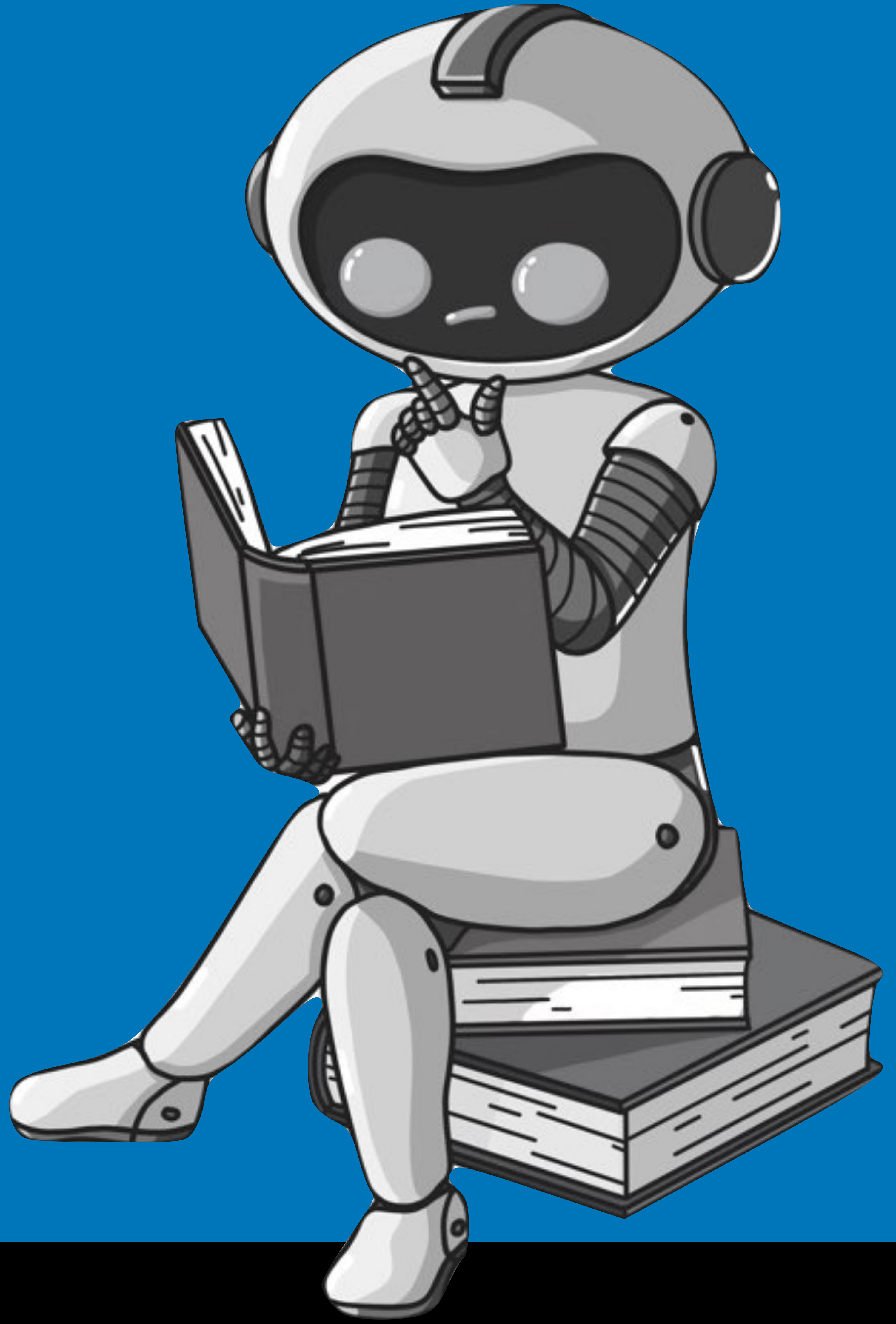
SELF-DESIGNING DATA SYSTEMS FOR THE AI ERA

Stratos Idreos

VE RI TAS

DASlab @ Harvard SEAS



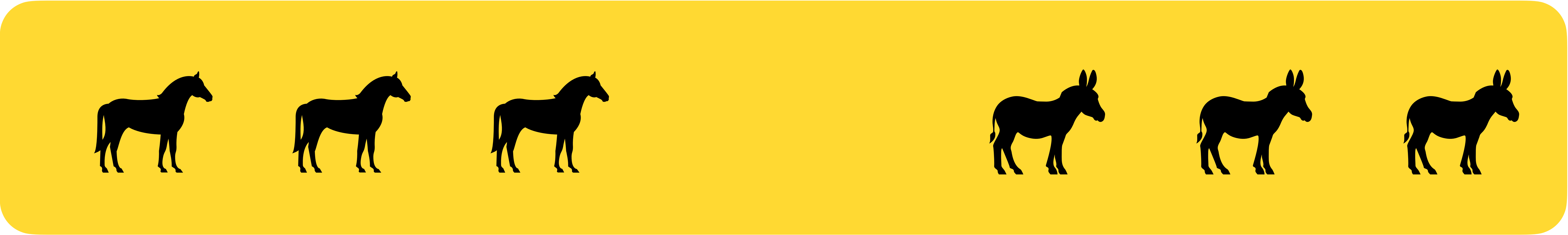
*What if we can **reason** about systems design?*

What is a data system?

Why do we need self-designing systems?

A TYPICAL BIG DATA TASK

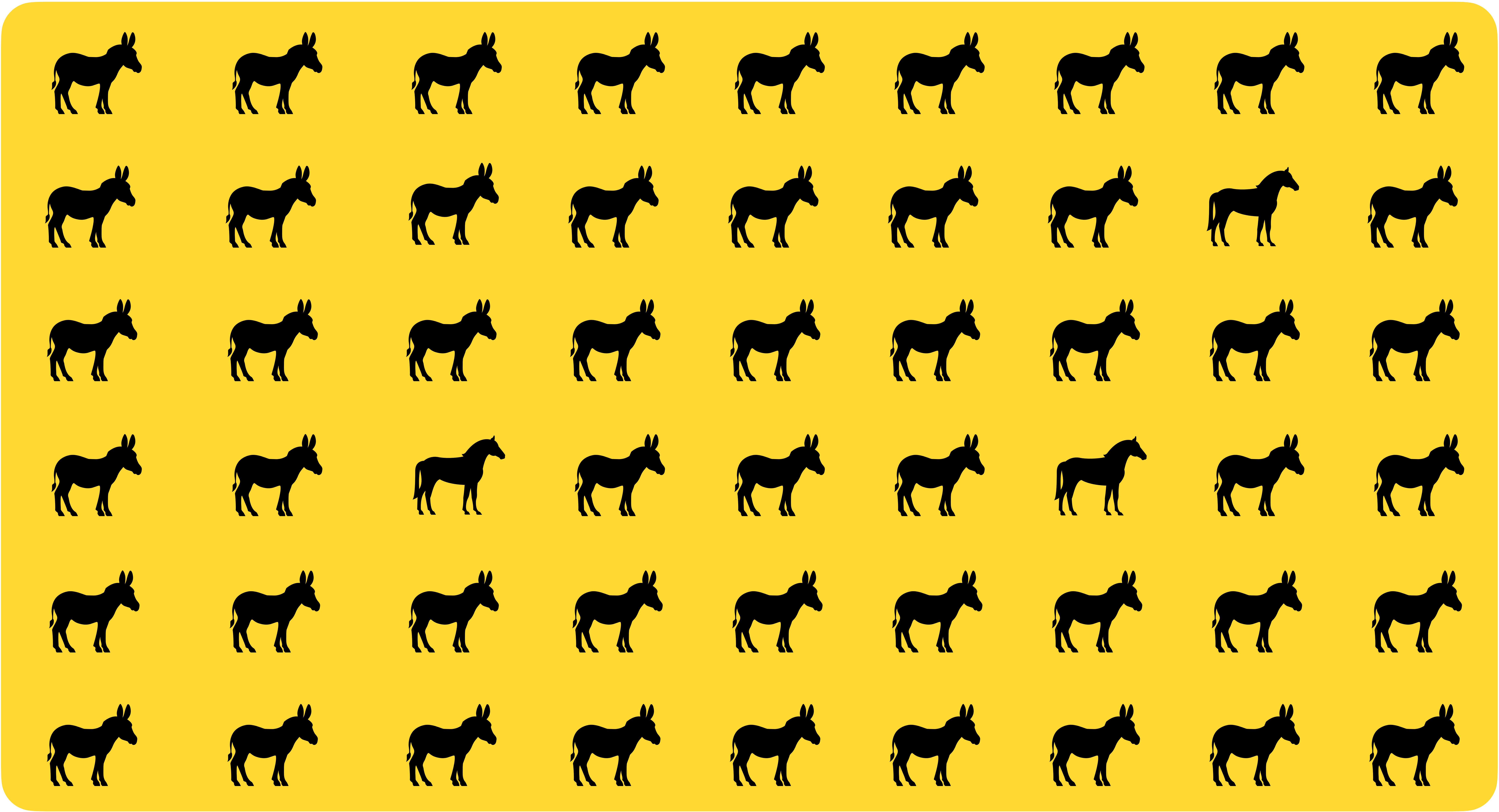
image analysis: e.g., detect the number of horses

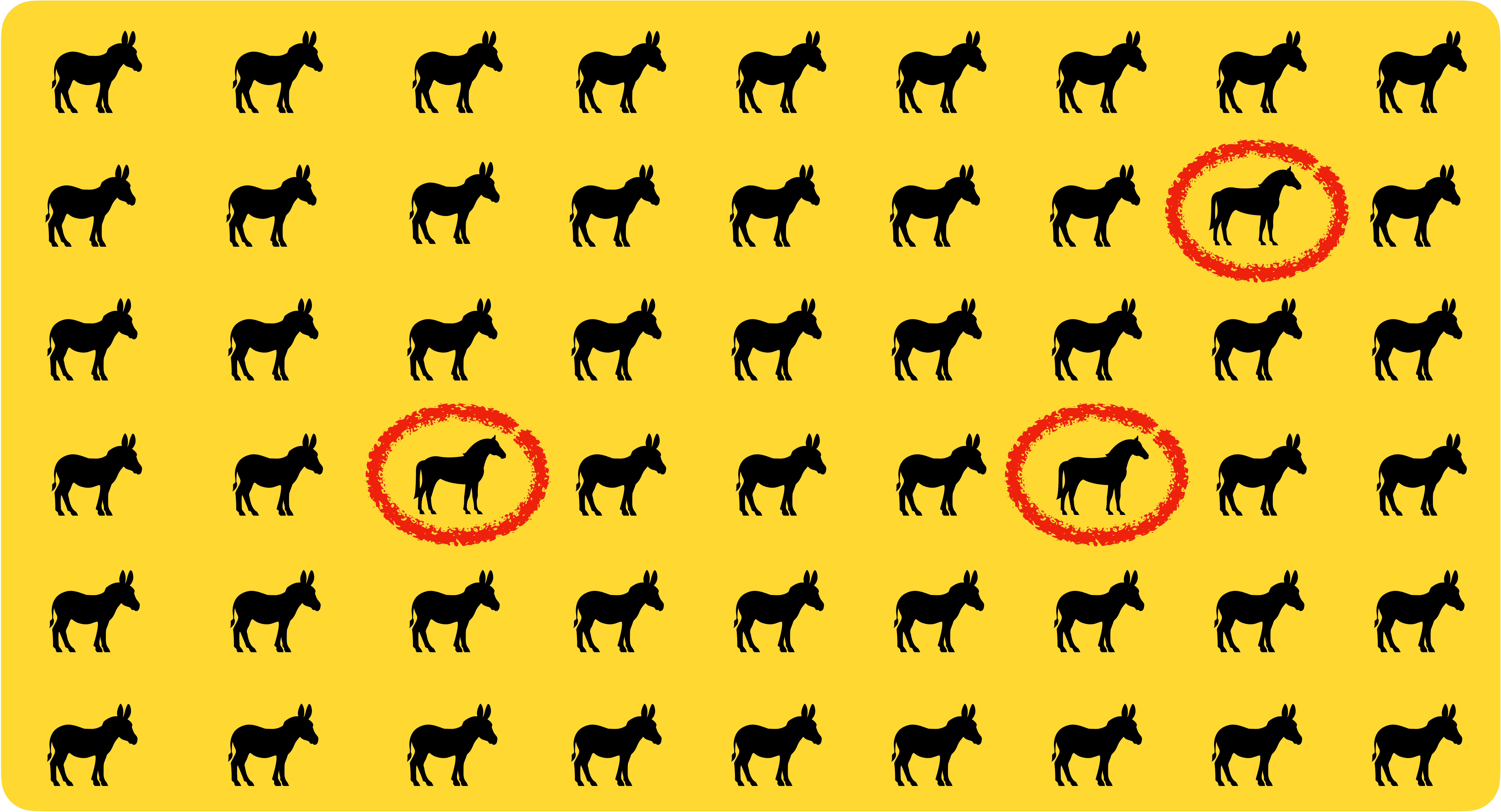


A TYPICAL BIG DATA TASK

image analysis: e.g., detect the number of horses

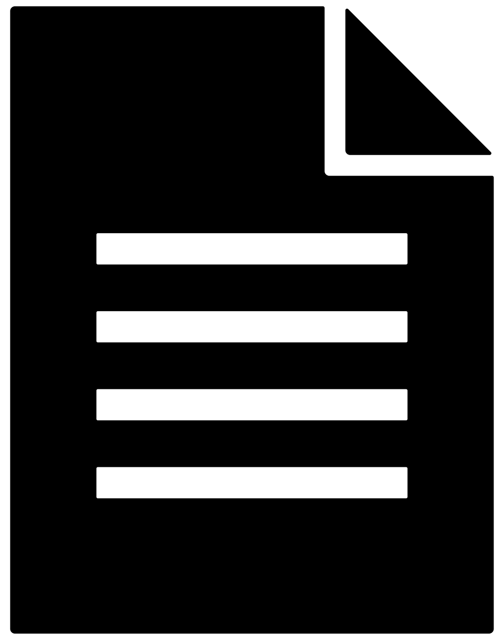
DASlab
@ Harvard SEAS



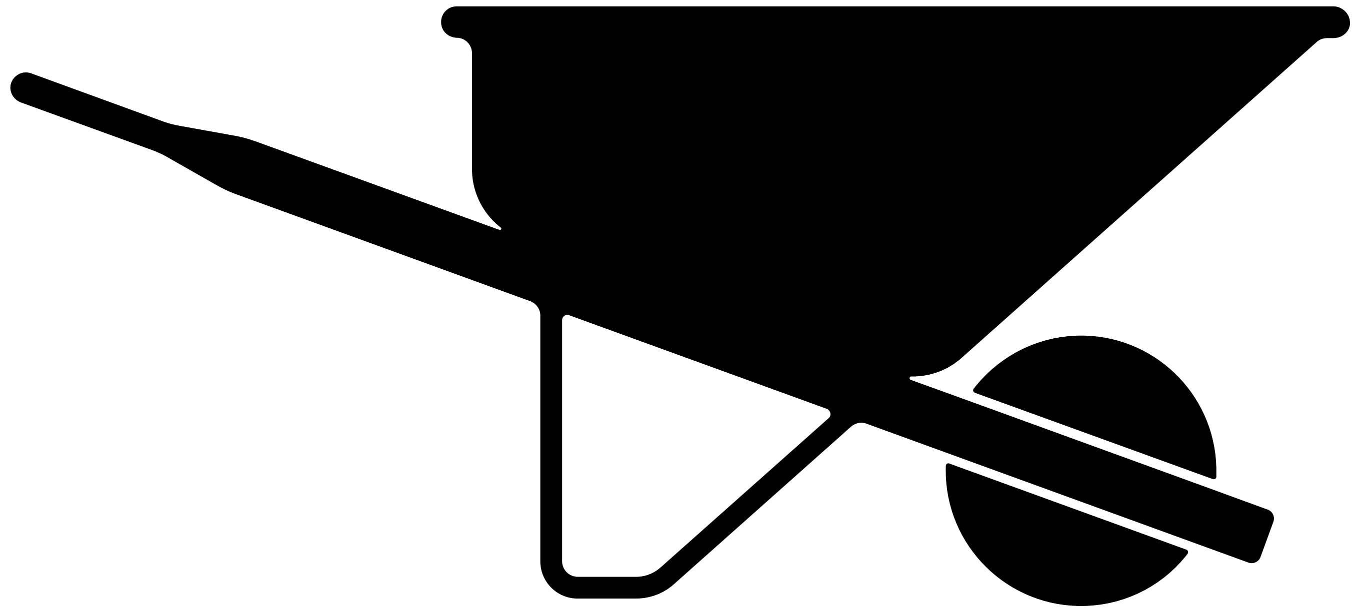



The core problem:
The size and organization of the data

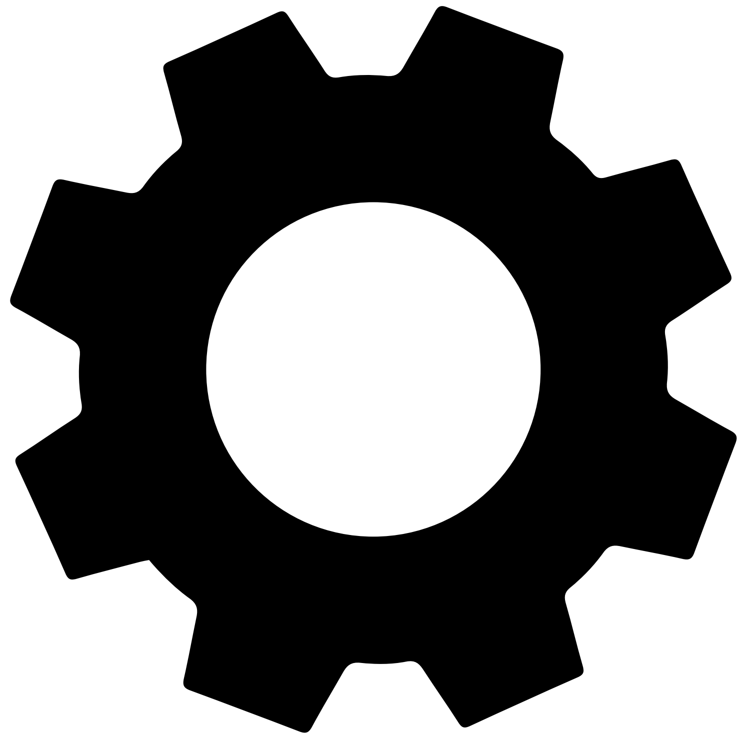
Three steps in big data/AI regardless of application



STORE

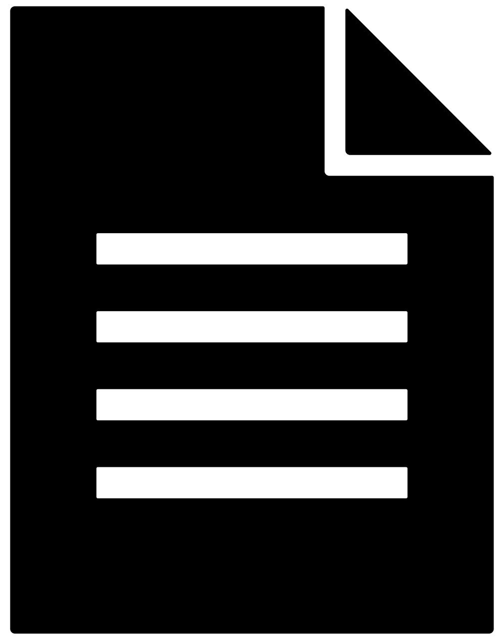


MOVE

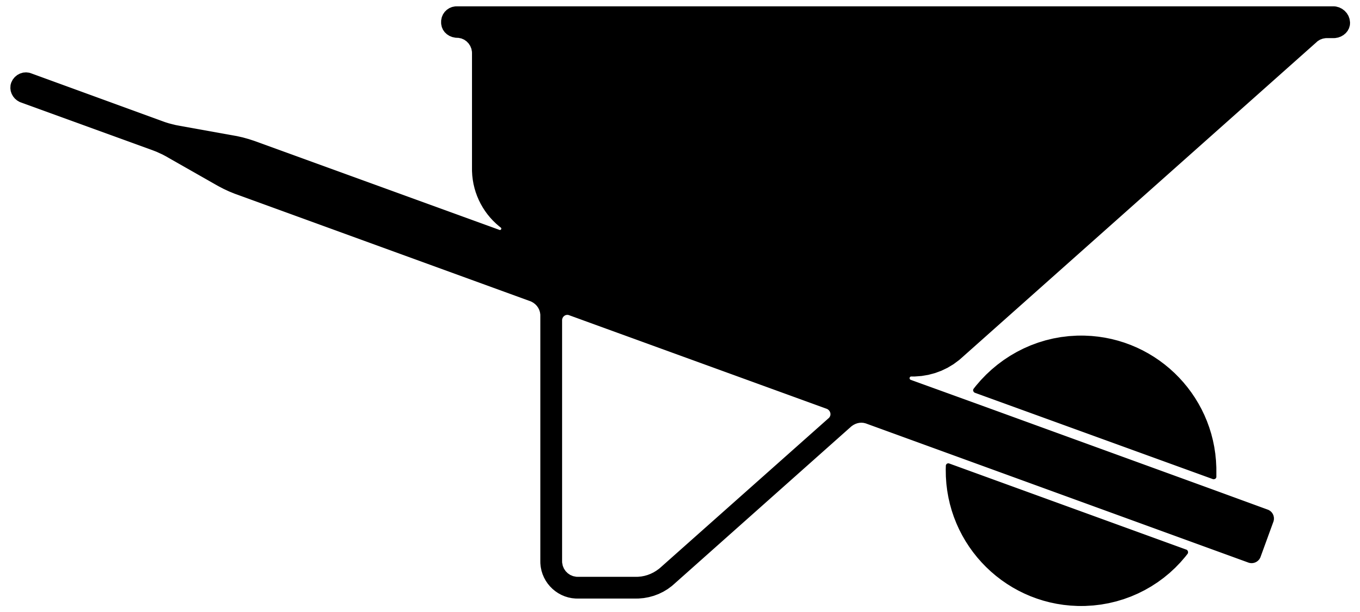


PROCESS

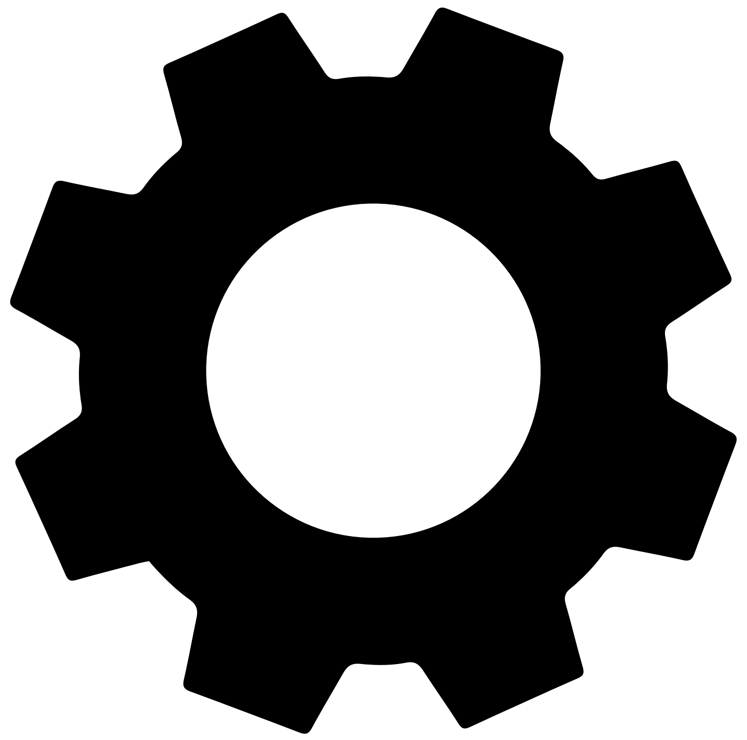
Three steps in big data/AI regardless of application



STORE



MOVE



PROCESS

How fast we can move and process data depends on the storage design decisions

What is a data system?

A data system is an end-to-end software system that:
manages storage, data movement, and provides access to data

What is a data system?

A data system is an end-to-end software system that:
manages storage, data movement, and provides access to data

1. For decades: data systems = SQL DBs
but with big data, the need for fast data systems is drastically broader than SQL

DASlab @ Harvard SEAS

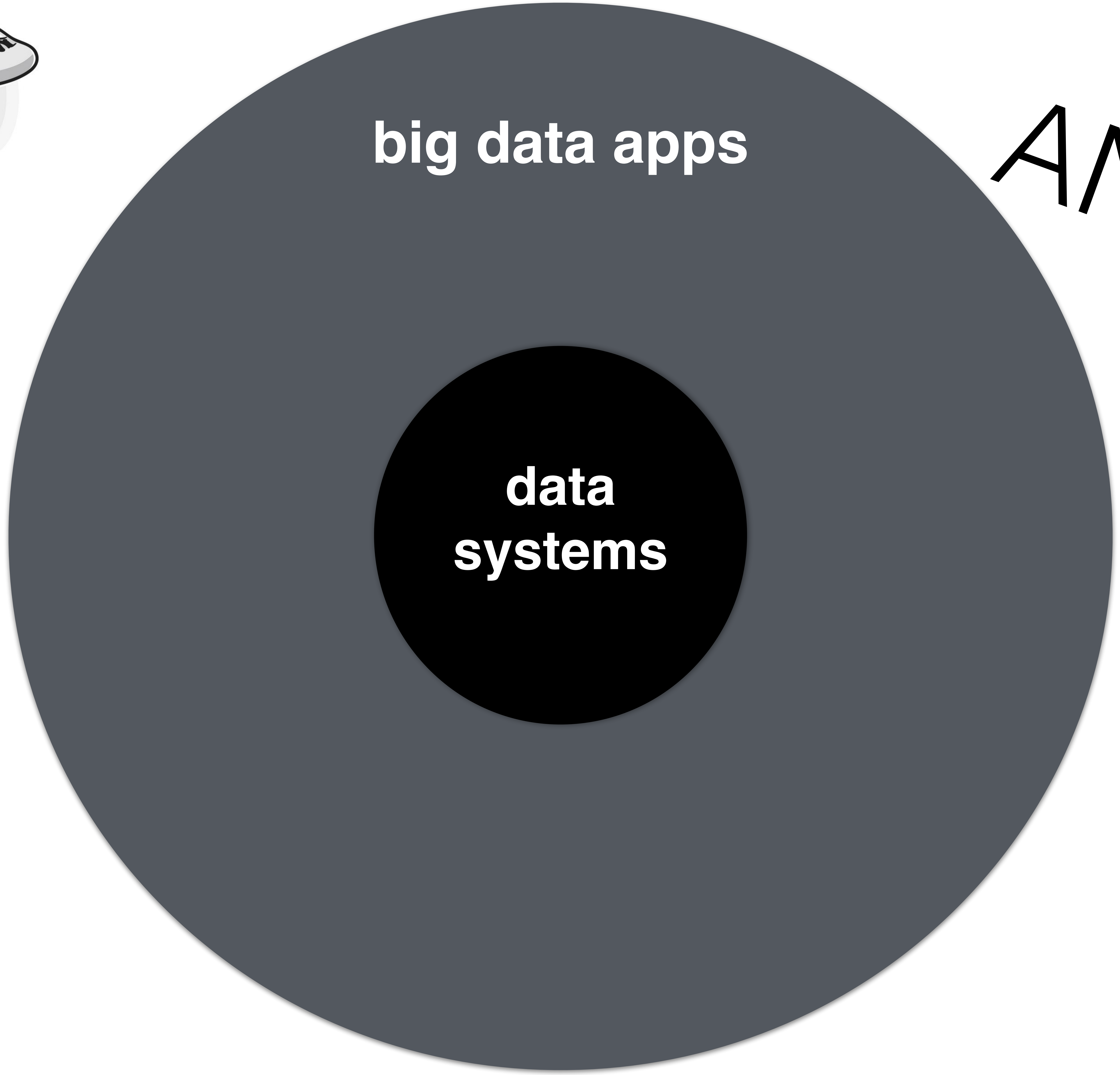
broader than SQL

store data X

get data with property Z

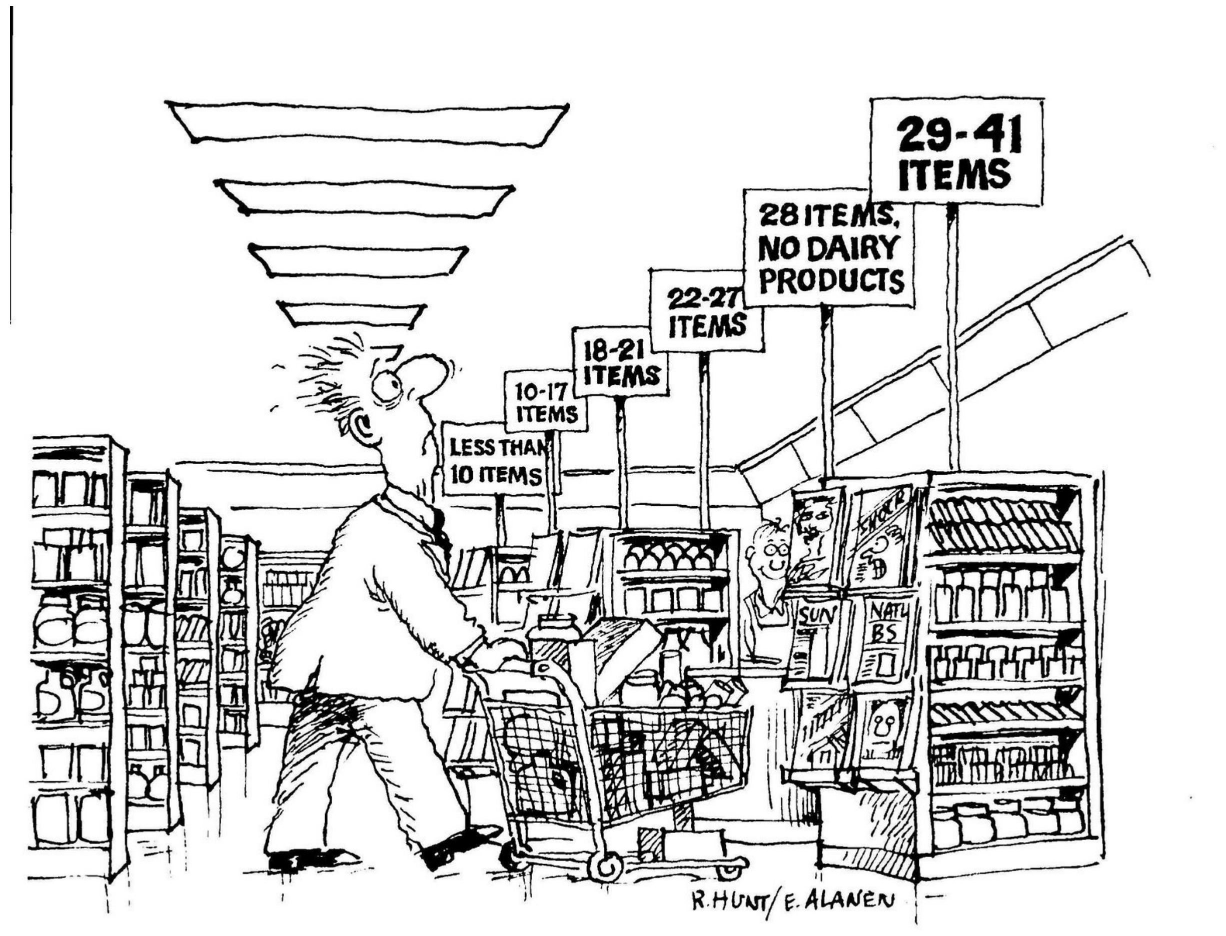
data systems

broader than SQL



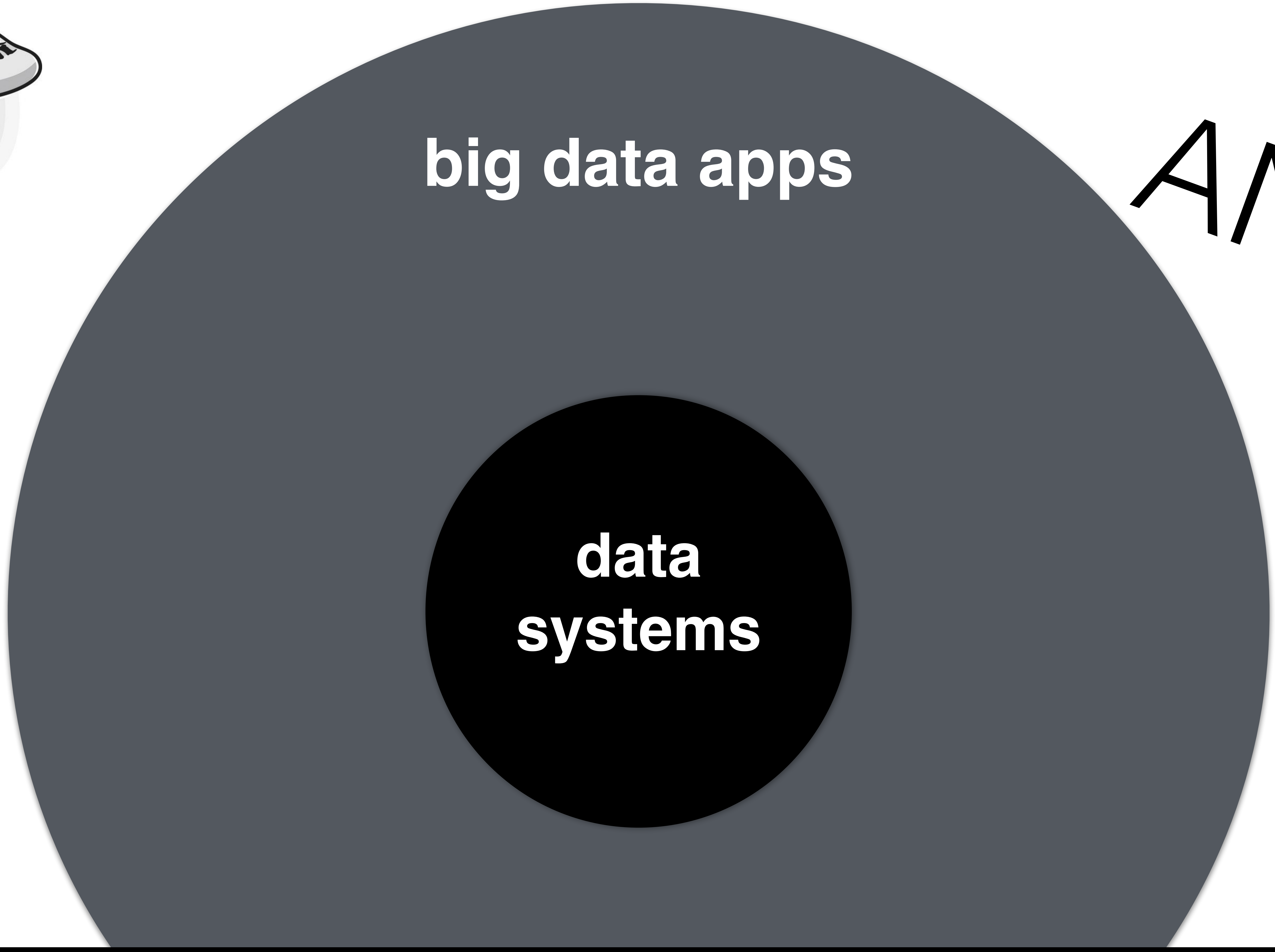
ANALYTICS

AI



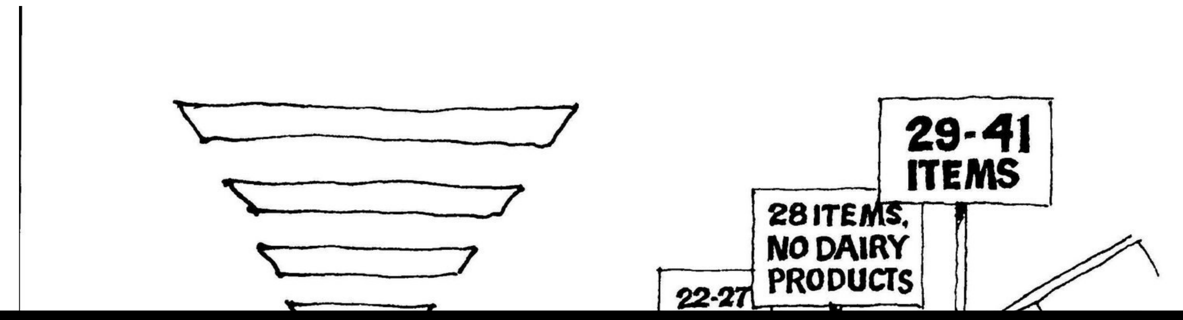
DASlab @ Harvard SEAS

broader than SQL



ANALYTICS

AI



New data systems to handle new requirements

TRANSACTIONS

Deposit money to my bank account

Transfer money from … to…

TRANSACTIONS

Deposit money to my bank account

Transfer money from … to…

ANALYTICS

How much do customers
of X spent on average every month?

broader than SQL

TRANSACTIONS

Deposit money to my bank account

Transfer money from … to…

ANALYTICS

How much do customers
of X spent on average every month?

AI

Is this transaction legal?

Should we give a loan to customer X?

DASlab
@ Harvard SEAS

SOCIAL NETWORKS: REVIEWS/POSTS

How many costumers on average
leave a 4 star review or better?

SOCIAL NETWORKS: REVIEWS/POSTS

How many costumers on average
leave a 4 star review or better?

AI

Is this new review a legitimate one?

SOCIAL NETWORKS: REVIEWS/POSTS

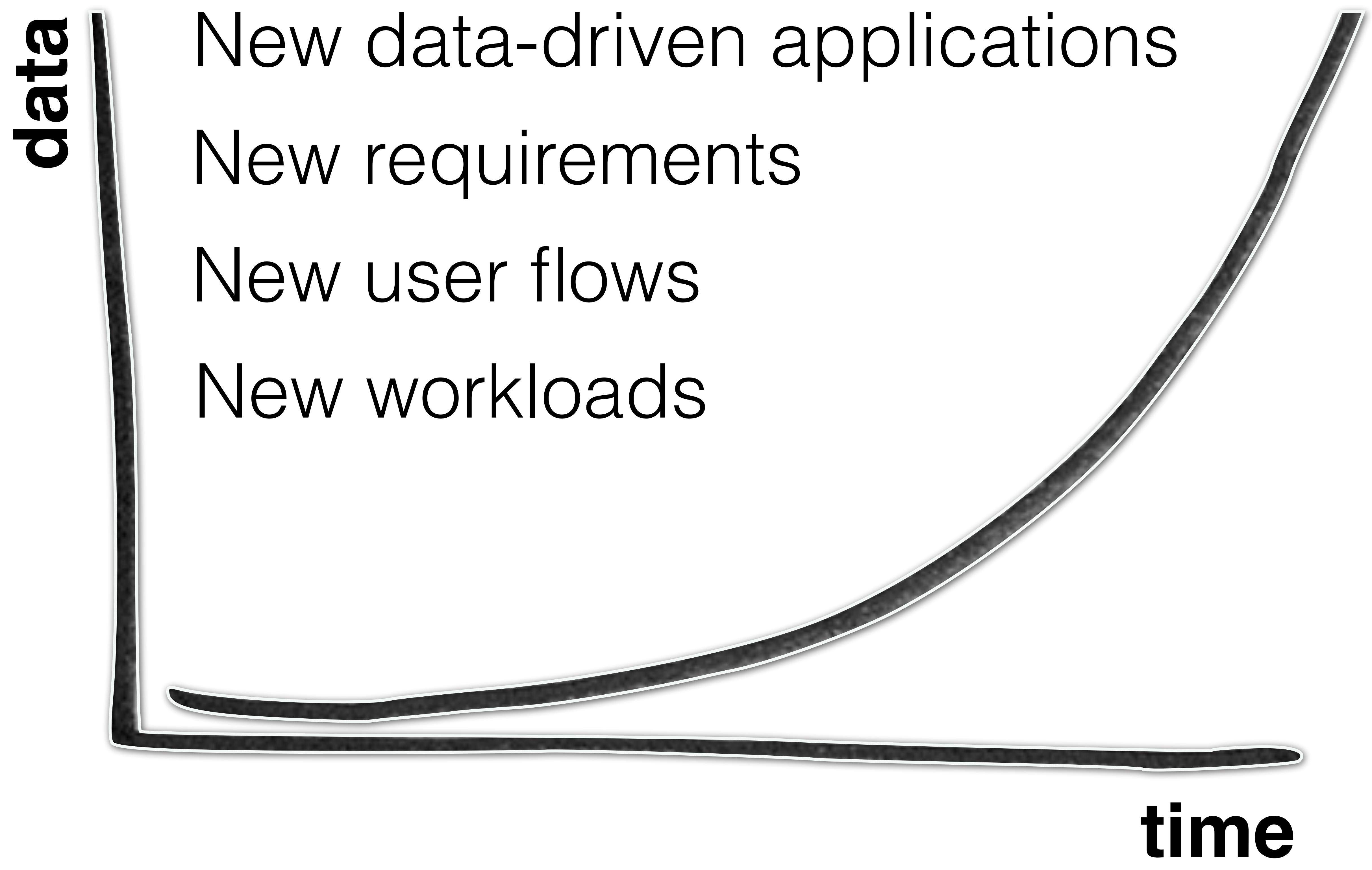
How many costumers on average
leave a 4 star review or better?

AI

Is this new review a legitimate one?

COMMUTING

Compute price for next Uber ride



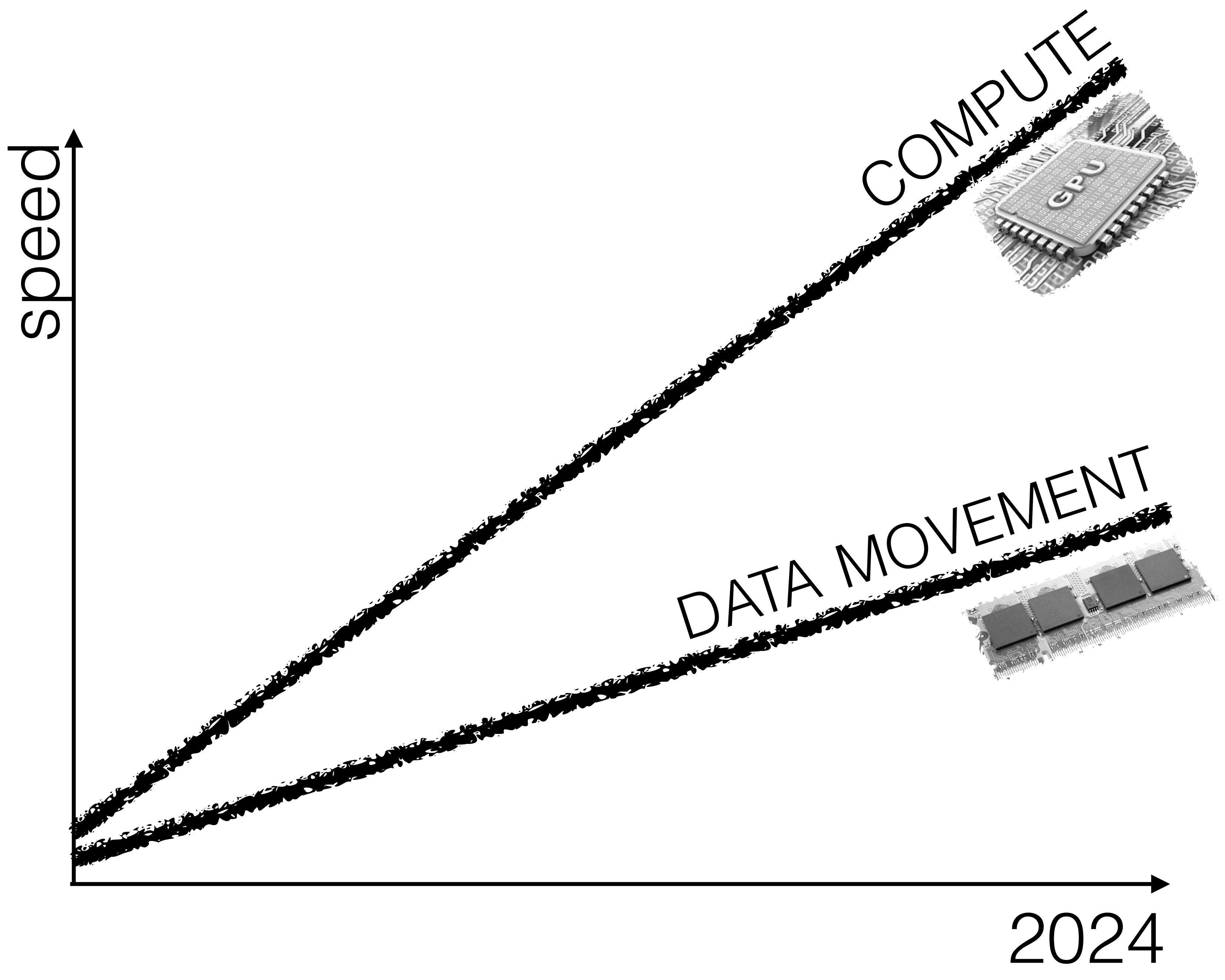
The need for data systems grows with data

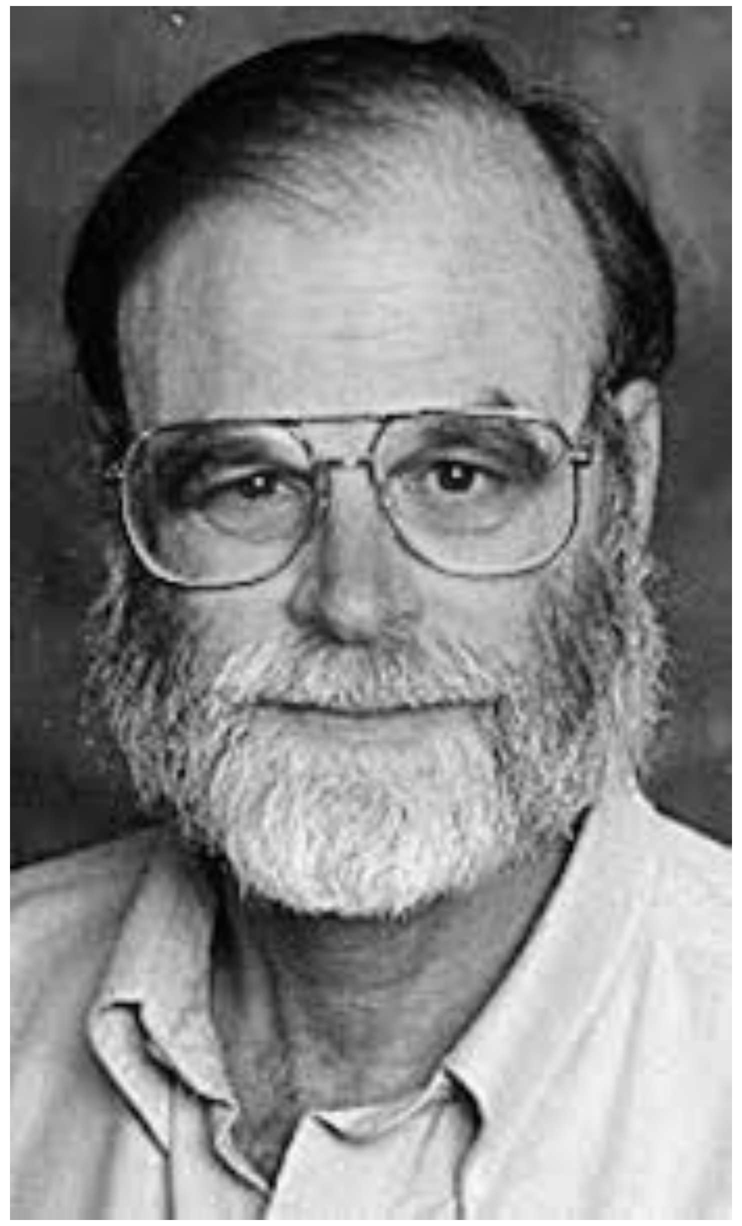
2. As data grows, having the right data system
for each application is increasingly more critical

2. As data grows, having the right data system
for each application is increasingly more critical

system architecture
it starts with storage

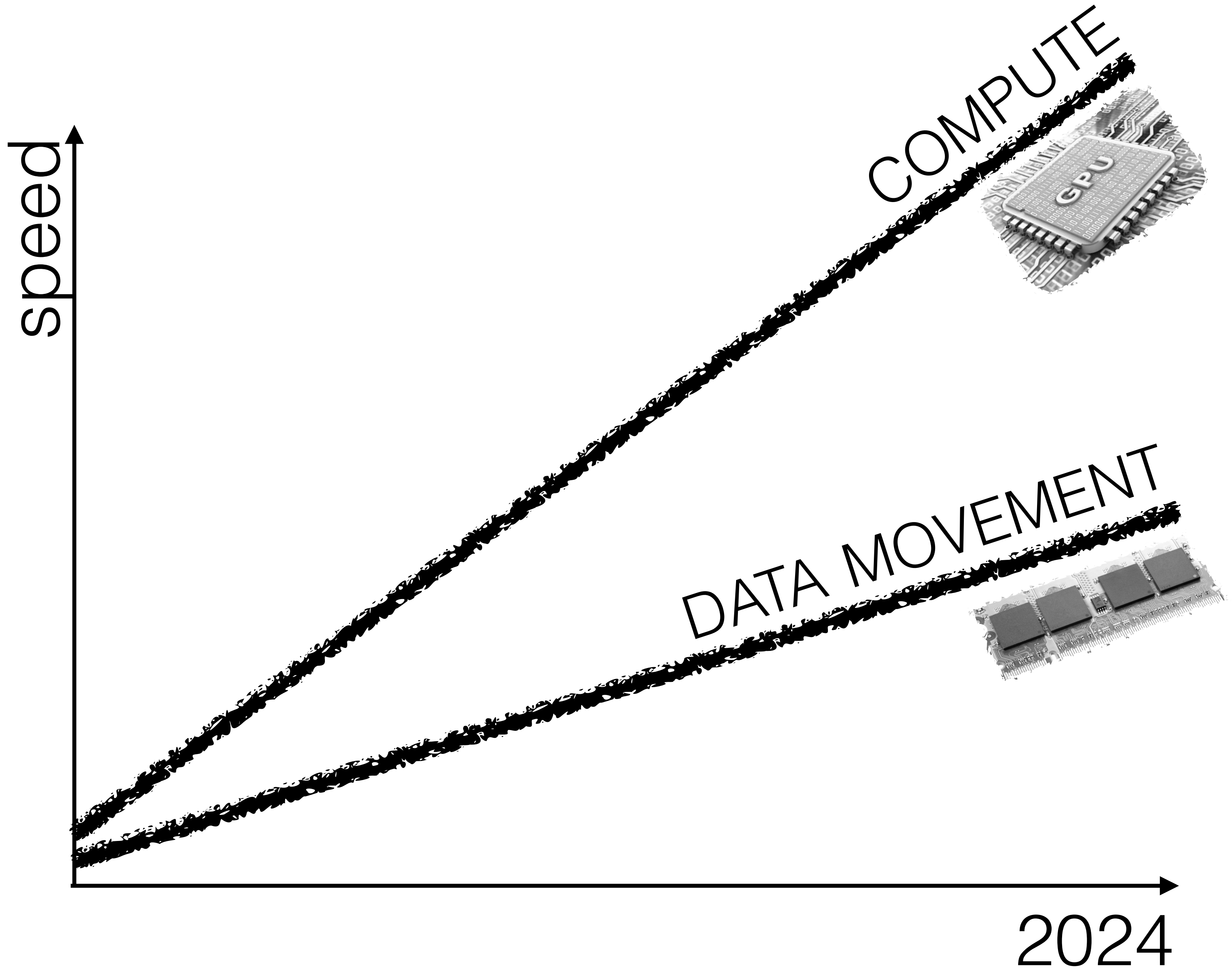
the right data system

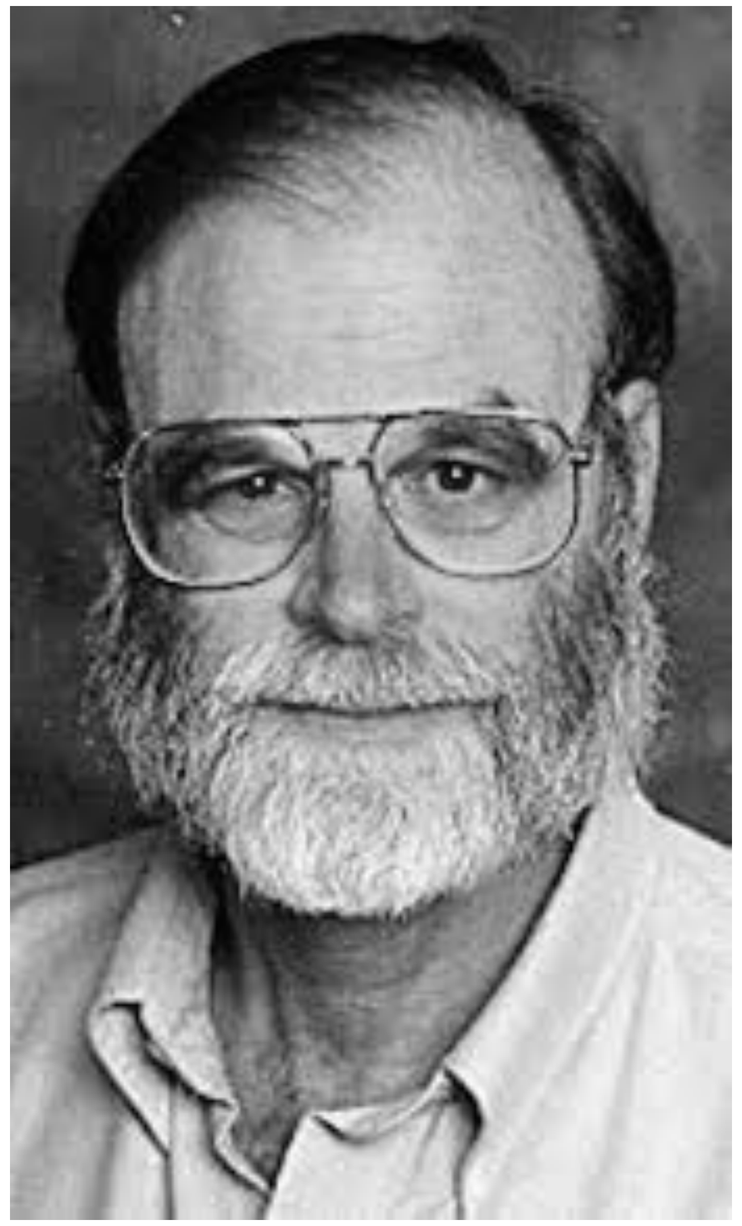




register = this room
caches = this city
memory = nearby city
disk = Pluto

Jim Gray, Turing Award 1998

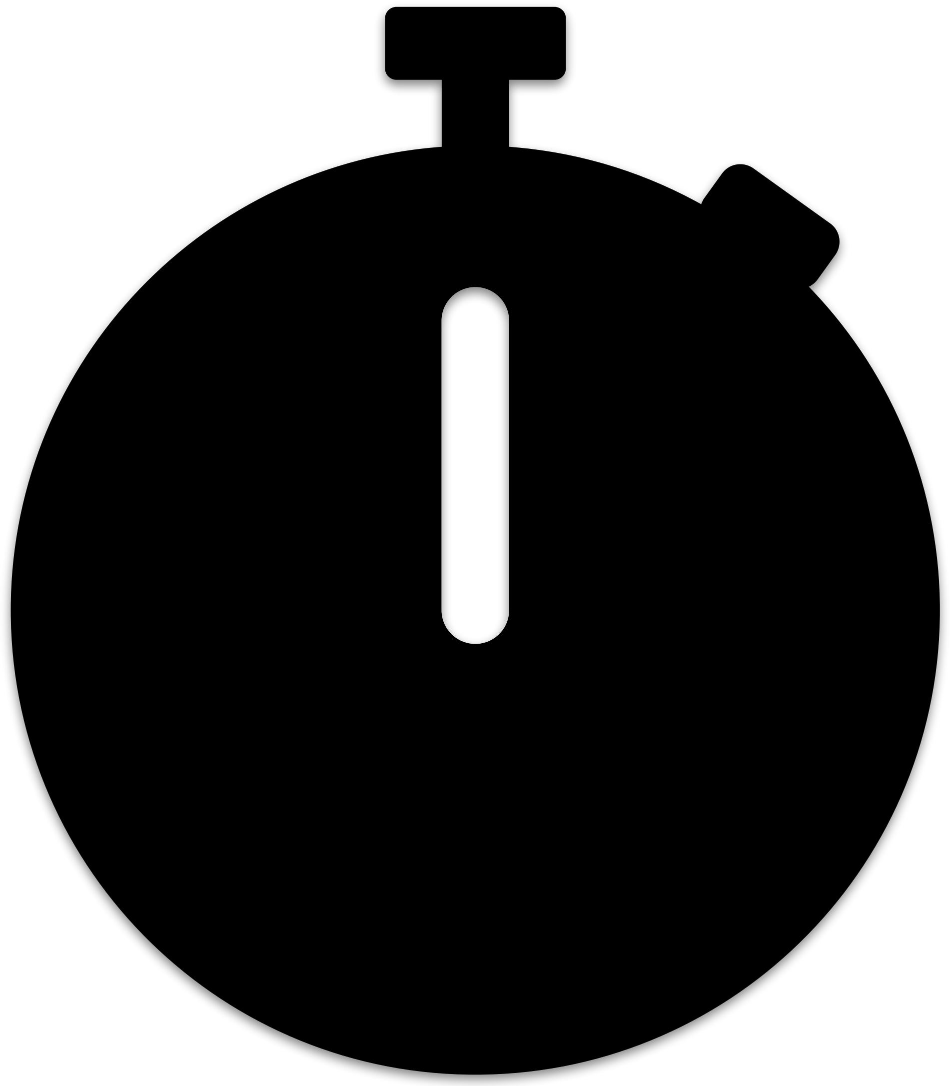




register = this room
caches = this city
memory = nearby city
disk = Pluto

Jim Gray, Turing Award 1998

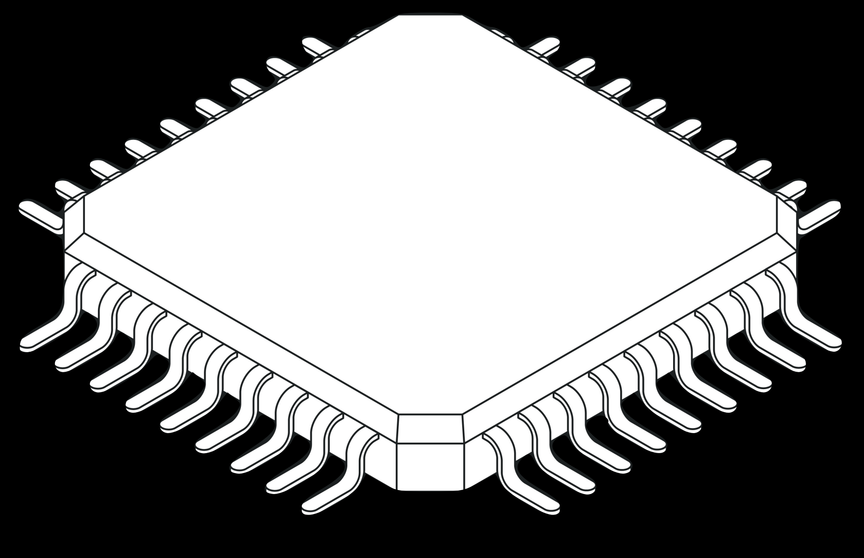
Data movement dominates everything

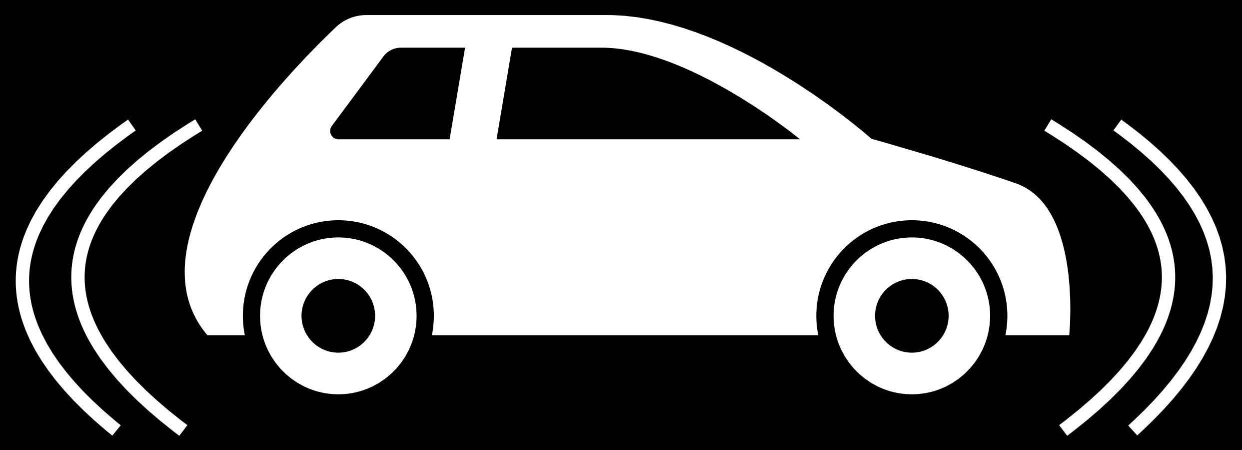


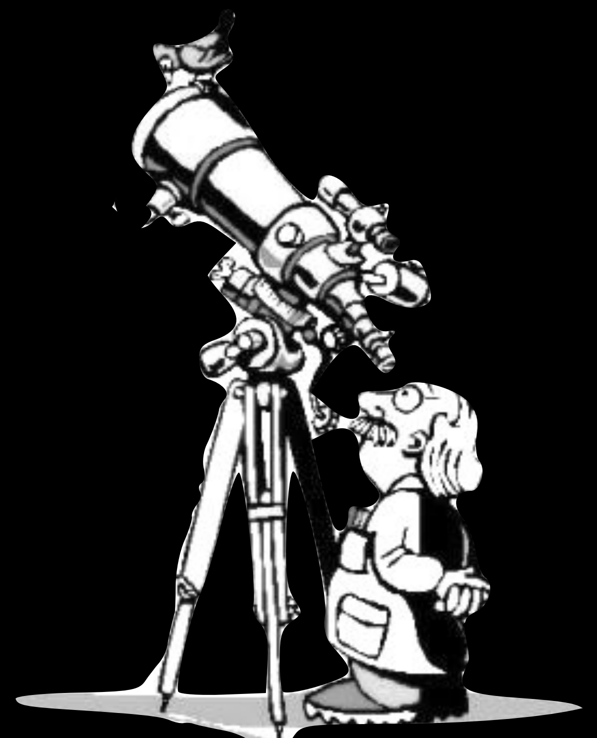
70-80% of processing costs go into data movement

computational hardware utilization: only 30-50%

The problem: as the big data/AI world keeps changing…

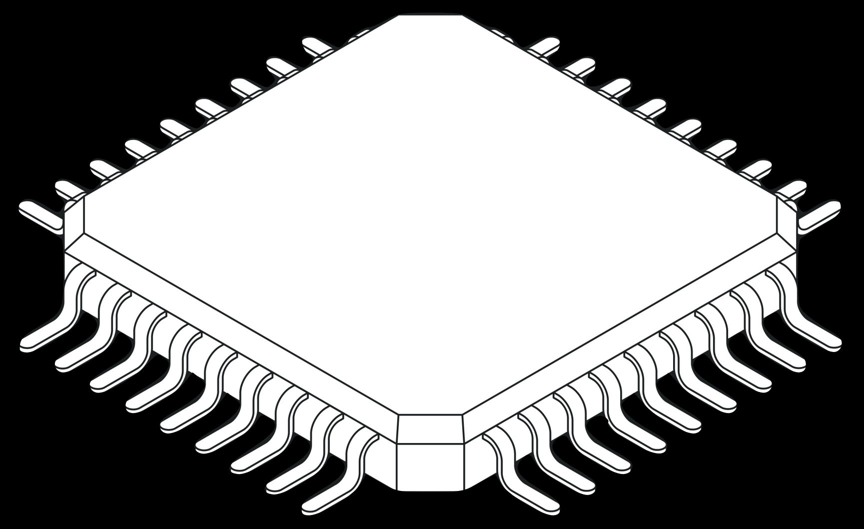


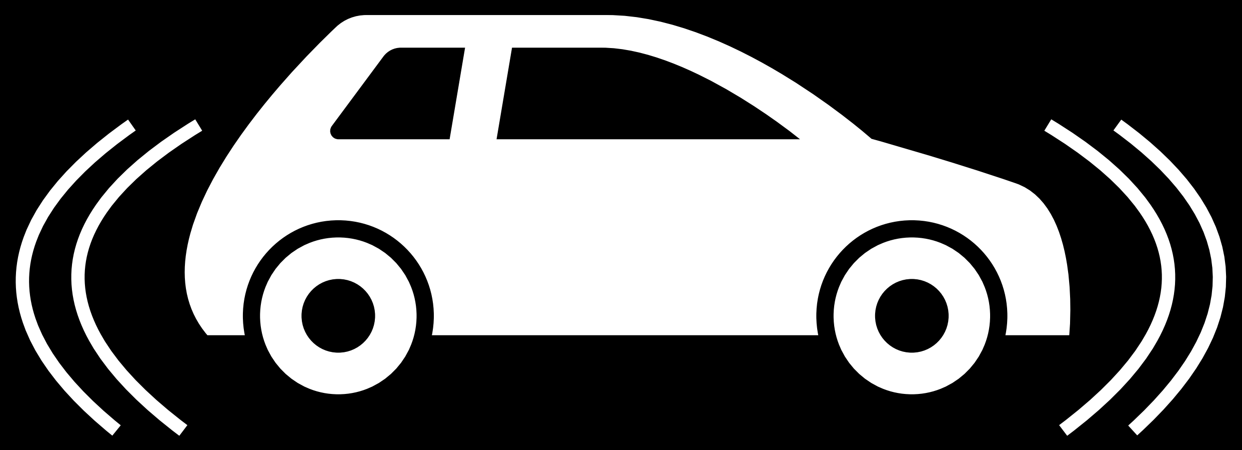


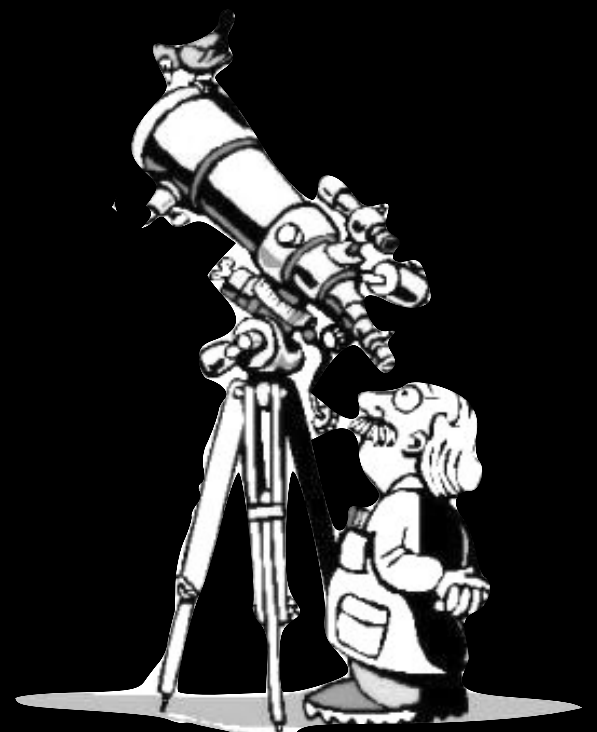


The problem: as the big data/AI world keeps changing…

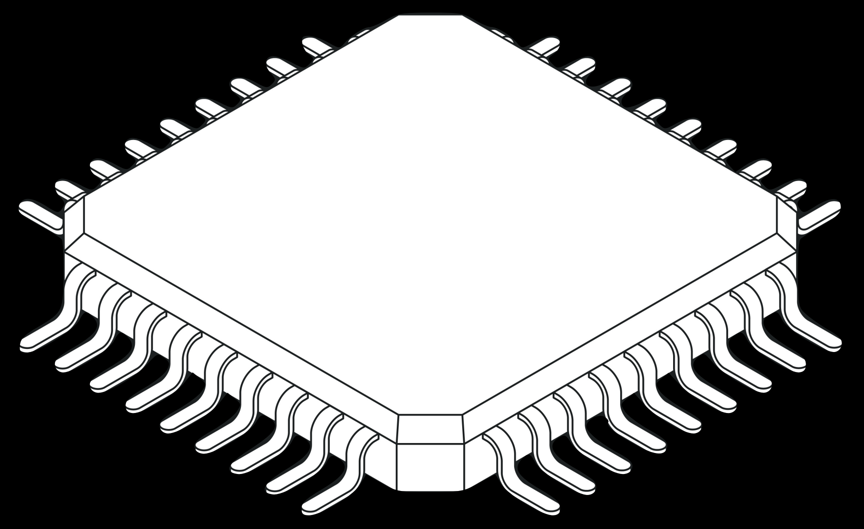
there is a continuous need for new data systems
but it is **extremely hard to design & build new systems**

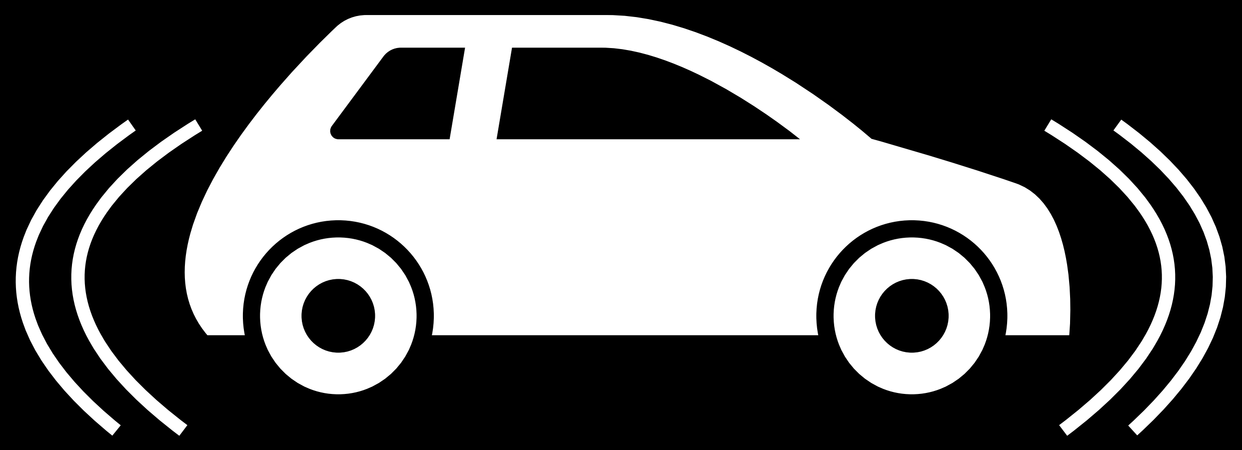


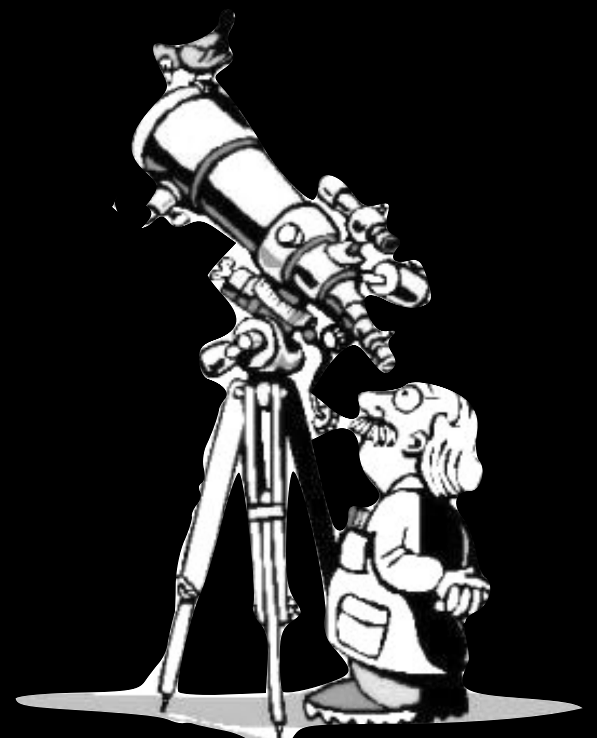




How do we design a data system that is **X times faster for a workload W**?

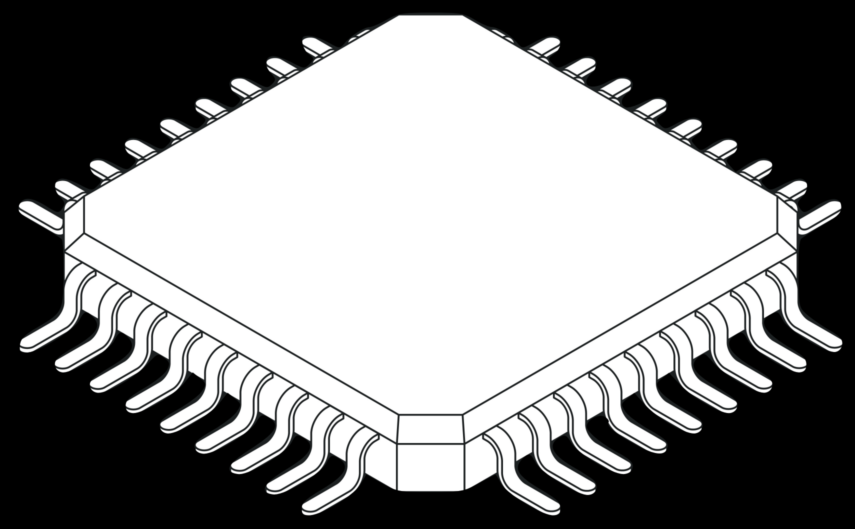


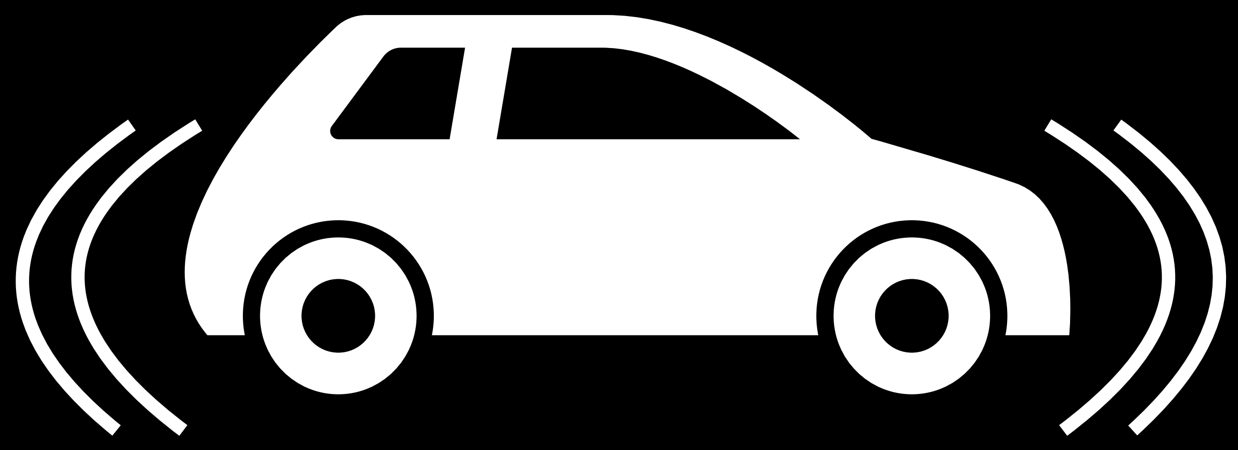


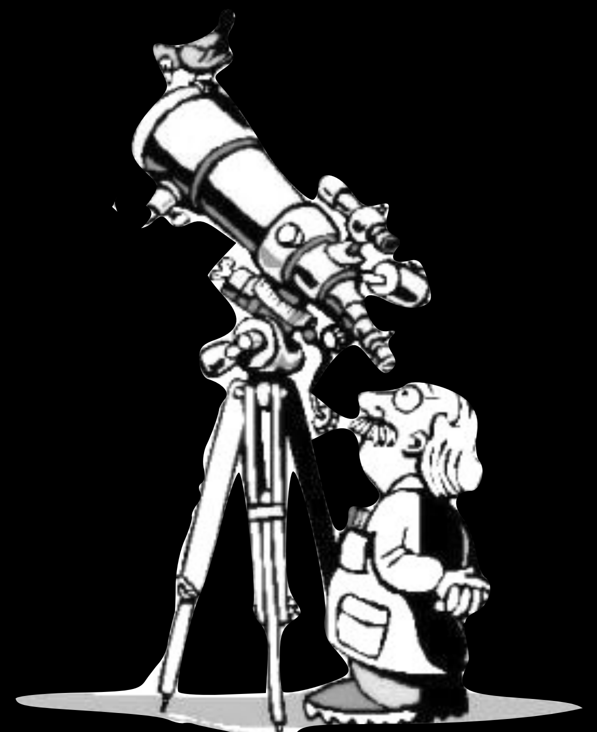


How do we design a data system that is **X times faster for a workload W**? 

 How do we design a data system that allows for control of **cloud cost**?





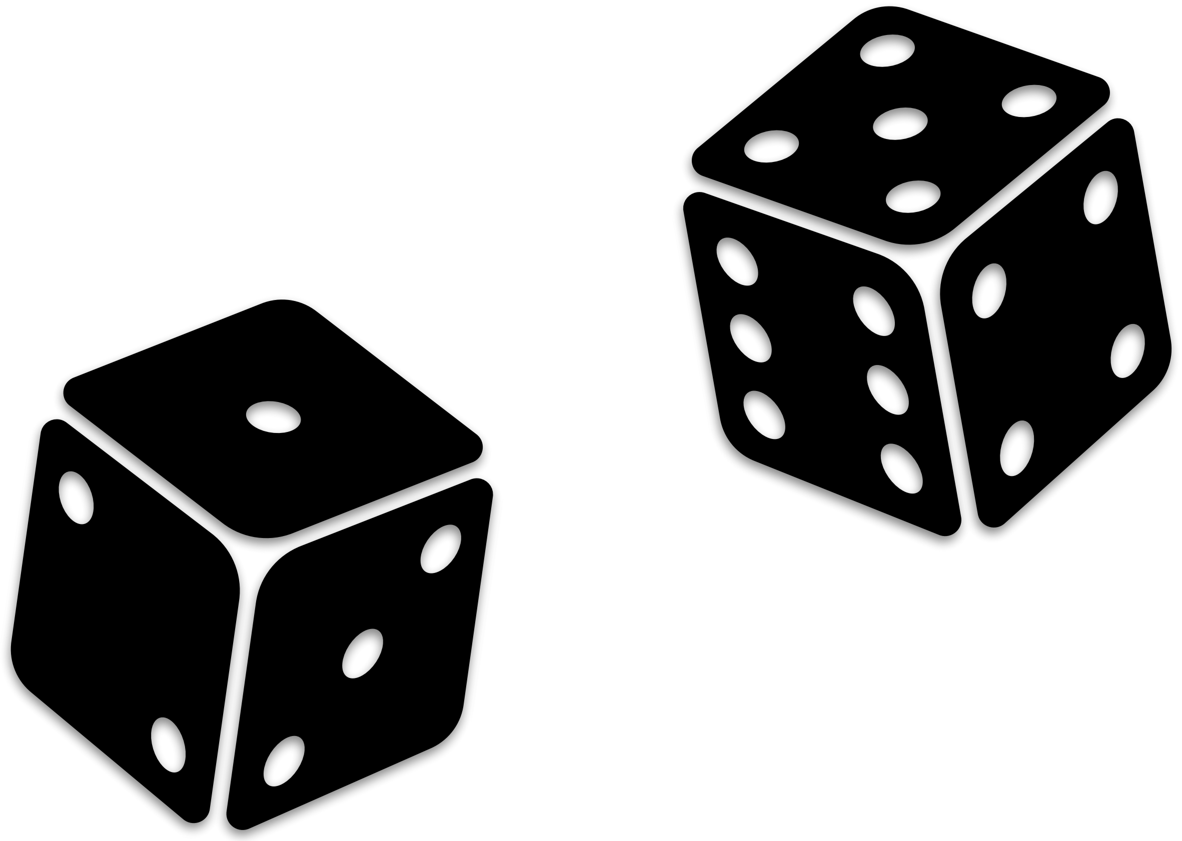


How do we design a data system that is **X times faster for a workload W**? 

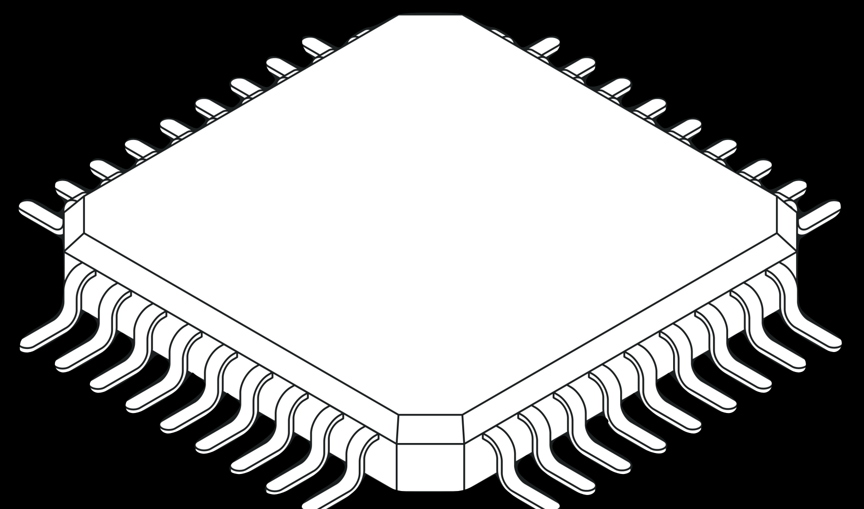
 How do we design a data system that allows for control of **cloud cost**?

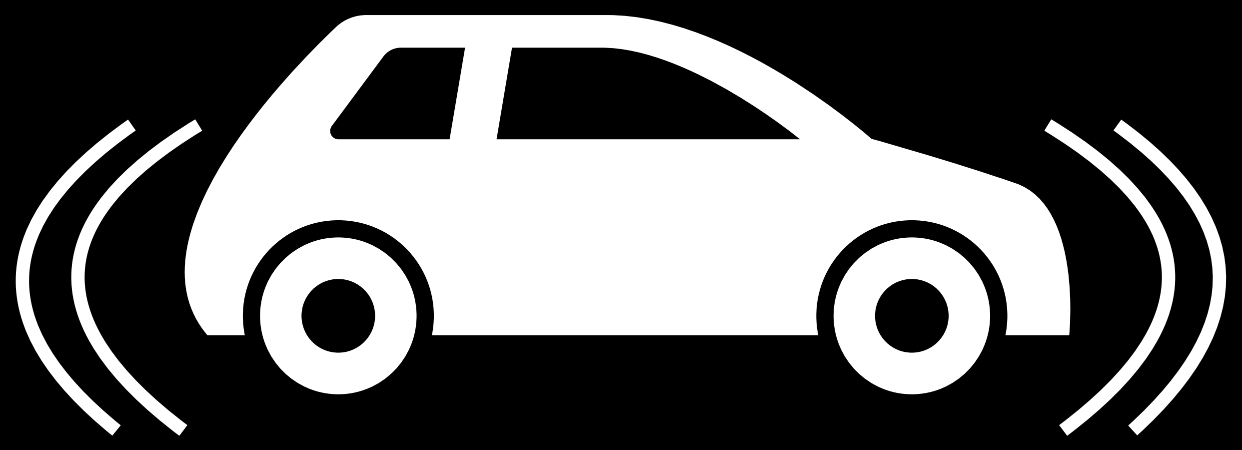
What happens if we introduce **new application feature** Y?

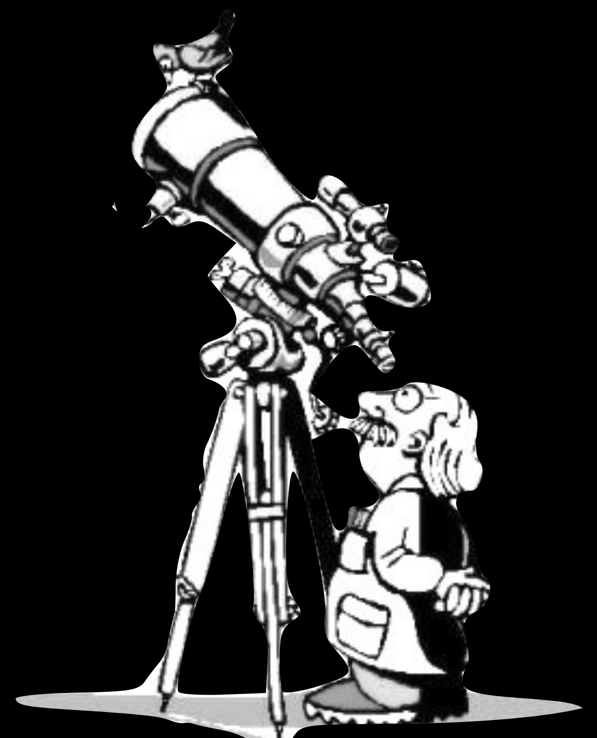
Should we **upgrade** to new version Z?



What will **break** our system?





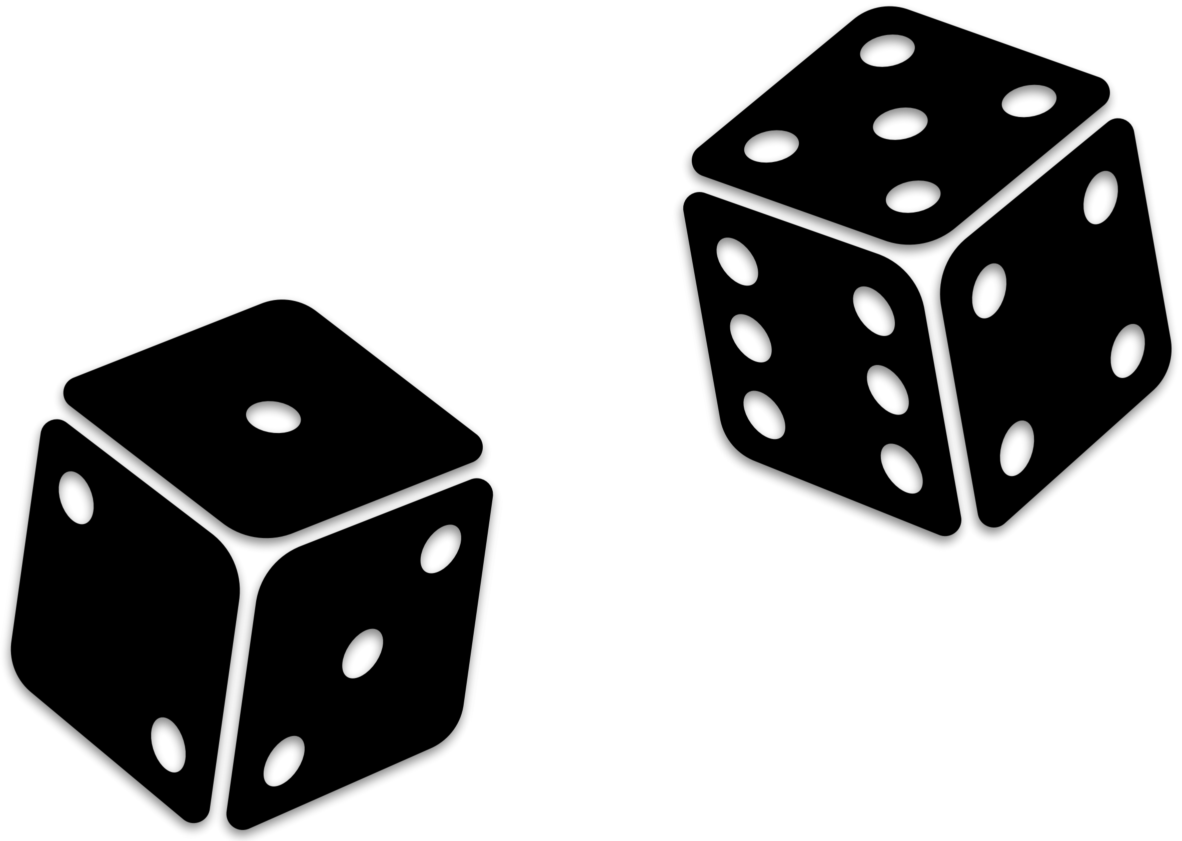


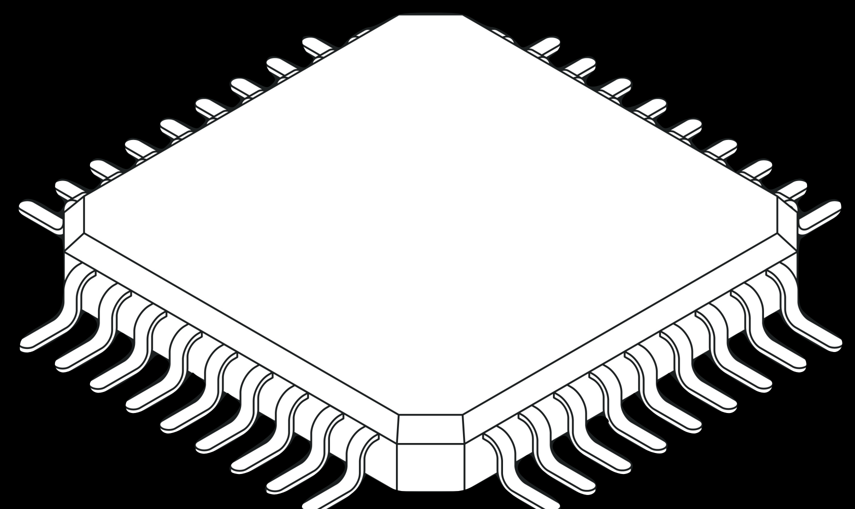
BOTTLENECK: SUB-OPTIMAL DATA SYSTEMS

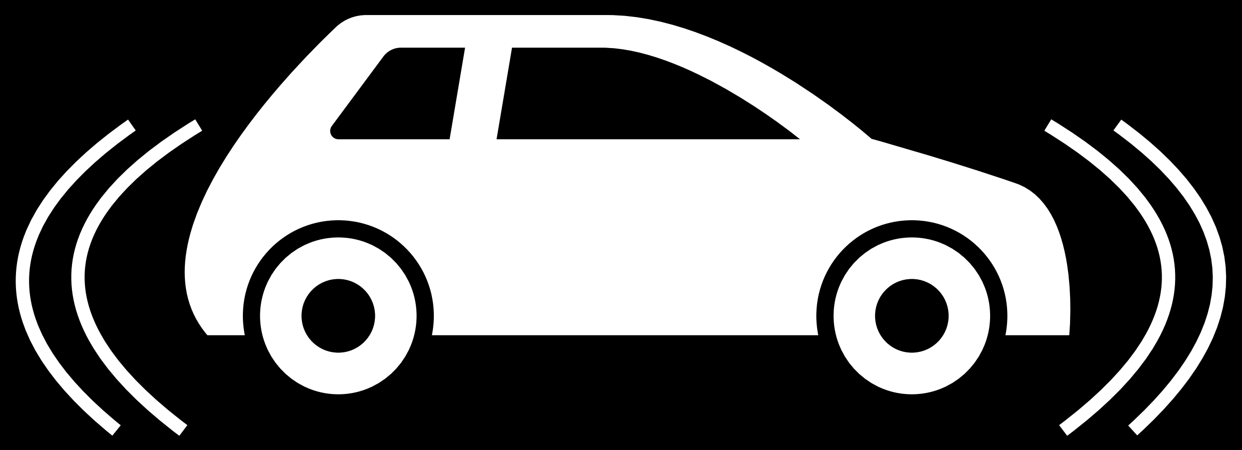
What happens if we introduce **new application feature** Y?

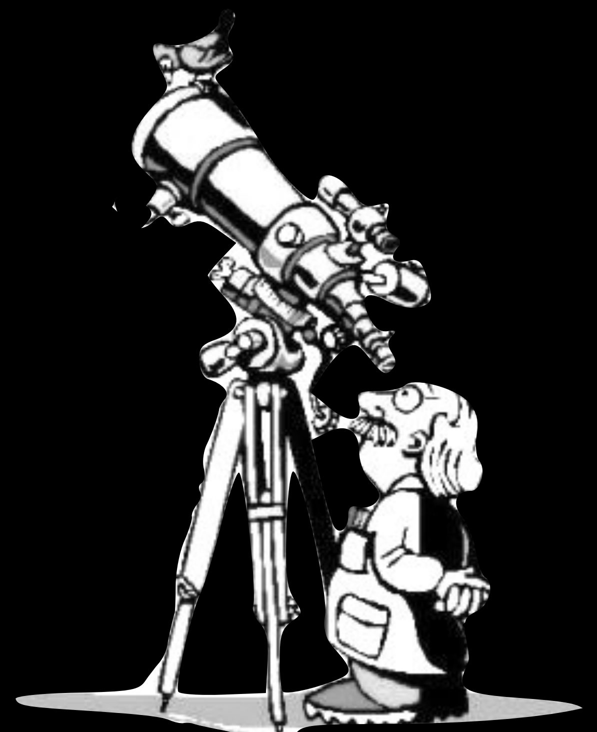
Should we **upgrade** to new version Z?

What will **break** our system?





