Data Calculator

Stratos Idreos & Data Systems Lab

DASlab
@ Harvard SEAS

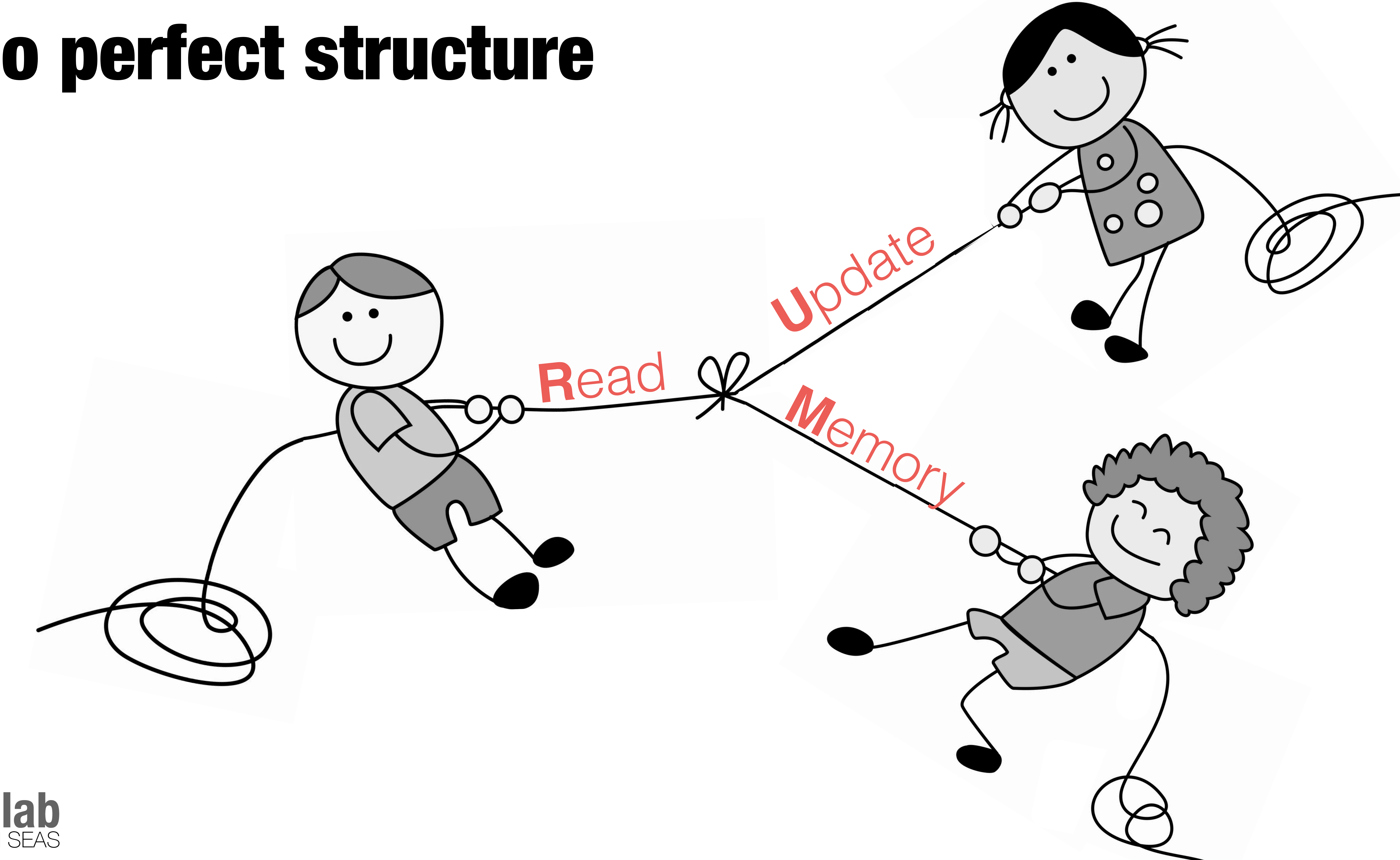
Data Calculator

Stratos Idreos & Data Systems Lab

DASlab
@ Harvard SEAS

WHAT IF WE COULD **REASON** ABOUT THE **DESIGN SPACE** OF DATA STRUCTURES?

1. no perfect structure



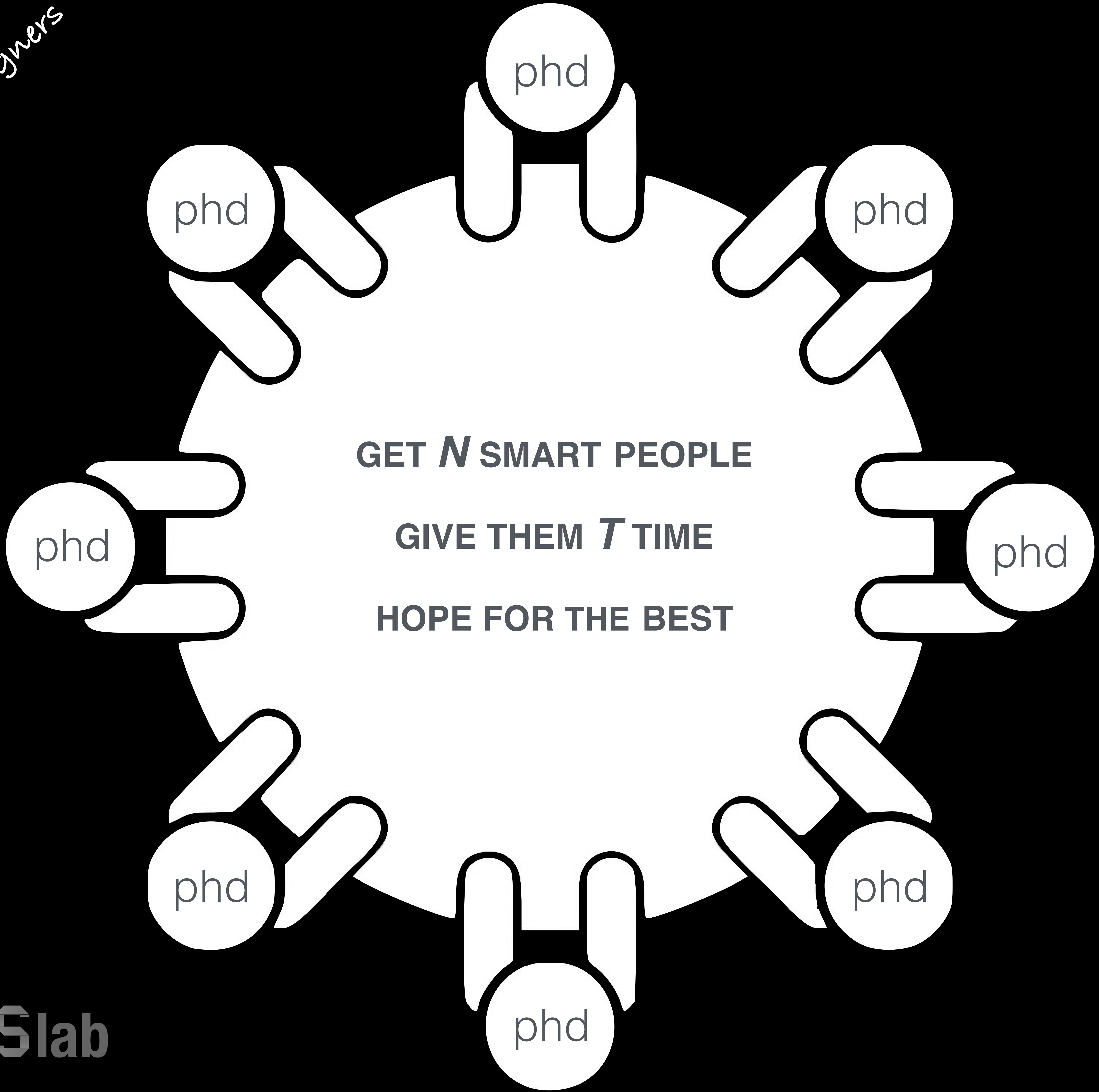
1. no perfect structure

Read Optimized
Hash
Point &Tree indexes
B-Tree
Trie
Skiplist
Cracking
Adaptive structures
Merging
PDT
LSM
PBT
Differential structures
MaSM
Sparse Index
Bloom filter
Approximate indexes
Bitmap
Write Optimized
Space Optimized
Read
Update
Memory

1. no perfect structure
2. apps & h/w evolve

Read Optimized
Hash
Point &Tree indexes
B-Tree
Trie
Skiplist
Cracking
Adaptive structures
Merging
PDT
LSM
PBT
Differential structures
MaSM
Sparse Index
Bloom filter
Approximate indexes
Bitmap
Write Optimized
Space Optimized
Read
Update
Memory

the dinning
systems designers

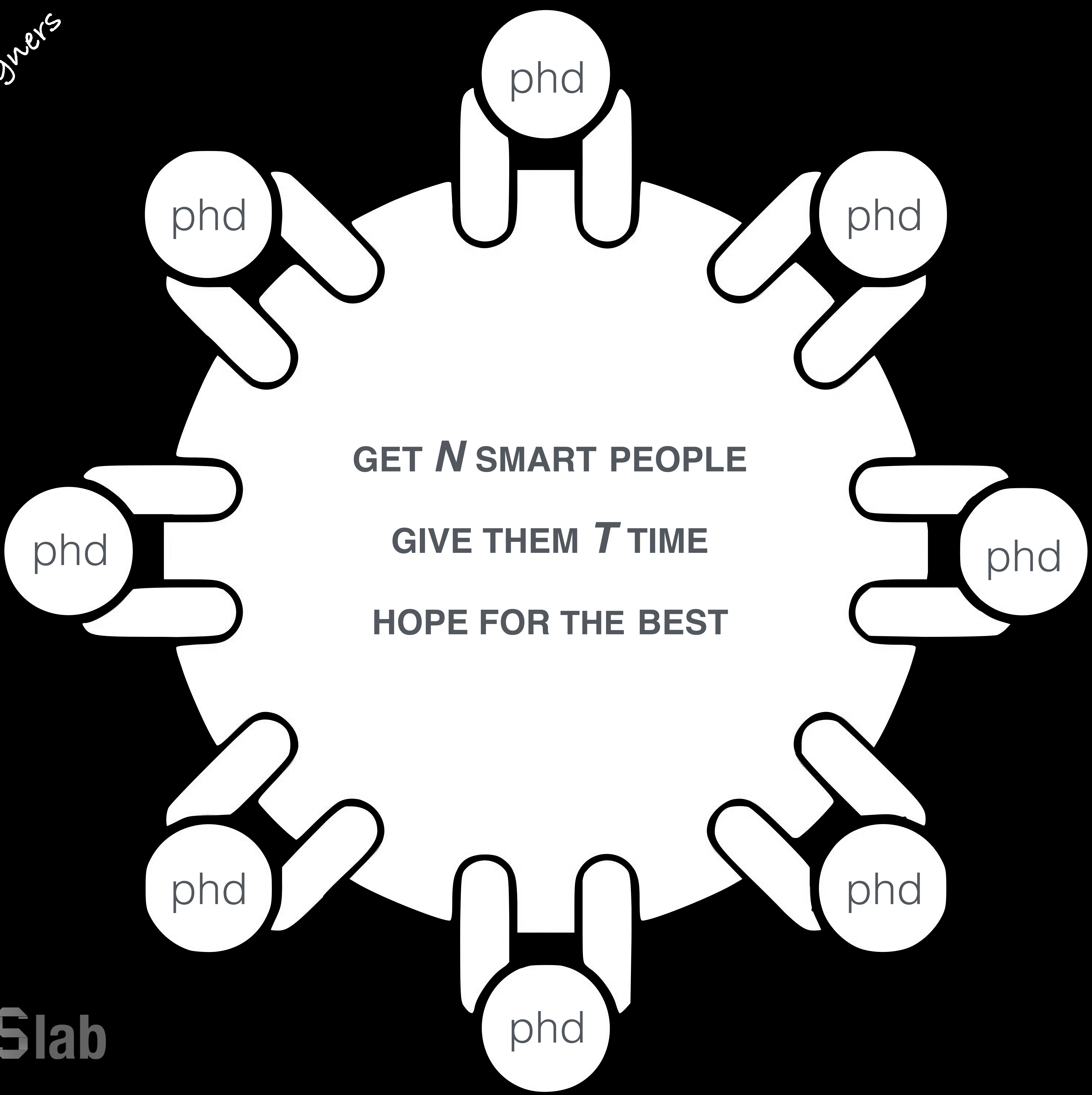


GET *N* SMART PEOPLE

GIVE THEM *T* TIME

HOPE FOR THE BEST

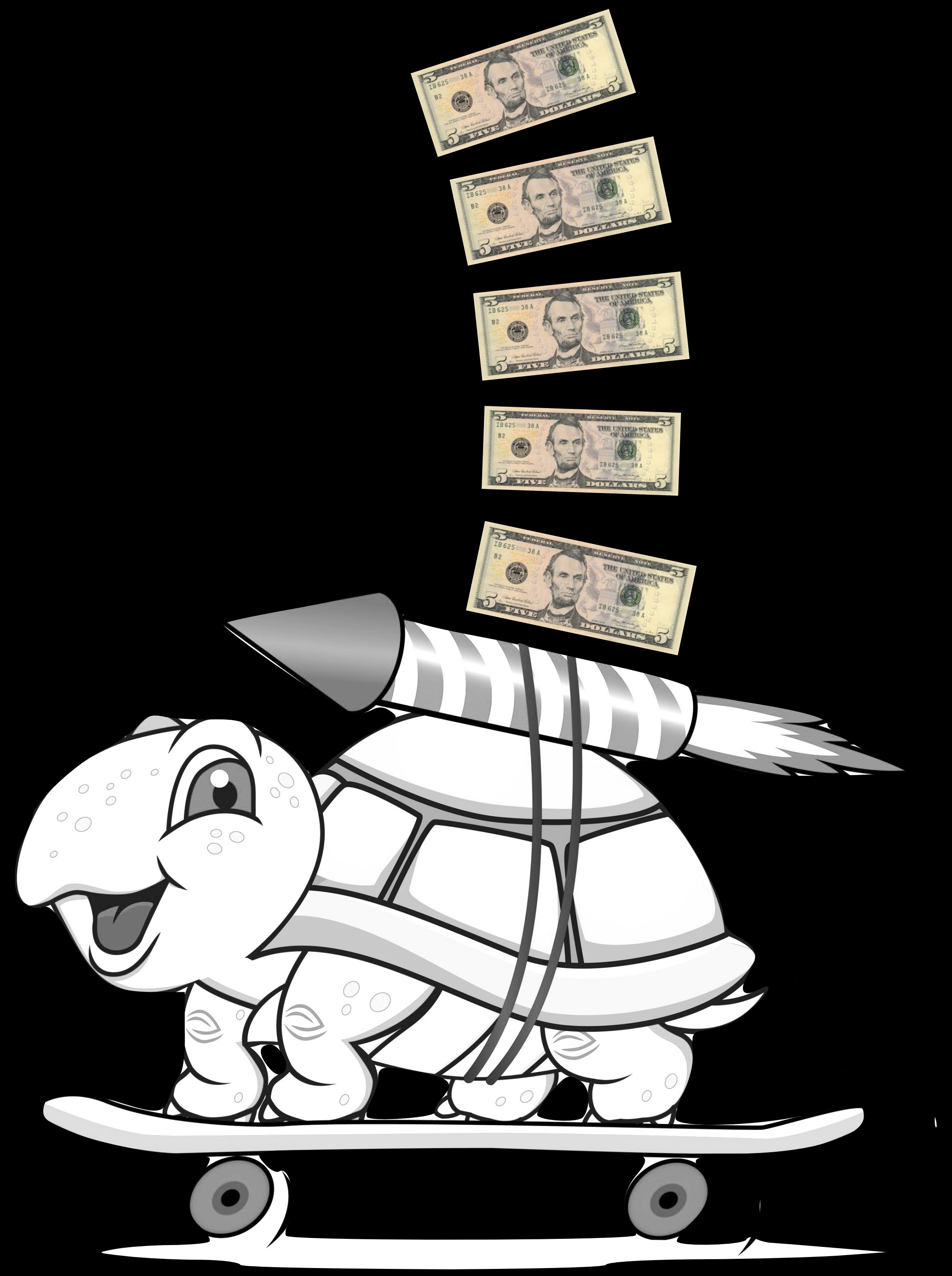
the dinning systems designers



GET *N* SMART PEOPLE

GIVE THEM *T* TIME

HOPE FOR THE BEST



AUTO DESIGN

ROBERT TARJAN

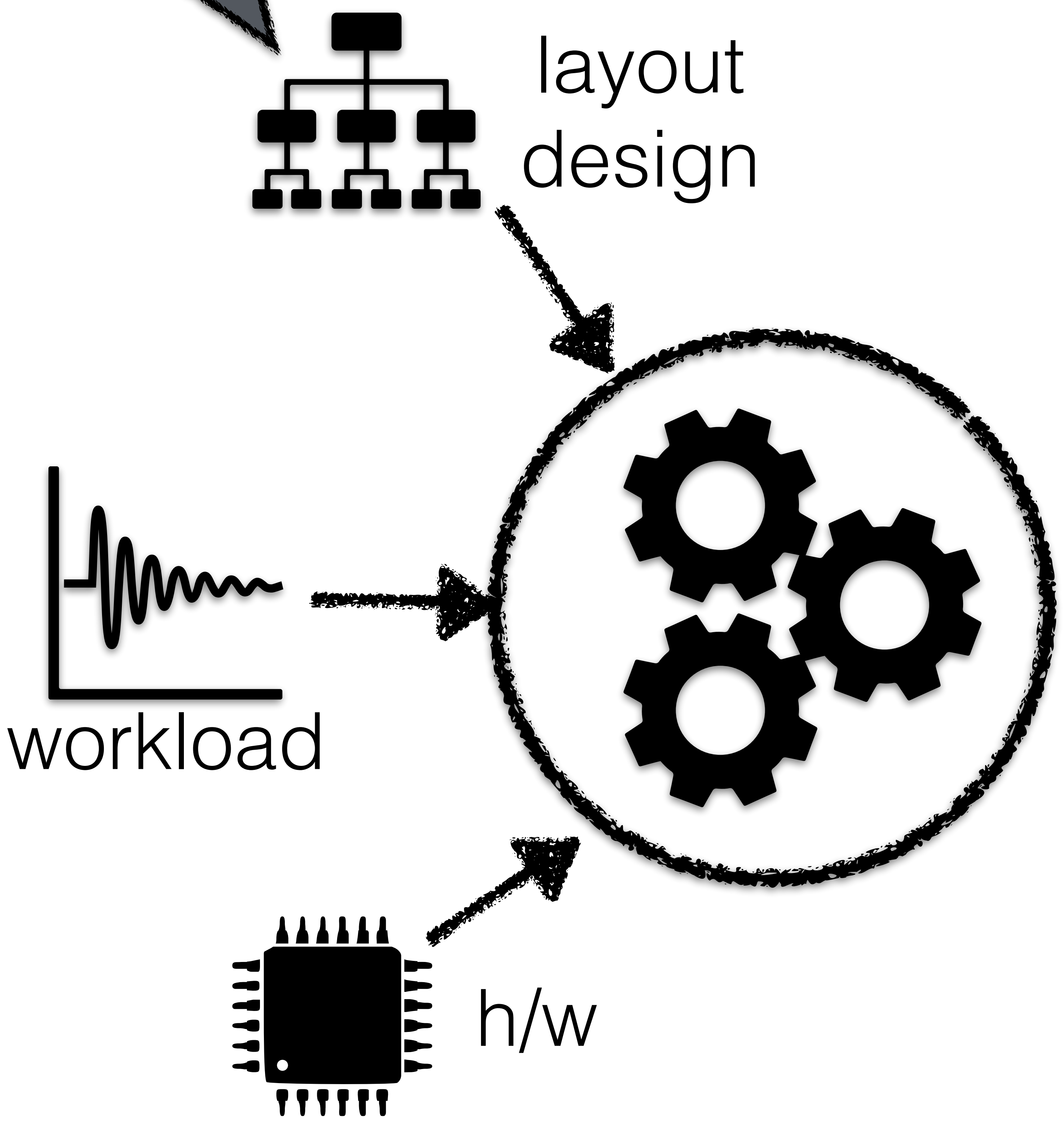
“**IS THERE A CALCULUS OF DATA STRUCTURES**

by which one can choose the appropriate representation and techniques for a given problem?” (SIAM,1978)

HOW MANY AND WHICH?

COMPUTE PERFORMANCE?

DESIGN SPACE OF POSSIBLE STORAGE LAYOUTS

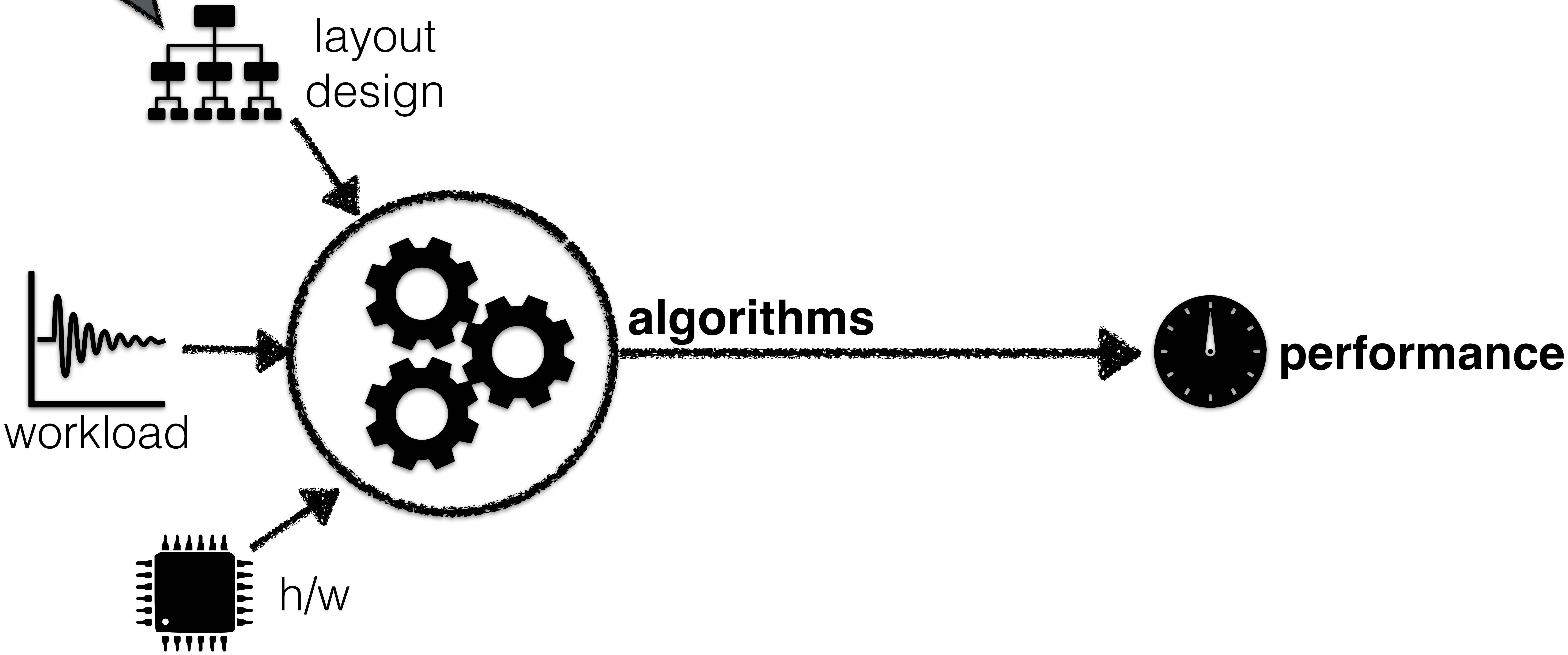


layout design

workload

h/w

DESIGN SPACE
OF POSSIBLE
STORAGE LAYOUTS



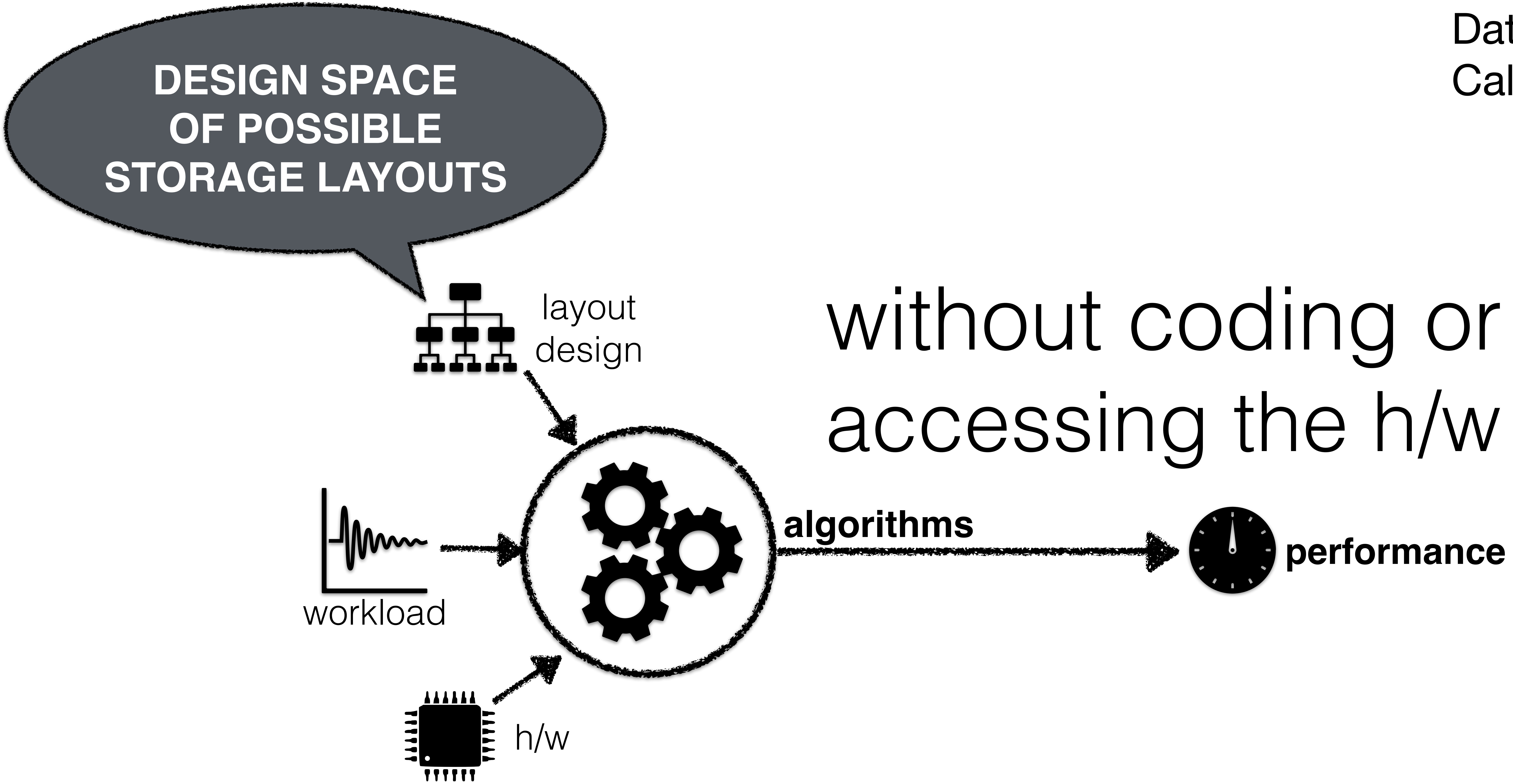
layout design

workload

h/w

algorithms

performance



without coding or accessing the h/w

What-if Design

Auto-Design

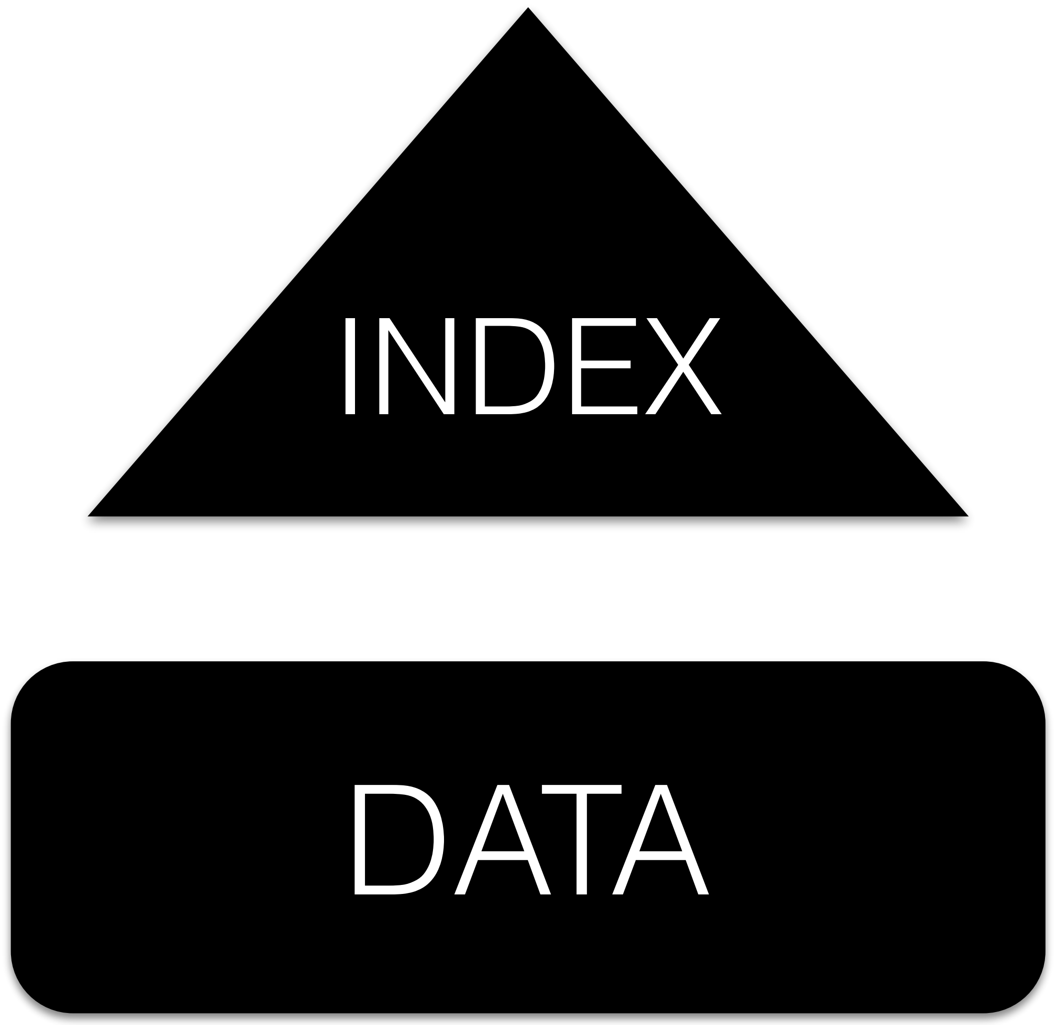
Self-designing Systems

DESIGN SPACE

COST SYNTHESIS

HOW TO USE

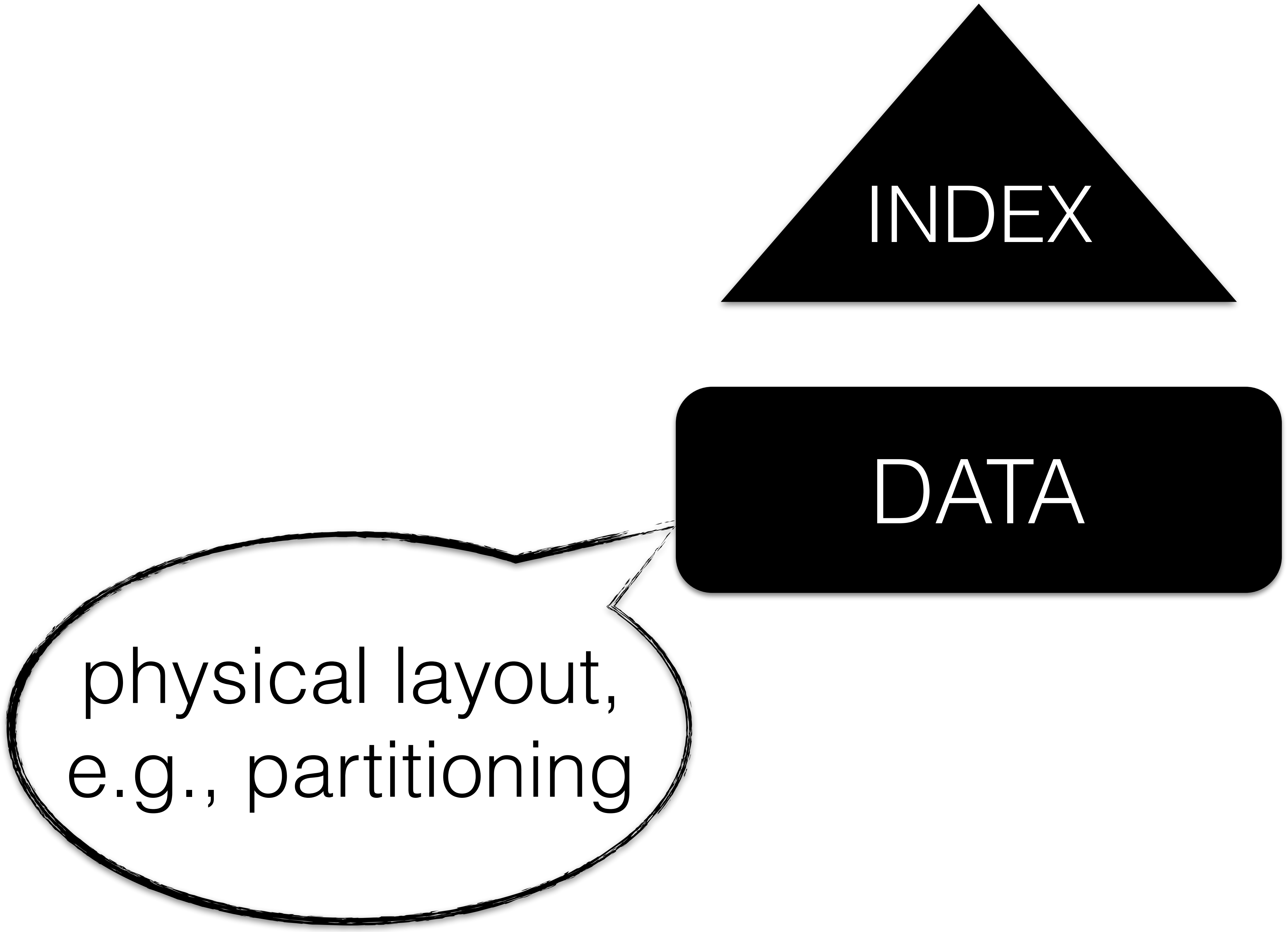
DESIGN SPACE



INDEX

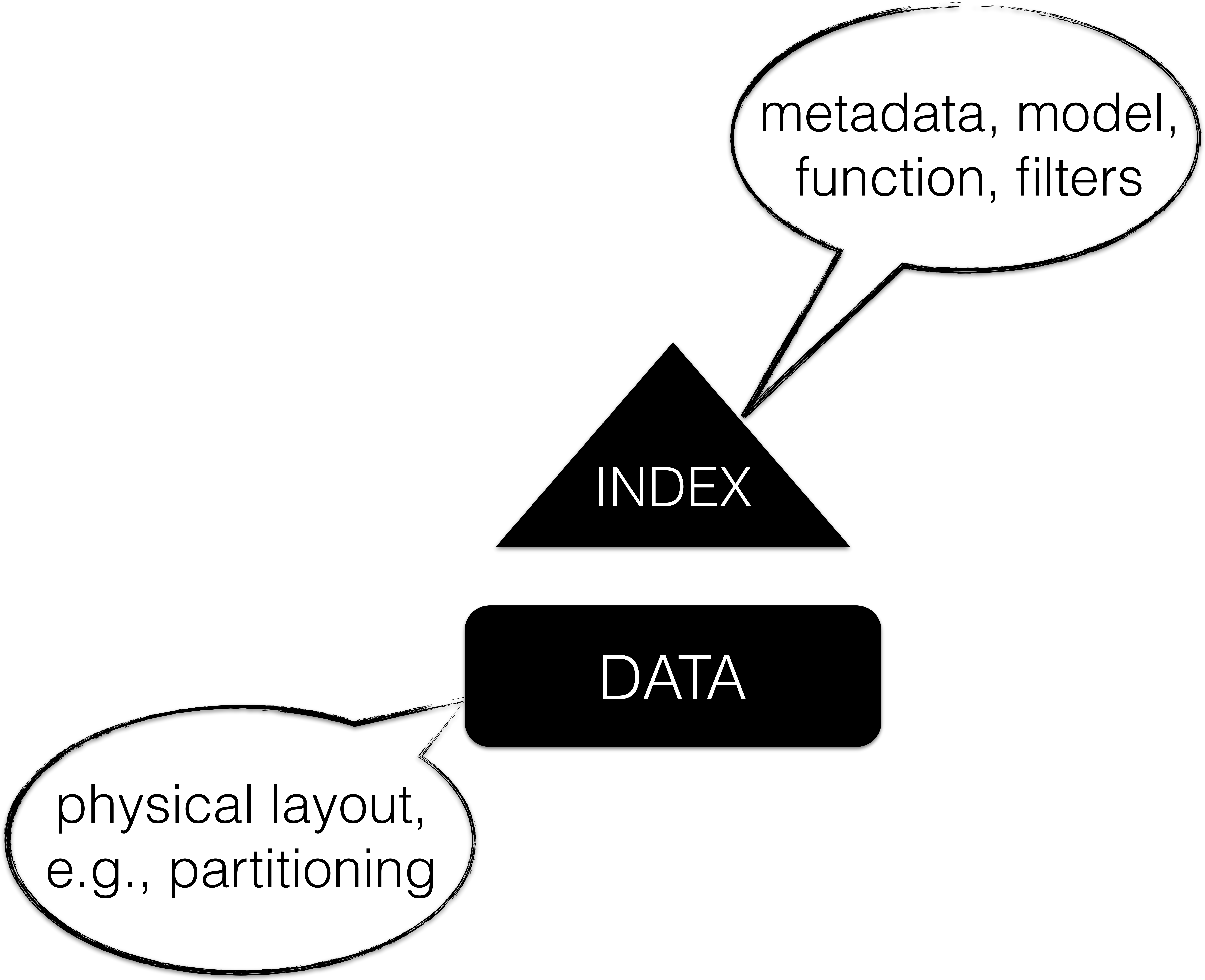
DATA

DESIGN SPACE

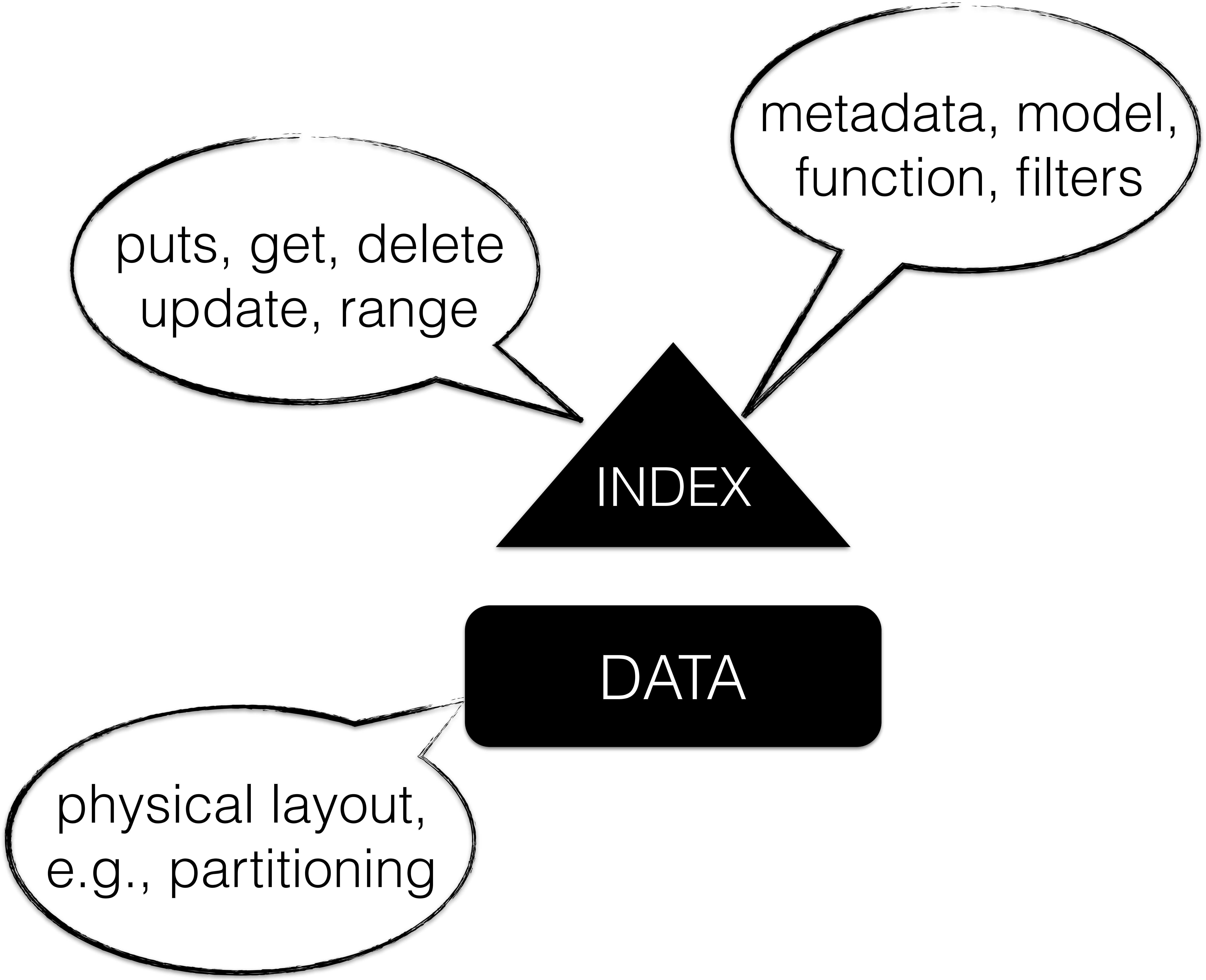


physical layout, e.g., partitioning

DESIGN SPACE


metadata, model, function, filters
INDEX
DATA
physical layout, e.g., partitioning

DESIGN SPACE



puts, get, delete
update, range

metadata, model,
function, filters

INDEX

DATA

physical layout,
e.g., partitioning

1

EACH DESIGN: A **SET** OF CONCEPTS

{partitioning, links, fence pointers,…}

2

DESIGN

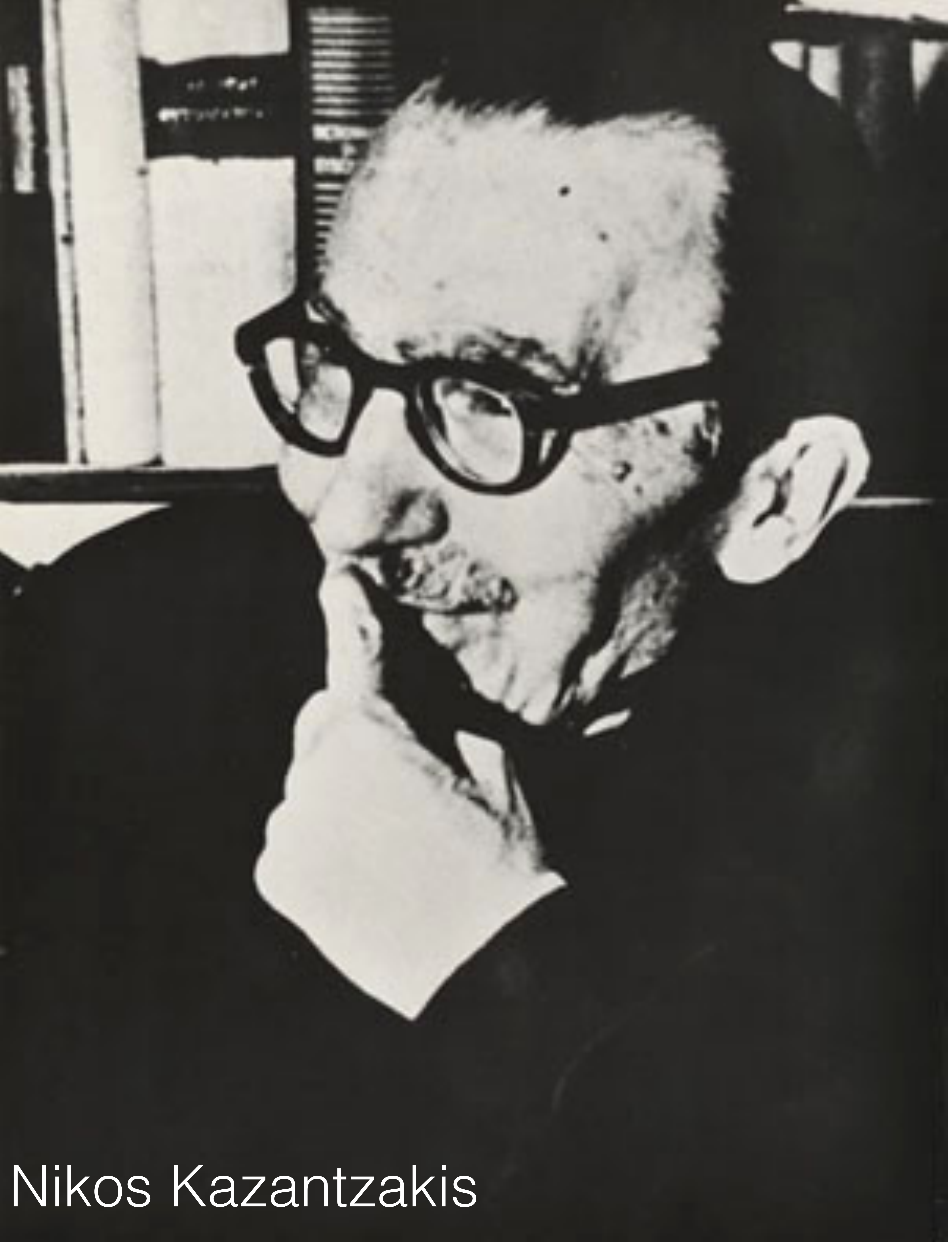
COMBINATION
of existing concepts

TUNING
of existing concepts

NEW
concept

3

(ALMOST) ALL DESIGNS ARE A COMBINATION/TUNING OF **EXISTING CONCEPTS**


Nikos Kazantzakis

I hope for nothing.
I fear nothing.
I am free.

if we know the **fundamental** building blocks,

if we know the **fundamental** building blocks,
how they combine and their properties,

if we know the **fundamental** building blocks,
how they combine and their properties,

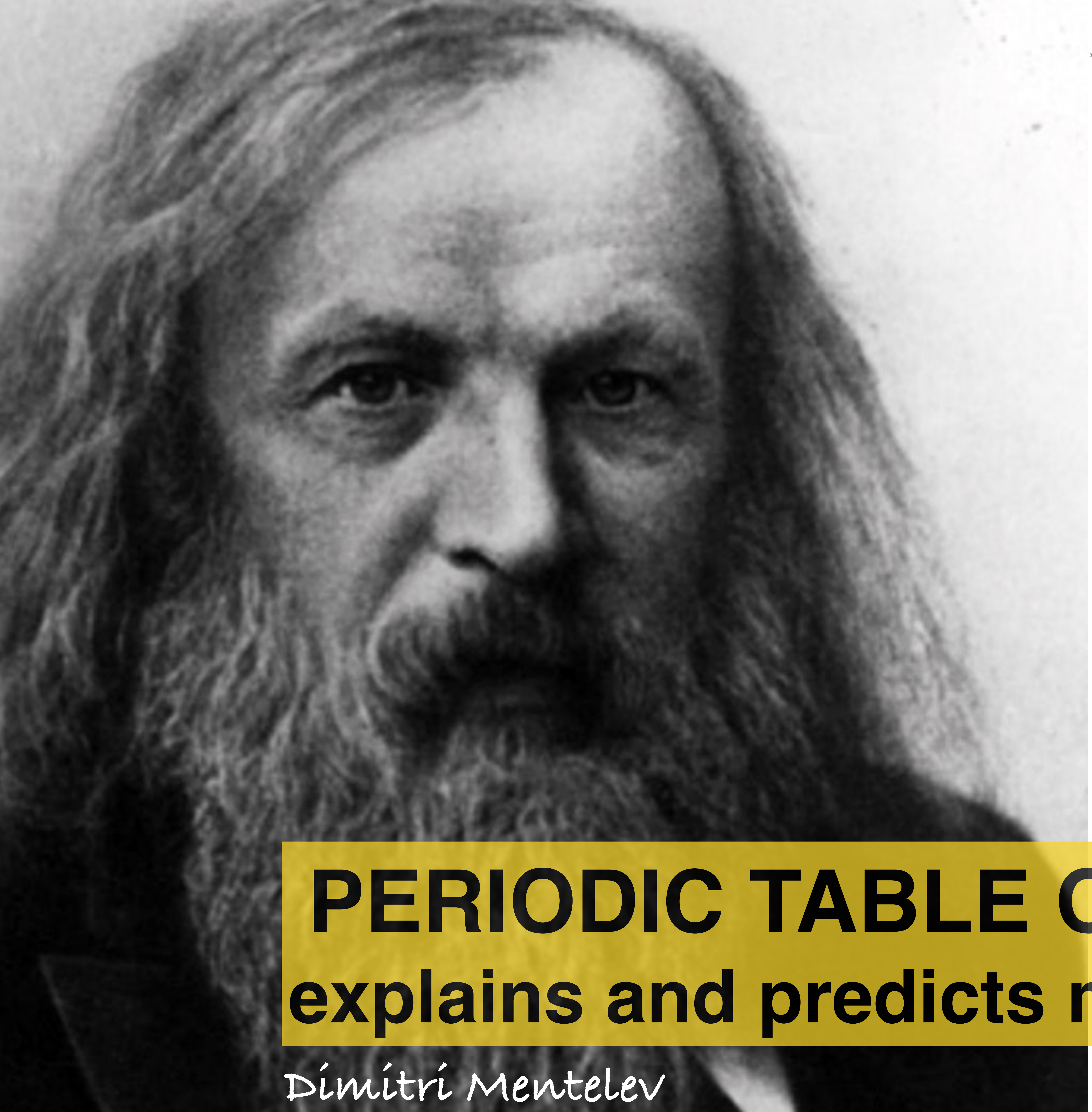
then we can **automate** the discovery of
novel combinations and tunings

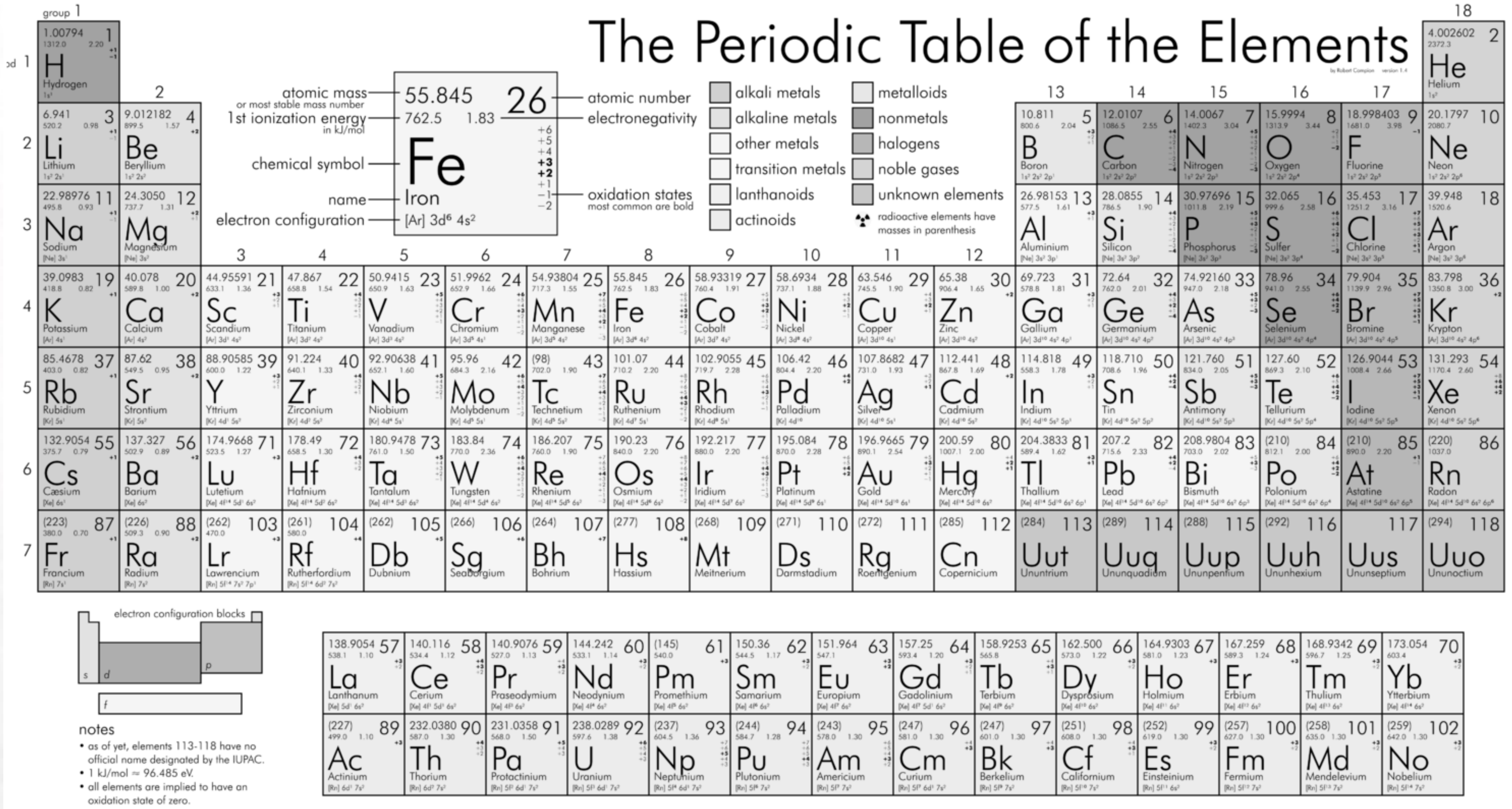
DESIGN SPACE

if we know the **fundamental** building blocks, how they combine and their properties,

then we can **automate** the discovery of novel combinations and tunings

NAVIGATION





The Periodic Table of the Elements

atomic mass or most stable mass number — 55.845 — 26 — atomic number
1st ionization energy in kJ/mol — 762.5 — 1.83 — electronegativity
chemical symbol — Fe — +6 +5 +4 **+3** **+2** +1 −1 −2 — oxidation states most common are bold
name — Iron
electron configuration — [Ar] 3d⁶ 4s²

- □ alkali metals
- □ alkaline metals
- □ other metals
- □ transition metals
- □ lanthanoids
- □ actinoids
- □ metalloids
- □ nonmetals
- □ halogens
- □ noble gases
- □ unknown elements
- radioactive elements have masses in parenthesis

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	1.00794 1 H Hydrogen																	4.002602 2 He Helium
2	6.941 3 Li Lithium	9.012182 4 Be Beryllium											10.811 5 B Boron	12.0107 6 C Carbon	14.0067 7 N Nitrogen	15.9994 8 O Oxygen	18.998403 9 F Fluorine	20.1797 10 Ne Neon
3	22.98976 11 Na Sodium	24.3050 12 Mg Magnesium											26.98153 13 Al Aluminium	28.0855 14 Si Silicon	30.97696 15 P Phosphorus	32.065 16 S Sulfur	35.453 17 Cl Chlorine	39.948 18 Ar Argon
4	39.0983 19 K Potassium	40.078 20 Ca Calcium	44.95591 21 Sc Scandium	47.867 22 Ti Titanium	50.9415 23 V Vanadium	51.9962 24 Cr Chromium	54.93804 25 Mn Manganese	55.845 26 Fe Iron	58.93319 27 Co Cobalt	58.6934 28 Ni Nickel	63.546 29 Cu Copper	65.38 30 Zn Zinc	69.723 31 Ga Gallium	72.64 32 Ge Germanium	74.92160 33 As Arsenic	78.96 34 Se Selenium	79.904 35 Br Bromine	83.798 36 Kr Krypton
5	85.4678 37 Rb Rubidium	87.62 38 Sr Strontium	88.90585 39 Y Yttrium	91.224 40 Zr Zirconium	92.90638 41 Nb Niobium	95.96 42 Mo Molybdenum	(98) 43 Tc Technetium	101.07 44 Ru Ruthenium	102.9055 45 Rh Rhodium	106.42 46 Pd Palladium	107.8682 47 Ag Silver	112.441 48 Cd Cadmium	114.818 49 In Indium	118.710 50 Sn Tin	121.760 51 Sb Antimony	127.60 52 Te Tellurium	126.9044 53 I Iodine	131.293 54 Xe Xenon
6	132.9054 55 Cs Caesium	137.327 56 Ba Barium	174.9668 71 Lu Lutetium	178.49 72 Hf Hafnium	180.9478 73 Ta Tantalum	183.84 74 W Tungsten	186.207 75 Re Rhenium	190.23 76 Os Osmium	192.217 77 Ir Iridium	195.084 78 Pt Platinum	196.9665 79 Au Gold	200.59 80 Hg Mercury	204.3833 81 Tl Thallium	207.2 82 Pb Lead	208.9804 83 Bi Bismuth	(210) 84 Po Polonium	(210) 85 At Astatine	(220) 86 Rn Radon
7	(223) 87 Fr Francium	(226) 88 Ra Radium	(262) 103 Lr Lawrencium	(261) 104 Rf Rutherfordium	(262) 105 Db Dubnium	(266) 106 Sg Seaborgium	(264) 107 Bh Bohrium	(277) 108 Hs Hassium	(268) 109 Mt Meitnerium	(271) 110 Ds Darmstadium	(272) 111 Rg Roentgenium	(285) 112 Cn Copernicium	(284) 113 Uut Ununtrium	(289) 114 Uuq Ununquadium	(288) 115 Uup Ununpentium	(292) 116 Uuh Ununhexium	117 Uus Ununseptium	(294) 118 Uuo Ununoctium

138.9054 57 La Lanthanum	140.116 58 Ce Cerium	140.9076 59 Pr Praseodymium	144.242 60 Nd Neodymium	(145) 61 Pm Promethium	150.36 62 Sm Samarium	151.964 63 Eu Europium	157.25 64 Gd Gadolinium	158.9253 65 Tb Terbium	162.500 66 Dy Dysprosium	164.9303 67 Ho Holmium	167.259 68 Er Erbium	168.9342 69 Tm Thulium	173.054 70 Yb Ytterbium
(227) 89 Ac Actinium	232.0380 90 Th Thorium	231.0358 91 Pa Protactinium	238.0289 92 U Uranium	(237) 93 Np Neptunium	(244) 94 Pu Plutonium	(243) 95 Am Americium	(247) 96 Cm Curium	(247) 97 Bk Berkelium	(251) 98 Cf Californium	(252) 99 Es Einsteinium	(257) 100 Fm Fermium	(258) 101 Md Mendelevium	(259) 102 No Nobelium

electron configuration blocks: s d p f

notes
- as of yet, elements 113-118 have no official name designated by the IUPAC.
- 1 kJ/mol ≈ 96.485 eV.
- all elements are implied to have an oxidation state of zero.

structures elements based on atomic number, electron configuration, and recurring chemical properties

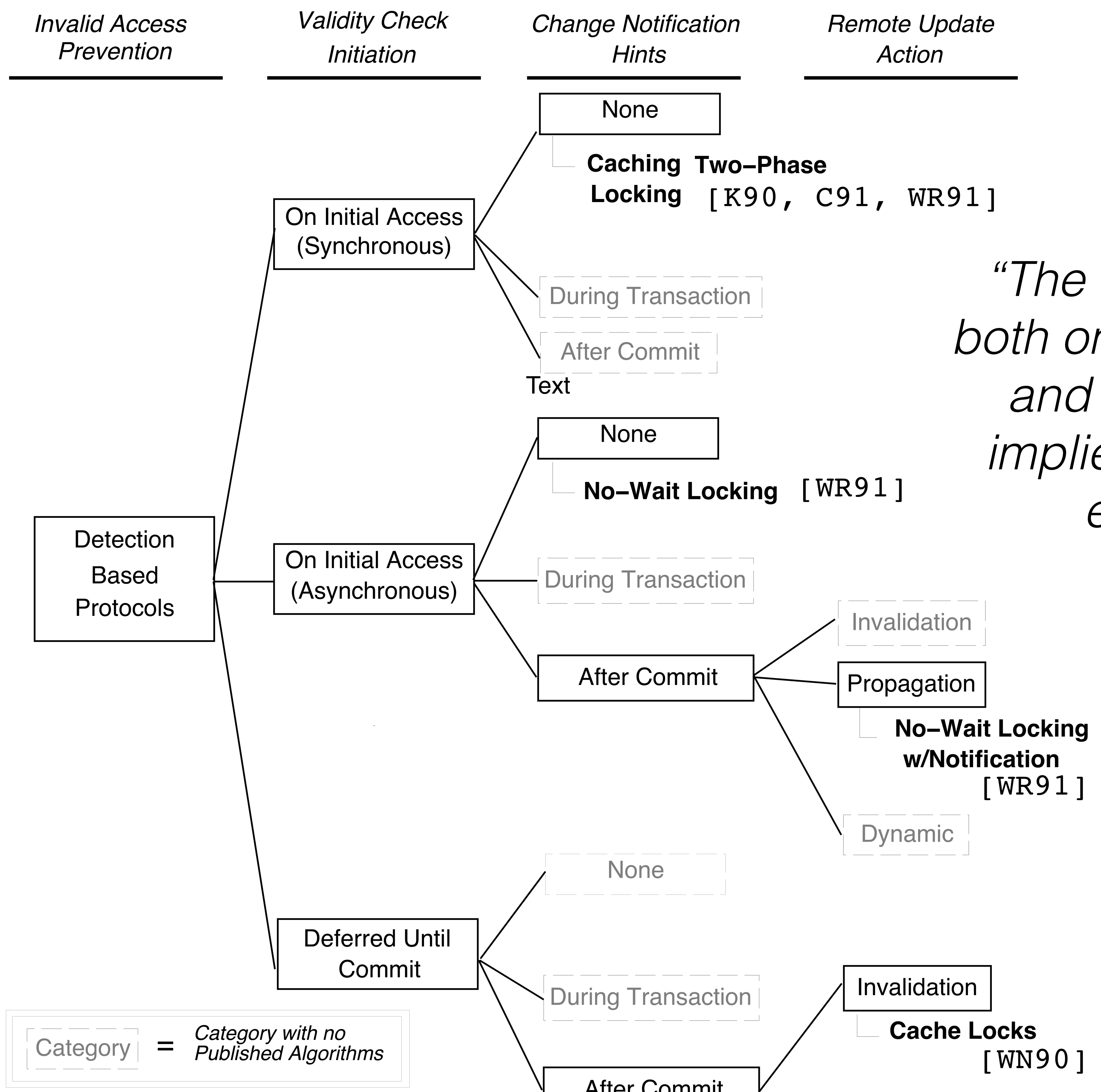
PERIODIC TABLE OF ELEMENTS

explains and predicts missing elements

Dimitri Mentelev

nihonium

Kosuke Morita



Invalid Access Prevention | *Validity Check Initiation* | *Change Notification Hints* | *Remote Update Action*

Detection Based Protocols

- On Initial Access (Synchronous)
  - None
    - **Caching Two–Phase Locking** [K90, C91, WR91]
  - During Transaction
  - After Commit
- Text
- On Initial Access (Asynchronous)
  - None
    - **No–Wait Locking** [WR91]
  - During Transaction
  - After Commit
    - Invalidation
    - Propagation
      - **No–Wait Locking w/Notification** [WR91]
    - Dynamic
- Deferred Until Commit
  - None
  - During Transaction
  - After Commit
    - Invalidation
      - **Cache Locks** [WN90]
    - Propagation
    - Dynamic
      - **Adaptive Optimistic** [A95]

Category = *Category with no Published Algorithms*

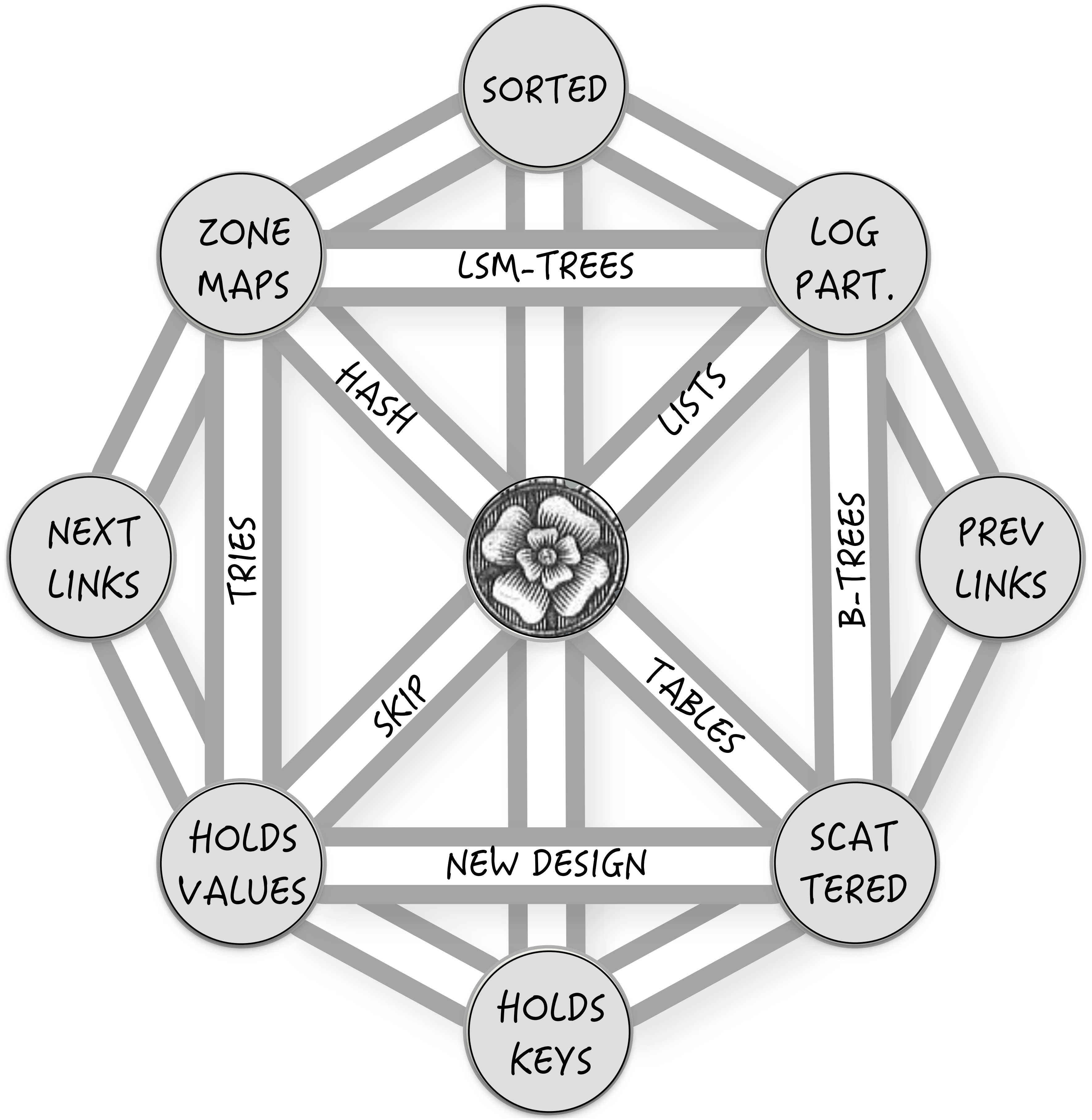
```
A95 – [Adya et al. 1995]
K90 – [Kim et al. 1990]
C91 – [Carey et al. 1991]
WR91 – [Wang and Rowe 1991]
WN90 – [Wilkinson and Neimat 1990]
```

“The taxonomy is used to shed light both on the nature of the design space and on the performance tradeoffs implied by many of the choices that exist in the design space.”

