CS 265

Stratos Idreos

BIG DATA SYSTEMS

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

Logistics:

Class forum (Ed): this weekend

Labs start next week
Three times a week. (schedule on class website).
Labs are for systems projects only. Research projects will have diff sessions.

Logistics:

Class forum (Ed): this weekend

Labs start next week
Three times a week. (schedule on class website).
Labs are for systems projects only. Research projects will have diff sessions.

Systems projects can start as of next week
Step 1: Go to labs to start to understand what is needed and how to start.
TFs will also release two intro sections next week

Logistics on NN Systems Project

Second MLsys systems project ready.
Optimize data movement for neural network training
One vision and one LLM model.
Implementing M2 paper.
Available on the class website.

μ-TWO: 3x Faster Multi-model Training with Orchestration and Memory Optimization

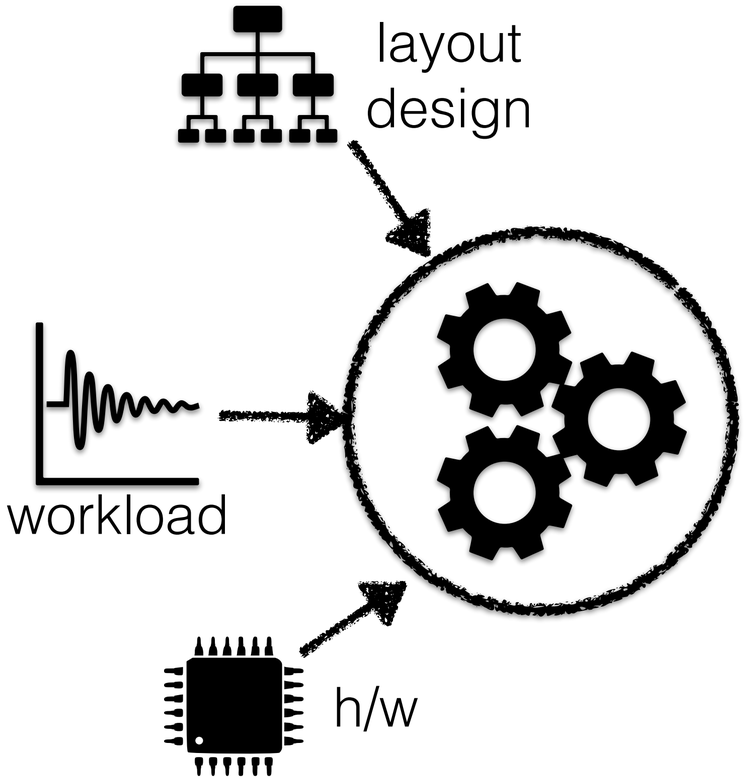
“Can I propose an idea for a research project?”

Absolutely. If it fits the following questions:

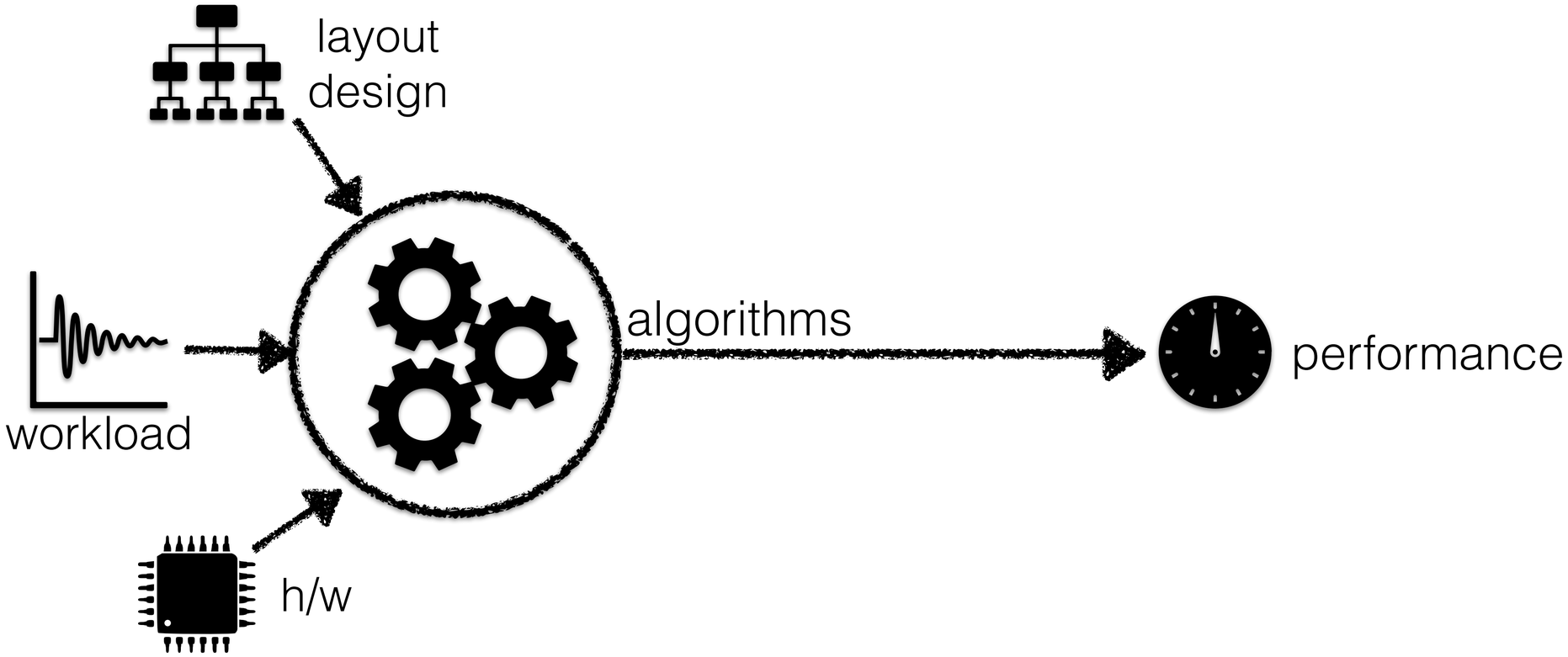
Making SQL,NoSQL, LLMs, Image AI:
faster, understanding design space, adding design automation

What do we want to achieve: **what if design example**

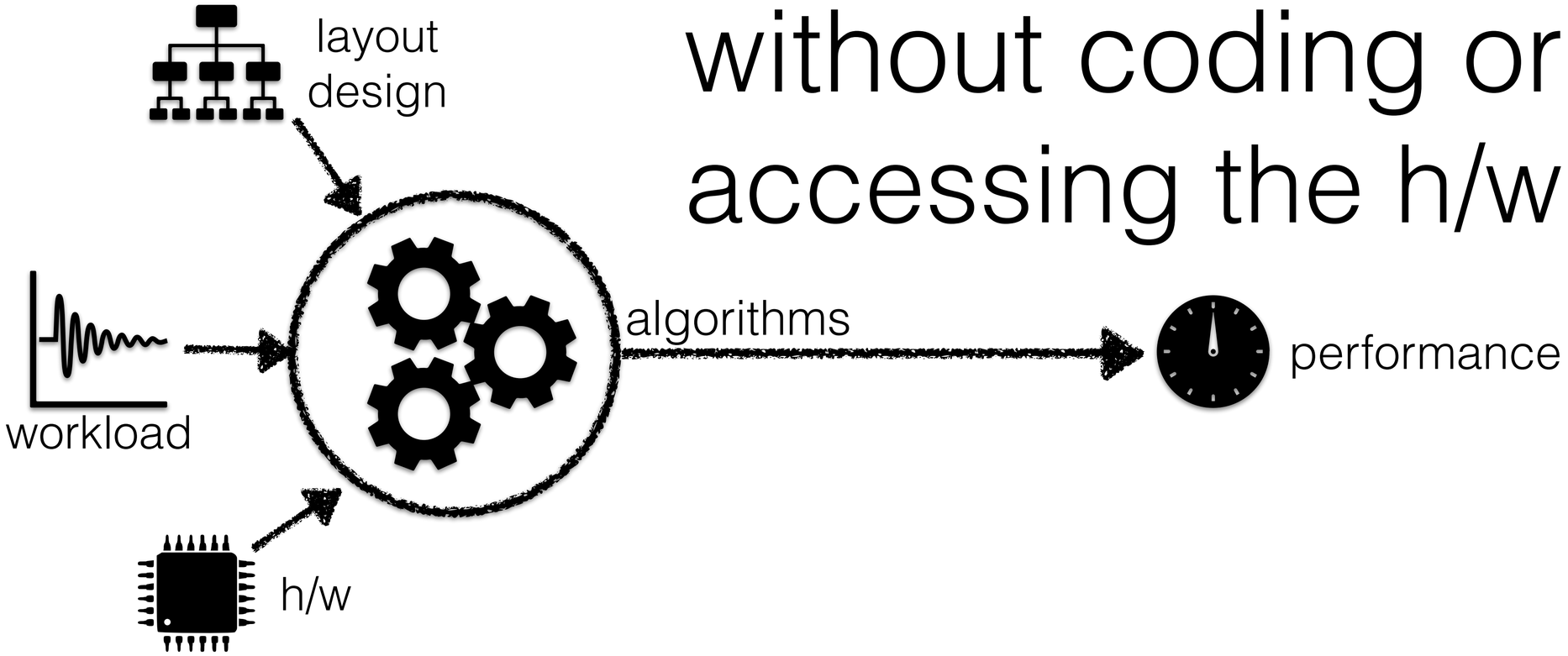
Data Calculator



Data Calculator

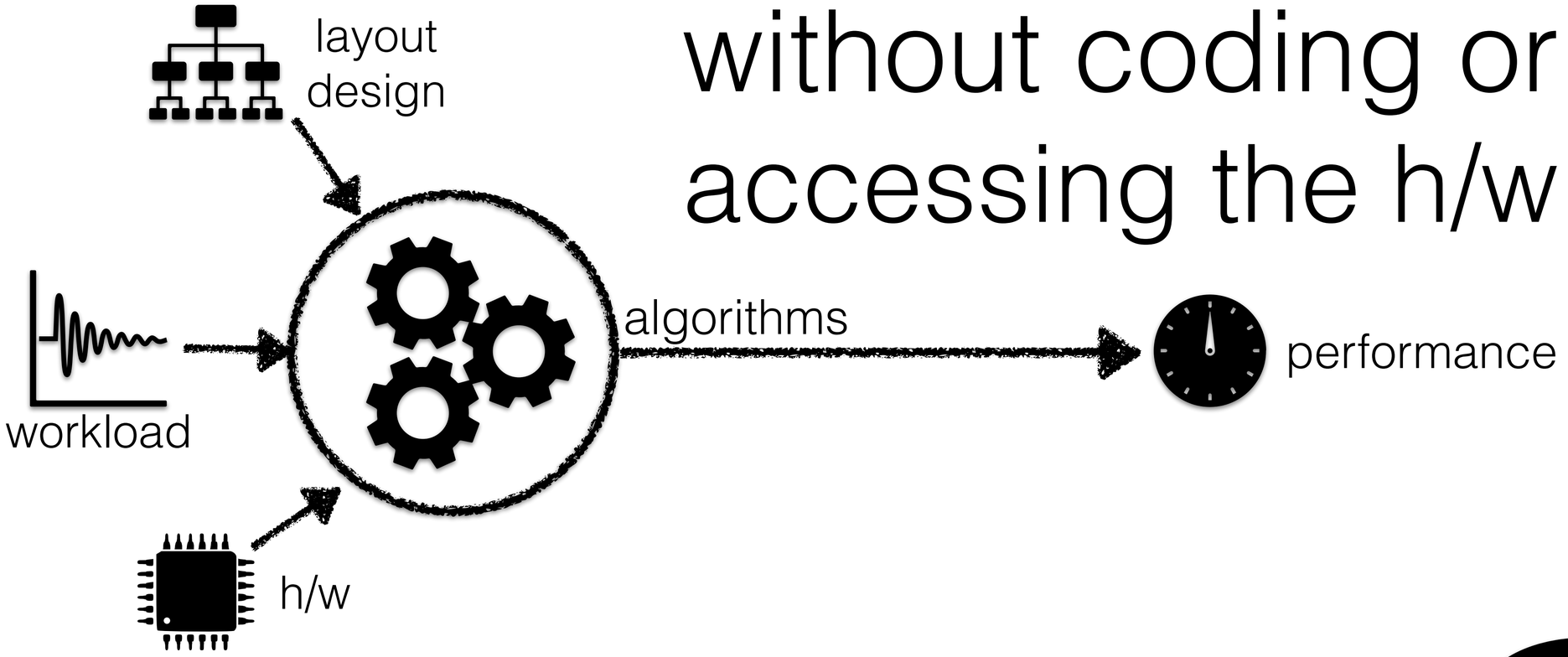


Data Calculator



without coding or accessing the h/w

Data Calculator



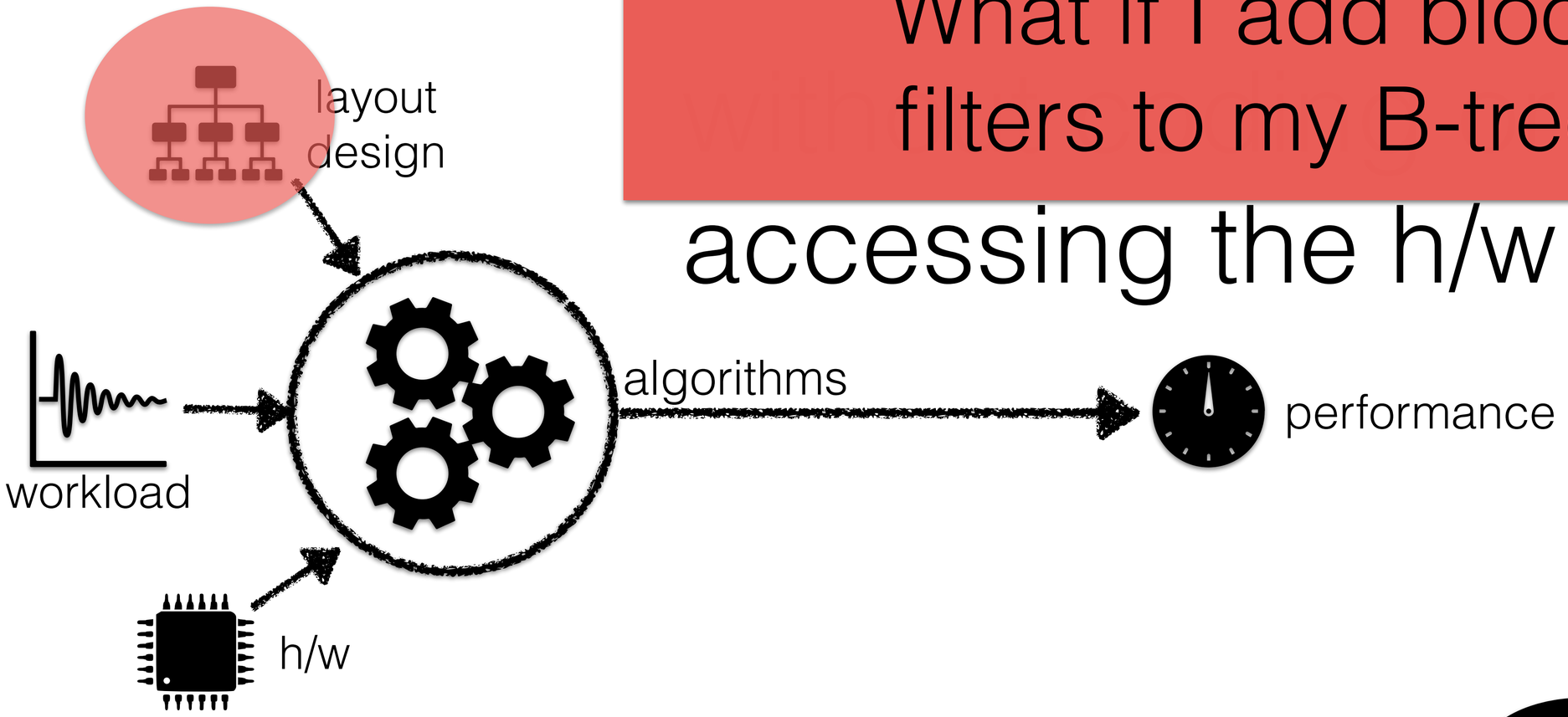
without coding or accessing the h/w

what-if design

Data Calculator

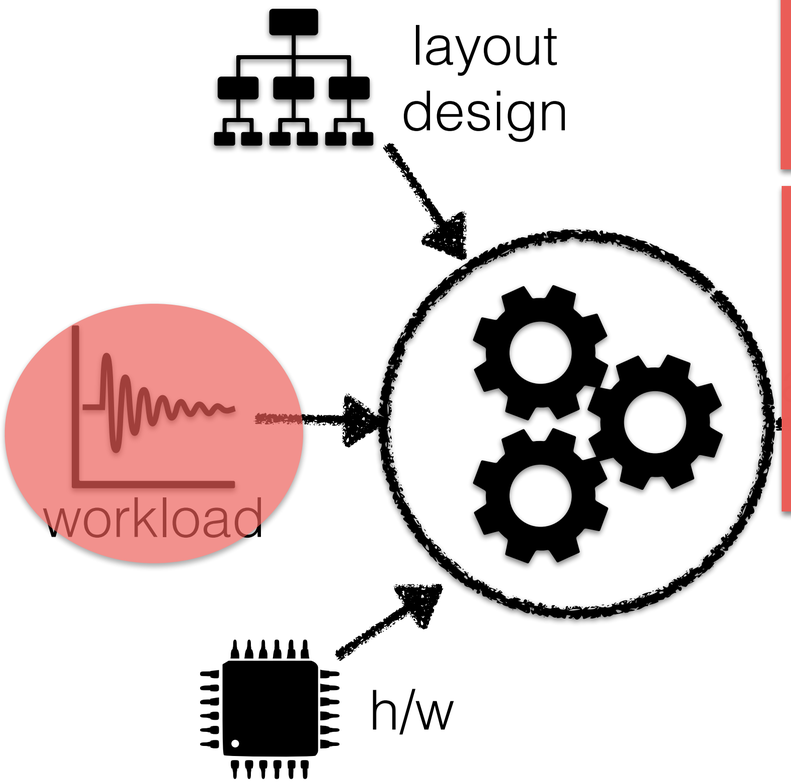
What if I add bloom filters to my B-tree?

accessing the h/w



what-if design

Data Calculator

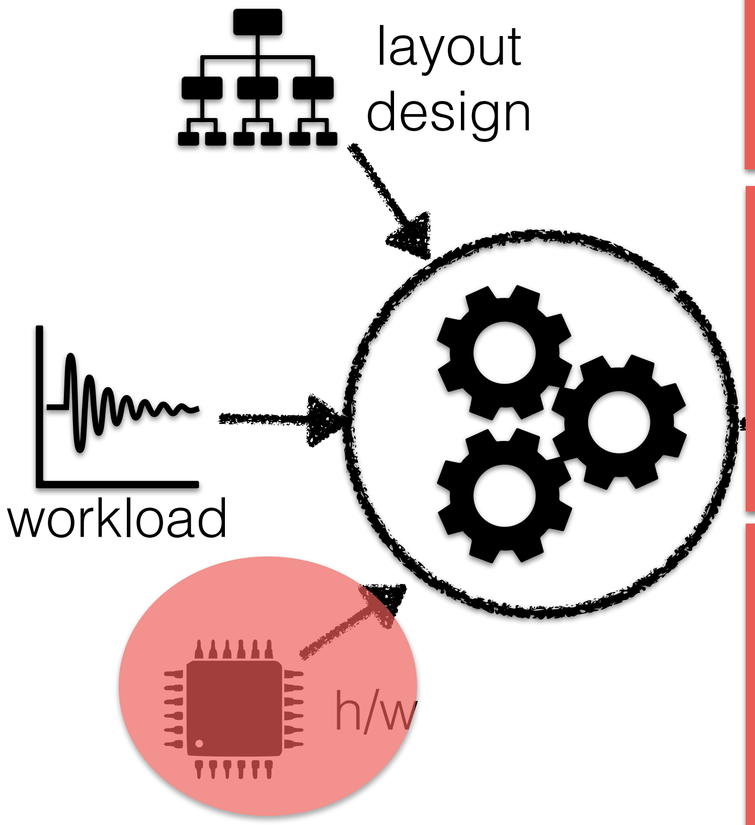


What if I add bloom filters to my B-tree?

What if I add feature X that brings 60% more writes?

what-if design

Data Calculator



What if I add bloom filters to my B-tree?

What if add feature X that brings 60% more writes?

What if I need to reduce memory by 50%?

what-if design

Data Calculator

Cost in Amazon Cloud?

What if I add bloom filters to my B-tree?

Which workload breaks my system?

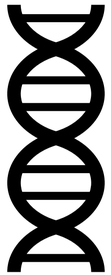
What if add feature X that brings 60% more writes?

Should I buy new hardware X?

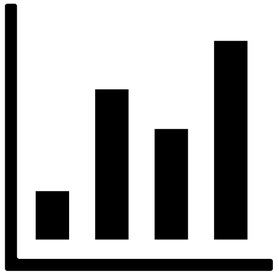
What if I need to reduce memory by 50%?

what-if design

Three steps required



DESIGN SPACE



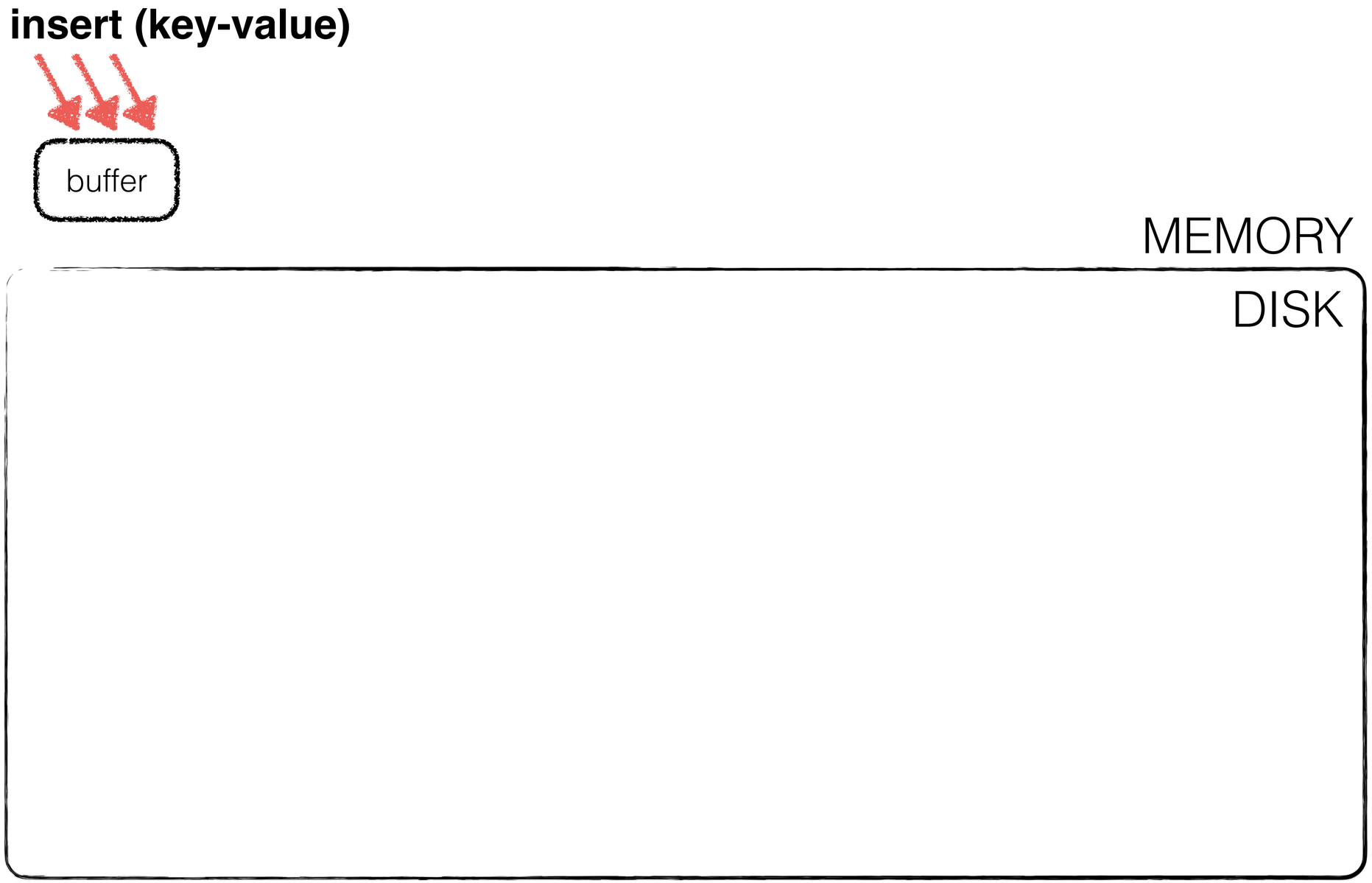
COST SYNTHESIS

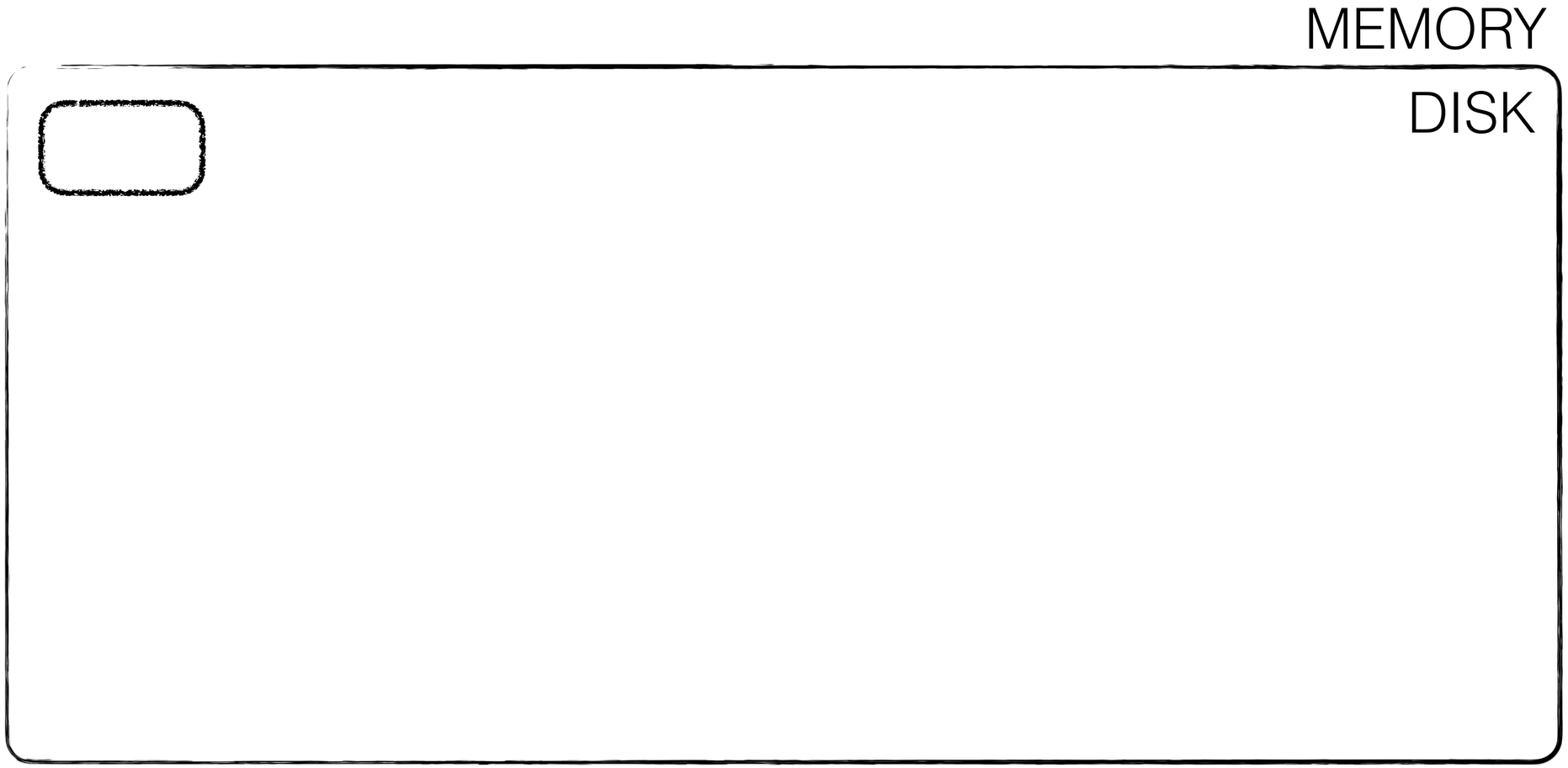
WHAT-IF

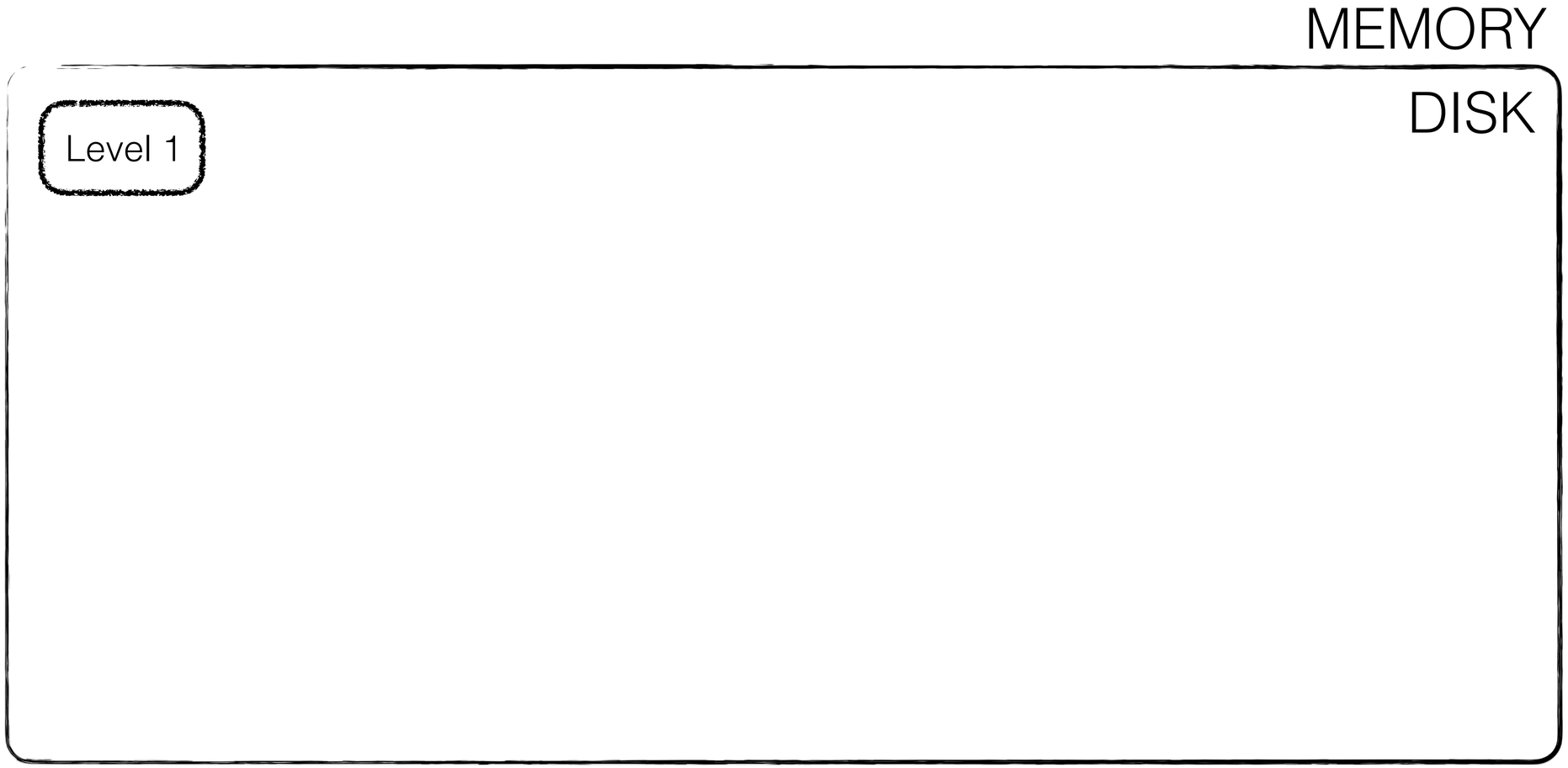
Today:

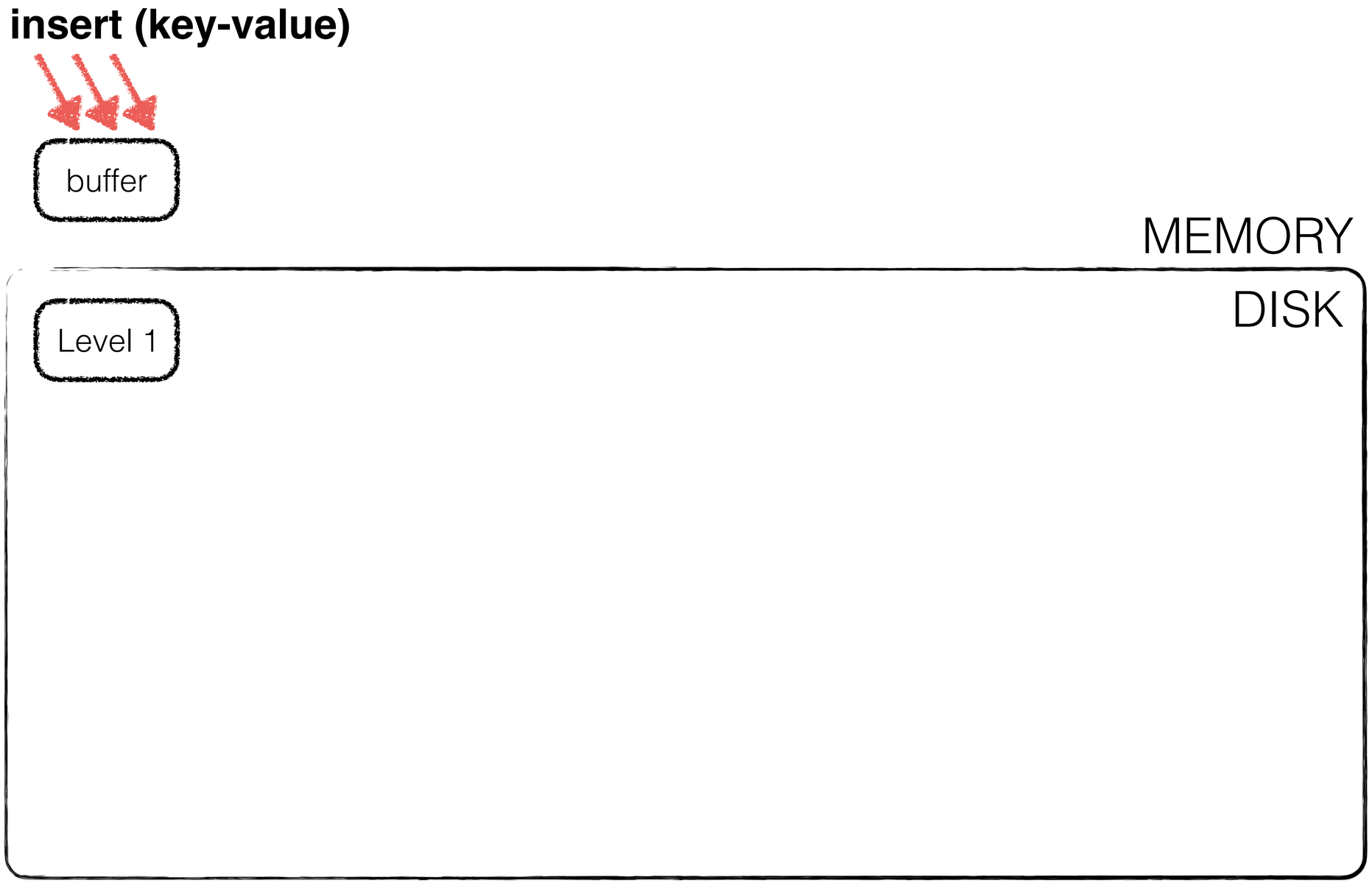
Building a design space in detail: Data structures

