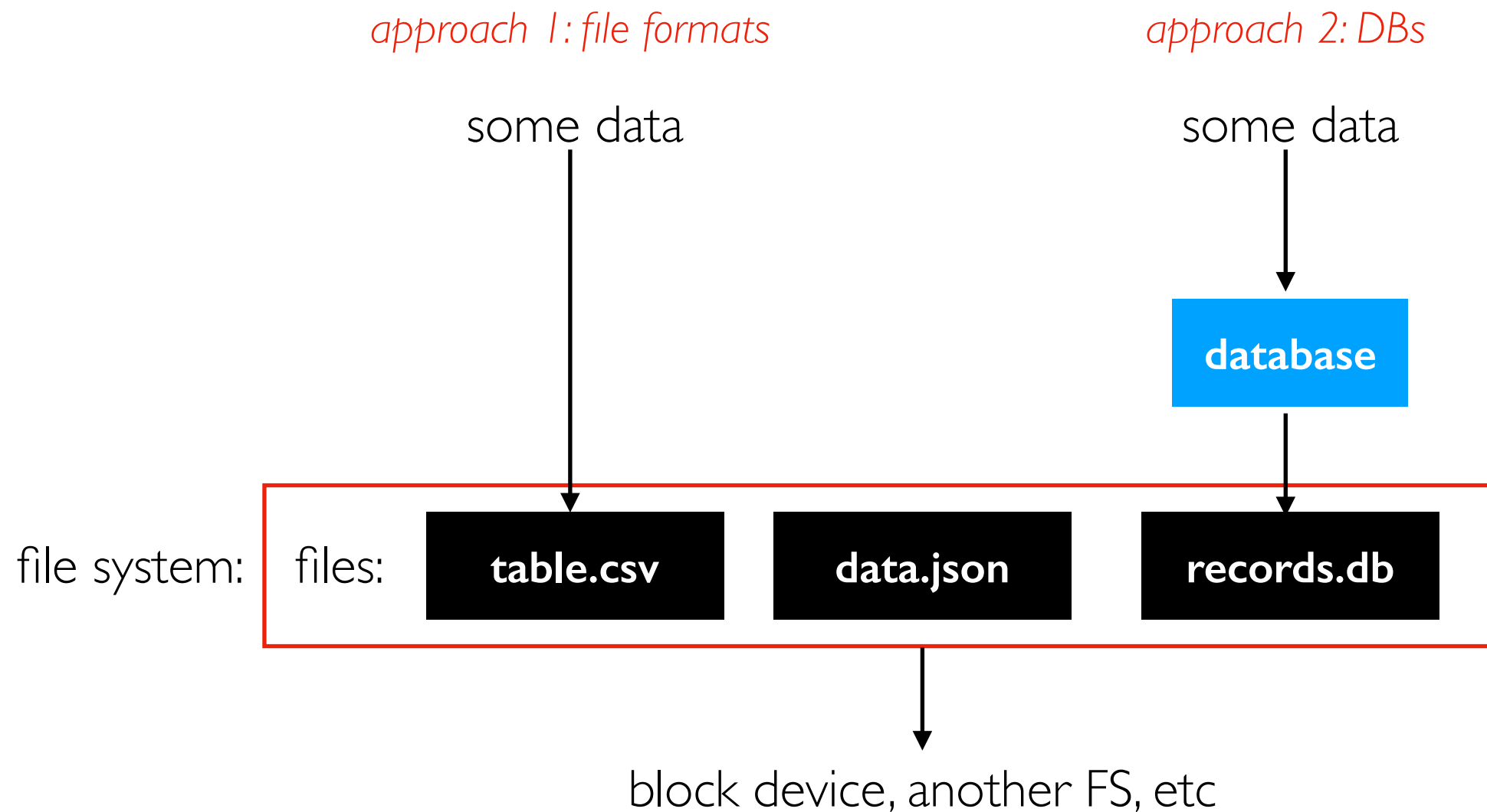# [544] File Formats

Tyler Caraza-Harter

# Learning Objectives

- describe different file formats in terms of orientation, encoding, compression, and schemas

- write code to use parquet files

- differentiate between transactions workloads and analytics workloads

- explain the motivation for using an ETL (extract transform load) process to copy data from an transactions processing system to an analytics processing system

File systems let us give names to sequences of bytes (files) and hierarchically organize those files (via directories). We usually want some structure for those bytes.

some data

some data

**database**

file system:  files:

**table.csv**    **data.json**    **records.db**

block device, another FS, etc

# Outline

|  | CSV | Parquet |
|---|---|---|
| **orientation** | row | column |
| **encoding** | text | binary |
| **compression** | none | snappy |
| **schemas** | inferred | explicit |

Demos

Databases

# File Layout

Goals

- efficient input/output from storage (large enough reads/writes, sequential accesses)
- minimize parsing/deserialization computation time

Assumptions

- many file systems will try to map consecutive bytes of a file to consecutive blocks on a storage device (but note that in some cases sequential file I/O becomes random disk I/O)
- need to clarify assumptions about how code will access the data (for example, one whole column? a row at a time?)

```
ACW00011604  17.1167  -61.7833    10.1     ST JOHNS COOLIDGE FLD
ACW00011647  17.1333  -61.7833    19.2     ST JOHNS
AE000041196  25.3330   55.5170    34.0     SHARJAH INTER. AIRP        GSN     41196
AEM00041194  25.2550   55.3640    10.4     DUBAI INTL                         41194
AEM00041217  24.4330   54.6510    26.8     ABU DHABI INTL                     41217
AEM00041218  24.2620   55.6090   264.9     AL AIN INTL                        41218
AF000040930  35.3170   69.0170  3366.0     NORTH-SALANG               GSN     40930
AFM00040938  34.2100   62.2280   977.2     HERAT                              40938
AFM00040948  34.5660   69.2120  1791.3     KABUL INTL                         40948
AFM00040990  31.5000   65.8500  1010.0     KANDAHAR AIRPORT                   40990
AG000060390  36.7167    3.2500    24.0     ALGER-DAR EL BEIDA         GSN     60390
AG000060590  30.5667    2.8667   397.0     EL-GOLEA                   GSN     60590
AG000060611  28.0500    9.6331   561.0     IN-AMENAS                  GSN     60611
AG000060680  22.8000    5.4331  1362.0     TAMANRASSET                GSN     60680
AGE00135039  35.7297    0.6500    50.0     ORAN-HOPITAL MILITAIRE
AGE00147704  36.9700    7.7900   161.0     ANNABA-CAP DE GARDE
AGE00147705  36.7800    3.0700    59.0     ALGIERS-VILLE/UNIVERSITE
AGE00147706  36.8000    3.0300   344.0     ALGIERS-BOUZAREAH
```

ghcnd-stations.txt

good: just read the one block containing the row

bad: need to read everything to access any one column

# File Layout

Goals

- efficient input/output from storage (large enough reads/writes, sequential accesses)
- minimize parsing/deserialization computation time

Assumptions

- many file systems will try to map consecutive bytes of a file to consecutive blocks on a storage device (but note that in some cases sequential file I/O becomes random disk I/O)
- need to clarify assumptions about how code will access the data (for example, one whole column?  a row at a time?)

Major access patterns

- transactions processing: *reading/changing a row (or few rows) as needed by an application* (note: "transaction" has other meanings for databases as well -- more later...)
- analytics processing: *computing over many rows for specific columns*

