# Course Review

Instructor: Matei Zaharia

cs245.stanford.edu

# What Are Data-Intensive Systems?

**Relational databases:** most popular type of data-intensive system (MySQL, Oracle, etc)

**Many systems facing similar concerns:** message queues, key-value stores, streaming systems, ML frameworks, your custom app?

**Goal:** learn the main issues and principles that span all data-intensive systems

# Typical System Challenges

**Reliability** in the face of hardware crashes, bugs, bad user input, etc

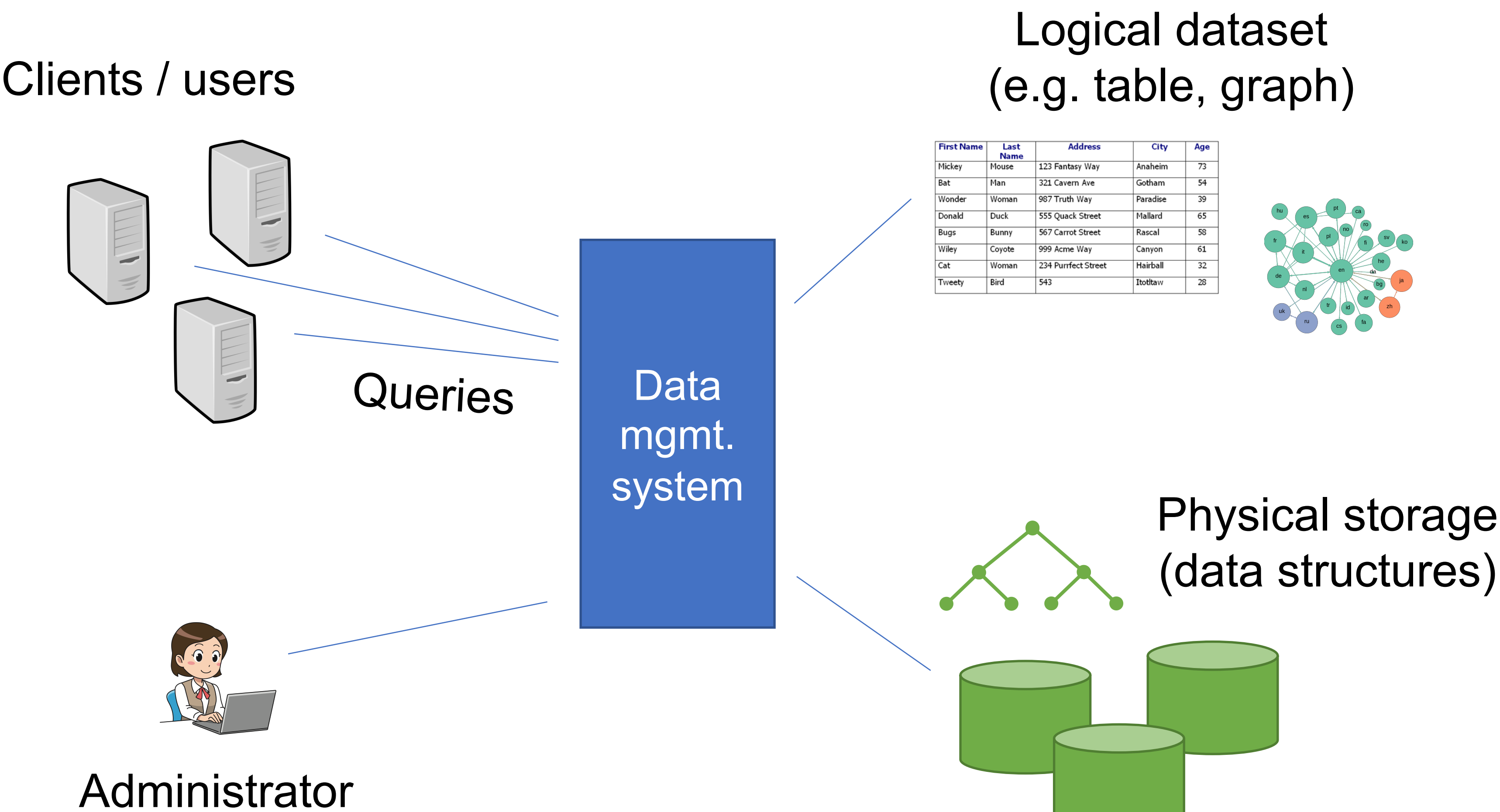
**Concurrency:** access by multiple users

