BOSTON UNIVERSITY

CS 561: Data Systems Architectures

class 3

Relational Recap & Column-Stores Basics

Prof. Manos Athanassoulis

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

what to do now?

A) read the syllabus and the website

B) register to Piazza + Gradescope

C) finish project 0 (due 2/1)

D) start working on project 1 (due 2/15)

E) register for the presentation (week 3)

F) start reading papers & prepare for tech. questions (week 3)

G) go over the class project (end of next week will be available)

H) start working on the proposal (week 3)

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
- **Data Structures for Data-Intensive Applications: Tradeoffs and Design Guidelines.**
  By M. Athanassoulis, S. Idreos, D. Shasha.
  Foundations and Trends in Databases, 2024

2) Start going over the papers

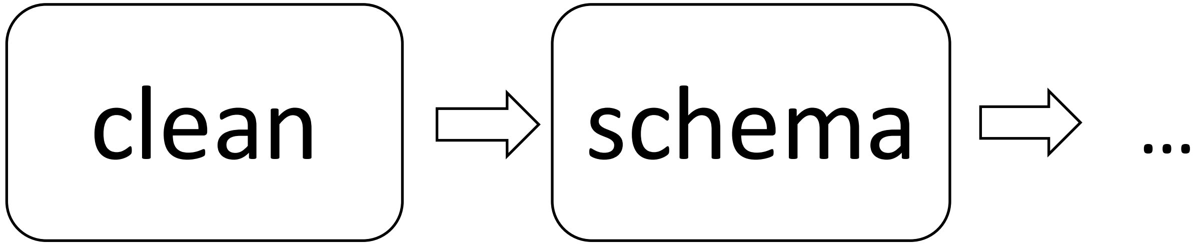
Database Design Abstraction Levels

Logical Design

Physical Design

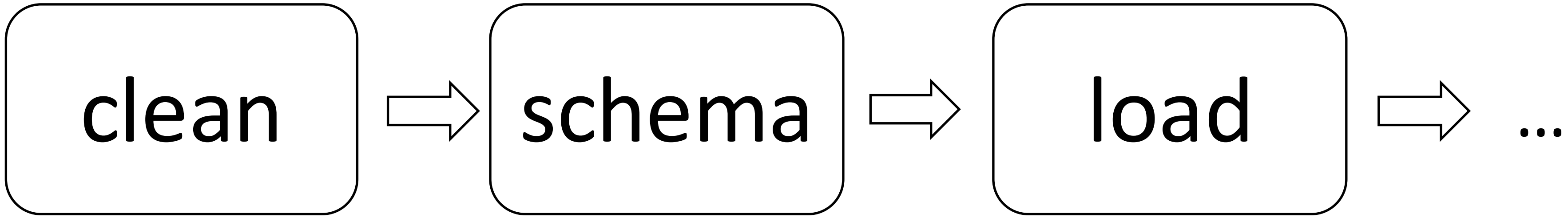
System Design

Data can be messy!



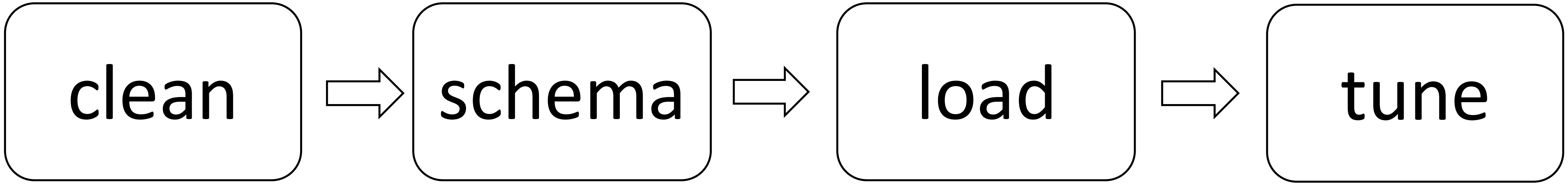
clean → schema → …

Data can be messy!



clean → schema → load → …

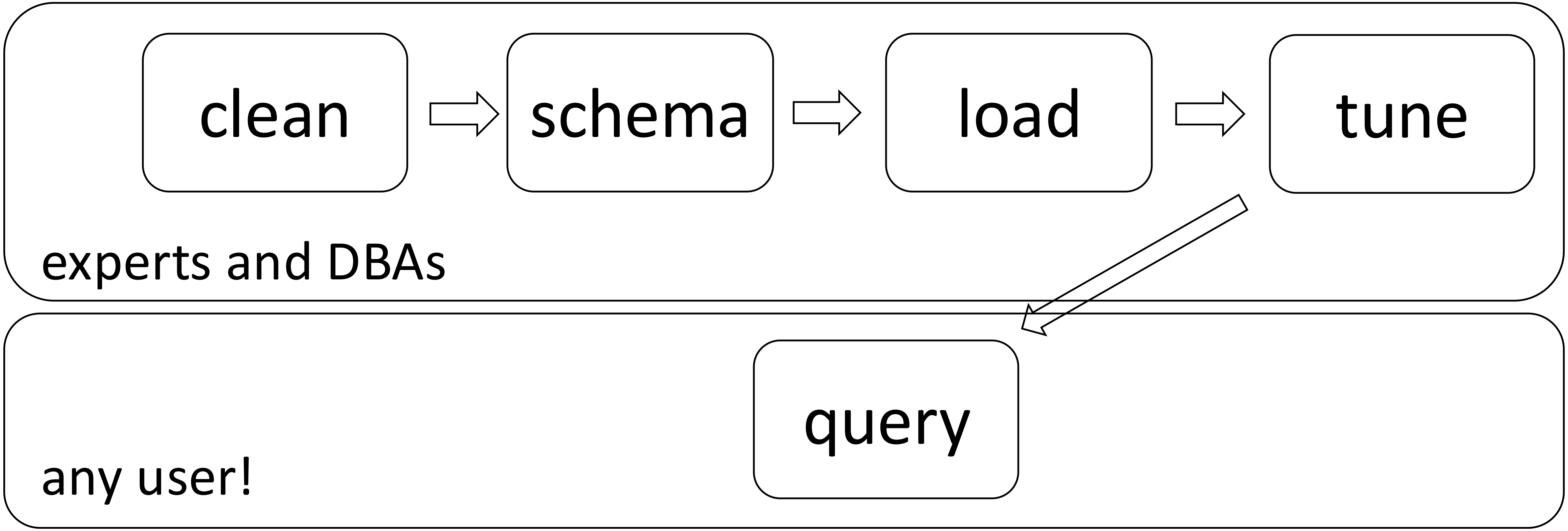
Data can be messy!



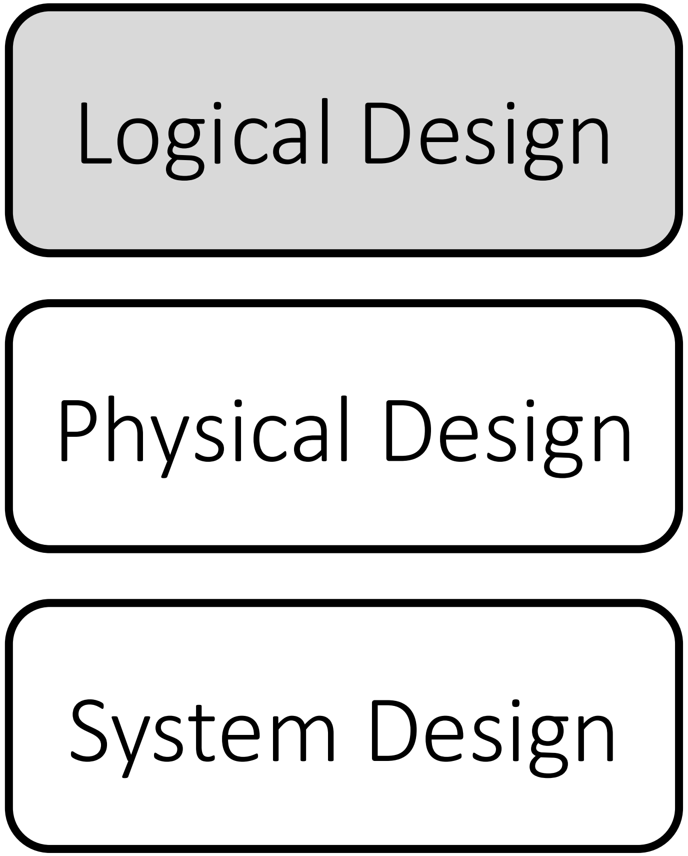
clean → schema → load → tune

what kind of indexes
size of memory buffer
how many threads to use
…

Data can be messy!



Database Design Abstraction Levels



Logical Design

Physical Design

System Design

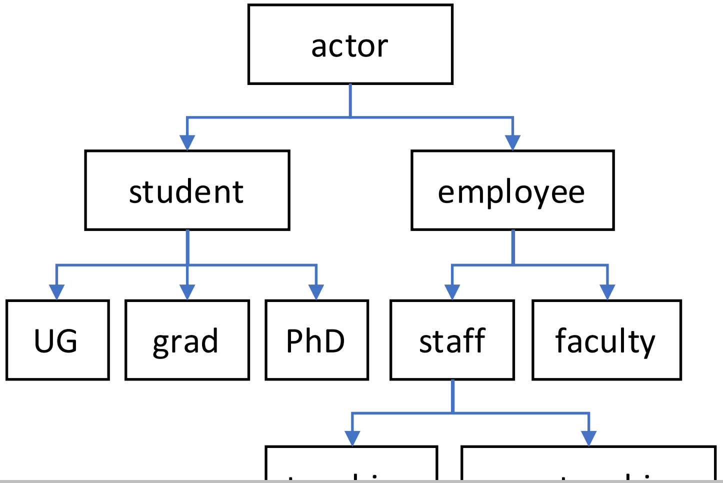
Logical design

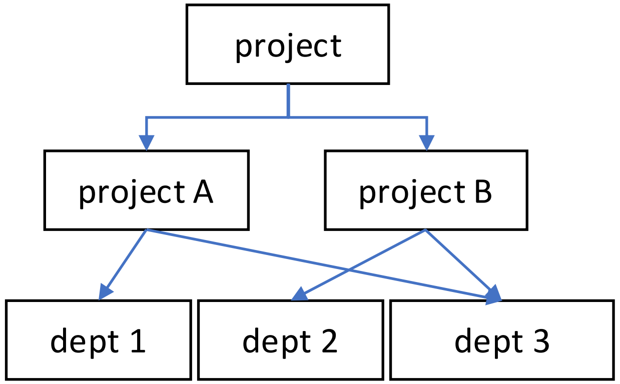
What is our data? How to model them?

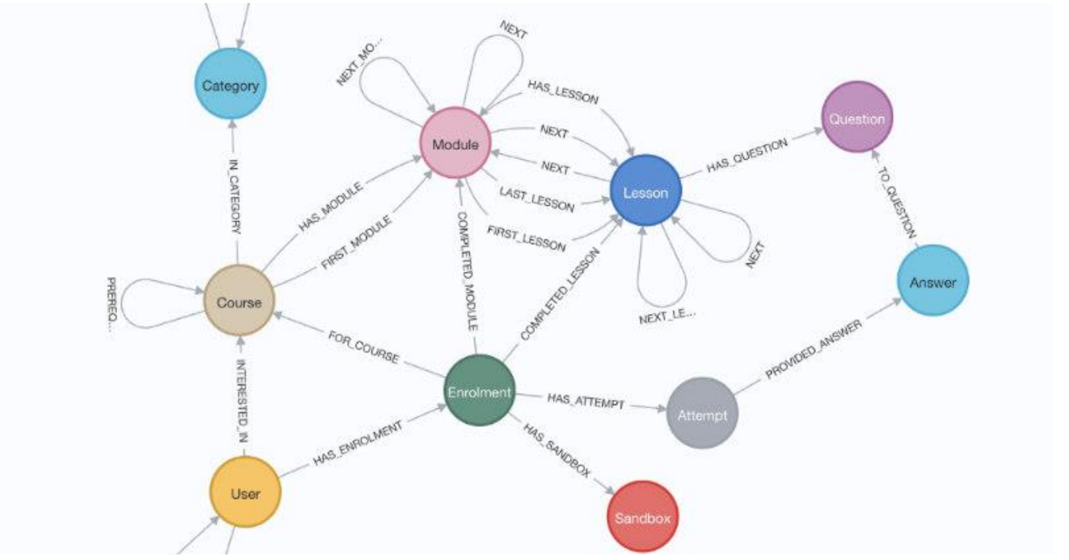
Hierarchical? Network? Object-oriented? Flat?

Logical design

What is our data? How to model them?







relational data model

key-value data model

Logical design

What is our data? How to model them?

Hierarchical? Network? Object-oriented? Flat?

Relational & Key-value

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*

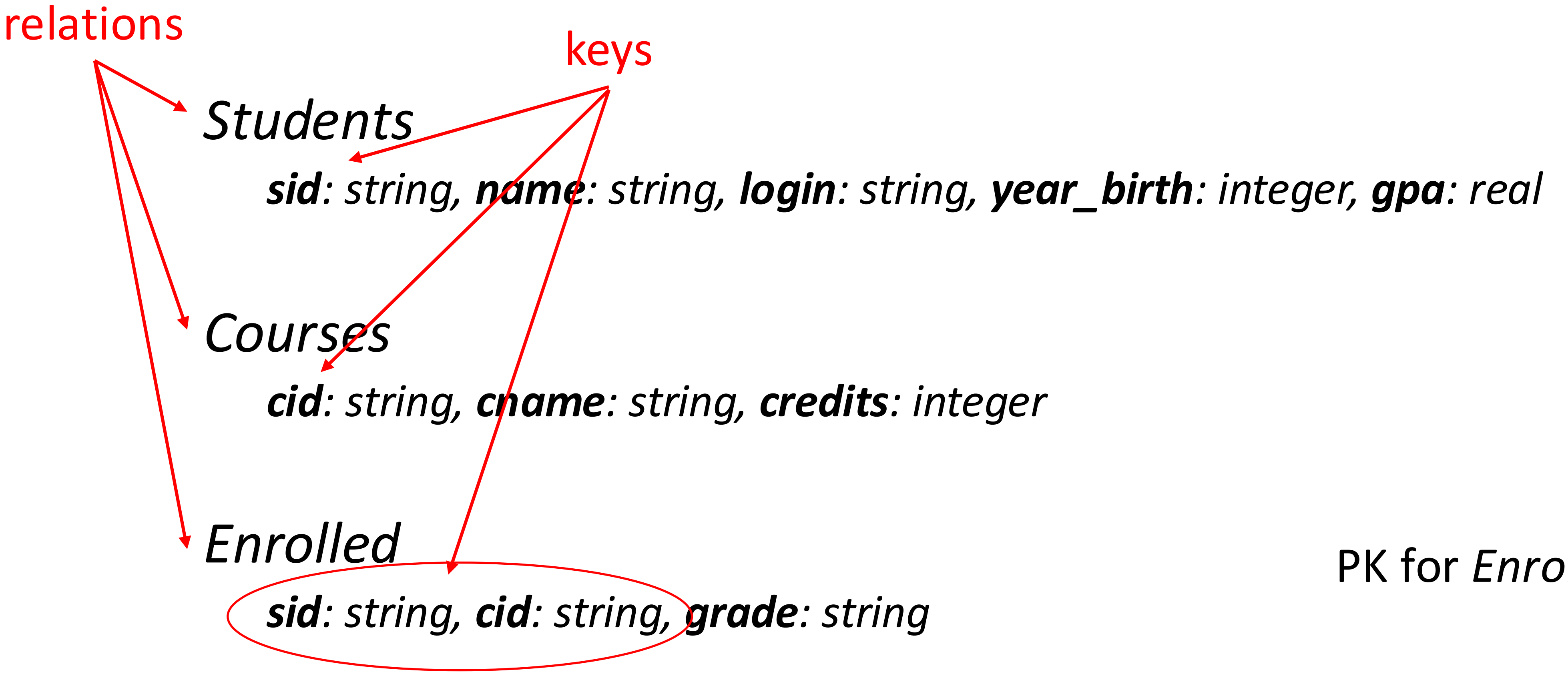
Enrolled

sid: *string*, **_cid_**: *string*, **_grade_**: *string*

attributes for *Enrolled* ?

