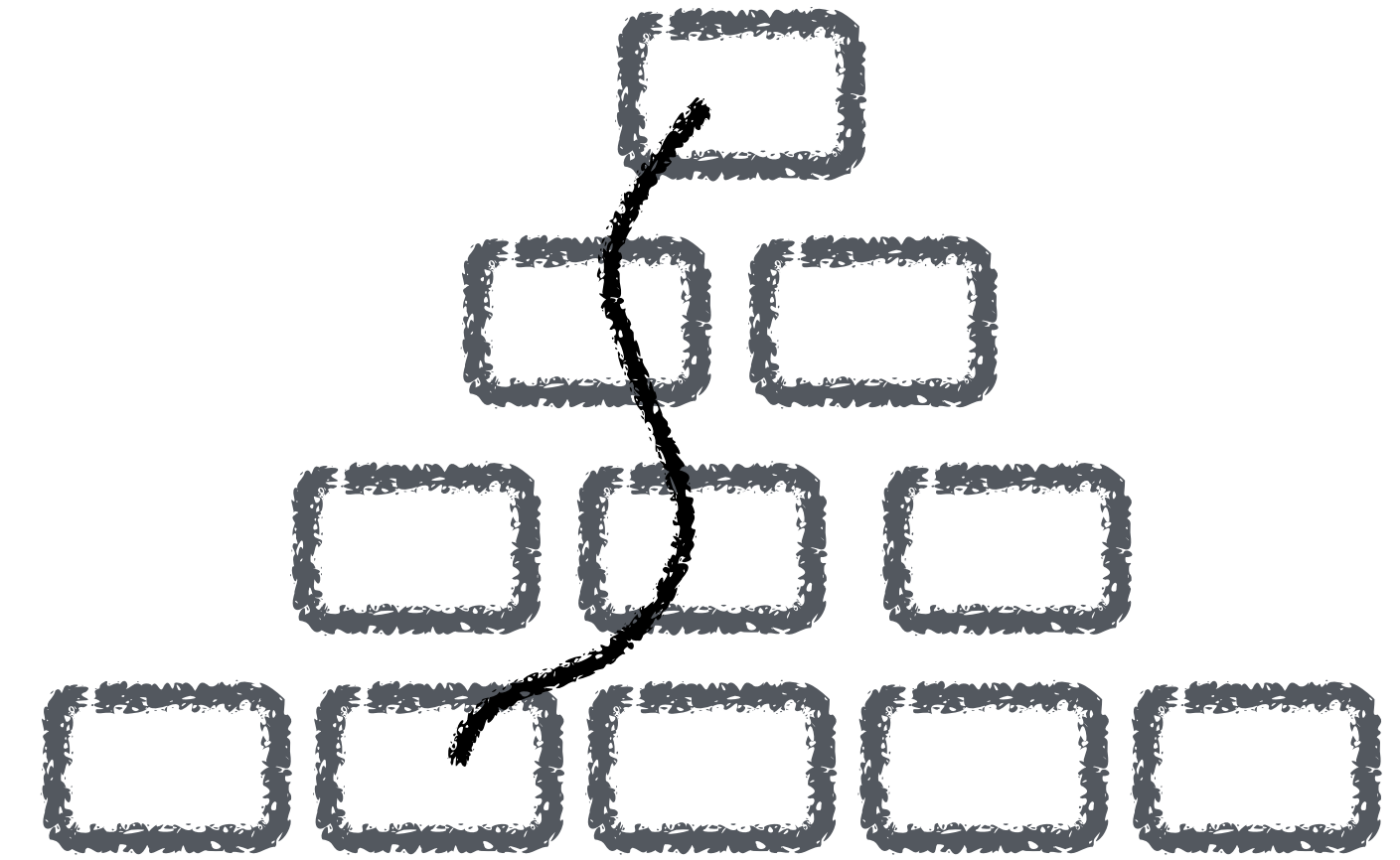
QUERYRYING



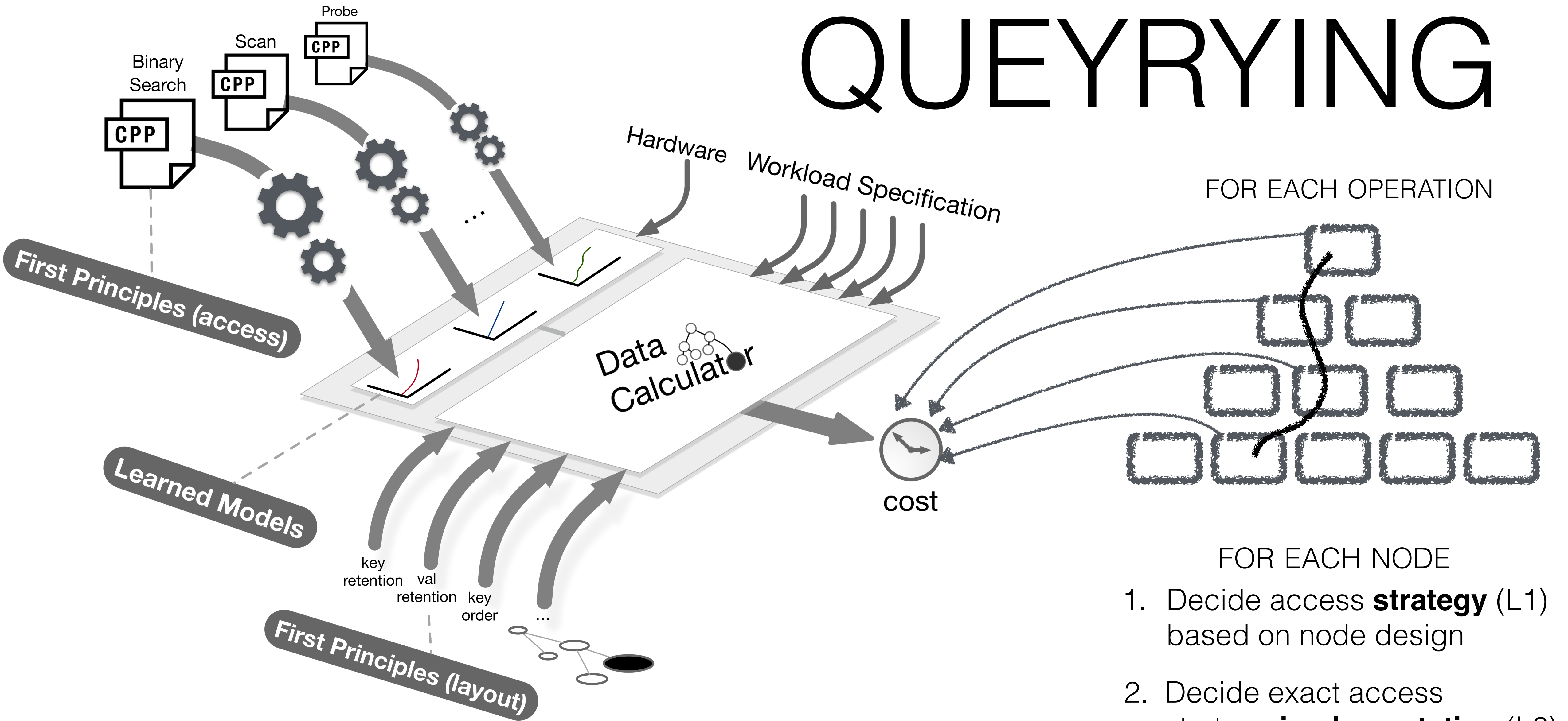
QUERYRYING

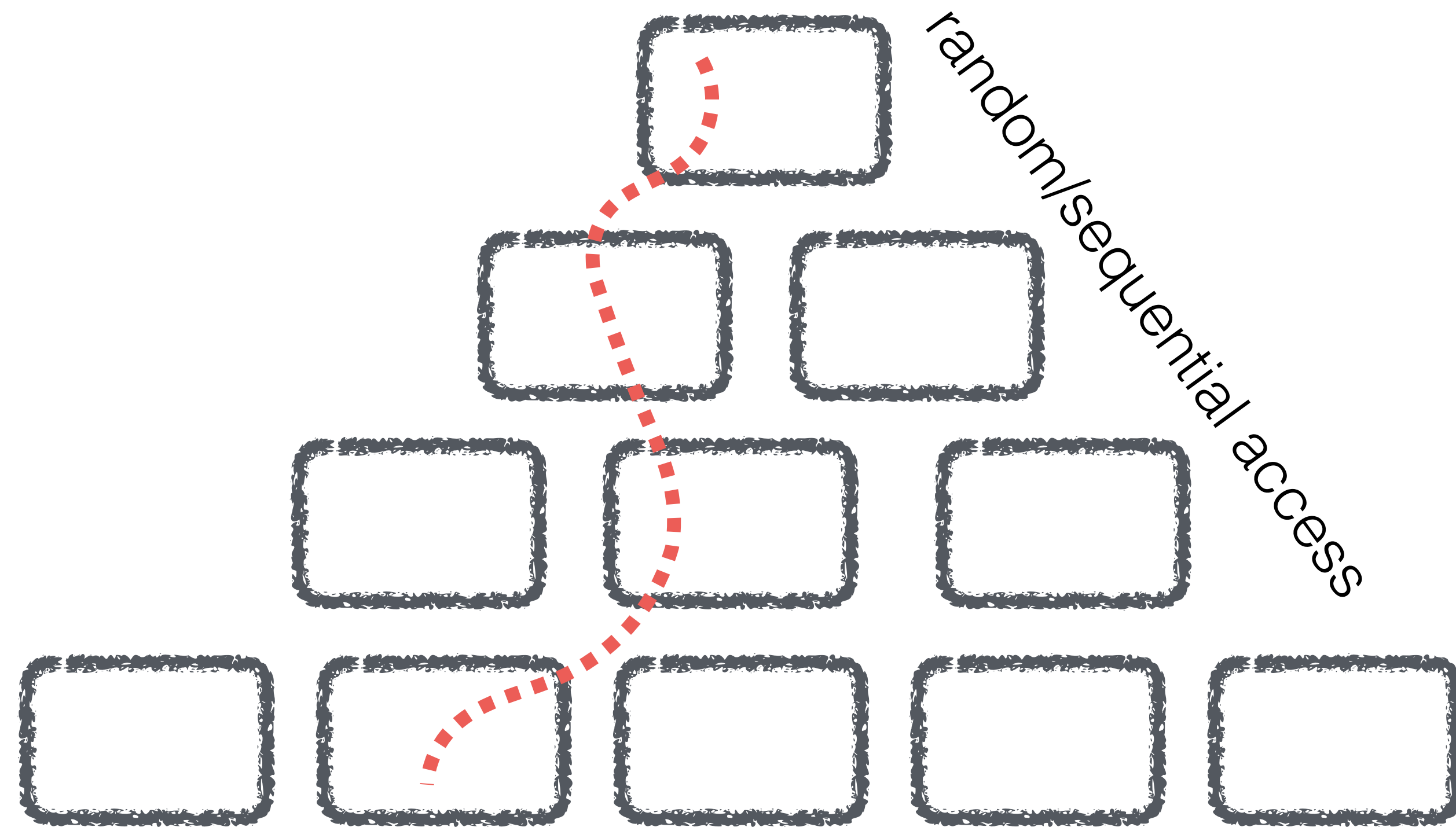


FOR EACH OPERATION



QUERYRYING

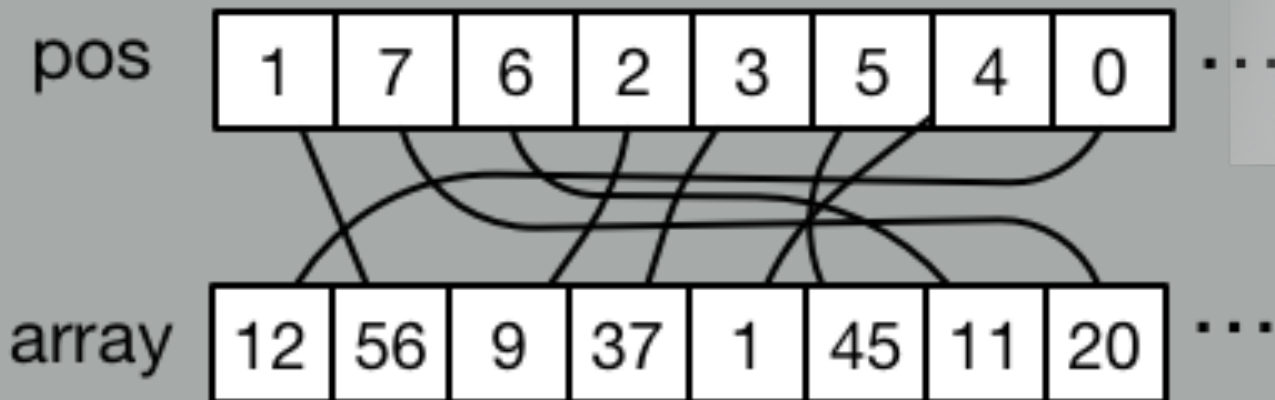




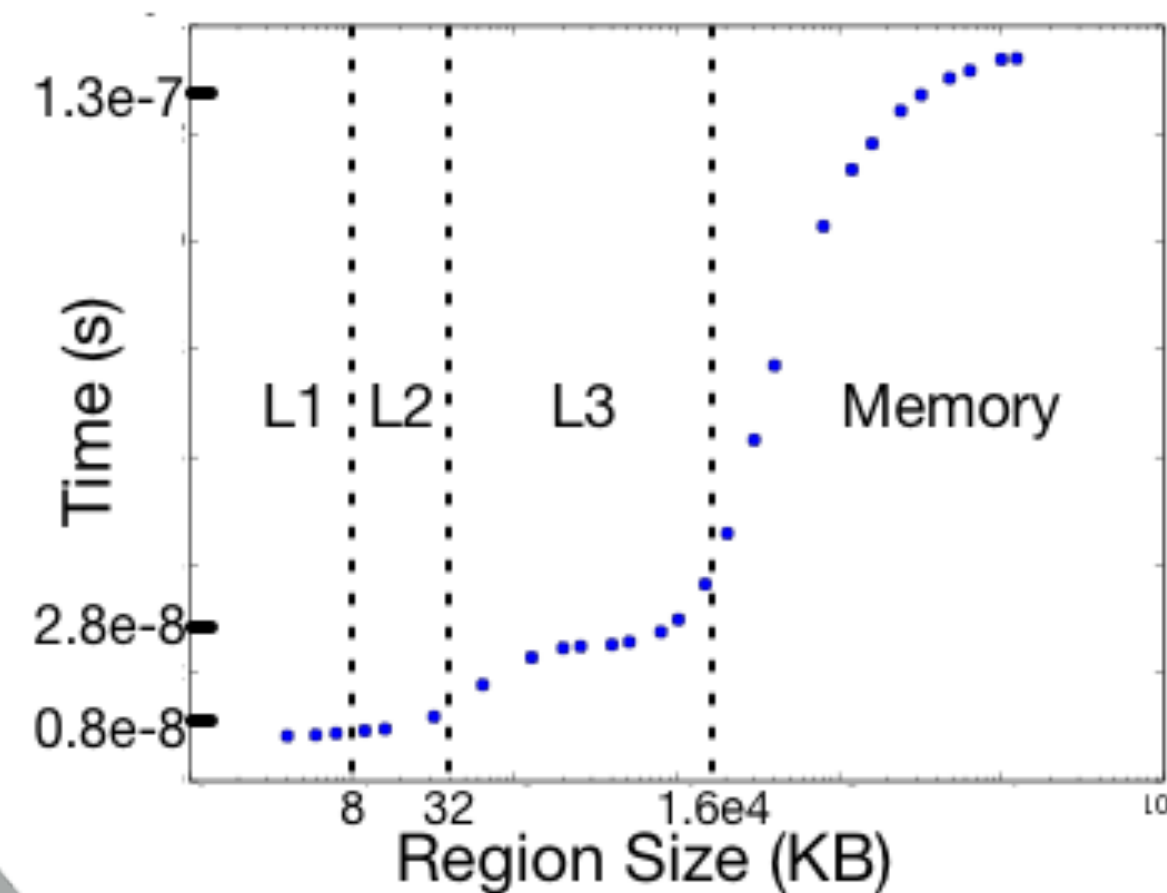
random access

C++

```
for(int i=0; i<size; i++)
  probe(array[pos[i]])
```

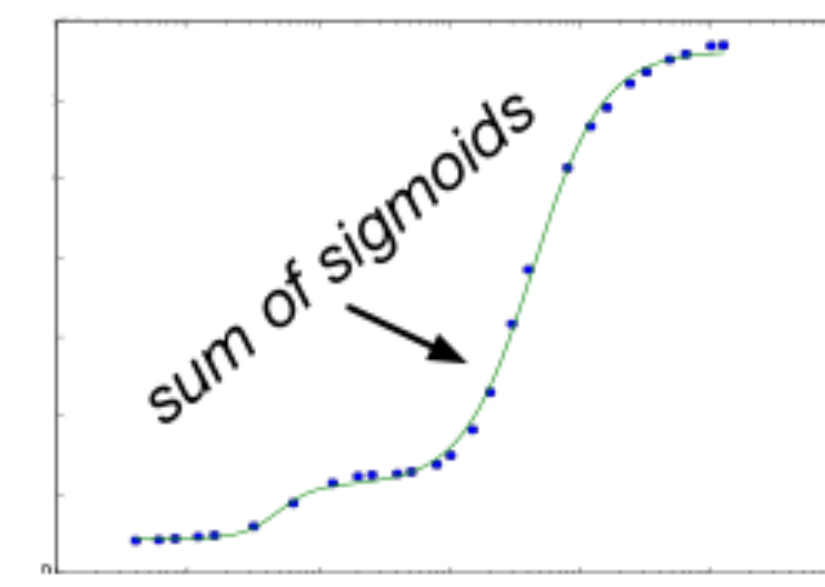


Run



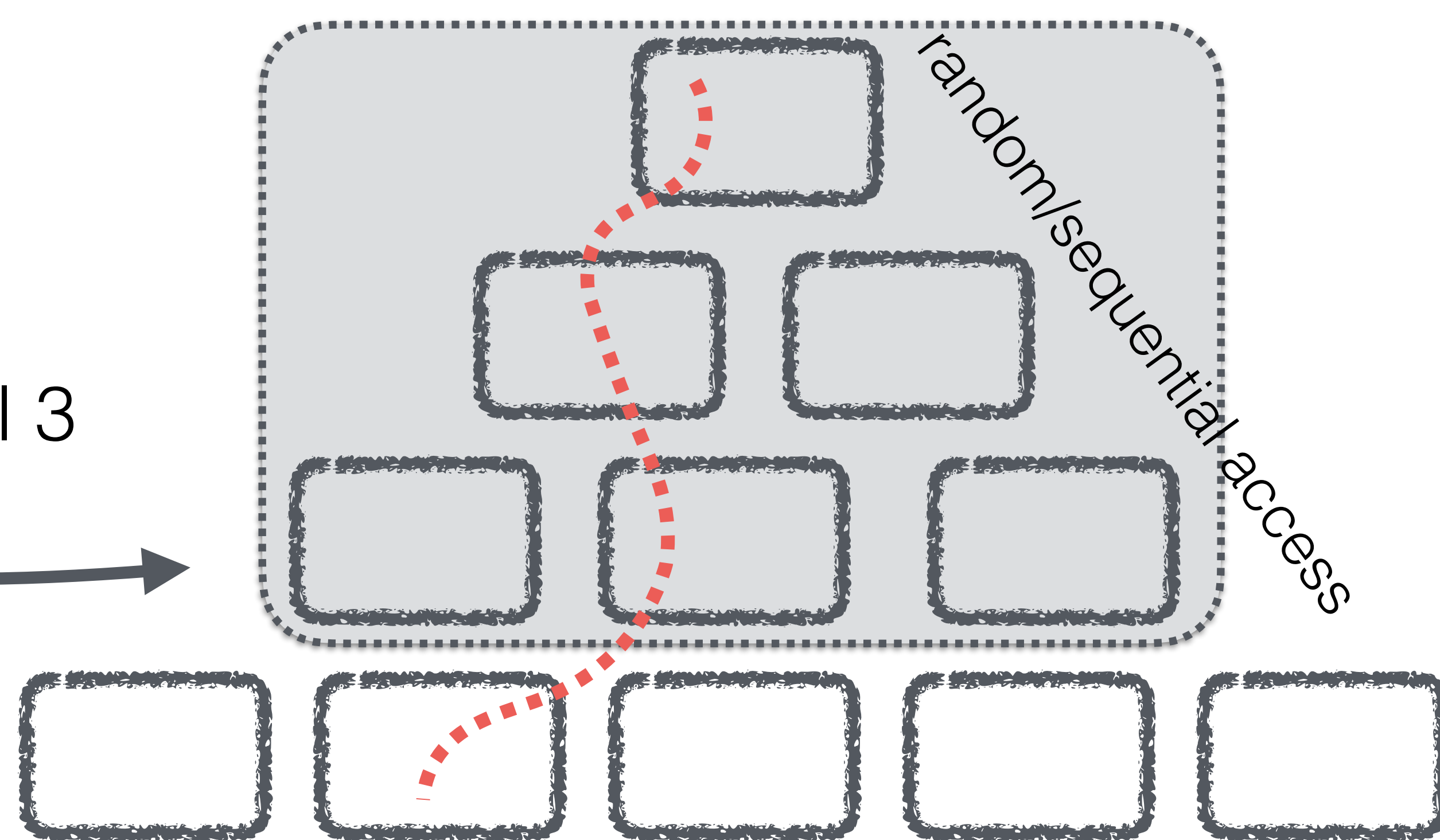
$f(x)$

Train



$$f(x) = \sum_i \frac{c_i}{1 + e^{-k_i(x - x_i)}}$$

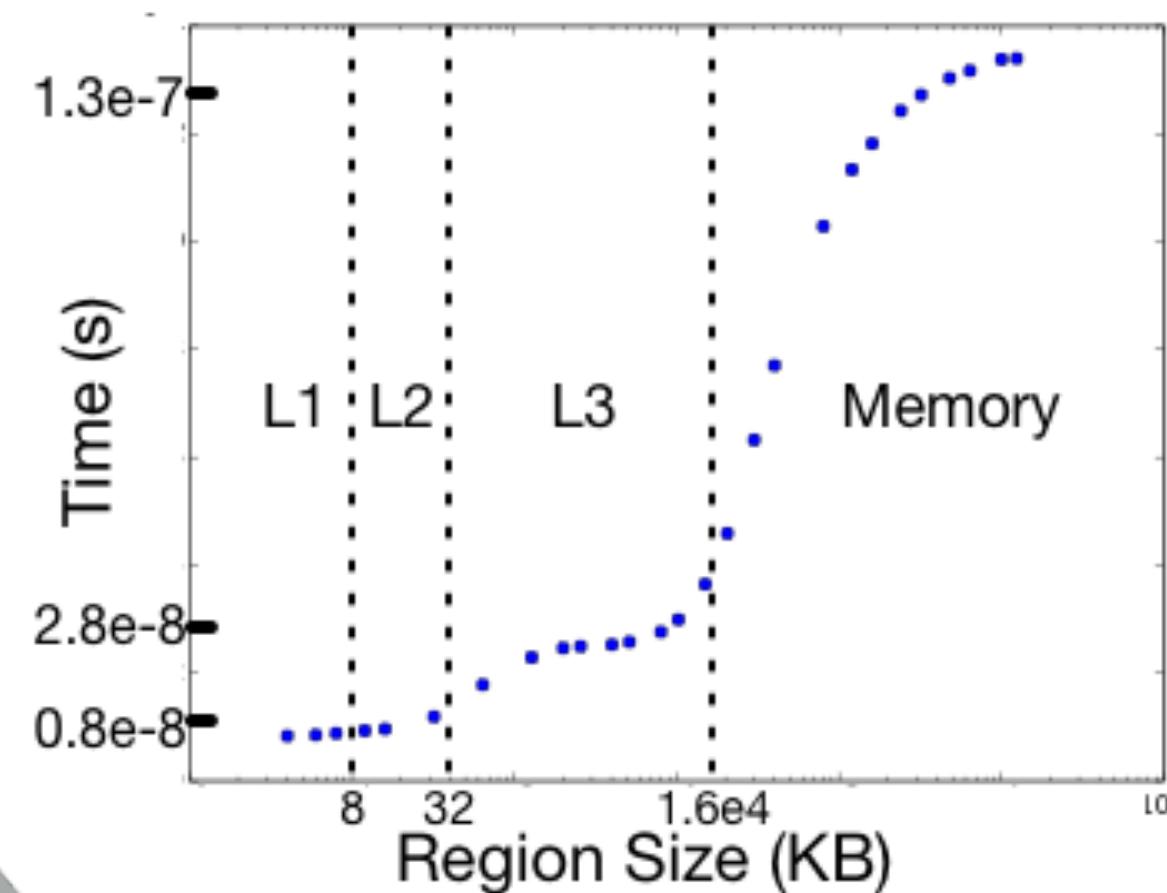
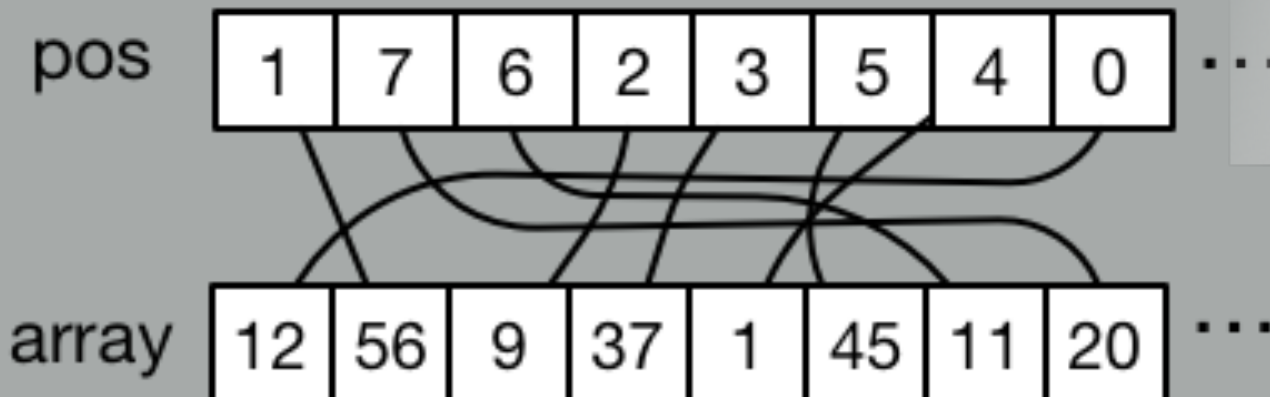
Accessing Level 3



random access

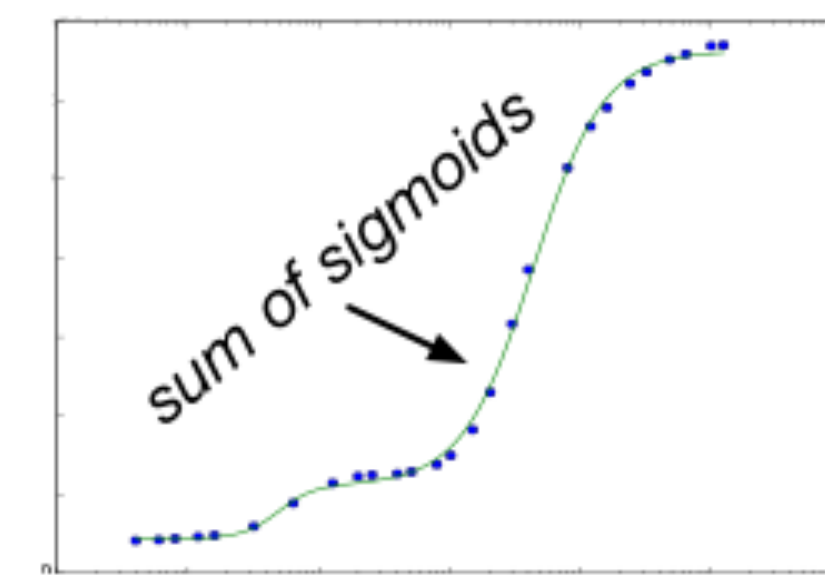
C++

```
for(int i=0; i<size; i++)
    probe(array[pos[i]])
```



