HARVARD
School of Engineering
and Applied Sciences

CS165
Data Systems

Have fun learning to design and build modern data systems

class 4

basic db architectures & layouts

prof. Stratos Idreos

HTTP://DASLAB.SEAS.HARVARD.EDU/CLASSES/CS165/

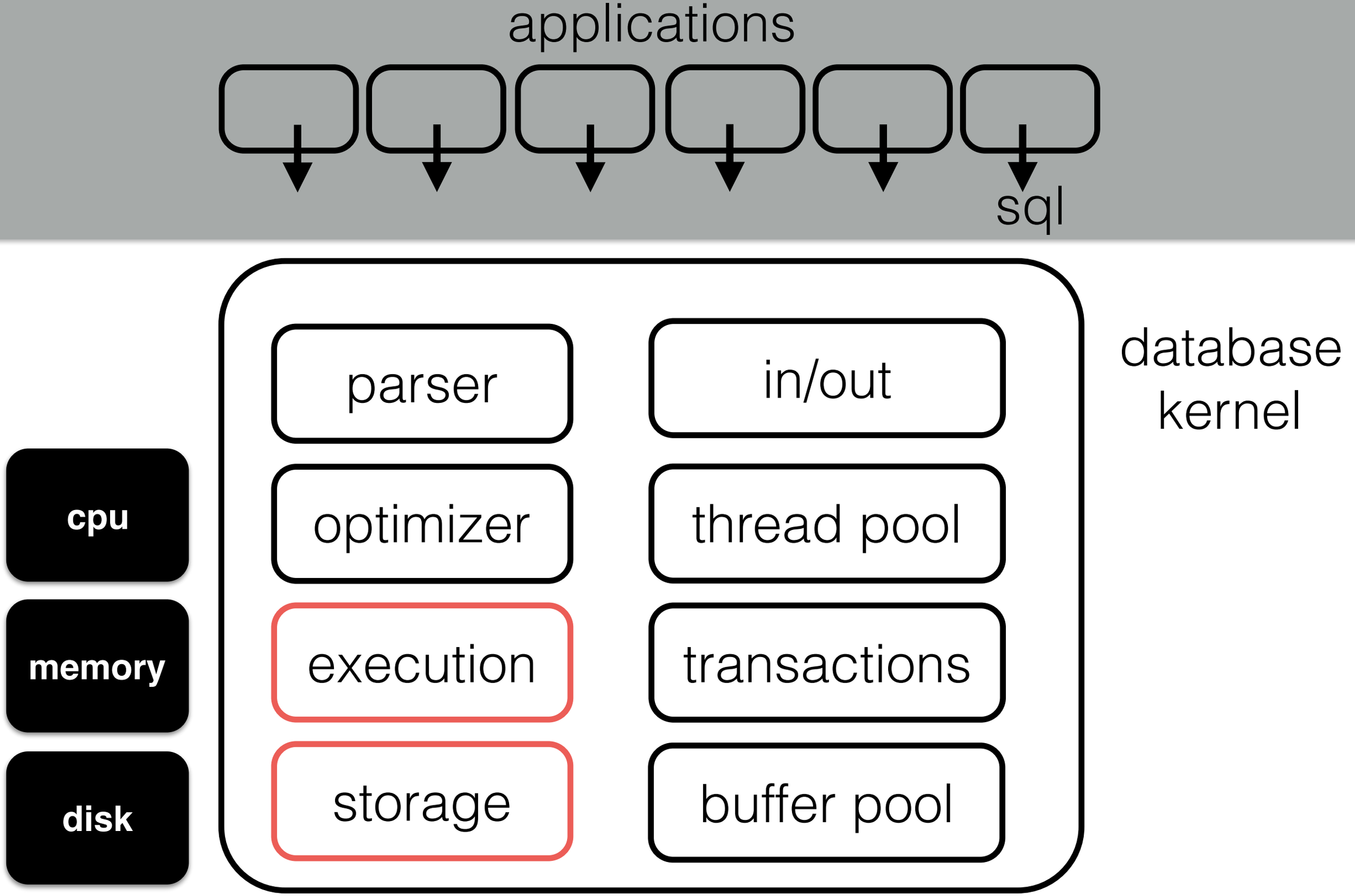
design

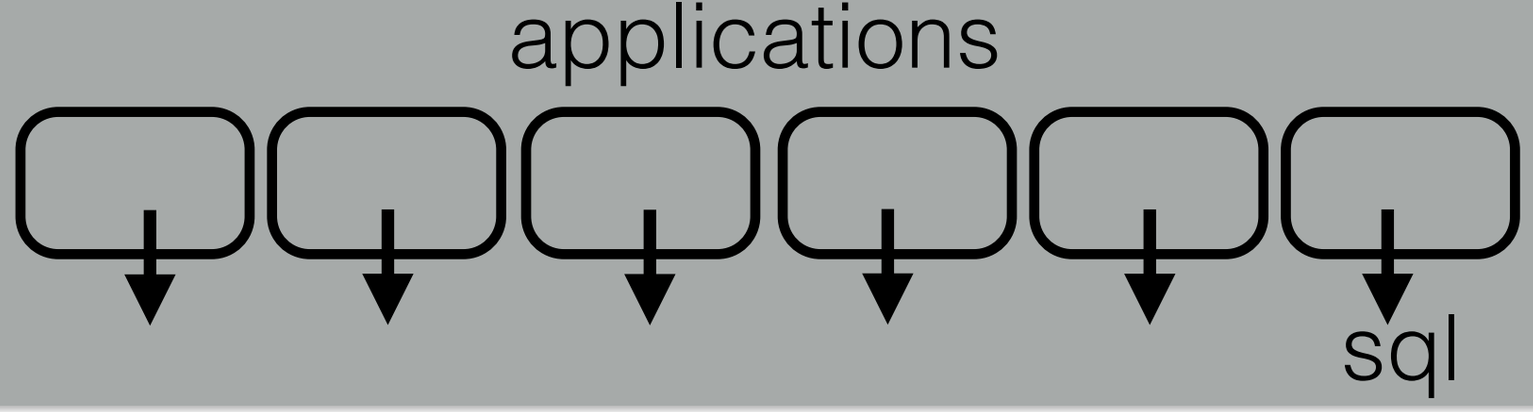
logical design

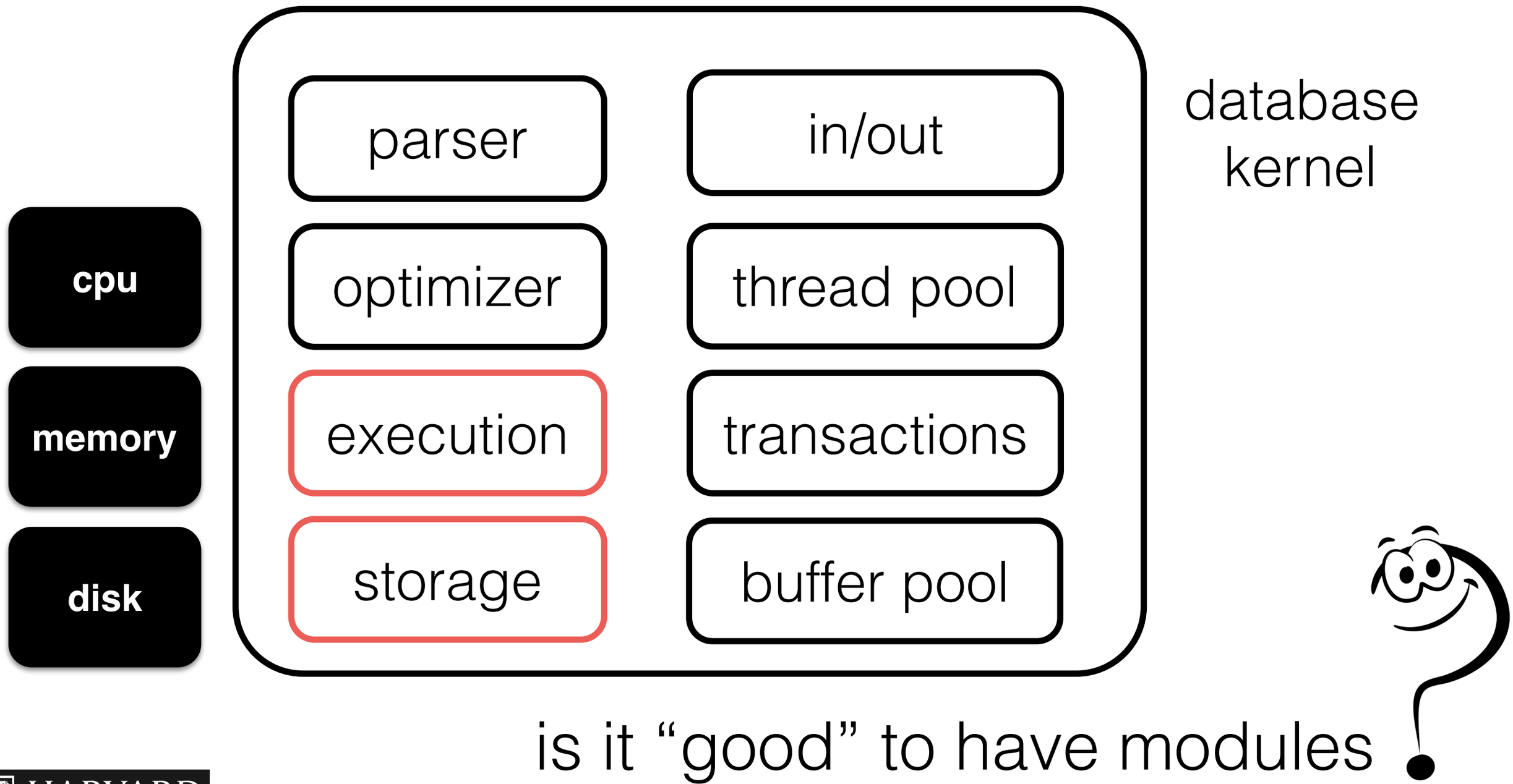
physical design

system design

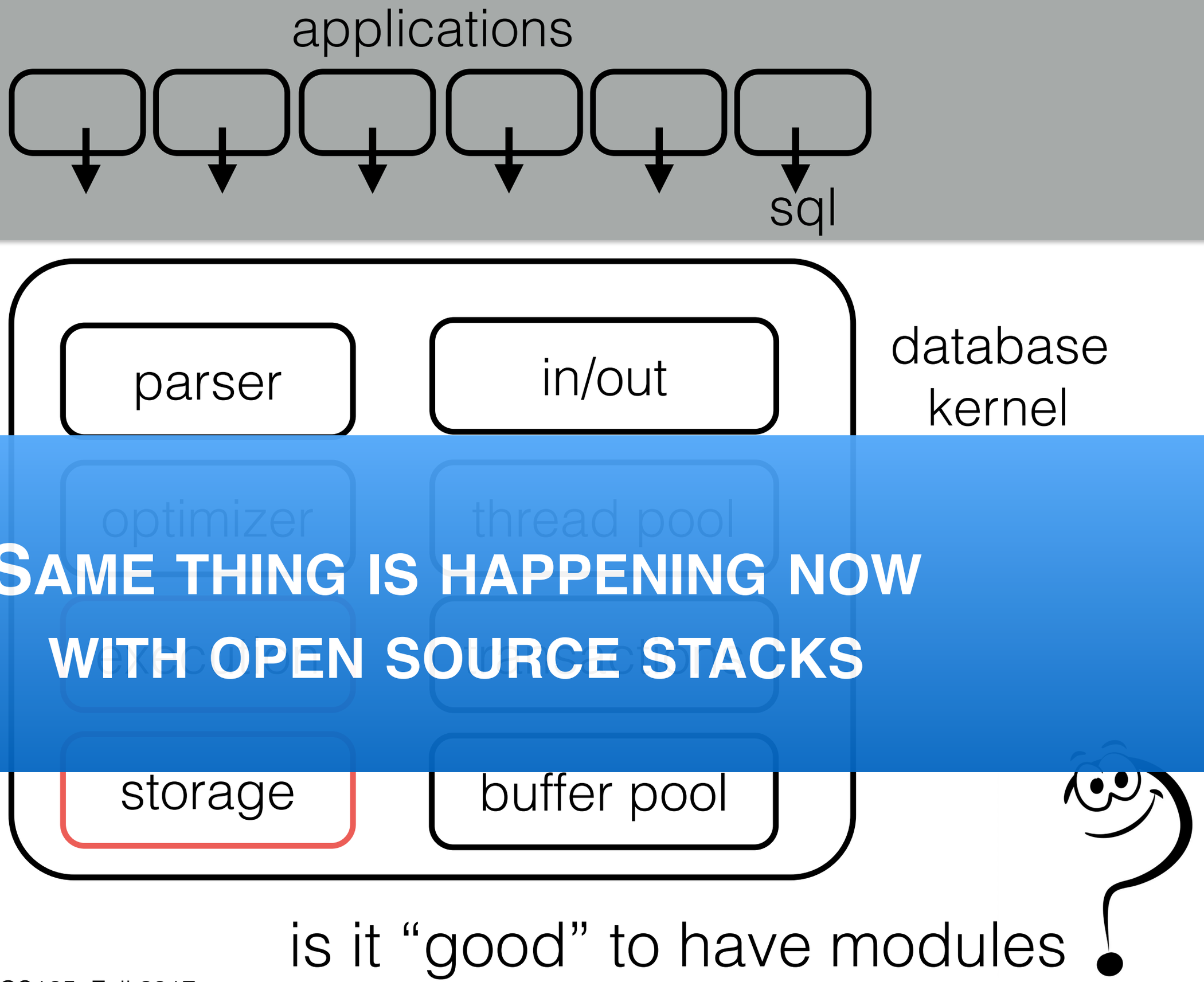
next up: db architectures 101


applications
sql
parser
in/out
optimizer
thread pool
execution
transactions
storage
buffer pool
database kernel
cpu
memory
disk



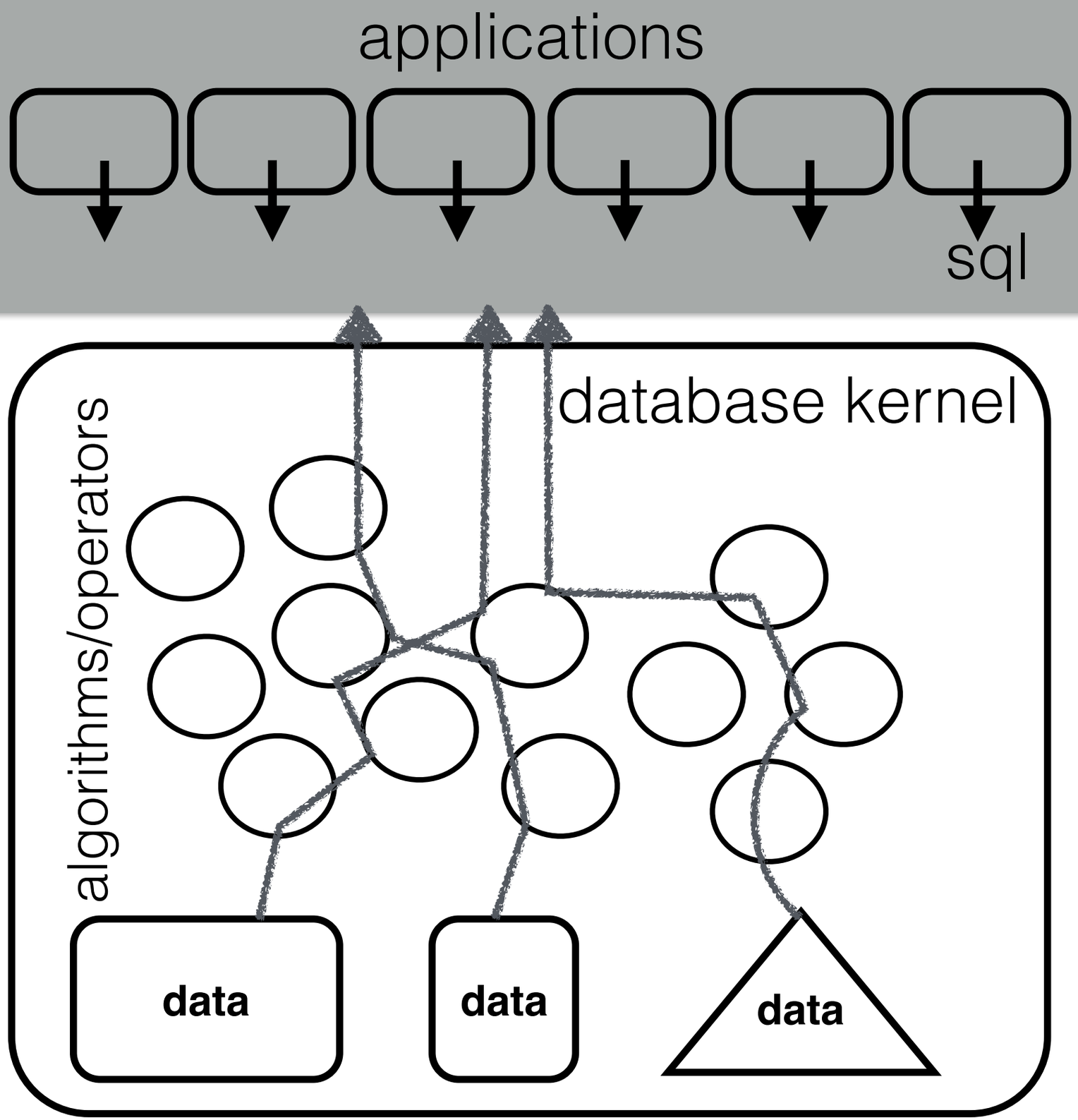


is it “good” to have modules



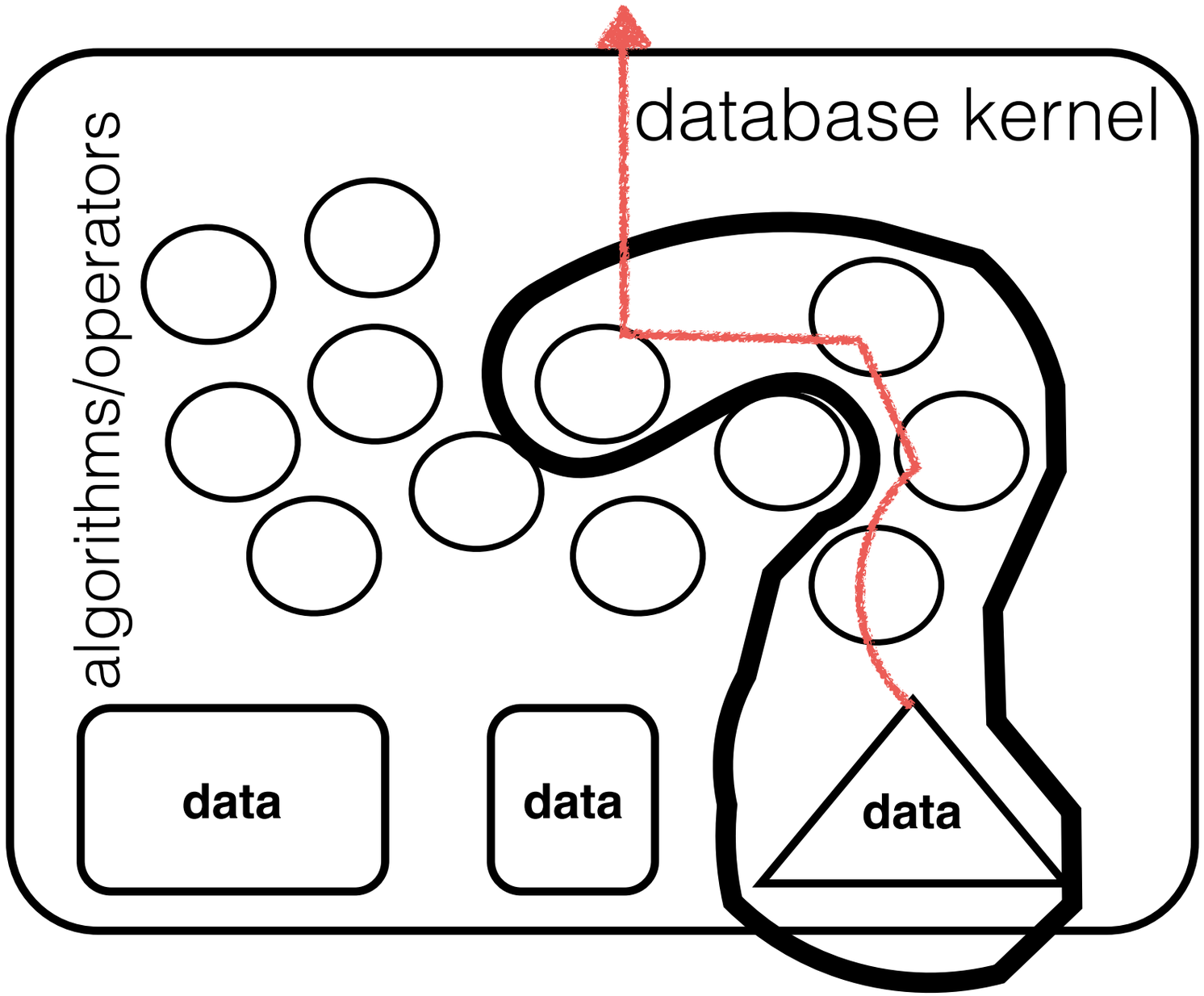
SAME THING IS HAPPENING NOW WITH OPEN SOURCE STACKS

