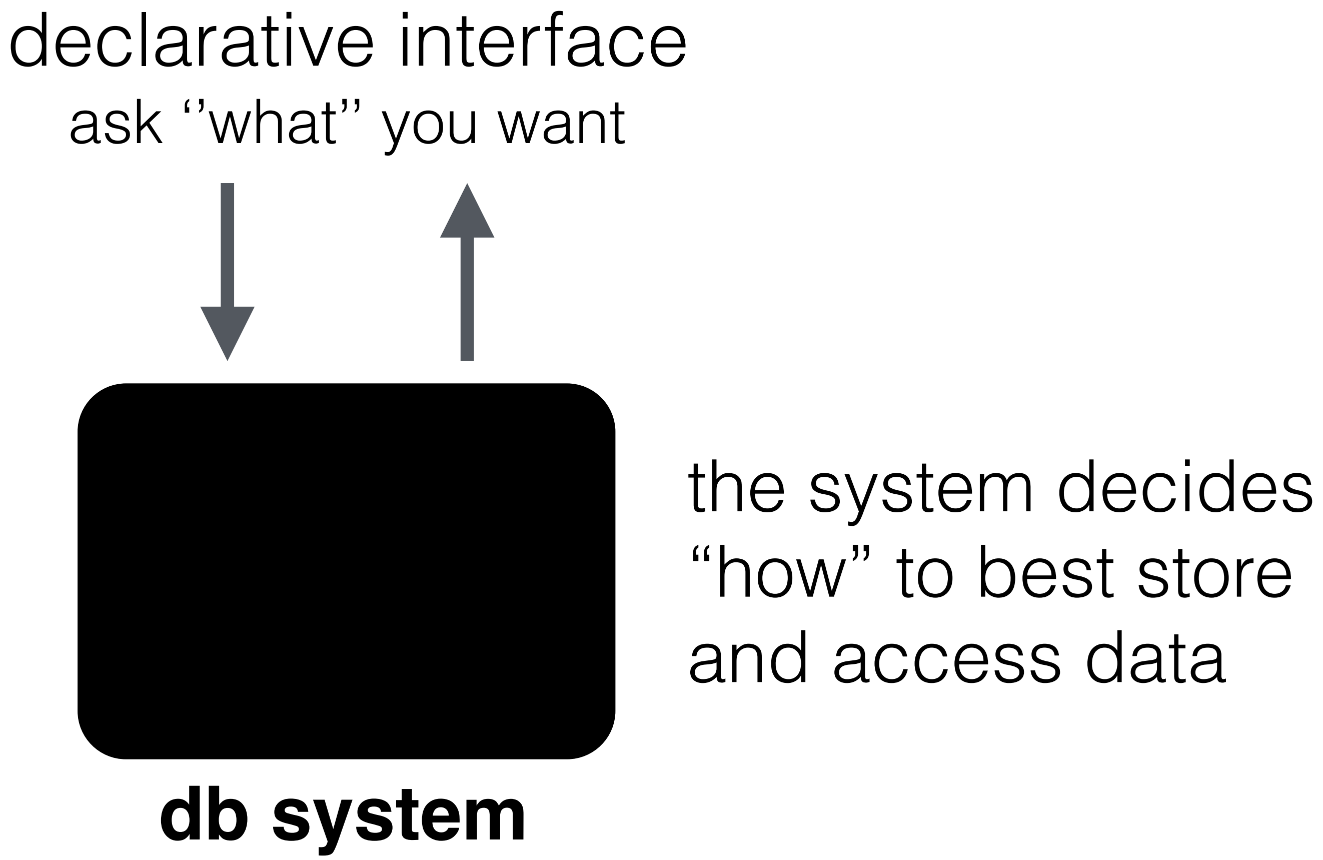
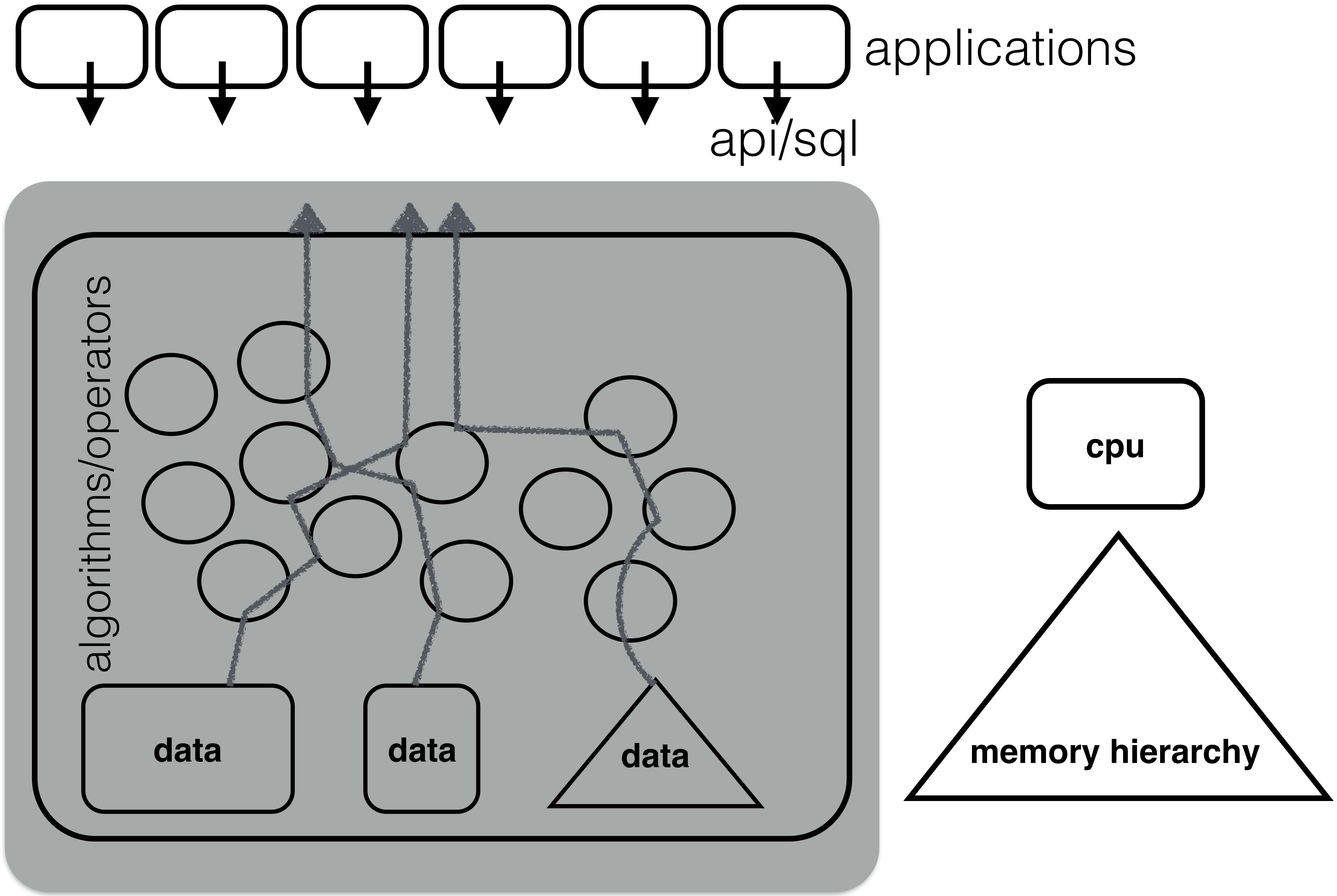
Design Tradeoffs of Data Access Methods

Manos Athanassoulis and Stratos Idreos

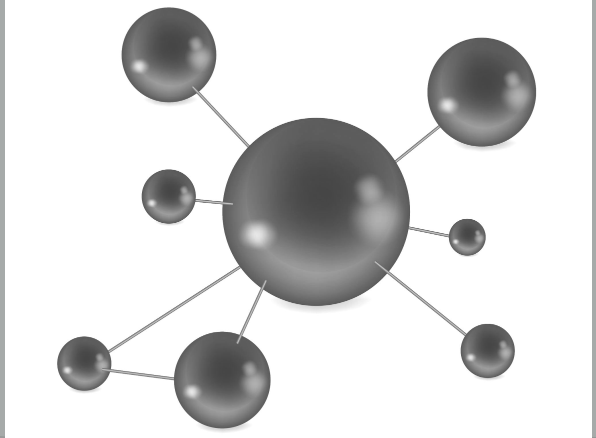

declarative interface
ask ‘’what’’ you want
the system decides “how” to best store and access data
db system



data system kernel:
a collection of access methods

an access method is a way to store and access data

layout

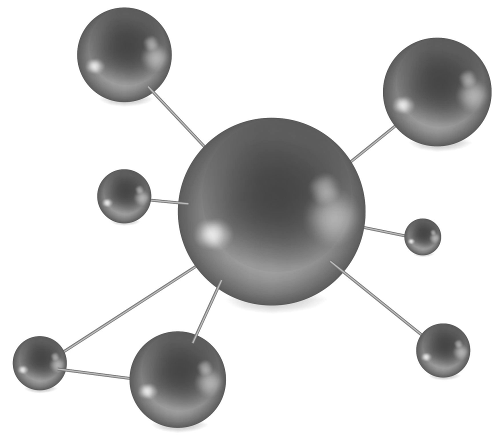


structure

navigation

an access method is a way to store and access data

layout e.g., array

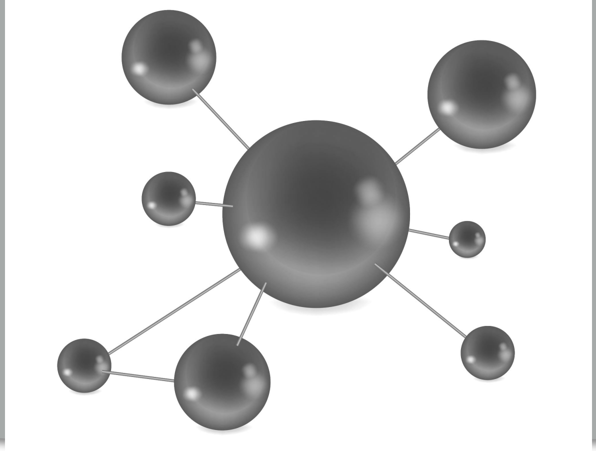


structure unordered

navigation scan

an access method is a way to store and access data

layout	**e.g., array**	**e.g., array**



structure	**unordered**	**ordered**

navigation	**scan**	**binary search**

TREES

HASH TABLES

SLOTTED PAGES

COLUMN-GROUPS

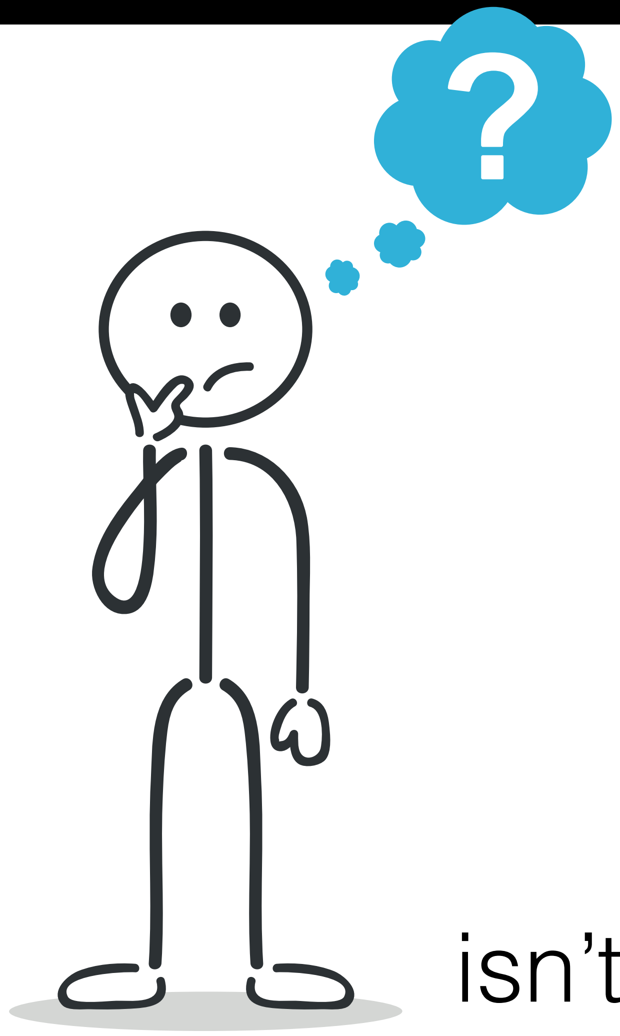
TRIES

COLUMNS

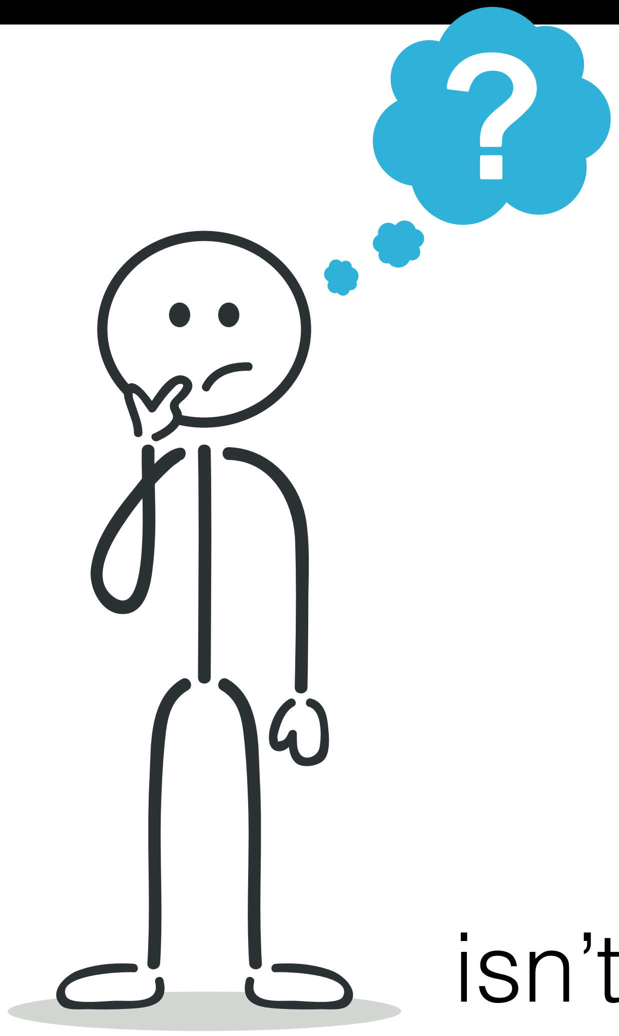
ARRAYS

LOG-STRUCTURED TREES

MULTI-DIMENTIONAL

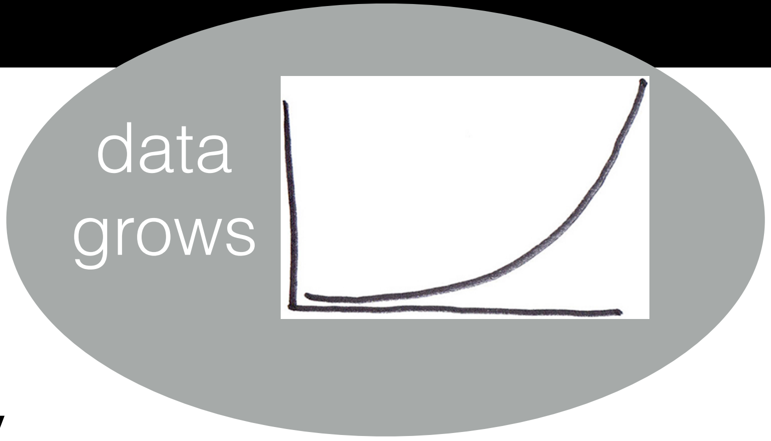


isn’t this a solved problem?



isn’t this a solved problem?

