Lecture 1:
Course Introduction

Welcome to CS 4423/6423!

- Joy Arulraj (School of Computer Science)
- This course is all about building database systems.
- Why do we even care about database systems?

Importance of Database Systems

- Banking
- Healthcare
- Airlines
- E-Commerce

Why take this course?

- Curiosity
- Scalability
- Efficiency
- Versatility

Why take this course?

- Storage Management
- Query Optimization
- Index Structures
- SIMD Instructions

Course Overview

Course Objectives

- Learn about building a database system from scratch.
- Become proficient in systems programming.
- Understand the impact of hardware trends on software design.

Course Topics

- This course focuses on the internals of a database system:
  - Logging and Recovery
  - Concurrency Control
  - Query Optimization
  - Potpourri of advanced topics

Previous Course (4420/6422)

- This course builds upon a prior course that covered:
  - Relational Databases
  - Storage Management
  - Index Structures
  - Query Execution

Expected Background

- Should have taken an introductory course on computer systems.
- All programming assignments will be in C++.
  - Programming assignment #1 will help get you caught up with C++.
  - If you have not encountered C++ before, need to put in extra effort.
  - Use a large language model like ChatGPT for assistance.
  - Relevant parts of C++ will be briefly covered in this course.

Course Logistics

- Course Website (link on Canvas)
- Discussion Tool: Ed (link on Canvas)
- Grading Tool: Gradescope (link on Canvas)
- In-Class Quiz Tool: Point Solutions (link on Canvas)

Course Rubric

- Exams (50%)
- Programming Assignments (20%)
- Exercise Sheets (15%)
- In-Class Quizzes (15%)
- Extra-Credit Project (+10%)

Course Policies

- Programming assignments & exercise sheets must be own work.
  - Not group assignments.
  - You may not copy source code from other people or the web.
  - Plagiarism will not be tolerated.
  - We will follow the late submission policy listed on Canvas.
- Academic Honesty
  - Refer to Georgia Tech Academic Honor Code.
  - If you are not sure, ask me.

Textbooks for Reference

- Silberschatz, Korth, & Sudarshan:
  - Database System Concepts. McGraw Hill, 2020.
- Hector Garcia-Molina, Jeff Ullman, and Jennifer Widom:
  - Database Systems: The Complete Book. Prentice-Hall, 2008.

Intro Sheet

- Upload a one-page PDF with your details on Gradescope.
  - Picture (ideally 2x2 inches of face).
  - Name, interests, and other details mentioned on Gradescope.
- Purpose of this sheet
  - Help me know more about your background for tailoring the course.
  - Recognize you in class.

In-Person Office Hours

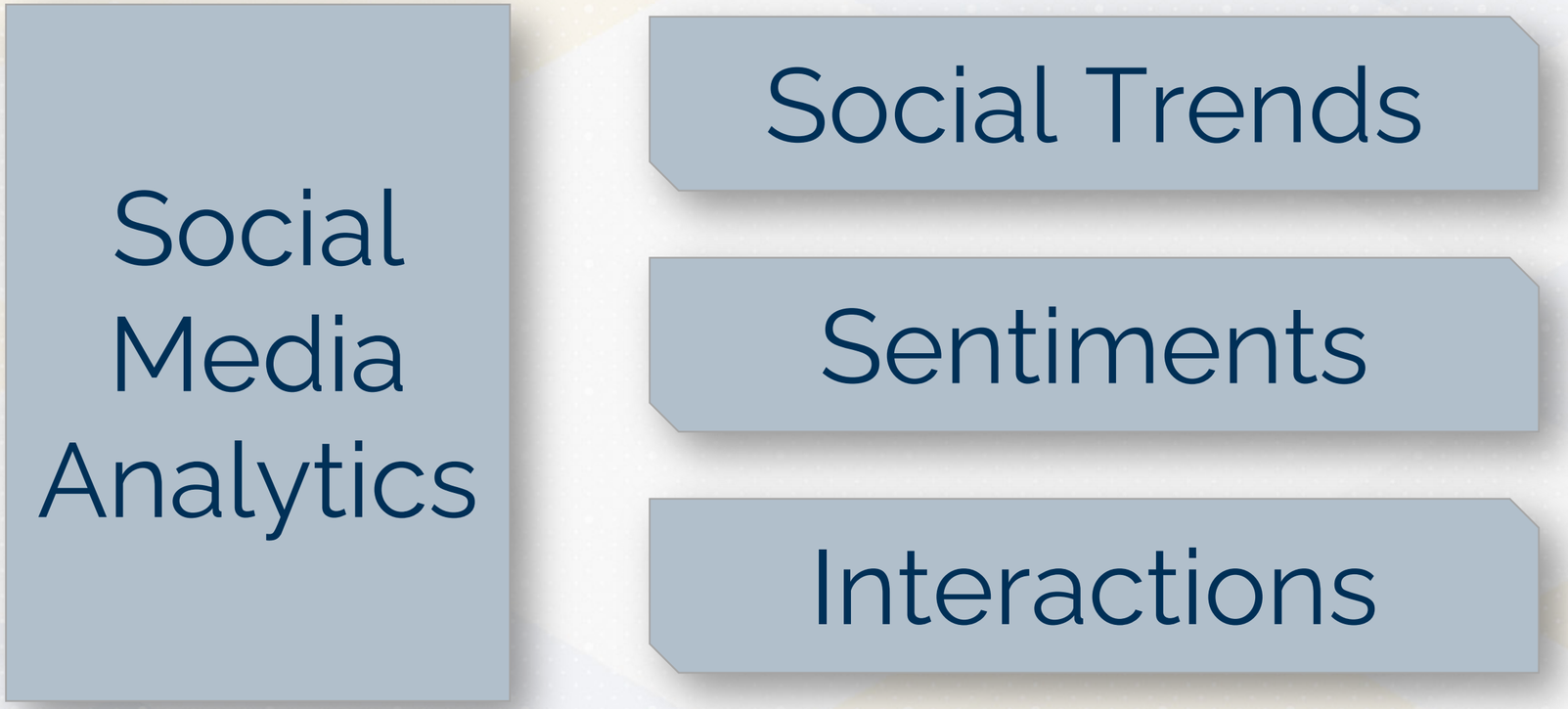
- Sign up for a ten-minute slot in the sign-up sheet (link on Canvas)
- Teaching assistants will guide you with assignments & sheets.

