BOSTON UNIVERSITY

CS 561: Data Systems Architectures

class 3

Column-Stores Basics

Prof. Manos Athanassoulis

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

Reviews

<u>4 reviews</u> and the **<u>rest single technical question</u>**

review (up to one page)

what is the problem & why it is important?
why is it hard & why older approaches are not enough?
what is key idea and why it works?
what is missing and how can we improve this idea?
does the paper supports its claims?
possible next steps of the work presented in the paper?

single technical question

to make sure the heart of the paper is clearly understood

Presentations

for every class, **2-3 students will be responsible for presenting** the paper (discussing all main points of a long review)

during the presentation **anyone can ask questions** (including me!) and each question is **addressed to all** (including me!)

the presenting student(s) will **prepare <u>slides</u> and <u>questions</u>**

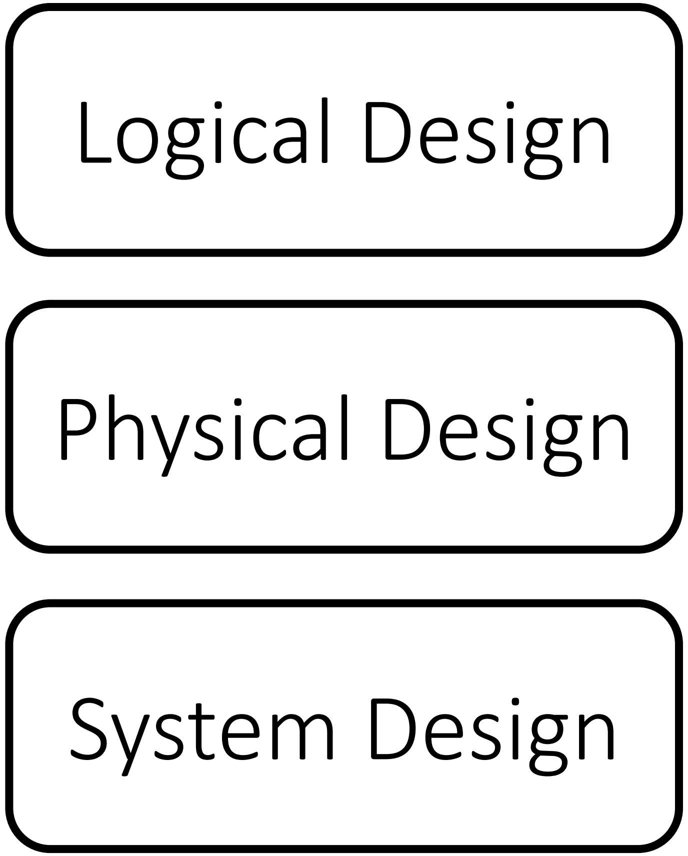
how can I prepare?

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

Database Design Abstraction Levels

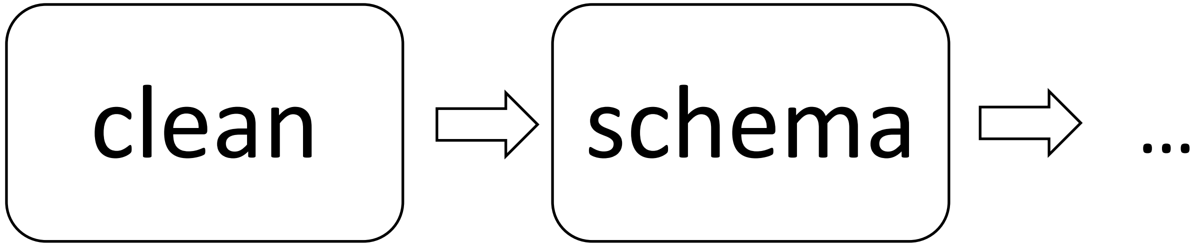


Logical Design

Physical Design

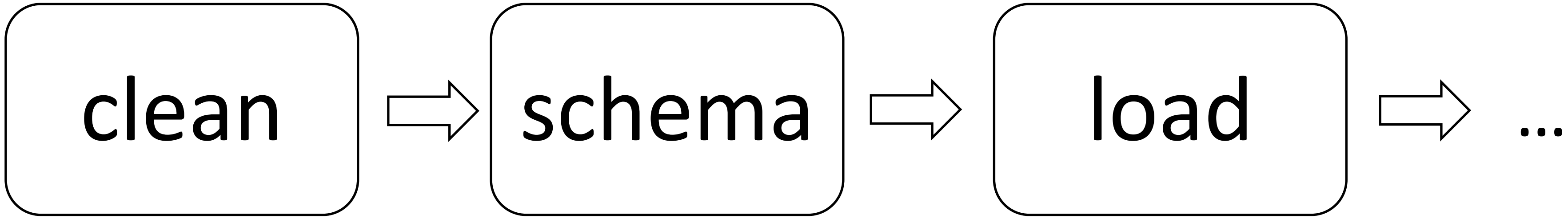
System Design

Data can be messy!



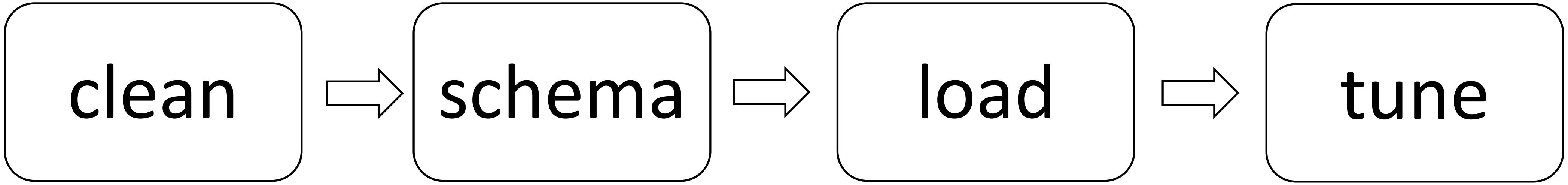
clean ⇨ schema ⇨ …

Data can be messy!



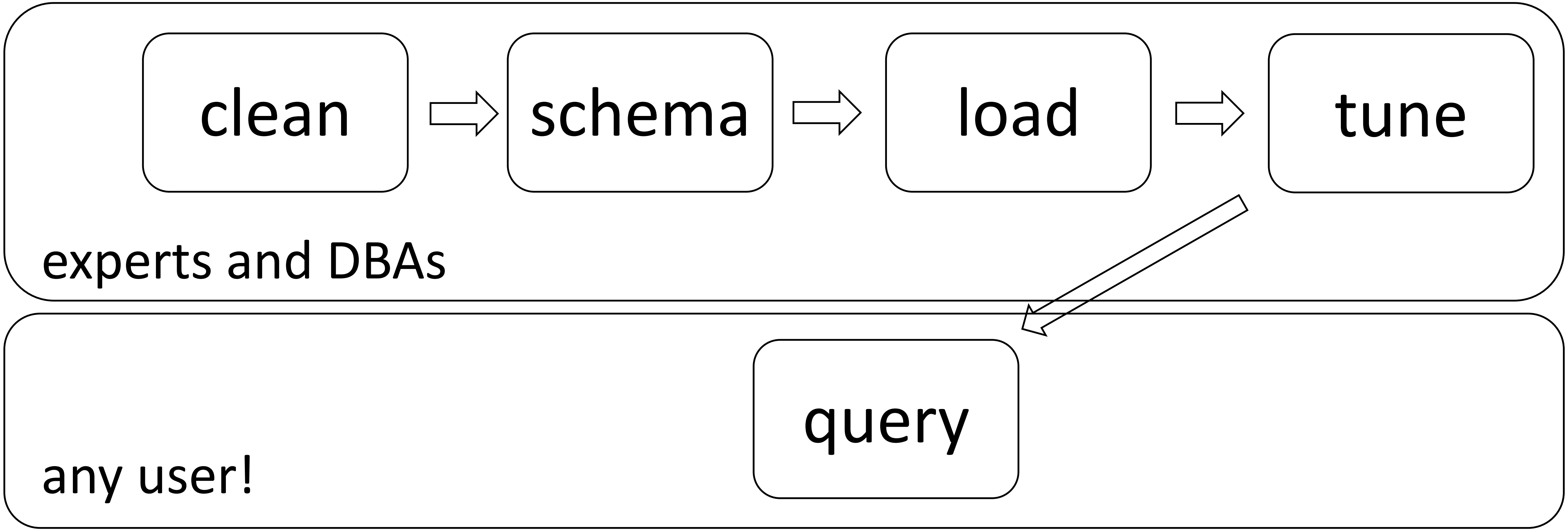
clean ⇨ schema ⇨ load ⇨ …

Data can be messy!



clean → schema → load → tune

Data can be messy!



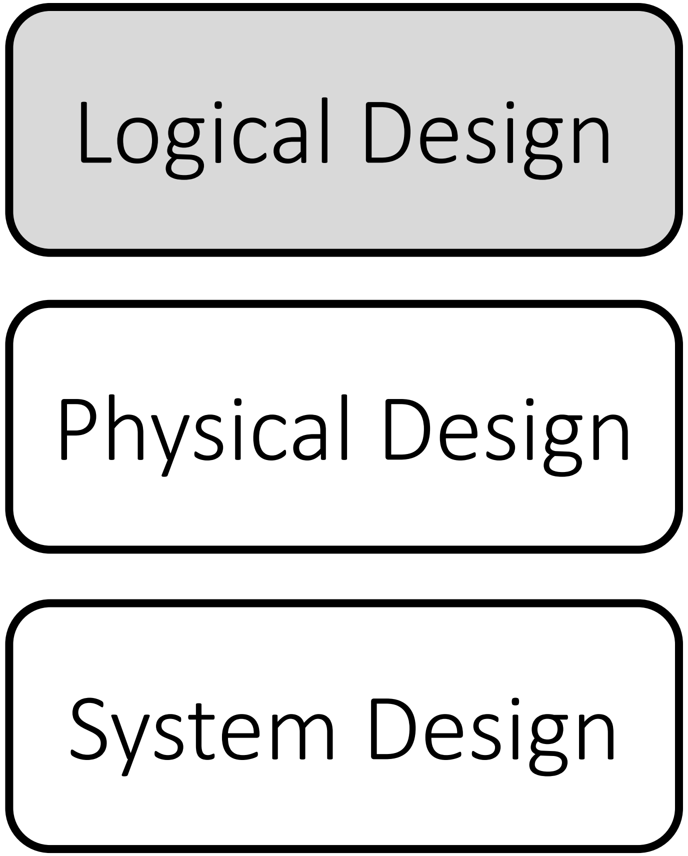
clean → schema → load → tune

experts and DBAs

query

any user!

Database Design Abstraction Levels



Logical Design

Physical Design

System Design

Logical design

What is our data? How to model them?

Hierarchical? Network? Object-oriented? Flat? Key-Value?

Relational!

A collection of **tables**, each being a collection of **rows and columns**

[schema: describes the columns of each table]

Logical design

What is our data? How to model them?

graph data
time-series data

A collection of **tables**, each being a collection of **rows and columns**

[schema: describes the columns of each table]

Logical Schema of “University” Database

Students

***sid**: string, **name**: string, **login**: string, **year_birth**: integer, **gpa**: real*

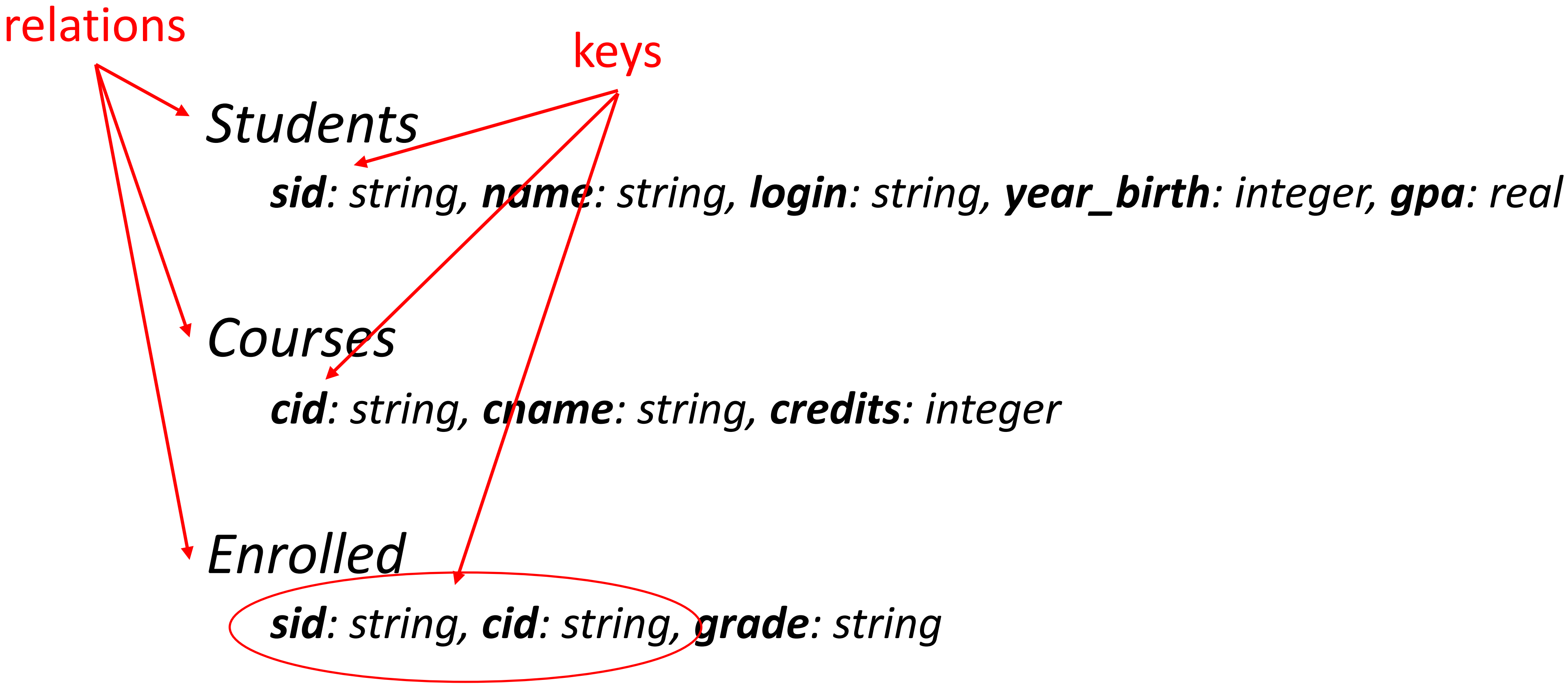
Courses

***cid**: string, **cname**: string, **credits**: integer*

Enrolled

***sid**: string, **cid**: string, **grade**: string*

Relational Model and SQL



relations

keys

Students

***sid**: string, **name**: string, **login**: string, **year_birth**: integer, **gpa**: real*

Courses

***cid**: string, **cname**: string, **credits**: integer*

Enrolled

***sid**: string, **cid**: string, **grade**: string*

Relational Model and SQL

how to create the table students?