access method design is now as important as ever

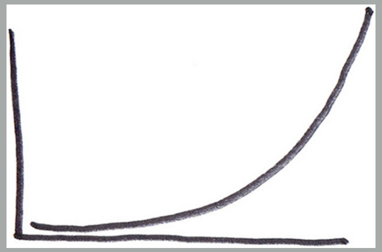


today

data systems are nearly everywhere…

continuous need for new and tailored data systems

data grows



today

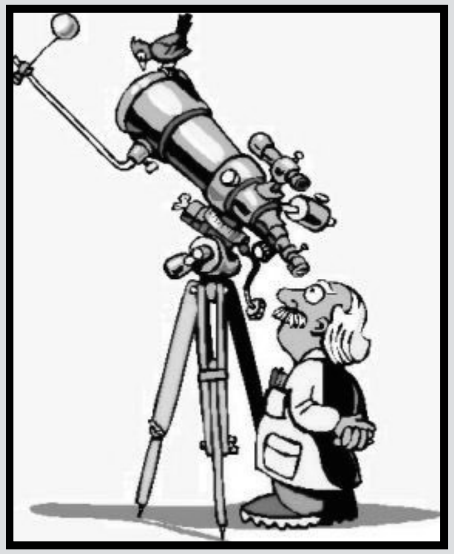
data systems are nearly everywhere…

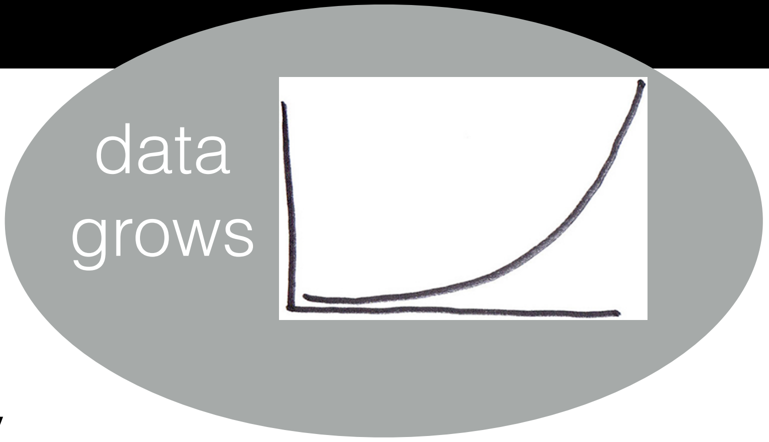
facebook

Google

continuous need for new and tailored data systems

tomorrow



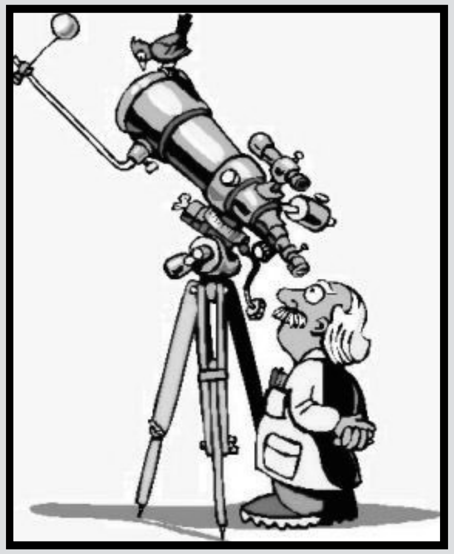


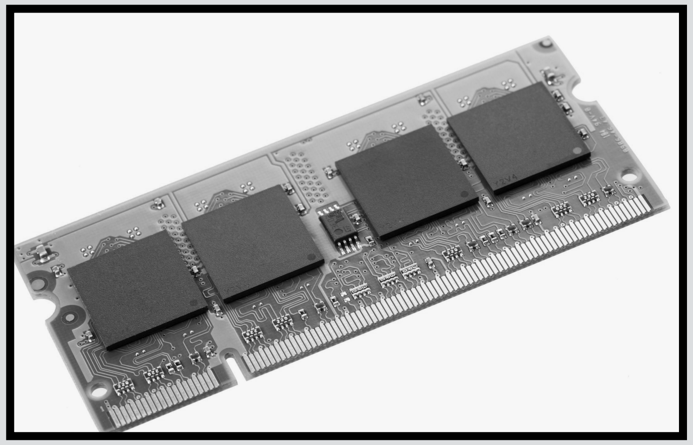
today

data systems are nearly everywhere…

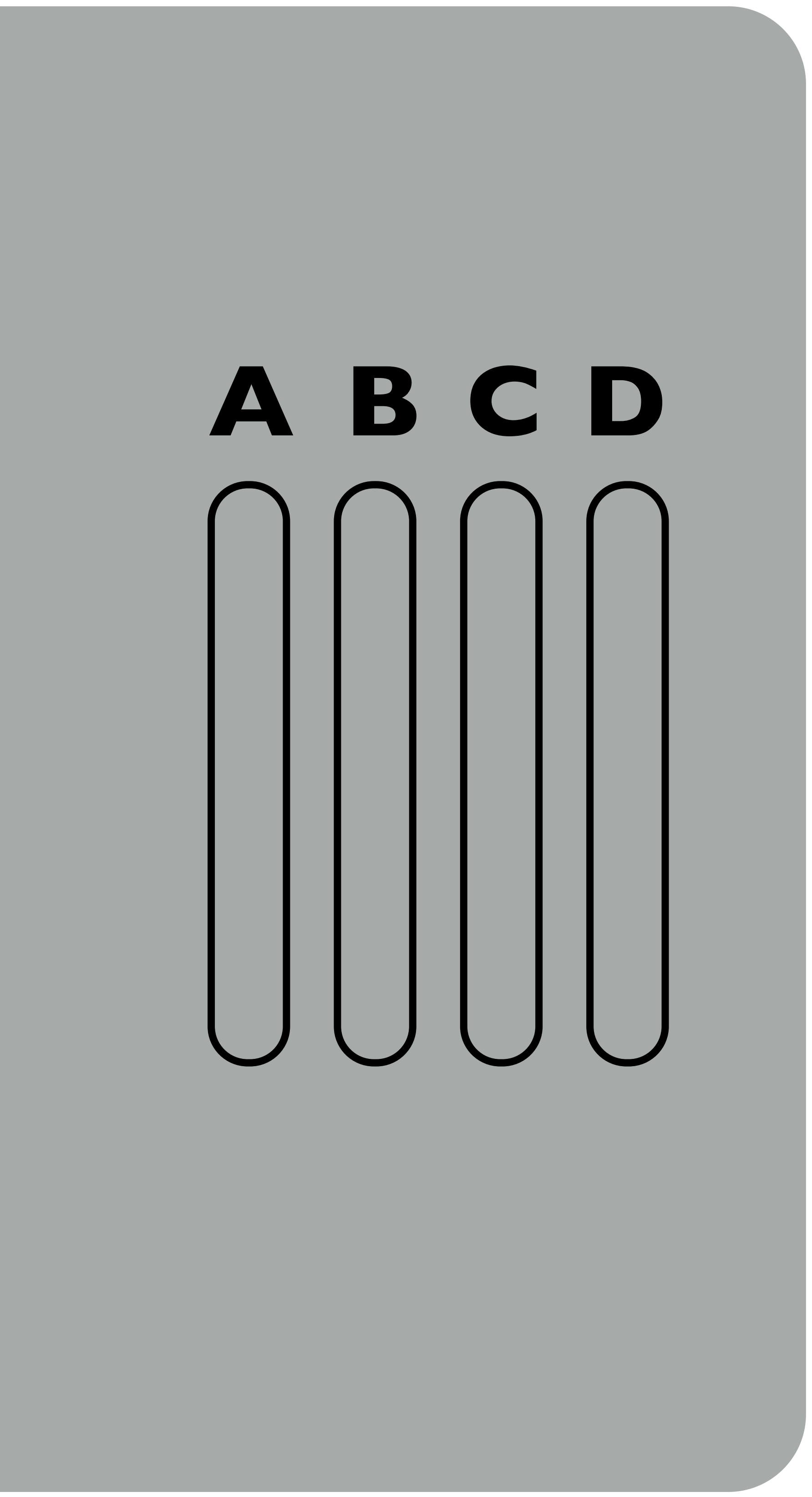
continuous need for new and tailored data systems