Auditing + Late Policy

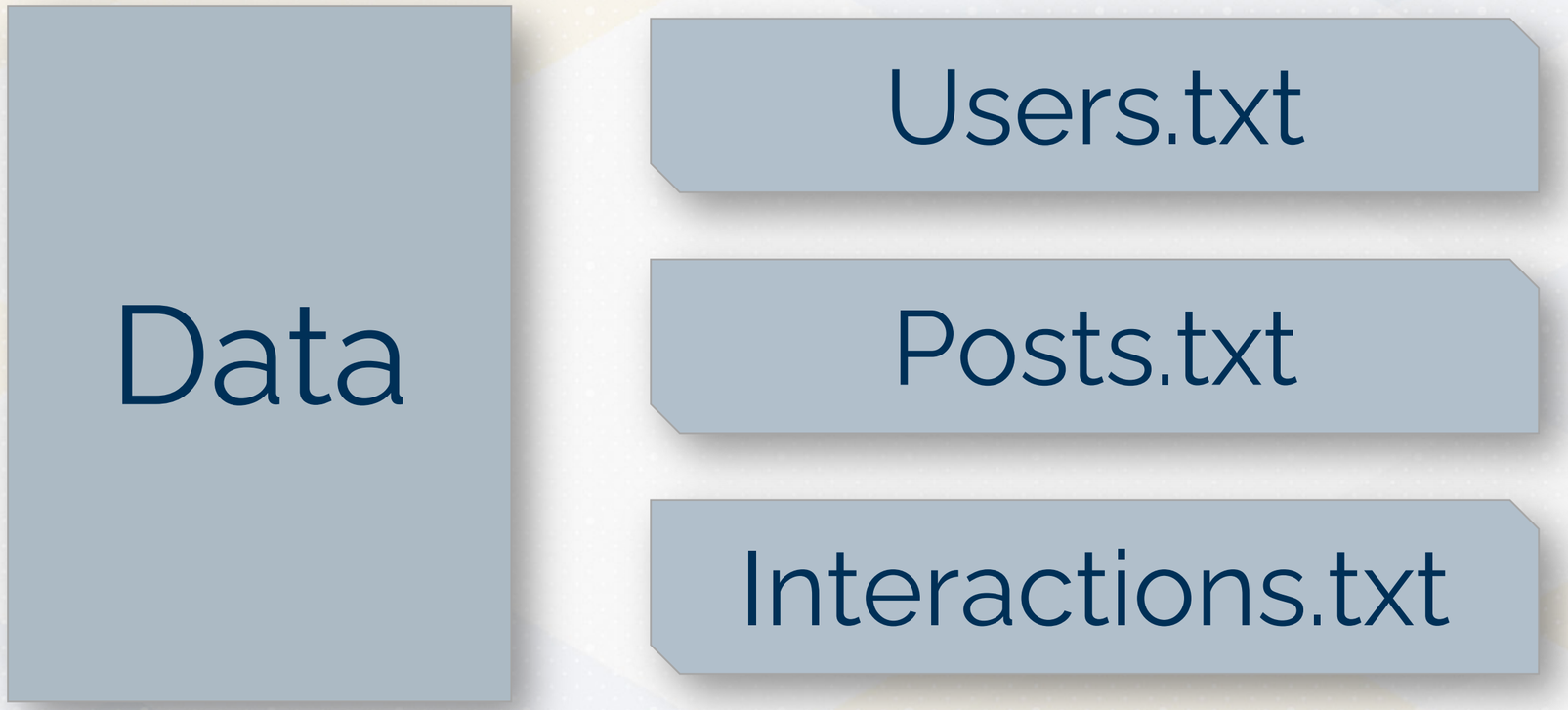
- Course not tailored for auditing or P/F mode
- Late Policy: 25% reduction in grade for every late day
- 4 penalty-free late days for the entire semester

Motivating Application

Social Media Analytics Application



Flat-File Database System



Users Text File

Users.txt

UserName,	Location
Timothée Chalamet,	Paris
Lana Condor,	Los Angeles
Liu Yifei,	Beijing
Burna Boy,	Lagos
Kriti Sanon,	Mumbai

Posts Text File

Posts.txt

PostID,	UserName,	Location
1001,	Timothée Chalamet,	Excited to start filming my new movie!
1002,	Lana Condor,	Had a great time at the beach today! 🌊☀️
1003,	Liu Yifei,	Enjoying the scenery in Beijing! 🏞️
1004,	Burna Boy,	Live performance tonight in Lagos! 🎤🎶
1005,	Kriti Sanon,	Loving the vibrant energy of Mumbai! 🌆

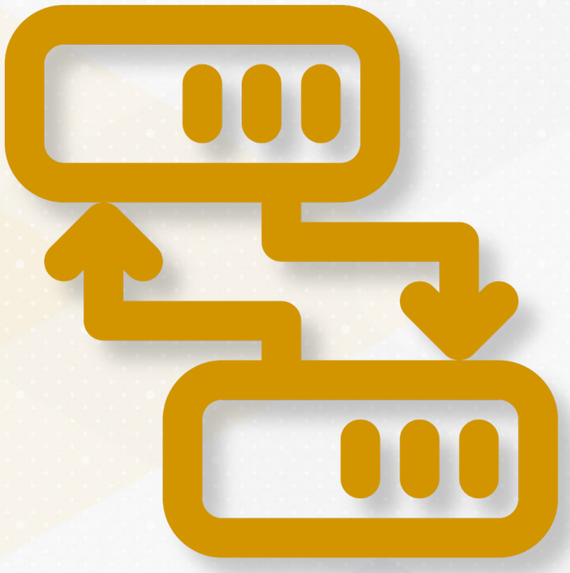
Interactions Text File

Interactions.txt

PostID,	UserName,	Reaction Type,	Comment
1001,	Lana Condor,	Comment,	Love it!
1002,	Liu Yifei,	Like,	-
1003,	Burna Boy	Like,	-
1004,	Kriti Sanon	Comment,	Wish I could be there!

Limitations of Flat-File Database

Limitation #1: Data Redundancy



PostID,	UserName,	PostContent
1001,	Timothée Chalamet,	Excited to start filming my new movie!
1006,	Timothée Chalamet,	Exploring the streets of Paris! 🇫🇷
1007,	Timothée Lamet,	Just wrapped up a day of filming 🎬
1008,	Timothée Chalamet,	Any book recommendations?