# Row-Oriented vs. Column-Oriented Layout

| col1 | col2 | col3 |
|------|------|------|
| 1 | 5 | A |
| 2 | 6 | B |
| 3 | 7 | C |
| 4 | 8 | D |

row-oriented file:   1 5 A 2 6 B 3 7 C 4 8 D

col-oriented file:   1 2 3 4 5 6 7 8 A B C D

→ position in file

# Row-Oriented vs. Column-Oriented Layout

| col1 | col2 | col3 |
|------|------|------|
| 1 | 5 | A |
| 2 | 6 | B |
| 3 | 7 | C |
| 4 | 8 | D |

row-oriented file: 1 5 A 2 6 B 3 7 C 4 8 D   fast

transactional access pattern

col-oriented file: 1 2 3 4 5 6 7 8 A B C D   slow

position in file

# Row-Oriented vs. Column-Oriented Layout

| col1 | col2 | col3 |
|------|------|------|
| 1 | 5 | A |
| 2 | 6 | B |
| 3 | 7 | C |
| 4 | 8 | D |

row-oriented file:  1 5 A 2 6 B 3 7 C 4 8 D  slow

col-oriented file:  1 2 3 4 5 6 7 8 A B C D  fast

analytics access pattern

position in file

# Row-Oriented vs. Column-Oriented Layout

| col1 | col2 | col3 |
|------|------|------|
| 1 | 5 | A |
| 2 | 6 | B |
| 3 | 7 | C |
| 4 | 8 | D |

row-oriented file:  1 5 A 2 6 B 3 7 C 4 8 D  CSV

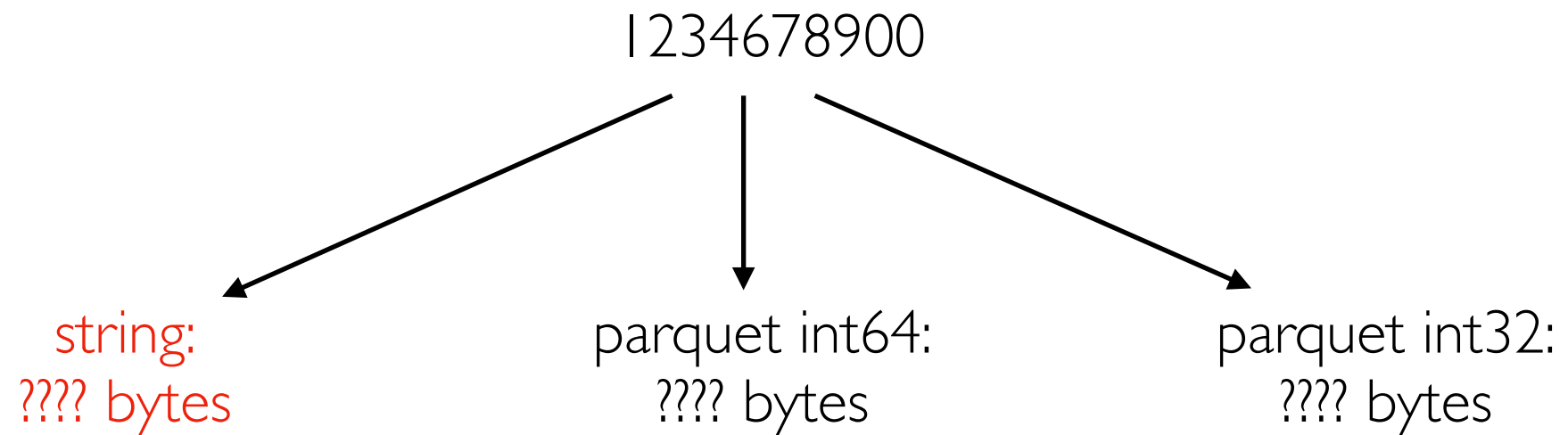col-oriented file:  1 2 3 4 5 6 7 8 A B C D  Parquet

→

position in file

# Outline

File Formats

|              | CSV      | Parquet  |
| ------------ | -------- | -------- |
| orientation  | row      | column   |
| encoding     | text     | binary   |
| compression  | none     | snappy   |
| schemas      | inferred | explicit |

Demos

Databases

# Text vs. Binary
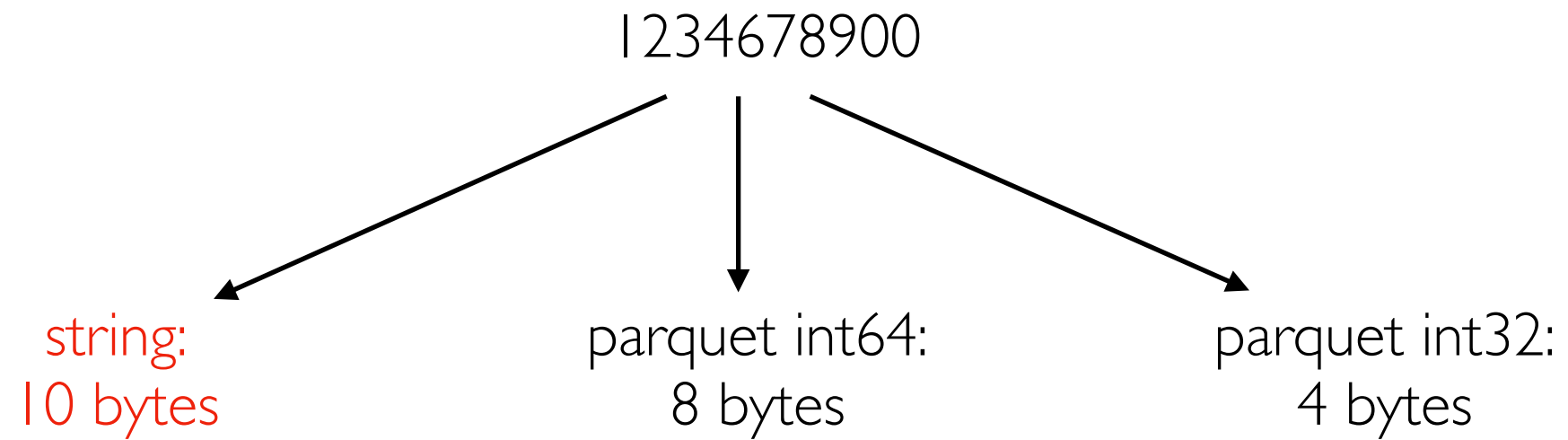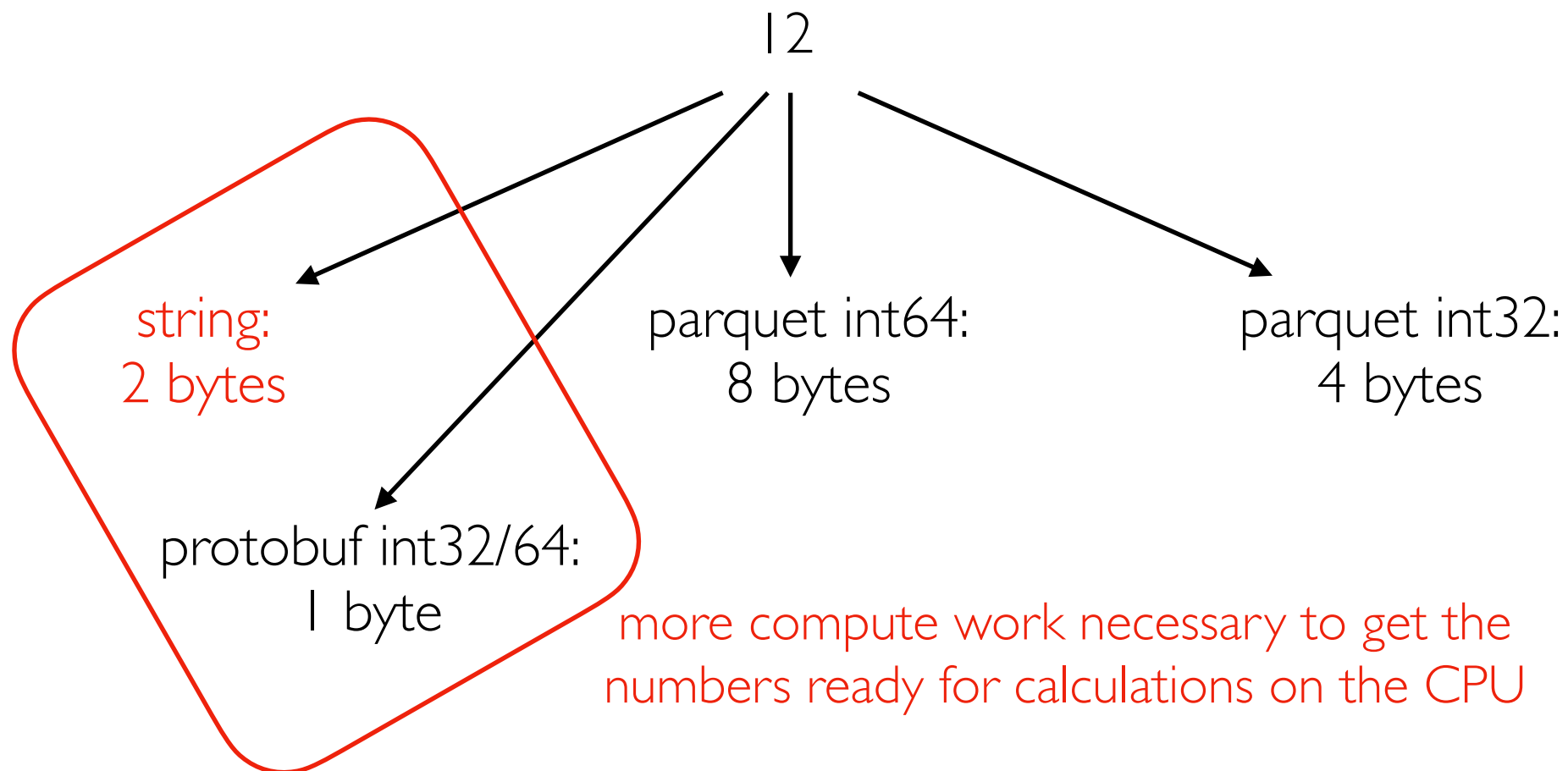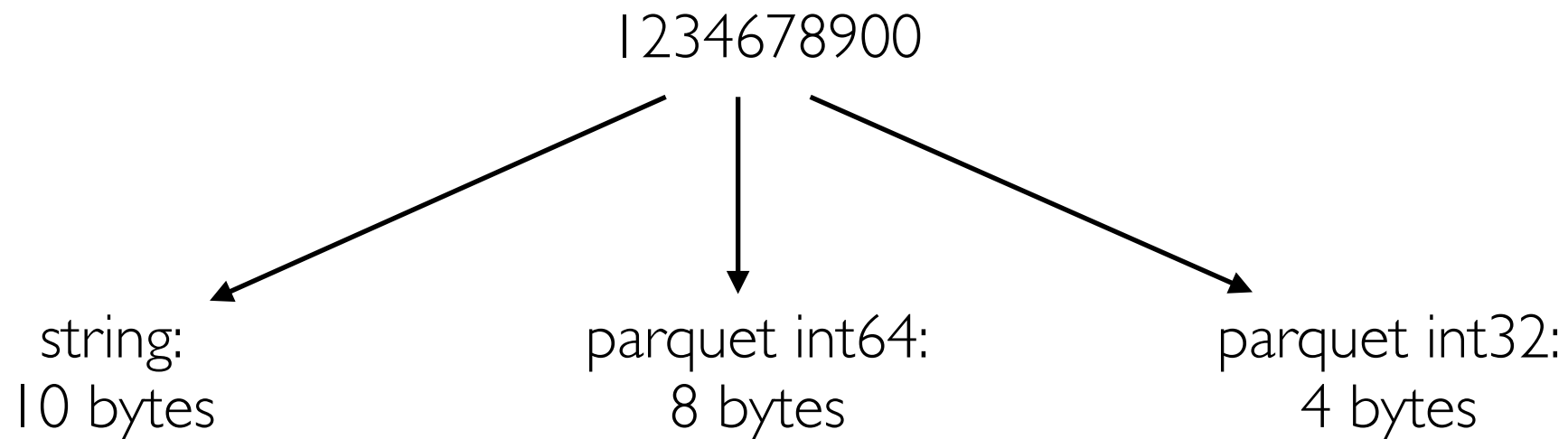
1234678900

