

COSI 167A

Advanced Data Systems

Class 3

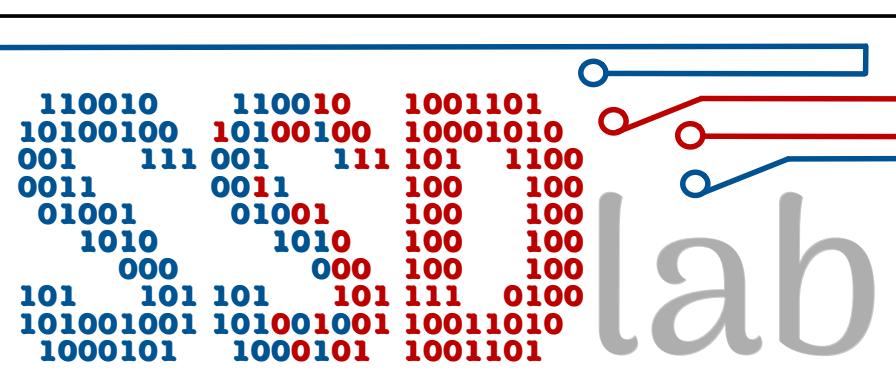
Data Systems Architecture

Prof. Subhadeep Sarkar



Brandeis
UNIVERSITY

<https://ssd-brandeis.github.io/COSI-167A/>



Today in COSI 127B

What's on the cards?

fundamentals of data storage

introduction to row-stores and column-stores

Class logistics

and administrivia

Project 1 (C++/Java) has been released (due on **Sep 20**).

C/C++ learning resources at: <https://ssd-brandeis.github.io/COSI-167A/assignments/>

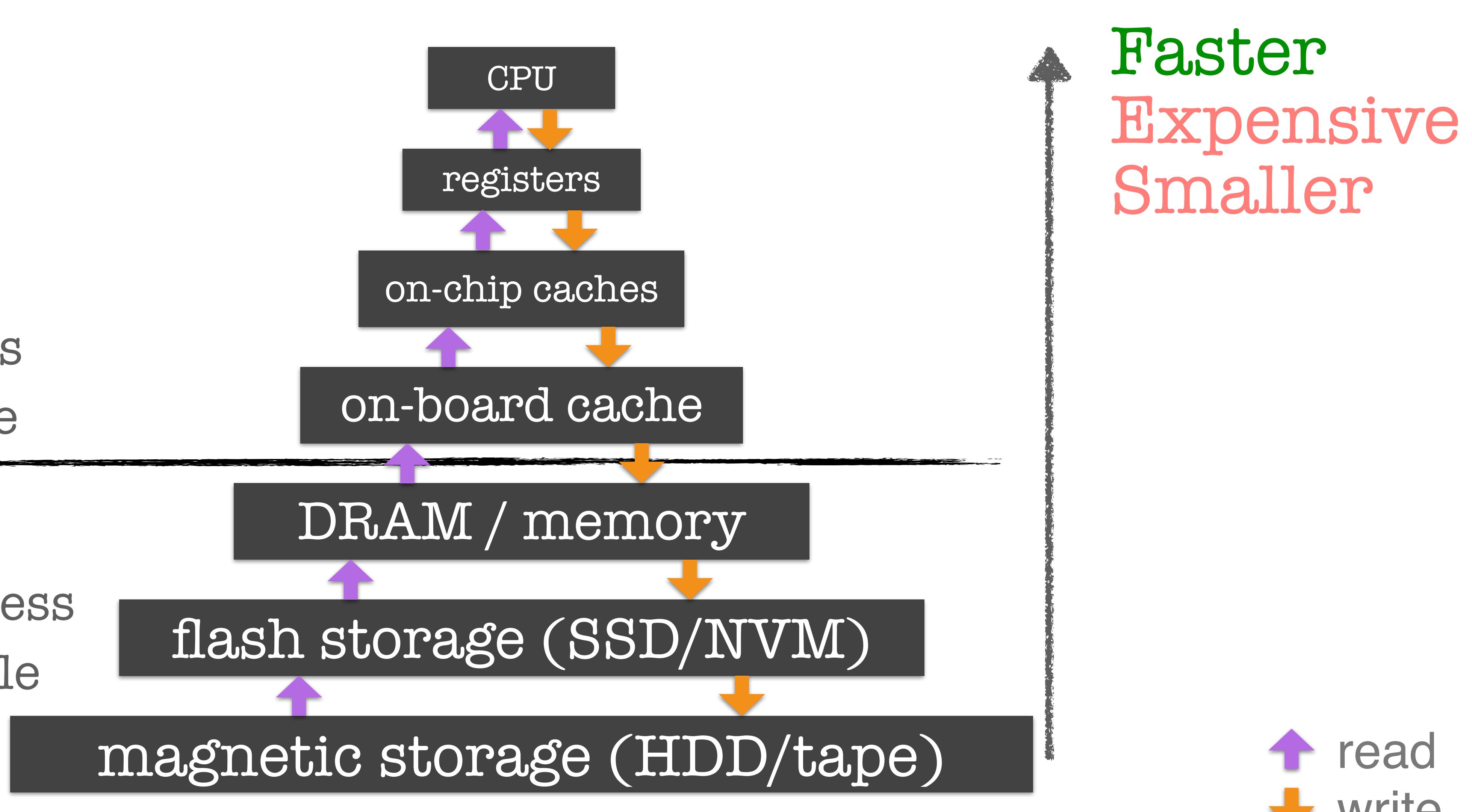
The **first technical question** is now available on the class website
(due **before the class** on **Sep 10**).

Recap: Storage hierarchy

How data moves!

Volatile
Random access
Byte accessible

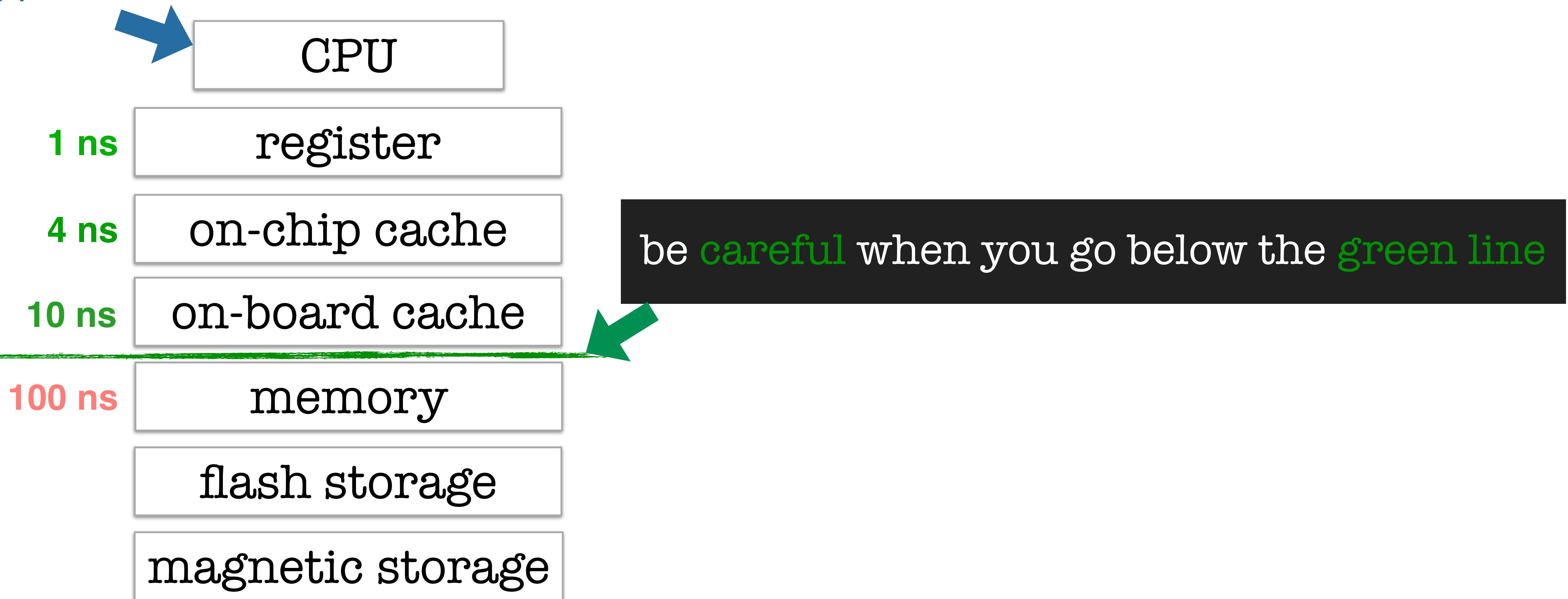
Non-volatile
Sequential access
Block accessible



Memory wall

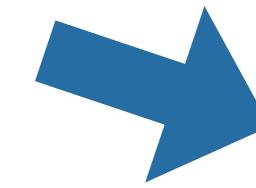
computations
happen here

Try not to jump the wall

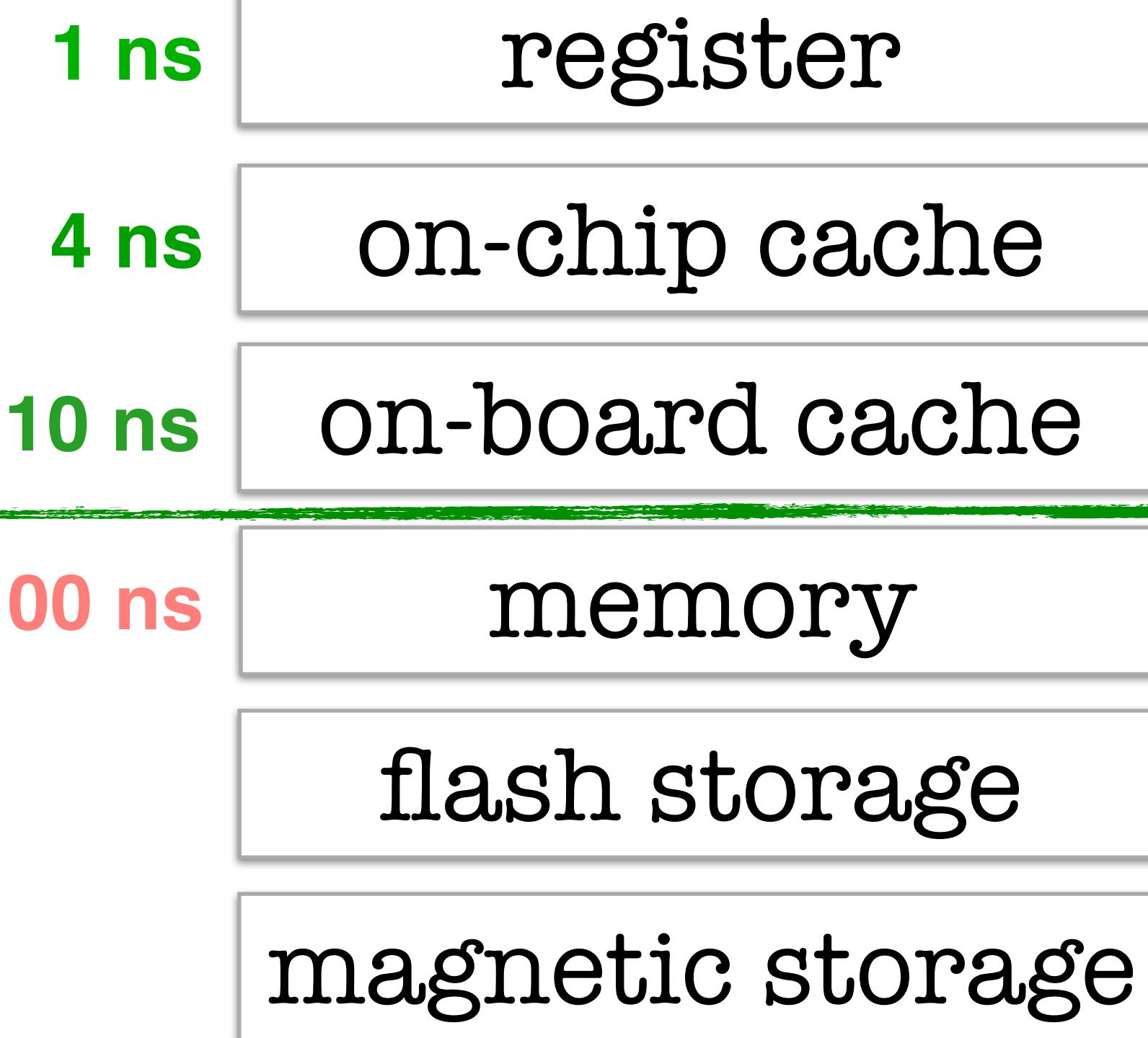


Memory wall

computations
happen here

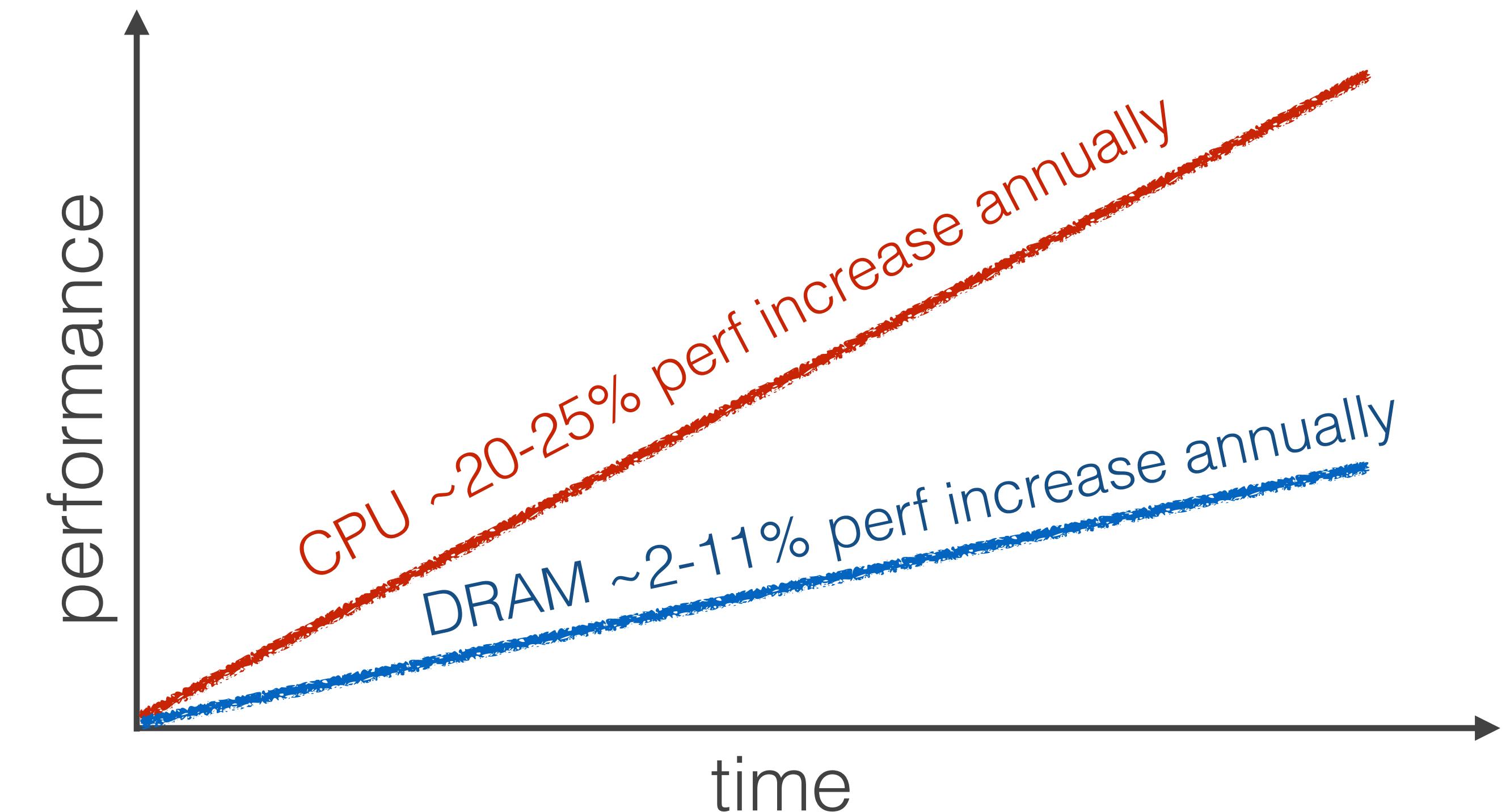


CPU



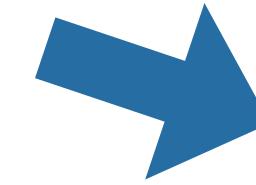
Try not to jump the wall

be **careful** when you go below the **green line**



Memory wall

computations
happen here



CPU

1 ns

register

4 ns

on-chip cache

10 ns

on-board cache

100 ns

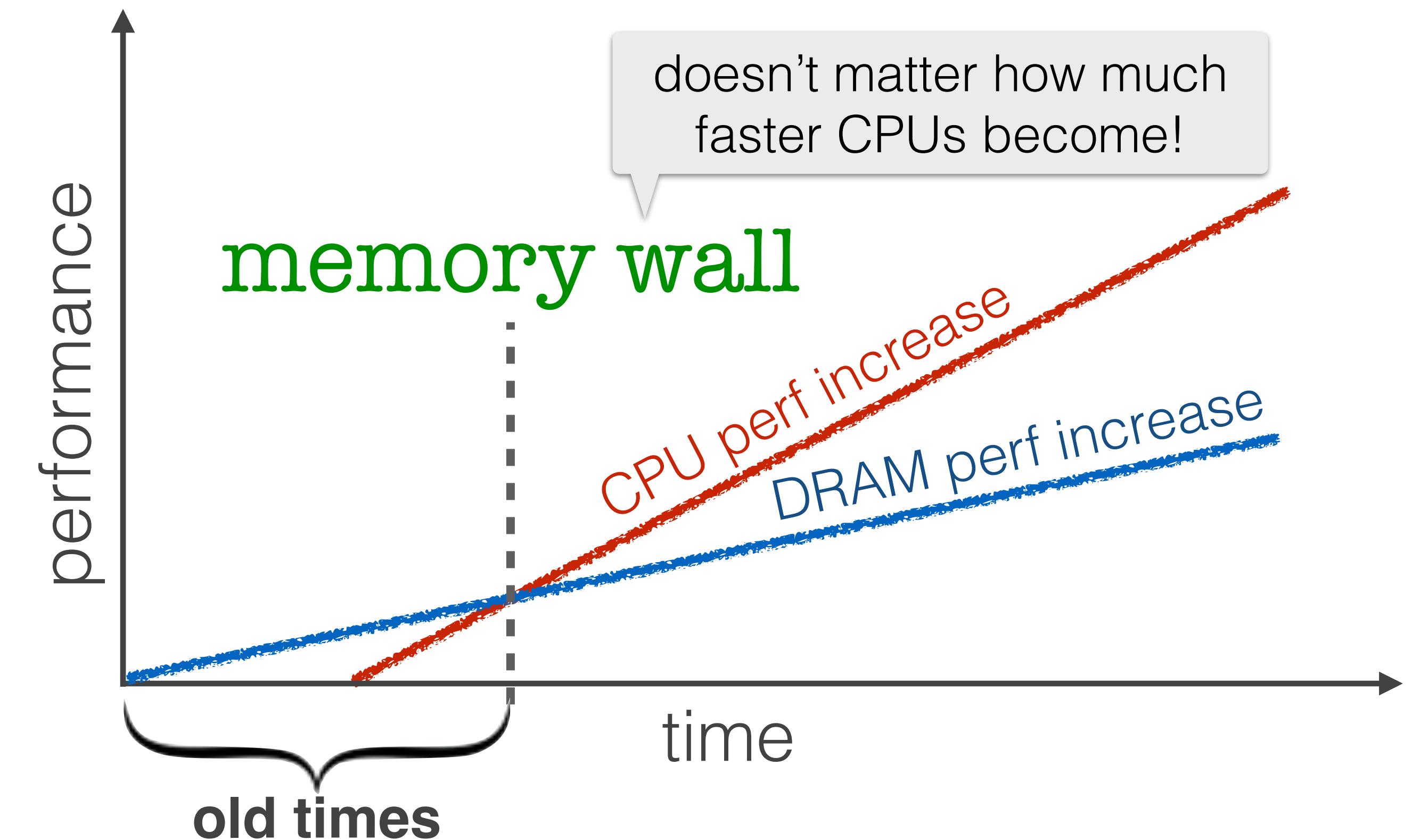
memory

flash storage

magnetic storage

Try not to jump the wall

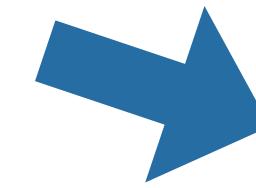
be **careful** when you go below the **green line**



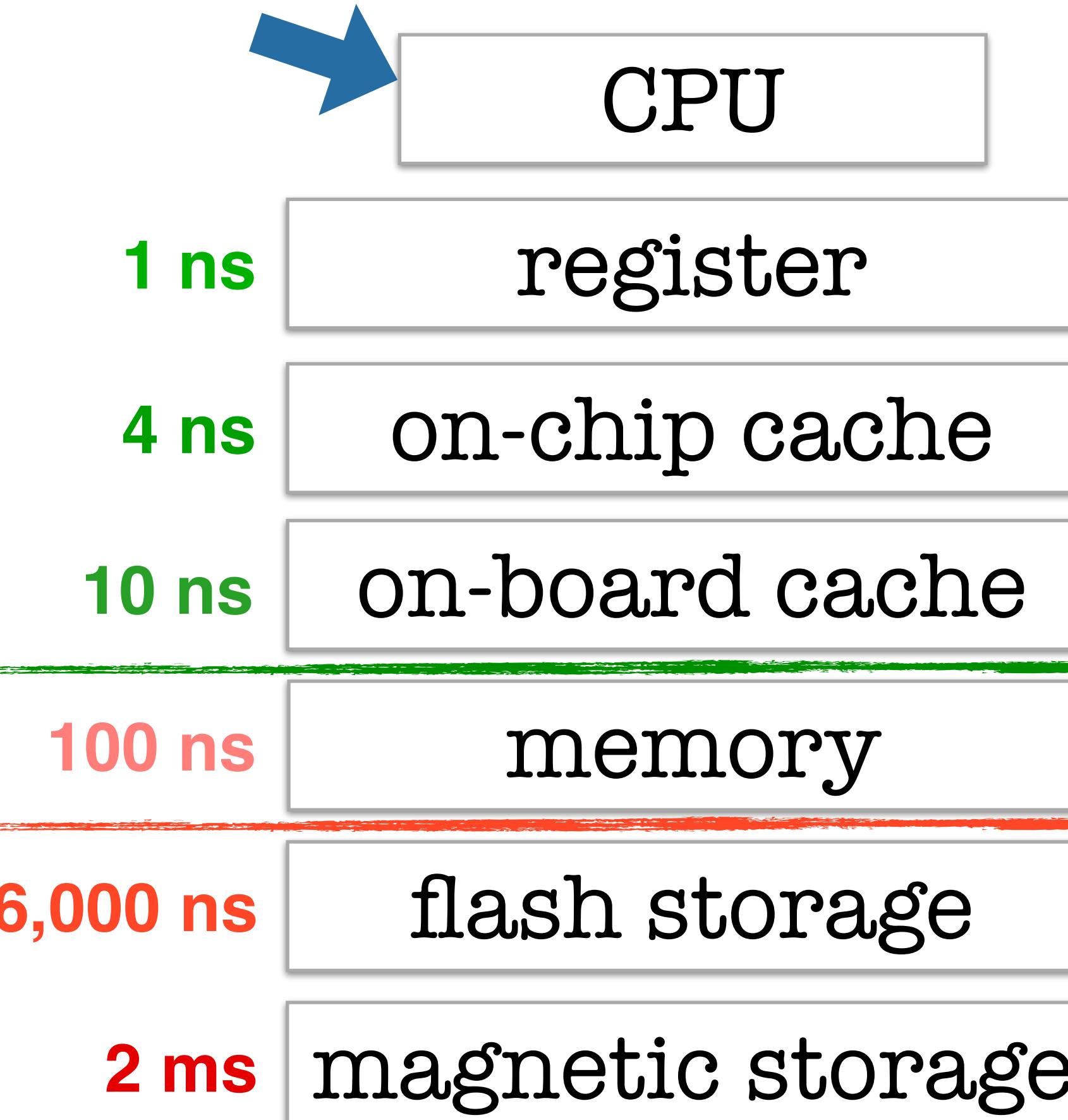
Brandeis
UNIVERSITY

Memory wall

computations
happen here



Try not to jump the wall



be **careful** when you go below the **green line**

be **VERY careful** when you go below the **red line**

Recap: **Storing** data

Things to keep in mind

Disk is **6 orders** of magnitude **slower** than CPU

SSDs are **4 orders** of magnitude **slower**

Memory is **3 orders** of magnitude **slower**

Recap: Random vs. Sequential access

So, be VERY careful!

Avoid disk accesses (reads/writes) whenever possible

I/Os to secondary storage is *always slow!*

Sequential access

read **each block exactly once**; process it; discard it; read next block

modern hardware can predict and **prefetch**; maximize performance

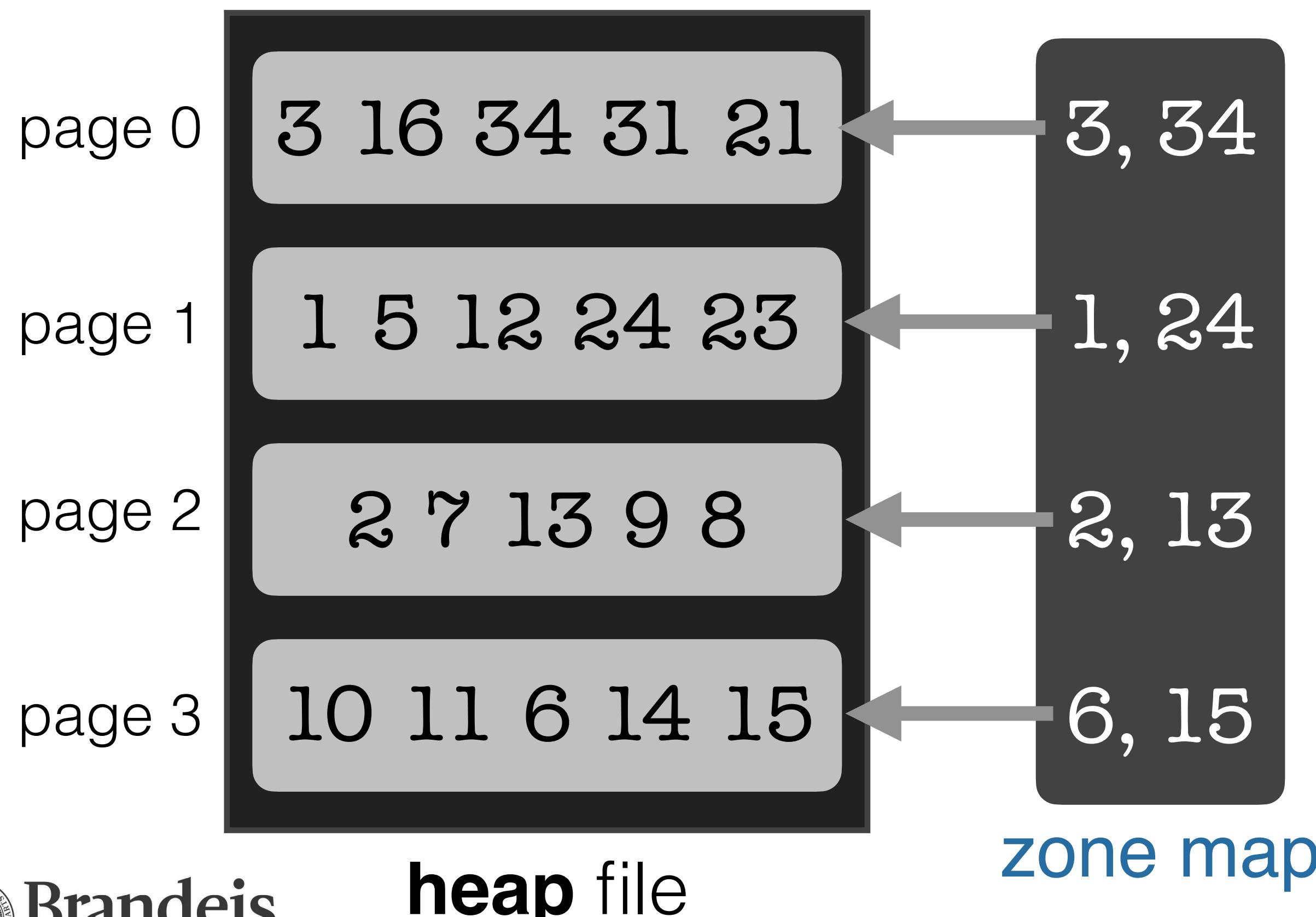
Random access

read a block; process it **partially**; discard it; may **read** the block **again**
often leads to **read amplification**

Project 1

Testing the waters!

