

# CS 561: Data Systems Architectures

class 2

## Data Systems 101

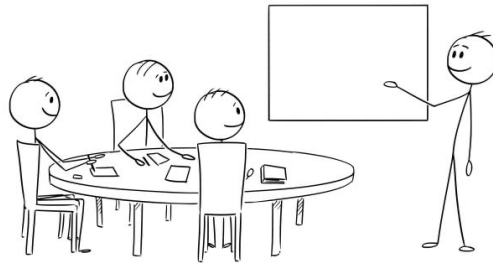
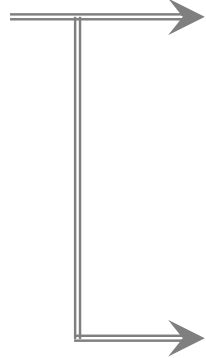
Tarikul Islam Papon

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

# What do we do in this class?



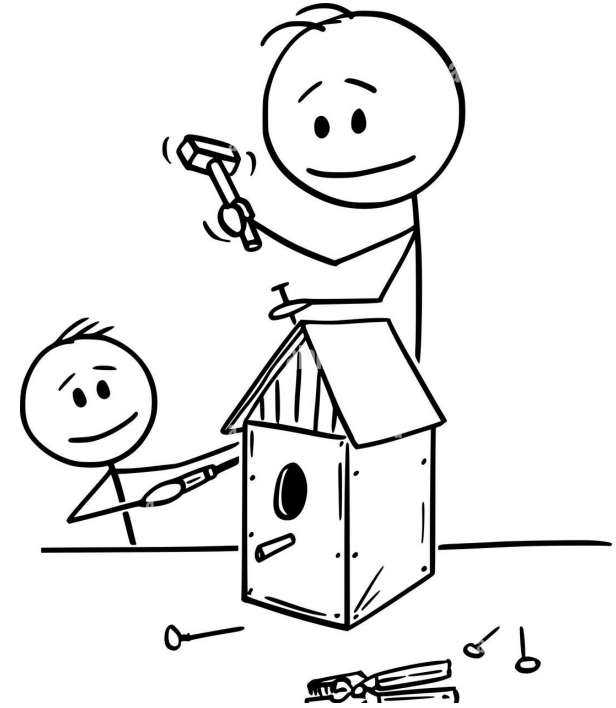
reading papers



presentations



reviews



projects

# Projects

## project 0

A small implementation project  
to sharpen dev skills

independent project



Due on Feb 2, 2024

**AND**

## project 1

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

groups of 3



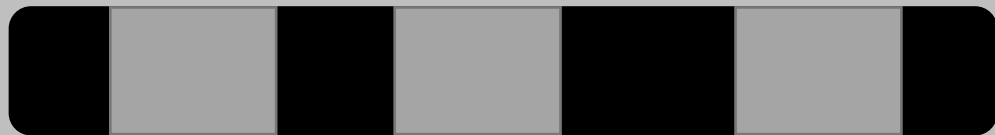
# Projects

**AND**

## project 0

A small implementation project  
to sharpen dev skills

independent project



Due on Feb 2, 2024

## project 1

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

groups of 3



Start forming groups  
Due on Feb 16, 2024



# Projects

# OR

**systems project**

**research project**

groups of 3

groups of 3

implementation-heavy C/C++ project

pick a subject (**list available in website**)

design & analysis

experimentation

```
01:0 cout<<endl<<endl<<"Iterations #<<"<<" X(1)<<"X(2)<<" X(3); //00011:00010111010:111001001
02:0 cout<<endl<<" 0<<setw(7)<<j<<setw(15)<<j2<<setw(14)<<j3; float temp[3]; //1001001:10100010:0
03:1 cout<<endl; //10010:10:01001:100111:0:0011011:110111001:100110100
04:1 //1001:10101:0101:000:101:100:0001:0:0001:1:10011000101:10011011
05:0000for(int s=1;s<=20;s++) //01:01:01100111:1100011000:01110100011010
06:00110:1:1:0:011:101:0010:111010:01:000:0:0110111001:001:001
07:1-0.0:1 temp[0]=1;temp[1]=2;temp[2]=3; //1:0:000:0:1:000001:01:01011
08:1101:1100 j1=a[3]-a[1]*temp[1]-a[2]*temp[2];a[0]; //11011100110:11100:101:01
09:110:0:0 j2=b[3]-b[0]*temp[0]-b[2]*temp[2];b[1]; //01:01101101011:000:011
10:1110:110 j3=(c[3]-c[0]*temp[0]-c[1]*temp[1])/c[2]; //1011:1011:0:011:0:1:0:000:1:0
11:000110100 cout<<" "<<s<<setw(17)<<j1<<setw(15)<<j2<<setw(14)<<j3<<endl; //01:001:01:001010:01:1:00
12:1 //10:0:001 //j1=temp[0]&&j2==temp[1]&&j3==temp[2]; //1:0:00:000:0:0:1:000:1:000:1000:01:01000:00:01:000:0
13:11010:110 break; //1:0:001:01:000:000:1:1000:0:1:1:000:10:11:011:01:01:00:1:1:1001:0100:1:1:001011011001:10011
14:10010101; //0:001111:0100:10111:0:011:011:1010:0:1:0001:101:00:1:000:1:000:101:000:1:000:101:101:011
15:0 } //0:00101:1100:101:101:0:1:0:100:110:000000:0:0000:100
16:0 //////////////////////////////////////////////////////////////////Function Of //000:111010001:0100101:10:1:000101:011:0000
17:1 Transcoding////////////////////////////////////////////////////////////////// //101:011100:001:100:1101100:10001001:000:001
18:1 void swap(float a[],float b[]) //function definition //1:011100 cout<<endl; //0:0:101:01011:0:0:101:01:10
19:1 float temp[4]; //000:1011:0010010:0000:1:01100:10:0:10111 //101:01000:1001000:1100:1001:1110111
20:00 //100111010:01:01100:11011000:0:000 cout<<"b[1]"=; //0:101101011001:100:1:000:0111100
21:0 //1101011:000:0:1:0111:10111:0100111:1000:1000101010 cin>>[3]; //01:01111011001:01101100:1001:100
22:0 } //000100100000:101:0:100 cout<<endl; //01:100100:10:0:101:101:10:110
23:01 cout<<"--PROCESSING--"<<endl; //000100100000:101:0:100 cout<<endl; //01:100100:10:0:101:101:10:110
24:11 cout<<"Preparing Encode X-Y con."<<endl; //0001:101:011:1011:1001:110100 cin>>[3]; //1:0:101:011101:110000:1000:001:100
25:1 //01:10:11:01:00:000:00111:10001:01:00:1110:101:01:00:11:00:11:01:101:01:01:01
26:1 //Procedure starting <<endl; //10:0:0:10:000:011:1:000:1:00:1:1:000:1:000:001:101101
27:1 for(int i=0;i<4;i++) //000101:010:1:100:000; //cin>>b[i]; //000:001:1:000:010010101:1011:01
28:1:0:01 //101001:1101110:1100111:0100:0001:101010:011 cout<<endl; //1:000:1001:000000101:000:0010
29:1 //10:10 temp[i]=a[i]; //001:10:0010:101:100:110:1:1111:1000:011011 //0011101:001:101001:1101:1001:10011
30:1 //0010 a[i]=b[i]; //11:00100:101:110:1011:00111:100:01 cout<<"(2)"; //1:0000:0110:101101:1100:1010:1:10101
31:1 //0000:1001:101:101:101:101:1001:11010100 cin>>[3]; //1:0:101:011101:1100000:1000:001:100
32:1 //0:10:10:11:01:10:100:000:00111:10001:01:00:1110:1:01:00:0:011:100:1:1:000:0:10000:1000:001:10000
33:1 //1:101110010:1101:00:100:0001111010:01:1:101 //for(int k=k;k<=n;k++) //000:0:11:00:0:1010:1:0111000:
34:000 cout<<"--Parsing--"<<endl; //1:10:0:0:1:0:10:1:100 //0001100:0001:100:010111:1:00011000:1:001
35:000 cout<<"X-Y transcode"<<[0]<<"X(1)" + "<<[1]<<"X(2)" + [110 cout<<"(3)*k+1<<"?"; //0:0011101001010:10
36:010<<[2]<<"X(3)" = "<<[3]<<endl; //1100:1101000:101:1:10101:100 cin>>[k]; //0:000101:101010:0:010101010
37:1 <<[0]V transcode<<[0]<<"X(1)" + "<<[1]<<"X(2)" //001101001011: cout<<endl; //1:001000111:001001:1011101:11
38:1 //<<[3]<<endl; //10:1101:011:11:10:0110000:00:011110 //1:100:11:01101:0101:10:100:000:00:11100
39:1 <<[4]Summary<<[0]<<"X(1)" + "<<[1]<<"X(2)" //0:0:0:1:0:10:000:1:0:000:1:0:10:11:01:1:01:101:011:001:00:1100
40:1 //<<[3]; //1:110:00011:00:0:10:0001:0:100 cin>>[3]; //1:1:1:0:11:0010:11:0:00101:101:100:1110
41:1 cout<<endl<<endl; //001001:1010001:010:0:10:110:1:000:10:0:0001:1001100:100:1110:101:10:11:0
42:1 //0001000:011011:01:0:0:10:1:101:011:00:11:11 //1010:00001101:10:1000100:0001:0011001:0100:011
43:1 //0001:0001:0001:0001:0001:0001:0001:0001:0001:0001:0001:0001:0001:0001:0001:0001:0001:0001:0001:0001:0001:0001:0001:0001:0001
```

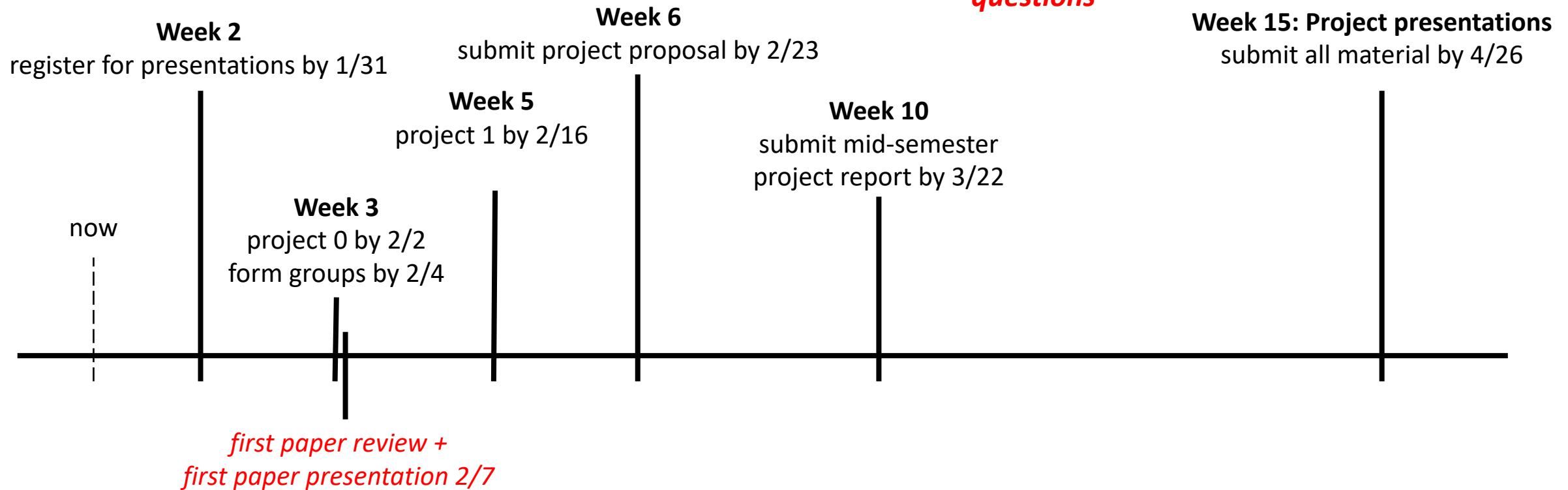
- 1. Proposal
- 2. Mid-semester report
- 3. Final report + Presentation