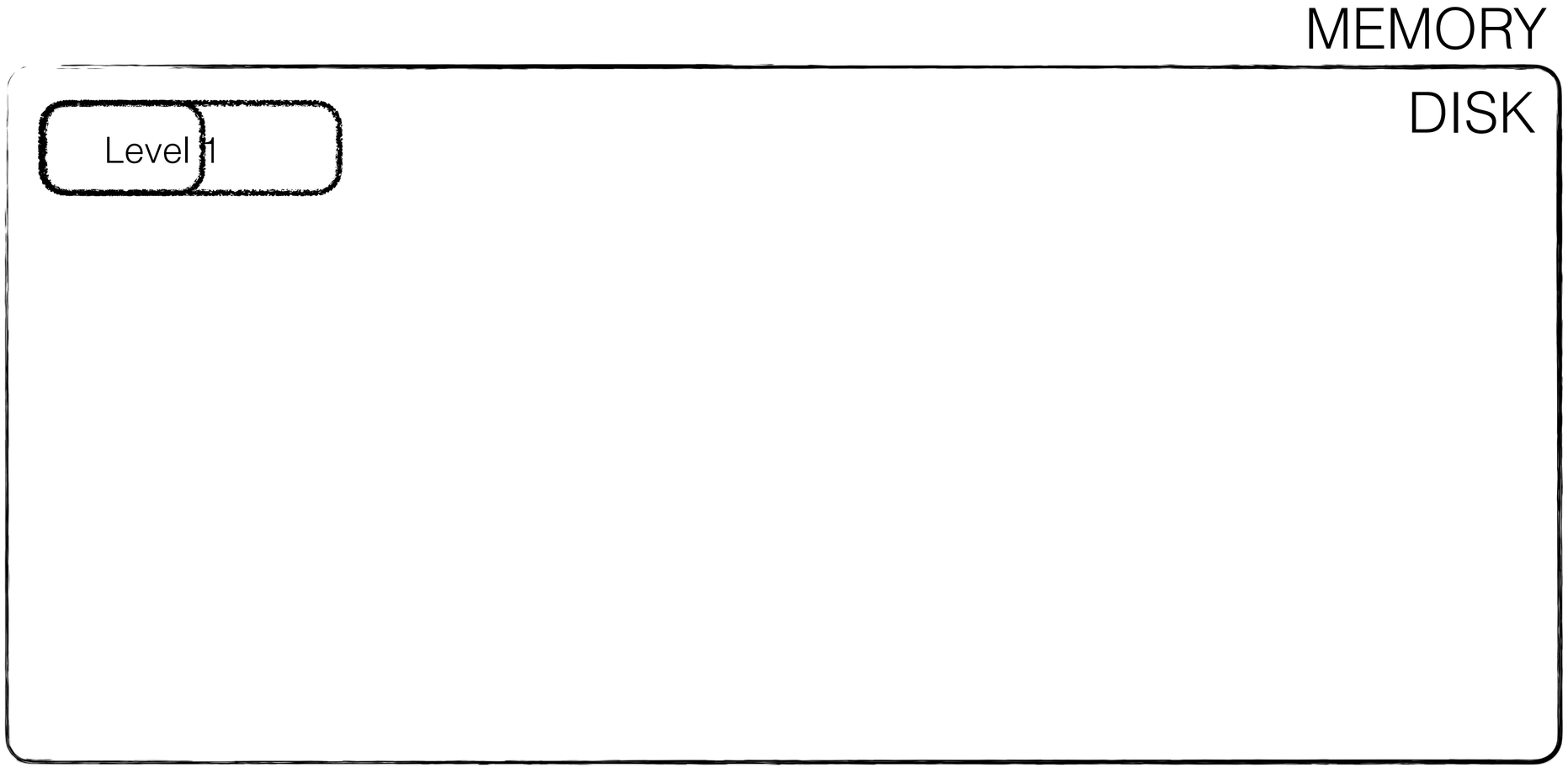
Next level of technical detail in KV-stores: merging/levels

