BOSTON UNIVERSITY

CS 561: Data Systems Architectures

class 2

Data Systems 101

Prof. Manos Athanassoulis

https://bu-disc.github.io/CS561/

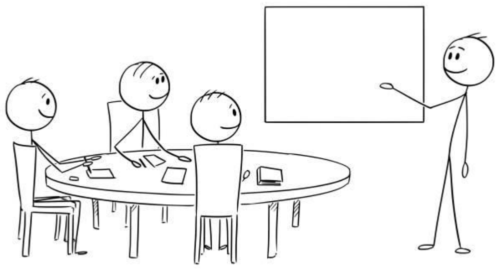
some reminders

no smartphones

no laptop

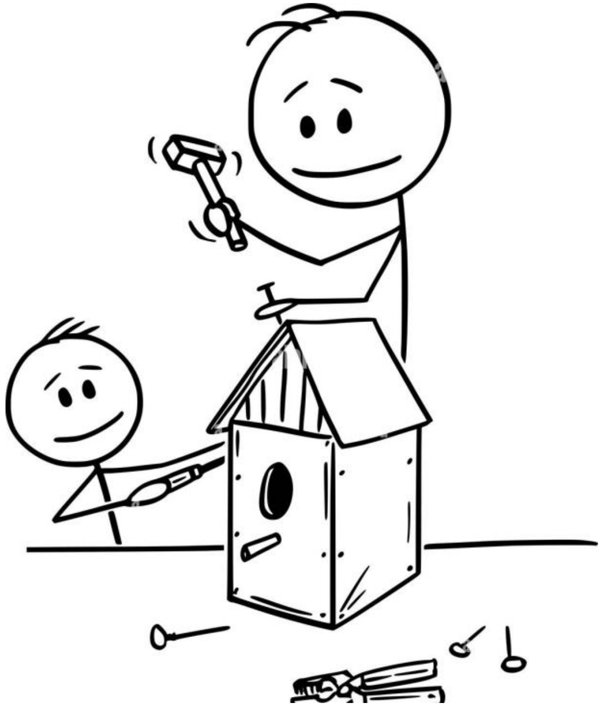
What do we do in this class?

reading papers



presentations

quizzes + exam



projects

class summary

2 classes per week & OH/Labs 4 multiple per week

each student

1 paper discussion (as presenter or critic or proponent) + Quizzes + exam

project 0 (individual project) + project 1 (group project)

systems or research project (group project)

proposal + mid-semester report + final report/presentation

Projects

AND

project 0

A small implementation project to sharpen dev skills

<u>independent project</u>

Due on Jan 31, 2025

project 1

A medium project to give you a flavor of large-scale production system

<u>groups of 3</u>

Projects

AND

project 0

A small implementation project
to sharpen dev skills

<u>independent project</u>

Due on Jan 31, 2025

project 1

A medium project to give you a flavor of
large-scale production system

<u>groups of 3</u>

Start forming groups NOW!

Due on Feb 14, 2025

Projects

OR

systems project

groups of 3

implementation-heavy C/C++ project

research project

groups of 3

pick a subject (list available on the website)

design & analysis

experimentation

Projects

OR

systems project

groups of 3

implementation-heavy C/C++ project

research project

groups of 3

pick a subject (list available on the website)

more to come!

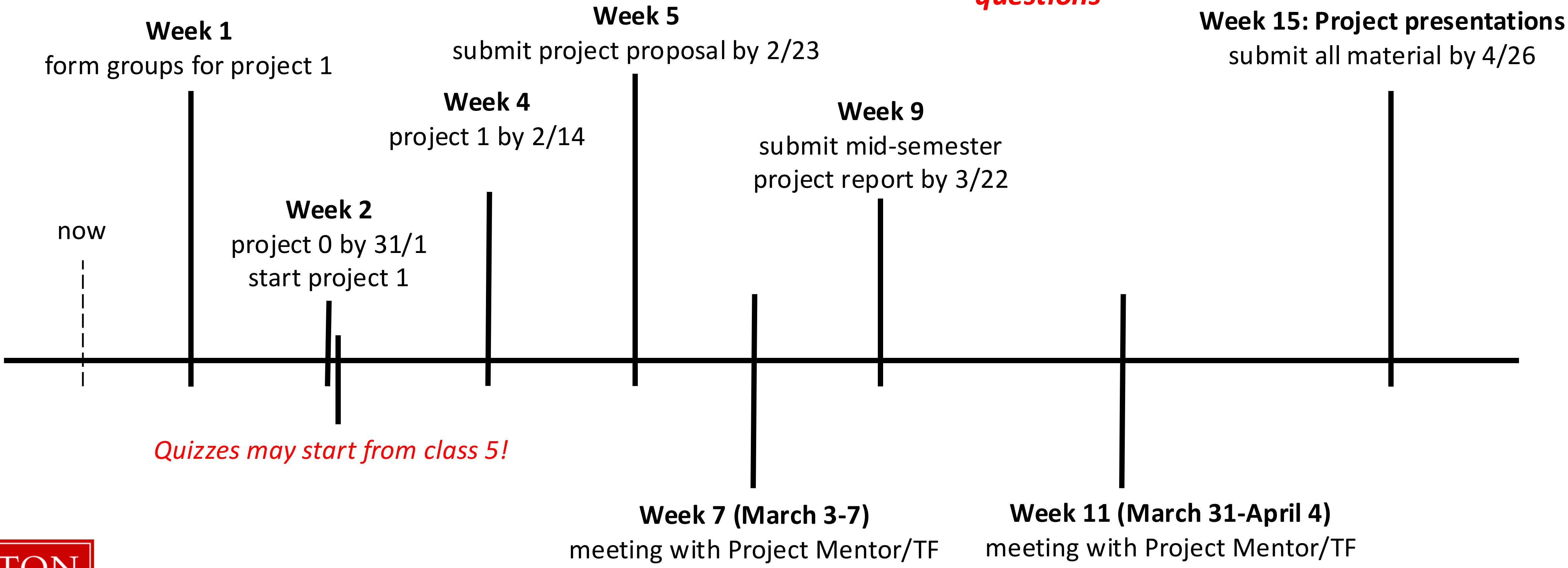
design & analysis

experimentation

1. Proposal
2. Mid-semester report
3. Final report + Presentation

class timeline

discussions
interaction in OH & Lab
questions



now

Week 1
form groups for project 1

Week 2
project 0 by 31/1
start project 1

Quizzes may start from class 5!

Week 4
project 1 by 2/14

Week 5
submit project proposal by 2/23

Week 7 (March 3-7)
meeting with Project Mentor/TF

Week 9
submit mid-semester
project report by 3/22

Week 11 (March 31-April 4)
meeting with Project Mentor/TF

Week 15: Project presentations
submit all material by 4/26

Piazza

2 classes per week & OH/Labs multiple times per week

all discussions & announcements

http://piazza.com/bu/spring2025/cs561/

also available on class website

We have added everyone who already registered!

Please double-check!

BOSTON UNIVERSITY

size (volume)

rate (velocity)

sources (variety)

veracity & value

big data

(it's not only about size)

The 3 V's

size (volume)

rate (velocity)

sources (variety)

veracity & value

big data

(it's not only about size)

The 3 V's

+ our ability to collect ***machine-generated*** data

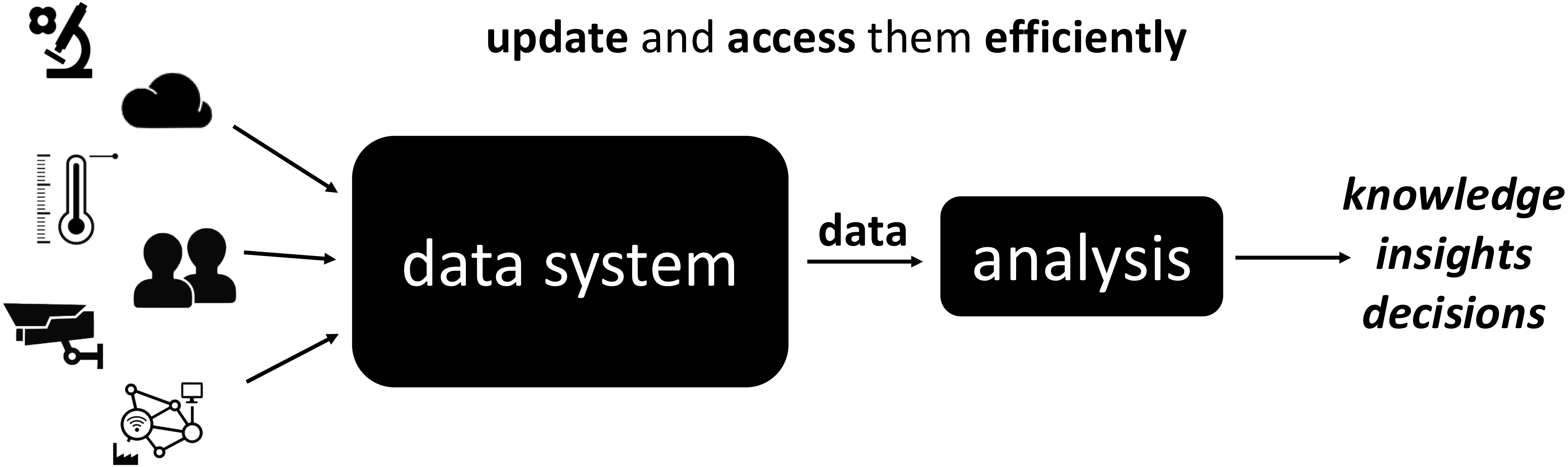
scientific experiments

sensors

social

Internet-of-things

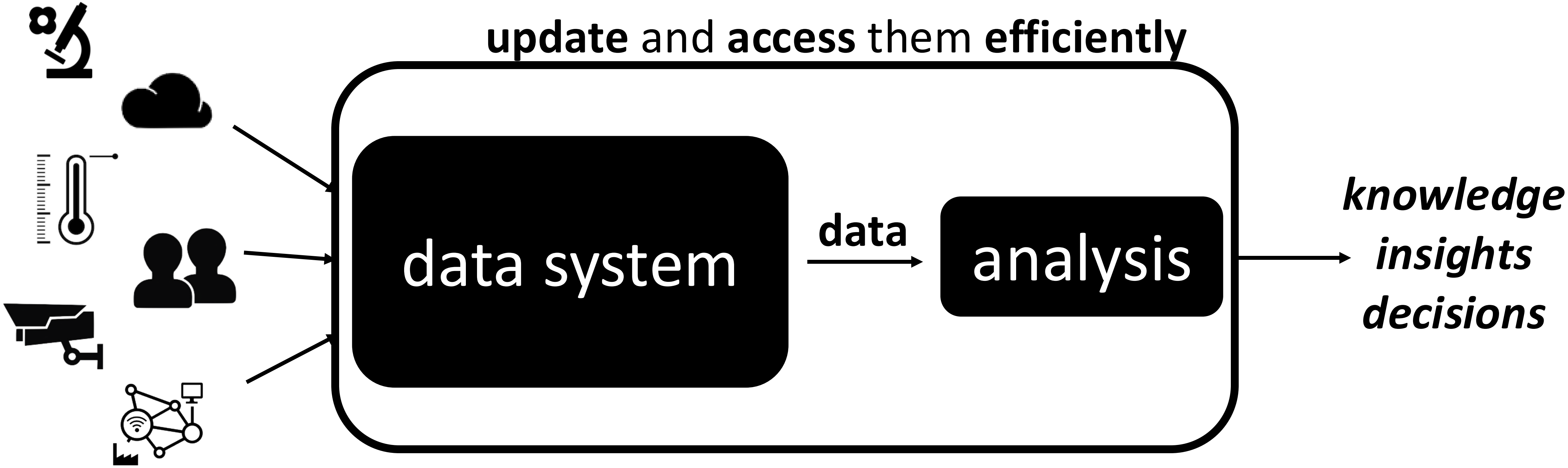
a **data system** is a large software system that **stores data**, and provides the **interface** to **update** and **access** them **efficiently**



a **data system** is a large software system

that **stores data**, and provides the **interface** to

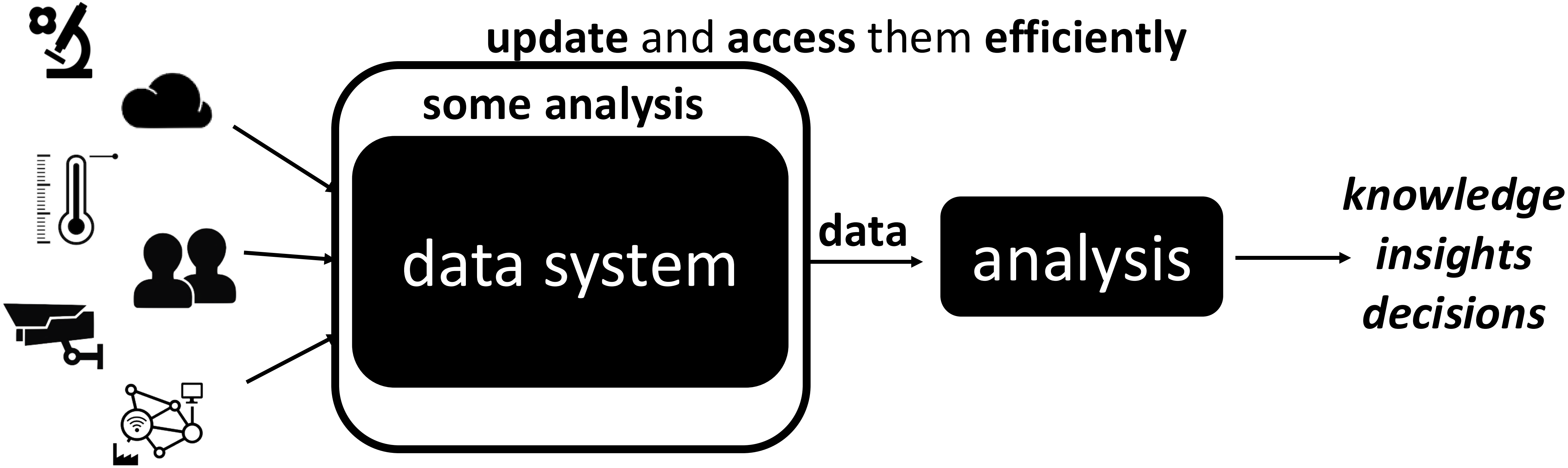
update and **access** them **efficiently**



a **data system** is a large software system

that **stores data**, and provides the **interface** to

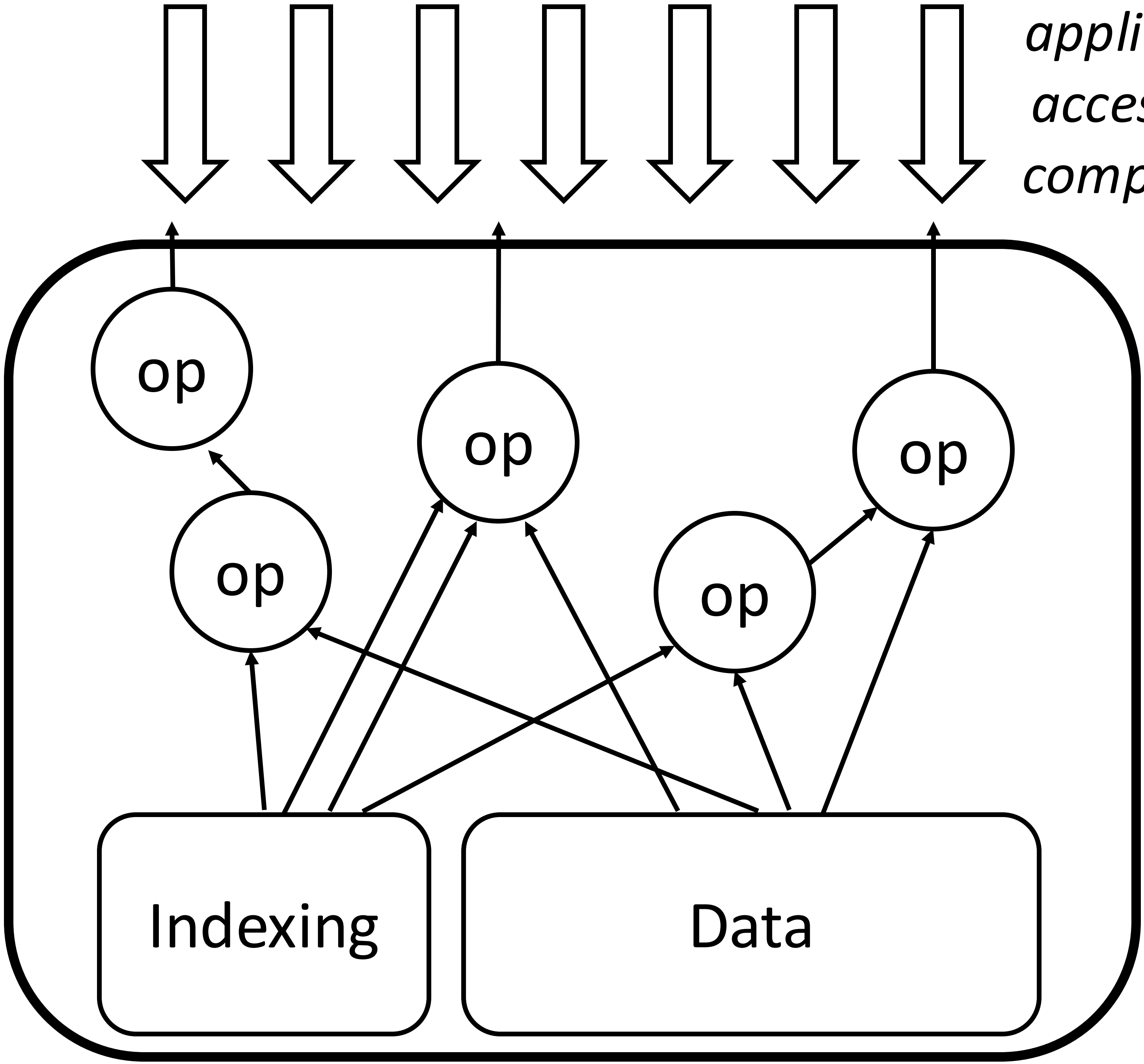
update and **access** them **efficiently**



