BOSTON UNIVERSITY

# CS 591: Data Systems Architectures

## class 2

# Data Systems 101

Prof. Manos Athanassoulis

https://midas.bu.edu/classes/CS591A1/

# some reminders

no smartphones 

no laptop 

# class summary

2 classes per week / OH 5 days per week

**each student**

1 presentation/discussion lead + 2 reviews/questions per week

systems or research project + proposal + mid-semester report

## systems project

implementation-heavy C/C++ project

groups of 2

## research project

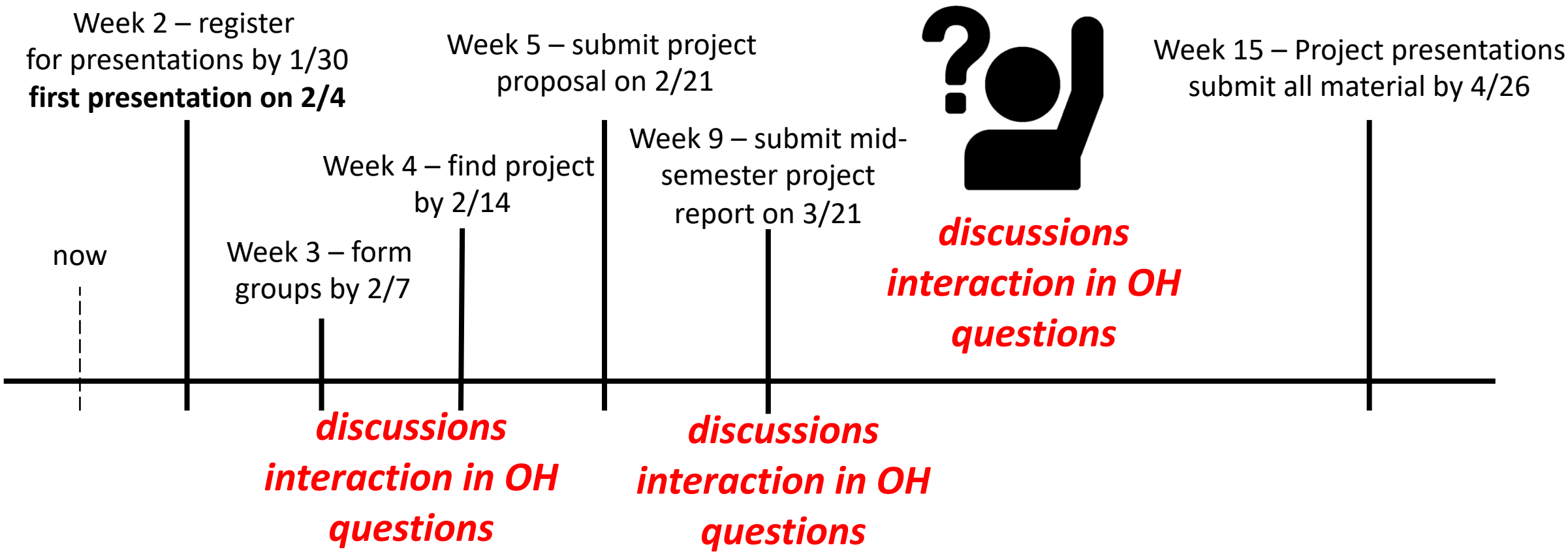
groups of 3

pick a subject (list will be available)

design & analysis

experimentation

# class timeline



# Piazza

all discussions & announcements

http://piazza.com/bu/spring2020/cs591a1/

also available on class website

10 already registered!

register so we can reach you easily

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

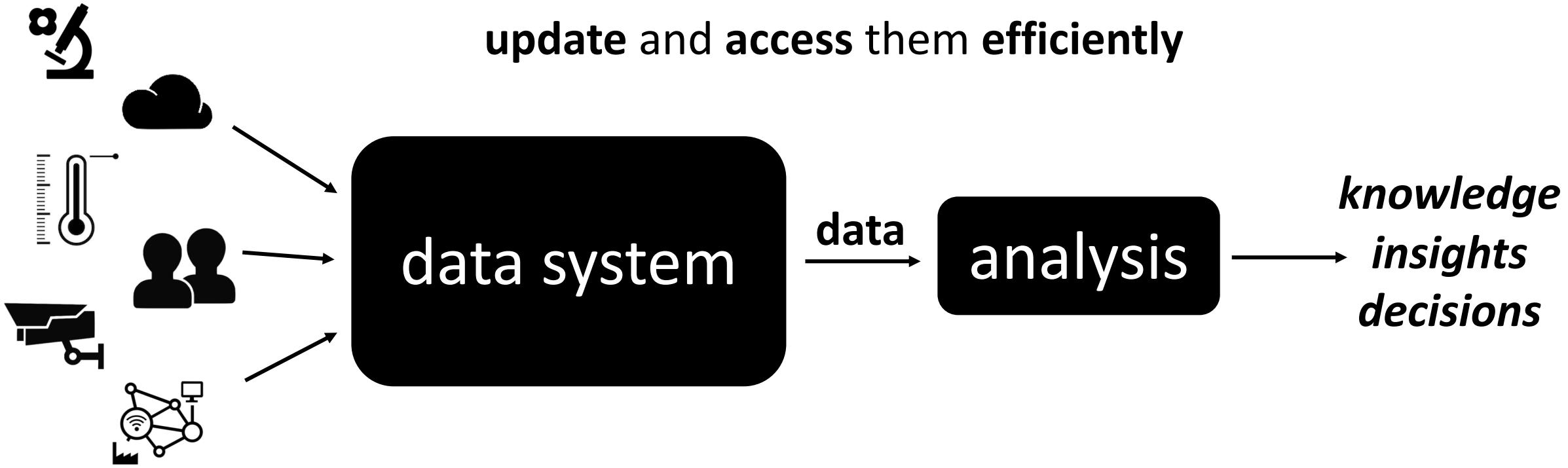
 sensors

social 

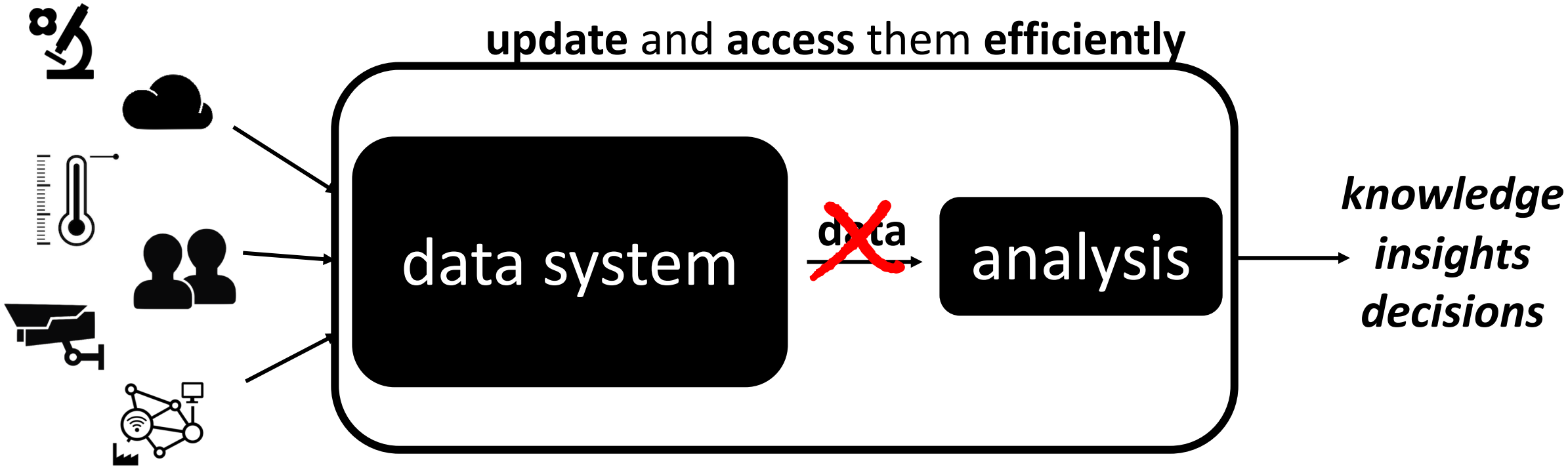
Internet-of-Things 

BOSTON UNIVERSITY

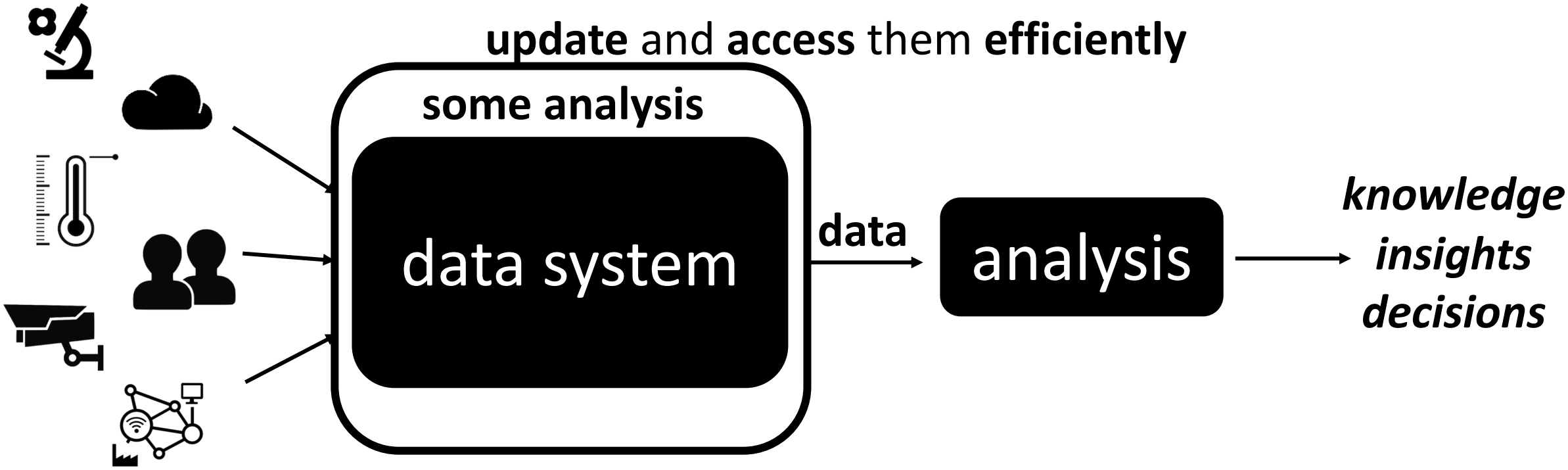
a **data system** is a large software system
that **stores data**, and provides the **interface** to
**update** and **access** them **efficiently**



