CS 265

Stratos Idreos

BIG DATA SYSTEMS

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

Paper/discussion phase to start after spring break.

1. FASTER: A Concurrent Key-Value Store with In-Place Updates.
2. The FastLanes Compression Layout: Decoding >100 Billion Integers per Second with Scalar Code.
3. PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel
4. Megatron LM: Training Multi-Billion Parameter Language Models Using Model Parallelism.
5. RAG-design from a system design space
6. Overlap vector search time with LLM inference time
7. CROSSBOW: Scaling Deep Learning with Small Batch Sizes on Multi-GPU Servers
8. Nautilus: An Optimized System for Deep Transfer Learning over Evolving Training Datasets

Presentations

Task: This week:
Register for presentation. First come first serve. 4 students per paper max.
Link on Forum

Task: **Prep for presentation** — >Slides, Content, Delivery, Q&A
Read the paper several times.
 Understand in-depth.
 Use labs for questions.
After having a slides draft, meet with the TFs to get feedback.
  TFs will schedule a meeting the week before class.

Preparing for presentations and reviews

review and slides should answer:

what is the problem
why is it important
why is it hard
why existing solutions do not work
what is the core intuition for the solution
solution step by step
does the paper prove its claims
exact setup of analysis/experiments
are there any gaps in the logic/proof
possible next steps

* follow a few citations to gain more background

And final question:
What can we use from this paper
technique or inspiration
towards self-designing systems?

how to prepare slides

no bullets 2 colors big text images animation for examples

how to prepare slides

no bullets 2 colors big text images animation for examples

story

one message per slide connection from slide to slide

how to prepare slides

no bullets 2 colors big text images animation for examples

story

one message per slide connection from slide to slide

all questions + discussion

add 3-4 interesting discussion points

Debug before presenting

1. Content: Did you answer all reviews questions?

2. Formatting: Did you apply all slide format rules?

Our goal:

Every presentation is better than the previous one

During discussion we will also touch on the presentation in every class

Grading will be relative to the presentation schedule

Reviews:

Submit in Canvas

One PDF page. 2-3 paragraphs max. Organize by Review questions.

The deadline is 30 minutes before class. Hard deadline.

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

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

One lecture on LLM+RAG

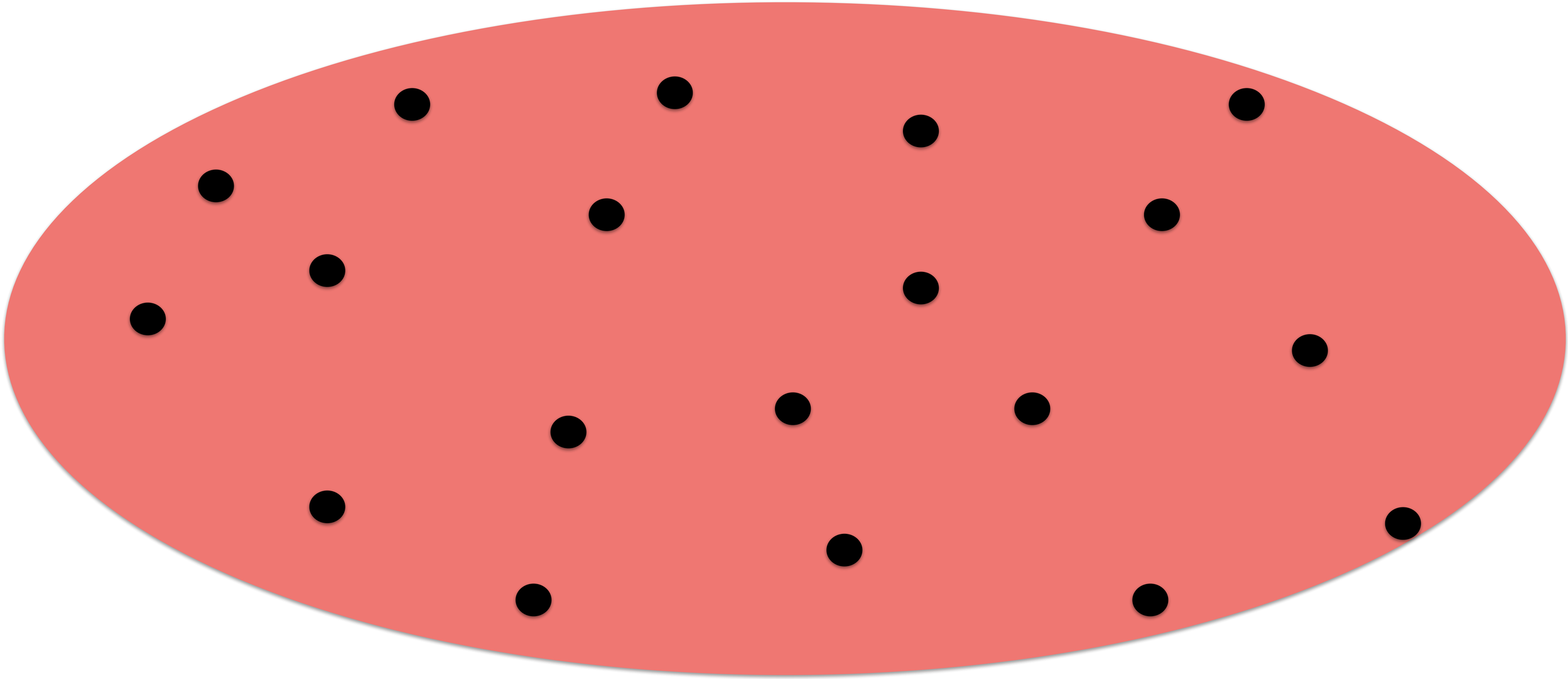
Then 3 lectures on end-to-end AI through reimagining storage

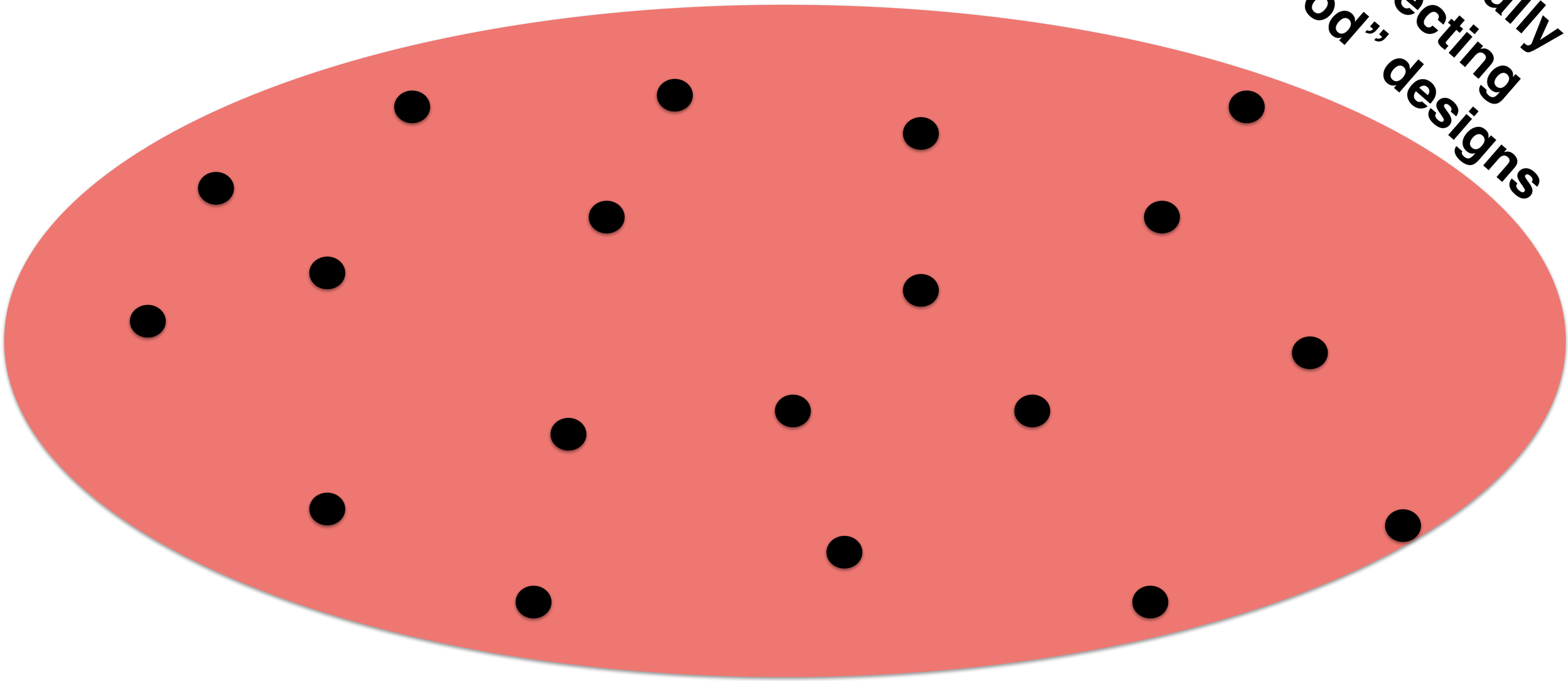
Then spring break

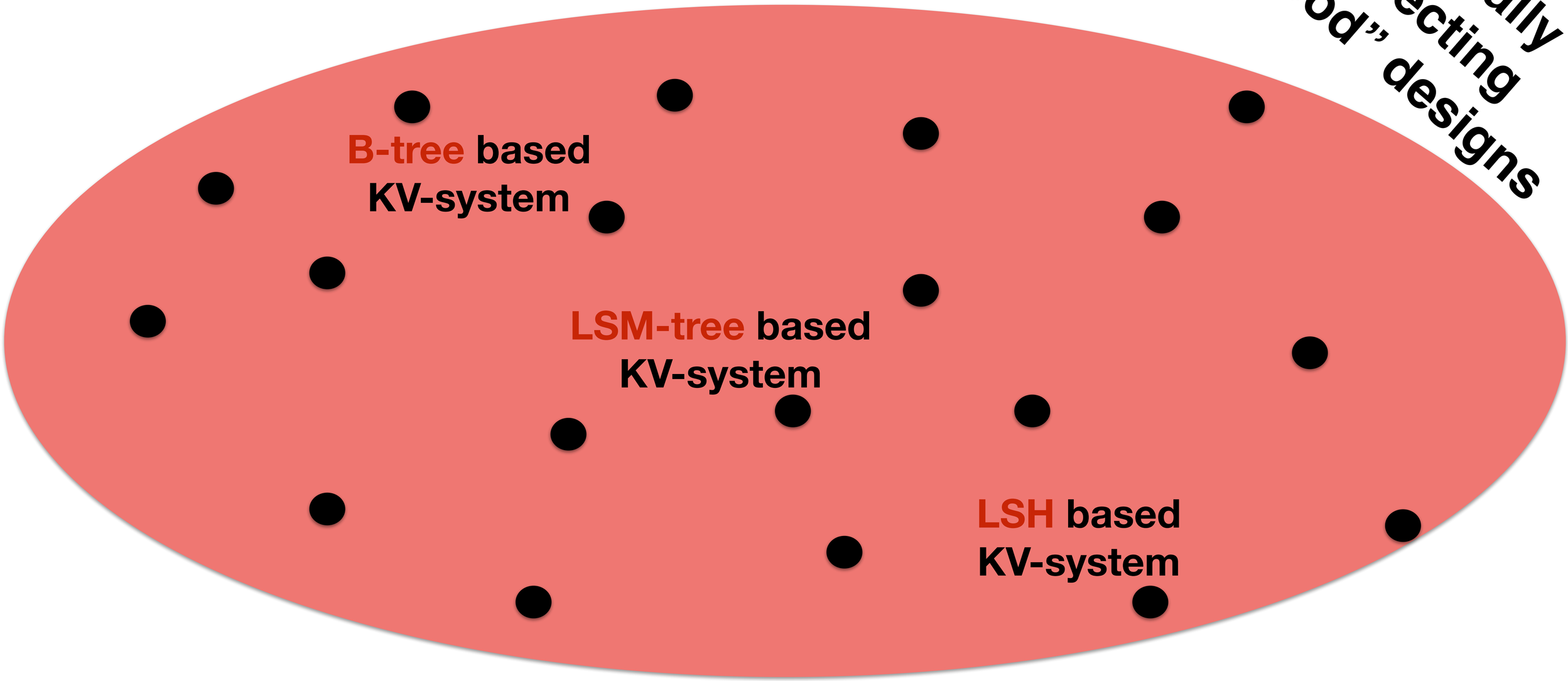
Then papers presentations

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

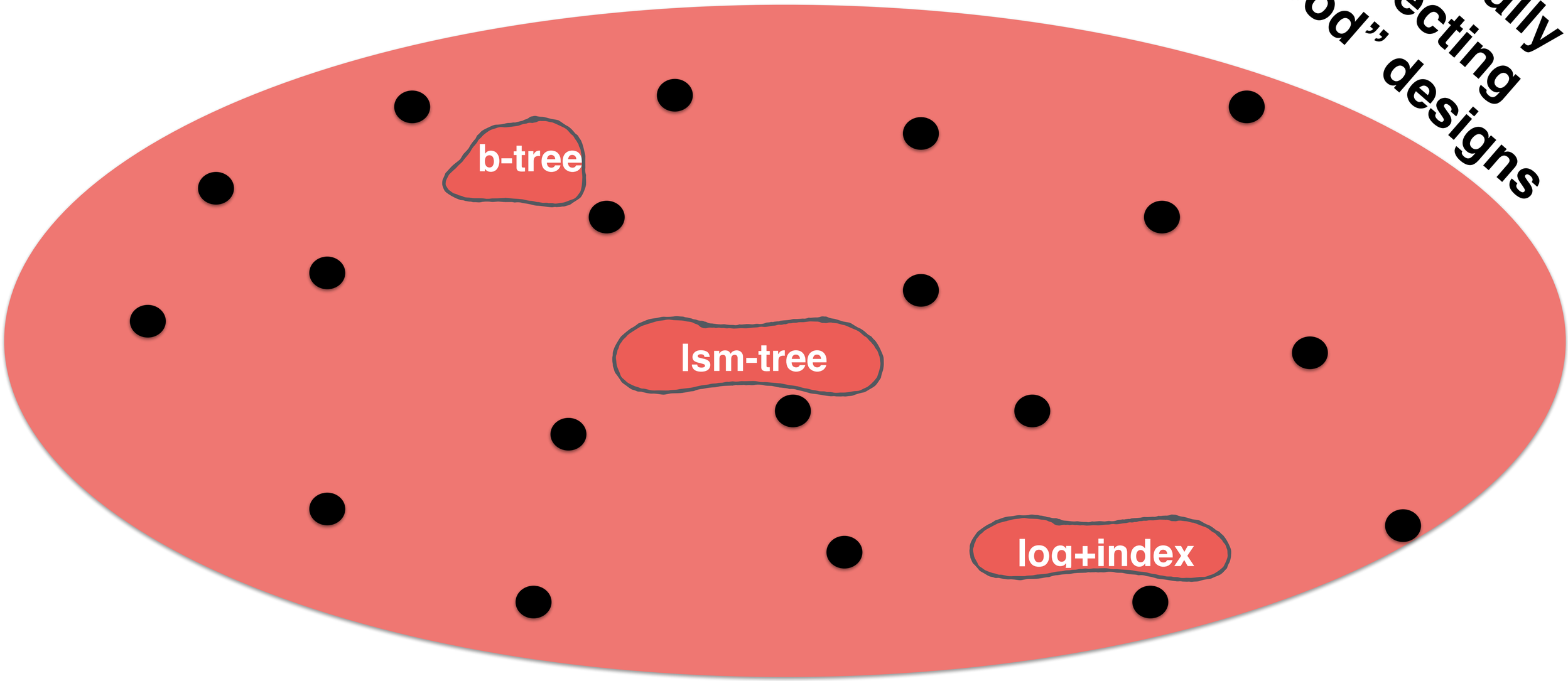
The design space of systems is even larger

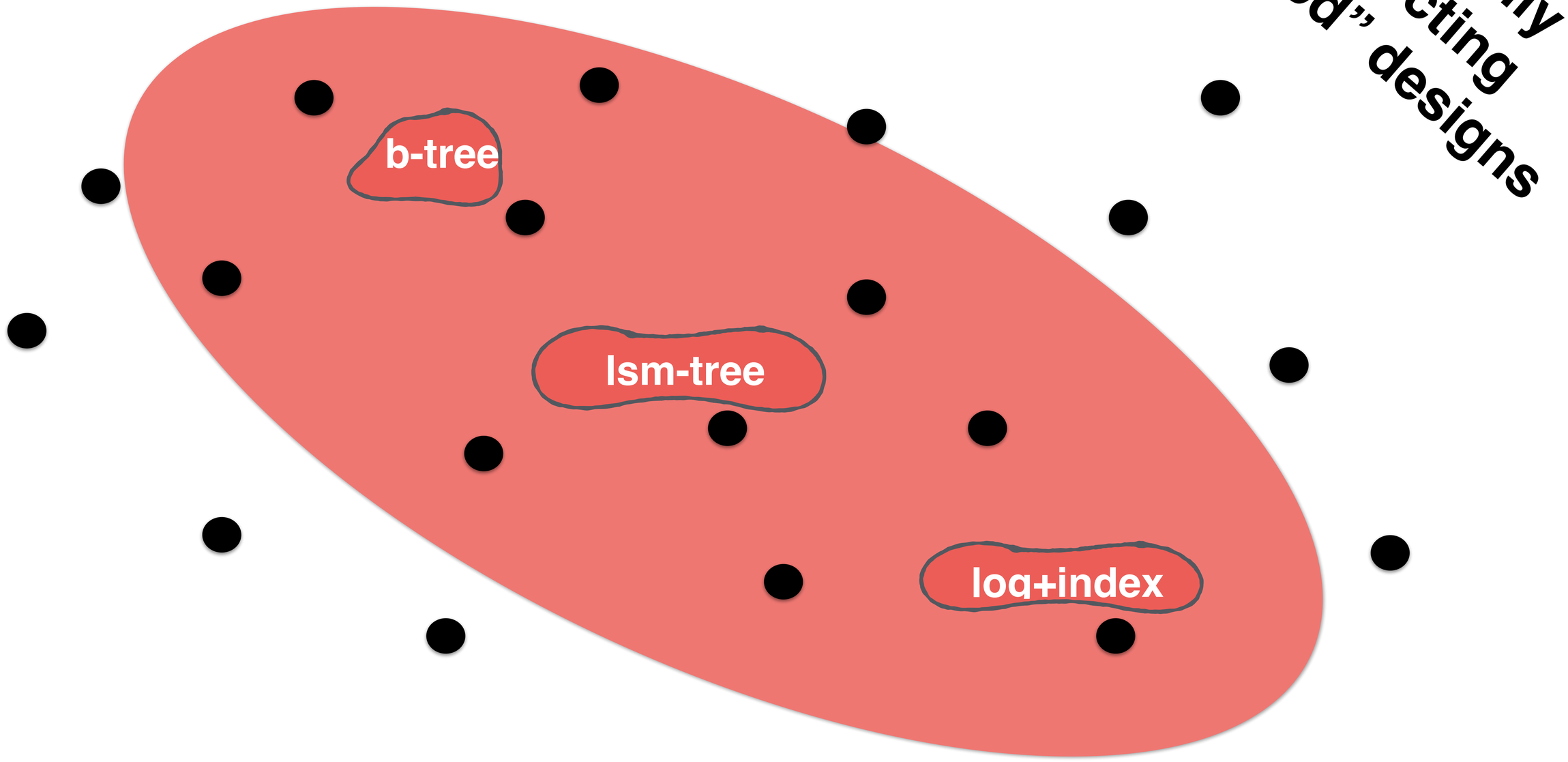



manually
selecting
“good” designs



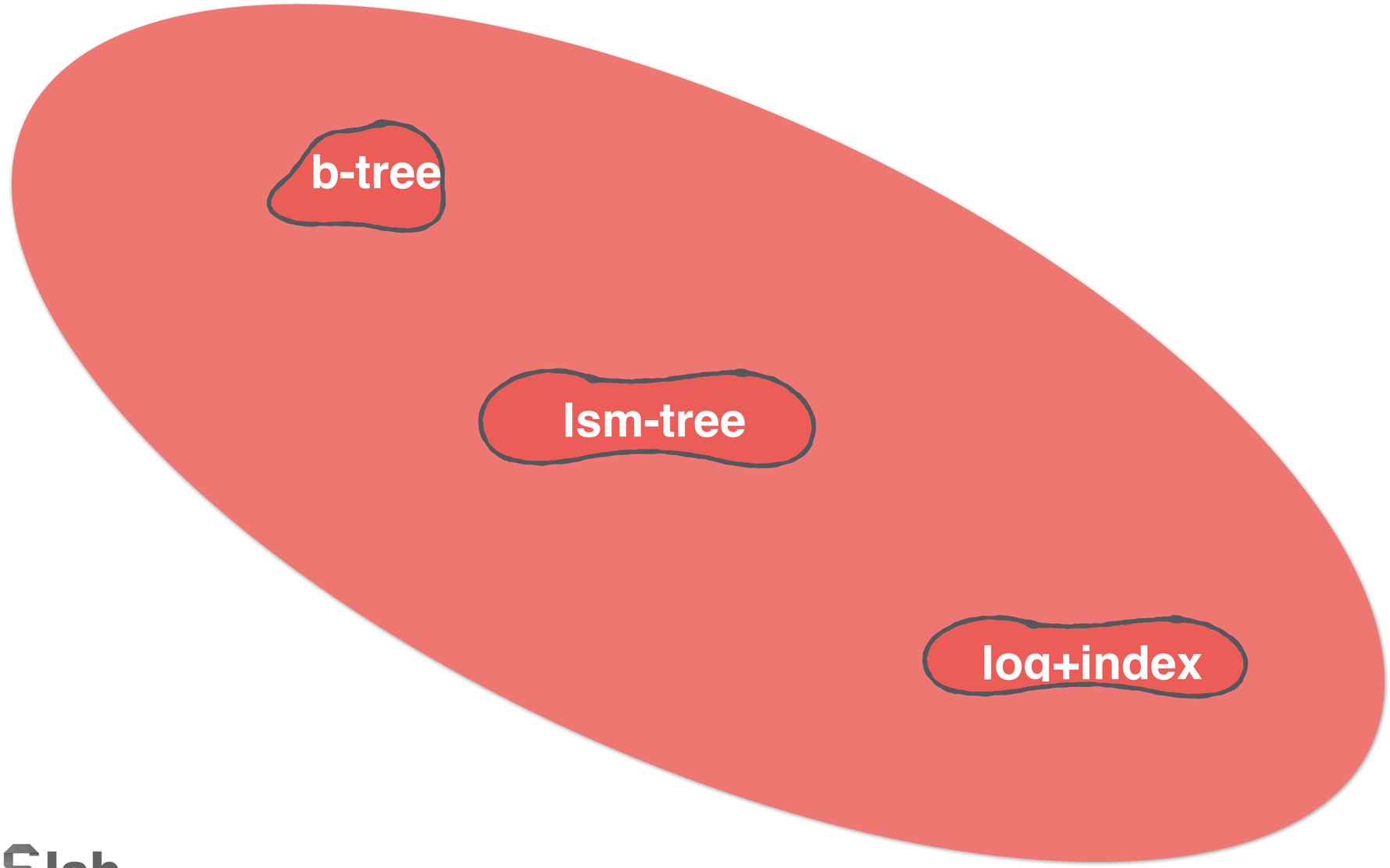
DASlab @ Harvard SEAS

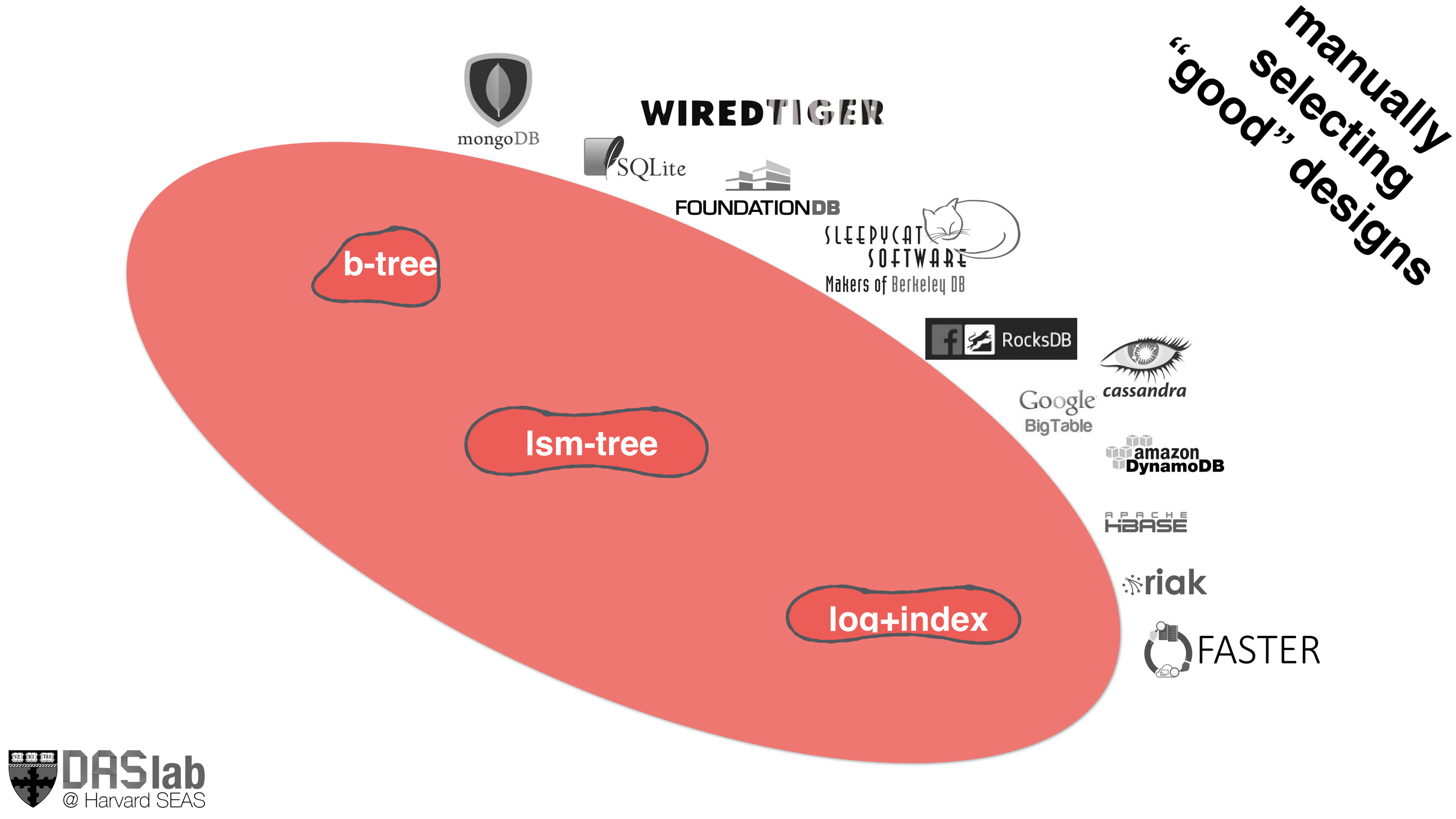

manually selecting "good" designs
b-tree
lsm-tree
log+index



manually selecting "good" designs

manually selecting "good" designs




manually
selecting
“good” designs
mongoDB
WIREDTIGER
SQLite
FOUNDATIONDB
SLEEPYCAT
SOFTWARE
Makers of Berkeley DB
b-tree
RocksDB
cassandra
Google
BigTable
lsm-tree
amazon
DynamoDB
APACHE
HBASE
riak
log+index
FASTER
DASlab
@ Harvard SEAS