data system: breaking the blackbox

application/SQL
access patterns
complex queries

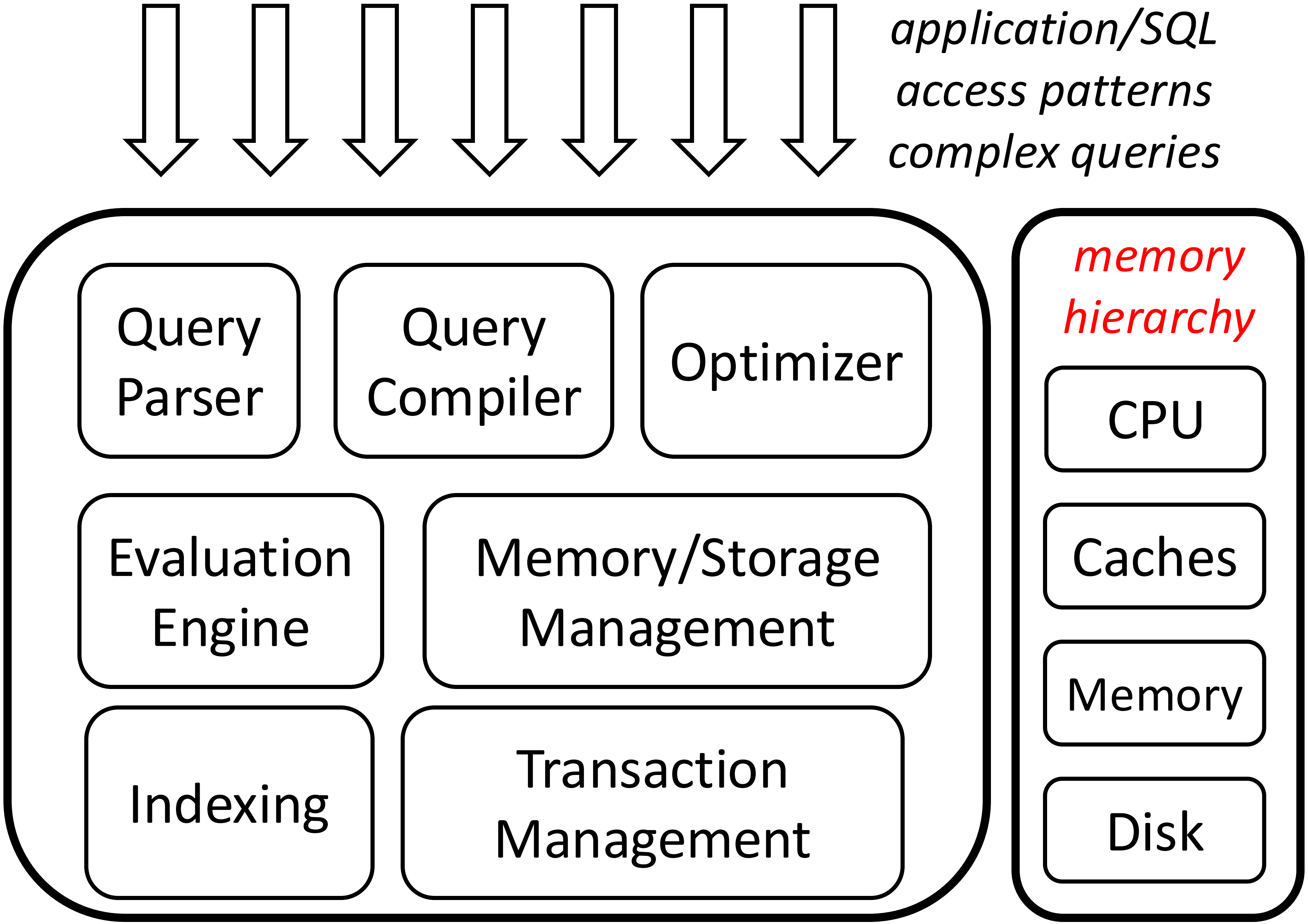
algorithms
&
operators



selection
projection
join
aggregate
hashing
sorting

data &
metadata

application/SQL
access patterns
complex queries



modules

Query Parser

Query Compiler

Optimizer

Evaluation Engine

Memory/Storage Management

Indexing

Transaction Management

memory hierarchy

CPU

Caches

Memory

Disk

growing environment

ORACLE facebook

Microsoft IBM Google

twitter

DB

ACID
large systems
complex
lots of tuning

noSQL

BASE
simple, clean
“just enough”

>$200B by 2020, growing at 11.7% every year

[The Forbes, 2016]

growing environment

DB

ACID
large systems
complex
lots of tuning

noSQL

BASE
simple, clean
“just enough”

ORACLE facebook

Microsoft IBM Google

twitter

>$200B by 2020, growing at 11.7% every year

[The Forbes, 2016]

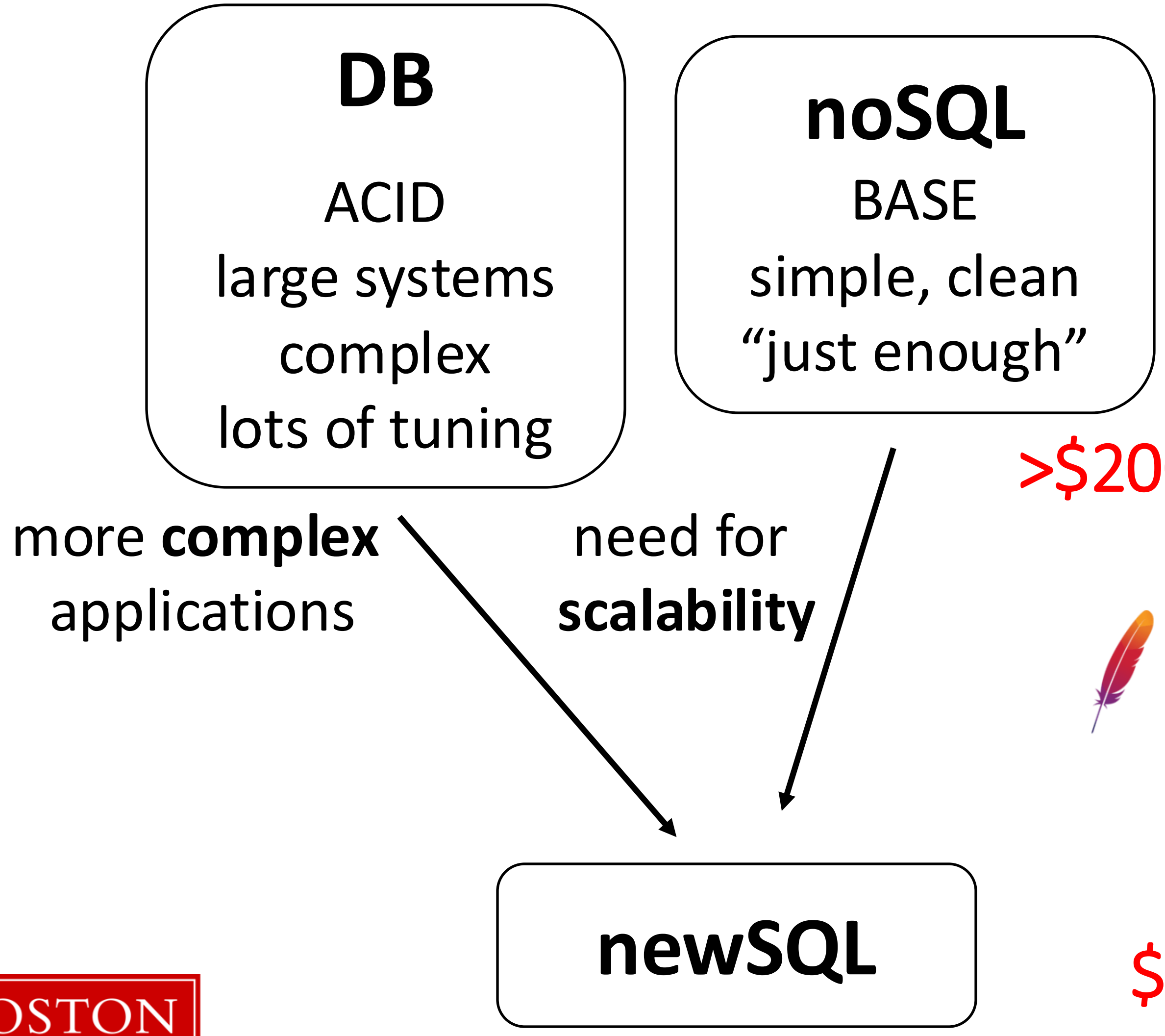
APACHE Cockroach Labs

mongoDB Couchbase SCYLLA

$3B by 2020, growing at 20% every year

[Forrester, 2016]

growing environment



DB

ACID
large systems
complex
lots of tuning

noSQL

BASE
simple, clean
“just enough”

more **complex** applications

need for **scalability**

newSQL

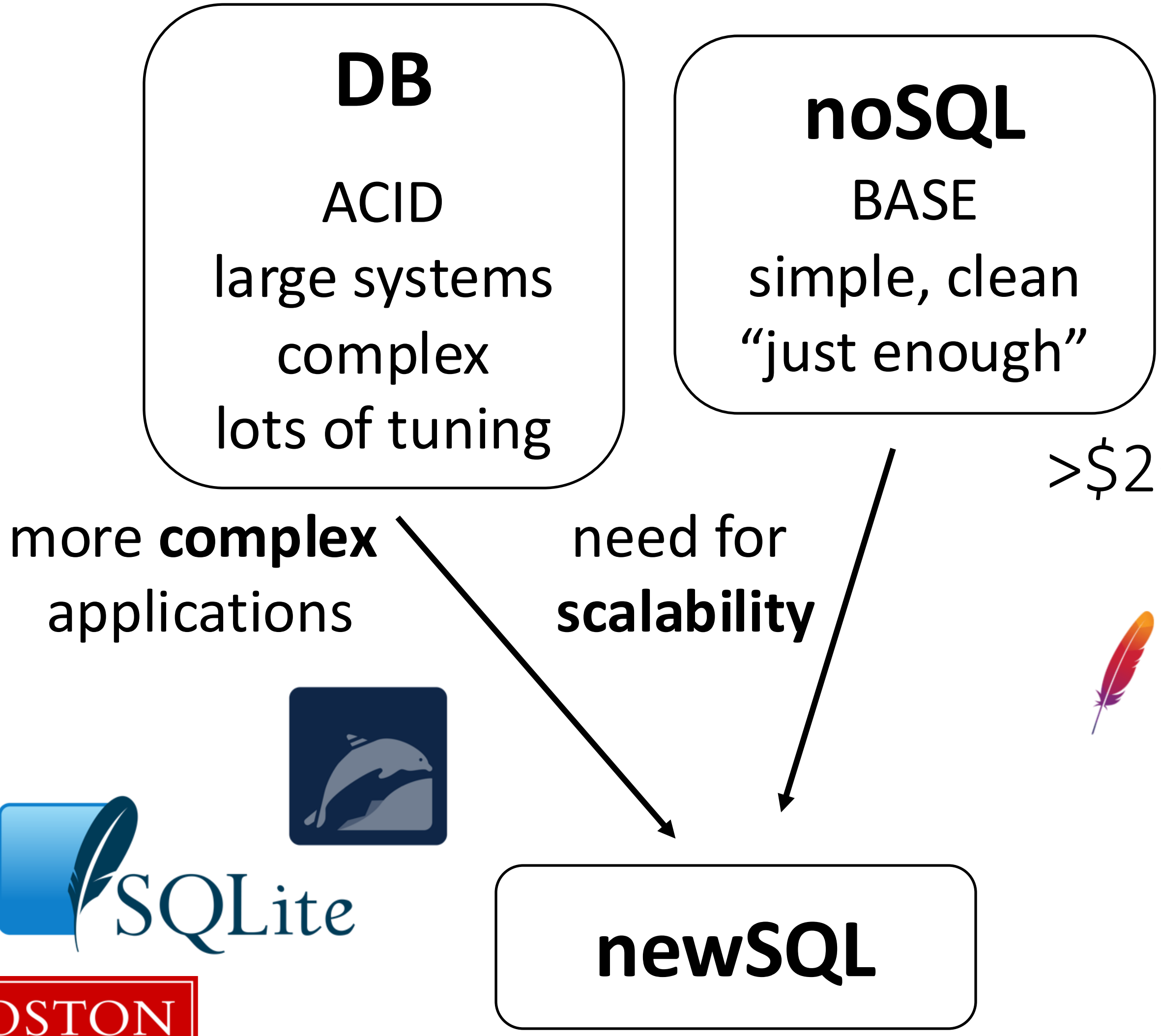
>$200B by 2020, growing at 11.7% every year

[The Forbes, 2016]

$3B by 2020, growing at 20% every year

[Forrester, 2016]

growing environment



DB

ACID
large systems
complex
lots of tuning

noSQL

BASE
simple, clean
“just enough”

more **complex** applications

need for **scalability**

newSQL

>$200B by 2020, growing at 11.7% every year

[The Forbes, 2016]

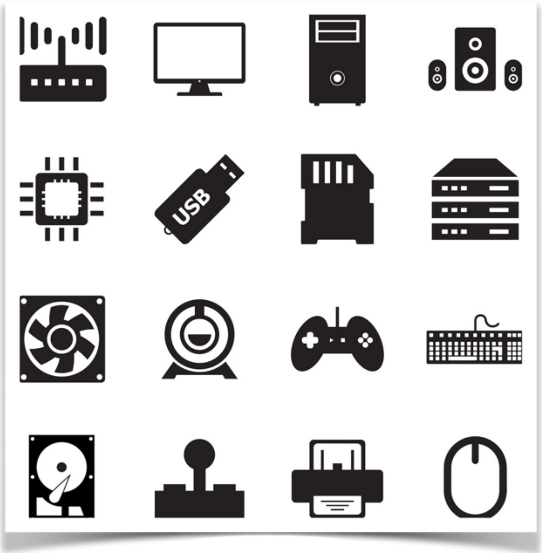
$3B by 2020, growing at 20% every year

[Forrester, 2016]

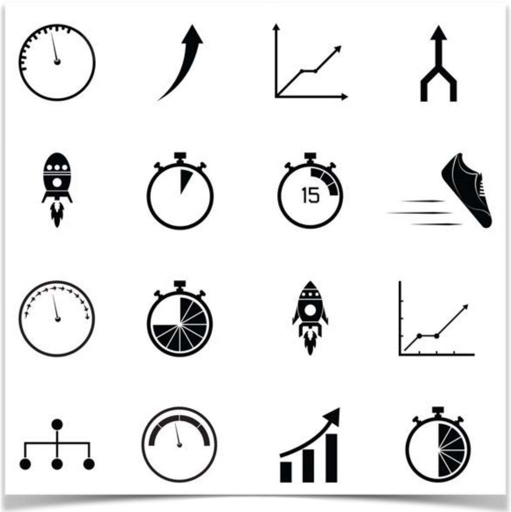
growing need for tailored systems

more data

new hardware



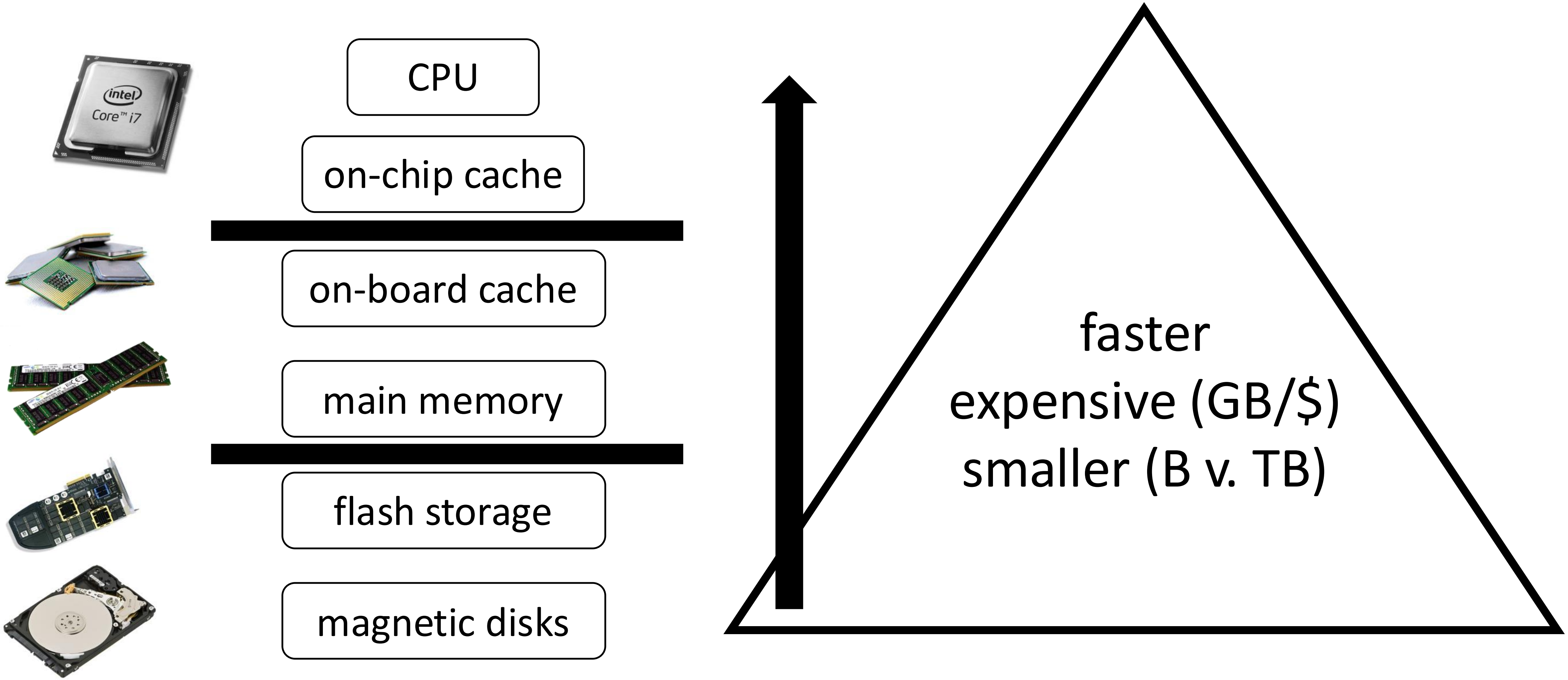
new applications



new performance goals

data systems & the hardware

memory hierarchy



CPU

on-chip cache

on-board cache

main memory

flash storage

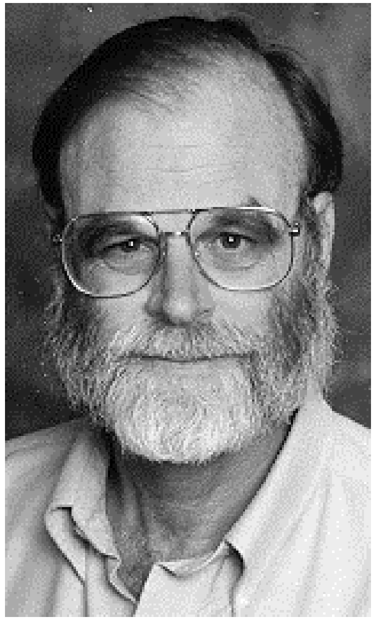
magnetic disks

faster

expensive (GB/$)

smaller (B v. TB)

memory hierarchy (by Jim Gray)