Limitation #2: Slow Operations

UserName	Location
Timothée Chalamet,	Paris
~~Lana Condor,~~	~~Los Angeles~~
Liu Yifei,	Beijing
Burna Boy,	Lagos
Kriti Sanon,	Mumbai

Limitation #3: Slow Queries

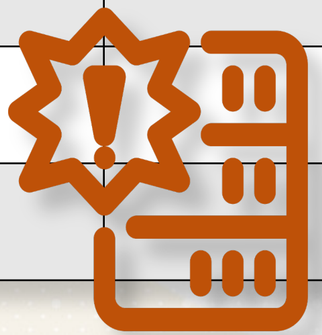
UserName,	Location
Timothée Chalamet,	Paris
Lana Condor,	Los Angeles
Liu Yifei,	Beijing
Burna Boy,	Lagos
Kriti Sanon,	Mumbai

Limitation #4: Concurrent Updates

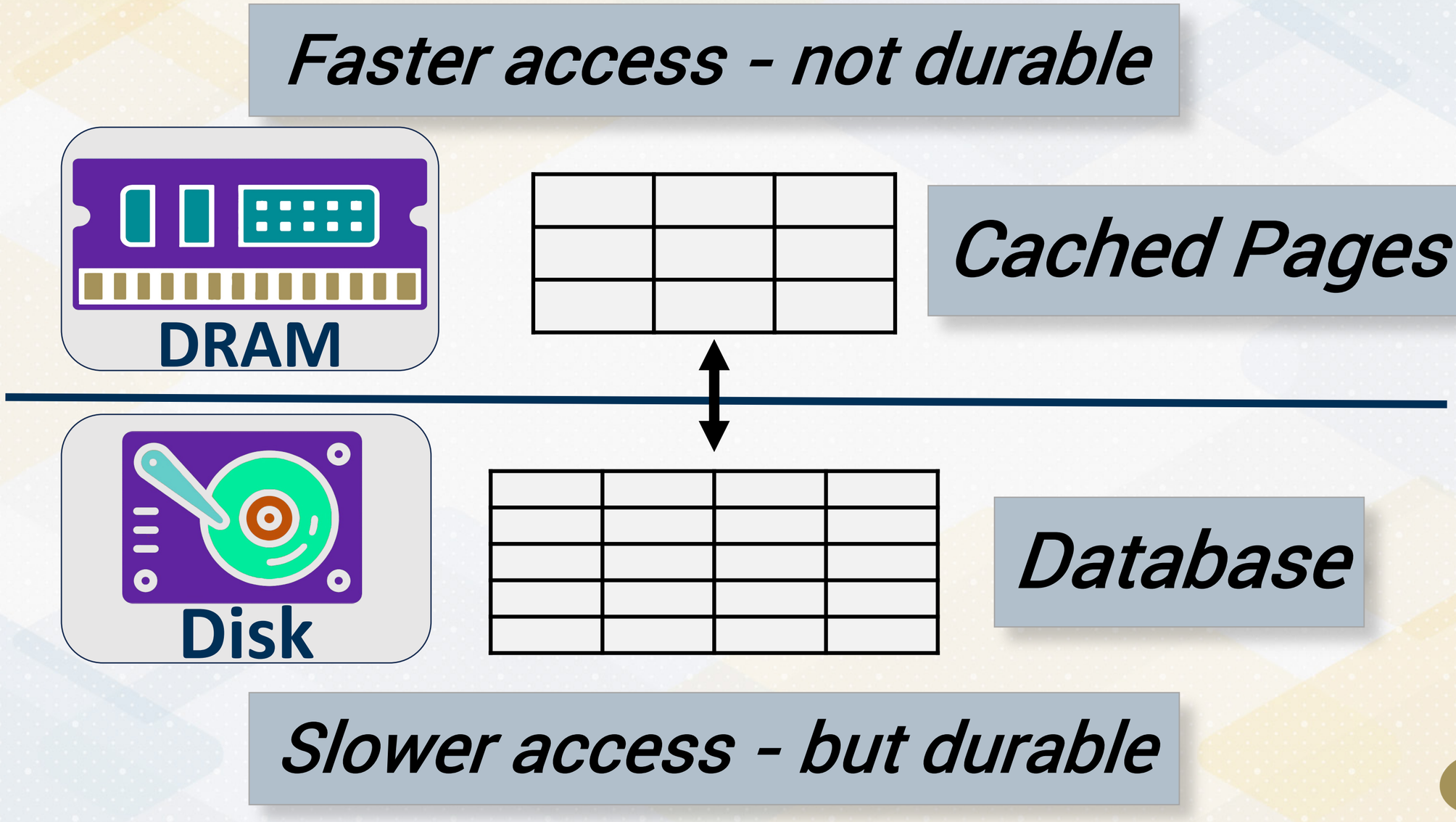
Limitation #5: Handling Disk Failure

Users.txt

UserName,	Location,	Country
Timothée Chalamet,	Paris,	France
Lana Condor,	Los Angeles,	USA
Liu Yifei,	Beijing,	China
Burna Boy,	Lagos	
Kriti Sanon,	Mumbai	



Limitation #6: Memory Management



Limitation #7: Usability

Custom Code

Comments Query Code

```
def get_comments_by_user(file_path, user_name):
    comments = []
    with open(file_path, 'r') as file:
        for line in file:
            post_id, user, reaction_type, comment_text = line.strip().split(', ')
            if user == user_name and reaction_type == "Comment":
                comments.append((post_id, comment_text))
    return comments
```

Relational Database

Relational Database

Column / Attribute

Row/ Tuple

UserName	Location
Timothée Chalamet	Paris
Liu Yifei	Beijing
Burna Boy	Lagos
Kriti Sanon	Mumbai

Relational Database

UserID	UserName	Location
1	Timothée Chalamet	Paris
2	Lana Condor	Los Angeles
3	Liu Yifei	Beijing
4	Burna Boy	Lagos
5	Kriti Sanon	Mumbai

Users

PostID	UserID	PostContent
1001	1	Excited to start filming my new movie!
1002	2	Had a great time at the beach today! 🌊☀️
1003	3	Enjoying the scenery in Beijing! 🏞️
1004	4	Live performance tonight in Lagos! 🎤🎶
1005	5	Loving the vibrant energy of Mumbai! 🌆

Posts

PostID	UserID	ReactionType	Content
1001	2	Comment	Love it!
1002	3	Like	-
1003	4	Like	-
1004	5	Comment	Wish I could be there!

Interactions

Relational Database

List of Tables

Logical

Physical

Storage Formats

Indexing Data Structures

Relational Database

Logical Database Design

Simple Query Language for Complex Data Manipulation

Physical Database Design

Optimize Indexing for Storage Hardware

Benefits of Relational Database

Benefit #1: No Data Redundancy

UserID	UserName	Location
1	Sir Timothée Chalamet	Paris

PostID	UserID	PostContent
1001	1	Excited to start filming my new movie!
1006	1	Exploring the streets of Paris!
1007	1	Just wrapped up a day of filming
1008	1	Any book recommendations?