**Performance:** throughput, latency, etc

**Access interface** from many, changing apps

**Security** and data privacy

# Basic Components




| First Name | Last Name | Address | City | Age |
|---|---|---|---|---|
| Mickey | Mouse | 123 Fantasy Way | Anaheim | 73 |
| Bat | Man | 321 Cavern Ave | Gotham | 54 |
| Wonder | Woman | 987 Truth Way | Paradise | 39 |
| Donald | Duck | 555 Quack Street | Mallard | 65 |
| Bugs | Bunny | 567 Carrot Street | Rascal | 58 |
| Wiley | Coyote | 999 Acme Way | Canyon | 61 |
| Cat | Woman | 234 Purrfect Street | Hairball | 32 |
| Tweety | Bird | 543 | Itotltaw | 28 |

# Two Big Ideas

## Declarative interfaces

» Apps specify *what* they want, not *how* to do it

» Example: “store a table with 2 integer columns”, but not how to encode it on disk

» Example: “count records where column1 = 5”

## Transactions

» Encapsulate multiple app actions into one *atomic* request (fails or succeeds as a whole)

» Concurrency models for multiple users

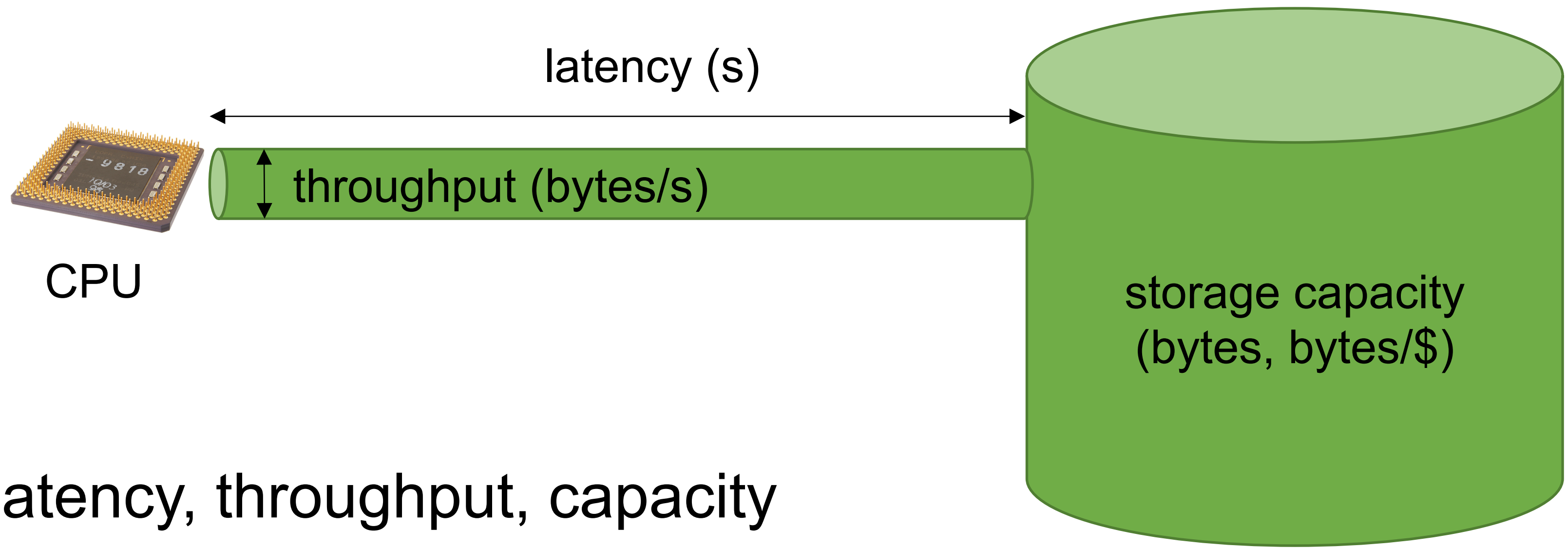
» Clear interactions with failure recovery