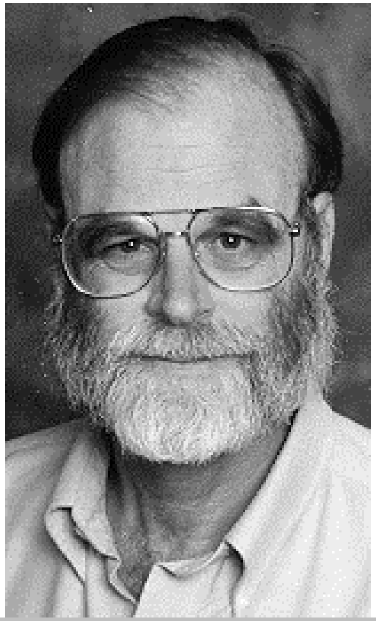
Jim Gray, IBM, Tandem, Microsoft, DEC

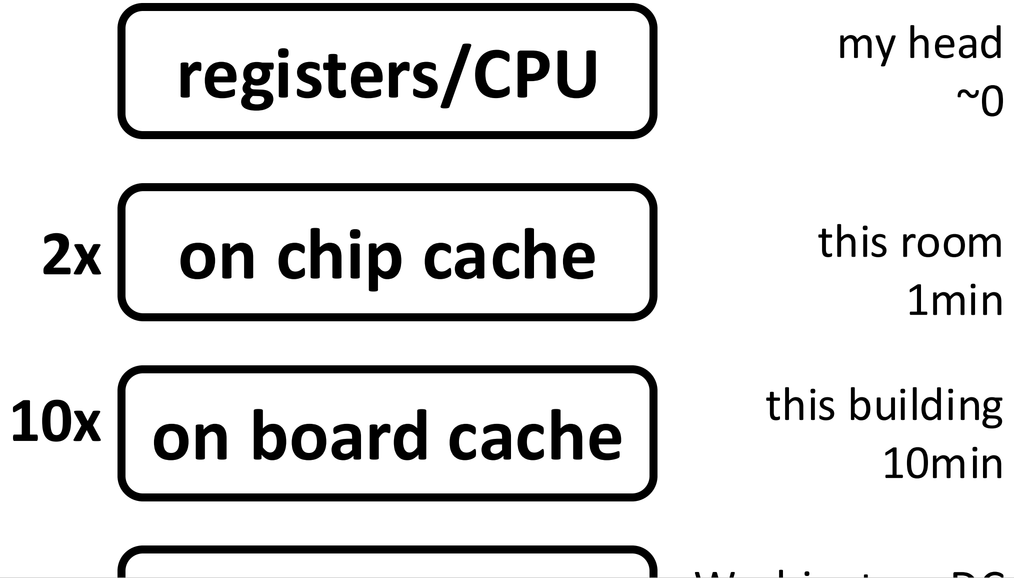
ACM Turing Award 1998

ACM SIGMOD Edgar F. Codd Innovations award 1993

	registers/CPU	my head ~0
2x	**on chip cache**	this room 1min
10x	**on board cache**	this building 10min
100x	**memory**	Washington, DC 5 hours
10^6x	**disk**	Pluto 2 years
10^9x	**tape**	Andromeda 2000 years

memory hierarchy (by Jim Gray)





	registers/CPU	my head ~0
2x	**on chip cache**	this room 1min
10x	**on board cache**	this building 10min

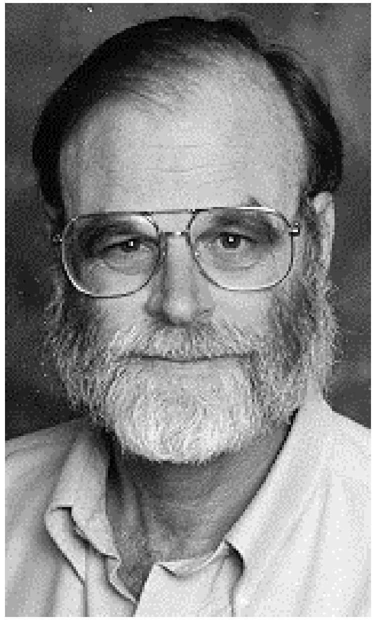
45TB @ $150

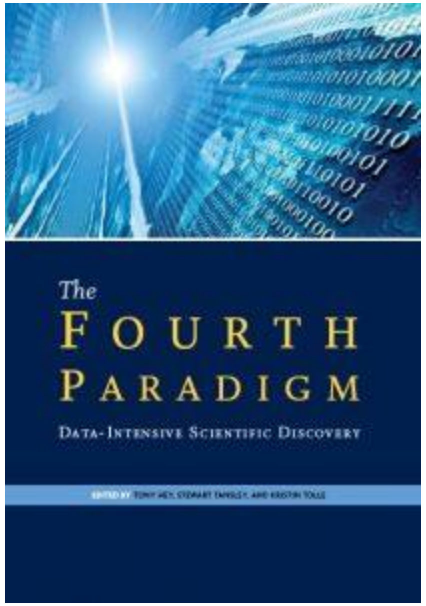
tape?

sequential-only magnetic storage

still a multi-billion industry

Jim Gray (a great scientist and engineer)

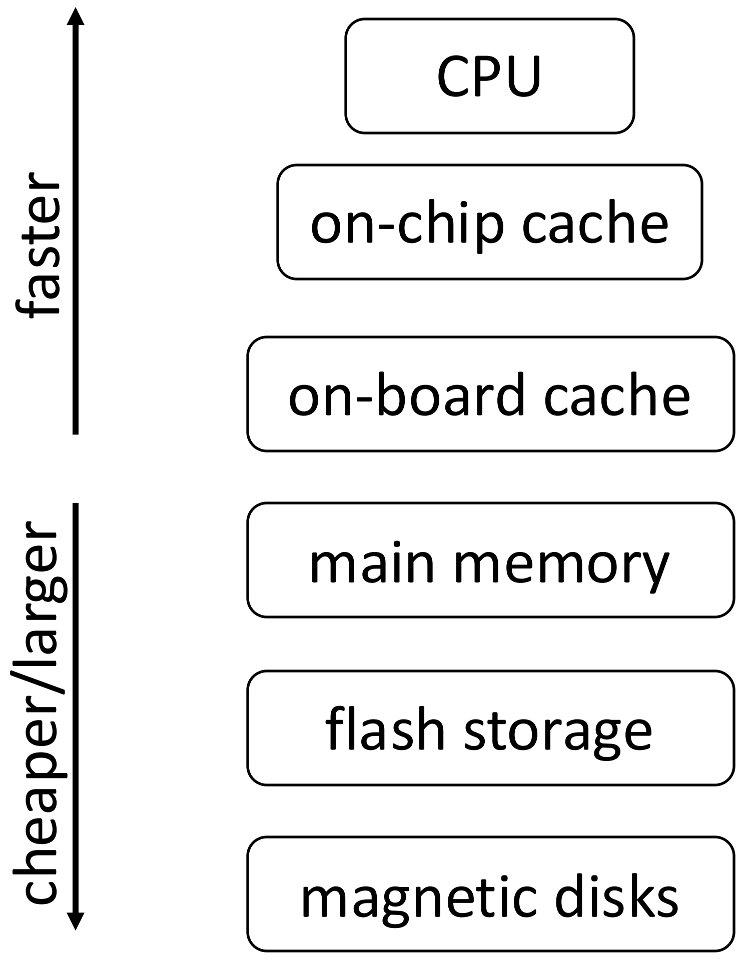



The
FOURTH
PARADIGM
DATA-INTENSIVE SCIENTIFIC DISCOVERY
EDITED BY TONY HEY, STEWART TANSLEY, AND KRISTIN TOLLE

the first collection of
technical visionary research on
a data-intensive scientific discovery

Jim Gray, IBM, Tandem, Microsoft, DEC
ACM Turing Award 1998
ACM SIGMOD Edgar F. Codd Innovations award 1993

memory wall



faster

cheaper/larger

CPU

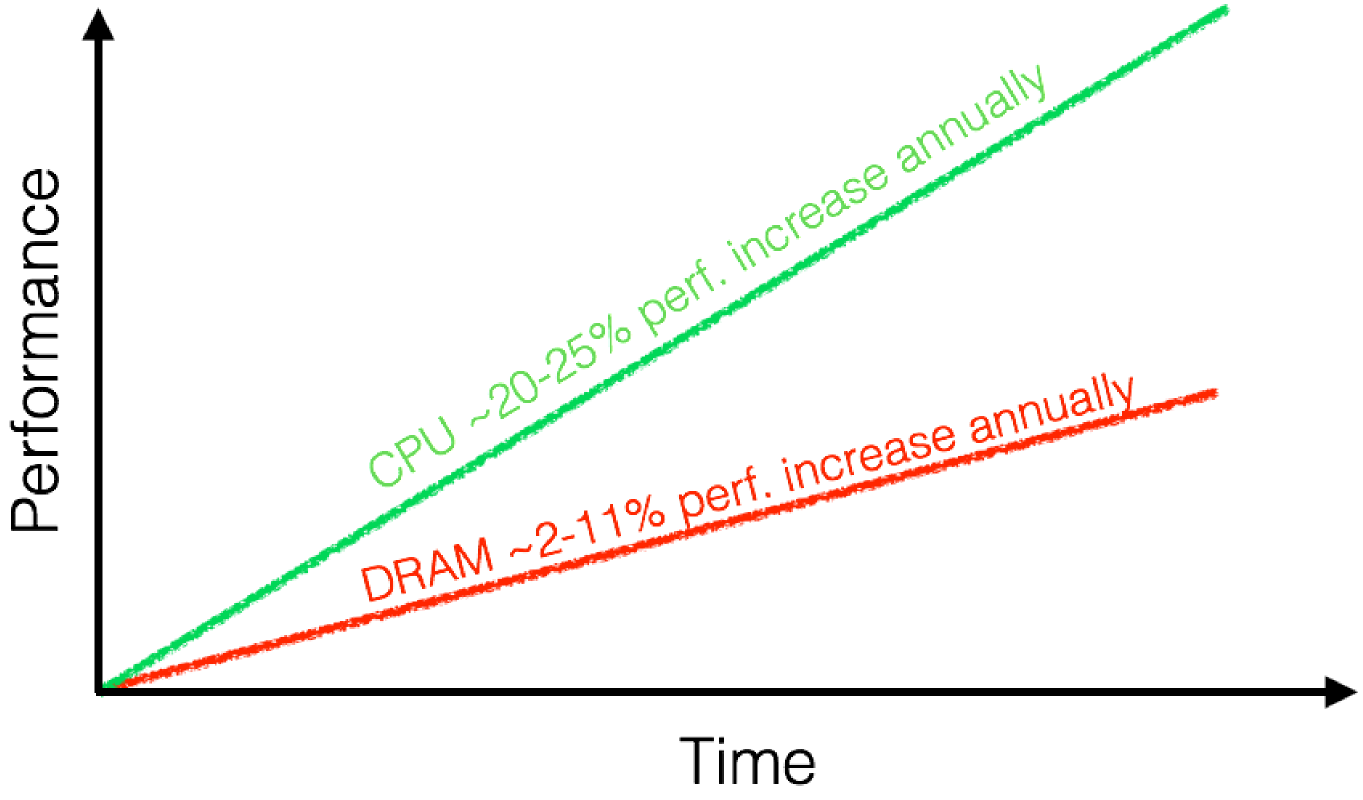
on-chip cache

on-board cache

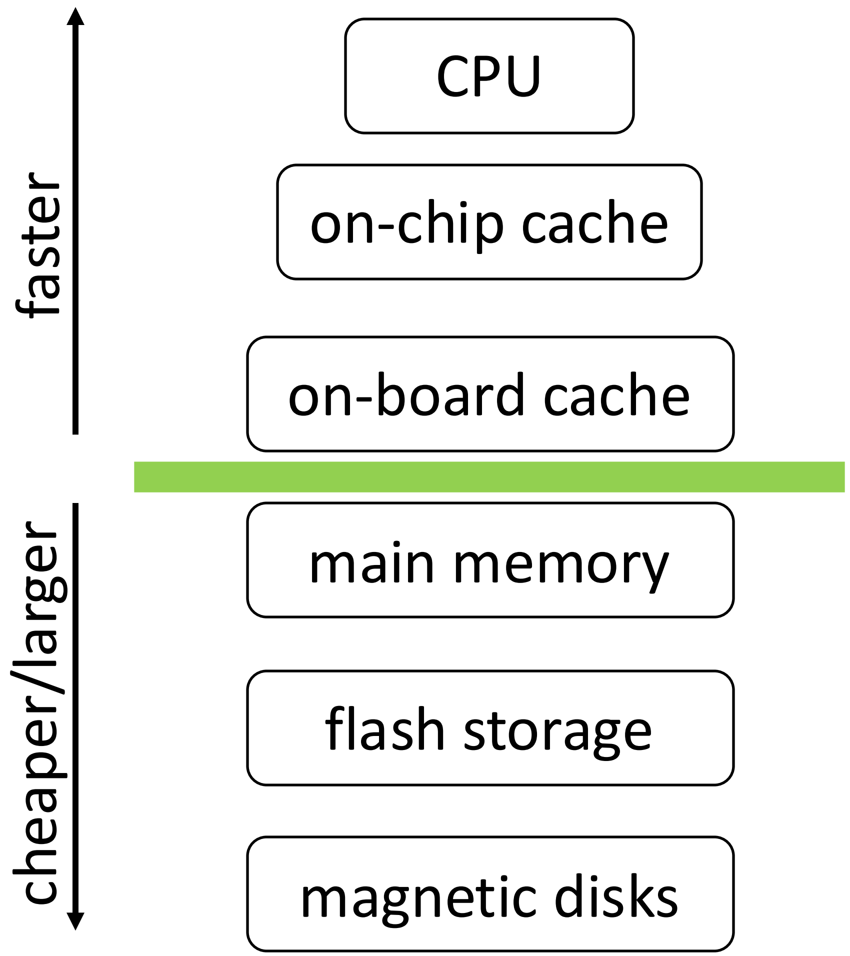
main memory

flash storage

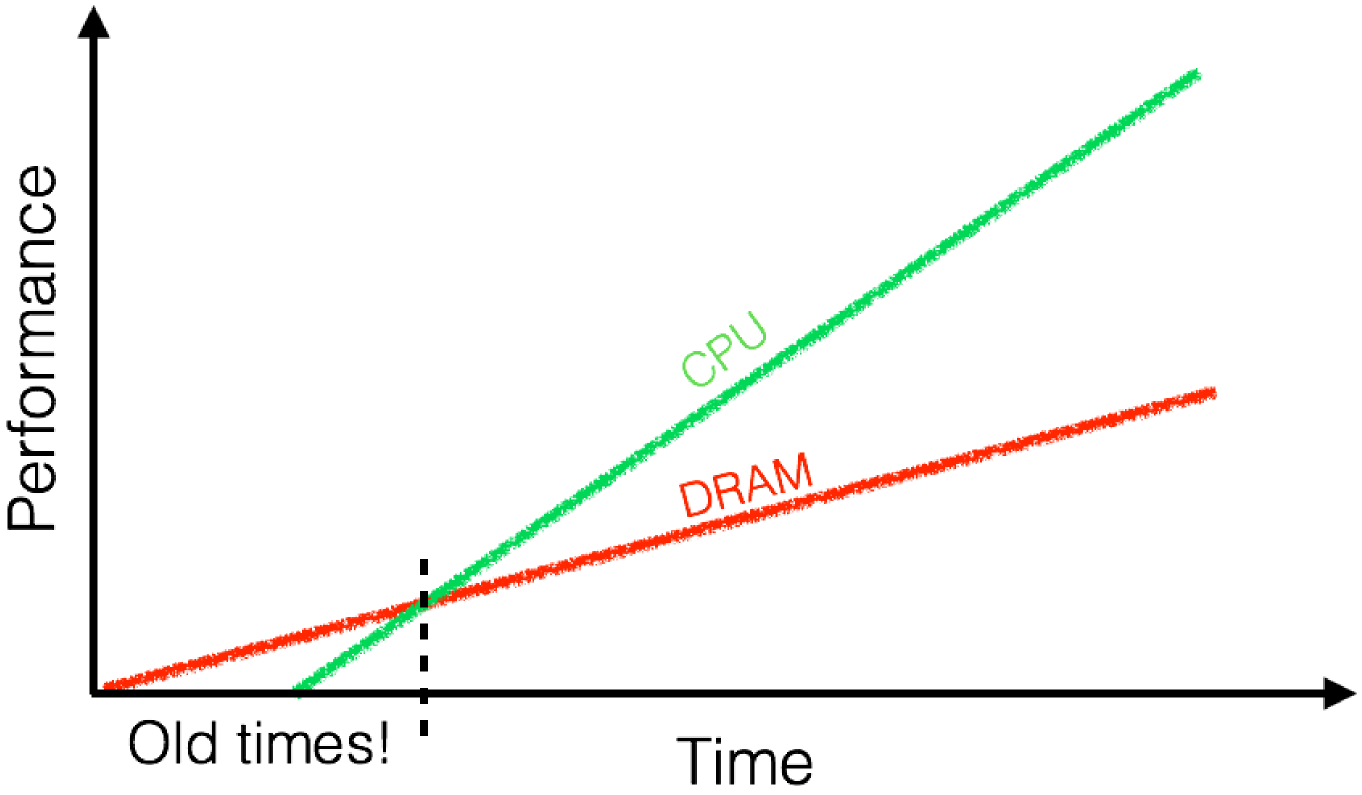
magnetic disks



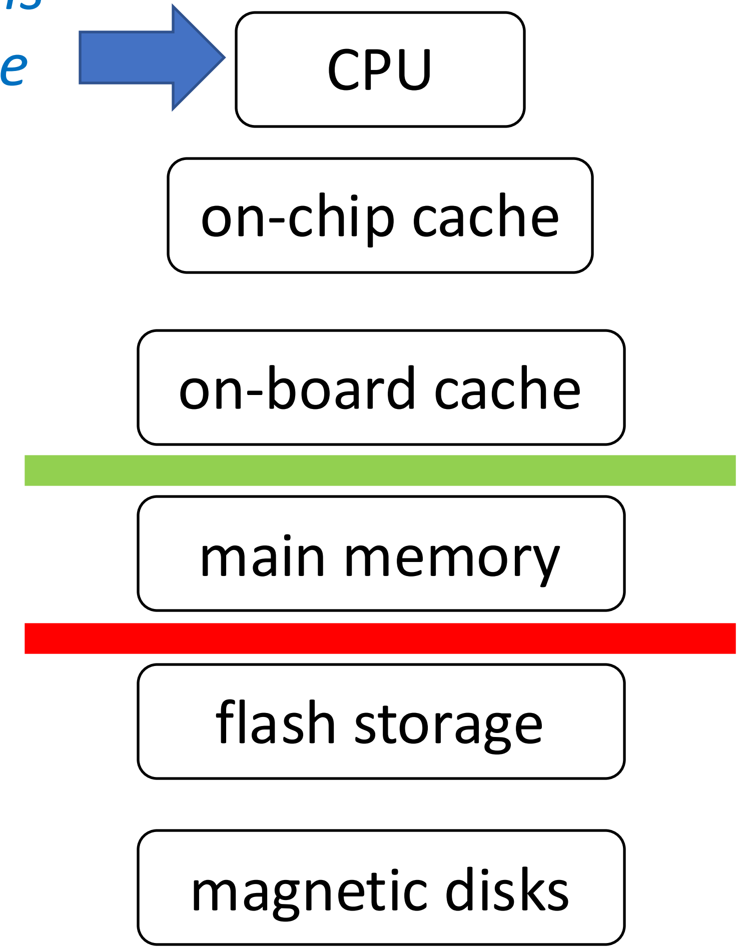
memory wall



be careful when you go below the green line



cache/memory misses



computations happen here

CPU

on-chip cache

on-board cache

main memory

flash storage

magnetic disks

be careful when you go below the green line

cache miss: looking for something that is not in the cache

what happens if I miss?