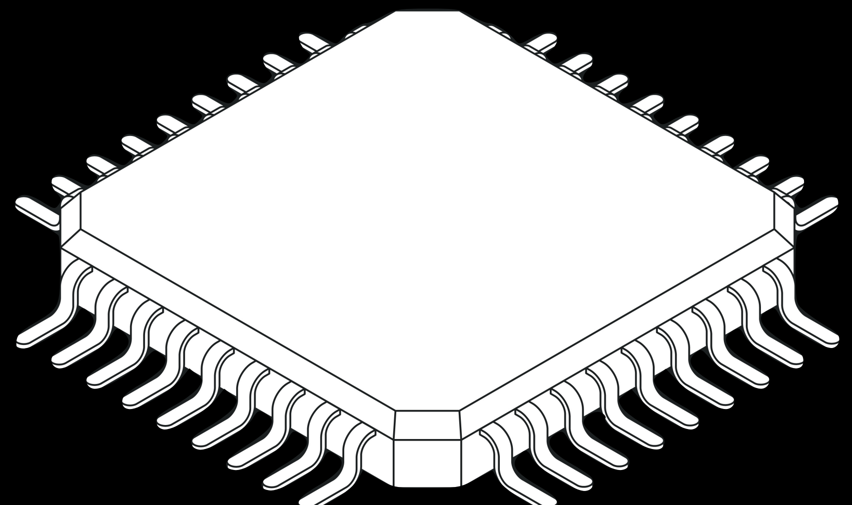


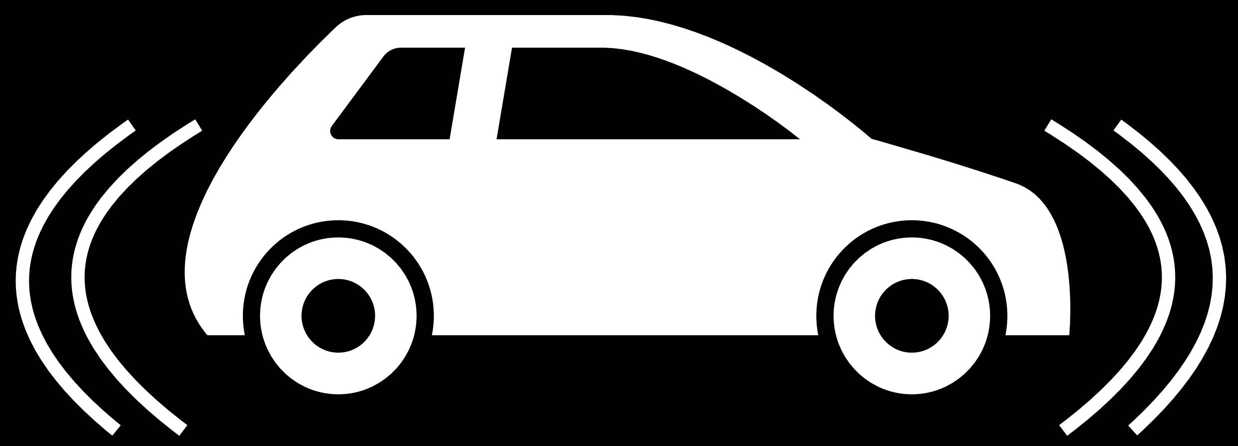


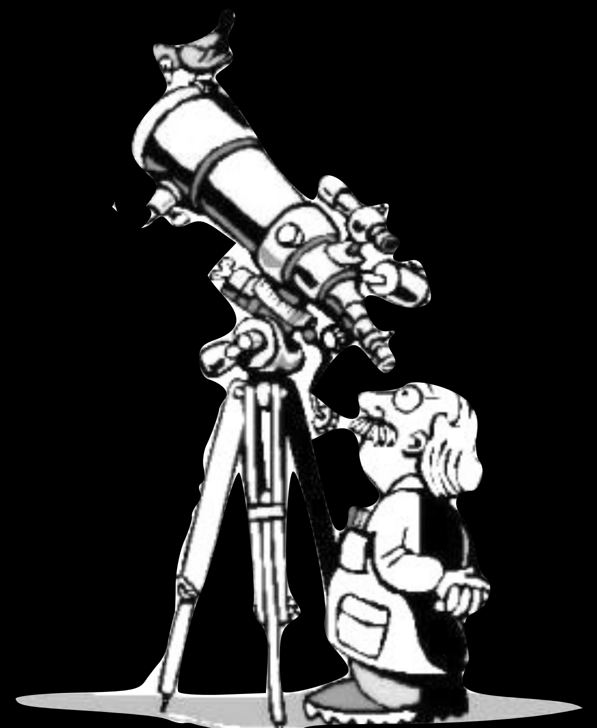
BOTTLENECK: SUB-OPTIMAL DATA SYSTEMS

huge cloud cost

environmental impact





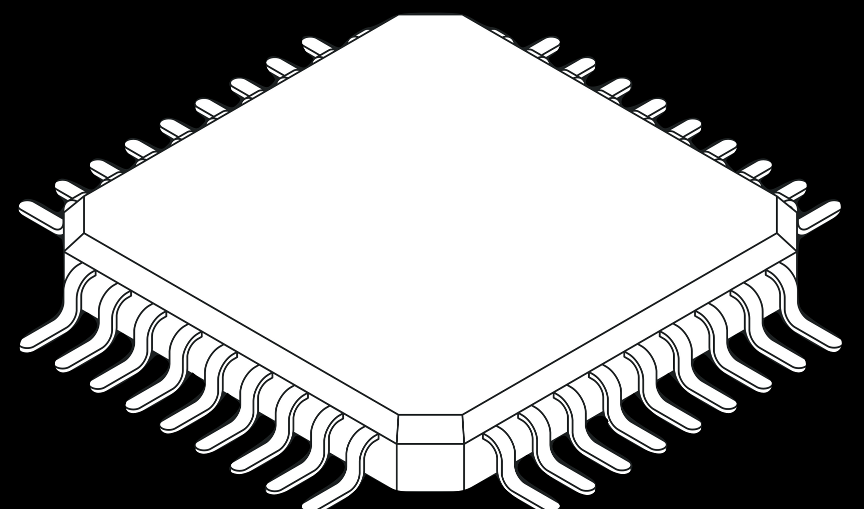


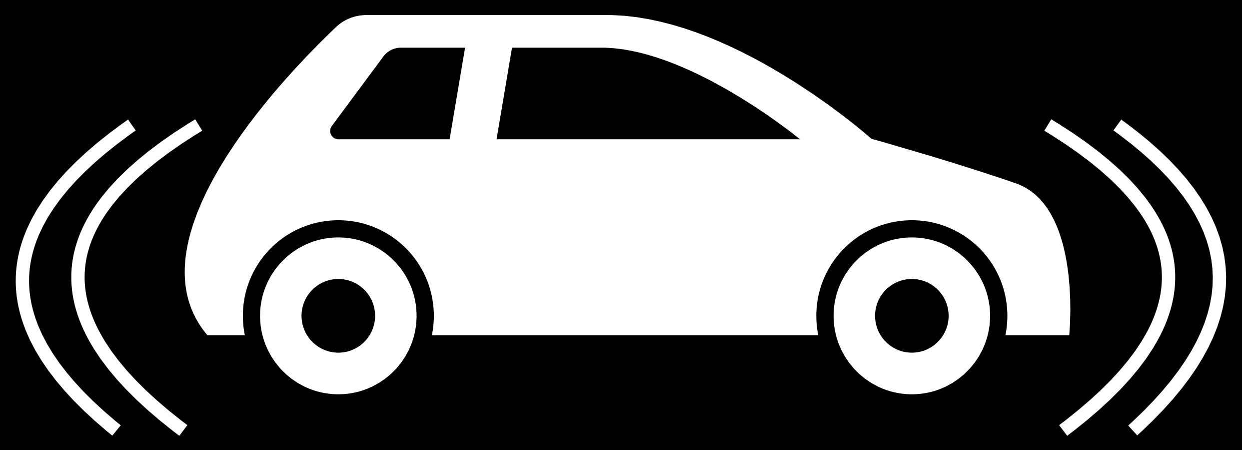
BOTTLENECK: SUB-OPTIMAL DATA SYSTEMS

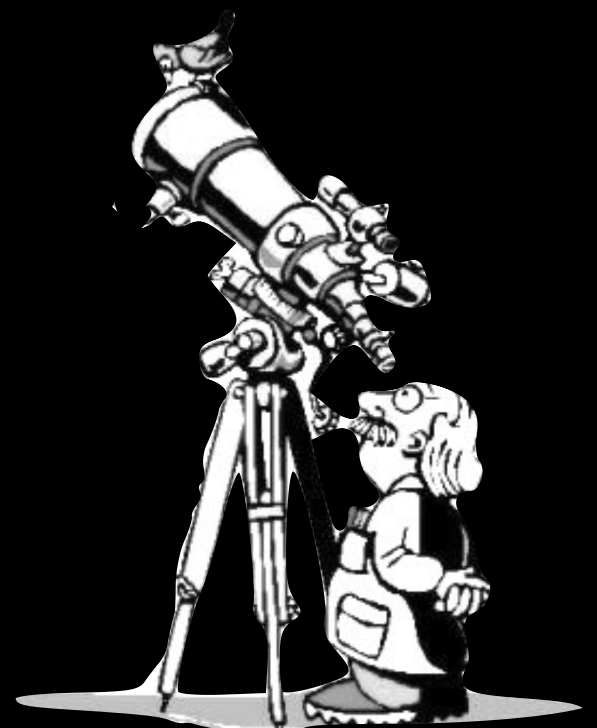
expensive transitions

huge cloud cost

environmental impact







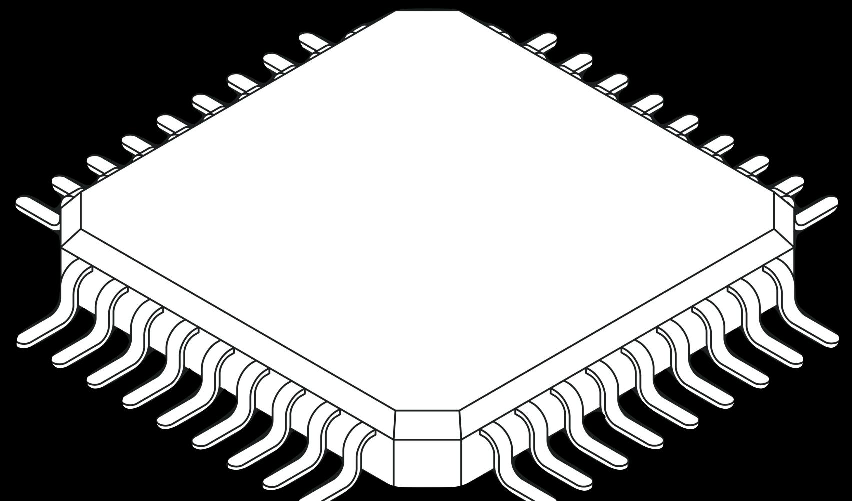
BOTTLENECK: SUB-OPTIMAL DATA SYSTEMS

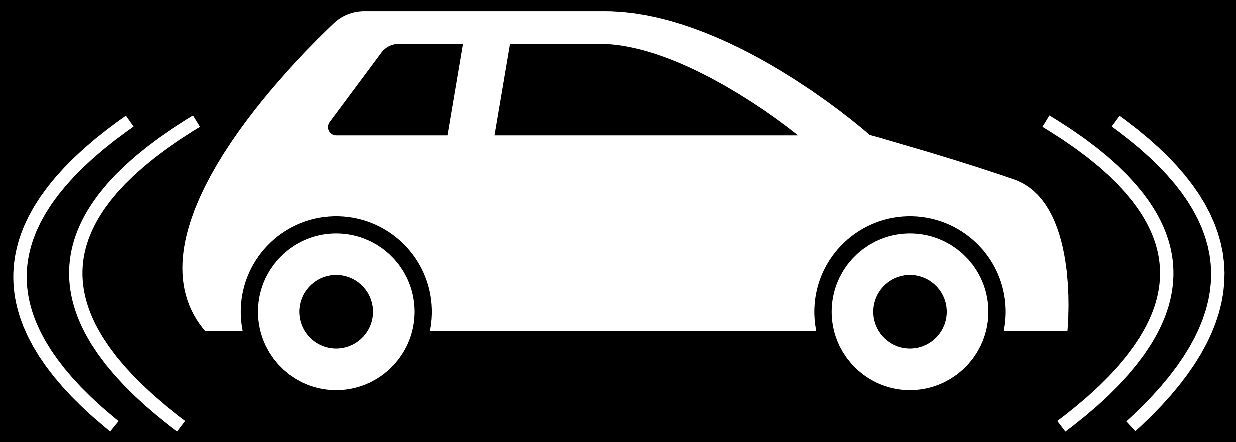
expensive transitions

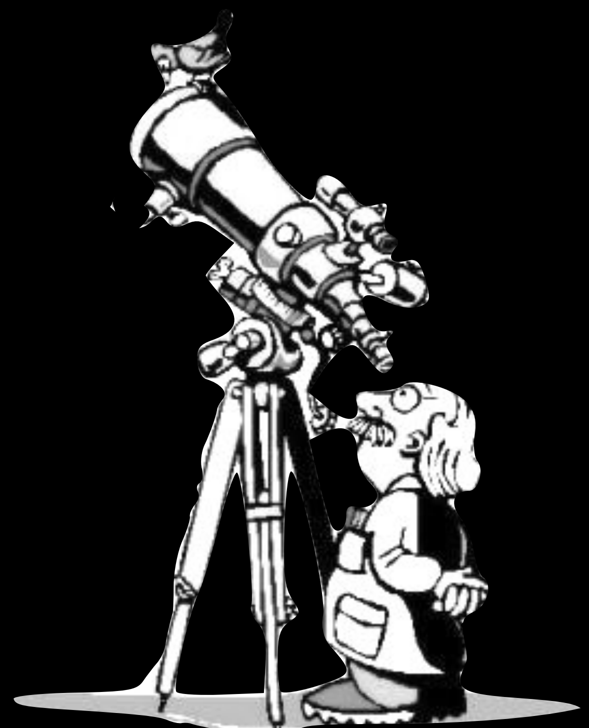
huge cloud cost

application feasibility

environmental impact





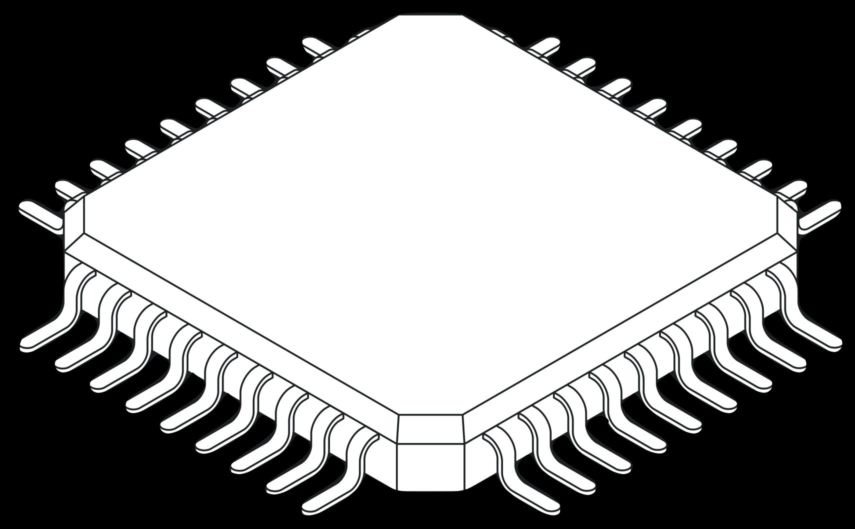


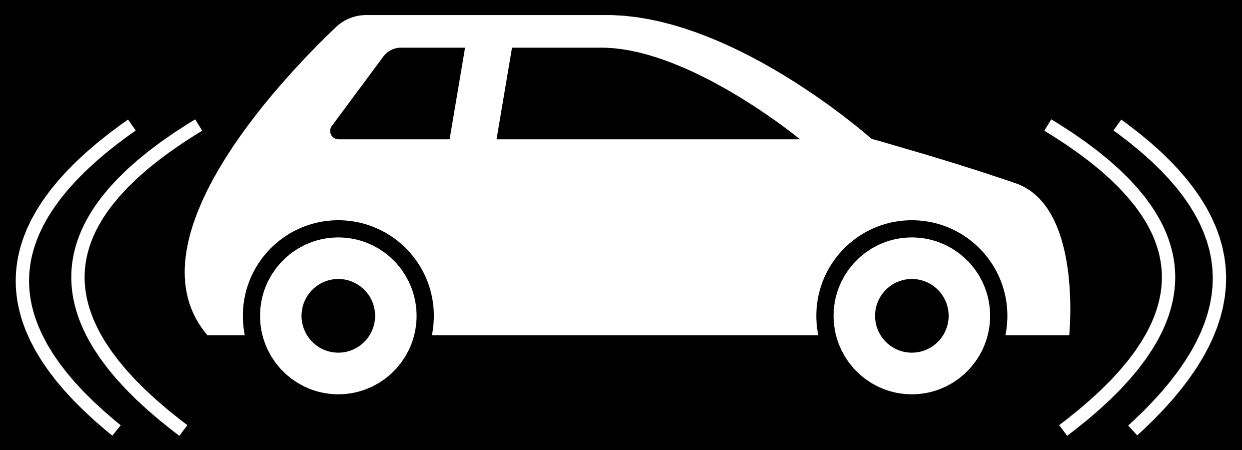
BOTTLENECK: SUB-OPTIMAL DATA SYSTEMS

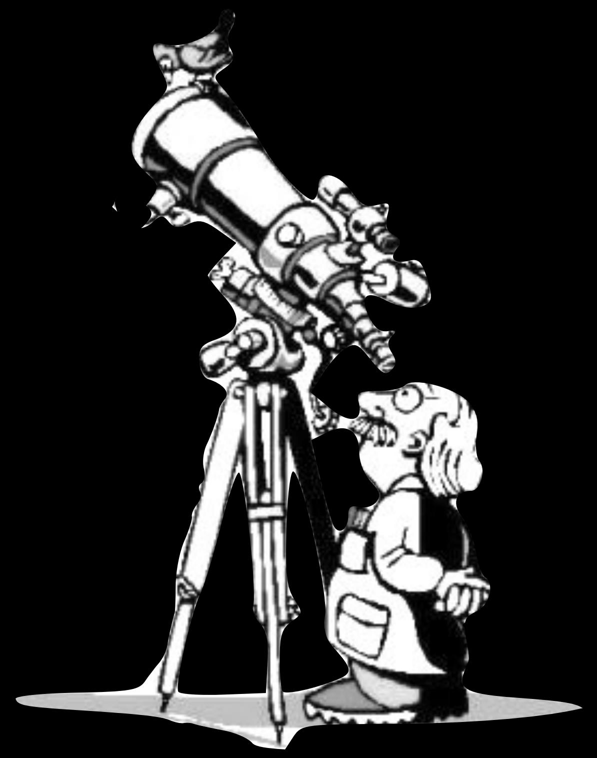
huge cloud cost | expensive transitions | application feasibility | environmental impact

complexity

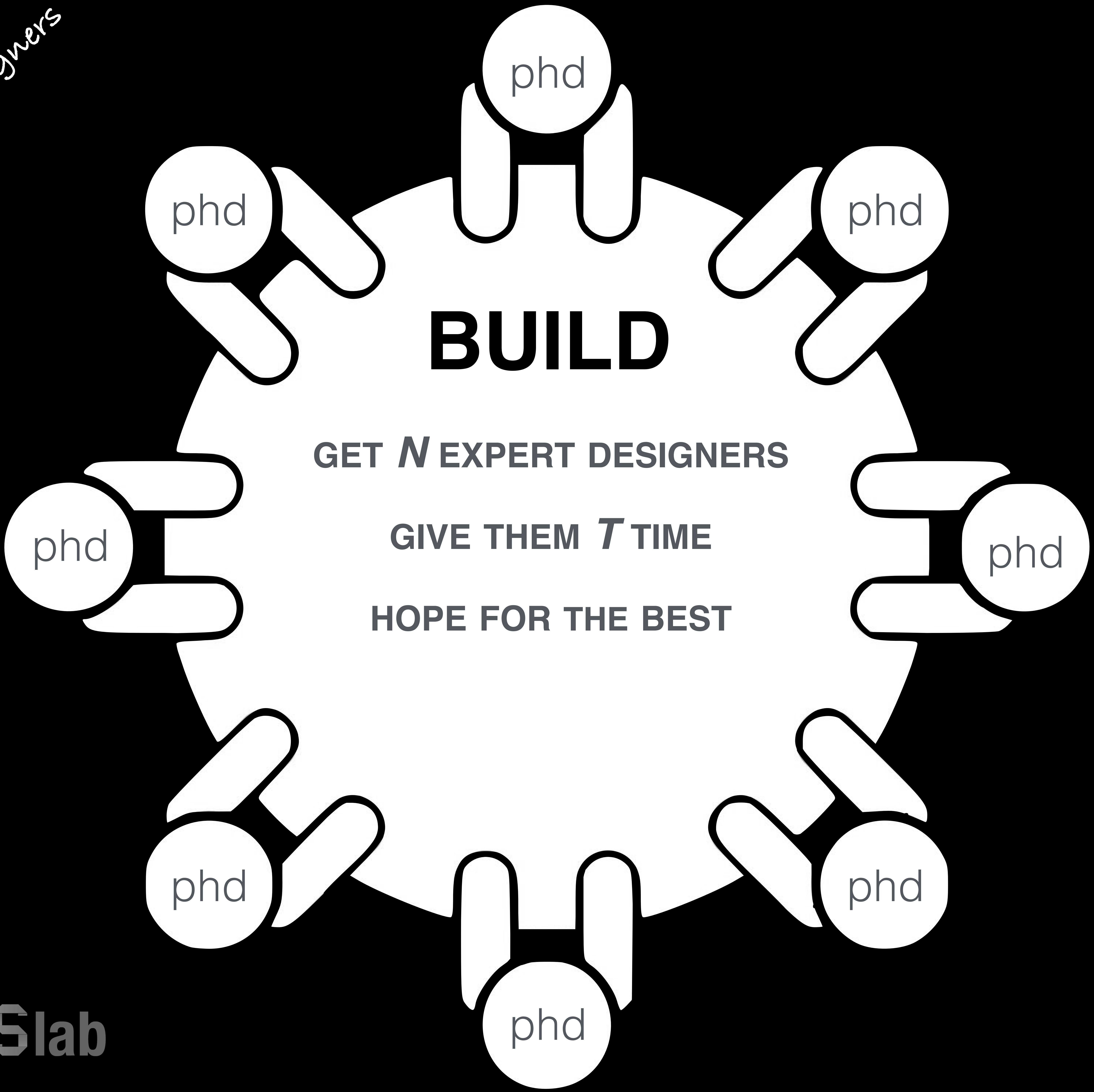
how we **BUILD** systems



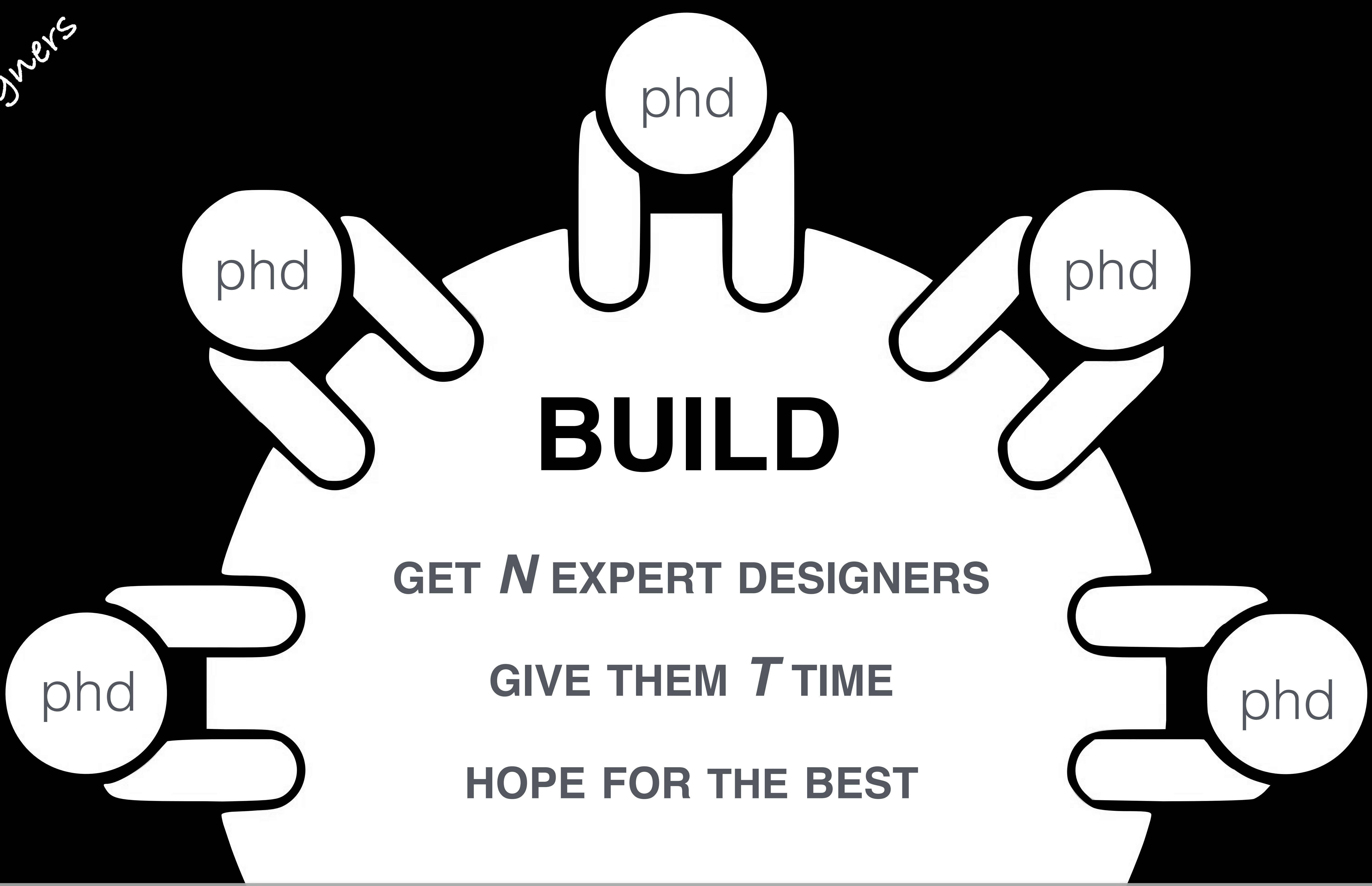


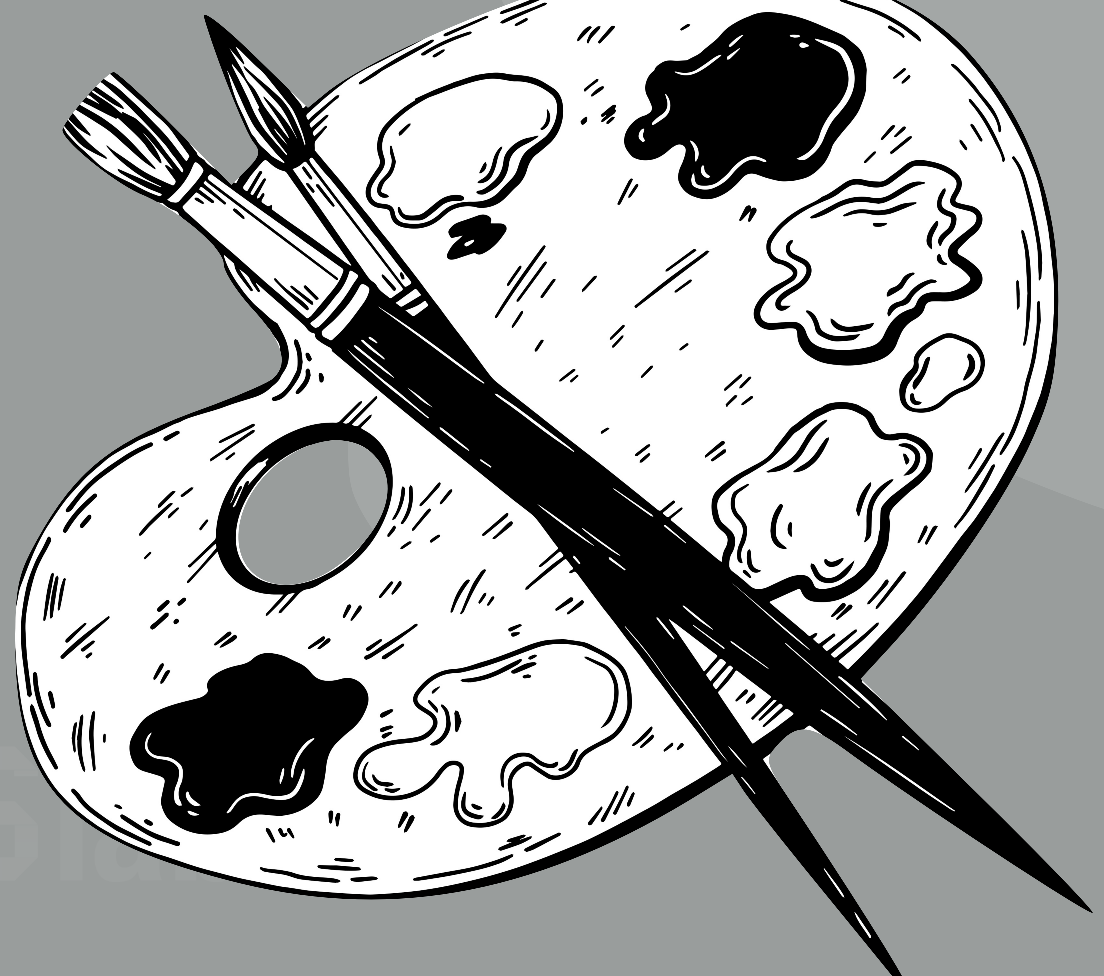


the dinning systems designers



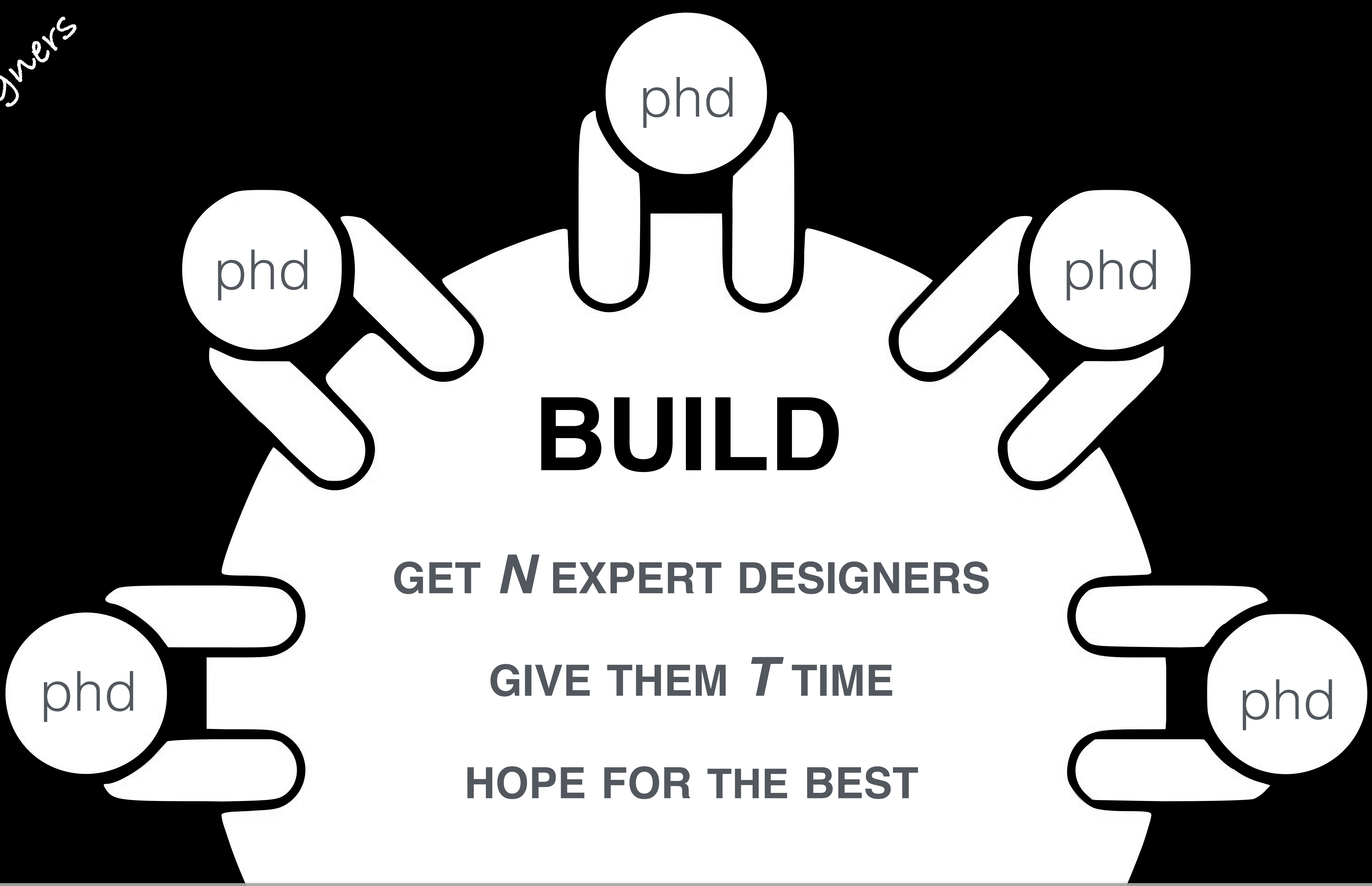
the dinning
systems designers

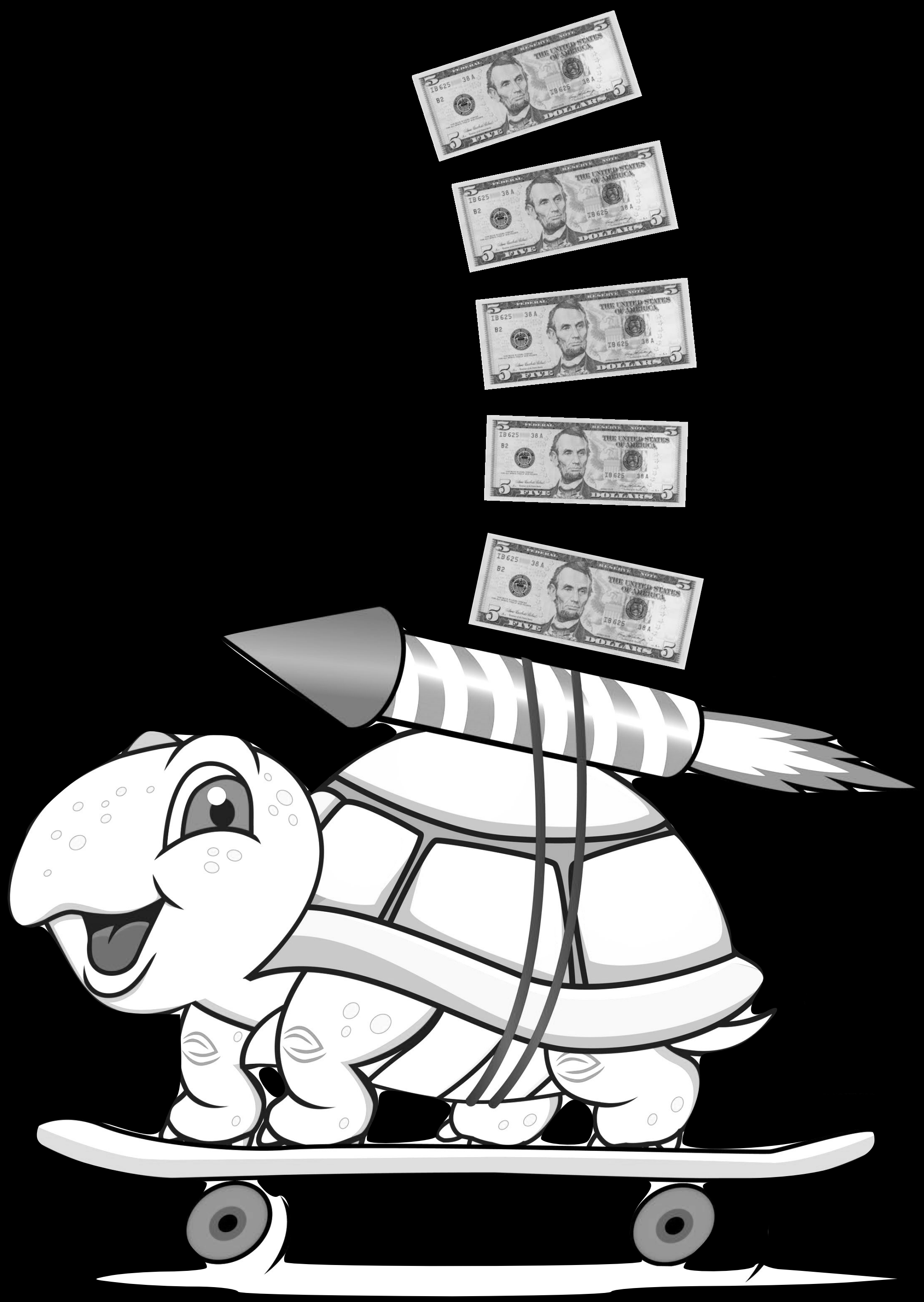


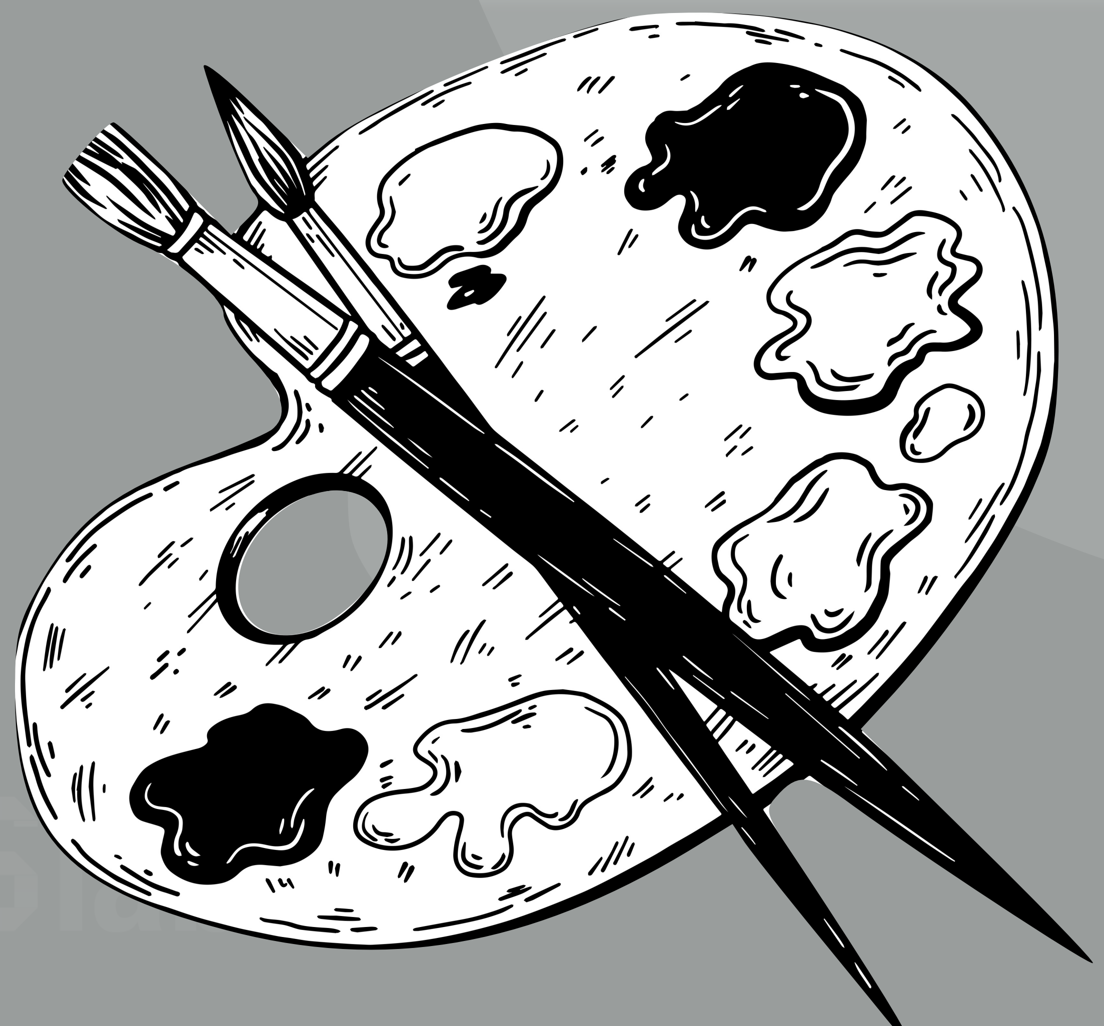


design is an art

the dinning
systems designers







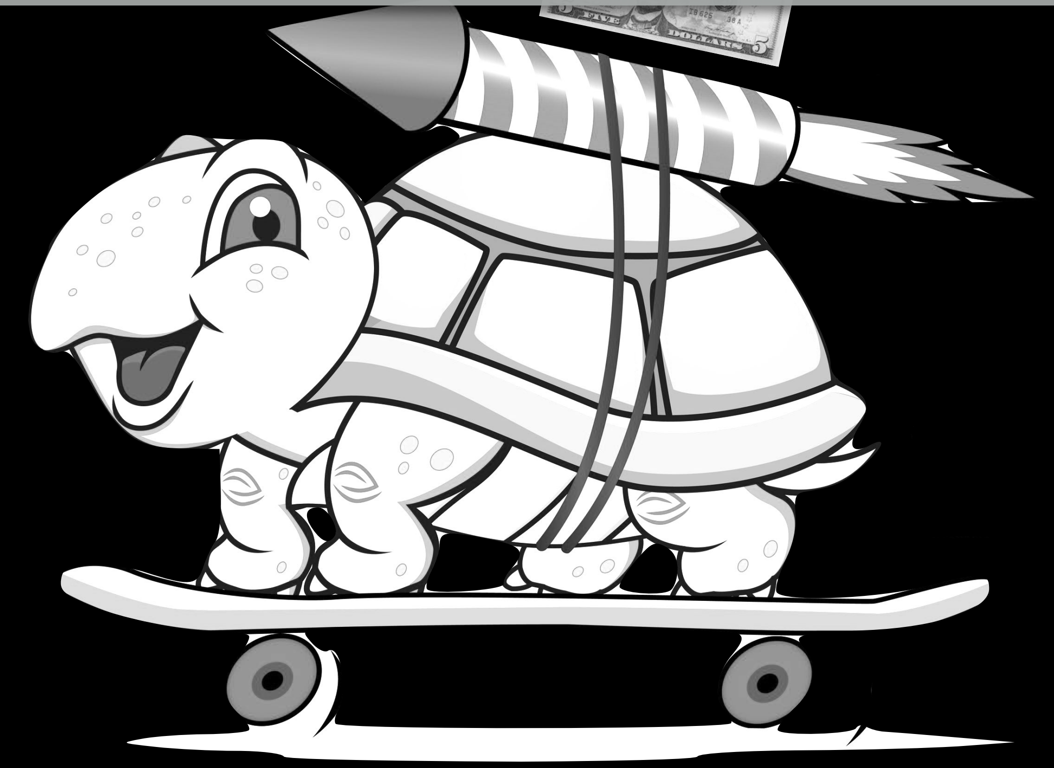
design is an art

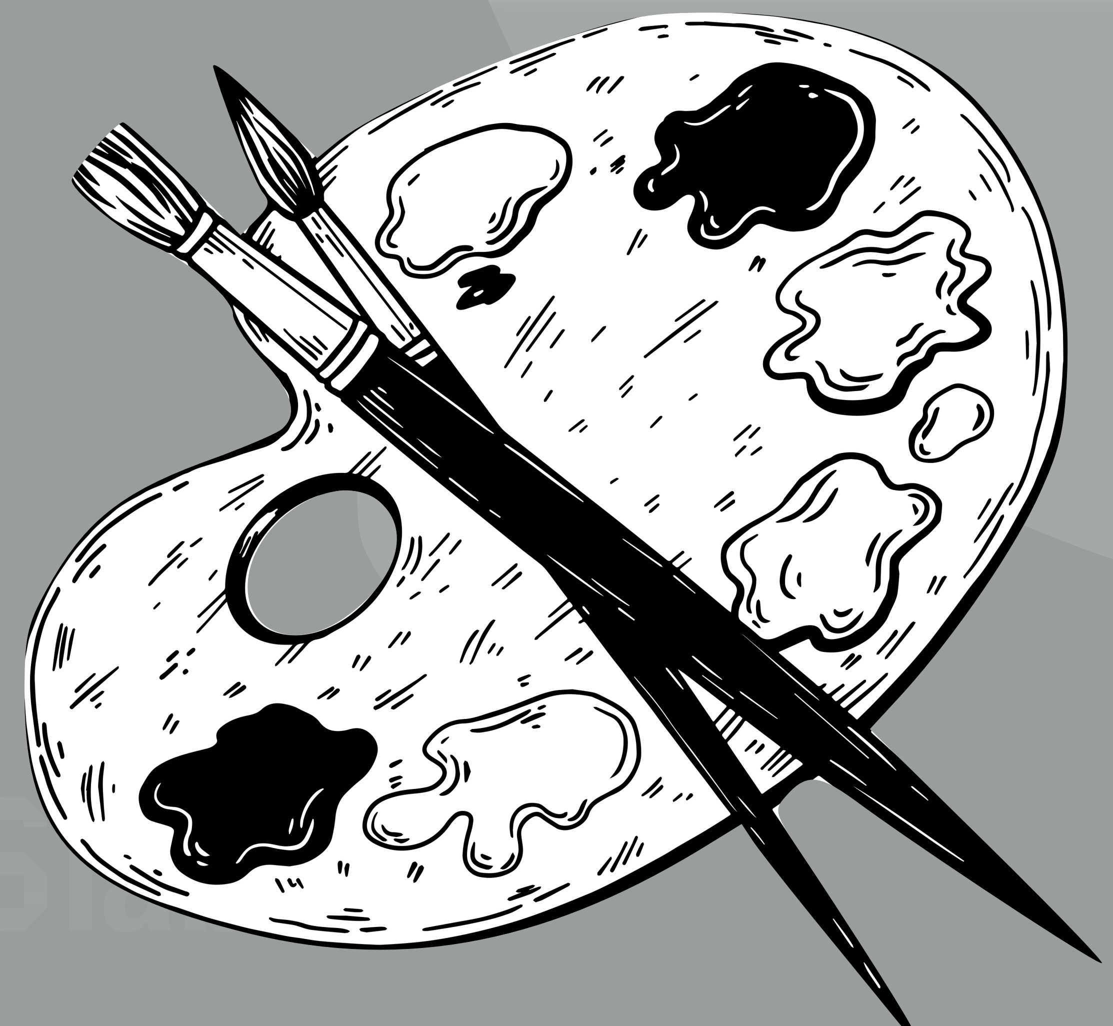
Design: 6-7 years
Reasoning: months/impossible

GET *N* EXPERT DESIGNERS

GIVE THEM *T* TIME

HOPE FOR THE BEST





design is an art

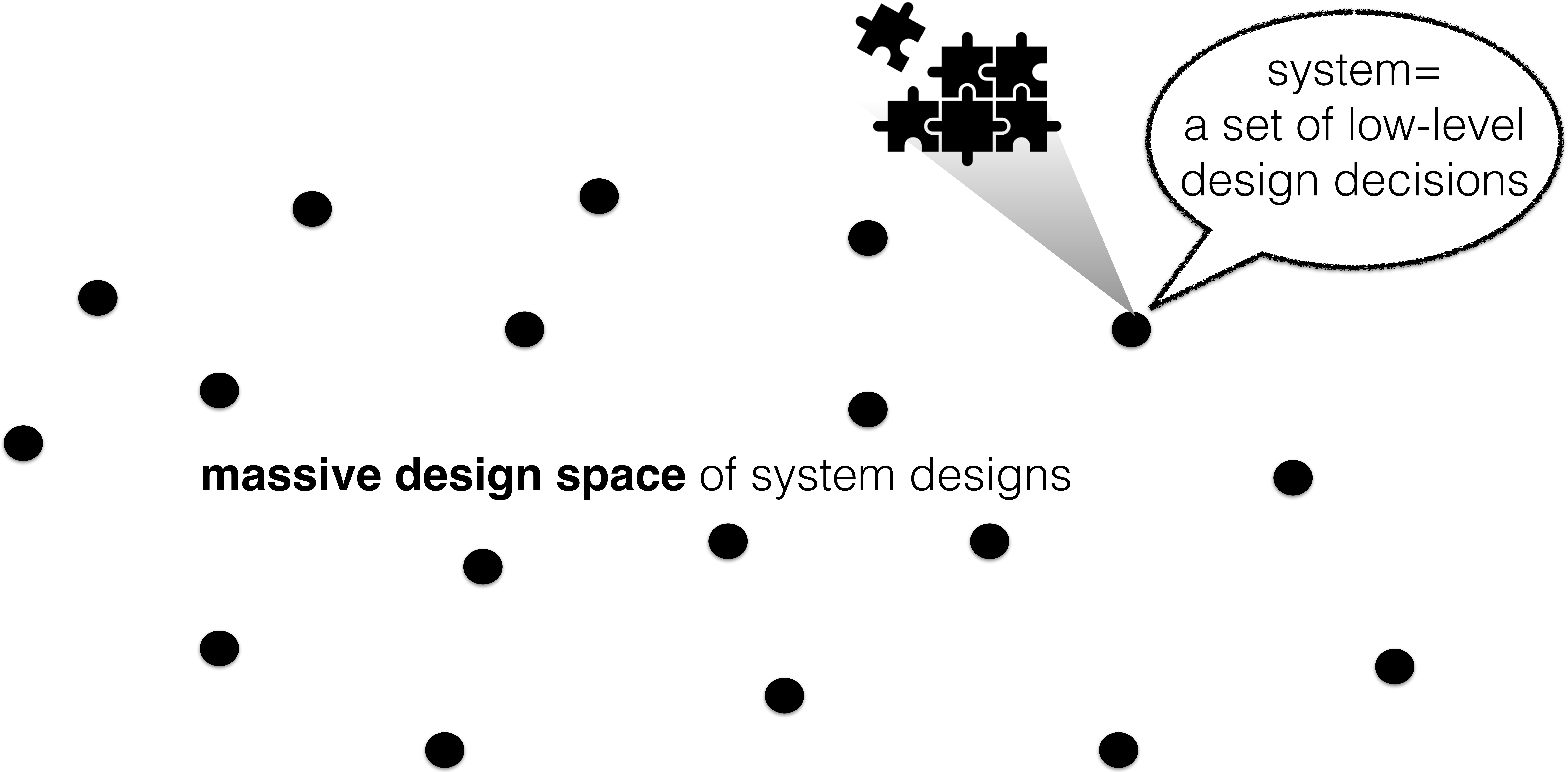
sign:
Re
mpossible
phd
phd
is an art


sign:
Re
mpossible
data
hardware
applications
years
phd
phd
is an art

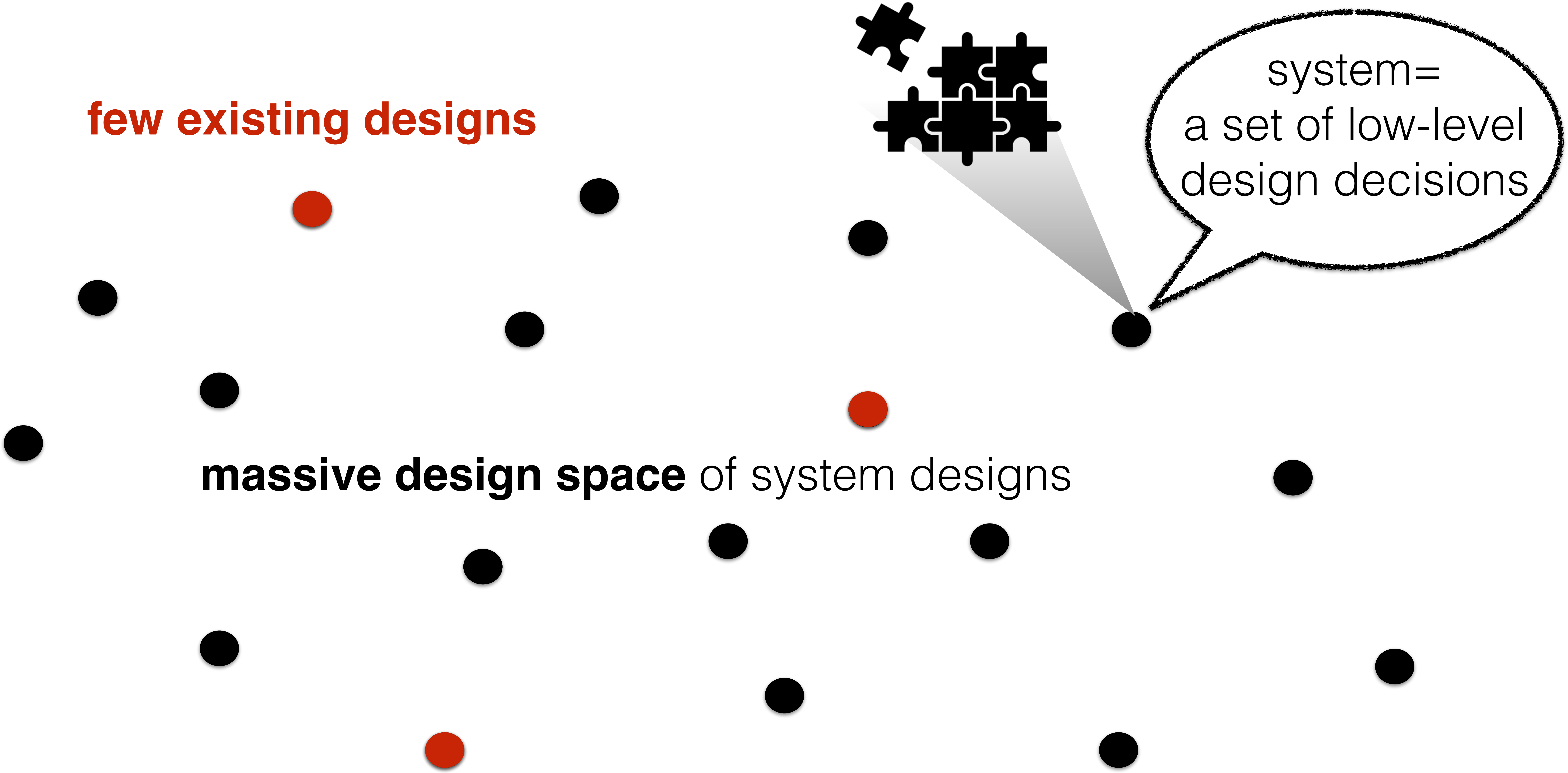
SELF-DESIGNING SYSTEMS

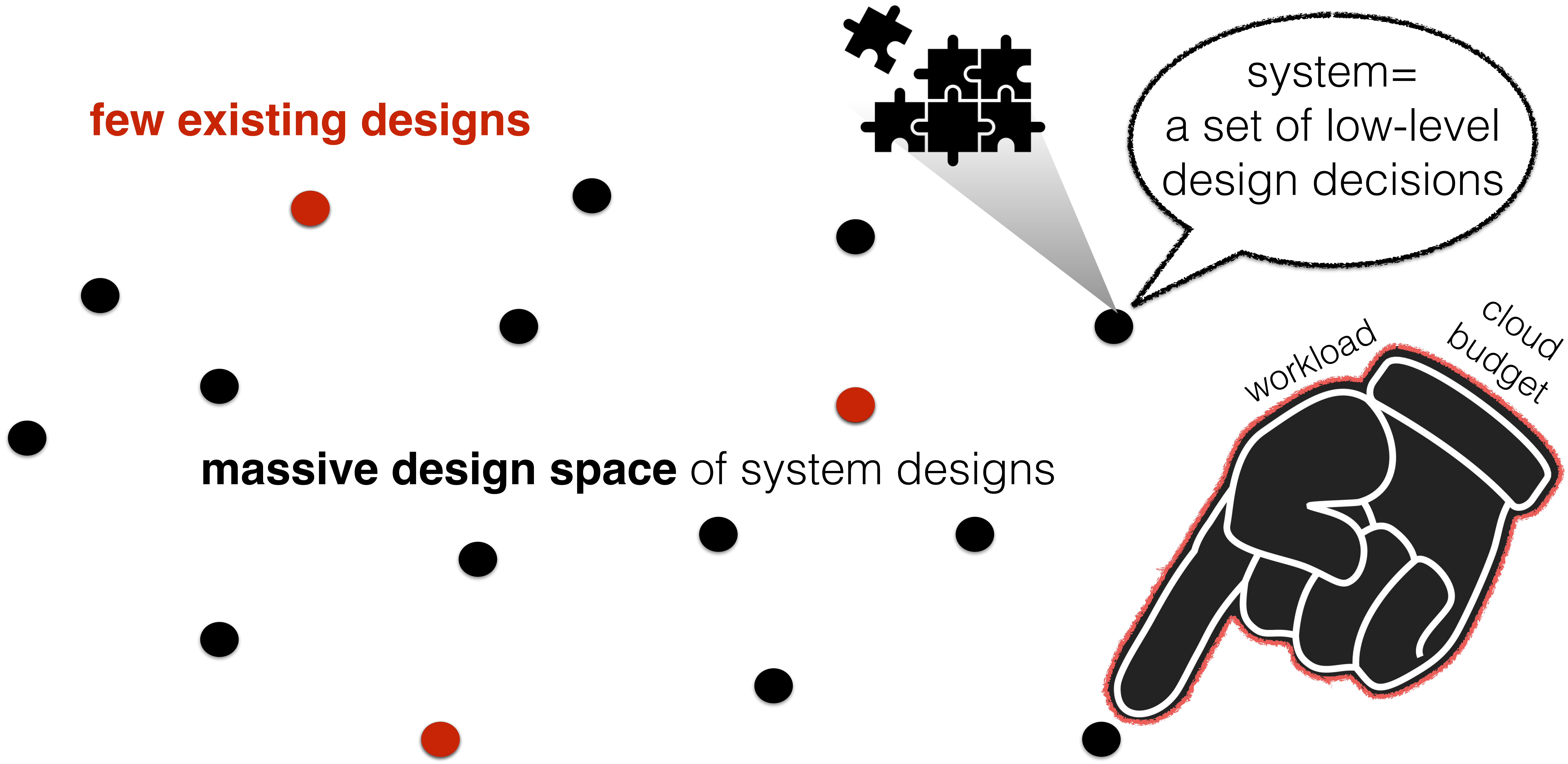
Automatically invent & build the perfect system for any new application

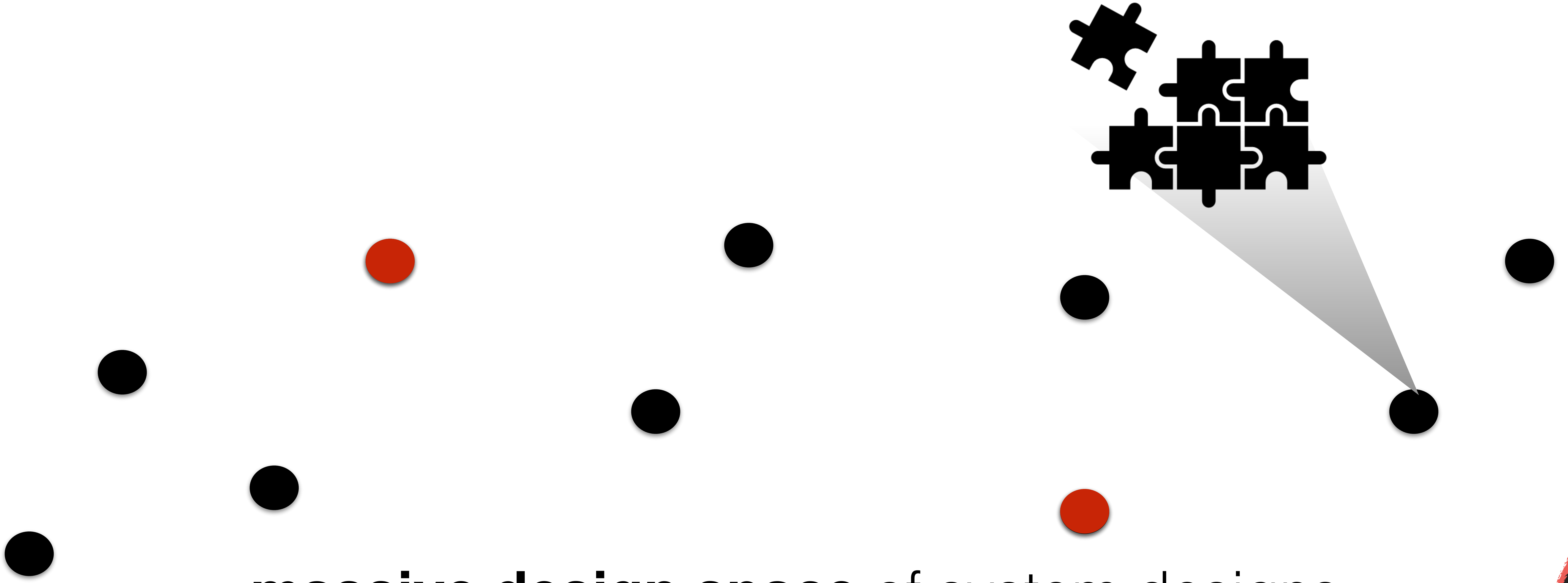
massive design space of system designs



DASlab
@ Harvard SEAS


few existing designs
system=
a set of low-level
design decisions
massive design space of system designs


few existing designs
massive design space of system designs
system=
a set of low-level
design decisions
workload
cloud
budget



massive design space of system designs

reasoning: understand all the design decisions & their impact

HOW DO WE START

HOW DO WE START

SLAs

robustness

concurrency

multi-tenancy

data types

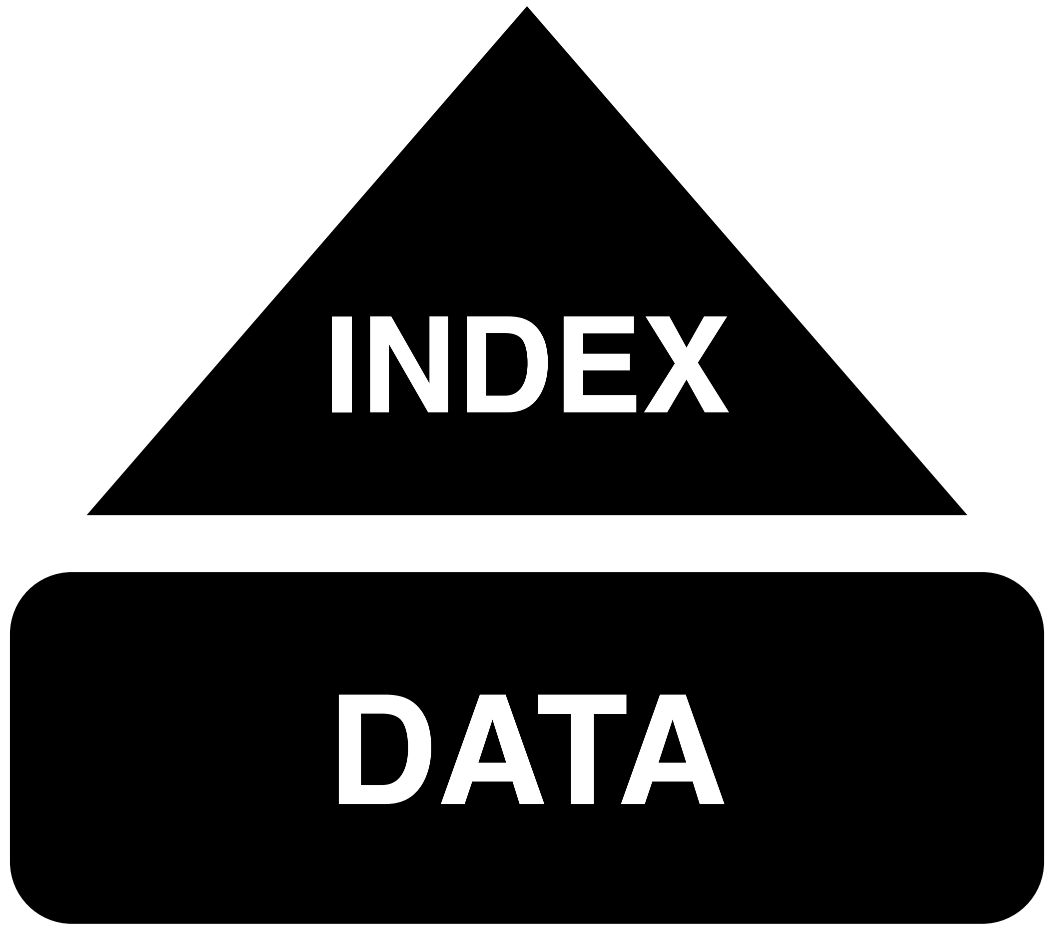
indexing

complex operations

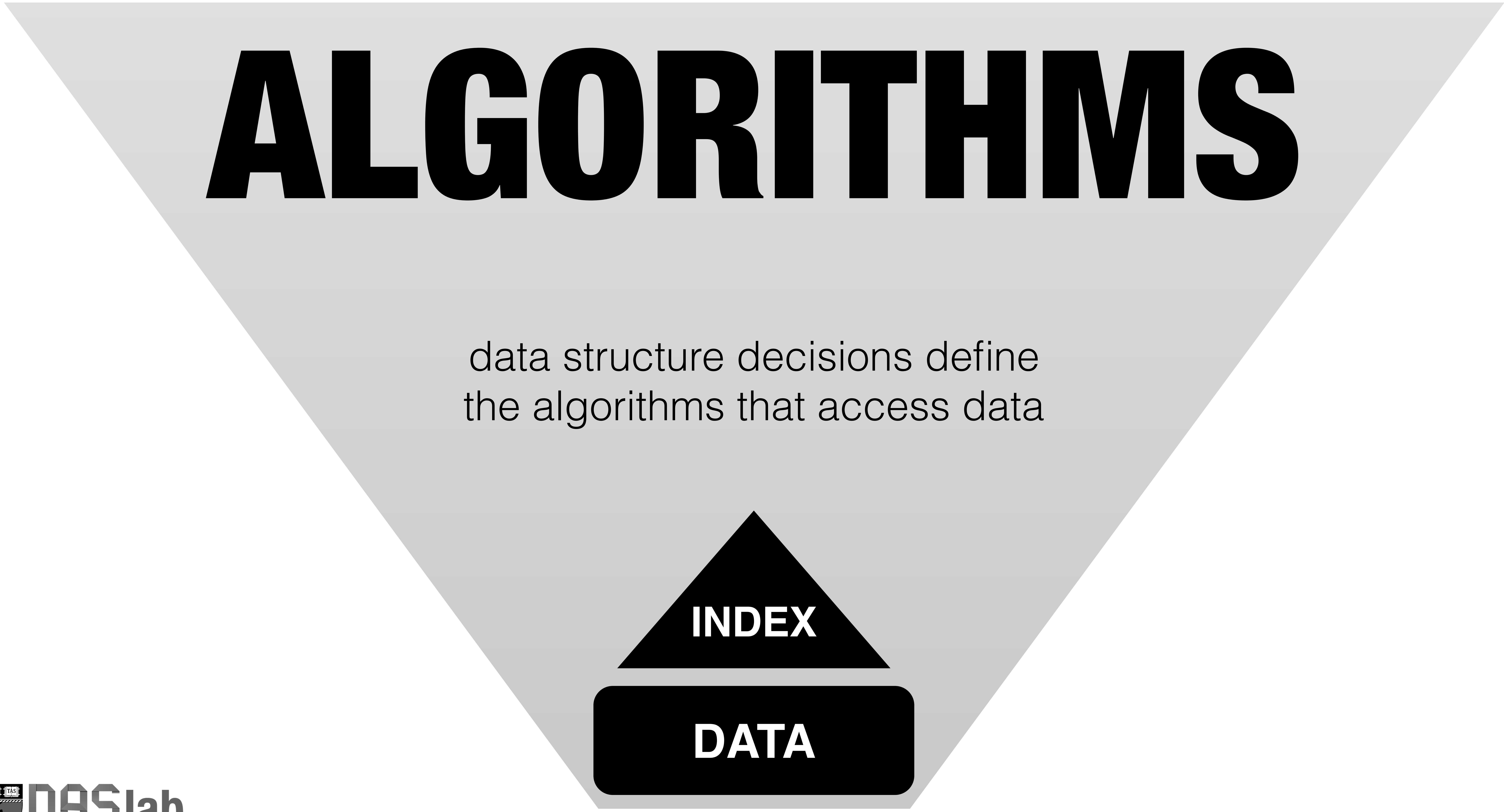
optimizer

cloud

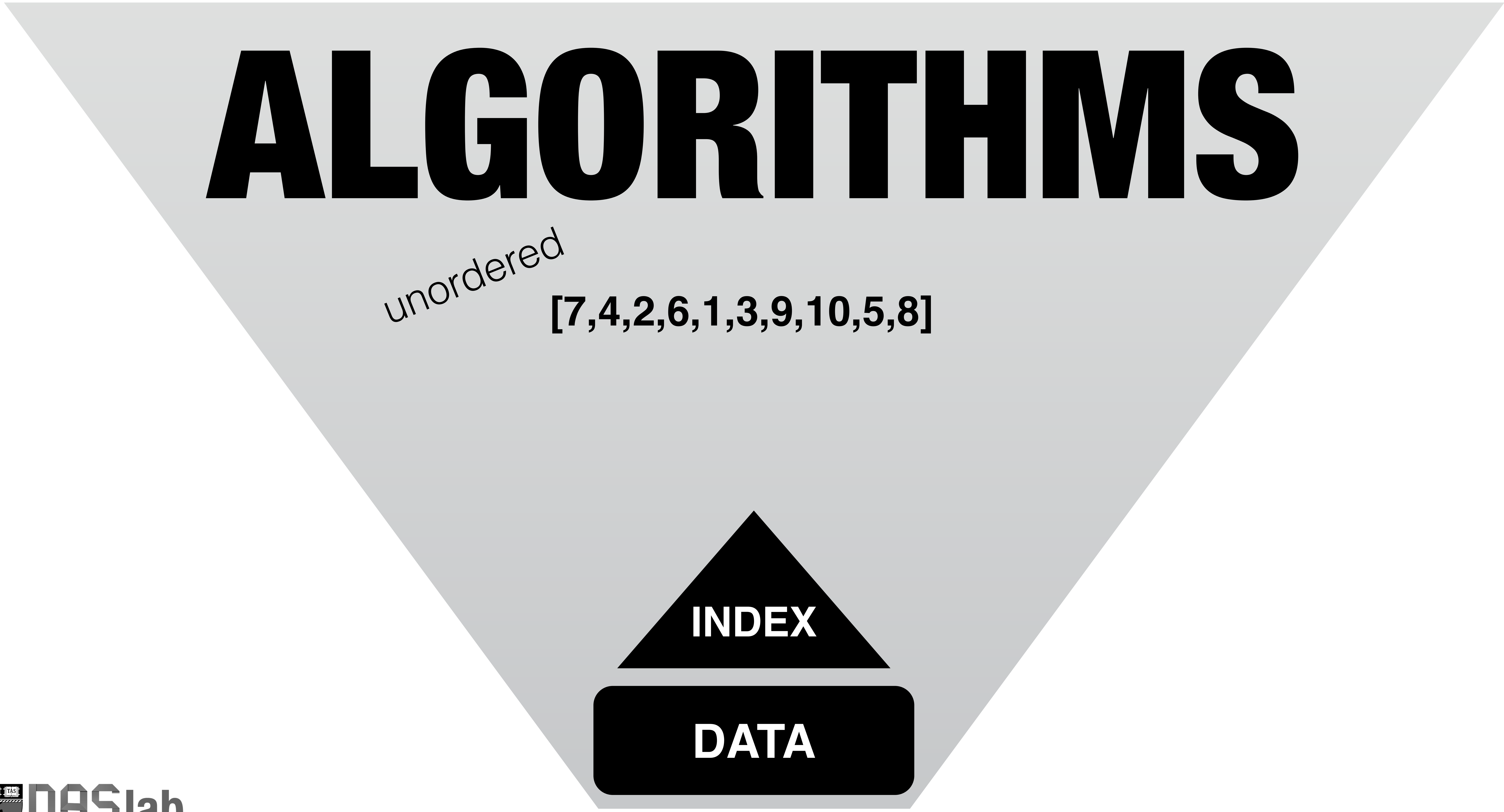
hardware

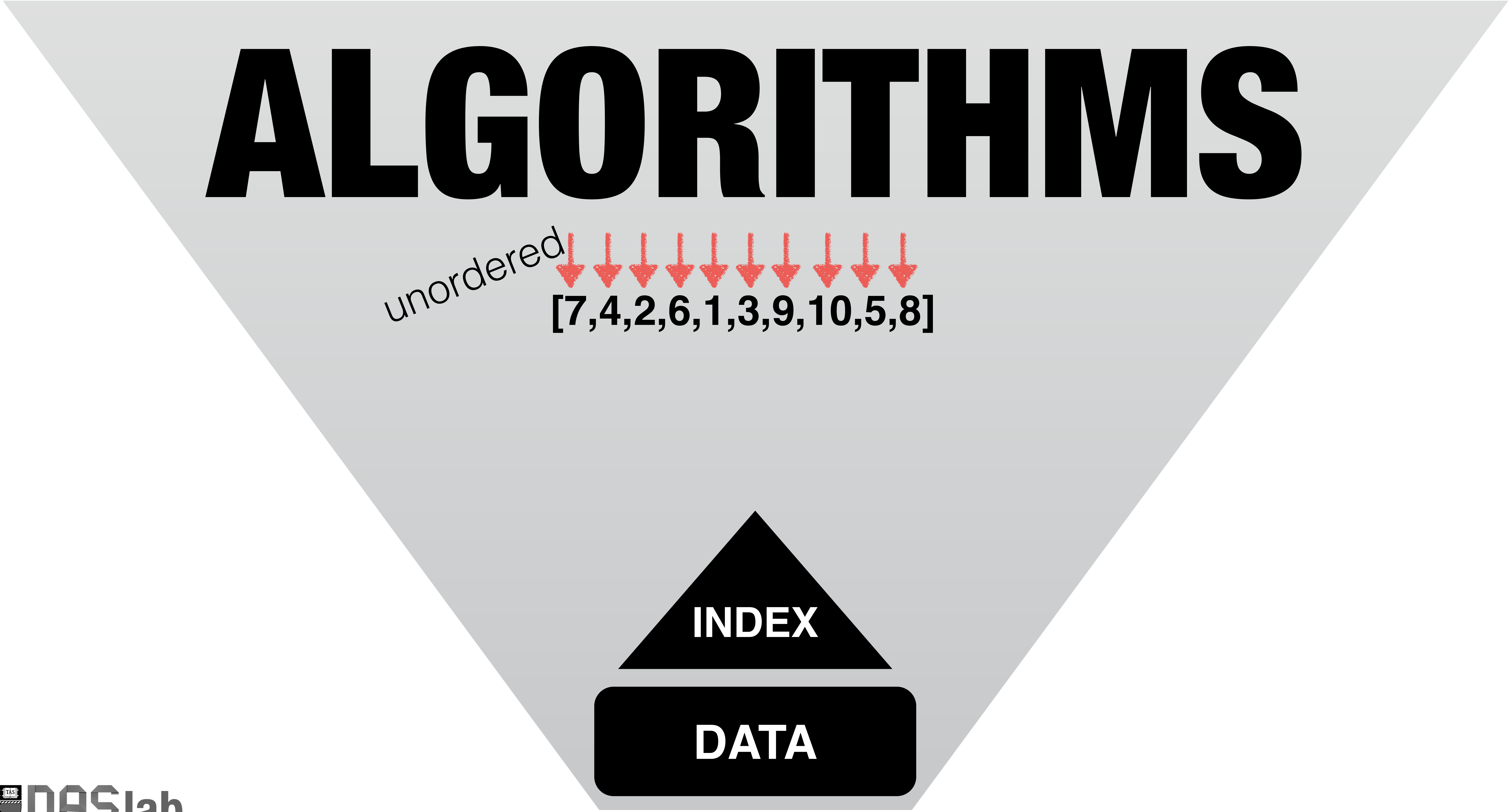


HOW
DO WE
START

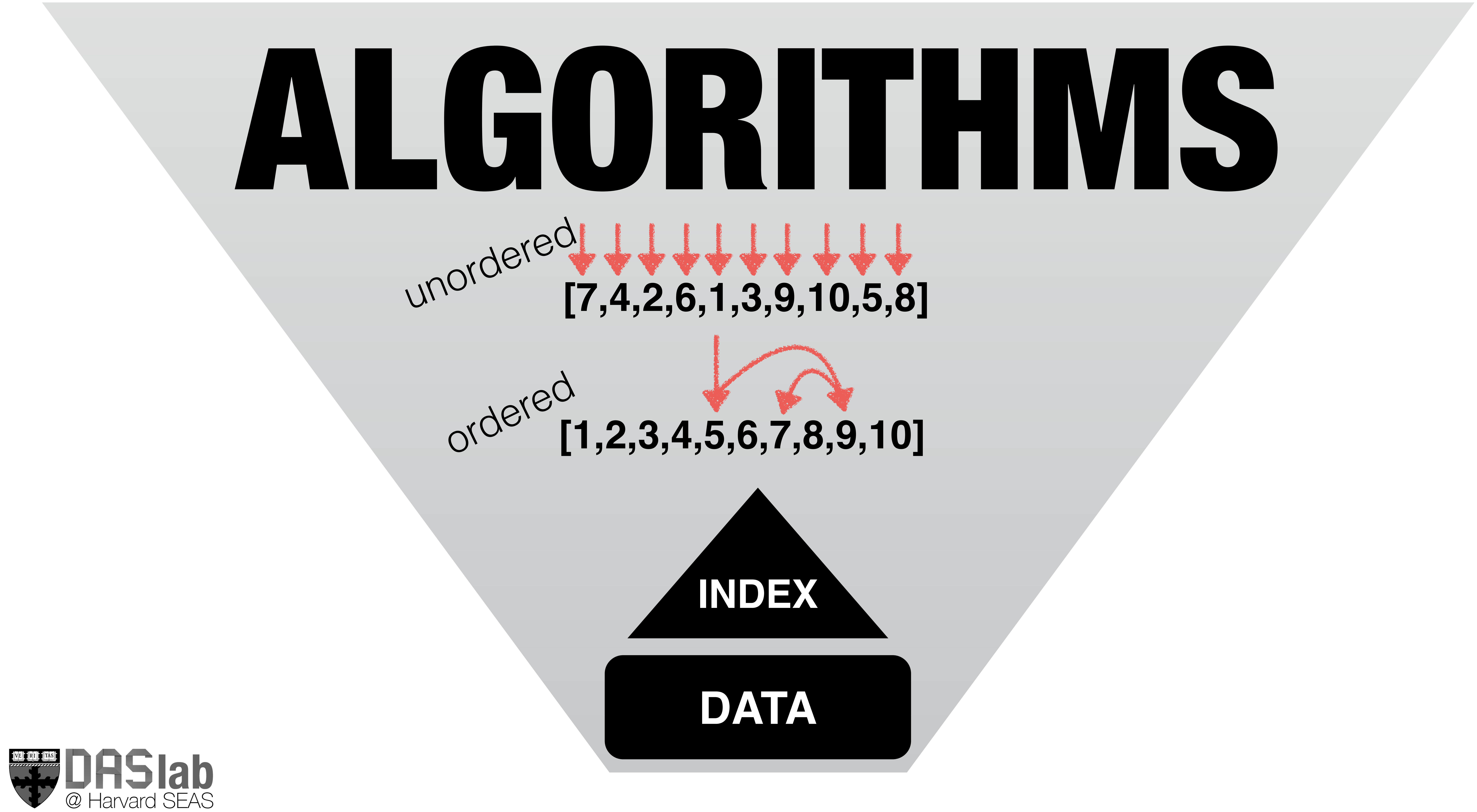


DASlab
@ Harvard SEAS


ALGORITHMS
unordered
[7,4,2,6,1,3,9,10,5,8]
INDEX
DATA

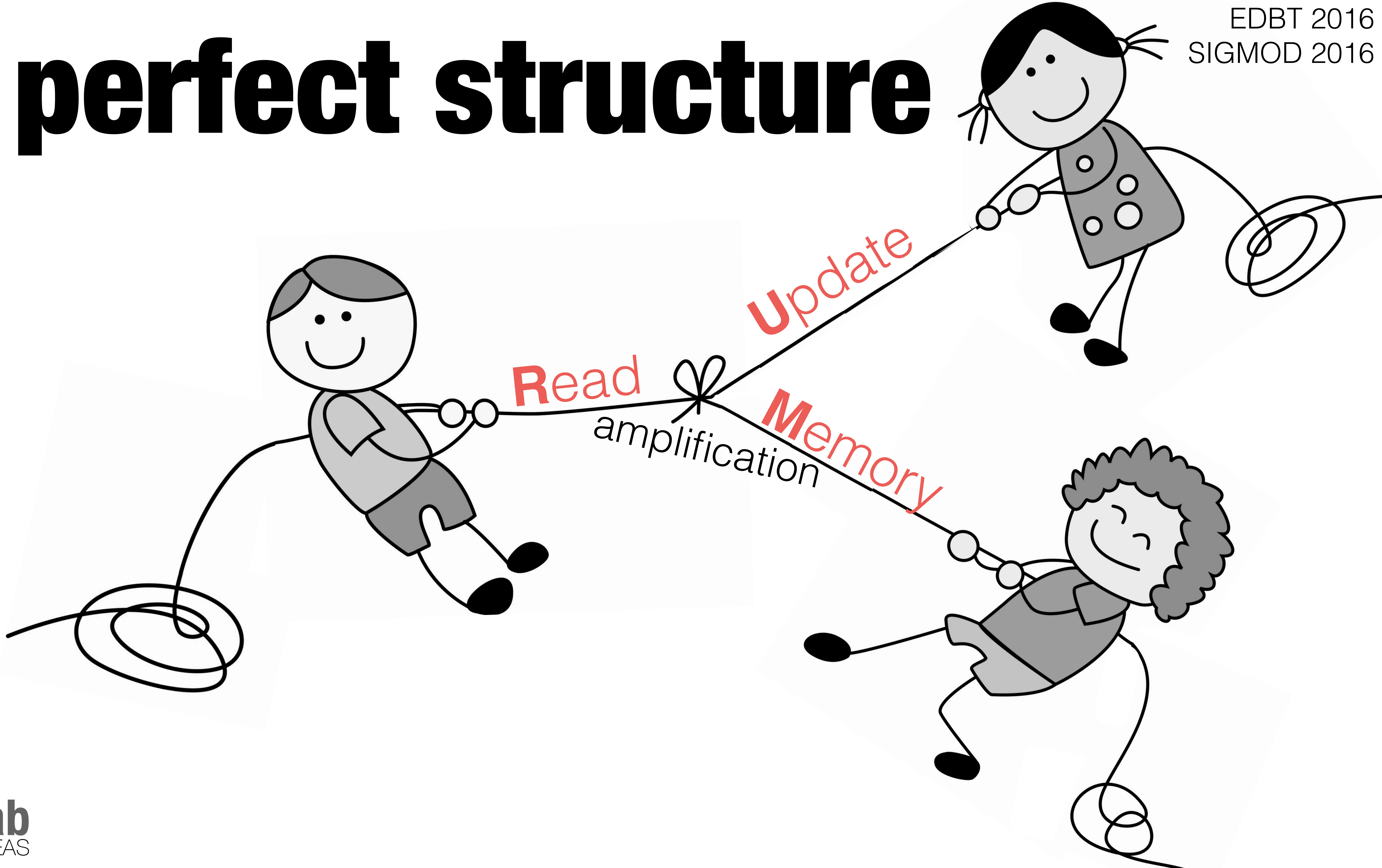


DASlab
@ Harvard SEAS


ALGORITHMS
unordered
[7,4,2,6,1,3,9,10,5,8]
ordered
[1,2,3,4,5,6,7,8,9,10]
INDEX
DATA
DASlab
@ Harvard SEAS