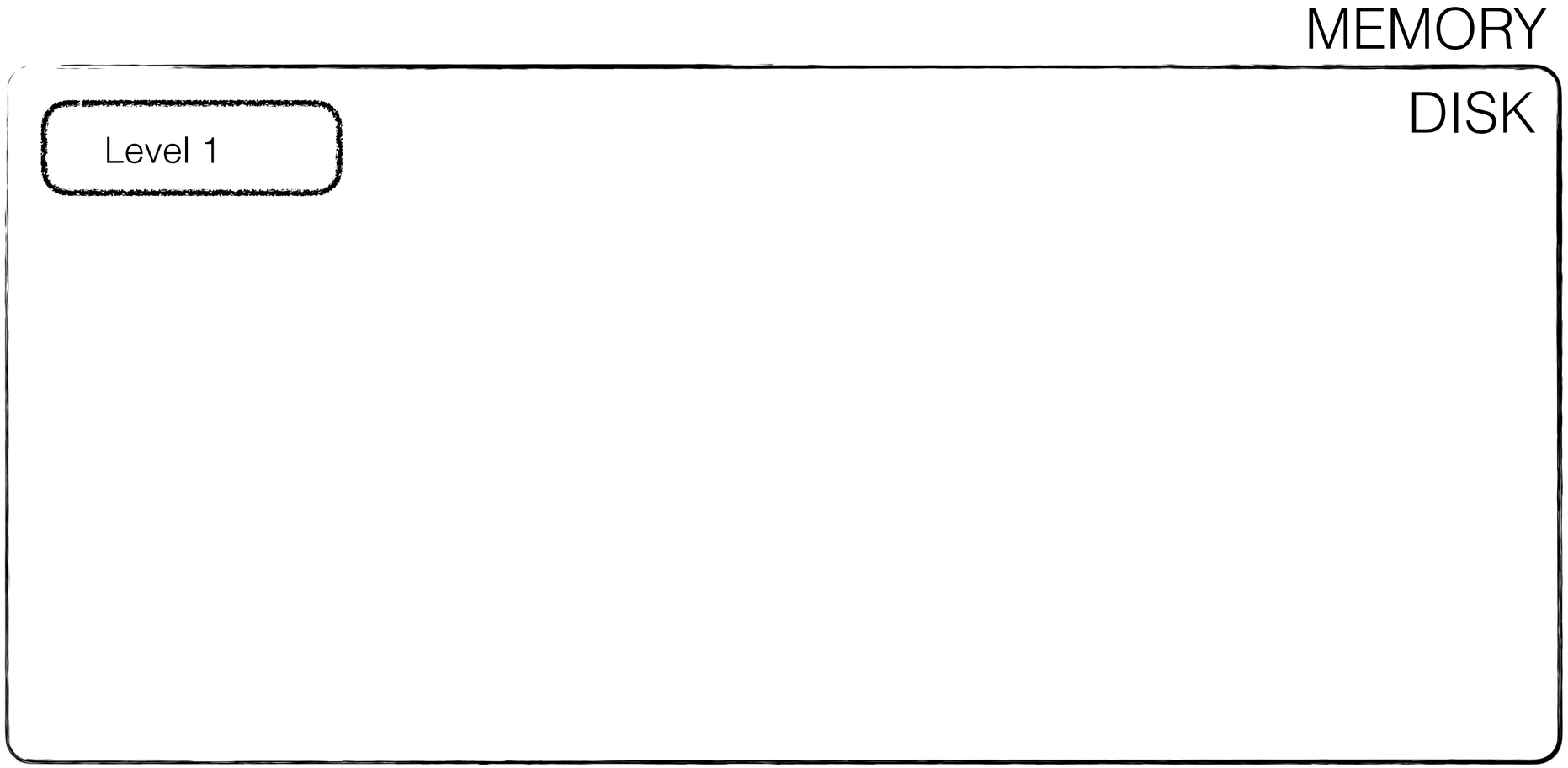

insert (key-value)
buffer
MEMORY
DISK

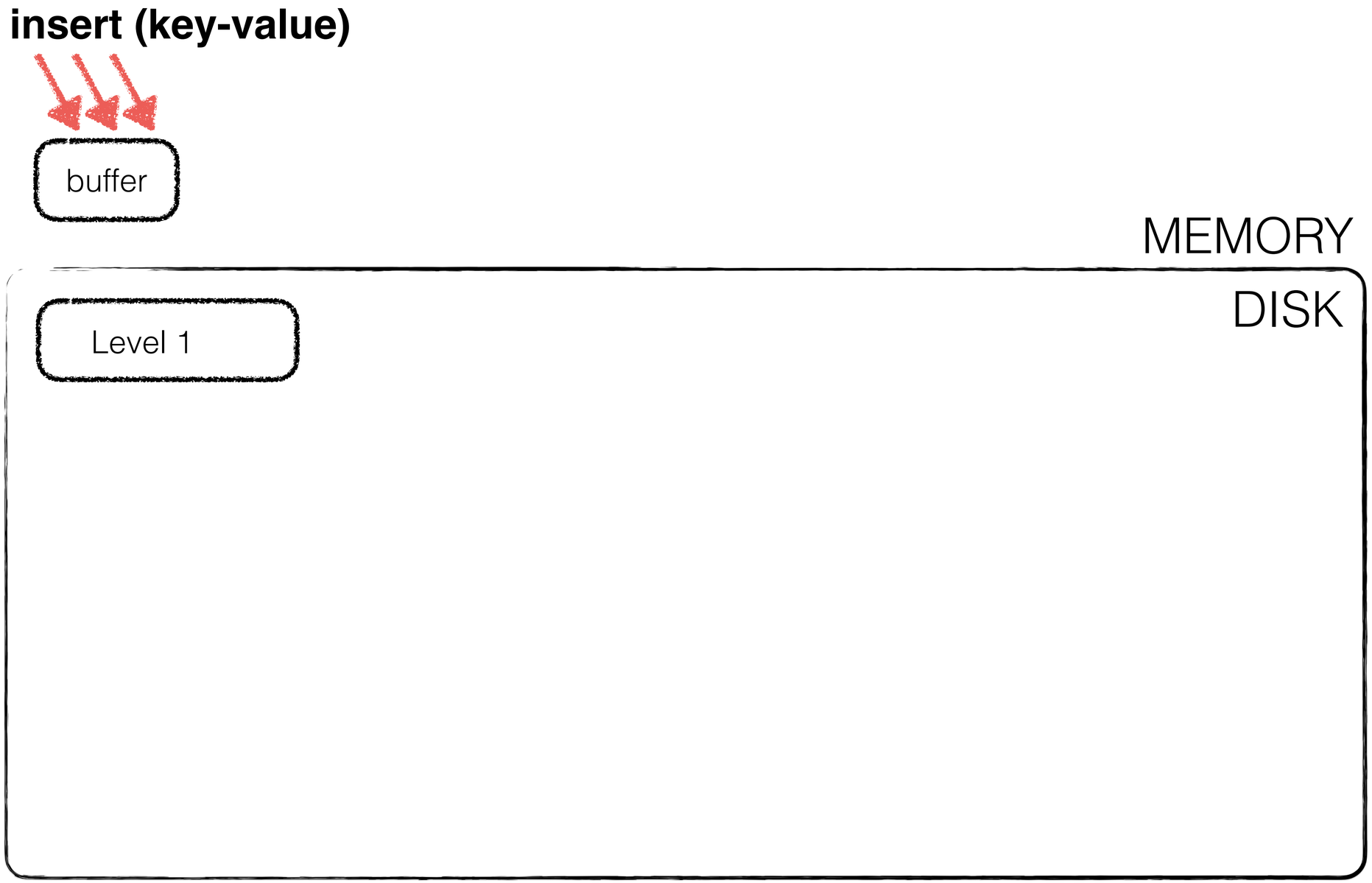

MEMORY
DISK

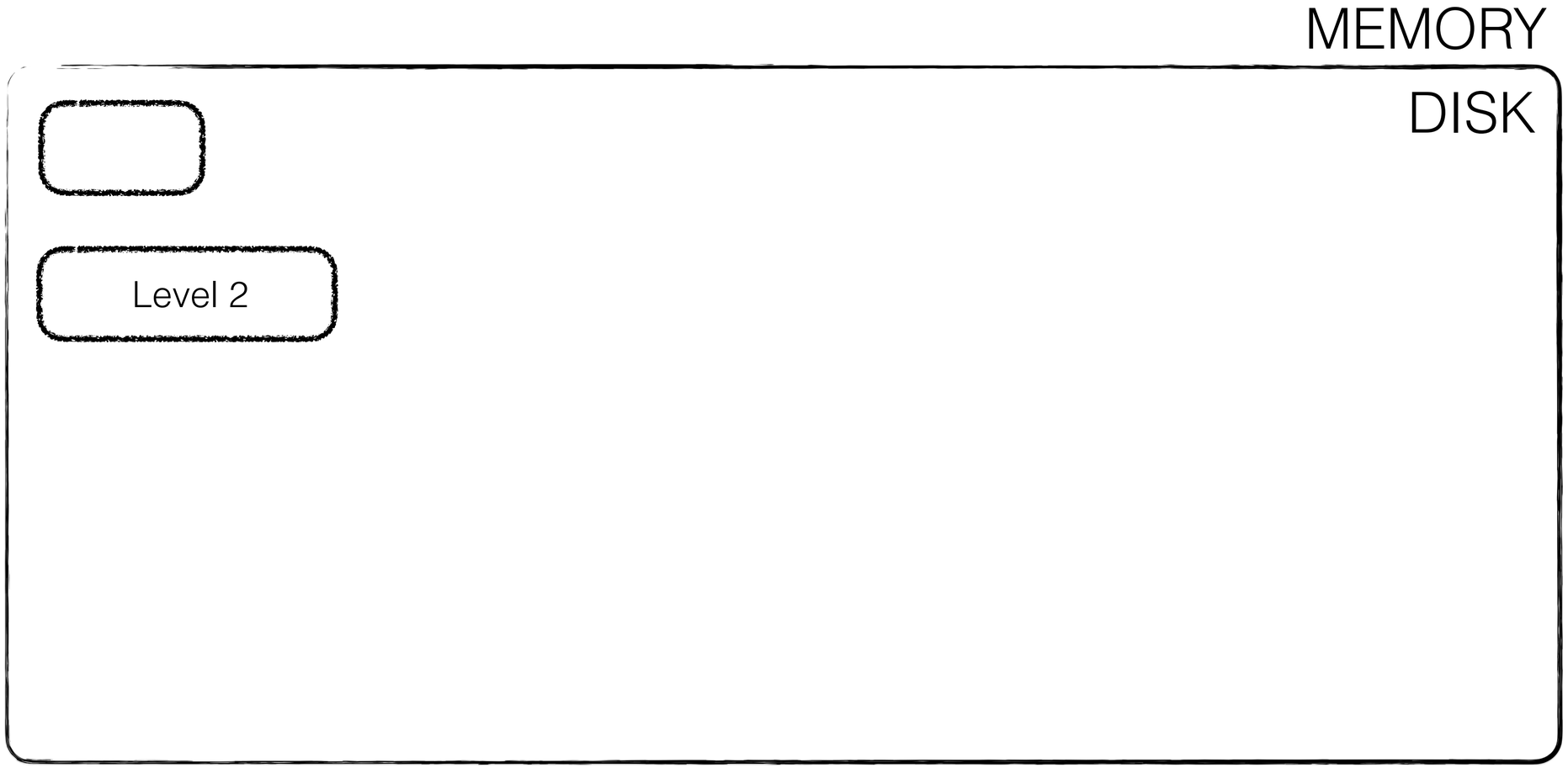

MEMORY
DISK
Level 1

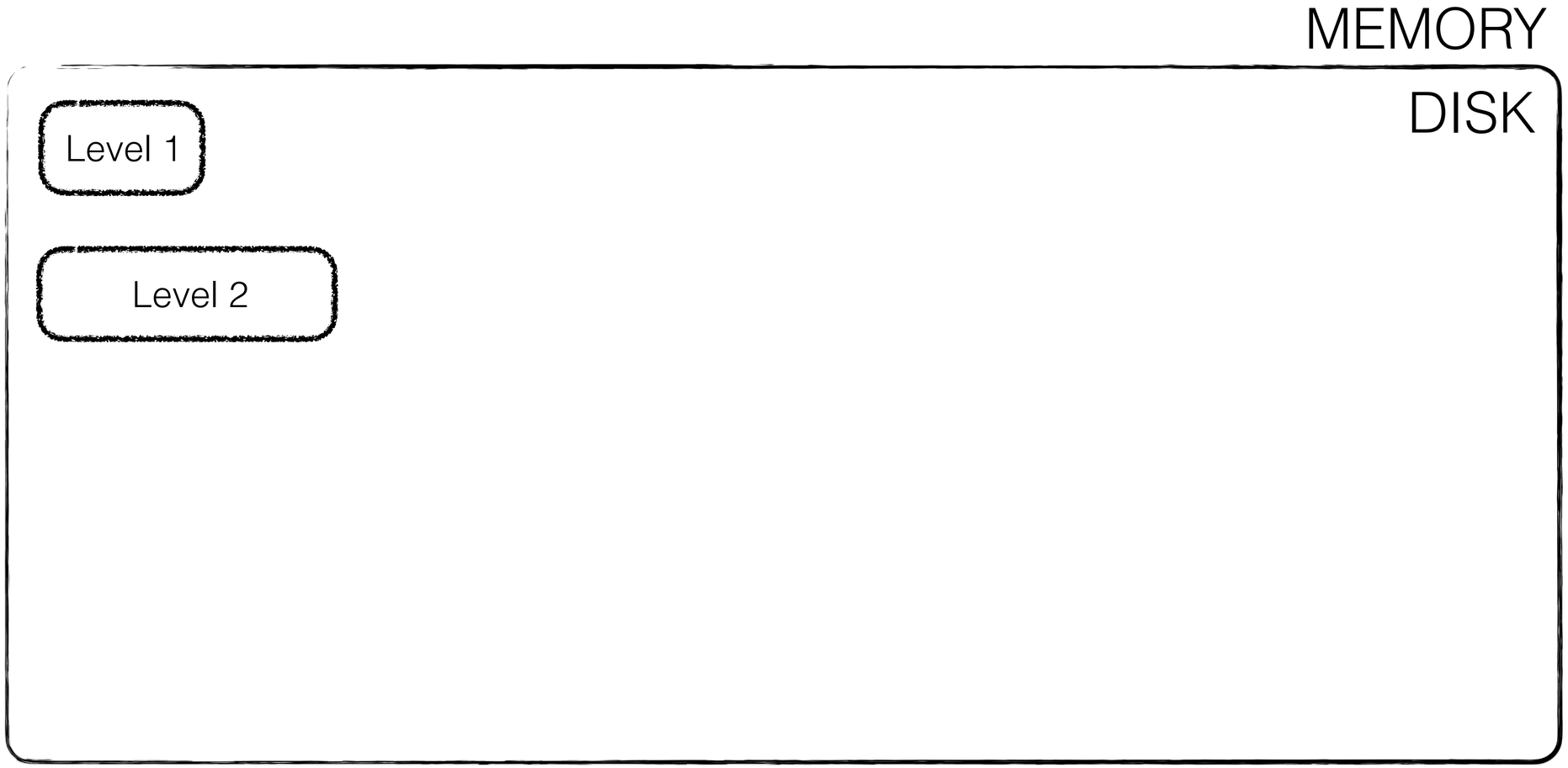

insert (key-value)
buffer
MEMORY
DISK
Level 1

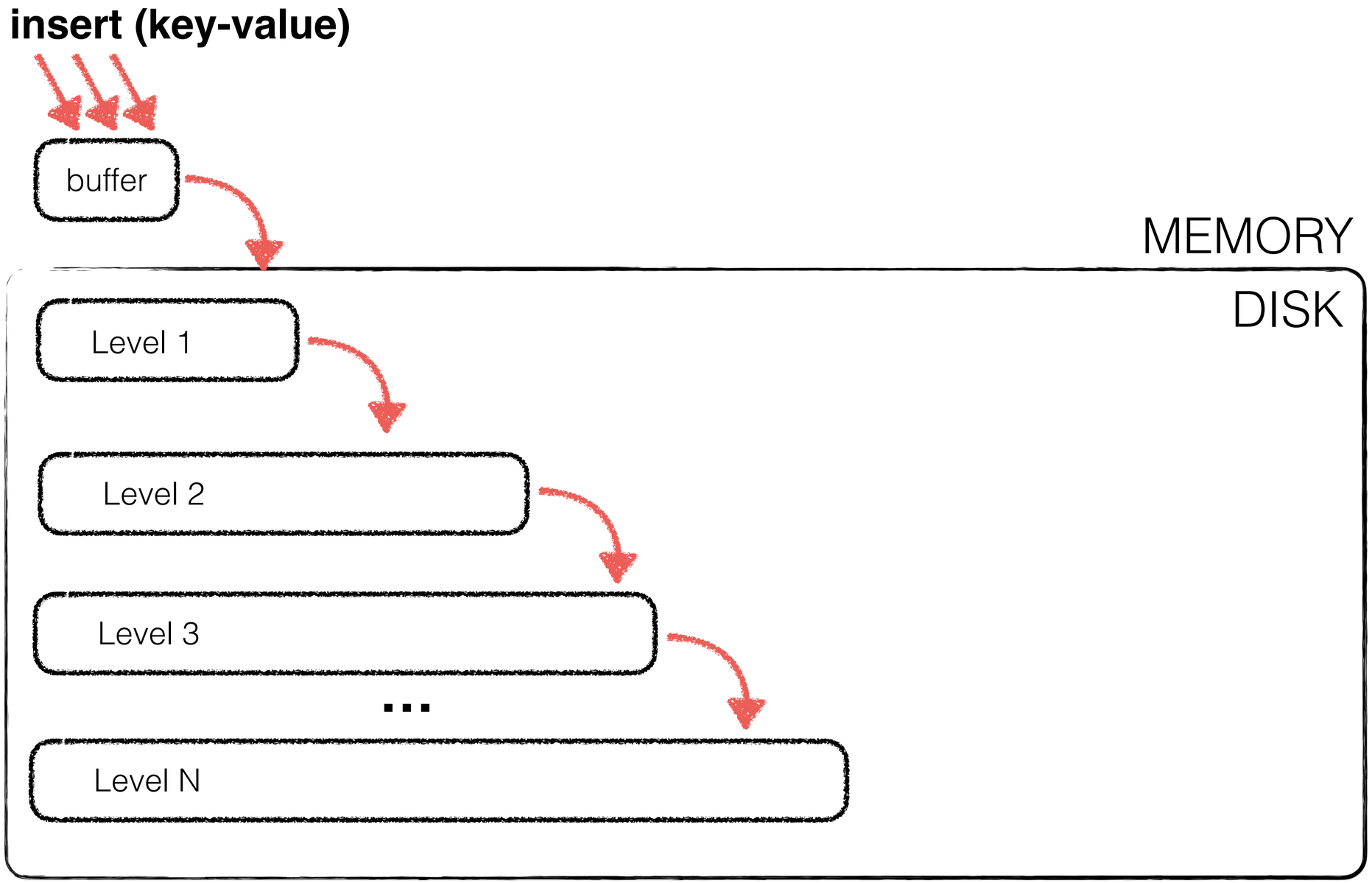

MEMORY
DISK
Level 1

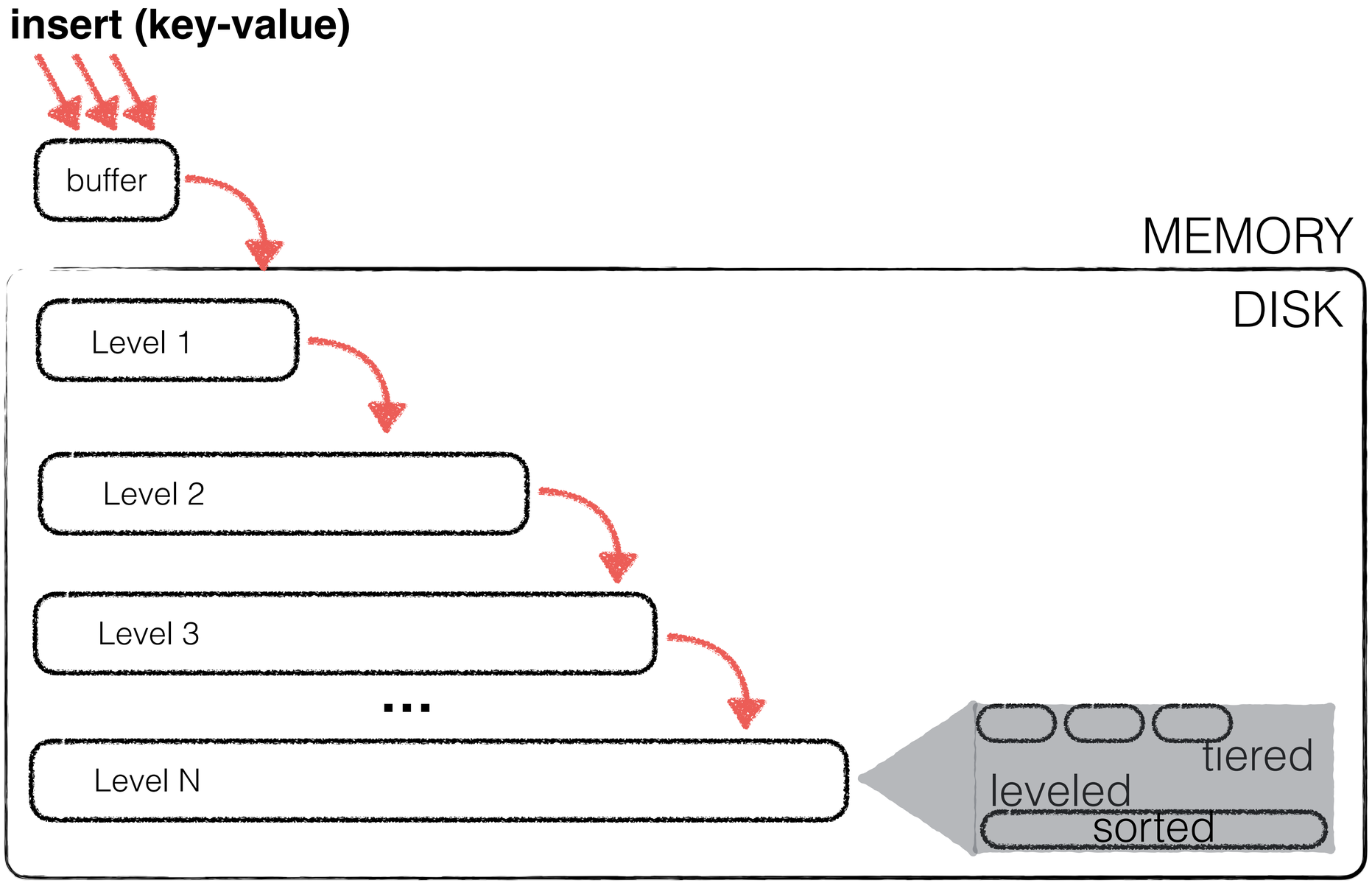

MEMORY
DISK
Level 1

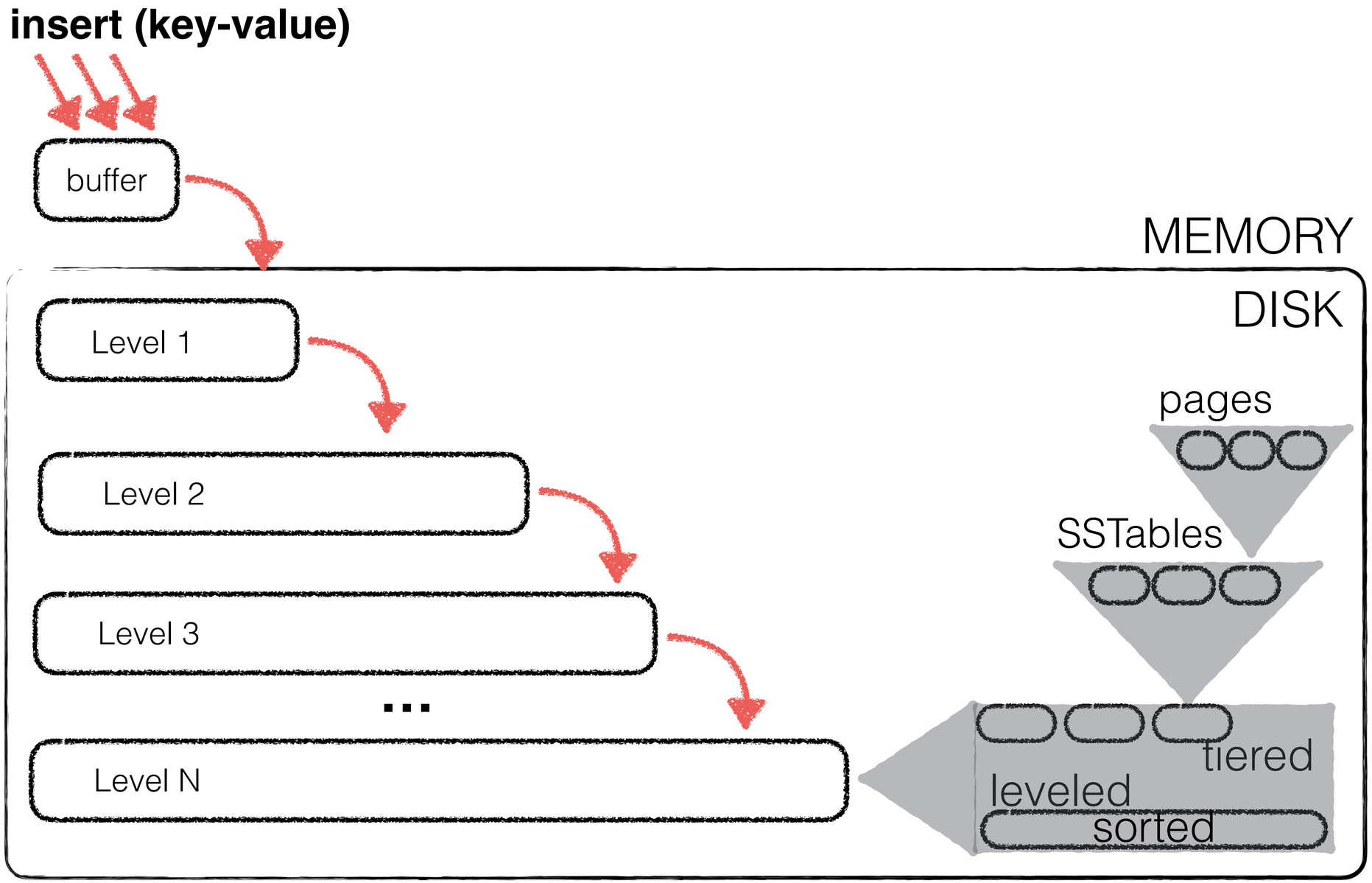

insert (key-value)
buffer
MEMORY
DISK
Level 1

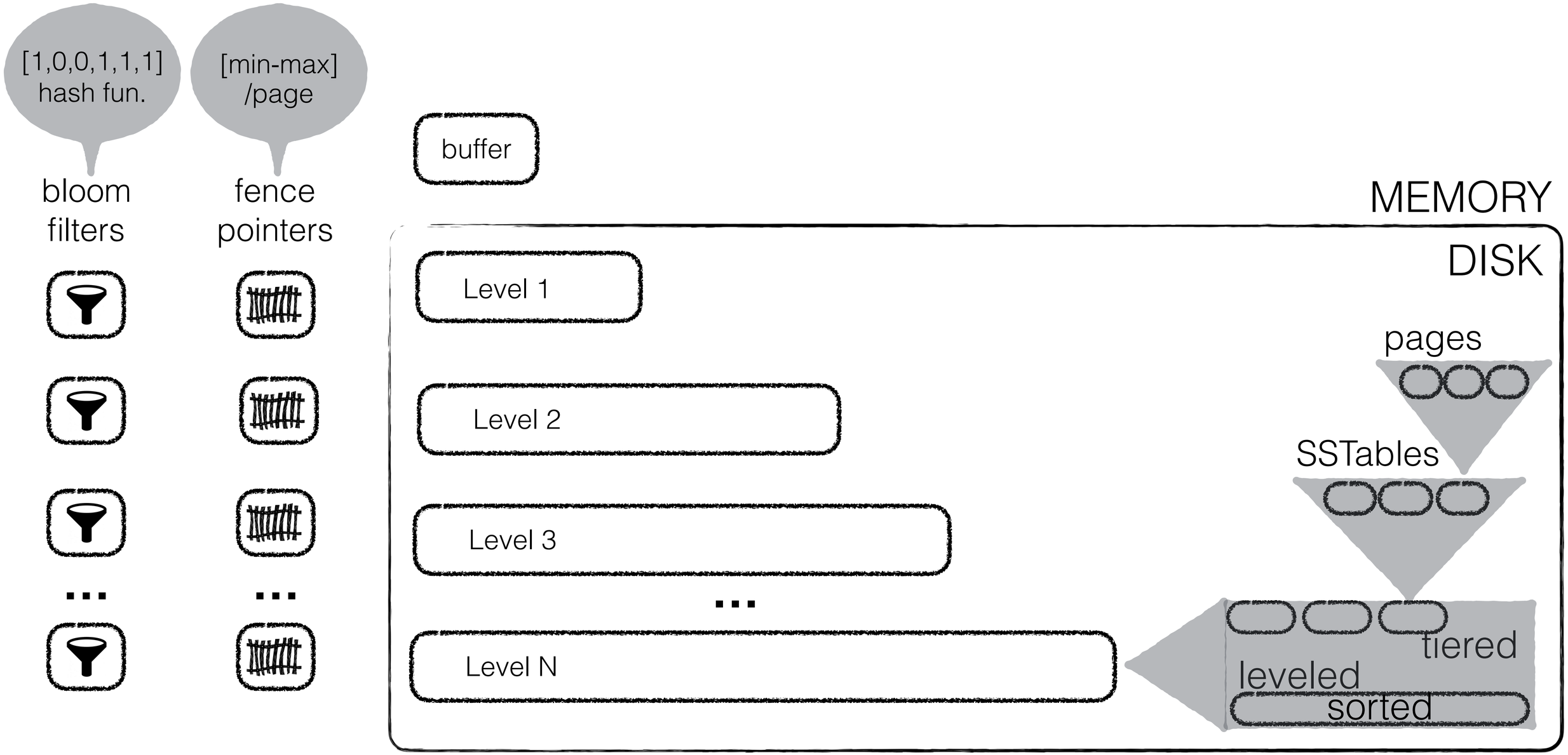

MEMORY
DISK
Level 2

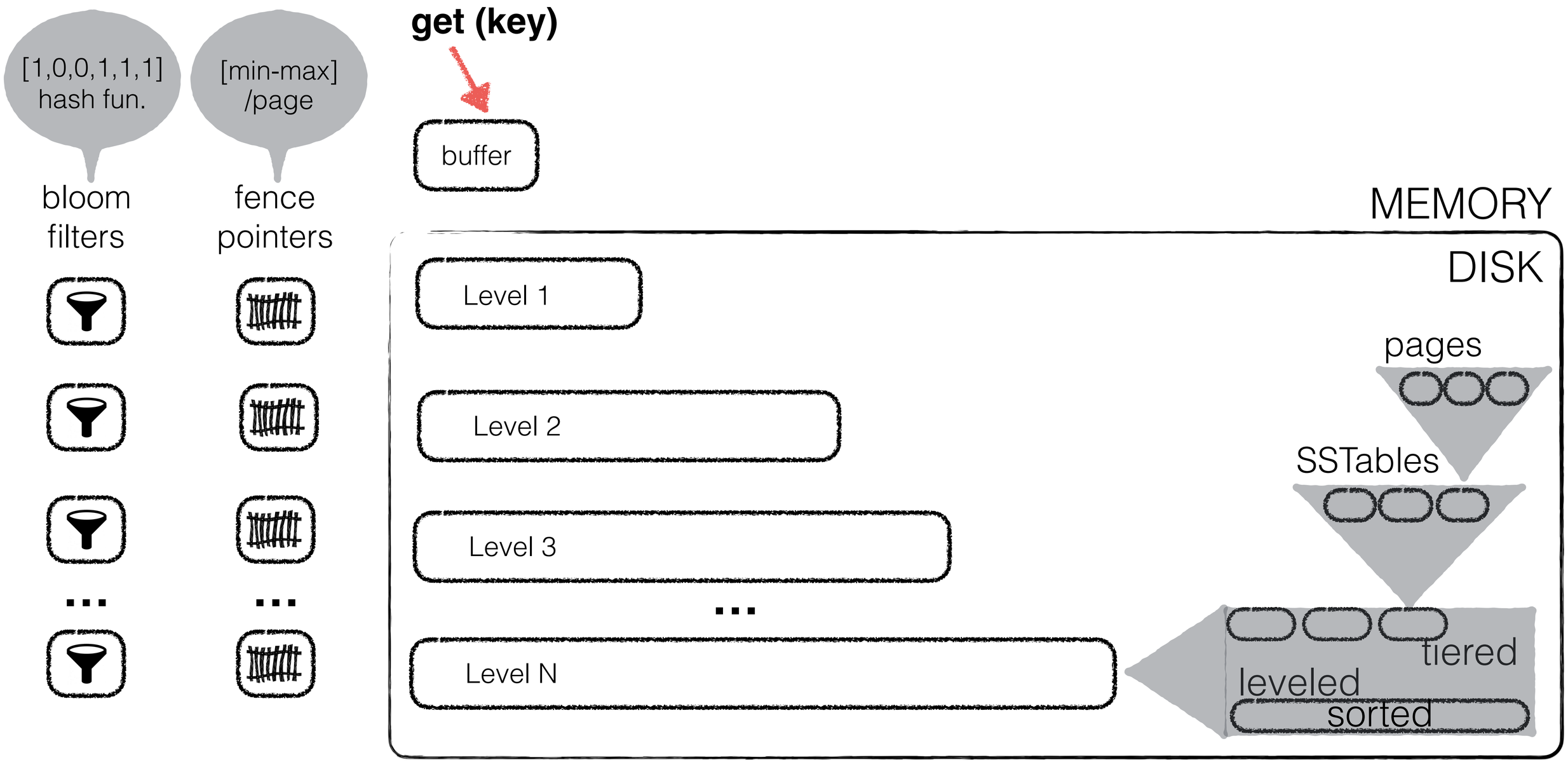

MEMORY
DISK
Level 1
Level 2

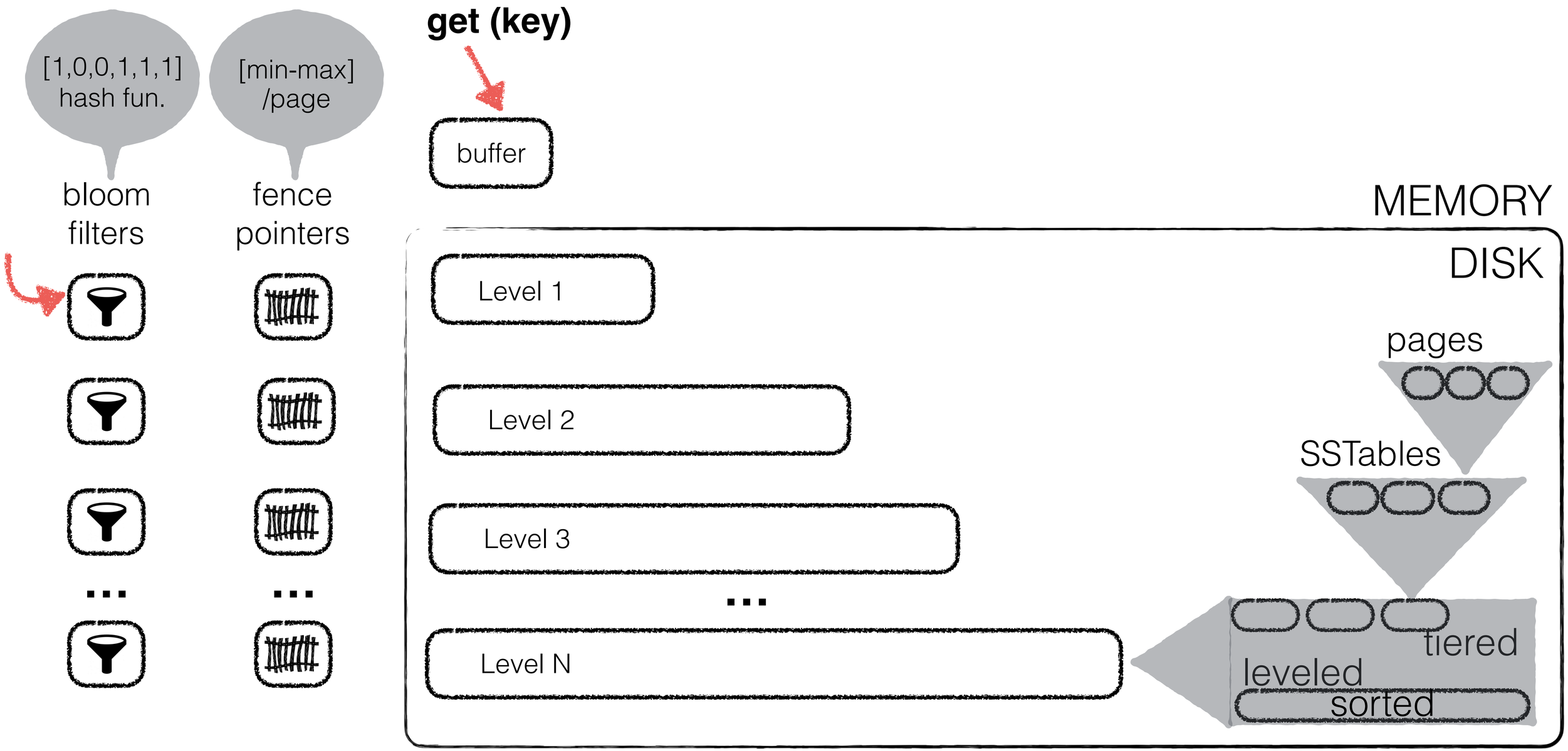

insert (key-value)
buffer
MEMORY
DISK
Level 1
Level 2
Level 3
...
Level N

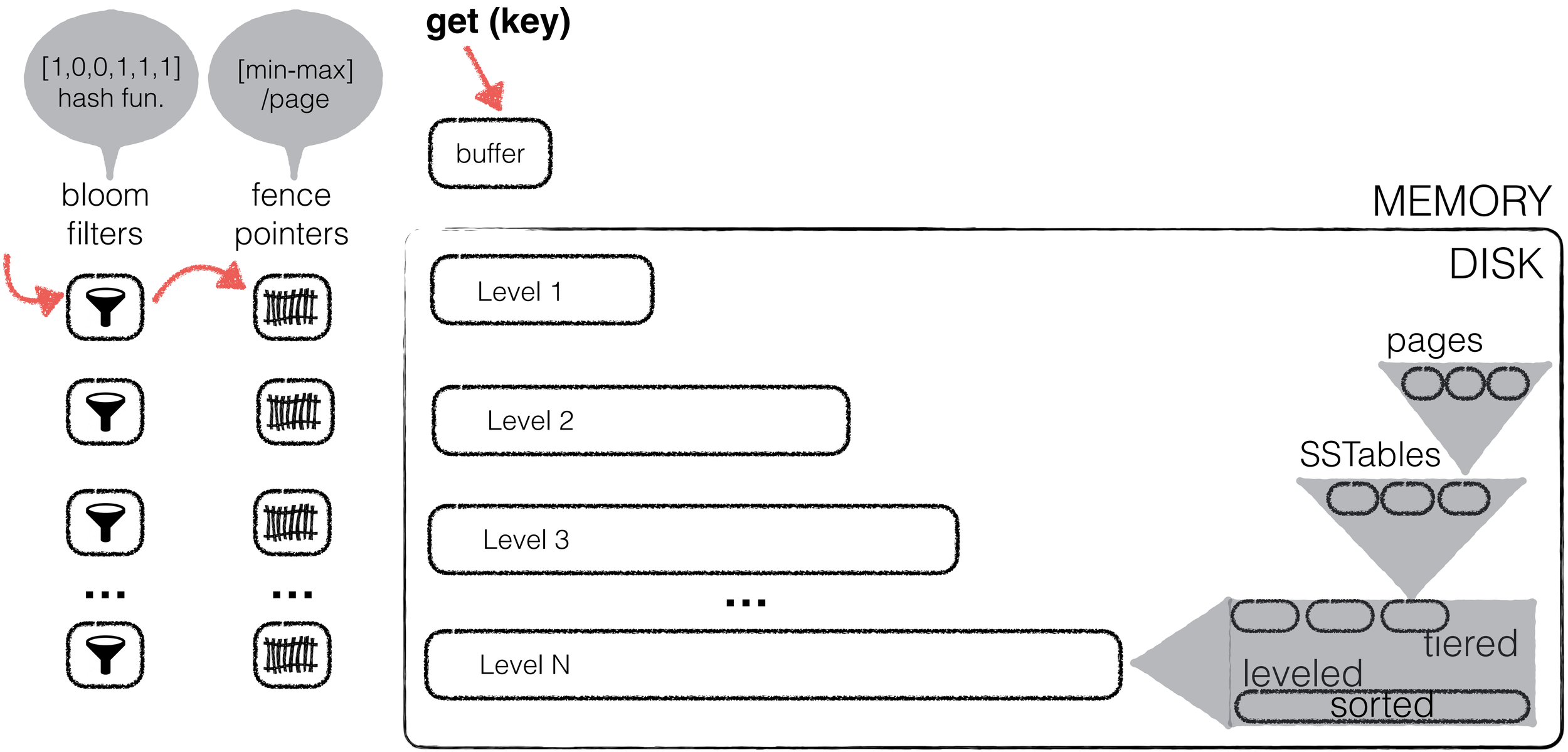

insert (key-value)
buffer
MEMORY
DISK
Level 1
Level 2
Level 3
...
Level N
tiered
leveled
sorted

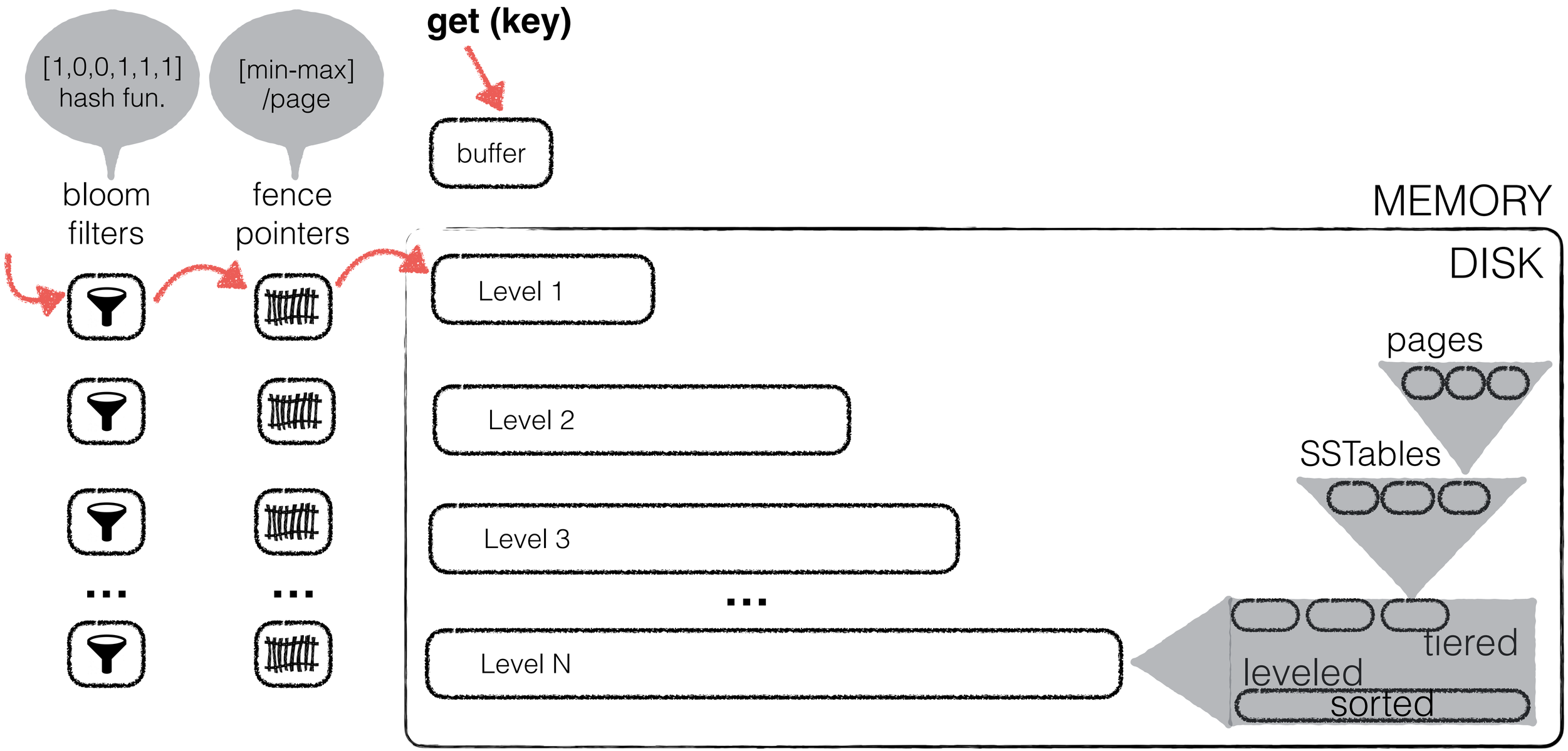

insert (key-value)
buffer
MEMORY
DISK
Level 1
Level 2
Level 3
...
Level N
pages
SSTables
tiered
leveled
sorted


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

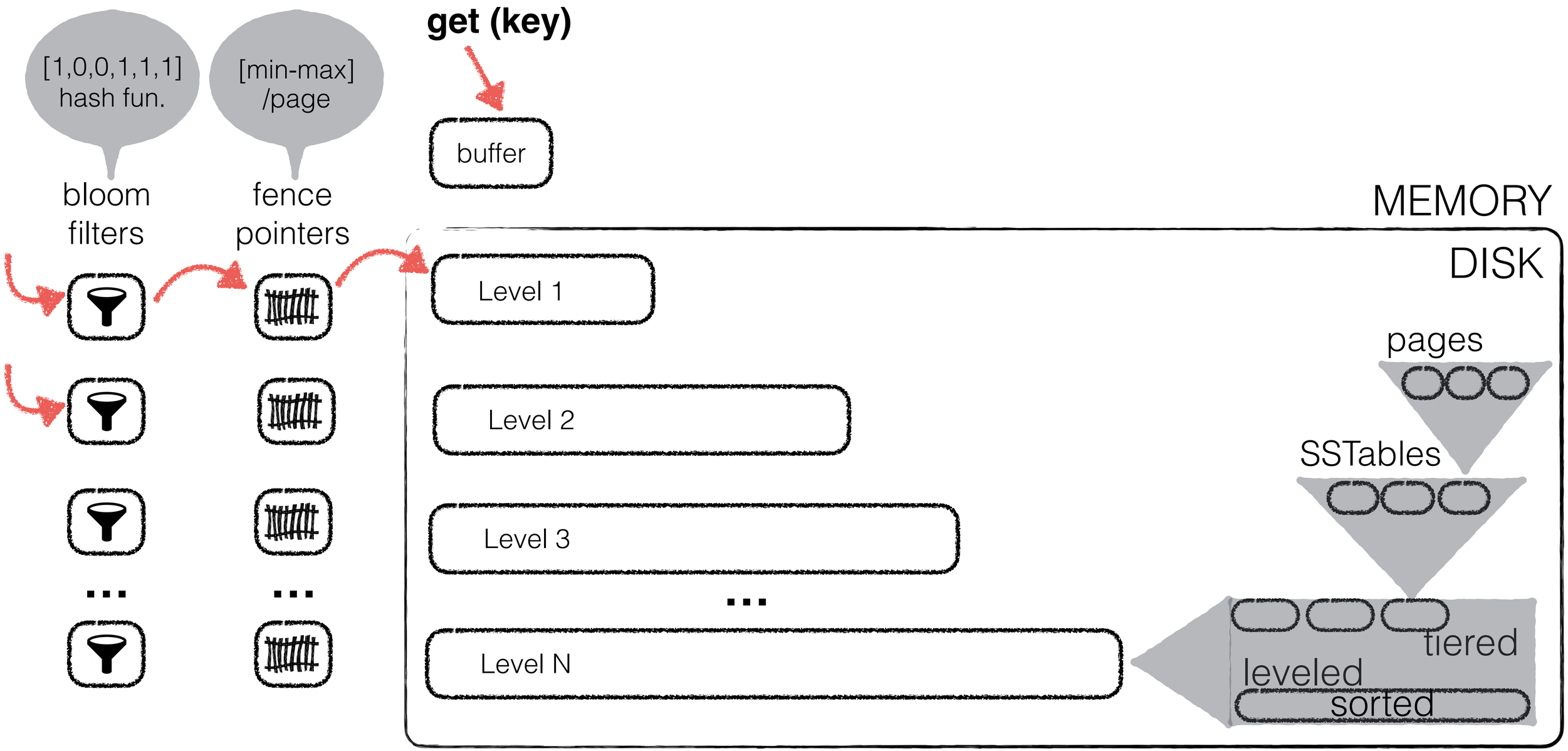

[1,0,0,1,1,1]
hash fun.
[min-max]
/page
bloom
filters
fence
pointers
...
...
get (key)
buffer
MEMORY
DISK
Level 1
Level 2
Level 3
...
Level N
pages
SSTables
tiered
leveled
sorted


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

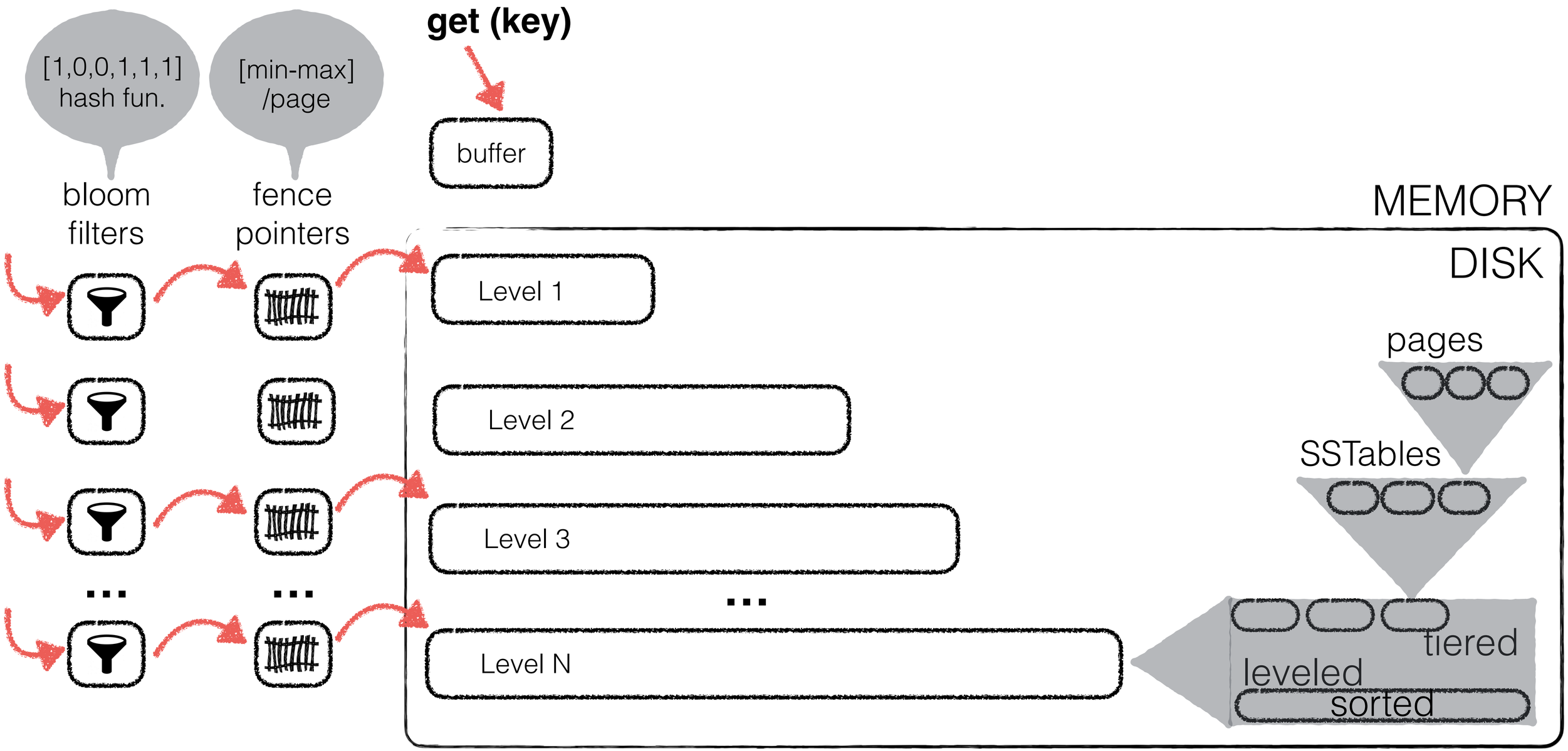

get (key)
[1,0,0,1,1,1]
hash fun.
[min-max]
/page
bloom
filters
fence
pointers
buffer
MEMORY
DISK
Level 1
Level 2
Level 3
...
Level N
pages
SSTables
tiered
leveled
sorted

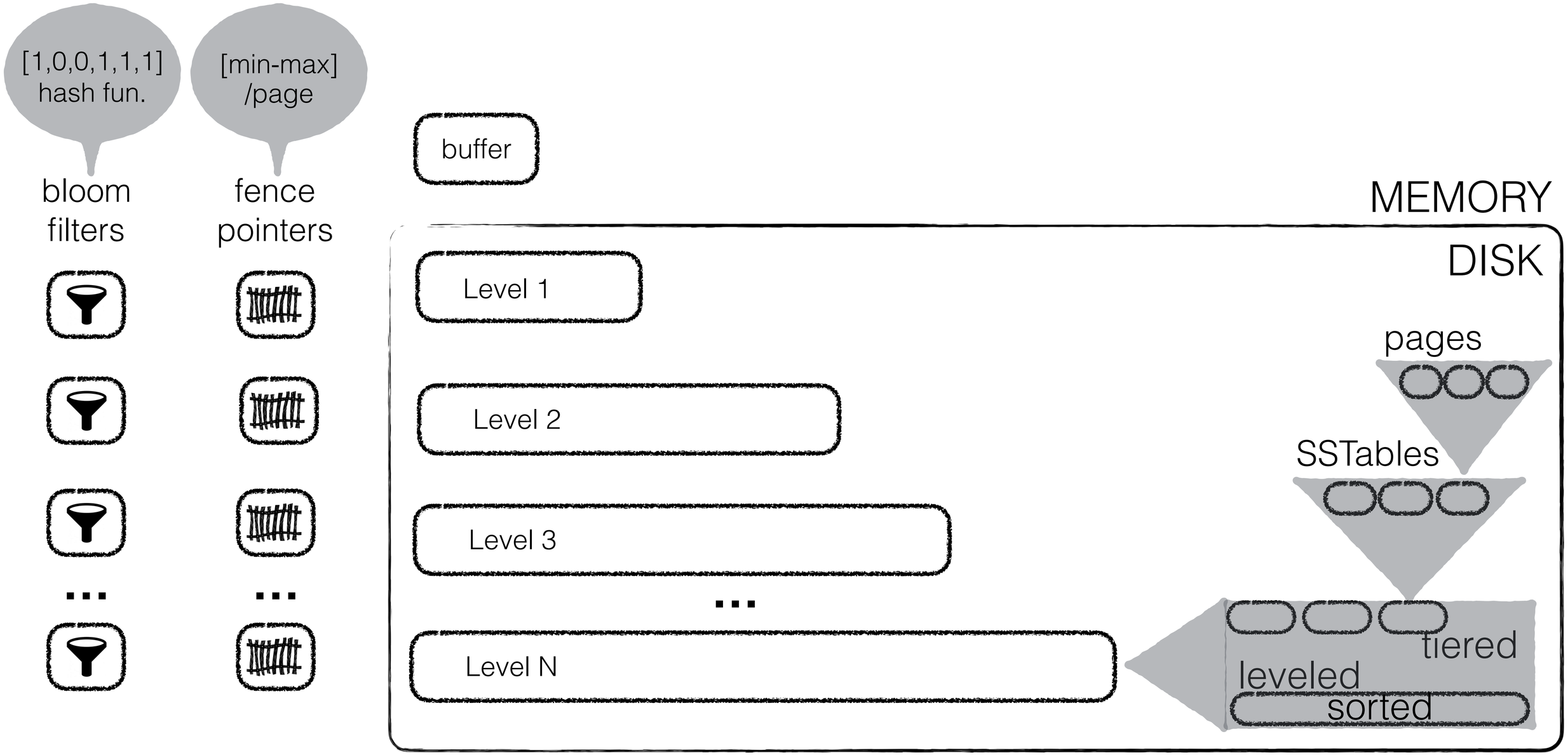


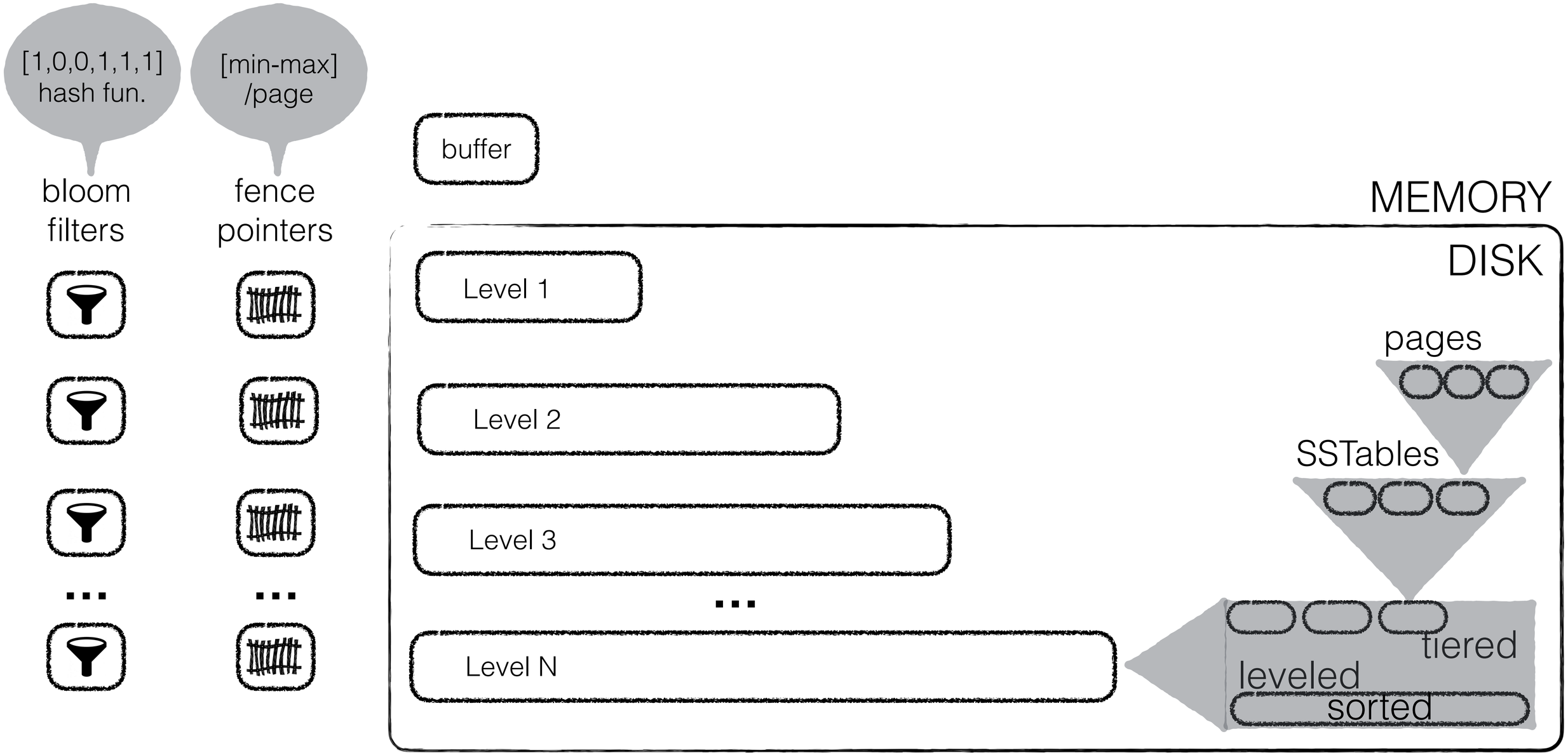
DASlab
@ Harvard SEAS

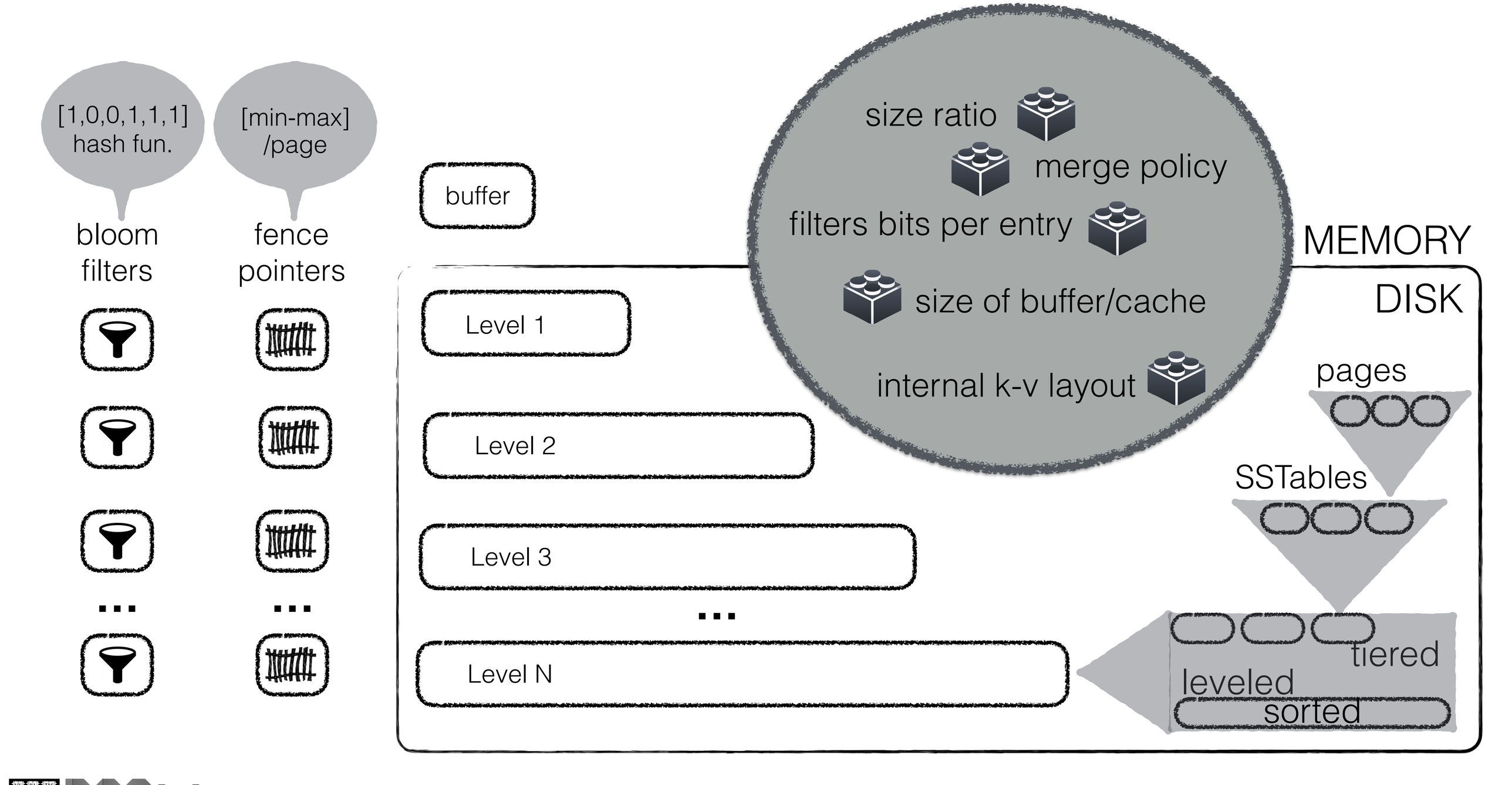


DASlab
@ Harvard SEAS

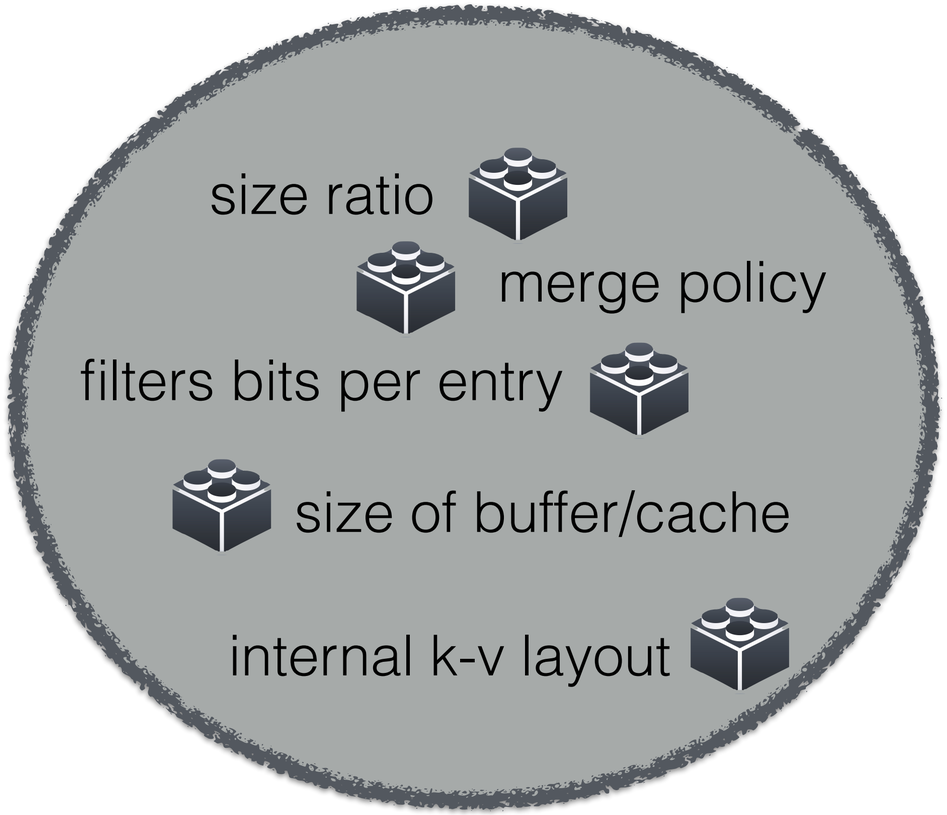

get (key)
[1,0,0,1,1,1]
hash fun.
[min-max]
/page
buffer
bloom
filters
fence
pointers
MEMORY
DISK
Level 1
Level 2
Level 3
...
Level N
pages
SSTables
tiered
leveled
sorted