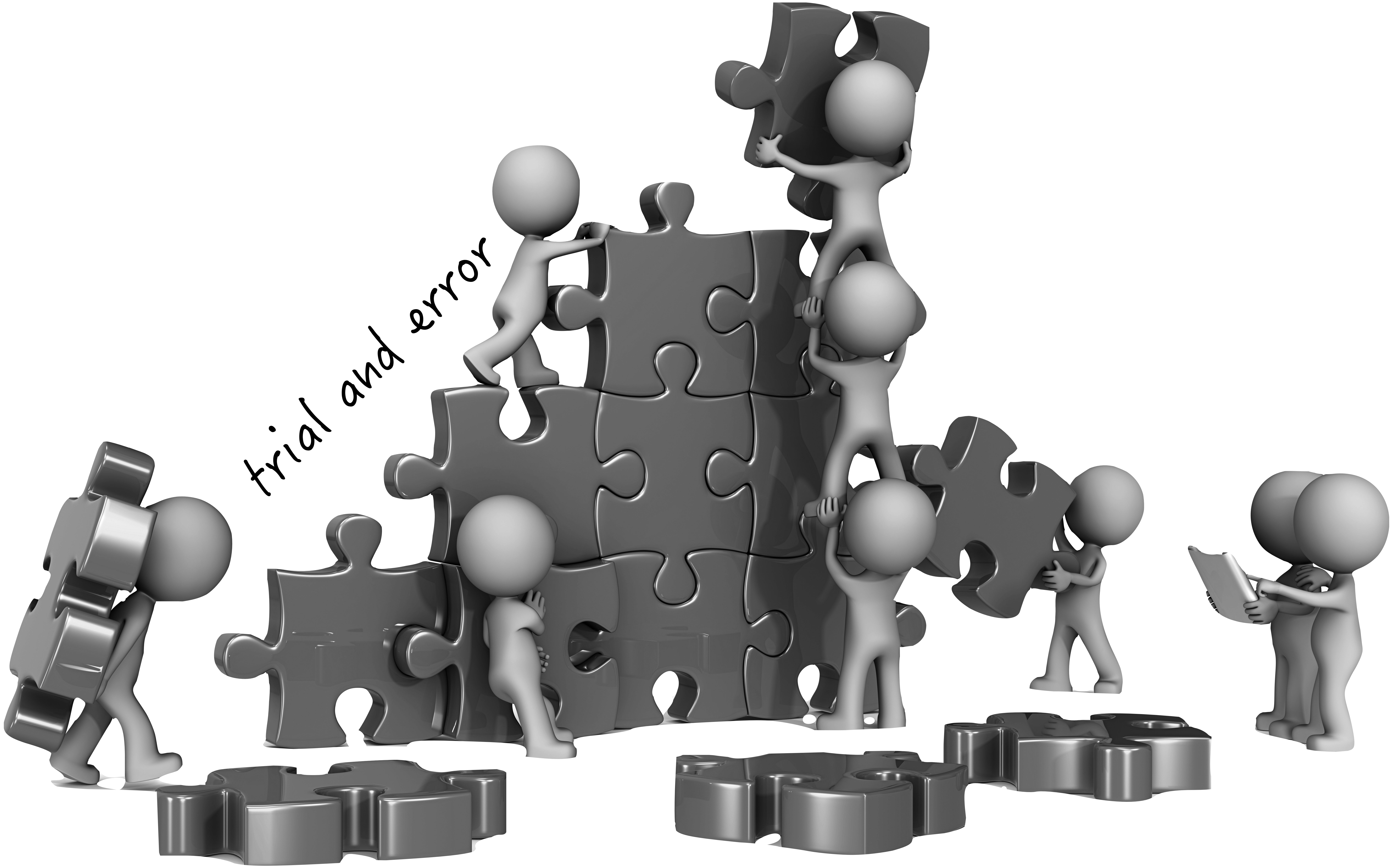
TAXONOMY OF COMPLEX ALGORITHMS

transactional cache consistency maintenance

Mike Franklin

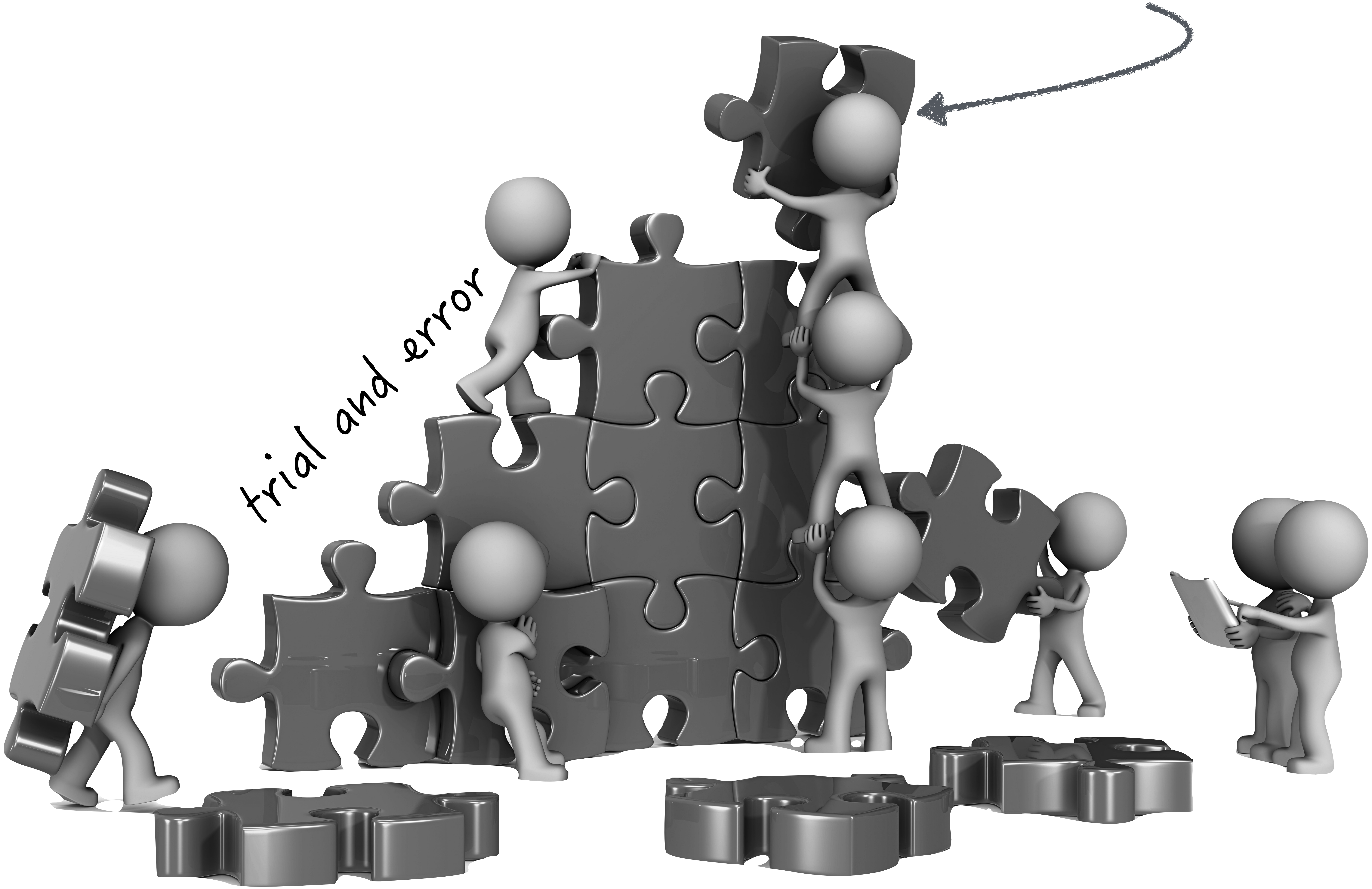


DASlab @ Harvard SEAS



DASlab @ Harvard SEAS

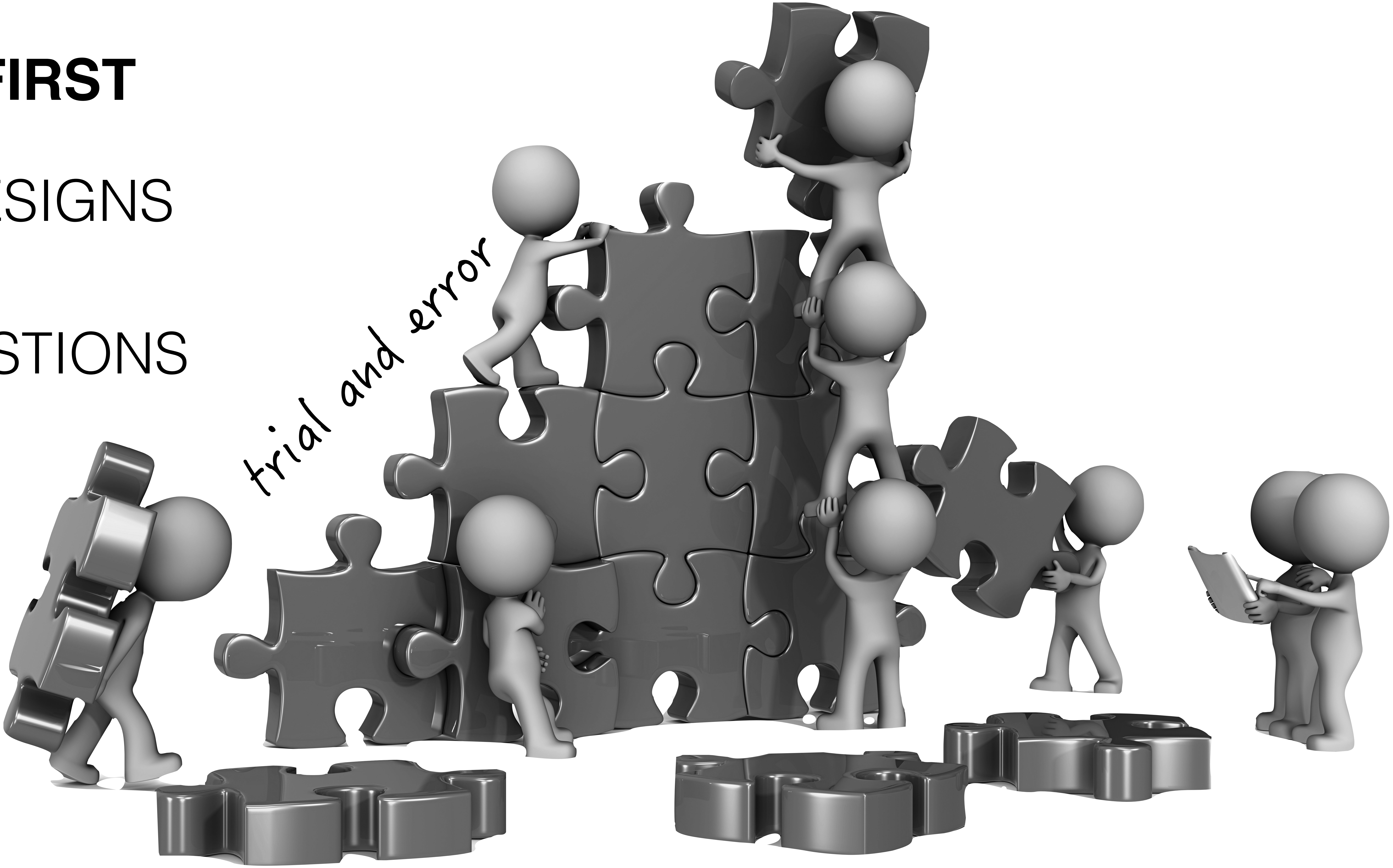
FIRST PRINCIPLE: DESIGN CONCEPT THAT IS NOT POSSIBLE OR MEANINGFUL TO BREAK FURTHER



MAP LAYOUT FIRST

1 KNOWN DESIGNS

2 OPEN QUESTIONS

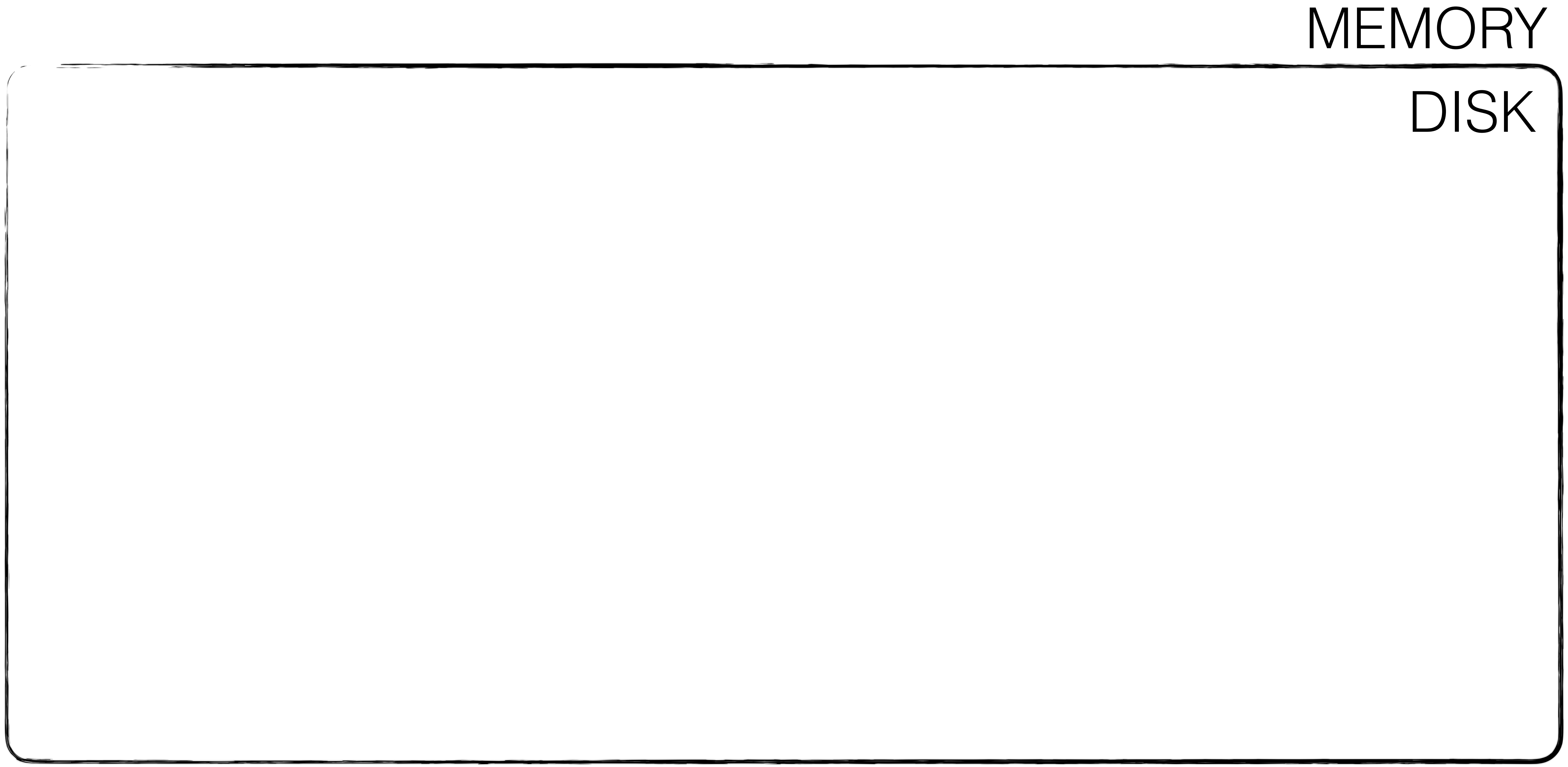


{arrays, logs, lsm-trees, b-trees}, filters, bitmaps, compression, stats

e.g., 1000x NoSQL k-v: bloom filter bits, merging policy
e.g., access path selection: scans vs b-tree depends on concurrency
e.g., robust scans with value by value lossy compression
e.g., updatable bitmap indexes
e.g., fast statistics/ML
. . .

EXAMPLE: The design space of **NoSQL Key-value Stores**

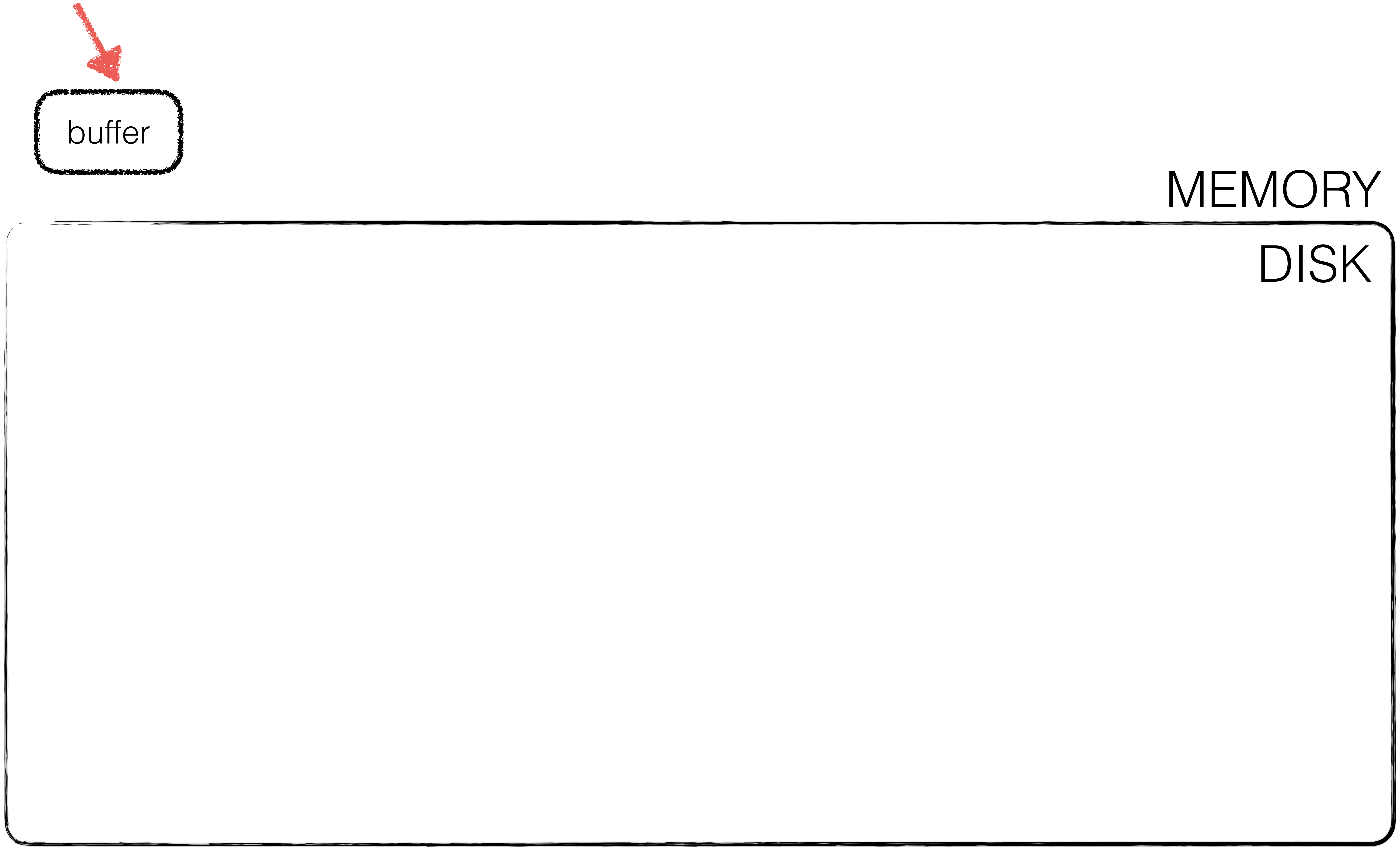
Google
BigTable
RocksDB
mongoDB
in
Couchbase
amazon
DynamoDB
SQLite
APACHE
HBASE
cassandra



MEMORY

DISK

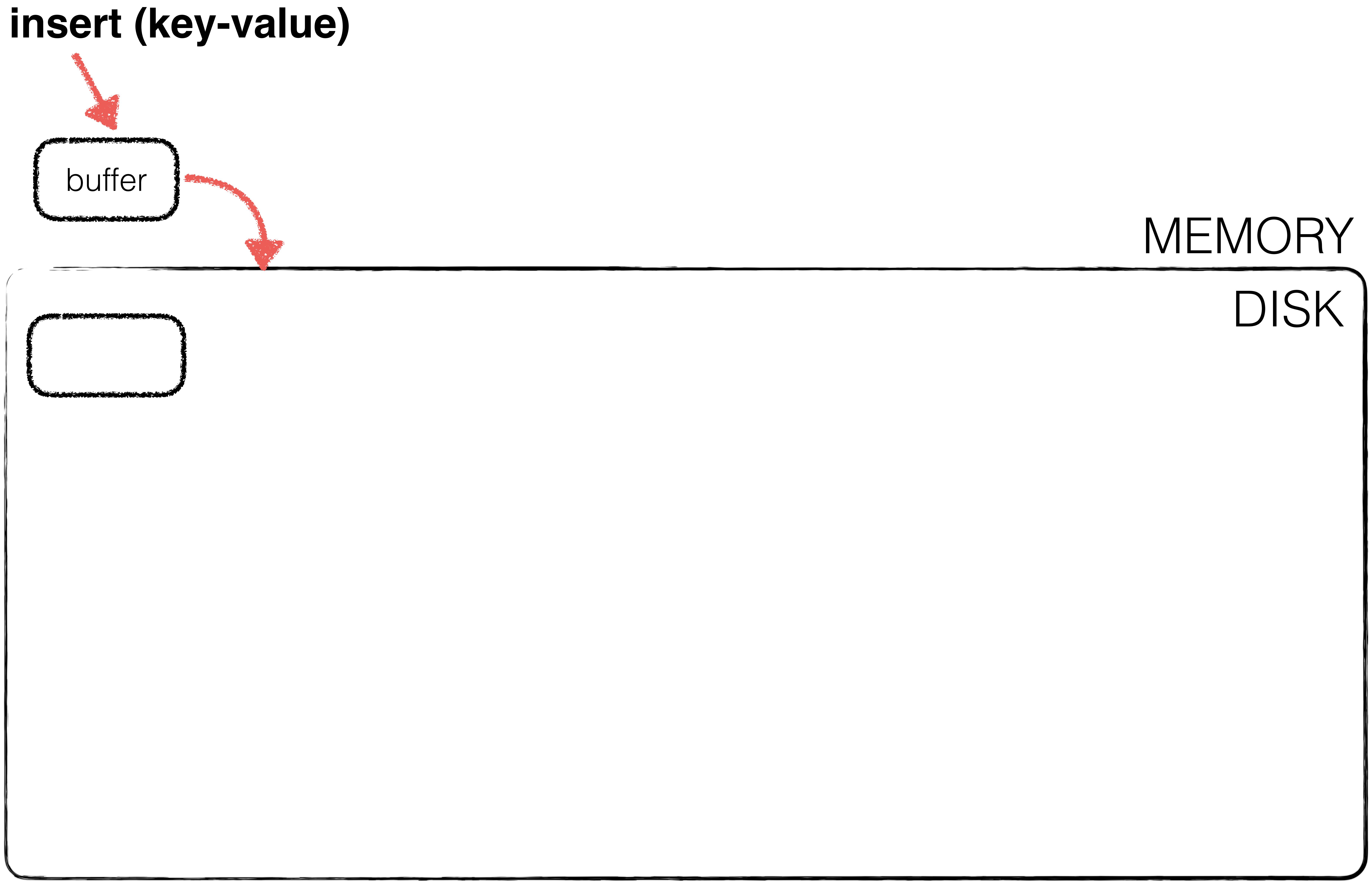
insert (key-value)



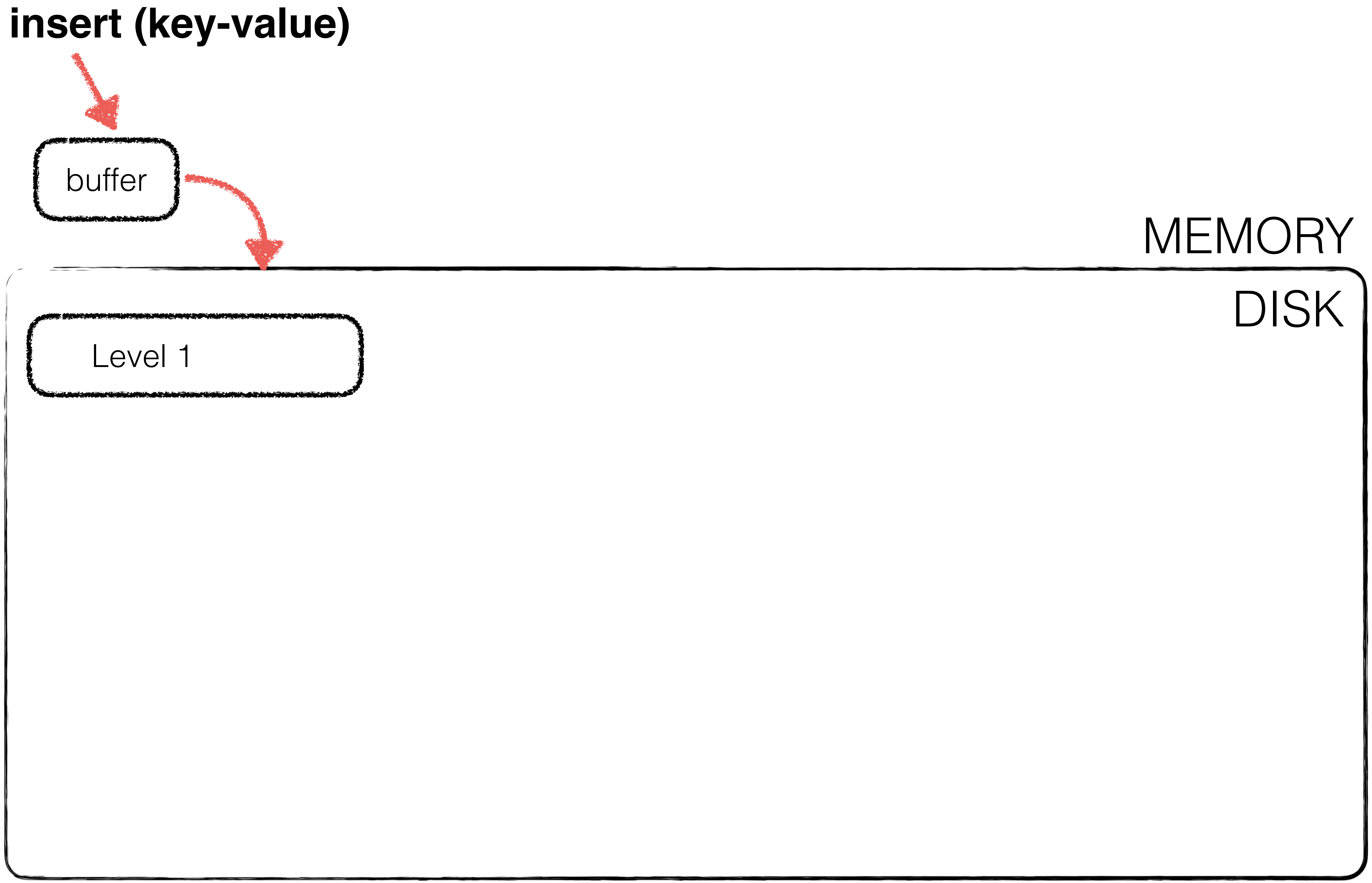
buffer

MEMORY

DISK



insert (key-value)
buffer
MEMORY
DISK



insert (key-value)

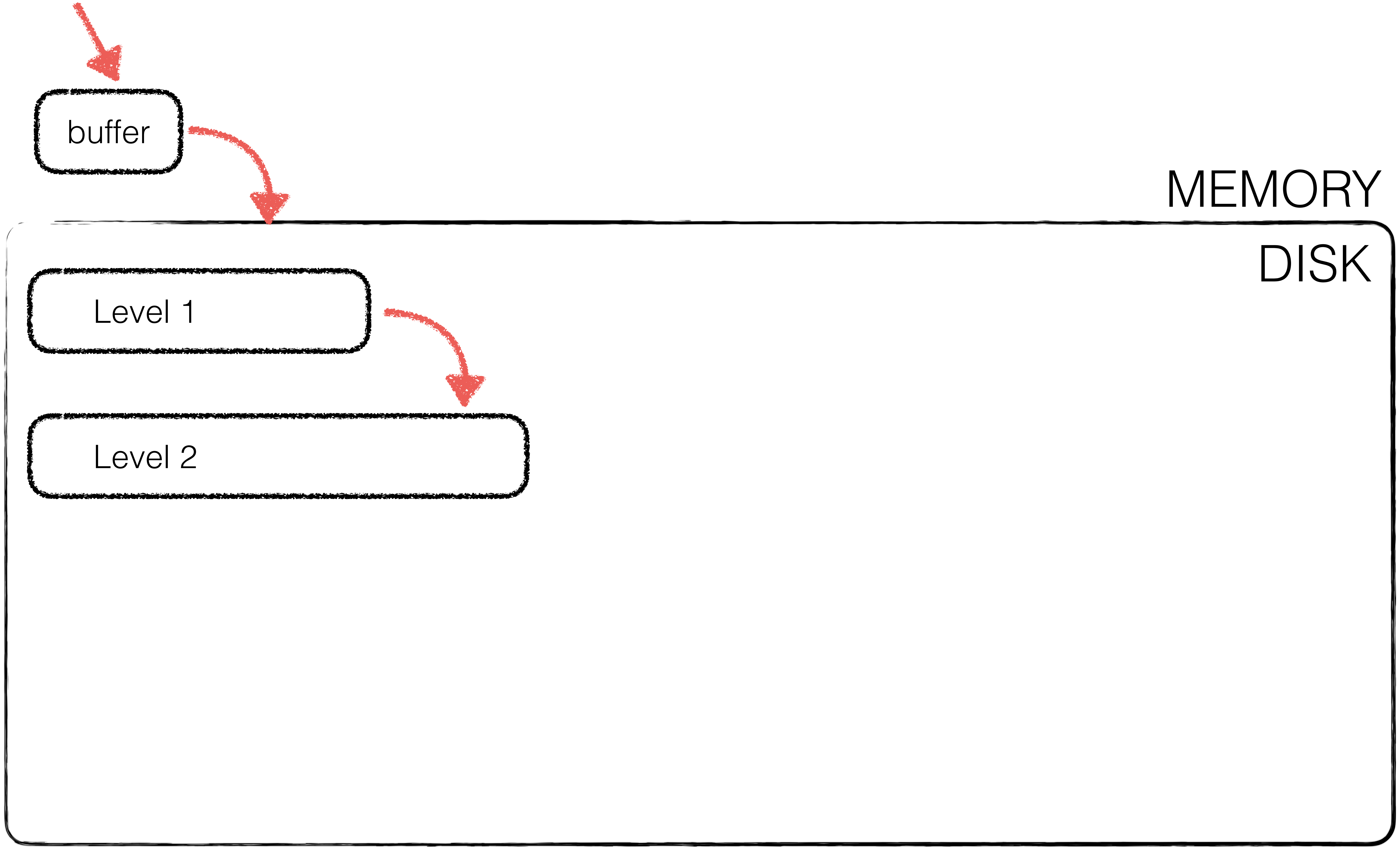
buffer

MEMORY

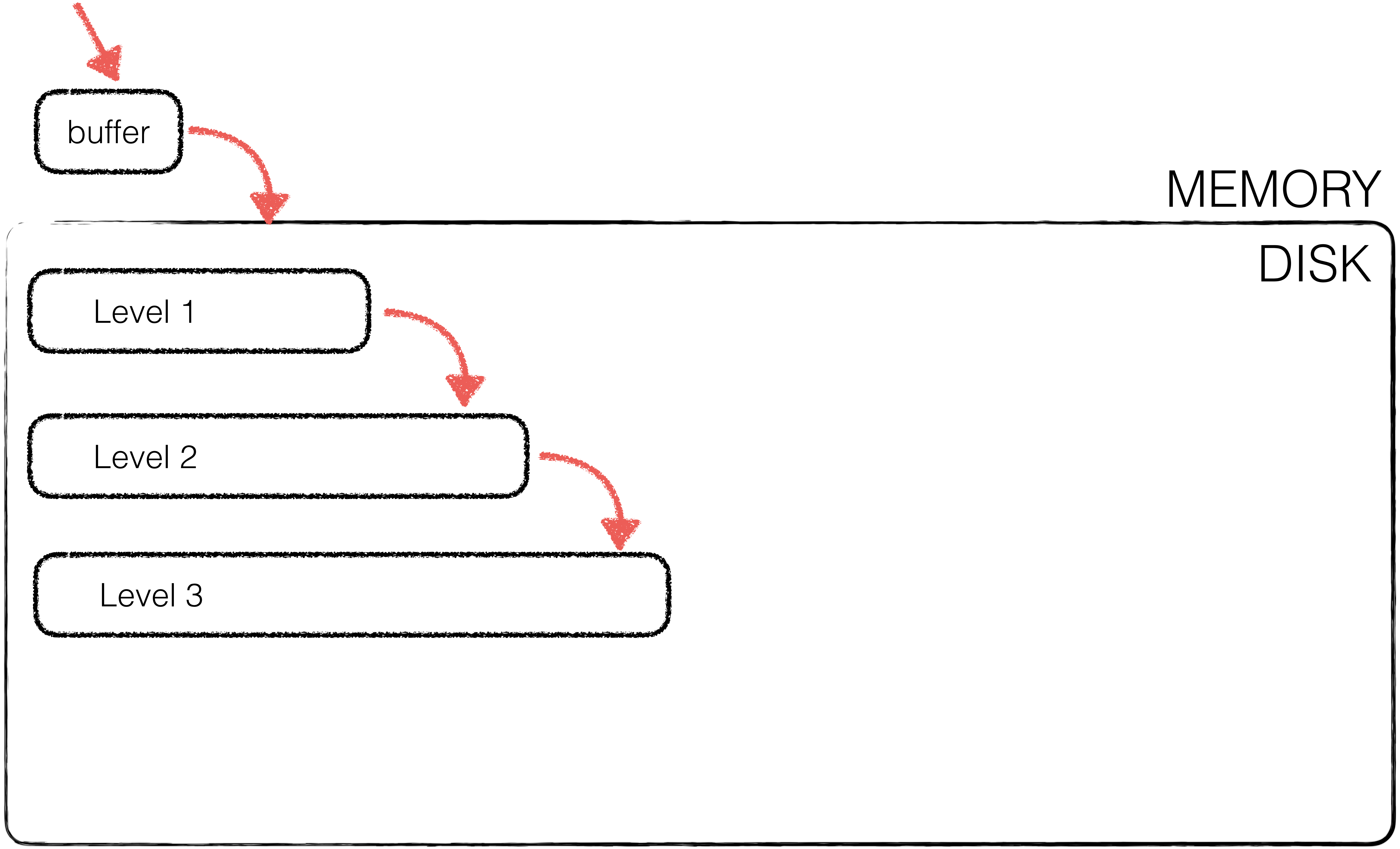
DISK

Level 1

insert (key-value)



insert (key-value)



buffer

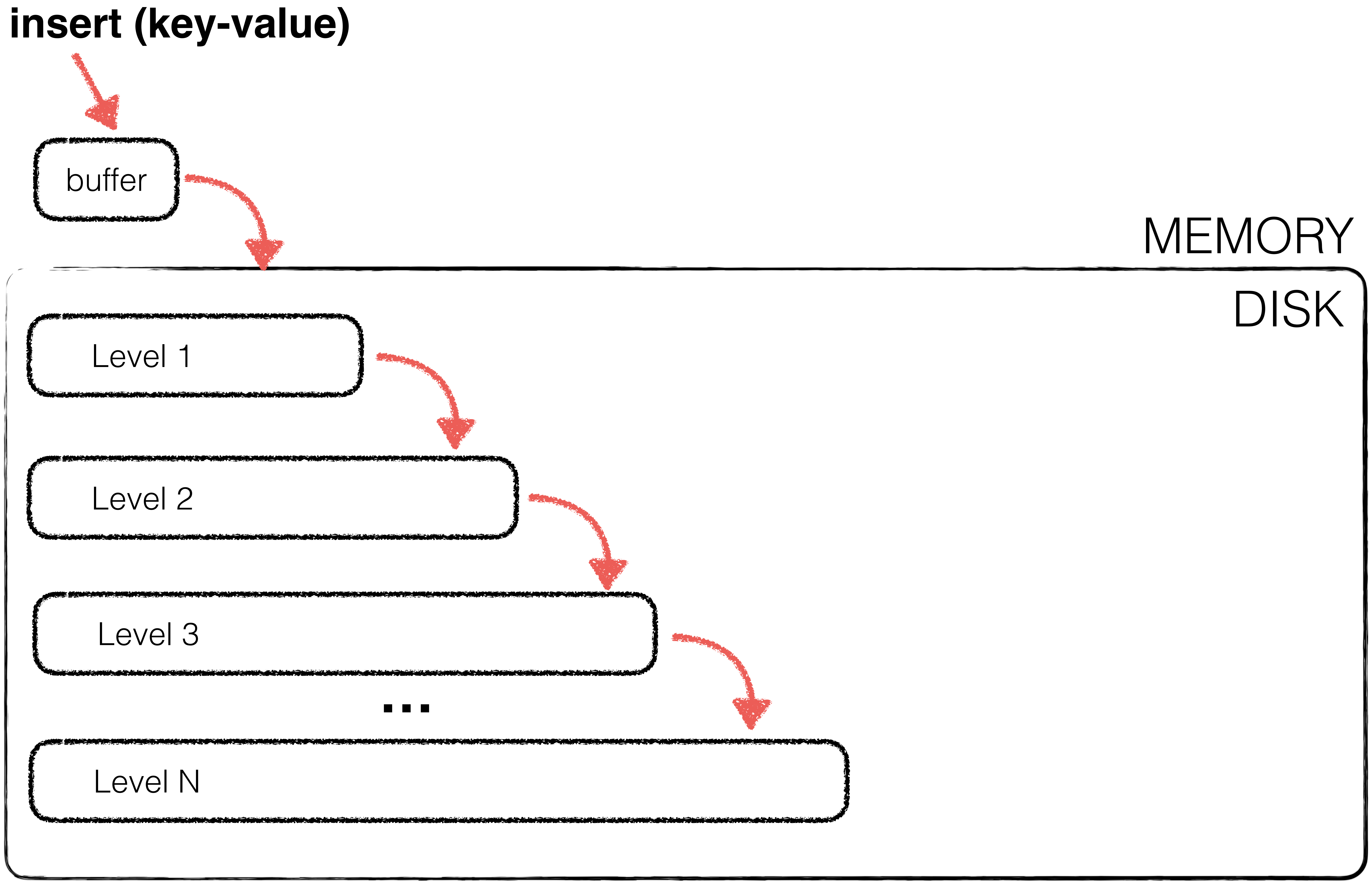
MEMORY

DISK

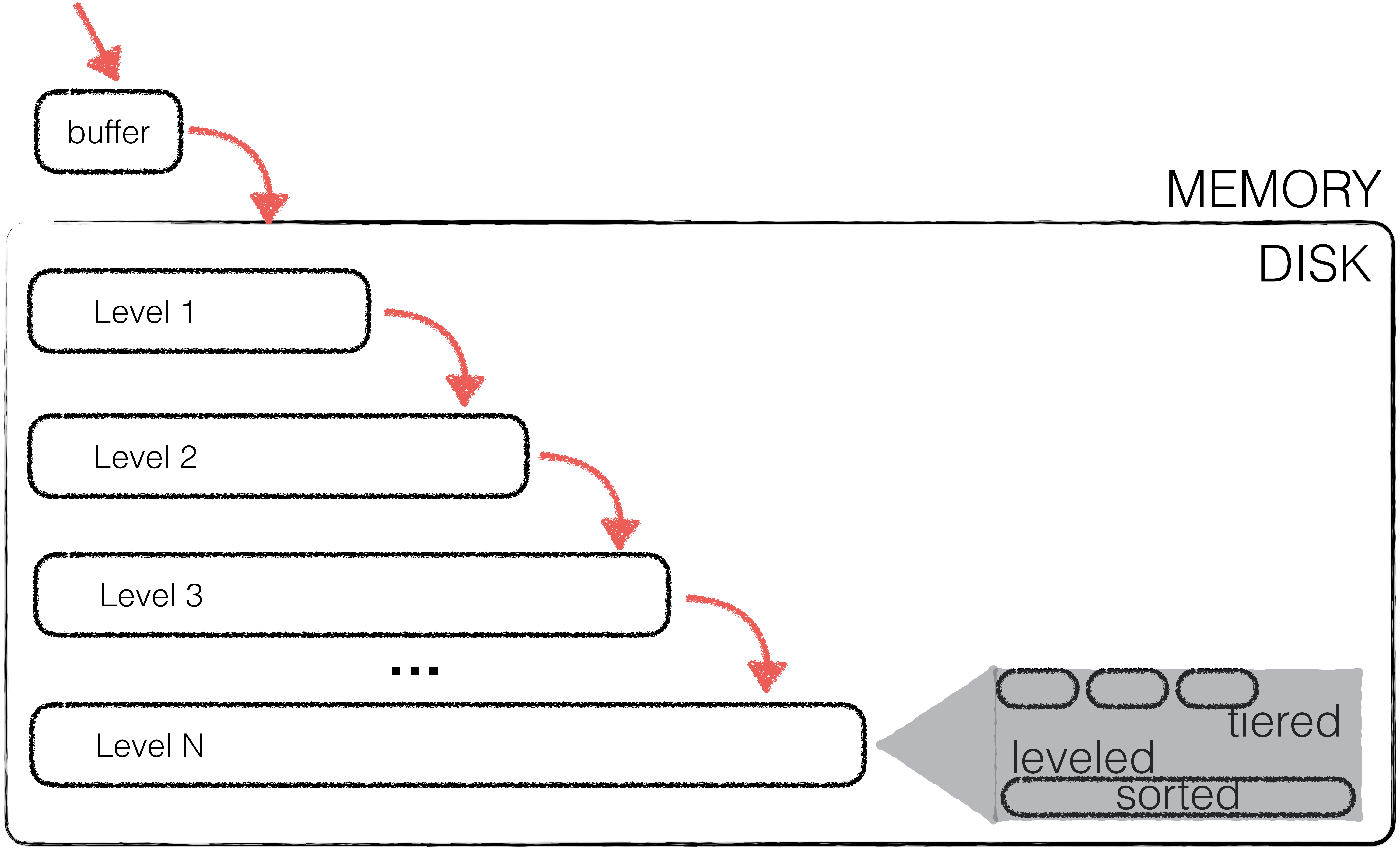
Level 1

Level 2

Level 3


insert (key-value)
buffer
MEMORY
DISK
Level 1
Level 2
Level 3
...
Level N

insert (key-value)



buffer

MEMORY

DISK

Level 1

Level 2

Level 3

...

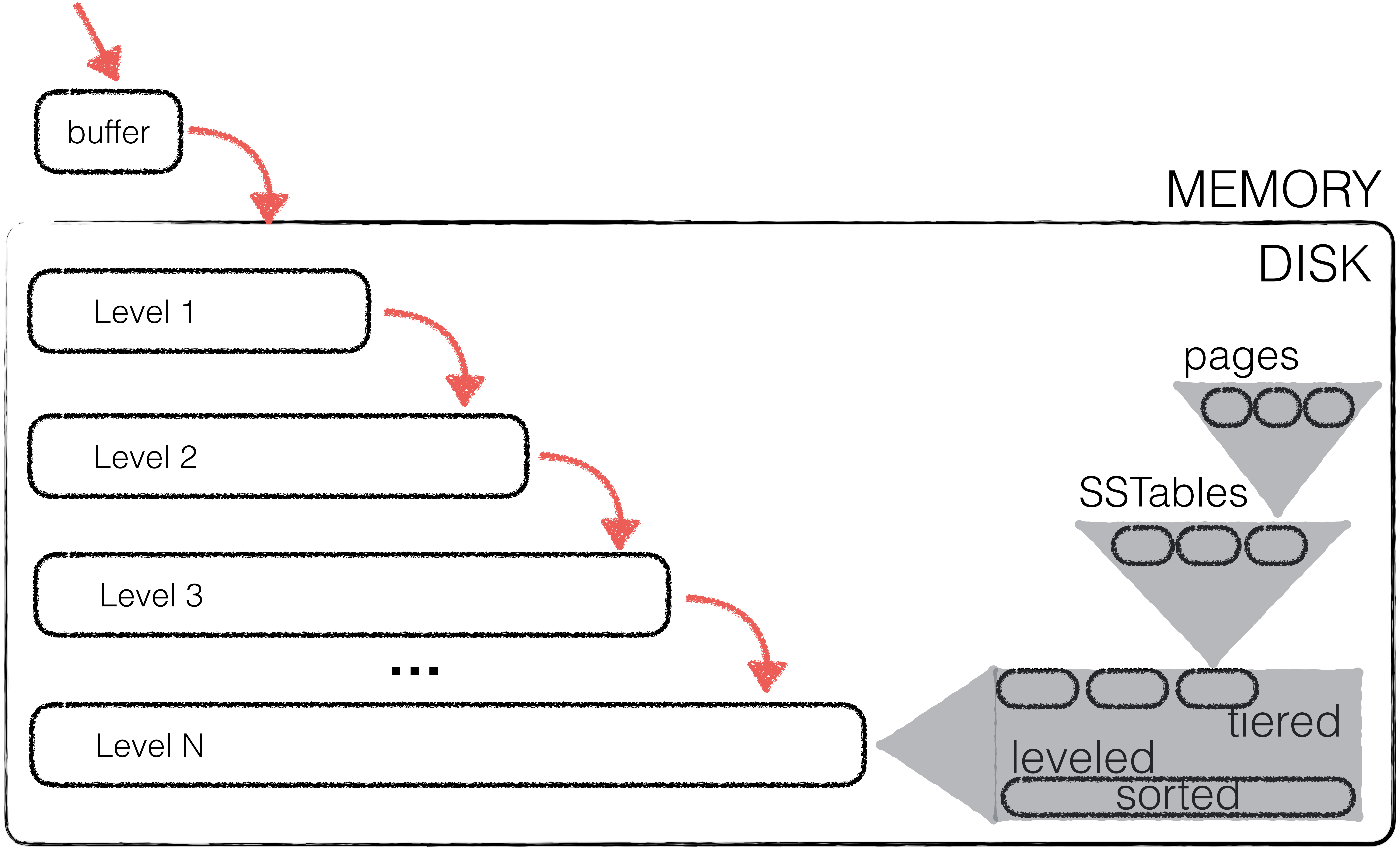
Level N

tiered

leveled

sorted

insert (key-value)



buffer

MEMORY

DISK

Level 1

Level 2

Level 3

...

Level N

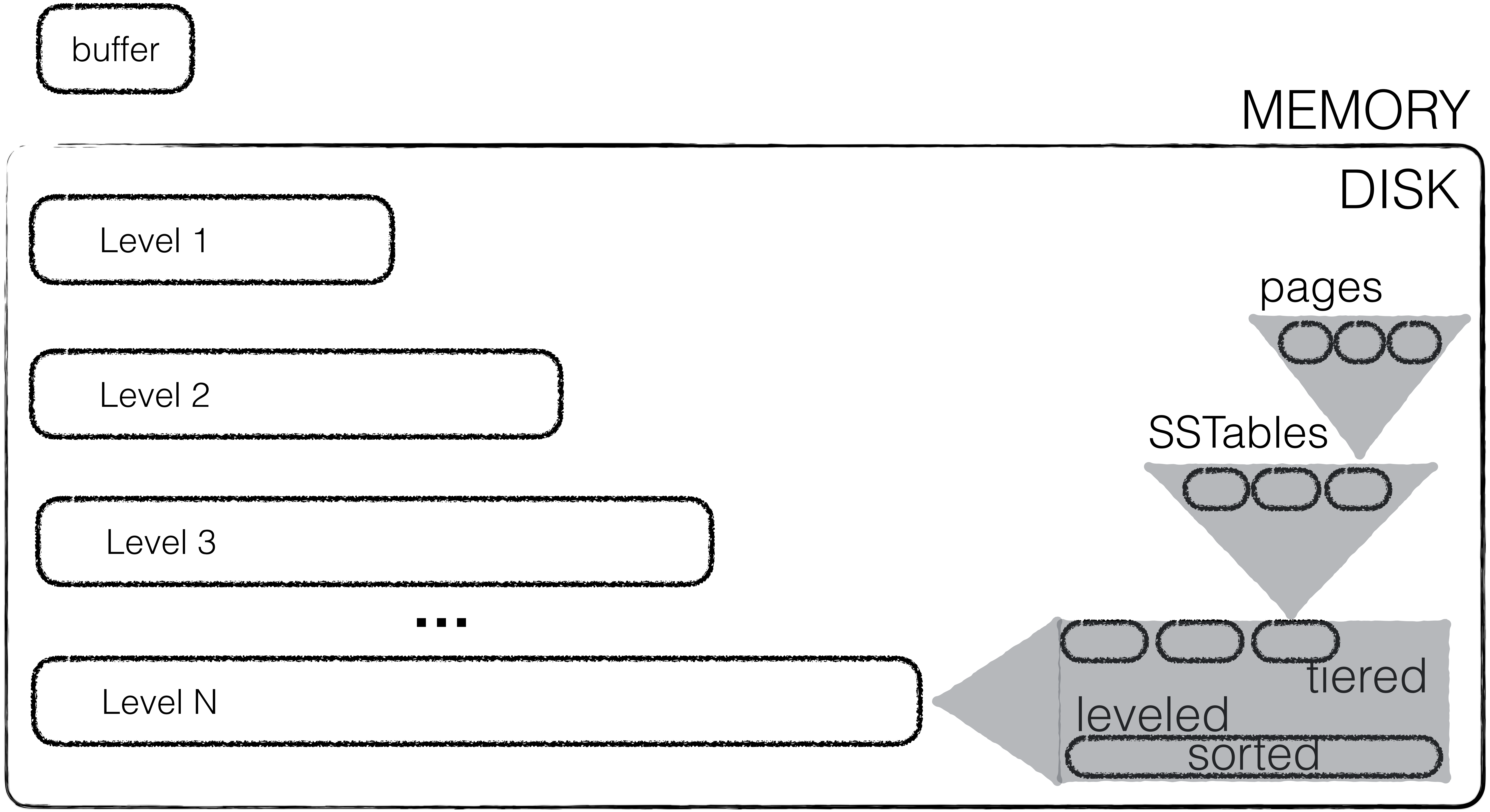
pages

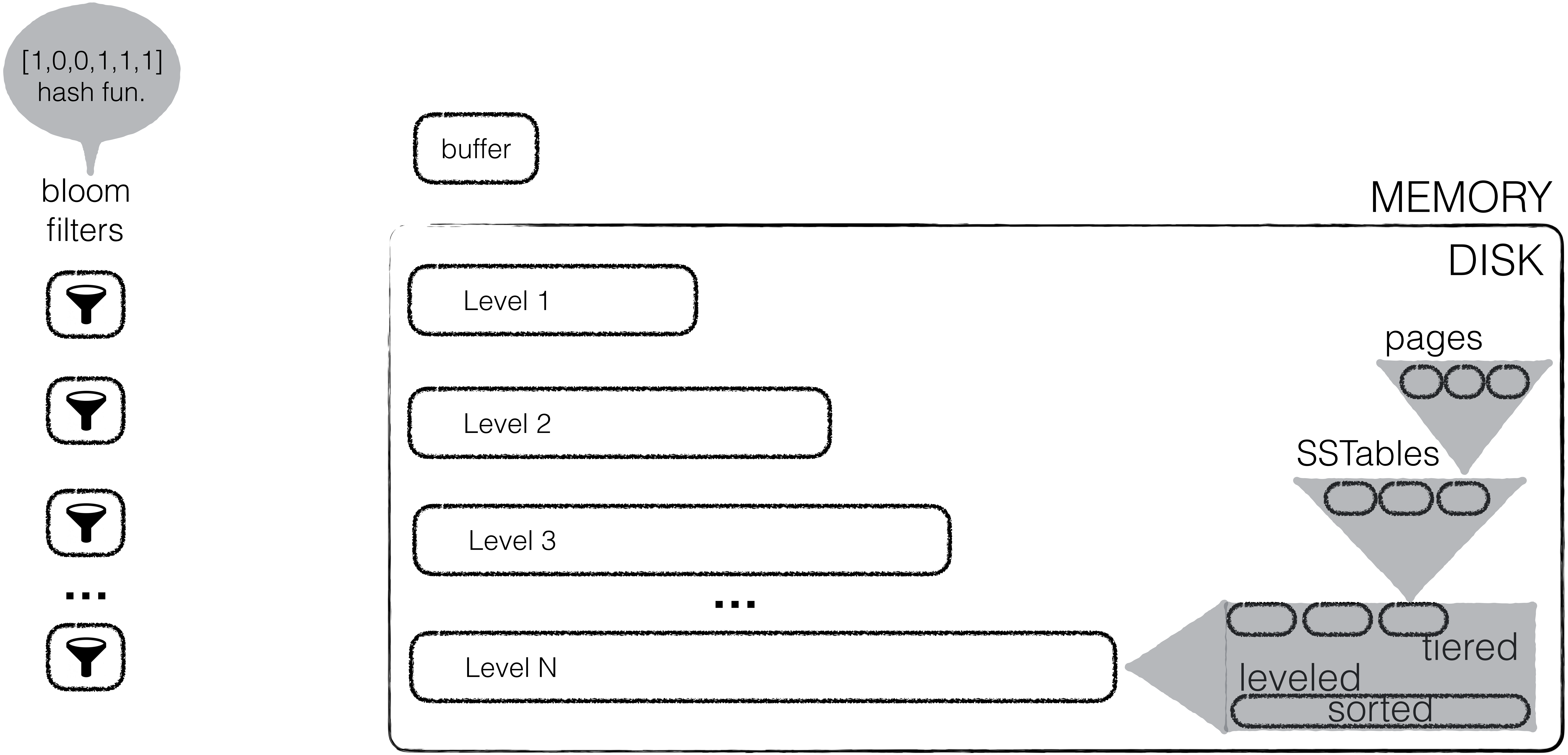
SSTables

tiered

leveled

sorted


buffer
MEMORY
DISK
Level 1
Level 2
Level 3
...
Level N
pages
SSTables
tiered
leveled
sorted



[1,0,0,1,1,1]
hash fun.

bloom
filters

...

buffer

MEMORY

DISK

Level 1

Level 2

Level 3

...