Implementing a Simple Zone Map



w/o ZM: queries take **4 I/Os**

with ZM: query: $x < 4$: **3 I/Os**

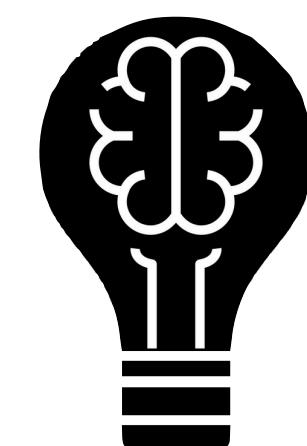
$x < 12$: **4 I/O**

$x = 1$: **1 I/O**

$x = 20$: **4 I/O**

Thought Experiment

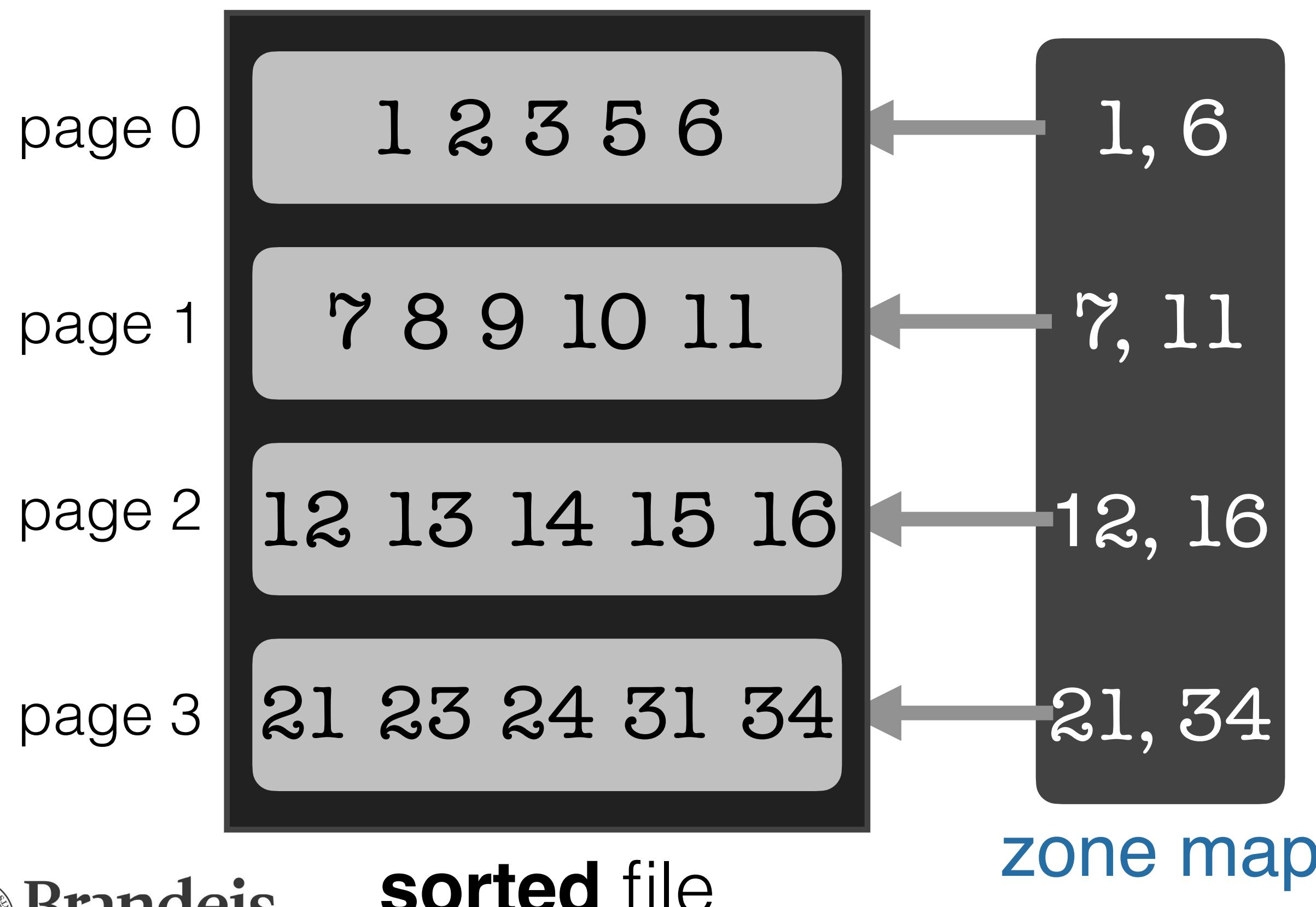
Are **zone maps** more or less useful if data is **sorted**?



Project 1

Testing the waters!

Implementing a Simple Zone Map



w/o ZM: queries take **4 I/Os**

with ZM: query: $x < 4$: **1 I/O**

$x < 12$: **2 I/Os**

$x = 1$: **1 I/O**

$x = 20$: **0 I/Os**

Sorting is inherent indexing!

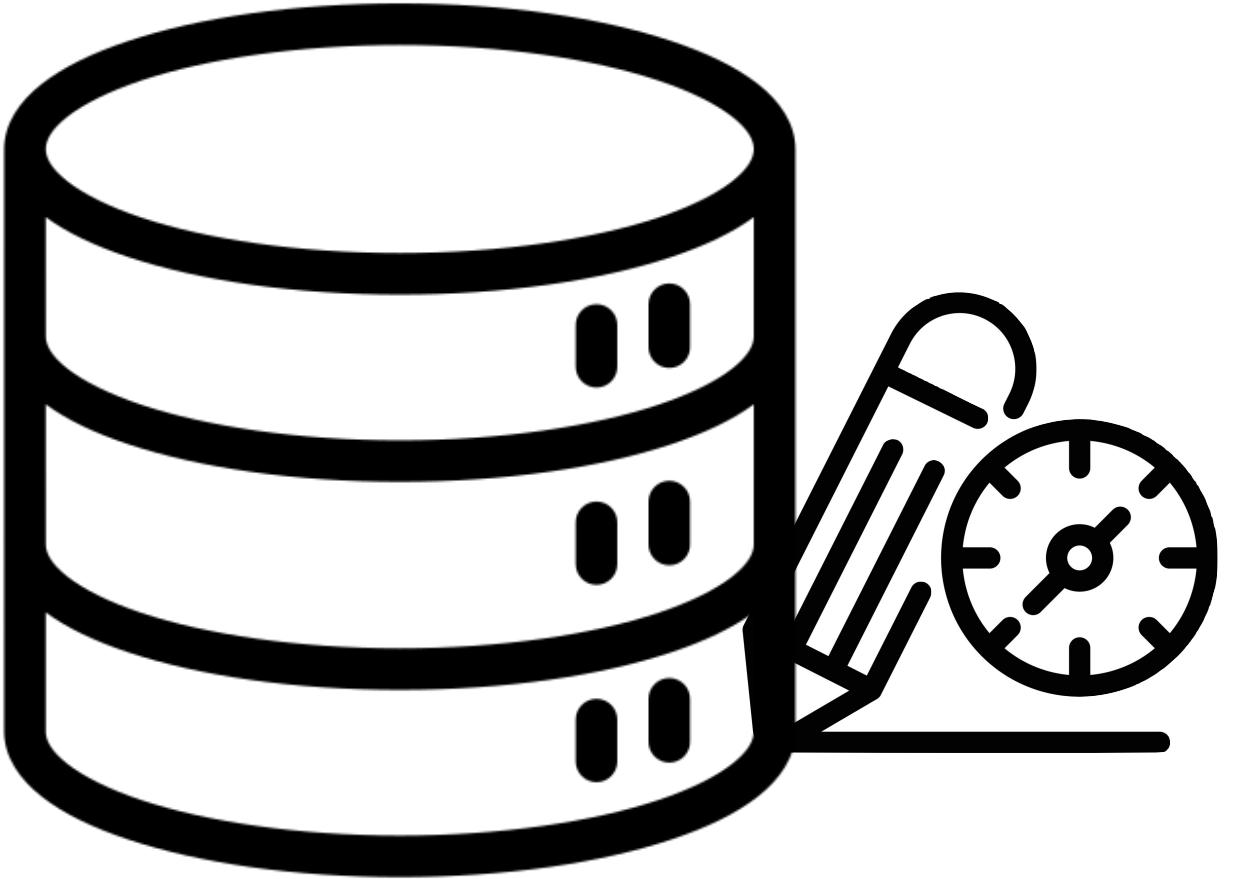
Zone map is a second index!



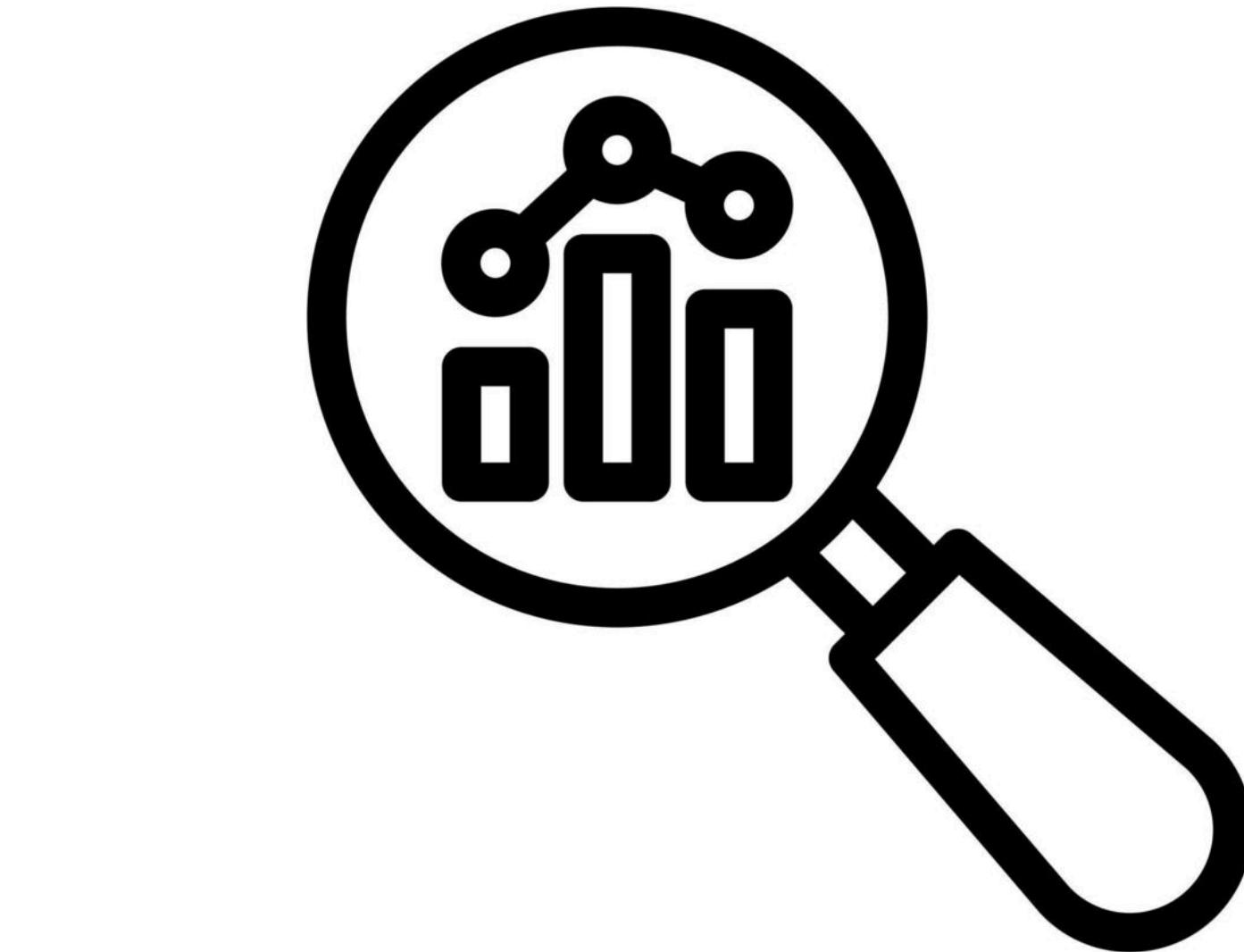
How we store (write) data heavily determines the performance of the system

The design goals

Building “efficient” data systems



build databases that can
write data **fast** ...



... and **process/analyze**
that data **quickly**

Well, just **get to work** then!

Performance tradeoff

The tug of war

hashmap

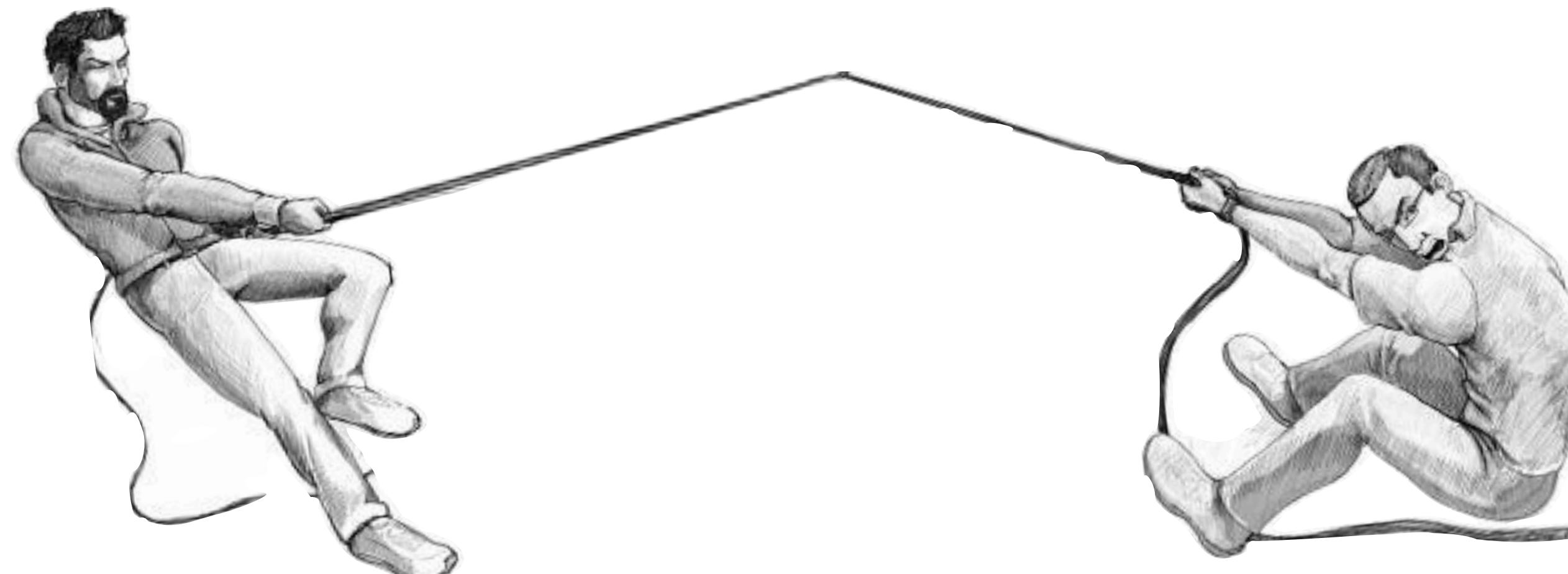
R: $\mathcal{O}(1)$

U: $\mathcal{O}(1)$

sorted
array

R: $\mathcal{O}(\log N)$

U: $\mathcal{O}(N)$



Query / **R**ead

Insert / **U**pdate

log

R: $\mathcal{O}(N)$

U: $\mathcal{O}(1)$

Performance tradeoff

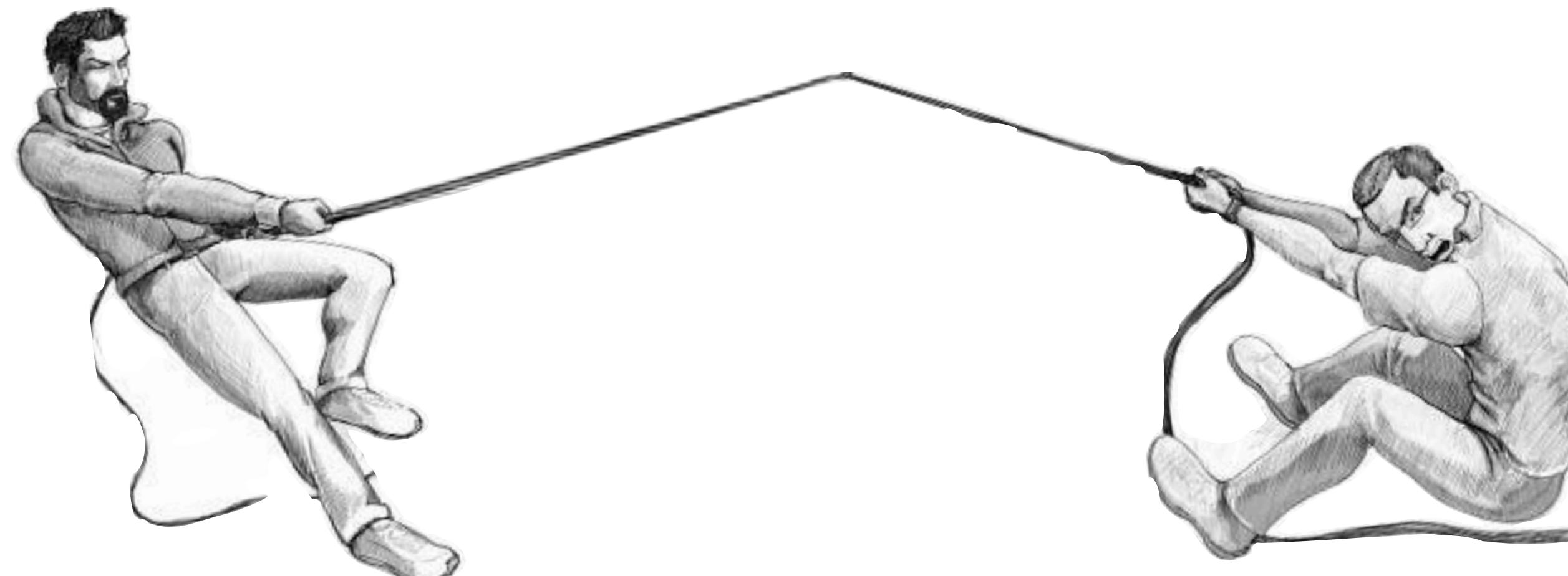
The tug of war

Storage / **M**emory

hashmap

R: $\mathcal{O}(1)$

U: $\mathcal{O}(1)$



sorted array

R: $\mathcal{O}(\log N)$

U: $\mathcal{O}(N)$

Query / **R**ead

Insert / **U**pdate

log

R: $\mathcal{O}(N)$

U: $\mathcal{O}(1)$



Performance tradeoff

The tug of war

Storage / **M**emory



hashmap

R: $\mathcal{O}(1)$

U: $\mathcal{O}(1)$



There is **NO** perfect data structure!