manually selecting "good" designs

mongoDB

WIREDTIGER

SQLite

FOUNDATIONDB

SLEEPYCAT SOFTWARE
Makers of Berkeley DB

b-tree

RocksDB

cassandra

Google BigTable

lsm-tree

amazon DynamoDB

APACHE HBASE

riak

log+index

FASTER

DASlab
@ Harvard SEAS

manually selecting "good" designs

mongoDB

WIREDTIGER

SQLite

FOUNDATIONDB

SLEEPYCAT SOFTWARE
Makers of Berkeley DB

b-tree

RocksDB

cassandra

Google BigTable

lsm-tree

amazon DynamoDB

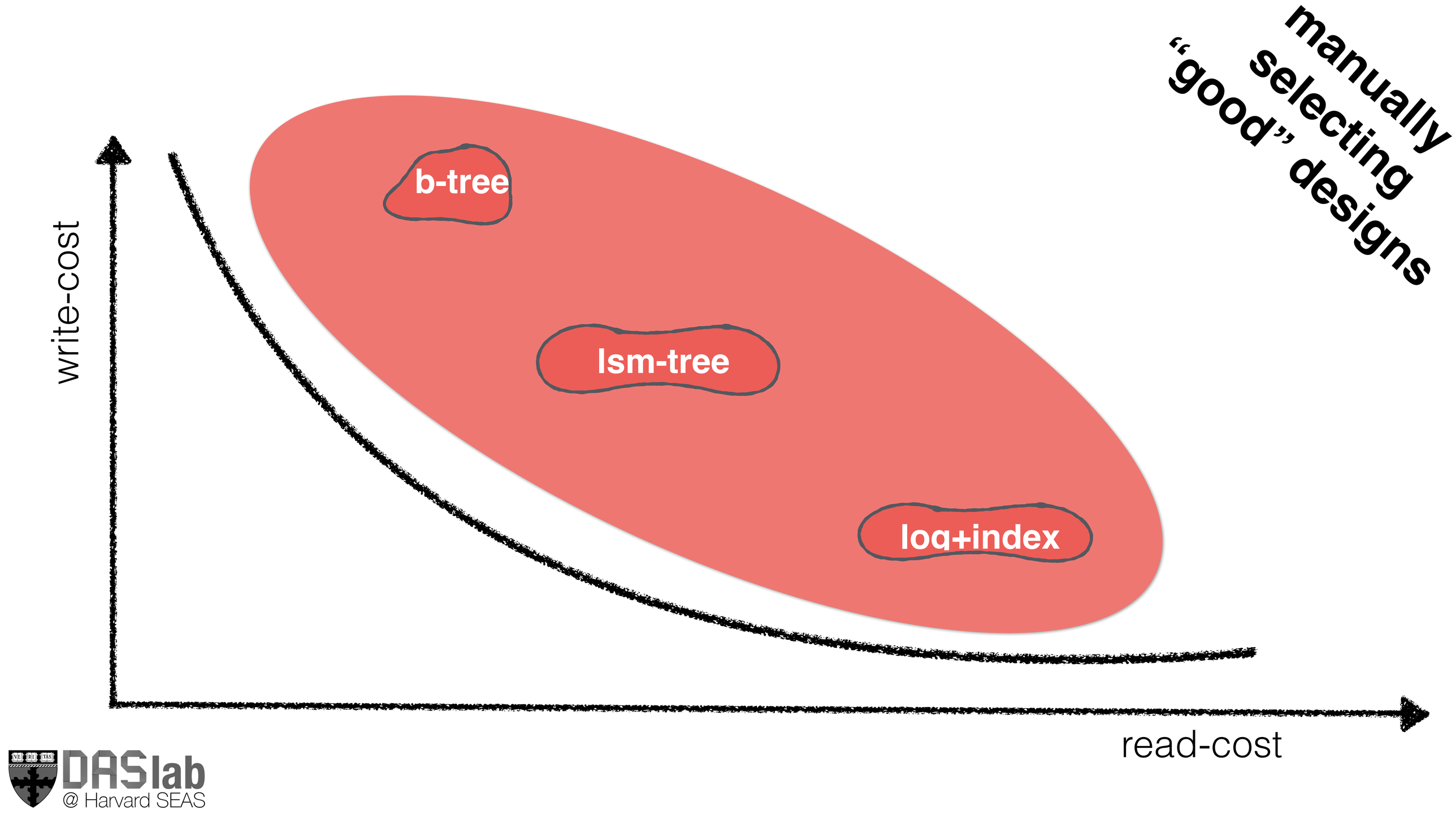
APACHE HBASE

riak

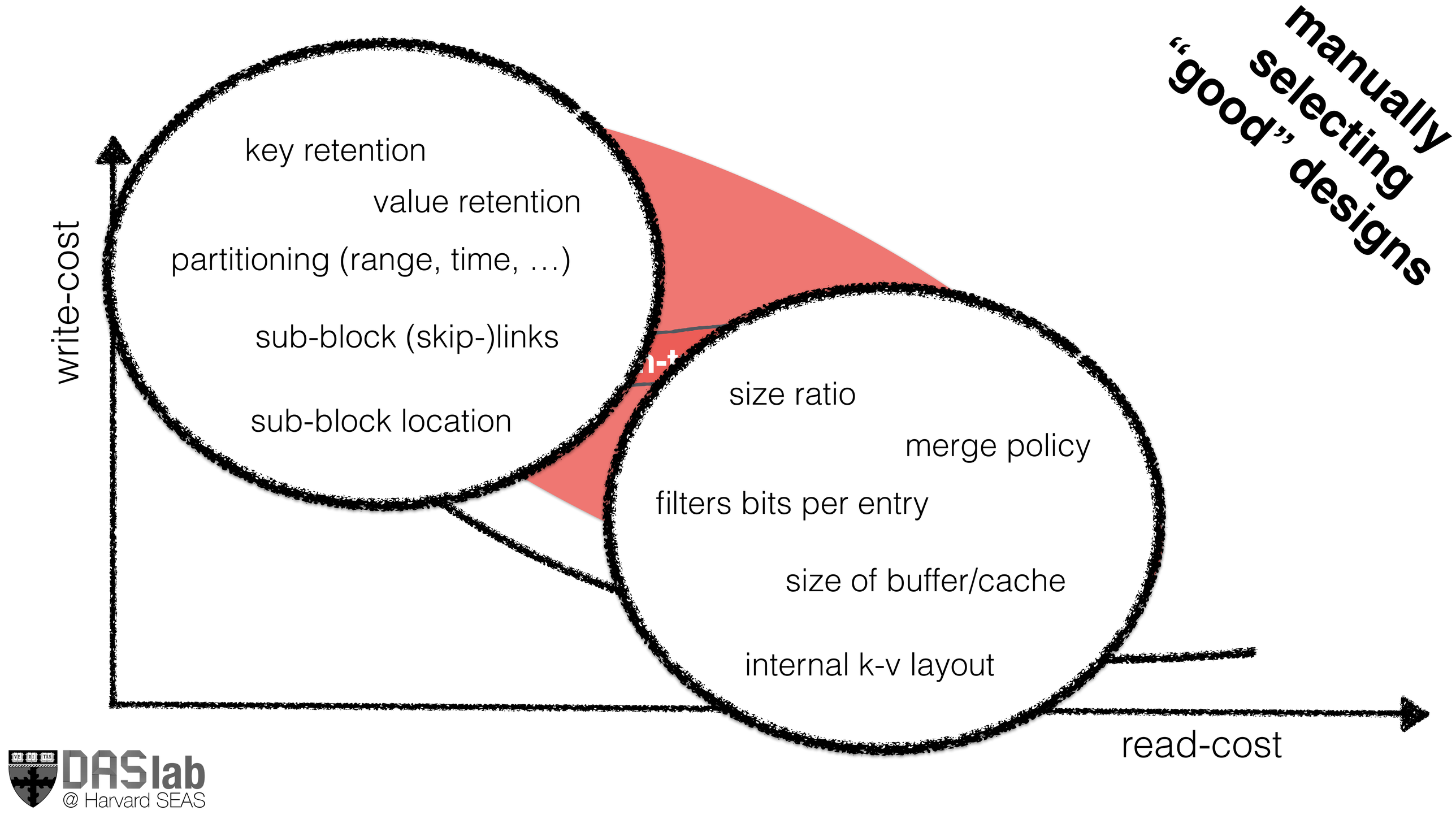
log+index

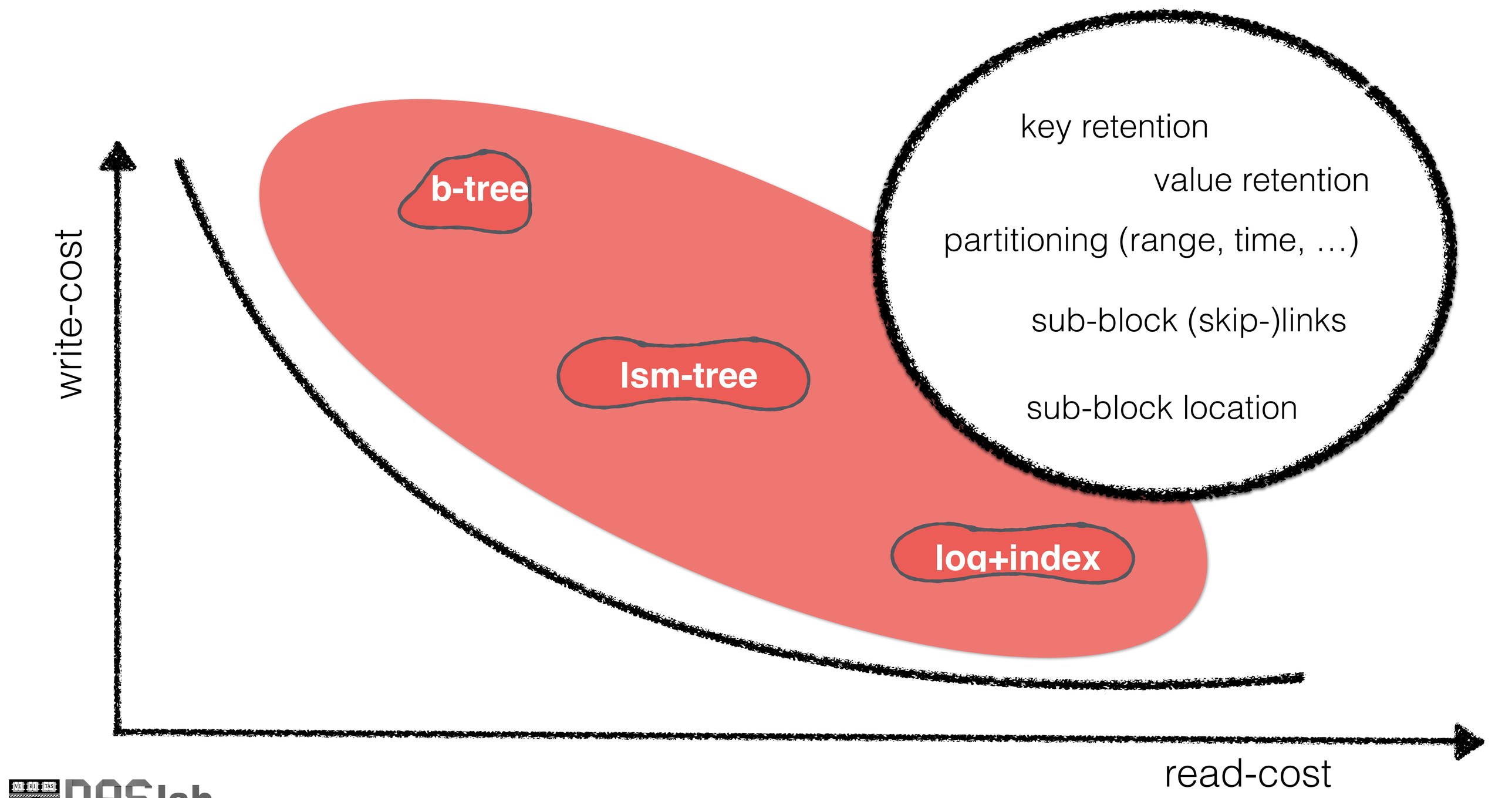
FASTER

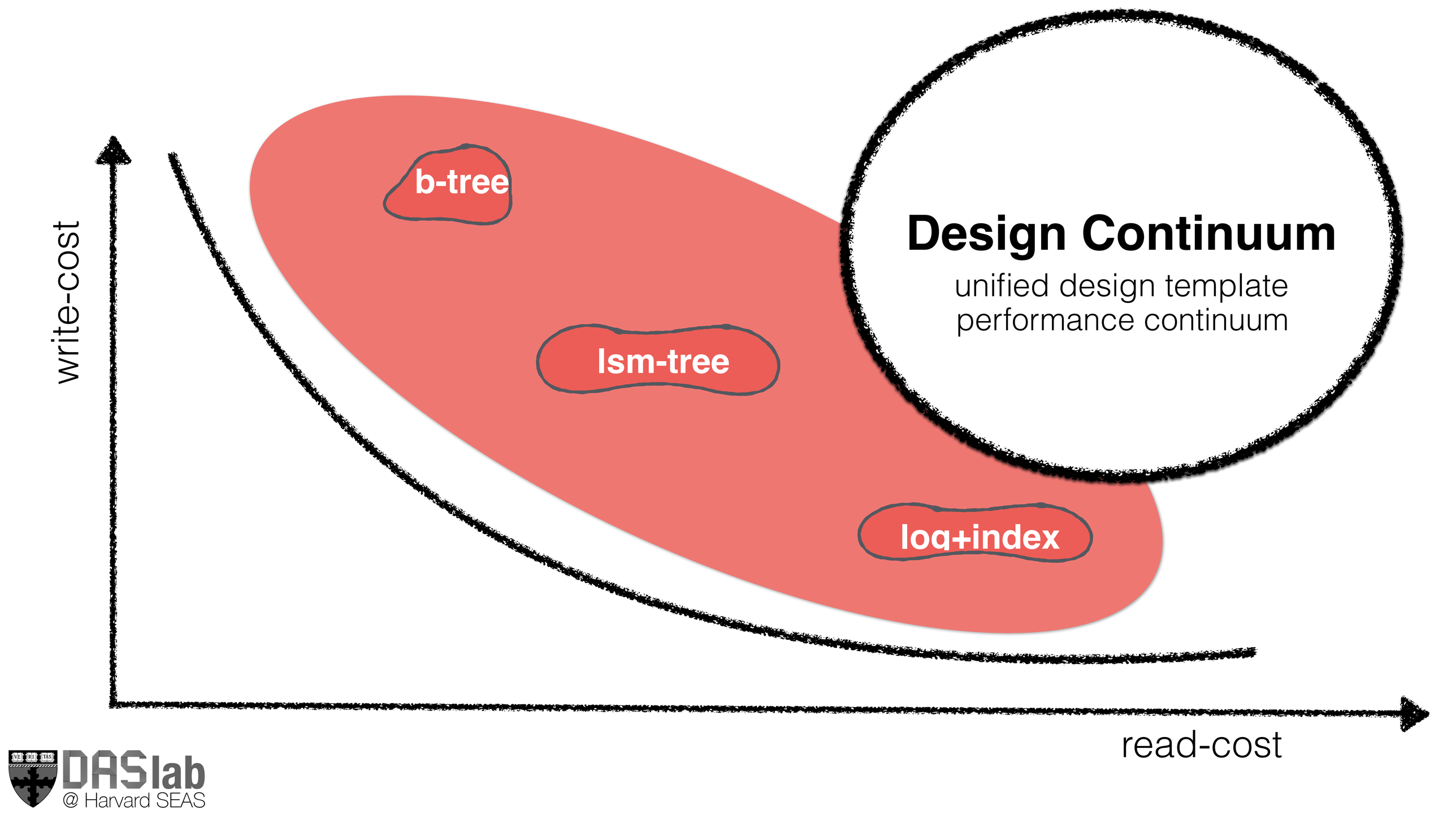
DASlab
@ Harvard SEAS



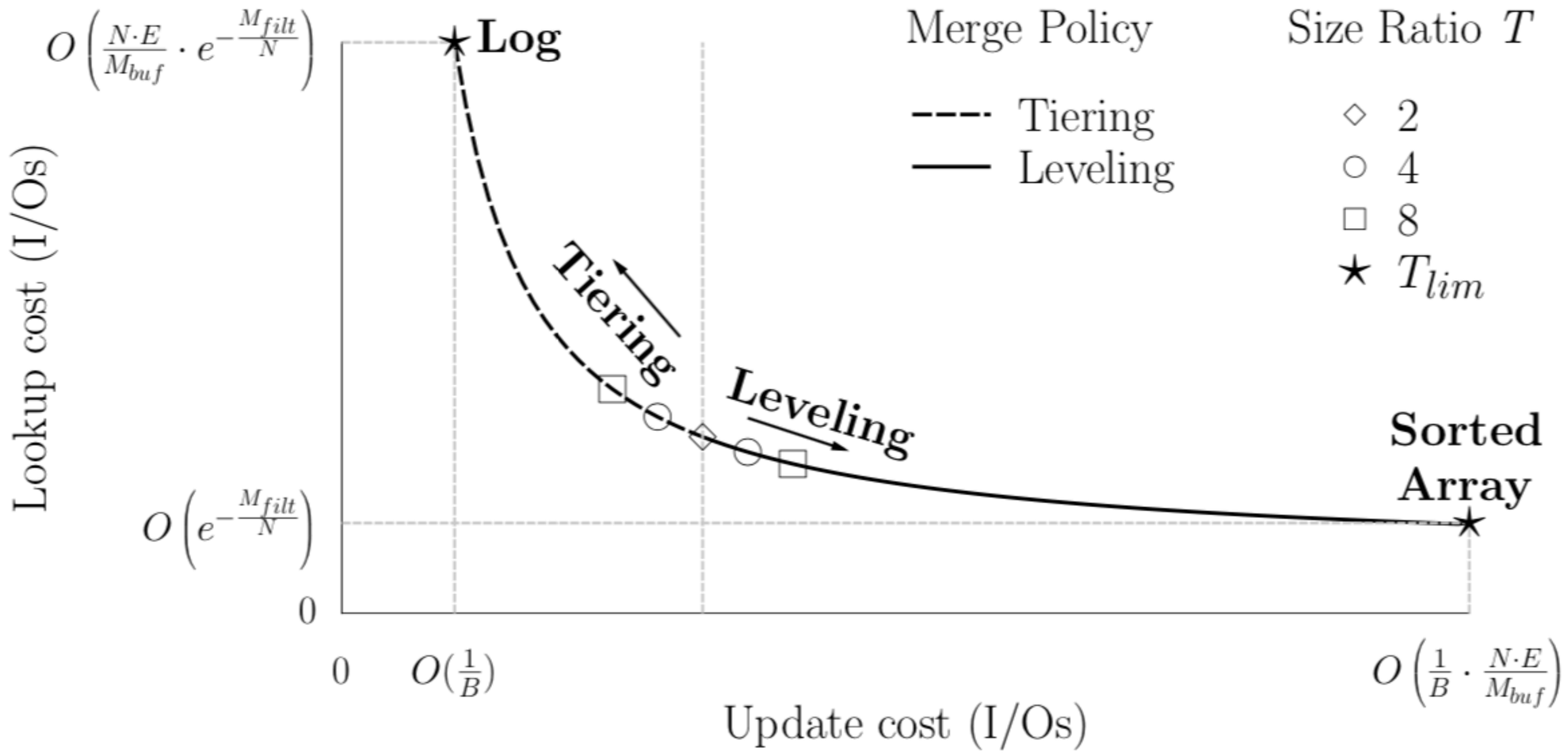
DASlab
@ Harvard SEAS


manually selecting "good" designs
write-cost
key retention
value retention
partitioning (range, time, …)
sub-block (skip-)links
sub-block location
size ratio
merge policy
filters bits per entry
size of buffer/cache
internal k-v layout
read-cost


b-tree
lsm-tree
log+index
key retention
value retention
partitioning (range, time, …)
sub-block (skip-)links
sub-block location
write-cost
read-cost

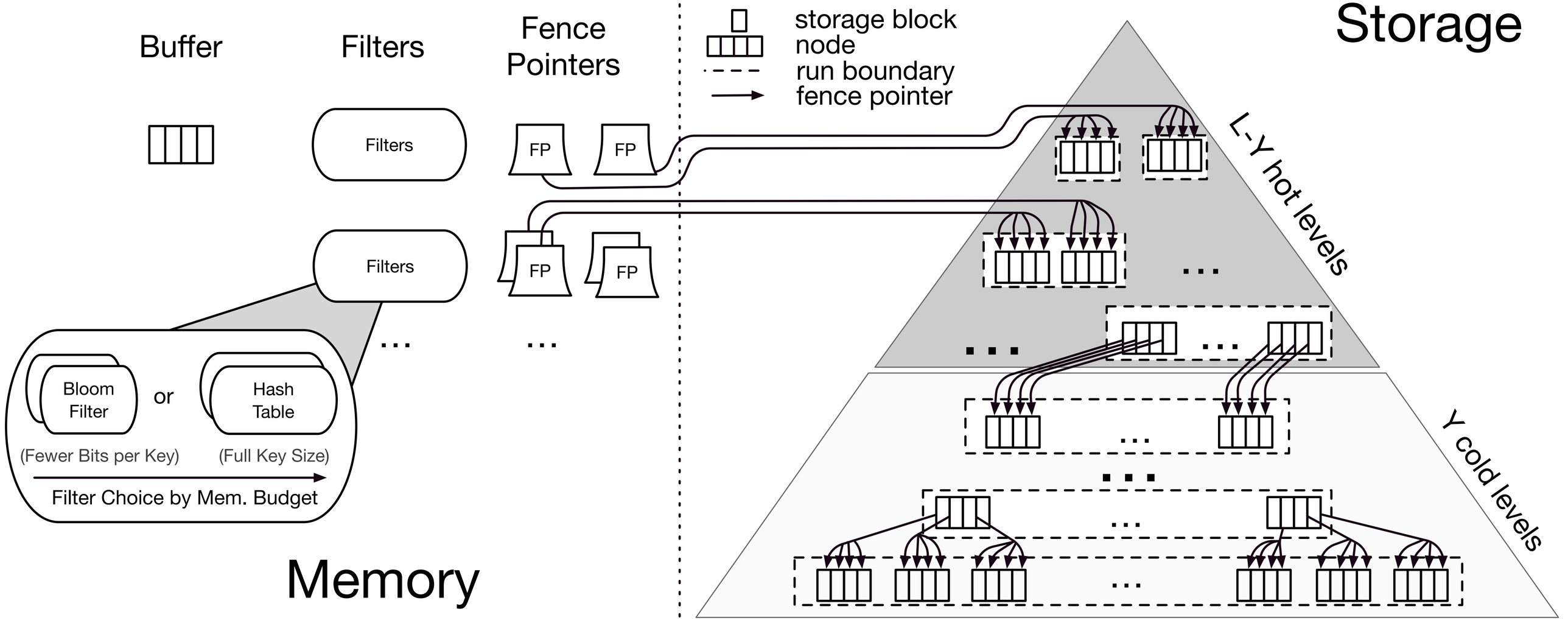


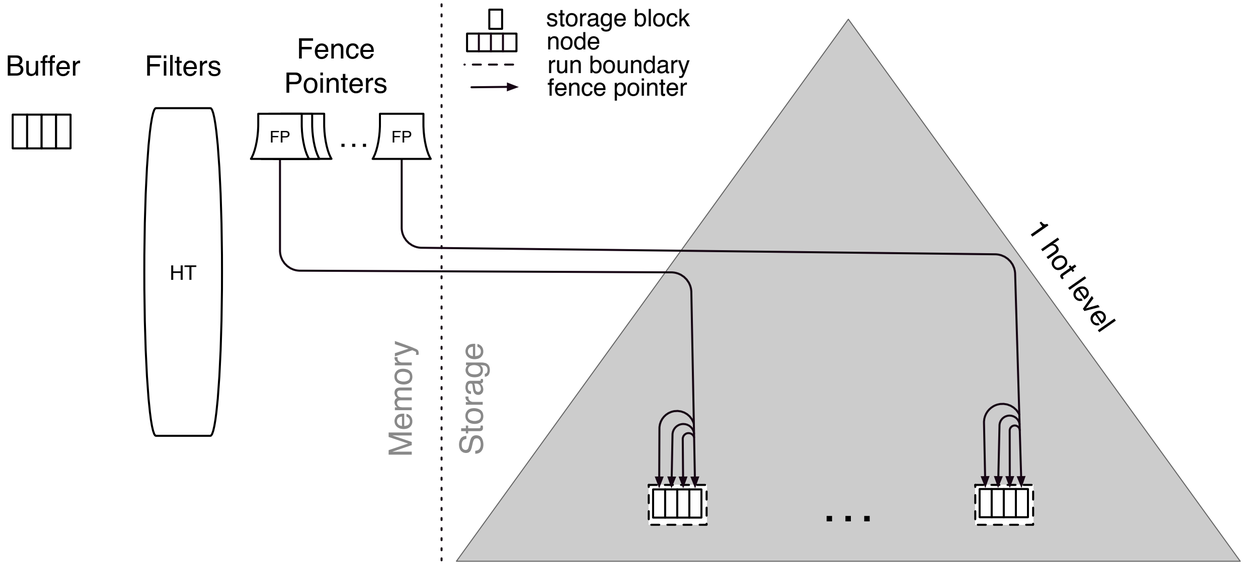
DASlab
@ Harvard SEAS


$O\left(\frac{N \cdot E}{M_{buf}} \cdot e^{-\frac{M_{filt}}{N}}\right)$
Log
Merge Policy
Size Ratio T
Tiering
Leveling
2
4
8
T_{lim}
Lookup cost (I/Os)
Tiering
Leveling
Sorted Array
$O\left(e^{-\frac{M_{filt}}{N}}\right)$
0
0
$O(\frac{1}{B})$
$O\left(\frac{1}{B} \cdot \frac{N \cdot E}{M_{buf}}\right)$
Update cost (I/Os)

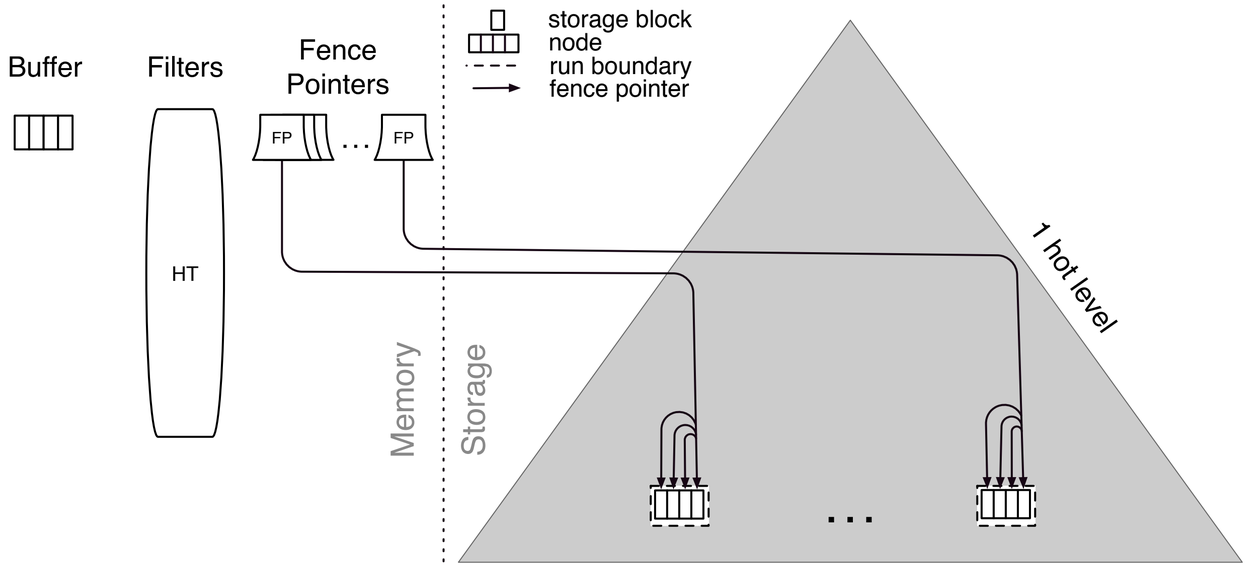
a unified design space

[log, log+hash, LSM-tree*, B$^{\varepsilon}$Tree, B-Tree, Sorted Array]

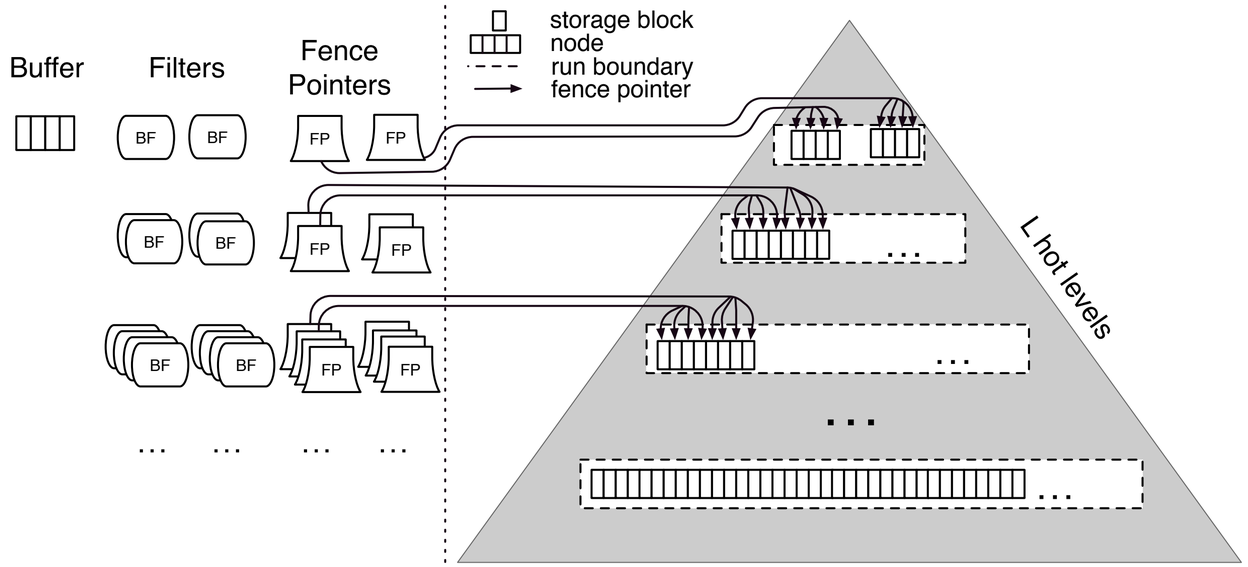



Buffer
Filters
Fence
Pointers
storage block
node
run boundary
fence pointer
FP
FP
HT
Memory
Storage
1 hot level