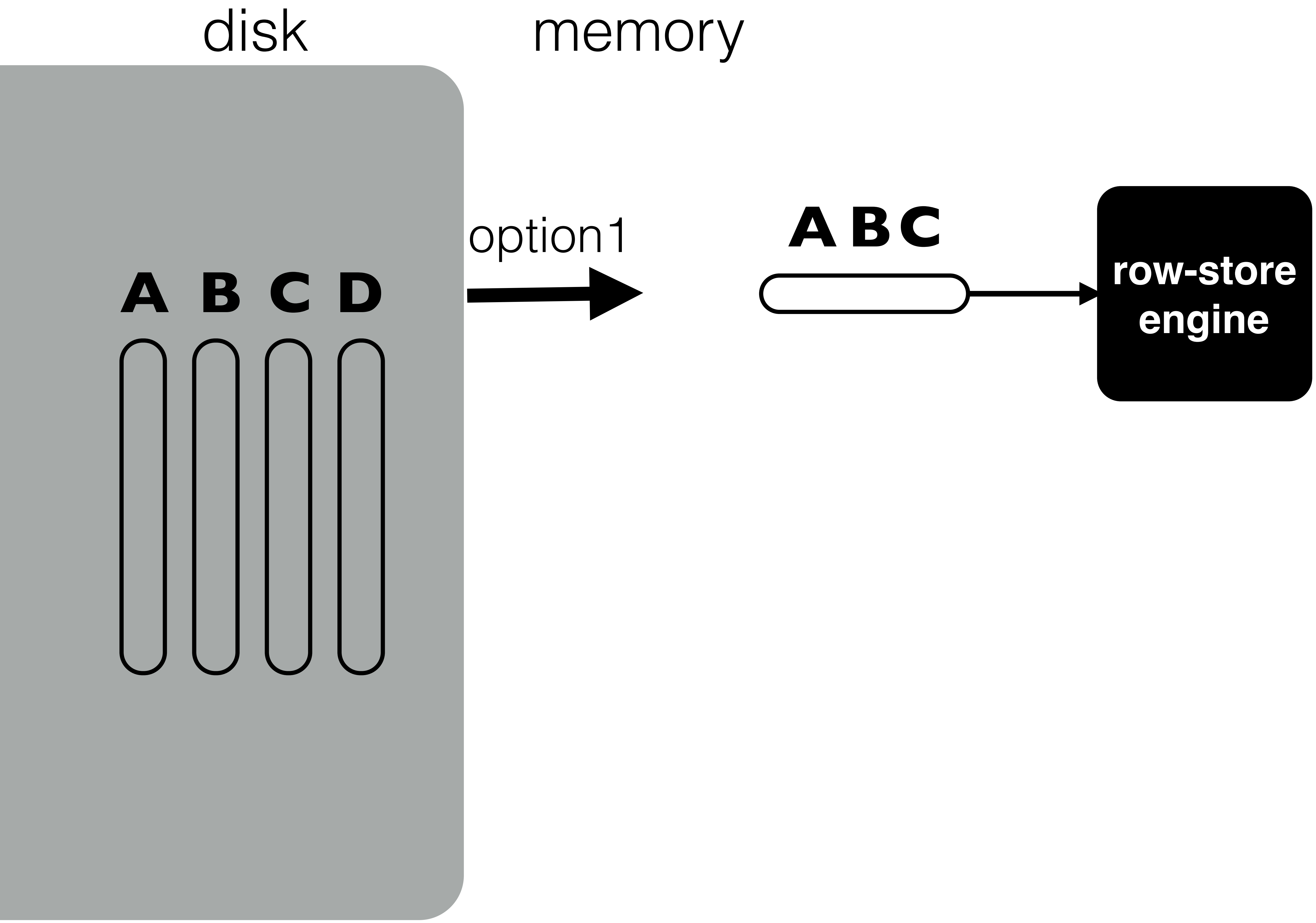
tomorrow

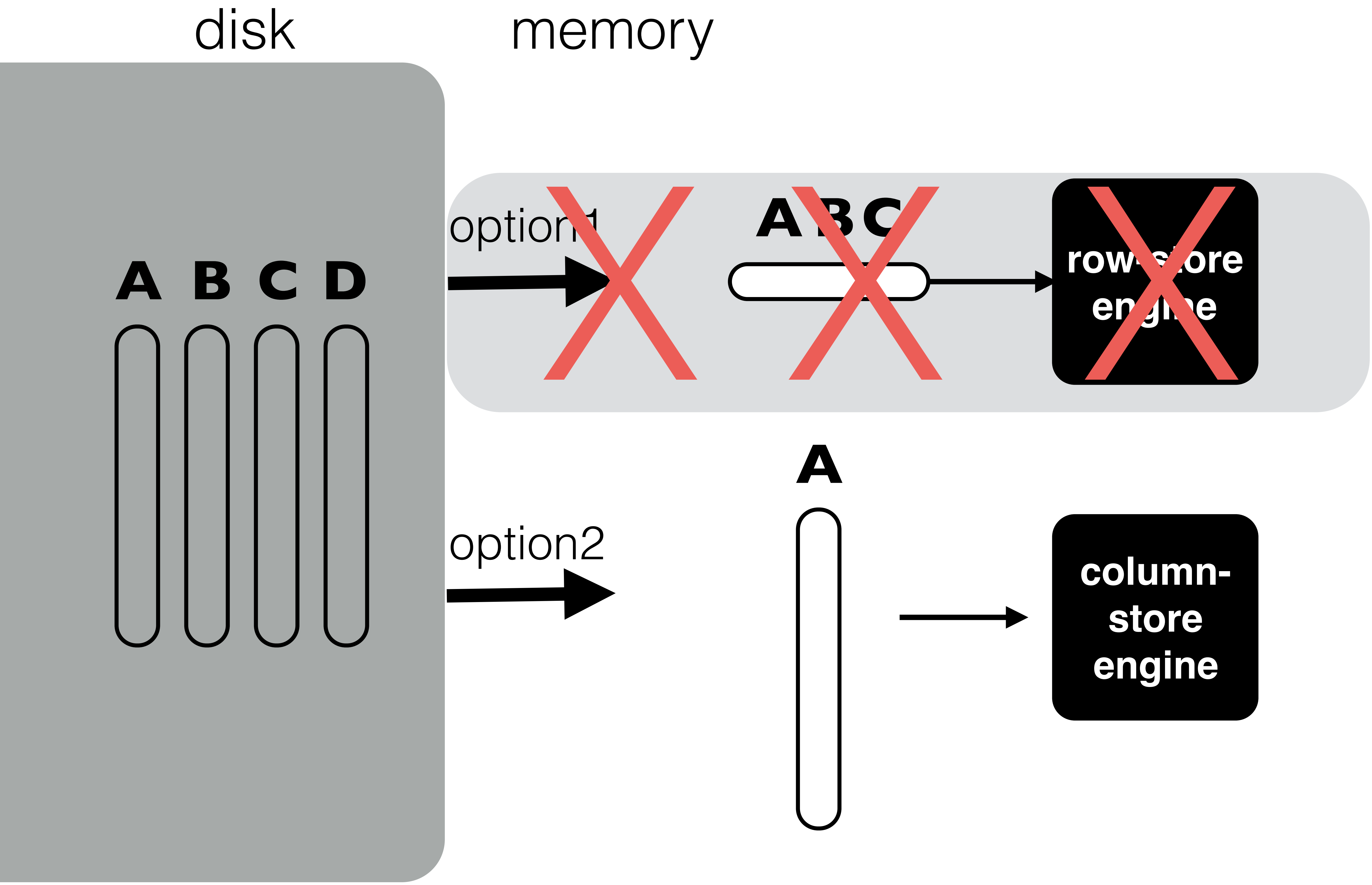


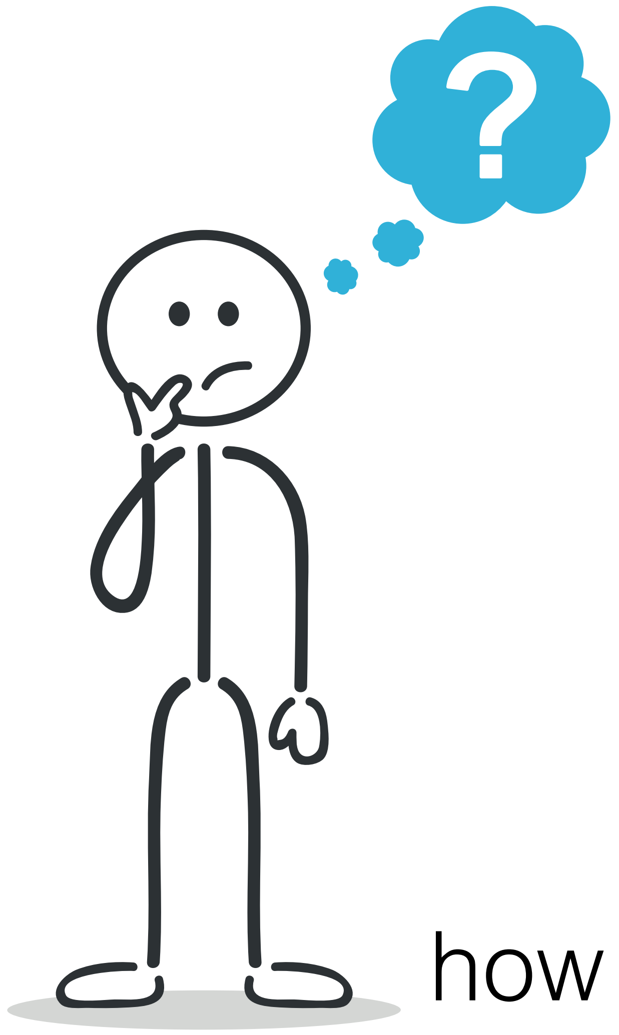


disk memory

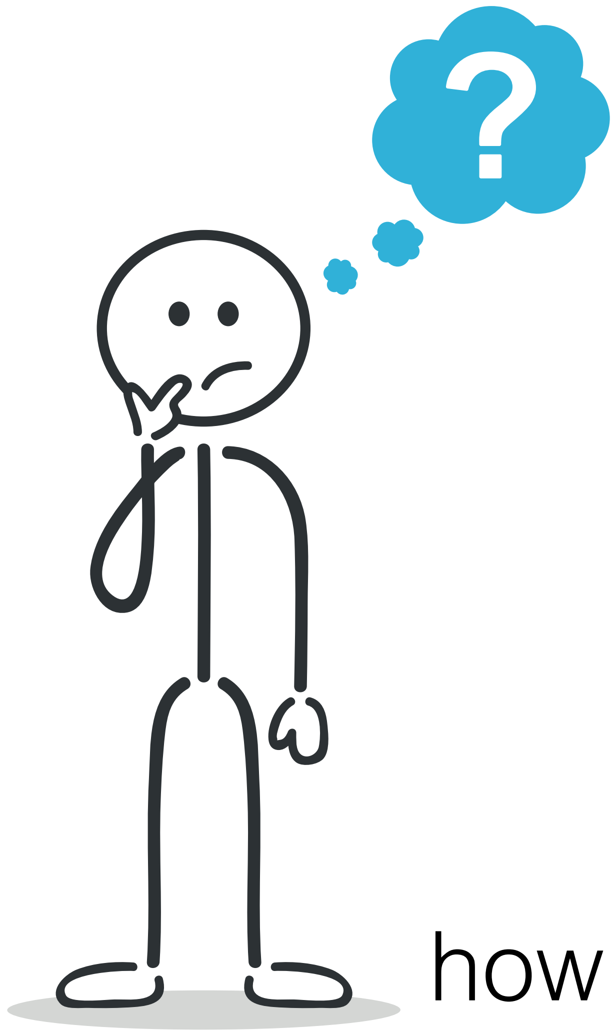



disk
memory
A B C D
option1
A B C
row-store engine


disk
memory
A B C D
option1
A B C
row-store engine
option2
A
column-store engine



how many more new access methods to design?



how many more new access methods to design?

it is not about radical new designs only!
design, tuning and variations

say the workload (read/write ratio) shifts (e.g., due to app features):
should we use a different data layout for base data - diff updates?
should we use different indexing or no indexing?

say the workload (read/write ratio) shifts (e.g., due to app features):
should we use a different data layout for base data - diff updates?
should we use different indexing or no indexing?

say we buy new hardware X (flash/memory):
should we change the size of b-tree nodes?
should we change the merging strategy in our LSM-tree?

say the workload (read/write ratio) shifts (e.g., due to app features):
should we use a different data layout for base data - diff updates?
should we use different indexing or no indexing?

say we buy new hardware X (flash/memory):
should we change the size of b-tree nodes?
should we change the merging strategy in our LSM-tree?

say we want to improve response time:
would it be beneficial if we would buy faster flash disks?
would it be beneficial if we buy more memory?

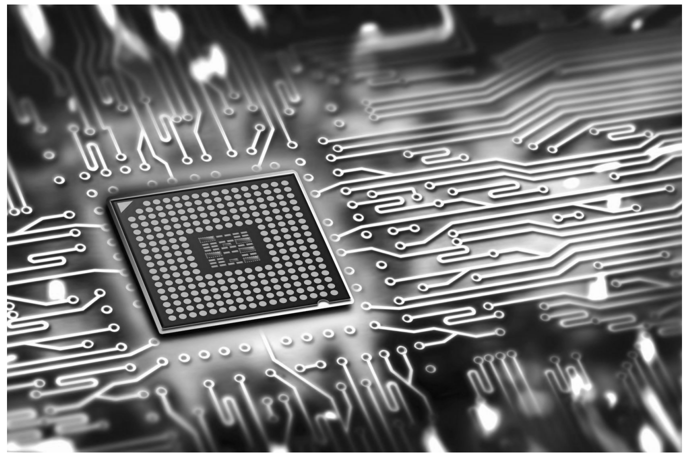
conflicting goals moving target

(hardware and requirements change continuously and rapidly)

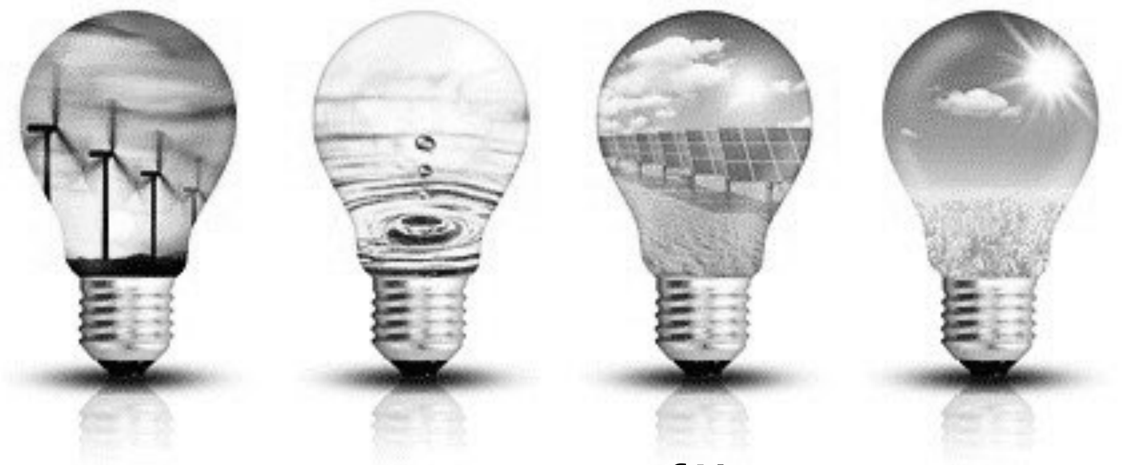
application requirements

performance

budget

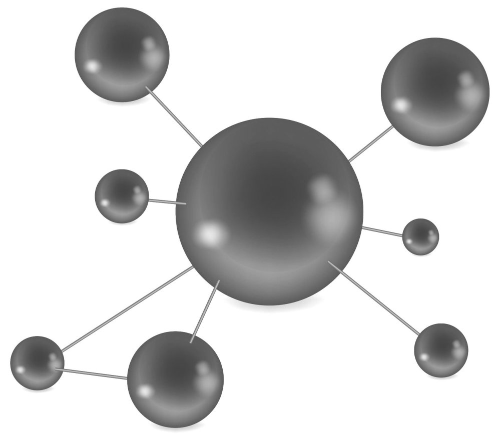


hardware



energy profile

move from design based on intuition & experience only
to a more formal and systematic way to design systems



goals and structure of the tutorial

structure design space & tradeoffs

highlight open problems towards easy to design methods

goals and structure of the tutorial

structure design space & tradeoffs

highlight open problems towards easy to design methods

basic tradeoffs
goals & vision

~30 min

[slides available at daslab.seas.harvard.edu]

design
space

~40 min

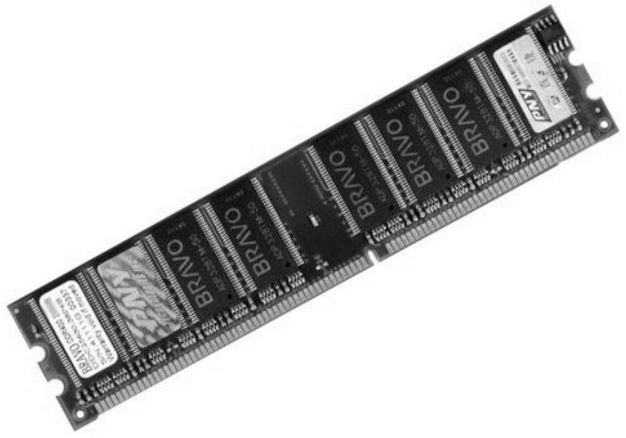
target audience = beginner to expert

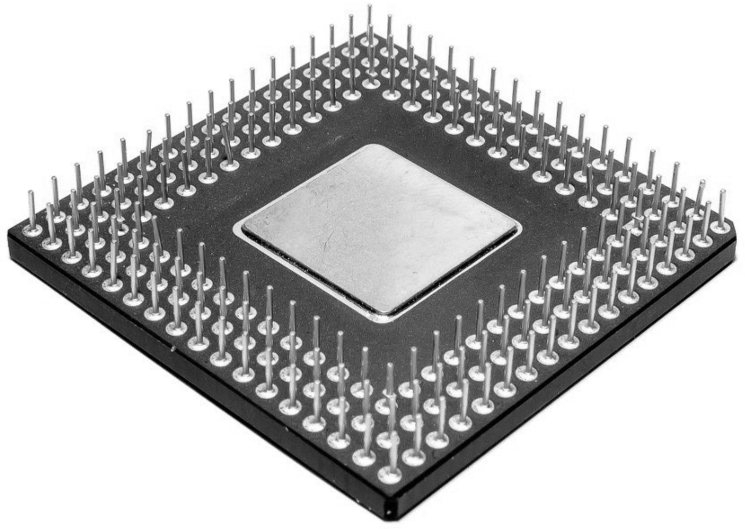
no new designs but new
connections & structure

NOT JUST SQL

+

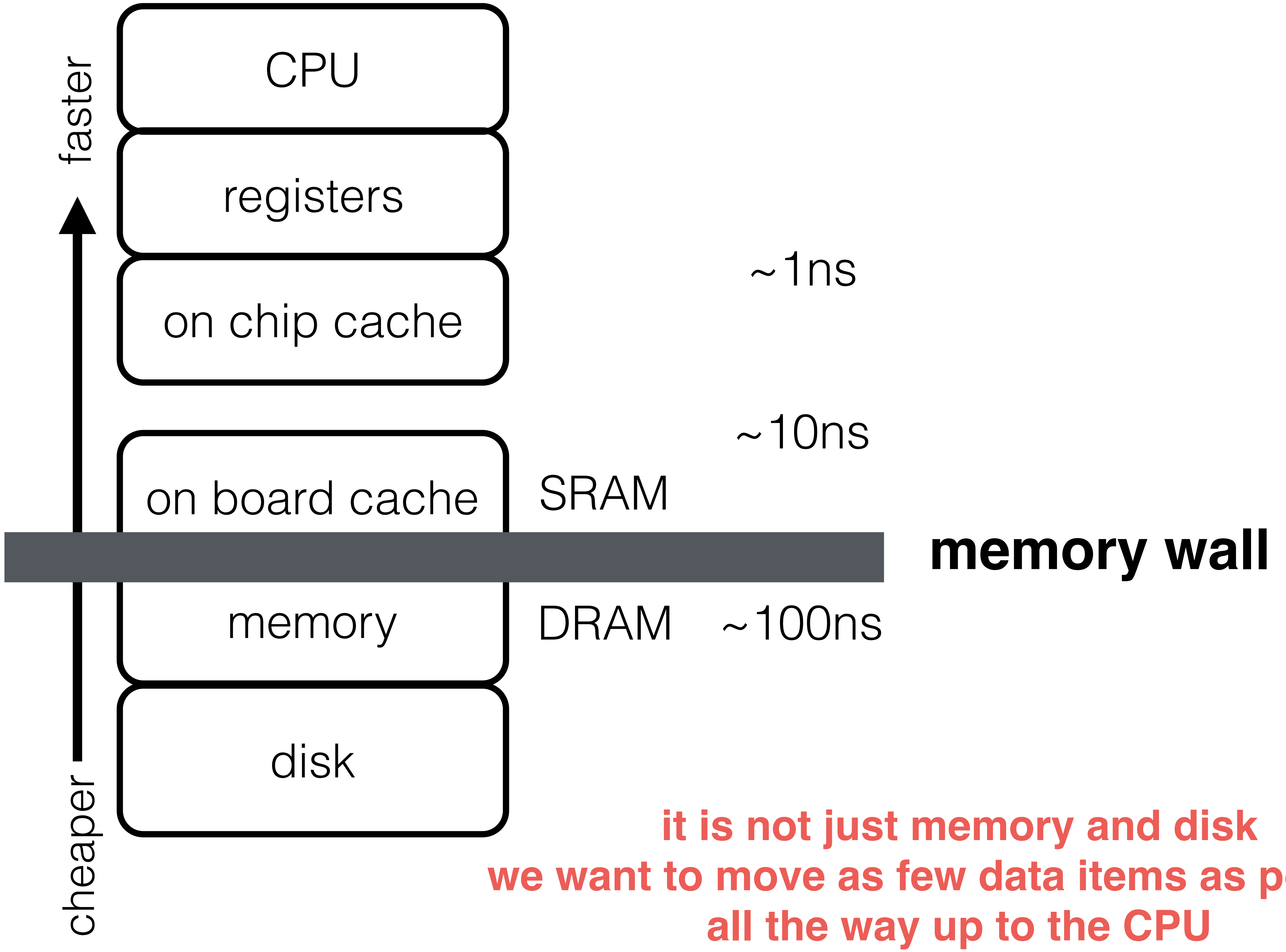
operating systems, no sql, sciences





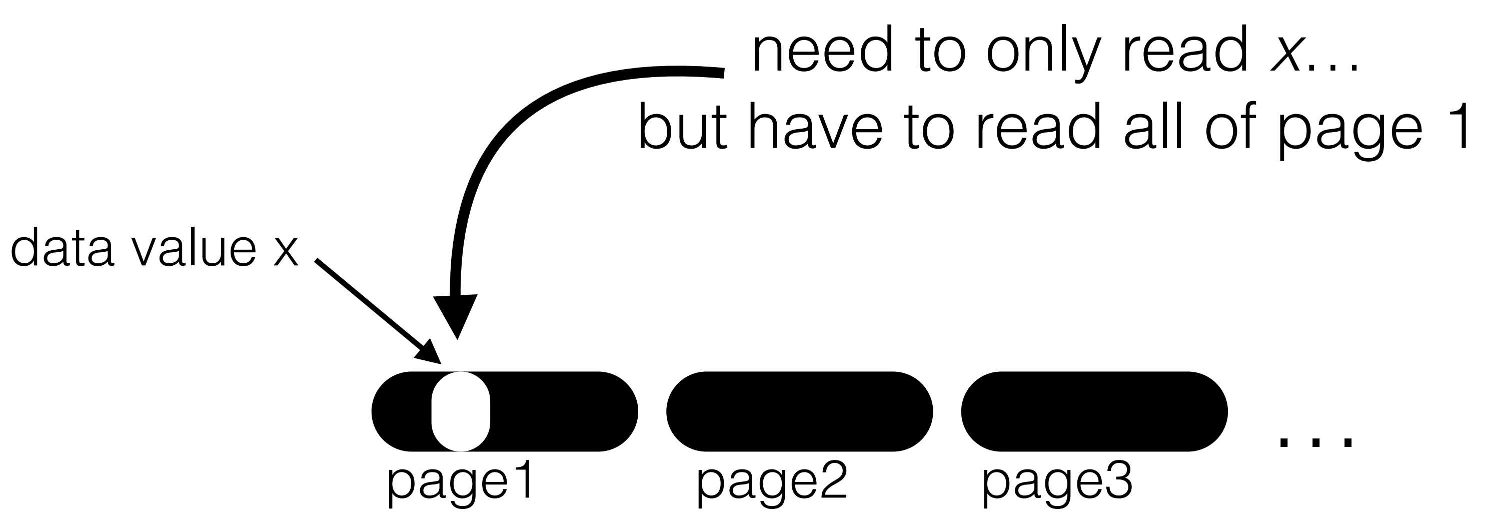
hardware is a big drive of access method (re)design