no perfect structure

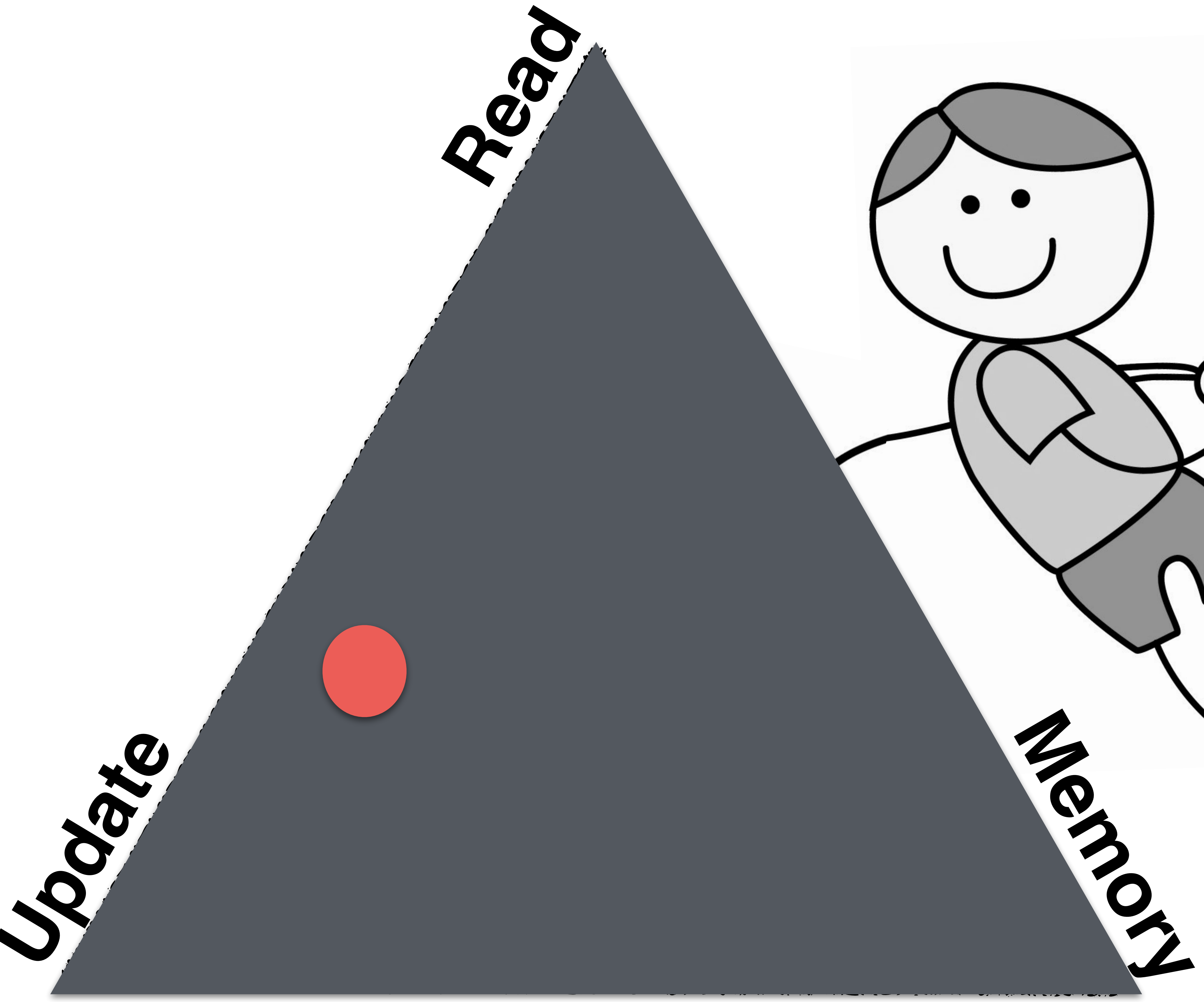
EDBT 2016
SIGMOD 2016

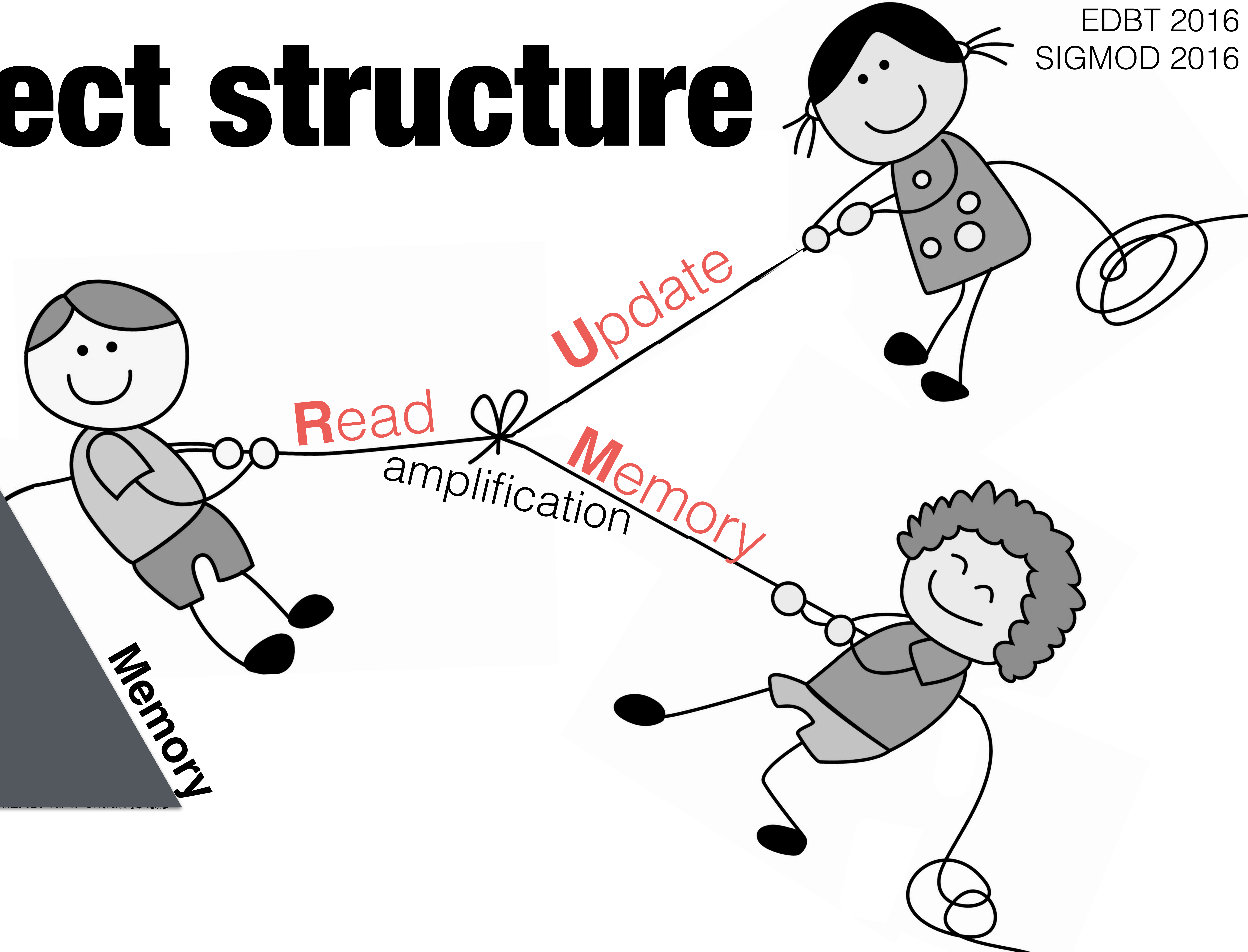


DASlab
@ Harvard SEAS

no perfect structure

EDBT 2016
SIGMOD 2016

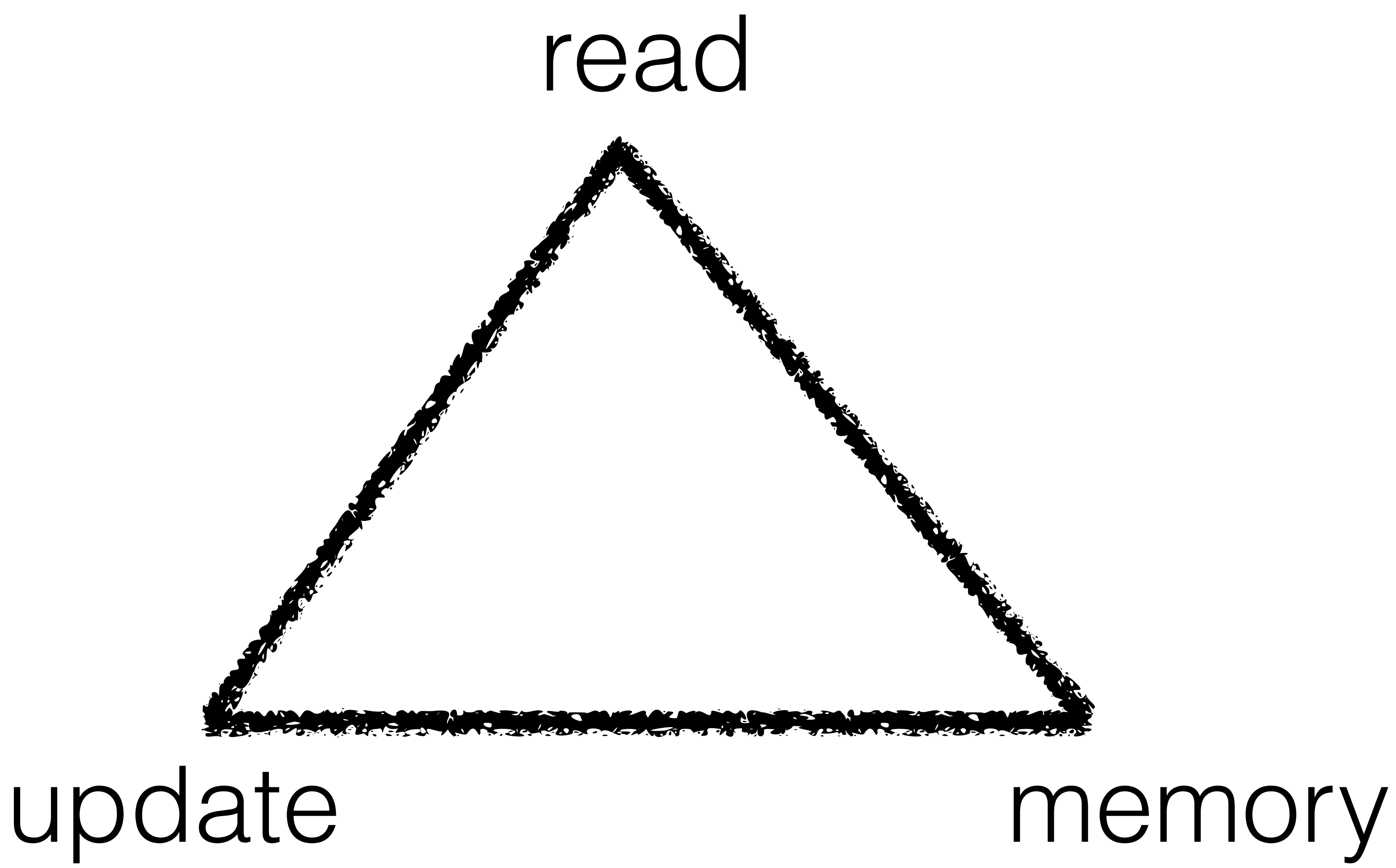


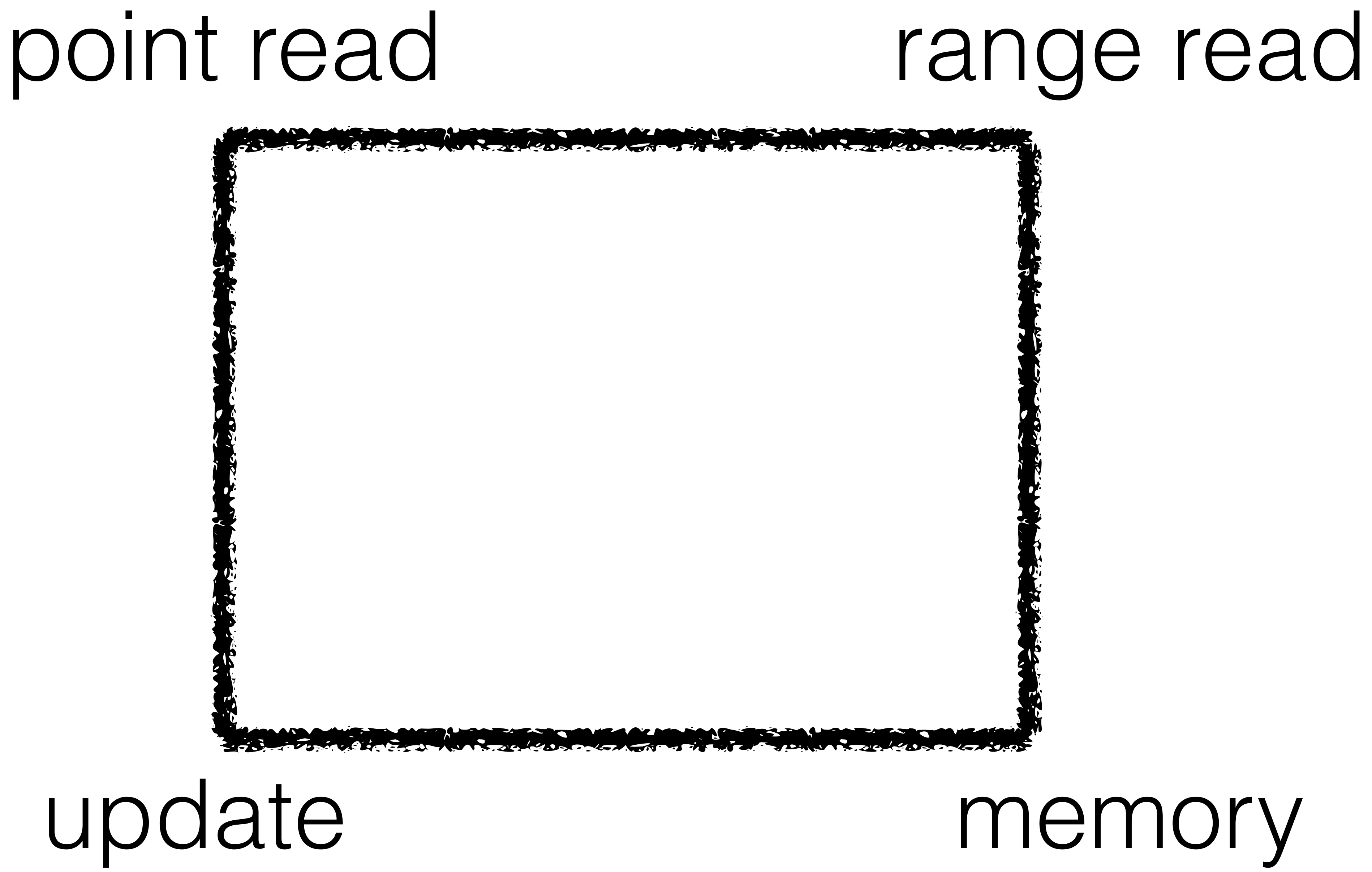


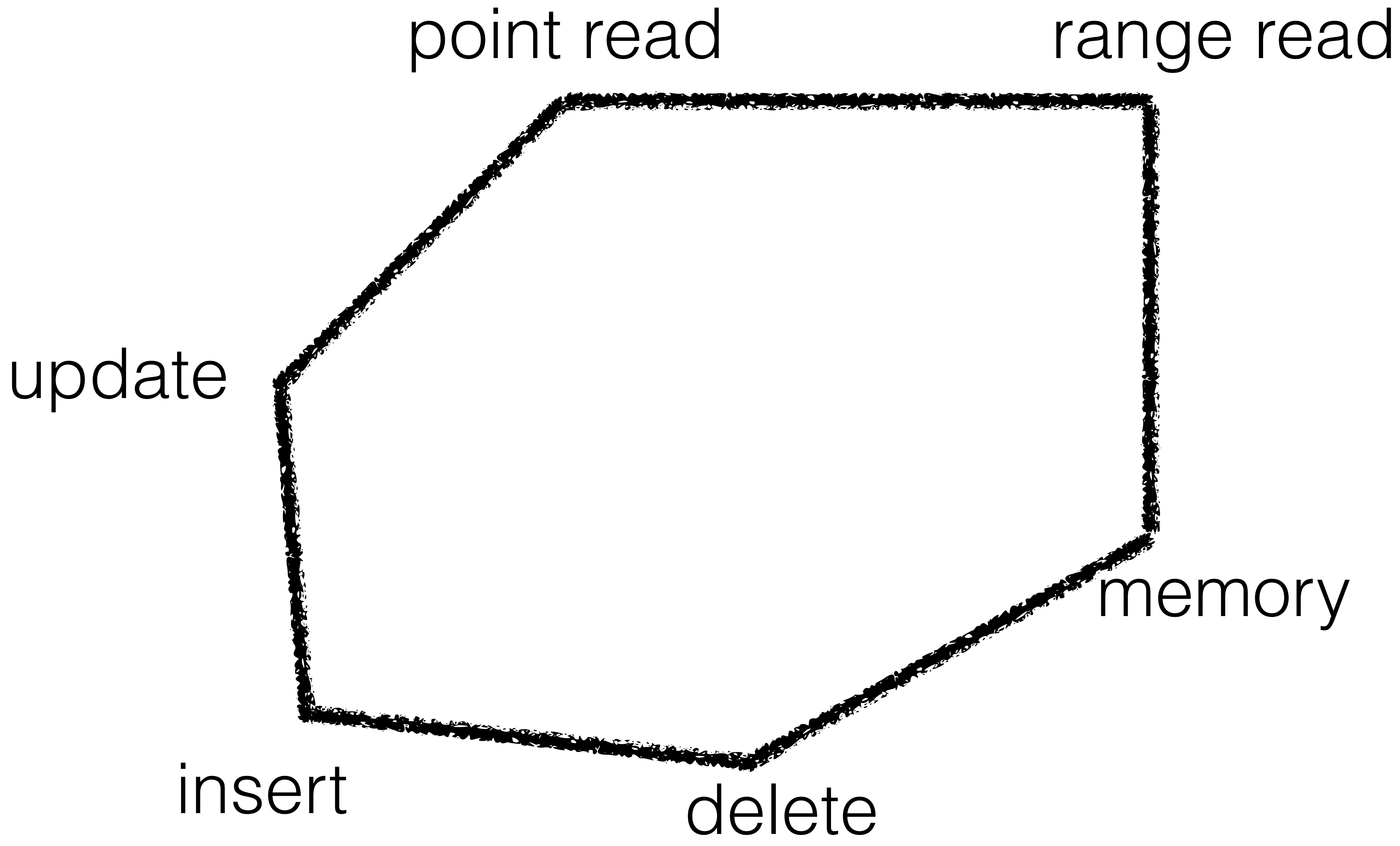
DASlab @ Harvard SEAS

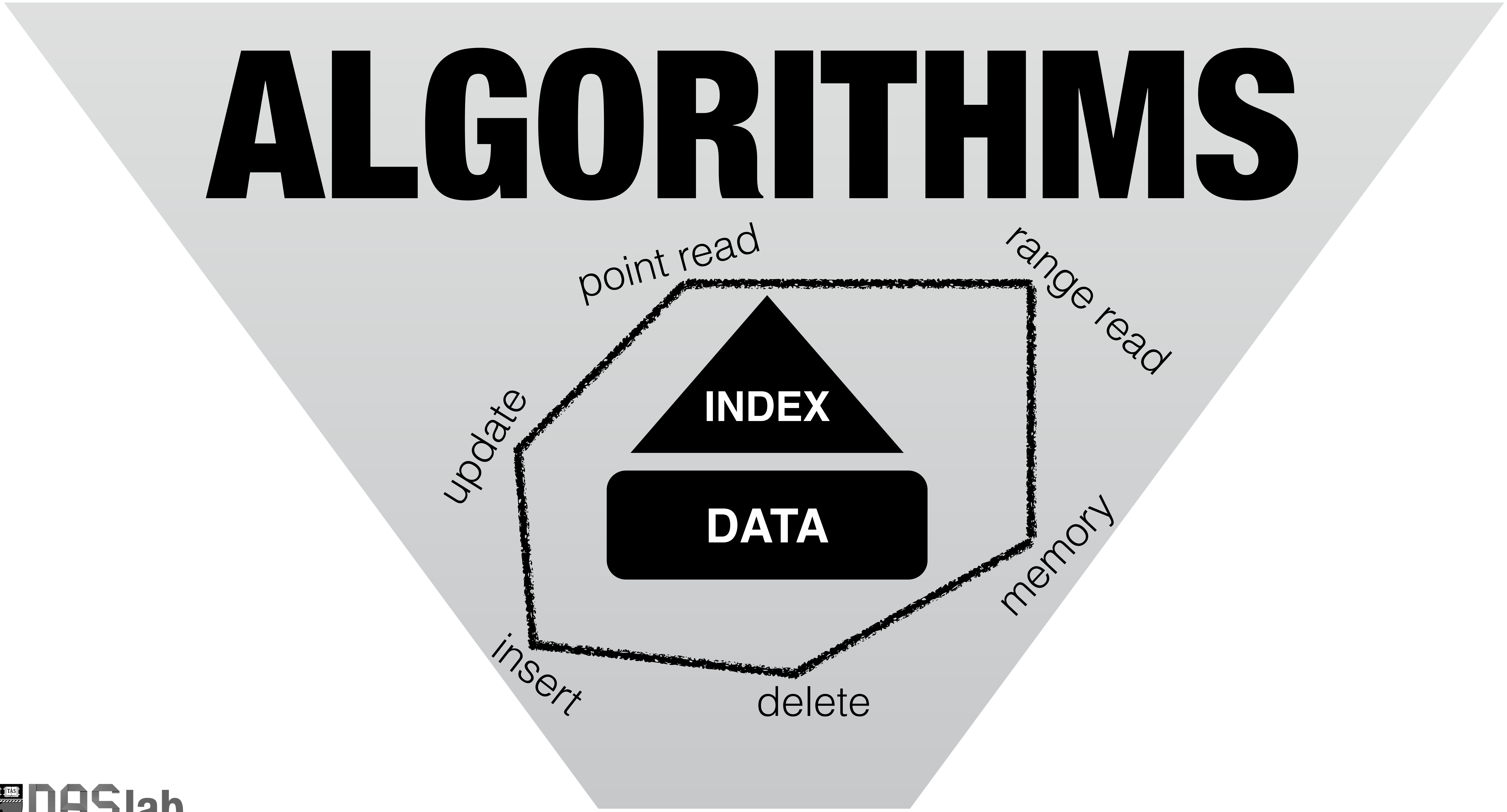
no perfect structure

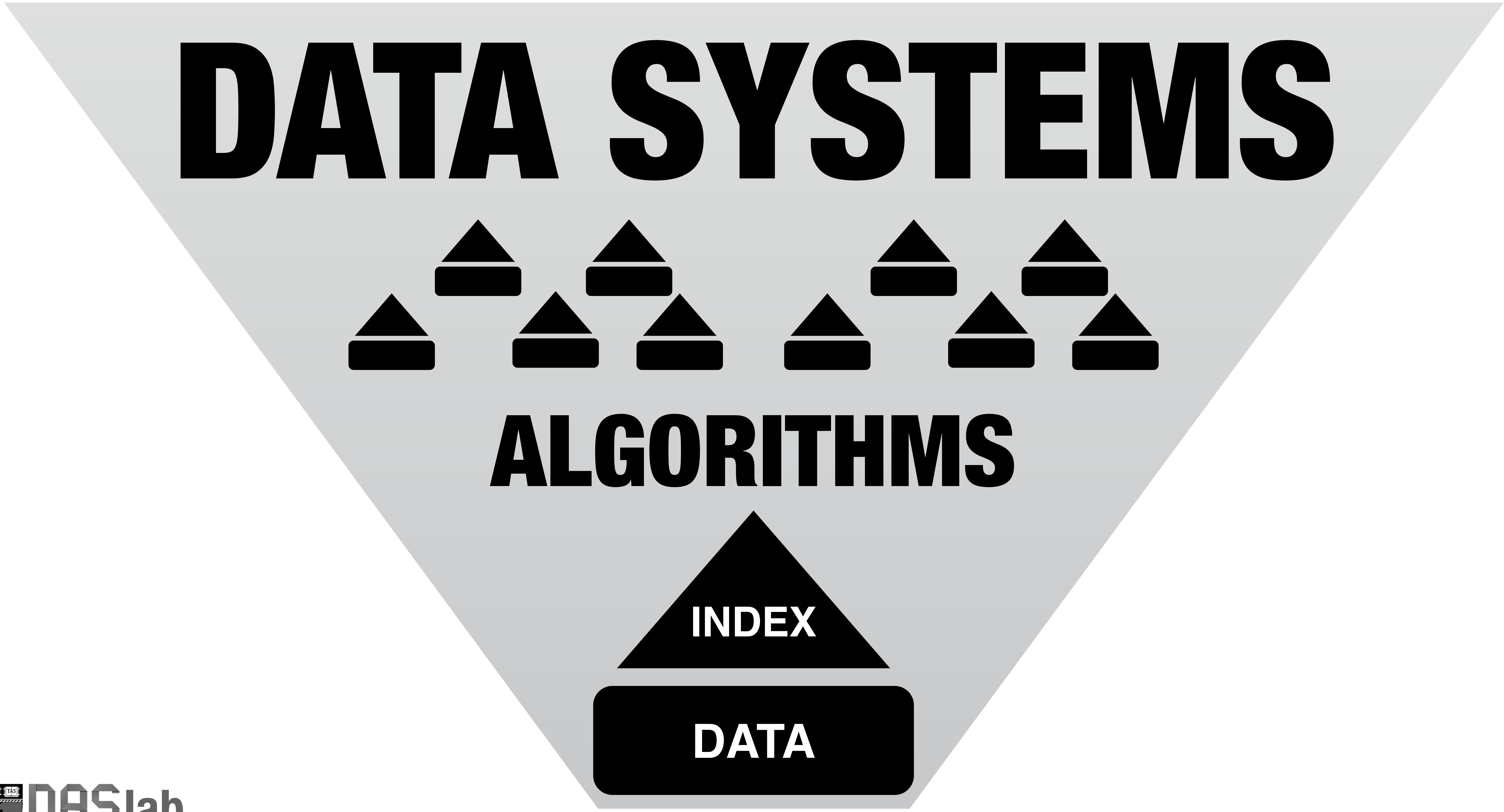
EDBT 2016
SIGMOD 2016


read
update
memory


point read
range read
update
memory


point read
range read
update
memory
insert
delete


ALGORITHMS
point read
range read
update
INDEX
DATA
memory
insert
delete



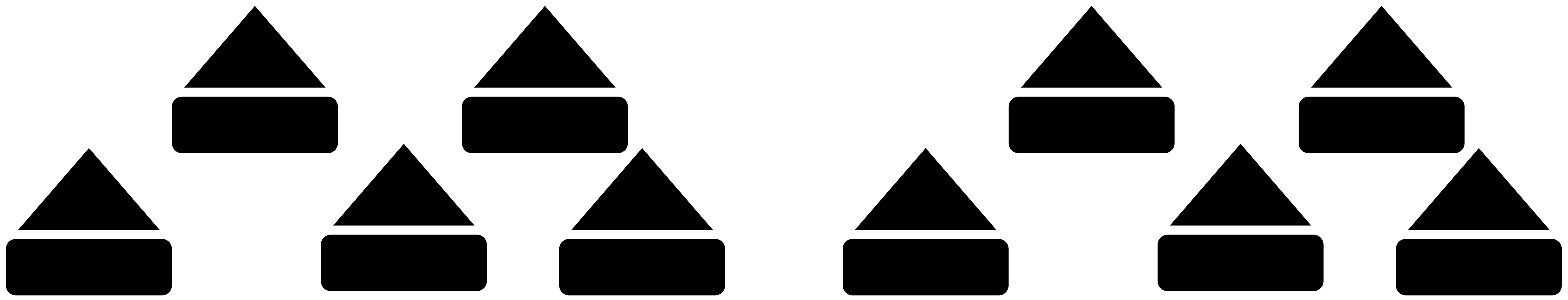
@ Harvard SEAS

NoSQL systems are the backbone of the BigData and AI era

LSM-tree
KV-stores

FACEBOOK, AMAZON, GOOGLE, TWITTER, LINKEDIN

MACHINE LEARNING, SQL, CRYPTO, SCIENCE

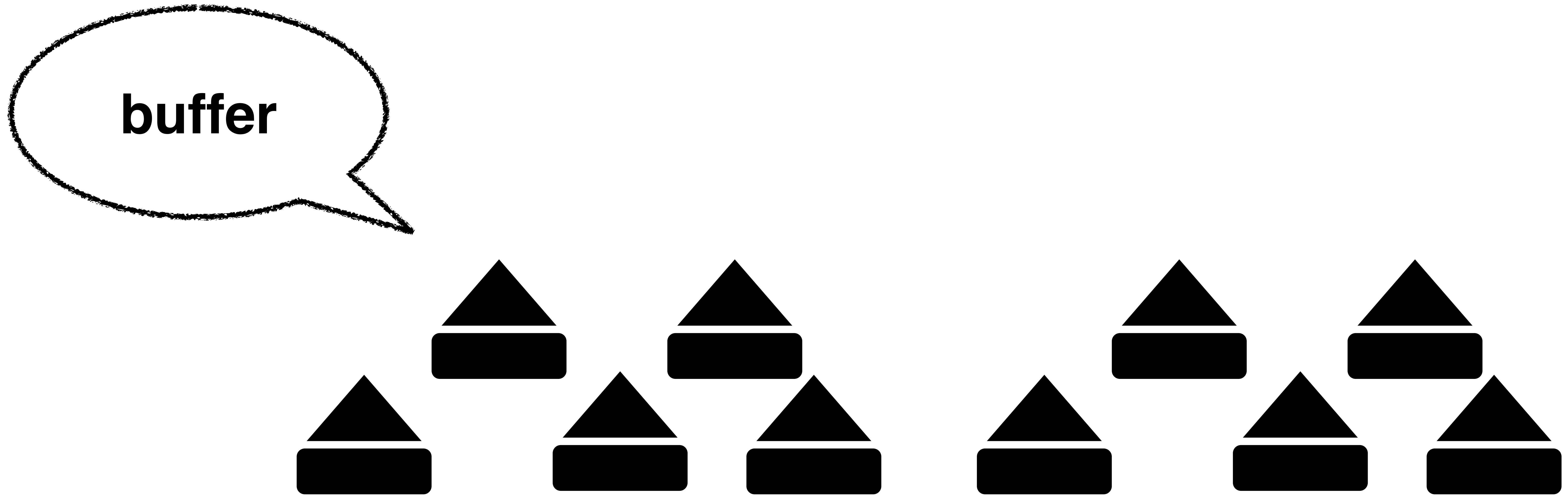


DASlab @ Harvard SEAS

NoSQL systems are the backbone of the BigData and AI era

LSM-tree
KV-stores

FACEBOOK, AMAZON, GOOGLE, TWITTER, LINKEDIN
MACHINE LEARNING, SQL, CRYPTO, SCIENCE

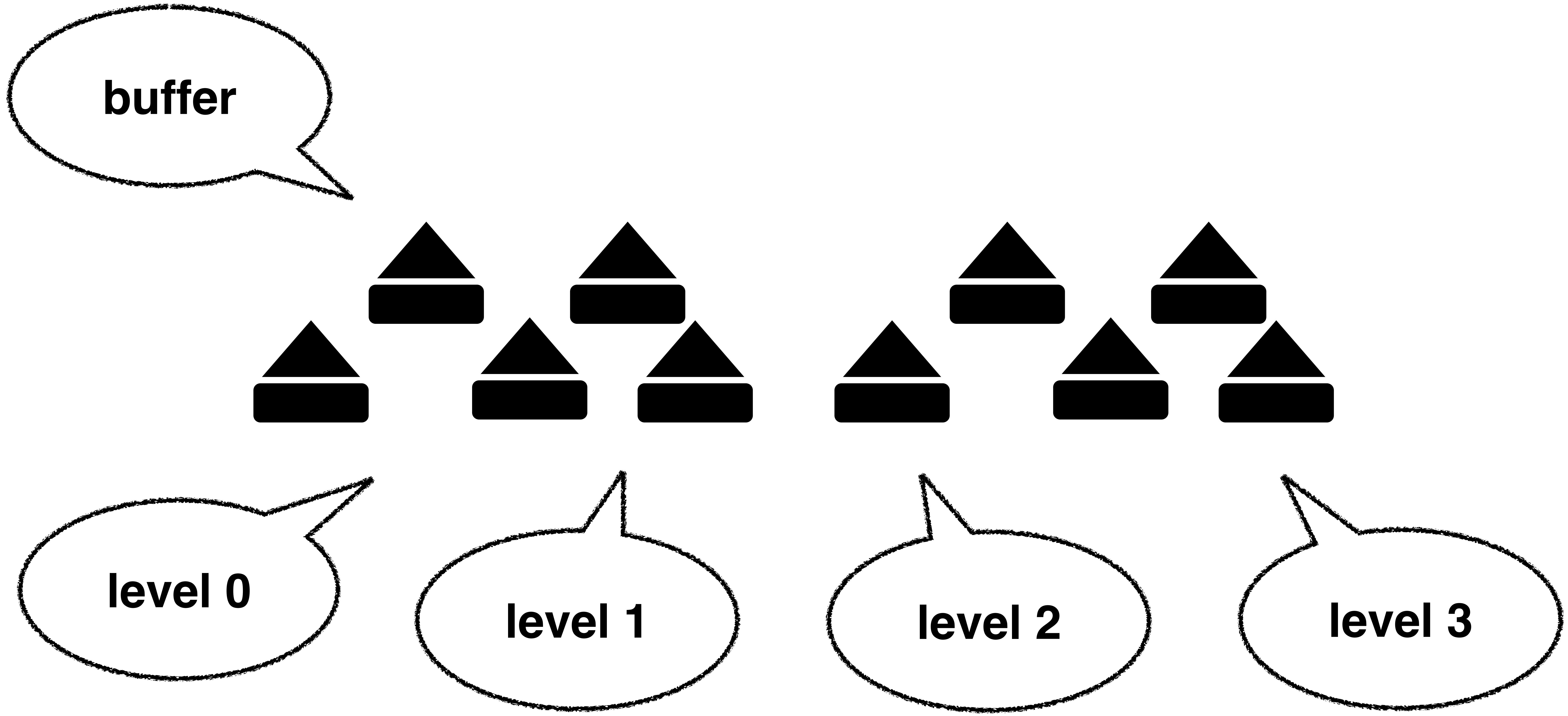


NoSQL systems are the backbone of the BigData and AI era

LSM-tree
KV-stores

FACEBOOK, AMAZON, GOOGLE, TWITTER, LINKEDIN

MACHINE LEARNING, SQL, CRYPTO, SCIENCE



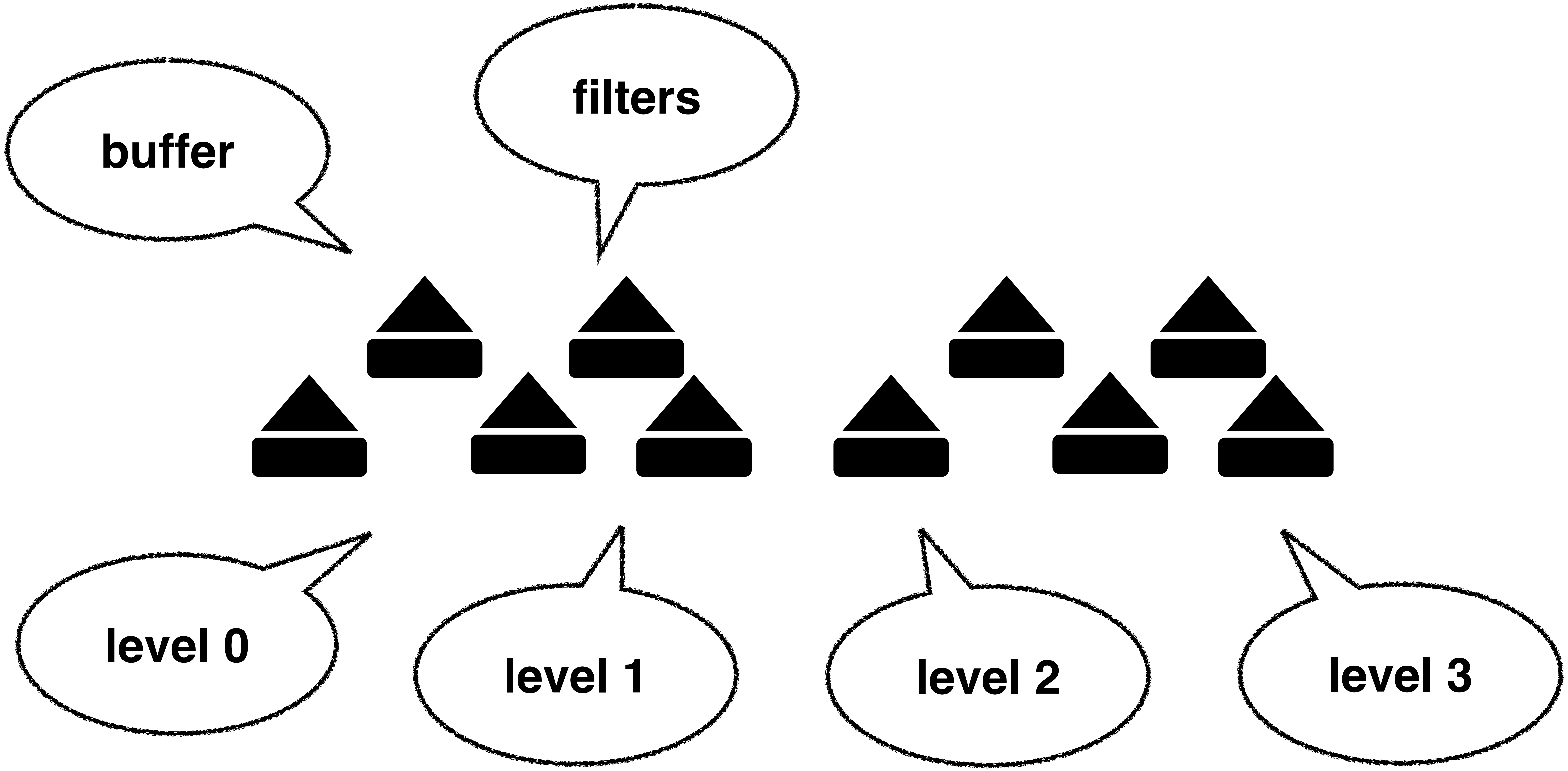
DASlab
@ Harvard SEAS

NoSQL systems are the backbone of the BigData and AI era

LSM-tree
KV-stores

FACEBOOK, AMAZON, GOOGLE, TWITTER, LINKEDIN

MACHINE LEARNING, SQL, CRYPTO, SCIENCE



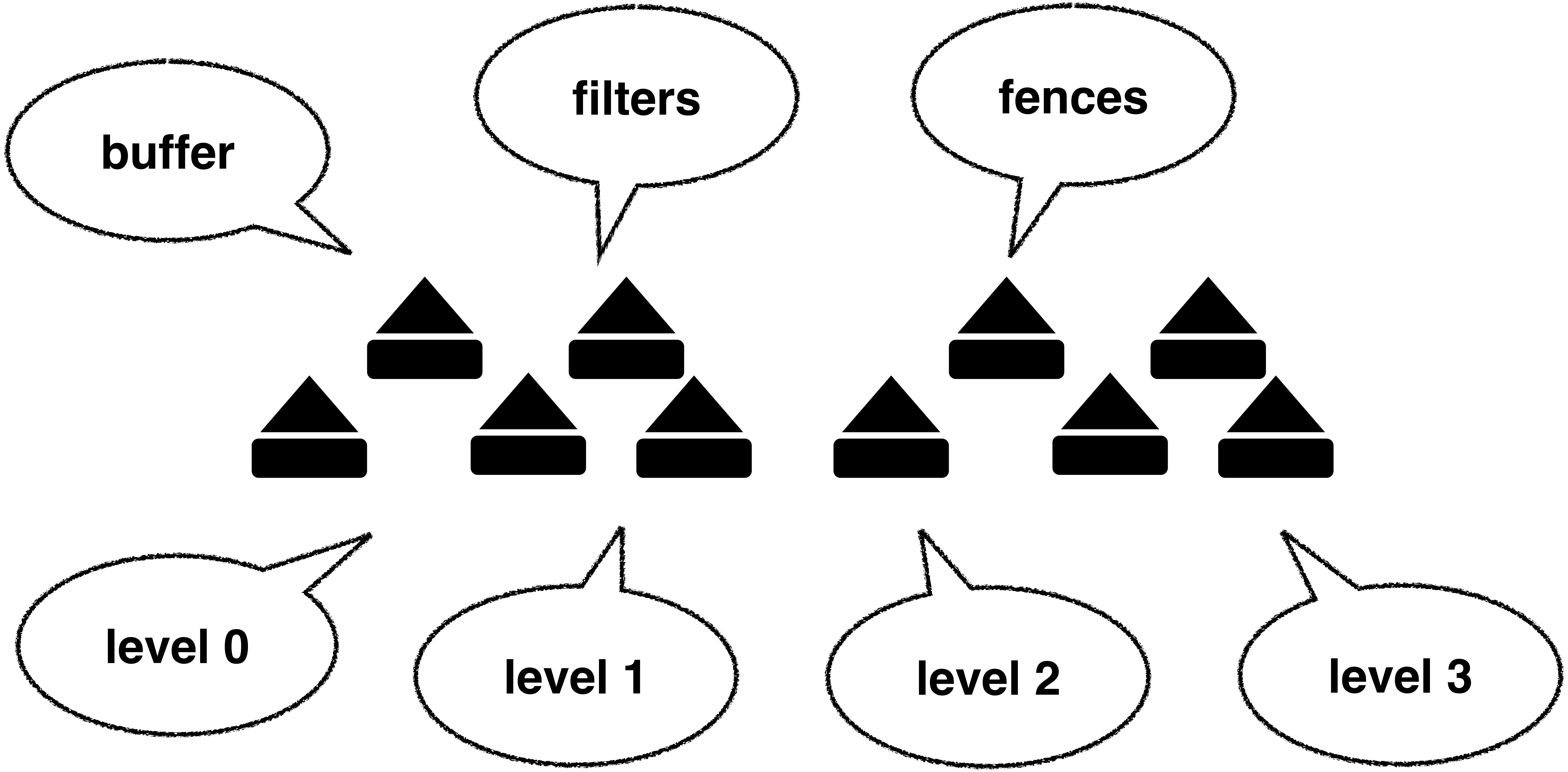
DASlab
@ Harvard SEAS

NoSQL systems are the backbone of the BigData and AI era

LSM-tree
KV-stores

FACEBOOK, AMAZON, GOOGLE, TWITTER, LINKEDIN

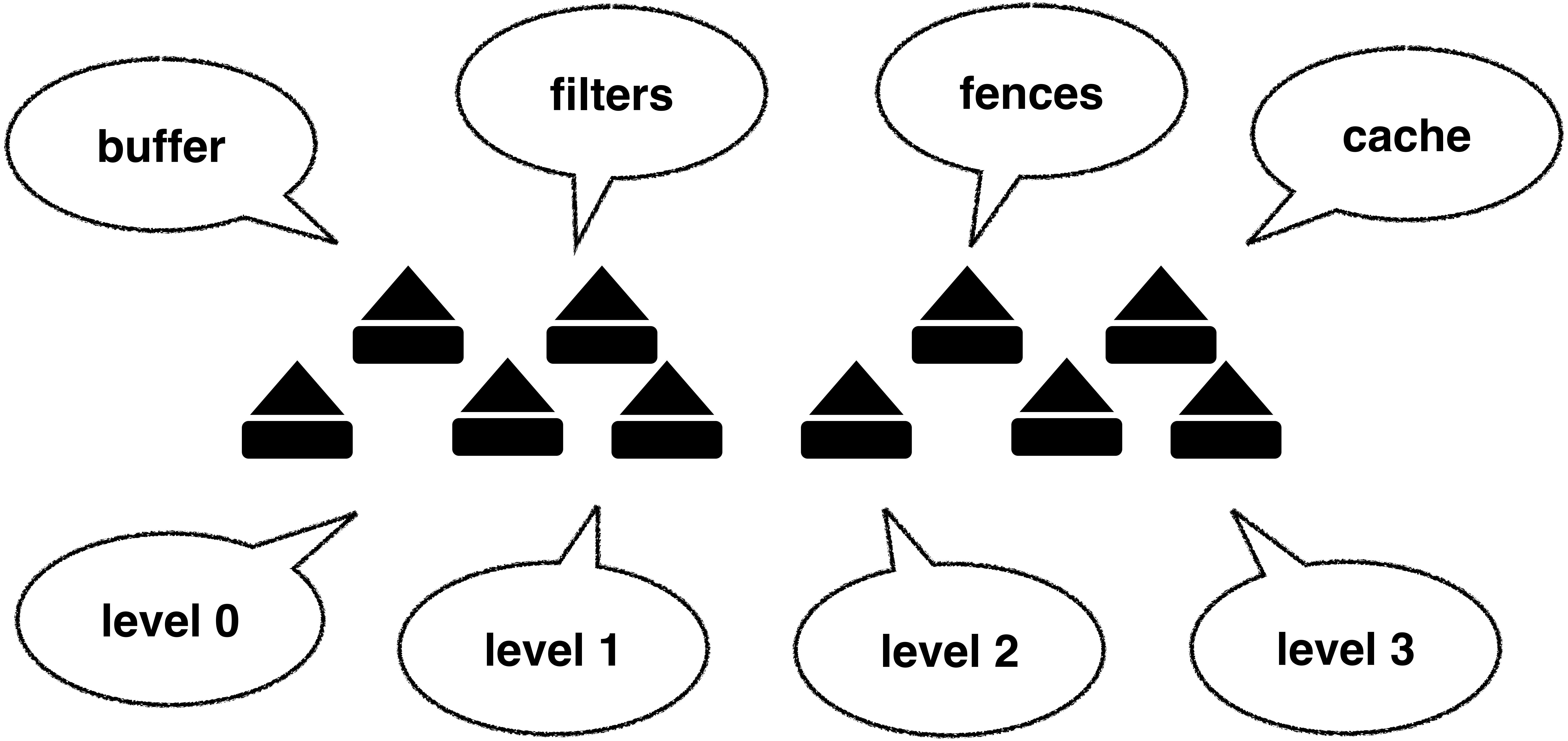
MACHINE LEARNING, SQL, CRYPTO, SCIENCE



NoSQL systems are the backbone of the BigData and AI era

LSM-tree
KV-stores

FACEBOOK, AMAZON, GOOGLE, TWITTER, LINKEDIN
MACHINE LEARNING, SQL, CRYPTO, SCIENCE



@ Harvard SEAS

NoSQL systems are the backbone of the BigData and AI era

LSM-tree
KV-stores

FACEBOOK, AMAZON, GOOGLE, TWITTER, LINKEDIN

MACHINE LEARNING, SQL, CRYPTO, SCIENCE

NoSQL systems are the backbone of the BigData and AI era

LSM-tree
KV-stores

FACEBOOK, AMAZON, GOOGLE, TWITTER, LINKEDIN
MACHINE LEARNING, SQL, CRYPTO, SCIENCE

NoSQL systems are the backbone of the BigData and AI era

LSM-tree
KV-stores

FACEBOOK, AMAZON, GOOGLE, TWITTER, LINKEDIN

MACHINE LEARNING, SQL, CRYPTO, SCIENCE

DASlab @ Harvard SEAS

NoSQL systems are the backbone of the BigData and AI era

LSM-tree
KV-stores

FACEBOOK, AMAZON, GOOGLE, TWITTER, LINKEDIN

MACHINE LEARNING, SQL, CRYPTO, SCIENCE

NoSQL systems are the backbone of the BigData and AI era

LSM-tree
KV-stores

FACEBOOK, AMAZON, GOOGLE, TWITTER, LINKEDIN

MACHINE LEARNING, SQL, CRYPTO, SCIENCE

NoSQL systems are the backbone of the BigData and AI era

LSM-tree
KV-stores

FACEBOOK, AMAZON, GOOGLE, TWITTER, LINKEDIN
MACHINE LEARNING, SQL, CRYPTO, SCIENCE

buffer **filters** **fences** **cache**

diverse data structures	interactions	hardware	parallelism	robustness cloud cost SLAs

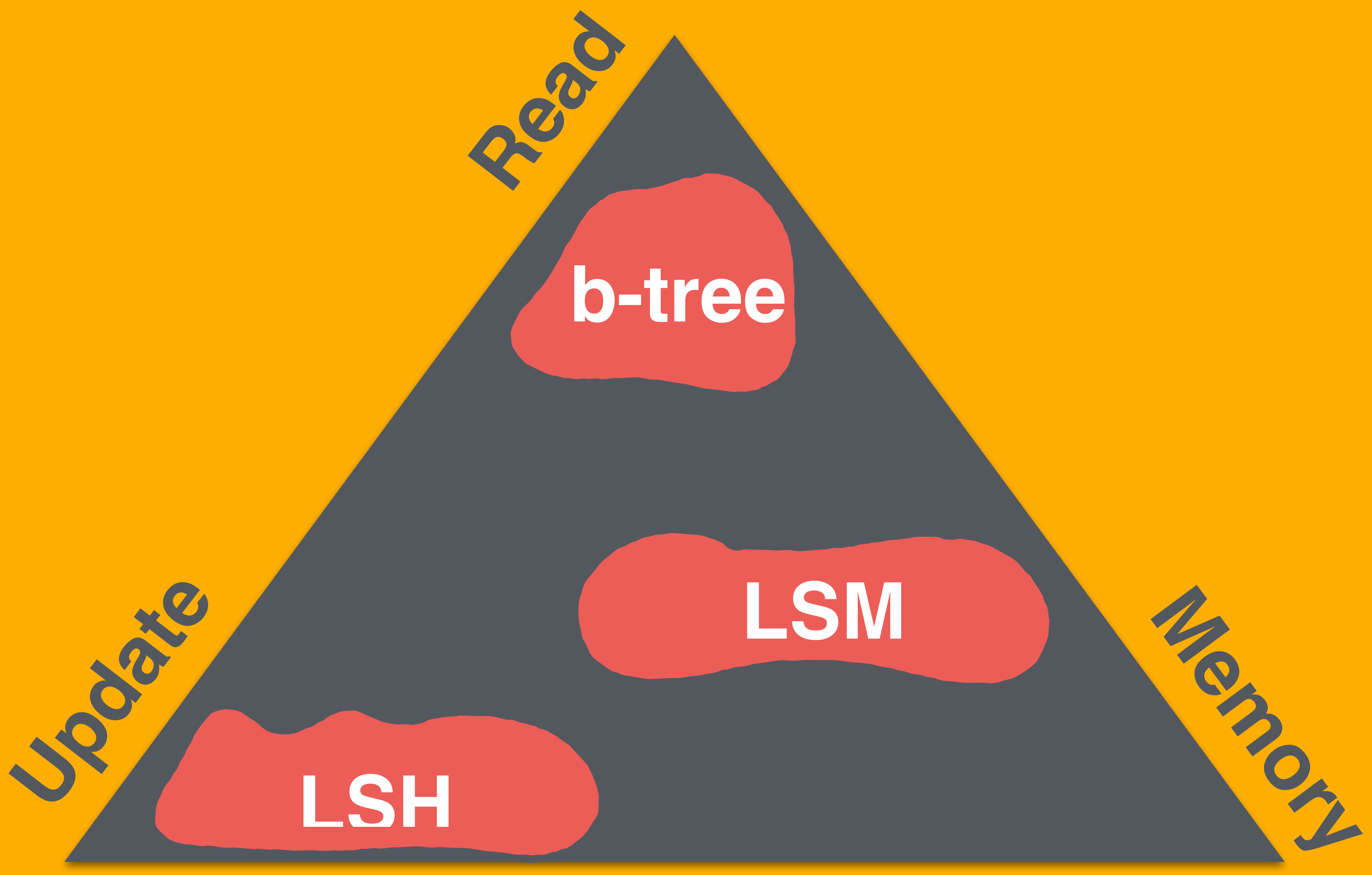
There exist numerous variations of NoSQL KV-stores

LSM-tree variants, B-trees (MongoDB), Hash-index (Microsoft)

diverse data structures | interactions | hardware | parallelism | robustness cloud cost SLAs

There exist numerous variations of NoSQL KV-stores

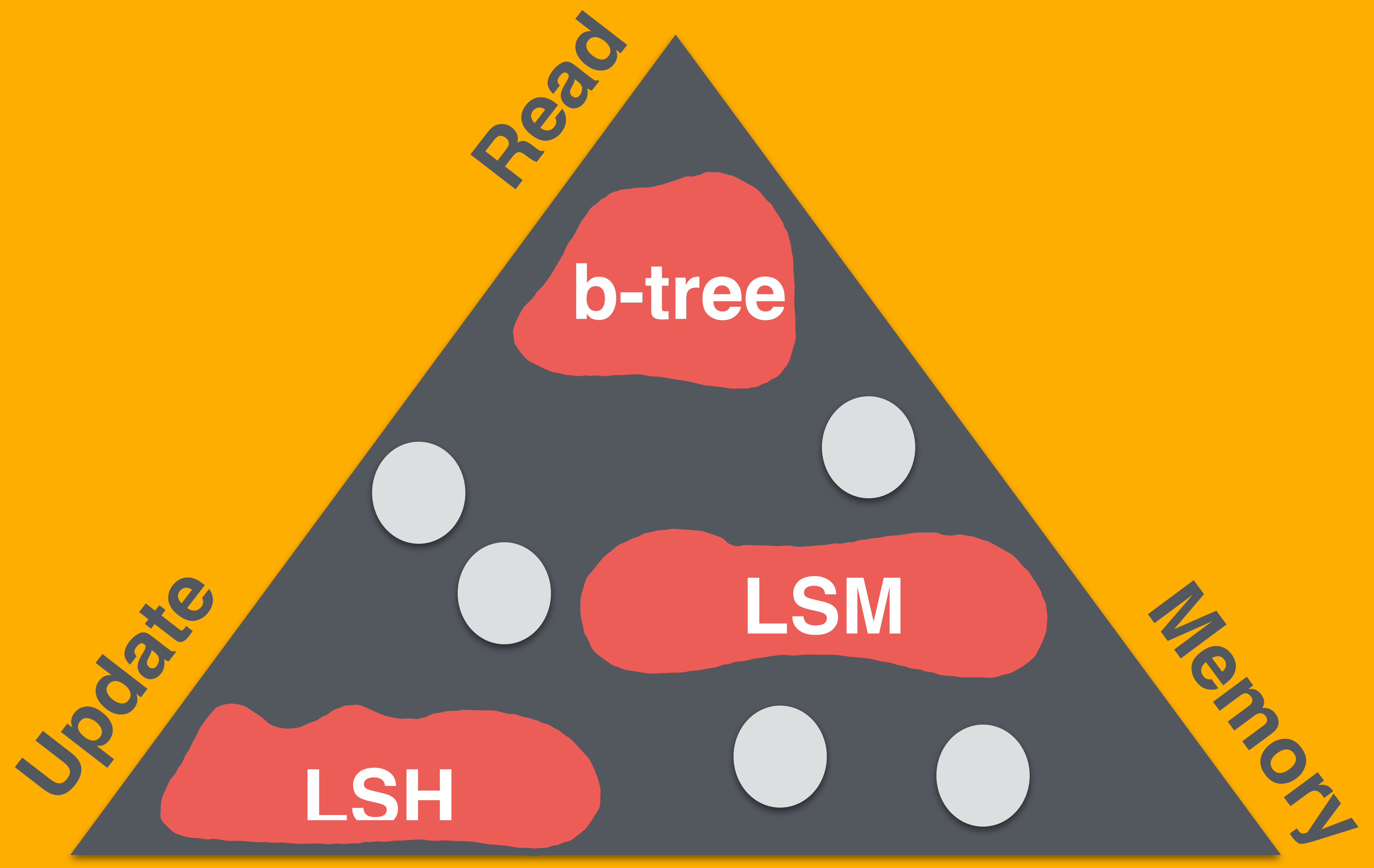
LSM-tree variants, B-trees (MongoDB), Hash-index (Microsoft)



diverse data structures | interactions | hardware | parallelism | robustness cloud cost SLAs

There exist numerous variations of NoSQL KV-stores

LSM-tree variants, B-trees (MongoDB), Hash-index (Microsoft)



Constant and increasing efforts for new system designs as applications & hardware change

diverse data structures	interactions	hardware	parallelism	robustness cloud cost SLAs

There exist numerous variations of NoSQL KV-stores

LSM-tree variants, B-trees (MongoDB), Hash-index (Microsoft)

diverse
data structures

interactions

hardware

parallelism

robustness
cloud cost
SLAs

Requirements/Goals

diverse
data structures

interactions

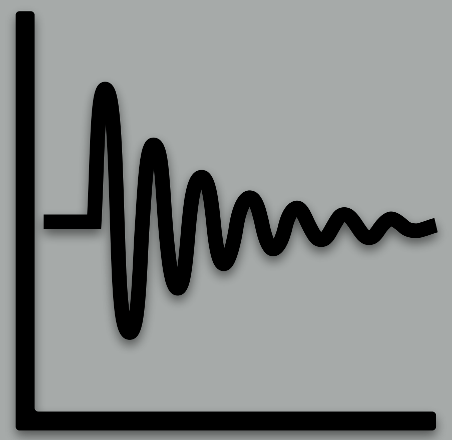
hardware

parallelism

robustness
cloud cost
SLAs

Requirements/Goals

data & queries

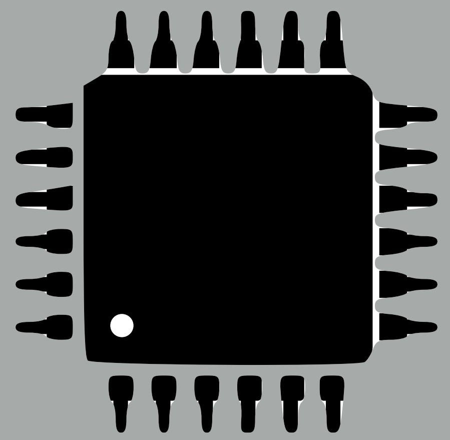


performance

budget

$$$

Context



SLA

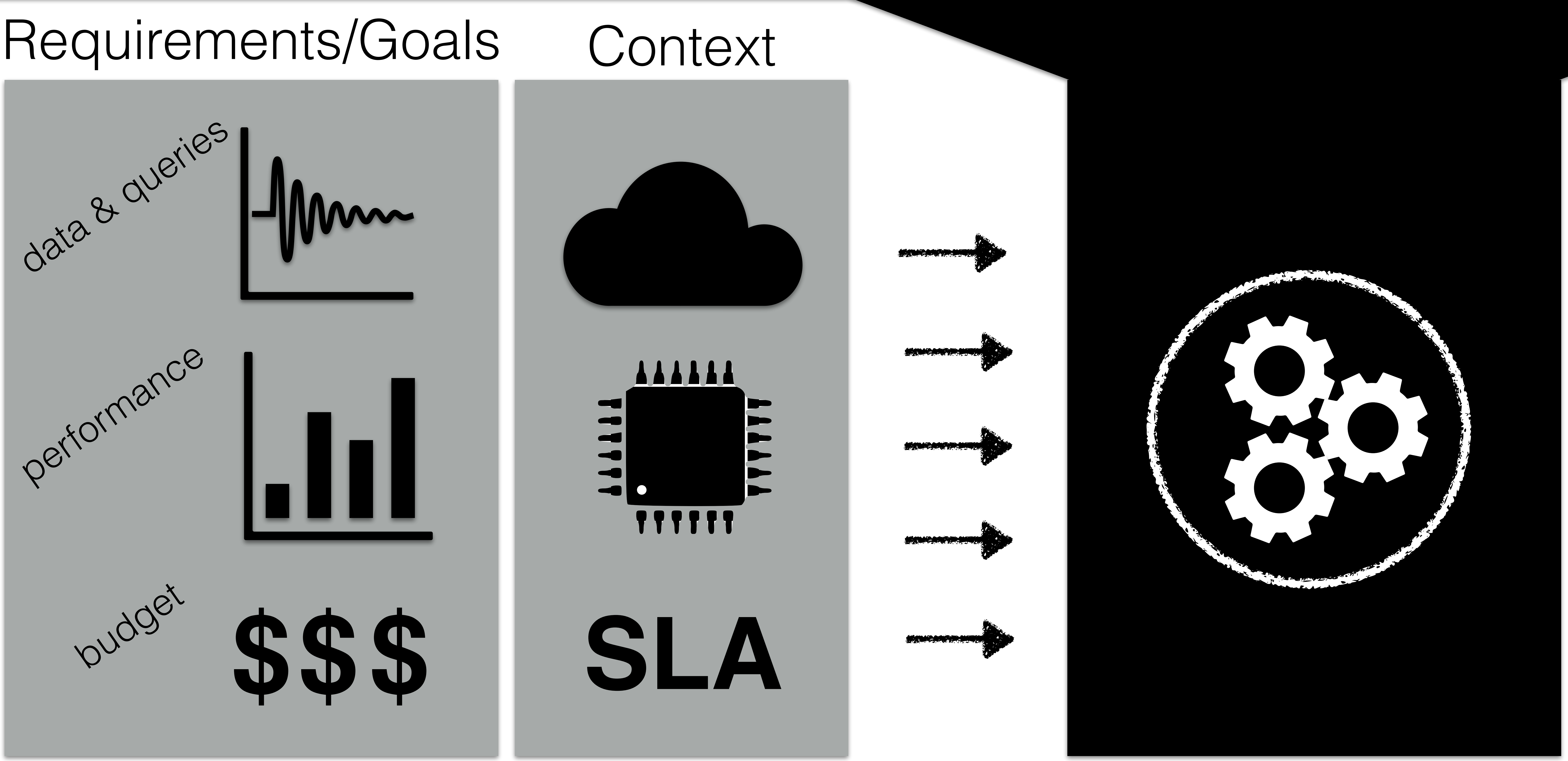
diverse data structures

interactions

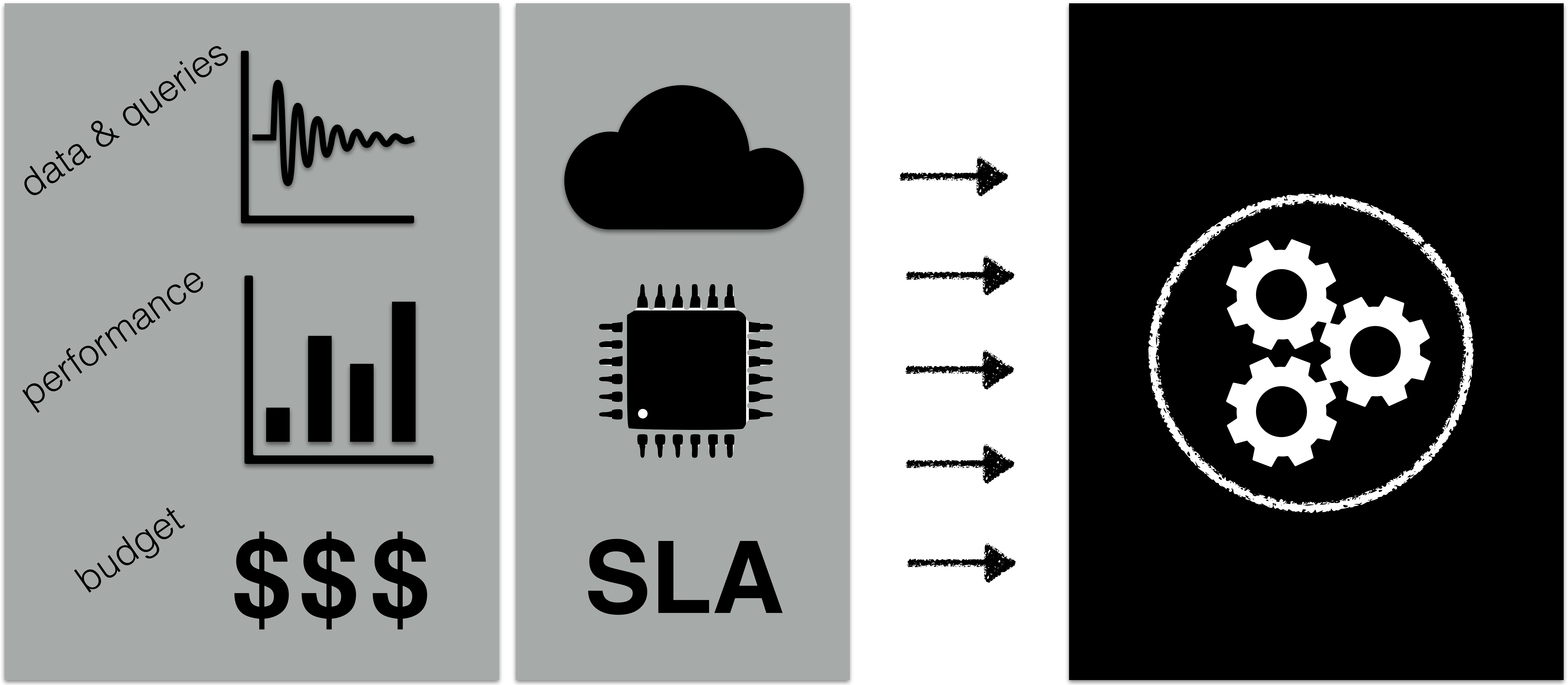
hardware

parallelism

robustness
cloud cost
SLAs

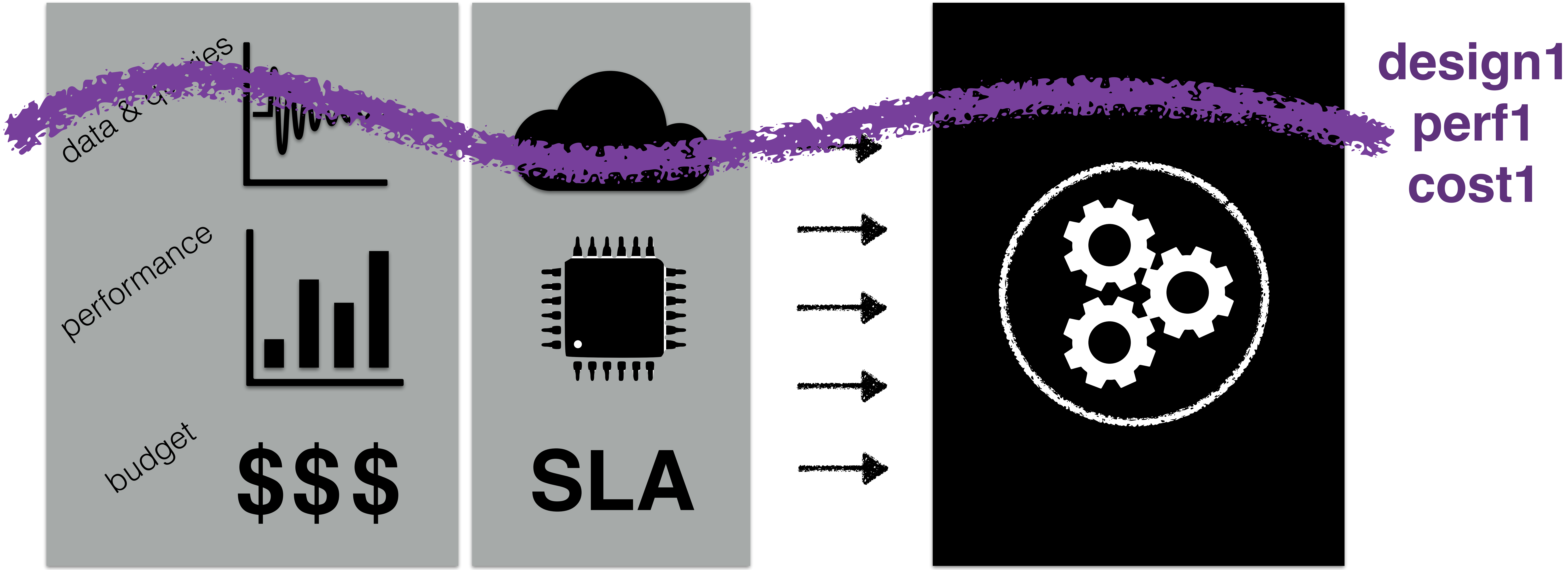



diverse
data structures
interactions
hardware
parallelism
robustness
cloud cost
SLAs
Requirements/Goals
Context
data & queries
performance
budget
$$$
SLA
best
DATA
SYSTEM
design & code



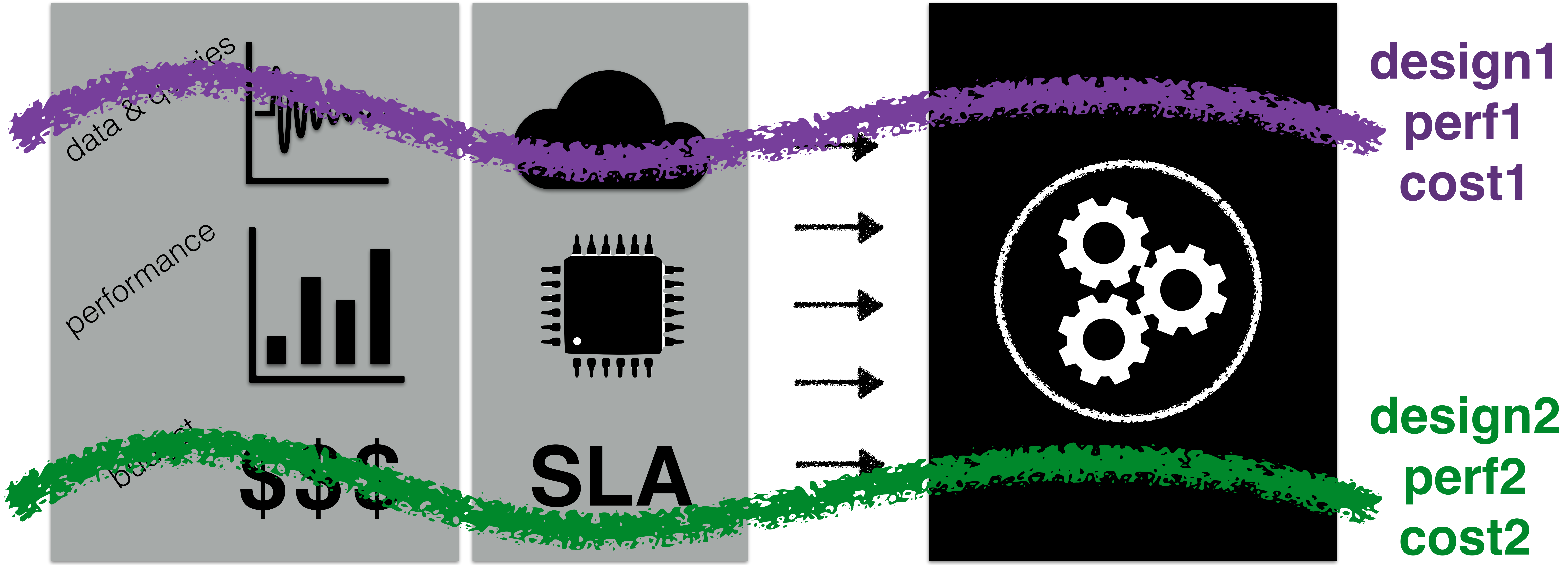
what-if reasoning

DASlab
@ Harvard SEAS



what-if reasoning

DASlab
@ Harvard SEAS



what-if reasoning

DASlab
@ Harvard SEAS

AUTO DESIGN

Rob Tarjan, Turing Award 1986

"IS THERE A CALCULUS OF DATA STRUCTURES

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

[*P* vs *NP, average case, constant factors vs asymptotic, low bounds*]

DASlab
@ Harvard SEAS

IS THERE A CALCULUS OF DATA SYSTEMS?

Rob Tarjan, Turing Award 1986

"**IS THERE A CALCULUS OF DATA STRUCTURES** by which one can choose the appropriate representation and techniques for a given problem?" (SIAM,1978)

[*P* vs *NP, average case, constant factors vs asymptotic, low bounds*] the **grammar** of data systems design

the **grammar** of data systems design

Nikos Kazantzakis, philosopher

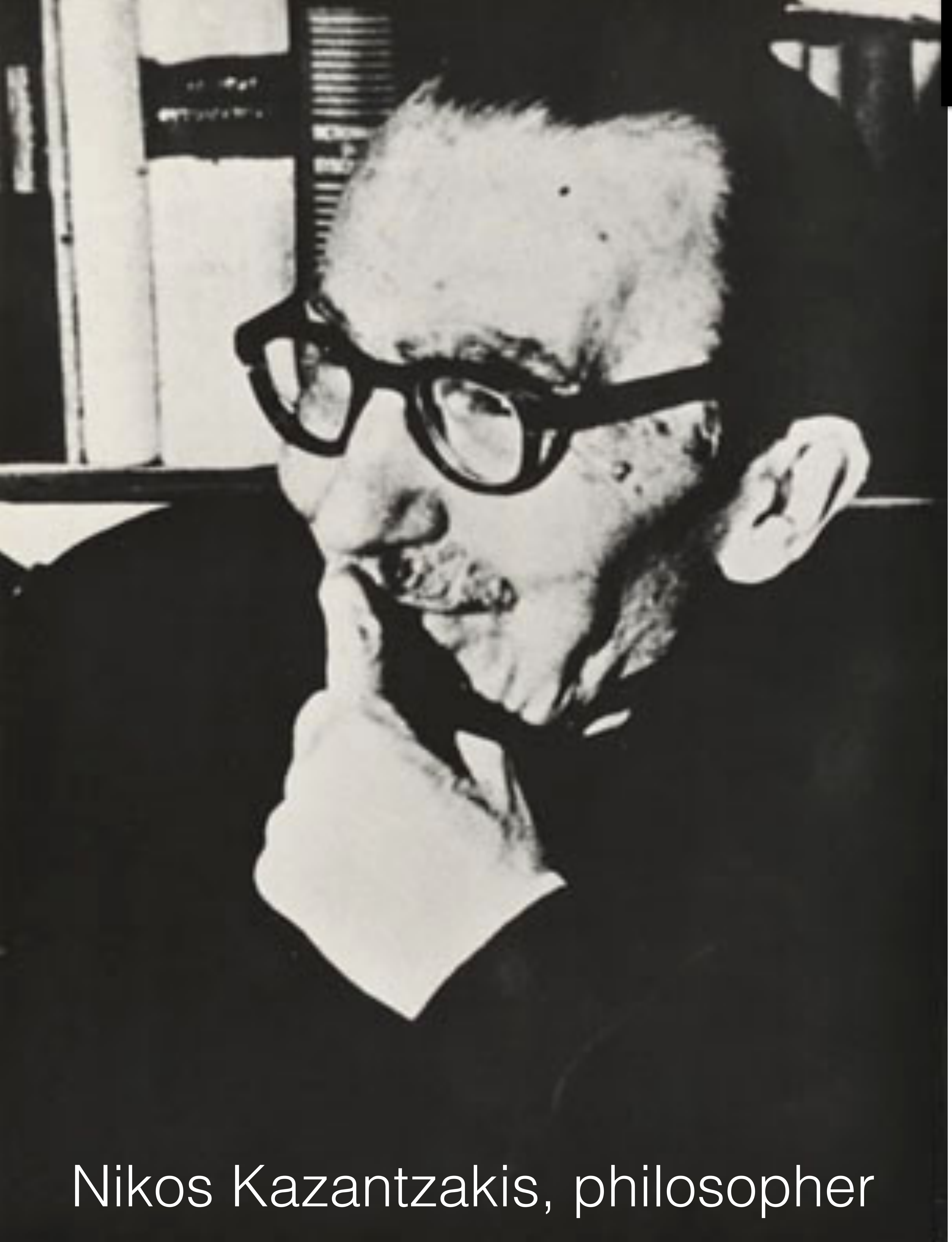
action *is* *for* *nothing* *holy*

hope *the* *most* *of*

fear *form*

am *free*

ultimate *I* *theory*

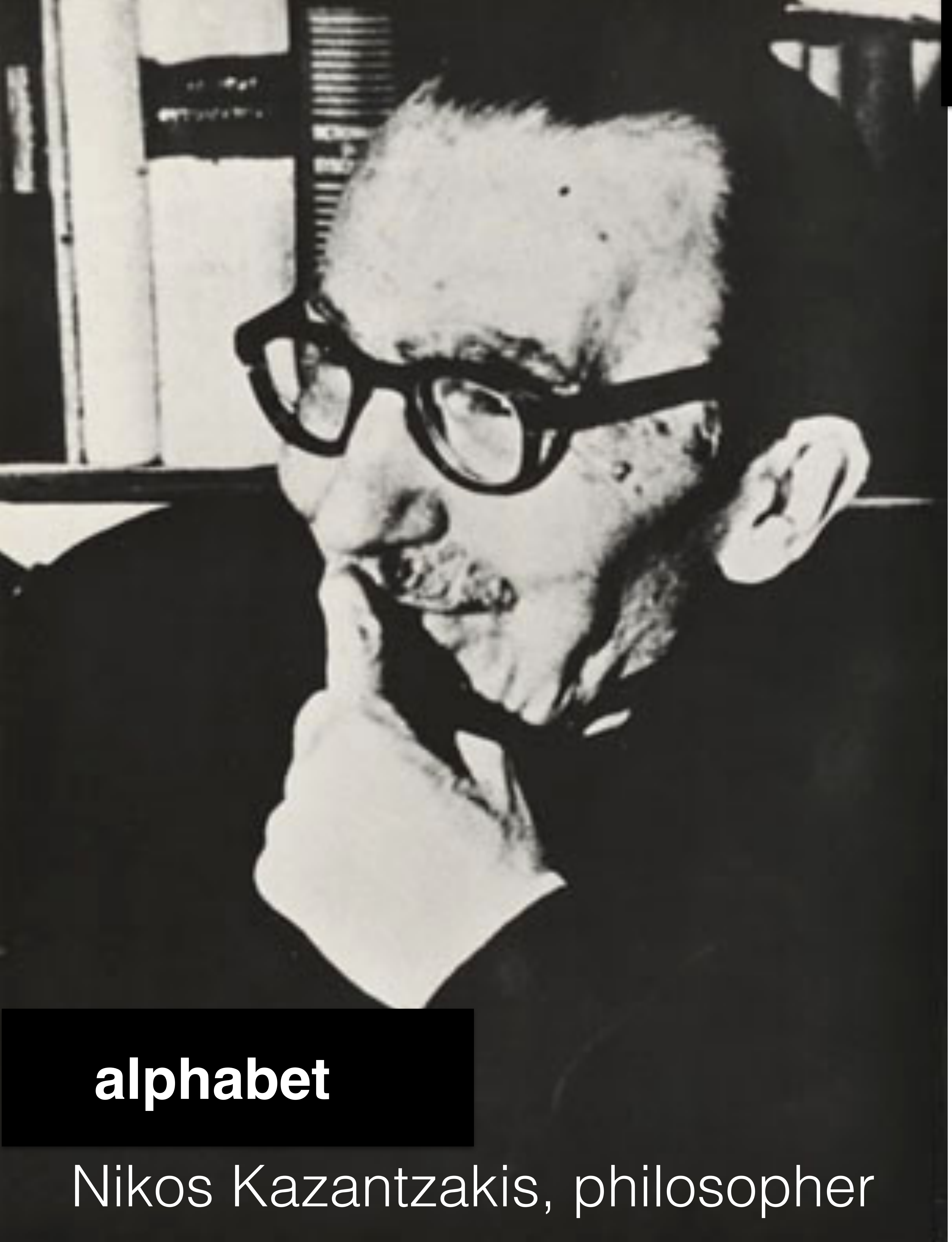


Nikos Kazantzakis, philosopher

action is the ultimate form of theory

the most holy form of theory

I hope for nothing
I fear nothing
I am free



Nikos Kazantzakis, philosopher

action is

the most holy

of

form

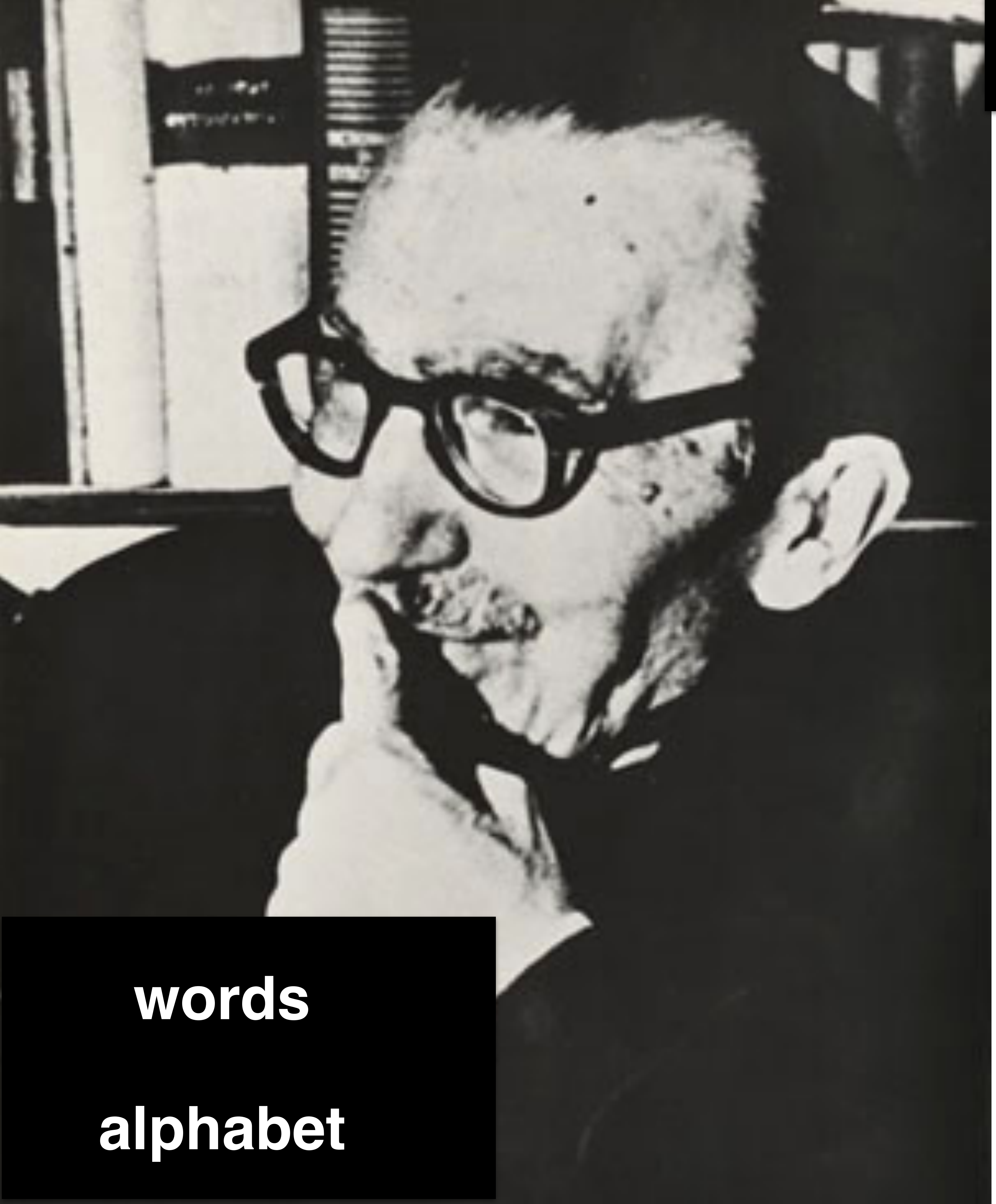
ultimate theory

I hope for nothing

I fear nothing

I am free

the **grammar** of data systems design



Nikos Kazantzakis, philosopher

action is

the most holy

of

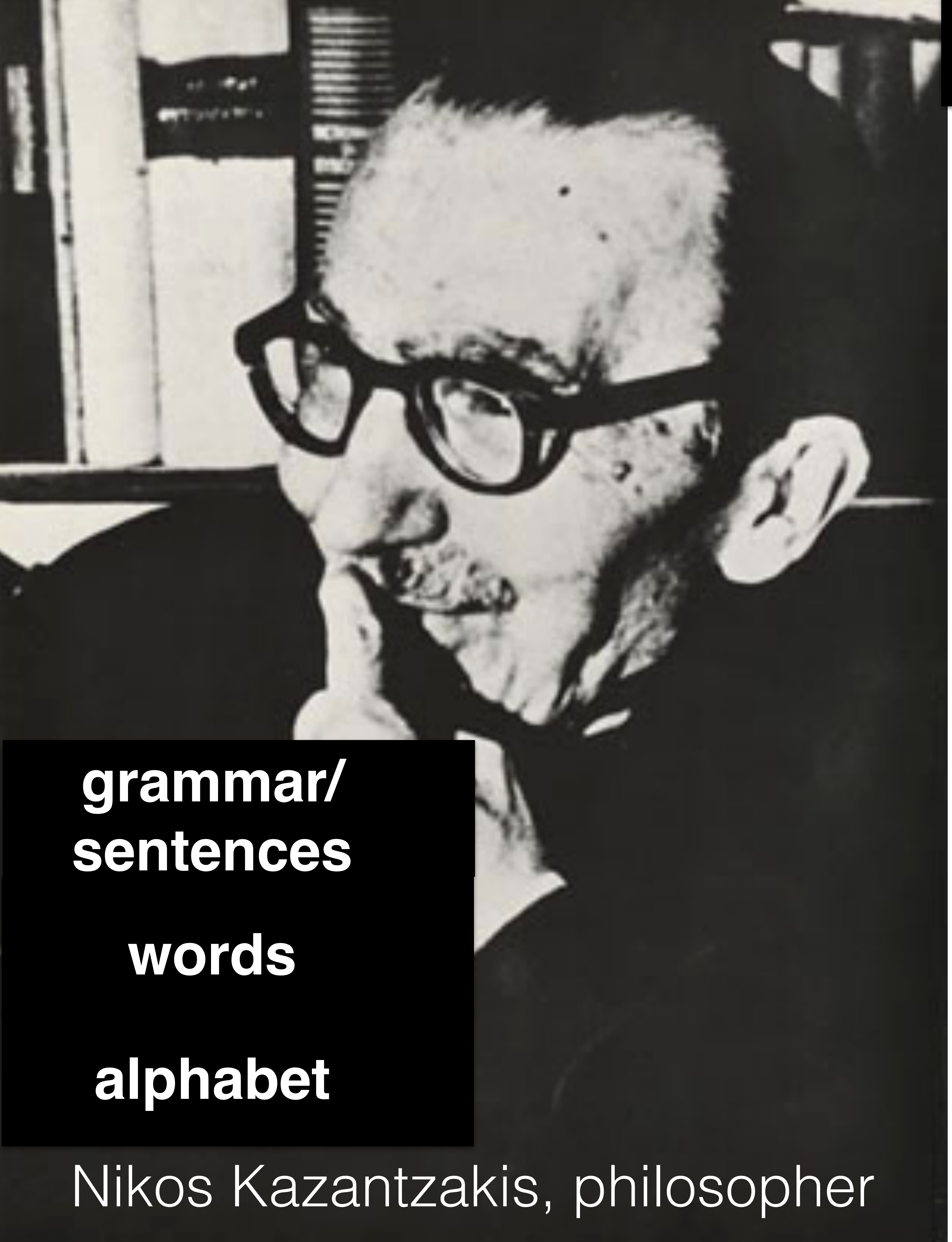
form

ultimate theory

I hope for nothing

I fear nothing

I am free



Nikos Kazantzakis, philosopher

action is

the most holy of

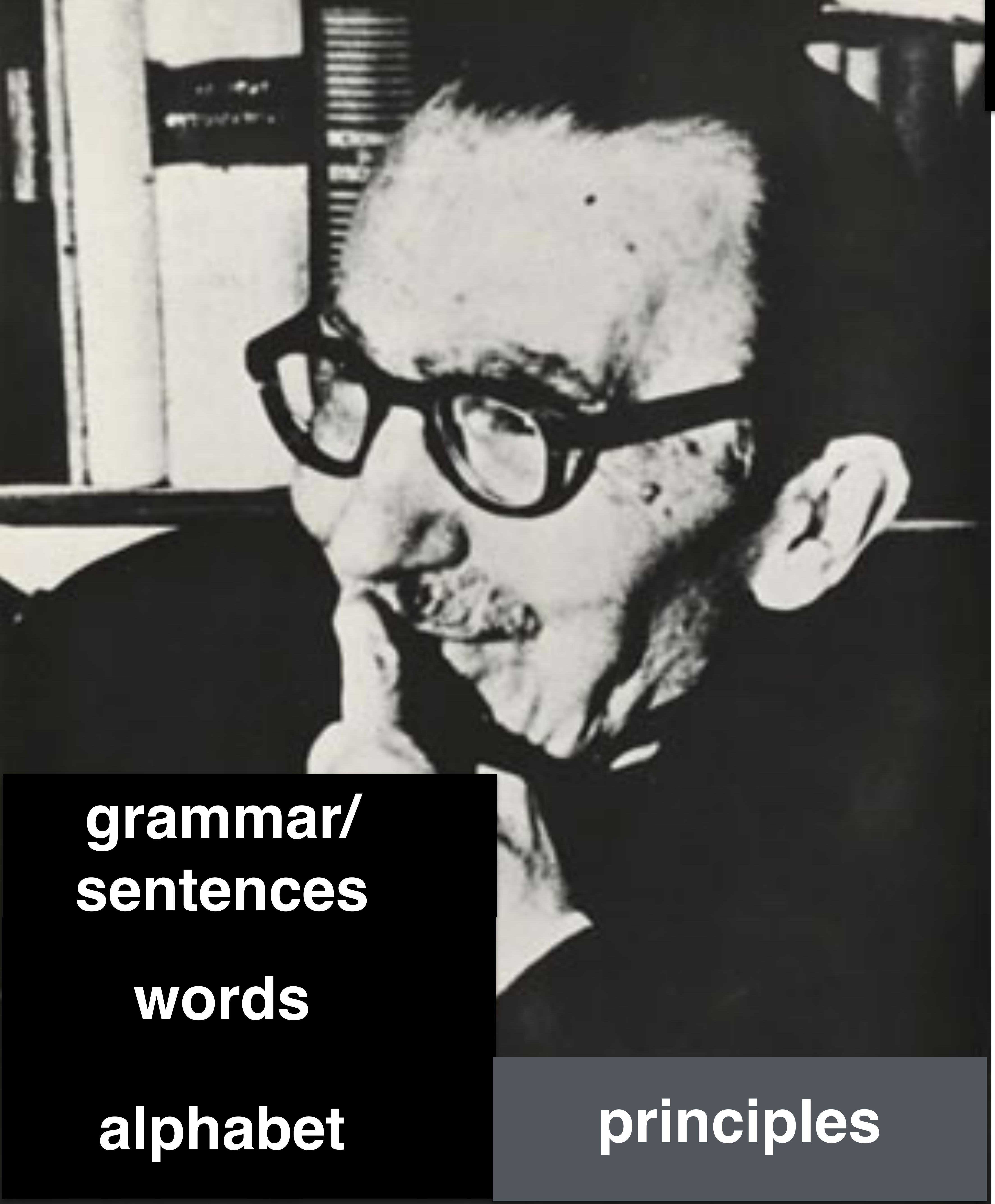
form

ultimate theory

I hope for nothing

I fear nothing

I am free

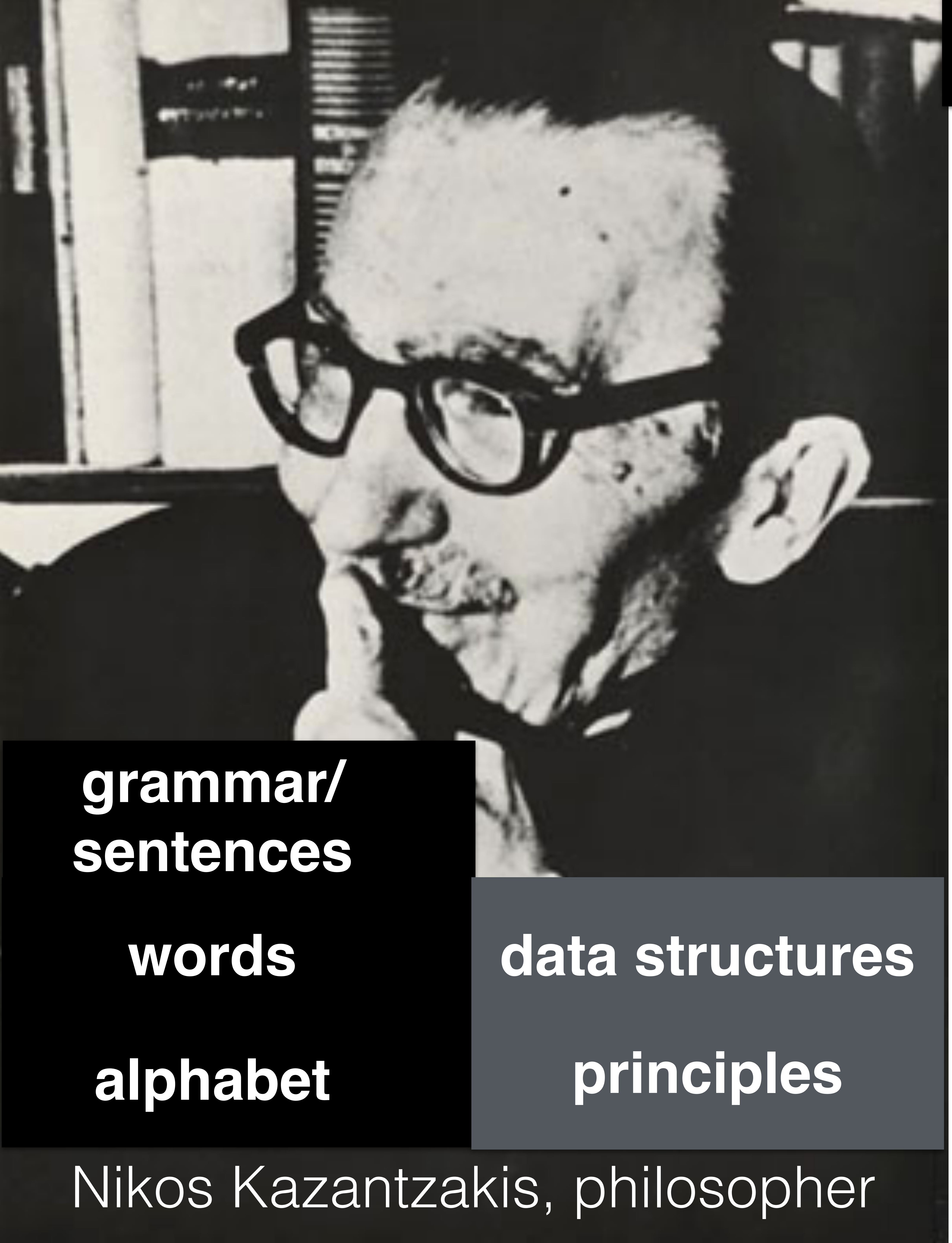


Nikos Kazantzakis, philosopher

action is
the most holy
form of
ultimate theory

I hope for nothing
I fear nothing
I am free

the **grammar** of data systems design



grammar/ sentences	
words	**data structures**
alphabet	**principles**

Nikos Kazantzakis, philosopher

action is the most holy form of theory

action is the ultimate form of theory

I hope for nothing
I fear nothing
I am free

the **grammar** of data systems design

grammar/ sentences	**interactions**
words	**data structures**
alphabet	**principles**

Nikos Kazantzakis, philosopher

action is
the most holy
of
form
ultimate theory

I hope for nothing
I fear nothing
I am free

the **grammar** of data systems design

grammar/ sentences	interactions
words	**data structures**
alphabet	**principles**

Nikos Kazantzakis, philosopher

action is
the most holy
of
form
ultimate theory

NEW

I hope for nothing
I fear nothing
I am free

action is

the most holy of

form

theory

which are "all"
possible *data systems*
we may ever invent?

grammar/ sentences	**interactions**
words	**data structures**
alphabet	**principles**

Nikos Kazantzakis, philosopher

I hope for nothing

I fear nothing

I am free

Trillions of possible data structures

Data Calculator @SIGMOD 2018

Trillions of possible data structures

Data Calculator @SIGMOD 2018

New NoSQL systems: 1000x faster

Cosine @PVLDB 2022 and Limousine @SIGMOD 2024

Trillions of possible data structures

Data Calculator @SIGMOD 2018

New NoSQL systems: 1000x faster

Cosine @PVLDB 2022 and Limousine @SIGMOD 2024

Synthesized statistics, 10x faster ML

Data Canopy @SIGMOD 2017

Trillions of possible data structures

Data Calculator @SIGMOD 2018

New NoSQL systems: 1000x faster

Cosine @PVLDB 2022 and Limousine @SIGMOD 2024

Synthesized statistics, 10x faster ML

Data Canopy @SIGMOD 2017

10x faster Neural Networks

MotherNets @MLSys 2020, and M2 @MLSys 2023

Trillions of possible data structures

Data Calculator @SIGMOD 2018

New NoSQL systems: 1000x faster

Cosine @PVLDB 2022 and Limousine @SIGMOD 2024

Synthesized statistics, 10x faster ML

Data Canopy @SIGMOD 2017

10x faster Neural Networks

MotherNets @MLSys 2020, and M2 @MLSys 2023

10x faster Image AI

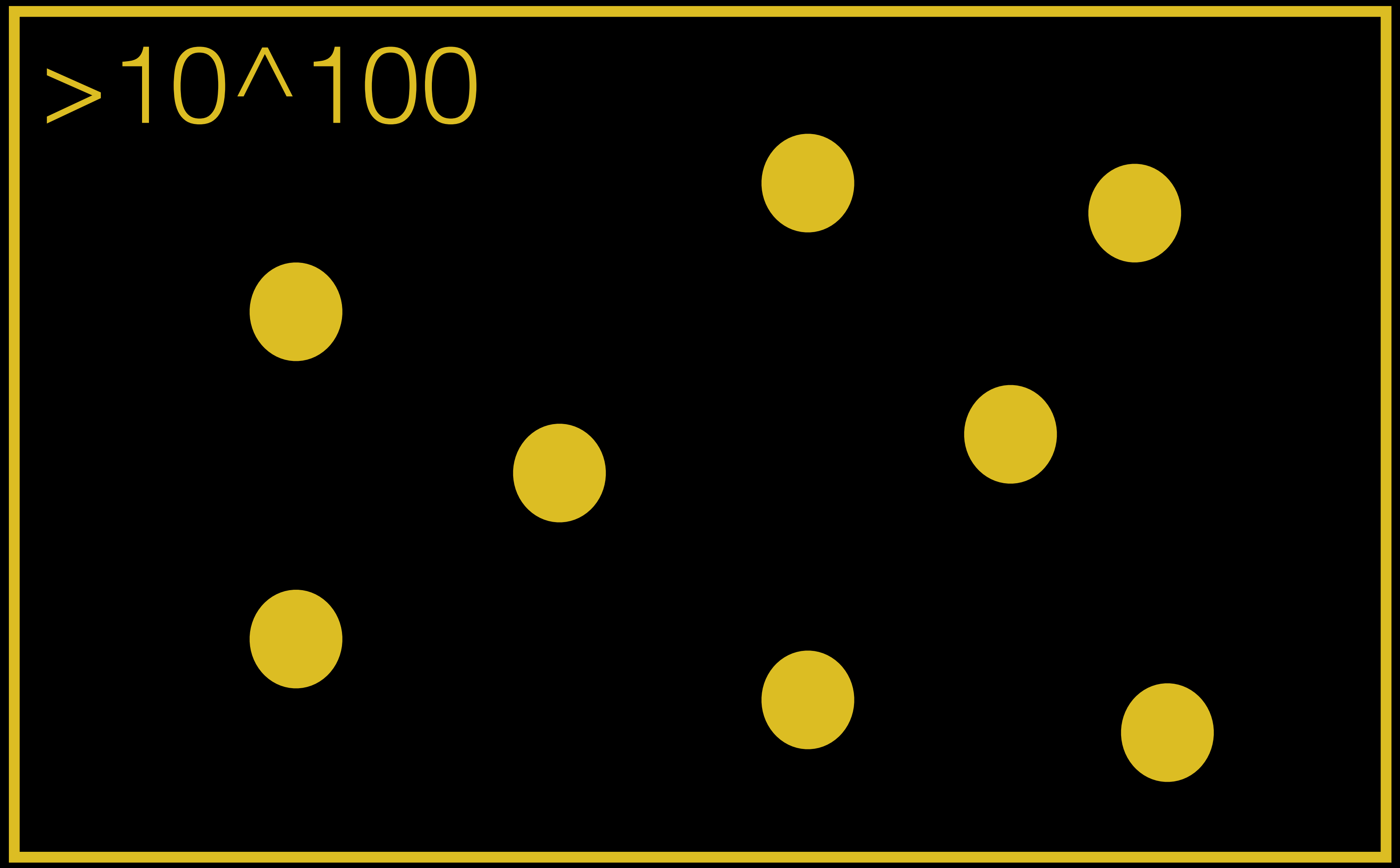
Image Calculator, SIGMOD 2024

1. DESIGN SPACE

data layout of data structures

algorithm design

systems: interactions of components

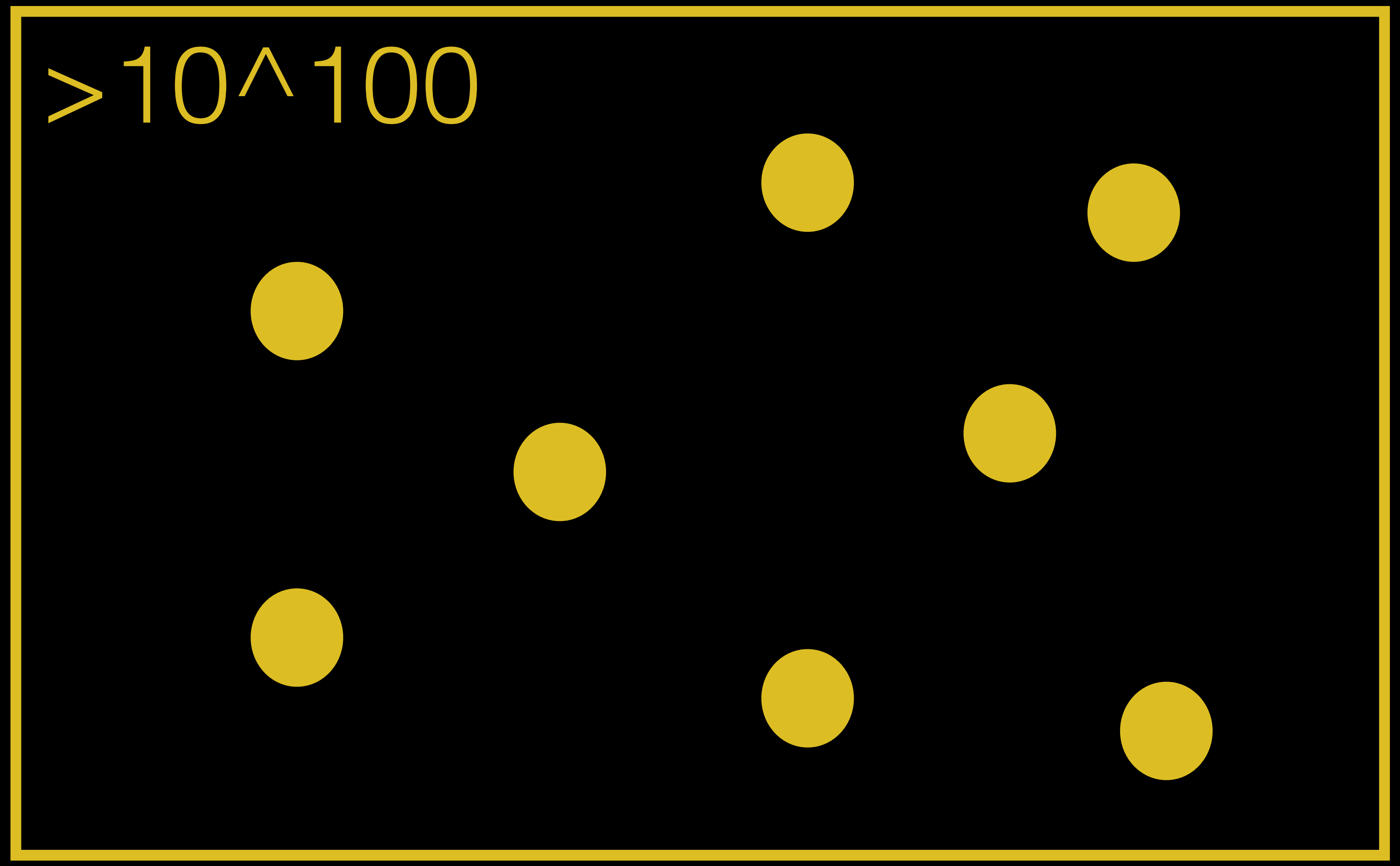


1. DESIGN SPACE

data layout of data structures

algorithm design

systems: interactions of components



2. NAVIGATE SEARCH SPACE

cost synthesis: computation and data movement

learned cost models in memory/parallelism

design continuums to shrink space

Design Primitives to Auto Generate Trillions of Data Structures

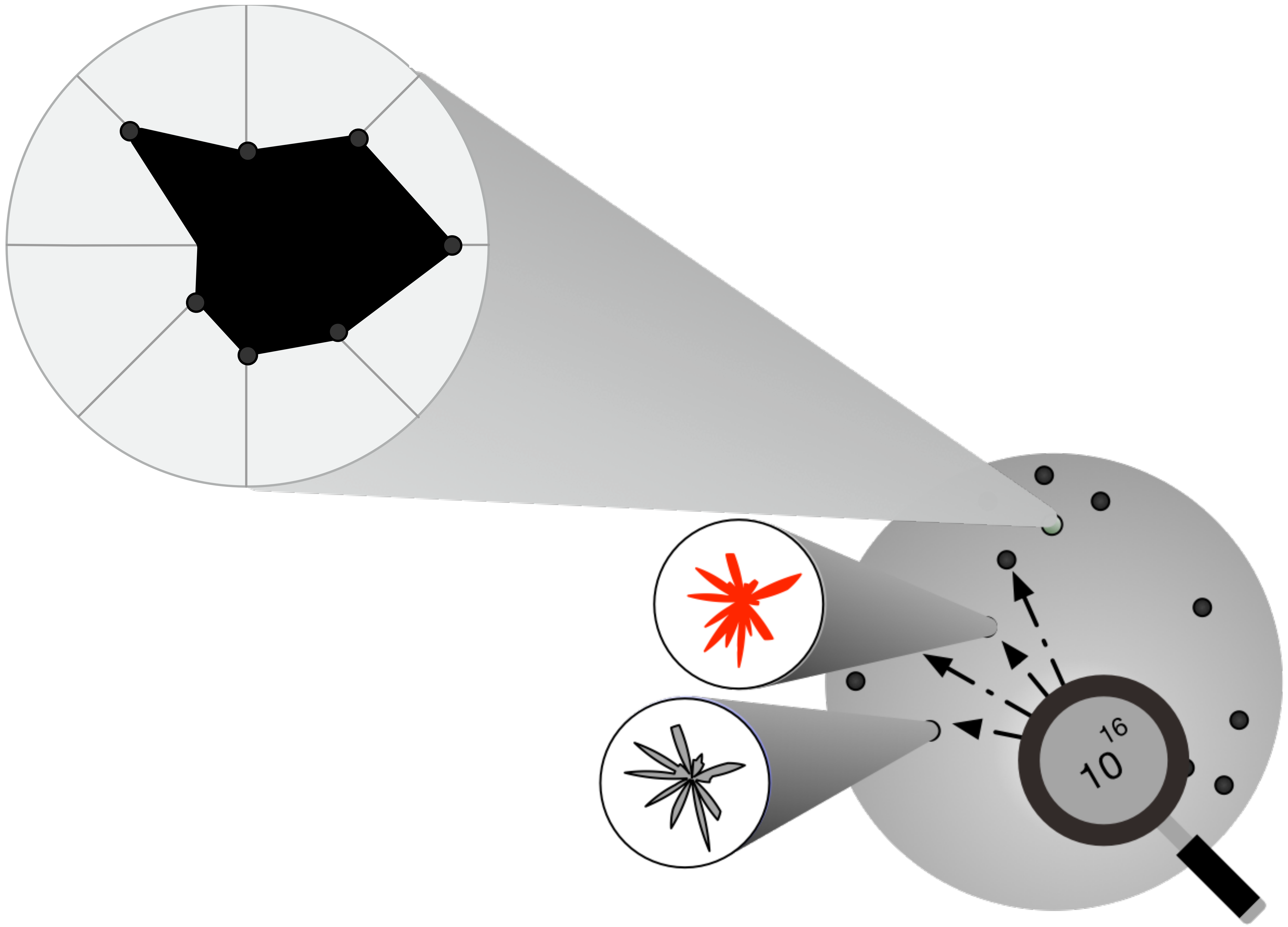
Unless otherwise specified, we use a reduced default values domain of 100 values for integers, 10 values for doubles, and 1 value for functions.

