BOSTON UNIVERSITY

CS 561: Data Systems Architectures

class 2

Data Systems 101

Prof. Manos Athanassoulis

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

If you are at home, make it full screen and focus on our discussion

some reminders

no smartphones

no laptop

class summary

2 classes per week / OH 5 days per week

each student

1 presentation/discussion lead + 1 review/question per week

project 0 + systems or research project

proposal + mid-semester report + final report/presentation

systems project

implementation-heavy C/C++ project

groups of 2

research project

groups of 3

pick a subject (list will be available)

design & analysis

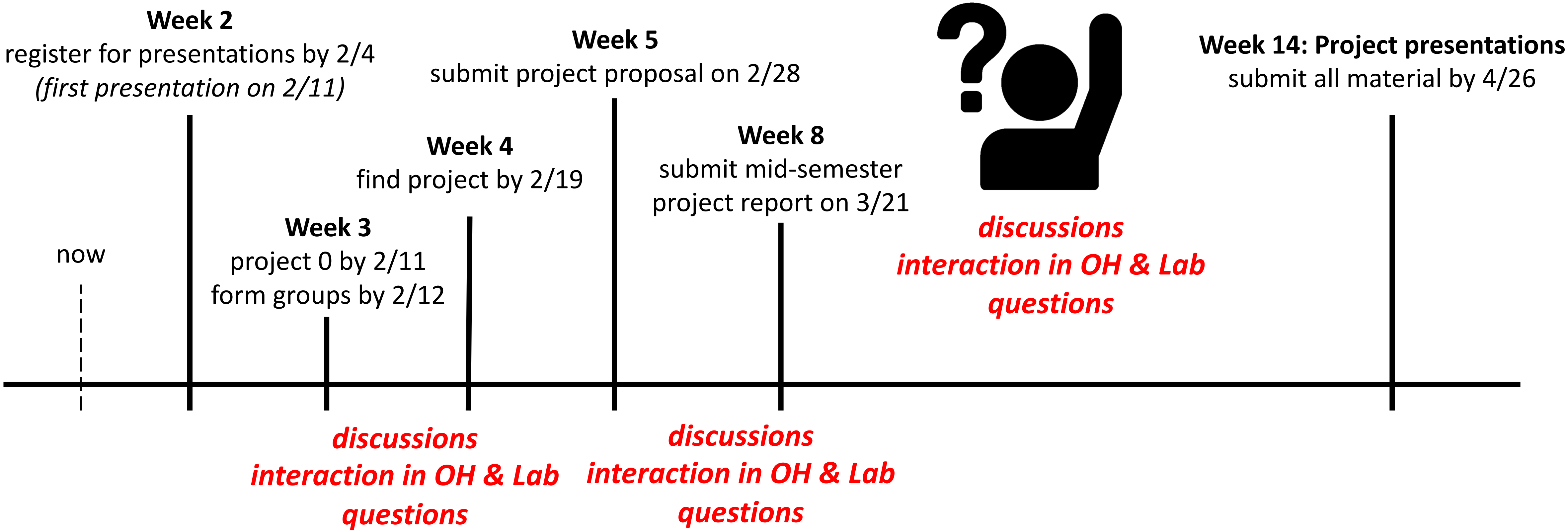
experimentation

Project 0:
A small implementation project to sharpen dev skills

<u>independent project</u>

more details this week

class timeline



now

Week 2
register for presentations by 2/4
(first presentation on 2/11)

Week 3
project 0 by 2/11
form groups by 2/12

discussions
interaction in OH & Lab
questions

Week 4
find project by 2/19

Week 5
submit project proposal on 2/28

discussions
interaction in OH & Lab
questions

Week 8
submit mid-semester
project report on 3/21

discussions
interaction in OH & Lab
questions

Week 14: Project presentations
submit all material by 4/26

Piazza

all discussions & announcements

http://piazza.com/bu/spring2021/cs561/

also available on class website

I have added everyone who already registered!

Please double-check!

size (volume)

rate (velocity)

sources (variety)

big data

(it's not only about size)

The 3 V's

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

scientific experiments

sensors

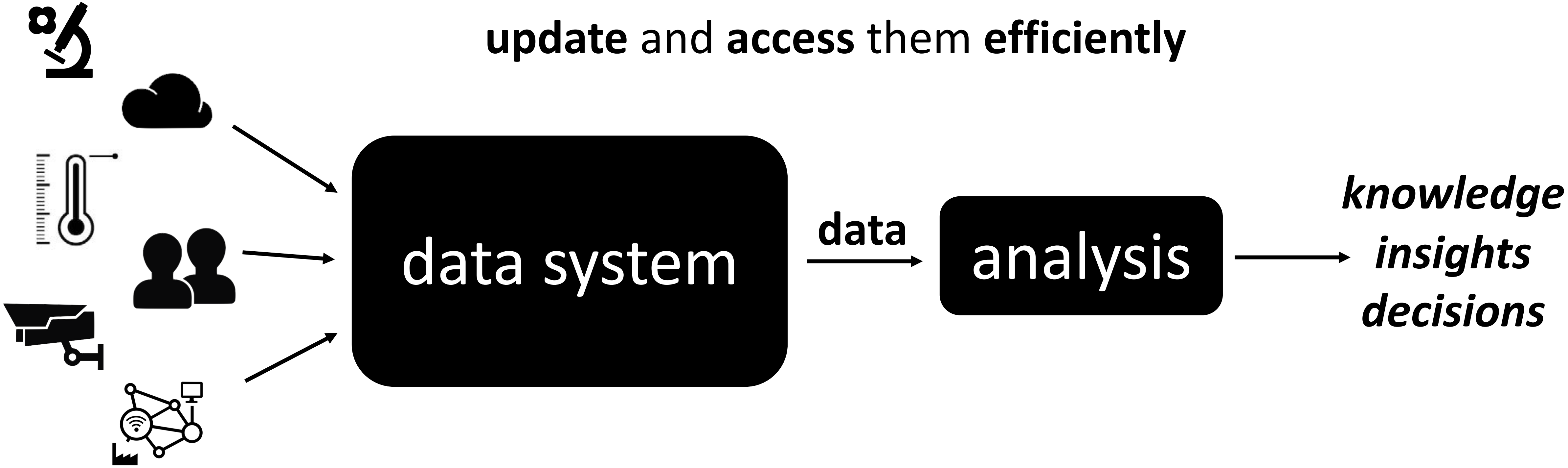
social

Internet-of-Things

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

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

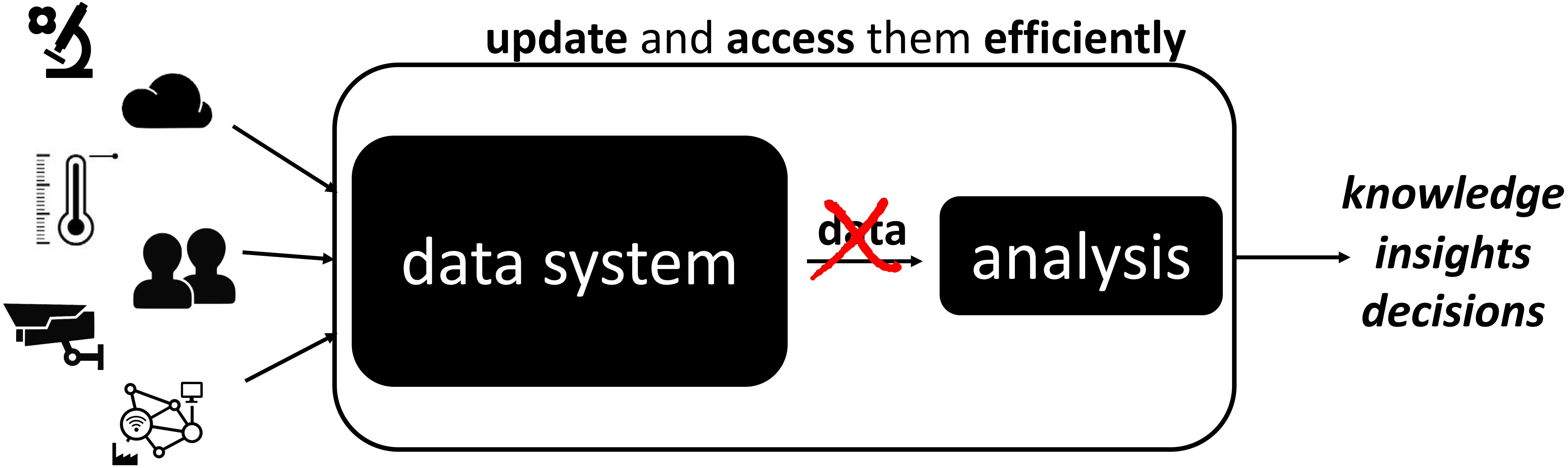
update and **access** them **efficiently**



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

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

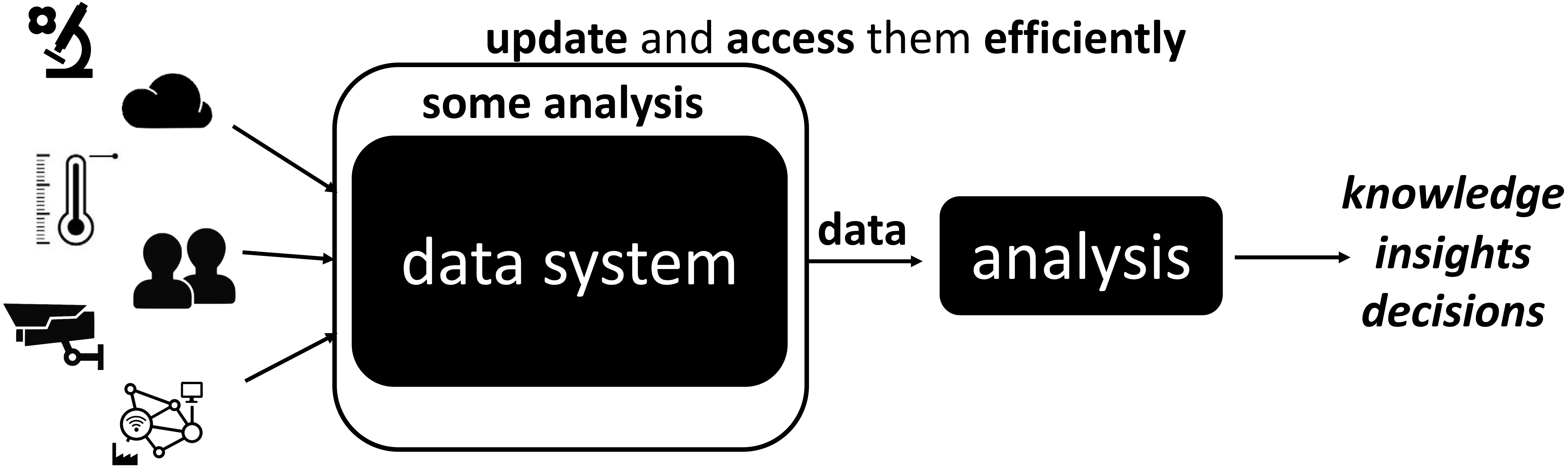
update and **access** them **efficiently**



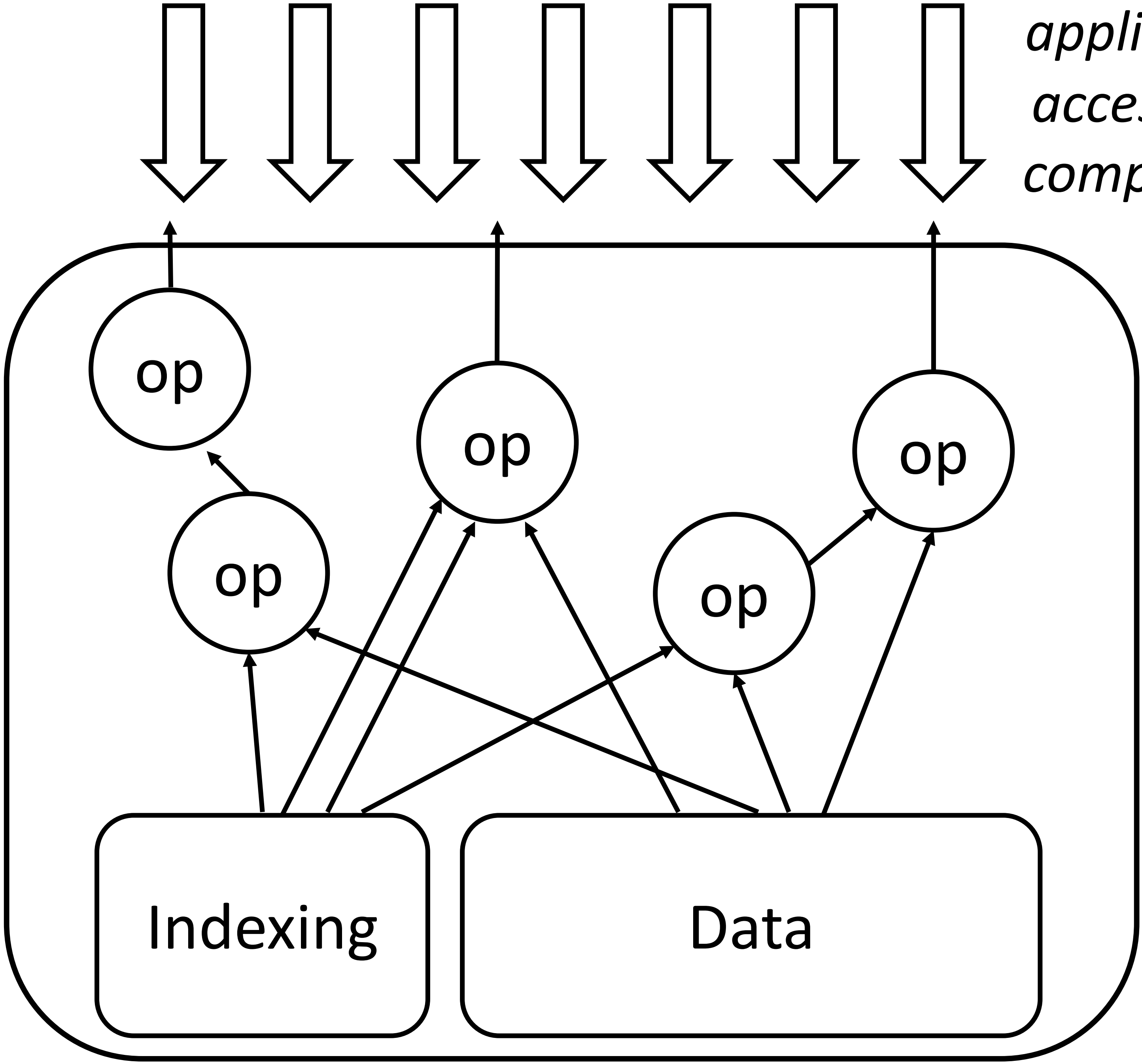
a **data system** is a large software system