BOSTON UNIVERSITY

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*

PK for *Enrolled* ?

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

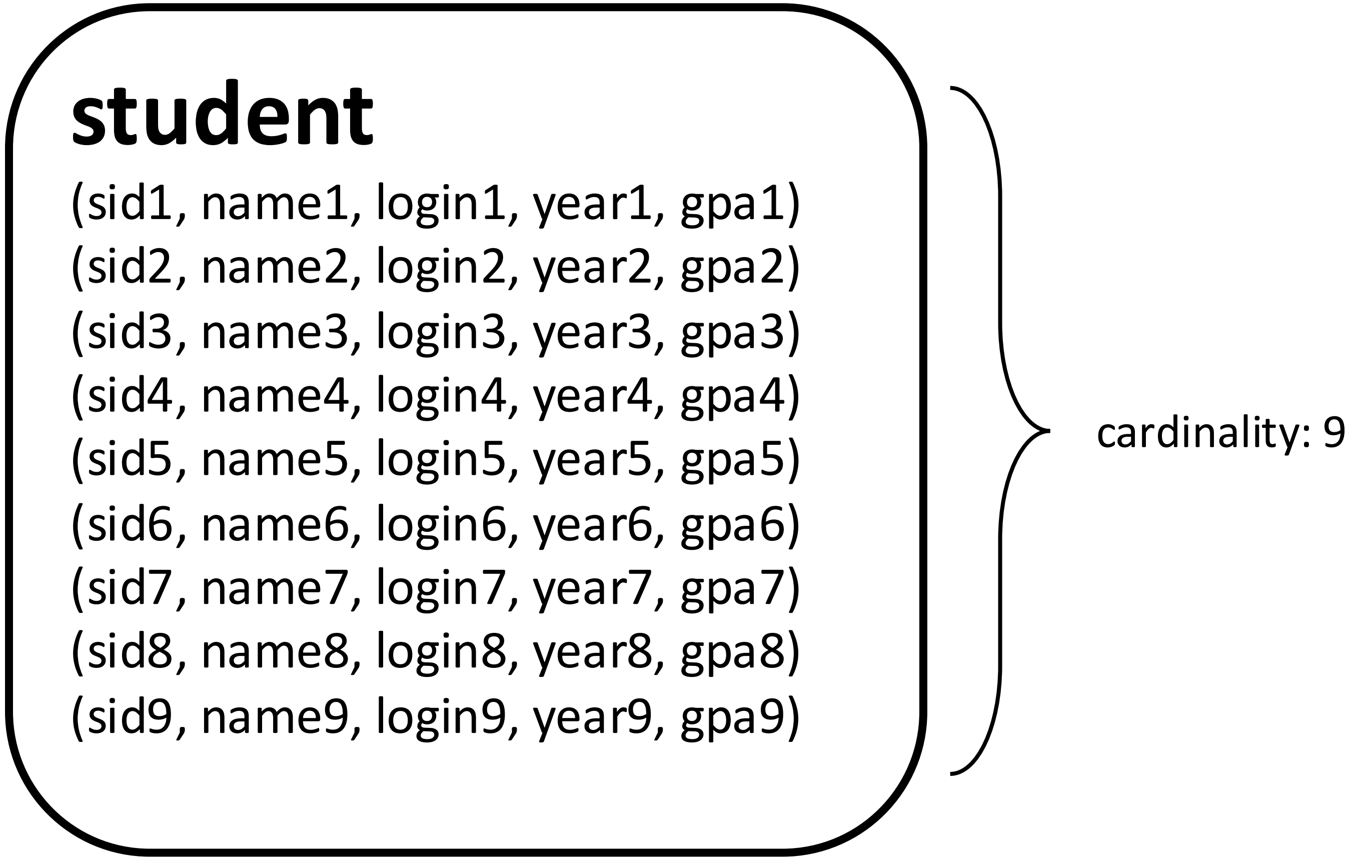
bring me the names of all students

__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)

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)

cardinality: 9

what if a student does not have a login ID 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)

cardinality: 9

what if a student does not have a login ID yet?

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

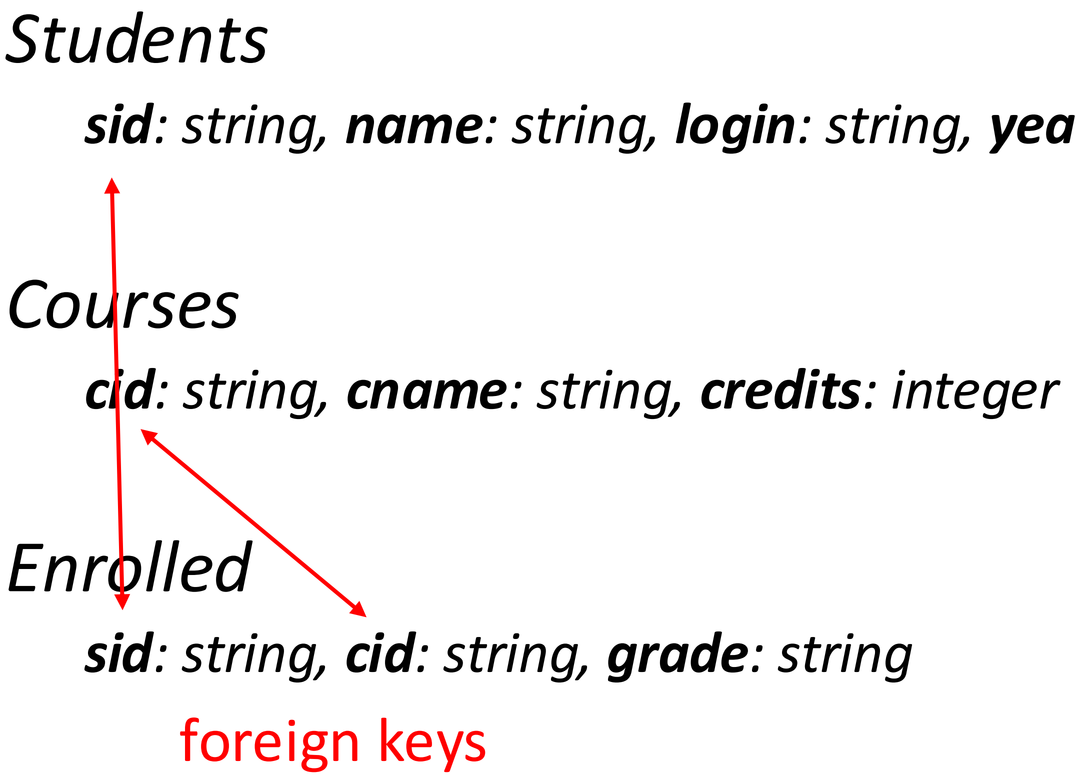
***cid**: string, **cname**: string, **credits**: integer*

Enrolled

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

Relational Model and SQL

how to show all enrollments in DSA?



Students

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

Courses

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

Enrolled

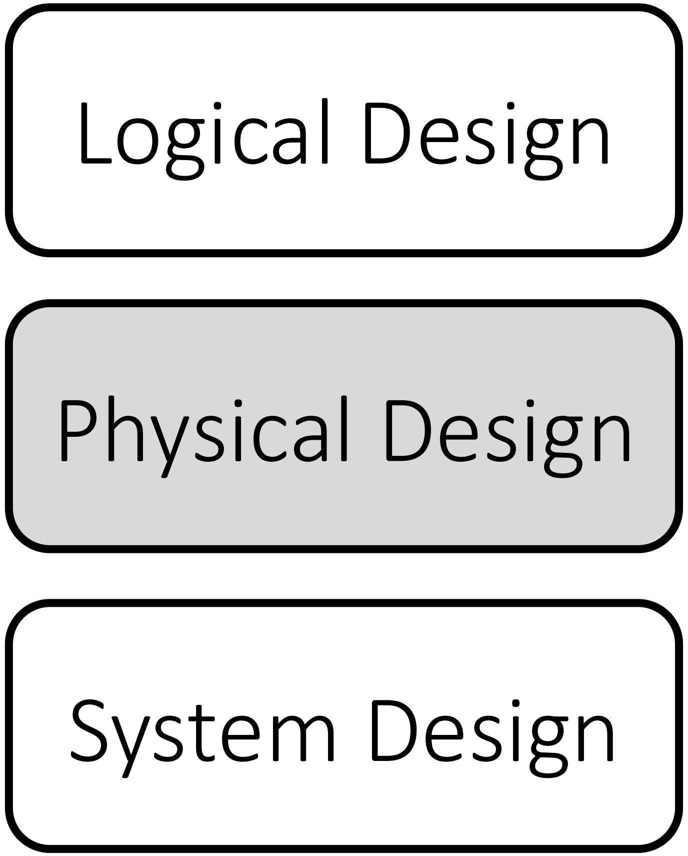
***sid**: string, **cid**: string, **grade**: string*

foreign keys

using foreign keys we can join information of all three tables

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

Database Design Abstraction Levels



Logical Design

Physical Design

System Design

Physical Design

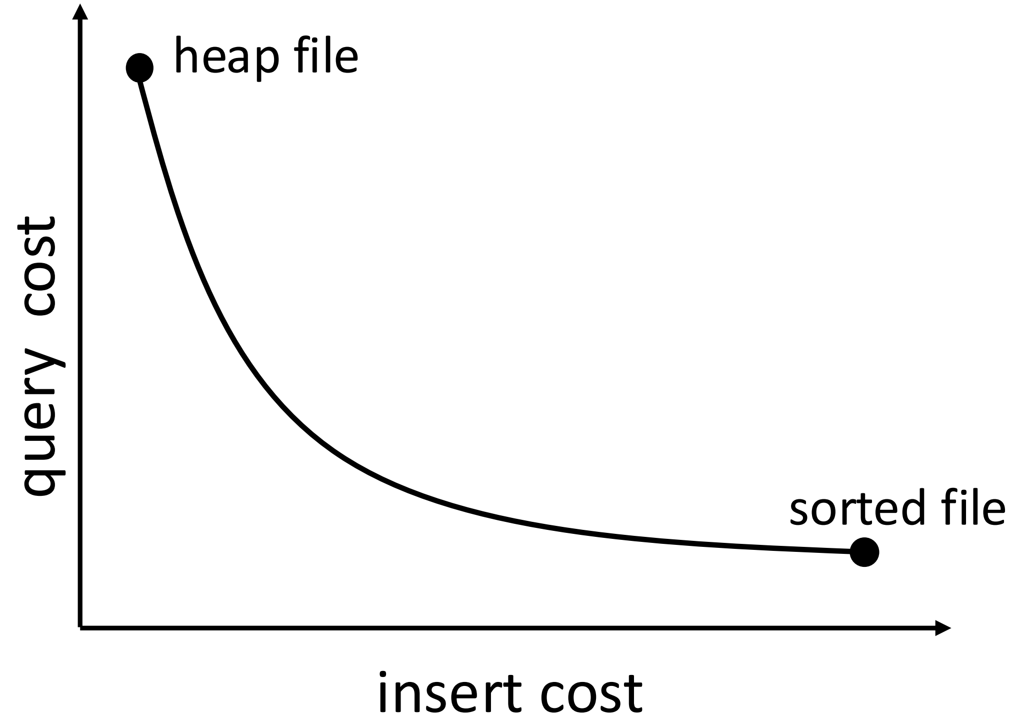
File Organization

heap files

sorted files

clustered files

more …



Physical Design

File Organization

heap files

sorted files

clustered files

more …

Indexes

should I build an index?

on which attributes/tables?

what index structure?

B-Tree

Trie

Hash

Bitmap

Zonemap

Physical Design

trie?

File Organization

heap files

sorted files

clustered files

more …

Indexes

should I build an index?

on which attributes/tables?

what index structure?

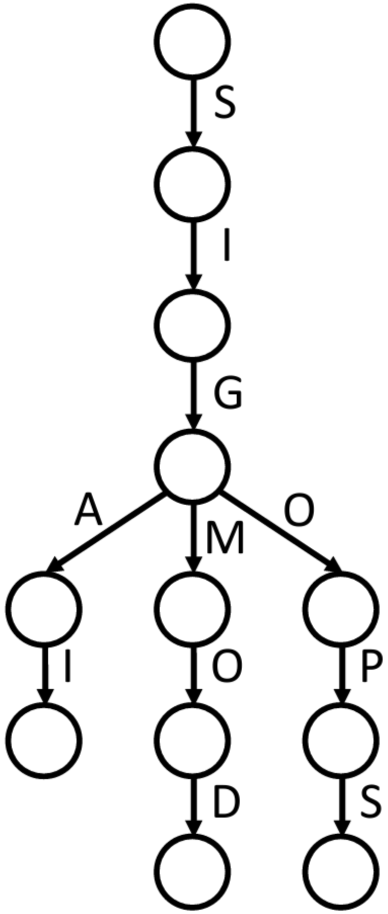
B-Tree

Trie

Hash

Bitmap

Zonemap



k-ary prefix tree

Physical Design

bitmap?

File Organization

heap files

sorted files

clustered files

more …

Indexes

should I build an index?

on which attributes/tables?

what index structure?