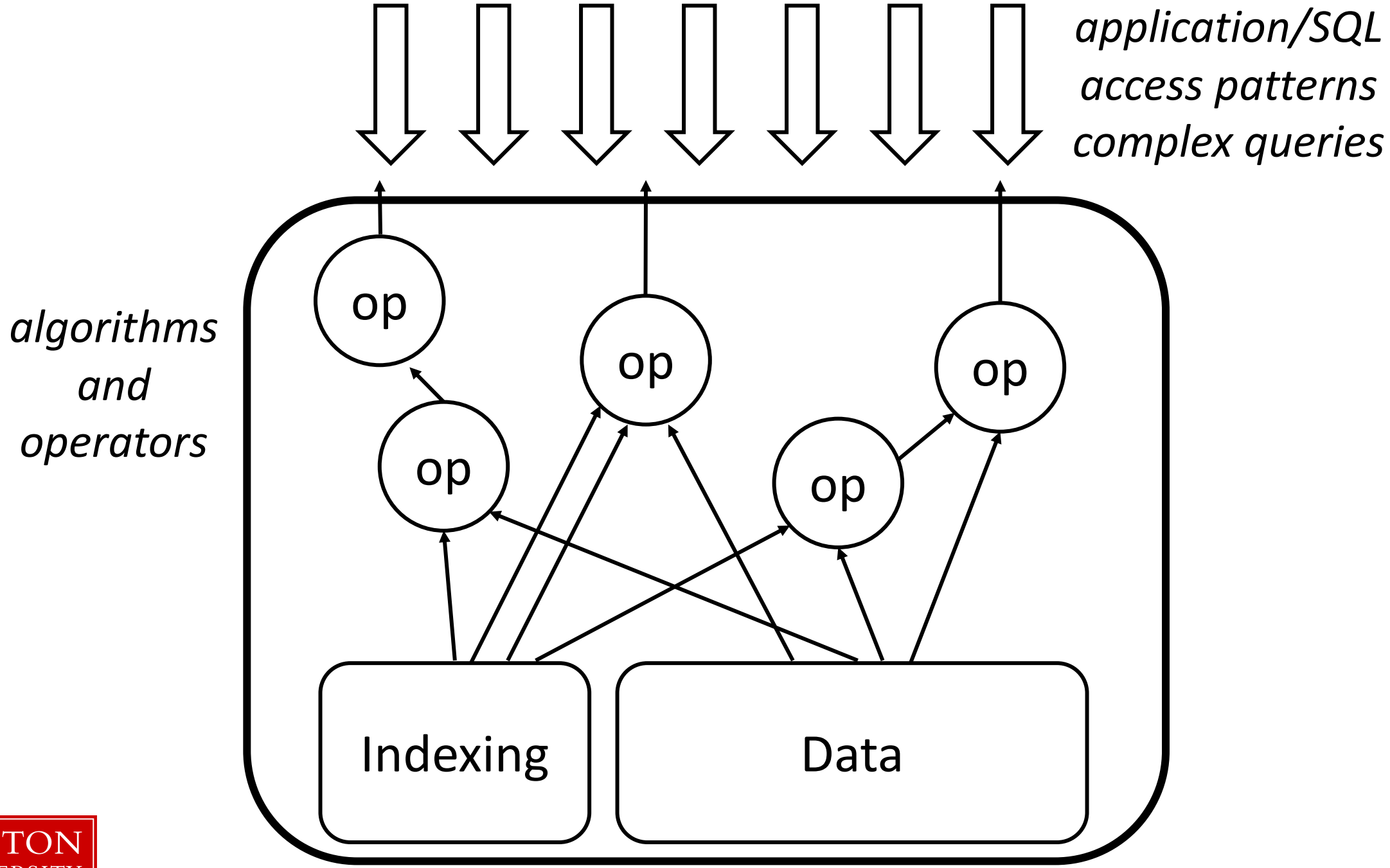
a **data system** is a large software system
that **stores data**, and provides the **interface** to
**update** and **access** them **efficiently**

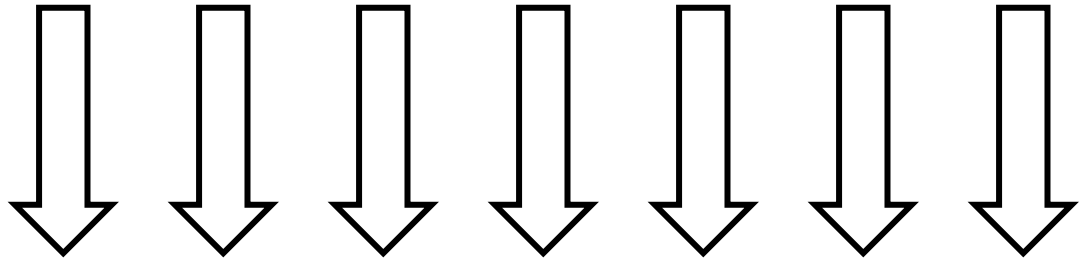


a **data system** is a large software system
that **stores data**, and provides the **interface** to
**update** and **access** them **efficiently**



# data system, what’s inside?





*application/SQL*
*access patterns*
*complex queries*

*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

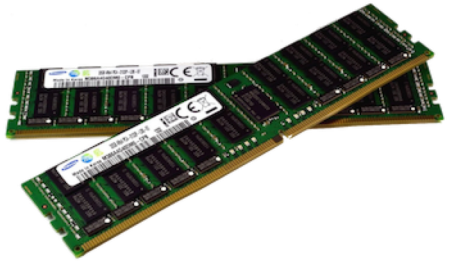
[Forrester, 2016]

BOSTON UNIVERSITY

# *growing need for tailored systems*

new applications

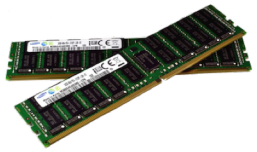
new hardware

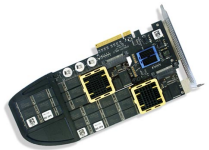


more data

# data system, what’s underneath?

# memory hierarchy



CPU

on-chip cache

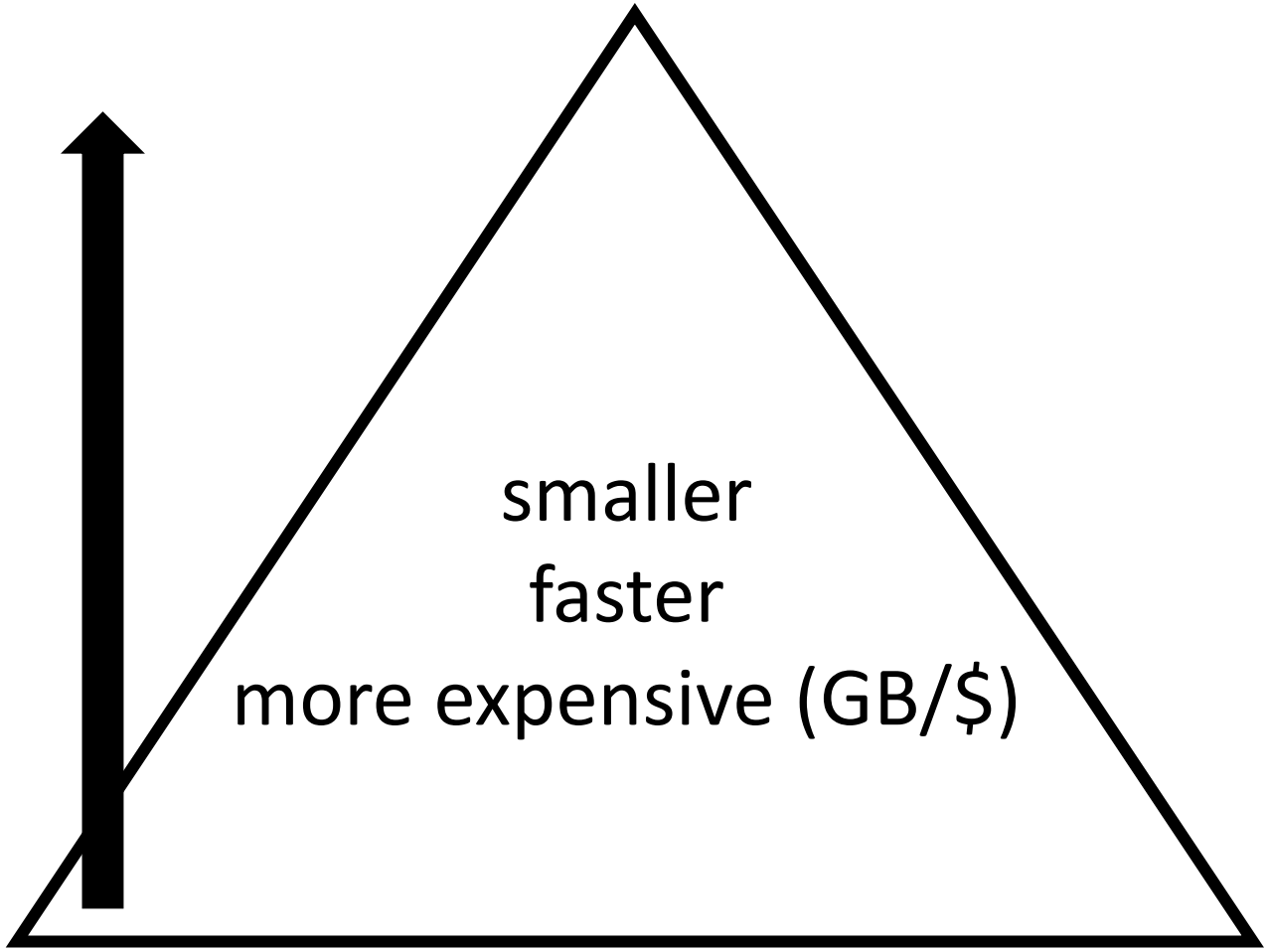
on-board cache

main memory

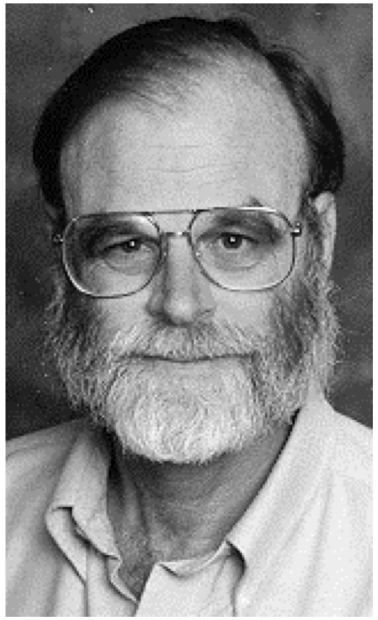
flash storage

disks

flash



# memory hierarchy (by Jim Gray)



Jim Gray, IBM, Tandem, Microsoft, DEC

"The Fourth Paradigm" is based on his vision

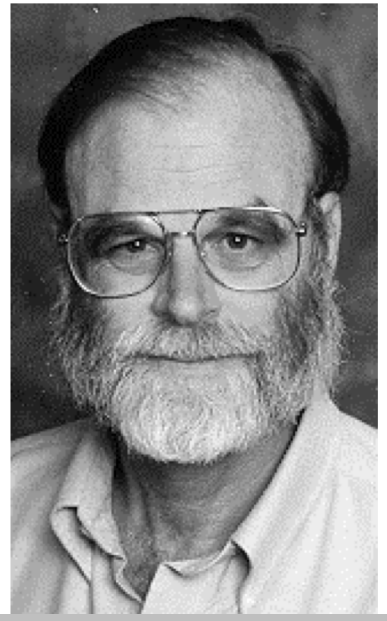
**ACM Turing Award 1998**

**ACM SIGMOD Edgar F. Codd Innovations award 1993**

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

BOSTON UNIVERSITY

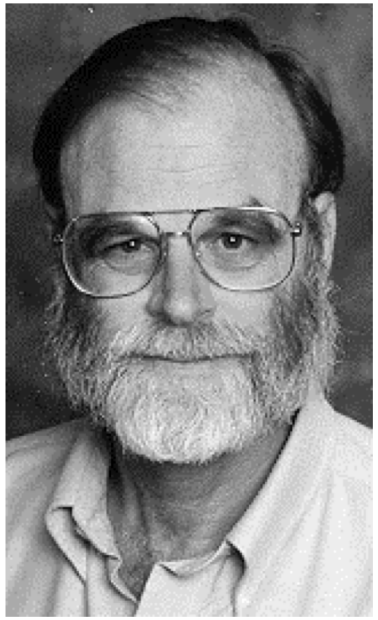
# memory hierarchy (by Jim Gray)