memory miss: looking for something that is not in memory

what happens if I miss again?

be very careful when you go below the green line

data movement

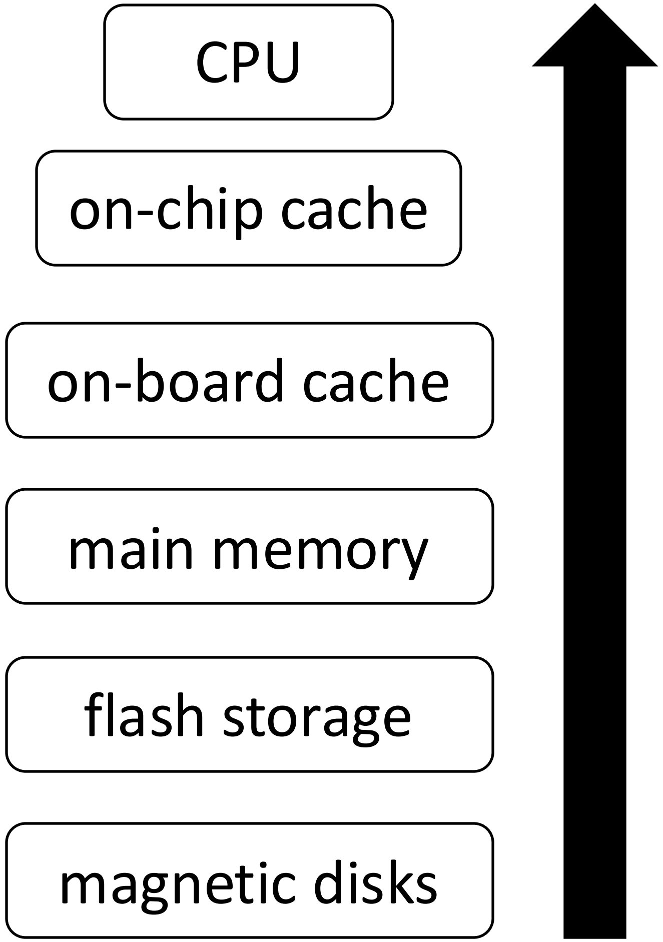
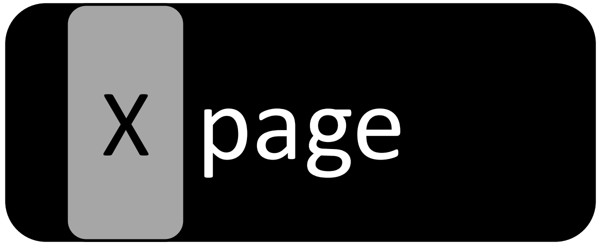


Photo by Gary Dineen/NBAE via Getty Images

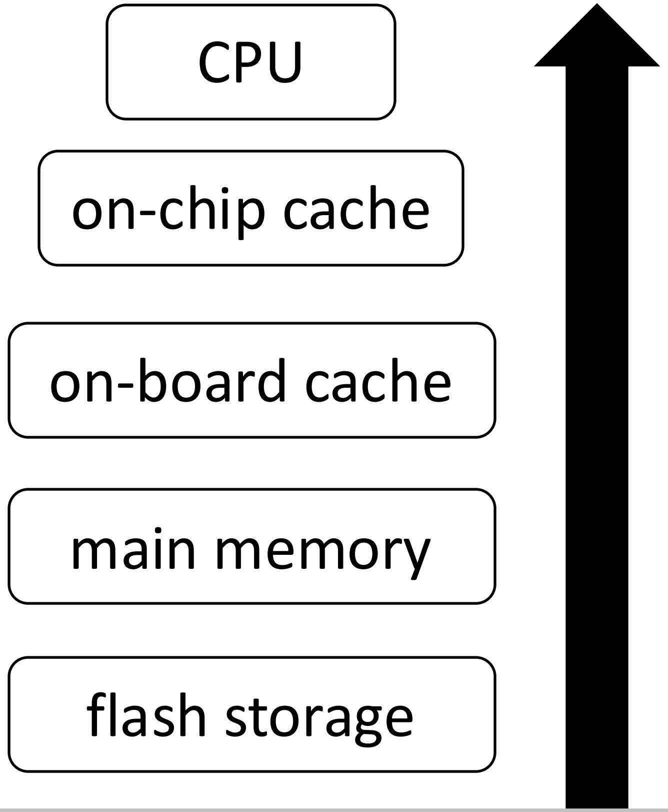
data goes through all necessary levels

also read ***unnecessary*** data

need to read only X
read the whole page



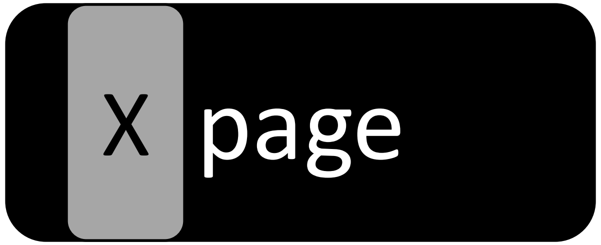
data movement



data goes through
all necessary levels

Photo by Gary Dineen/NBAE via Getty Images

need to read only X
read the whole page

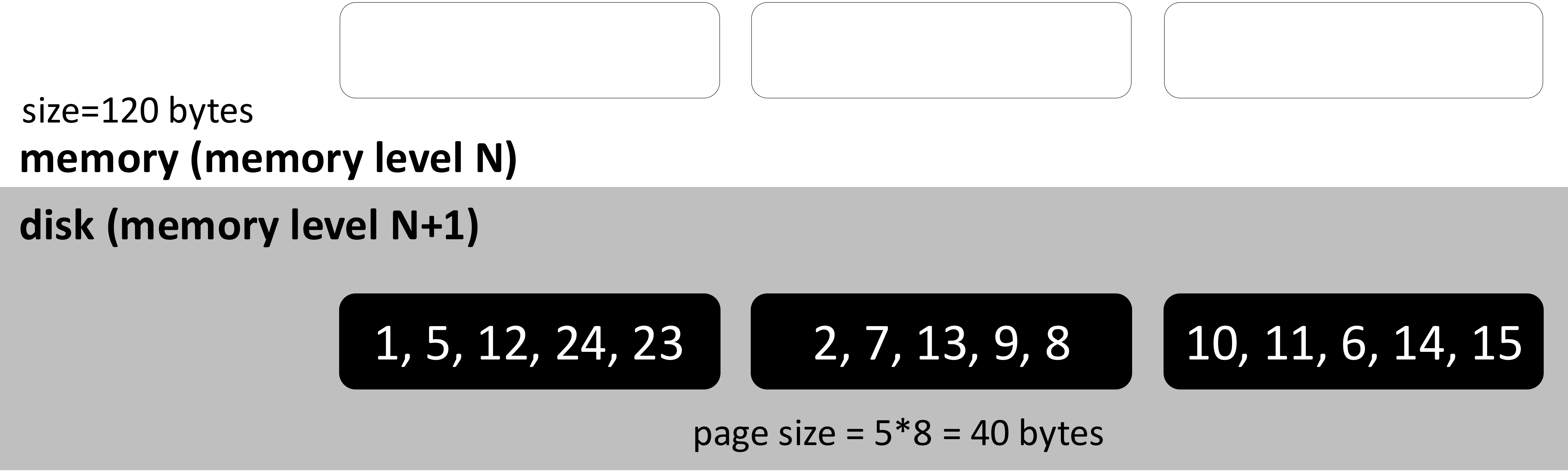


also read
unnecessary data

remember!
disk is millions (mem, hundreds) of times slower than CPU

page-based access & random access

query x<7



size=120 bytes

memory (memory level N)

disk (memory level N+1)

1, 5, 12, 24, 23

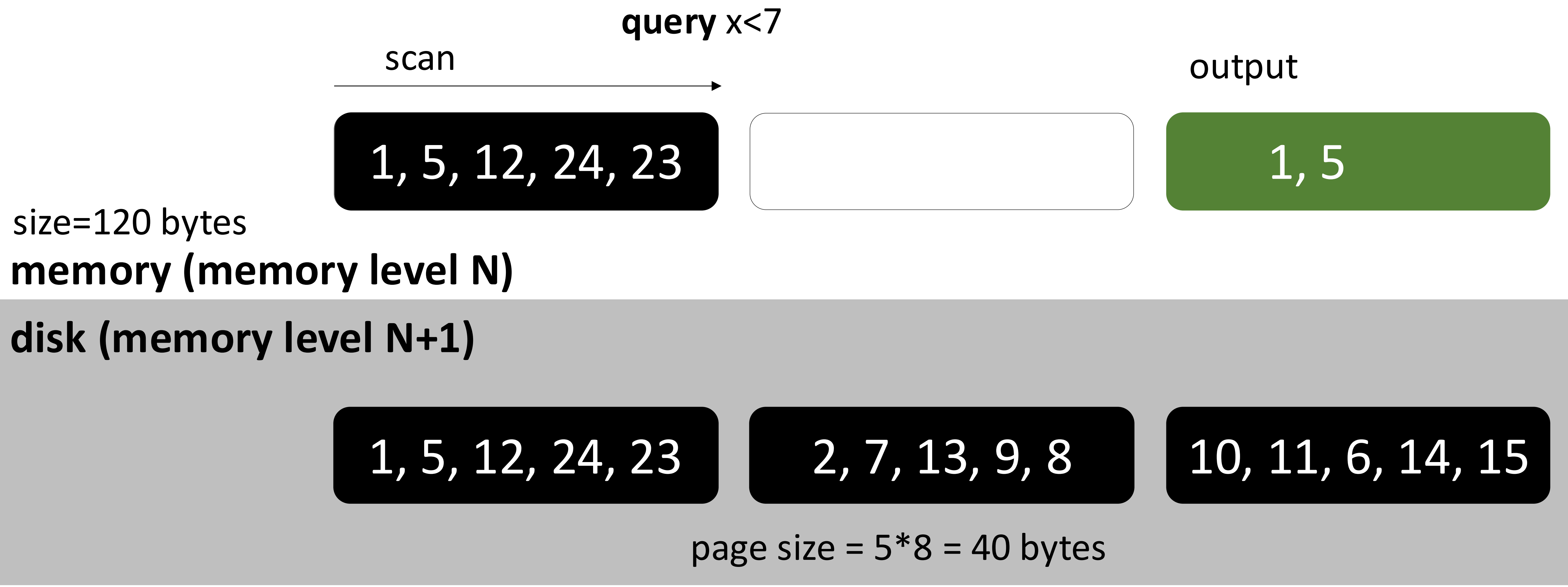
2, 7, 13, 9, 8

10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

$ 40 bytes

page-based access & random access



query x<7

scan

output

1, 5, 12, 24, 23

1, 5

size=120 bytes

memory (memory level N)

disk (memory level N+1)

1, 5, 12, 24, 23

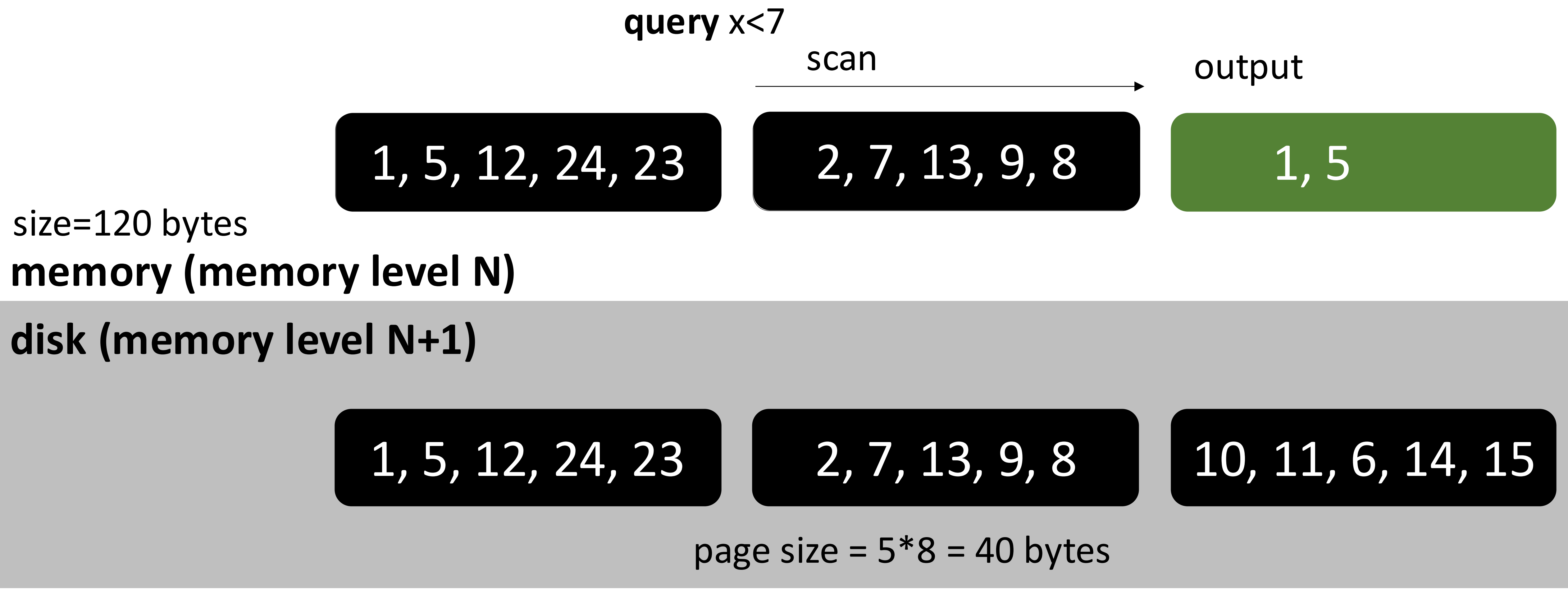
2, 7, 13, 9, 8

10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

$ 40 bytes

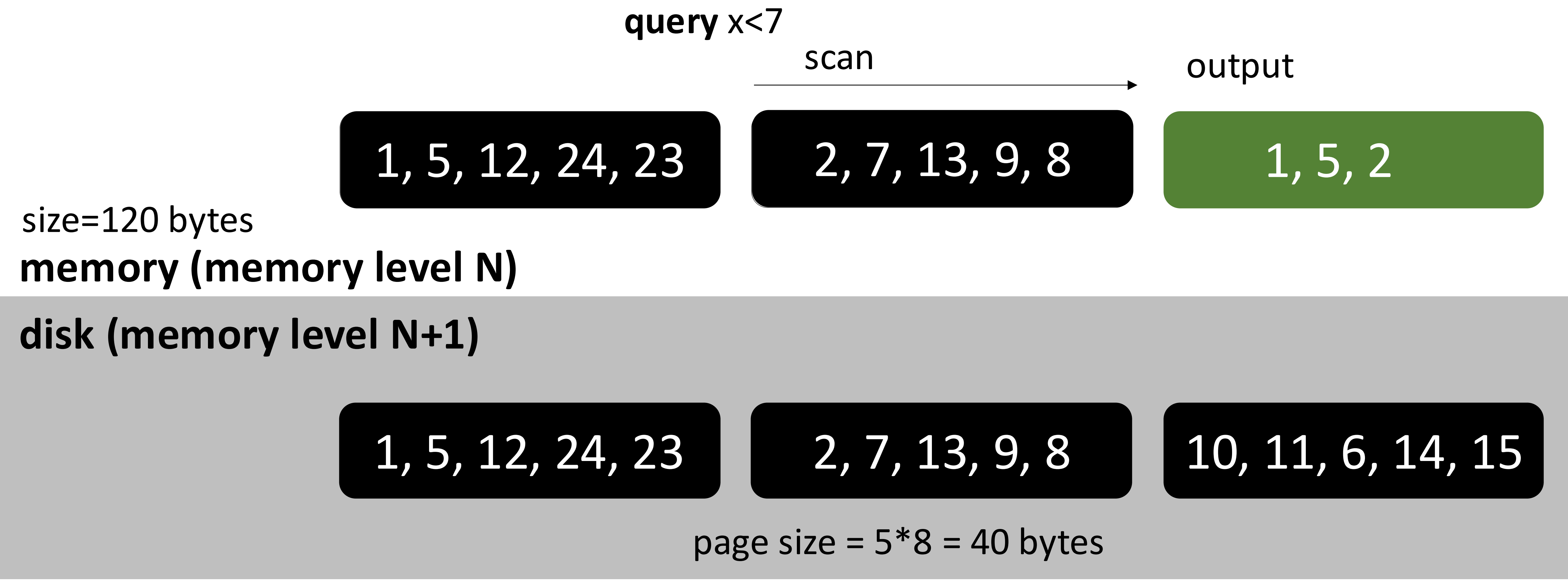
page-based access & random access



query x<7

$ 40 bytes

page-based access & random access



query x<7

scan

output

1, 5, 12, 24, 23

2, 7, 13, 9, 8

1, 5, 2

size=120 bytes

memory (memory level N)

disk (memory level N+1)