sorted array

R: $\mathcal{O}(\log N)$

U: $\mathcal{O}(N)$



Query / **R**ead



Insert / **U**pdate

log

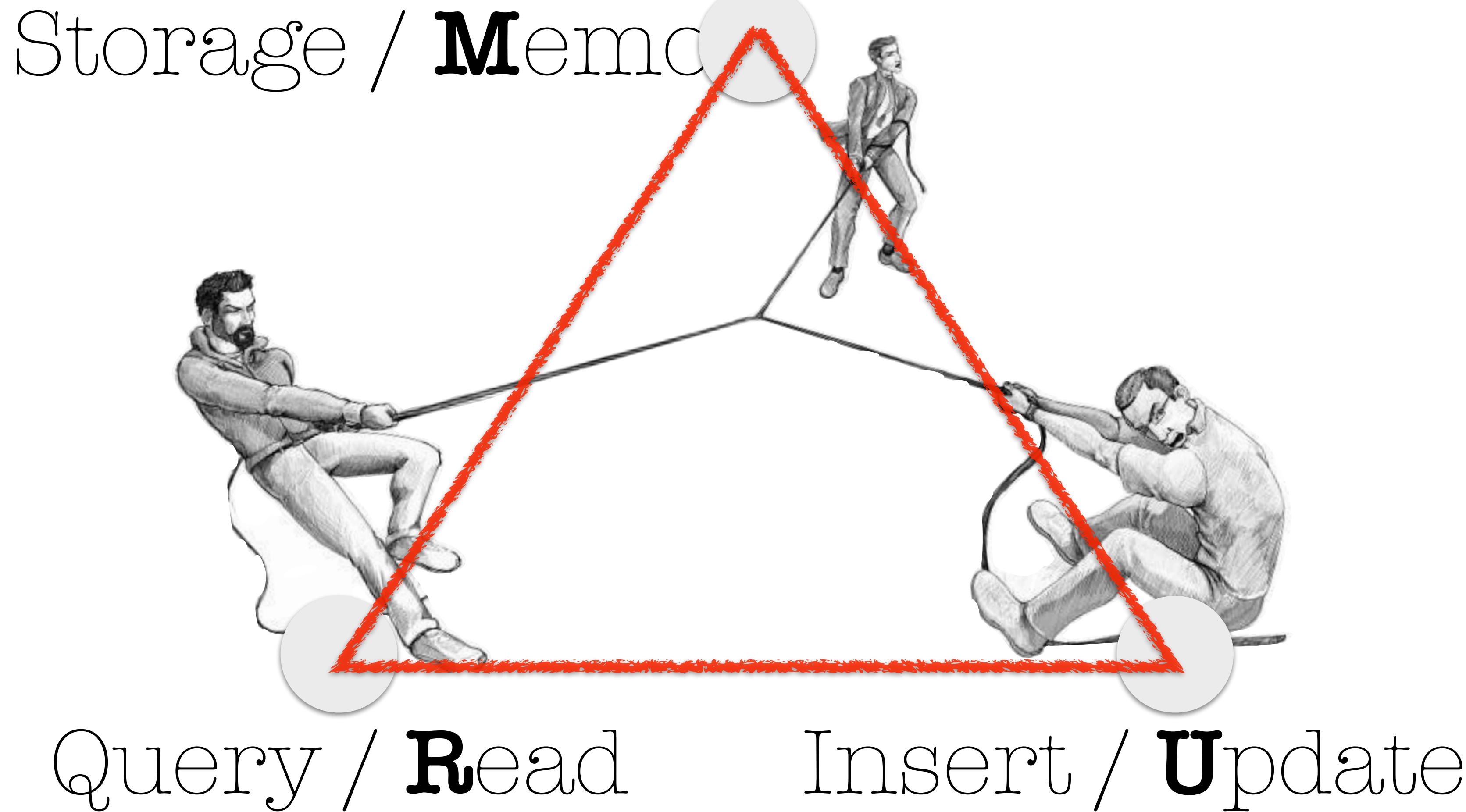
R: $\mathcal{O}(N)$

U: $\mathcal{O}(1)$



RUM conjecture

A three-way tradeoff

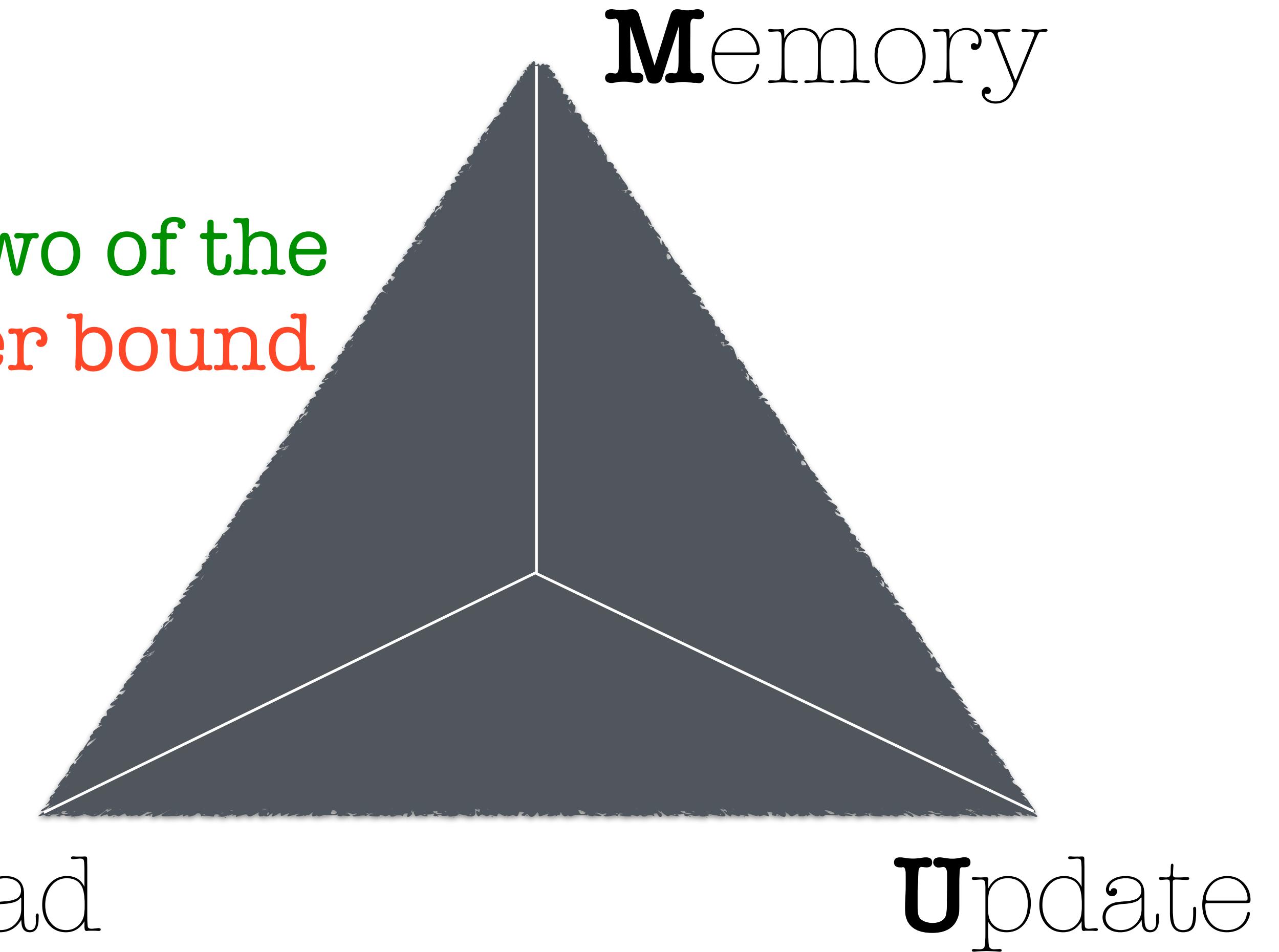


RUM conjecture

A three-way tradeoff

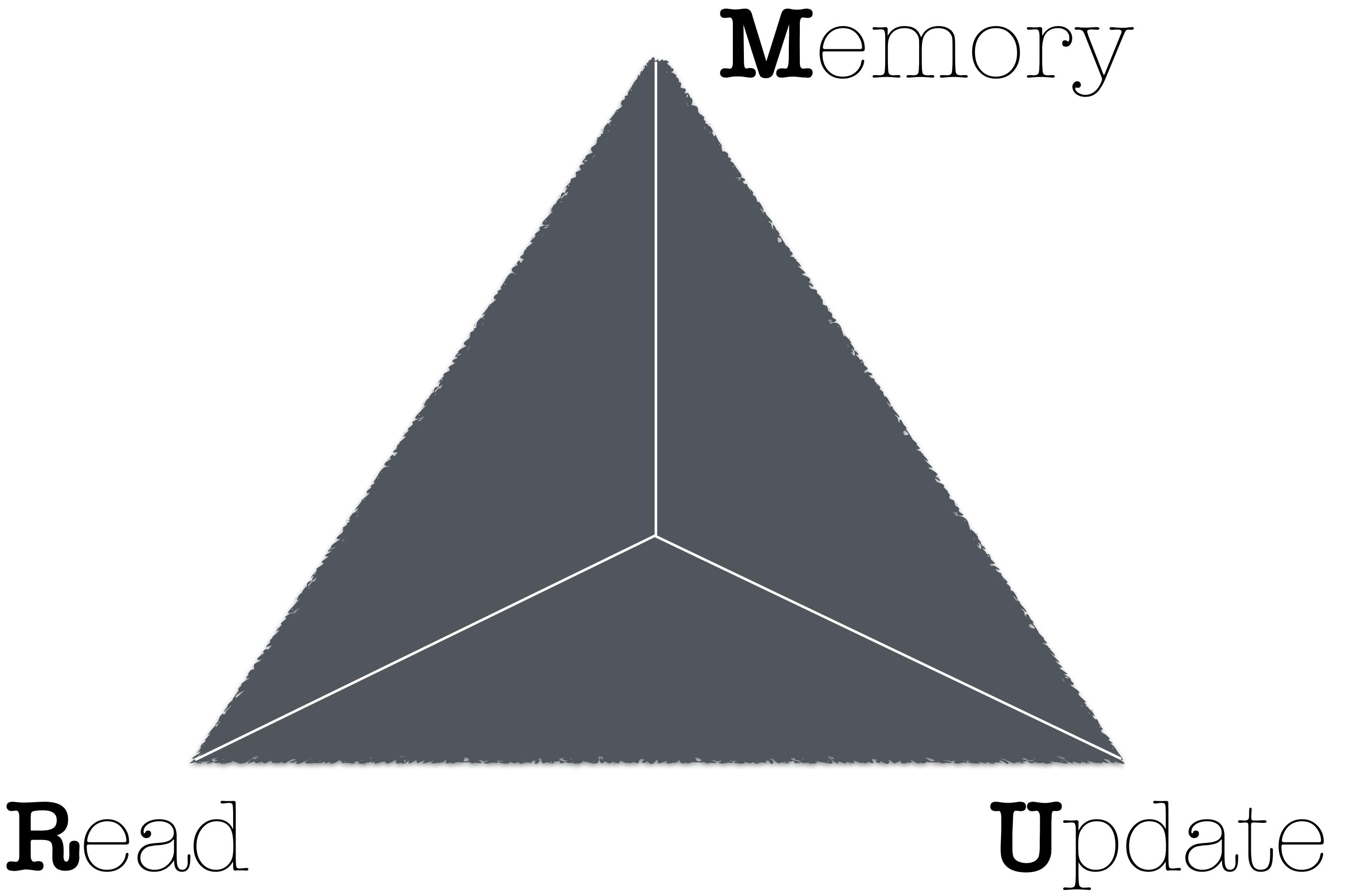
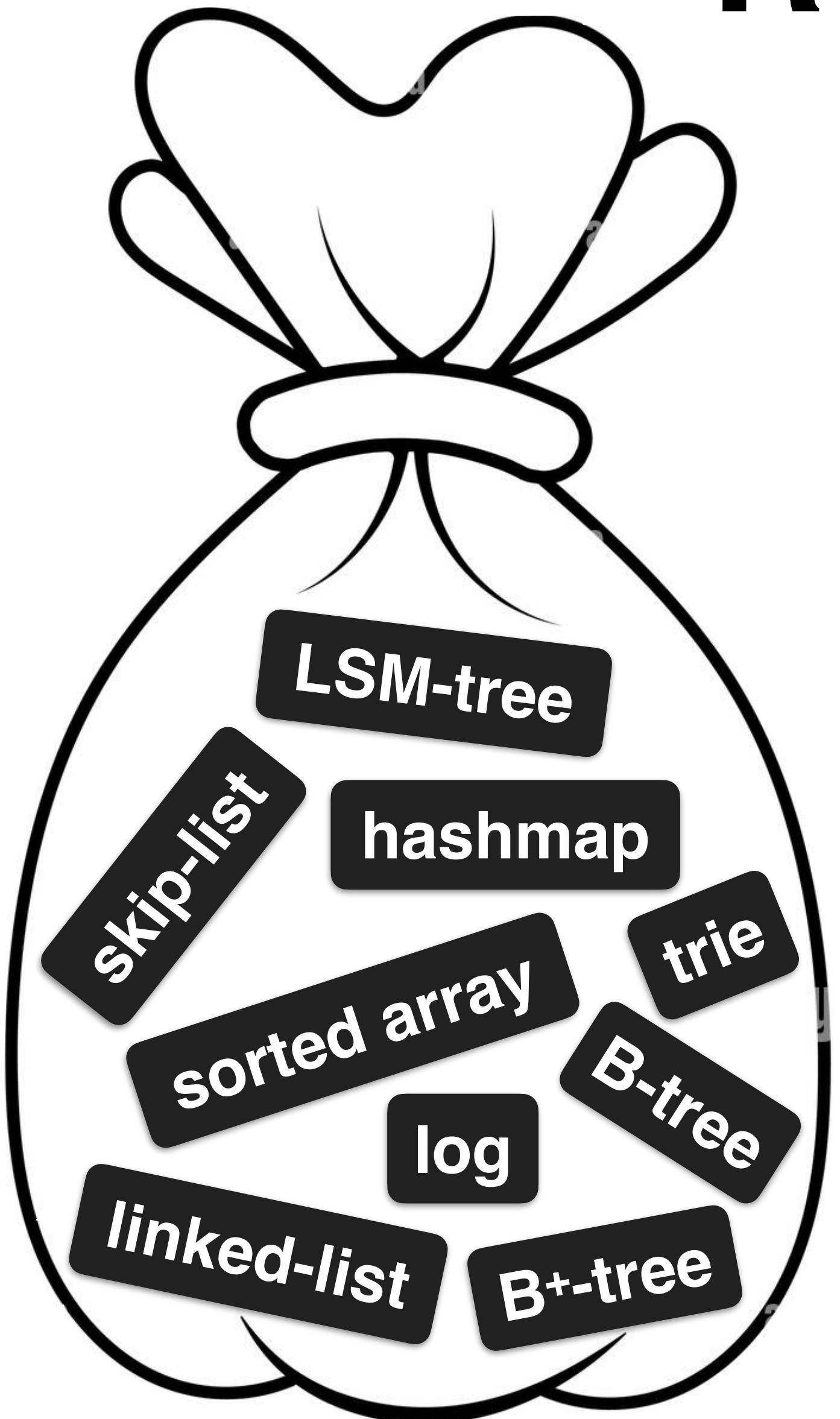
66

... setting an **upper bound** for **two of the RUM axes**, implies a **hard lower bound** for the **third axis**



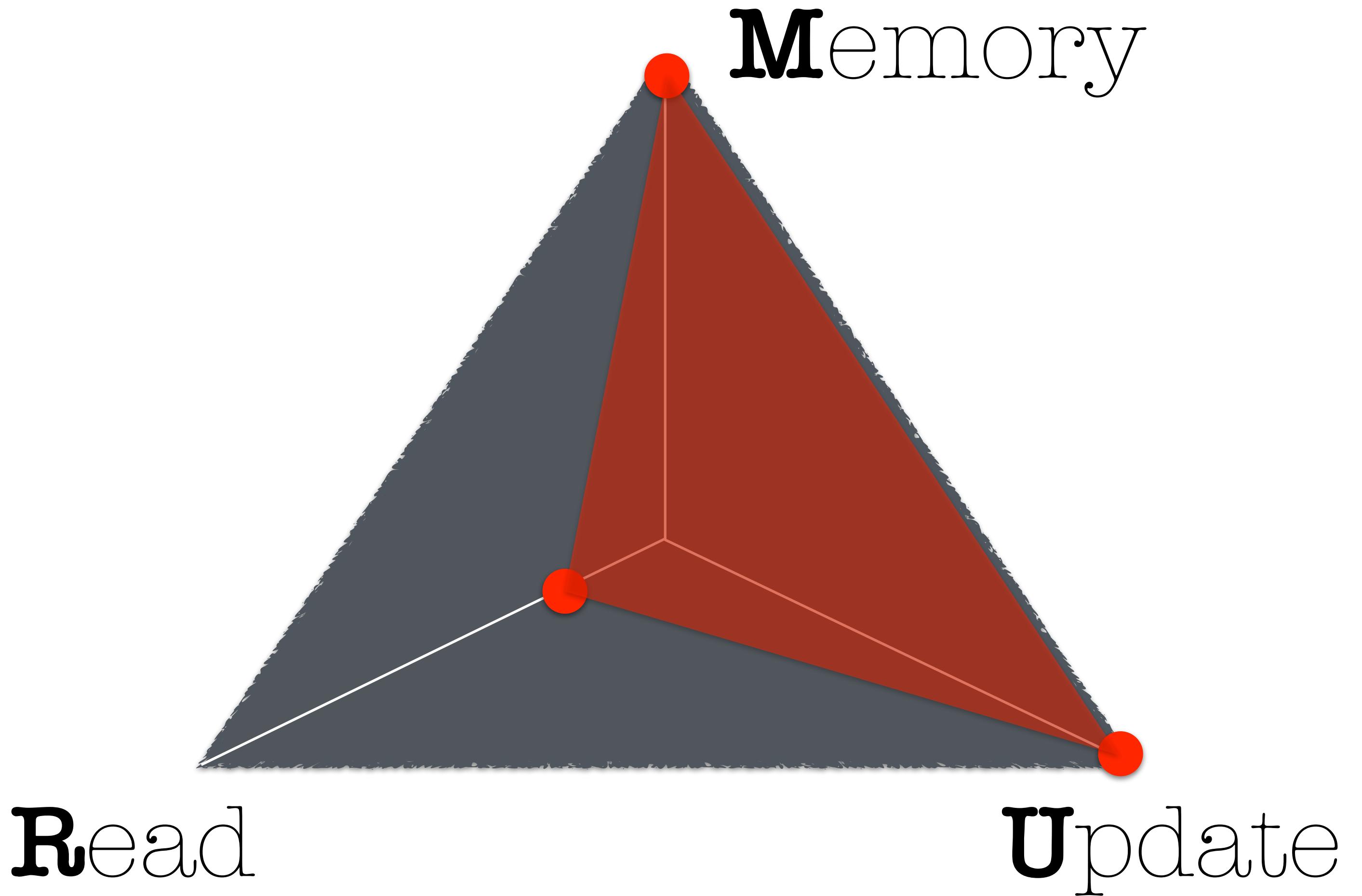
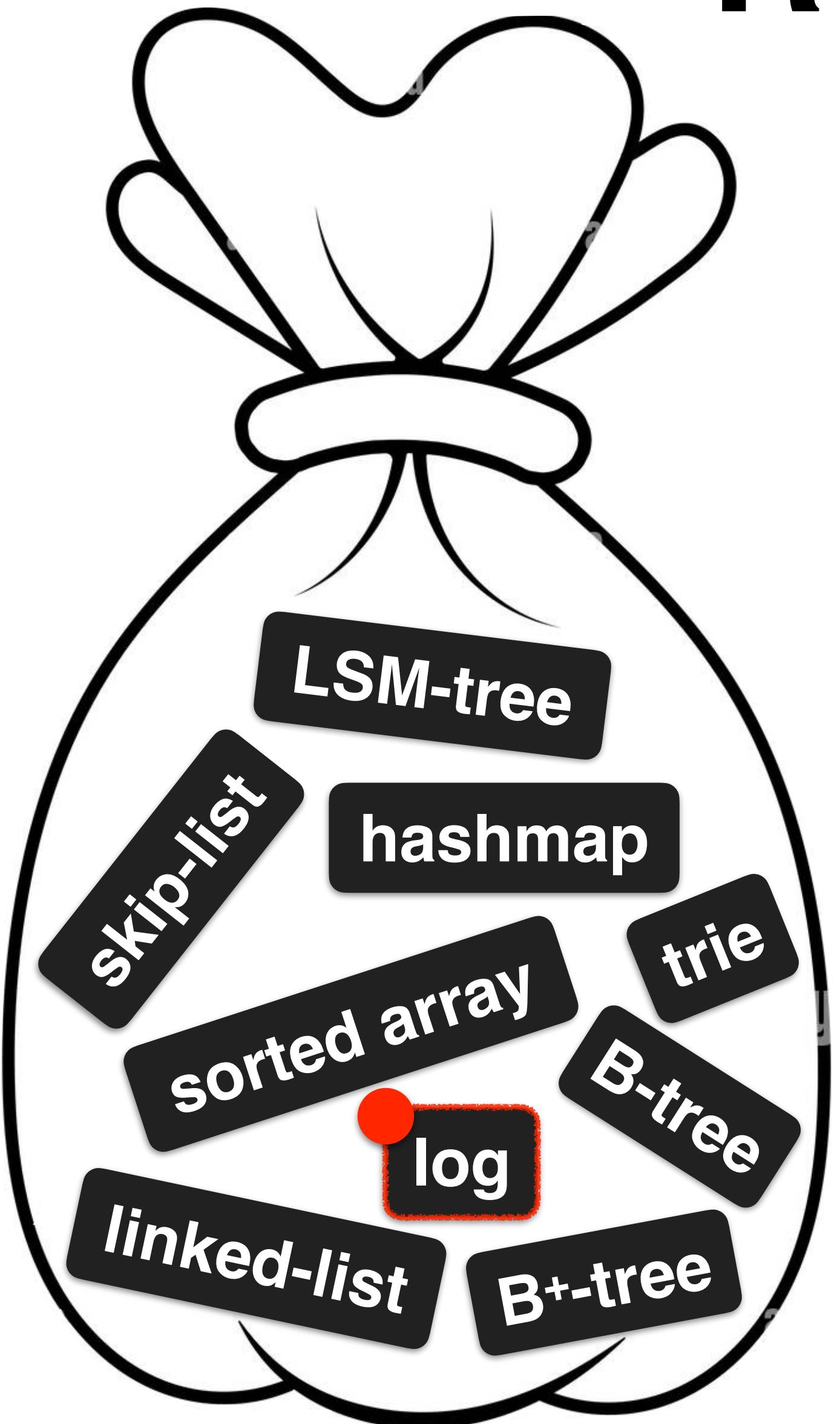
RUM conjecture

A three-way tradeoff



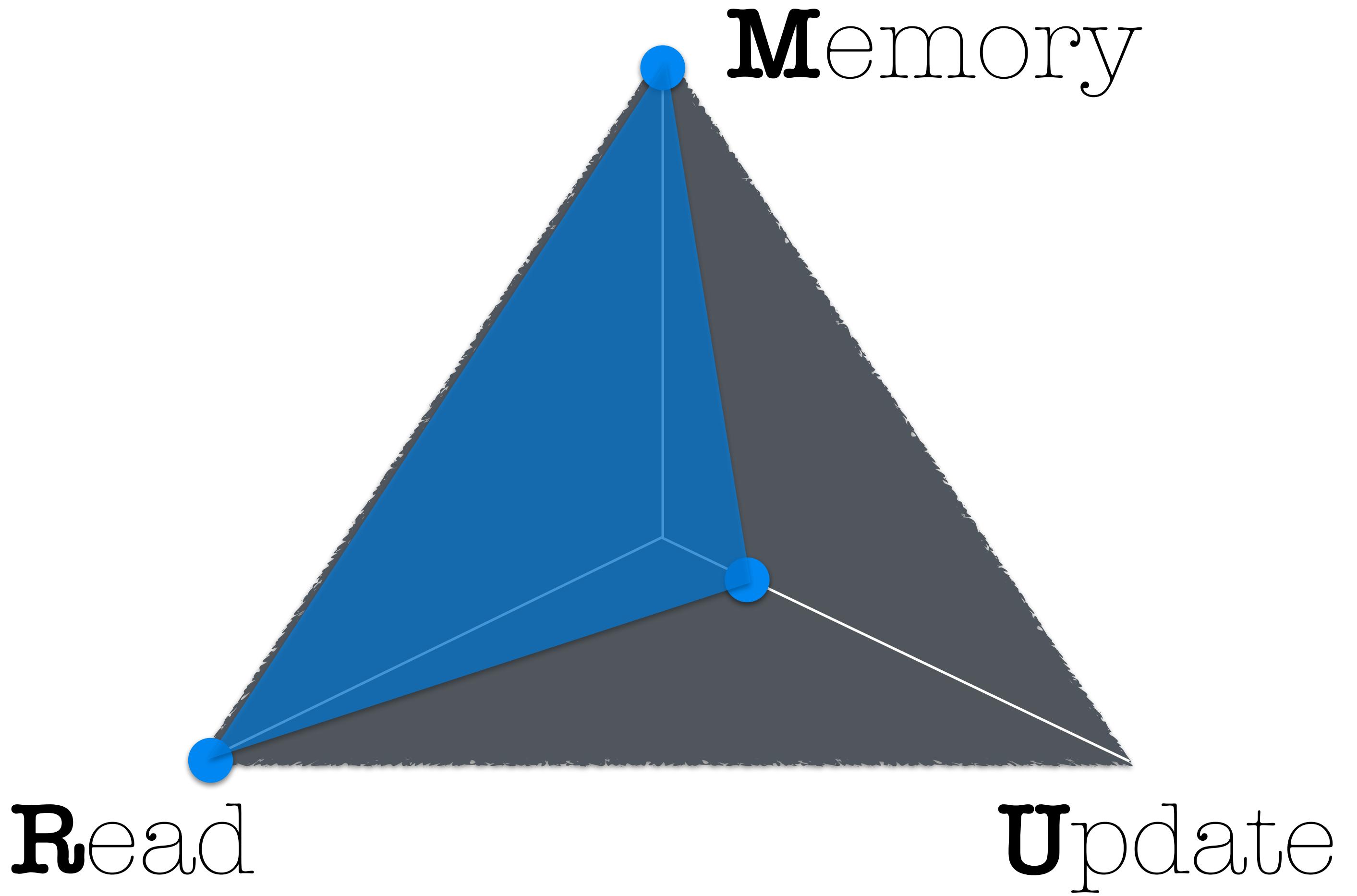
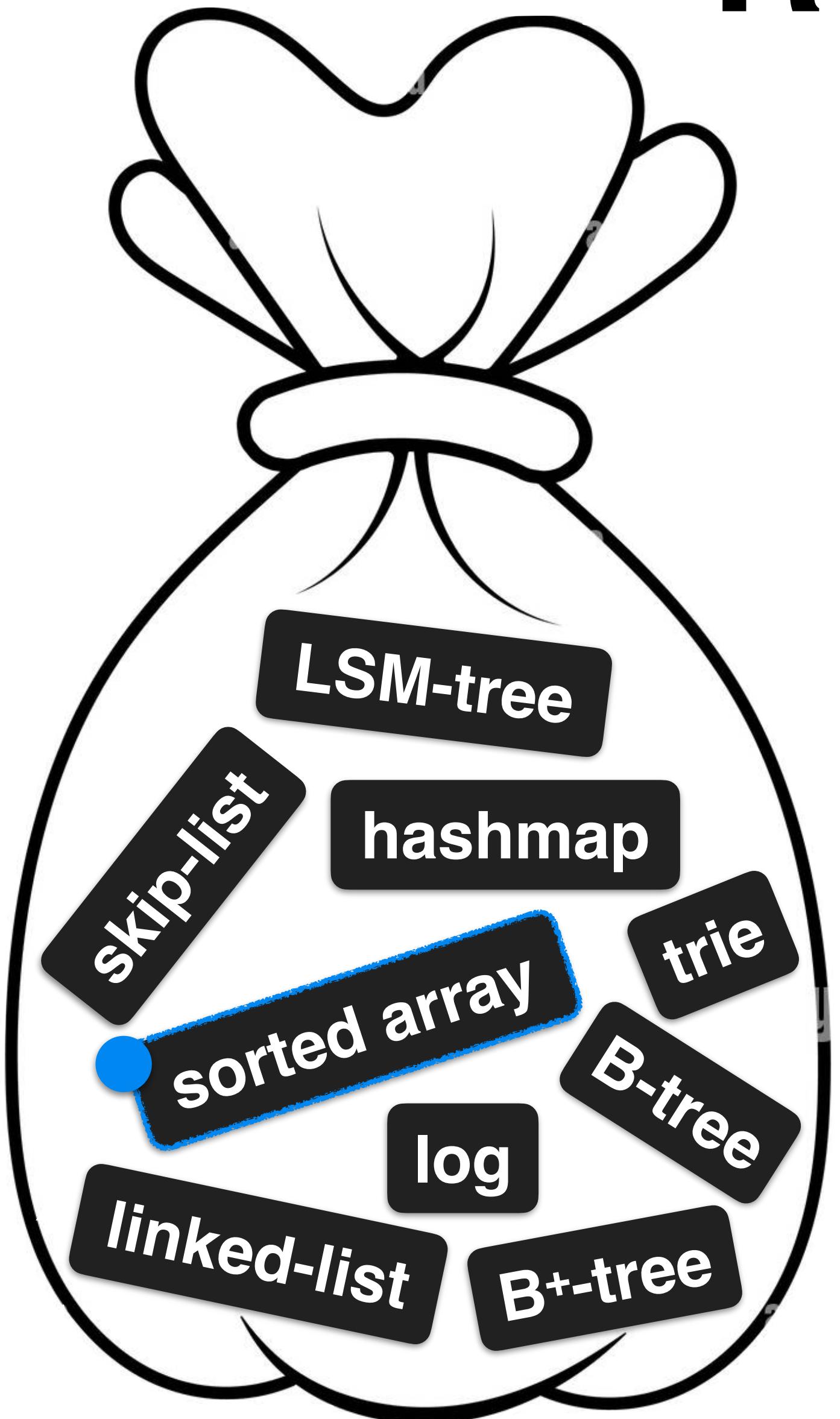
RUM conjecture

A three-way tradeoff



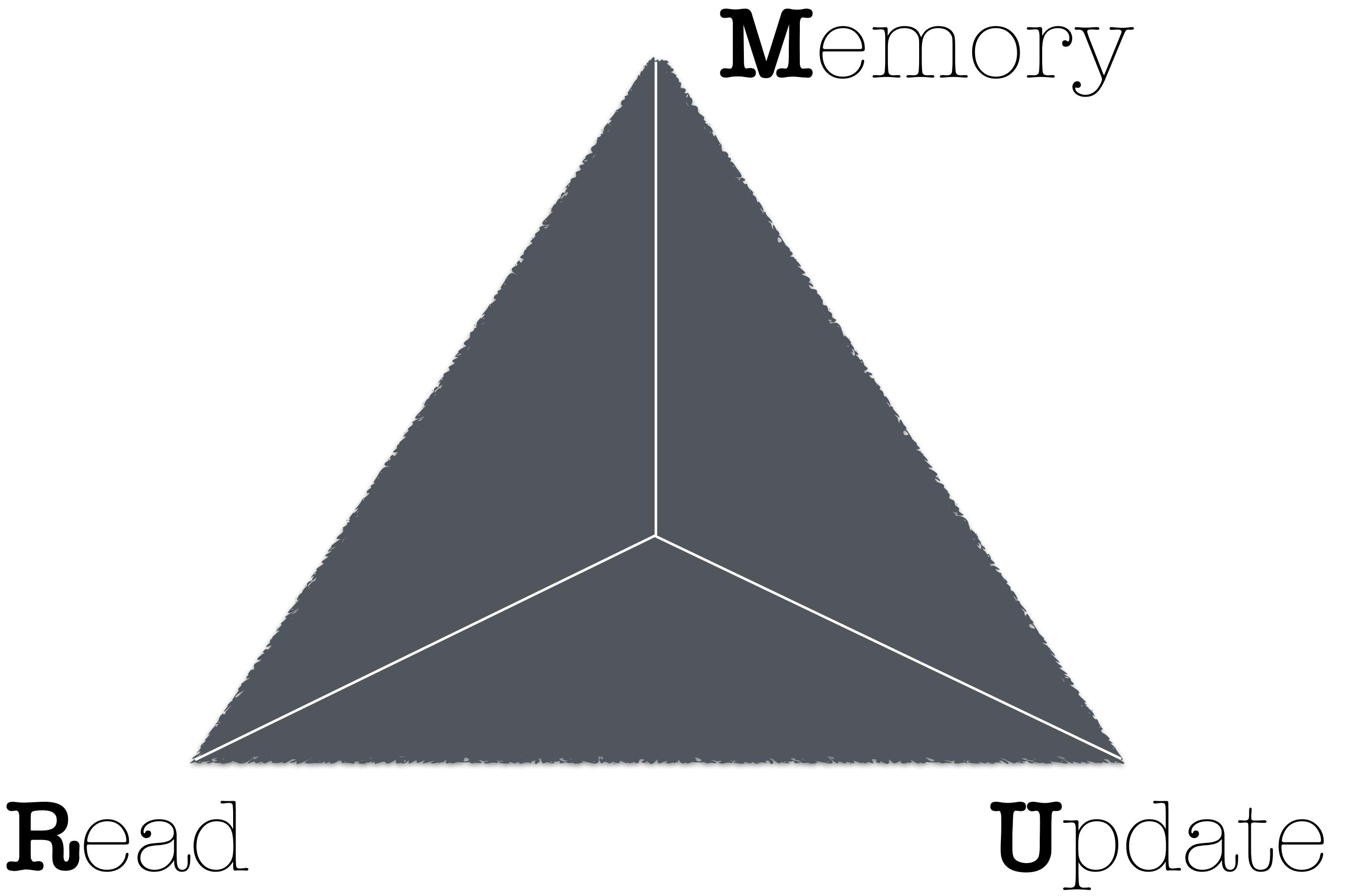
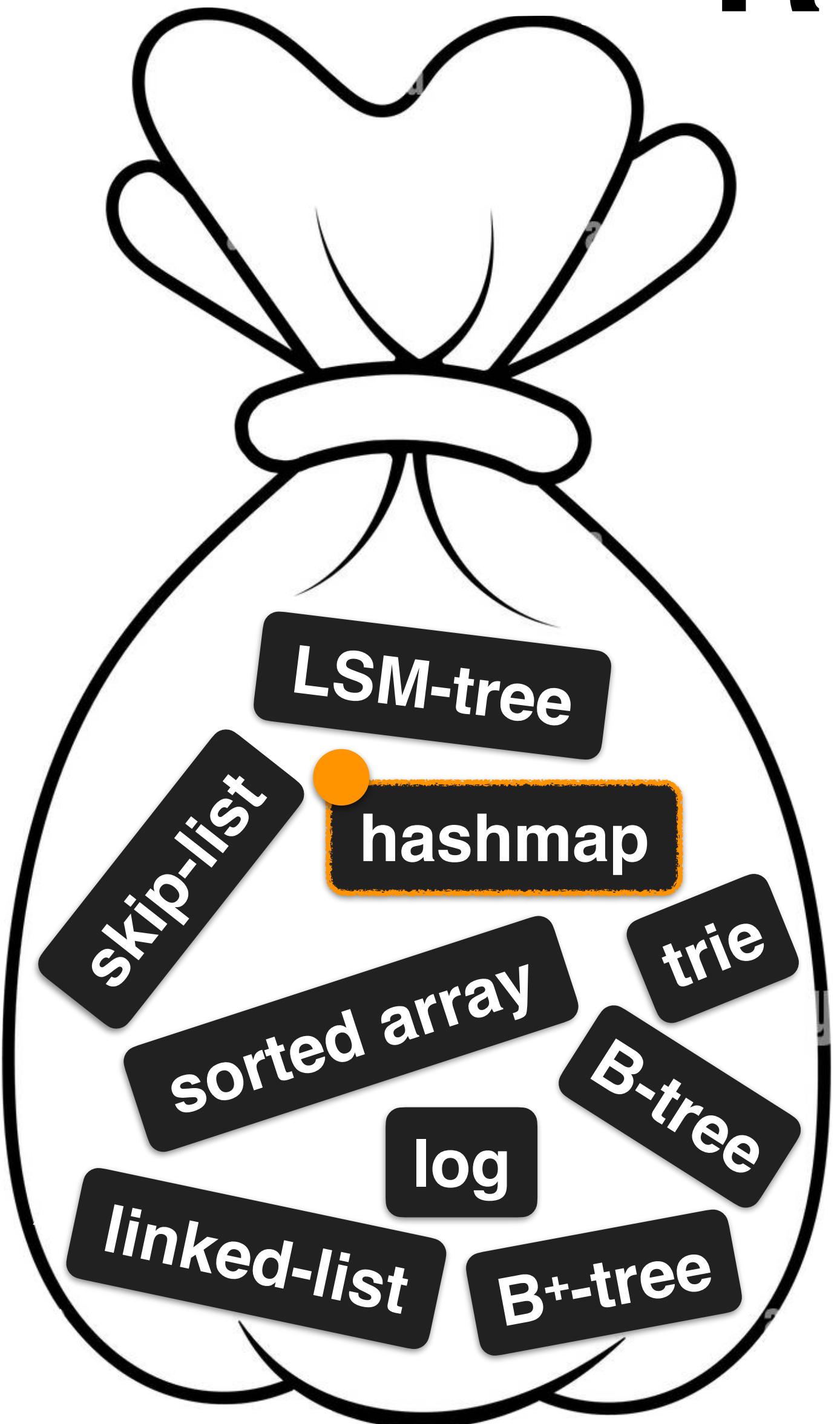
RUM conjecture