***create table** students (sid:char(10), name:char(40), login:char(8), age:integer, …)*

Students

***sid**: string, **name**: string, **login**: string, **year_birth**: integer, **gpa**: real*

how to add a new student?

***insert into** students (U1398217312, John Doe, john19, 19, …)*

Courses

***cid**: string, **cname**: string, **credits**: integer*

bring me the names of all students

***select** name **from** students **where** GPA > 3.5*

Enrolled

***sid**: string, **cid**: string, **grade**: string*

Relational Model and SQL

student

(sid1, name1, login1, year1, gpa1)
(sid2, name2, login2, year2, gpa2)
(sid3, name3, login3, year3, gpa3)
(sid4, name4, login4, year4, gpa4)
(sid5, name5, login5, year5, gpa5)
(sid6, name6, login6, year6, gpa6)
(sid7, name7, login7, year7, gpa7)
(sid8, name8, login8, year8, gpa8)
(sid9, name9, login9, year9, gpa9)

insert into student (sid1, name1, login1, year1, gpa1)

cardinality: 9

Relational Model and SQL

student

(sid1, name1, login1, year1, gpa1)
(sid2, name2, login2, year2, gpa2)
(sid3, name3, login3, year3, gpa3)
(sid4, name4, login4, year4, gpa4)
(sid5, name5, login5, year5, gpa5)
(sid6, name6, login6, year6, gpa6)
(sid7, name7, login7, year7, gpa7)
(sid8, name8, login8, year8, gpa8)
(sid9, name9, **login9**, year9, gpa9)

insert into student (sid1, name1, login1, year1, gpa1)

cardinality: 9

what if a student does not have their login yet?

Relational Model and SQL

student

(sid1, name1, login1, year1, gpa1)
(sid2, name2, login2, year2, gpa2)
(sid3, name3, login3, year3, gpa3)
(sid4, name4, login4, year4, gpa4)
(sid5, name5, login5, year5, gpa5)
(sid6, name6, login6, year6, gpa6)
(sid7, name7, login7, year7, gpa7)
(sid8, name8, login8, year8, gpa8)
(sid9, name9, **NULL**, year9, gpa9)

insert into student (sid1, name1, login1, year1, gpa1)

cardinality: 9

what if a student does not have their login yet?

NULL values do not exist

Relational Model and SQL

how to show all enrollments in CS561?

keys

Students

***sid**: string, **name**: string, **login**: string, **year_birth**: integer, **gpa**: real*

Courses

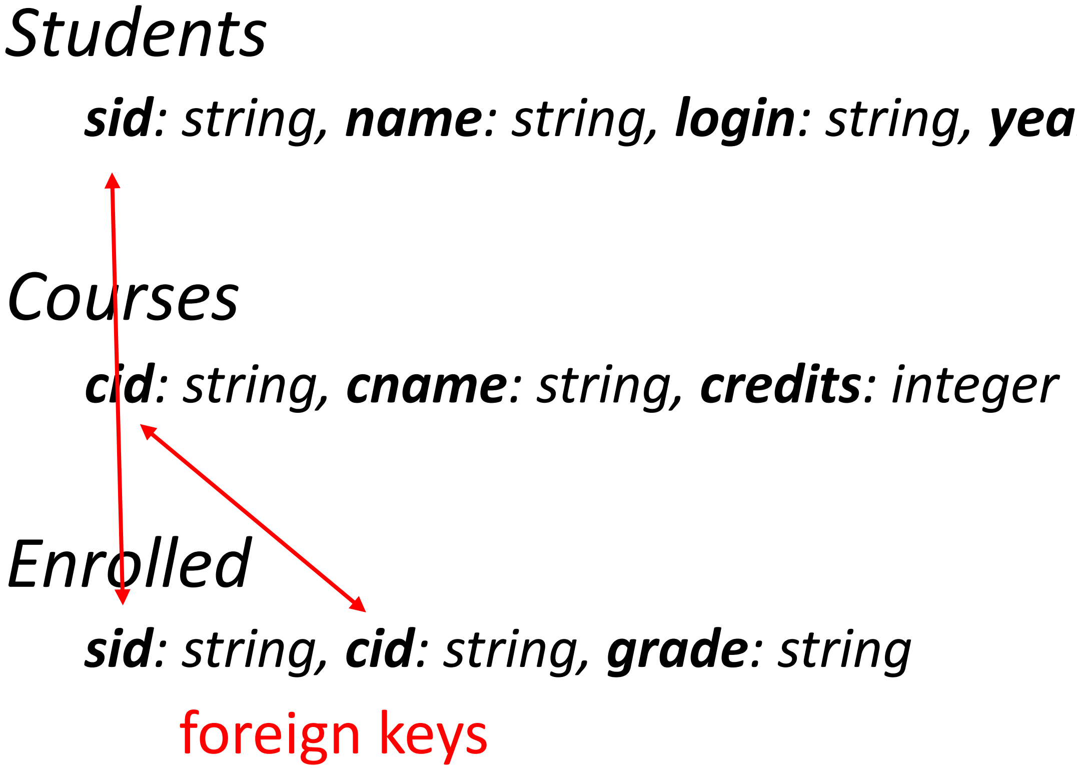
***cid**: string, **cname**: string, **credits**: integer*

Enrolled

***sid**: string, **cid**: string, **grade**: string*

Relational Model and SQL

how to show all enrollments in CS561?



Students

sid*: string,* ***name****: string,* ***login****: string,* ***year_birth****: integer,* ***gpa****: real*

Courses

cid*: string,* ***cname****: string,* ***credits****: integer*

Enrolled

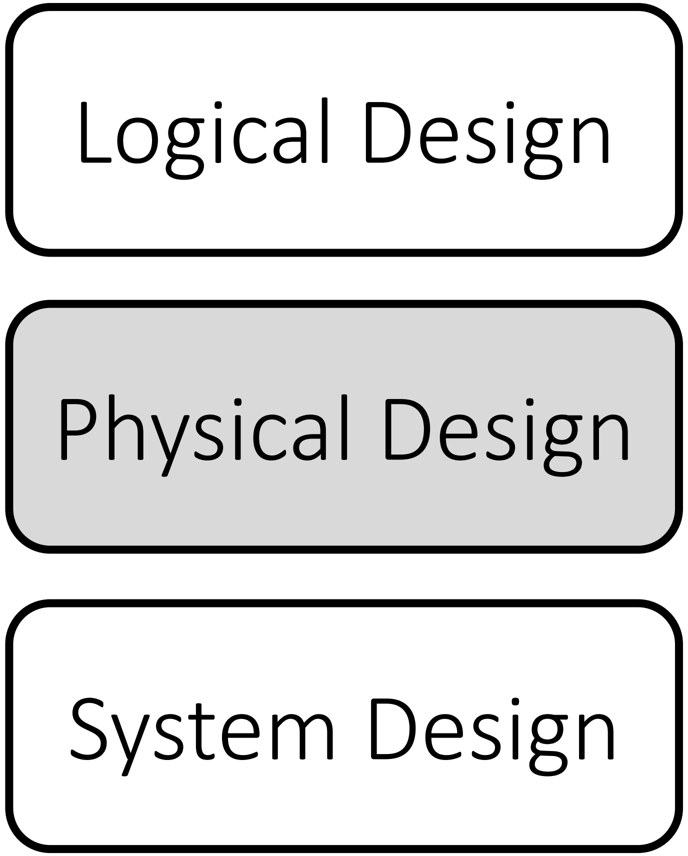
sid*: string,* ***cid****: string,* ***grade****: string*

foreign keys

using foreign keys we can join information of all three tables

select student.name
from students, courses, enrolled
where course.cname="CS561"
and course.cid=enrolled.cid
and student.sid=enrolled.sid

Database Design Abstraction Levels



Logical Design

Physical Design

System Design

Physical Design

File Organization

heap files

sorted files

clustered files

more …

Indexes

should I build?

on which attributes/tables?

what index structure?

B-Tree

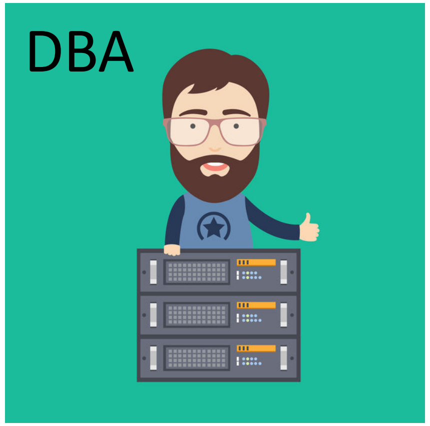
Tries

Hash

Bitmap

Zonemaps

Data systems are declarative!



ask ***what*** you want

data system

system decides ***how***
to store & access

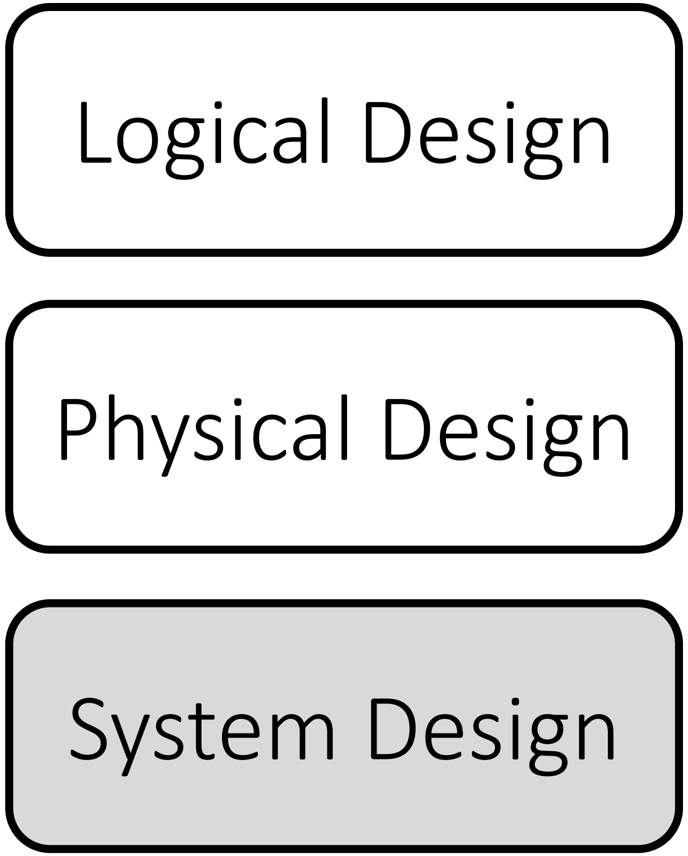
design decisions, physical design
indexing, tuning knobs

research to automate!