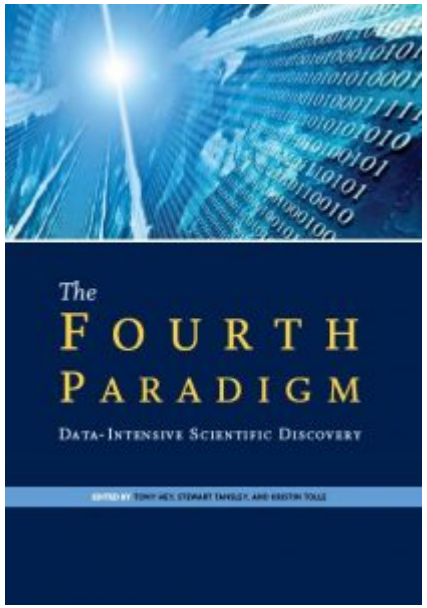


| | | |
|---|---|---|
| | **registers/CPU** | my head<br>~0 |
| **2x** | **on chip cache** | this room<br>1min |
| **10x** | **on board cache** | this building<br>10min |

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

# Jim Gray (a great scientist and engineer)






Jim Gray, IBM, Tandem, Microsoft, DEC

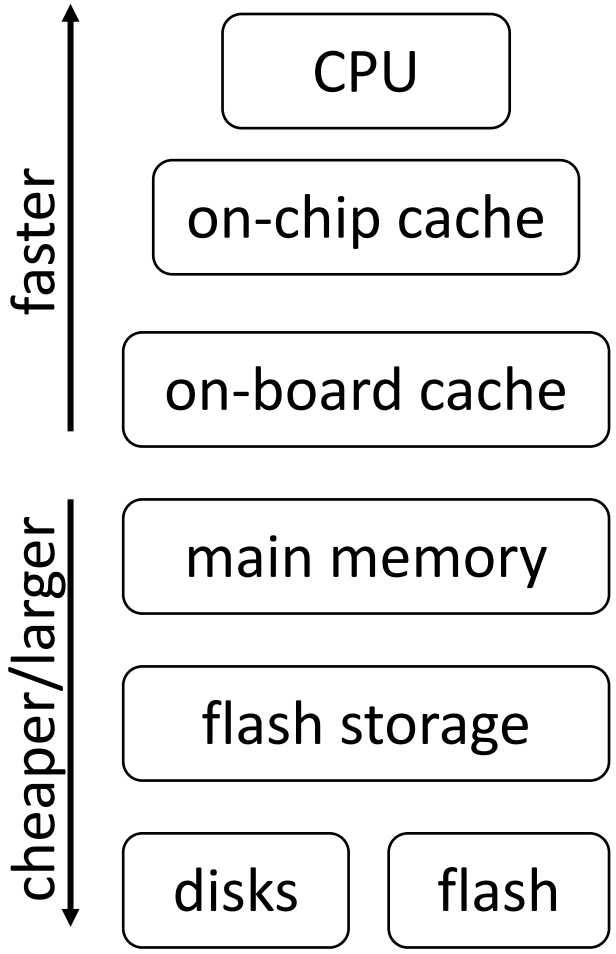
"The Fourth Paradigm" is based on his vision

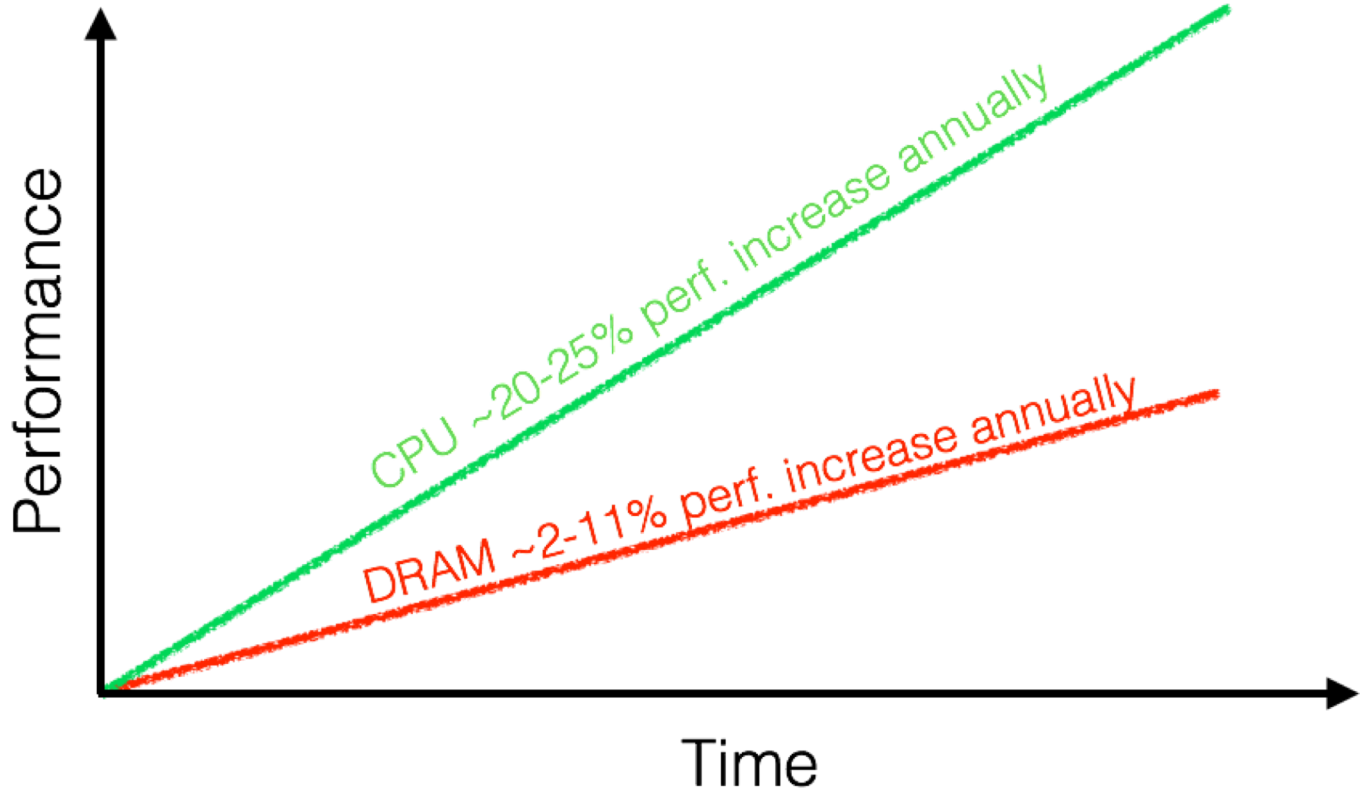
**ACM Turing Award 1998**

**ACM SIGMOD Edgar F. Codd Innovations award 1993**

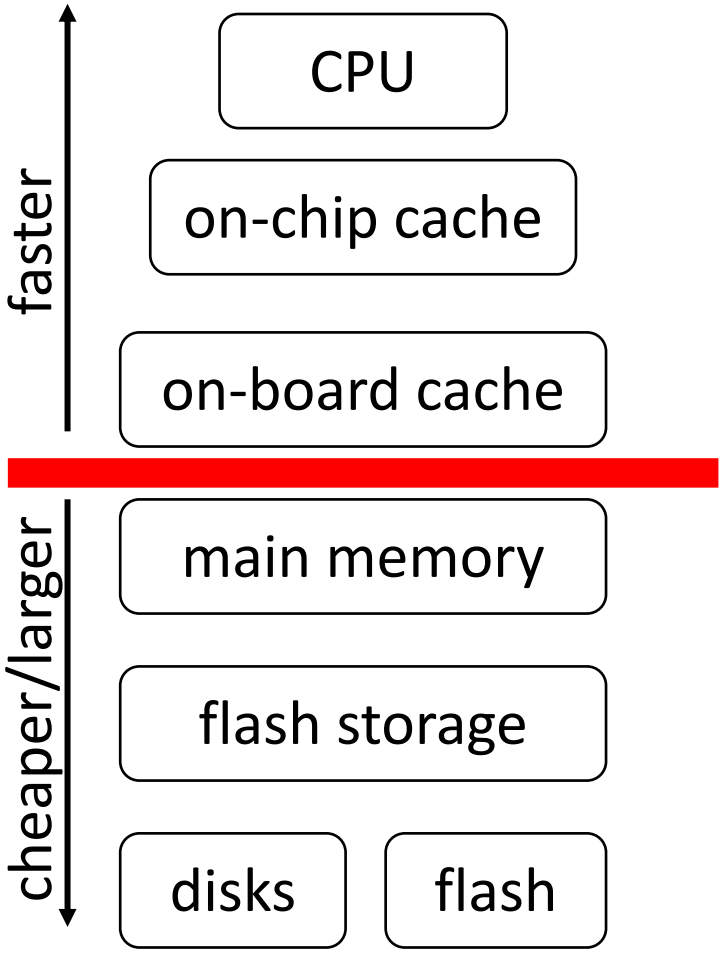
*the first collection of*
*technical visionary research on*
*a data-intensive scientific discovery*