# class timeline



*discussions*  
*interaction in OH & Lab*  
*questions*



# Piazza



2 classes per week & OH/Labs 5 days per week

all discussions & announcements

<http://piazza.com/bu/spring2022/cs561/>

also available on class website

**We have added everyone who already registered!**

**Please double-check!**



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

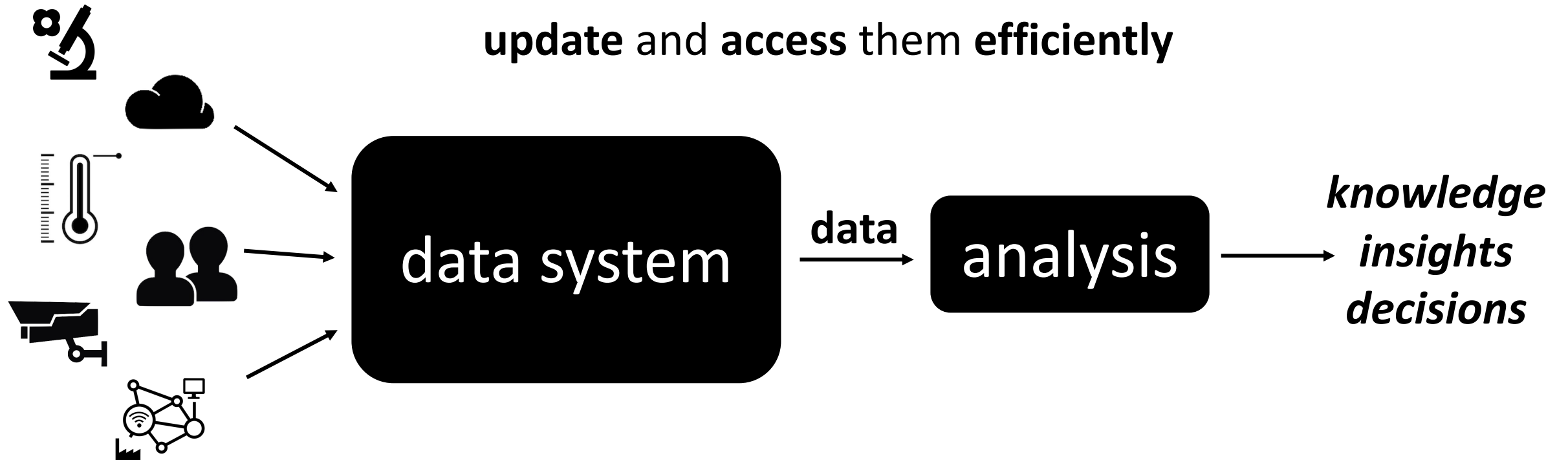
 scientific experiments

 sensors

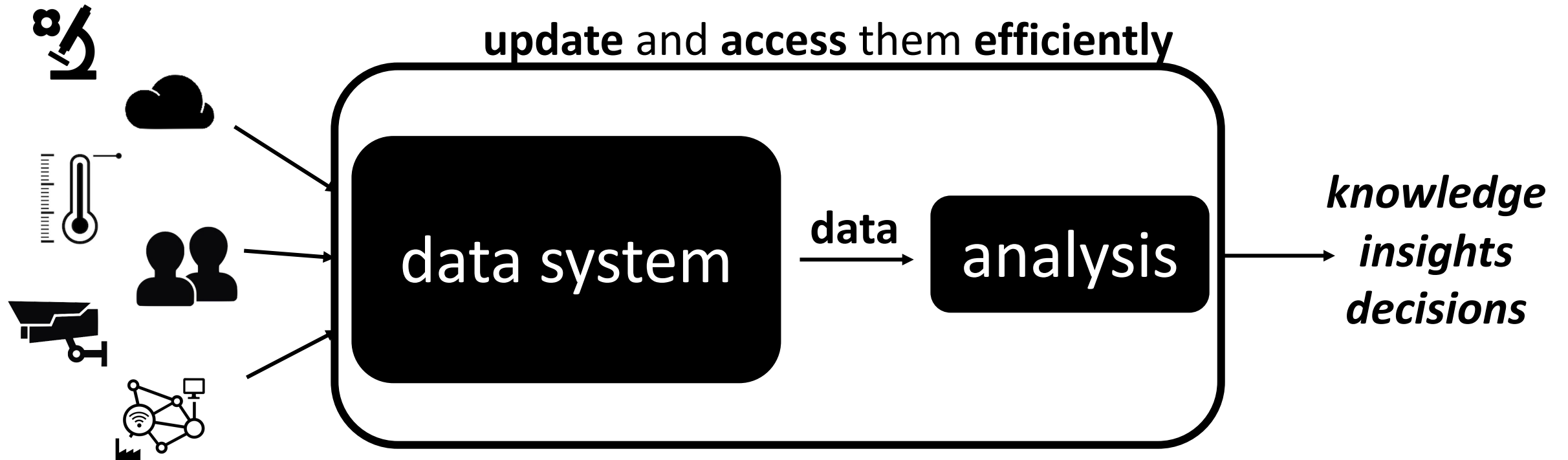
social 

Internet-of-things 

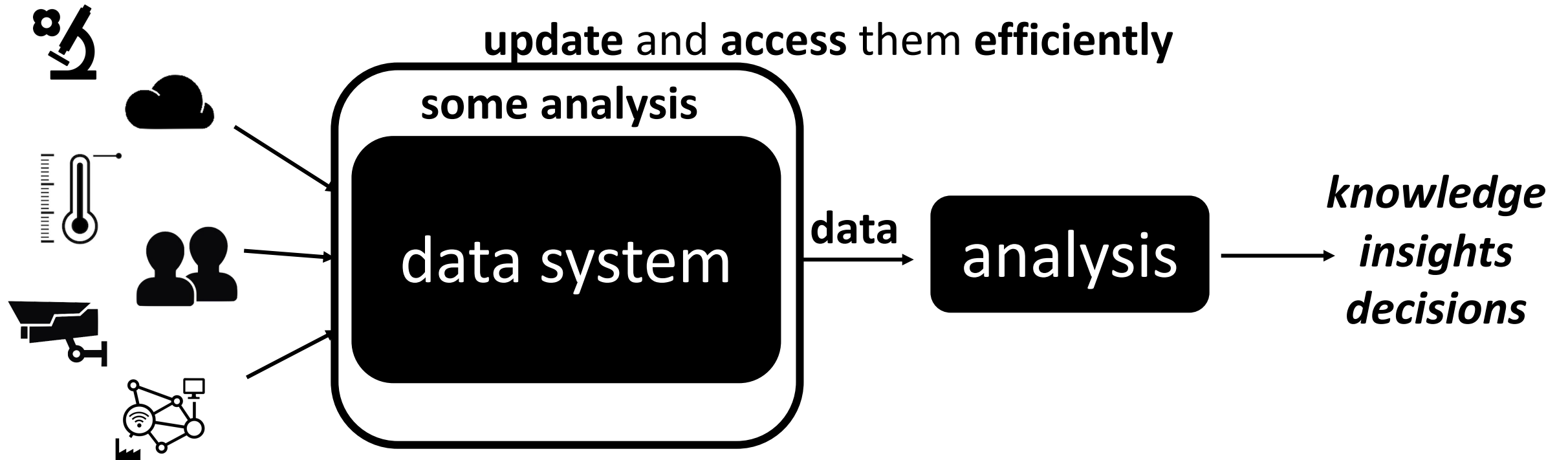
a **data system** is a large software system that **stores data**, and provides the **interface** to **update** and **access** them **efficiently**



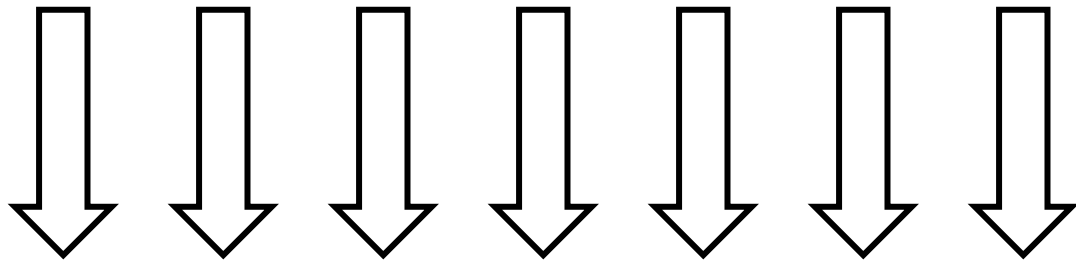
a **data system** is a large software system that **stores data**, and provides the **interface** to **update** and **access** them **efficiently**



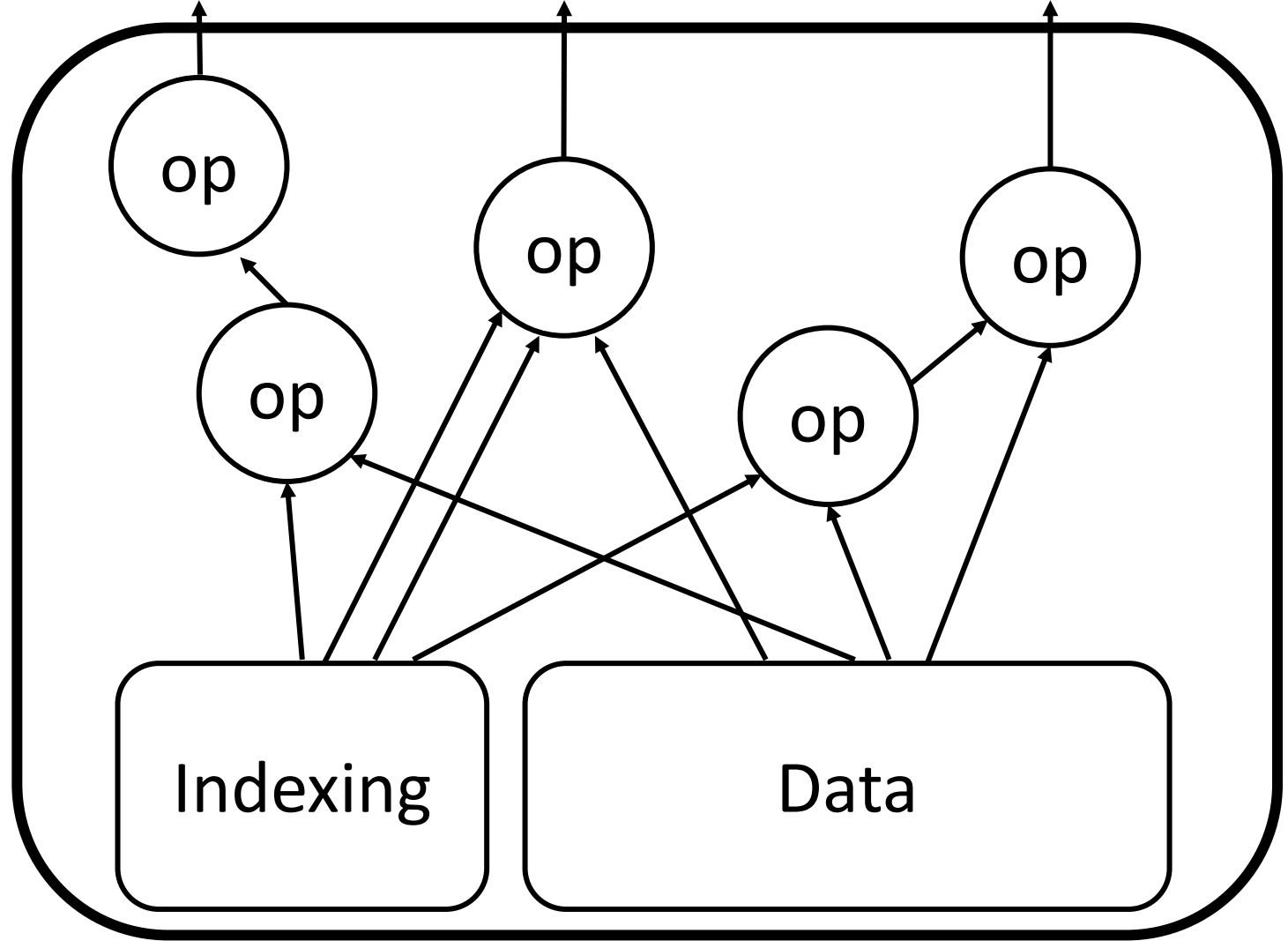
a **data system** is a large software system that **stores data**, and provides the **interface** to **update** and **access** them **efficiently**