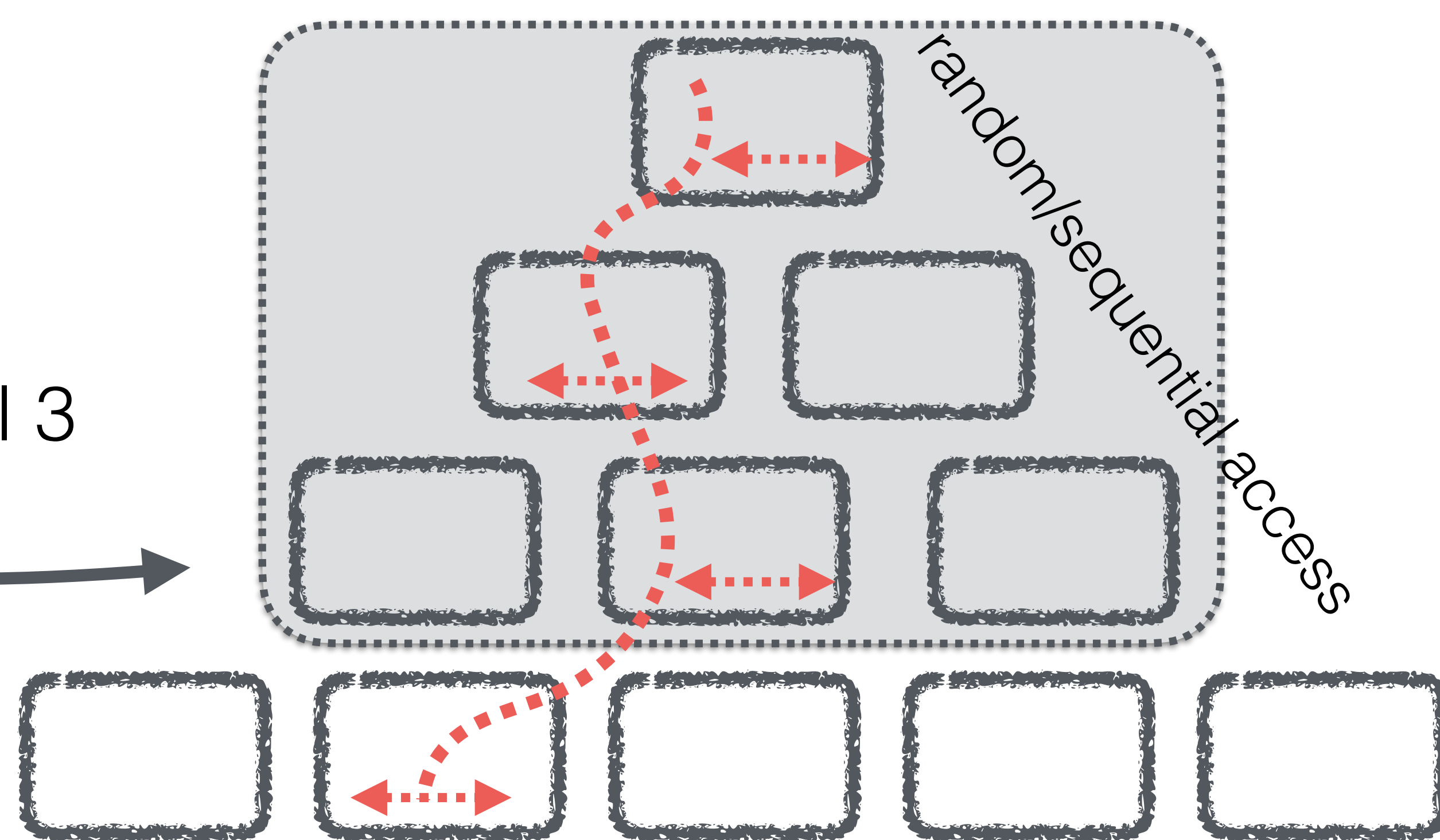
$f(x)$

Train



$$f(x) = \sum_i \frac{c_i}{1 + e^{-k_i(x - x_i)}}$$

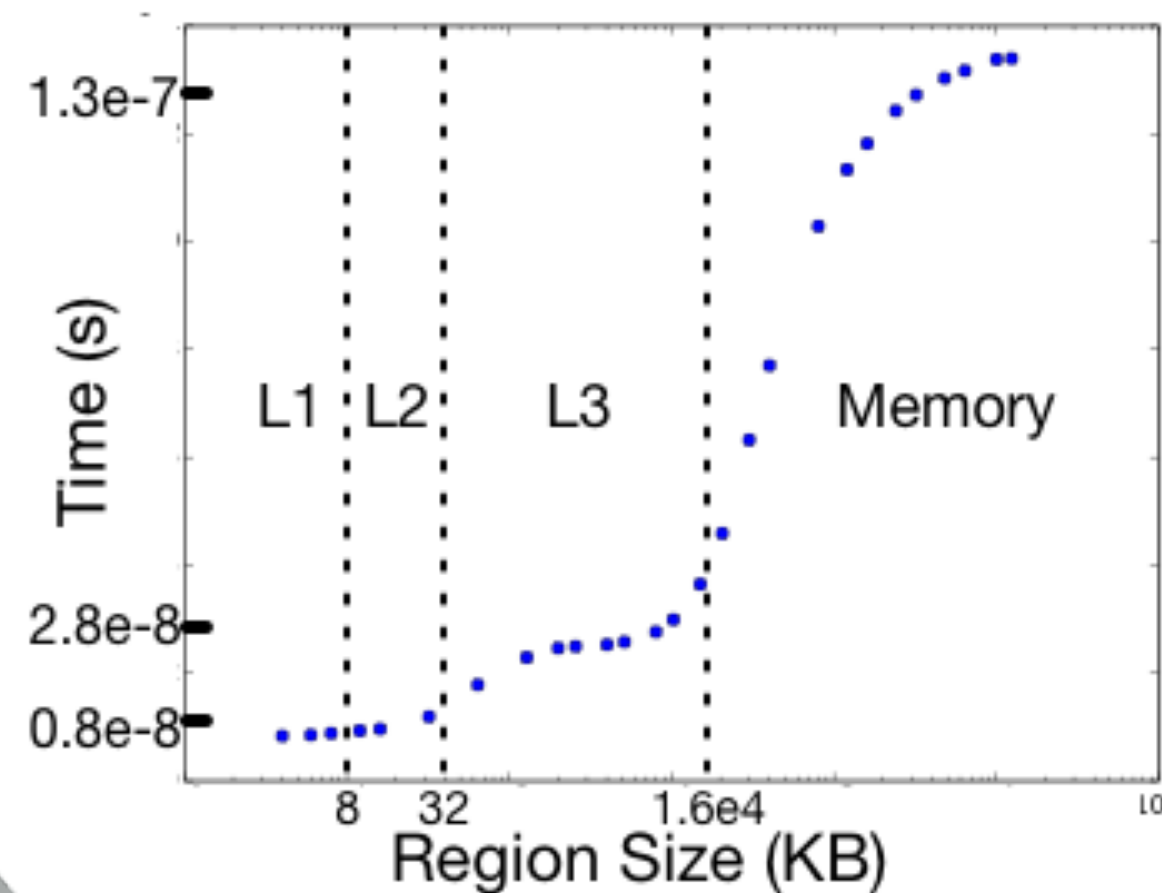
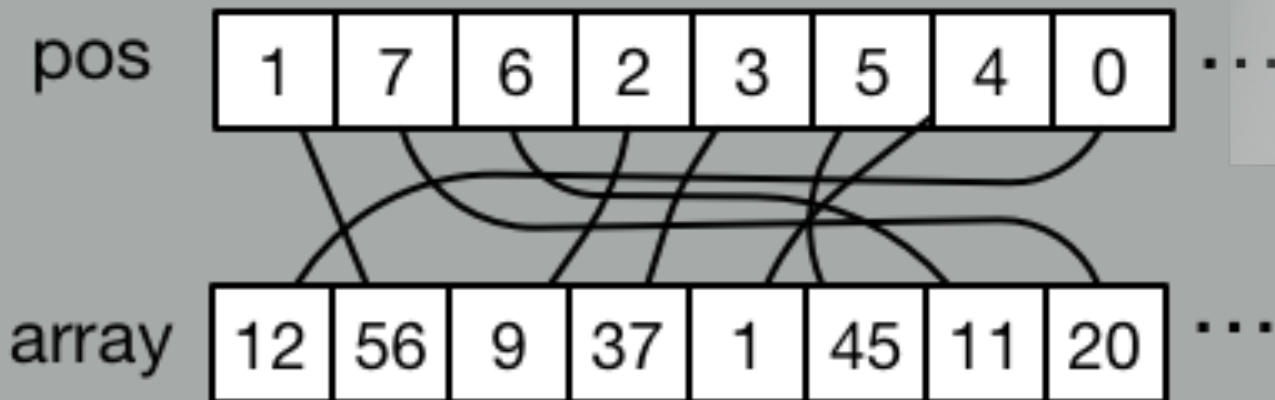
Accessing Level 3



random access

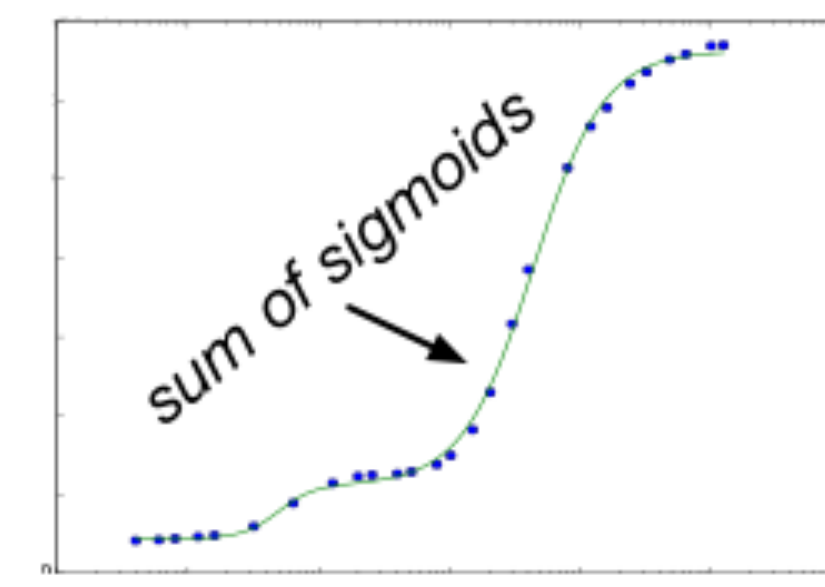
C++

```
for(int i=0; i<size; i++)
  probe(array[pos[i]])
```

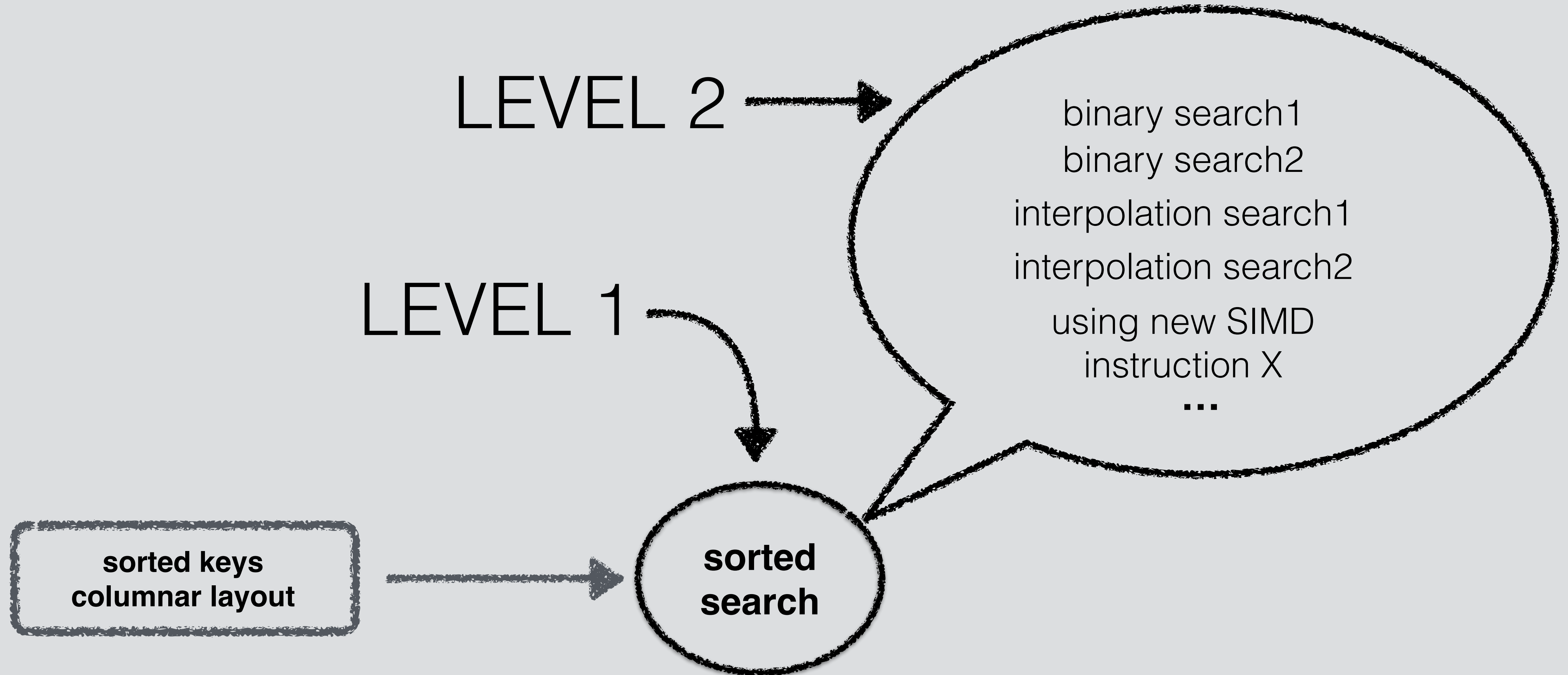


$f(x)$

Train



$$f(x) = \sum_i \frac{c_i}{1 + e^{-k_i(x - x_i)}}$$



EASY EXTENSIBILITY OF LEVEL 2 ACCESS PRIMITIVES

just adding a new benchmark for a Level 1 primitive
can be used in any design!

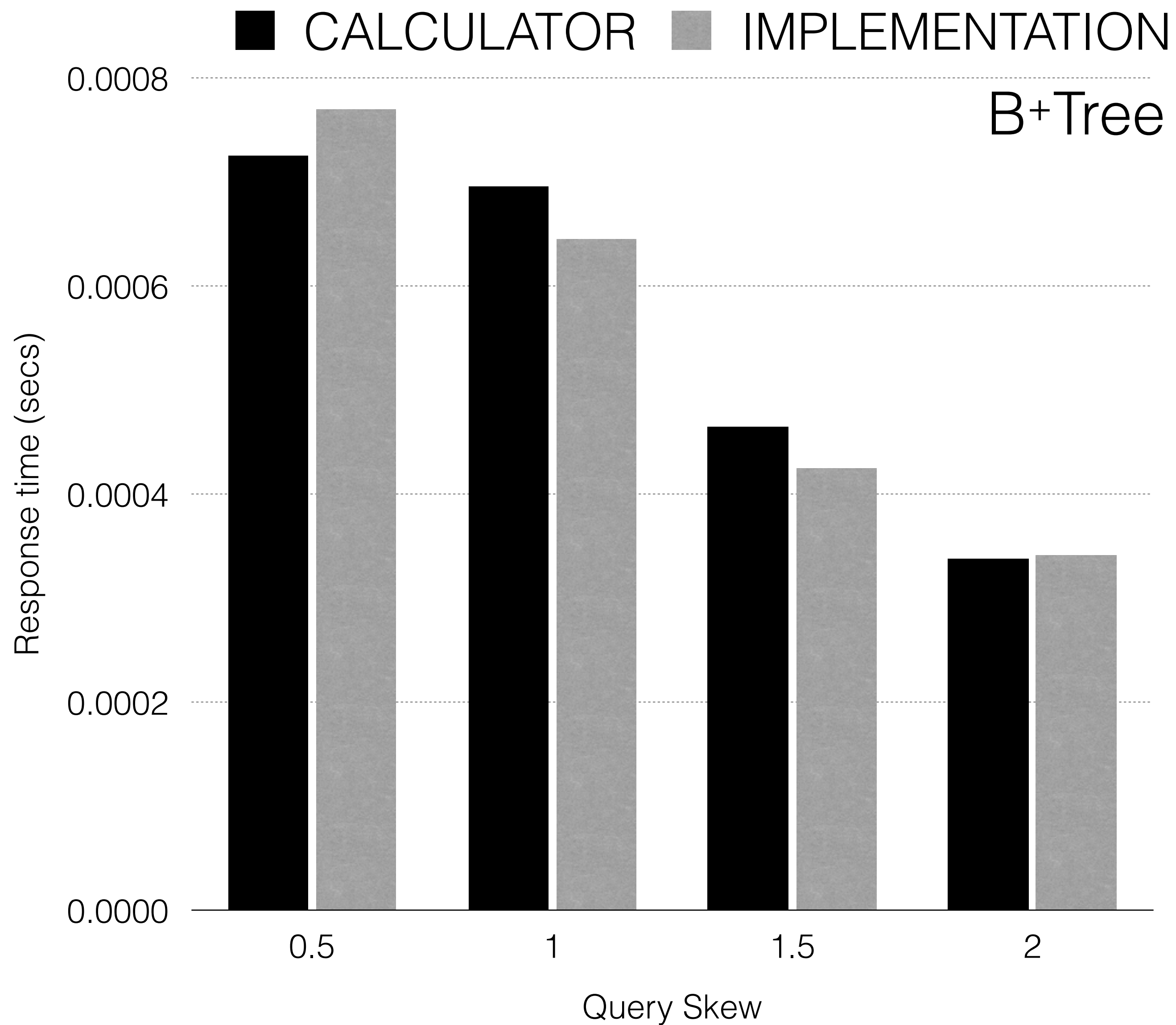
ACCESS PRIMITIVES

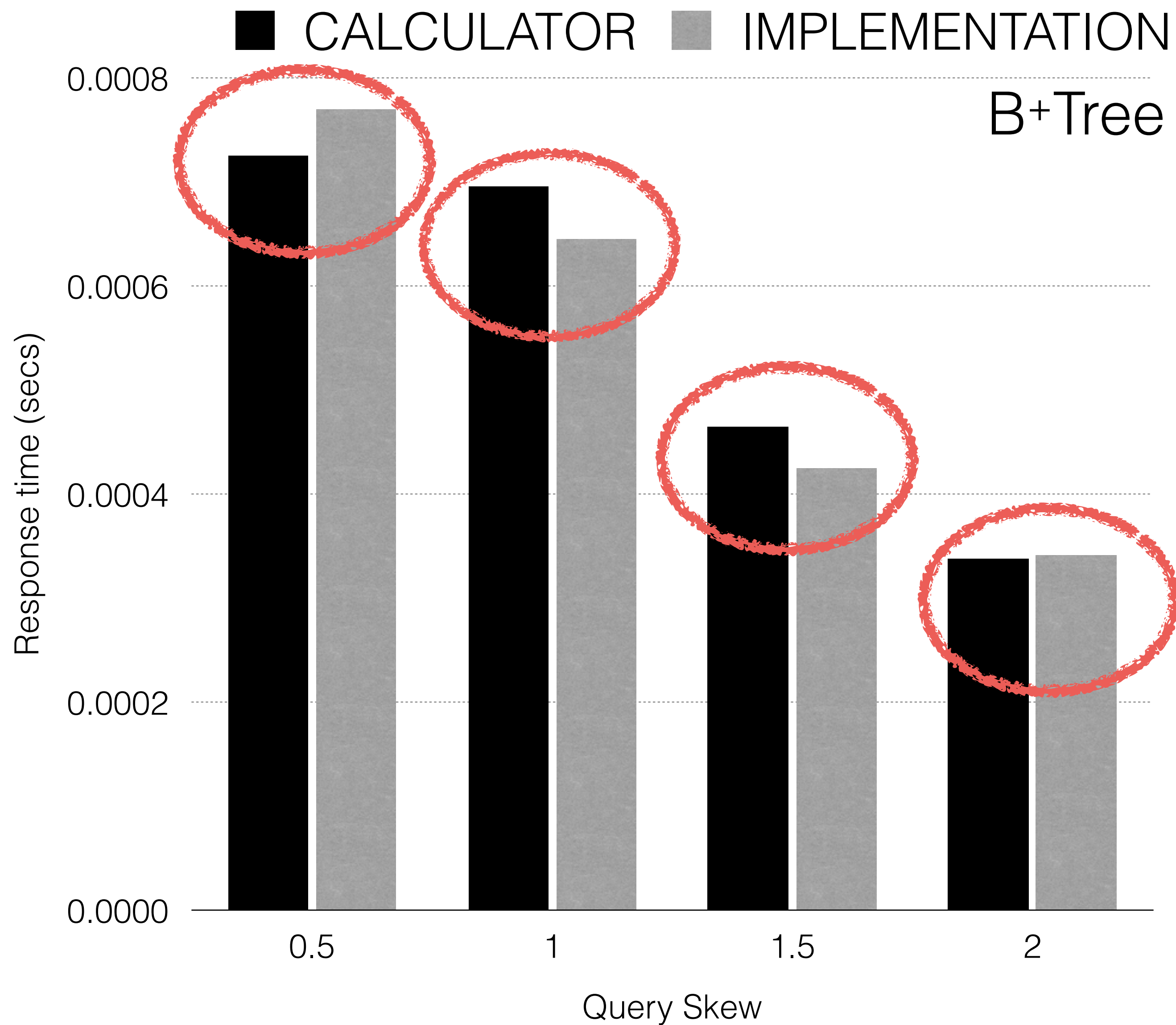
	Data Access Primitives Level 1 (required parameters ; optional parameters)	Model Parameters	Data Access Primitives Layer 2	Fitted Models
1	Scan (Element Size, Comparison, Data Layout; None)	Data Size	Scalar Scan (RowStore, Equal)	Linear Model (1)
2			Scalar Scan (RowStore, Range)	Linear Model (1)
3			Scalar Scan (ColumnStore, Equal)	Linear Model (1)
4			Scalar Scan (ColumnStore, Range)	Linear Model (1)
5			SIMD-AVX Scan (ColumnStore, Equal)	Linear Model (1)
6			SIMD-AVX Scan (ColumnStore, Range)	Linear Model (1)
7	Sorted Search (Element Size, Data Layout;)	Data Size	Binary Search (RowStore)	Log-Linear Model (2)
8			Binary Search (ColumnStore)	Log-Linear Model (2)
9			Interpolation Search (RowStore)	Log + LogLog Model (3)
10			Interpolation Search (ColumnStore)	Log + LogLog Model (3)
11	Hash Probe (; Hash Family)	Structure Size	Linear Probing (Multiply-shift [29])	Sum of Sigmoids (5), Weighted Nearest Neighbors (7)
12			Linear Probing (k-wise independent, k=2,3,4,5)	Sum of Sigmoids (5), Weighted Nearest Neighbors (7)
13	Bloom Filter Probe (; Hash Family)	Structure Size, Number of Hash Functions	Bloom Filter Probe (Multiply-shift [29])	Sum of Sum of Sigmoids (6), Weighted Nearest Neighbors (7)
14			Bloom Filter Probe (k-wise independent, k=2,3,4,5)	Sum of Sum of Sigmoids (6), Weighted Nearest Neighbors (7)
15	Sort (Element Size; Algorithm)	Data Size	QuickSort	NLogN Model (4)
16			MergeSort	NLogN Model (4)
17			ExternalMergeSort	NLogN Model (4)
18	Random Memory Access	Region Size	Random Memory Access	Sum of Sigmoids (5), Weighted Nearest Neighbors (7)
19	Batched Random Memory Access	Region Size	Batched Random Memory Access	Sum of Sigmoids (5), Weighted Nearest Neighbors (7)
20	Unordered Batch Write (Layout;)	Write Data Size	Contiguous Write (RowStore)	Linear Model (1)
21			Contiguous Write (ColumnStore)	Linear Model (1)
22	Ordered Batch Write (Layout;)	Write Data Size, Data Size	Batch Ordered Write (RowStore)	Linear Model (1)
23			Batch Ordered Write (ColumnStore)	Linear Model (1)
24	Scattered Batch Write	Number of Elements, Region Size	ScatteredBatchWrite	Sum of Sum of Sigmoids (6), Weighted Nearest Neighbors (7)

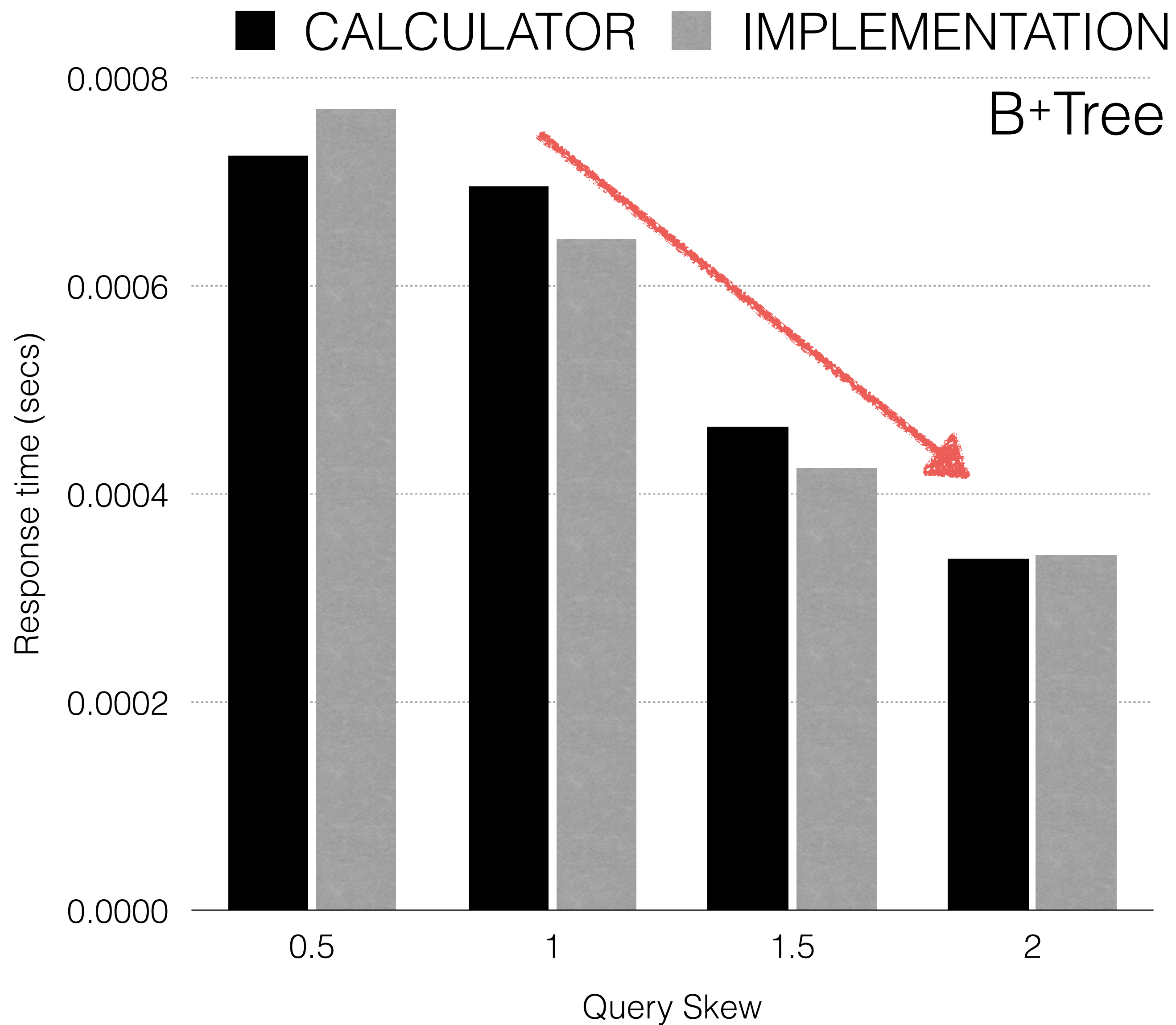
CAN WE COMPUTE PERFORMANCE ACCURATELY?

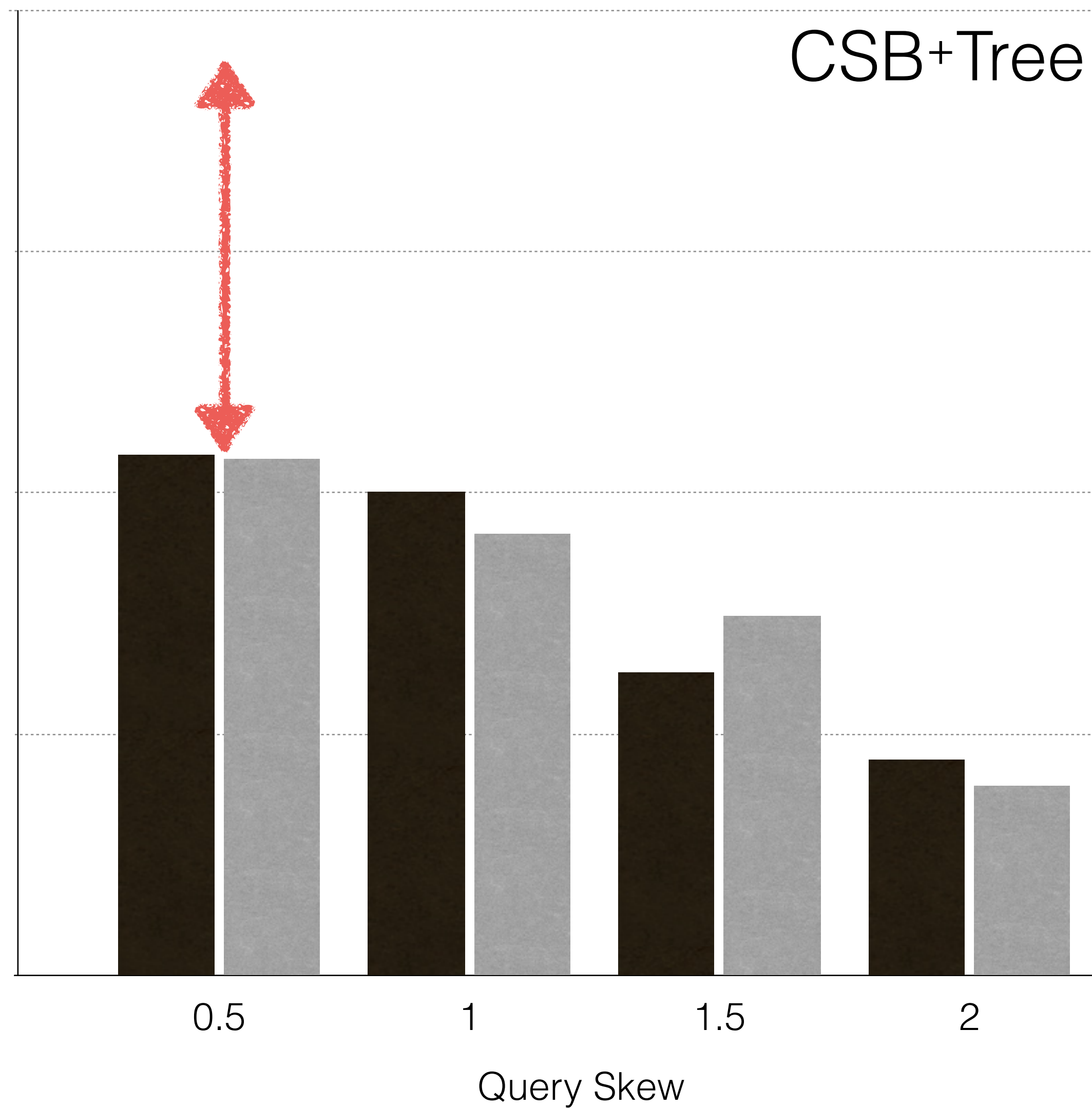
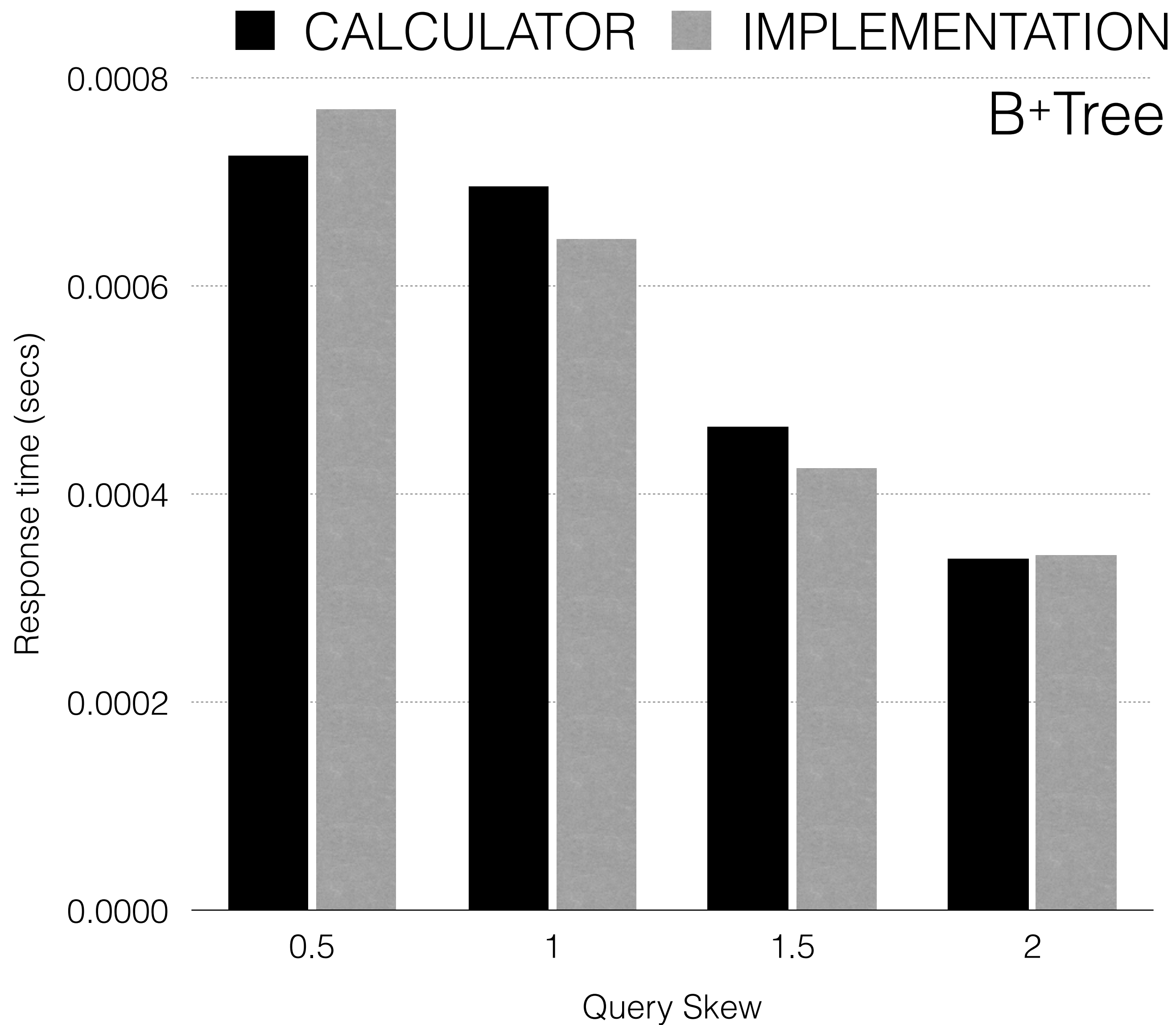
CAN WE COMPUTE PERFORMANCE ACCURATELY?

