Welcome to

CS 460: Introduction to Database Systems

https://bu-disc.github.io/CS460/

Instructor: *Manos Athanassoulis*

email: mathan@bu.edu

Course Summary

We will learn how to:

- **Model data** and design good databases

  ER Model, Relational

- **Query data** with SQL
- **Store & manage** data

  Bits to Files to Disks, Storage Layouts, Indexes, Sorting

- **Reason** about query performance

  Query evaluation & optimization

- **Update** data

  Transactions, logging, ACID properties

Today

big data

data-driven world

databases & database systems

when you see this, I want you to speak up!
[and you can always interrupt me]

Big Data

buzzword …

but …

science / government / business / personal data

exponentially growing data collections

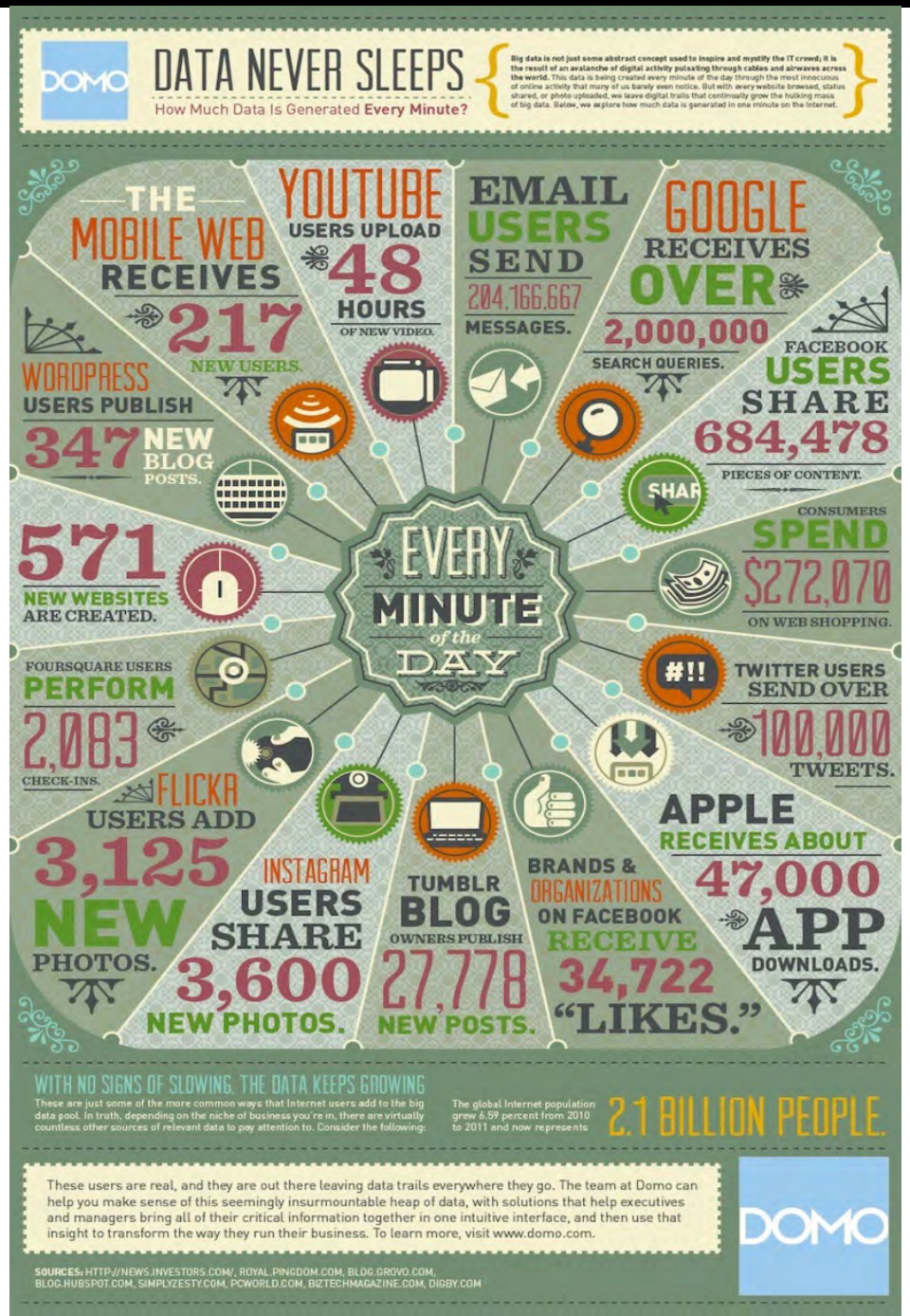
So, it is all good!

How big is “Big”?