B-Tree

Trie

Hash

Bitmap

Zonemap

rid	Column
1	30
2	20
3	30
4	10
5	20
6	10
7	30
8	20

data

rid	10	20	30
1	0	0	1
2	0	1	0
3	0	0	1
4	1	0	0
5	0	1	0
6	1	0	0
7	0	0	1
8	0	1	0

bitmap

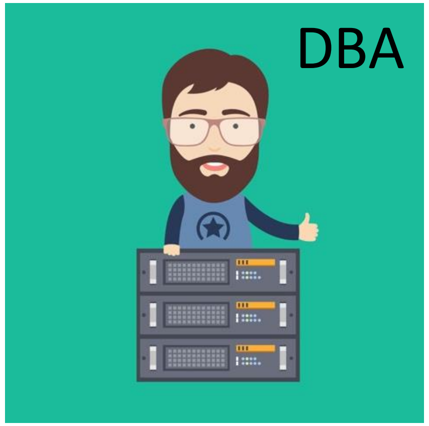
works great for columns with few distinct values

Data systems are declarative!

ask ***what*** you want

data system

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



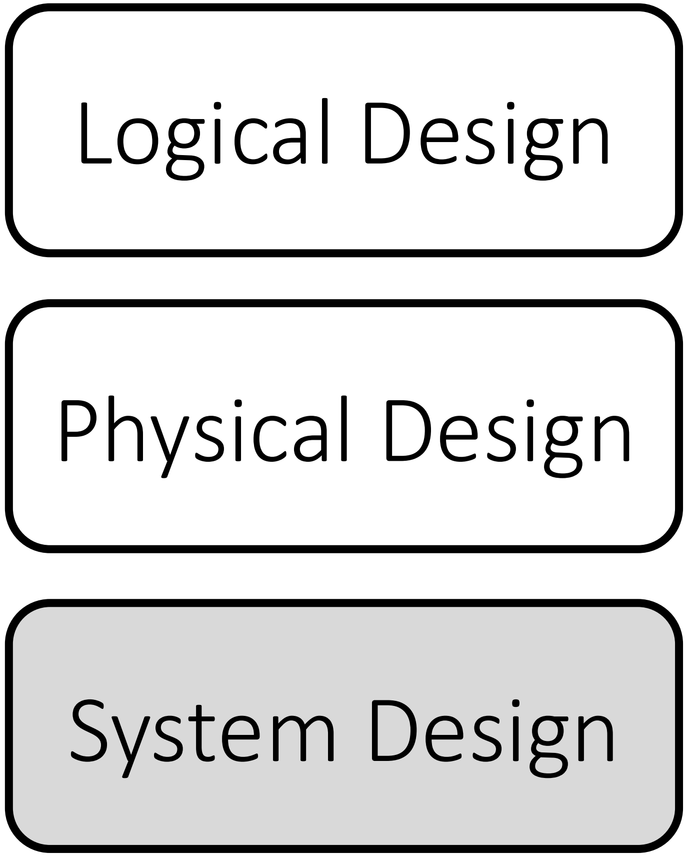
design decisions, physical design
indexing, tuning knobs

research to automate!

adaptivity

autotuning

Database Design Abstraction Levels



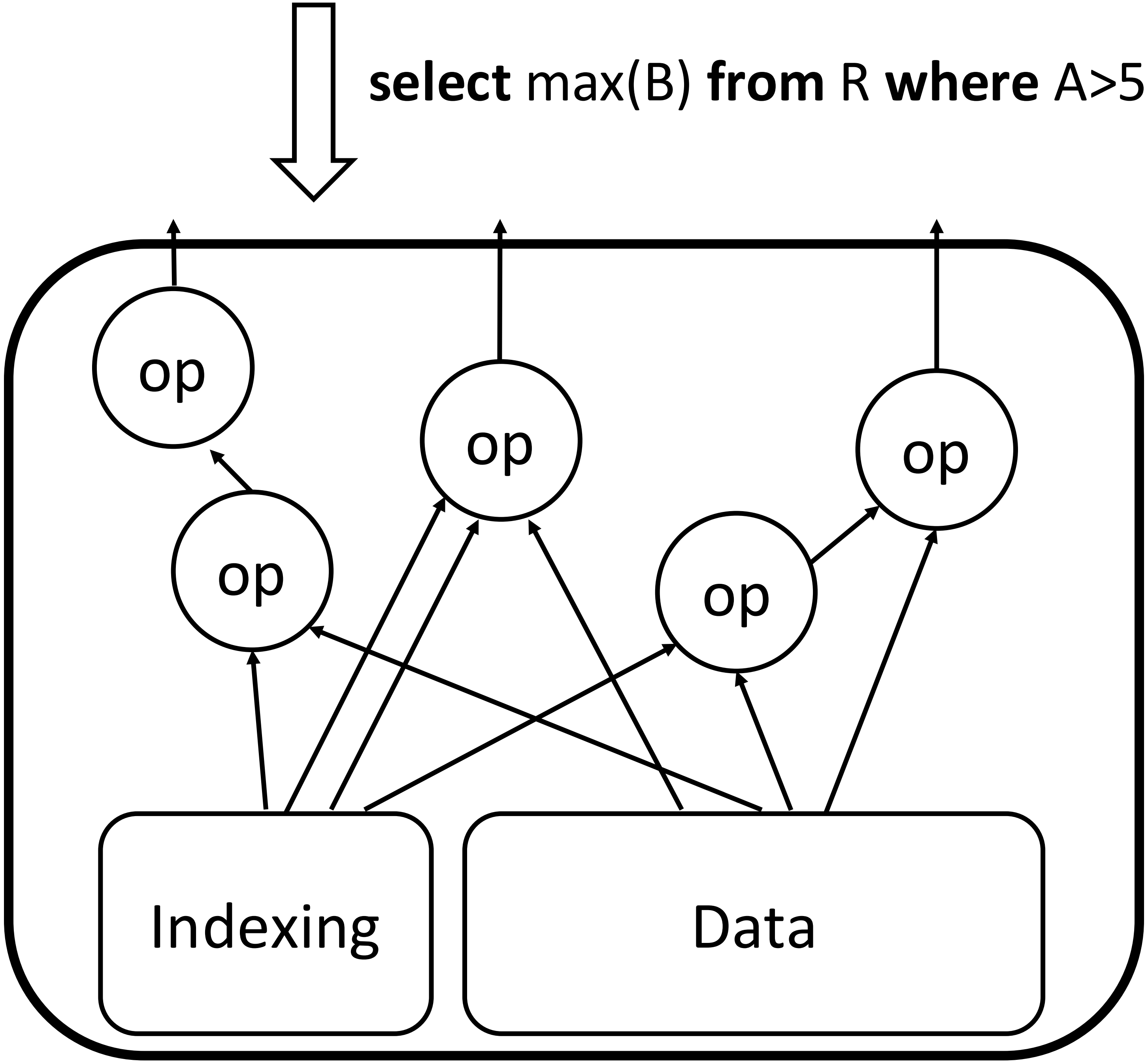
Logical Design

Physical Design

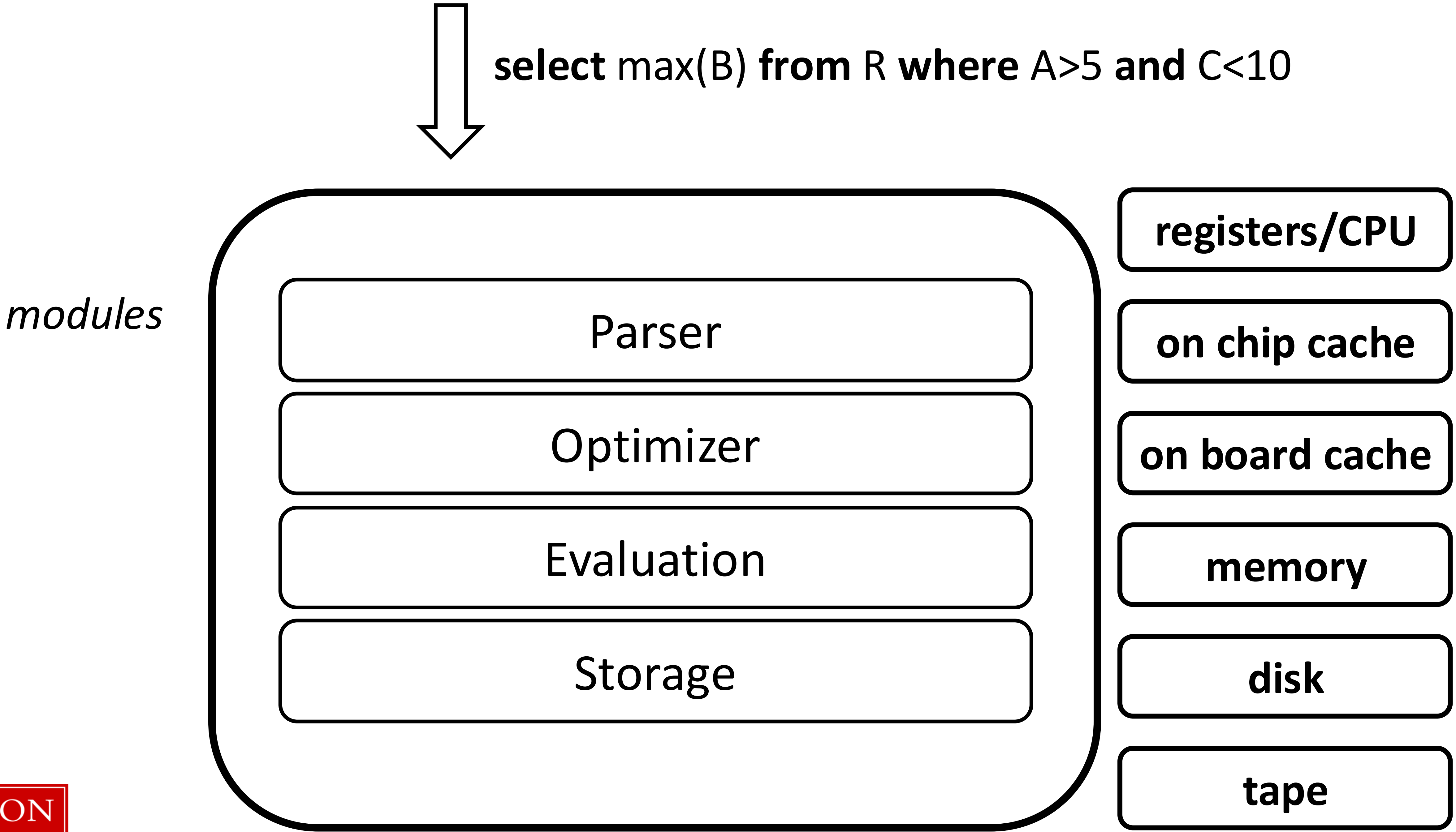
System Design

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

algorithms and operators



BOSTON UNIVERSITY



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

modules

Parser

Optimizer

Evaluation

Storage

registers/CPU

on chip cache

on board cache

memory

disk

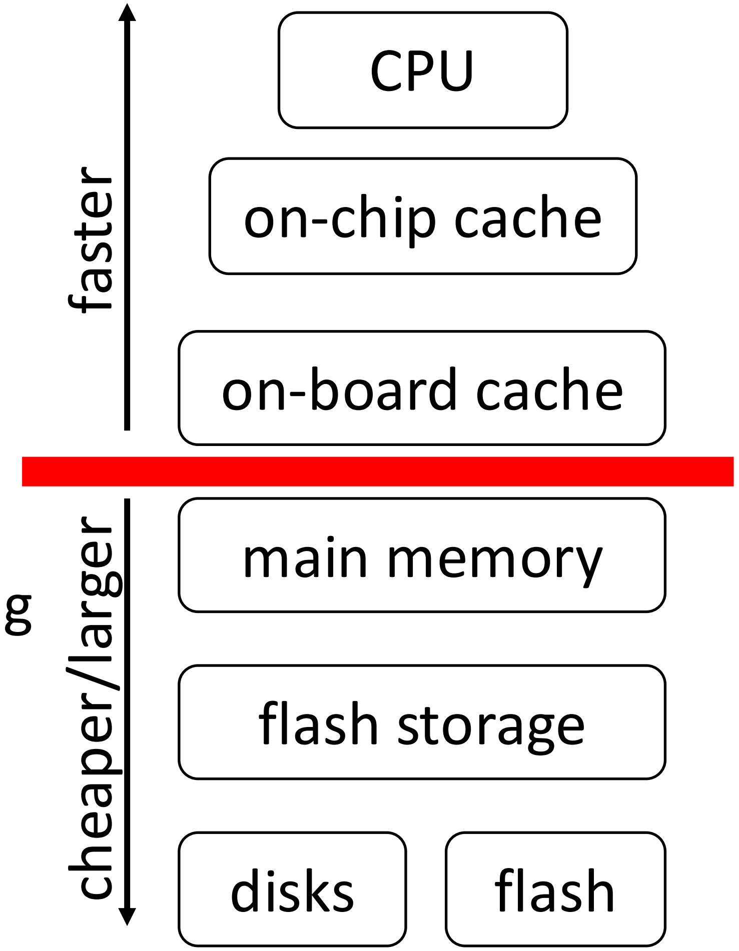
tape

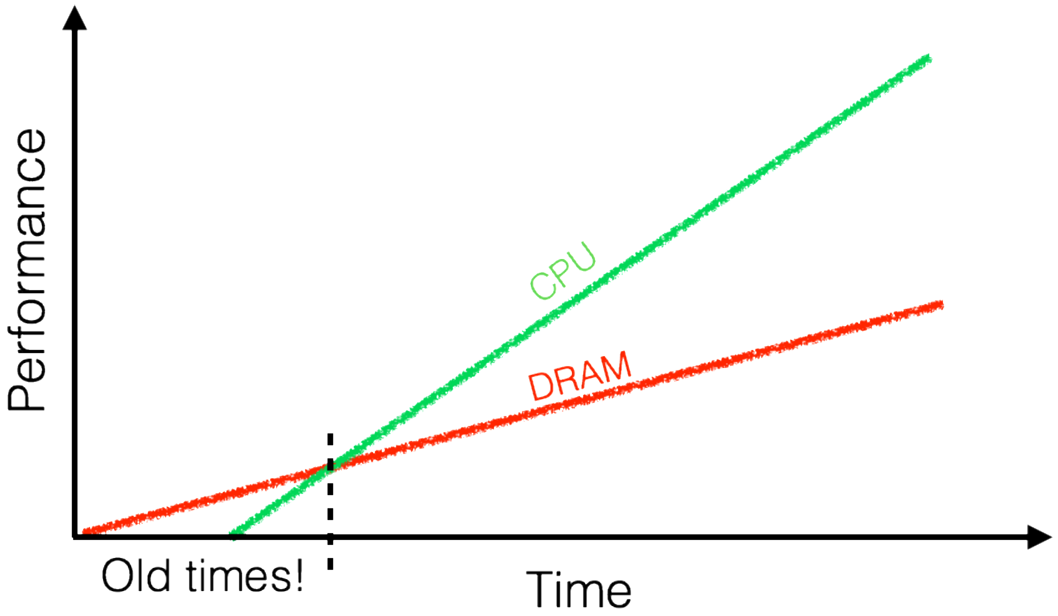
	registers/CPU	my head ~0
2x	on chip cache	this room 1min
10x	on board cache	this building 10min
100x	memory	Washington, DC 5 hours
10^6x	disk	Pluto 2 years
10^9x	tape	Andromeda 2000 years

memory wall

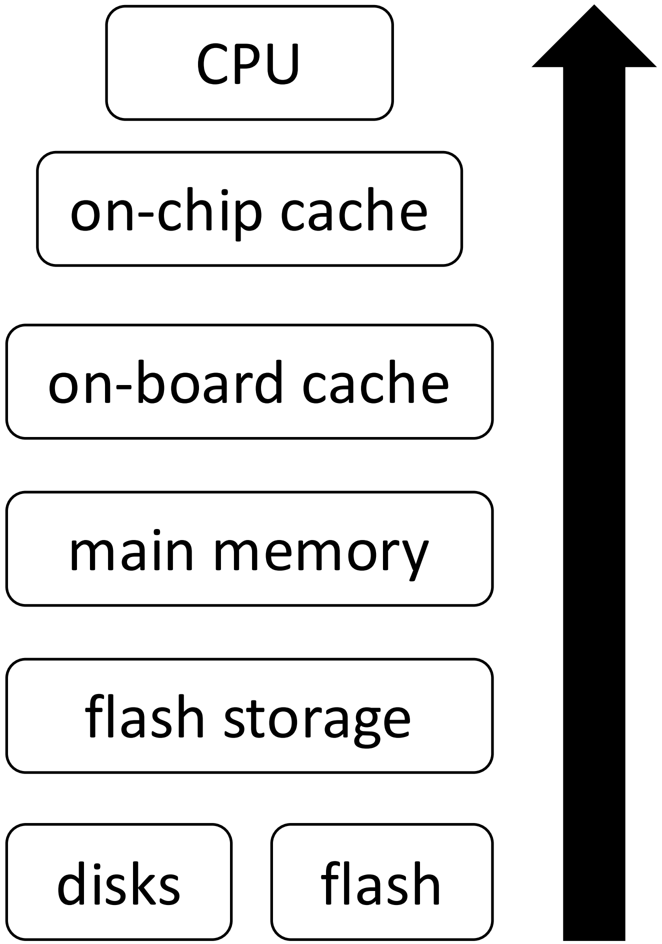
cache miss: looking for something that is not in the cache