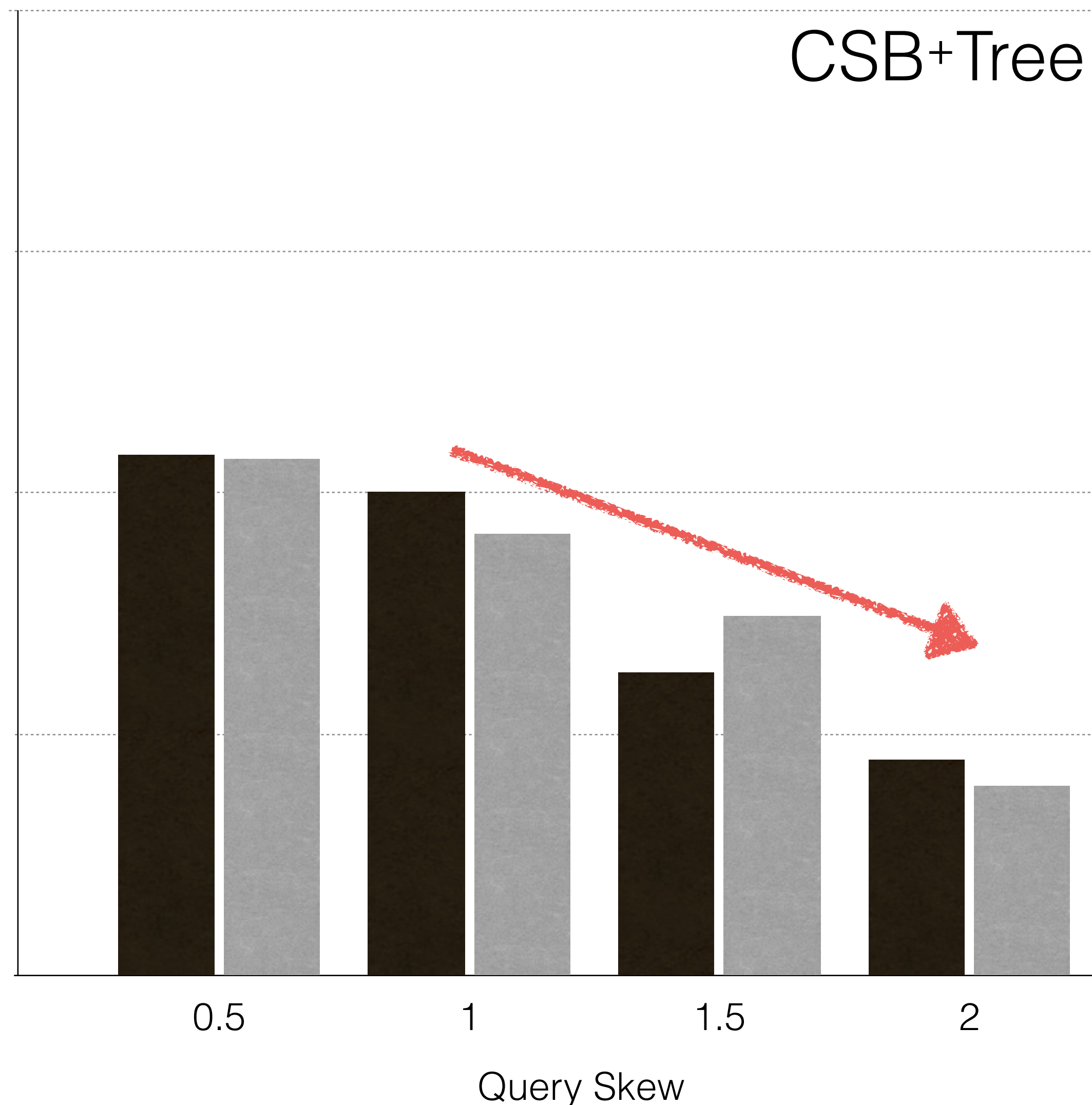
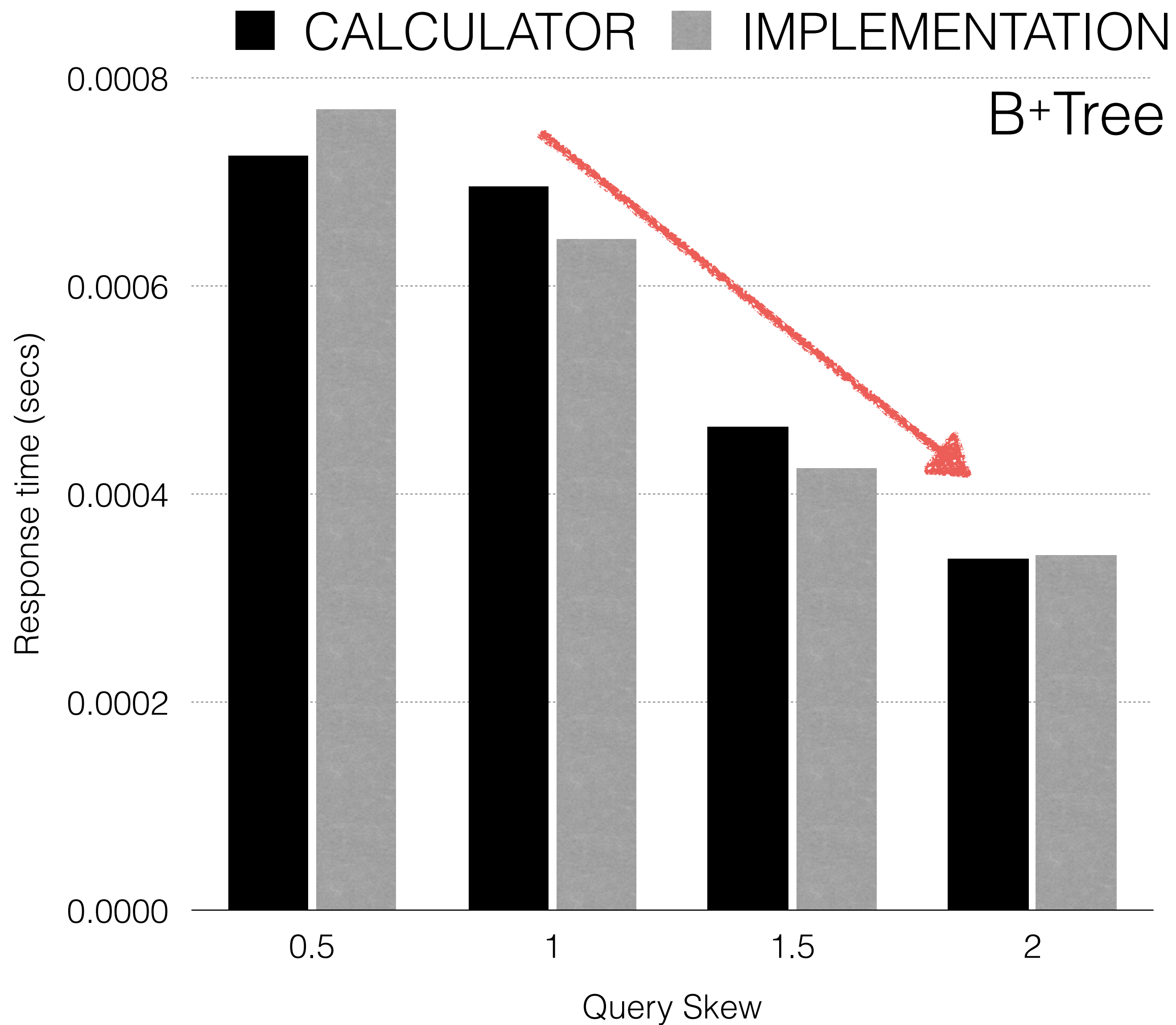






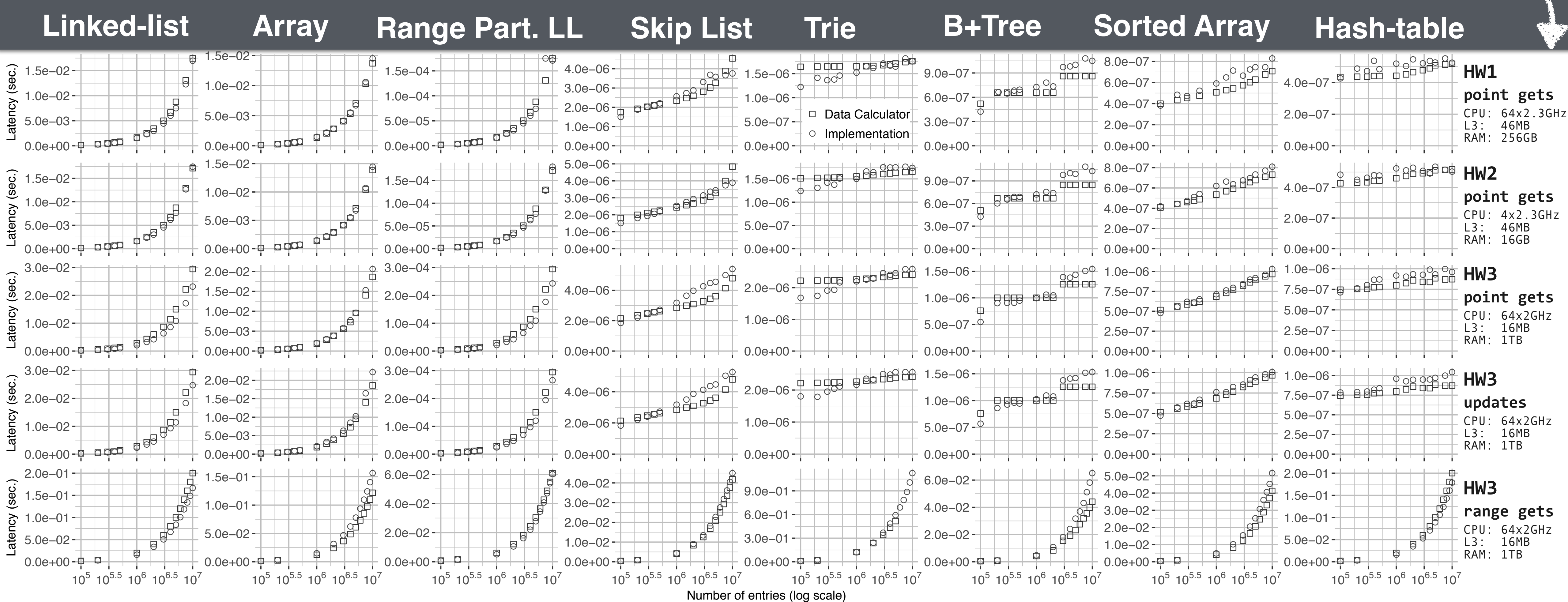




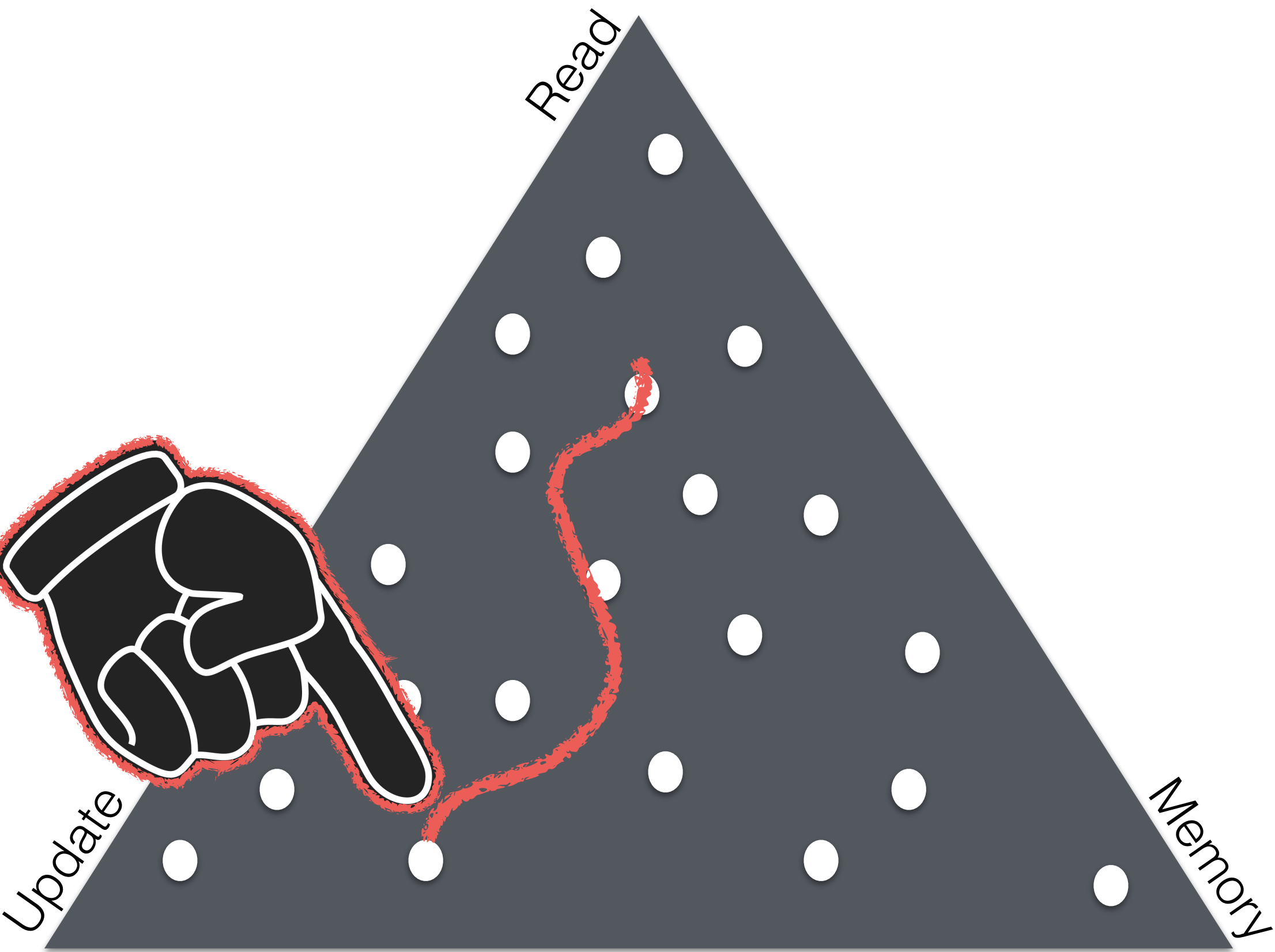
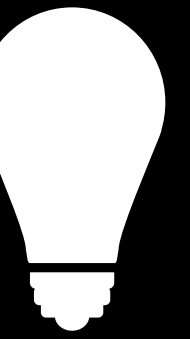


It works for numerous data structure classes and for diverse hardware and operations

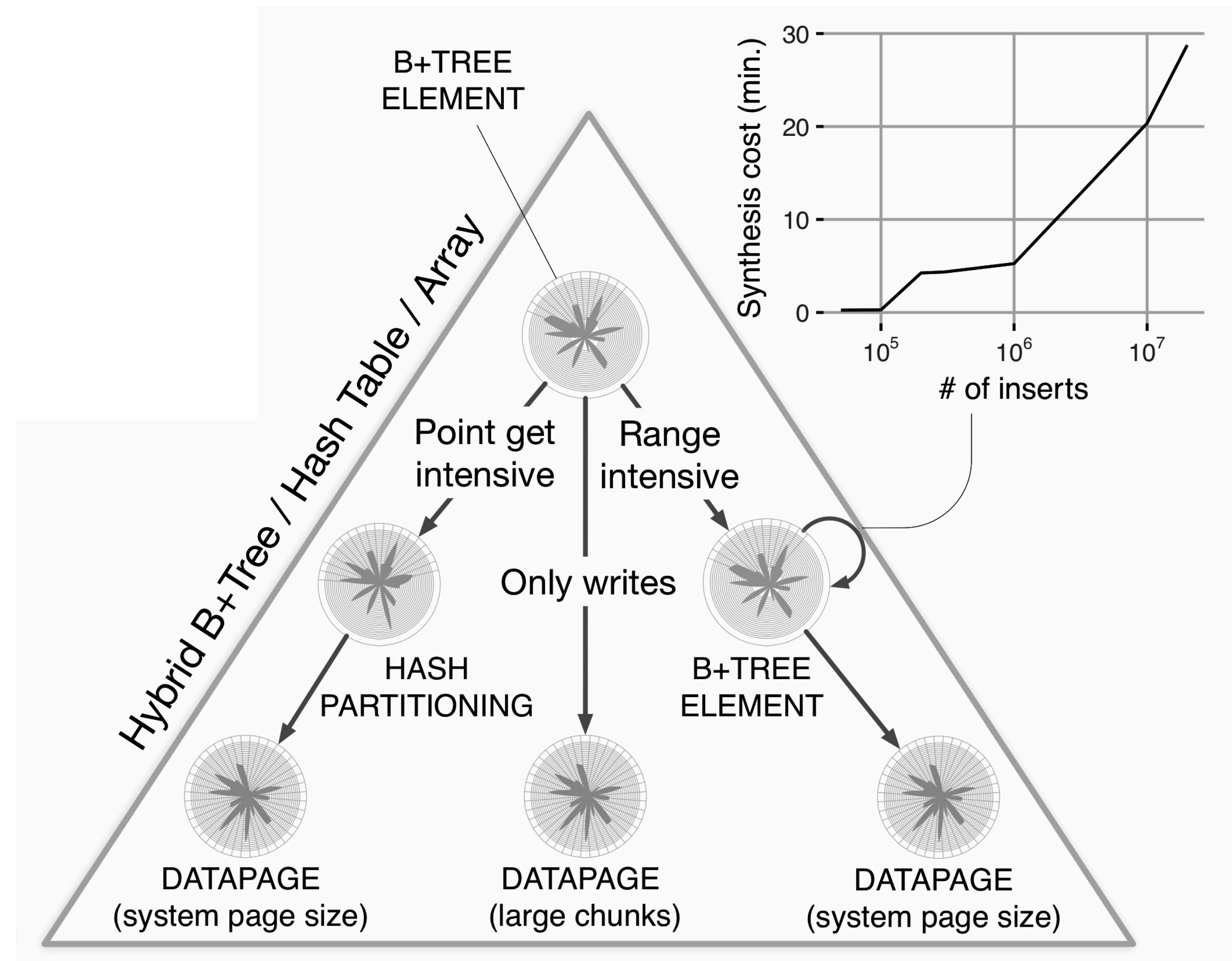
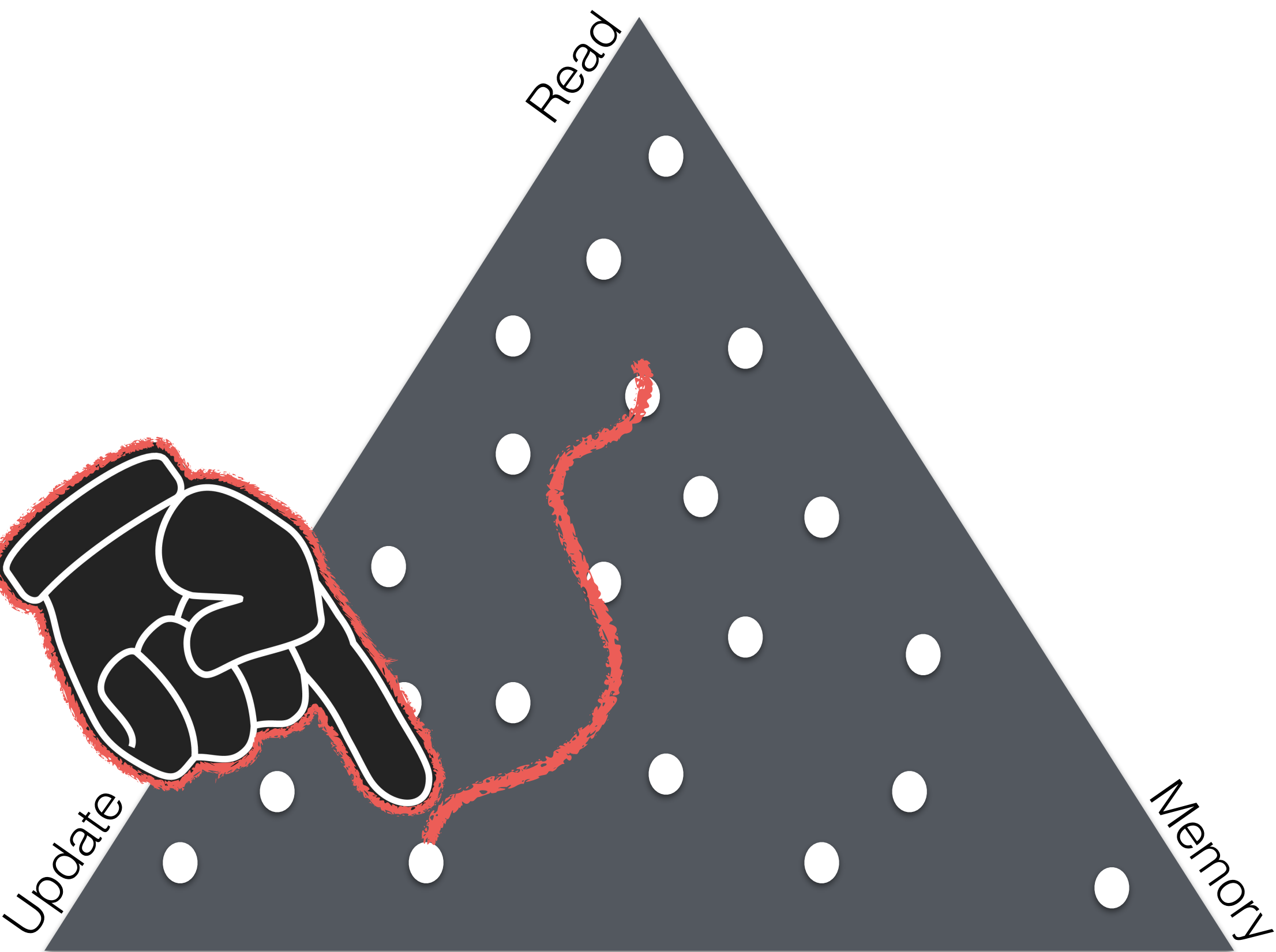
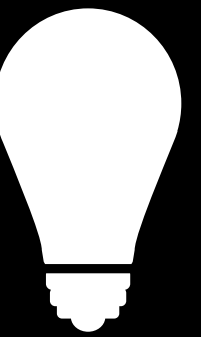
Training cost 50-100 secs



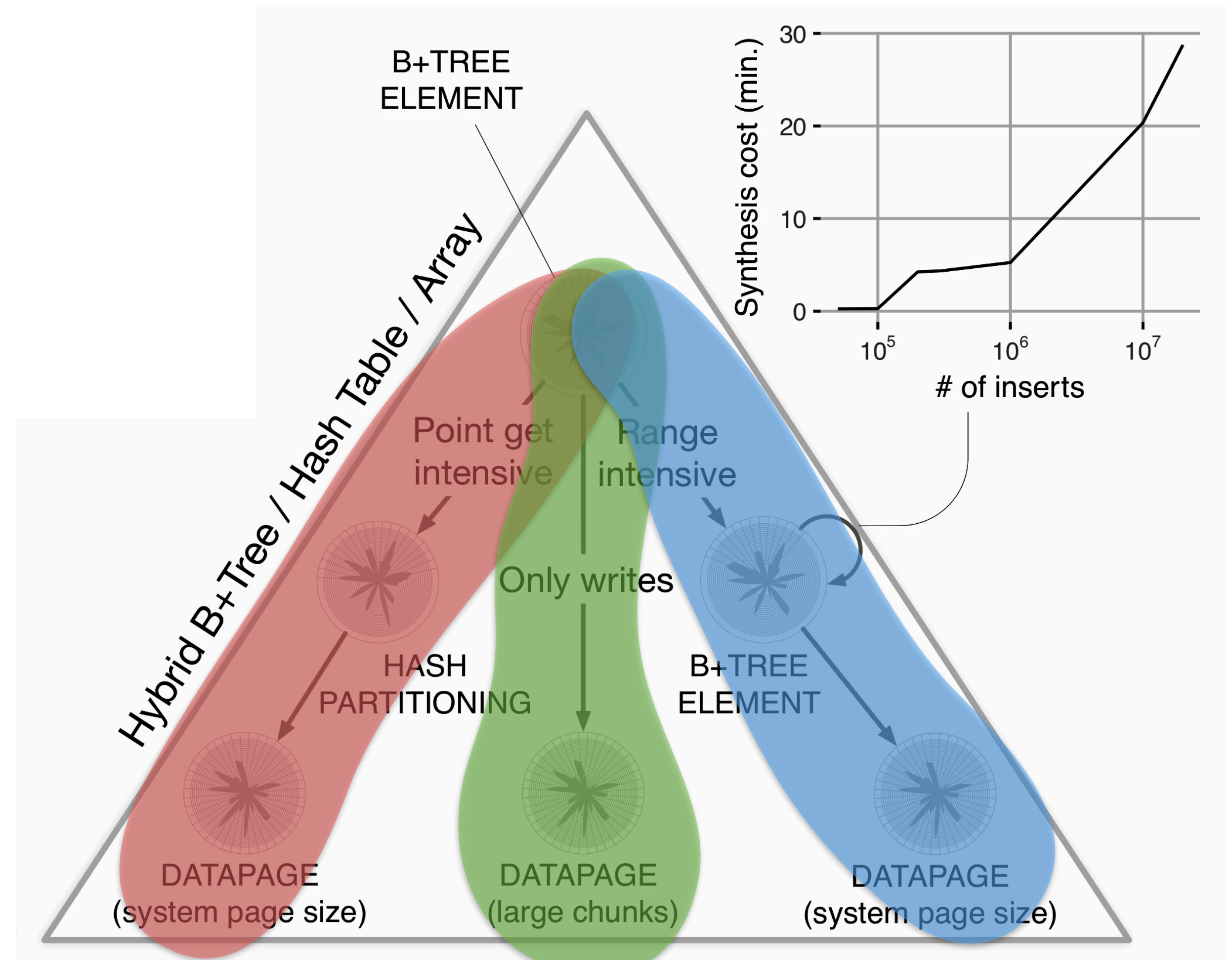
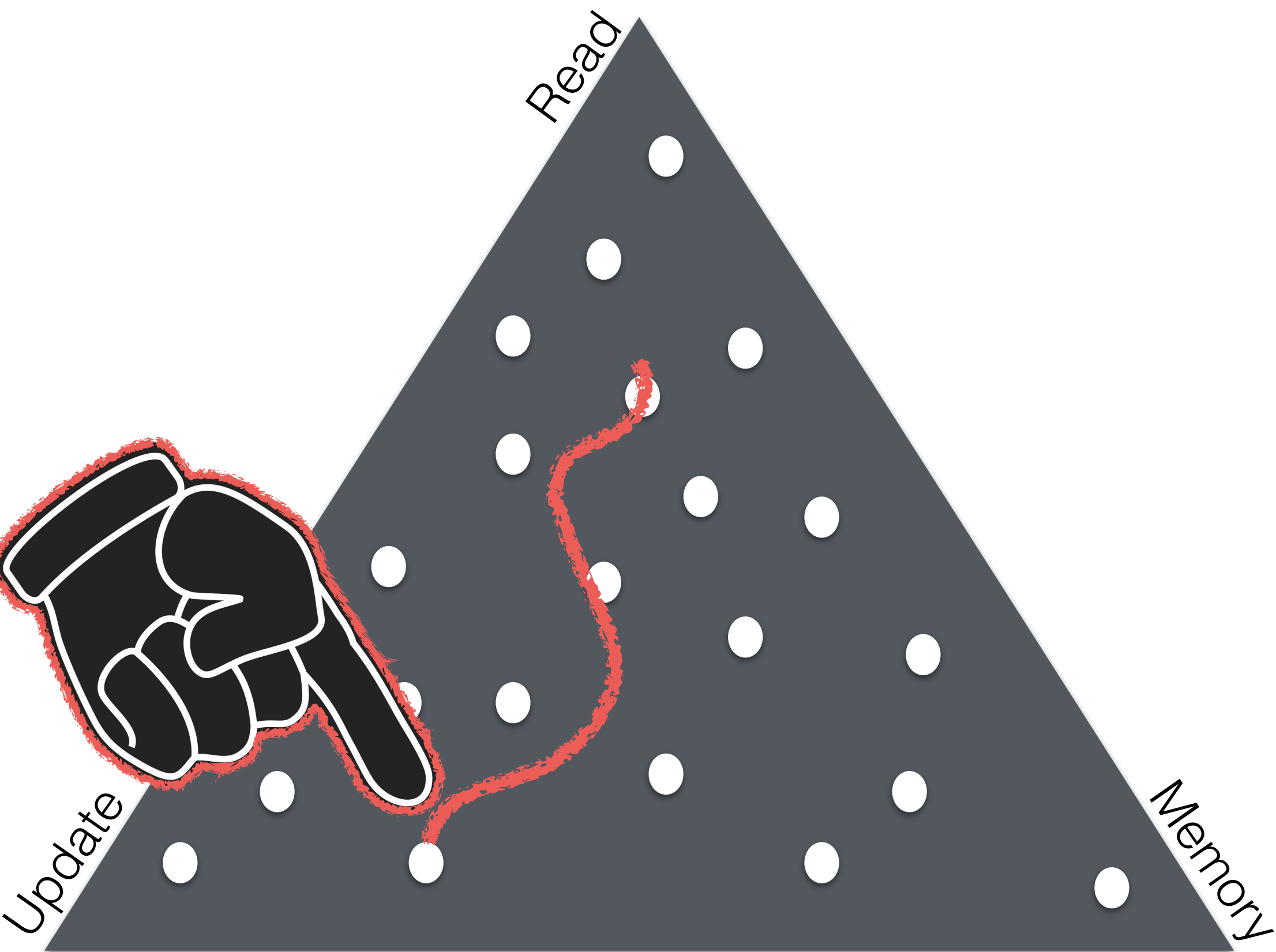
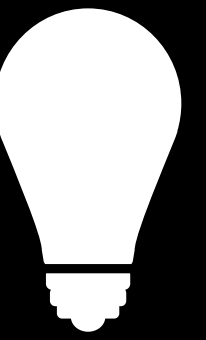
WE CAN AUTOMATICALLY DESIGN/DEBUG/FILL IN