Categories		Primitive	Domain	size	Hash Table: H	Hash Table / LPL: LL	Hash Table / LPL: UDP	B+Tree/CSB+Tree/FAST: B+	CSB+	FAST	ODP
Node organization	1	**Key retention.** <u>No:</u> node contains no real key data, e.g., intermediate nodes of b+trees and linked lists. <u>*Yes:*</u> contains complete key data, e.g., nodes of b-trees, and arrays. <u>*Function:*</u> 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.** <u>*No:*</u> node contains no real value data, e.g., intermediate nodes of b+trees, and linked lists. <u>*Yes:*</u> contains complete value data, e.g., nodes of b-trees, and arrays. <u>*Function:*</u> 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. <u>Rules:</u> 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. <u>Rules:</u> 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									

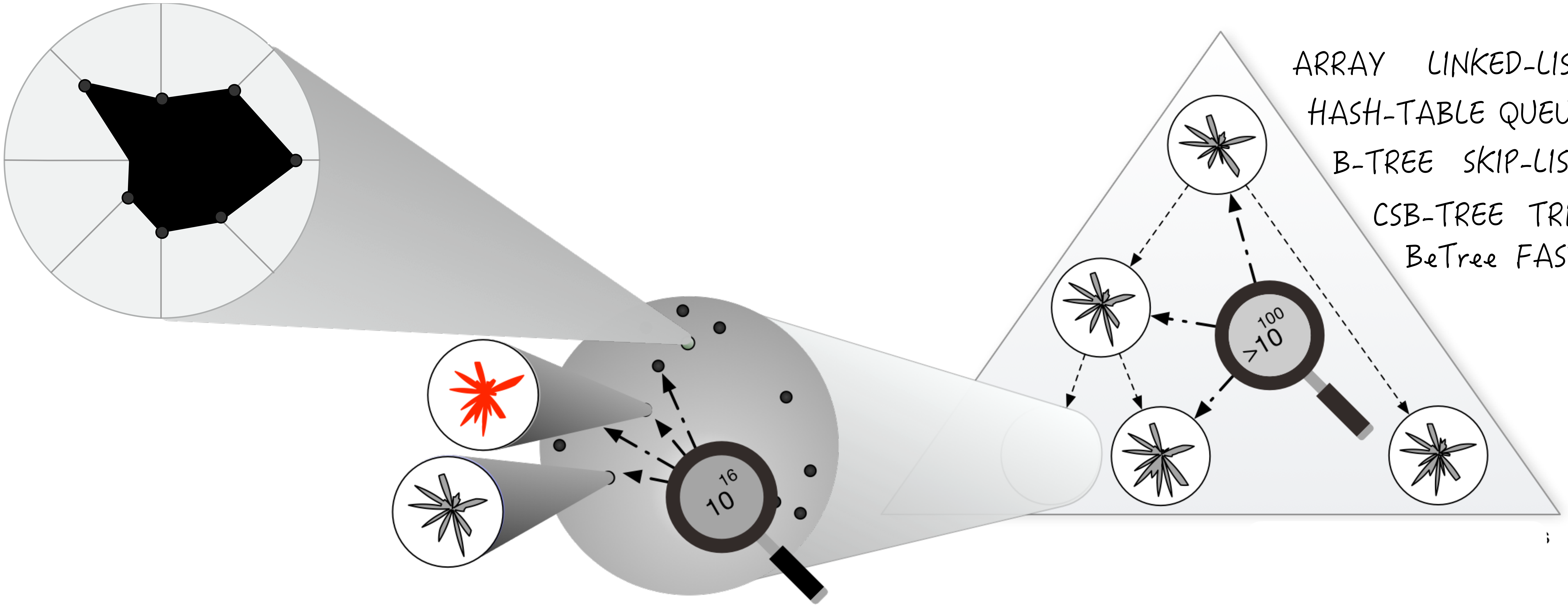
Partiti[…]		or balanced (i.e., all sub-blocks have the same size), unrestricted or functional. Rules: requires key partitioning != none.	stricted | function(func) *(up to 10 different fixed capacity values are considered)*	13	unrestri	fixed(25		balance	balance	balance	
	13	**Immediate node links.** Whether 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)	

Categories

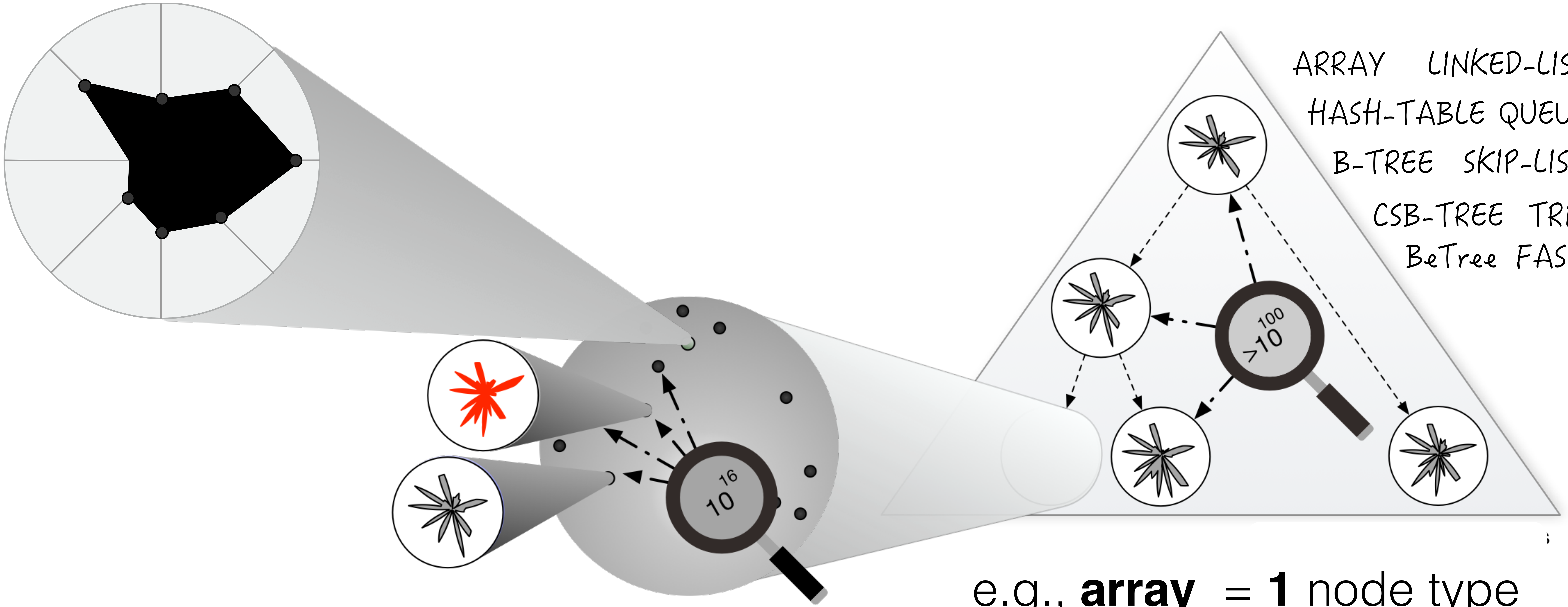
DESIGN PRINCIPLES   POSSIBLE NODE DESIGNS



DESIGN PRINCIPLES POSSIBLE NODE DESIGNS POSSIBLE STRUCTURES

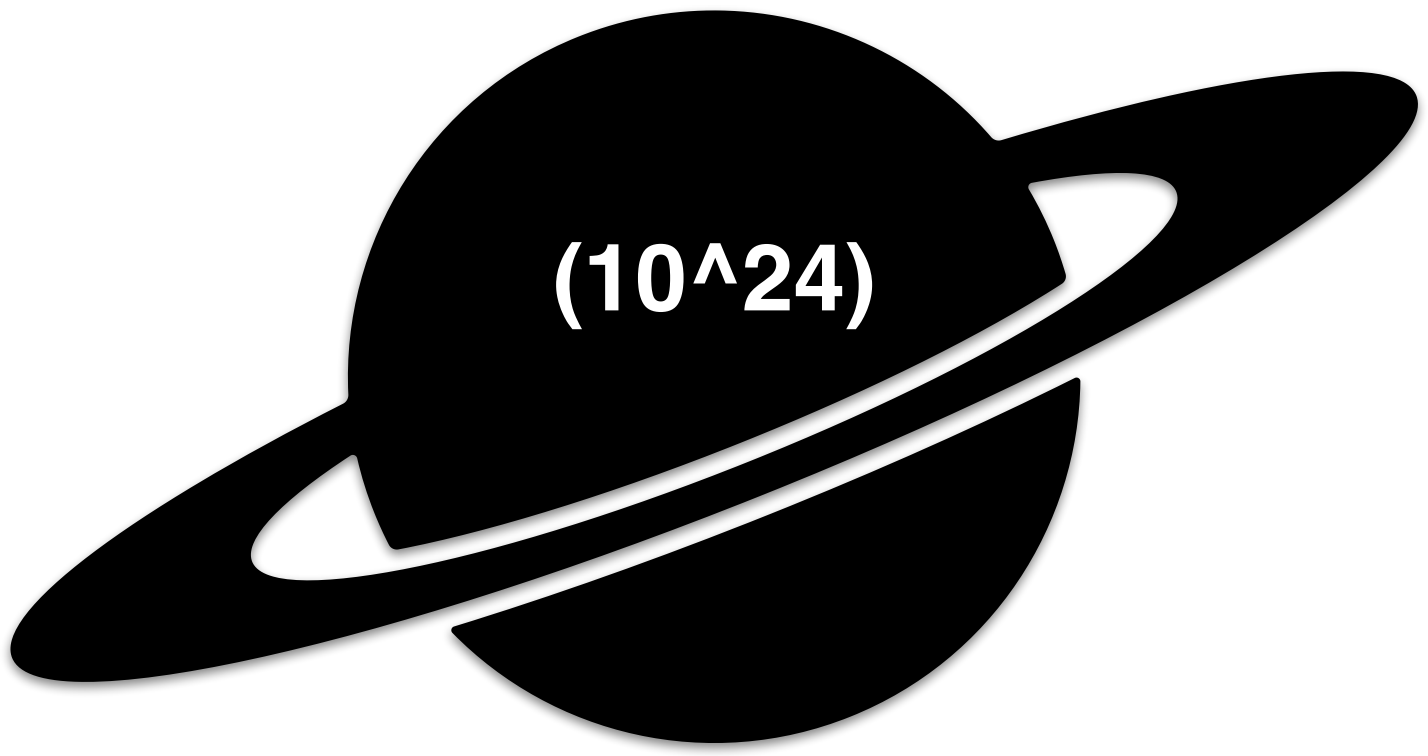


DESIGN PRINCIPLES POSSIBLE NODE DESIGNS POSSIBLE STRUCTURES

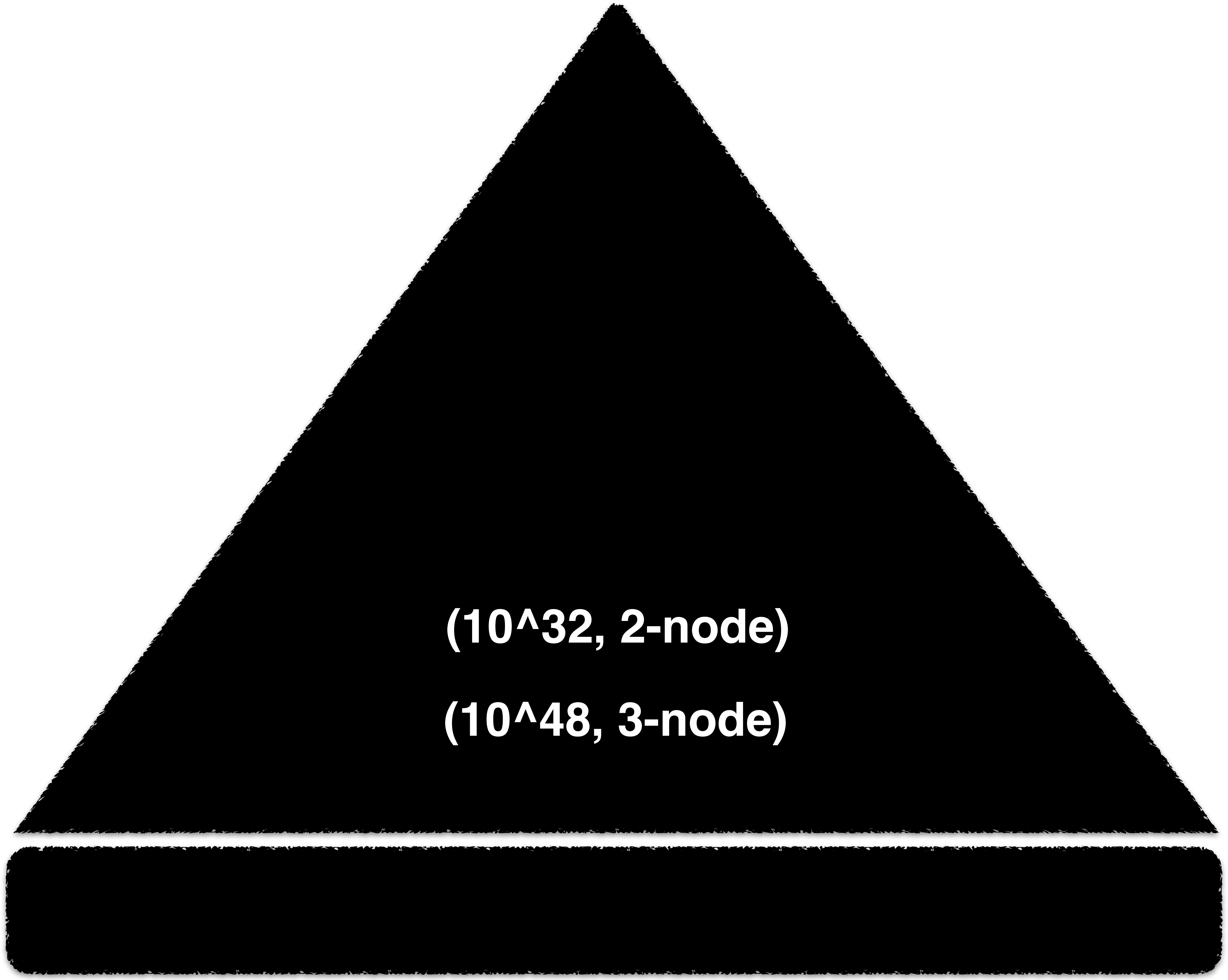


e.g., **array** = **1** node type

e.g., **b-tree** = **2** node types

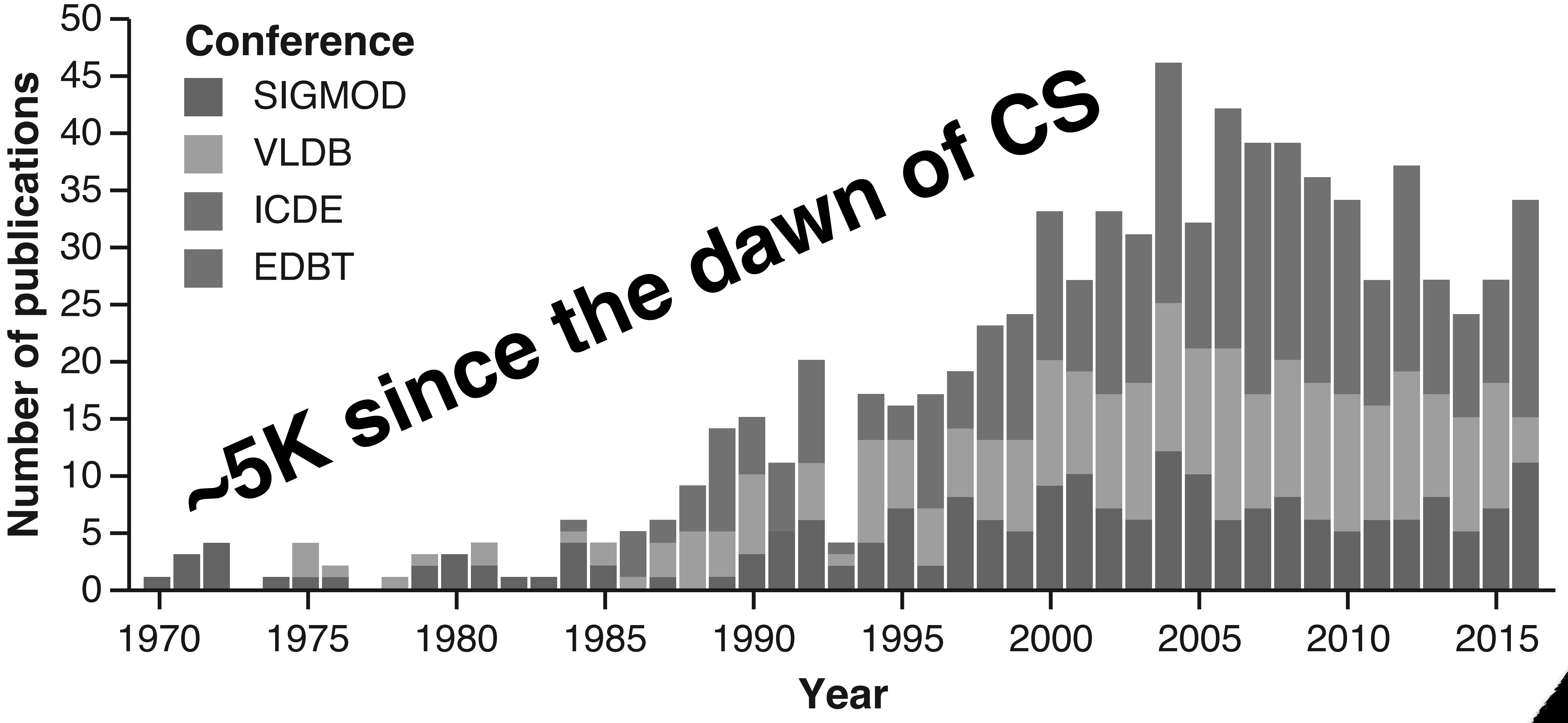


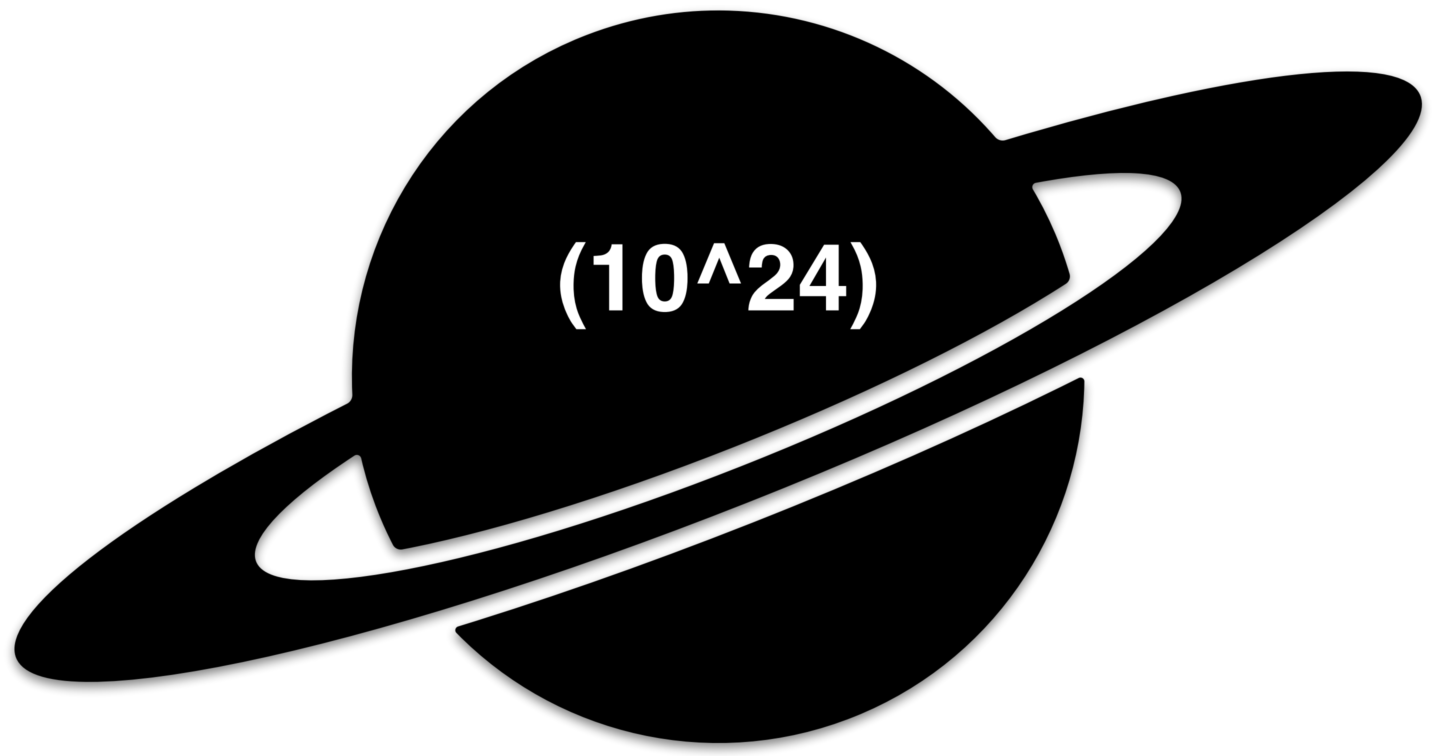
STARS ON THE SKY

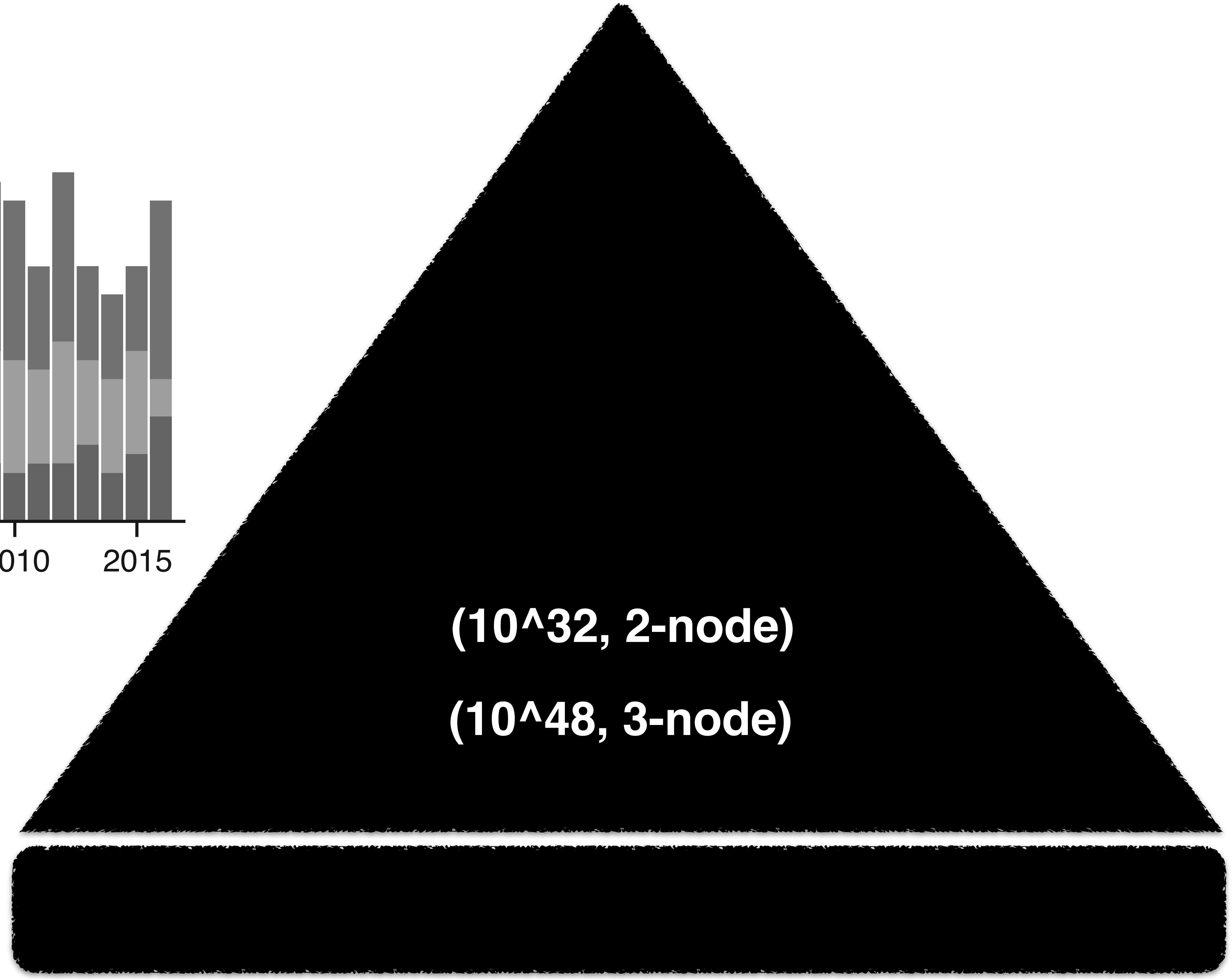


POSSIBLE DATA STRUCTURES

DASlab
@ Harvard SEAS

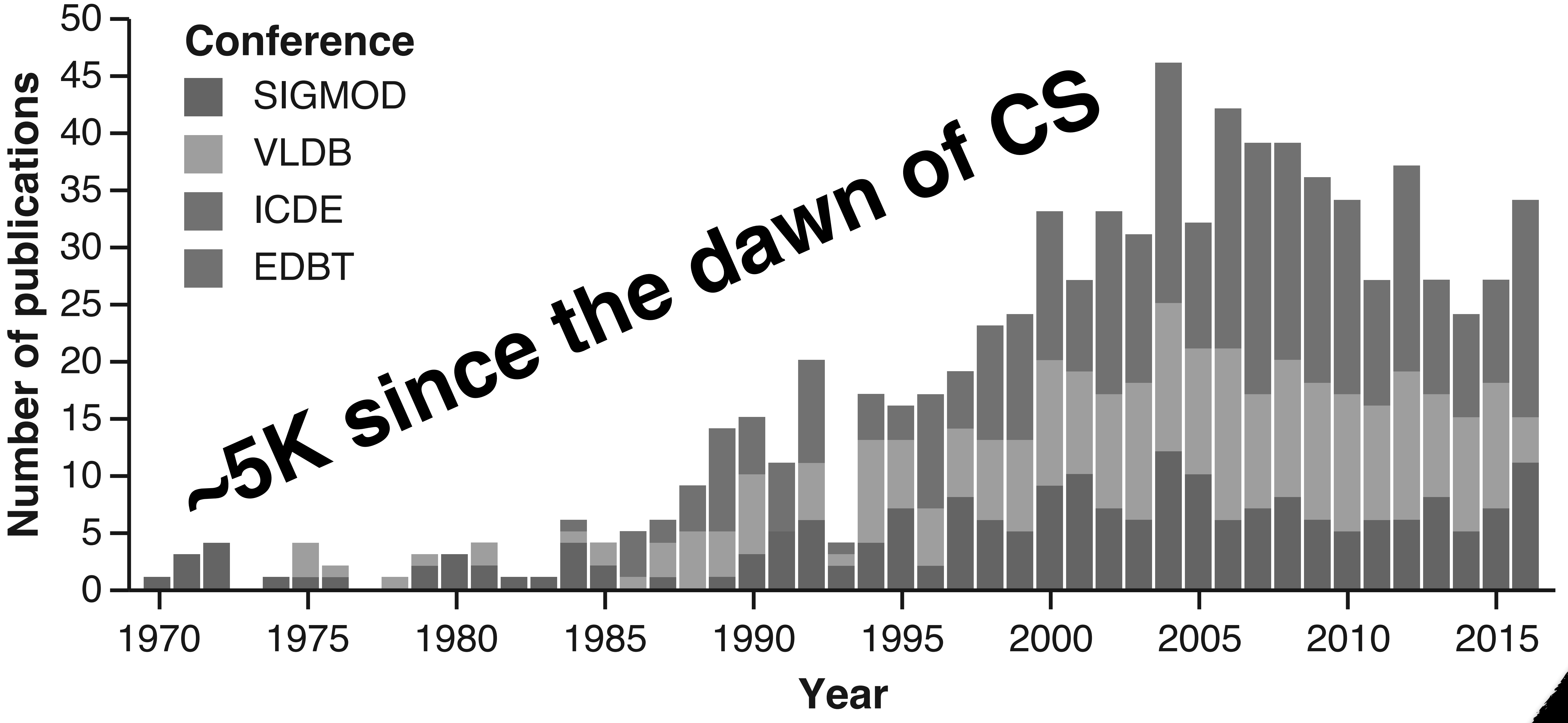


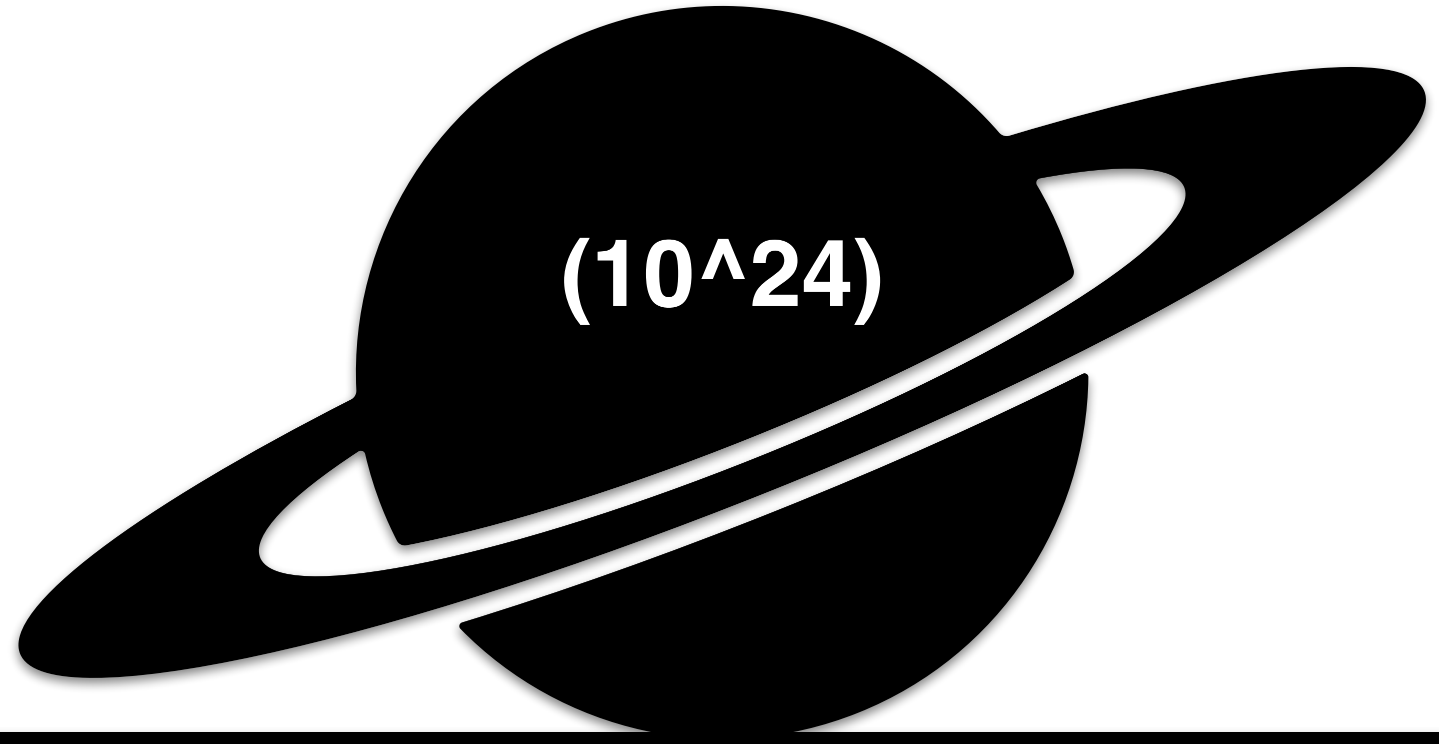


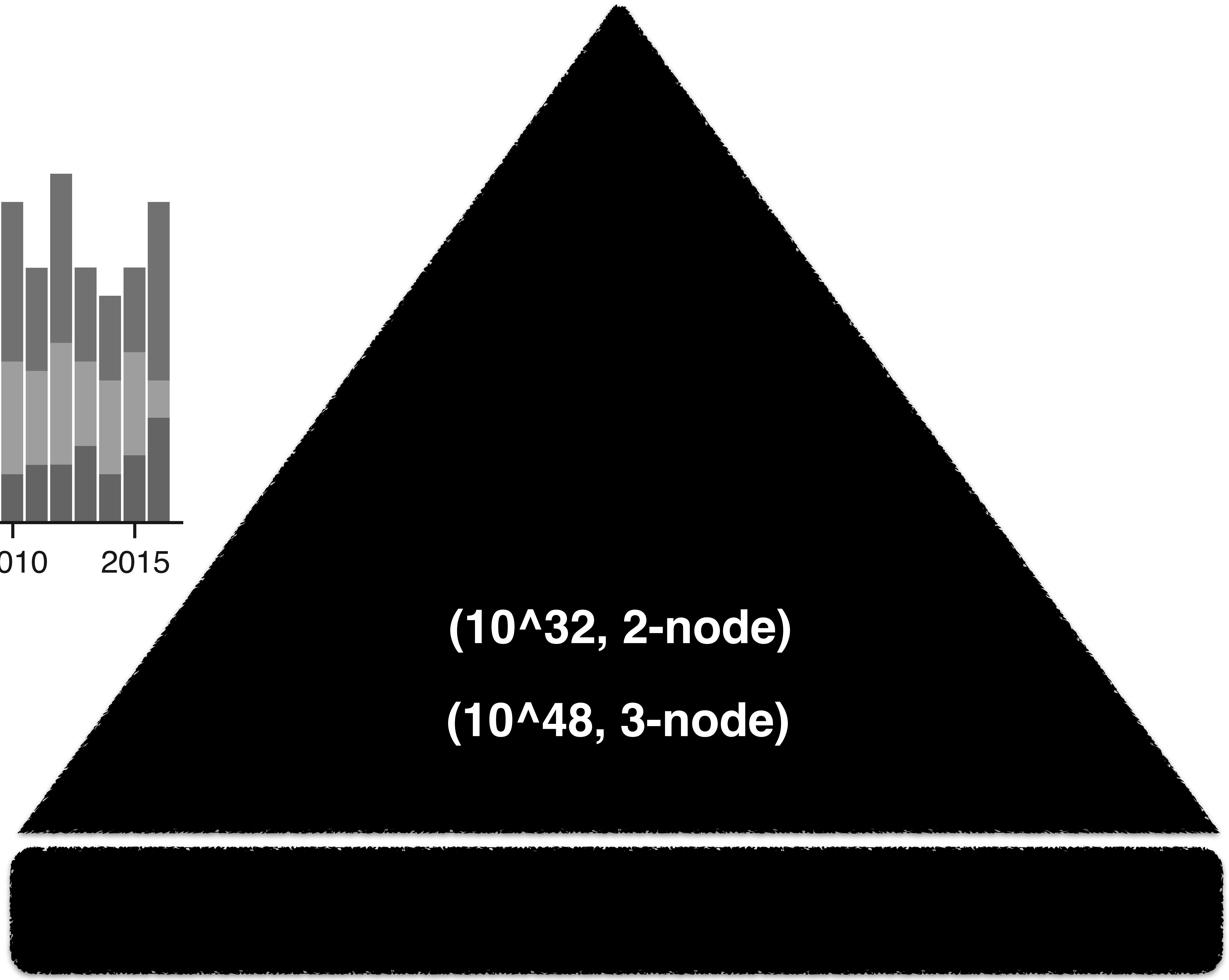


STARS ON THE SKY

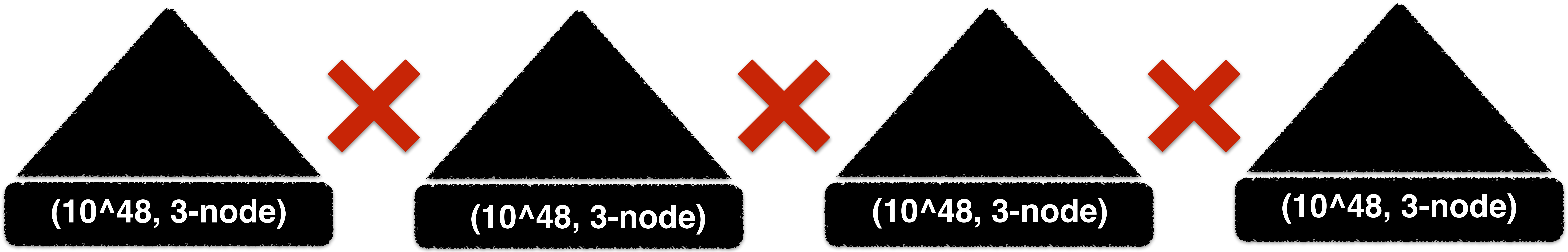
POSSIBLE DATA STRUCTURES





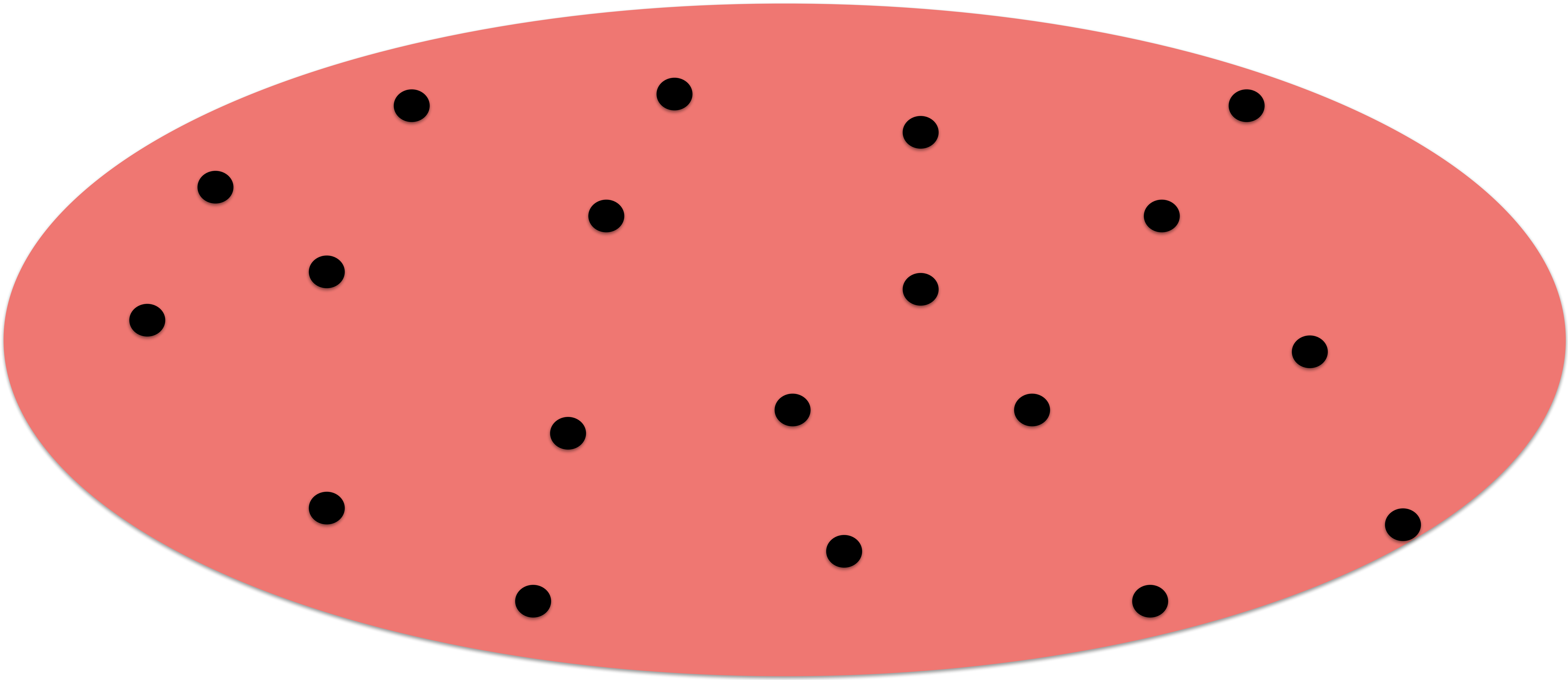


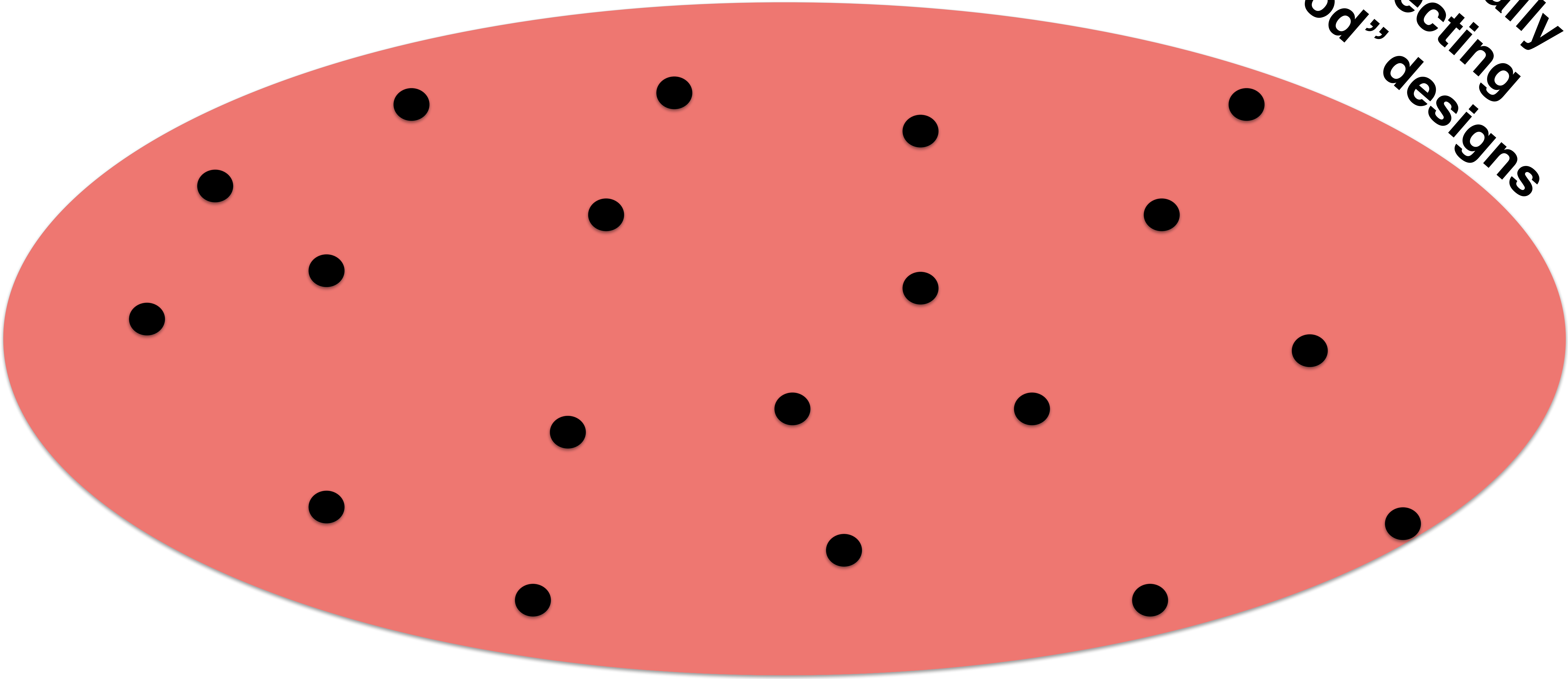
$10^{48} - 5 \times 10^{3} = 10^{48}$ **zero progress**



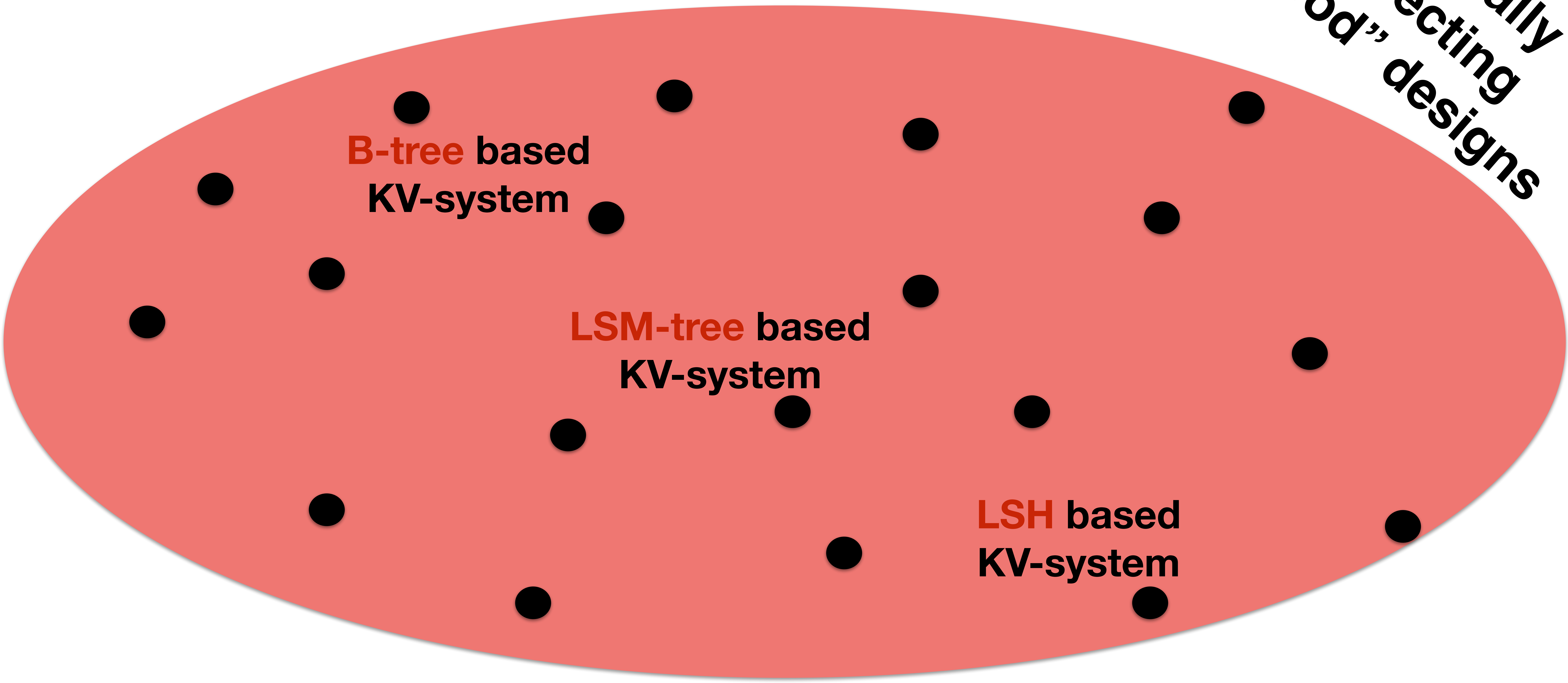
The design space of systems is even larger

The design space of systems is even larger



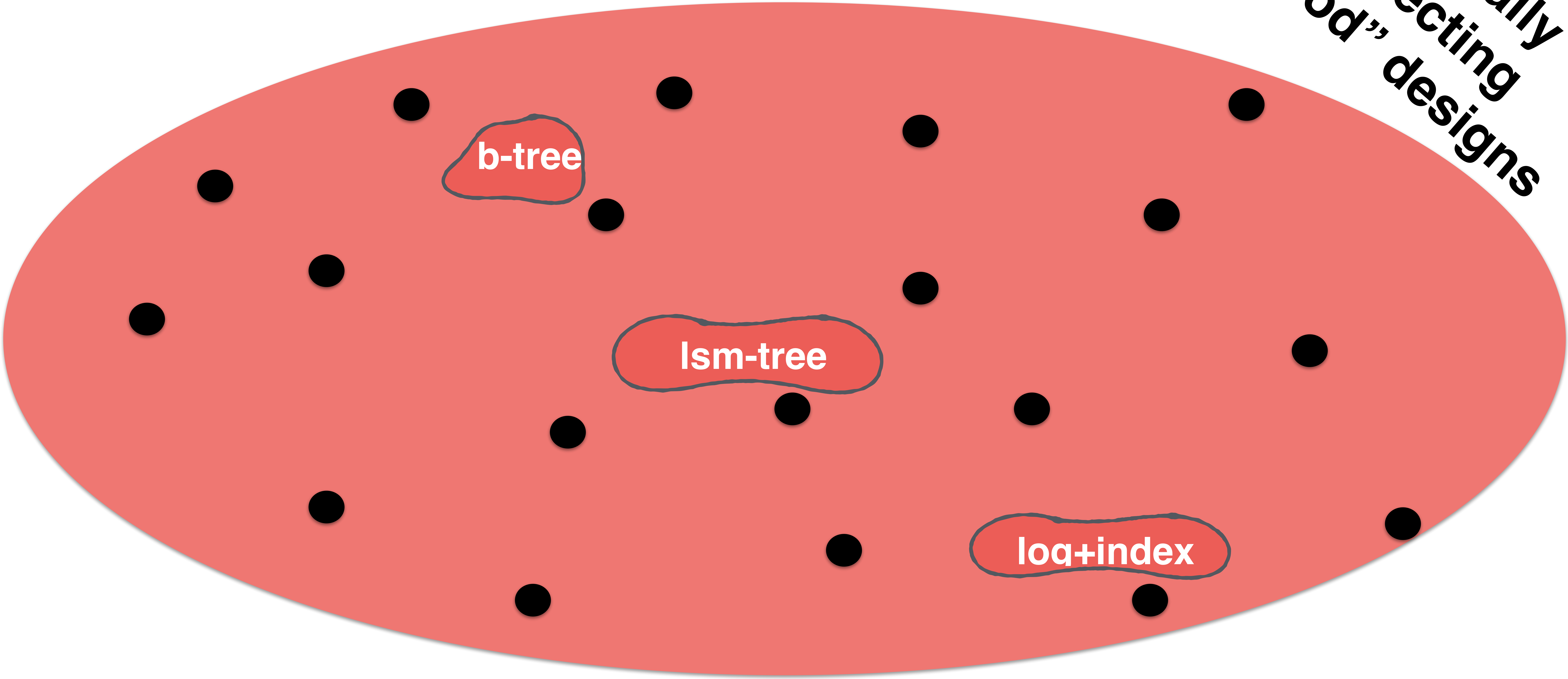


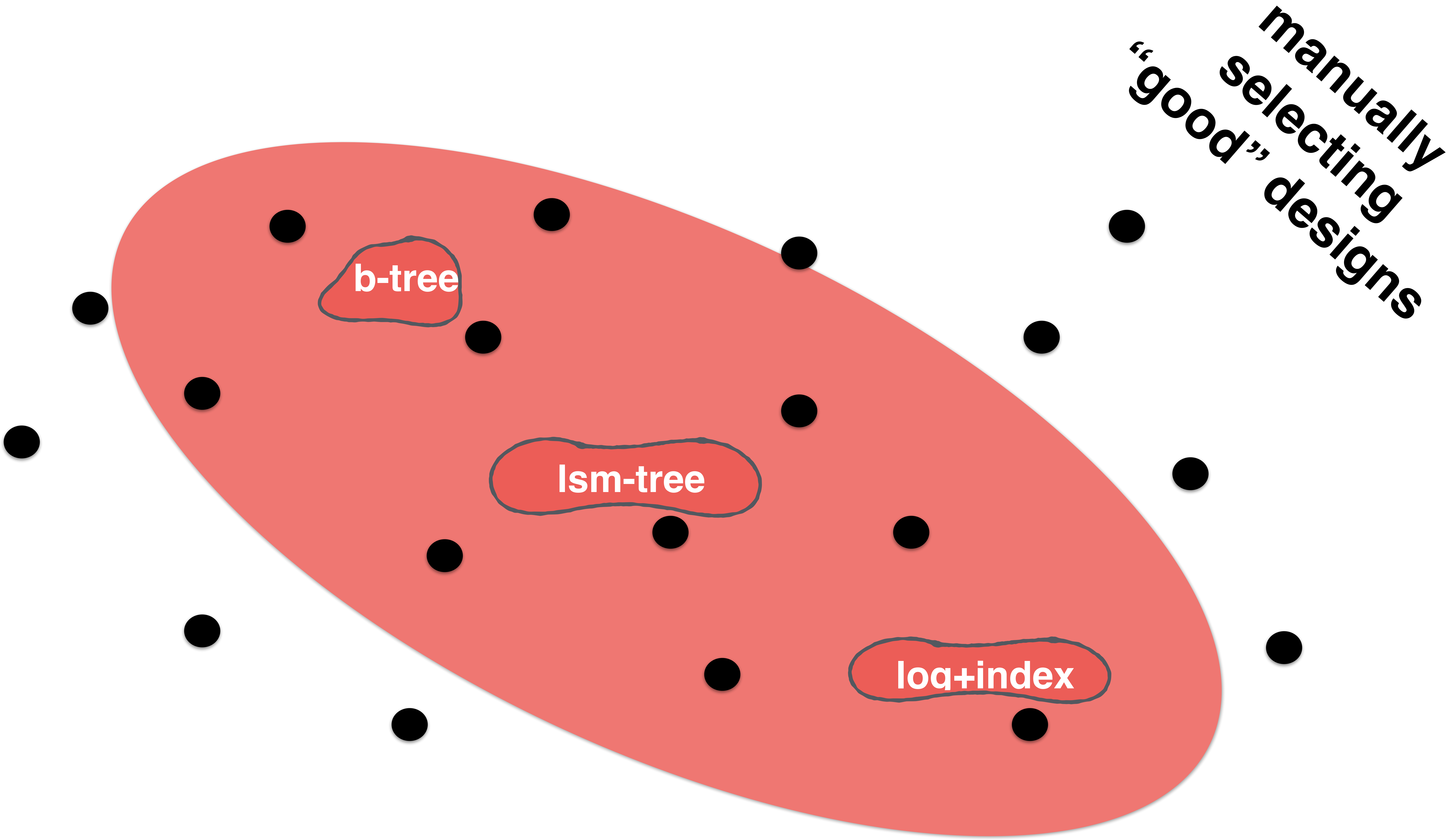
manually selecting "good" designs



manually
selecting
“good” designs
B-tree based
KV-system
LSM-tree based
KV-system
LSH based
KV-system
DASlab
@ Harvard SEAS

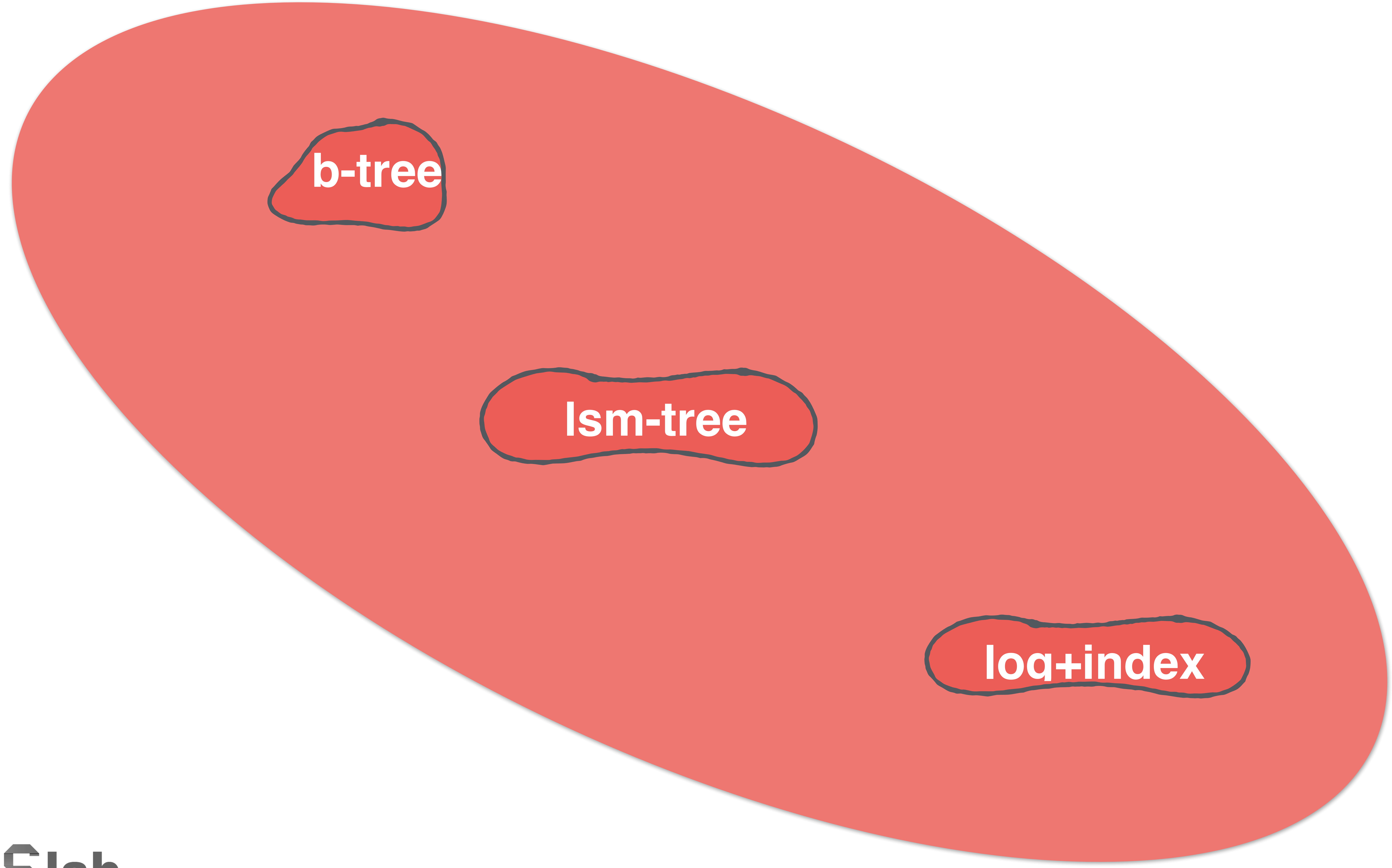
manually selecting “good” designs





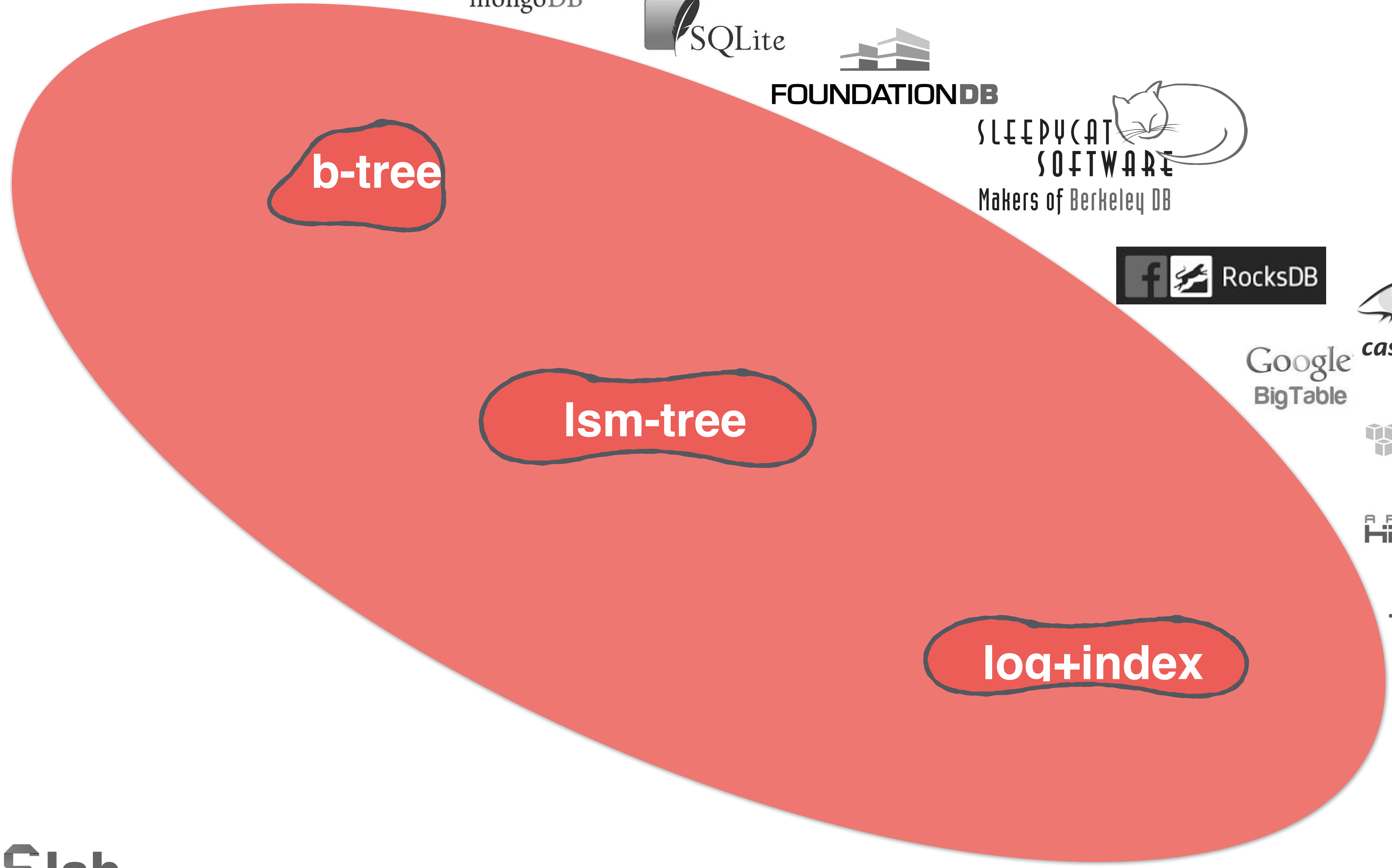
DASlab
@ Harvard SEAS

manually selecting "good" designs



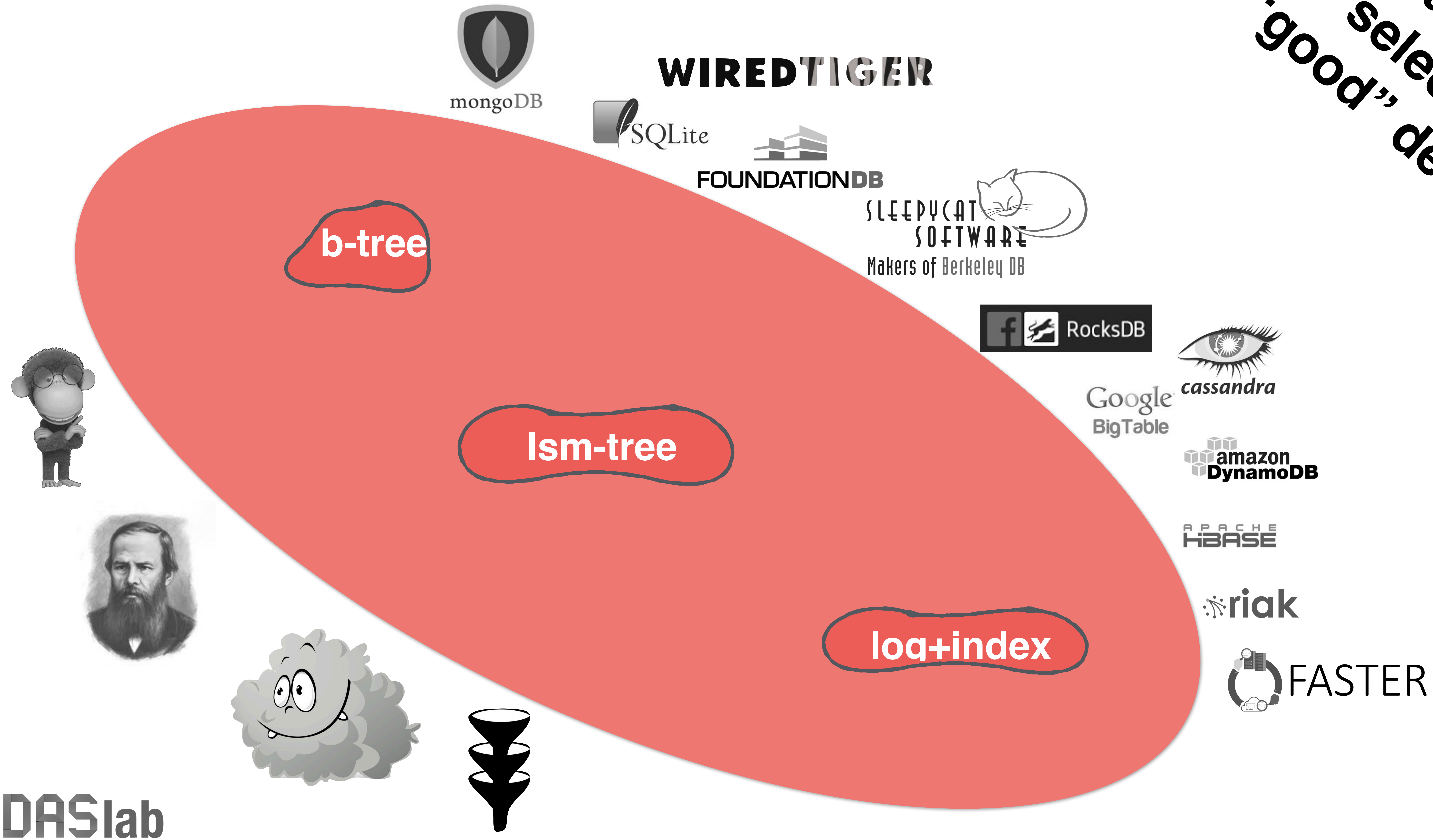
DASlab @ Harvard SEAS

manually selecting “good” designs

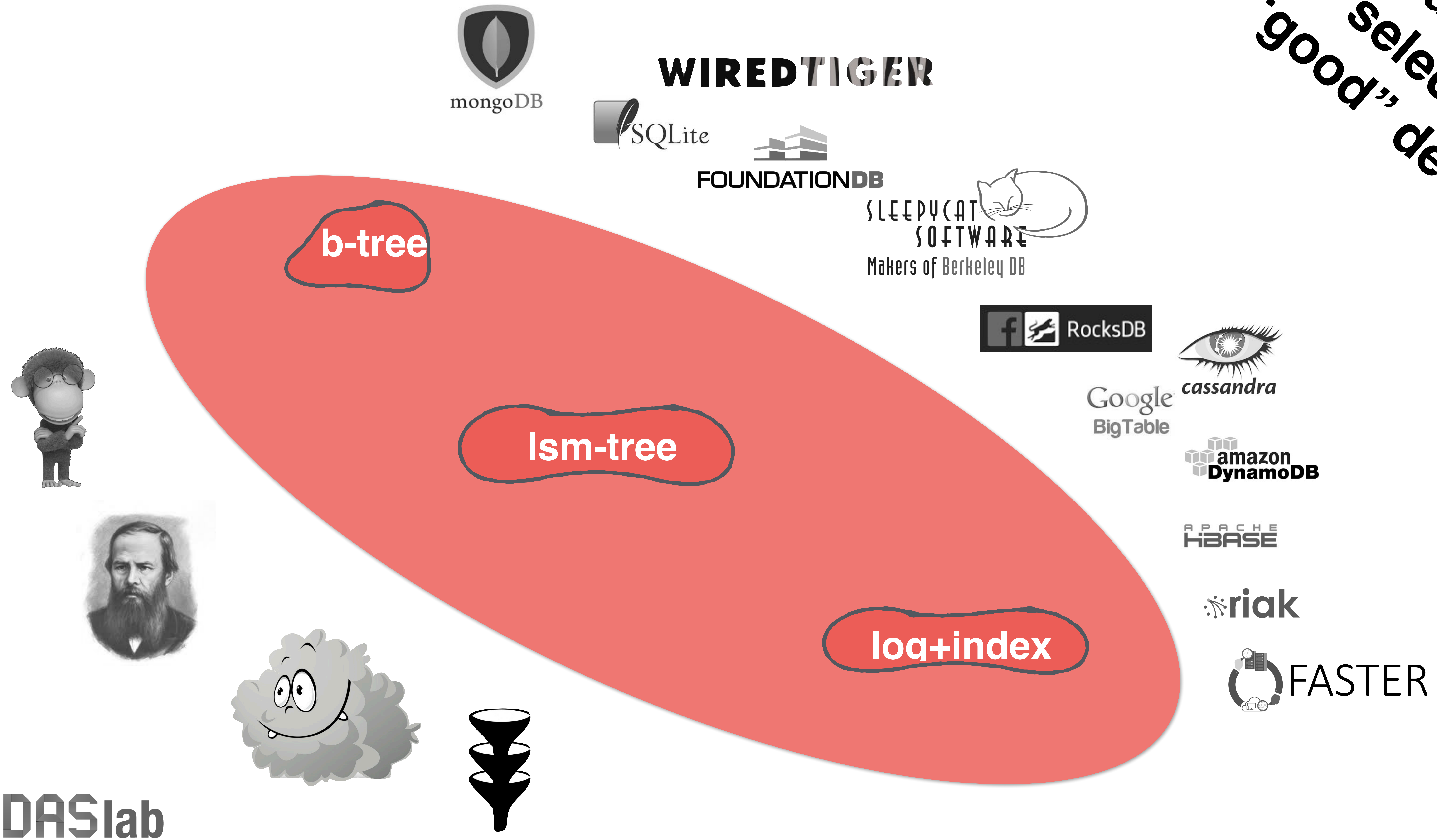


mongoDB
WIREDTIGER
SQLite
FOUNDATIONDB
SLEEPYCAT SOFTWARE
Makers of Berkeley DB
RocksDB
cassandra
Google BigTable
amazon DynamoDB
APACHE HBASE
riak
FASTER

manually selecting "good" designs

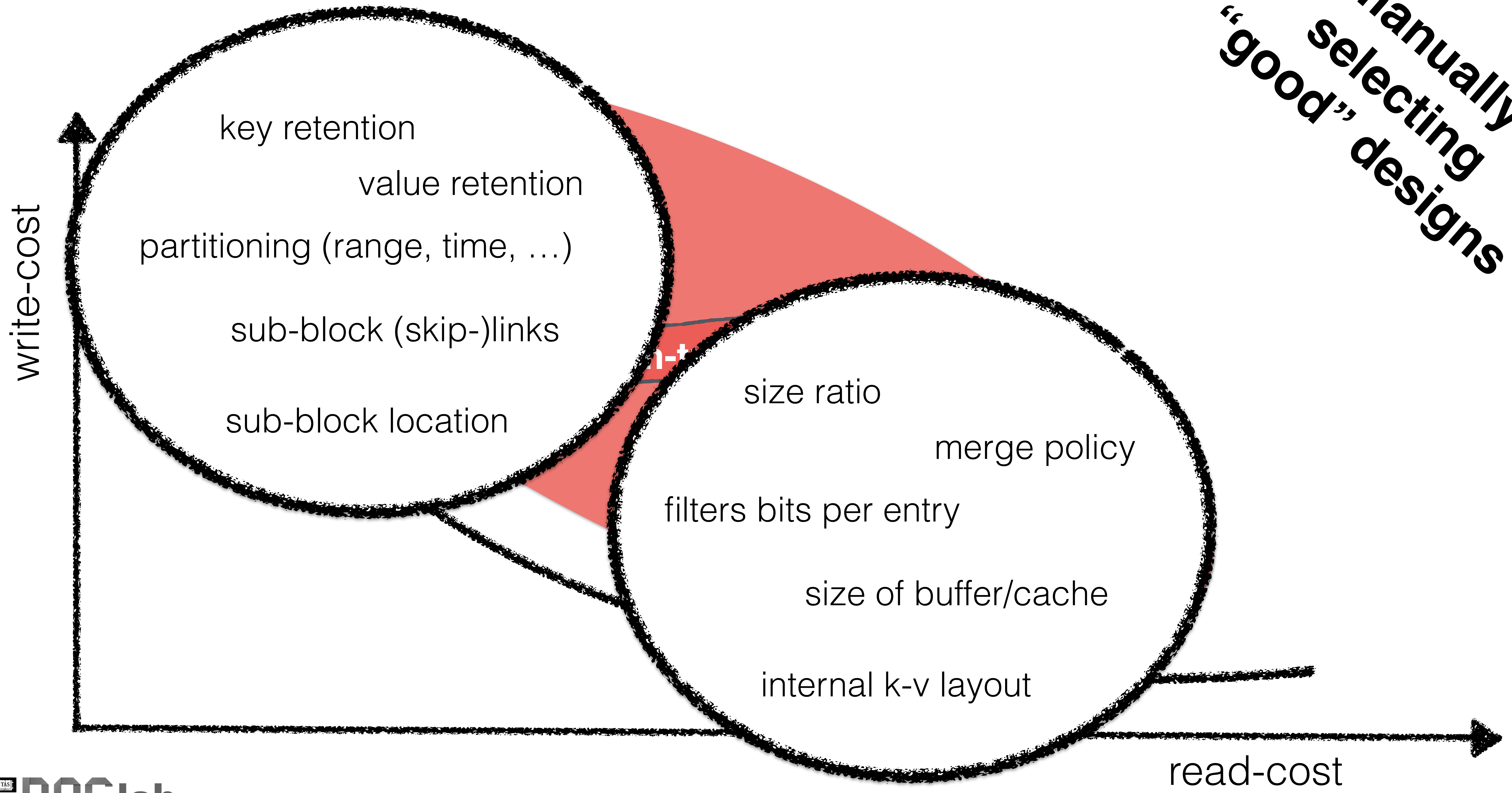


manually selecting “good” designs

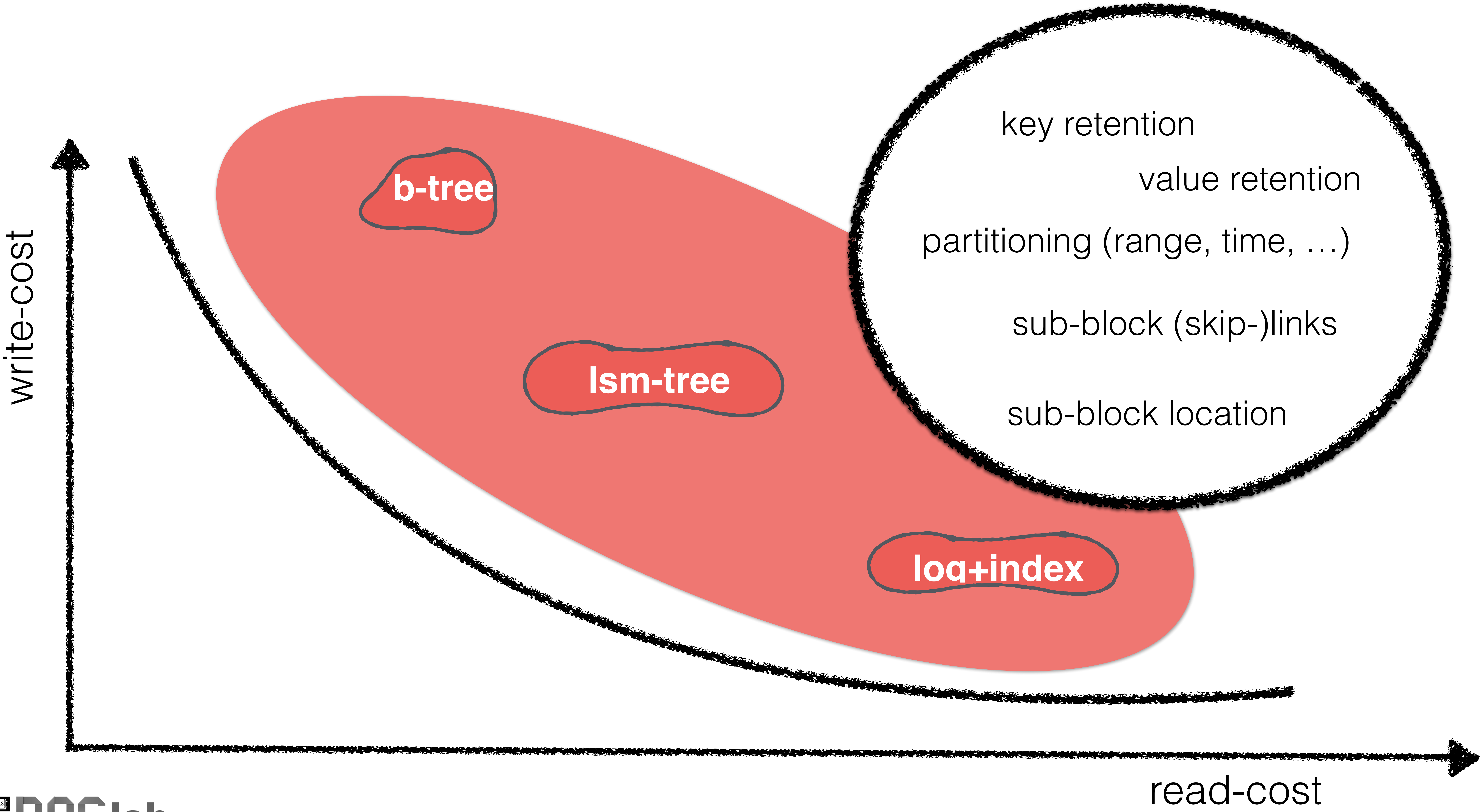



manually
selecting
"good" designs
write-cost
read-cost
b-tree
lsm-tree
log+index
DASlab
@ Harvard SEAS

manually selecting “good” designs



DASlab
@ Harvard SEAS



DASlab @ Harvard SEAS

Design Continuum
unified design template
performance continuum
b-tree
lsm-tree
log+index
write-cost
read-cost
DASlab
@ Harvard SEAS

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

LAYOUT PRIMITIVES (rows 1–9) — Design and hardware specification — *initialized by search through engine design space*

ALGORITHMIC ABSTRACTIONS (rows 10–17) — Data access / Parallelism — *derived with empirically verified rules*

	Design Abstractions of Template	Type/Domain	Example templates for diverse data structures: LSM variants	B-Tree variants	LSH variants	A new design
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)
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)
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.

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			

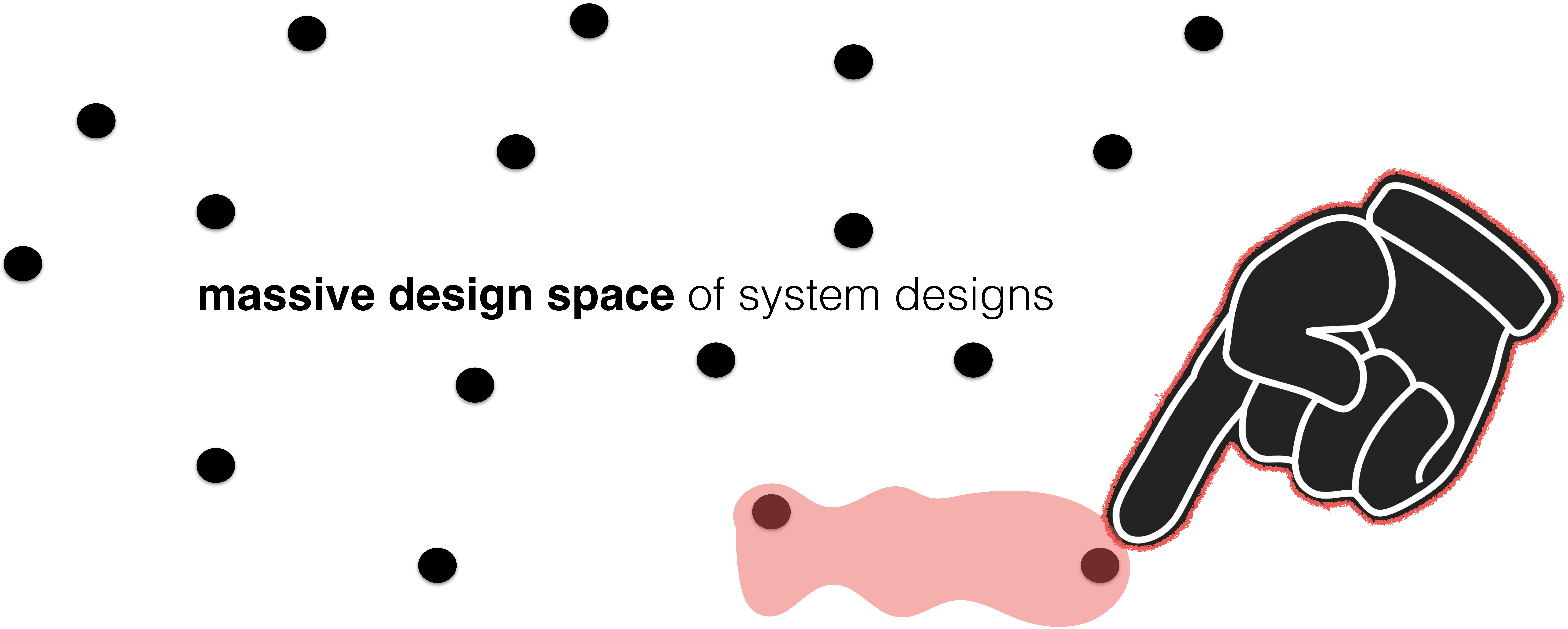
Storage engine template in Cosine and example initializations for diverse storage engine designs.

from write to read optimized

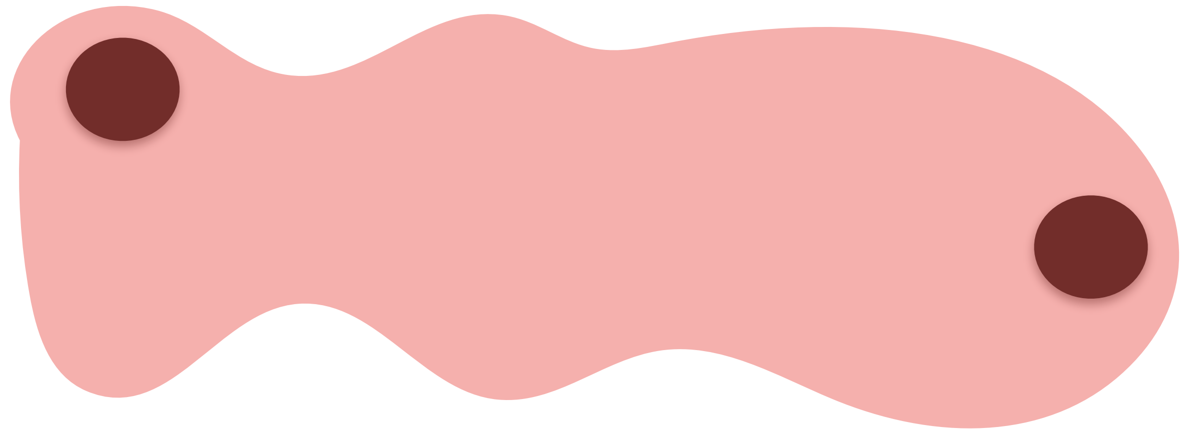
Design Continuums @CDIR2019

Terms \ Designs	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^{ϵ}Tree [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	$O(\frac{N \cdot E}{M_B} \cdot \frac{s}{B})$	$O(\frac{N \cdot E}{M_B} \cdot \frac{s}{B})$	$O(T \cdot \frac{s}{B})$	$O(\frac{s}{B})$	$O(\frac{s}{B})$	$O(\frac{s}{B})$	$O(\frac{s}{B})$	$O(\frac{s}{B})$	$O(\frac{s}{B})$	$O(\frac{s}{B})$

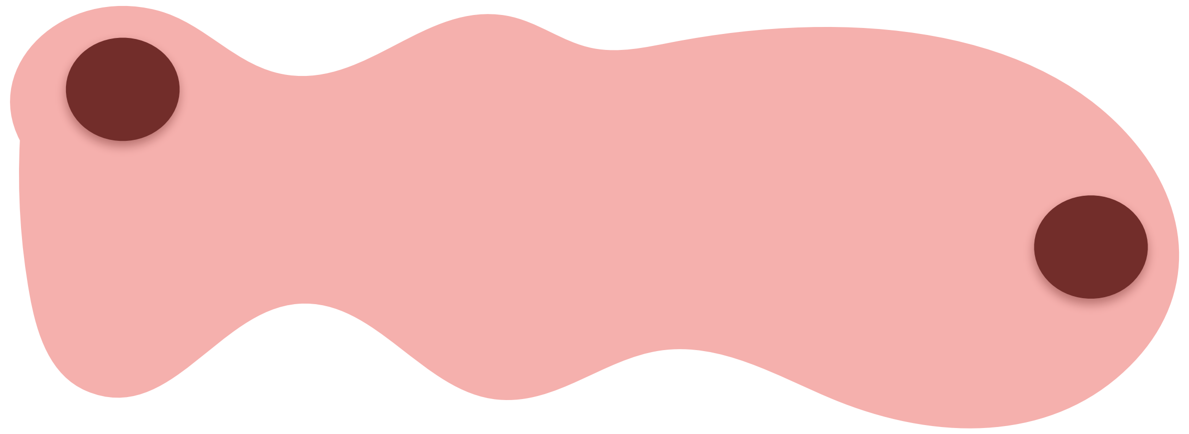
massive design space of system designs


massive design space of system designs

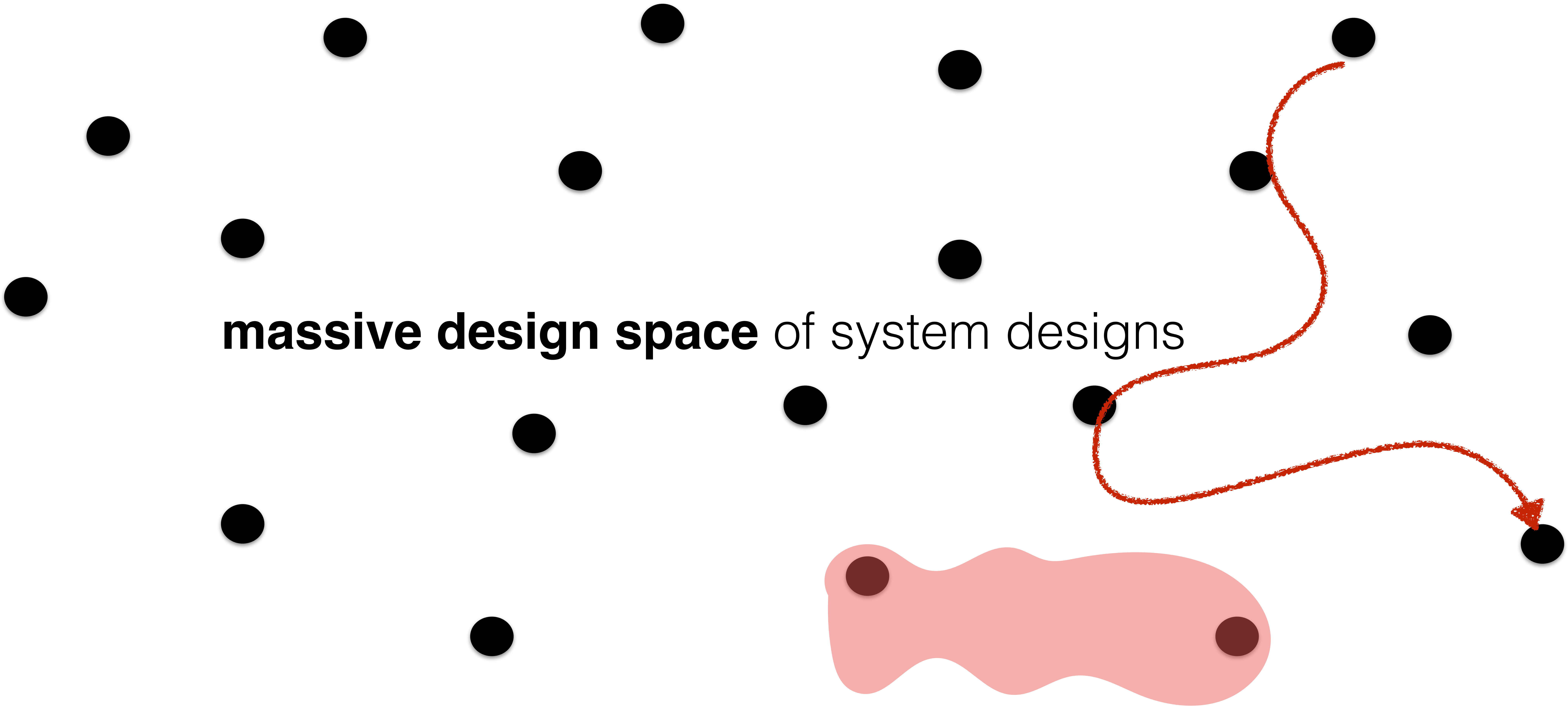
workload/hardware → performance, cloud cost, robustness



workload/hardware performance, cloud cost, robustness

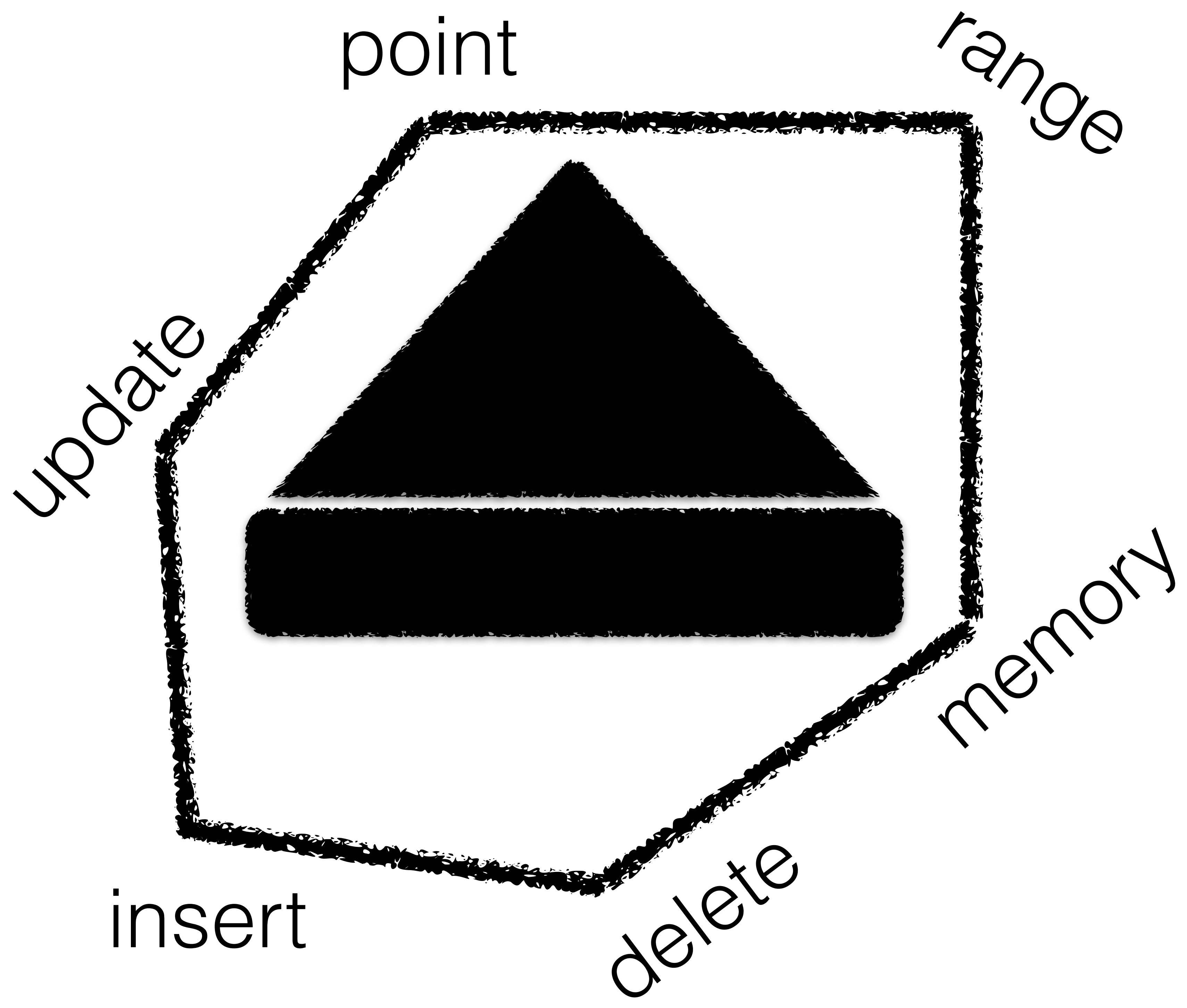


without having to code

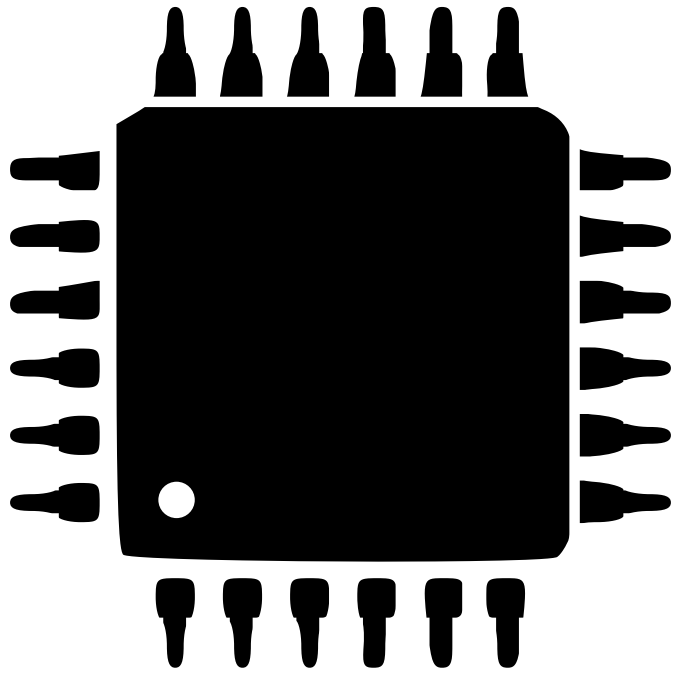

massive design space of system designs

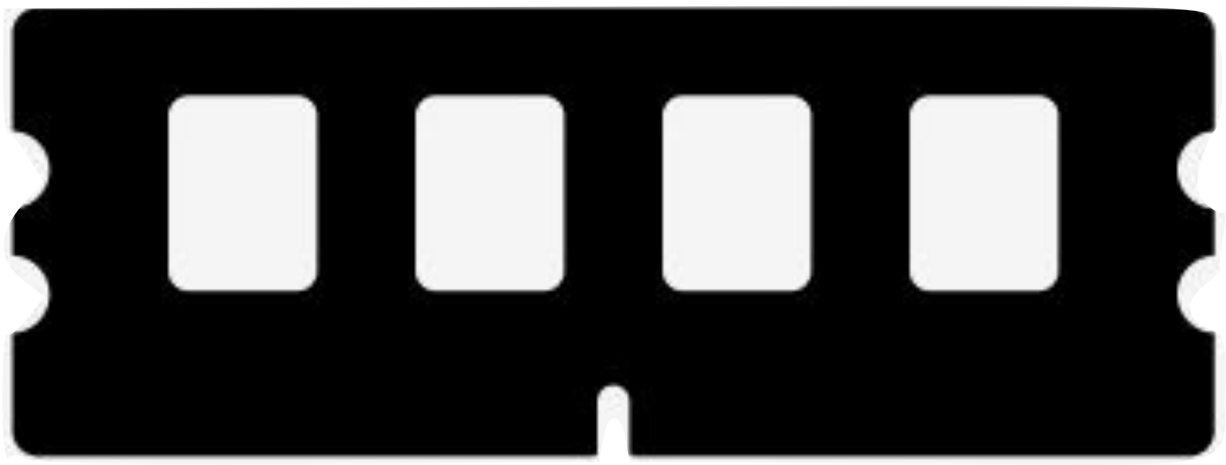
data layouts → algorithm & cost synthesis