memory miss: looking for something that is not in memory





data movement & page-based access



CPU

on-chip cache

on-board cache

main memory

flash storage

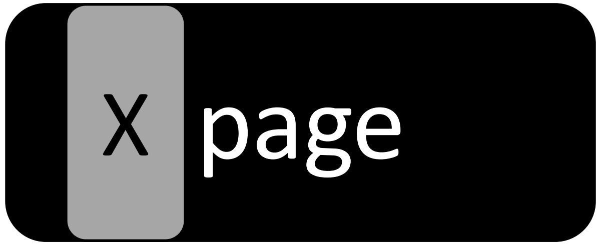
disks

flash

data go through
all necessary levels

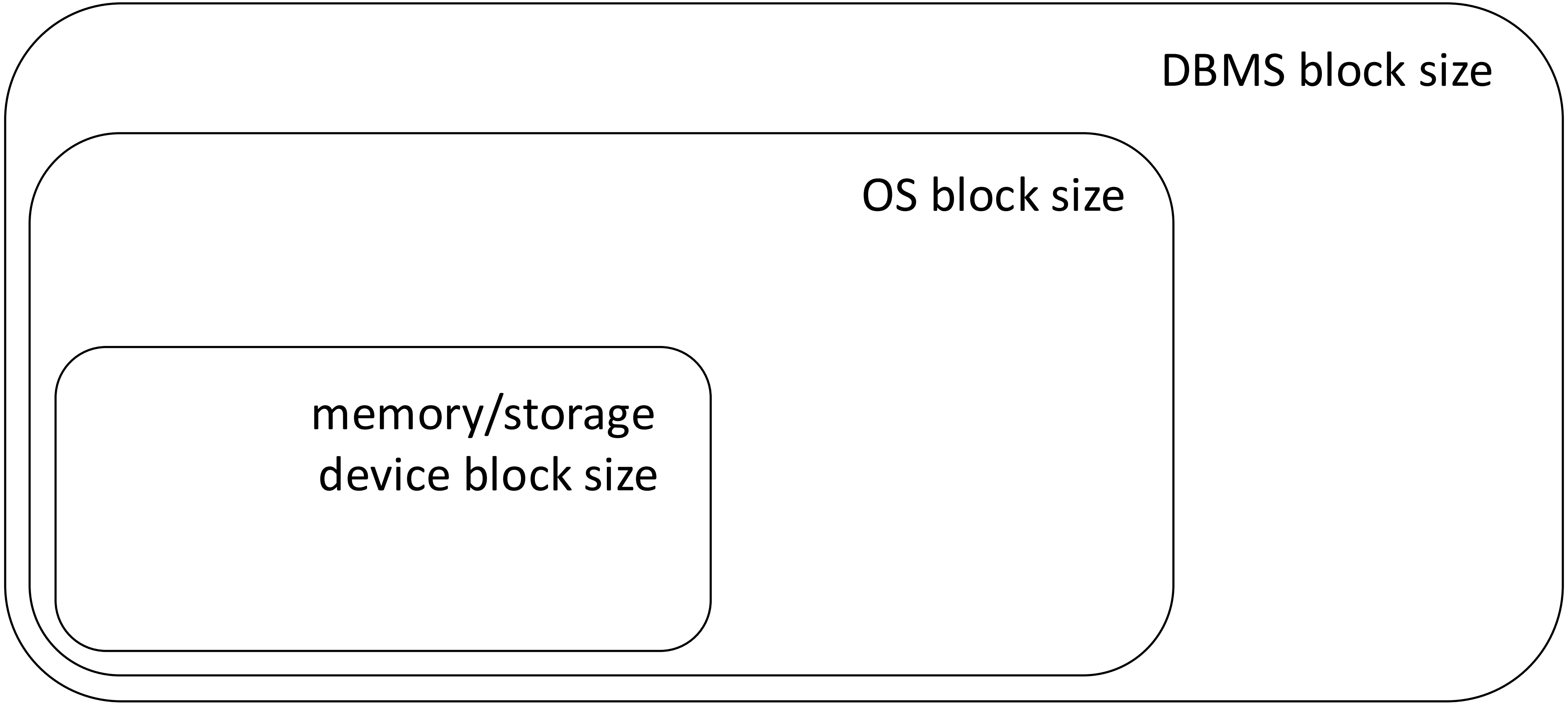
also read
unnecessary data

need to read only X
read the whole page



X page

access granularity



file system and DBMS “pages”

understanding data placement

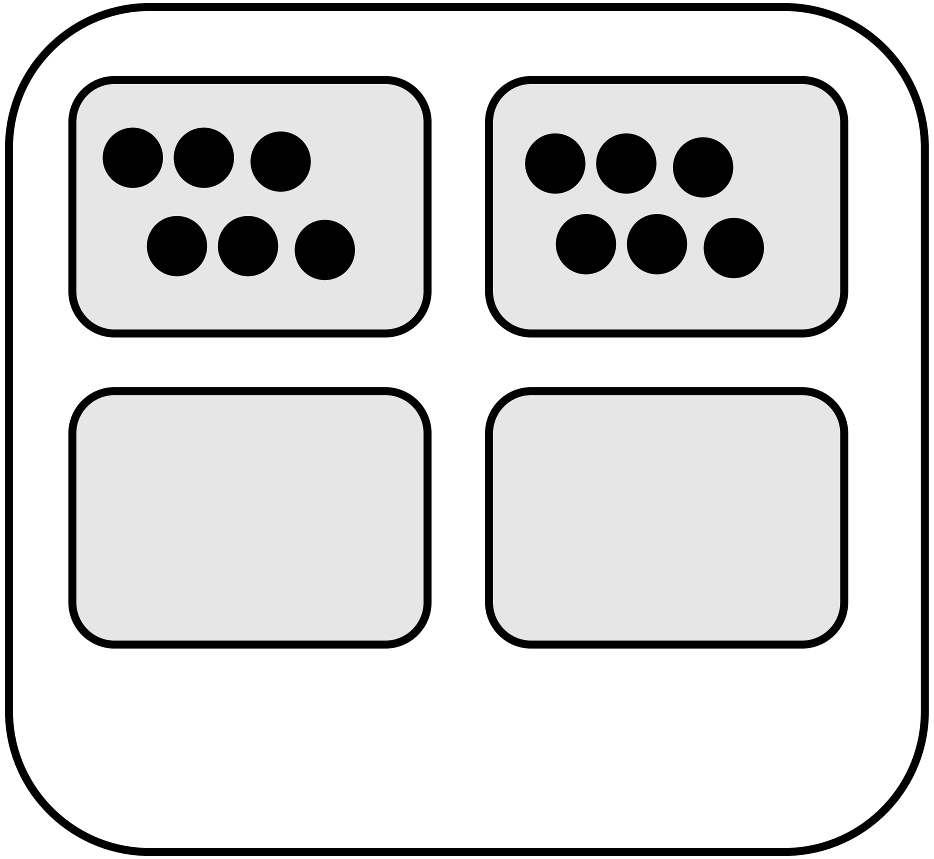
data storage

how to physically place data?

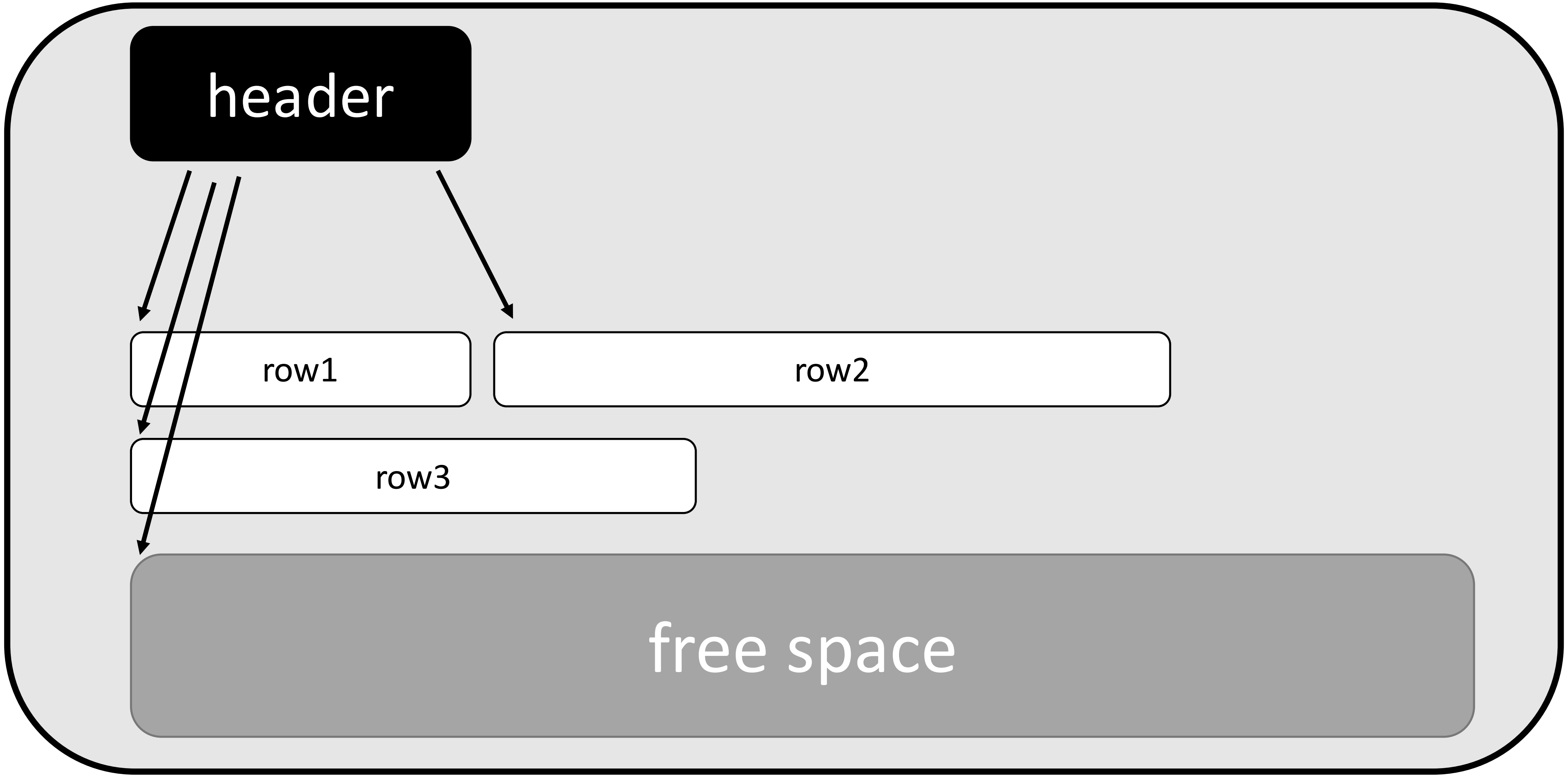
*Student (**sid**: string, **name**: string, **login**: string, **year_birth**: integer, **gpa**: real)*

student

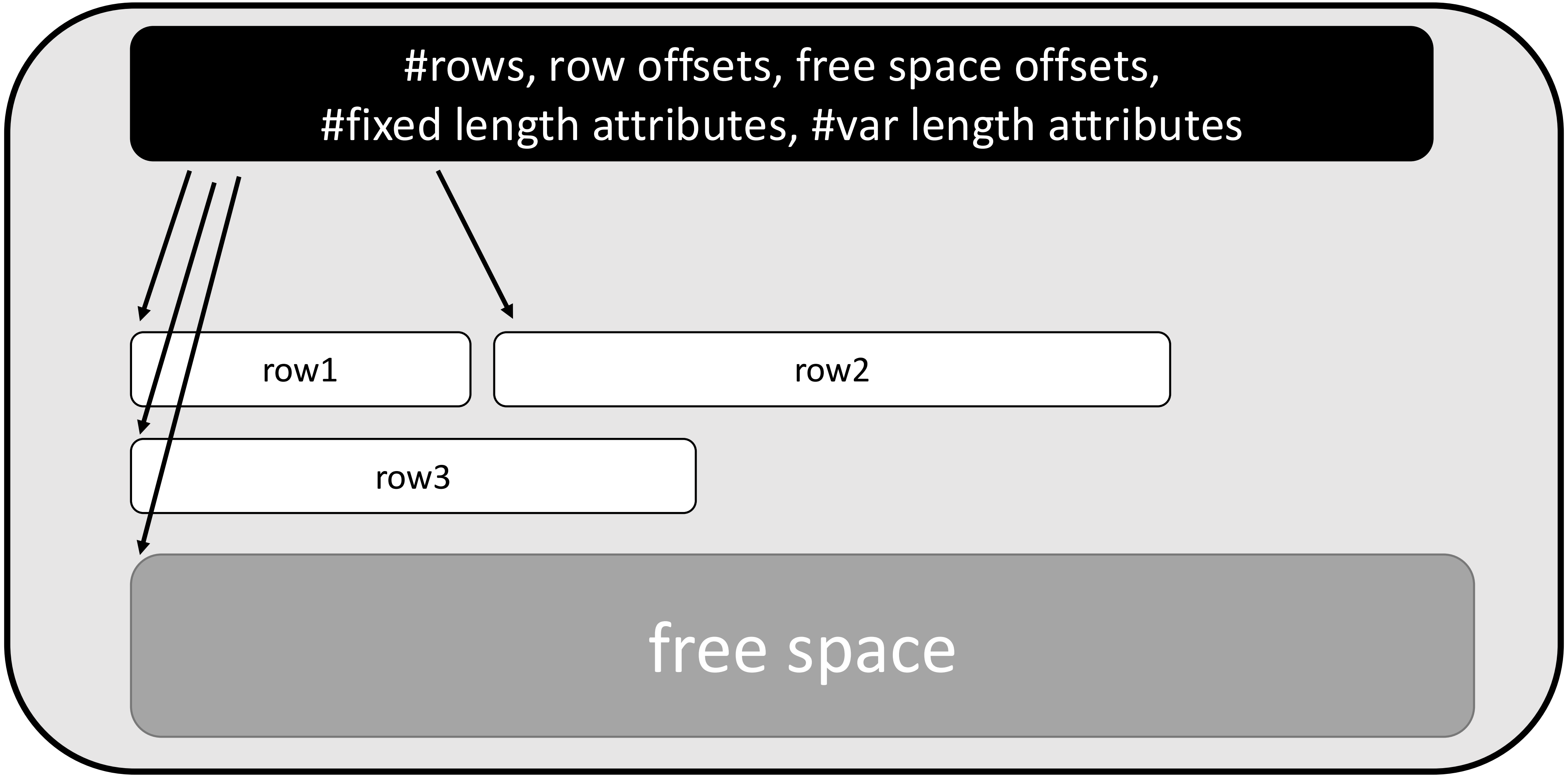
(sid1, name1, login1, year1, gpa1)
(sid2, name2, login2, year2, gpa2)
(sid3, name3, login3, year3, gpa3)
(sid4, name4, login4, year4, gpa4)
(sid5, name5, login5, year5, gpa5)
(sid6, name6, login6, year6, gpa6)
(sid7, name7, login7, year7, gpa7)
(sid8, name8, login8, year8, gpa8)
(sid9, name9, login9,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