is it “good” to have modules

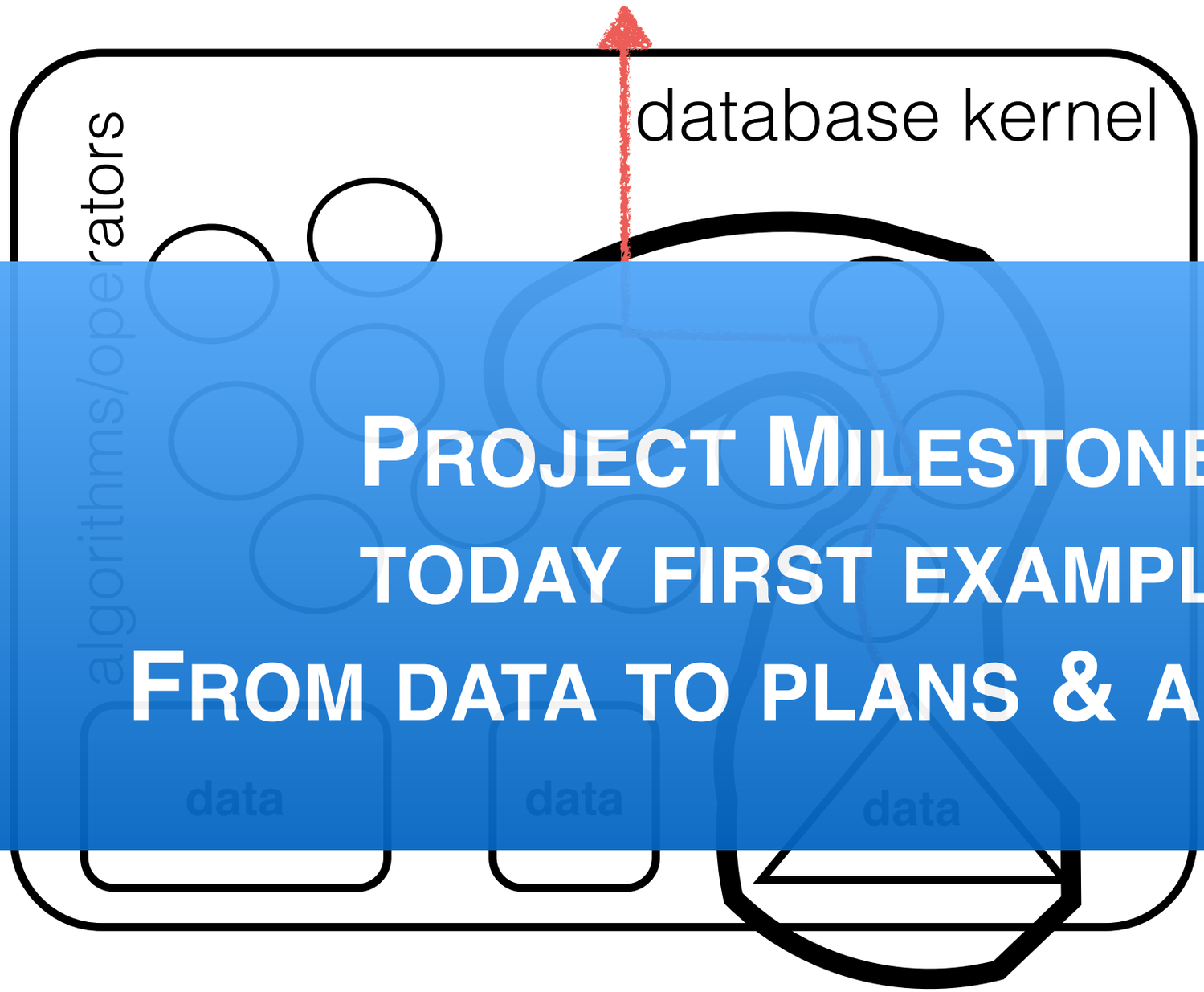


design/implement
numerous
possible algorithms
+ data representations

choose the best
data source, algorithms
and path for each query



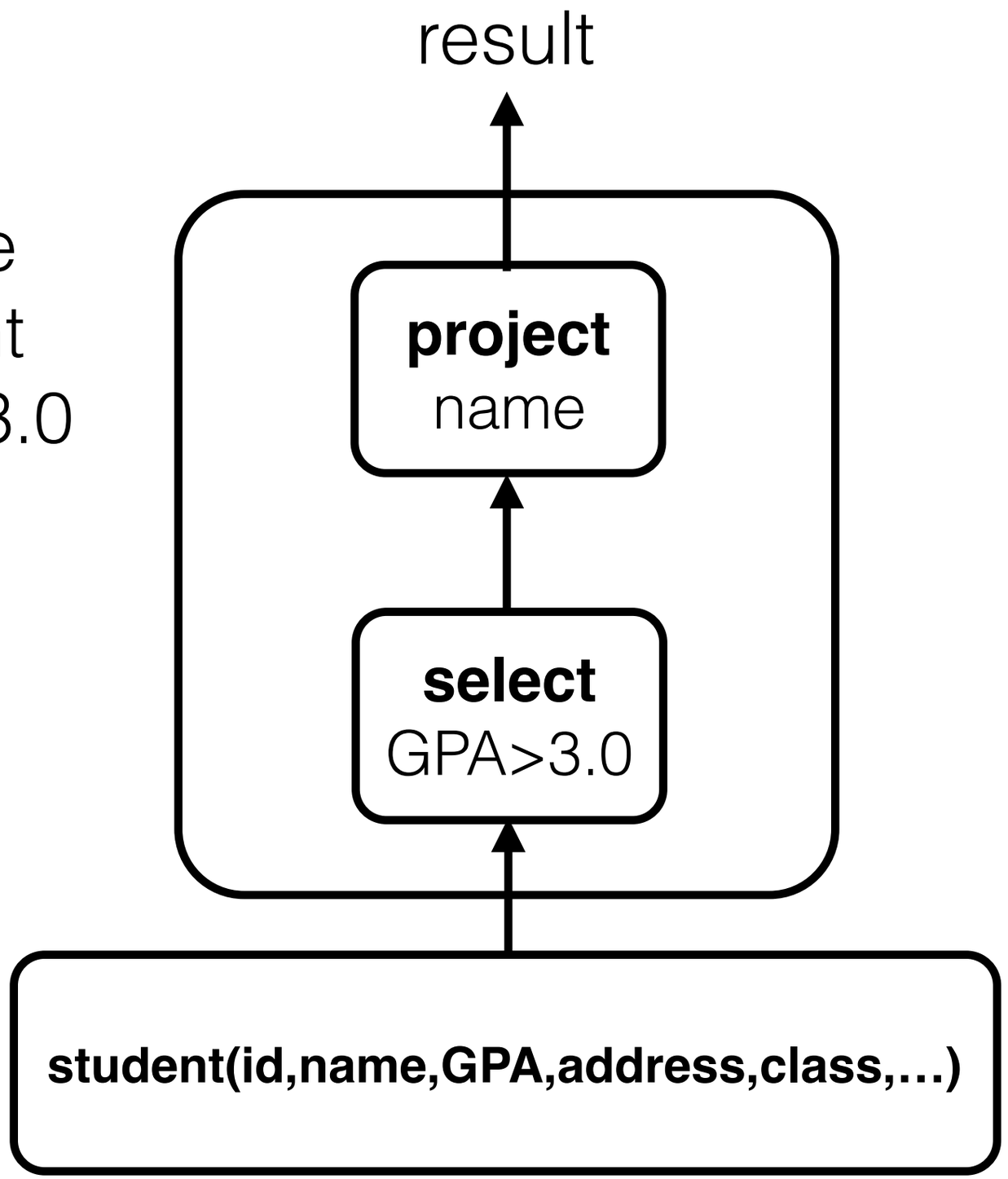
query plan



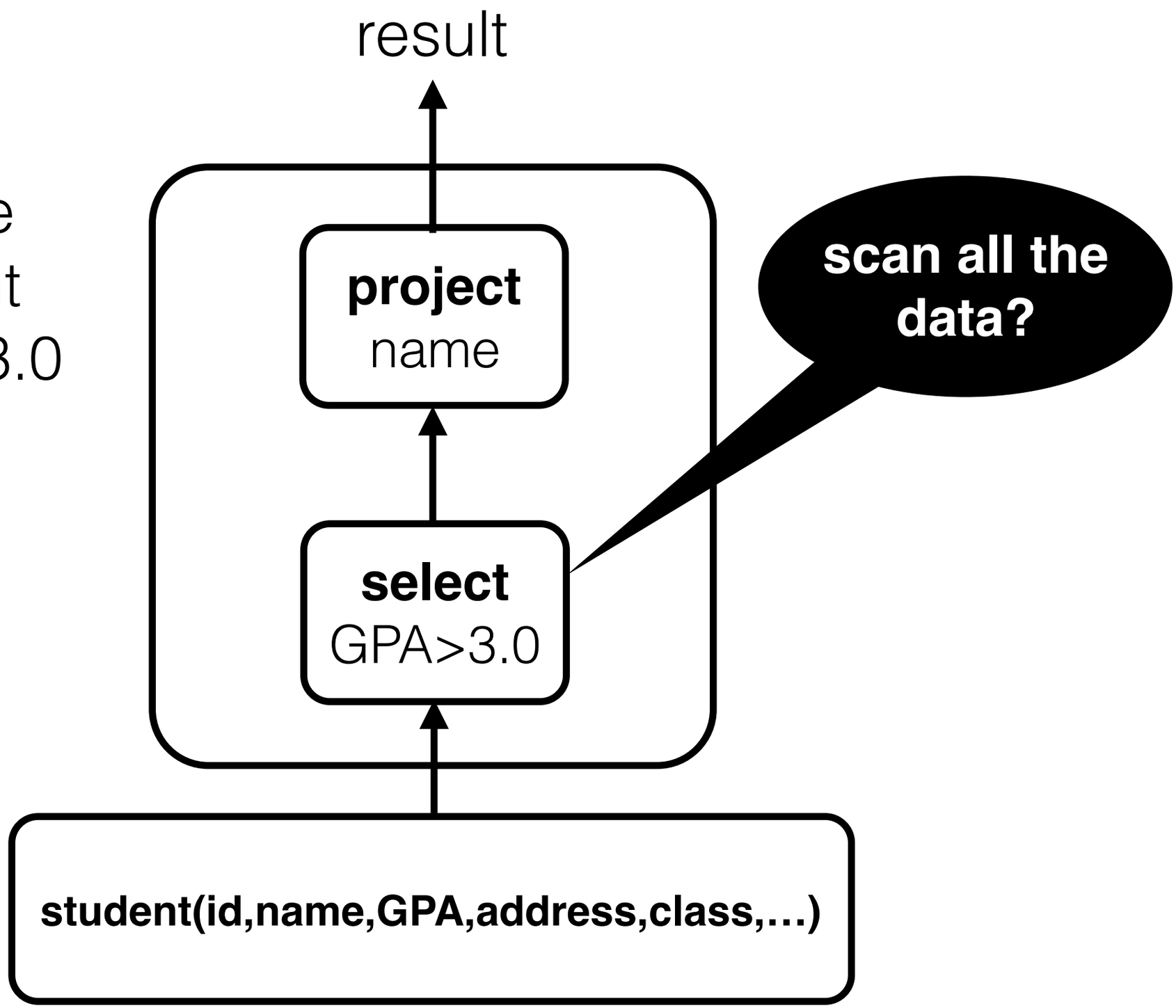
PROJECT MILESTONE 1

TODAY FIRST EXAMPLE:
FROM DATA TO PLANS & ANSWERS

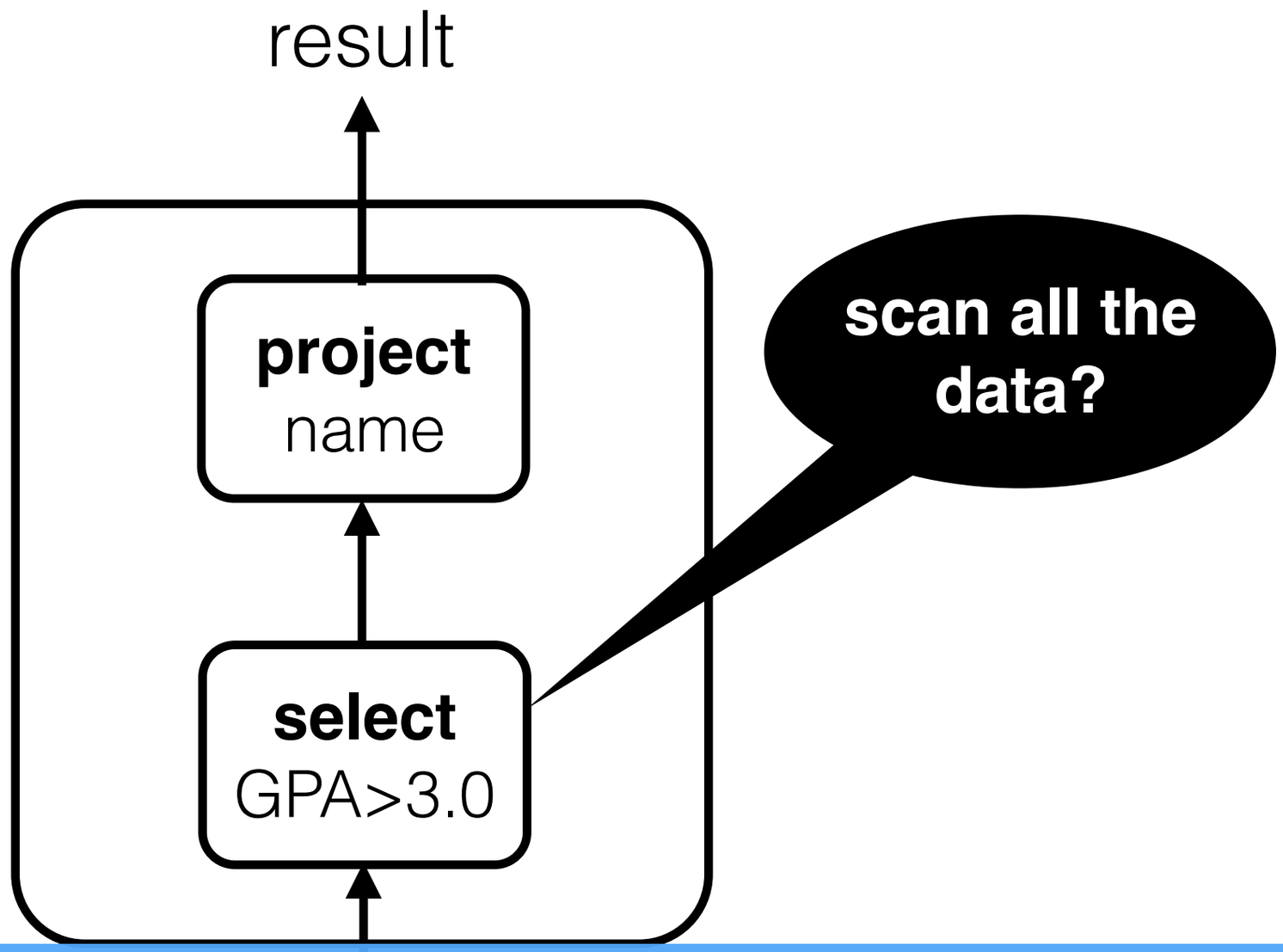
select name
from student
where GPA>3.0



select name
from student
where GPA>3.0



select name
from student
where GPA>3.0

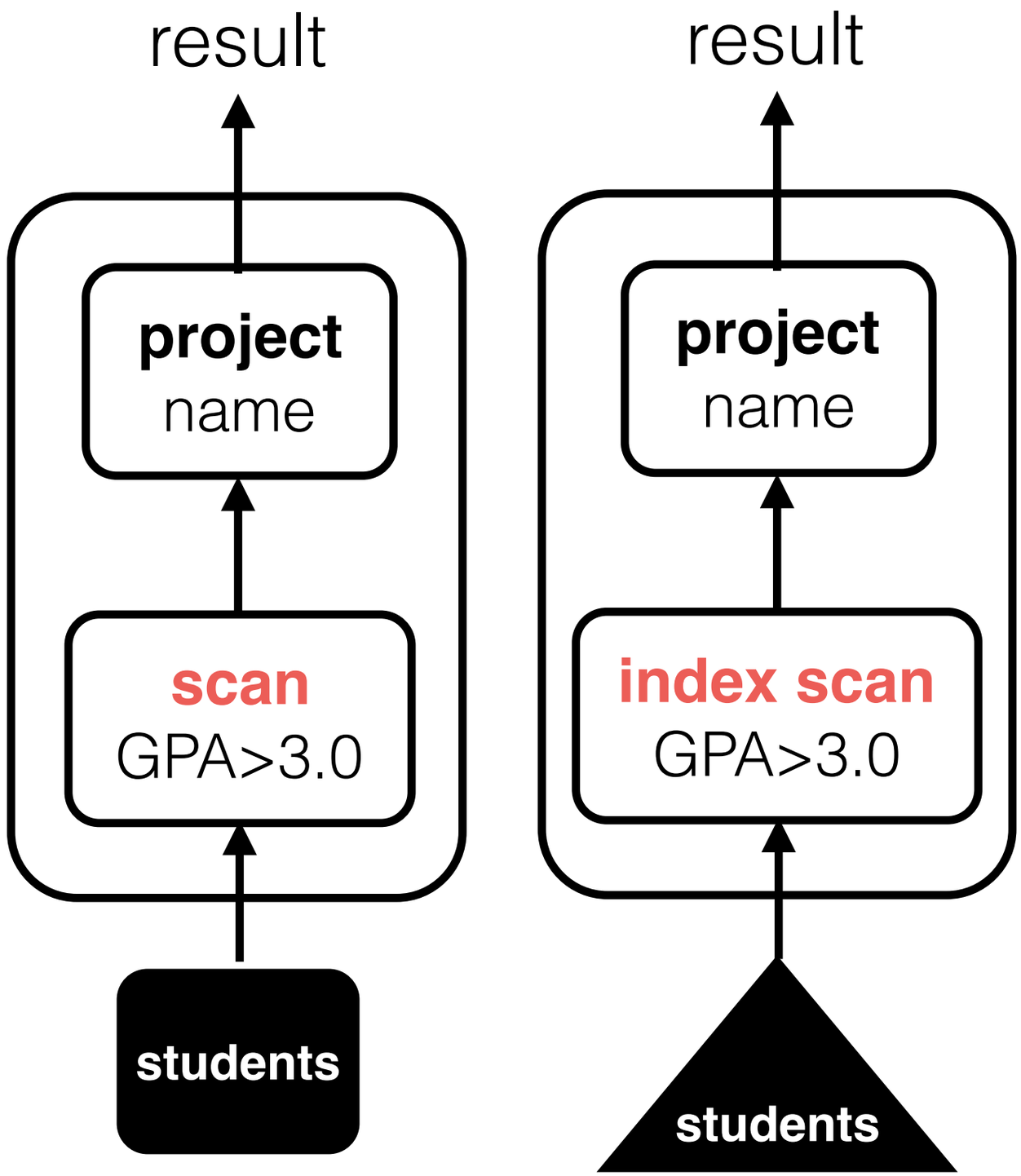


student(id,name,GPA,address,class,...)