1, 5, 12, 24, 23

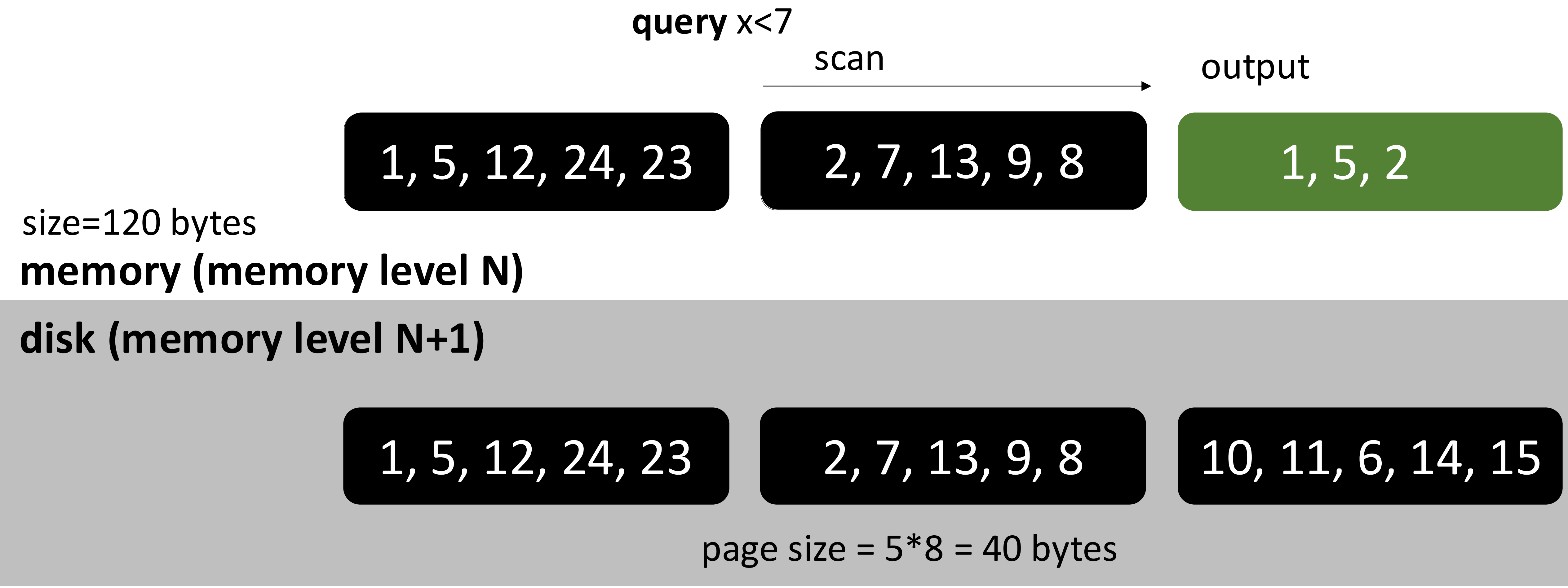
2, 7, 13, 9, 8

10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

$ 80 bytes

page-based access & random access



query x<7

scan

output

1, 5, 12, 24, 23

2, 7, 13, 9, 8

1, 5, 2

size=120 bytes

memory (memory level N)

disk (memory level N+1)

1, 5, 12, 24, 23

2, 7, 13, 9, 8

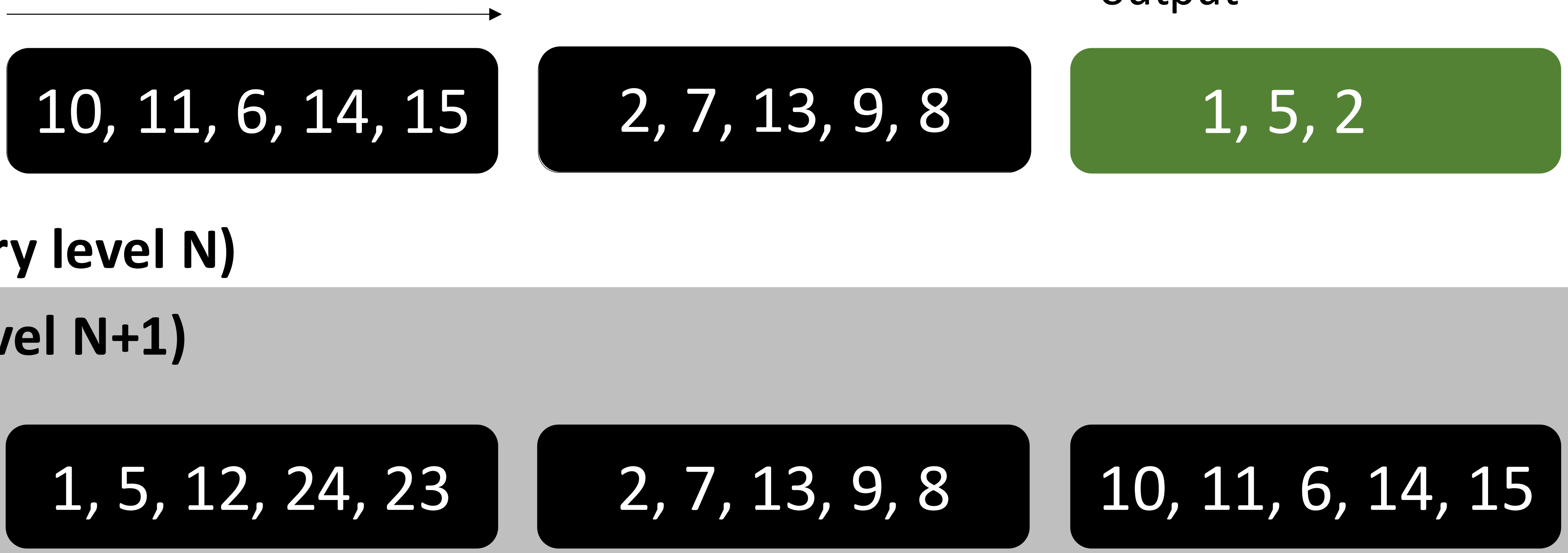
10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

page-based access & random access

$ 80 bytes

query x<7



scan

output

10, 11, 6, 14, 15

2, 7, 13, 9, 8

1, 5, 2

size=120 bytes

memory (memory level N)

disk (memory level N+1)

1, 5, 12, 24, 23

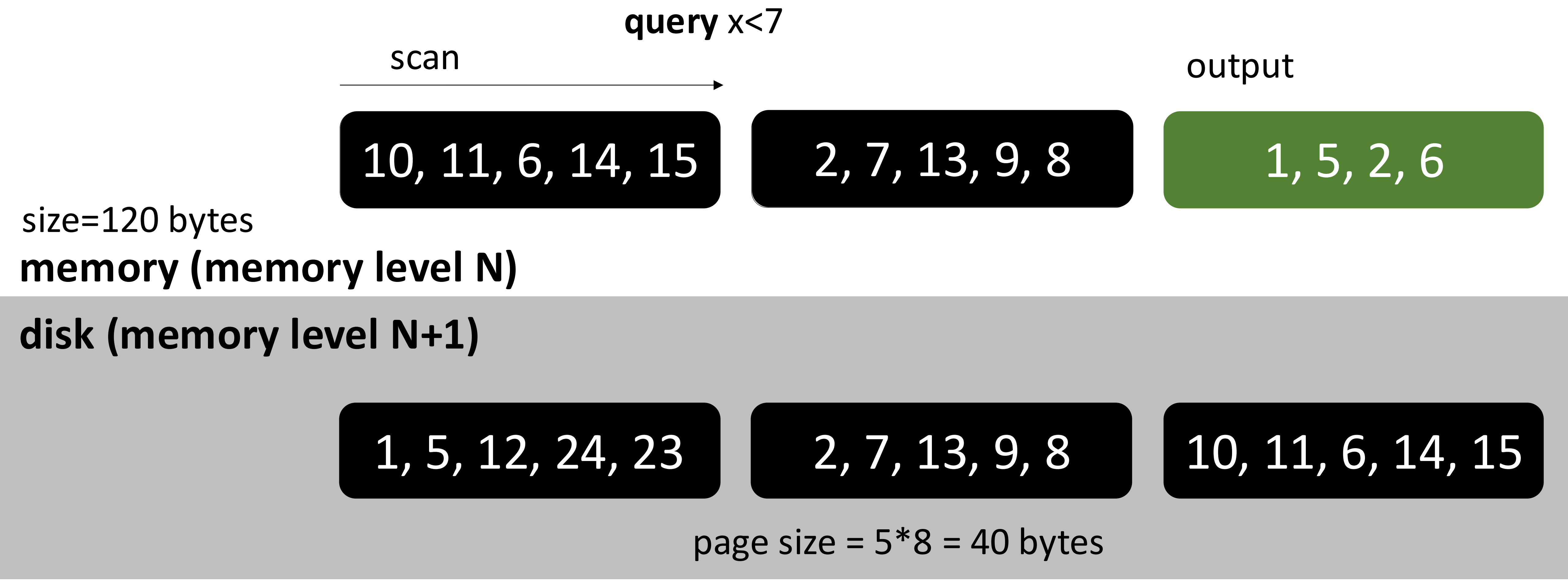
2, 7, 13, 9, 8

10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

$ 80 bytes

page-based access & random access



query x<7

scan

output

10, 11, 6, 14, 15

2, 7, 13, 9, 8

1, 5, 2, 6

size=120 bytes

memory (memory level N)

disk (memory level N+1)

1, 5, 12, 24, 23

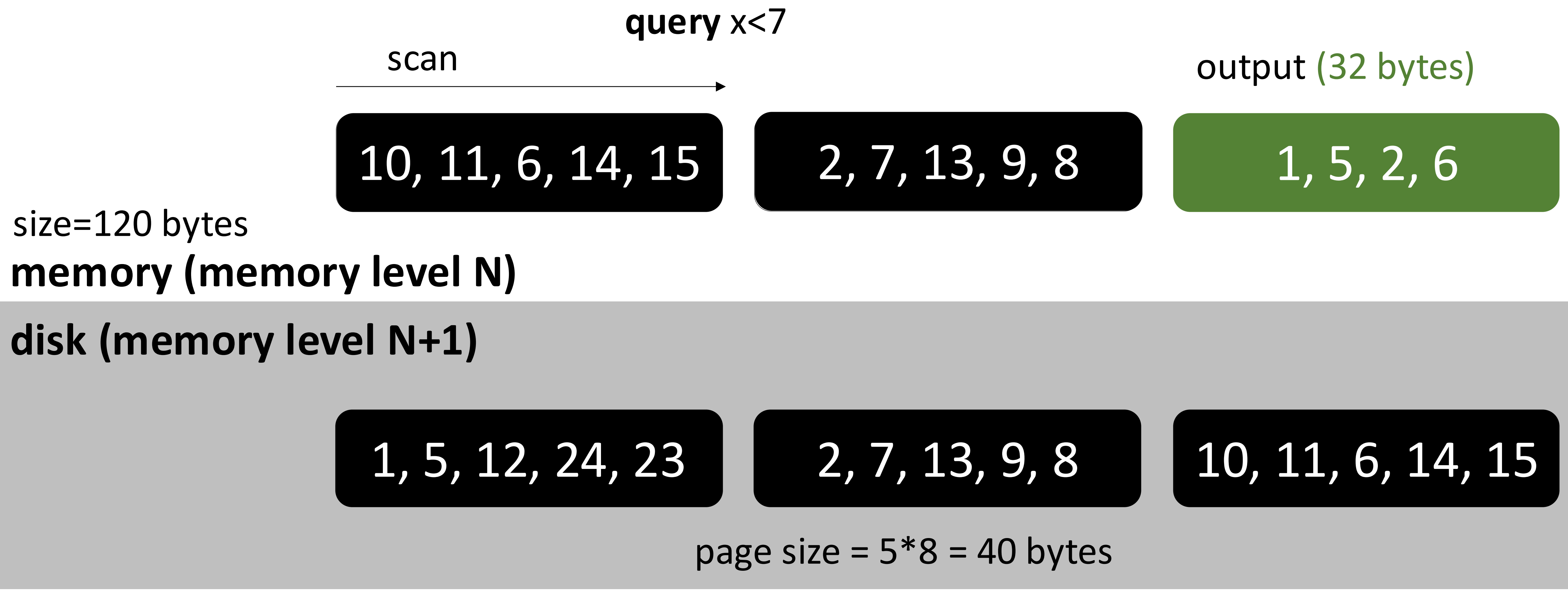
2, 7, 13, 9, 8

10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

$ 120 bytes

page-based access & random access



query x<7

scan

output (32 bytes)

10, 11, 6, 14, 15

2, 7, 13, 9, 8

1, 5, 2, 6

size=120 bytes

memory (memory level N)

disk (memory level N+1)

1, 5, 12, 24, 23

2, 7, 13, 9, 8

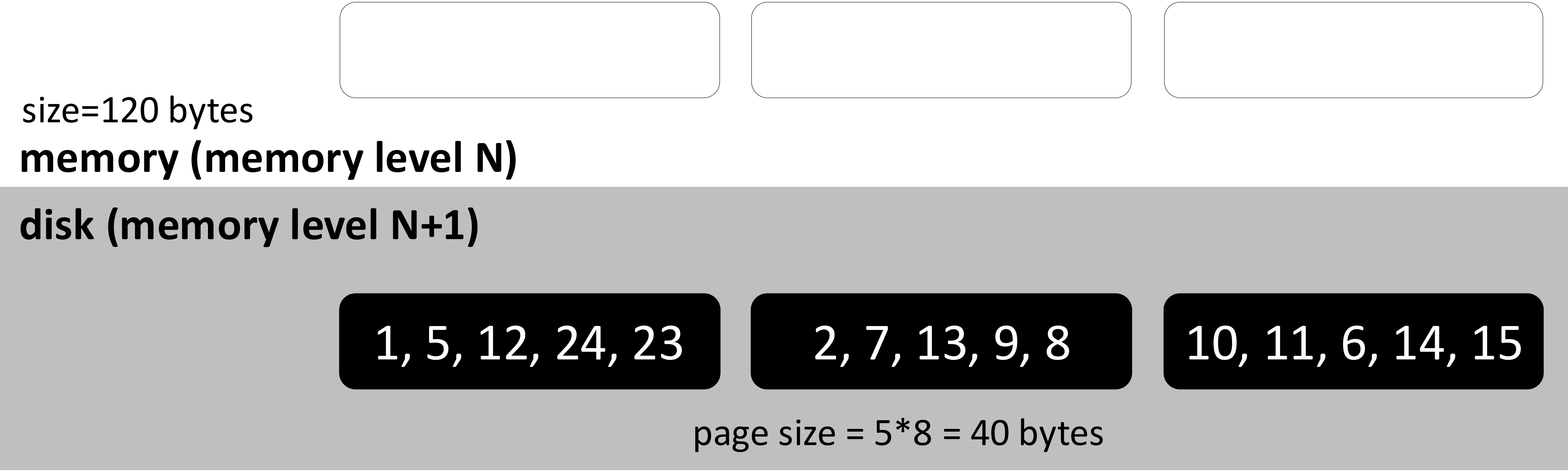
10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

what if we had an oracle (perfect index)?

page-based access & random access

query x<7



size=120 bytes

memory (memory level N)

disk (memory level N+1)

1, 5, 12, 24, 23

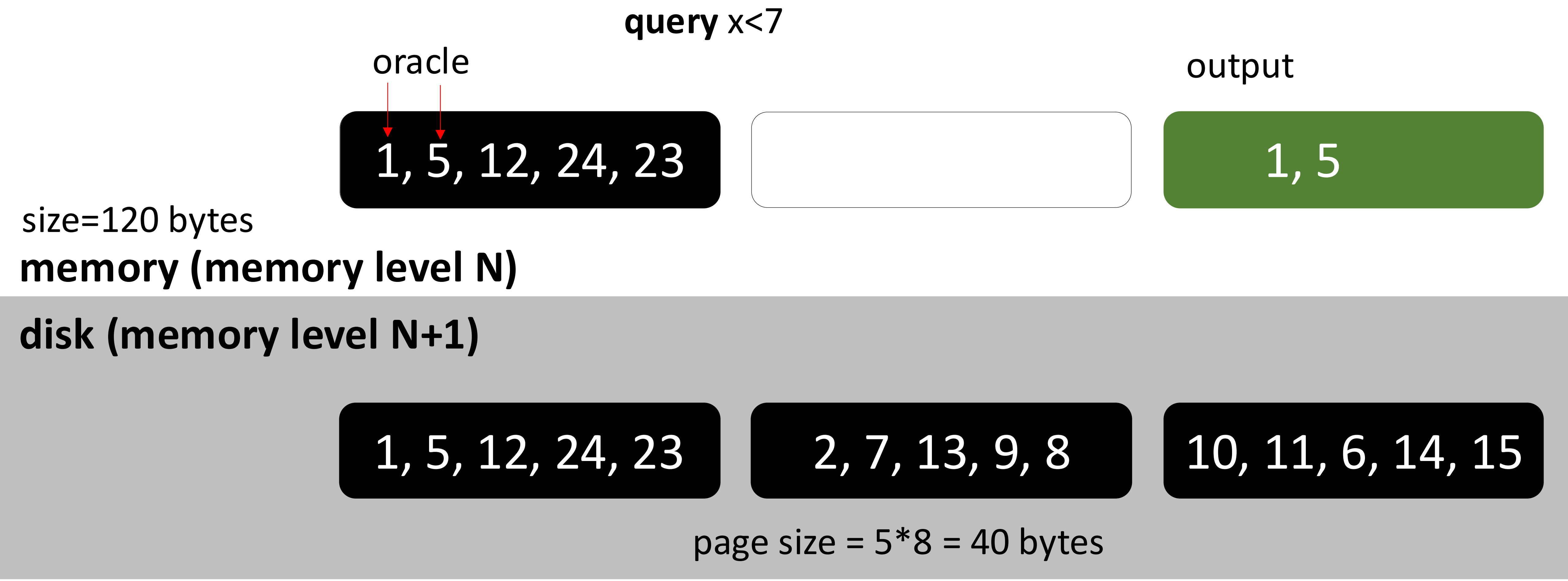
2, 7, 13, 9, 8

10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

$ 40 bytes

page-based access & random access



query x<7

oracle

1, 5, 12, 24, 23

output

1, 5

size=120 bytes

memory (memory level N)

disk (memory level N+1)