data system: breaking the blackbox



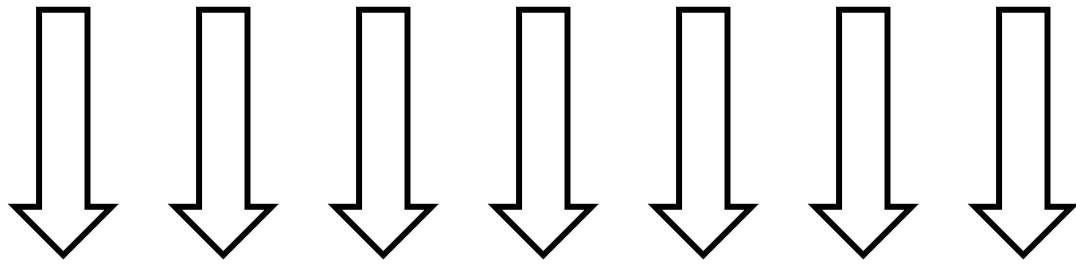
*application/SQL  
access patterns  
complex queries*



*algorithms  
&  
operators*

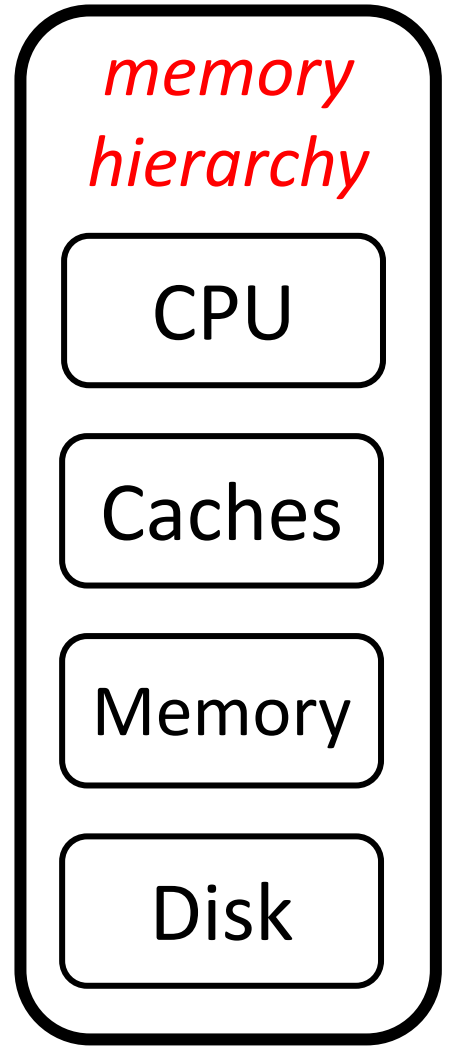
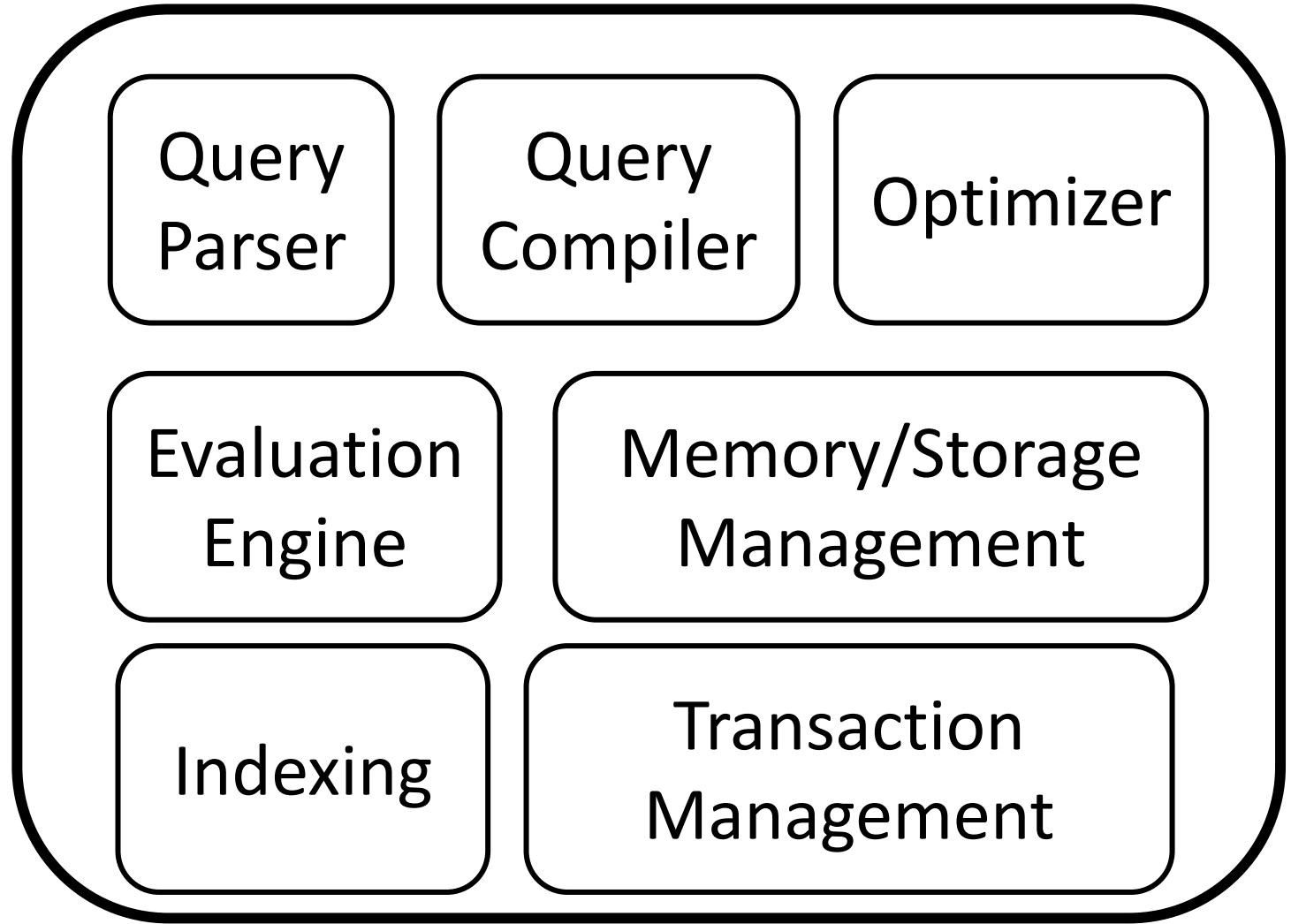
*selection  
projection  
join  
aggregate  
hashing  
sorting*

*data &  
metadata*



*application/SQL  
access patterns  
complex queries*

*modules*





growing environment

ORACLE®

facebook.

**DB**

ACID

large systems

complex

lots of tuning

**noSQL**

BASE

simple, clean

“just enough”

  
Microsoft

IBM

Google

twitter  


SAP®

>\$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.



IBM

Google



SAP

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

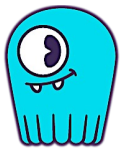
[The Forbes, 2016]



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**

ORACLE®

facebook

Microsoft

IBM

Google

twitter

SAP

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

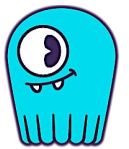
[The Forbes, 2016]



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]



**APACHE**



Cockroach Labs



mongoDB®



Couchbase

SCYLLA

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

[Forrester, 2016]