string:
???? bytes

parquet int64:
???? bytes

parquet int32:
???? bytes

# Text vs. Binary

1234678900

string:
10 bytes

parquet int64:
8 bytes

parquet int32:
4 bytes

# Text vs. Binary

1234678900

string:
10 bytes

parquet int64:
8 bytes

parquet int32:
4 bytes

12

string:
2 bytes

protobuf int32/64:
1 byte

parquet int64:
8 bytes

parquet int32:
4 bytes

more compute work necessary to get the
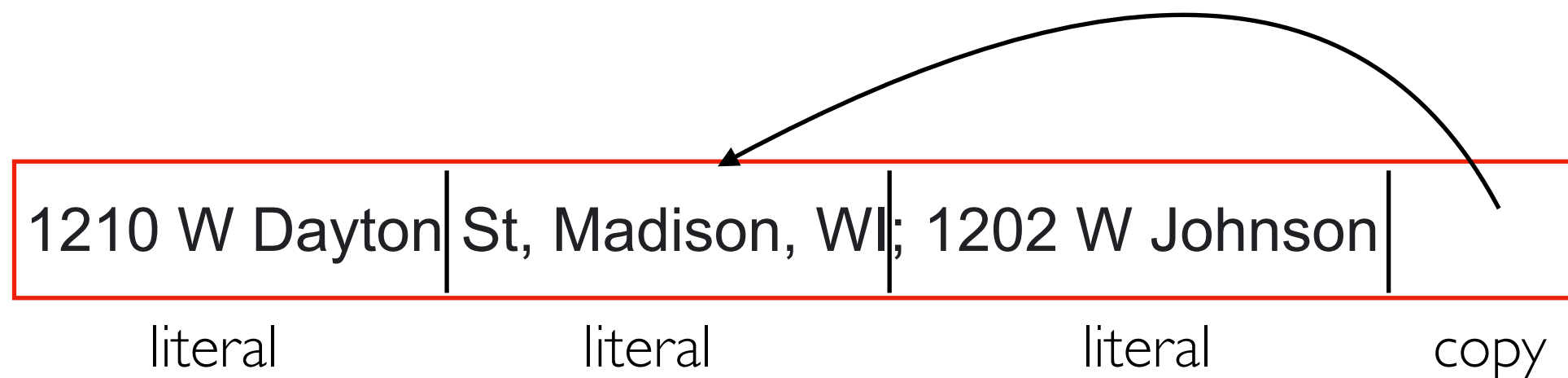numbers ready for calculations on the CPU

# Outline

# Compression

Idea: avoid repeating yourself
- repetitive datasets are more compressible
- more compute time finding repetition => better compression ratio (original/compressed size)

Example: snappy compression (parquet default):

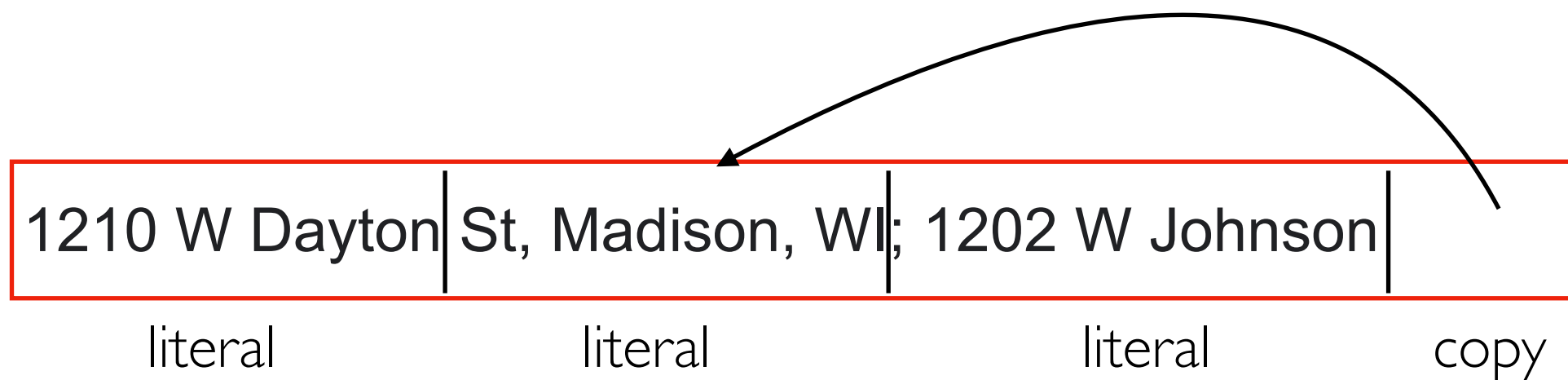| 1210 W Dayton | St, Madison, WI | ; 1202 W Johnson | |
|---|---|---|---|
| literal | literal | literal | copy |

"[Snappy] does not aim for maximum compression, or compatibility with any other compression library; instead, it aims for very high speeds and reasonable compression."

Snappy documentation
- https://github.com/google/snappy
- https://github.com/google/snappy/blob/main/format_description.txt

# Challenge: Small Updates

| 1210 W Dayton | St, Madison, WI | ; 1202 W Johnson | |
|:---:|:---:|:---:|:---:|
| literal | literal | literal | copy |

can't just update this first address in isolation
(need to rewrite other parts of the file)

# Compression Window/Block

"the current Snappy compressor works in 32 kB blocks and
does not do matching across blocks"

row-oriented file:　1 5 A 2 6 B 3 7 A 4 8 B

col-oriented file:　1 2 3 4 5 6 7 8 A B A B

→ position in file

will compression generally work better for row-oriented
formats or column-oriented formats?

# Outline

File Formats

|  | CSV | Parquet |
|---|---|---|
| **orientation** | row | column |
| **encoding** | text | binary |
| **compression** | none | snappy |
| **schemas** | inferred | explicit |

Demos

Databases

# Schemas

Schema: "A description of the structure of some data, including its fields and datatypes." -- Kleppmann

schema specified as a dict

CSVs:

- in the file, everything is text
- pd.read_csv("file.csv", dtype={"col1": str, "col2": int, ...})  # specify schema (annoying)
- pd.read_csv("file.csv", dtype=None)     # infer schema (slow, error prone!)

parquet files:

- type specification is part of the file
- fast: no need for very slow schema inference

# Outline

File Formats

Demos...