Benefit #2: Fast Operations

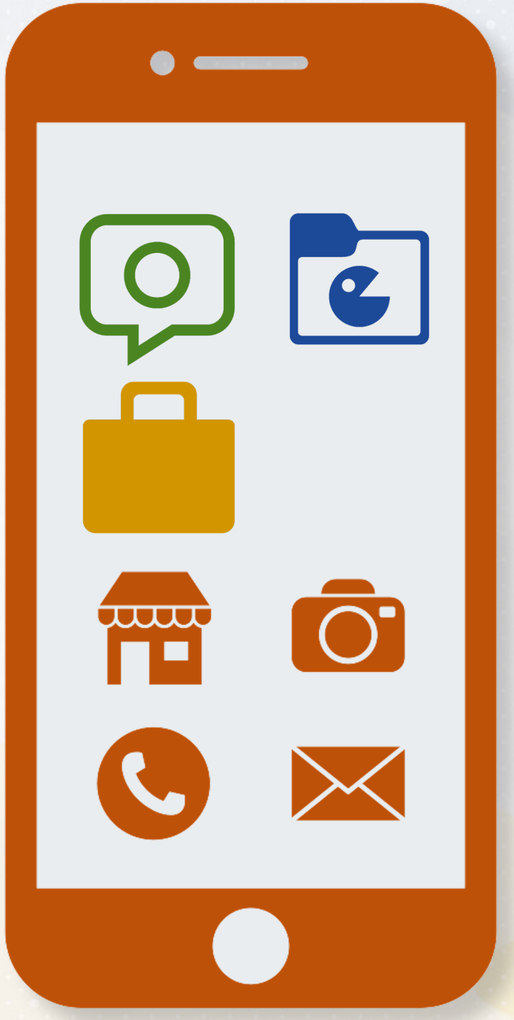
- Efficient Data Deletion
- User (Tuple) Removal
- Fast Deletion

Benefit #3: Fast Queries

Index Database

Apps in labeled folders

Location-based index

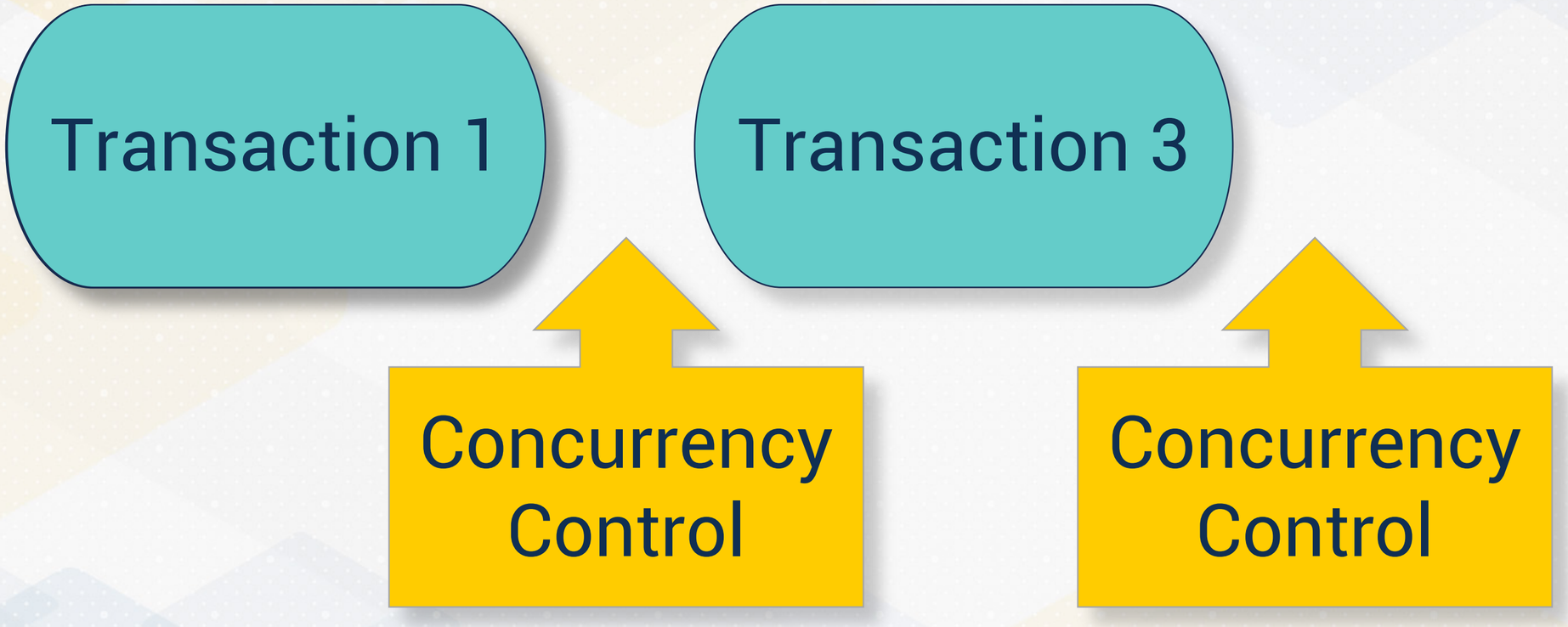


Benefit #3: Fast Queries

```
SELECT *
FROM Users
WHERE LOCATION = 'Mumbai';
```

UserName,	Location
Timothée Chalamet,	Paris
Lana Condor,	Los Angeles
Liu Yifei,	Beijing
Burna Boy,	Lagos
Kriti Sanon,	Mumbai

Benefit #4: Concurrent Updates



USER 1

Timothée Chalamet,
Paris

USER 2

Xavier Laurent,
New York

USER 2

Timothée Chalamet,
Paris

Benefit #5: Handling Failures

Atomic Transactions → "All or Nothing"