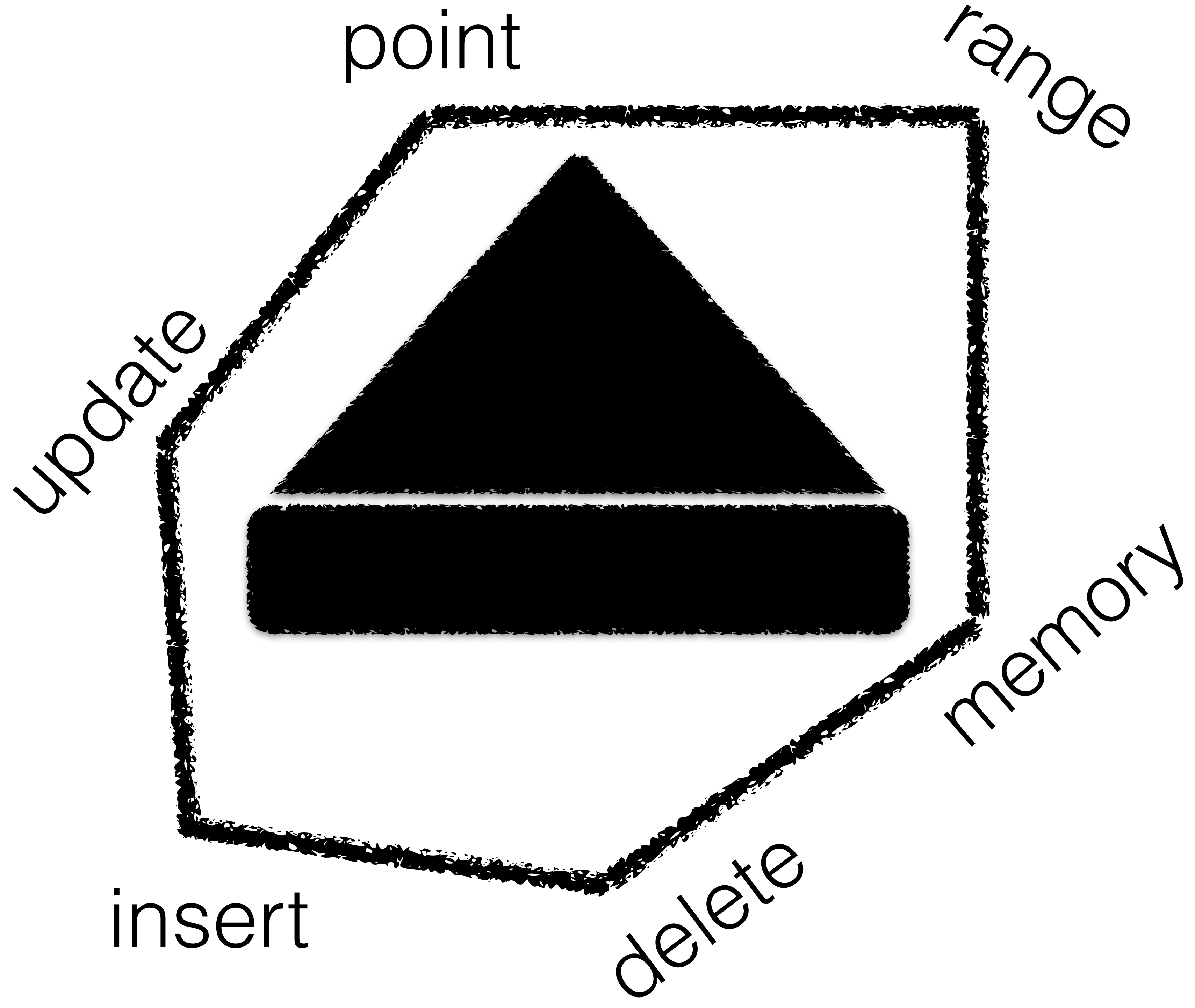
algorithm & cost synthesis



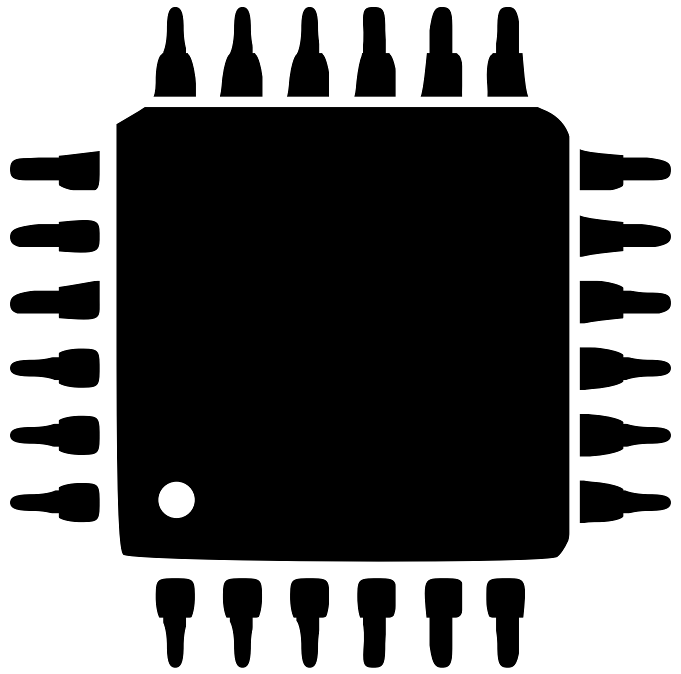
algorithm & cost synthesis

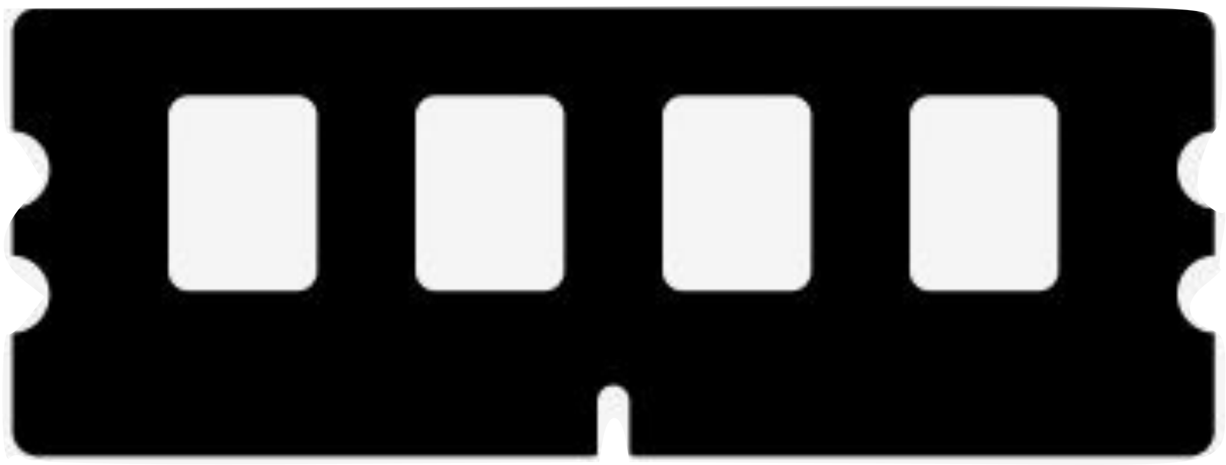


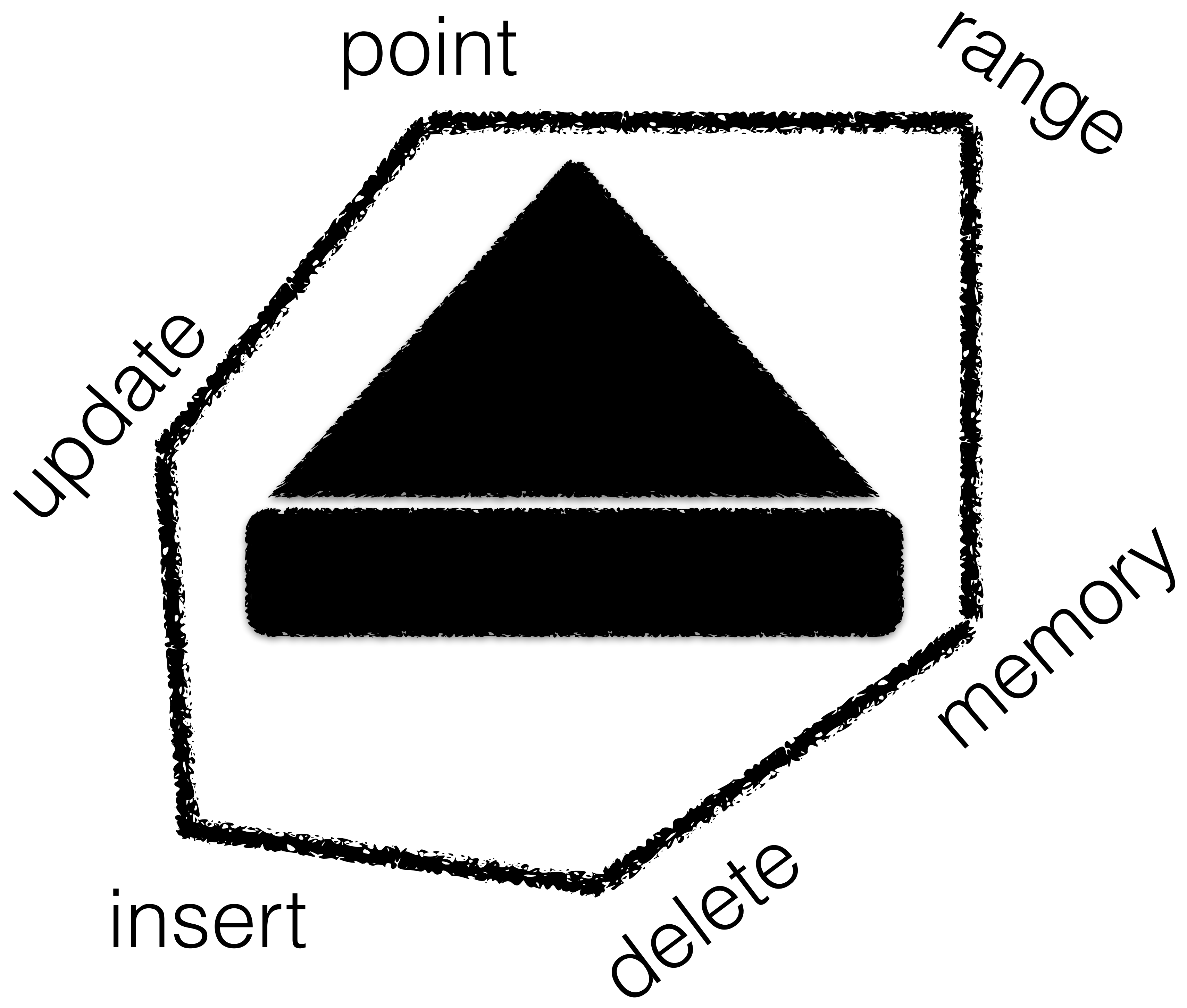


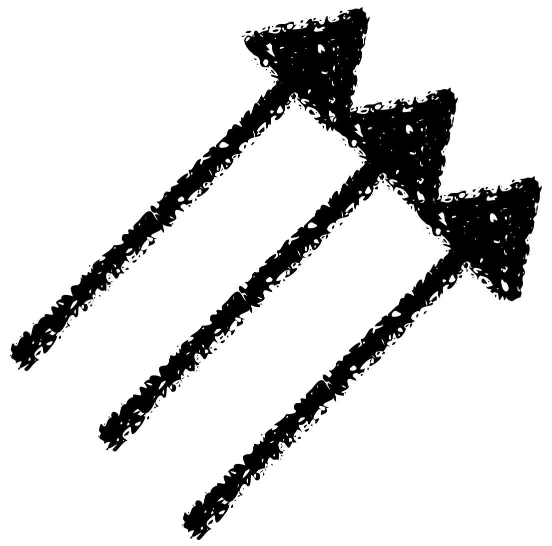


algorithm & cost synthesis



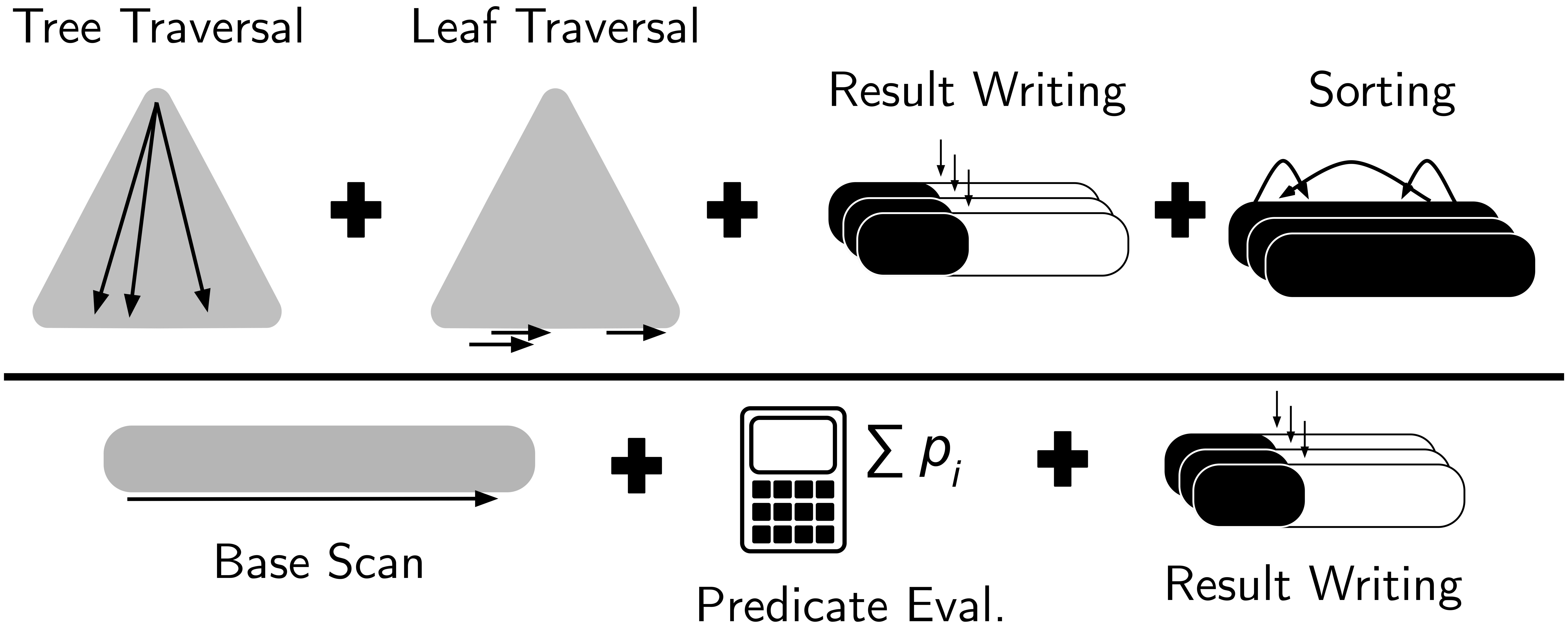






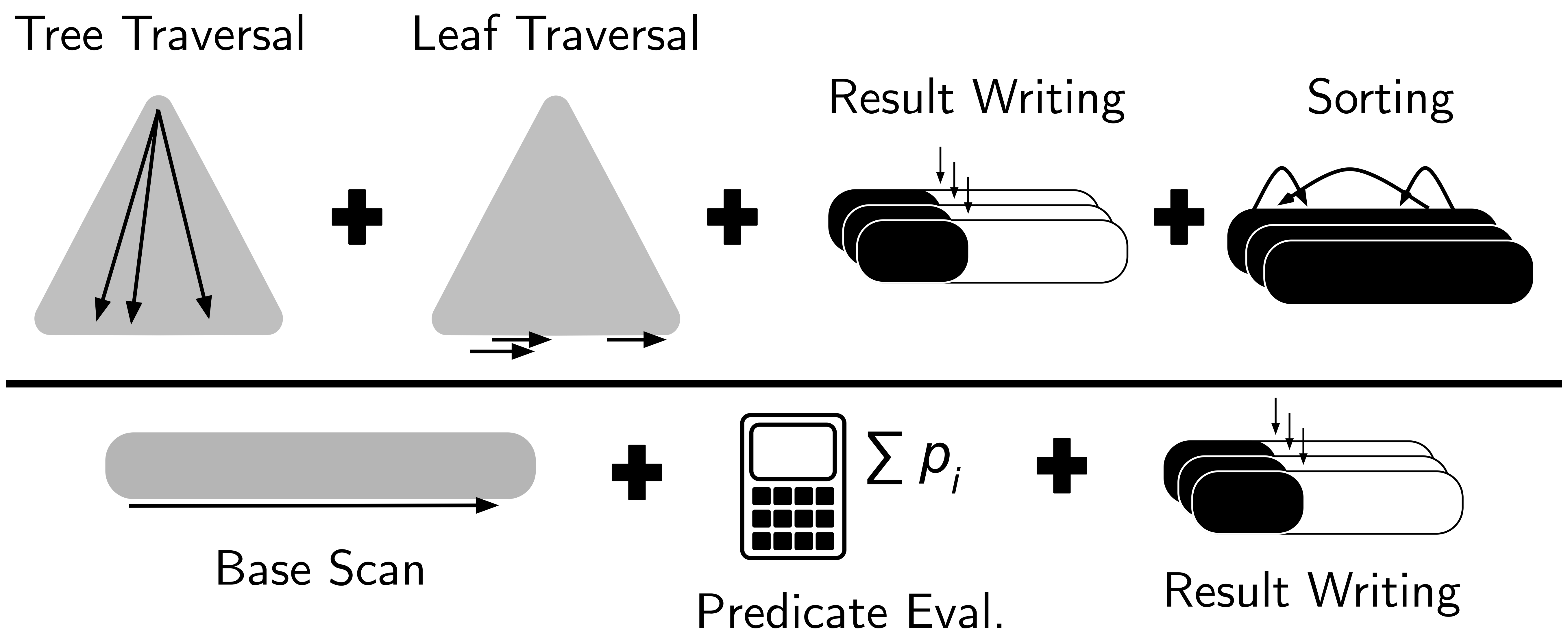
query &
hardware parallelism

Should I scan or should I probe? @SIGMOD2016



$$APS(q, S_{tot}) = \frac{q \cdot \frac{1+\lceil log_b(N) \rceil}{N} \cdot \left(BW_S \cdot C_M + \frac{b \cdot BW_S \cdot C_A}{2} + \frac{b \cdot BW_S \cdot f_p \cdot p}{2} \right)}{max\left(ts, 2 \cdot f_p \cdot p \cdot q \cdot BW_S\right) + S_{tot} \cdot rw \cdot \frac{BW_S}{BW_R}}$$
$$+ \frac{S_{tot}\left(\frac{BW_S \cdot C_M}{b} + (aw + ow) \cdot \frac{BW_S}{BW_I} + rw \cdot \frac{BW_S}{BW_R} \right)}{max\left(ts, 2 \cdot f_p \cdot p \cdot q \cdot BW_S\right) + S_{tot} \cdot rw \cdot \frac{BW_S}{BW_R}}$$
$$+ \frac{S_{tot} \cdot log_2\left(S_{tot} \cdot N\right) \cdot BW_S \cdot C_A}{max\left(ts, 2 \cdot f_p \cdot p \cdot q \cdot BW_S\right) + S_{tot} \cdot rw \cdot \frac{BW_S}{BW_R}}$$

Should I scan or should I probe? @SIGMOD2016

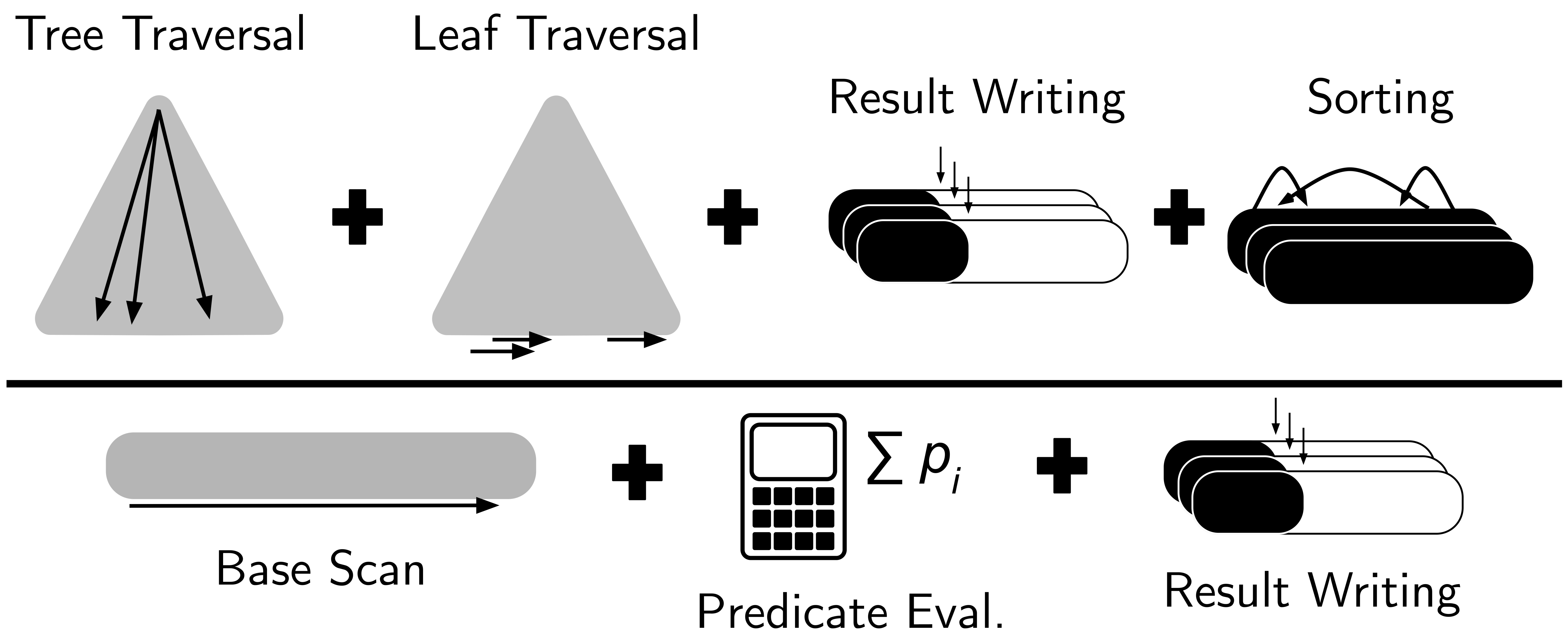


Workload	q	number of queries
	s_i	selectivity of query i
	S_{tot}	total selectivity of the workload
Dataset	N	data size (tuples per column)
	ts	tuple size (bytes per tuple)
Hardware	C_A	L1 cache access (sec)
	C_M	LLC miss: memory access (sec)
	BW_S	scanning bandwidth (GB/s)
	BW_R	result writing bandwidth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

But we have a massive design space to cover…..

$$APS(q, S_{tot}) = \frac{q \cdot \frac{1+\lceil log_b(N) \rceil}{N} \cdot \left(BW_S \cdot C_M + \frac{b \cdot BW_S \cdot C_A}{2} + \frac{b \cdot BW_S \cdot f_p \cdot p}{2} \right)}{max\left(ts, 2 \cdot f_p \cdot p \cdot q \cdot BW_S\right) + S_{tot} \cdot rw \cdot \frac{BW_S}{BW_R}}$$
$$+ \frac{S_{tot} \left(\frac{BW_S \cdot C_M}{b} + (aw + ow) \cdot \frac{BW_S}{BW_I} + rw \cdot \frac{BW_S}{BW_R} \right)}{max\left(ts, 2 \cdot f_p \cdot p \cdot q \cdot BW_S\right) + S_{tot} \cdot rw \cdot \frac{BW_S}{BW_R}}$$
$$+ \frac{S_{tot} \cdot log_2\left(S_{tot} \cdot N\right) \cdot BW_S \cdot C_A}{max\left(ts, 2 \cdot f_p \cdot p \cdot q \cdot BW_S\right) + S_{tot} \cdot rw \cdot \frac{BW_S}{BW_R}}$$

Should I scan or should I probe? @SIGMOD2016



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

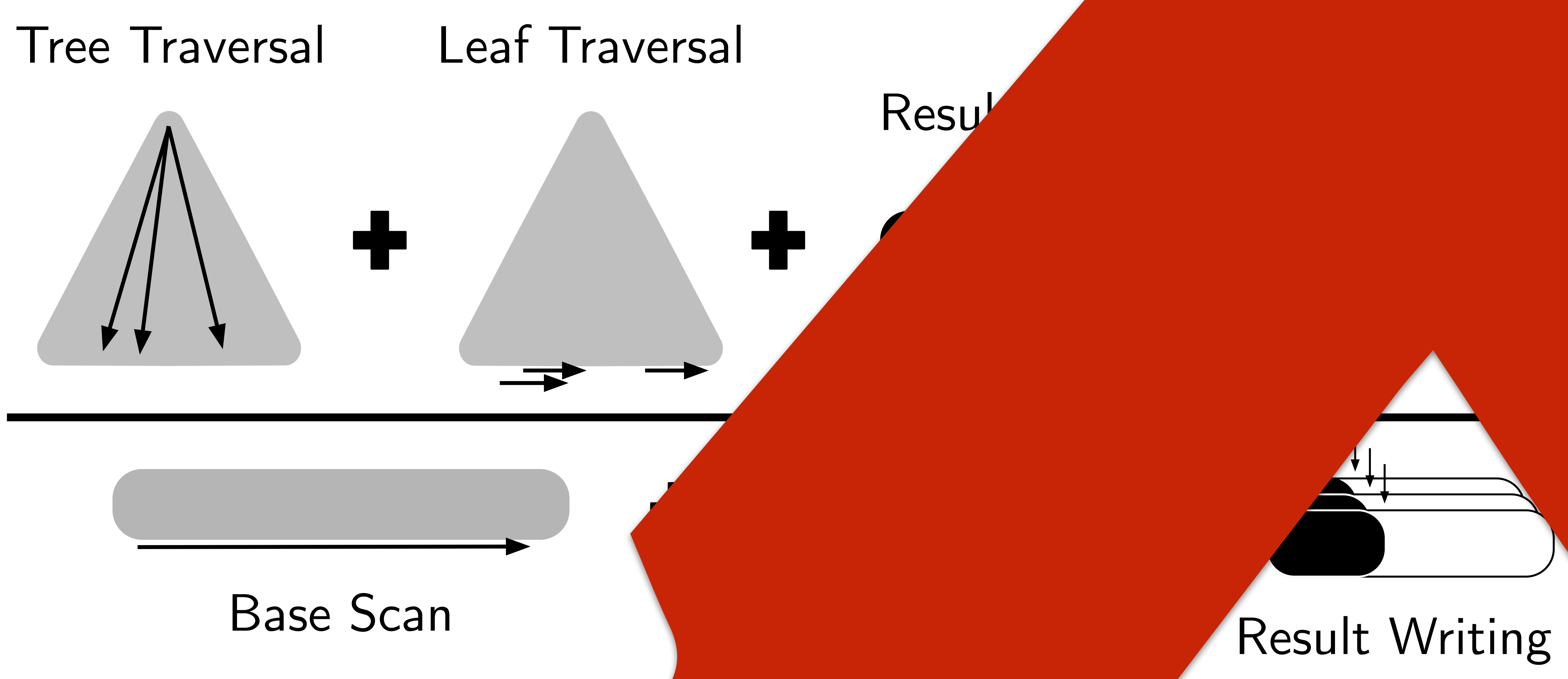
But we have massive des space to cove

$$APS(q, S_{tot}) = \frac{q \cdot 1 \ldots + \frac{b \cdot BW_S \cdot C_A}{2} + \frac{b \cdot BW_S \cdot f_p \cdot p}{2})}{\ldots q \cdot BW_S) + S_{tot} \cdot rw \cdot \frac{BW_S}{BW_R}}$$

$$\frac{\ldots w + ow) \cdot \frac{BW_S}{BW_I} + rw \cdot \frac{BW_S}{BW_R})}{\ldots p \cdot p \cdot q \cdot BW_S) + S_{tot} \cdot rw \cdot \frac{BW_S}{BW_R}}$$

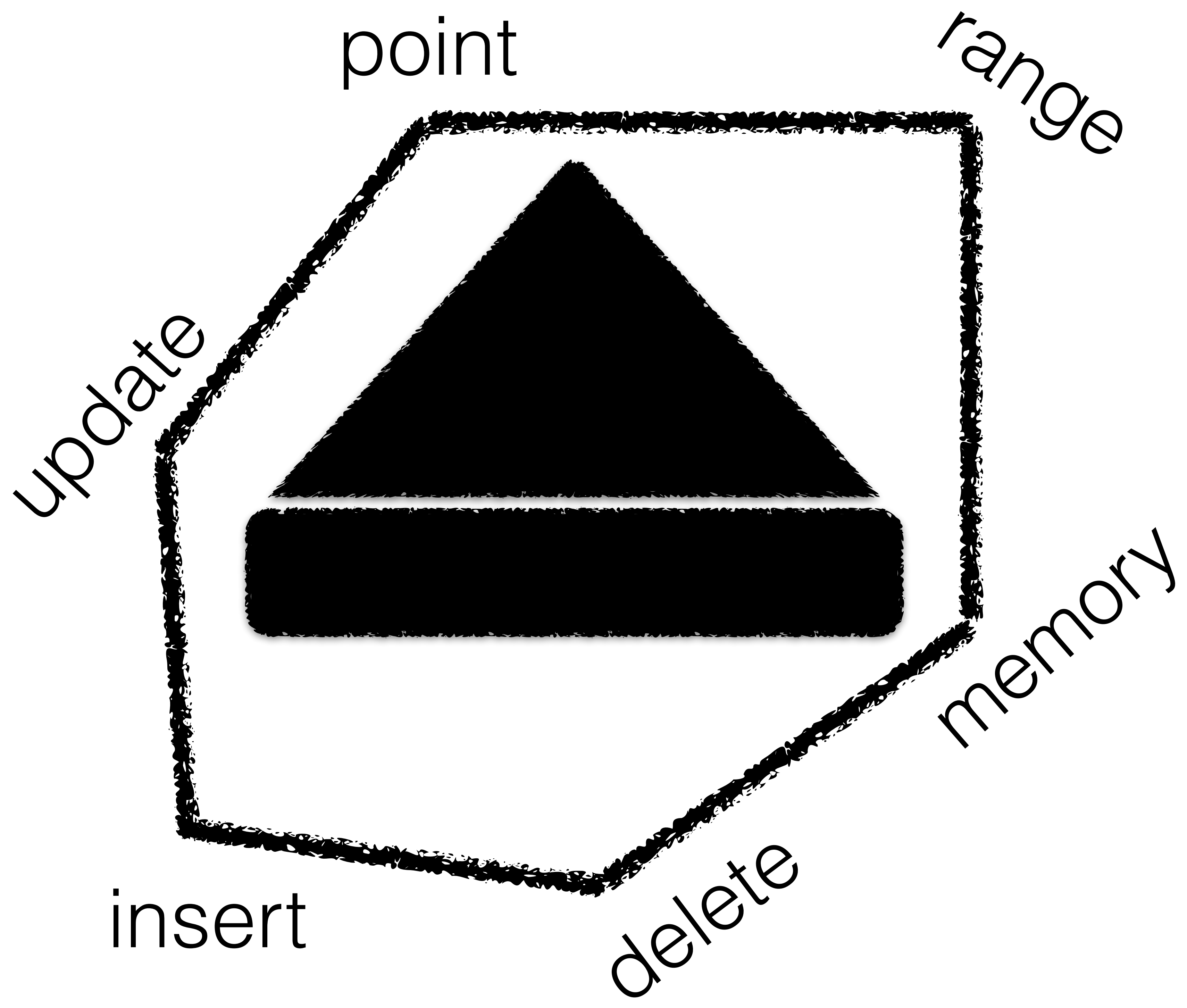
$$\frac{\ldots \cdot log_2(S_{tot} \cdot N) \cdot BW_S \cdot C_A}{\ldots , 2 \cdot f_p \cdot p \cdot q \cdot BW_S) + S_{tot} \cdot rw \cdot \frac{BW_S}{BW_R}}$$

Should I scan or should I probe? @

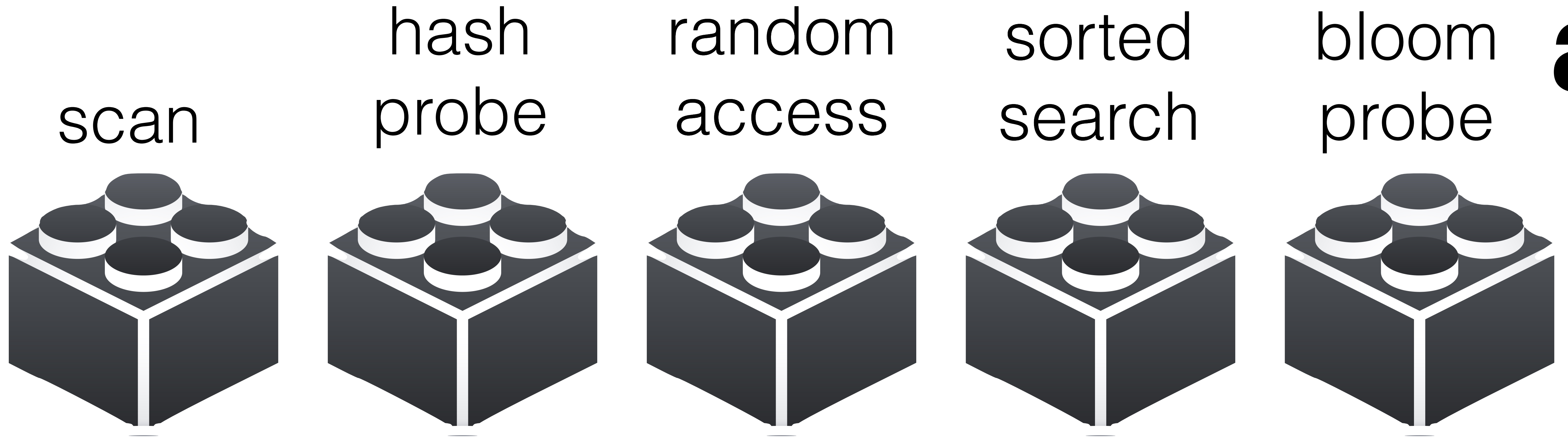


kload	q	number of queries
	s_i	selectivity of query i
	S_{tot}	total selectivity of the workload
	N	data size (tuples per column)
	ts	tuple size (bytes per tuple)
	C_A	L1 cache access (sec)
		LLC miss: memory access (sec)
		scanning bandwidth (GB/s)
		result writing bandwidth (GB/s)
		af traversal bandwidth (GB/s)
		inverse of CPU frequency
		or accounting for pipelining
		ult width (bytes per output tuple)
		ee fanout
		attribute width (bytes of the indexed column)
		offset width (bytes of the index column offset)

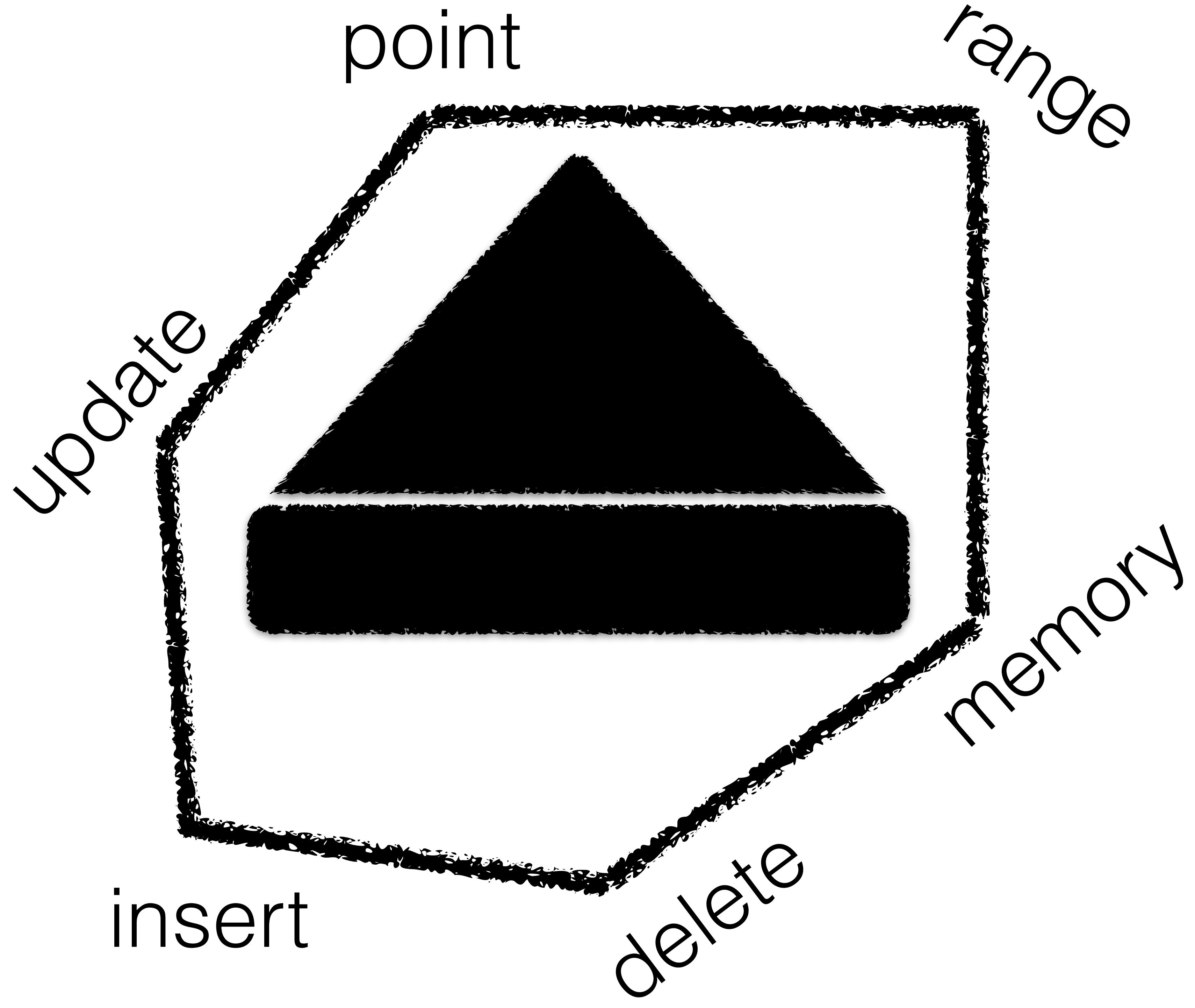
algorithm & cost synthesis



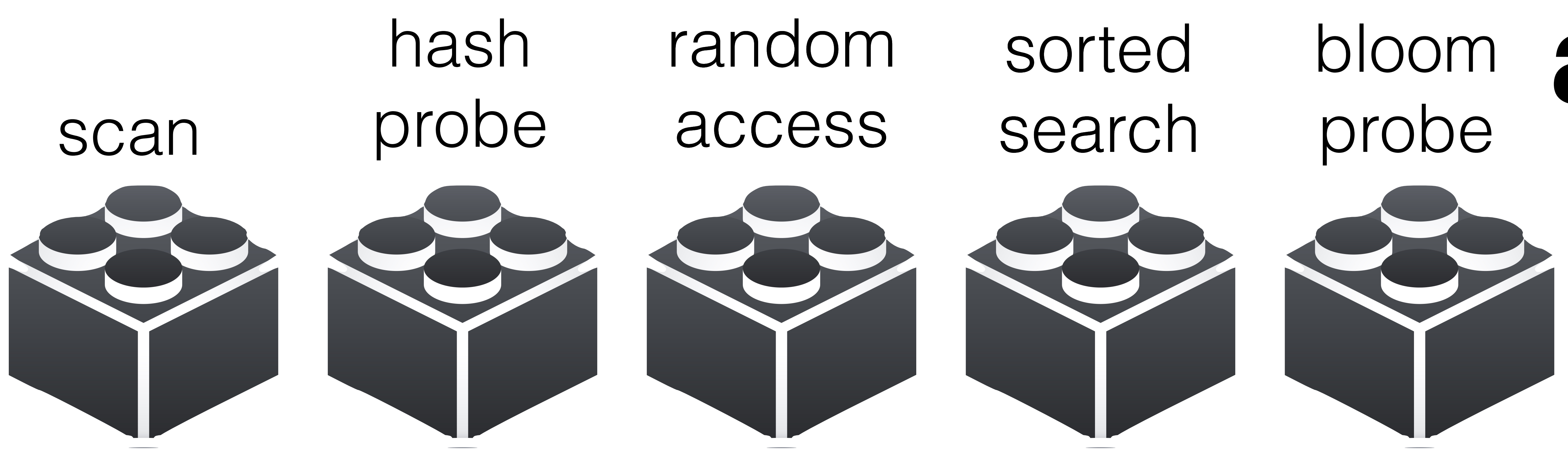
algorithm & cost synthesis



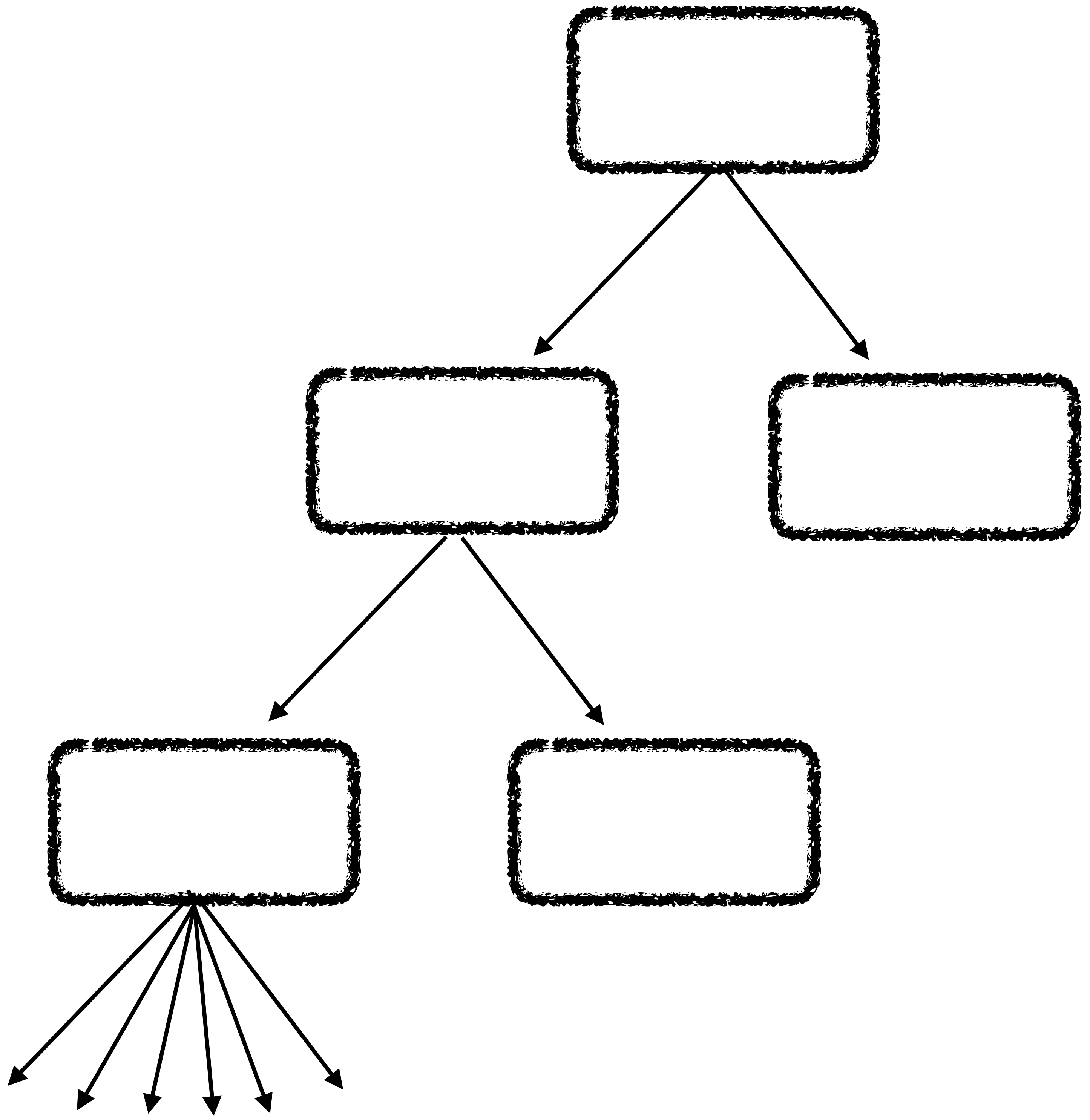
synthesize from first principles

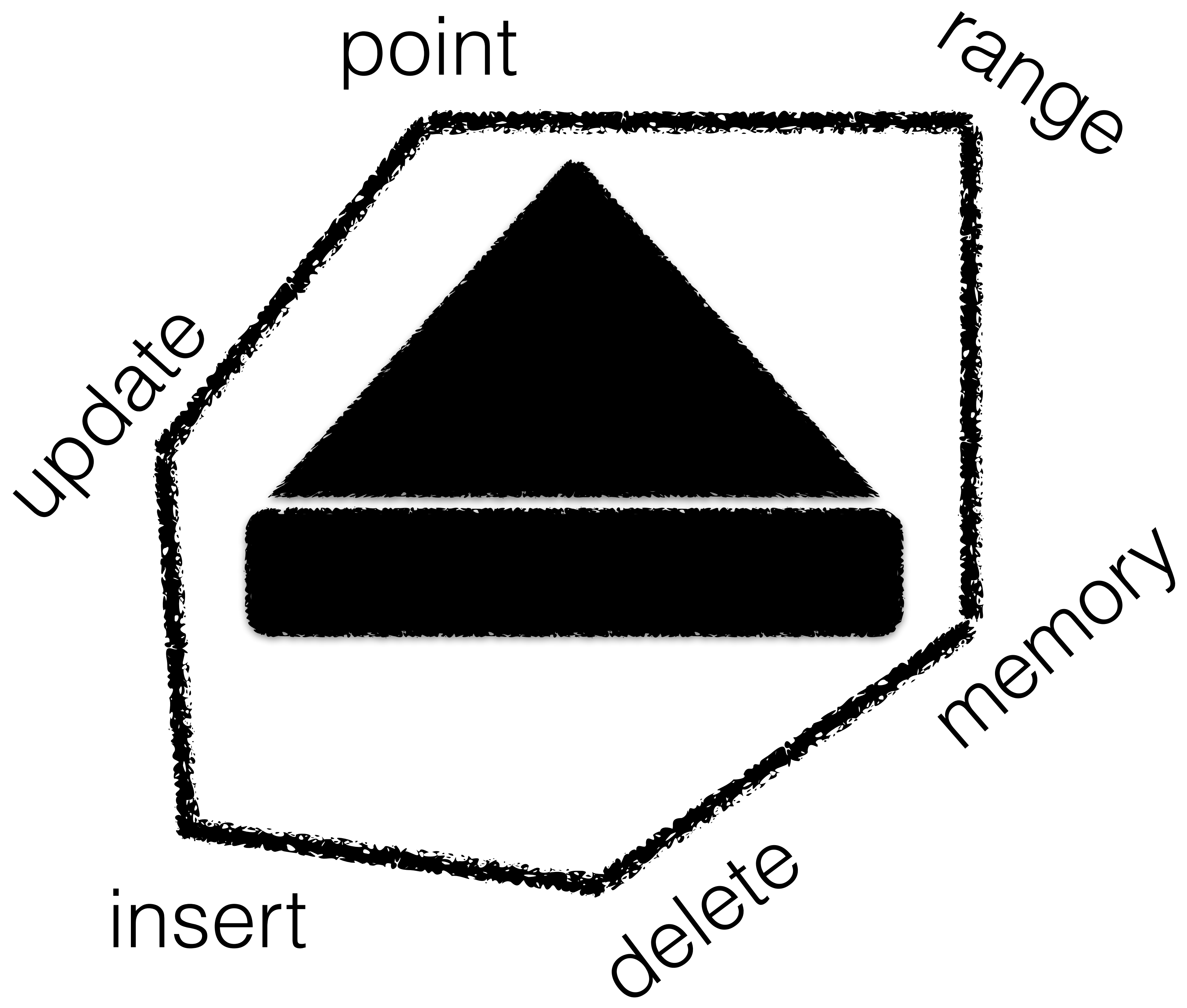


algorithm & cost synthesis

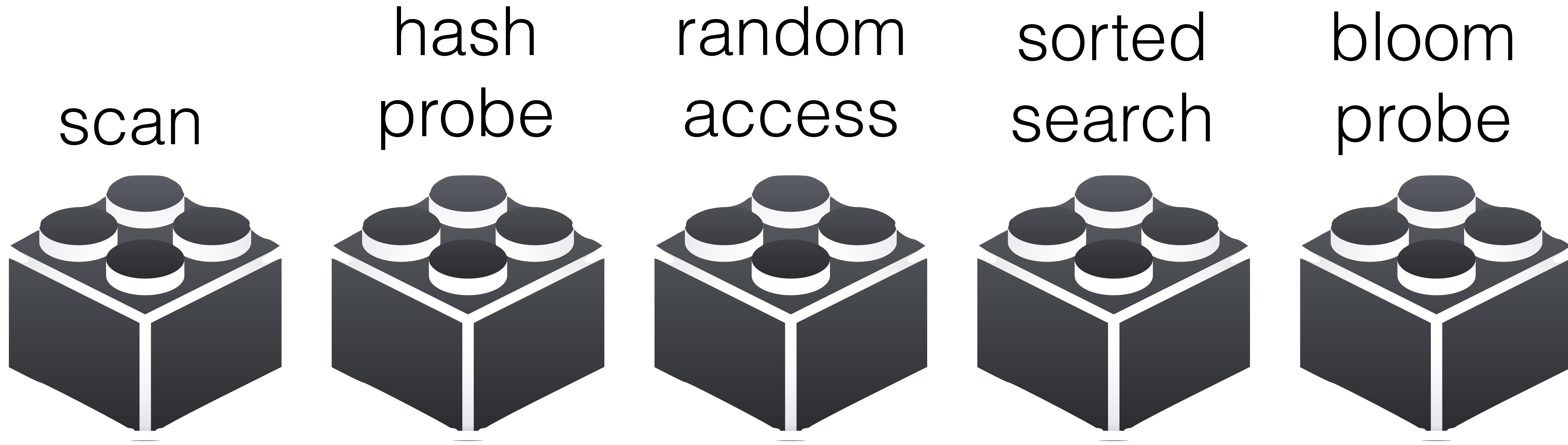


synthesize from first principles

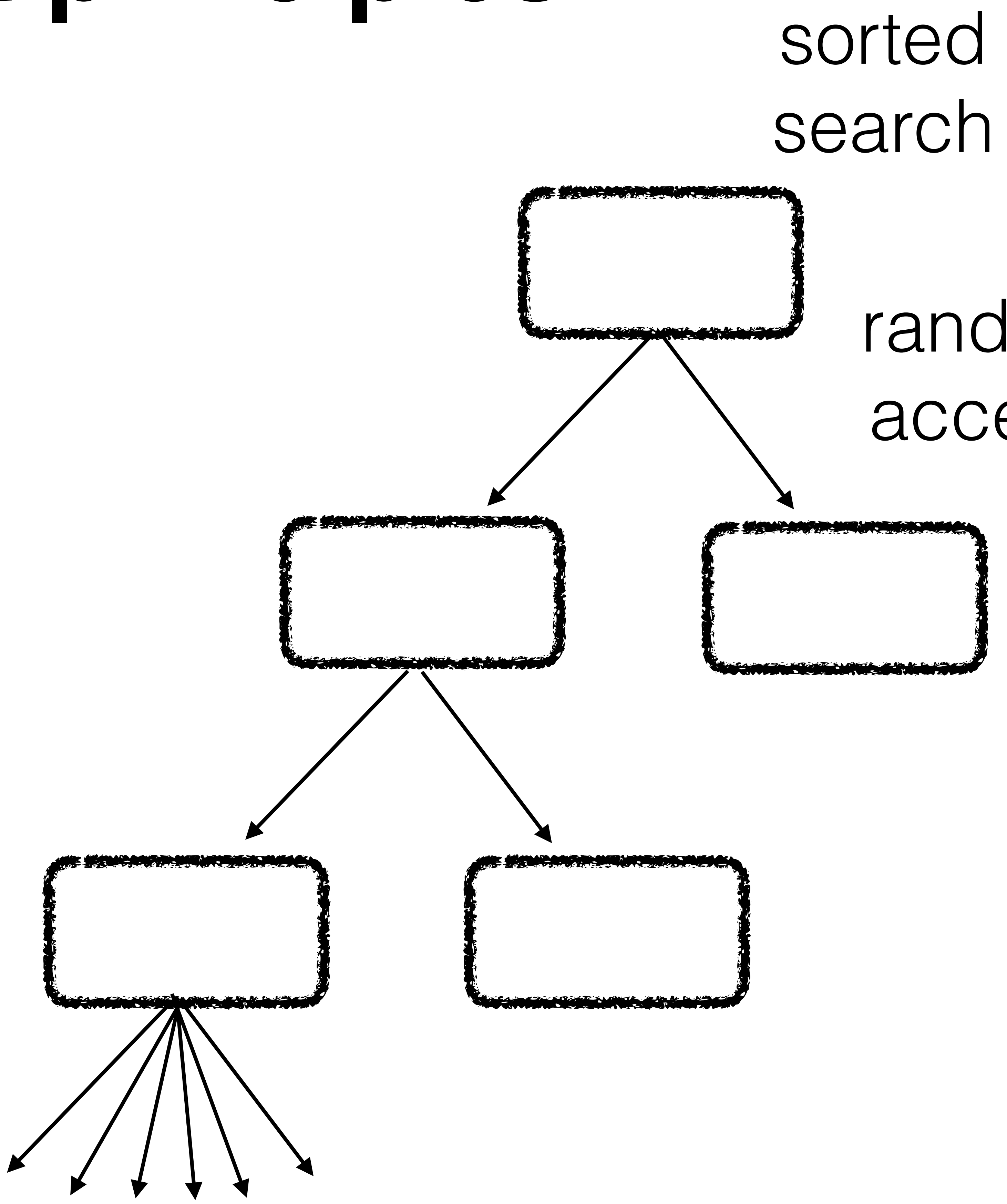


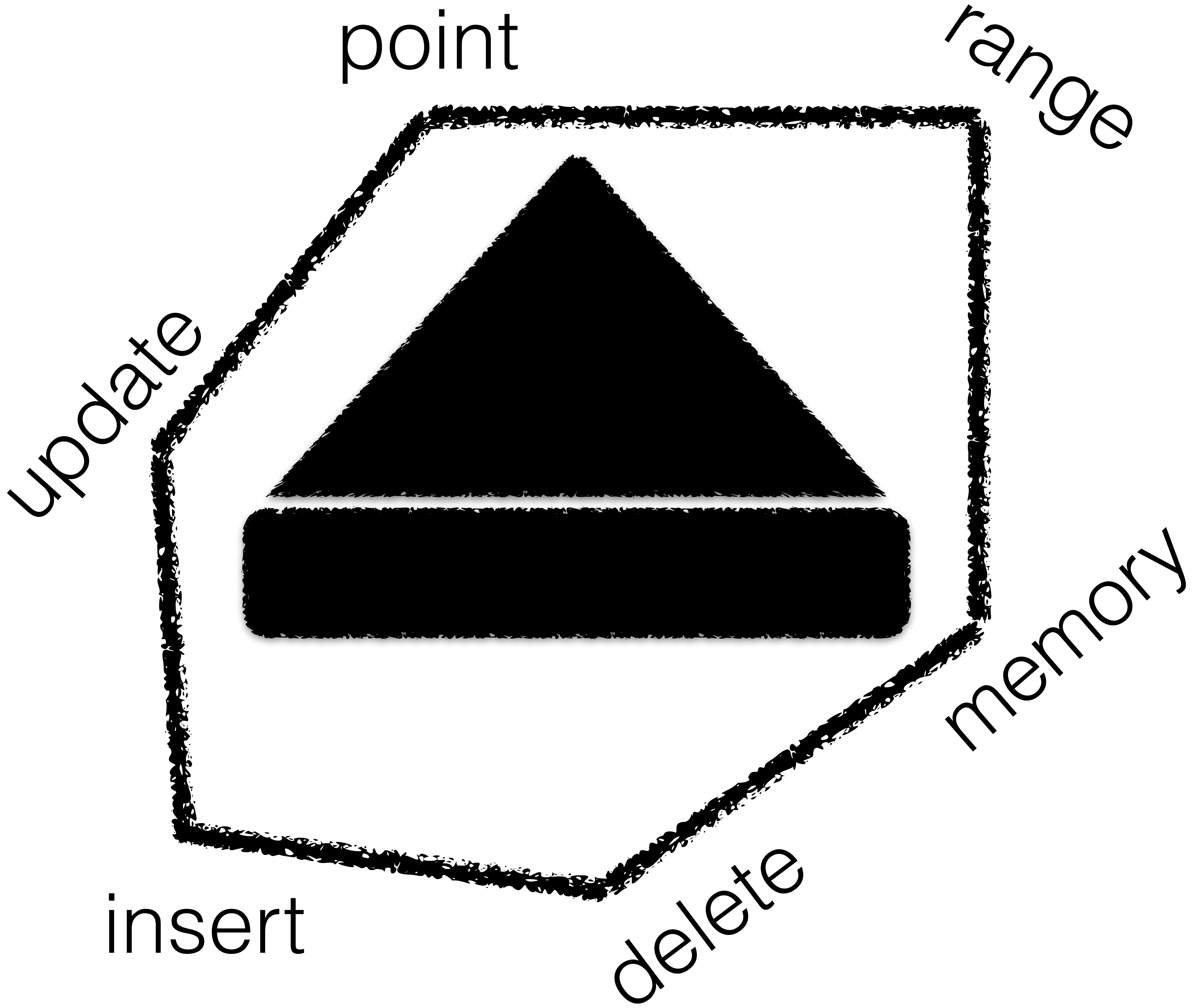


algorithm & cost synthesis



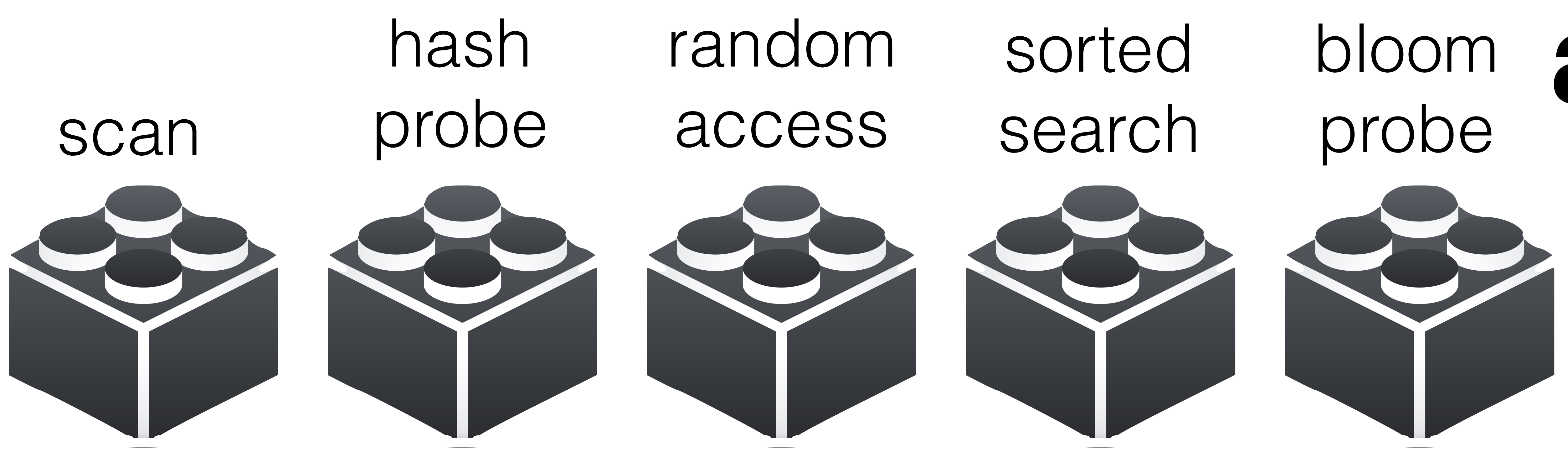
synthesize from first principles



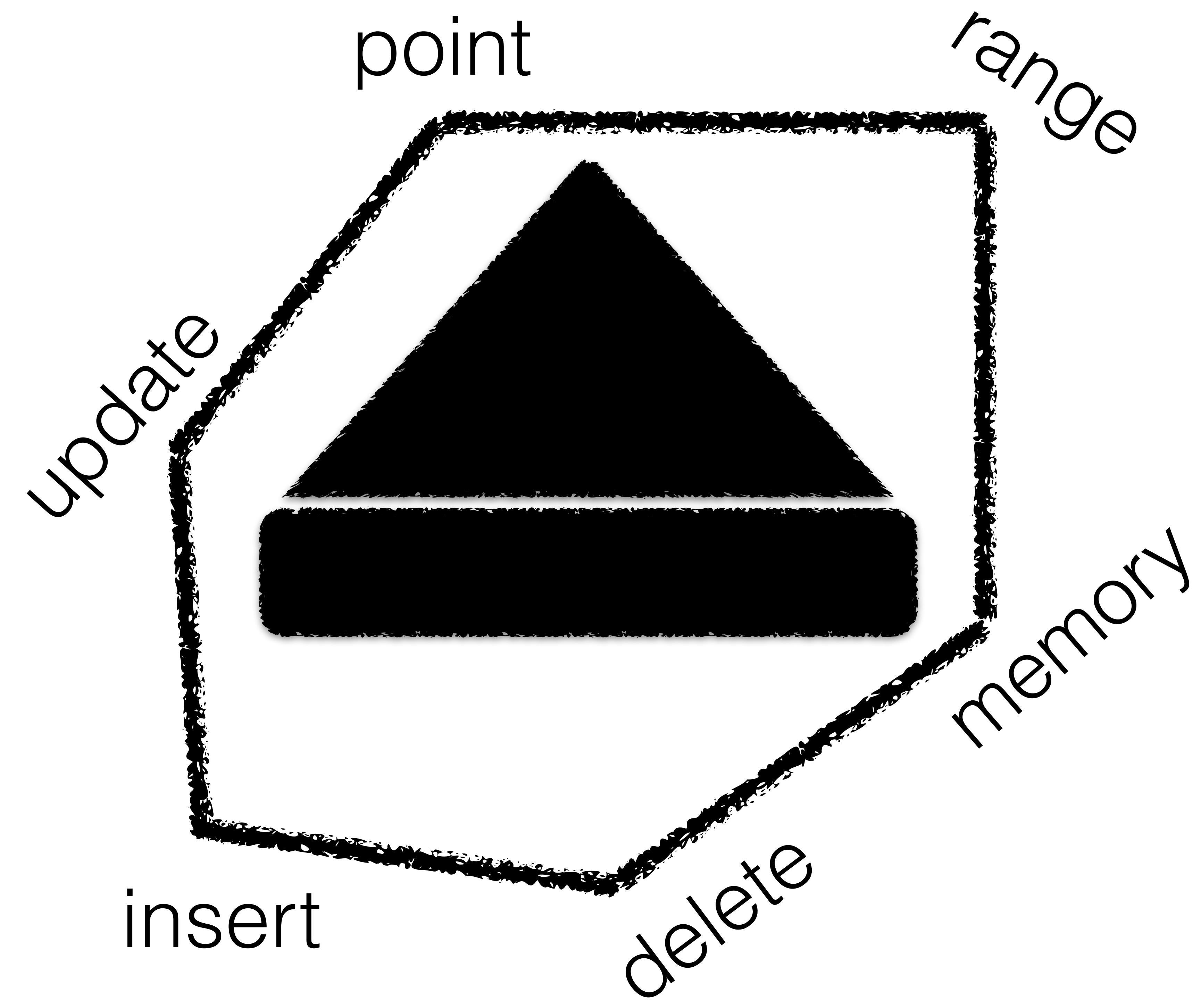


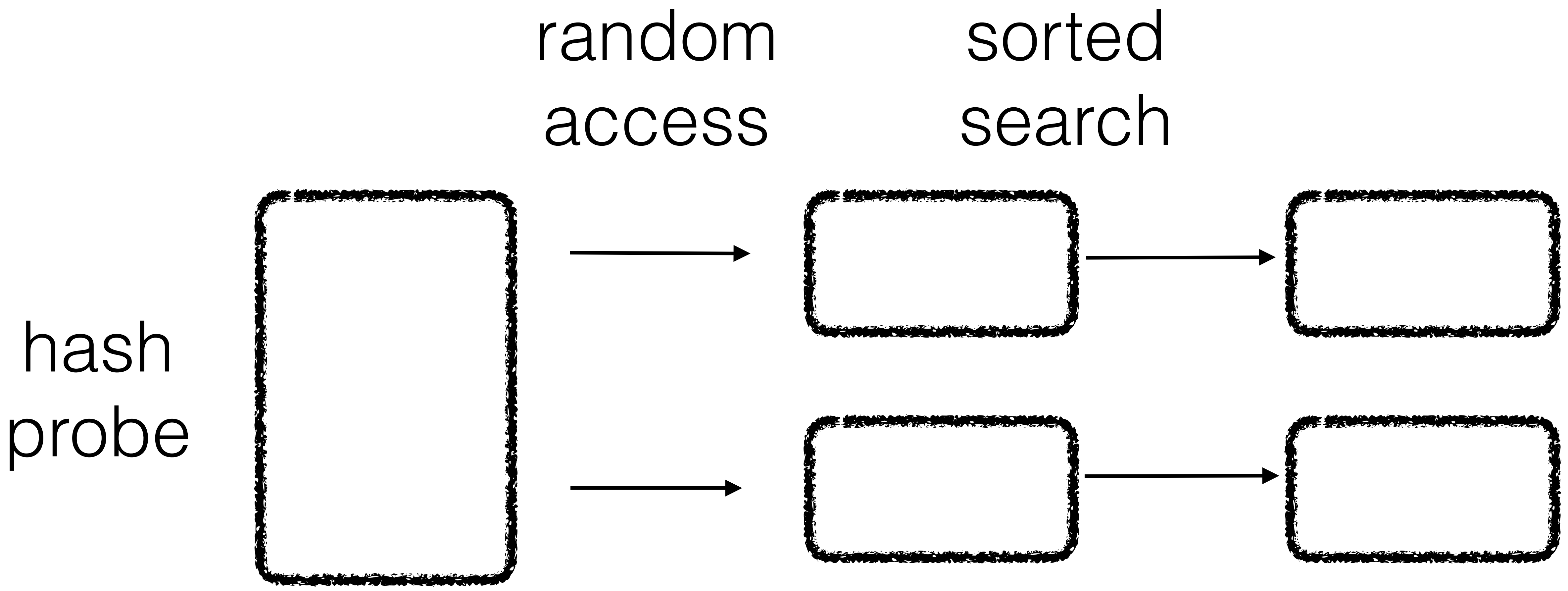
DASlab
@ Harvard SEAS

algorithm & cost synthesis



synthesize from first principles



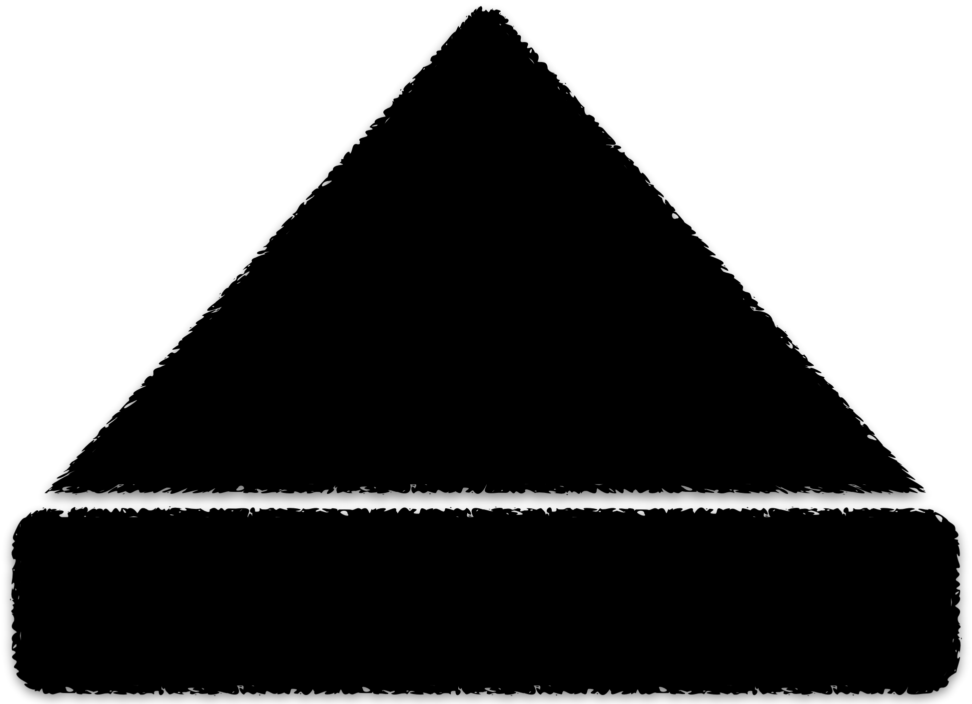


DASlab @ Harvard SEAS

algorithm & cost synthesis

scan | hash probe | random access | sorted search | bloom probe

synthesize from first principles

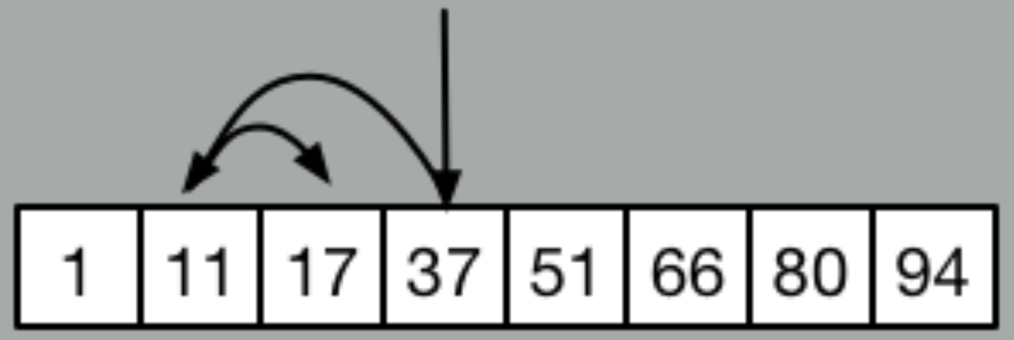


1. MINIMAL CODE

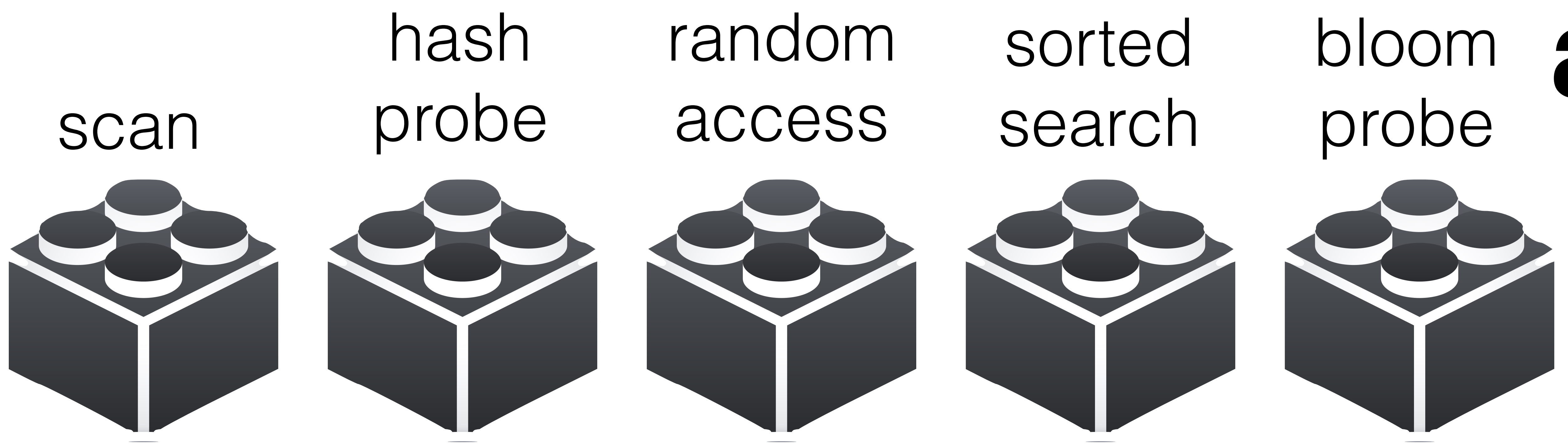
e.g., binary search

C++

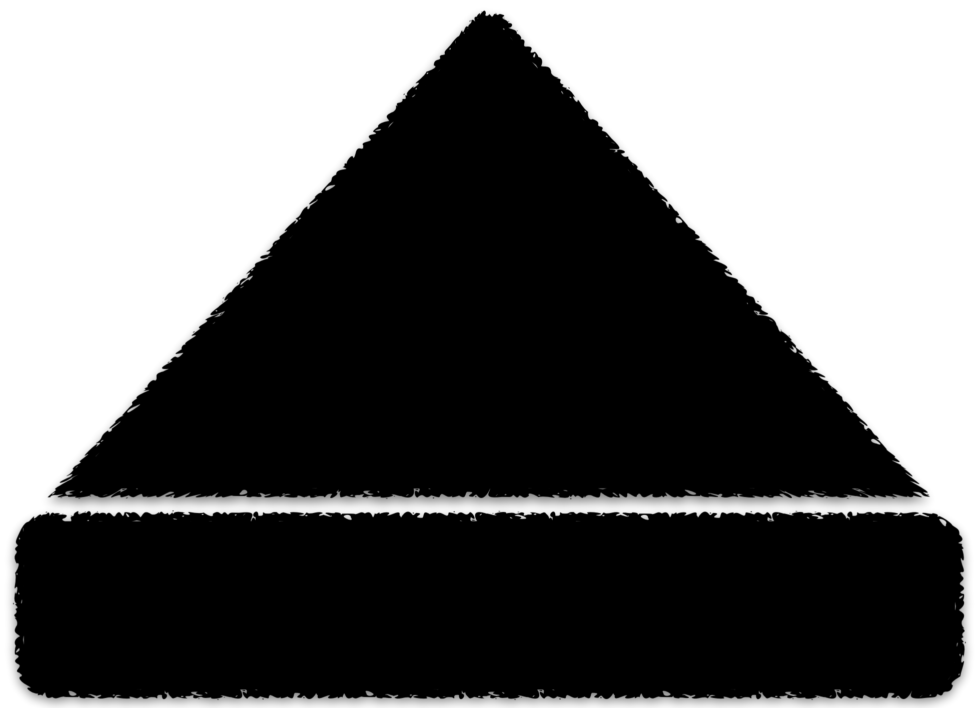
```
if (data[middle] < search_val) {
    low = middle + 1;
} else {
    high = middle;
}
middle = (low + high)/2;
```



algorithm & cost synthesis



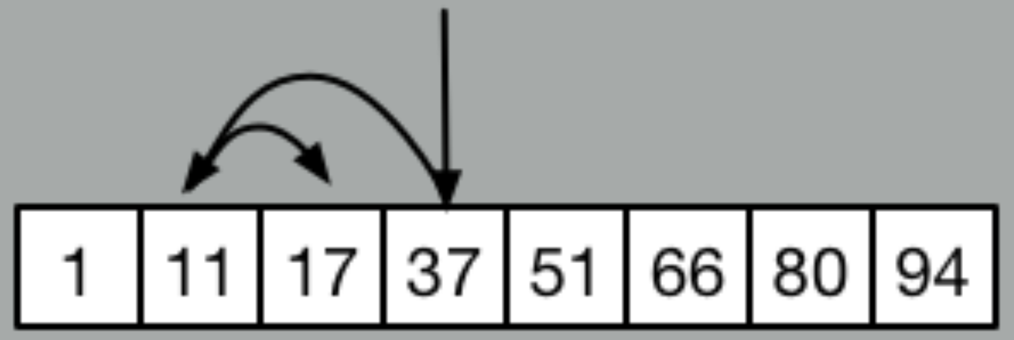
synthesize from first principles

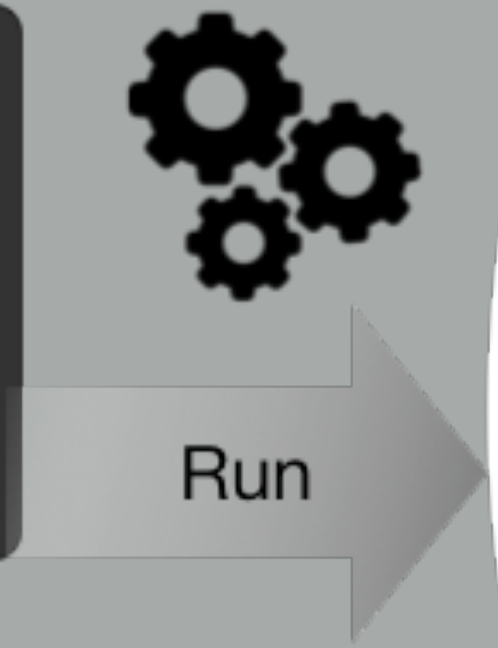


1. MINIMAL CODE

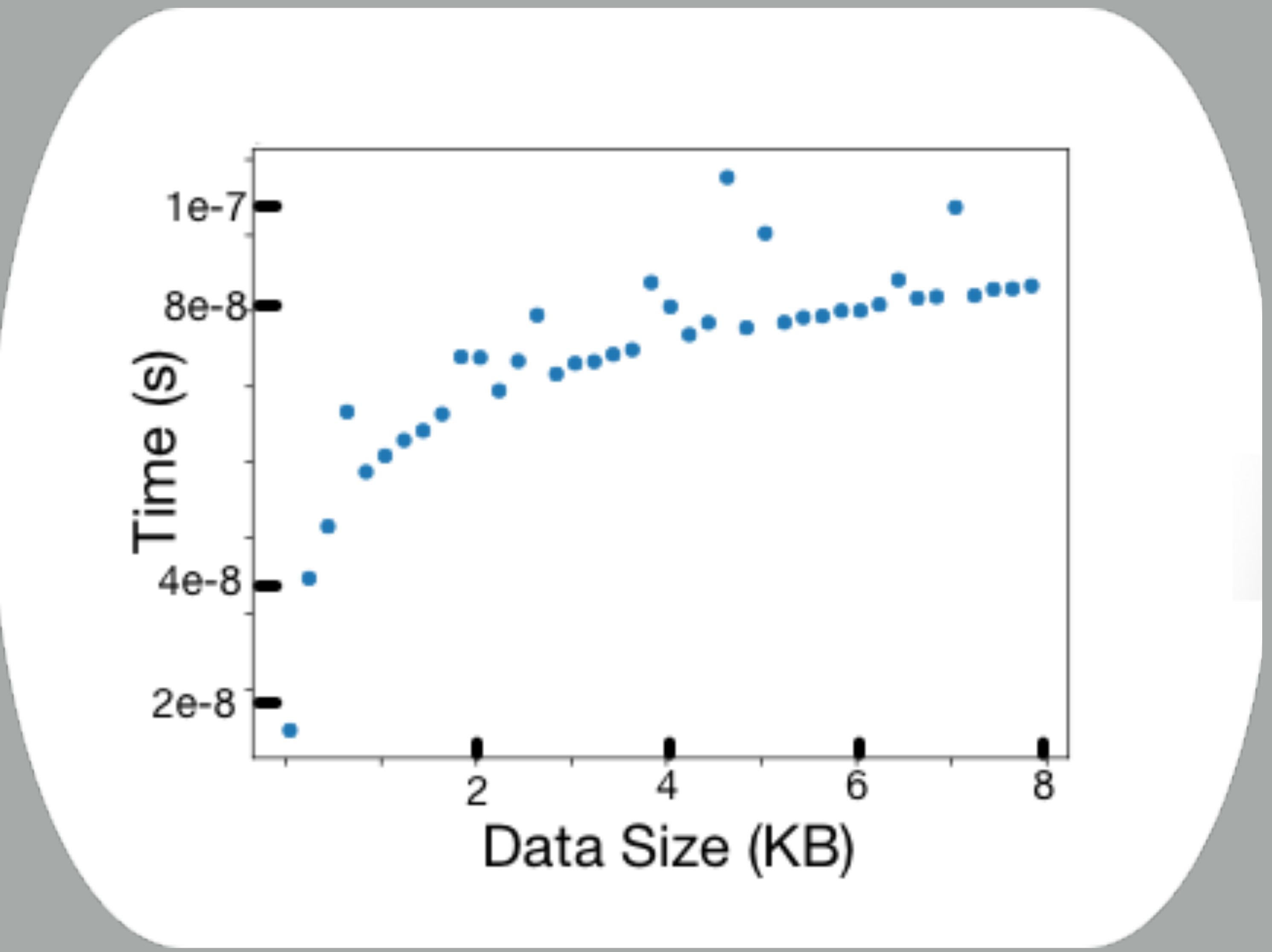
e.g., binary search

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

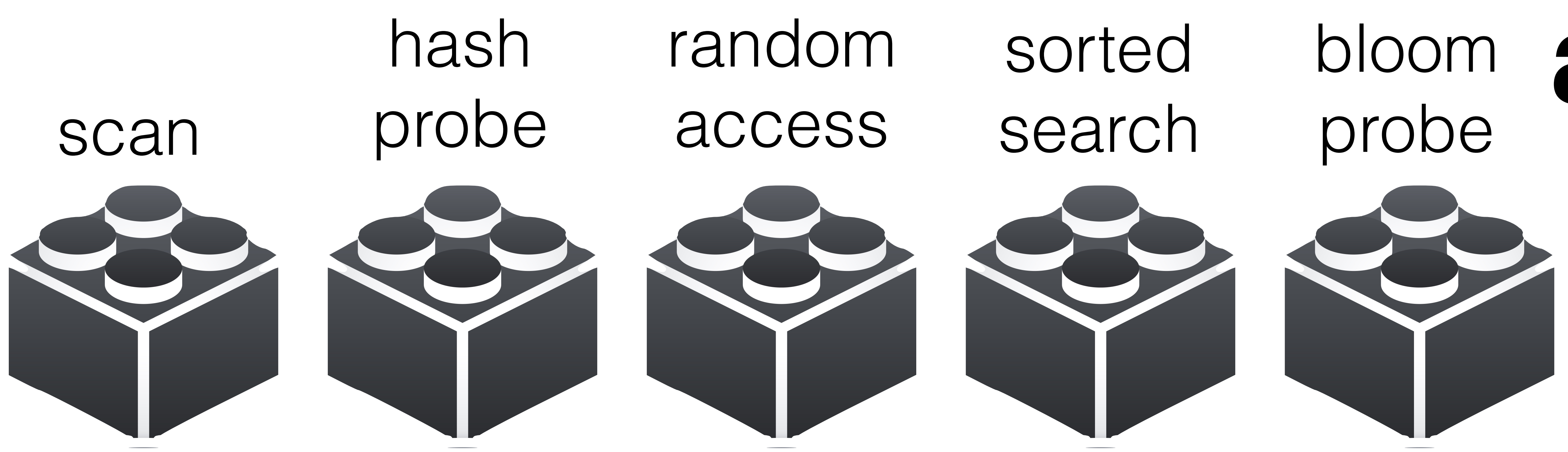




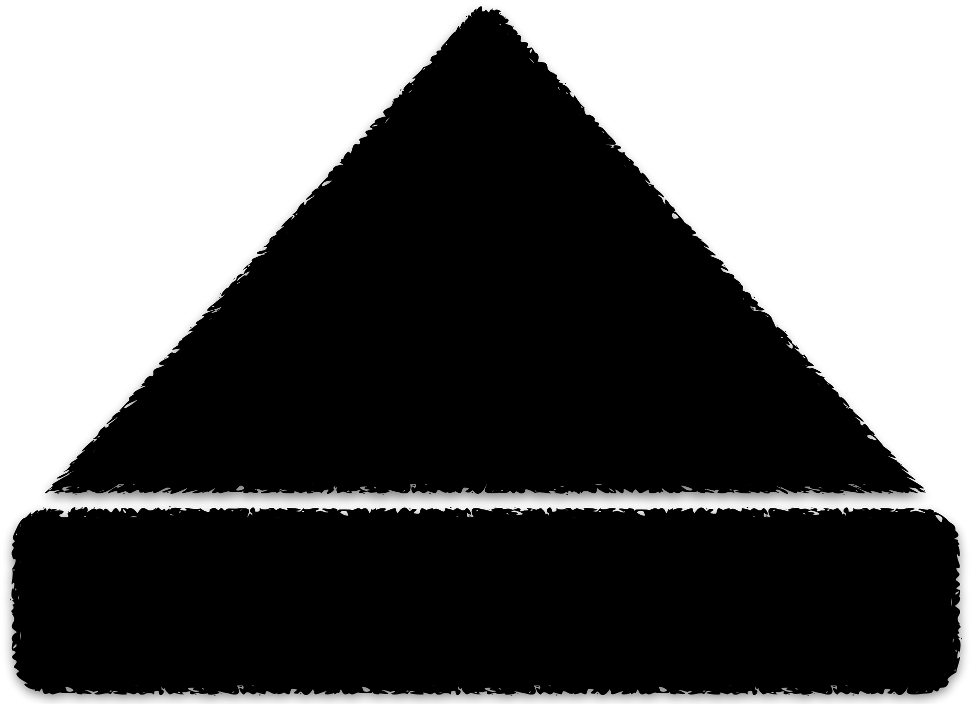
2. BENCHMARK



algorithm & cost synthesis



synthesize from first principles



algorithm & cost synthesis

Learned Cost Models @SIGMOD2018

unified design
storage engine template

unified design
storage engine template

DASlab @ Harvard SEAS

unified design
storage engine template

DASlab
@ Harvard SEAS

unified design
storage engine template

DASlab @ Harvard SEAS

unified design

storage engine template

DASlab @ Harvard SEAS

unified design
storage engine template

DASlab @ Harvard SEAS

unified design
storage engine template

DASlab @ Harvard SEAS

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

learned CPU model

cloud cost mapping & SLA

unified closed form

h/w, parallelism

AWS, Azure, Google

workload
budget
perf.

analytical I/O model

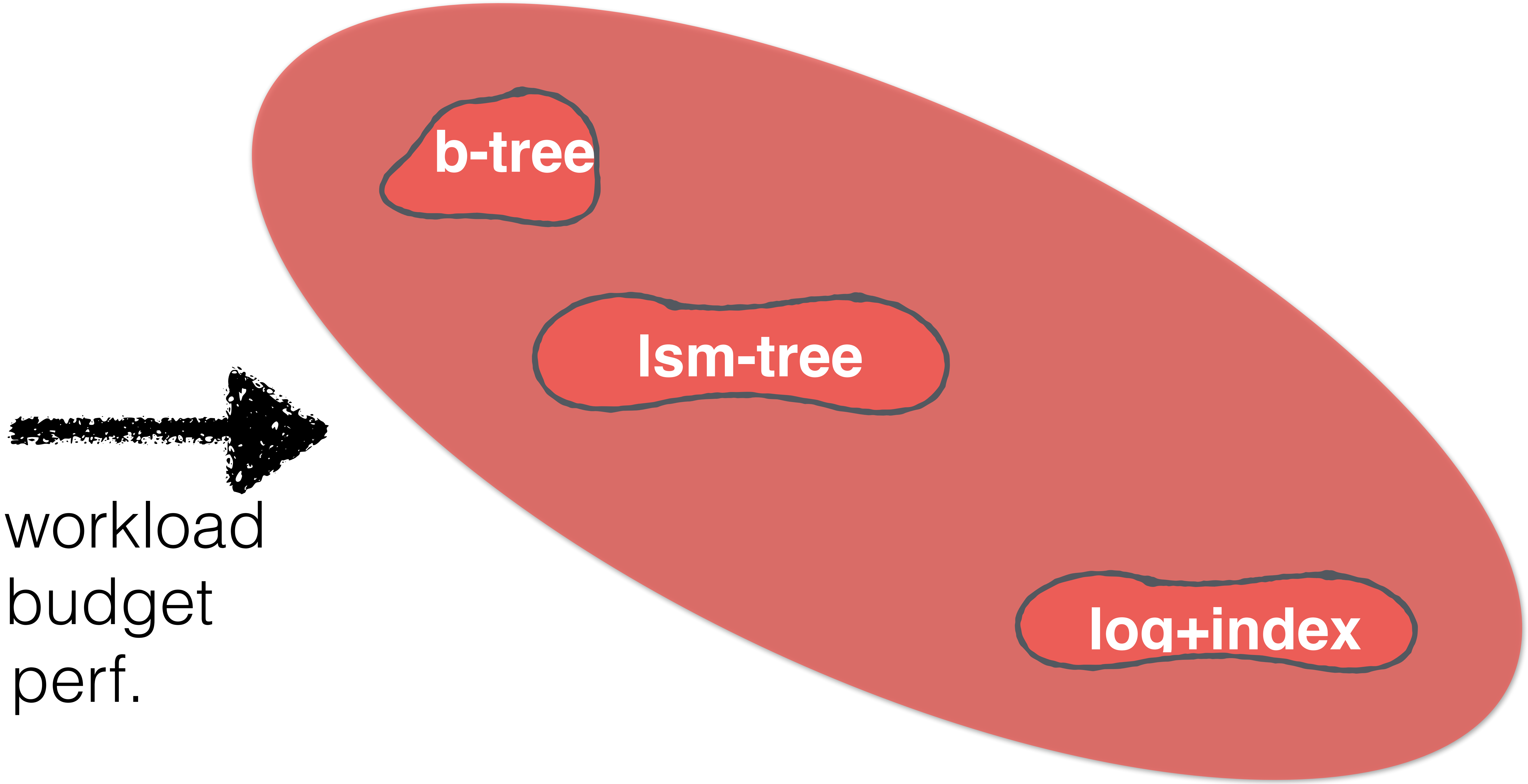
unified
closed form

learned CPU model

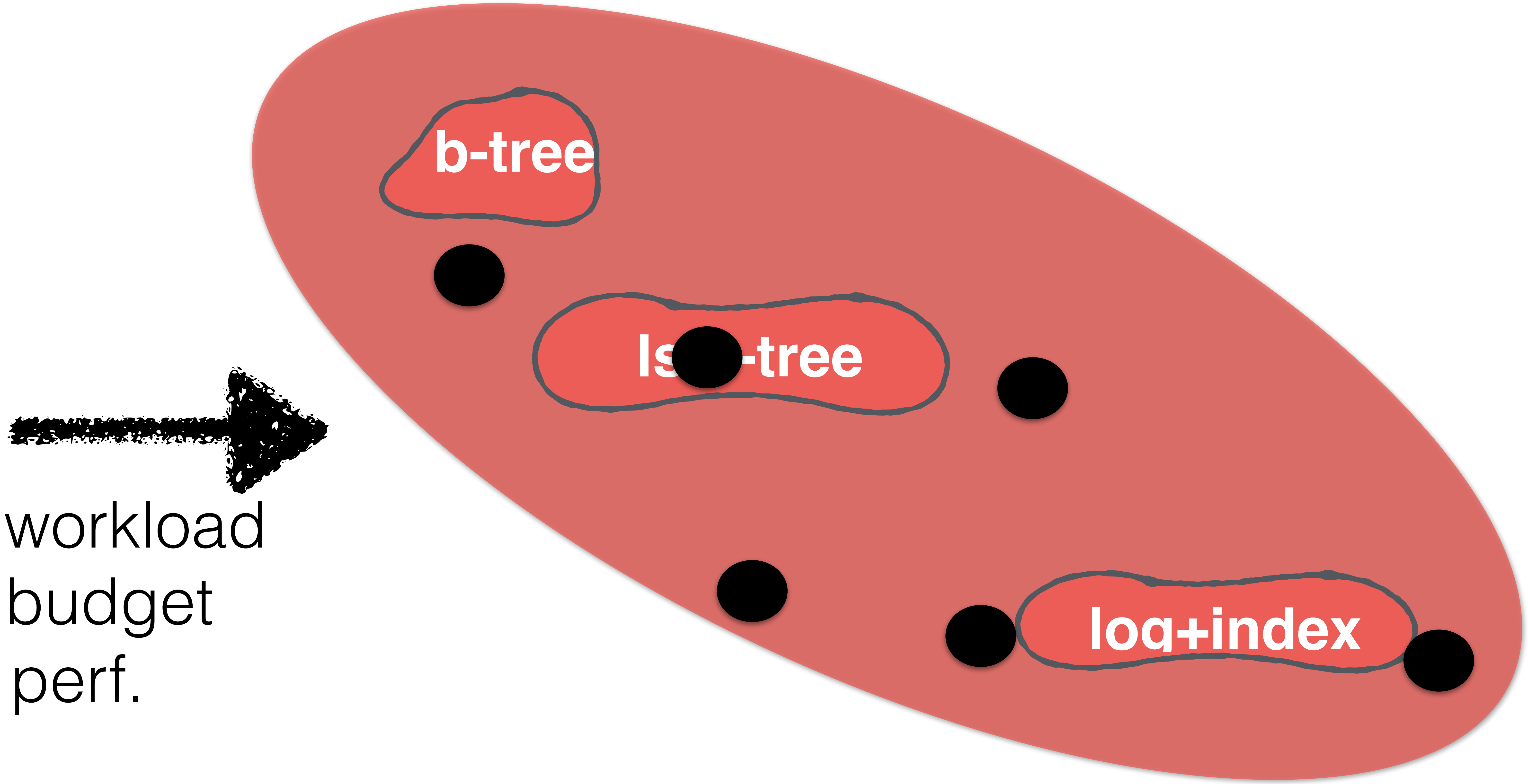
h/w,
parallelism

cloud cost mapping & SLA

AWS, Azure, Google



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



analytical I/O model

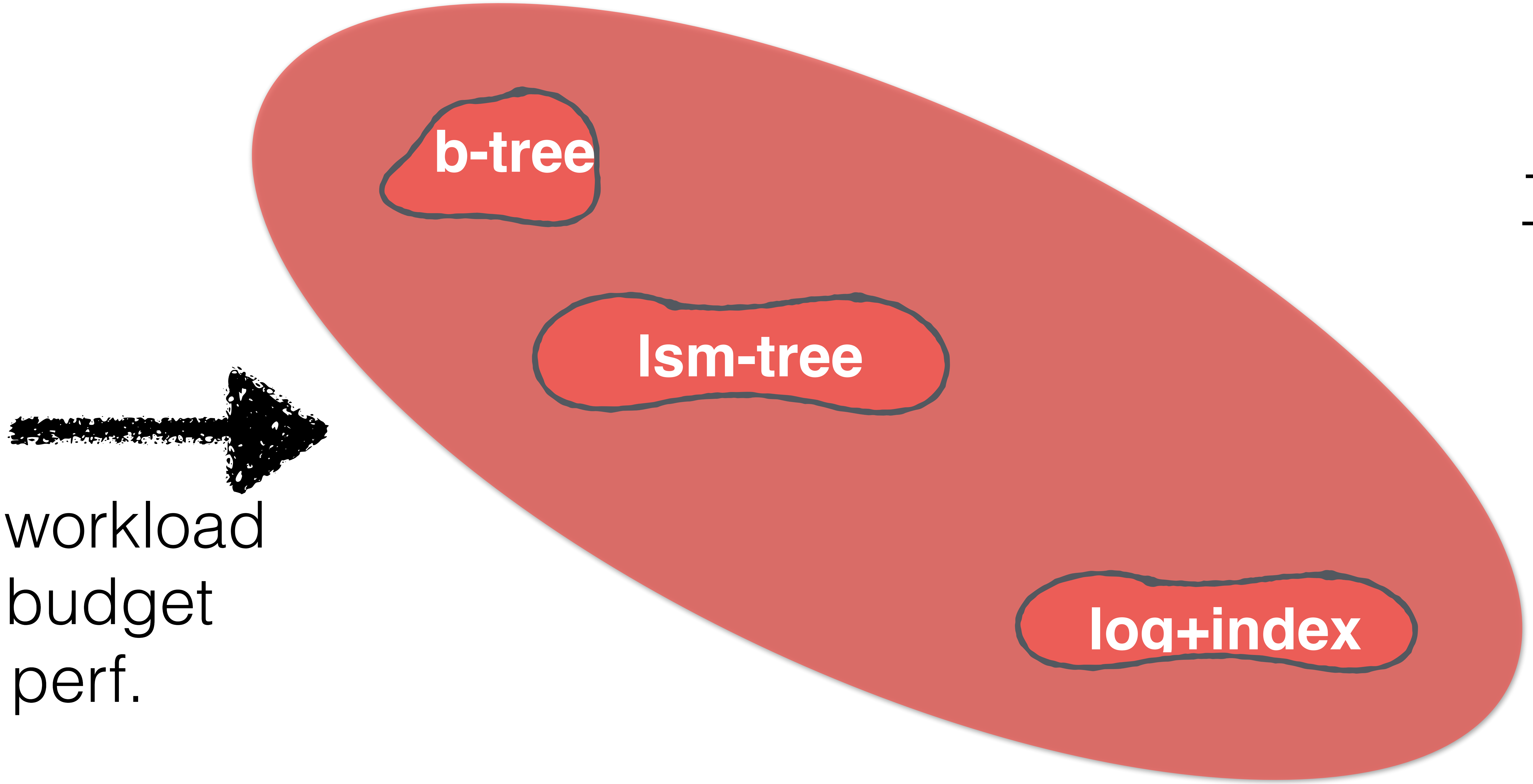
unified closed form

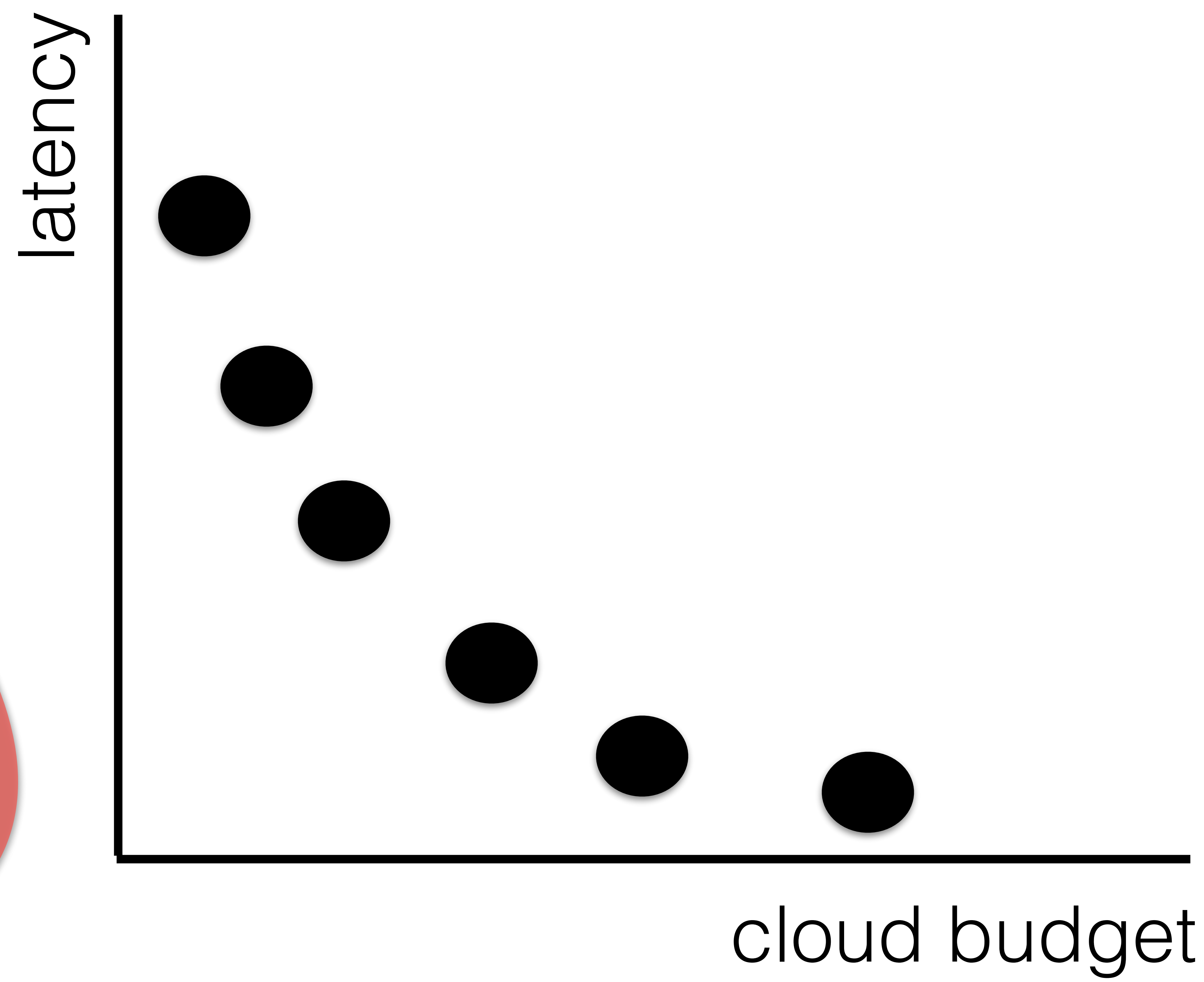
learned CPU model

h/w, parallelism

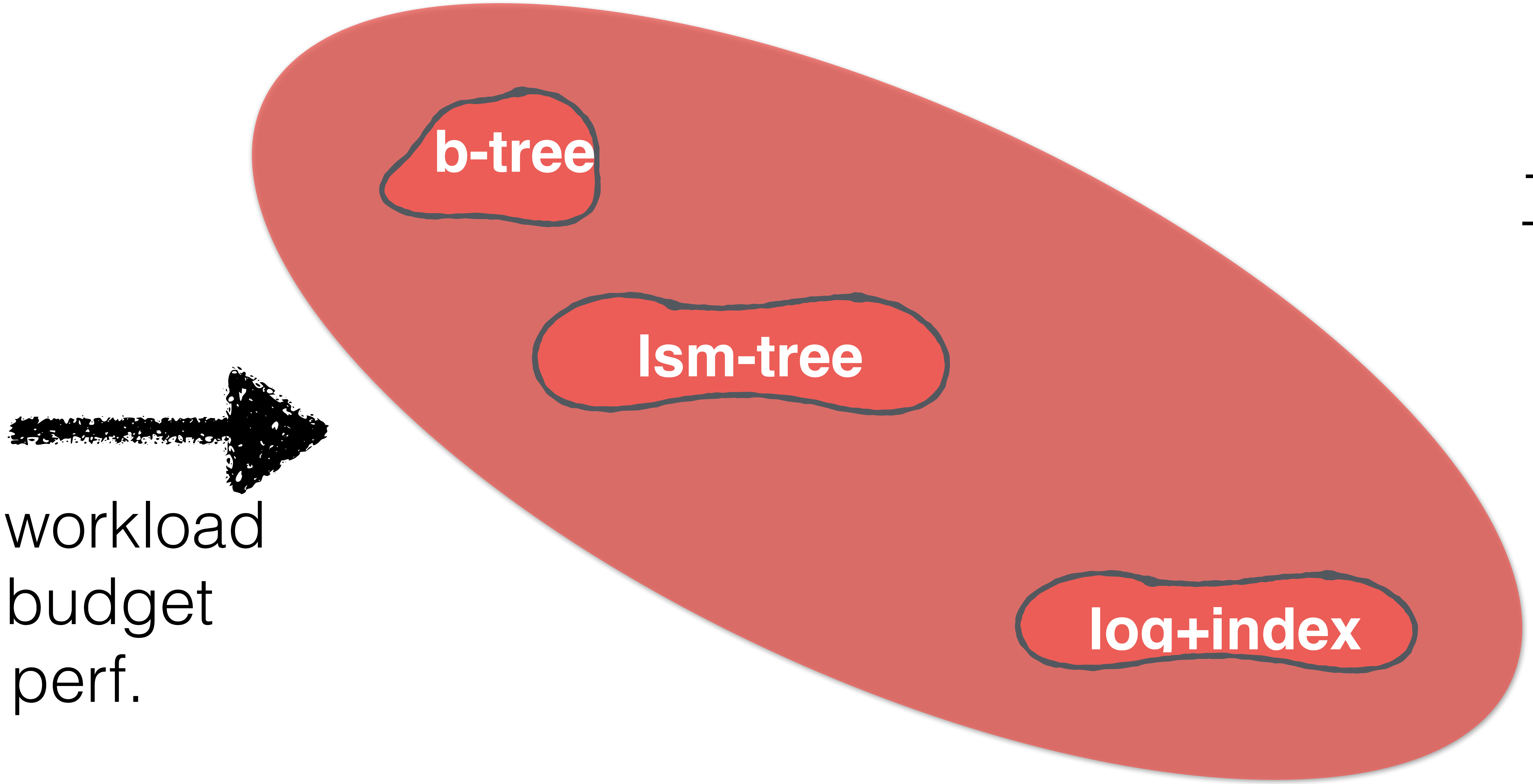
cloud cost mapping & SLA

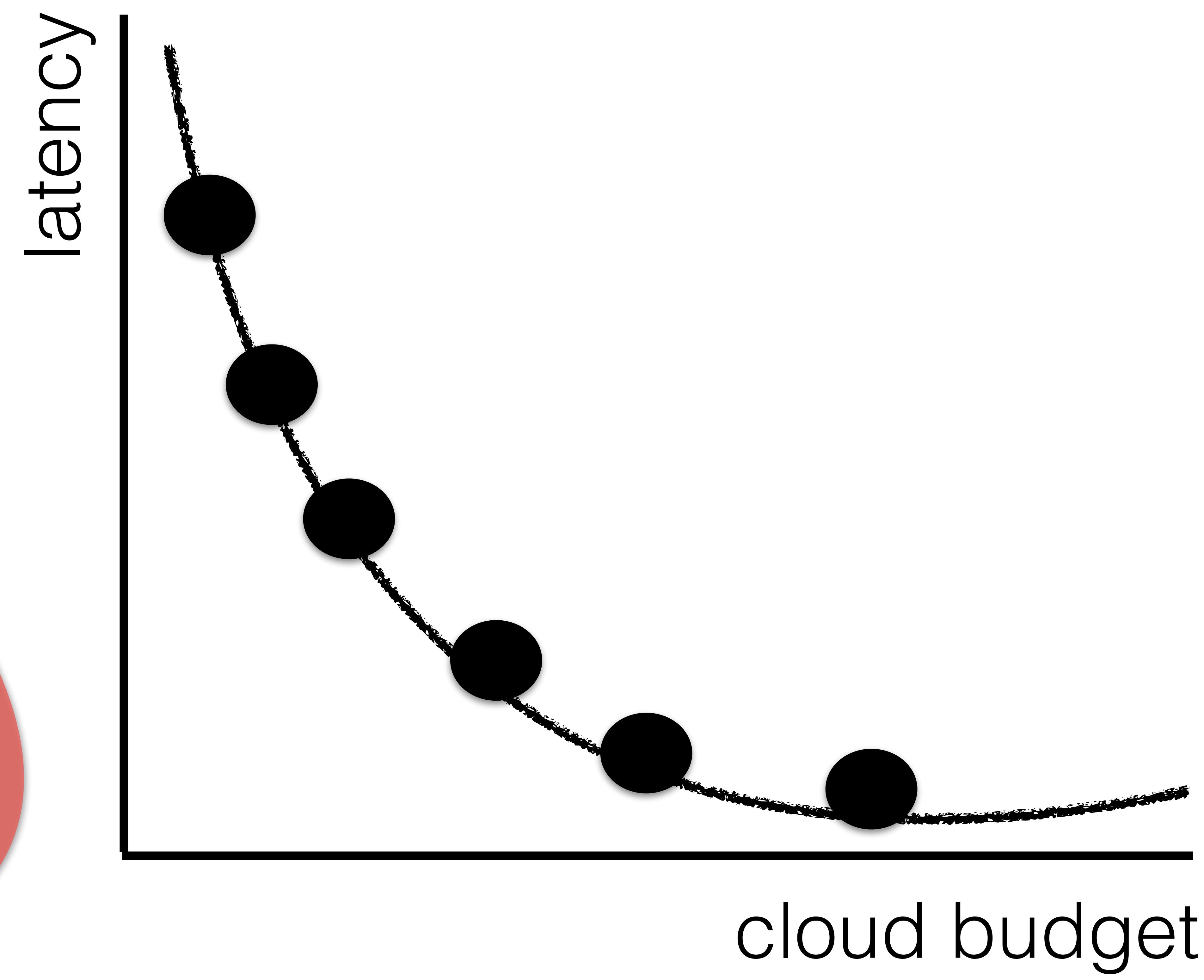
AWS, Azure, Google



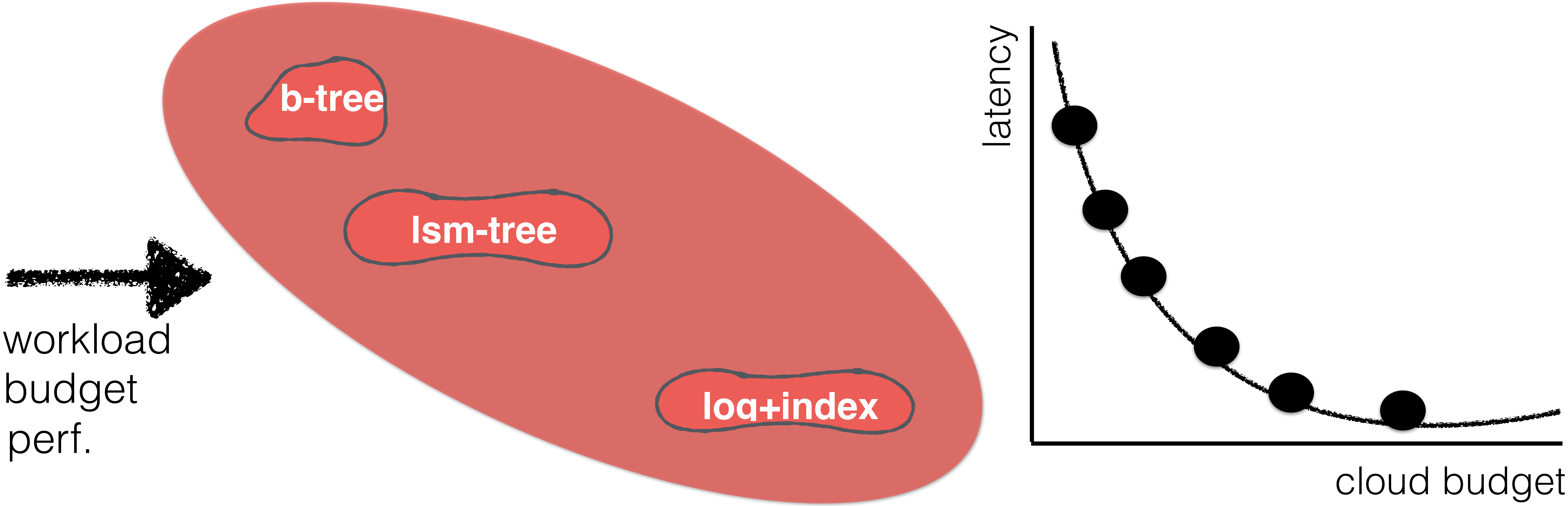


analytical I/O model	learned CPU model	cloud cost mapping & SLA
unified closed form	h/w, parallelism	AWS, Azure, Google