adaptivity

autotuning

Database Design Abstraction Levels



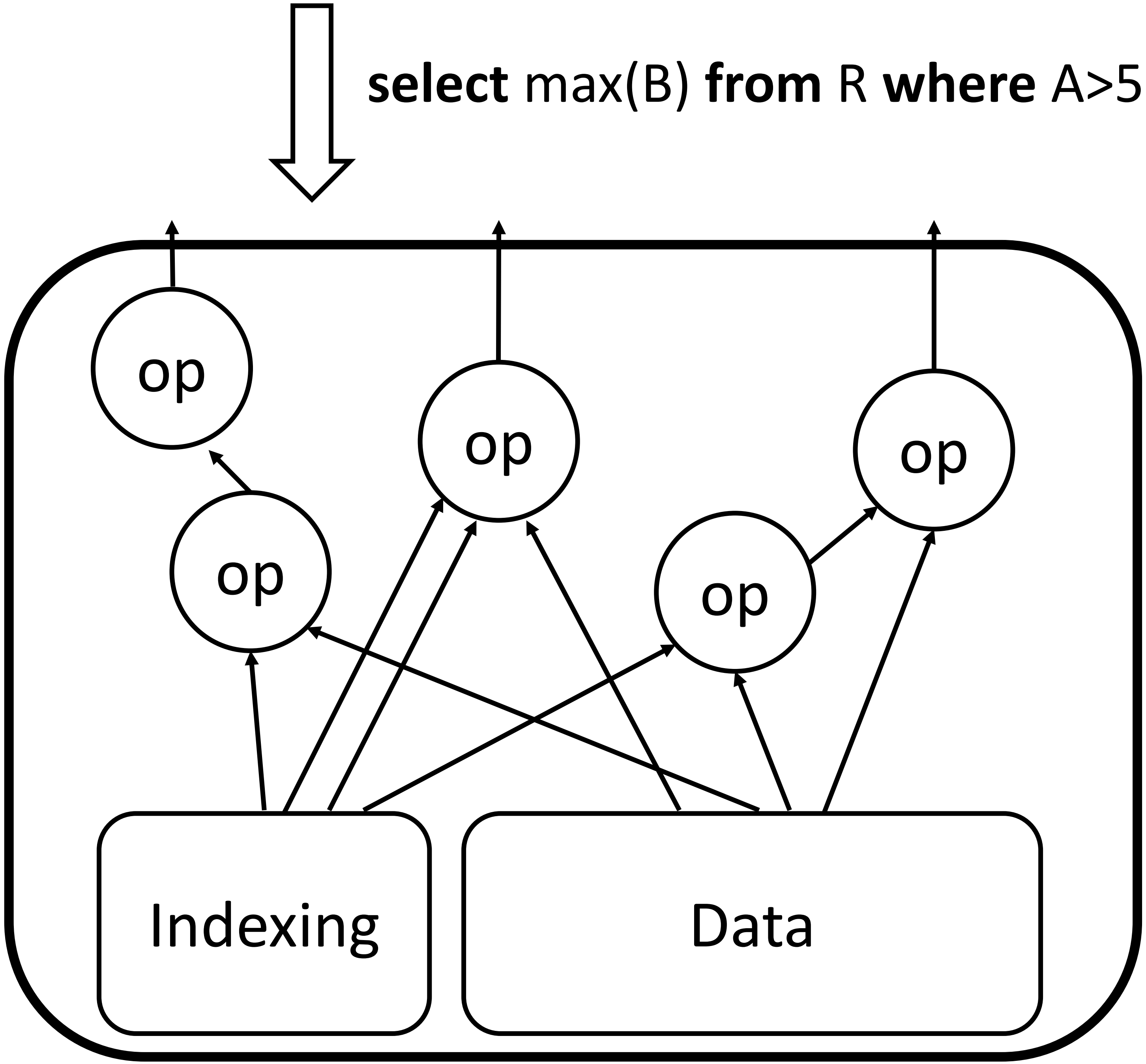
Logical Design

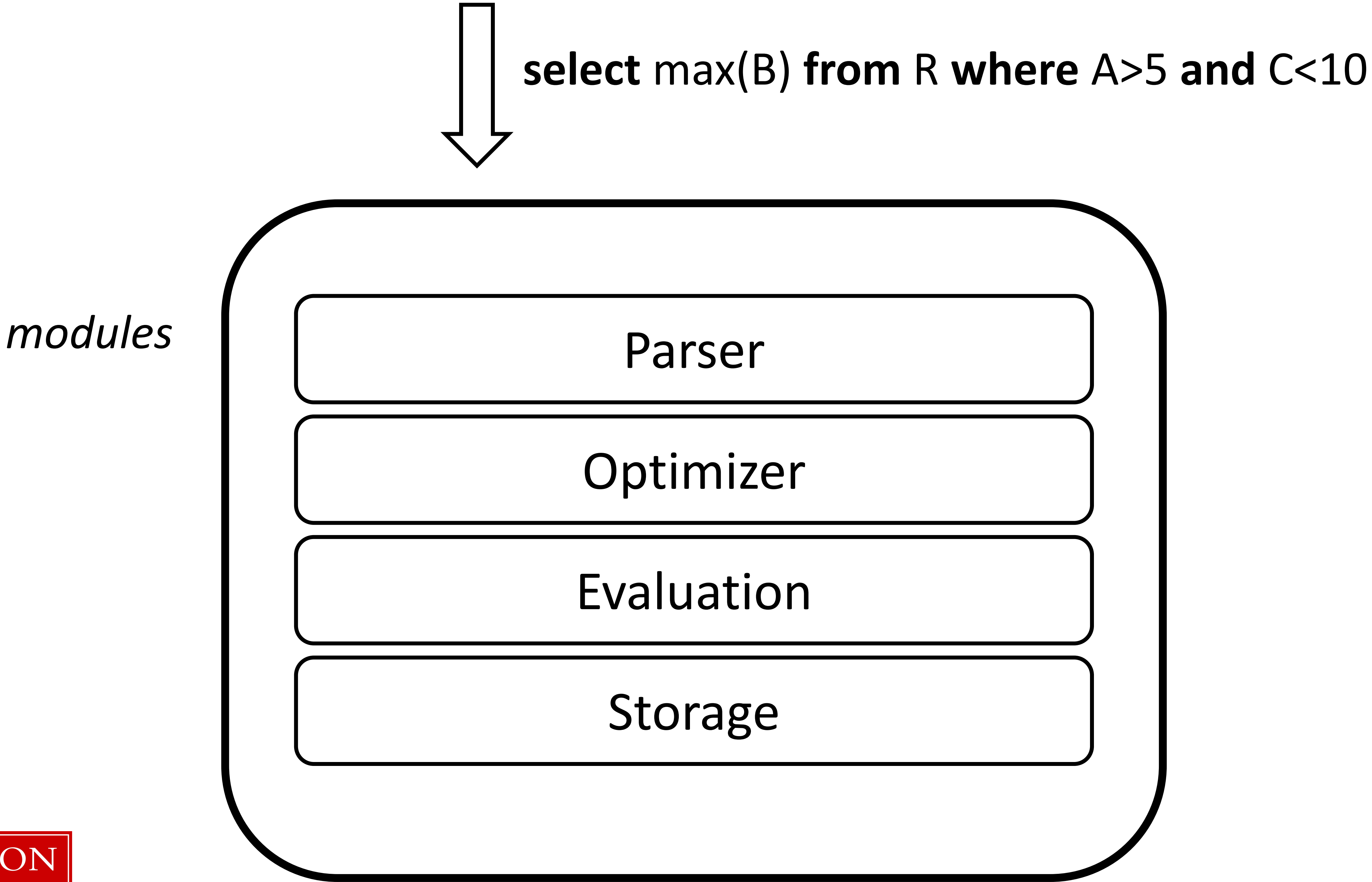
Physical Design

System Design

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

algorithms and operators





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

modules

Parser

Optimizer

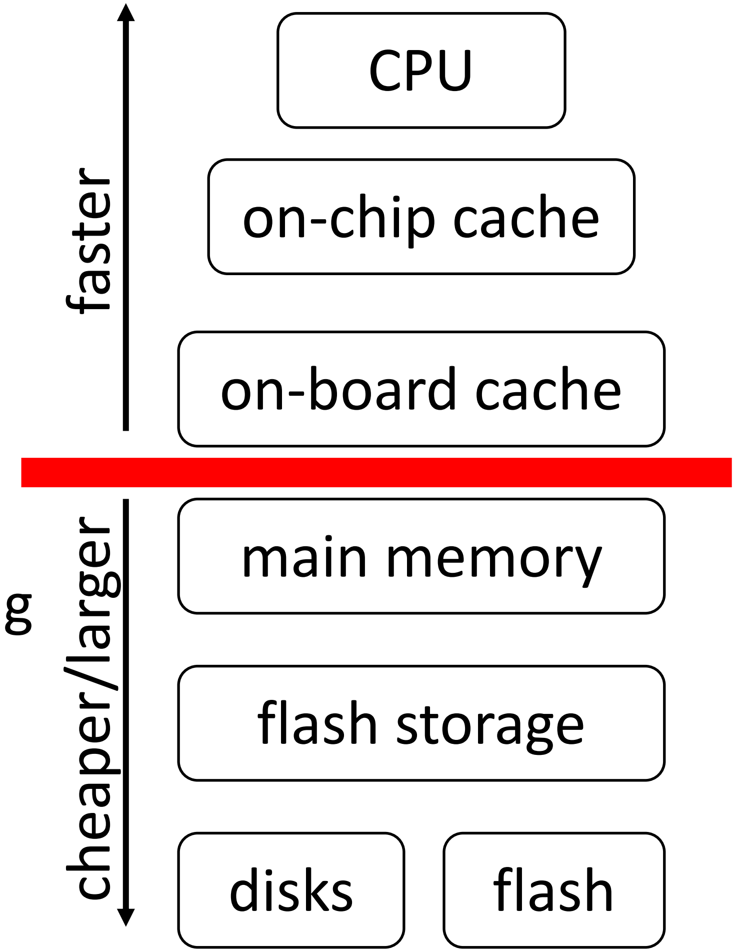
Evaluation

Storage

memory wall

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

memory miss: looking for something that is not in memory


faster

cheaper/larger

CPU

on-chip cache

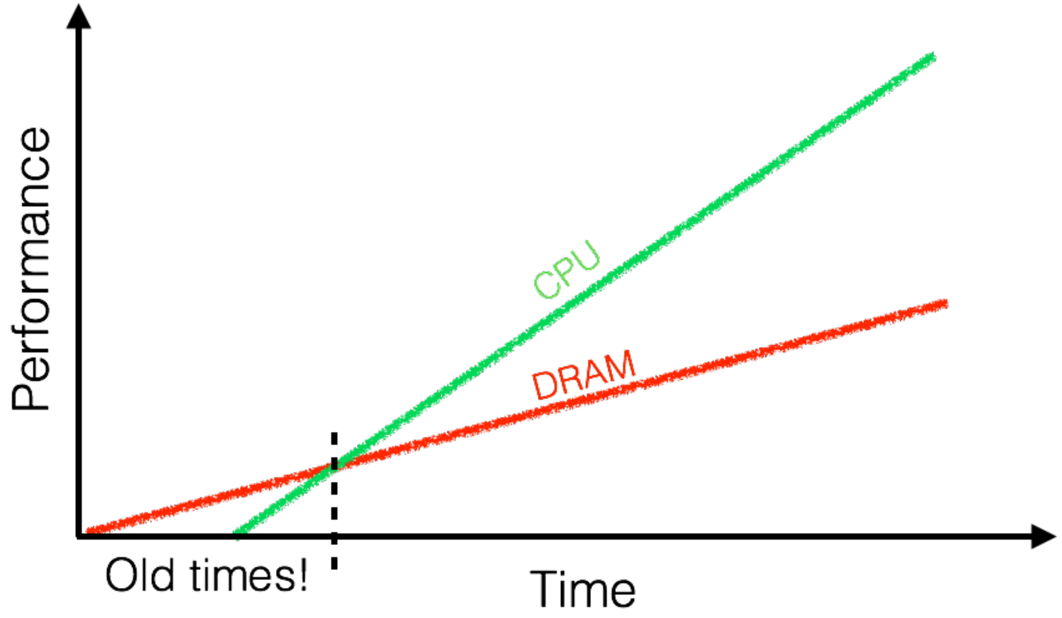
on-board cache

main memory

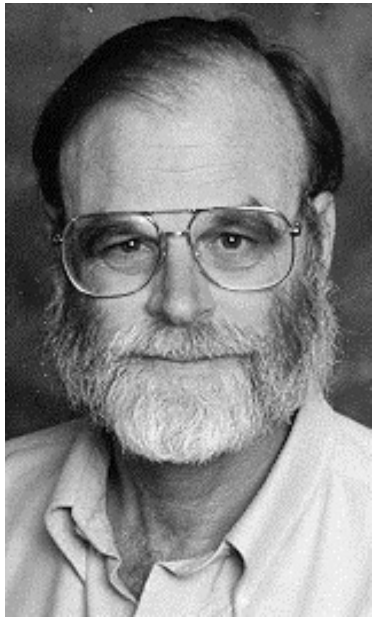
flash storage

disks

flash



memory hierarchy (by Jim Gray)