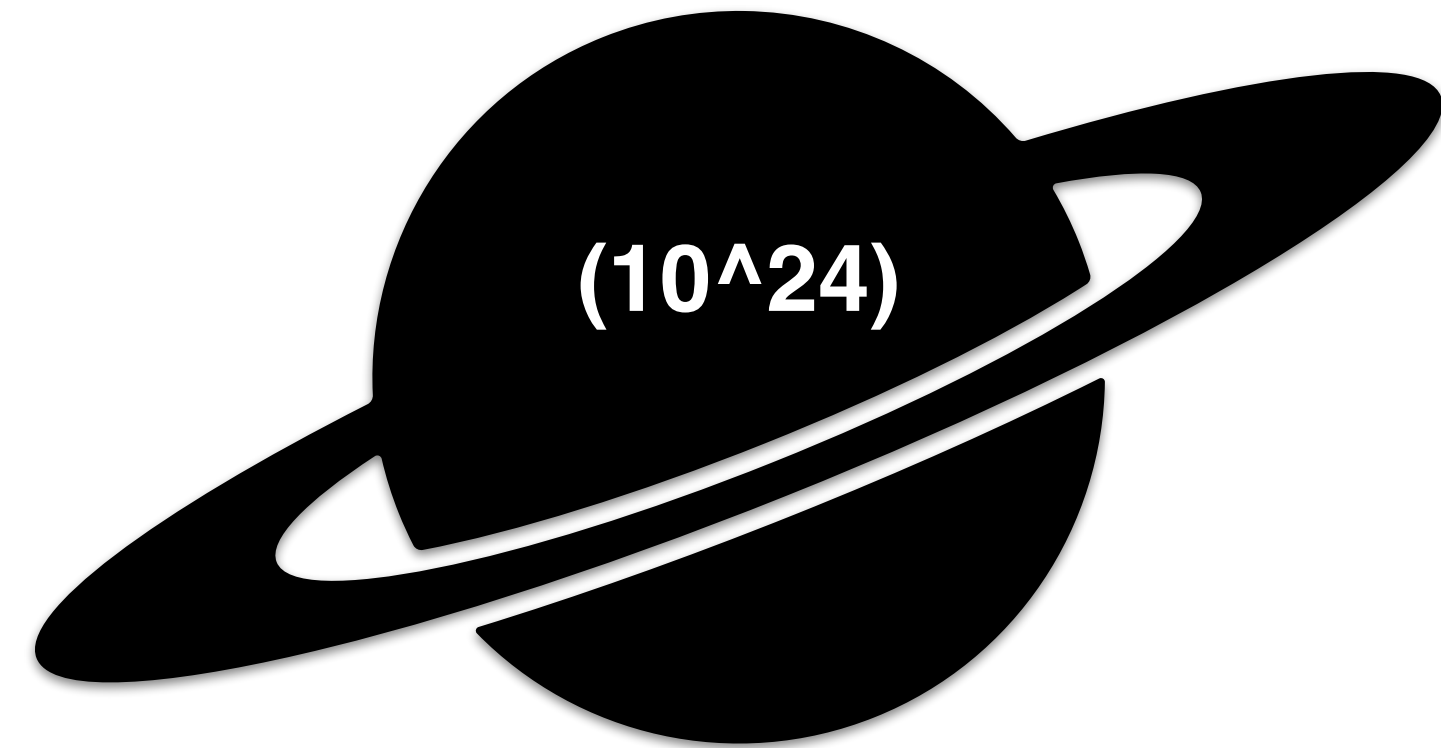


WE CAN AUTOMATICALLY DESIGN/DEBUG/FILL IN

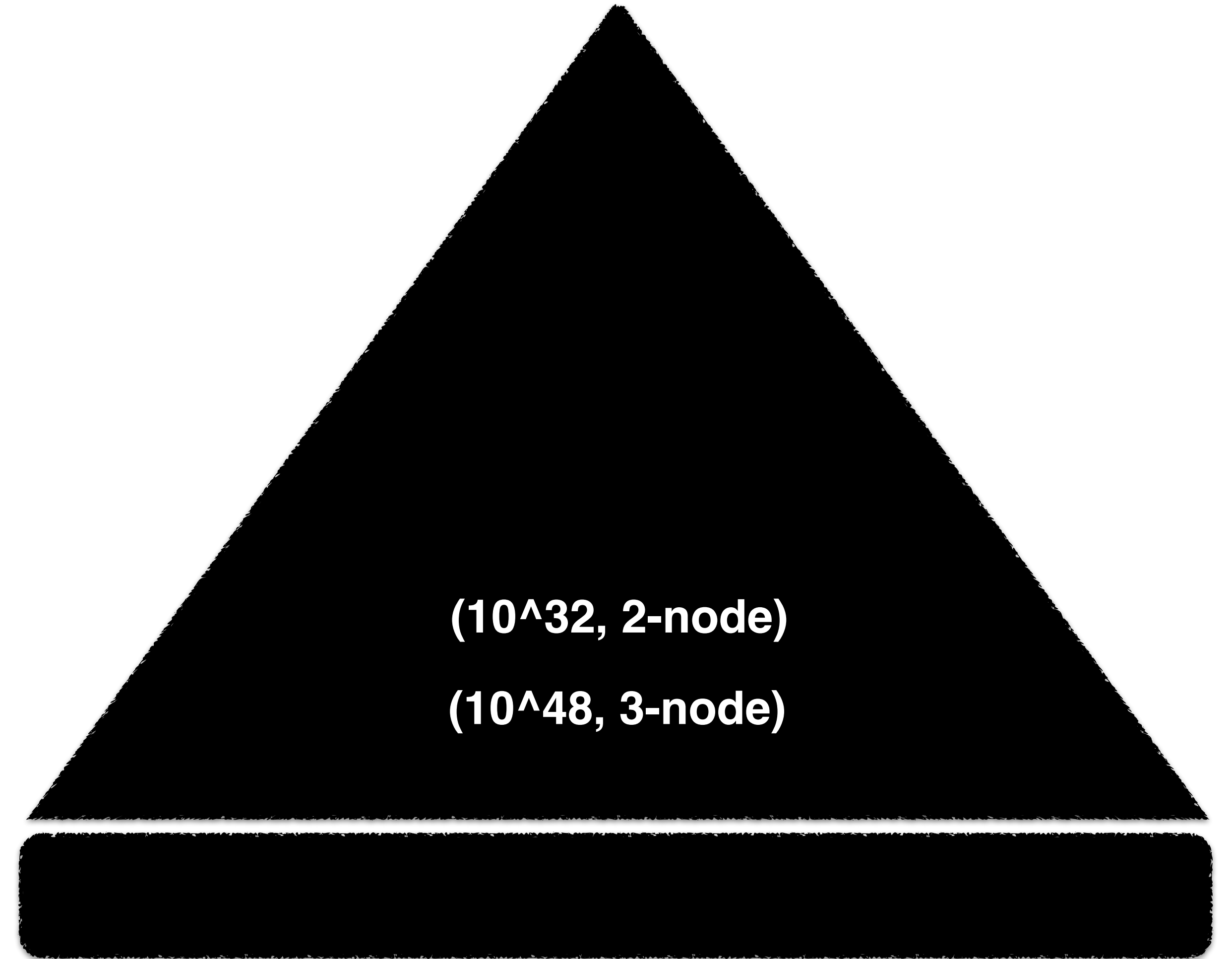


WE CAN AUTOMATICALLY DESIGN/DEBUG/FILL IN

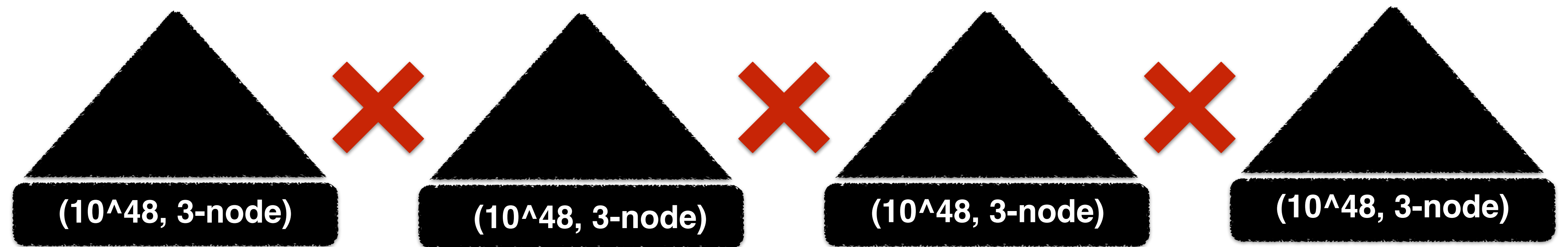




STARS ON THE SKY

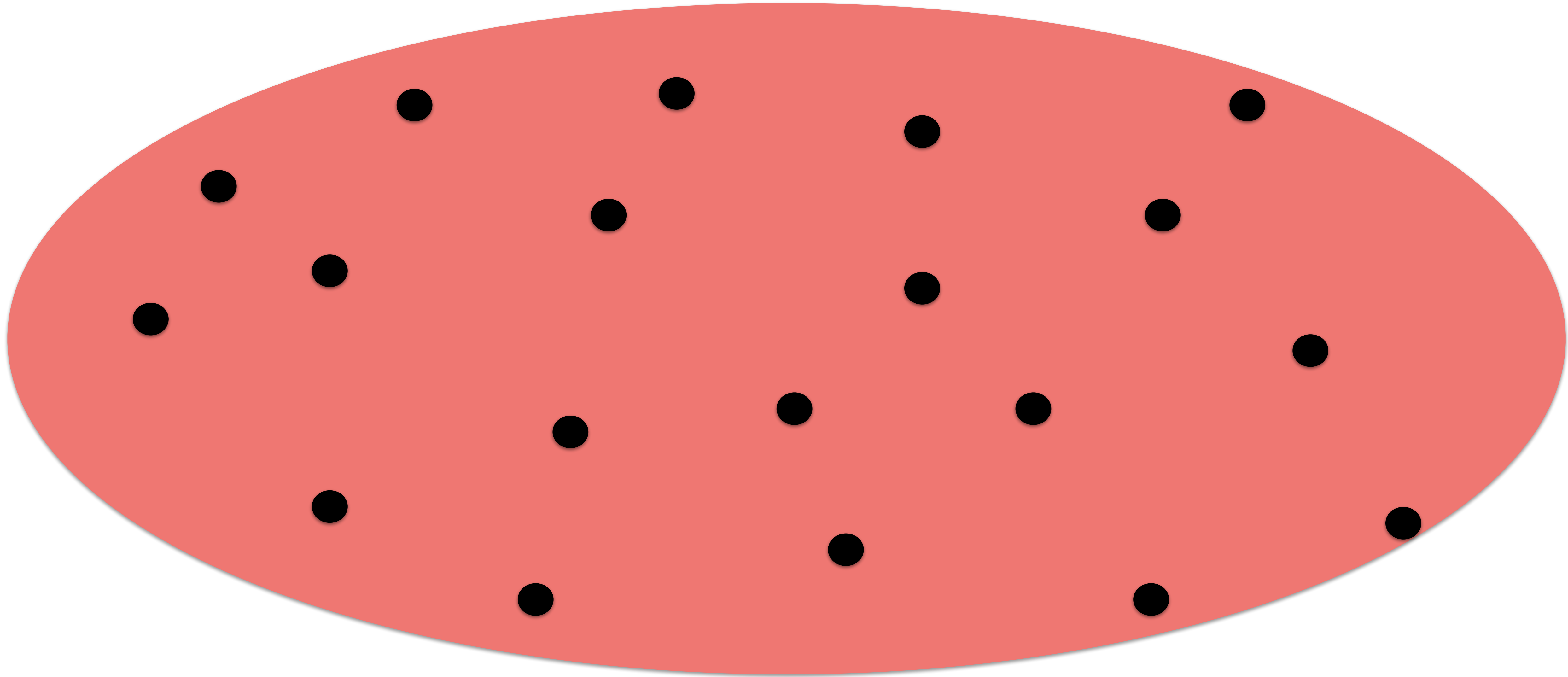


POSSIBLE DATA STRUCTURES

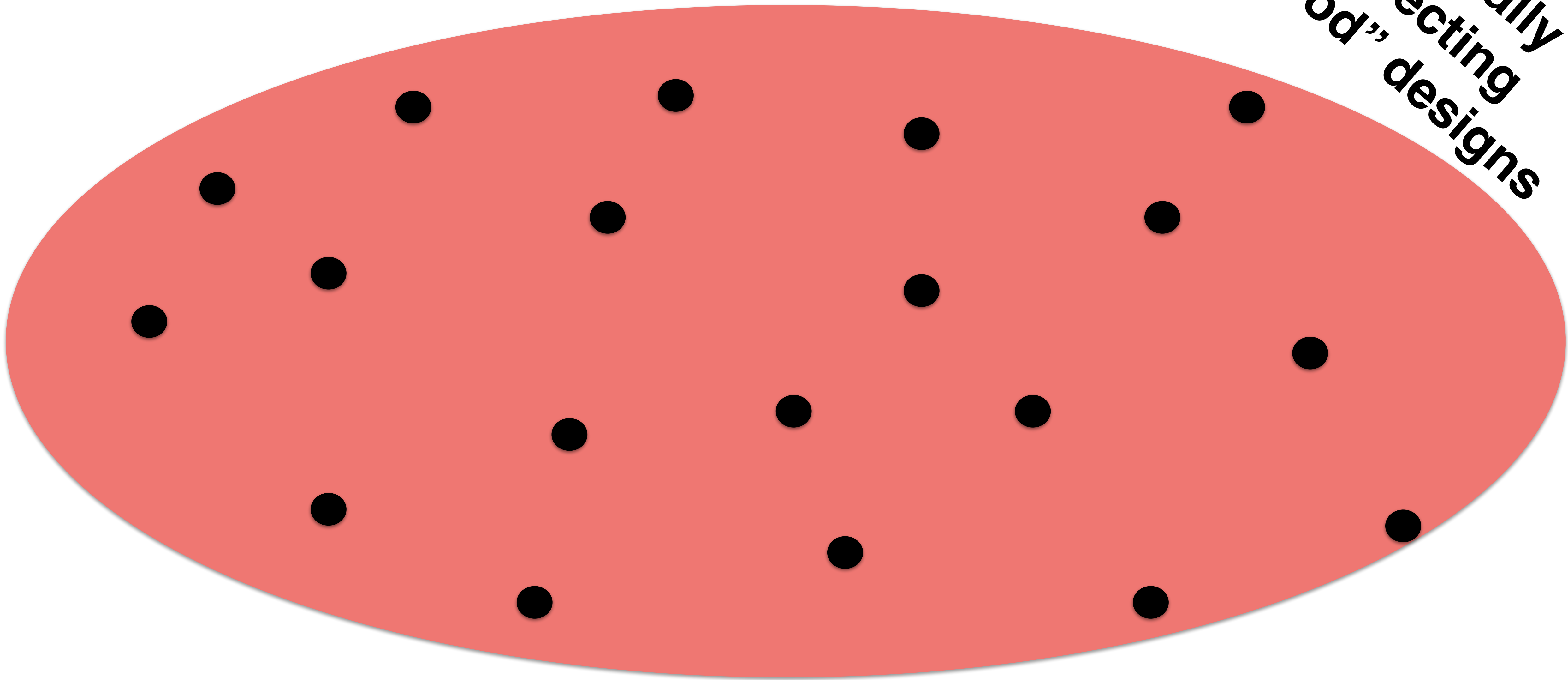


The design space of systems is even larger

The design space of systems is even larger



manually
selecting
“good” designs



manually
selecting
“good” designs

B-tree based
KV-system

LSM-tree based
KV-system

LSH based
KV-system

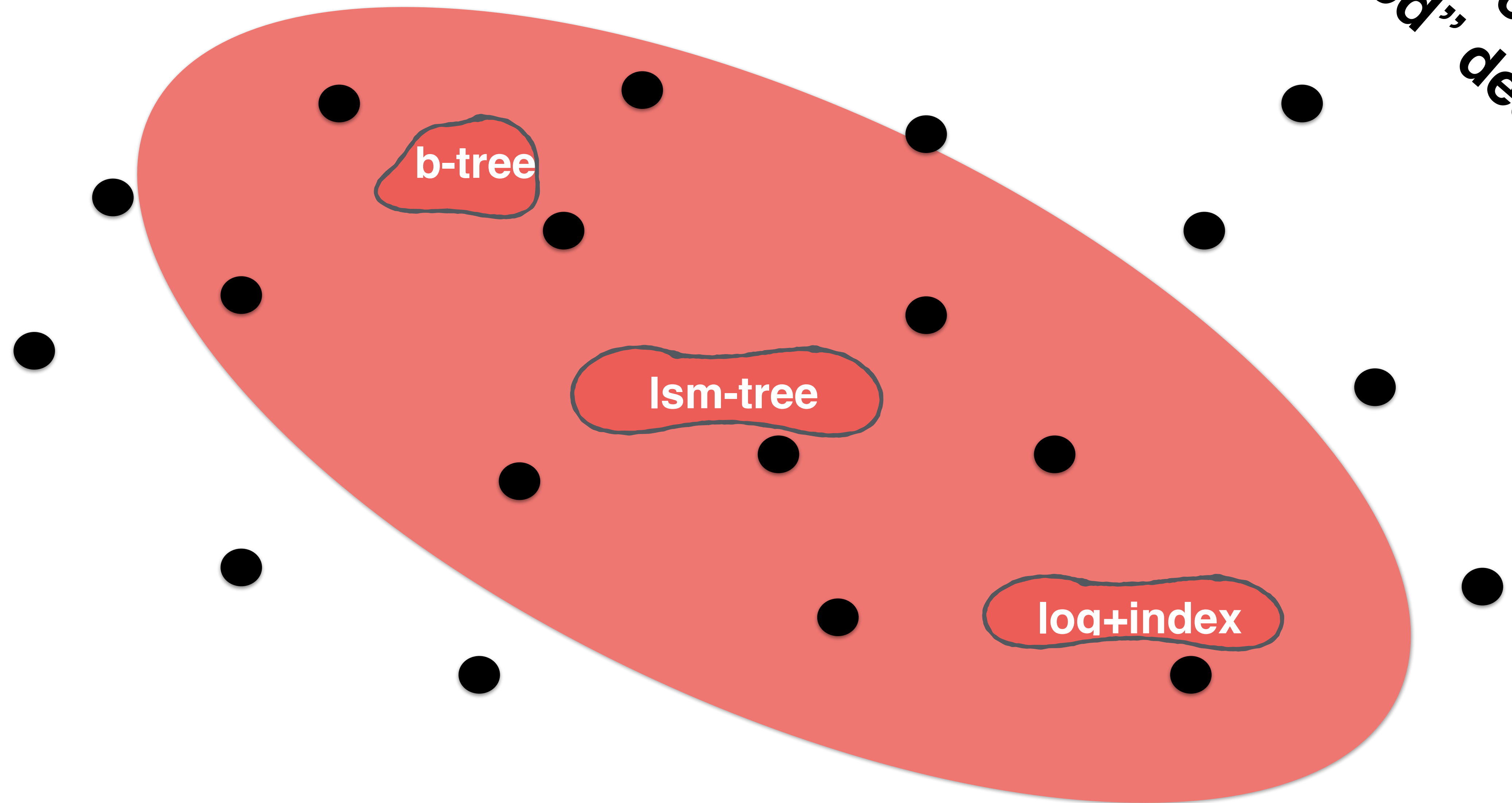
manually
selecting
“good” designs

b-tree

lsm-tree

log+index

manually
selecting
“good” designs



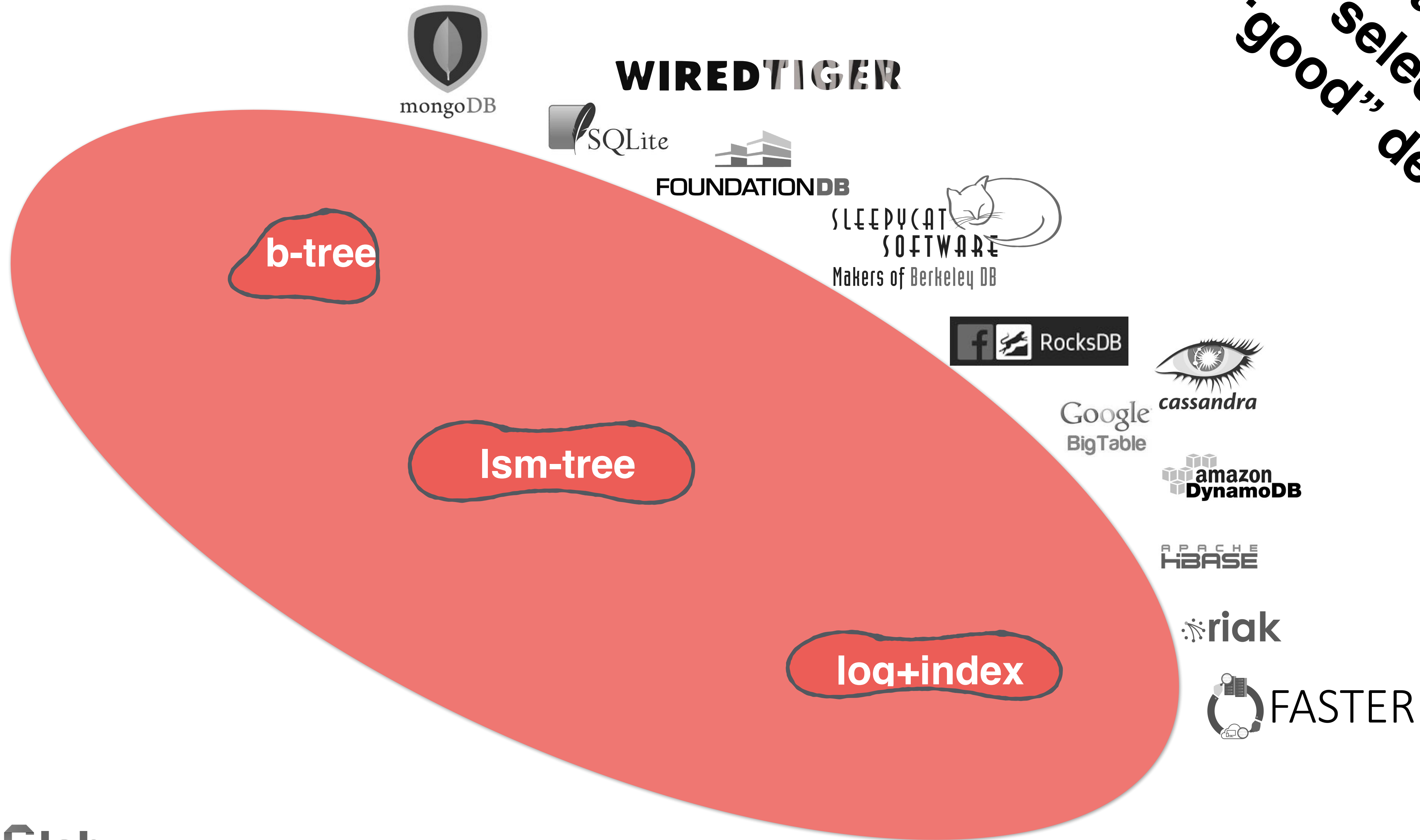
manually
selecting
“good” designs

b-tree

lsm-tree

log+index

manually
selecting
“good” designs



manually
selecting
“good” designs

WIREDTIGER



FOUNDATIONDB



Google
BigTable



cassandra



APACHE
HBASE

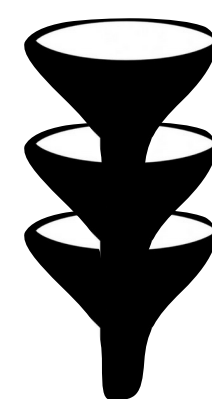
riak



b-tree

lsm-tree

log+index



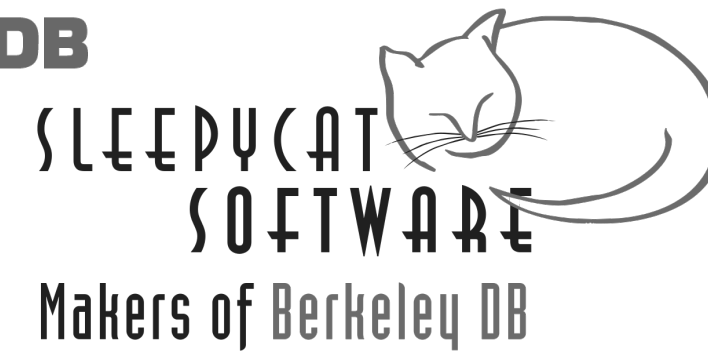
manually
selecting