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

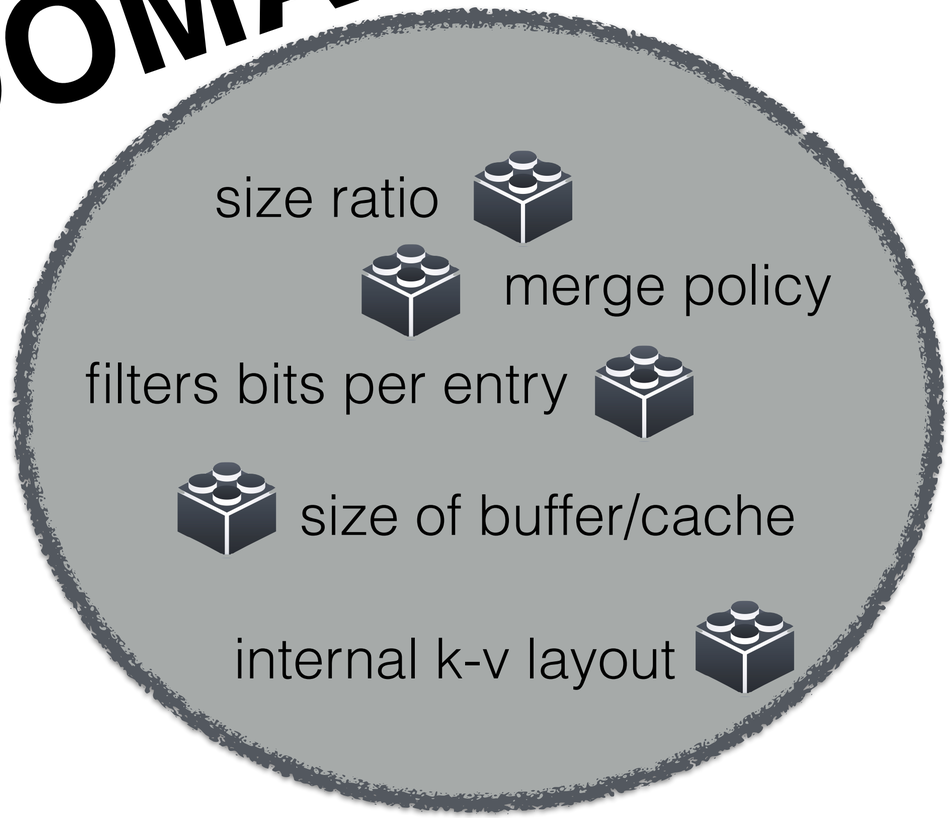

[1,0,0,1,1,1]
hash fun.
[min-max]
/page
bloom
filters
fence
pointers
buffer
MEMORY
DISK
Level 1
Level 2
Level 3
...
Level N
pages
SSTables
tiered
leveled
sorted



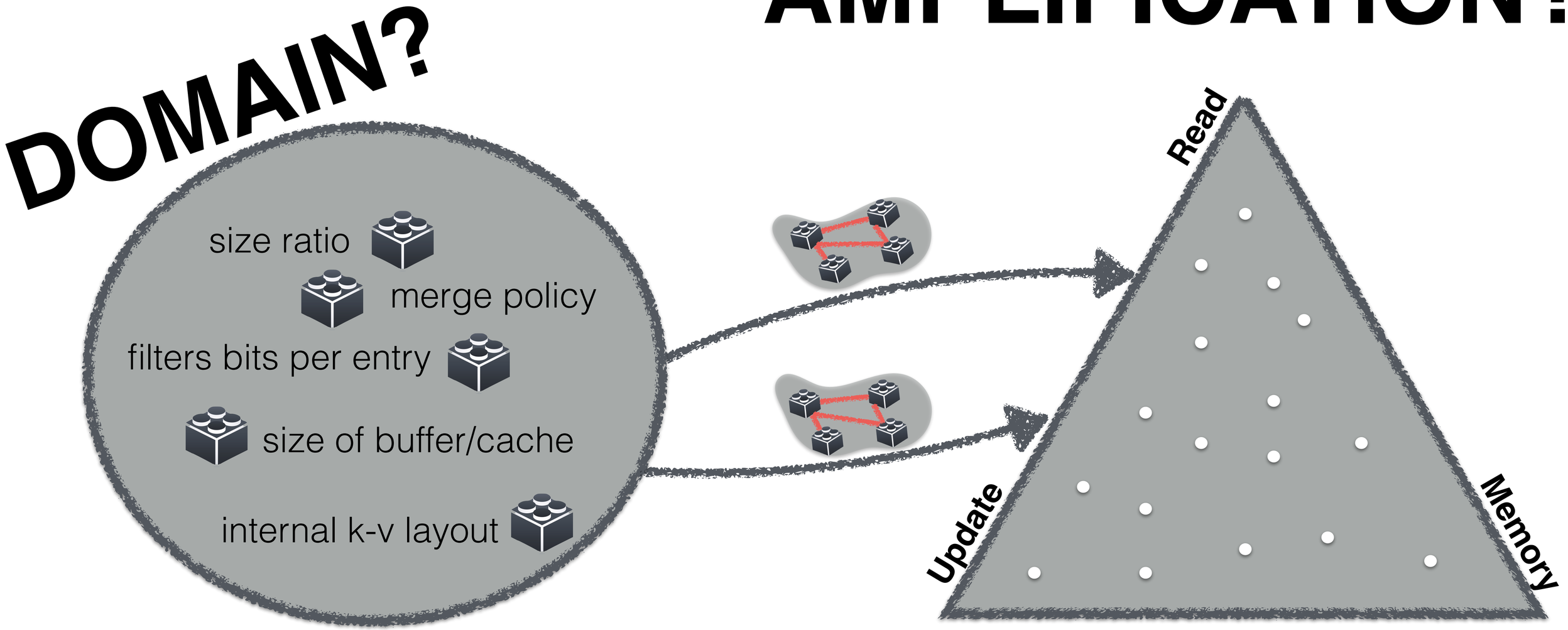
DASlab
@ Harvard SEAS


size ratio
merge policy
filters bits per entry
size of buffer/cache
internal k-v layout

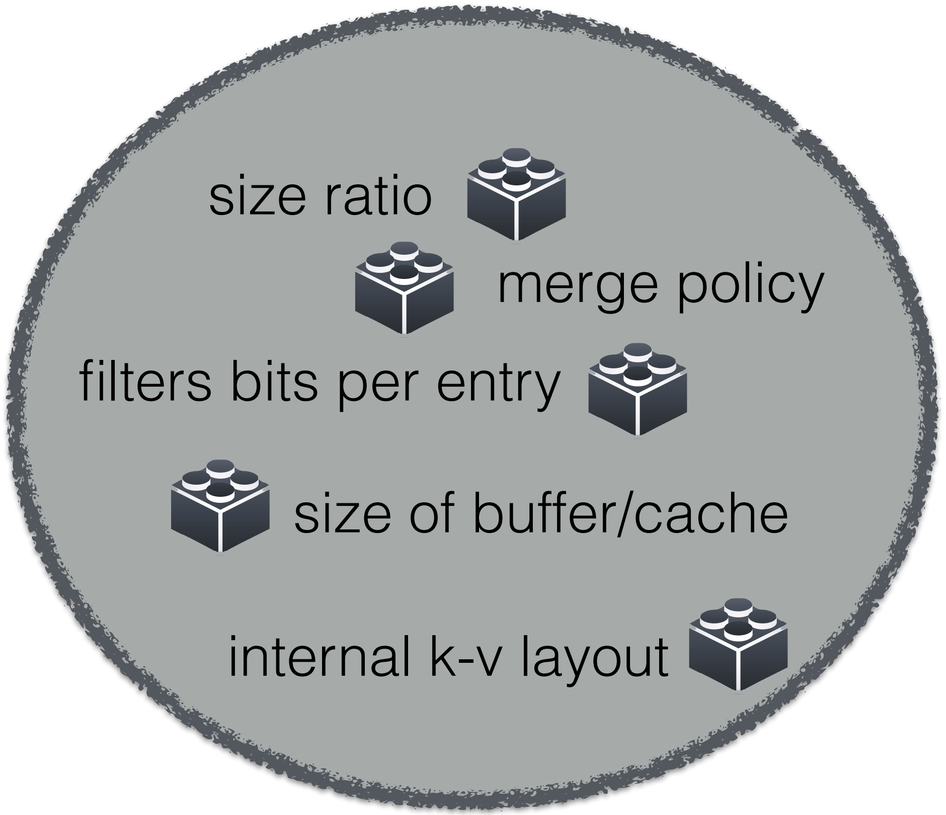
DOMAIN?


size ratio
merge policy
filters bits per entry
size of buffer/cache
internal k-v layout

AMPLIFICATION?



LSM-trees


size ratio
merge policy
filters bits per entry
size of buffer/cache
internal k-v layout