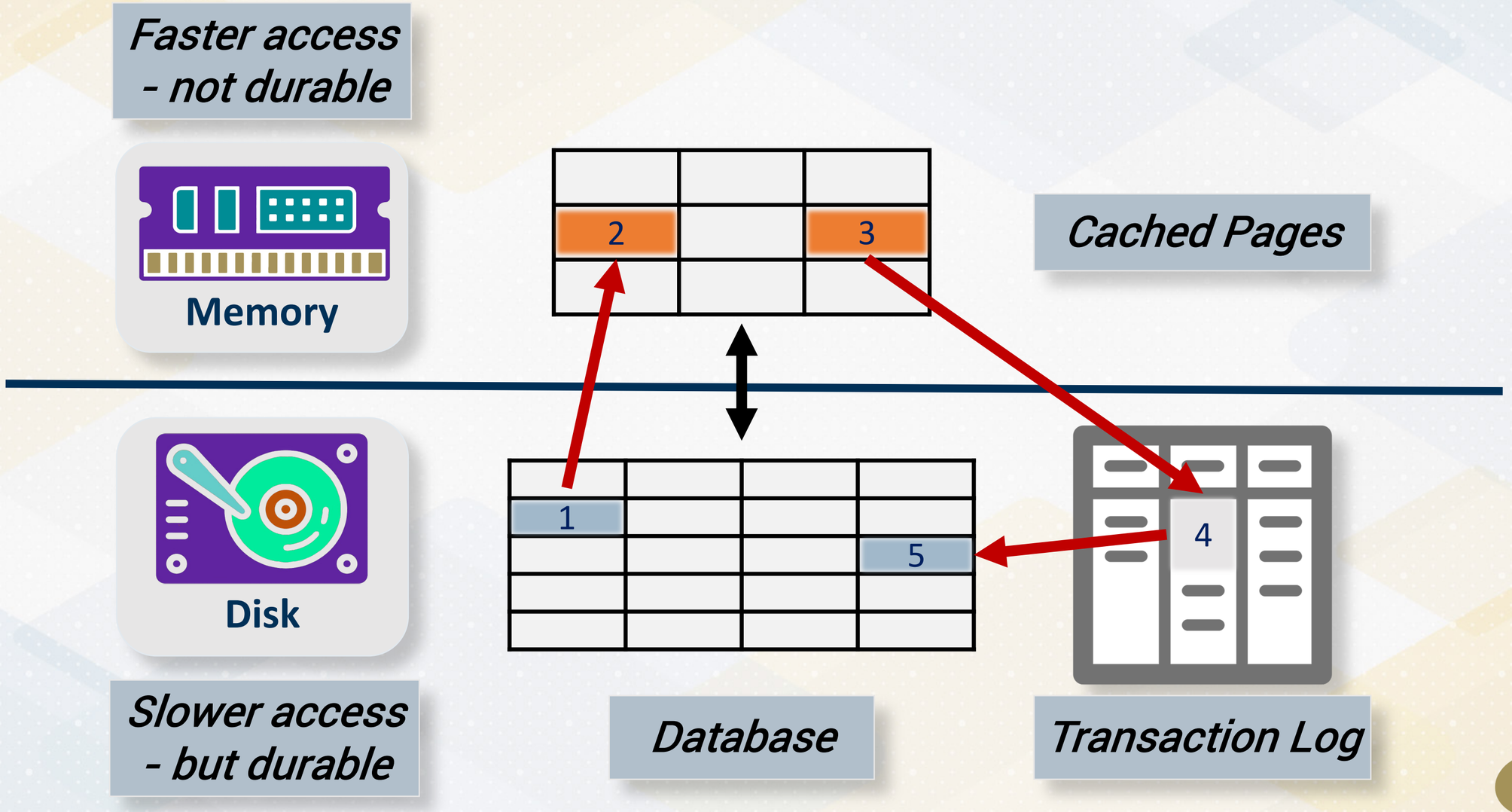
UserName	Location	Country
Timothée Chalamet	Paris	France
Lana Condor	Los Angeles	USA
Liu Lifei	Beijing	China
Burna Boy	Lagos	
Kriti Sanon	Mumbai	

Reversion

Benefit #5: Handling Failures

UserName	Location	Country
Timothée Chalamet	Paris	France
Lana Condor	Los Angeles	USA
Liu Lifei	Beijing	China
Burna Boy	Lagos	
Kriti Sanon	Mumbai	

Benefit #6: Memory Management



Benefit #7: Usability

UserName	Location
Timothée Chalamet	Paris
Lana Condor	Los Angeles
Liu Yifei	Beijing
Burna Boy	Lagos
Kriti Sanon	Mumbai

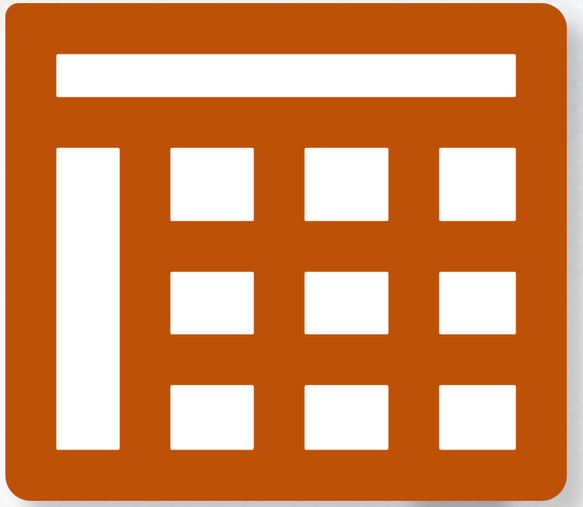
SQL = Declarative

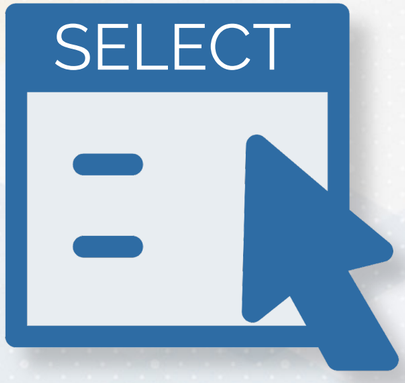
Python, C++ = Imperative

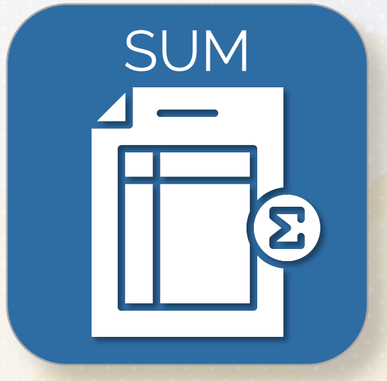
Relational Operators

Relational Operators

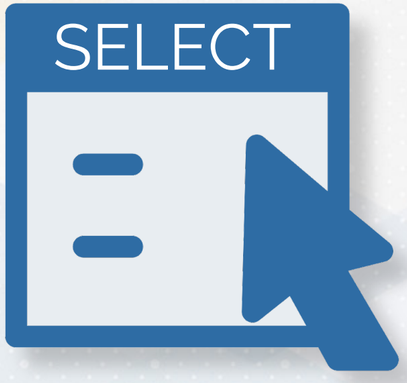
Relations

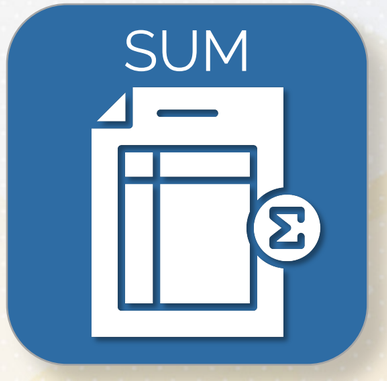




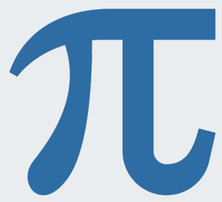


Relational Operators





SELECT (Projection Operator)