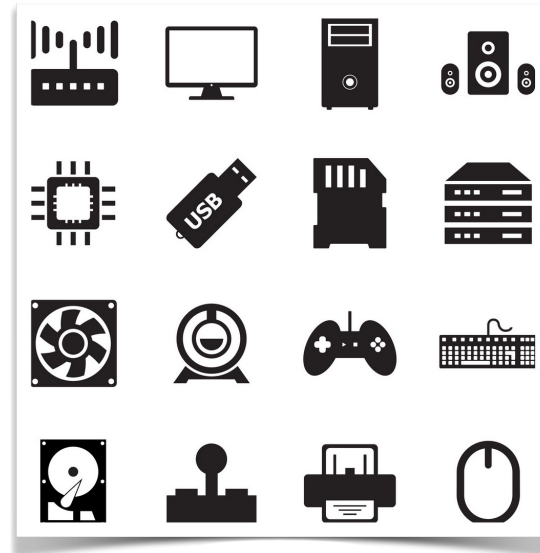
# *growing need for tailored systems*



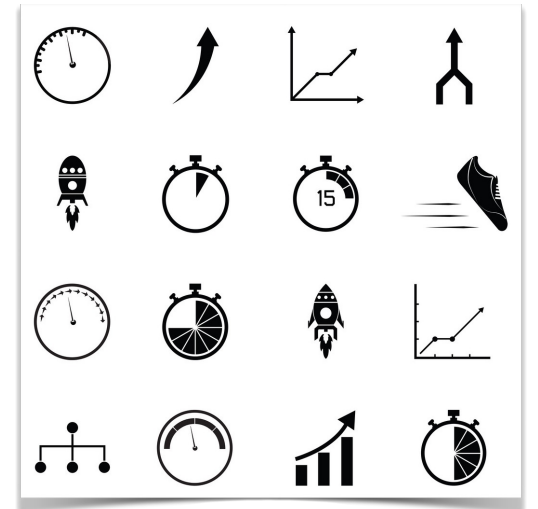
more data



new hardware



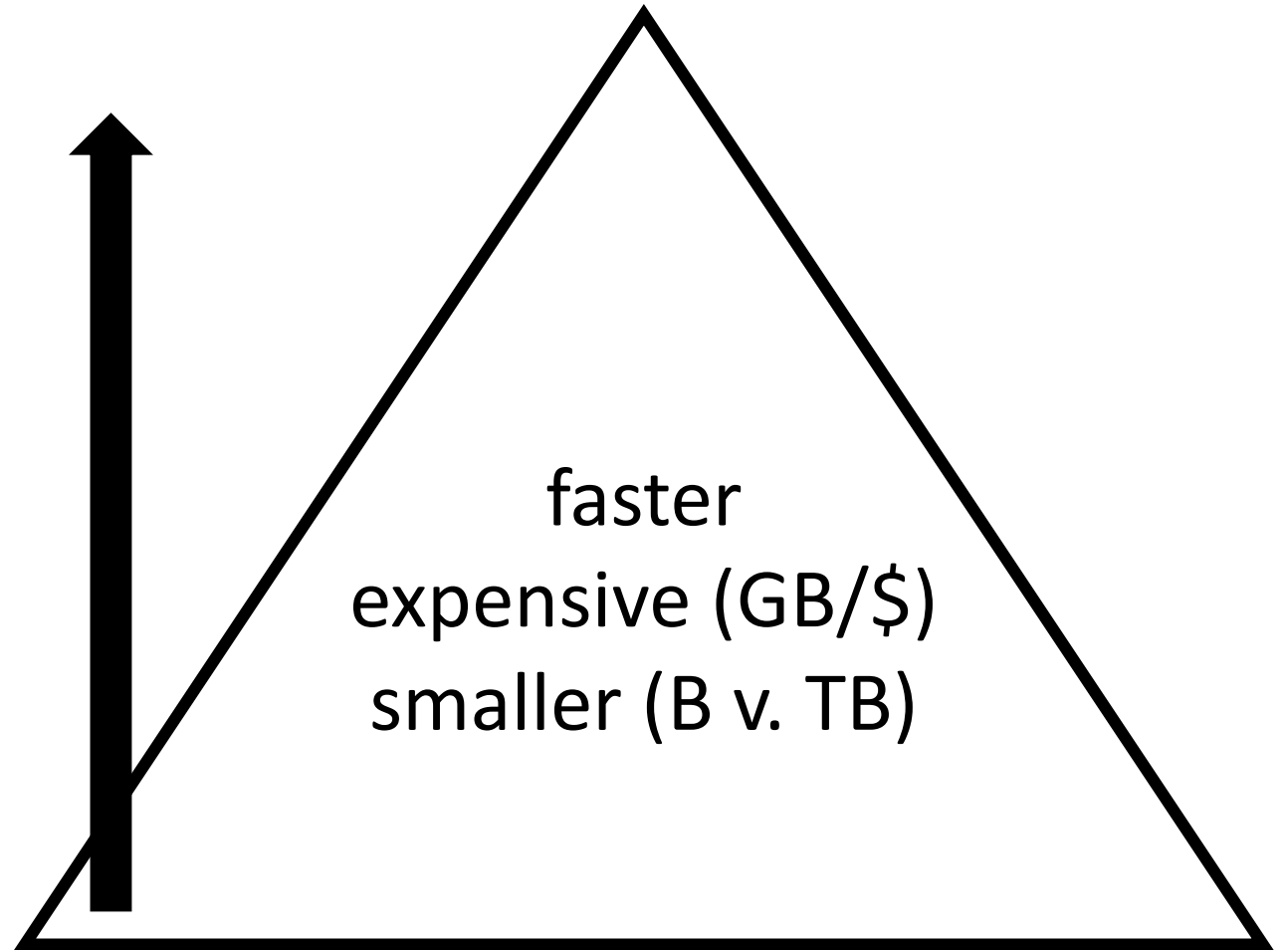
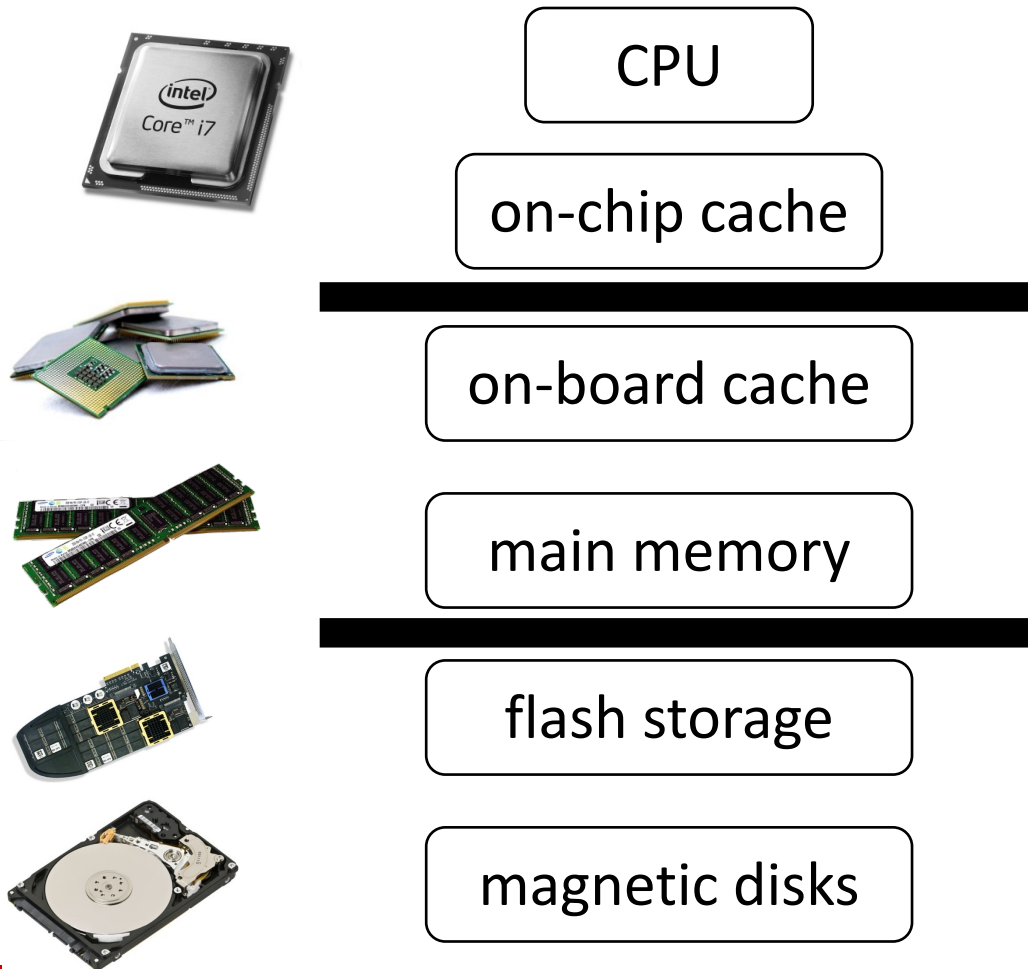
new applications



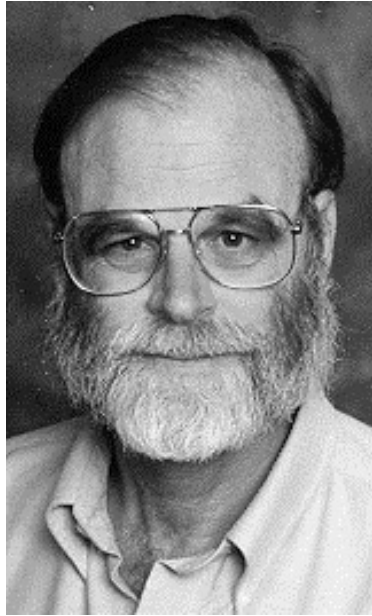
new performance goals

data systems & the hardware

# memory hierarchy



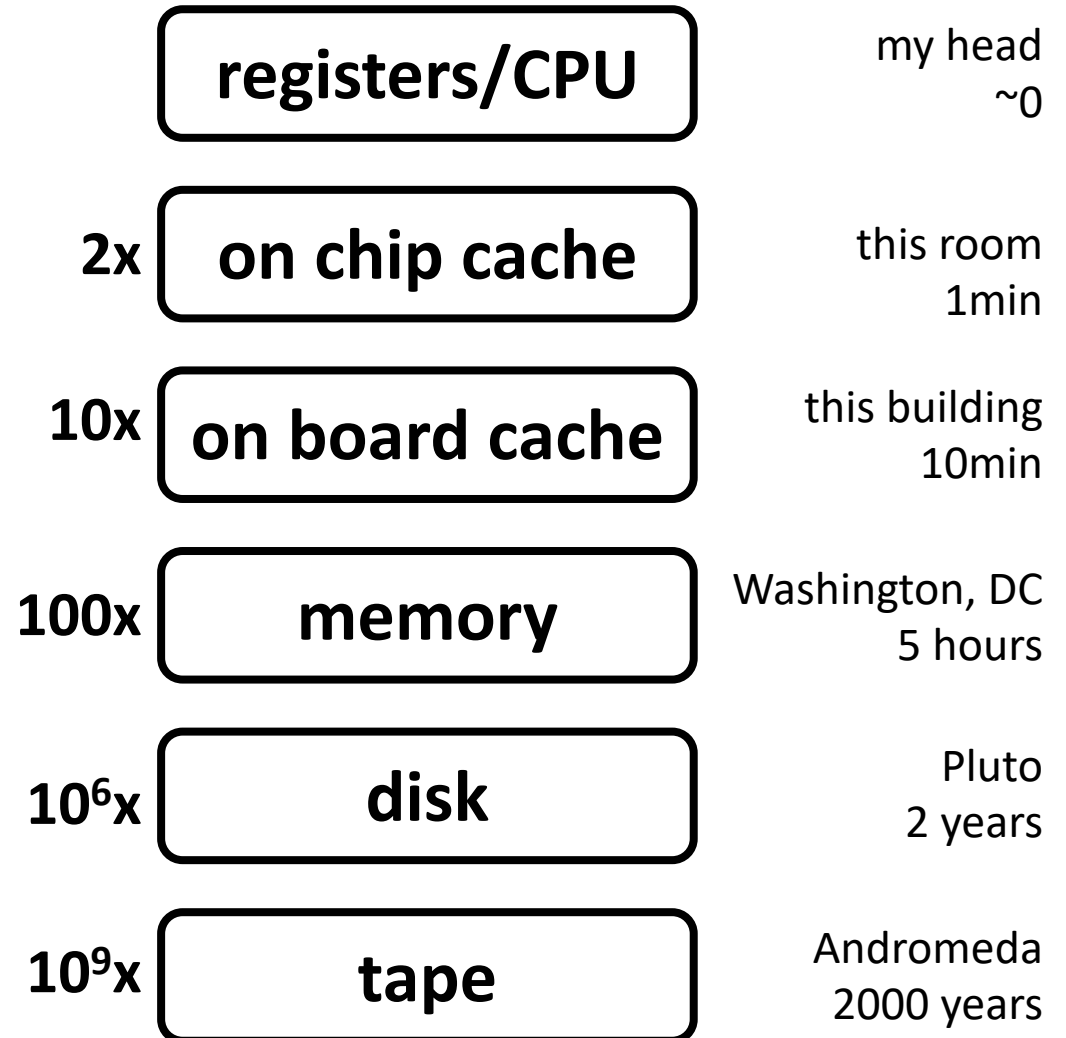
# memory hierarchy (by Jim Gray)