A three-way tradeoff



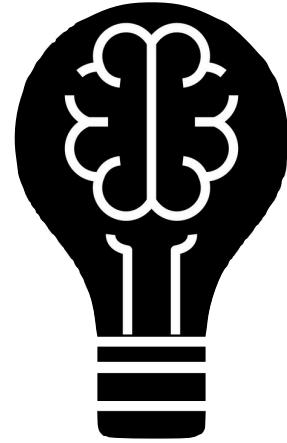
RUM conjecture

A three-way tradeoff



RUM conjecture

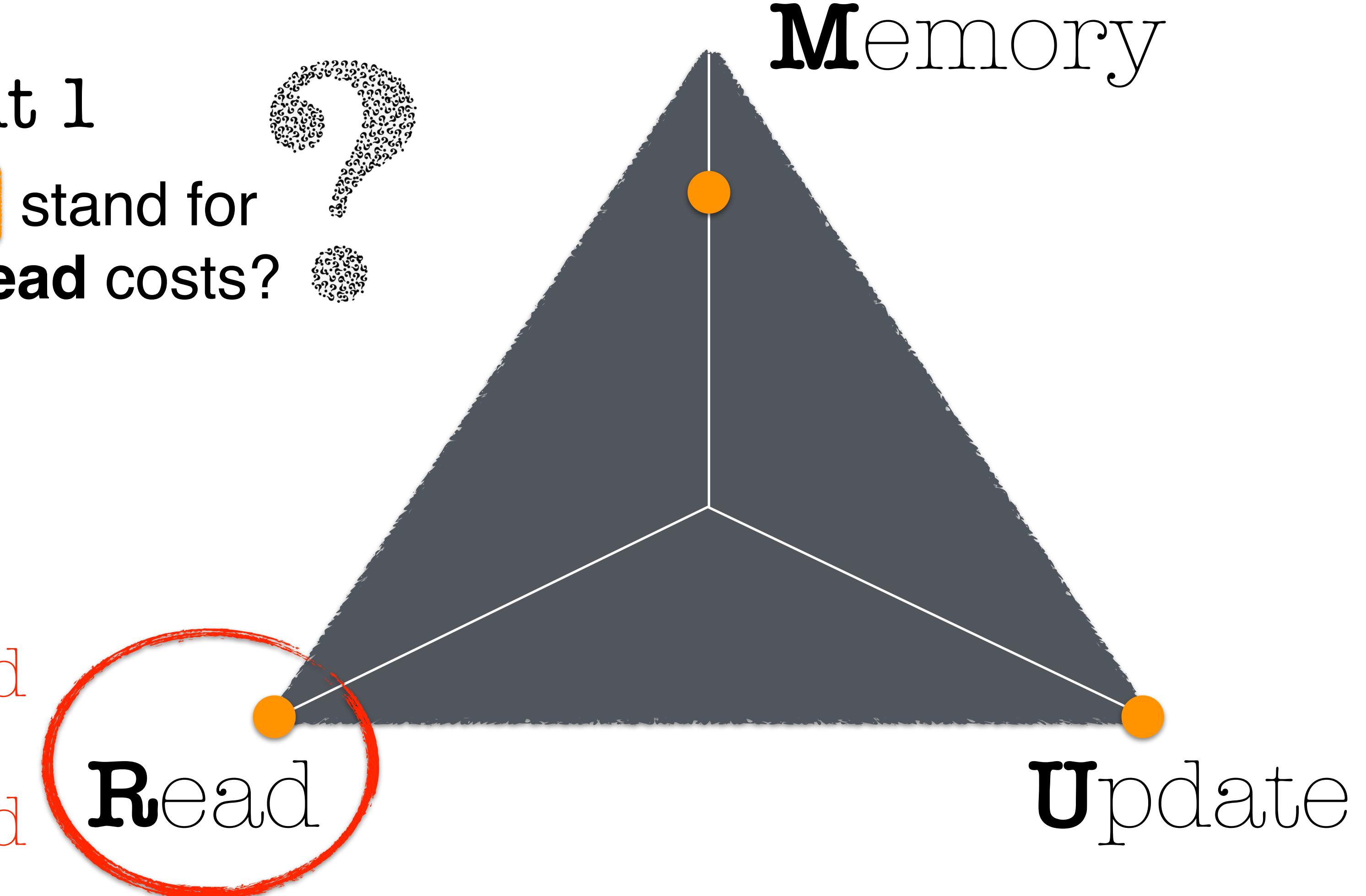
A three-way tradeoff



Thought Experiment 1

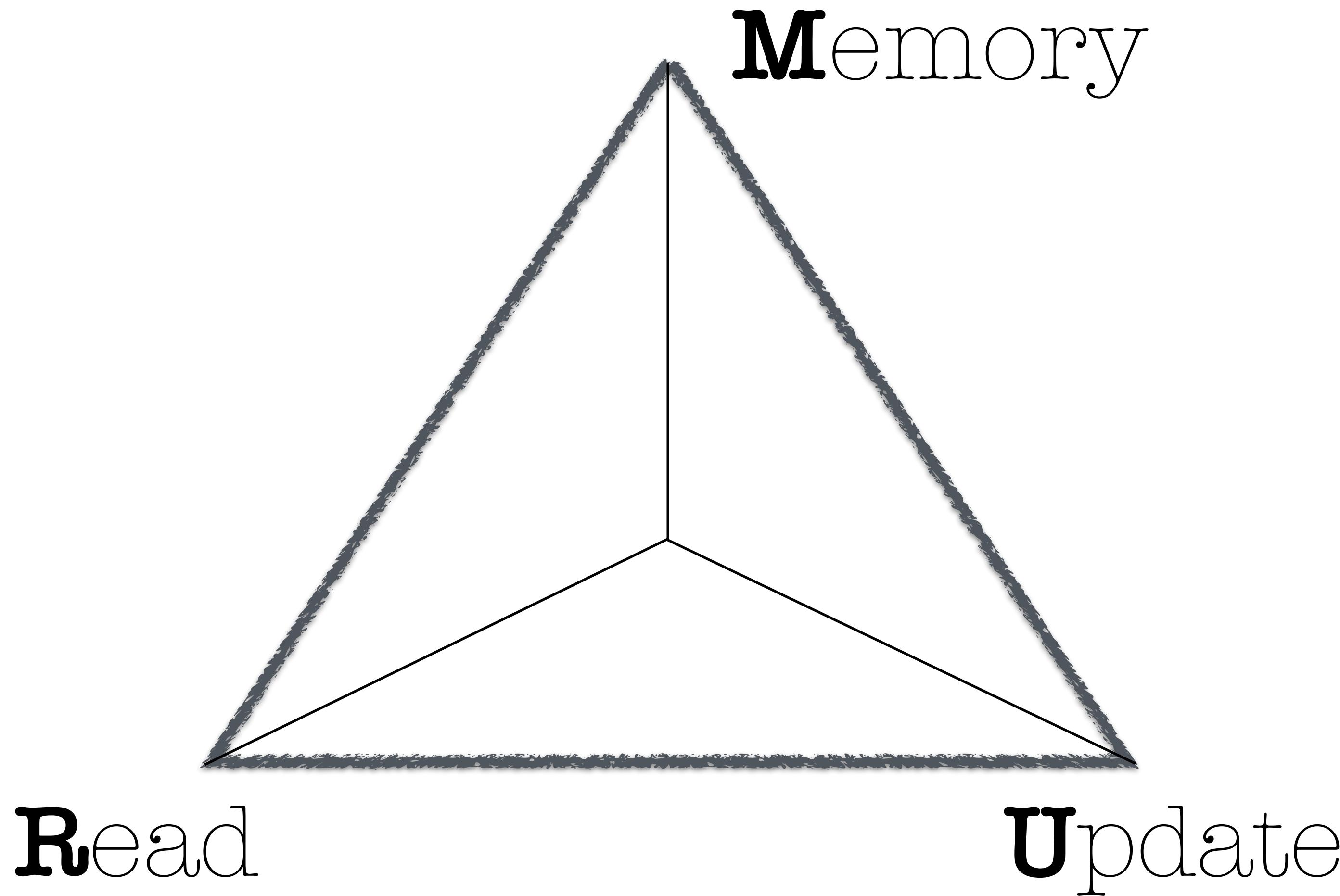
Where does a **hashmap** stand for
memory, **update**, and **read** costs?

point read
vs
range read



RUM conjecture

A three-way tradeoff

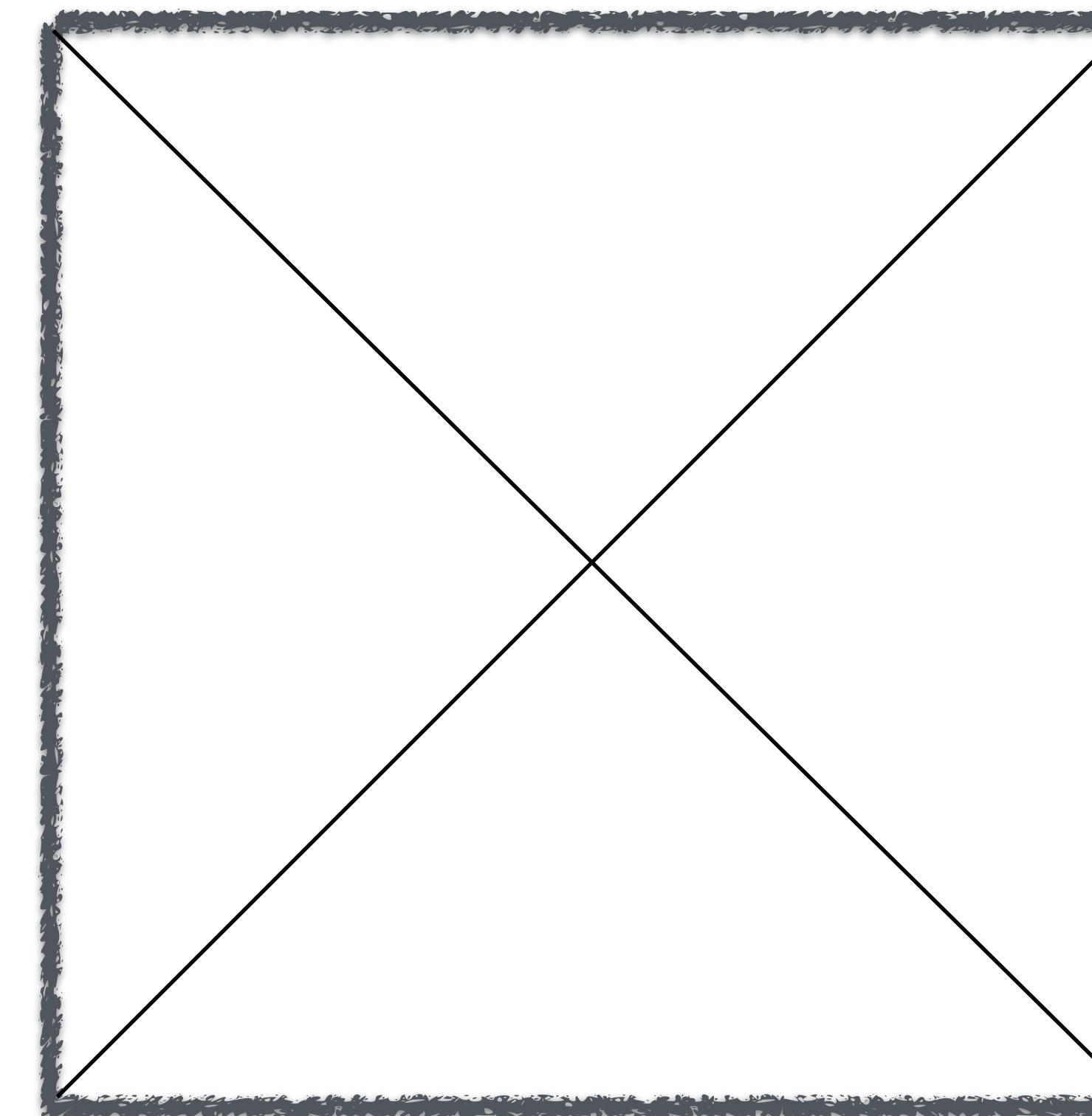


Extending the **RUM** tradeoff

A multi-way tradeoff

Point read

Memory



Range read

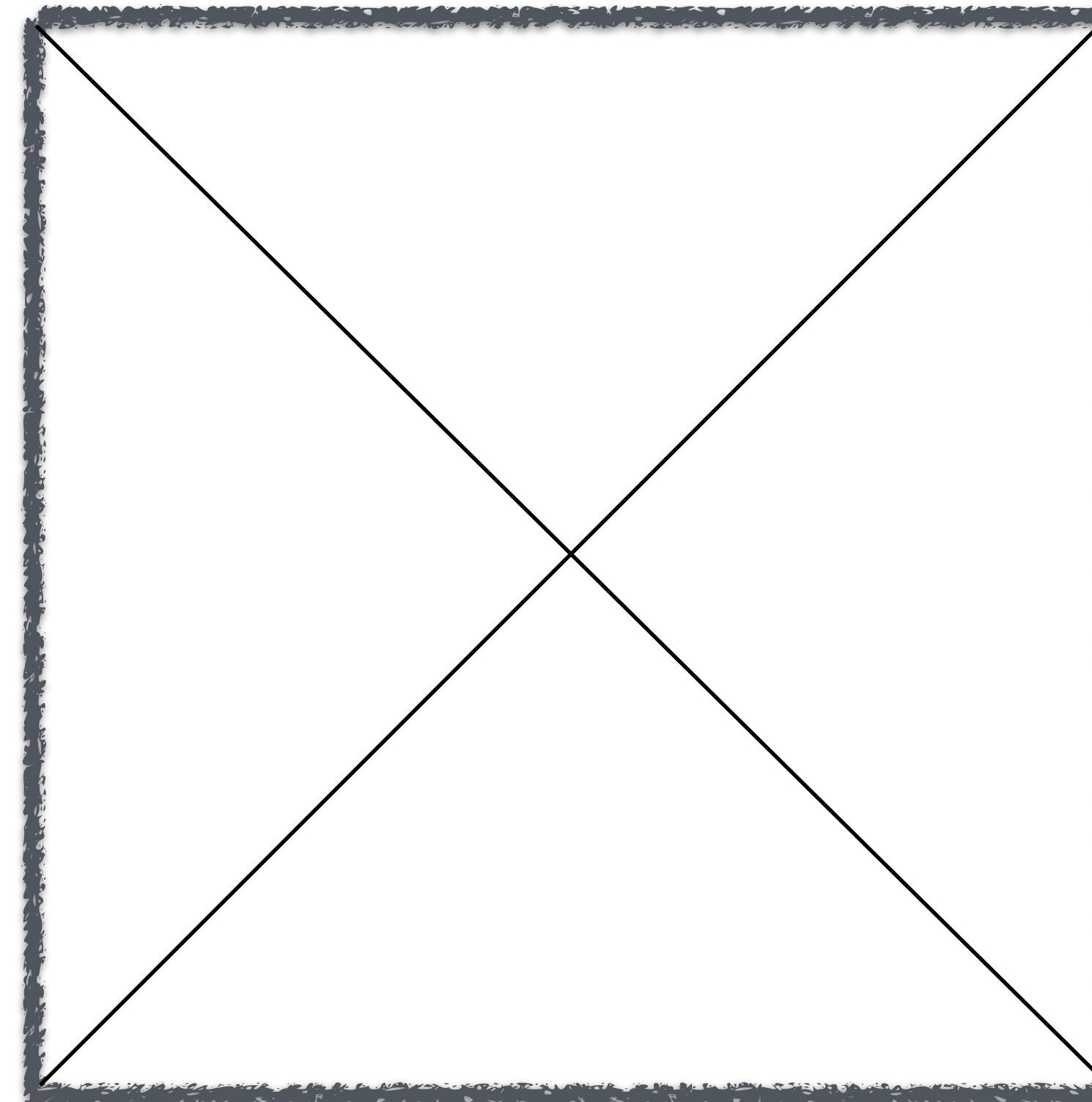
Update

Extending the RUM tradeoff

A multi-way tradeoff

Point read

Memory



Range read

Update

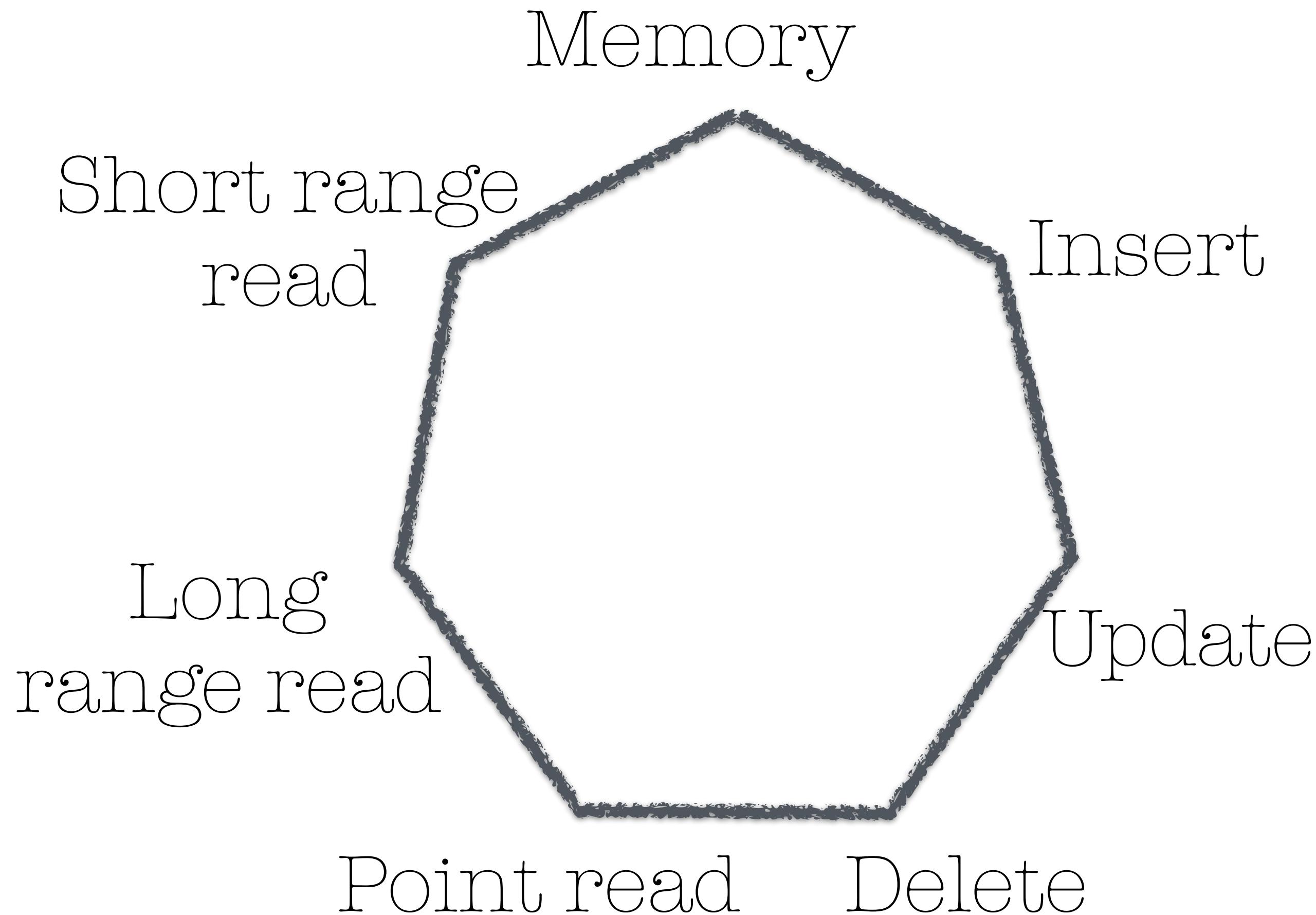
But, what about **deletes**?

Are **updates** the same as **inserts**?

What about **selectivity** of range reads?

Extending the RUM tradeoff

A multi-way tradeoff



But, what about **deletes**?

Are **updates** the same as **inserts**?

What about **selectivity** of range reads?

hardware

cloud
cost

performance
tradeoffs

Designing data systems = **HARD PROBLEM**

index
design

access method

application
requirements

Designing data systems

Solving a hard problem



Ask for the **HiPPO**

Highest Paid Person's Opinion

Designing data systems

Solving a hard problem



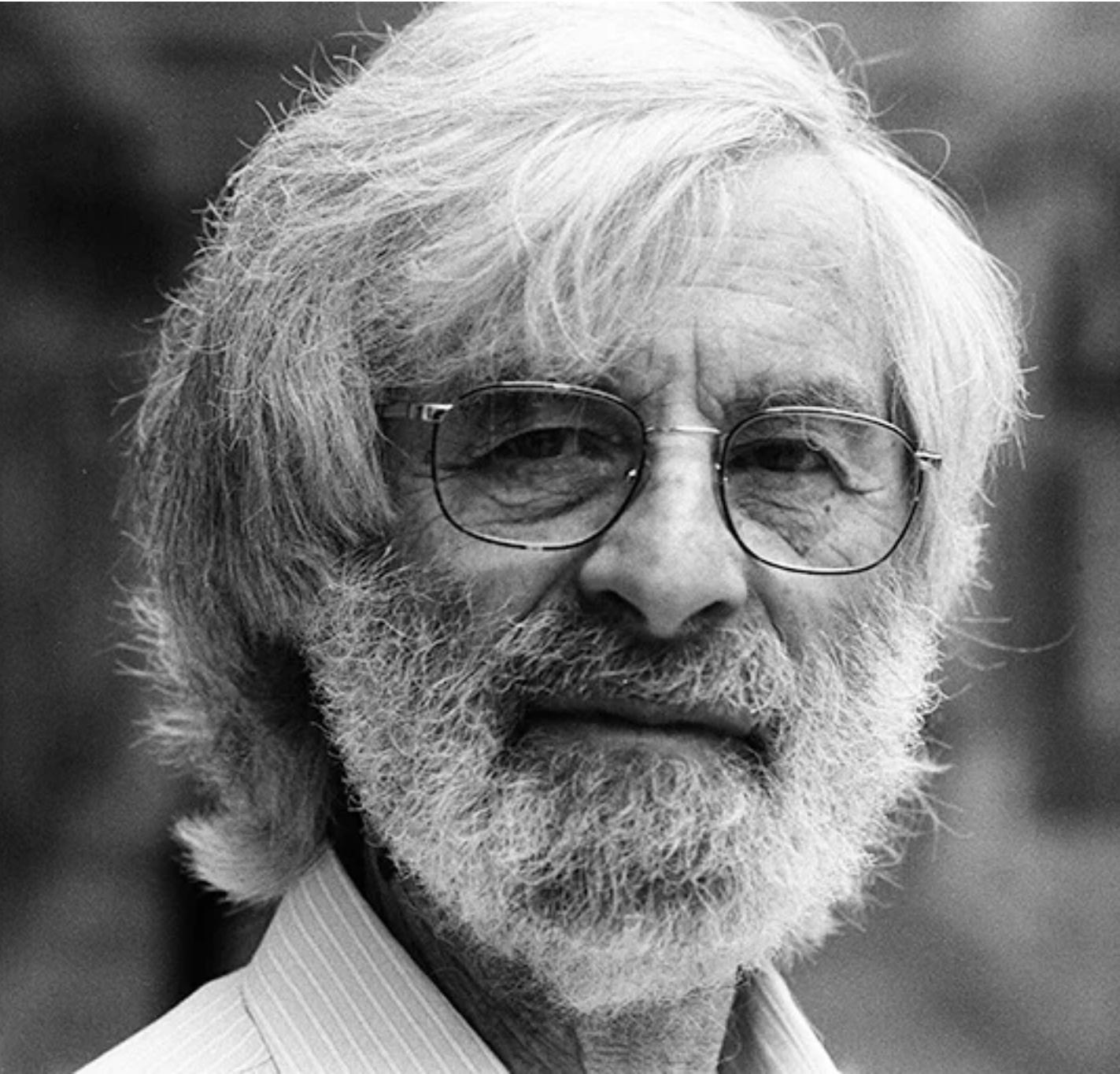
No one knows everything!

Ask for the HiPPO

Highest Paid Person's Opinion

Designing data systems

Solving a hard problem



Lesley Lamport, Microsoft Research
MA '63, PhD '72 Brandeis University
ACM Turing Award 2013

No one knows everything!

The Part-Time Parliament

LESLIE
Digital E

Paxos Made Simple

Leslie Lamport

01 Nov 2001

Abstract

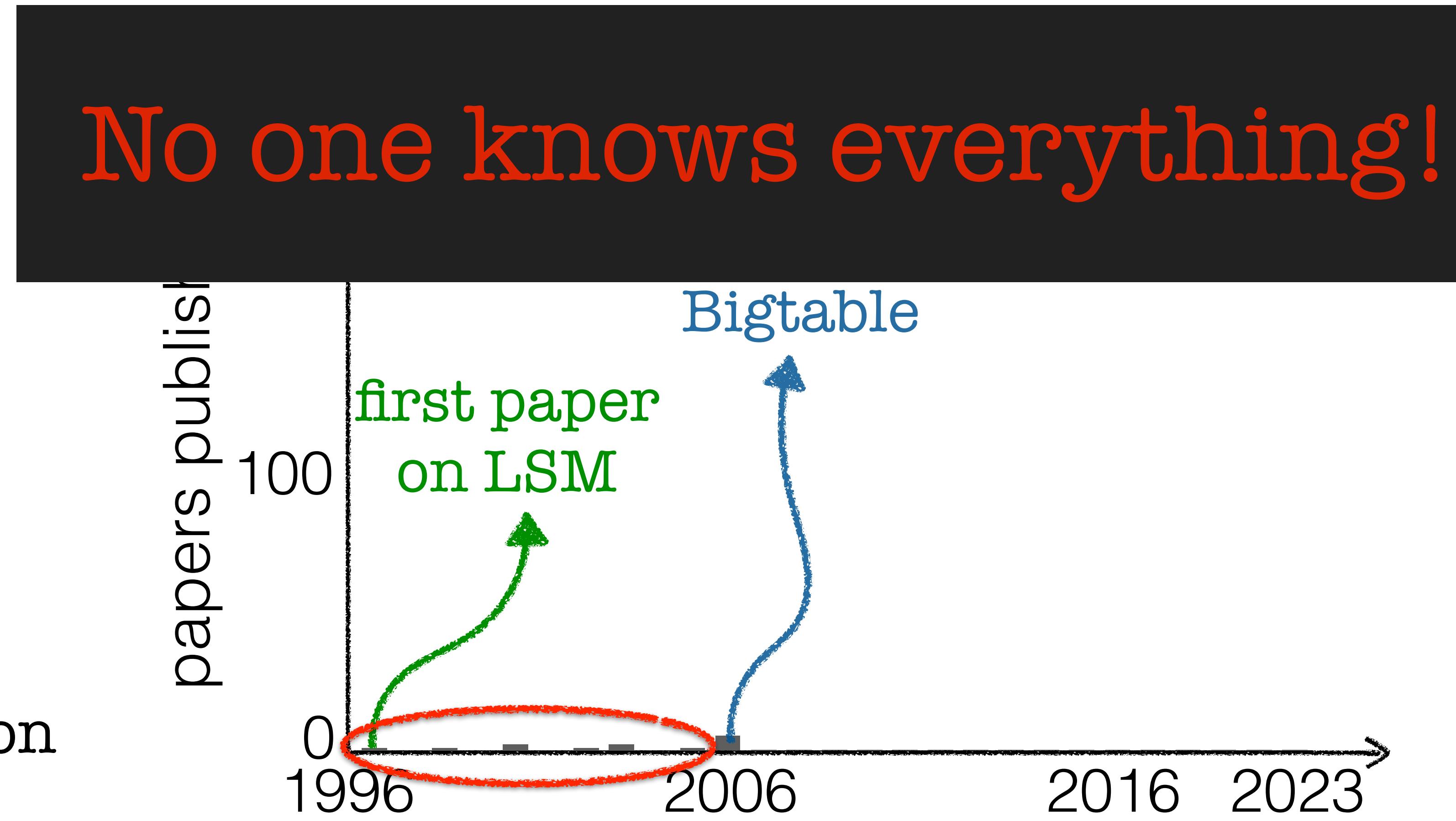
The Paxos algorithm, when presented in plain English, is very simple.