# Key Concepts: Architecture

**Traditional RDBMS:** self-contained end to end system

**Data lake:** separate storage from compute engines to let many engines use same data

# Key Concepts: Hardware




Latency, throughput, capacity

Random vs sequential I/Os

Caching & 5-minute rule

# Key Concepts: Data Storage

**Field encoding**

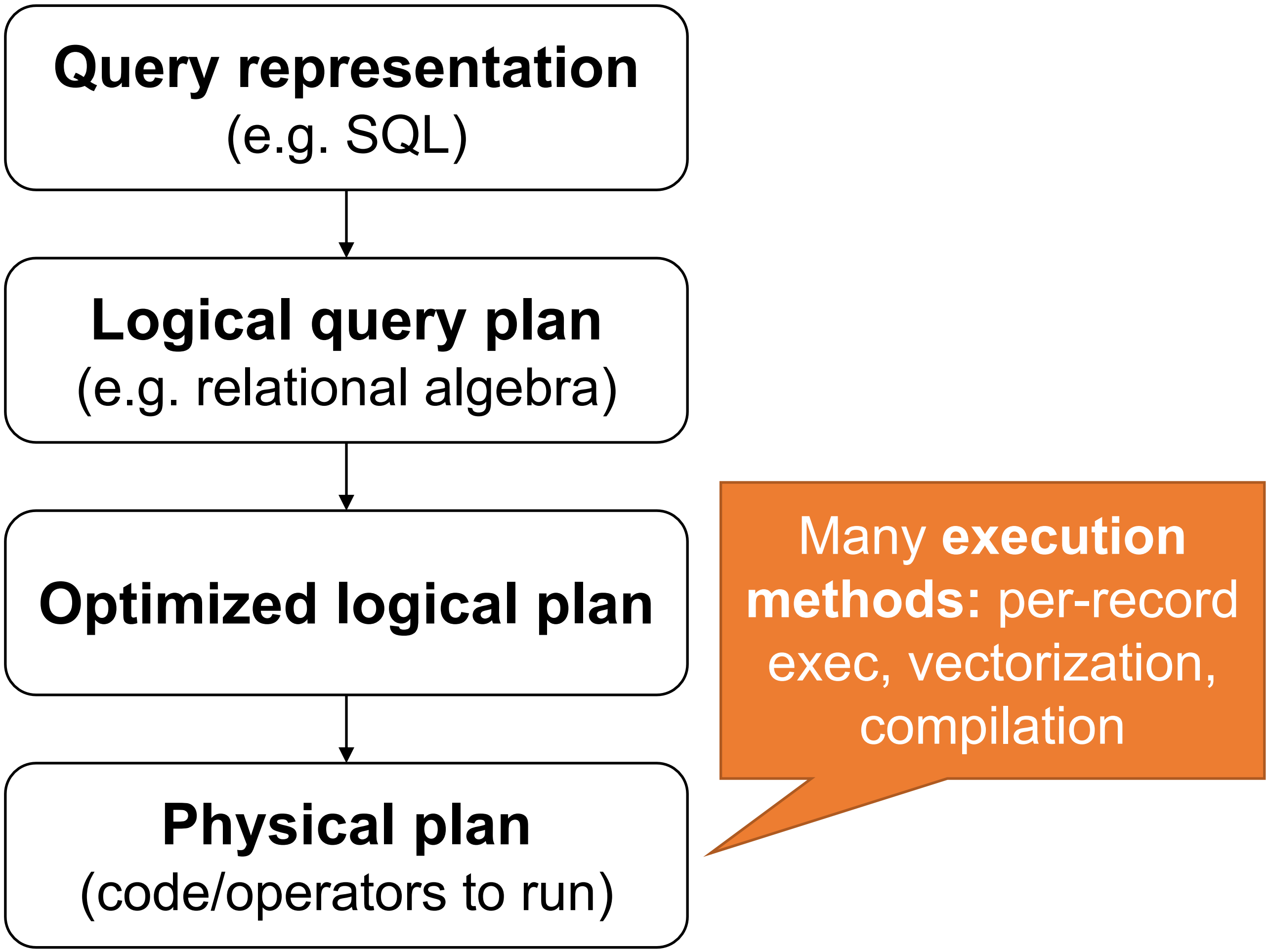
**Record encoding:** fixed/variable format, etc