Level N

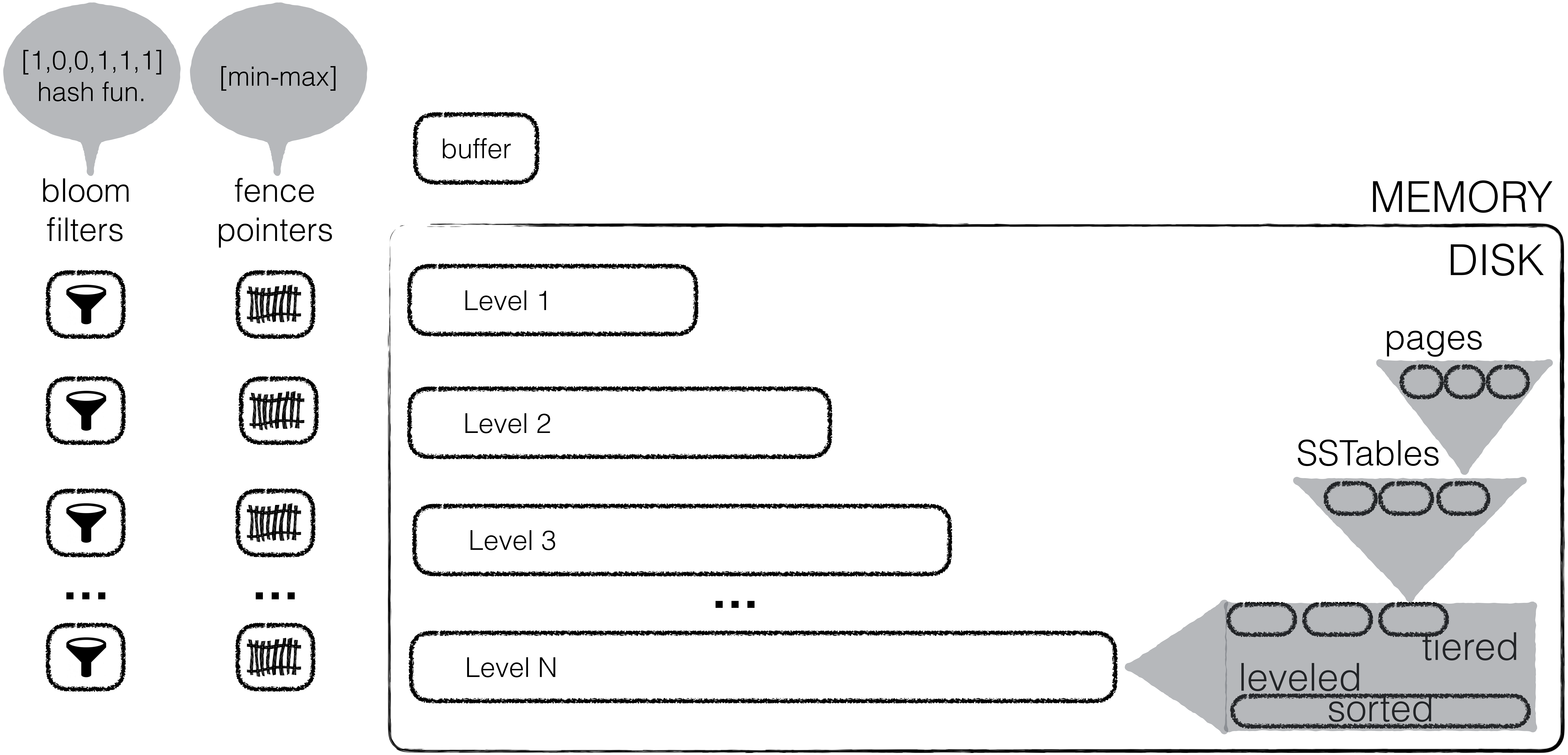
pages

SSTables

tiered

leveled

sorted



[1,0,0,1,1,1]
hash fun.

[min-max]

bloom filters

fence pointers

buffer

MEMORY

DISK

Level 1

Level 2

Level 3

...

Level N

...

...

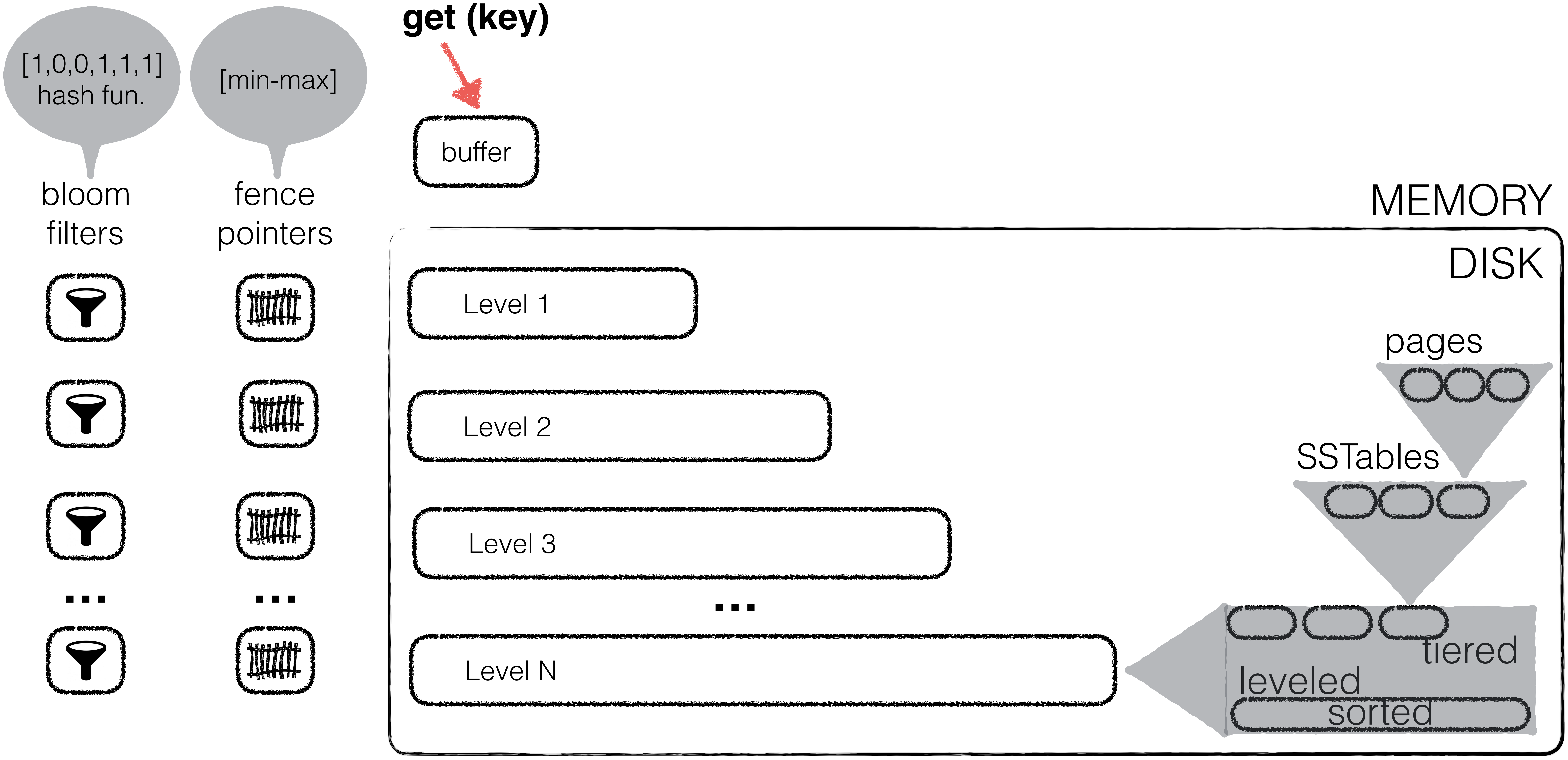
pages

SSTables

tiered

leveled

sorted



get (key)

buffer

MEMORY

DISK

[1,0,0,1,1,1]
hash fun.

[min-max]

bloom
filters

fence
pointers

...

...

Level 1

Level 2

Level 3

...

Level N

pages

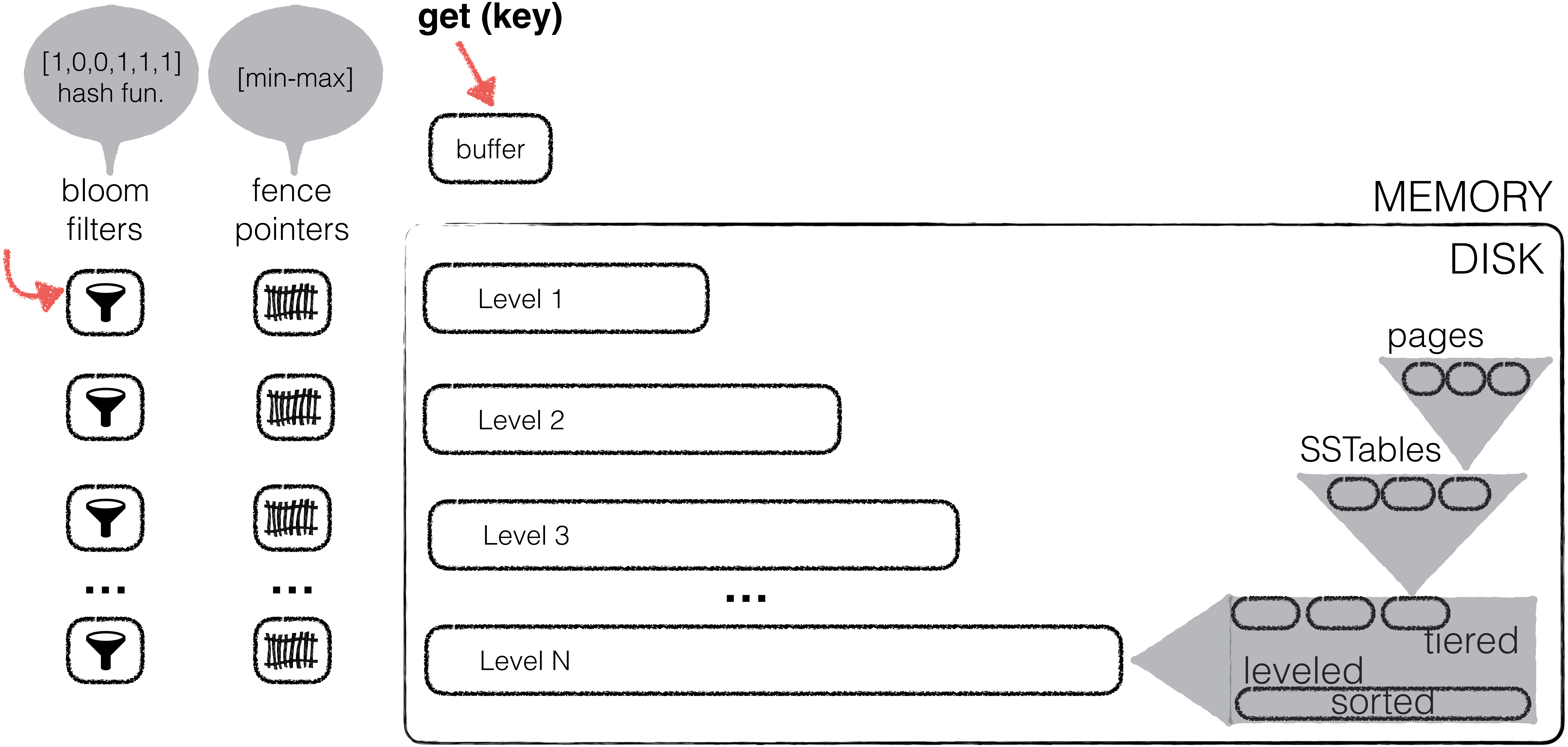
SSTables

tiered

leveled

sorted

get (key)



[1,0,0,1,1,1]
hash fun.

[min-max]

bloom
filters

fence
pointers

buffer

MEMORY

DISK

Level 1

Level 2

Level 3

...

Level N

pages

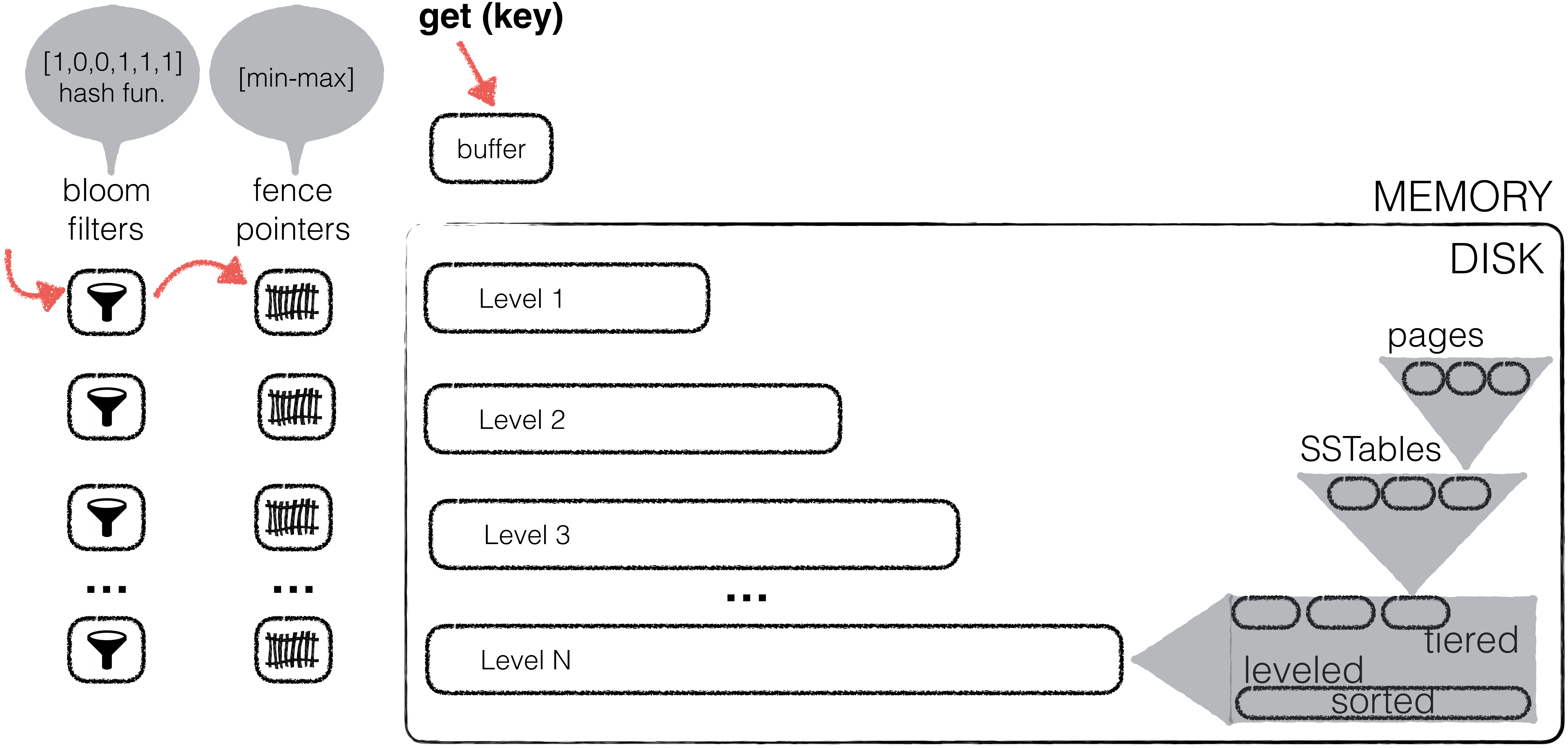
SSTables

tiered

leveled

sorted

get (key)



[1,0,0,1,1,1]
hash fun.

[min-max]

bloom filters

fence pointers

buffer

MEMORY

DISK

Level 1

Level 2

Level 3

...

Level N

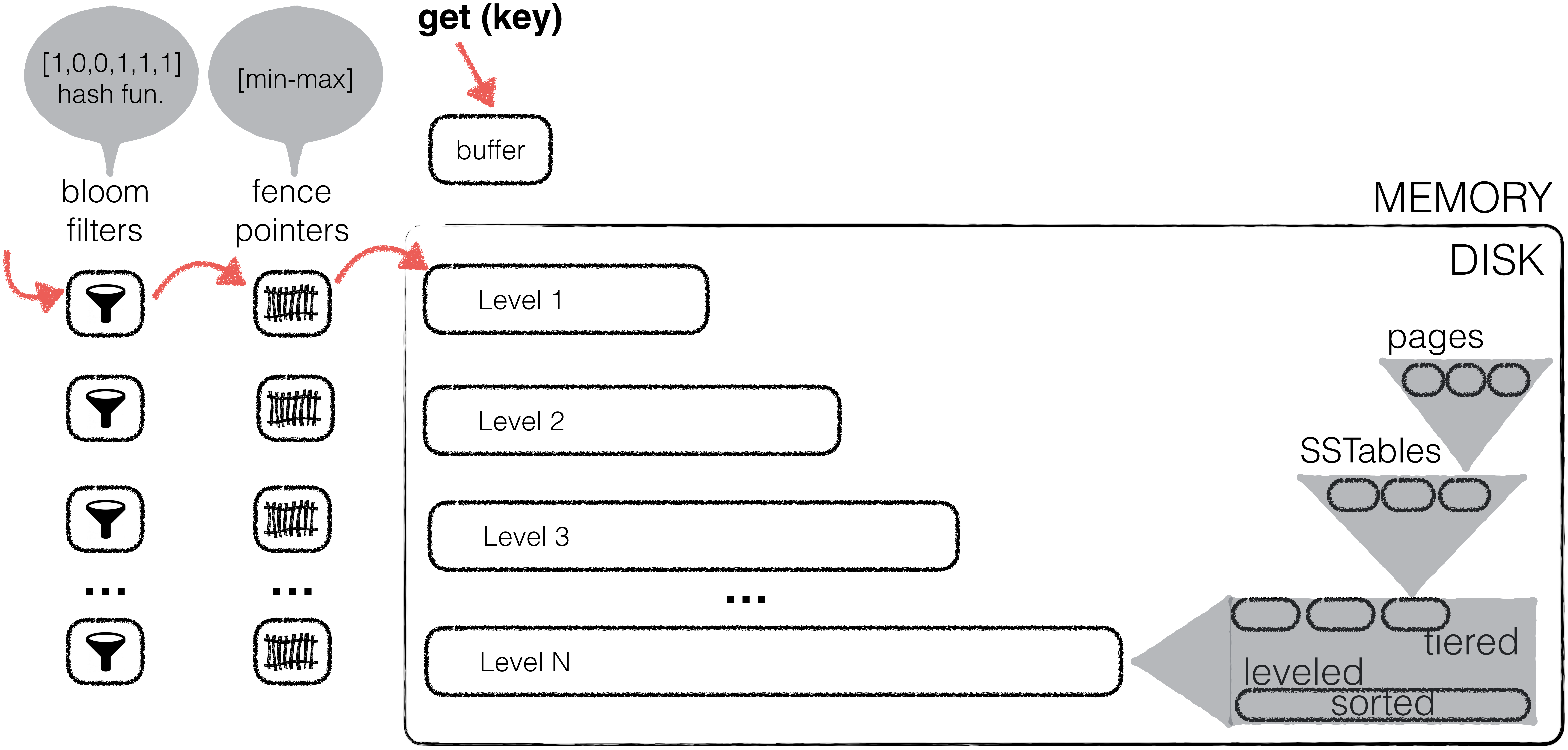
pages

SSTables

tiered

leveled

sorted



get (key)

[1,0,0,1,1,1]
hash fun.

[min-max]

buffer

bloom filters

fence pointers

MEMORY

DISK

Level 1

Level 2

Level 3

...

Level N

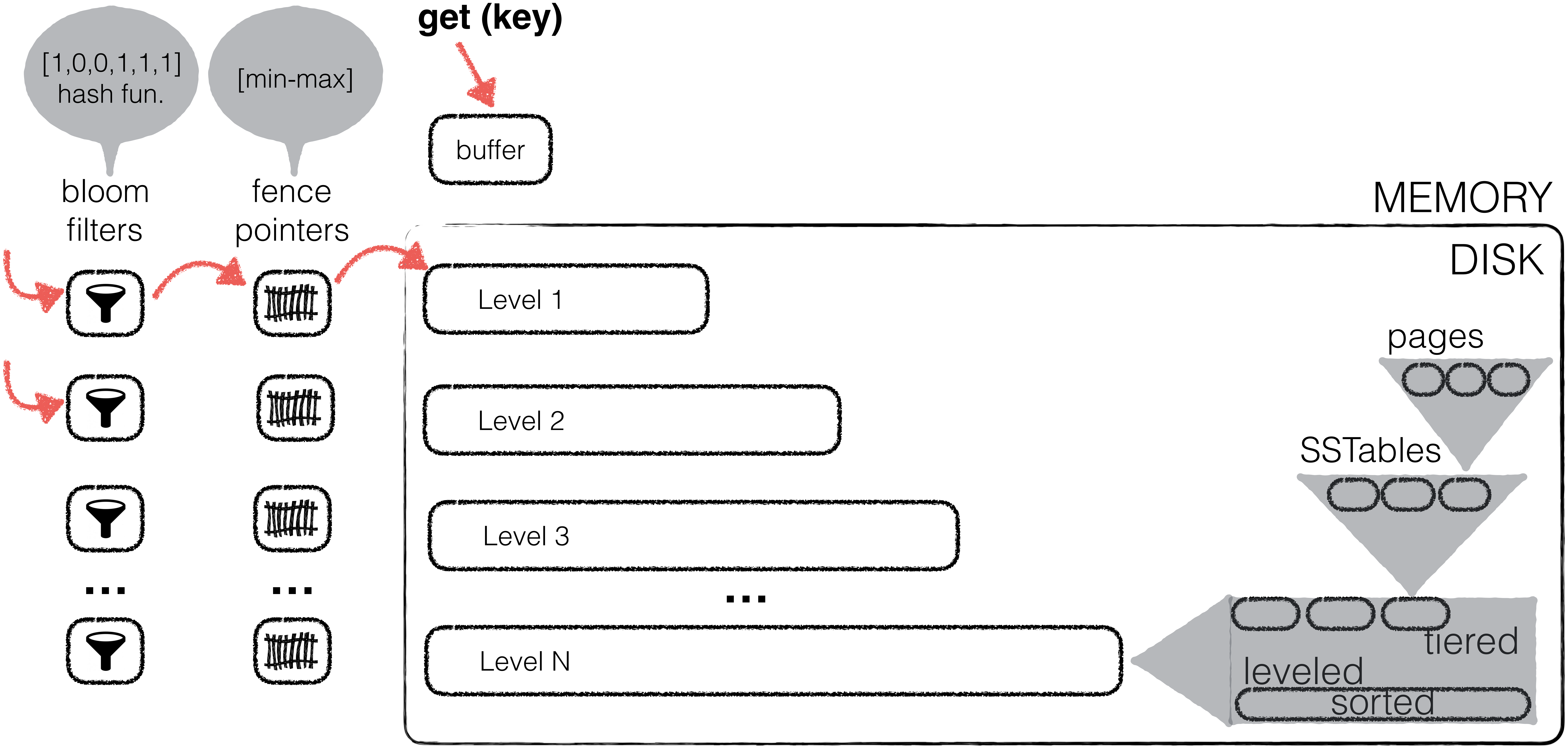
pages

SSTables

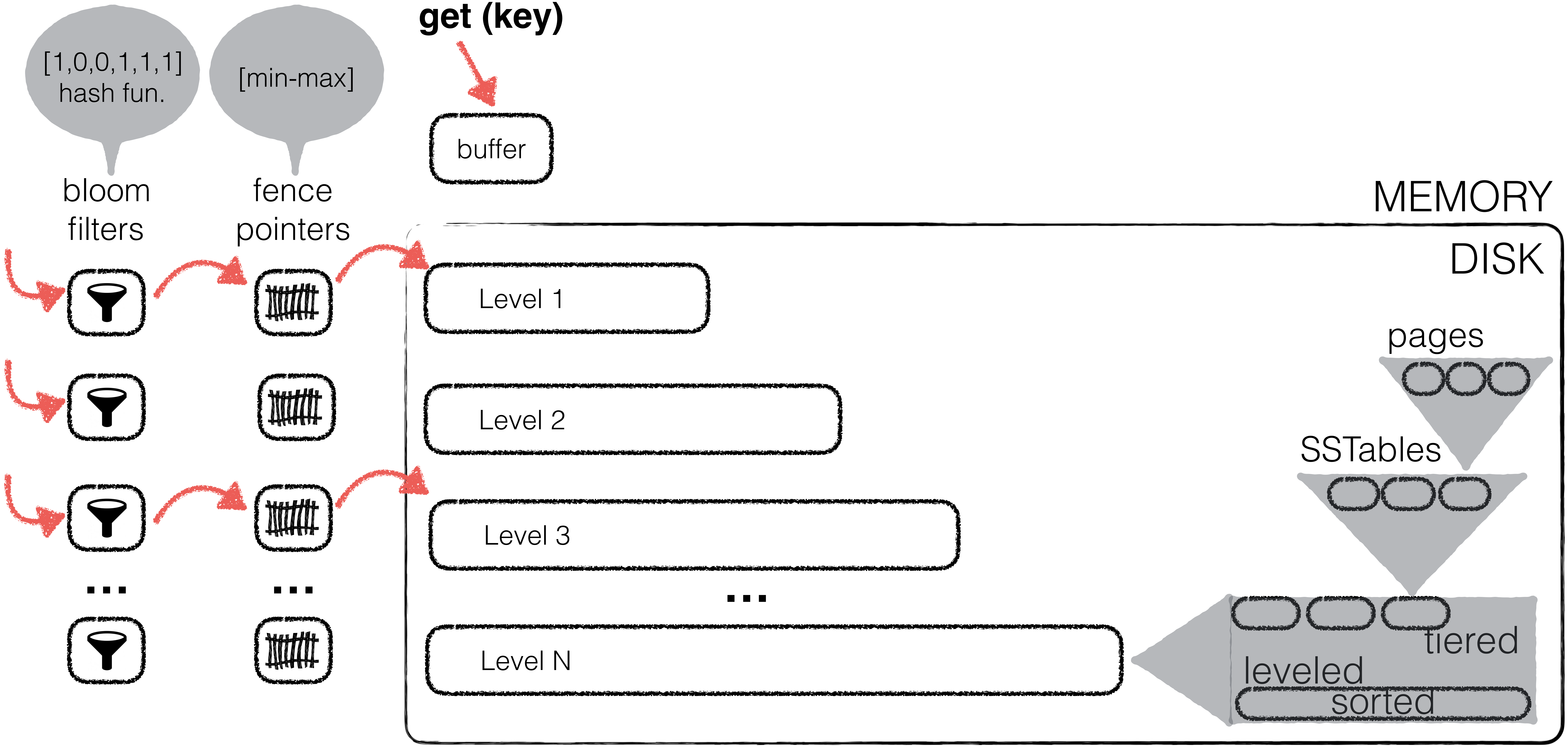
tiered

leveled

sorted


get (key)
[1,0,0,1,1,1]
hash fun.
[min-max]
buffer
bloom filters
fence pointers
MEMORY
DISK
Level 1
Level 2
Level 3
...
Level N
...
...
pages
SSTables
tiered
leveled
sorted

get (key)



[1,0,0,1,1,1]
hash fun.

[min-max]

buffer

bloom filters

fence pointers

MEMORY

DISK

Level 1

Level 2

Level 3

...

Level N

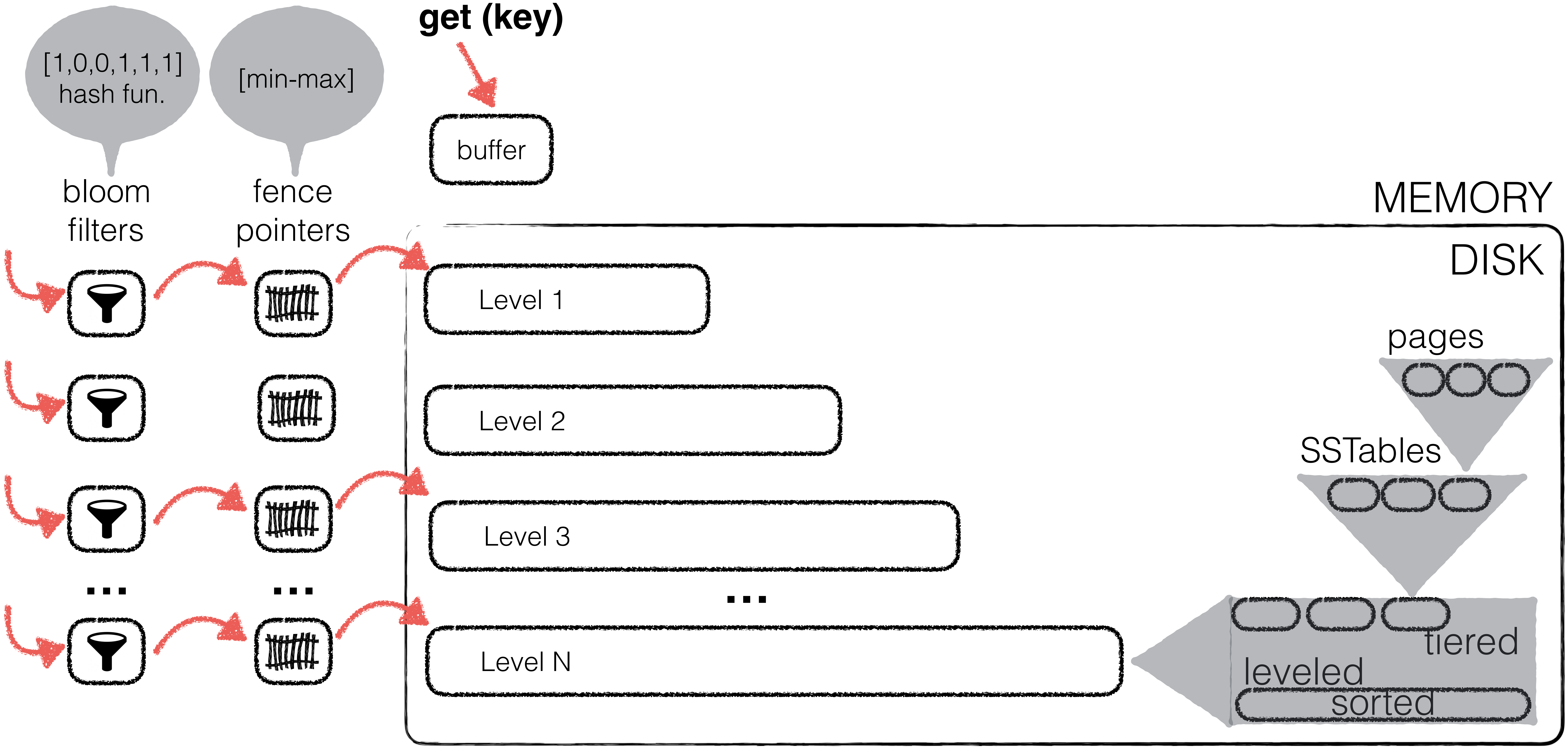
pages

SSTables

tiered

leveled

sorted



get (key)

[1,0,0,1,1,1]
hash fun.

[min-max]

buffer

bloom
filters

fence
pointers

MEMORY

DISK

Level 1

Level 2

Level 3

...

Level N

...

...

pages

SSTables

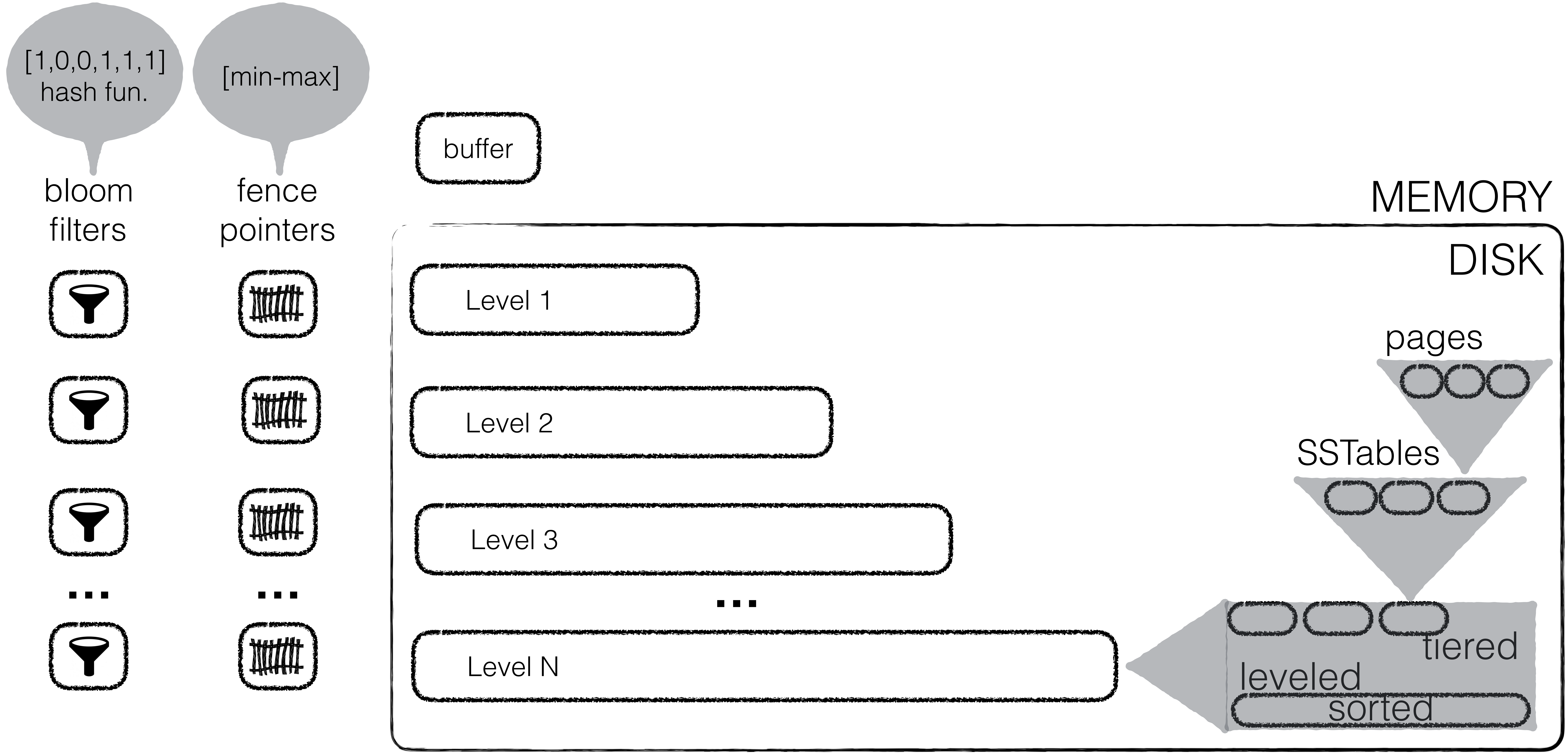
tiered

leveled

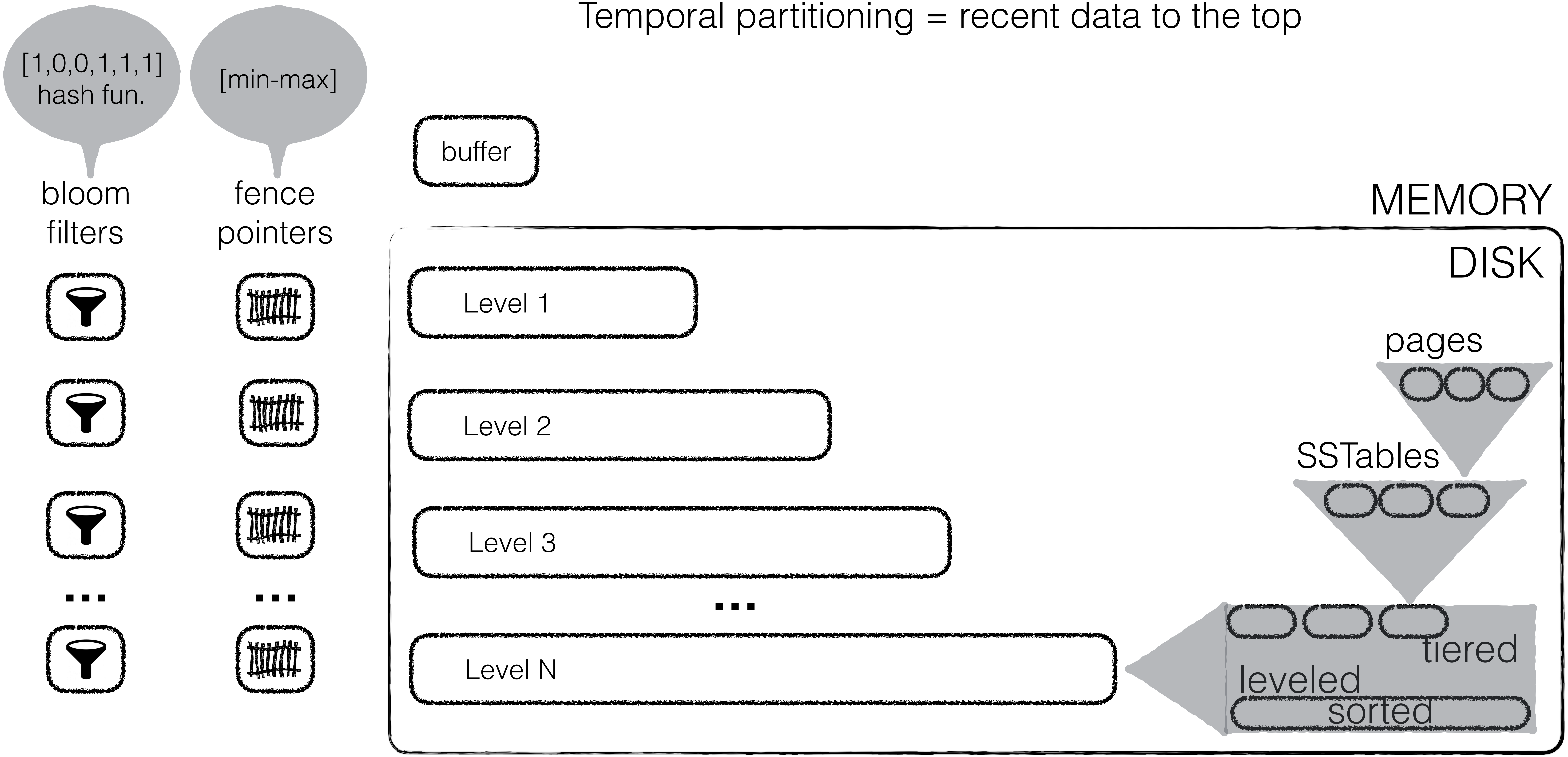
sorted

DASlab
@ Harvard SEAS

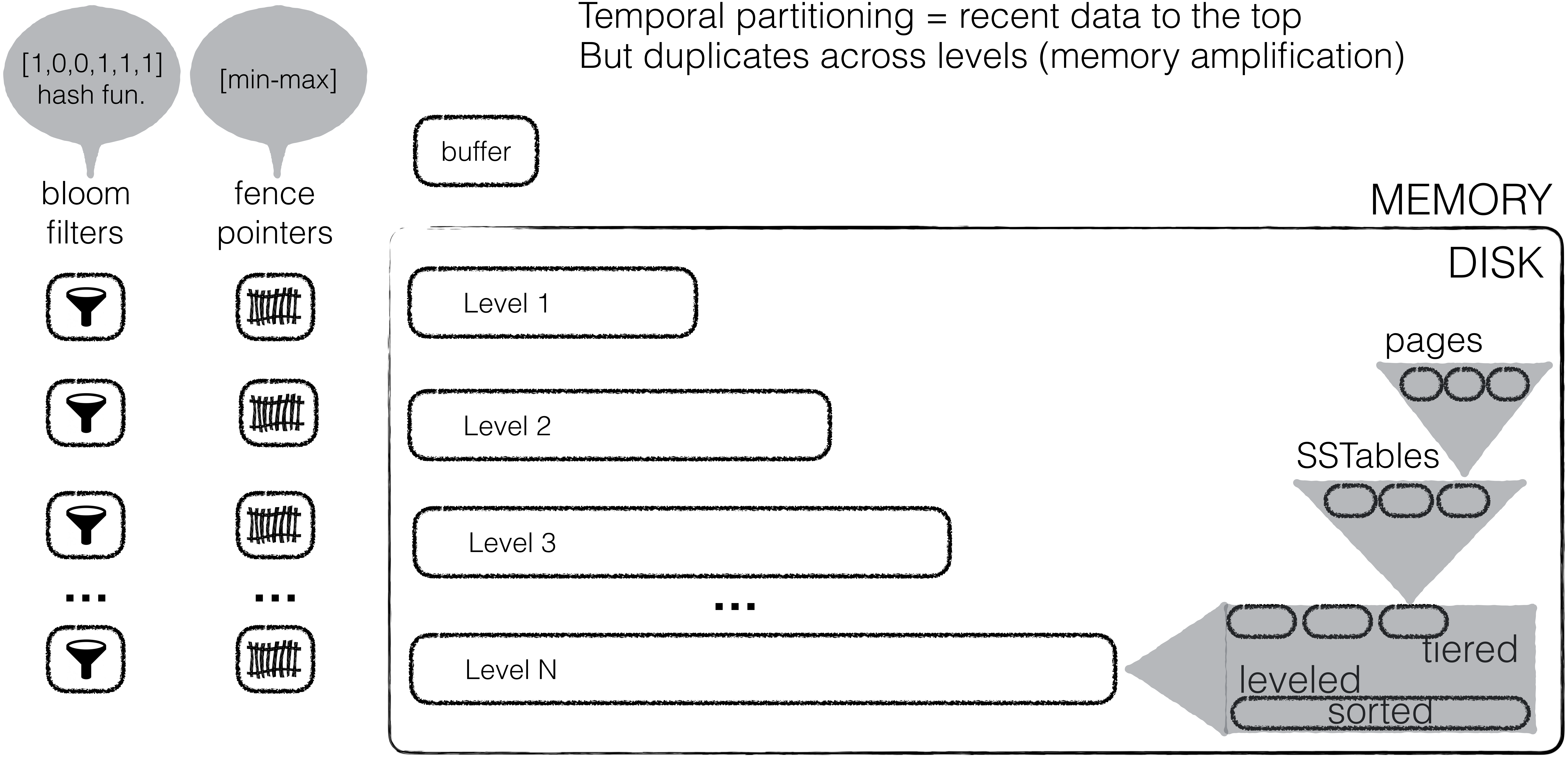
Instant writes, but overtime have to be merged >1



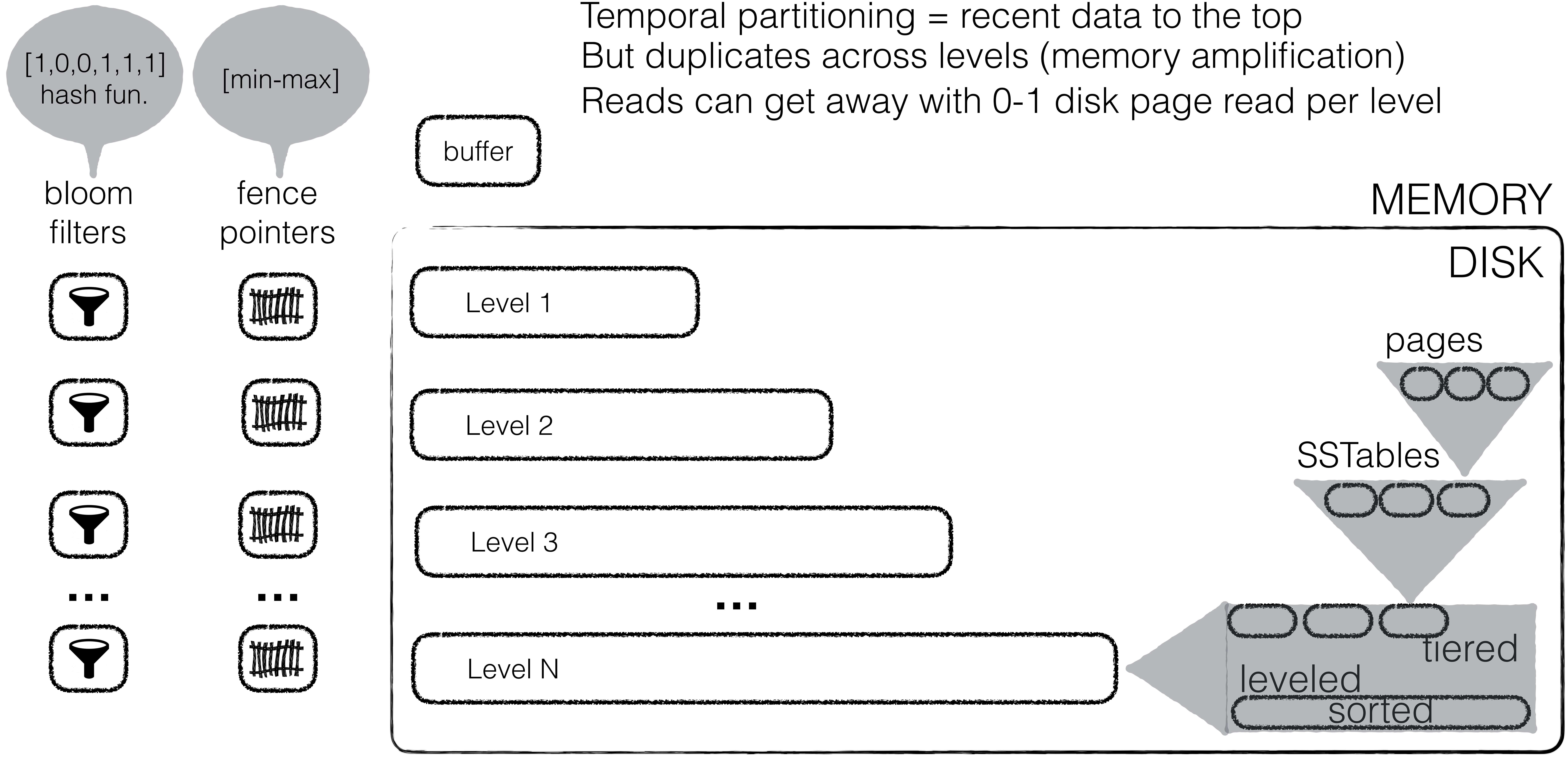
Instant writes, but overtime have to be merged >1
Temporal partitioning = recent data to the top



Instant writes, but overtime have to be merged >1
Temporal partitioning = recent data to the top
But duplicates across levels (memory amplification)



Instant writes, but overtime have to be merged >1
Temporal partitioning = recent data to the top
But duplicates across levels (memory amplification)
Reads can get away with 0-1 disk page read per level



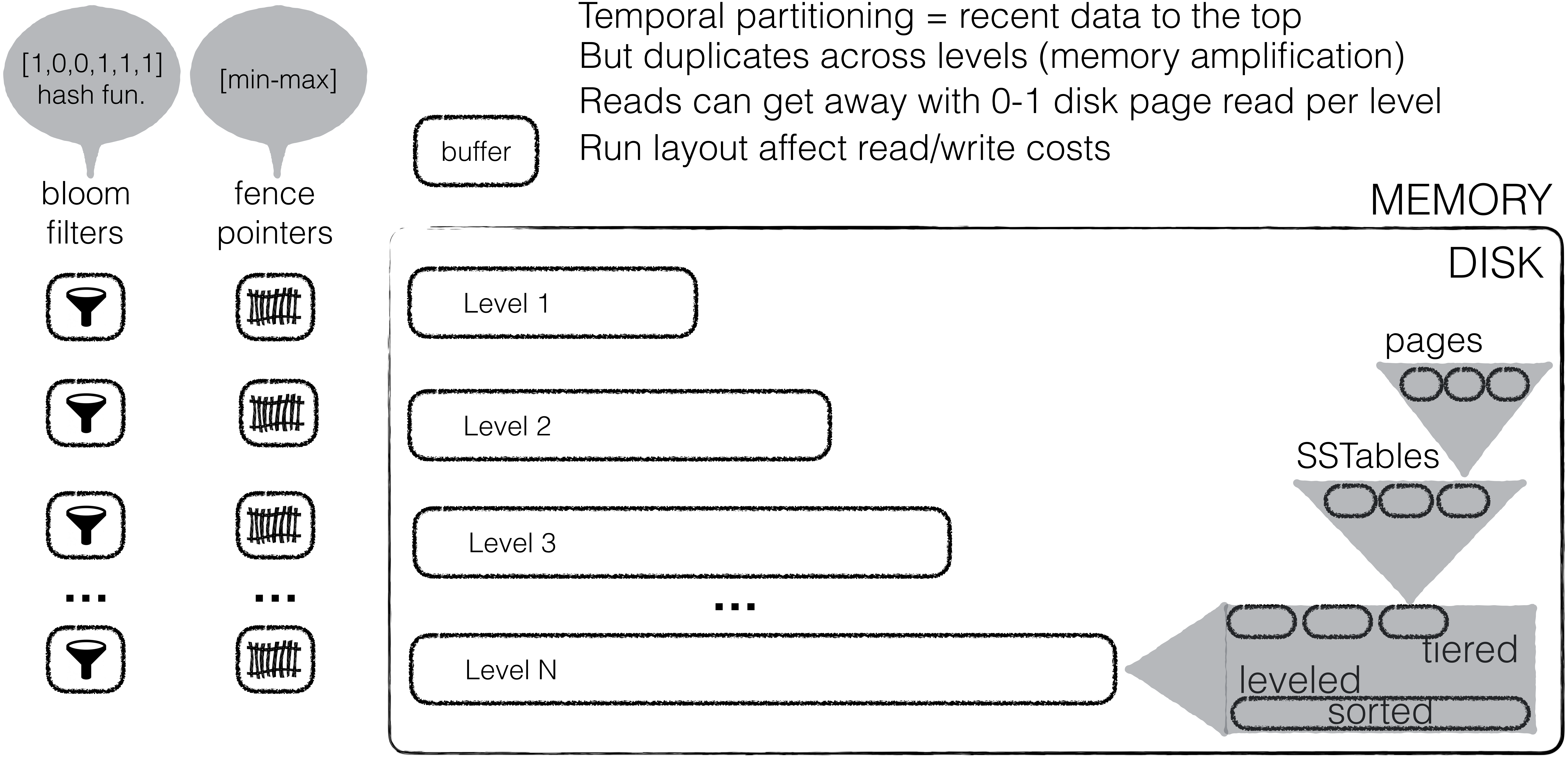
Instant writes, but overtime have to be merged >1

Temporal partitioning = recent data to the top

But duplicates across levels (memory amplification)

Reads can get away with 0-1 disk page read per level

Run layout affect read/write costs



Instant writes, but overtime have to be merged >1

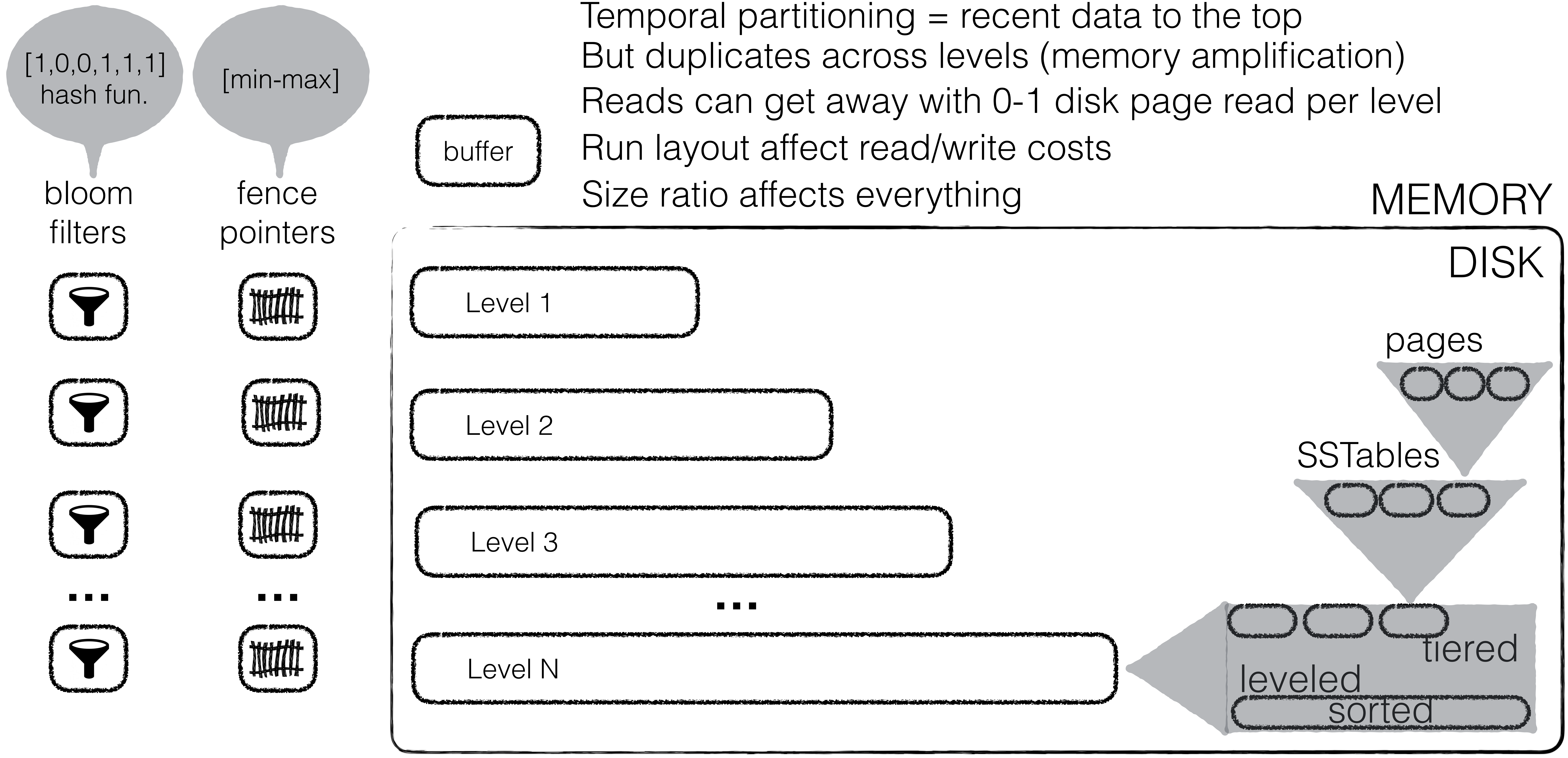
Temporal partitioning = recent data to the top

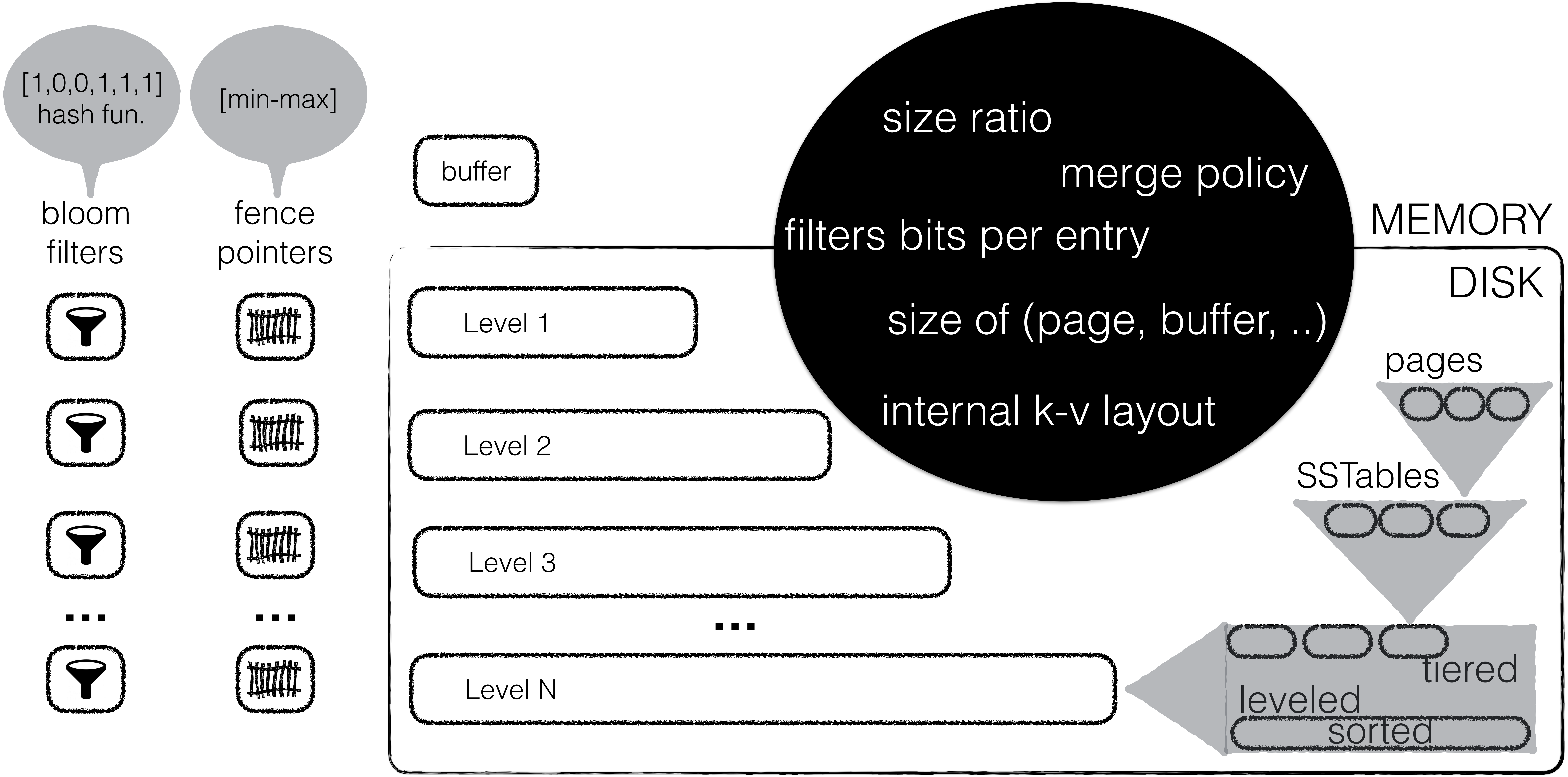
But duplicates across levels (memory amplification)

Reads can get away with 0-1 disk page read per level

Run layout affect read/write costs

Size ratio affects everything





[1,0,0,1,1,1]
hash fun.

bloom
filters

[min-max]

fence
pointers

…

…

buffer

MEMORY

DISK

Level 1

Level 2

Level 3

…

Level N

size ratio

merge policy

filters bits per entry

size of (page, buffer, ..)

internal k-v layout

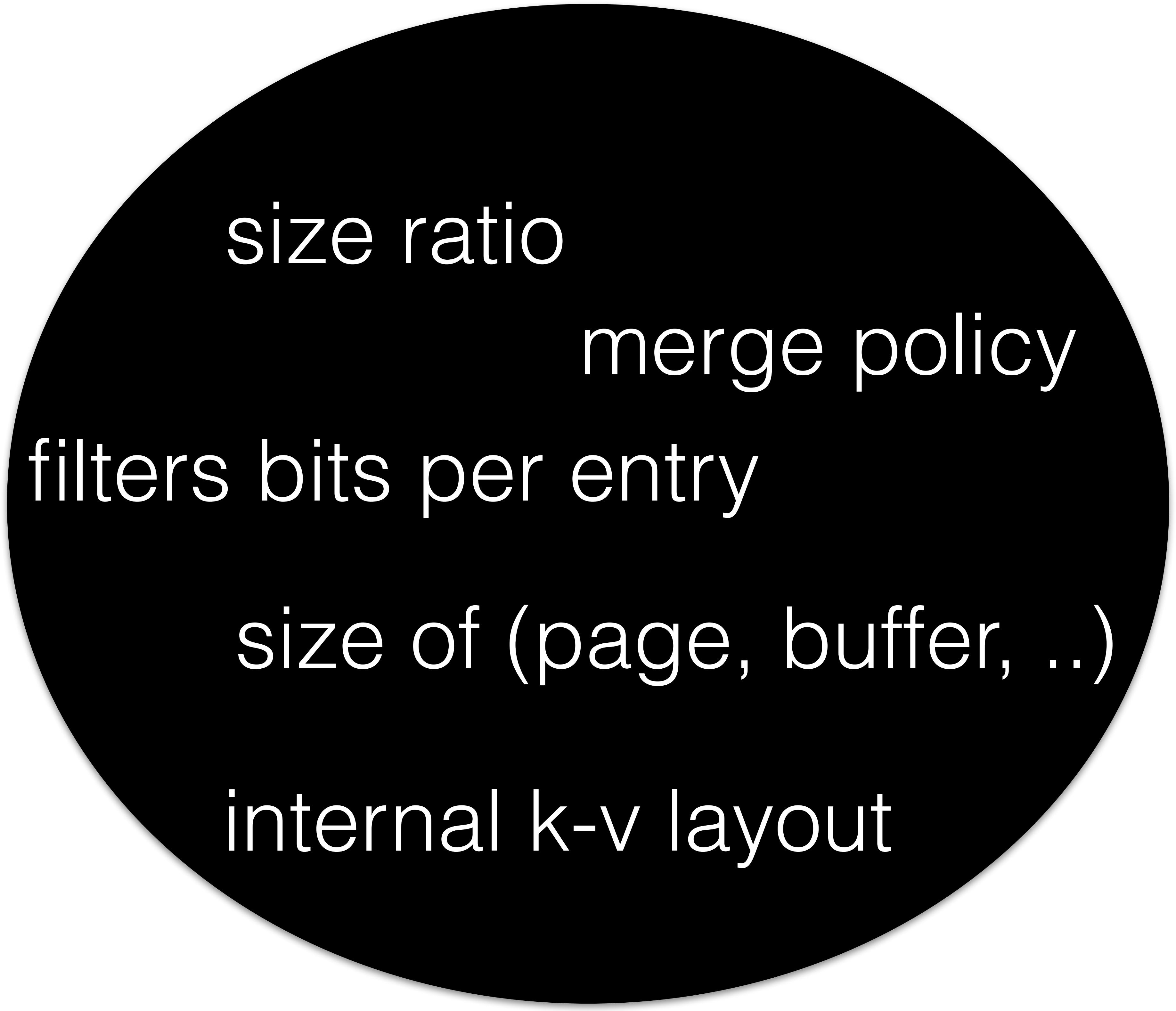
pages

SSTables

tiered

leveled

sorted



size ratio

merge policy

filters bits per entry

size of (page, buffer, ..)

internal k-v layout

For every design principle X:

1. Which are X’s meaningful values?
2. How does X affect read, update and memory amplification?
3. Should X be a design principle or can it be optimized out?