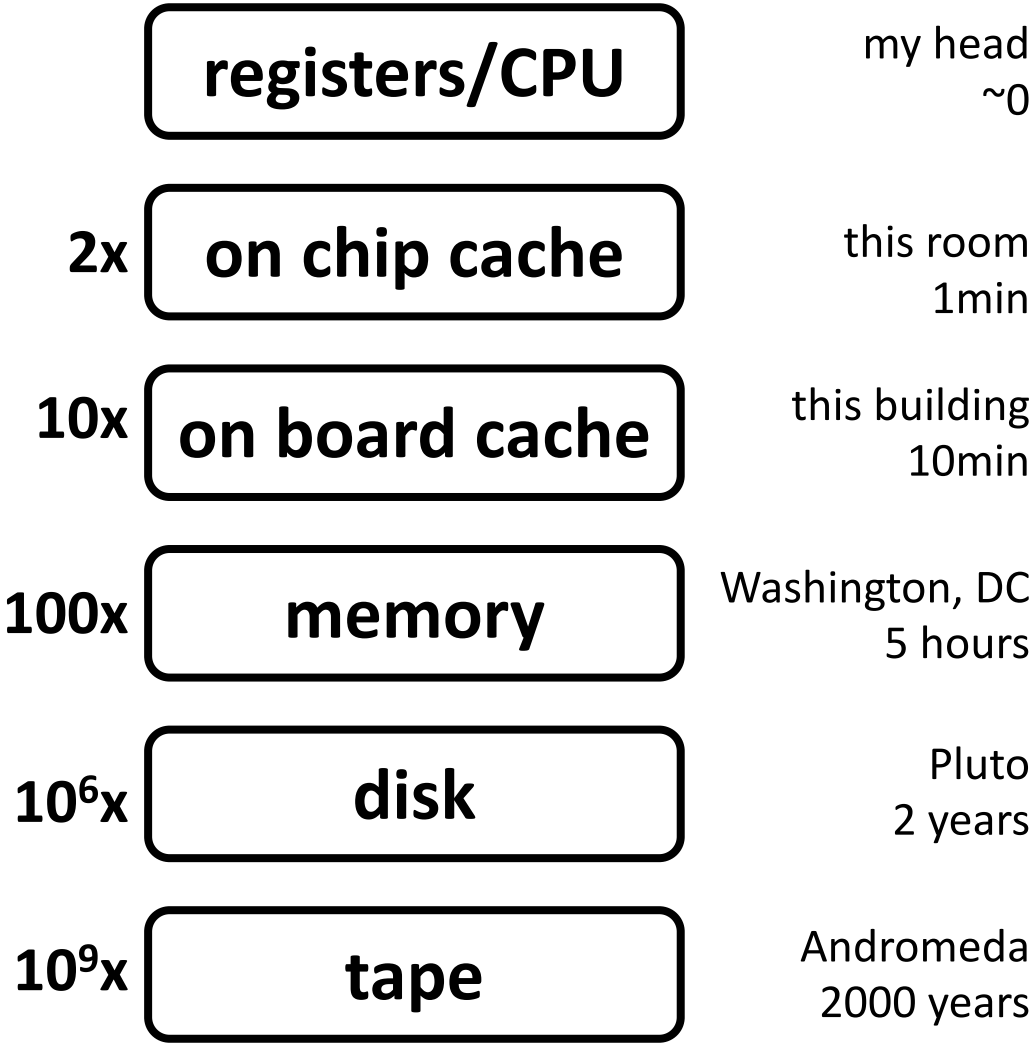


Jim Gray, IBM, Tandem, Microsoft, DEC

“The Fourth Paradigm” is based on his vision

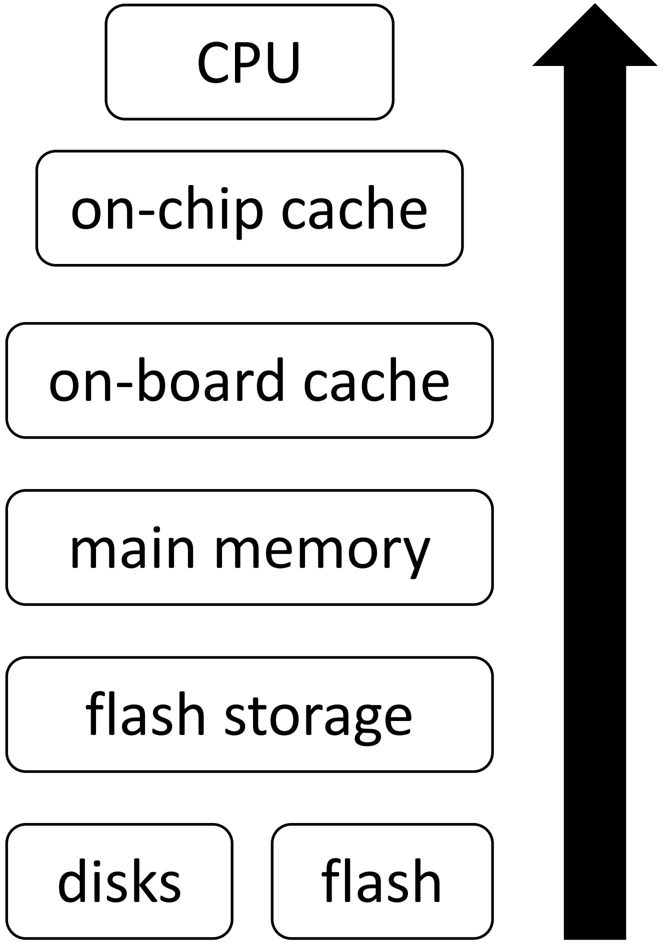
ACM Turing Award 1998

ACM SIGMOD Edgar F. Codd Innovations award 1993



	registers/CPU	my head ~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

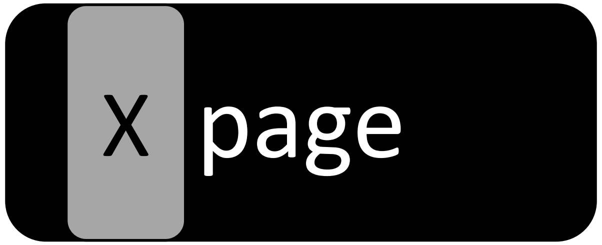
data movement & page-based access



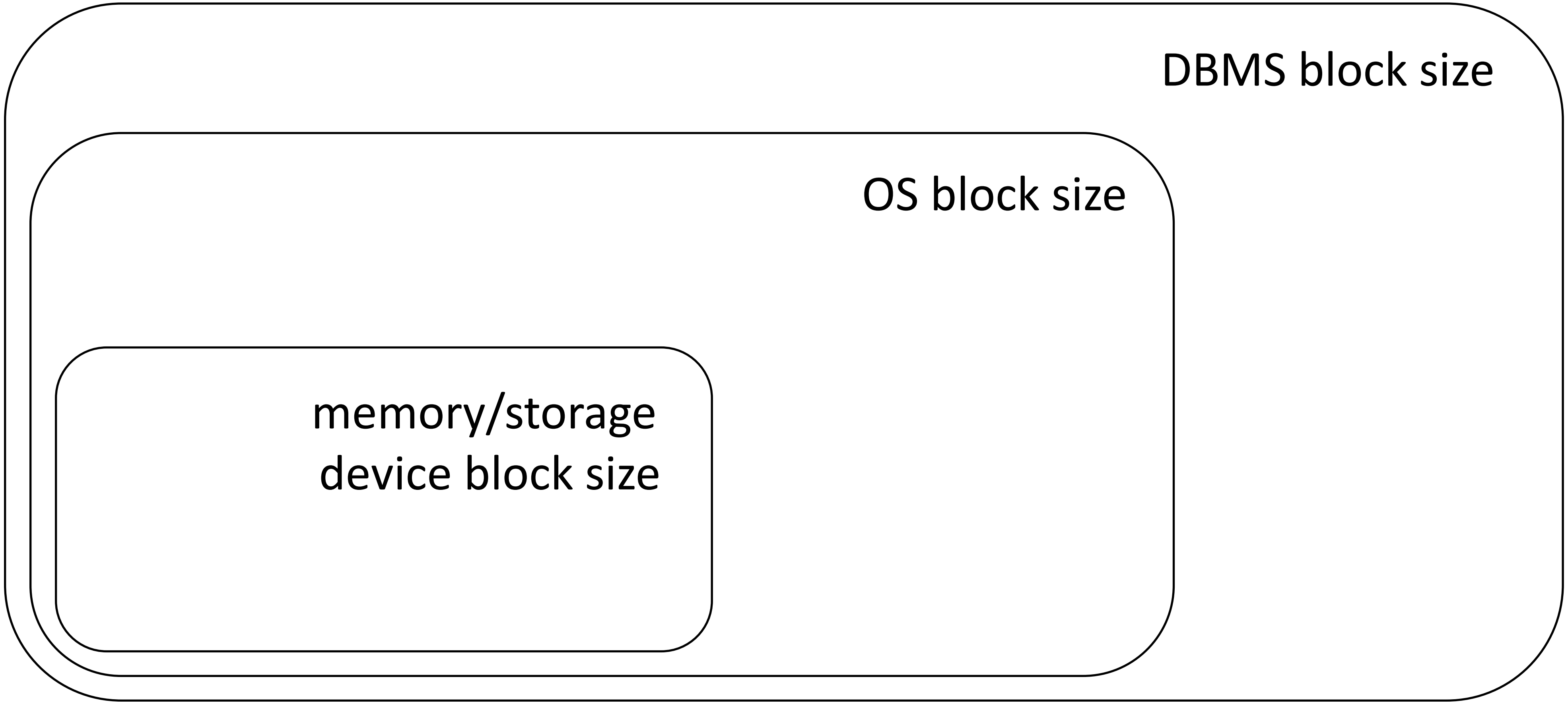
data go through
all necessary levels

also read
unnecessary data

need to read only X
read the whole page



access granularity



file system and DBMS “pages”

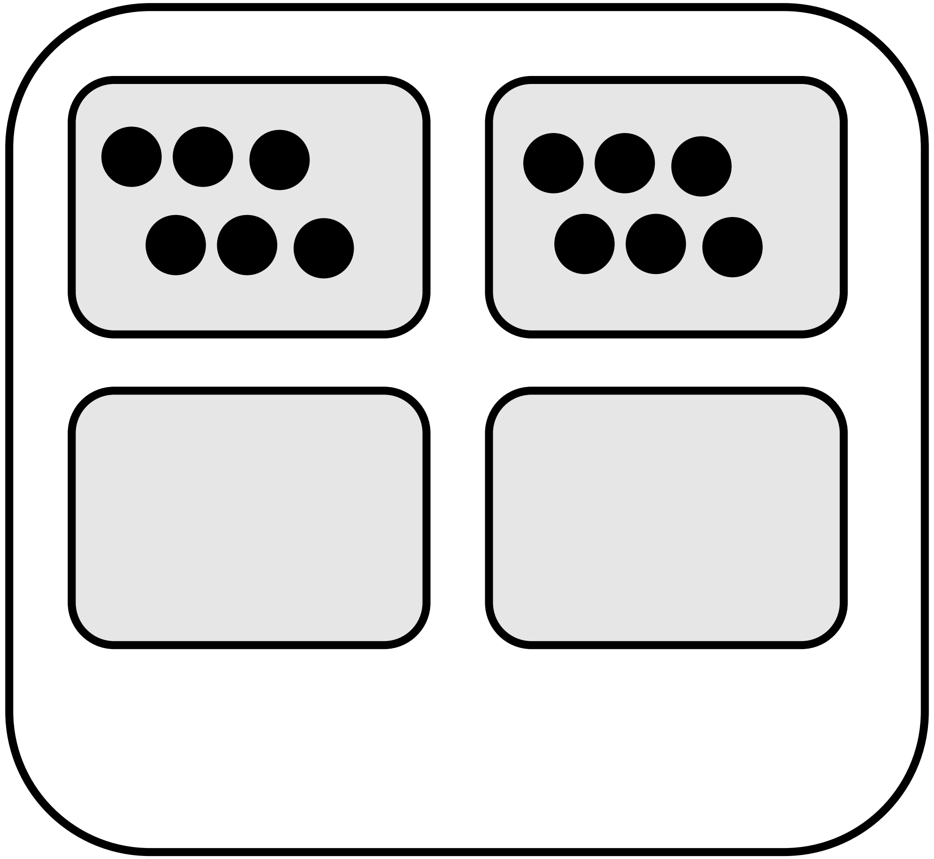
data storage

how to physically place data?

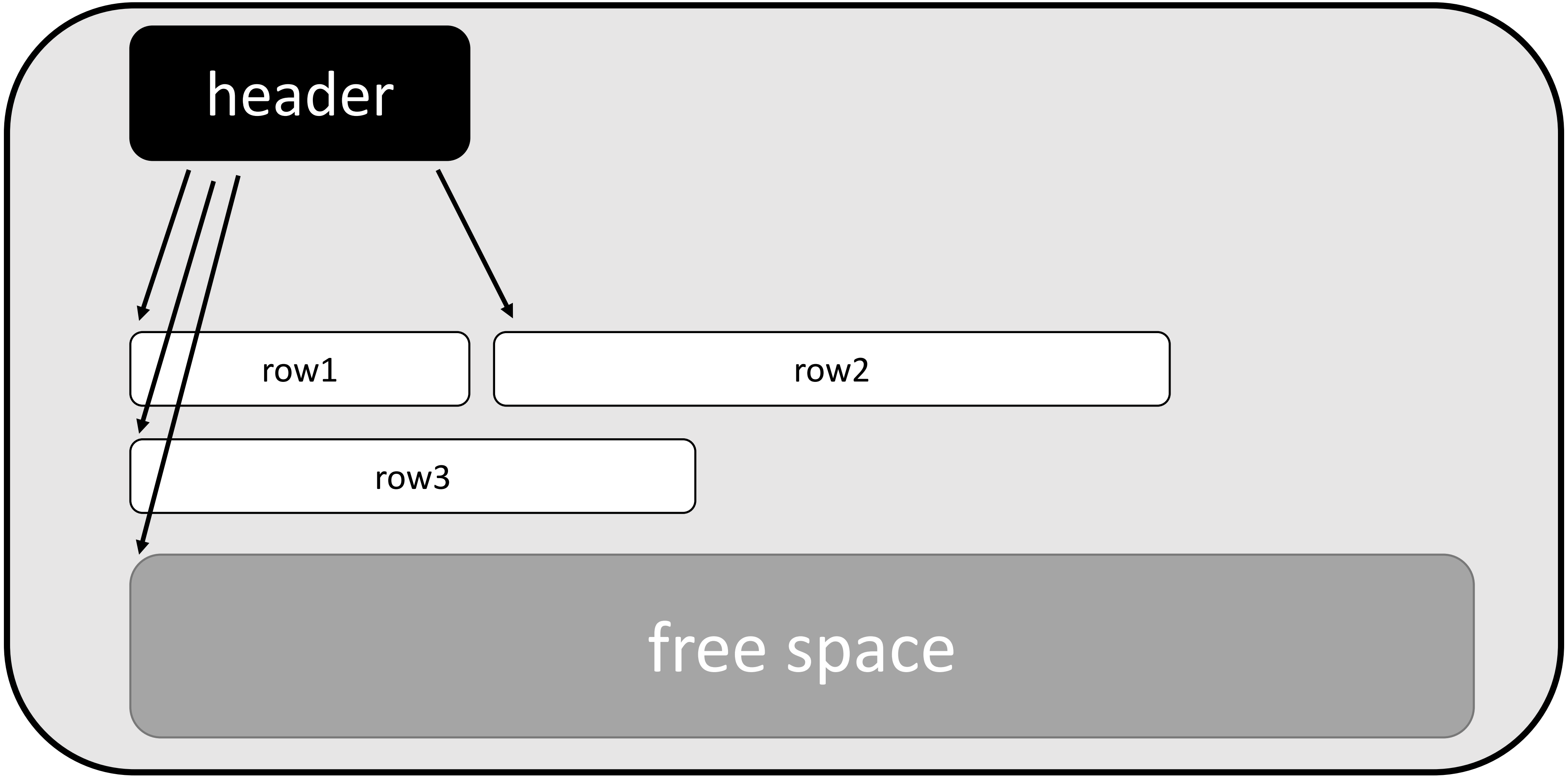
*Student (**sid**: string, **name**: string, **login**: string, **year_birth**: integer, **gpa**: real)*