Jim Gray, IBM, Tandem, Microsoft, DEC

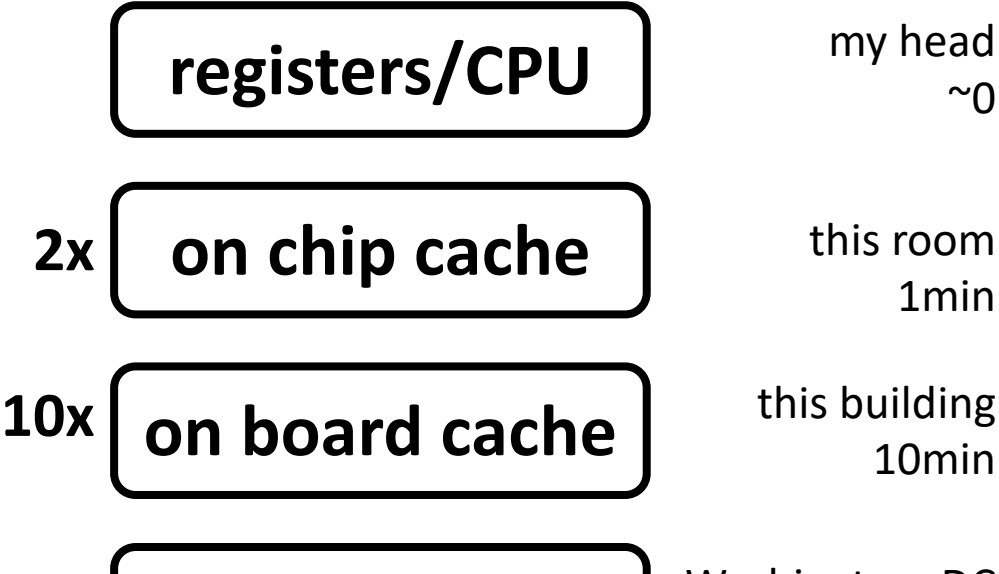
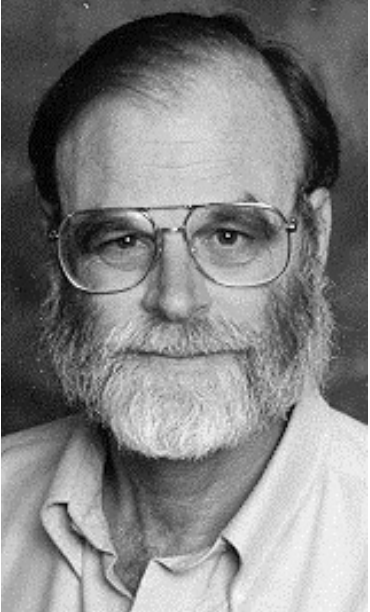
**ACM Turing Award 1998**

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





# memory hierarchy (by Jim Gray)

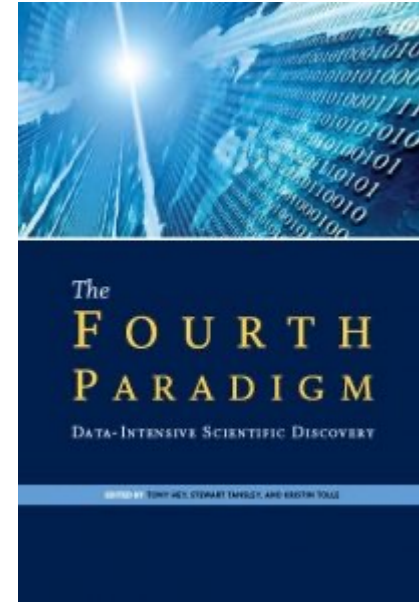
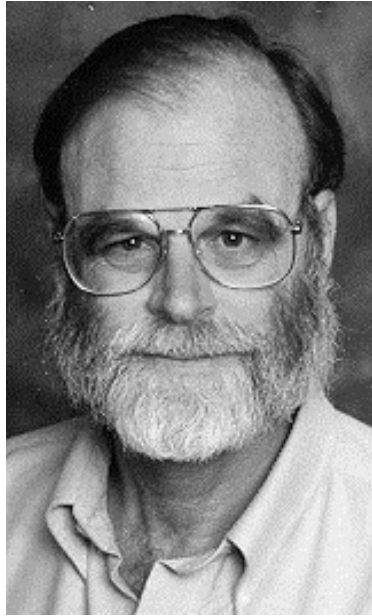


45TB @ \$150

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



# Jim Gray (a great scientist and engineer)



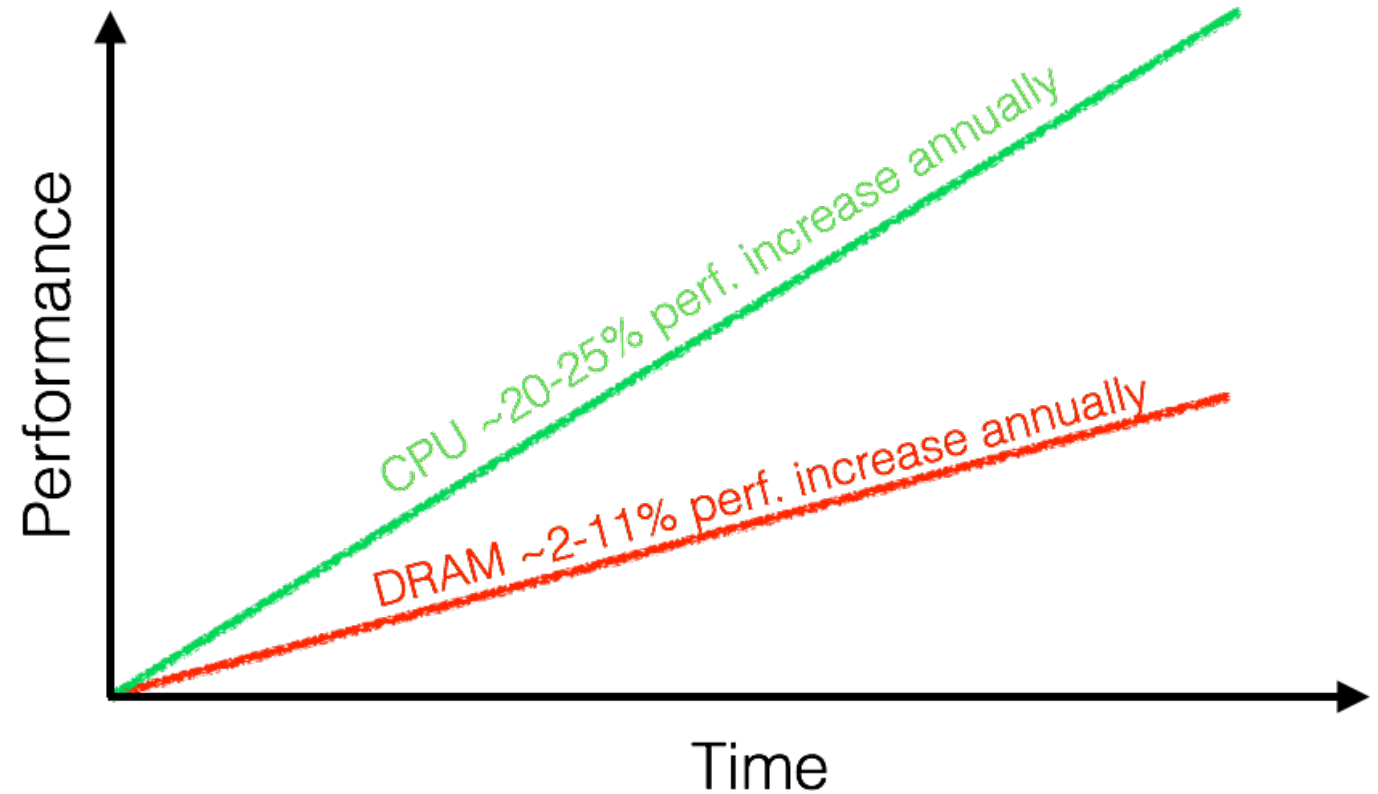
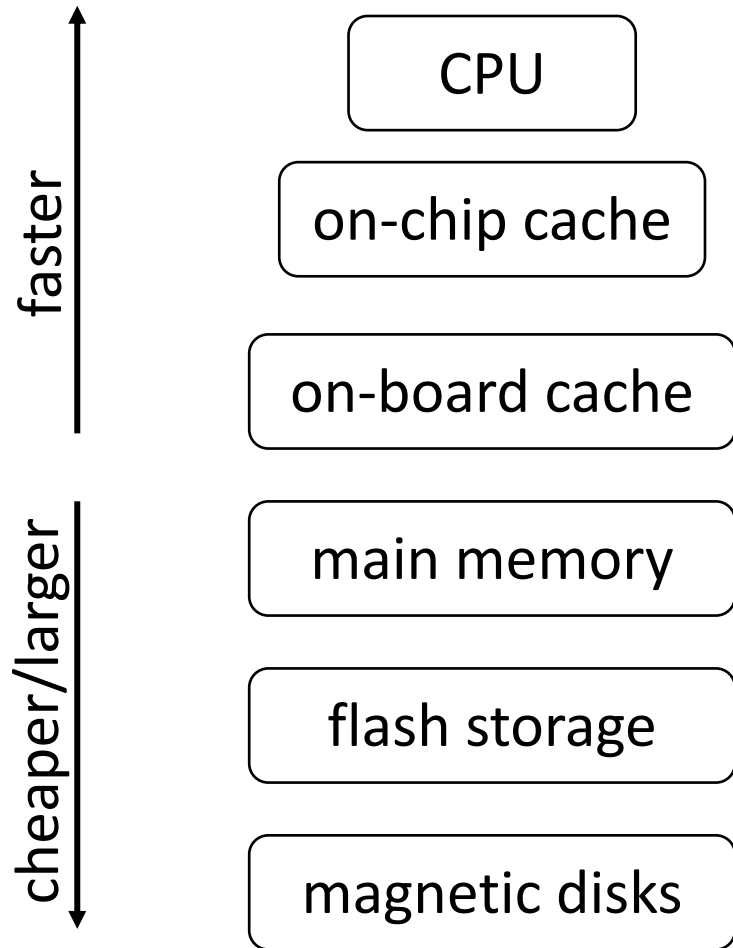
*the first collection of  
technical visionary research on  
a data-intensive scientific discovery*