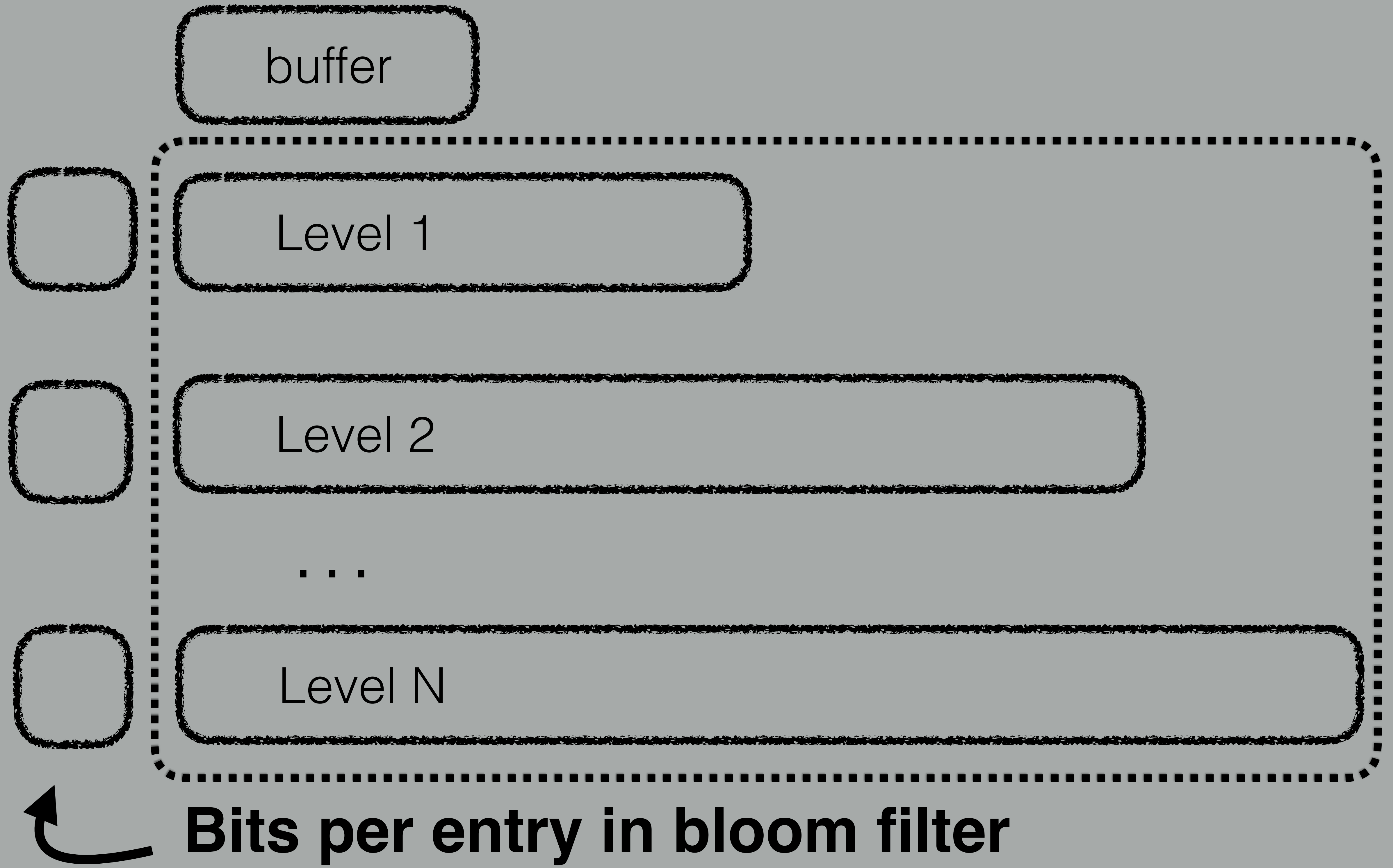
bits per entry in filters

monkey@SIGMOD2017

Monkey: **O**ptimal **N**avigable **Key**-Value Store standard design: fixed per run

at most one I/O per level

worst case I/O: sum of false positive rates



Bits per entry in bloom filter

bits per entry in filters

monkey@SIGMOD2017

Monkey: **O**ptimal **N**avigable **Key**-Value Store

bits per entry in filters

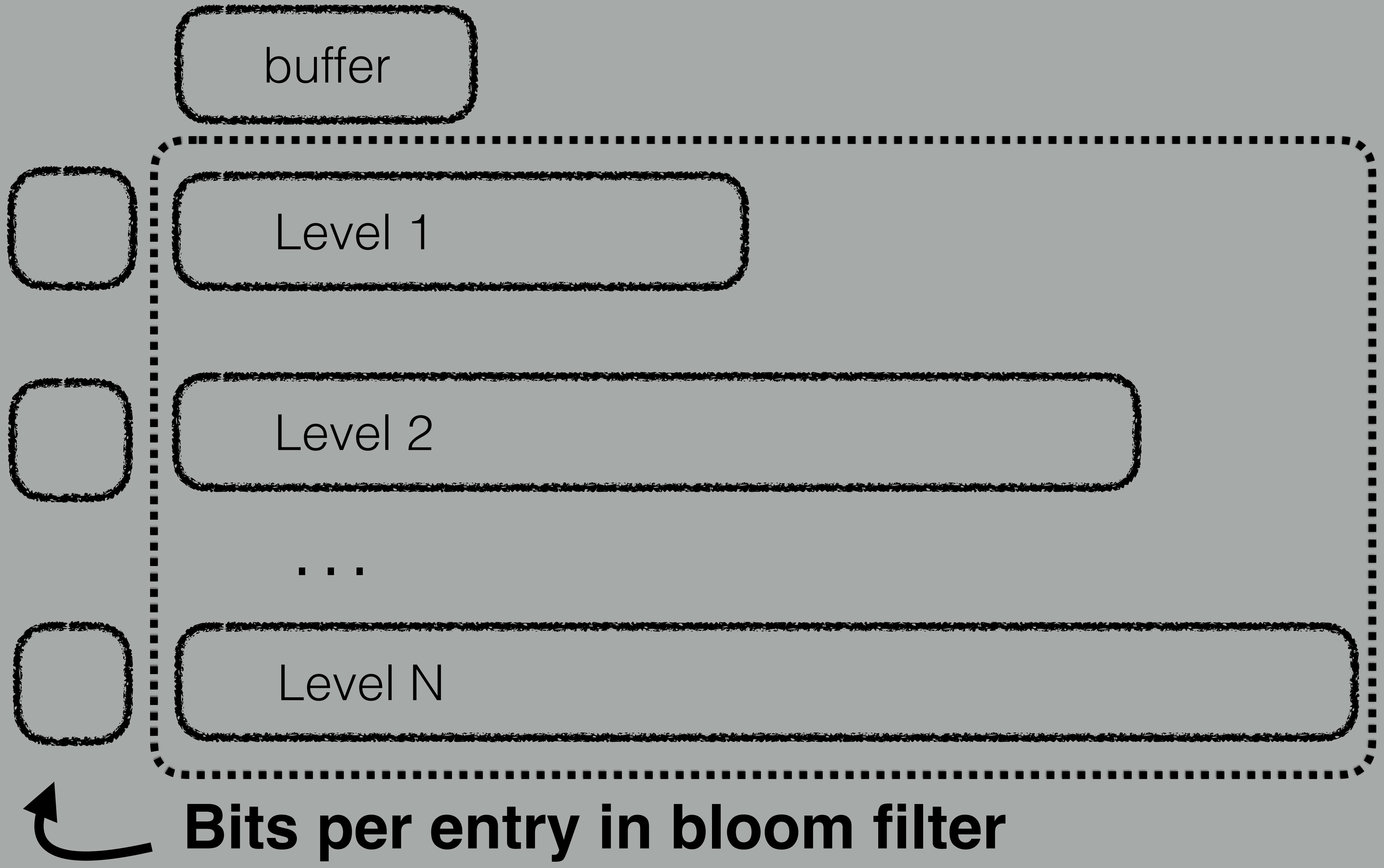
monkey@SIGMOD2017

Monkey: **O**ptimal **N**avigable

Key-Value Store

minimize sum of FPRs

"move" filters memory budget from big to small levels



Bits per entry in bloom filter

bits per entry in filters

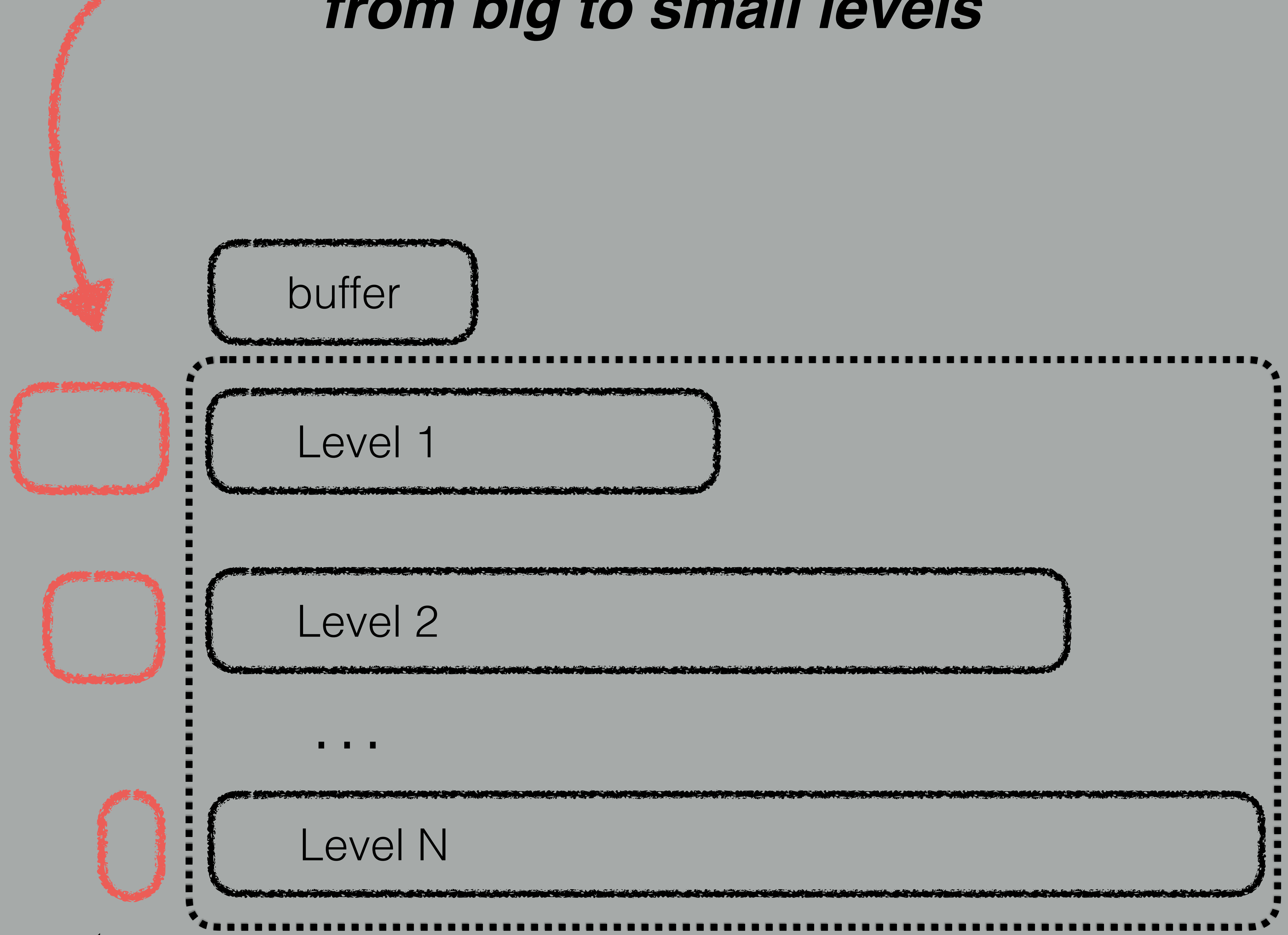
monkey@SIGMOD2017

Monkey: **O**ptimal **N**avigable

Key-Value Store

minimize sum of FPRs

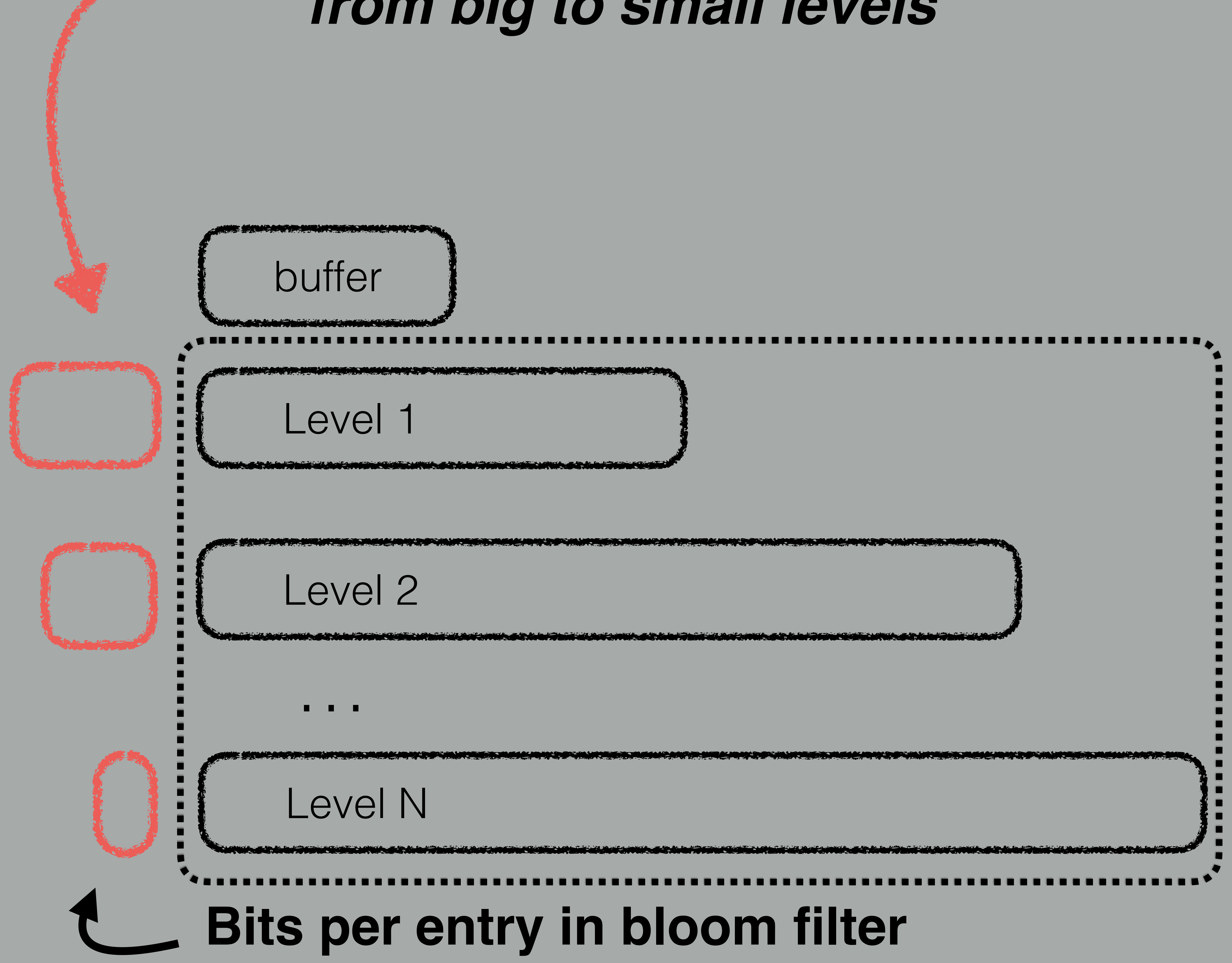
"move" filters memory budget
from big to small levels



Bits per entry in bloom filter

minimize sum of FPRs

“move" filters memory budget from big to small levels

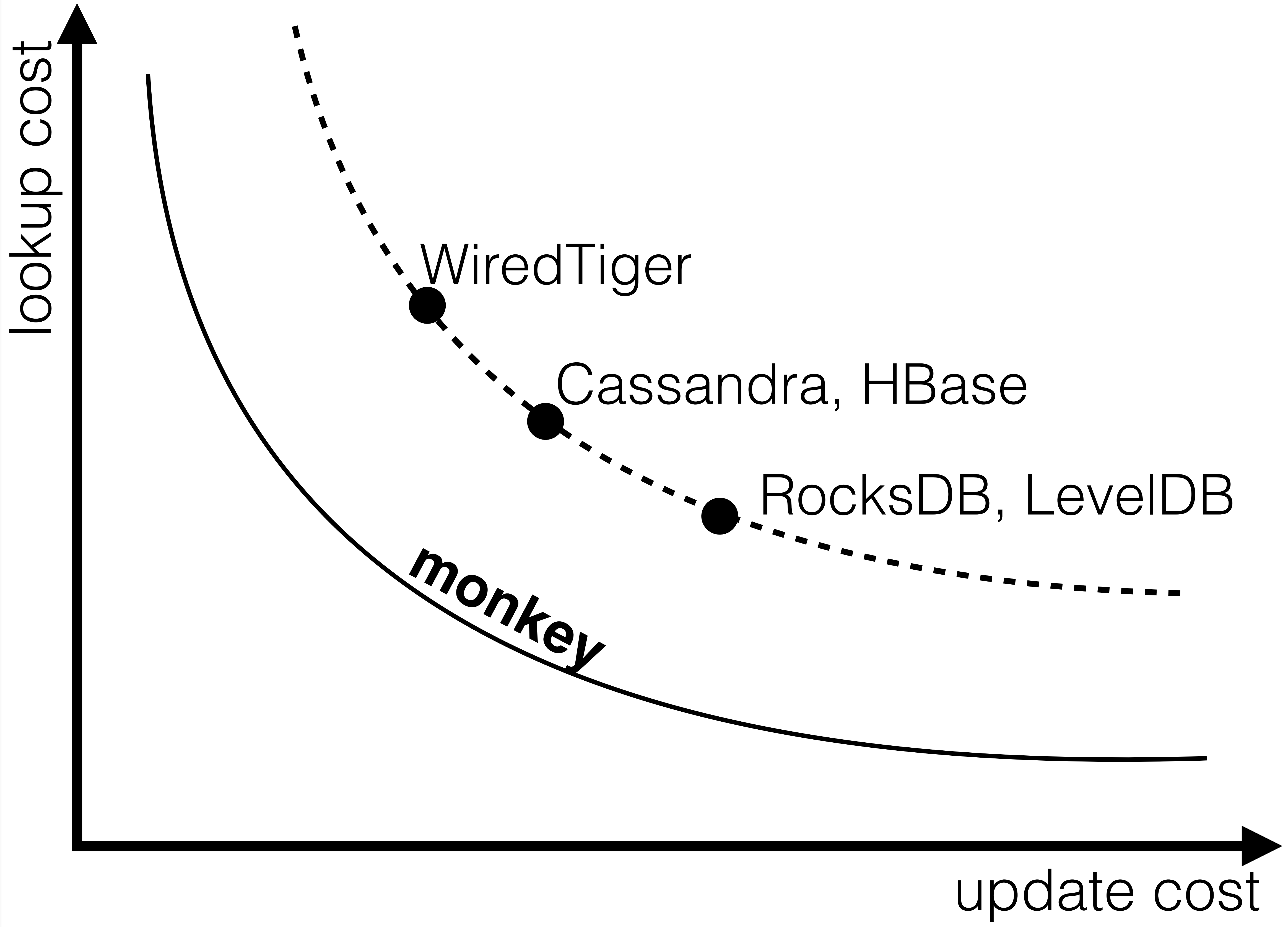


Bits per entry in bloom filter

bits per entry in filters

monkey@SIGMOD2017

Monkey: **O**ptimal **N**avigable **Key**-Value Store

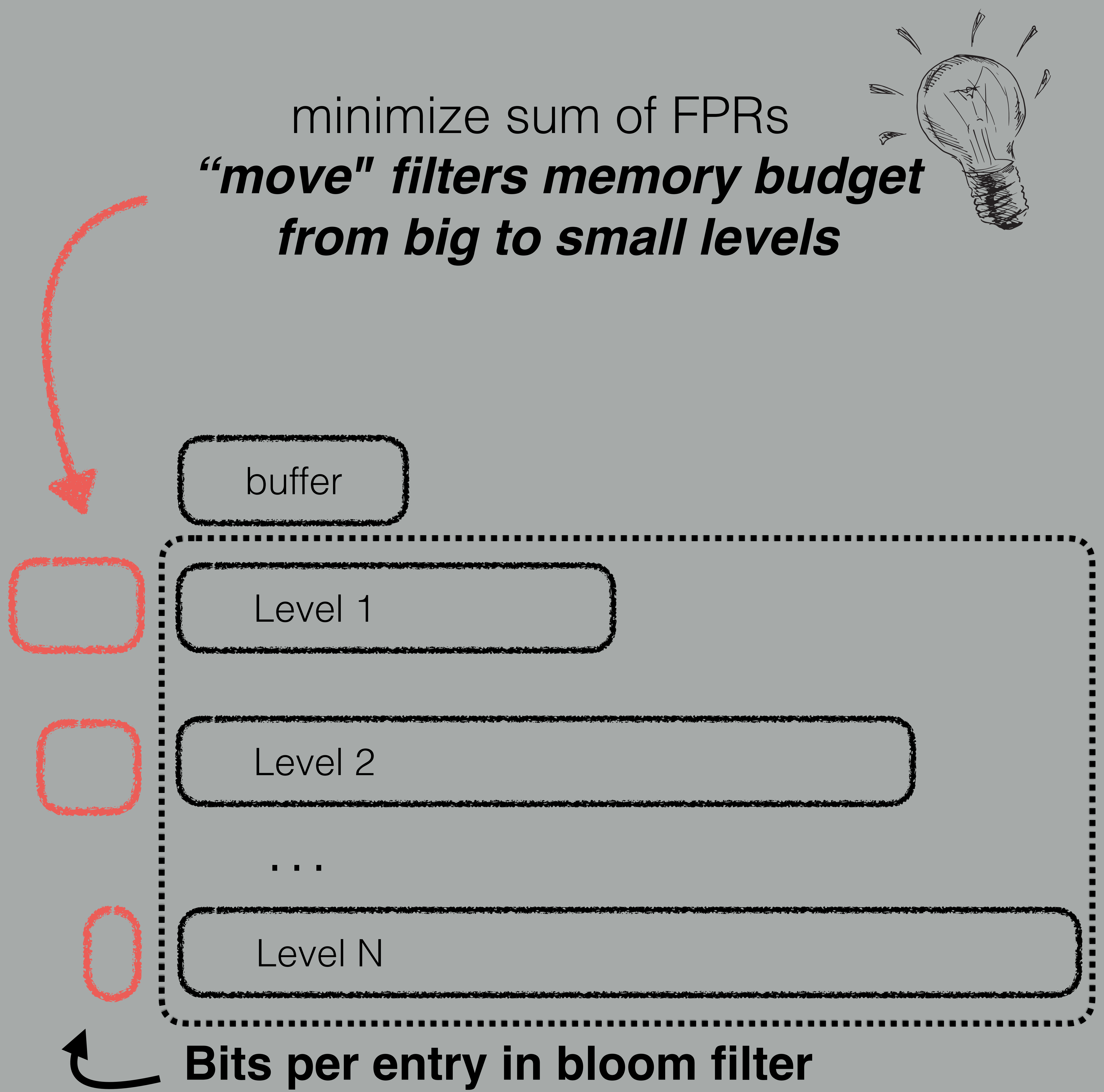


bits per entry in filters

monkey@SIGMOD2017

Monkey: **O**ptimal **N**avigable

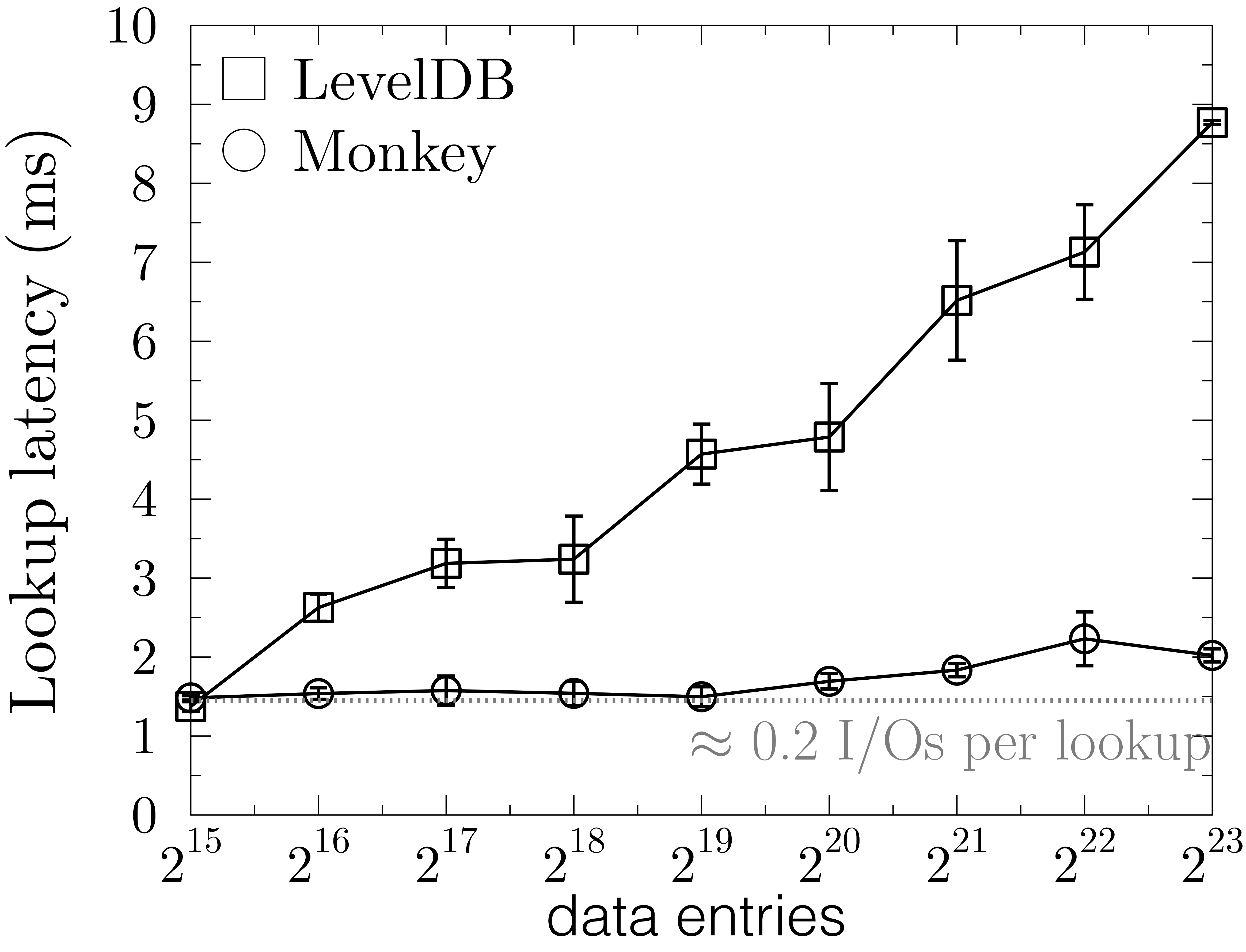
Key-Value Store



minimize sum of FPRs

“move" filters memory budget from big to small levels

Bits per entry in bloom filter



uniform, zero result, point queries, entry size=1KB

merge policy

dostoevsky@SIGMOD2018

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

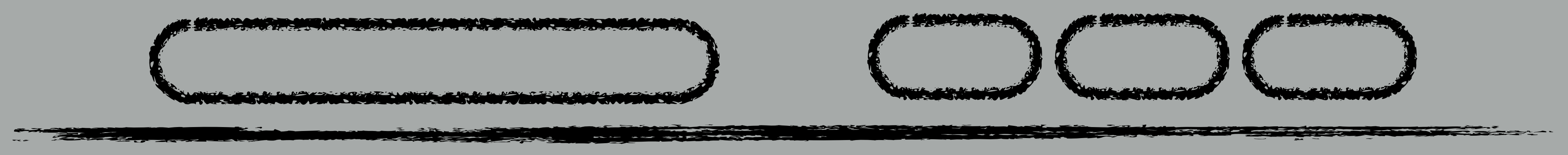
merge policy

dostoevsky@SIGMOD2018

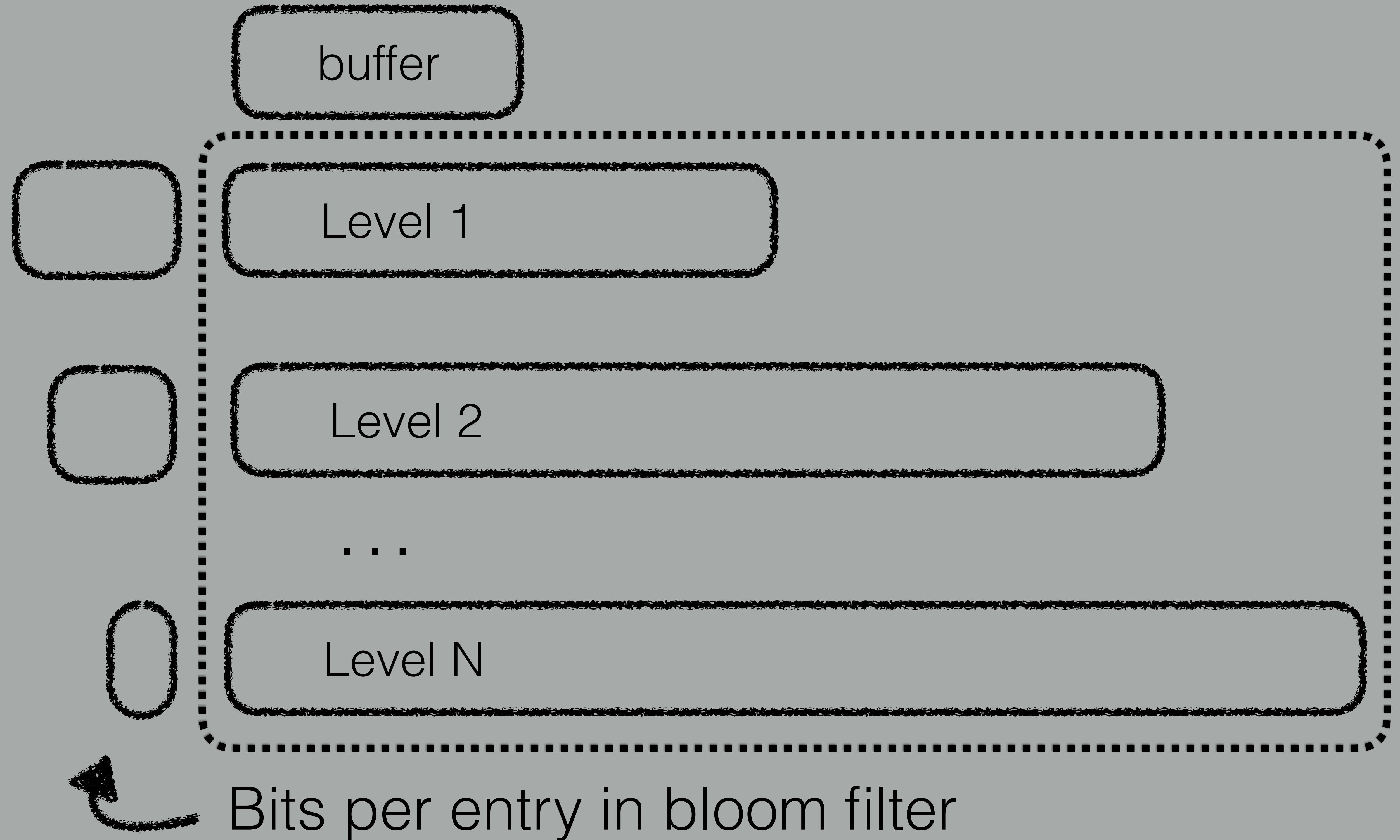
Dostoevsky: **S**pace-**T**ime **O**ptimized

Evolvable **S**calable **K**ey-Value Store

standard design: fixed across levels



Leveled helps reads; tiered helps writes



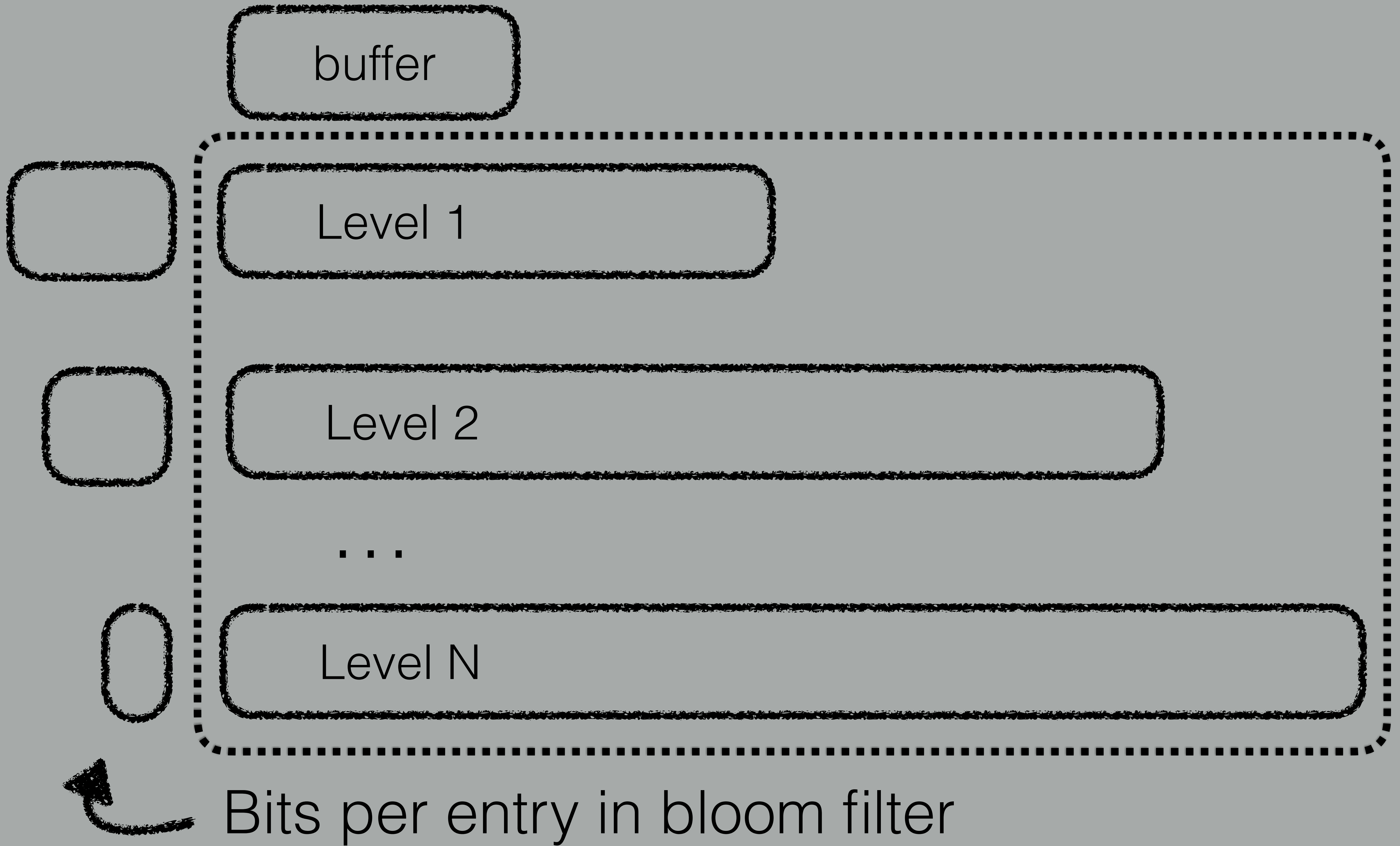
Bits per entry in bloom filter

merge policy

dostoevsky@SIGMOD2018

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

hybrid merge policy tiered for small levels



buffer

Level 1

Level 2

. . .

Level N

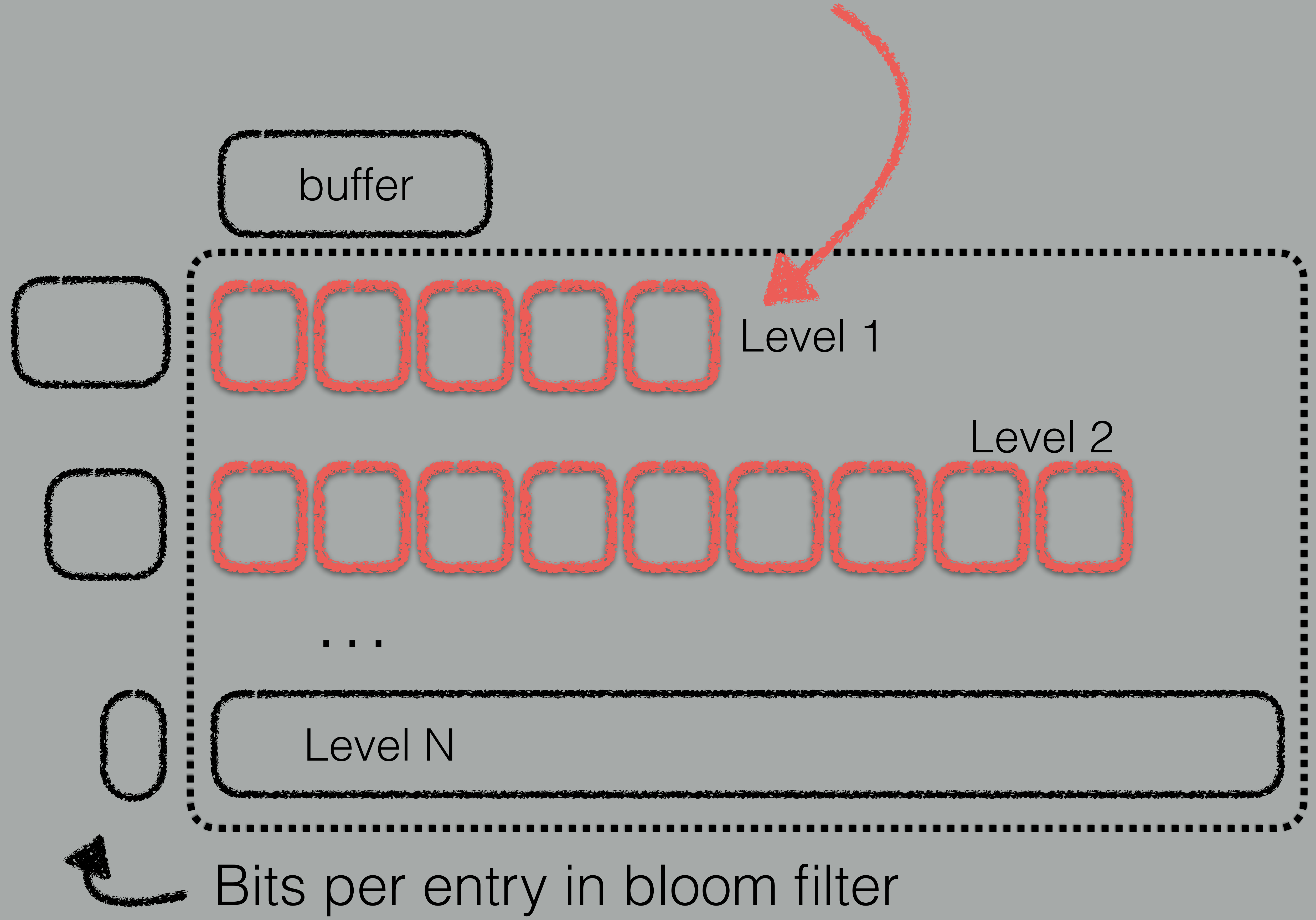
Bits per entry in bloom filter

merge policy

dostoevsky@SIGMOD2018

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

hybrid merge policy
tiered for small levels

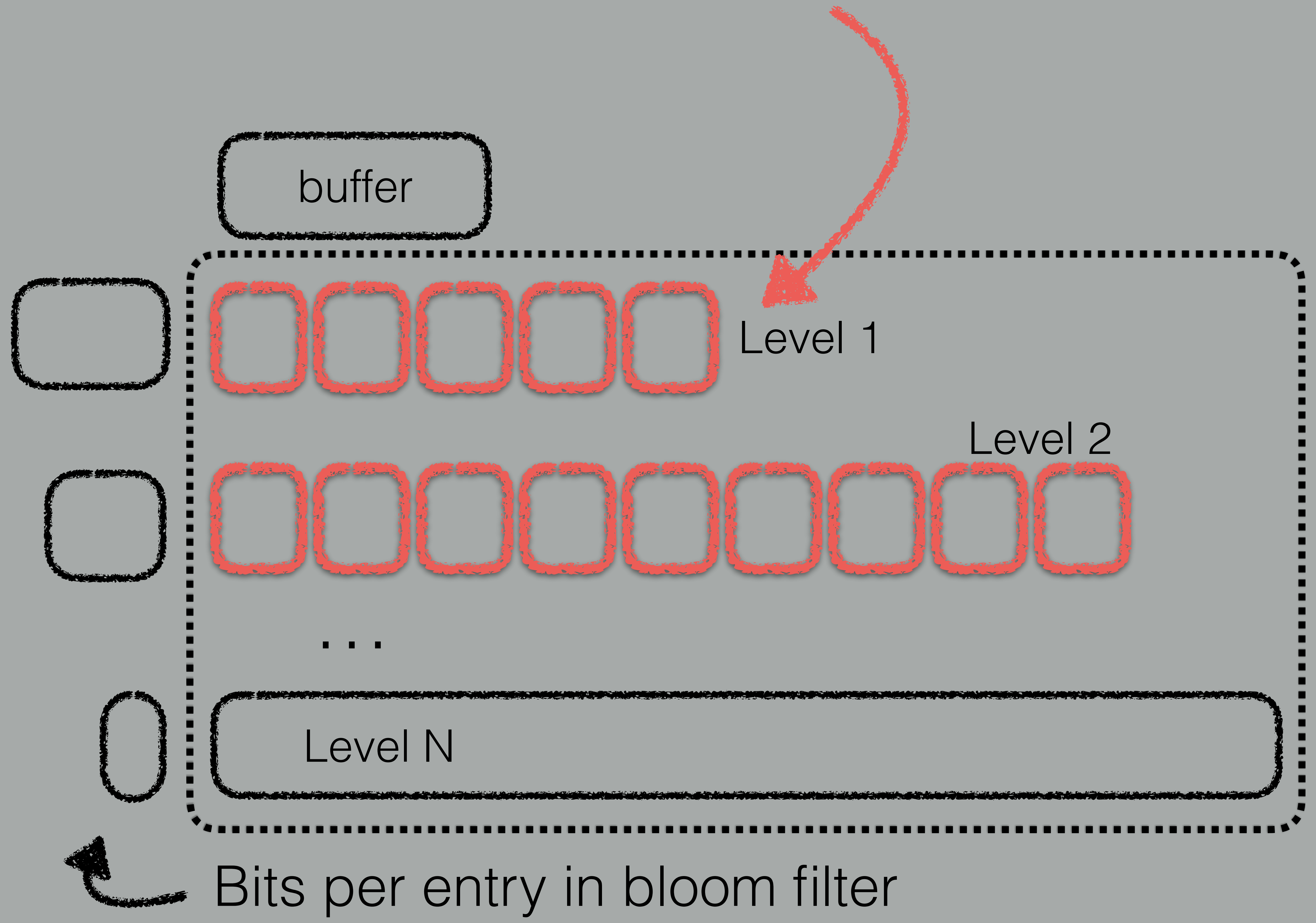


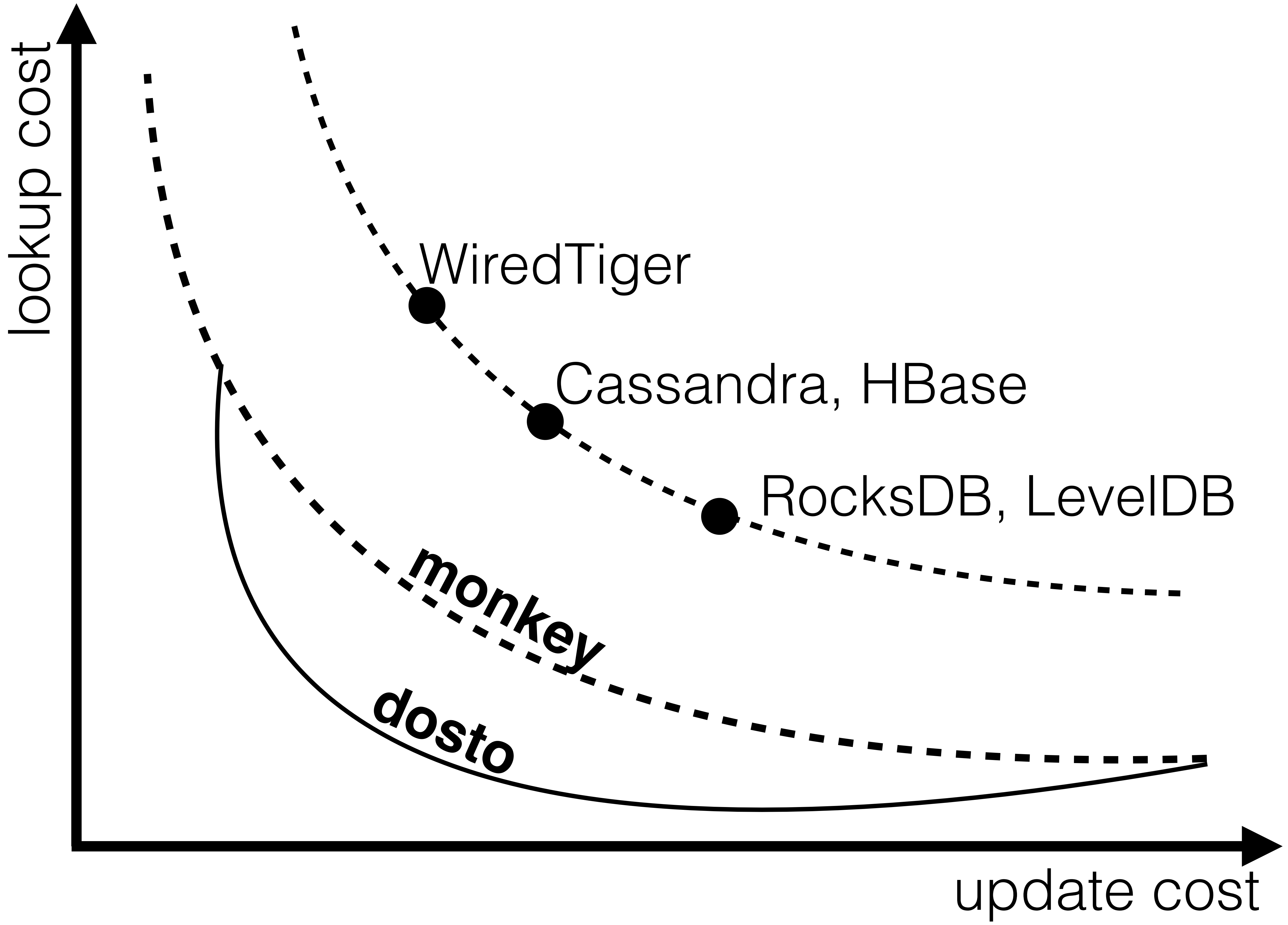
merge policy

dostoevsky@SIGMOD2018

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

hybrid merge policy
tiered for small levels



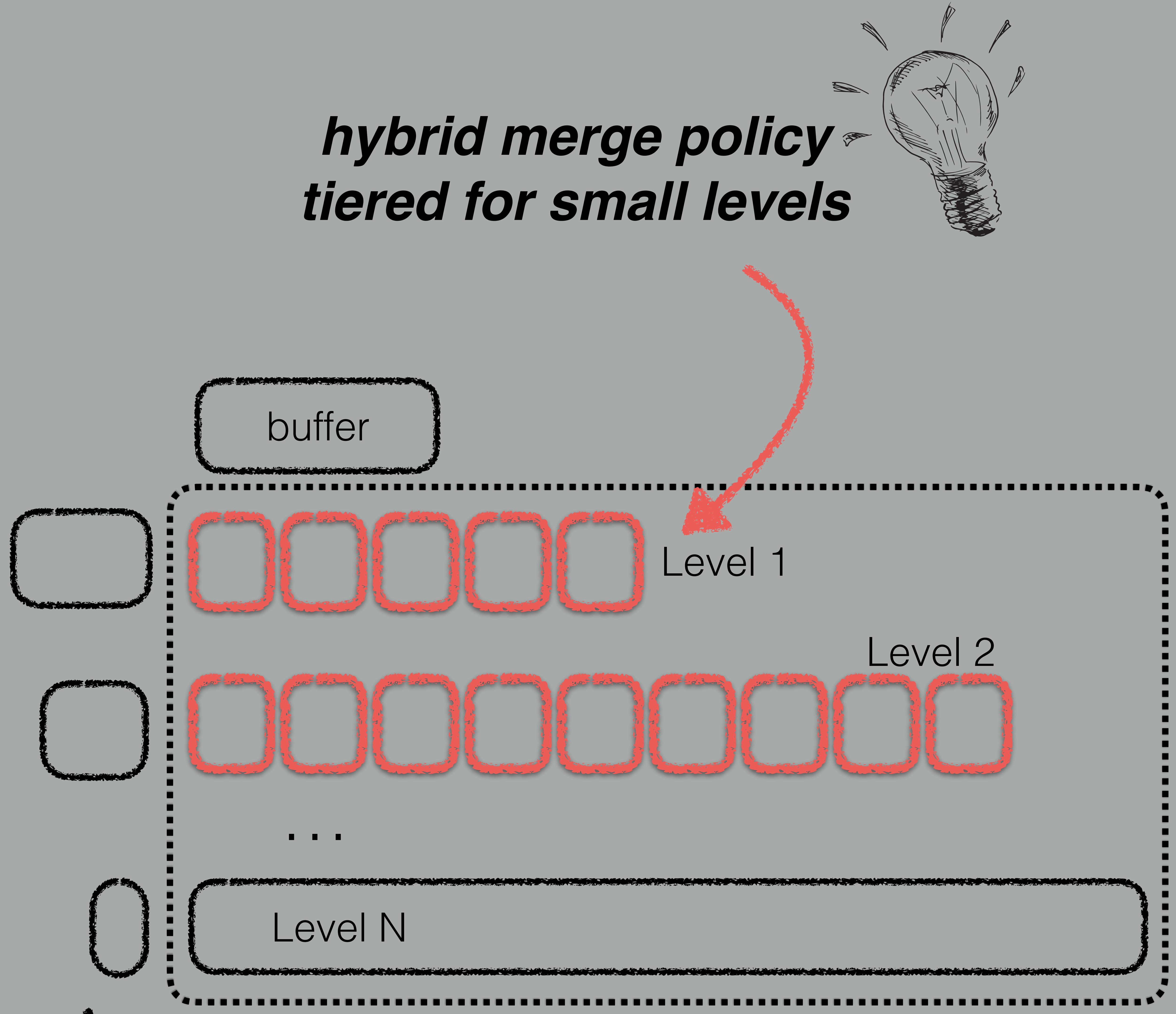


merge policy

dostoevsky@SIGMOD2018

Dostoevsky: **S**pace-**T**ime **O**ptimized

Evolvable **S**calable **Ke**y-Value Store



hybrid merge policy
tiered for small levels

buffer

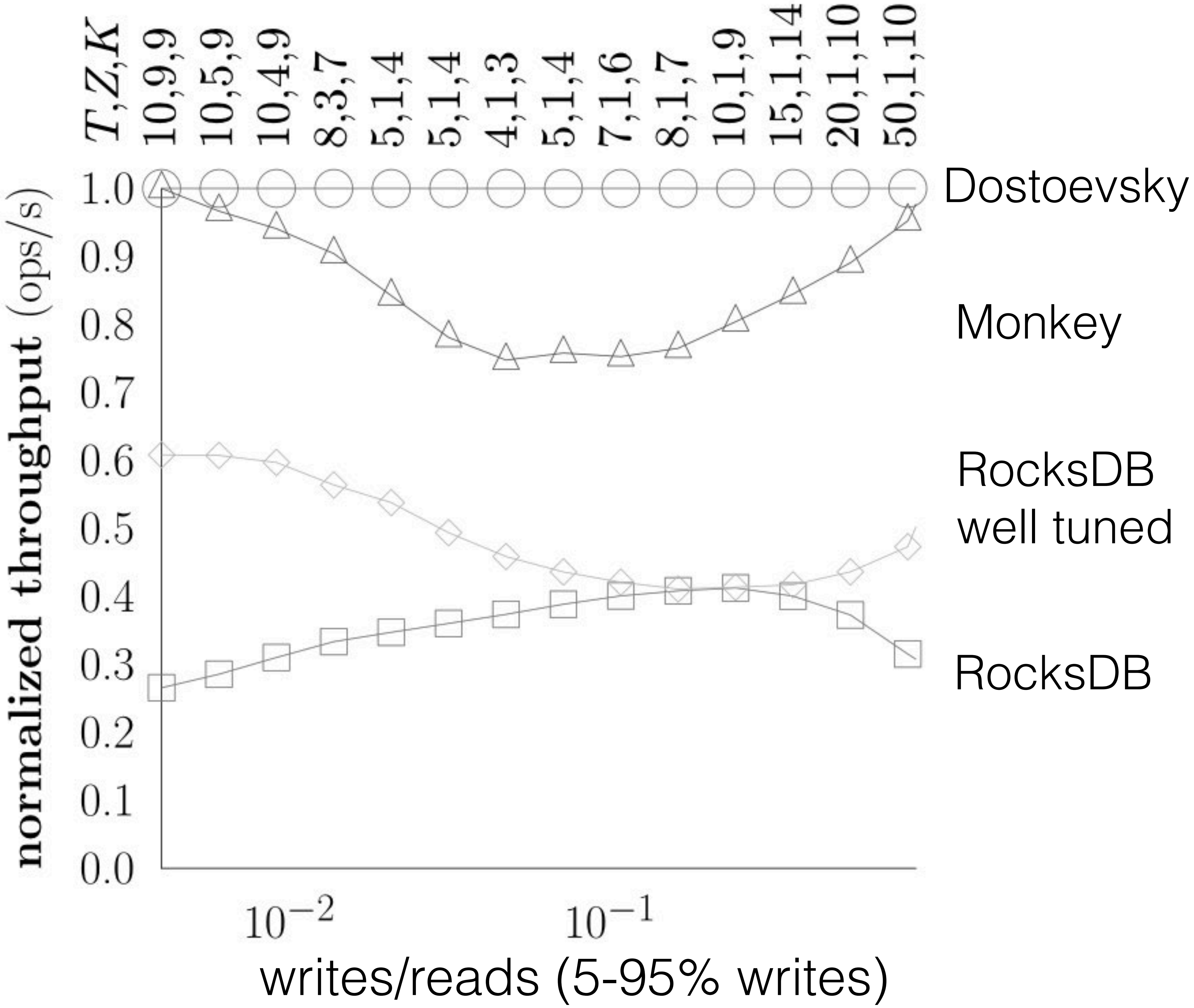
Level 1

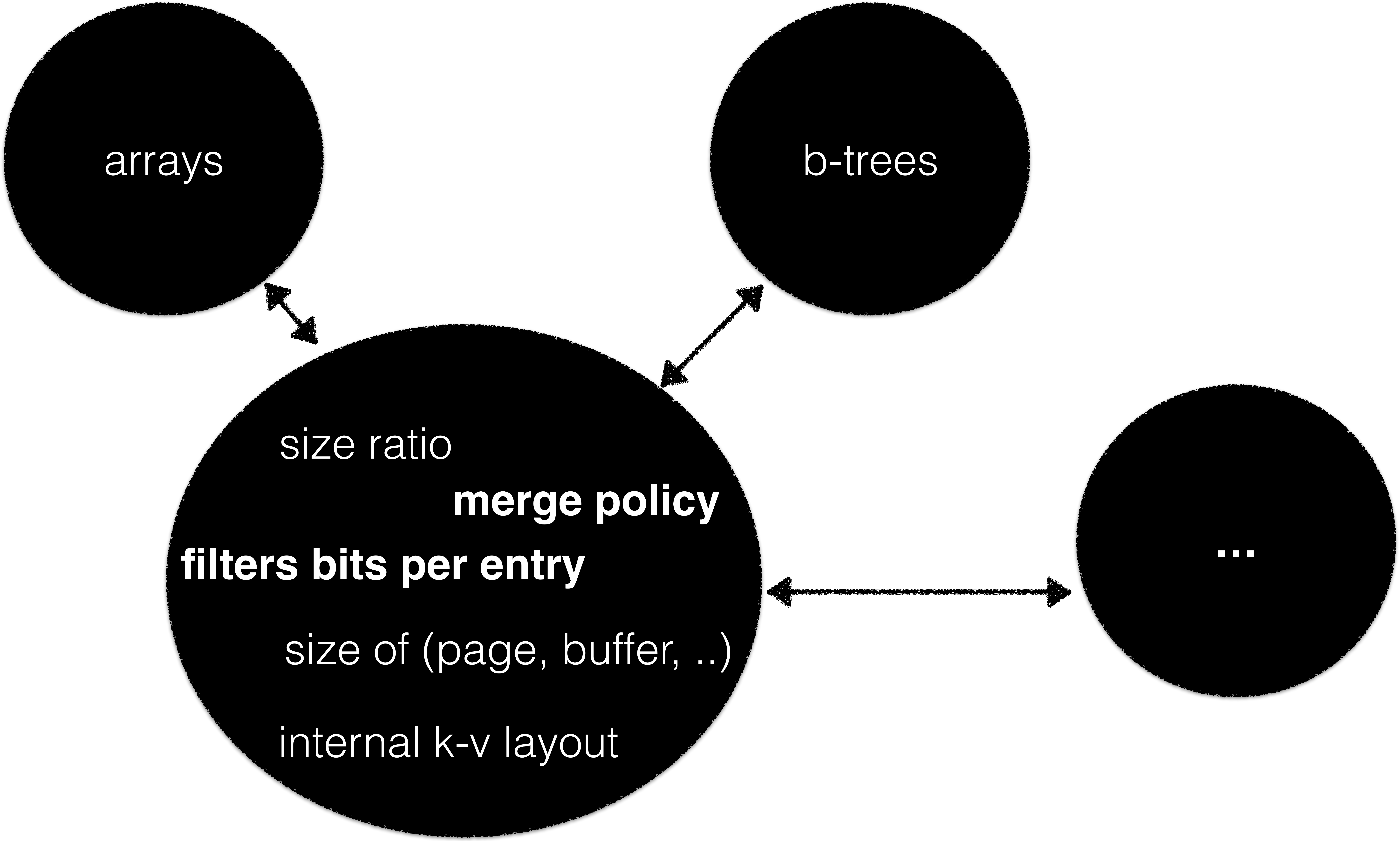
Level 2

. . .