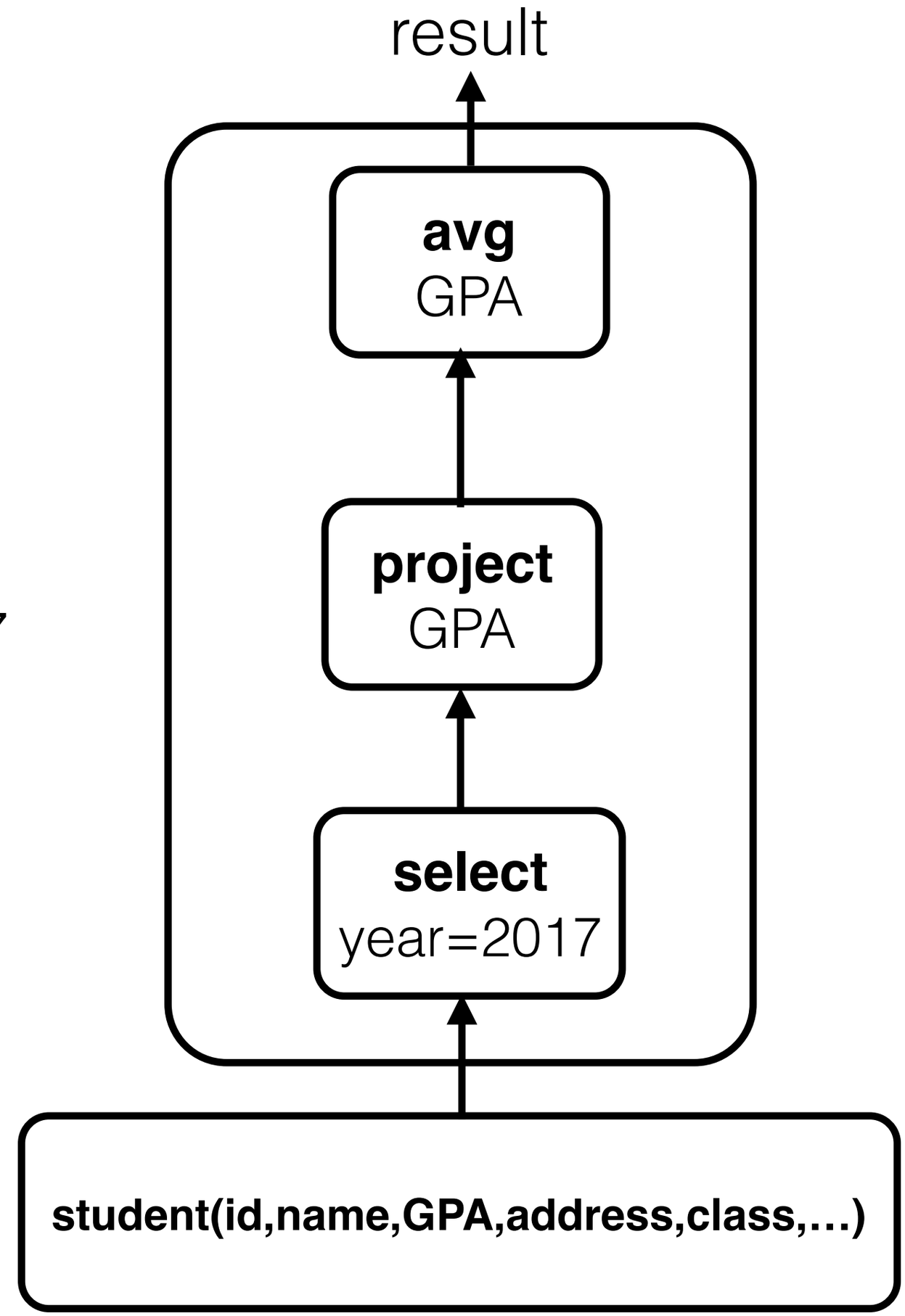
LOGICAL PLAN

select name
from student
where GPA>3.0

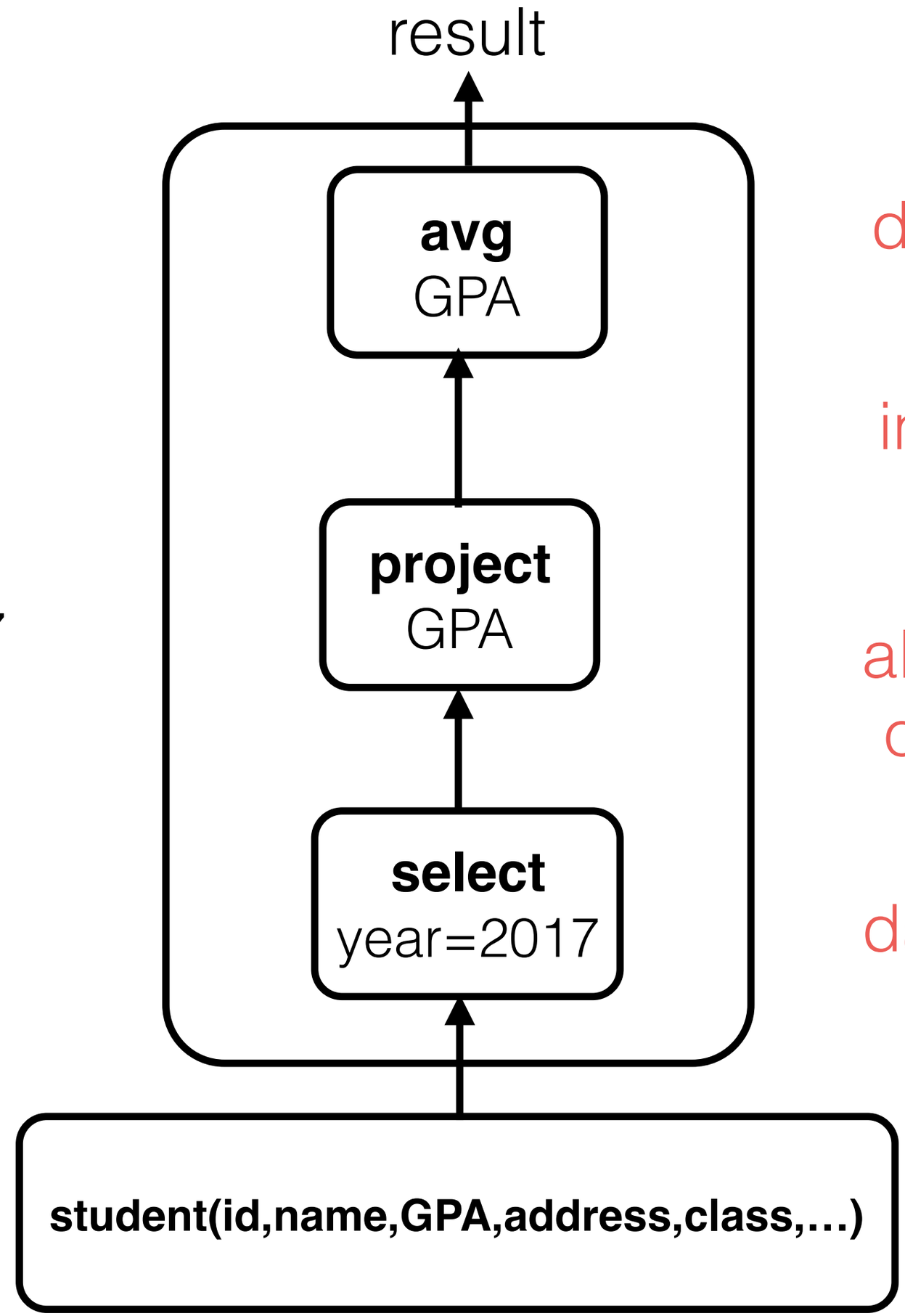
physical plans



select avg(GPA)
from student
where class=2017



select avg(GPA)
from student
where class=2017



data flow

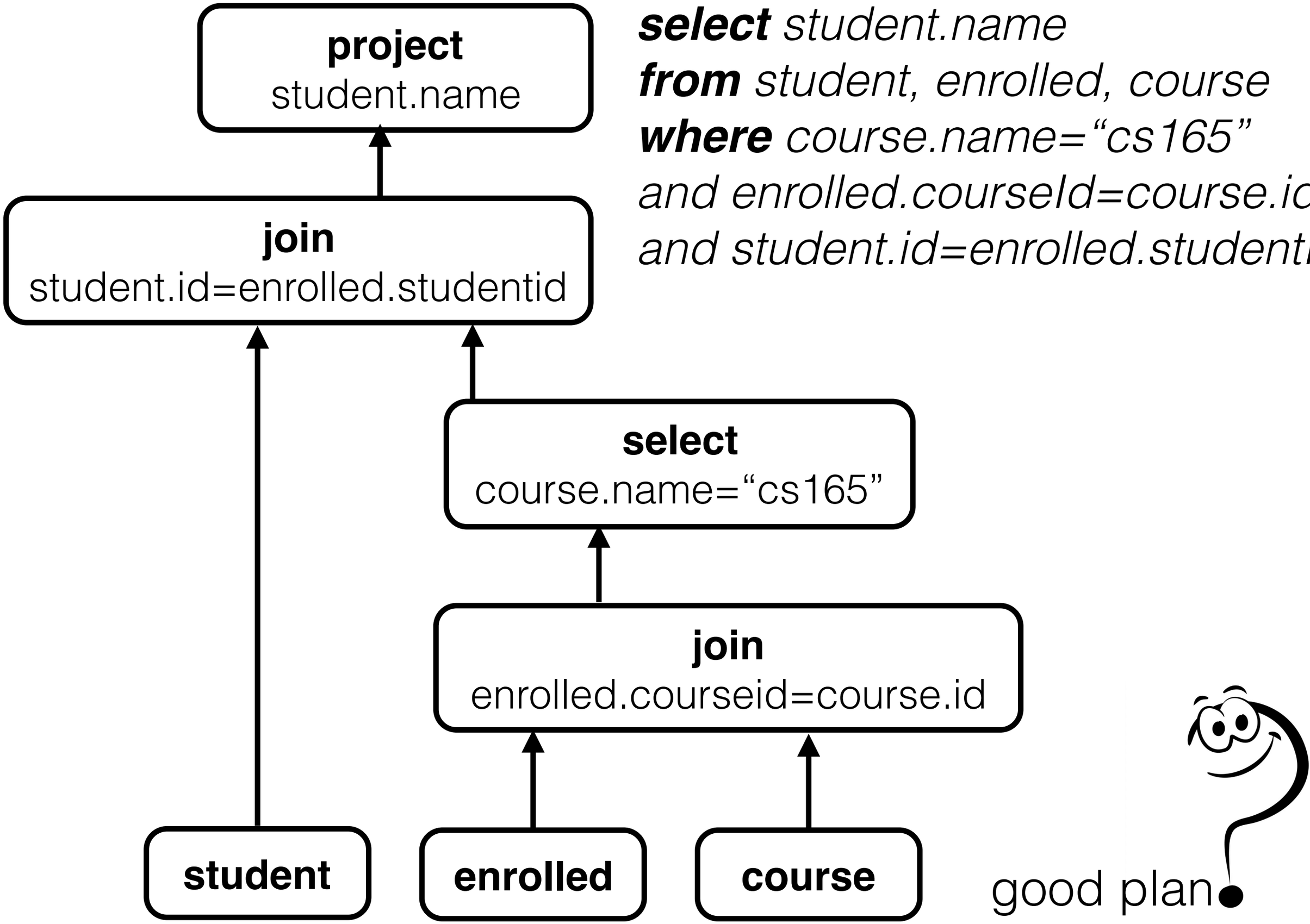
interfaces

algorithms/
operators

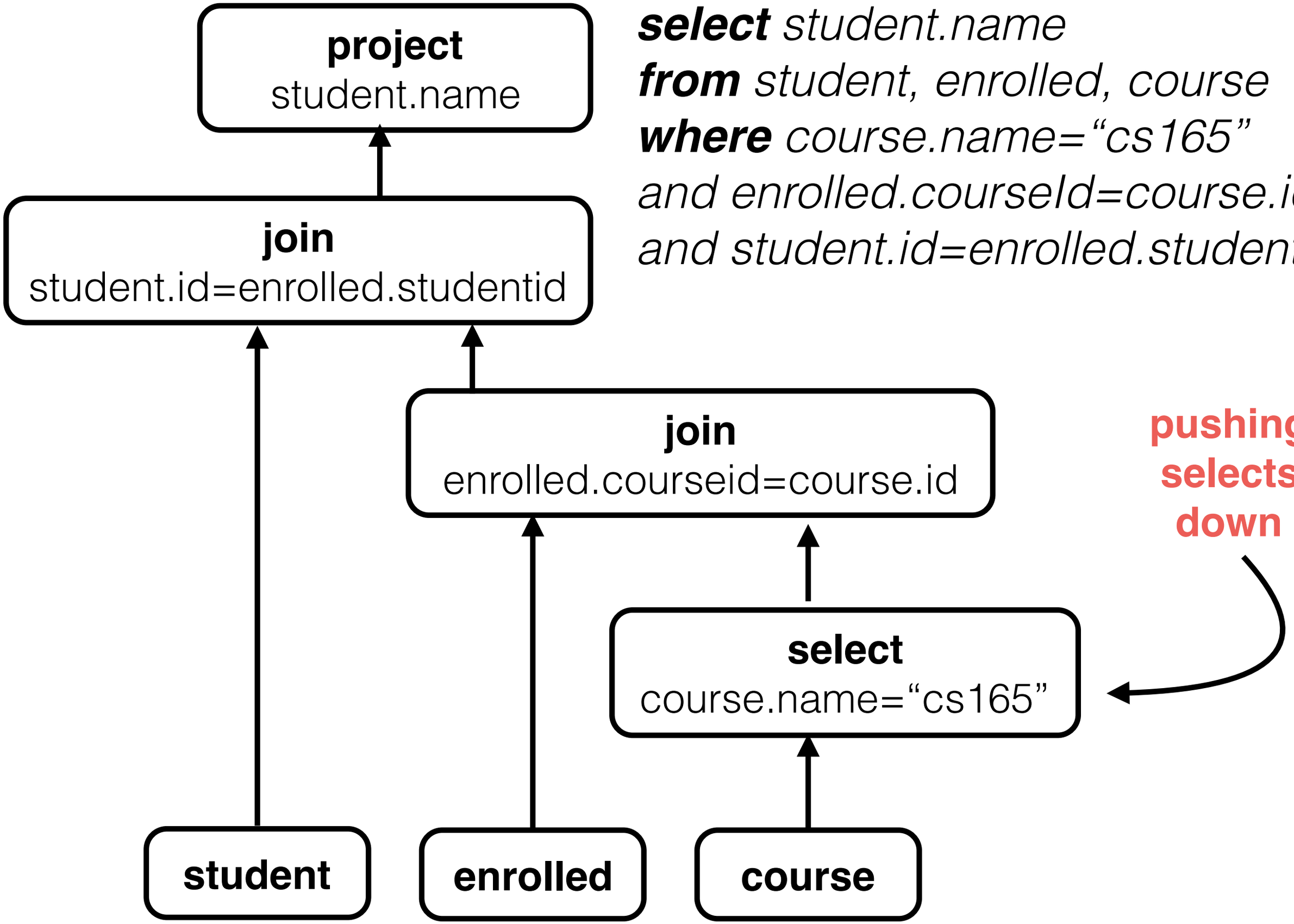
data layout

give me all students enrolled in cs165

select *student.name*
from *student, enrolled, course*
where *course.name=“cs165”*
and enrolled.courseId=course.id
and student.id=enrolled.studentId



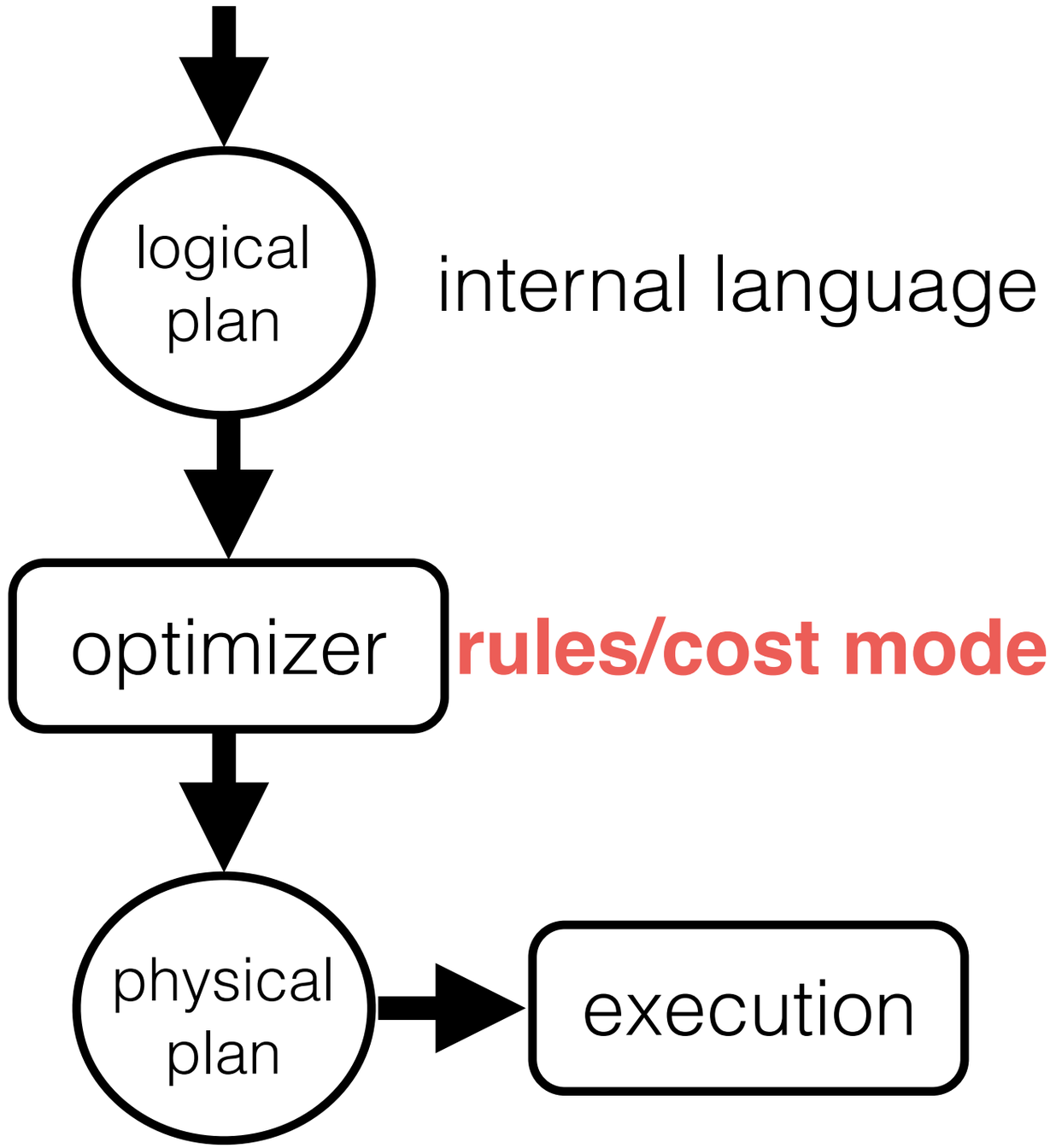
select *student.name*
from *student, enrolled, course*
where *course.name="cs165"*
and enrolled.courseId=course.id
and student.id=enrolled.studentId