(and it continuously evolves)



it is not just memory and disk
we want to move as few data items as possible
all the way up to the CPU

random access & page-based access



what is the perfect access method?

what is the perfect access method?

$$\begin{tikzcd}
\text{what is the perfect access method?} \arrow[d, bend right] \\
\text{no single answer; it depends}
\end{tikzcd}$$

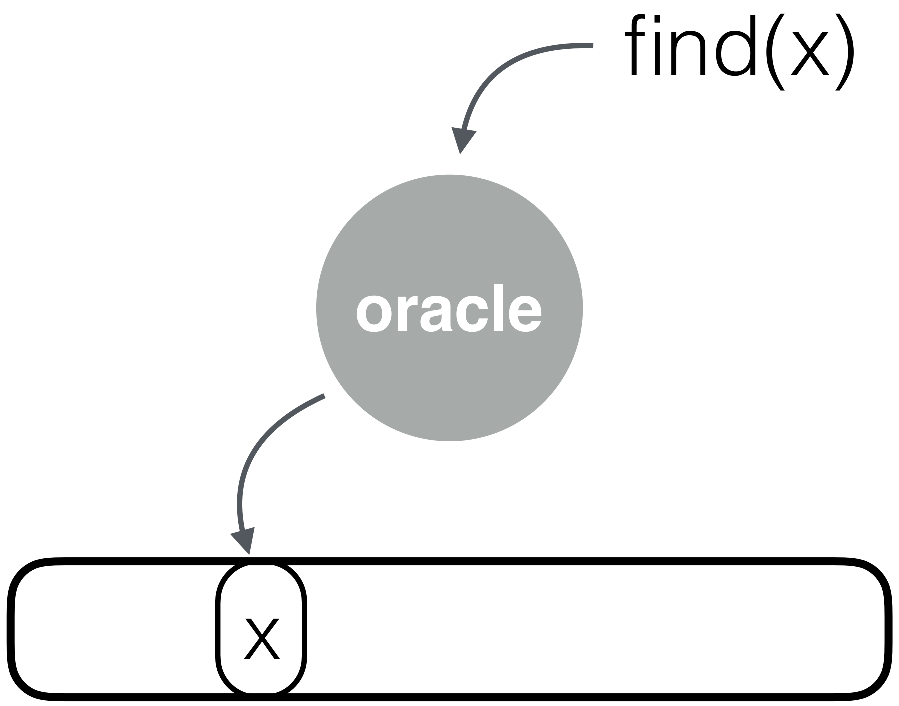
no single answer; it depends

what is the perfect access method?

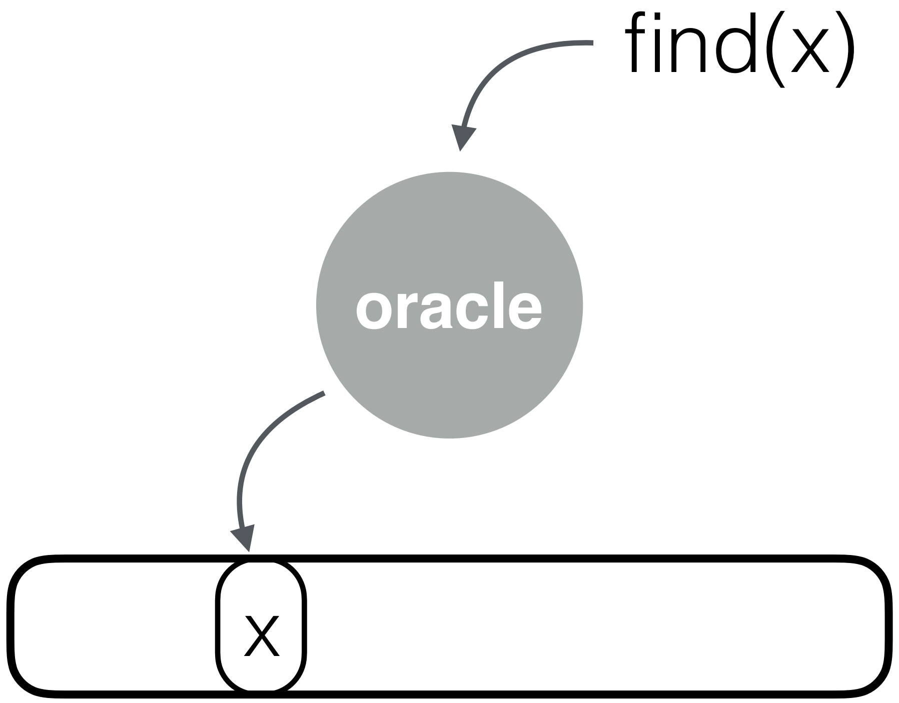
no single answer; it depends

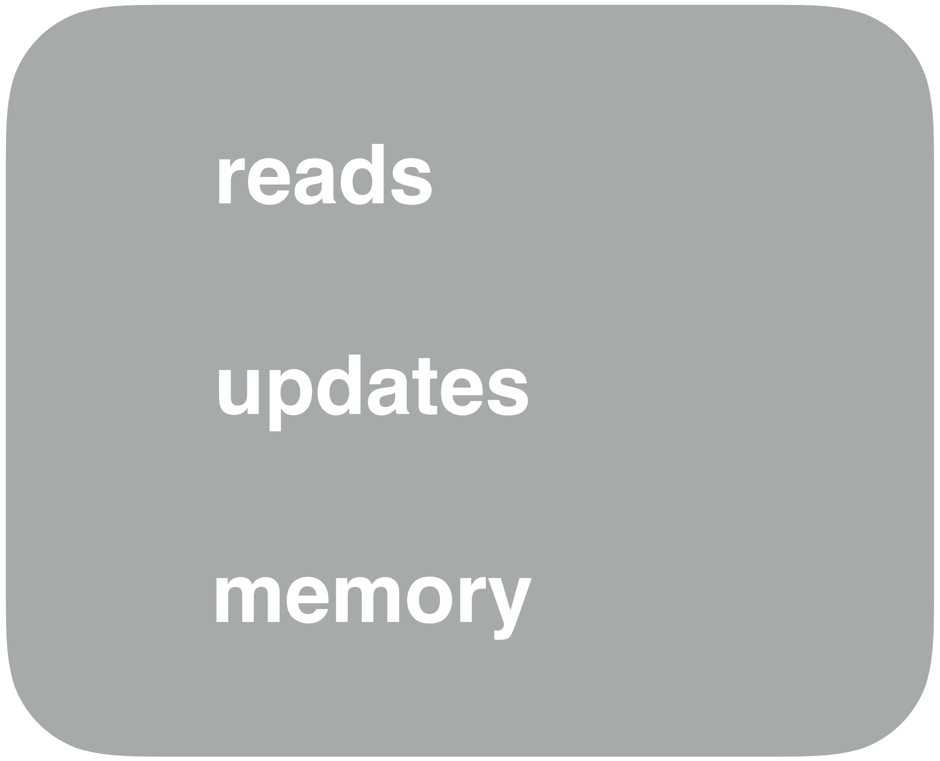
what is the application
read patterns
write patterns
reads/writes ratios
hardware (CPU, memory, etc)
SLAs

a perfect access method for reads (point queries)

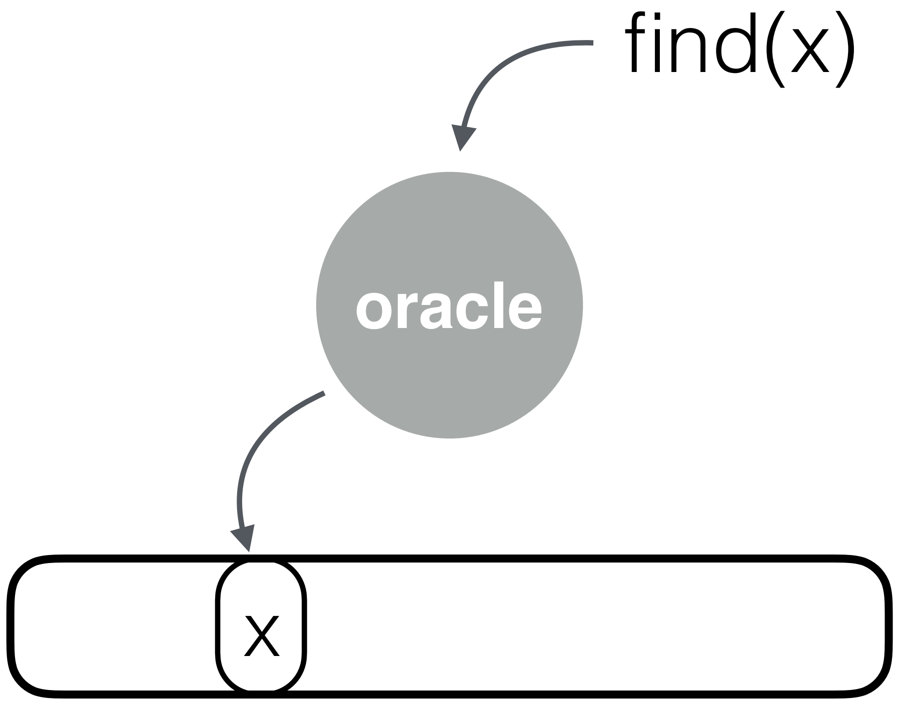


a perfect access method for reads (point queries)





a perfect access method for reads (point queries)

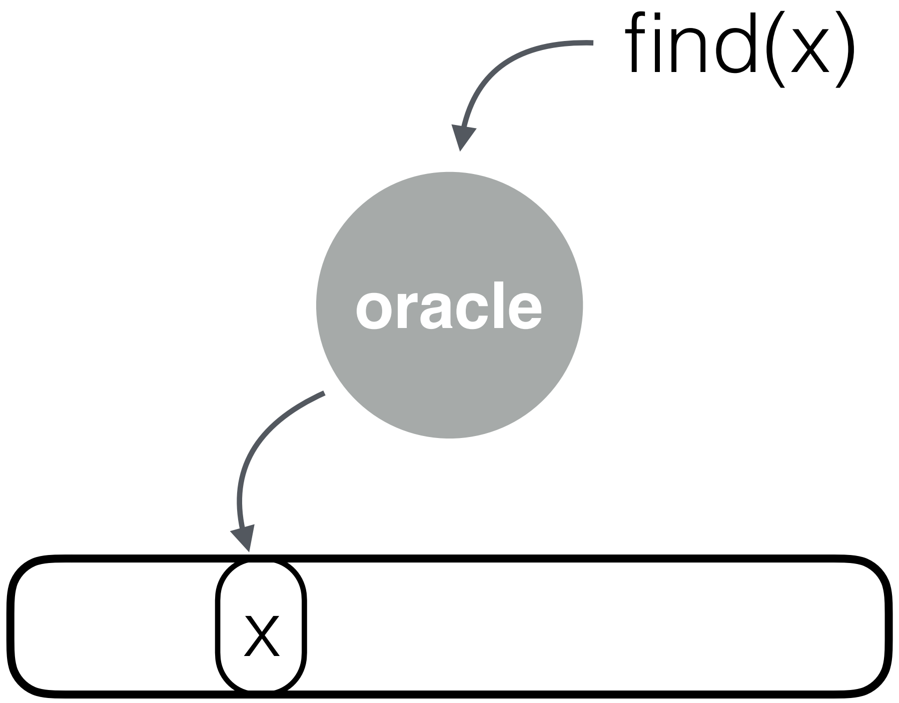


reads ✓

updates

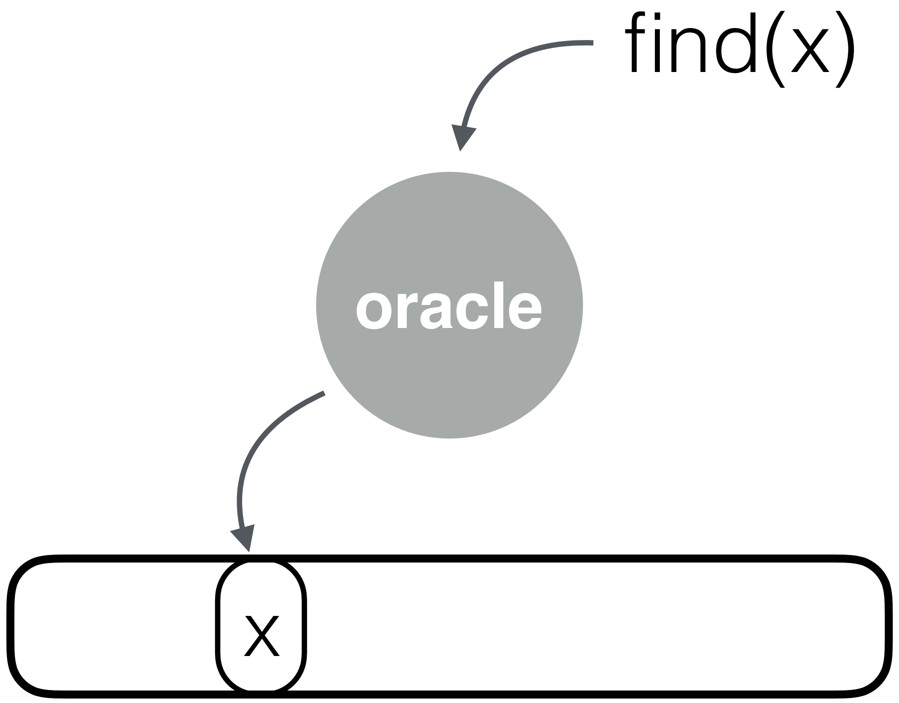
memory

a perfect access method for reads (point queries)



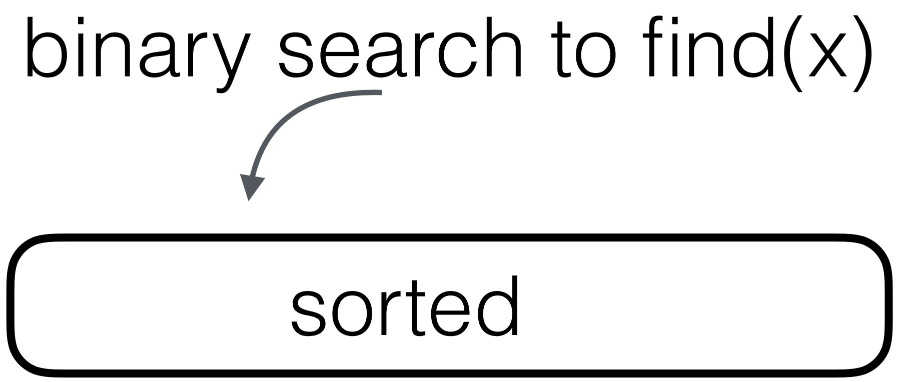
reads	✓
updates	✗
memory	

a perfect access method for reads (point queries)