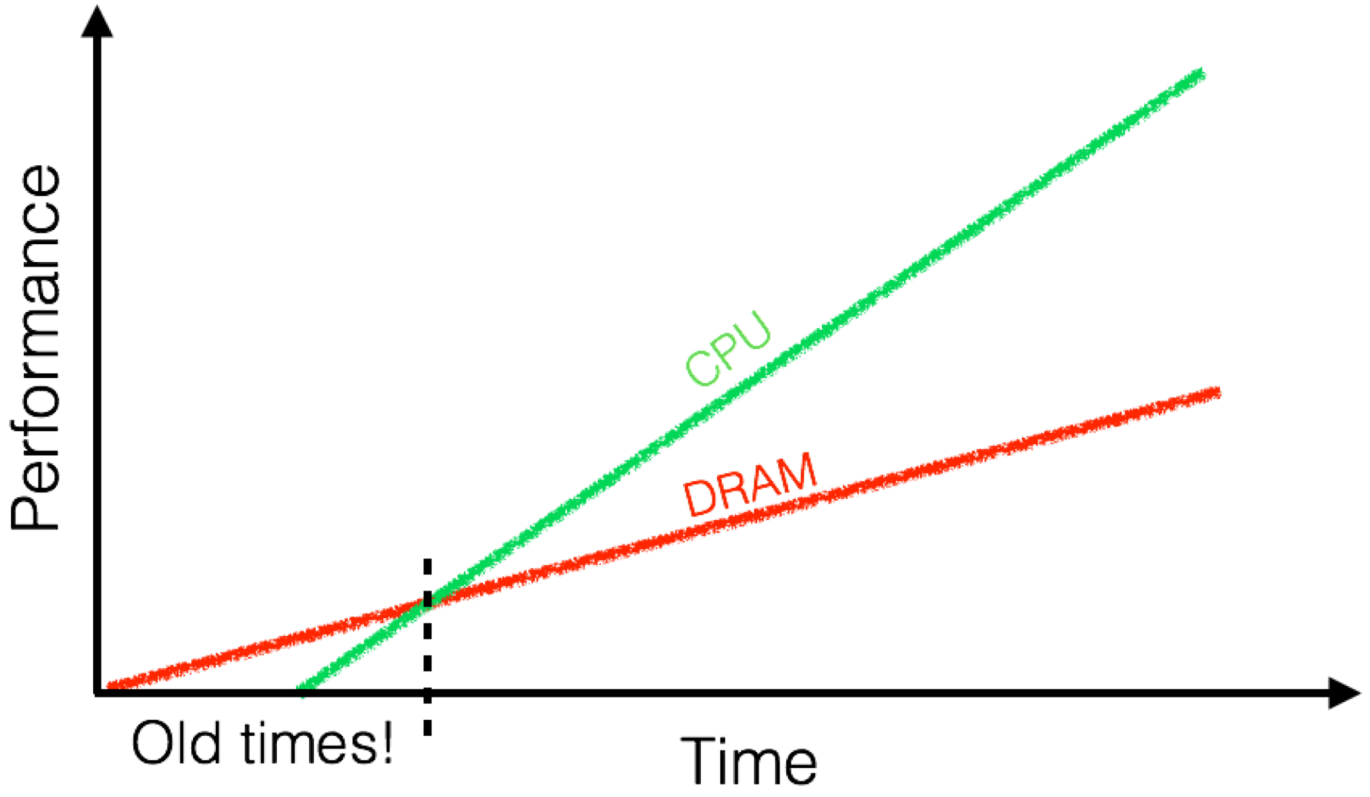
# memory wall



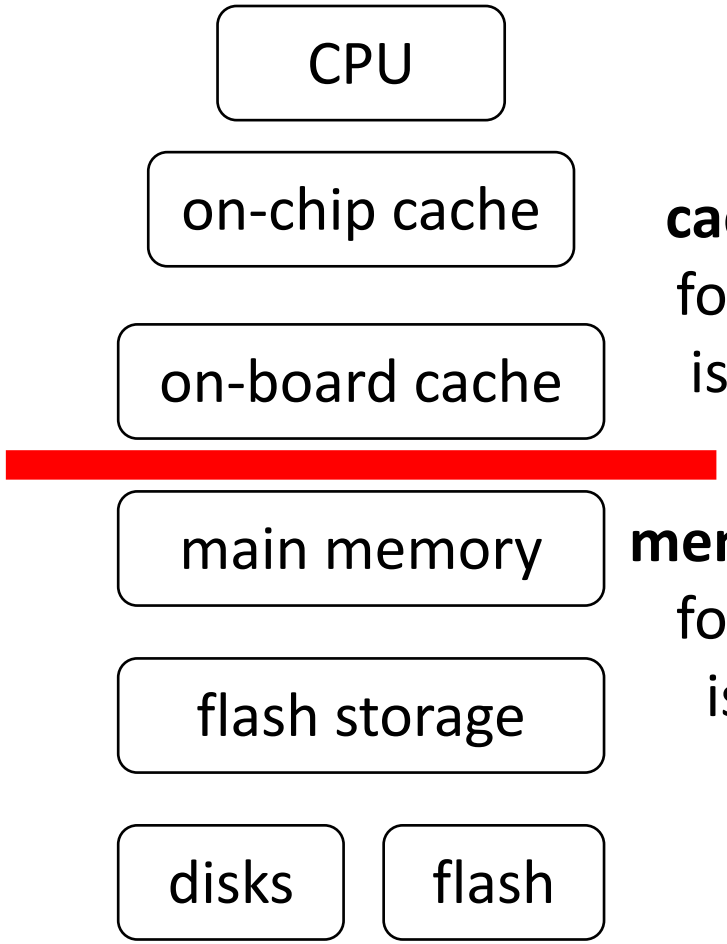



# memory wall



# cache/memory misses

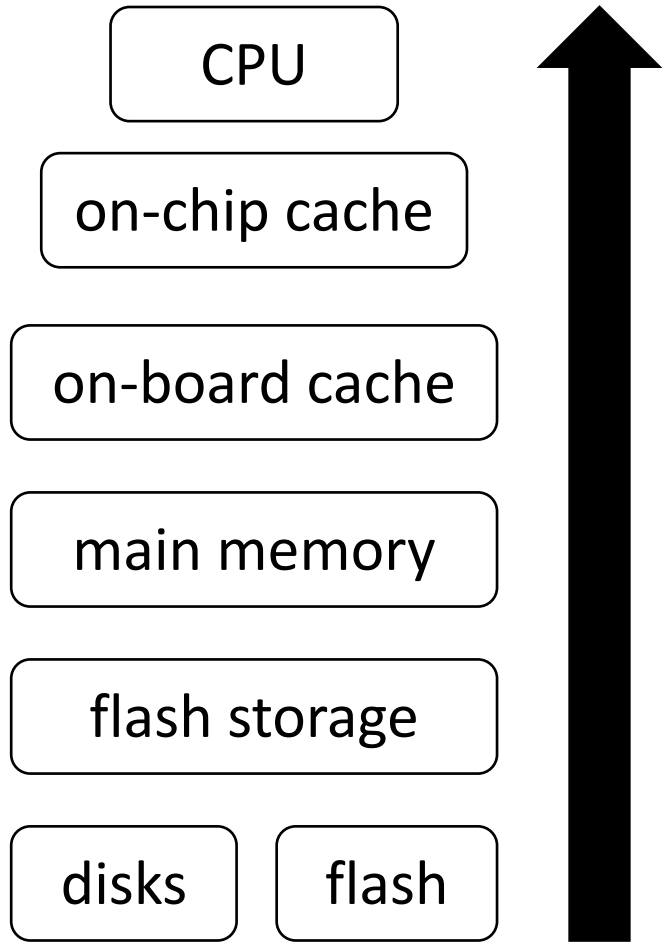



**cache miss**: looking for something that is not in the cache

**memory miss**: looking for something that is not in memory

**what happens if I miss?**

# data movement




data go through
all necessary levels

also read
***unnecessary*** data

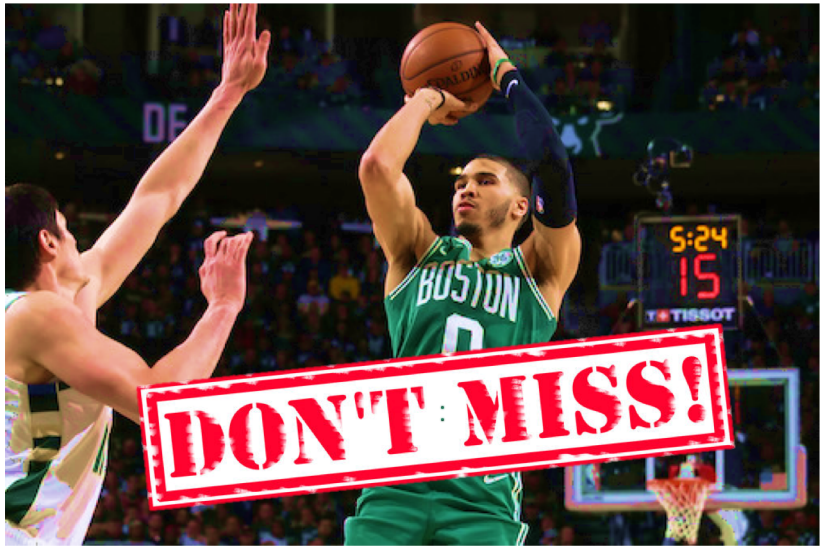



*Photo by Gary Dineen/NBAE via Getty Images*