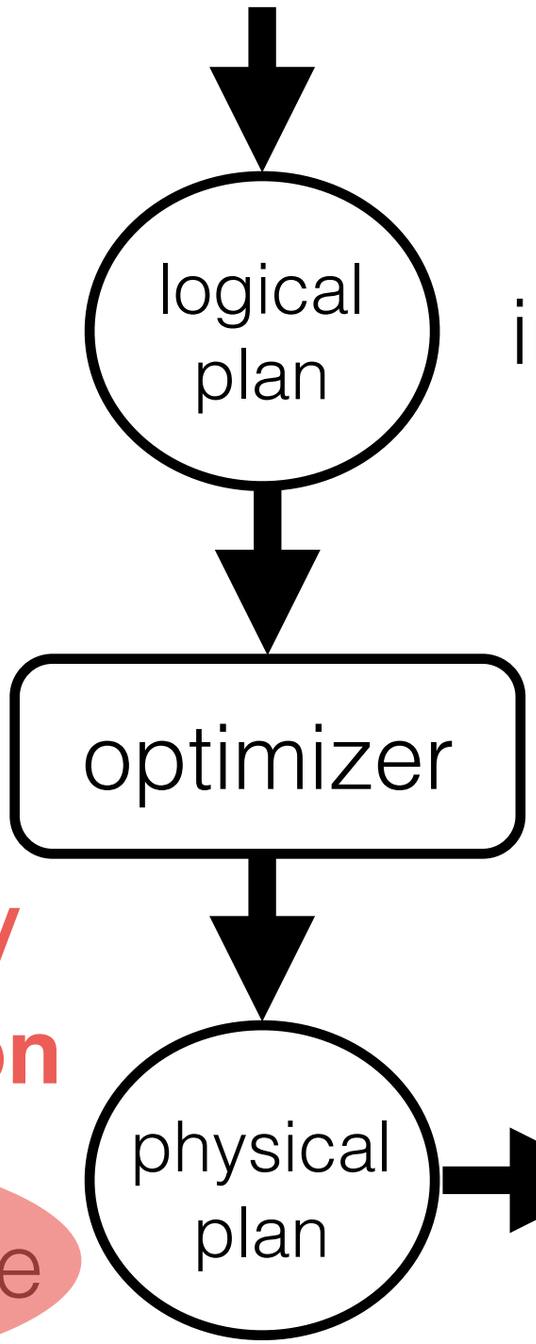
select *student.name*
from *student, enrolled, course*
where *course.name="cs165"*
and enrolled.courseId=course.id
and student.id=enrolled.studentId

pushing selects down

select min(A) **from** R **where** B<10 and C<80



select min(A) **from** R **where** B<10 and C<80

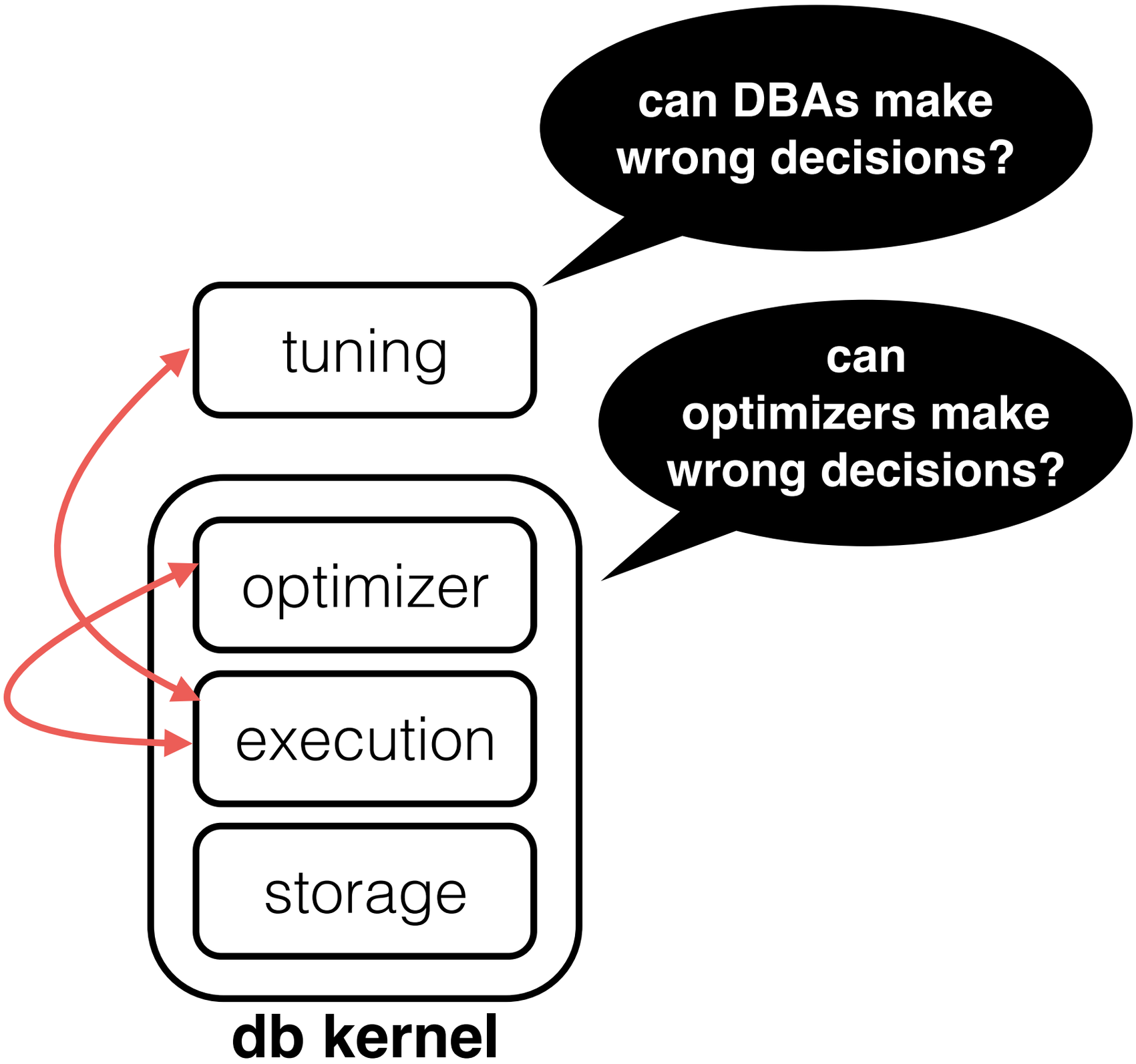


internal language

rules/cost model/statistics

project interface/ level of abstraction

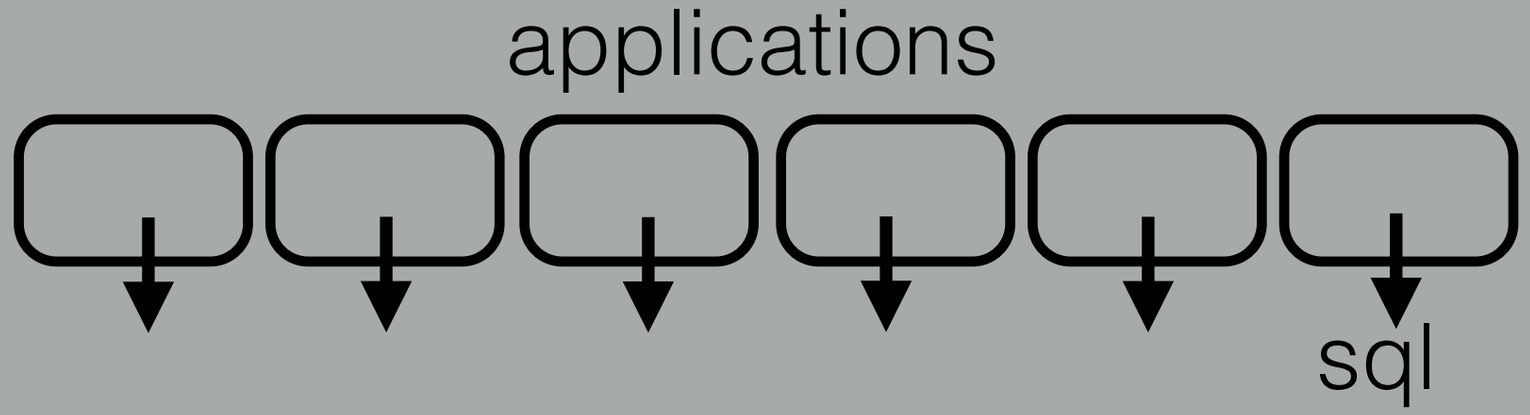
internal language

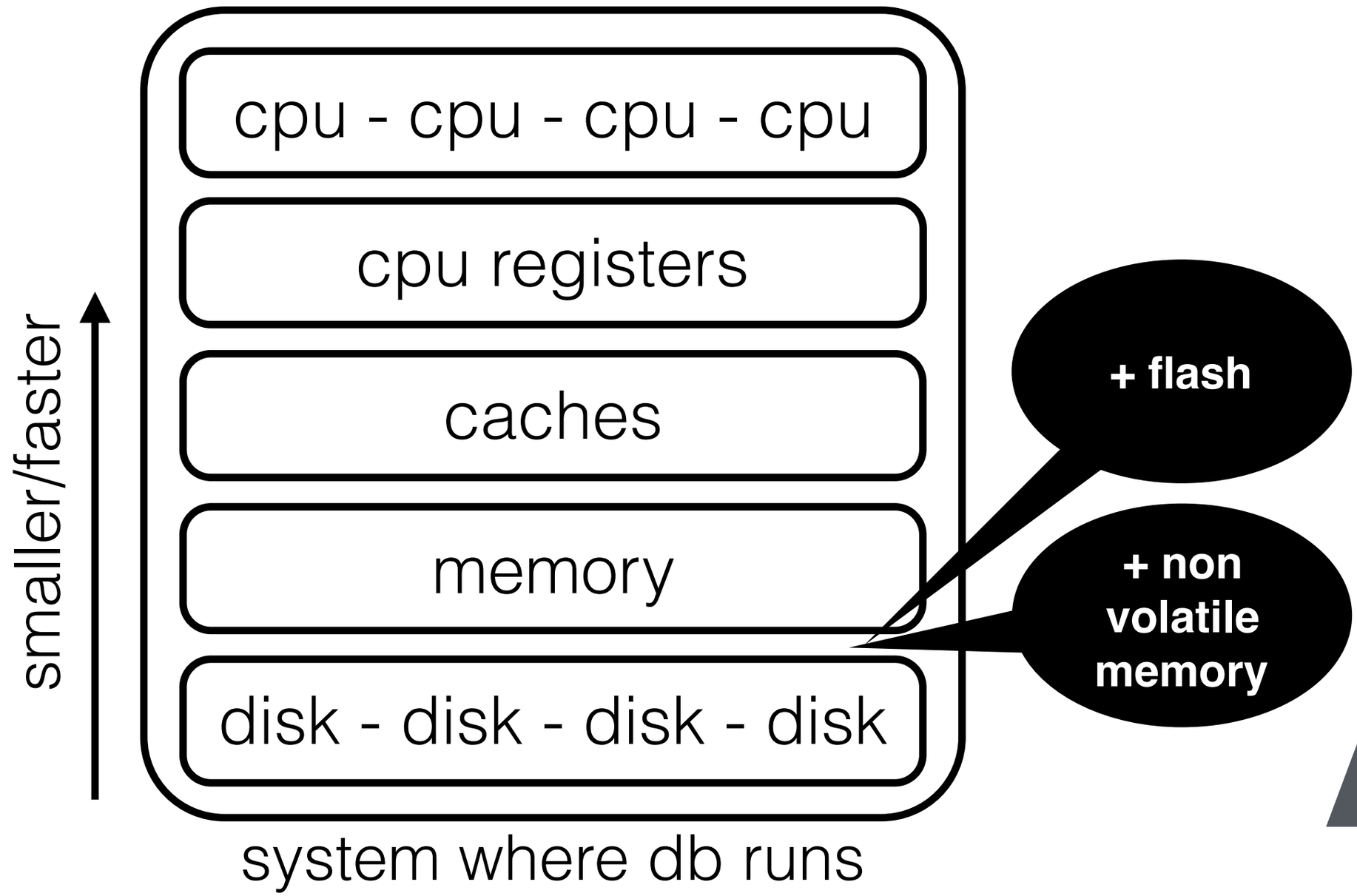

can DBAs make wrong decisions?
tuning
can optimizers make wrong decisions?
optimizer
execution
storage
db kernel

memory hierarchy

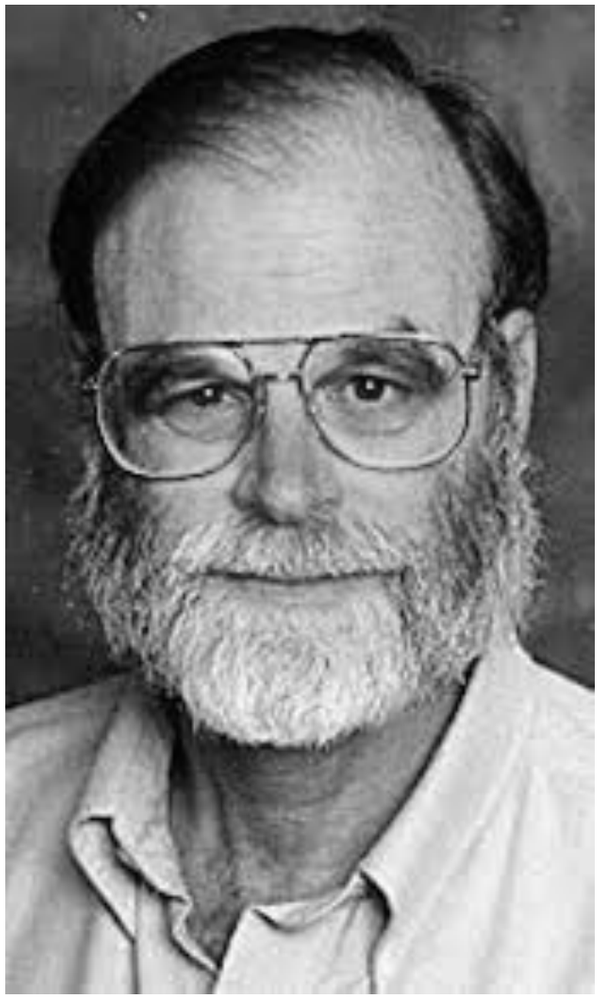
data layouts

column-stores basics


applications
sql

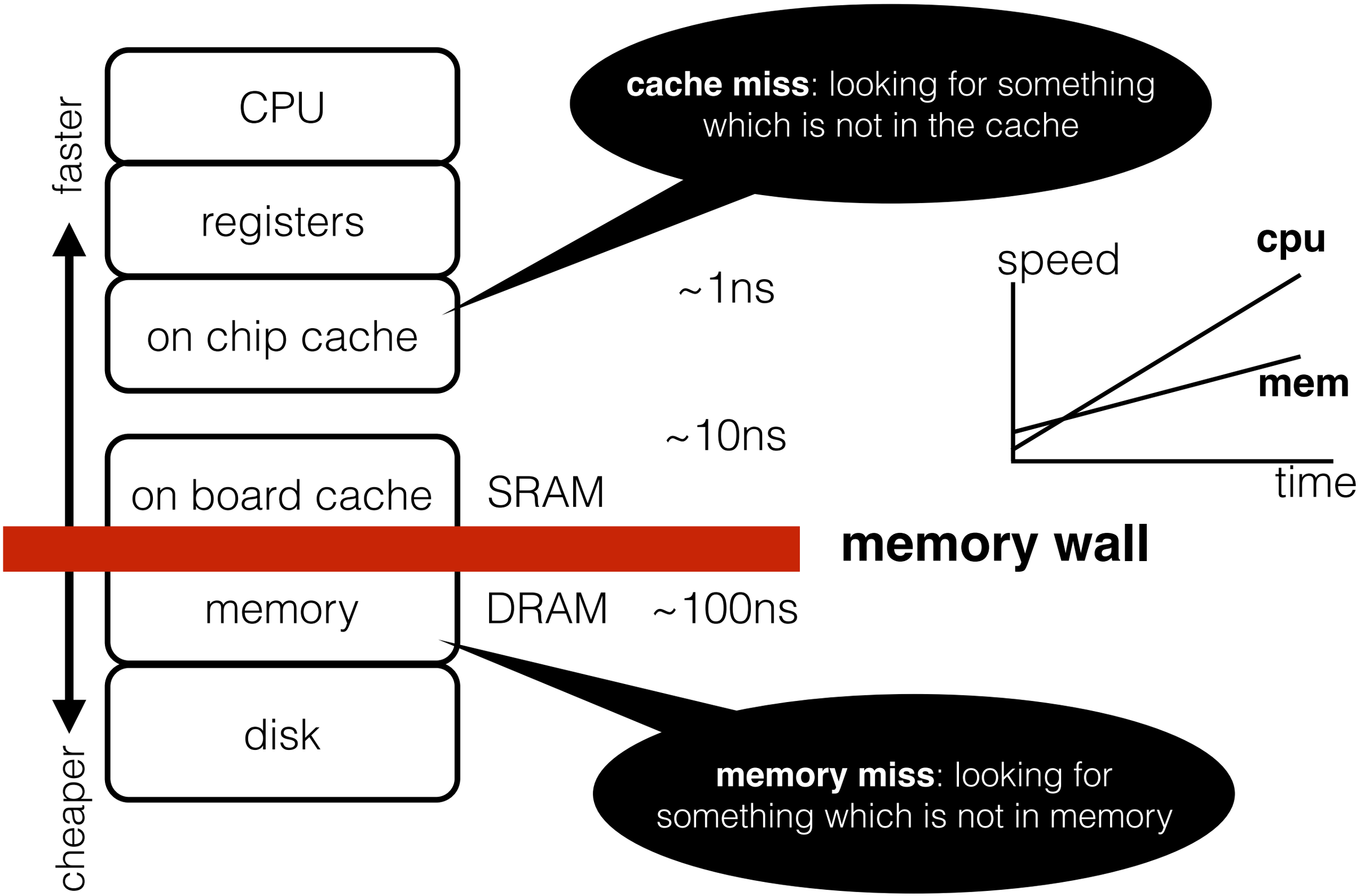

cpu - cpu - cpu - cpu
cpu registers
caches
memory
disk - disk - disk - disk
smaller/faster
system where db runs
+ flash
+ non volatile memory


memory hierarchy

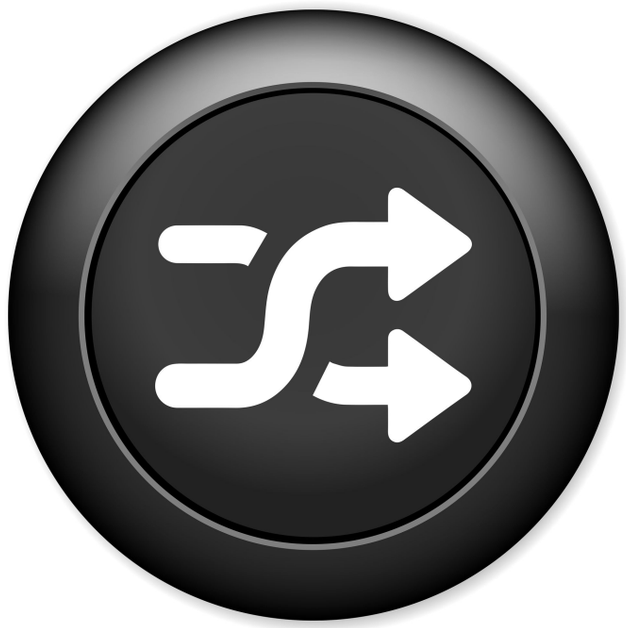


Jim Gray, IBM, Tandem, DEC, Microsoft
ACM Turing award
ACM SIGMOD Edgar F. Codd Innovations Award

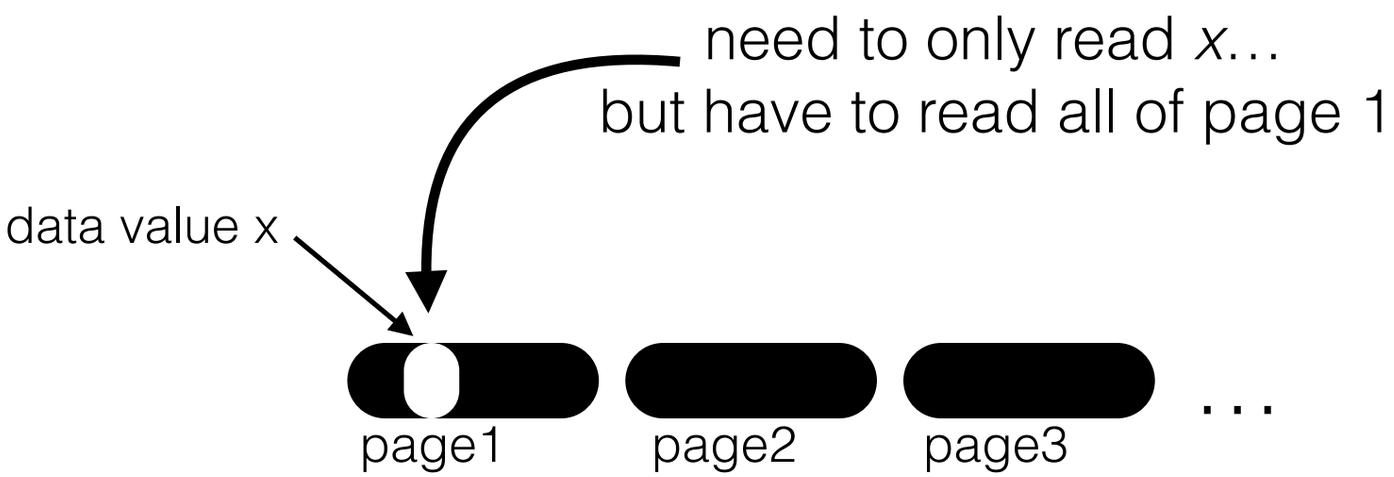
registers	**my head** ~0
2x on chip cache	**this room** 1 min
10x on board cache	**this building** 10 min
100x memory	**New York** 1.5 hours
100Kx disk	**Pluto** 2 years