Level N

Bits per entry in bloom filter





arrays

b-trees

size ratio

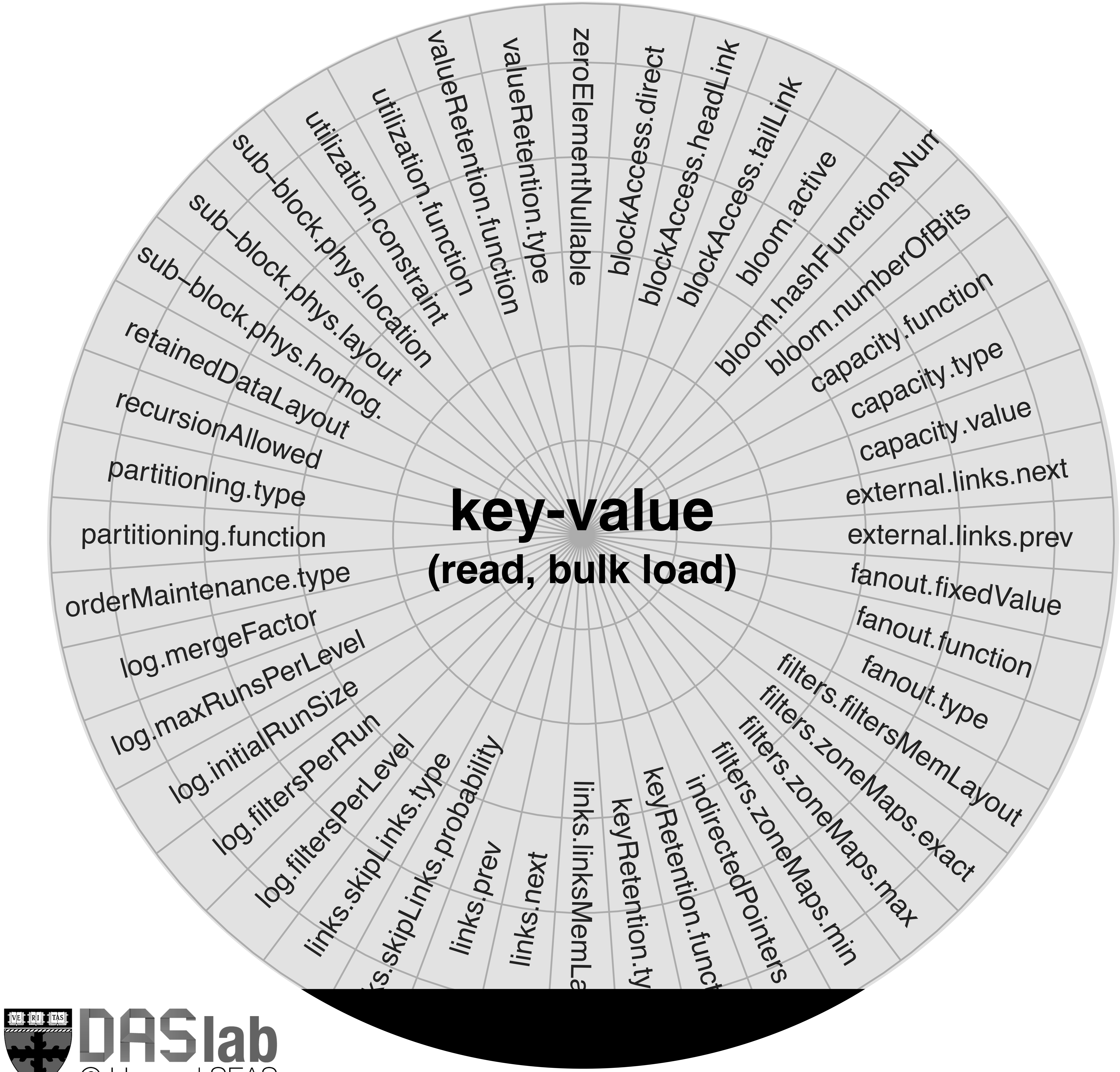
merge policy

filters bits per entry

size of (page, buffer, ..)

internal k-v layout

...



key-value

(read, bulk load)

blockAccess.direct

blockAccess.headLink

blockAccess.tailLink

bloom.active

bloom.hashFunctionsNum

bloom.numberOfBits

capacity.function

capacity.type

capacity.value

external.links.next

external.links.prev

fanout.fixedValue

fanout.function

fanout.type

filters.filtersMemLayout

filters.zoneMaps.exact

filters.zoneMaps.max

filters.zoneMaps.min

indirectedPointers

keyRetention.funct

keyRetention.ty

links.linksMemLa

links.next

links.prev

s.skipLinks.probability

links.skipLinks.type

log.filtersPerLevel

log.filtersPerRun

log.initialRunSize

log.maxRunsPerLevel

log.mergeFactor

orderMaintenance.type

partitioning.function

partitioning.type

recursionAllowed

retainedDataLayout

sub–block.phys.homog.

sub–block.phys.layout

sub–block.phys.location

utilization.constraint

utilization.function

valueRetention.function

valueRetention.type

zeroElementNullable

DESCRIBE ONE DATA BLOCK AT A TIME

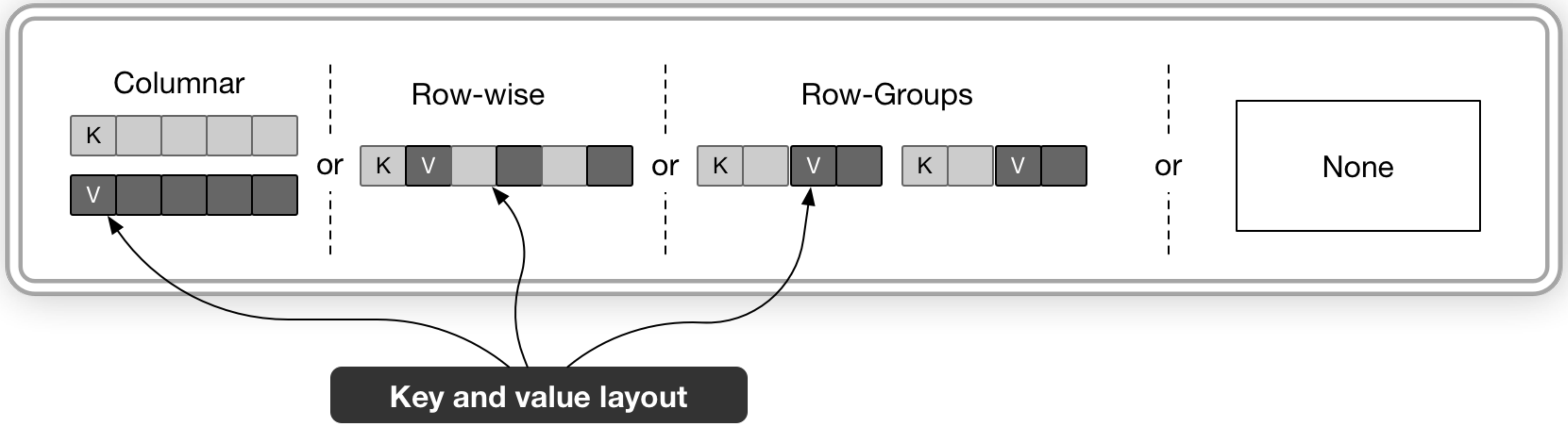
AS A SET OF CONCEPTS

physical layout and domain partitioning

Are keys retained? (yes, no, function)

Are values retained?

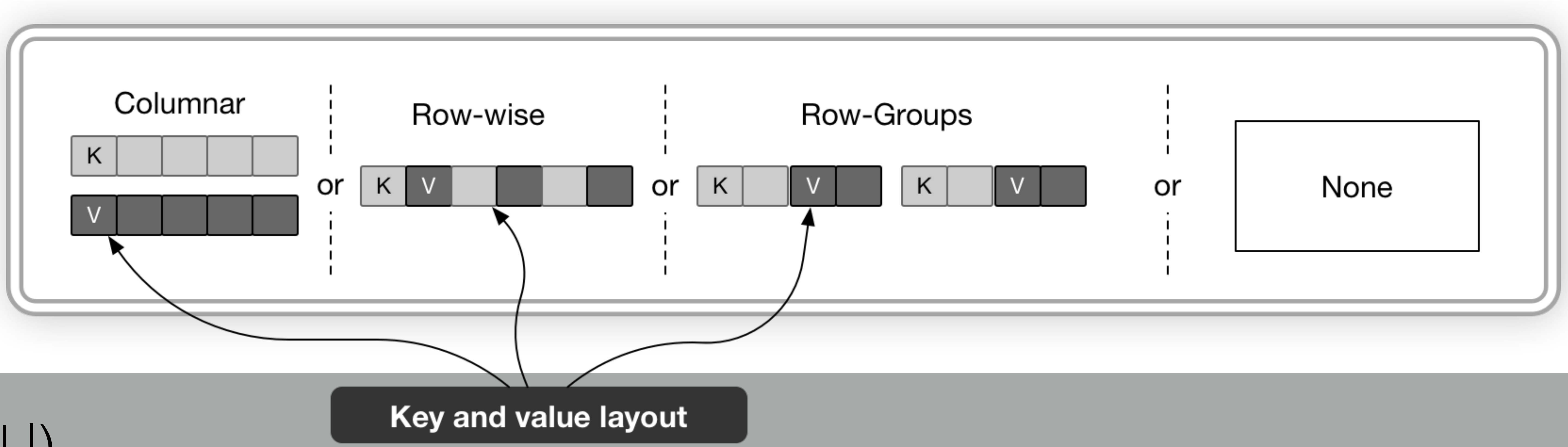
Utilization? (e.g., >50%)



Are keys retained? (yes, no, function)

Are values retained?

Utilization? (e.g., >50%)



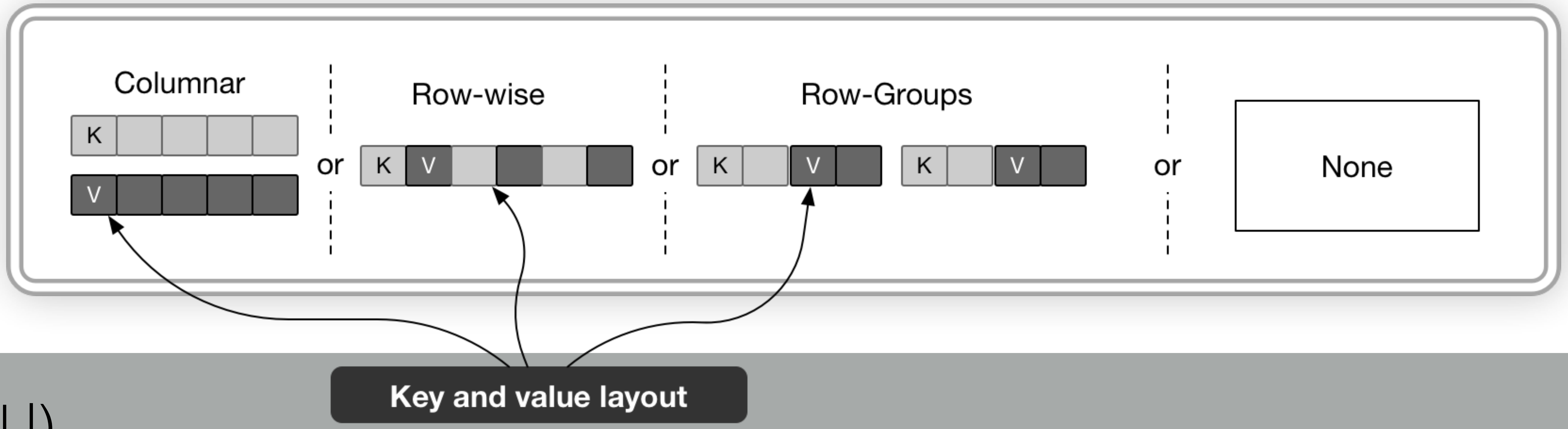
Fanout (fixed/functional | unlimited | terminal |)

Key partitioning (none(fw-append | bw-append) | sorted | range() | radix() | function (func) | temporal(…))

Are keys retained? (yes, no, function)

Are values retained?

Utilization? (e.g., >50%)



Fanout (fixed/functional | unlimited | terminal |)

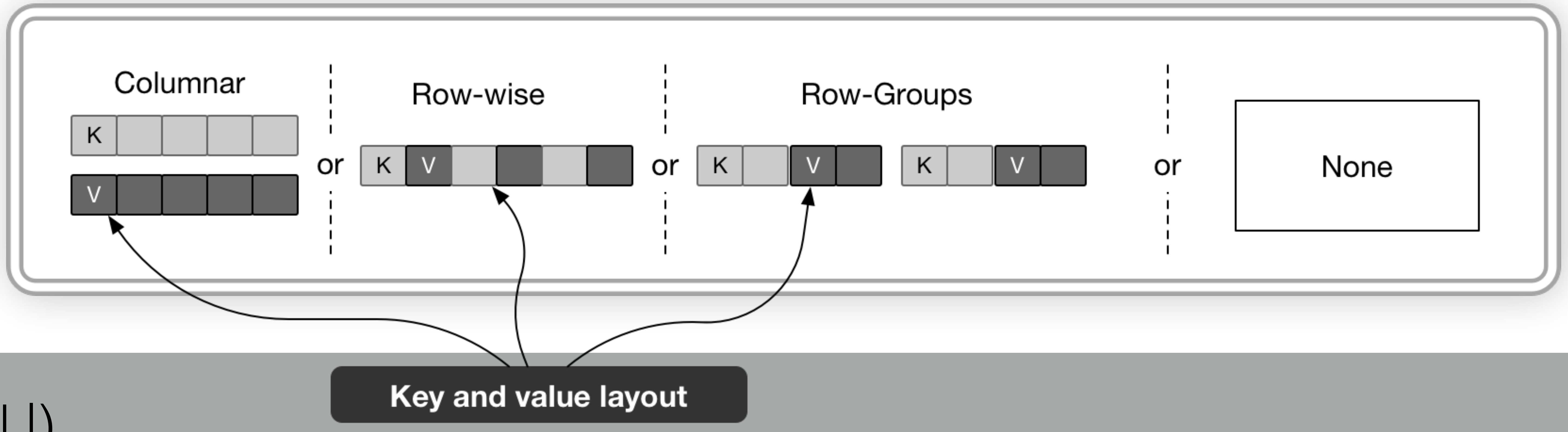
Key partitioning (none(fw-append | bw-append) | sorted | range() | radix() | function (func) | temporal(…))

Intra node access (direct | head_link | tail_link | link_function(func))

Are keys retained? (yes, no, function)

Are values retained?

Utilization? (e.g., >50%)



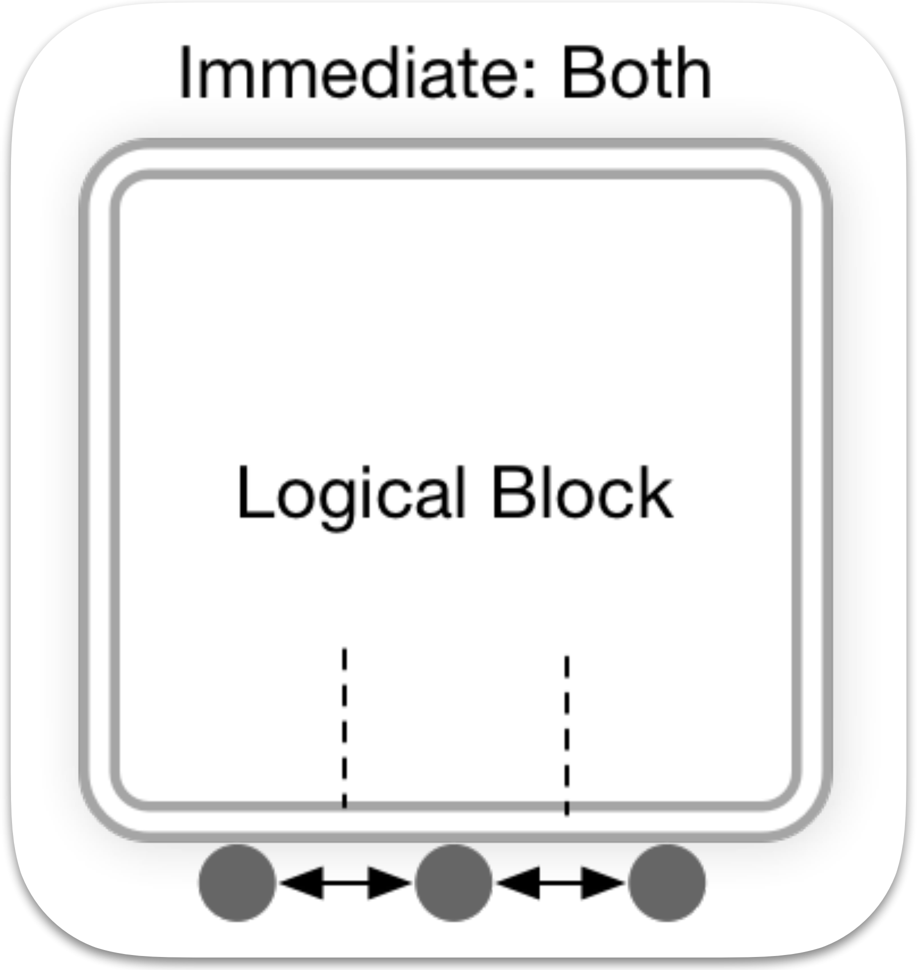
Fanout (fixed/functional | unlimited | terminal |)

Key partitioning (none(fw-append | bw-append) | sorted | range() | radix() | function (func) | temporal(…))

Intra node access (direct | head_link | tail_link | link_function(func))

Sub block links (next | previous | both | none)

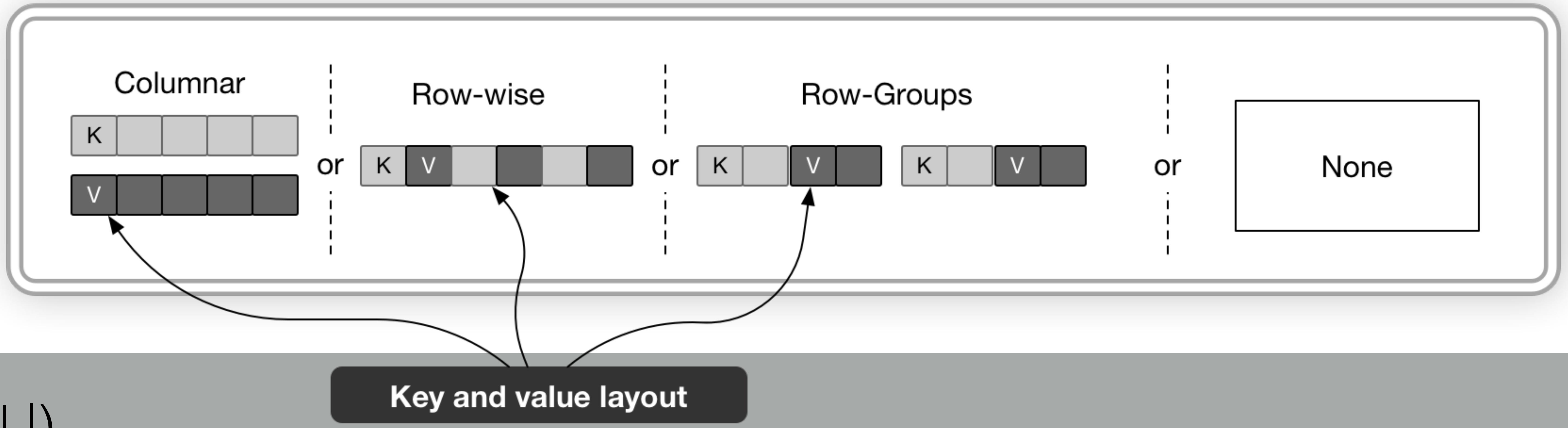
Sub block skip links (perfect | randomized(prob: double) | function(func) | none)



Are keys retained? (yes, no, function)

Are values retained?

Utilization? (e.g., >50%)



Fanout (fixed/functional | unlimited | terminal |)

Key partitioning (none(fw-append | bw-append) | sorted | range() | radix() | function (func) | temporal(…))

Intra node access (direct | head_link | tail_link | link_function(func))

Sub block links (next | previous | both | none)

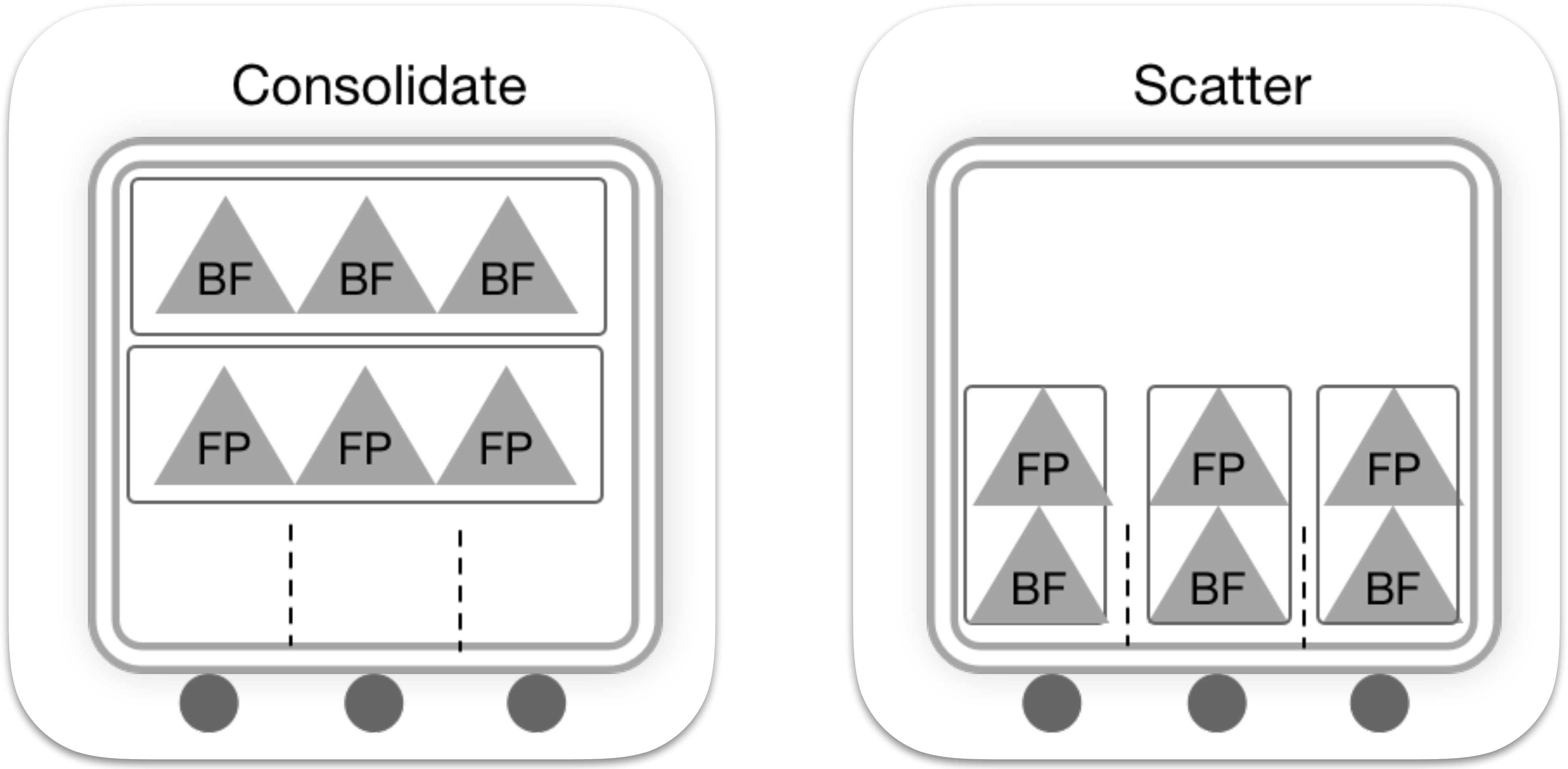
Sub block skip links (perfect | randomized(prob: double) | function(func) | none)

Zone Maps (min | max | both | exact | off)

Bloom filters (off | on(num_hashes: int, num_bits: int))

Filters layout (consolidate | scatter)

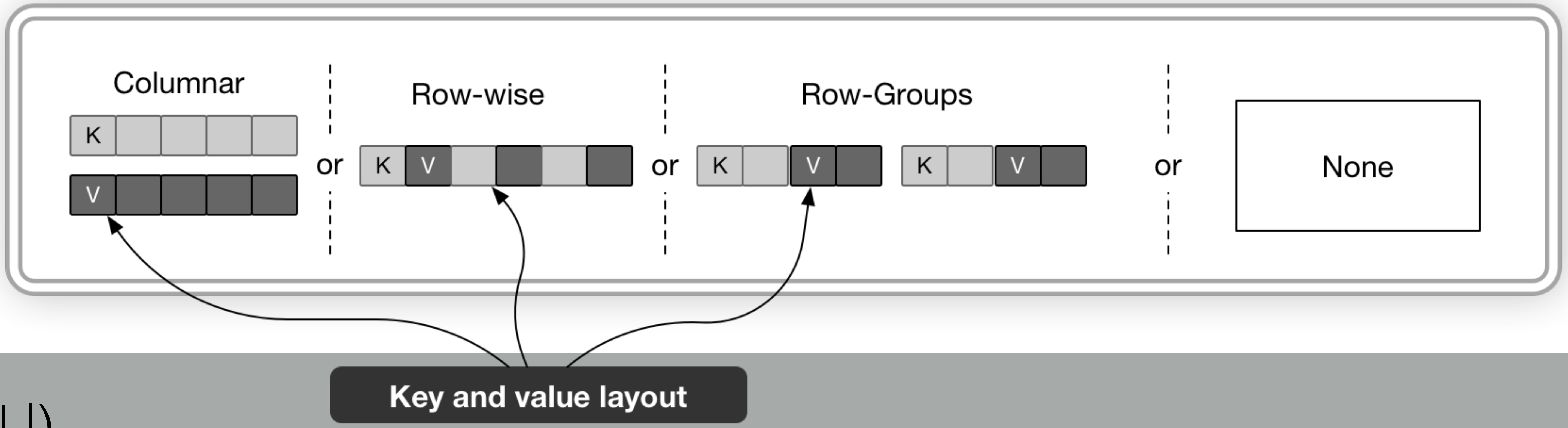
Links layout (consolidate | scatter)



Are keys retained? (yes, no, function)

Are values retained?

Utilization? (e.g., >50%)



Fanout (fixed/functional | unlimited | terminal |)

Key partitioning (none(fw-append | bw-append) | sorted | range() | radix() | function (func) | temporal(…))

Intra node access (direct | head_link | tail_link | link_function(func))

Sub block links (next | previous | both | none)

Sub block skip links (perfect | randomized(prob: double) | function(func) | none)

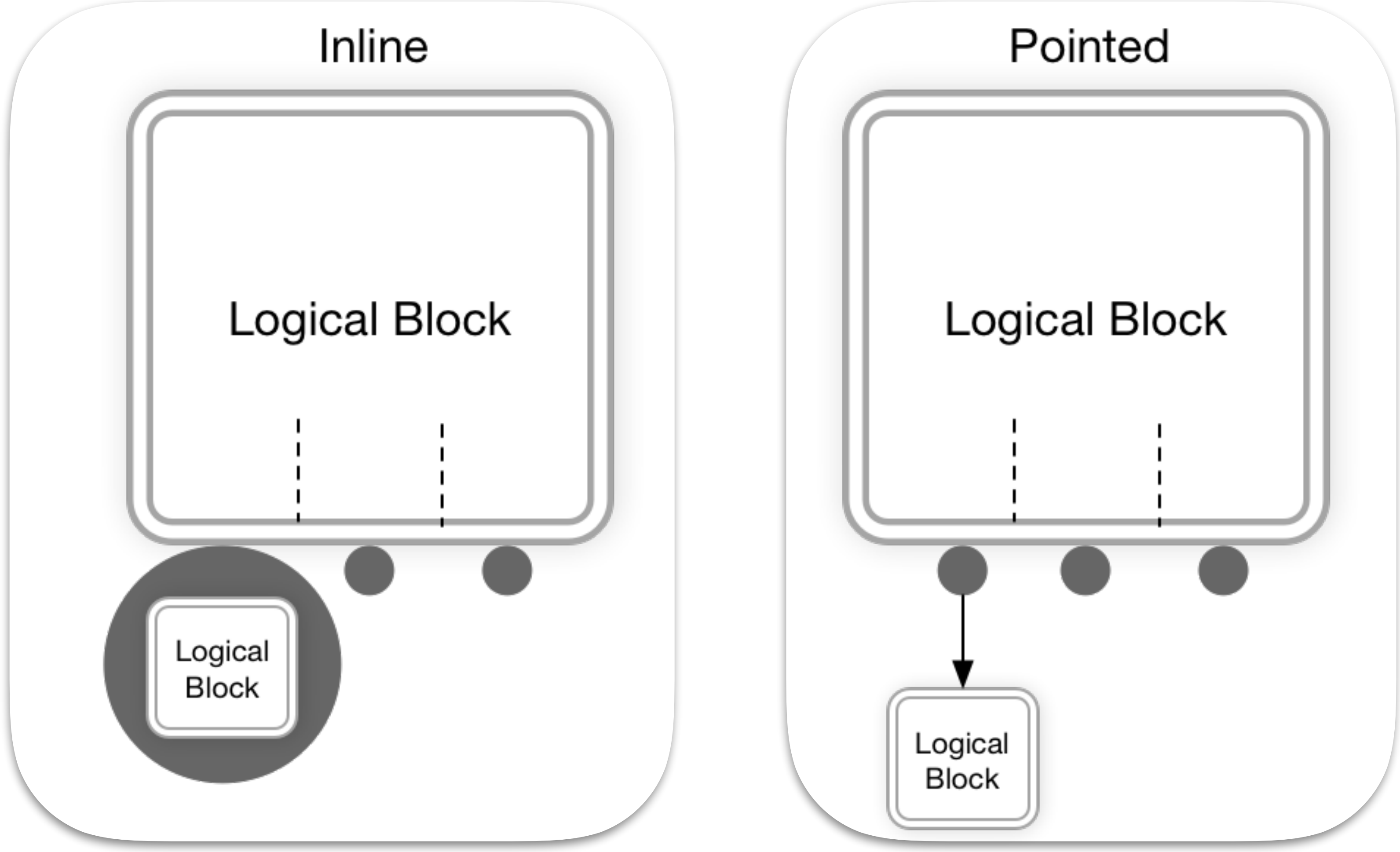
Zone Maps (min | max | both | exact | off)

Bloom filters (off | on(num_hashes: int, num_bits: int))

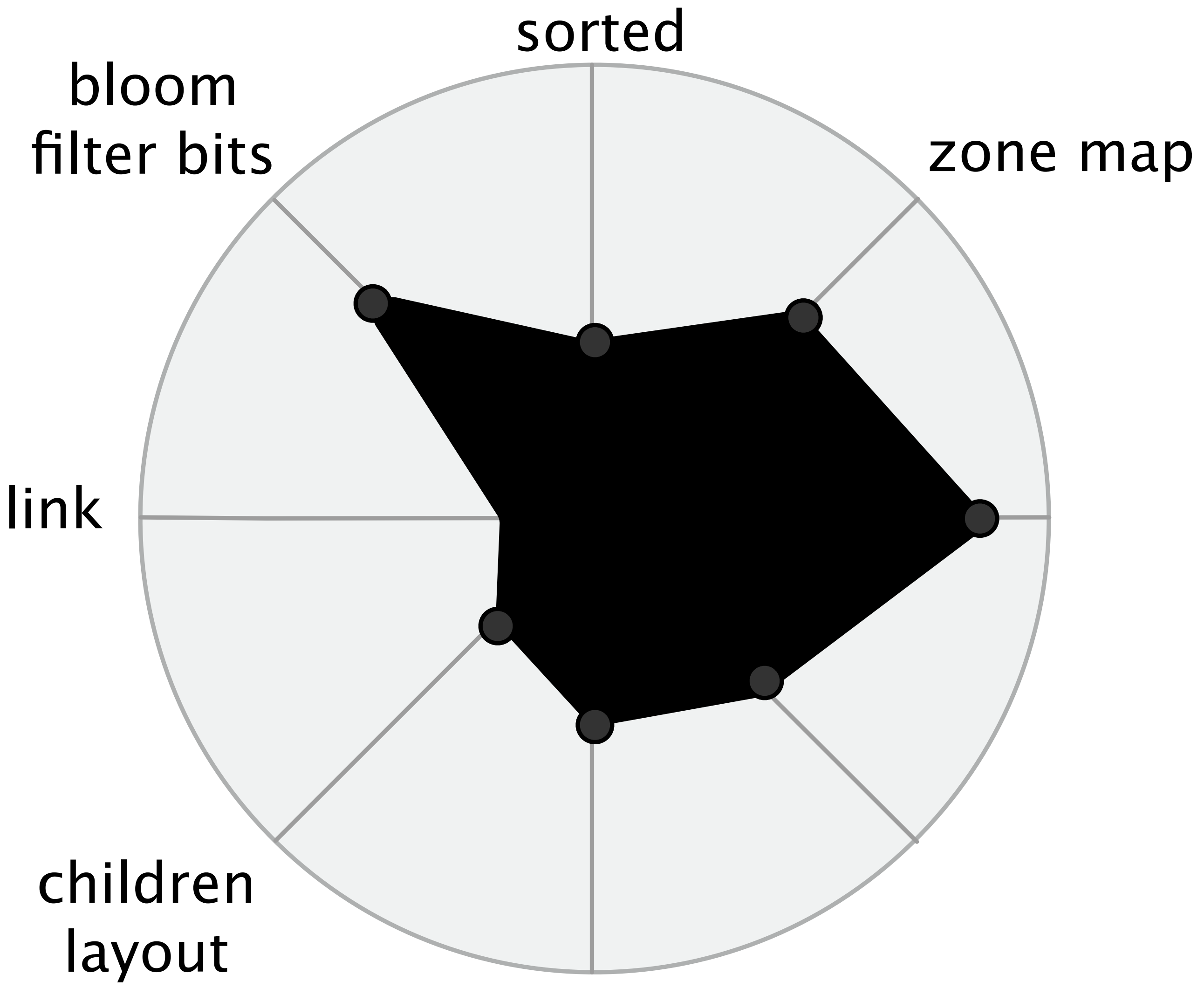
Filters layout (consolidate | scatter)

Links layout (consolidate | scatter)

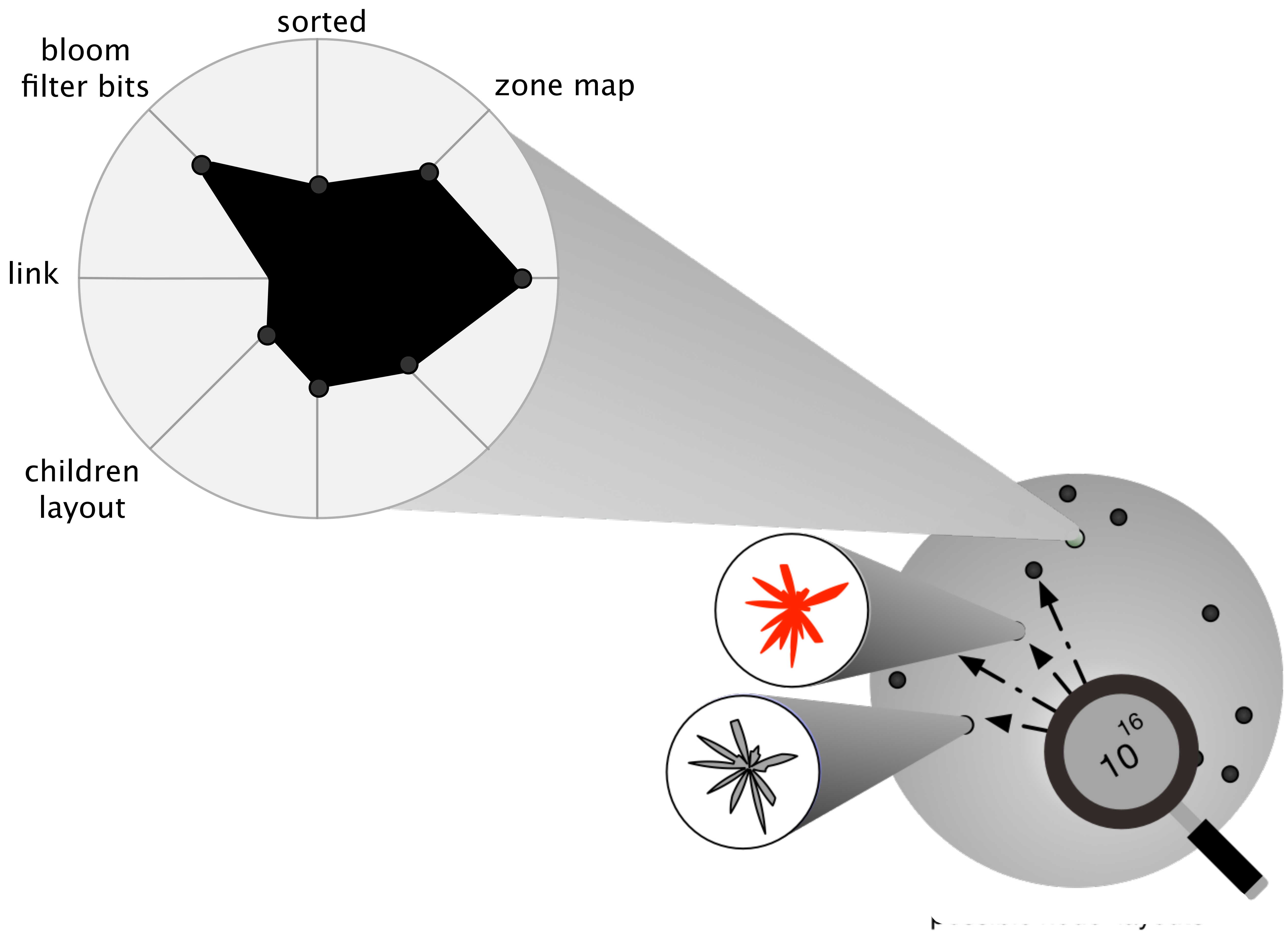
Physical location (inline | pointed | double- pointed)
Physical layout (BFS | scatter)



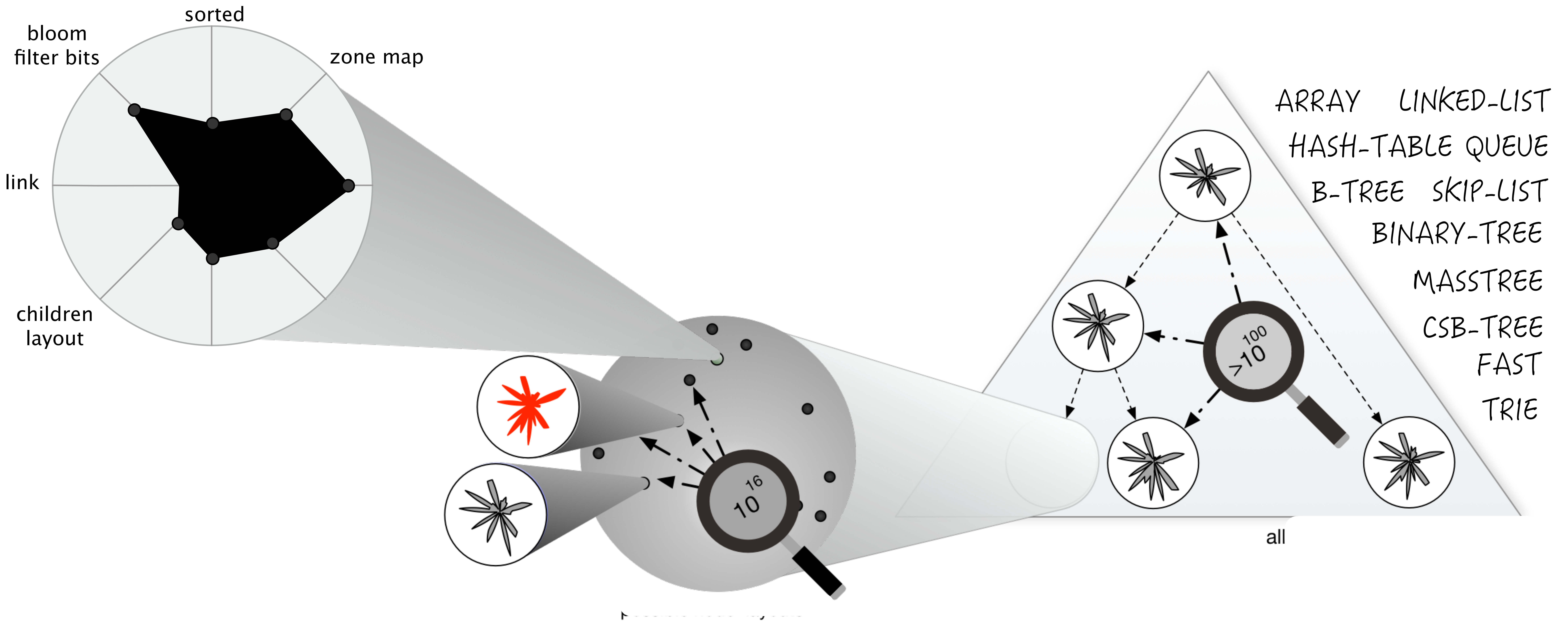
SETS OF CONCEPTS



SETS OF CONCEPTS   POSSIBLE NODE DESIGNS



SETS OF CONCEPTS POSSIBLE NODE DESIGNS POSSIBLE STRUCTURES



@IEEE.EngBul18

the periodic table of data structures

classes of designs *classes of primitives*	B-trees & Variants	Tries & Variants	LSM-Trees & Variants	Differential Files	Membership Tests	Zone maps & Variants	Bitmaps & Variants	Hashing	Base Data & Columns	
Partitioning	DONE	DONE	DONE					DONE	DONE	↓↑↑ R U M
Logarithmic Design	DONE	DONE	DONE							↓↓↑ R U M
Fractional Cascading	DONE		DONE	DONE						↓↑↑ R U M
Log-Structured	DONE		DONE	DONE						↑↓↑ R U M
Buffering	DONE			DONE			DONE			↓↕↑ R U M
Differential Updates	DONE			DONE						↑↓↓ R U M
Sparse Indexing	DONE				DONE	DONE				↓↕↑ R U M
Adaptivity	DONE								DONE	

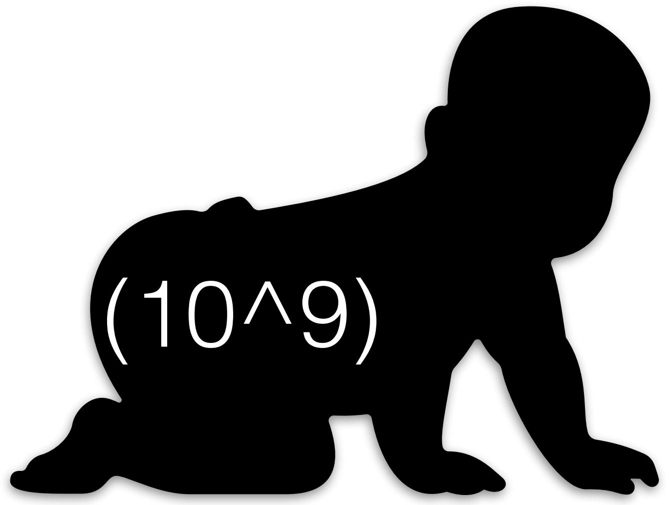
DASlab @ Harvard SEAS

@IEEE.EngBul18

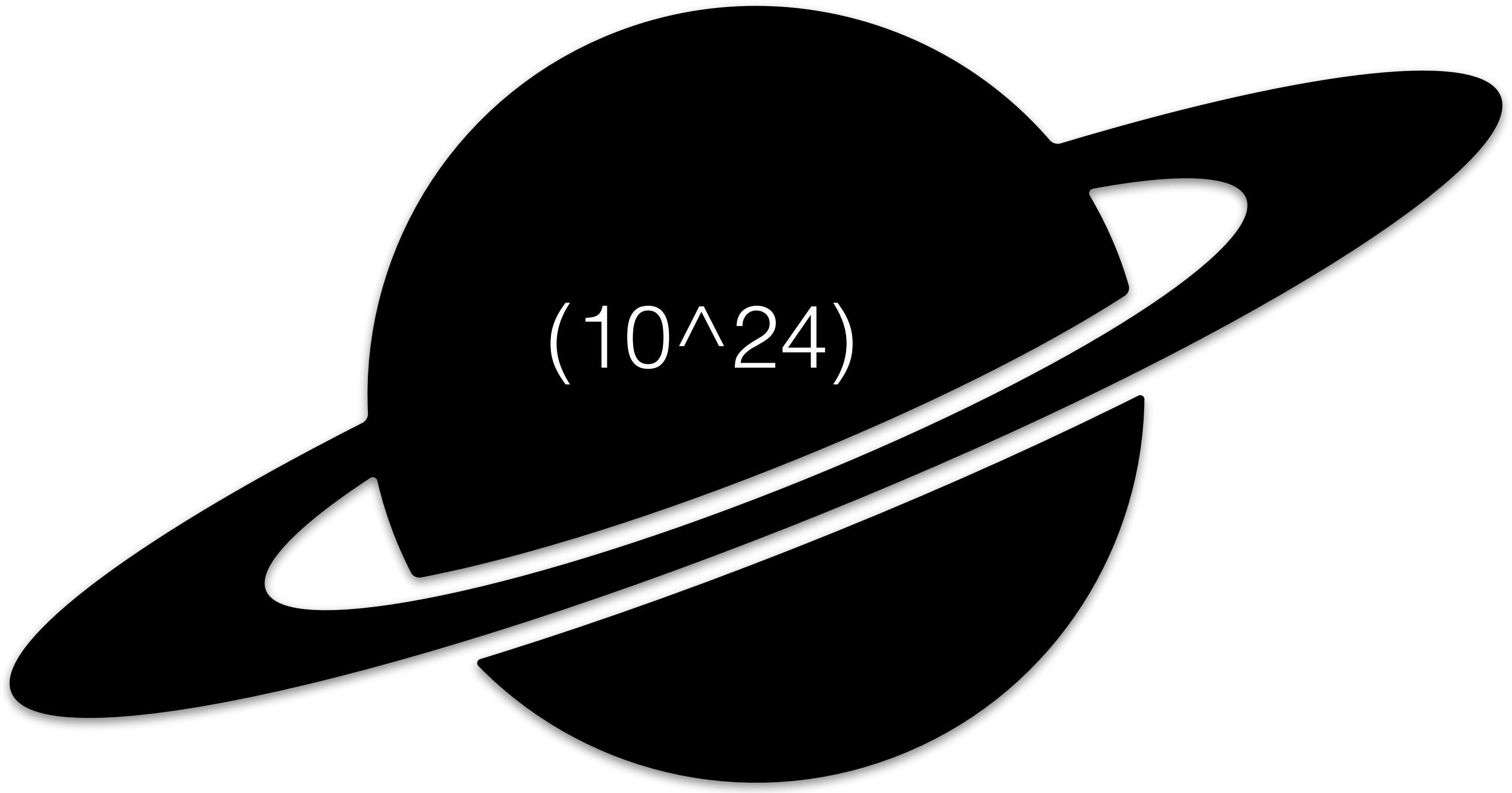
the periodic table of data structures

classes of designs *classes of primitives*	B-trees & Variants	Tries & Variants	LSM-Trees & Variants	Differential Files	Membership Tests	Zone maps & Variants	Bitmaps & Variants	Hashing	Base Data & Columns	
Partitioning	DONE	DONE	DONE					DONE	DONE	↓↑↑ R U M
Logarithmic	DONE	DONE	DONE							↓↓↑
Differential Updates	DONE			DONE						↑↓↓ R U M
Sparse Indexing	DONE				DONE	DONE				↓↕↑ R U M
Adaptivity	DONE								DONE	