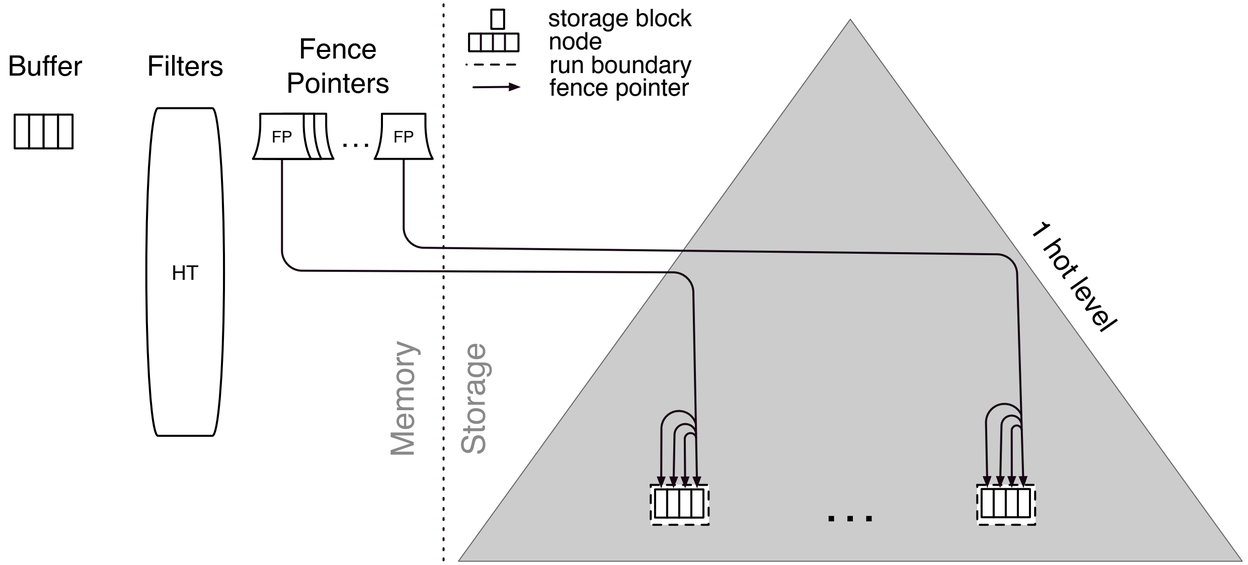
LSH-Table



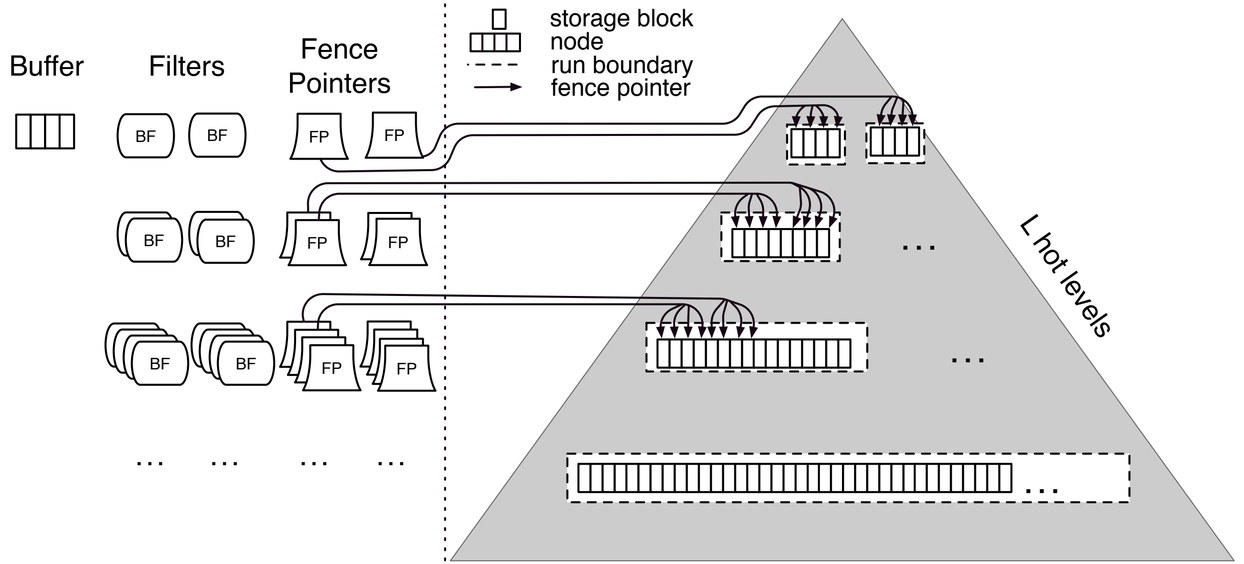
LSH-Table



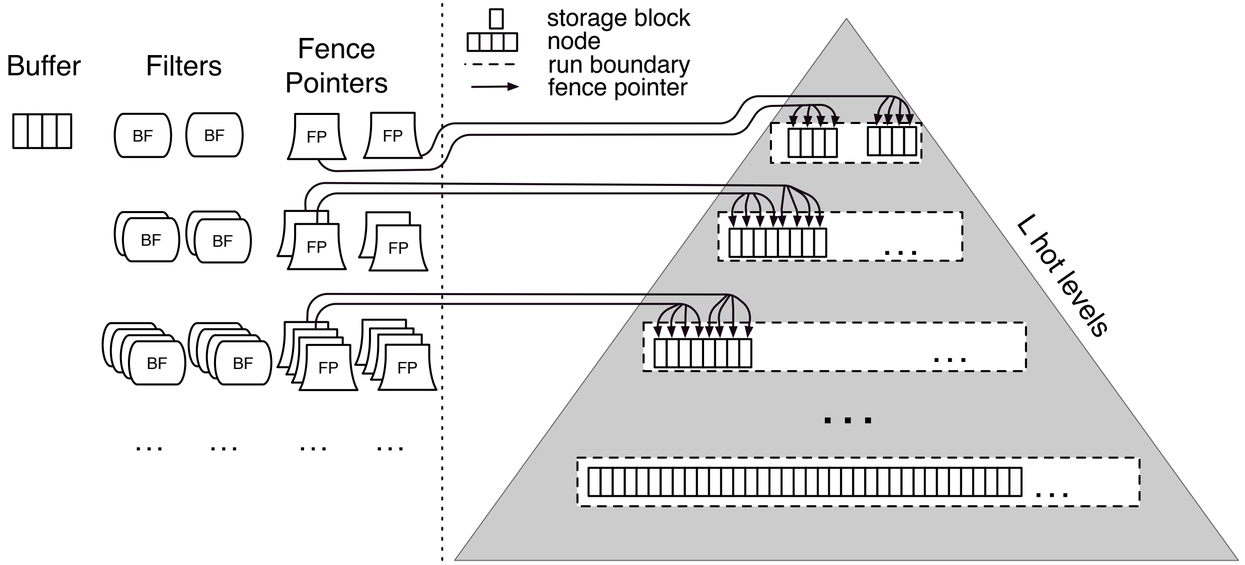
Leveled LSM-Tree



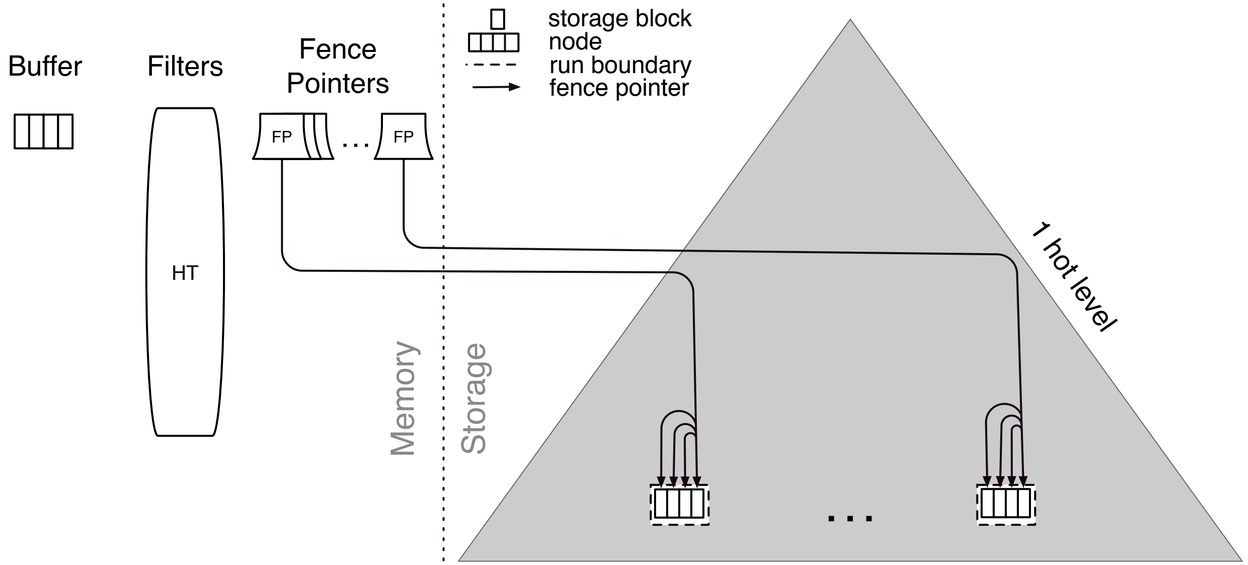
LSH-Table



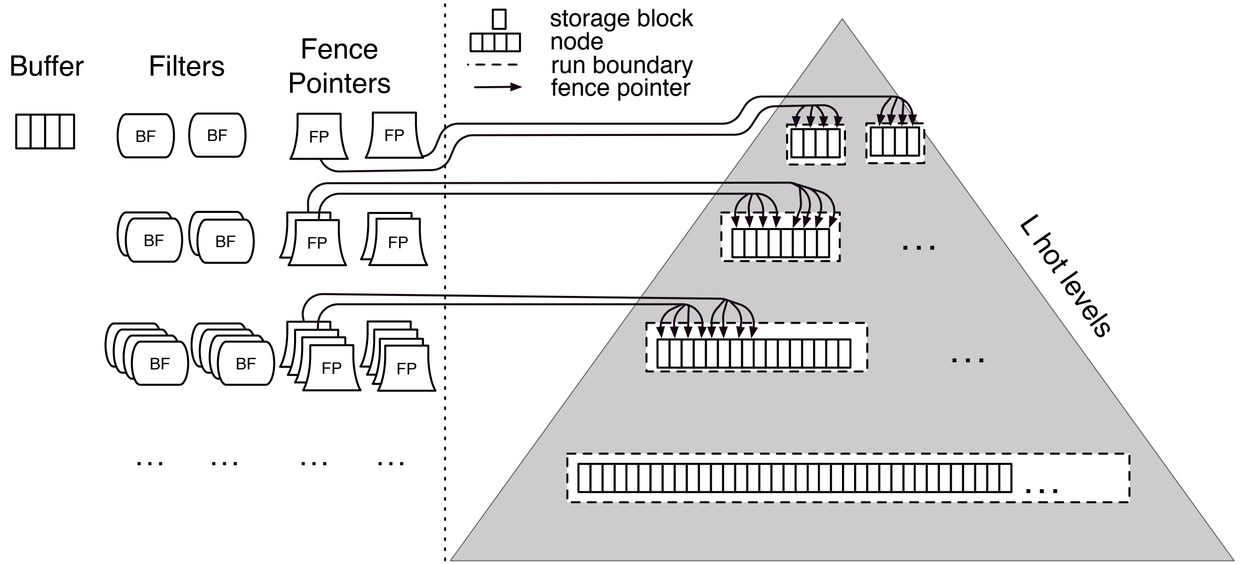
Lazy Leveled LSM-Tree



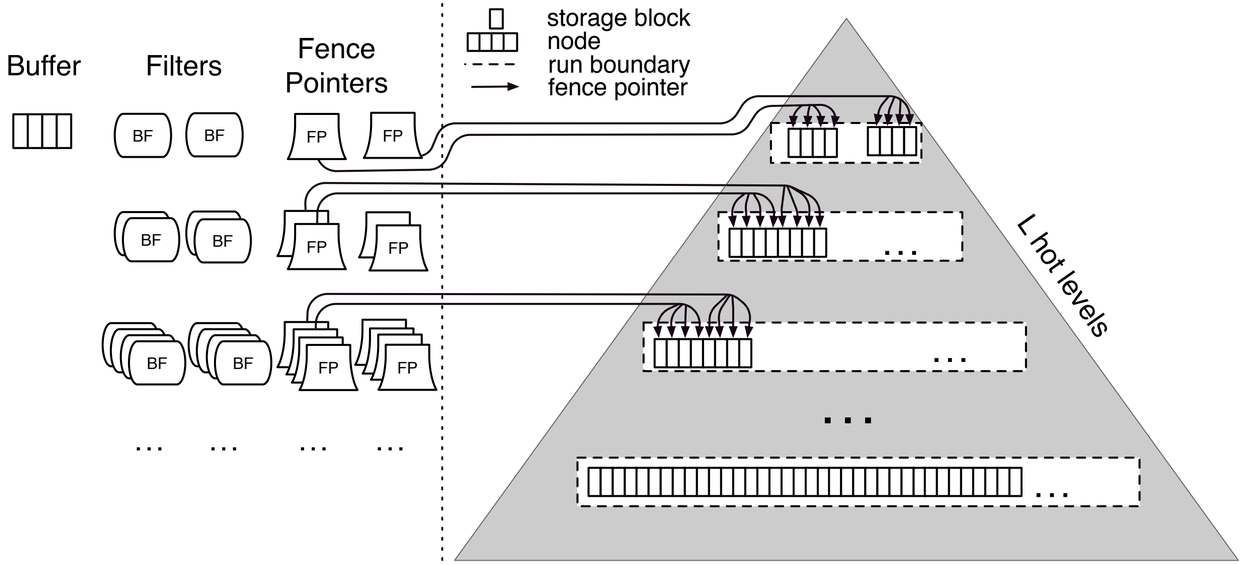
Leveled LSM-Tree



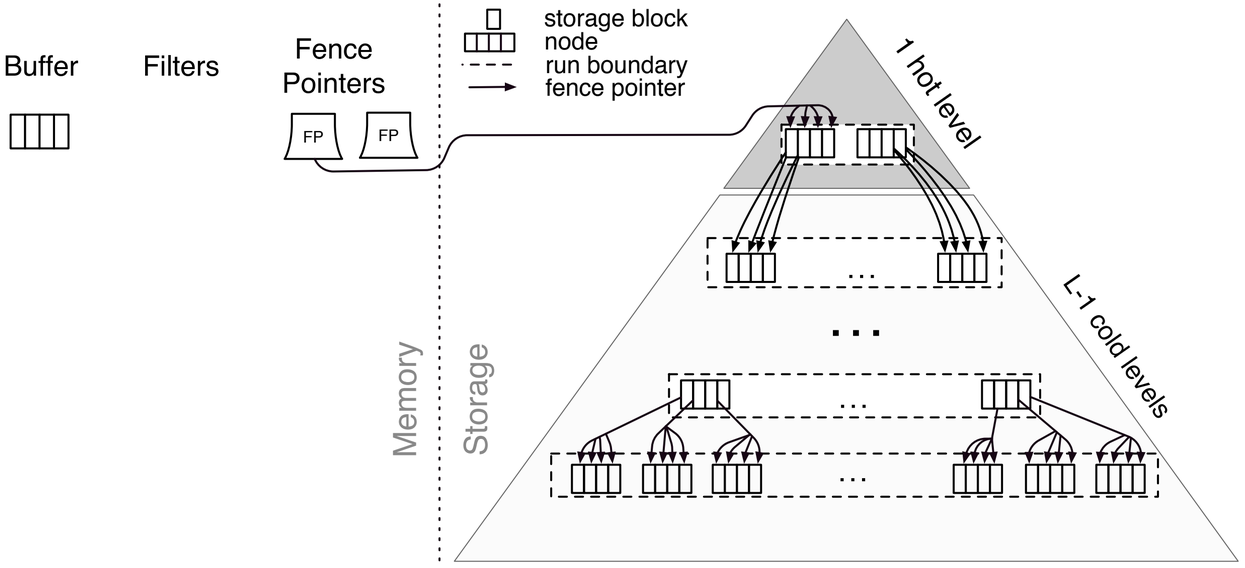
LSH-Table



Lazy Leveled LSM-Tree



Leveled LSM-Tree



Bε Tree

components

Environment Parameters

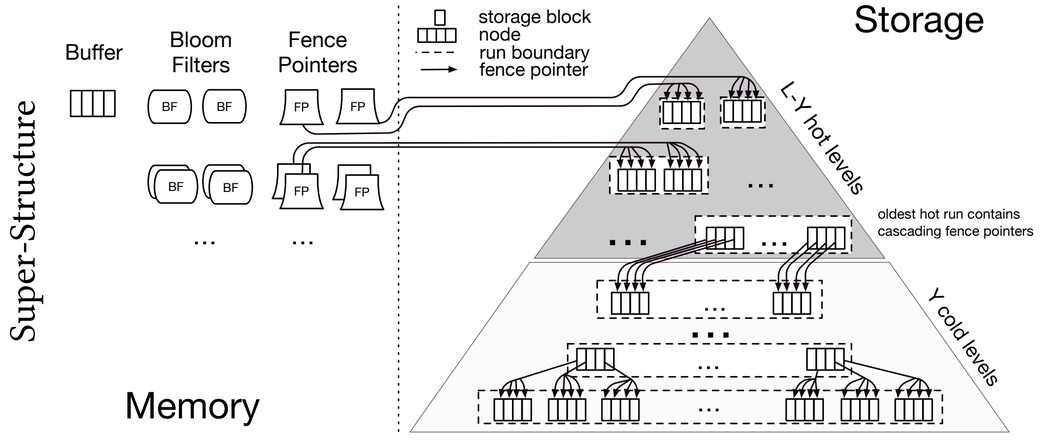
Term	Name	Description	Min. Value	Max. Value	Units
B	Block Size	# data entries that fit in a storage block.			Entries
M	Memory	Total main memory budget.	$B \cdot E + \frac{F \cdot T \cdot M_B}{E \cdot B}$	$N \cdot E$	Bits
N	Dataset Size	# total data entries in the dataset.			Entries
E	Entry Size	Size of an entry.			Bits
F	Key Size	Size of a key, also used to approximate size of a fence (fence key and pointer).			Bits
s	Avg. Selectivity	Average selectivity of a long range query.			Entries

Design Parameters

Term	Name	Description	Min. Value	Max. Value	Units
T	Growth Factor	Capacity ratio between adjacent levels.	2	B	Ratio
K	Hot Merge Threshold	Maximum # runs per hot level.	1	$T-1$	Runs
Z	Cold Merge Threshold	Maximum # runs per cold level.	1	$T-1$	Runs
D	Max. Node Size	Maximum size of a node; defines a contiguous data region.	1	$\frac{N}{B}$	Blocks
M_F	Fence & Filter Memory Budget	# bits of main memory budgeted to fence pointers and filters.	$\frac{F \cdot T \cdot M_B}{E \cdot B}$	M	Bits

Derived Term	Expression	Units
L (# total levels)	$\lceil \log_T \frac{N \cdot E}{M_B} \rceil$	Levels
X (Filters Memory Threshold)	$\frac{1}{\ln 2^2} \cdot (\frac{\ln T}{T-1} + \frac{\ln K - \ln Z}{T})$	Bits per Entry
$M_{F_{HI}}$ (M_F Threshold: Hot Levels Saturation)	$N \cdot (\frac{X}{T} + \frac{F}{B})$	Bits
$M_{F_{LO}}$ (M_F Threshold: Cold Levels Saturation)	$\frac{M_B \cdot F \cdot T}{E \cdot B}$	Bits
Y (# Cold Levels)	$\begin{cases} 0 & \text{if } M_F \geq M_{F_{HI}} \\ \lfloor \log_T \frac{N}{M_F} \cdot (\frac{X}{T} + \frac{F}{B}) \rfloor & \text{if } M_{F_{LO}} < M_F < M_{F_{HI}} \\ L-1 & \text{if } M_F = M_{F_{LO}} \end{cases}$	Levels
M_{FP} (Fence Pointer Memory Budget)	$T^{L-Y+1} \cdot F \cdot \frac{M_b}{E \cdot B} \cdot \frac{T}{T-1}$	Bits
M_{BF} (Filter Memory Budget)	$M_F - M_{FP}$	Bits
M_B (Buffer Memory Budget)	$B \cdot E + (M - M_F)$	Bits
p_{sum} (Sum of BF False Positive Rates)	$e^{-\frac{M_{BF}}{N} \cdot \ln(2)^2 \cdot T^Y} \cdot Z^{\frac{T-1}{T}} \cdot K^{\frac{1}{T}} \cdot \frac{T^{\frac{T}{T-1}}}{T-1}$	
p_i (BF False Positive Rate at Level i)	$\begin{cases} 1 & \text{if } i > L-Y \\ \frac{p_{sum}}{Z} \cdot \frac{T-1}{T} & \text{if } i = L-Y \\ \frac{p_{sum}}{K} \cdot \frac{T-1}{T} \cdot \frac{1}{T^{L-Y-i}} & \text{if } i < L-Y \end{cases}$	Probability

Derived Design Rules



General Cost Model

Operation	Cost Expression (I/O)
Update	$O(\frac{1}{B} \cdot (\frac{T}{K} \cdot (L-Y-1) + \frac{T}{Z} \cdot (Y+1)))$
Zero Result Lookup	$O(Z \cdot e^{-\frac{M_{BF}}{N} \cdot T^Y} + Y \cdot Z)$
Single Result Lookup	$O(1 + Z \cdot e^{-\frac{M_{BF}}{N} \cdot T^Y} + Y \cdot Z)$
Short Scan	$O(K \cdot (L-Y-1) + Z \cdot (Y+1))$
Long Scan	$O(\frac{s \cdot Z}{B})$

DASlab @ Harvard SEAS

components

environment

design

derived

unified

I/O

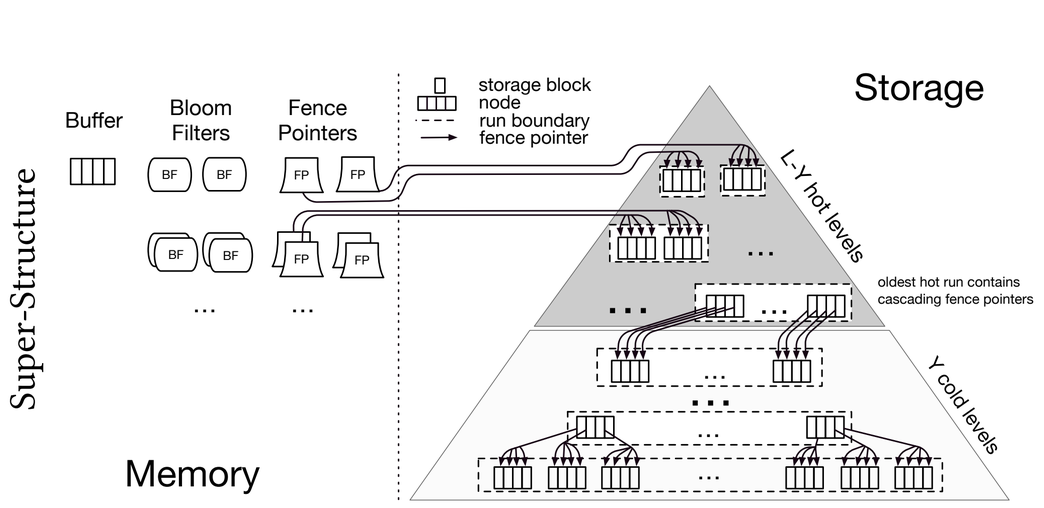
models

Environment Parameters

Term	Name	Description	Min. Value	Max. Value	Units
B	Block Size	# data entries that fit in a storage block.			Entries
M	Memory	Total main memory budget.	$B \cdot E + \frac{F \cdot T \cdot M_B}{E \cdot B}$	$N \cdot E$	Bits
N	Dataset Size	# total data entries in the dataset.			Entries
E	Entry Size	Size of an entry.			Bits
F	Key Size	Size of a key, also used to approximate size of a fence (fence key and pointer).			Bits
s	Avg. Selectivity	Average selectivity of a long range query.			Entries

Design Parameters

Term	Name	Description	Min. Value	Max. Value	Units
T	Growth Factor	Capacity ratio between adjacent levels.	2	B	Ratio
K	Hot Merge Threshold	Maximum # runs per hot level.	1	$T-1$	Runs
Z	Cold Merge Threshold	Maximum # runs in cold levels.	1	$T-1$	Runs
D	Max. Node Size	Maximum size of a node; a run is a contiguous data region.	1	$\frac{N}{B}$	Blocks
M_F	Fence & Filter Memory Budget	# bits of main memory budgeted to fence pointers and filters.	$\frac{F \cdot T \cdot M_B}{E \cdot B}$	M	Bits



Derived Term	Expression	Units
L (# total levels)	$\lceil \log_T \frac{N \cdot E}{M_B} \rceil$	Levels
X (Filters Memory Threshold)	$\frac{1}{\ln 2^2} \cdot (\frac{\ln T}{T-1} + \frac{\ln K - \ln Z}{T})$	Bits per Entry
$M_{F_{HI}}$ (M_F Threshold: Hot Levels Saturation)	$N \cdot (\frac{X}{T} + \frac{F}{B})$	Bits
$M_{F_{LO}}$ (M_F Threshold: Cold Levels Saturation)	$M_{F_{HI}} - N \cdot \frac{T-1}{T} \cdot \frac{1}{B}$	Bits
Y (# Cold Levels)	$\begin{cases} 1 & \text{if } M_F \geq M_{F_{HI}} \\ \lfloor \log_T \frac{N}{M_F} \cdot (\frac{X}{T} + \frac{F}{B}) \rfloor & \text{if } M_{F_{LO}} < M_F < M_{F_{HI}} \\ L-1 & \text{if } M_F = M_{F_{LO}} \end{cases}$	Levels
M_{FP} (Fence Pointer Memory Budget)	$T^{L-Y+1} \cdot F \cdot \frac{M_b}{E \cdot B} \cdot \frac{T}{T-1}$	Bits
M_{BF} (Filter Memory Budget)	$M_F - M_{FP}$	Bits
M_B (Buffer Memory Budget)	$B \cdot E + (M - M_F)$	Bits
p_{sum} (Sum of BF False Positive Rates)	$e^{-\frac{M_{BF}}{N} \cdot \ln(2)^2 \cdot T^Y} \cdot Z^{\frac{T-1}{T}} \cdot K^{\frac{1}{T}} \cdot \frac{T^{\frac{T}{T-1}}}{T-1}$	
p_i (BF False Positive Rate at Level i)	$\begin{cases} 1 & \text{if } i > L-Y \\ \frac{p_{sum}}{Z} \cdot \frac{T-1}{T} & \text{if } i = L-Y \\ \frac{p_{sum}}{K} \cdot \frac{T-1}{T} \cdot \frac{1}{T^{L-Y-i}} & \text{if } i < L-Y \end{cases}$	Probability

Derived Design Rules

General Cost Model

Operation	Cost Expression (I/O)
Update	$O(\frac{1}{B} \cdot (\frac{T}{K} \cdot (L-Y-1) + \frac{T}{Z} \cdot (Y+1)))$
Zero Result Lookup	$O(Z \cdot e^{-\frac{M_{BF}}{N} \cdot T^Y} + Y \cdot Z)$
Single Result Lookup	$O(1 + Z \cdot e^{-\frac{M_{BF}}{N} \cdot T^Y} + Y \cdot Z)$
Short Scan	$O(K \cdot (L-Y-1) + Z \cdot (Y+1))$
Long Scan	$O(\frac{s}{B})$

DASlab @ Harvard SEAS

components

environment

Environment Parameters

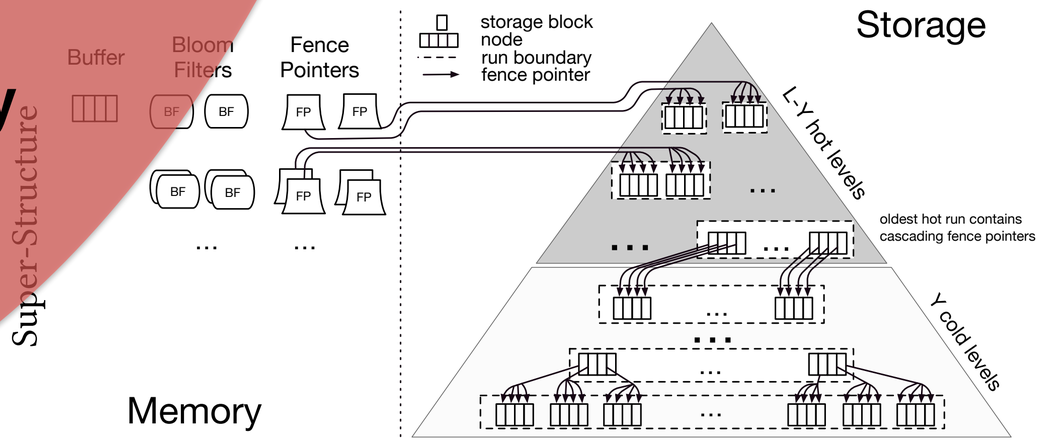
Term	Name	Description	Min. Value	Max. Value	Units
B	Block Size	# data entries that fit in a storage block.			Entries
M	Memory	Total main memory budget.	$B \cdot E + \frac{F \cdot T \cdot M_B}{E \cdot B}$	$N \cdot E$	Bits
N	Dataset Size	# total data entries in the dataset.			Entries
E	Entry Size	Size of an entry.			Bits
F	Key Size	Size of a key, also used to approximate size of a fence (fence key and pointer).			Bits
s	Avg. Selectivity	Average selectivity of a long range query.			Entries

design

Design Parameters

Term	Name	Description	Min. Value	Max. Value	Units
T	Growth Factor	Capacity ratio between adjacent levels.	2	B	Ratio
K	Hot Merge Threshold	Maximum # runs per hot level.	1	$T-1$	Runs
Z	Cold Merge Threshold	Maximum # runs in cold levels.	1	$T-1$	Runs
D	Max. Node Size	Maximum size of a node; a node is a contiguous data region.	1	$\frac{N}{B}$	Blocks
M_F	Fence & Filter Memory Budget	# bits of main memory budgeted to fence pointers and filters.	$\frac{F \cdot T \cdot M_B}{E \cdot B}$	M	Bits

- **growth factor**
- **hot merge threshold**
- **cold merge threshold**
- **fence/filter memory**
- **max node size**



derived

Derived Term	Expression	Units
L (# total levels)	$\lceil \log_T \frac{N \cdot E}{M_B} \rceil$	Levels
X (Filters Memory Threshold)	$\frac{1}{\ln 2^2} \cdot (\frac{\ln T}{T-1} + \frac{\ln K - \ln Z}{T})$	Bits per Entry
$M_{F_{HI}}$ (M_F Threshold: Hot Levels Saturation)	$N \cdot (\frac{X}{T} + \frac{F}{B})$	Bits
$M_{F_{LO}}$ (M_F Threshold: Cold Levels Saturation)	$M_{F_{HI}} \cdot \frac{T-1}{T}$	Bits
Y (# Cold Levels)	$\begin{cases} 1 & \text{if } M_F \geq M_{F_{HI}} \\ \lfloor \log_T \frac{N}{M_F} \cdot (\frac{X}{T} + \frac{F}{B}) \rfloor & \text{if } M_{F_{LO}} < M_F < M_{F_{HI}} \\ L-1 & \text{if } M_F = M_{F_{LO}} \end{cases}$	Levels
M_{FP} (Fence Pointer Memory Budget)	$T^{L-Y+1} \cdot F \cdot \frac{M_b}{E \cdot B} \cdot \frac{T}{T-1}$	Bits
M_{BF} (Filter Memory Budget)	$M_F - M_{FP}$	Bits
M_B (Buffer Memory Budget)	$B \cdot E + (M - M_F)$	Bits
p_{sum} (Sum of BF False Positive Rates)	$e^{-\frac{M_{BF}}{N} \cdot \ln(2)^2 \cdot T^Y} \cdot Z^{\frac{T-1}{T}} \cdot K^{\frac{1}{T}} \cdot \frac{T^{\frac{T}{T-1}}}{T-1}$	
p_i (BF False Positive Rate at Level i)	$\begin{cases} 1 & \text{if } i > L-Y \\ \frac{p_{sum}}{Z} \cdot \frac{T-1}{T} & \text{if } i = L-Y \\ \frac{p_{sum}}{K} \cdot \frac{T-1}{T} \cdot \frac{1}{T^{L-Y-i}} & \text{if } i < L-Y \end{cases}$	Probability

Derived Design Rules

unified I/O models

General Cost Model

Operation	Cost Expression (I/O)
Update	$O(\frac{1}{B} \cdot (\frac{T}{K} \cdot (L-Y-1) + \frac{T}{Z} \cdot (Y+1)))$
Zero Result Lookup	$O(Z \cdot e^{-\frac{M_{BF}}{N} \cdot T^Y} + Y \cdot Z)$
Single Result Lookup	$O(1 + Z \cdot e^{-\frac{M_{BF}}{N} \cdot T^Y} + Y \cdot Z)$
Short Scan	$O(K \cdot (L-Y-1) + Z \cdot (Y+1))$
Long Scan	$O(\frac{s}{B})$

DASlab @ Harvard SEAS

from write to read optimized

Design Continuums @CDIR2019

Designs / Terms	Log	LSH Table [80, 19, 82, 74, 58, 2, 89]	Tiered LSM-Tree [55, 23, 43]	Lazy Leveled LSM-Tree [25]	Leveled LSM-Tree [32, 29, 23]	COLA [15, 45]	FD-Tree [57]	B^ϵTree [16, 15, 44, 70, 9, 45]	B+Tree [13]	Sorted Array
T (Growth Factor)	$\frac{N \cdot E}{M_B}$	$\frac{N \cdot E}{M_B}$	$[2, B]$	$[2, B]$	$[2, B]$	2	$[2, B]$	$[2, B]$	B	$\frac{N \cdot E}{M_B}$
K (Hot Merge Threshold)	$T-1$	$T-1$	$T-1$	$T-1$	1	1	1	1	1	1
Z (Cold Merge Threshold)	$T-1$	$T-1$	$T-1$	1	1	1	1	1	1	1
D (Max. Node Size)	1	1	$[1, \frac{N}{B}]$	$[1, \frac{N}{B}]$	$[1, \frac{N}{B}]$	$\frac{N}{B}$	$\frac{N}{B}$	1	1	$\frac{N}{B}$
M_F (Fence & Filter Mem.)	$\frac{N \cdot F}{B}$	$N \cdot F \cdot (1 + \frac{1}{B})$	$N \cdot (\frac{F}{B} + 10)$	$N \cdot (\frac{F}{B} + 10)$	$N \cdot (\frac{F}{B} + 10)$	$\frac{F \cdot T \cdot M_B}{E \cdot B}$	$\frac{F \cdot T \cdot M_B}{E \cdot B}$	$\frac{F \cdot T \cdot M_B}{E \cdot B}$	$\frac{F \cdot T \cdot M_B}{E \cdot B}$	$\frac{N \cdot F}{B}$
Update	$O(\frac{1}{B})$	$O(\frac{1}{B})$	$O(\frac{L}{B})$	$O(\frac{1}{B} \cdot (T + L))$	$O(\frac{T}{B} \cdot L)$	$O(\frac{L}{B})$	$O(\frac{T}{B} \cdot L)$	$O(\frac{T}{B} \cdot L)$	$O(L)$	$O(\frac{N \cdot E}{M_B \cdot B})$
Zero Result Lookup	$O(\frac{N \cdot E}{M_B})$	$O(0)$	$O(T \cdot e^{\frac{-M_{BF}}{N}})$	$O(e^{-\frac{M_{BF}}{N}})$	$O(e^{-\frac{M_{BF}}{N}})$	$O(L)$	$O(L)$	$O(L)$	$O(L)$	$O(1)$
Existing Lookup	$O(\frac{N \cdot E}{M_B})$	$O(1)$	$O(1+T \cdot e^{\frac{-M_{BF}}{N}})$	$O(1)$	$O(1)$	$O(L)$	$O(L)$	$O(L)$	$O(L)$	$O(1)$
Short Scan	$O(\frac{N \cdot E}{M_B})$	$O(\frac{N \cdot E}{M_B})$	$O(L \cdot T)$	$O(1 + T \cdot (L-1))$	$O(L)$	$O(L)$	$O(L)$	$O(L)$	$O(L)$	$O(1)$
Long Scan	$O(\frac{N \cdot E}{M_B} \cdot \frac{s}{B})$	$O(\frac{N \cdot E}{M_B} \cdot \frac{s}{B})$	$O(T \cdot \frac{s}{B})$	$O(\frac{s}{B})$	$O(\frac{s}{B})$	$O(\frac{s}{B})$	$O(\frac{s}{B})$	$O(\frac{s}{B})$	$O(\frac{s}{B})$	$O(\frac{s}{B})$

Cosine
@PVLDB2022

Cloud-cost Optimized

Self Designing

Key-value Store

			Design Abstractions of Template	Type/Domain	Example templates for diverse data structures			
					LSM variants	B-Tree variants	LSH variants	A new design
LAYOUT PRIMITIVES	Design and hardware specification	*initialized by search through engine design space*	1.	**Key size:** Denotes the size of keys in the workload.	unsigned int	auto-configured from the sample workload			
			2.	**Value size:** Denotes the size of values in the workload. All values are accepted as variable-length strings.	string/slice *max size set to 1 GB*	auto-configured from the sample workload			
			3.	**Size ratio (T):** The maximum number of entries in a block (e.g. growth factor in LSM trees or fanout of B-trees.	unsigned integer | function (func)	[2,.. 32]	[32, 64, 128, 256, ..]	[1000, 1001, …] (T is large)	2
			4.	**Runs per hot level (K):** At what capacity hot levels are compacted. Rule: should be less than size ratio.	unsigned int	[1.. T]		[T-1]	7
			5.	**Runs per cold level (Z):** At what capacity cold levels are compacted. Rule: should be less than size ratio.	unsigned int	[1.. T]	[1]		32
			6.	**Logical block size (B):** Number of consecutive disk blocks.	unsigned int	[2048, 4096, …]			
			7.	**Buffer capacity** (M_B)**:** Denotes the amount of memory allocated to in-memory buffer/memtables. Configurable w.r.t file size.	64-bit floating point | function (func)	[64 MB, 128 MB, …]	[1 MB, 2 MB, …]	[64 MB, 128 MB, …]	h/w dependent
			8.	**Indexes** (M_{FP})**:** Amount of memory allocated to indexes (fence pointers/hashtables).	64-bit floating point | function (func)	memory to cover L	memory for first level	memory for hash table	h/w dependent
ALGORITHMIC ABSTRACTIONS			9.	**Bloom filter memory** (M_{BF})**:** Denotes the bits/entry assigned to Bloom filters.	64-bit float | func(FPR)	10 bits/key			func(FPR)
	Data access	*derived with empirically verified rules*	10.	**Bloom filter design:** Denotes the granularity of Bloom filters, e.g., one Bloom filter instance per block or per file or per run. The default is file.	block | file | run	file			file
			11.	**Compaction/Restructuring algorithm:** Full does level-to-level compaction; partial is file-to-file; and hybrid uses both full and partial at separate levels.	partial | full | hybrid	full, partial	partial	partial	hybrid
			12.	**Run strategy:** Denotes which run to be picked for compaction (only for partial/ hybrid compaction).	first | last_full | fullest	first, fullest, last_full		first	fullest
			13.	**File picking strategy:** Denotes which file to be picked for compaction (for partial/ hybrid compaction). For LSM-trees we set default to dense_fp as it empirically works the best. B-trees pick the first file found to be full. LSH-table restructures at the granularity of runs.	oldest_merged | oldest_flushed | dense_fp | sparse_fp | choose_first	dense_fp	choose_first		dense_fp (hot), choose_first (cold)
			14.	**Merge threshold:** If a level is more than x% full, a compaction is triggered.	64-bit floating point	[0.7..1]	0.5		0.75
			15.	**Full compaction levels:** Denotes how many levels will have full compaction (only for hybrid compaction). The default is set to 2.	unsigned integer | function (func)	[1..L]			L-Y (from optimal config)
	Parallelism		16.	**No. of CPUs:** Number of available cores to use in a VM.	unsigned int	Use all available cores			
			17.	**No of threads:** Denotes how many threads are used to process the workload.	unsigned int	Use 1 thread per CPU core			

Storage engine template in Cosine and example initializations for diverse storage engine designs.