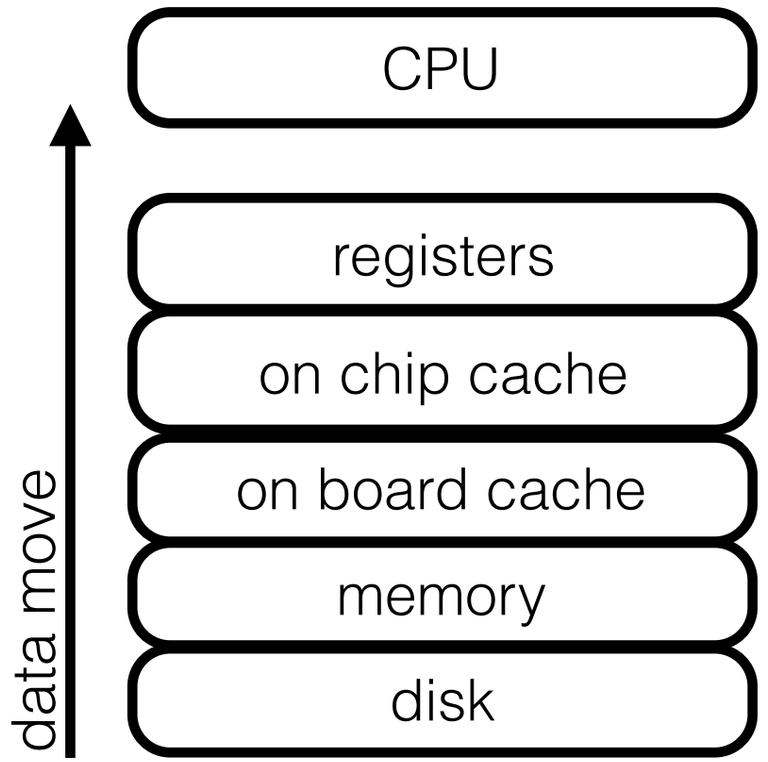

faster
CPU
registers
on chip cache
on board cache
memory
disk
cheaper
~1ns
~10ns
SRAM
DRAM
~100ns
cache miss: looking for something which is not in the cache
memory wall
memory miss: looking for something which is not in memory
speed
cpu
mem
time

design of storage/access methods/algorithms
should minimize: data & instruction misses



page-based access





query x<5

(size=120 bytes)

memory level N

memory level N-1

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

...

page size: 5x8 bytes

query x<5

scan

5 10 6 4 12

(size=120 bytes)

memory level N

memory level N-1

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

...

page size: 5x8 bytes

query x<5

scan

5 10 6 4 12

(size=120 bytes)

4

memory level N

memory level N-1

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

...

page size: 5x8 bytes

40 bytes

query x<5

scan

5 10 6 4 12

(size=120 bytes)

4

memory level N

memory level N-1

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

...

page size: 5x8 bytes

40 bytes

query x<5

scan

scan

5 10 6 4 12

2 8 9 7 6

4

(size=120 bytes)

memory level N

memory level N-1

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

...

page size: 5x8 bytes

40 bytes

query x<5

scan

scan

5 10 6 4 12

2 8 9 7 6

4 2

(size=120 bytes)

memory level N

memory level N-1

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

...

page size: 5x8 bytes

80 bytes

query x<5

scan

scan

5 10 6 4 12

2 8 9 7 6

4 2

(size=120 bytes)

memory level N

memory level N-1

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

...

page size: 5x8 bytes

80 bytes

query x<5

scan

(size=120 bytes)

7 11 3 9 6

2 8 9 7 6

4 2

memory level N

memory level N-1

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

...

page size: 5x8 bytes

80 bytes

query x<5

scan

(size=120 bytes)

7 11 3 9 6

2 8 9 7 6

4 2 3

memory level N

memory level N-1

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

...

page size: 5x8 bytes

120 bytes

query x<5

scan

(size=120 bytes)

7 11 3 9 6

2 8 9 7 6

4 2 3

memory level N

memory level N-1

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

...

page size: 5x8 bytes

an oracle gives us the positions

query x<5

(size=120 bytes)

memory level N

memory level N-1

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

...

page size: 5x8 bytes

an oracle gives us the positions

query x<5

oracle

5 10 6 4 12

(size=120 bytes)

memory level N

memory level N-1

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

...

page size: 5x8 bytes

an oracle gives us the positions

query x<5

oracle

5 10 6 4 12

4

(size=120 bytes)

memory level N

memory level N-1

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

...

page size: 5x8 bytes

an oracle gives us the positions

40 bytes

query x<5

oracle

5 10 6 4 12

(size=120 bytes)

4

memory level N

memory level N-1

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

...

page size: 5x8 bytes

an oracle gives us the positions

40 bytes

query x<5

oracle → 5 10 6 4 12

oracle → 2 8 9 7 6

4

(size=120 bytes)

memory level N

memory level N-1

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

...

page size: 5x8 bytes

an oracle gives us the positions

40 bytes

query x<5

oracle → 5 10 6 4 12

oracle → 2 8 9 7 6

4 2

(size=120 bytes)

memory level N

memory level N-1

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

...

page size: 5x8 bytes

an oracle gives us the positions

80 bytes

query x<5

oracle → 5 10 6 4 12

oracle → 2 8 9 7 6

4 2

(size=120 bytes)

memory level N

memory level N-1

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

...

page size: 5x8 bytes

an oracle gives us the positions

80 bytes

query x<5

oracle

(size=120 bytes)

7 11 3 9 6

2 8 9 7 6

4 2

memory level N

memory level N-1

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

...

page size: 5x8 bytes

an oracle gives us the positions

80 bytes

query x<5

oracle

(size=120 bytes)

7 11 3 9 6

2 8 9 7 6

4 2 3

memory level N

memory level N-1

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

...

page size: 5x8 bytes

an oracle gives us the positions

120 bytes

query x<5

oracle

(size=120 bytes)

7 11 3 9 6

2 8 9 7 6

4 2 3

memory level N

memory level N-1

5 10 6 4 12

2 8 9 7 6

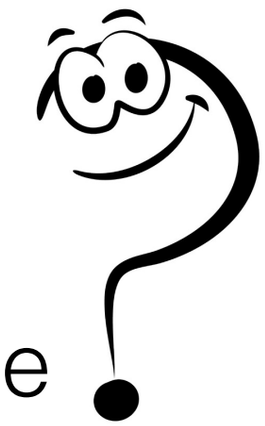
7 11 3 9 6

...

page size: 5x8 bytes

scan=120bytes vs oracle=120bytes

(and there is no such thing as an Oracle so Oracle is not for free…)

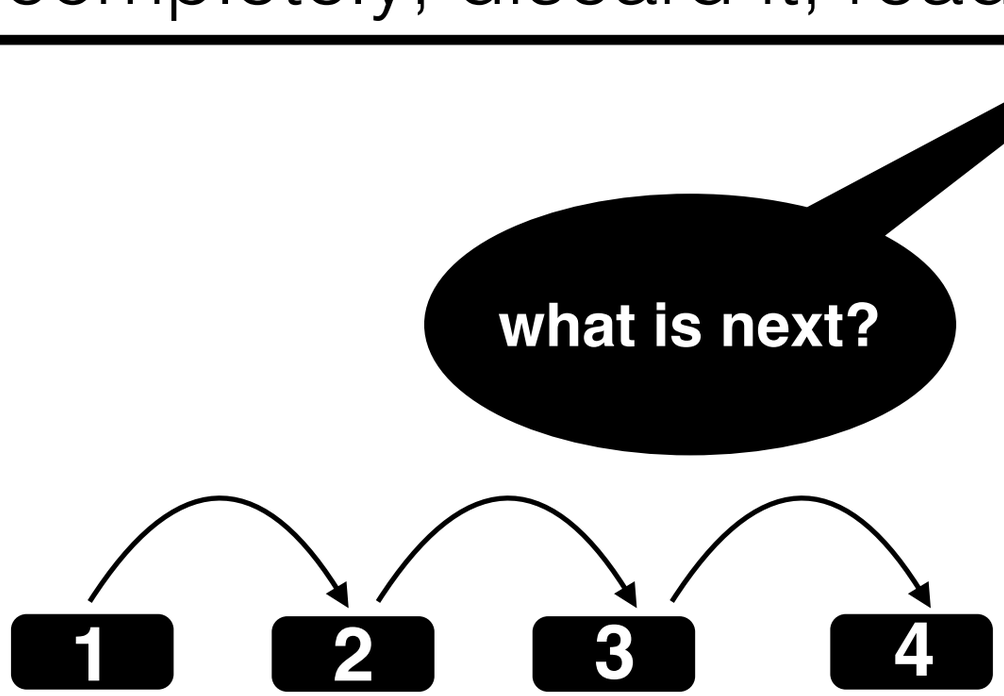


when does it make sense to have an oracle

sequential access:

read one block; consume it completely; discard it; read **next**

in parallel/prefetching



hardware/software can better predict/buffer sequential pages to be read

random access:
read one block; consume it partially; discard it;
1) read **“random” next**
2) might have to read it **again** in the future

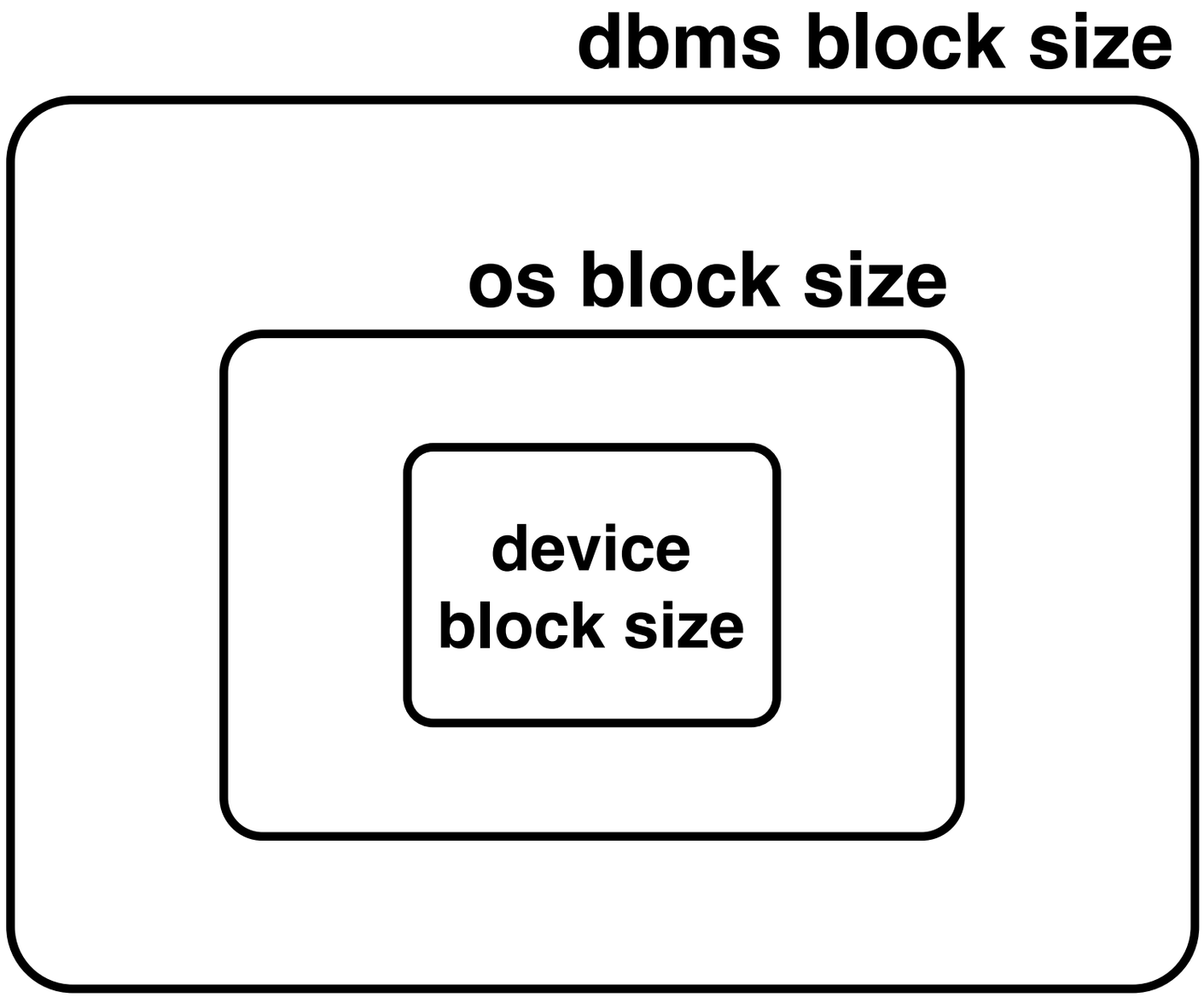
level N

buffer pool

remember **hot** blocks

level N-1

why not use OS caching

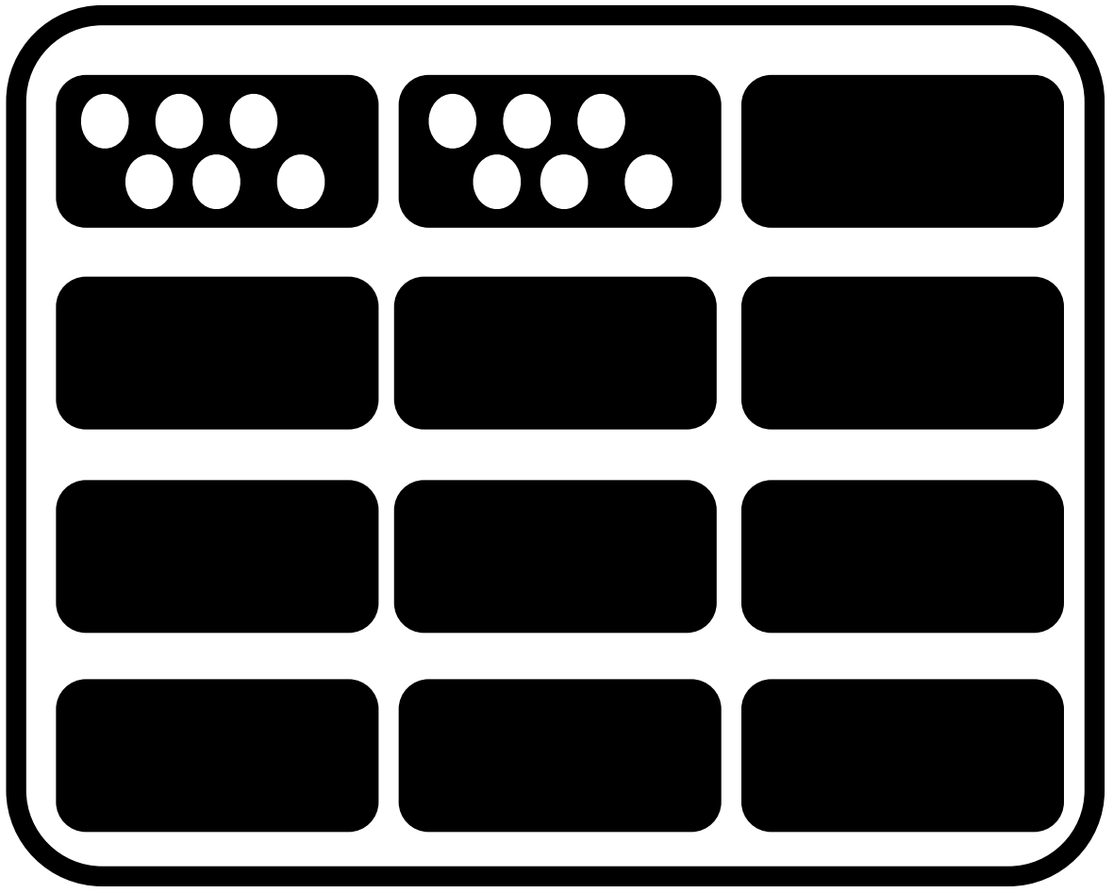


os and db will typically refer to **pages**
hardware vendors might also refer to **sectors**

employee
(id:int, name:varchar(50), office:char(5),
telephone:char(10), city:varchar(30), salary:int)

(1, name1, office1, tel1, city1, salary1)
(2, name2, office2, tel2, city2, salary2)
(3, name3, office3, tel3, city3, salary3)
(4, name4, office4, tel4, city4, salary4)
(5, name5, office5, tel5, city5, salary5)
(6, name6, office6, tel6, city6, salary6)
(7, name7, office7, tel7, city7, salary7)
(8, name8, office8, tel8, city8, salary8)
(9, name9, office9, tel9, city9, salary9)
…

data storage
blocks < pages < files

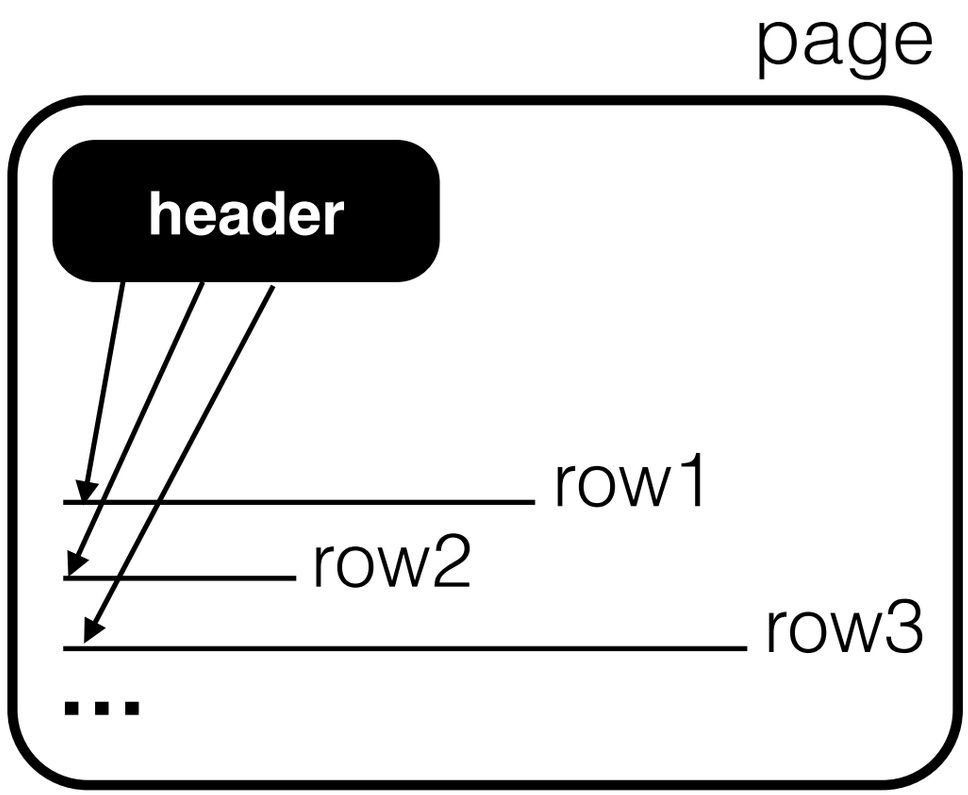


file

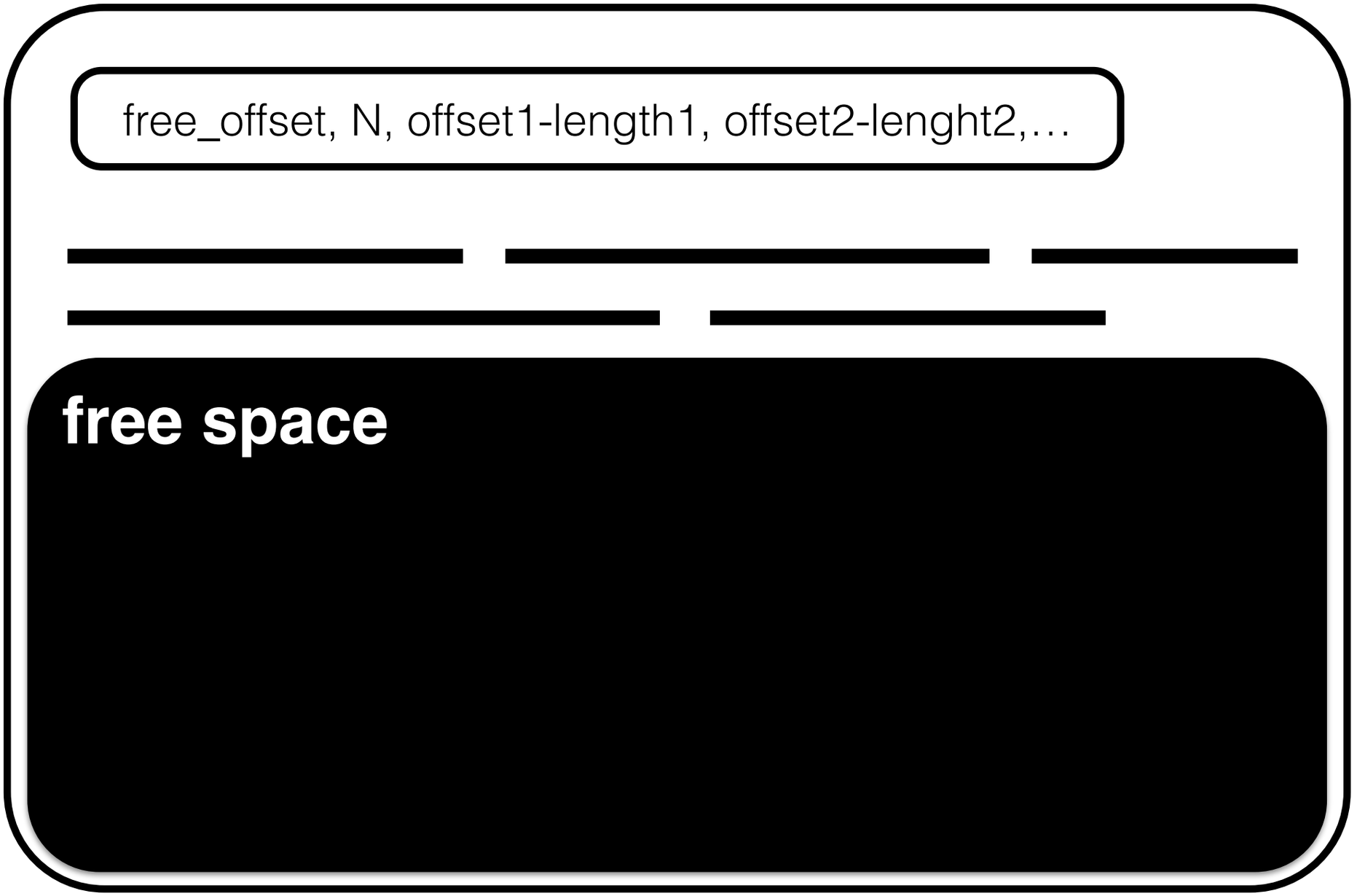
remember: the way we store data defines the best possible way we can access it

employee
(id:int, name:varchar(50), office:char(5),
telephone:char(10), city:varchar(30), salary:int)

