DB Architecture

CIS 6500

Ryan Marcus

Today’s lesson: **databases are magical made of decidely unmagical components**

Why is making a DB hard?

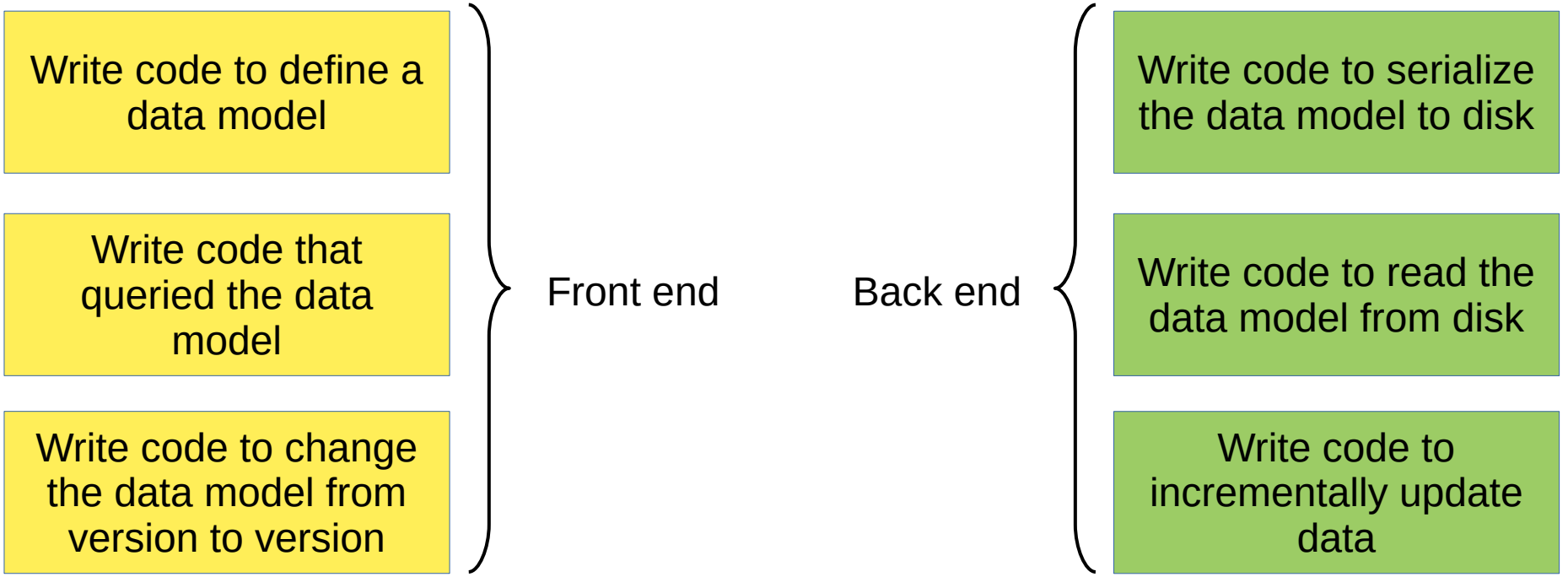
- Design goals of a DBMS
- Uniform interface to access and update data
- Works efficiently on large volumes of data
- Works efficiently on different hardware

Why is making a DB hard?

- Design goals of a DBMS
- **Uniform interface to access and update data**
- Works efficiently on large volumes of data
- Works efficiently on different hardware

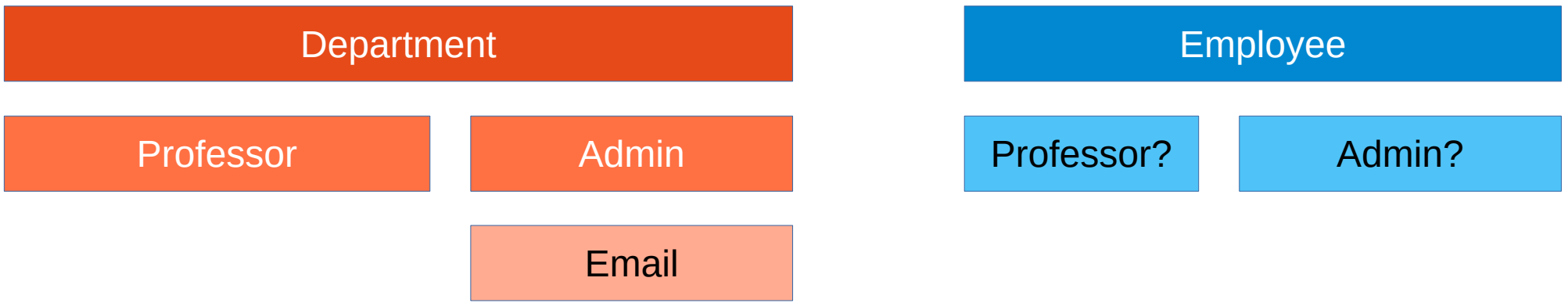
Uniform Interface

- 1960s – IBM **I**nformation **M**anagement **S**ystem

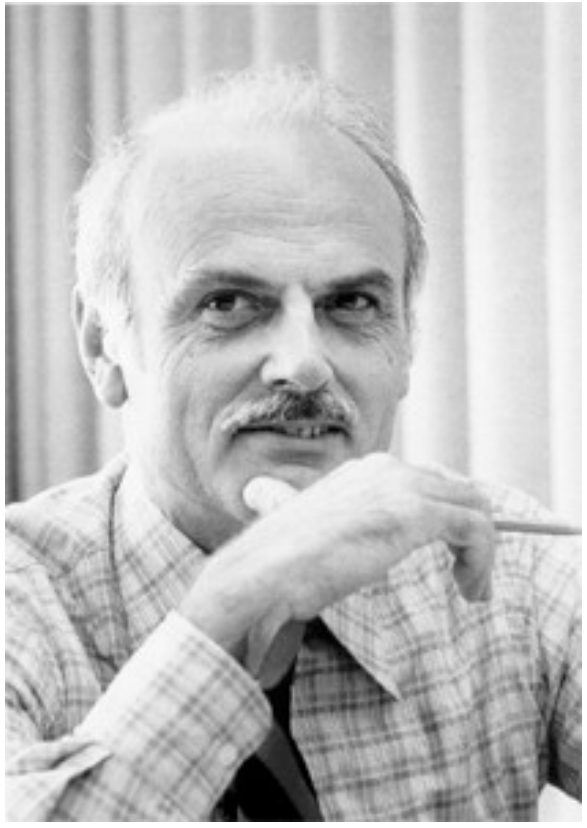


Uniform Interface

- Hierarchical was the norm



Name That Old Dead White Guy



- Edgar F. Codd, creator of the relational data model

Name	Dept
Ryan	CIS
Zack	CIS
Boon	CIS
Susan	CIS
Martha	Psych

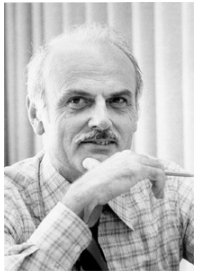
Dept	Admin	Email
CIS	Cheryl	che@...
Psych	Martin	mar@...

The Relational Model & 3NF

- Codd says we have relations with attributes, and that’s it!
- Some attributes *determine* other attributes
- When an attribute determines the whole relation, we call that attribute a key.

The attributes in the table are determined by...

The key, the whole key, and nothing but the key, so help me Codd



Uniform Interface

- The relational model is always *just about* to be replaced
  - Object-oriented databases
  - XML databases
  - “NoSQL” databases (MongoDB)
- … and yet here we are.

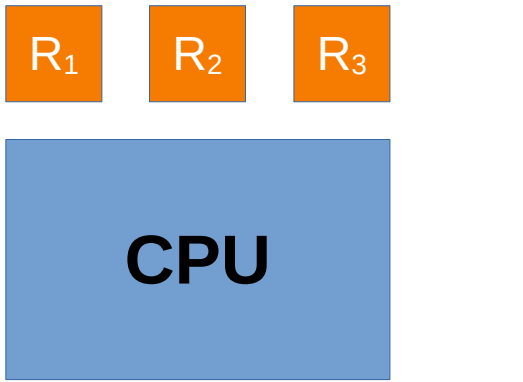
Why is making a DB hard?