Jim Gray, IBM, Tandem, Microsoft, DEC

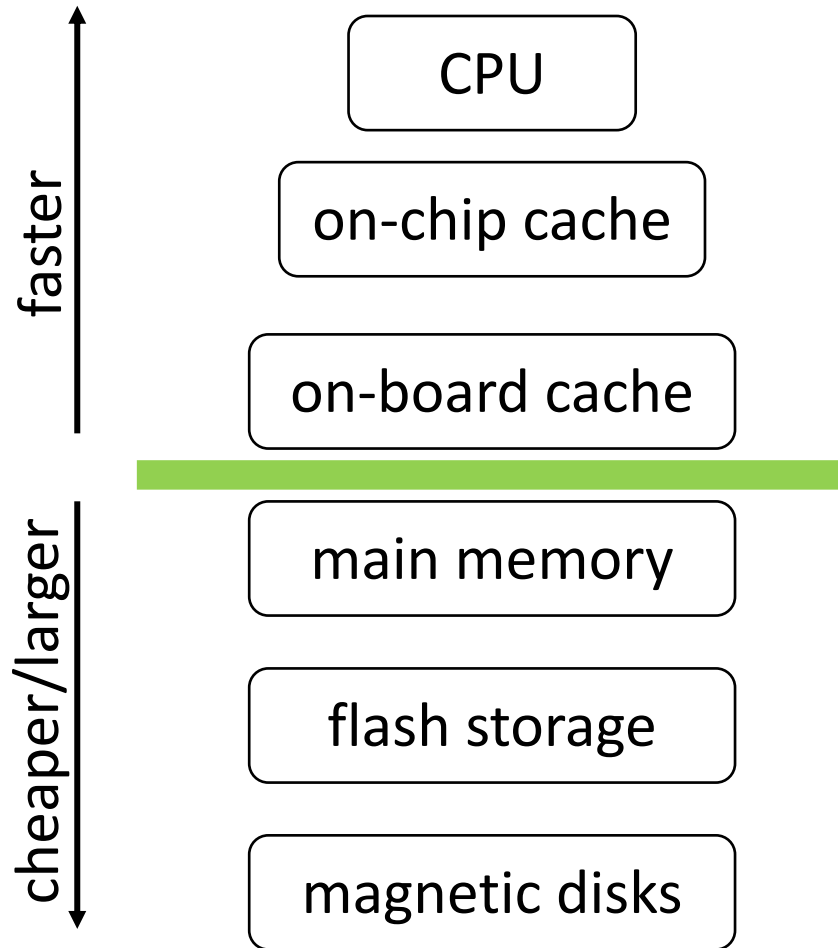
**ACM Turing Award 1998**

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

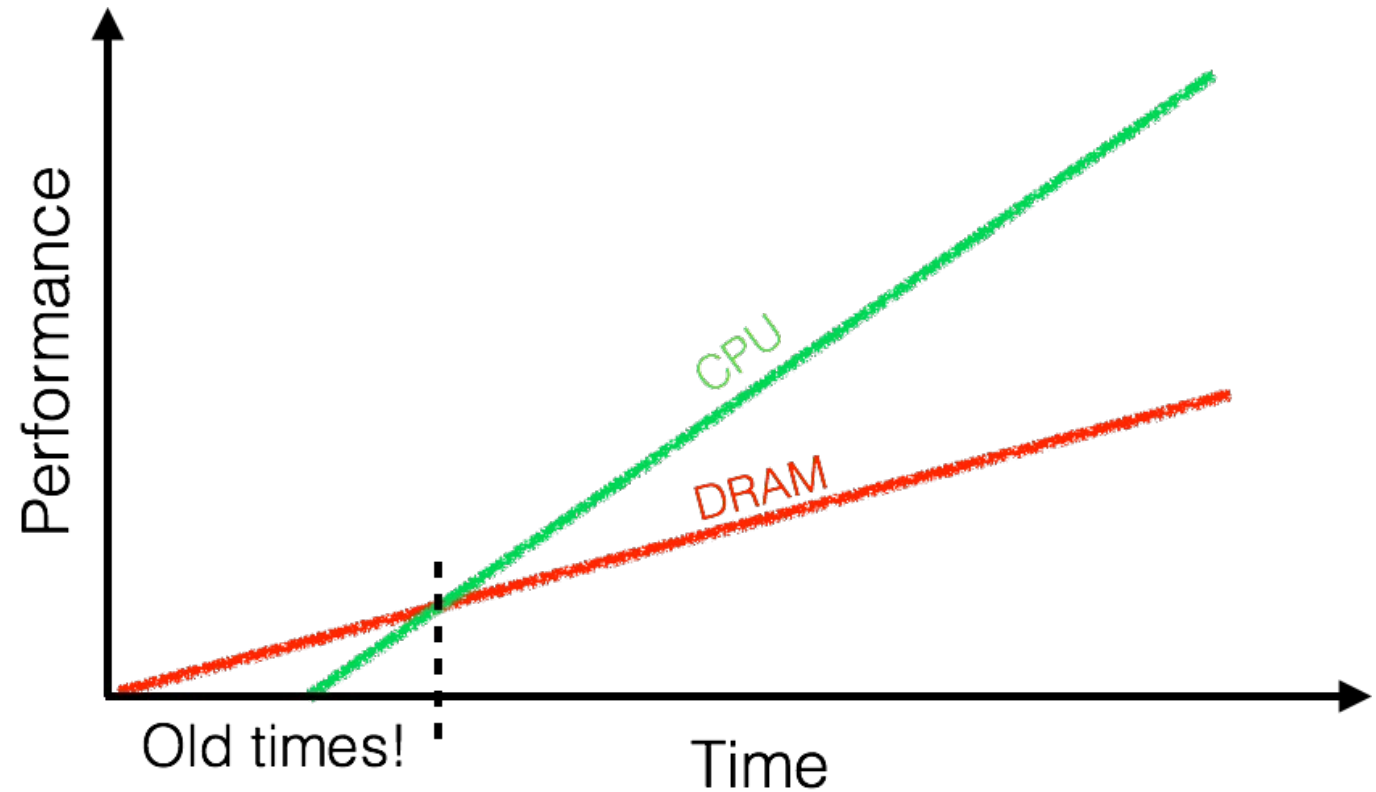
# memory wall



# memory wall

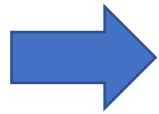


*be careful when you go below the green line*



# 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?**



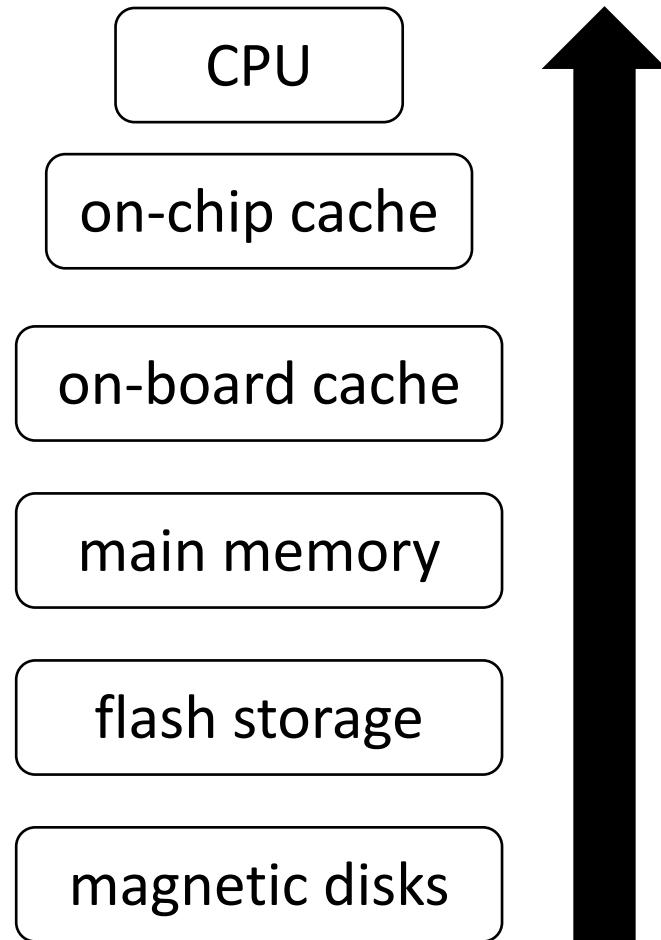
**memory miss:** looking for something that is not in memory

**what happens if I miss again?**



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

# data movement



data goes through  
all necessary levels

also read  
*unnecessary* data

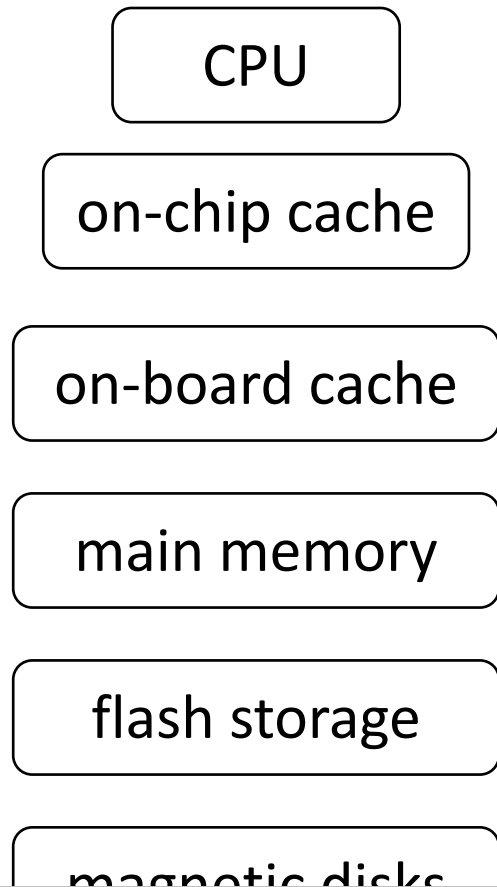


Photo by Gary Dineen/NBAE via Getty Images

need to read only X  
read the whole page



# data movement



data goes through  
all necessary levels

also read  
*unnecessary* data



remember!