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

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

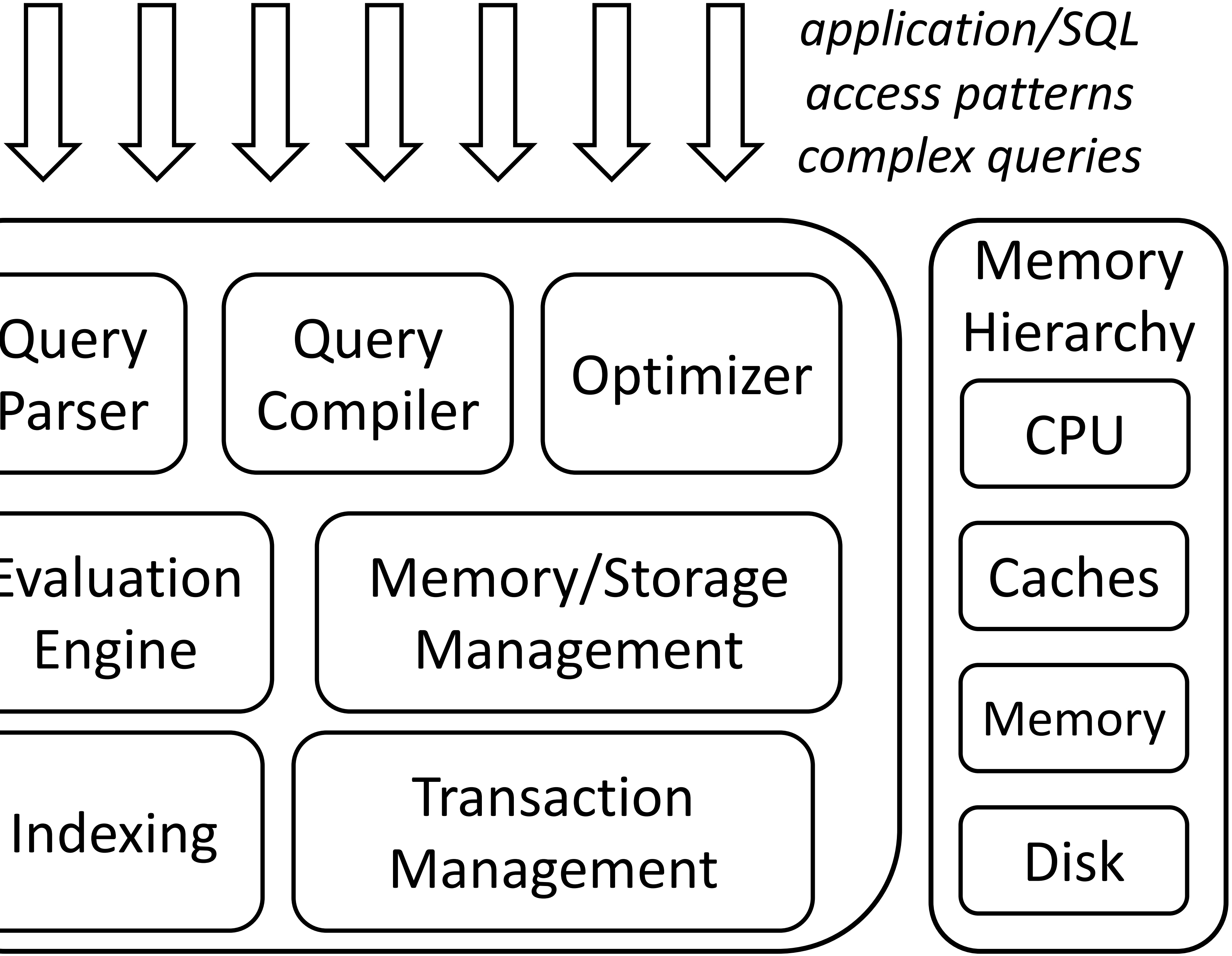


data system, what’s inside?



application/SQL
access patterns
complex queries

algorithms
and
operators



application/SQL
access patterns
complex queries

modules

Query Parser

Query Compiler

Optimizer

Evaluation Engine

Memory/Storage Management

Indexing

Transaction Management

Memory Hierarchy

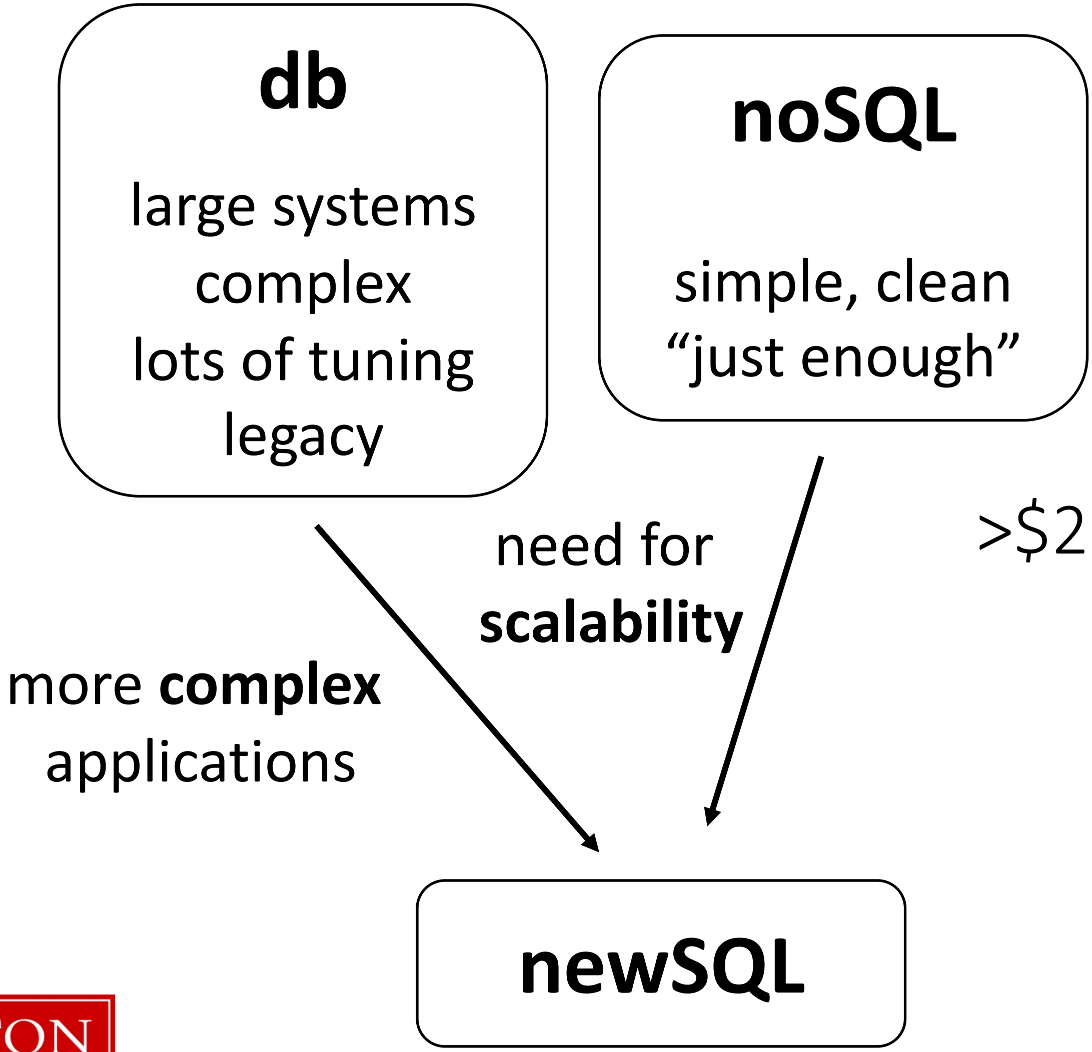
CPU

Caches

Memory

Disk

growing environment



db

large systems
complex
lots of tuning
legacy

noSQL

simple, clean
"just enough"

more **complex** applications

need for **scalability**

newSQL

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

[The Forbes, 2016]

[noSQL]

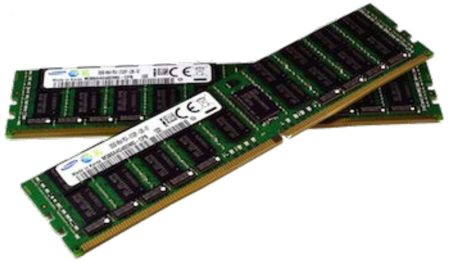
$3B by 2020, growing at 20% every year

[Forrester, 2016]

growing need for tailored systems

new applications

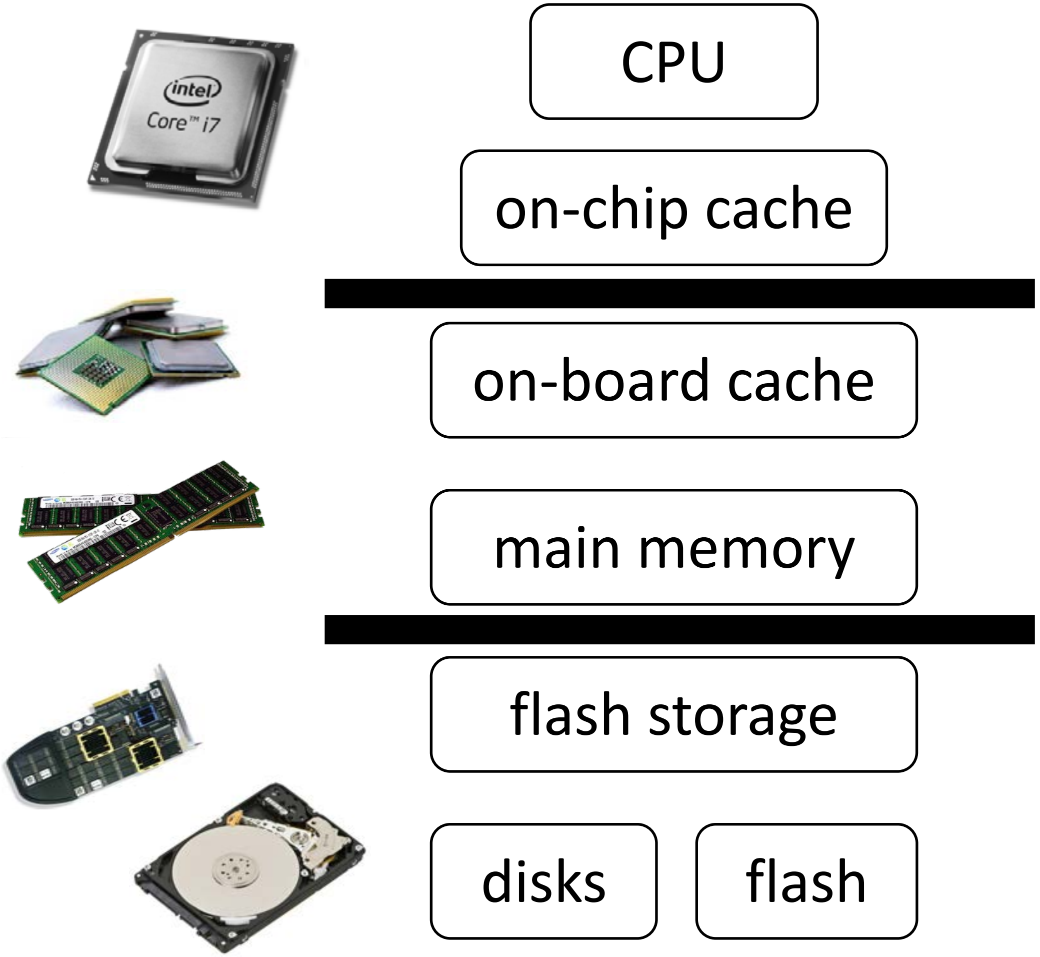
new hardware



more data

data system, what’s underneath?

memory hierarchy



CPU

on-chip cache

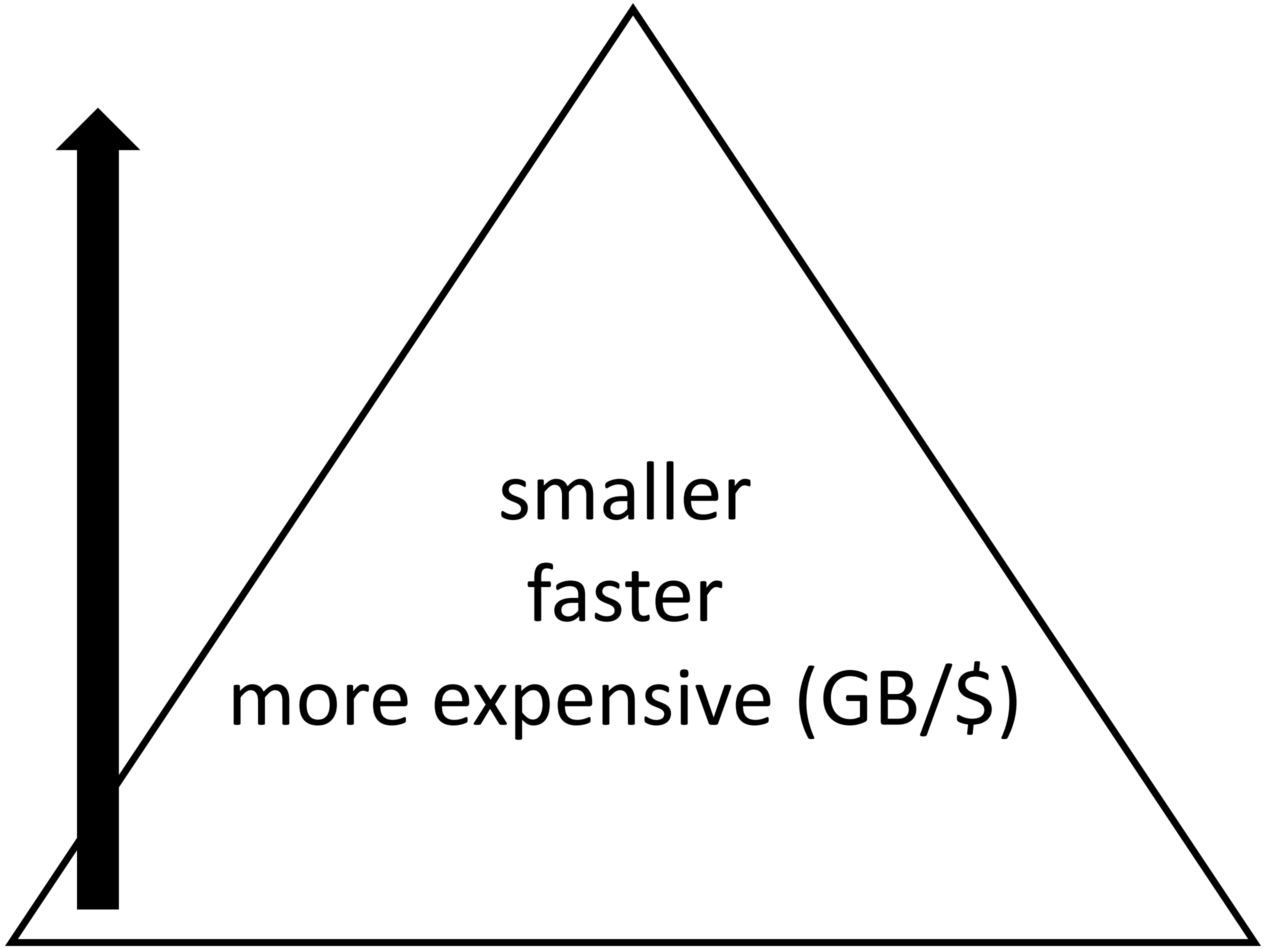
on-board cache

main memory

flash storage

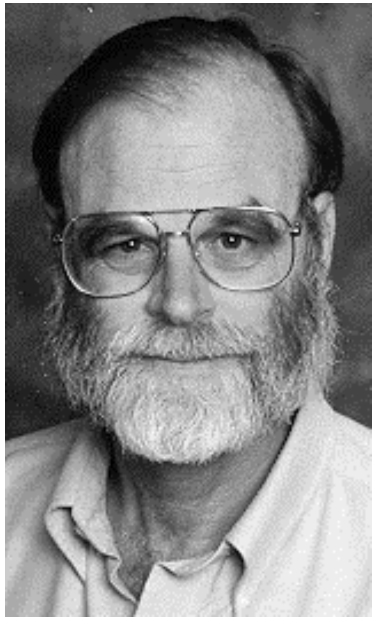
disks

flash



smaller
faster
more expensive (GB/$)

memory hierarchy (by Jim Gray)