(1, name1, office1, tel1, city1, salary1)
(2, name2, office2, tel2, city2, salary2)
(3, name3, office3, tel3, city3, salary3)
(4, name4, office4, tel4, city4, salary4)
(5, name5, office5, tel5, city5, salary5)
(6, name6, office6, tel6, city6, salary6)
(7, name7, office7, tel7, city7, salary7)
(8, name8, office8, tel8, city8, salary8)
(9, name9, office9, tel9, city9, salary9)
…



slotted page

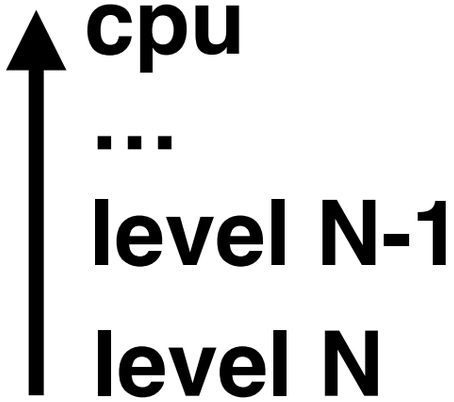


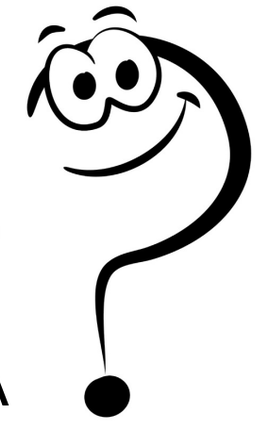
scan
null
update
var length
…

things to “worry” about

how much data we transfer through the memory hierarchy

how many computations we do

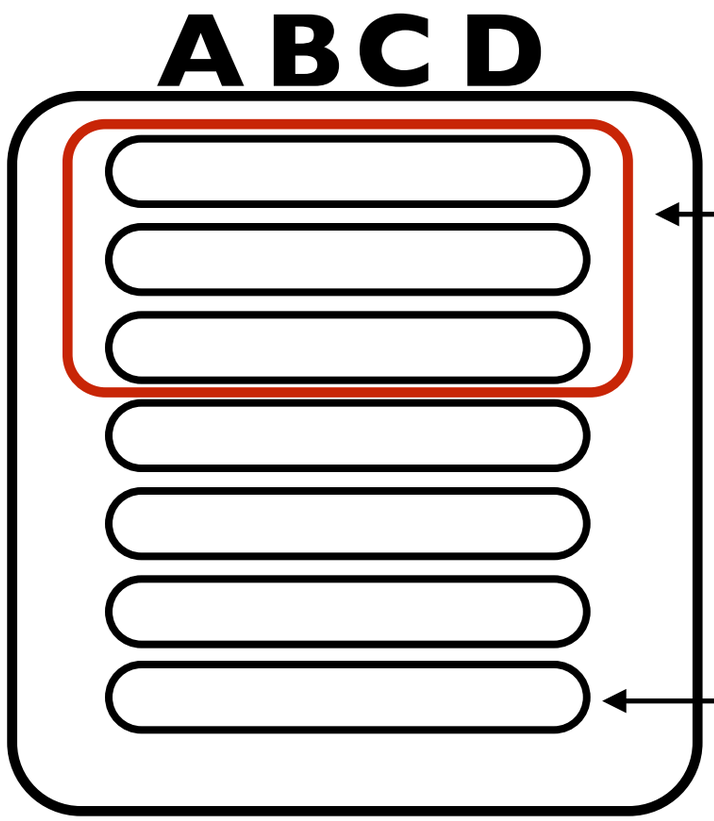




select A,B,C,D

select A

row-store



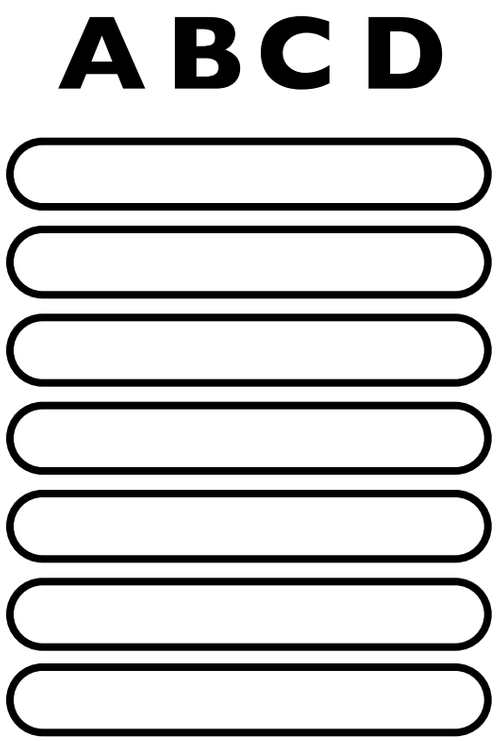
each page contains all fields for each entry
(i.e., all attributes values)

stored continuously

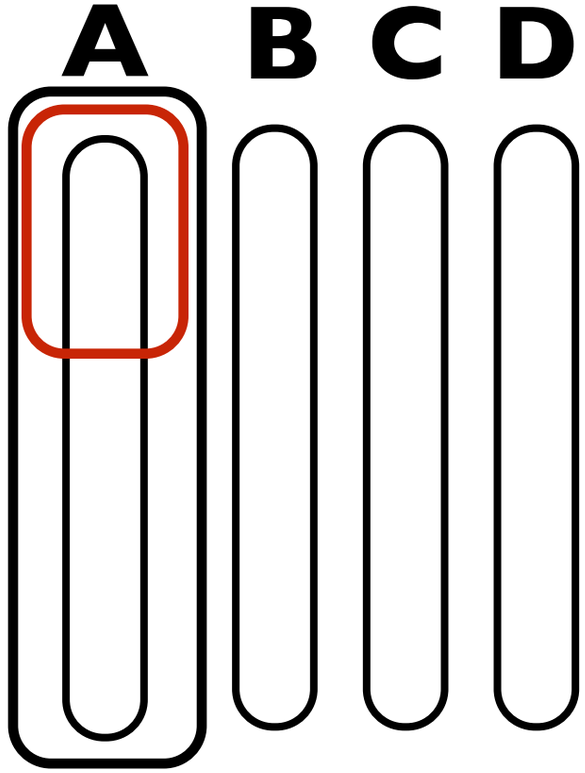
select A,B,C,D

select A

row-store

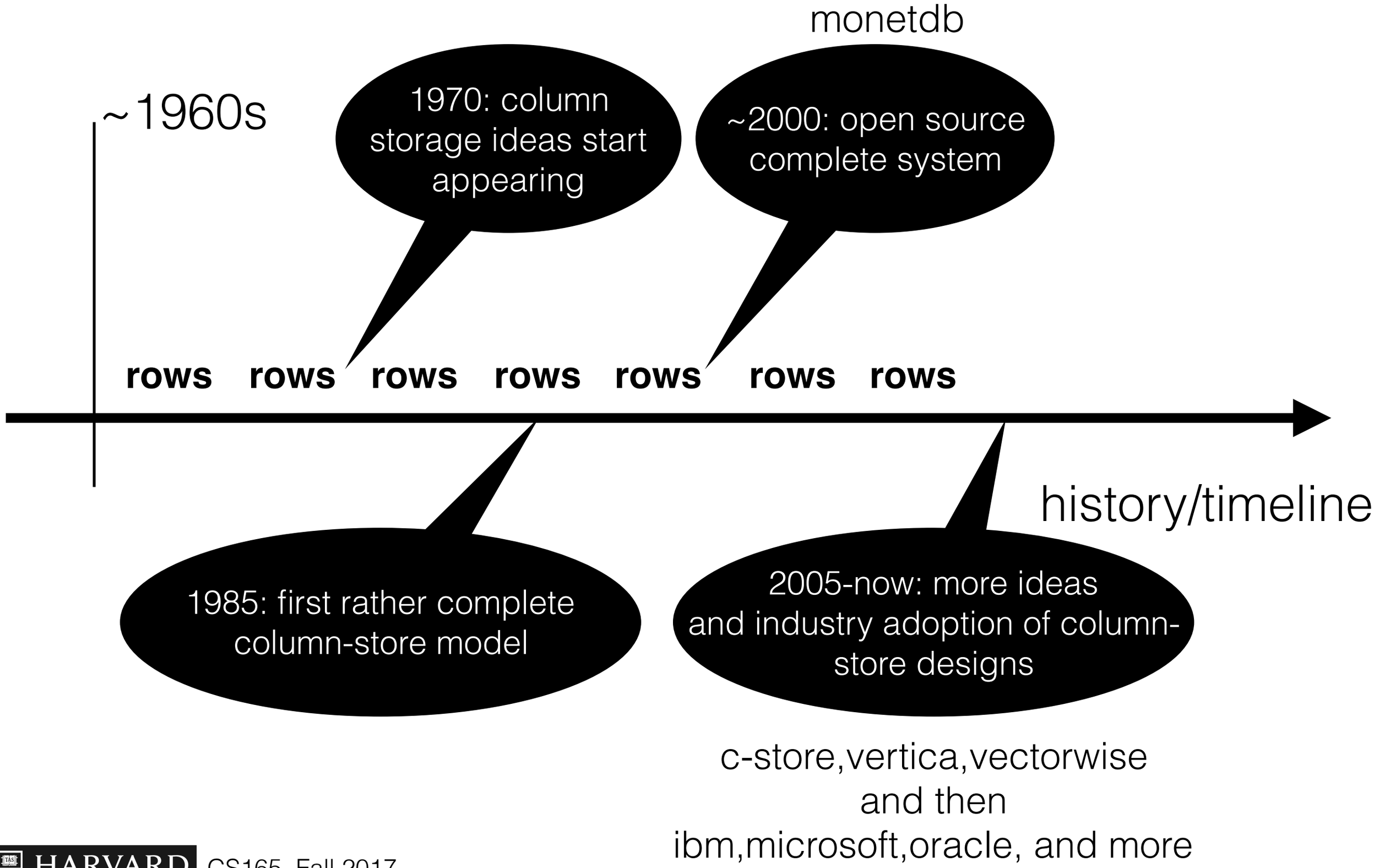


column-store

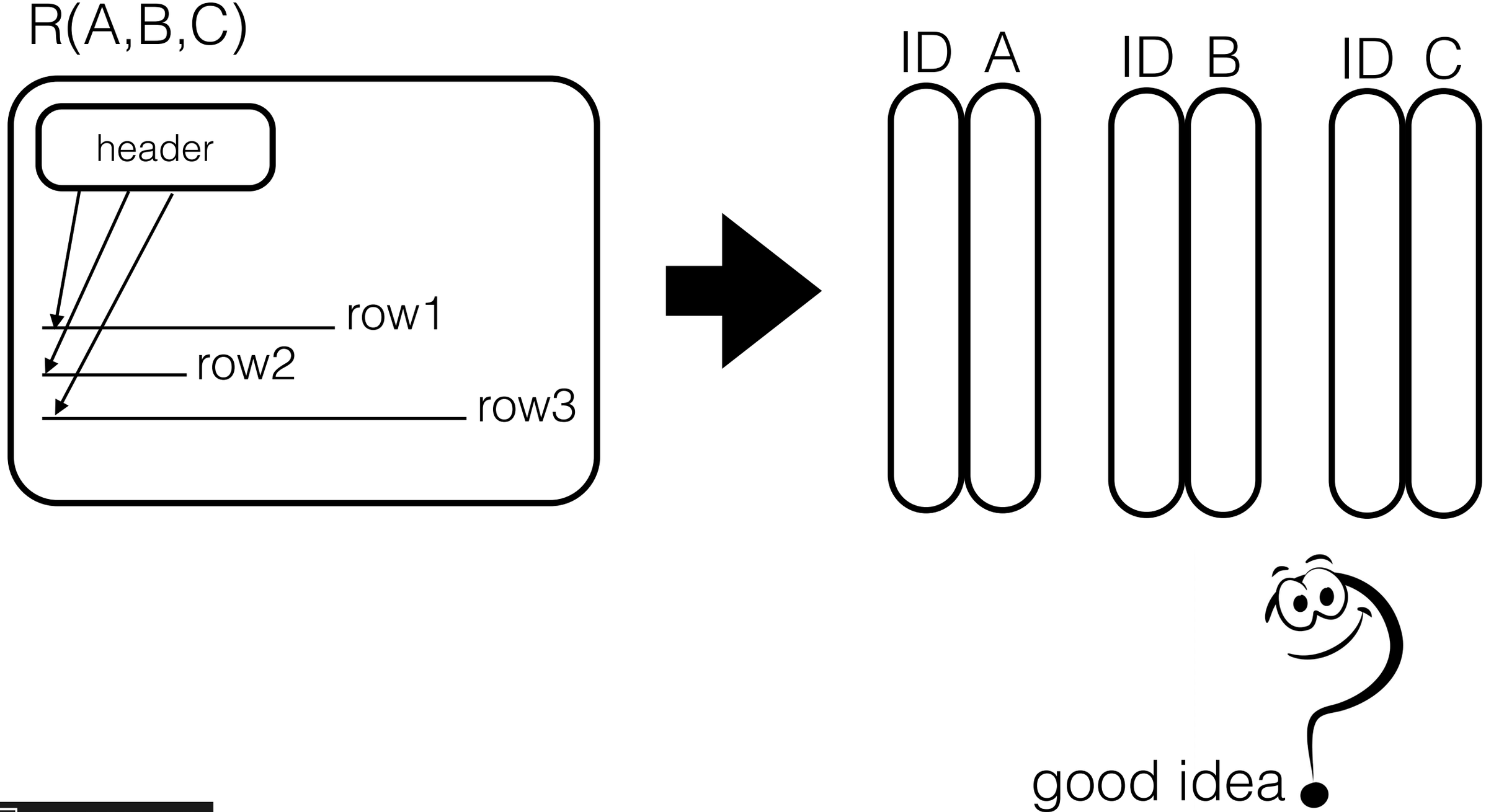


each page contains fields of a single attribute

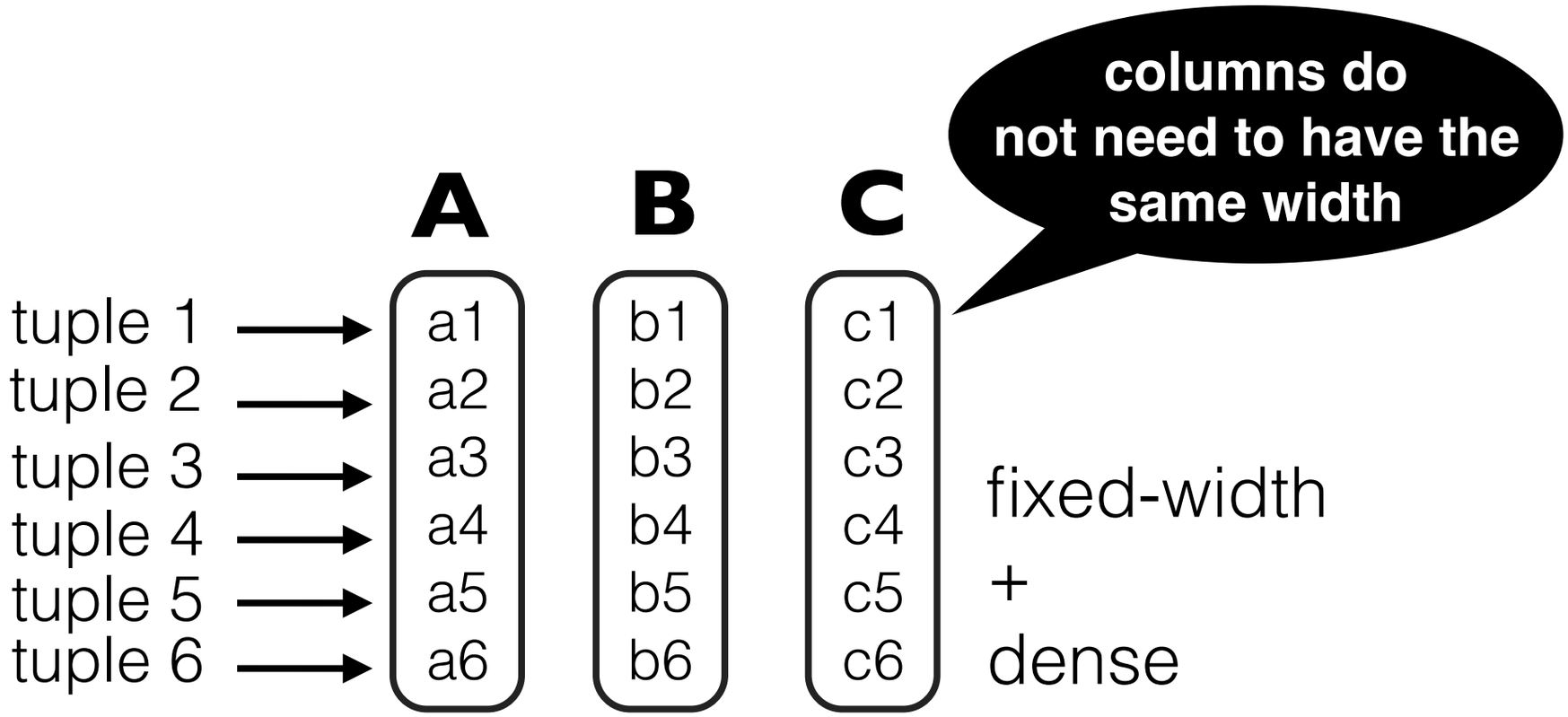
stored continuously


~1960s
1970: column storage ideas start appearing
monetdb
~2000: open source complete system
rows rows rows rows rows rows rows
history/timeline
1985: first rather complete column-store model
2005-now: more ideas and industry adoption of column-store designs
c-store,vertica,vectorwise
and then
ibm,microsoft,oracle, and more