Select specific columns from a table

Example: Retrieve locations of all users.

```
SELECT Location
FROM Users;
```

WHERE (Selection Operator)

Filters rows based on specified conditions

Example: Find all interactions that are "Like" reactions.

```
SELECT *
FROM Interactions
WHERE ReactionType = 'Like';
```

GROUP BY (Grouping Operator)

γ

- Groups rows of same values
- Used with aggregate functions like SUM

Example: Count the number of reactions that each post received.

```
SELECT PostID,
       COUNT(*) AS ReactionCount
FROM Interactions
GROUP BY PostID;
```

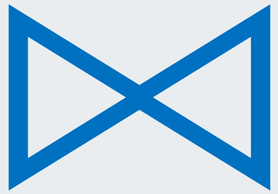
SUM (Aggregation Operator)

- Adds group values
- Defined by GROUP BY clause

Example: Total number of posts made by each user, grouping the results by UserID.

```
SELECT UserID,
    COUNT(PostID) AS TotalPosts
FROM Posts
GROUP BY UserID;
```

JOIN (Join Operator)



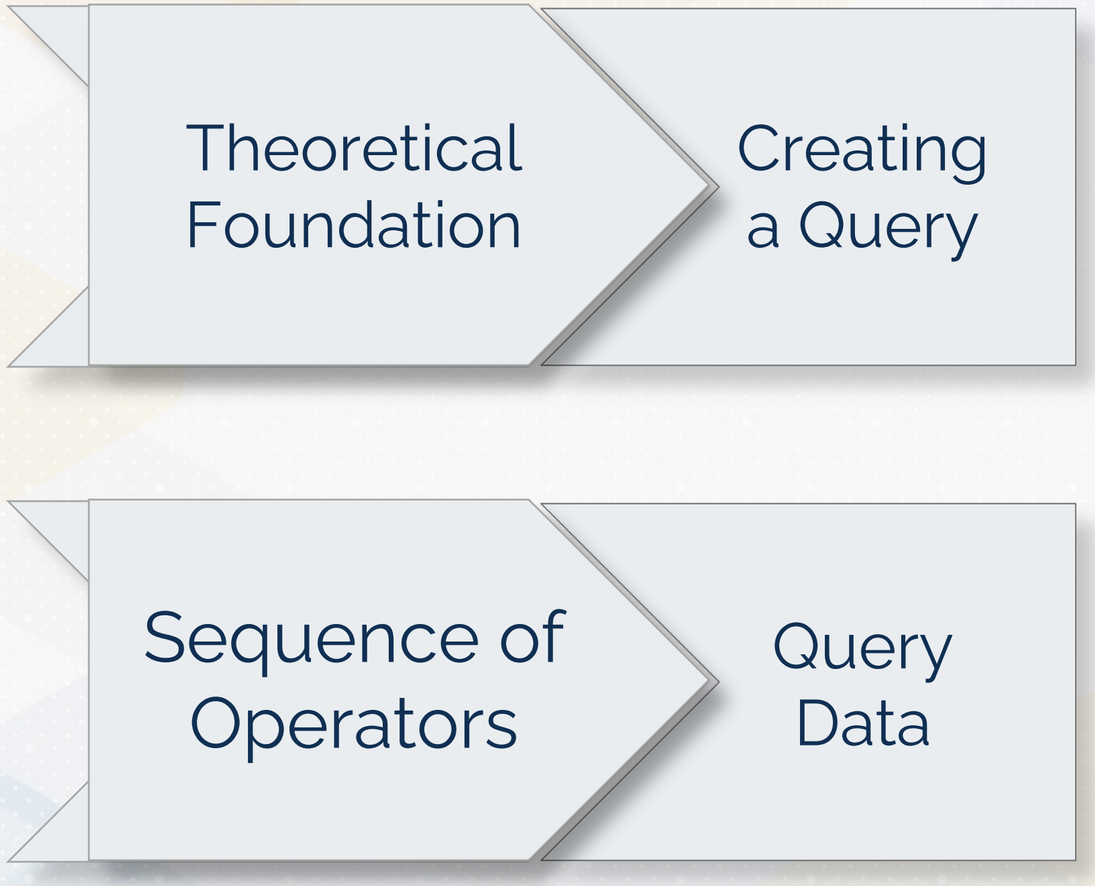
- Links rows from two different tables
- Combine information from both

Example: Total number of interactions each post receives.

```
SELECT Posts.PostID,
    COUNT(Interactions.ReactionType) AS TotalInteractions
FROM Posts
JOIN Interactions ON Posts.PostID = Interactions.PostID
GROUP BY Posts.PostID;
```

Relational Algebra

Relational Algebra



Relational Algebra

- Combine Operators
- Filter Interactions
- Combine Tables
- Group & Count Results
- Project Fields
- Sort Popular Posts

```
SELECT Interactions.PostID,
       COUNT(*) AS Likes,
       Users.UserID, Users.Username
FROM Interactions
JOIN Users ON Interactions.UserID =
Users.UserID
WHERE Interactions.ReactionType = 'Like'
GROUP BY Interactions.PostID,
         Users.UserID,
         Users.Username
ORDER BY Likes DESC;
```