“good” designs



WIREDTIGER



FOUNDATIONDB



Google
BigTable

cassandra



APACHE
HBASE

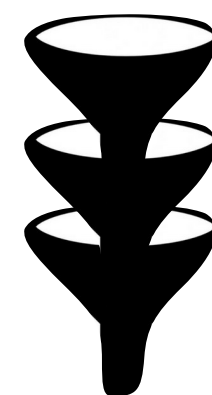
riak

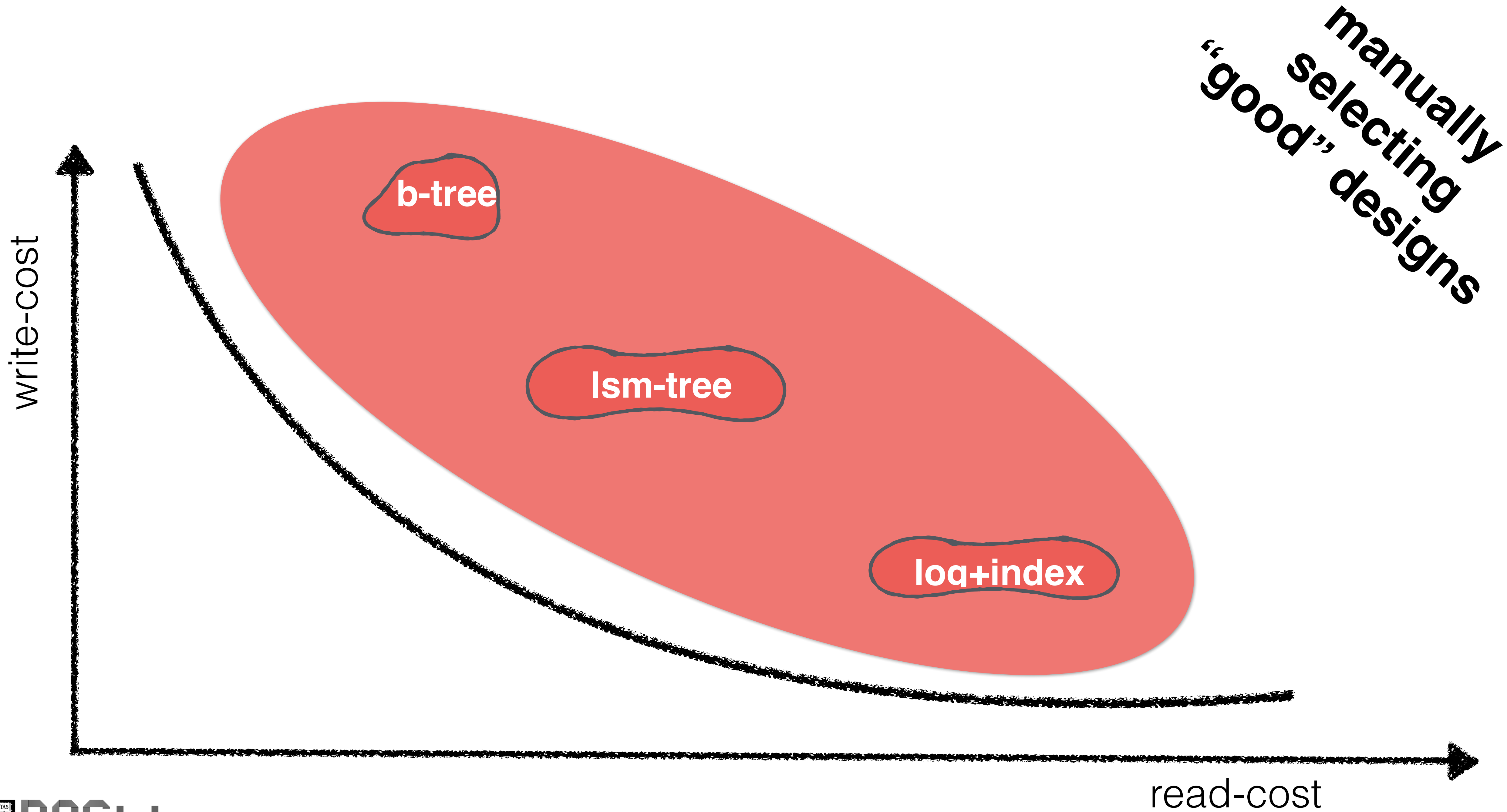


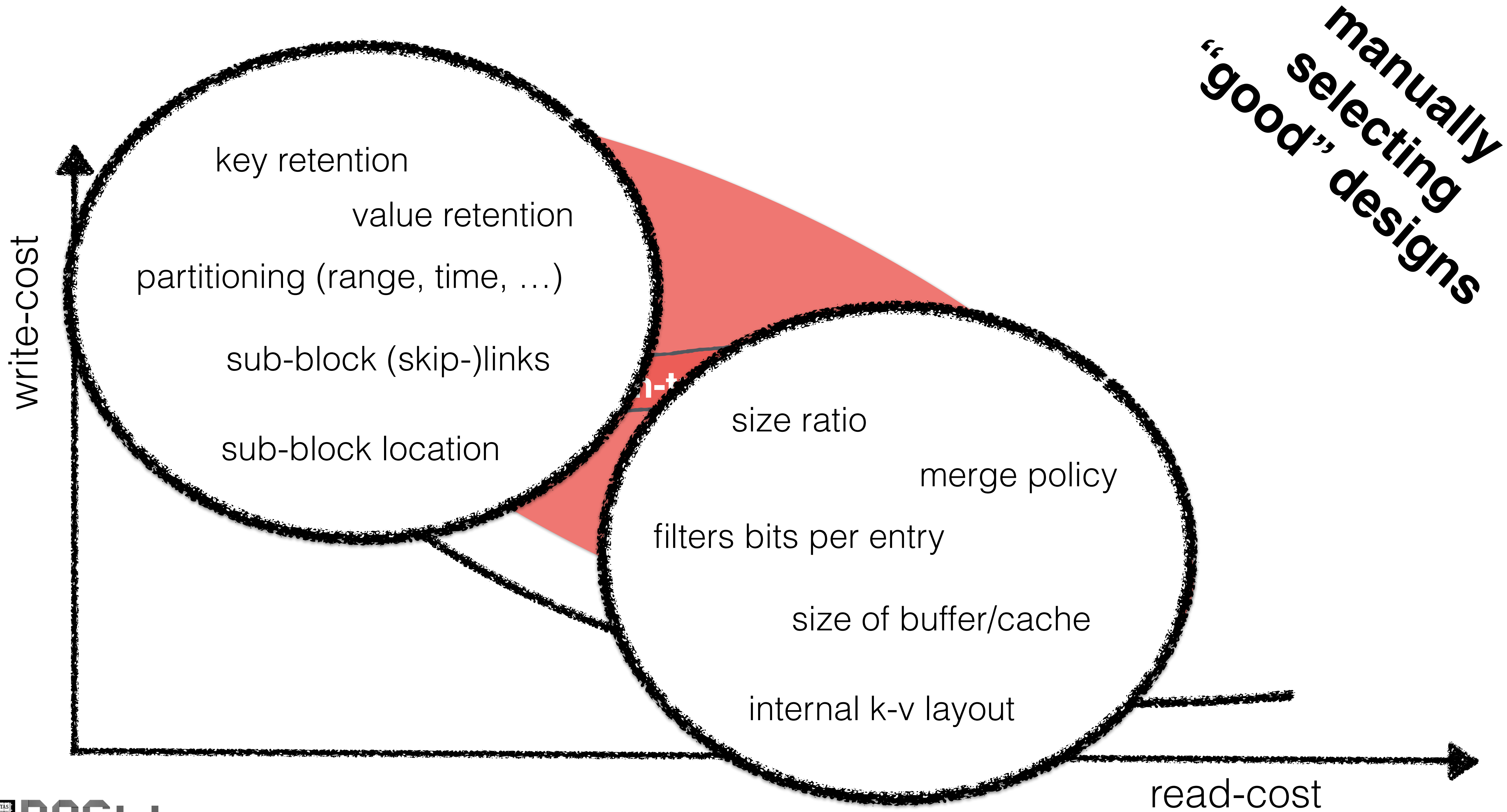
b-tree

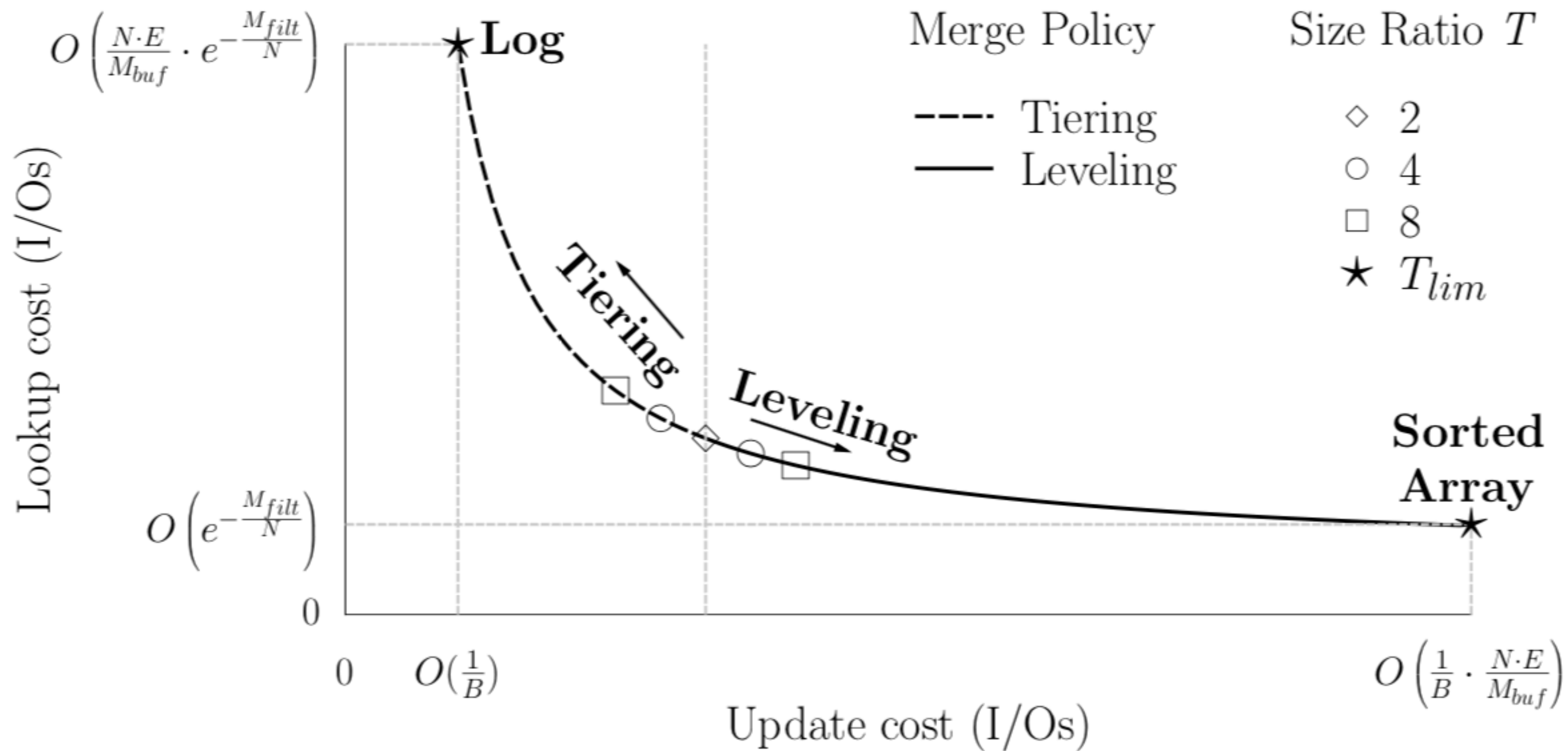
lsm-tree

log+index



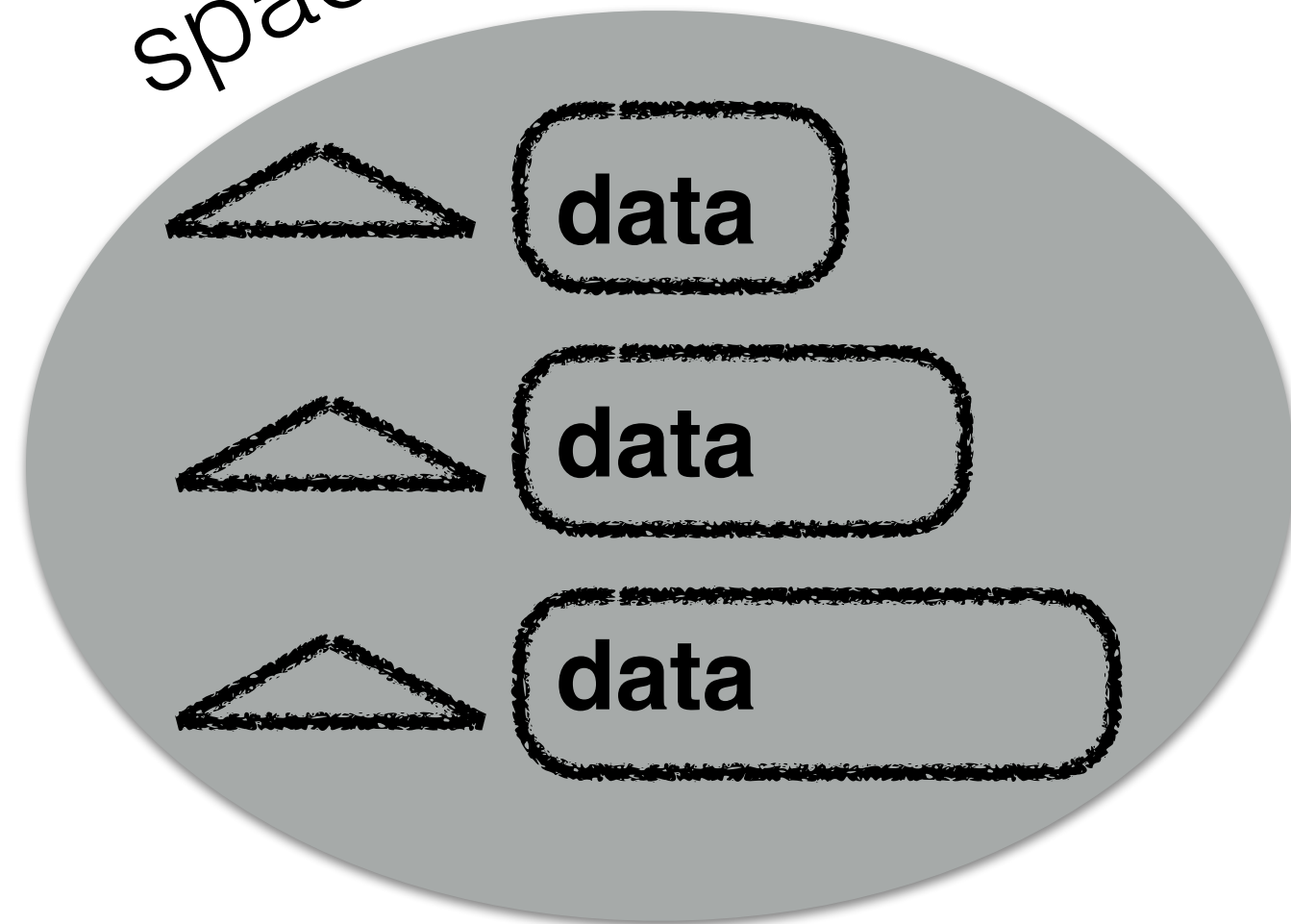






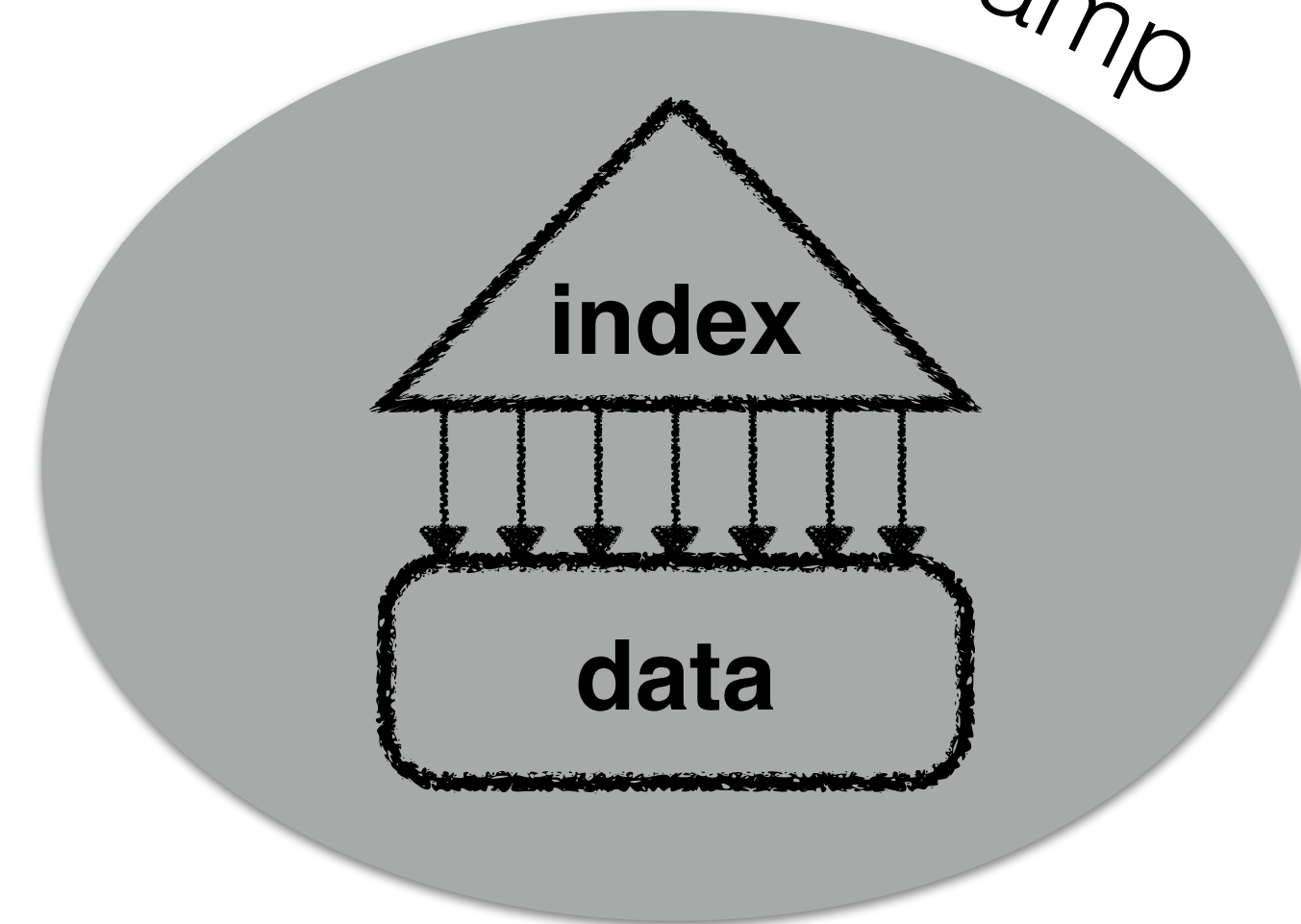
discrete

great writes
decent reads
space amp



LSM-tree

good reads
ok updates
space amp



B-tree

discrete

 **riak**

APACHE
HBASE



mongoDB

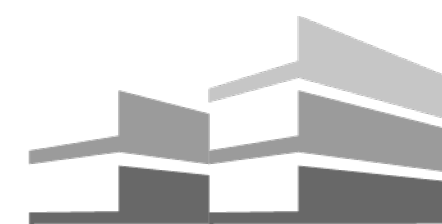
Google
BigTable



SQLite



cassandra



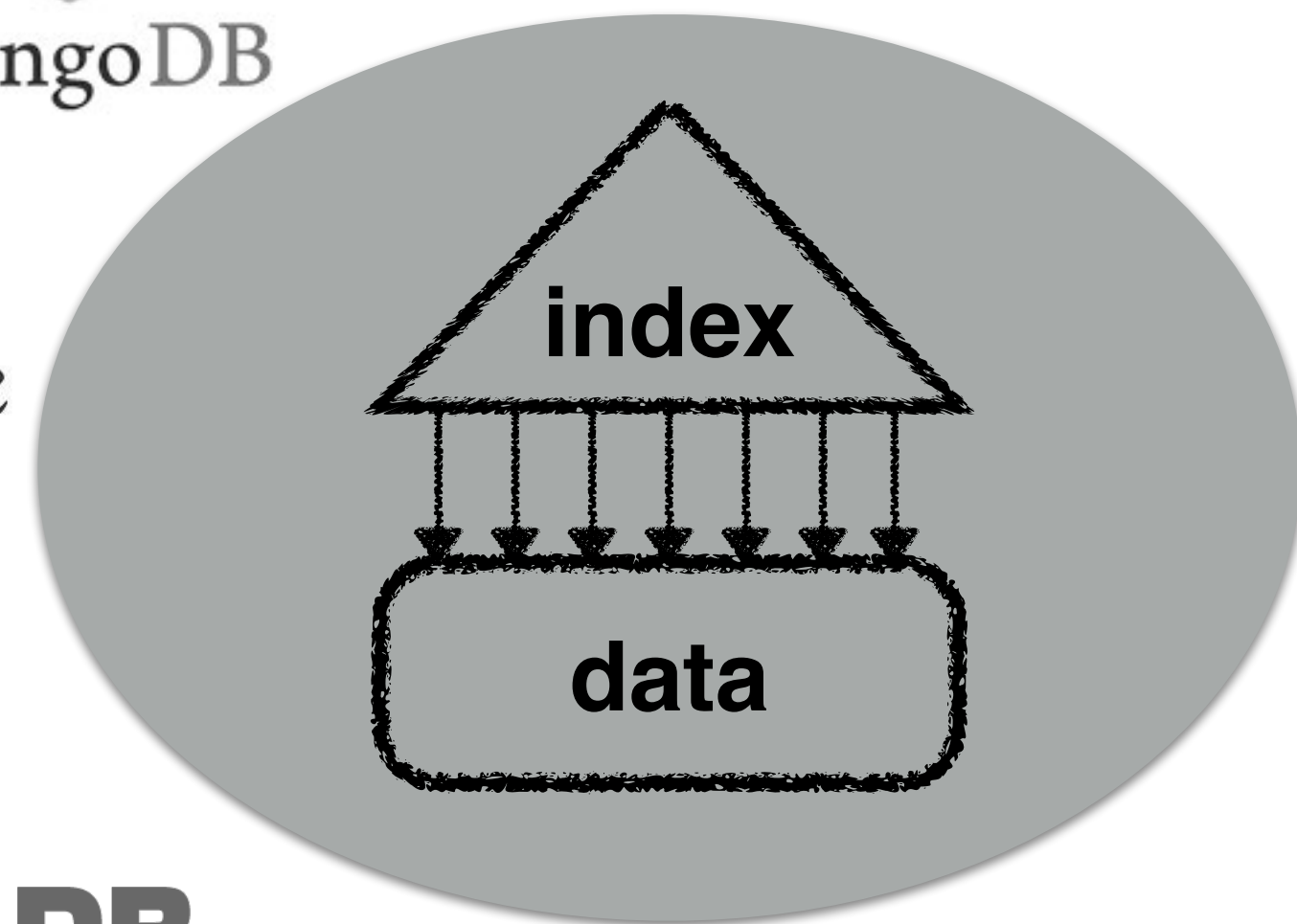
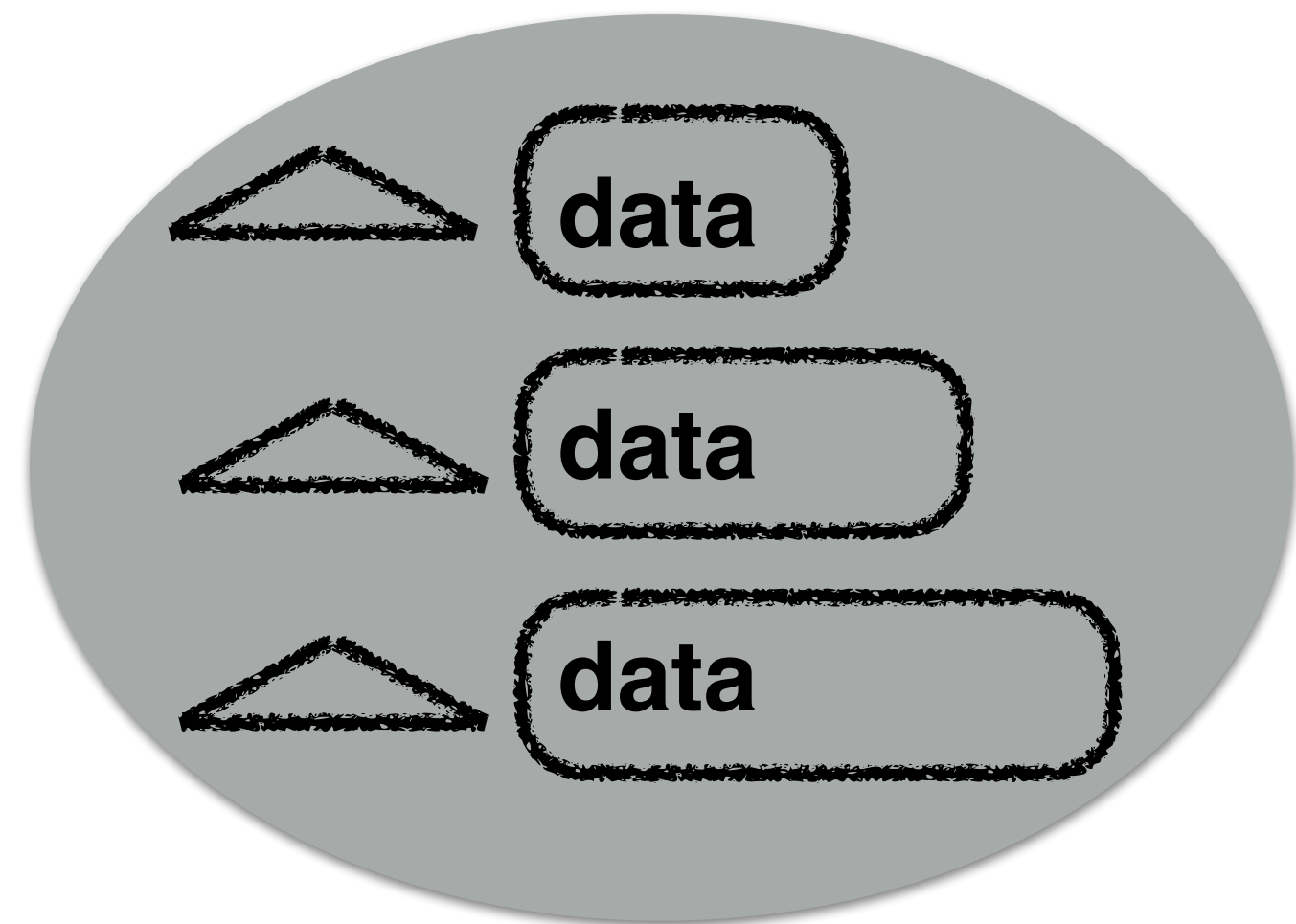
FOUNDATIONDB

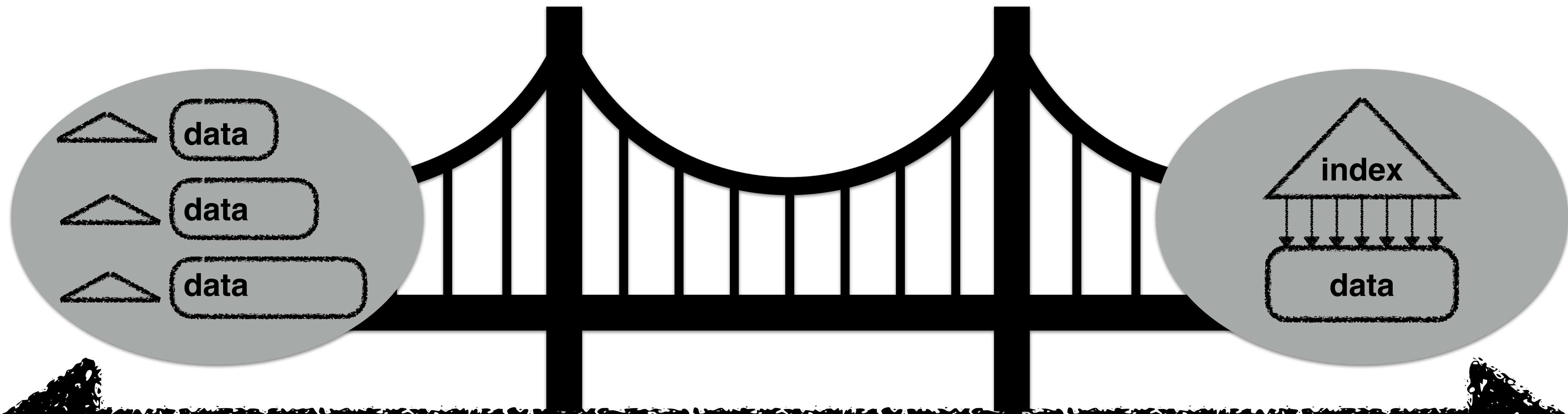
 **amazon
DynamoDB**

WIREDTIGER

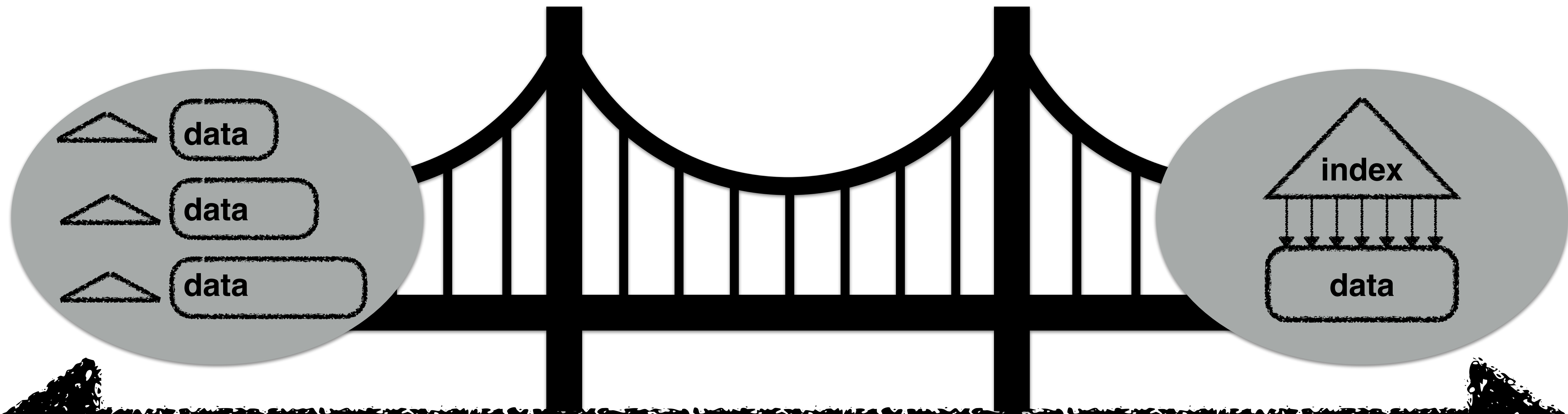
  **RocksDB**

SLEEPYCAT
SOFTWARE
Makers of Berkeley DB



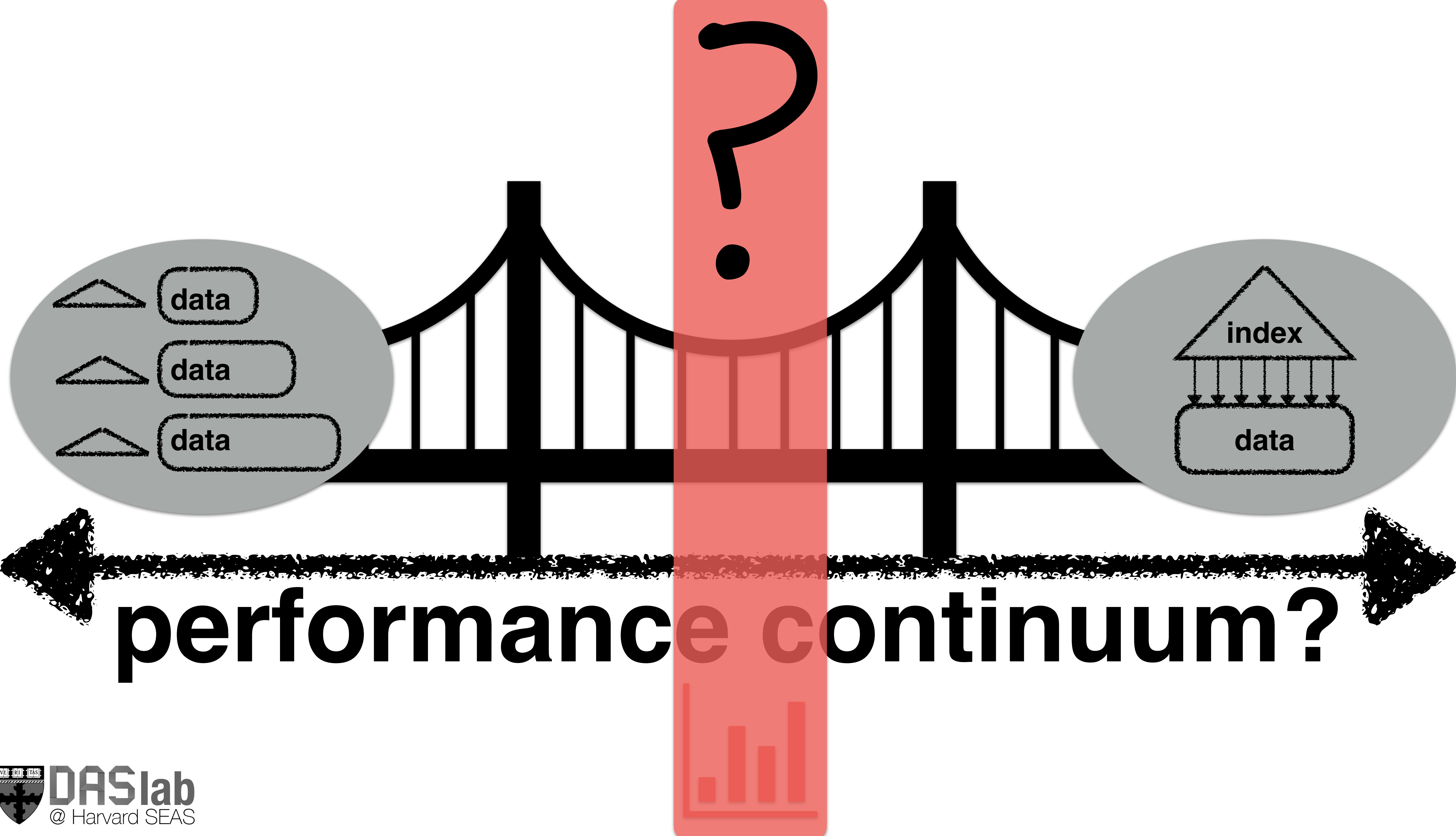


performance continuum?

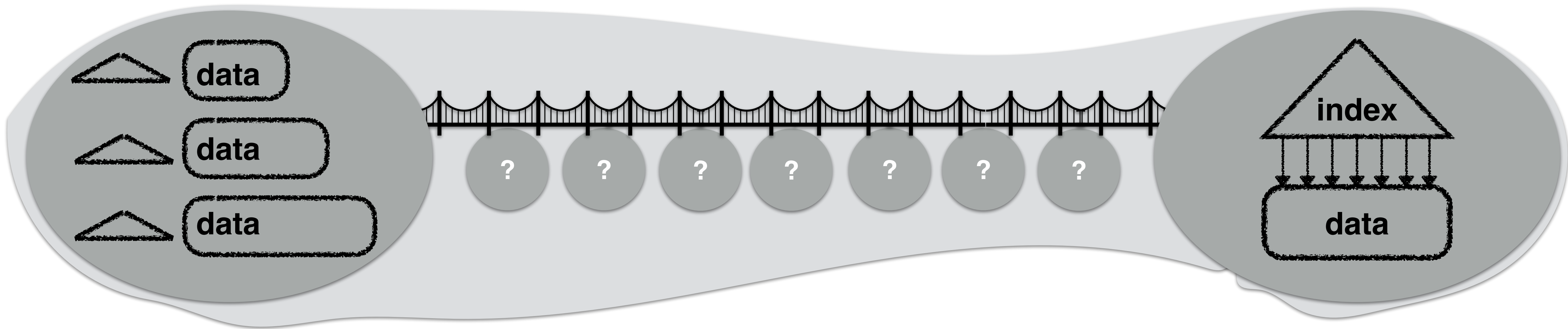


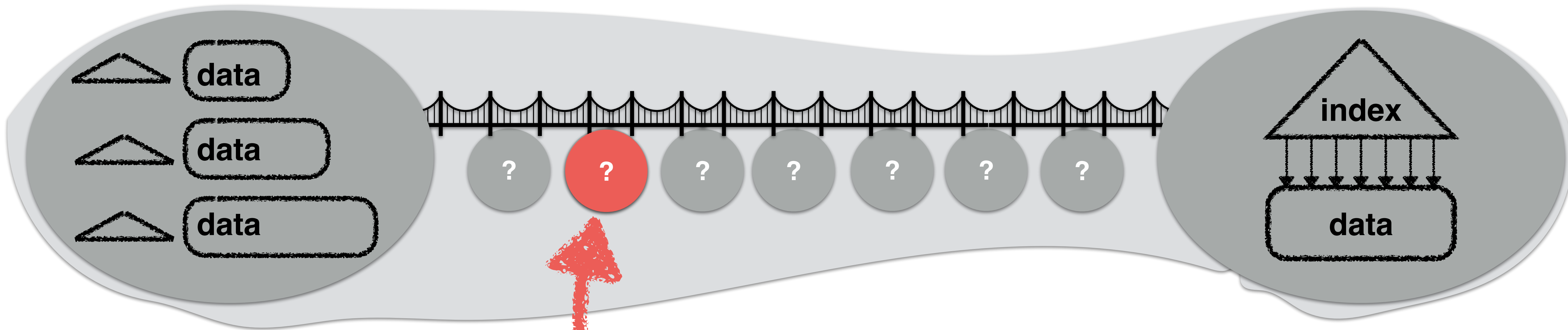
performance continuum?





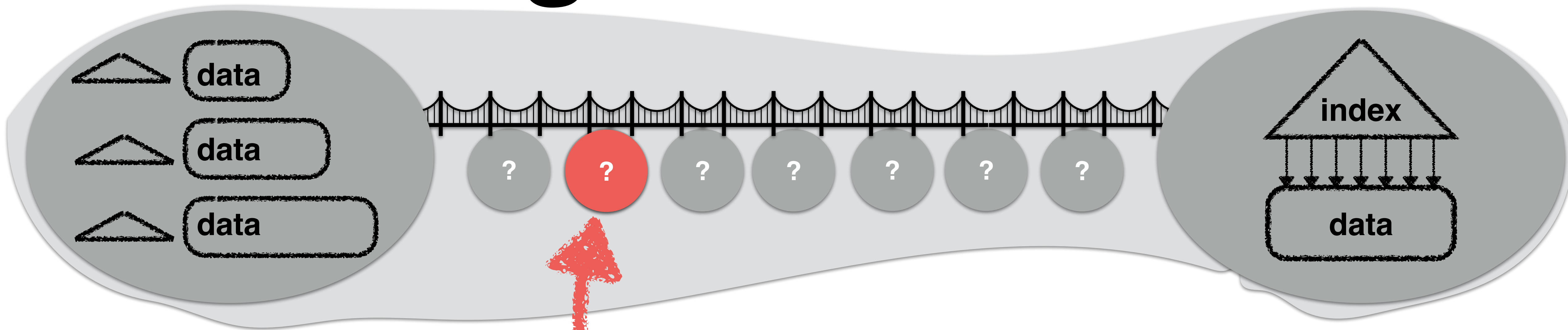
performance continuum?