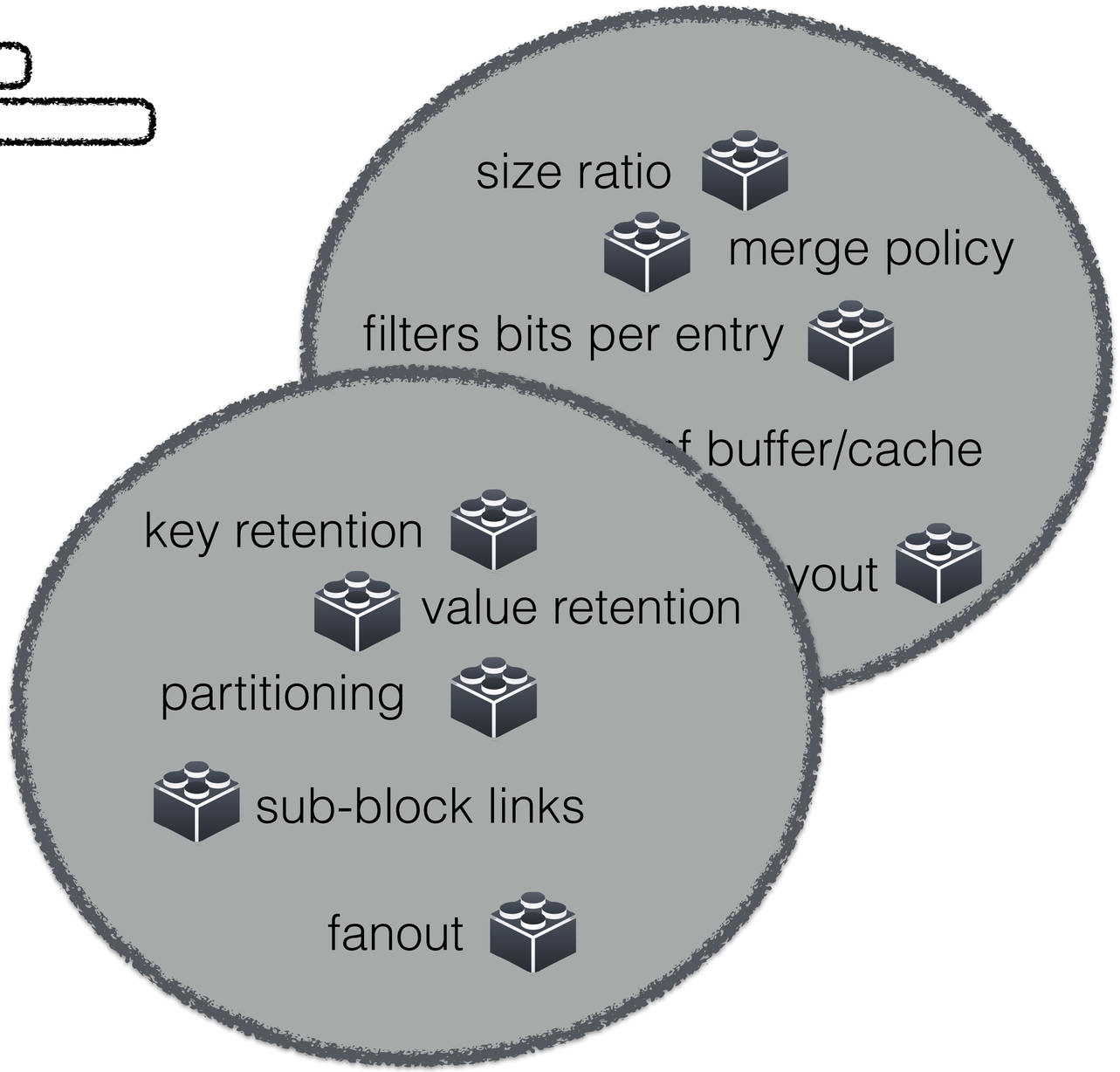
LSM-trees

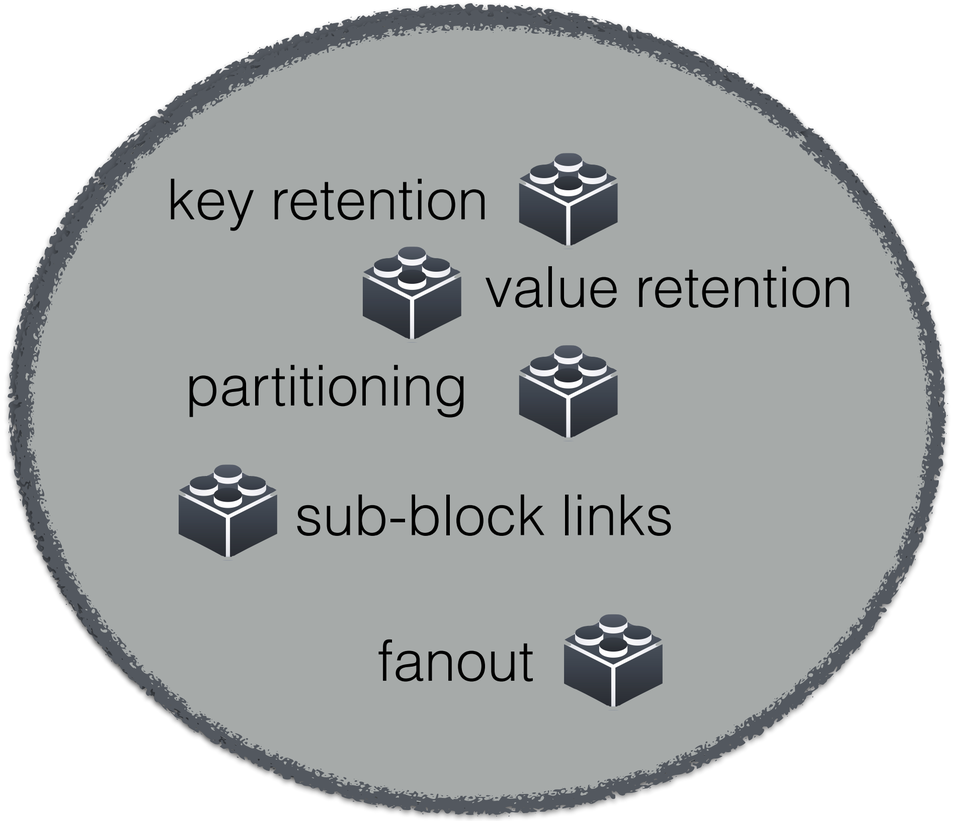
B-trees

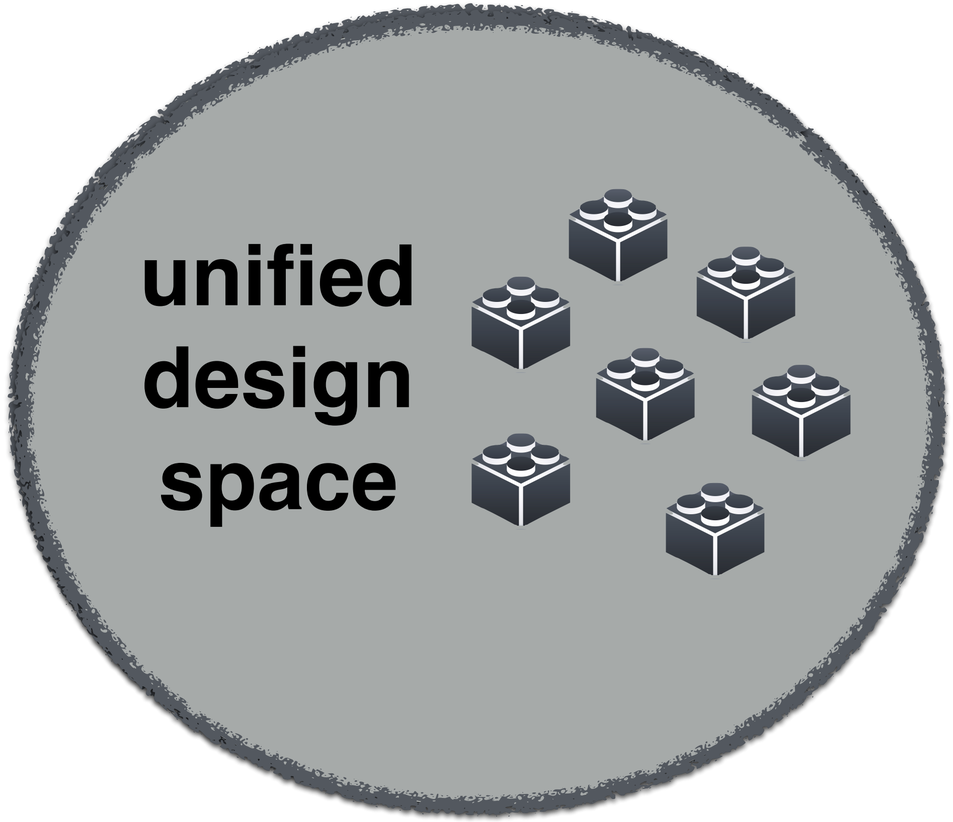
Logs

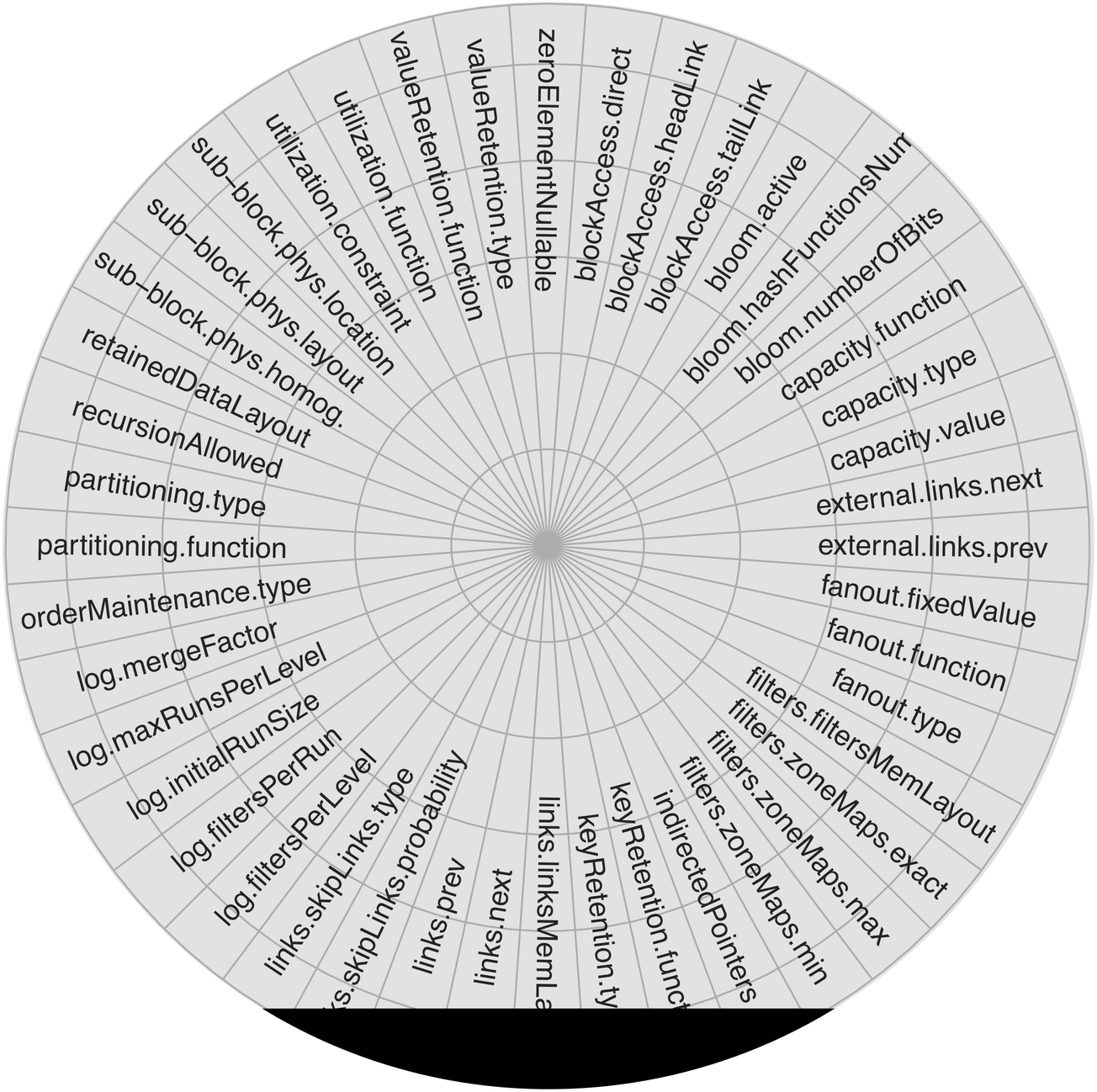
Arrays

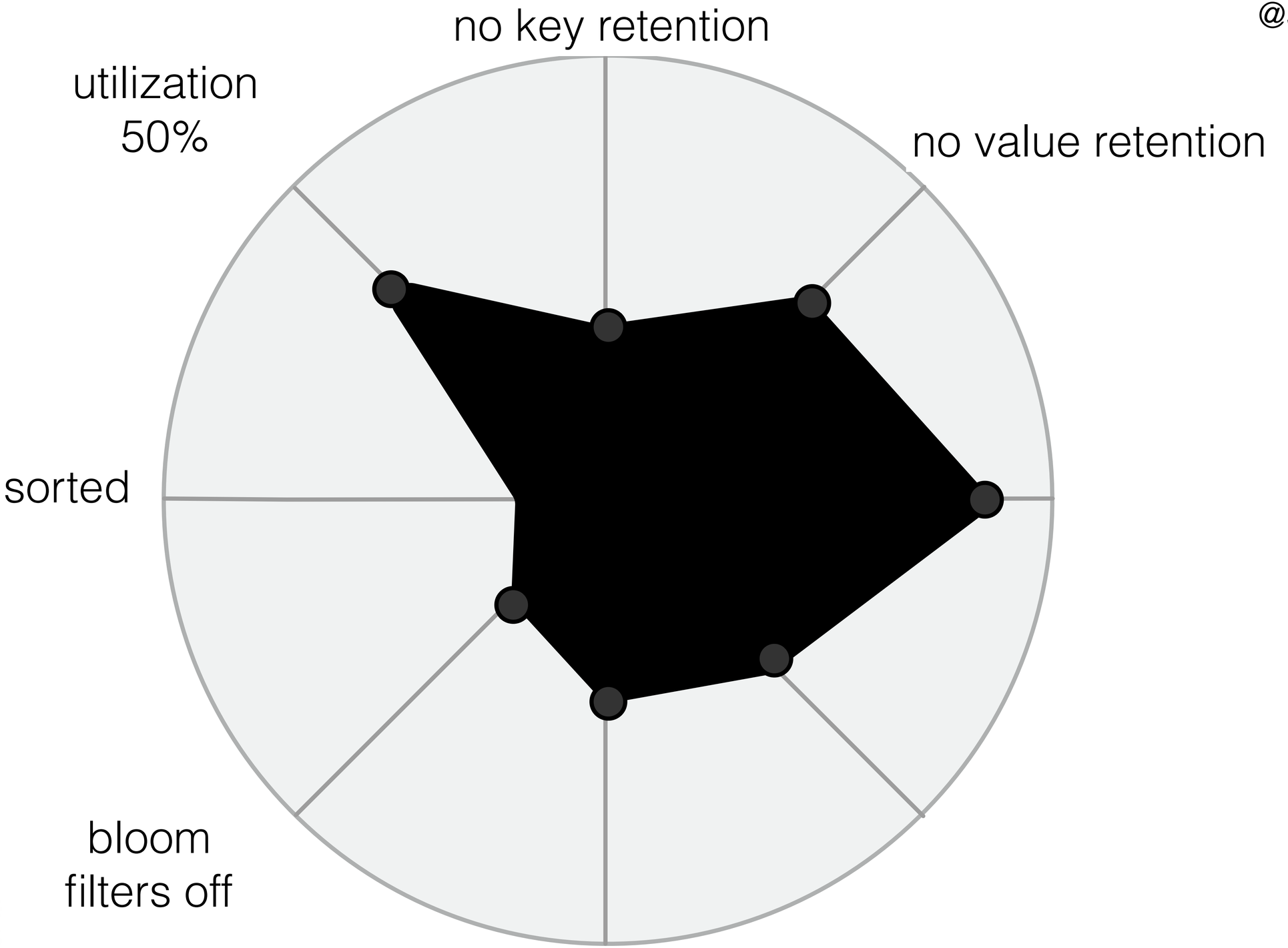
Bitmaps

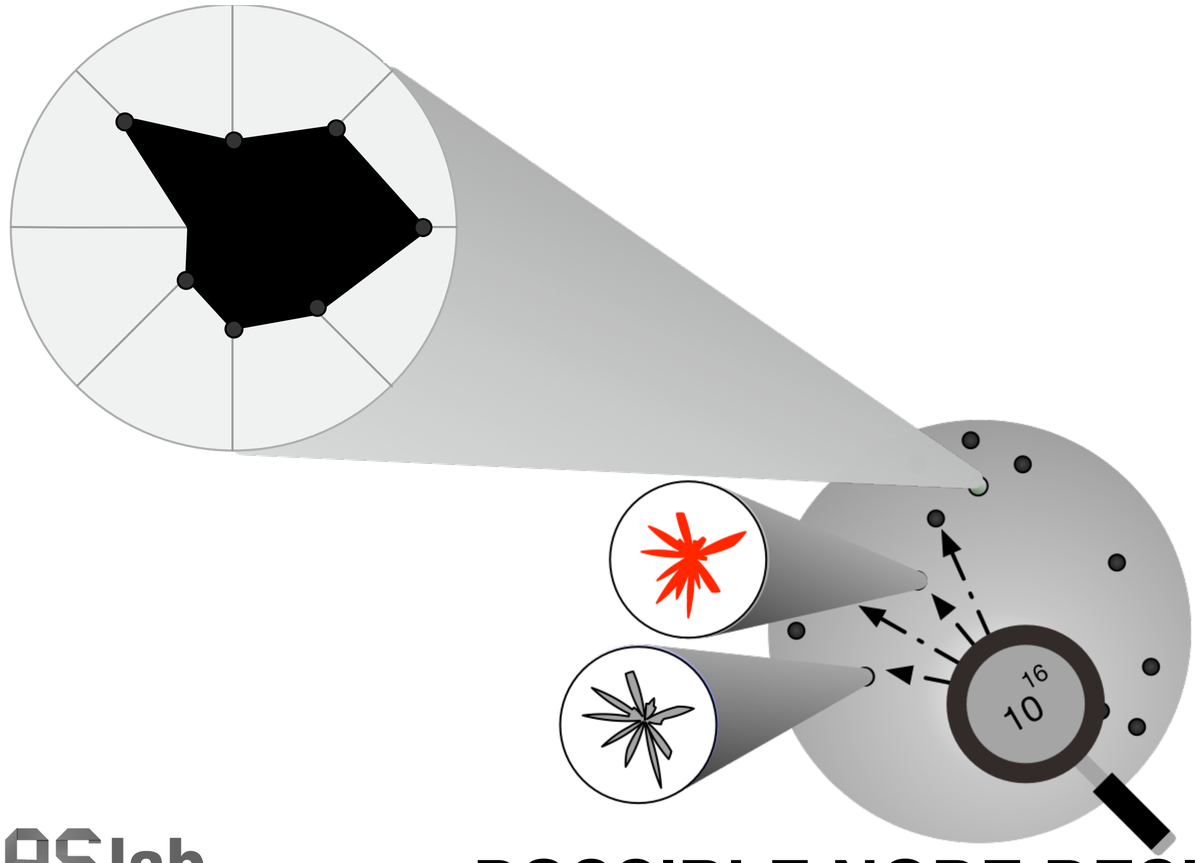



key retention
value retention
partitioning
sub-block links
fanout

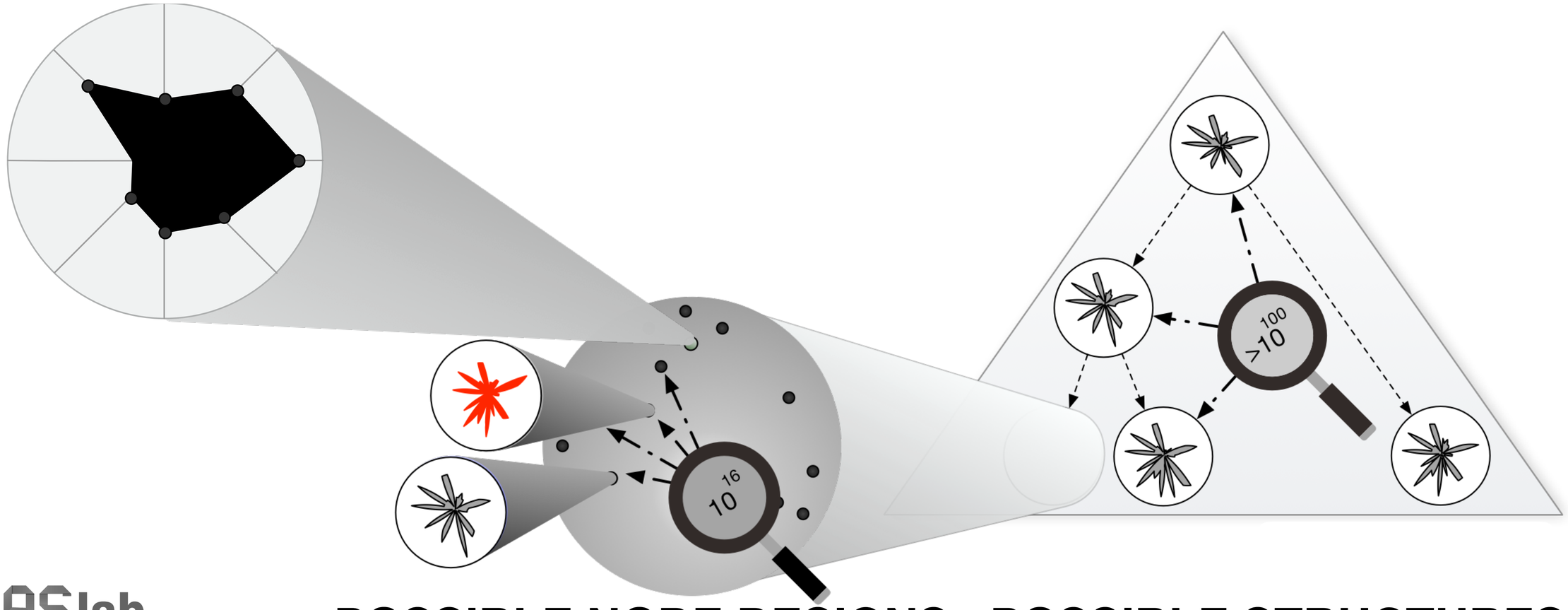

unified
design
space


zeroElementNullable
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

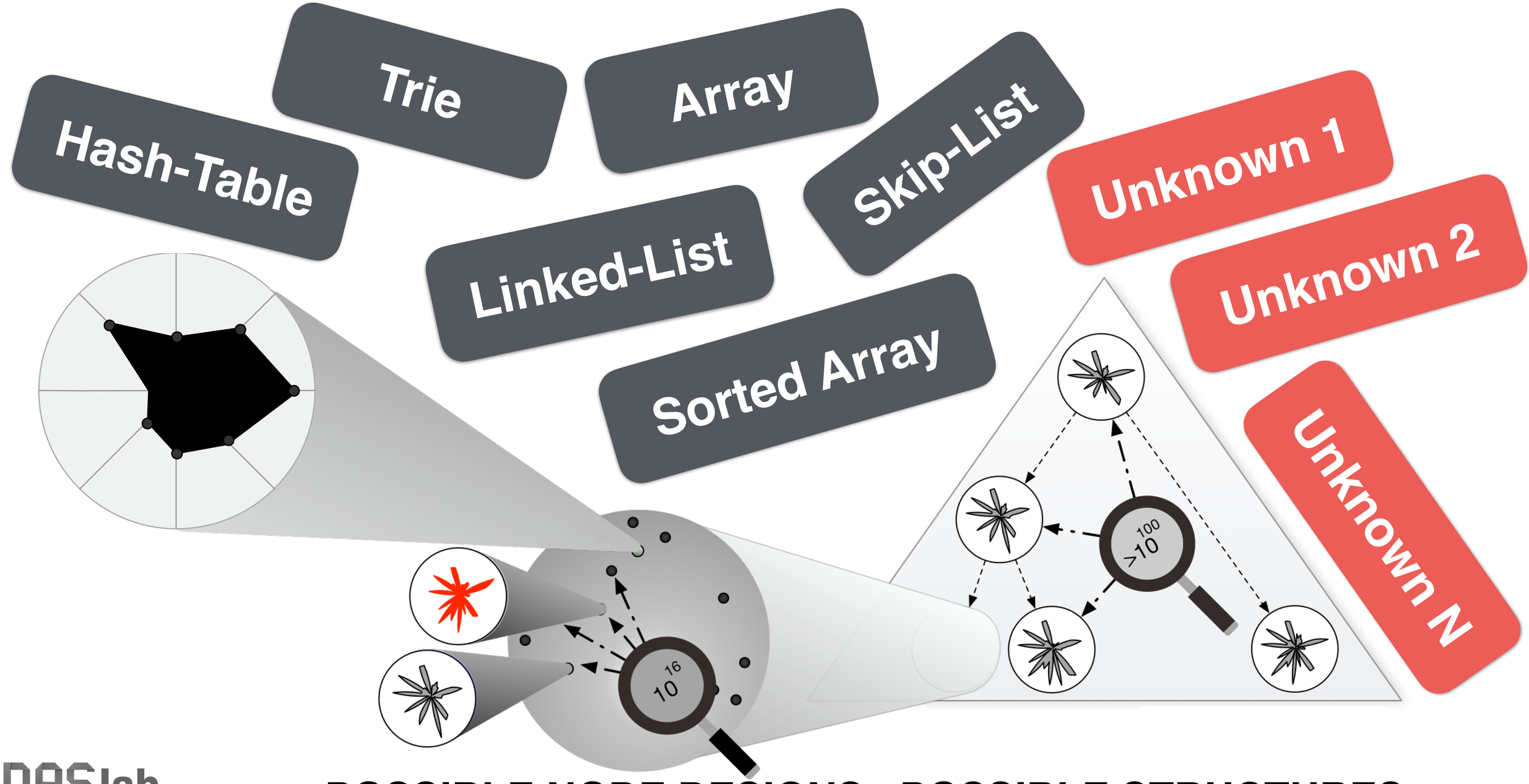

no key retention
no value retention
sorted
bloom
filters off
utilization
50%



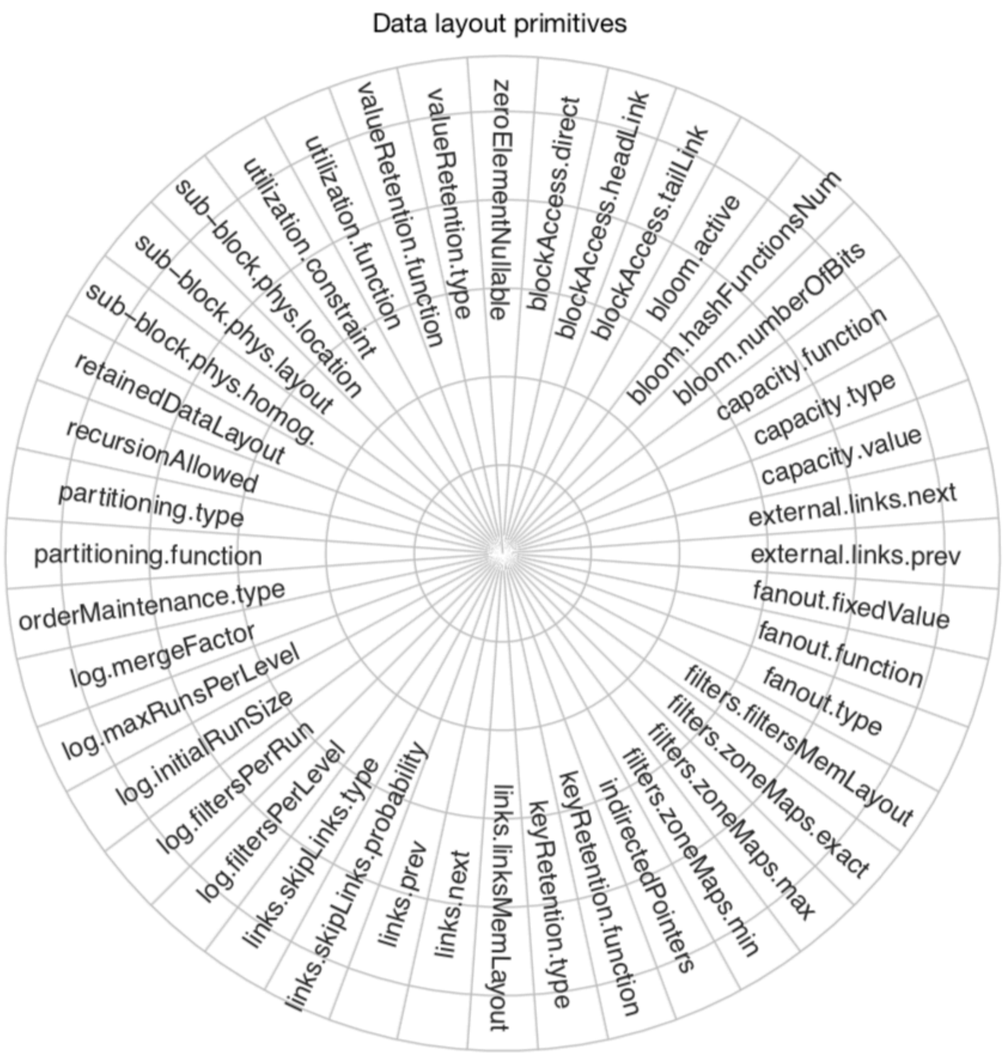
POSSIBLE NODE DESIGNS

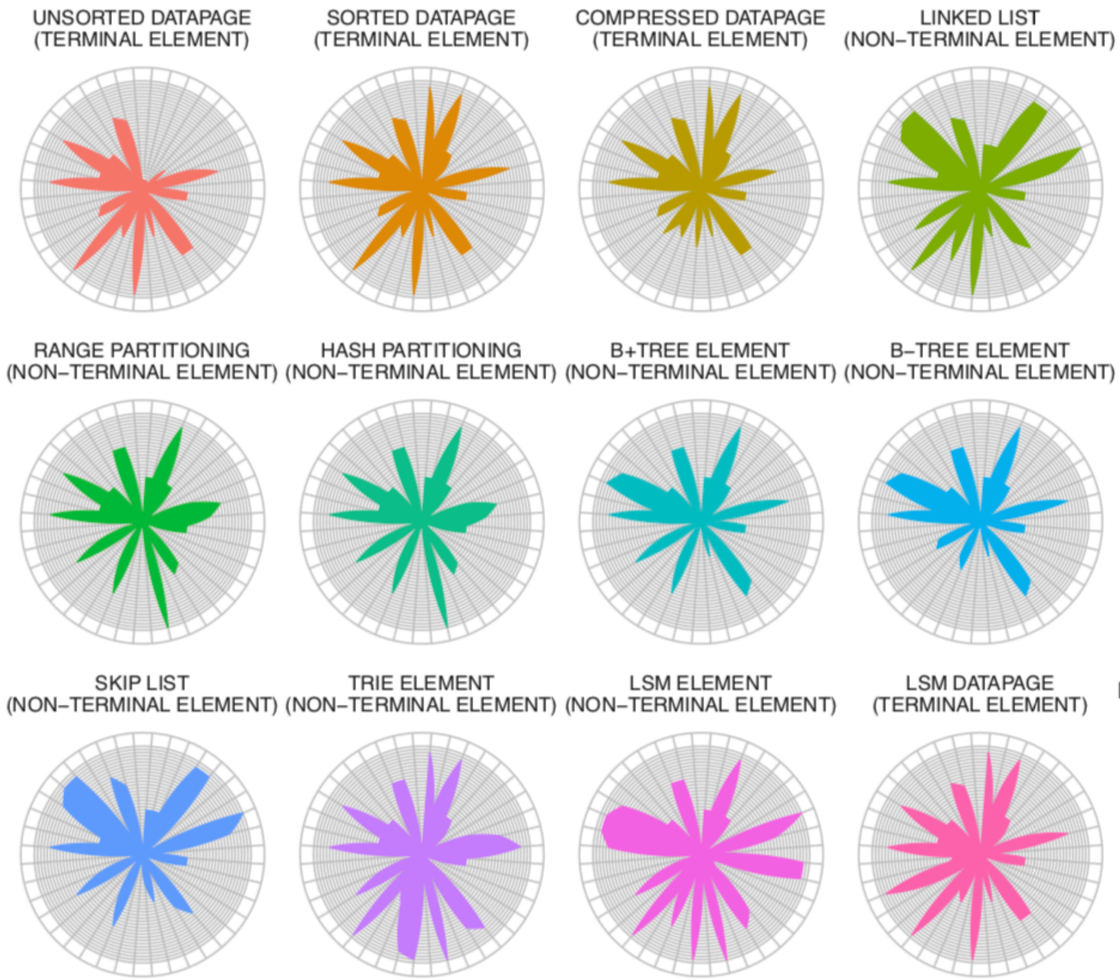


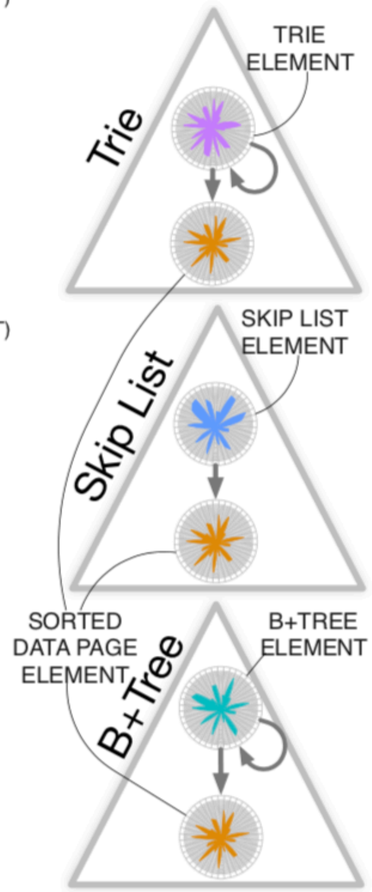
POSSIBLE NODE DESIGNS **POSSIBLE STRUCTURES**



POSSIBLE NODE DESIGNS POSSIBLE STRUCTURES


Data layout primitives
zeroElementNullable
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.function
keyRetention.type
links.linksMemLayout
links.next
links.prev
links.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

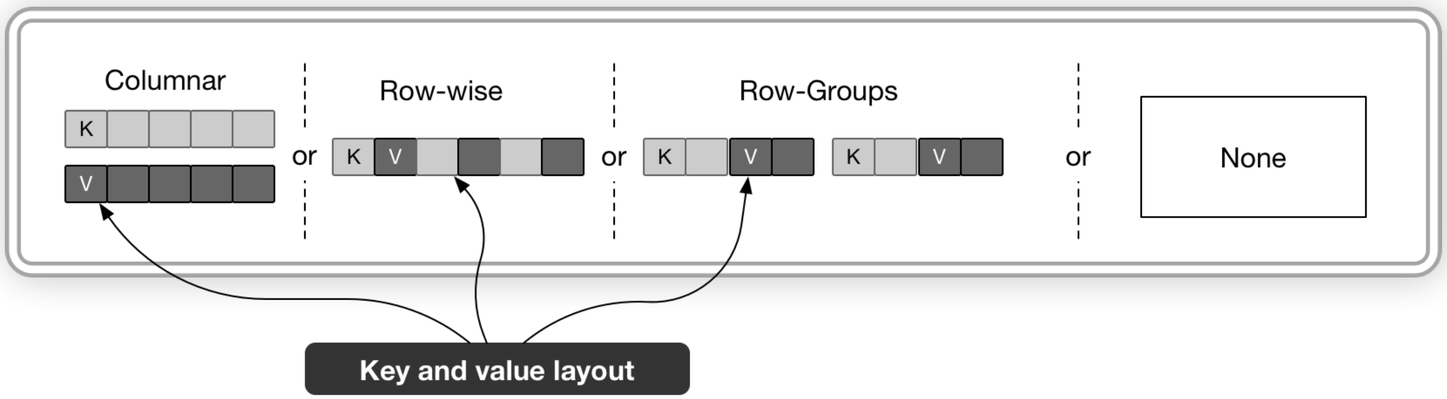

UNSORTED DATAPAGE (TERMINAL ELEMENT)
SORTED DATAPAGE (TERMINAL ELEMENT)
COMPRESSED DATAPAGE (TERMINAL ELEMENT)
LINKED LIST (NON-TERMINAL ELEMENT)
RANGE PARTITIONING (NON-TERMINAL ELEMENT)
HASH PARTITIONING (NON-TERMINAL ELEMENT)
B+TREE ELEMENT (NON-TERMINAL ELEMENT)
B-TREE ELEMENT (NON-TERMINAL ELEMENT)
SKIP LIST (NON-TERMINAL ELEMENT)
TRIE ELEMENT (NON-TERMINAL ELEMENT)
LSM ELEMENT (NON-TERMINAL ELEMENT)
LSM DATAPAGE (TERMINAL ELEMENT)


Trie
TRIE ELEMENT
Skip List
SKIP LIST ELEMENT
B+Tree
SORTED DATA PAGE ELEMENT
B+TREE ELEMENT

Are keys retained? (yes, no, function)

Are values retained?

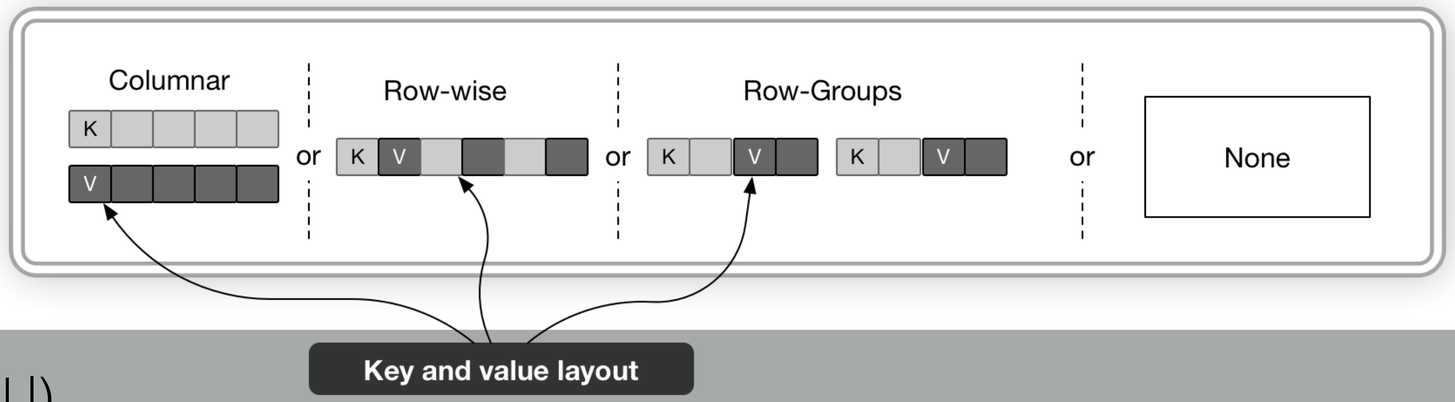
Utilization? (e.g., >50%)



Are keys retained? (yes, no, function)

Are values retained?

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



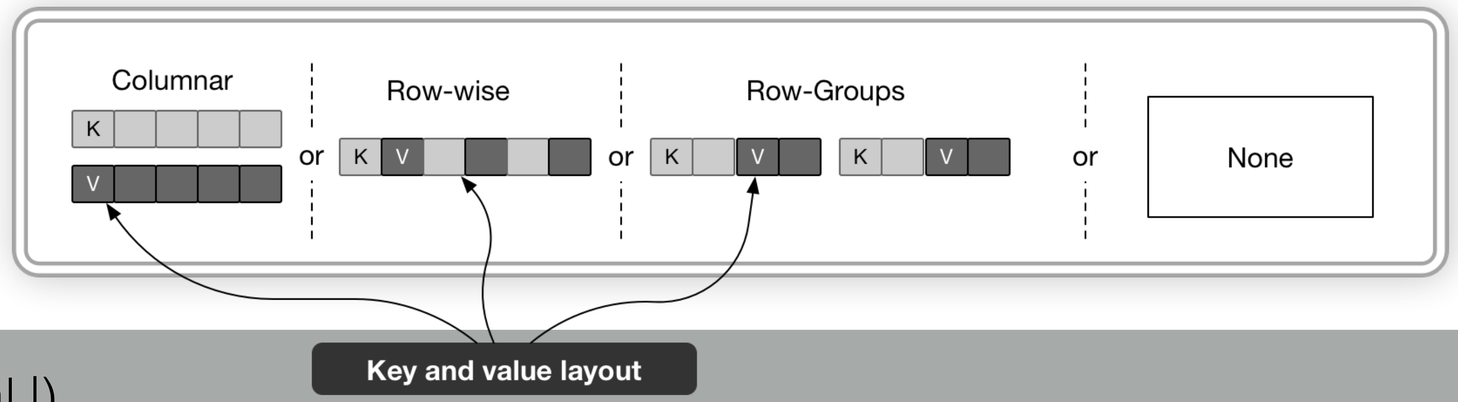
Fanout (fixed/functional | unlimited | terminal |)

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 |)

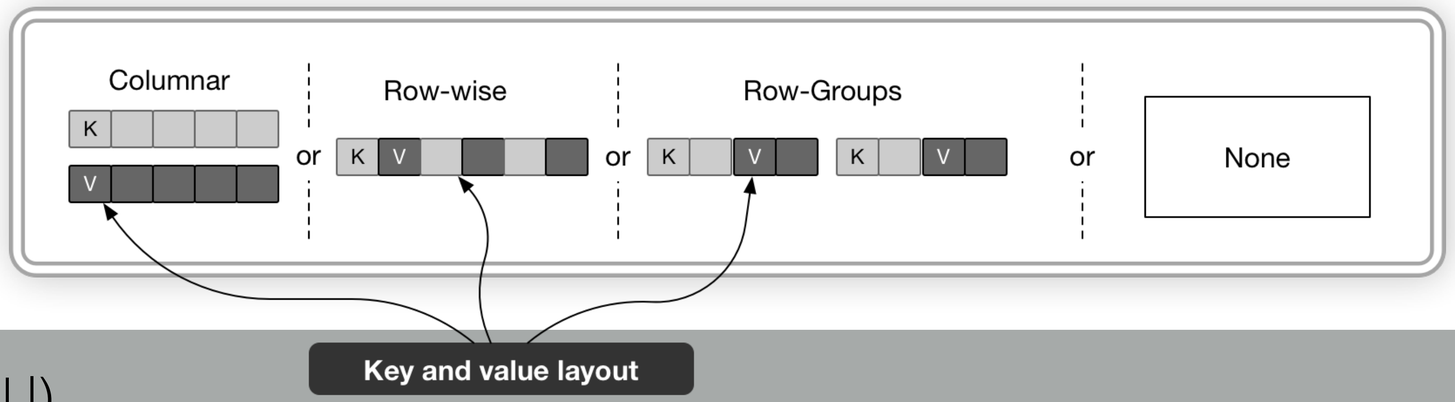
Key partitioning (none(fw-append | bw-append) | sorted | range() | radix() | function (func) | temporal(…))

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

Are keys retained? (yes, no, function)

Are values retained?

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



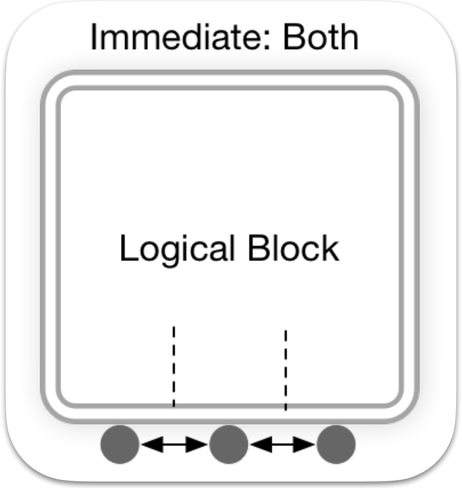
Fanout (fixed/functional | unlimited | terminal |)

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)

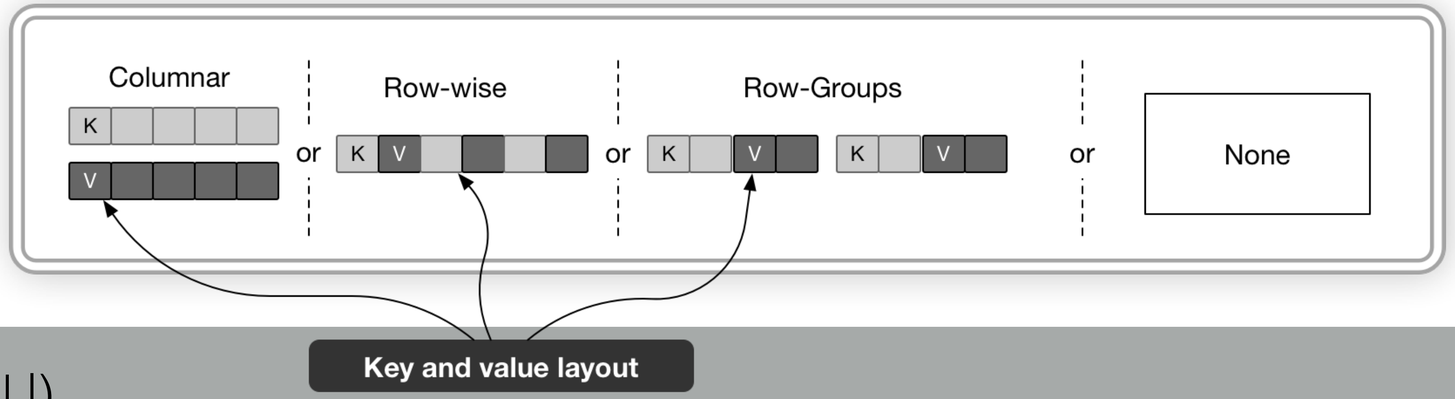
Sub block skip links (perfect | randomized(prob: double) | function(func) | none)



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)

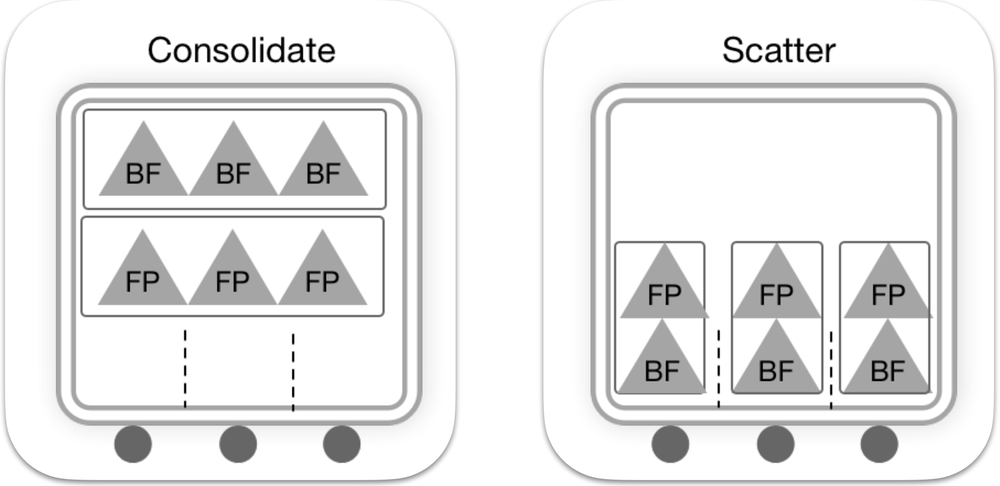
Sub block skip links (perfect | randomized(prob: double) | function(func) | none)

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

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

Filters layout (consolidate | scatter)

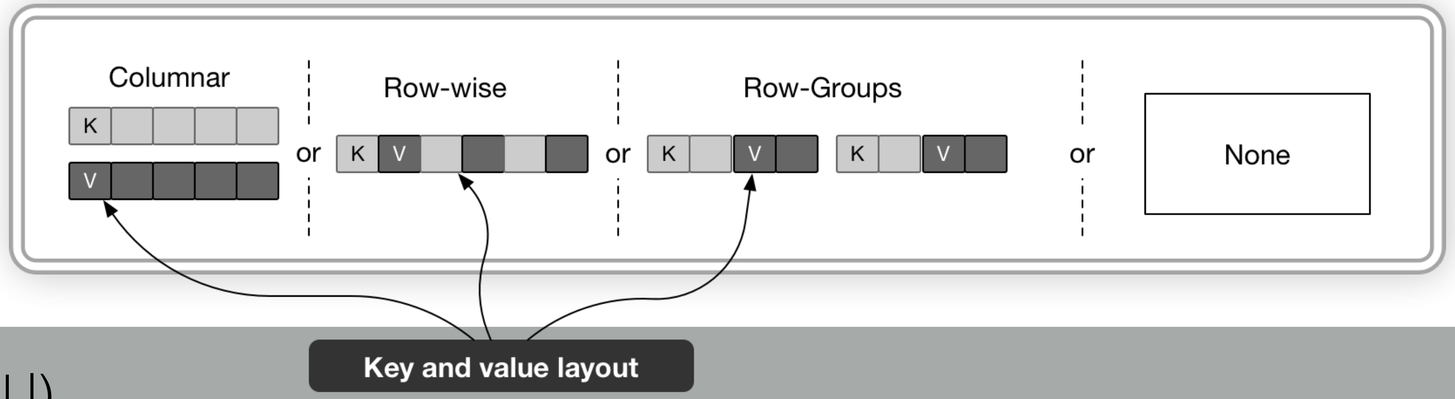
Links layout (consolidate | scatter)



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)

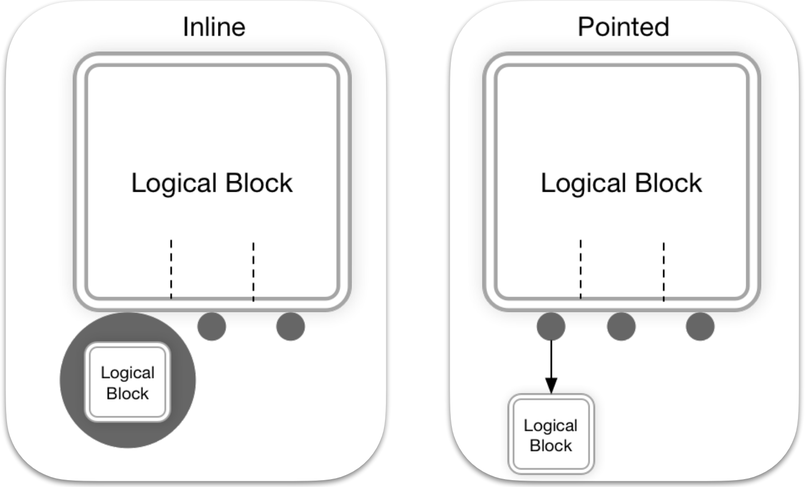
Bloom filters (off | on(num_hashes: int, num_bits: int))

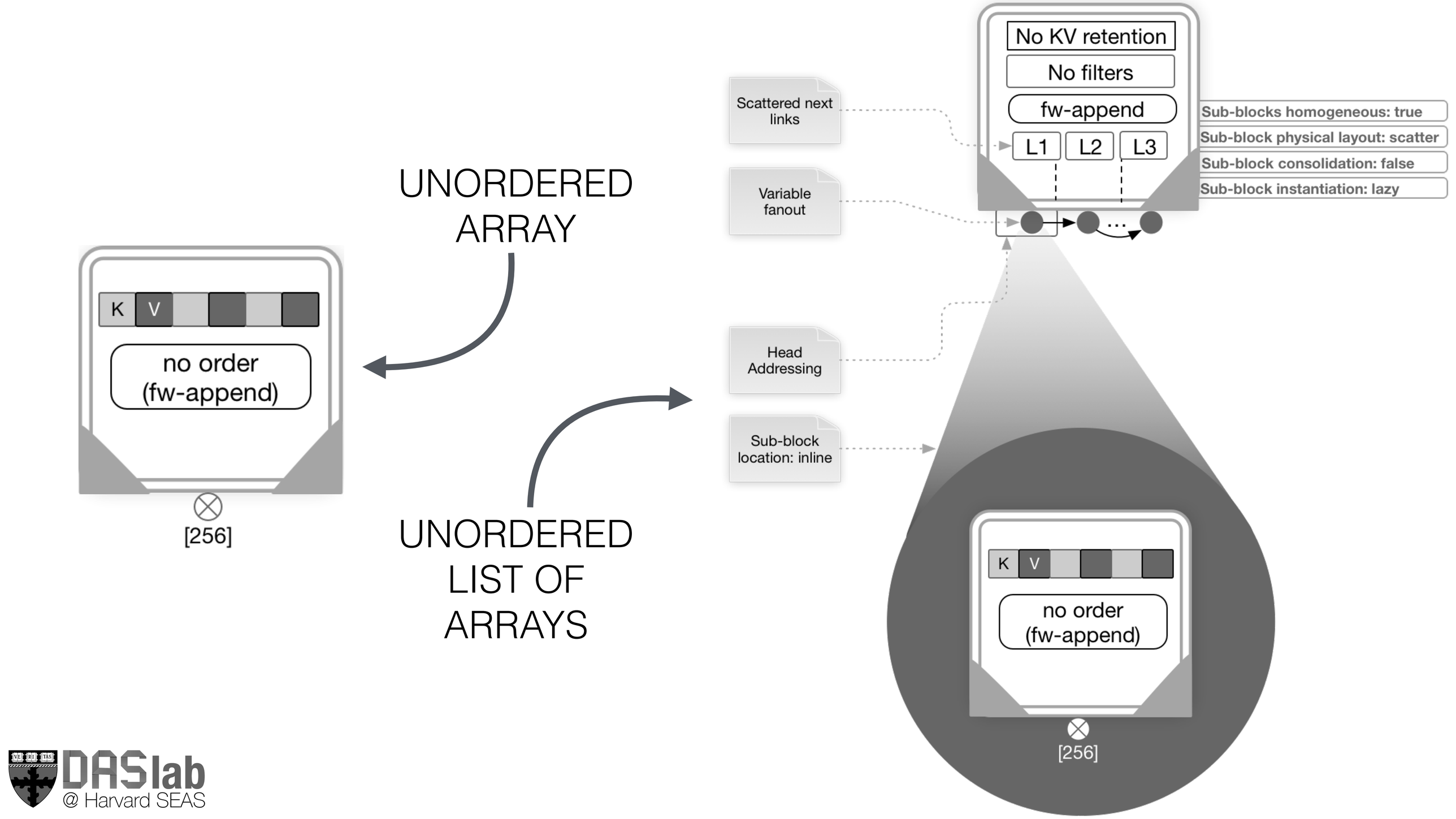
Filters layout (consolidate | scatter)

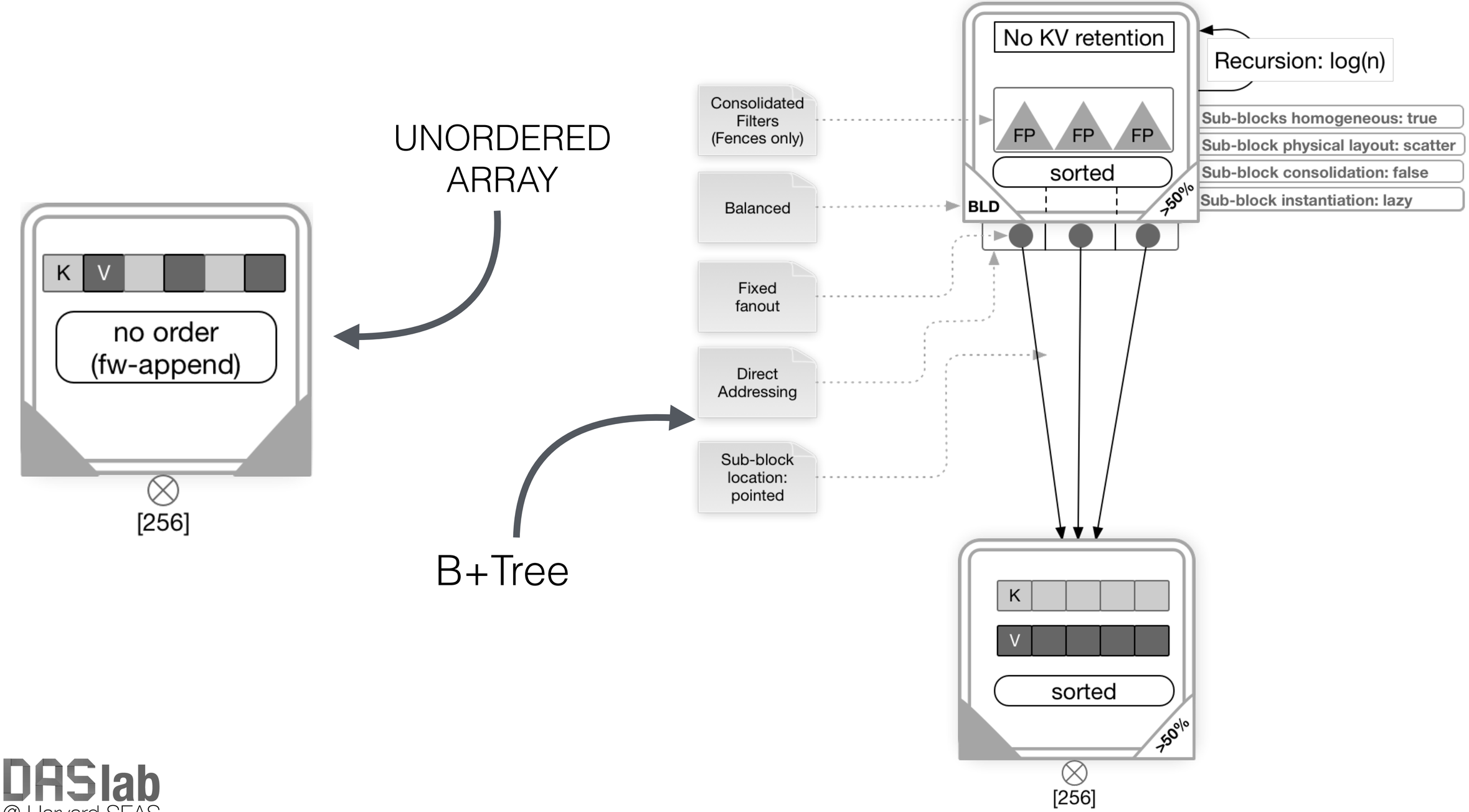
Links layout (consolidate | scatter)

Physical location (inline | pointed | double- pointed)

Physical layout (BFS | scatter)




UNORDERED ARRAY
K
V
no order (fw-append)
[256]
UNORDERED LIST OF ARRAYS
Scattered next links
Variable fanout
Head Addressing
Sub-block location: inline
No KV retention
No filters
fw-append
L1
L2
L3
Sub-blocks homogeneous: true
Sub-block physical layout: scatter
Sub-block consolidation: false
Sub-block instantiation: lazy
K
V
no order (fw-append)
[256]
DASlab
@ Harvard SEAS


UNORDERED ARRAY
K
V
no order (fw-append)
[256]
B+Tree
Consolidated Filters (Fences only)
Balanced
Fixed fanout
Direct Addressing
Sub-block location: pointed
No KV retention
Recursion: log(n)
FP
FP
FP
sorted
BLD
>50%
Sub-blocks homogeneous: true
Sub-block physical layout: scatter
Sub-block consolidation: false
Sub-block instantiation: lazy
K
V
sorted
>50%
[256]

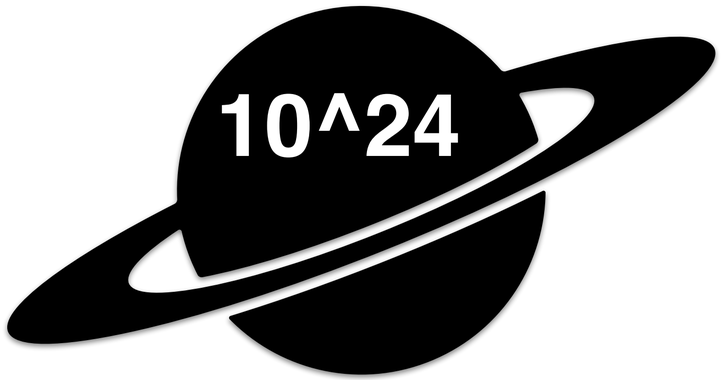
Primitives and Instances

Categories

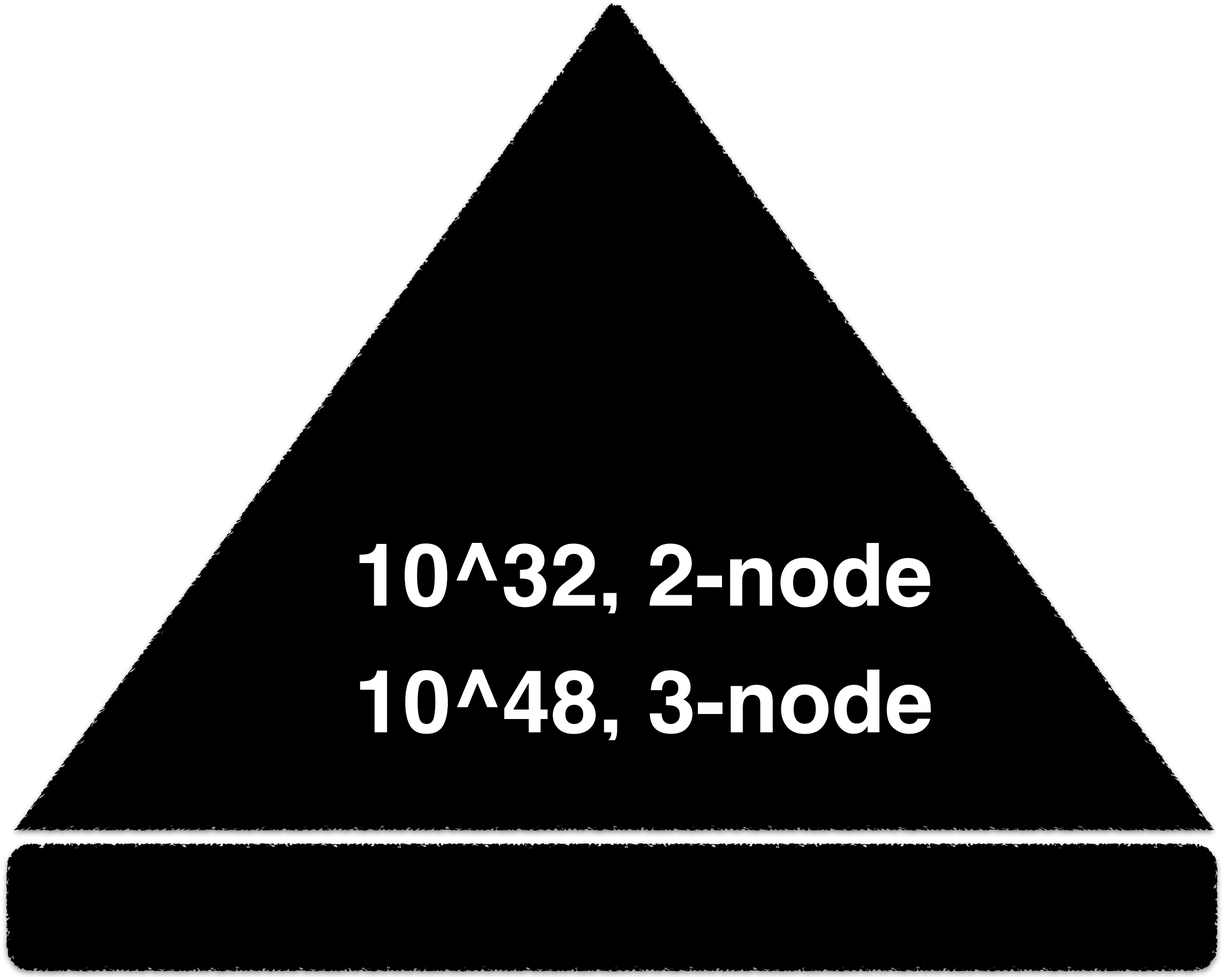
Unless otherwise specified, we use a reduced default values domain of 100 values for integers, 10 values for doubles, and 1 value for functions.