- Design goals of a DBMS
- Uniform interface to access and update data
- **Works efficiently on large volumes of data**
- Works efficiently on different hardware

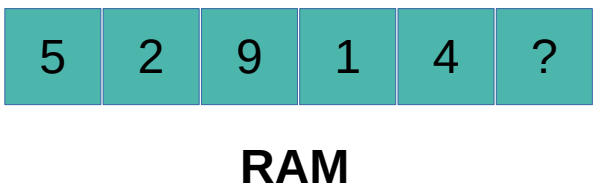
Large Volumes

- Always growing
  - 2013: 9 ZB. 2023: 57 ZB.
- Median tech org: 4 PB (Gardner ‘19)
- Can’t just wing it

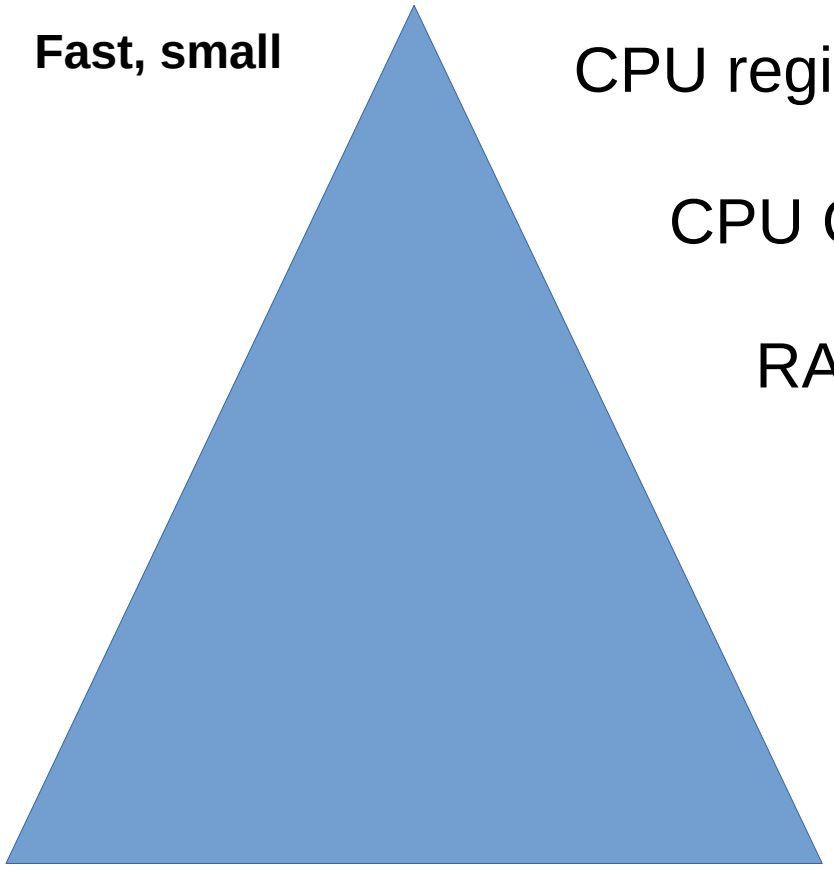
How do computer's process data?




```
LOAD <mem addr>, <reg>
STOR <reg>, <mem addr>
ADDR <dst reg>, <reg>, <reg>
```



Memory Hierarchy



CPU registers (< 1ns)

CPU Cache (10ns)

RAM (100ns)

Disk (1ms)

Local network (10ms)

Remote network (100ms)

Buffer Management

Queries:

RAM

Capacity 2

Disk

Capacity 4

S3

Capacity 8

Buffer Management

Queries: Read A, B, C

RAM

Capacity 2

Disk

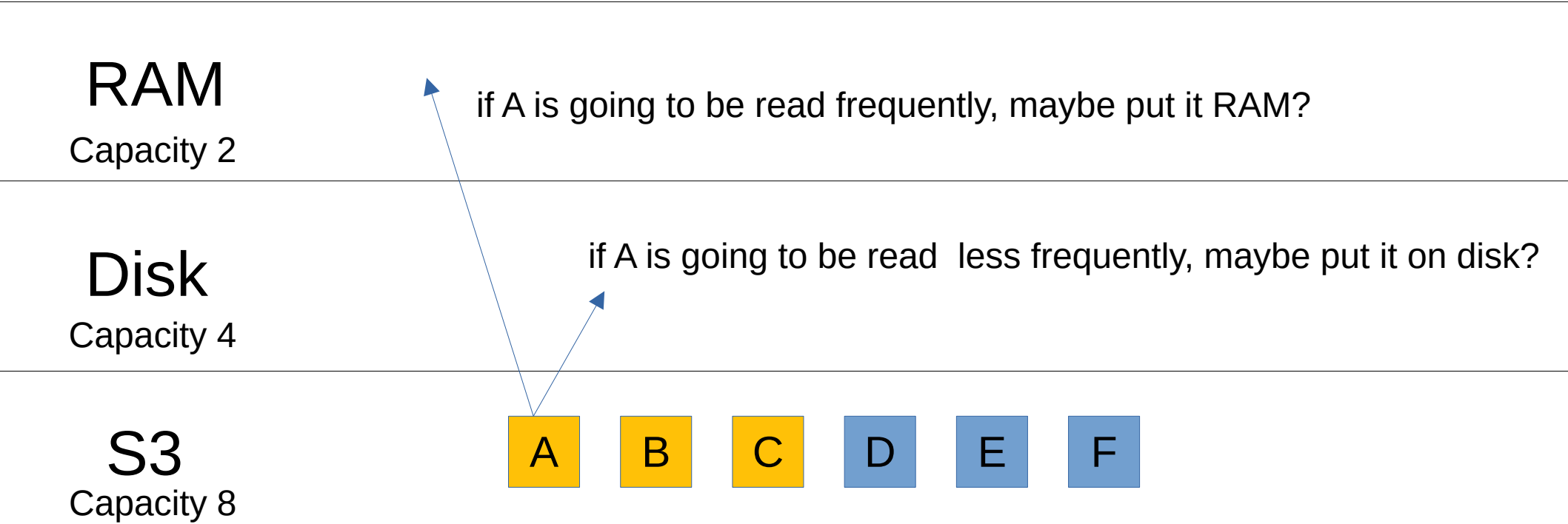
Capacity 4

S3

Capacity 8

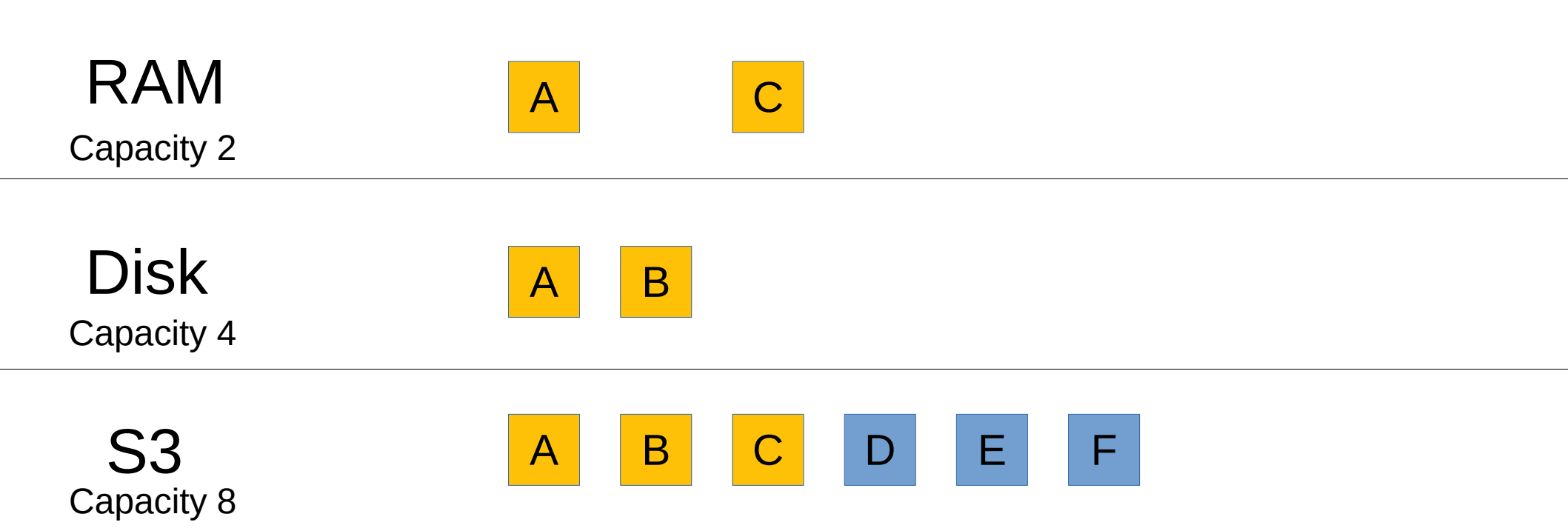
Buffer Management

Queries: Read A, B, C



Buffer Management

Queries: Read A, B, C

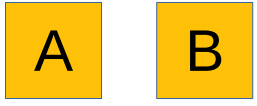


Buffer Management

Queries: Read A, B, C
Next Q: Read A, C 👍

RAM
Capacity 2

Disk
Capacity 4



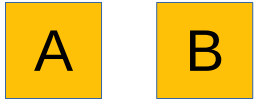
S3
Capacity 8

Buffer Management

Queries: Read A, B, C
Next Q: Read B, D 👎

RAM
Capacity 2

Disk
Capacity 4



S3
Capacity 8

Why is making a DB hard?