Photo by Gary Dineen/NBAE via Getty Images

need to read only X  
read the whole page



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

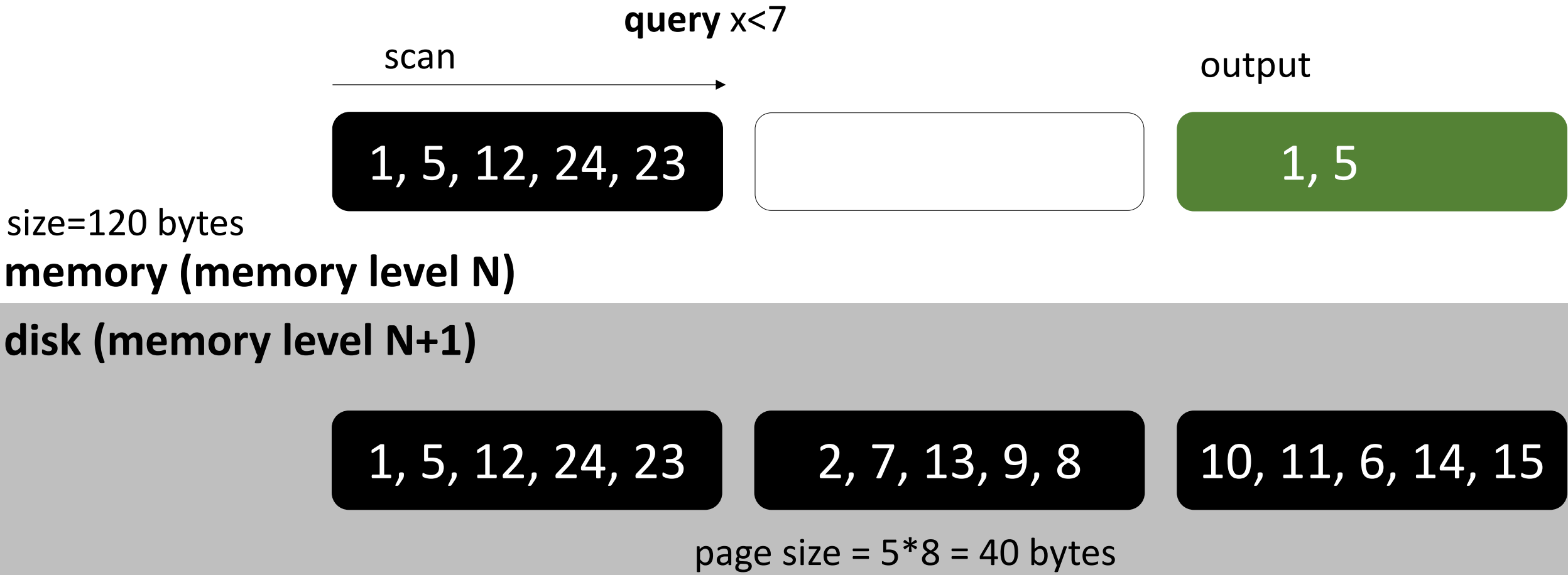
10, 11, 6, 14, 15

page size =  $5 * 8 = 40$  bytes



\$ 40 bytes

# page-based access & random access



\$ 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

\$ 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

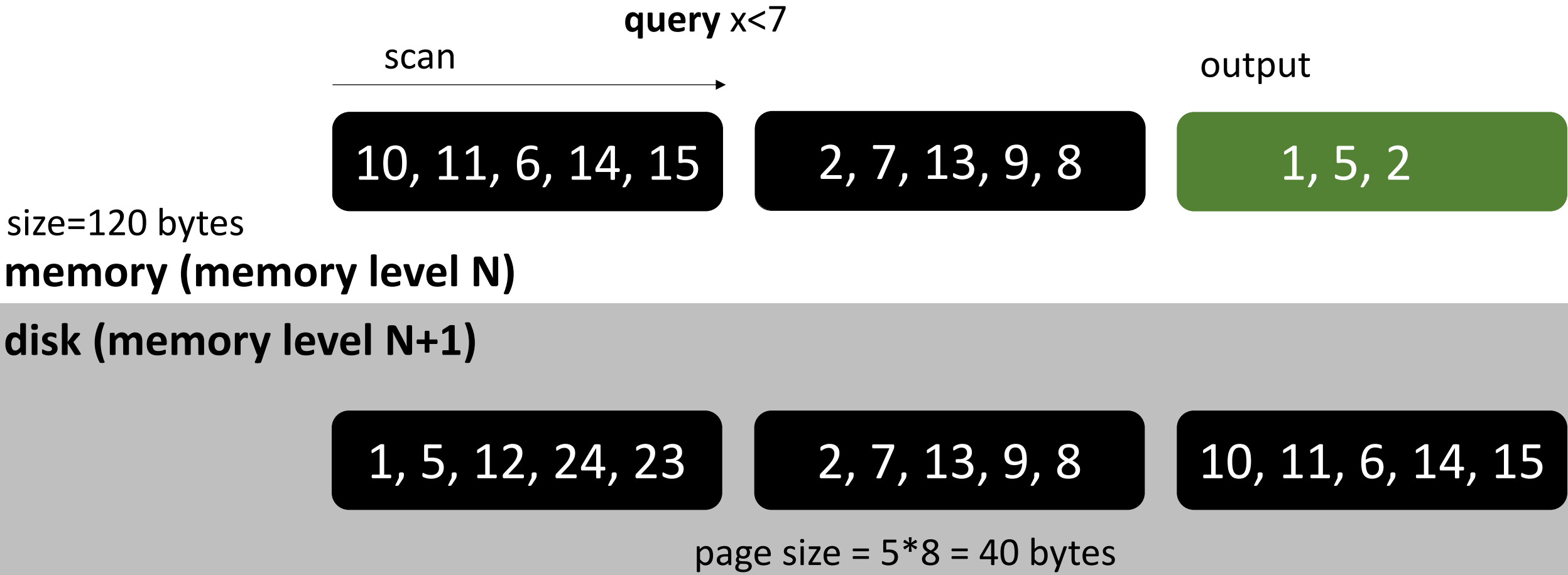
2, 7, 13, 9, 8

10, 11, 6, 14, 15

page size =  $5 * 8 = 40$  bytes

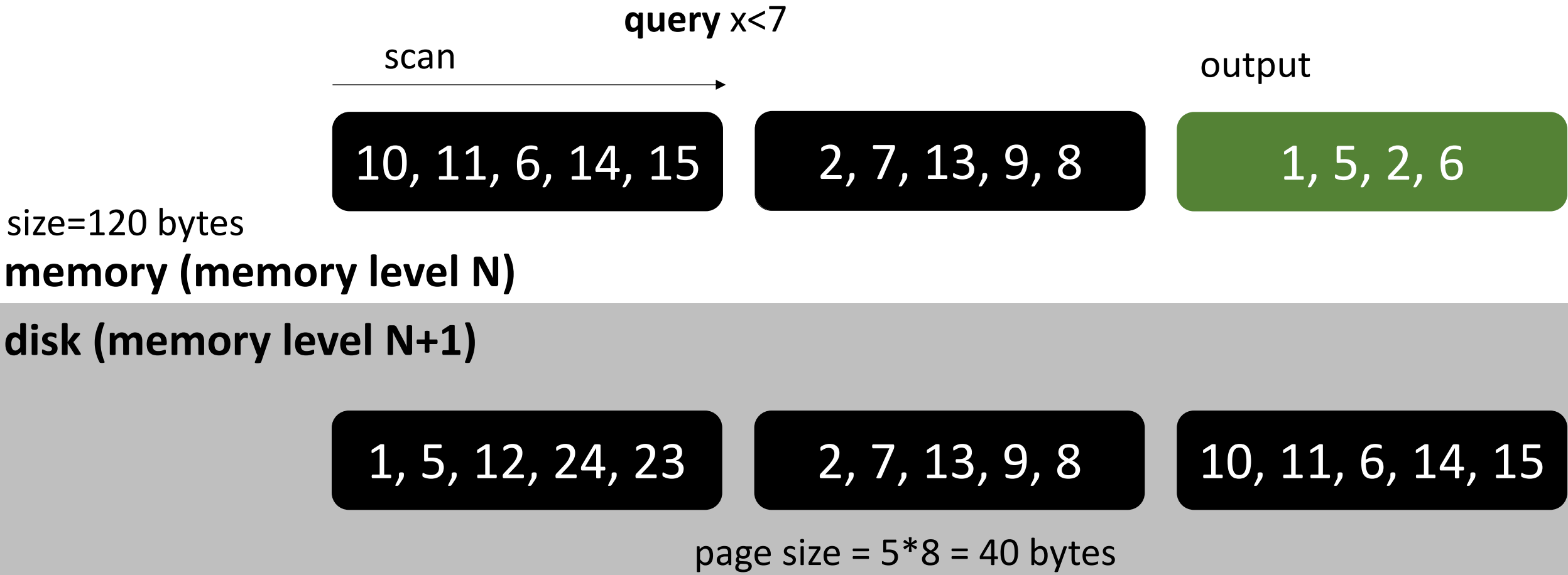
\$ 80 bytes

# page-based access & random access



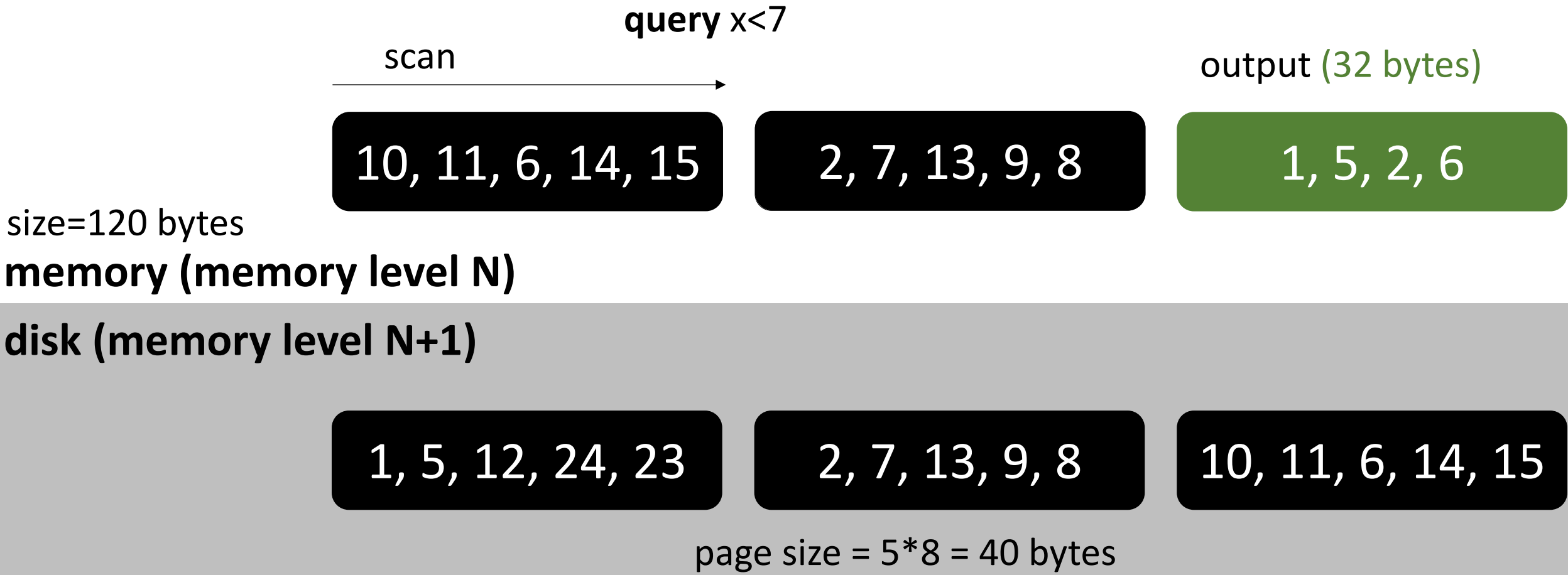
\$ 80 bytes

# page-based access & random access



\$ 120 bytes

# page-based access & random access



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

\$ 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

\$ 80 bytes

# page-based access & random access

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

\$ 80 bytes

# page-based access & random access

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