need to read only X
read the whole page

# data movement

CPU

on-chip cache

on-board cache

main memory

flash storage

data go through
all necessary levels

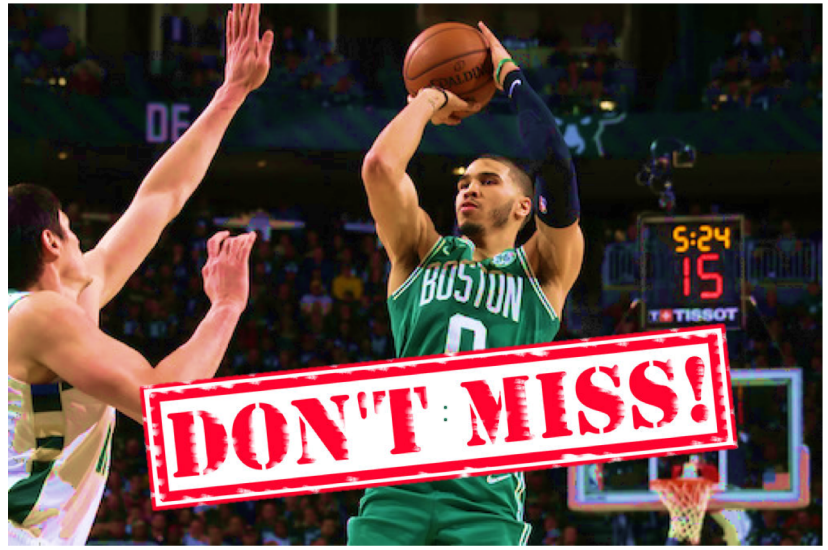



*Photo by Gary Dineen/NBAE via Getty Images*

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

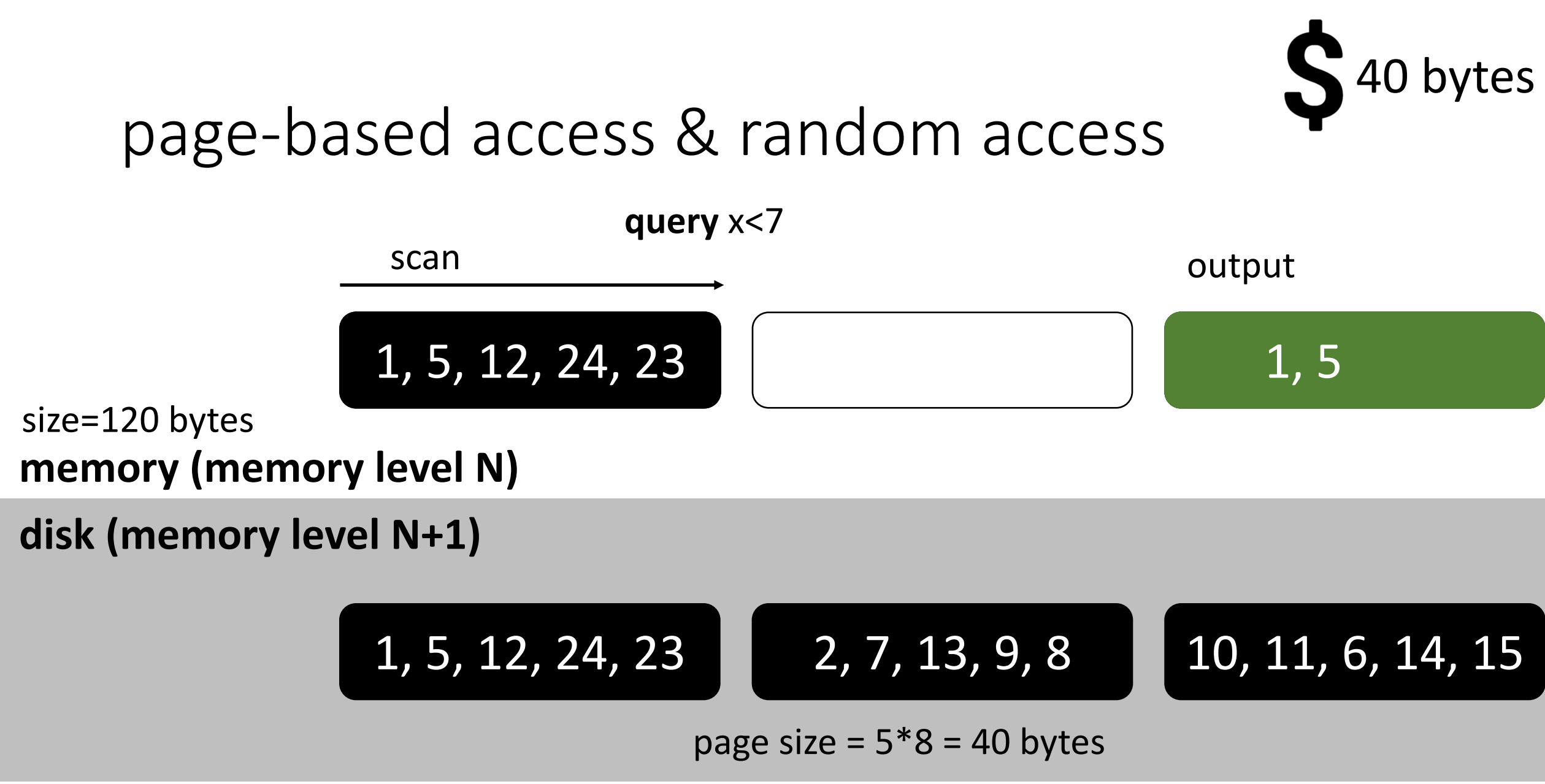
**memory (memory level N)**

**disk (memory level N+1)**

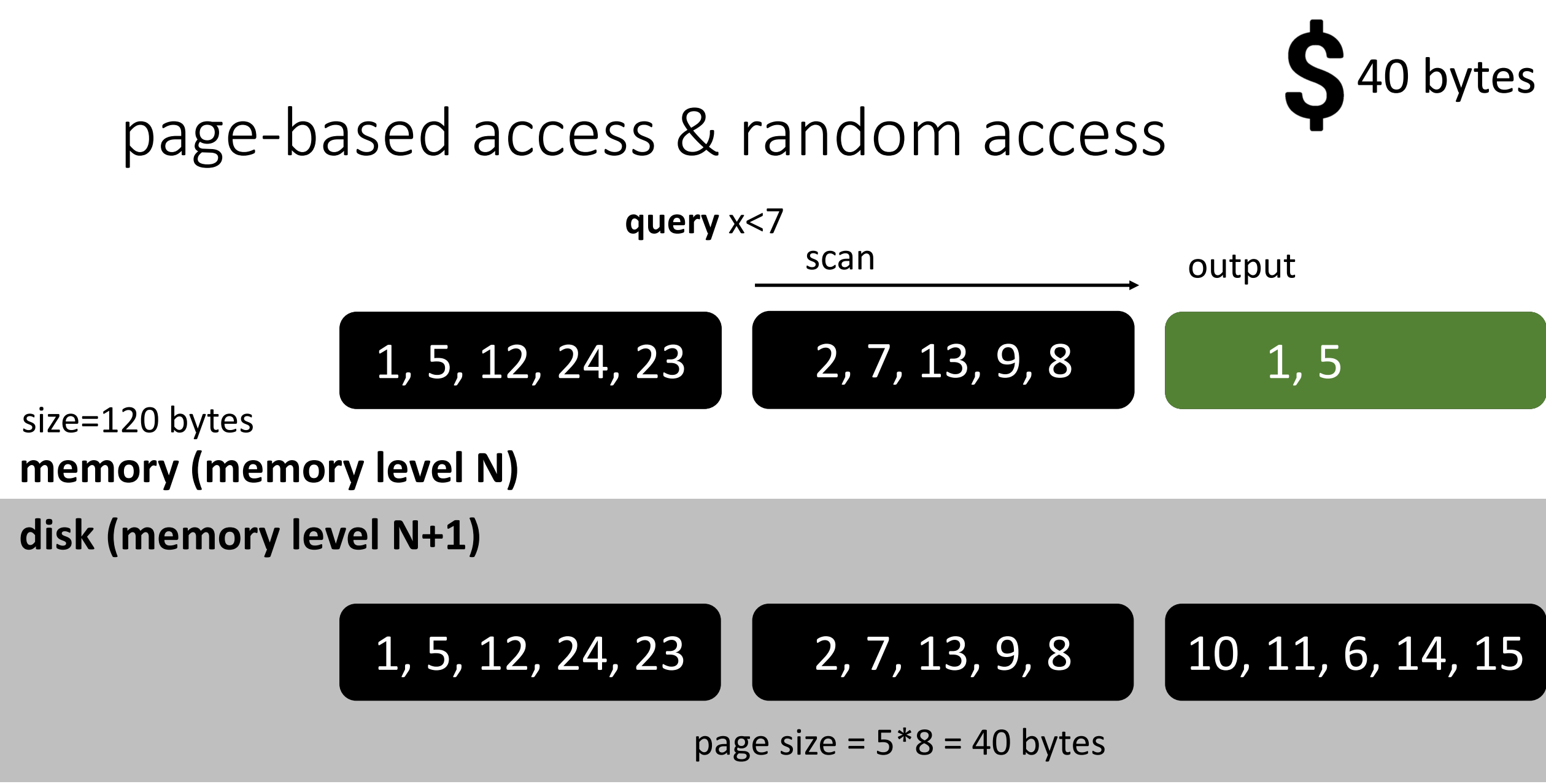
1, 5, 12, 24, 23 | 2, 7, 13, 9, 8 | 10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