student

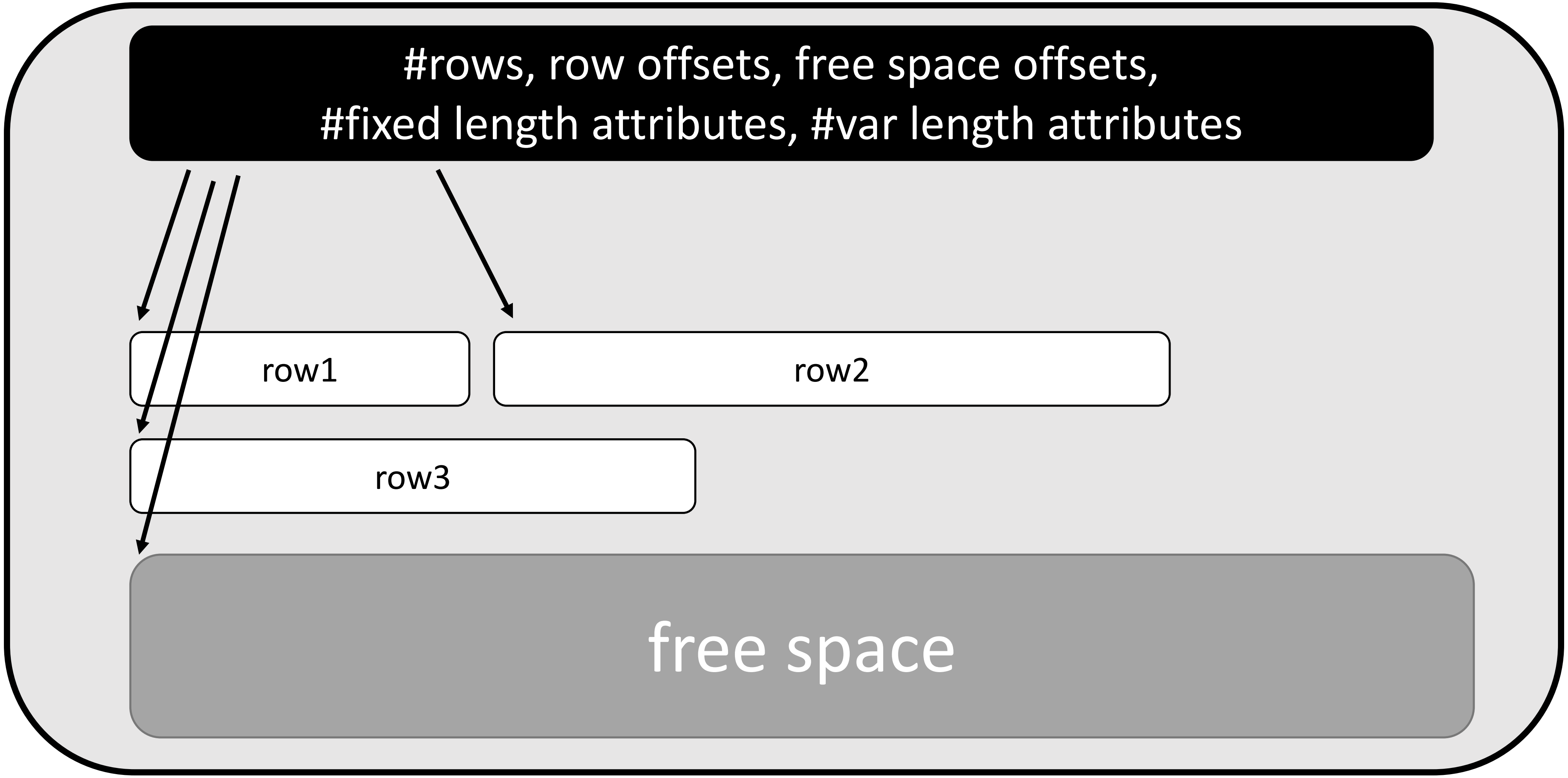
(sid1, name1, login1, year1, gpa1)
(sid2, name2, login2, year2, gpa2)
(sid3, name3, login3, year3, gpa3)
(sid4, name4, login4, year4, gpa4)
(sid5, name5, login5, year5, gpa5)
(sid6, name6, login6, year6, gpa6)
(sid7, name7, login7, year7, gpa7)
(sid8, name8, login8, year8, gpa8)
(sid9, name9, login9, year9, gpa9)



slotted page

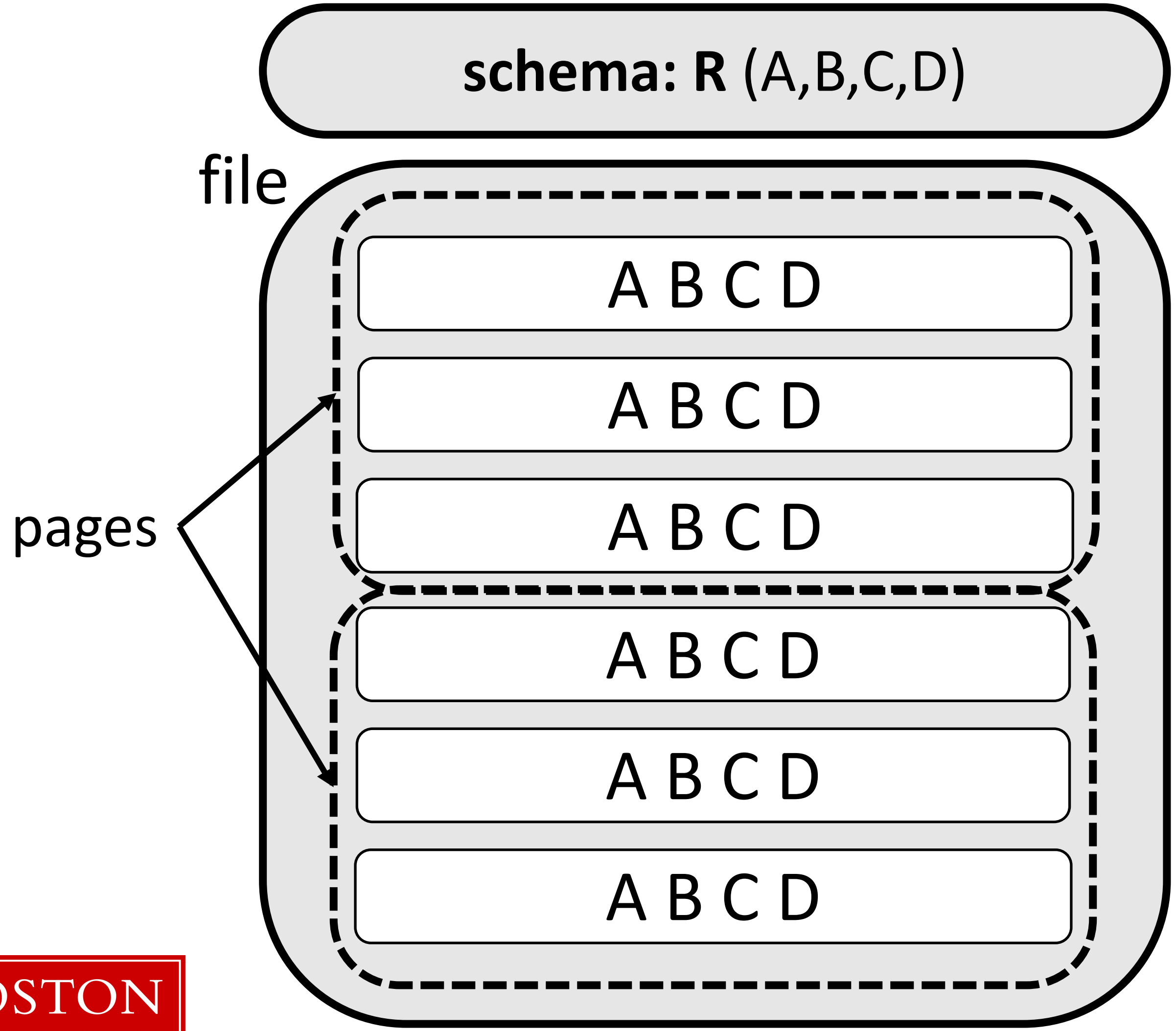


slotted page



#rows, row offsets, free space offsets,
#fixed length attributes, #var length attributes

querying over slotted pages



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

file

pages

A B C D

A B C D

A B C D

A B C D

A B C D

A B C D

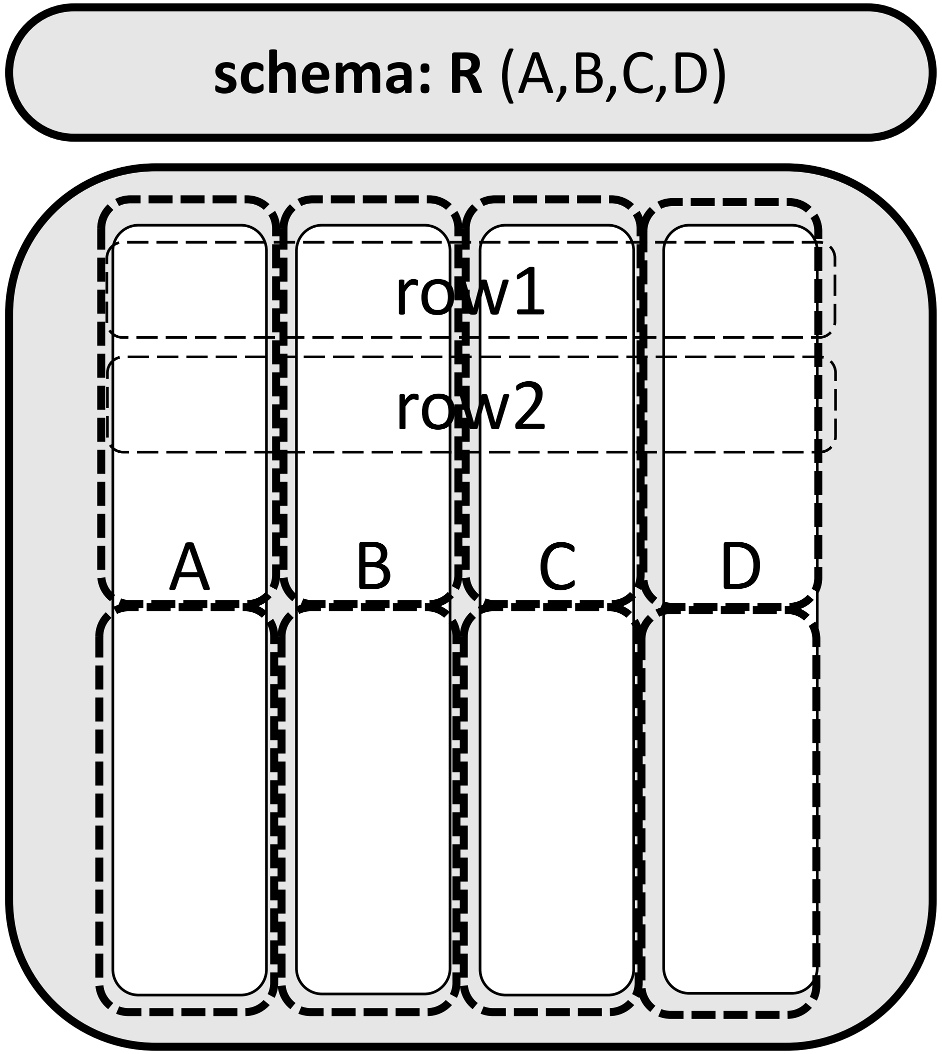
select A,B,C,D **from** R

select A **from** R

each page contains **entire** rows (all their columns)

rows are **contiguous**
(with possible free space at the end)

querying over slotted pages



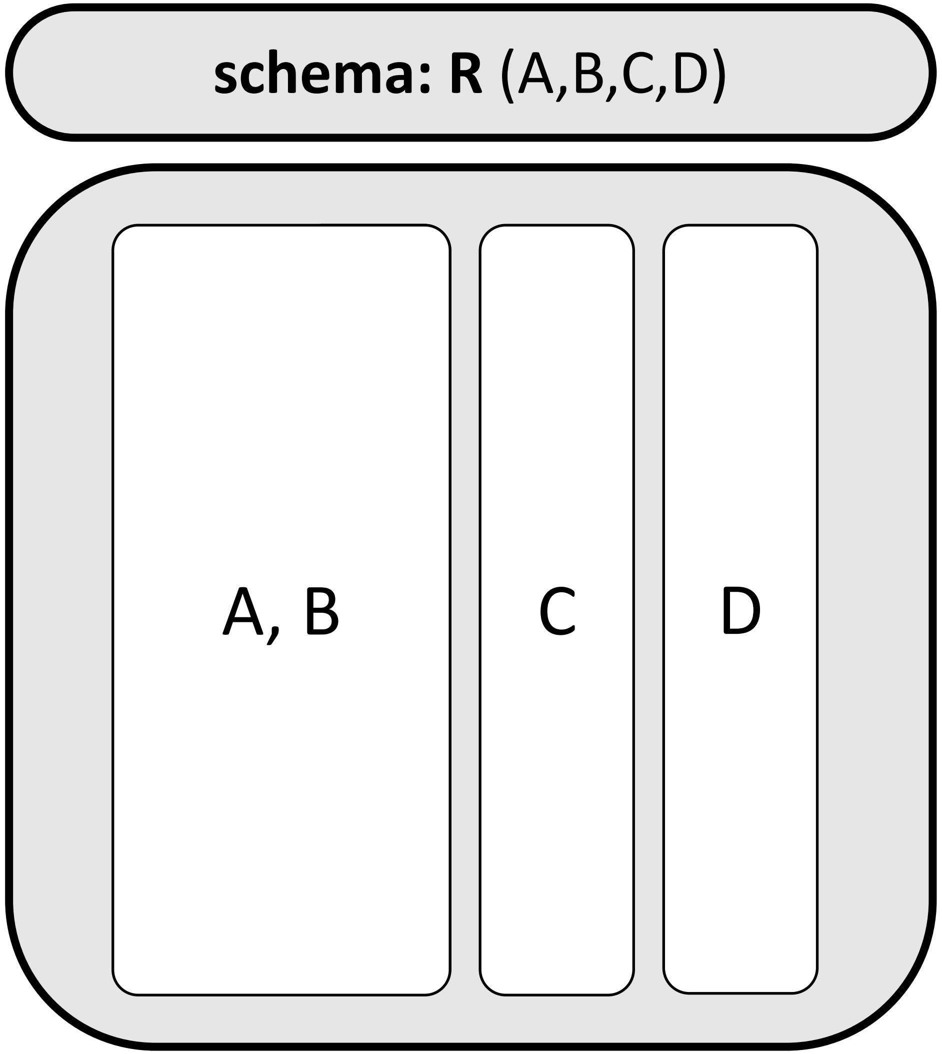
each page contains **columns!**

select A,B,C,D **from** R

select A **from** R

select (A+B) **from** R

querying over slotted pages



select A,B,C,D **from** R

select A **from** R

select (A+B) **from** R

each page contains **columns** or ***groups of columns*****!**

what if I had both queries?