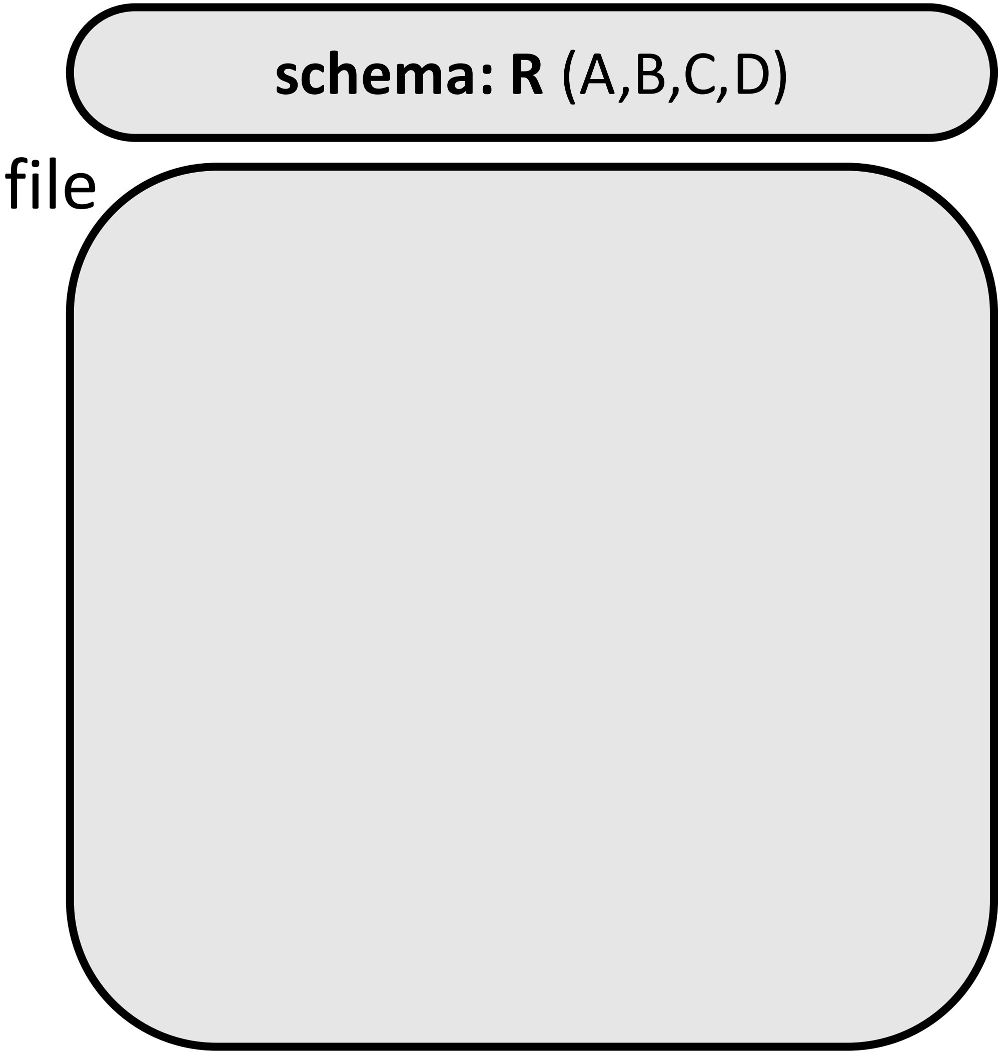
slotted page



slotted page

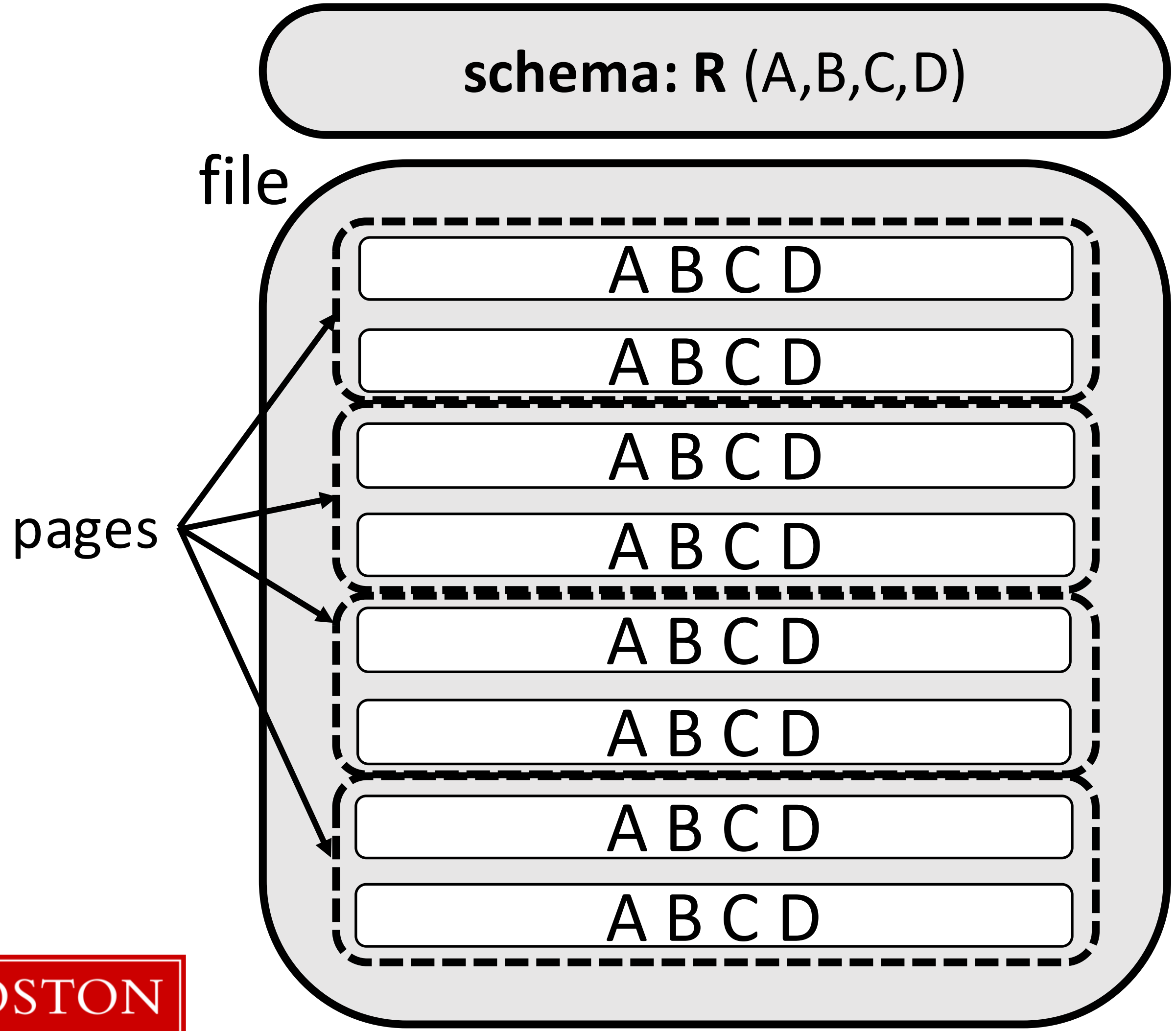

#rows, row offsets, free space offsets,
#fixed length attributes, #var length attributes
row1
row2
row3
free space

querying over slotted pages



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

querying over slotted pages



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

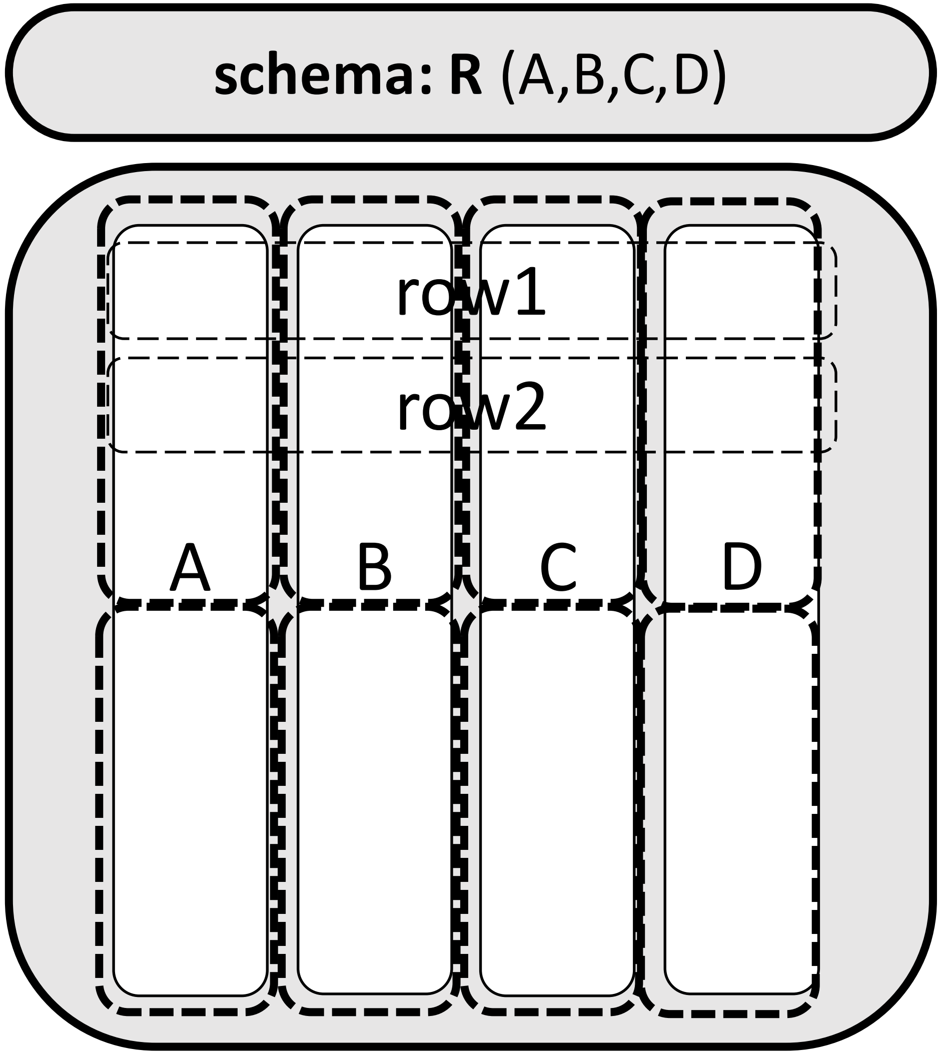
select A **from** R

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

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

BOSTON UNIVERSITY

querying over slotted pages



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

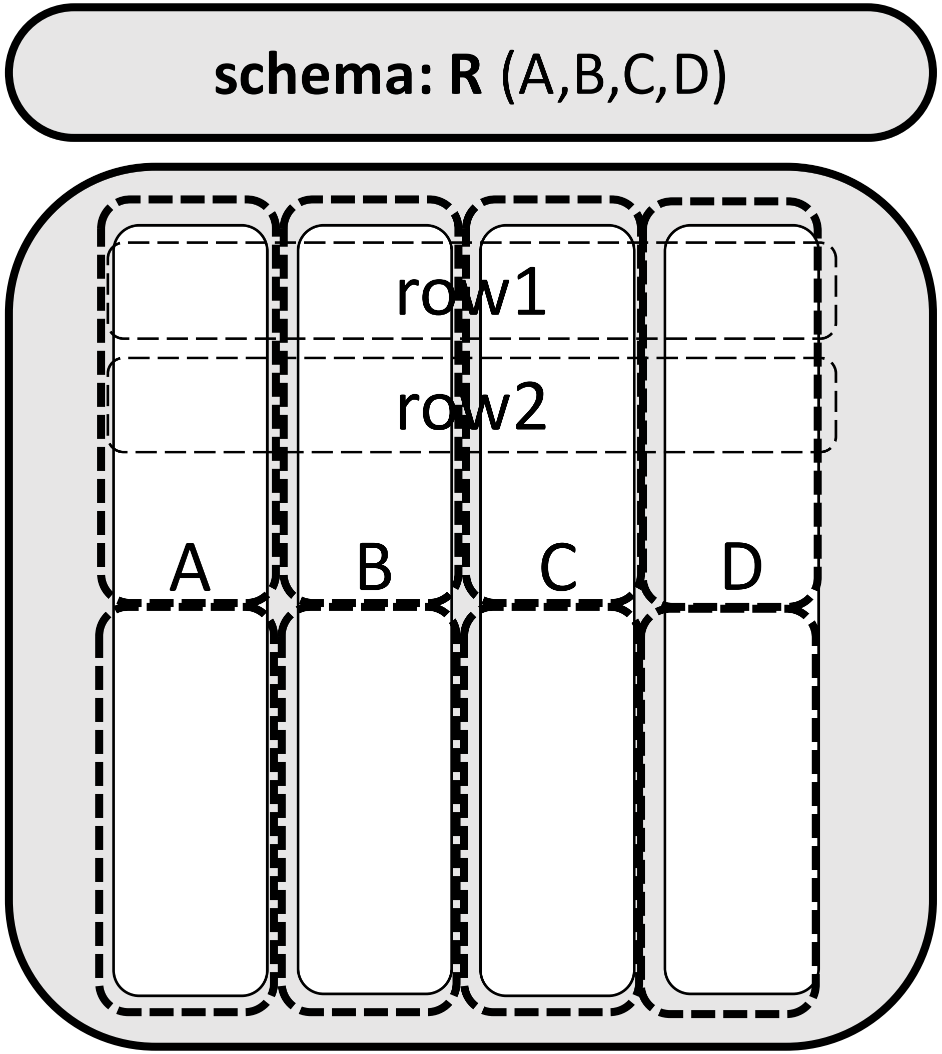
select A **from** R

any drawbacks?