1, 5, 12, 24, 23

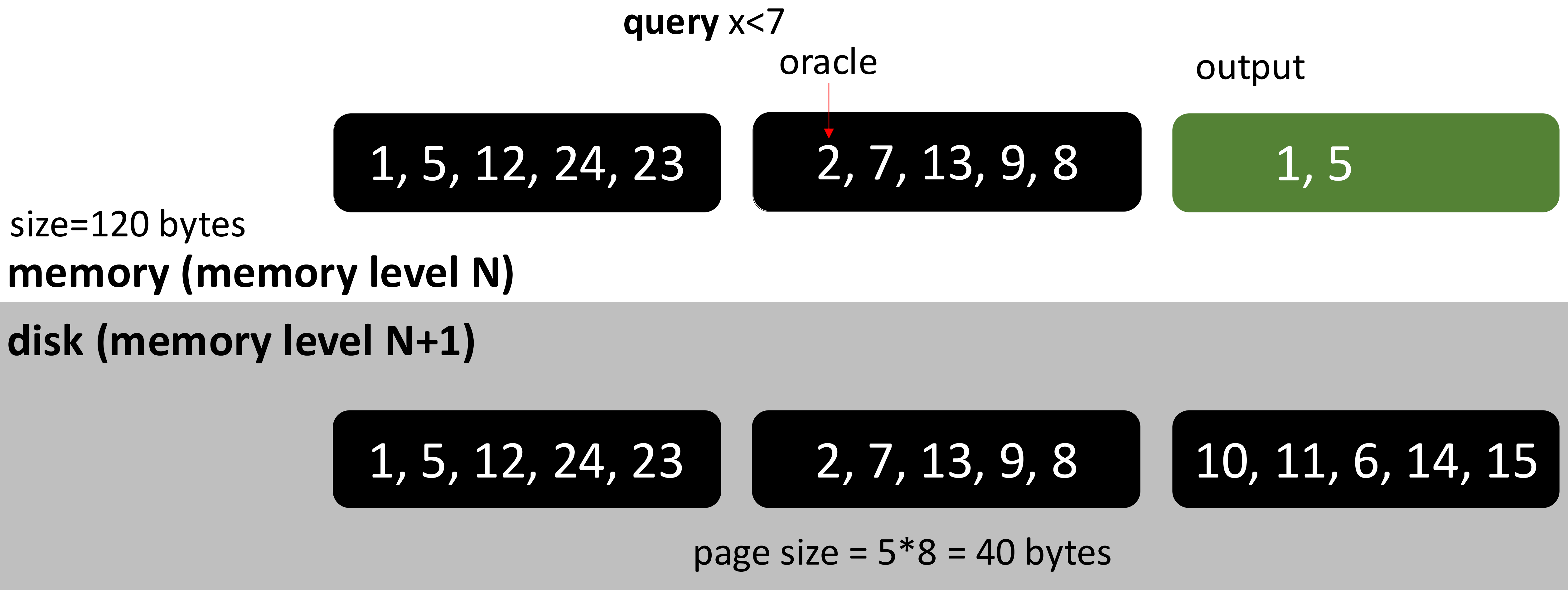
2, 7, 13, 9, 8

10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

$ 40 bytes

page-based access & random access



query x<7

oracle

output

1, 5, 12, 24, 23

2, 7, 13, 9, 8

1, 5

size=120 bytes

memory (memory level N)

disk (memory level N+1)

1, 5, 12, 24, 23

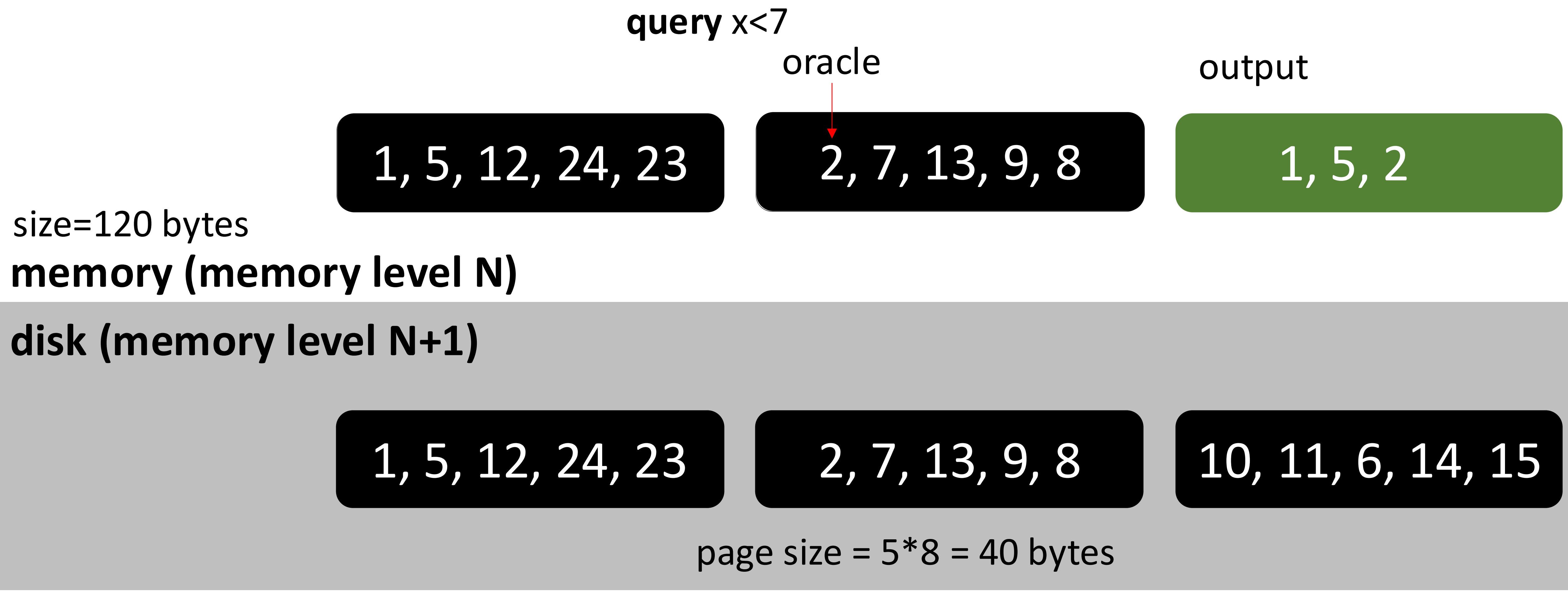
2, 7, 13, 9, 8

10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

$ 40 bytes

page-based access & random access



query x<7

oracle

output

1, 5, 12, 24, 23

2, 7, 13, 9, 8

1, 5, 2

size=120 bytes

memory (memory level N)

disk (memory level N+1)

1, 5, 12, 24, 23

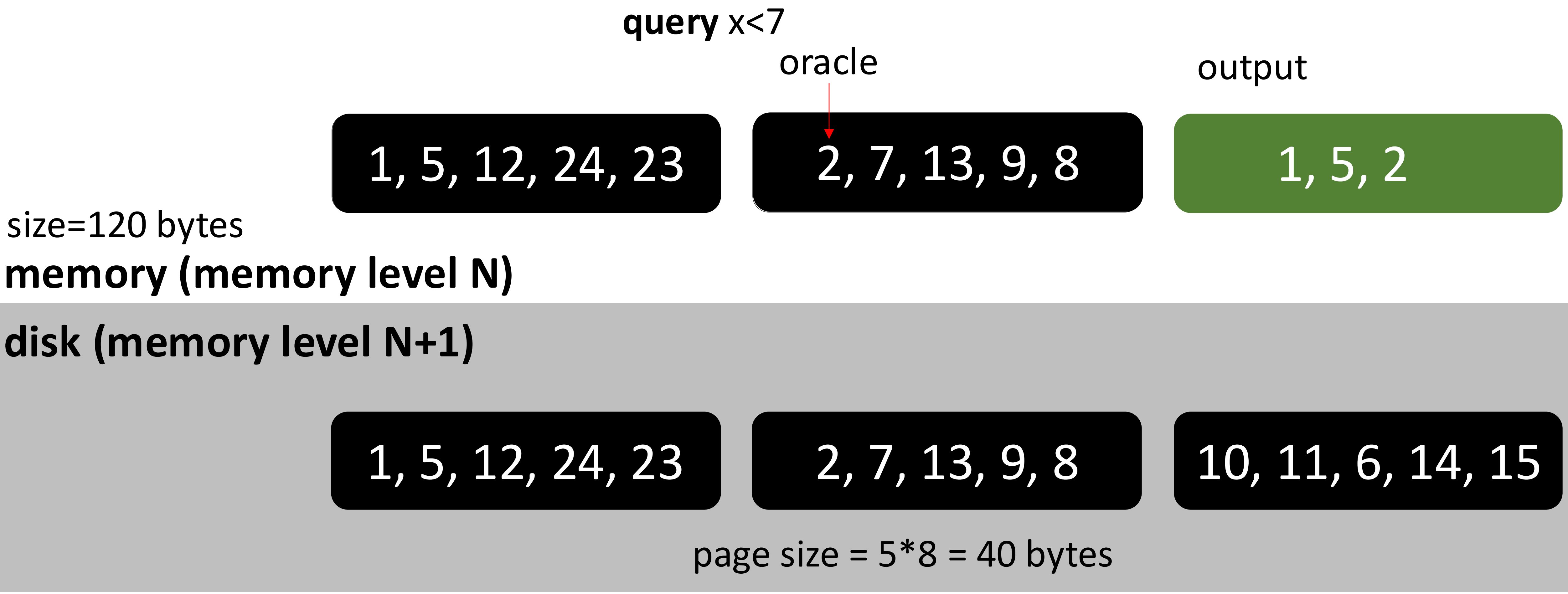
2, 7, 13, 9, 8

10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

page-based access & random access

$ 80 bytes



query x<7

oracle

output

1, 5, 12, 24, 23

2, 7, 13, 9, 8

1, 5, 2

size=120 bytes

memory (memory level N)

disk (memory level N+1)

1, 5, 12, 24, 23

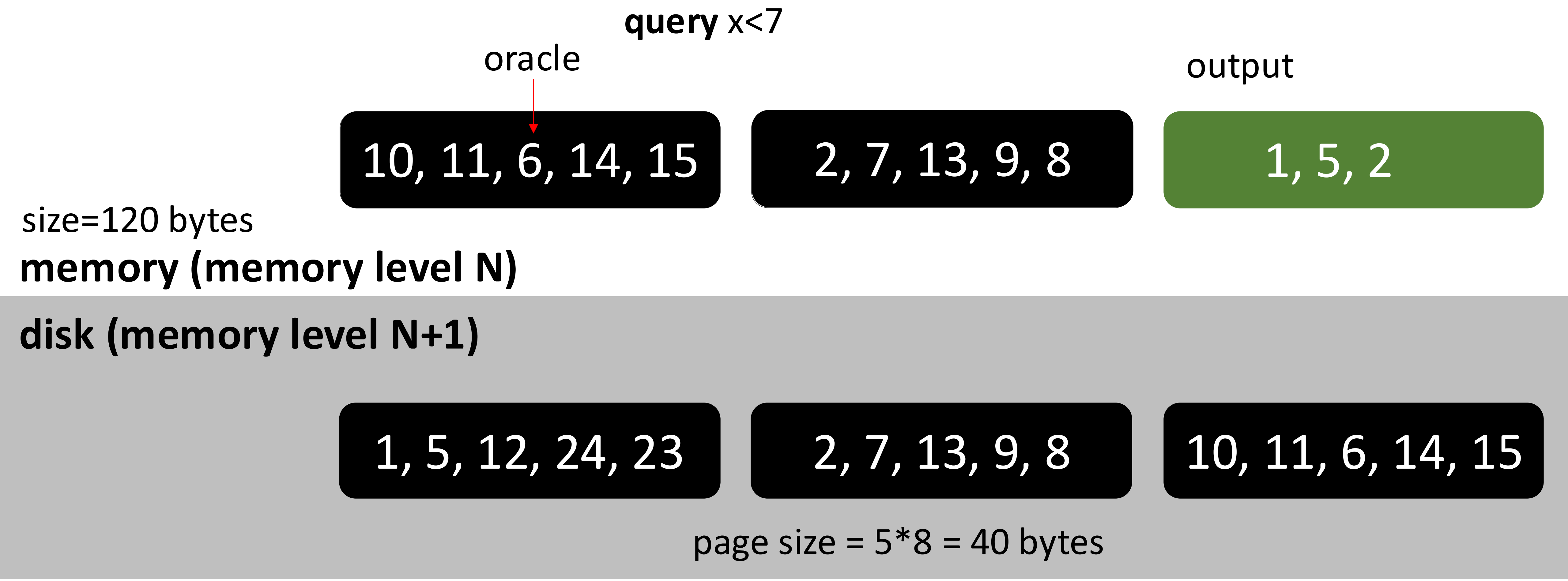
2, 7, 13, 9, 8

10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

page-based access & random access

$ 80 bytes



query x<7

oracle

10, 11, 6, 14, 15

2, 7, 13, 9, 8

output

1, 5, 2

size=120 bytes

memory (memory level N)

disk (memory level N+1)

1, 5, 12, 24, 23

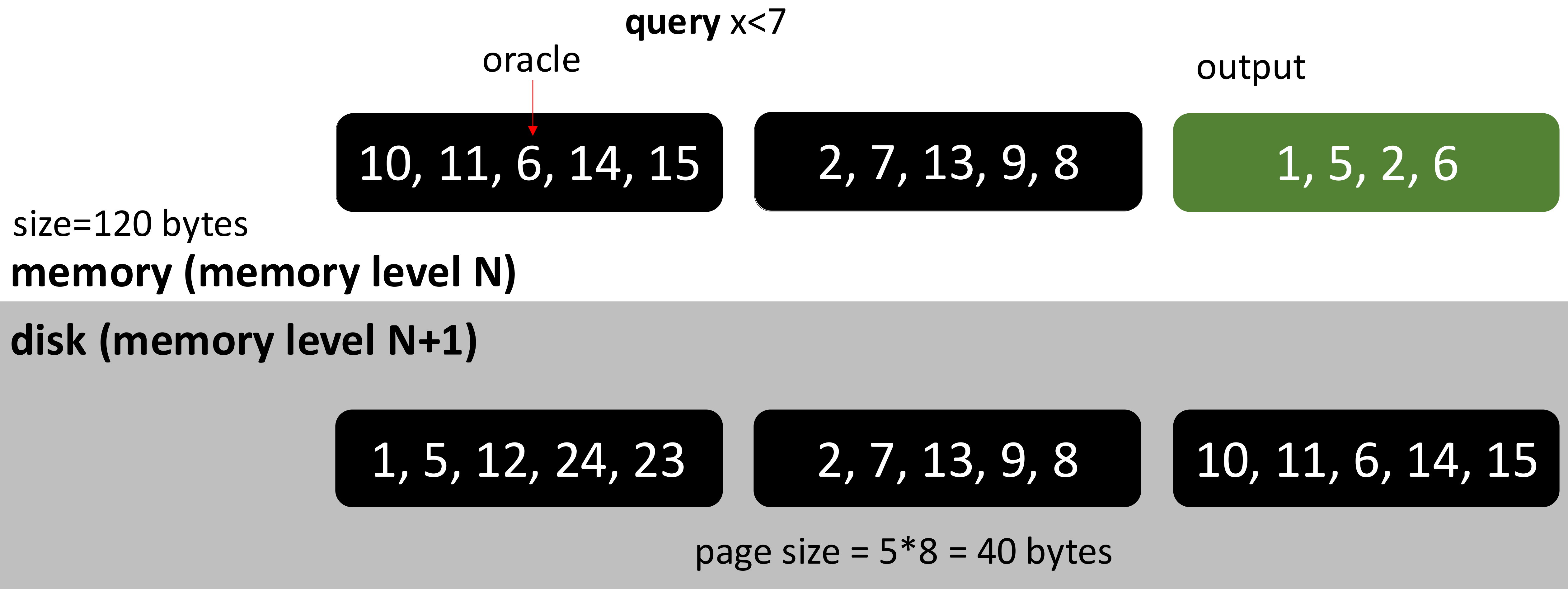
2, 7, 13, 9, 8

10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

page-based access & random access

$ 80 bytes



query x<7

oracle

output

10, 11, 6, 14, 15

2, 7, 13, 9, 8

1, 5, 2, 6

size=120 bytes

memory (memory level N)

disk (memory level N+1)

1, 5, 12, 24, 23

2, 7, 13, 9, 8

10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

page-based access & random access

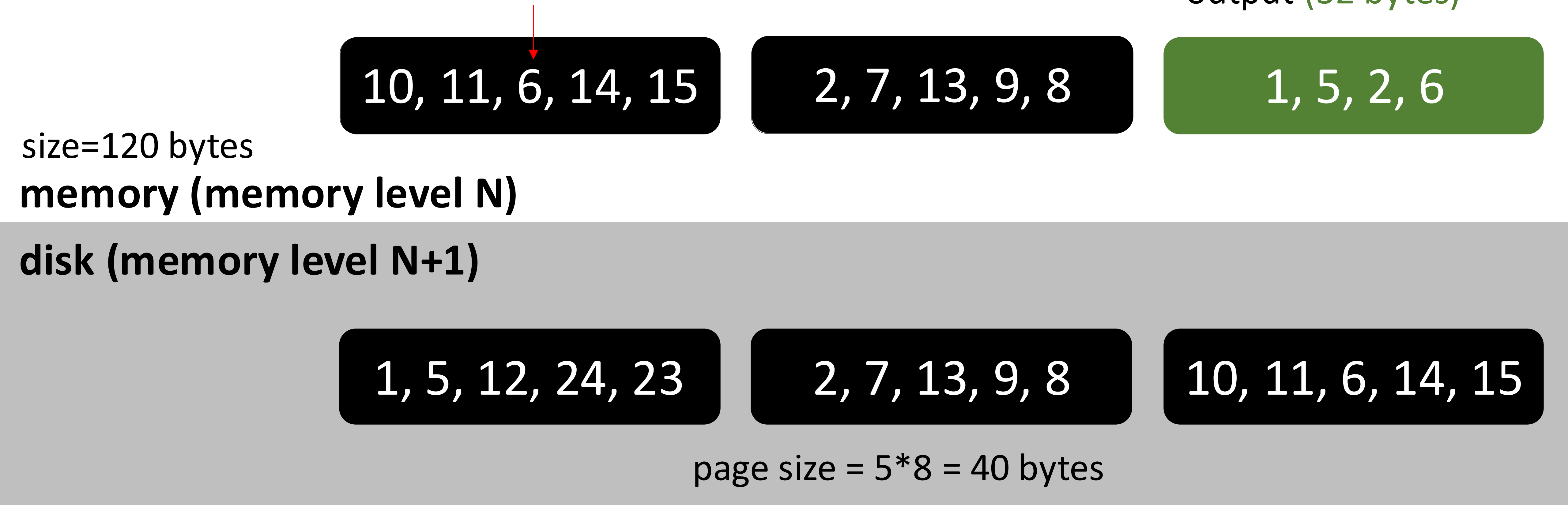
$ 120 bytes

query x<7

was the oracle helpful?

oracle

output (32 bytes)



10, 11, 6, 14, 15

2, 7, 13, 9, 8

1, 5, 2, 6

size=120 bytes

memory (memory level N)

disk (memory level N+1)

1, 5, 12, 24, 23

2, 7, 13, 9, 8

10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

BOSTON UNIVERSITY

when is the oracle helpful?

for which query would an oracle help us?

how to decide whether to use the oracle or not?