Cosine
@PVLDB2022

Cloud-cost Optimized

Self Designing

Key-value Store

			Design Abstractions of Template	Type/Domain	Example templates for diverse data structures			
					LSM variants	B-Tree variants	LSH variants	A new design
LAYOUT PRIMITIVES	Design and hardware specification	*initialized by search through engine design space*	1.	**Key size:** Denotes the size of keys in the workload.	unsigned int	auto-configured from the sample workload		
			2.	**Value size:** Denotes the size of values in the workload. All values are accepted as variable-length strings.	string/slice *max size set to 1 GB*	auto-configured from the sample workload		
			3.	**Size ratio (T):** The maximum number of entries in a block (e.g. growth factor in LSM trees or fanout of B-trees.	unsigned integer | function (func)	[2,.. 32]	[32, 64, 128, 256, ..]	[1000, 1001, …] (T is large)	2
			4.	**Runs per hot level (K):** At what capacity hot levels are compacted. Rule: should be less than size ratio.	unsigned int	[1.. T]		[T-1]	7
			5.	**Runs per cold level (Z):** At what capacity cold levels are compacted. Rule: should be less than size ratio.	unsigned int	[1.. T]	[1]		32
			6.	**Logical block size (B):** Number of consecutive disk blocks.	unsigned int	[2048, 4096, …]			
			7.	**Buffer capacity** (M_B)**:** Denotes the amount of memory allocated to in-memory buffer/memtables. Configurable w.r.t file size.	64-bit floating point | function (func)	[64 MB, 128 MB, …]	[1 MB, 2 MB, …]	[64 MB, 128 MB, …]	h/w dependent
			8.	**Indexes** (M_{FP})**:** Amount of memory allocated to indexes (fence pointers/hashtables).	64-bit floating point | function (func)	memory to cover L	memory for first level	memory for hash table	h/w dependent
ALGORITHMIC ABSTRACTIONS			9.	**Bloom filter memory** (M_{BF})**:** Denotes the bits/entry assigned to Bloom filters.	64-bit float | func(FPR)	10 bits/key			func(FPR)
	Data access	*derived with empirically verified rules*	10.	**Bloom filter design:** Denotes the granularity of Bloom filters, e.g., one Bloom filter instance per block or per file or per run. The default is file.	block | file | run	file			file
			11.	**Compaction/Restructuring algorithm:** Full does level-to-level compaction; partial is file-to-file; and hybrid uses both full and partial at separate levels.	partial | full | hybrid	full, partial	partial	partial	hybrid
			12.	**Run strategy:** Denotes which run to be picked for compaction (only for partial/ hybrid compaction).	first | last_full | fullest	first, fullest, last_full		first	fullest
			13.	**File picking strategy:** Denotes which file to be picked for compaction (for partial/ hybrid compaction). For LSM-trees we set default to dense_fp as it empirically works the best. B-trees pick the first file found to be full. LSH-table restructures at the granularity of runs.	oldest_merged | oldest_flushed | dense_fp | sparse_fp | choose_first	dense_fp	choose_first		dense_fp (hot), choose_first (cold)
			14.	**Merge threshold:** If a level is more than x% full, a compaction is triggered.	64-bit floating point	[0.7..1]	0.5		0.75
			15.	**Full compaction levels:** Denotes how many levels will have full compaction (only for hybrid compaction). The default is set to 2.	unsigned integer | function (func)	[1..L]			L-Y (from optimal config)
	Parallelism		16.	**No. of CPUs:** Number of available cores to use in a VM.	unsigned int	Use all available cores			
			17.	**No of threads:** Denotes how many threads are used to process the workload.	unsigned int	Use 1 thread per CPU core			

Storage engine template in Cosine and example initializations for diverse storage engine designs.

Cosine
@PVLDB2022

Cloud-cost Optimized

Self Designing

Key-value Store

			Design Abstractions of Template	Type/Domain	Example templates for diverse data structures			
					LSM variants	B-Tree variants	LSH variants	A new design
LAYOUT PRIMITIVES	Design and hardware specification	*initialized by search through engine design space*	1.	**Key size:** Denotes the size of keys in the workload.	unsigned int	auto-configured from the sample workload		
			2.	**Value size:** Denotes the size of values in the workload. All values are accepted as variable-length strings.	string/slice *max size set to 1 GB*	auto-configured from the sample workload		
			3.	**Size ratio (T):** The maximum number of entries in a block (e.g. growth factor in LSM trees or fanout of B-trees.	unsigned integer | function (func)	[2,.. 32]	[32, 64, 128, 256, ..]	[1000, 1001, …] (T is large)	2
			4.	**Runs per hot level (K):** At what capacity hot levels are compacted. Rule: should be less than size ratio.	unsigned int	[1.. T]		[T-1]	7
			5.	**Runs per cold level (Z):** At what capacity cold levels are compacted. Rule: should be less than size ratio.	unsigned int	[1.. T]	[1]		32
			6.	**Logical block size (B):** Number of consecutive disk blocks.	unsigned int	[2048, 4096, …]			
			7.	**Buffer capacity** (M_B)**:** Denotes the amount of memory allocated to in-memory buffer/memtables. Configurable w.r.t file size.	64-bit floating point | function (func)	[64 MB, 128 MB, …]	[1 MB, 2 MB, …]	[64 MB, 128 MB, …]	h/w dependent
			8.	**Indexes** (M_{FP})**:** Amount of memory allocated to indexes (fence pointers/hashtables).	64-bit floating point | function (func)	memory to cover L	memory for first level	memory for hash table	h/w dependent
ALGORITHMIC ABSTRACTIONS			9.	**Bloom filter memory** (M_{BF})**:** Denotes the bits/entry assigned to Bloom filters.	64-bit float | func(FPR)	10 bits/key			func(FPR)
	Data access	*derived with empirically verified rules*	10.	**Bloom filter design:** Denotes the granularity of Bloom filters, e.g., one Bloom filter instance per block or per file or per run. The default is file.	block | file | run	file			file
			11.	**Compaction/Restructuring algorithm:** Full does level-to-level compaction; partial is file-to-file; and hybrid uses both full and partial at separate levels.	partial | full | hybrid	full, partial	partial	partial	hybrid
			12.	**Run strategy:** Denotes which run to be picked for compaction (only for partial/ hybrid compaction).	first | last_full | fullest	first, fullest, last_full		first	fullest
			13.	**File picking strategy:** Denotes which file to be picked for compaction (for partial/ hybrid compaction). For LSM-trees we set default to dense_fp as it empirically works the best. B-trees pick the first file found to be full. LSH-table restructures at the granularity of runs.	oldest_merged | oldest_flushed | dense_fp | sparse_fp | choose_first	dense_fp	choose_first		dense_fp (hot), choose_first (cold)
			14.	**Merge threshold:** If a level is more than x% full, a compaction is triggered.	64-bit floating point	[0.7..1]	0.5		0.75
			15.	**Full compaction levels:** Denotes how many levels will have full compaction (only for hybrid compaction). The default is set to 2.	unsigned integer | function (func)	[1..L]			L-Y (from optimal config)
	Parallelism		16.	**No. of CPUs:** Number of available cores to use in a VM.	unsigned int	Use all available cores			
			17.	**No of threads:** Denotes how many threads are used to process the workload.	unsigned int	Use 1 thread per CPU core			

Storage engine template in Cosine and example initializations for diverse storage engine designs.

Cosine
@PVLDB2022

Cloud-cost Optimized

Self Designing

Key-value Store

			Design Abstractions of Template	Type/Domain	Example templates for diverse data structures			
					LSM variants	B-Tree variants	LSH variants	A new design
LAYOUT PRIMITIVES	Design and hardware specification	*initialized by search through engine design space*	1.	**Key size:** Denotes the size of keys in the workload.	unsigned int	auto-configured from the sample workload		
			2.	**Value size:** Denotes the size of values in the workload. All values are accepted as variable-length strings.	string/slice *max size set to 1 GB*	auto-configured from the sample workload		
			3.	**Size ratio (T):** The maximum number of entries in a block (e.g. growth factor in LSM trees or fanout of B-trees.	unsigned integer | function (func)	[2,.. 32]	[32, 64, 128, 256, ..]	[1000, 1001, …] (T is large)	2
			4.	**Runs per hot level (K):** At what capacity hot levels are compacted. Rule: should be less than size ratio.	unsigned int	[1.. T]		[T-1]	7
			5.	**Runs per cold level (Z):** At what capacity cold levels are compacted. Rule: should be less than size ratio.	unsigned int	[1.. T]	[1]		32
			6.	**Logical block size (B):** Number of consecutive disk blocks.	unsigned int	[2048, 4096, …]			
			7.	**Buffer capacity** (M_B)**:** Denotes the amount of memory allocated to in-memory buffer/memtables. Configurable w.r.t file size.	64-bit floating point | function (func)	[64 MB, 128 MB, …]	[1 MB, 2 MB, …]	[64 MB, 128 MB, …]	h/w dependent
			8.	**Indexes** (M_{FP})**:** Amount of memory allocated to indexes (fence pointers/hashtables).	64-bit floating point | function (func)	memory to cover L	memory for first level	memory for hash table	h/w dependent
			9.	**Bloom filter memory** (M_{BF})**:** Denotes the bits/entry assigned to Bloom filters.	64-bit float | func(FPR)	10 bits/key			func(FPR)
ALGORITHMIC ABSTRACTIONS	Data access	*derived with empirically verified rules*	10.	**Bloom filter design:** Denotes the granularity of Bloom filters, e.g., one Bloom filter instance per block or per file or per run. The default is file.	block | file | run	file			file
			11.	**Compaction/Restructuring algorithm:** Full does level-to-level compaction; partial is file-to-file; and hybrid uses both full and partial at separate levels.	partial | full | hybrid	full, partial	partial	partial	hybrid
			12.	**Run strategy:** Denotes which run to be picked for compaction (only for partial/ hybrid compaction).	first | last_full | fullest	first, fullest, last_full		first	fullest
			13.	**File picking strategy:** Denotes which file to be picked for compaction (for partial/ hybrid compaction). For LSM-trees we set default to dense_fp as it empirically works the best. B-trees pick the first file found to be full. LSH-table restructures at the granularity of runs.	oldest_merged | oldest_flushed | dense_fp | sparse_fp | choose_first	dense_fp	choose_first		dense_fp (hot), choose_first (cold)
	Parallelism		14.	**Merge threshold:** If a level is more than x% full, a compaction is triggered.	64-bit floating point	[0.7..1]	0.5		0.75
			15.	**Full compaction levels:** Denotes how many levels will have full compaction (only for hybrid compaction). The default is set to 2.	unsigned integer | function (func)	[1..L]			L-Y (from optimal config)
			16.	**No. of CPUs:** Number of available cores to use in a VM.	unsigned int	Use all available cores			
			17.	**No of threads:** Denotes how many threads are used to process the workload.	unsigned int	Use 1 thread per CPU core			

Storage engine template in Cosine and example initializations for diverse storage engine designs.

Cosine
@PVLDB2022

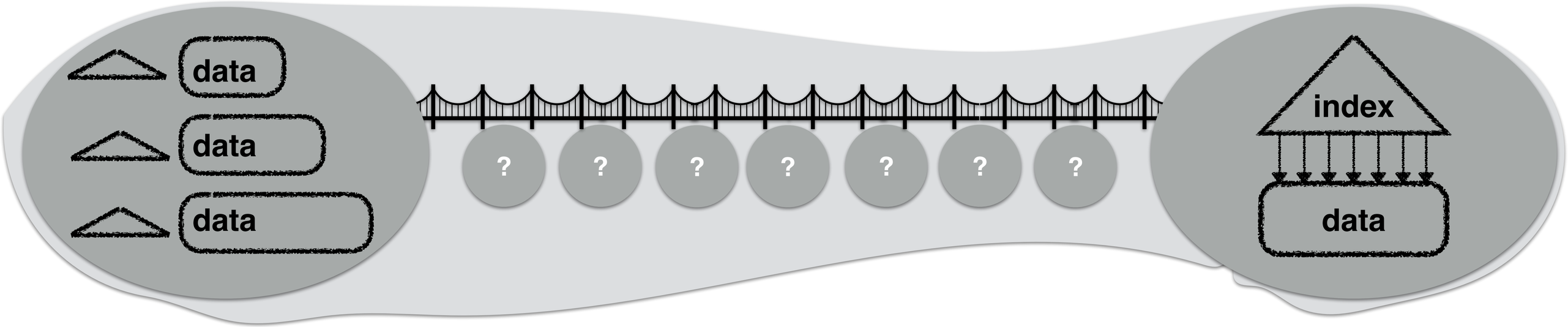
Cloud-cost Optimized

Self Designing

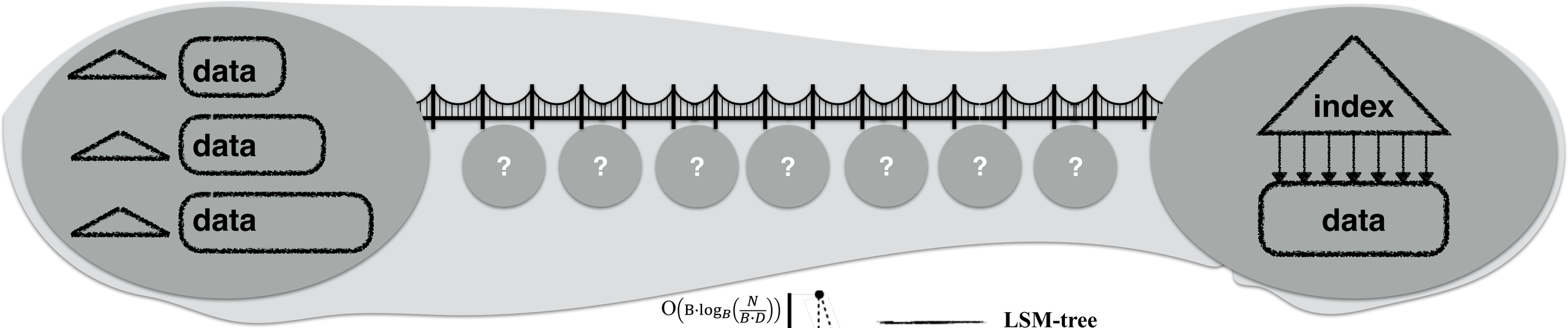
Key-value Store

			Design Abstractions of Template	Type/Domain	Example templates for diverse data structures			
					LSM variants	B-Tree variants	LSH variants	A new design
LAYOUT PRIMITIVES	Design and hardware specification	*initialized by search through engine design space*	1.	**Key size:** Denotes the size of keys in the workload.	unsigned int	auto-configured from the sample workload		
			2.	**Value size:** Denotes the size of values in the workload. All values are accepted as variable-length strings.	string/slice *max size set to 1 GB*	auto-configured from the sample workload		
			3.	**Size ratio (T):** The maximum number of entries in a block (e.g. growth factor in LSM trees or fanout of B-trees.	unsigned integer | function (func)	[2,.. 32]	[32, 64, 128, 256, ..]	[1000, 1001, …] (T is large)	2
			4.	**Runs per hot level (K):** At what capacity hot levels are compacted. Rule: should be less than size ratio.	unsigned int	[1.. T]		[T-1]	7
			5.	**Runs per cold level (Z):** At what capacity cold levels are compacted. Rule: should be less than size ratio.	unsigned int	[1.. T]	[1]		32
			6.	**Logical block size (B):** Number of consecutive disk blocks.	unsigned int	[2048, 4096, …]			
			7.	**Buffer capacity** (M_B)**:** Denotes the amount of memory allocated to in-memory buffer/memtables. Configurable w.r.t file size.	64-bit floating point | function (func)	[64 MB, 128 MB, …]	[1 MB, 2 MB, …]	[64 MB, 128 MB, …]	h/w dependent
			8.	**Indexes** (M_{FP})**:** Amount of memory allocated to indexes (fence pointers/hashtables).	64-bit floating point | function (func)	memory to cover L	memory for first level	memory for hash table	h/w dependent
			9.	**Bloom filter memory** (M_{BF})**:** Denotes the bits/entry assigned to Bloom filters.	64-bit float | func(FPR)	10 bits/key			func(FPR)
ALGORITHMIC ABSTRACTIONS	Data access	*derived with empirically verified rules*	10.	**Bloom filter design:** Denotes the granularity of Bloom filters, e.g., one Bloom filter instance per block or per file or per run. The default is file.	block | file | run	file			file
			11.	**Compaction/Restructuring algorithm:** Full does level-to-level compaction; partial is file-to-file; and hybrid uses both full and partial at separate levels.	partial | full | hybrid	full, partial	partial	partial	hybrid
			12.	**Run strategy:** Denotes which run to be picked for compaction (only for partial/ hybrid compaction).	first | last_full | fullest	first, fullest, last_full		first	fullest
			13.	**File picking strategy:** Denotes which file to be picked for compaction (for partial/ hybrid compaction). For LSM-trees we set default to dense_fp as it empirically works the best. B-trees pick the first file found to be full. LSH-table restructures at the granularity of runs.	oldest_merged | oldest_flushed | dense_fp | sparse_fp | choose_first	dense_fp	choose_first		dense_fp (hot), choose_first (cold)
	Parallelism		14.	**Merge threshold:** If a level is more than x% full, a compaction is triggered.	64-bit floating point	[0.7..1]	0.5		0.75
			15.	**Full compaction levels:** Denotes how many levels will have full compaction (only for hybrid compaction). The default is set to 2.	unsigned integer | function (func)	[1..L]			L-Y (from optimal config)
			16.	**No. of CPUs:** Number of available cores to use in a VM.	unsigned int	Use all available cores			
			17.	**No of threads:** Denotes how many threads are used to process the workload.	unsigned int	Use 1 thread per CPU core			

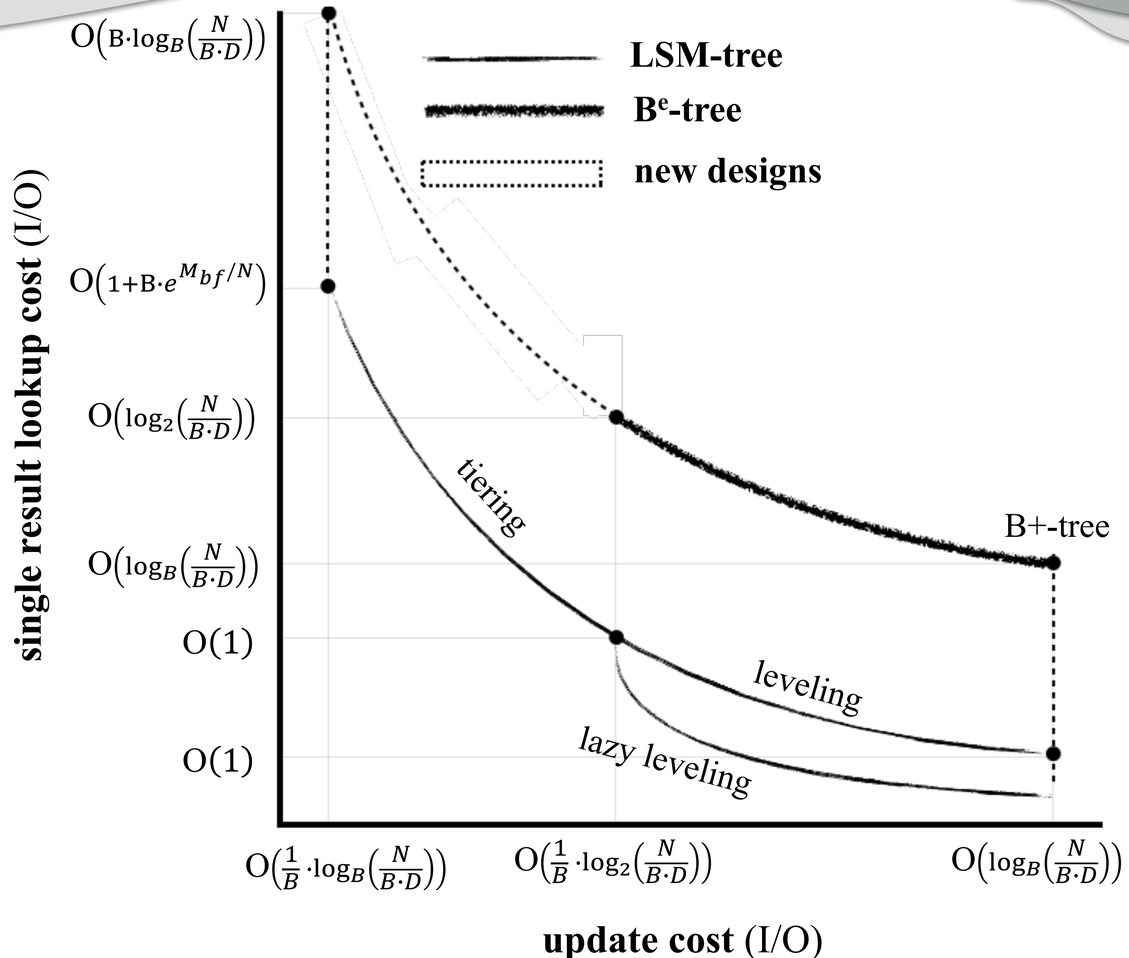
Storage engine template in Cosine and example initializations for diverse storage engine designs.