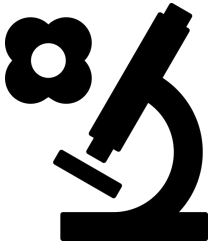
Every day, we create 2.5 exabytes* of data — 90% of the data in the world today has been created in the last two years alone.

[Understanding Big Data, IBM]

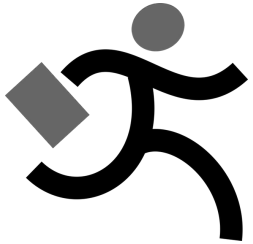
*exabyte = 10^9 GB



Using Big Data



experimental physics (IceCube, CERN)
biology
neuroscience



data mining business datasets
machine learning for corporate and consumer

data analysis for fighting crime

… are only some examples

Data-Driven World

Big Data V's

Volume

Velocity

Variety

Veracity

Information is transforming traditional business.

["Data, data everywhere", Economist]

CS460

we live in a ***data-driven*** world

CS460 is about the ***basics*** for
storing, ***using***, and ***managing*** data

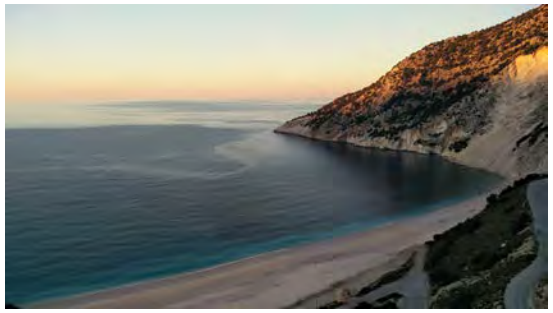
your lecturer (that's me!)

Manos Athanassoulis

name in greek: Μάνος Αθανασούλης

photo for VISA / conferences

grew up in Greece
enjoys playing basketball and the sea



Myrtos, Kefalonia, Greece

BSc and MSc @ University of Athens, Greece
PhD @ EPFL, Switzerland
Research Intern @ IBM Research Watson, NY
Postdoc @ Harvard University

some awards:
Facebook Faculty Research Award
NSF CRII Research Award
Best of SIGMOD 2017, VLDB 2017

http://cs-people.bu.edu/mathan/
Office: MCS 106
Office Hours: after class 3:30pm

your awesome TAs

Aneesh Raman
PhD student in DB
aneeshr@bu.edu

Tarikul Islam Papon
PhD student in DB
papon@bu.edu

Subhadeep Sarkar
Postdoc in DB
ssarkar1@bu.edu

Data

to make data usable and manageable

we organize them in collections

Databases

a large, integrated, *structured* collection of data

intended to model some <u>real-world</u> enterprise

<u>Examples</u>: a university, a company, social media

<u>Social media:</u> users, posts, photos

what is missing?

-- how to connect these?

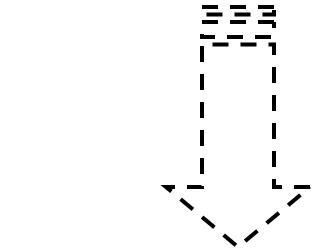
-- shares, likes, friend-relationship

Database Systems

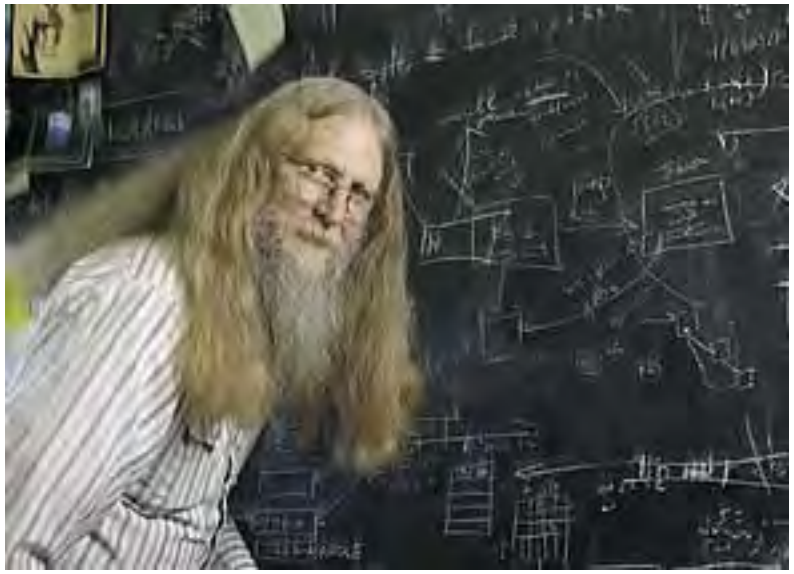
a.k.a. database management systems (DBMS)
a.k.a. data systems

Sophisticated
pieces of software…

… which store, manage,
organize, and facilitate
access to my databases …



... so I can do things (and ask questions) that are
otherwise hard or impossible



“relational databases are the foundation of western civilization”

Bruce Lindsay, IBM Research

ACM SIGMOD Edgar F. Codd Innovations award 2012

Ok but what really IS a database system?