- Design goals of a DBMS
- Uniform interface to access and update data
- Works efficiently on large volumes of data
- **Works efficiently on different hardware**

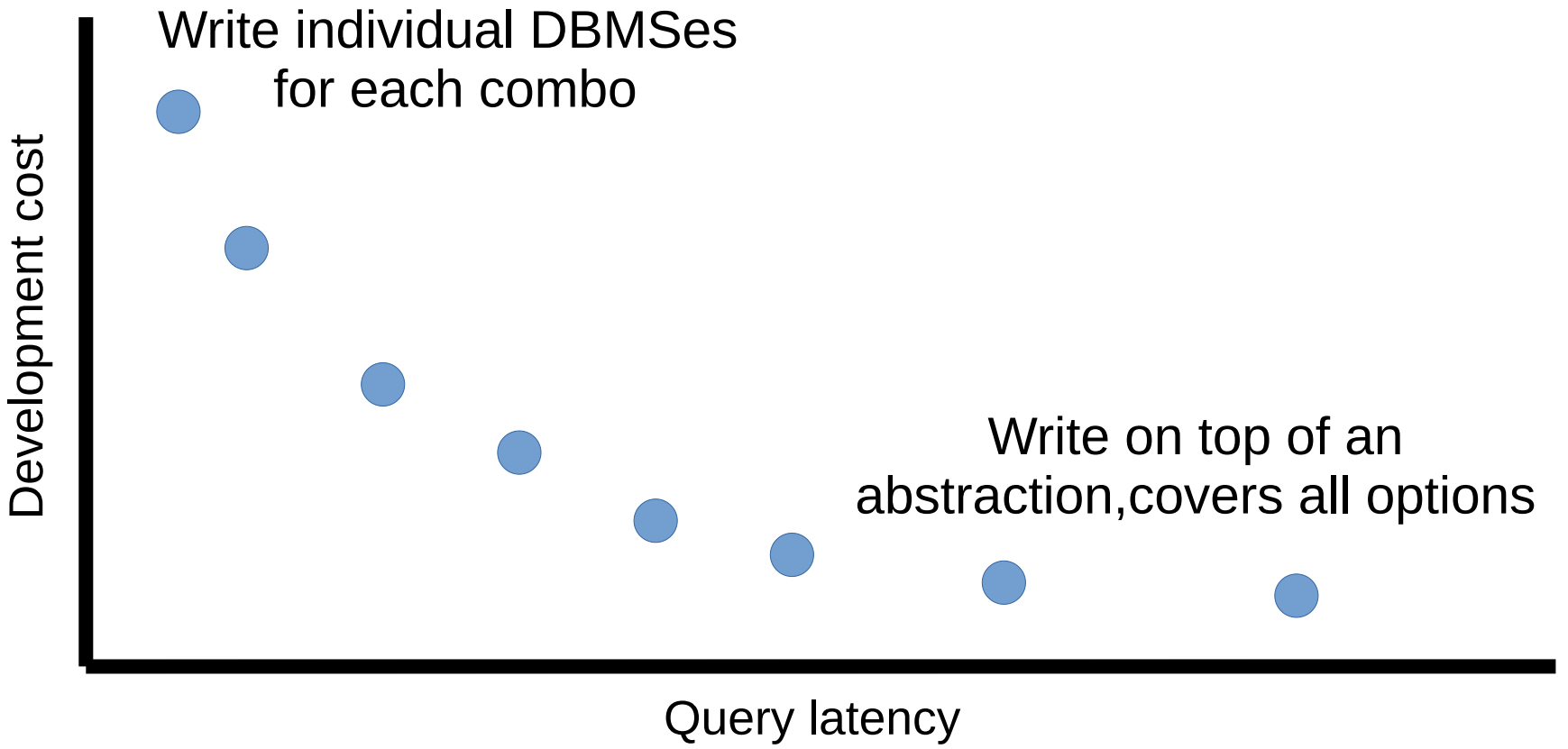
Hardware Diversity

CPU
AMD
Intel
ARM

Storage
SSD
HDD
NVMe
S3

Network
Ethernet
Infiniband
RDMA

Hardware Diversity

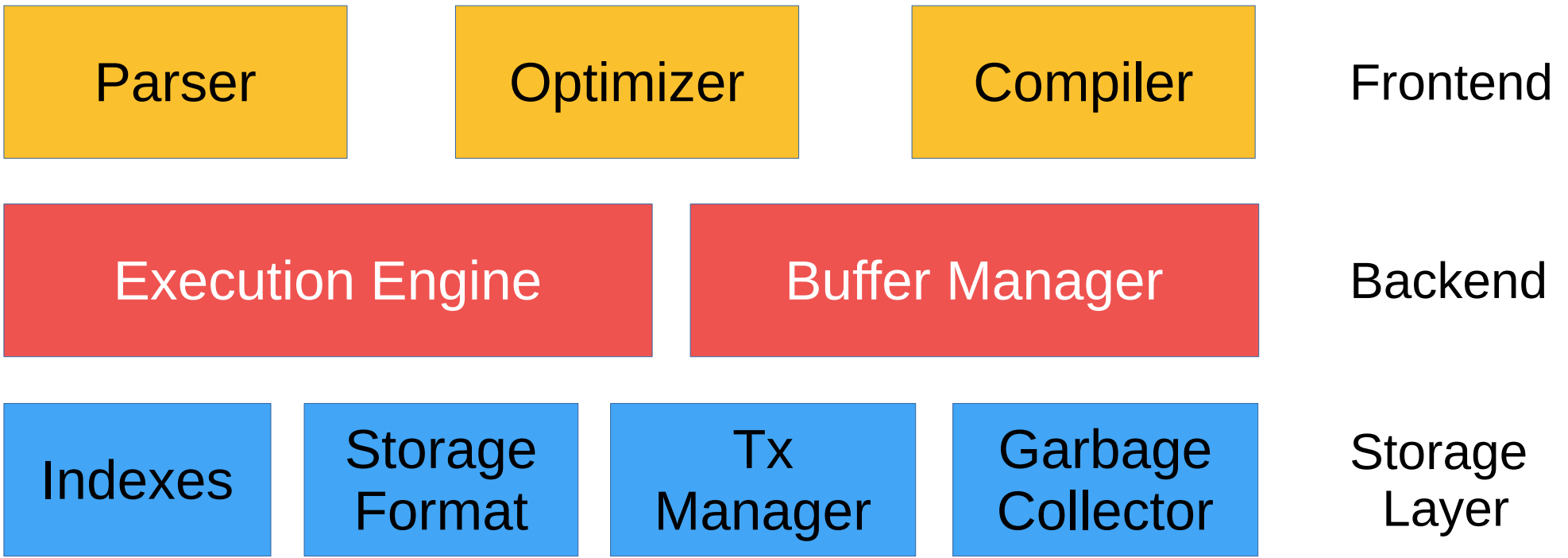


DB Architecture

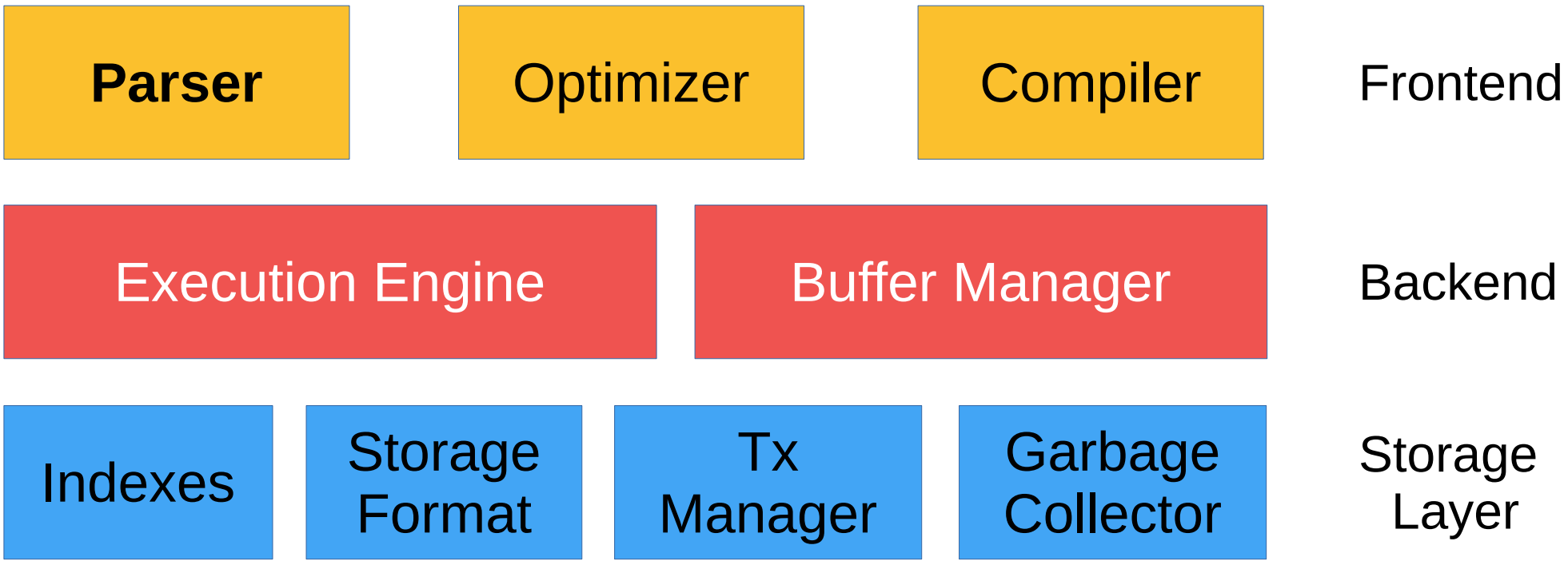
- How to build a software system that has a uniform **interface**, efficient performance on **large data**, and can run **everywhere**?

- Look at the general-purpose architecture that makes this happen. Specific examples from Vertica.