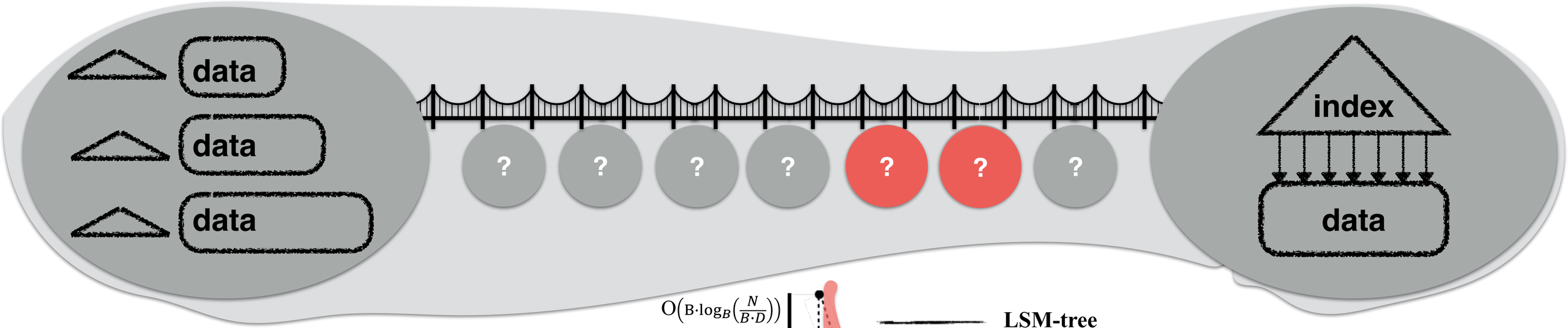
assisted discovery



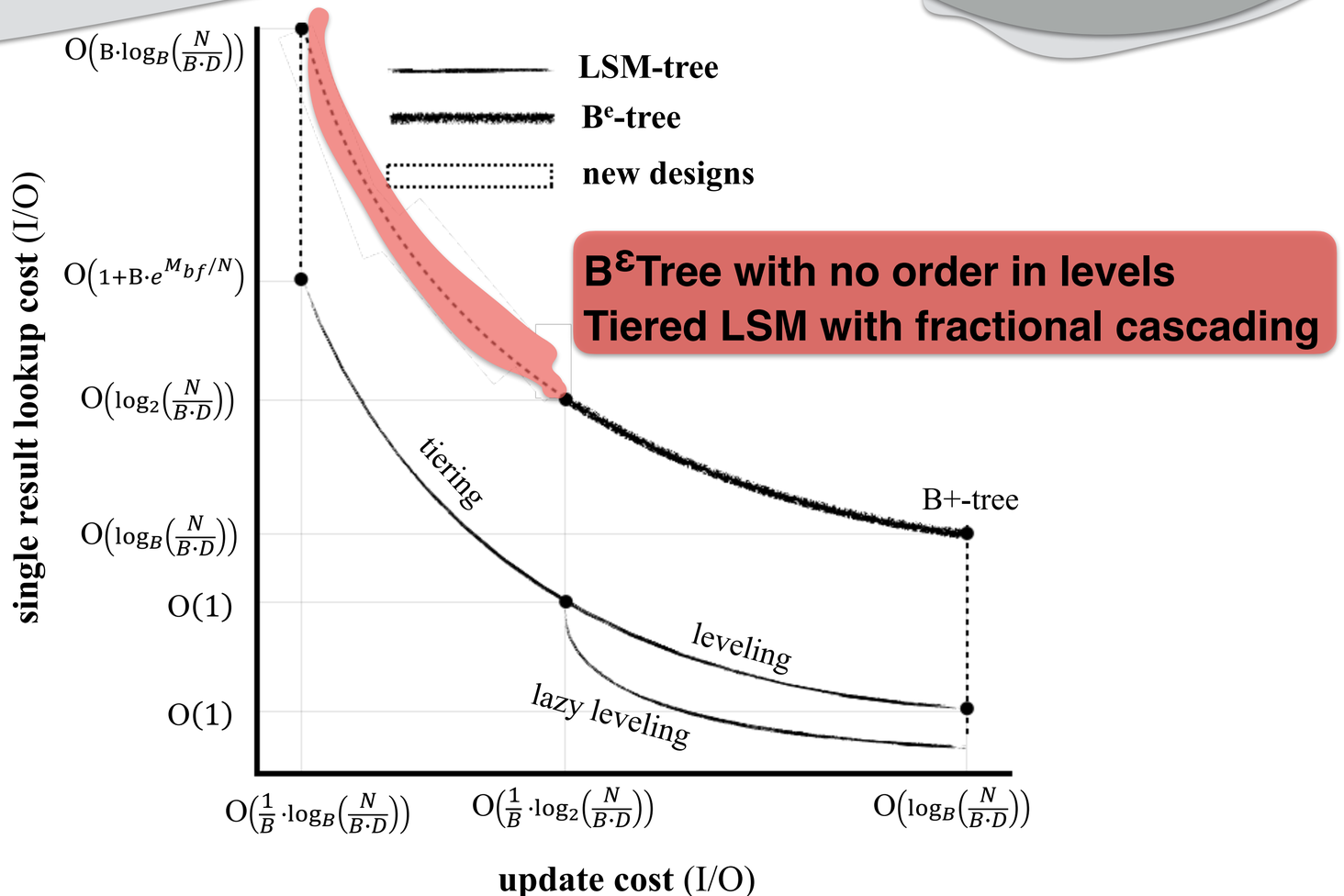
assisted discovery



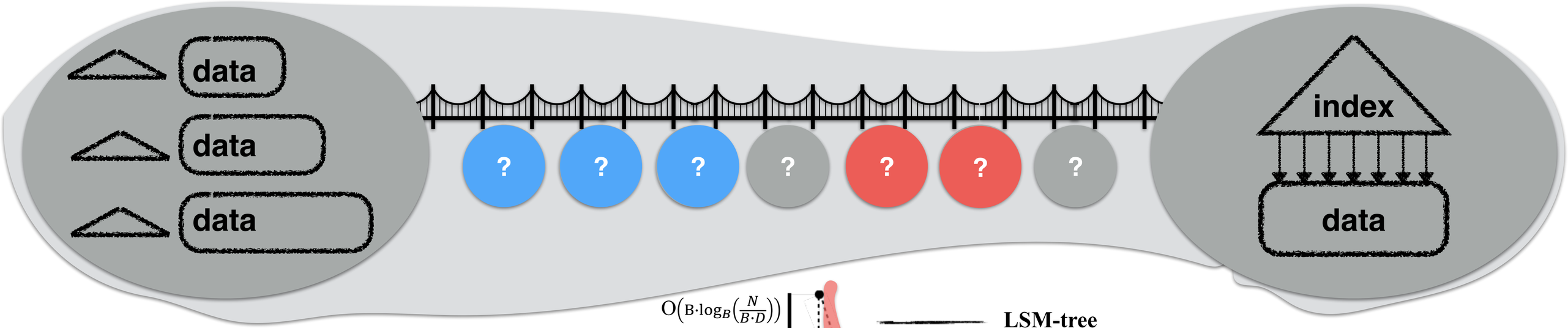
DASlab @ Harvard SEAS



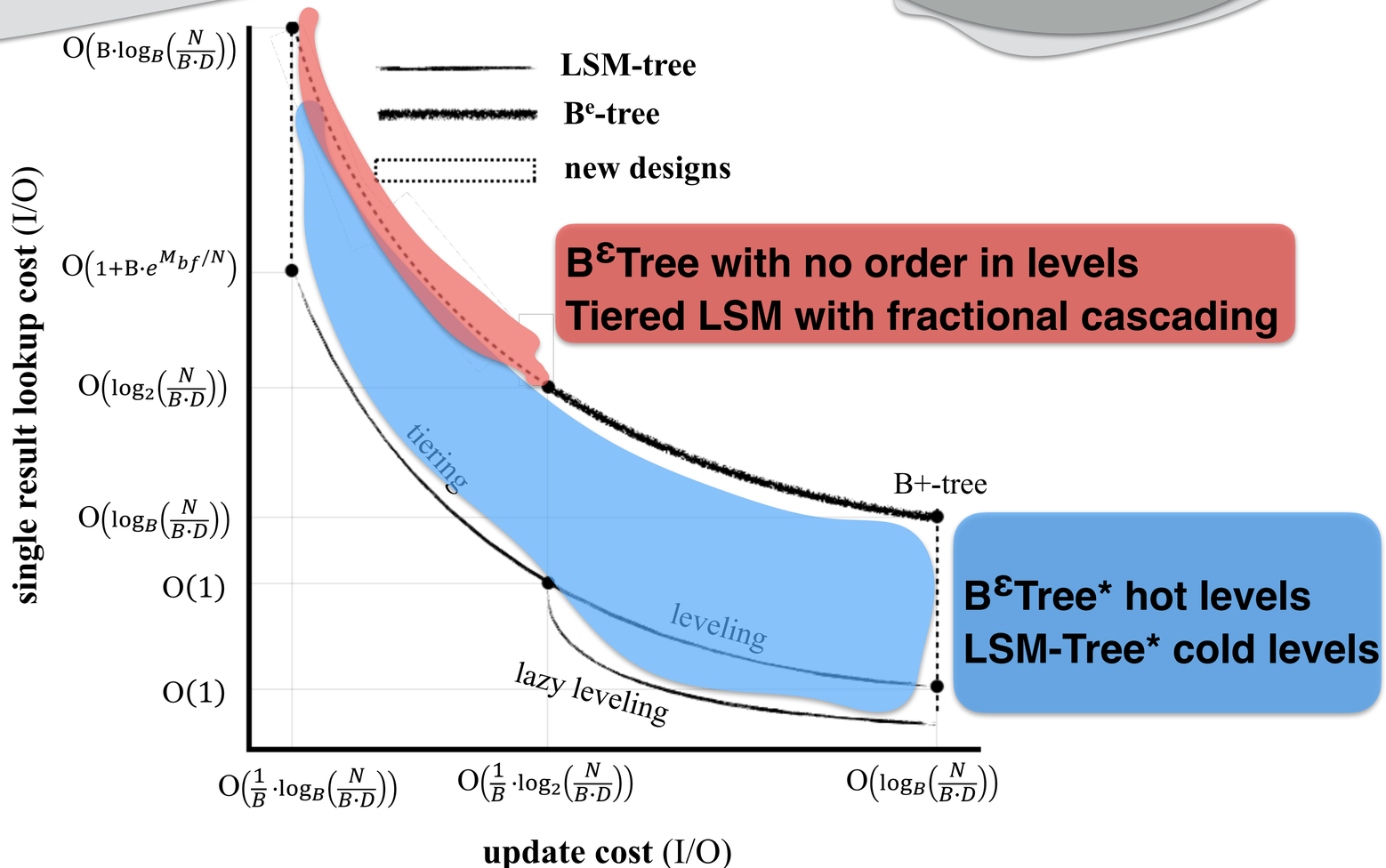
assisted discovery



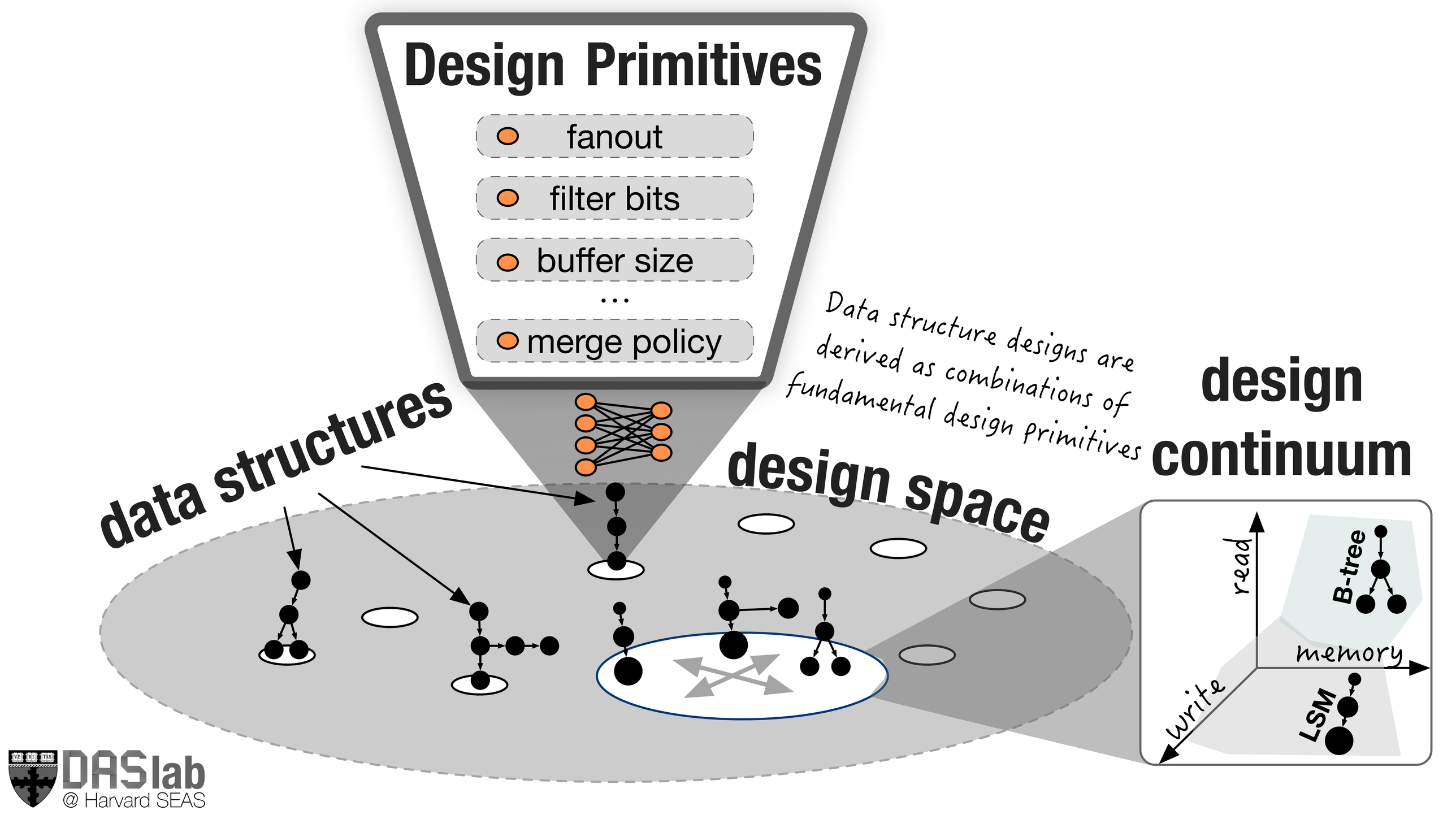
DASlab @ Harvard SEAS



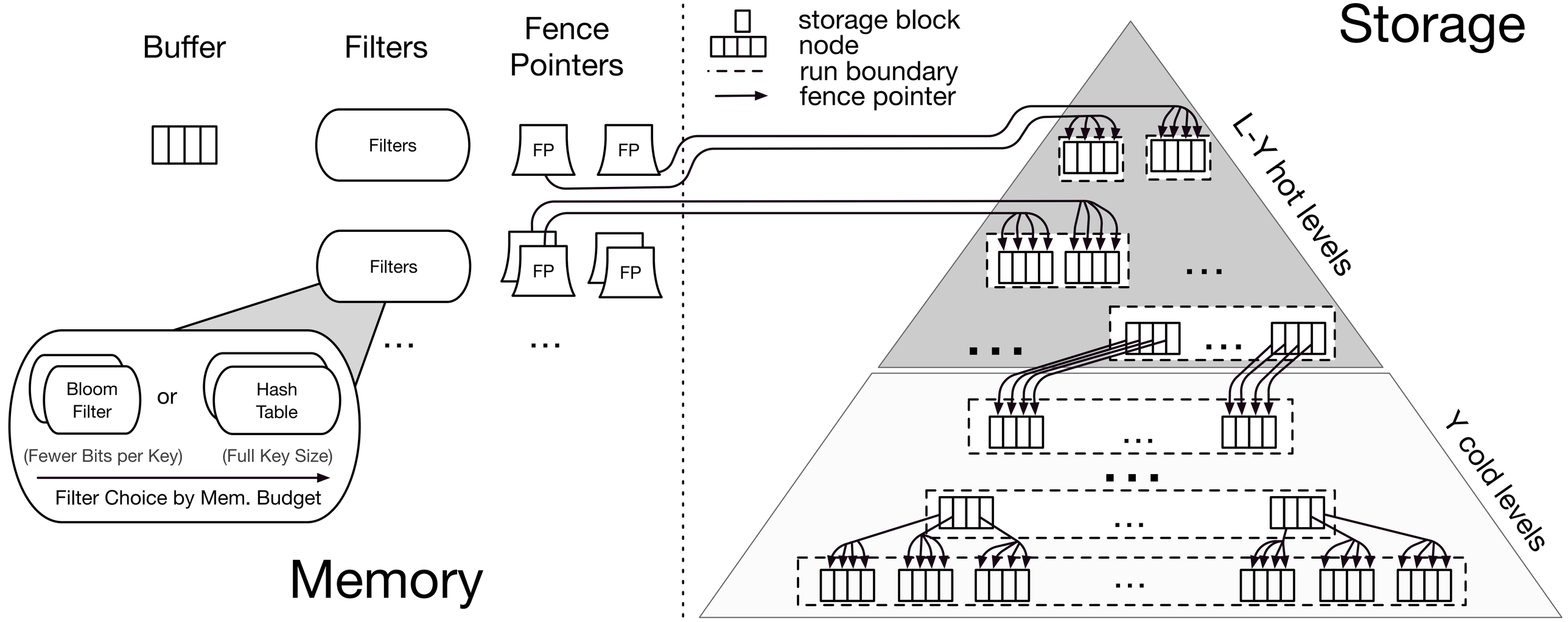
assisted discovery



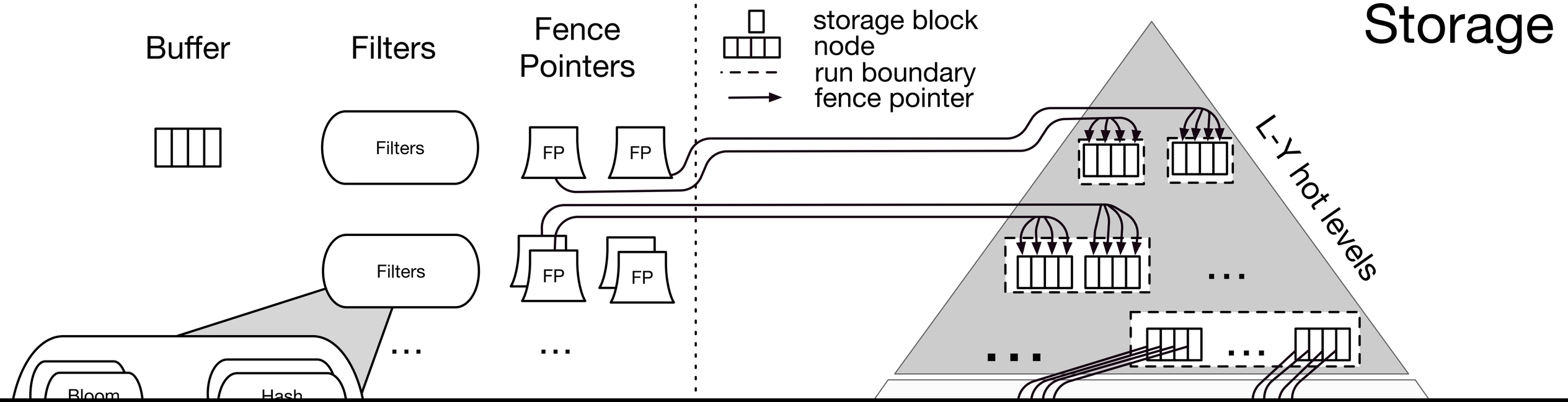
DASlab @ Harvard SEAS


Design Primitives
fanout
filter bits
buffer size
...
merge policy
Data structure designs are derived as combinations of fundamental design primitives
data structures
design space
design continuum
read
write
memory
B-tree
LSM
DASlab
@ Harvard SEAS

unified design
storage engine template



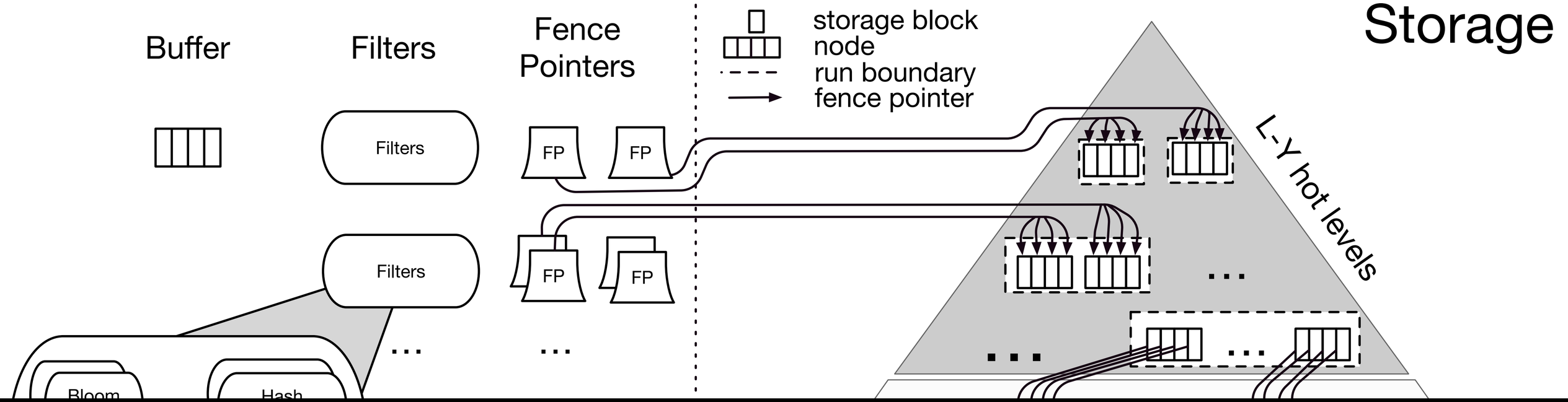
unified design
storage engine template



analytical I/O model

unified closed form

unified design
storage engine template



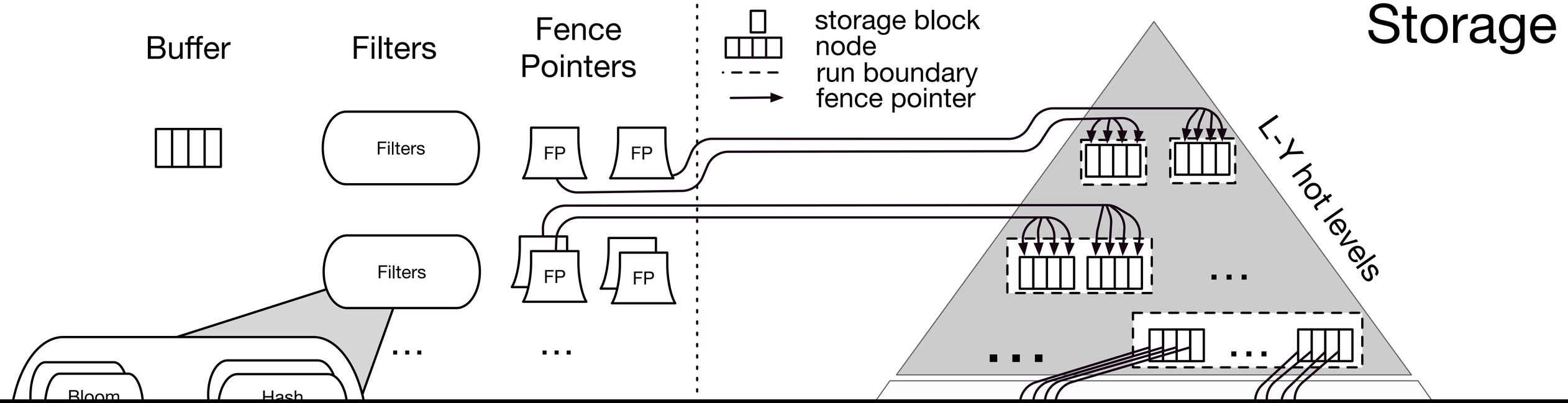
analytical I/O model

unified closed form

learned CPU model

h/w, parallelism

unified design
storage engine template



analytical I/O model

unified closed form

learned CPU model

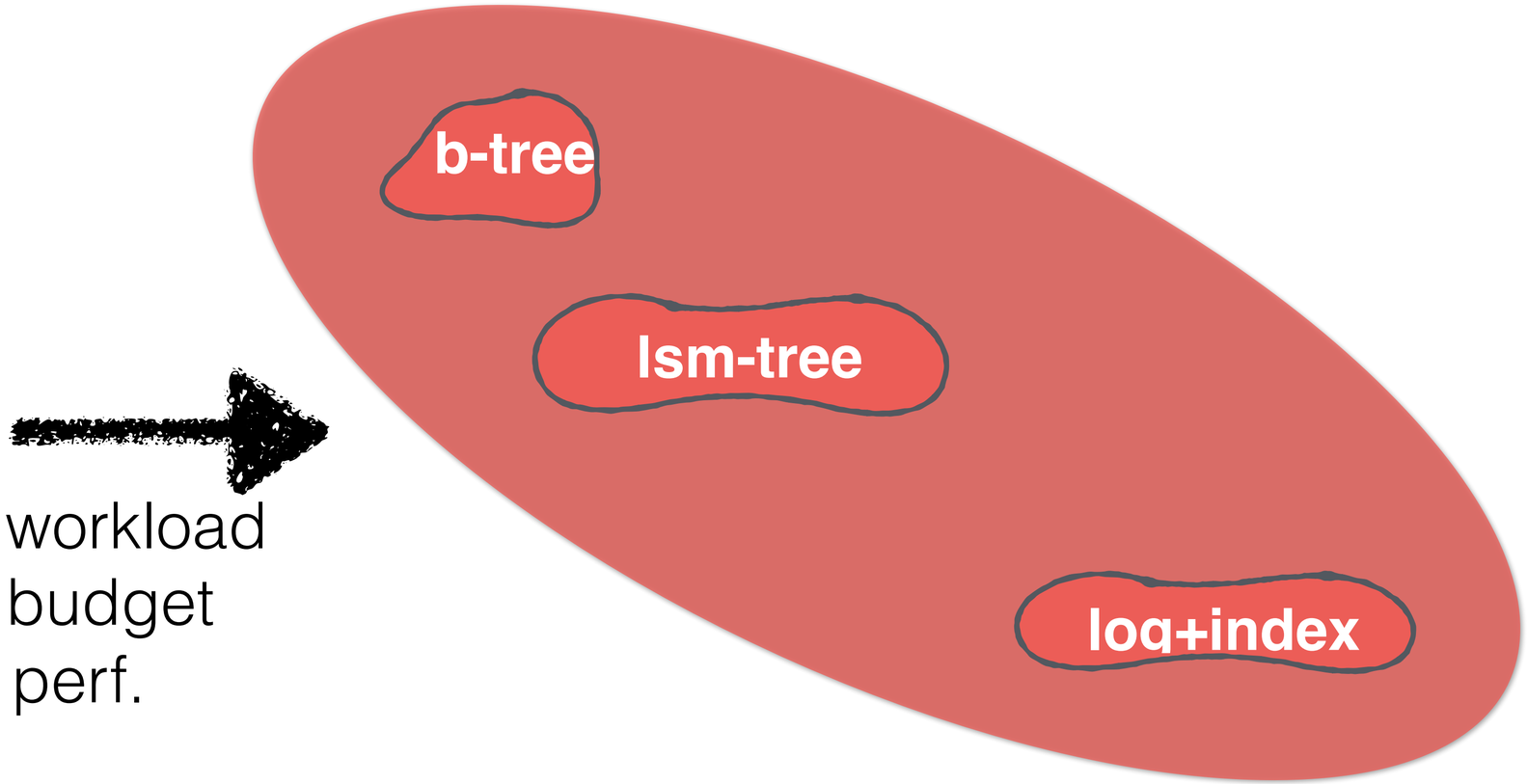
h/w, parallelism

cloud cost mapping & SLA

AWS, Azure, Google

workload
budget
perf.

analytical I/O model	**learned CPU model**	**cloud cost mapping & SLA**
unified closed form	h/w, parallelism	AWS, Azure, Google