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



analytical I/O model

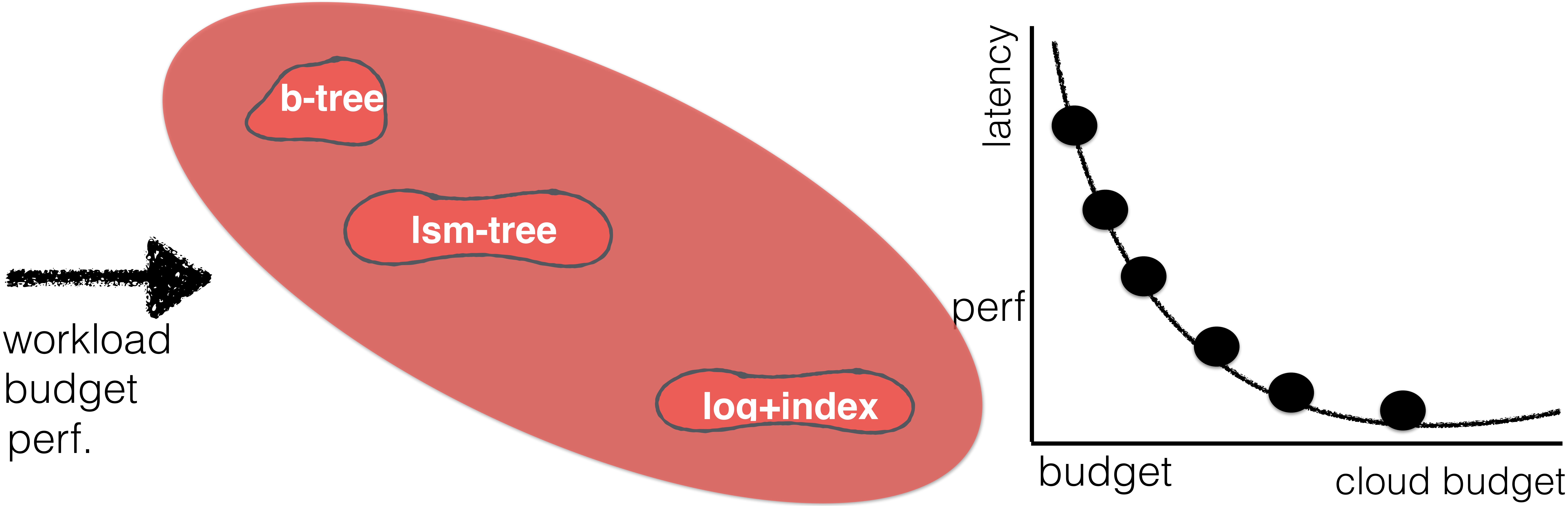
unified closed form

learned CPU model

h/w, parallelism

cloud cost mapping & SLA

AWS, Azure, Google



analytical I/O model

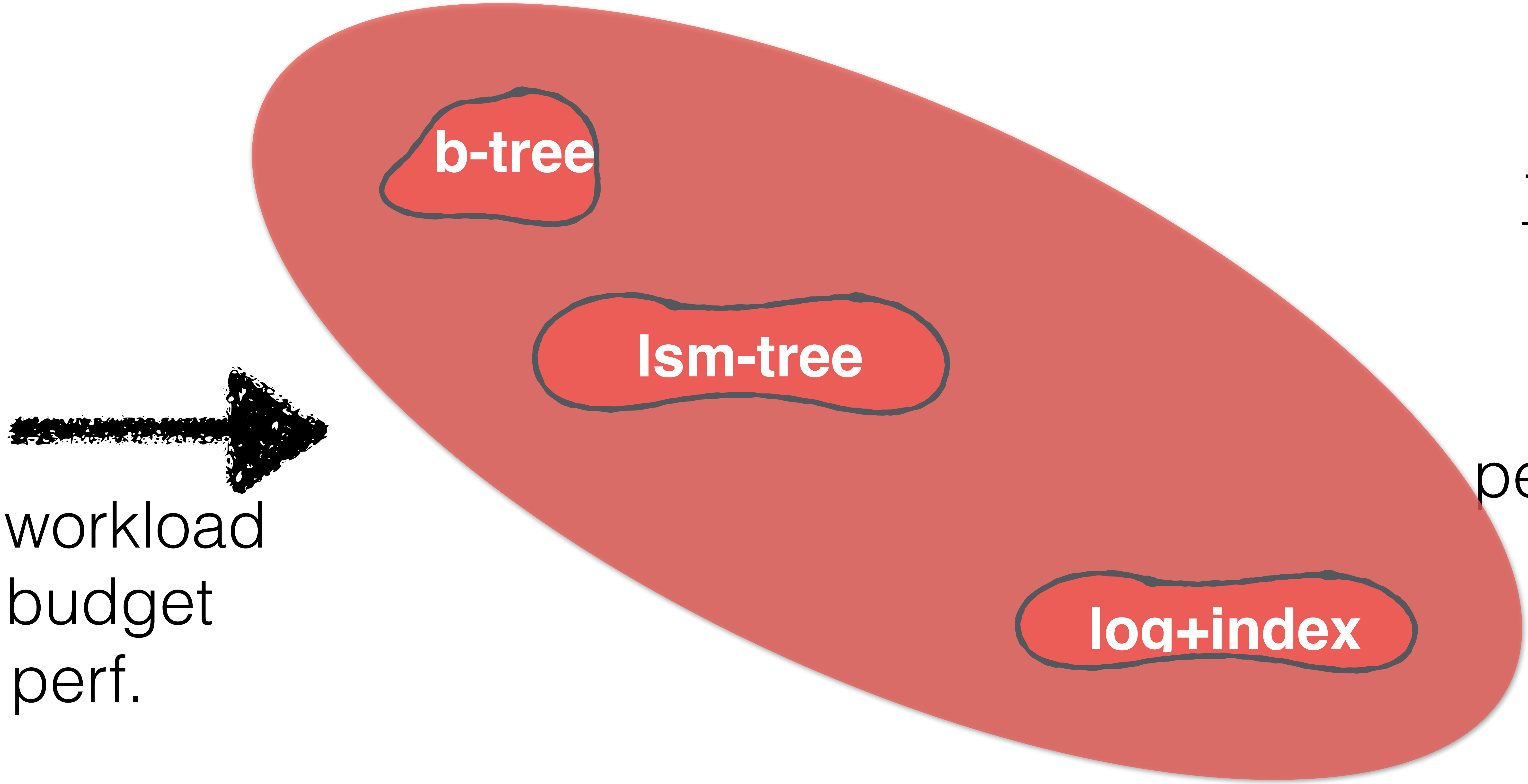
unified
closed form

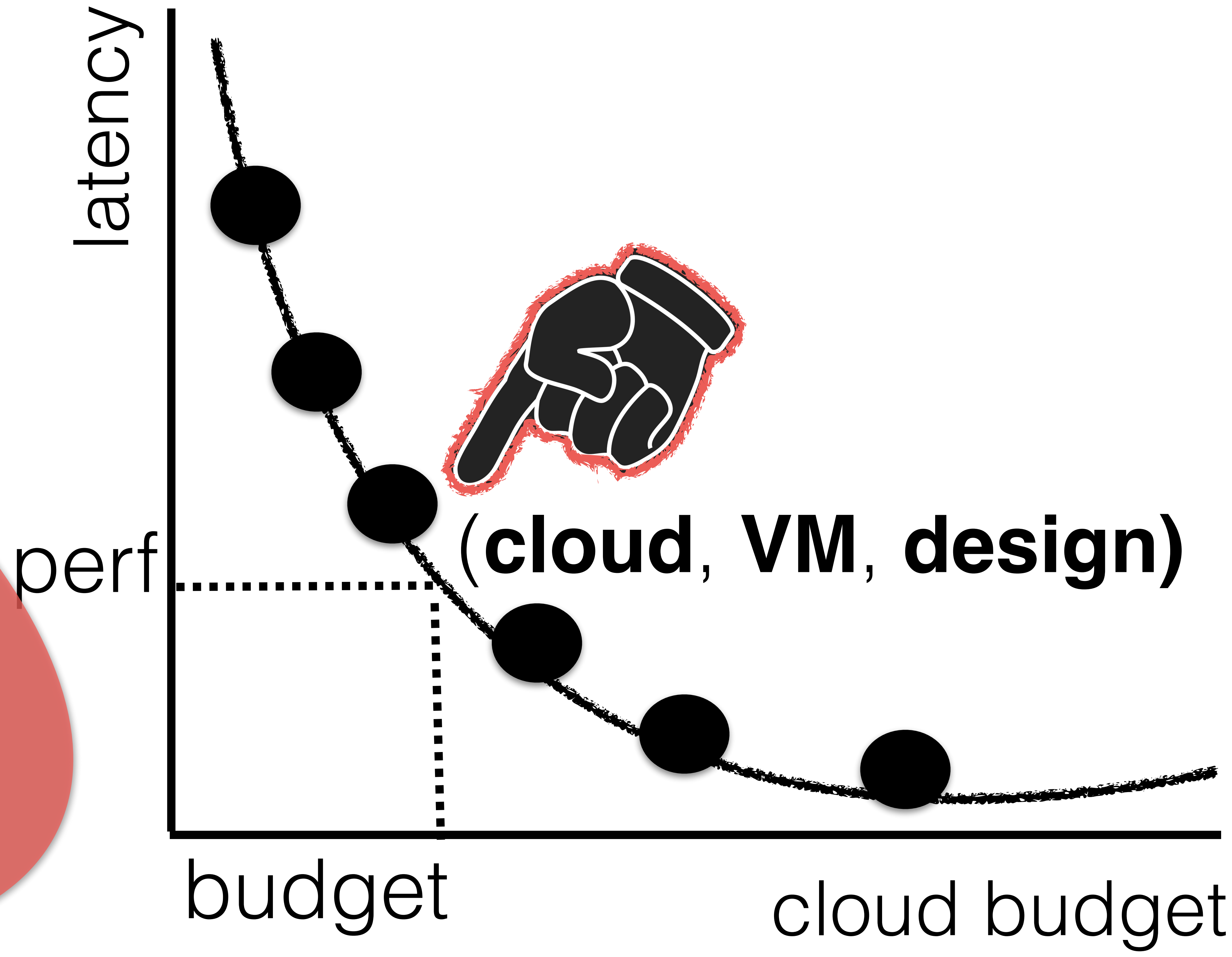
learned CPU model

h/w,
parallelism

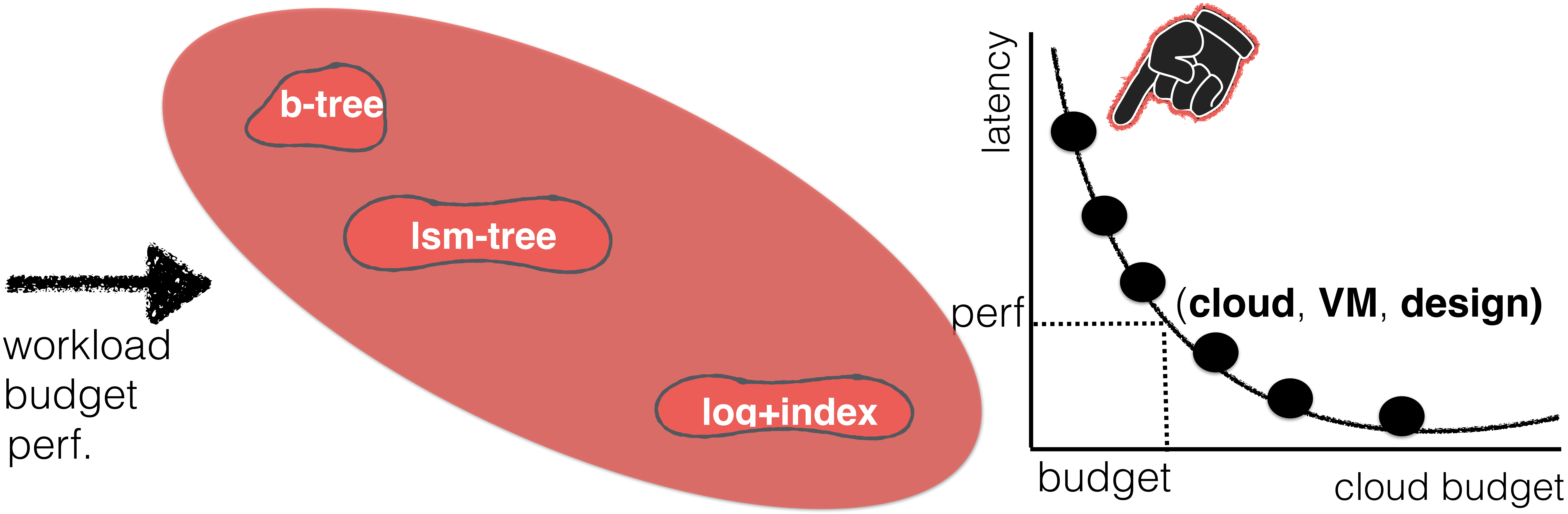
cloud cost mapping & SLA

AWS, Azure, Google





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



analytical I/O model

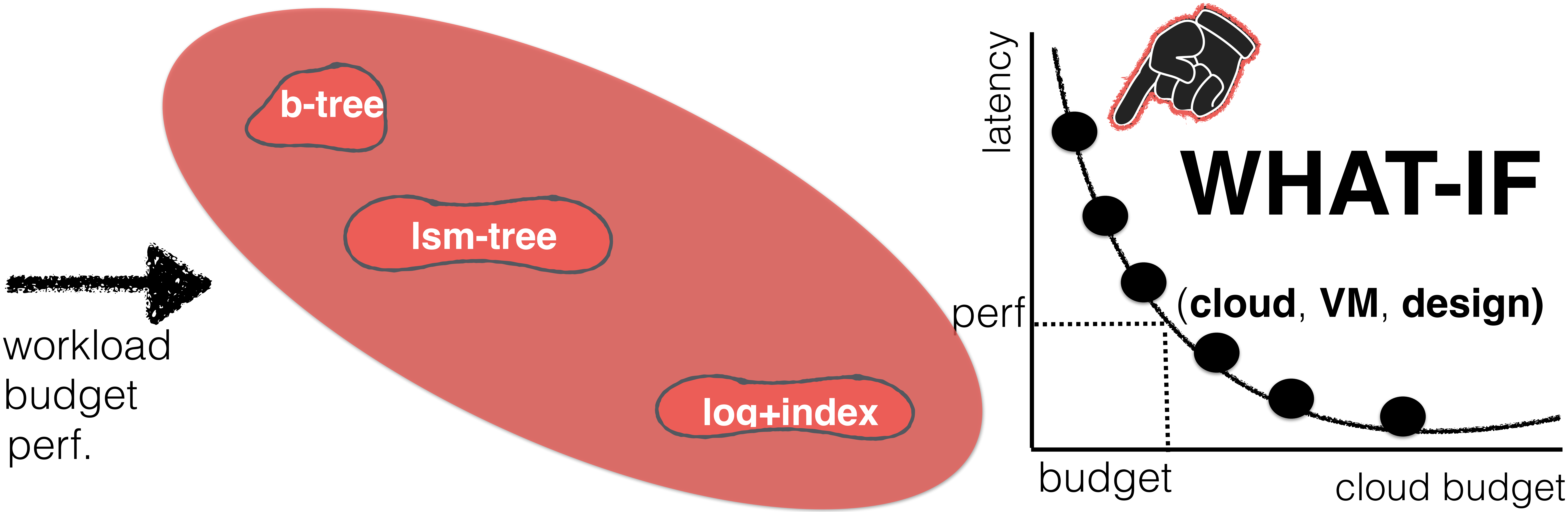
unified closed form

learned CPU model

h/w, parallelism

cloud cost mapping & SLA

AWS, Azure, Google



analytical I/O model

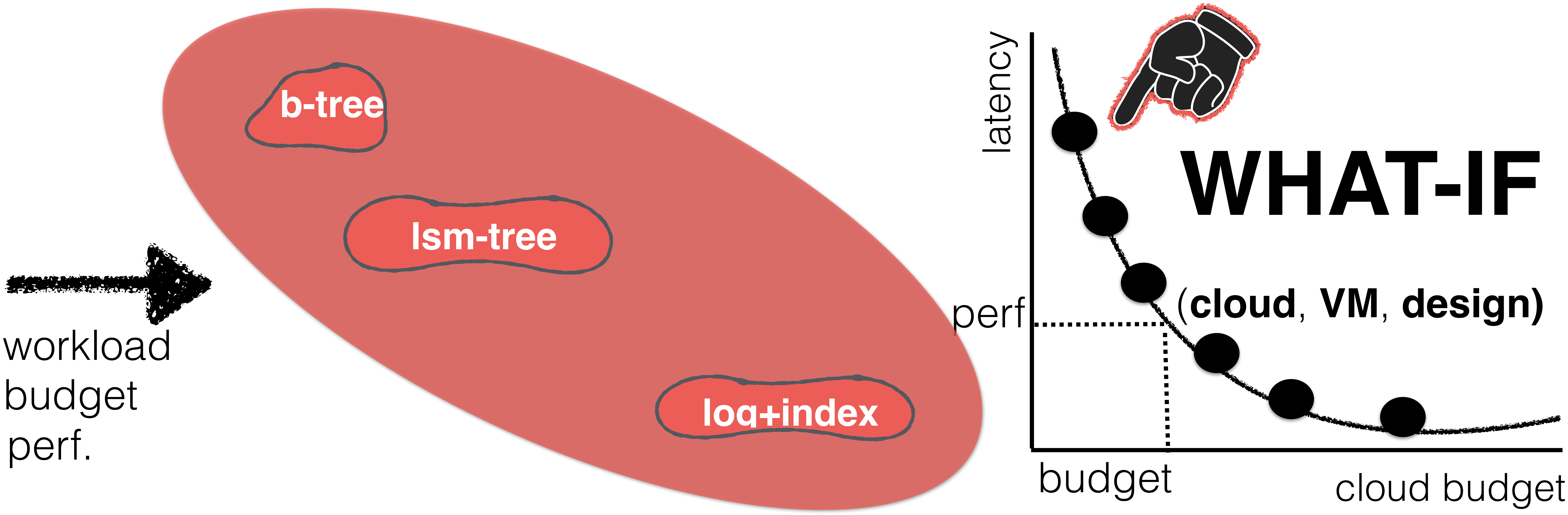
unified closed form

learned CPU model

h/w, parallelism

cloud cost mapping & SLA

AWS, Azure, Google



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

How to test?

workload/budget diversity

How to test? workload/budget diversity

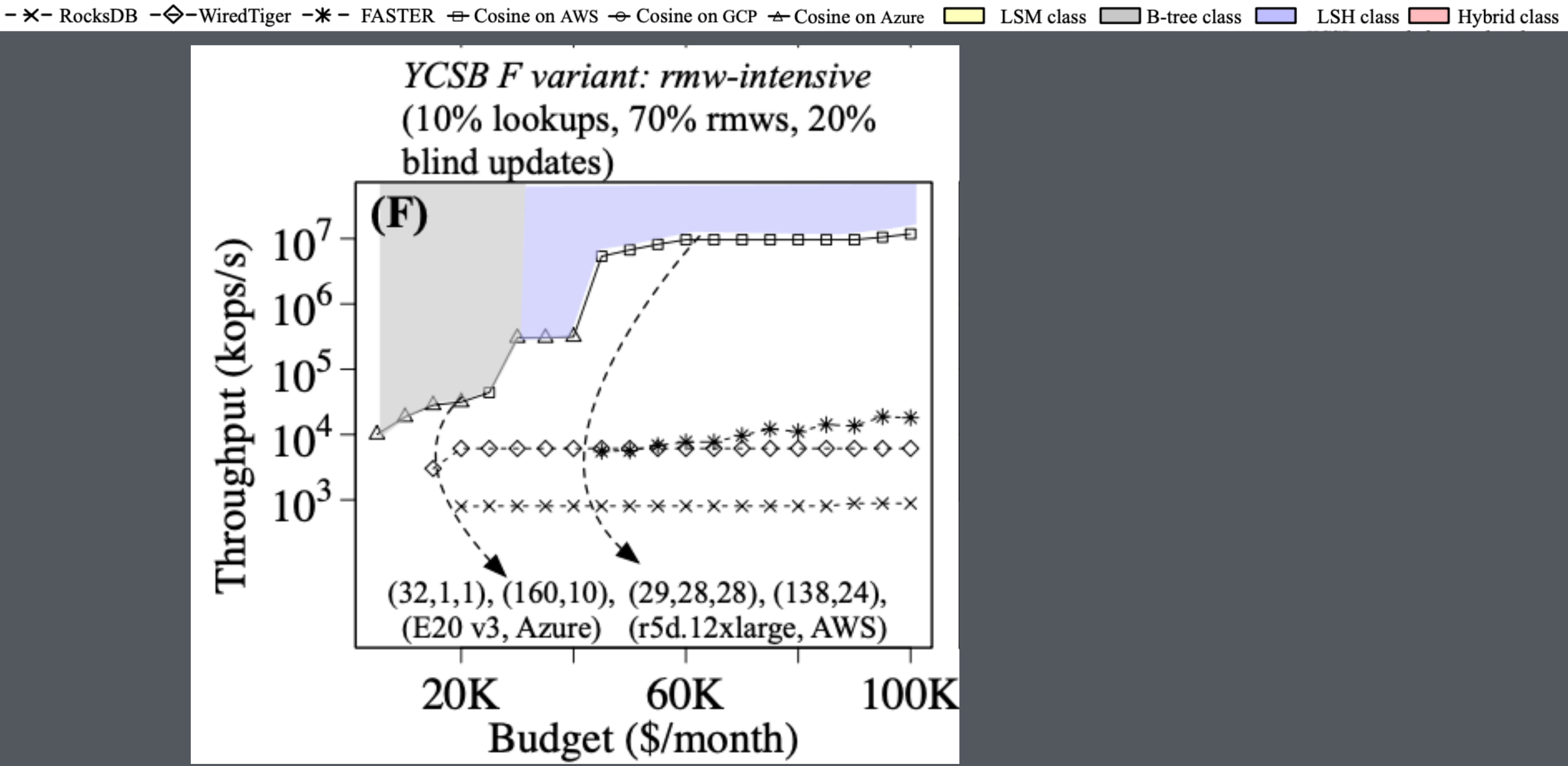
Can we beat the best system for every workload?

How to test? workload/budget diversity

Can we beat the best system for every workload?

workload/budget diversity

DASlab @ Harvard SEAS



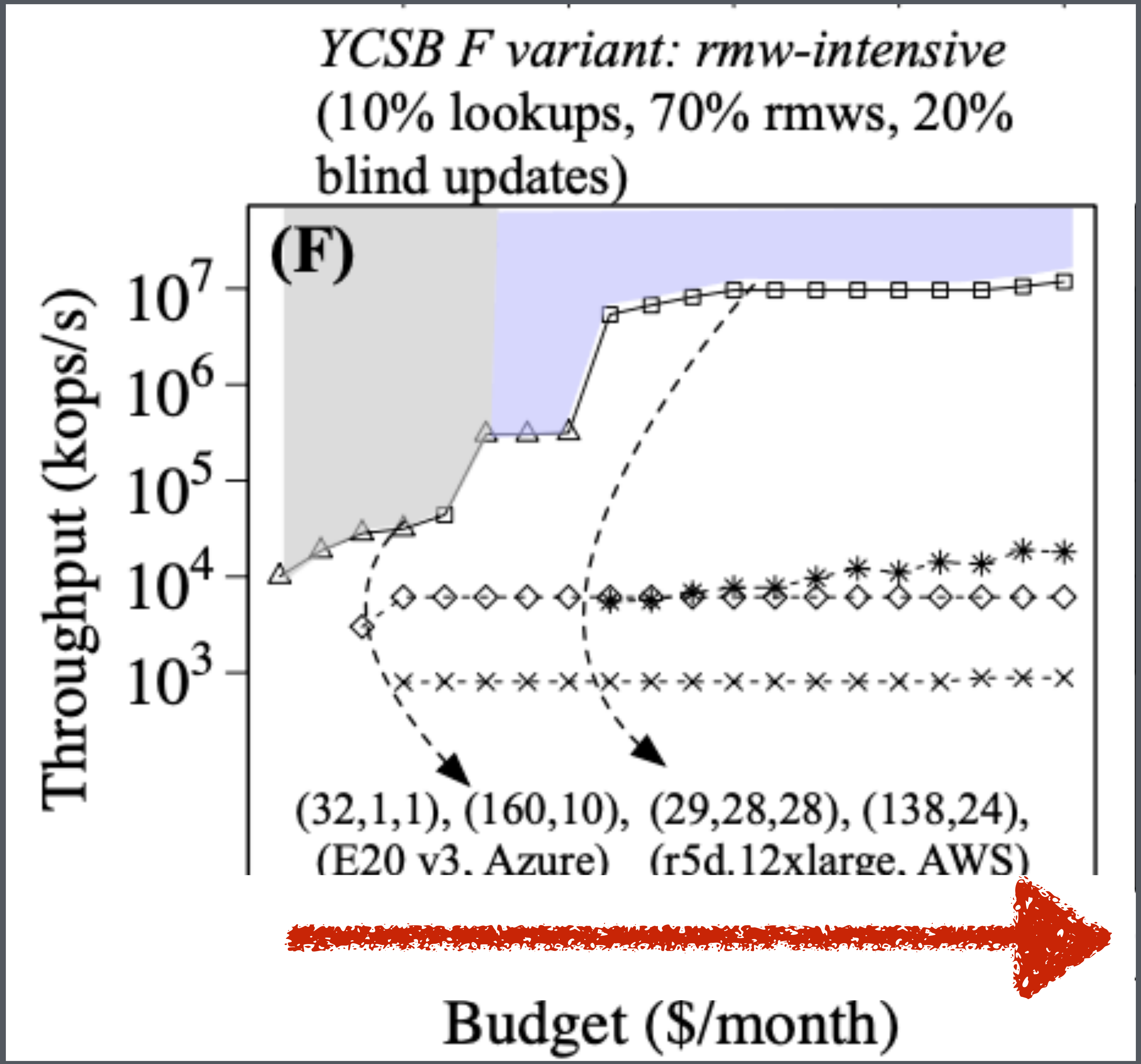
workload/budget diversity

DASlab @ Harvard SEAS

workload/budget diversity

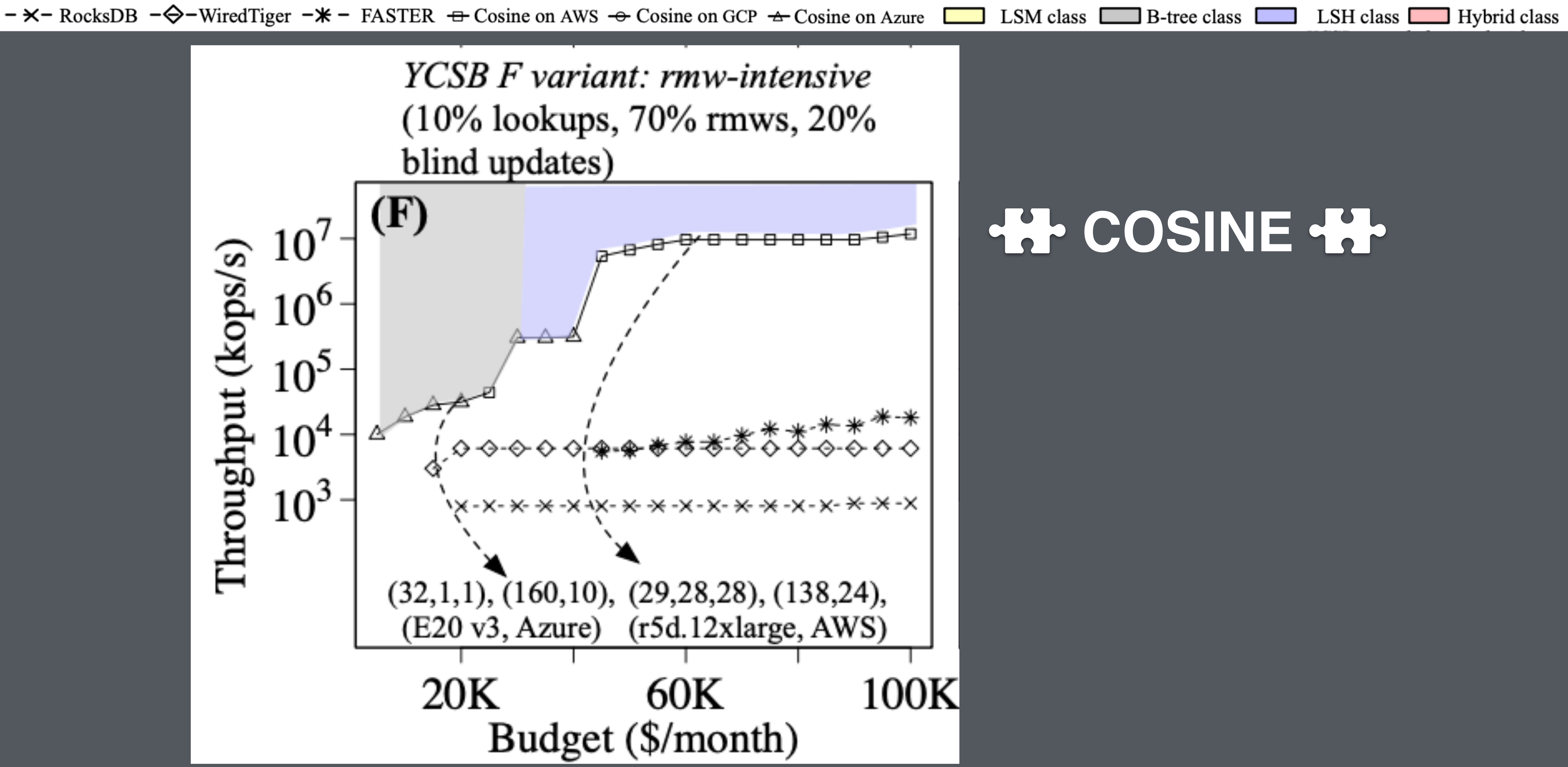
DASlab @ Harvard SEAS

-×- RocksDB -◇-WiredTiger -✳- FASTER -⊟- Cosine on AWS -⊖- Cosine on GCP -△- Cosine on Azure LSM class B-tree class LSH class Hybrid class



workload/budget diversity

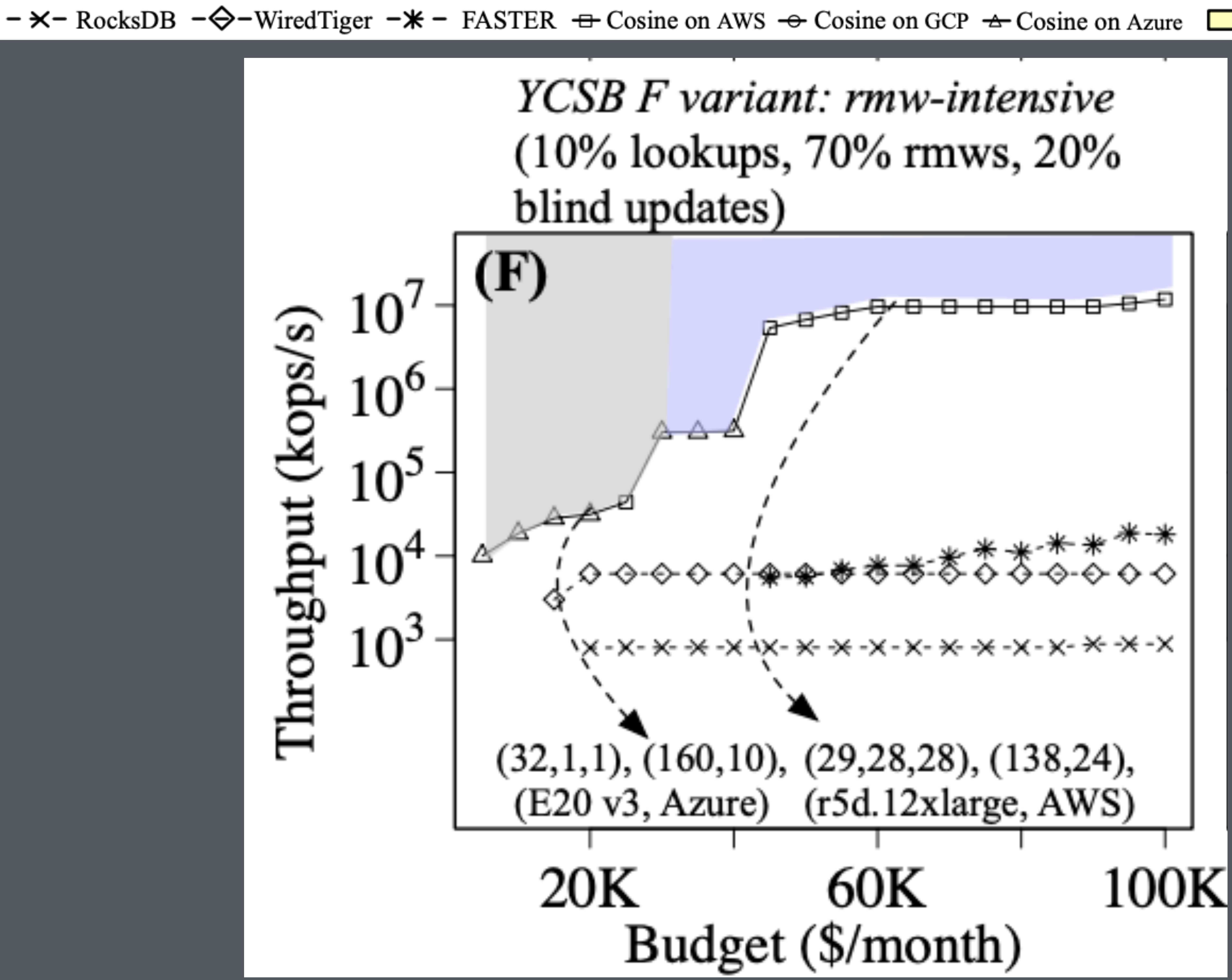
DASlab @ Harvard SEAS



COSINE

workload/budget diversity

DASlab @ Harvard SEAS

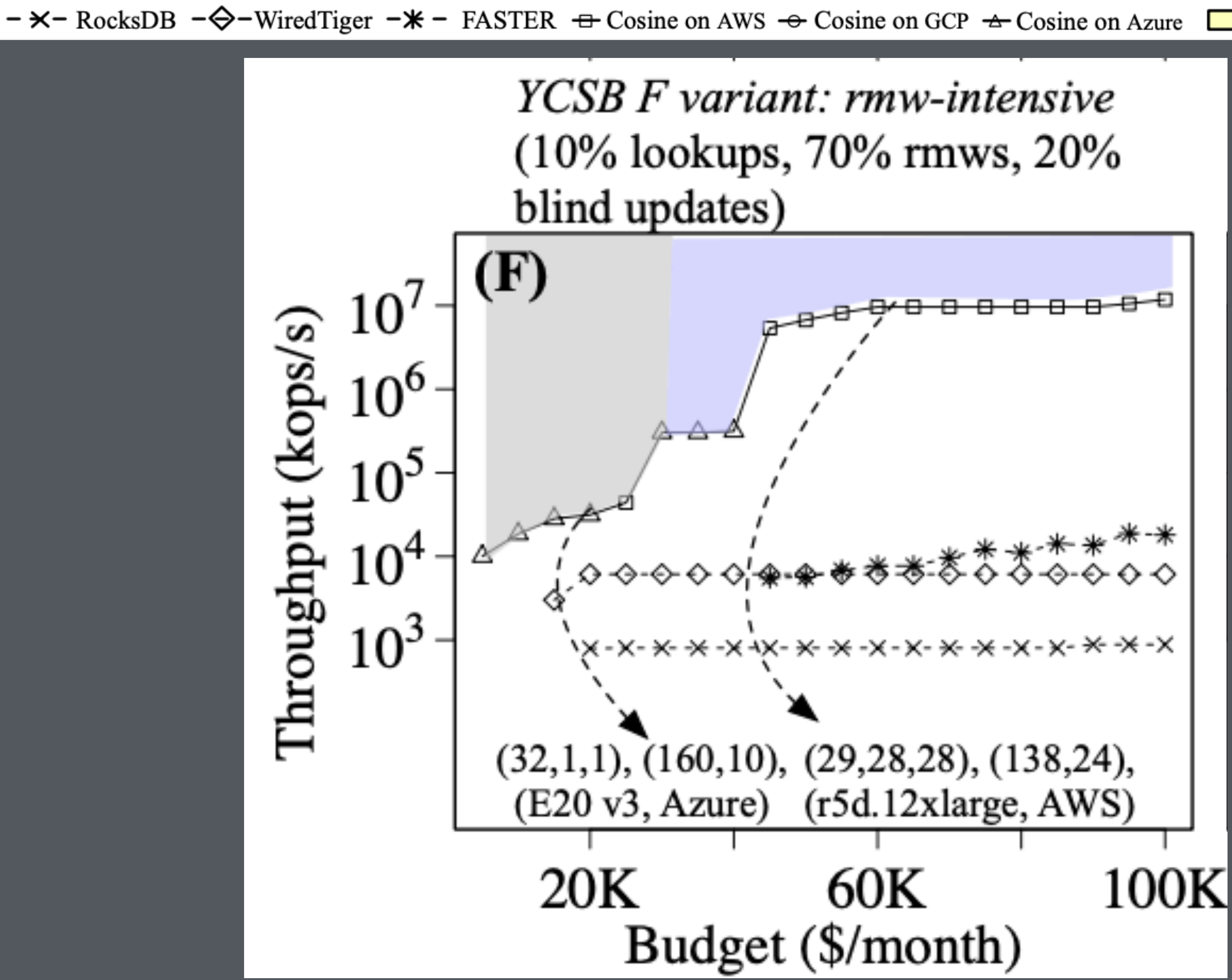


COSINE

State of the Art
Meta, Microsoft, Mongo

workload/budget diversity

DASlab @ Harvard SEAS



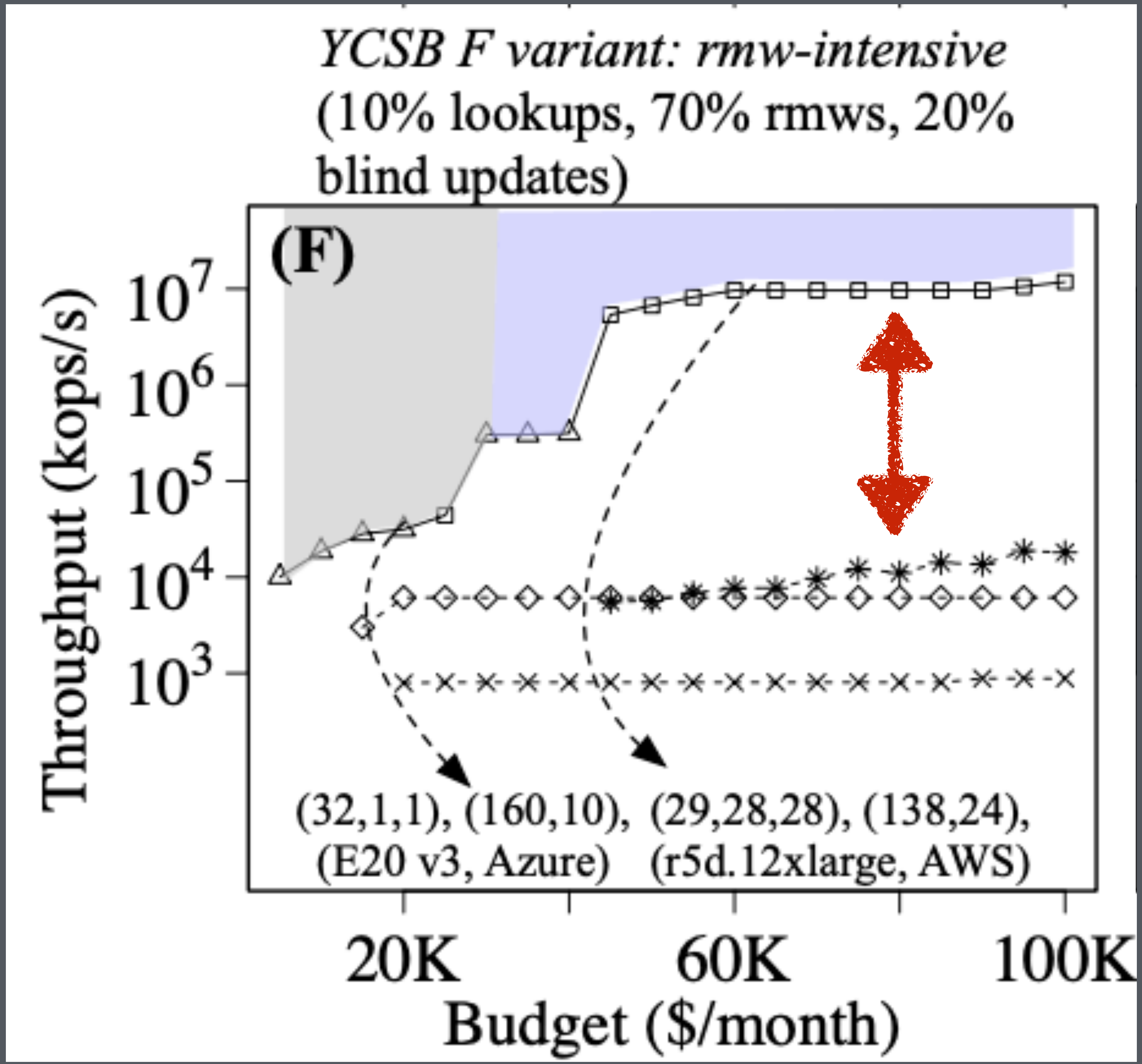
COSINE

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

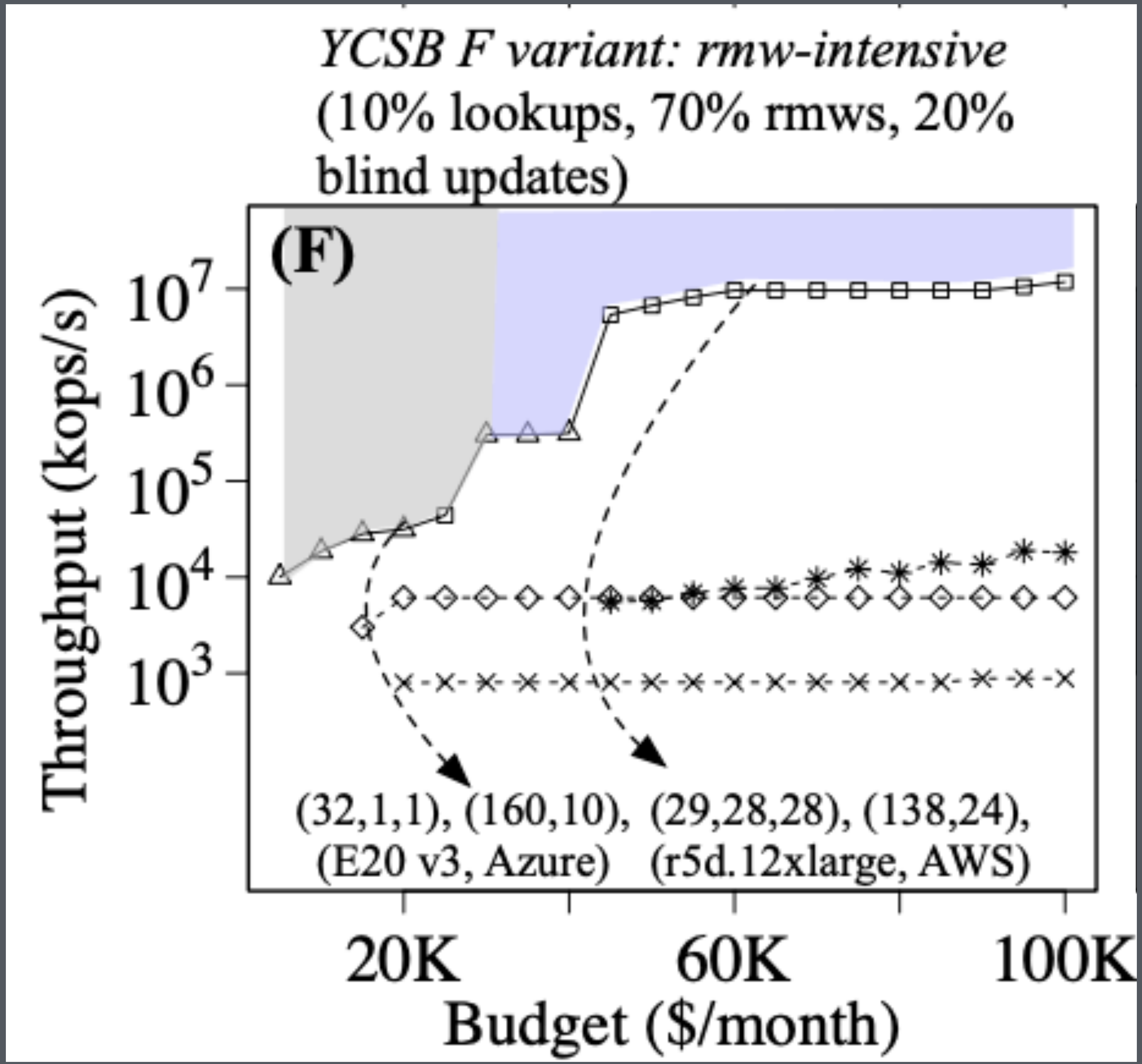
Better throughput/cost

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

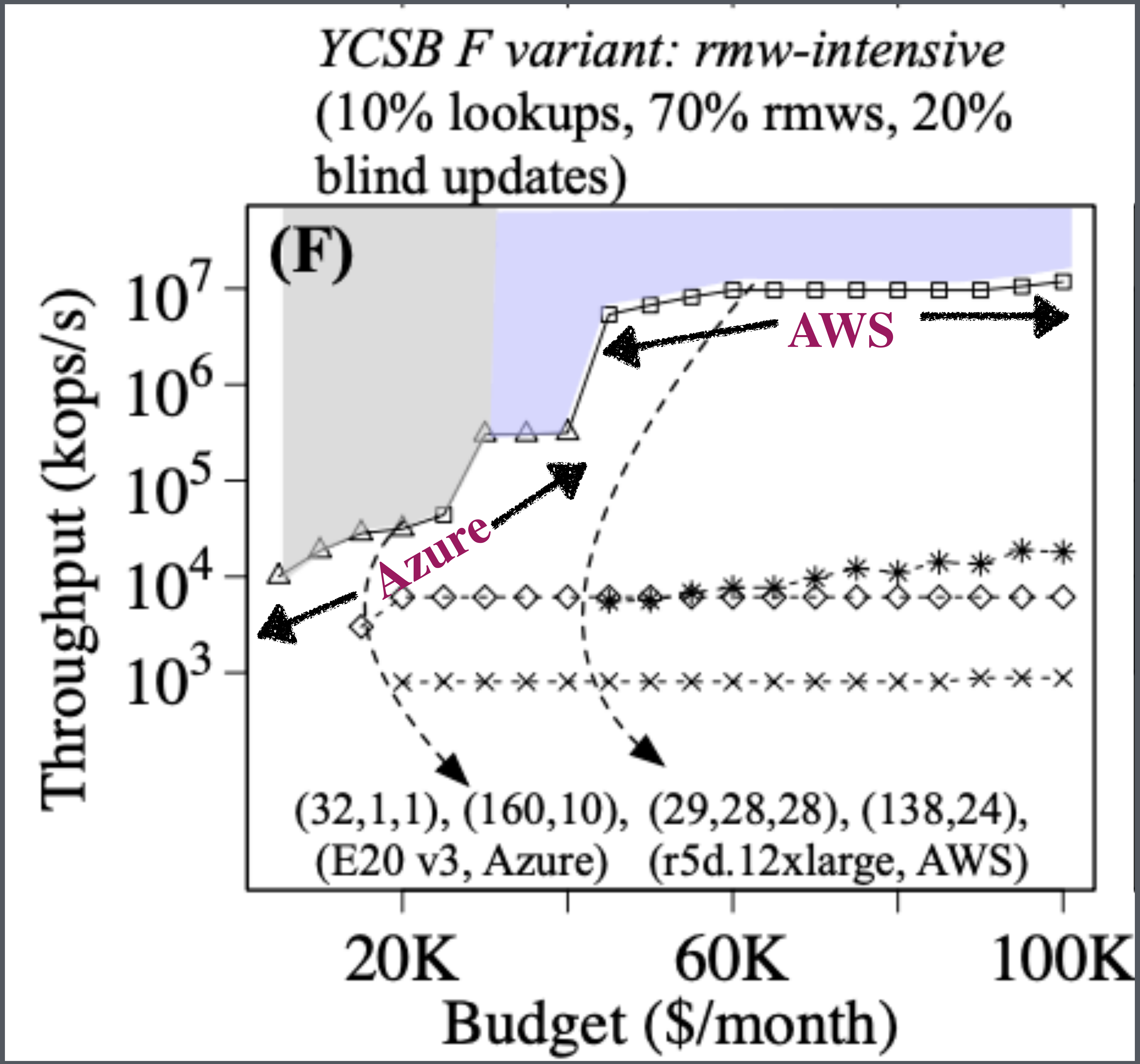
Self-designs

State of the Art
Meta, Microsoft, Mongo

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

Better throughput/cost

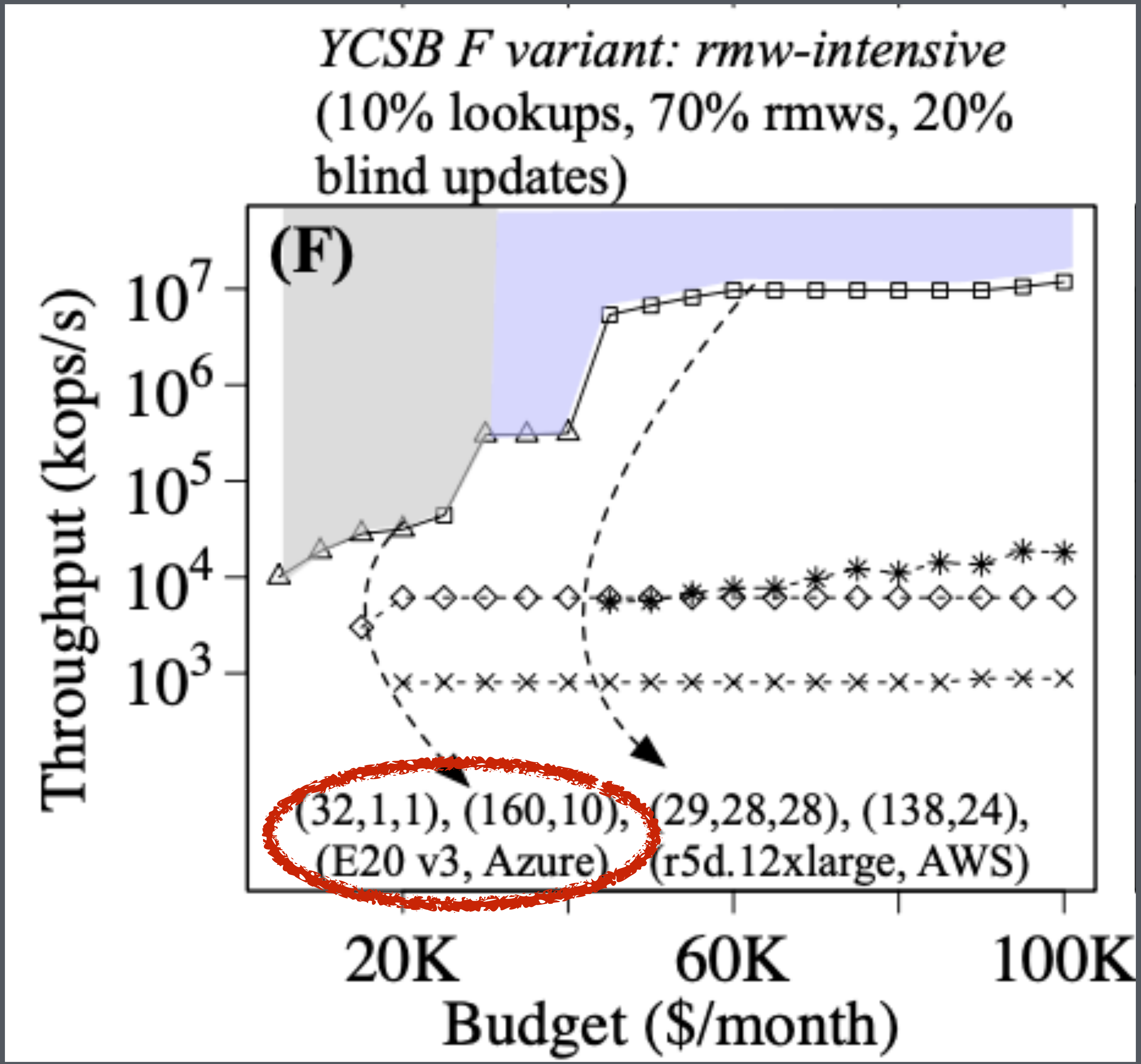
Self-designs

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

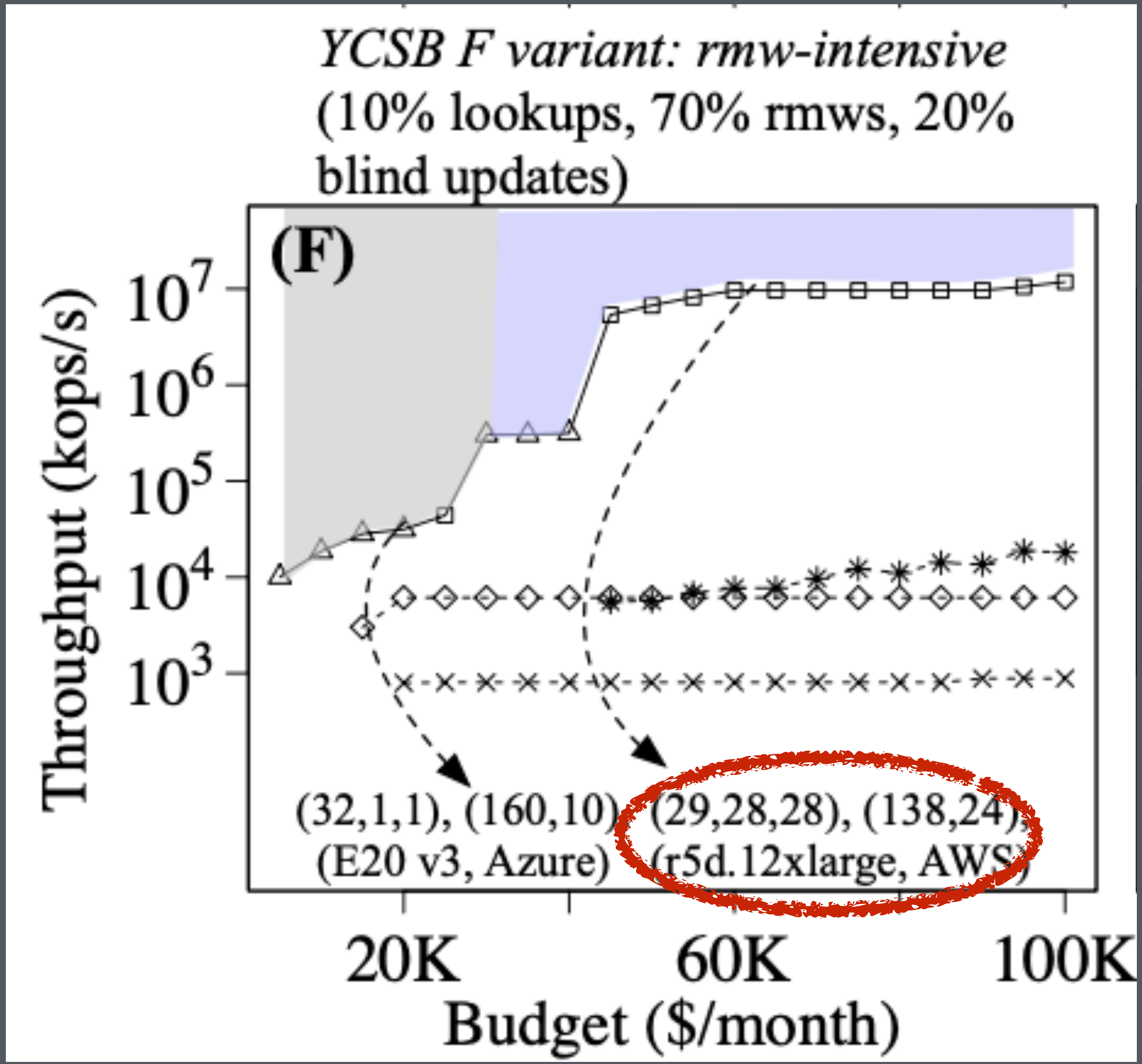
Better throughput/cost

Self-designs

State of the Art
Meta, Microsoft, Mongo

DASlab @ Harvard SEAS

workload/budget diversity



COSINE

Better throughput/cost

Self-designs

State of the Art
Meta, Microsoft, Mongo

workload/budget diversity

DASlab @ Harvard SEAS

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

workload/budget diversity

DASlab @ Harvard SEAS

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

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

YCSB A variant: blind-update-intensive (10% lookups, 20% rmws, 70% blind updates)

YCSB B variant: lookup-intensive (70% lookups, 25% inserts, 5% blind updates)

YCSB A+D variant: insert-intensive (70% inserts, 30% blind updates)

YCSB extended: mixed without range (25% lookups, 25% inserts, 25% rmws, 25% blind updates)

YCSB extended: mixed with range (10% lookups, 25% inserts, 5% rmws, 30% blind updates, 10% non-empty, 20% empty ranges)

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

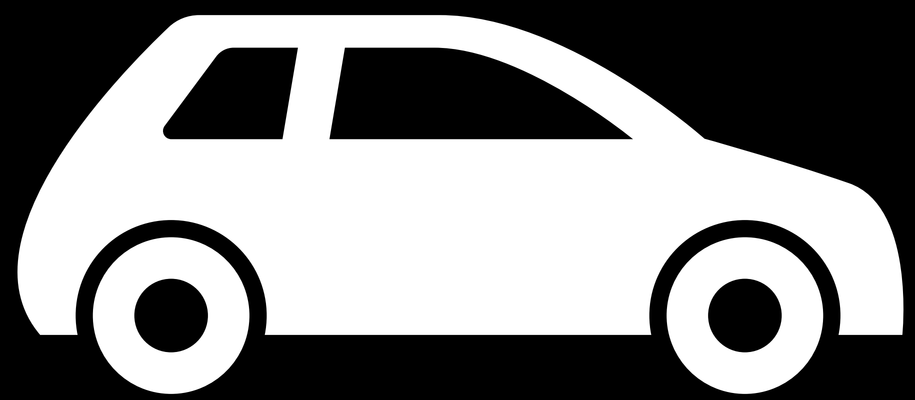
How do these concepts translate to the other big data areas

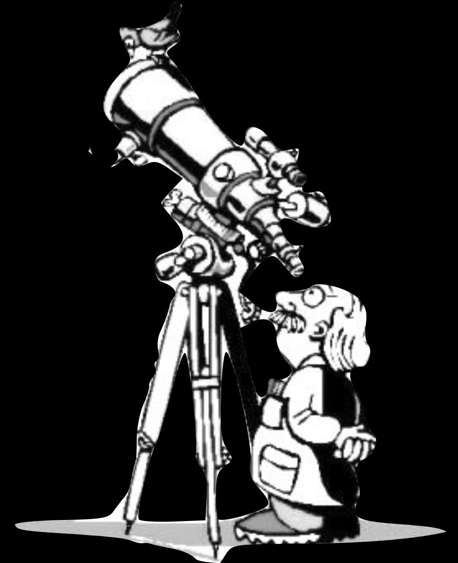
neural networks, image AI, Blockchain, …?

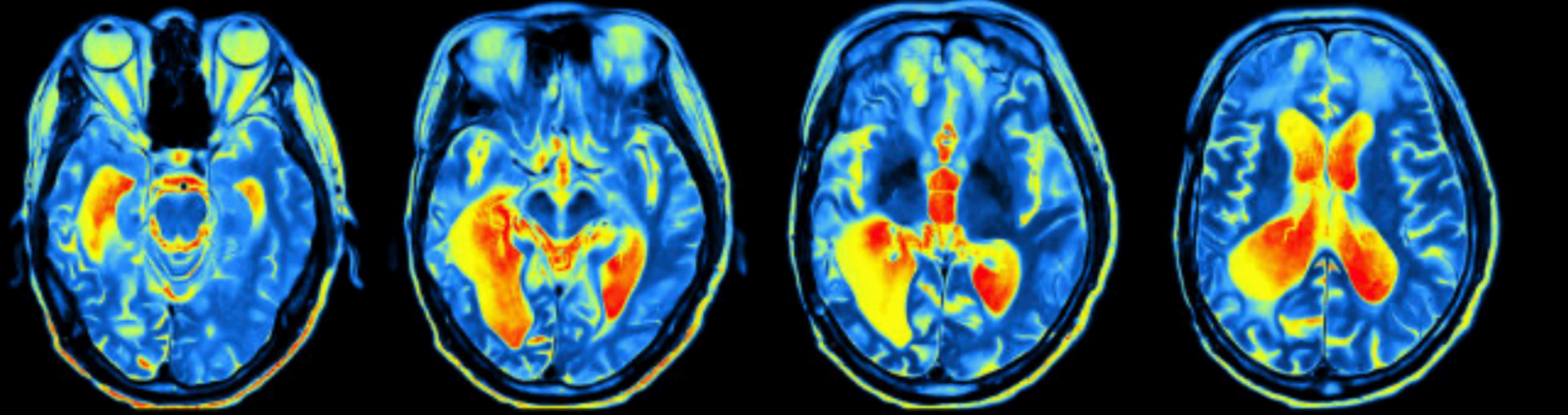
again, it all starts from the storage design space

seeing is at the very center of AI
because it is at the center of human life

image processing

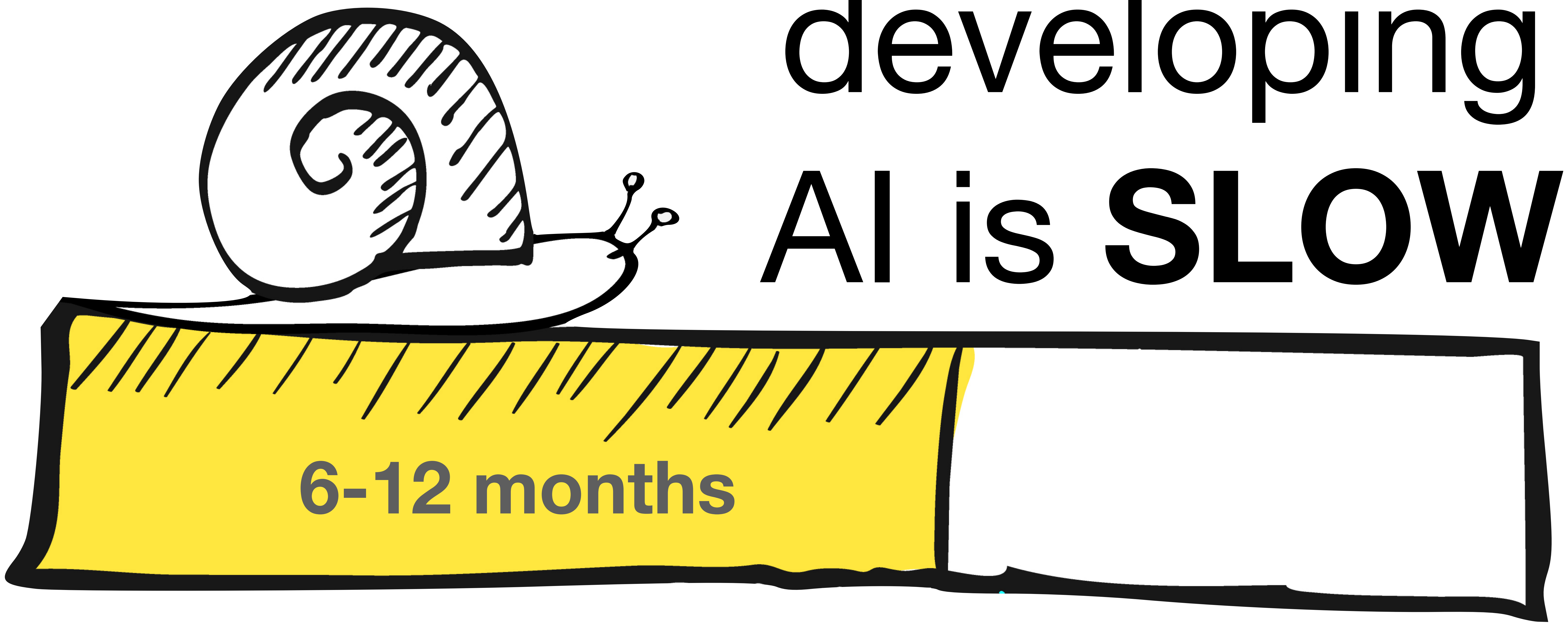






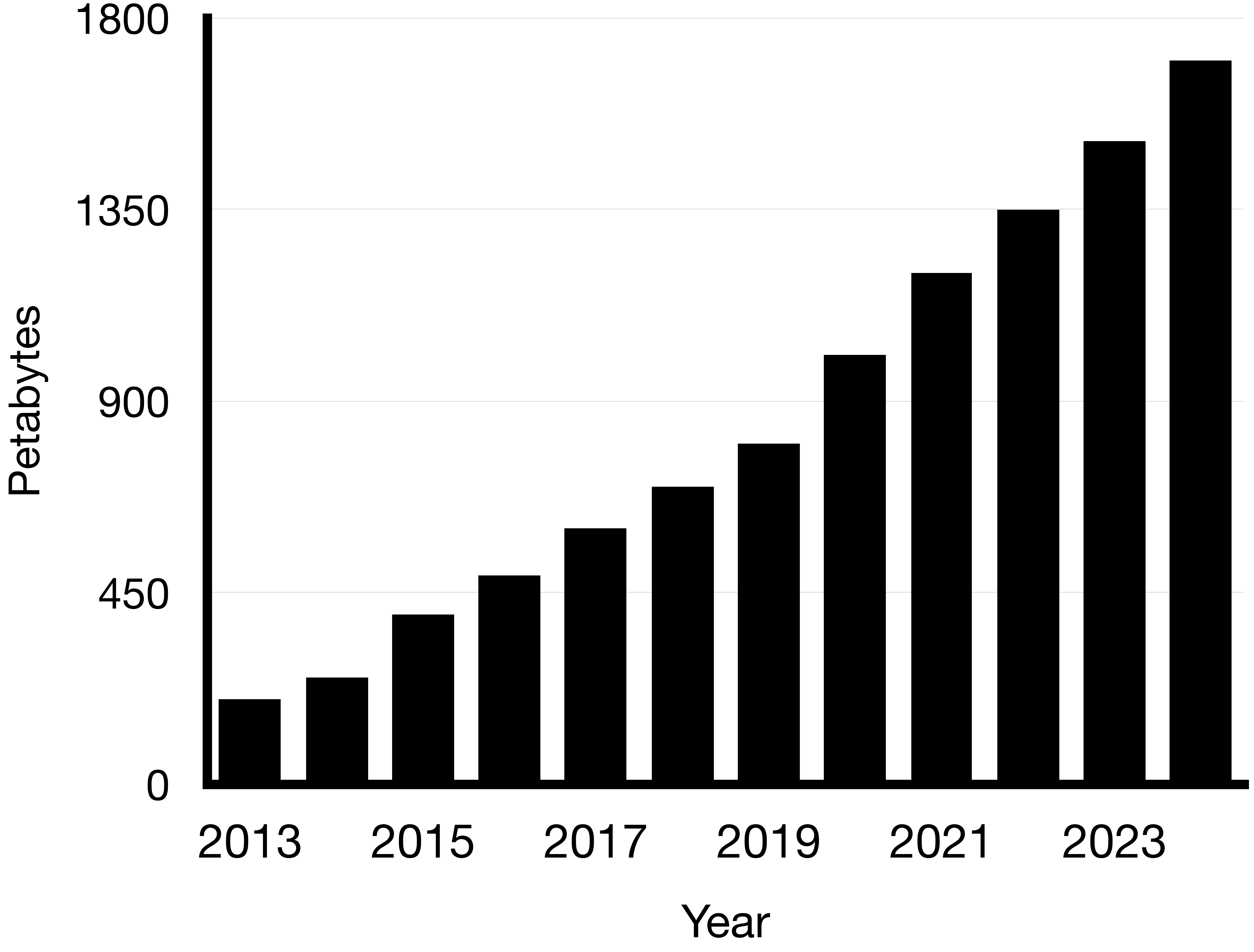
developing

AI is **SLOW**



Data size Model size

Data size

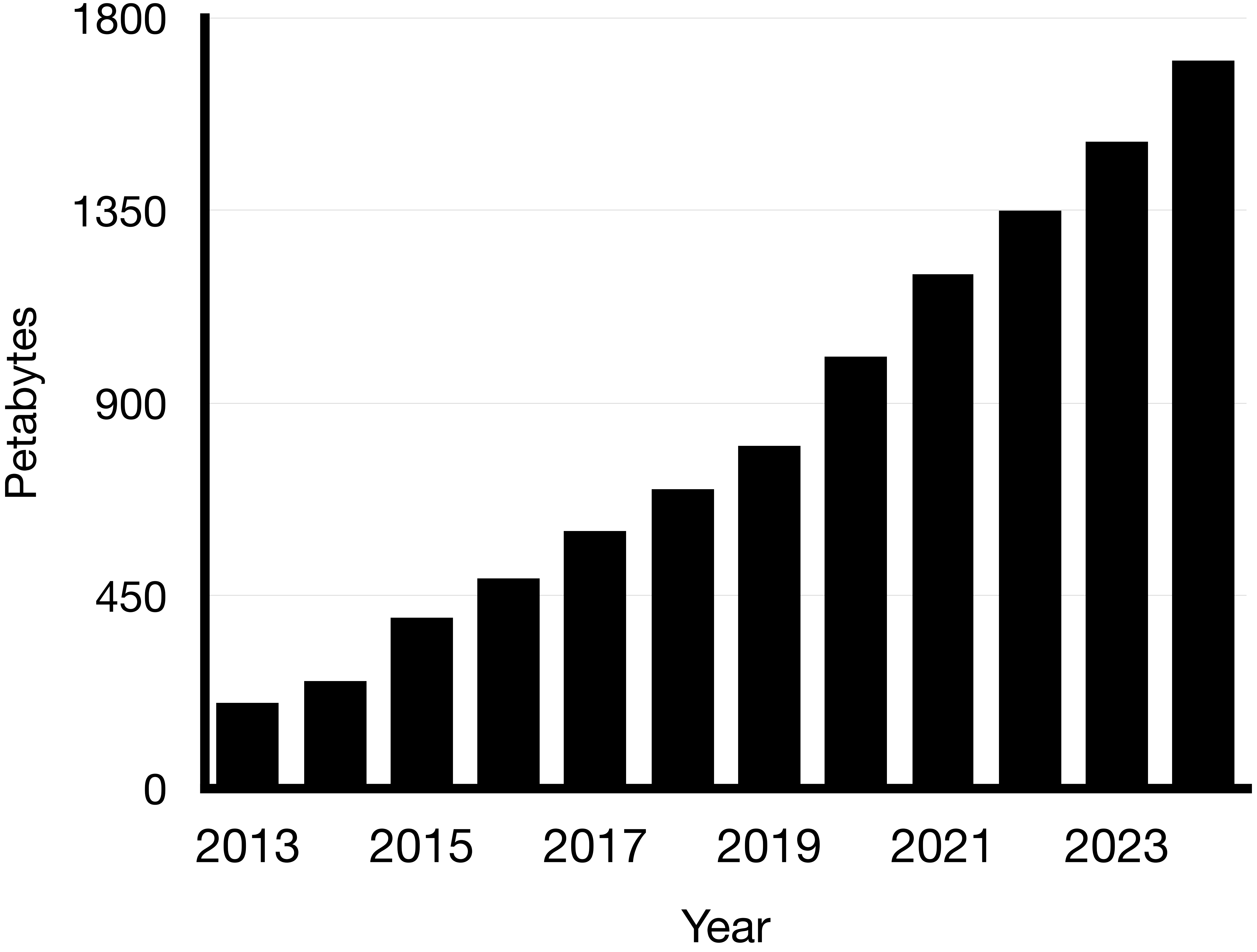


Model size

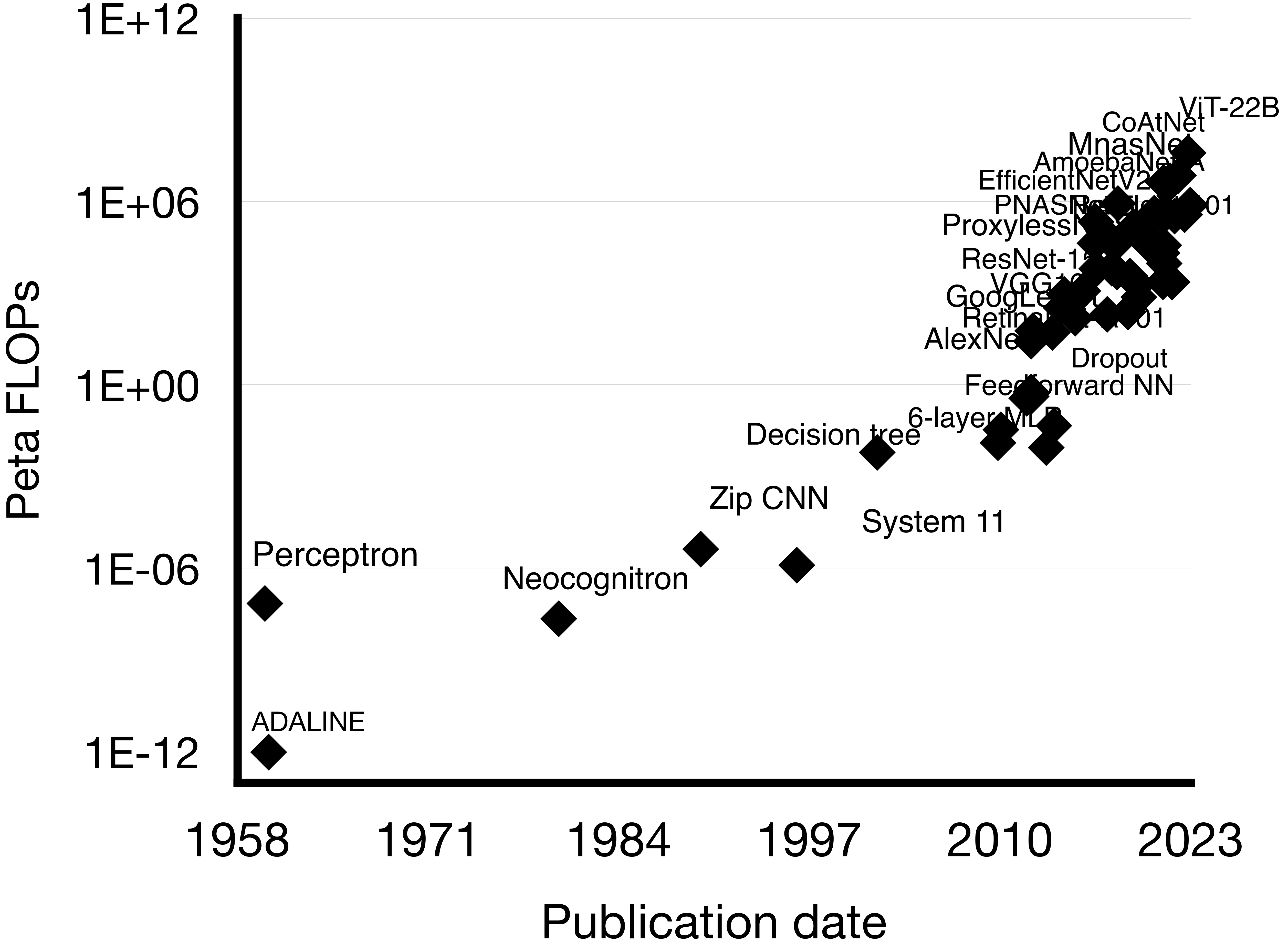
[1] https://ourworldindata.org/artificial-intelligence
[2] https://ourworldindata.org/grapher/artificial-intelligence-training-computation

DASlab @ Harvard SEAS

Data size



Model size

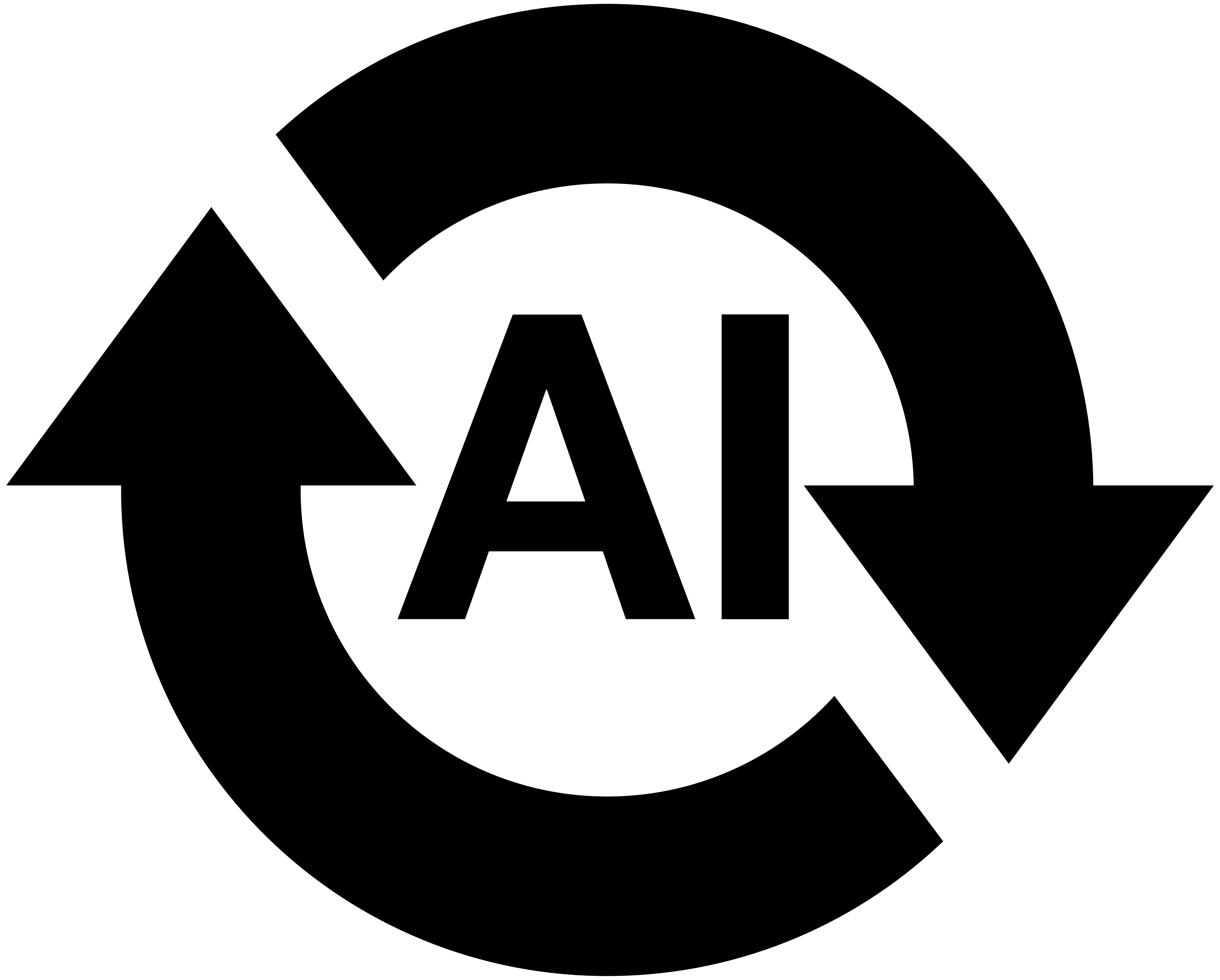


DASlab @ Harvard SEAS

[1] https://ourworldindata.org/artificial-intelligence

[2] https://ourworldindata.org/grapher/artificial-intelligence-training-computation

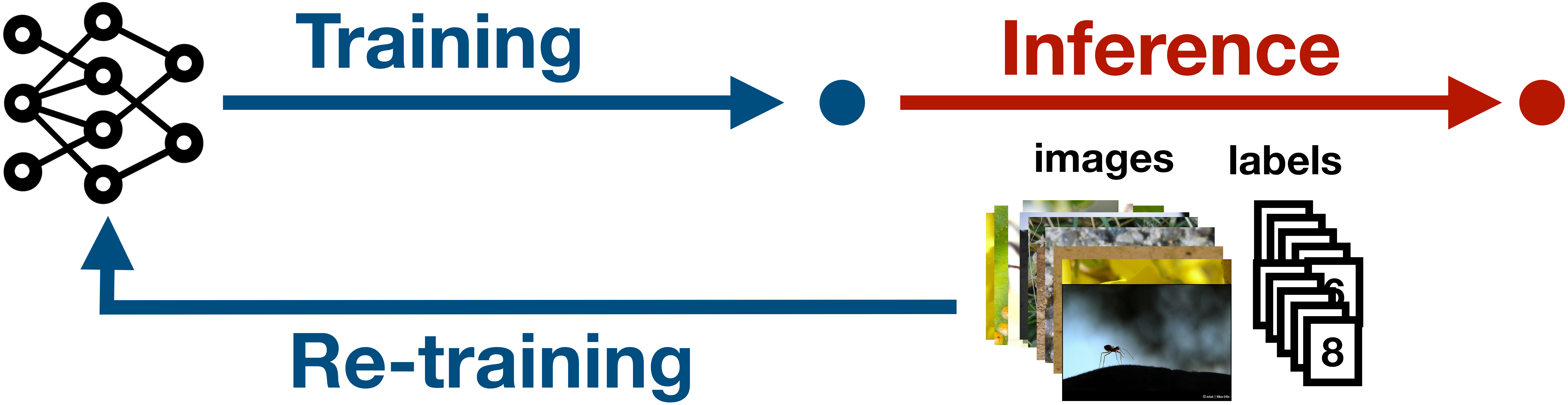
Training



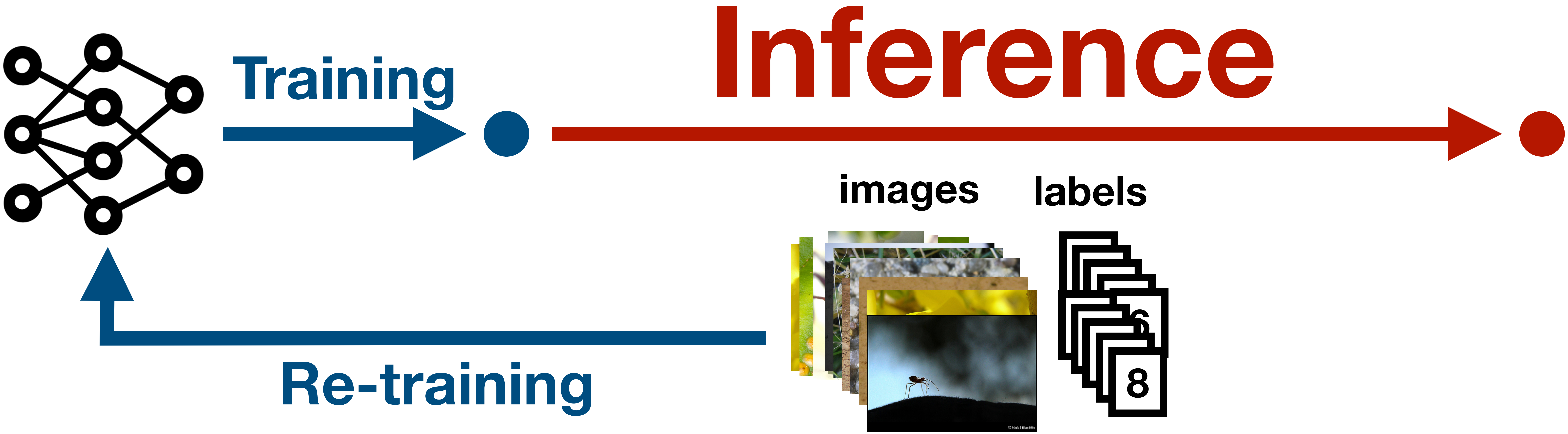
Inference


Training
AI
Inference


What's this?
Training
AI
It's a dog!
Inference


Training
Inference
images
labels
6
8
Re-training

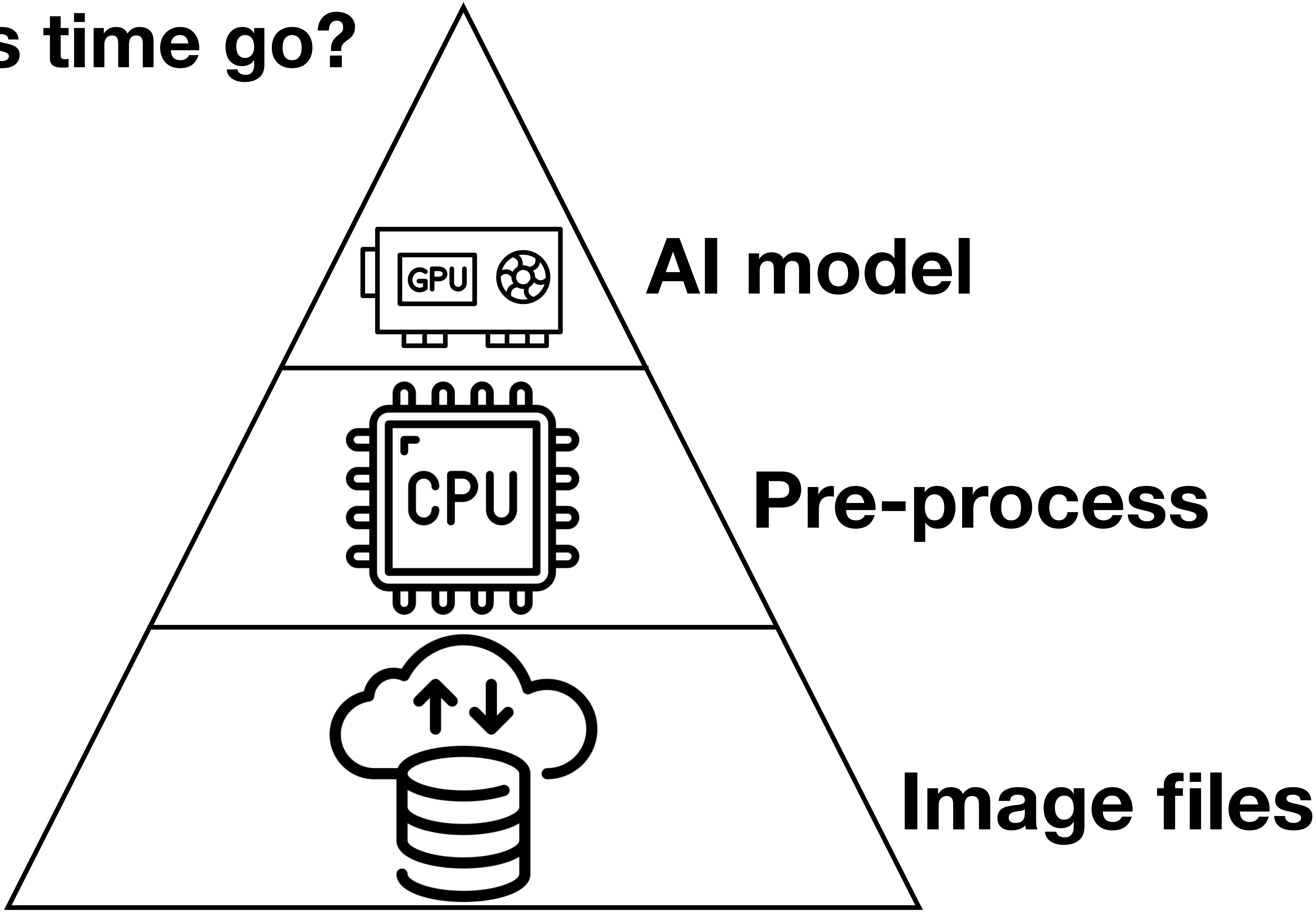

Training
Inference
images
labels
8
Re-training



[1] https://aws.amazon.com/blogs/aws/amazon-ec2-update-inf1-instances-with-aws-inferentia-chips-for-high-performance-cost-effective-inferencing/
[2] https://www.forbes.com/sites/moorinsights/2019/05/09/google-cloud-doubles-down-on-nvidia-gpus-for-inference/?sh=4c9817226792

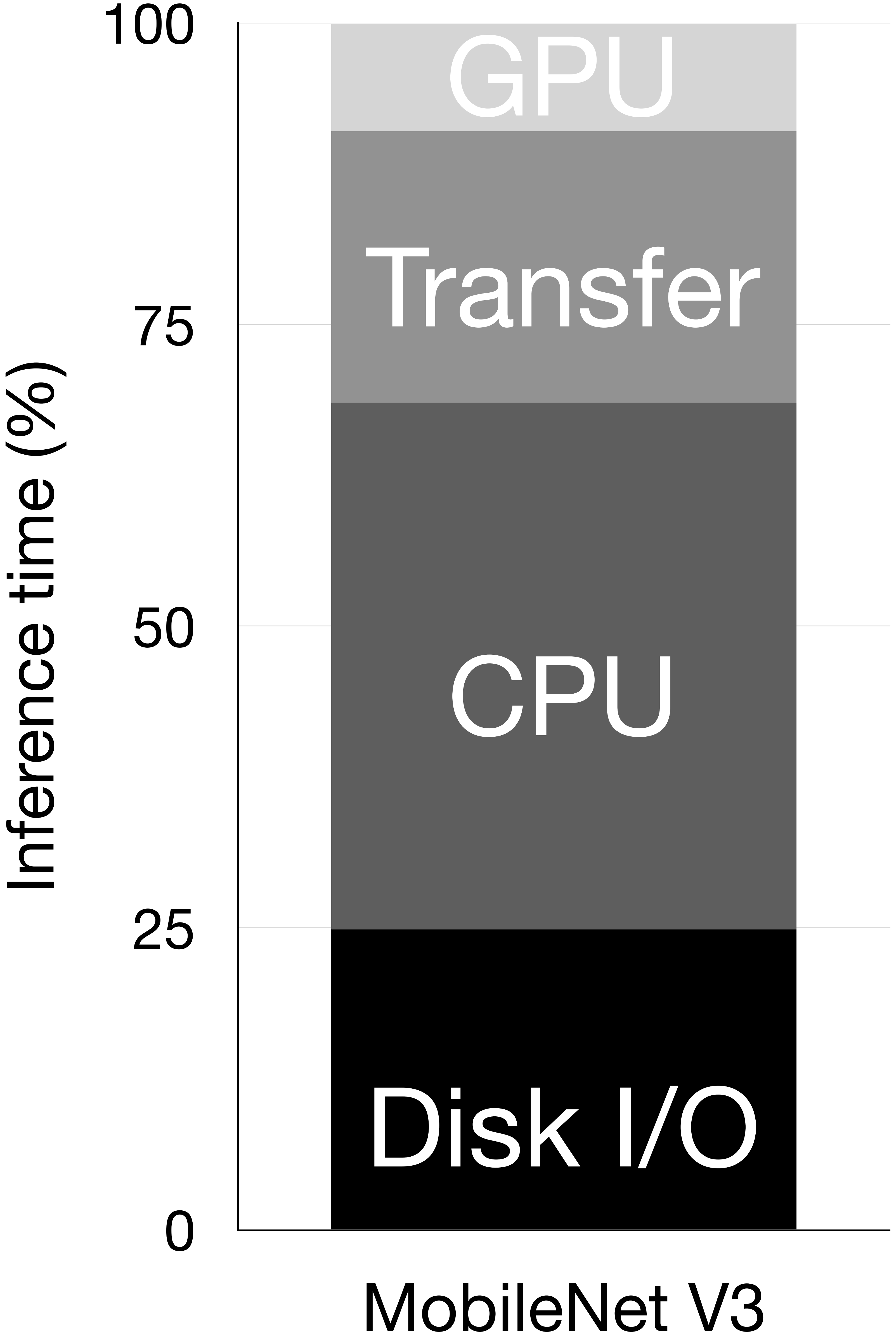
[1] https://aws.amazon.com/blogs/aws/amazon-ec2-update-inf1-instances-with-aws-inferentia-chips-for-high-performance-cost-effective-inferencing/
[2] https://www.forbes.com/sites/moorinsights/2019/05/09/google-cloud-doubles-down-on-nvidia-gpus-for-inference/?sh=4c9817226792

Where does time go?