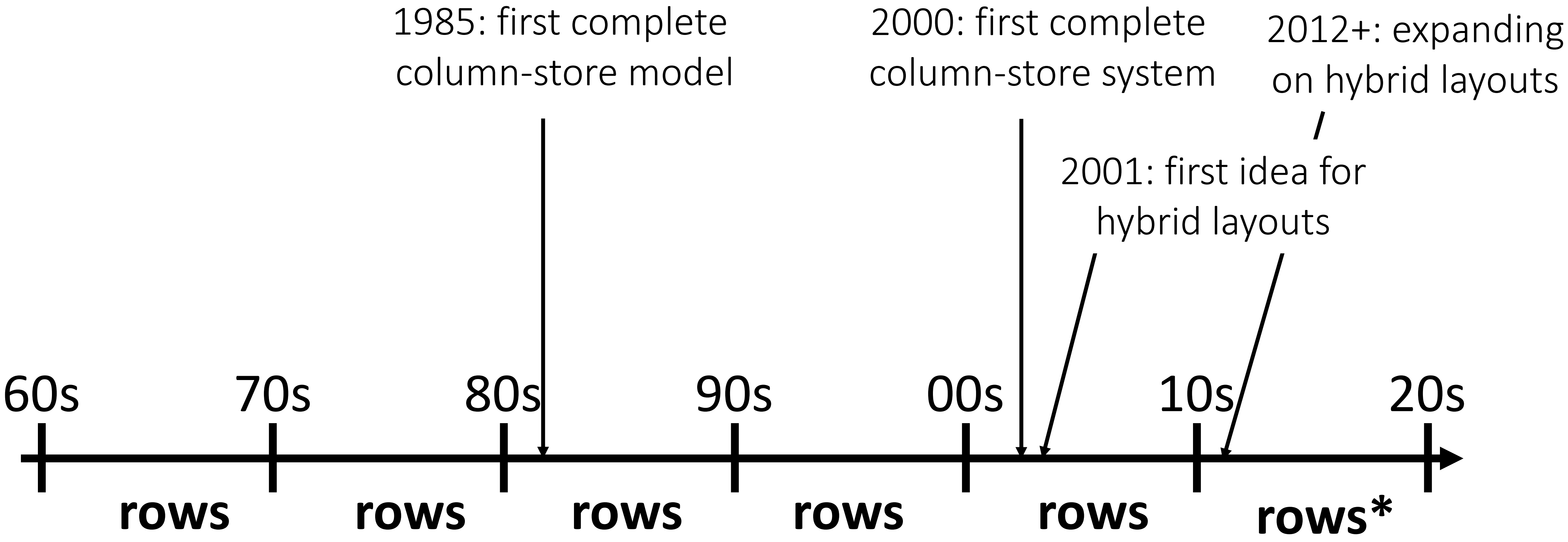
not clear!

other hybrids?

what if only inserts?

BOSTON UNIVERSITY

column-stores history line



1985: first complete column-store model

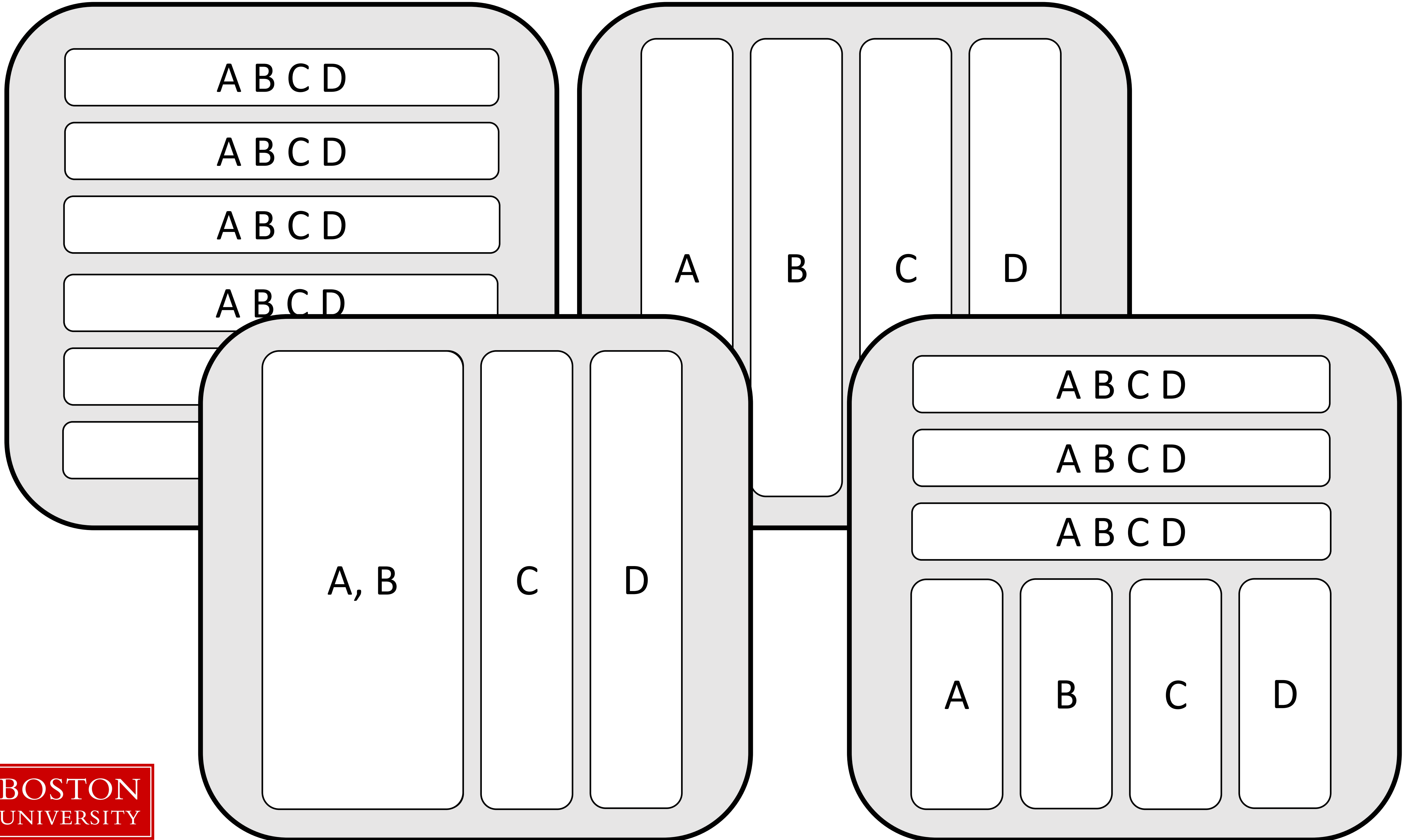
2000: first complete column-store system

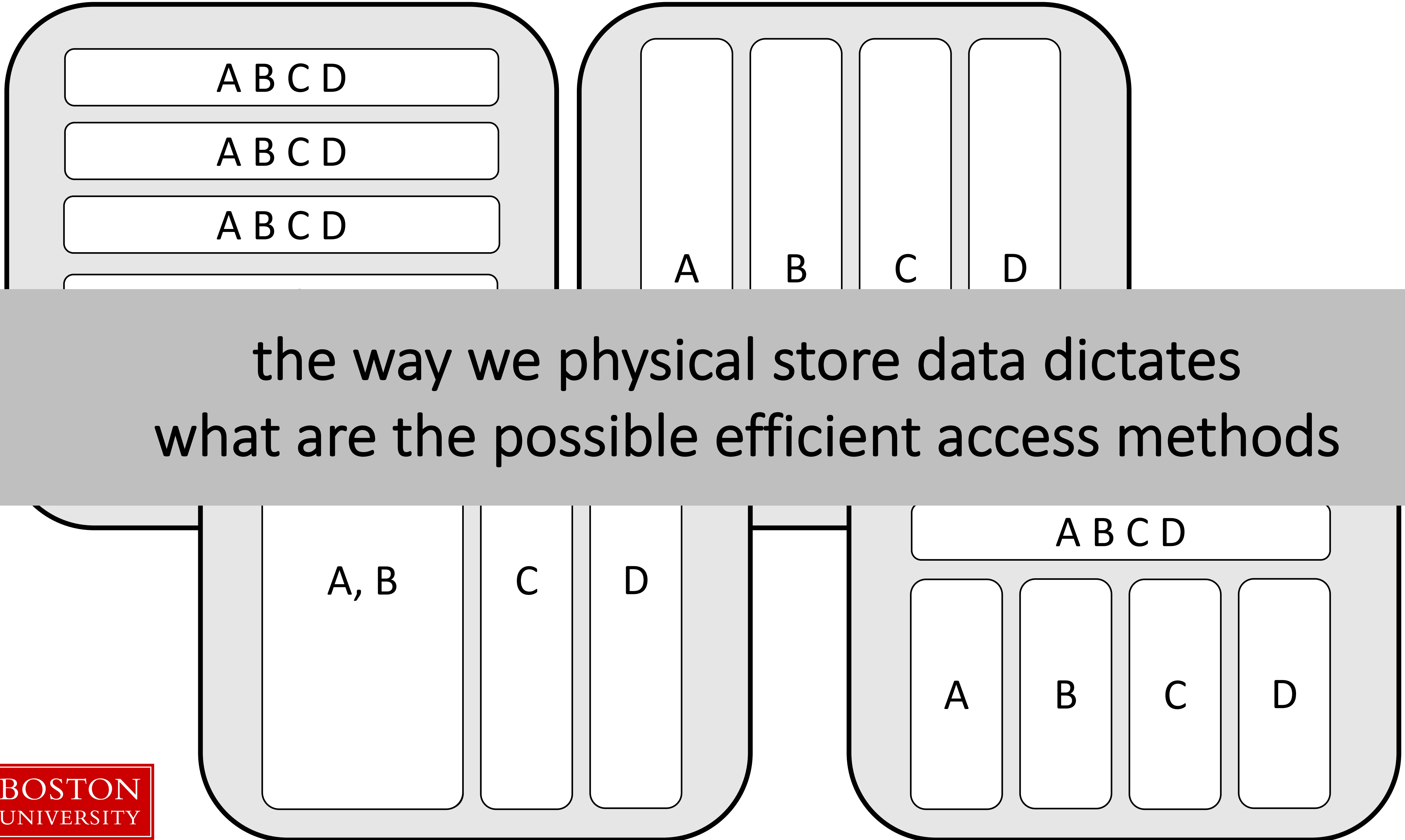
2012+: expanding on hybrid layouts

/

2001: first idea for hybrid layouts

BOSTON UNIVERSITY

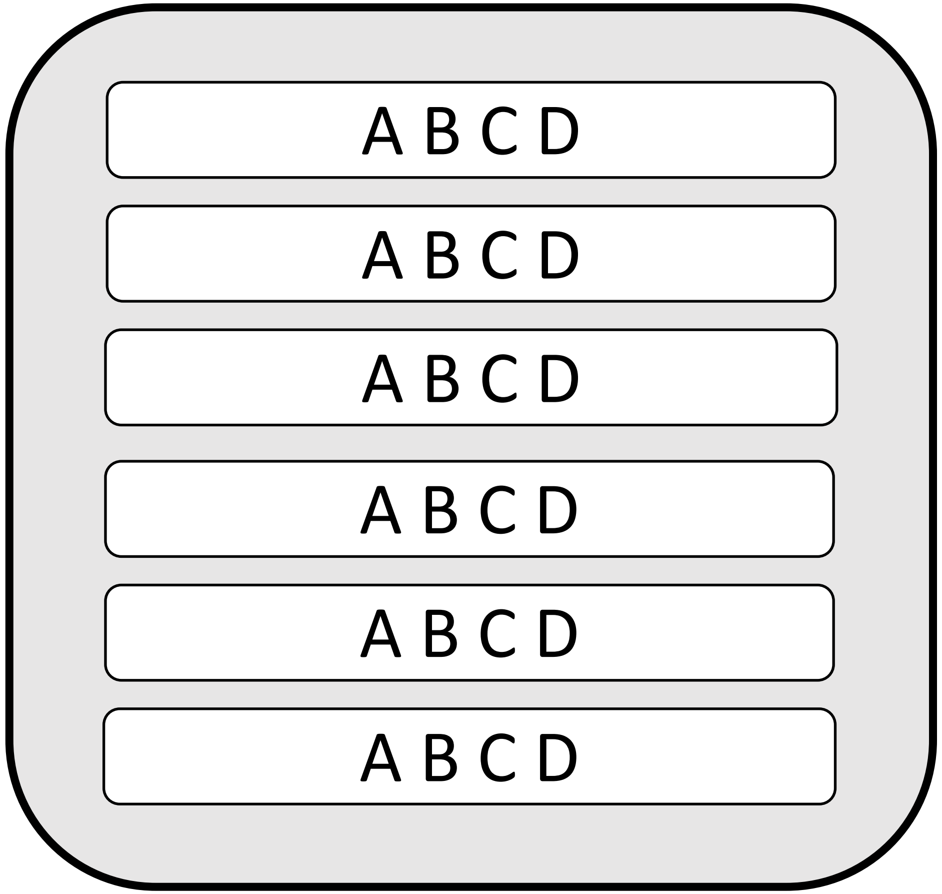

A B C D
A B C D
A B C D
A B C D
A
B
C
D
A, B
C
D
A B C D
A B C D
A B C D
A
B
C
D



the way we physical store data dictates
what are the possible efficient access methods