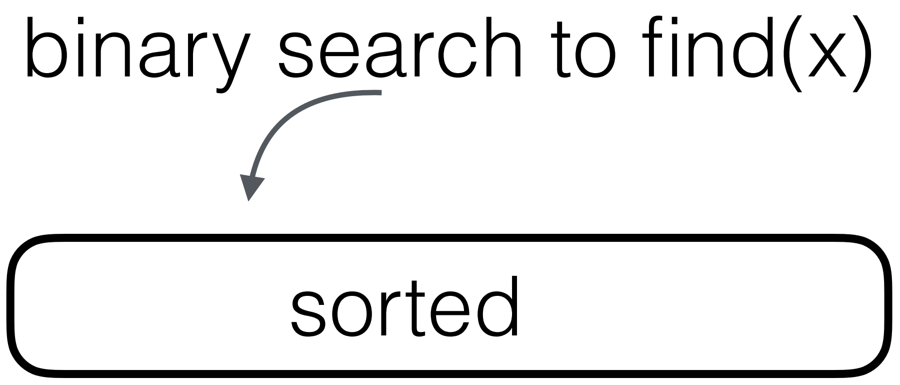


reads	✓
updates	✗
memory	✗

a perfect access method for reads (point queries)
but with no memory overhead



a perfect access method for reads (point queries)
but with no memory overhead

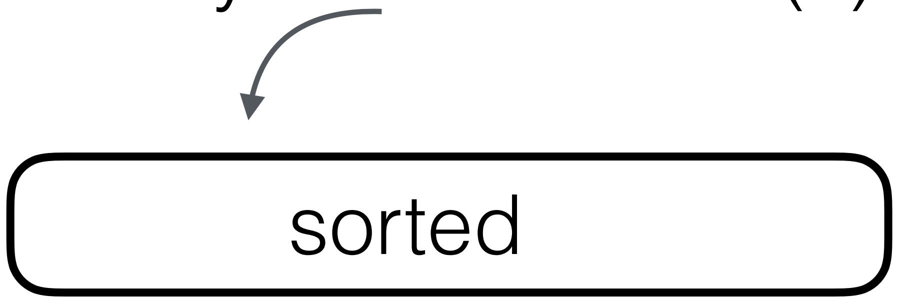


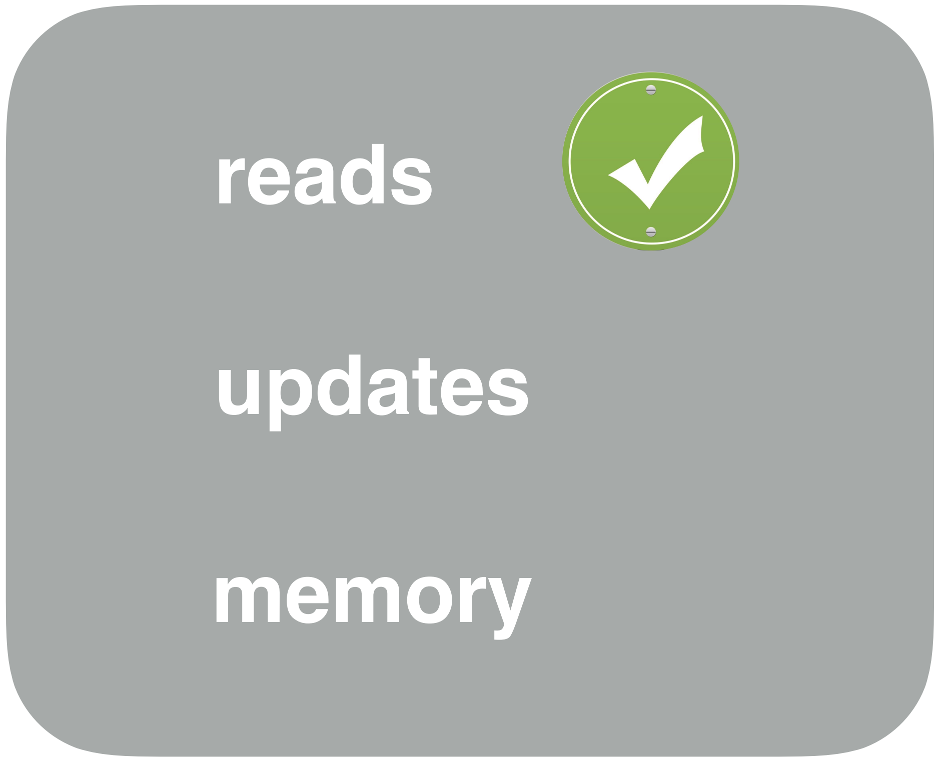
reads

updates

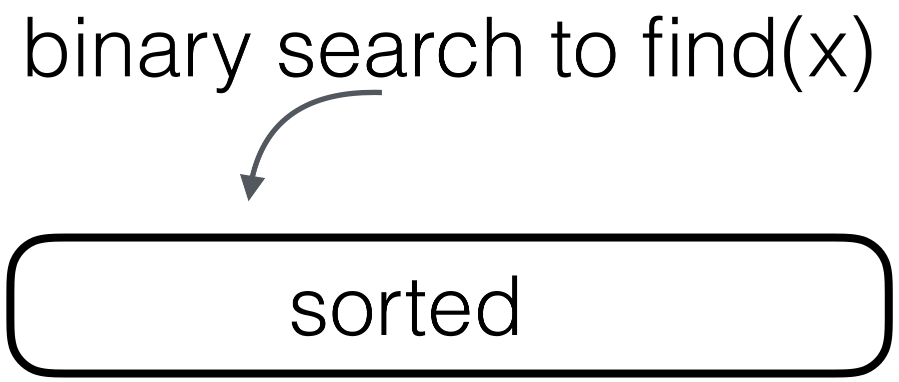
memory

a perfect access method for reads (point queries)
but with no memory overhead



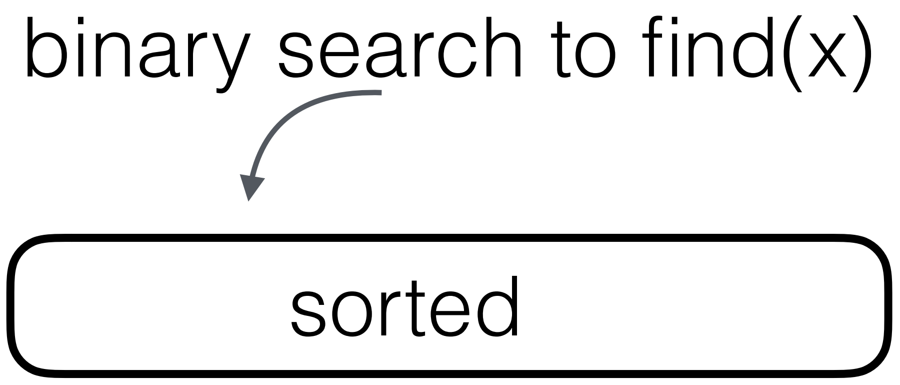


a perfect access method for reads (point queries)
but with no memory overhead



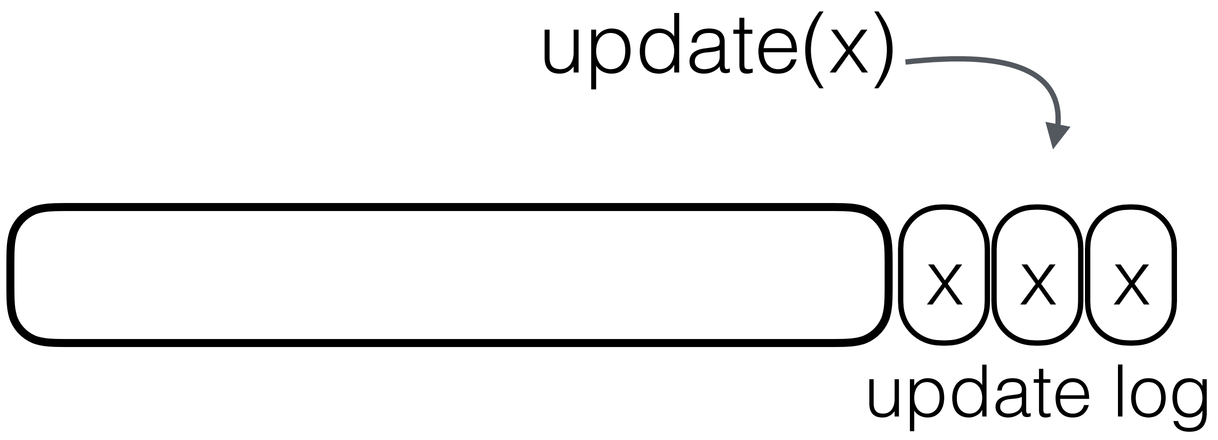
reads	✓
updates	✗
memory	

a perfect access method for reads (point queries)
but with no memory overhead

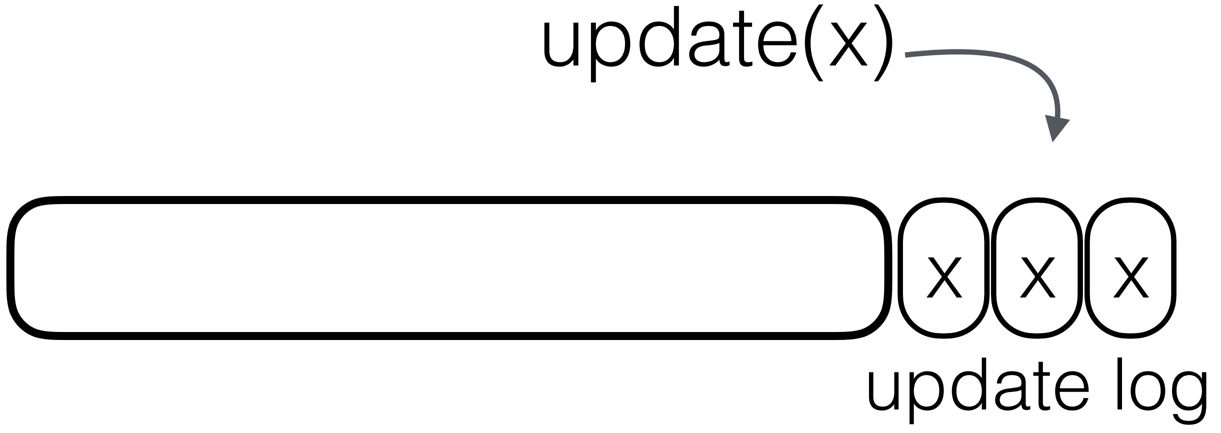


reads	✓
updates	✗
memory	✓

a perfect access method for writes (point writes)



a perfect access method for writes (point writes)

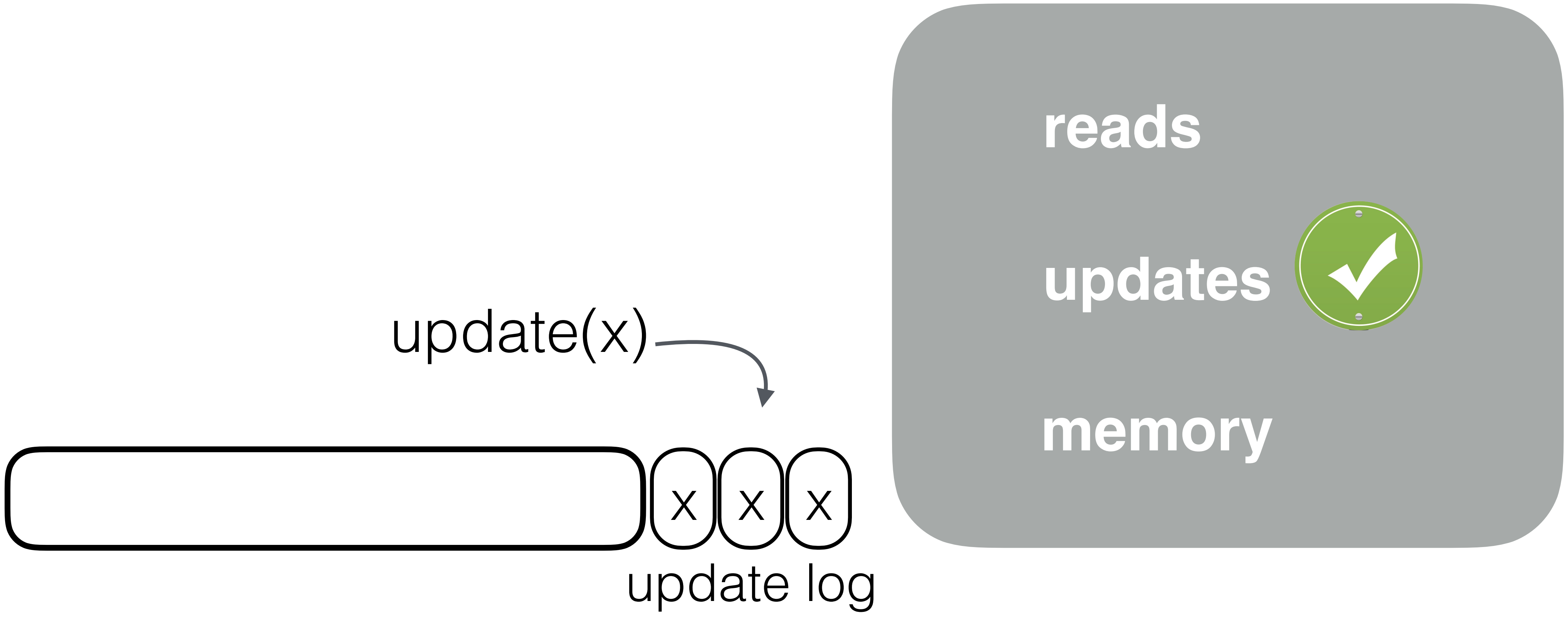


reads

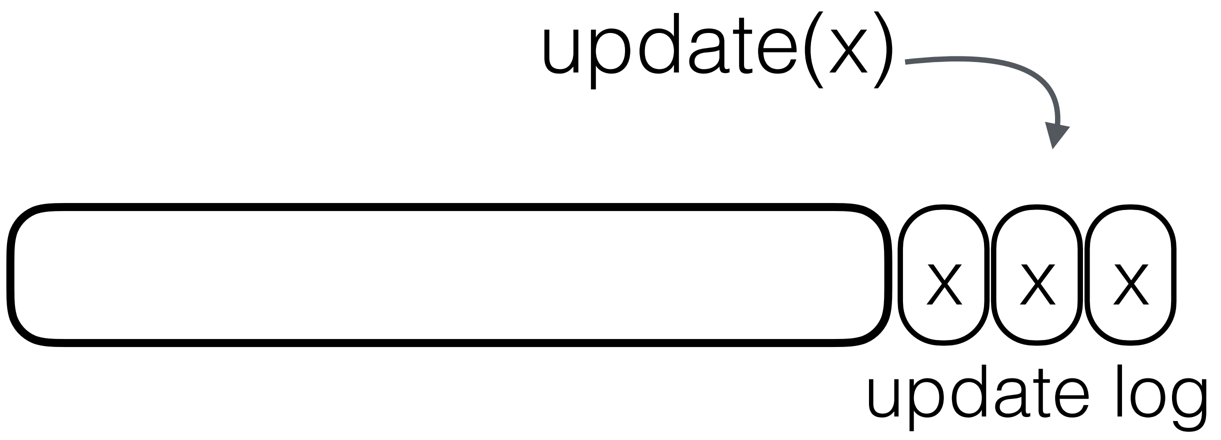
updates

memory

a perfect access method for writes (point writes)



a perfect access method for writes (point writes)