column-store with
materialized IDs



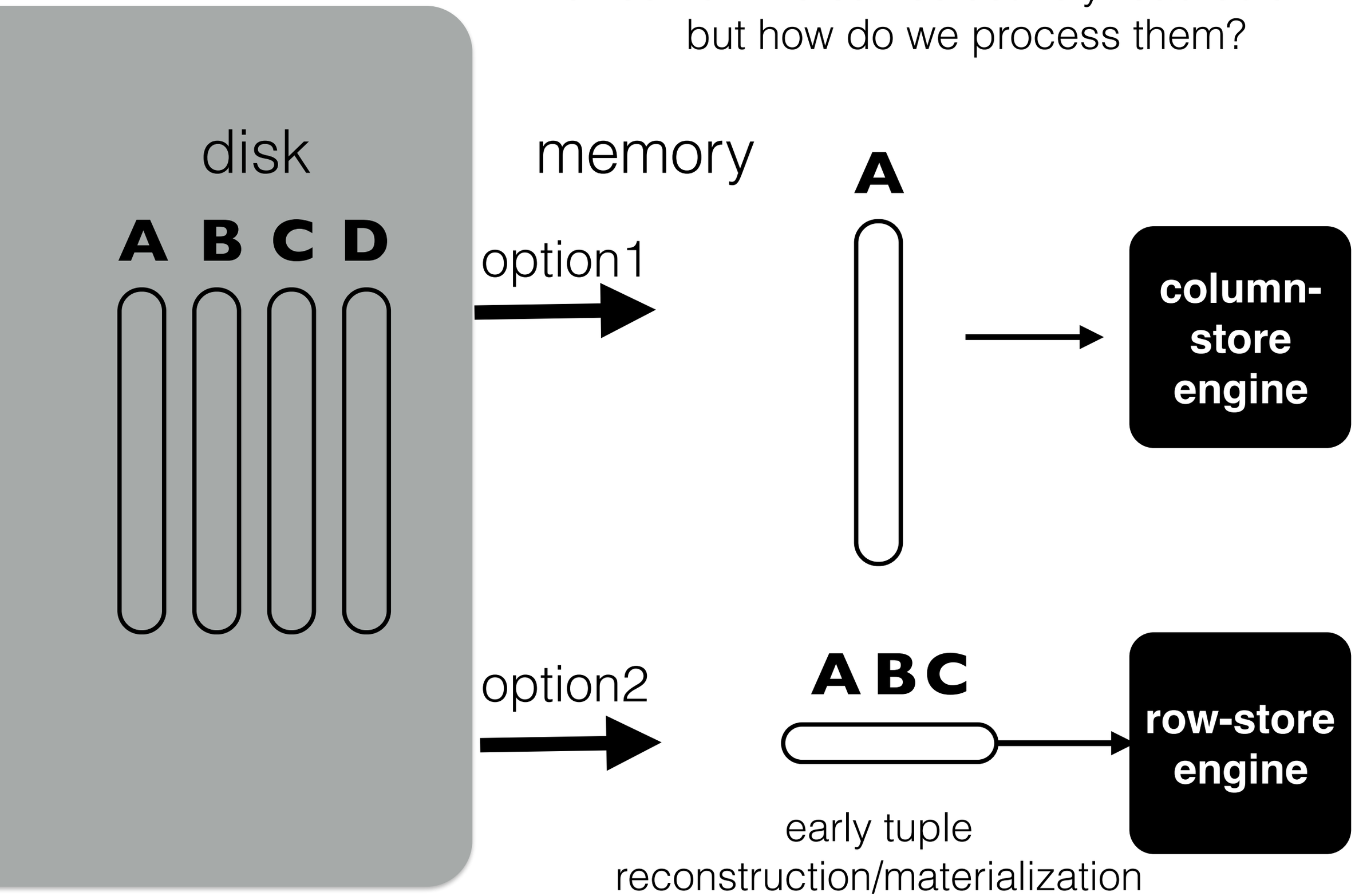
virtual ids/ positional alignment

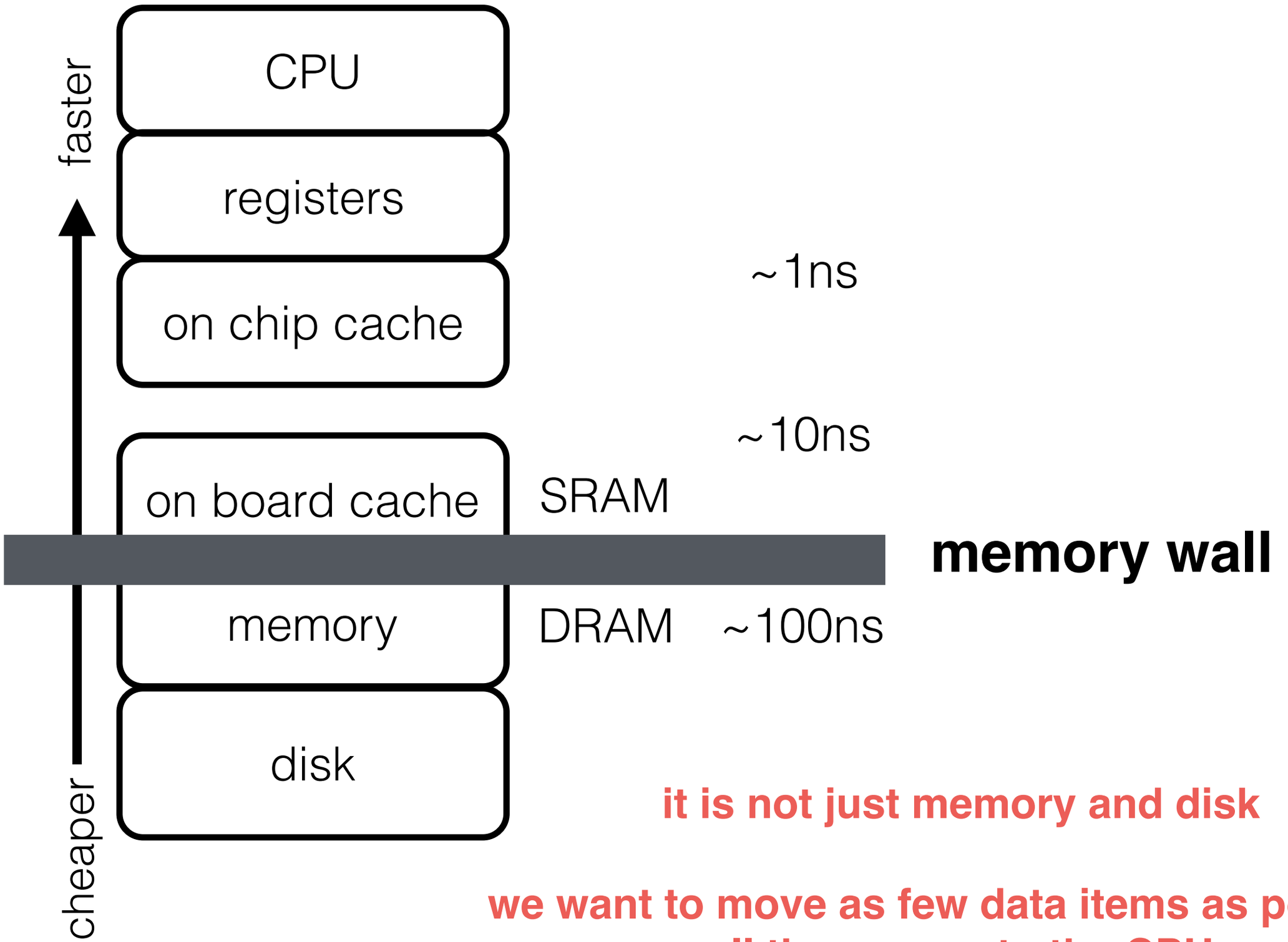


positional lookups/joins

$$A(i) = A + i * width(A)$$

ok so now we can selectively read columns
but how do we process them?

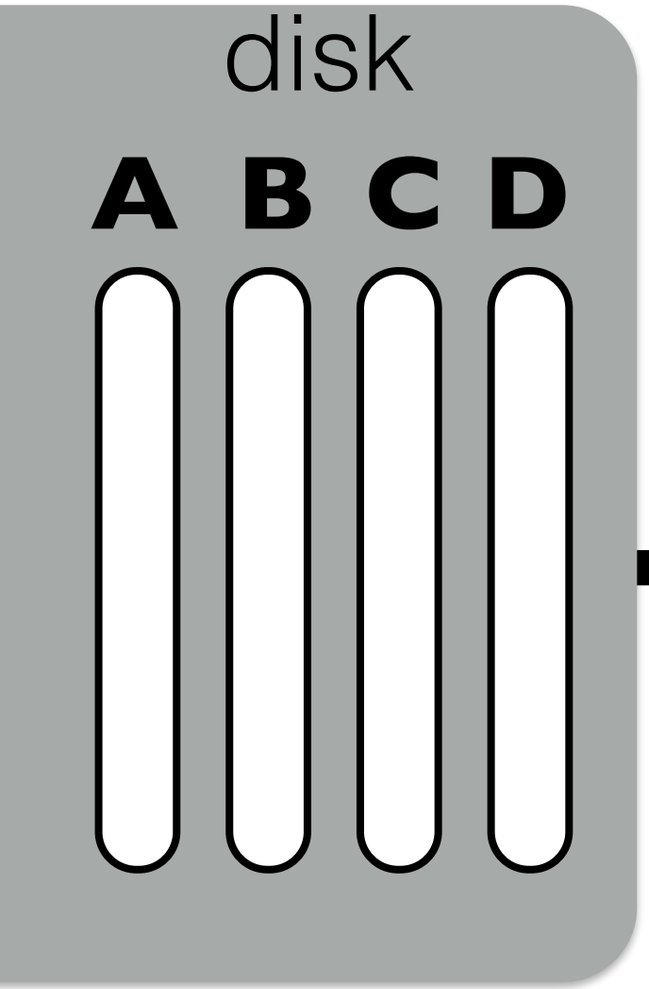




it is not just memory and disk

we want to move as few data items as possible all the way up to the CPU

select min(C) **from** R **where** A<10 & B<20



memory

write the query plan
and the code/logic of each operator

do not forget about intermediate results
describe data layouts at each step

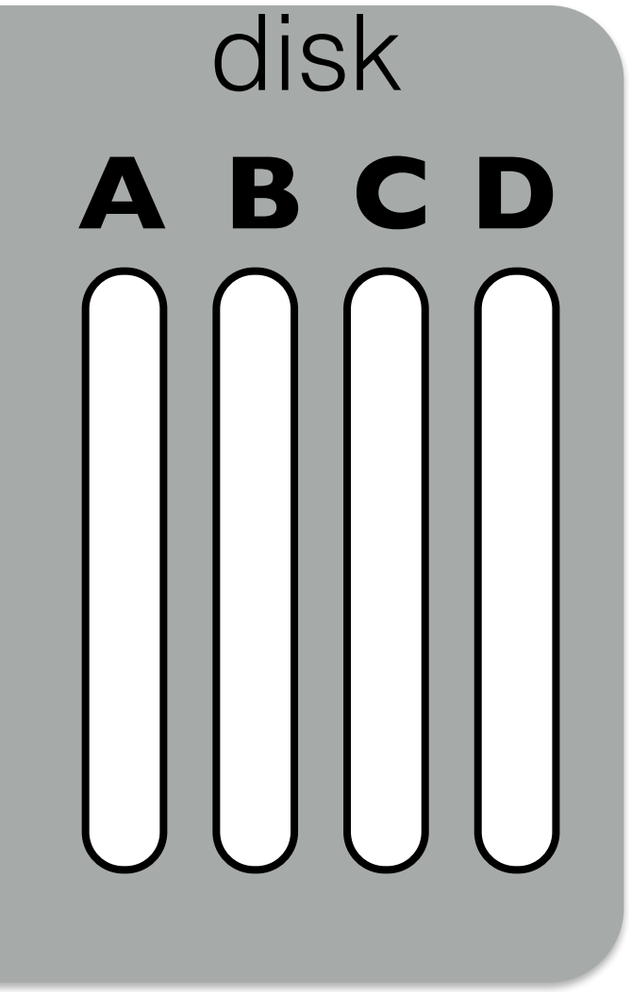
(milestone1 of project)

no precise final answer is OK
understanding what matters is key
concepts & designs will be repeated >>1