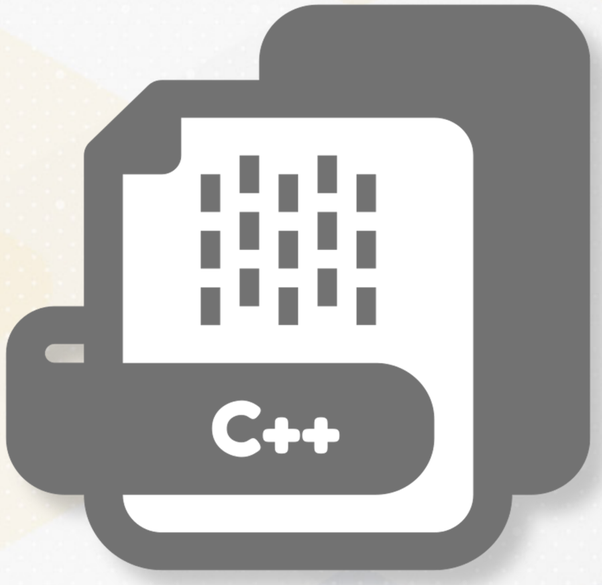
Relational Algebra

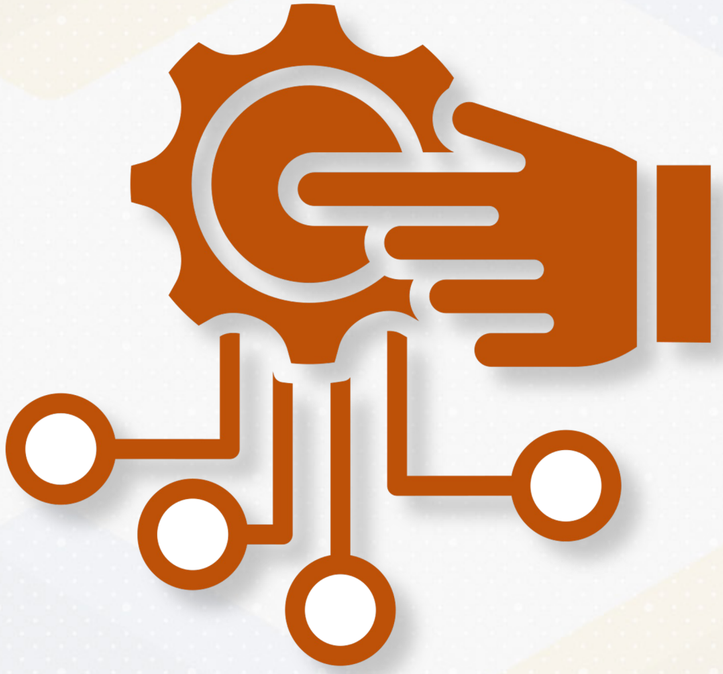
- Filters Interactions
- Combine Tables
- Group & Aggregate
- Project Output
- Order Posts

```
τ Likes DESC
 (π PostID, Likes, UserID, Username
  (γ PostID, UserID, Username;
COUNT(*)→Likes
   (σ ReactionType='Like'
    (Interactions) ⋈ Users)
   )
  )
```

BuzzDB

BuzzDB





Insertion Operator in C++

UserName,	Location
Timothée Chalamet,	Paris

```
void BuzzDB::insert(int key, int value) {
    Tuple newTuple = {key, value};
    table.push_back(newTuple);   // Add to main table vector
    index[key].push_back(value); // Also, update the index map
 }
```

Key-Based Tuple Retrieval

Populating the Database

BuzzDB Insert Method Creation

```
int main() {
    BuzzDB db;
    // Populating the database
    db.insert(1, 100); db.insert(1, 200);
    db.insert(2, 50);
    db.insert(3, 200); db.insert(3, 200); db.insert(3, 100);
    db.insert(4, 500);
    // Executing aggregation query
    db.selectGroupBySum();
    return 0;
 }
```

Aggregation Query

selectGroupBySum method

Tally Summarization

Iterate Over Keys

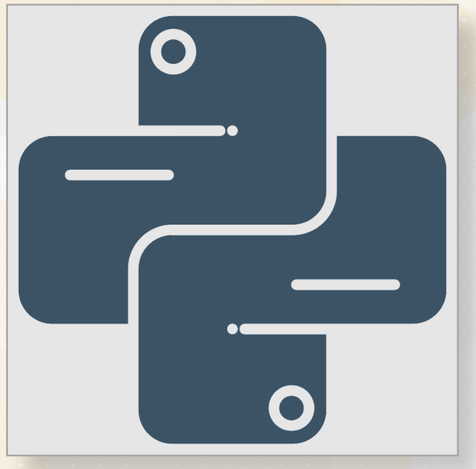
Iterate Over Values

```
void BuzzDB::selectGroupBySum() {
  // Iterate over each key
  for (auto const &pair : index) {
    int sum = 0;
    // Sum values for this key
    for (auto const &value : pair.second) {
      sum += value;
    }
    std::cout << "key: " << pair.first << ", sum: " << sum
<< '\n';
  }
}
```

Aggregation Query


```
int main() {
    ...
    // Executing aggregation query
    db.selectGroupBySum();
    return 0;
}
// PROGRAM OUTPUT
key: 1, sum: 300
key: 2, sum: 50
key: 3, sum: 500
key: 4, sum: 500
```