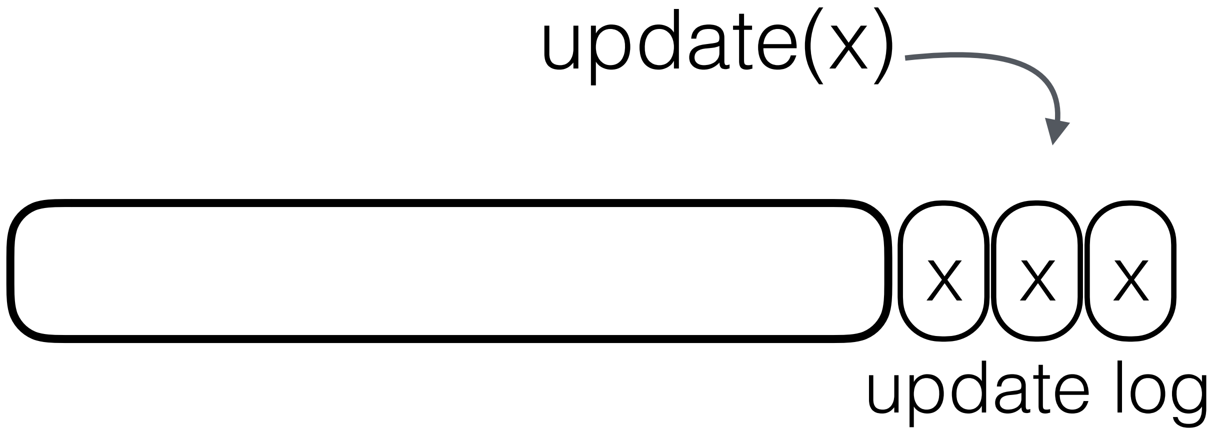


reads

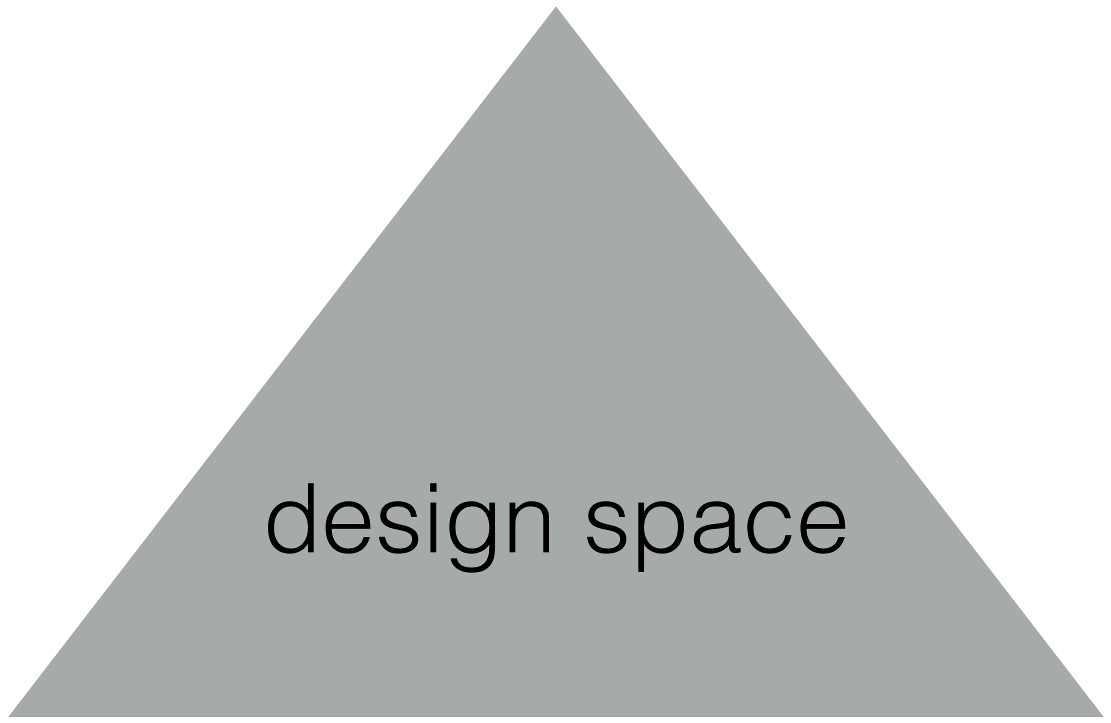
updates ✓

memory ✗

a perfect access method for writes (point writes)