DB Architecture



DB Architecture



Parser

- Not going to be a focus of this class, but a surprisingly complex problem

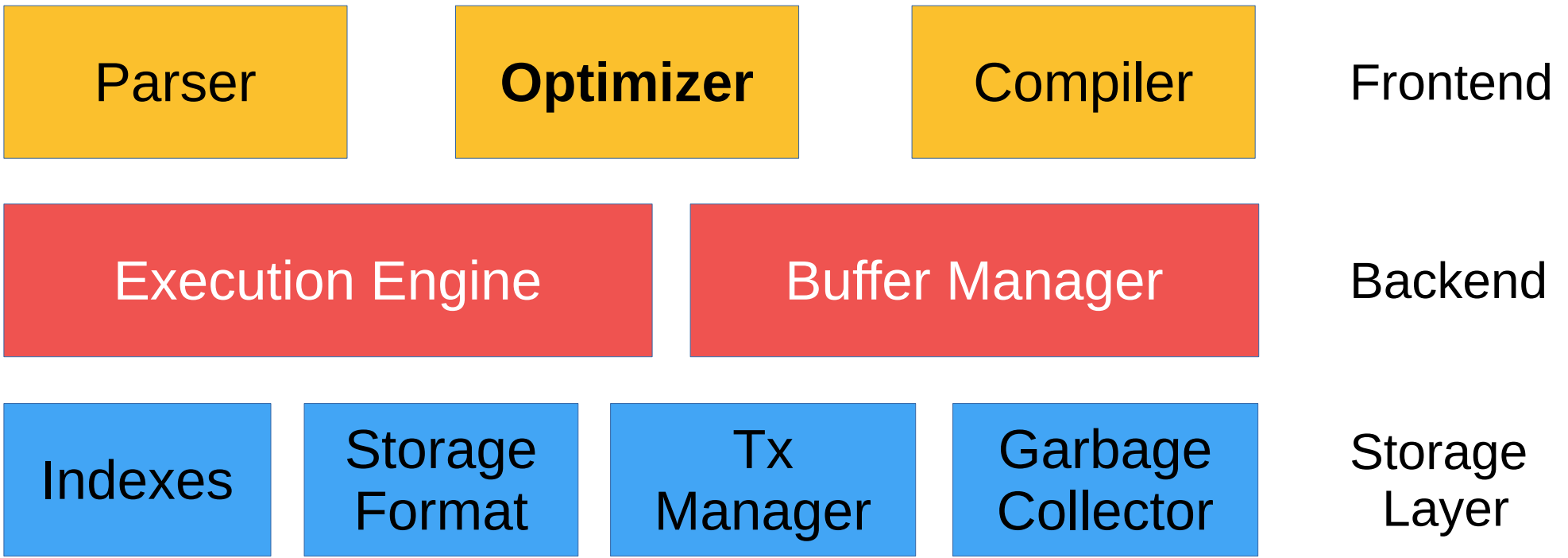
```
SELECT t1.c1, b.c3
FROM t1 a, t2 b
WHERE t1.c1 = b.c2 AND c8 = t2.c1
```

- Resolve entities, point to errors

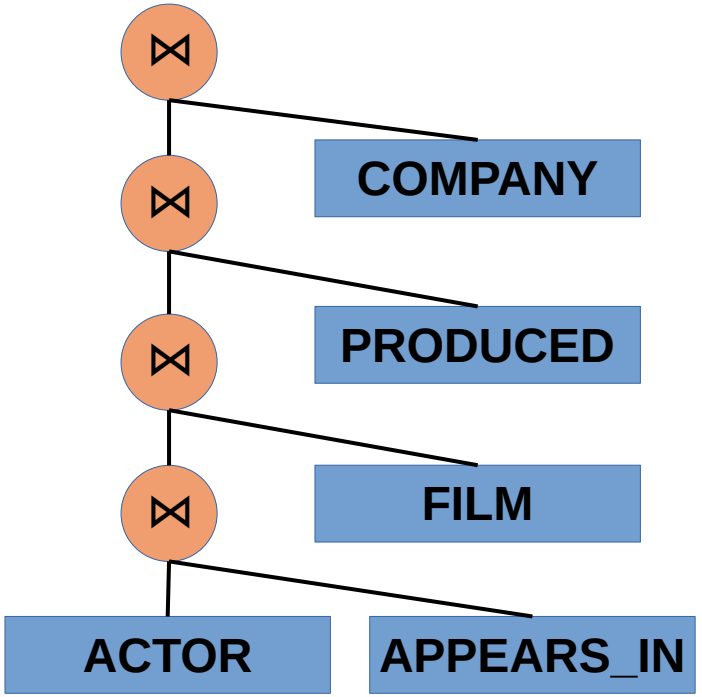
```
WITH
    recursive_cte AS
    (
        SELECT
            #cats.cat_id
            ,#cats.cat
            ,#cats.ancestor_id
            ,CAST(NULL AS VARCHAR(20)) as ancestor
            ,0 as level
            ,CAST(#cats.cat as VARCHAR(MAX)) as lineage
        FROM
            #cats
        WHERE
            ancestor_id IS NULL

        UNION ALL
        SELECT
            #cats.cat_id
            ,#cats.cat
            ,#cats.ancestor_id
            ,recursive_cte.cat
            ,level + 1
            ,recursive_cte.lineage + ' > ' + #cats.cat
        FROM
            #cats
                INNER JOIN recursive_cte
                 ON recursive_cte.cat_id = #cats.ancestor_id    )
```

DB Architecture



Query Optimization



ACTOR

a_id	name	YOB
1	Scarl	84
2	BradP	63
3	JonTr	54
…		

FILM

f_id	name	RAT
1	Her	86
2	Aveng	81
3	PulpF	93
…		

COMPANY

c_id	name
1	Sony
2	Fox
3	MGM
…	

APPEARS_IN

a_id	f_id
1	1
1	2
3	3
…	

PRODUCED

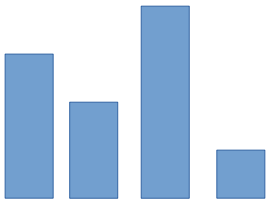
c_id	f_id
1	1
2	2
3	2
…	

Find all movies with SJ. Then, filter those by movies produced by Sony.

Query Optimization

Optimizer

Cardinality Estimation



There are 100 Penn students in a room.
50 are CIS majors
20 have taken advanced DB

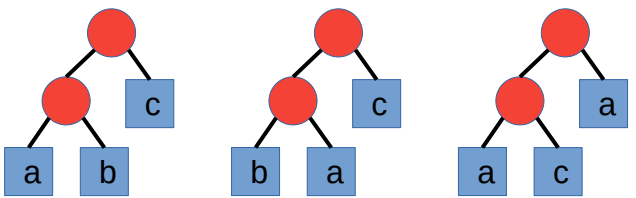
Ask the CE, how CIS majors haven't taken adv DB?