each page contains **columns!**

column store

querying over slotted pages



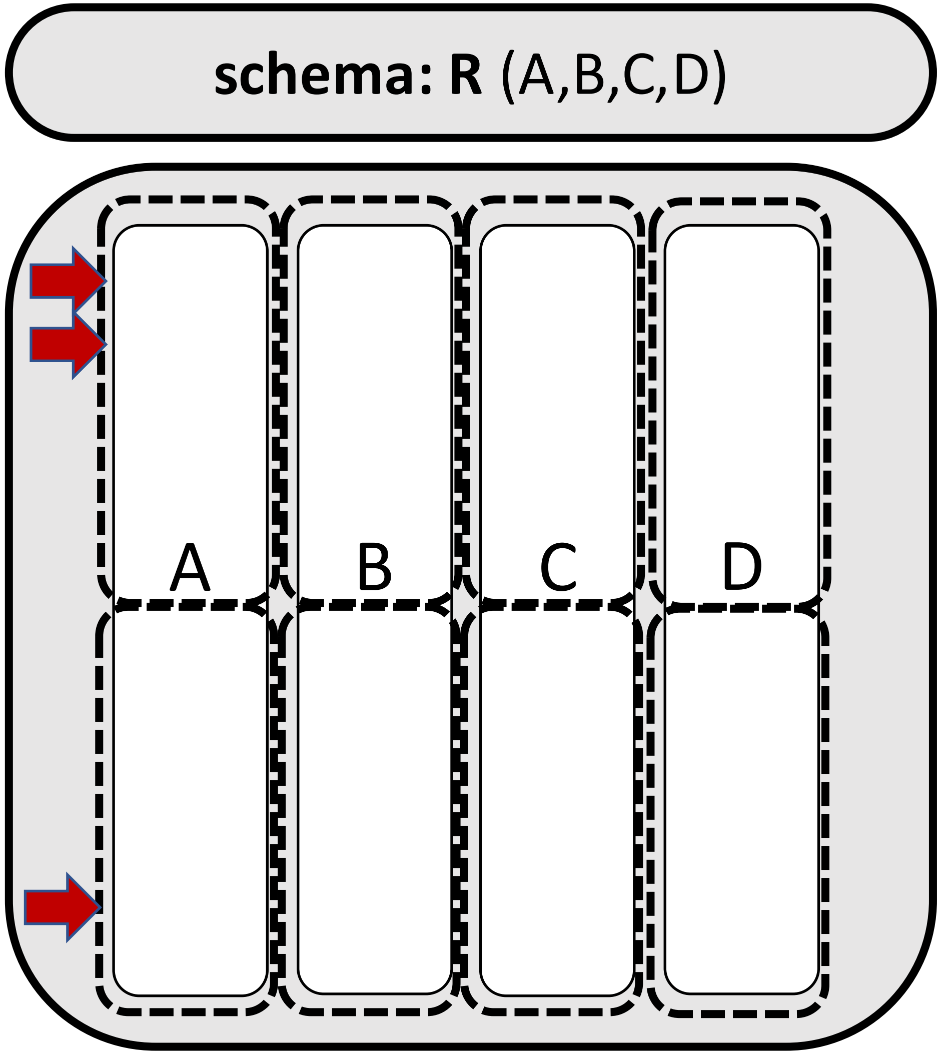
each page contains **columns!**

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

select A **from** R

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

querying over slotted pages



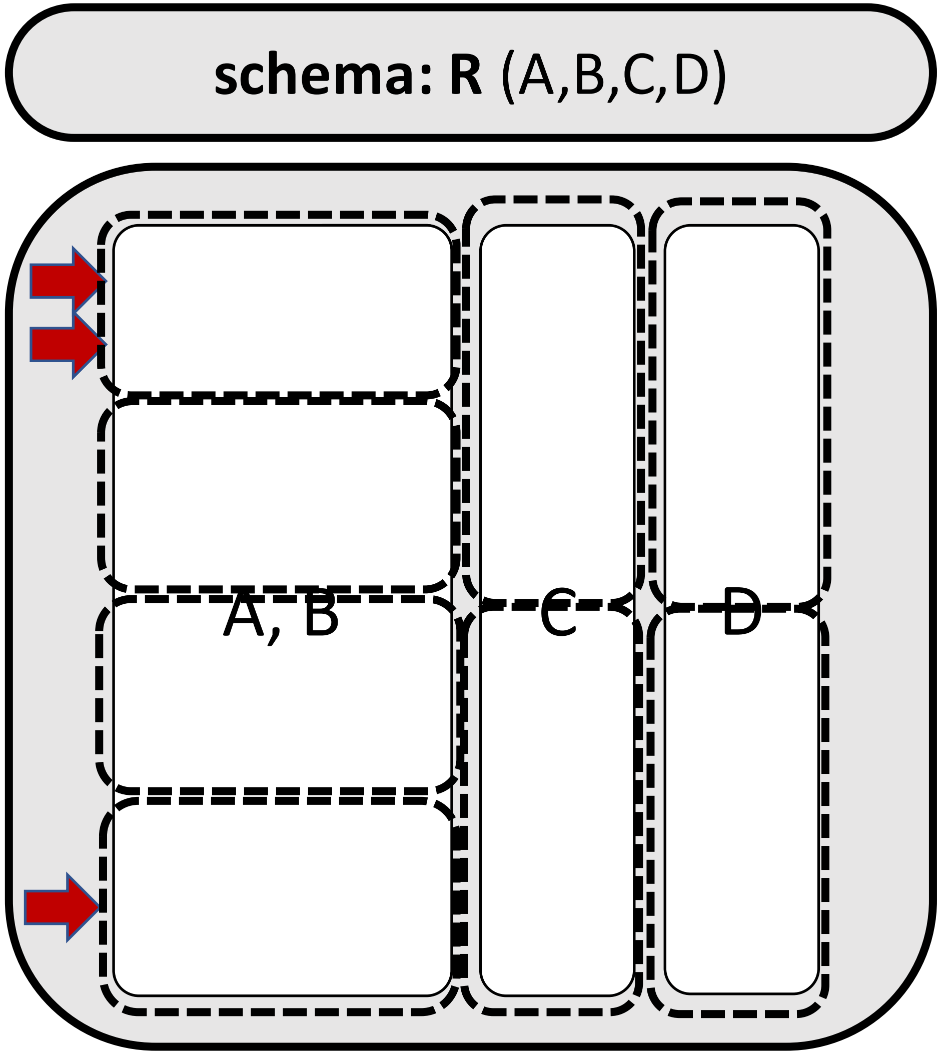
select A,B,C,D **from** R

select A **from** R

select (A+B) **from** R **where** A>10

each page contains **columns!**

querying over slotted pages



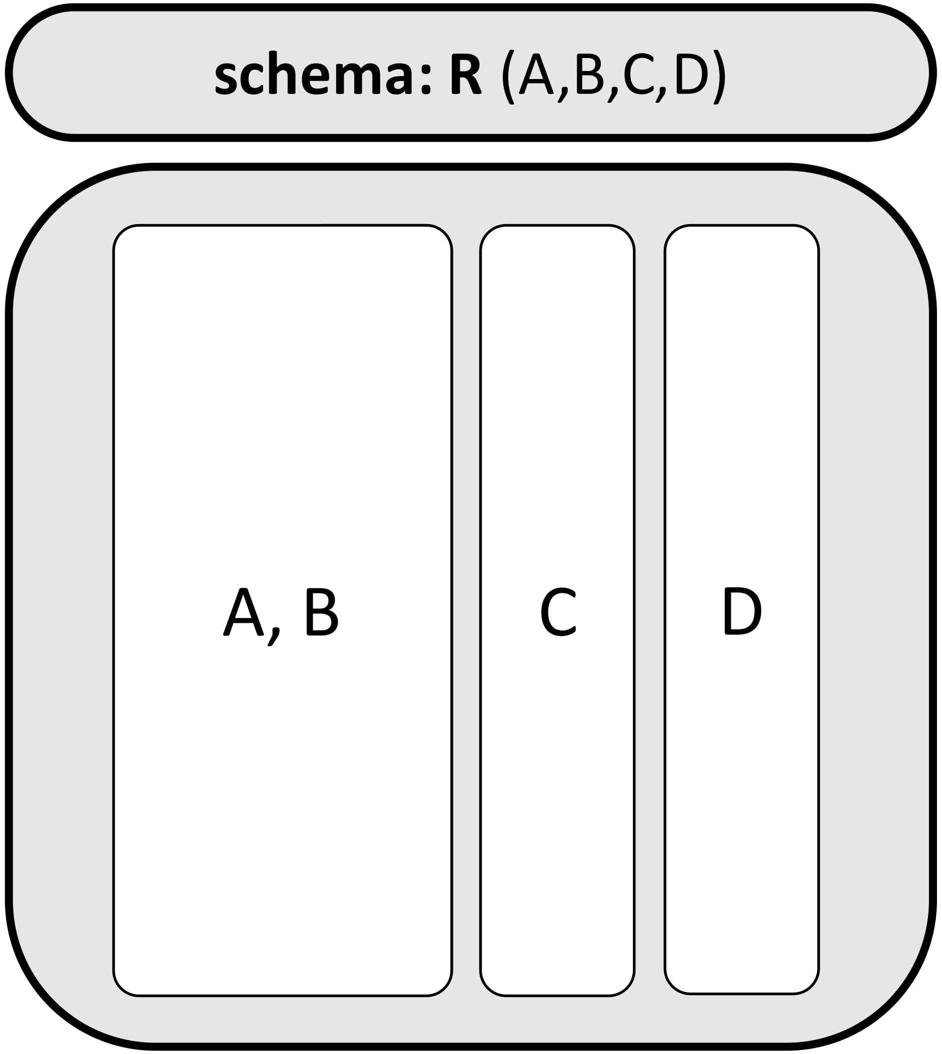
select A,B,C,D **from** R

select A **from** R

select (A+B) **from** R **where** A>10

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

querying over slotted pages



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

select A **from** R

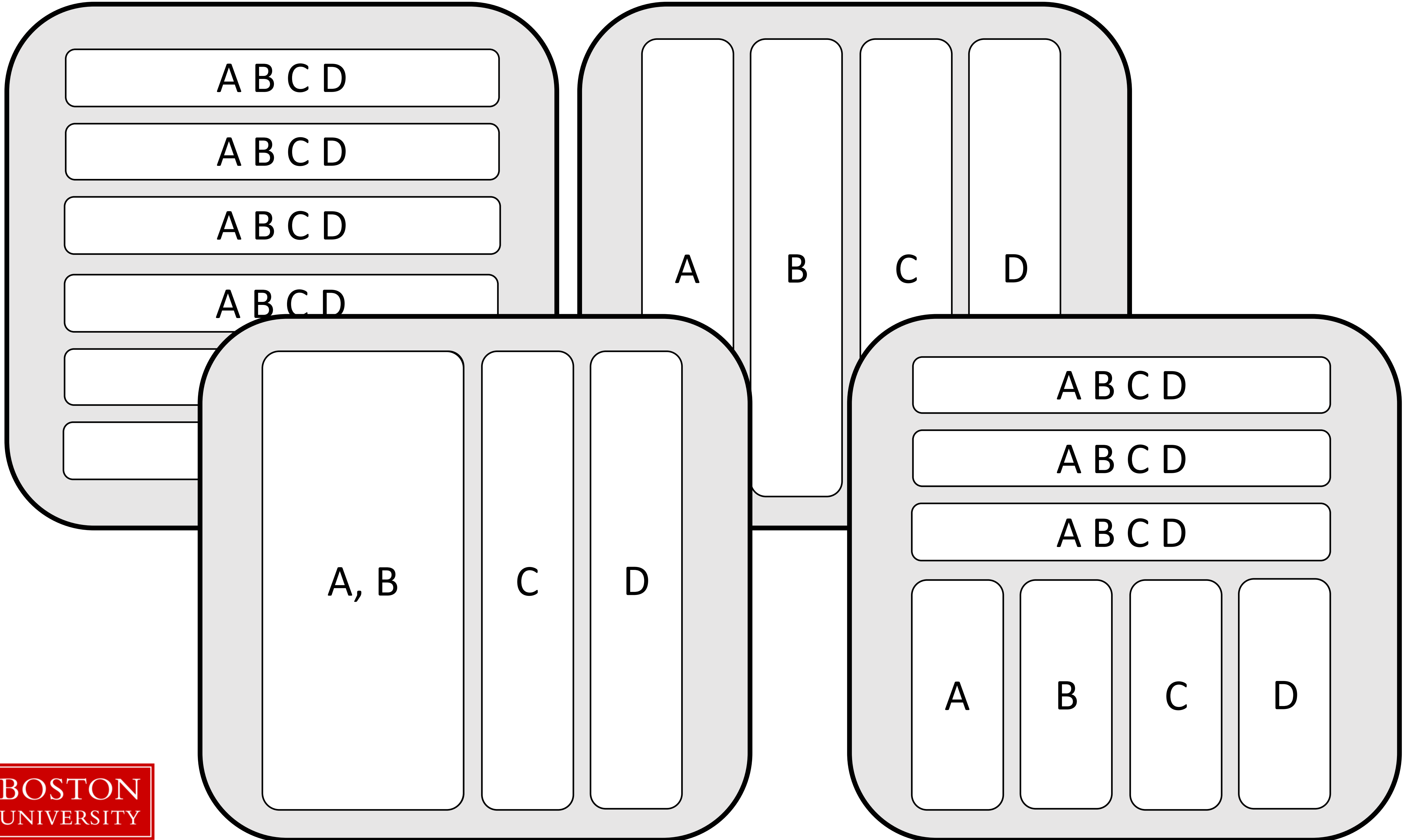
select (A+B) **from** R **where** A>10

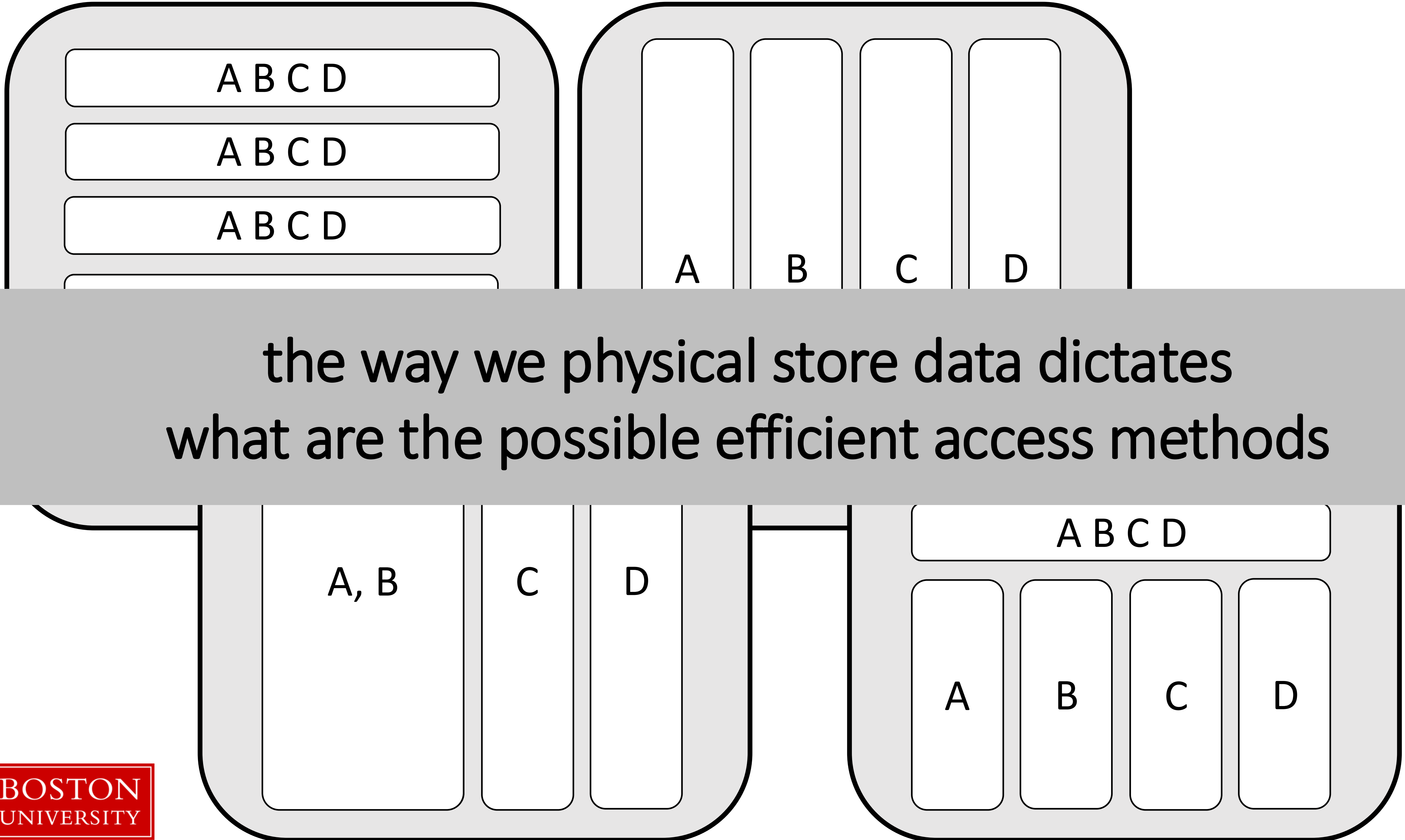
each page contains **columns** or ***groups of columns***!

what if I had all three queries?

what if only inserts/updates?