		Primitive	Domain	size	Hash Table			B+Tree/CSB+Tree/FAST			
						LPL					
					H	LL	UDP	B+	CSB+	FAST	ODP
Node organization	1	**Key retention.** *No:* node contains no real key data, e.g., intermediate nodes of b+trees and linked lists. *Yes:* contains complete key data, e.g., nodes of b-trees, and arrays. *Function:* contains only a subset of the key, i.e., as in tries.	yes | no | function(func)	3	no	no	yes	no	no	no	yes
Node organization	2	**Value retention.** *No:* node contains no real value data, e.g., intermediate nodes of b+trees, and linked lists. *Yes:* contains complete value data, e.g., nodes of b-trees, and arrays. *Function:* contains only a subset of the values.	yes | no | function(func)	3	no	no	yes	no	no	no	yes
Node organization	3	**Key order.** Determines the order of keys in a node or the order of fences if real keys are not retained.	none | sorted | k-ary (k: int)	12	none	none	none	sorted	sorted	4-ary	sorted
Node organization	4	**Key-value layout.** Determines the physical layout of key-value pairs. Rules: requires key retention != no or value retention != no.	row-wise | columnar | col-row-groups(size: int)	12			col.				col.
Node organization	5	**Intra-node access.** Determines how sub-blocks (one or more keys of this node) can be addressed and retrieved within a node, e.g., with direct links, a link only to the first or last block, etc.	direct | head_link | tail_link | link_function(func)	4	direct	head	direct	direct	direct	direct	direct
Node organization	6	**Utilization.** Utilization constraints in regards to capacity. For example, >= 50% denotes that utilization has to be greater than or equal to half the capacity.	= (X%) | function(func) | none *(we currently only consider X=50)*	3	none	none	none	>= 50%	>= 50%	>= 50%	none
Node filters	7	**Bloom filters.** A node's sub-block can be filtered using bloom filters. Bloom filters get as parameters the number of hash functions and number of bits.	off | on(num_hashes: int, num_bits: int) *(up to 10 num_hashes considered)*	1001	off	off	off	off	off	off	off
Node filters	8	**Zone map filters.** A node's sub-block can be filitered using zone maps, e.g., they can filter based on mix/max keys in each sub-block.	min | max | both | exact | off	5	off	off	off	min	min	min	off
Node filters	9	**Filters memory layout.** Filters are stored contiguously in a single area of the node or scattered across the sub-blocks. Rules: requires bloom filter != off or zone map filters != off.	consolidate | scatter	2				scatter	scatter	scatter	
	10	**Fanout/Radix.** Fanout of current node in terms of sub-blocks. This can either be unlimited (i.e., no restriction on the number of sub-blocks), fixed to a number, decided by a function or the node is terminal and thus has a fixed capacity.	fixed(value: int) | function(func) mited | terminal(cap: int) *(up to 10 different capacities and up to 10 fixed fanout values are considered)*	22	fixed(100)	unlimited	term(256)	fixed(20)	fixed(20)	fixed(16)	term(256)
	11	**Key partitioning.** Set if there is a pre-defined key partitioning imposed, e.g., the									

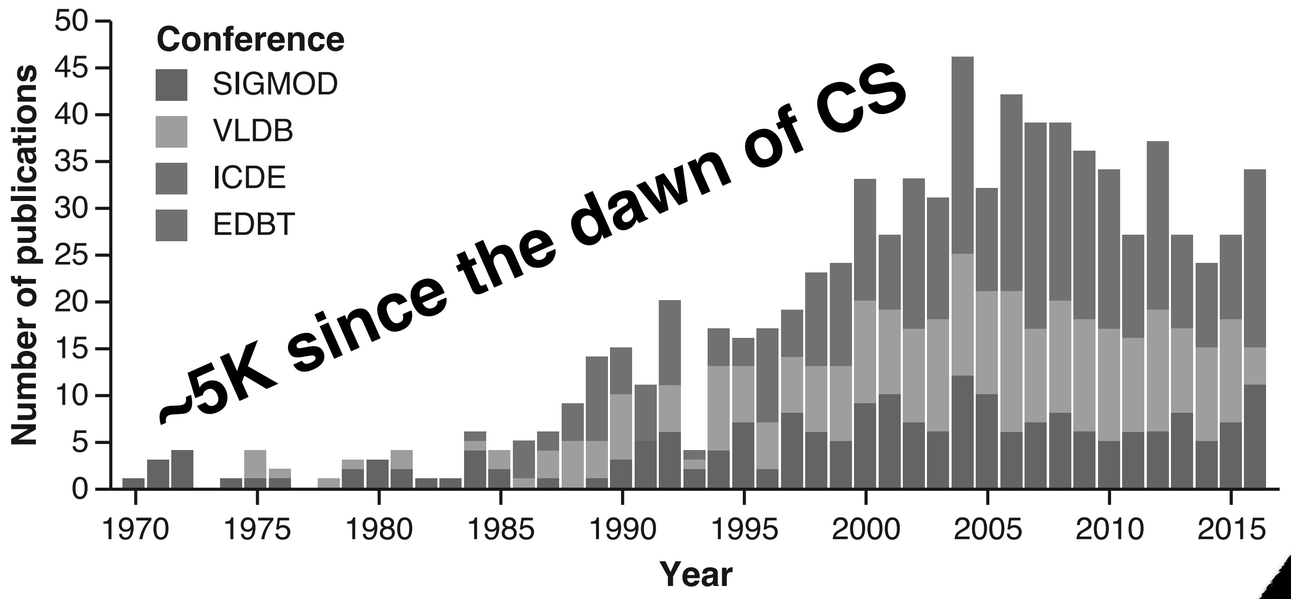
Categories											
Partitio		or balanced (i.e., all sub-blocks have the same size), unrestricted or functional. Rules: requires key partitioning != none.	stricted | function(func) *(up to 10 different fixed capacity values are considered)*	13	unrestri	fixed(25		balance	balance	balance	
	13	**Immediate node links.** Whether and how sub-blocks are connected.	next | previous | both | none	4	none	next	none	none	none	none	none
	14	**Skip node links.** Each sub-block can be connected to another sub-block (not only the next or previous) with skip-links. They can be perfect, randomized or custom.	perfect | randomized(prob: double) | function(func) | none	13	none	none	none	none	none	none	none
	15	**Area-links.** Each sub-tree can be connected with another sub-tree at the leaf level throu area links. Examples include the linked leaves of a B+Tree.	forward | backward | both | none	4	none	none	forw.	none	none	none	none
Children layout	16	**Sub-block physical location.** This represents the physical location of the sub-blocks. Pointed: in heap, Inline: block physically contained in parent. Double-pointed: in heap but with pointers back to the parent. Rules: requires fanout/radix != terminal.	inline | pointed | double-pointed	3	pointed	inline		pointed	pointed	pointed	
	17	**Sub-block physical layout.** This represents the physical layout of sub-blocks. Scatter: random placement in memory. BFS: laid out in a breadth-first layout. BFS layer list: hierarchical level nesting of BFS layouts. Rules: requires fanout/radix != terminal.	BFS | BFS layer(level-grouping: int) | scatter *(up to 3 different values for layer-grouping are considered)*	5	scatter	scatter		scatter	BFS	BFS-LL	
	18	**Sub-blocks homogeneous.** Set to true if all sub-blocks are of the same type. Rules: requires fanout/radix != terminal.	boolean	2	true	true		true	true	true	
	19	**Sub-block consolidation.** Single children are merged with their parents. Rules: requires fanout/radix != terminal.	boolean	2	false	false		false	false	false	
	20	**Sub-block instantiation.** If it is set to eager, all sub-blocks are initialized, otherwise they are initialized only when data are available (lazy). Rules: requires fanout/radix != terminal.	lazy | eager	2	lazy	lazy		lazy	lazy	lazy	
	21	**Sub-block links layout.** If there exist links, are they all stored in a single array (consolidate) or spread at a per partition level (scatter). Rules: requires immediate node links != none or skip links != none.	consolidate | scatter	2		scatter					
Recursion	22	**Recursion allowed.** If set to yes, sub-blocks will be subsequently inserted into a node of the same type until a maximum depth (expressed as a function) is reached. Then the terminal node type of this data structure will be used. Rules: requires fanout/radix != terminal.	yes(func) | no	3	no	no		yes(logn)	yes(logn)	yes(logn)	

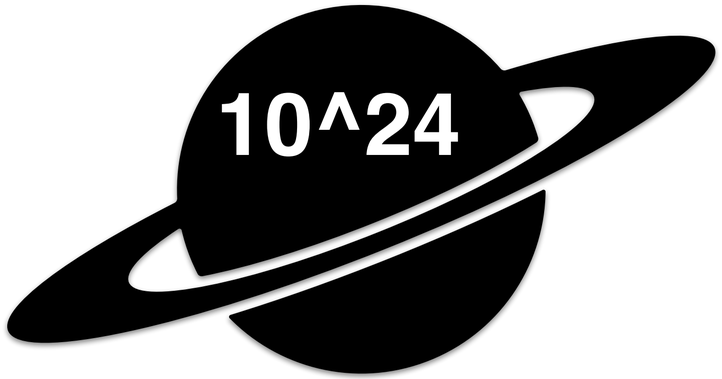


STARS IN THE SKY

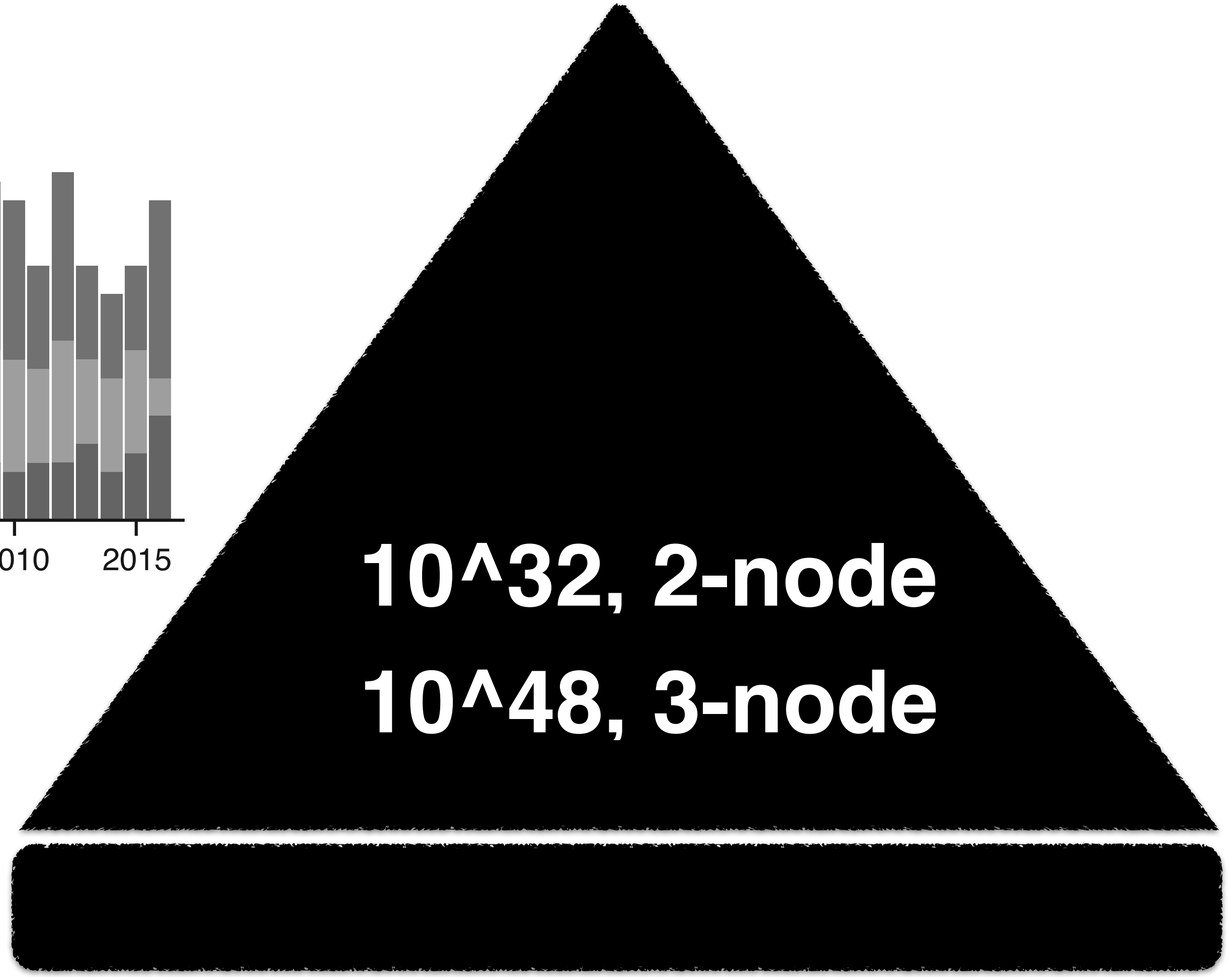


POSSIBLE DATA STRUCTURES

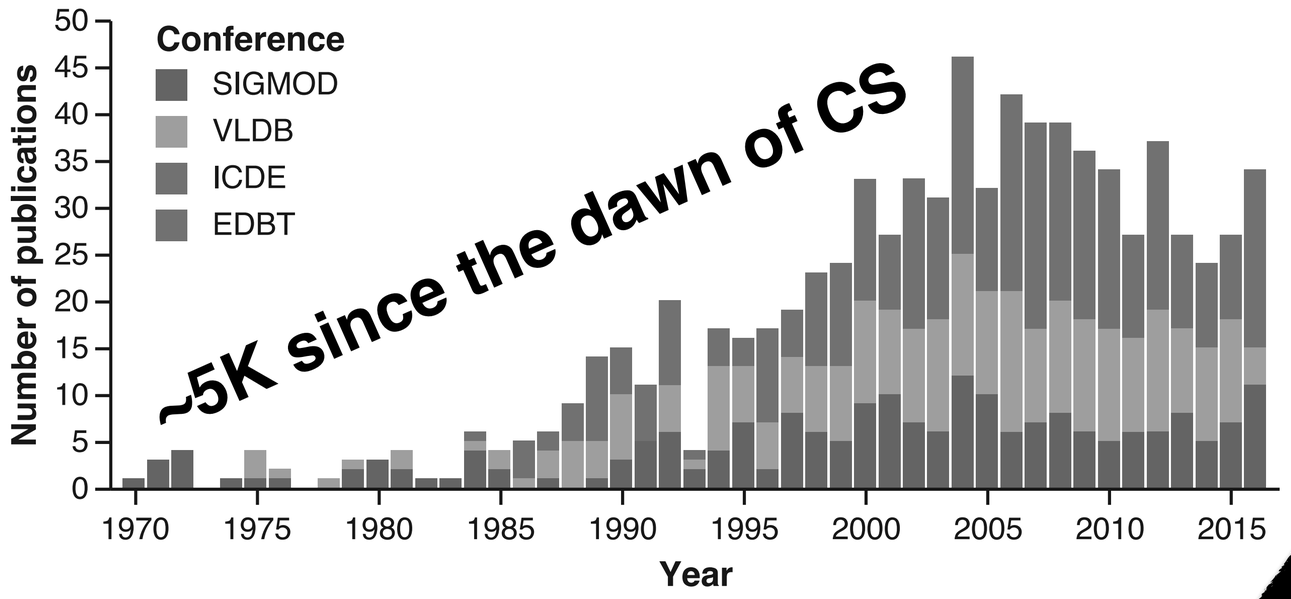


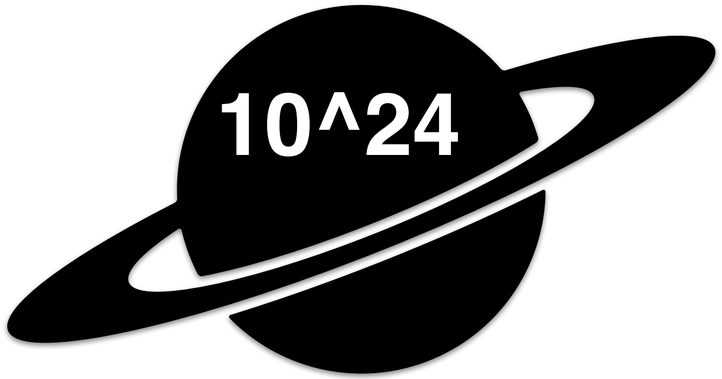


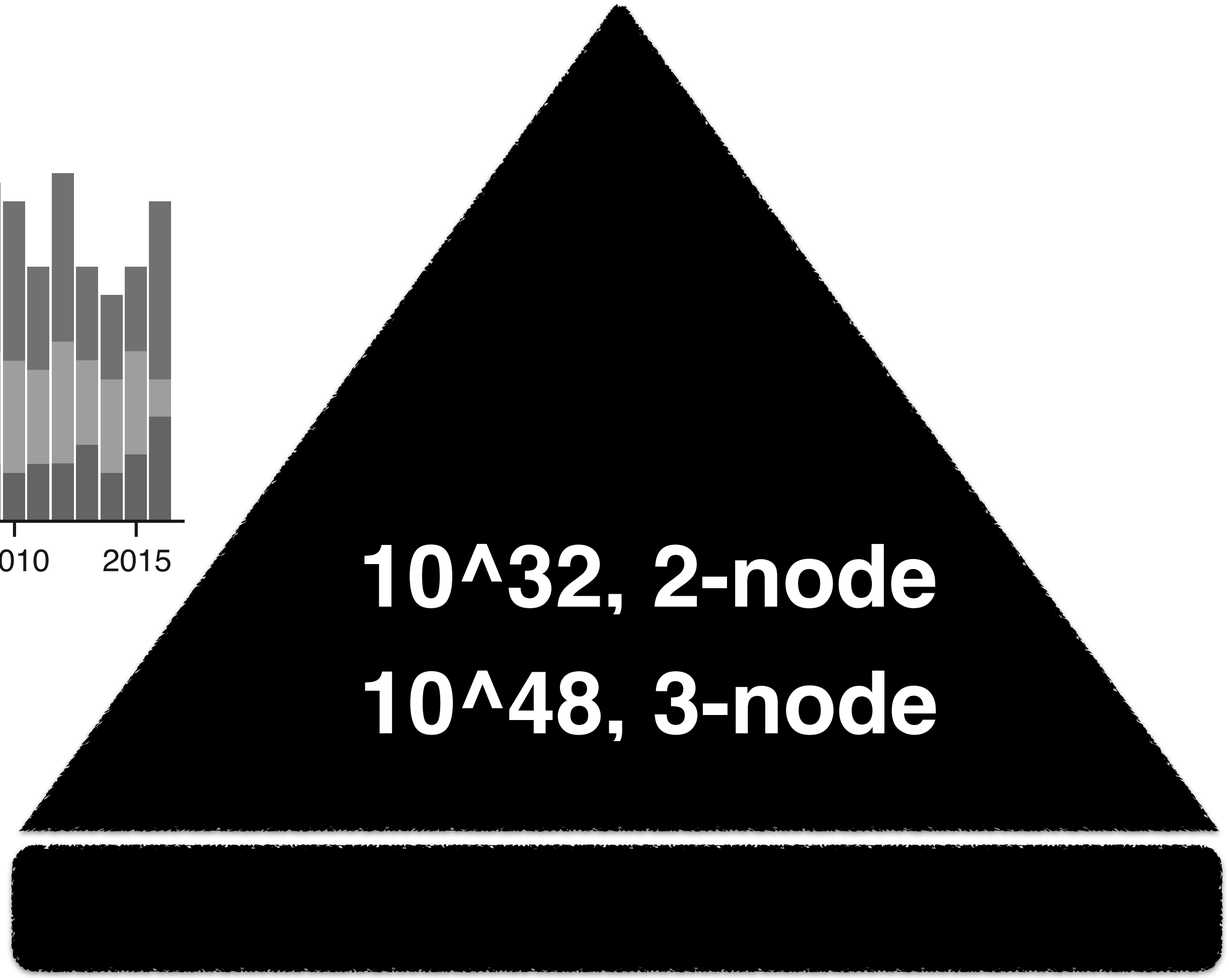
STARS IN THE SKY



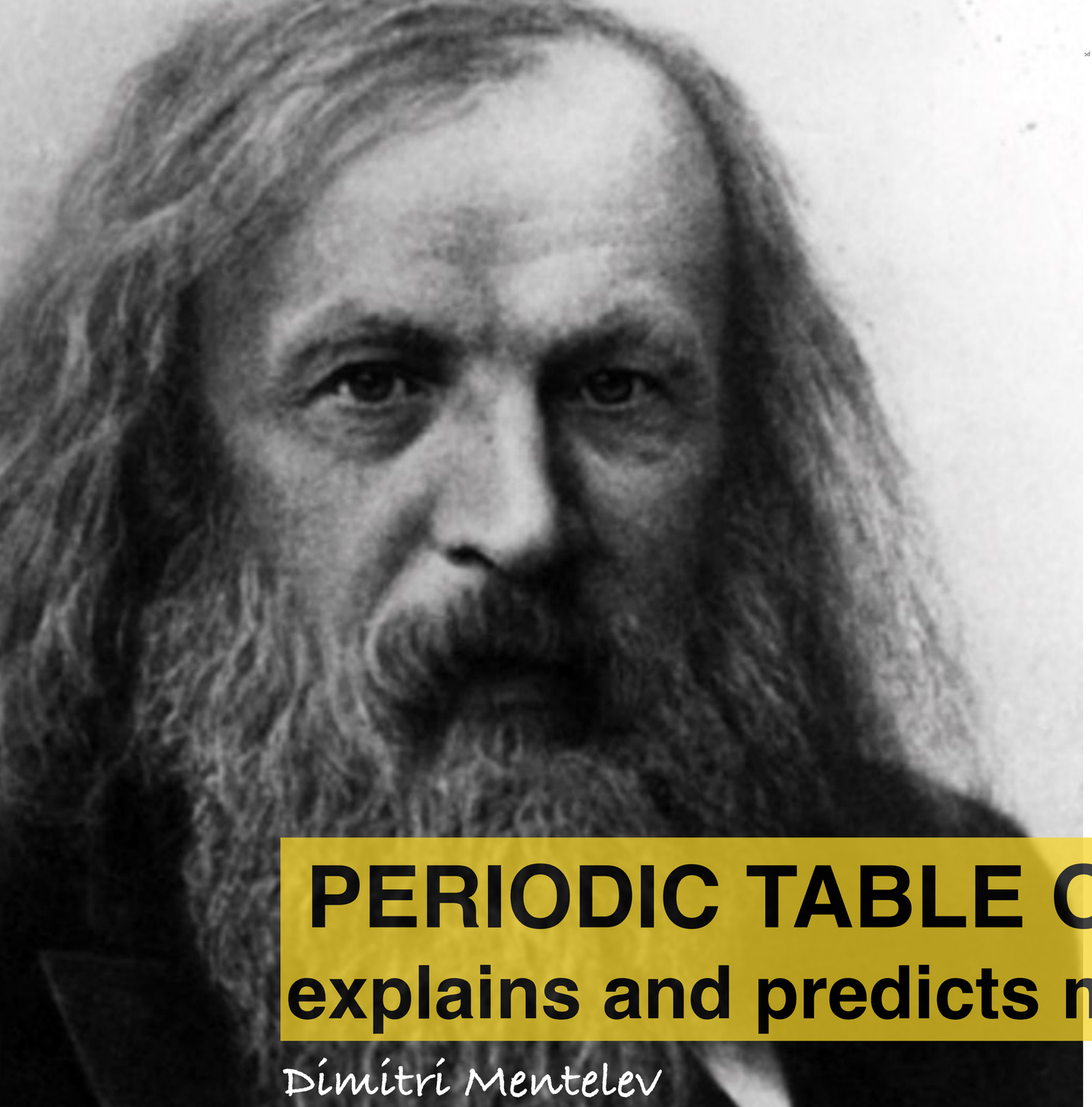
POSSIBLE DATA STRUCTURES

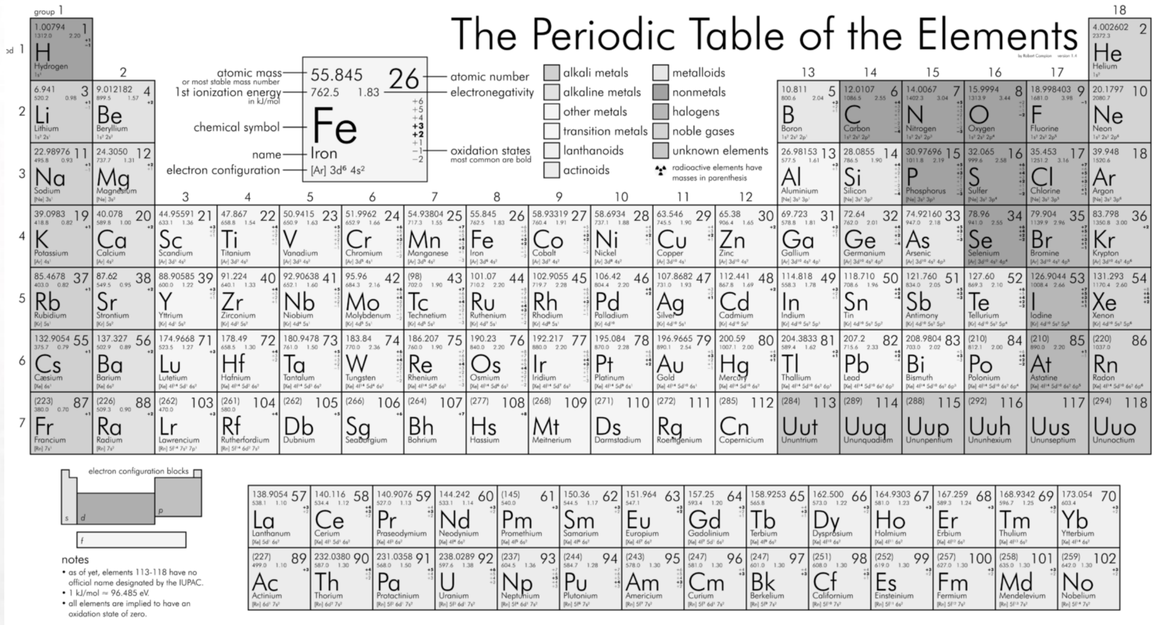






10^{48}-5x10^{3} = 10^{48} **zero progress?**





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

PERIODIC TABLE OF ELEMENTS

explains and predicts missing elements

Dimitri Mentelev

原子番号（陽子数）
元素名（和名）
元素記号
1
水素
H
nihonium
Kosuke Morita

periodic table of data structures

classes of primitives \ classes of designs	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	RUM
Logarithmic Design	DONE	DONE	DONE							RUM
Fractional Cascading	DONE		DONE	DONE						RUM
Log-Structured	DONE		DONE	DONE						RUM
Buffering	DONE			DONE			DONE			RUM
Differential Updates	DONE			DONE						RUM
Sparse Indexing	DONE				DONE	DONE				RUM
Adaptivity	DONE								DONE	

periodic table of data structures

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

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	RUM
Logarithmic	DONE	DONE	DONE							
Differential Updates	DONE			DONE						RUM
Sparse Indexing	DONE				DONE	DONE				RUM
Adaptivity	DONE								DONE	

PAPER MACHINE

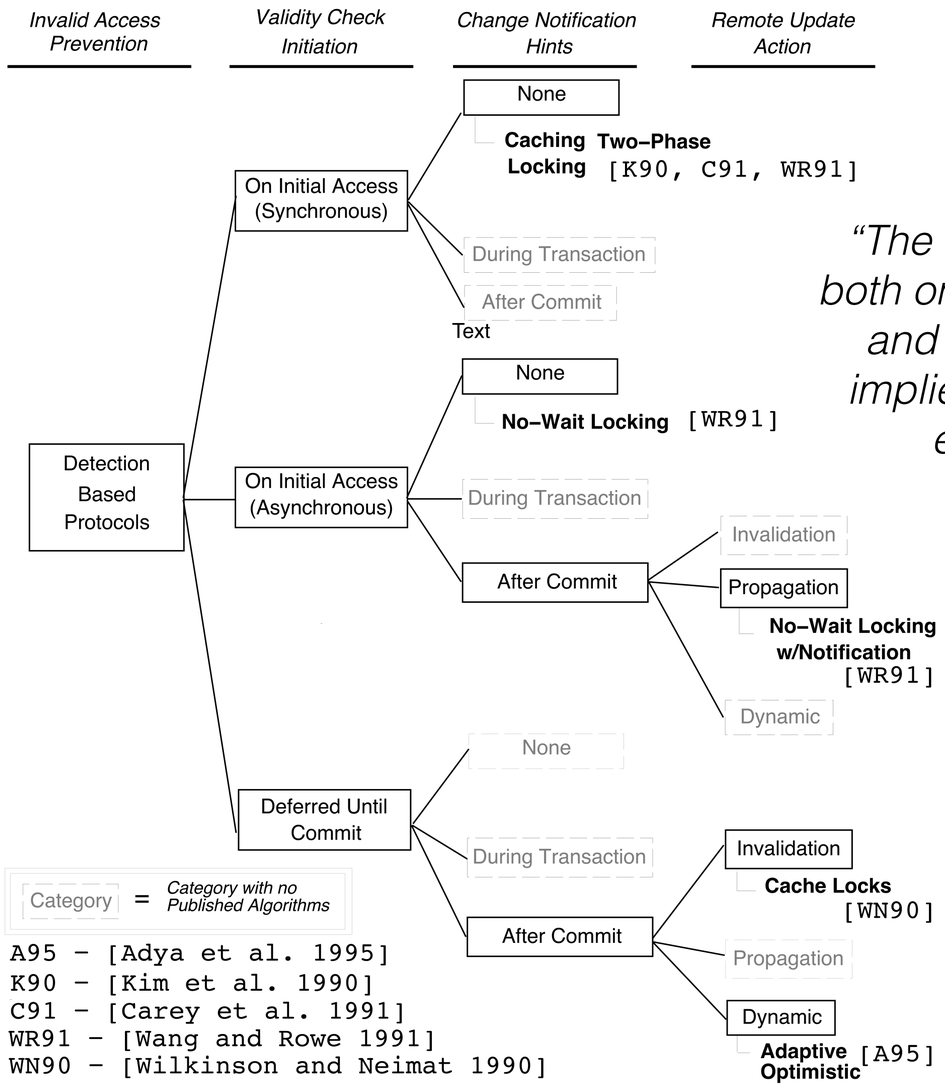
periodic table of data structures

classes of primitives \ classes of designs	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	RUM
Logarithmic	DONE	DONE	DONE							
Differential Updates	DONE			DONE			✓			RUM
Sparse Indexing	DONE				DONE	DONE	✓			RUM
Adaptivity	DONE						✓		DONE	

PAPER MACHINE

updatable bitmap indexes
@SIGMOD16

Mike Franklin

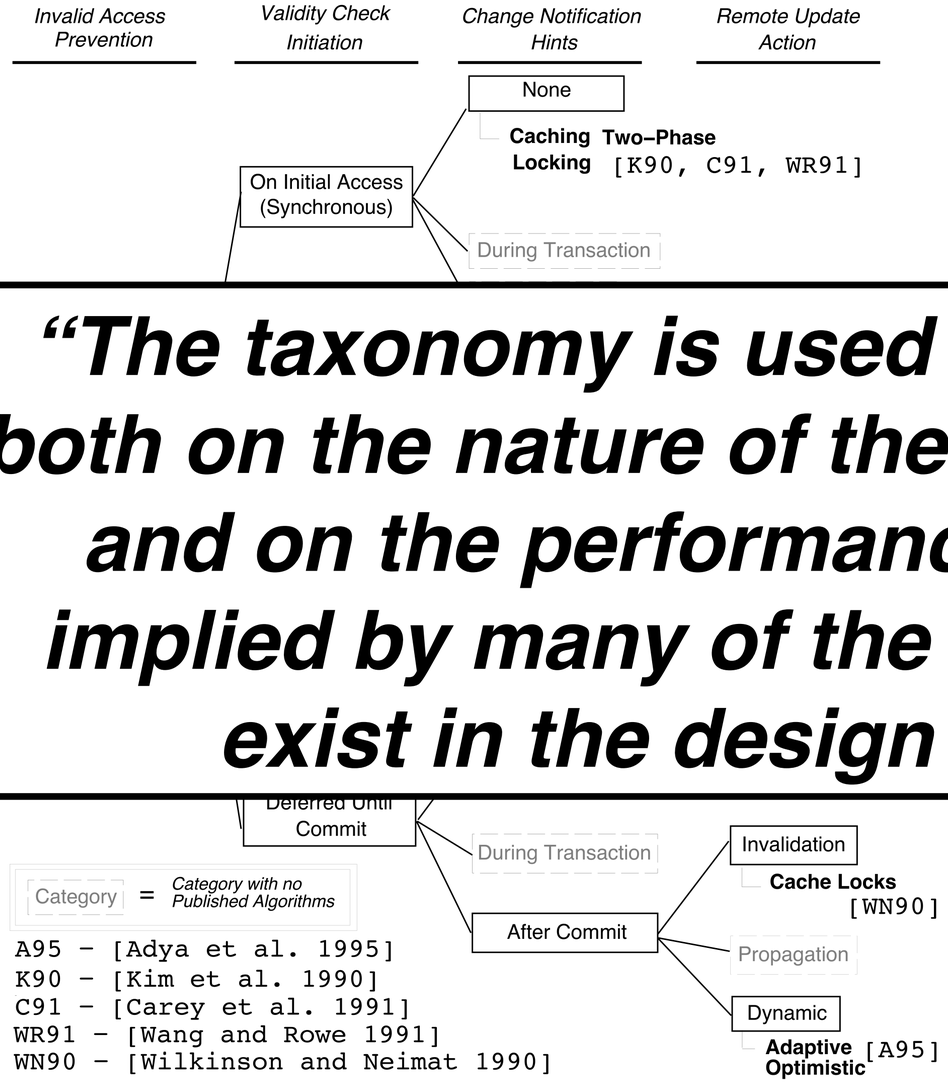


"The taxonomy is used to shed light both on the nature of the design space and on the performance tradeoffs implied by many of the choices that exist in the design space."