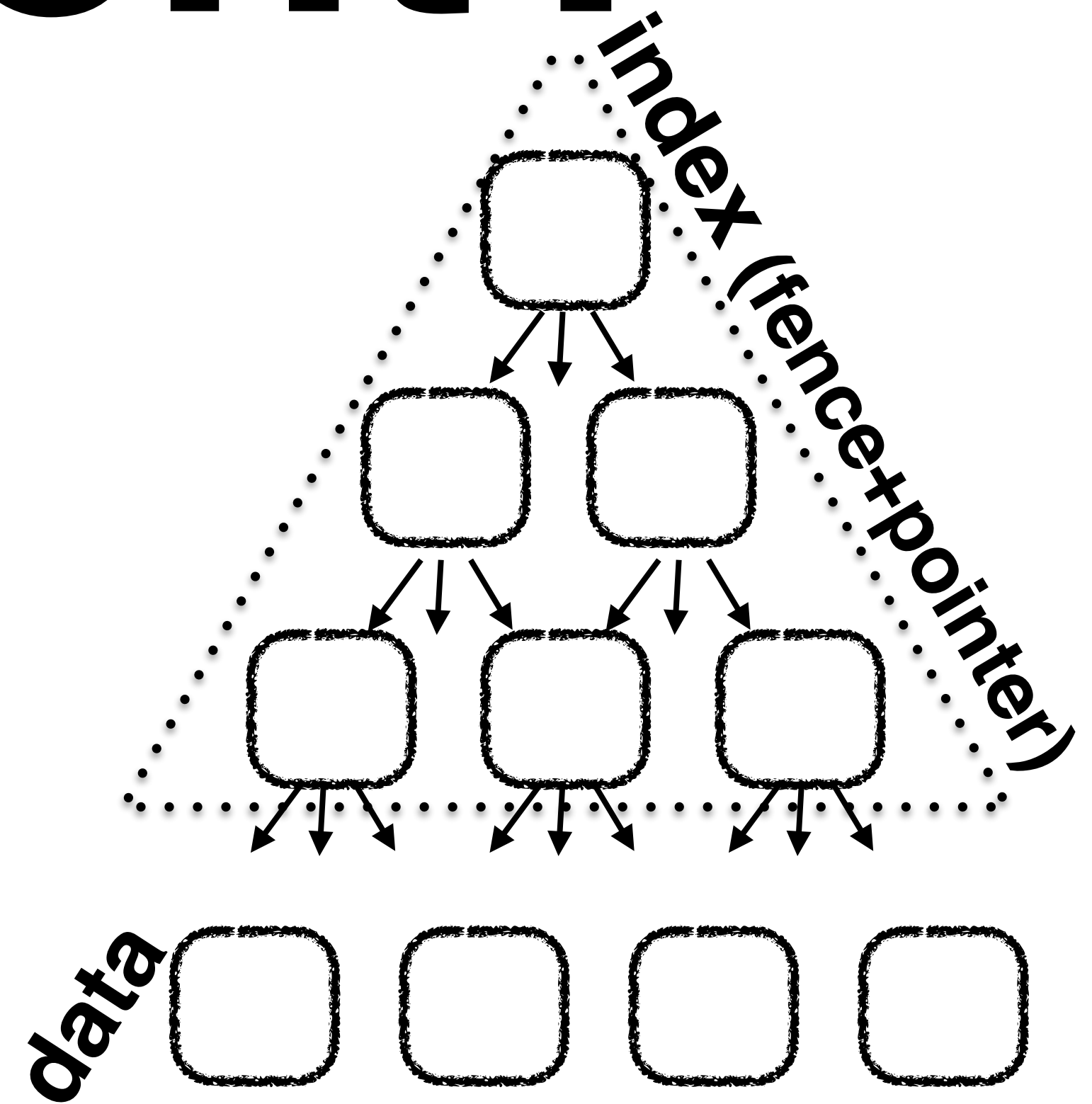
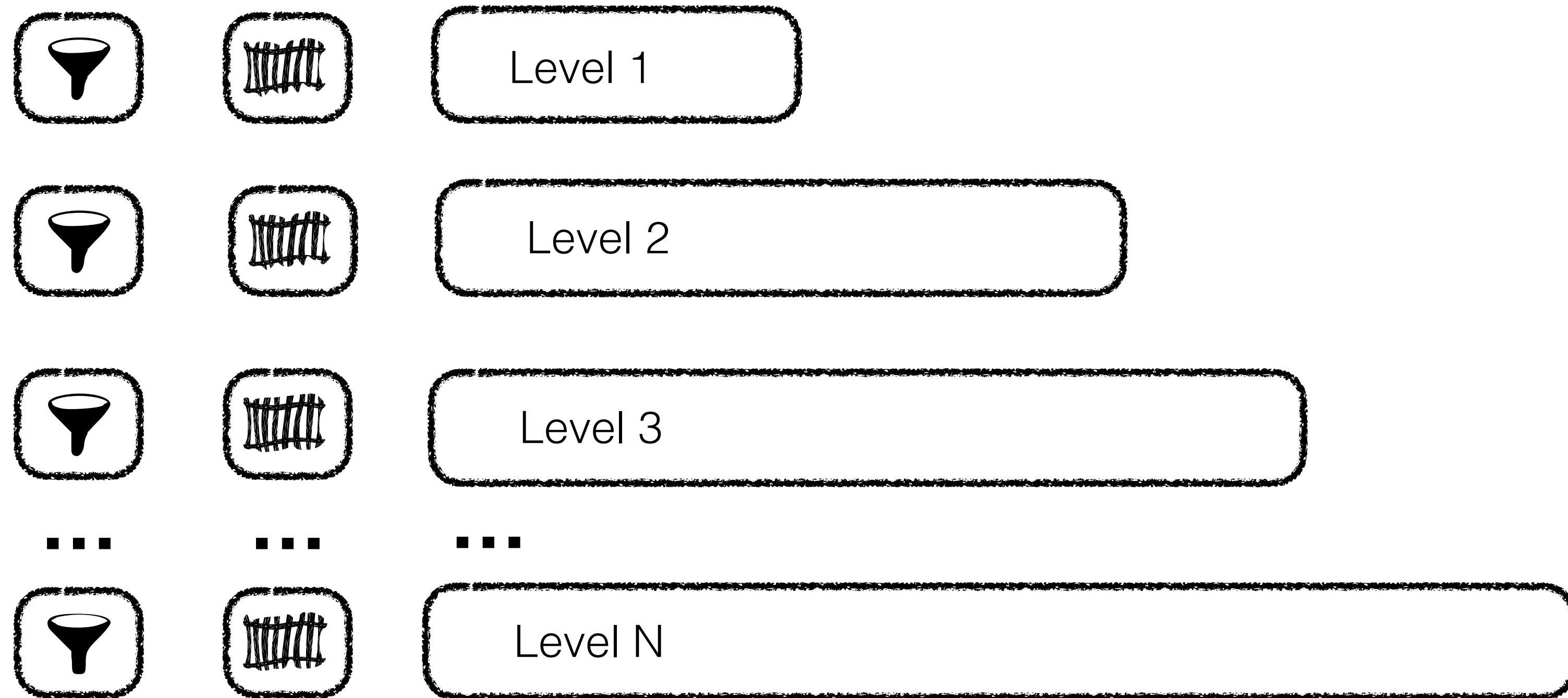
not LSM-tree, not B-tree
mixed design principles
hybrid performance properties

design continuum

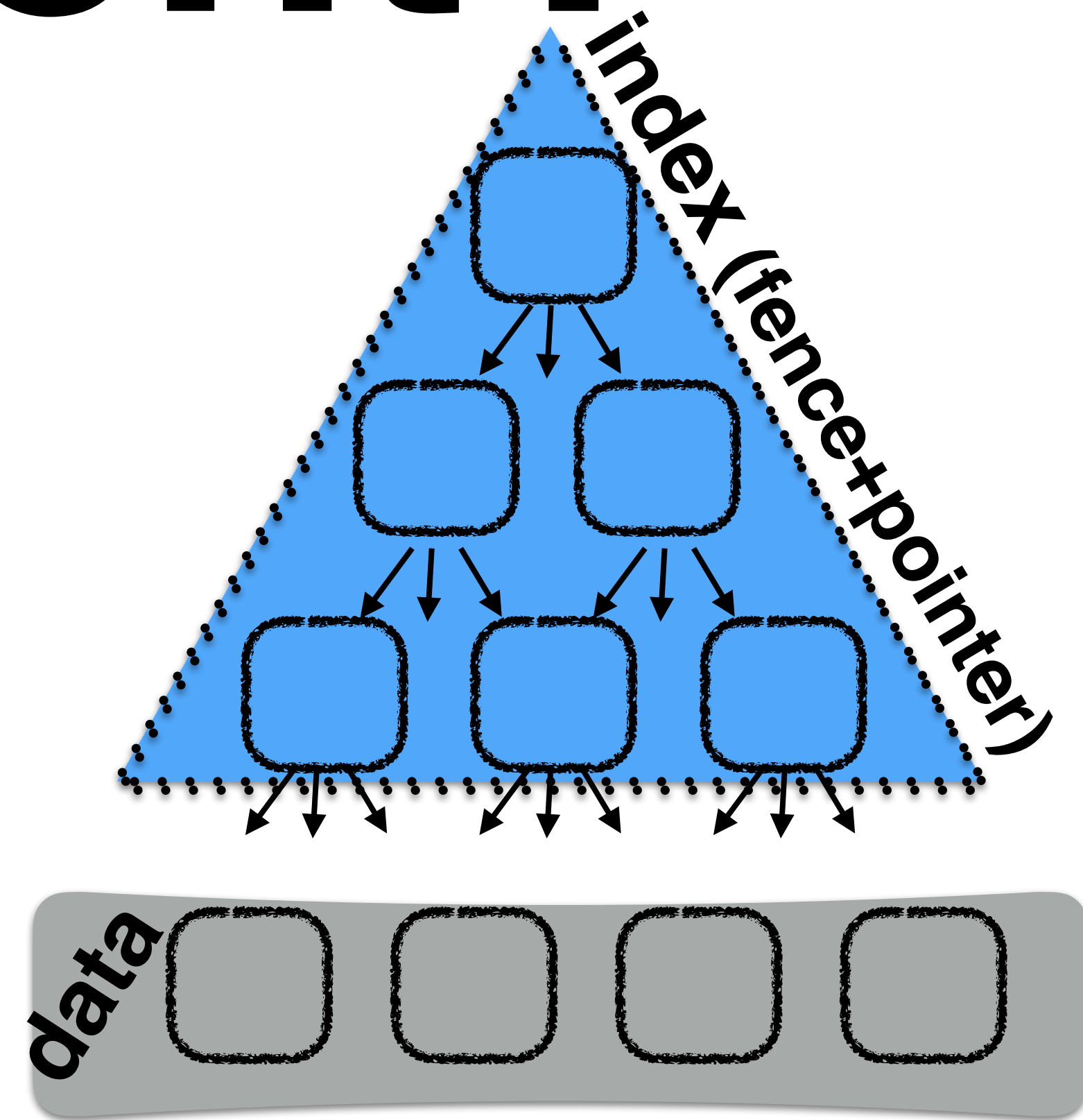
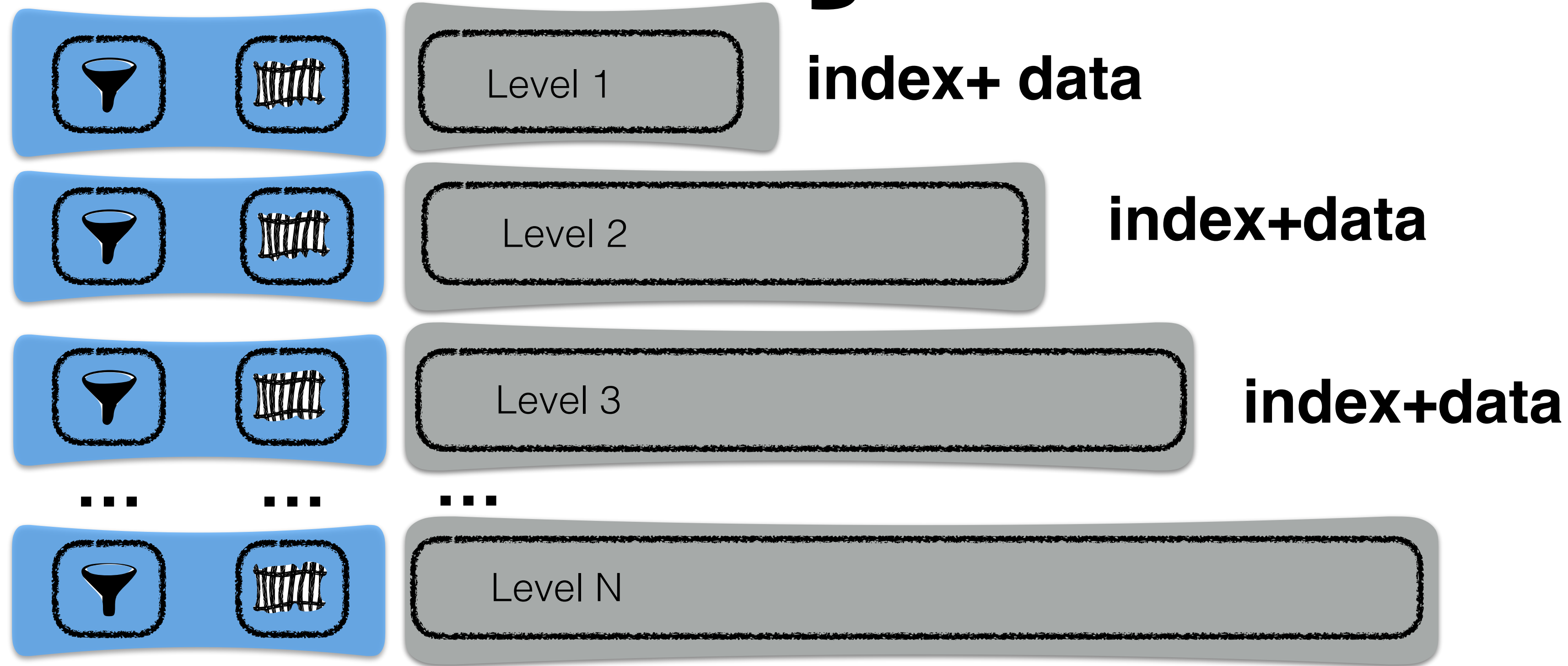


not LSM-tree, not B-tree
mixed design principles
hybrid performance properties

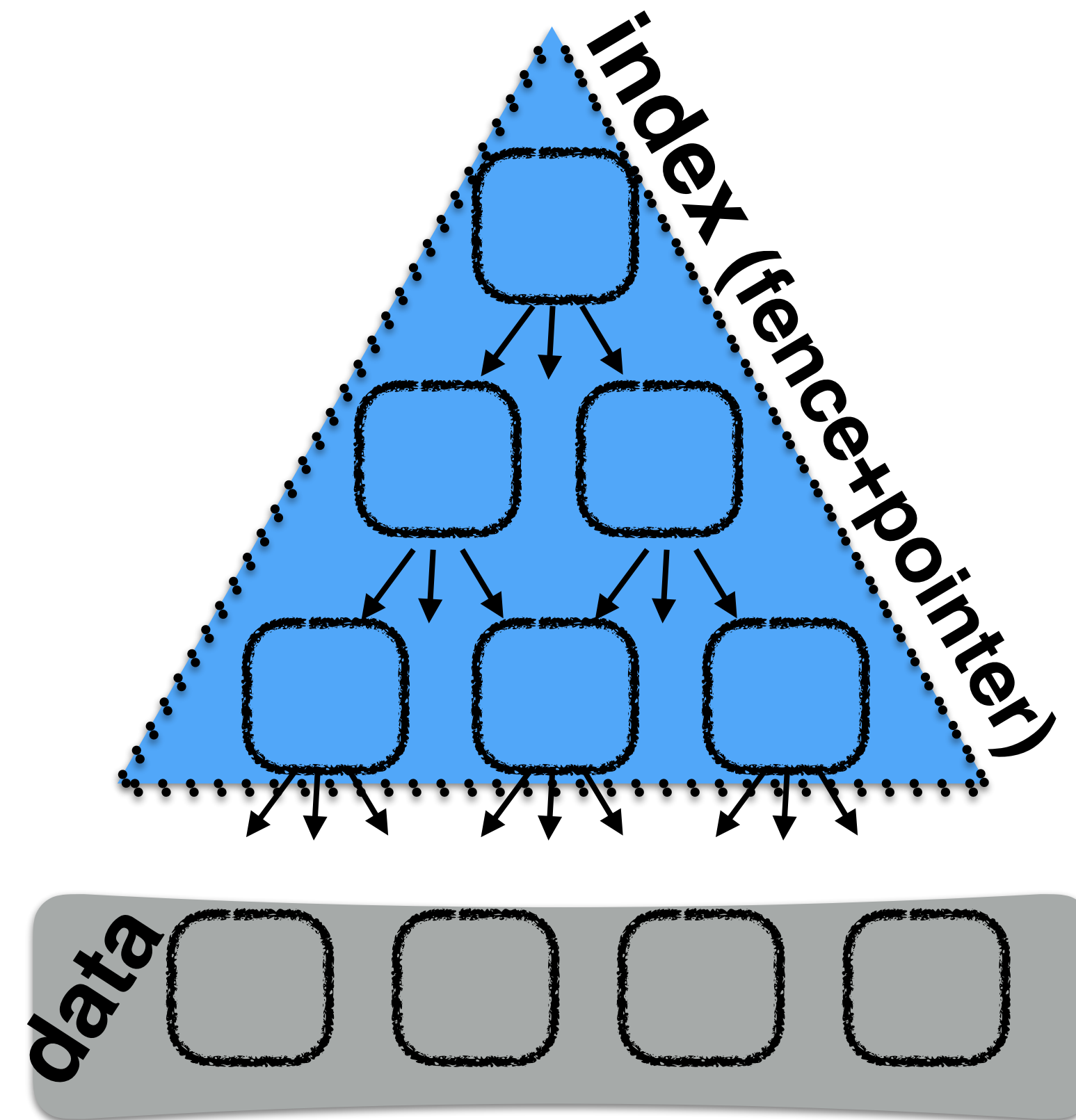
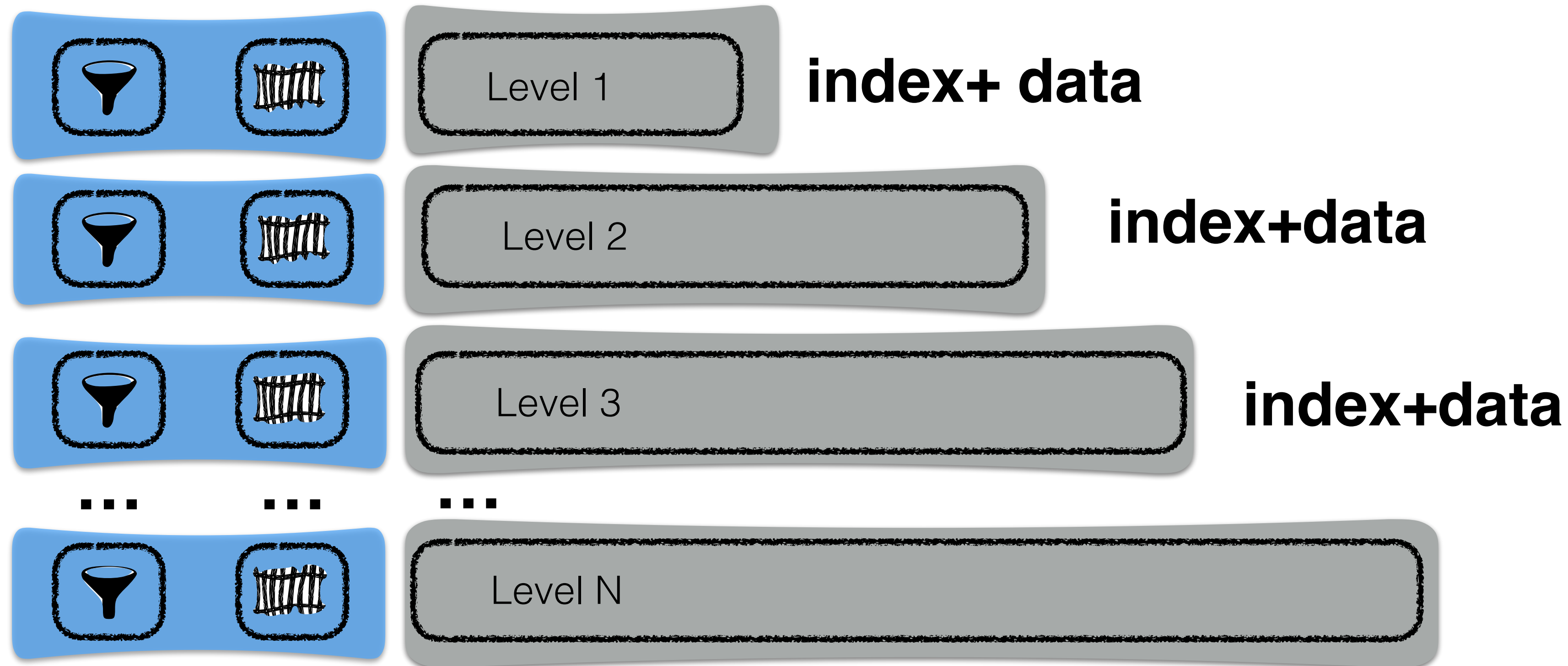
totally different?



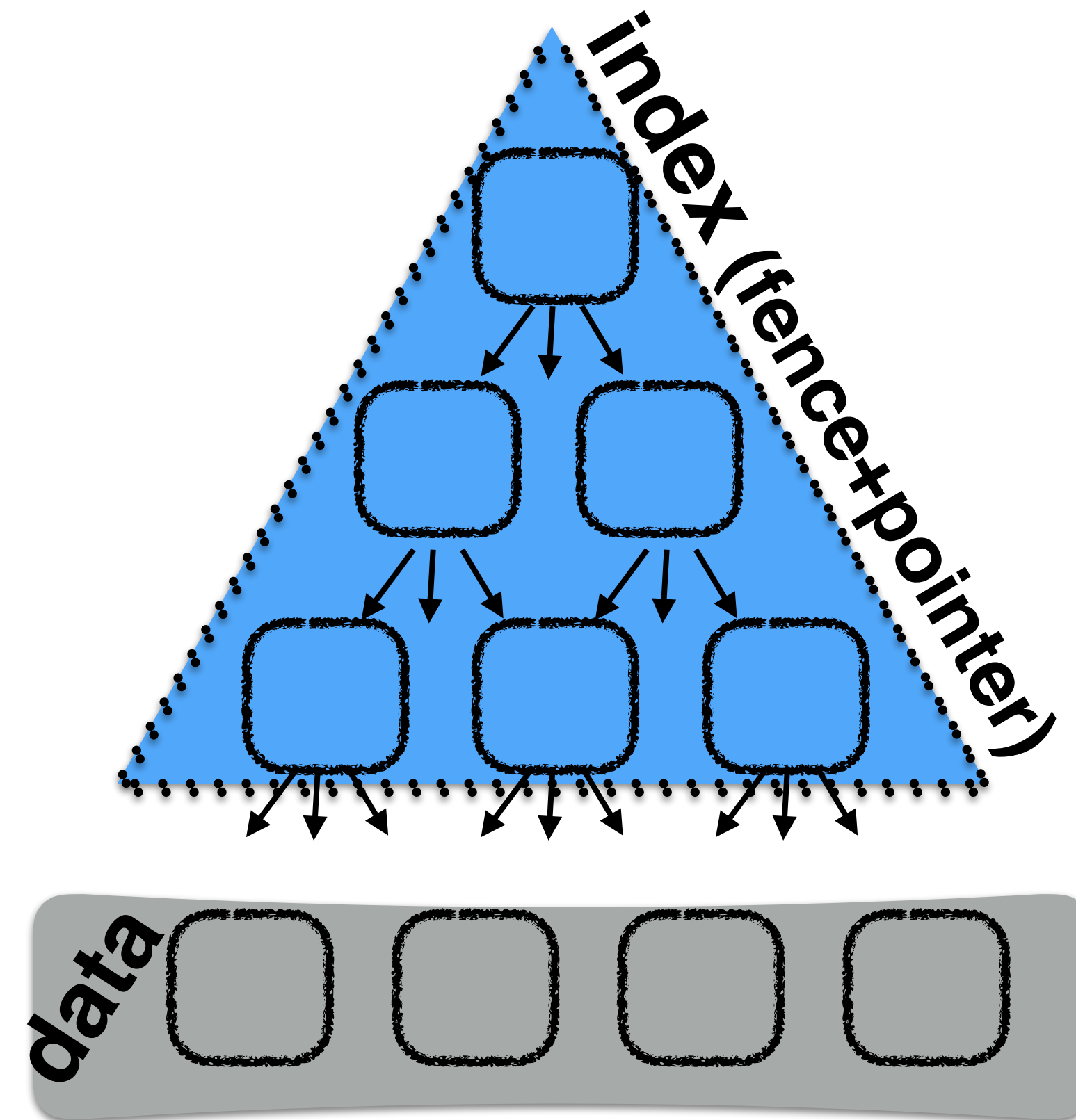
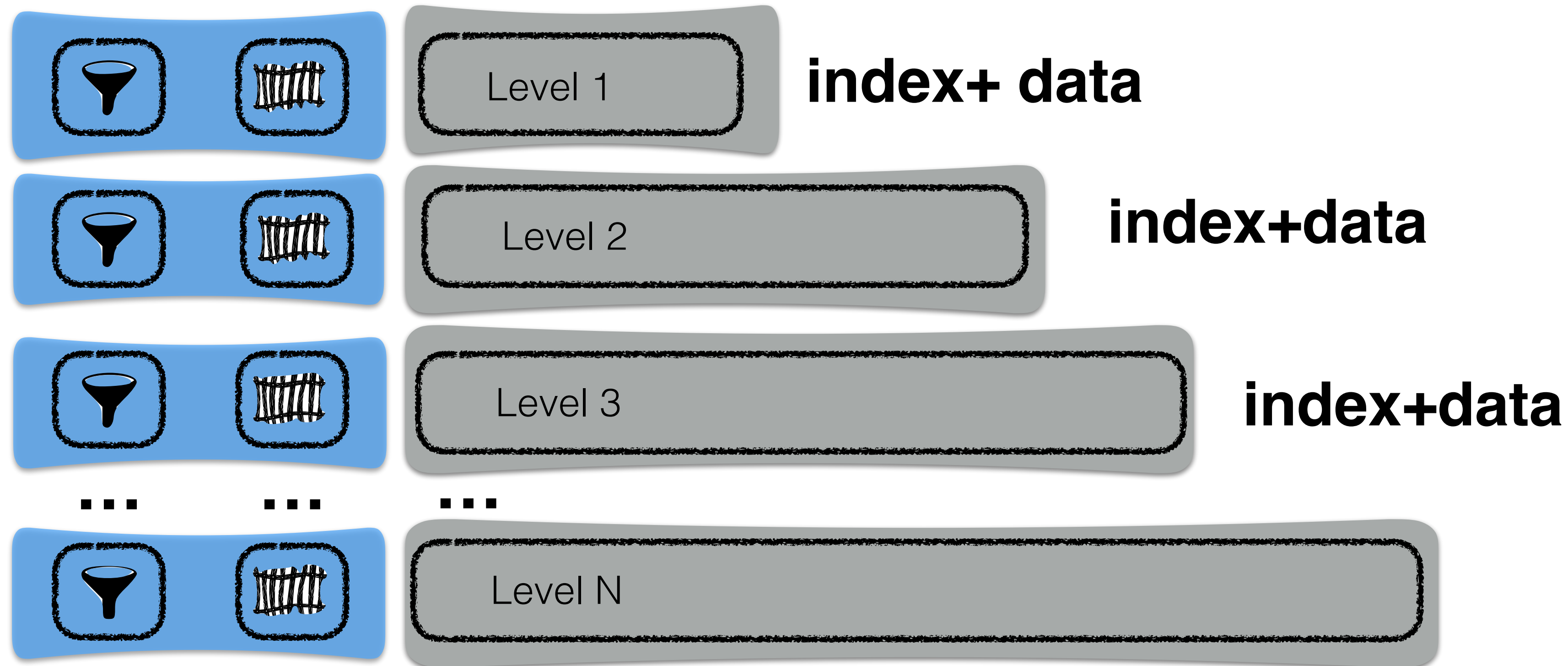
totally different?