1, 5, 12, 24, 23

2, 7, 13, 9, 8

10, 11, 6, 14, 15

every **byte** counts

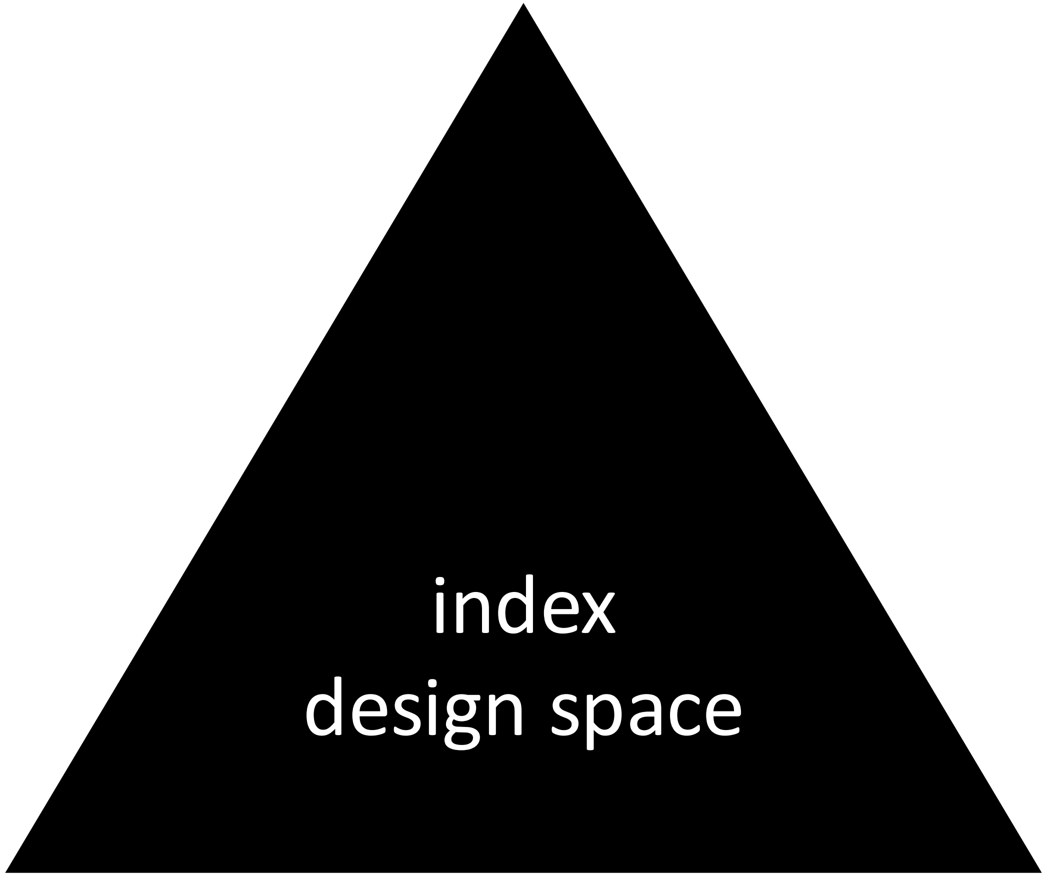
overheads and tradeoffs

how we store data

layouts, indexes

know the **query**

access path selection



index
design space

rules of thumb

sequential access

read one block; consume it completely; discard it; read next

hardware can predict and start prefetching

prefetching can exploit full memory/disk bandwidth

random access

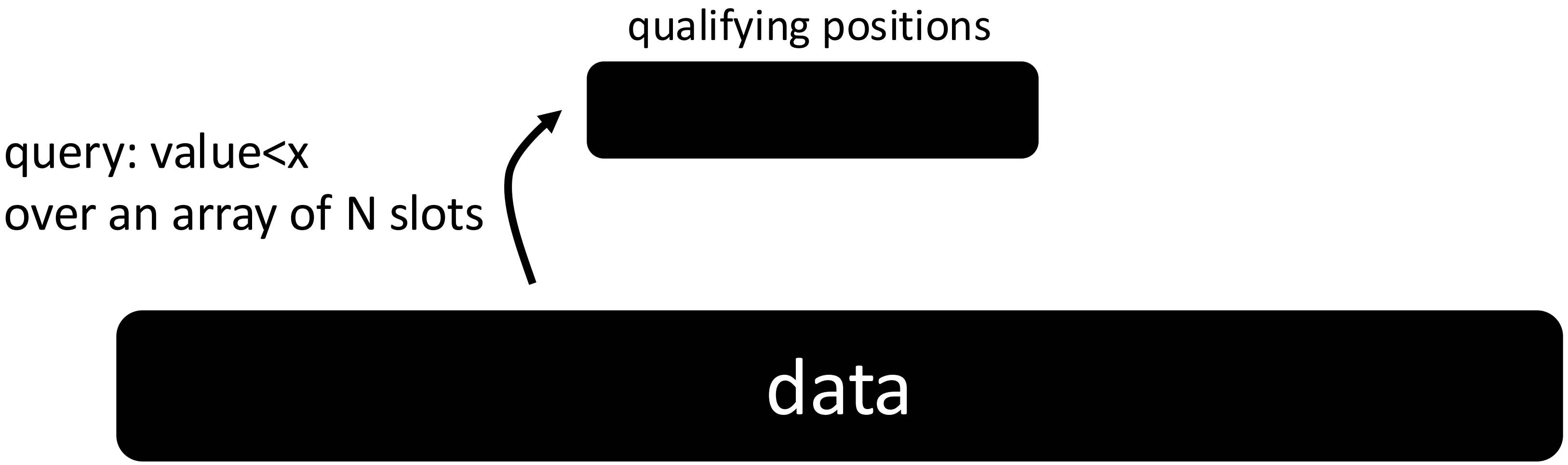
read one block; consume it partially; discard it; (may re-use)

are random accesses always bad?

the one that helps us **avoid a large number of accesses** (random or sequential)

a “simple” database operator

select operator (scan)



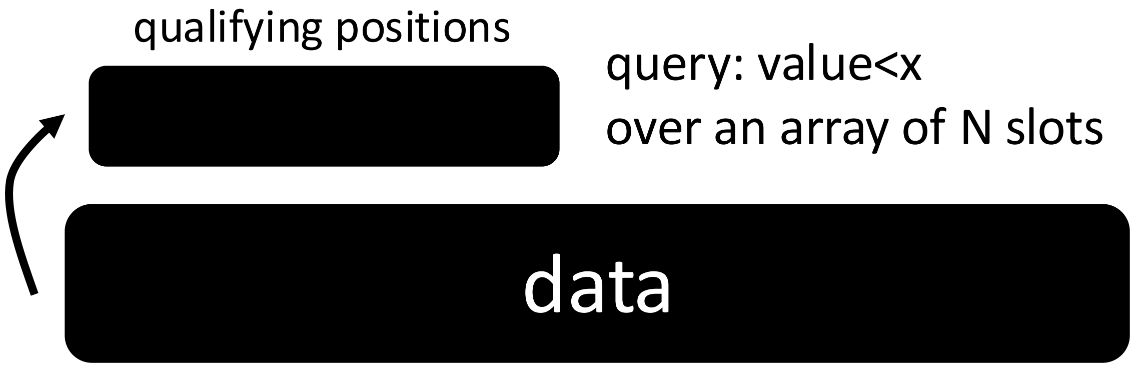
qualifying positions

query: value<x
over an array of N slots

data

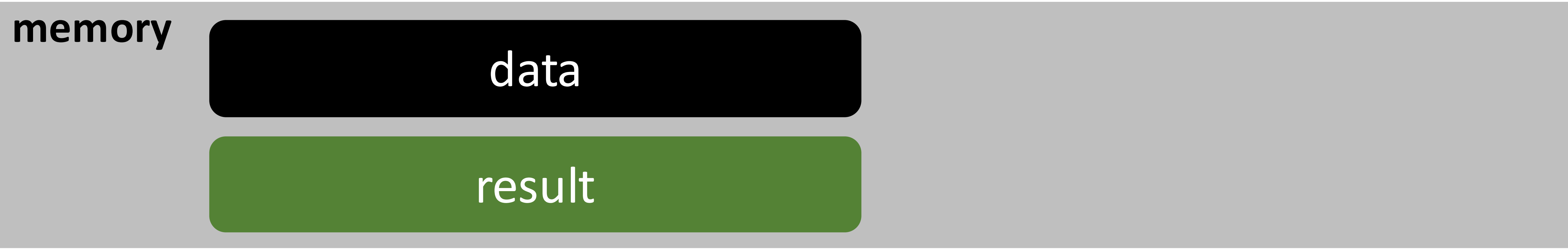
how to implement it?

```
result = new array[data.size];
j=0;
for (i=0; i<data.size; i++)
    if (data[i]<x)
        result[j++]=i;
```

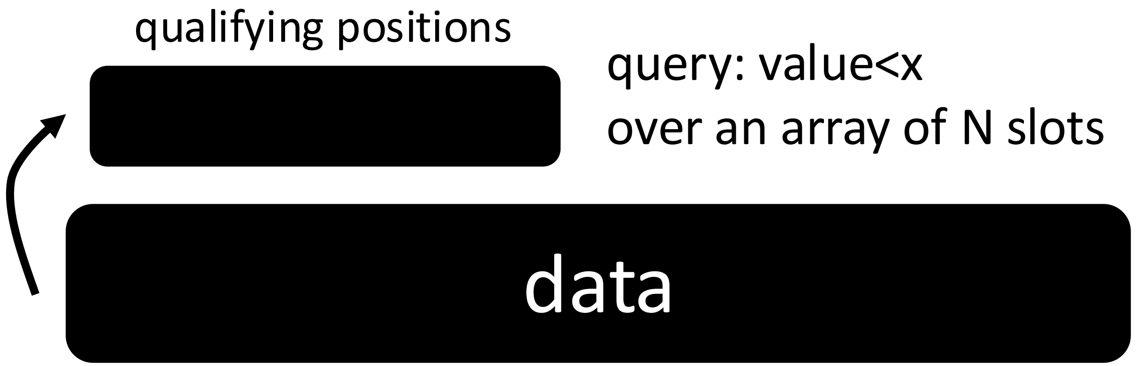


query: value<x
over an array of N slots

what if only 0.1% qualifies?



how to implement it?



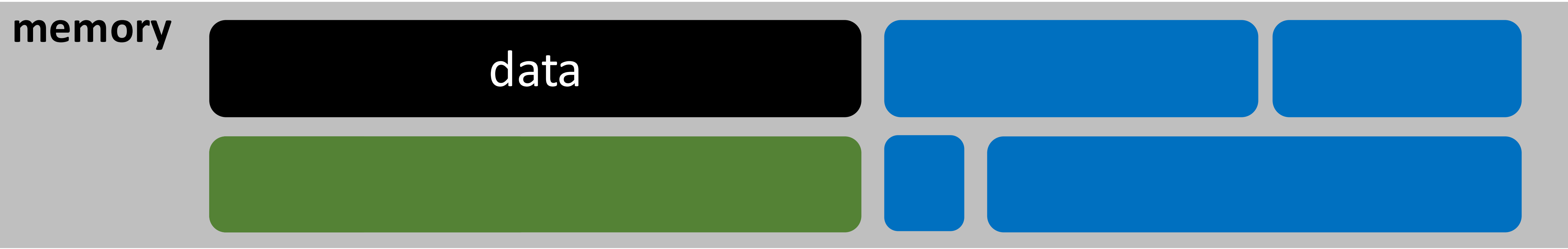
qualifying positions

query: value<x
over an array of N slots

data

```
result = new array[data.size];
j=0;
for (i=0; i<data.size; i++)
    if (data[i]<x)
        result[j++]=i;
```

what if only 0.1% qualifies?



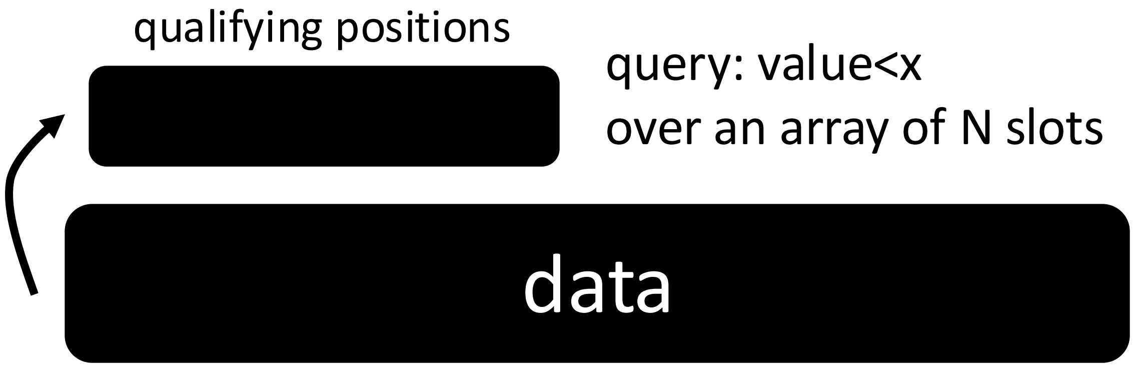
memory

data

how to implement it?

```
result = new array[data.size];
j=0;
for (i=0; i<data.size; i++)
    if (data[i]<x)
        result[j++]=i;
```

```
result = new array[data.size];
j=0;
for (i=0; i<data.size; i++)
    result[j+=(data[i]<x)]=i;
```



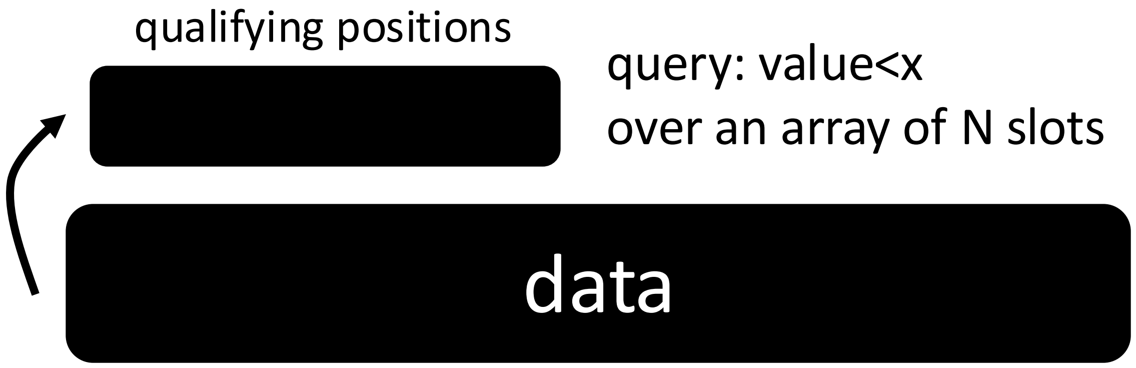
query: value<x
over an array of N slots

what if 99% qualifies?

how can we know?

branches (if statements)
are bad for the processors,
can we avoid them?

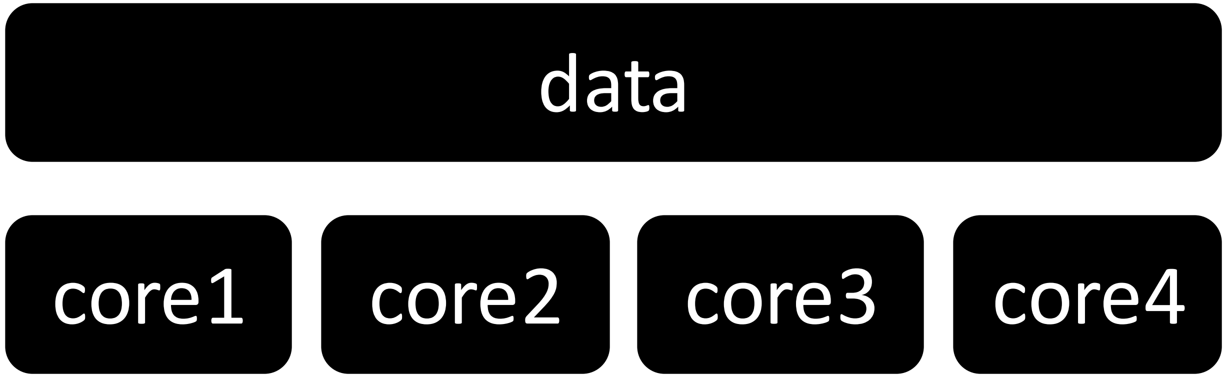
how to bring the values?
(remember we have the positions)



query: value<x
over an array of N slots

```
result = new array[data.size];
j=0;
for (i=0; i<data.size; i++)
    if (data[i]<x)
        result[j++]=i;
```

what about multi-core?
NUMA? SIMD? GPU?