query evaluation

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



A B C D

A B C D

A B C D

A B C D

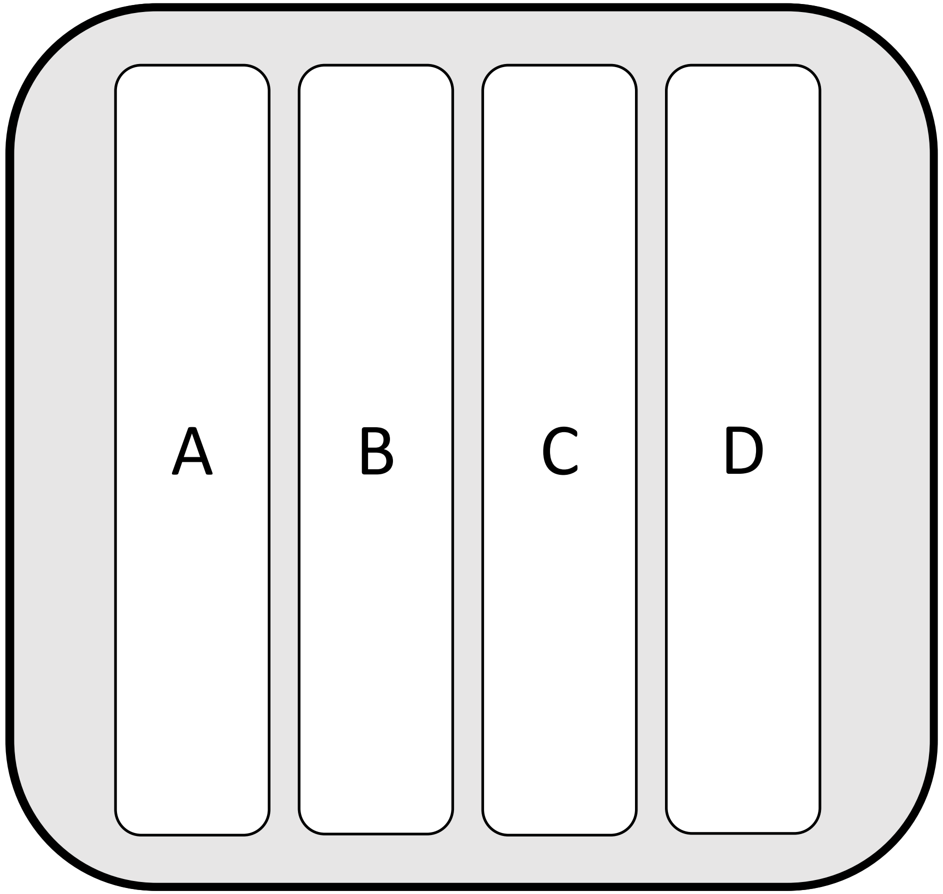
A B C D

A B C D

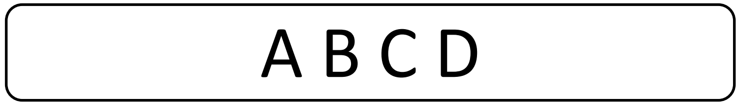
A B C D

one row at a time

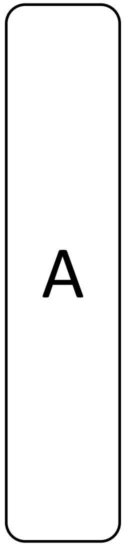
select max(B) **from** R **where** A>5 **and** C<10



tuple reconstruction/early materialization



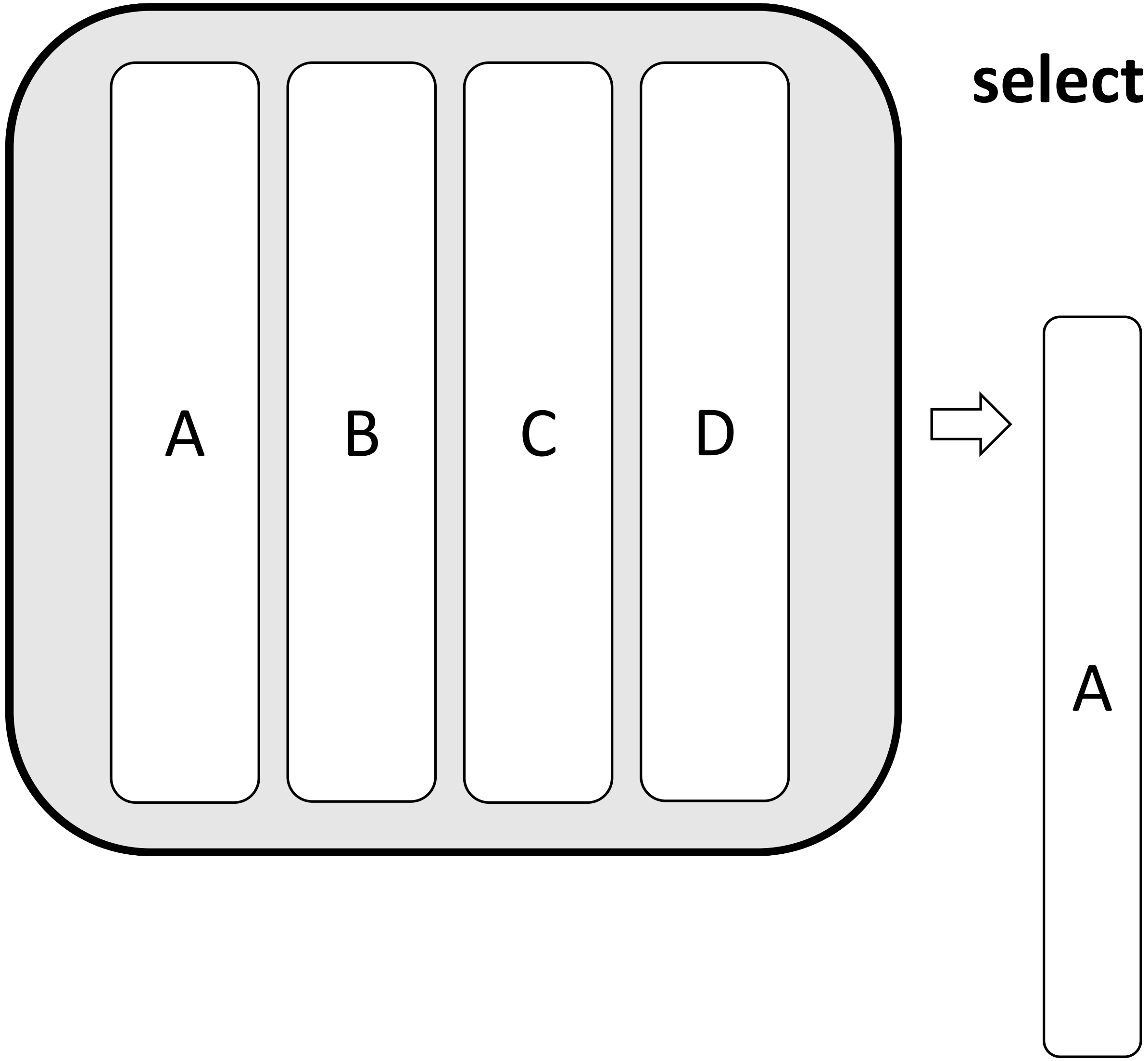
one row at a time



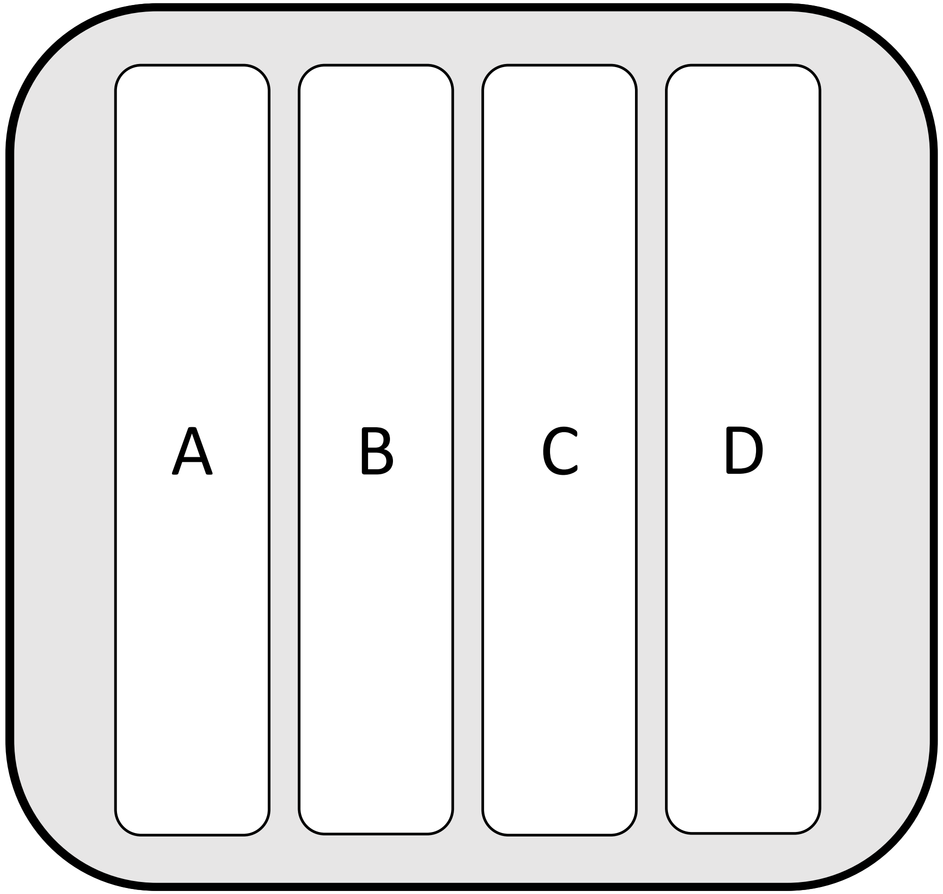
late materialization

column at a time

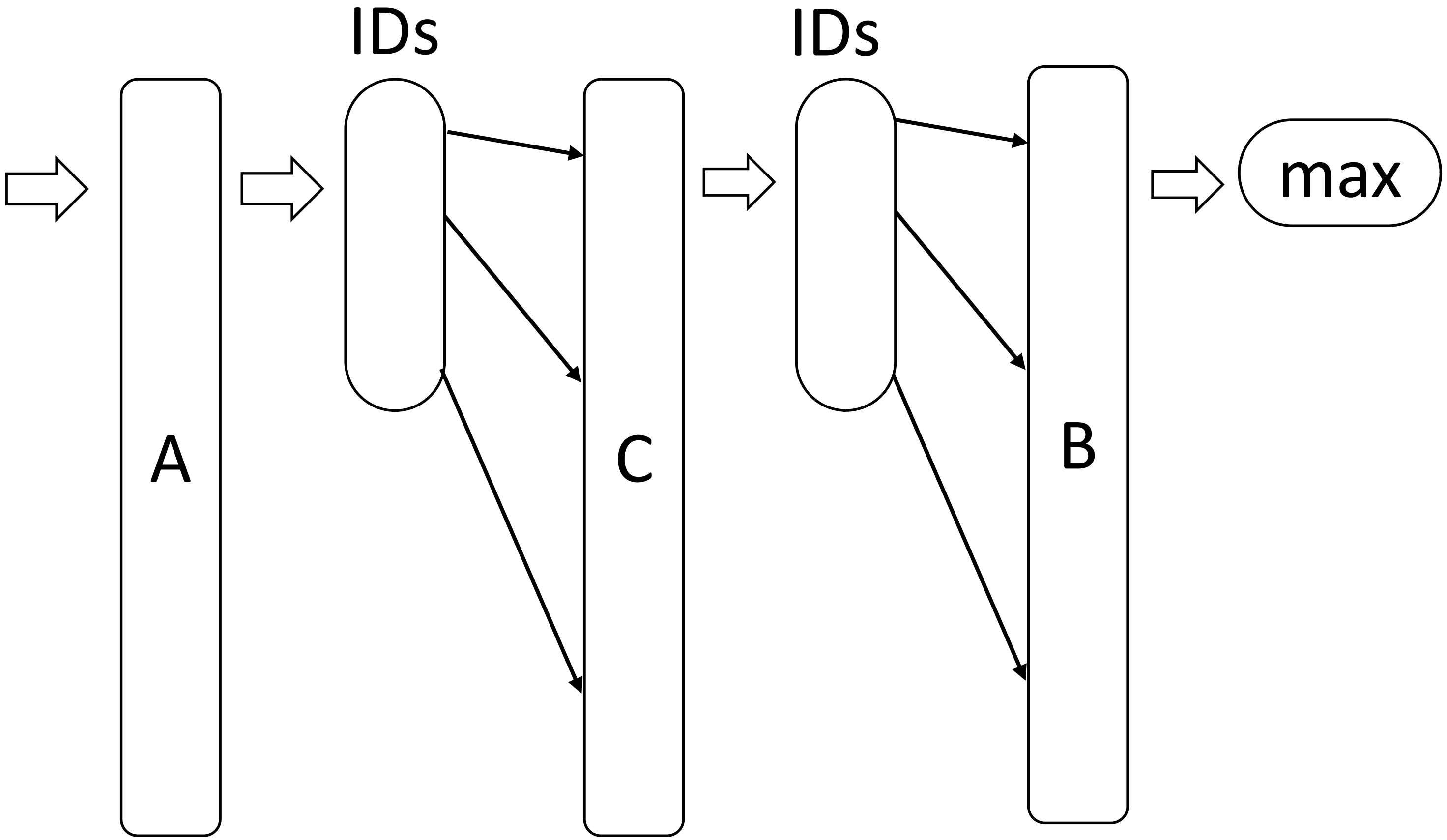
select max(B) **from** R **where** A>5 **and** C<10



```
int* input=A;
int* output; /*needs allocation*/
for (i=0; i<num_tuples; i++,input++)
    if (*input>5)
    {
        *output=i;
        output++;
    }
```