Jim Gray, IBM, Tandem, Microsoft, DEC

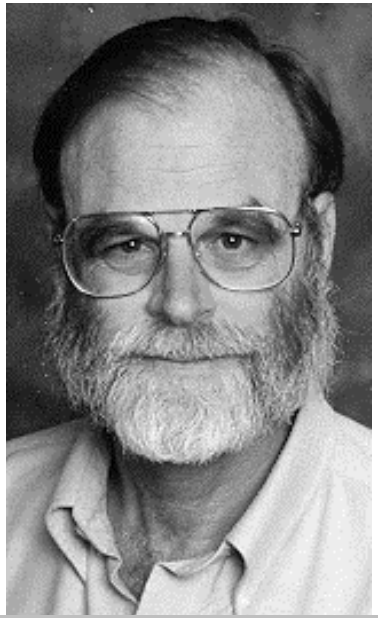
“The Fourth Paradigm” is based on his vision

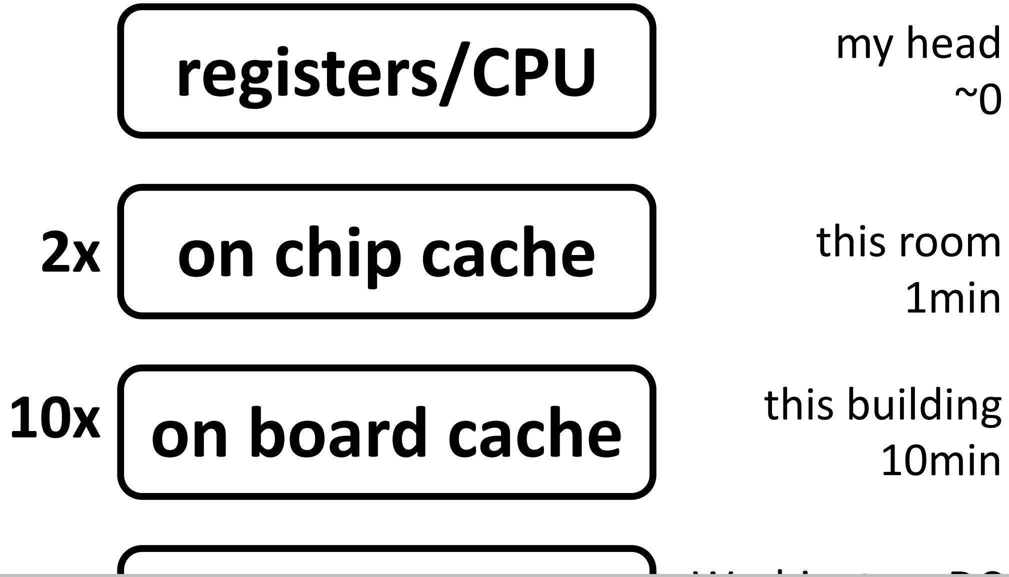
ACM Turing Award 1998

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)

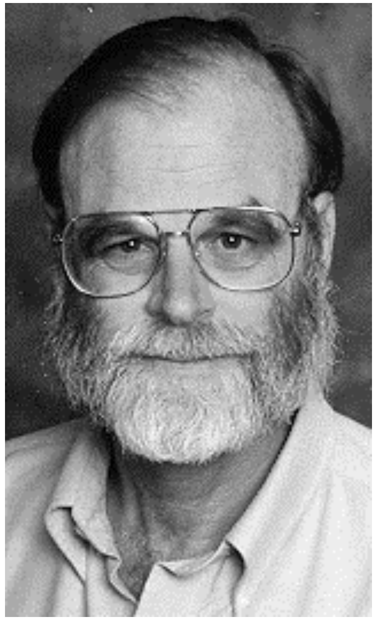


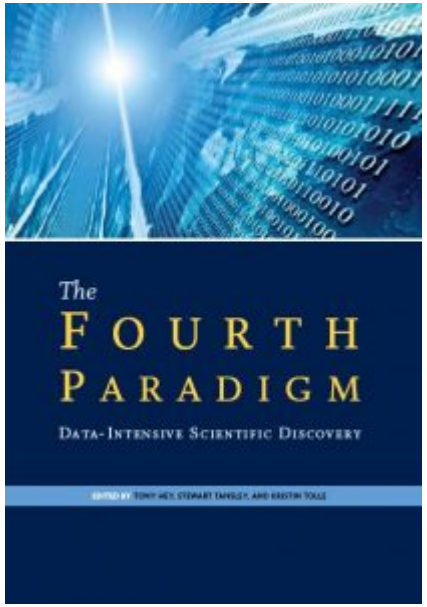


	registers/CPU	my head ~0
2x	on chip cache	this room 1min
10x	on board cache	this building 10min

tape?
sequential-only magnetic storage
still a multi-billion industry

Jim Gray (a great scientist and engineer)

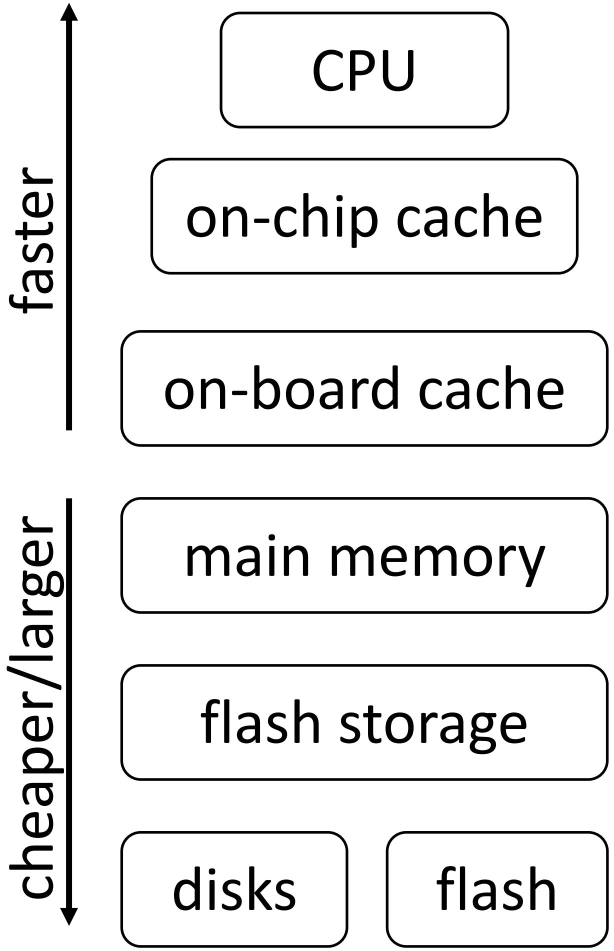



The
Fourth
Paradigm
Data-Intensive Scientific Discovery
Edited by Tony Hey, Stewart Tansley, and Kristin Tolle

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

Jim Gray, IBM, Tandem, Microsoft, DEC
“The Fourth Paradigm” is based on his vision
ACM Turing Award 1998
ACM SIGMOD Edgar F. Codd Innovations award 1993