Designing data systems

Solving a hard problem

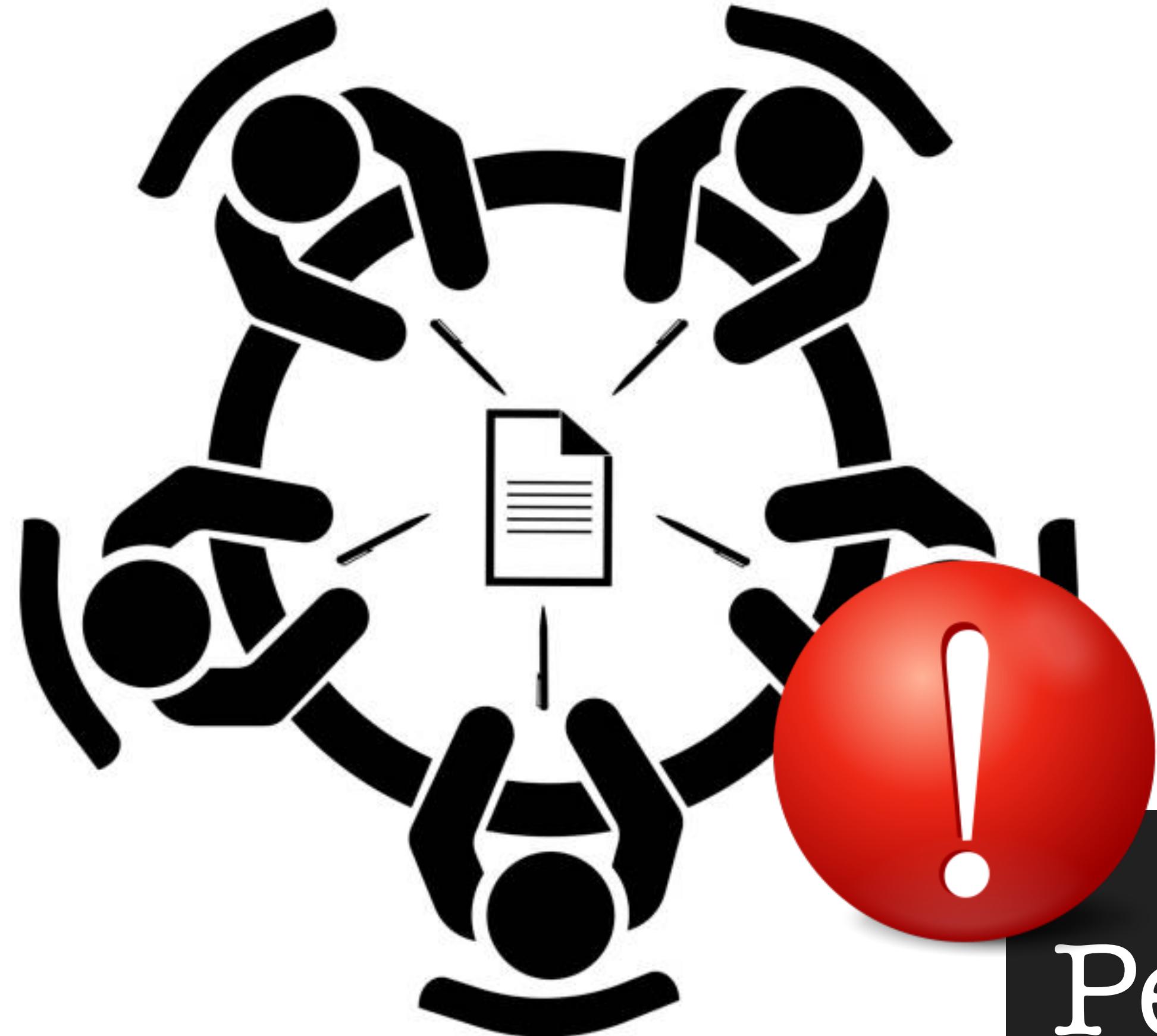


Patrick O'Neil, UMass Boston
Co-inventor of the LSM-tree

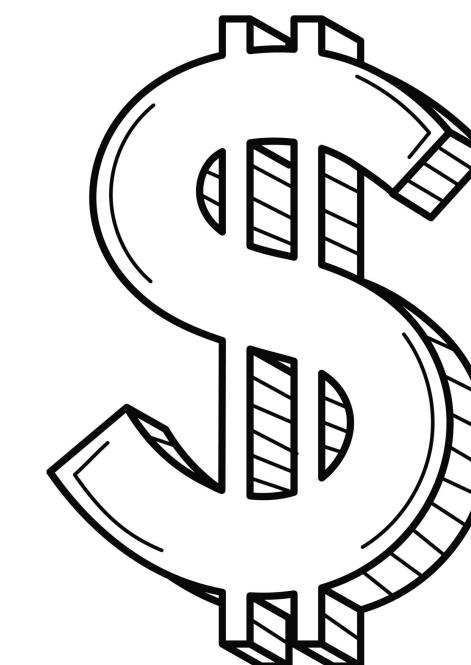
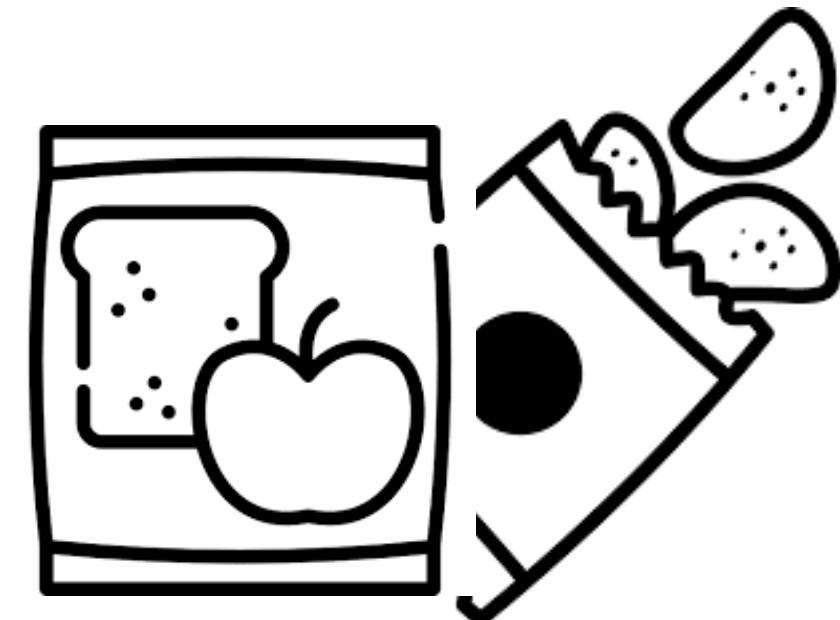


Designing data systems

Solving a hard problem



Get the COSI 167 students



Perfectly tuned data system!

Designing data systems

Solving a hard problem

Manual designing &
hand-tuning hundreds of knobs
do NOT SCALE

Designing data systems

Solving a hard problem

Manual designing &
hand-tuning hundreds of knobs
do NOT SCALE

Self-designing, self-tuning, and
adaptive data systems

Designing data systems

Solving a hard problem

Self-designing, self-tuning, and adaptive data systems

Adaptive data layouts: row stores vs. column stores vs. hybrids

Self-tuning LSM-engines

Adaptive (adaptive) indexing

Workload-aware data re-organization

Self-designing storage engines

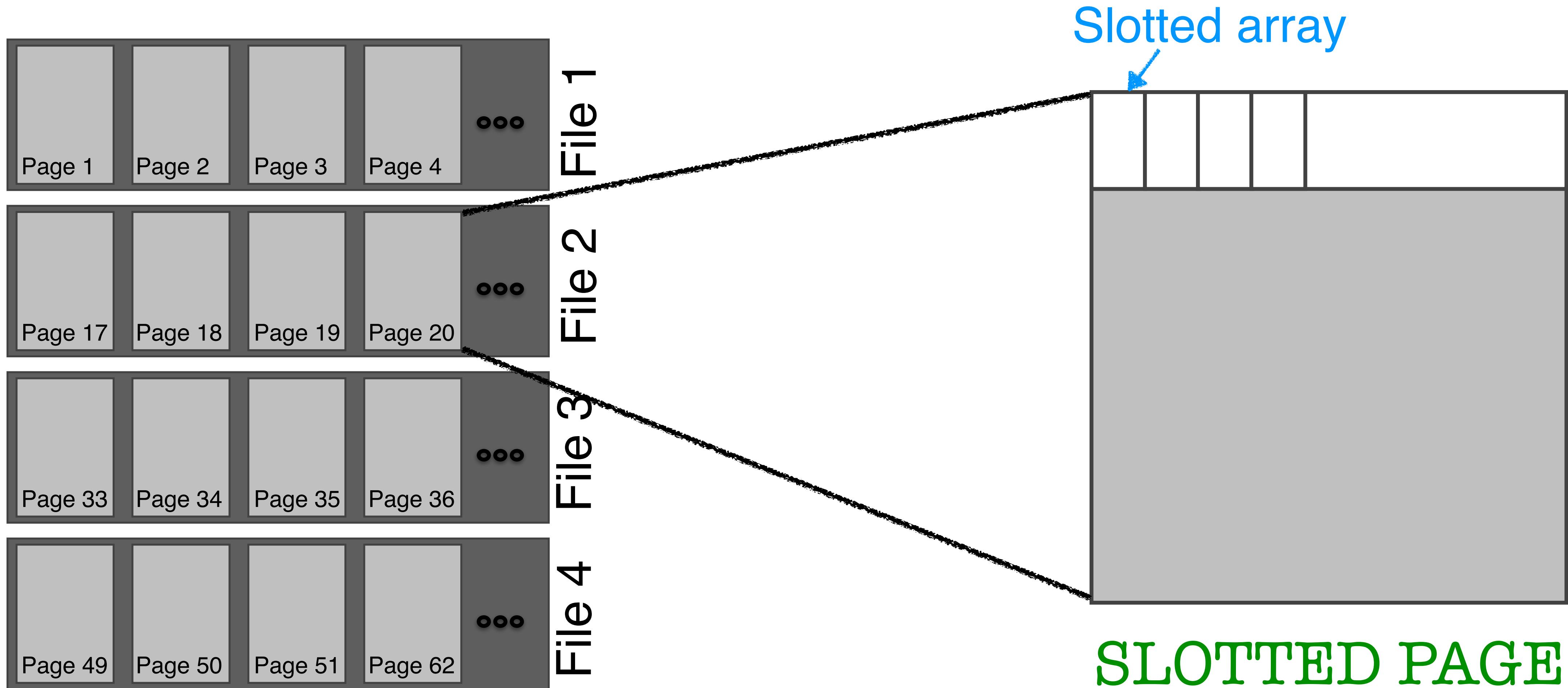
Hardware-conscious memory manager

Cloud cost-optimized data systems

Understanding data placement

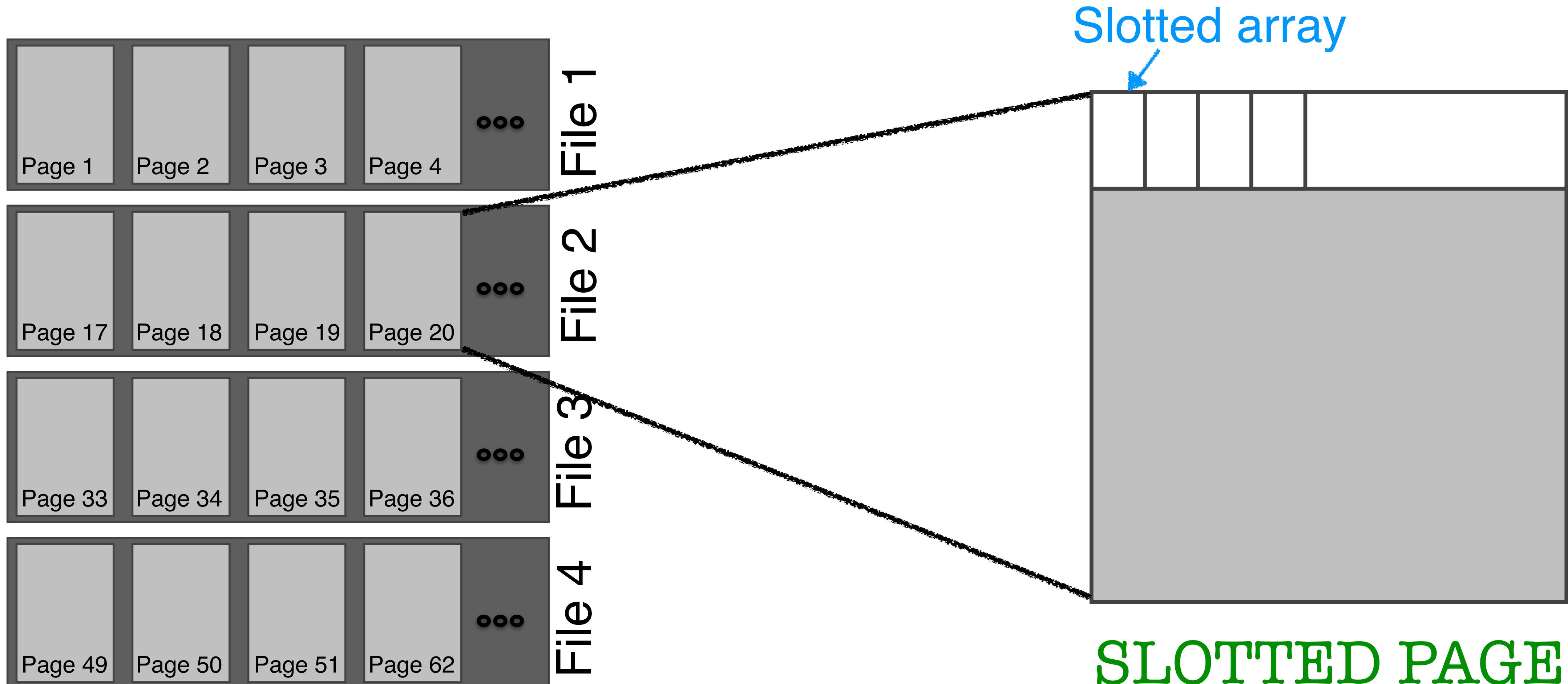
Understanding data placement

Files, pages, tuples



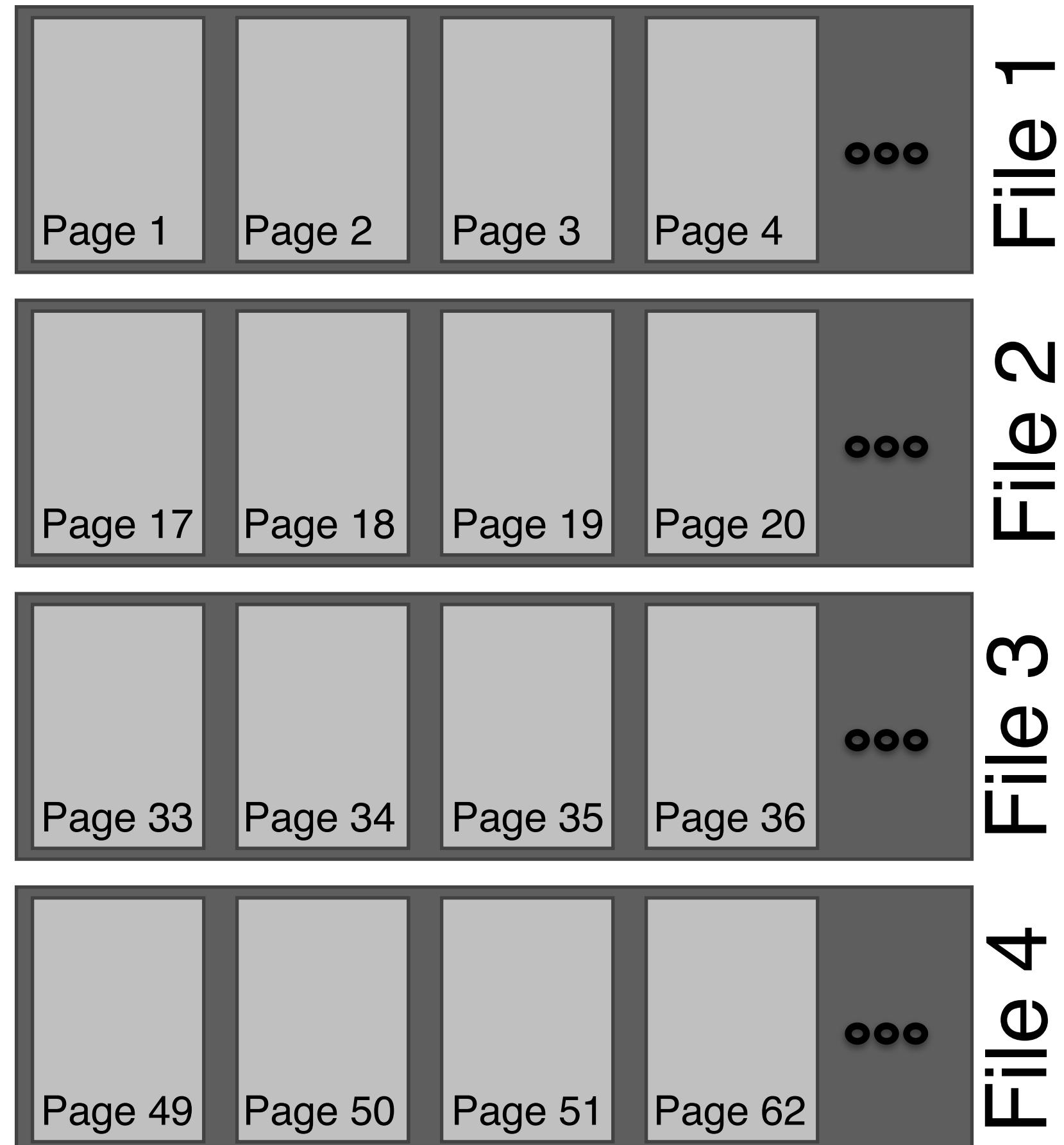
Understanding data placement

Files, pages, tuples

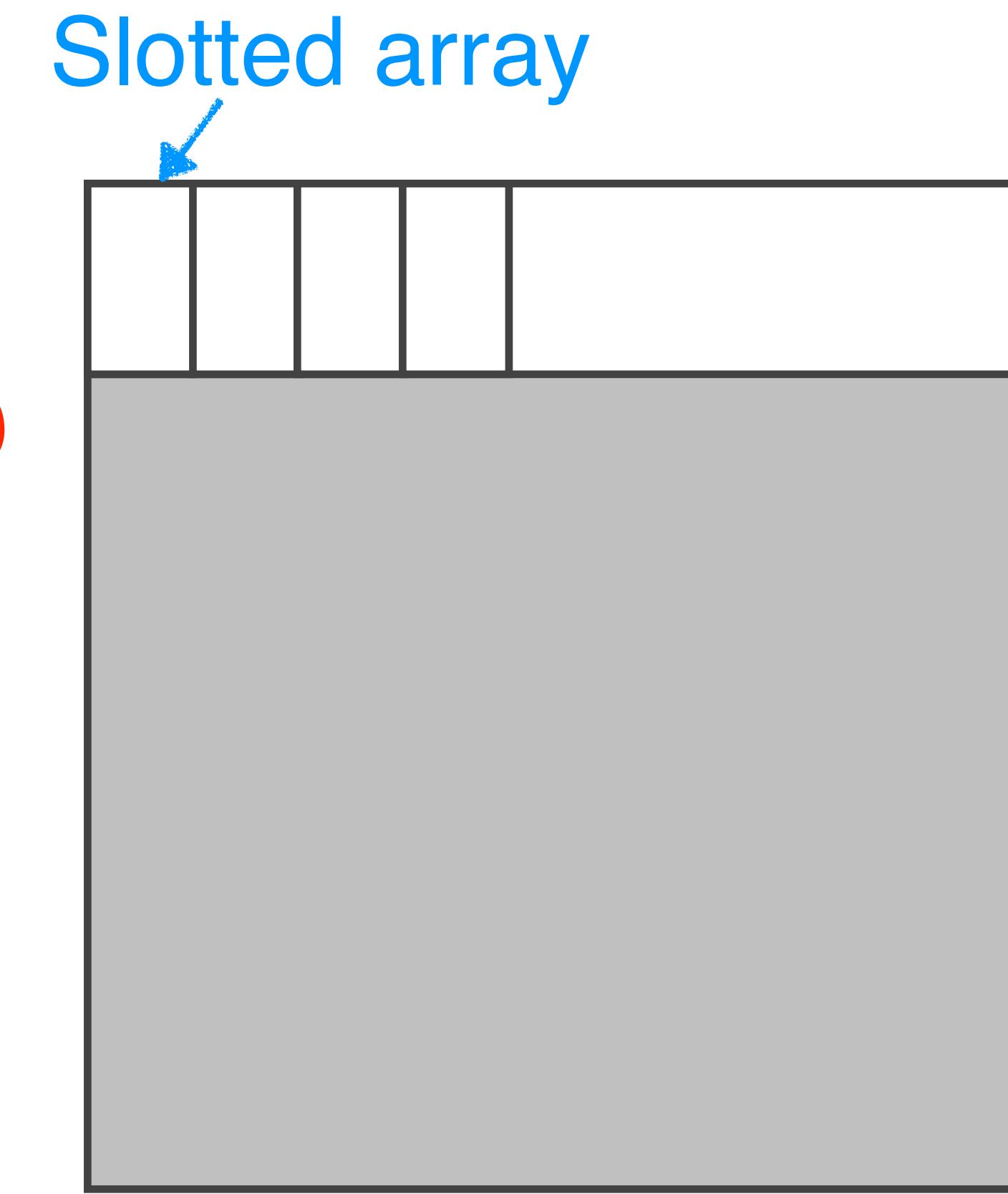


Understanding data placement

Files, pages, tuples

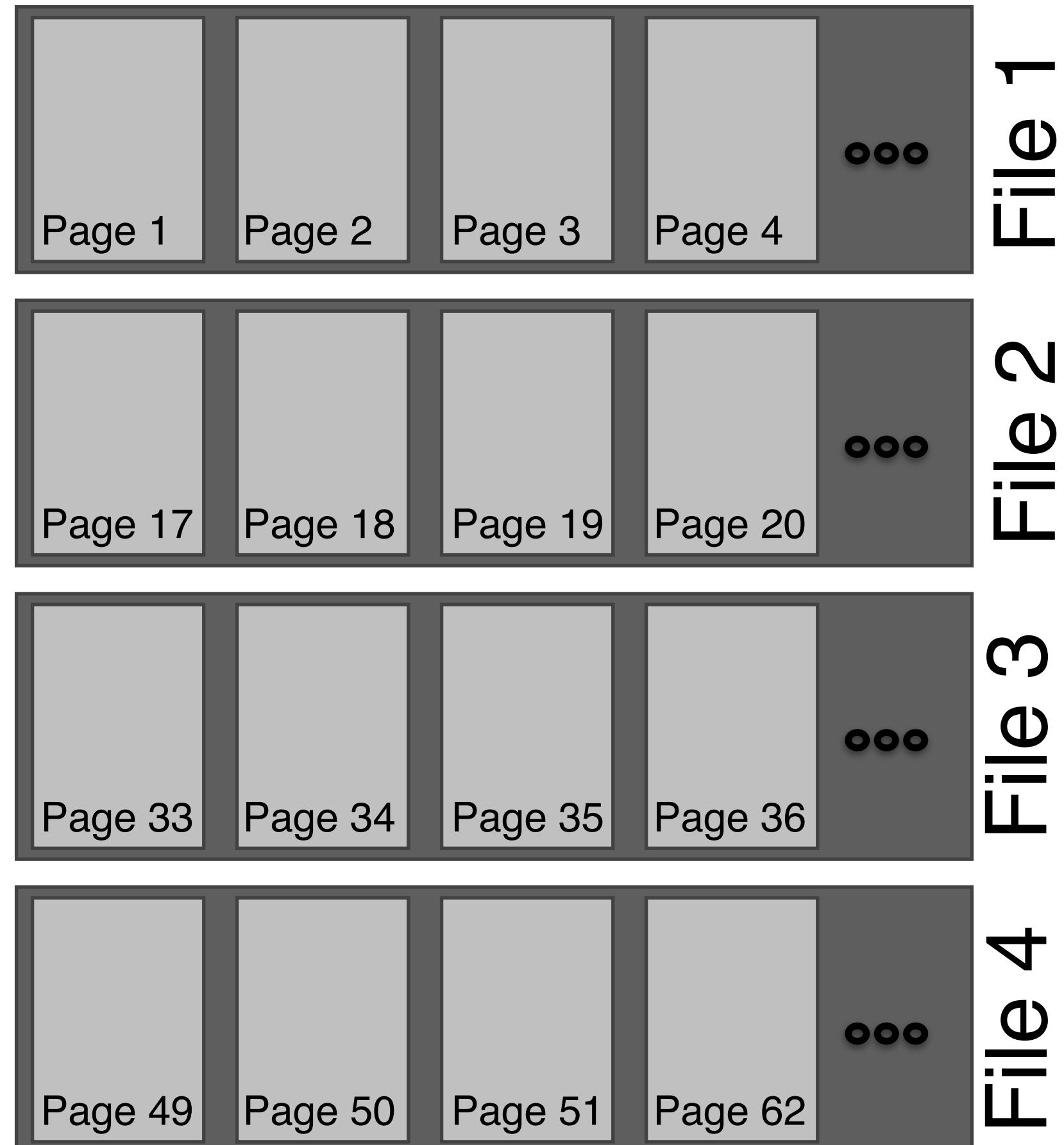


insert(Tuple 1)

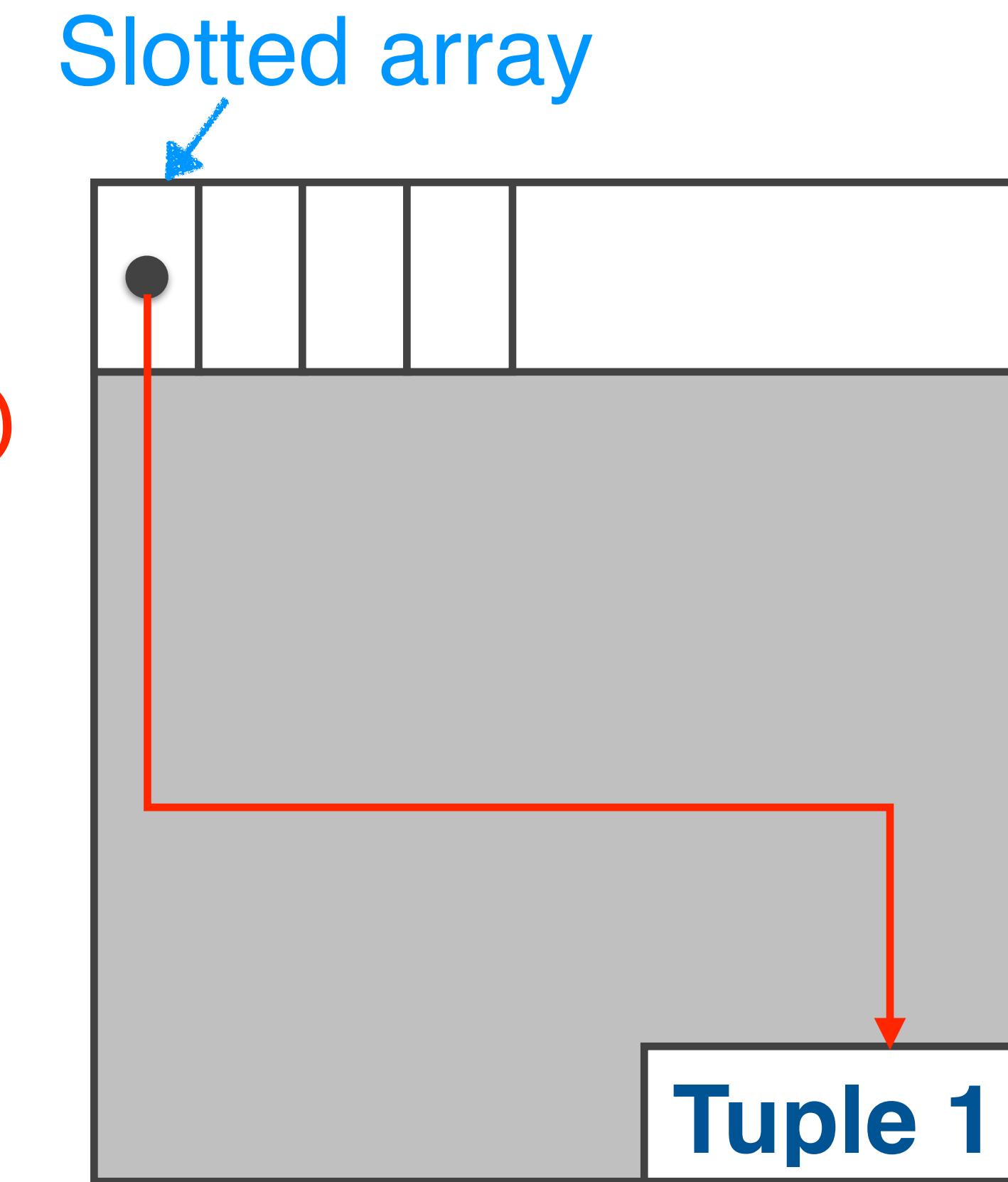


Understanding data placement

Files, pages, tuples



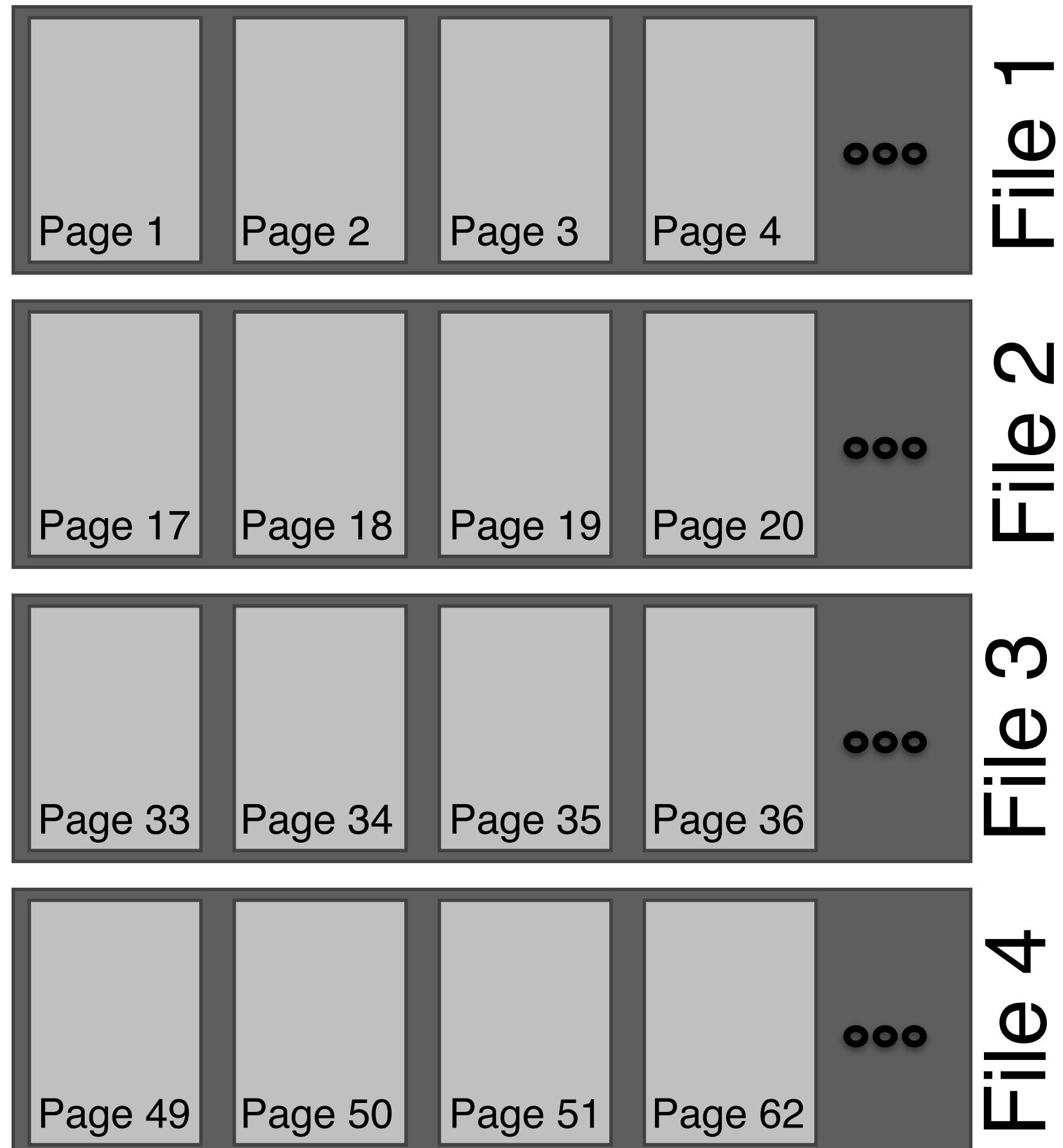
insert(Tuple 1)