TAXONOMY OF COMPLEX ALGORITHMS

transactional cache consistency maintenance

Mike Franklin



"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."

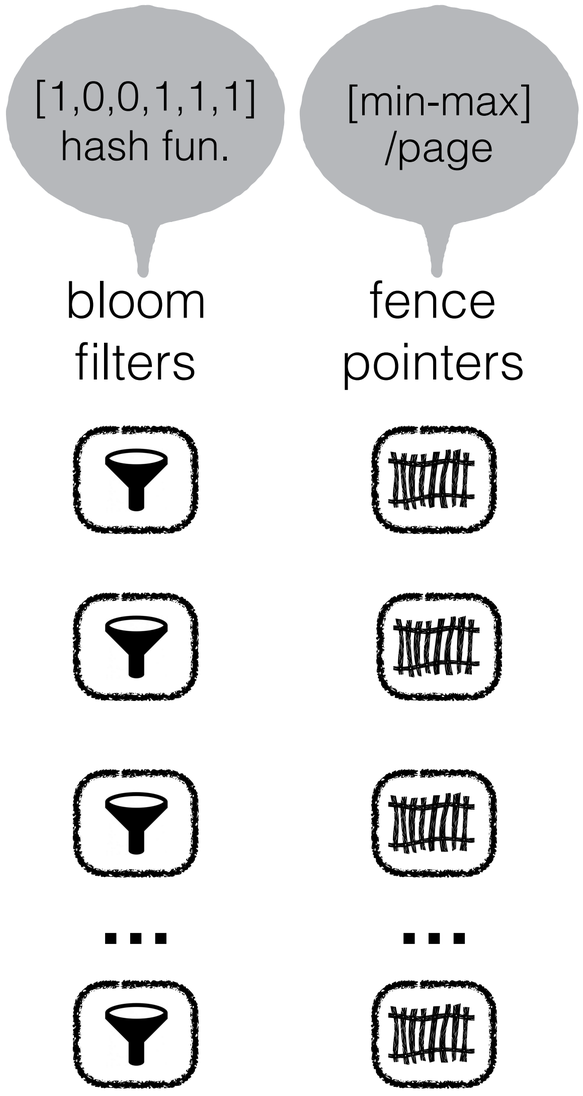
TAXONOMY OF COMPLEX ALGORITHMS

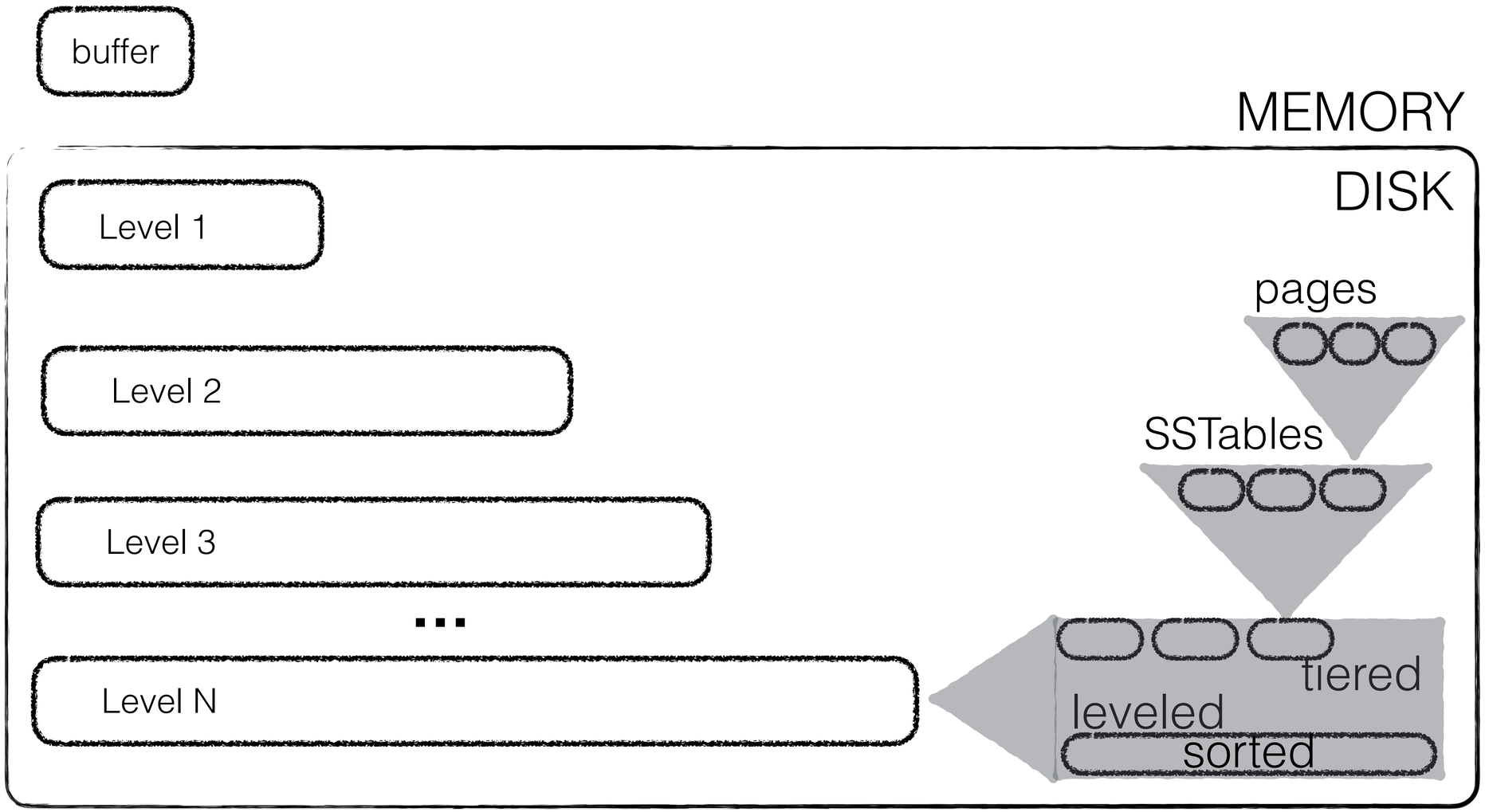
transactional cache consistency maintenance