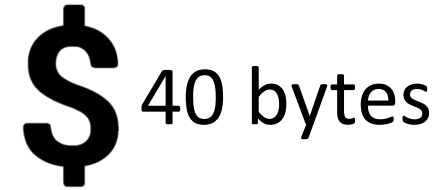
# page-based access & random access



# page-based access & random access



# 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)**

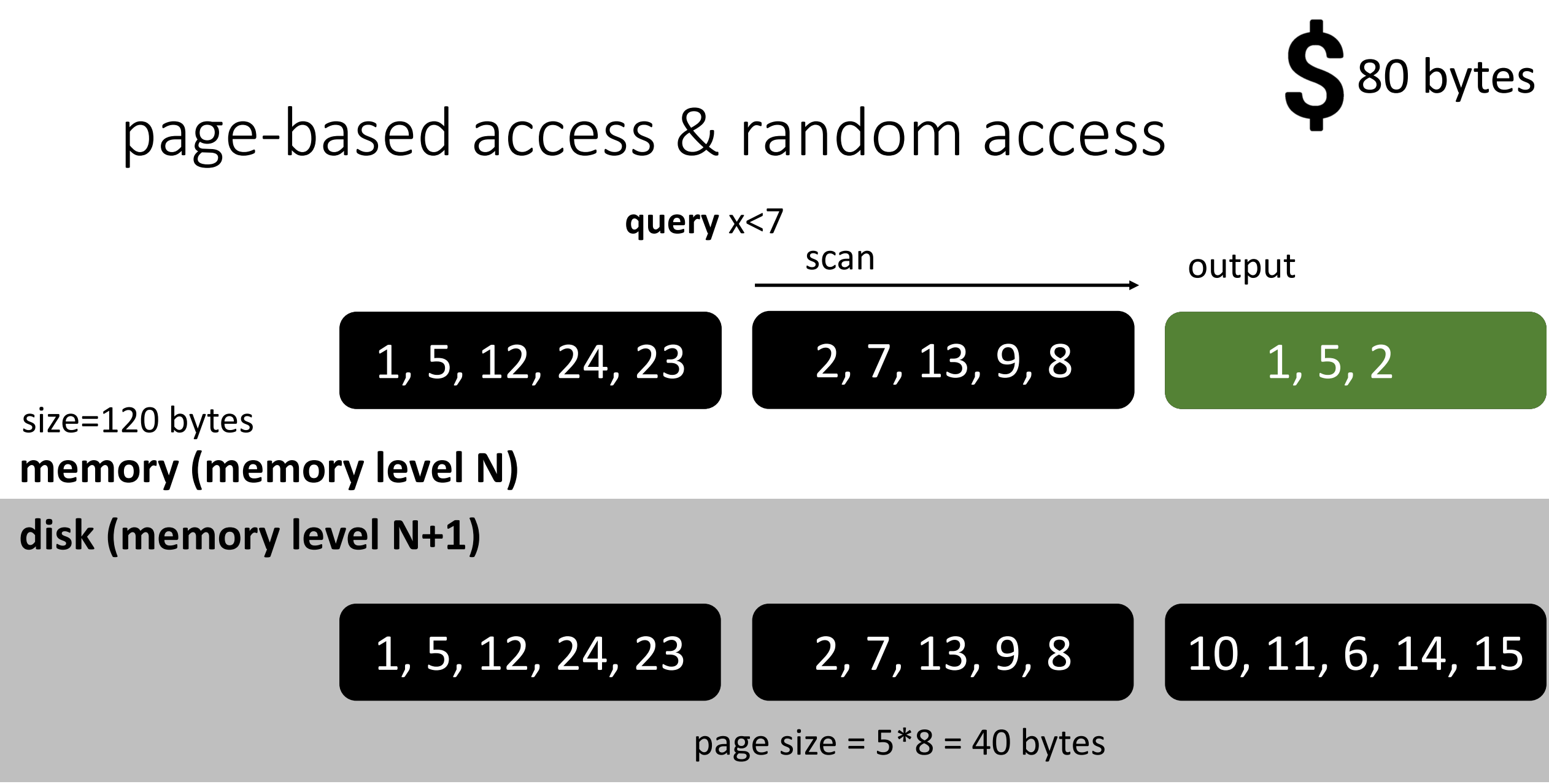
1, 5, 12, 24, 23

2, 7, 13, 9, 8

10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

# page-based access & random access




# page-based access & random access

**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)**

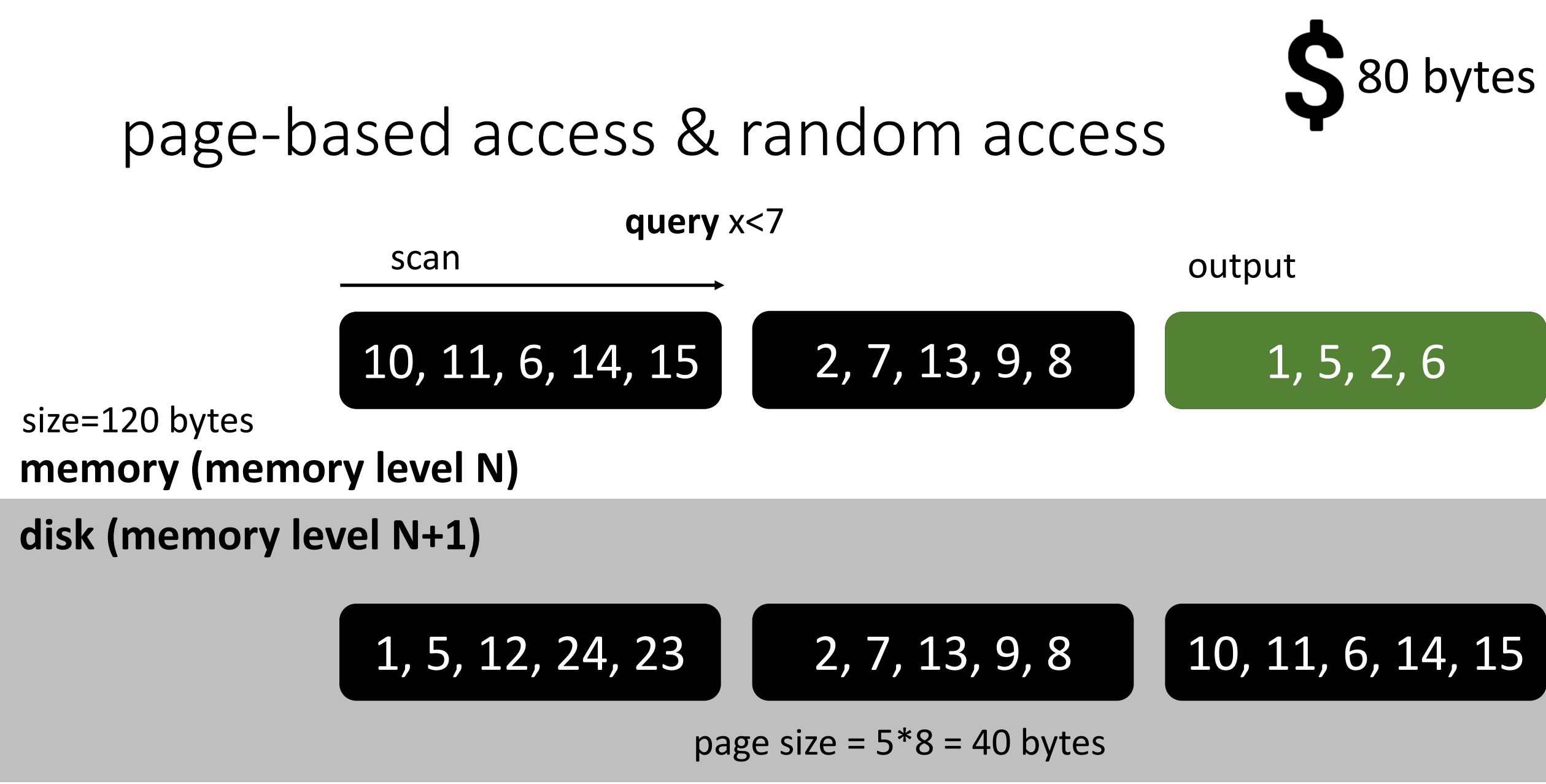
1, 5, 12, 24, 23

2, 7, 13, 9, 8

10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

# page-based access & random access




# 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

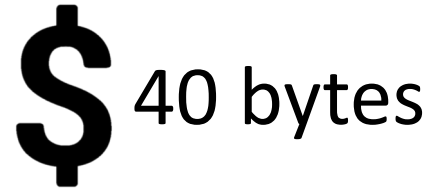
**memory (memory level N)**

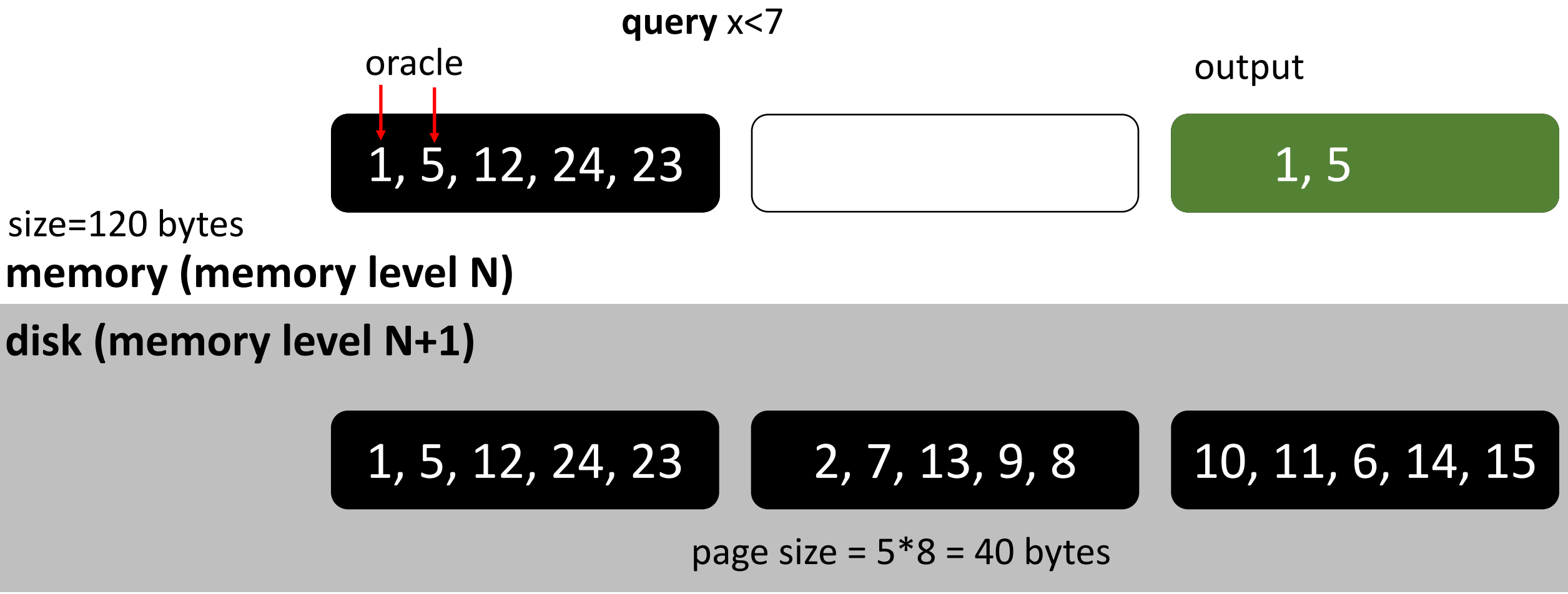
**disk (memory level N+1)**

1, 5, 12, 24, 23 | 2, 7, 13, 9, 8 | 10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