memory wall



faster

cheaper/larger

CPU

on-chip cache

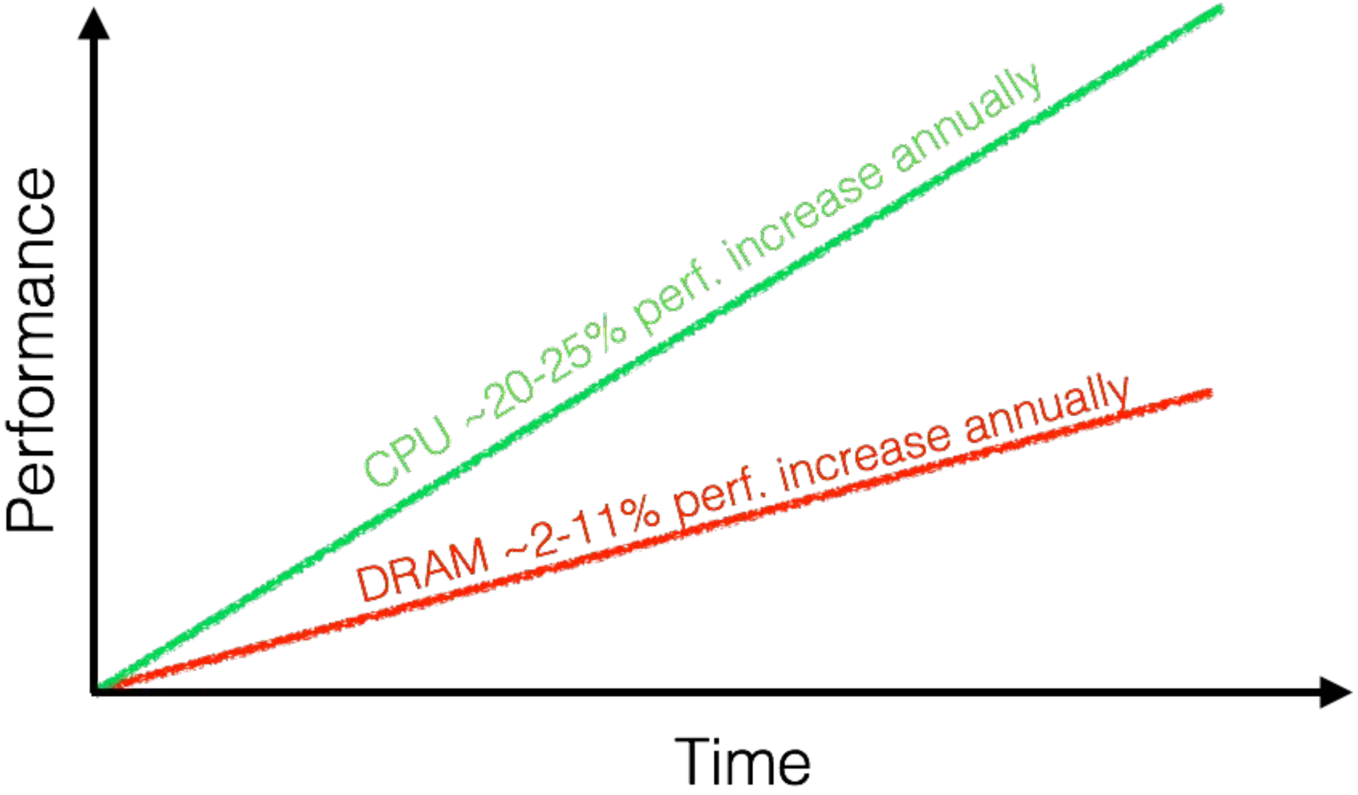
on-board cache

main memory

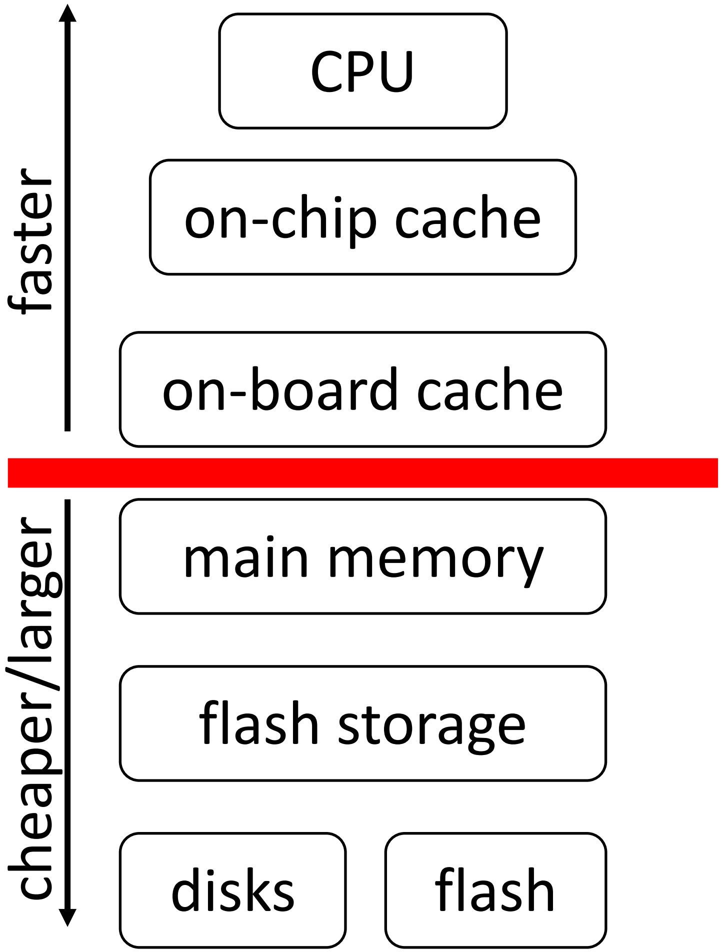
flash storage

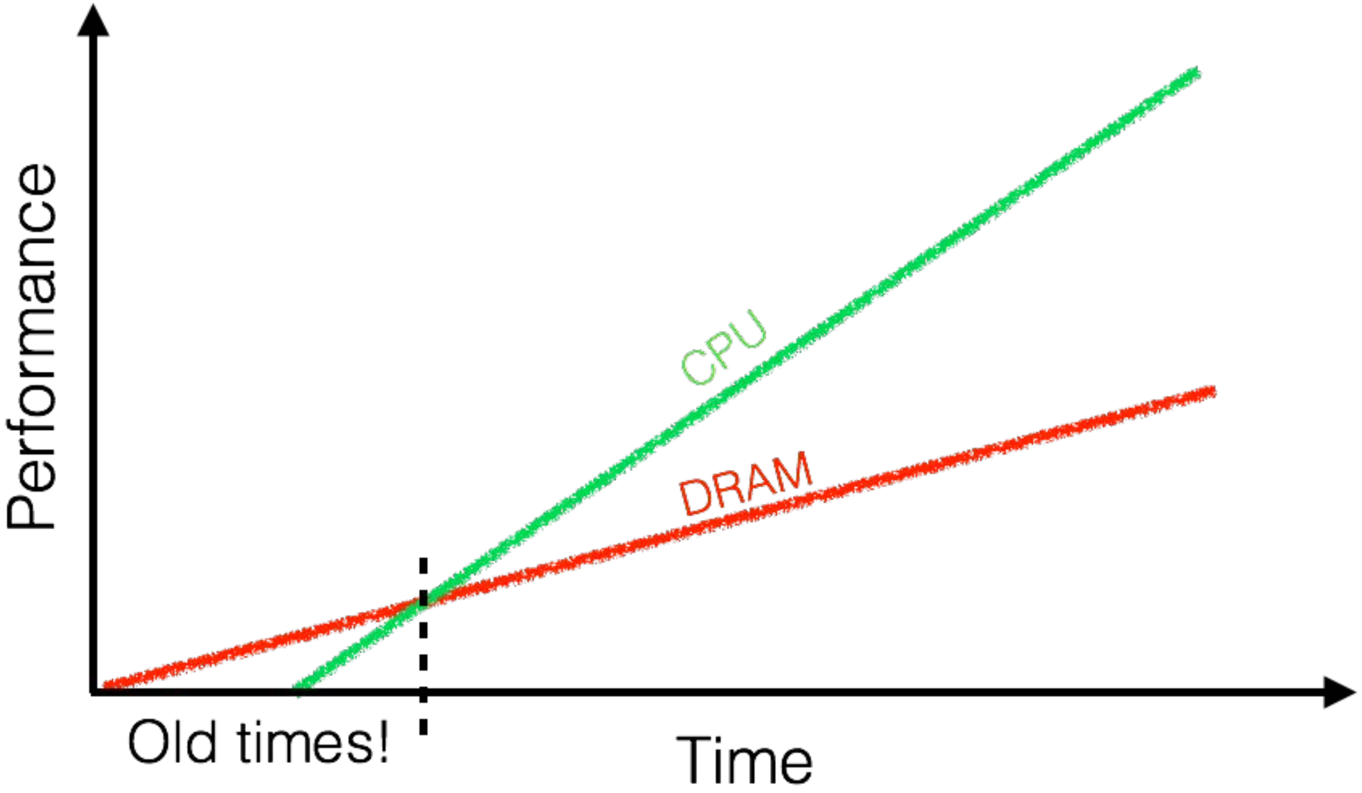
disks

flash

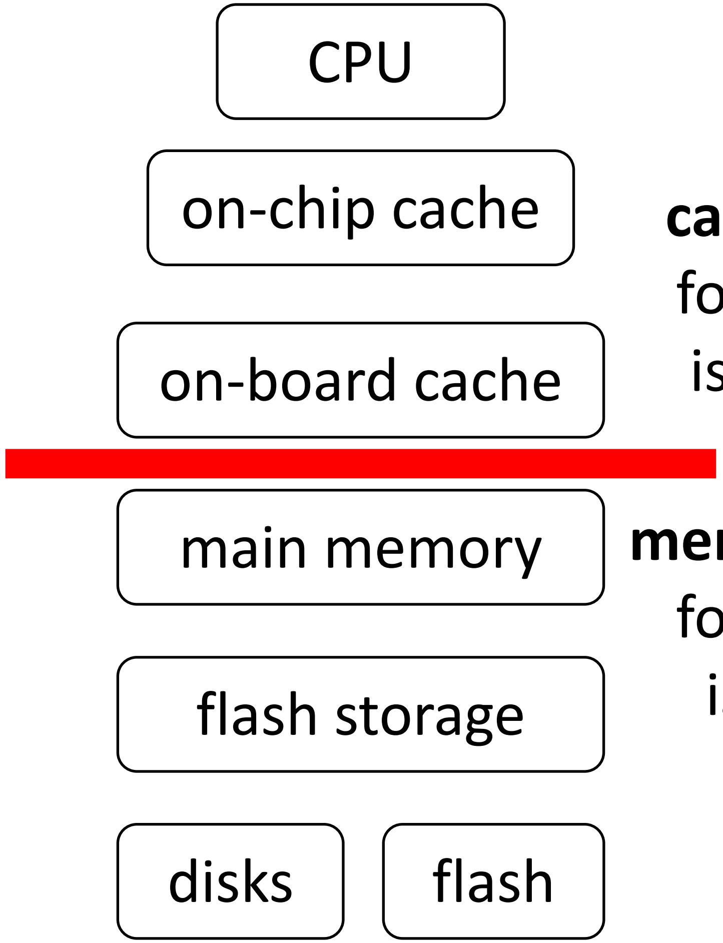


memory wall


faster
cheaper/larger
CPU
on-chip cache
on-board cache
main memory
flash storage
disks
flash


Performance
CPU
DRAM
Old times!
Time

cache/memory misses

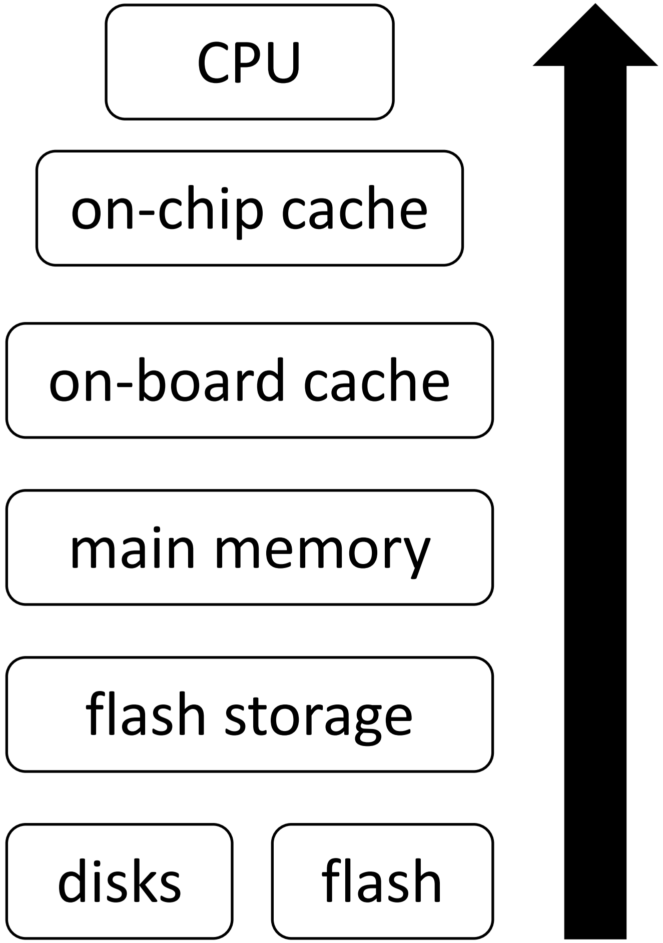


cache miss: looking for something that is not in the cache

memory miss: looking for something that is not in memory

what happens if I miss?

data movement



CPU

on-chip cache

on-board cache

main memory

flash storage

disks

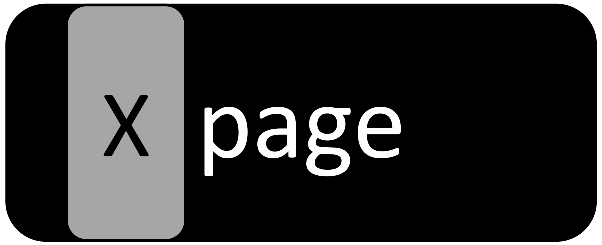
flash

data go through
all necessary levels

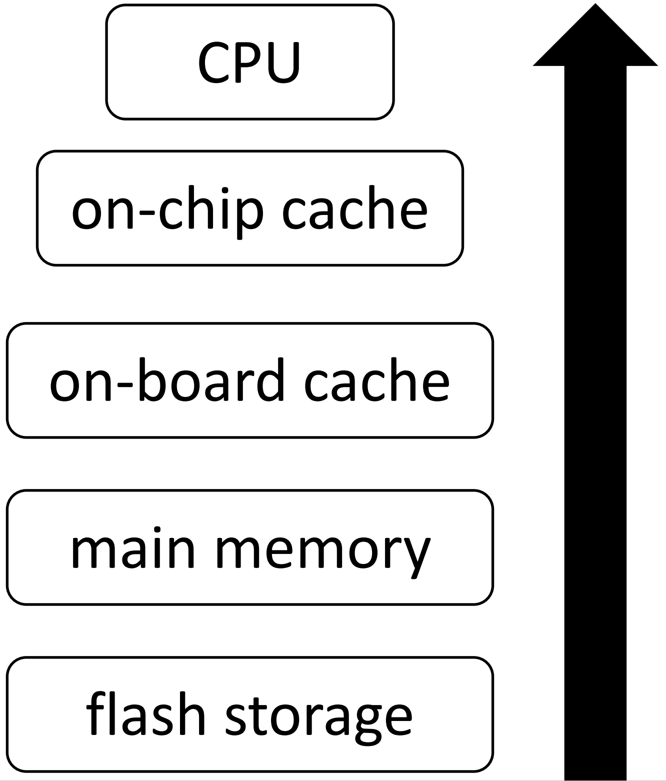
also read
unnecessary data

Photo by Gary Dineen/NBAE via Getty Images

need to read only X
read the whole page



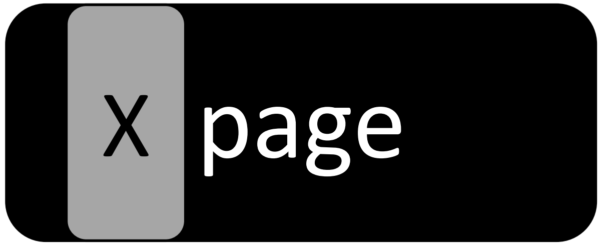
data movement



data go through
all necessary levels

Photo by Gary Dineen/NBAE via Getty Images

need to read only X
read the whole page



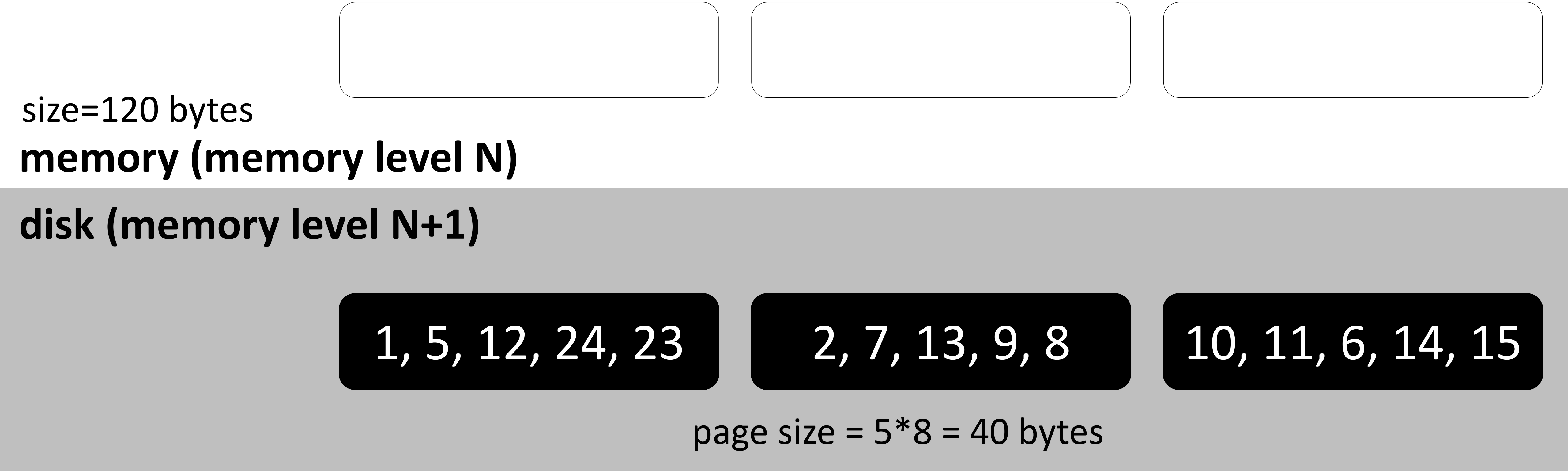
also read
unnecessary data

remember!

disk is millions (mem, hundreds) 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