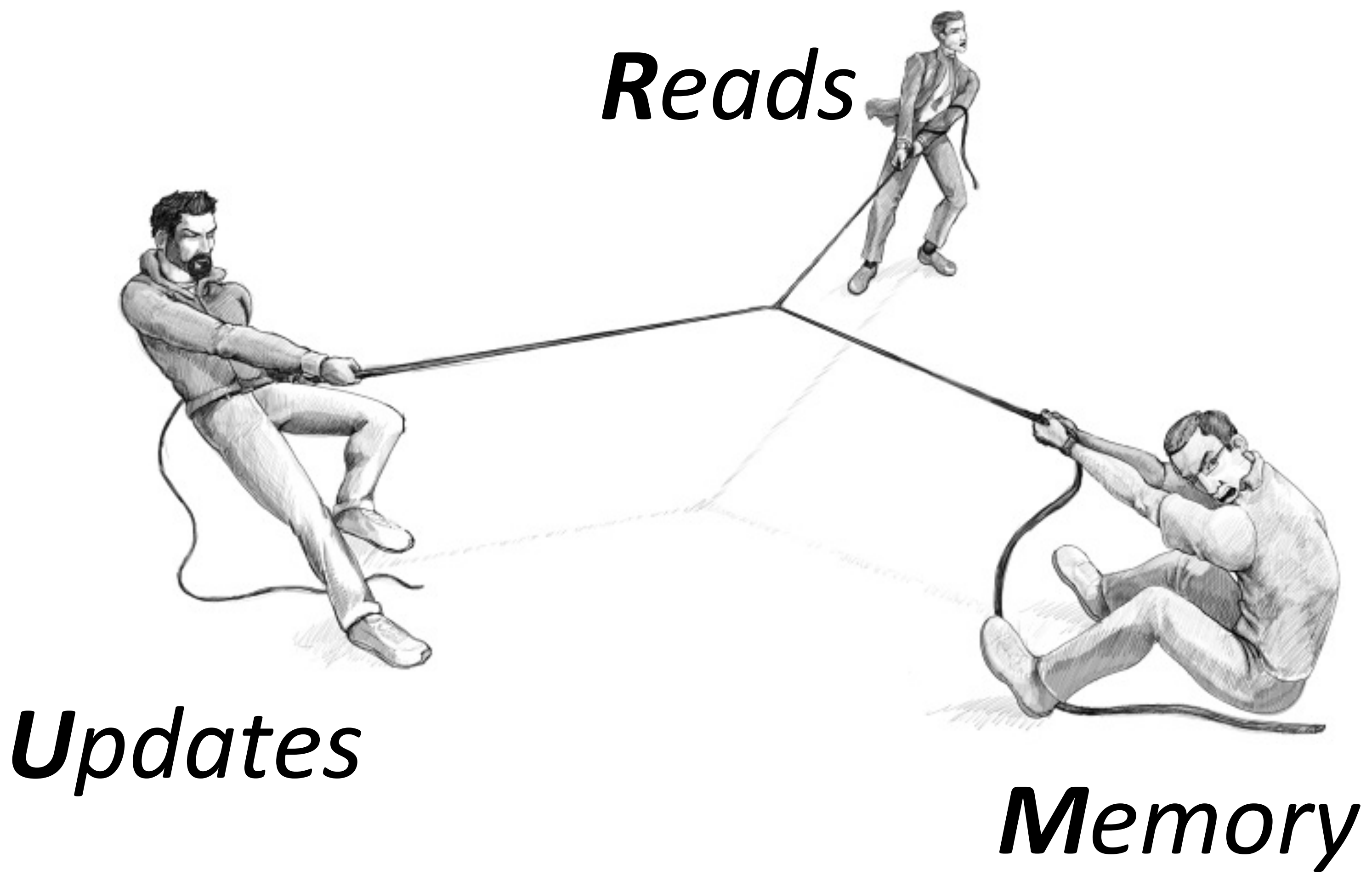
reads	✗
updates	✓
memory	✗

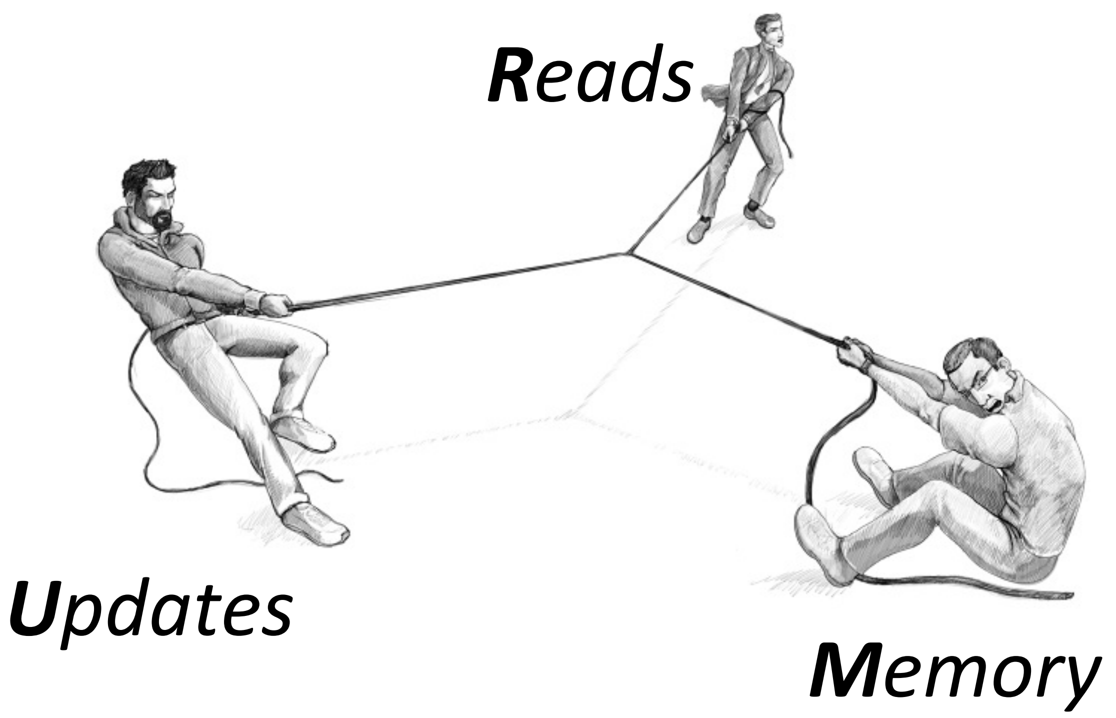


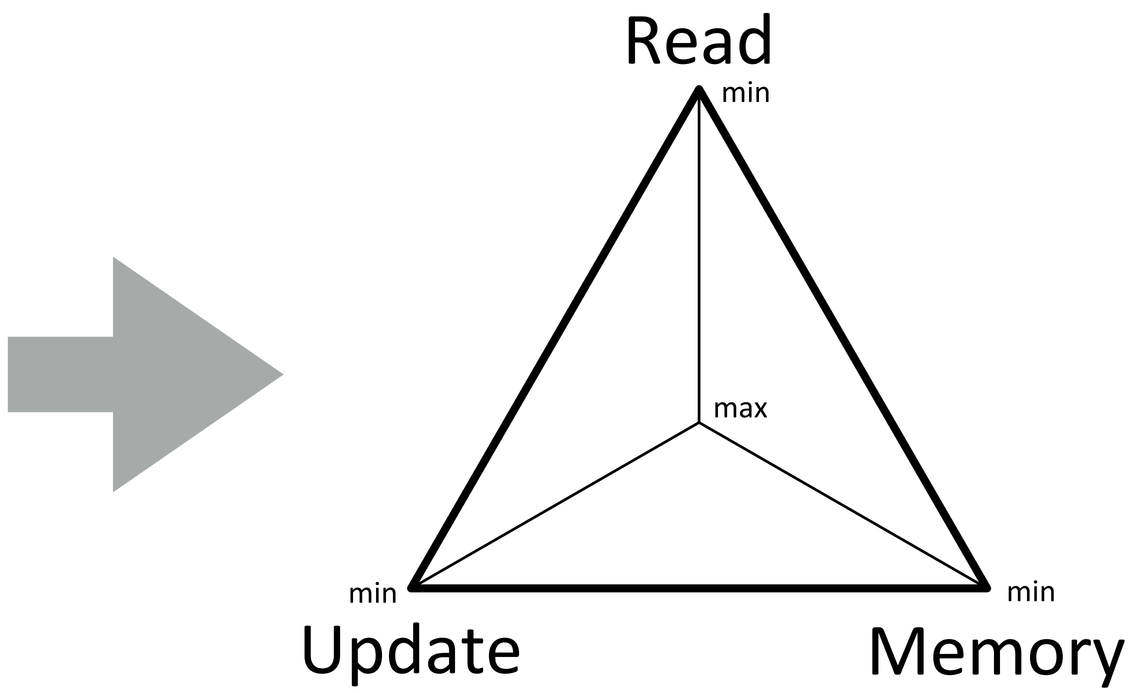
it all starts with how we store data

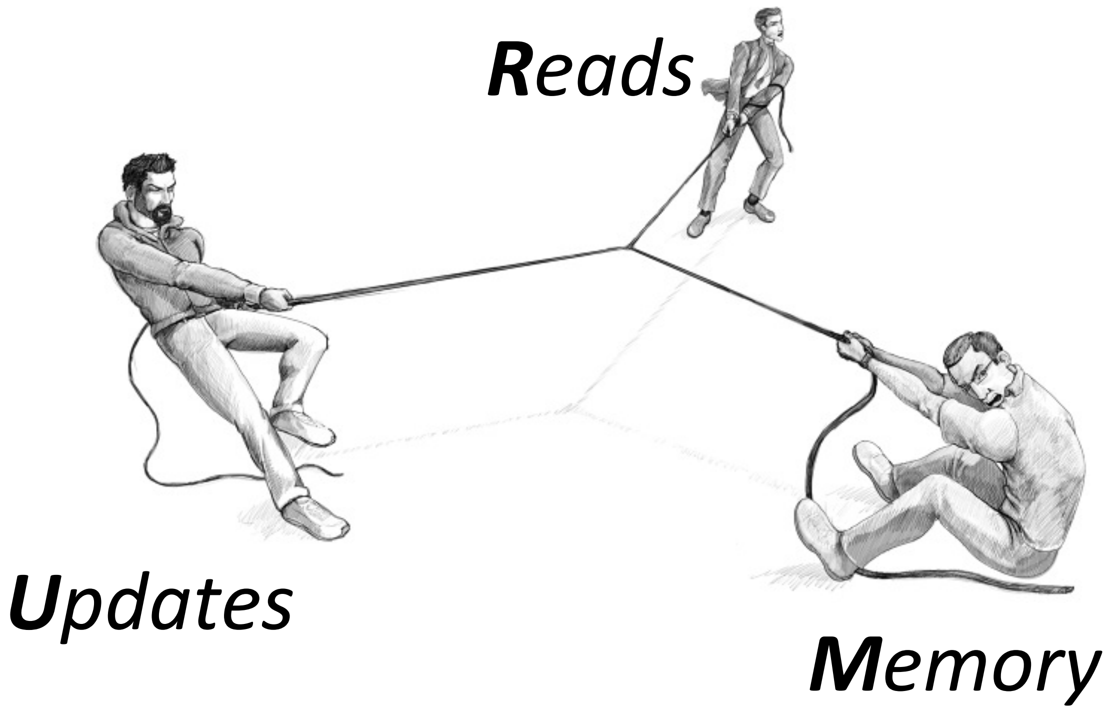
every bit matters

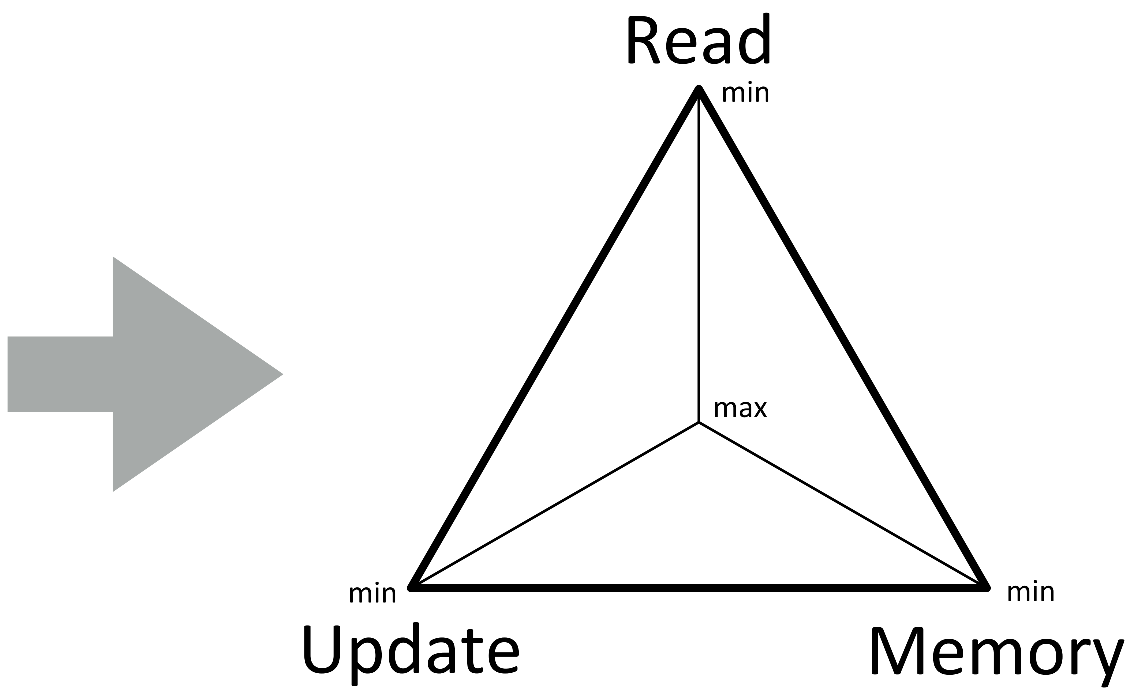
basic tradeoffs

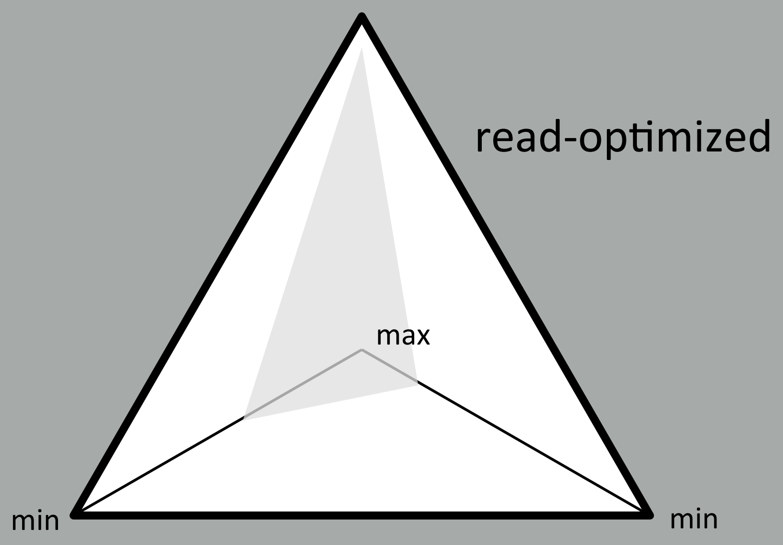


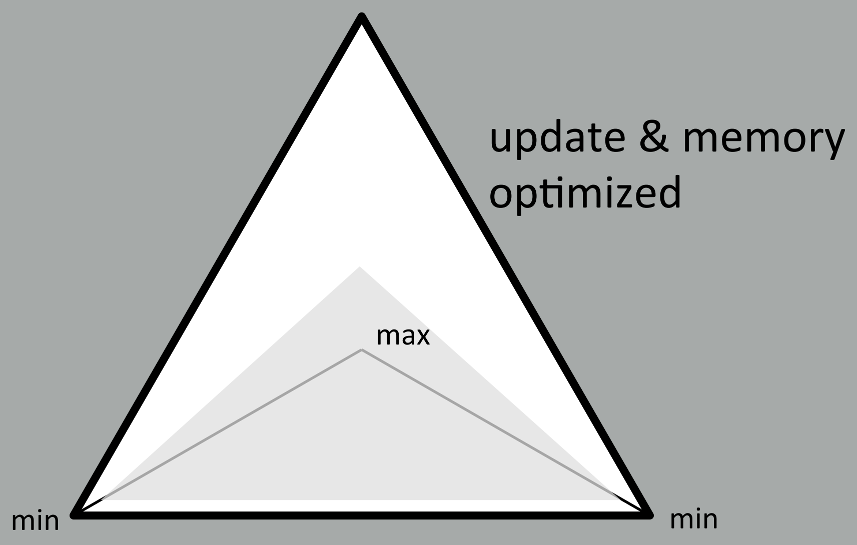

Reads
Updates
Memory

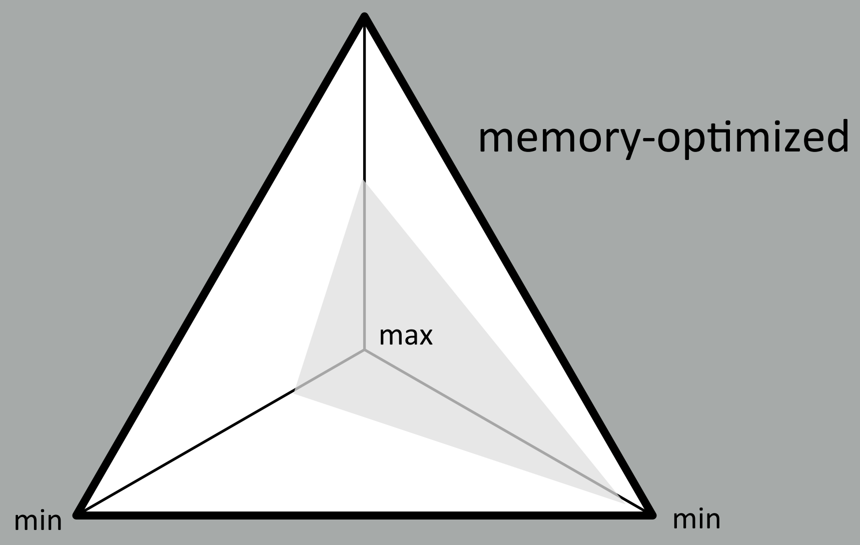

Read
min
max
min
Update
min
Memory


Reads
Updates
Memory


Read
min
max
min
Update
min
Memory


read-optimized
max
min
min


update & memory optimized
max
min
min


memory-optimized
max
min
min

DASlab

HARVARD School of Engineering and Applied Sciences

Fractional Cascading

Partitioning

Fractional Cascading

Log-structured Updates

Differential Updates

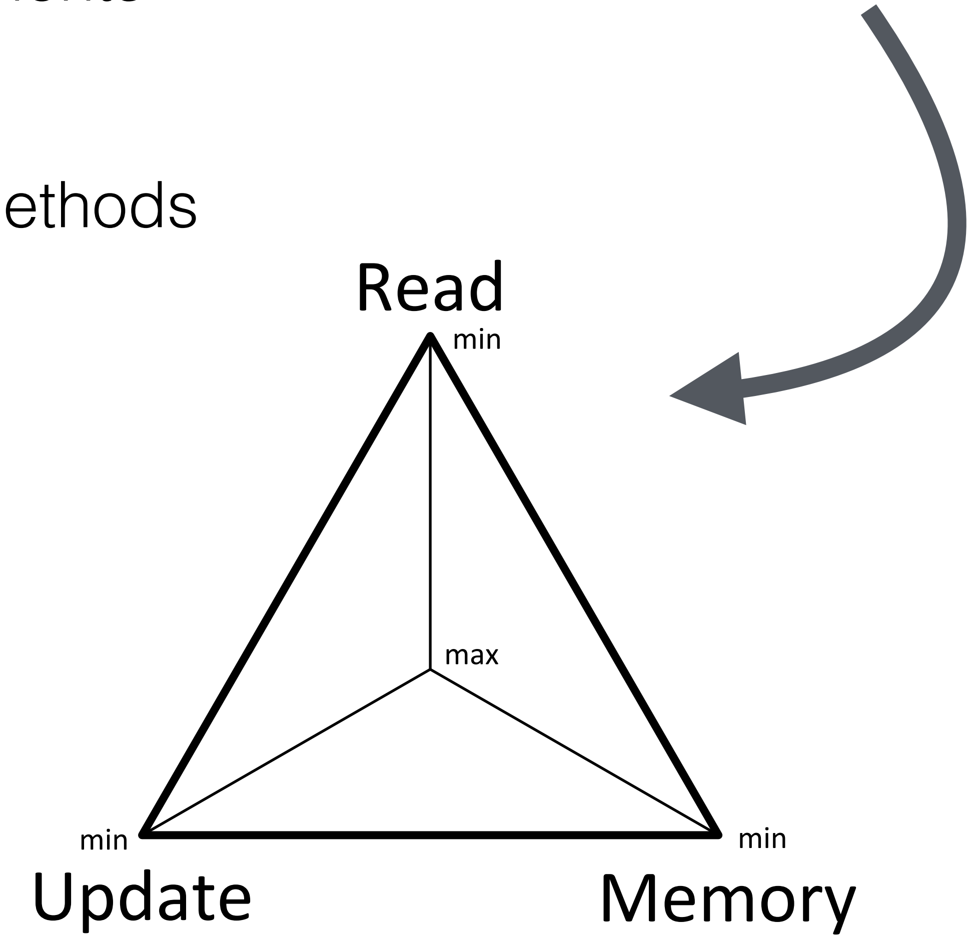
Sparse Indexing

Logarithmic Design

study basic access methods design components

how they affect the RUM tradeoffs

how are they combined in existing access methods



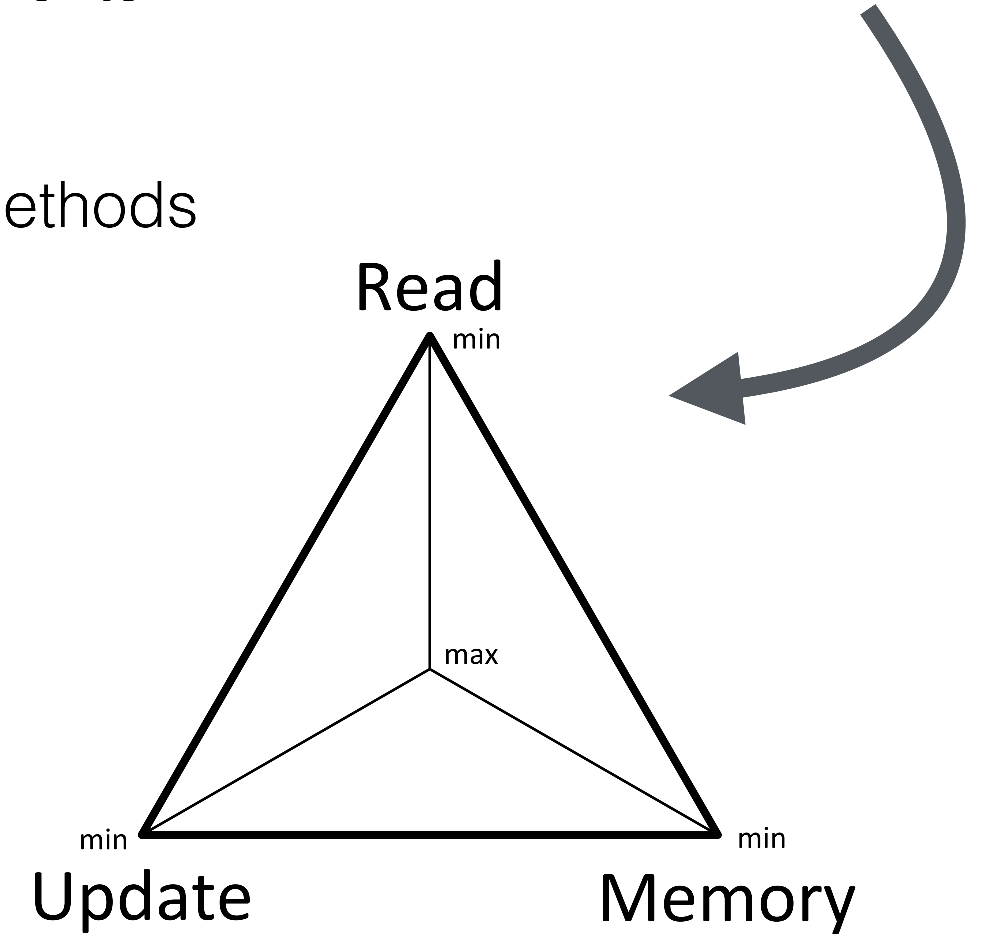
			Fractional Cascading	Partitioning
			Fractional Cascading	Log-structured Updates
			Differential Updates	Sparse Indexing	Logarithmic Design

study basic access methods design components

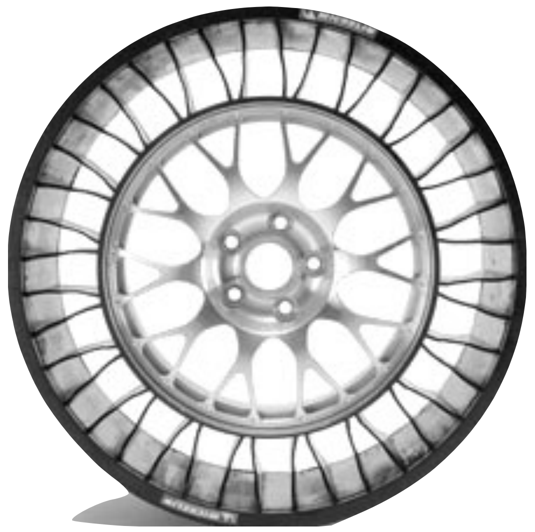
how they affect the RUM tradeoffs

how are they combined in existing access methods

Part 2

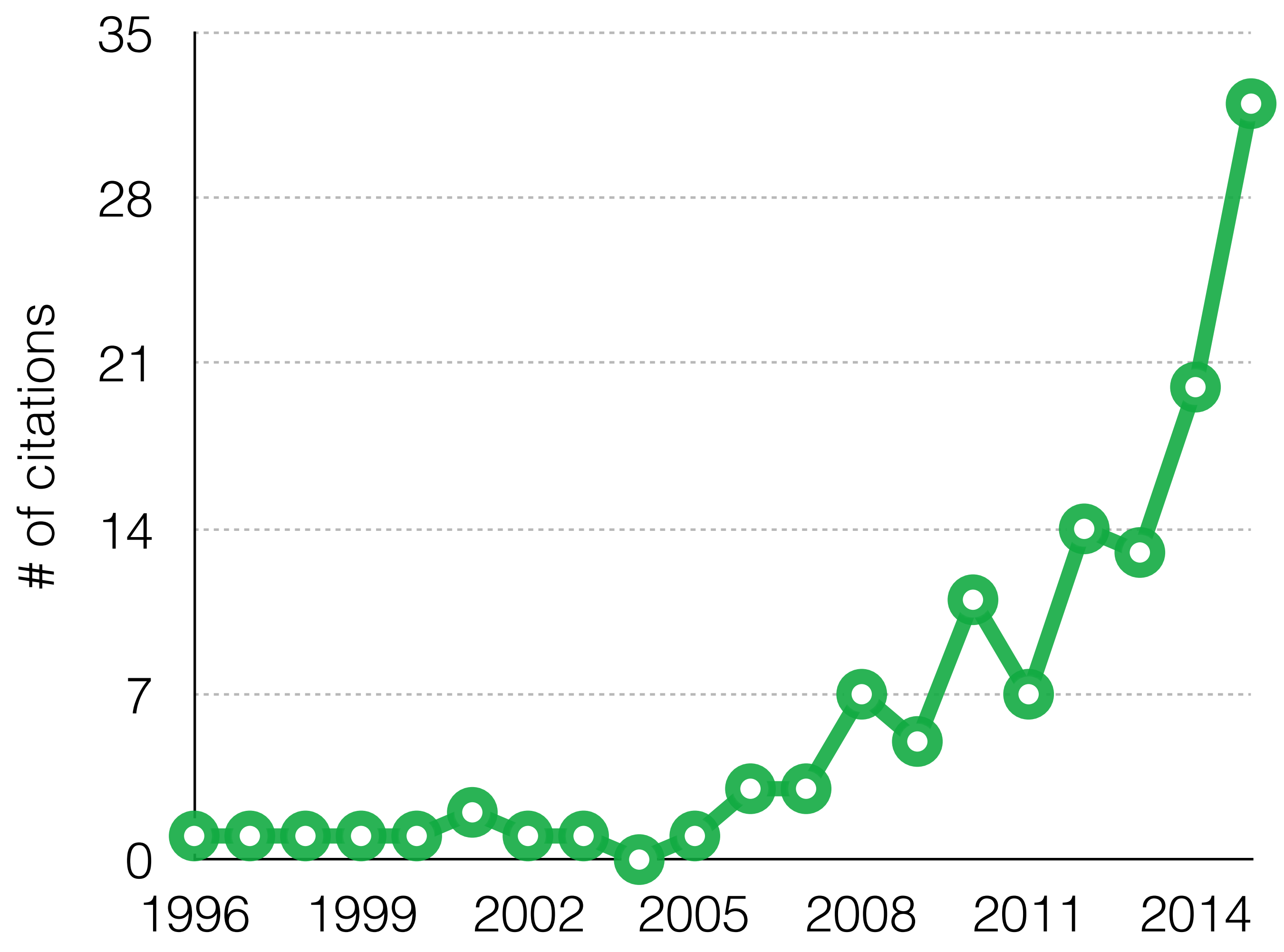


can we make it easy to design/tune access methods?



disk memory flash . . .