Databases

# Outline

File Formats

Demos...

<span style="color:red">Databases</span>
- <span style="color:red">tables and queries</span>
- architecture
- transactions vs. analytics

# Tables

| id | loan_purpose |
|----|-------------|
| 1 | Home purchase |
| 2 | Home improvement |
| 3 | Refinancing |

tbl_state

| code | abbr | name |
|------|------|------|
| 1 | AL | Alabama |
| 2 | AK | Alaska |
| 4 | AZ | Arizona |
| 5 | AR | Arkansas |
| 6 | CA | California |
| 8 | CO | Colorado |
| 9 | CT | Connecticut |
| 10 | DE | Delaware |
| ... | ... | ... |

string

tbl_action

| id | action_taken |
|----|-------------|
| 1 | Loan originated |
| 2 | Application approved but not accepted |
| 3 | Application denied by financial institution |
| 4 | Application withdrawn by applicant |
| 5 | File closed for incompleteness |
| 6 | Loan purchased by the institution |
| 7 | Preapproval request denied by financial |
| 8 | Preapproval request approved but not accepted |

tbl_loan

| id | purpose | action | state | amount | rate |
|----|---------|--------|-------|--------|------|
| 1 | 2 | 1 | 2 | 20000 | 5.0 |
| 2 | 1 | 1 | 8 | 300000 | 3.0 |
| 3 | 1 | 4 | 10 | 450000 | 3.2 |
| ... | ... | ... | ... | ... | ... |

int    float

Databases store a collection of tables

- schemas define the columns/types for each table
- IDs/keys let us relate multiple tables
  (for example, the first loan is in Alaska)

# Queries

### tbl_purpose

| id | loan_purpose |
|---|---|
| 1 | Home purchase |
| 2 | Home improvement |
| 3 | Refinancing |

### tbl_state

| code | abbr | name |
|---|---|---|
| 1 | AL | Alabama |
| 2 | AK | Alaska |
| 4 | AZ | Arizona |
| 5 | AR | Arkansas |
| 6 | CA | California |
| 8 | CO | Colorado |
| 9 | CT | Connecticut |
| 10 | DE | Delaware |
| ... | ... | ... |

### tbl_action

| id | action_taken |
|---|---|
| 1 | Loan originated |
| 2 | Application approved but not accepted |
| 3 | Application denied by financial institution |
| 4 | Application withdrawn by applicant |
| 5 | File closed for incompleteness |
| 6 | Loan purchased by the institution |
| 7 | Preapproval request denied by financial |
| 8 | Preapproval request approved but not accepted |

### tbl_loan

| id | purpose | action | state | amount | rate |
|---|---|---|---|---|---|
| 1 | 2 | 1 | 2 | 20000 | 5.0 |
| 2 | 1 | 1 | 8 | 300000 | 3.0 |
| 3 | 1 | 4 | 10 | 450000 | 3.2 |
| ... | ... | ... | ... | ... | ... |

## Queries let us

- ask questions about the data
  (like, what is the name of the state with "WI" as an abbreviation)
- make changes to the data
  (like insert Puerto Rico as a row in tbl_state)

# SQL

### tbl_purpose

| id | loan_purpose |
|----|--------------|
| 1 | Home purchase |
| 2 | Home improvement |
| 3 | Refinancing |

### tbl_state

| code | abbr | name |
|------|------|------|
| 1 | AL | Alabama |
| 2 | AK | Alaska |
| 4 | AZ | Arizona |
| 5 | AR | Arkansas |
| 6 | CA | California |
| 8 | CO | Colorado |
| 9 | CT | Connecticut |
| 10 | DE | Delaware |
| ... | ... | ... |

### tbl_action

| id | action_taken |
|----|--------------|
| 1 | Loan originated |
| 2 | Application approved but not accepted |
| 3 | Application denied by financial institution |
| 4 | Application withdrawn by applicant |
| 5 | File closed for incompleteness |
| 6 | Loan purchased by the institution |
| 7 | Preapproval request denied by financial |
| 8 | Preapproval request approved but not accepted |

### tbl_loan

| id | purpose | action | state | amount | rate |
|----|---------|--------|-------|--------|------|
| 1 | 2 | 1 | 2 | 20000 | 5.0 |
| 2 | 1 | 1 | 8 | 300000 | 3.0 |
| 3 | 1 | 4 | 10 | 450000 | 3.2 |
| ... | ... | ... | ... | ... | ... |

Structure Query Language (SQL)
- most popular/famous query language
- ask questions about the data: SELECT
- make changes to the data: INSERT, UPDATE, DELETE

# SQL

## tbl_purpose

| id | loan_purpose |
|----|----|
| 1 | Home purchase |
| 2 | Home improvement |
| 3 | Refinancing |

## tbl_state

| code | abbr | name |
|----|----|----|
| 1 | AL | Alabama |
| 2 | AK | Alaska |
| 4 | AZ | Arizona |
| 5 | AR | Arkansas |
| 6 | CA | California |
| 8 | CO | Colorado |
| 9 | CT | Connecticut |
| 10 | DE | Delaware |
| ... | ... | ... |

## tbl_action

| id | action_taken |
|----|----|
| 1 | Loan originated |
| 2 | Application approved but not accepted |
| 3 | Application denied by financial institution |
| 4 | Application withdrawn by applicant |
| 5 | File closed for incompleteness |
| 6 | Loan purchased by the institution |
| 7 | Preapproval request denied by financial |
| 8 | Preapproval request approved but not accepted |

## tbl_loan

| id | purpose | action | state | amount | rate |
|----|----|----|----|----|----|
| 1 | 2 | 1 | 2 | 20000 | 5.0 |
| 2 | 1 | 1 | 8 | 300000 | 3.0 |
| 3 | 1 | 4 | 10 | 450000 | 3.2 |
| ... | ... | ... | ... | ... | ... |

Structure Query Language (SQL)

- most popular/famous query language
- ask questions about the data: SELECT
- make changes to the data: INSERT, UPDATE, DELETE

SELECT AVG(rate) FROM tbl_loan;

*analytics (calculate over many/all rows, few colums)*

SELECT amount, rate FROM tbl_loan WHERE id = 544;

INSERT INTO tbl_loan (...) VALUES (...);

*transactions (working with whole row
or few rows at a time)*

# Outline

File Formats

Demos

Databases
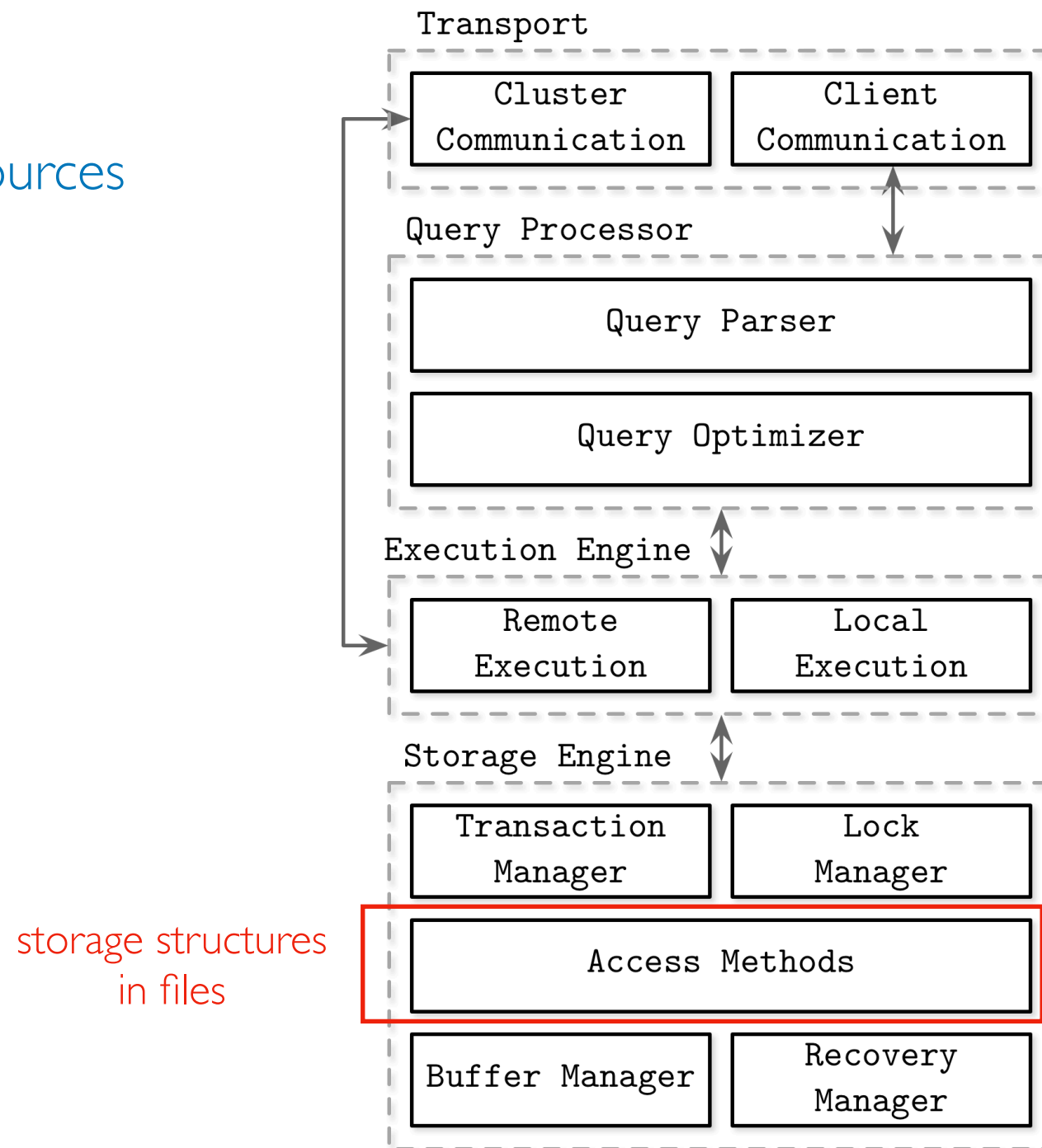- tables and queries
- architecture
- transactions vs. analytics