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



what is the benefit?

sequential access patterns

read only useful data

easy to code: working over fixed width and dense columns

scan

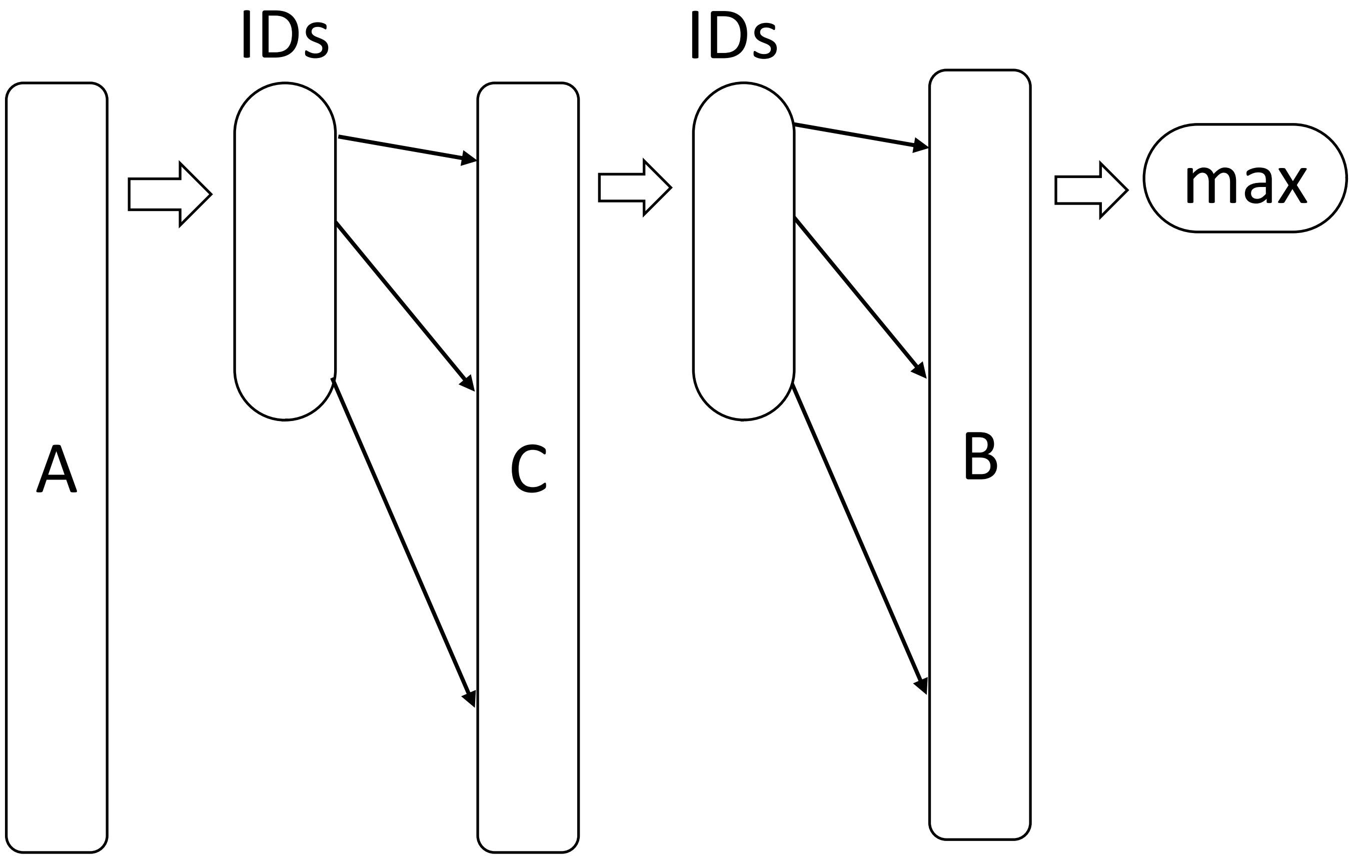
```
for (i=0,j=0; i<size; i++)
    if (column[i] qualifies)
        res[j++]=i;
```

no complex checks
no function calls
no aux metadata
easy to prefetch
as few ifs as possible

fetch

```
for (i=0,j=0; i<fetch_size; i++)
    intermediate_result[j++]=column[ids[i]];
```

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



alternatives query plans

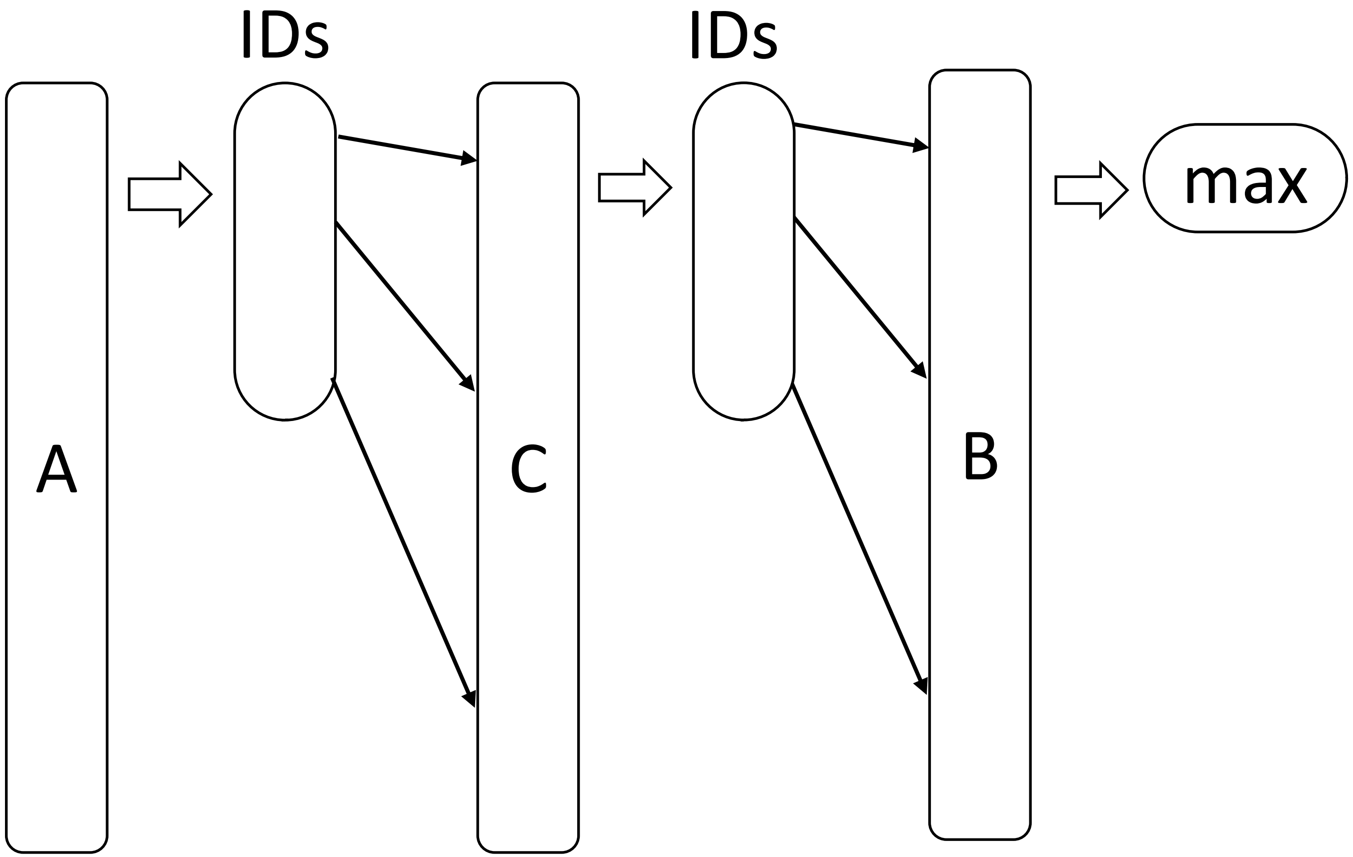
scan A & C in parallel and merge

start from C (why?)

use bit vectors (why?)

BOSTON UNIVERSITY

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



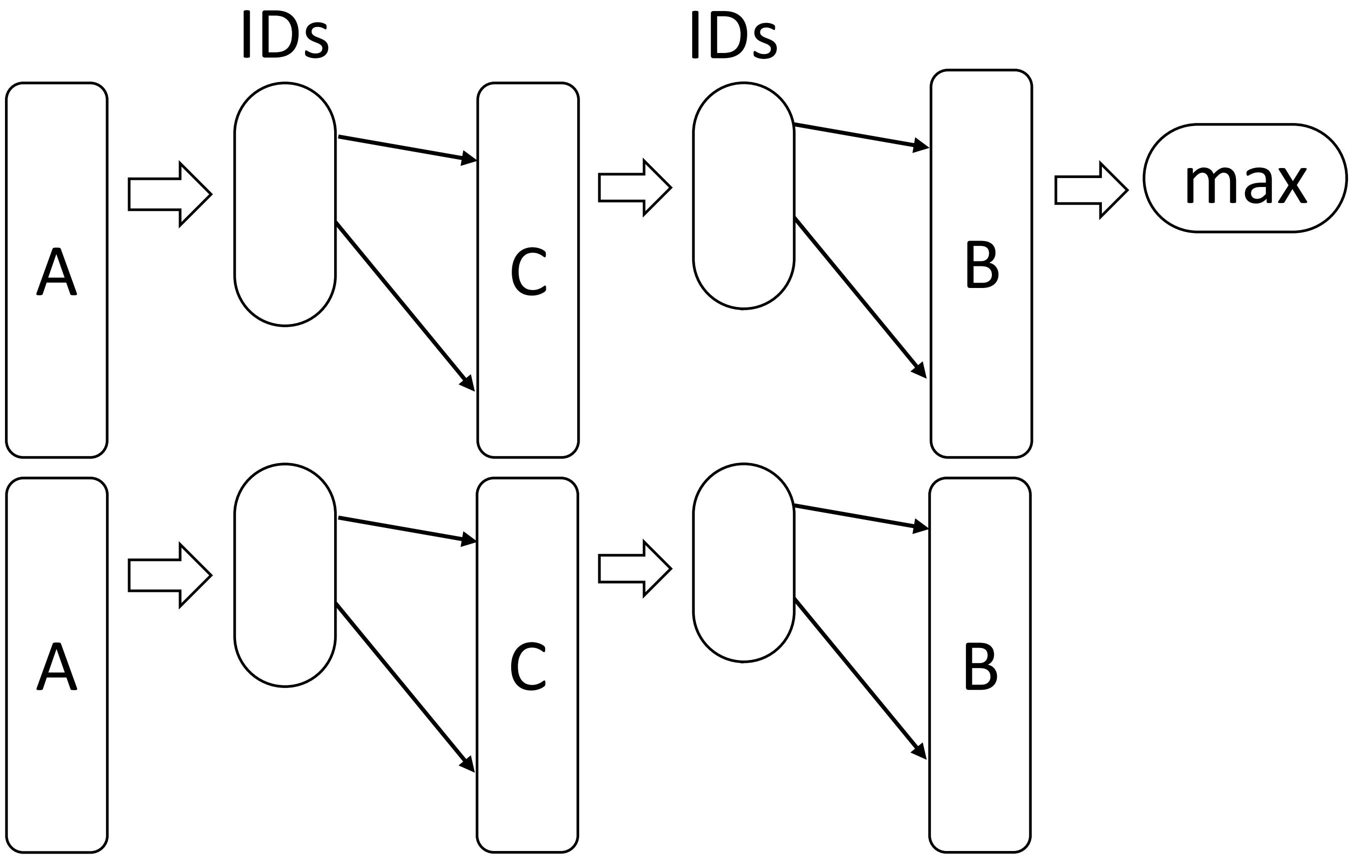
whole column?

row at a time

column at a time

block/vector at a time

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



whole column?

row at a time

column at a time

block/vector at a time

why column-stores are here now?

late materialization – no need to reconstruct tuples

read only useful data

minimize data movement across the memory hierarchy

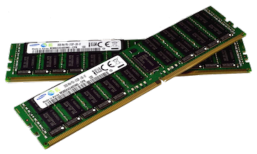
but it required a complete re-write

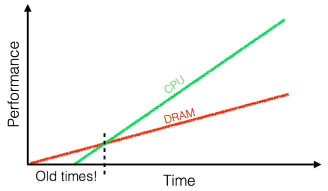
why not before?

legacy technology to catch up

more important: **analytical workloads** (as opposed to only OLTP)

new hardware: **larger memories & memory wall**





BOSTON UNIVERSITY

CS 561: Data Systems Architectures

class 3

Column-Stores Basics

Prof. Manos Athanassoulis

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