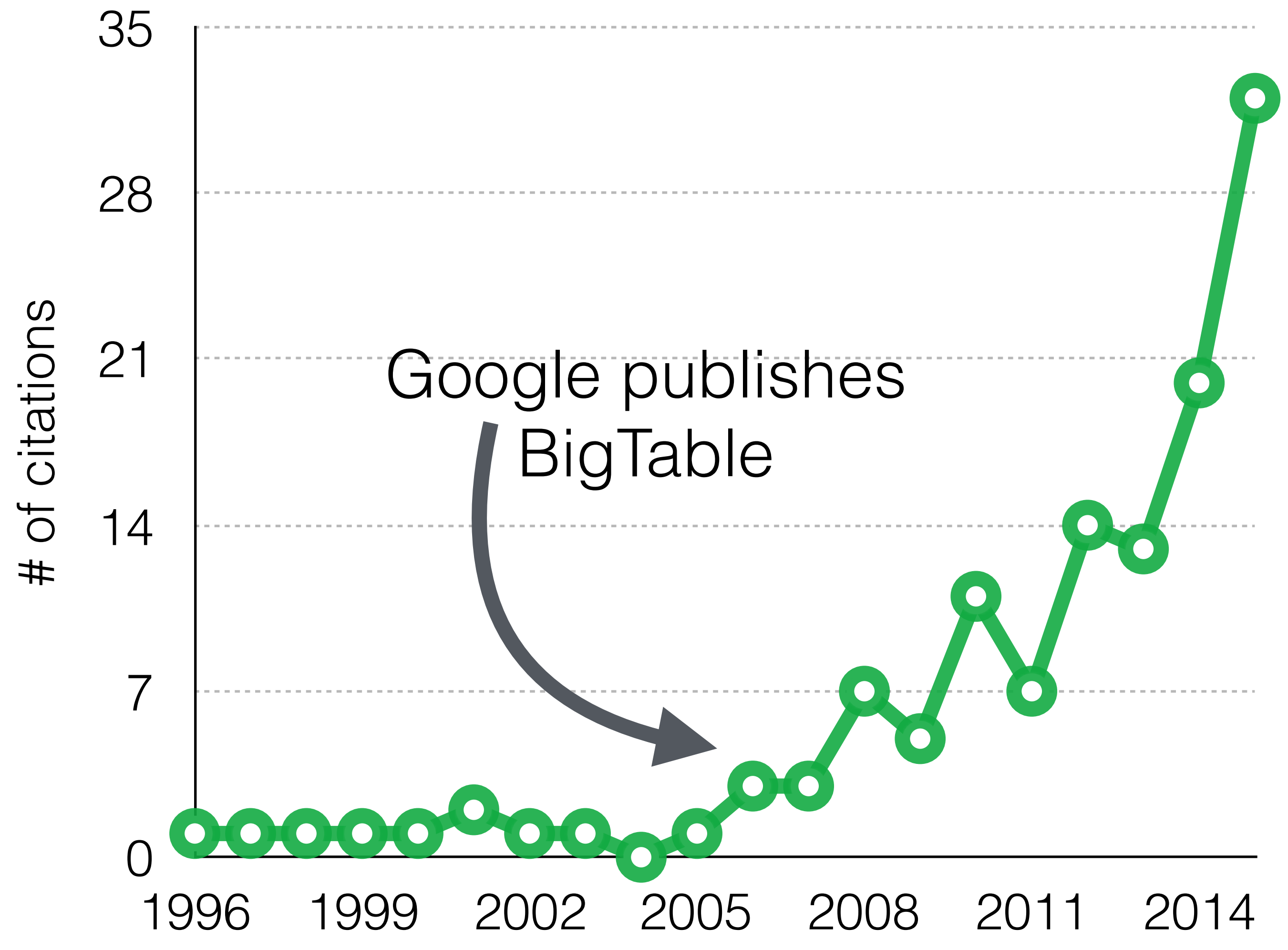
1 easily utilize past concepts



P. O'Neil, E. Cheng, D. Gawlick, E, O'Neil
The log-structured merge-tree (LSM-tree)
Acta Informatica 33 (4): 351–385, 1996

2

do not miss out on cool ideas and concepts

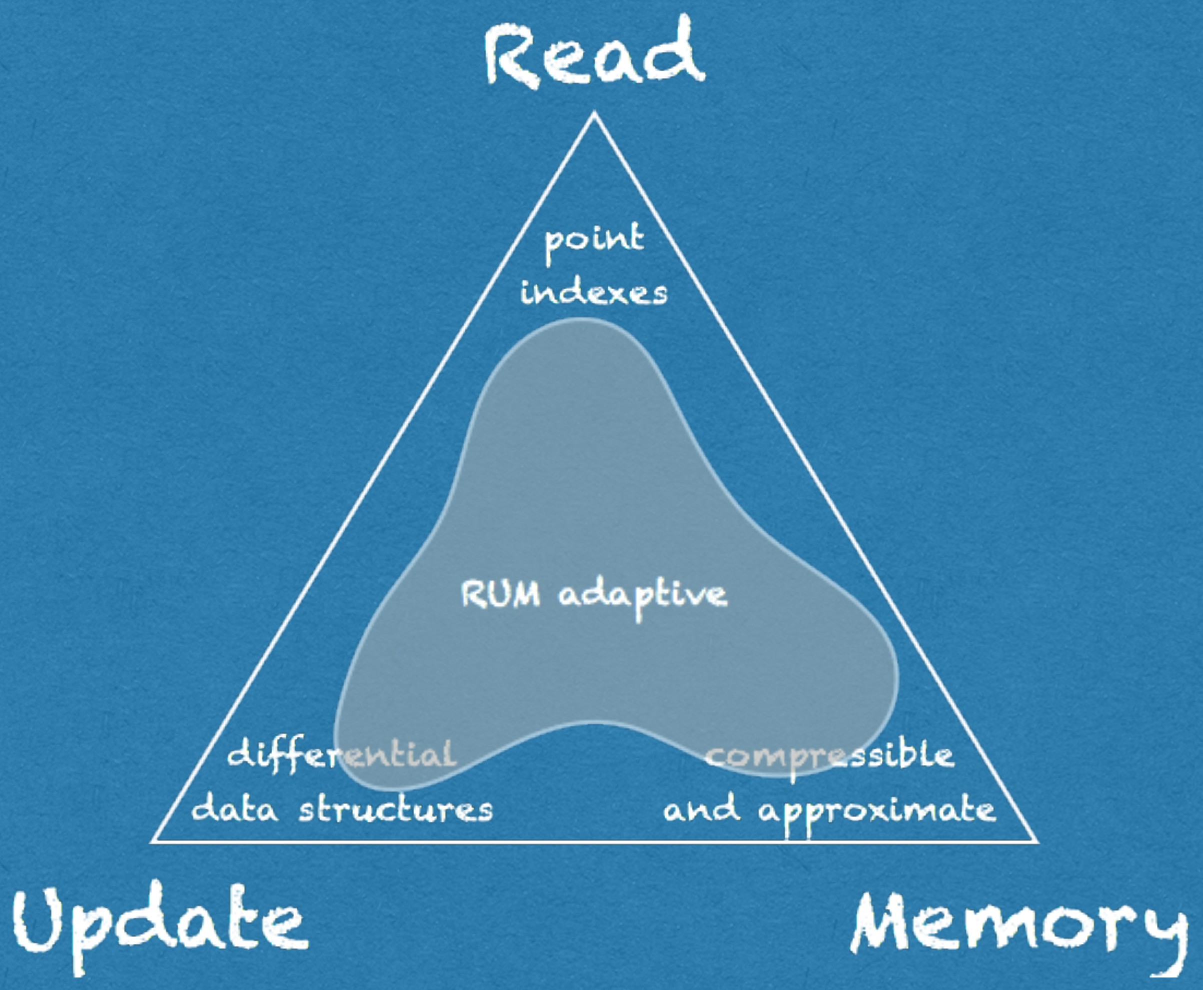


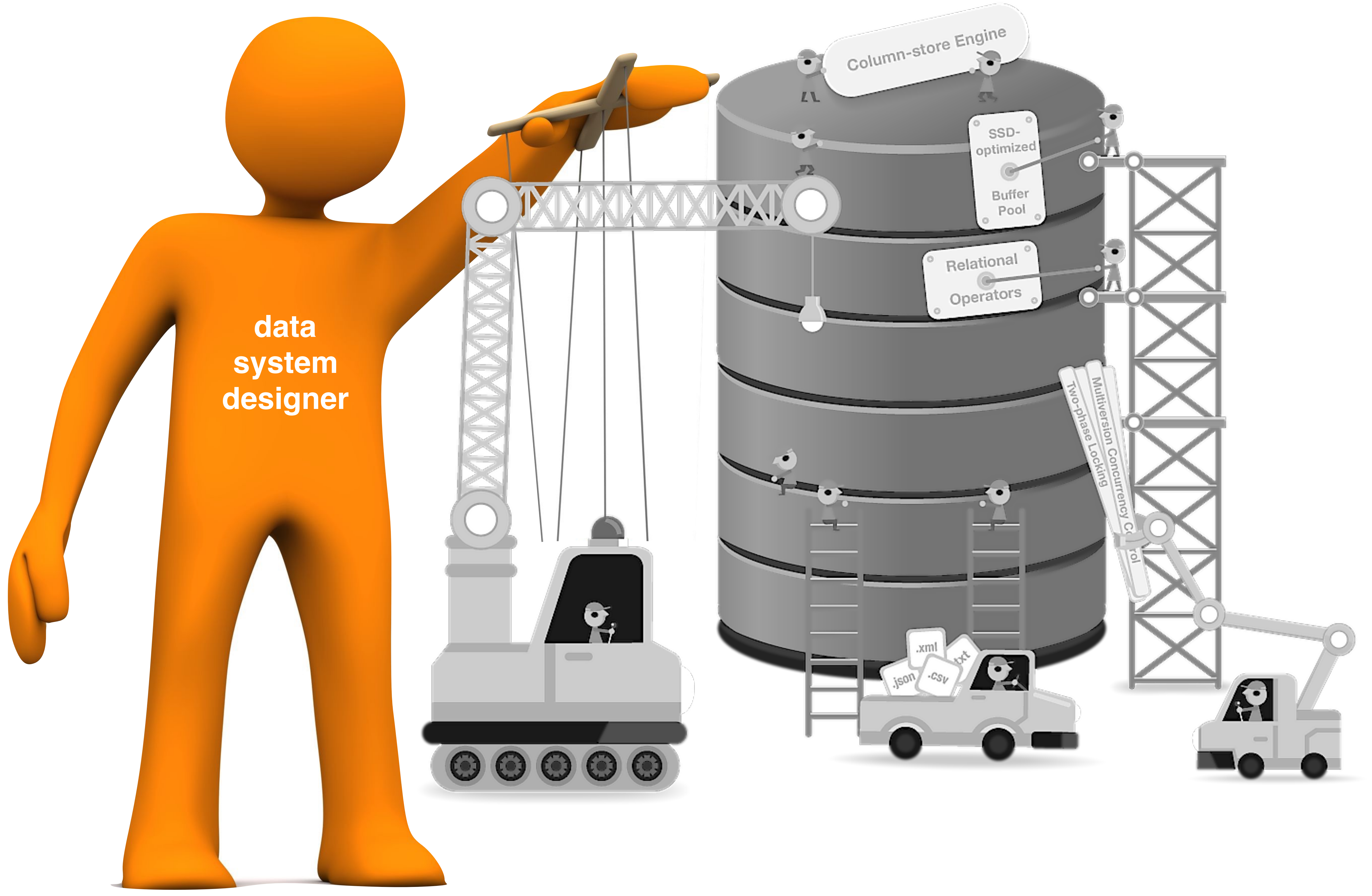
P. O'Neil, E. Cheng, D. Gawlick, E, O'Neil
The log-structured merge-tree (LSM-tree)
Acta Informatica 33 (4): 351–385, 1996

2 do not miss out on cool ideas and concepts

move from design based on intuition & experience only to a more formal and systematic way to design systems

construct access methods
out of basic components
(and their tradeoffs)
e.g., scan*, tree*, bloom filters,
bitmaps, hash tables, etc.





INTERACTIVE DATA SYSTEM DESIGN/TUNING/TESTING

once we have a “complete” & navigable set of design modules

learn from: s/w engineering, modular dbs, compilers,
goes all the way back to basic texts

once we have a “complete” & navigable set of design modules

learn from: s/w engineering, modular dbs, compilers,
goes all the way back to basic texts

easy to change/adapt

easy to design

once we have a “complete” & navigable set of design modules

learn from: s/w engineering, modular dbs, compilers,
goes all the way back to basic texts

easy to change/adapt

easy to design

universal
development
platform

testing

once we have a “complete” & navigable set of design modules

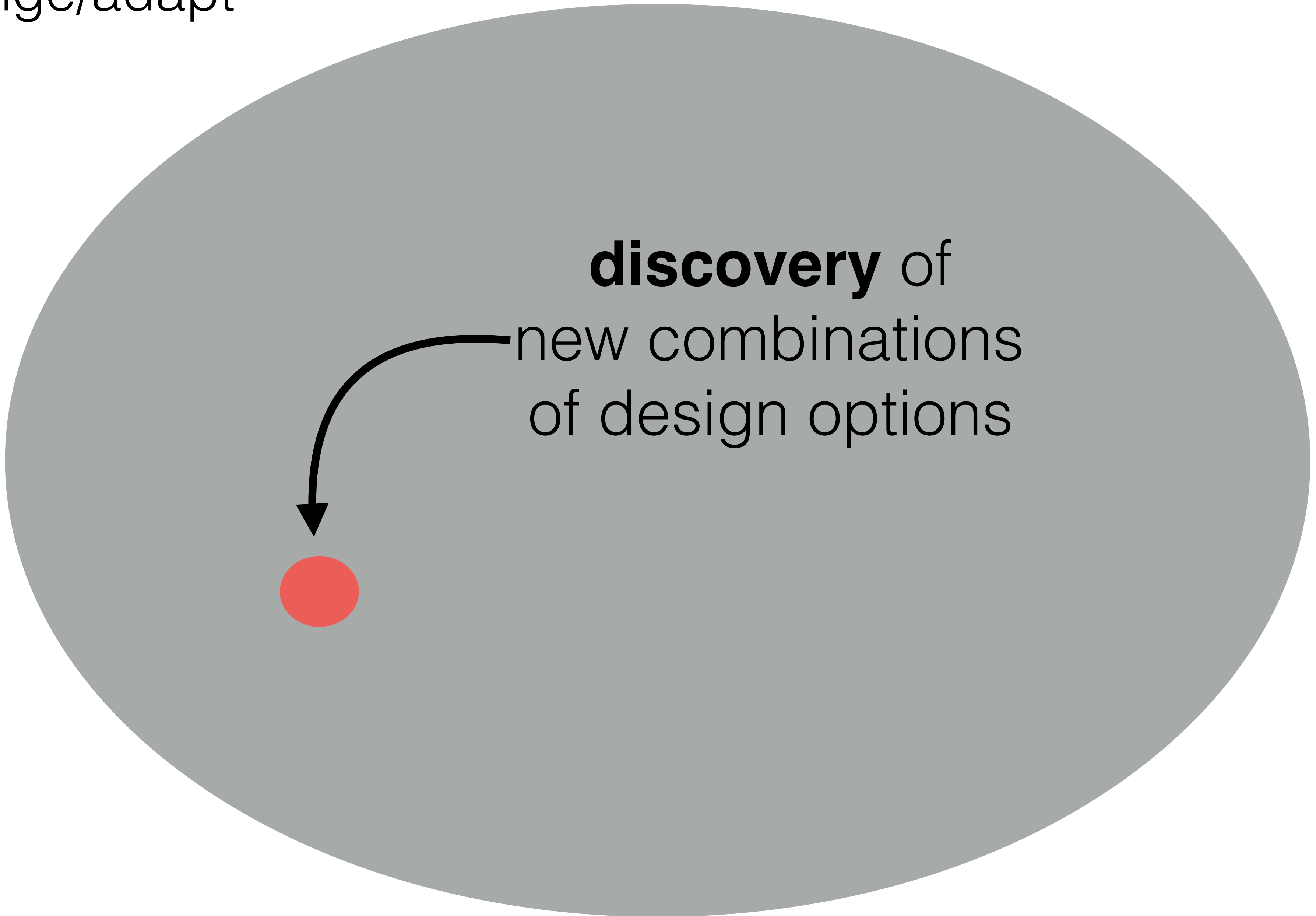
learn from: s/w engineering, modular dbs, compilers, goes all the way back to basic texts

easy to change/adapt

easy to design

universal development platform

testing



Part 2: observe how papers fill in gaps in the structure and existing open gaps