late reconstruction/materialization

select min(C) **from** R **where** A<10 & B<20

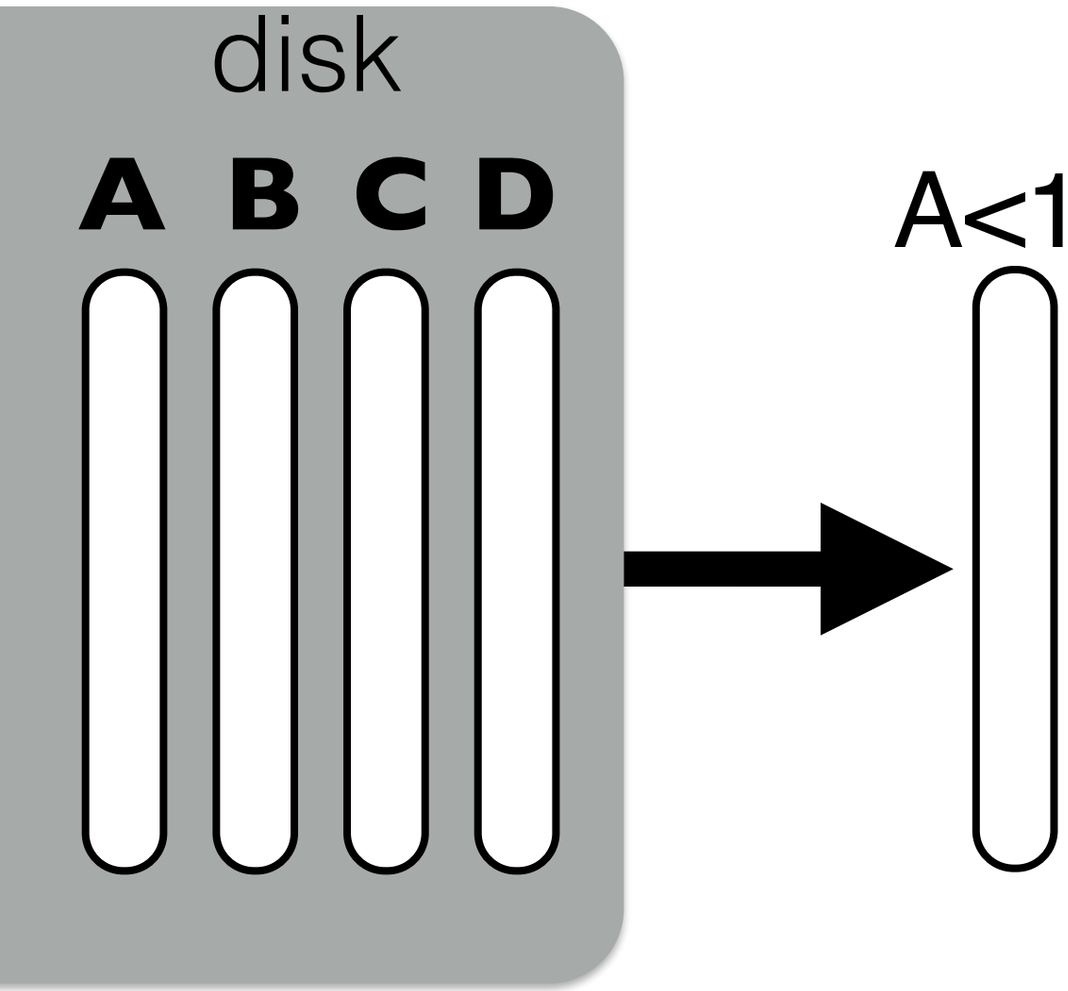
disk



memory

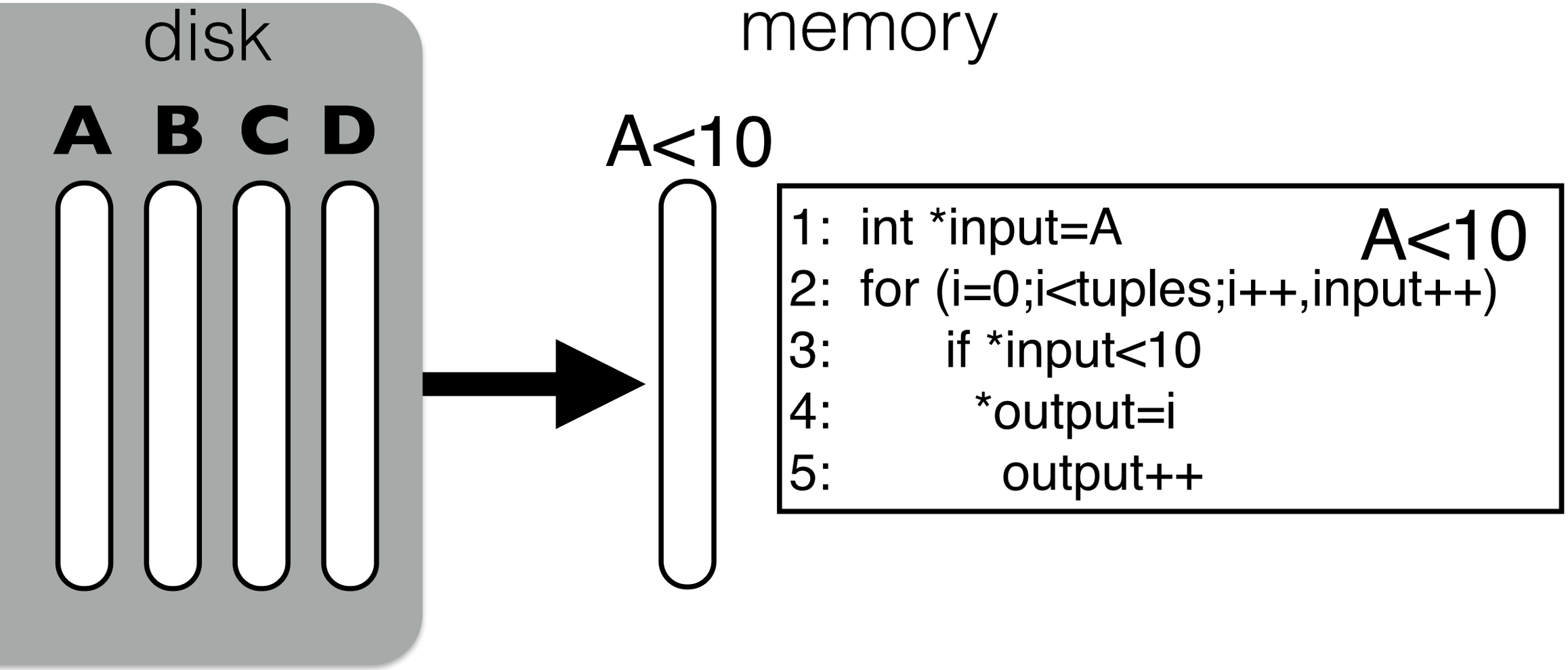
late reconstruction/materialization

select min(C) **from** R **where** A<10 & B<20



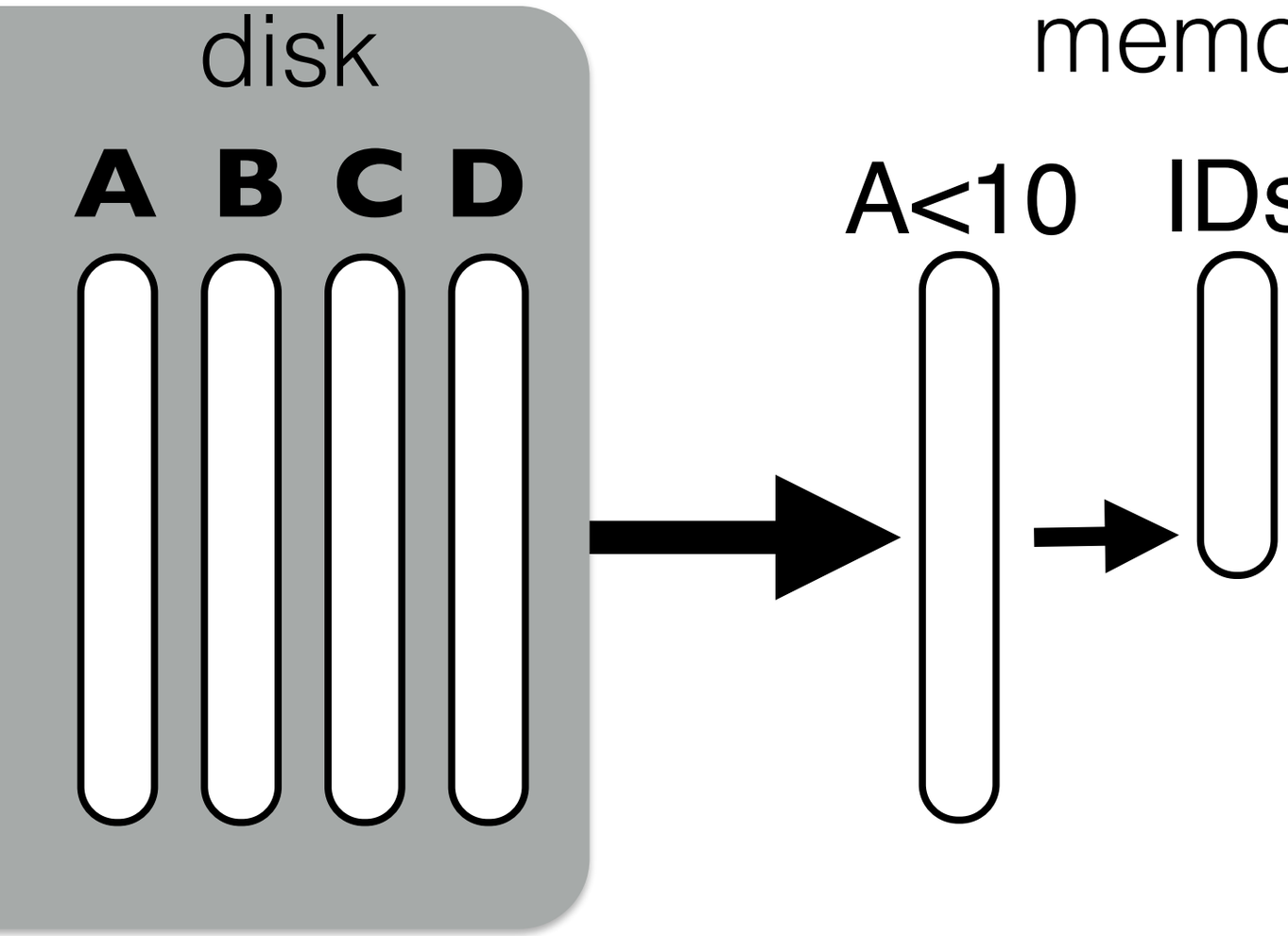
late reconstruction/materialization

select min(C) **from** R **where** A<10 & B<20



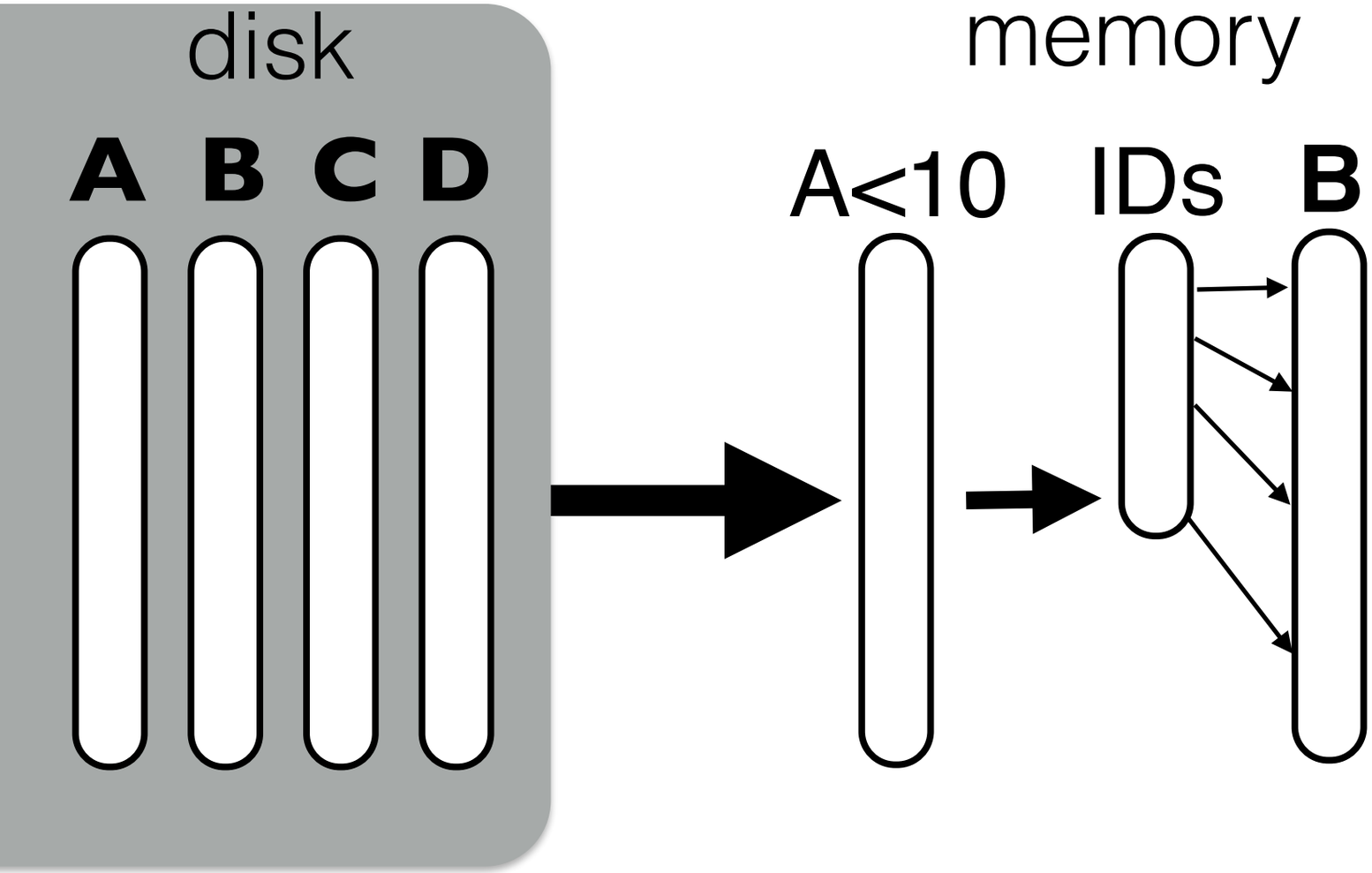
late reconstruction/materialization

select min(C) **from** R **where** A<10 & B<20



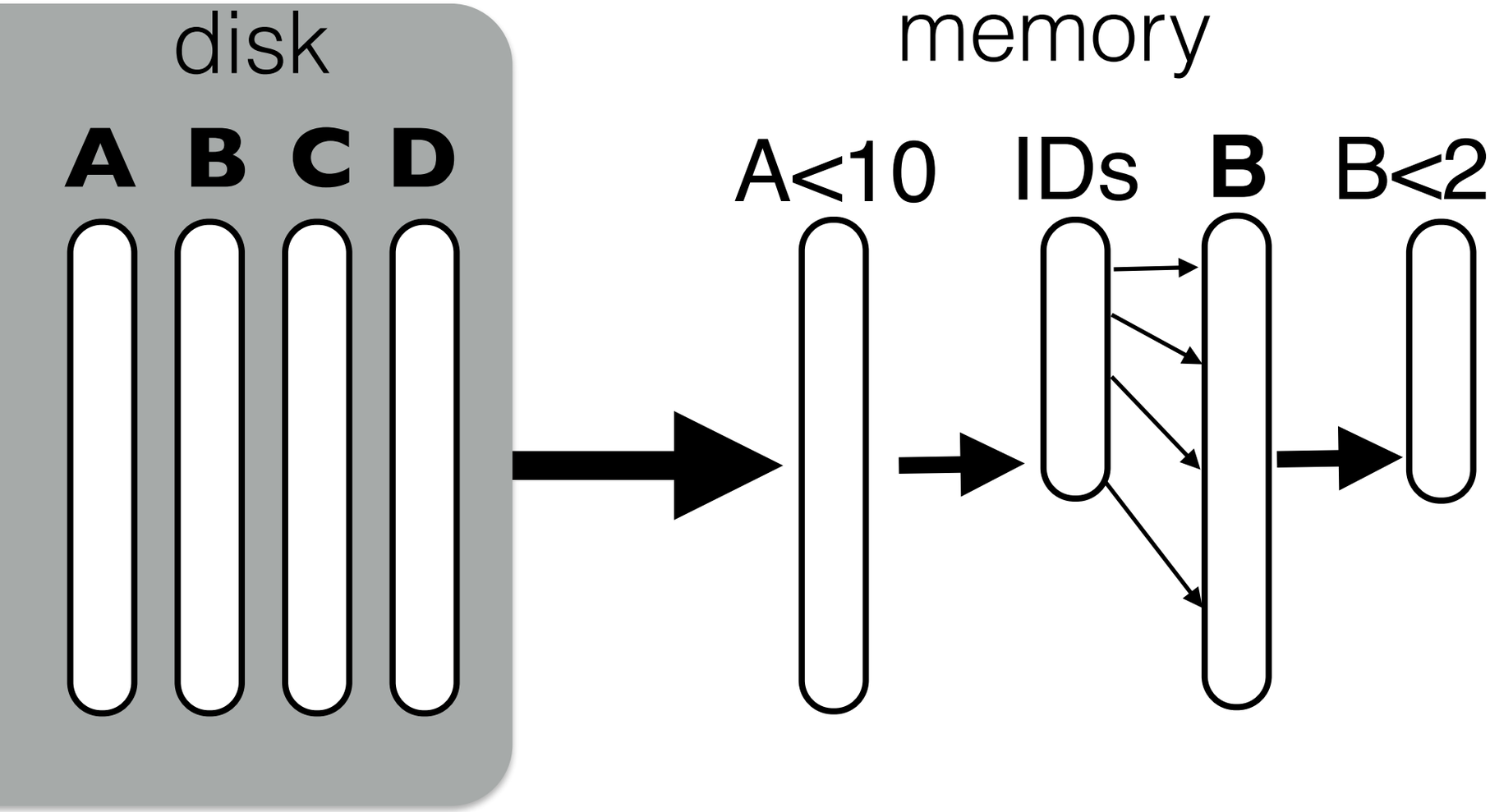
late reconstruction/materialization

select min(C) **from** R **where** A<10 & B<20



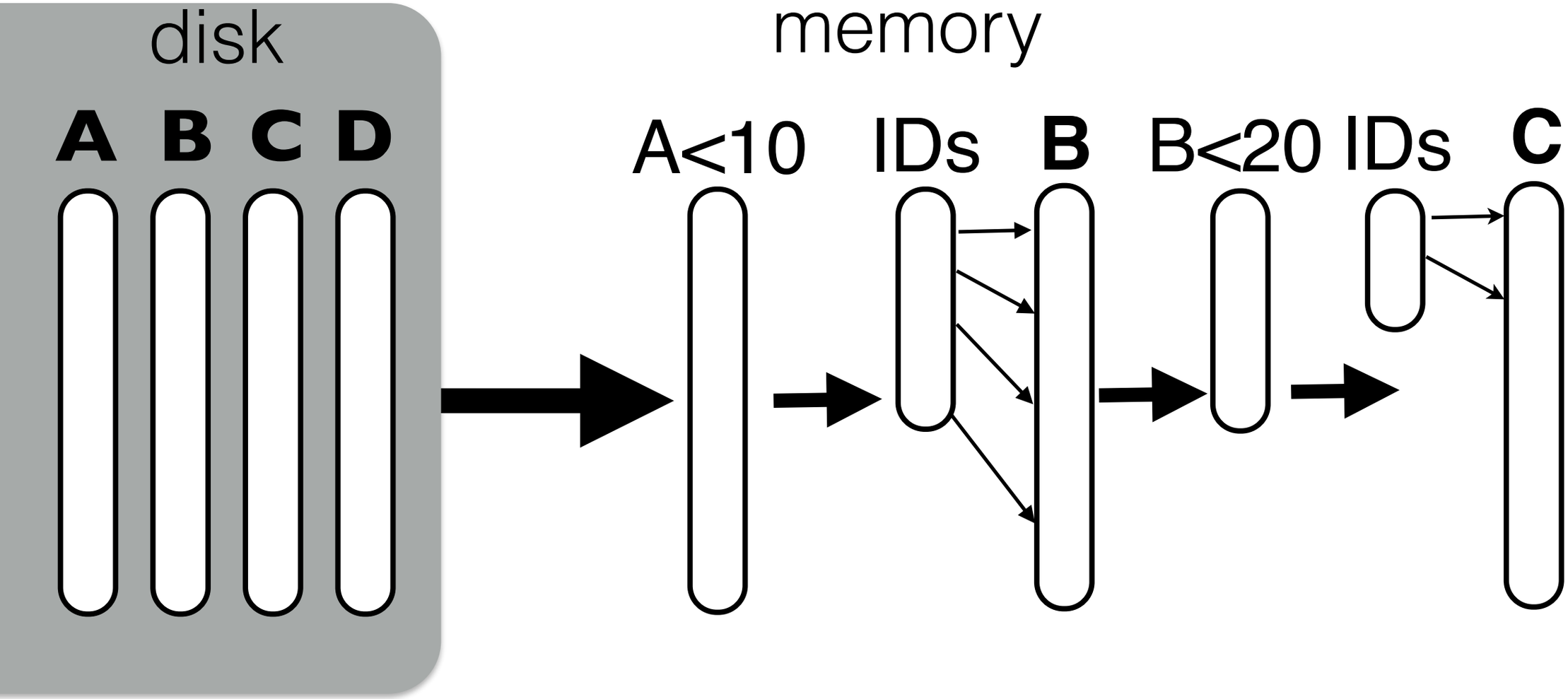
late reconstruction/materialization

select min(C) **from** R **where** A<10 & B<20



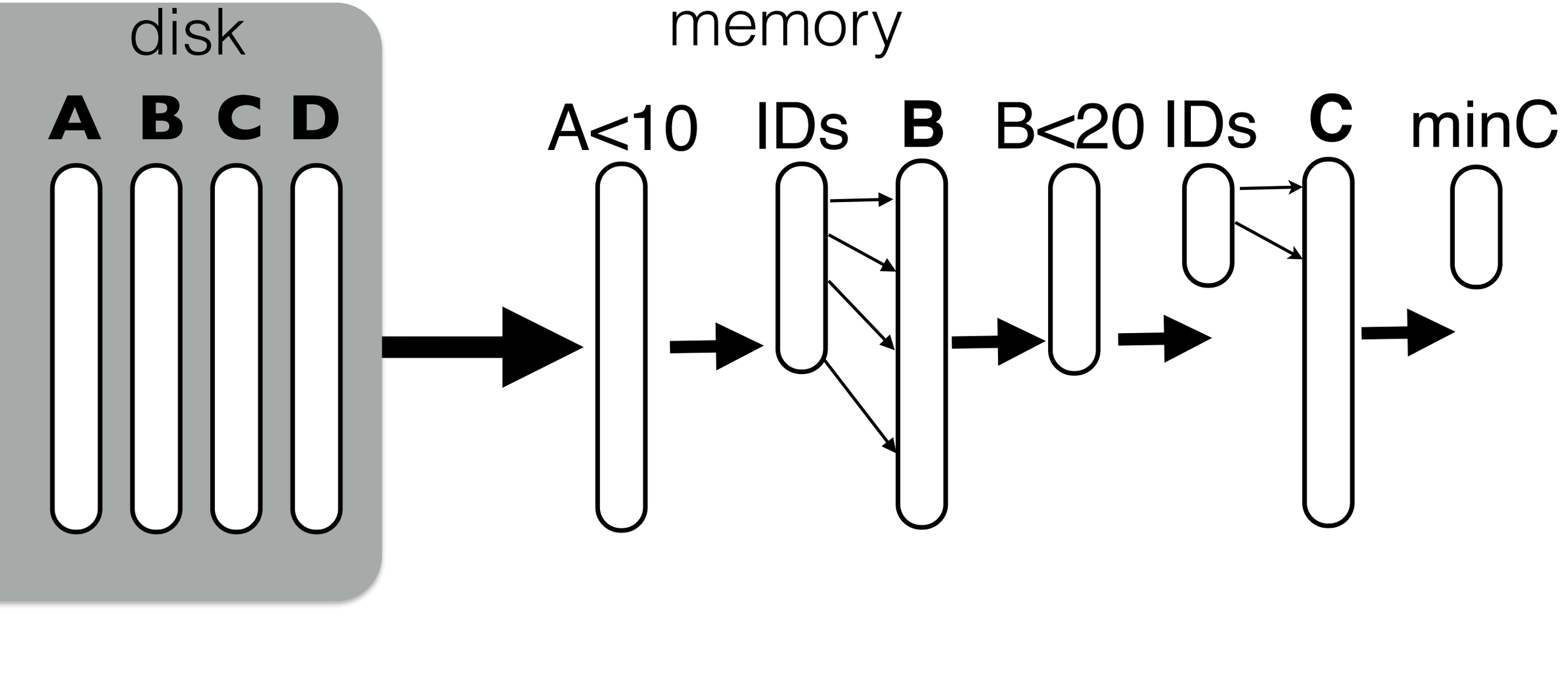
late reconstruction/materialization

select min(C) **from** R **where** A<10 & B<20



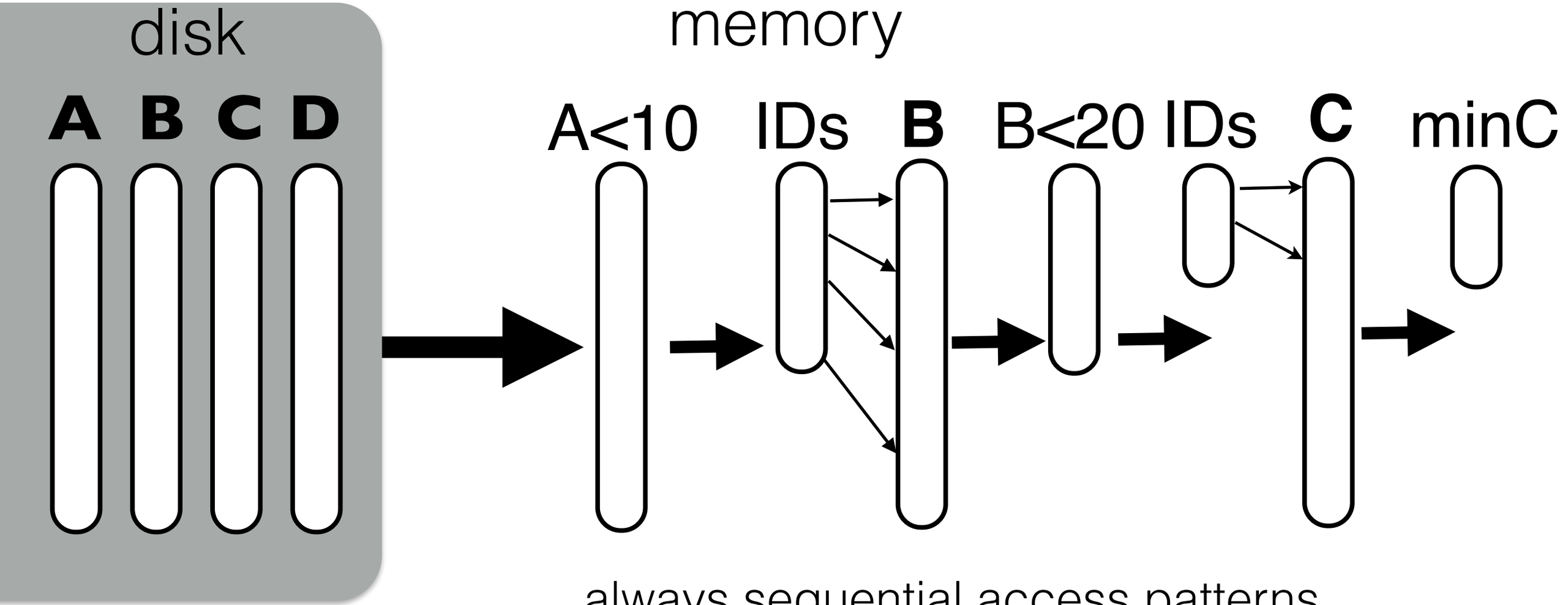
late reconstruction/materialization

select min(C) **from** R **where** A<10 & B<20



late reconstruction/materialization

select min(C) **from** R **where** A<10 & B<20



always sequential access patterns

memory contains only what is needed at any point in time

Notes to remember

column-stores vs row-stores

it all starts with how we store the data

still basic concepts are the same

moving data is a major cost component

it is not just about disk…

the whole memory hierarchy matters

keep up with reading!

Read: **Architecture of a Database System** (Sections 1,2,3,4)
by J. Hellerstein, M. Stonebraker and J. Hamilton

Read: **The Design and Implementation of Modern Column-store Database Systems**
by D. Abadi, P. Boncz, S. Harizopoulos, S. Idreos, S. Madden

basic db architectures & layouts

DATA SYSTEMS

prof. Stratos Idreos