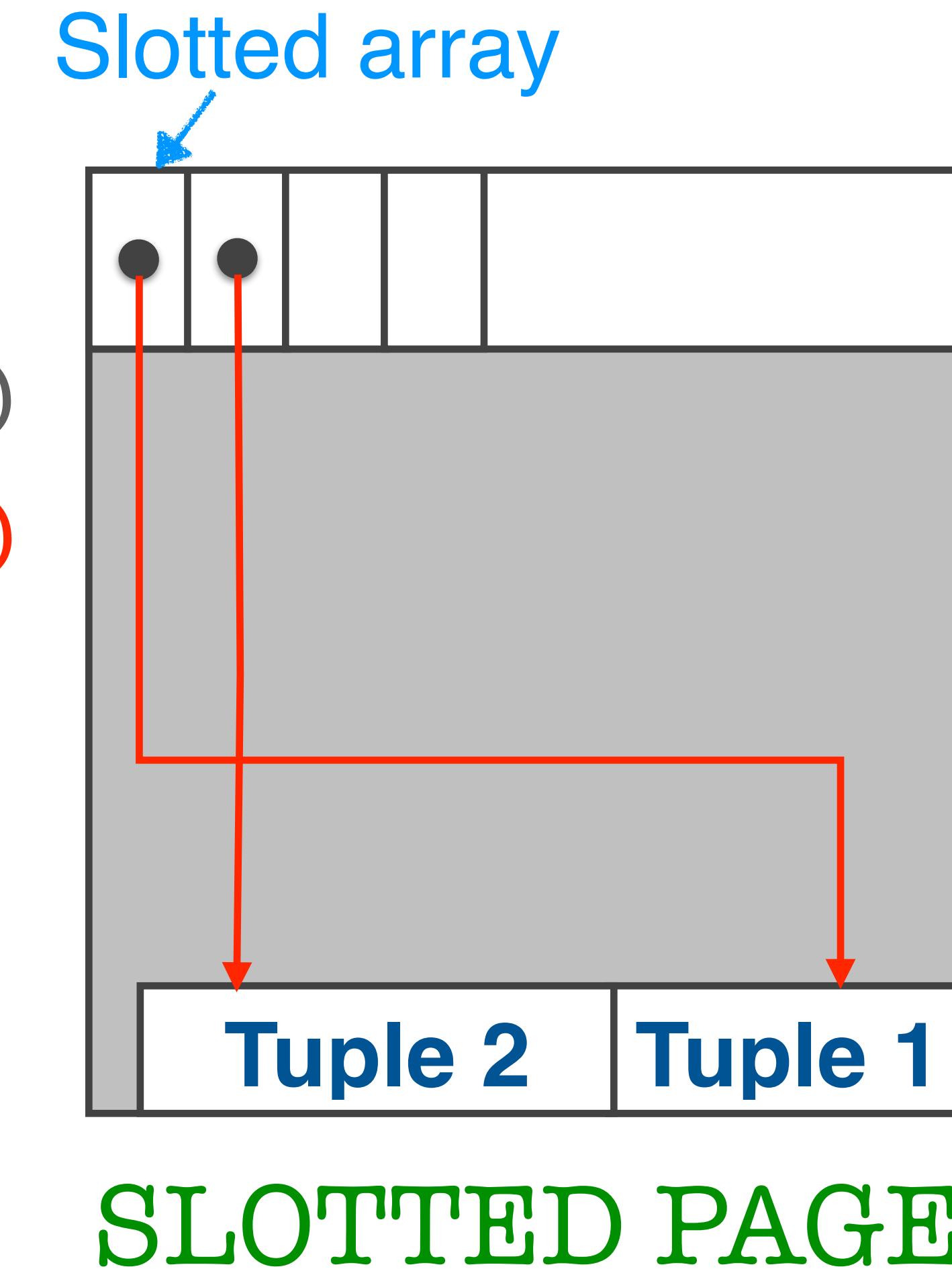
SLOTTED PAGE

Understanding data placement

Files, pages, tuples

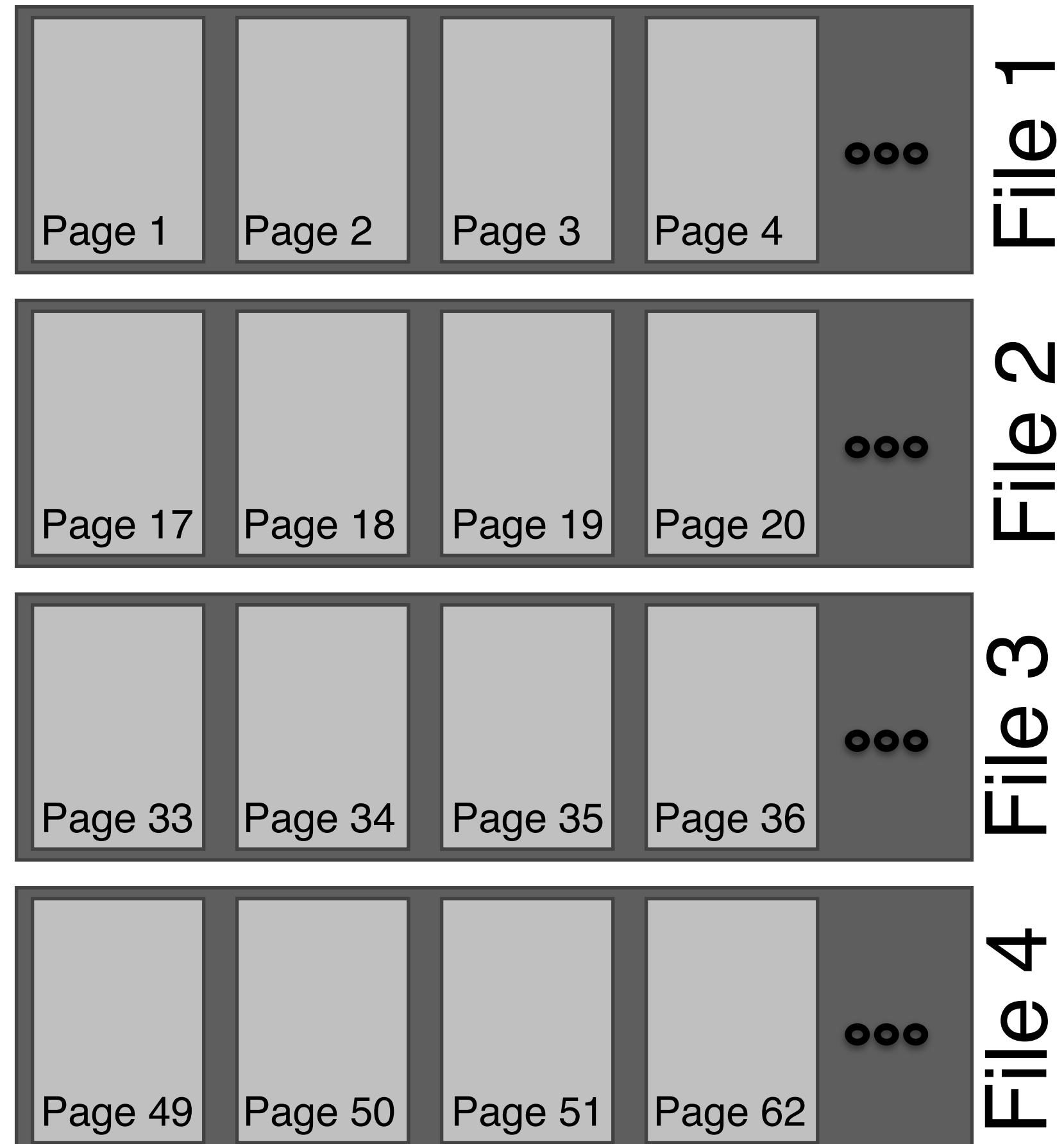


`insert(Tuple 1)`
`insert(Tuple 2)`

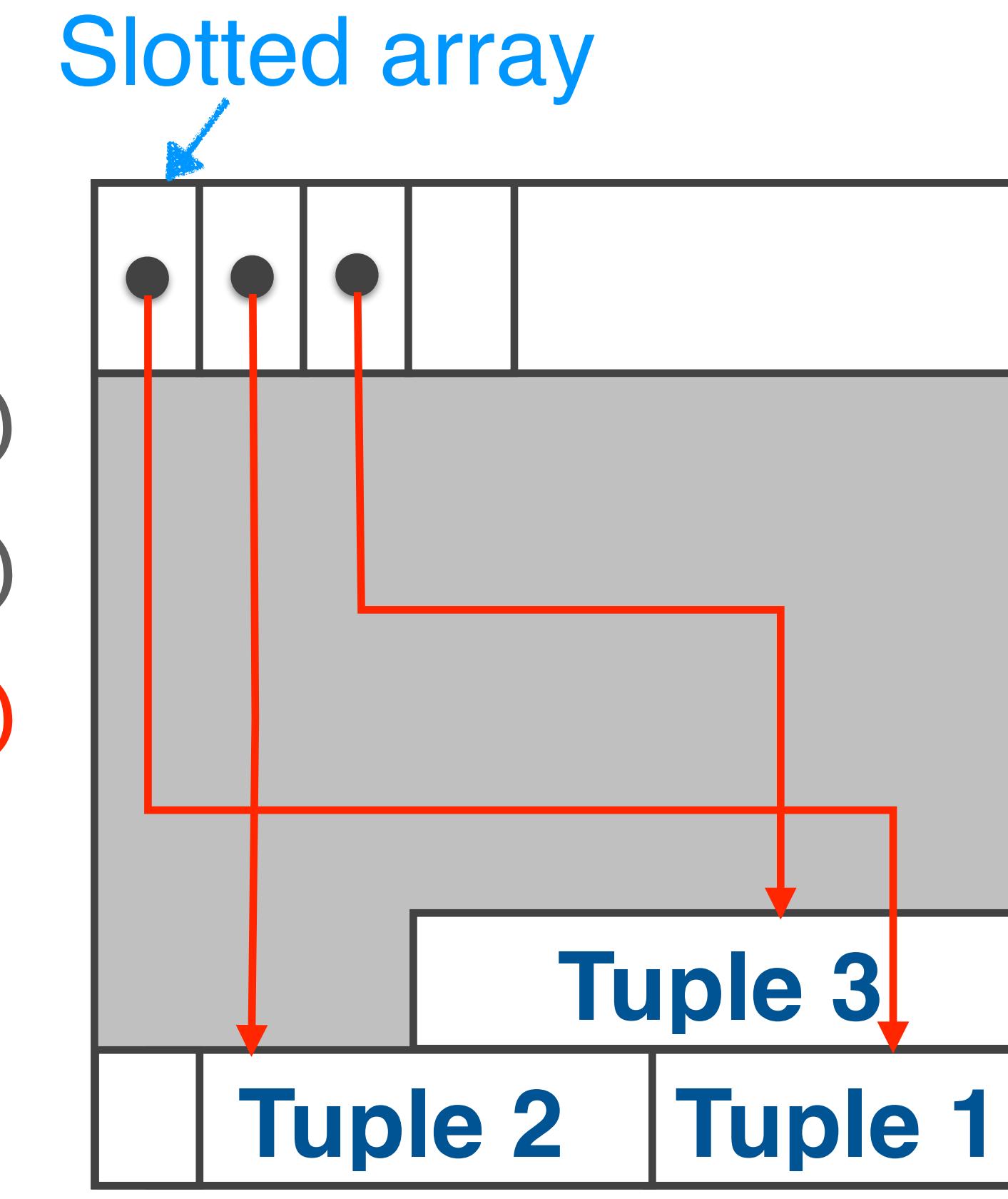


Understanding data placement

Files, pages, tuples



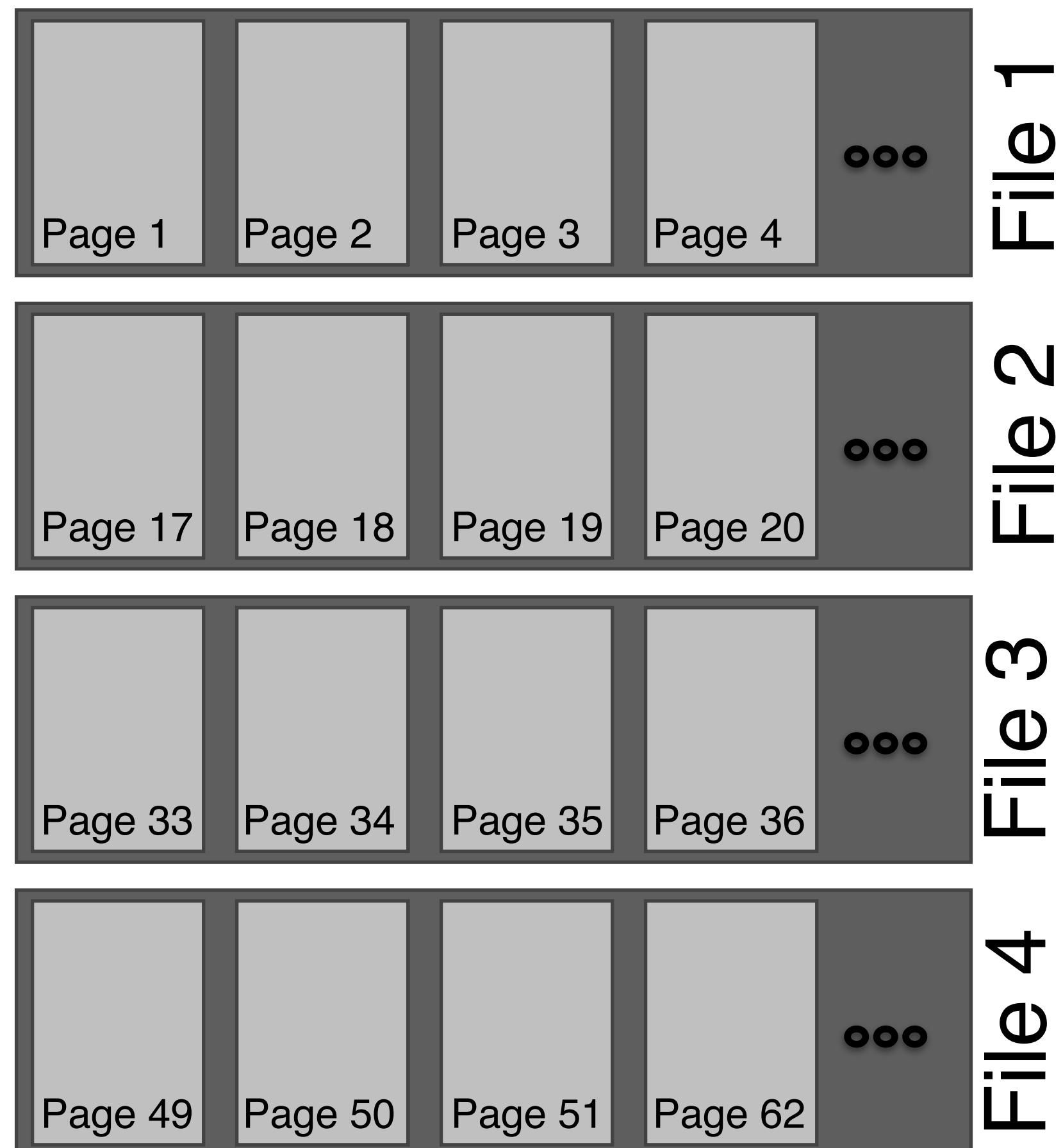
insert(Tuple 1)
insert(Tuple 2)
insert(Tuple 3)



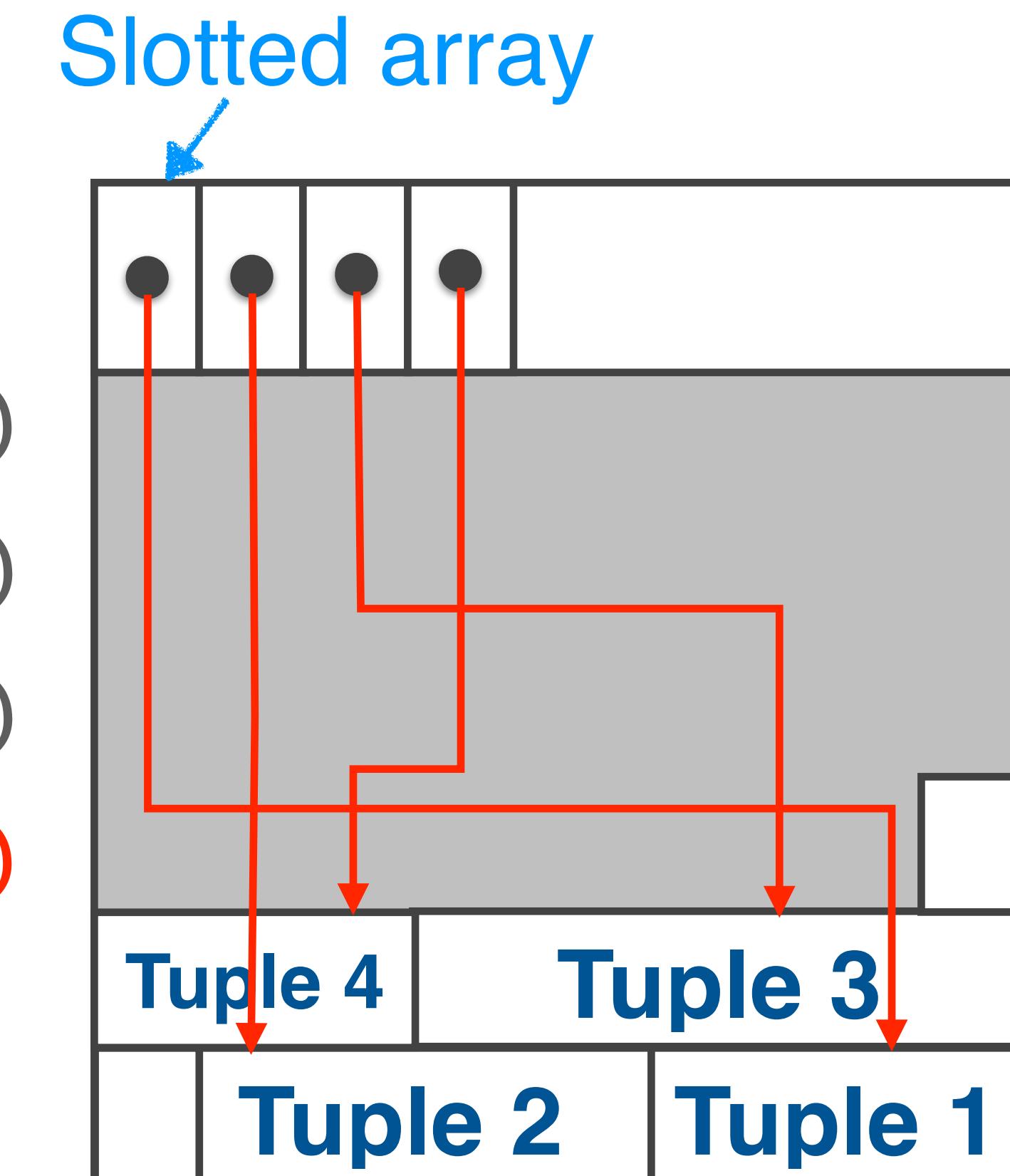
SLOTTED PAGE

Understanding data placement

Files, pages, tuples



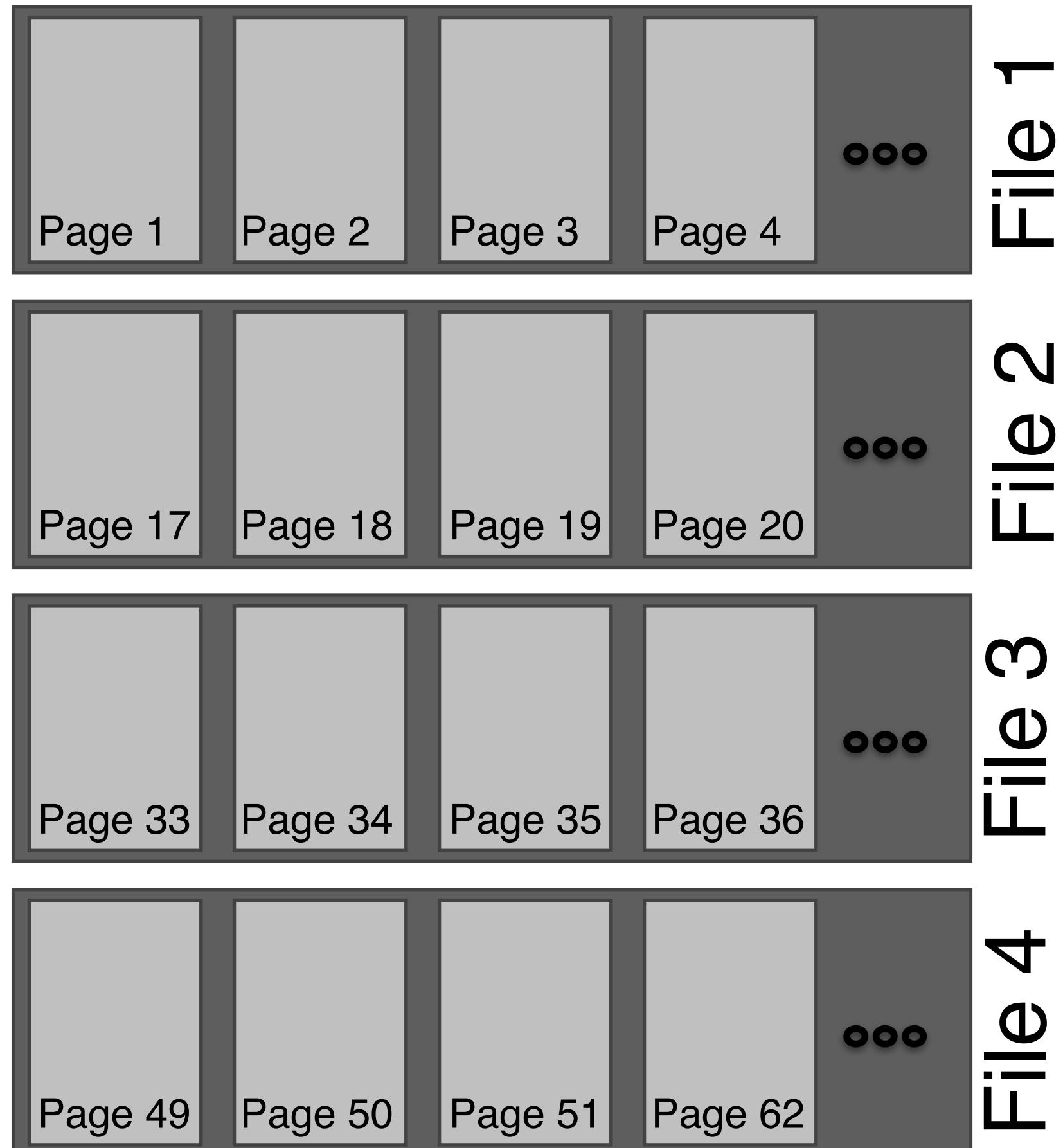
insert(Tuple 1)
insert(Tuple 2)
insert(Tuple 3)
insert(Tuple 4)



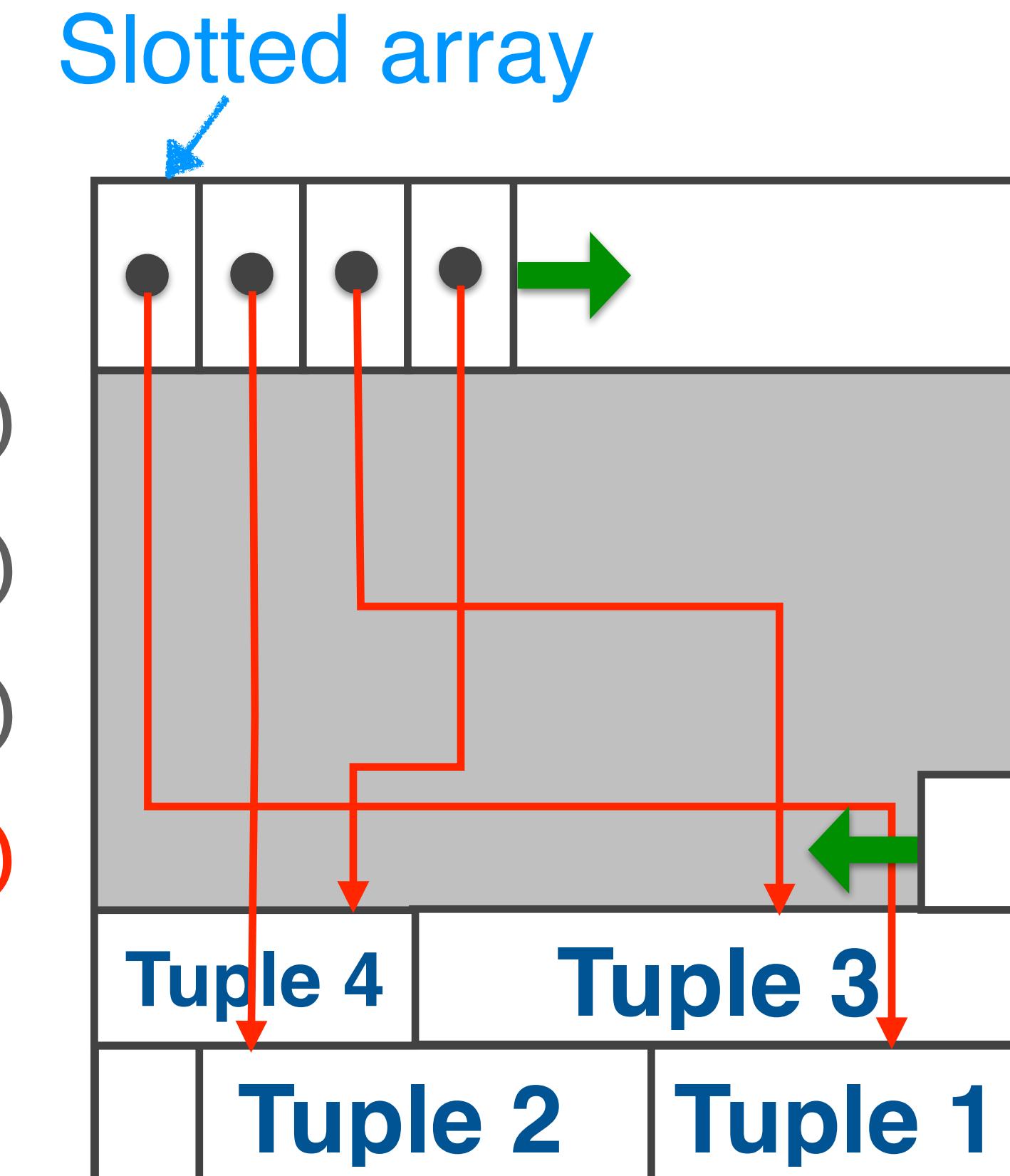
SLOTTED PAGE

Understanding data placement

Files, pages, tuples



insert(Tuple 1)
insert(Tuple 2)
insert(Tuple 3)
insert(Tuple 4)