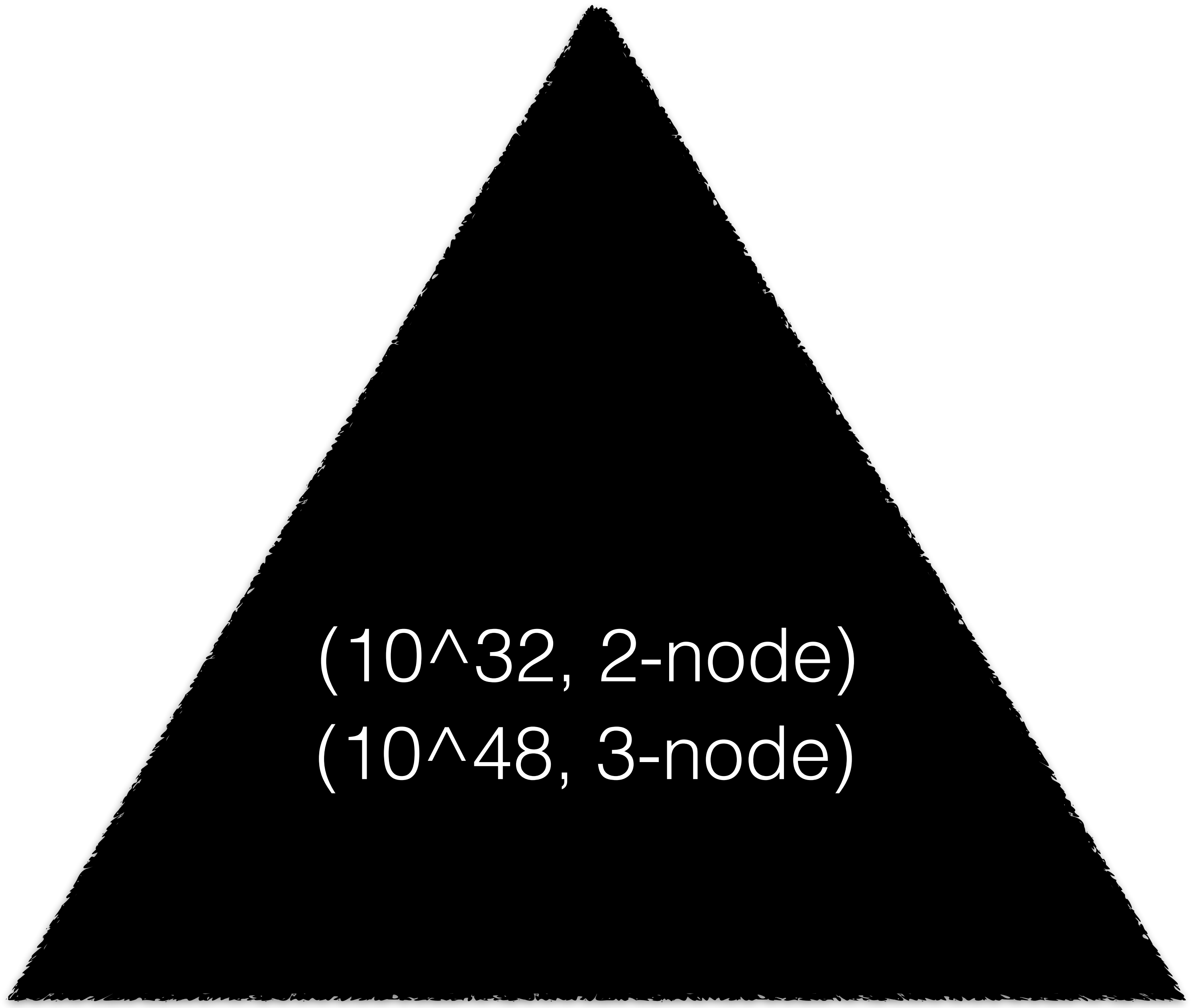
PAPER MACHINE



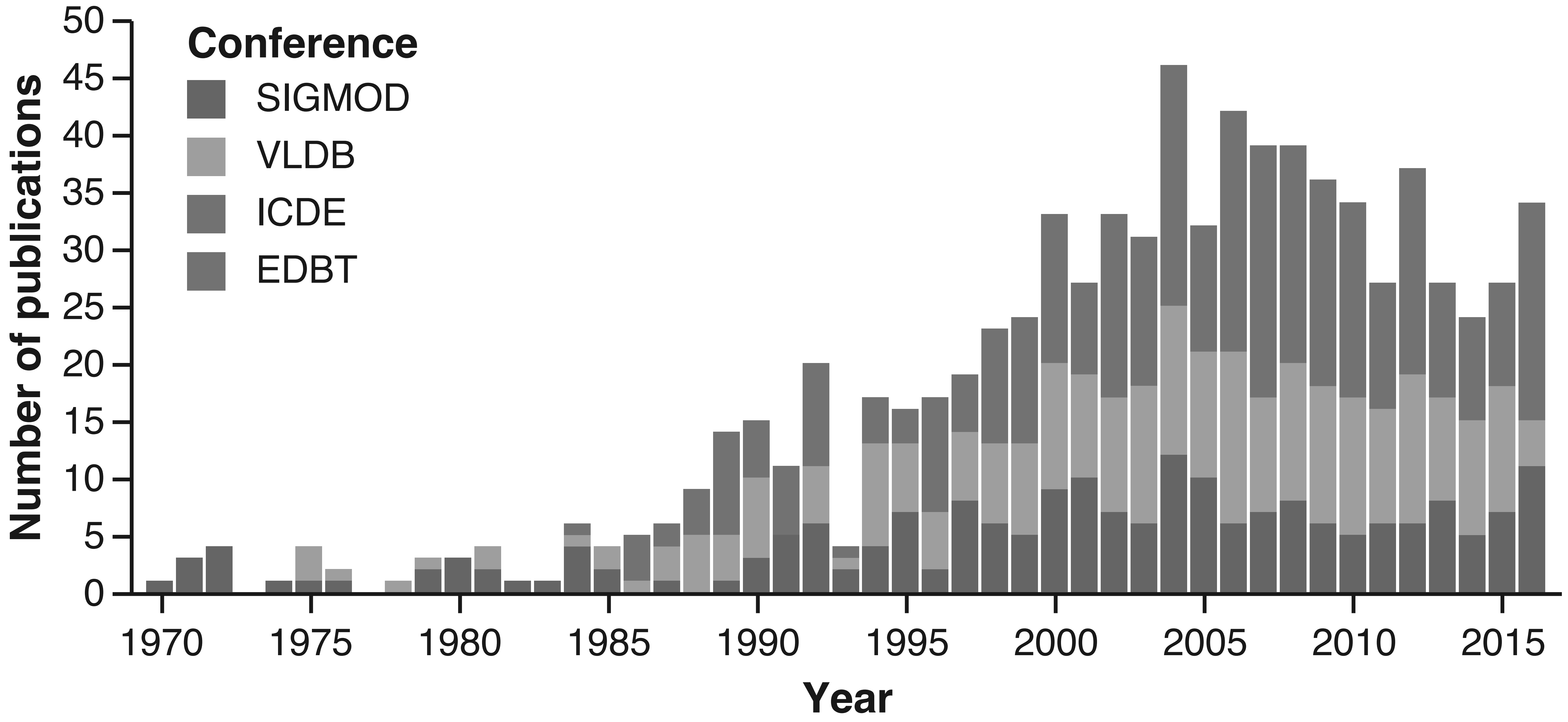
PEOPLE ON EARTH



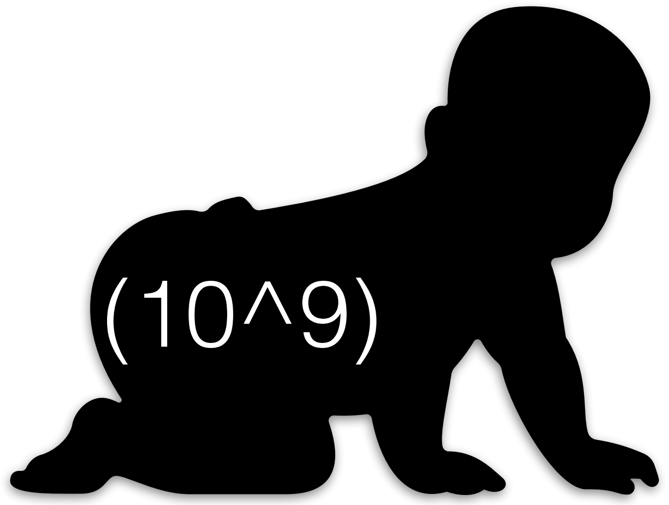
STARS IN THE SKY

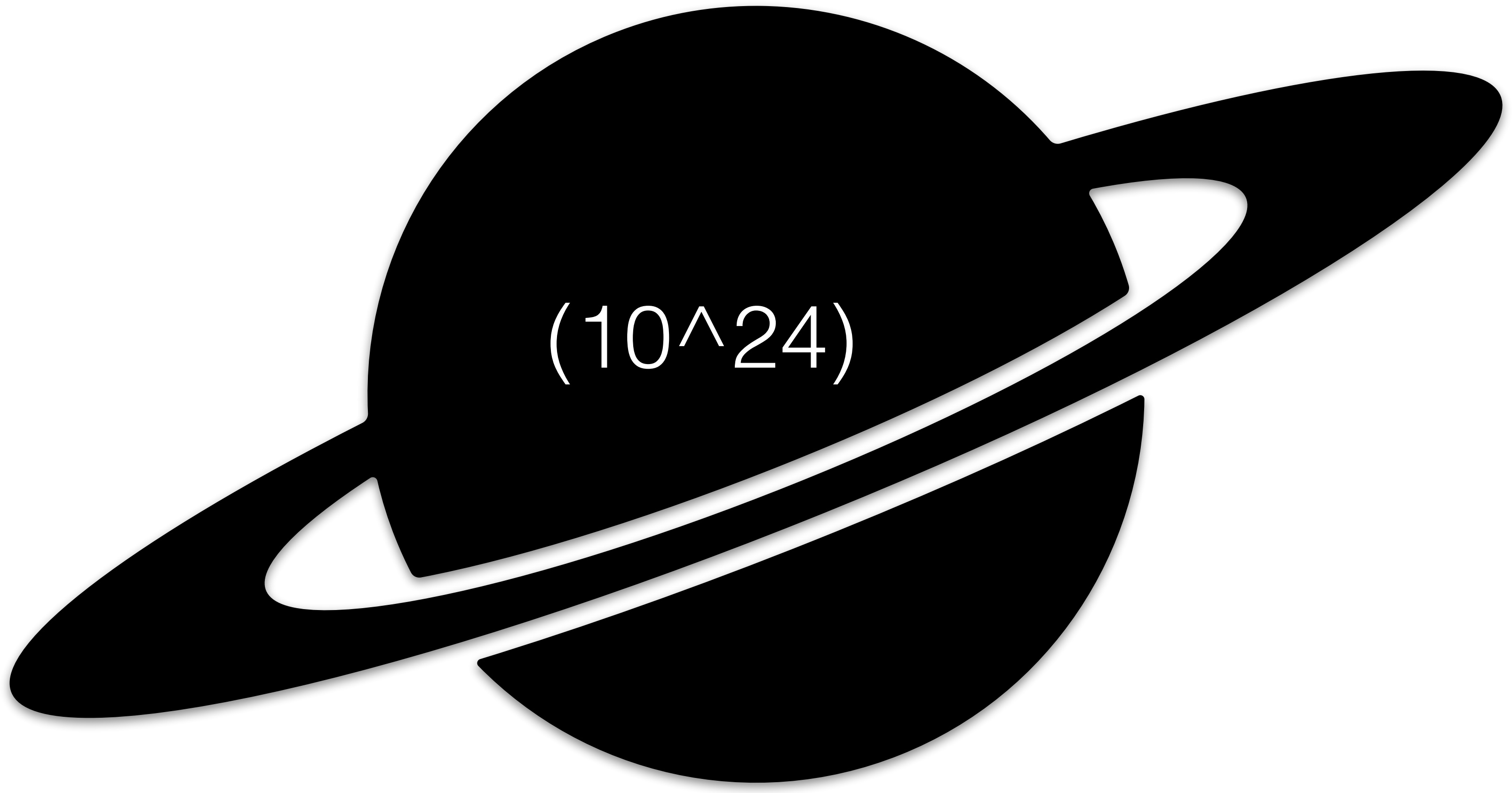


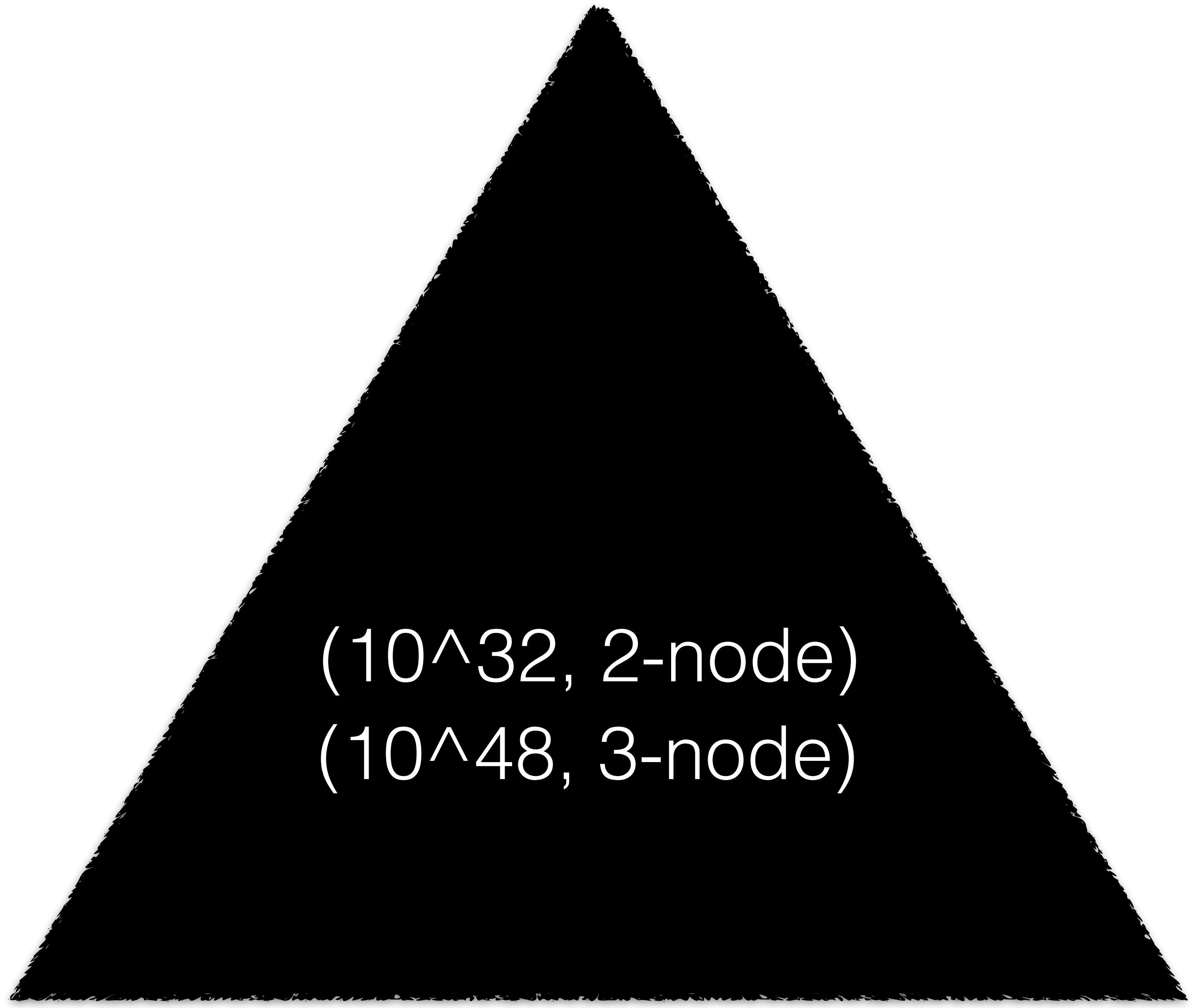
POSSIBLE DATA STRUCTURES



~5K since the dawn of CS







PEOPLE ON EARTH

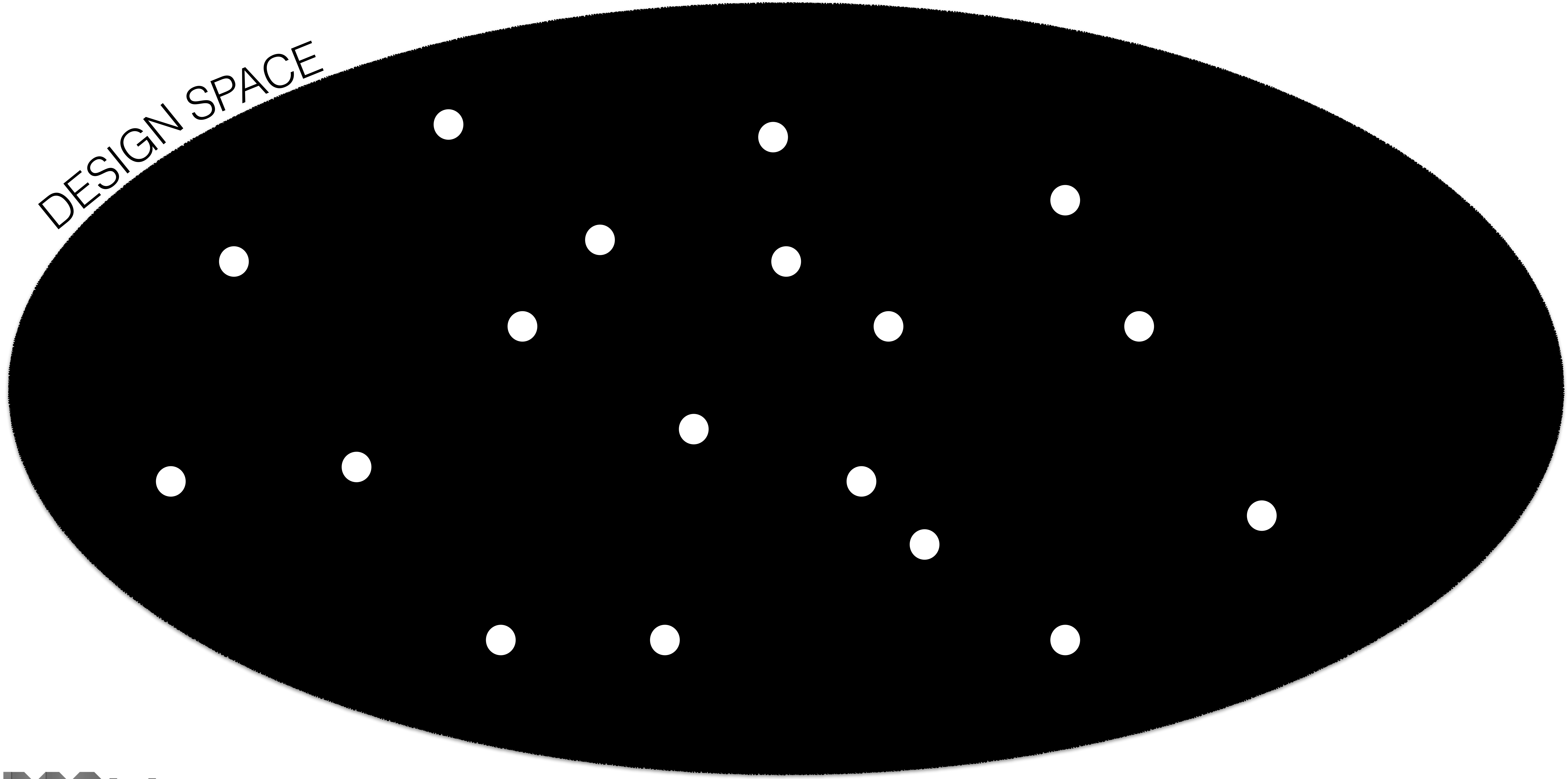
STARS IN THE SKY

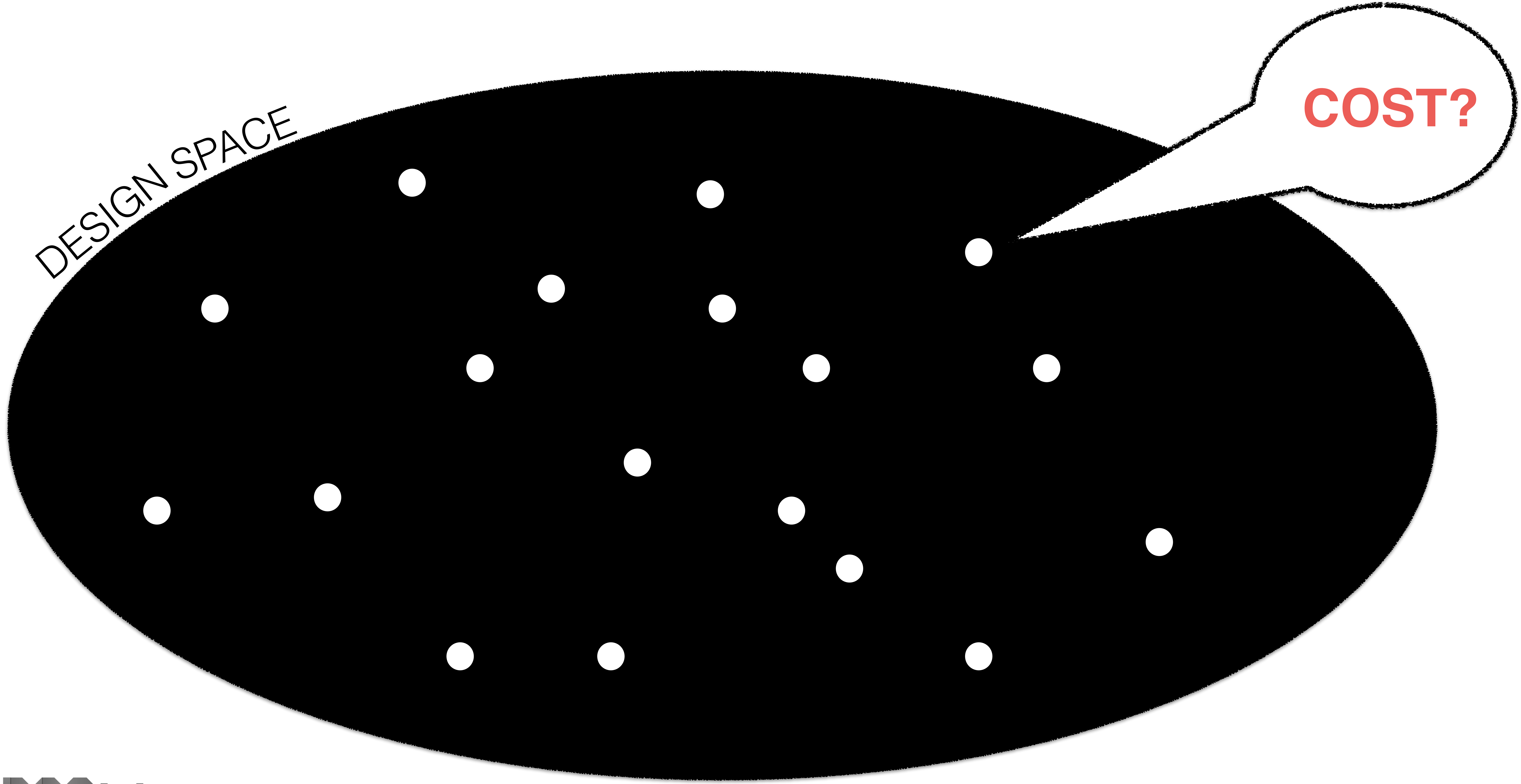
POSSIBLE DATA STRUCTURES

DESIGN SPACE

if we know the fundamental building blocks, how they combine and their **properties**,

then we can automate the discovery of novel combinations and tunings


DESIGN SPACE


DESIGN SPACE
COST?

DESIGN SPACE

COST SYNTHESIS

HOW TO USE

HOW TO JUDGE A DESIGN?

1

COMPLEXITY ANALYSIS

2

IMPLEMENTATION & TESTING

3

GENERALIZED MODELS

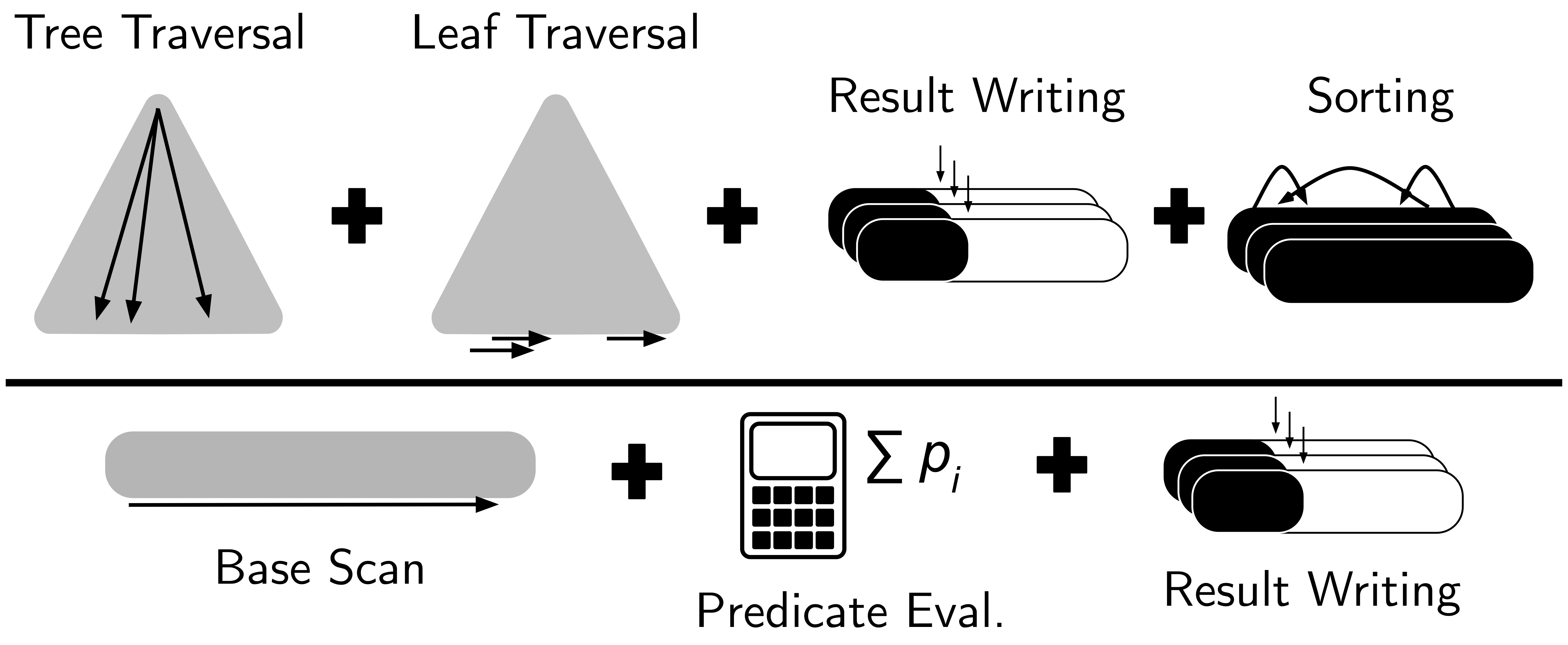
HARD & SLOW

$$APS(q,S_{tot}) = \frac{q \cdot \frac{1+\lceil log_b(N)\rceil}{N} \cdot \left(BW_S \cdot C_M + \frac{b \cdot BW_S \cdot C_A}{2} + \frac{b \cdot BW_S \cdot f_p \cdot p}{2}\right)}{max\left(ts, 2 \cdot f_p \cdot p \cdot q \cdot BW_S\right) + S_{tot} \cdot rw \cdot \frac{BW_S}{BW_R}}$$

$$+ \frac{S_{tot}\left(\frac{BW_S \cdot C_M}{b} + (aw+ow) \cdot \frac{BW_S}{BW_I} + rw \cdot \frac{BW_S}{BW_R}\right)}{max\left(ts, 2 \cdot f_p \cdot p \cdot q \cdot BW_S\right) + S_{tot} \cdot rw \cdot \frac{BW_S}{BW_R}}$$

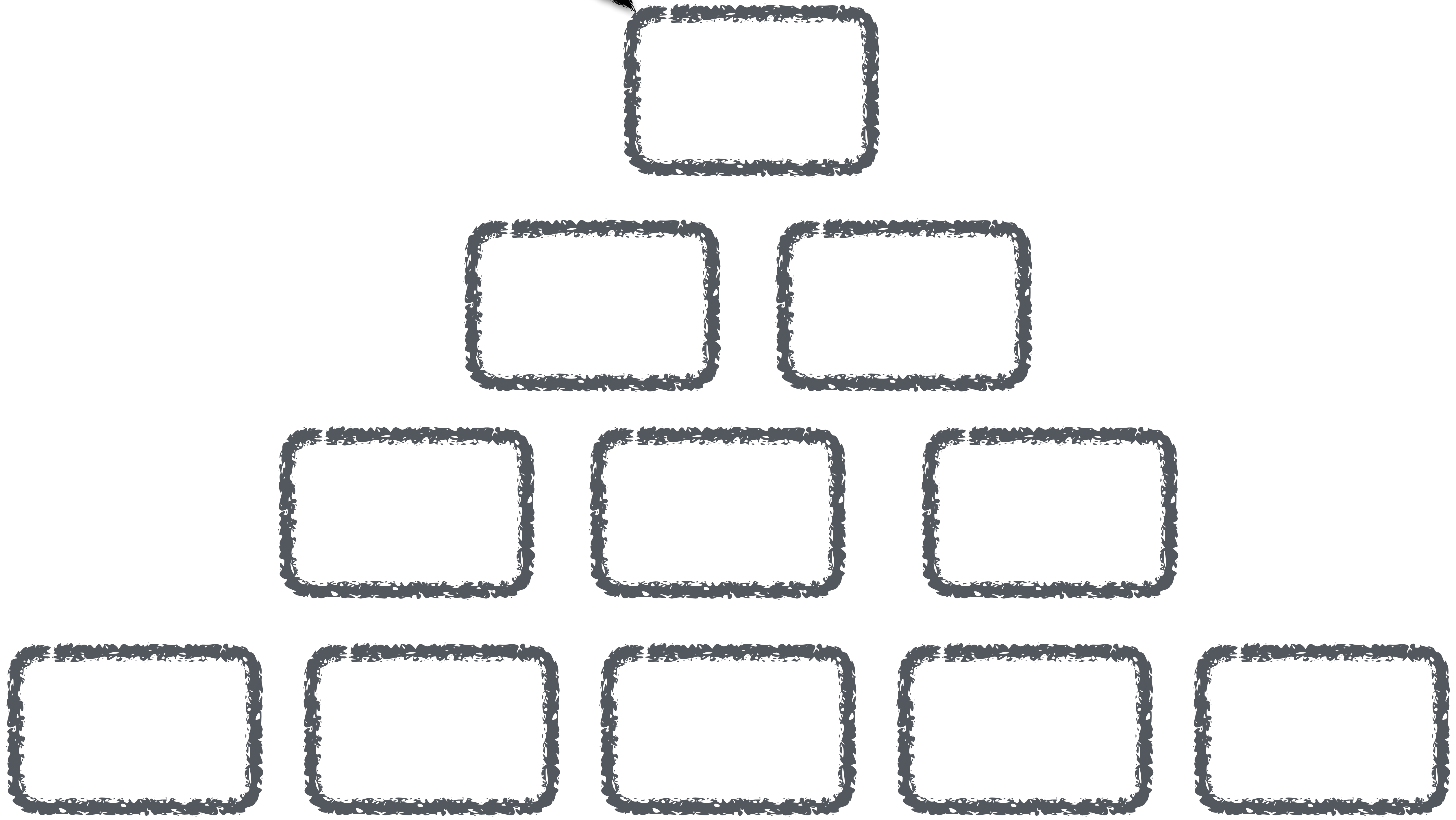
$$+ \frac{S_{tot} \cdot log_2\left(S_{tot} \cdot N\right) \cdot BW_S \cdot C_A}{max\left(ts, 2 \cdot f_p \cdot p \cdot q \cdot BW_S\right) + S_{tot} \cdot rw \cdot \frac{BW_S}{BW_R}}$$

Access path selection @SIGMOD2017

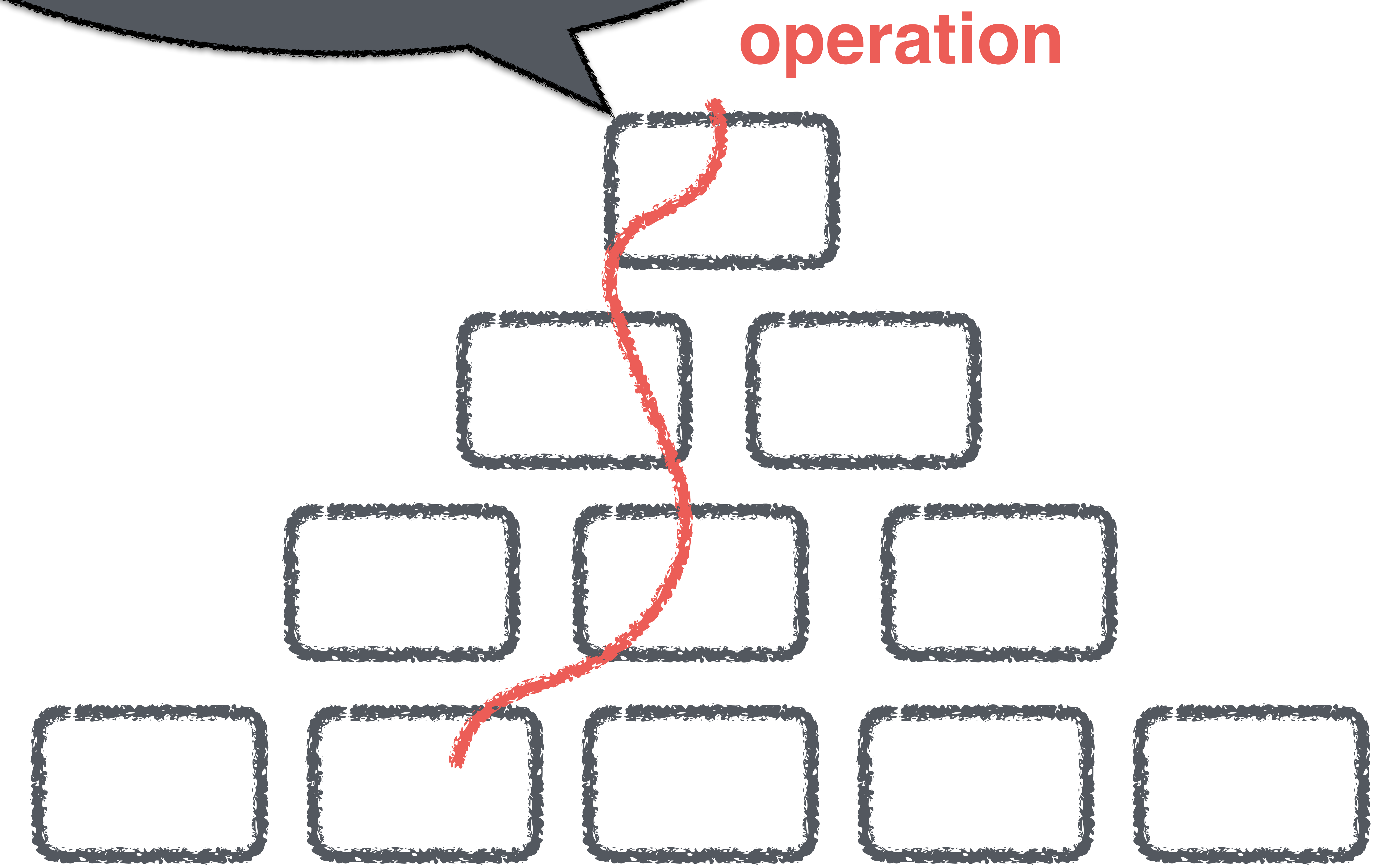


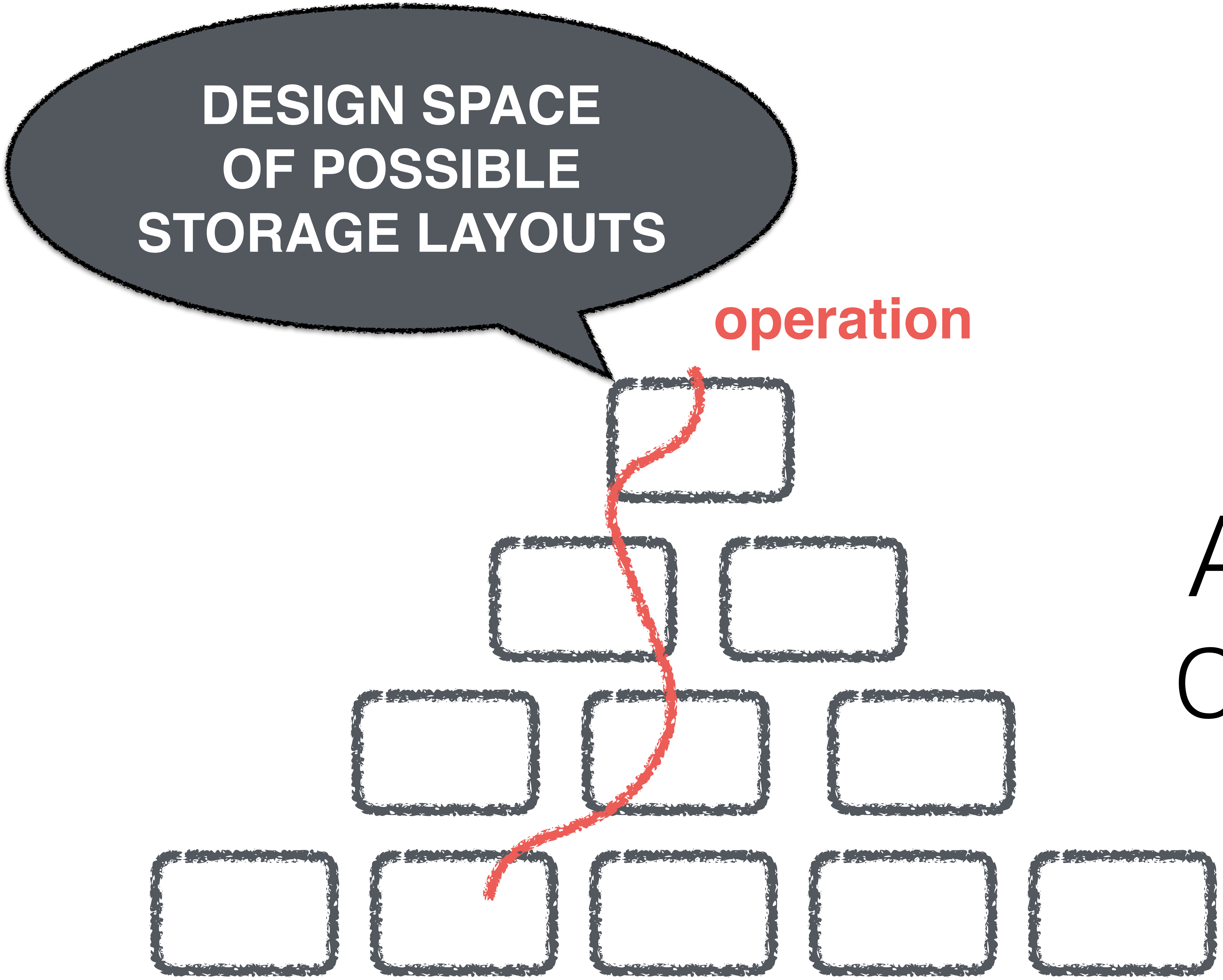
Workload	q	number of queries
	s_i	selectivity of query i
	S_{tot}	total selectivity of the workload
Dataset	N	data size (tuples per column)
	ts	tuple size (bytes per tuple)
Hardware	C_A	L1 cache access (sec)
	C_M	LLC miss: memory access (sec)
	BW_S	scanning bandwidth (GB/s)
	BW_R	result writing bandwidth (GB/s)
	BW_I	leaf traversal bandwidth (GB/s)
	p	The inverse of CPU frequency
	f_p	Factor accounting for pipelining
Scan & Index	rw	result width (bytes per output tuple)
	b	tree fanout
	aw	attribute width (bytes of the indexed column)
	ow	offset width (bytes of the index column offset)

DESIGN SPACE
OF POSSIBLE
STORAGE LAYOUTS



DESIGN SPACE
OF POSSIBLE
STORAGE LAYOUTS

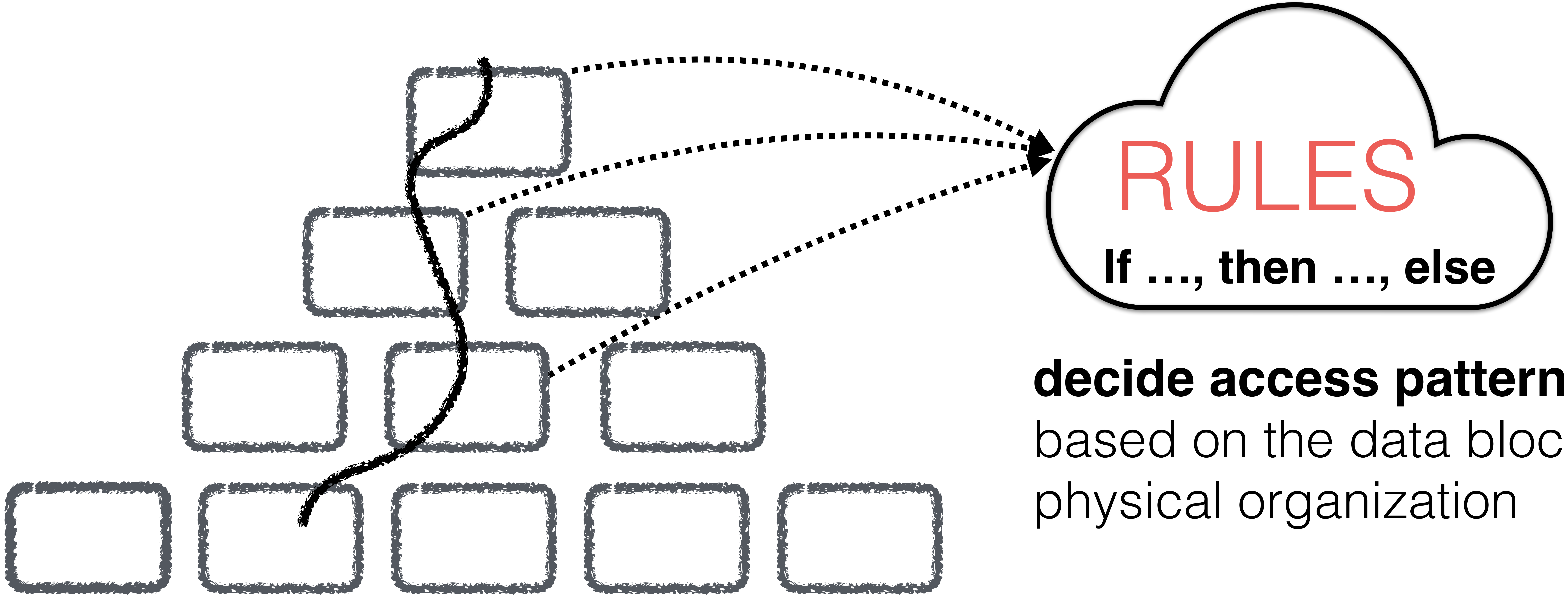




DESIGN SPACE
OF POSSIBLE
STORAGE LAYOUTS

operation

ALGORITHM & COST SYNTHESIS

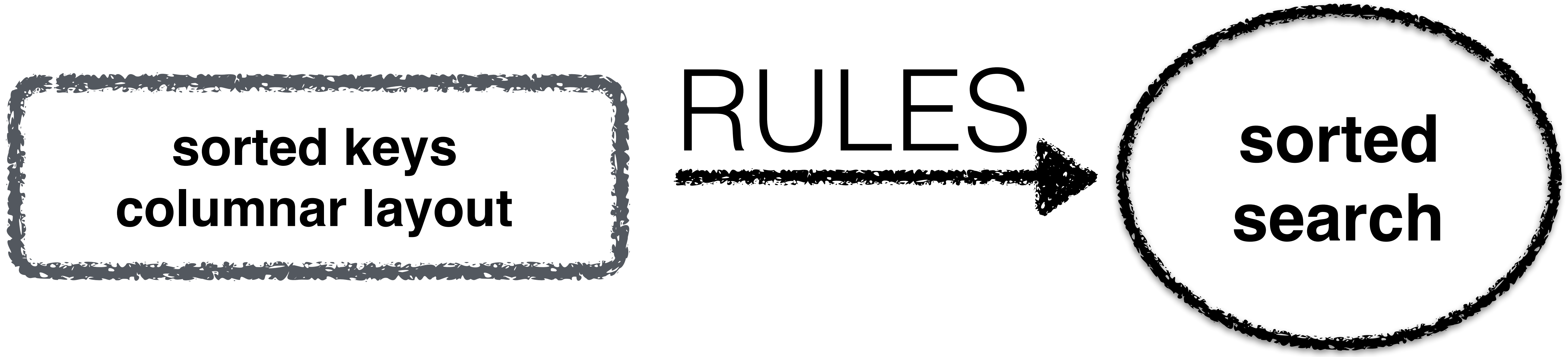


RULES

If …, then …, else

decide access pattern
based on the data block’s
physical organization

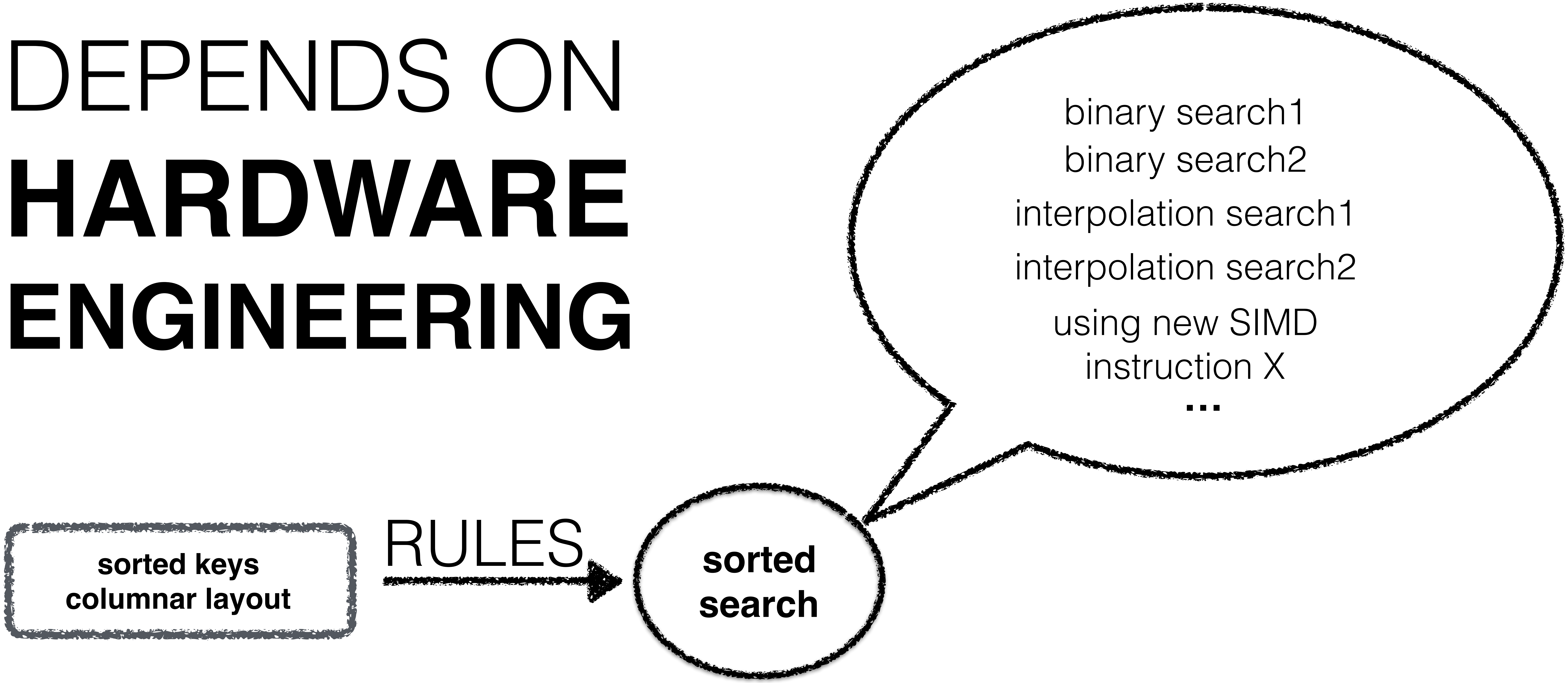
sorted keys
columnar layout


sorted keys
columnar layout

RULES

sorted
search

DEPENDS ON **HARDWARE ENGINEERING**



sorted keys columnar layout

RULES

sorted search

binary search1

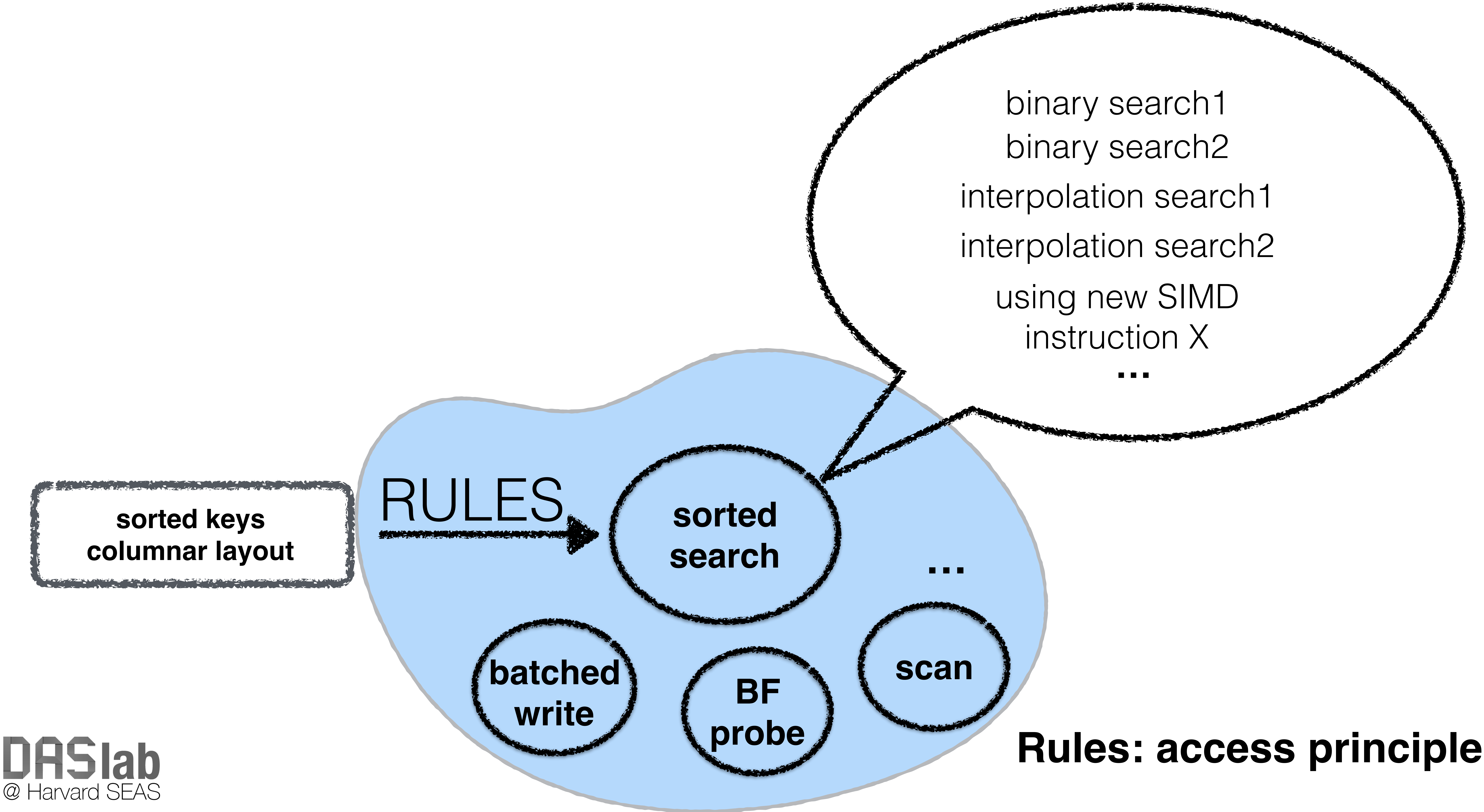
binary search2

interpolation search1

interpolation search2

using new SIMD instruction X

...



sorted keys
columnar layout

RULES

sorted search

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

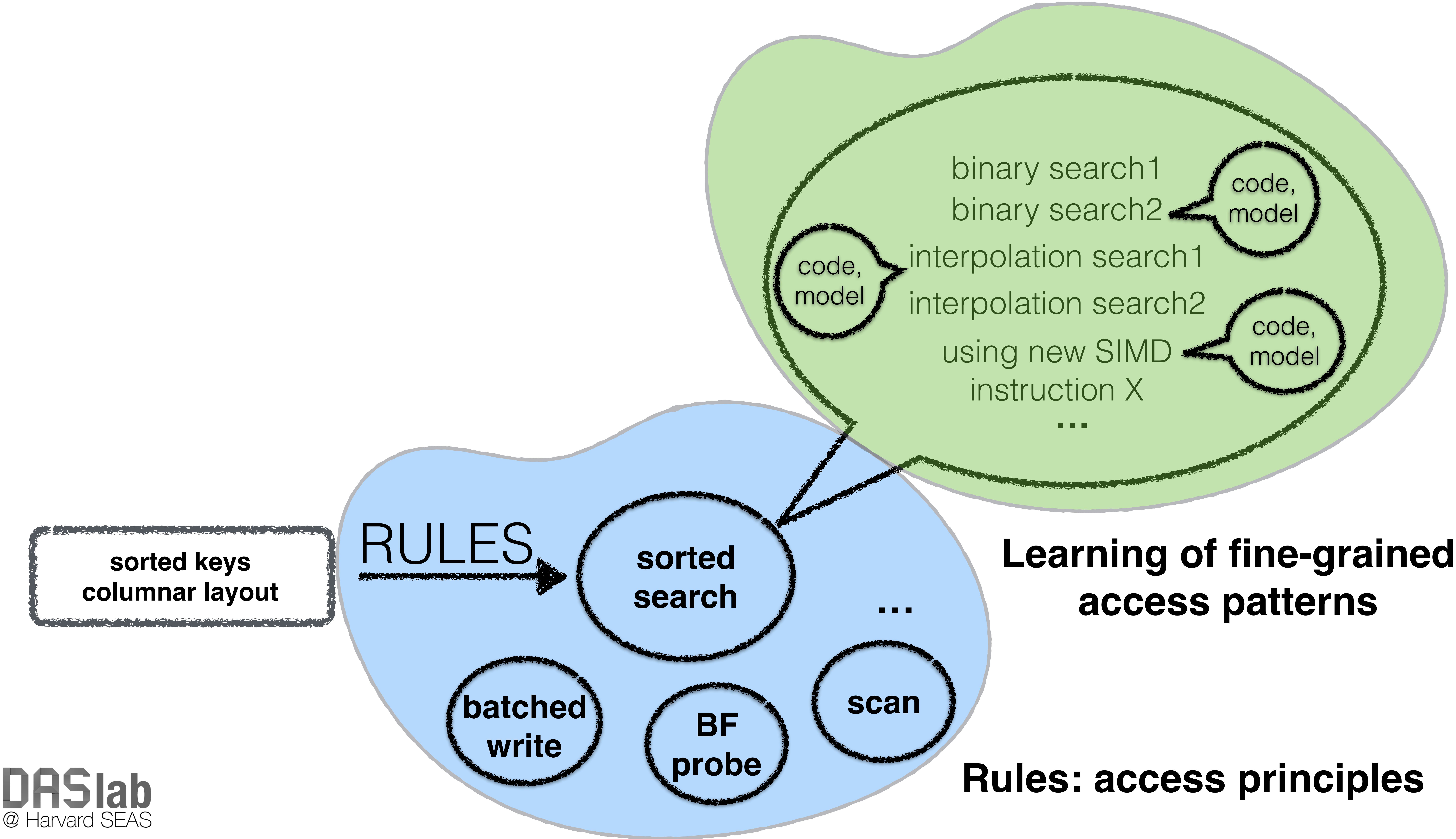
batched write

BF probe

scan

...

Rules: access principles



sorted keys
columnar layout

RULES

sorted search

batched write

BF probe

scan

...

binary search1

binary search2

interpolation search1

interpolation search2

using new SIMD instruction X

...

code, model

code, model

code, model

Learning of fine-grained access patterns

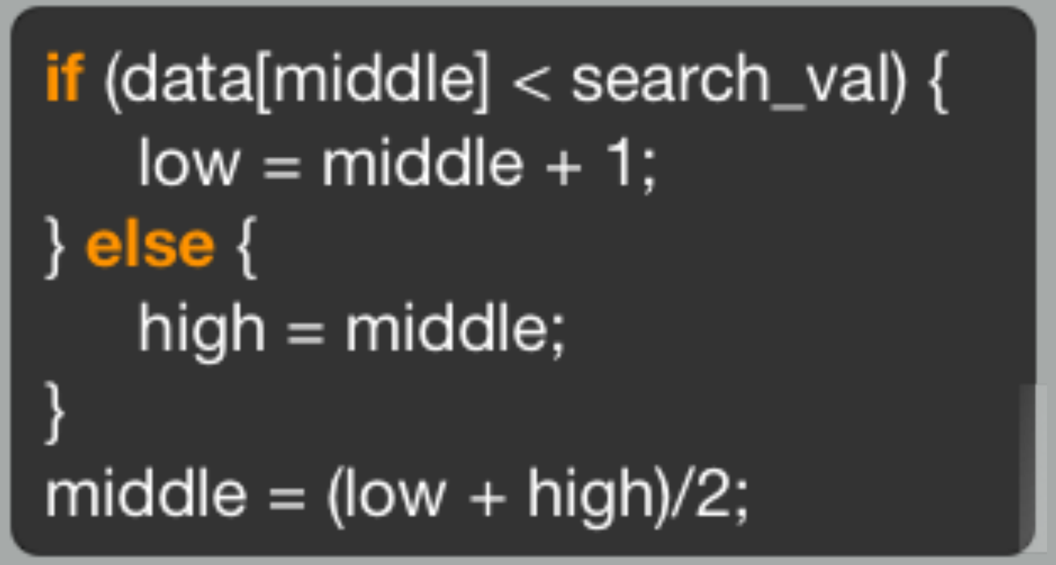
Rules: access principles

SYNTHESIS FROM LEARNED MODELS

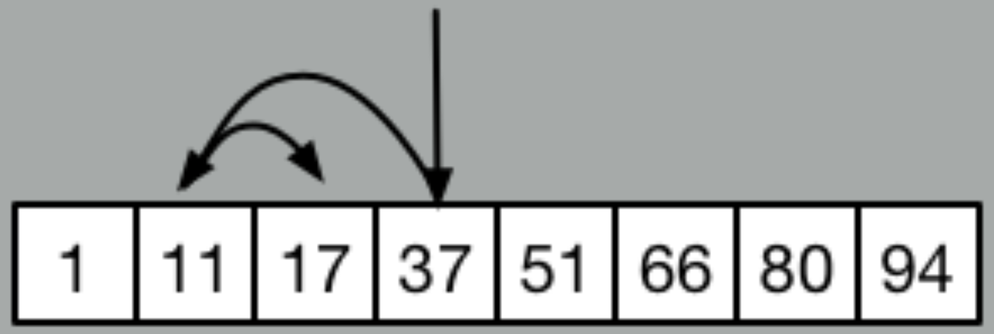
coding, modeling, generalized models, and a touch of ML

1. MINIMAL CODE

e.g., binary search



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



1	11	17	37	51	66	80	94

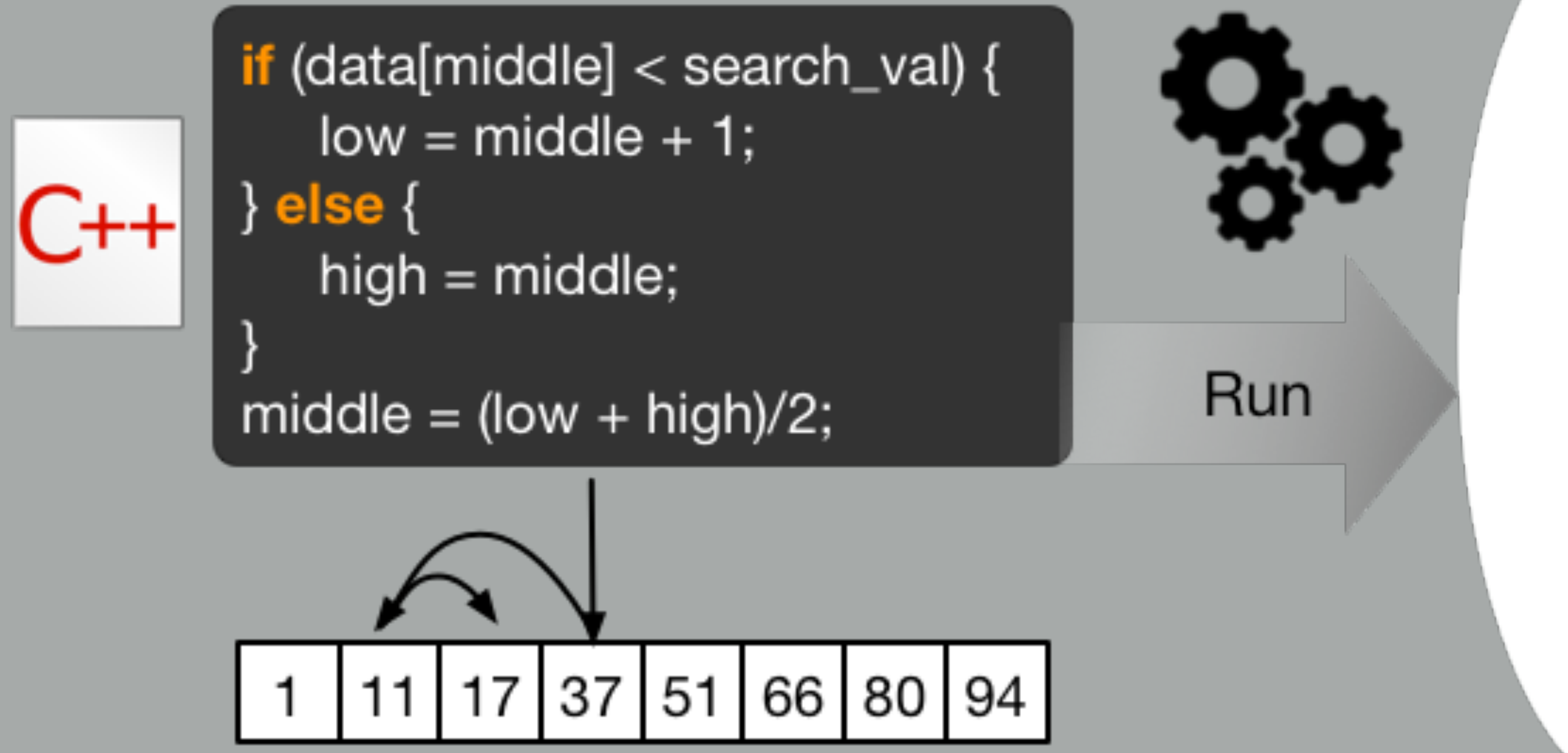
DASlab @ Harvard SEAS

SYNTHESIS FROM LEARNED MODELS

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

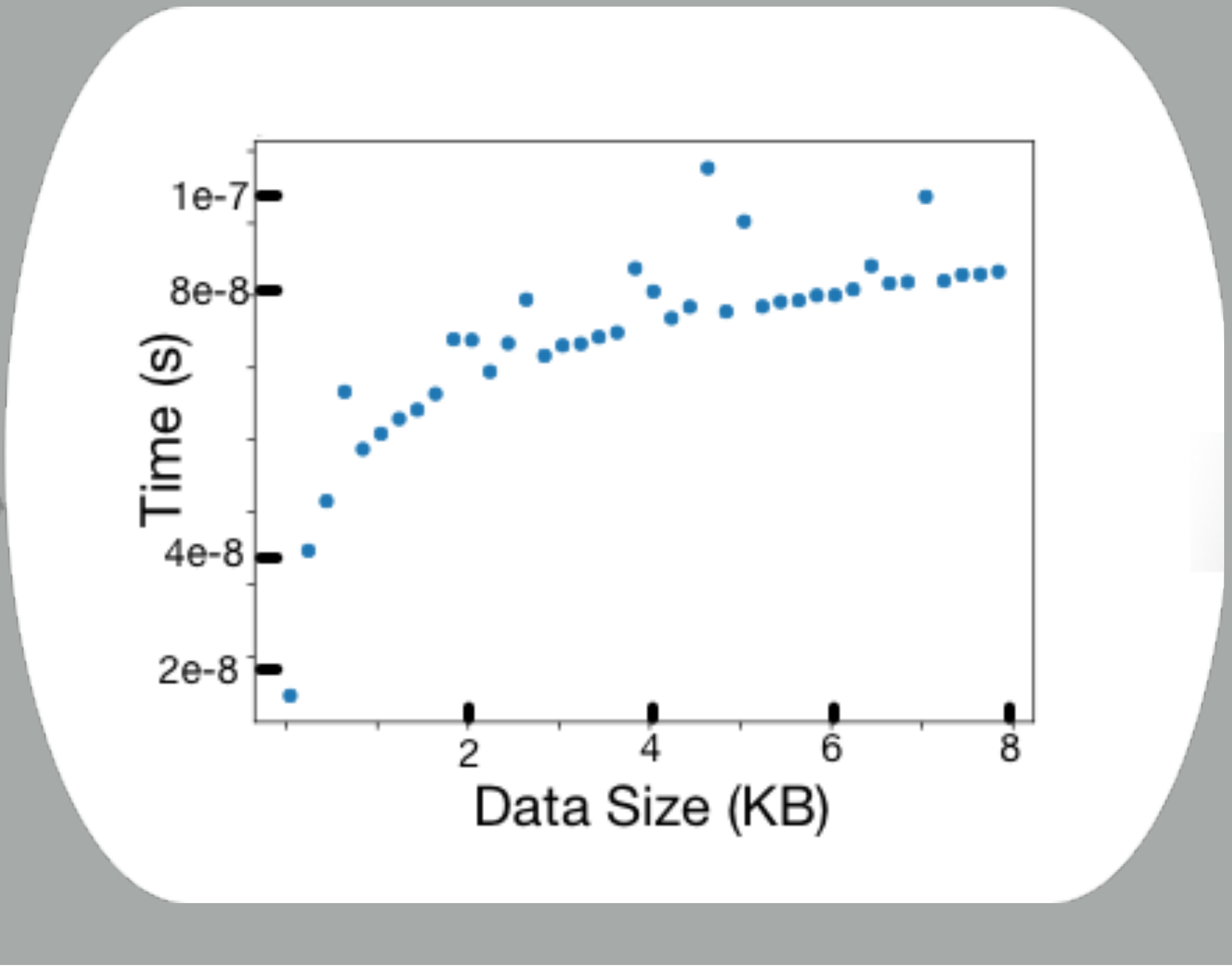
1. MINIMAL CODE

e.g., binary search



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

2. BENCHMARK



DASlab @ Harvard SEAS

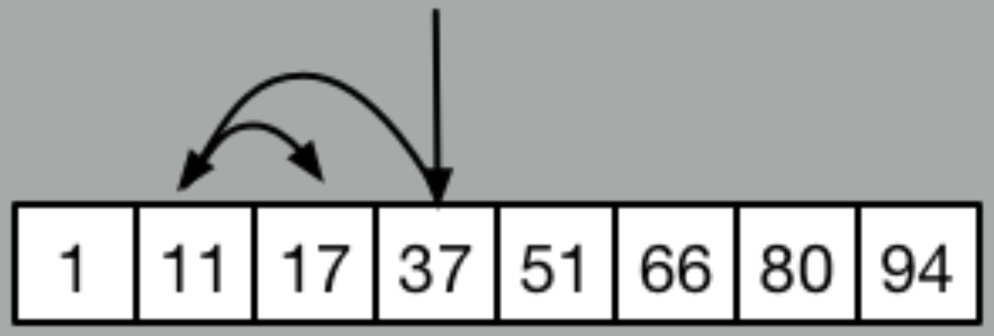
SYNTHESIS FROM LEARNED MODELS

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

1. MINIMAL CODE

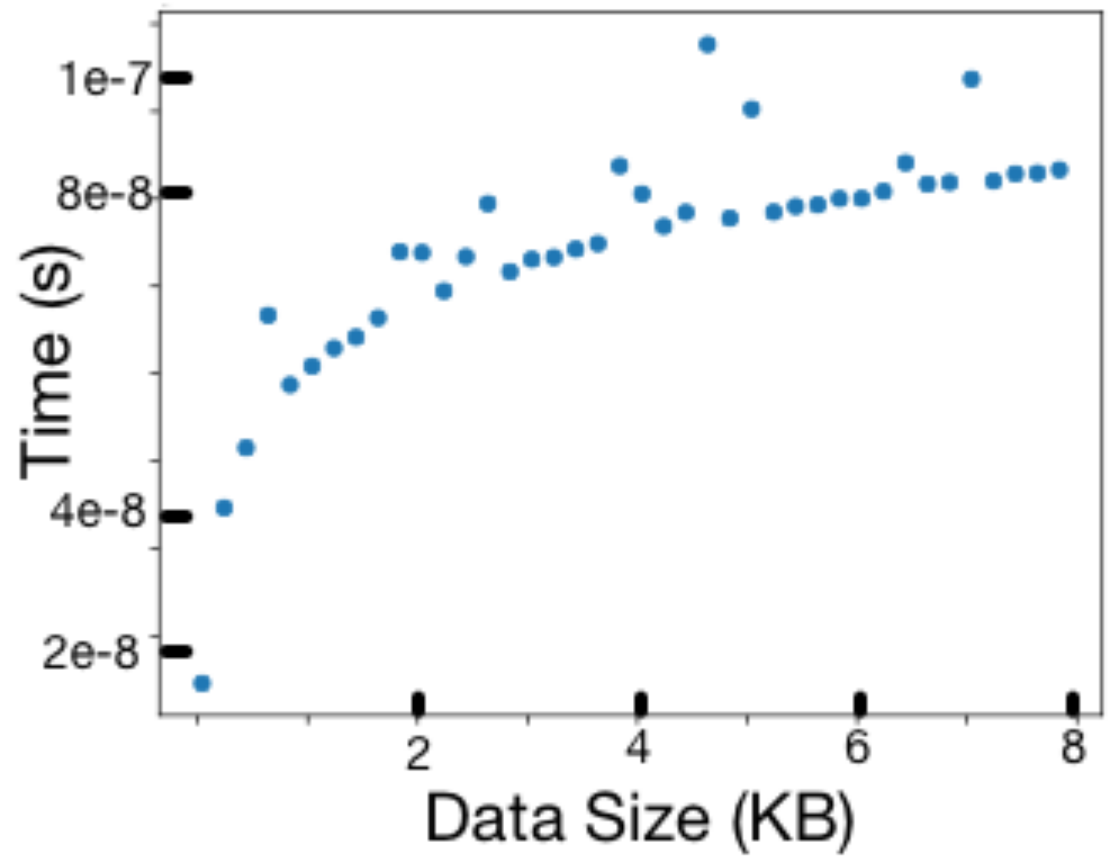
e.g., binary search

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



Run

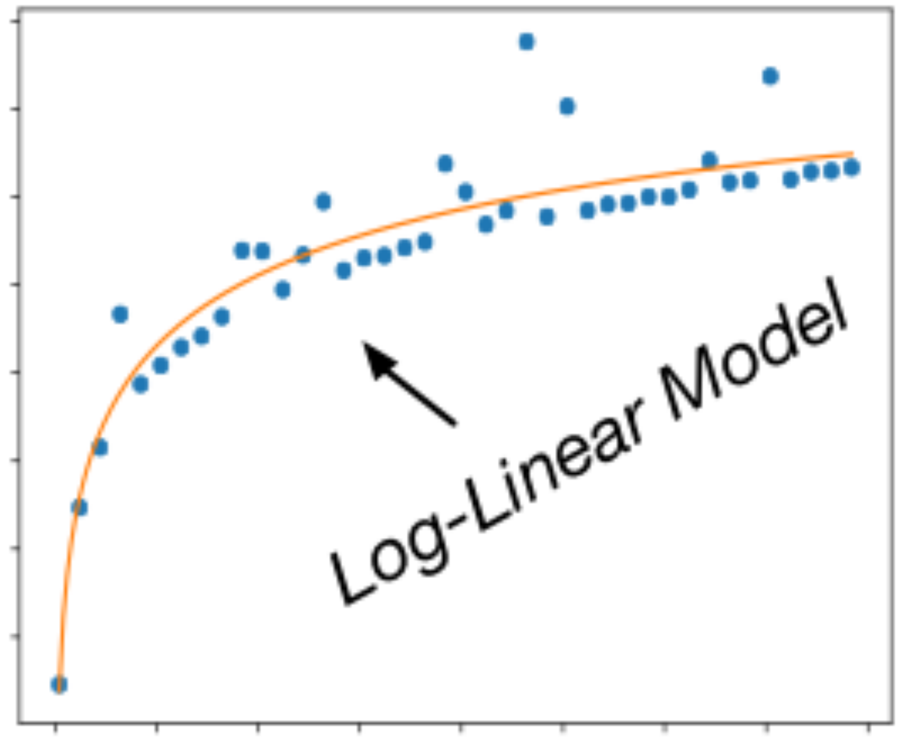
2. BENCHMARK



f(x)