analytical I/O model

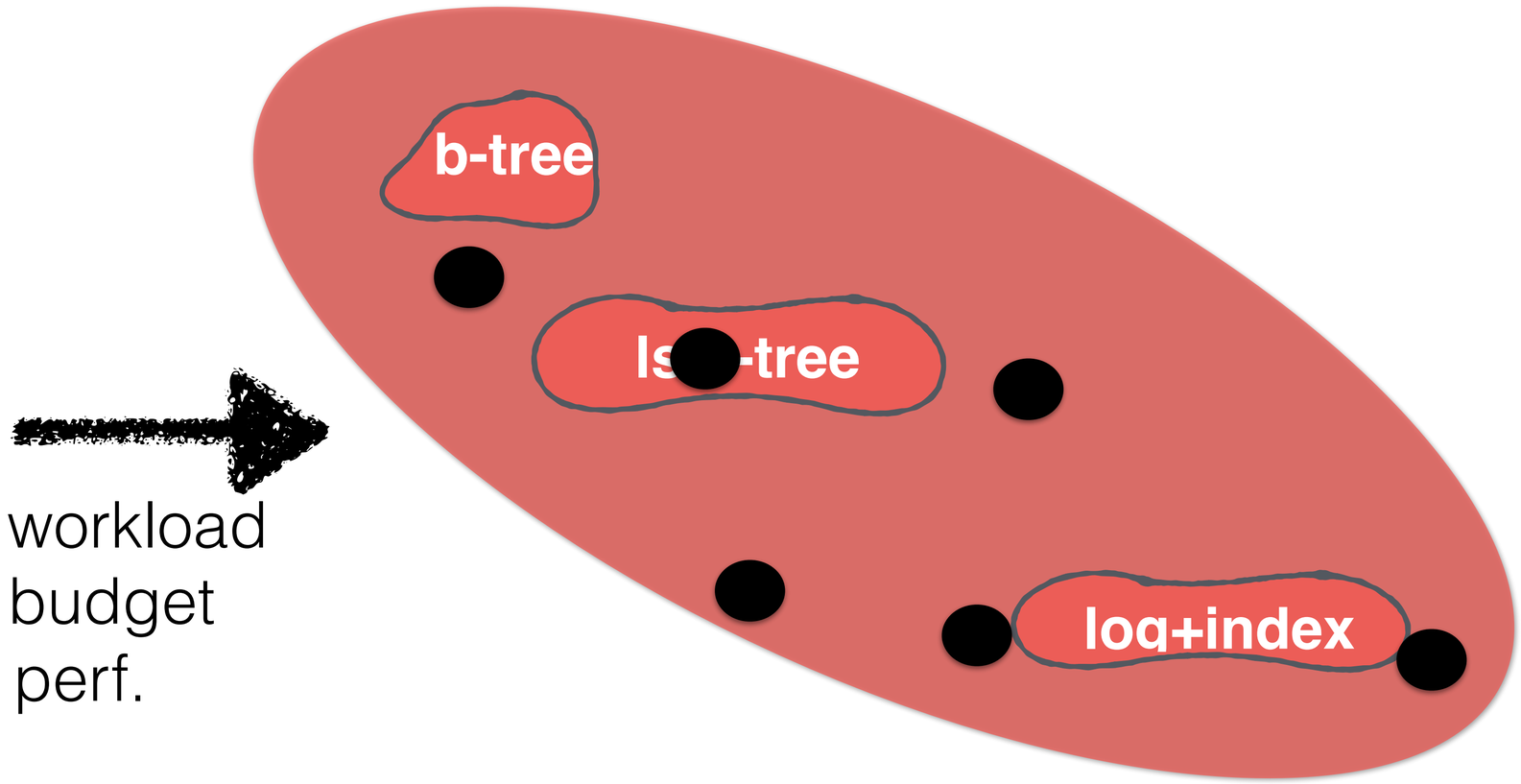
unified closed form

learned CPU model

h/w, parallelism

cloud cost mapping & SLA

AWS, Azure, Google



analytical I/O model

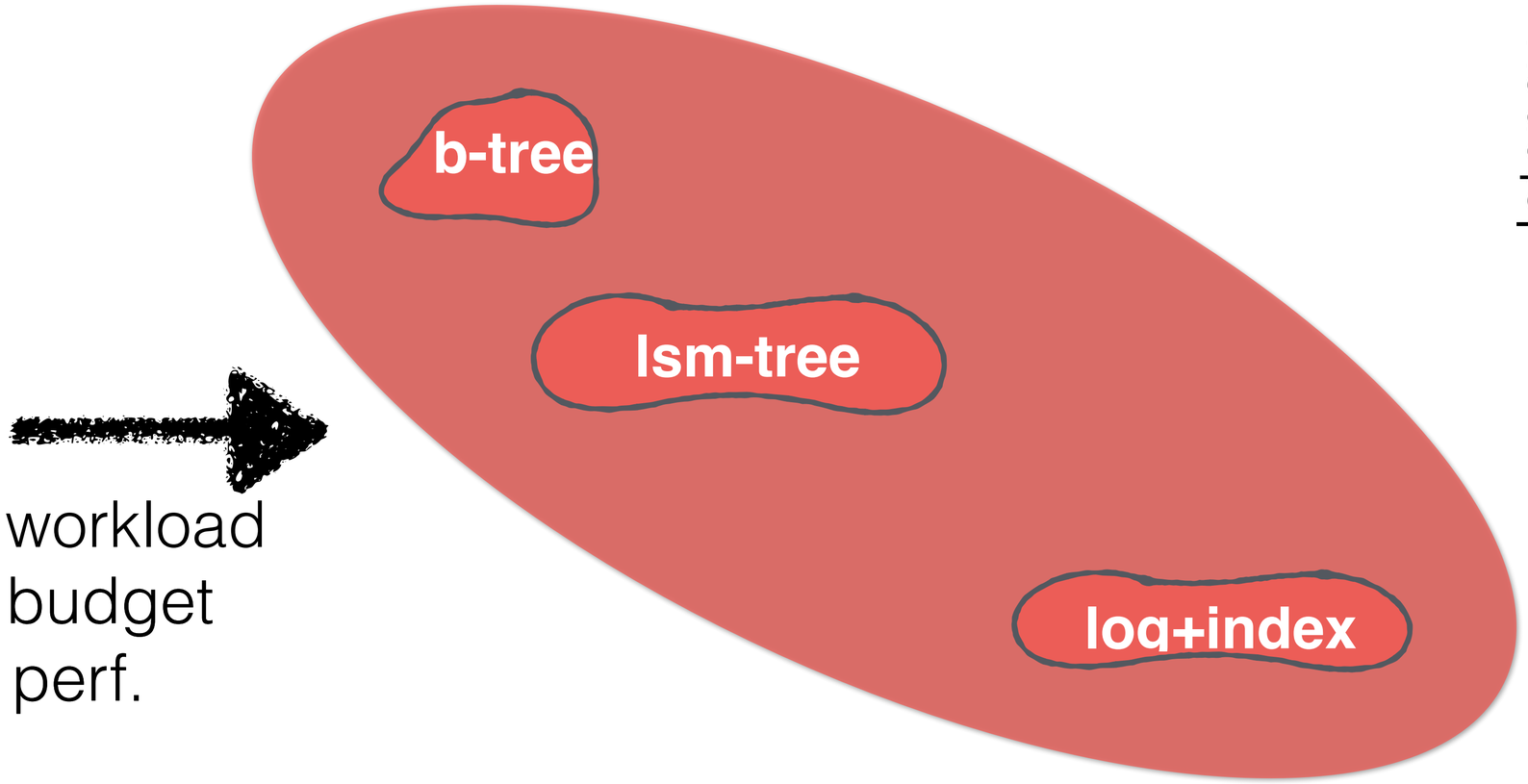
unified closed form

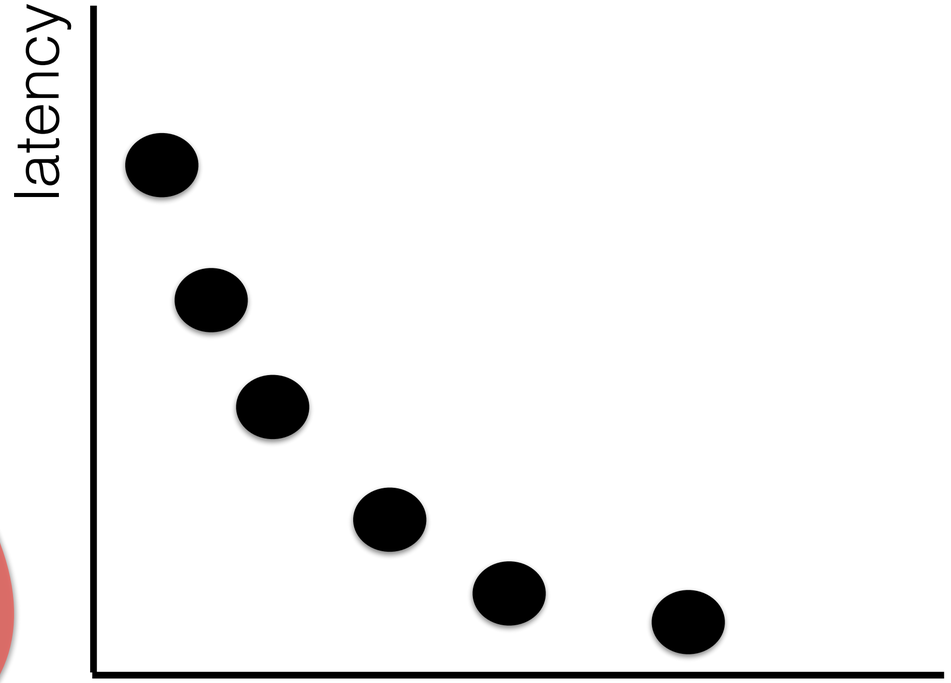
learned CPU model

h/w, parallelism

cloud cost mapping & SLA

AWS, Azure, Google





analytical I/O model

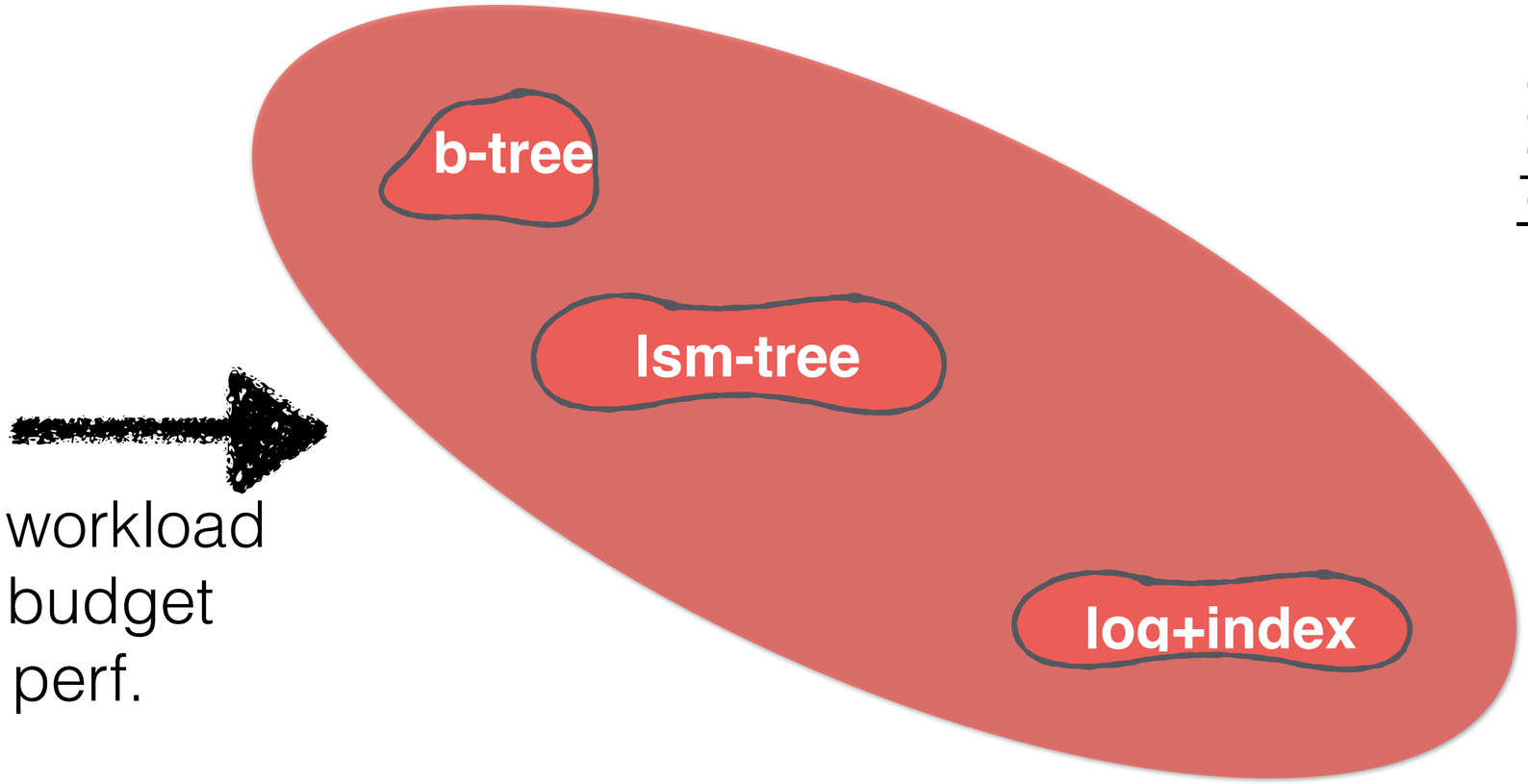
unified closed form

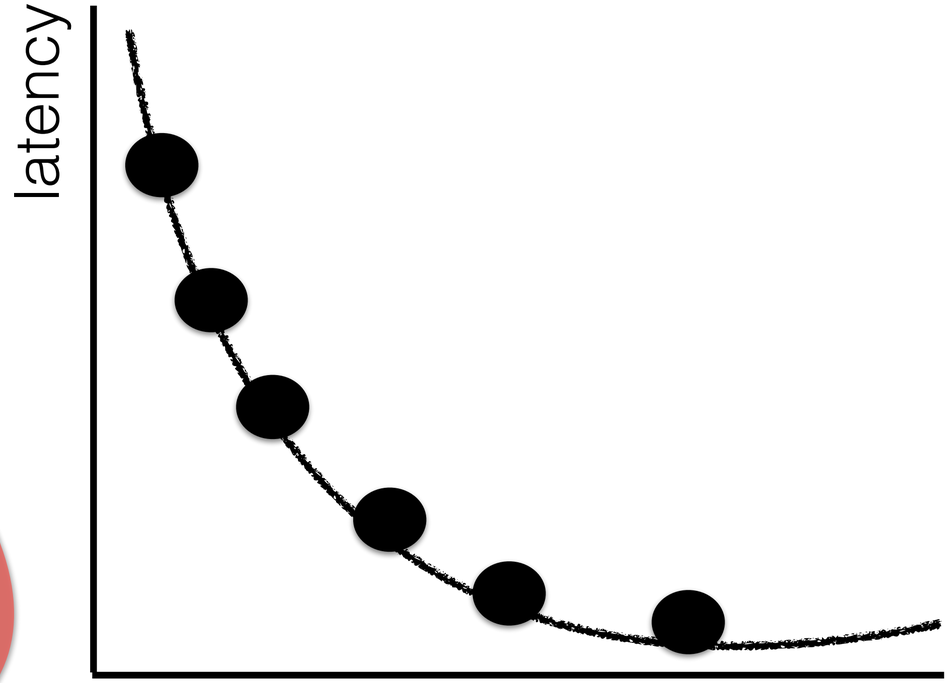
learned CPU model

h/w, parallelism

cloud cost mapping & SLA

AWS, Azure, Google





analytical I/O model

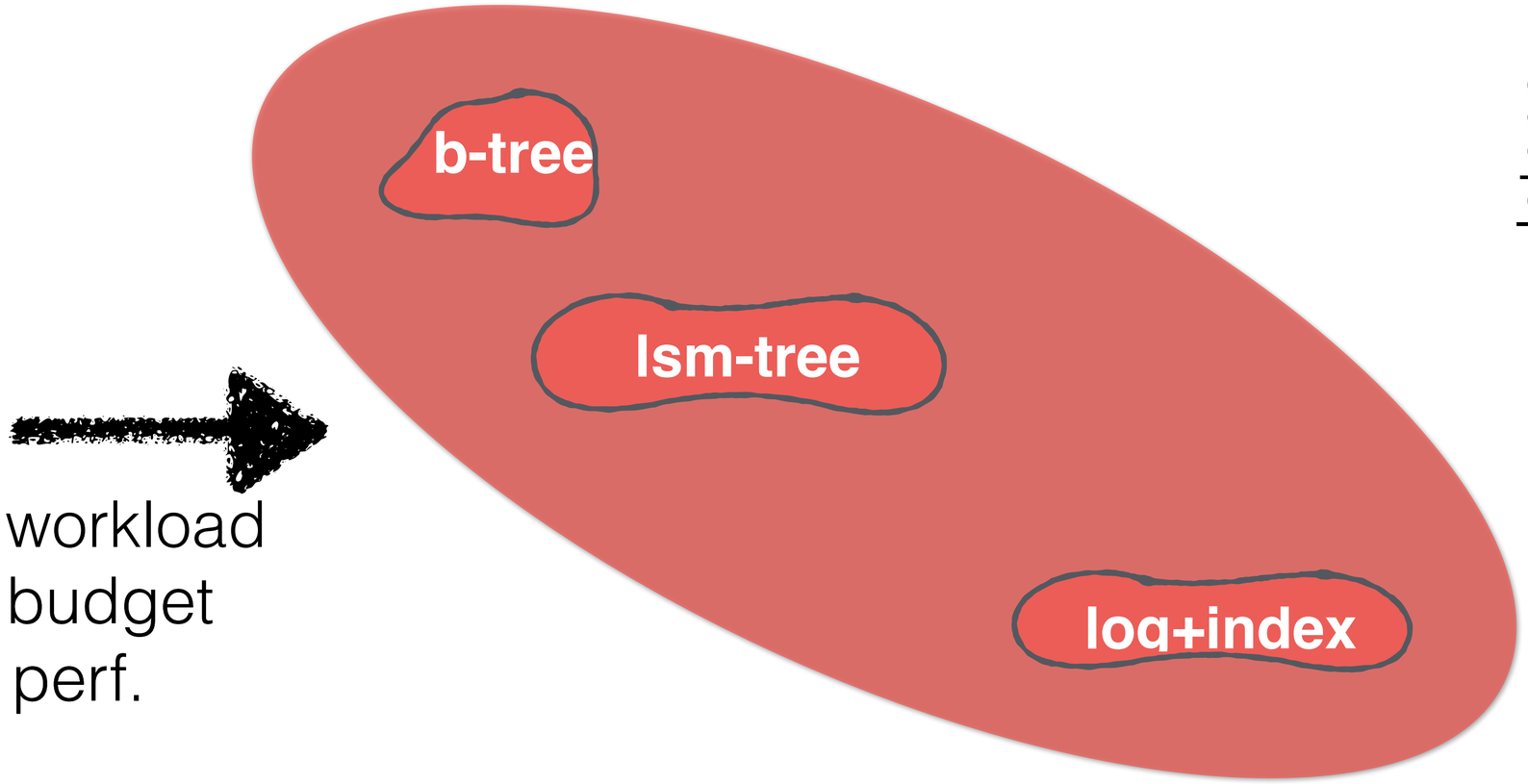
unified closed form

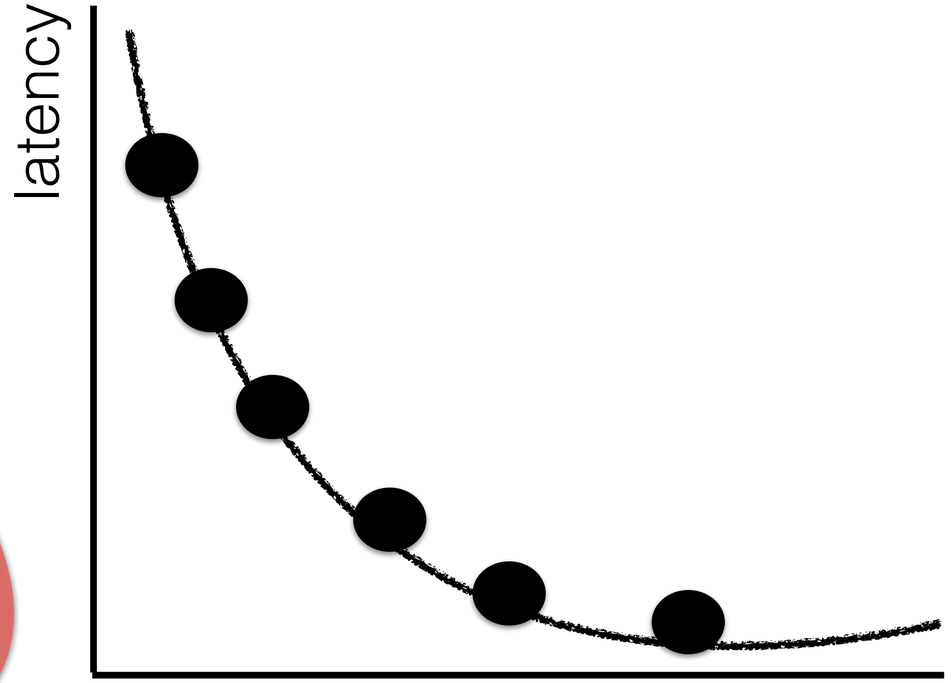
learned CPU model

h/w, parallelism

cloud cost mapping & SLA

AWS, Azure, Google





analytical I/O model

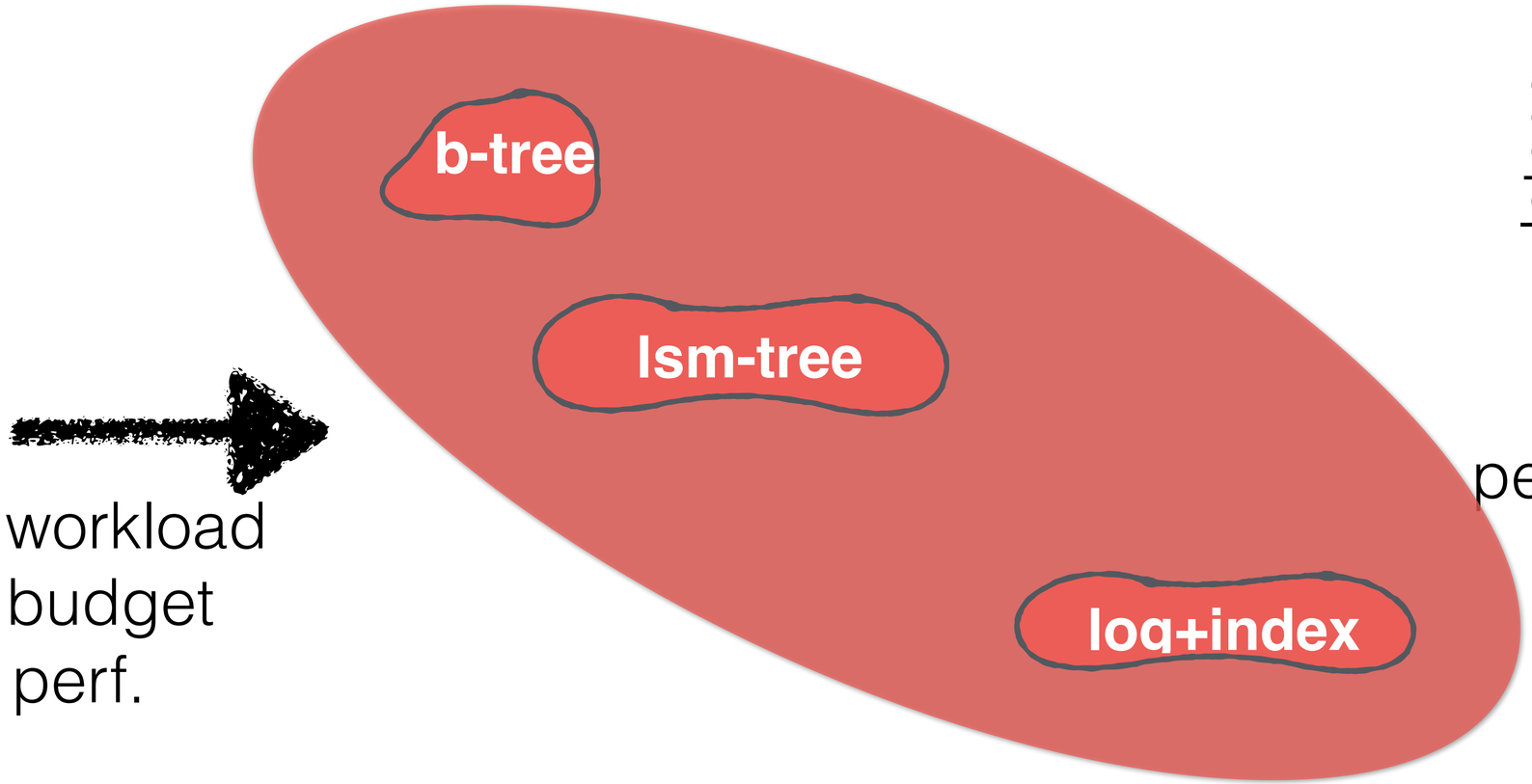
unified closed form

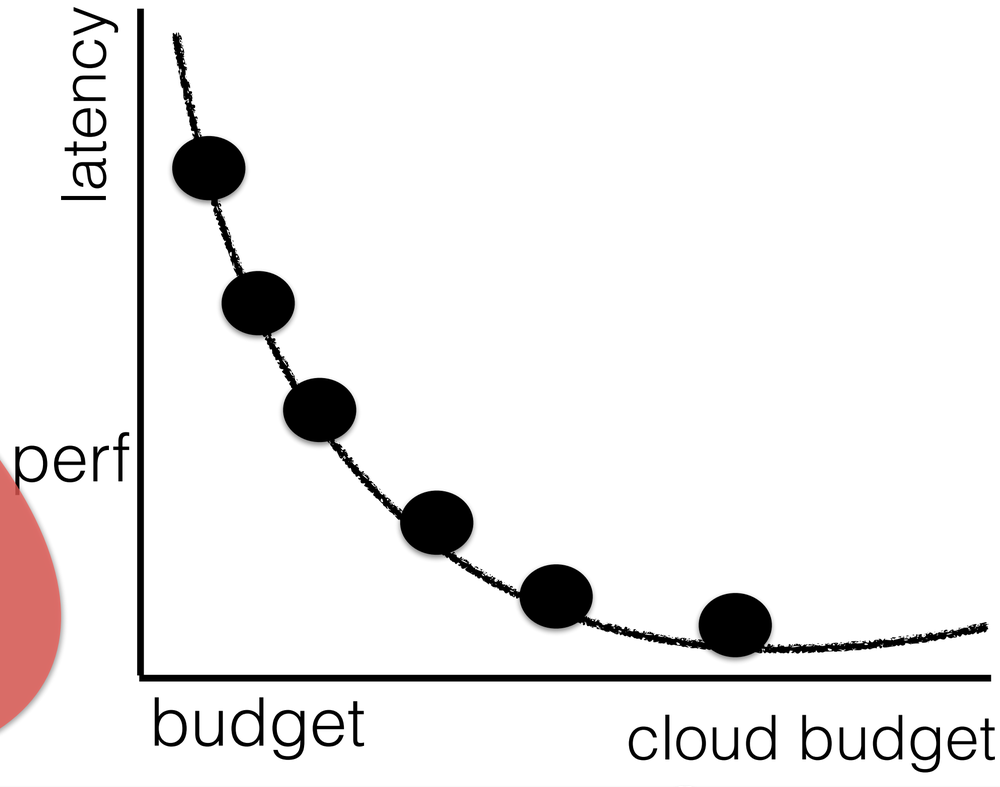
learned CPU model

h/w, parallelism

cloud cost mapping & SLA

AWS, Azure, Google





analytical I/O model

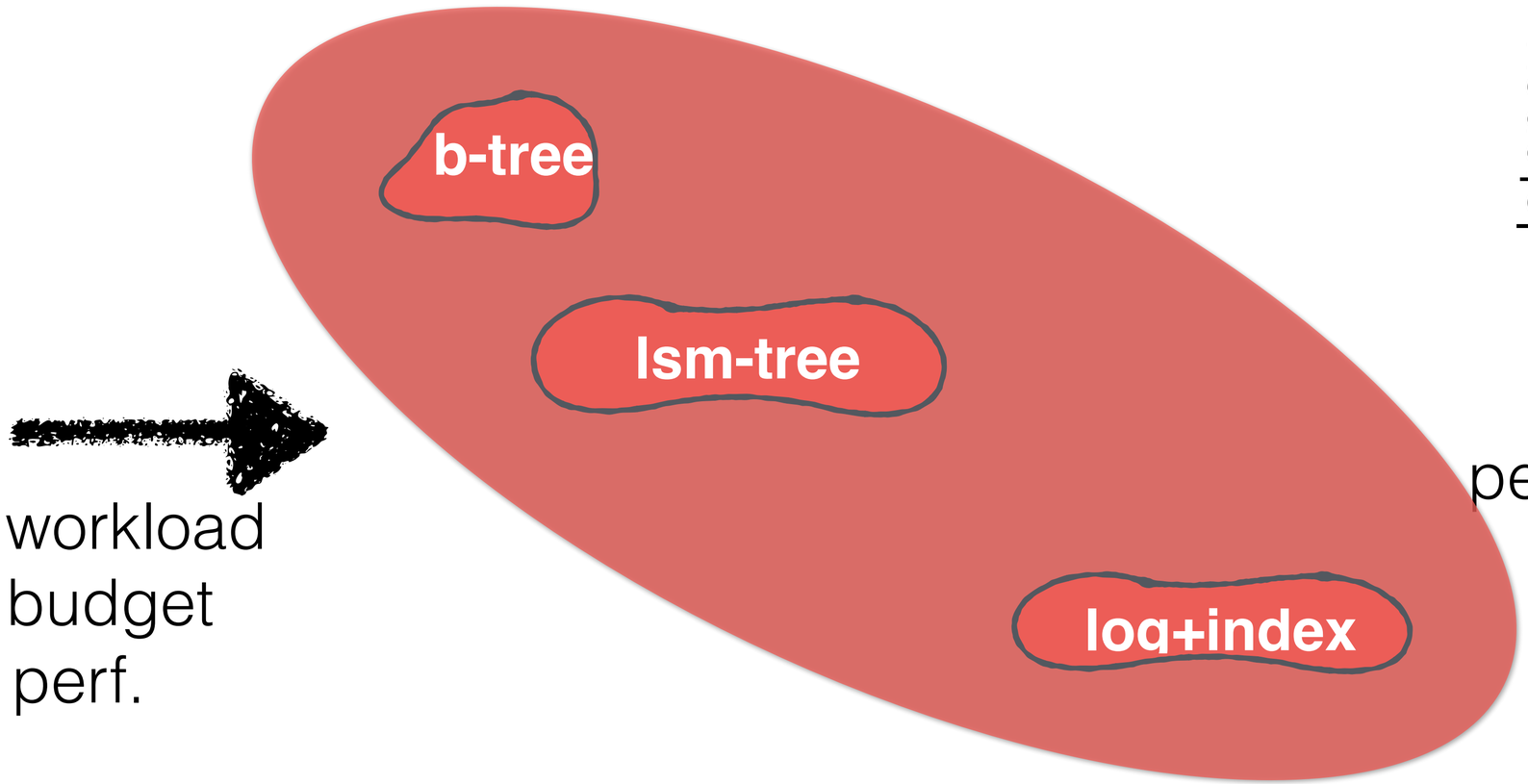
unified closed form

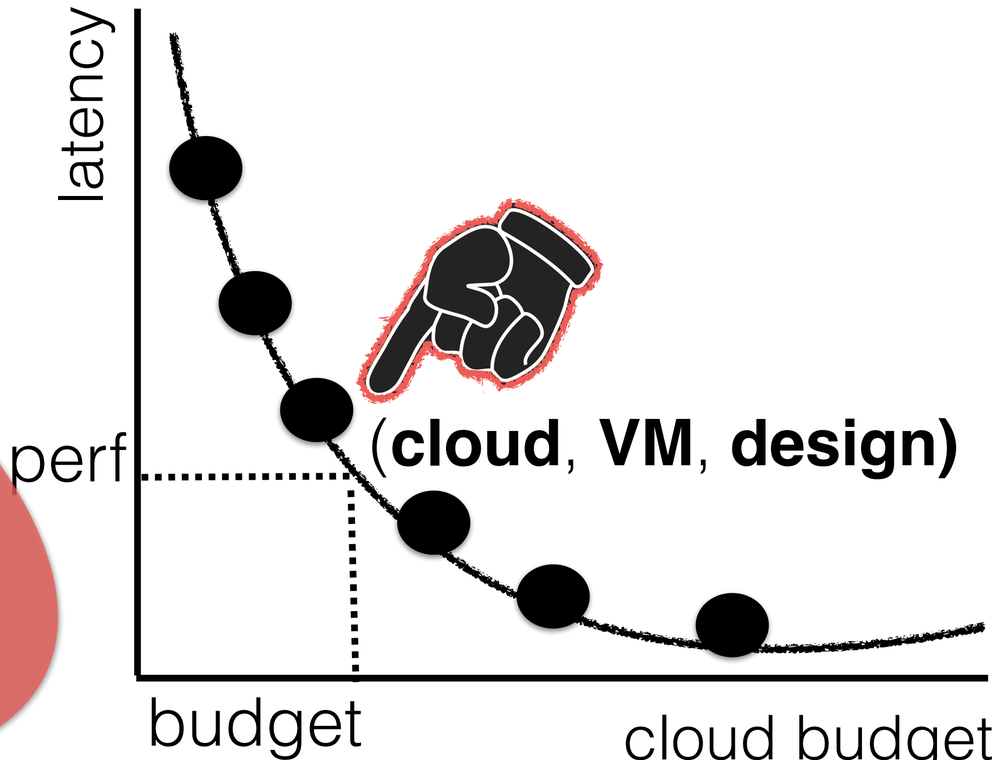
learned CPU model

h/w, parallelism

cloud cost mapping & SLA

AWS, Azure, Google





analytical I/O model

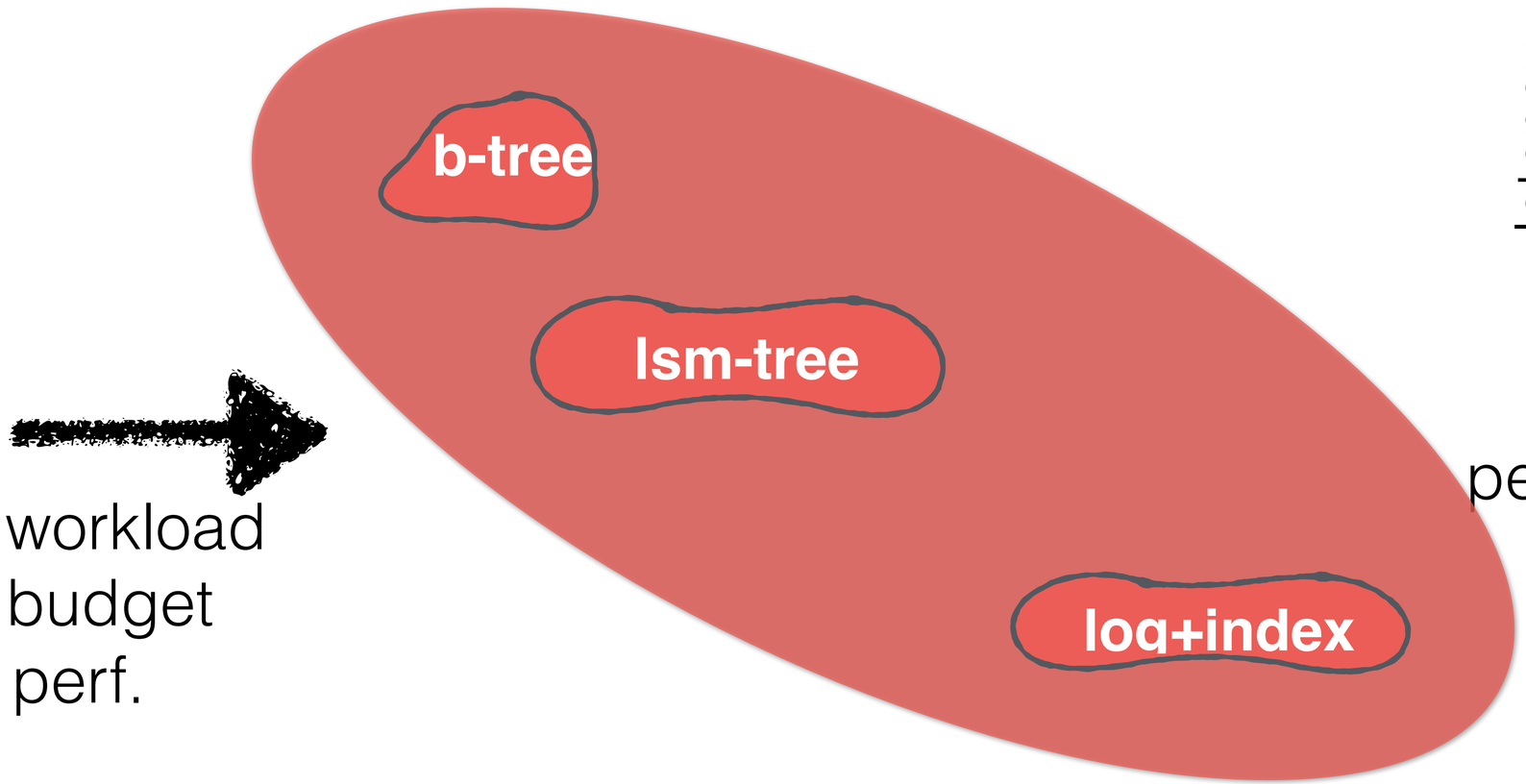
unified closed form

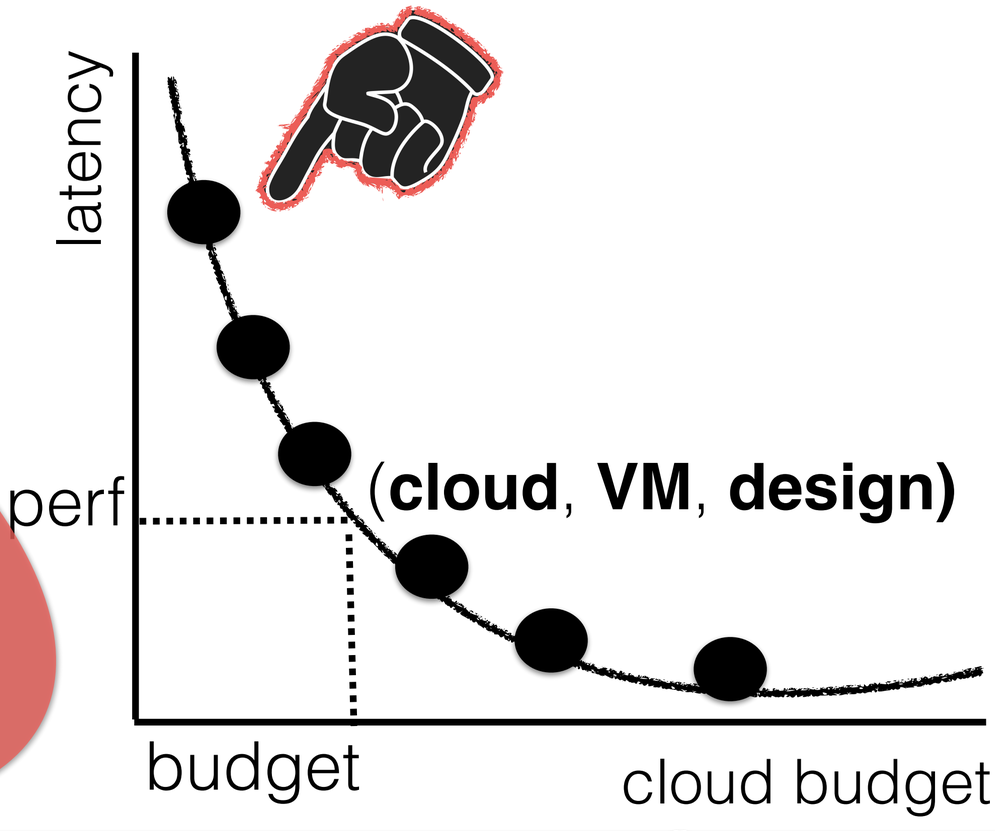
learned CPU model

h/w, parallelism

cloud cost mapping & SLA

AWS, Azure, Google





analytical I/O model

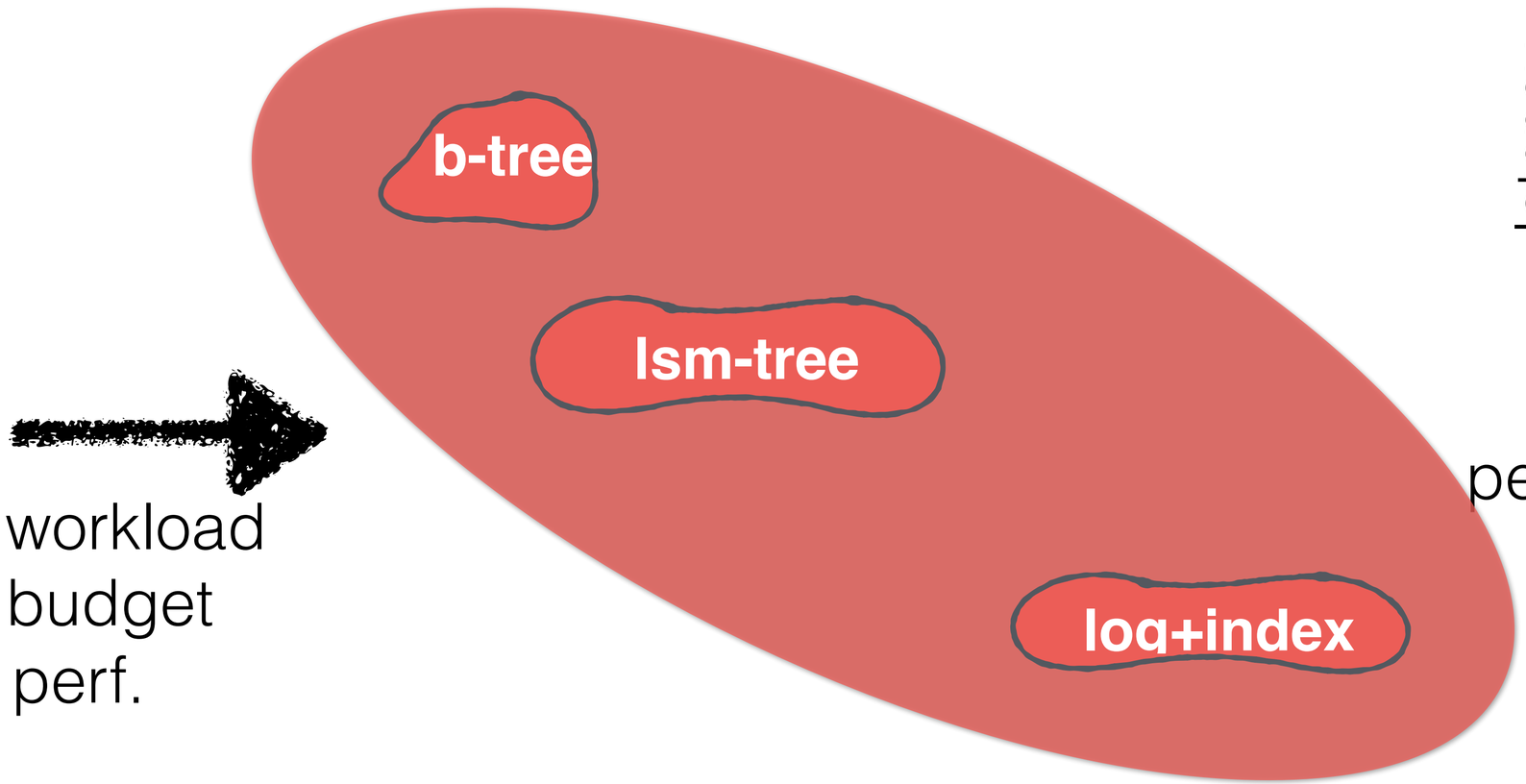
unified closed form

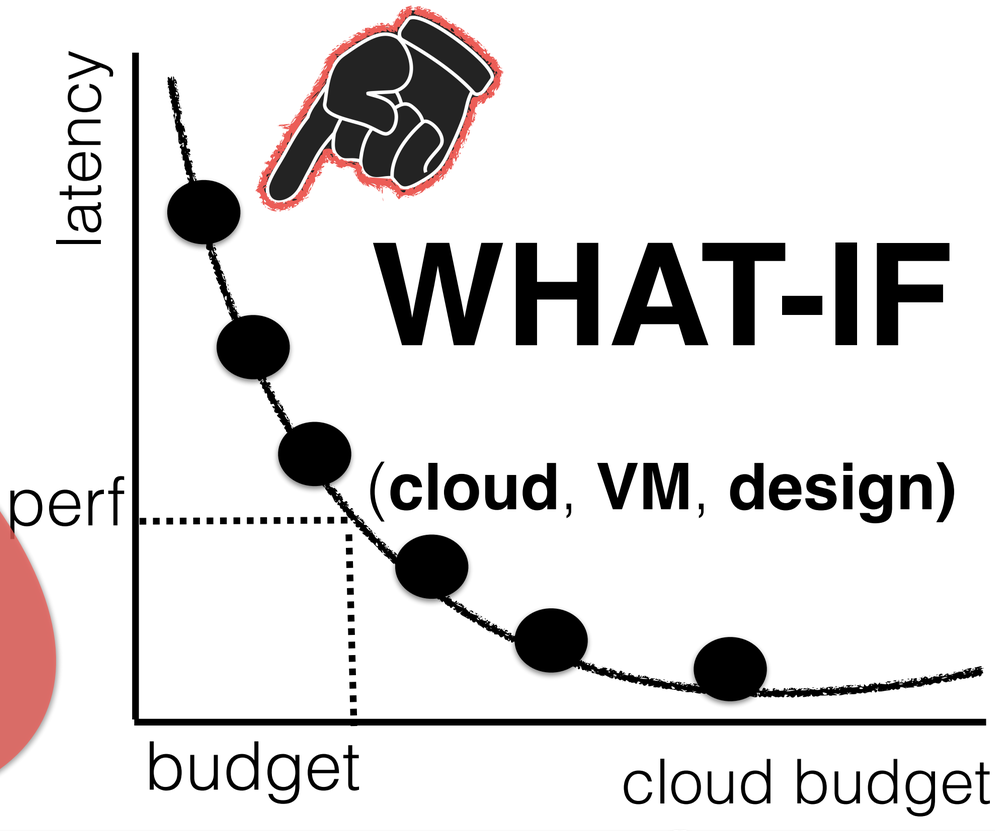
learned CPU model

h/w, parallelism