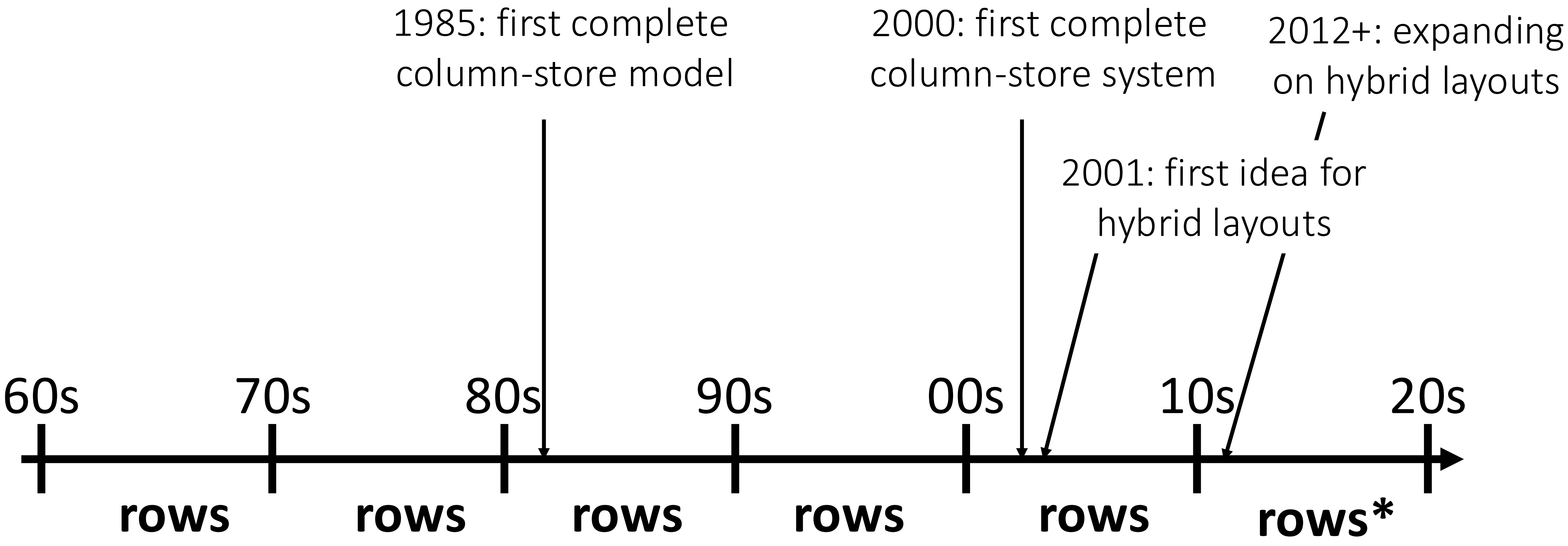
can there be something in between?


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

column-stores history line



1985: first complete column-store model

2000: first complete column-store system

2012+: expanding on hybrid layouts

/

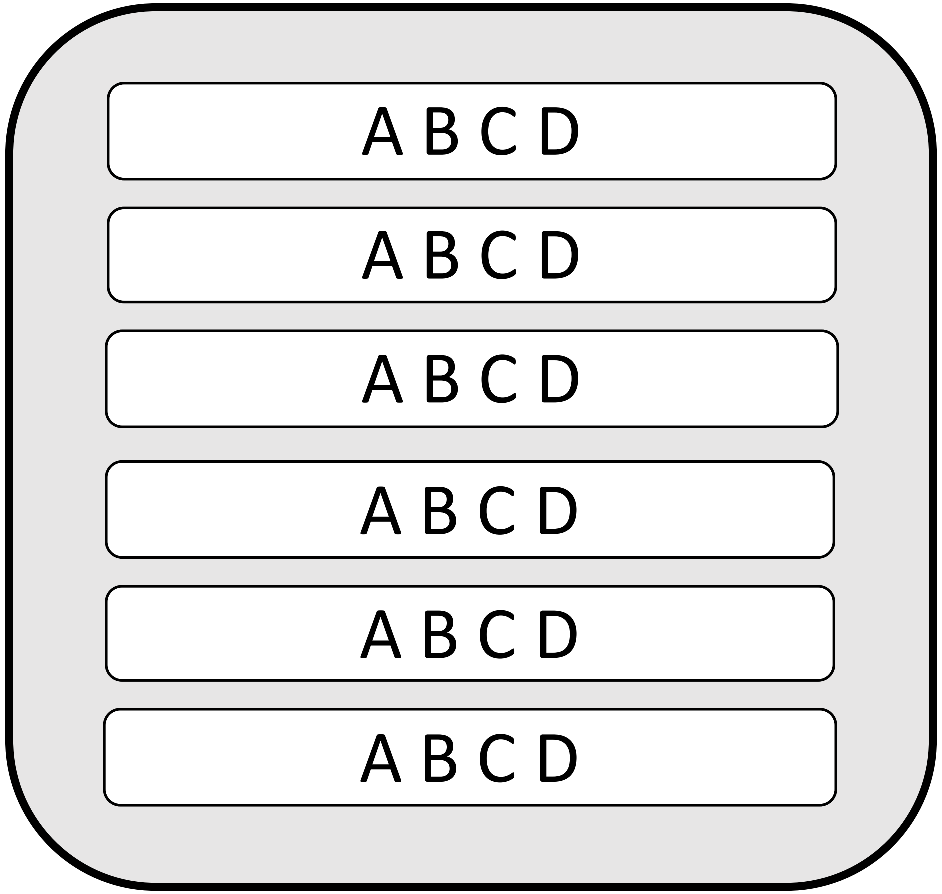
2001: first idea for hybrid layouts

60s 70s 80s 90s 00s 10s 20s

rows **rows** **rows** **rows** **rows** **rows***

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

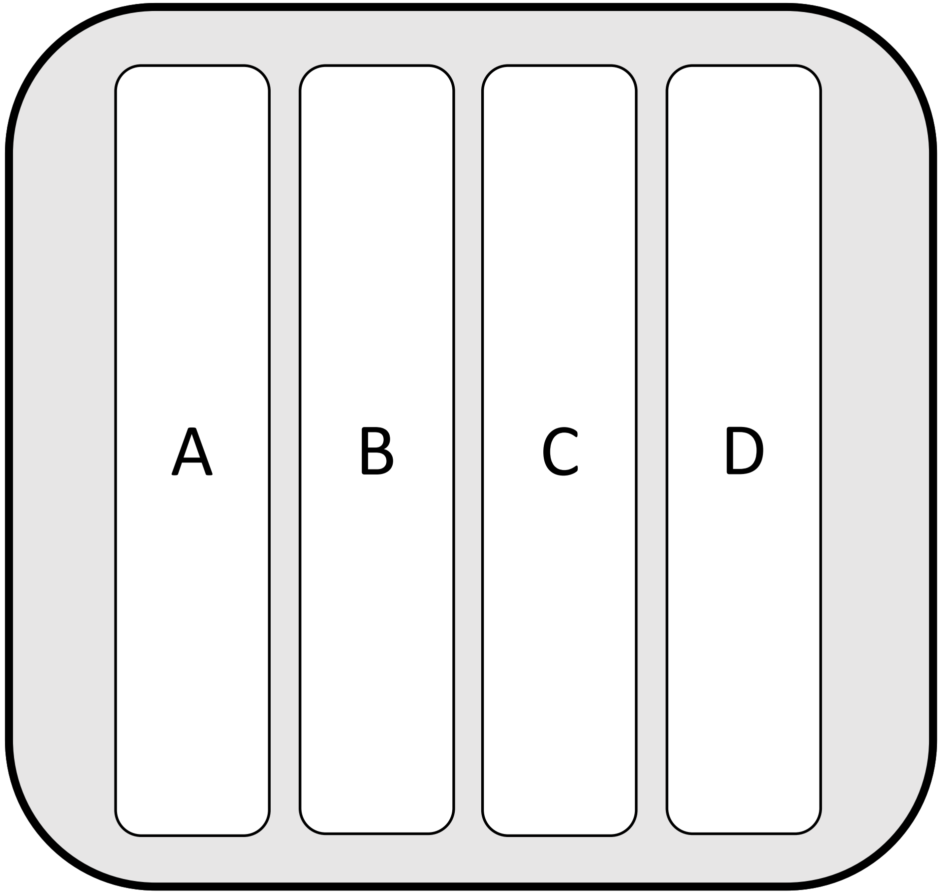
A B C D

A B C D

tuple reconstruction/early materialization

A B C D

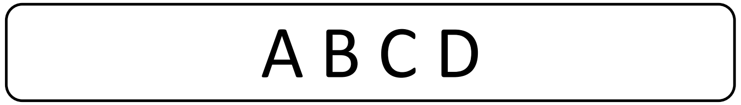
one row at a time



A B C D

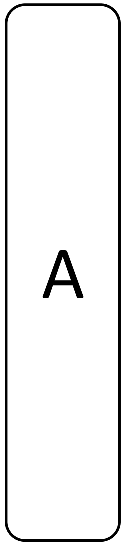
select max(B) **from** R **where** A>5 **and** C<10

tuple reconstruction/early materialization



A B C D

one row at a time

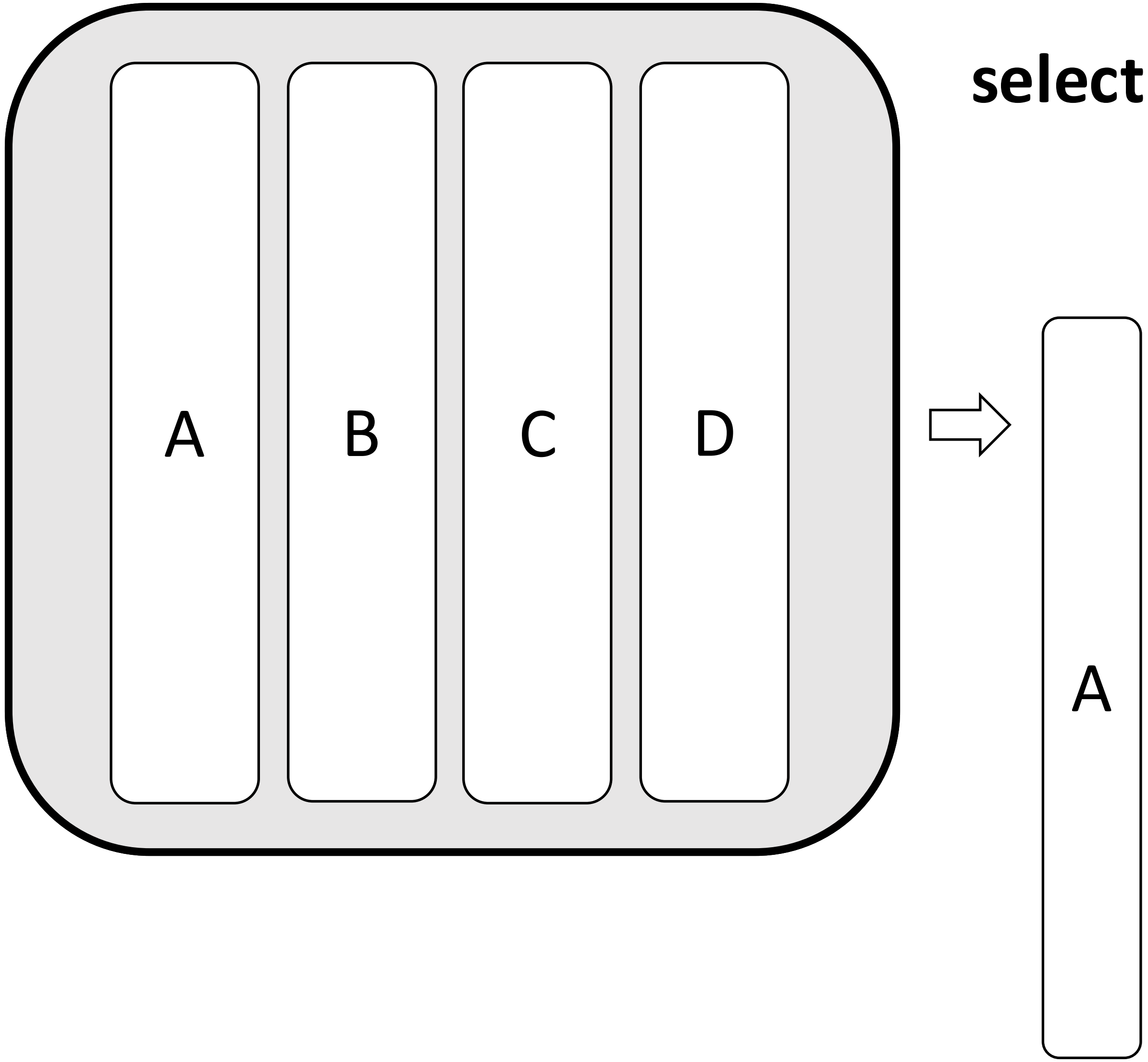


A

late materialization

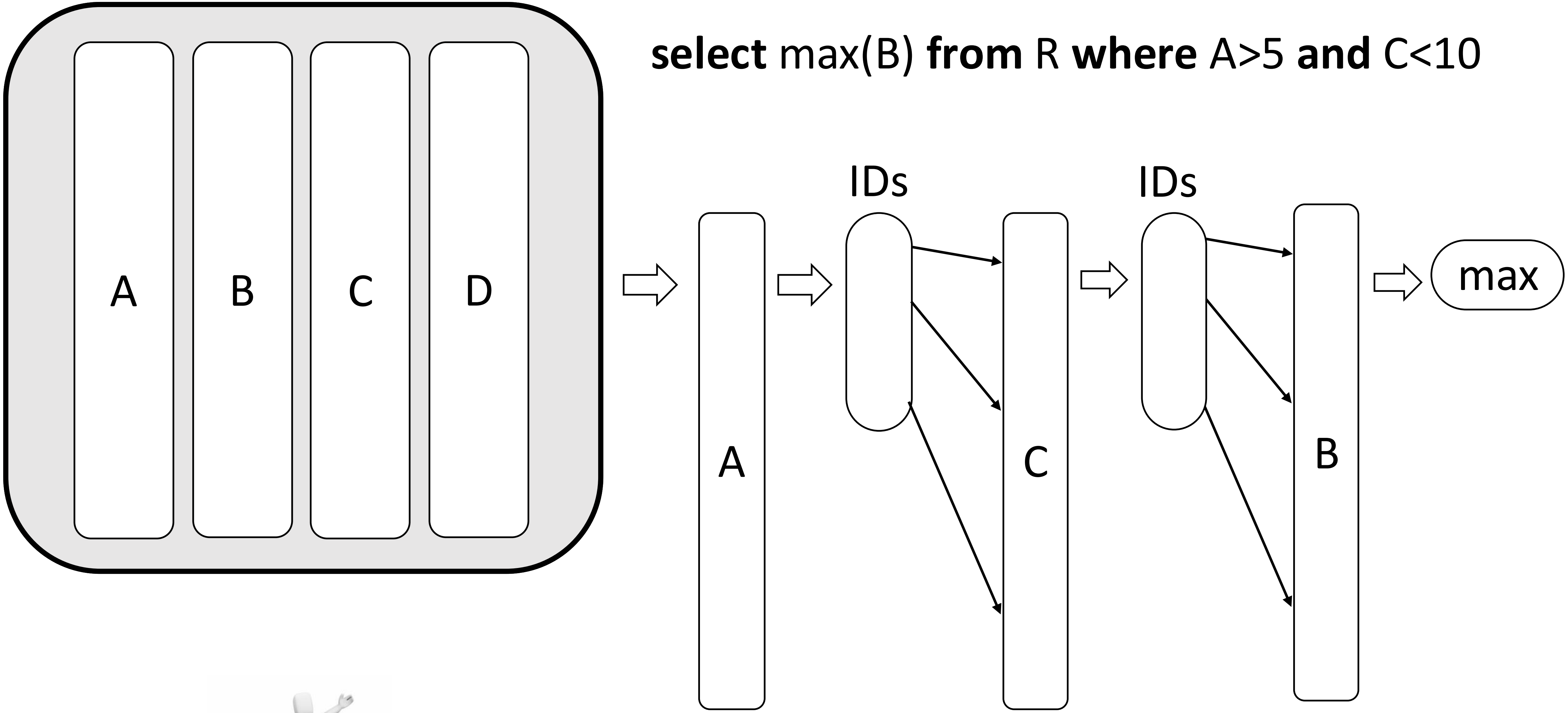
column at a time

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?