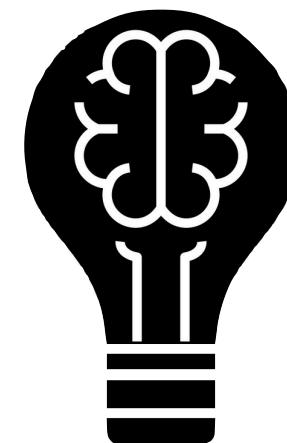


SLOTTED PAGE

Querying over slotted pages

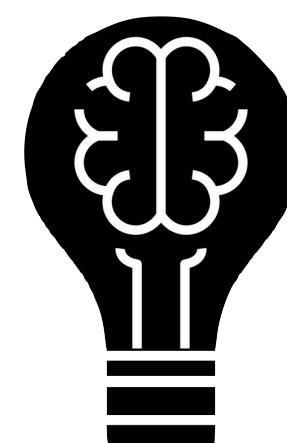
Understanding the schema

file	A	B	C	D
1	A	B	C	D
2	A	B	C	D
3	A	B	C	D
4	A	B	C	D
5	A	B	C	D
6	A	B	C	D
7	A	B	C	D
8	A	B	C	D
9	A	B	C	D
10	A	B	C	D
11	A	B	C	D
12	A	B	C	D
13	A	B	C	D
14	A	B	C	D
15	A	B	C	D
16	A	B	C	D
17	A	B	C	D
18	A	B	C	D
19	A	B	C	D
20	A	B	C	D
21	A	B	C	D
22	A	B	C	D
23	A	B	C	D
24	A	B	C	D
25	A	B	C	D
26	A	B	C	D
27	A	B	C	D
28	A	B	C	D
29	A	B	C	D
30	A	B	C	D
31	A	B	C	D
32	A	B	C	D
33	A	B	C	D
34	A	B	C	D
35	A	B	C	D
36	A	B	C	D
37	A	B	C	D
38	A	B	C	D
39	A	B	C	D
40	A	B	C	D
41	A	B	C	D
42	A	B	C	D
43	A	B	C	D
44	A	B	C	D
45	A	B	C	D
46	A	B	C	D
47	A	B	C	D
48	A	B	C	D
49	A	B	C	D
50	A	B	C	D
51	A	B	C	D
52	A	B	C	D
53	A	B	C	D
54	A	B	C	D
55	A	B	C	D
56	A	B	C	D
57	A	B	C	D
58	A	B	C	D
59	A	B	C	D
60	A	B	C	D
61	A	B	C	D
62	A	B	C	D
63	A	B	C	D
64	A	B	C	D
65	A	B	C	D
66	A	B	C	D
67	A	B	C	D
68	A	B	C	D
69	A	B	C	D
70	A	B	C	D
71	A	B	C	D
72	A	B	C	D
73	A	B	C	D
74	A	B	C	D
75	A	B	C	D
76	A	B	C	D
77	A	B	C	D
78	A	B	C	D
79	A	B	C	D
80	A	B	C	D
81	A	B	C	D
82	A	B	C	D
83	A	B	C	D
84	A	B	C	D
85	A	B	C	D
86	A	B	C	D
87	A	B	C	D
88	A	B	C	D
89	A	B	C	D
90	A	B	C	D
91	A	B	C	D
92	A	B	C	D
93	A	B	C	D
94	A	B	C	D
95	A	B	C	D
96	A	B	C	D
97	A	B	C	D
98	A	B	C	D
99	A	B	C	D
100	A	B	C	D



Thought Experiment 2

select A,B,C,D from R



Thought Experiment 3

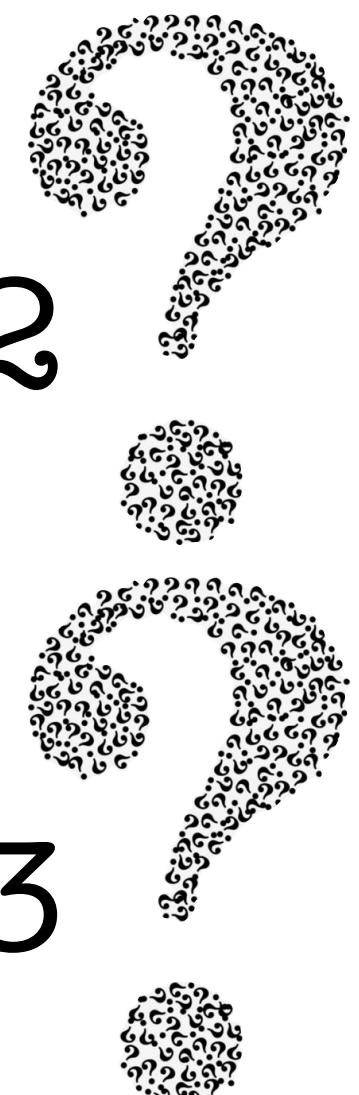
select A from R

Problem?

Read amplification

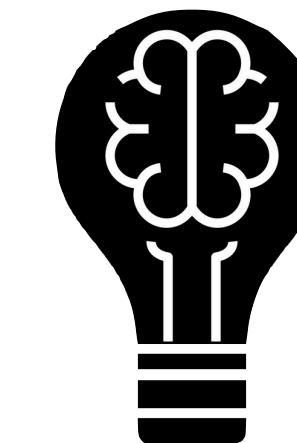
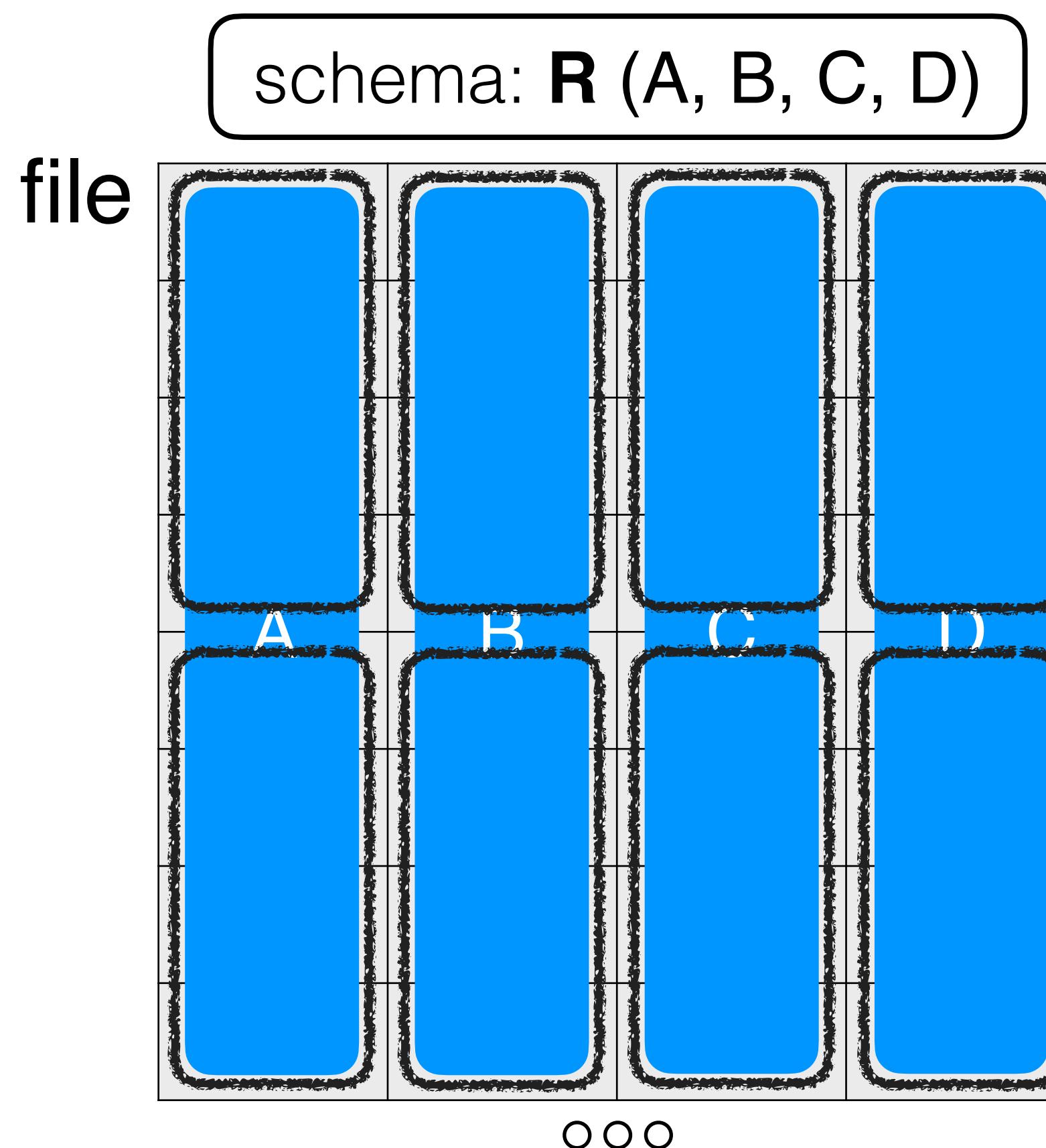
Solution?

Column stores

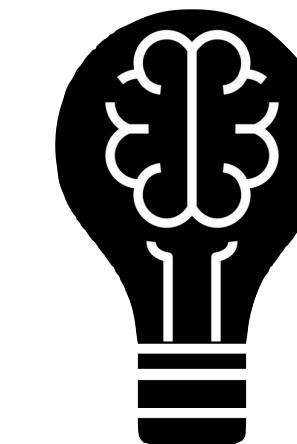


Querying over slotted pages

Understanding the schema



Thought Experiment 3
select A from R



Thought Experiment 2
select A,B,C,D from R

Problem?

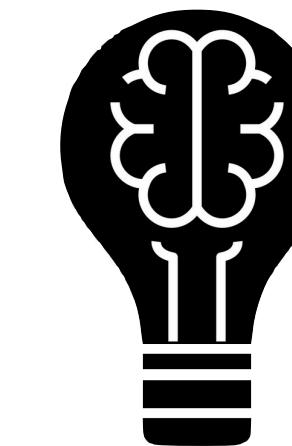
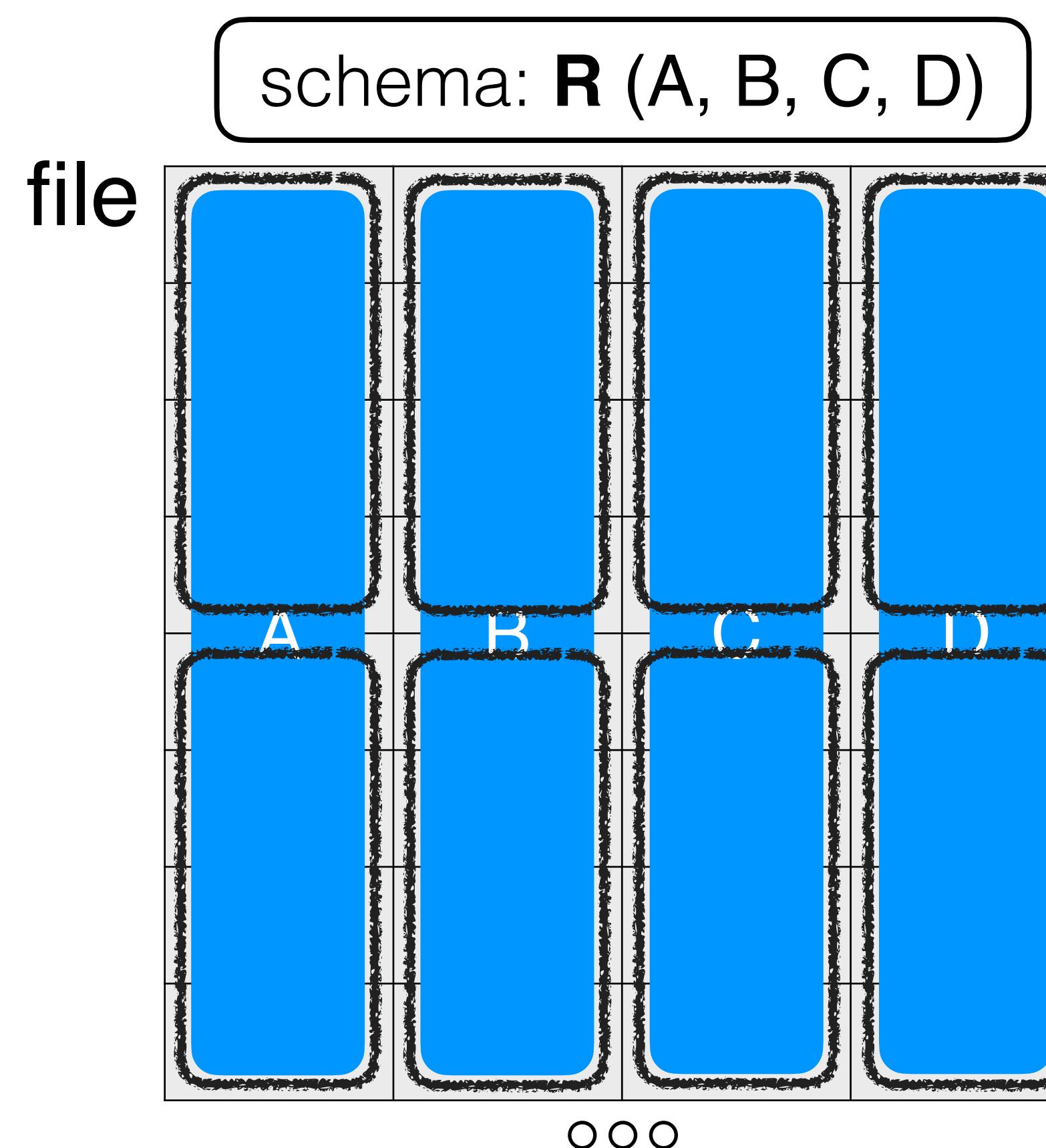
Tuple reconstruction



Brandeis
UNIVERSITY

Querying over slotted pages

Understanding the schema



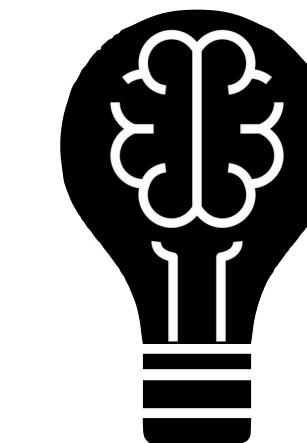
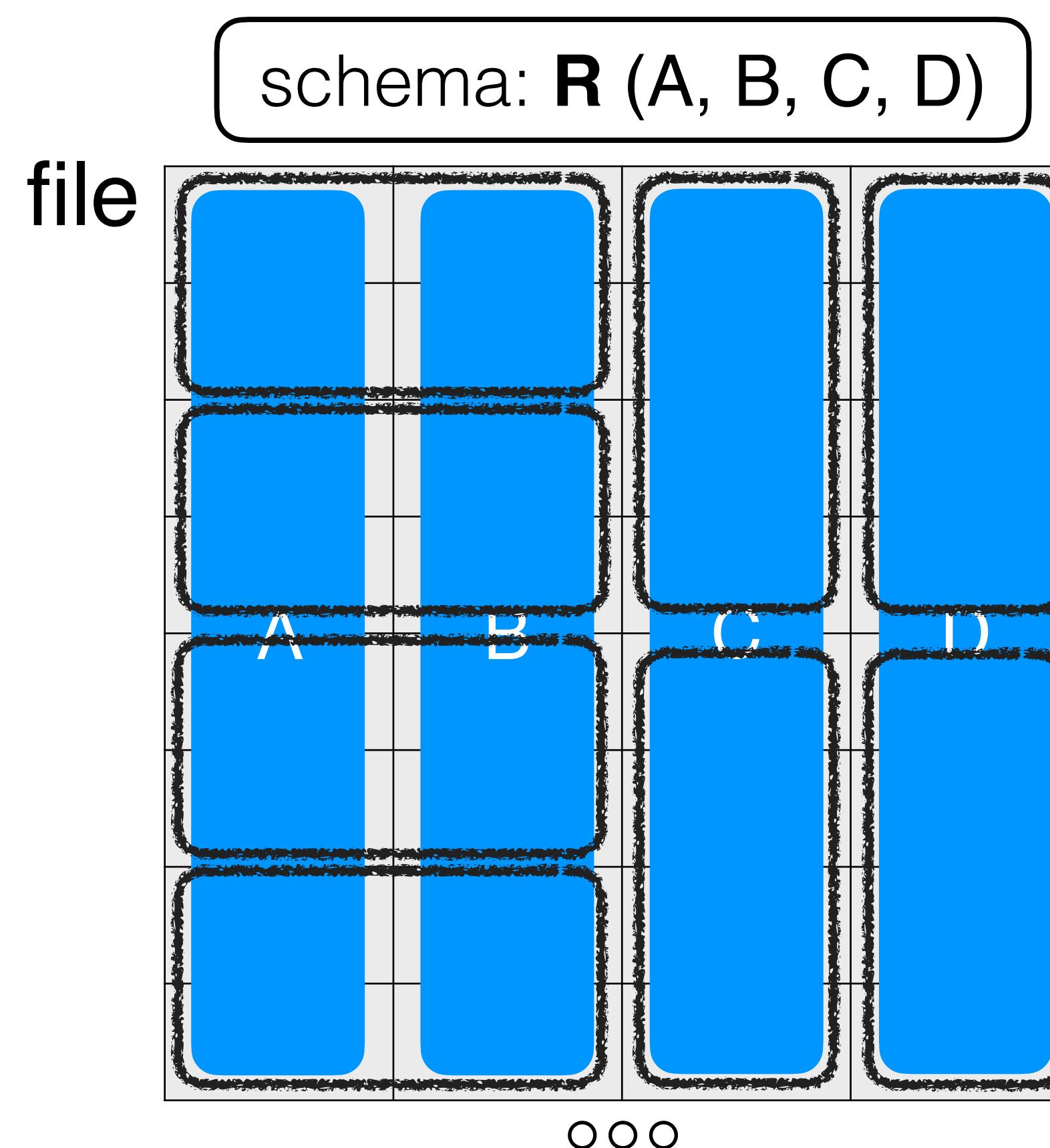
Thought Experiment 4
select A+B from R

Can we do something better?



Querying over slotted pages

Understanding the schema



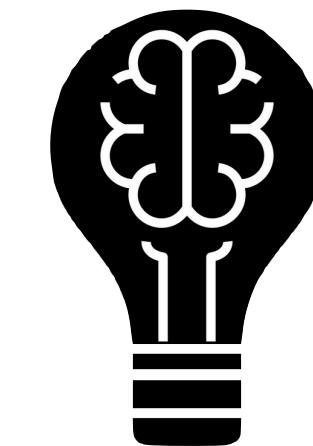
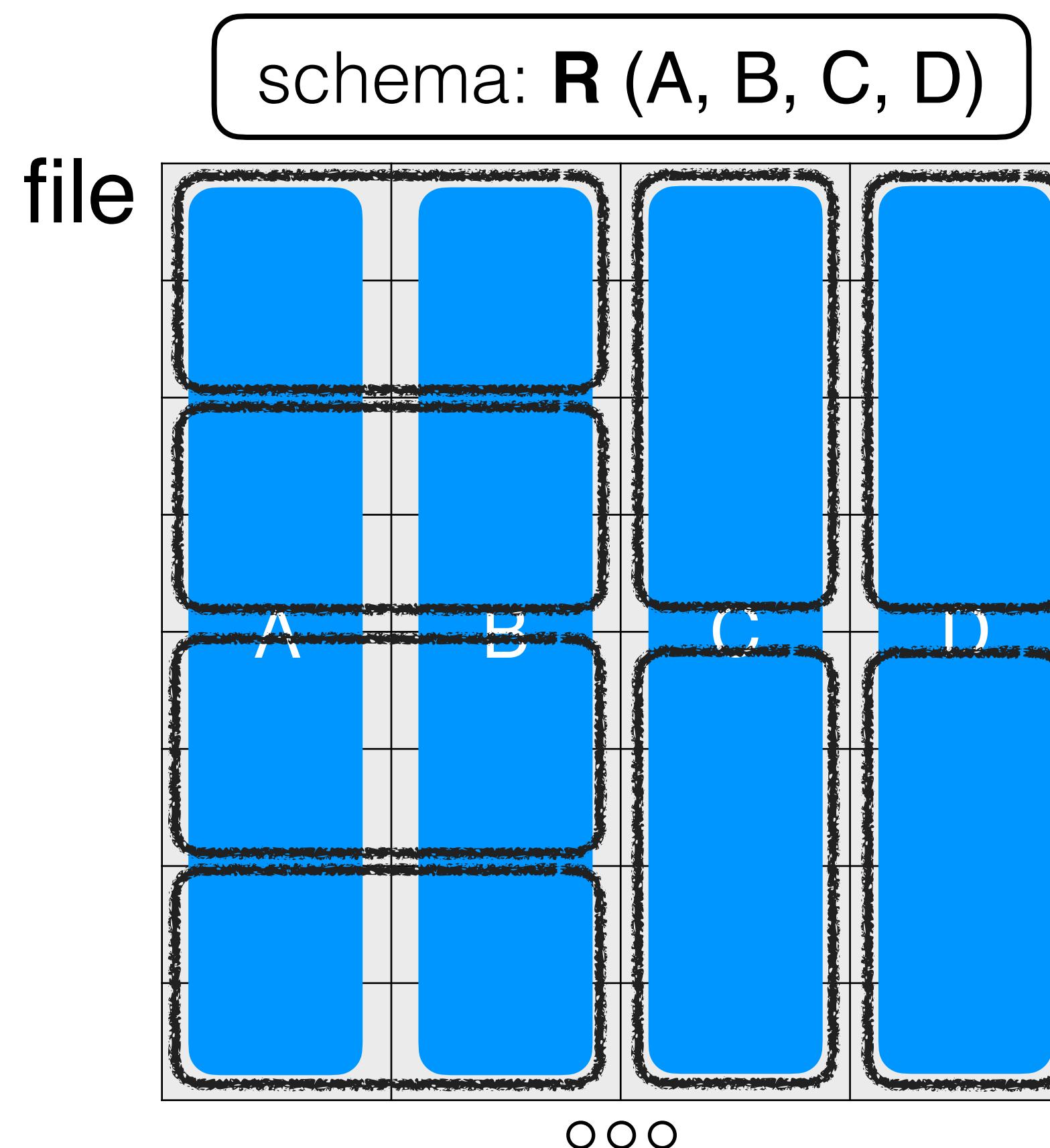
Thought Experiment 4
select A+B from R

Can we do something better?



Querying over slotted pages

Understanding the schema



Thought Experiment 5

select A+B from R

select A,B,C,D from R

select A from R

What if we have all three queries?

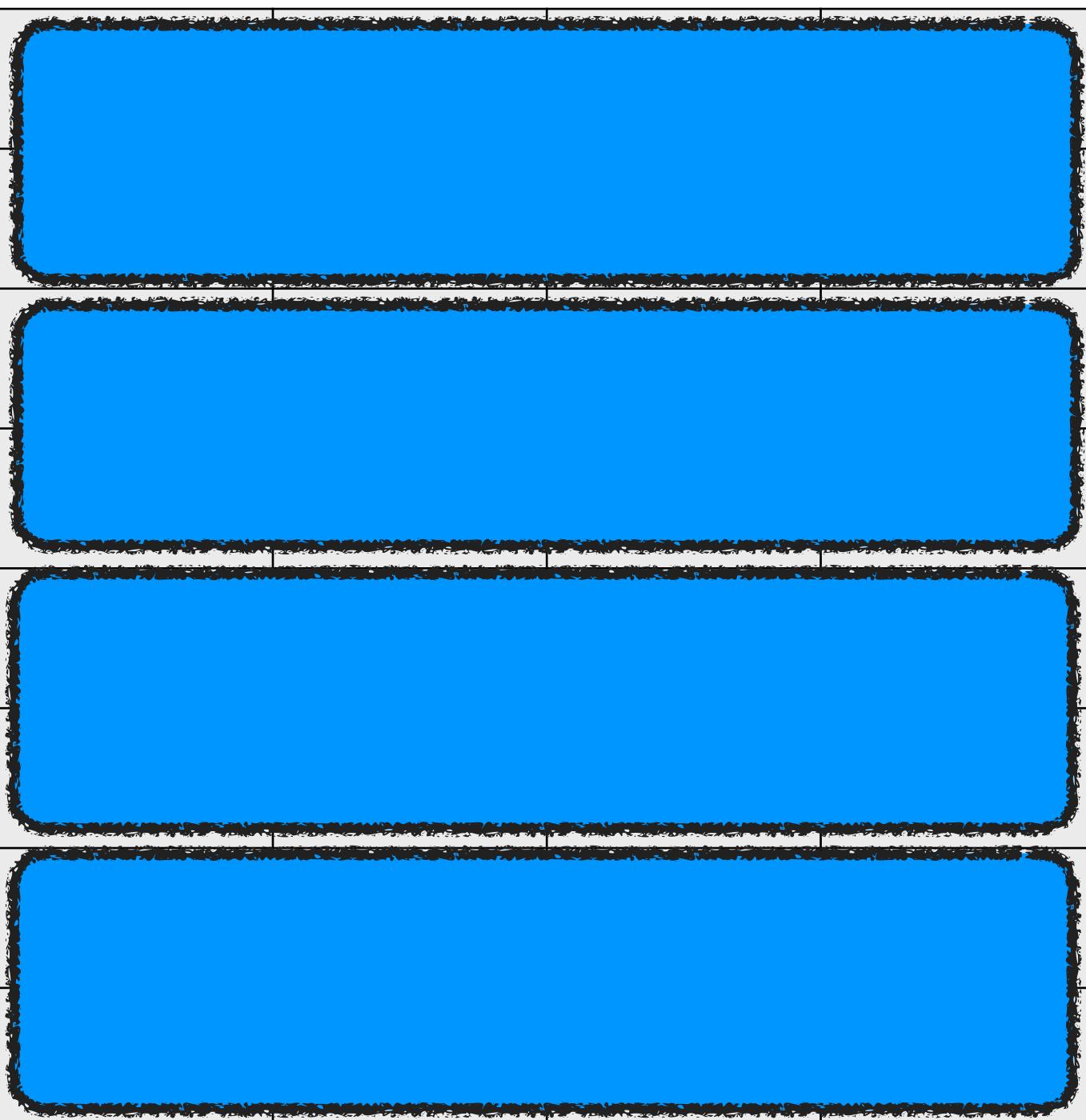
What if we only have inserts/updates?