Train

3. FIT MODEL



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

DASlab @ Harvard SEAS

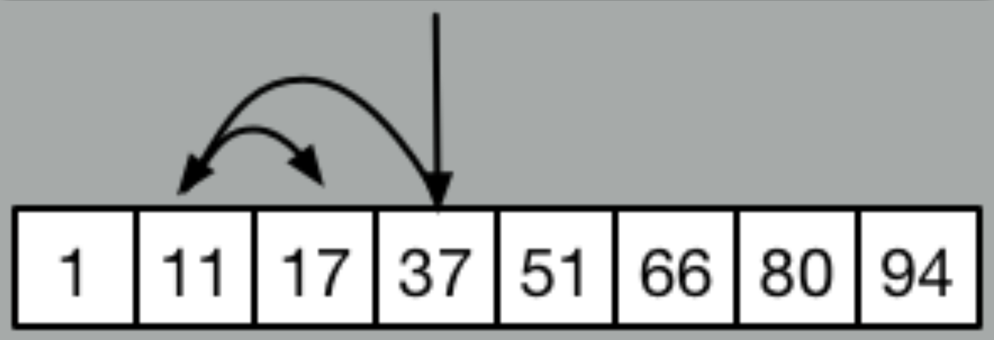
SYNTHESIS FROM LEARNED MODELS

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

1. MINIMAL CODE

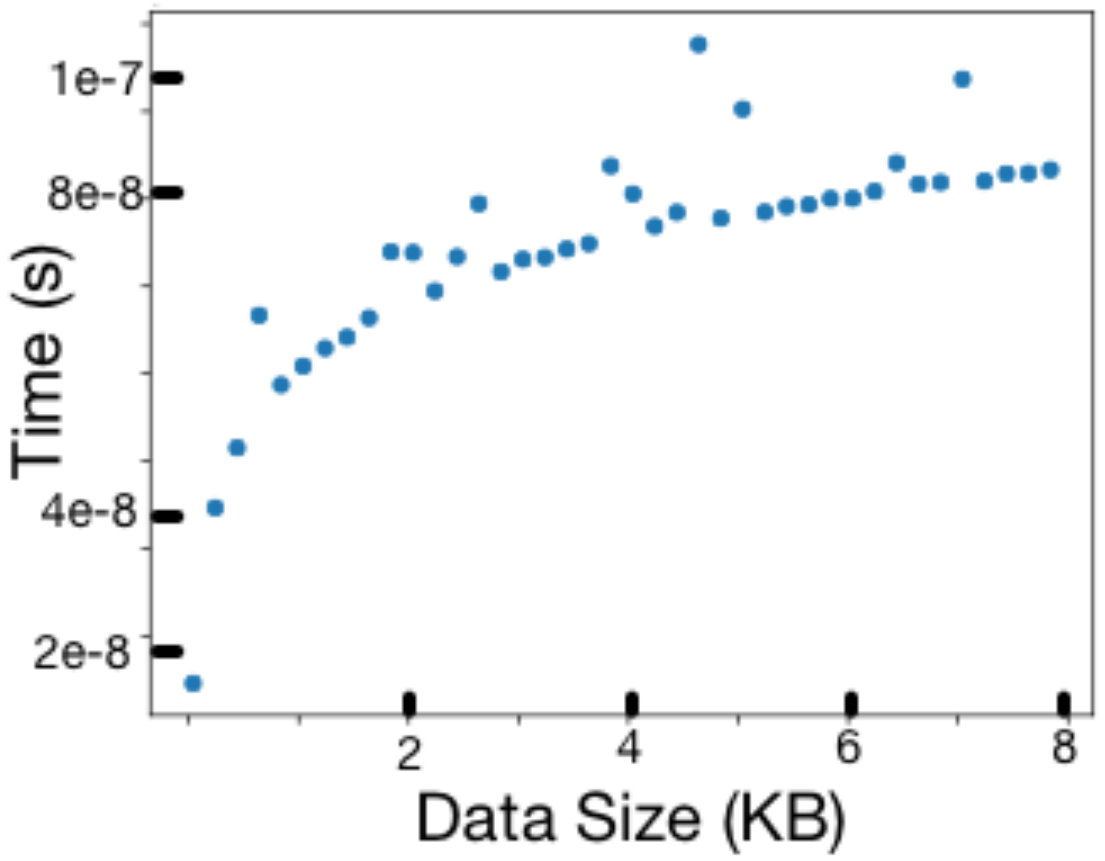
e.g., binary search

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



Run

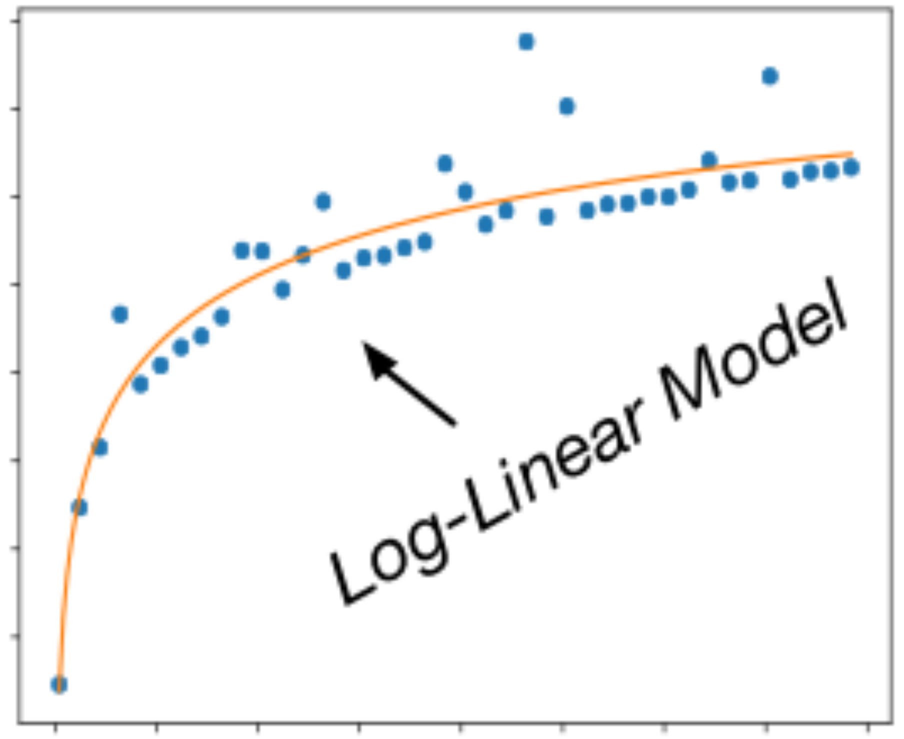
2. BENCHMARK



$f(x)$

Train

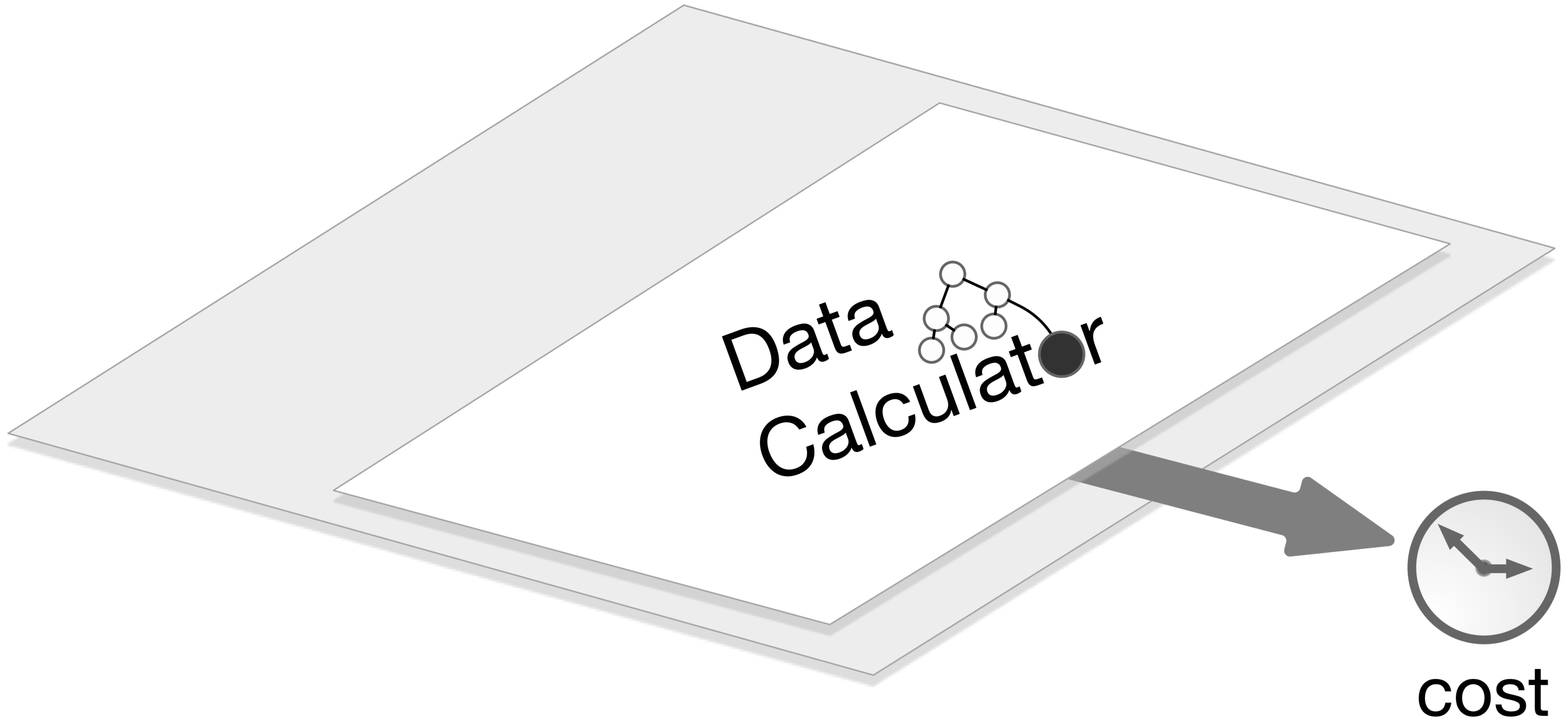
3. FIT MODEL



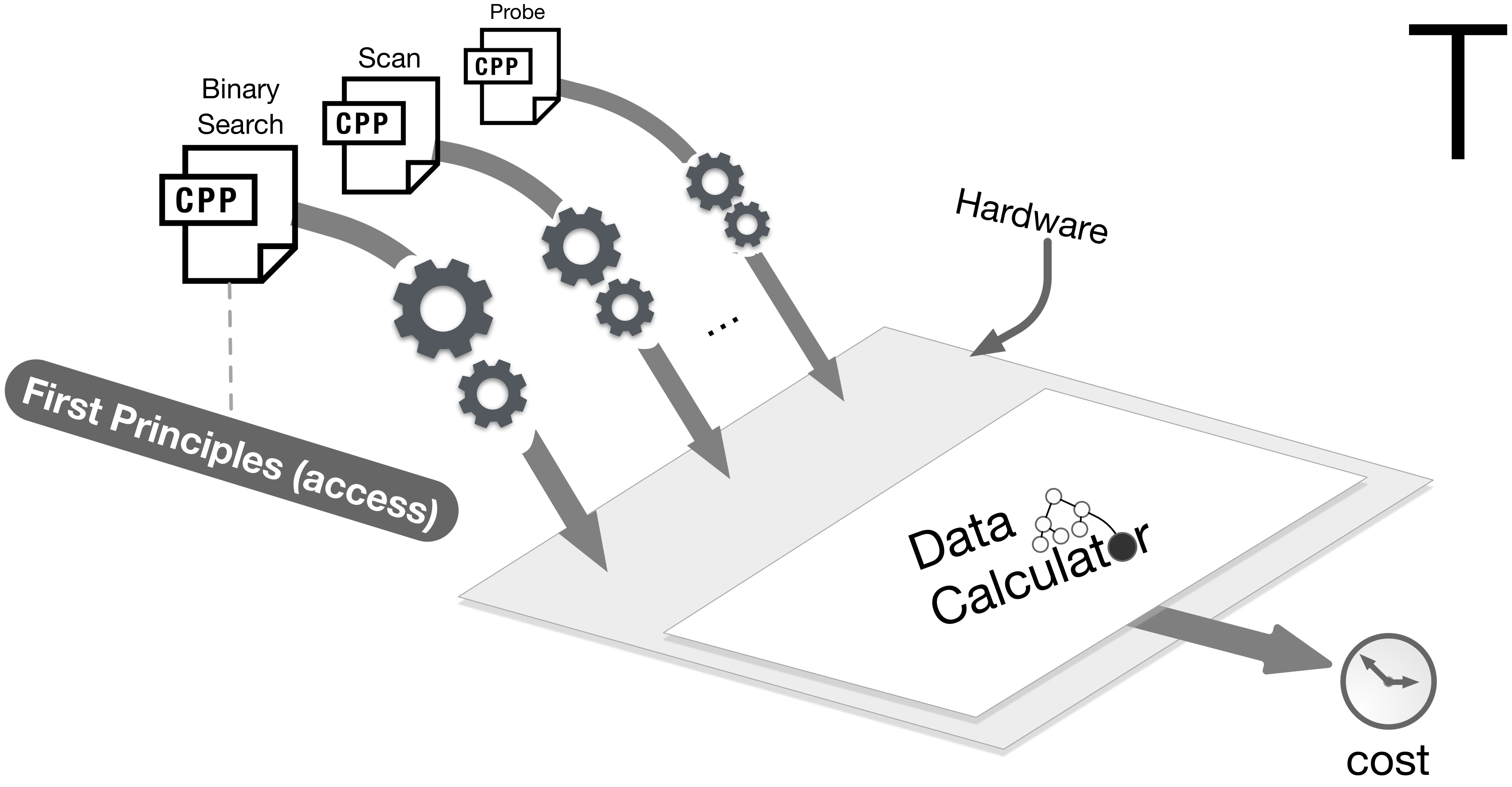
$$f(x) = ax + b\log x + c$$

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

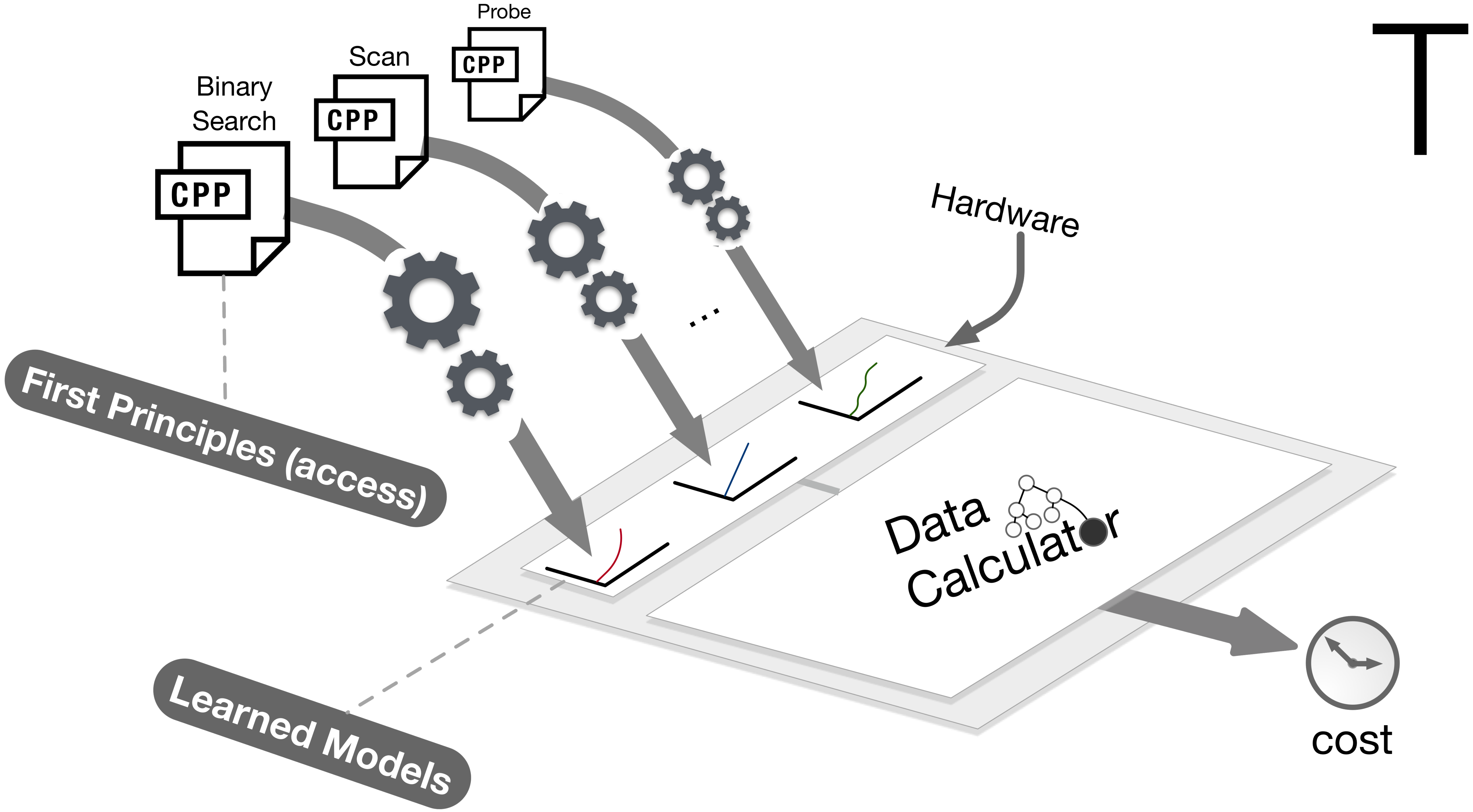
TRAINING



TRAINING

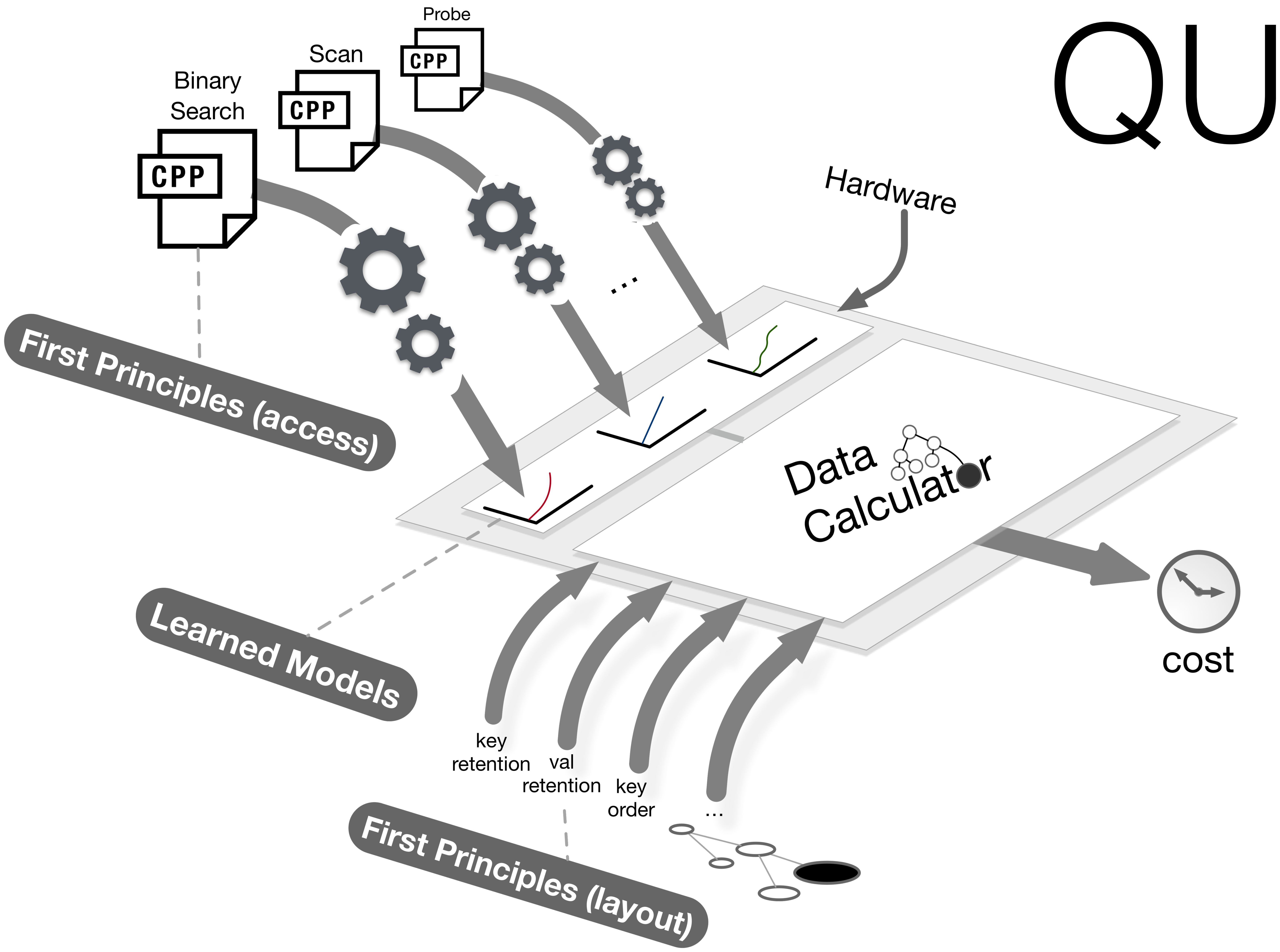

Probe
Scan
Binary Search
CPP
CPP
CPP
...
Hardware
First Principles (access)
Data Calculator
cost

TRAINING



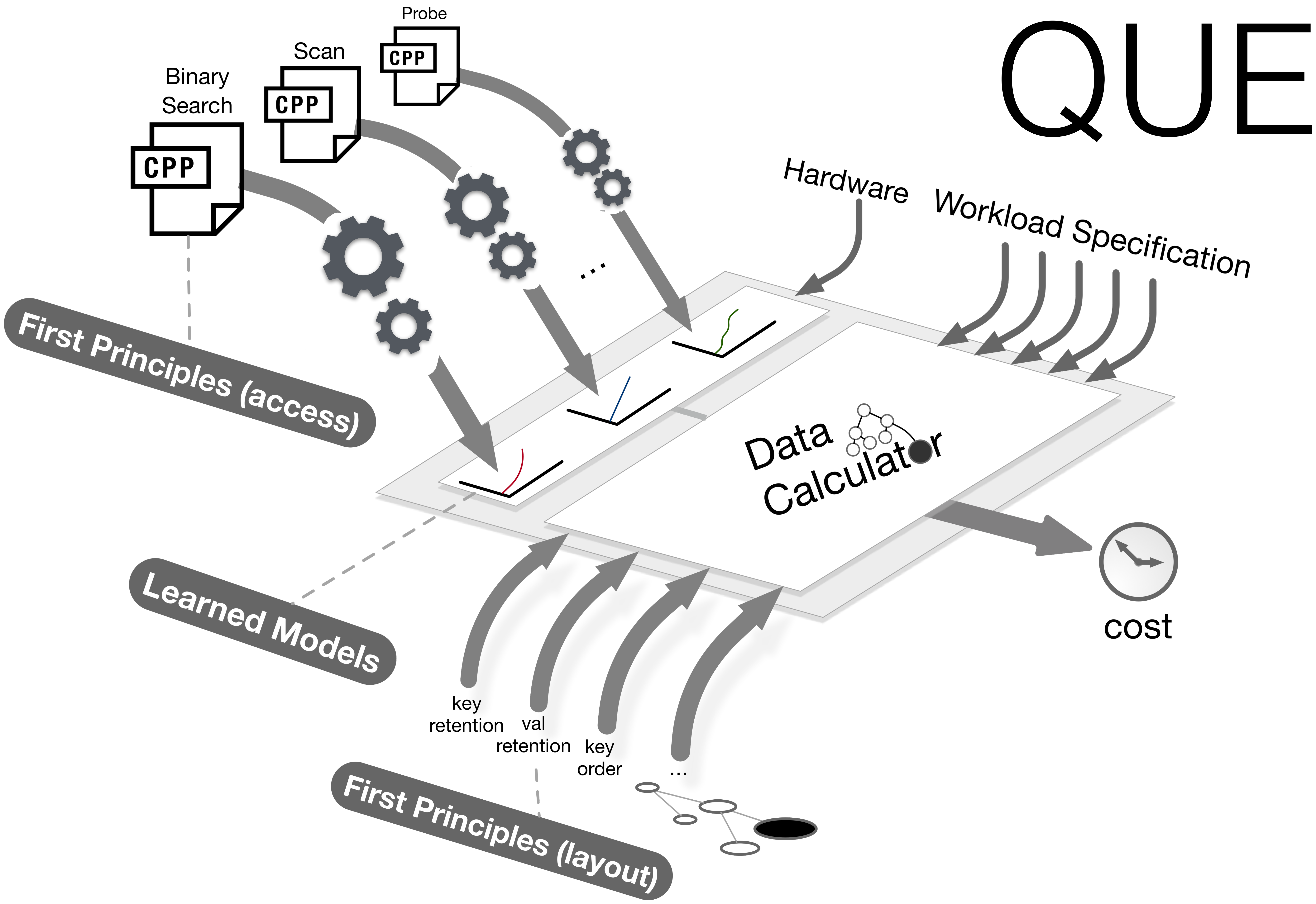
DASlab @ Harvard SEAS

QUEYRYING



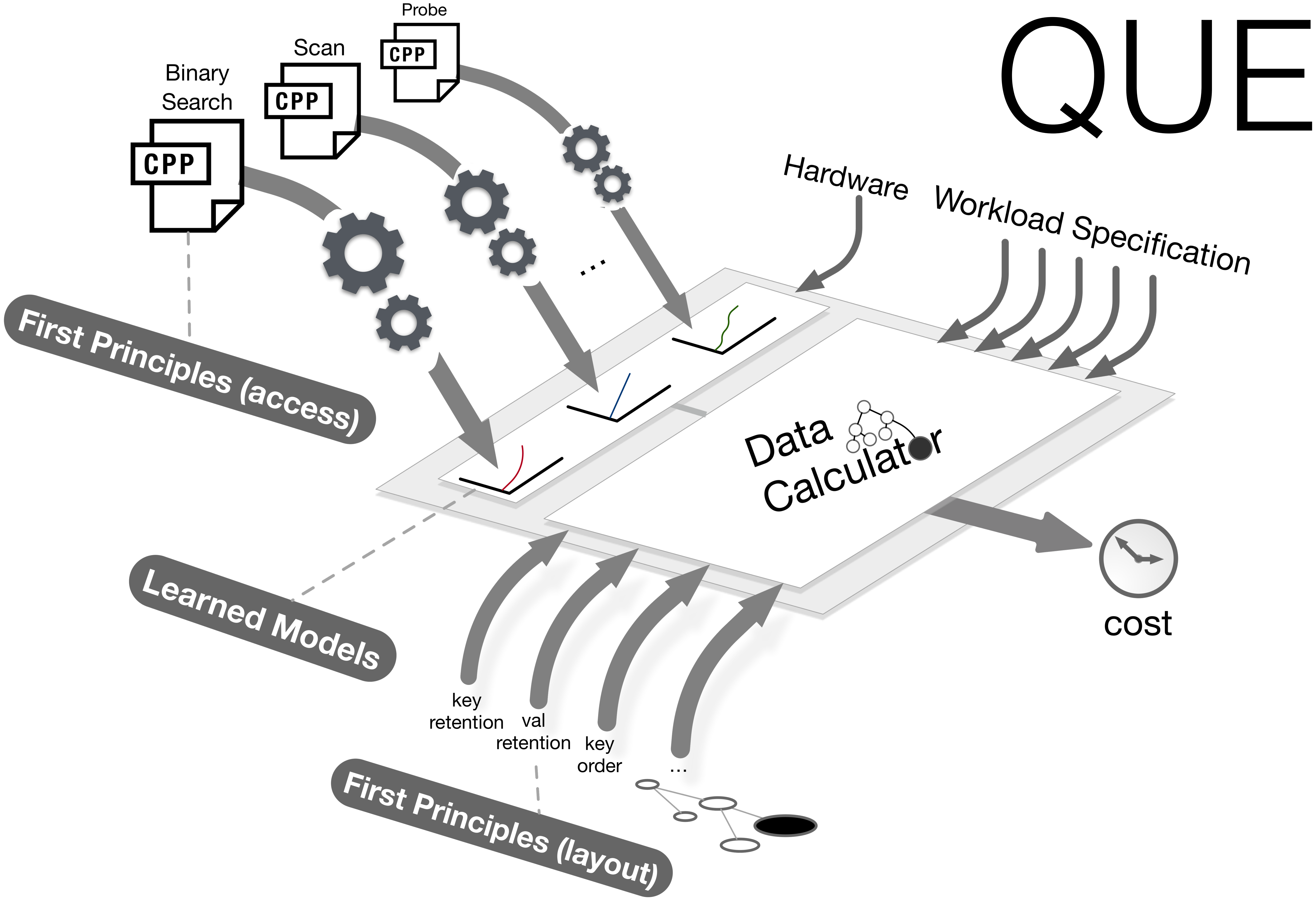
DASlab @ Harvard SEAS

QUEYRYING

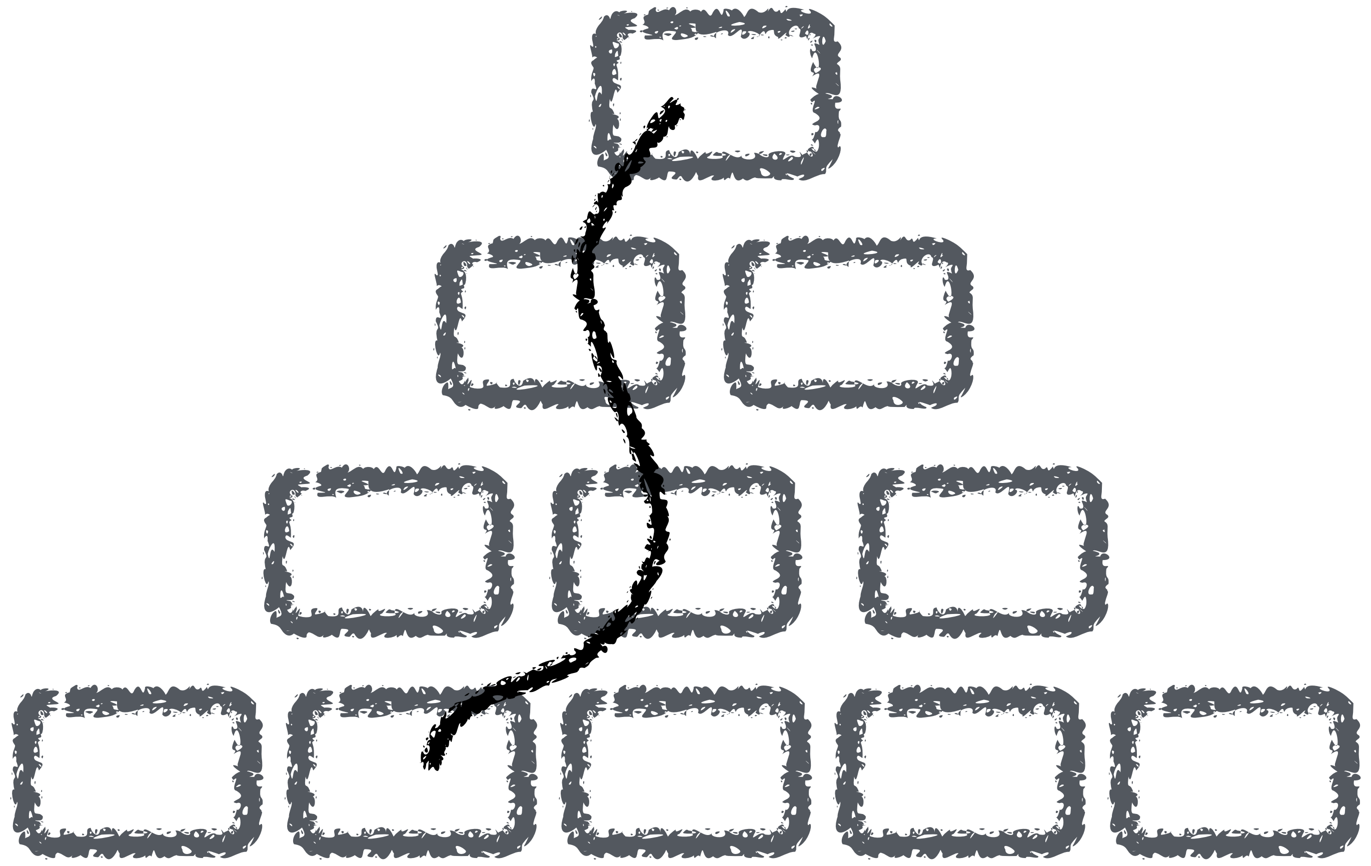


DASlab @ Harvard SEAS

QUEYRYING



FOR EACH OPERATION



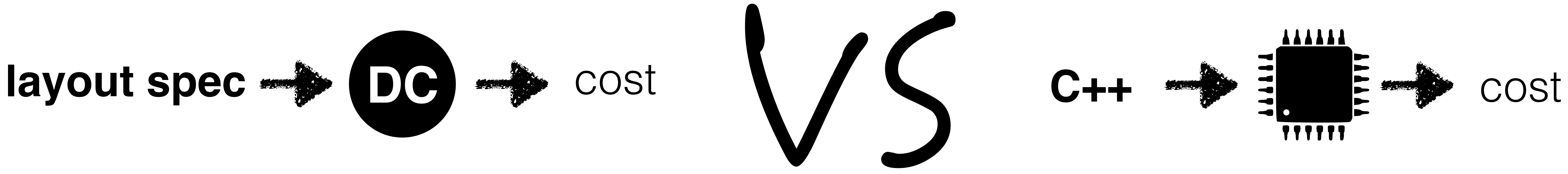
QUEYRYING

Probe
CPP
Scan
CPP
Binary Search
CPP
...
First Principles (access)
Hardware
Workload Specification
Data Calculator
Learned Models
key retention
val retention
key order
...
First Principles (layout)
cost
FOR EACH OPERATION

FOR EACH NODE

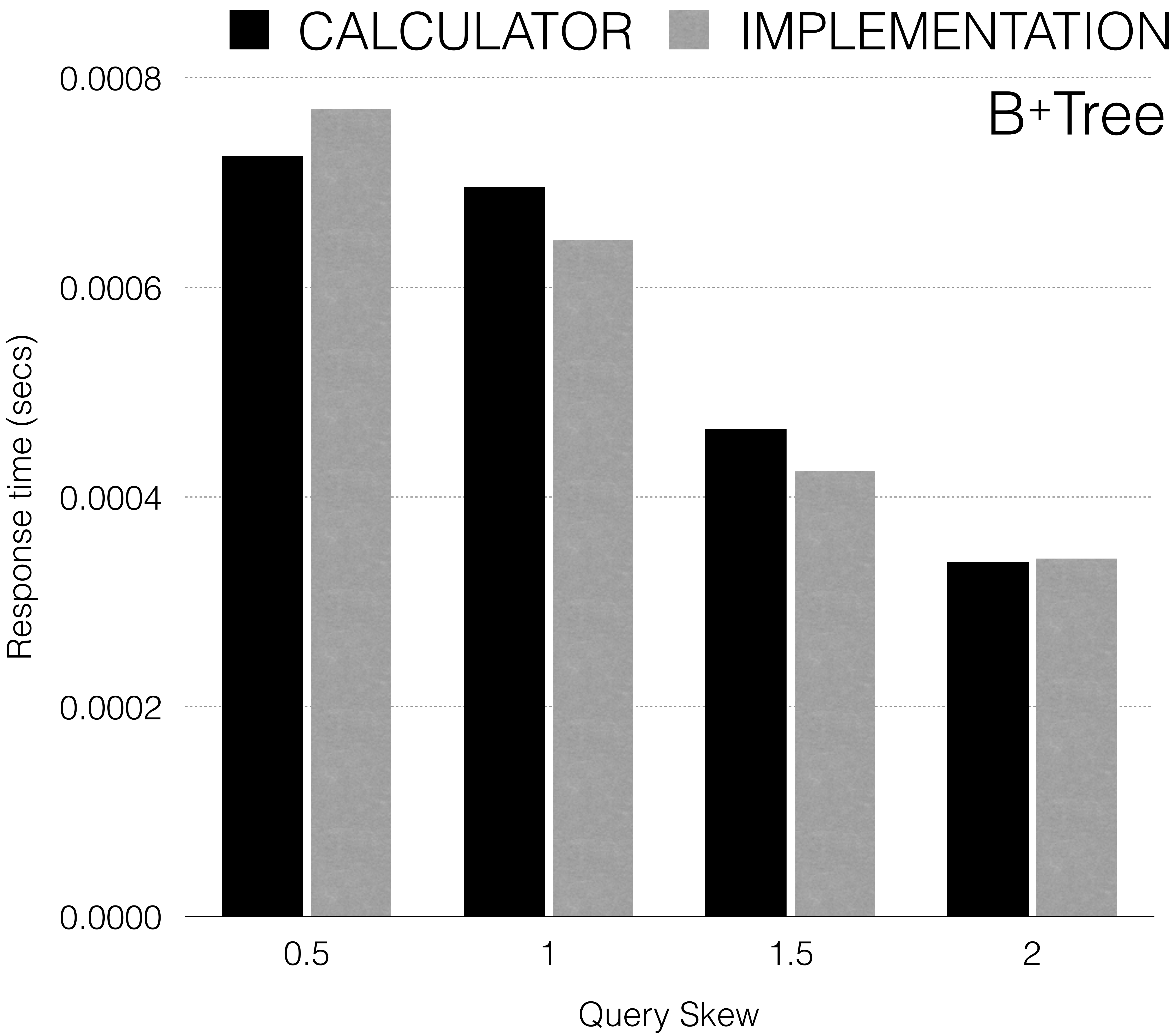
1. Decide access **strategy** (L1) based on node design
2. Decide exact access strategy **implementation** (L2) based on available models
3. **Get cost** for chosen model

CAN WE COMPUTE PERFORMANCE ACCURATELY?



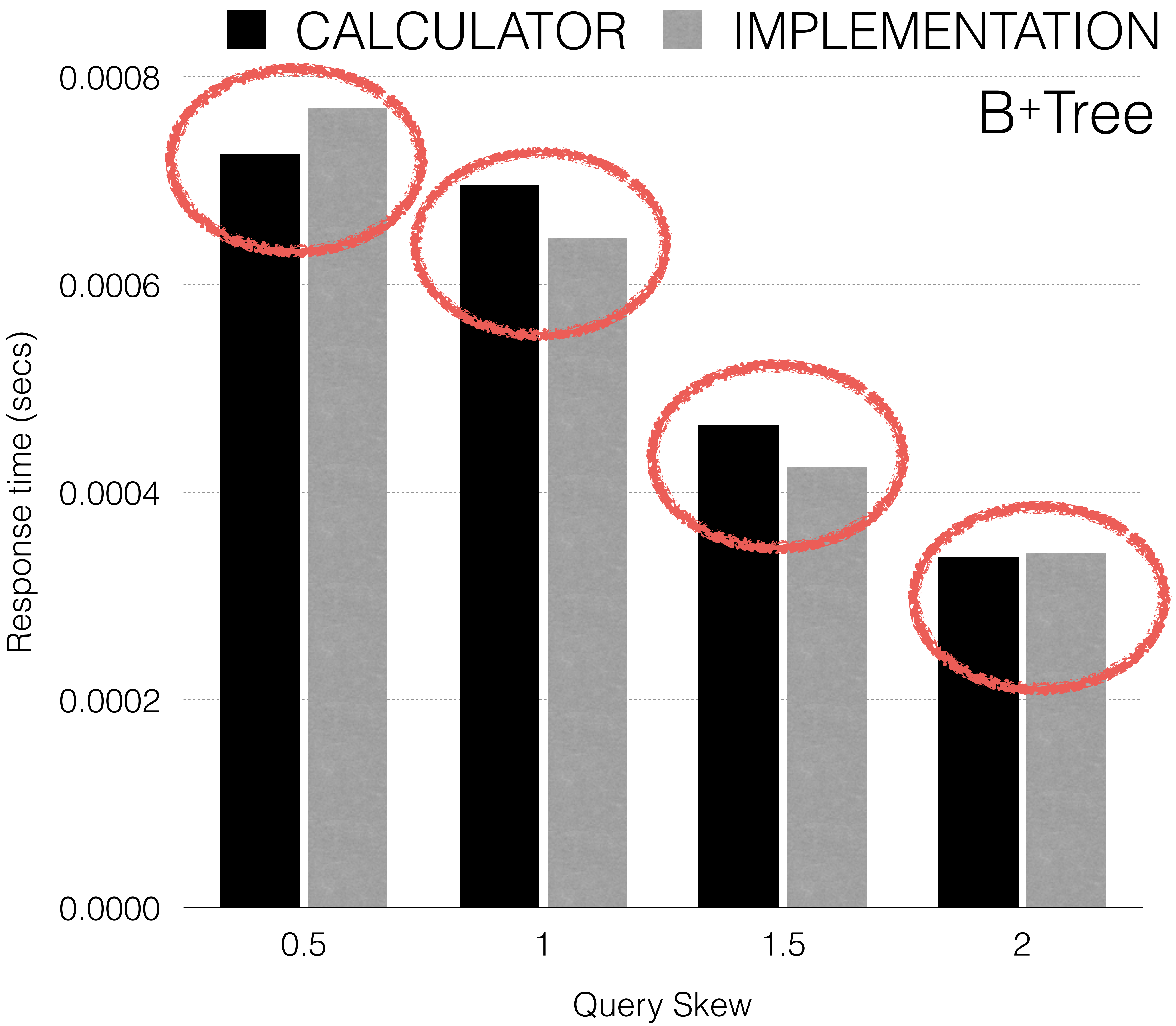
layout spec → DC → cost VS C++ → → cost

(same workload, hardware, data)

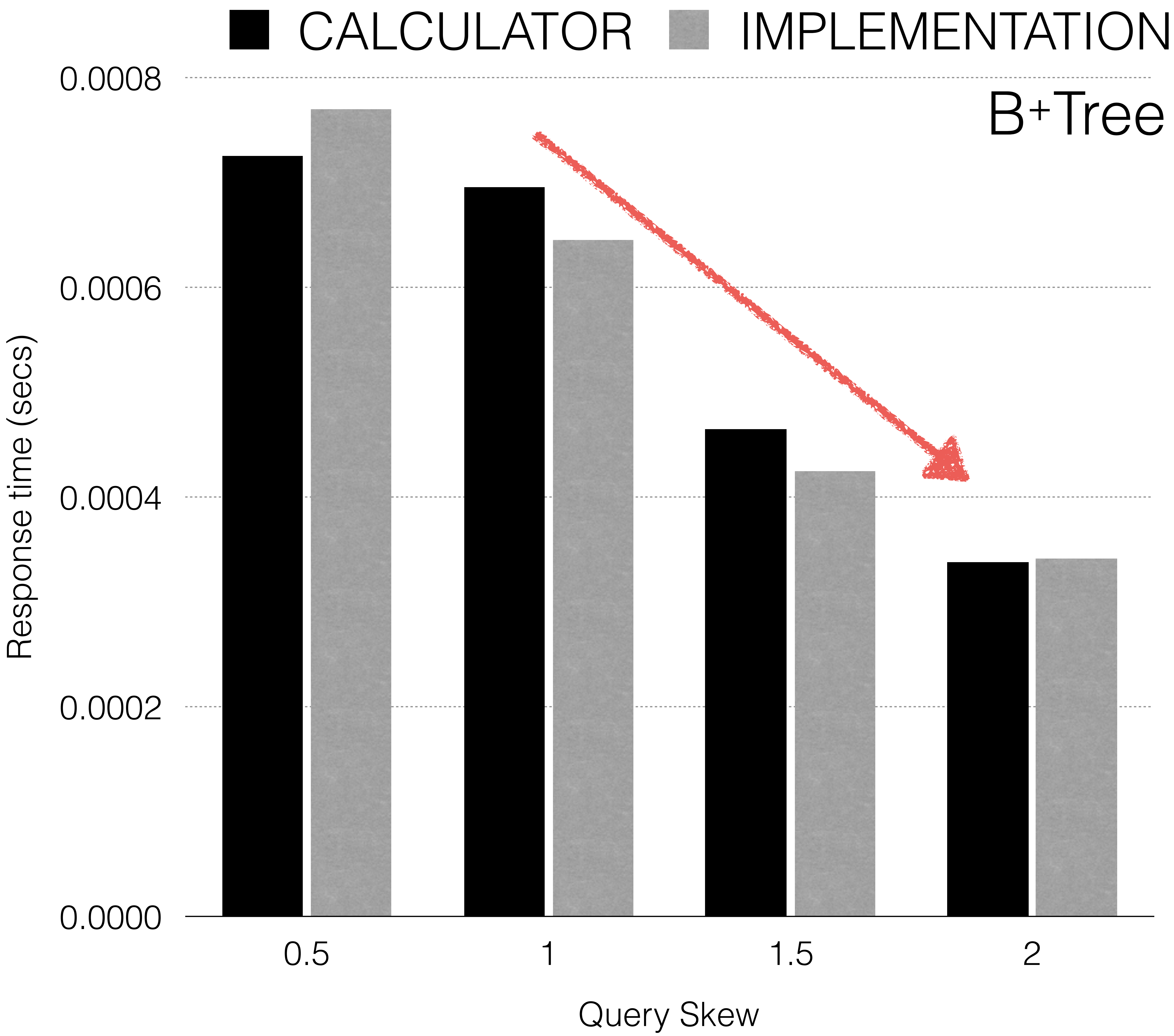


DASlab @ Harvard SEAS

{10M (uniform) k-v pairs, 100 point queries (skewed)}

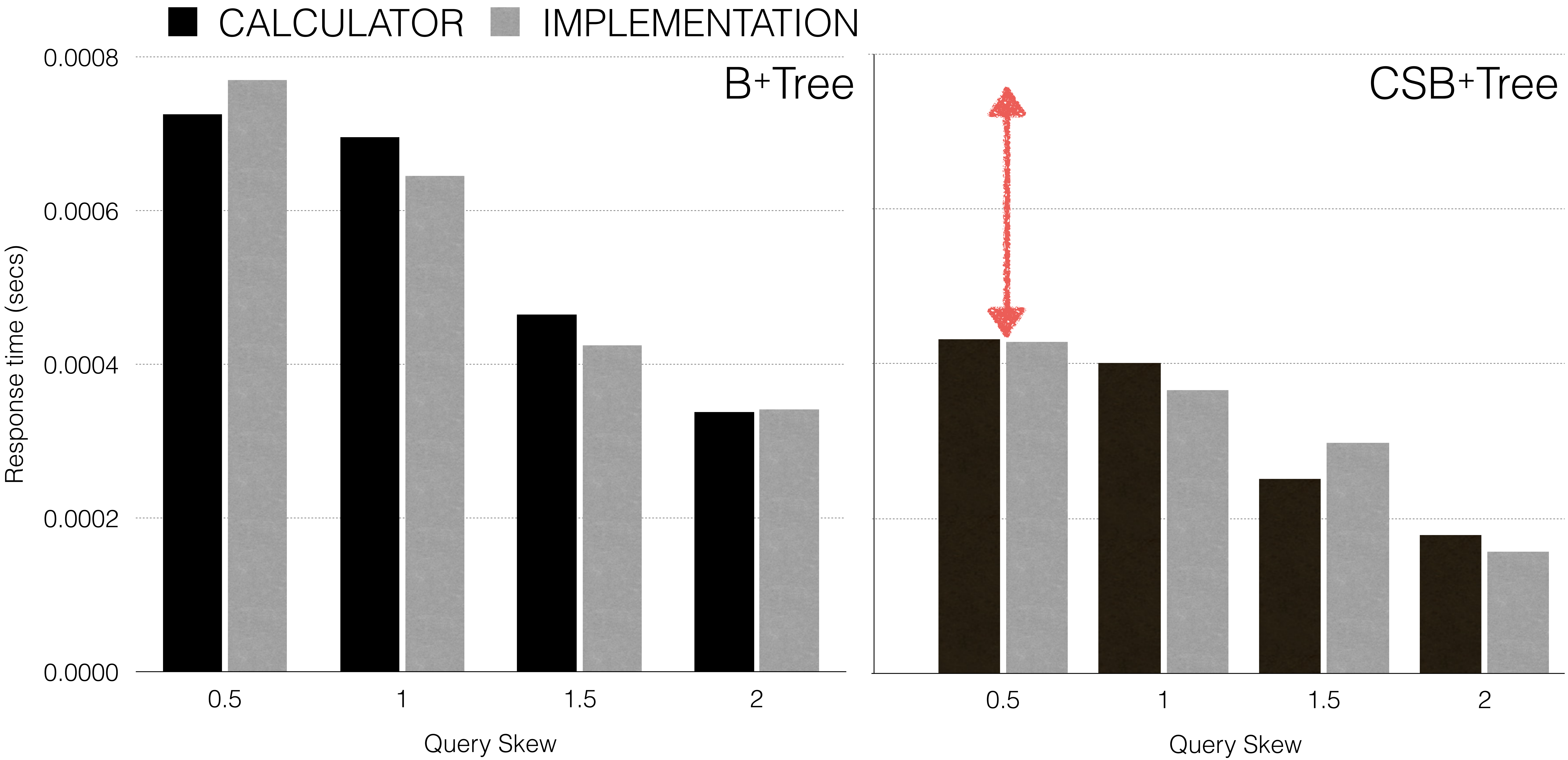


{10M (uniform) k-v pairs, 100 point queries (skewed)}



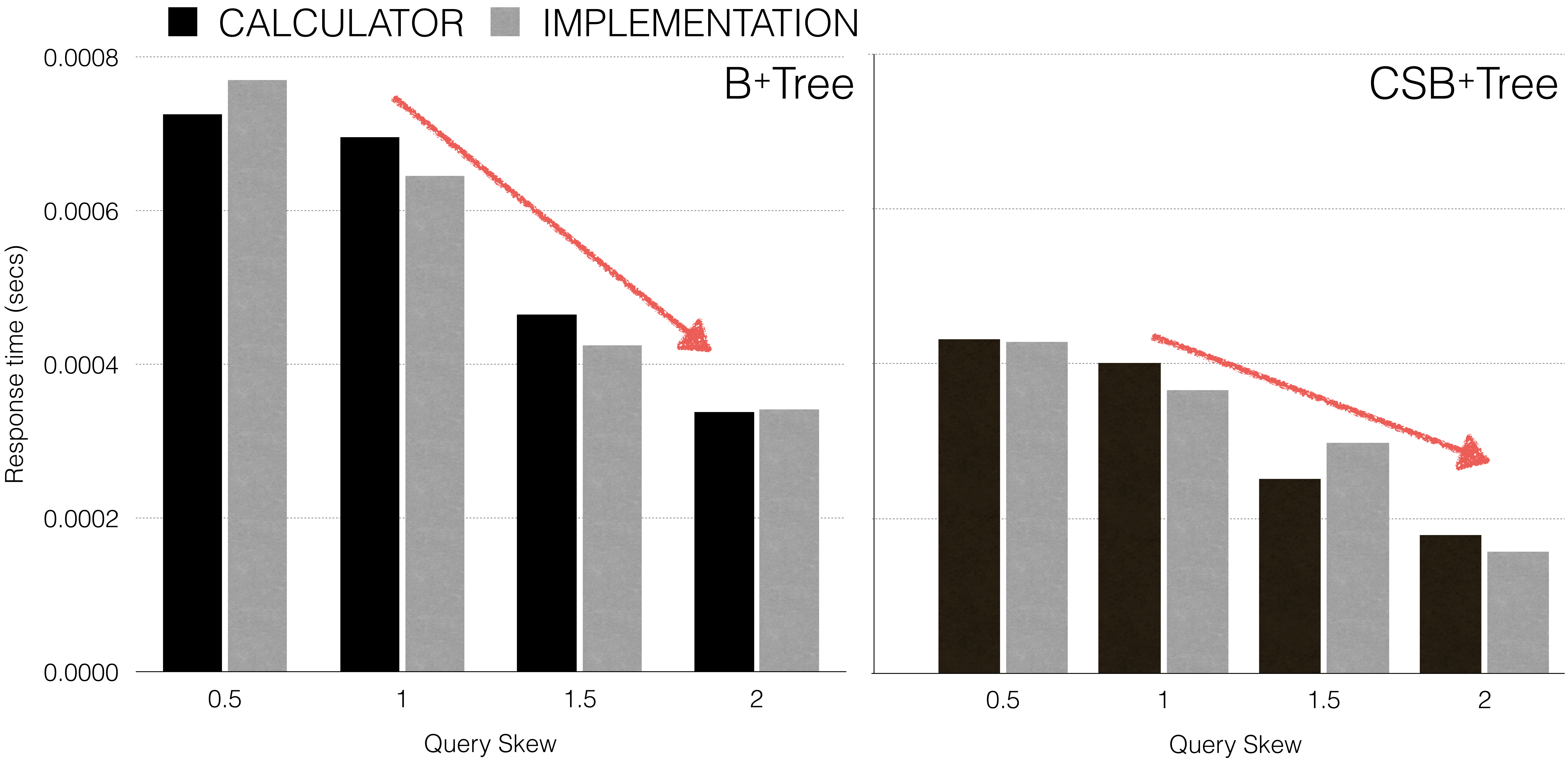
{10M (uniform) k-v pairs, 100 point queries (skewed)}