DON BATORY

BARBARA LISKOV

Understanding the KV design space in more detail: size ratio and merging





merging

writes

reads

when we do more

merging

writes

reads

when we do more

merging

writes ↑

↓ **reads**

merging

Tiering
write-optimized

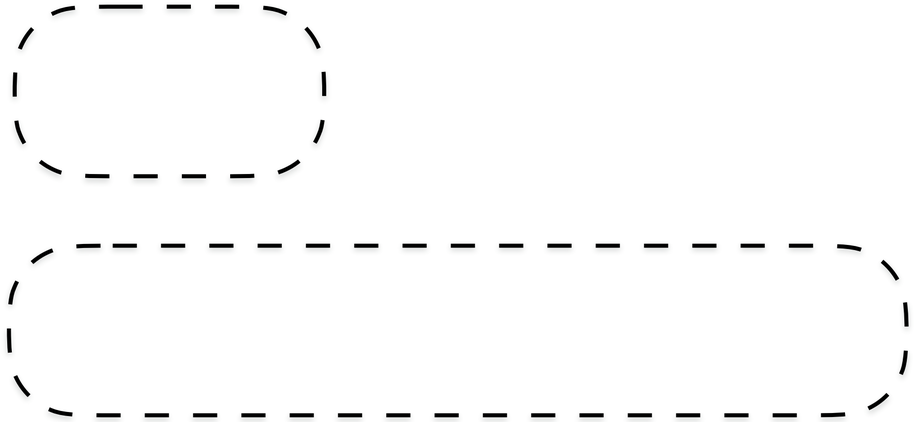
cassandra

Leveling
read-optimized

RocksDB

Tiering

write-optimized



Leveling

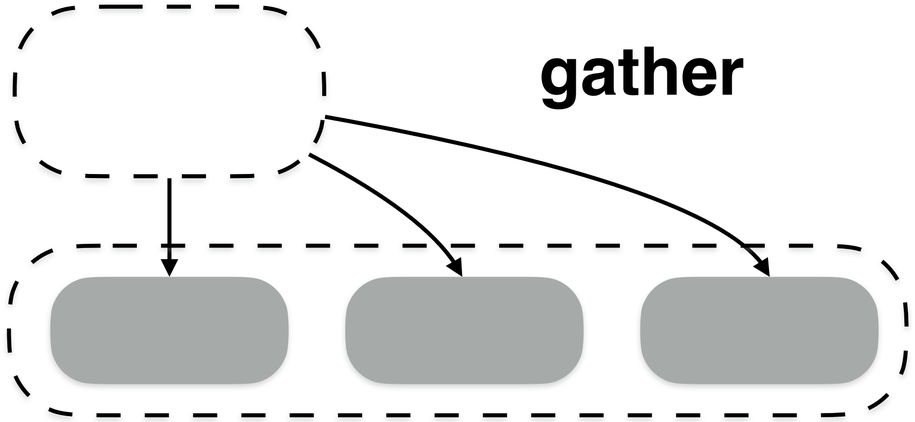
read-optimized

Tiering

write-optimized

Leveling

read-optimized

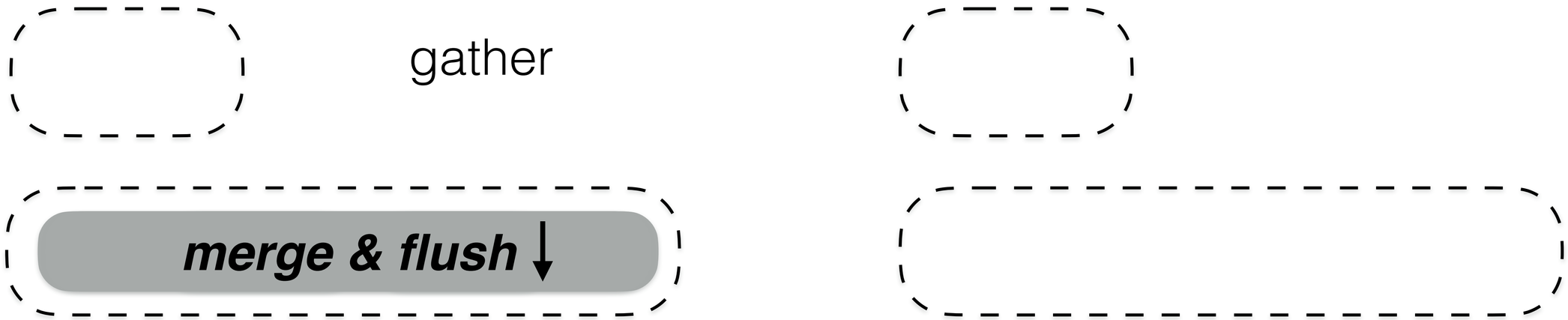


Tiering

write-optimized

Leveling

read-optimized

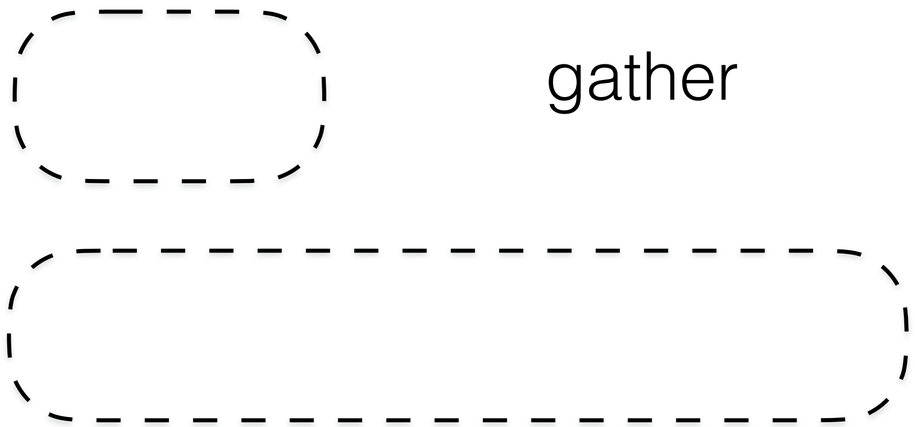


Tiering

write-optimized

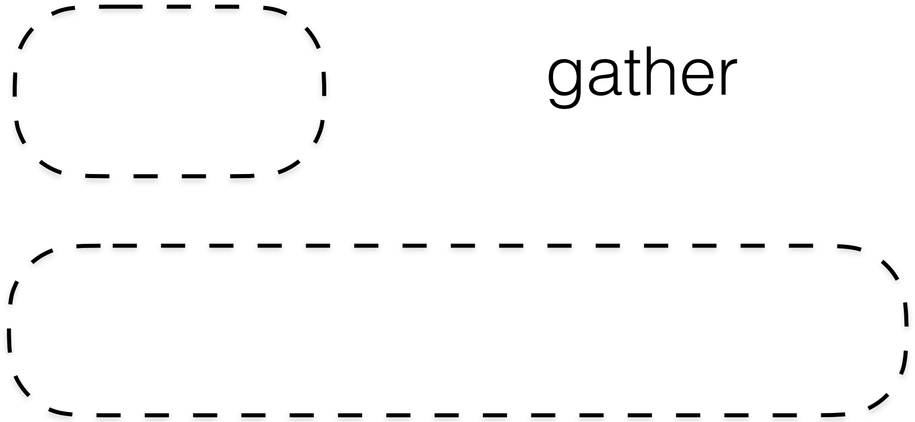
Leveling

read-optimized



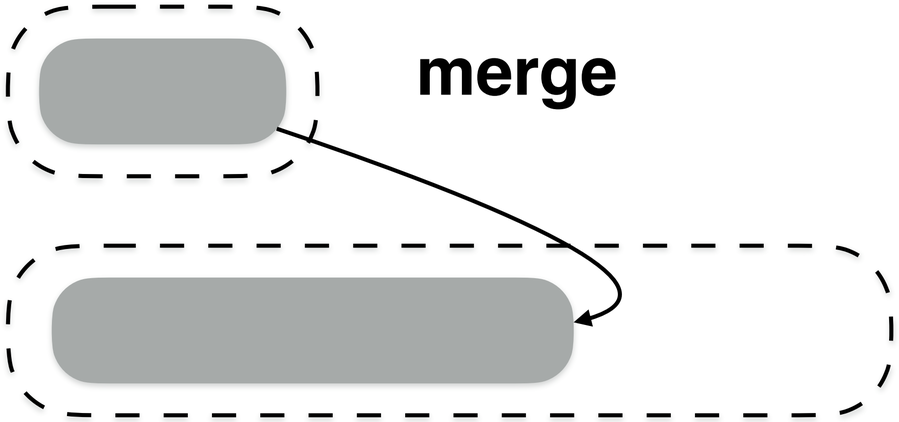
Tiering

write-optimized



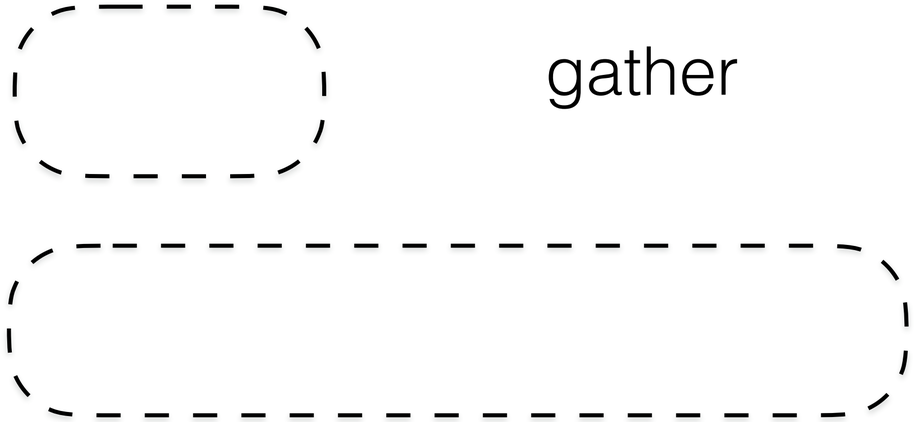
Leveling

read-optimized



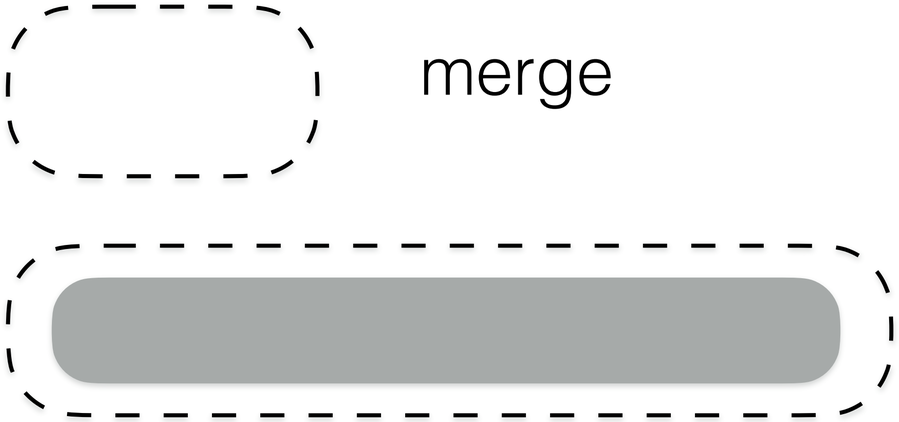
Tiering

write-optimized



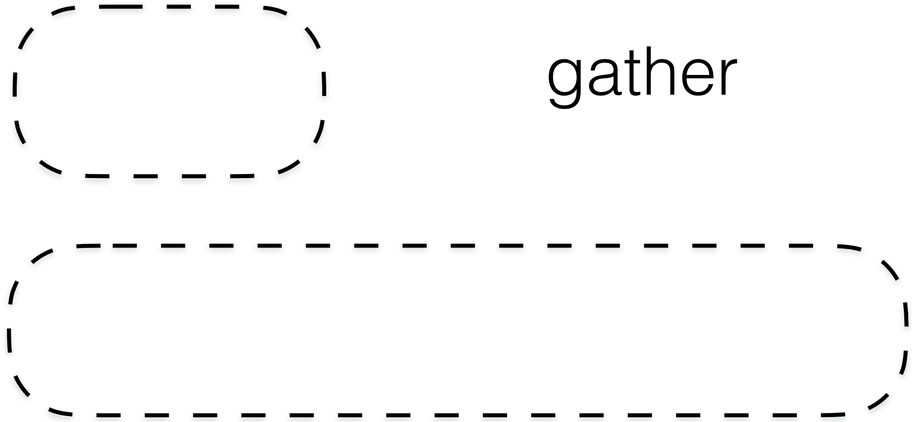
Leveling

read-optimized



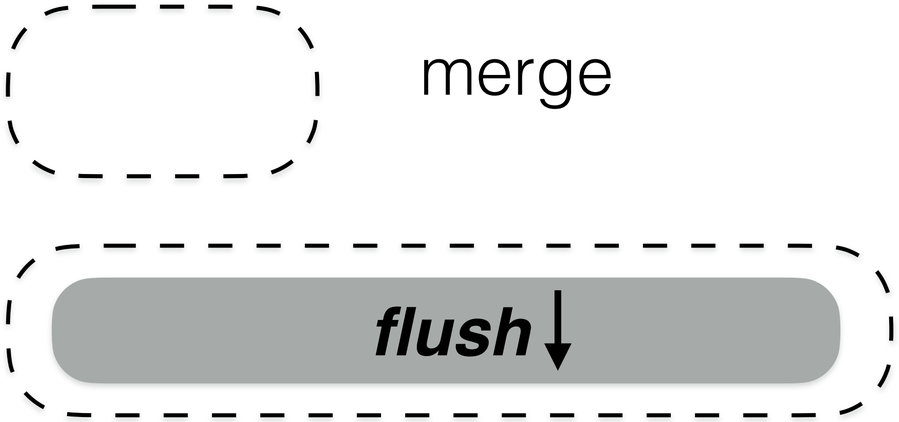
Tiering

write-optimized



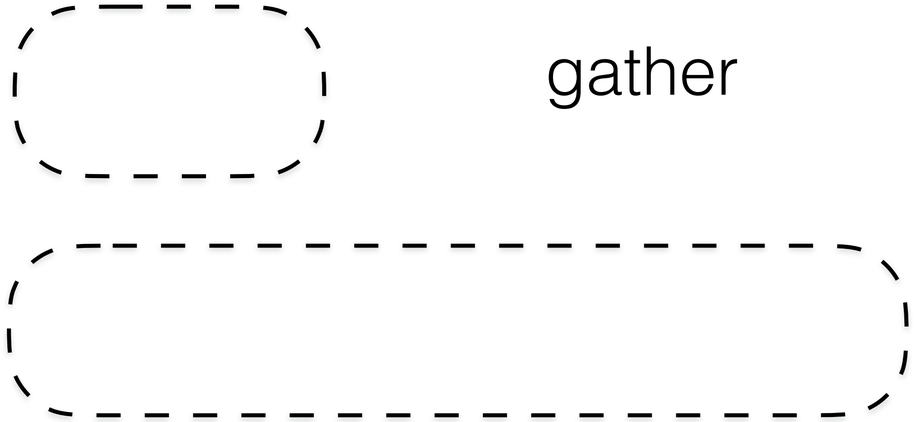
Leveling

read-optimized



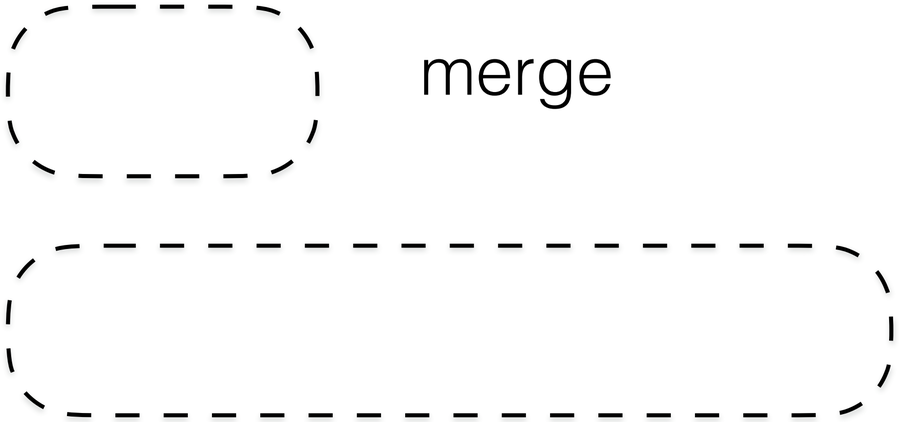
Tiering

write-optimized



Leveling

read-optimized



Tiering

write-optimized

Leveling

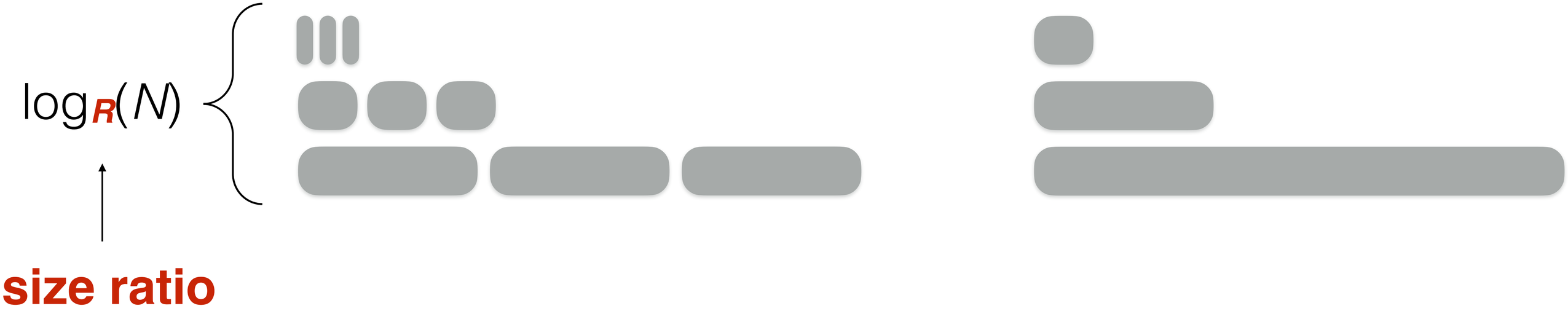
read-optimized

Tiering

write-optimized

Leveling

read-optimized

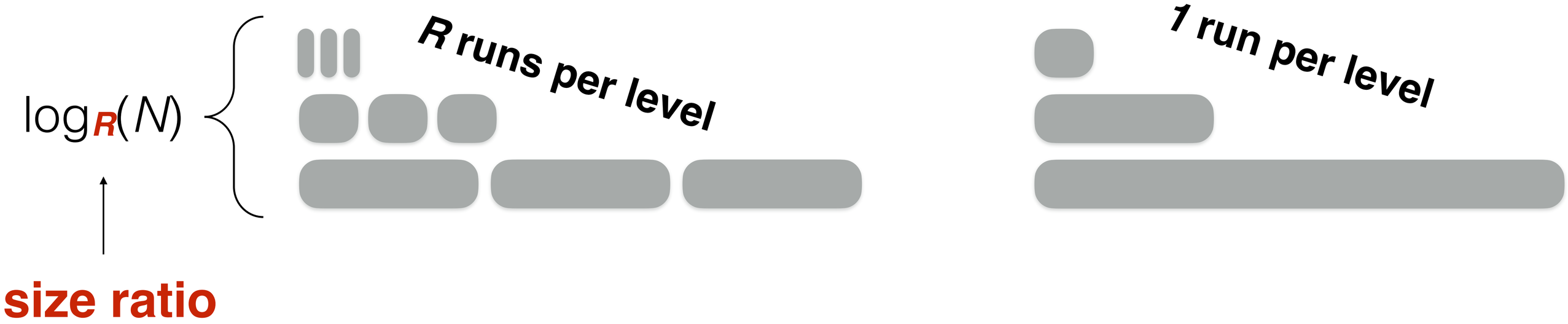


Tiering

write-optimized

Leveling

read-optimized



Tiering

write-optimized

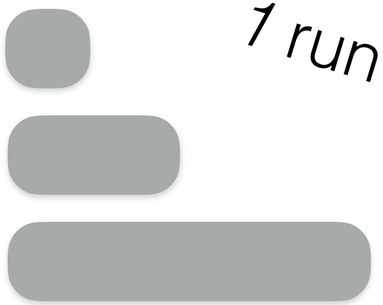
Leveling

read-optimized

size ratio *R*

Tiering

write-optimized



1 run per level

Leveling

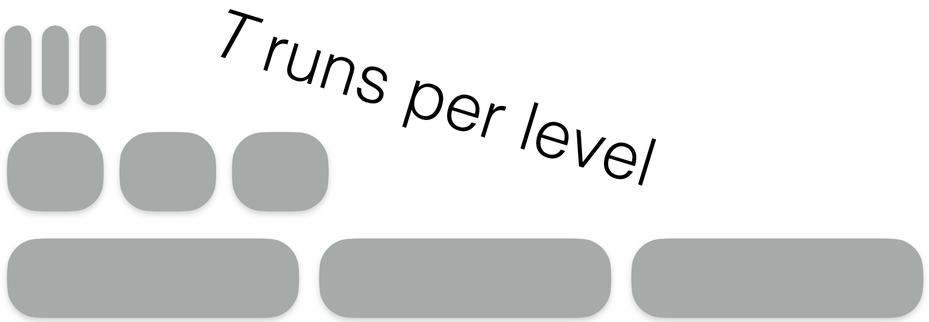
read-optimized

1 run per level

size ratio *R*

Tiering

write-optimized



Leveling

read-optimized

size ratio *R*

Tiering

write-optimized

O(*N*) runs per level

Leveling

read-optimized

1 run per level

size ratio *R* ⌃

Tiering

write-optimized

Leveling

read-optimized

O(*N*) runs per level

log

1 run per level

size ratio *R* ⌃

Tiering

write-optimized

O(*N*) runs per level

log

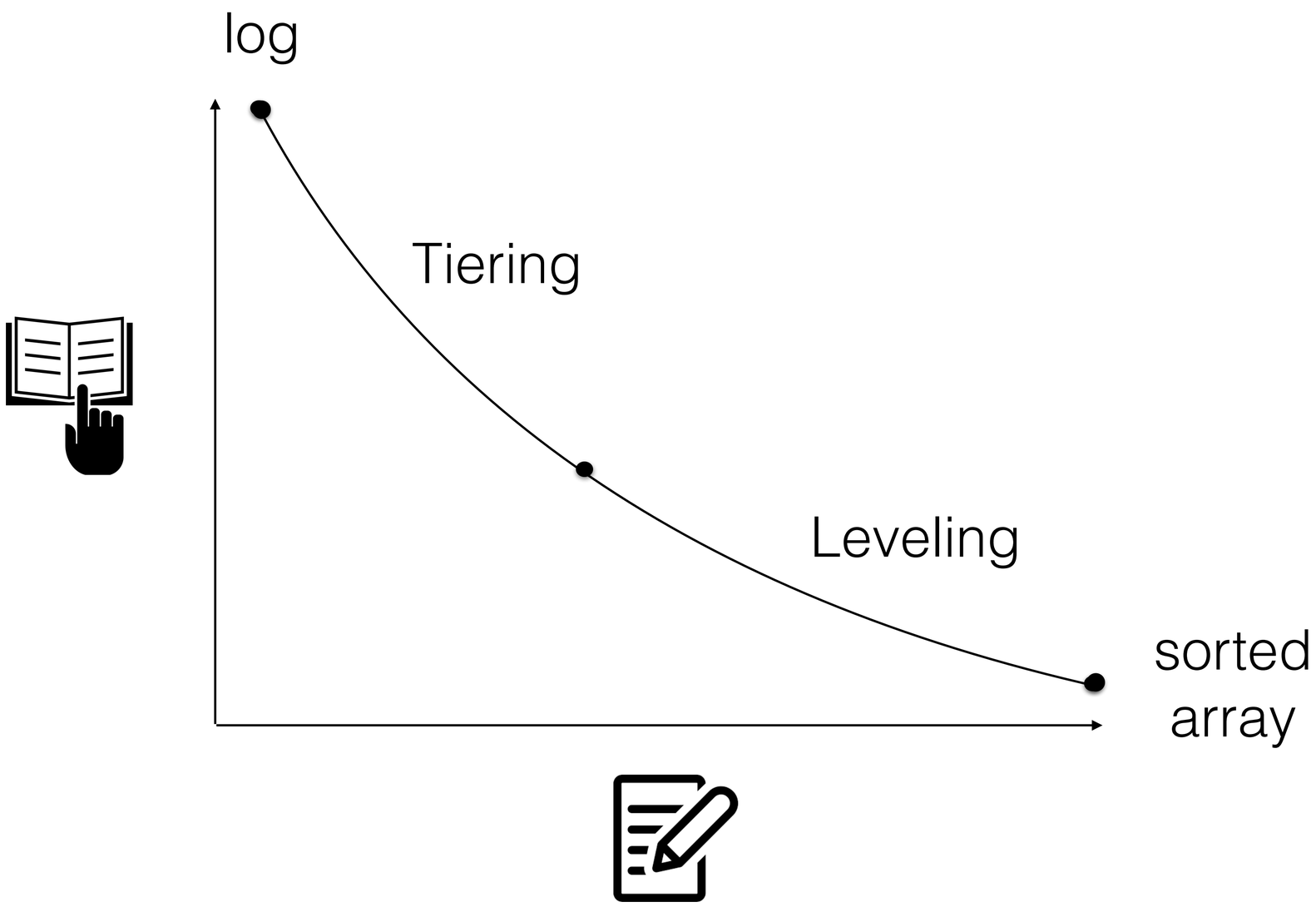
Leveling

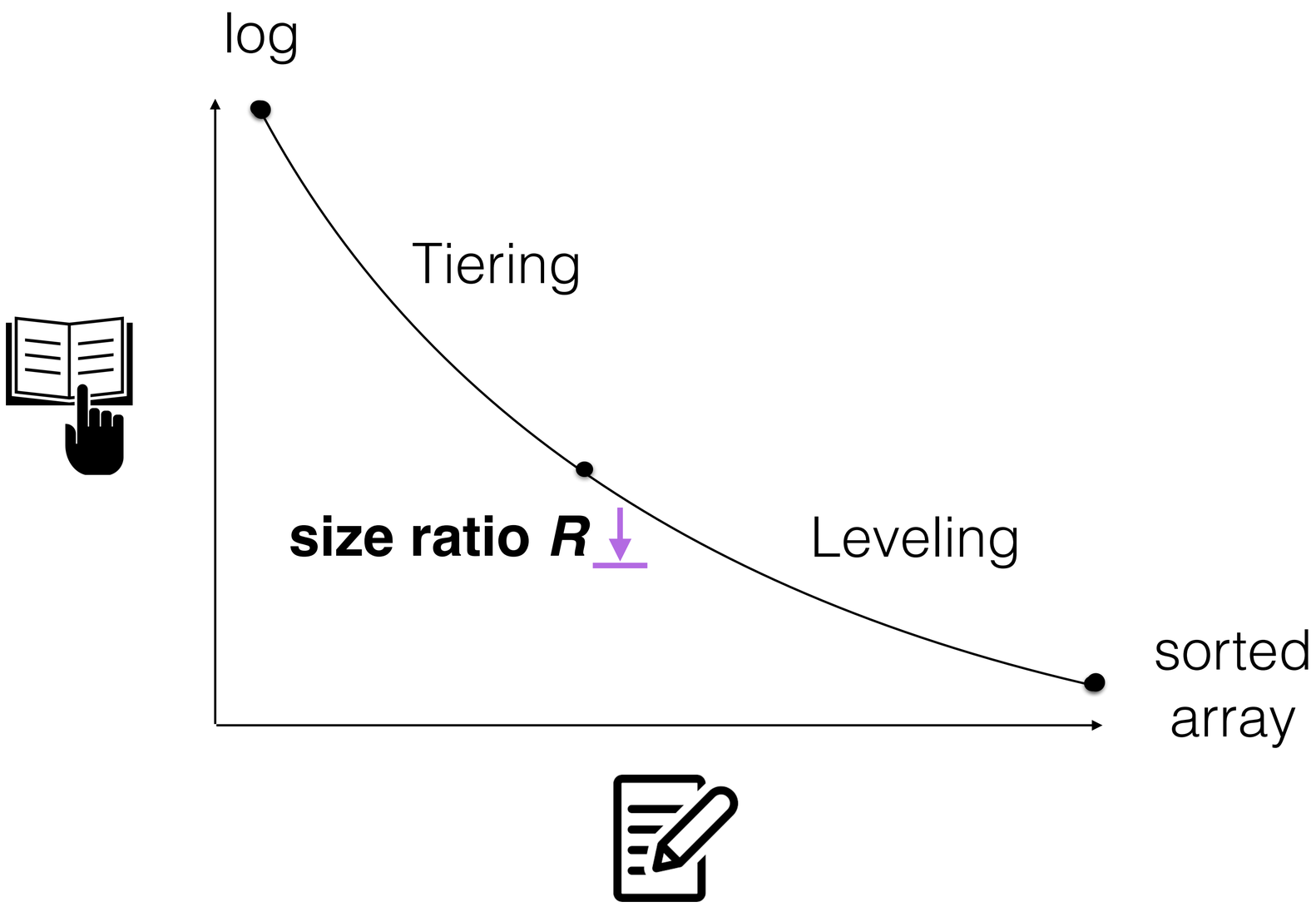
read-optimized

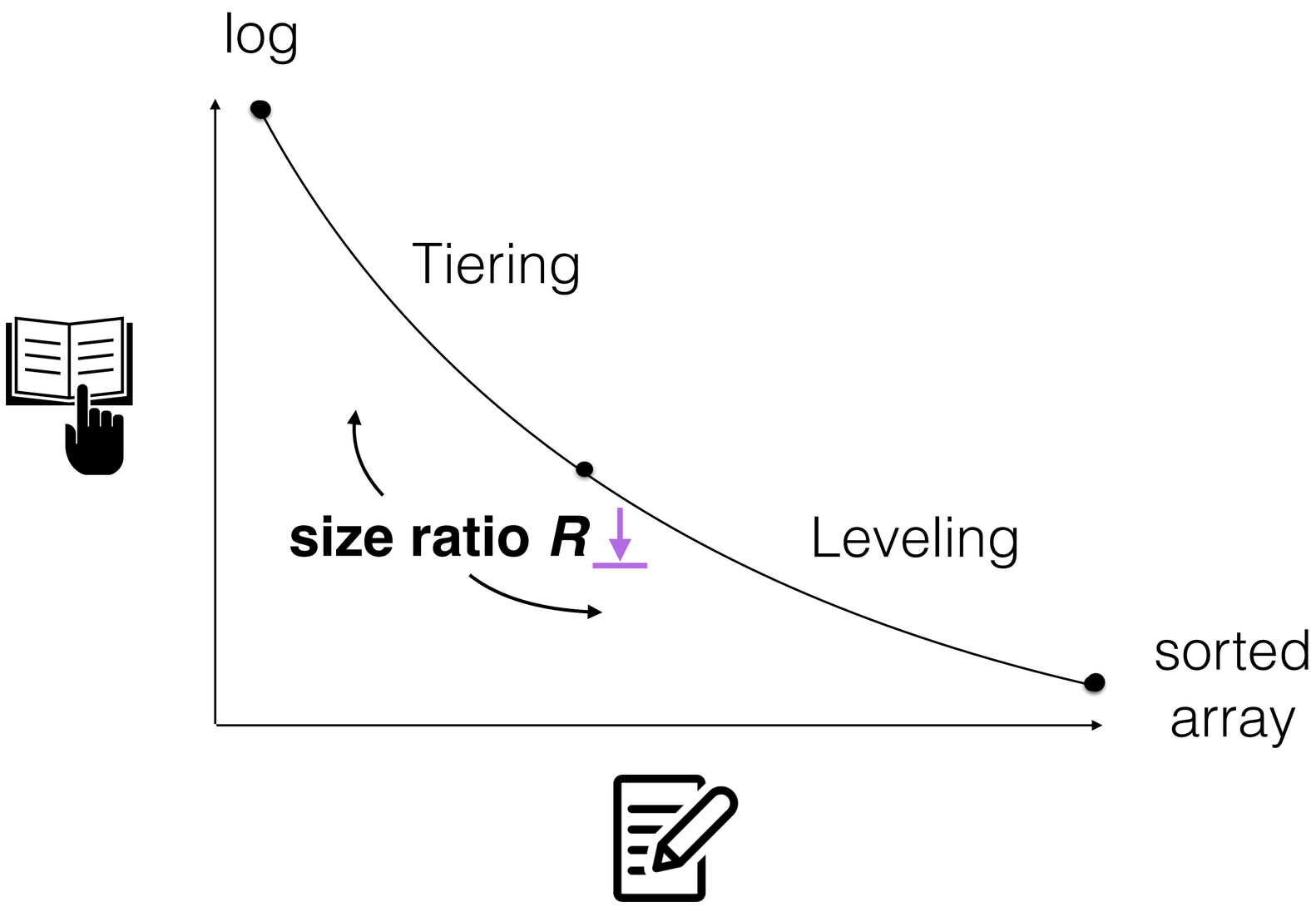
1 run per level

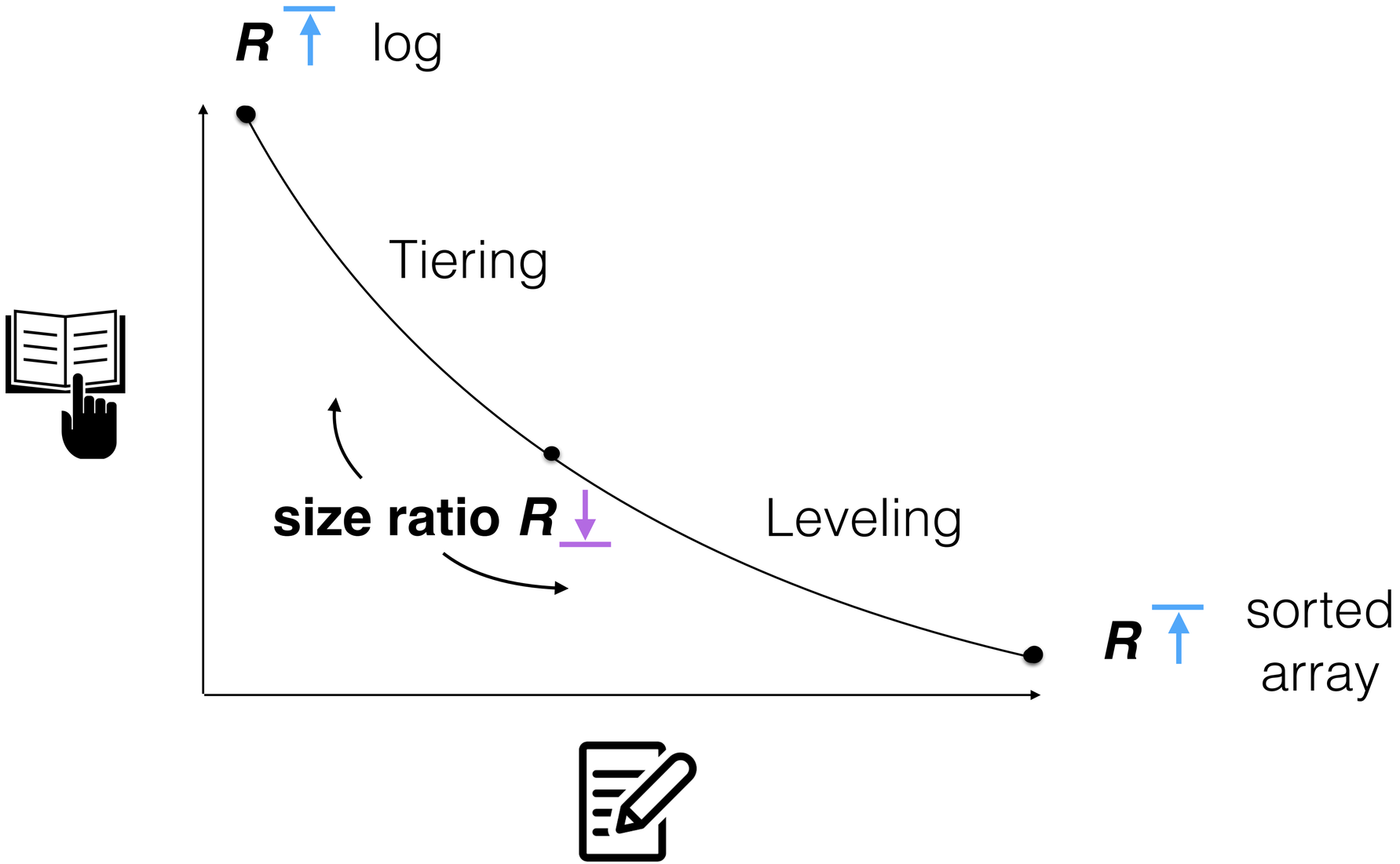
sorted array

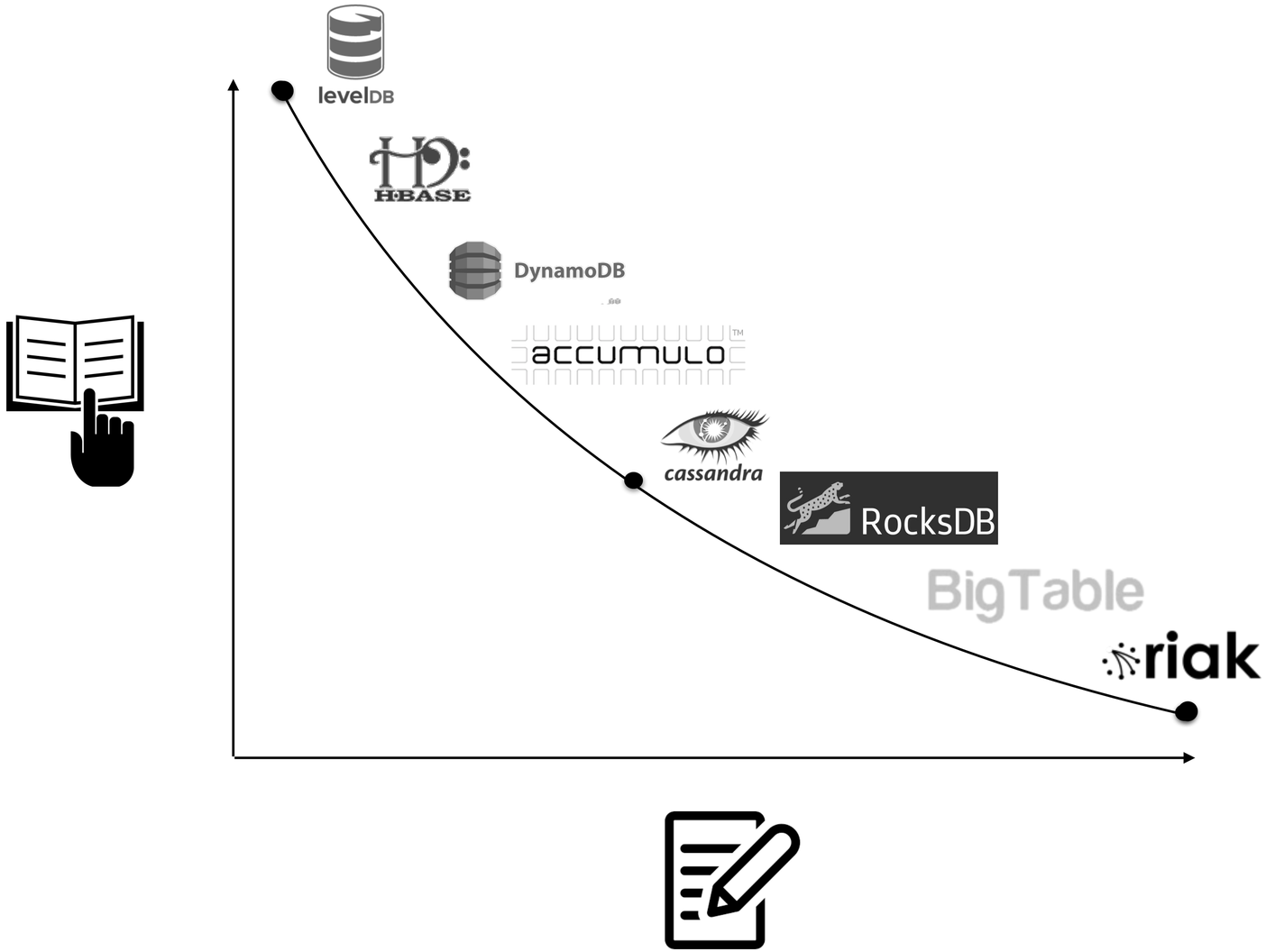
size ratio *R* ⌃

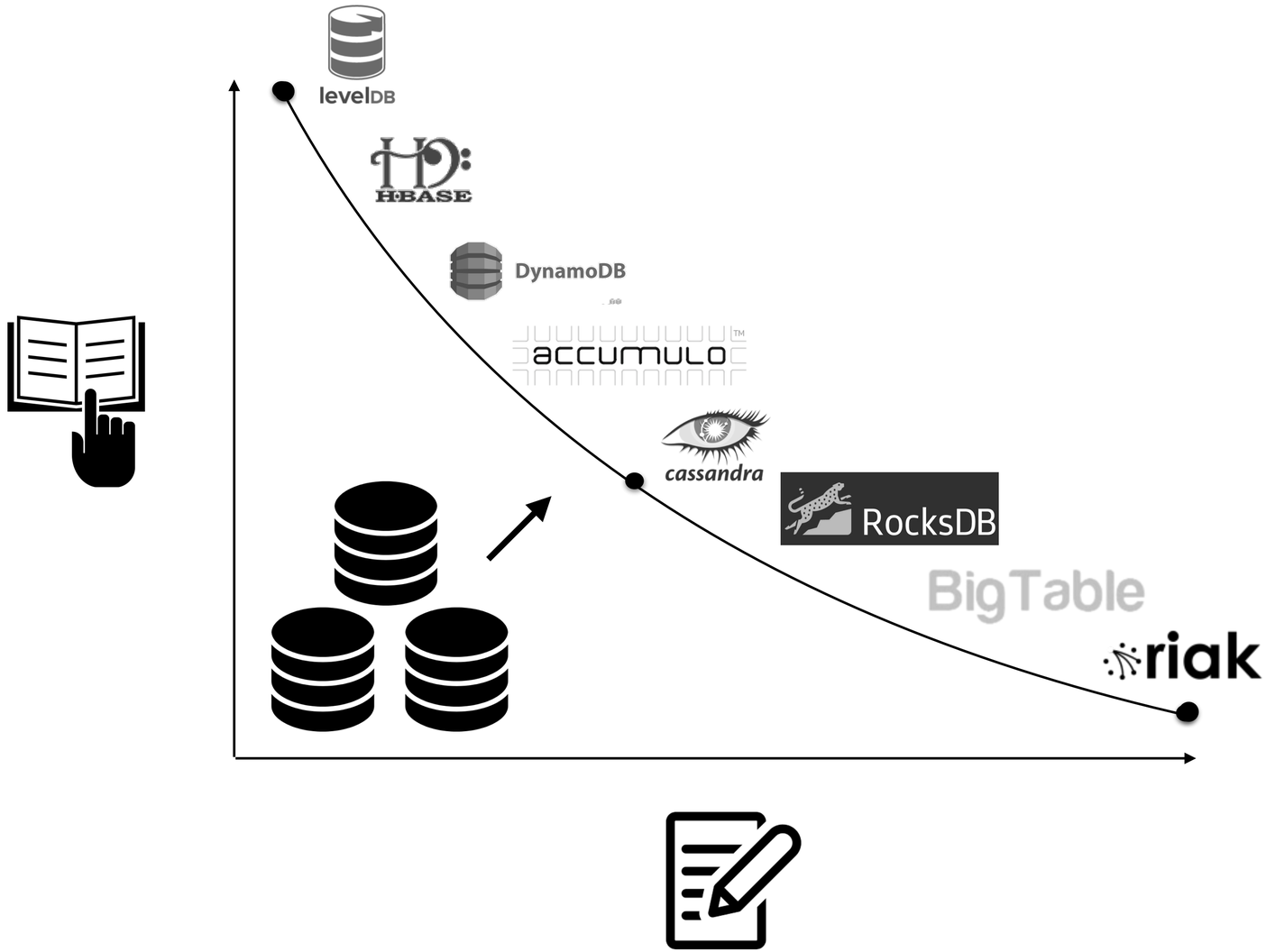

log
Tiering
Leveling
sorted
array


log
Tiering
size ratio R
Leveling
sorted
array

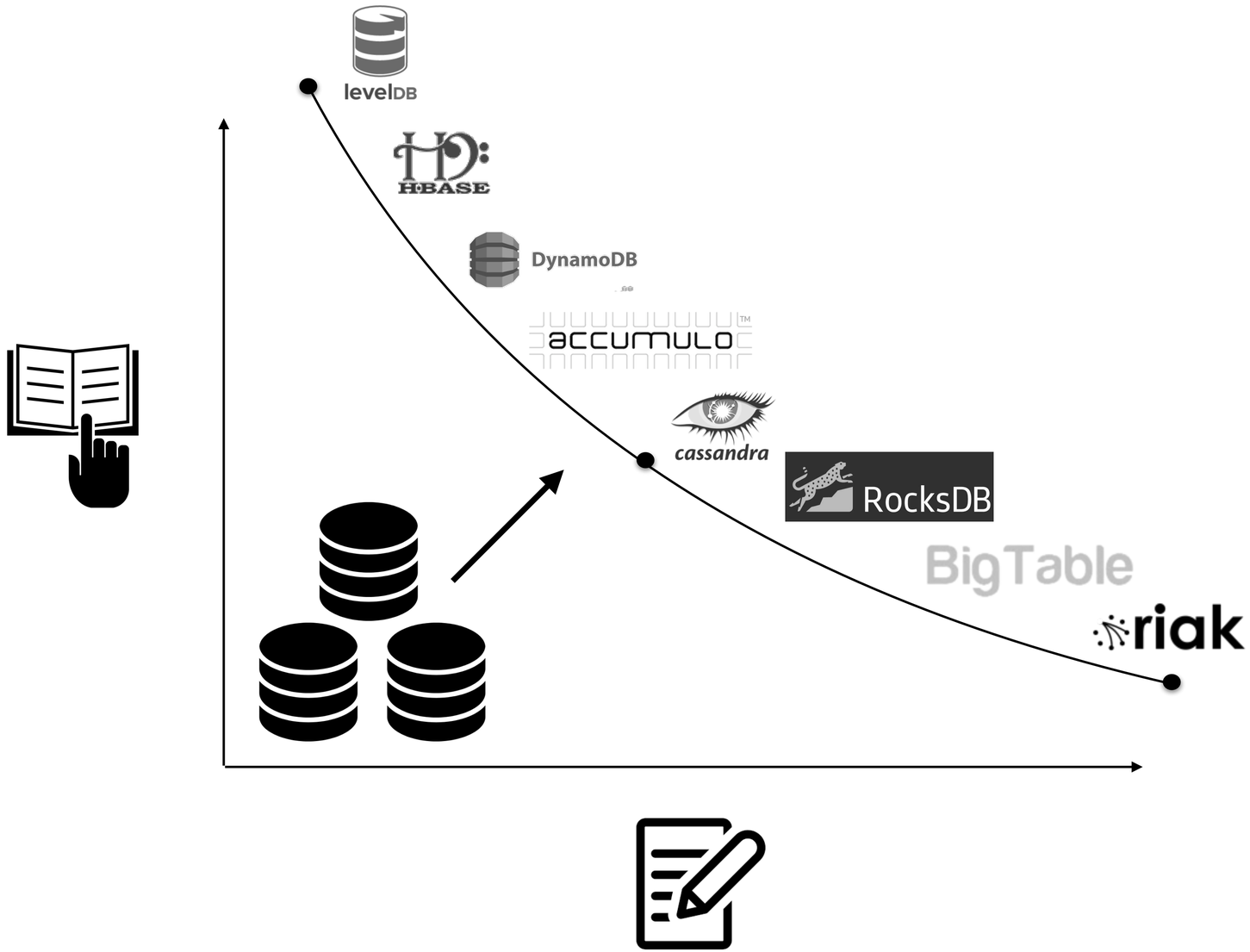

log
Tiering
size ratio R
Leveling
sorted
array


R
log
Tiering
size ratio R
Leveling
R
sorted
array


levelDB
HBASE
DynamoDB
accumulo
cassandra
RocksDB
BigTable
riak

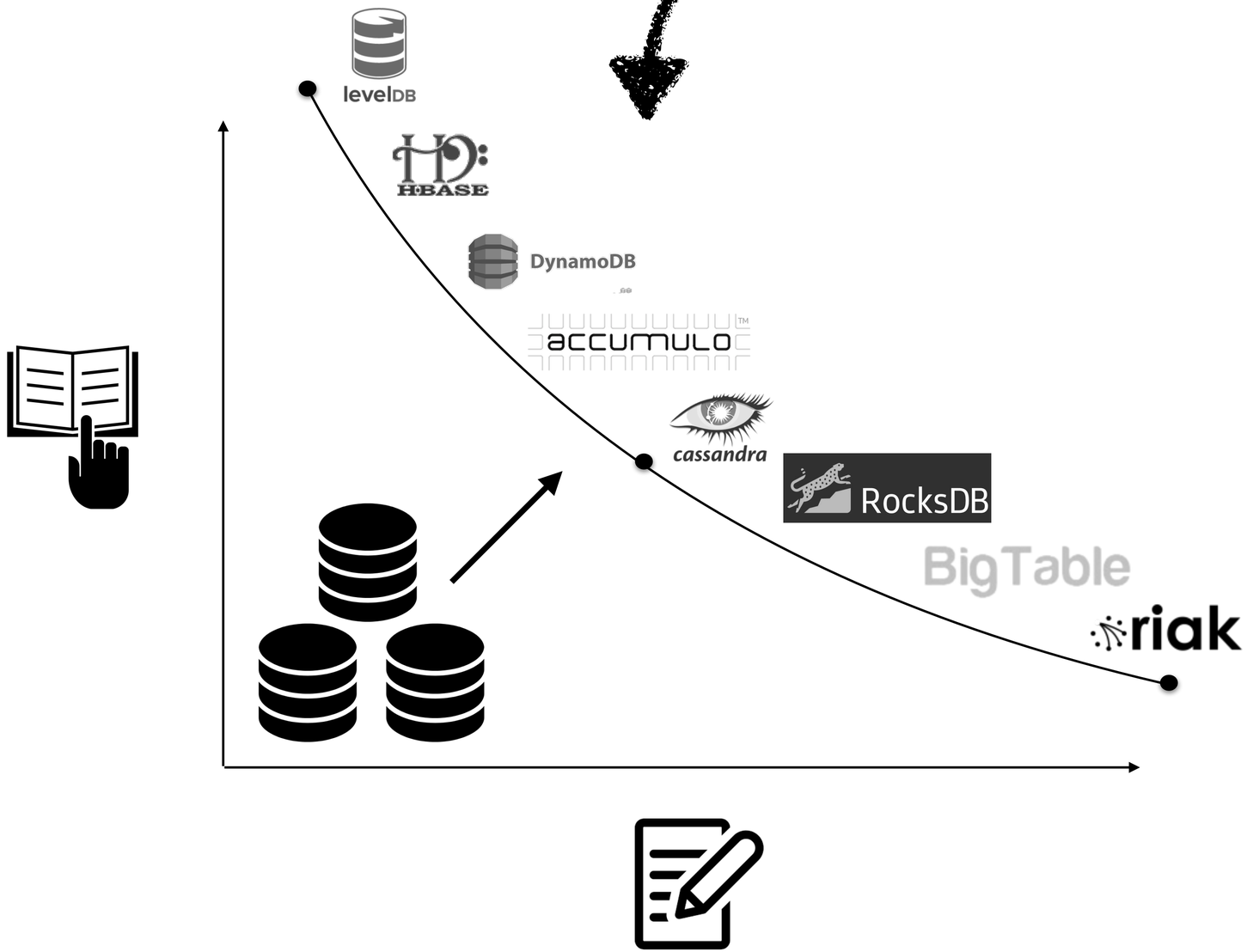


what happens as we collect more data?

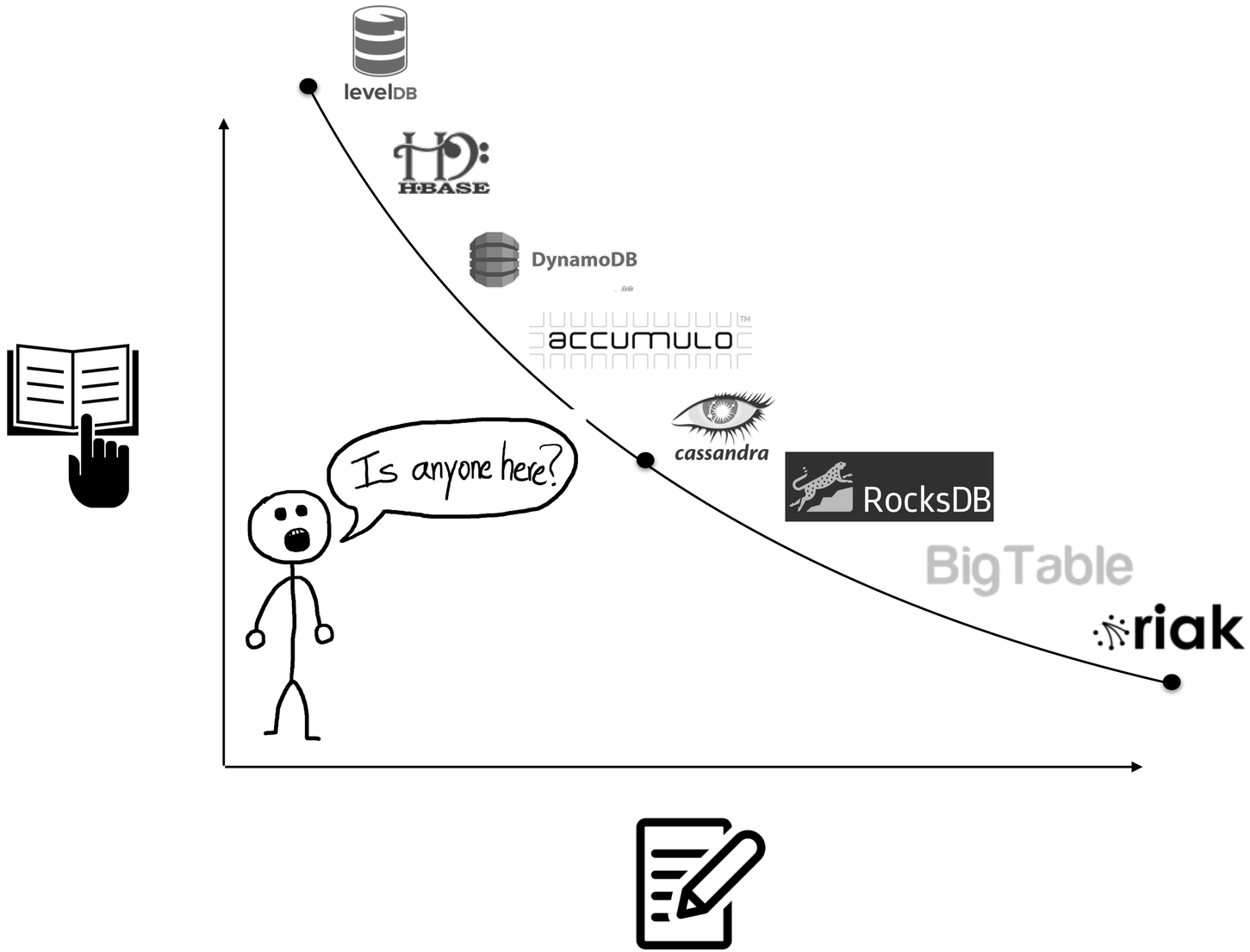


what happens as we collect more data?

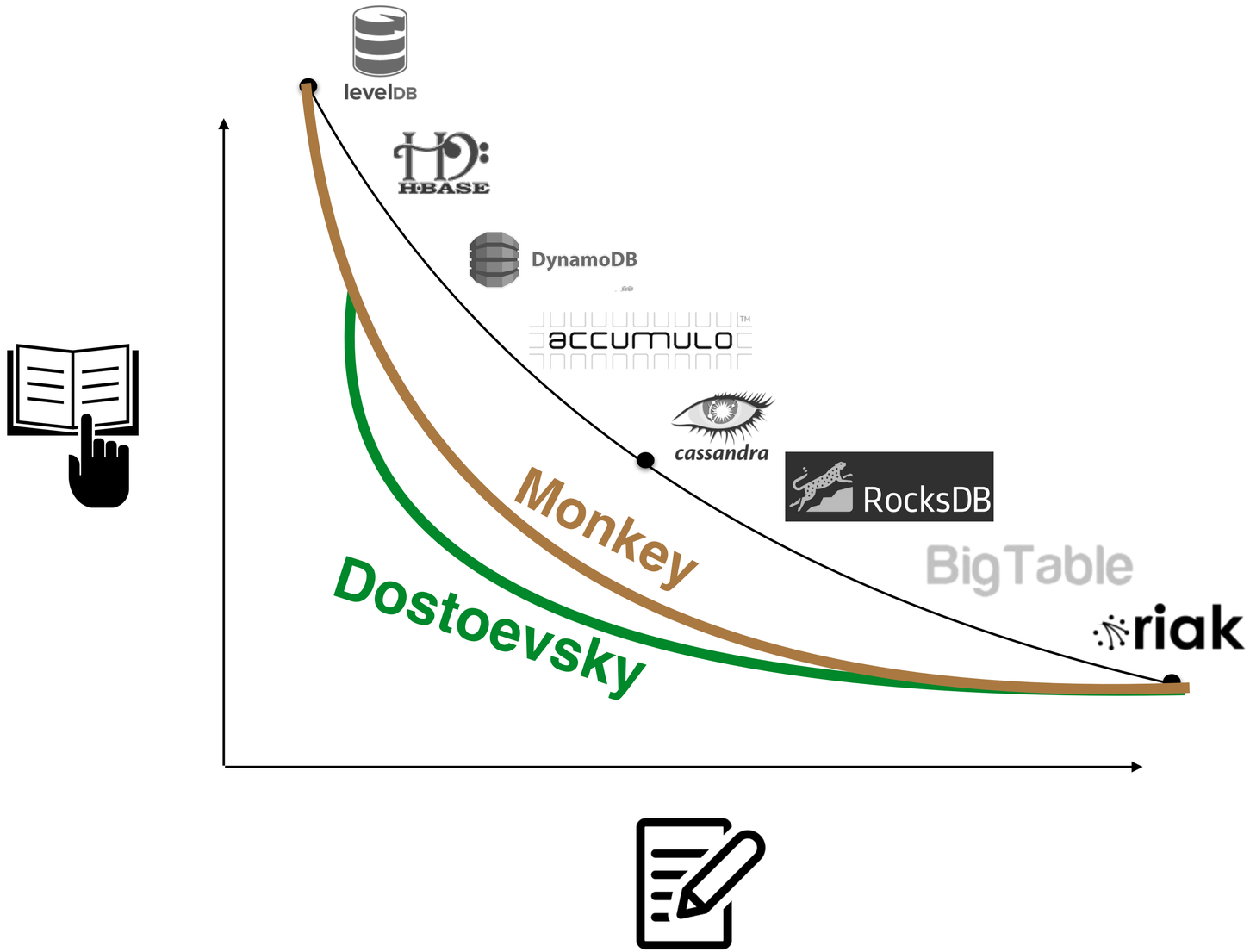
both reads and writes get worse!



what happens as we collect more data?



what happens as we collect more data?



what happens as we collect more data?

Readings for this week (and systems project)

Monkey: Optimal Navigable Key-Value Store. Niv Dayan, Manos Athanassoulis, Stratos Idreos. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 2017

Dostoevsky: Better Space-Time Trade-Offs for LSM-Tree Based Key-Value Stores via Adaptive Removal of Superfluous Merging. Niv Dayan, Stratos Idreos. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 2018

The Log-Structured Merge-Bush & the Wacky Continuum. Niv Dayan, Stratos Idreos. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 2019

CS 265

Stratos Idreos

BIG DATA SYSTEMS

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