# page-based access & random access



# 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

# 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)**

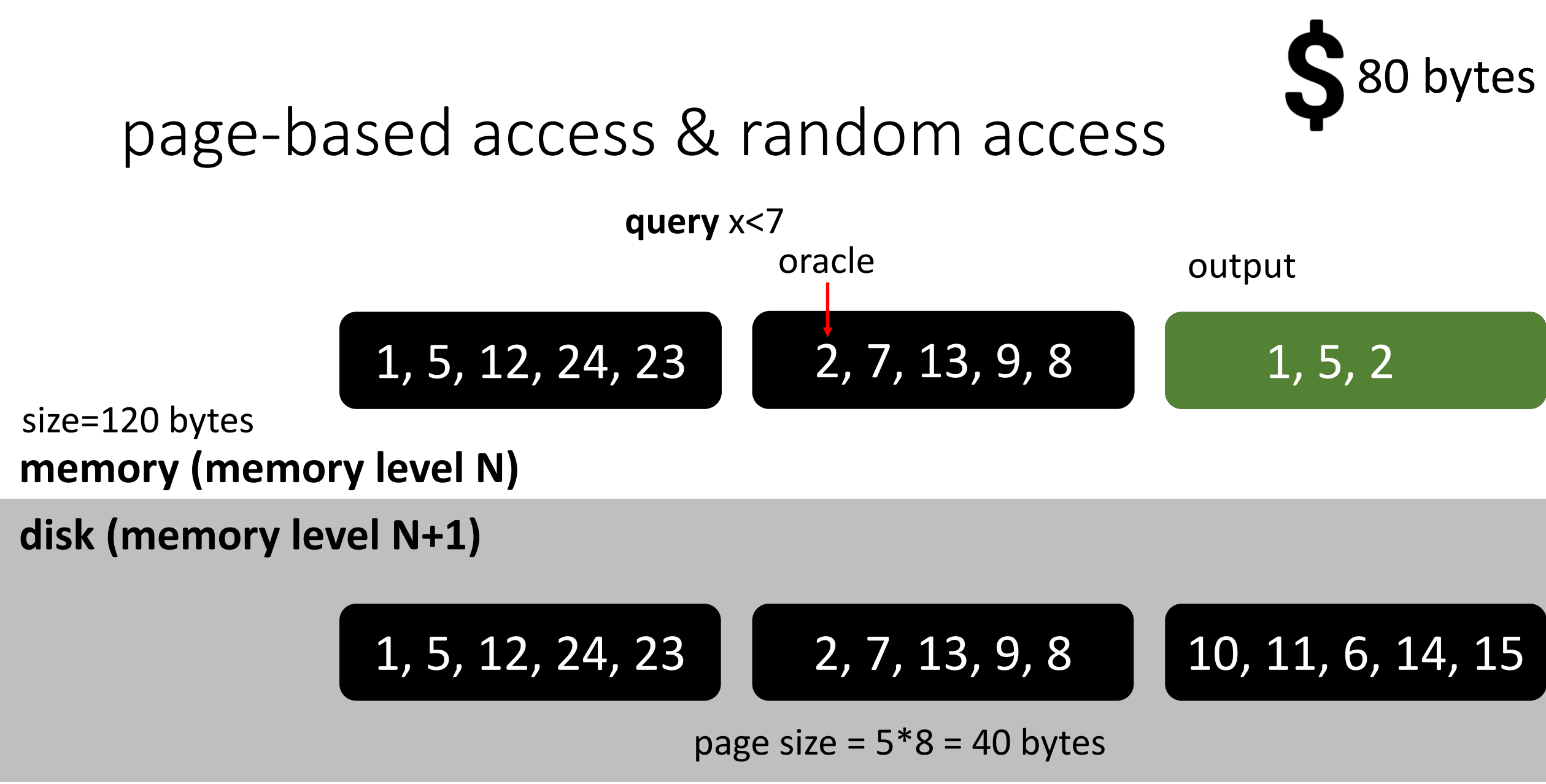
1, 5, 12, 24, 23

2, 7, 13, 9, 8

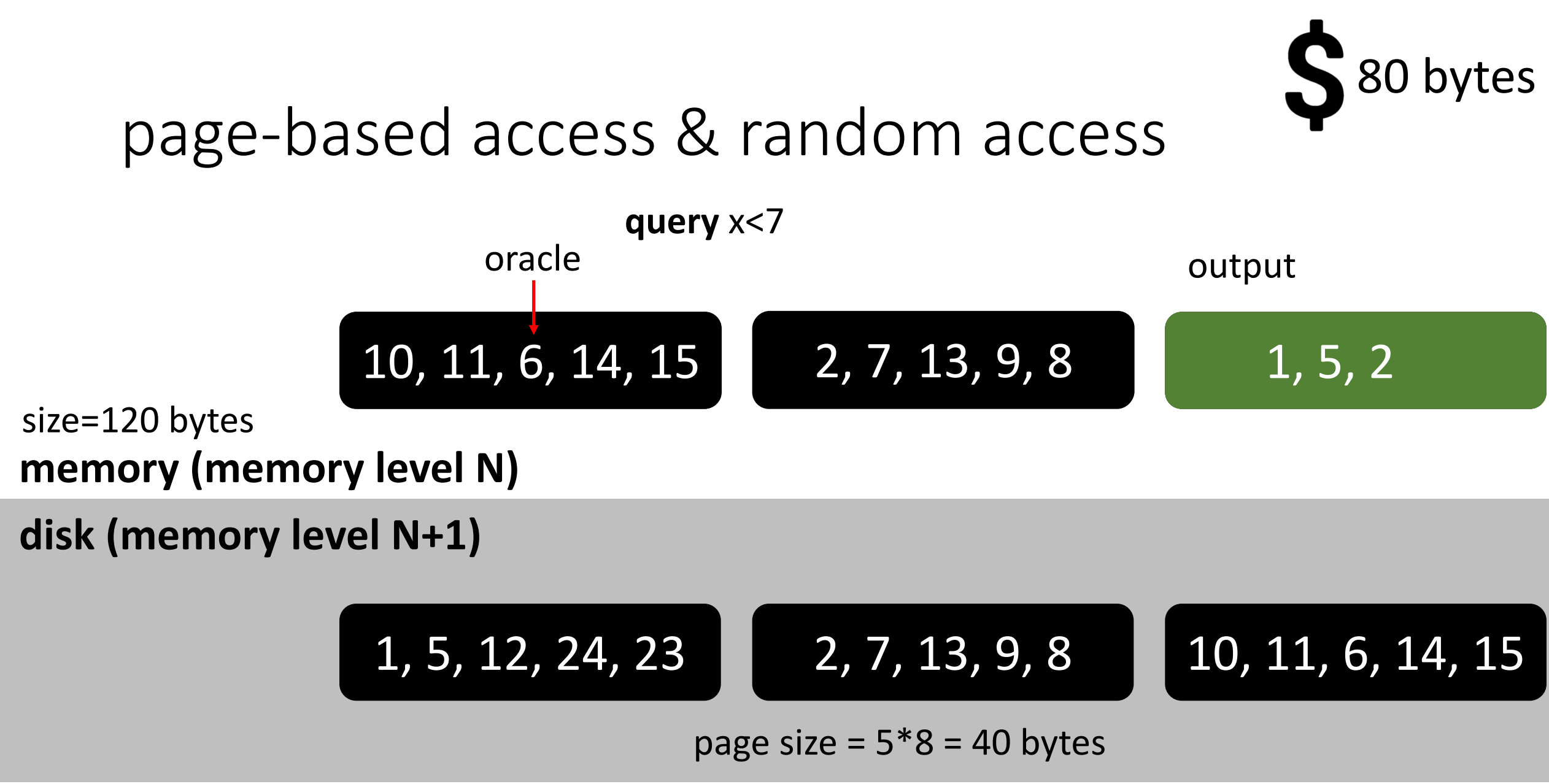
10, 11, 6, 14, 15

page size = 5*8 = 40 bytes

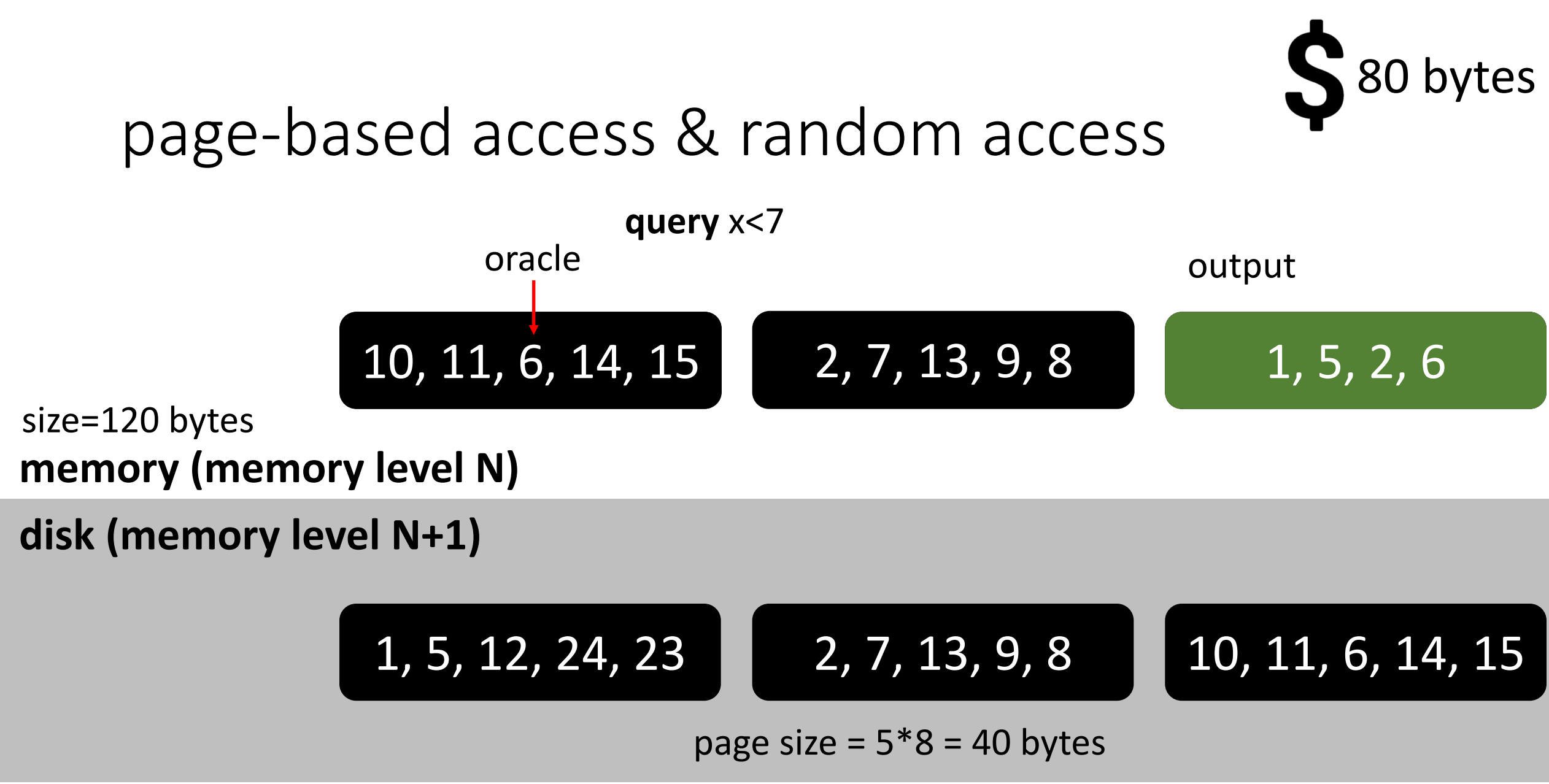
# page-based access & random access



# page-based access & random access



# page-based access & random access



# page-based access & random access

$ 120 bytes

*was the oracle helpful?*

**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

# 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

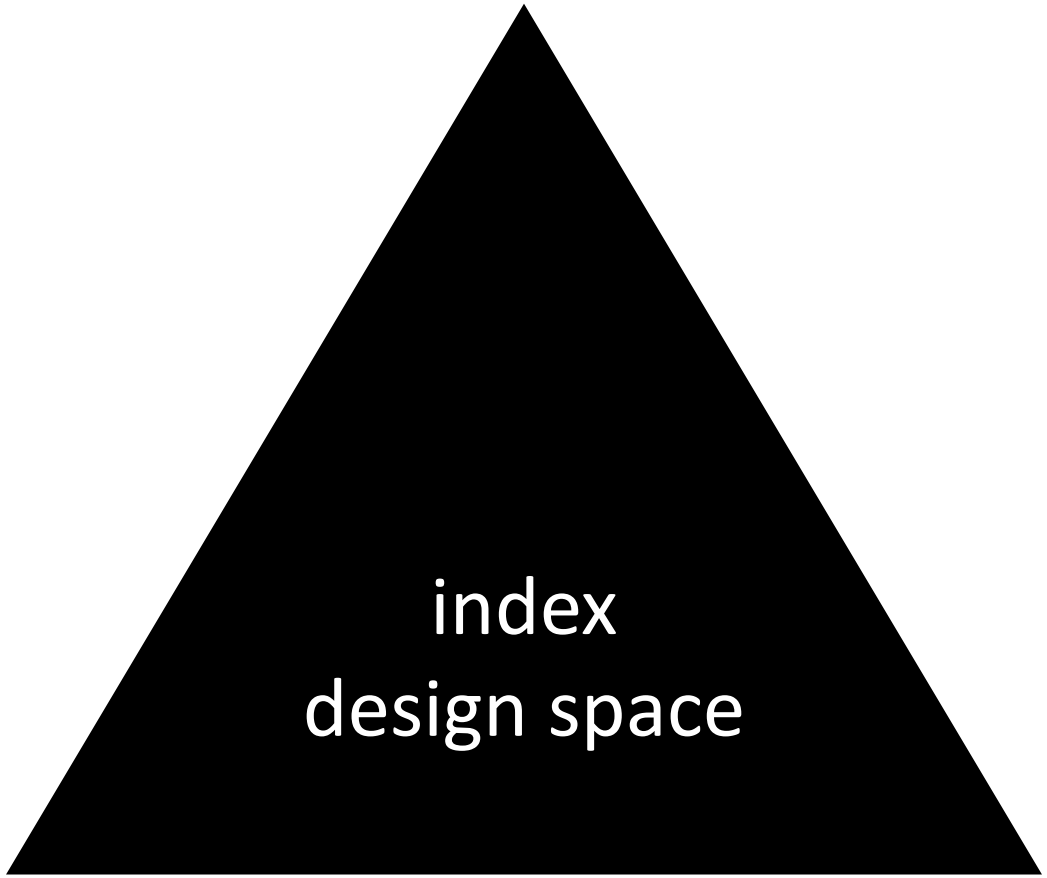
layouts, indexes

every **byte** counts

overheads and tradeoffs

know the **query**

access path selection



# 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)***

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;
```

qualifying positions

query: value<x
over an array of N slots

data

what if only 0.1% qualifies?

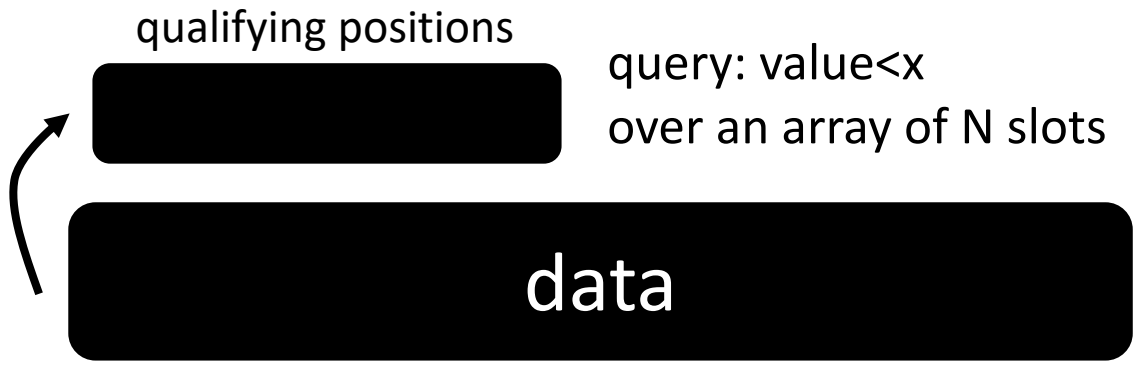
**memory**

data