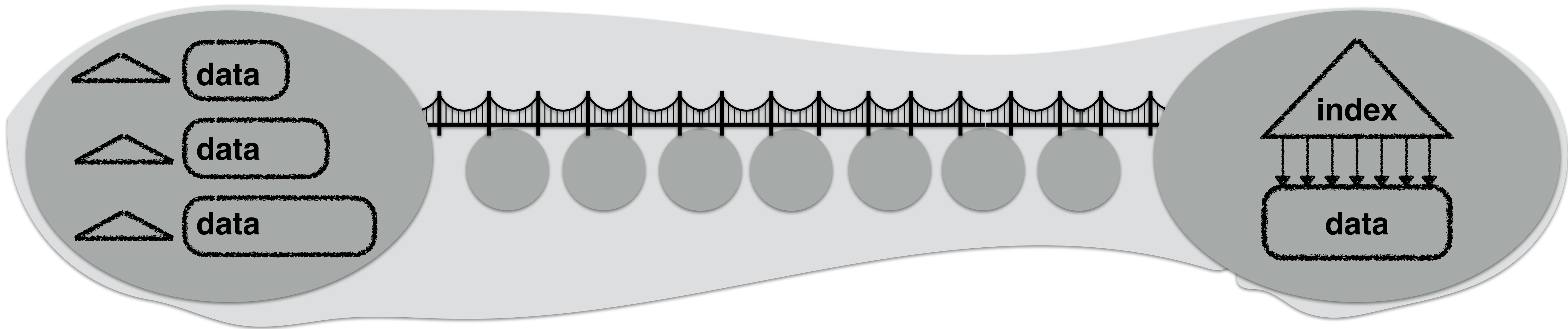
>1 B-trees that have not (yet) been merged



>1 B-trees that have not (yet) been merged

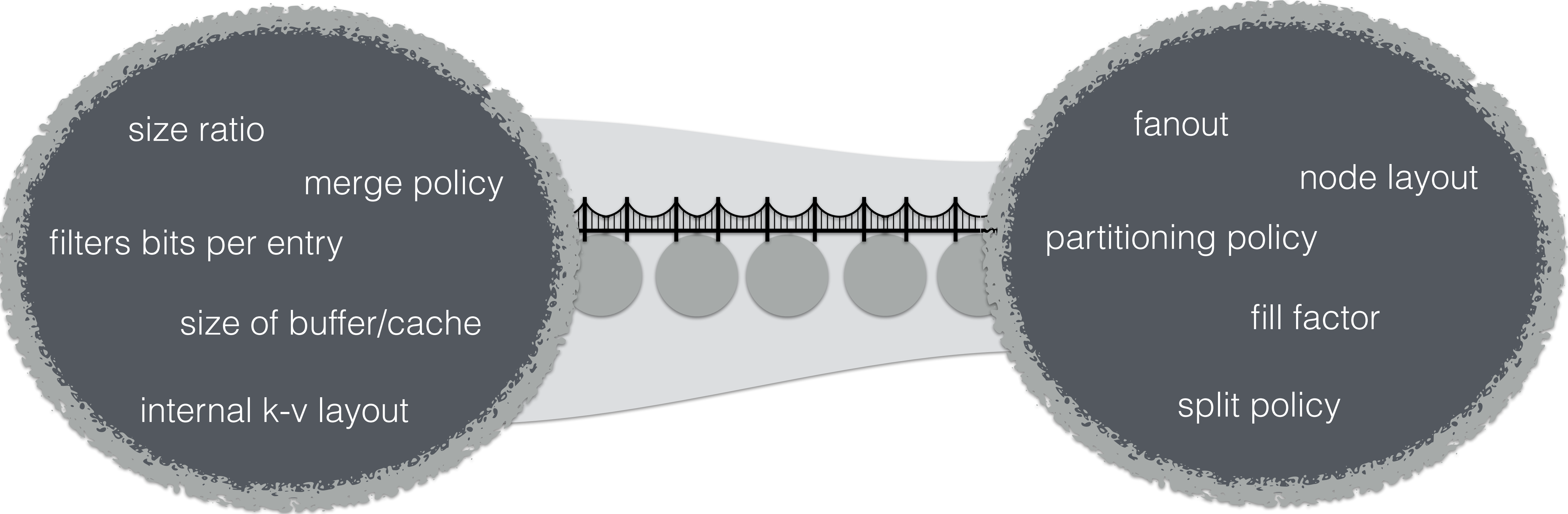


B-tree entries have not been propagated to the leaves



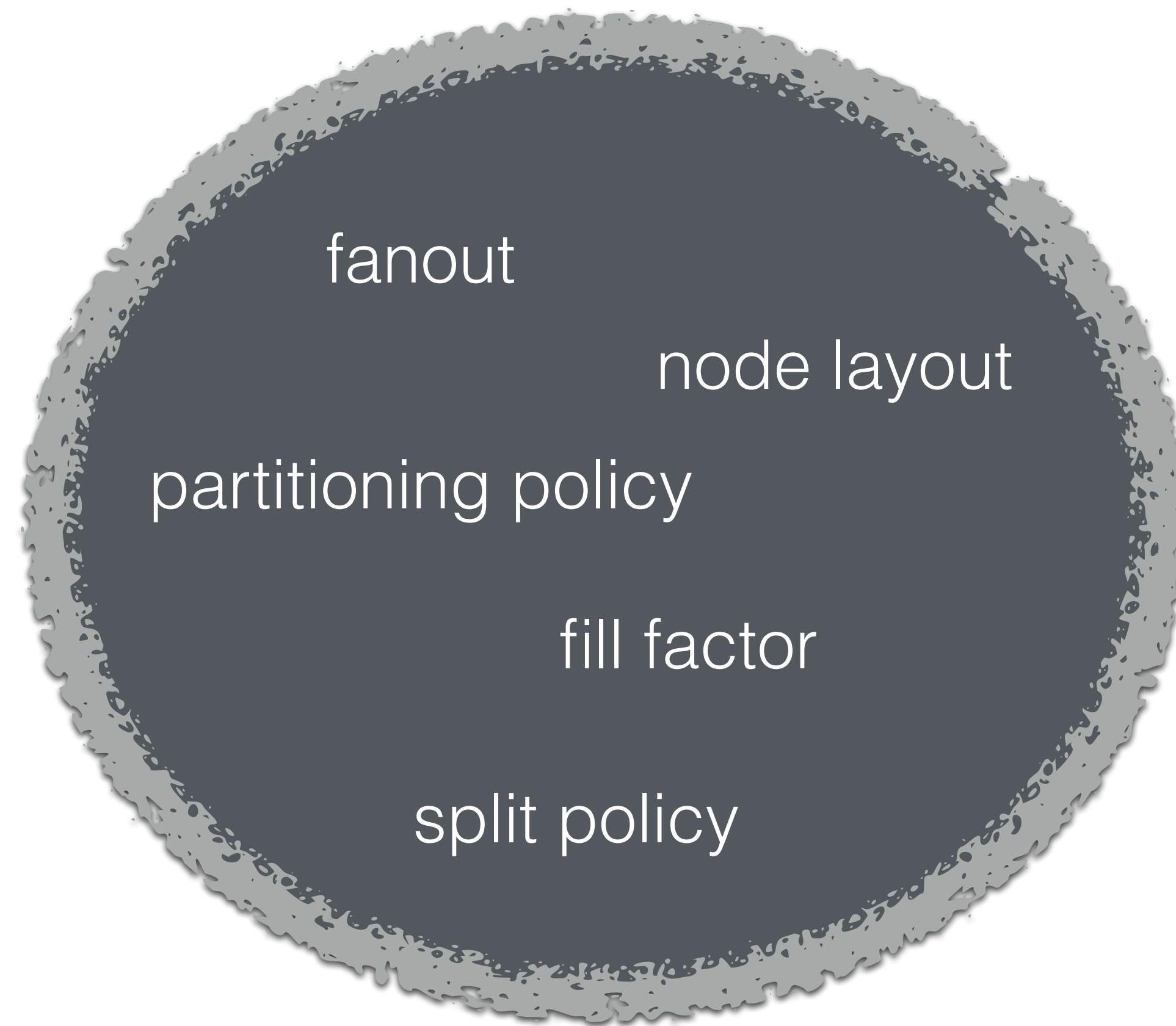
LSM-Trees

B-Trees



a unified design space

[log, log+hash, LSM-tree*, B^εTree, B-Tree, Sorted Array]



a unified design space

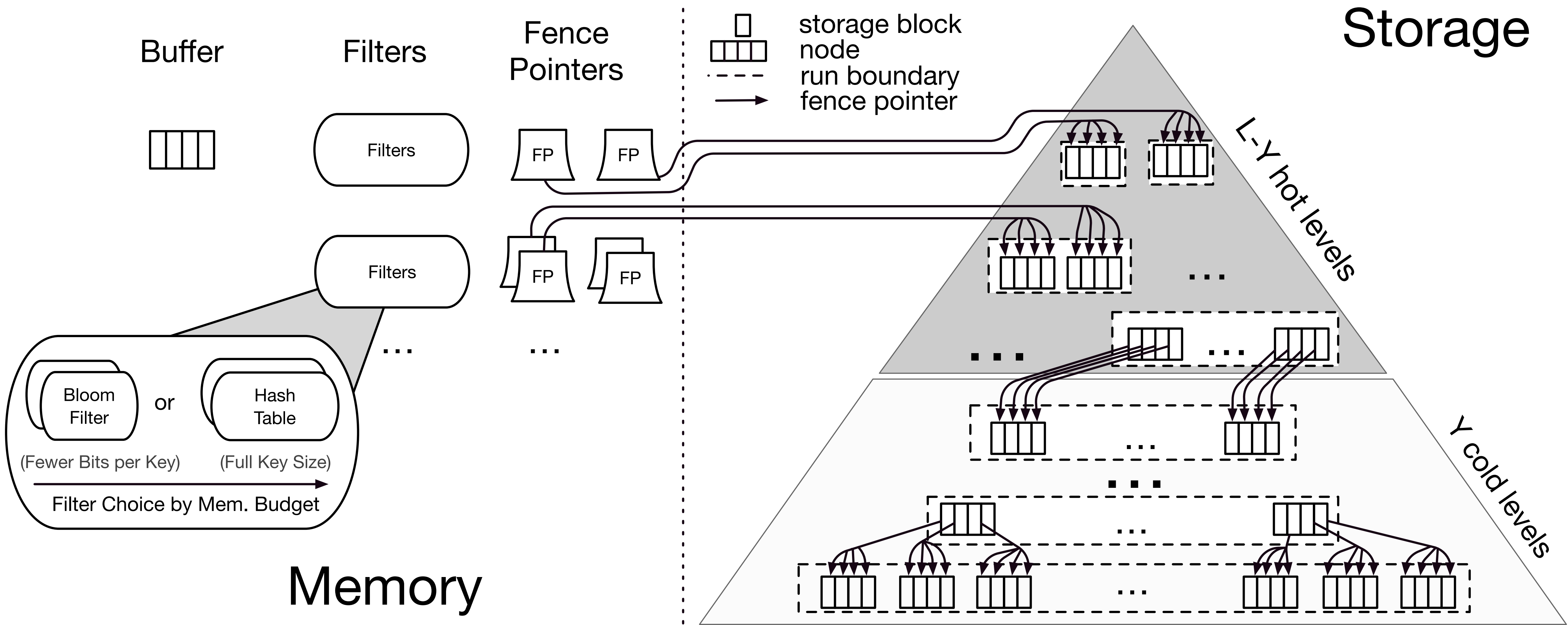
[log, log+hash, LSM-tree*, B^εTree, B-Tree, Sorted Array]



**design principles
for >1 structures**

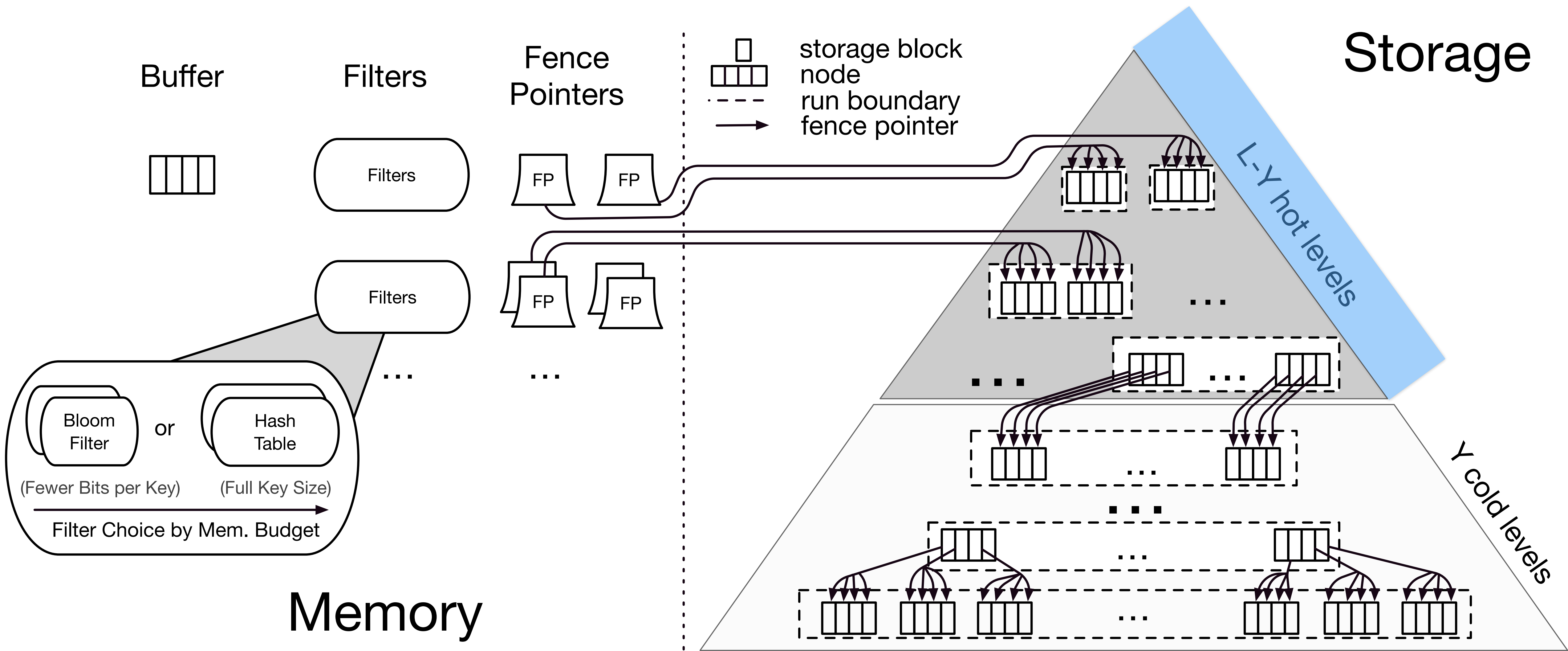
a unified design space

[log, log+hash, LSM-tree*, B^εTree, B-Tree, Sorted Array]



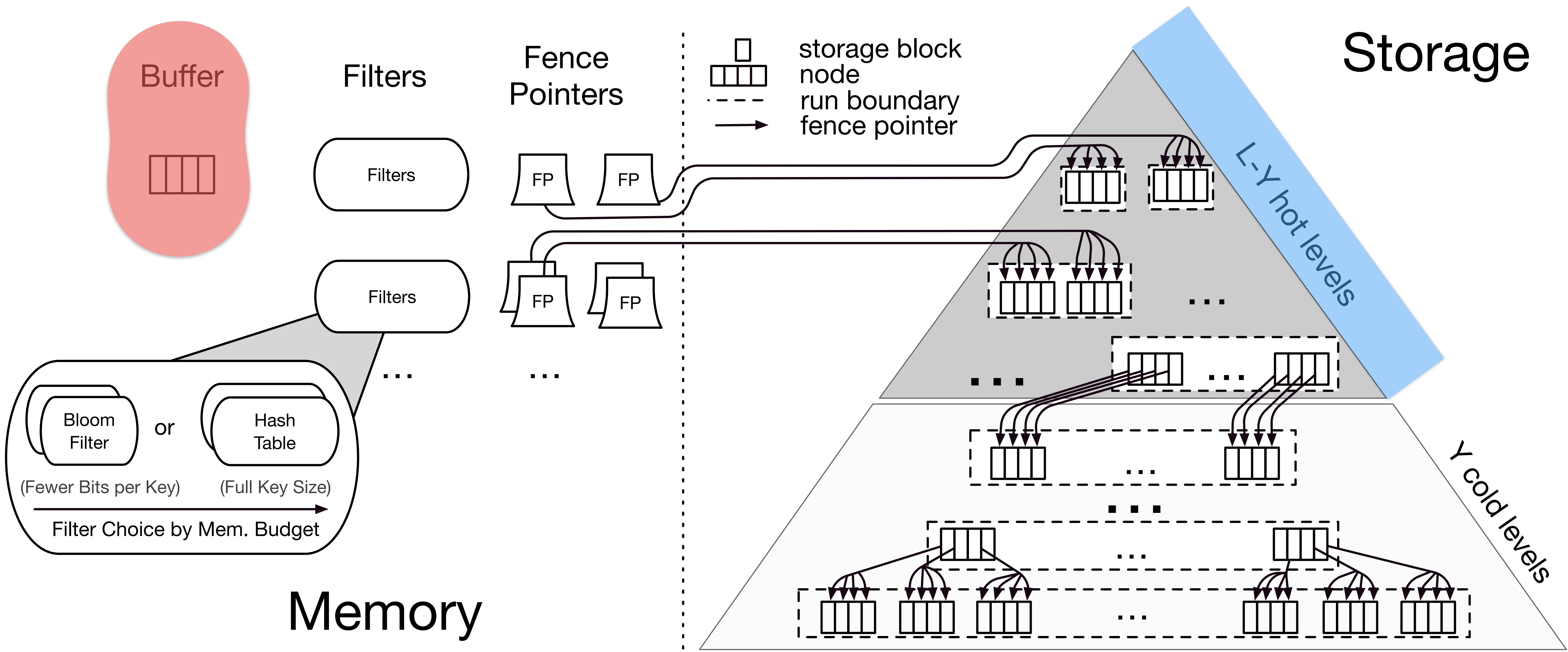
a unified design space

[log, log+hash, LSM-tree*, B^εTree, B-Tree, Sorted Array]



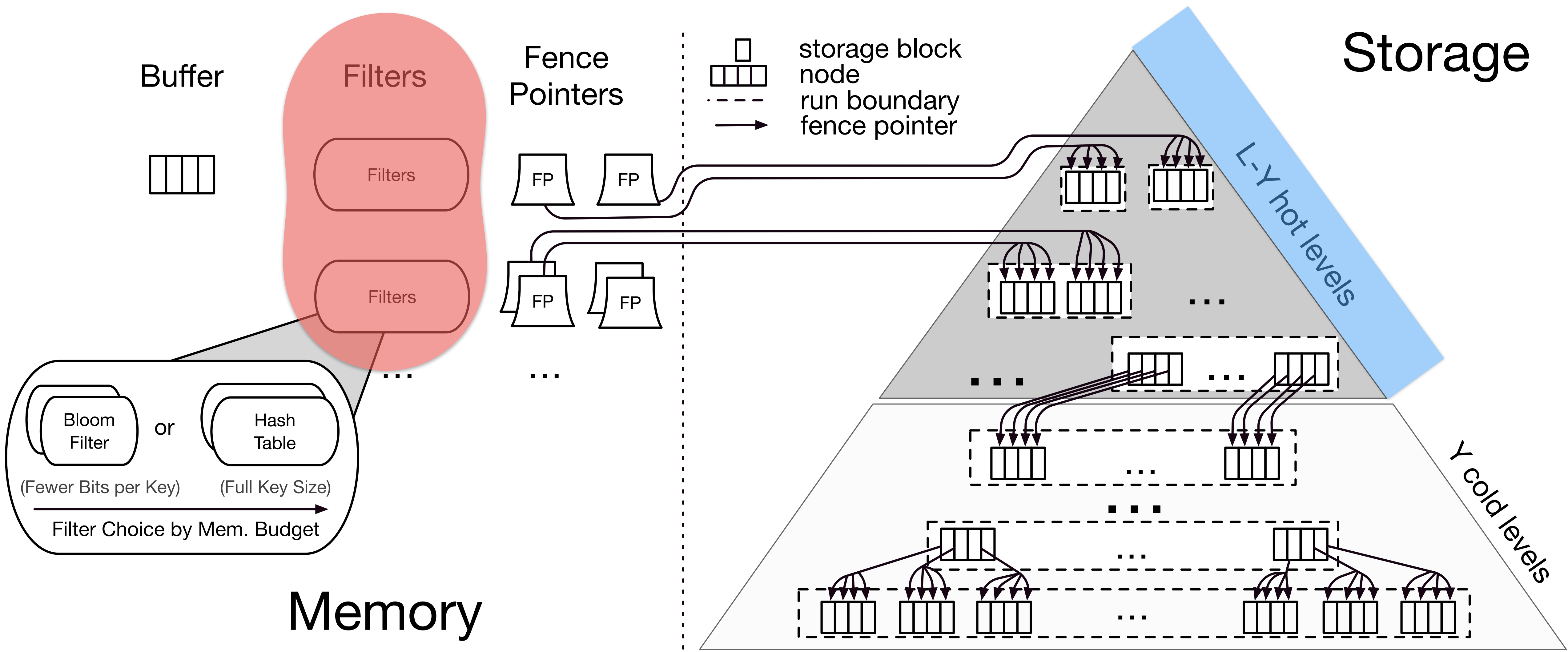
a unified design space

[log, log+hash, LSM-tree*, B^εTree, B-Tree, Sorted Array]



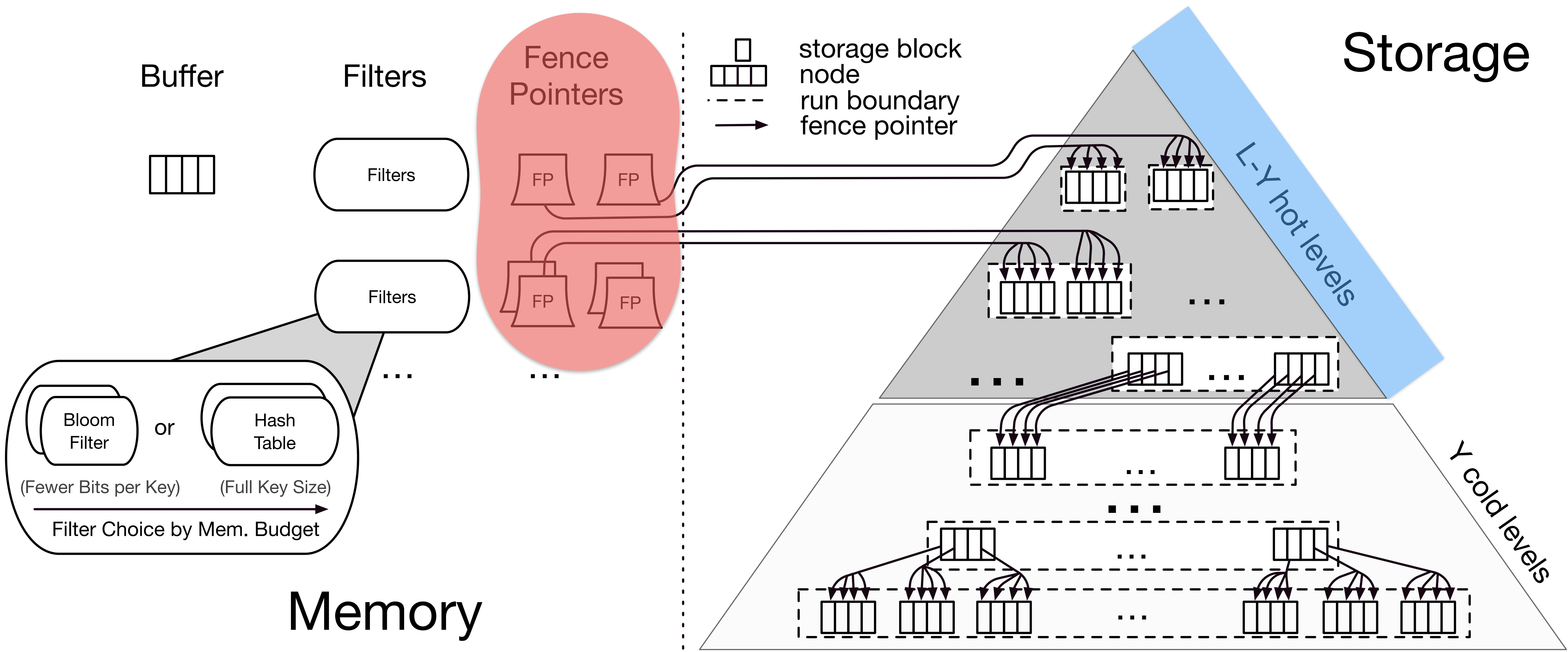
a unified design space

[log, log+hash, LSM-tree*, B^εTree, B-Tree, Sorted Array]



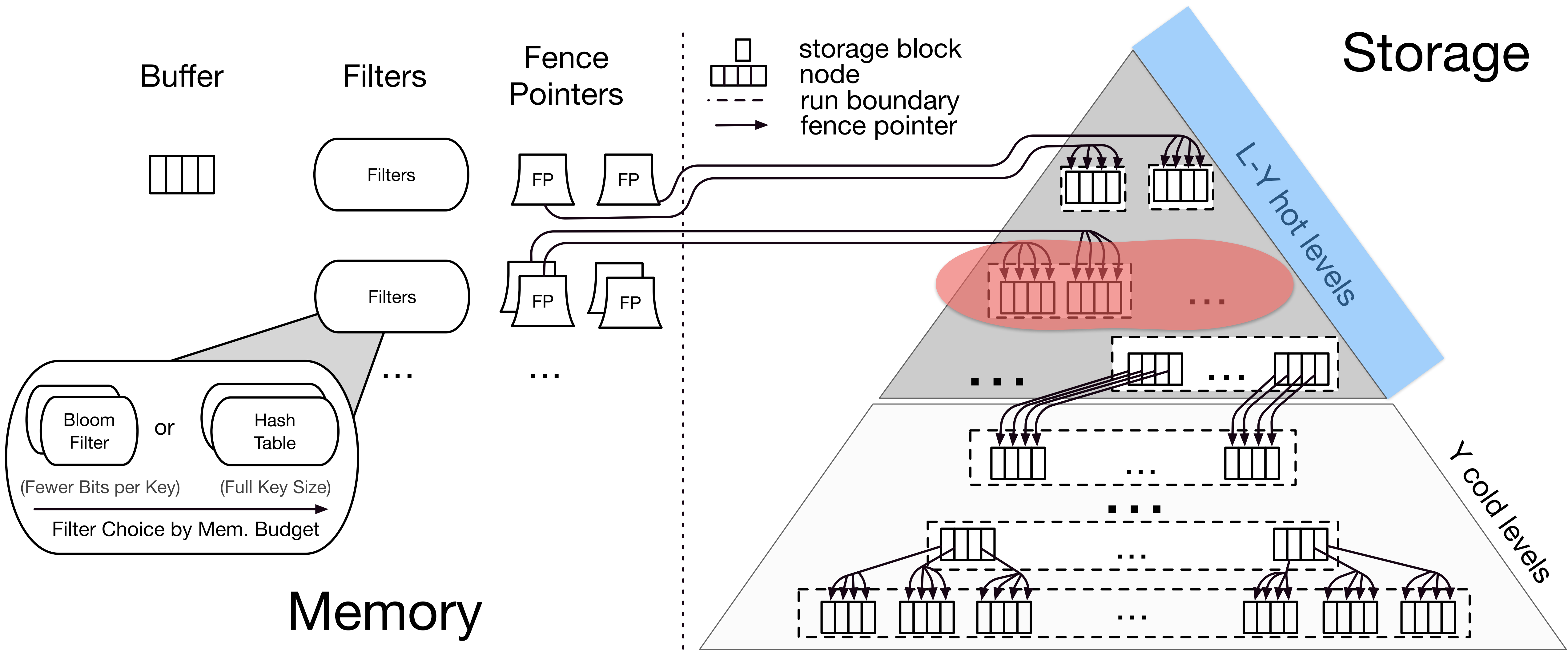
a unified design space

[log, log+hash, LSM-tree*, B^εTree, B-Tree, Sorted Array]



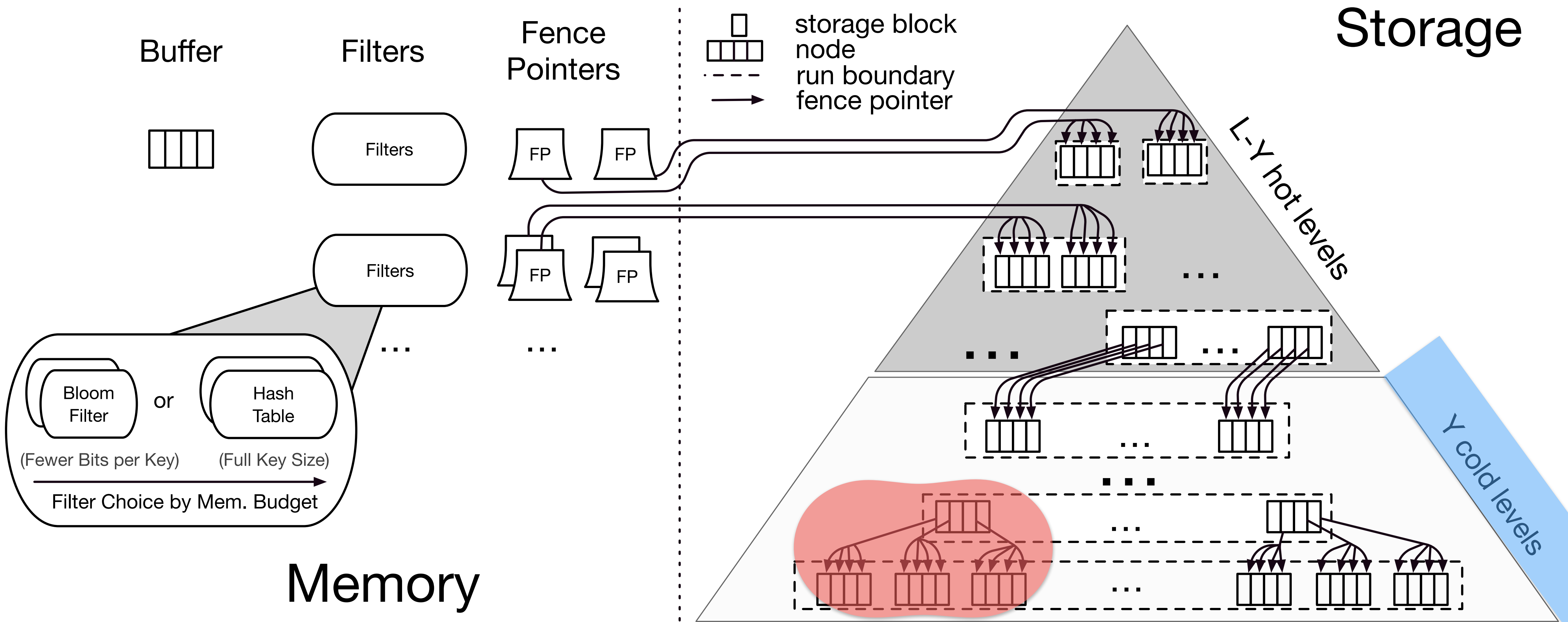
a unified design space

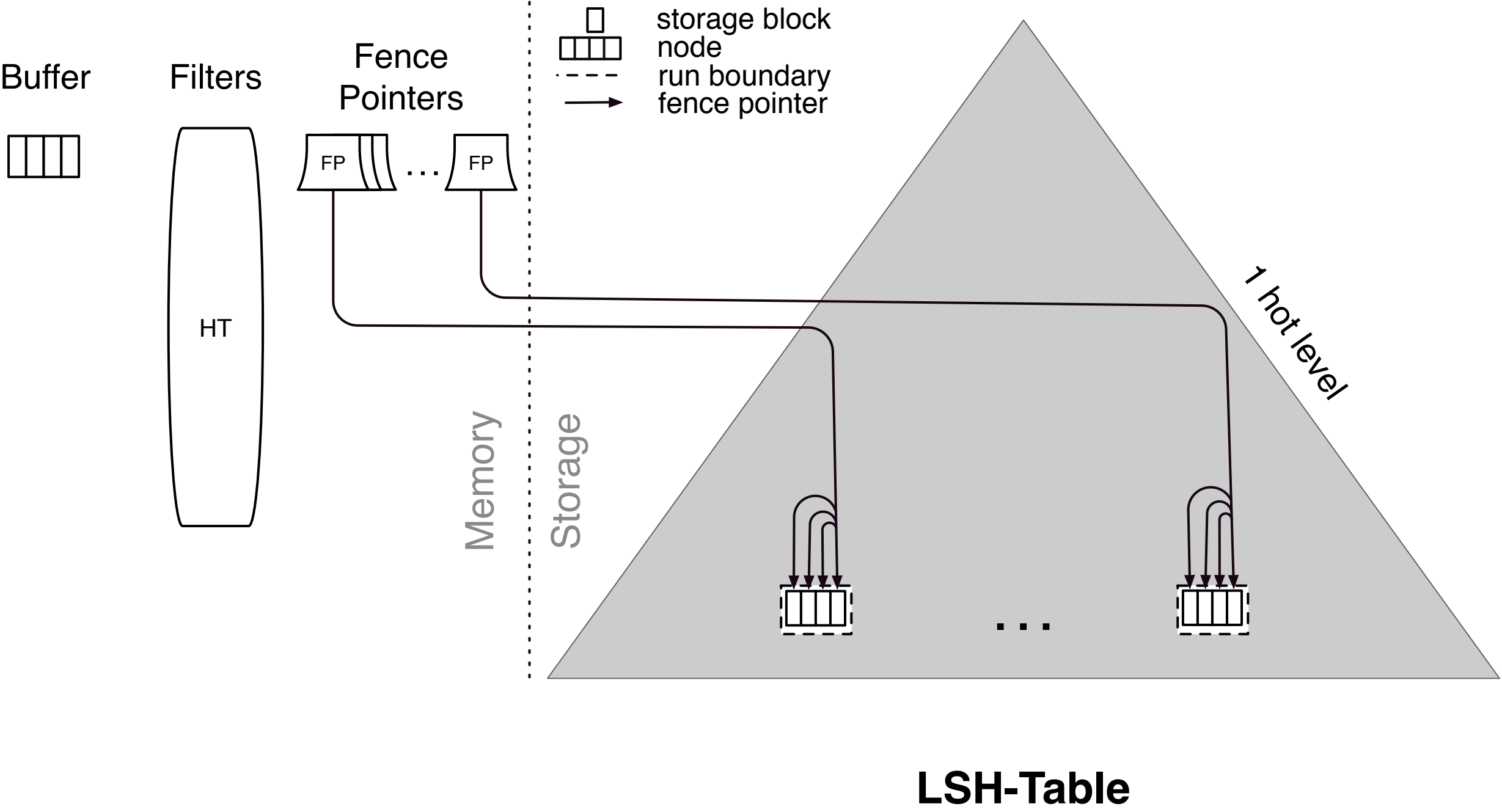
[log, log+hash, LSM-tree*, B^εTree, B-Tree, Sorted Array]