DASlab @ Harvard SEAS



{10M (uniform) k-v pairs, 100 point queries (skewed)}

DASlab @ Harvard SEAS



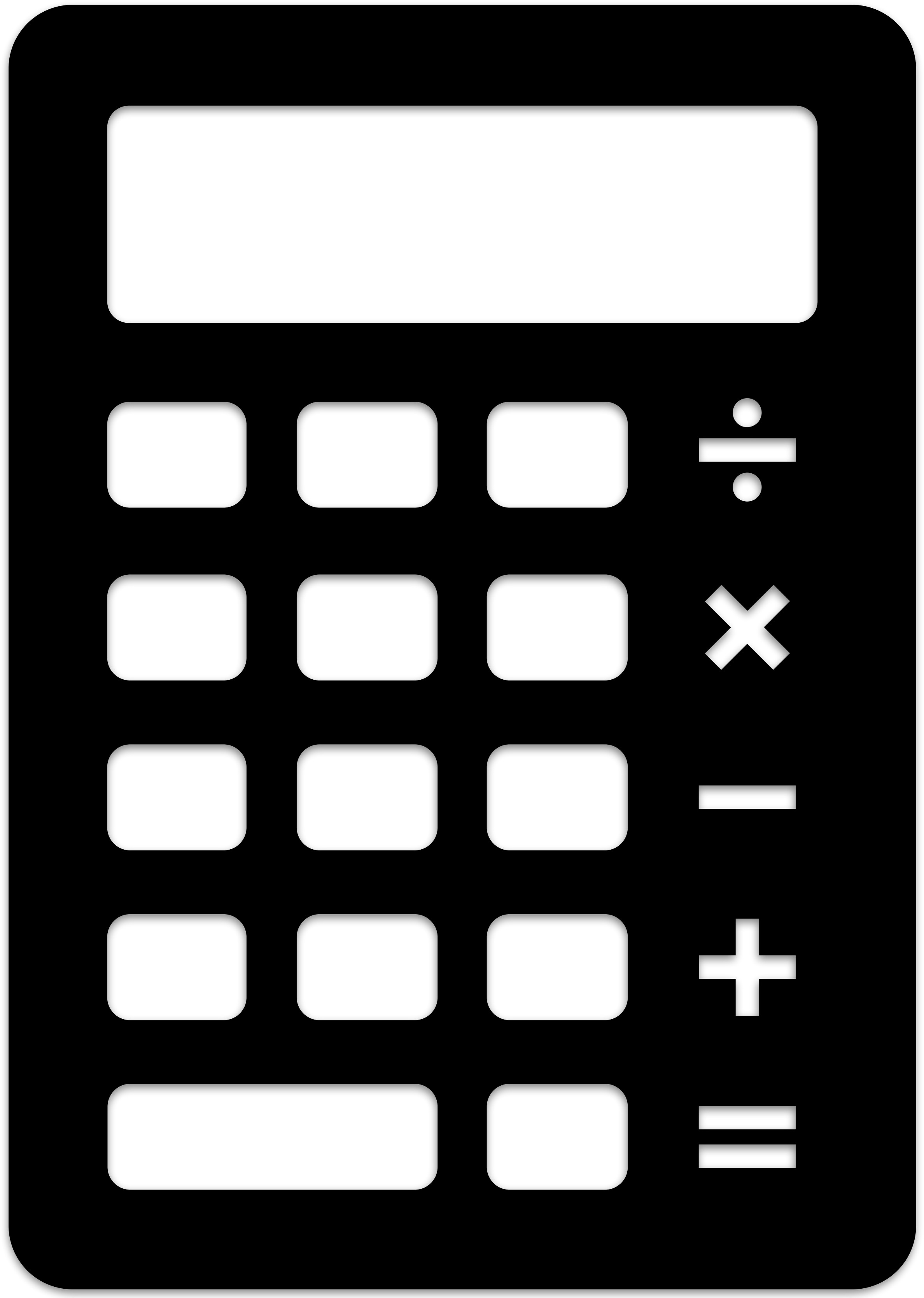
DASlab @ Harvard SEAS

{10M (uniform) k-v pairs, 100 point queries (skewed)}

DESIGN SPACE

COST SYNTHESIS

WHO AND
HOW TO USE

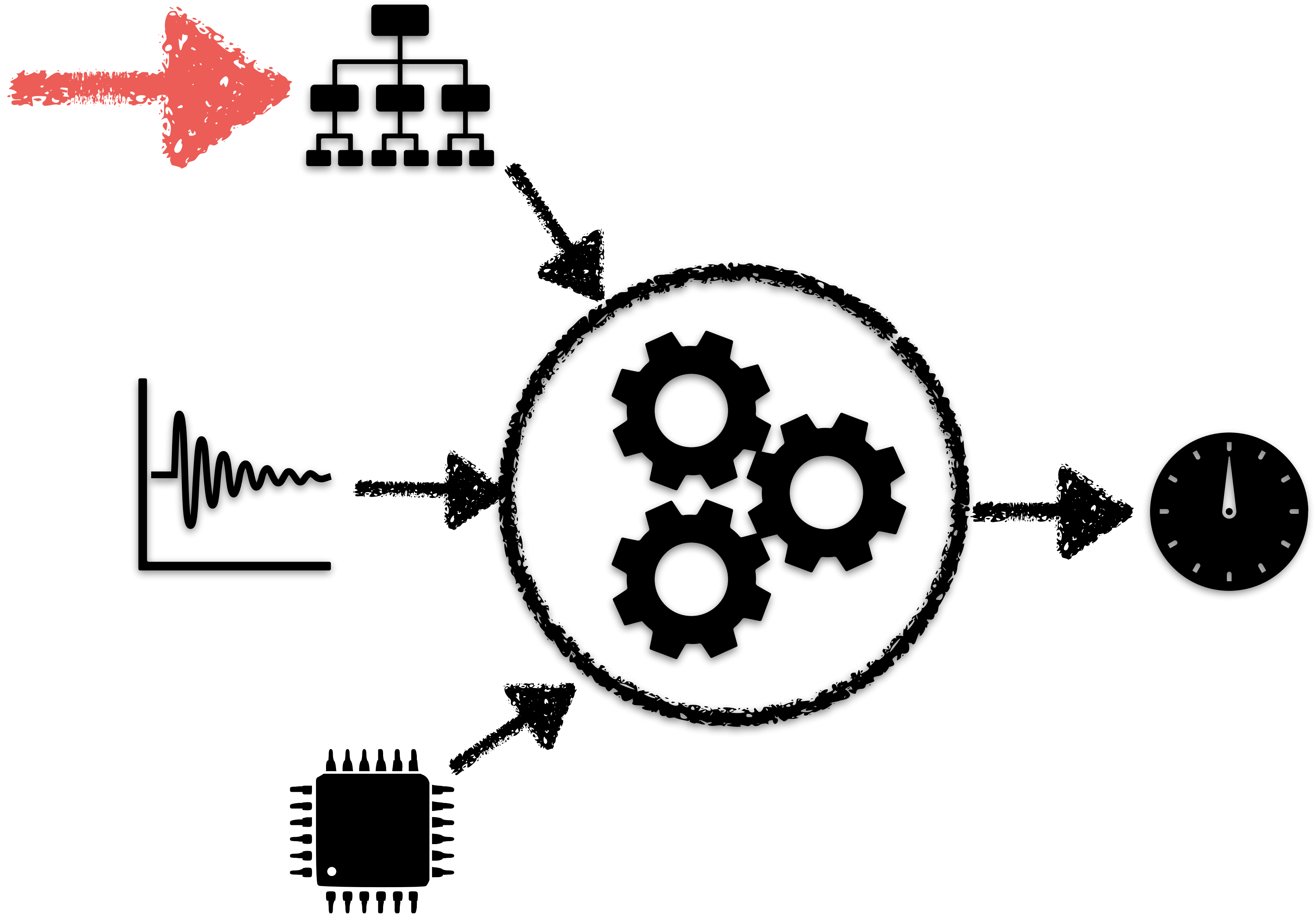


WHO AND HOW TO USE

what-if.design

“IMAGINE’’ DESIGNS

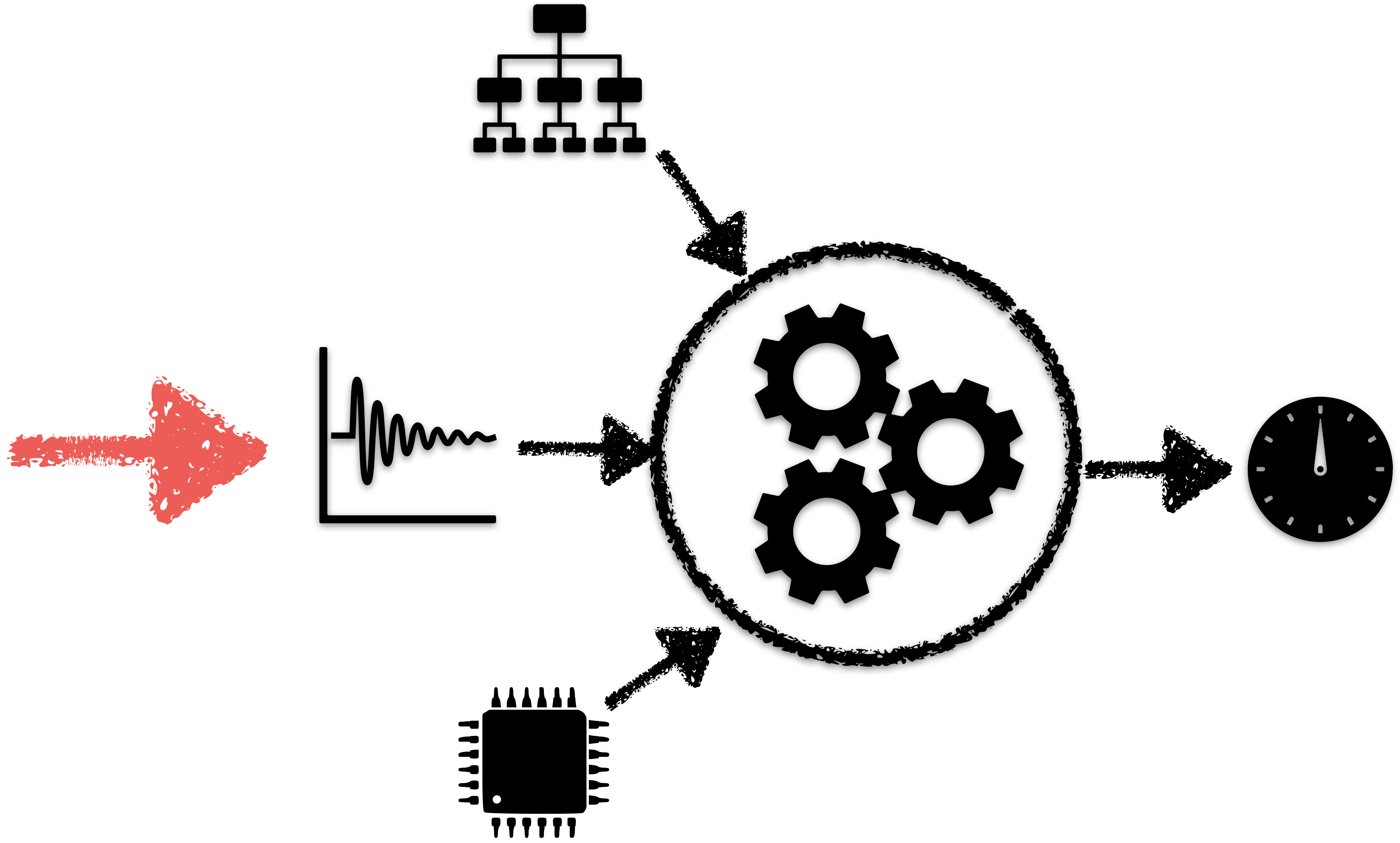
*What-if we **add bloom filters** in the hash-table buckets?*



~20 SECONDS

(workload:10 Million entries, 100 queries)

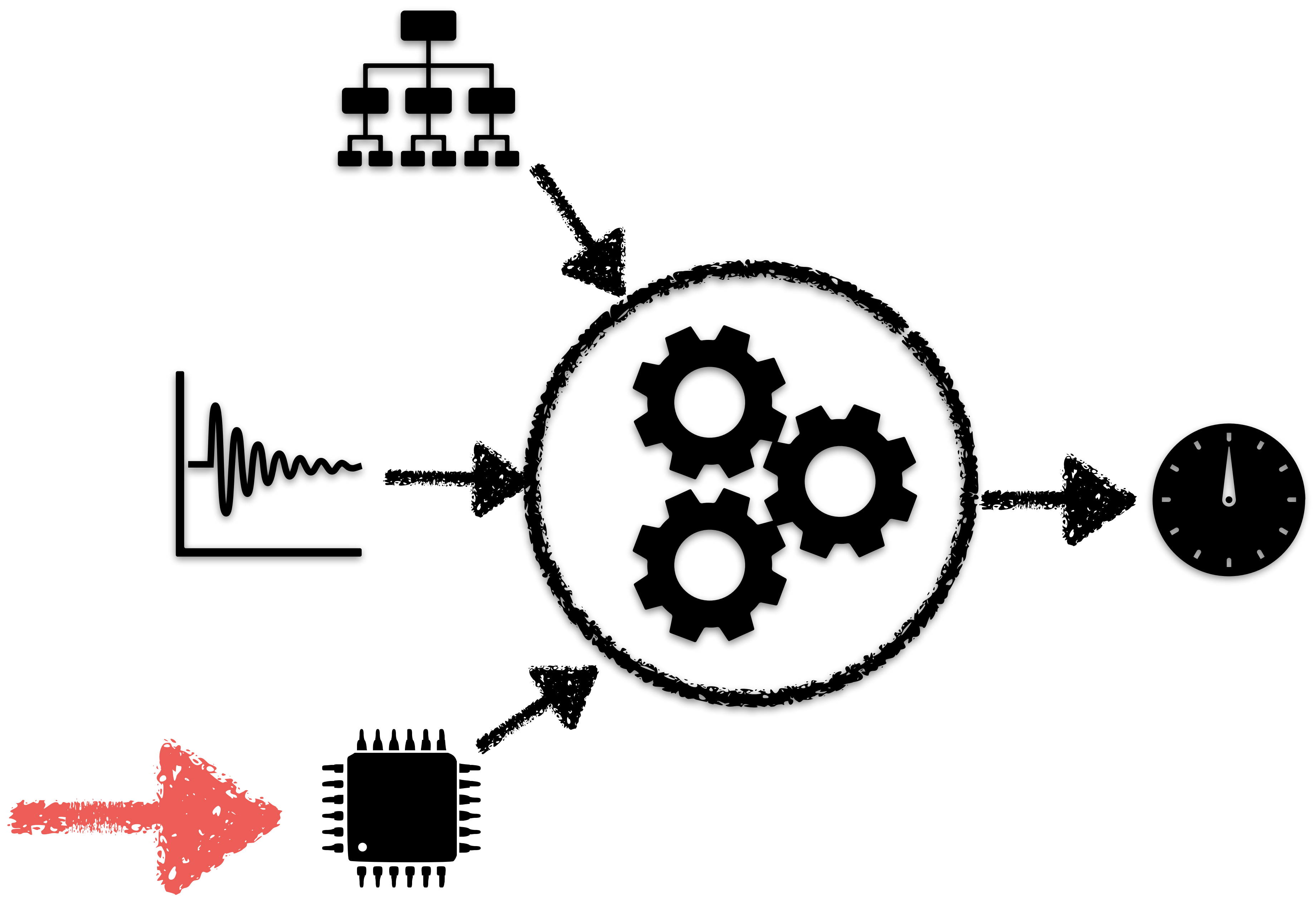
*What-if the workload changes to **90% writes**?*



~20 SECONDS

(workload:10 Million entries, 100 queries)

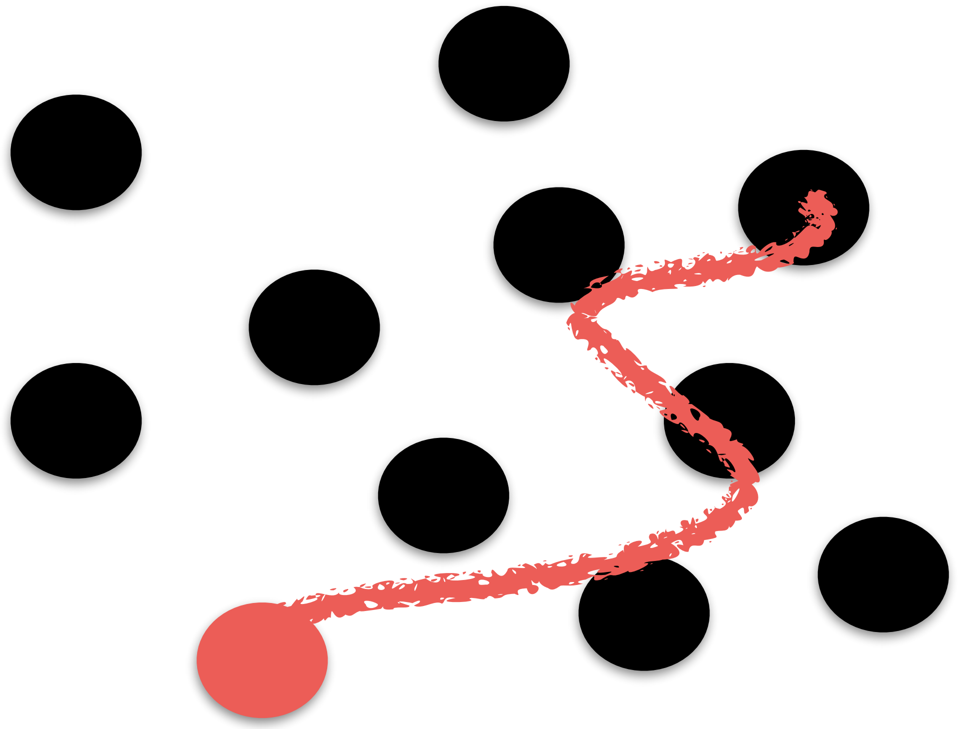
What-if we ***buy faster CPU X****?*



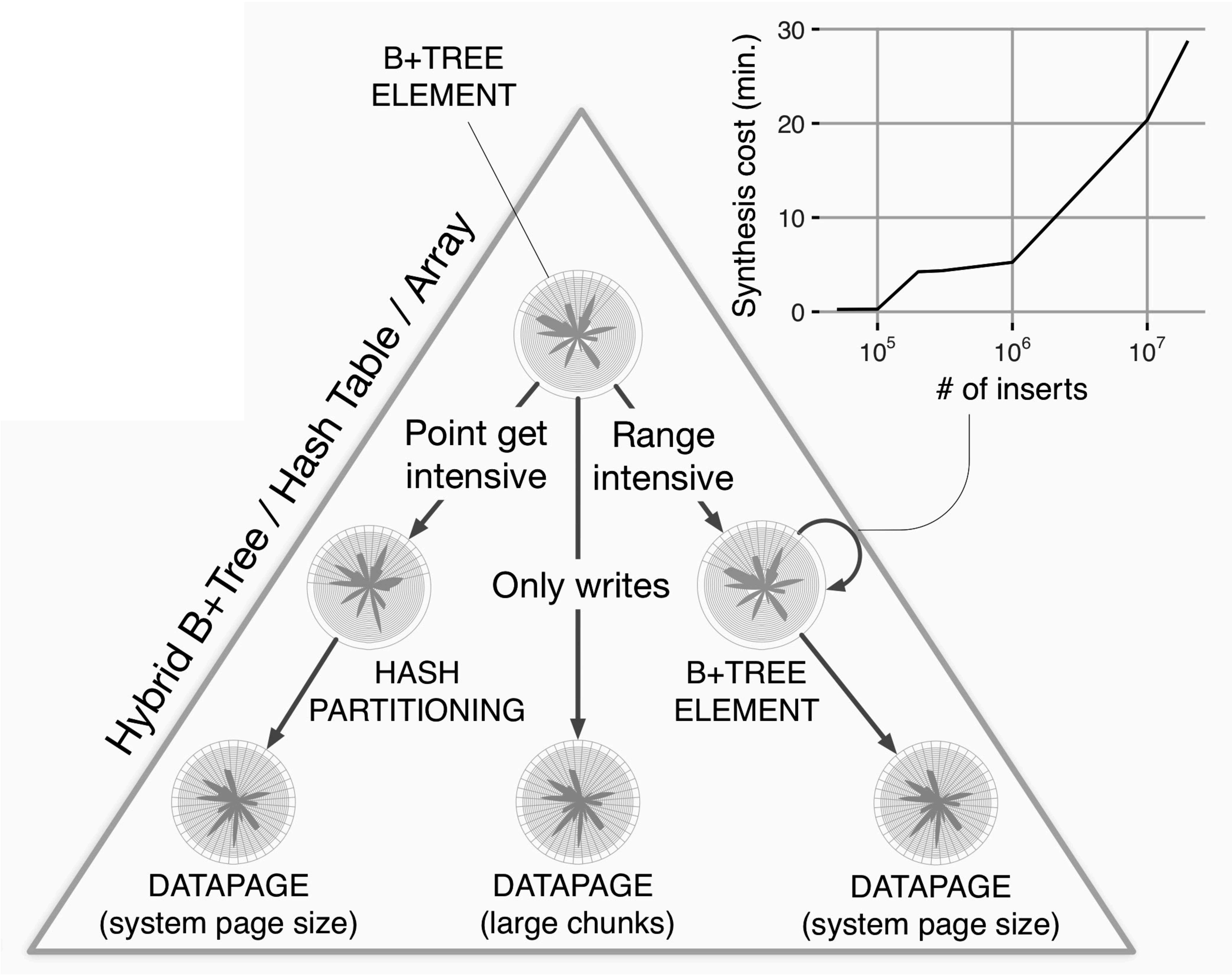
~20 SECONDS

(workload:10 Million entries, 100 queries)

auto.design



dynamic programming
genetic algorithms
reinforcement learning



Hybrid B+Tree / Hash Table / Array

B+TREE ELEMENT

Point get intensive

Range intensive

Only writes

HASH PARTITIONING

B+TREE ELEMENT

DATAPAGE (system page size)

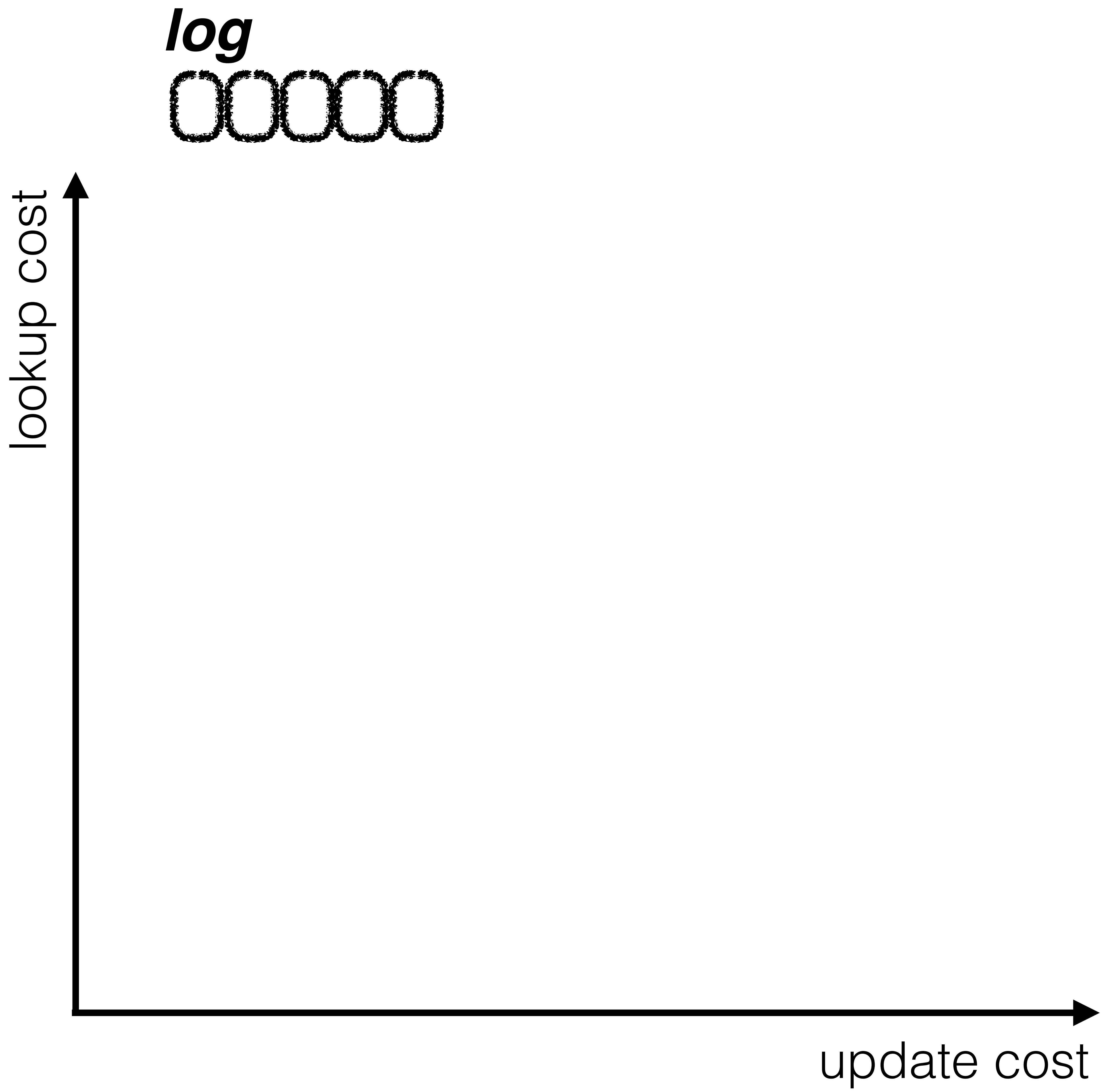
DATAPAGE (large chunks)

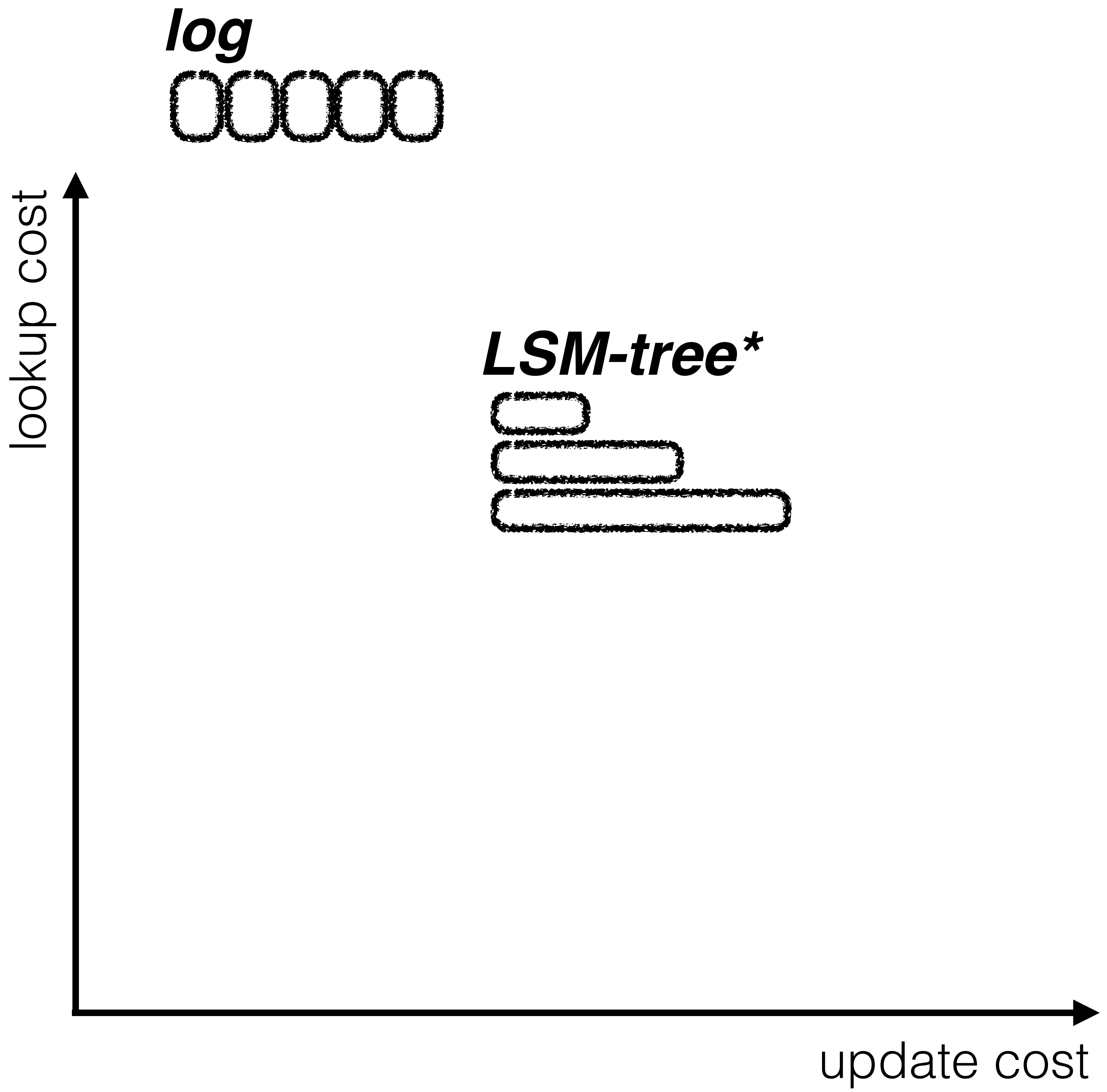
DATAPAGE (system page size)

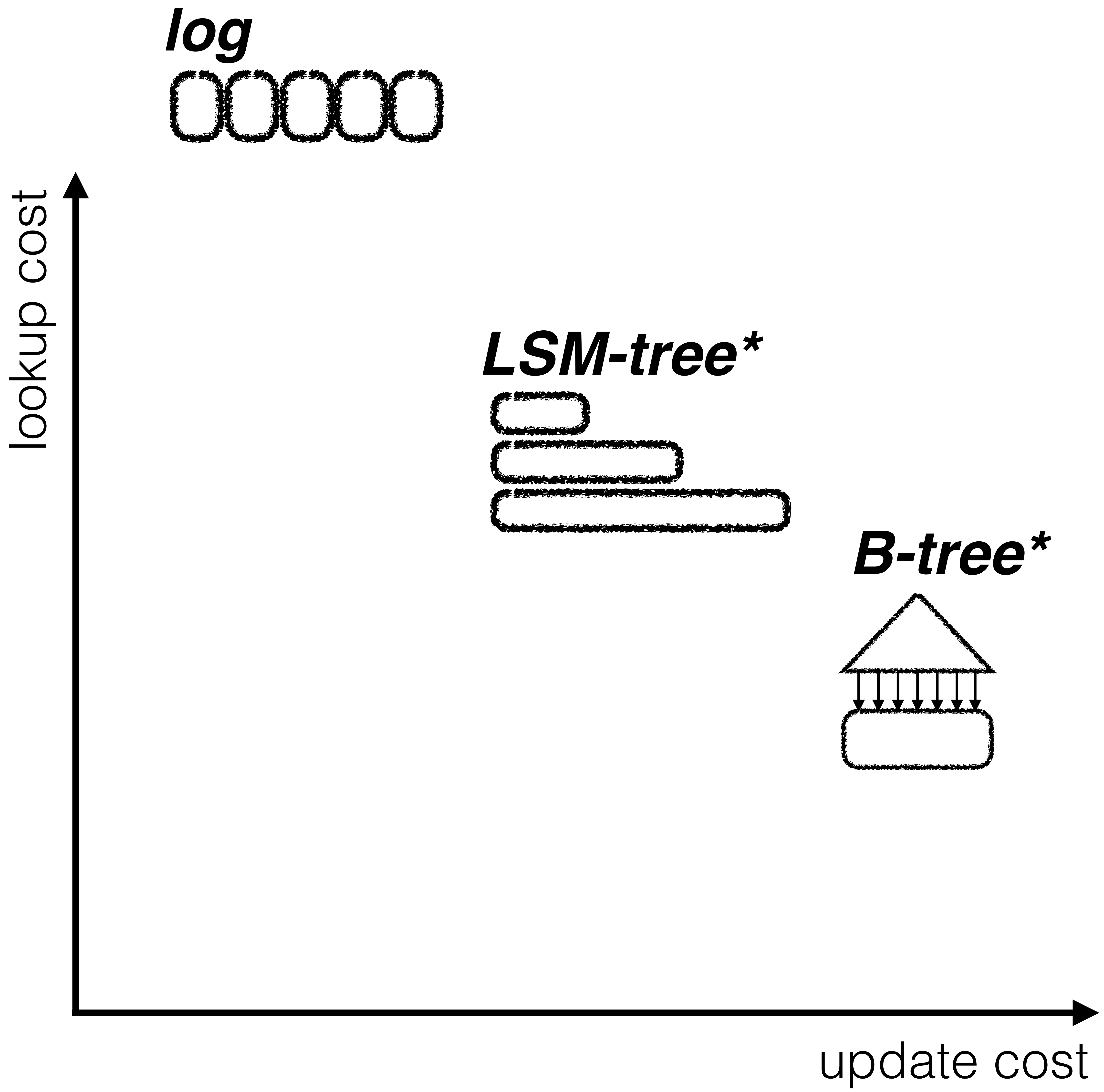
design.continuums

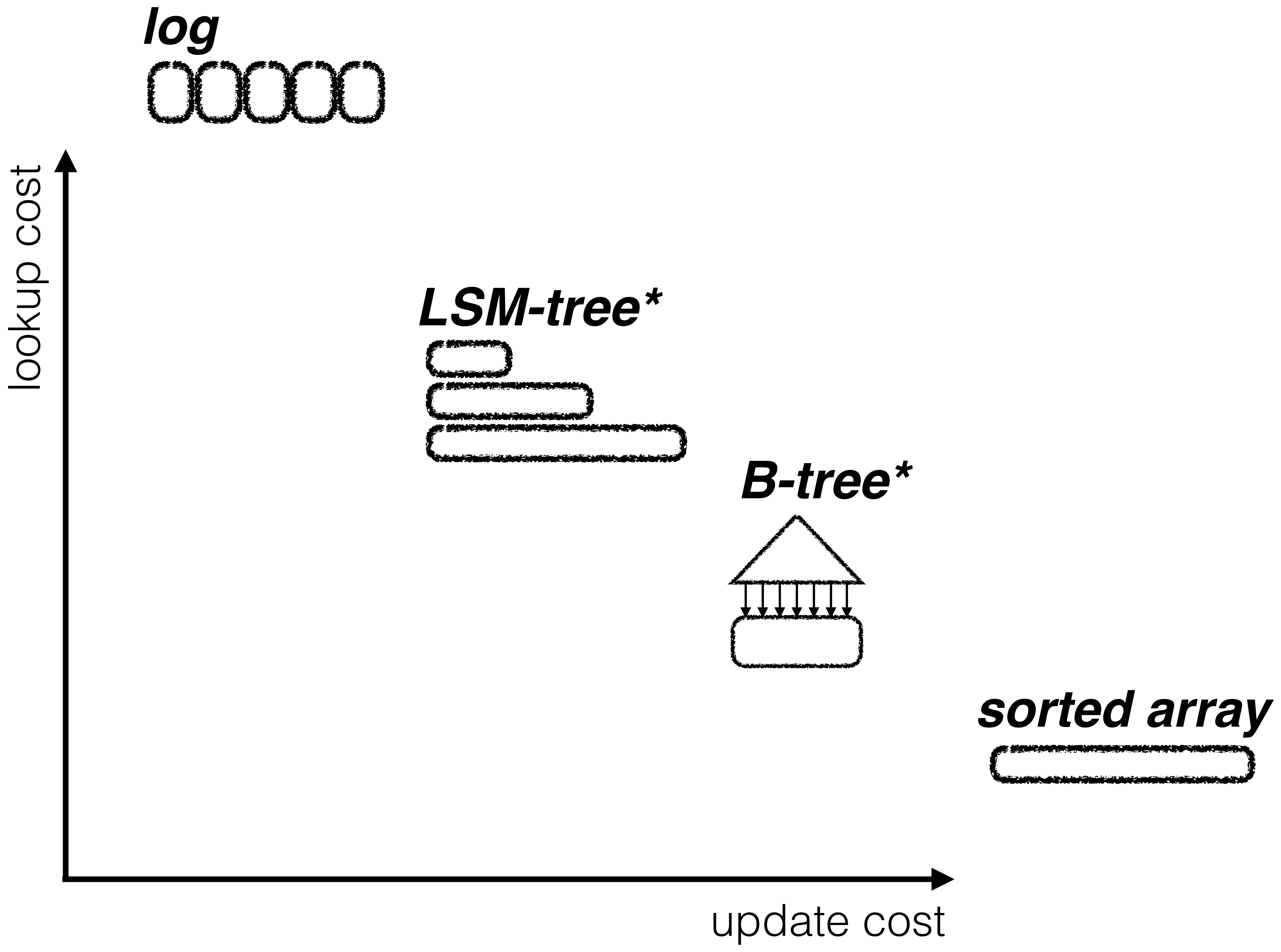
UNIFY DESIGNS FOR FAST NAVIGATION

lookup cost
update cost

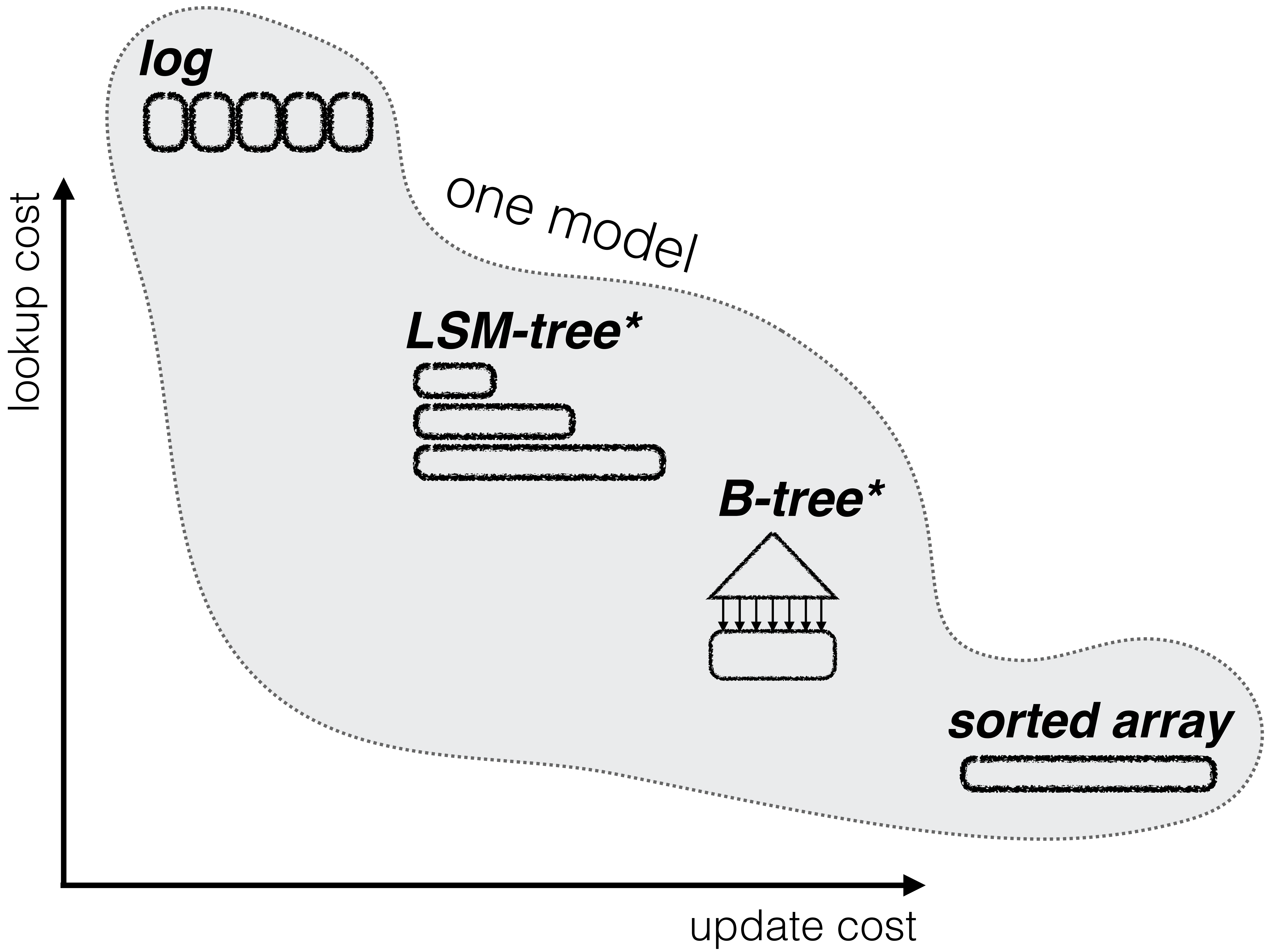

log
lookup cost
update cost

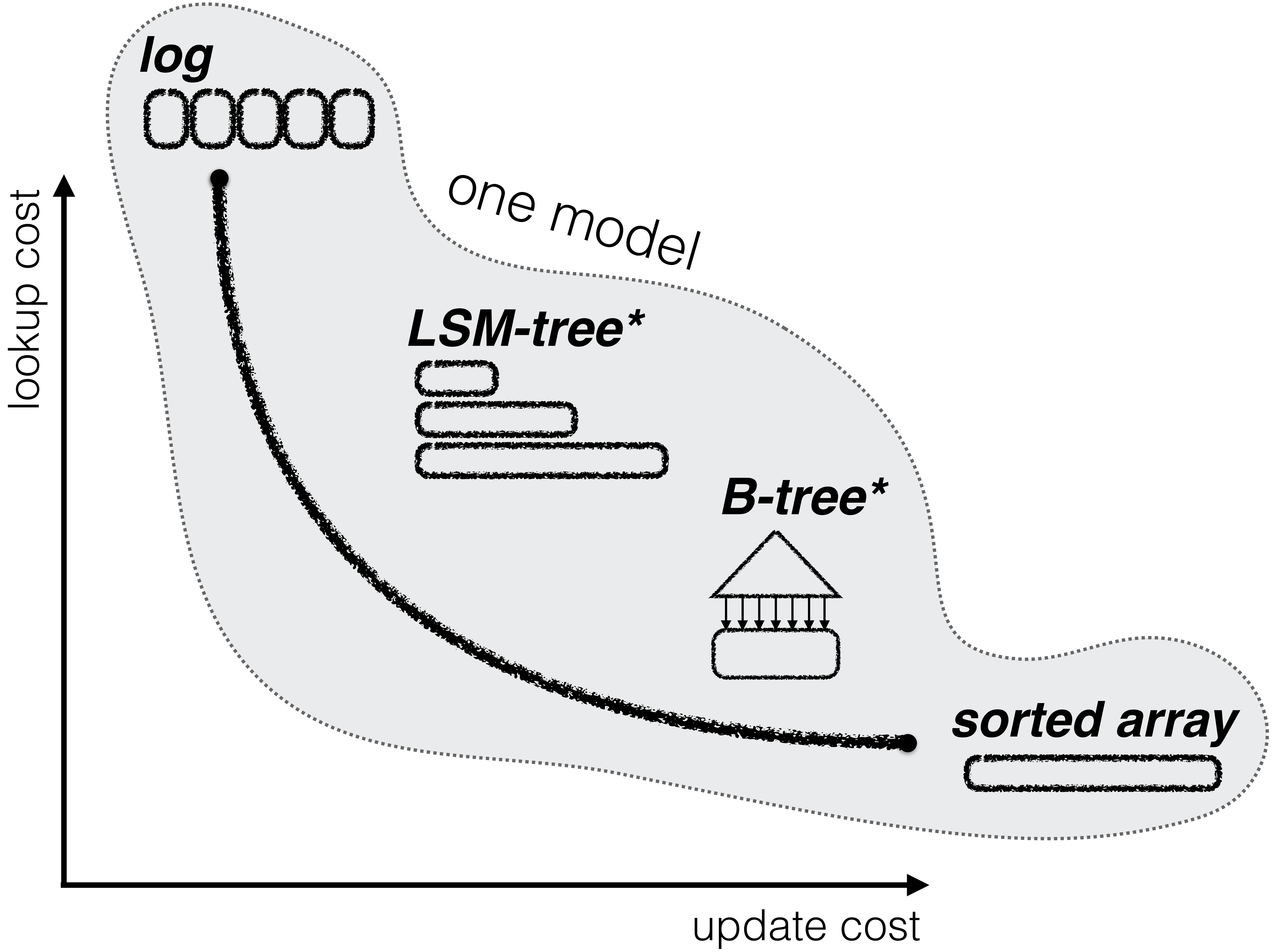

log
LSM-tree*
lookup cost
update cost


log
LSM-tree*
B-tree*
lookup cost
update cost

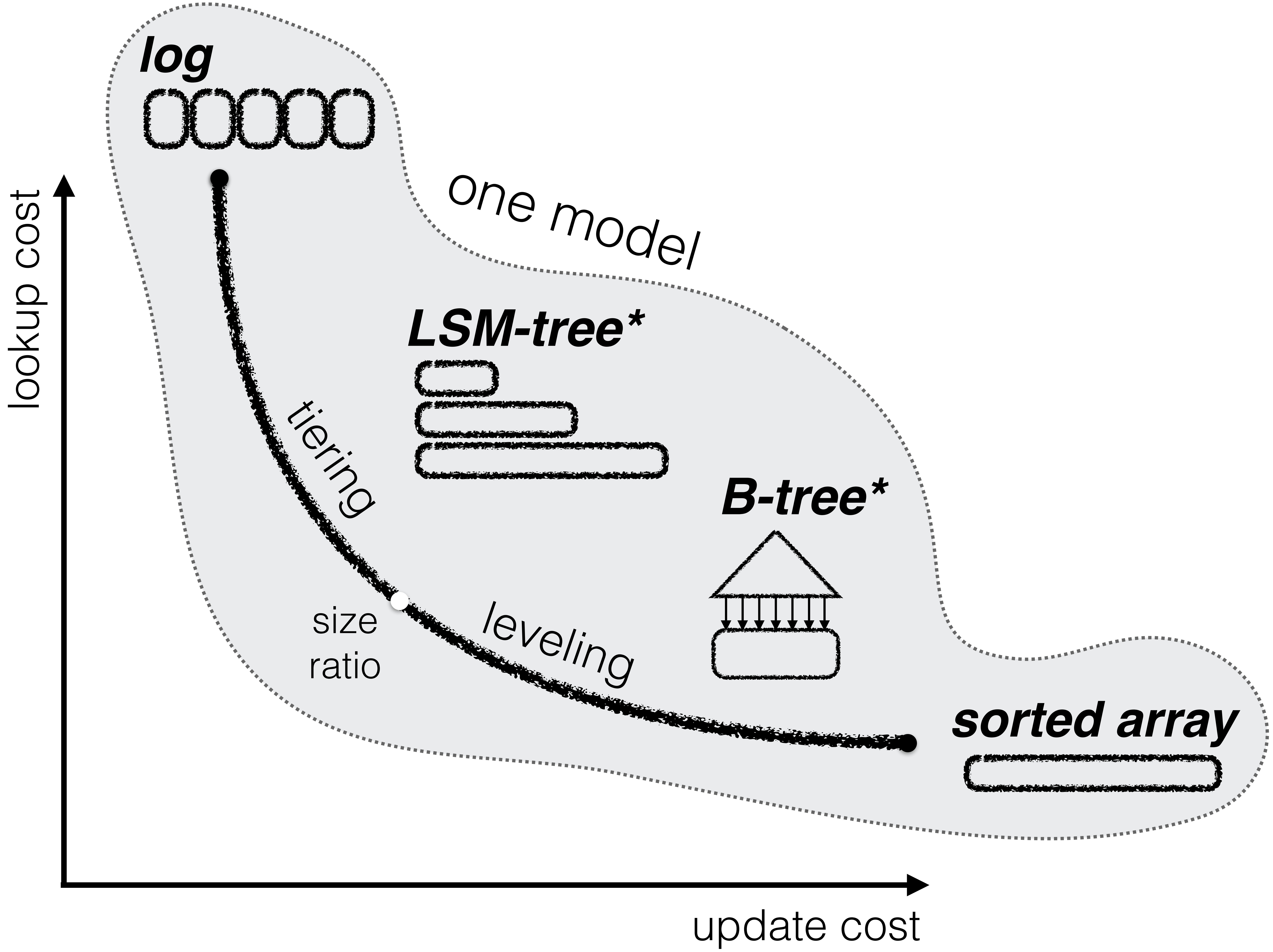

log
LSM-tree*
B-tree*
sorted array
lookup cost
update cost

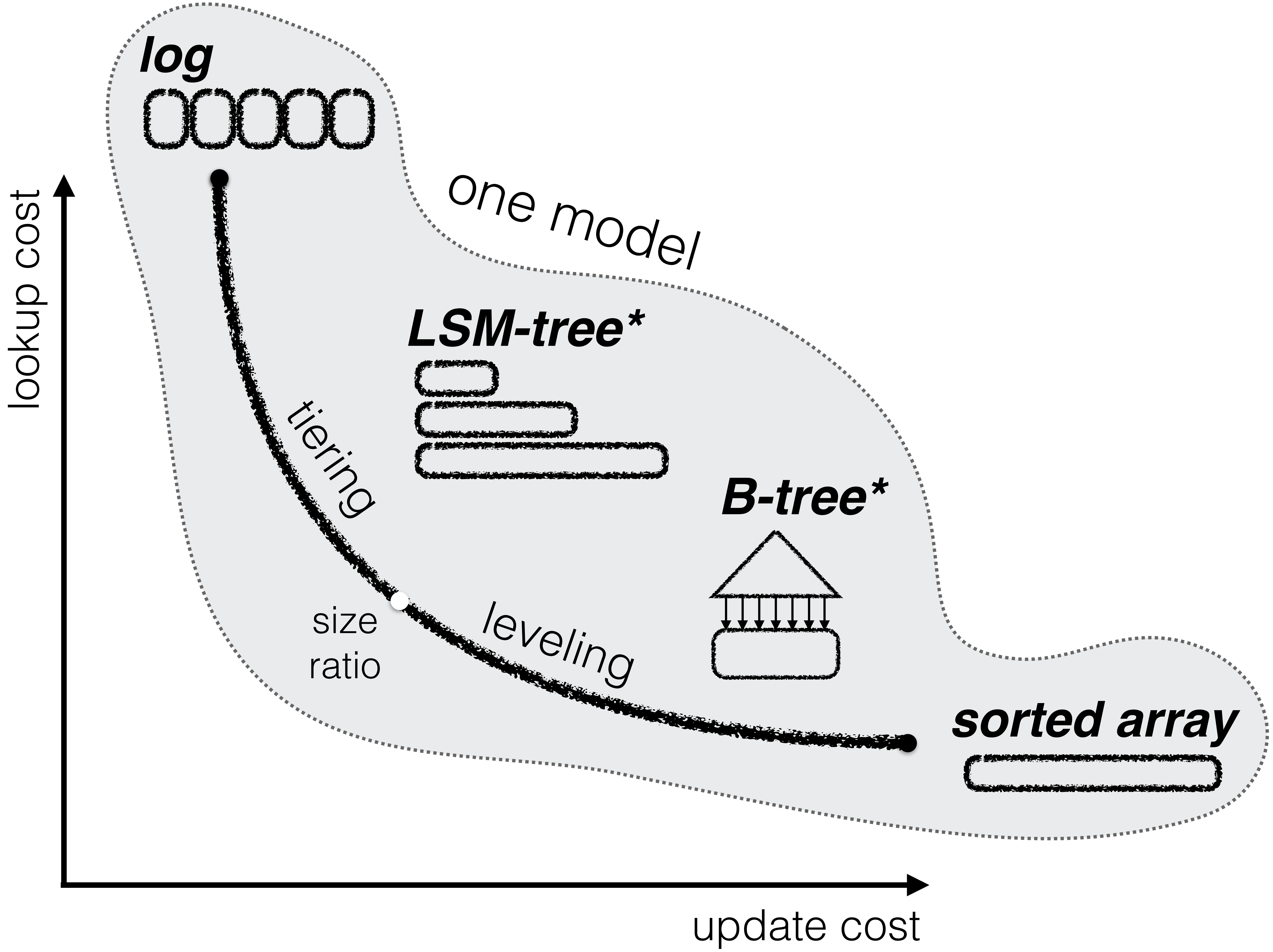
@SIGMOD17/18


log
one model
LSM-tree*
B-tree*
sorted array
lookup cost
update cost

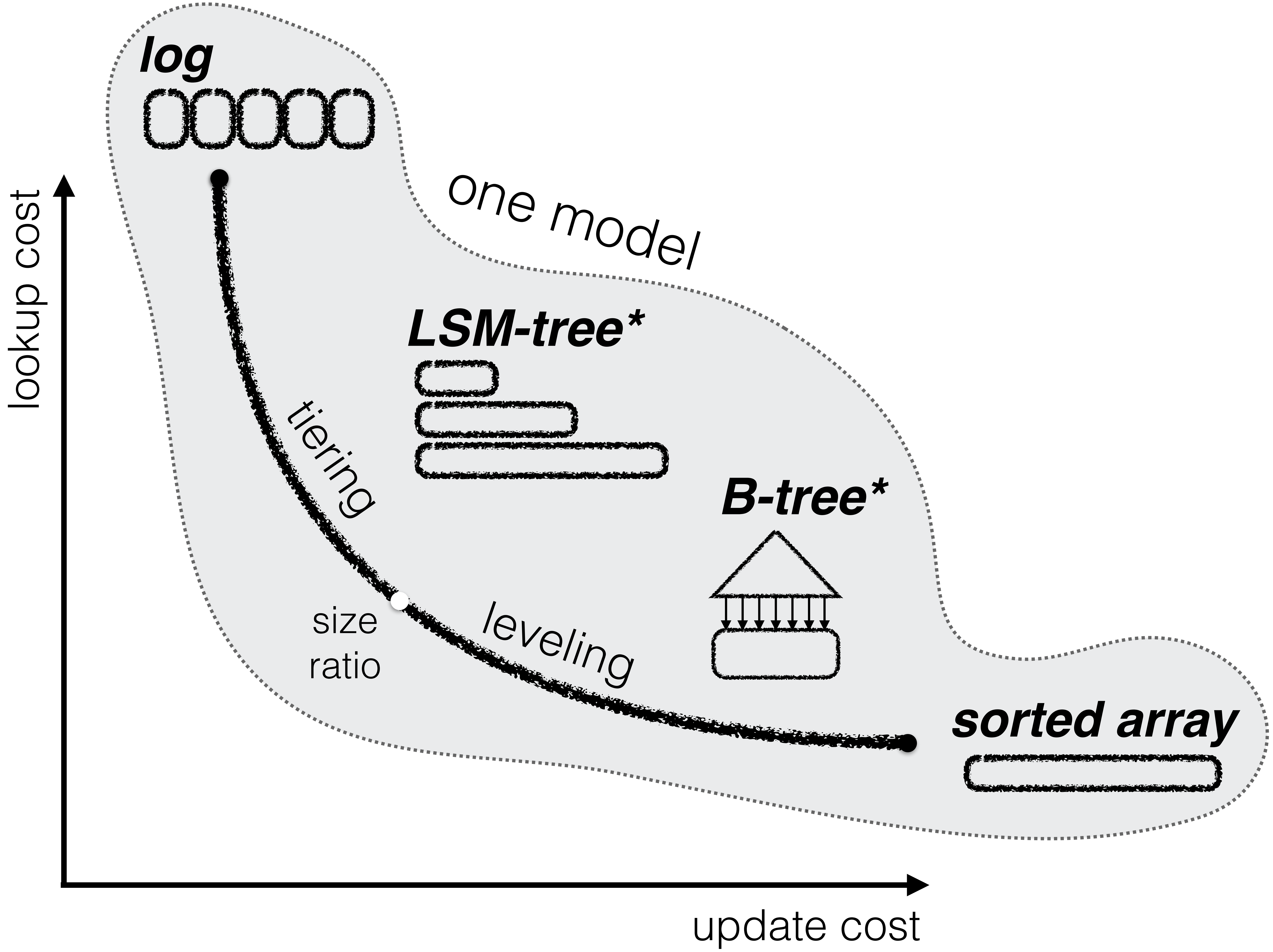

log
one model
LSM-tree*
B-tree*
sorted array
lookup cost
update cost

@SIGMOD17/18


log
one model
LSM-tree*
tiering
B-tree*
size
ratio
leveling
sorted array
lookup cost
update cost

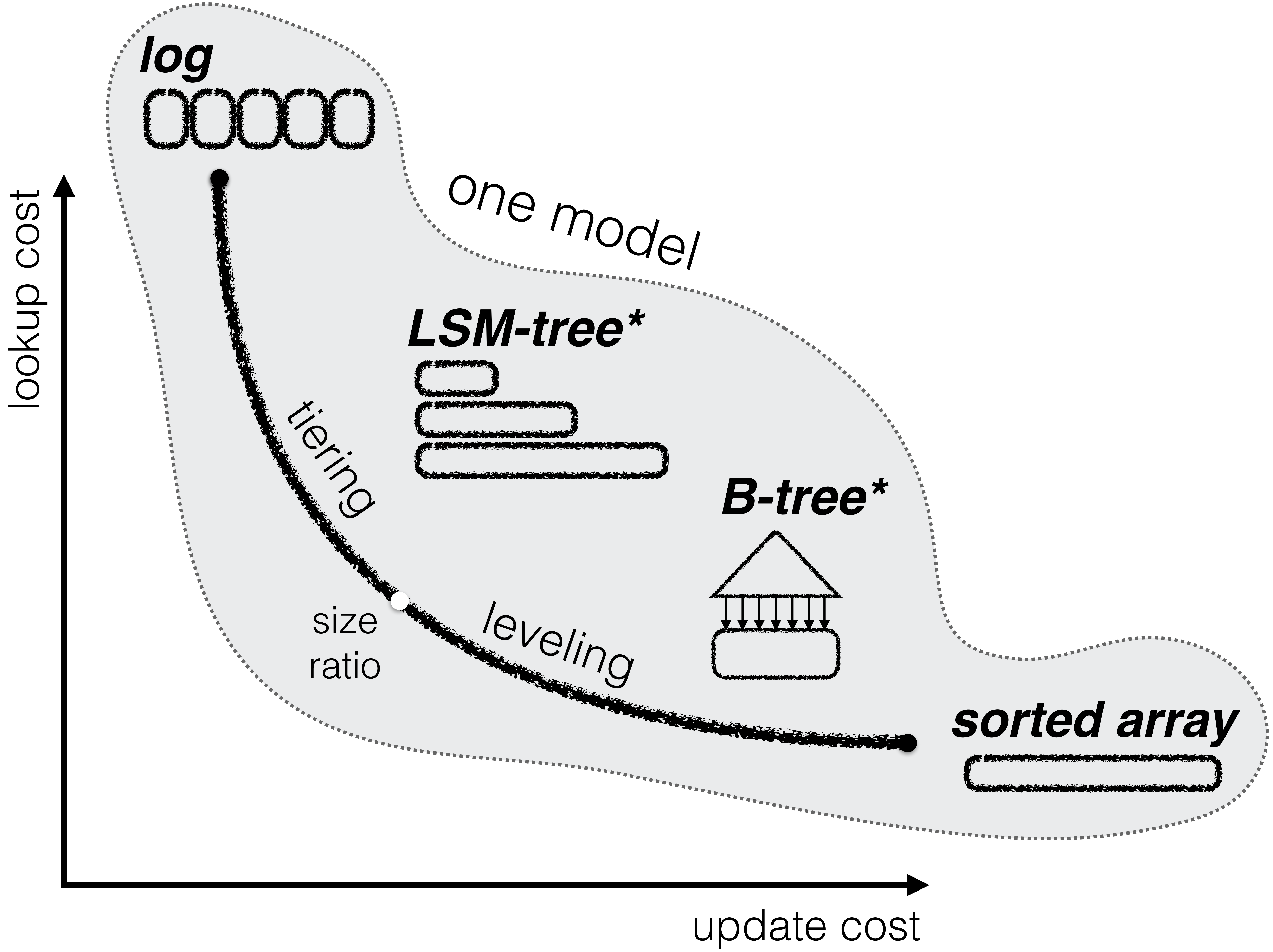


few continuous
parameters
&
closed form
formulas for metrics



few continuous
parameters
&
closed form
formulas for metrics

near **instant** design space navigation



few continuous parameters & **closed form** formulas for metrics

⬇

near **instant** design space navigation

⬇

CrimsonDB

a self-designing key-value store

S. BING YAO
models/advisors

DON BATORY
modular synthesis

JOE HELLERSTEIN
extensible indexing

STEFAN MANEGOLD
model synthesis

DASlab
@ Harvard SEAS

daslab.seas.harvard.edu

calculator infrastructure

DESIGN SPACE (updates, concurrency)

COST SYNTHESIS (accuracy, scalability)

EASY EXTENSIBILITY (plug & play rules)

study structure & gaps

DESIGN CONTINUUMS (optimizations)

GO AFTER GAPS (new design opport.)

DESIGN GUIDE (static design rule sys)

building more on top of

AUTO-SEARCH (ML/algo hybrids, hints)

SELF-DESIGNING (log to sorted arrays)

DSL & COMPILERS (productivity & perf)

DASlab
@ Harvard SEAS

daslab.seas.harvard.edu

THANKS!

Manos Athanassoulis
Postdoctoral Researcher

Niv Dayan
Postdoctoral Researcher

Kostas Zoumpatianos
Postdoctoral Researcher

Michael Kester
Ph.D. Researcher

Lukas Maas
Ph.D. Researcher

Abdul Wasay
Ph.D. Researcher

Brian Hentschel
Ph.D. Researcher

Wilson Qin
Ph.D. Researcher

Angelo Kastroulis
Graduate Researcher

Franco Solleza
Graduate Researcher

Mali Akmanalp
Graduate Researcher

Rachna Sha
Graduate Researcher

Dhruv Gupta
Undergraduate Researcher

Demi Guo
Undergraduate Researcher

Yiyou Sun
Undergraduate Research Intern

Mo Sun
Undergraduate Research Intern

Yuze Liao
Undergraduate Research Intern

calculator infrastructure

DESIGN SPACE (updates, concurrency)

COST SYNTHESIS (accuracy, scalability)

EASY EXTENSIBILITY (plug & play rules)

study structure & gaps

DESIGN CONTINUUMS (optimizations)

GO AFTER GAPS (new design opport.)

DESIGN GUIDE (static design rule sys)

building more on top of

AUTO-SEARCH (ML/algo hybrids, hints)

SELF-DESIGNING (log to sorted arrays)

DSL & COMPILERS (productivity & perf)