**Table encoding:** row or column oriented

**Data ordering**

**Indexes:** dense, sparse, B+ trees, hashing, multi-dimensional

# Key Concepts: Query Execution



# Key Concepts: Relational Algebra

$\cap$, $\cup$, $-$, $\times$, $\sigma$, $\Pi$, $\bowtie$, $G$

Algebraic rules involving these

# Key Concepts: Optimization

**Rule-based:** systematically replace some expressions with other expressions

**Cost-based:** propose several execution plans and pick best based on a **cost model**

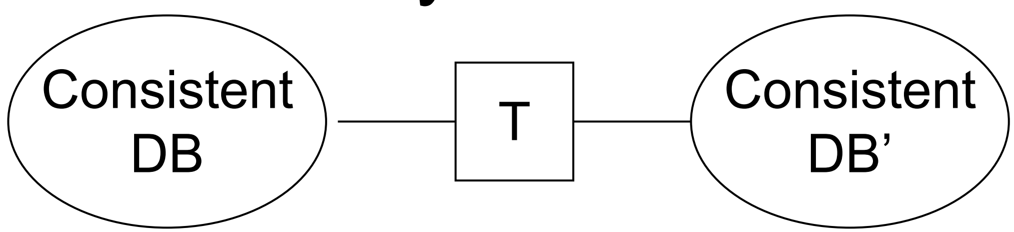
**Adaptive:** update execution plan at runtime

**Data statistics:** can be computed or estimated cheaply to guide decisions

# Key Concepts: Correctness

**Consistency constraints:** generic way to define correctness with Boolean predicates

**Transaction:** collection of actions that preserve consistency




**Transaction API:** commit, abort, etc

# Key Concepts: Recovery

**Failure models**

**Undo, redo, and undo/redo logging**

**Recovery rules** for various algorithms (including handling crashes during recovery)

**Checkpointing** and its effect on recovery

**External actions** → idempotence, 2PC

# Key Concepts: Concurrency

**Isolation levels**, especially **serializability**

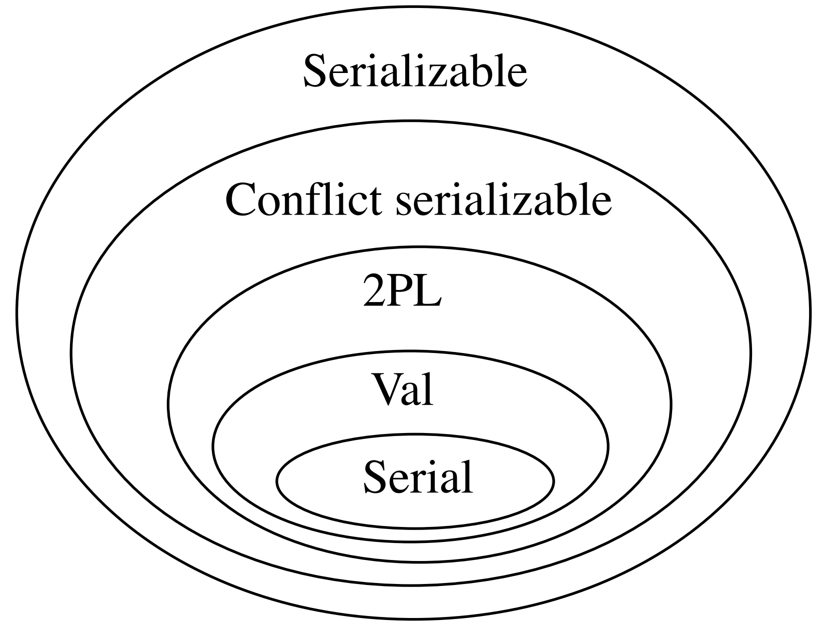
» Testing for serializability: conflict serializability, precedence graphs