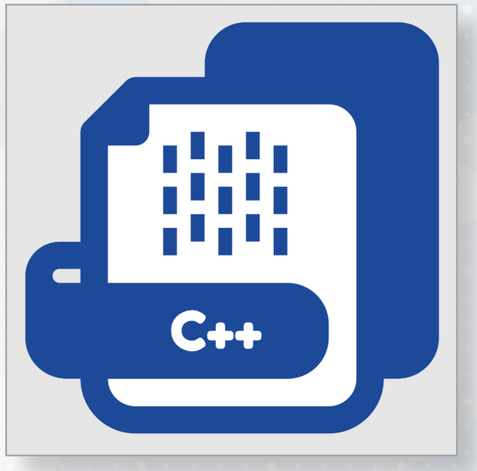
C++ vs Python



Superior Performance

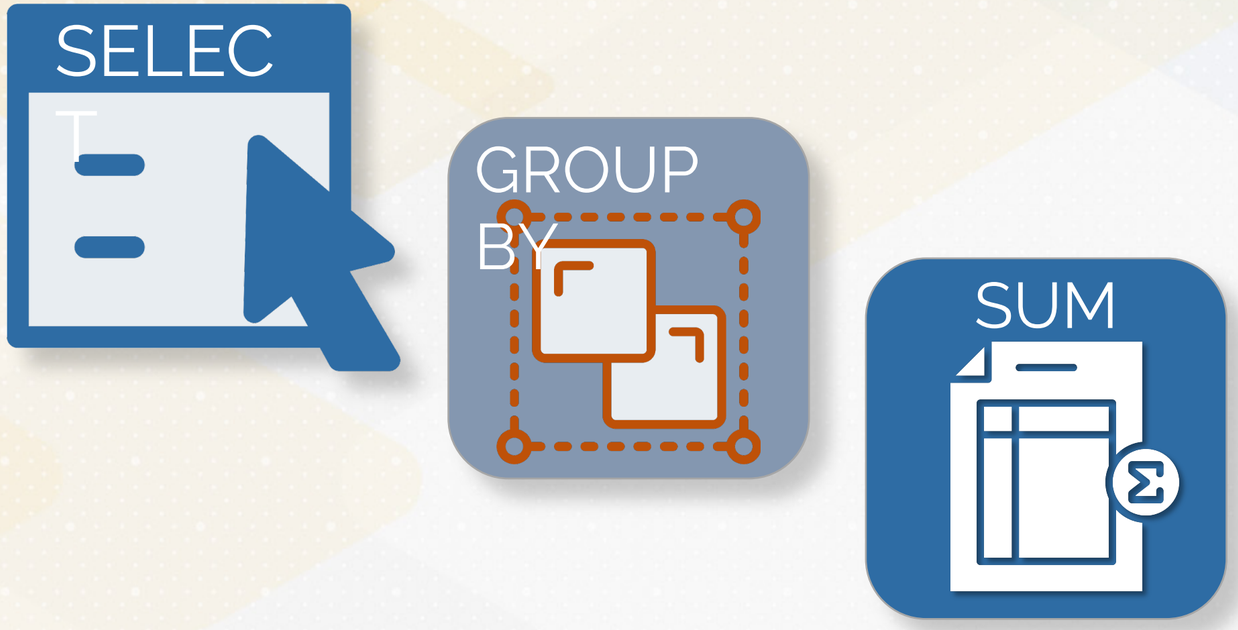
Executable Code

Fast & Efficient



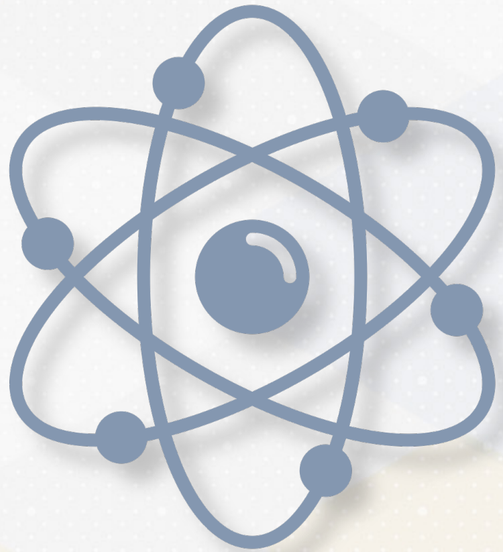
Transactions

Queries vs Transactions



✓ Multiple Database Changes

Transaction: Atomicity Property



Transaction: Atomicity Property

Transaction: Consistency Property

$100 from Account 1 to Account 2

Update Failure → No Partial Updates

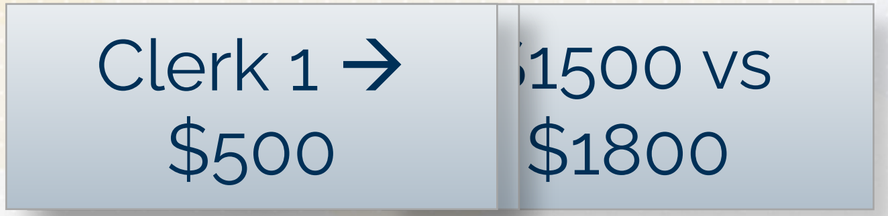
```
BEGIN TRANSACTION;

-- Withdraw $100 from account 1
UPDATE ACCOUNT SET Balance = Balance - 100 WHERE AccountID = 1;

-- Deposit $100 into account 2
UPDATE ACCOUNT SET Balance = Balance + 100 WHERE AccountID = 2;

COMMIT;
```

Transaction: Isolation Property

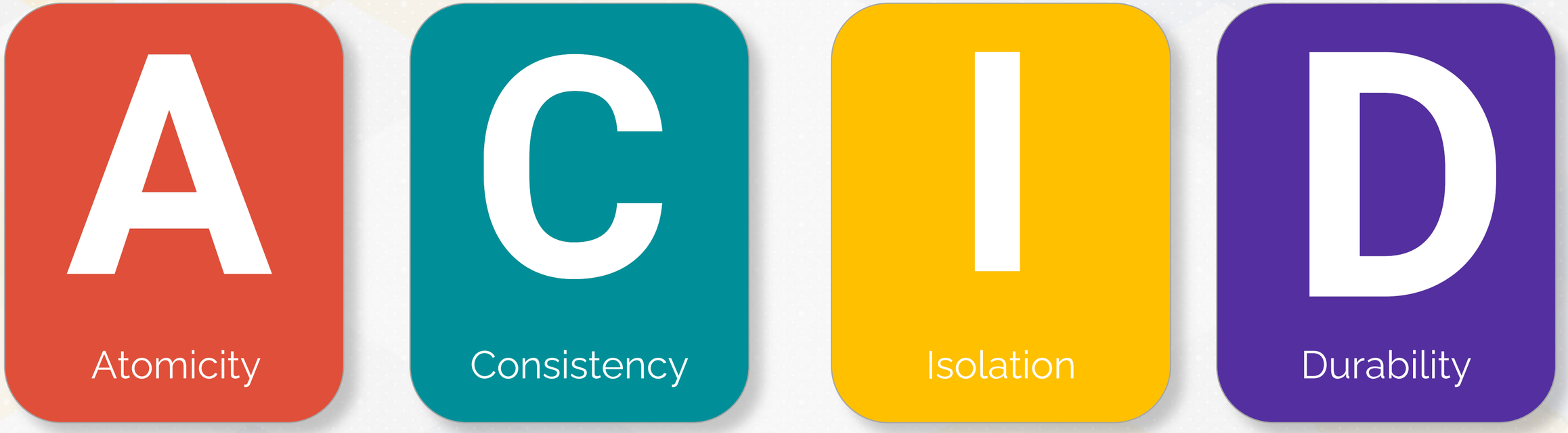



```
-- Clerk 1: Deposits $500 into account 1
BEGIN TRANSACTION;
SELECT Balance FROM ACCOUNT WHERE AccountID = 1; -- Suppose it returns $1000
UPDATE ACCOUNT SET Balance = 1000 + 500 WHERE AccountID = 1;
COMMIT;
-- Clerk 2: Deposits $300 into account 1 almost at the same time
BEGIN TRANSACTION;
SELECT Balance FROM ACCOUNT WHERE AccountID = 1; -- Suppose it still returns $1000
UPDATE ACCOUNT SET Balance = 1000 + 300 WHERE AccountID = 1;
COMMIT;
```

Transaction: Durability Property

```sql
BEGIN TRANSACTION;
UPDATE ACCOUNT SET Balance = Balance - 500 WHERE AccountID = 1; -- Customer withdraws $500
COMMIT;
```

ACID Properties



History of “ACID”

Andreas Reuter (1983)

Reliable Management

Multi-User Capability

Photo: Gülay Keskin/Heidelberg Institute for Theoretical Studies (HITS)

History of “ACID”

Andreas Reuter (1983)

Reliable Management

Multi-User Capability

Photo: Gülay Keskin/Heidelberg Institute for Theoretical Studies (HITS)

Conclusion

- Illustrative Social Media Analytics
- Limitations of a Flat-file Database System
- Benefits of a Relational Database System
- Relational Algebra
- ACID Properties