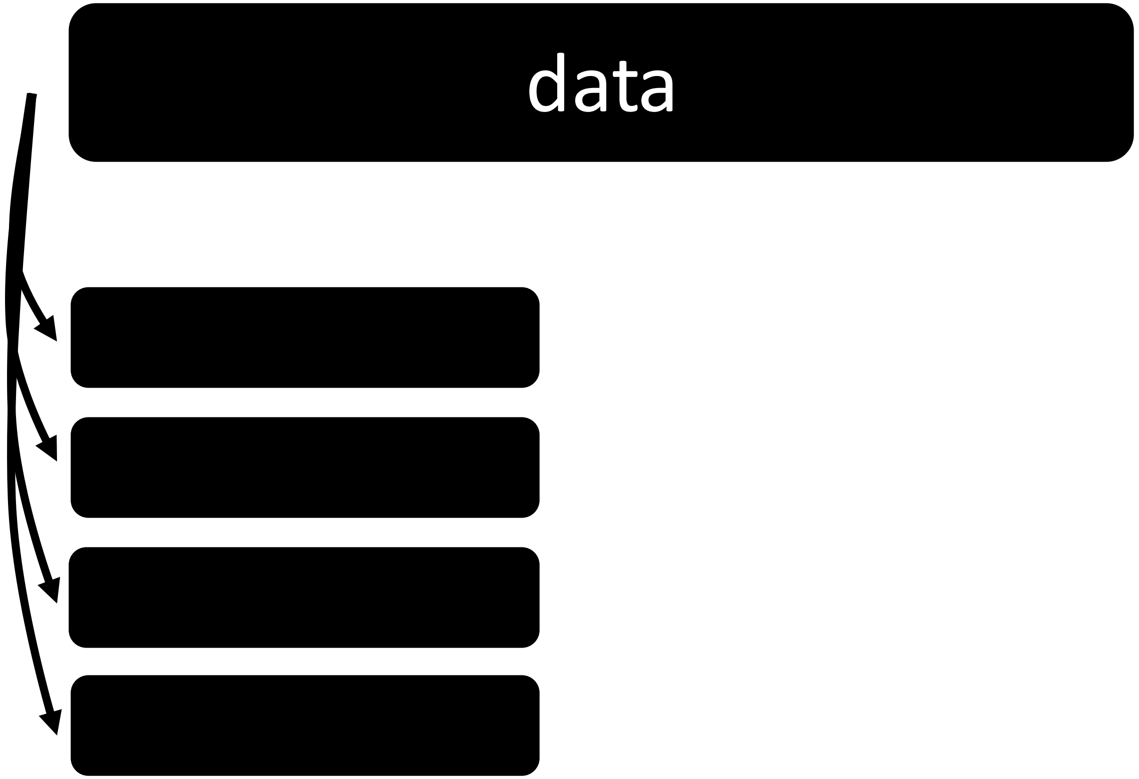


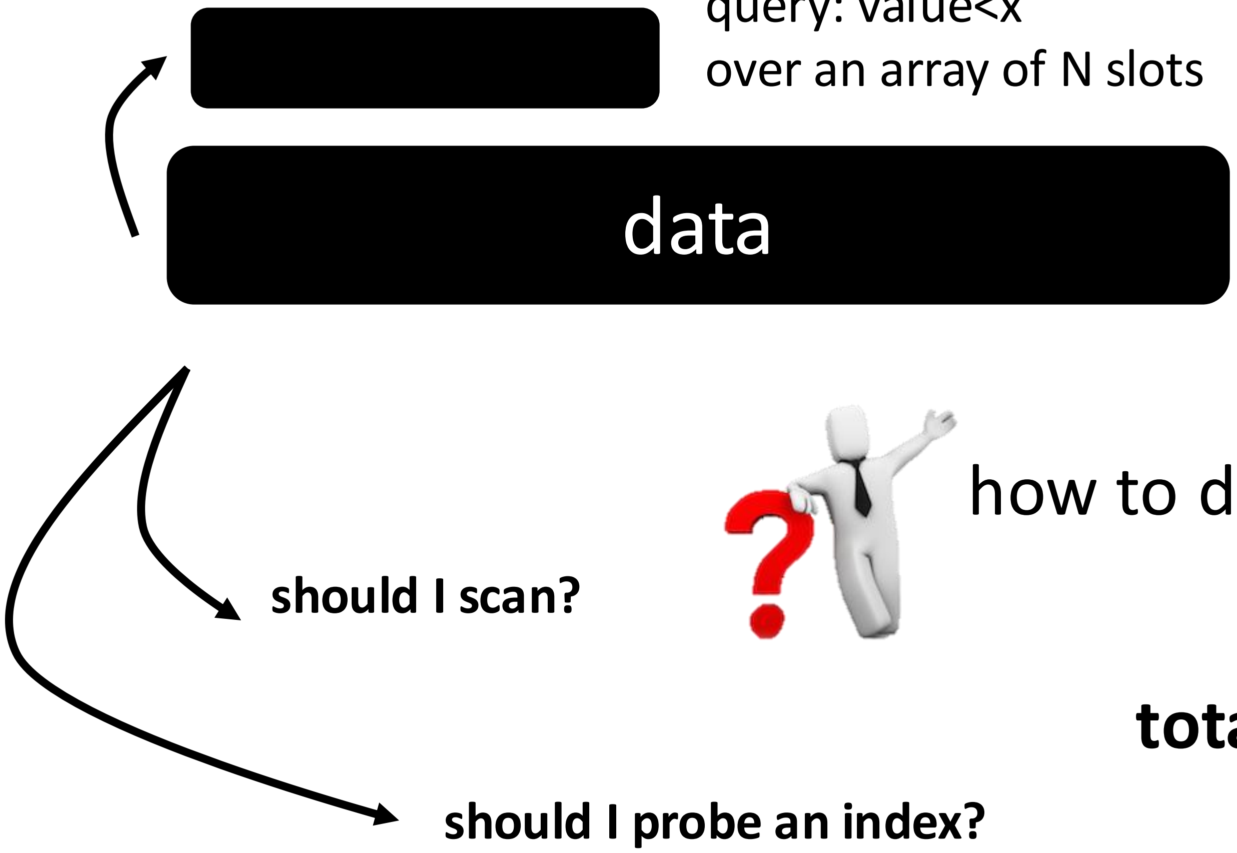
needs coordination!
what about result writing?

what about having multiple queries?

query1: value<x1
query2: value<x2 …

```
result = new array[data.size];
j=0;
for (i=0; i<data.size; i++)
    if (data[i]<x)
        result[j++]=i;
```




query: value<x
over an array of N slots

should I scan?

should I probe an index?

how to decide which one is best?

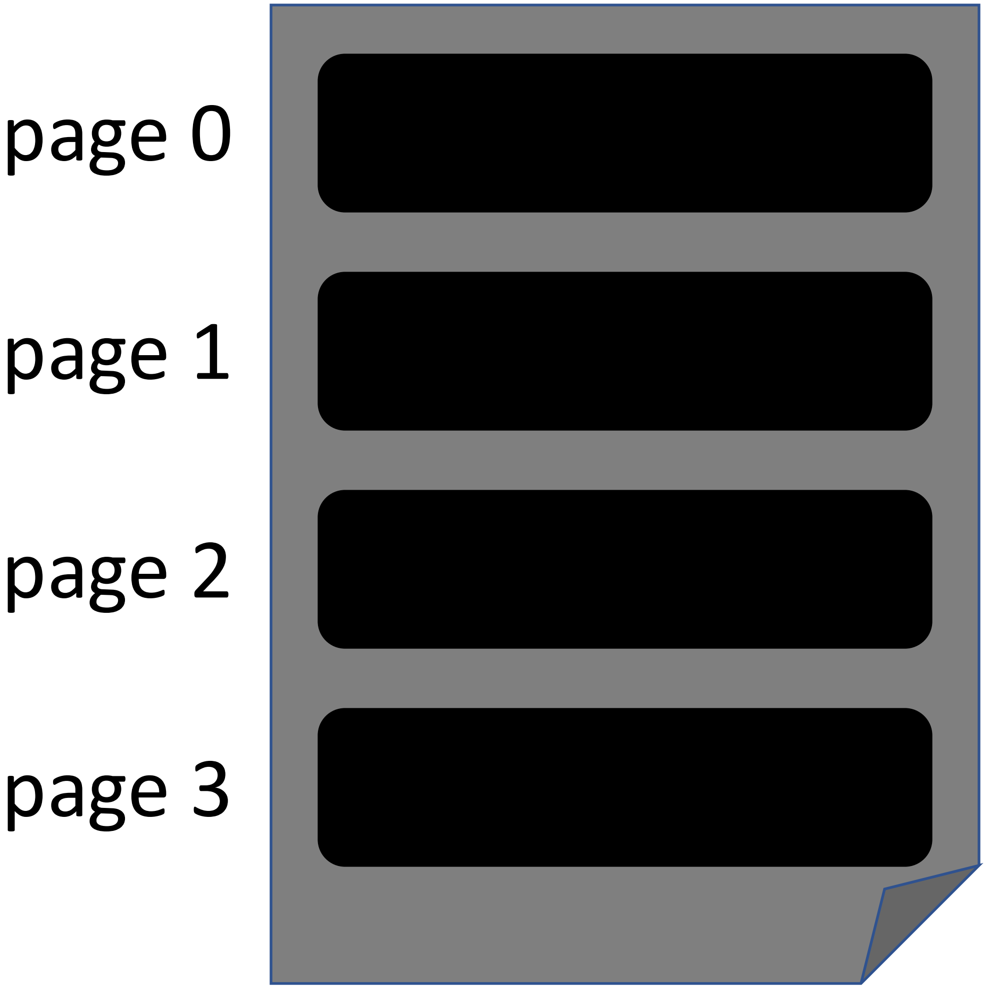
total data movement

&

computation

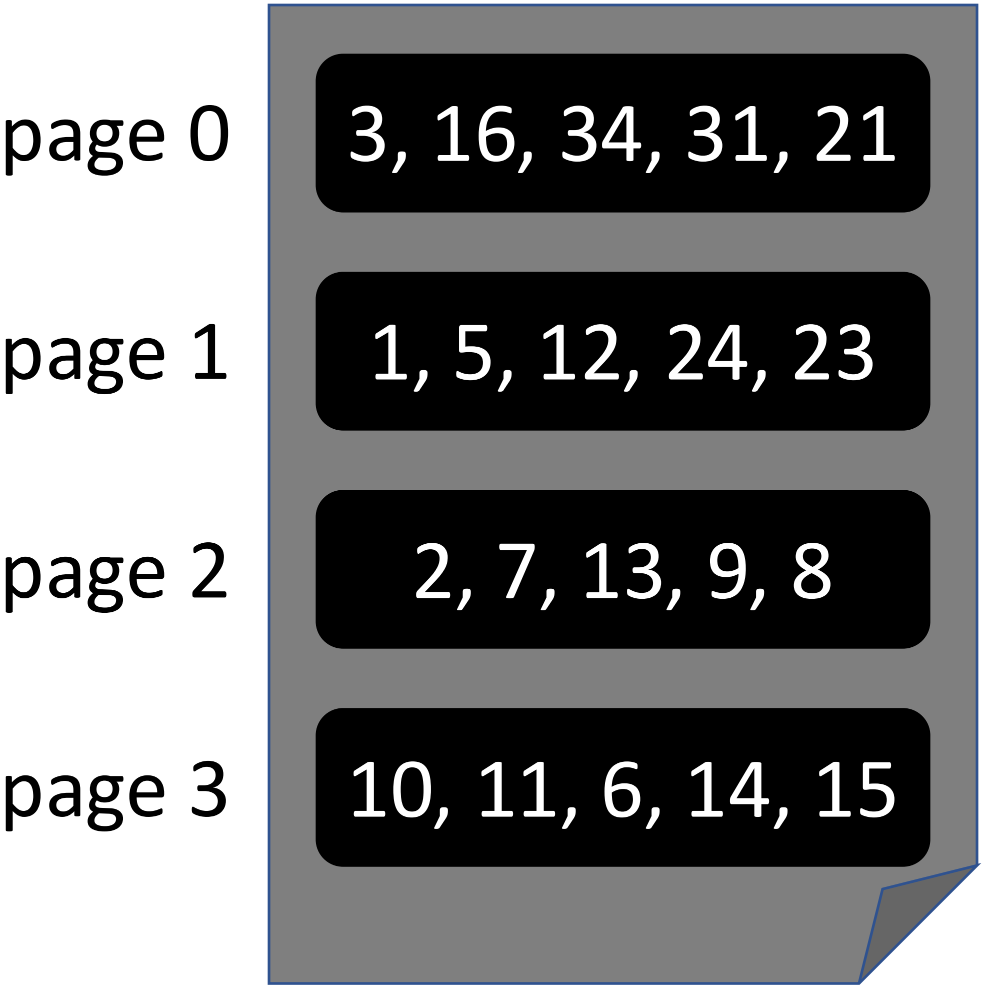
zonemaps

file = collection of pages



zonemaps

file = collection of pages



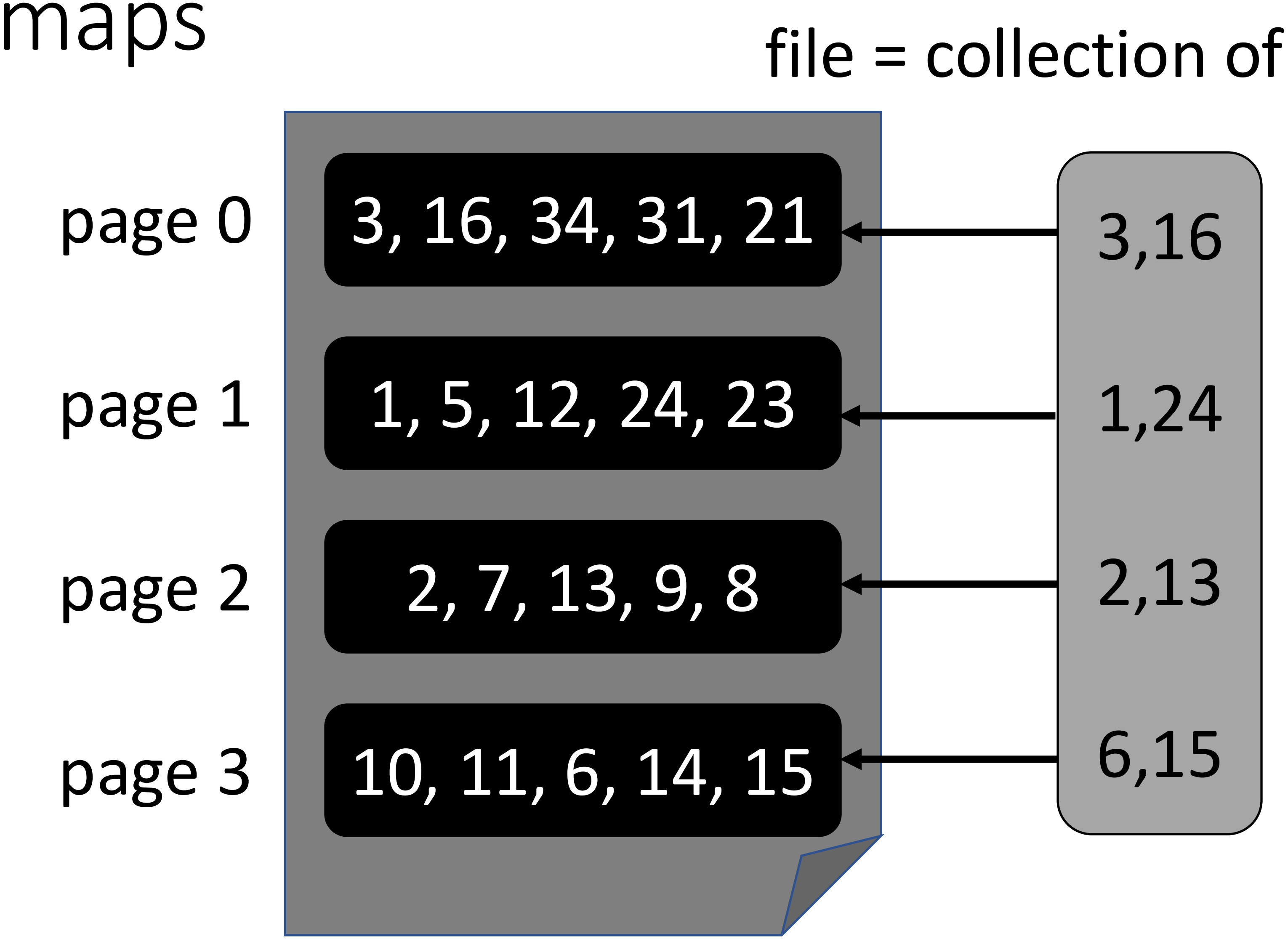
page 0 3, 16, 34, 31, 21

page 1 1, 5, 12, 24, 23

page 2 2, 7, 13, 9, 8

page 3 10, 11, 6, 14, 15

zonemaps



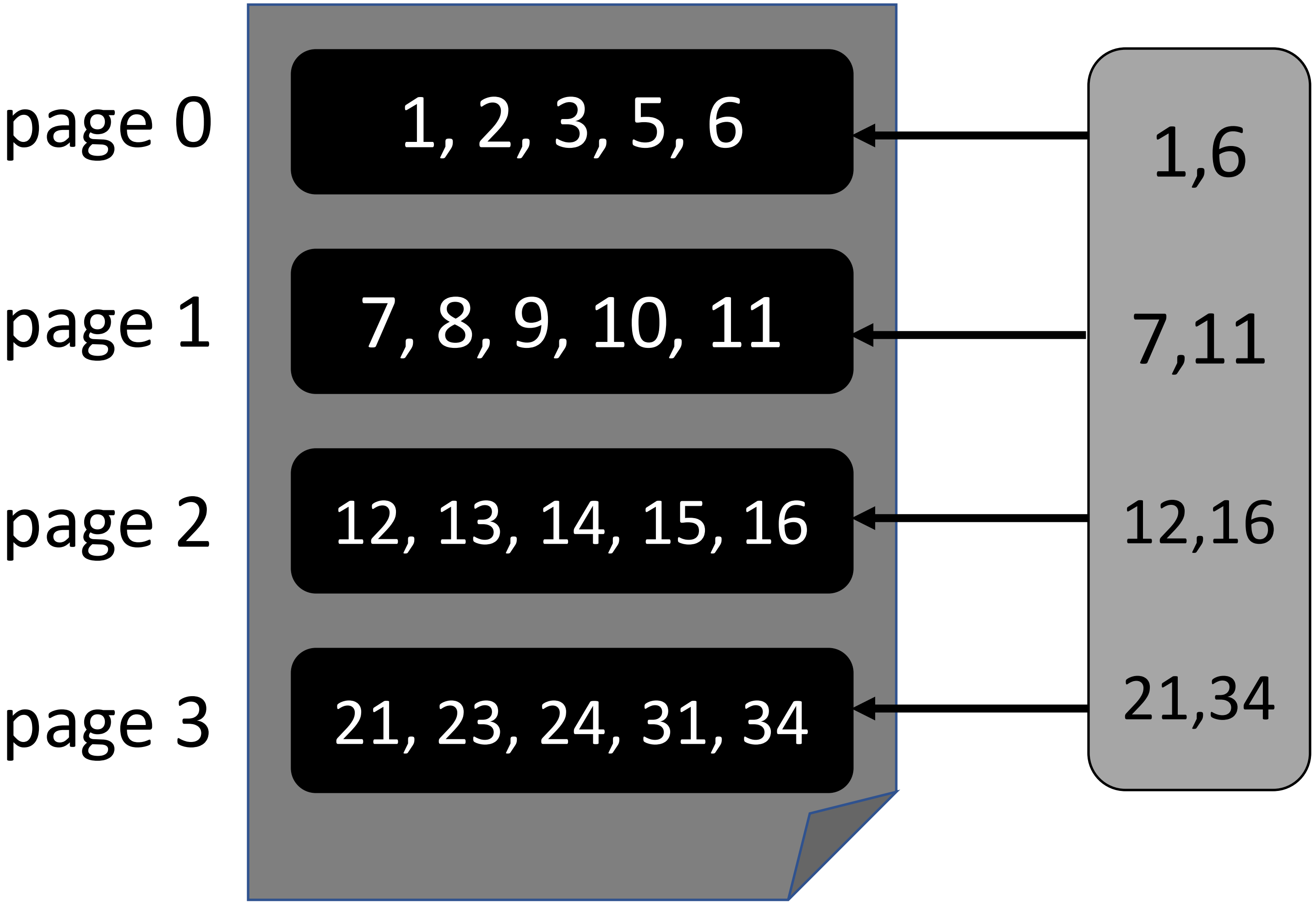
light-weight

typically retained in memory

But what if the data is sorted?

zonemaps

file = collection of pages



page 0	1, 2, 3, 5, 6	1,6
page 1	7, 8, 9, 10, 11	7,11
page 2	12, 13, 14, 15, 16	12,16
page 3	21, 23, 24, 31, 34	21,34

light-weight

typically retained in memory

But what if the data is sorted?

the language of efficient systems: C/C++

why?

fewer assumptions

low-level control over hardware

make decisions about physical data placement and consumptions

the language of efficient systems: C/C++

why?

fewer assumptions

we want you in the project to make low-level decisions

BOSTON
UNIVERSITY

CS 561: Data Systems Architectures

class 2

Data Systems 101

next :

modern main-memory data systems

&

semester project