# Data Base Management Systems

Architecture: *big picture of a system's components/subsystems*

Databases manage all the resources we've learned about:

- storage
- memory
- network
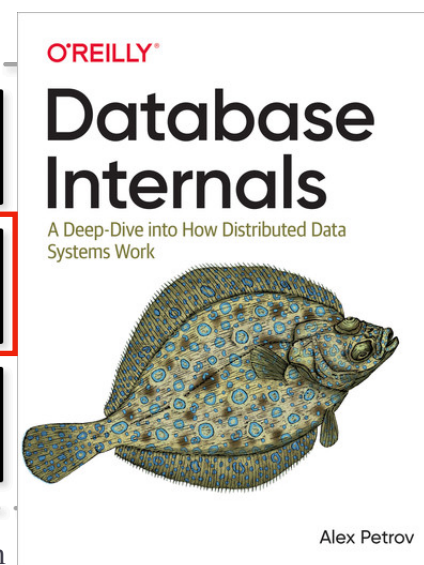- compute

example database architecture:

Transport

| Cluster Communication | Client Communication |

Query Processor

| Query Parser |

| Query Optimizer |

Execution Engine

| Remote Execution | Local Execution |

Storage Engine

| Transaction Manager | Lock Manager |

| Access Methods |

storage structures in files

| Buffer Manager | Recovery Manager |

Figure 1-1. Architecture of a database management system (Chapter 1 of Database Internals, by Petrov)

O'REILLY®
Database Internals
A Deep-Dive into How Distributed Data Systems Work

Alex Petrov

# Data Base Management Systems

Architecture: *big picture of a system's components/subsystems*

Databases manage all the resources we've learned about:

- storage
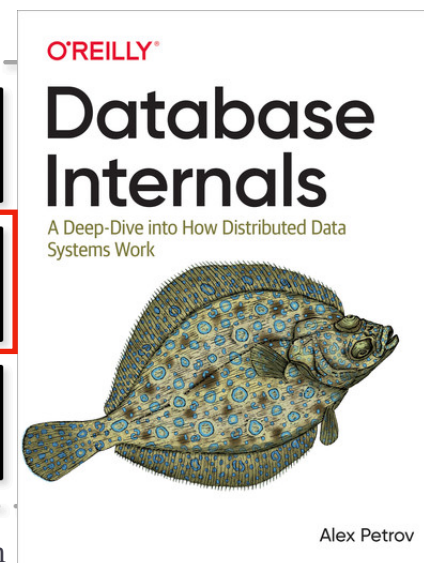- memory
- network
- compute

example database architecture:

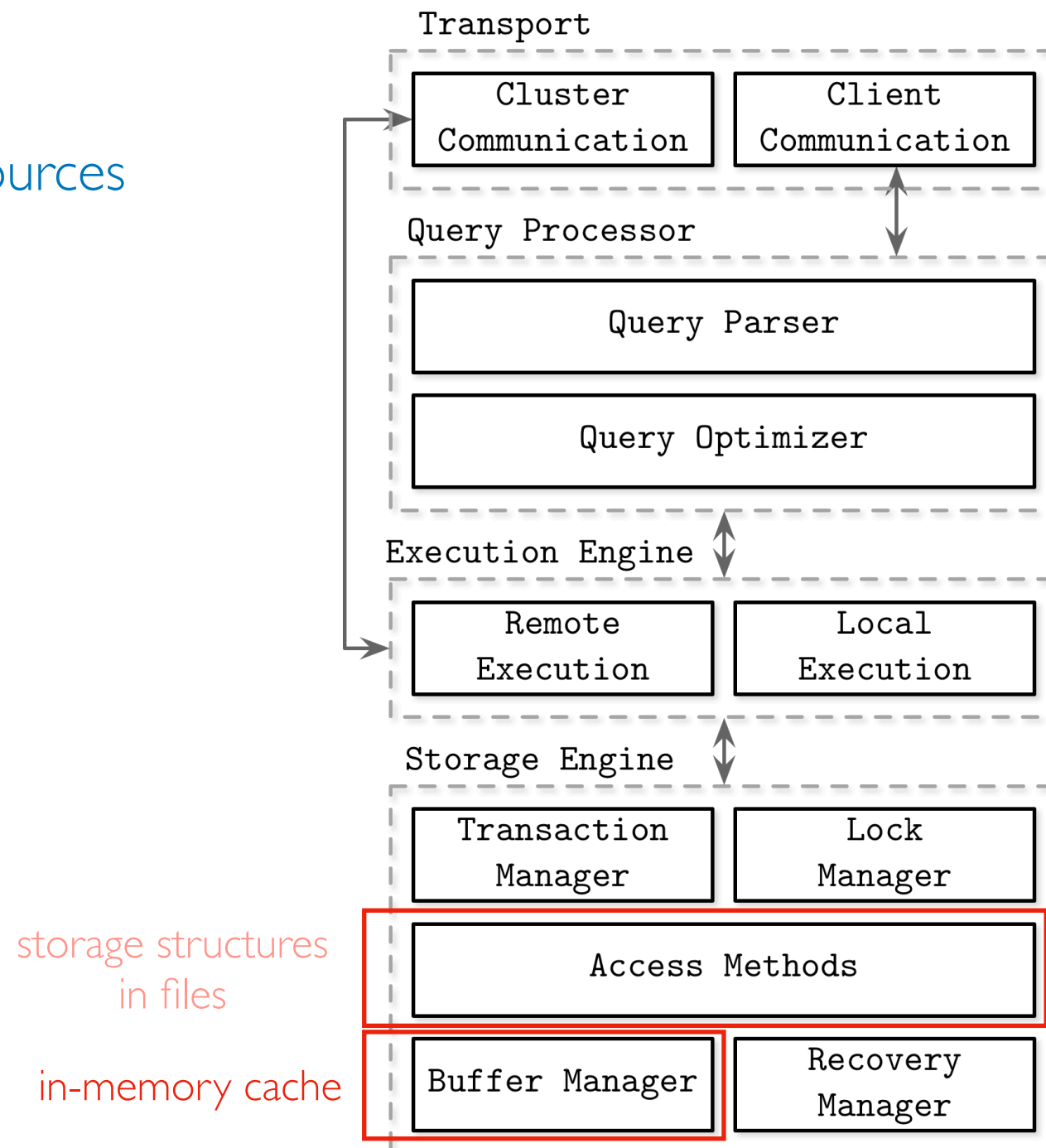Transport

| Cluster Communication | Client Communication |

Query Processor

| Query Parser |

| Query Optimizer |

Execution Engine

| Remote Execution | Local Execution |

Storage Engine

| Transaction Manager | Lock Manager |

| Access Methods |

storage structures in files

| Buffer Manager | Recovery Manager |

in-memory cache

Figure 1-1. Architecture of a database management system (Chapter 1 of Database Internals, by Petrov)