CE: 50/100 * 80/100 = 40

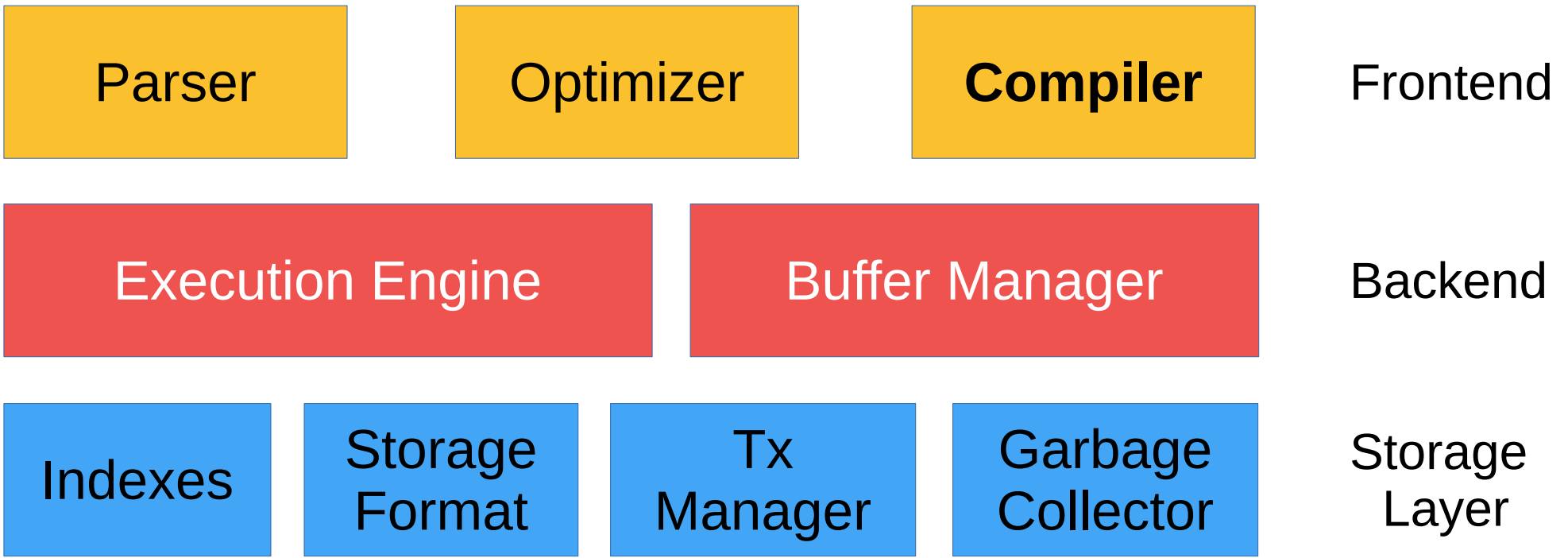
Cost model

HJ(x, y) = |x| + |y|

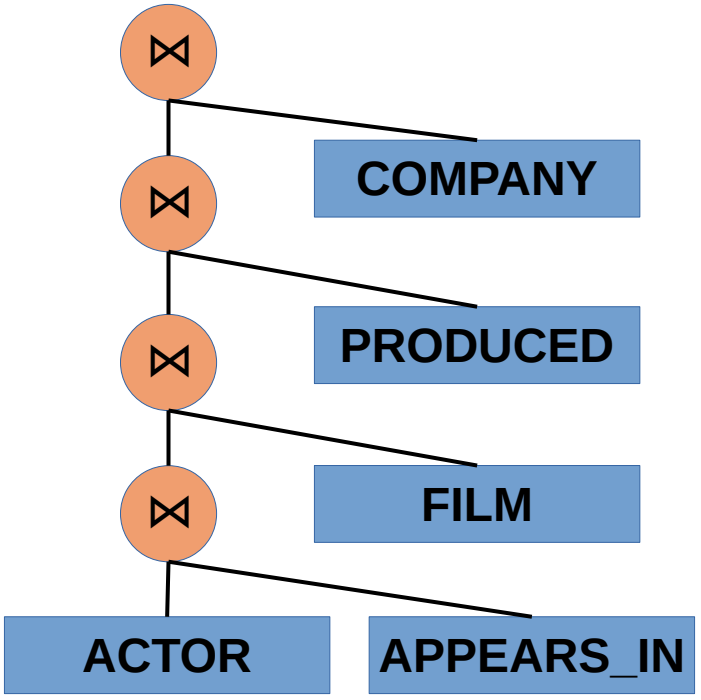
Enumerator



DB Architecture

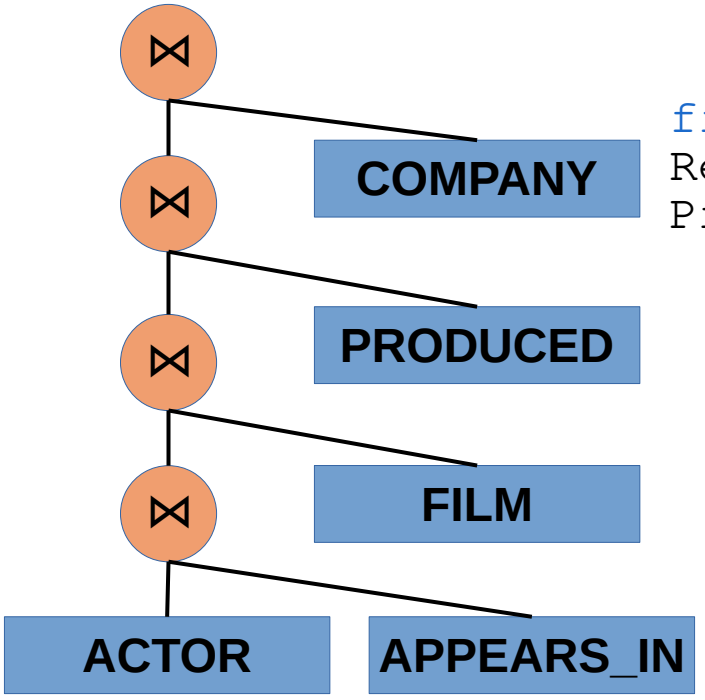


Compiler



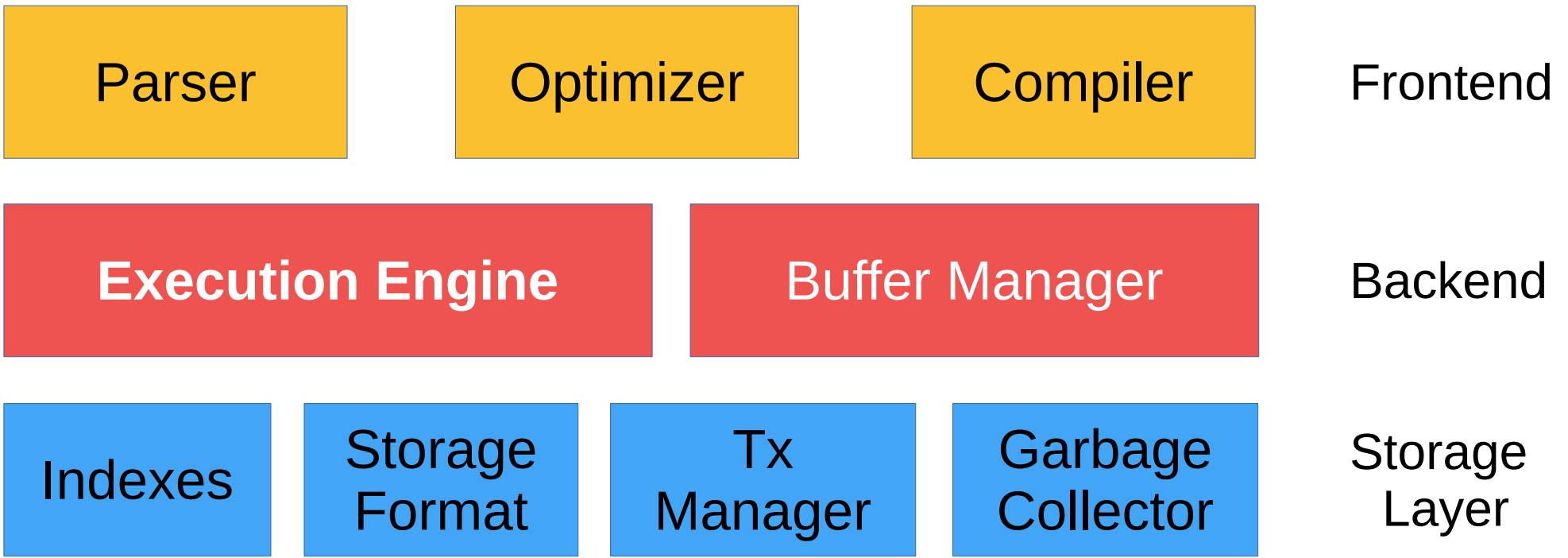
- Optimizer might not produce every last detail
- File paths
- Encodings
- Exact predicates

Compiler

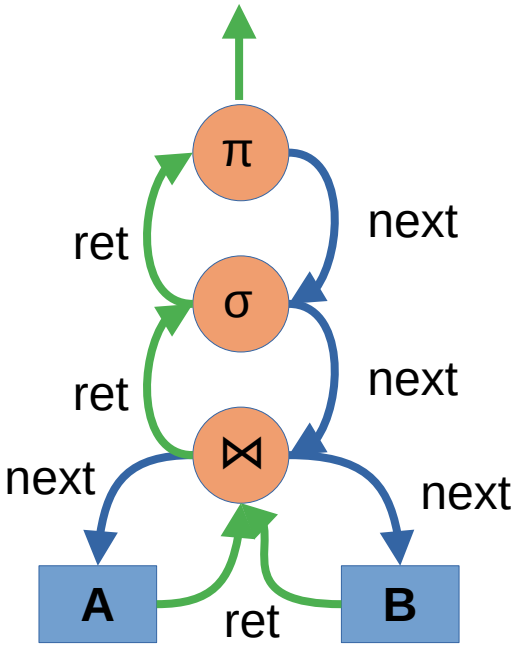



```
file://var/lib/pg_data/company.toast
Reader: GZIP
Predicate: name == 'Sony'
```

DB Architecture



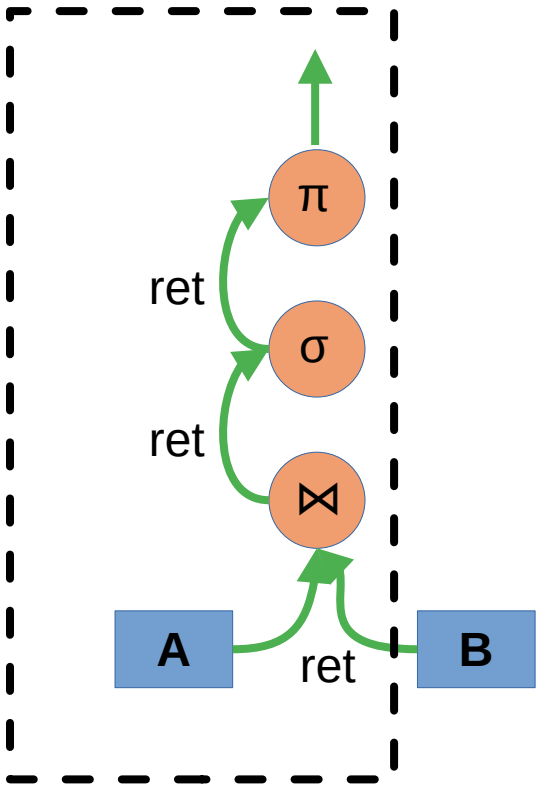
“Volcano” style




```
interface Operator {
   tuple next();
}

class Filter implements Operator {
  Operator child;
  Predicate f;
  tuple next() {
    while (tmp = self.child.read()) != null {
      if (self.f.test(tmp)) return tmp;
    }
    return null;
  }
```

“Push” style




```
runPipeline(Proj p, Filt f, Join j, Scan s) {
  while (tmp = s.read()) != null {
    tmp = j.join(tmp);
    if !f.test(tmp) continue;
    tmp = p.project(tmp);
    output.add(tmp);
  }
}
```