Where does time go?

Data: ImageNet
AI Model: MobileNet V3
Machine: V100, PCIe
Xeon, SSD
Framework: PyTorch v1

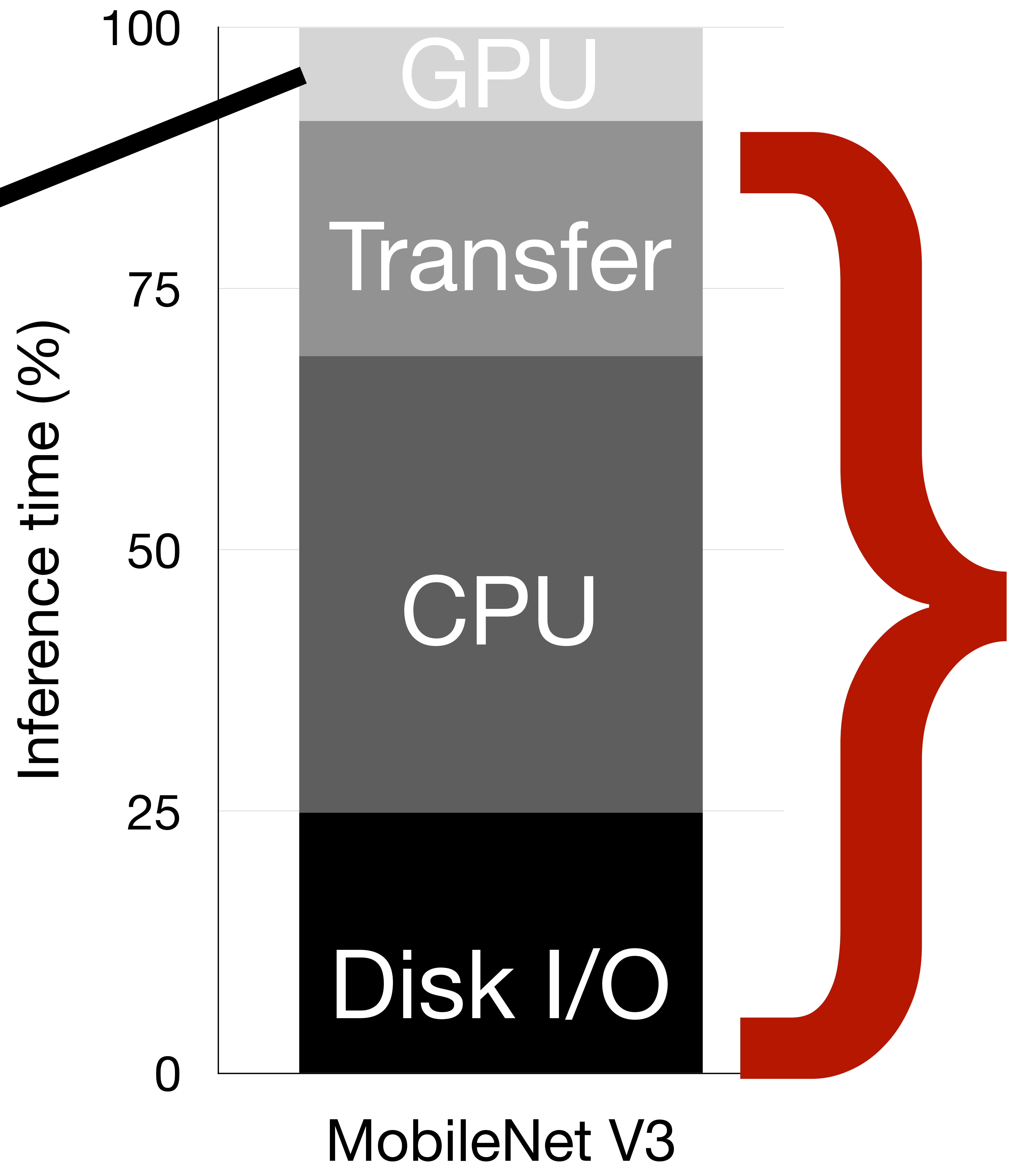


DASlab @ Harvard SEAS

Where does time go?

Data: ImageNet
AI Model: MobileNet V3
Machine: V100, PCIe
Xeon, SSD
Framework: PyTorch v1

Only 10% is GPU!

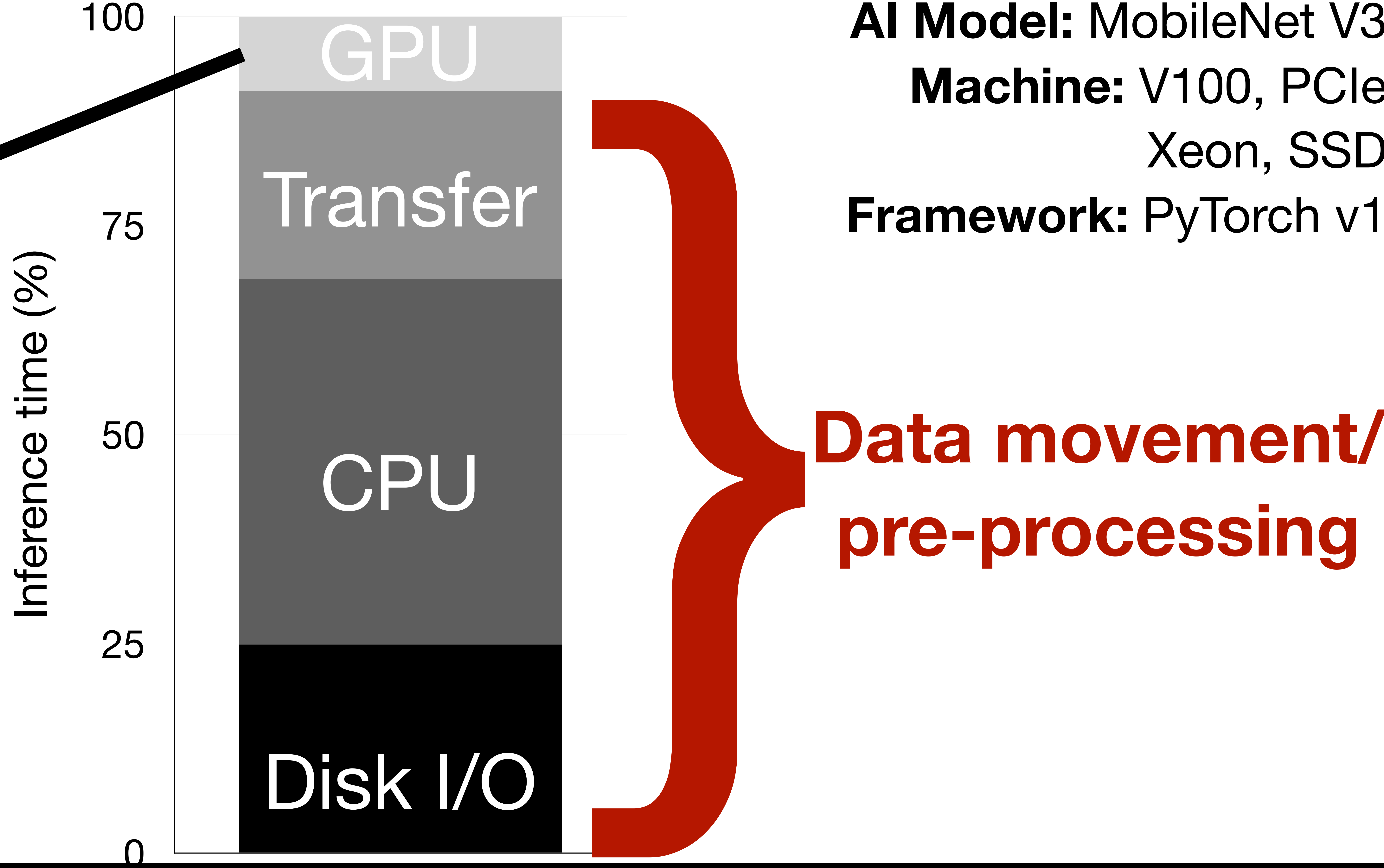


DASlab @ Harvard SEAS

Where does time go?

Data: ImageNet
AI Model: MobileNet V3
Machine: V100, PCIe
Xeon, SSD
Framework: PyTorch v1

Only 10% is GPU!



Re-think Storage for Image AI

How do machines store images today?

How do machines store images today?

JPEG

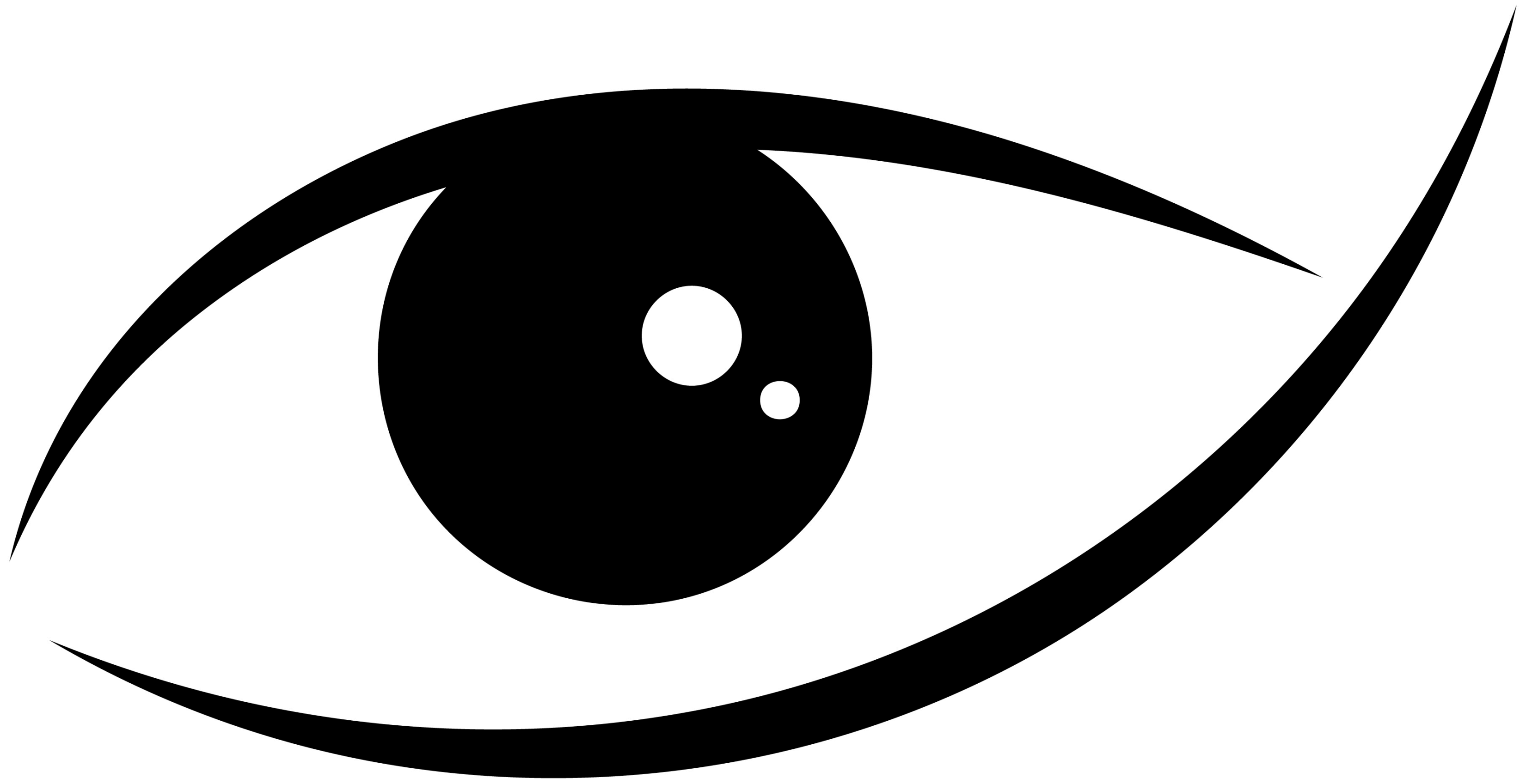
Joint Photographic Experts Group

How do machines store images today?

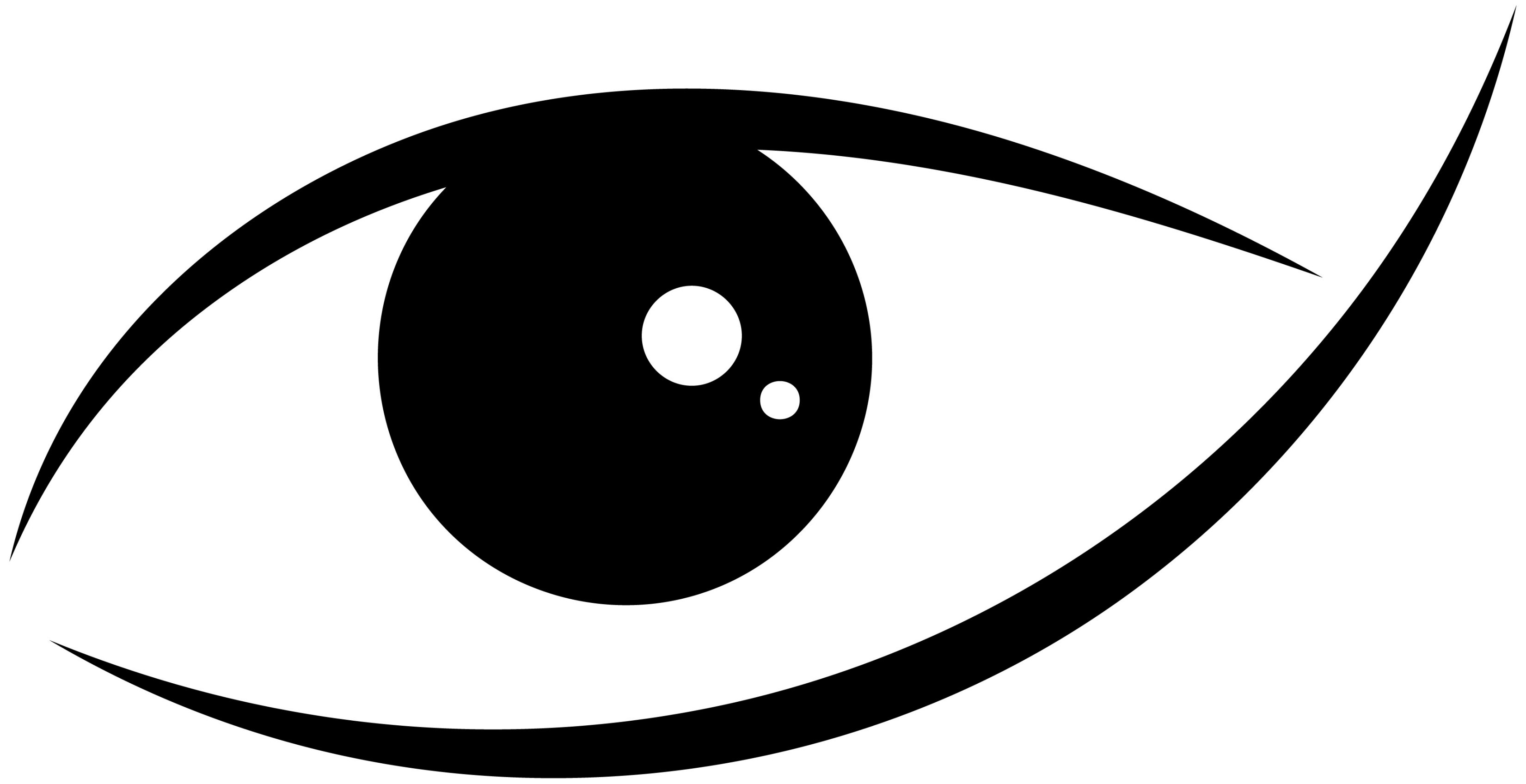
compression

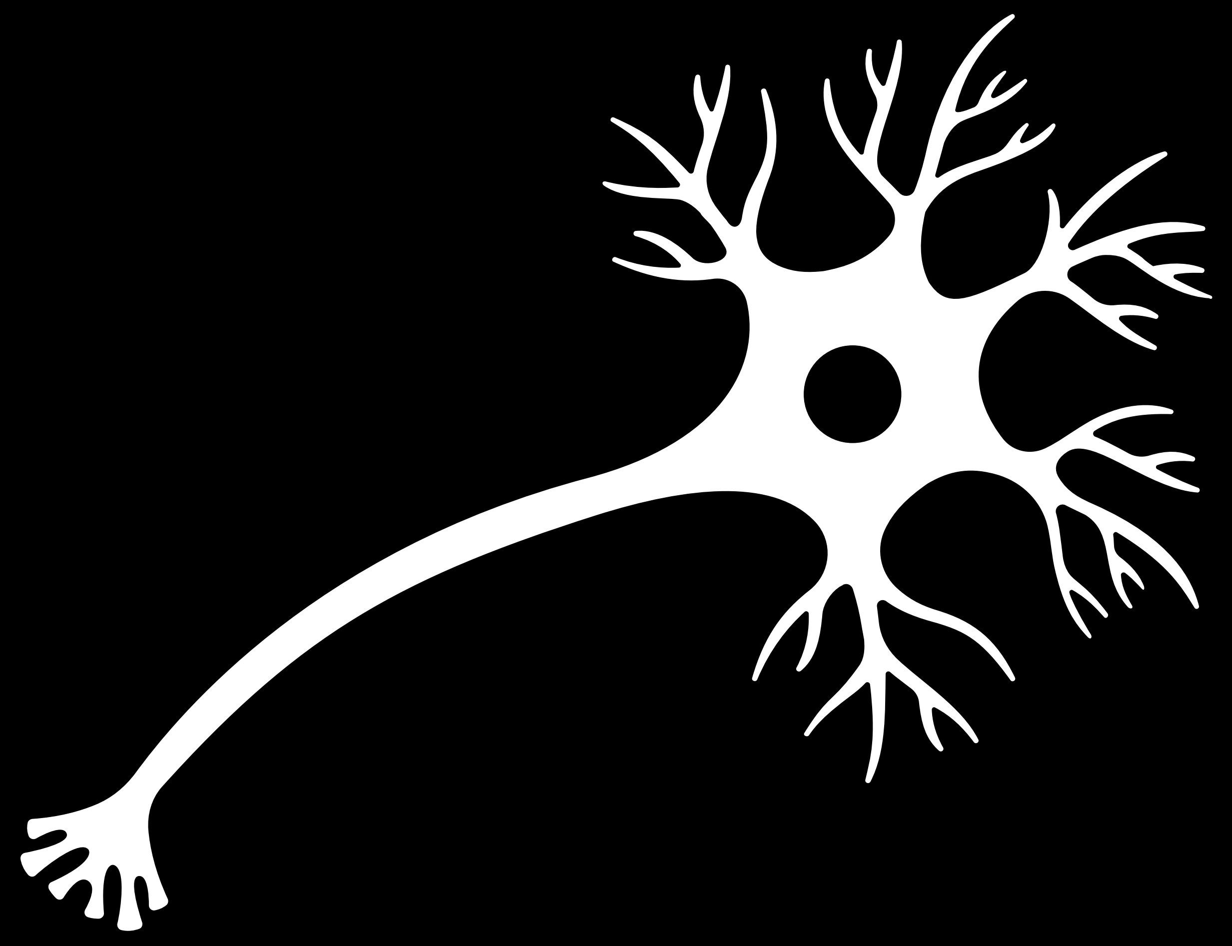
standard

JPEG is tailored for the properties of the human eye



JPEG is tailored for the properties of the human eye



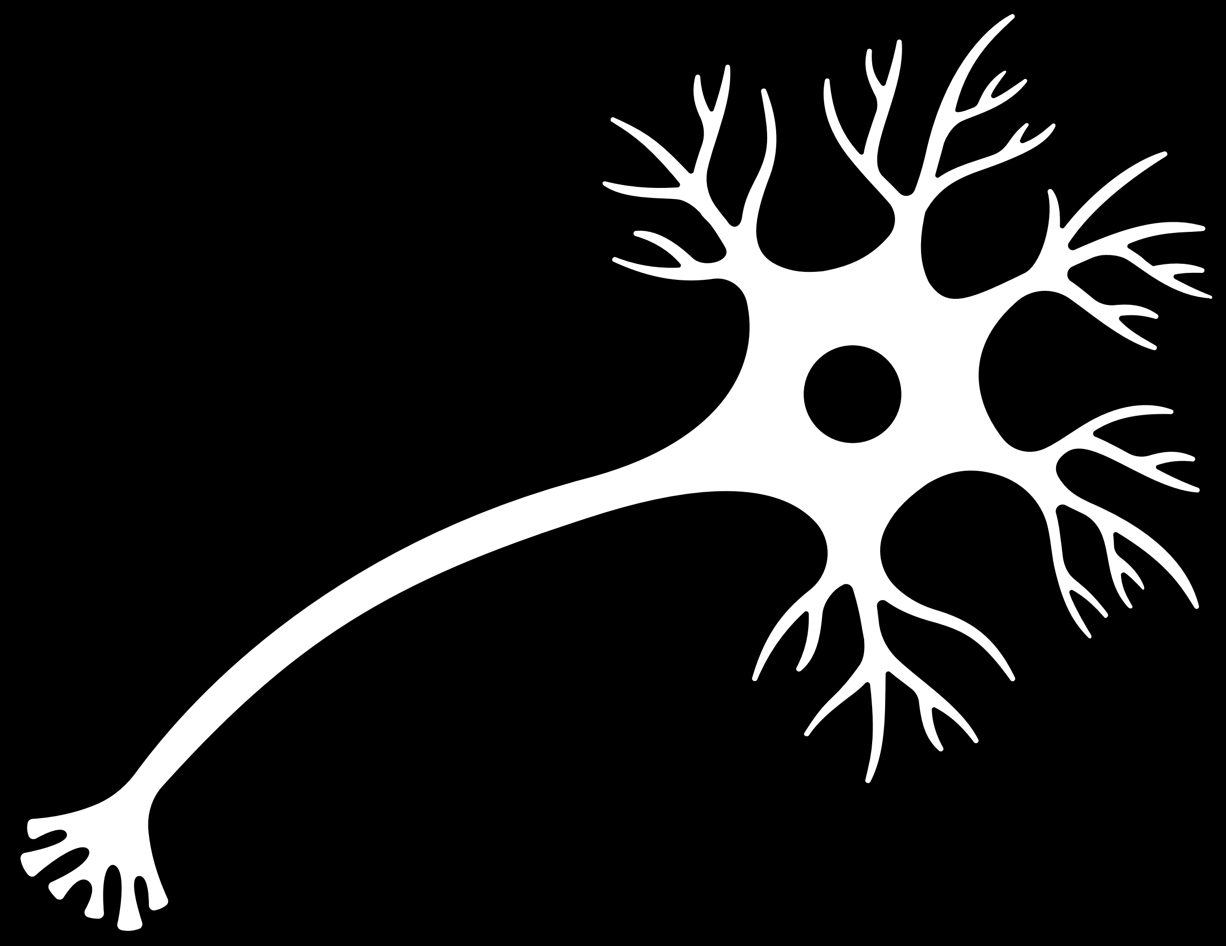


images for AI are seen by algorithms, not humans

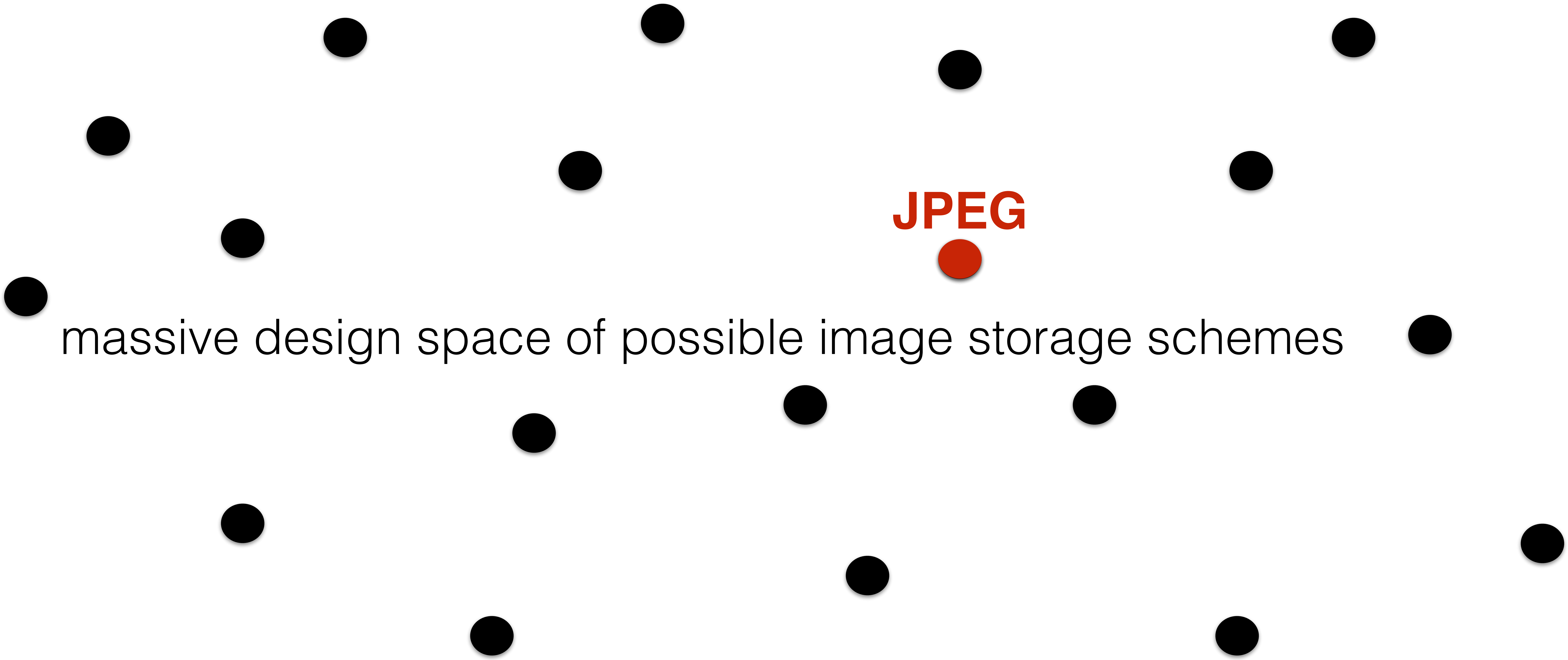
there are more possible ways to store an image than

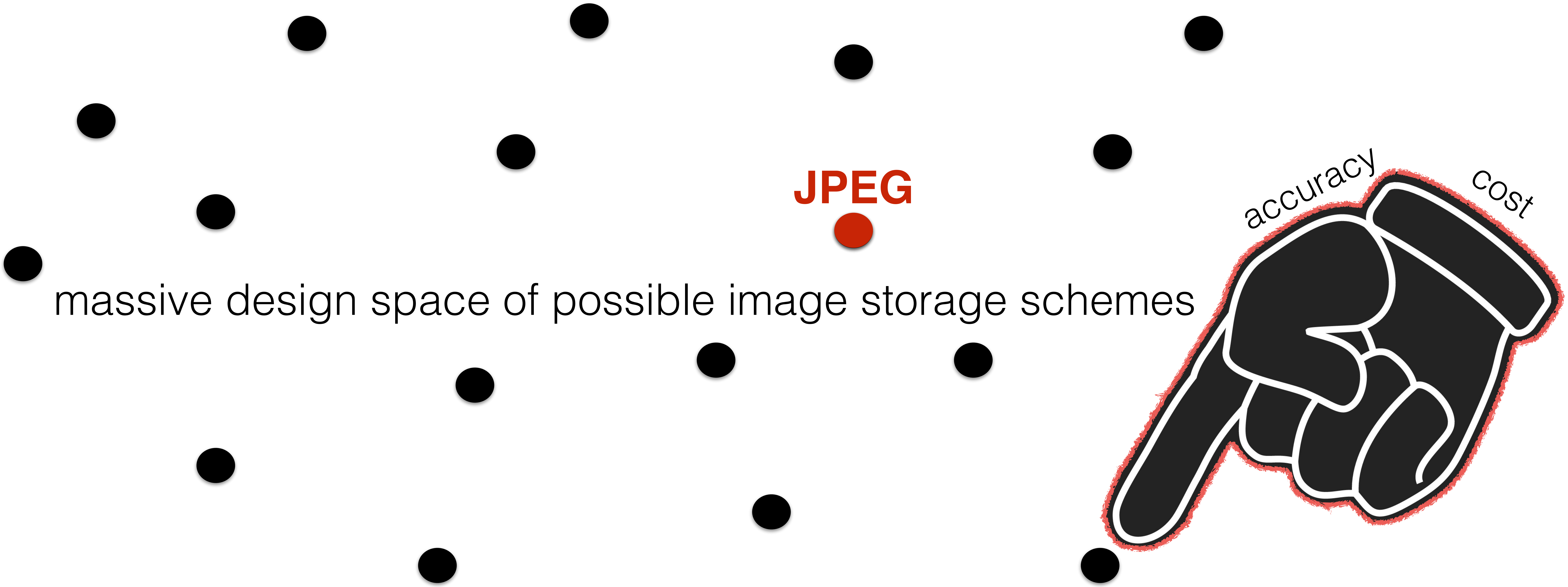
$10^{100} > 10^{24}$

stars on the sky

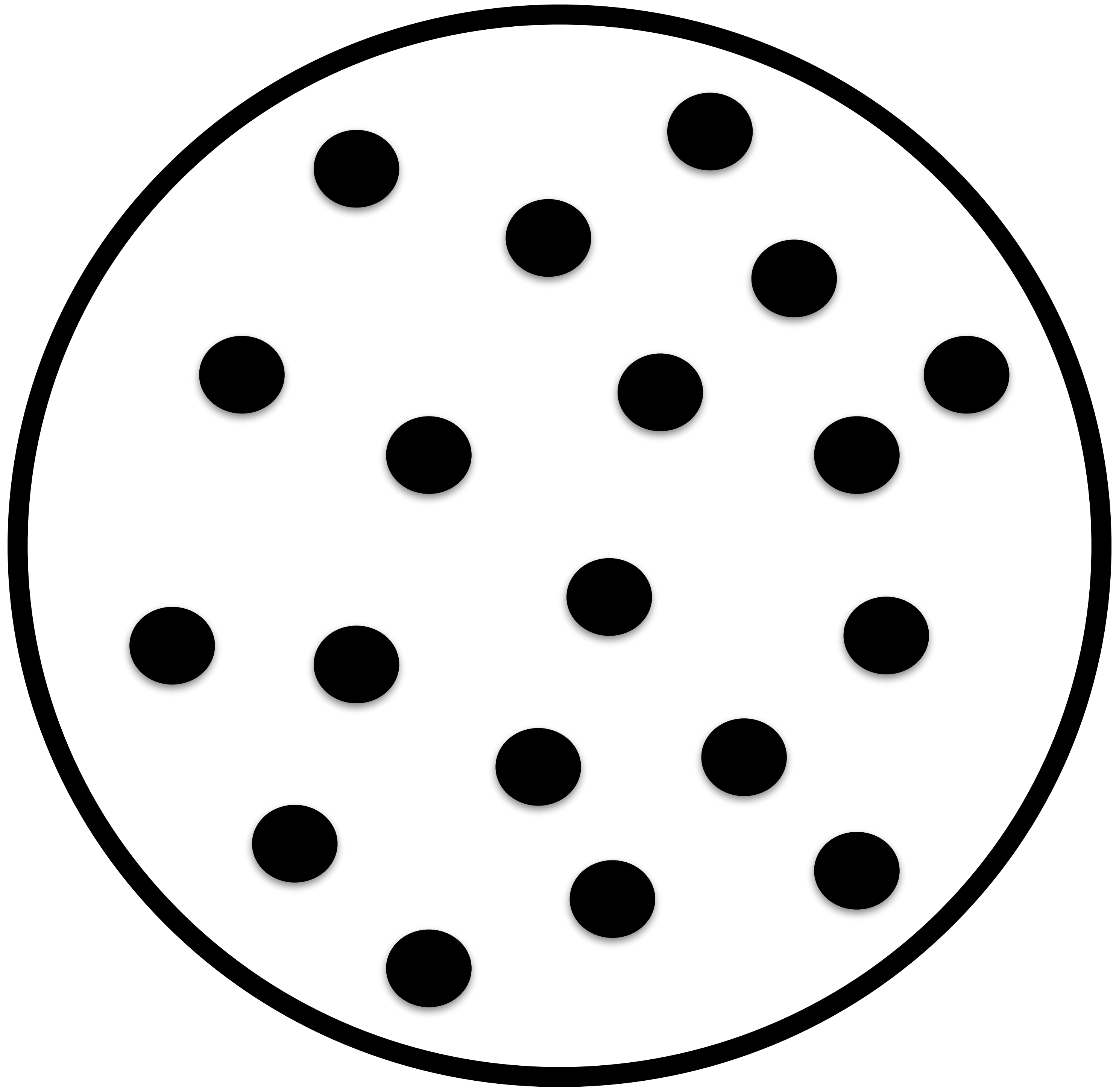


images for AI are seen by algorithms, not humans massive design space of possible image storage schemes

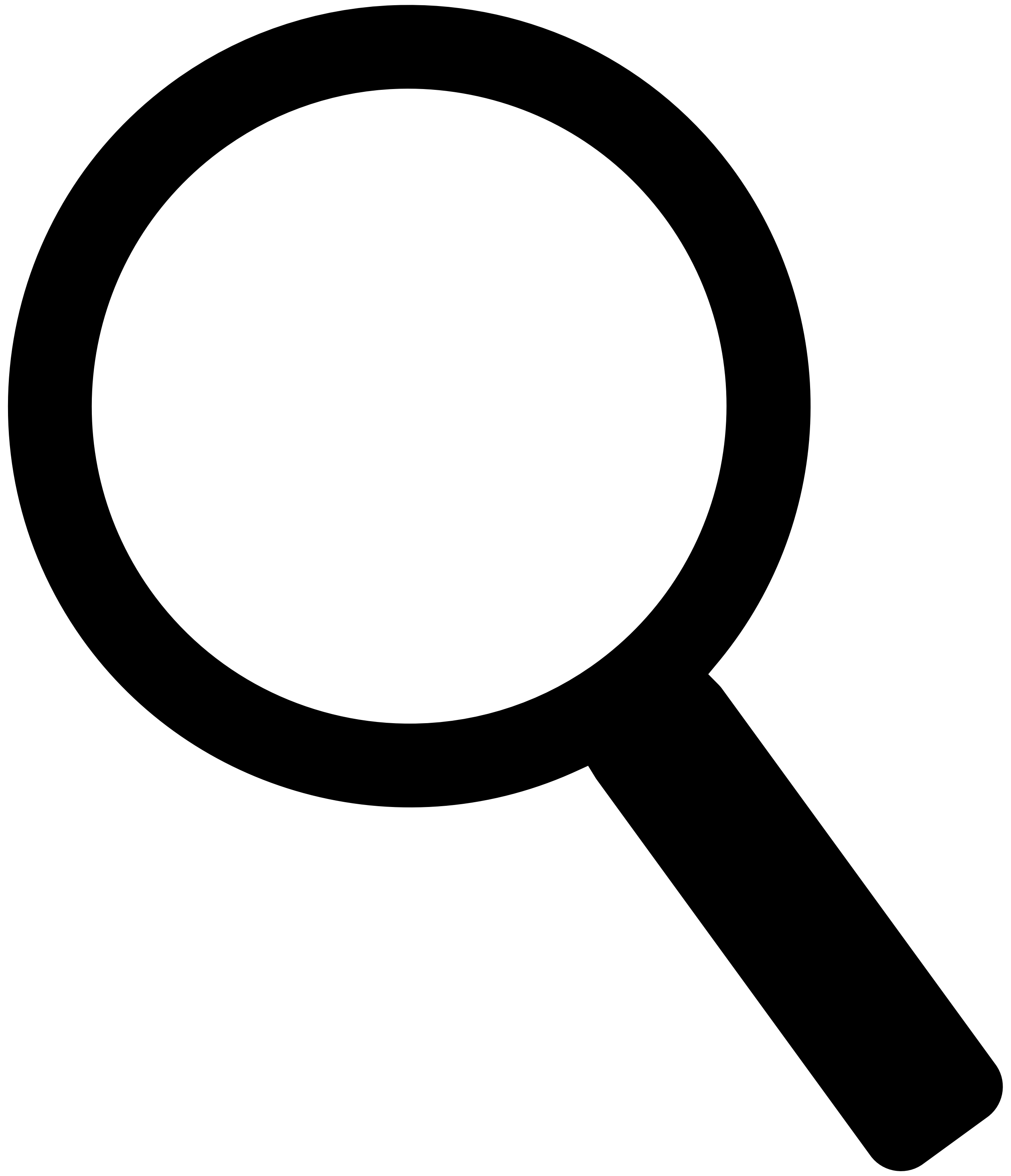

JPEG
massive design space of possible image storage schemes


JPEG
massive design space of possible image storage schemes
accuracy
cost

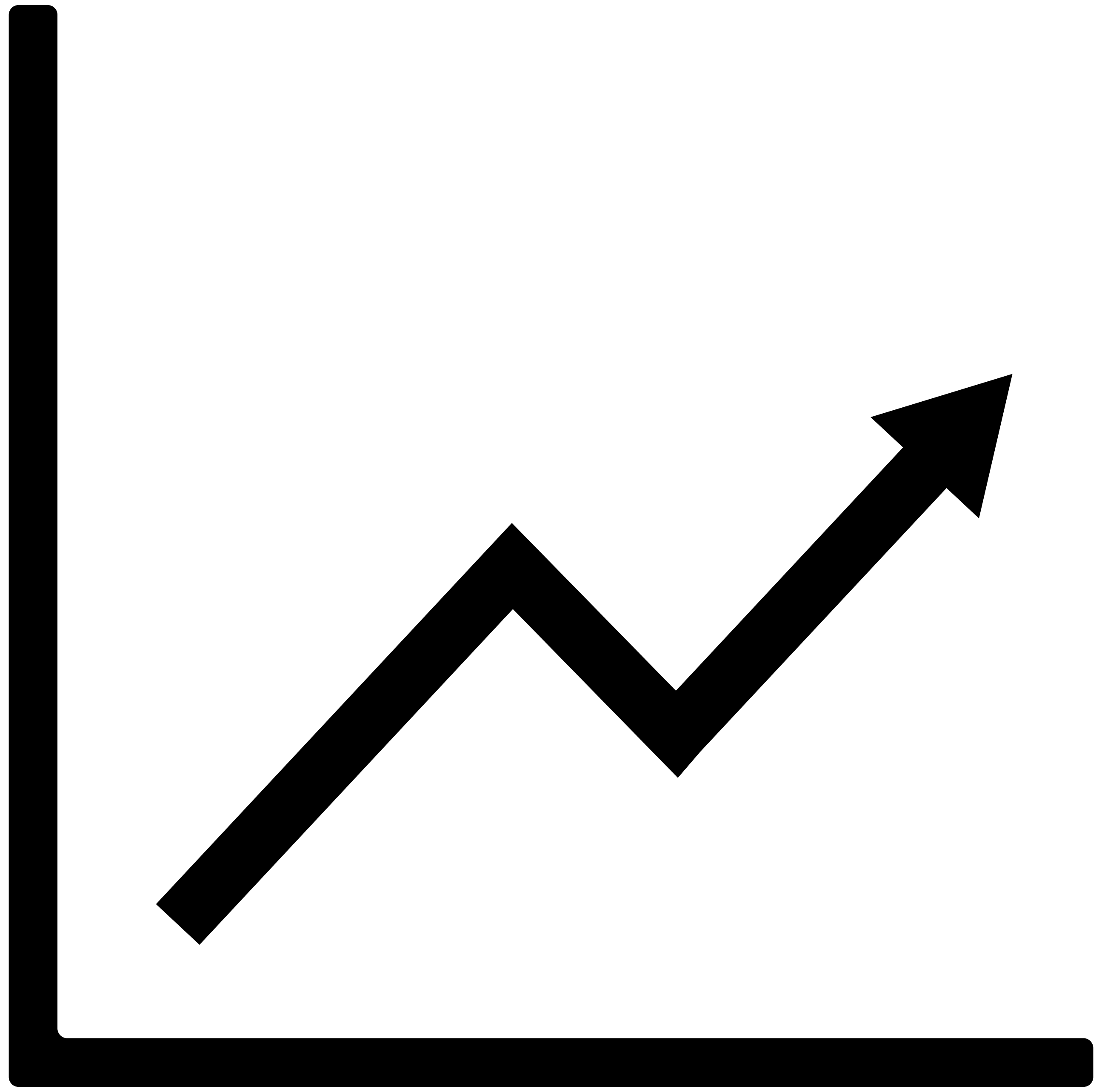
Design space



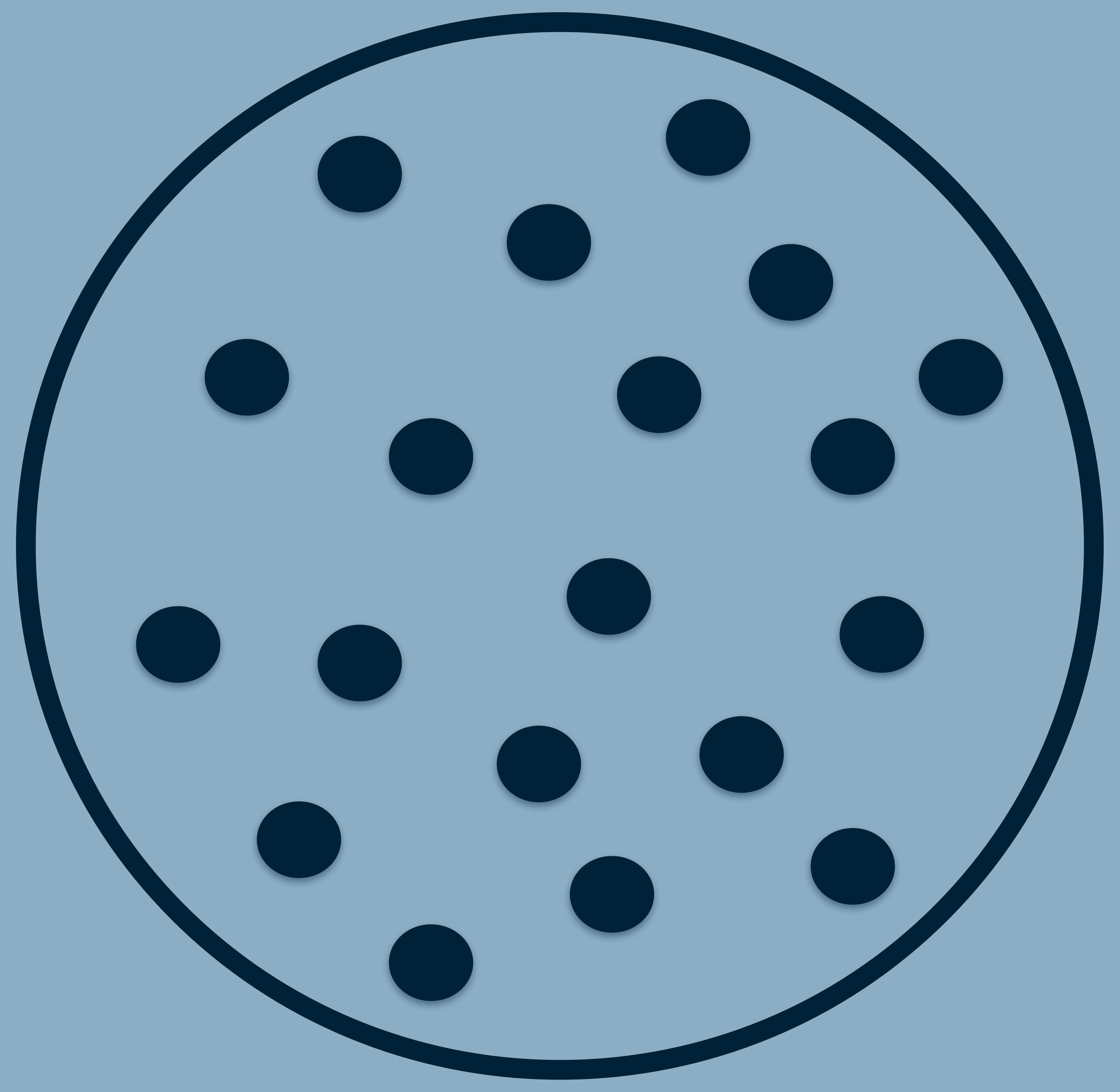
Search



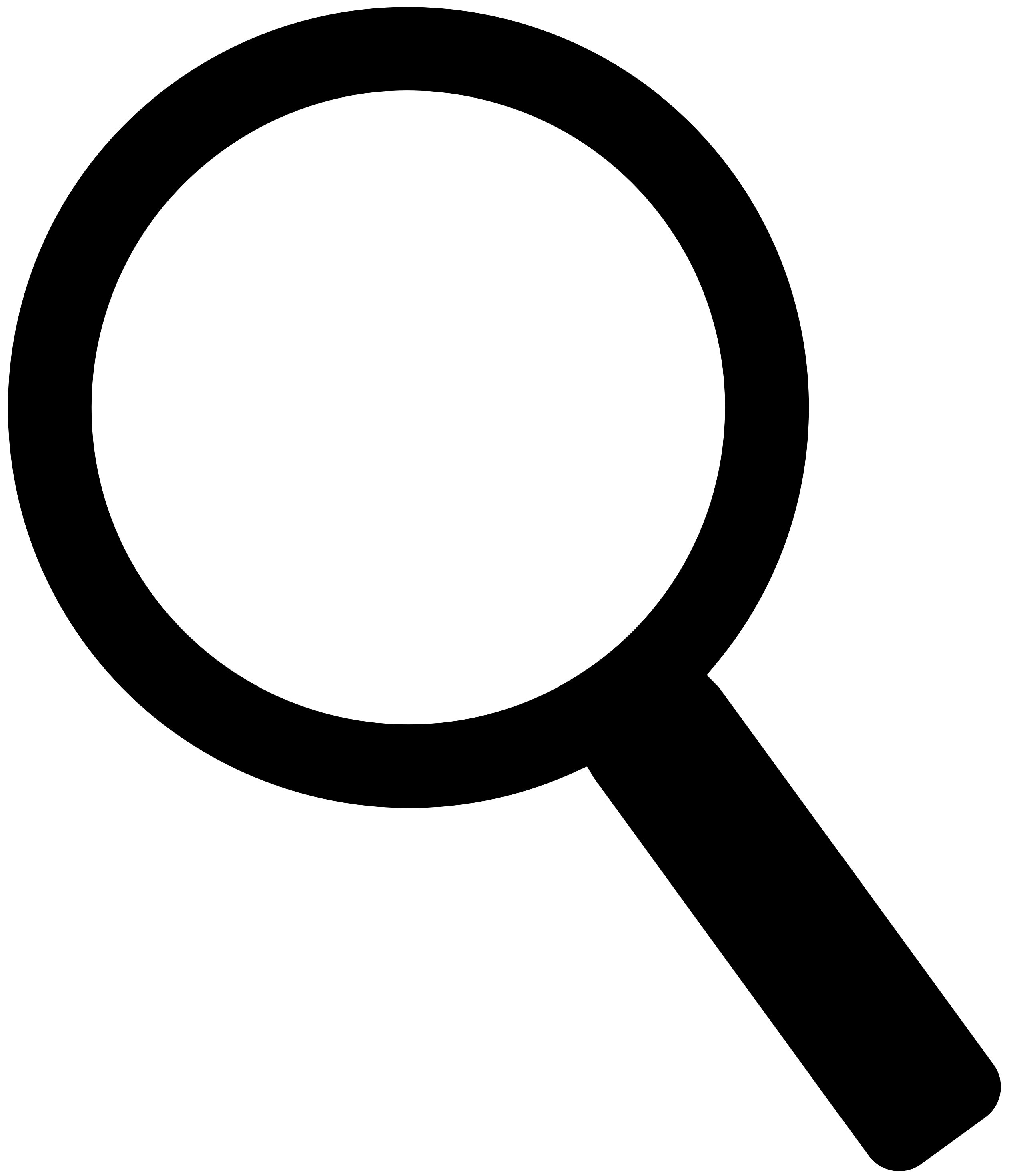
14x faster



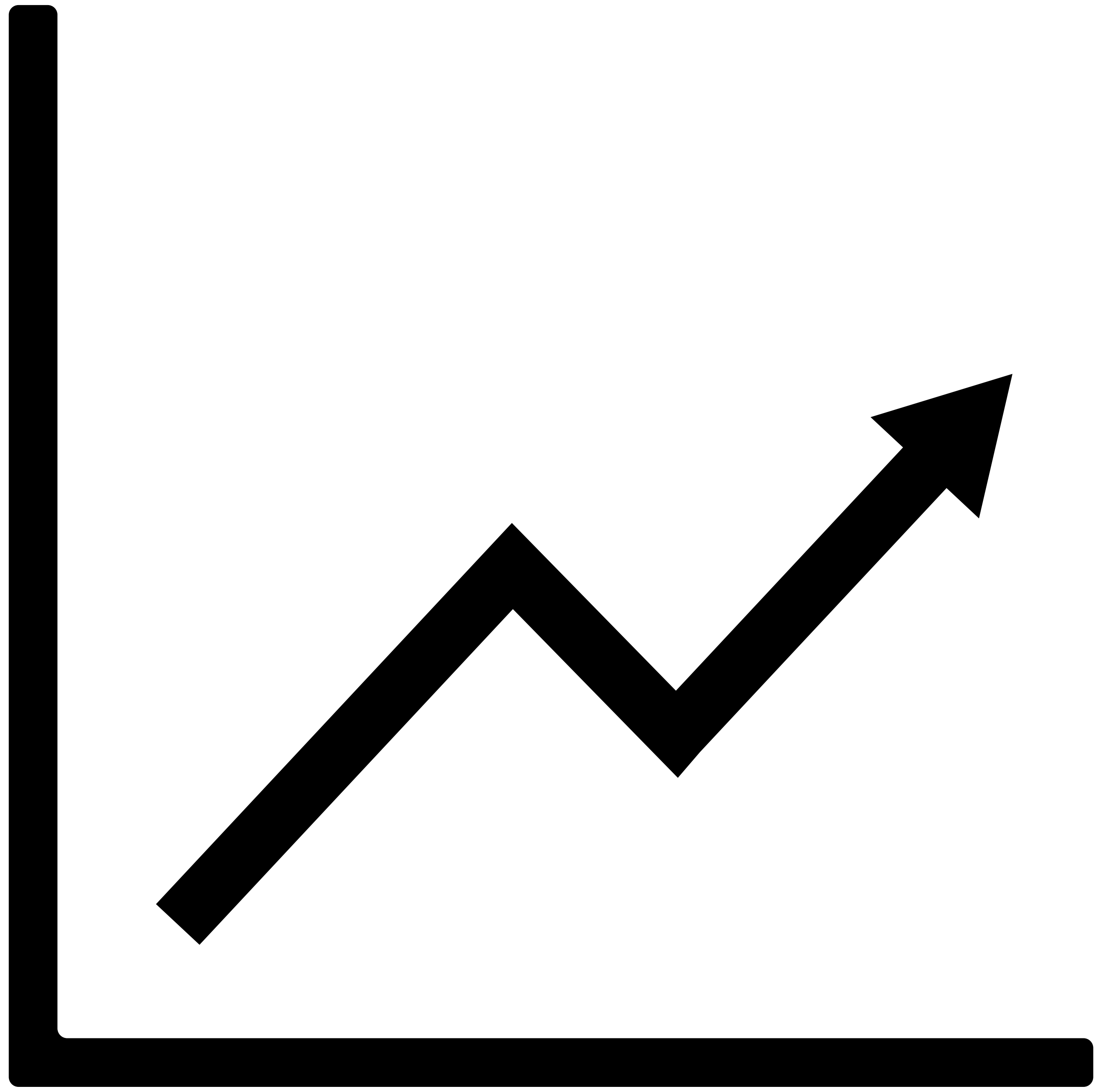
Design space

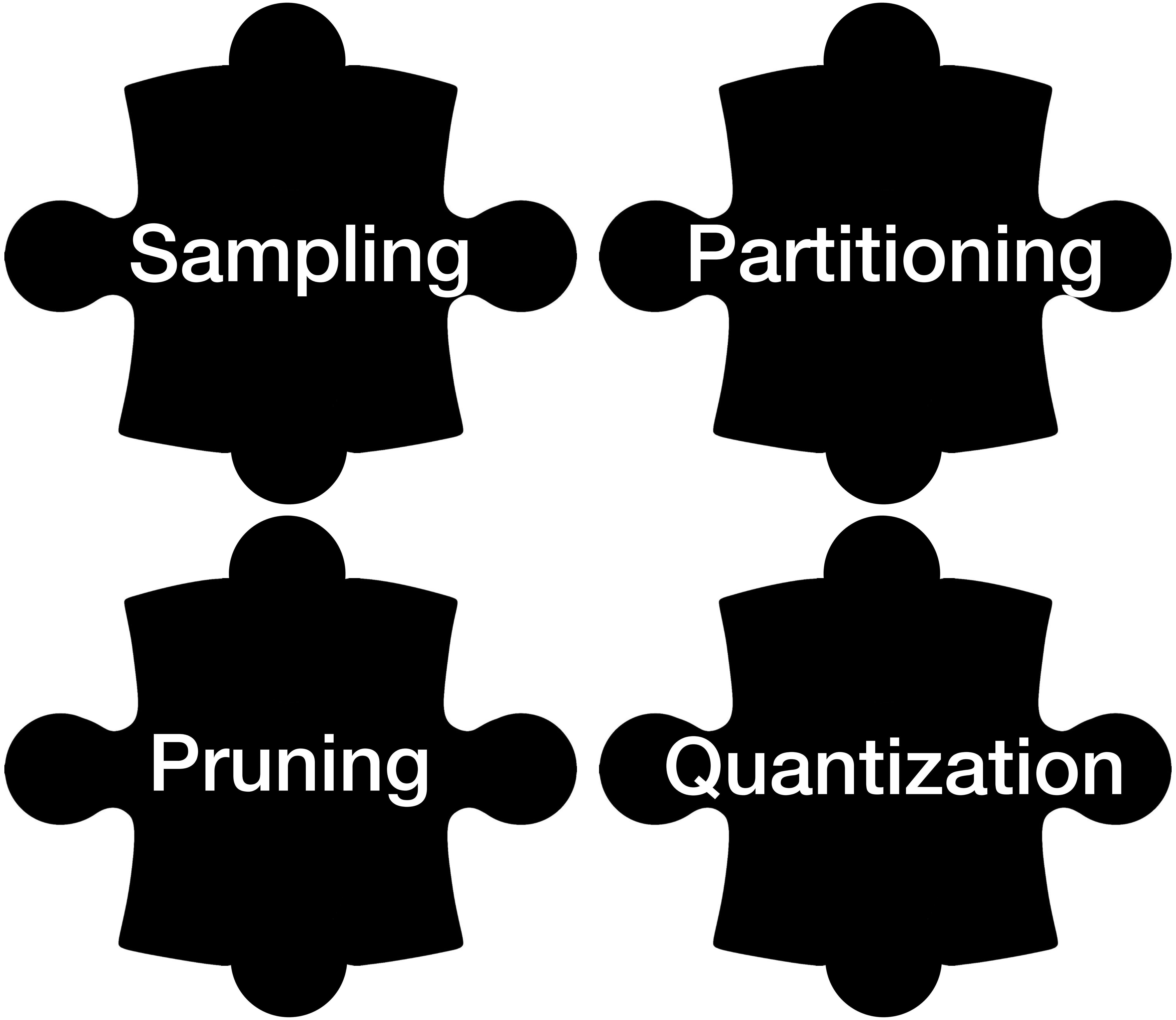


Search

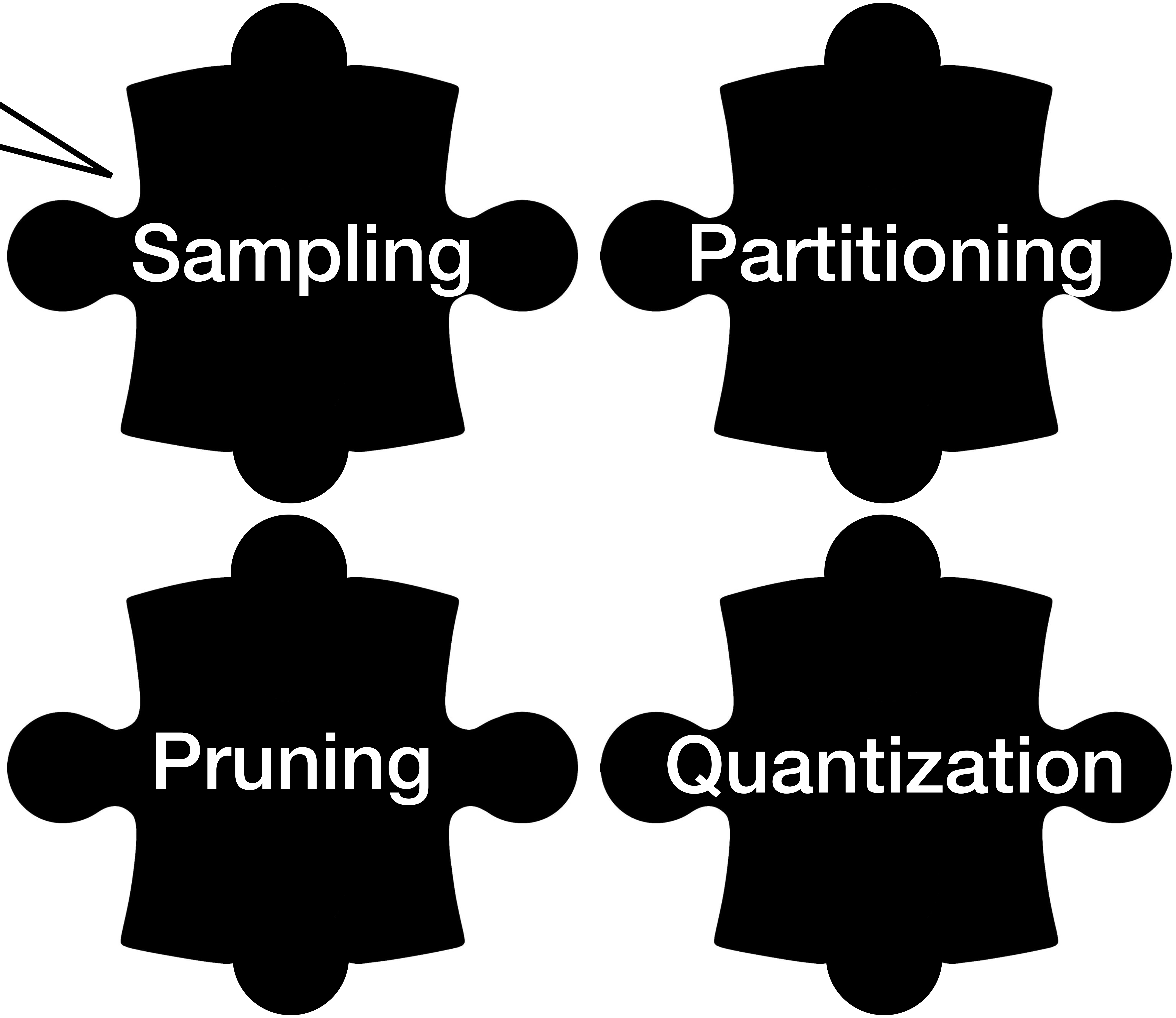


14x faster




Sampling
Partitioning
Pruning
Quantization

Remove rows/ columns



DASlab @ Harvard SEAS


Remove rows/
columns
Processing
granularity
Sampling
Partitioning
Pruning
Quantization
Remove
unuseful data


Remove rows/
columns
Processing
granularity
Sampling
Partitioning
Pruning
Quantization
Remove
unuseful data
Magnitude
reduction

Remove rows/
col

Processing
larity

10^{150K} possibilities

Re
unuseful data

itude
reduction

How can we prune the design space?

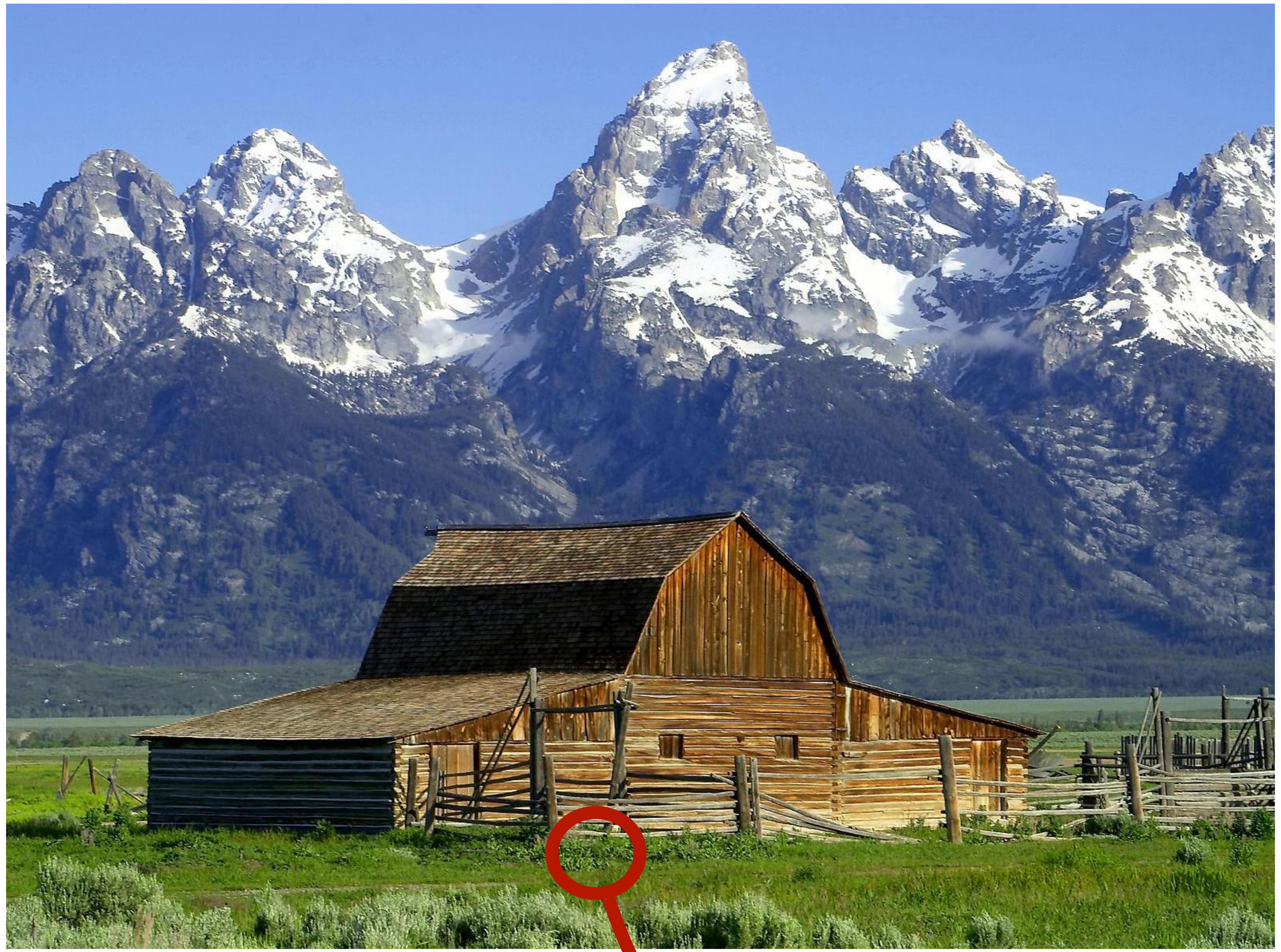
Pruning

Spatial domain

Frequency domain

102	90	80	72	152	95	70	44	82	90	60	14	92	90	50	12
76	89	72	39	99	70	65	45	80	65	40	21	82	60	32	20
56	58	49	30	78	58	30	20	77	40	24	20	60	40	30	4
54	49	32	5	40	32	20	2	70	60	59	30	20	10	8	8
200	150	100	98	180	154	134	102	90	86	82	76	120	102	64	50
180	172	160	150	120	134	103	95	85	80	72	45	90	82	70	54
170	165	120	112	103	90	83	74	70	63	53	45	80	65	55	30
92	90	70	49	94	89	80	24	60	54	40	30	70	50	30	22
112	108	100	78	90	85	80	78	110	95	85	78	104	92	82	70
97	85	65	48	82	70	65	49	98	90	76	40	95	80	70	35
90	78	74	30	78	65	54	32	68	49	31	20	68	52	32	24
72	45	24	5	74	45	30	21	31	20	10	4	52	32	15	2

Spatial domain



A pixel

Frequency domain

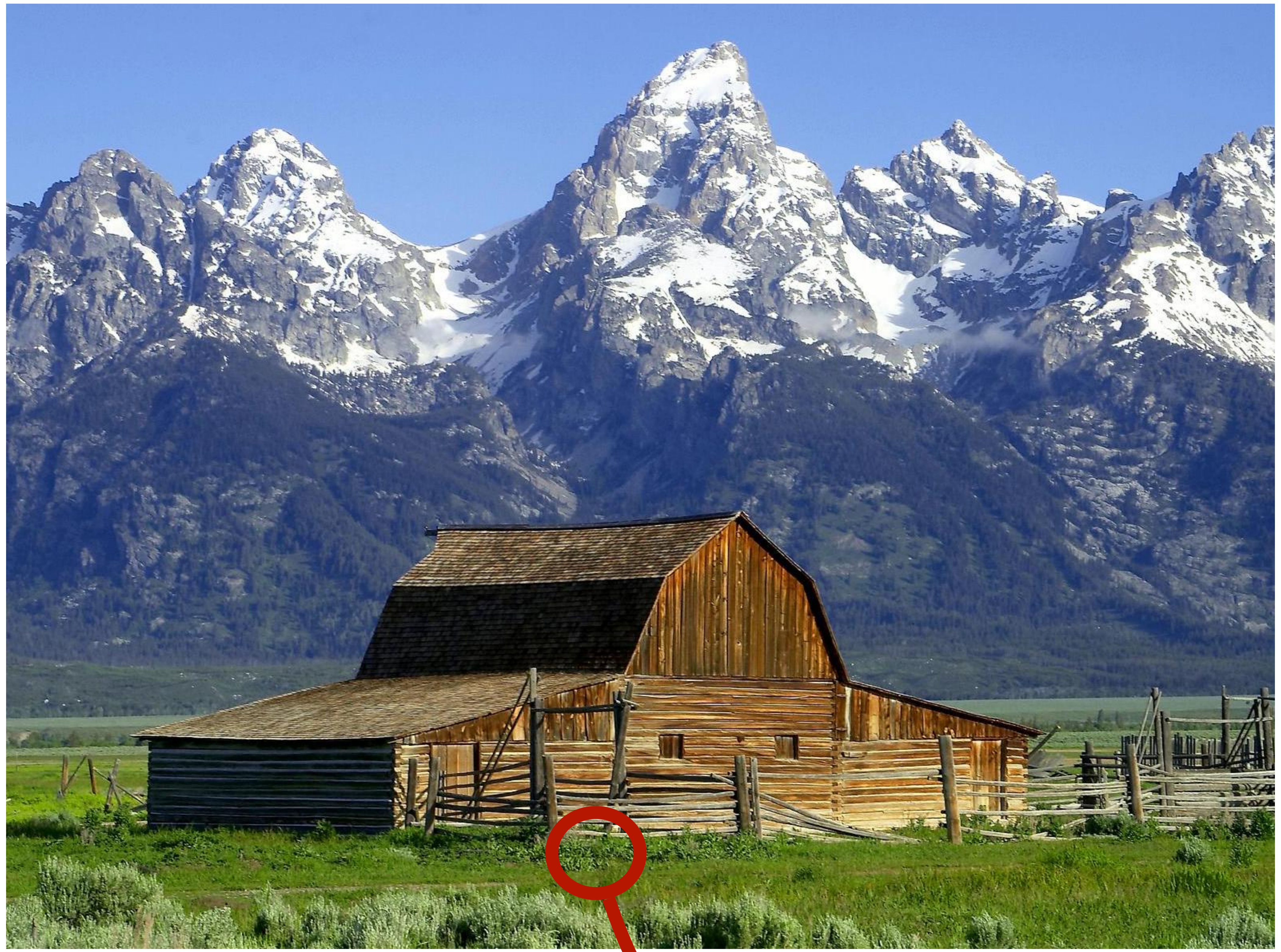
102	90	80	72	152	95	70	44	82	90	60	14	92	90	50	12
76	89	72	39	99	70	65	45	80	65	40	21	82	60	32	20
56	58	49	30	78	58	30	20	77	40	24	20	60	40	30	4
54	49	32	5	40	32	20	2	70	60	59	30	20	10	8	8
200	150	100	98	180	154	134	102	90	86	82	76	120	102	64	50
180	172	160	150	120	134	103	95	85	80	72	45	90	82	70	54
170	165	120	112	103	90	83	74	70	63	53	45	80	65	55	30
92	90	70	49	94	89	80	24	60	54	40	30	70	50	30	22
112	108	100	78	90	85	80	78	110	95	85	78	104	92	82	70
97	85	65	48	82	70	65	49	98	90	76	40	95	80	70	35
90	78	74	30	78	65	54	32	68	49	31	20	68	52	32	24
72	45	24	5	74	45	30	21	31	20	10	4	52	32	15	2

A frequency coefficient

DASlab @ Harvard SEAS

A block

Spatial domain



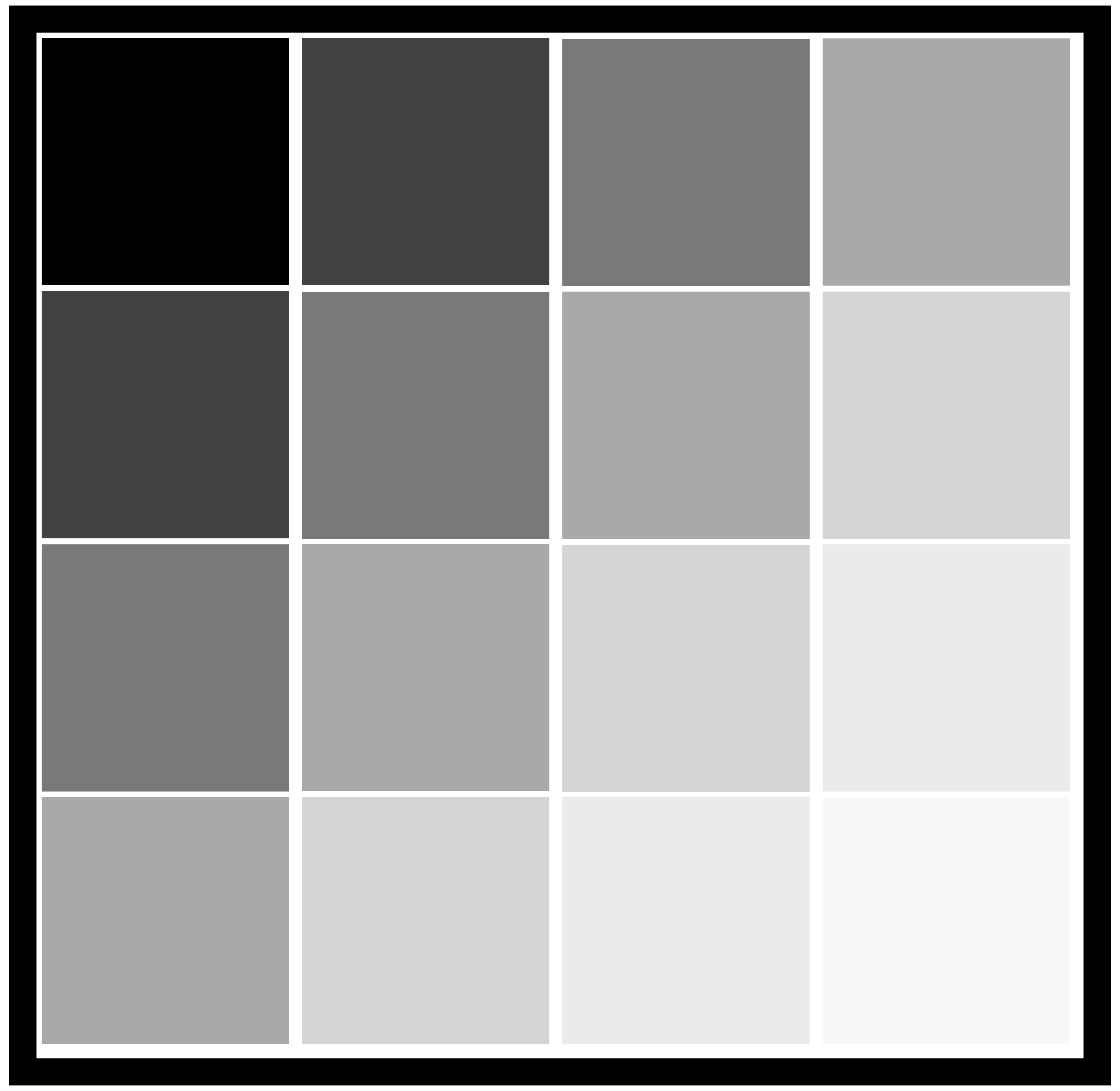
Frequency domain

102	90	80	72	52	95	70	44	82	90	60	14	92	90	50	12
76	89	72	39	99	70	65	45	80	65	40	21	82	60	32	20
56	58	49	30	78	58	30	20	77	40	24	20	60	40	30	4
54	49	32	5	40	32	20	2	70	60	59	30	20	10	8	8
200	150	100	98	180	154	134	102	90	86	82	76	120	102	64	50
180	172	160	150	120	134	103	95	85	80	72	45	90	82	70	54
170	165	120	112	103	90	83	74	70	63	53	45	80	65	55	30
92	90	70	49	94	89	80	24	60	54	40	30	70	50	30	22
112	108	100	78	90	85	80	78	110	95	85	78	104	92	82	70
97	85	65	48	82	70	65	49	98	90	76	40	95	80	70	35
90	78	74	30	78	65	54	32	68	49	31	20	68	52	32	24
72	45	24	5	74	45	30	21	31	20	10	4	52	32	15	2

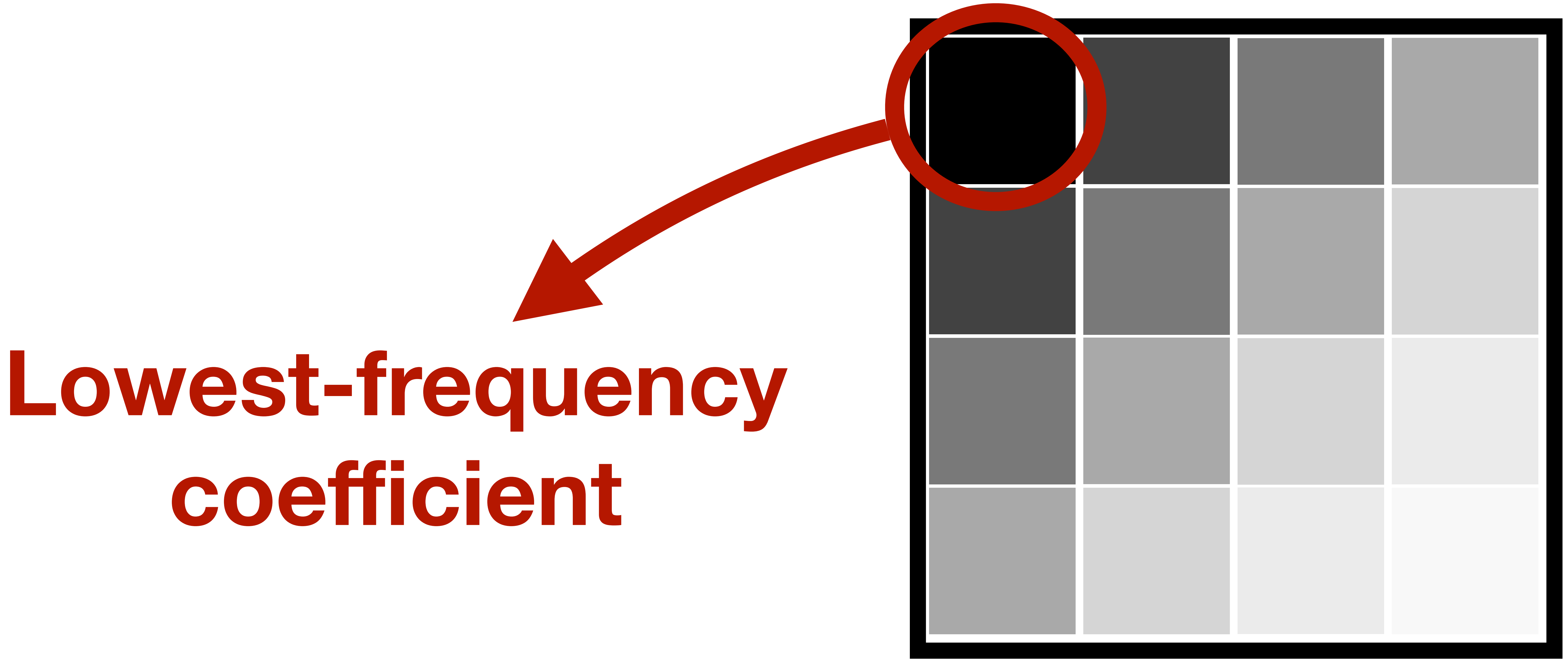
A pixel

A frequency coefficient

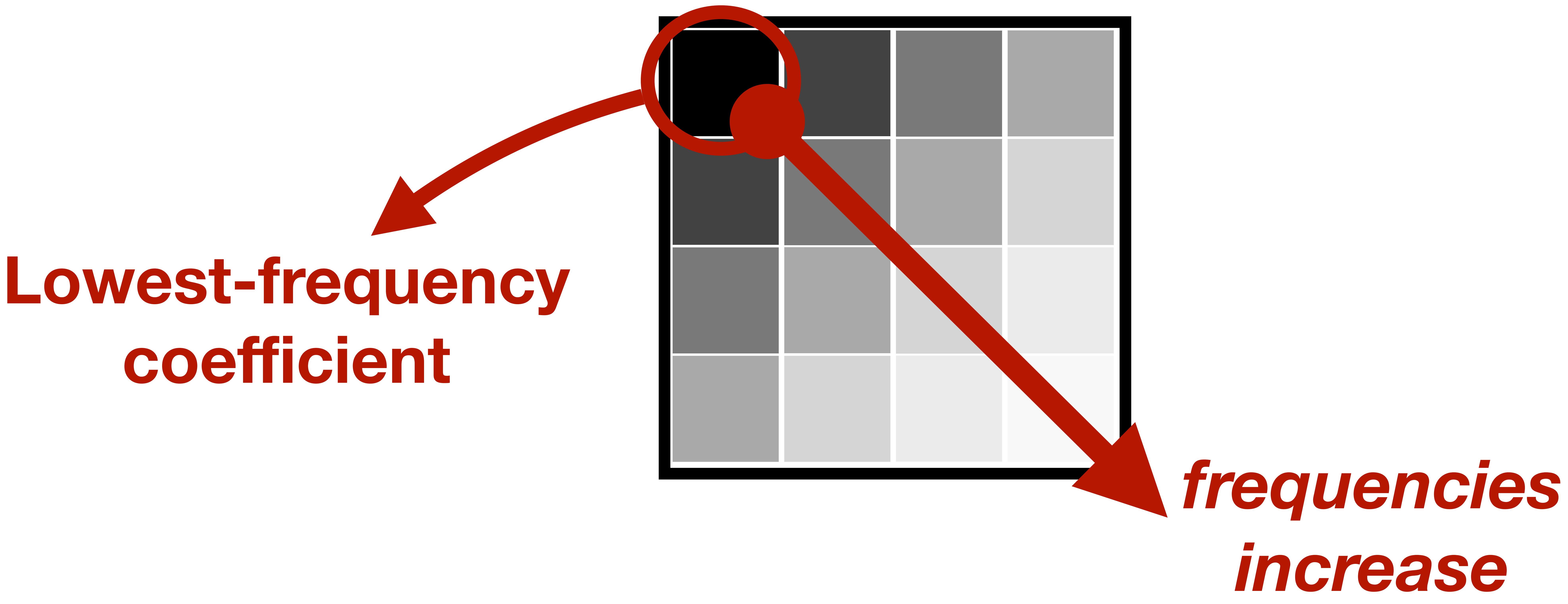
A single block



A single block



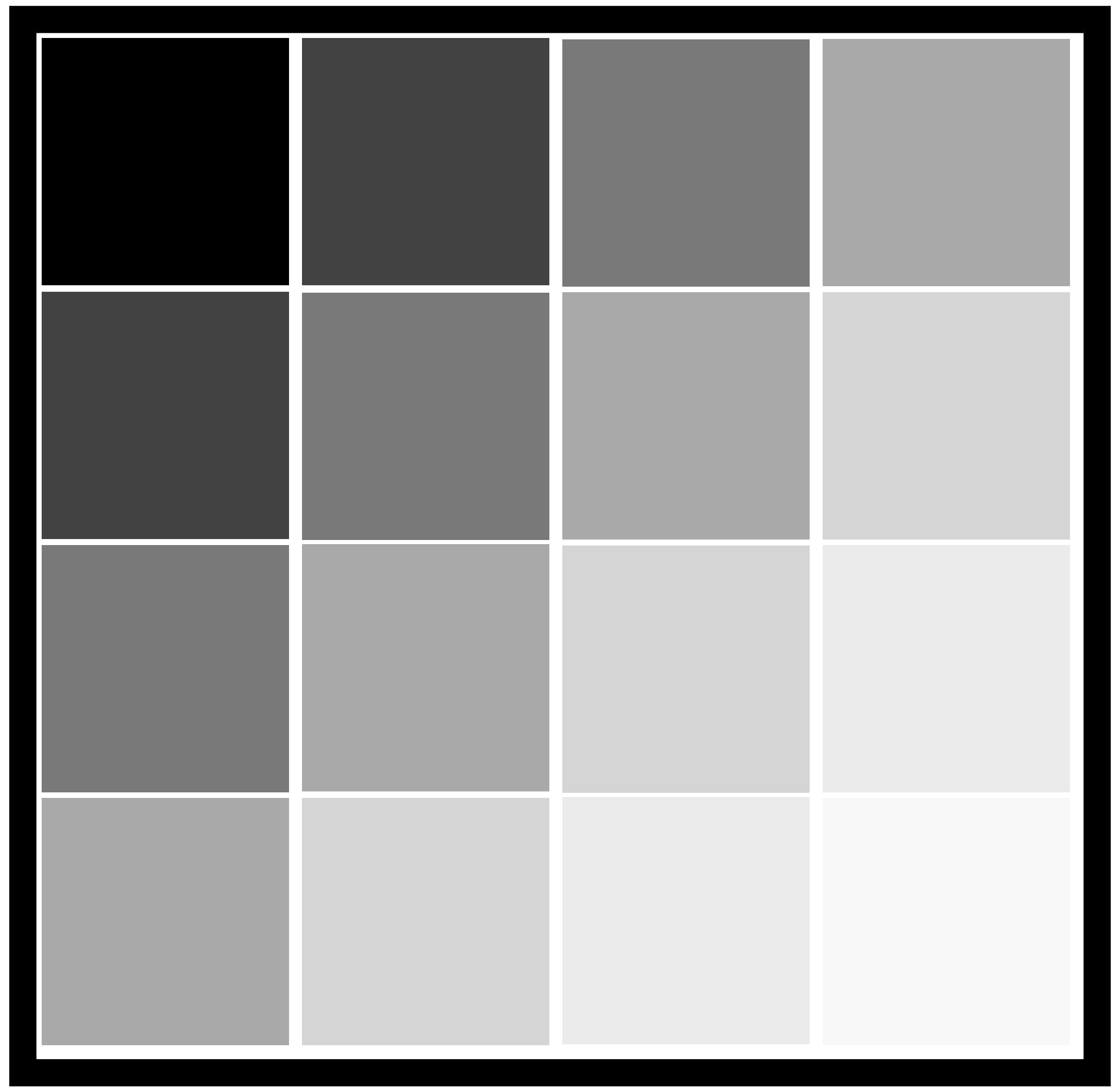
A single block



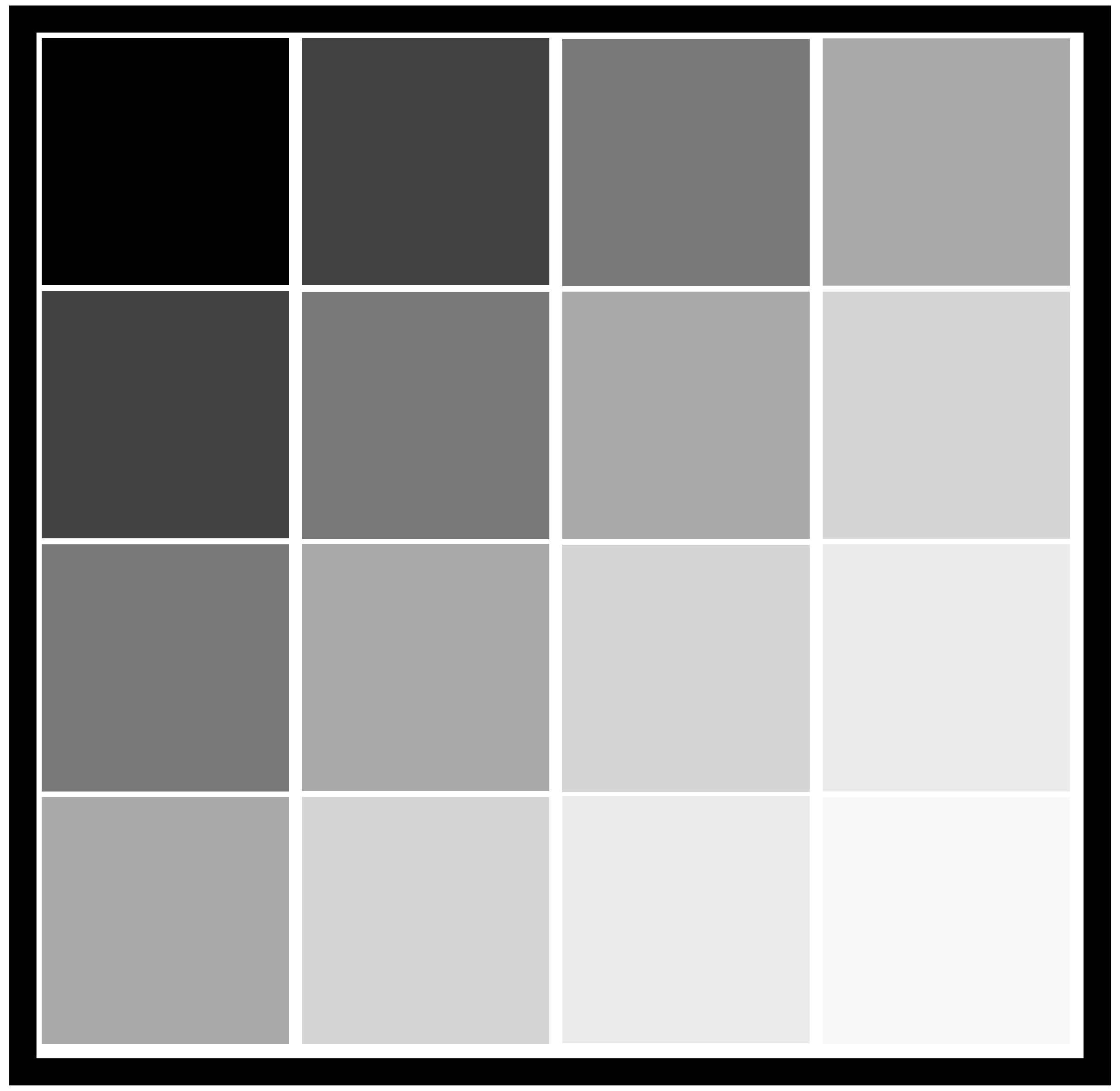
Pruning strategy #1

Pruning strategy #2

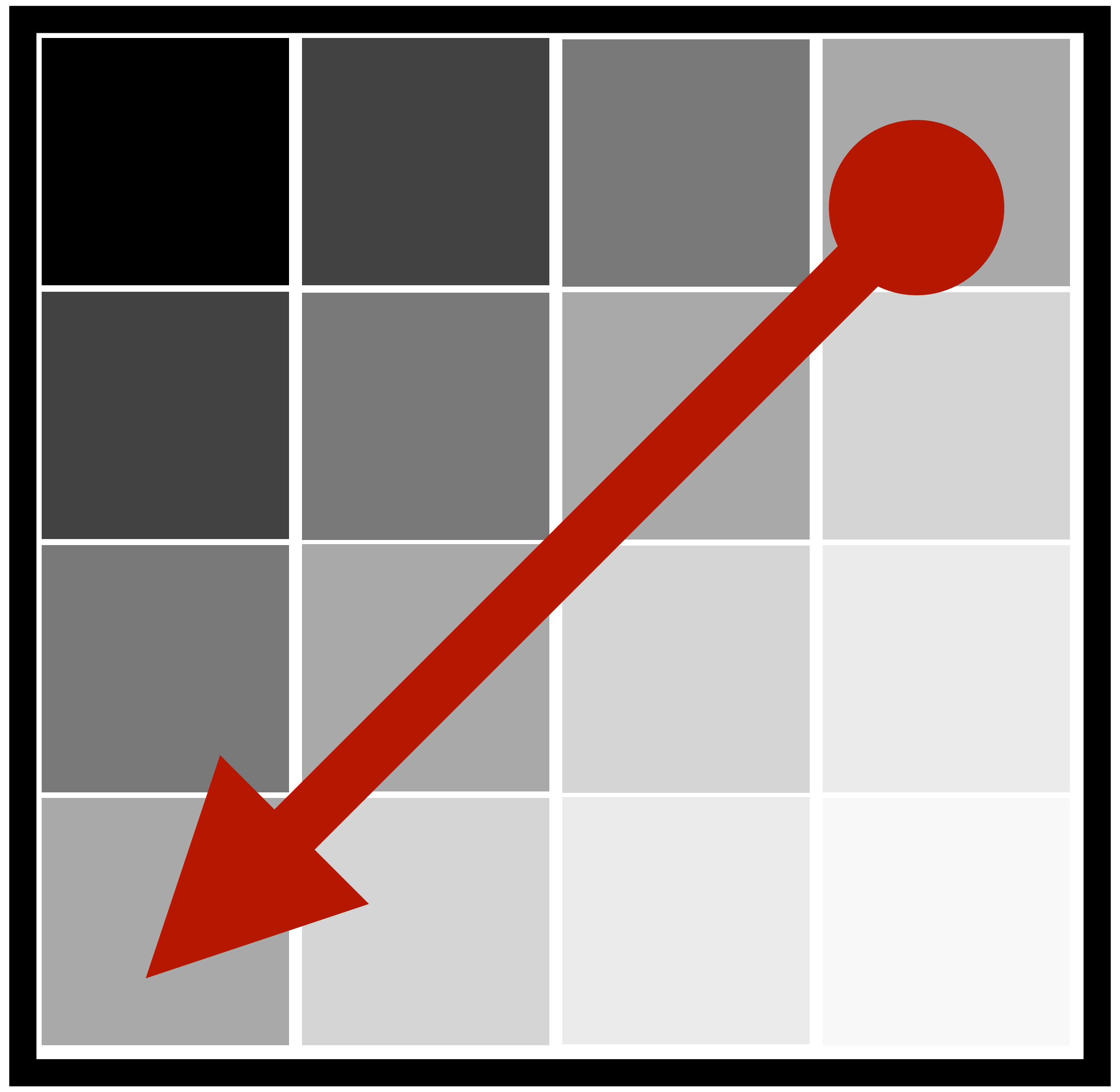
Strategy 3



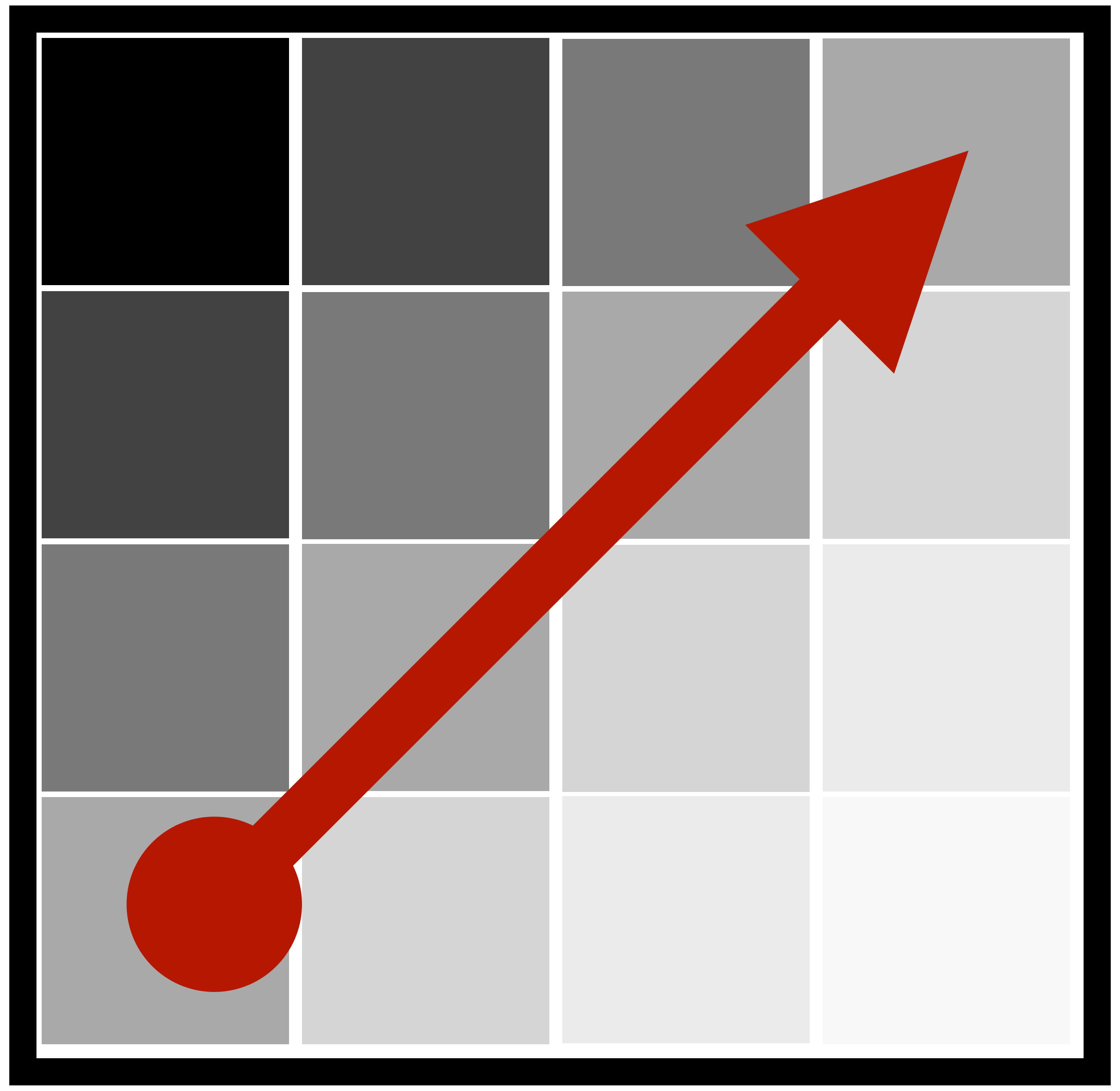
Strategy 4



Strategy 3



Strategy 4



DASlab
@ Harvard SEAS

ResNet50, A100, PyTorch v1

DASlab @ Harvard SEAS

Strategy 1
Strategy 2
Strategy 3
Strategy 4
Accuracy
1
0.5
0
ImageNet-5c
Blood-cell
Weather
0
256
512
768
1024
Number of coefficients