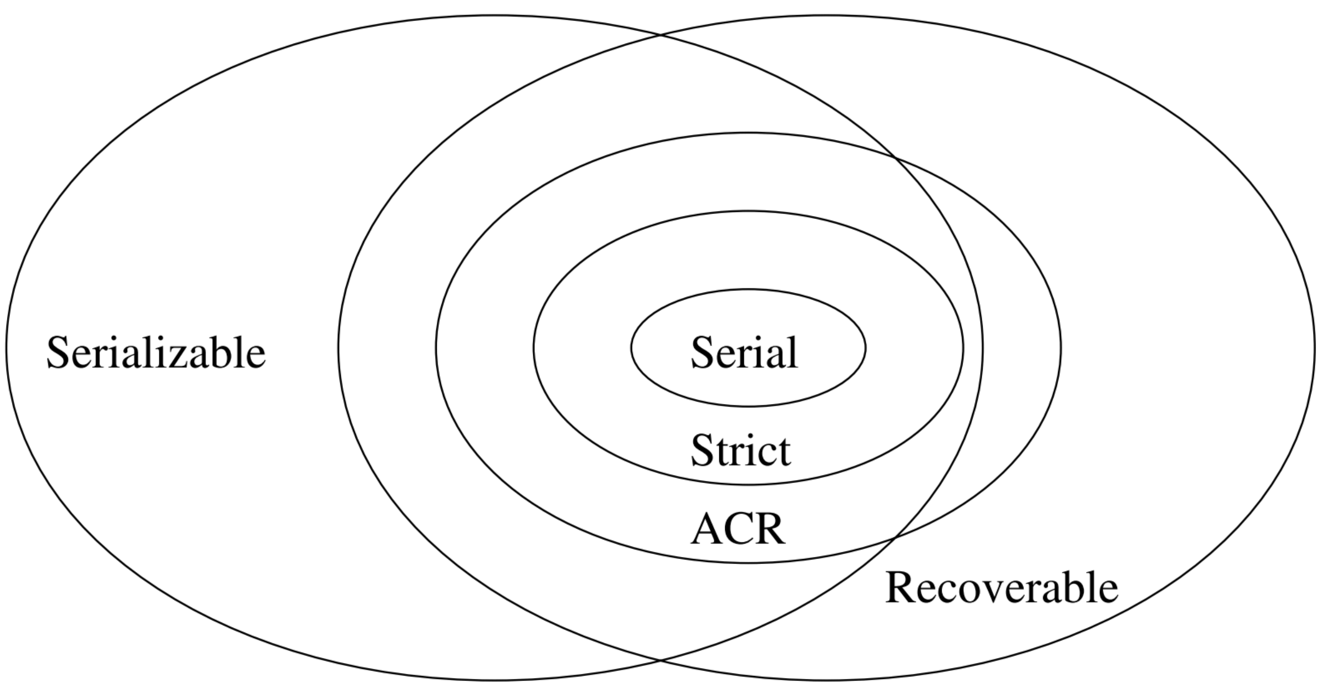
**Locking:** lock modes, hierarchical locks, and lock schedules (well formed, legal, 2PL)

**Optimistic validation:** rules and pros+cons

**Recoverable, ACR & strict** schedules

# Categories of Schedules



# Key Concepts: Distributed

**Partitioning** and **replication**

**Consensus:** nodes eventually agree on one value despite up to F failures

**2-Phase commit:** parties all agree to commit unless one aborts (no permanent failures)

**Parallel queries:** comm cost, load balance, faults

**BASE** and relaxing consistency

# Key Concepts: Security and Data Privacy

**Threat models**

**Security goals:** authentication, authorization, auditing, confidentiality, integrity etc

**Differential privacy:** definitions, computing sensitivity & stability

# Putting These Concepts Together

How can you integrate these different concepts into a coherent system design?

How to change system to meet various goals (performance, concurrency, security, etc)?

# Send Us Your Feedback!

We want to keep improving the course and tuning the content, so write a course eval