cloud cost mapping & SLA

AWS, Azure, Google





analytical I/O model

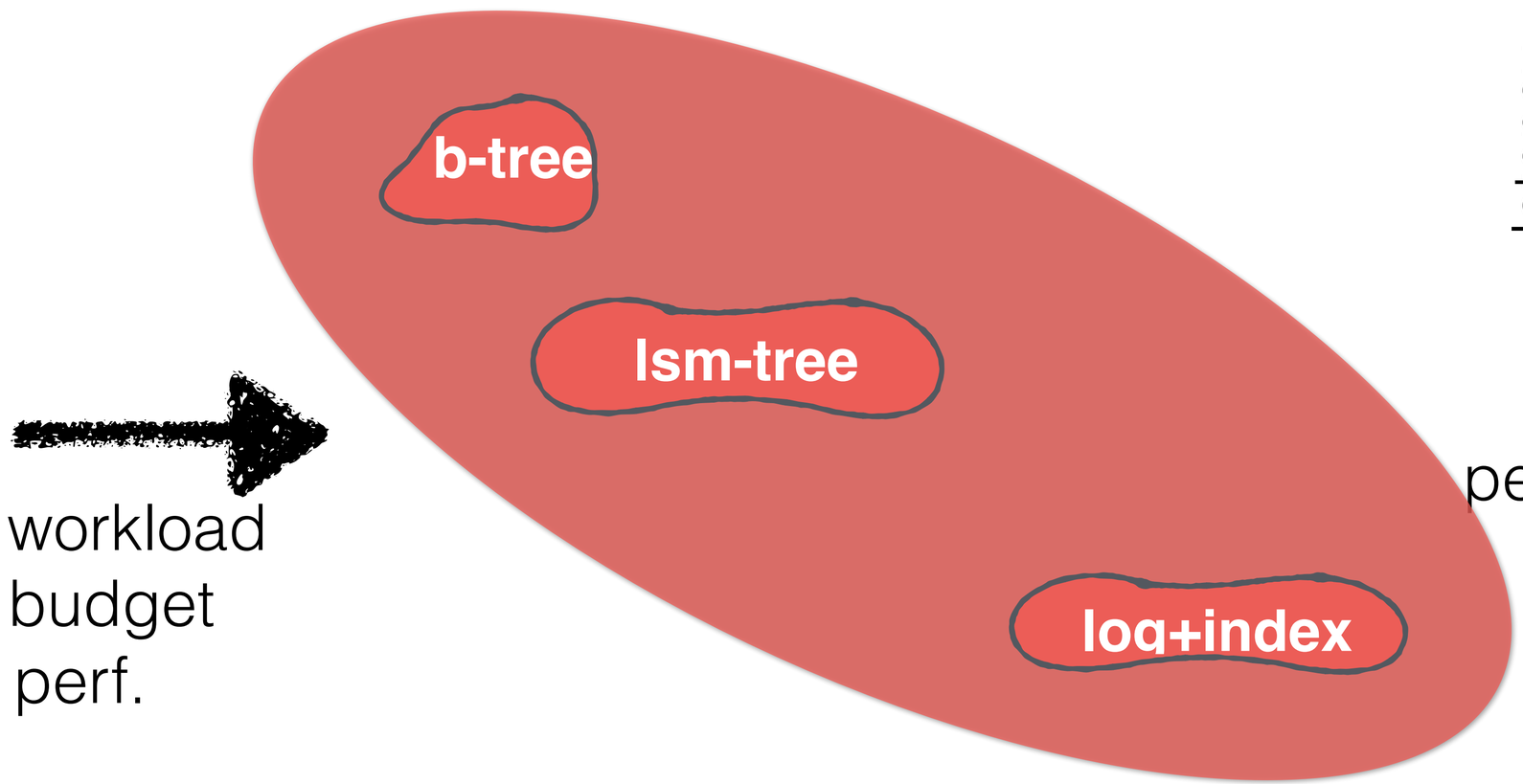
unified closed form

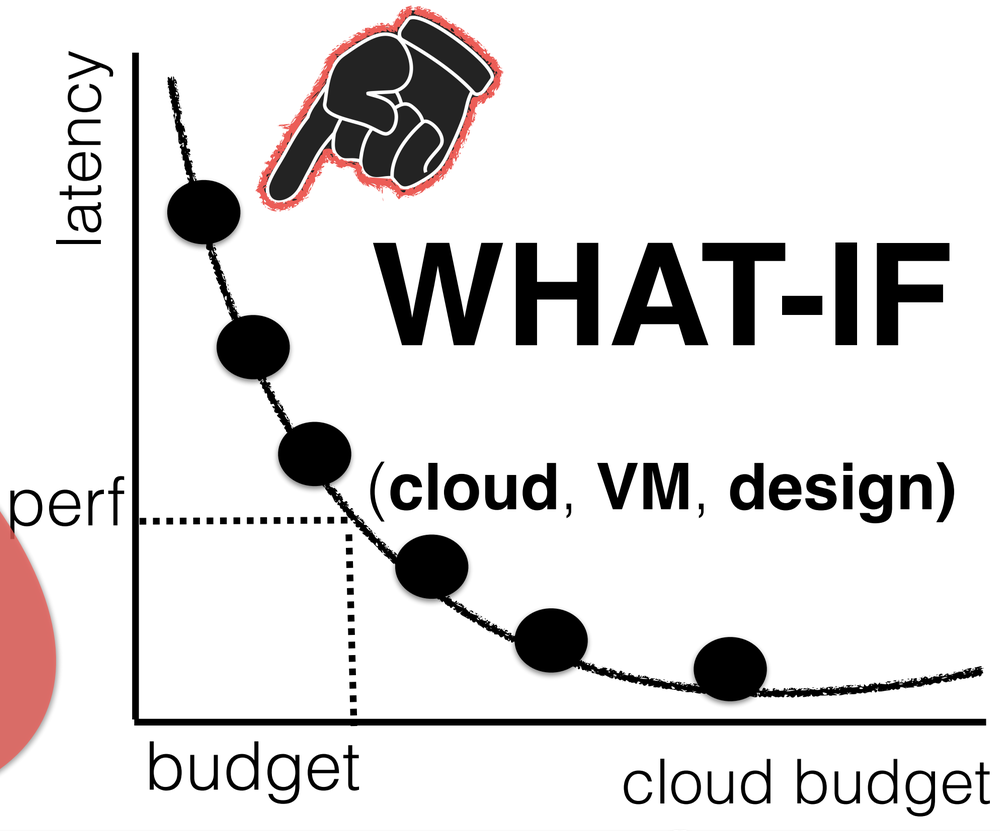
learned CPU model

h/w, parallelism

cloud cost mapping & SLA

AWS, Azure, Google





analytical I/O model

unified closed form

learned CPU model

h/w, parallelism

cloud cost mapping & SLA

AWS, Azure, Google

How to test?

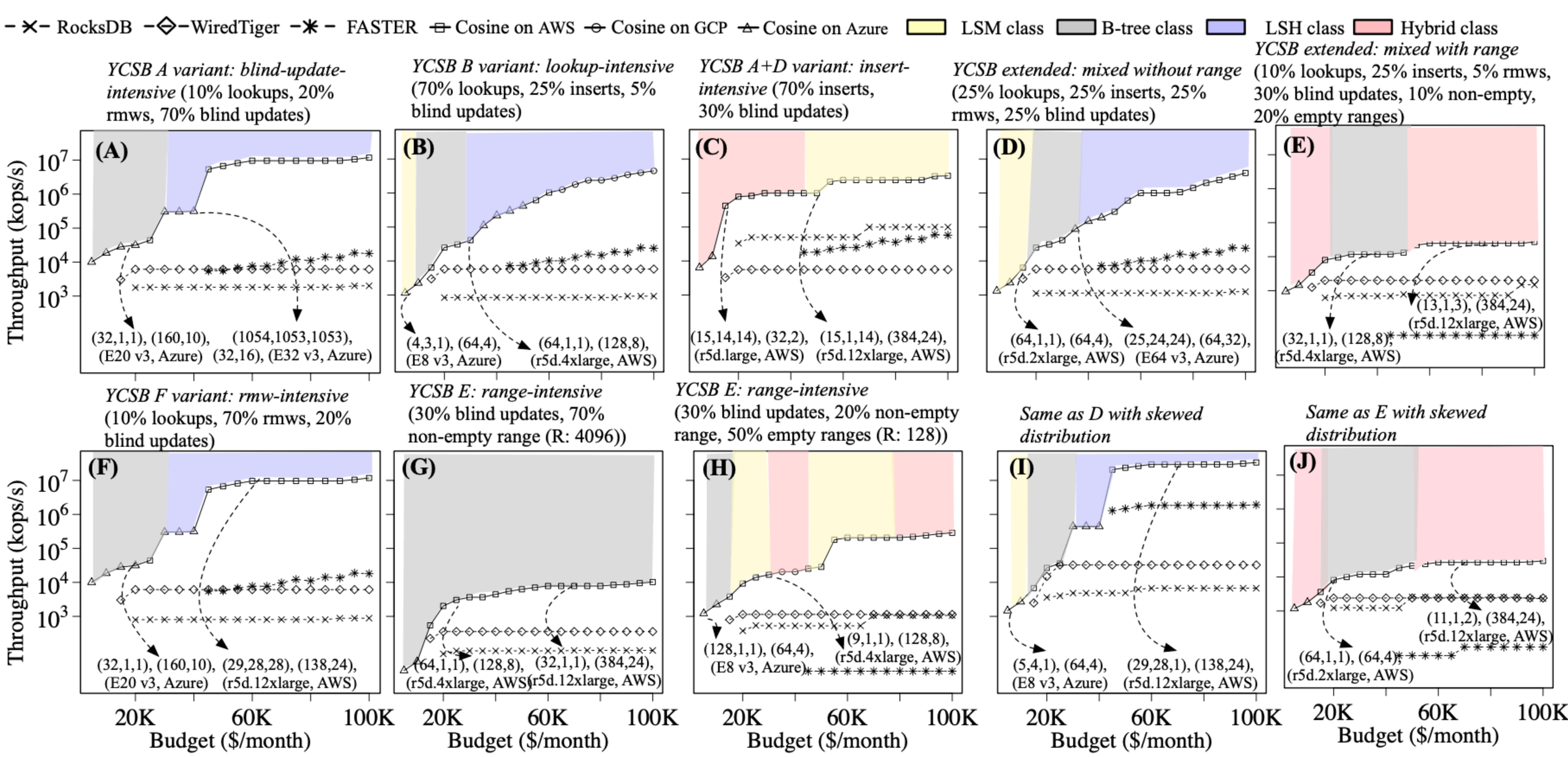
workload/budget diversity

How to test? workload/budget diversity

Can we beat the best system for every workload?

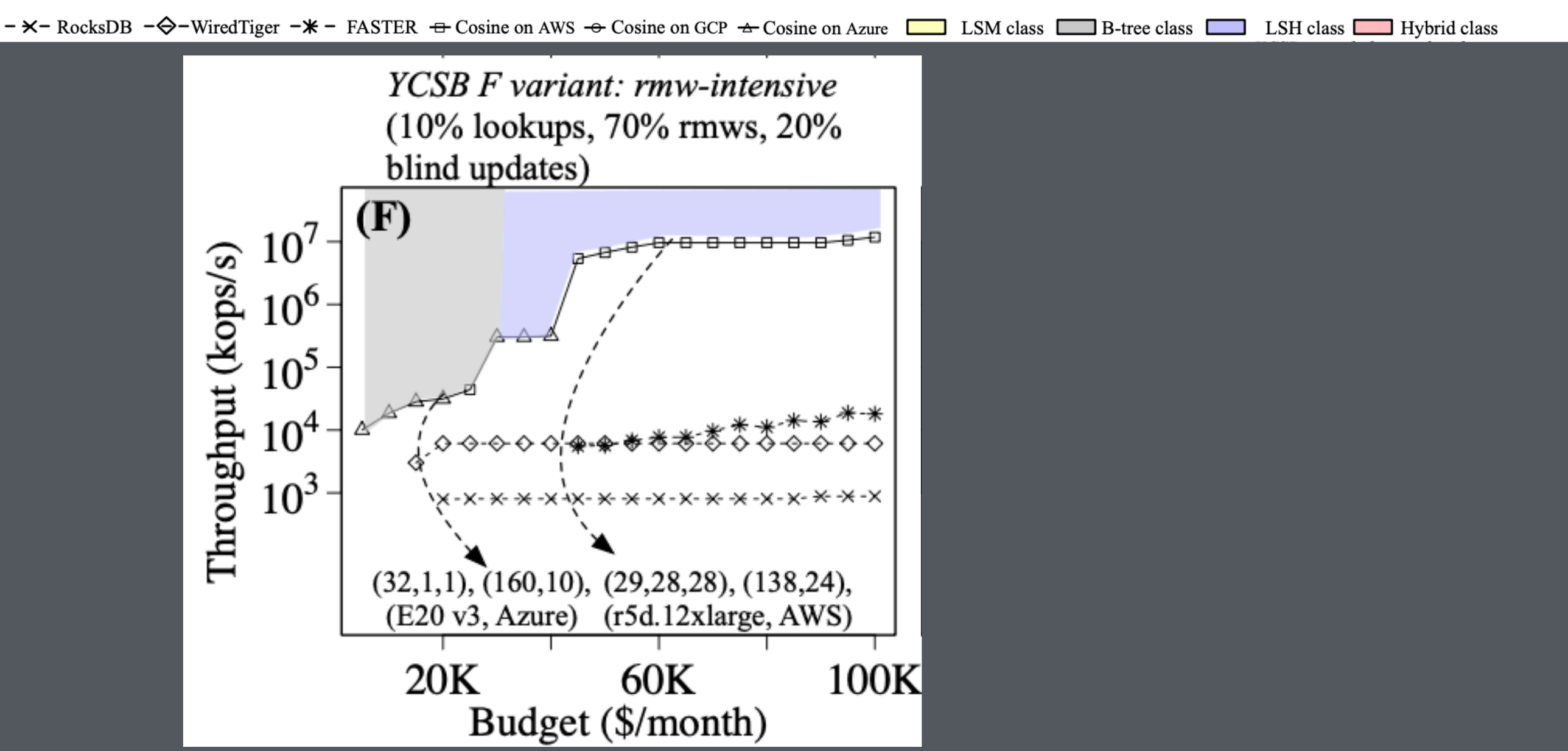
How to test? workload/budget diversity

Can we beat the best system for every workload?



workload/budget diversity

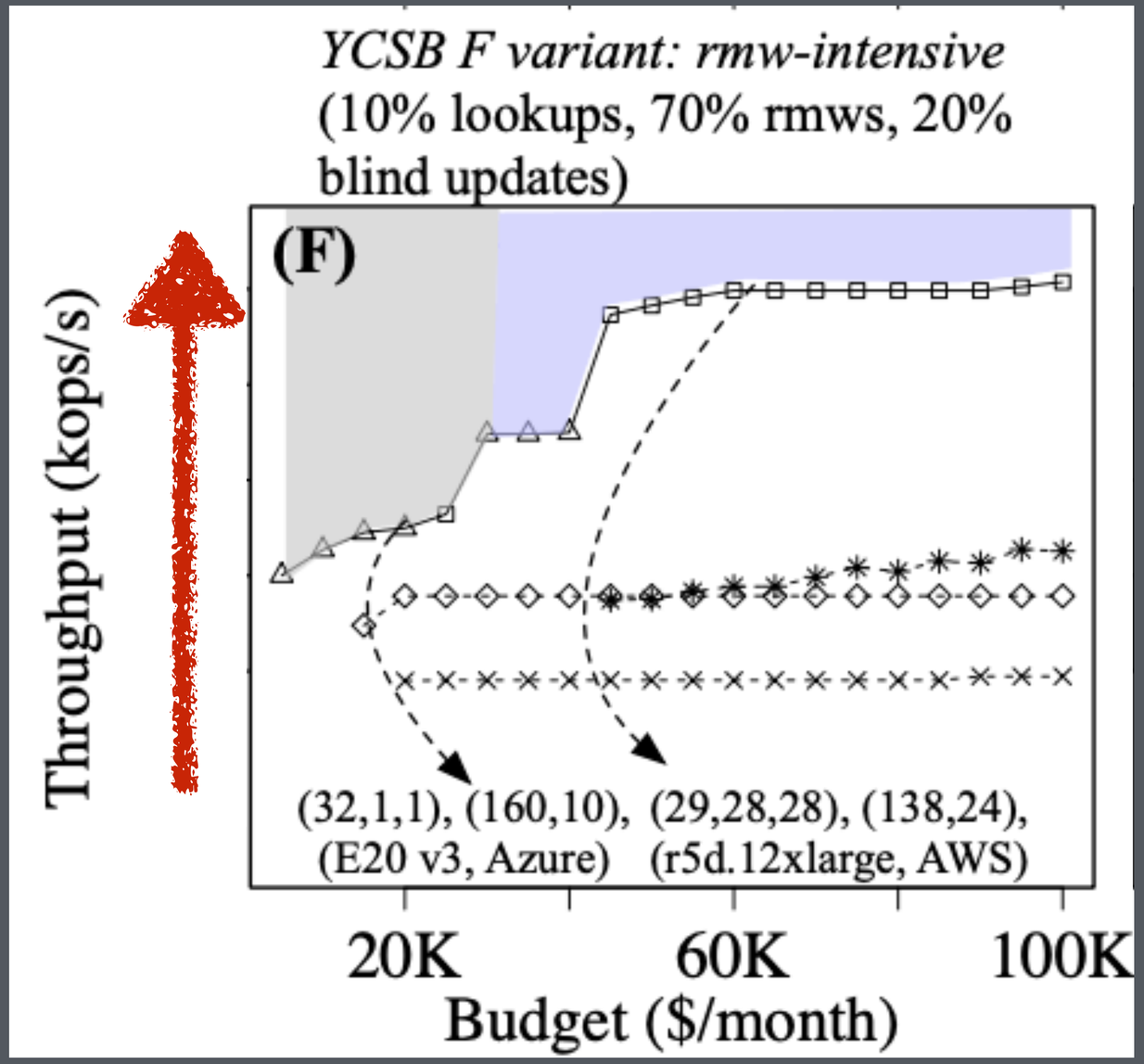
DASlab @ Harvard SEAS



DASlab @ Harvard SEAS

workload/budget diversity

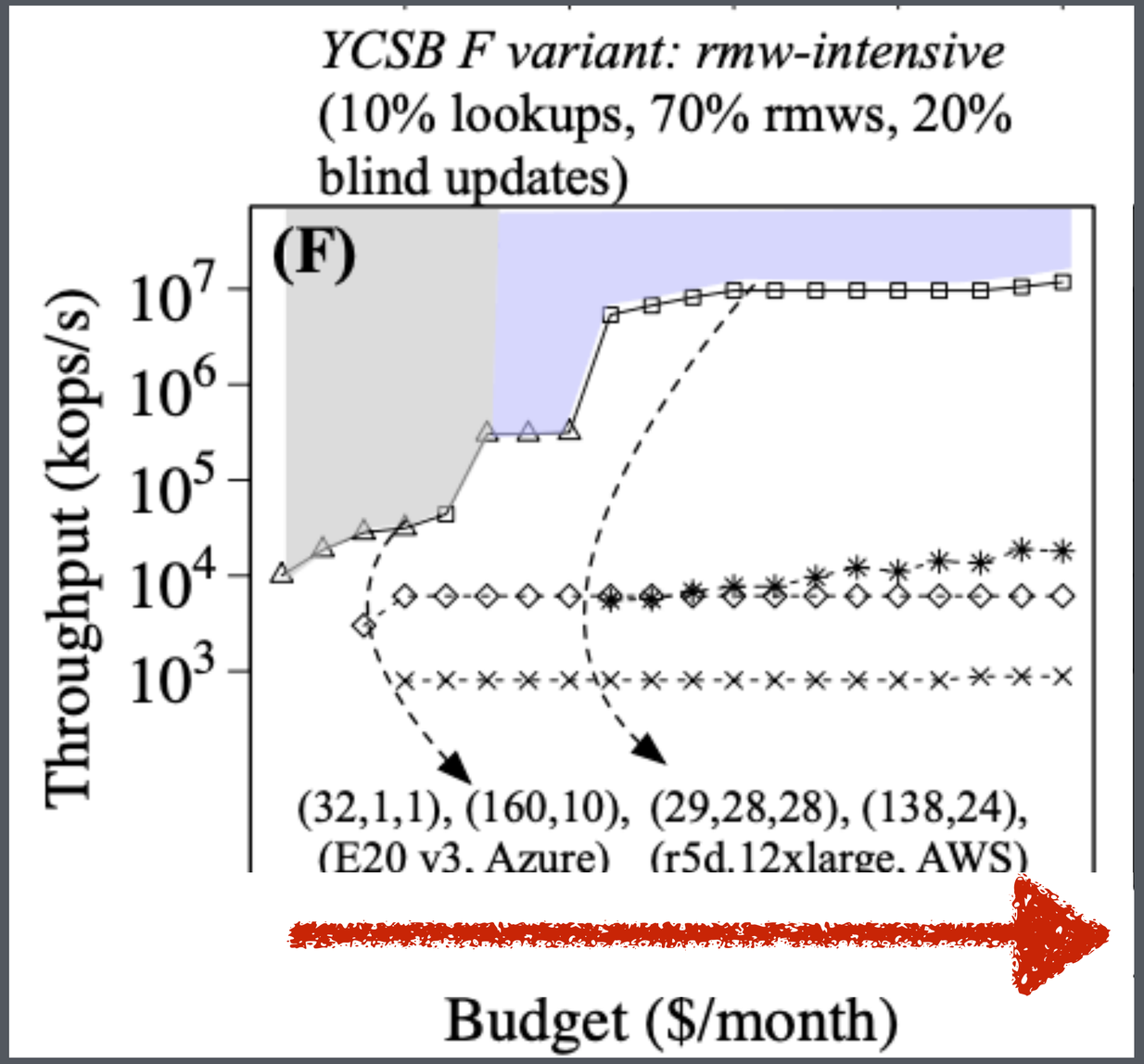
RocksDB WiredTiger FASTER Cosine on AWS Cosine on GCP Cosine on Azure LSM class B-tree class LSH class Hybrid class



workload/budget diversity

DASlab @ Harvard SEAS

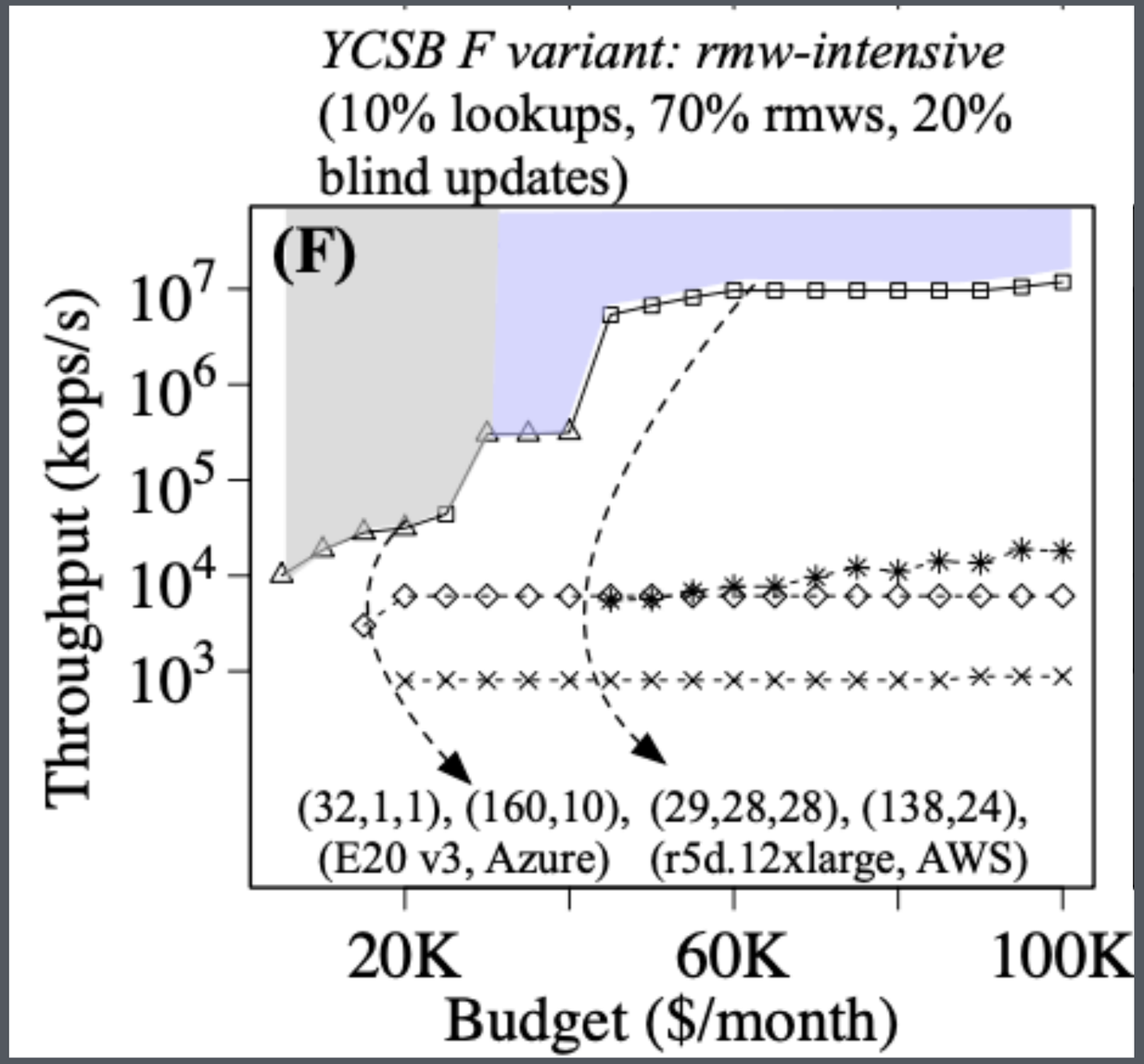
RocksDB WiredTiger FASTER Cosine on AWS Cosine on GCP Cosine on Azure LSM class B-tree class LSH class Hybrid class



DASlab @ Harvard SEAS

workload/budget diversity

RocksDB WiredTiger FASTER Cosine on AWS Cosine on GCP Cosine on Azure LSM class B-tree class LSH class Hybrid class

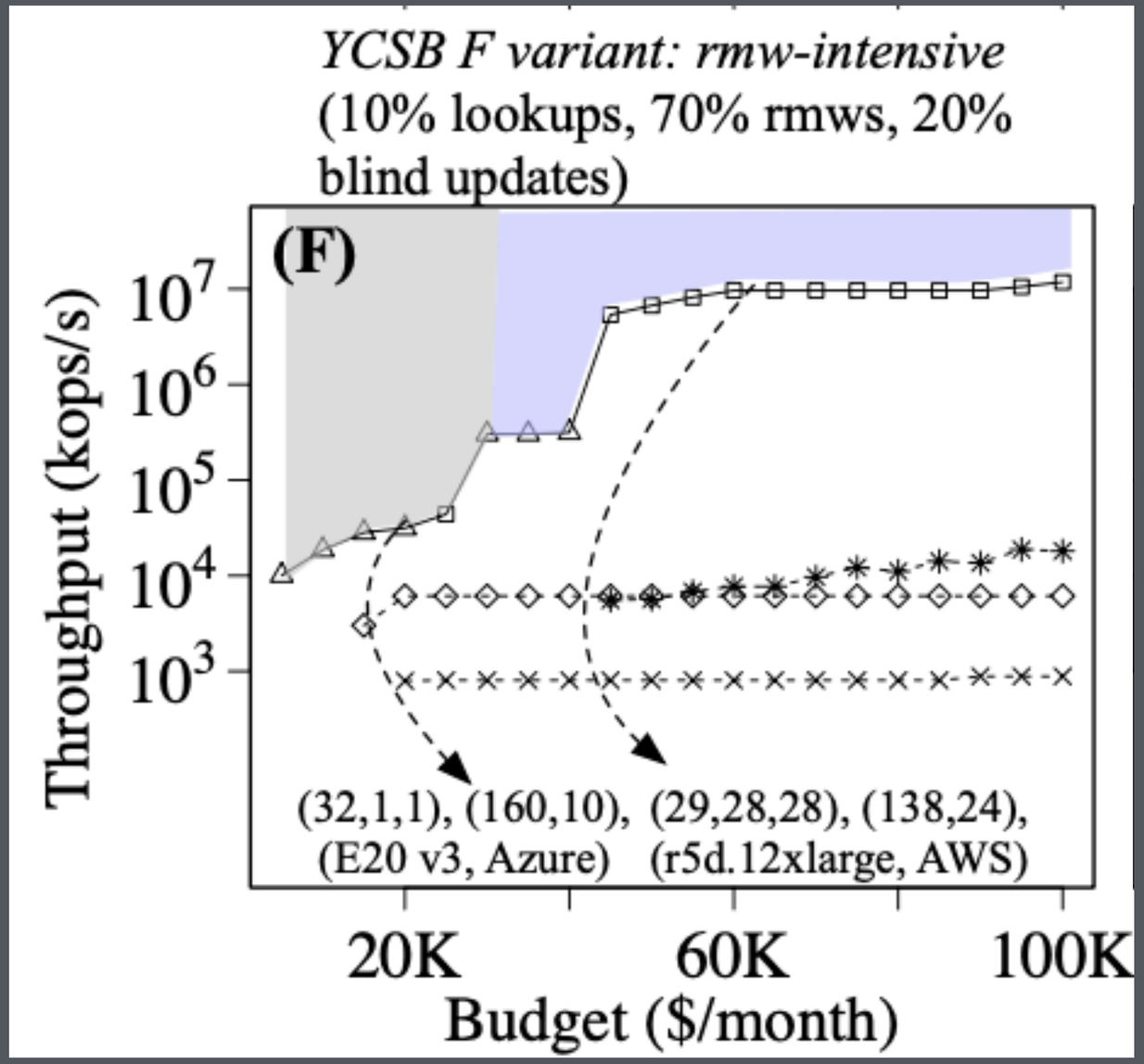


COSINE

workload/budget diversity

DASlab @ Harvard SEAS

RocksDB WiredTiger FASTER Cosine on AWS Cosine on GCP Cosine on Azure LSM class B-tree class LSH class Hybrid class