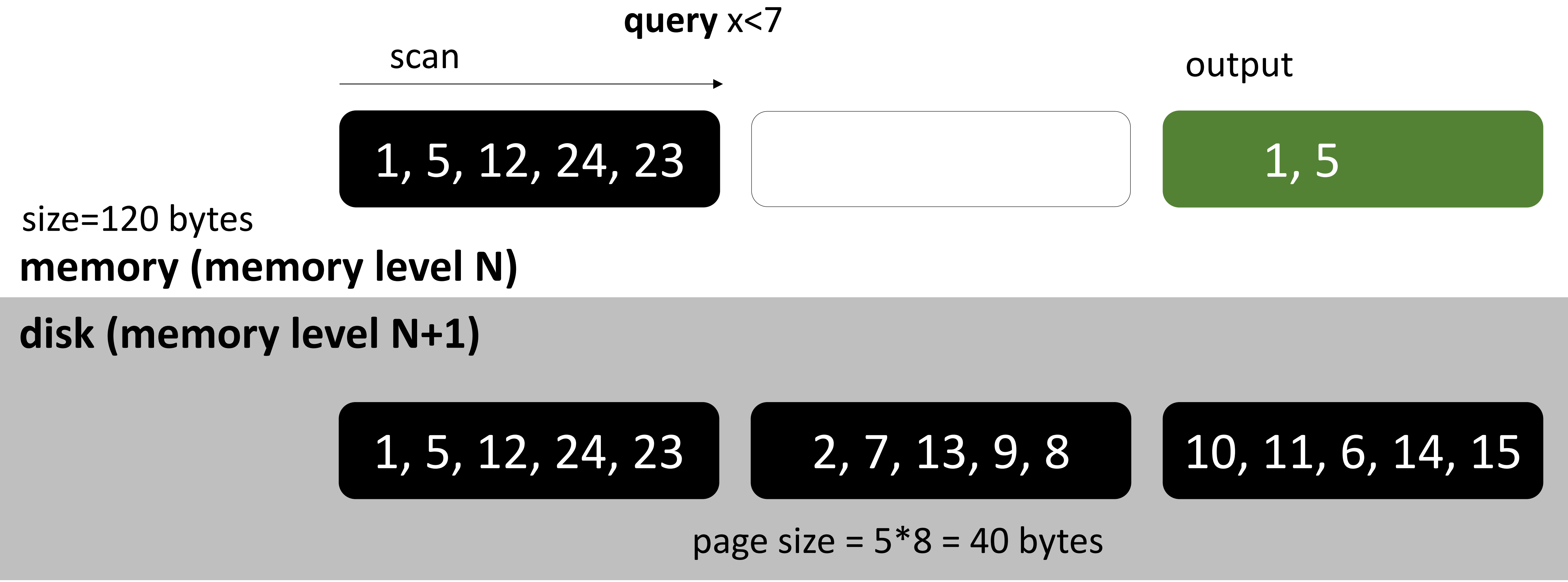
2, 7, 13, 9, 8

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

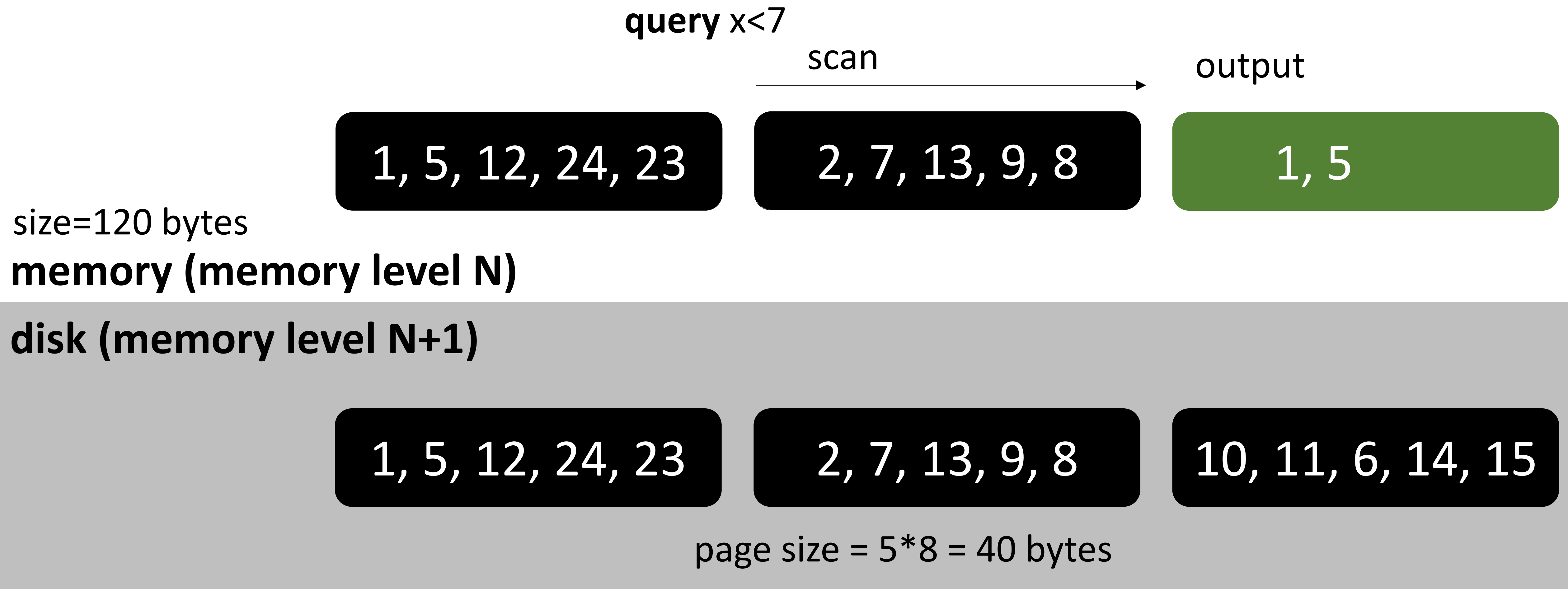
2, 7, 13, 9, 8

10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

$ 40 bytes

page-based access & random access



query x<7

scan

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

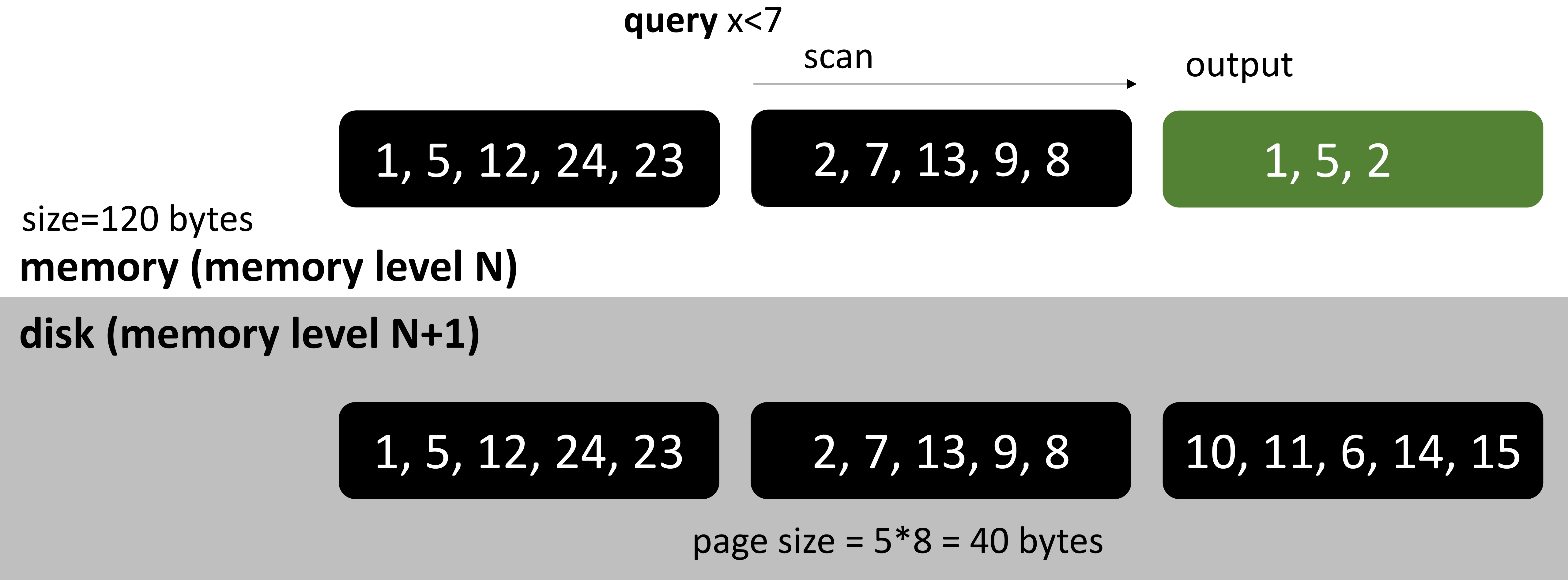
2, 7, 13, 9, 8

10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

page-based access & random access

$ 40 bytes



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

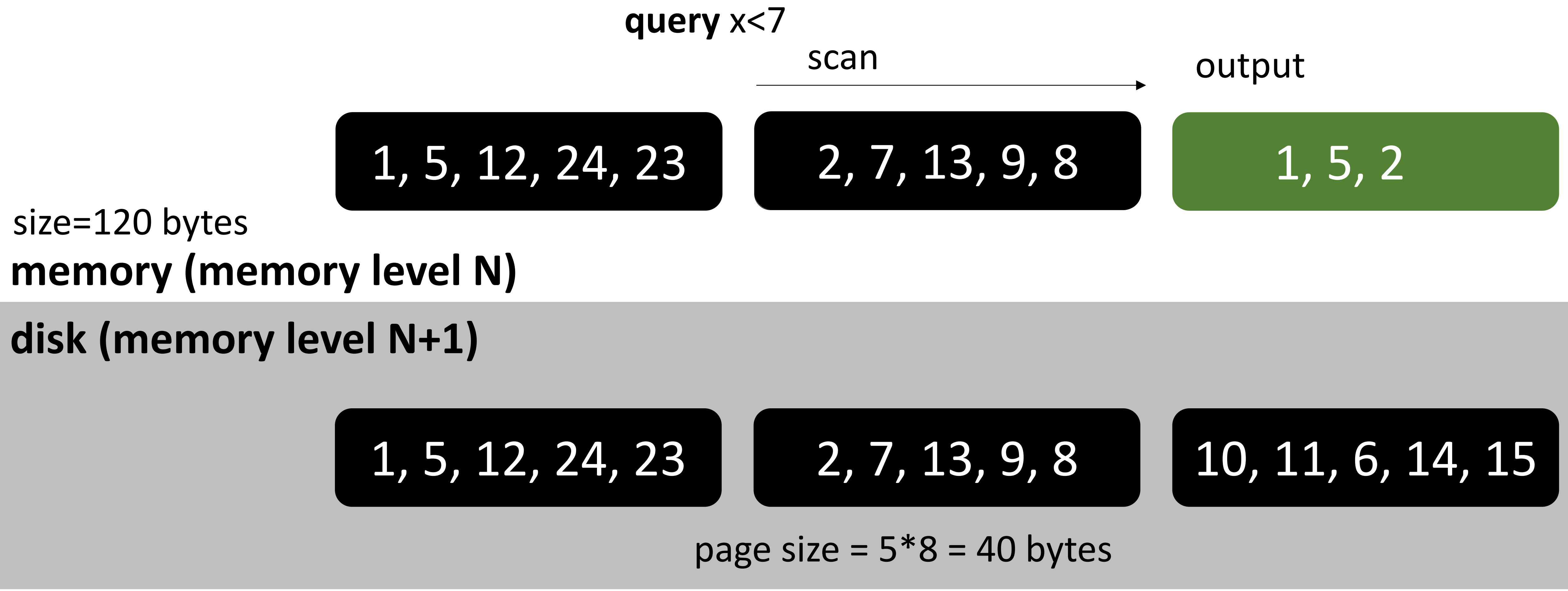
2, 7, 13, 9, 8

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

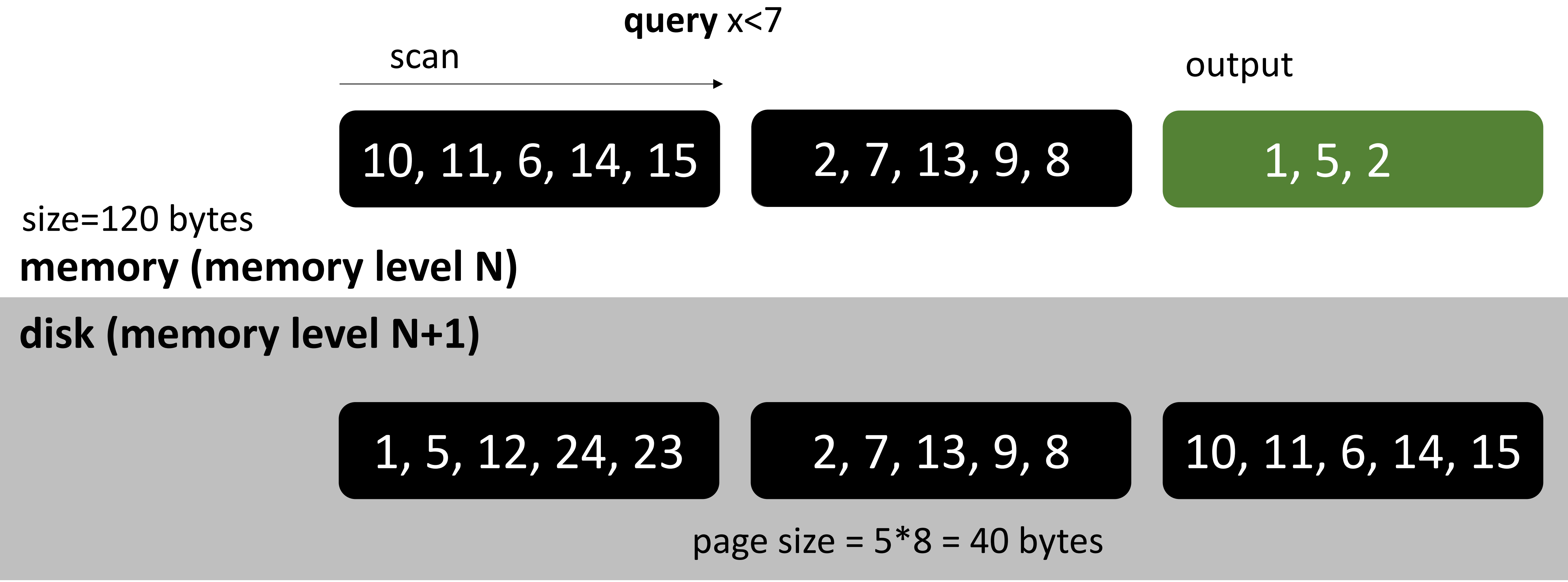
2, 7, 13, 9, 8

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

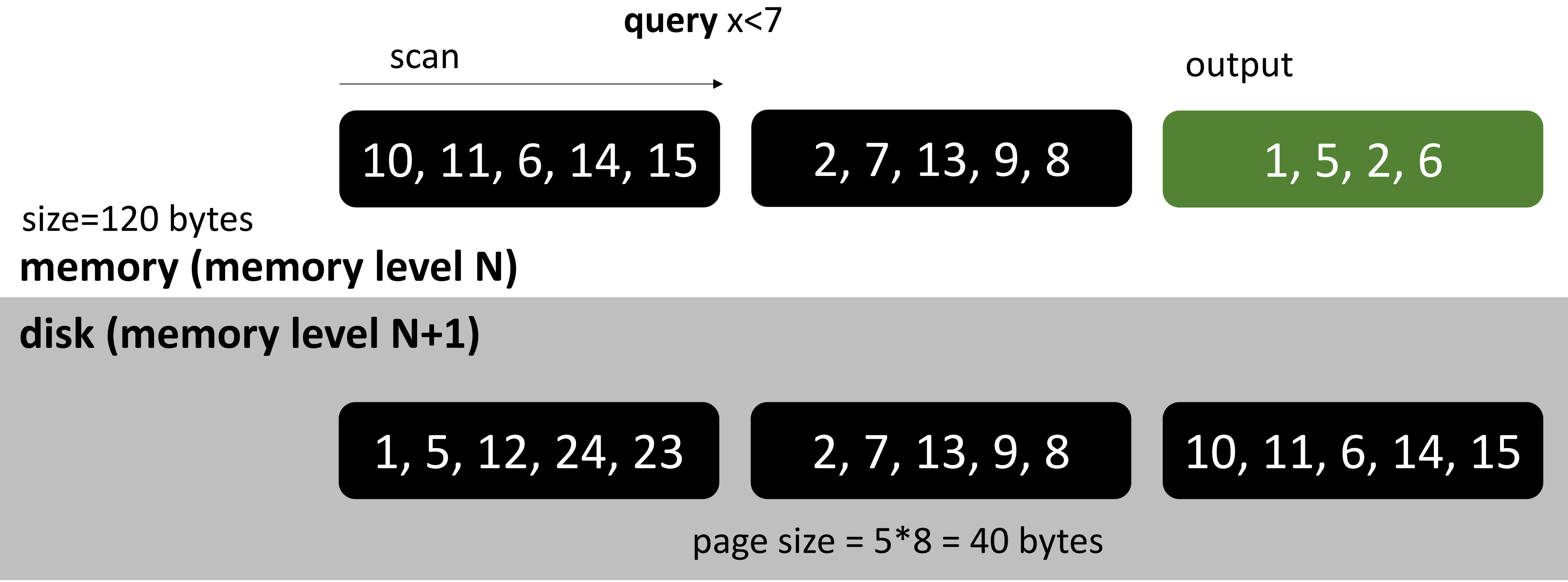
2, 7, 13, 9, 8

10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

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, 6

size=120 bytes

memory (memory level N)

disk (memory level N+1)