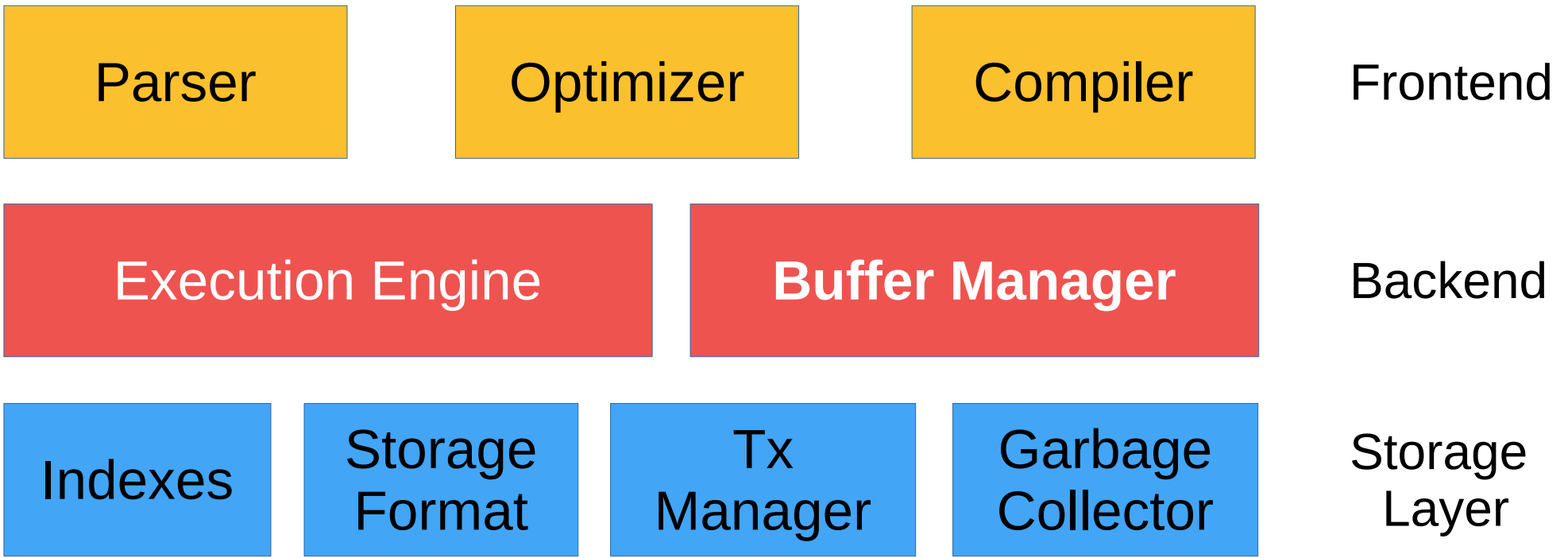
Execution Engine

- Interpreted engines
  - Generally “pull” model
  - Tuple-at-a-time or vectorized
- Compiling engines
  - Pipelined / generally “push” model

Simple
Extendable

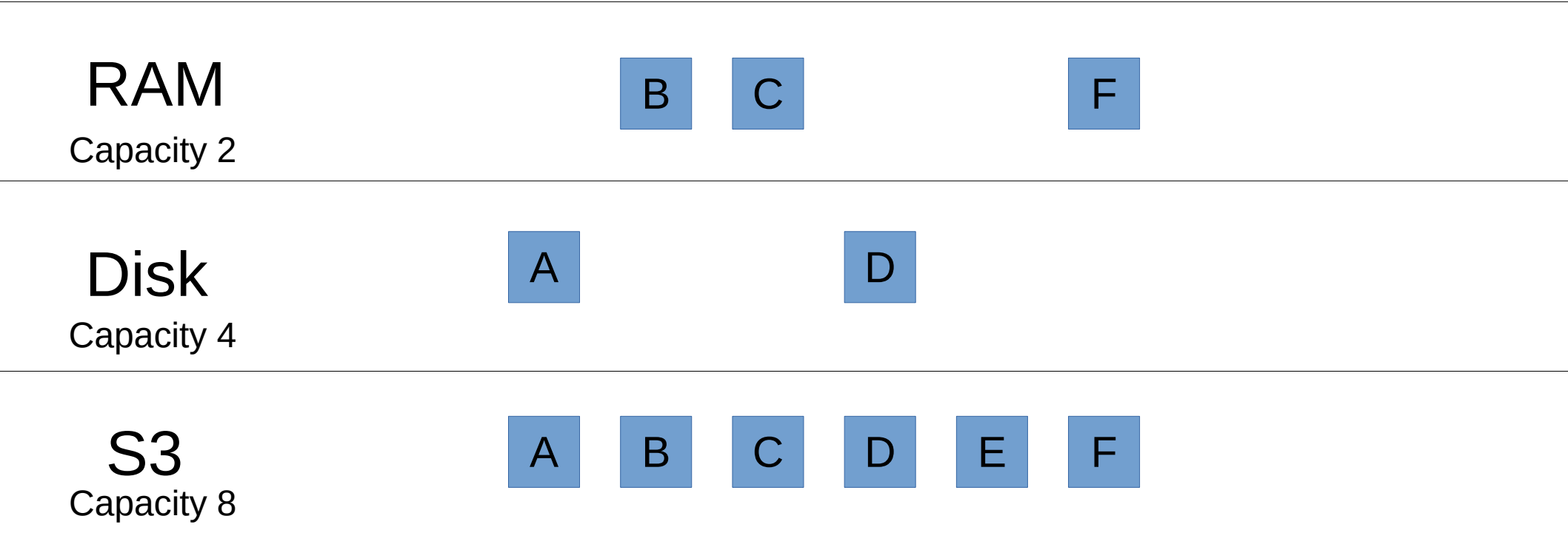
Performance

DB Architecture



Buffer Manager

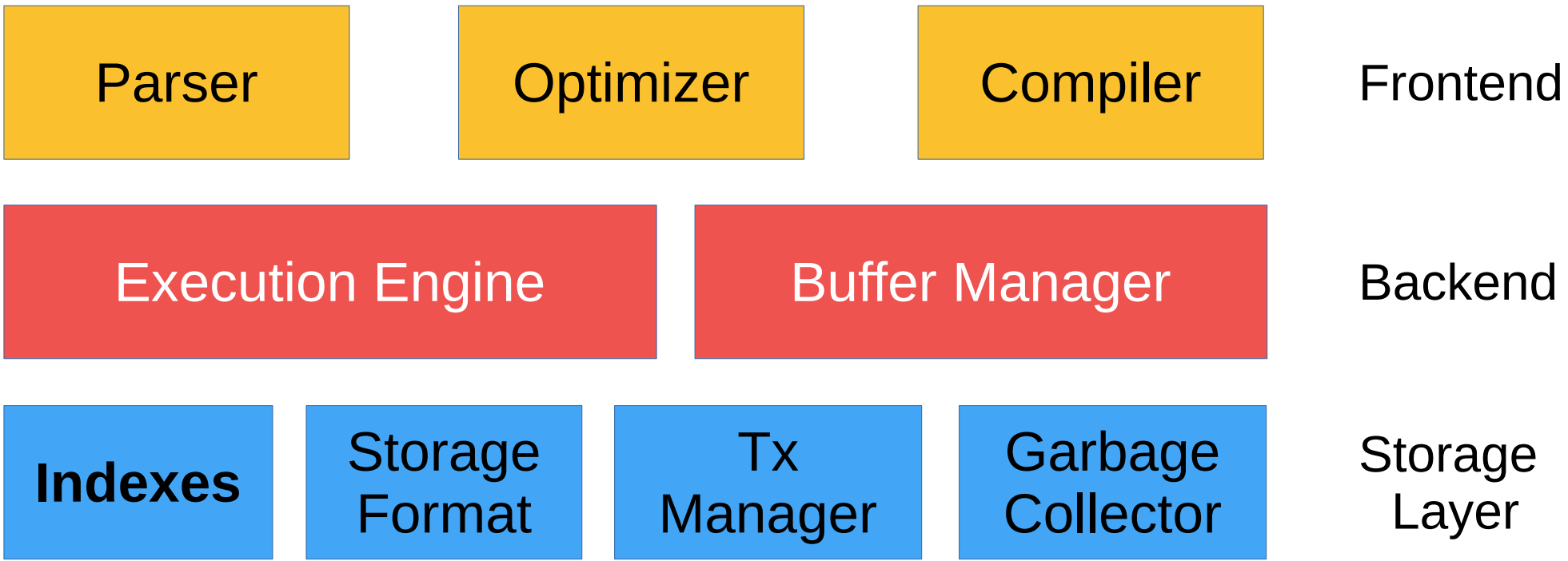
Queries:



Buffer Manager

- *Also* needs to allocate memory between query processing and data pages

DB Architecture



Indexes

- Two options to find relevant data: scan or index.