Querying over slotted pages

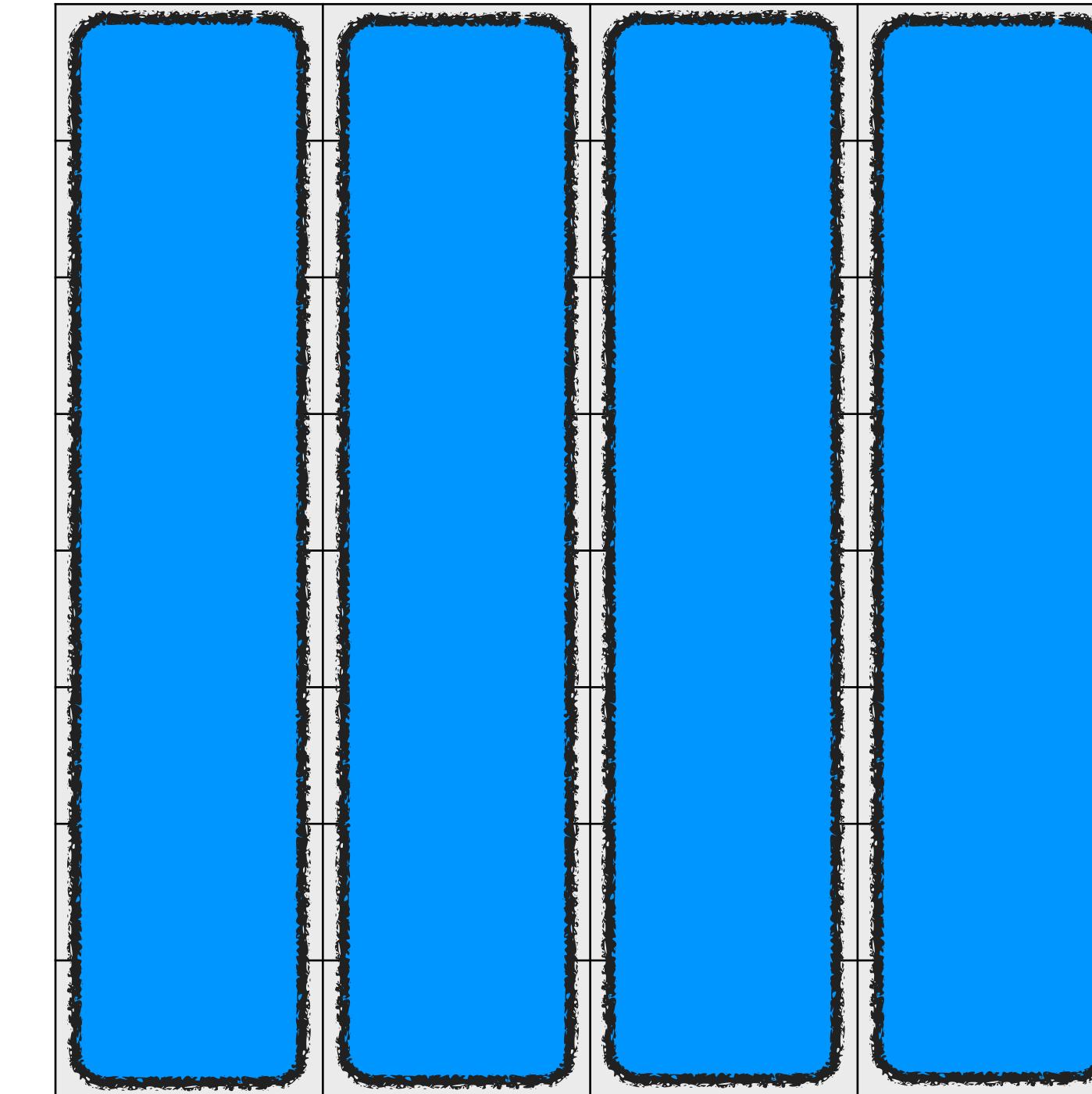
Understanding the schema

schema: **R** (A, B, C, D)



row store

schema: **R** (A, B, C, D)



column store

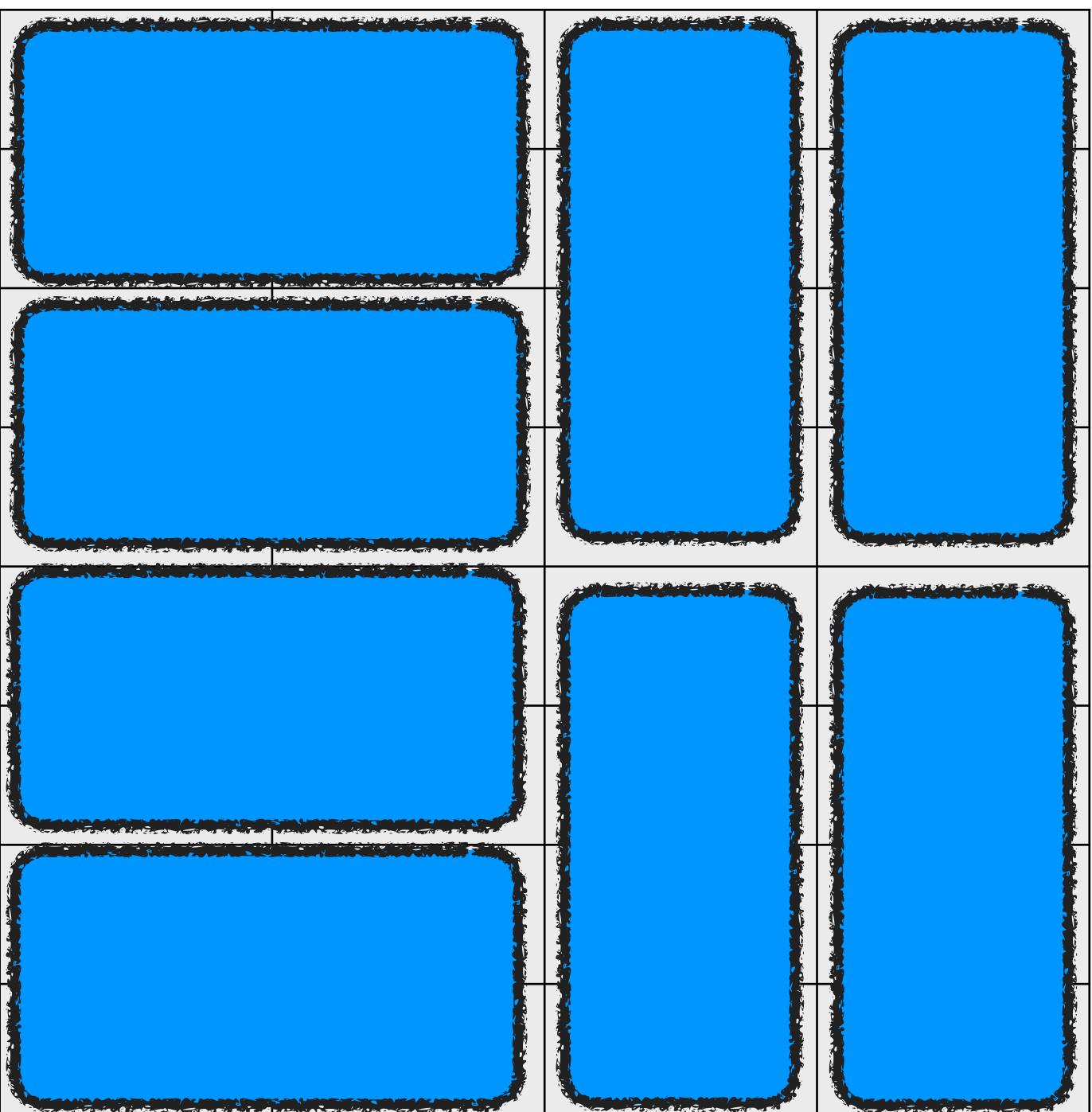
Querying over slotted pages

$R (A, B, C, D)$

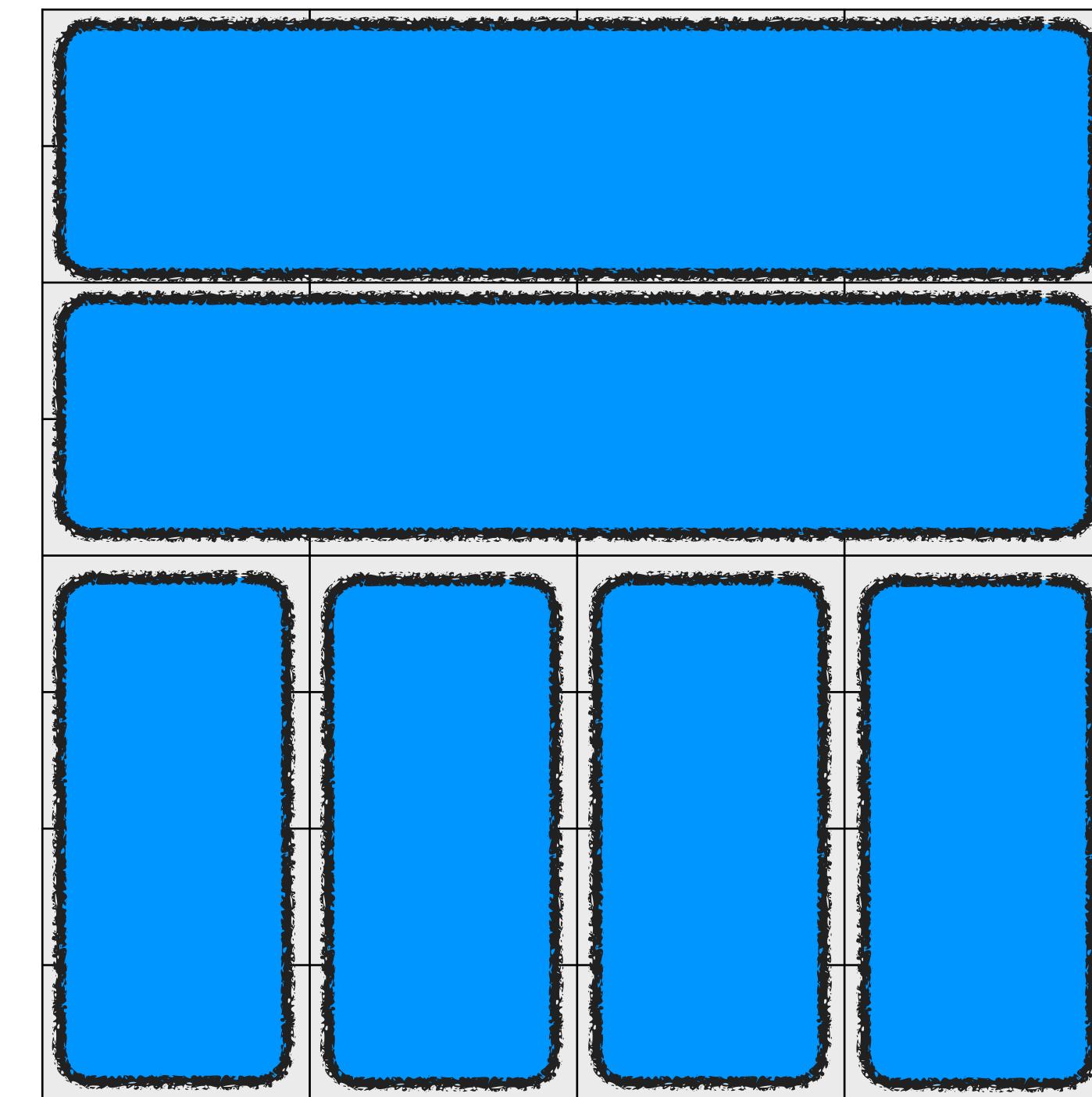


store

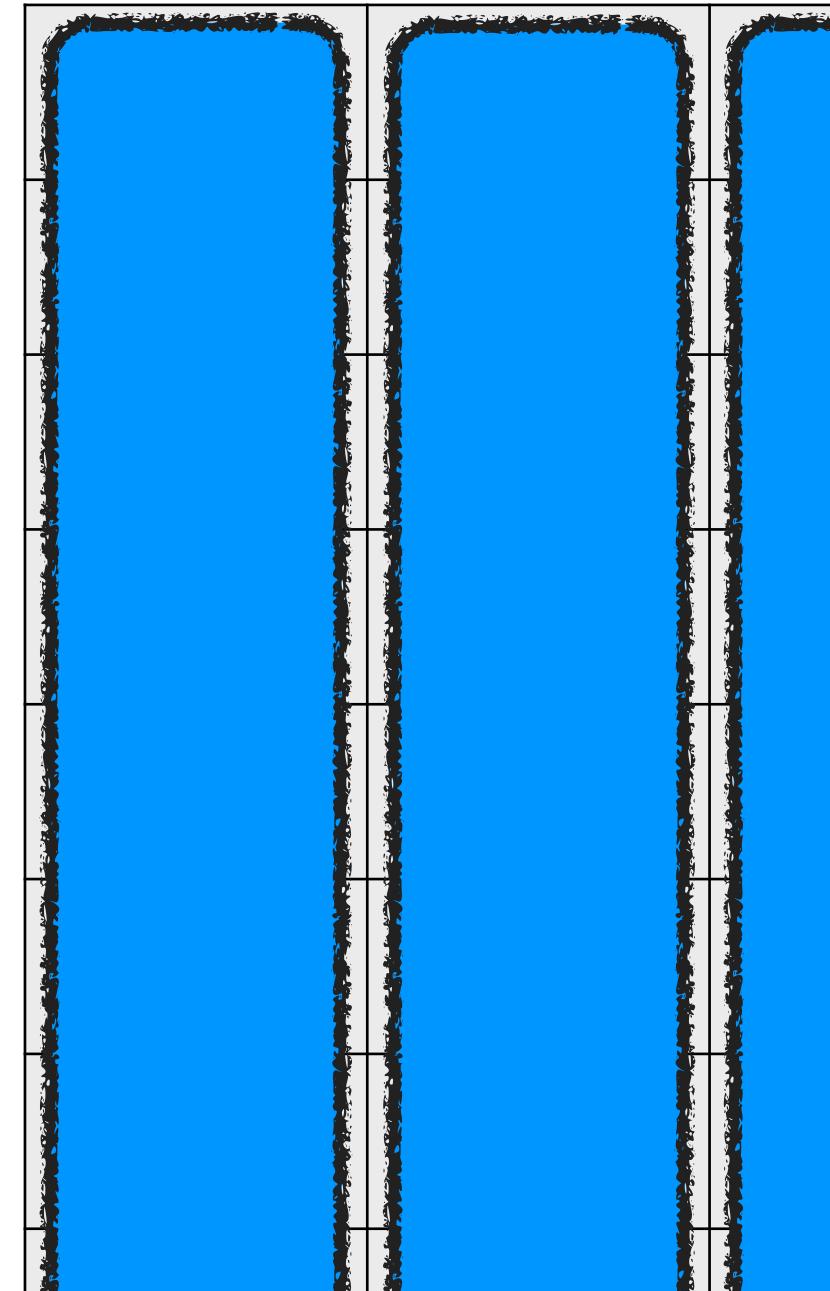
schema: $R (A, B, C, D)$



schema: $R (A, B, C, D)$



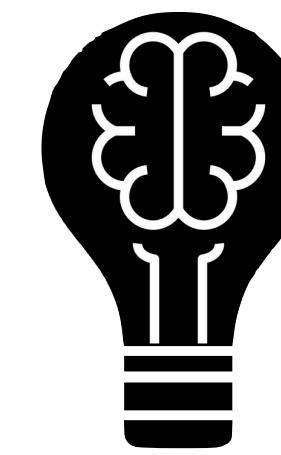
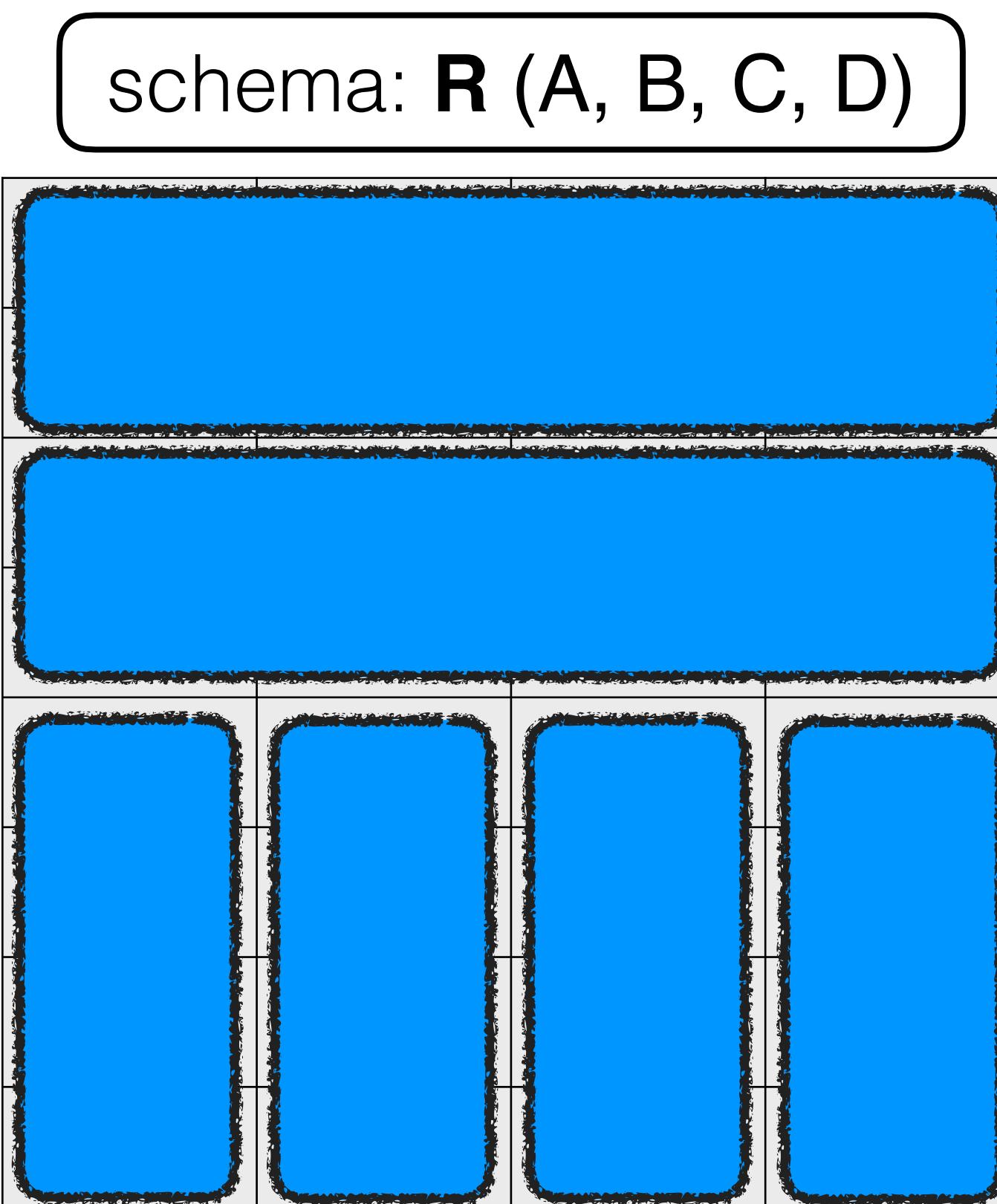
schema: $R (A,$



hybrid data layouts

Querying over slotted pages

Understanding the schema



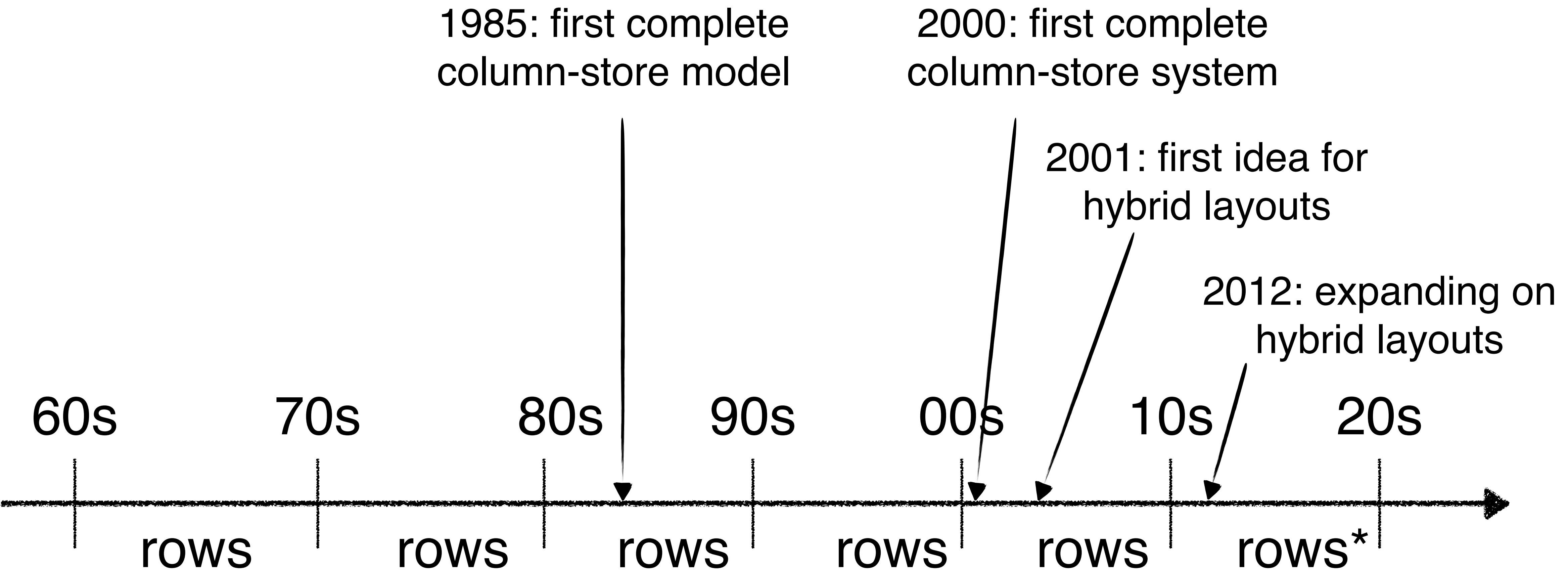
Thought Experiment 6
When would you have this
data layout?

Queries on a subset of data



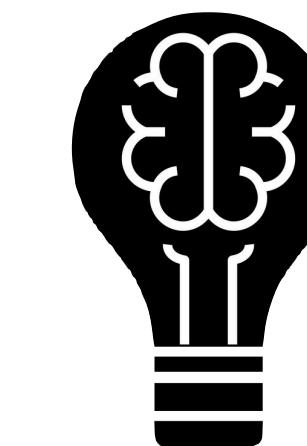
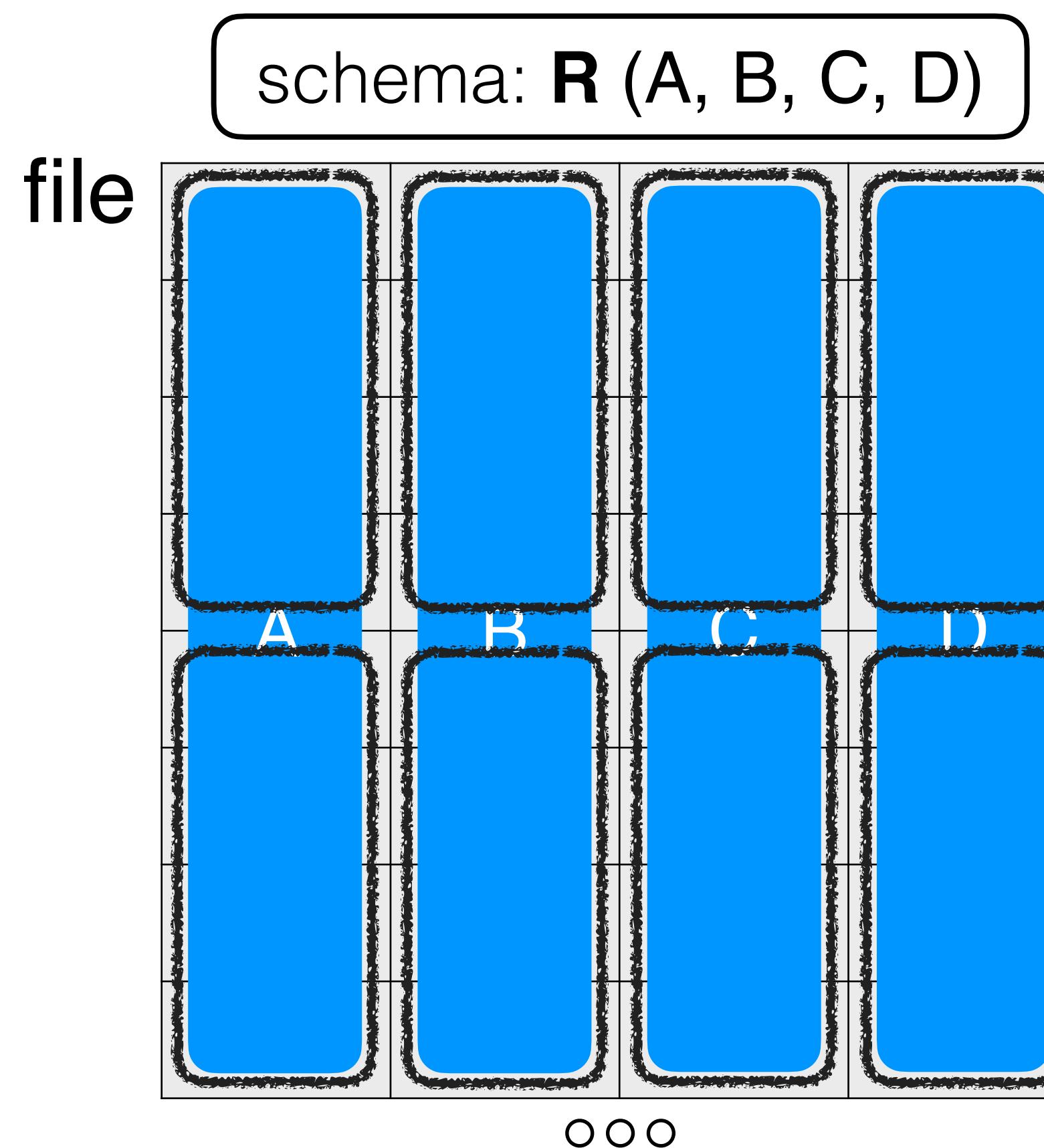
Evolution of column store

From row stores to column stores



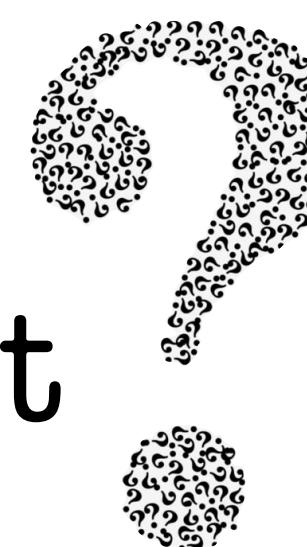
Querying over slotted pages

Understanding the schema



Thought Experiment

select max(B) from R
where A>5 and C<10



Home work!



Brandeis
UNIVERSITY

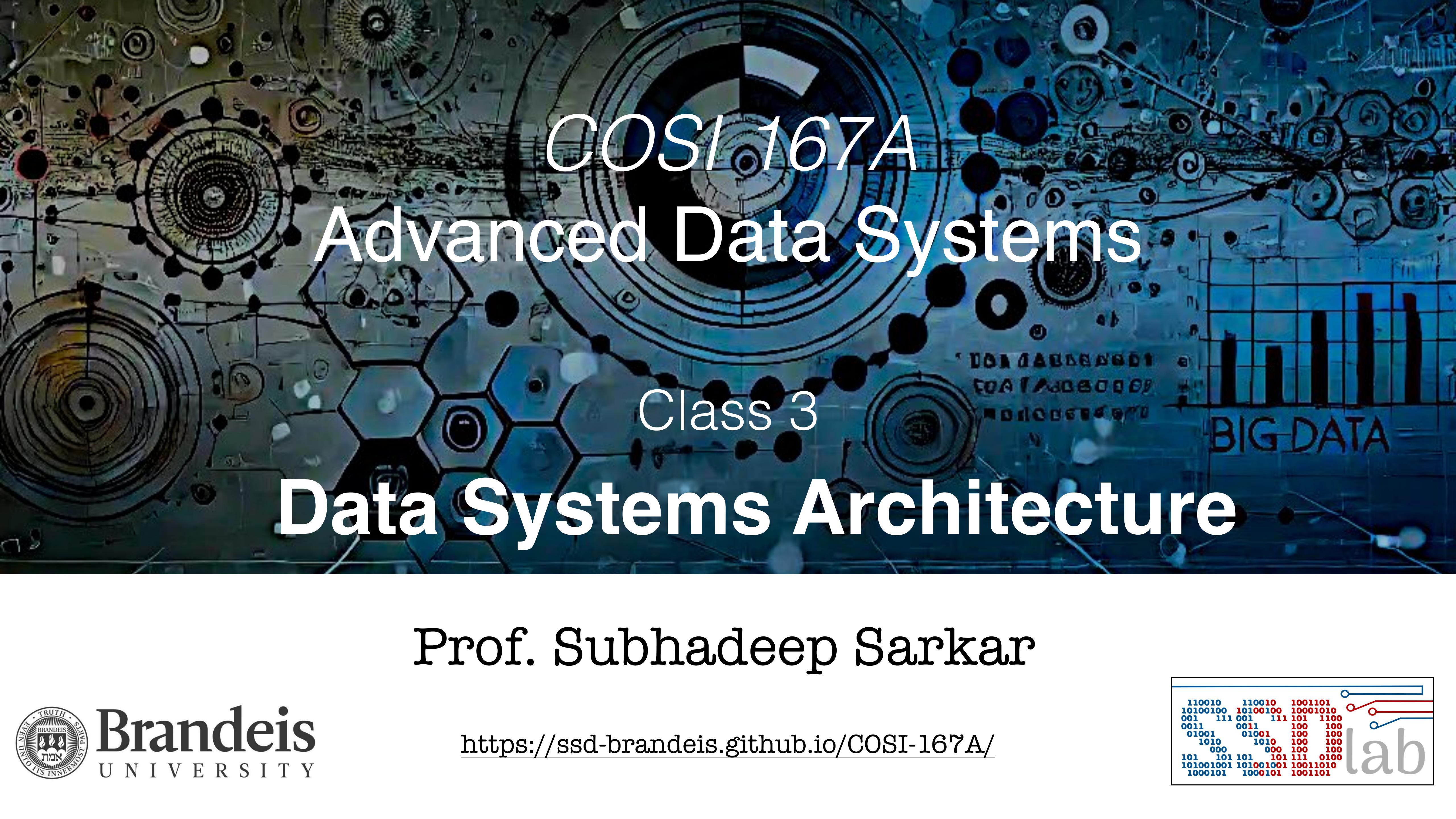
Next time in COSI 167A

Intro. + Administrivia

[P] ["Column-Stores vs. Row-Stores: How Different Are They Really?", SIGMOD, 2008](#)

TECHNICAL QUESTION 1

[B] ["C-Store: A Column-oriented DBMS", VLDB, 2005](#)



COSI 167A

Advanced Data Systems

Class 3

Data Systems Architecture

Prof. Subhadeep Sarkar



Brandeis
UNIVERSITY

<https://ssd-brandeis.github.io/COSI-167A/>