1, 5, 12, 24, 23

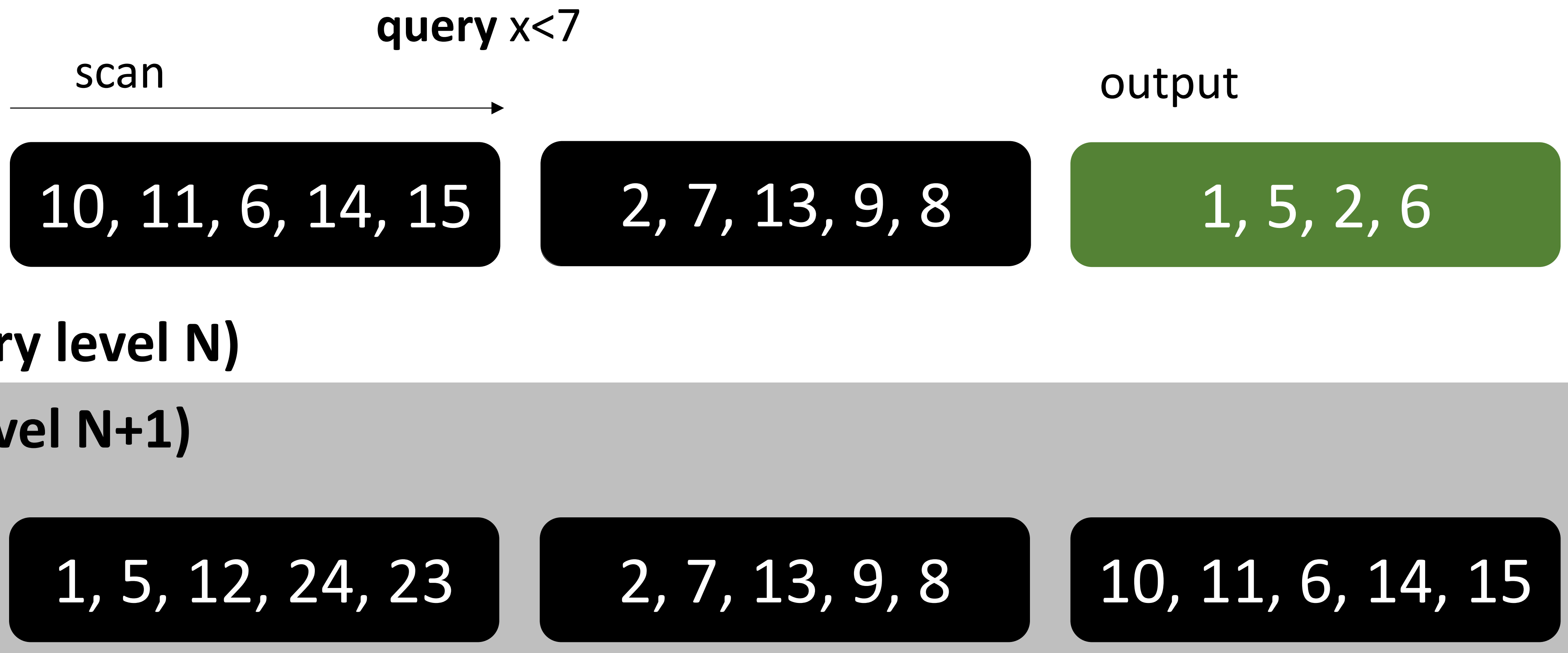
2, 7, 13, 9, 8

10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

$ 120 bytes

page-based access & random access



query x<7

scan

output

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

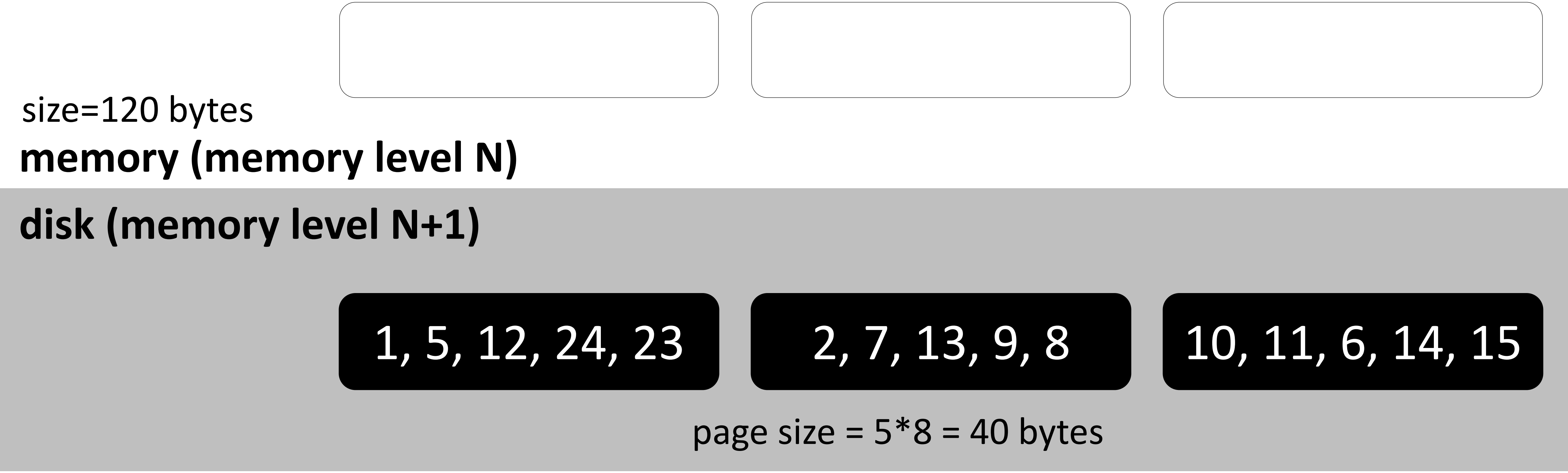
10, 11, 6, 14, 15

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

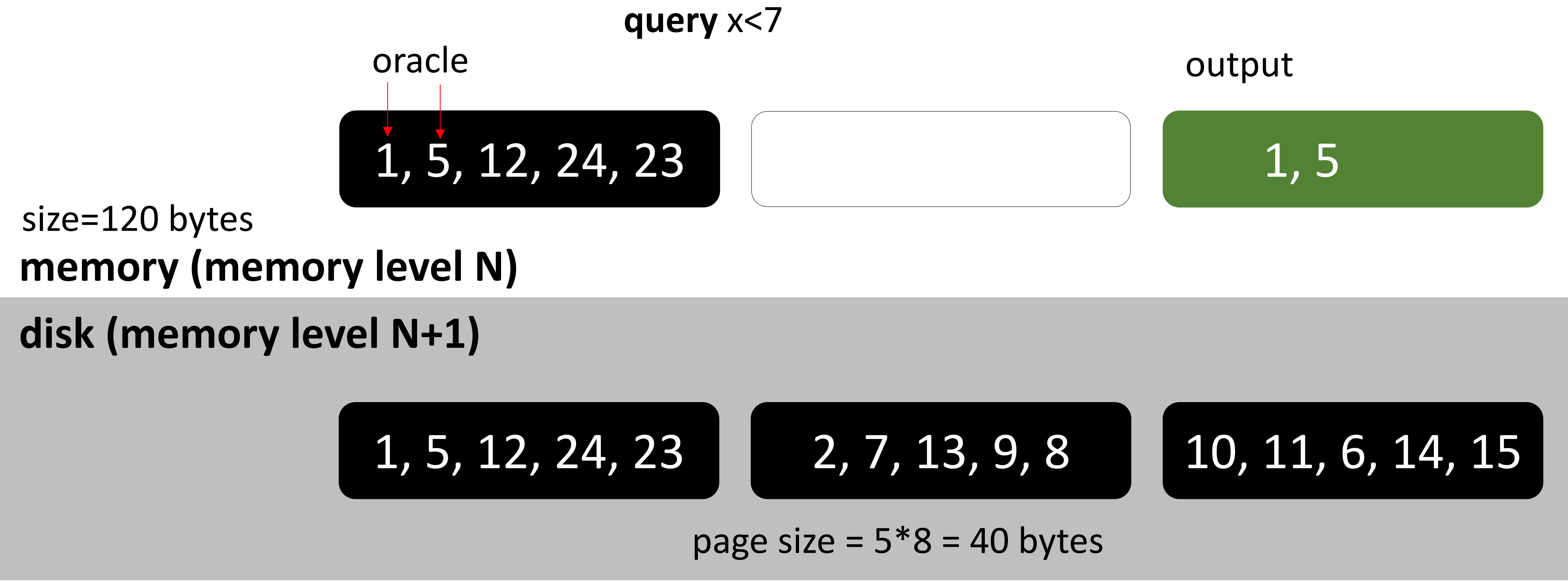
2, 7, 13, 9, 8

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

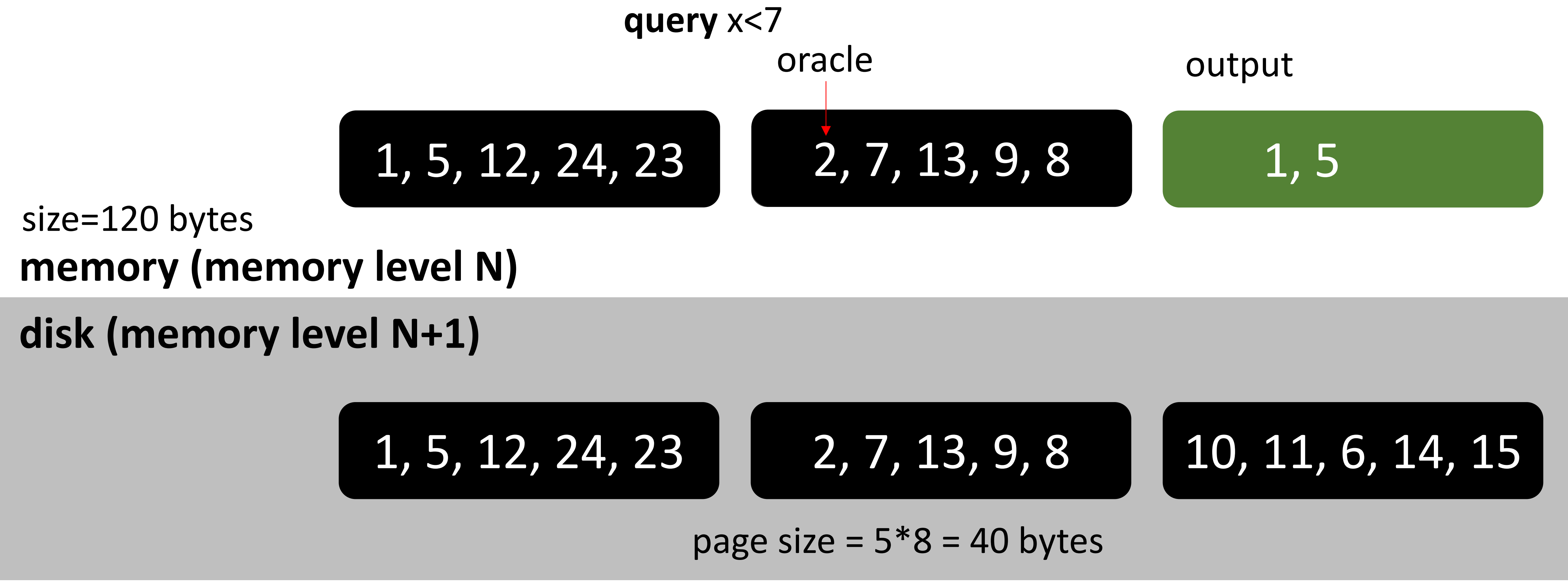
2, 7, 13, 9, 8

10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

page-based access & random access

$ 40 bytes



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

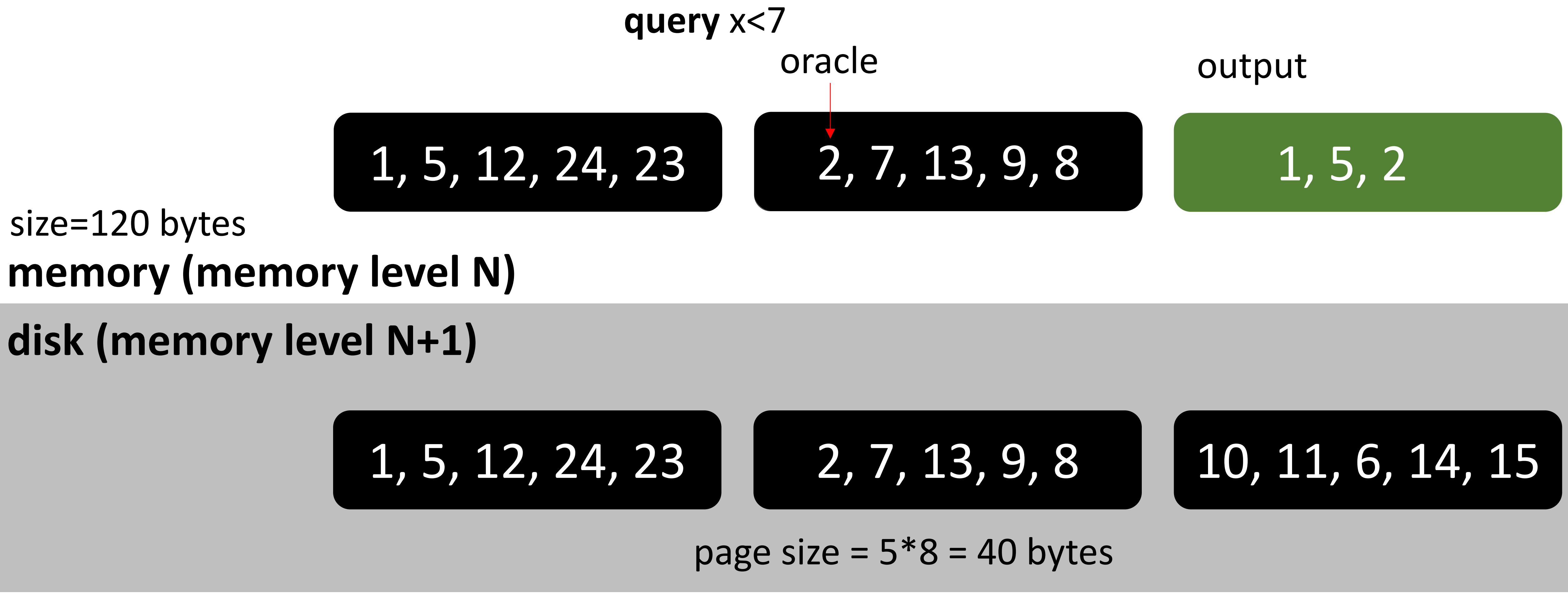
2, 7, 13, 9, 8

10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

page-based access & random access

$ 40 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

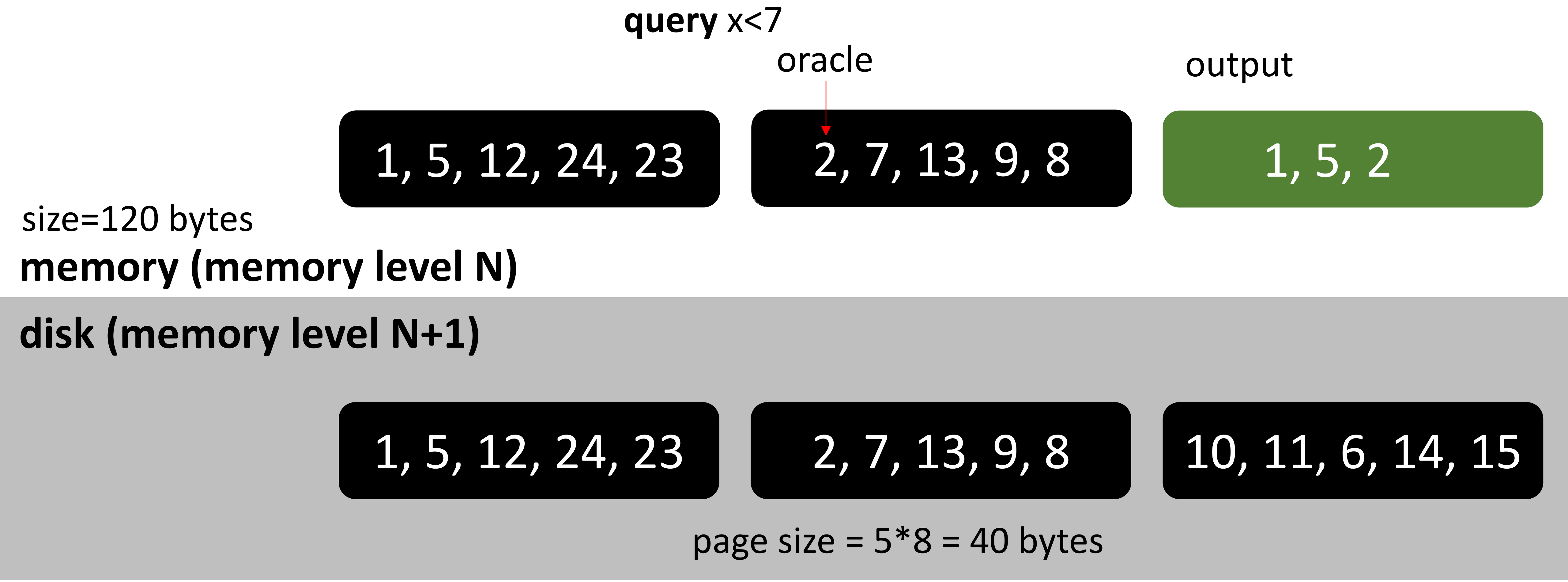
2, 7, 13, 9, 8

10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

$ 80 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

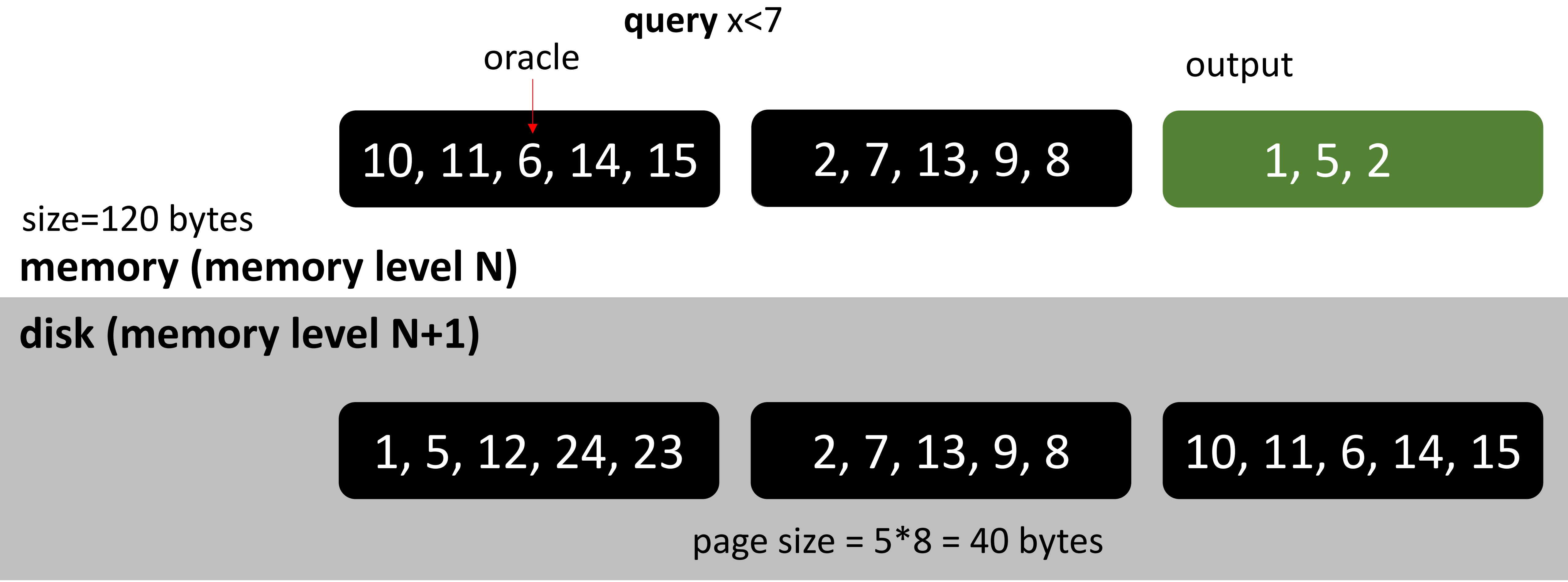
2, 7, 13, 9, 8

10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

page-based access & random access

$ 80 bytes



query x<7

oracle

output

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