	Scan	Index
Q: x >= 5	3	4
Q: x >= 200	3	1
Q: x >= 102	3	2

B1: 5-90
B2: 32-105
B3: 0-100

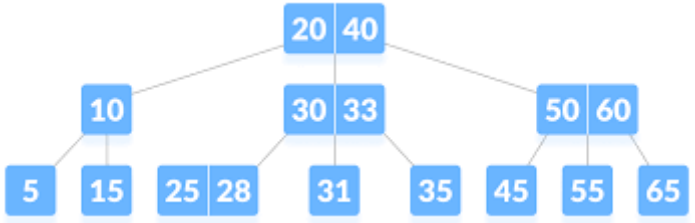
5, 10, 15, 23, 55, 90

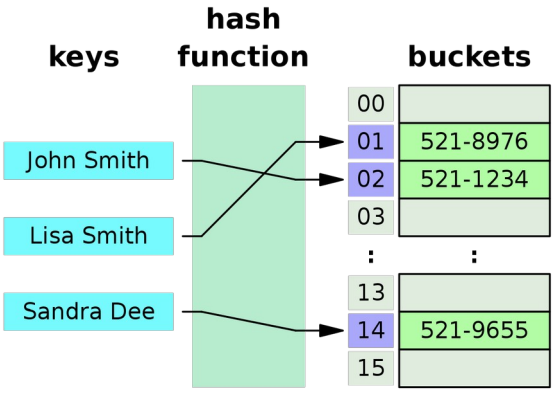
32, 92, 104, 105, 105

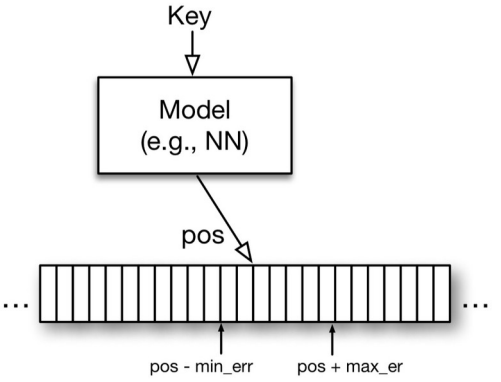
0, 1, 2, 3, 4, 100

Indexes

- Two options to find relevant data: scan or index.

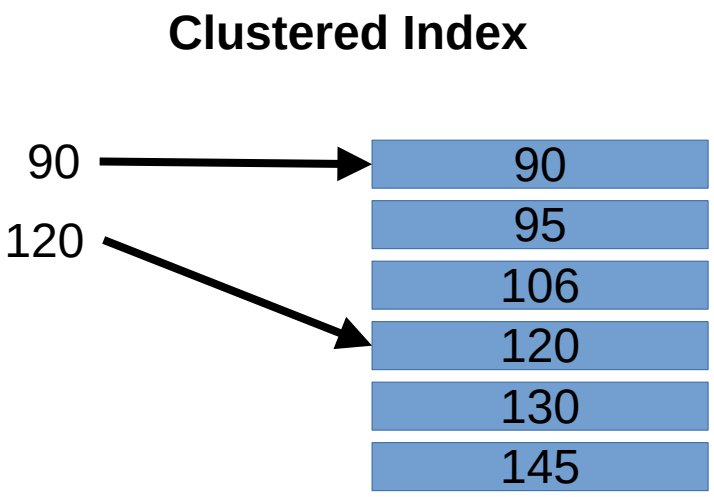


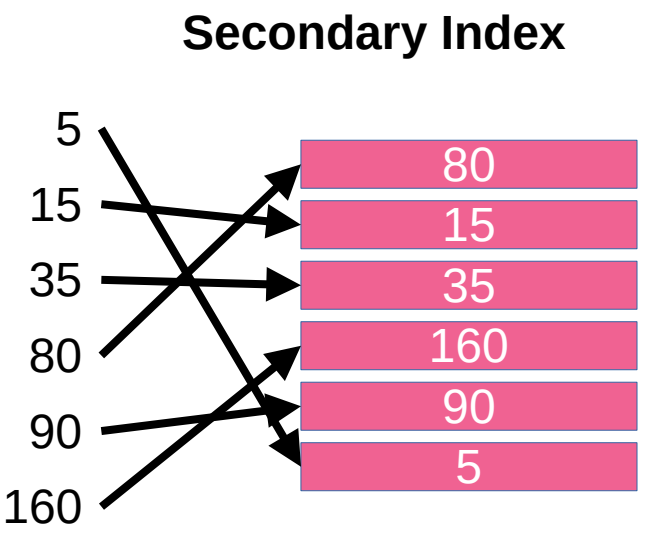




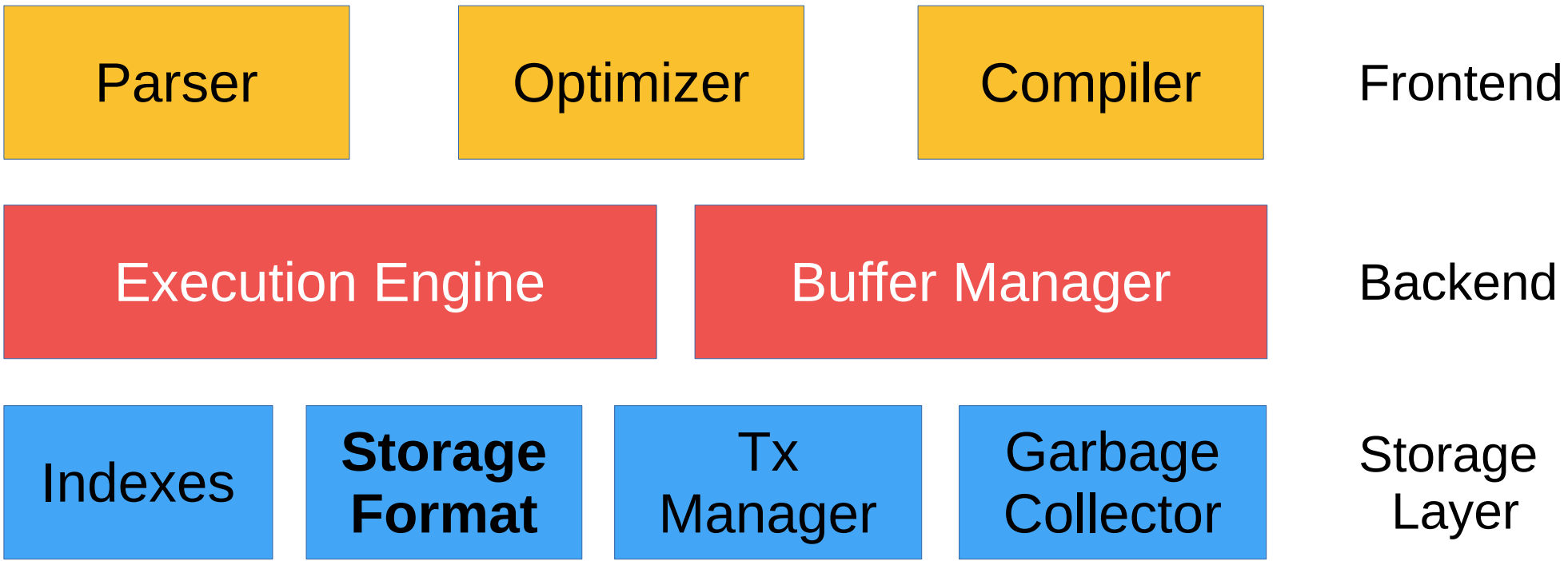
Indexes

- Two options to find relevant data: scan or index.





DB Architecture



Storage Format

- Data layout impacts read sizes

A	B	C	D
Alpha	1	Red	UPenn
Beta	2	Black	Cornell
Gamma	3	Blue	Harvard
Delta	4	Purple	Dartmouth

```
SELECT B, C FROM T;
```

Storage Format

- Data layout impacts read sizes

A	B	C	D
Alpha	1	Red	UPenn
Beta	2	Black	Cornell
Gamma	3	Blue	Harvard
Delta	4	Purple	Dartmouth

Row order

```
SELECT B, C FROM T;
```

4 blocks

Storage Format

- Data layout impacts read sizes

A	B	C	D
Alpha	1	Red	UPenn
Beta	2	Black	Cornell
Gamma	3	Blue	Harvard
Delta	4	Purple	Dartmouth

Column order

```
SELECT B, C FROM T;
```

2 blocks

Storage Format

- Data layout impacts read sizes

A	B	C	D
Alpha	1	Red	UPenn
Beta	2	Black	Cornell
Gamma	3	Blue	Harvard
Delta	4	Purple	Dartmouth

```
INSERT INTO T
(A, B, C, D)
VALUES (Omega, 5,
Pink, Yale)
```

Storage Format

- Data layout impacts read sizes

A	B	C	D
Alpha	1	Red	UPenn
Beta	2	Black	Cornell
Gamma	3	Blue	Harvard
Delta	4	Purple	Dartmouth

Row order

```
INSERT INTO T
(A, B, C, D)
VALUES (Omega, 5,
Pink, Yale)
```

1 block

Storage Format

- Data layout impacts read sizes

A	B	C	D
Alpha	1	Red	UPenn
Beta	2	Black	Cornell
Gamma	3	Blue	Harvard
Delta	4	Purple	Dartmouth