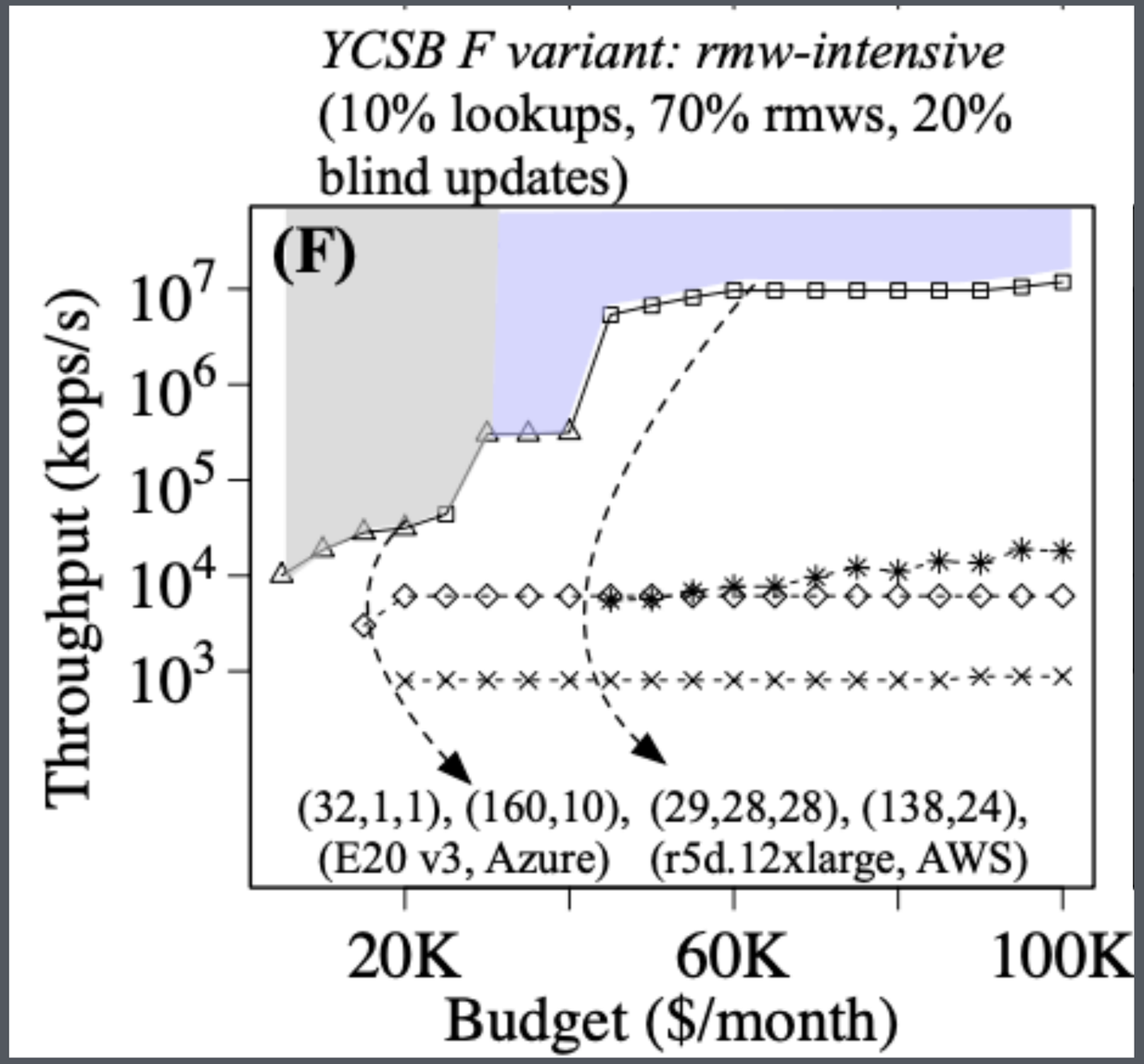
COSINE

{ State of the Art
Meta, Microsoft, Mongo

DASlab @ Harvard SEAS

workload/budget diversity

RocksDB WiredTiger FASTER Cosine on AWS Cosine on GCP Cosine on Azure LSM class B-tree class LSH class Hybrid class



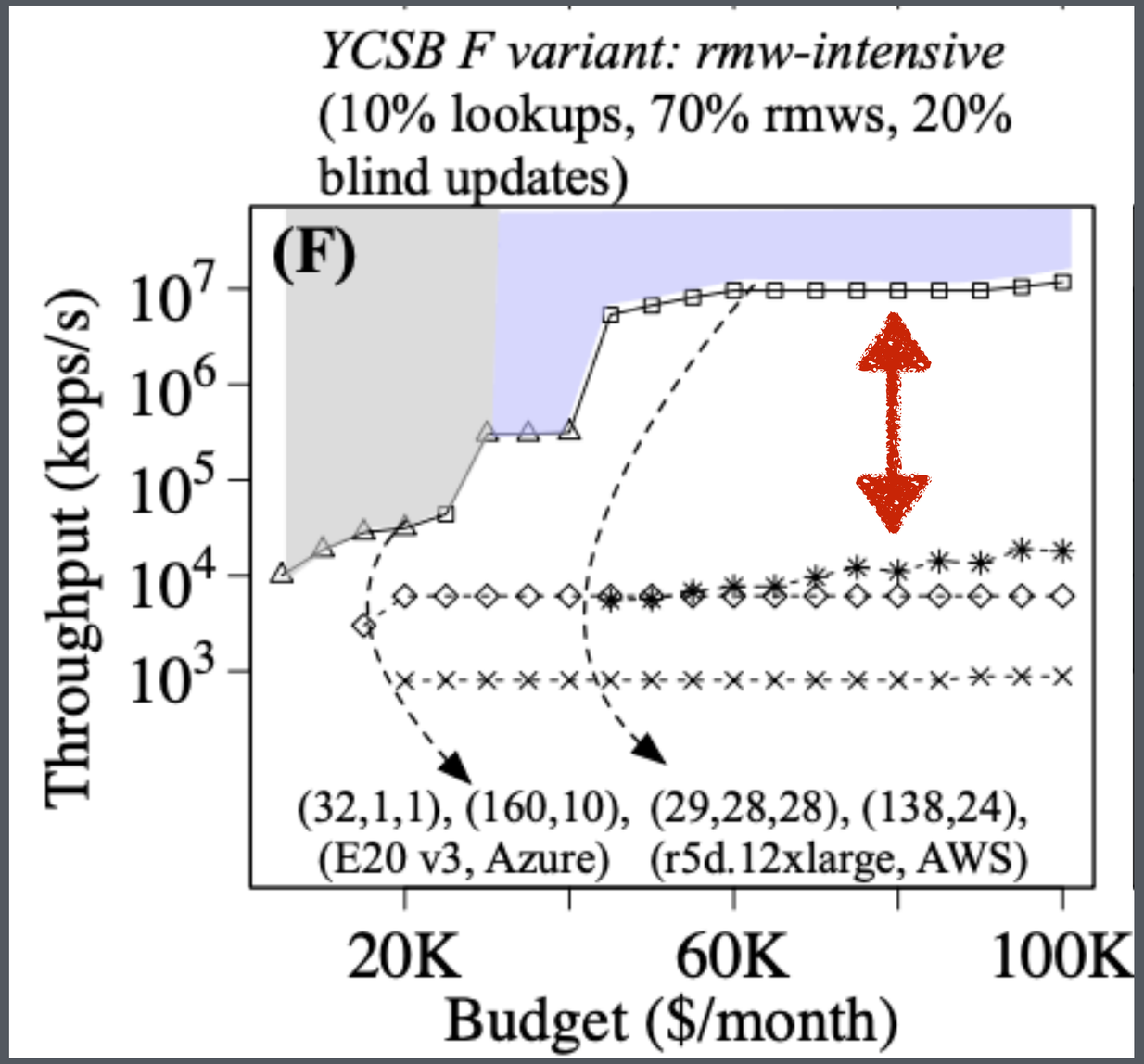
COSINE

{ State of the Art
Meta, Microsoft, Mongo

DASlab @ Harvard SEAS

workload/budget diversity

RocksDB WiredTiger FASTER Cosine on AWS Cosine on GCP Cosine on Azure LSM class B-tree class LSH class Hybrid class



COSINE

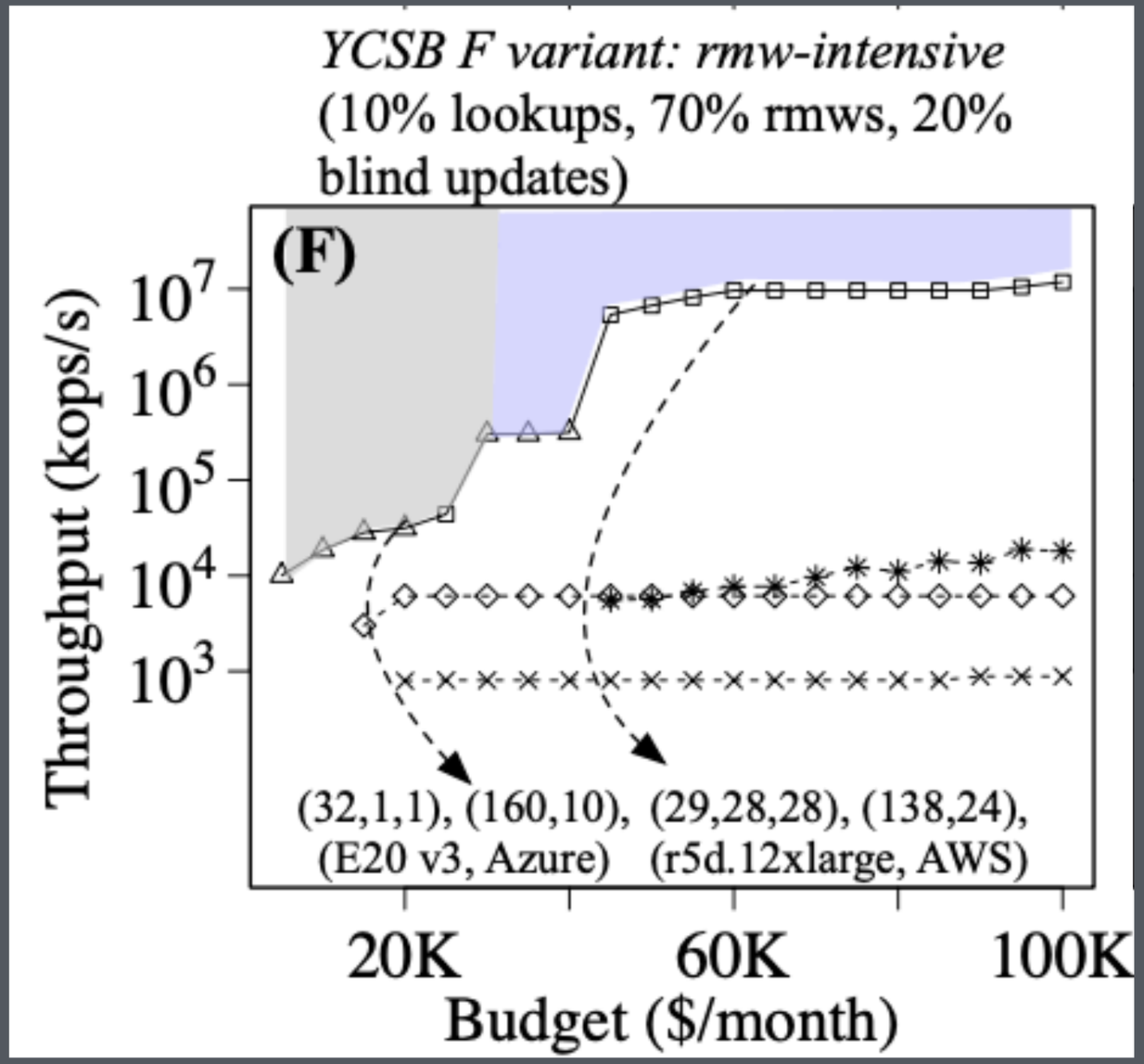
Better throughput/cost

{ State of the Art
Meta, Microsoft, Mongo

DASlab @ Harvard SEAS

workload/budget diversity

RocksDB WiredTiger FASTER Cosine on AWS Cosine on GCP Cosine on Azure LSM class B-tree class LSH class Hybrid class



COSINE

Better throughput/cost

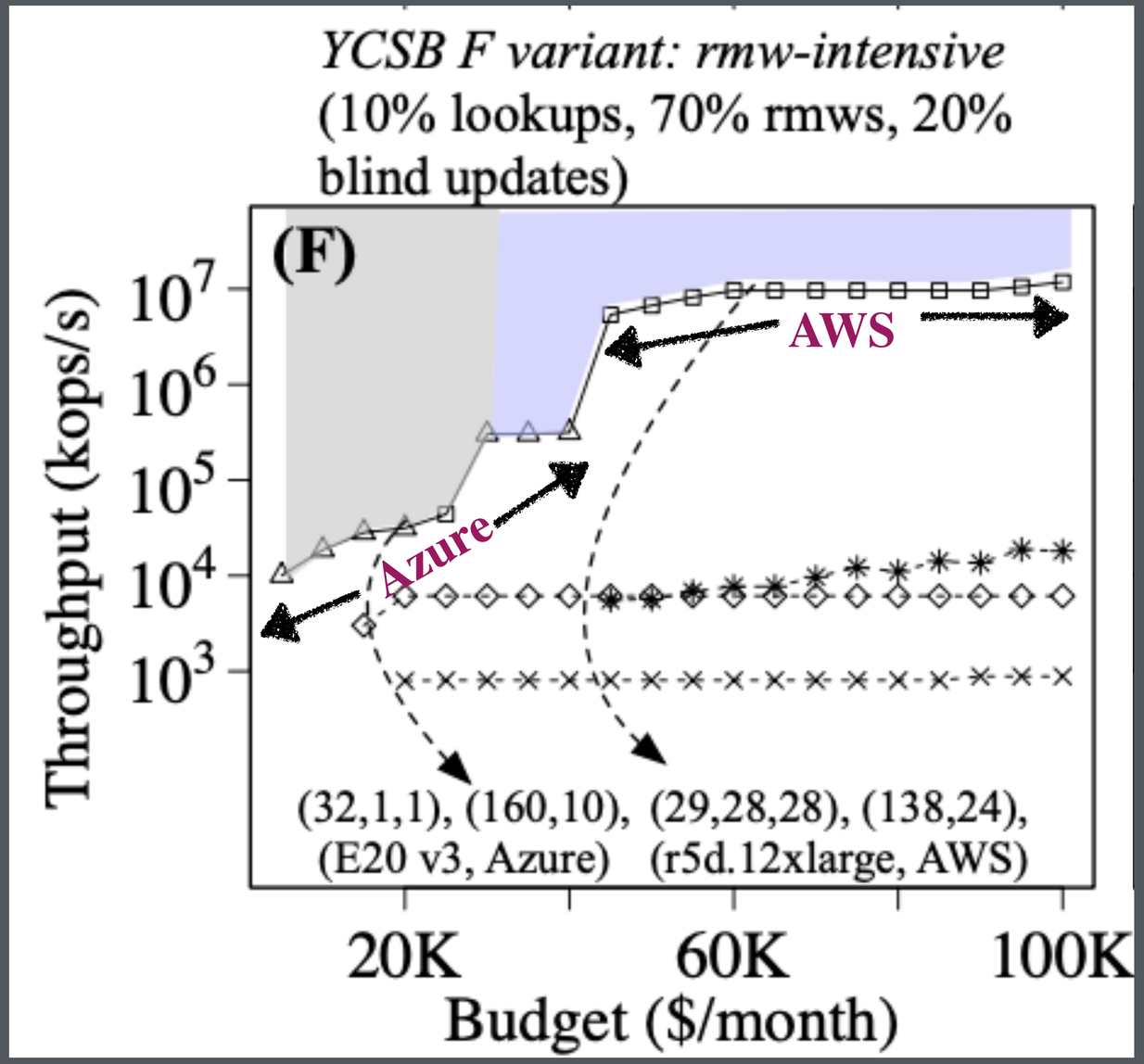
Self-designs

{ State of the Art
Meta, Microsoft, Mongo

DASlab @ Harvard SEAS

workload/budget diversity

RocksDB WiredTiger FASTER Cosine on AWS Cosine on GCP Cosine on Azure LSM class B-tree class LSH class Hybrid class



COSINE

Better throughput/cost

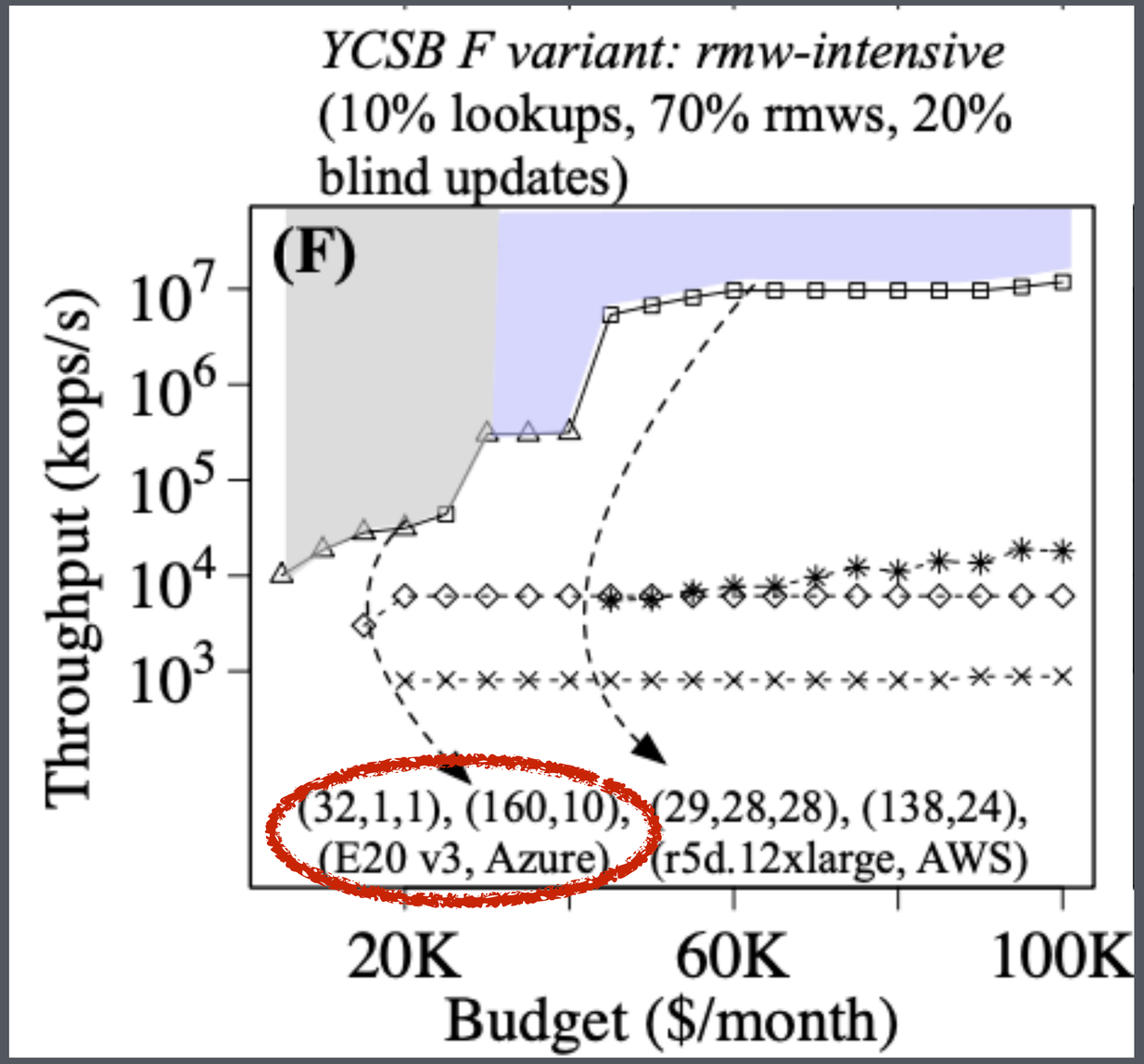
Self-designs

{ State of the Art
Meta, Microsoft, Mongo

DASlab @ Harvard SEAS

workload/budget diversity

RocksDB WiredTiger FASTER Cosine on AWS Cosine on GCP Cosine on Azure LSM class B-tree class LSH class Hybrid class



COSINE

Better throughput/cost

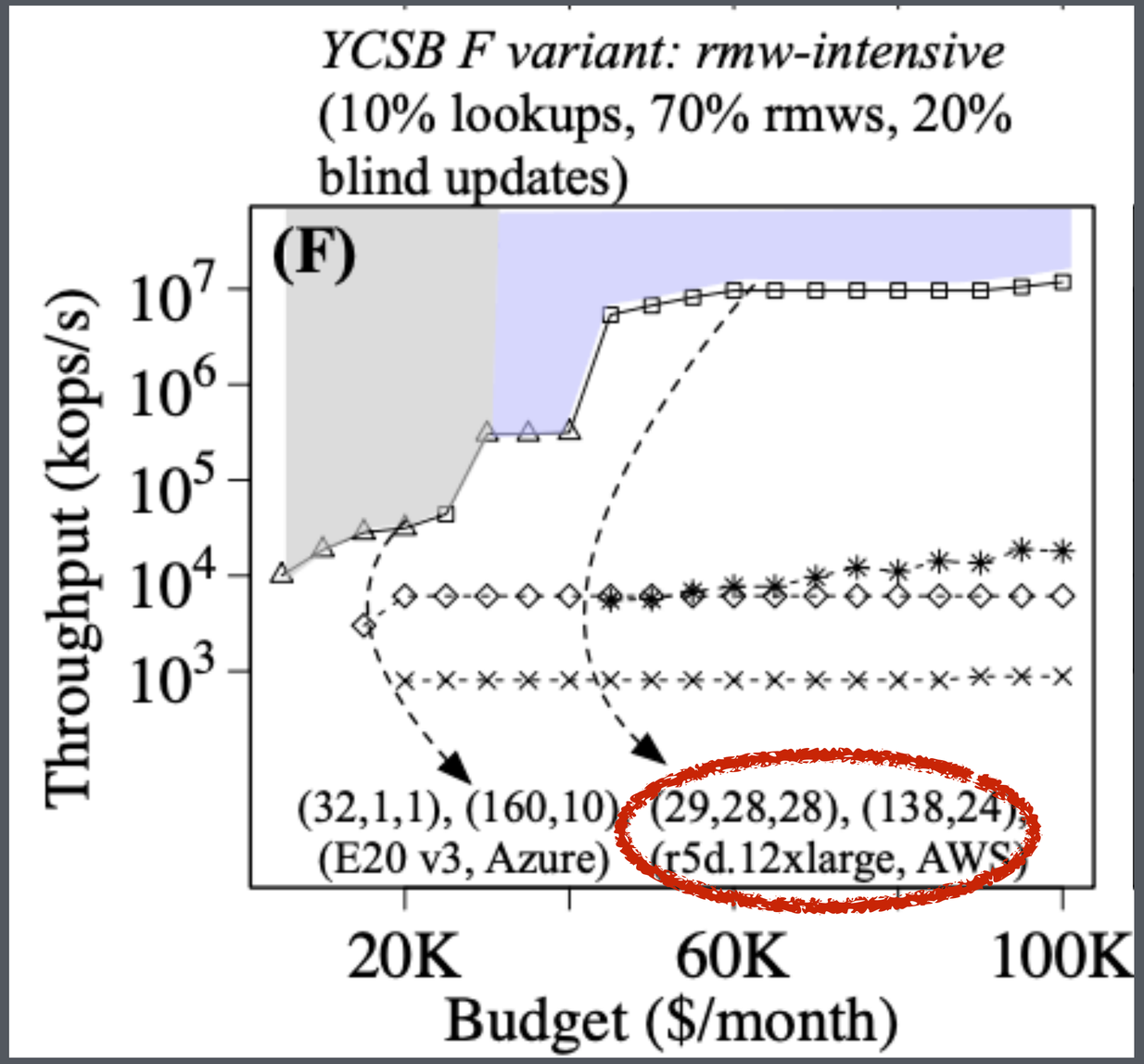
Self-designs

{ State of the Art
Meta, Microsoft, Mongo

DASlab @ Harvard SEAS

workload/budget diversity

RocksDB WiredTiger FASTER Cosine on AWS Cosine on GCP Cosine on Azure LSM class B-tree class LSH class Hybrid class



COSINE

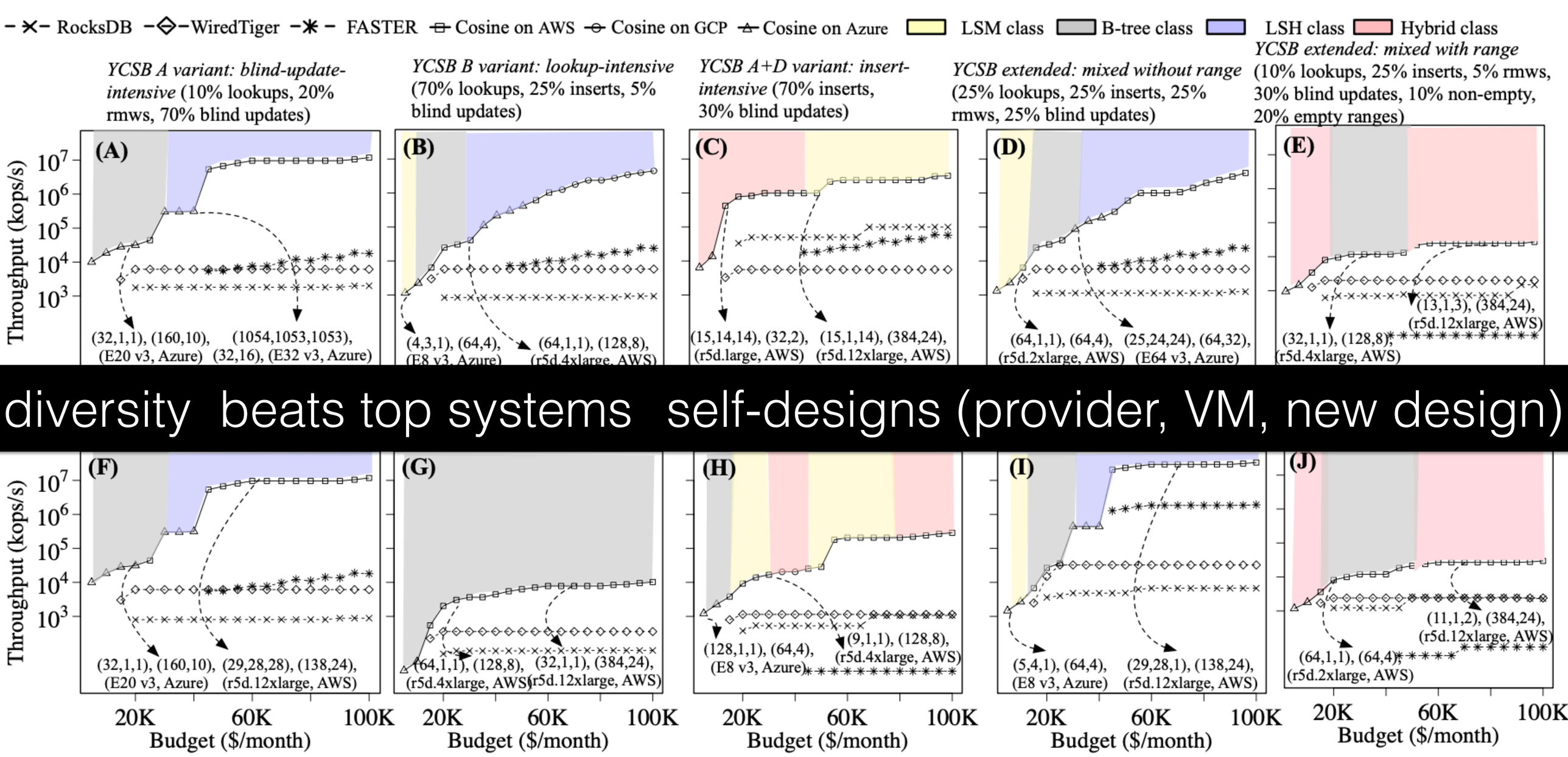
Better throughput/cost

Self-designs

{ State of the Art
Meta, Microsoft, Mongo

DASlab @ Harvard SEAS

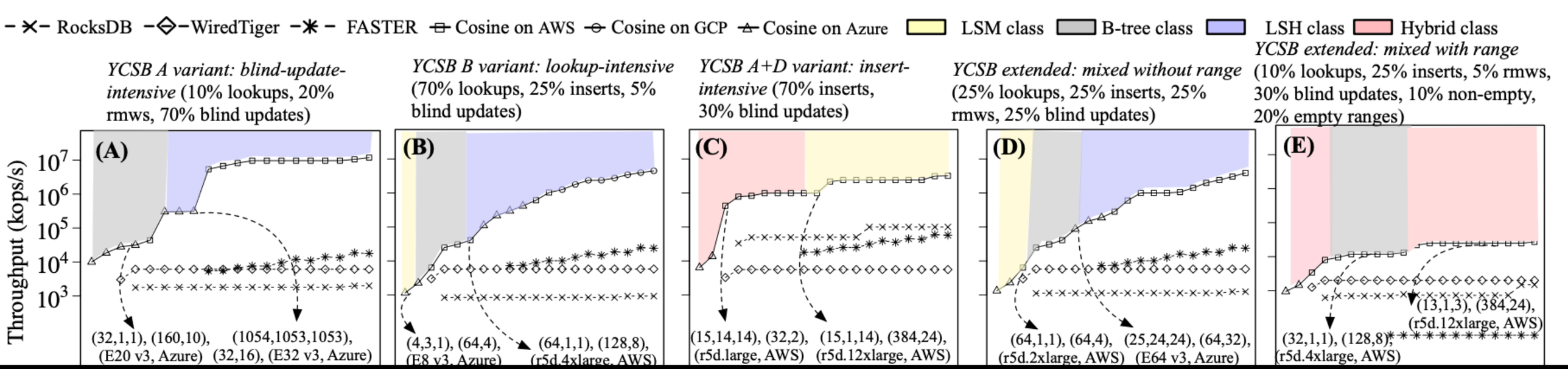
workload/budget diversity



diversity beats top systems self-designs (provider, VM, new design)

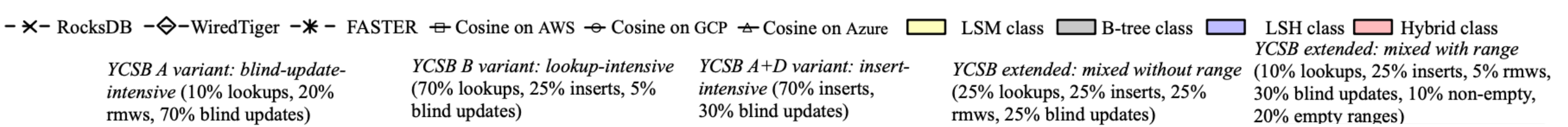
DASlab @ Harvard SEAS

workload/budget diversity



diversity beats top systems self-designs (provider, VM, new design)

Cosine achieves the best perf. across all workloads by automatically designing a new system every time.



Microsoft Microsoft Meta Microsoft mongo

diversity beats top systems self-designs (provider, VM, new design)

Microsoft mongo Meta Microsoft Meta

Cosine achieves the best perf. across all workloads by automatically designing a new system every time.

We can automatically design 1000x faster new NoSQL systems

1) design space
2) navigation (math/ML)
3) code generation

Papers: **Cosine** PVLDB 2023, and new **Limousine** at SIGMOD 2024

We can automatically design 1000x faster new NoSQL systems

1) design space
2) navigation (math/ML)
3) code generation

Papers: **Cosine** PVLDB 2023, and new **Limousine** at SIGMOD 2024

How do these concepts translate to the other big data areas

neural networks, image AI, Blockchain, …?

We can automatically design 1000x faster new NoSQL systems

1) design space
2) navigation (math/ML)
3) code generation

Papers: **Cosine** PVLDB 2023, and new **Limousine** at SIGMOD 2024

How do these concepts translate to the other big data areas

neural networks, image AI, Blockchain, …?

again, it all starts from the storage design space

Limousine: Blending Learned and Classical Indexes to Self-Design Larger-than-Memory Cloud Storage Engines.
Subarna Chatterjee, Mark F. Pekala, Lev Kruglyak, and Stratos Idreos.
In Proceedings of the ACM Management of Data 2, 1, Article 47 (February 2024), (SIGMOD), 2024

Cosine: A Cloud-Cost Optimized Self-Designing Key-Value Storage Engine.
Subarna Chatterjee, Meena Jagadeesan, Wilson Qin, and Stratos Idreos.
In Proceedings of the Very Large Databases Endowment, (PVLDB), 2022

DASlab
@ Harvard SEAS

The Image Calculator: 10x Faster Image-AI Inference by Replacing JPEG with Self-designing Storage Format.

Utku Sirin and Stratos Idreos.

In Proceedings of the ACM Management of Data 2, 1, Article 52 (February 2024), (SIGMOD), 2024

DASlab @ Harvard SEAS

Project 1:
What are the boundaries between LSM-trees and B-trees?
Can we have a single strictly better design for big data NoSQL?

Project 2:
Can we design new systems using the Cosine/Limousine design space and cost/algo synthesis and an LLM?

CS 265

Stratos Idreos

BIG DATA SYSTEMS

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