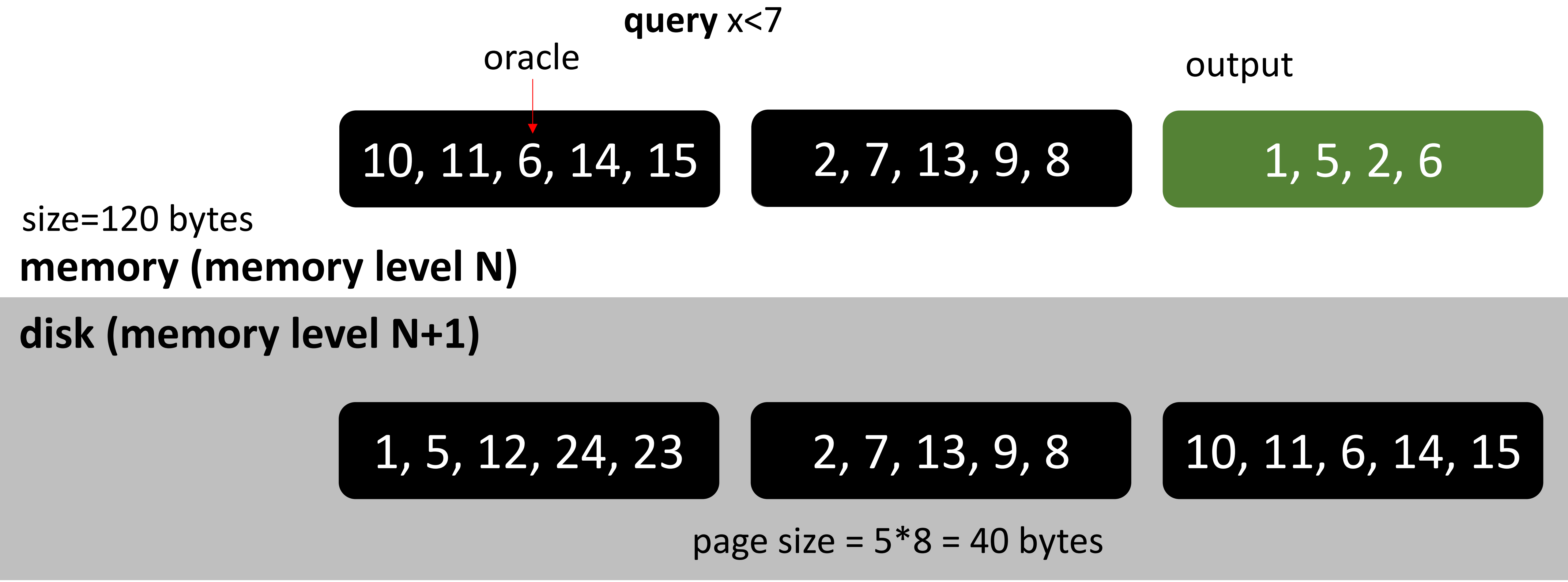
2, 7, 13, 9, 8

10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

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

$ 120 bytes

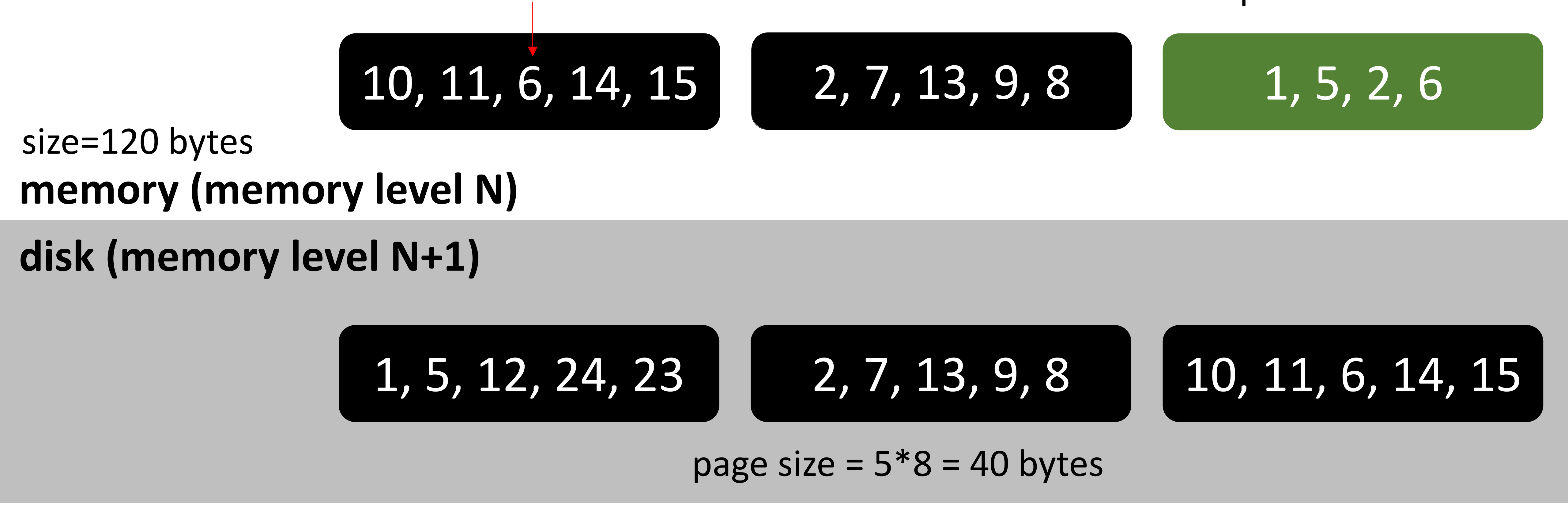
page-based access & random access

query x<7

oracle

was the oracle helpful?

output



when is the oracle helpful?

for which query would an oracle help us?

how to decide whether to use the oracle?

1, 5, 12, 24, 23

2, 7, 13, 9, 8

10, 11, 6, 14, 15

how we store data

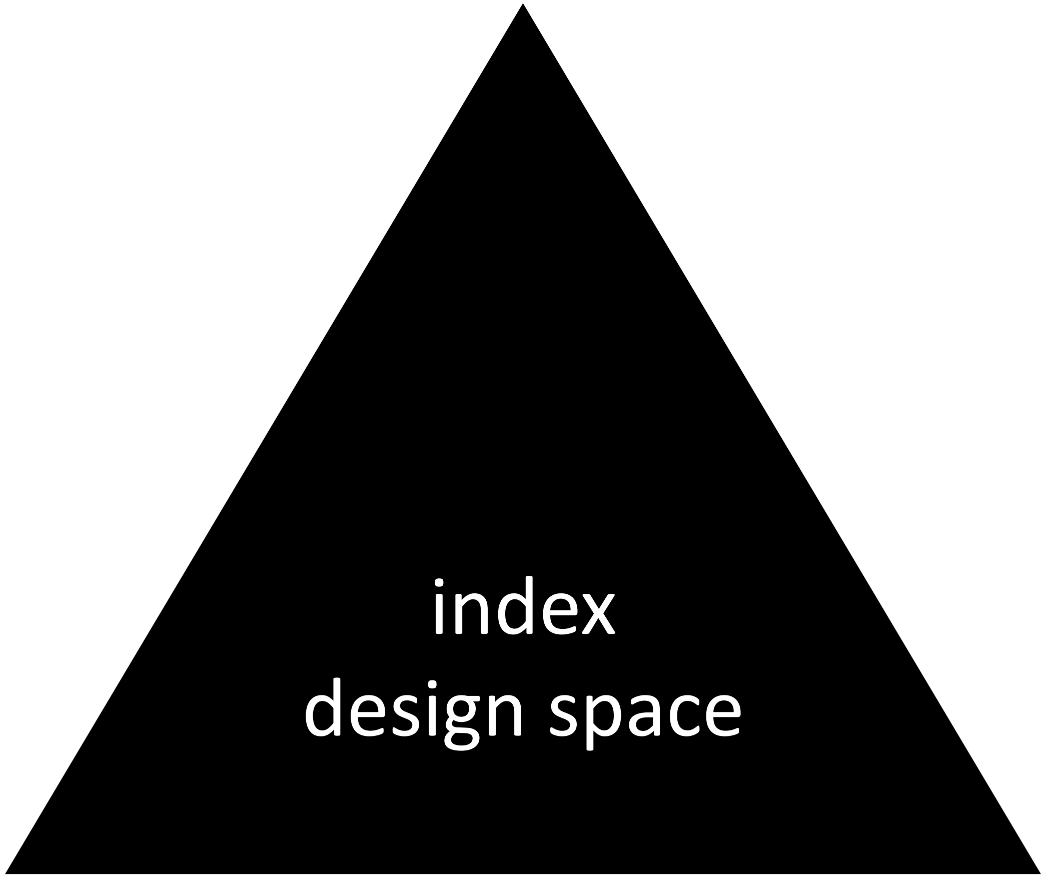
layouts, indexes

every **byte** counts

overheads and tradeoffs

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);

read random next;

ideal random access?

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

the language of efficient systems: C/C++

why?

low-level control over hardware

make decisions about physical data placement and consumptions

fewer assumptions

the language of efficient systems: C/C++

why?

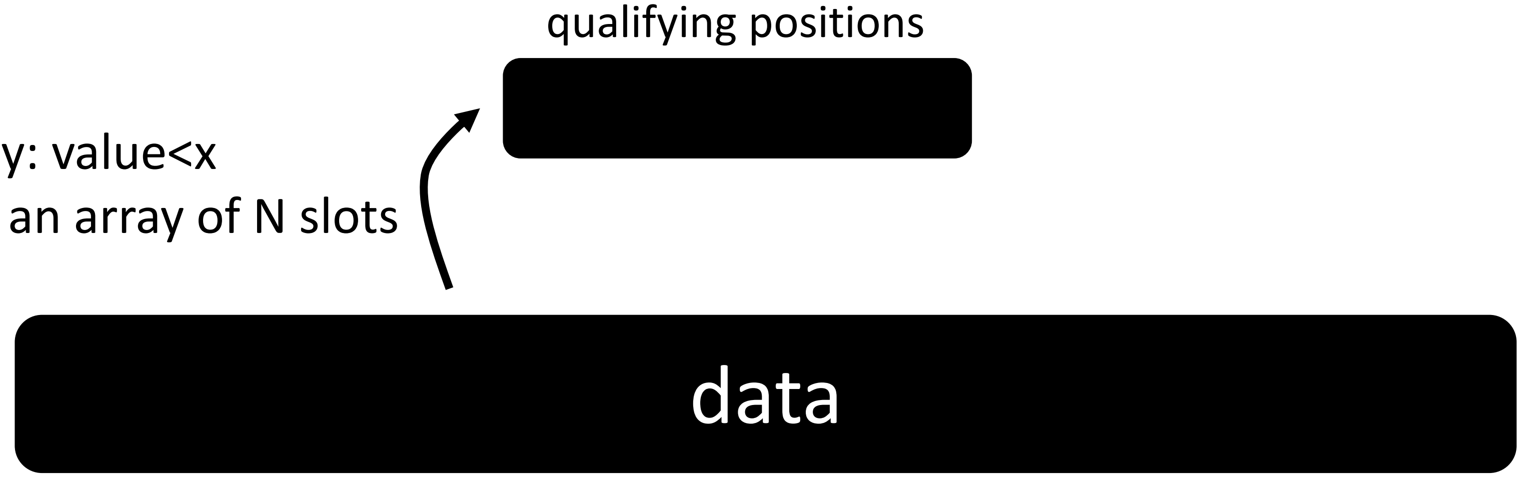
low-level control over hardware

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

a “simple” database operator

select operator (scan)

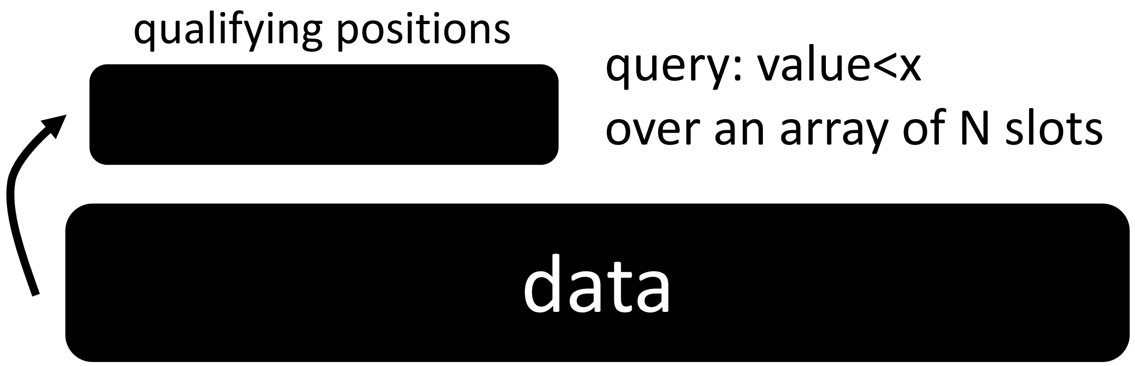
query: value<x
over an array of N slots



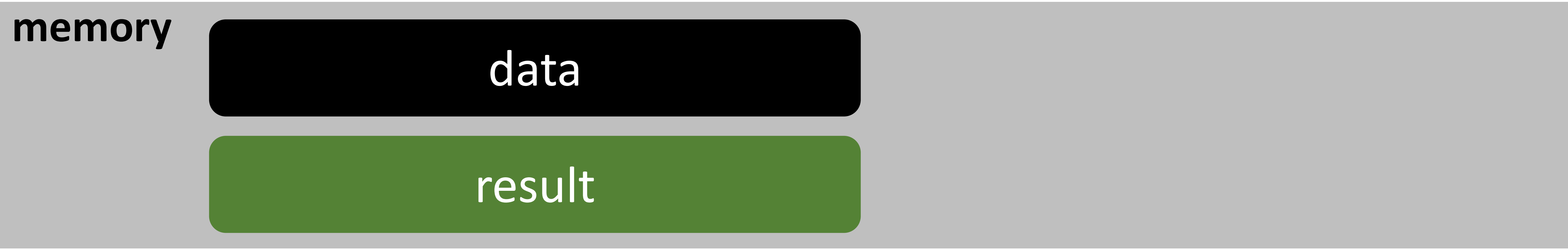
how to implement it?