Column order

```
INSERT INTO T
(A, B, C, D)
VALUES (Omega, 5,
Pink, Yale)
```

4 blocks

Storage Format

A
5
5
5
5
7
7
7

Normal

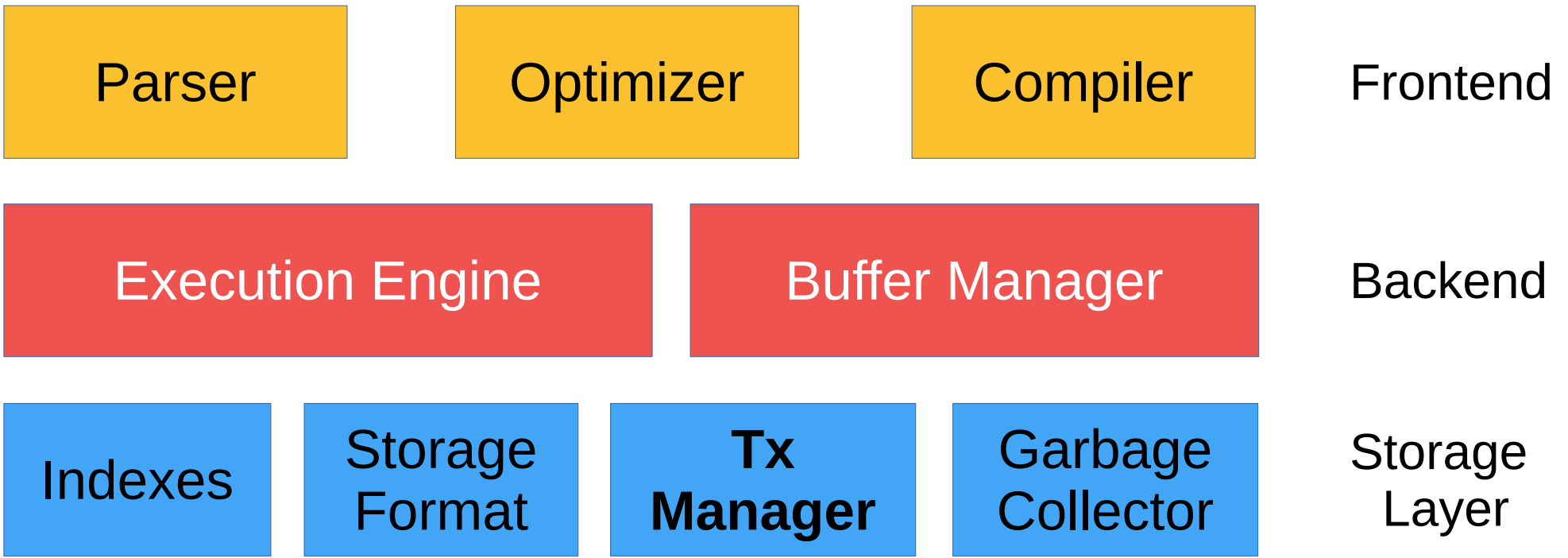
A
4x 5
3x 7

Run-length

A
5
0
0
0
2
0
0

Delta

DB Architecture



Transaction Manager

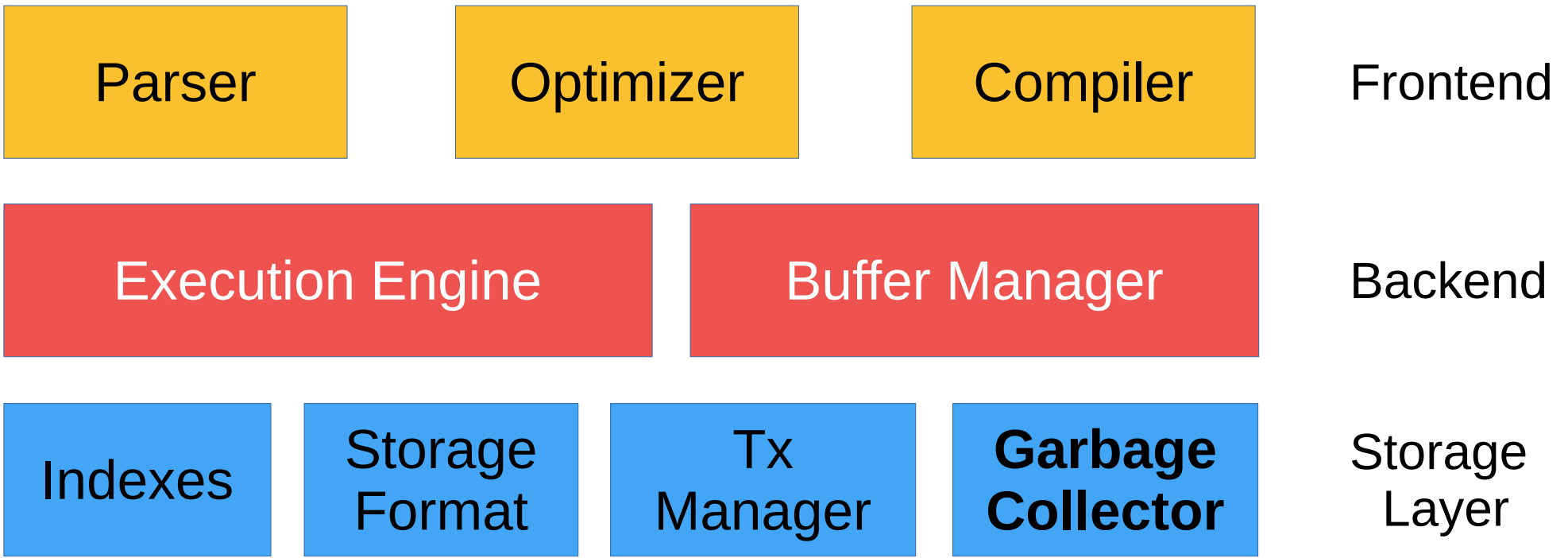
- Tons of techniques – not the focus of this class.

Tx1: set 2 to X, set 5 to Y, lookup 7

Write-ahead log
Tx1 SET 2 to X (was B)
Tx1 SET 5 to Y (was E)
Tx1 COMMIT

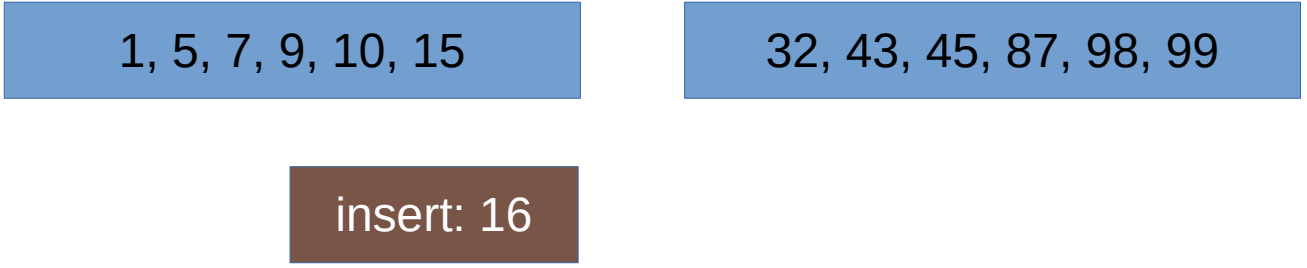
Key	Value
1	A
2	B
3	C
4	D
5	E
6	F
7	G

DB Architecture



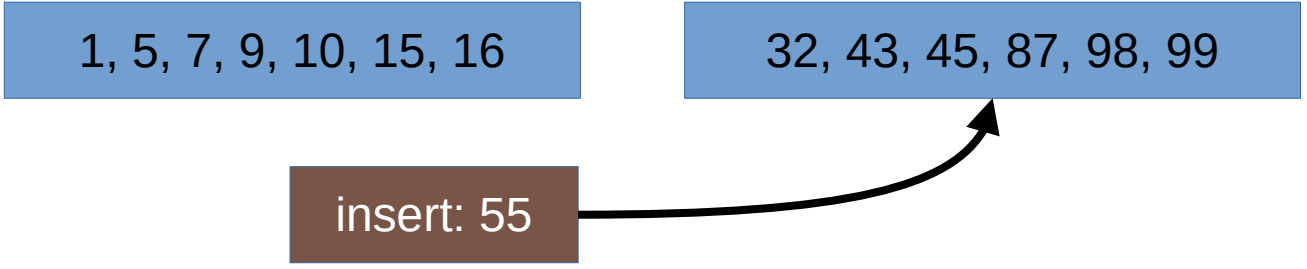
Garbage Collector

- DELETEs and re-layouts



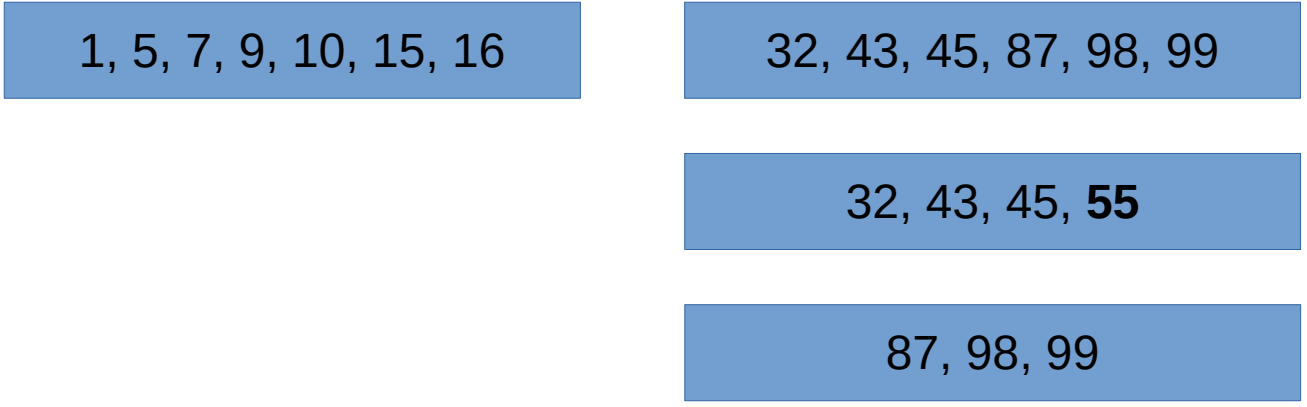
Garbage Collector

- DELETEs and re-layouts



Garbage Collector

- DELETEs and re-layouts



Garbage Collector

- DELETEs and re-layouts