Sampling

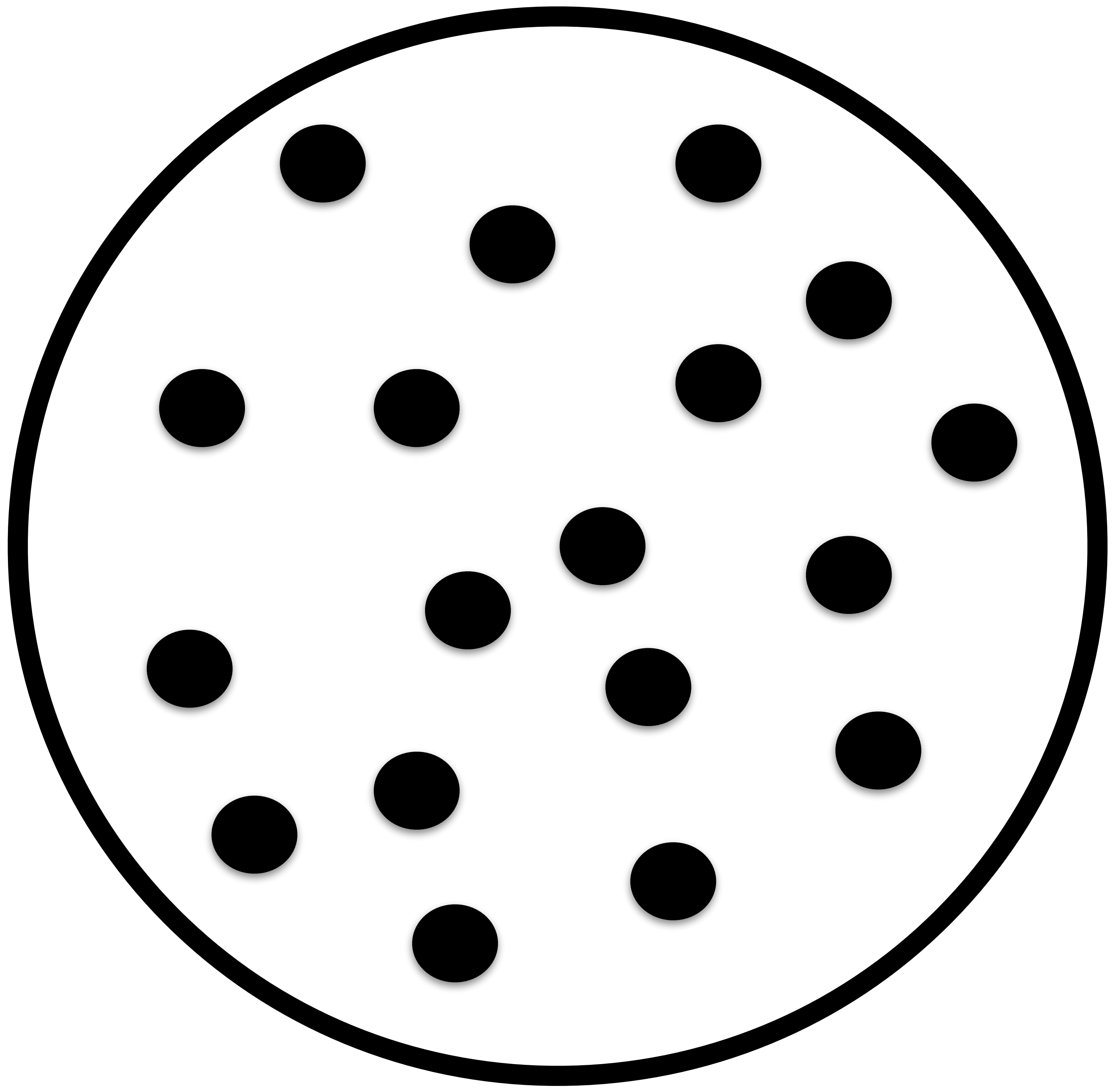
Partitioning

Pruning

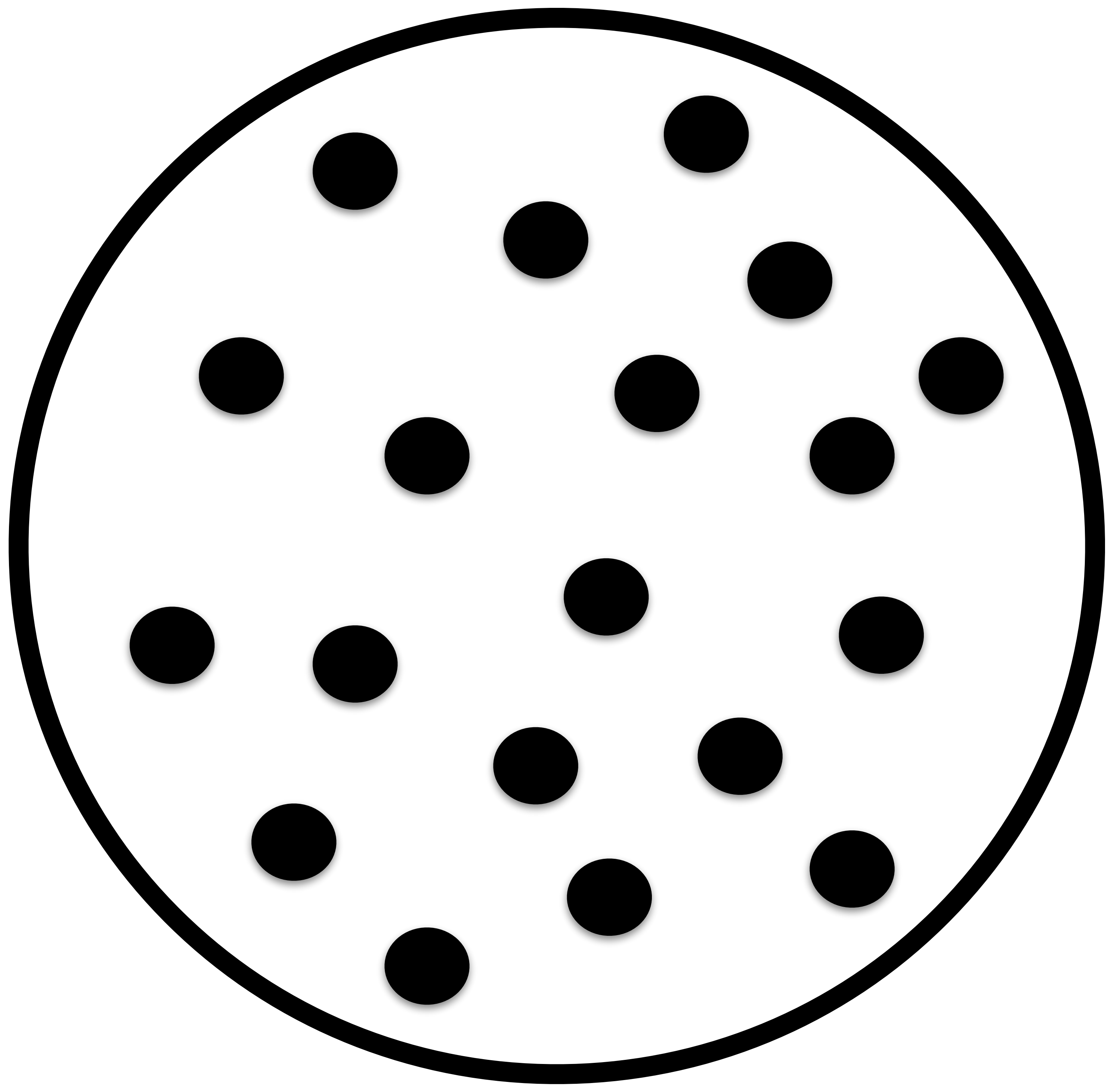
Quantization

$$10^{150K} \rightarrow 6048$$

6048 new designs

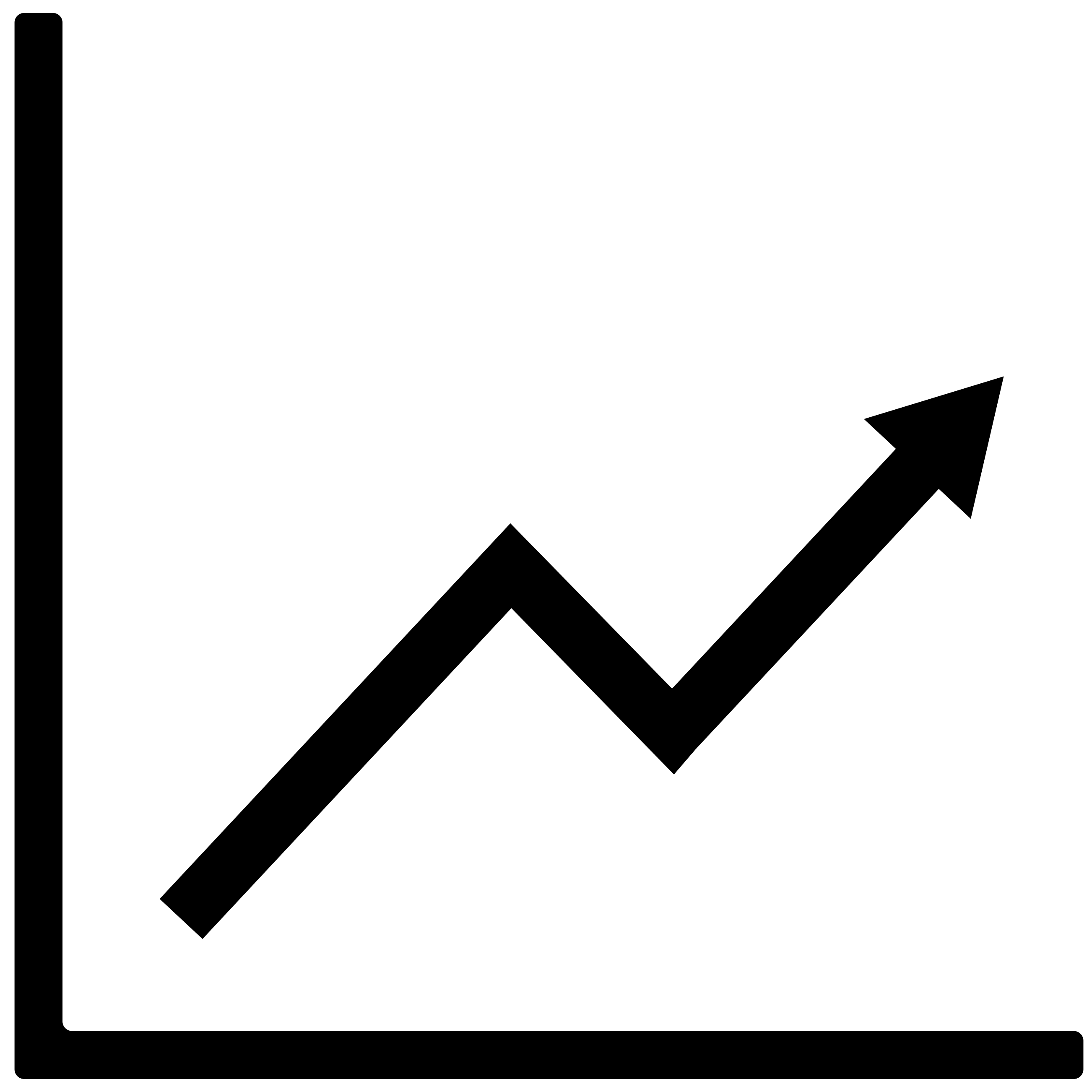


Design space

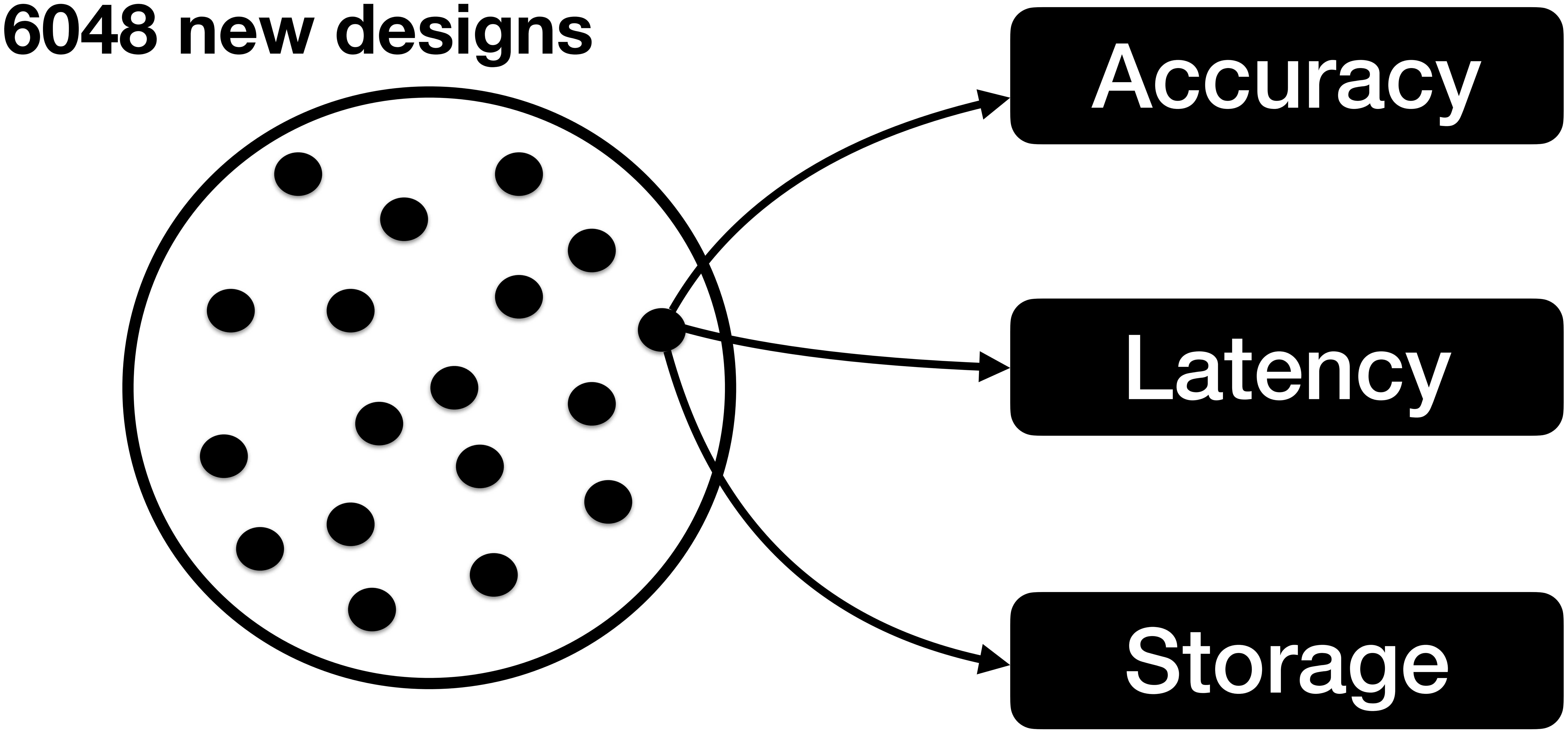


Search

14x faster



Performance models



Accuracy model

sort

Most-compressed to least-compressed

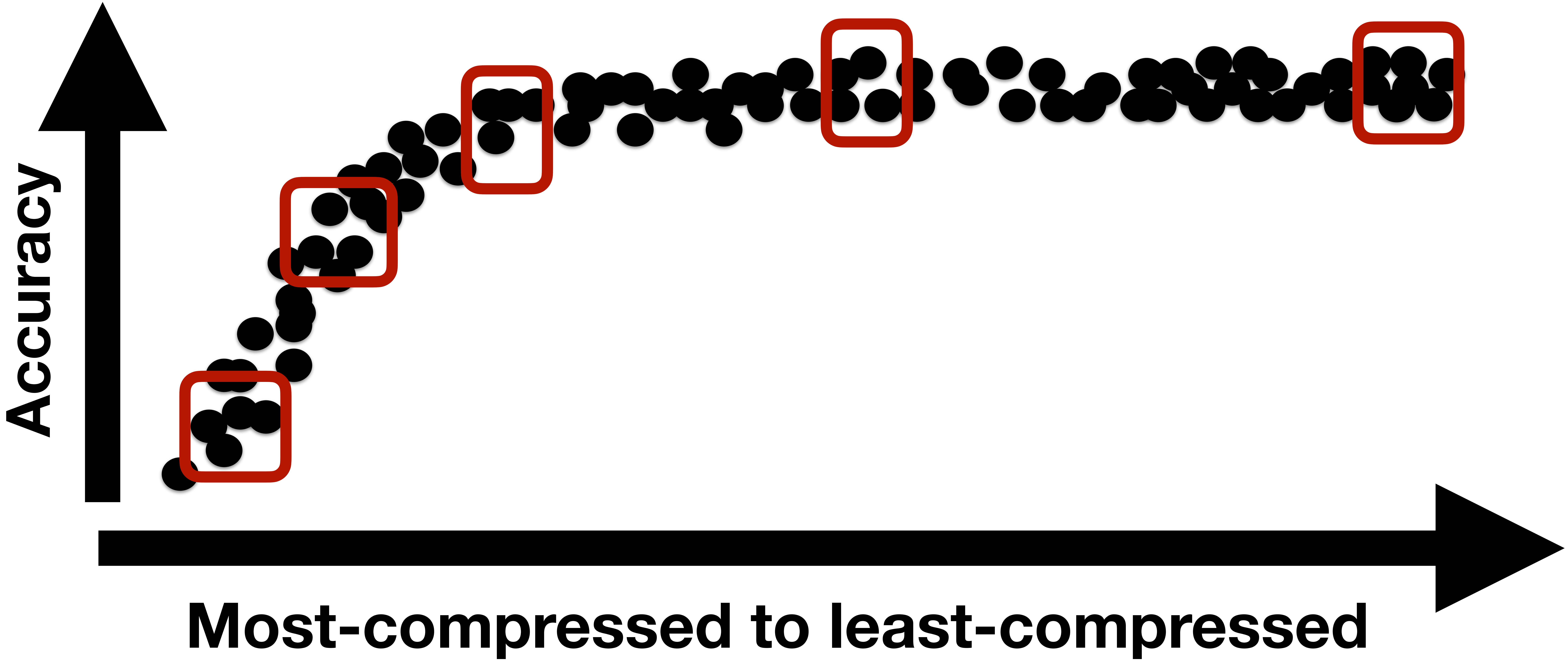
sort & analyze

ImageNet-5c, ResNet50
A100, PyTorch v1

@ Harvard SEAS

sort & analyze — sample & interpolate

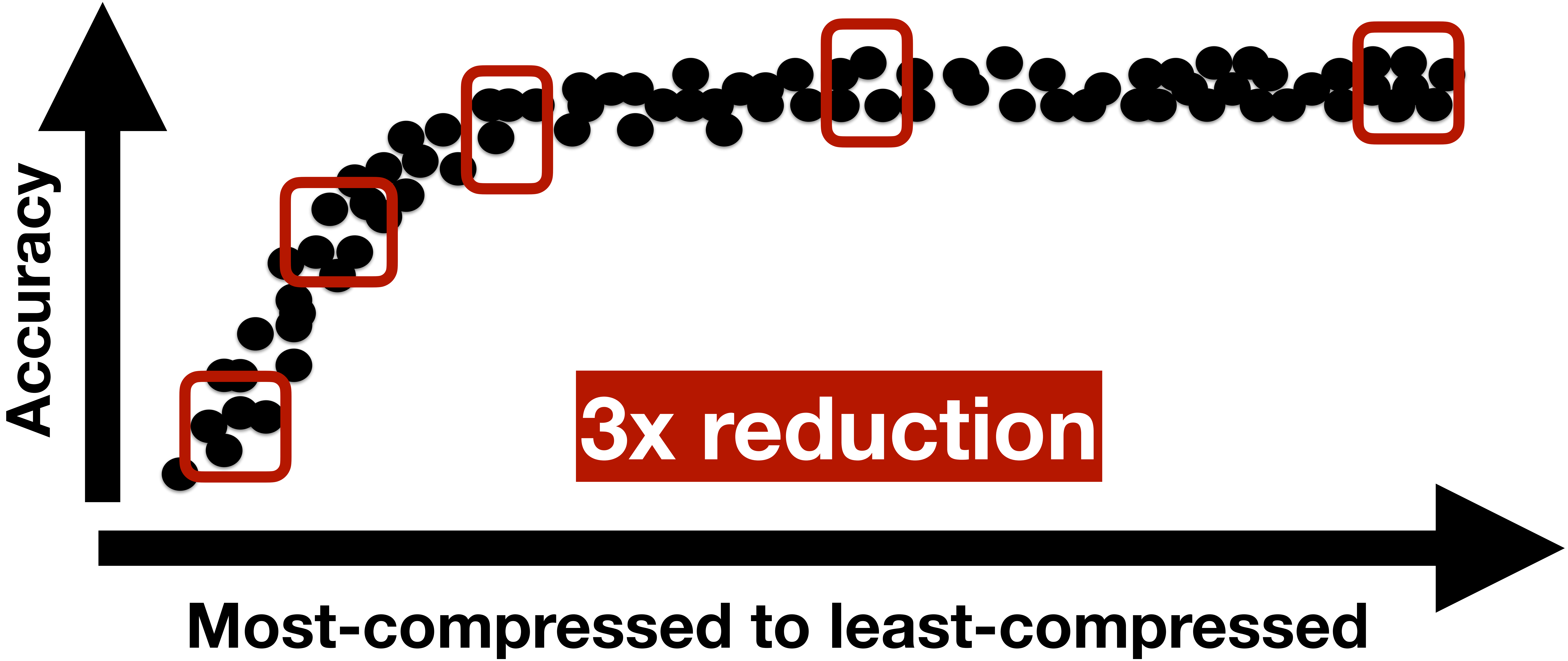
ImageNet, ResNet50
A100, PyTorch v1



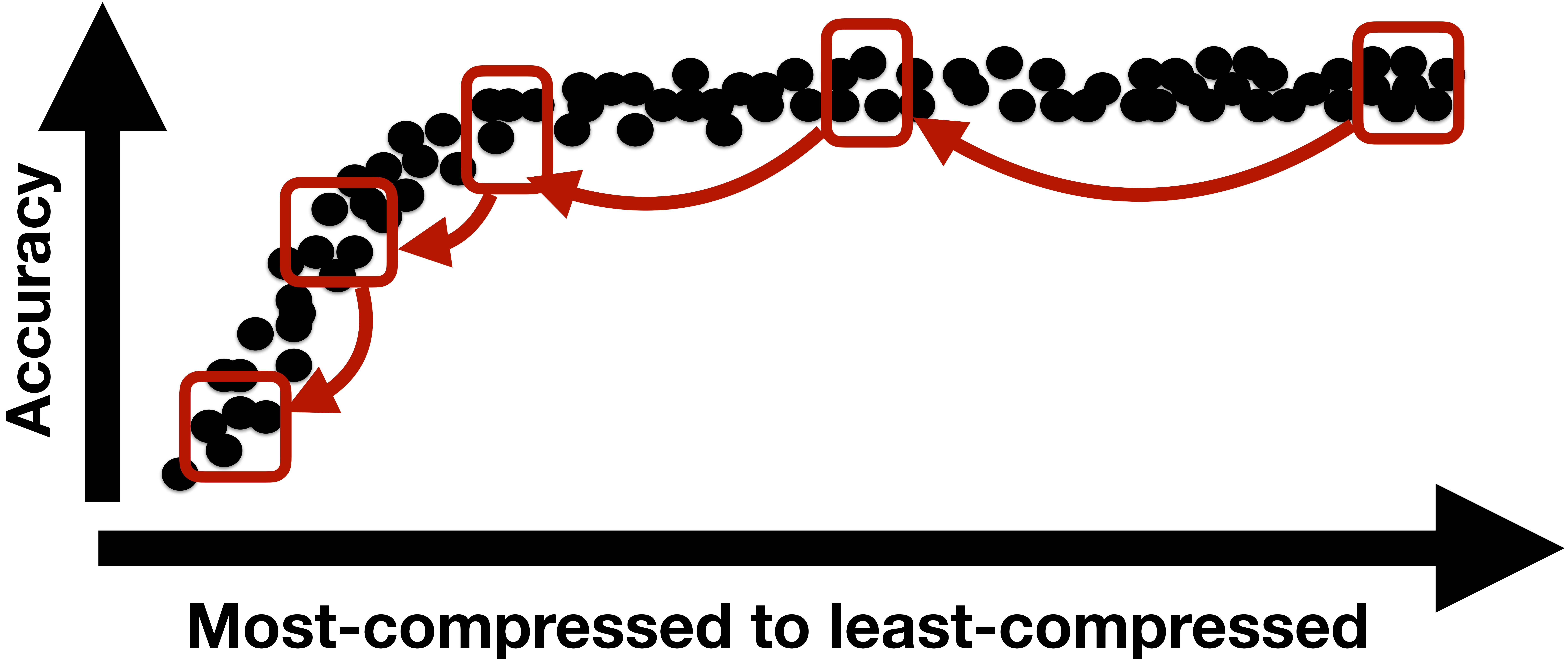
DASlab @ Harvard SEAS

sort & analyze — sample & interpolate

ImageNet, ResNet50
A100, PyTorch v1

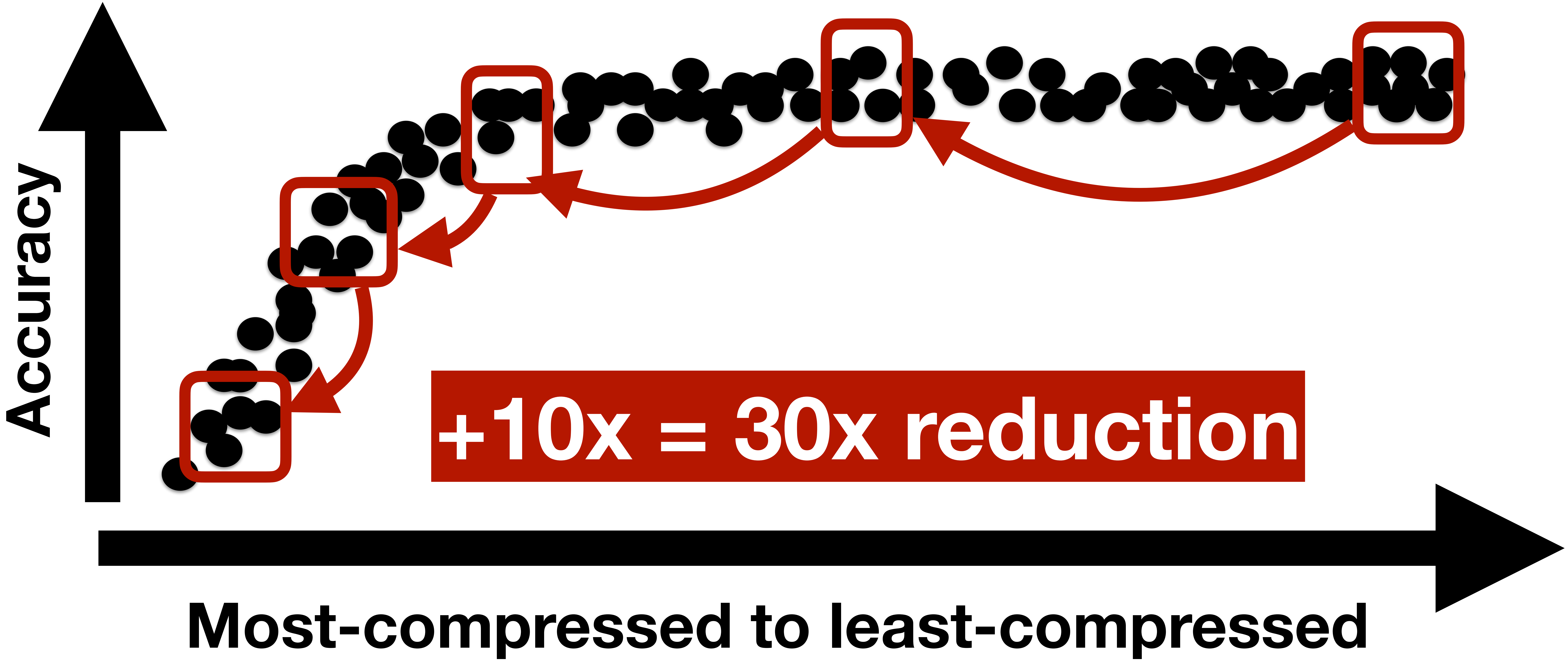


sort & analyze — sample & interpolate + transfer learn



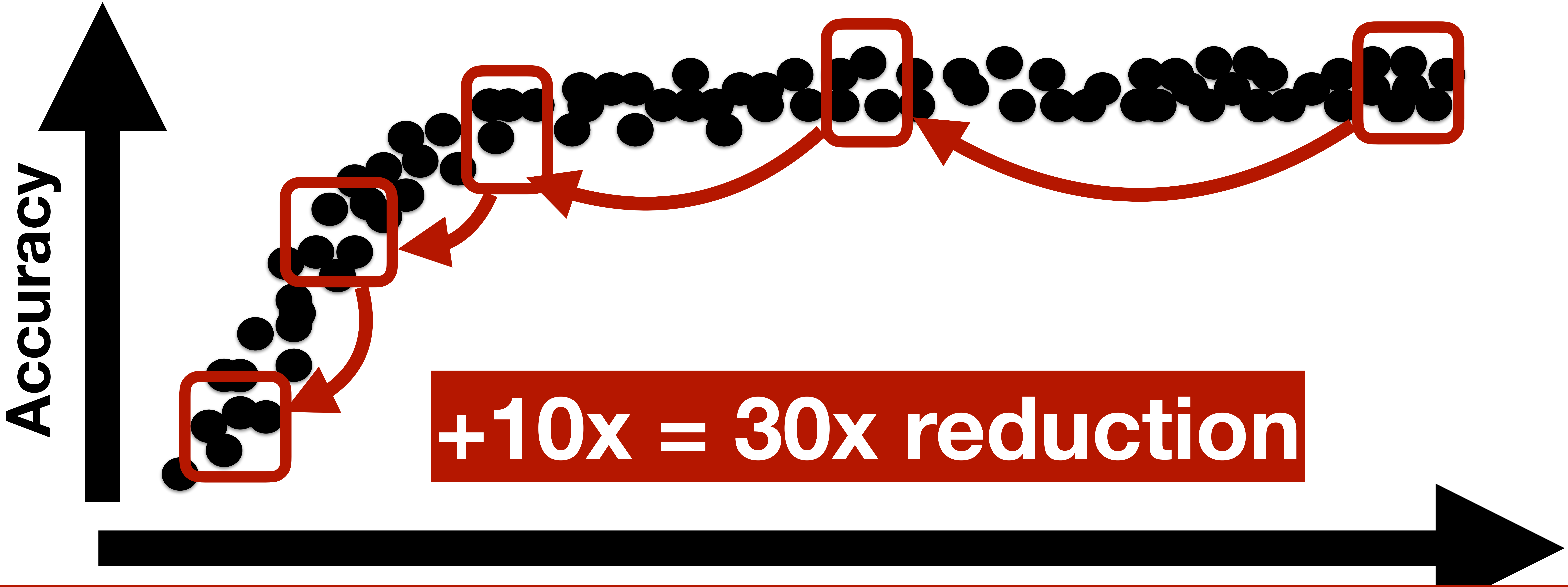
DASlab
@ Harvard SEAS

sort & analyze — sample & interpolate + transfer learn



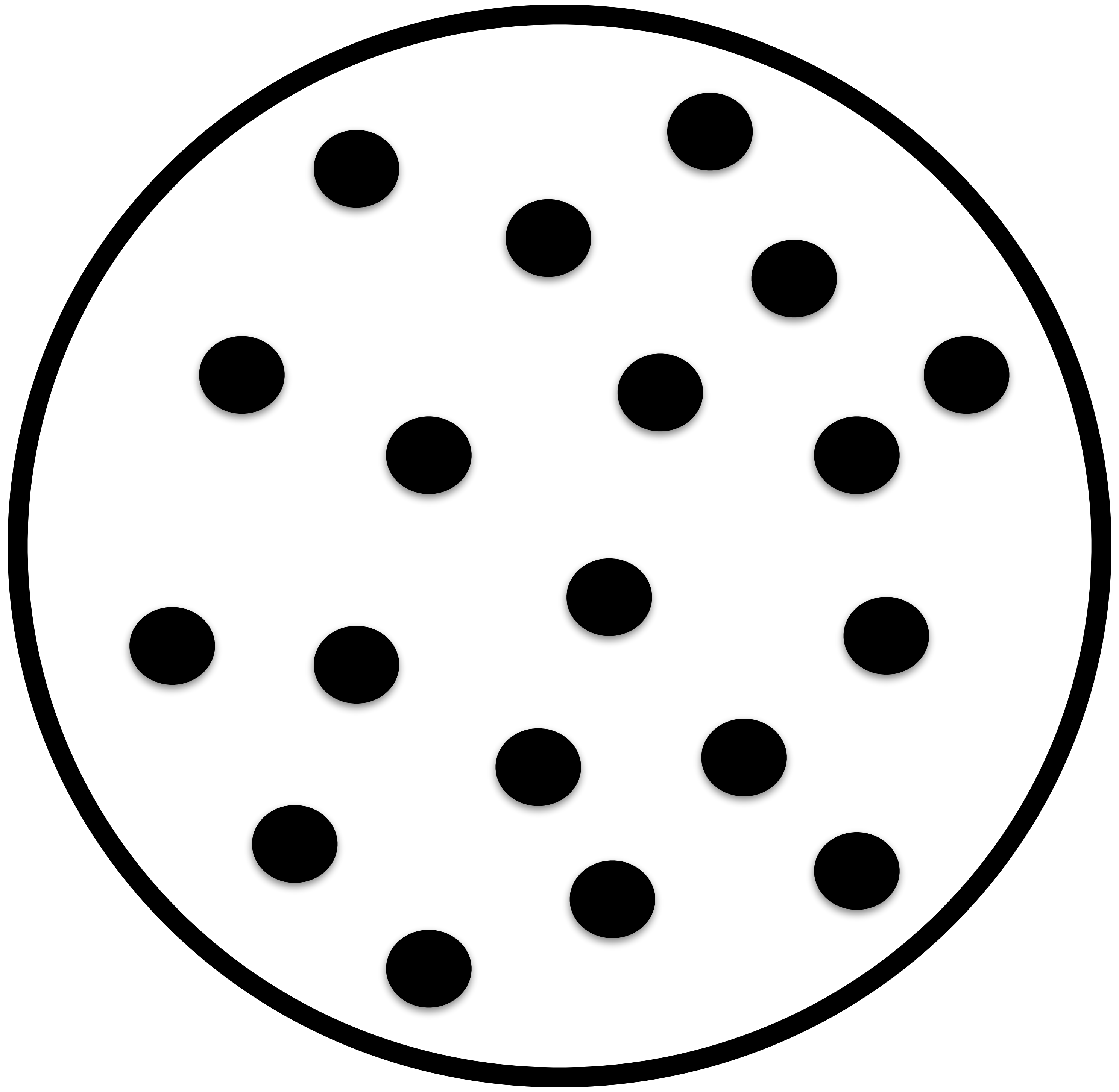
DASlab
@ Harvard SEAS

sort & analyze — sample & interpolate + transfer learn

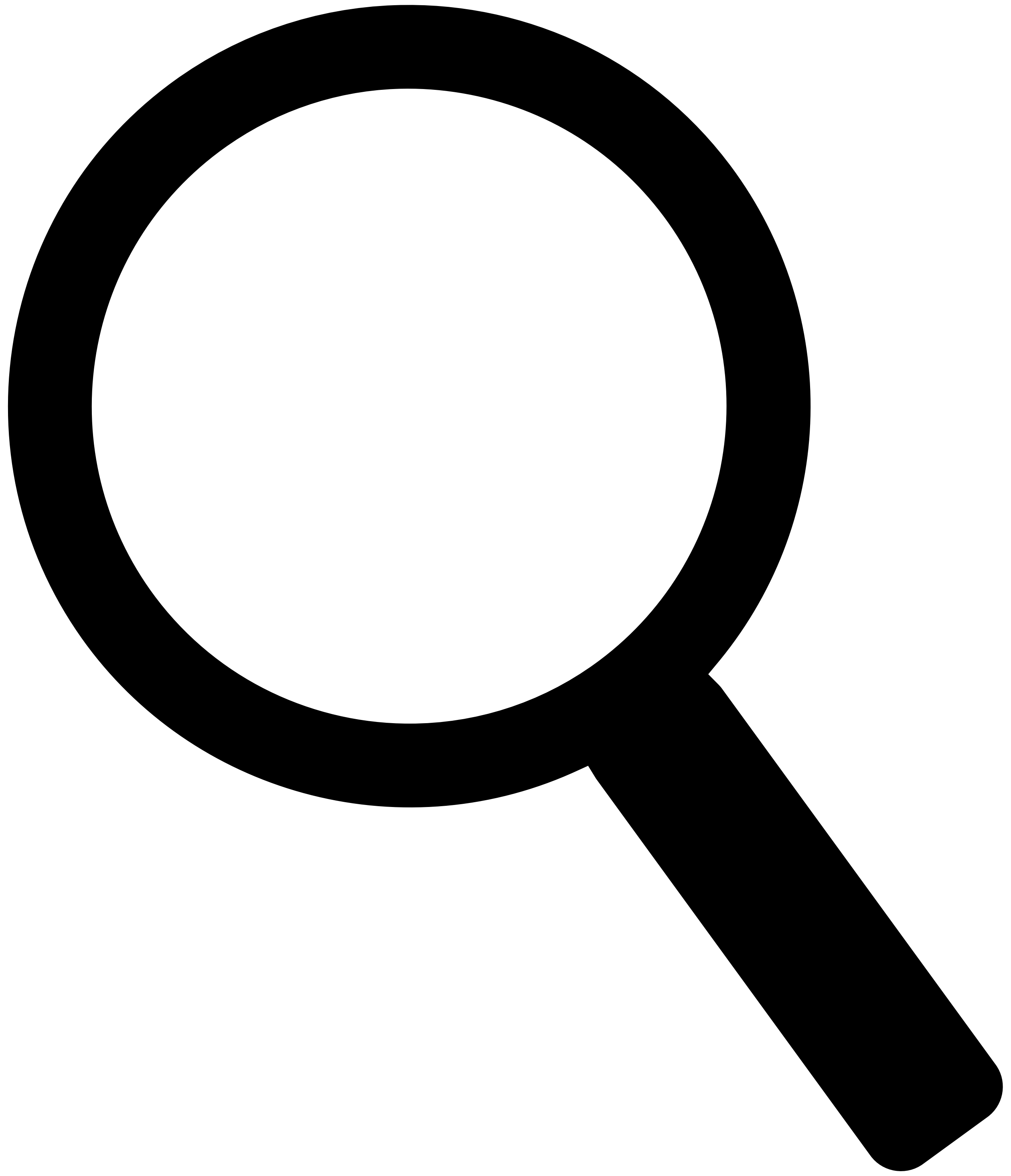


6048 → 200 training

Design space



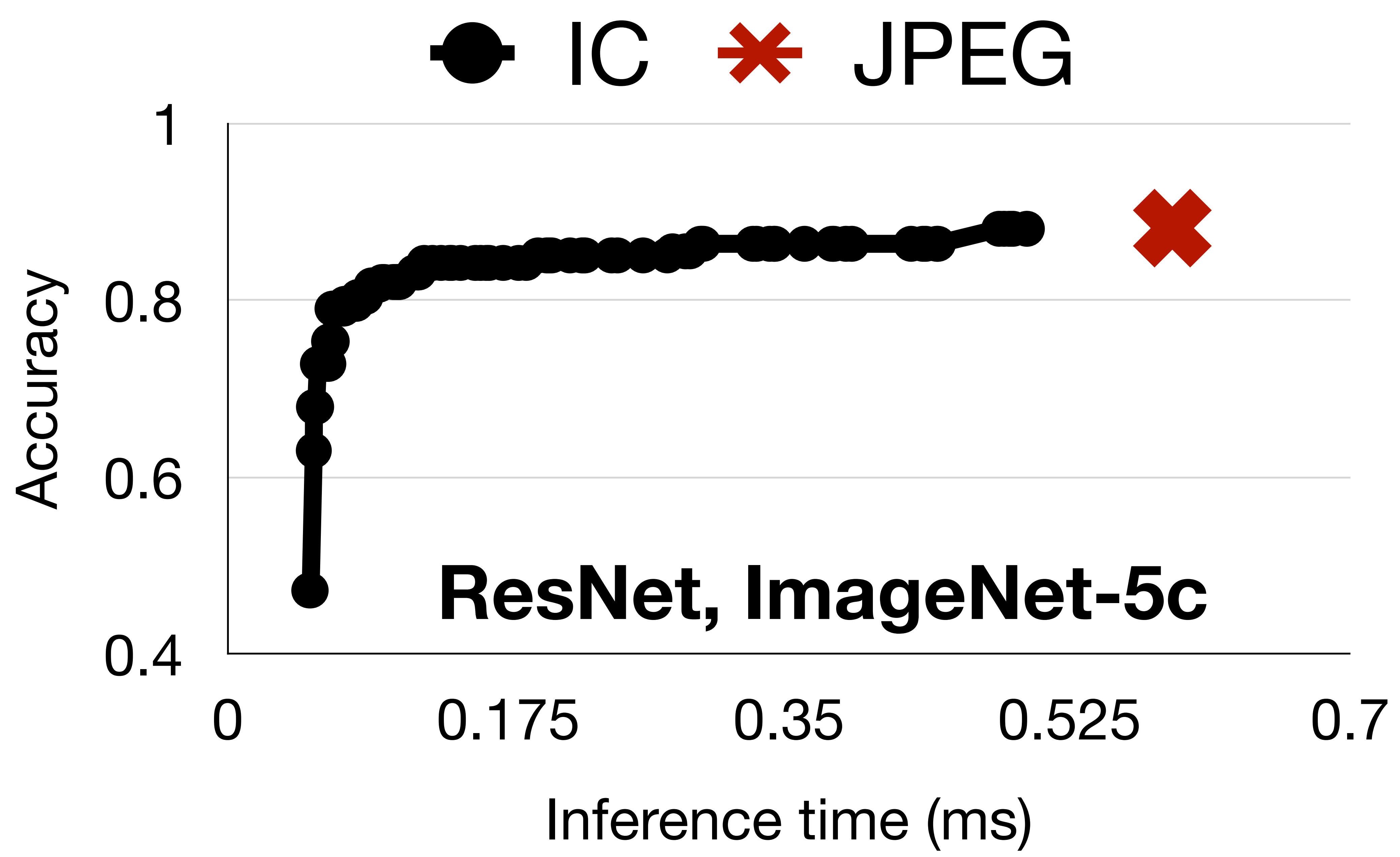
Search



14x faster

IC brings benefits on diverse datasets

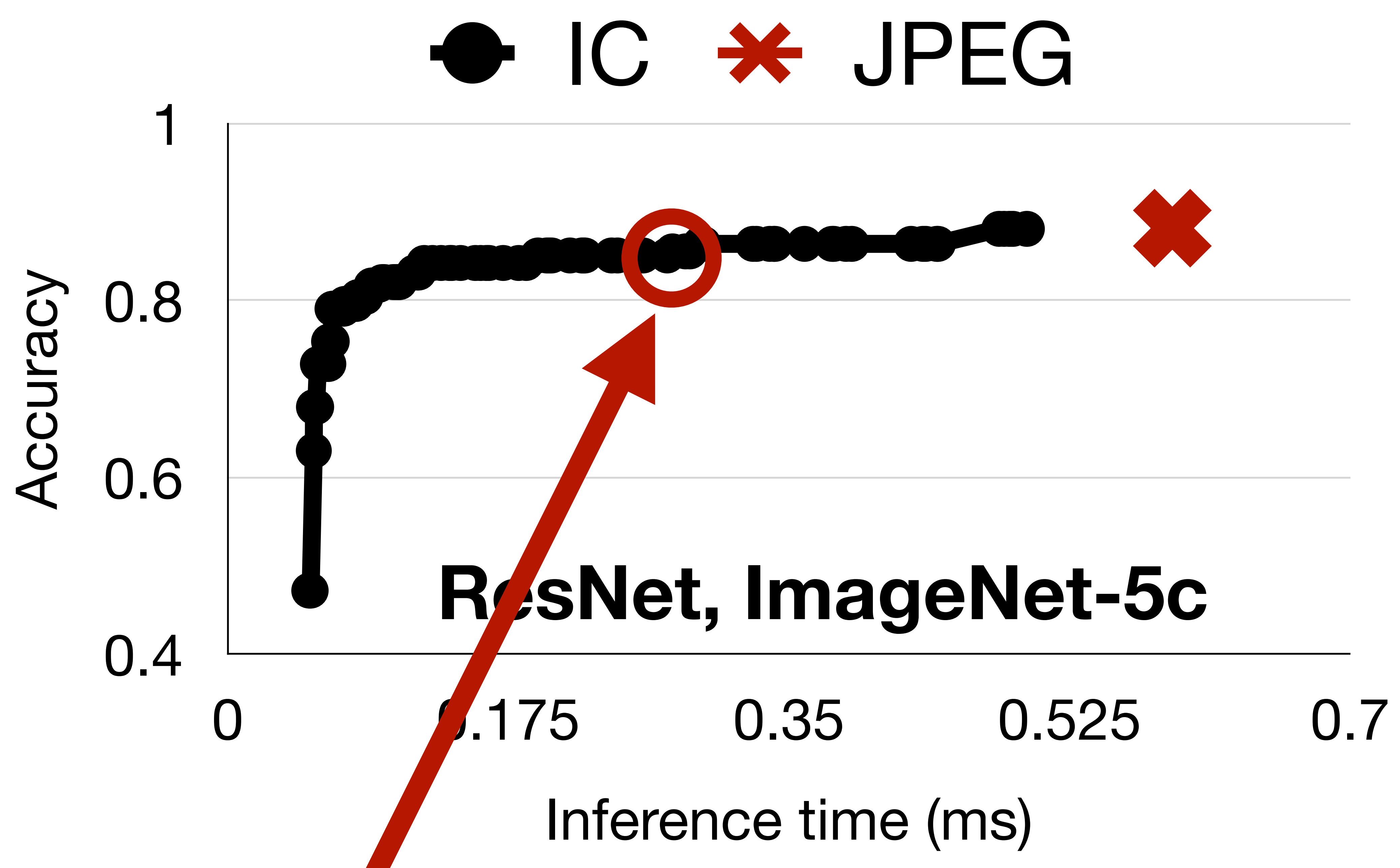
ResNet50
A100
PyTorch v1



DASlab @ Harvard SEAS

IC brings benefits on diverse datasets

ResNet50
A100
PyTorch v1

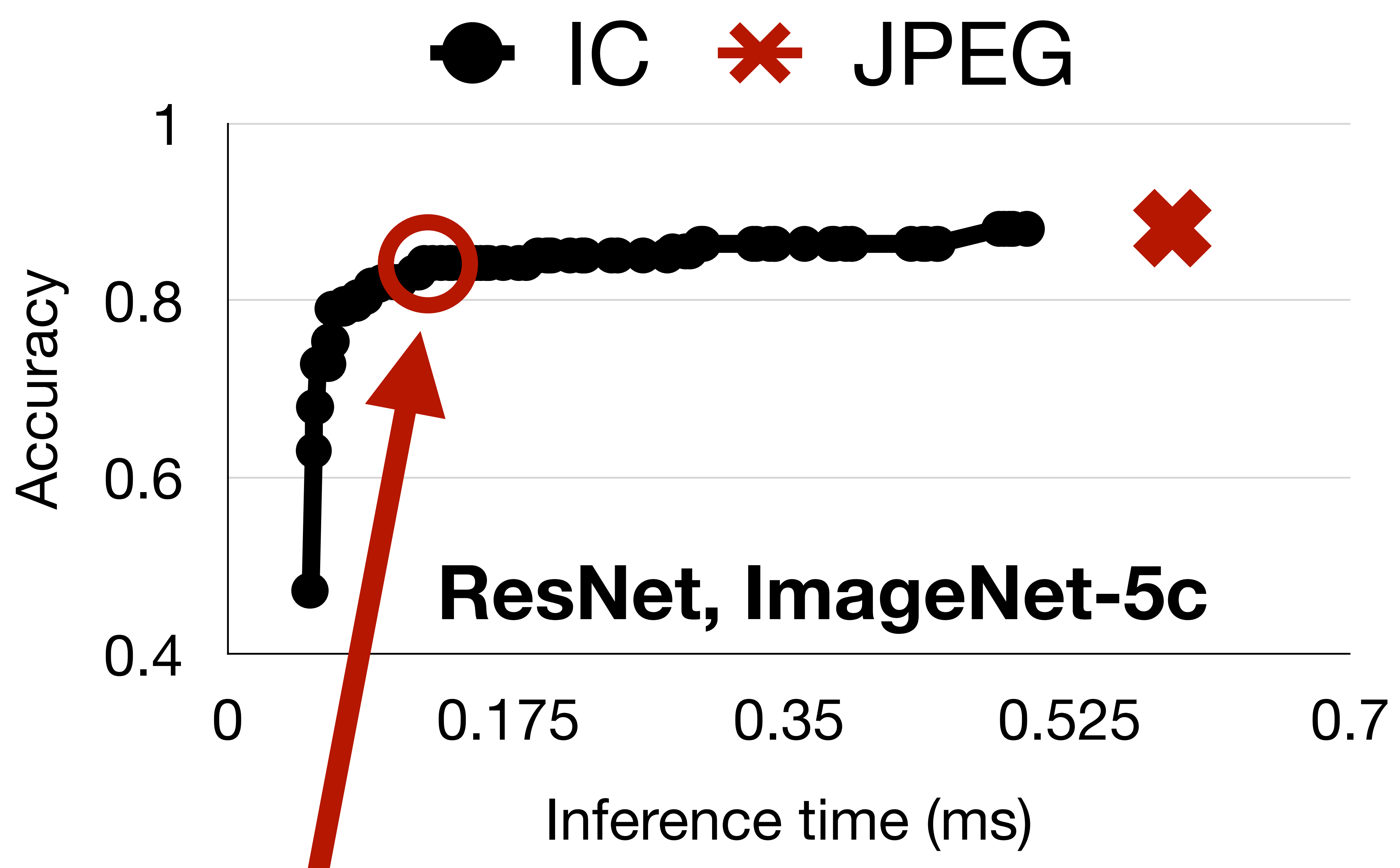


2% loss vs.
2x gain

DASlab
@ Harvard SEAS

IC brings benefits on diverse datasets

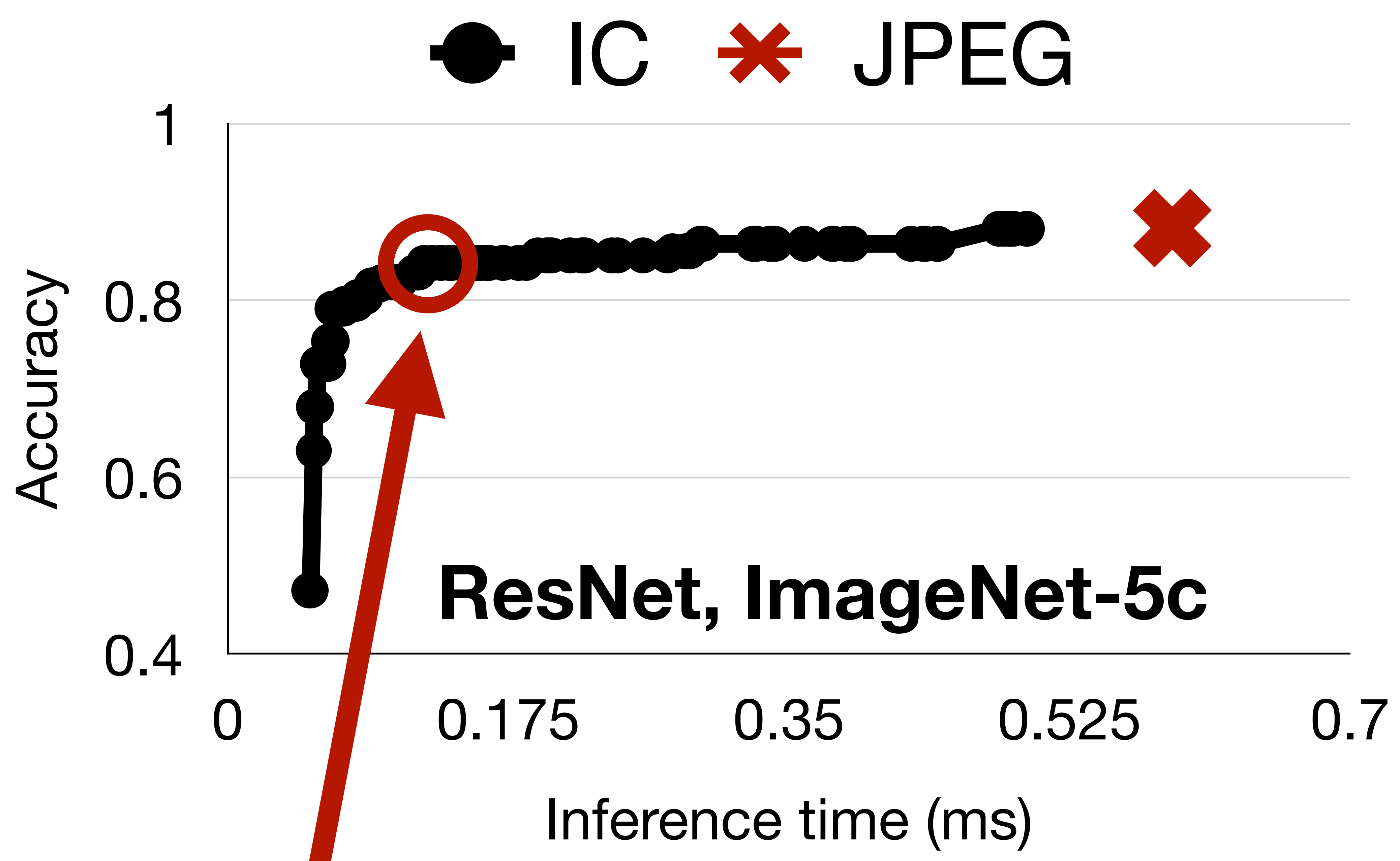
ResNet50
A100
PyTorch v1



4% loss vs.
4x gain

IC brings benefits on diverse datasets

ResNet50
A100
PyTorch v1

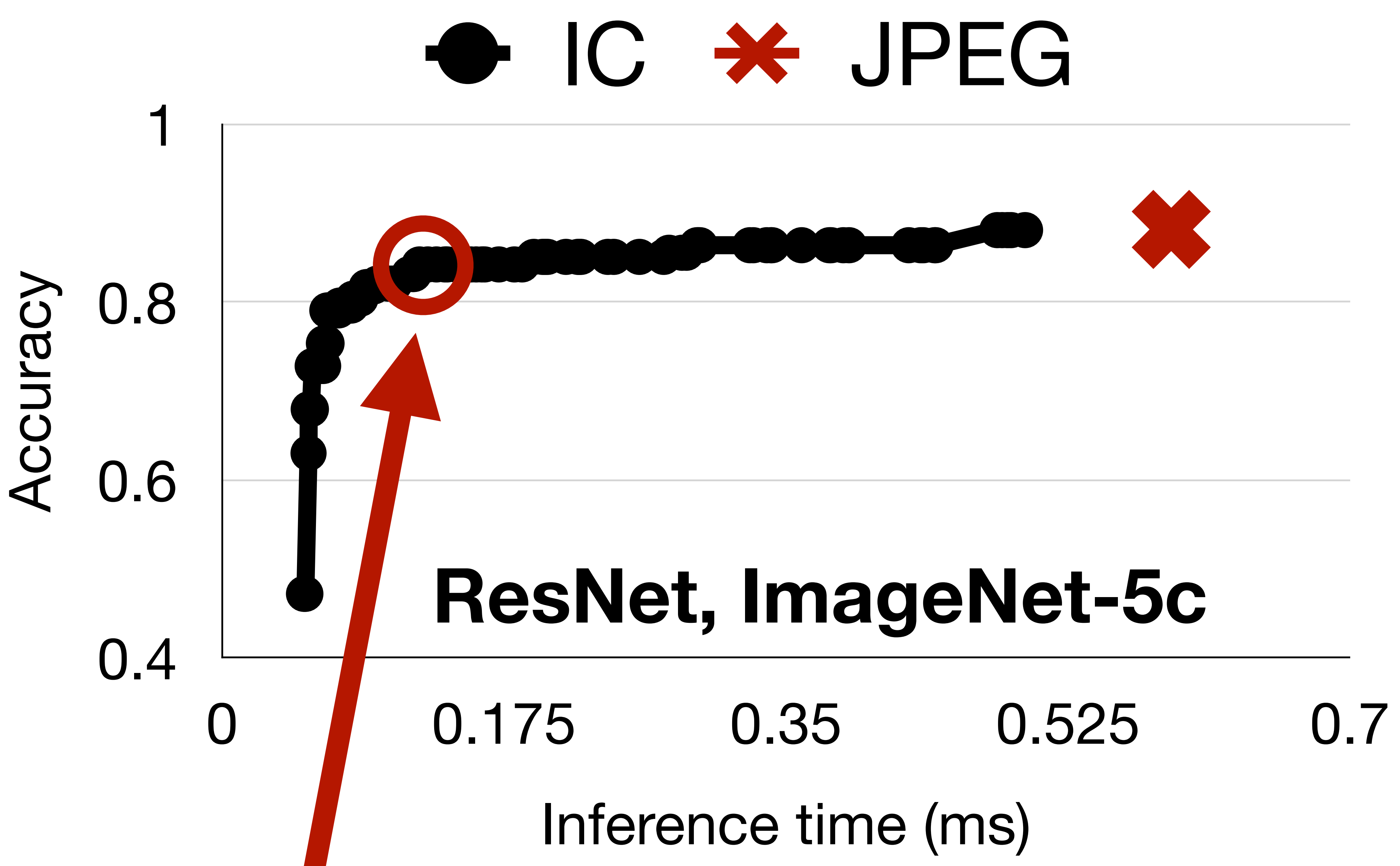


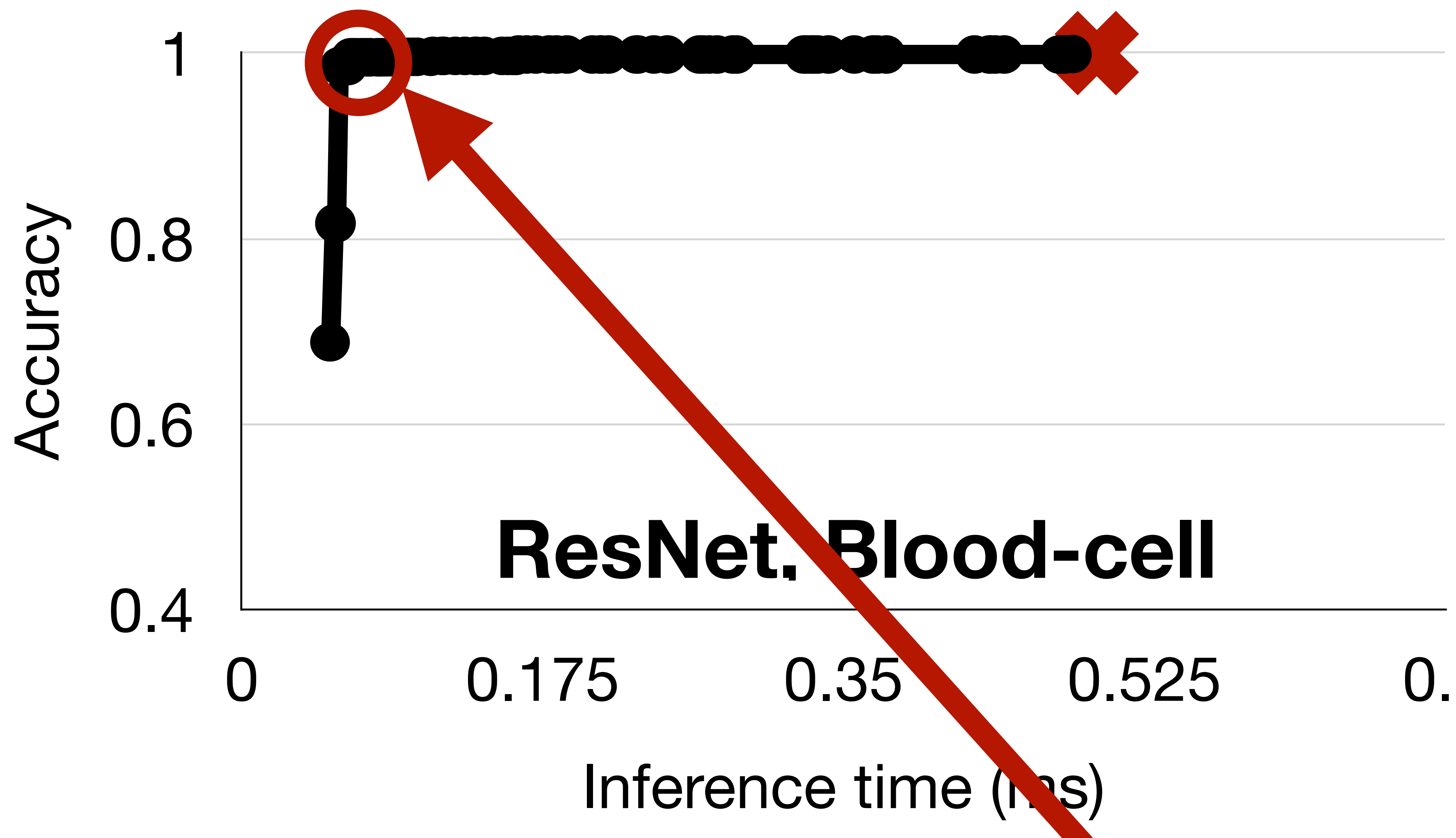
4% loss vs.
4x gain

DASlab @ Harvard SEAS

IC brings benefits on diverse datasets

ResNet50
A100
PyTorch v1



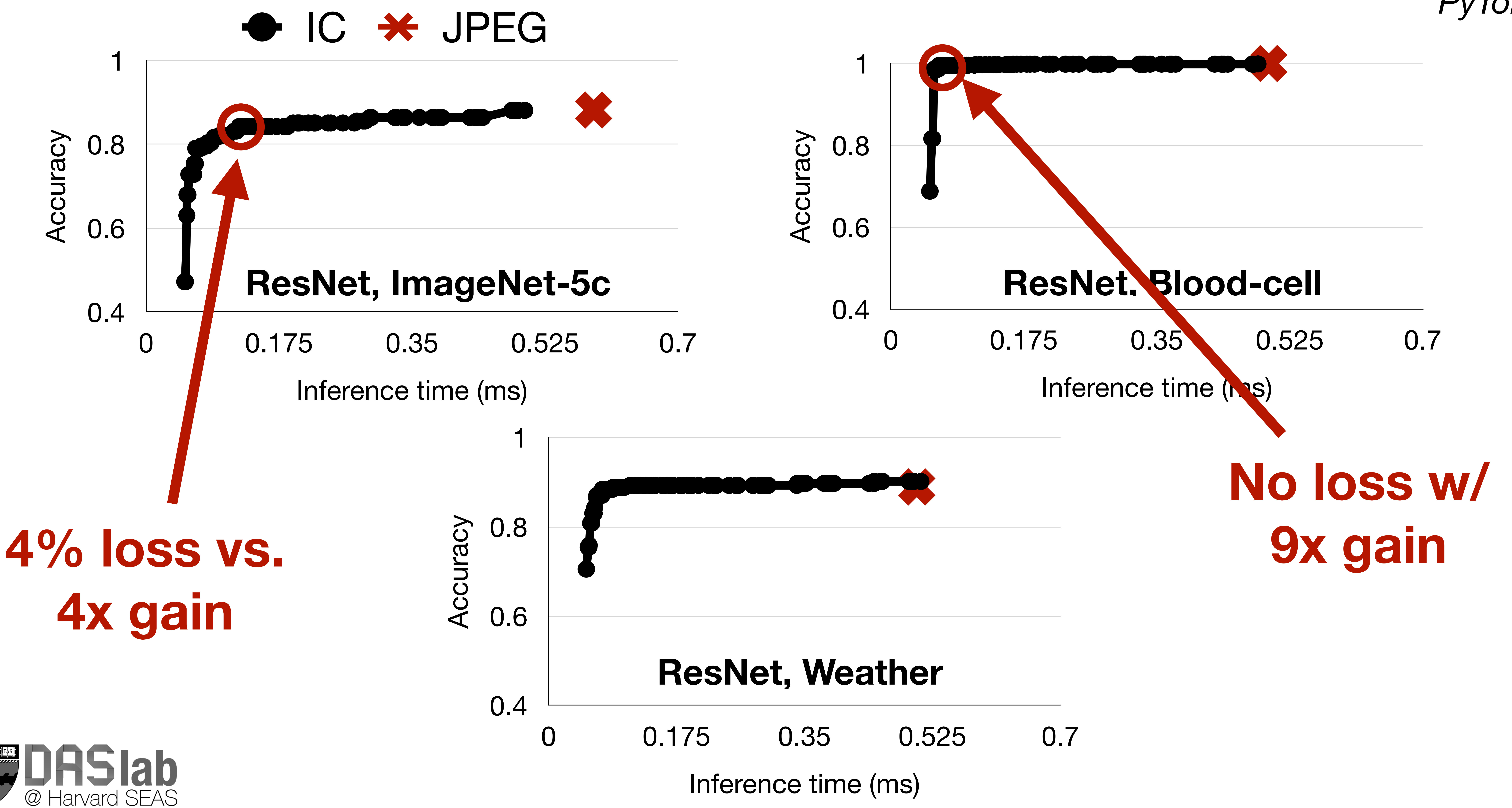


4% loss vs. 4x gain

No loss w/ 9x gain

IC brings benefits on diverse datasets

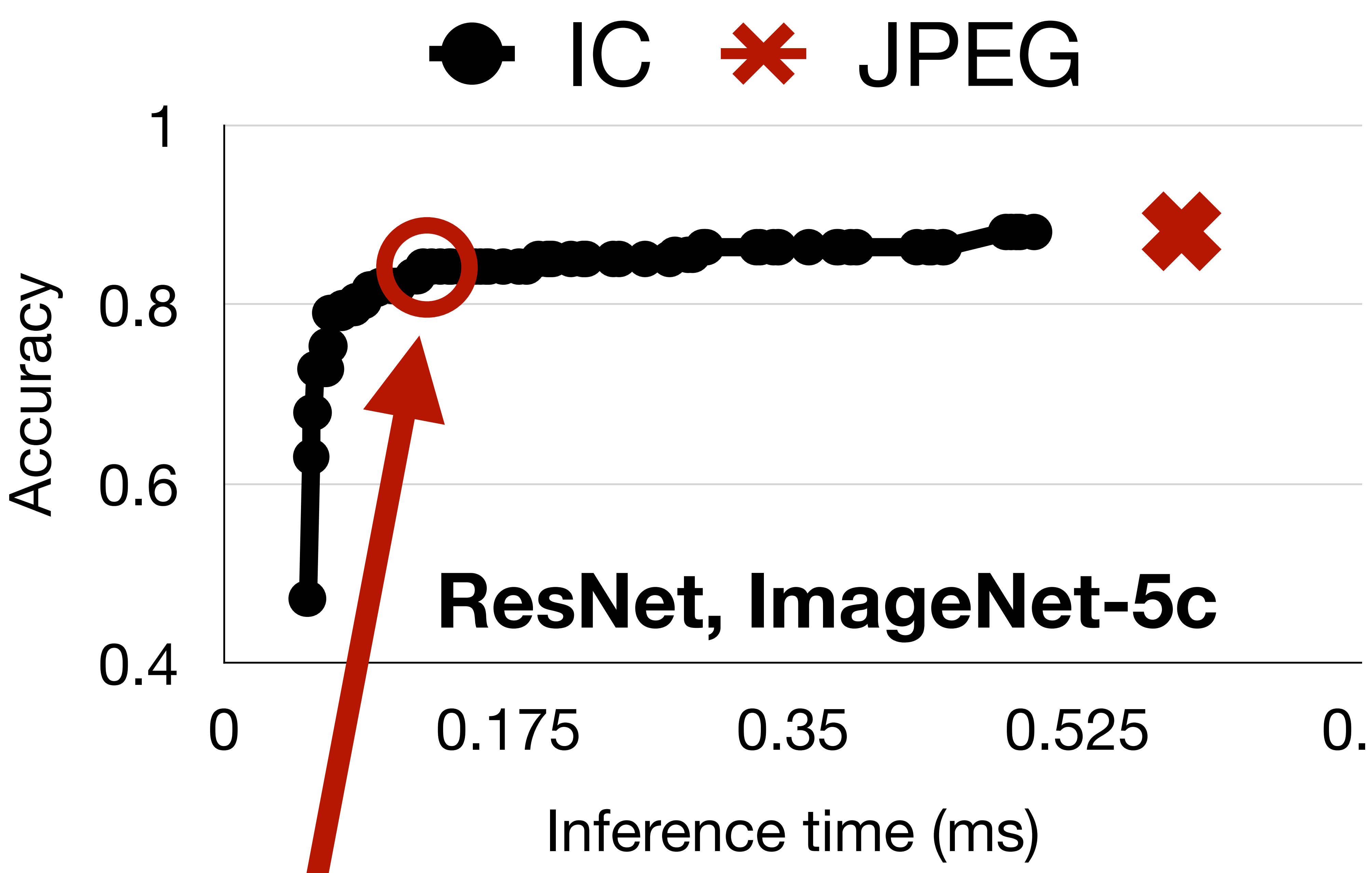
ResNet50
A100
PyTorch v1

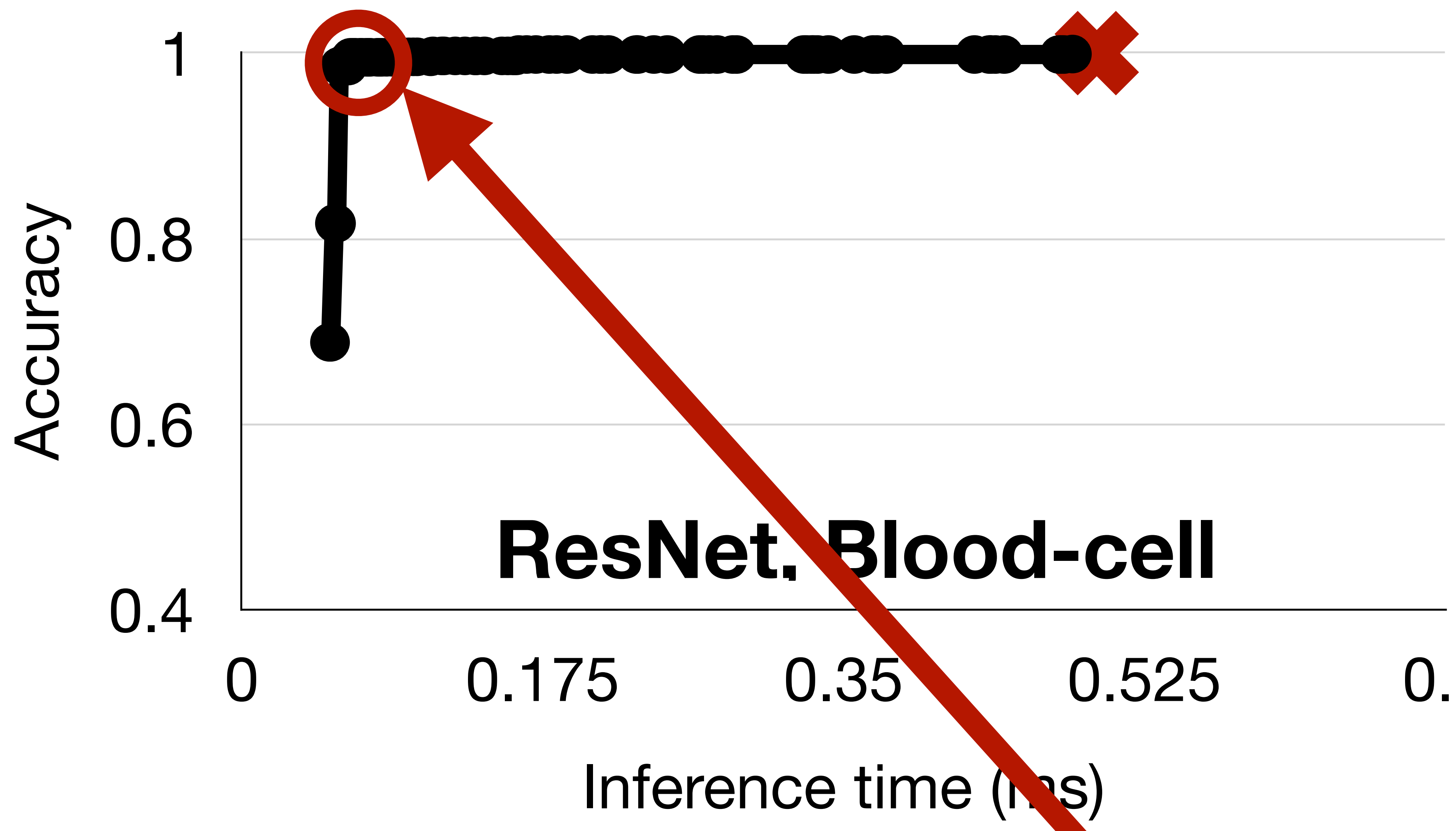


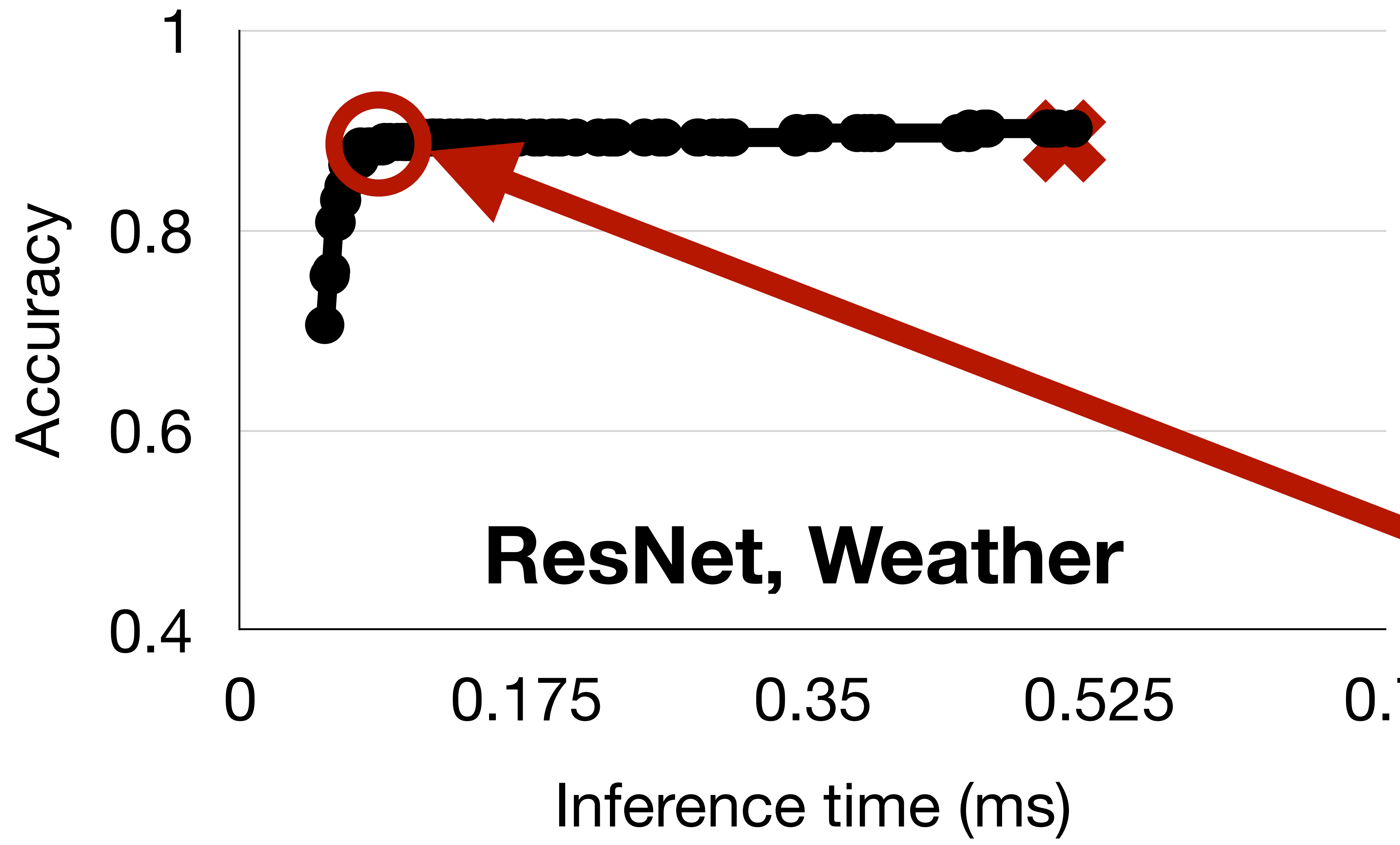
DASlab @ Harvard SEAS

IC brings benefits on diverse datasets

ResNet50
A100
PyTorch v1







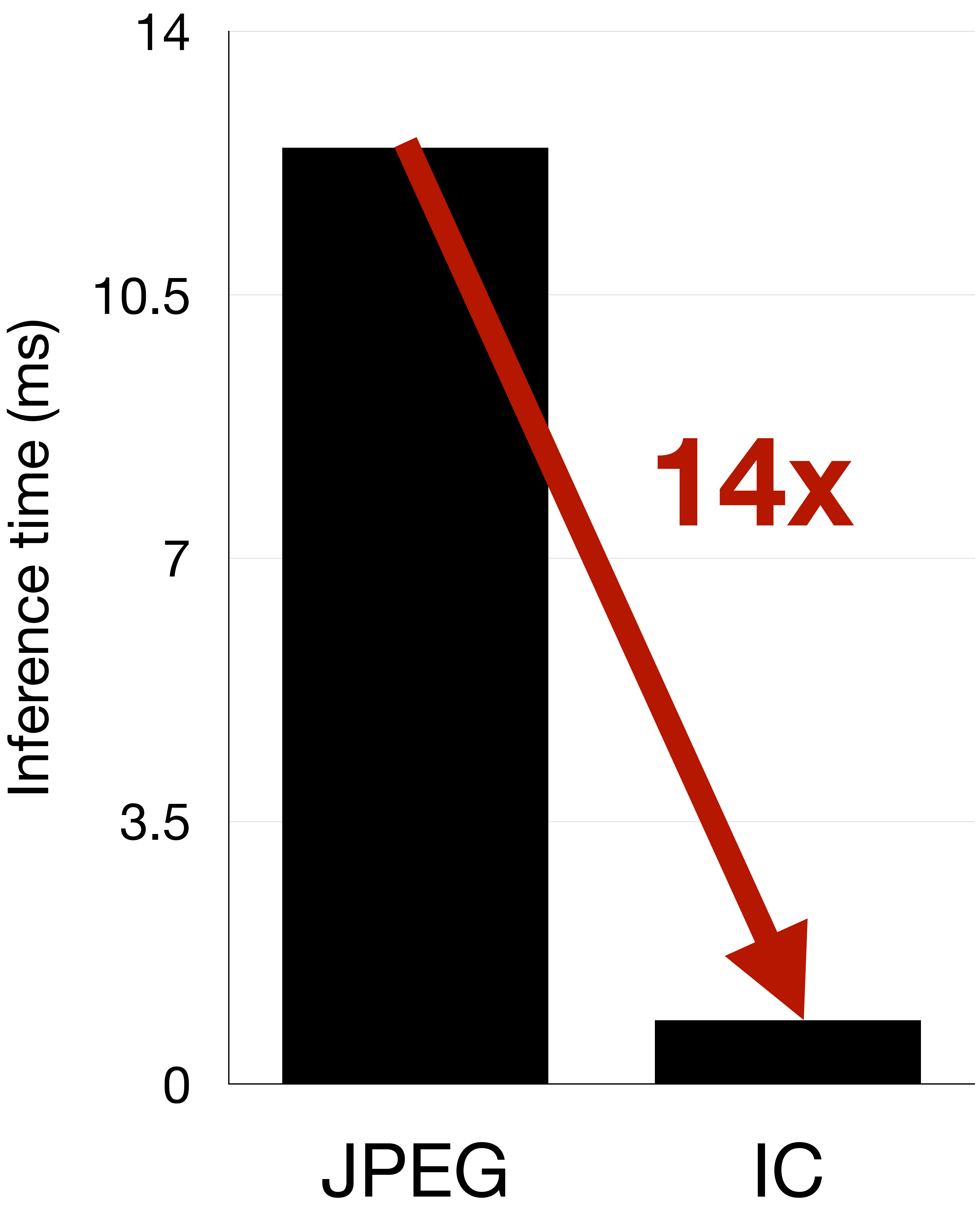
4% loss vs.
4x gain

No loss w/
9x gain

No loss w/
7x gain

IC improves on cheap CPUs

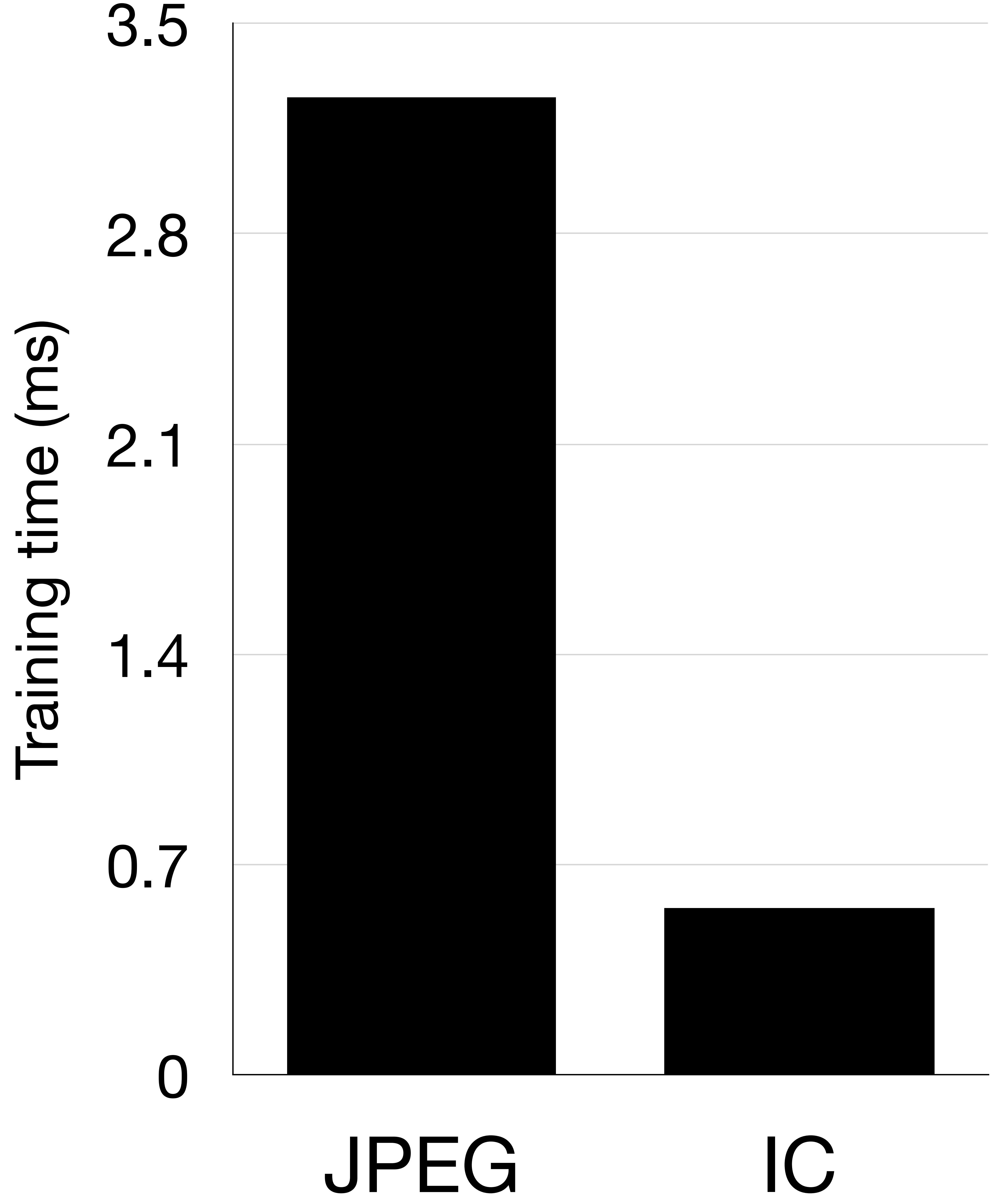
Blood-cell
MobileNetV3
Intel CPU
PyTorch v1



DASlab @ Harvard SEAS

IC reduces training time

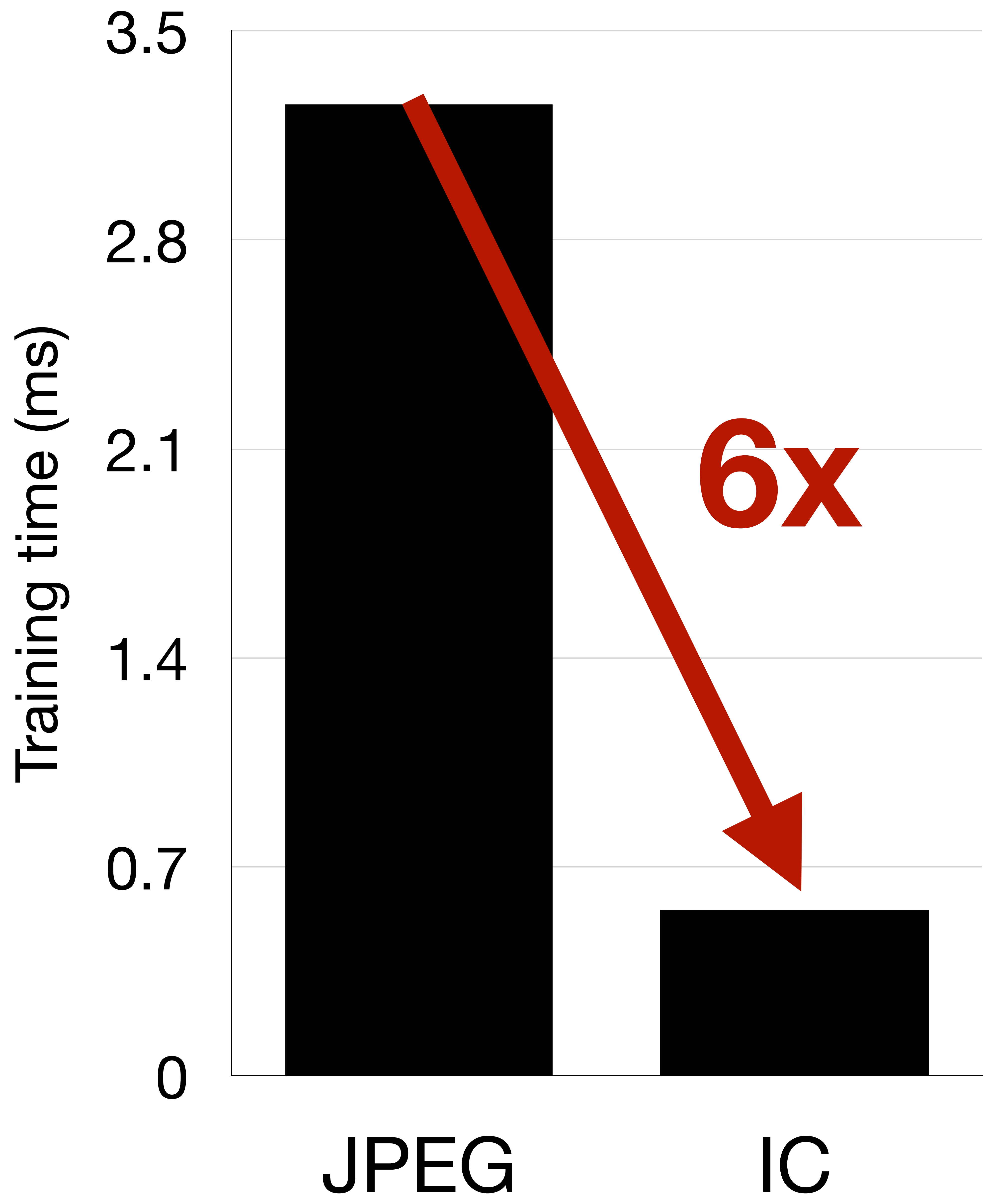
Blood-cell, ResNet50
A100, PyTorch v1



DASlab @ Harvard SEAS

IC reduces training time

Blood-cell, ResNet50
A100, PyTorch v1



DASlab @ Harvard SEAS

Generating design spaces for whole systems.
Reasoning: rules, math and ML to create entirely new designs

Generating design spaces for whole systems.
Reasoning: rules, math and ML to create entirely new designs

Primer: **The Periodic Table of Data Structures**
IEEE Data Eng. Bull. 41(3), 2018







DASlab
@ Harvard SEAS
daslab.seas.harvard.edu



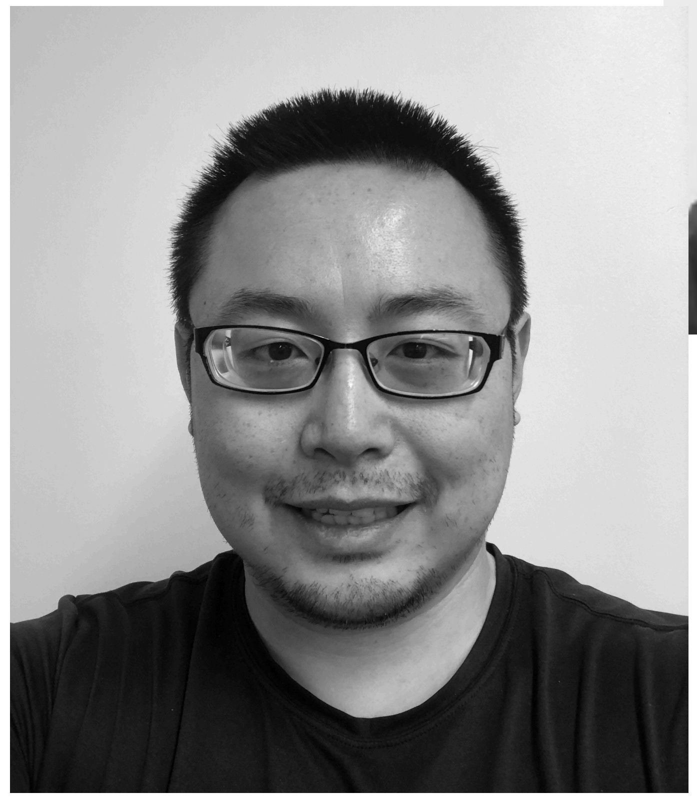



















DASlab
@ Harvard SEAS
daslab.seas.harvard.edu

THANKS!