O'REILLY®

## Database Internals

A Deep-Dive into How Distributed Data Systems Work

Alex Petrov

# Data Base Management Systems

Architecture: *big picture of a system's components/subsystems*

Databases manage all the resources we've learned about:

- storage
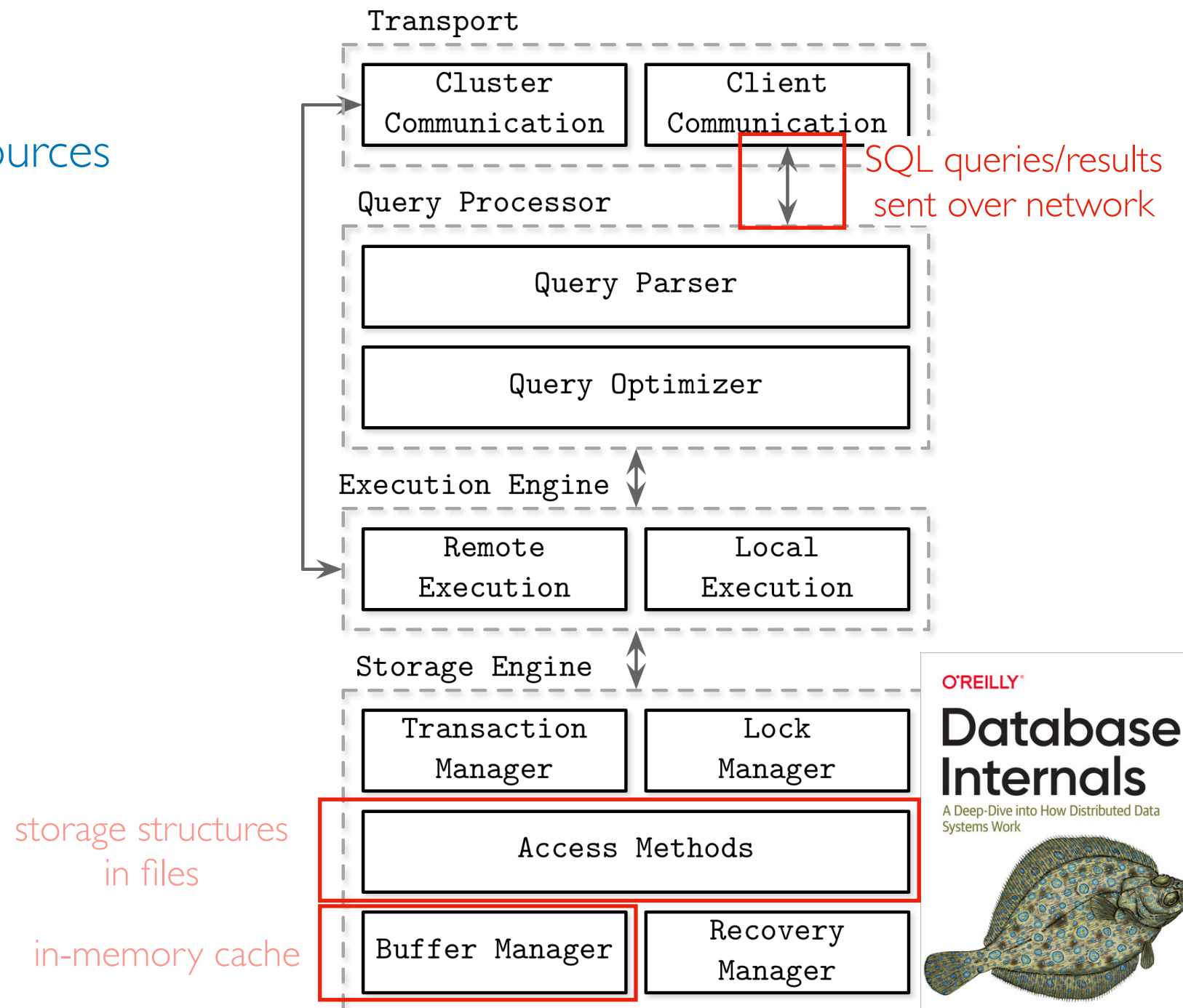- memory
- network
- compute

example database architecture:

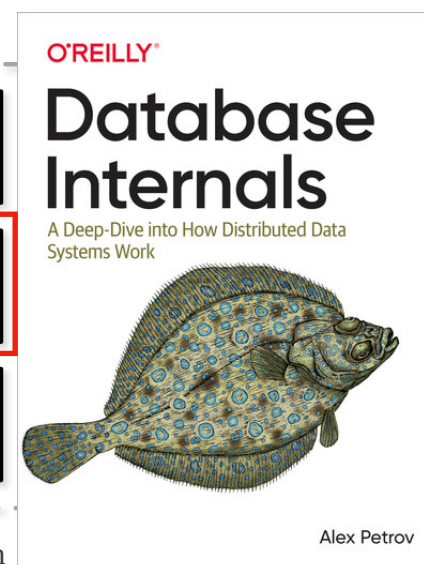Transport

| Cluster Communication | Client Communication |
|---|---|

SQL queries/results sent over network

Query Processor

| Query Parser |
|---|

| Query Optimizer |
|---|

Execution Engine

| Remote Execution | Local Execution |
|---|---|

Storage Engine

| Transaction Manager | Lock Manager |
|---|---|

storage structures in files

| Access Methods |
|---|

in-memory cache

| Buffer Manager | Recovery Manager |
|---|---|

Figure 1-1. Architecture of a database management system (Chapter 1 of Database Internals, by Petrov)

O'REILLY®

# Database Internals

A Deep-Dive into How Distributed Data Systems Work

Alex Petrov

# Data Base Management Systems

Architecture: *big picture of a system's components/subsystems*

Databases manage all the resources we've learned about:

- storage
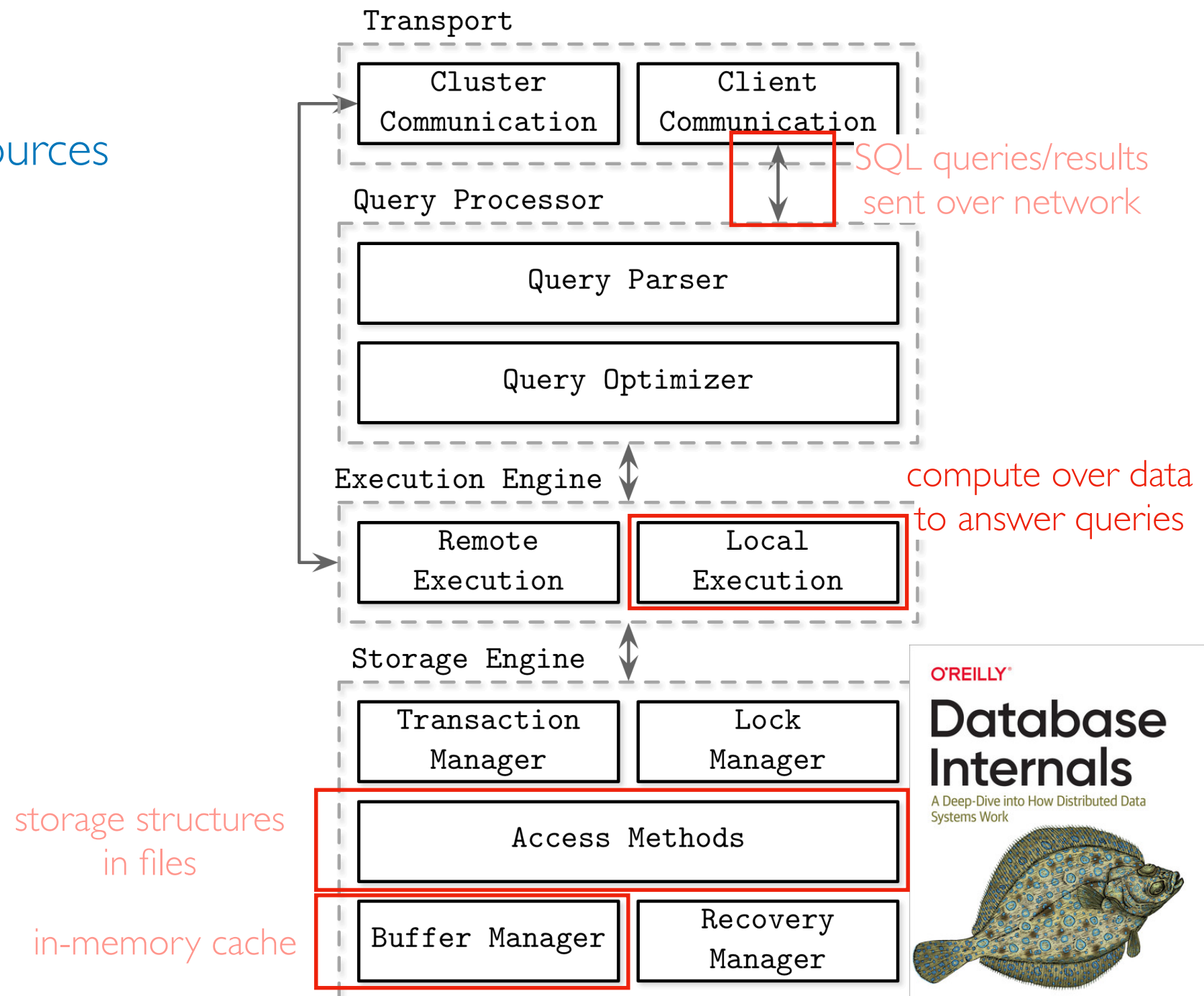- memory
- network
- compute

example database architecture:

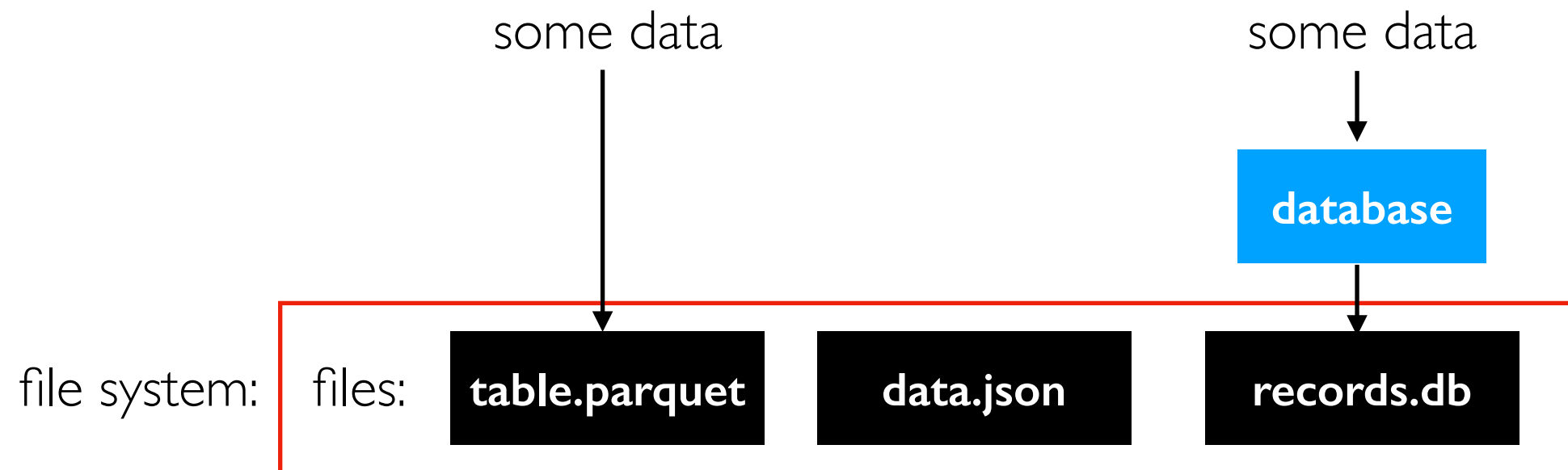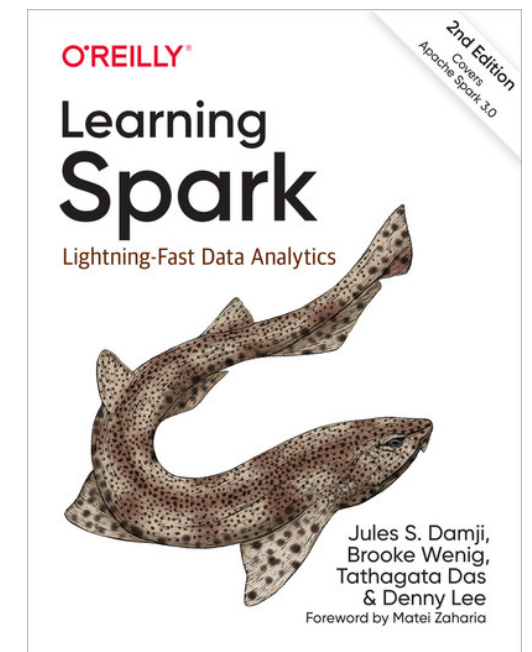Transport

| Cluster Communication | Client Communication |
|---|---|

SQL queries/results sent over network

Query Processor

| Query Parser |
|---|

| Query Optimizer |
|---|

Execution Engine

compute over data to answer queries

| Remote Execution | Local Execution |
|---|---|

Storage Engine

| Transaction Manager | Lock Manager |
|---|---|

storage structures in files

| Access Methods |
|---|

in-memory cache

| Buffer Manager | Recovery Manager |
|---|---|

Figure 1-1. Architecture of a database management system
(Chapter 1 of Database Internals, by Petrov)

O'REILLY®
Database Internals
A Deep-Dive into How Distributed Data Systems Work

Alex Petrov

# Files vs. Databases (storage+compute coupling)

some data

some data

**database**

file system:    files:    **table.parquet**    **data.json**    **records.db**

Databases pros/cons (relative to just using files):

- *"[databases]* tightly couple *their internal layout of the data and indexes in on-disk files with their highly optimized query processing engines, thus providing* very fast computations *on the stored data..."*

- *"Databases store data in complex (often proprietary) formats that are typically highly optimized for only that database's SQL processing engine to read. This means* other processing tools, like machine learning and deep learning systems, cannot efficiently access the data *(except by inefficiently reading all the data from the database)."*

O'REILLY®

2nd Edition
Covers
Apache Spark 3.0

Learning
Spark

Lightning-Fast Data Analytics

Jules S. Damji,
Brooke Wenig,
Tathagata Das
& Denny Lee

Foreword by Matei Zaharia

# Outline

File Formats

Demos

<span style="color:red">Databases</span>
- tables and queries
- architecture
- <span style="color:red">transactions vs. analytics</span>

# Transactions vs. Analytics

```
SELECT AVG(rate) FROM tbl_loan;
```
*analytics (calculate over many/all rows, few colums)*

```
SELECT amount, rate FROM tbl_loan WHERE id = 544;
INSERT INTO tbl_loan (...) VALUES (...);
```
*transactions (working with whole row or few rows at a time)*

SQL (as a language) works great for both transactions and analytics

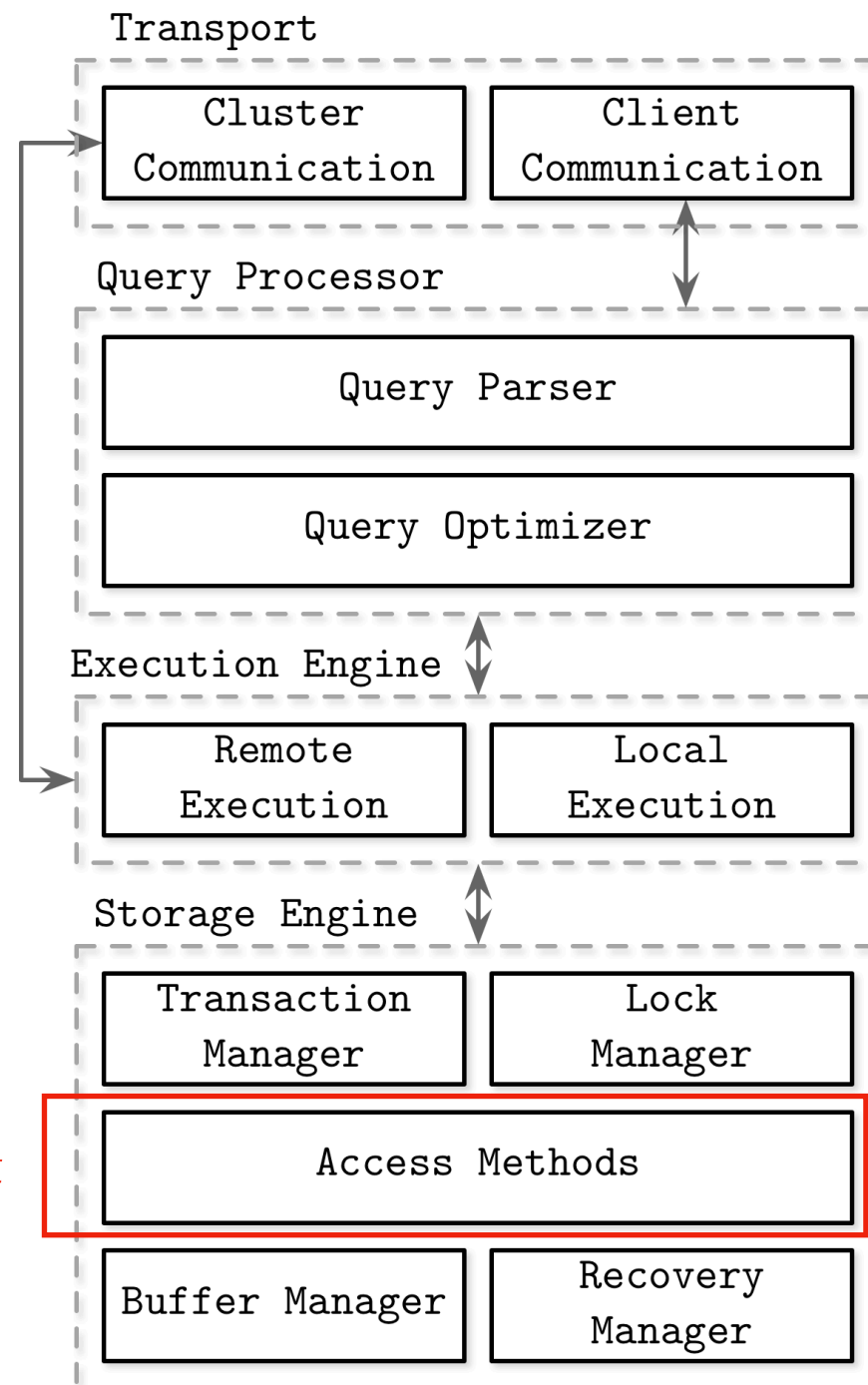Problem: it's hard for a single database (SQL or otherwise) to be good at both