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

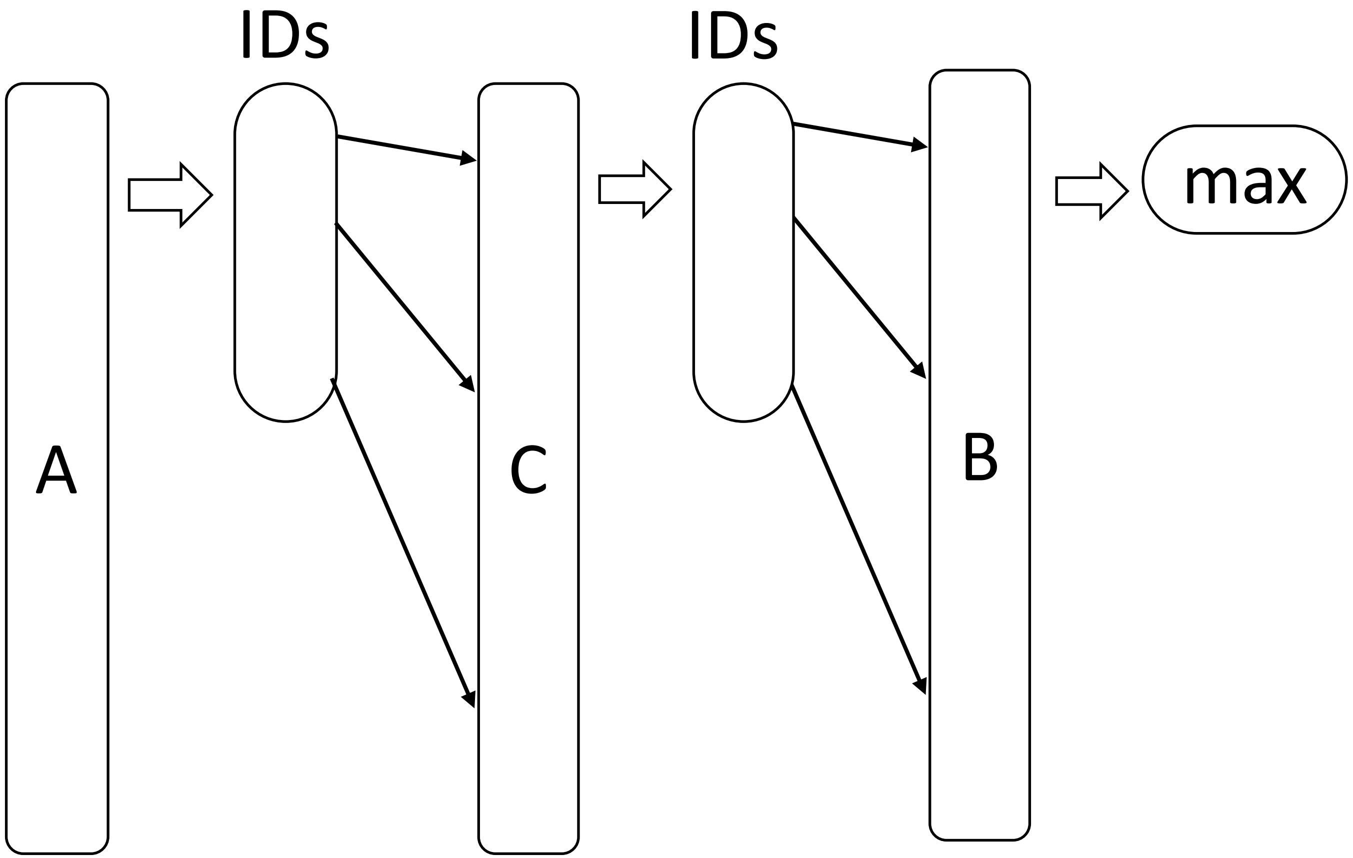
easy to prefetch

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

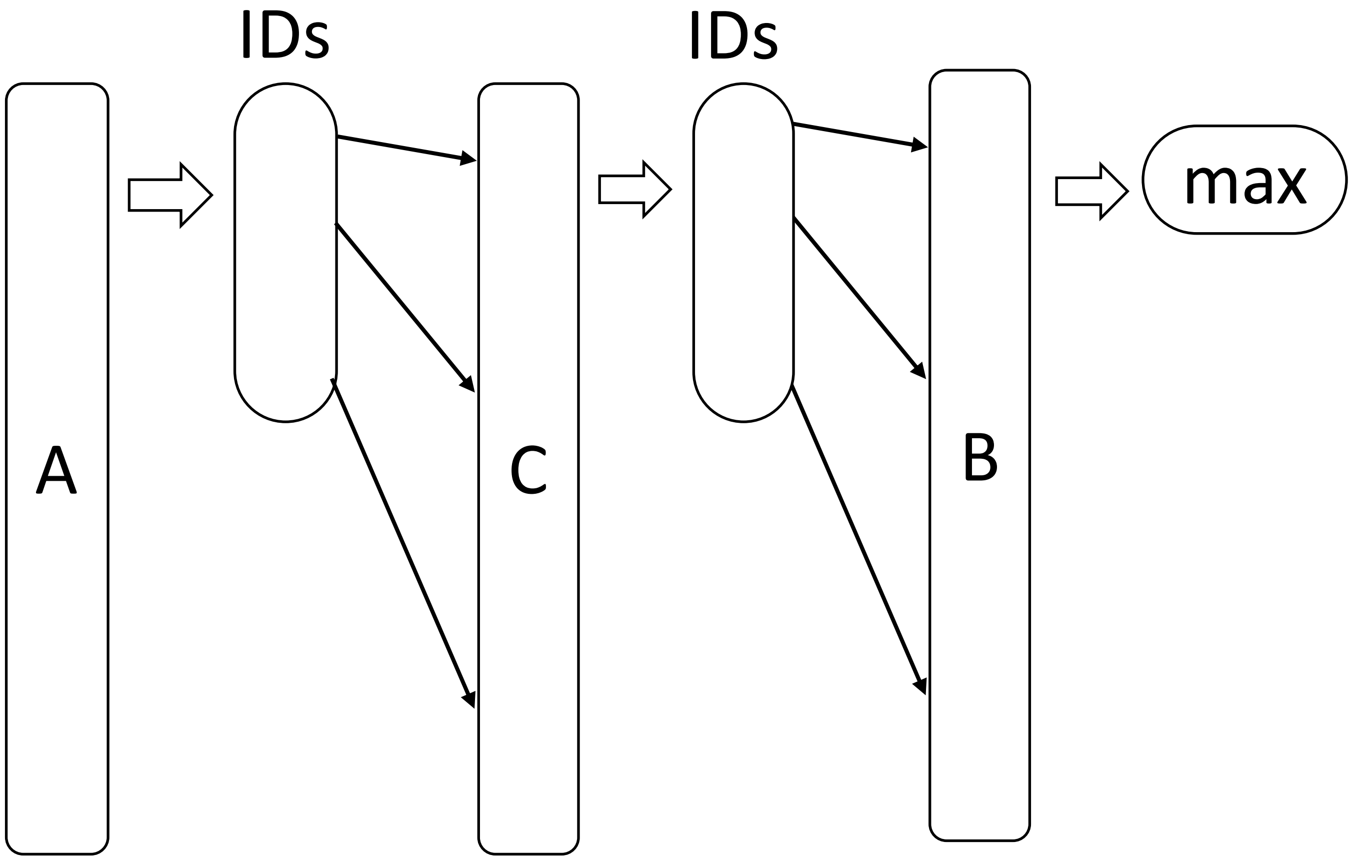


alternative query plans

start from C (why?)

scan A & C in parallel and merge

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



whole column?

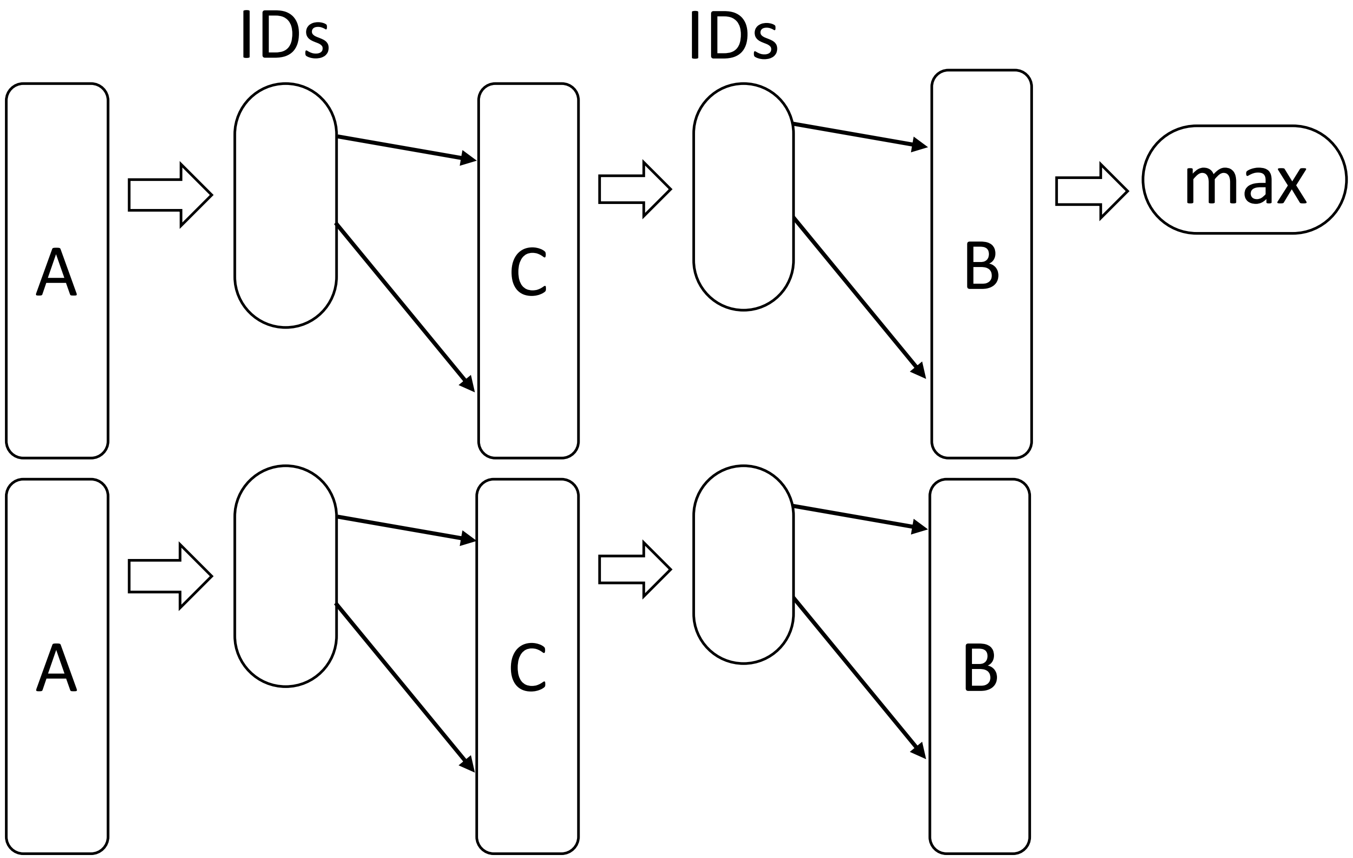
row at a time

column at a time

block/vector at a time

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

why column-stores are here now?

late materialization – no need to reconstruct tuples

read only useful data

minimize data movement across the memory hierarchy

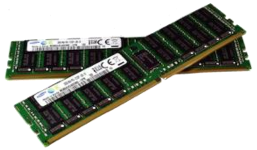
but it required a complete re-write

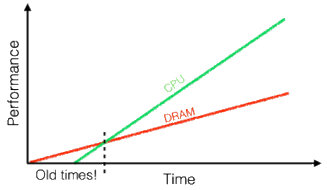
why not before?

legacy technology to catch up

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

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





Project details are now online (more to come)

Google BigTable
APACHE HBASE
AsterixDB
Cassandra
amazon DynamoDB
RocksDB
Couchbase
mongoDB

detailed discussion in next class

Readings **for the project**

The Log-Structured Merge-Tree (LSM-Tree) by Patrick E. O'Neil, Edward Cheng, Dieter Gawlick, Elizabeth J. O'Neil. Acta Inf. 33(4): 351-385, 1996

Monkey: Optimal Navigable Key-Value Store by Niv Dayan, Manos Athanassoulis, Stratos Idreos. SIGMOD Conference 2017

More readings *(for some research projects)*

Measures of Presortedness and Optimal Sorting Algorithms by Heikki Mannila. IEEE Trans. Computers 34(4): 318-325 (1985)

Small Materialized Aggregates: A Light Weight Index Structure for Data Warehousing by Guido Moerkotte. VLDB 1998

The adaptive radix tree: ARTful indexing for main-memory databases by Viktor Leis, Alfons Kemper, Thomas Neumann. ICDE 2013: 38-49

programming language: C/C++

it gives you **control over exactly** what is happening
it helps you **learn the impact** of design decisions

avoid using libraries unless asked to do,
so you can control storage and access patterns

BOSTON UNIVERSITY

CS 561: Data Systems Architectures

class 3

Relational Recap & Column-Stores Basics

Prof. Manos Athanassoulis

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