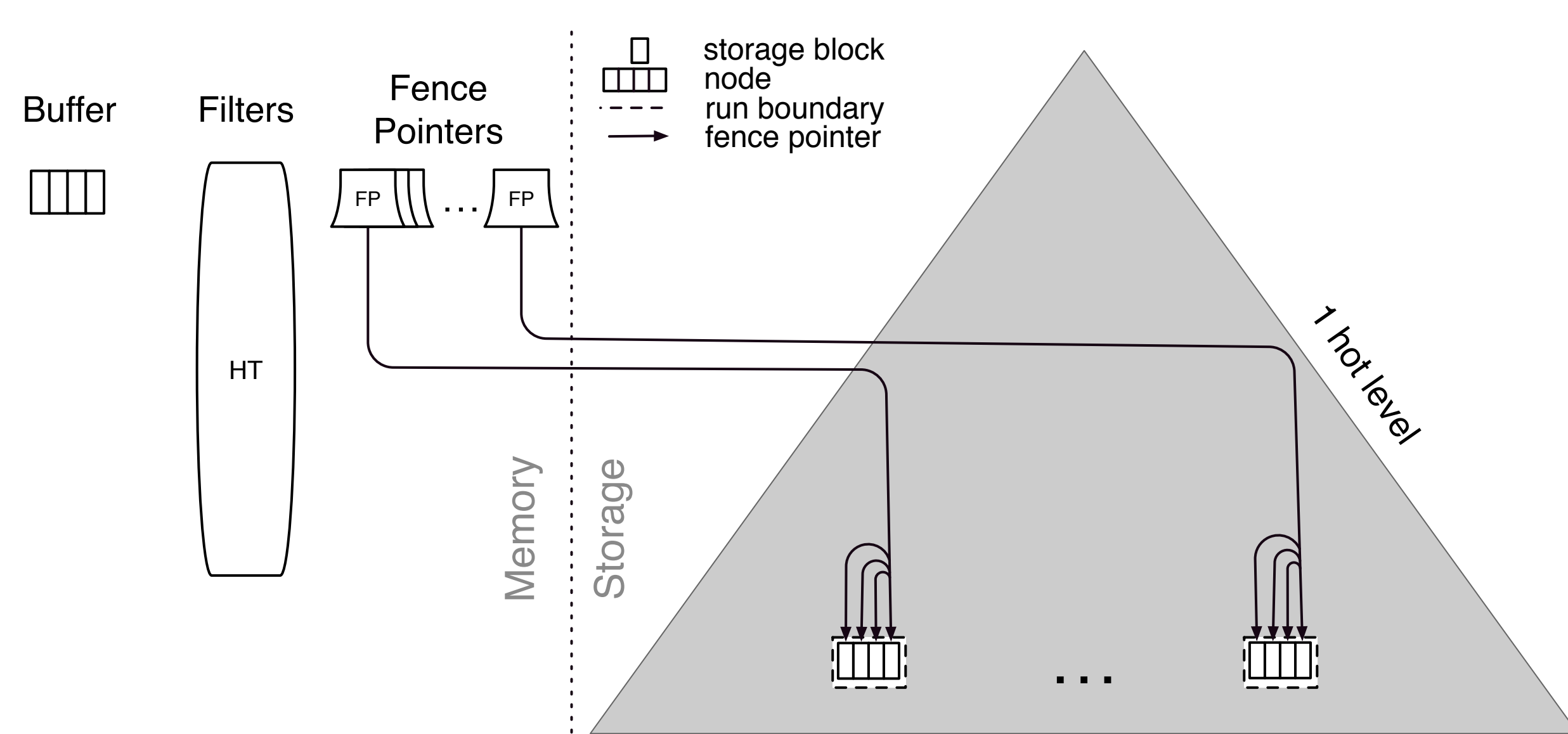


a unified design space

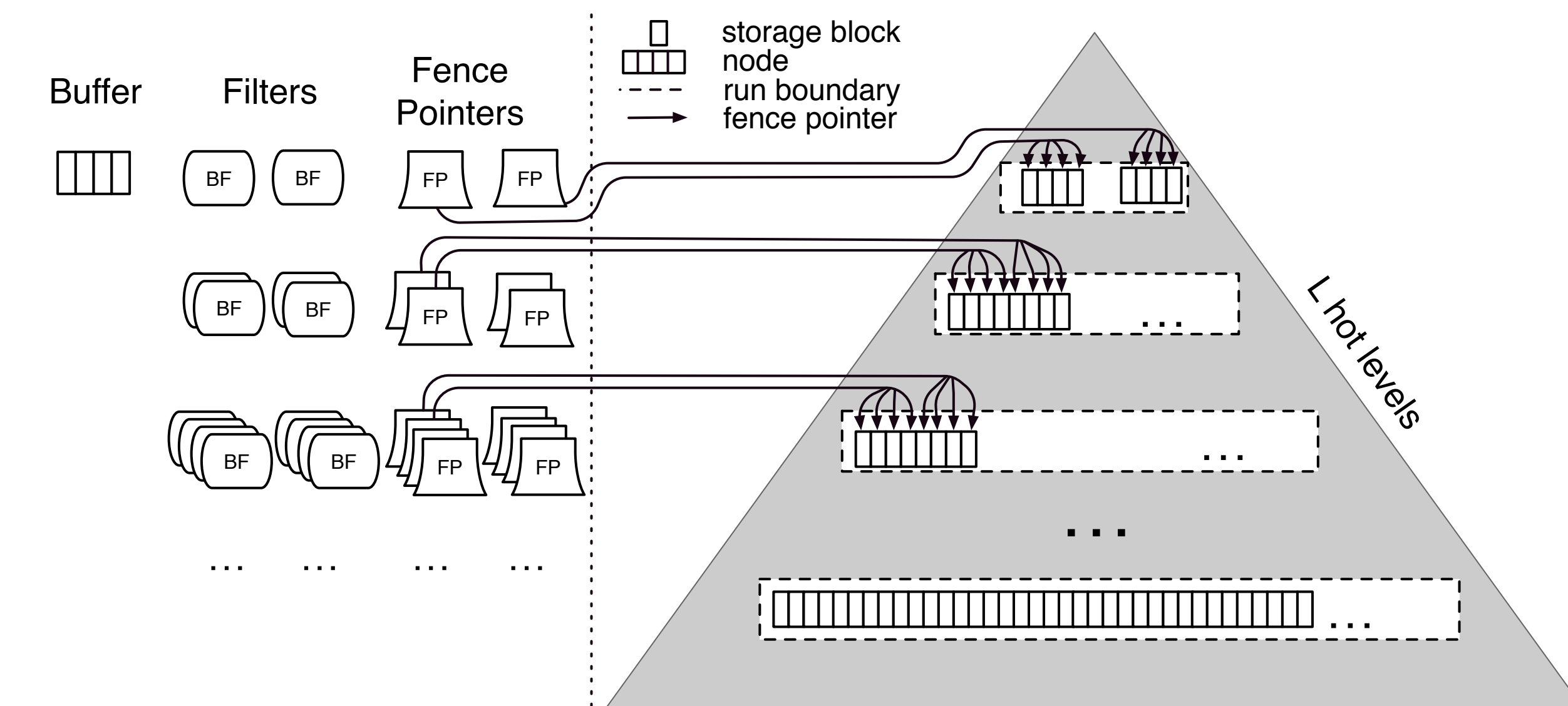
[log, log+hash, LSM-tree*, B^εTree, B-Tree, Sorted Array]



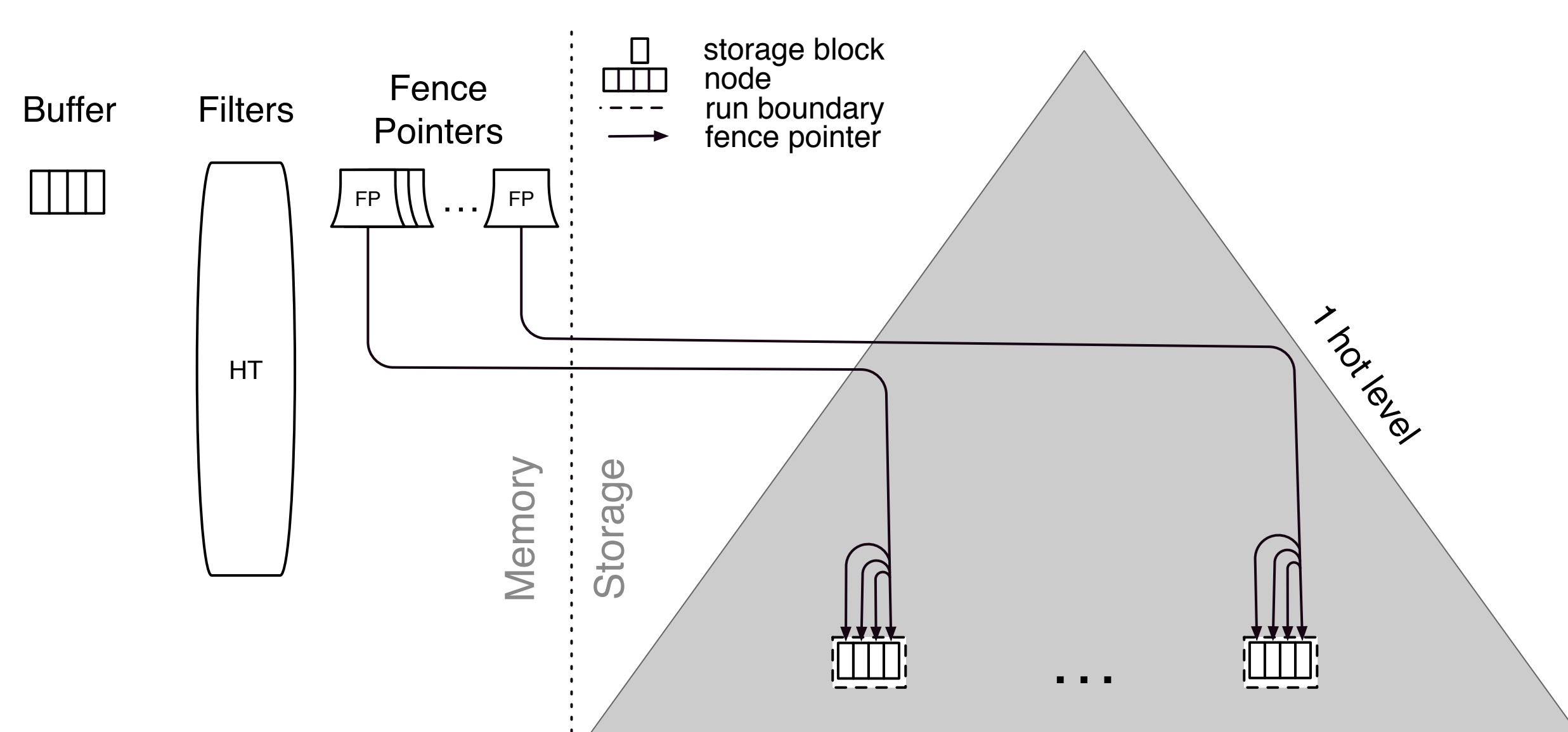




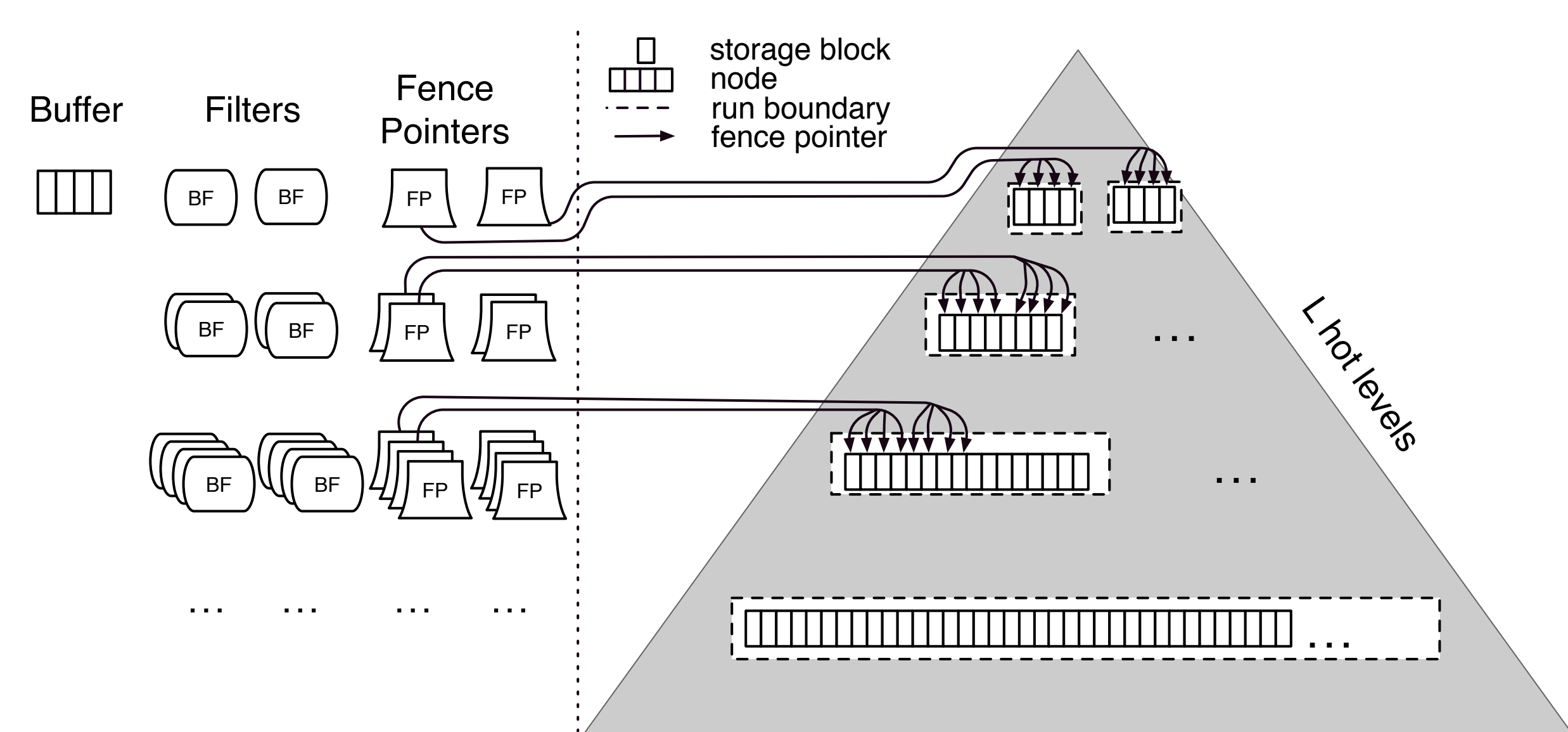
LSH-Table



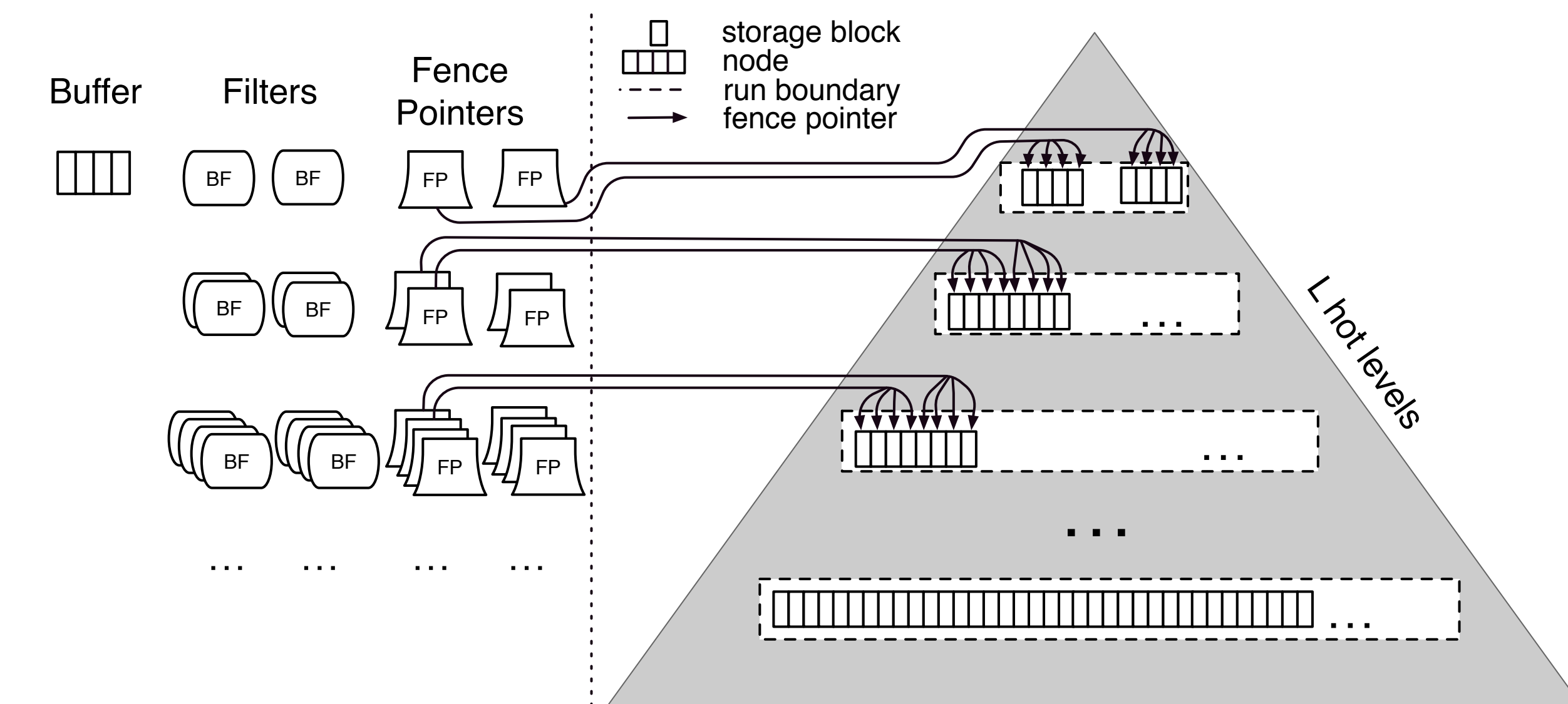
Leveled LSM-Tree



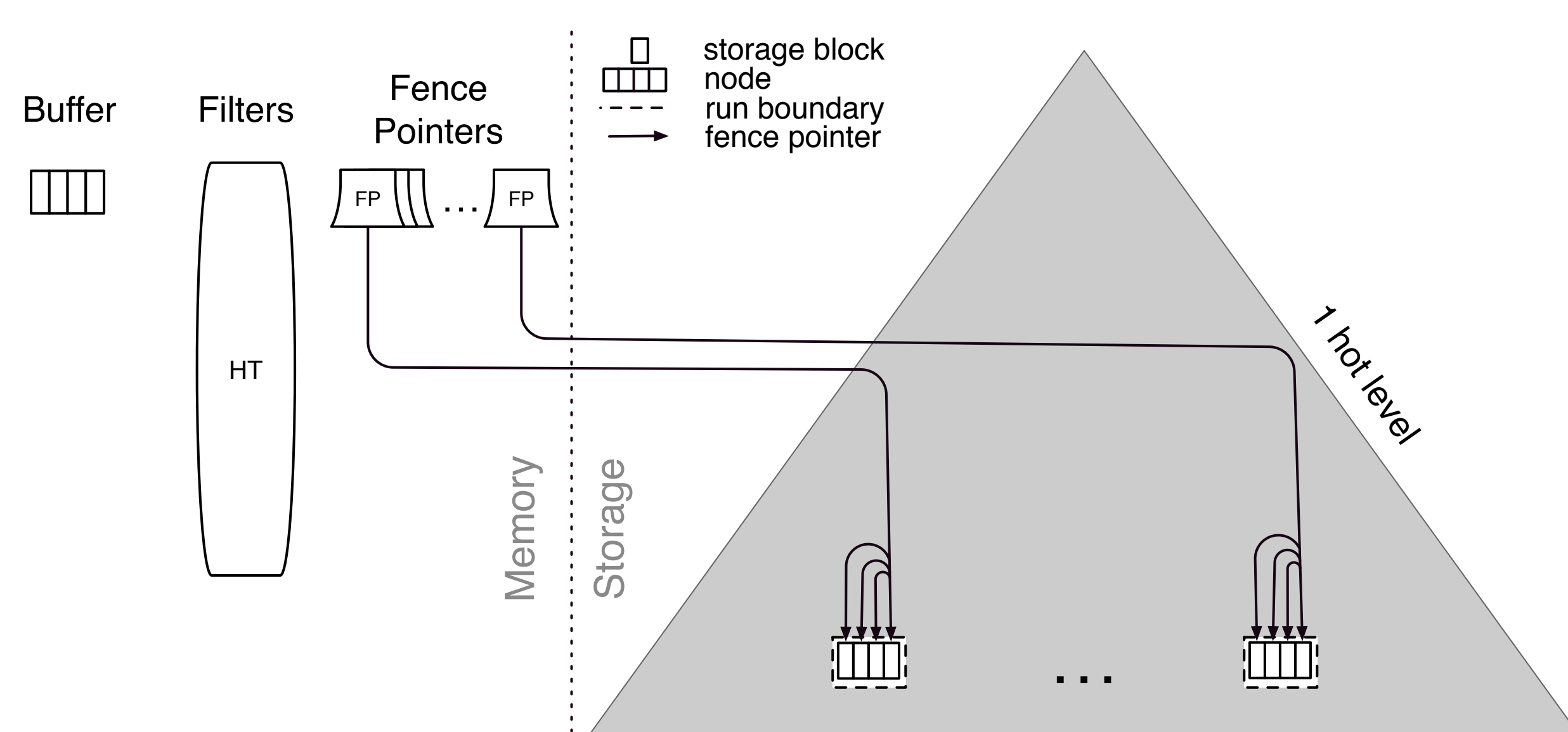
LSH-Table



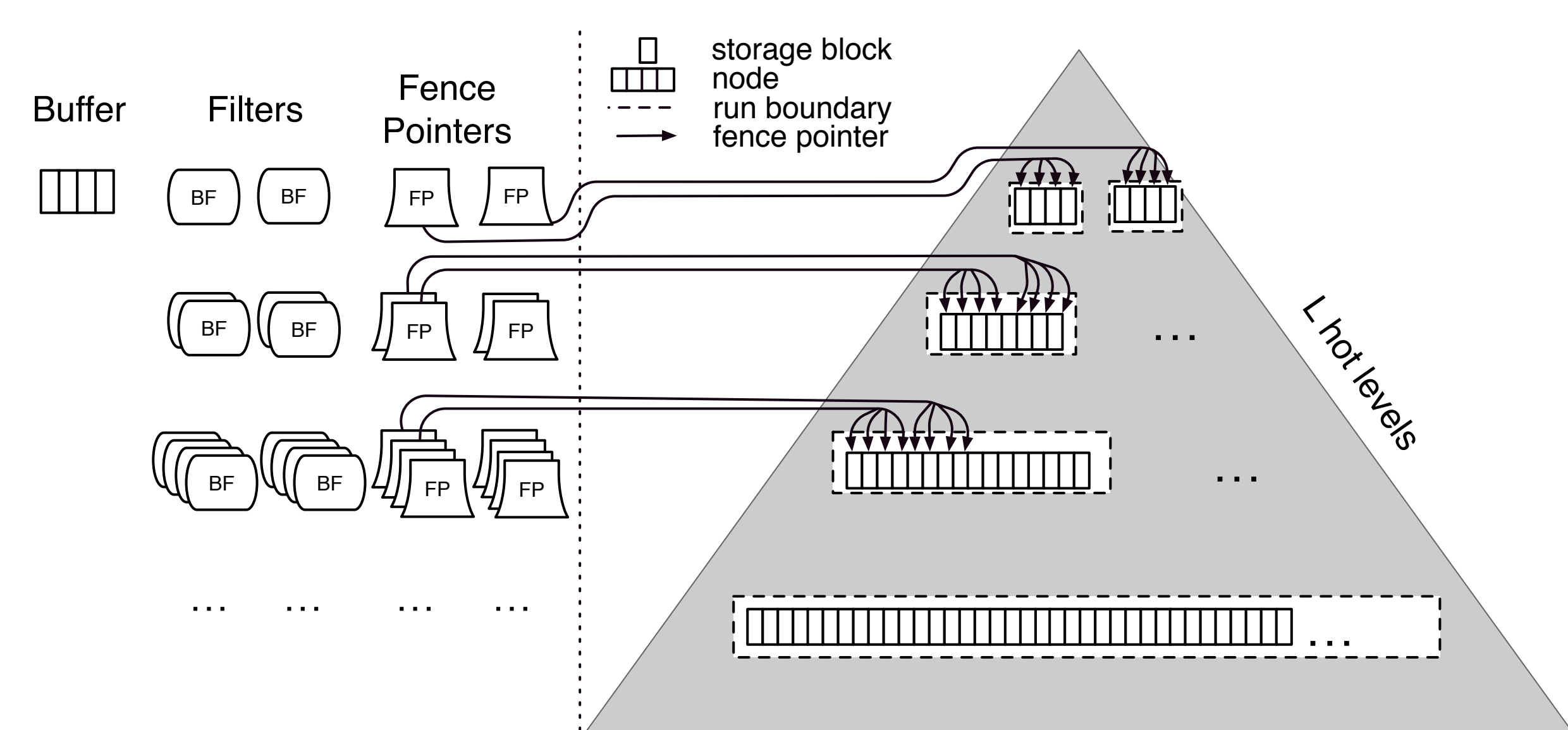
Lazy Levelled LSM-Tree



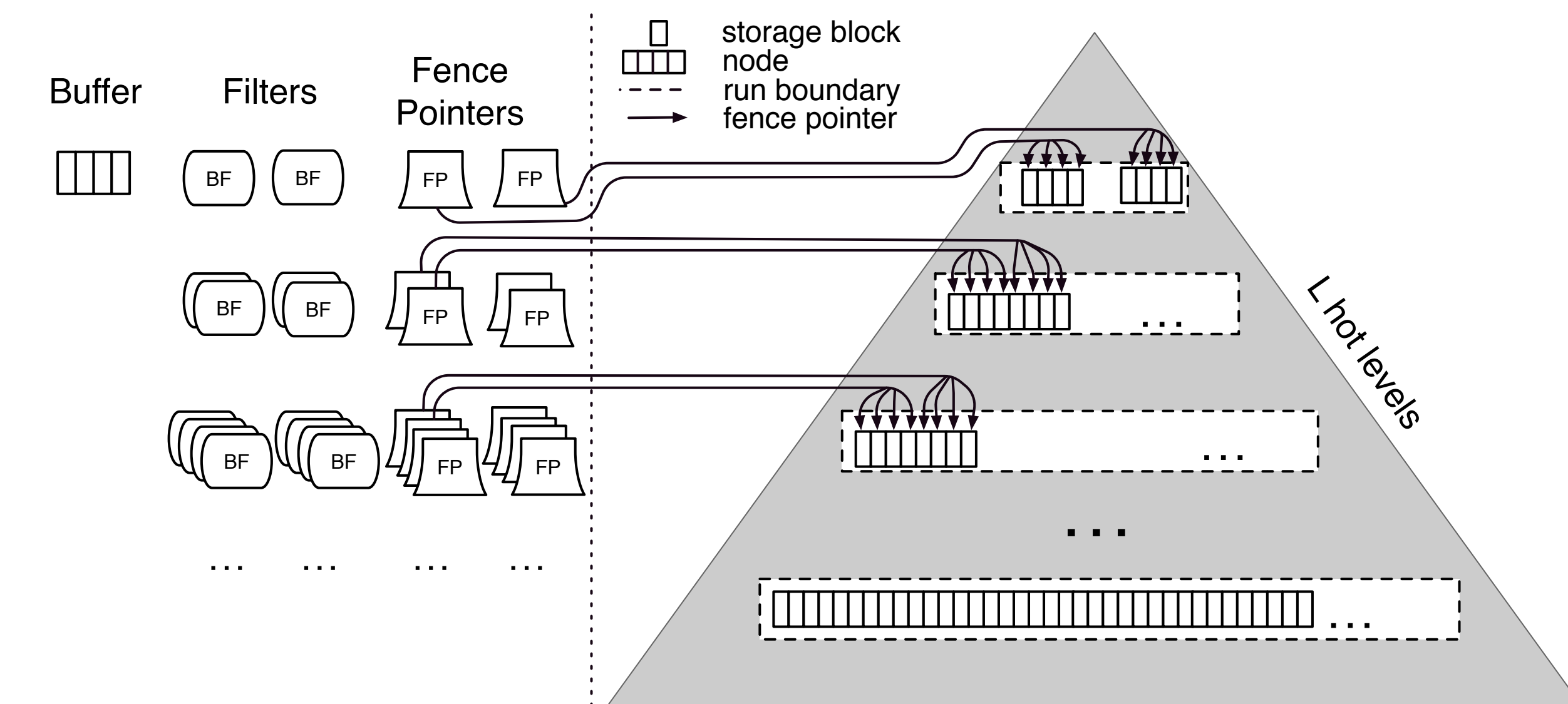
Levelled LSM-Tree



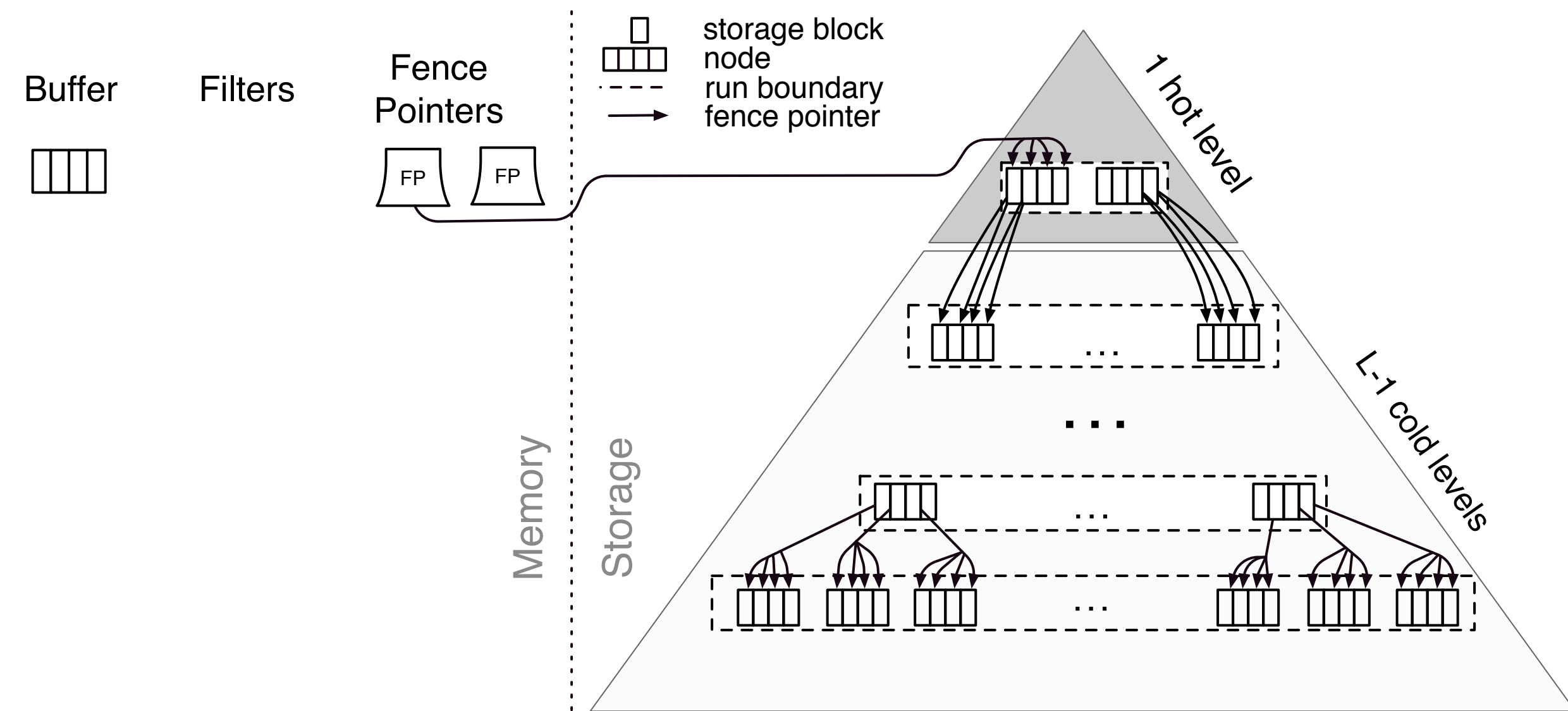
LSH-Table



Lazy Levelled LSM-Tree



Levelled LSM-Tree



B_ϵ Tree

CS 265

Stratos Idreos

BIG DATA SYSTEMS

NoSQL | Neural Networks | Image AI | LLMs | Data Science