Main database types:
- OL**T**P (online transactions processing)
- OL**A**P (online analytics processing)

*"The meaning of online in OLAP is unclear; it probably refers to the fact that queries are not just for predefined reports, but that analysts use the OLAP system interactively for explorative queries." ~ Kleppmann.*

# Transactions vs. Analytics

example database architecture:

Transport

| Cluster Communication | Client Communication |

Query Processor

| Query Parser |

| Query Optimizer |

Execution Engine

| Remote Execution | Local Execution |

Storage Engine

| Transaction Manager | Lock Manager |

| Access Methods |

| Buffer Manager | Recovery Manager |

**Typical storage design**
OLTP: *row oriented data layout*
OLAP: *col oriented data layout*
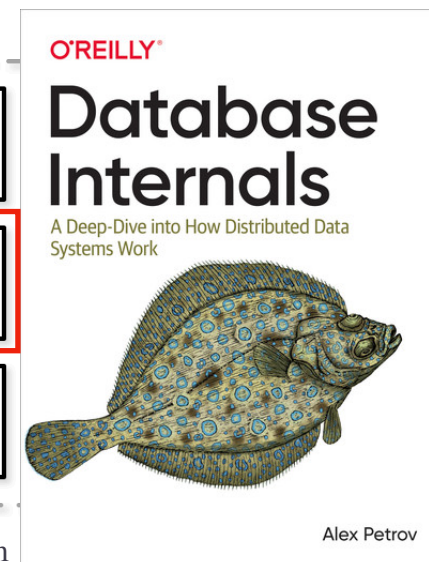
Figure 1-1. Architecture of a database management system
(Chapter 1 of Database Internals, by Petrov)

O'REILLY®

# Database Internals

A Deep-Dive into How Distributed Data Systems Work

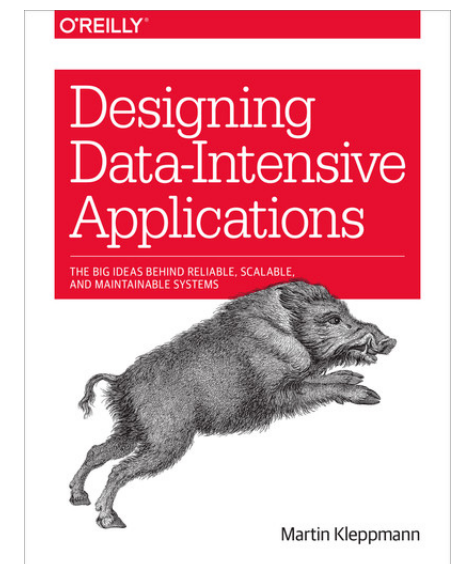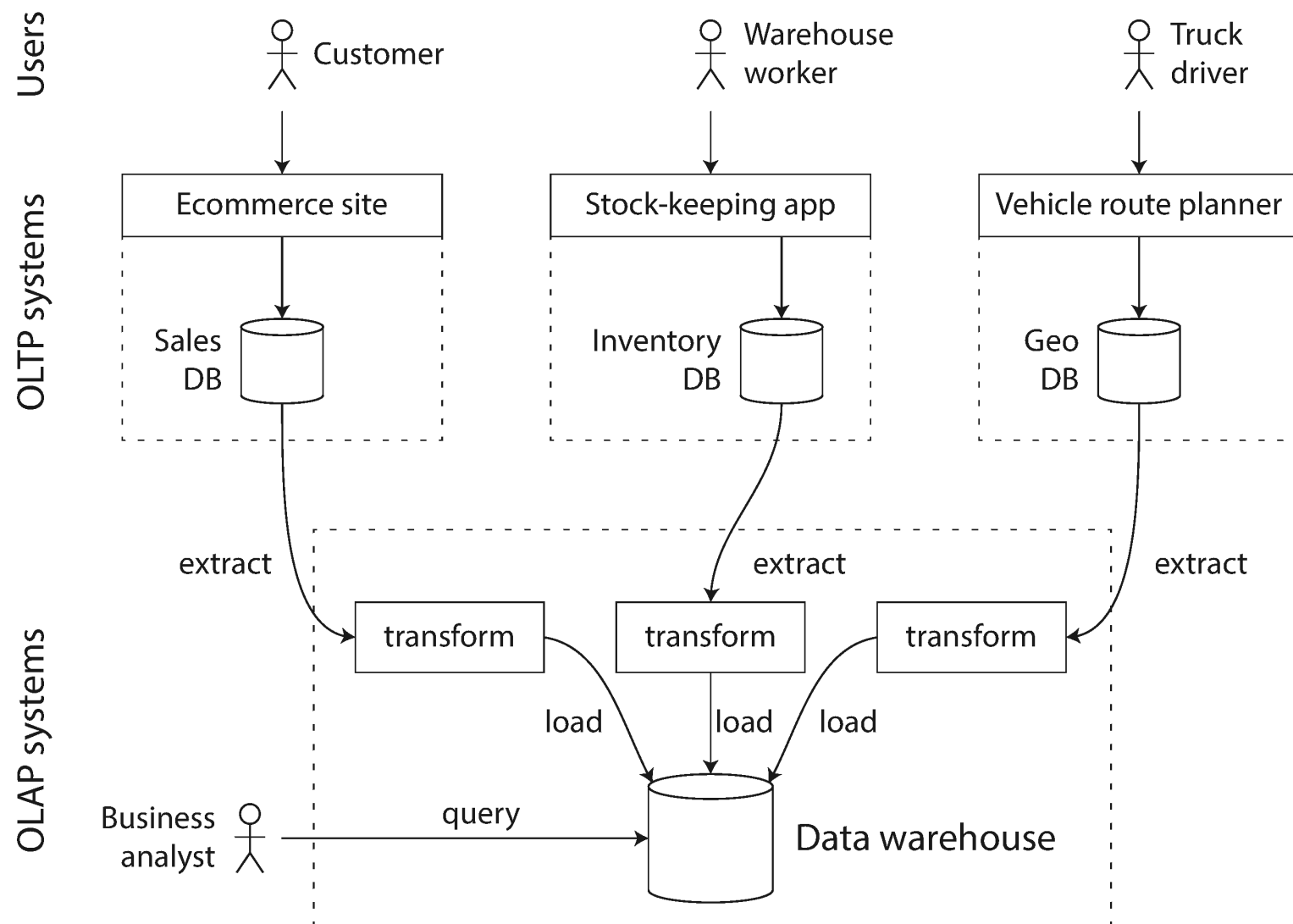Alex Petrov

# What if you need transactions AND analytics?



Figure 3-8. Simplified outline of ETL into a data warehouse.
(Chapter 3 of Data-Intensive Applications, by Kleppmann)

Vocab

- Data warehouse: *the OLAP database where we combine data from many sources*
- ETL: *extract-transform-load (process for getting data out of OLTP DBs and into OLAP DB)*