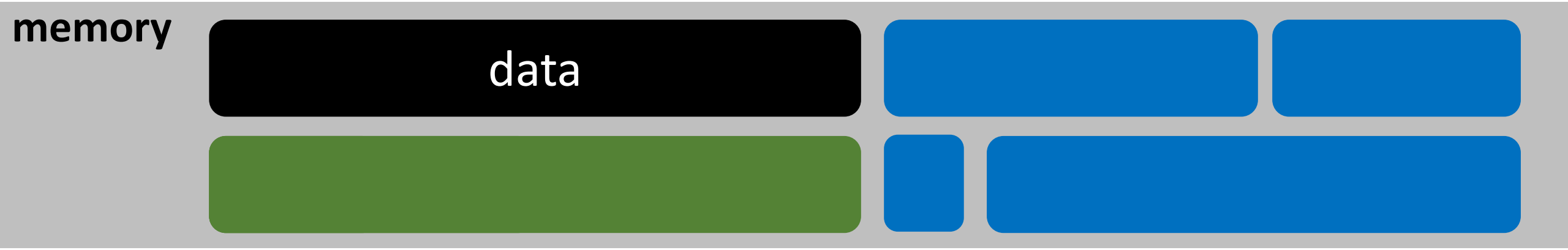
result

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;
```



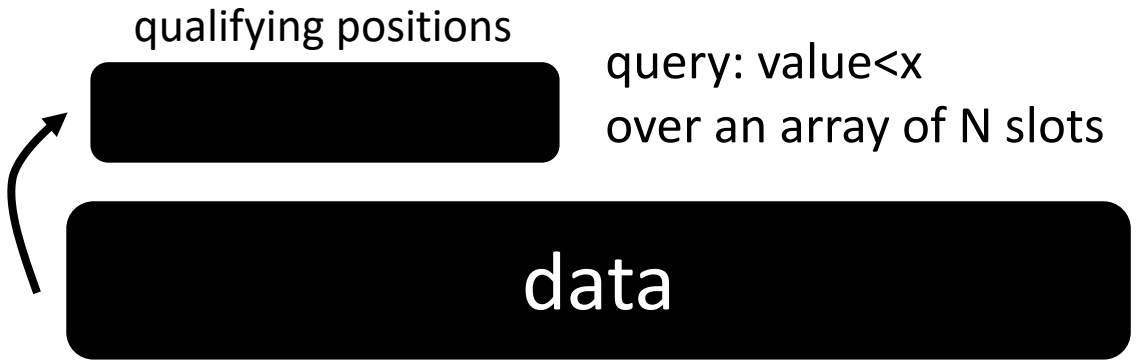

what if only 0.1% qualifies?



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?

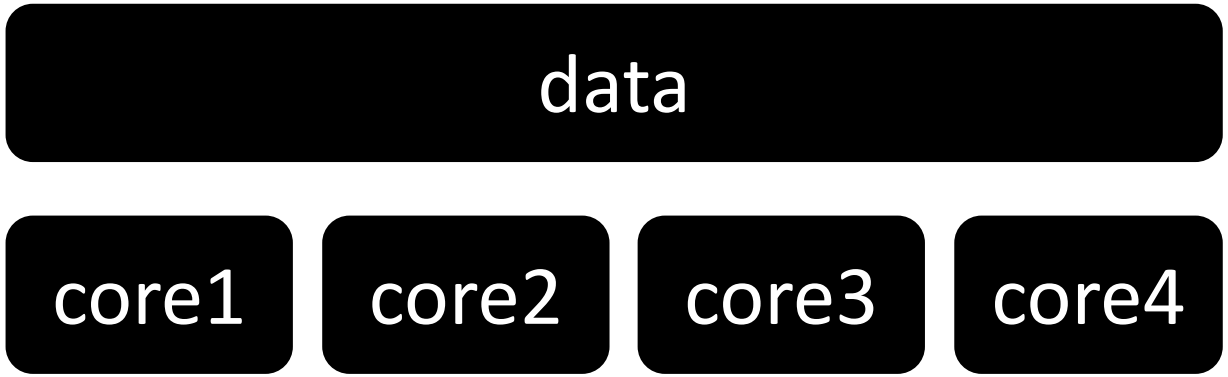
how to bring the values?
(remember we have the positions)

qualifying 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?

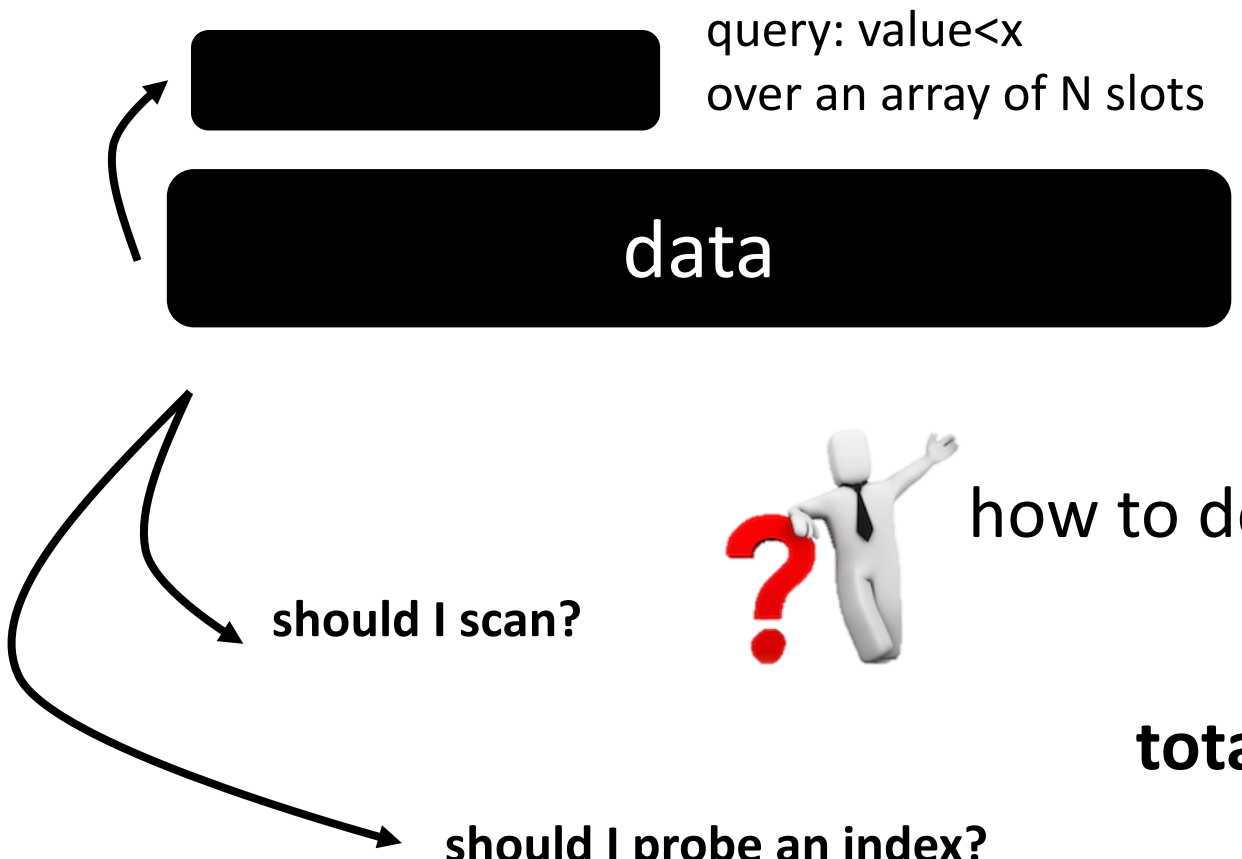



needs coordination!
what about result writing?

# 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;
```






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://midas.bu.edu/classes/CS591A1/

**piazza website:** http://piazza.com/bu/spring2020/cs591a1/

**presentation registration:** https://tinyurl.com/S2020-CS591-presentations

**gradescope entry-code:** 9568G3

**office hours:** Manos (Tu/Th, before class)
Andy (M/W 3-4pm), Ju Hyoung (M 11am-noon / F 3-4pm)

**material:** papers available from BU network

BOSTON UNIVERSITY

# CS 591: Data Systems Architectures

class 2

## Data Systems 101

**next week:** modern main-memory data systems & semester project