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

what if only 0.1% qualifies?

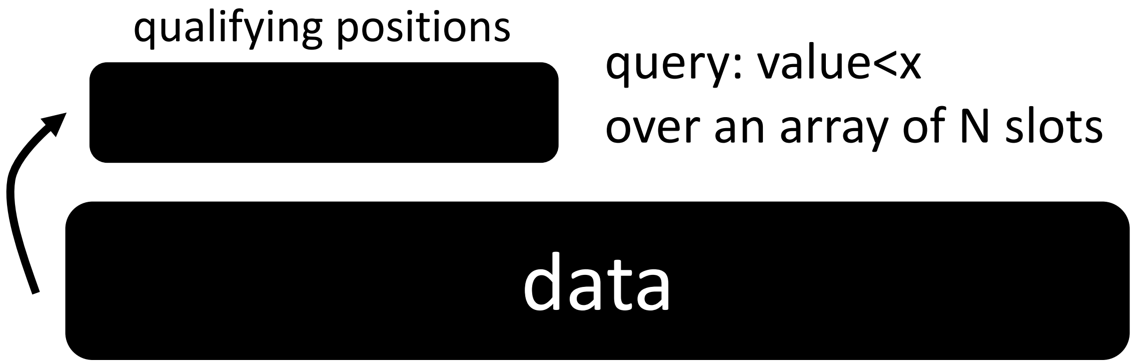


query: value<x
over an array of N slots



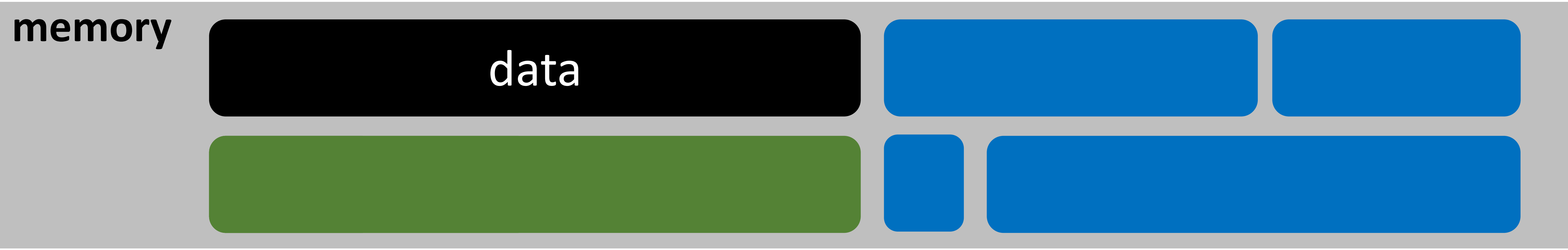
how to implement it?

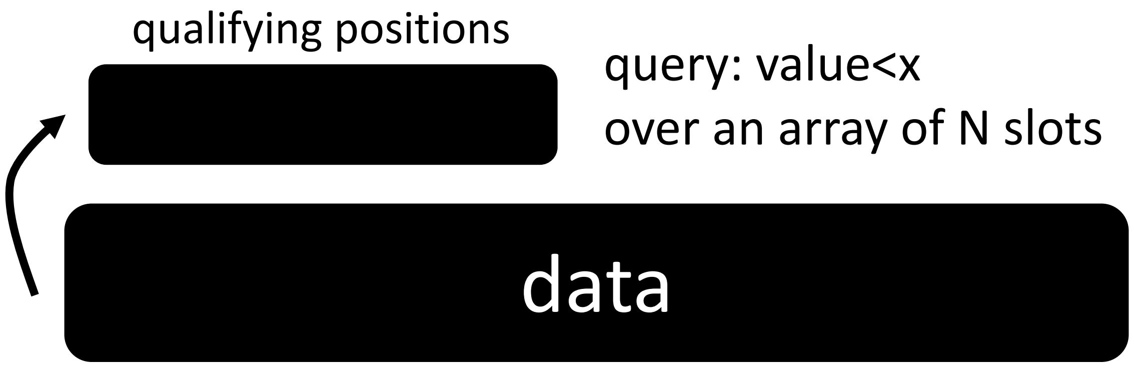
```
result = new array[data.size];
j=0;
for (i=0; i<data.size; i++)
    if (data[i]<x)
        result[j++]=i;
```



query: value<x
over an array of N slots

what if only 0.1% qualifies?





query: value<x
over an array of N slots

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;
```

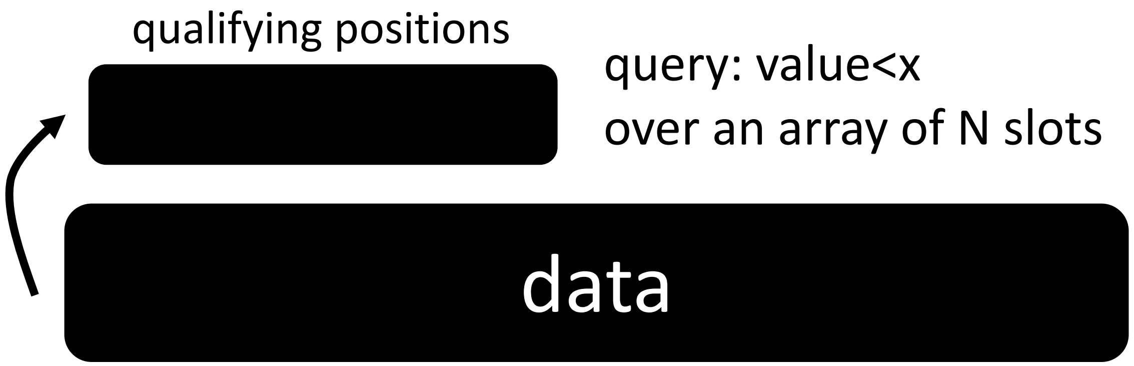
```
result = new array[data.size];
j=0;
for (i=0; i<data.size; i++)
    result[j+=(data[i]<x)]=i;
```

what if 99% qualifies?

how can we know?

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

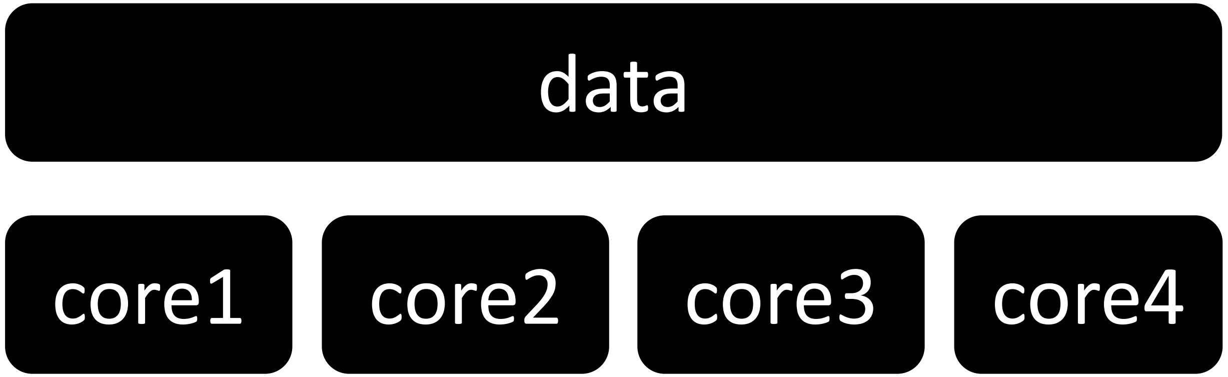
how to bring the values?
(remember we have the positions)



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?

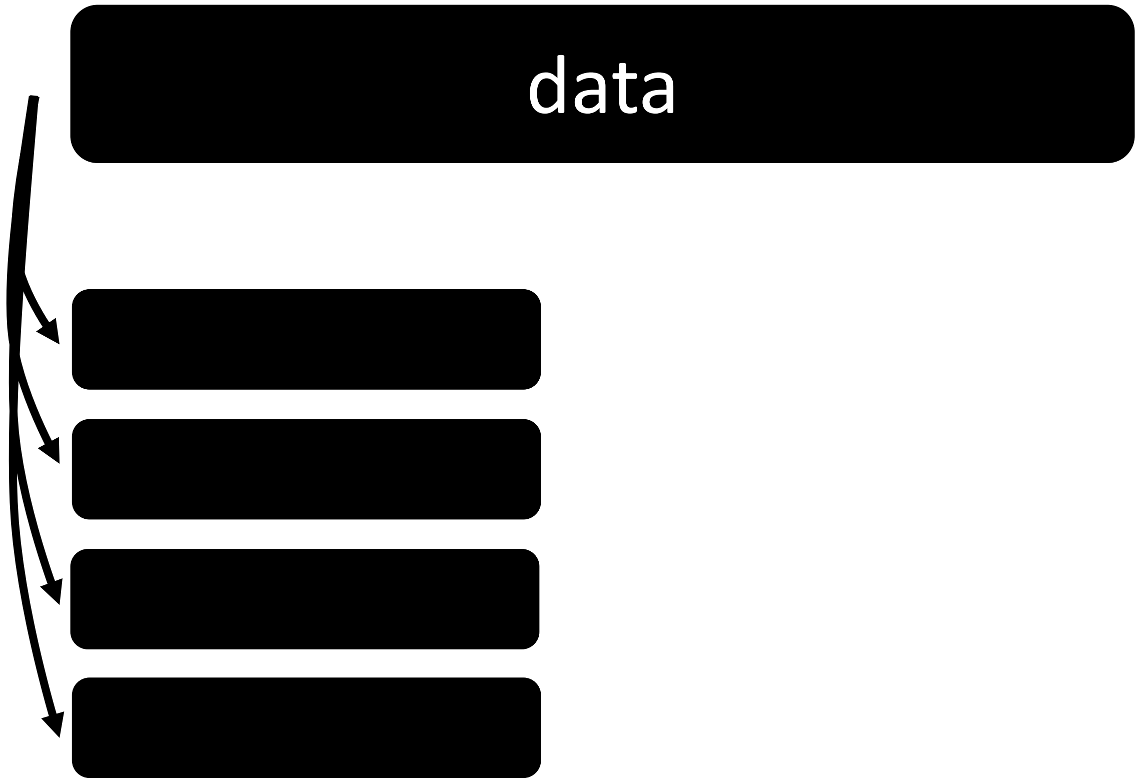


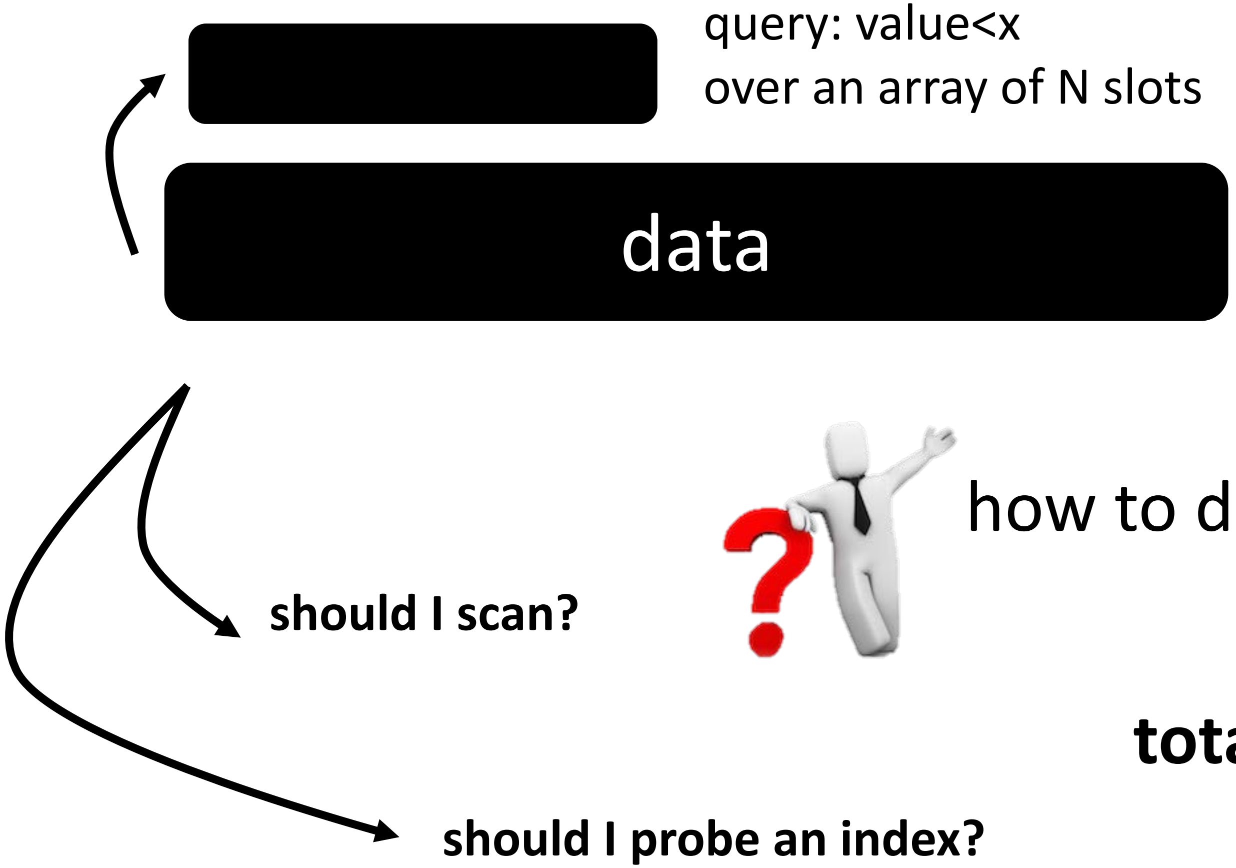
needs coordination!
what about result writing?

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

how can I prepare?

1) Read background research material

- **Architecture of a Database System**. By J. Hellerstein, M. Stonebraker and J. Hamilton. Foundations and Trends in Databases, 2007
- **The Design and Implementation of Modern Column-store Database Systems**. By D. Abadi, P. Boncz, S. Harizopoulos, S. Idreos, S. Madden. Foundations and Trends in Databases, 2013
- **Massively Parallel Databases and MapReduce Systems**. By Shivnath Babu and Herodotos Herodotou. Foundations and Trends in Databases, 2013

2) Start going over the papers

what to do now?

A) read the syllabus and the website

B) register to piazza

C) register to gradescope

D) register for the presentation (**early next week!**)

E) start submitting paper reviews (week 3)

F) go over the project (next week will be available)

G) start working on the proposal (week 3)

survival guide

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

piazza website: https://piazza.com/bu/spring2021/cs561

presentation registration: https://tinyurl.com/S21-CS561-presentations

gradescope: https://www.gradescope.com/courses/236591 (**2RBY82**)

office hours: Manos (T/Th 2-3pm)
Papon, Aneesh, Ju Hyoung (see in Piazza)

material: papers available from BU network

BOSTON UNIVERSITY

CS 561: Data Systems Architectures

class 2

Data Systems 101

next week: modern main-memory data systems

&

semester project