*was the oracle helpful?*

query  $x < 7$

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



# 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

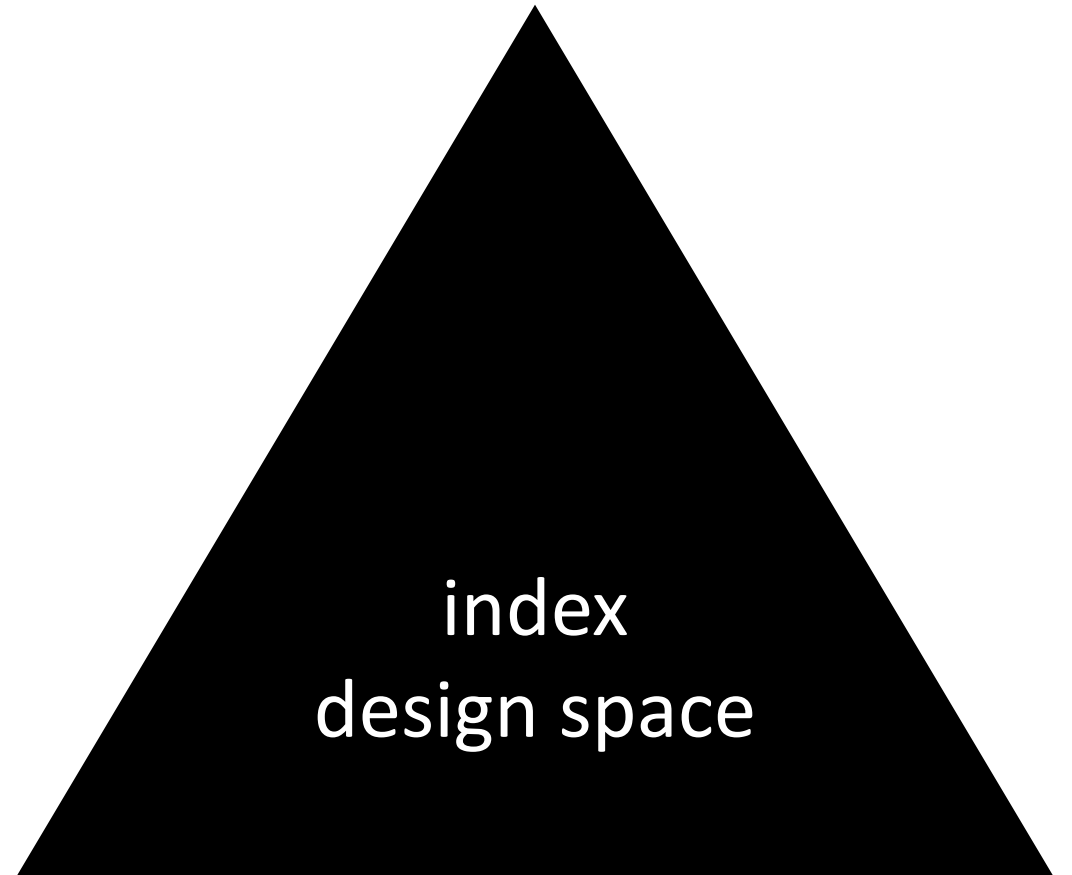
overheads and tradeoffs

how we store data

layouts, indexes

know the **query**

access path selection



# 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)

# zonemaps

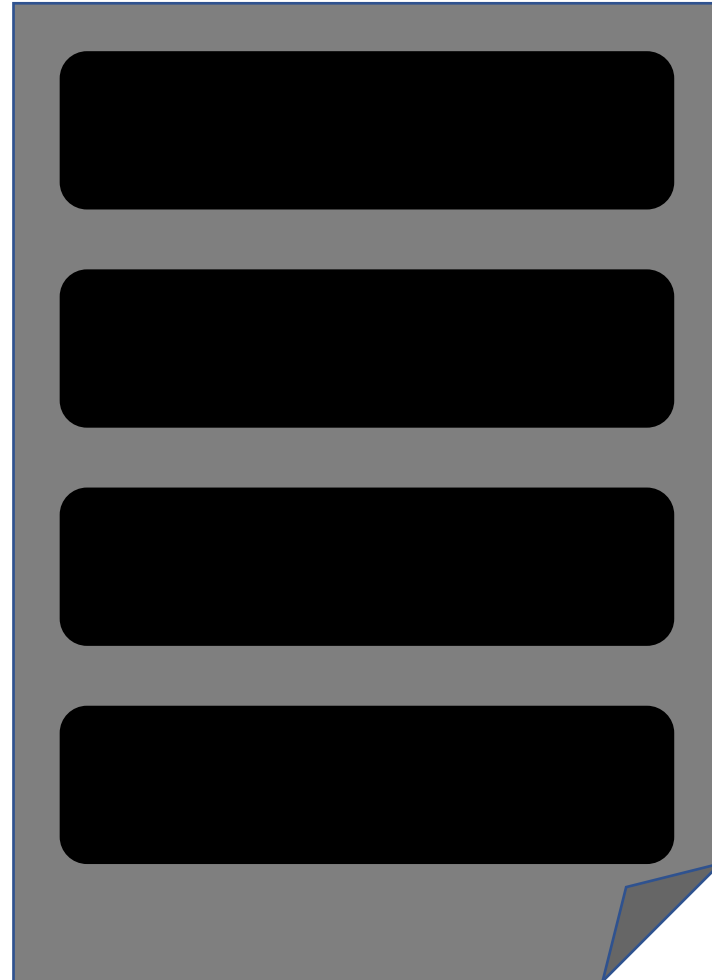
file = collection of pages

page 0

page 1

page 2

page 3



# zonemaps

file = collection of pages

page 0

3, 16, 34, 31, 21

page 1

1, 5, 12, 24, 23

page 2

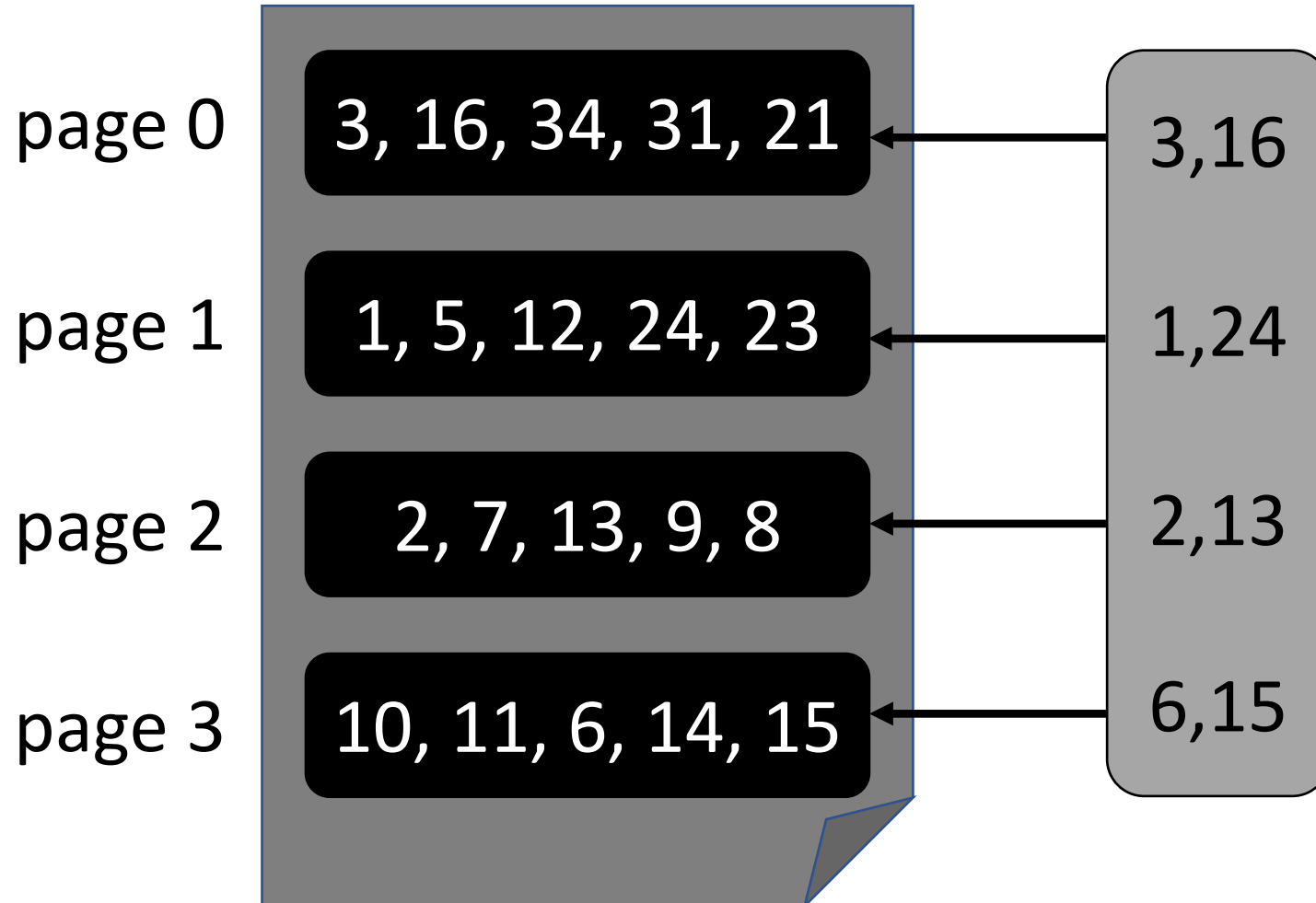
2, 7, 13, 9, 8

page 3

10, 11, 6, 14, 15

# zonemaps

file = collection of pages



*light-weight*

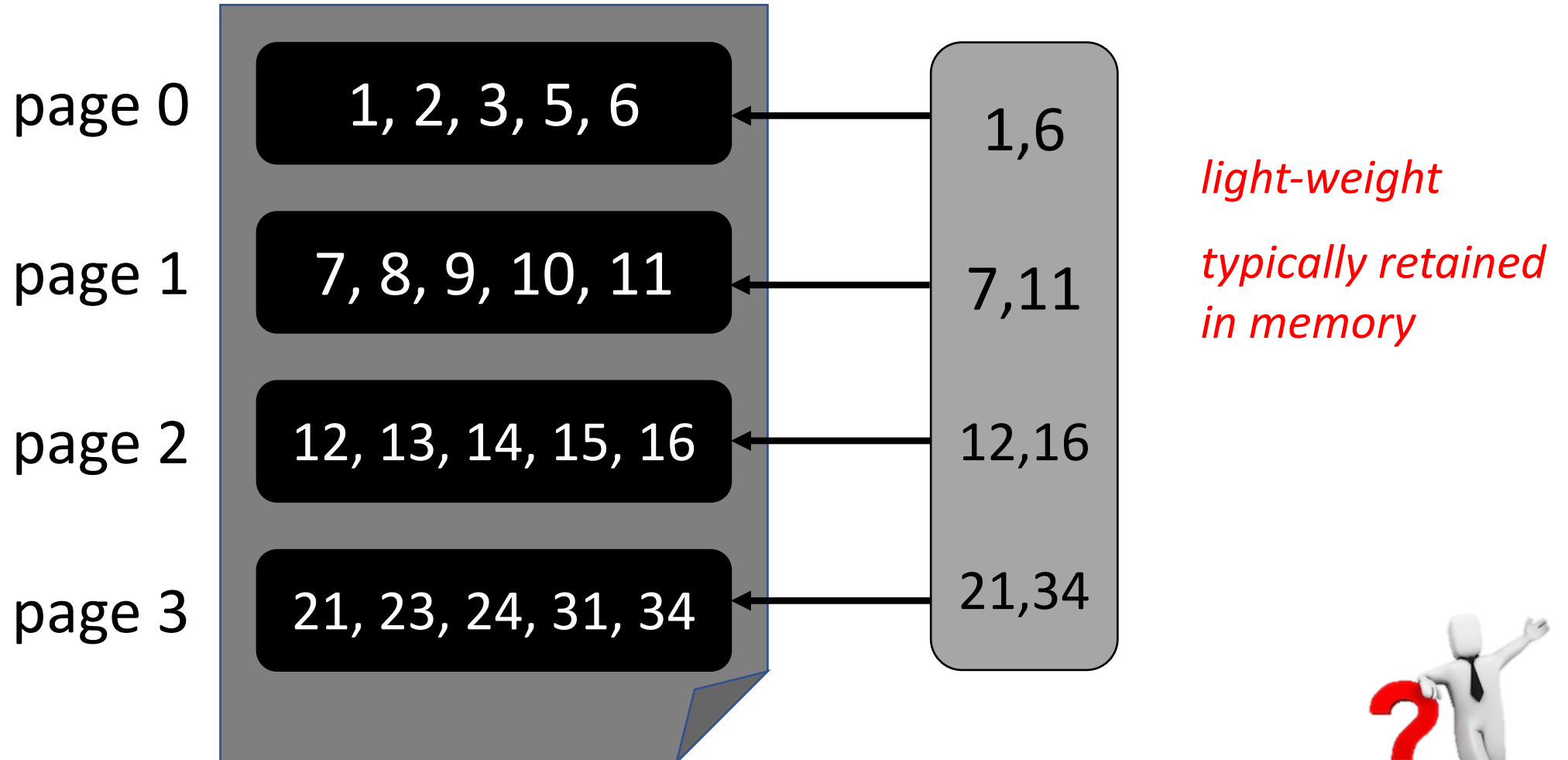
*typically retained  
in memory*

But what if the data is sorted?



# zonemaps

file = collection of pages



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

# CS 561: Data Systems Architectures

class 2

## Data Systems 101

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