Is the Internet a DBMS?

Is a File System a DBMS?

Are Social Media a DBMS?

Is the Internet a DBMS?

Not really!

Fairly sophisticated search available

web crawler *indexes* pages for fast search

.. but

data is unstructured and untyped

not well-defined "correct answer"

cannot update the data

freshness? consistency? fault tolerance?

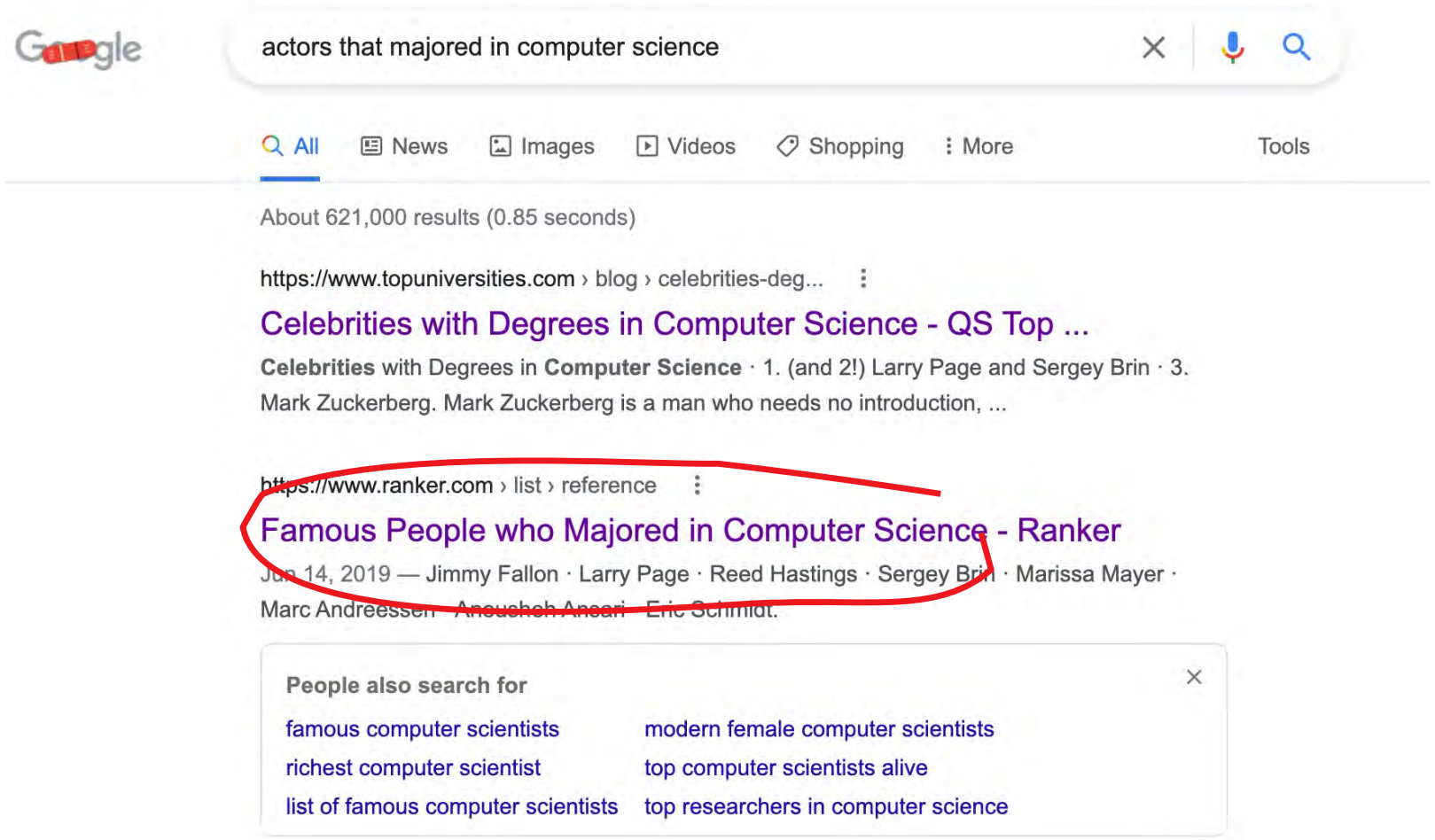
web sites **use** a *DBMS* to provide these functions

e.g., amazon.com (Oracle), facebook.com (MySQL and others)

"Search" vs. Query

What if you wanted to find out actors that have studied computer science?

Try "actors that majored in computer science" in your favorite search engine.



“Search” vs. Query

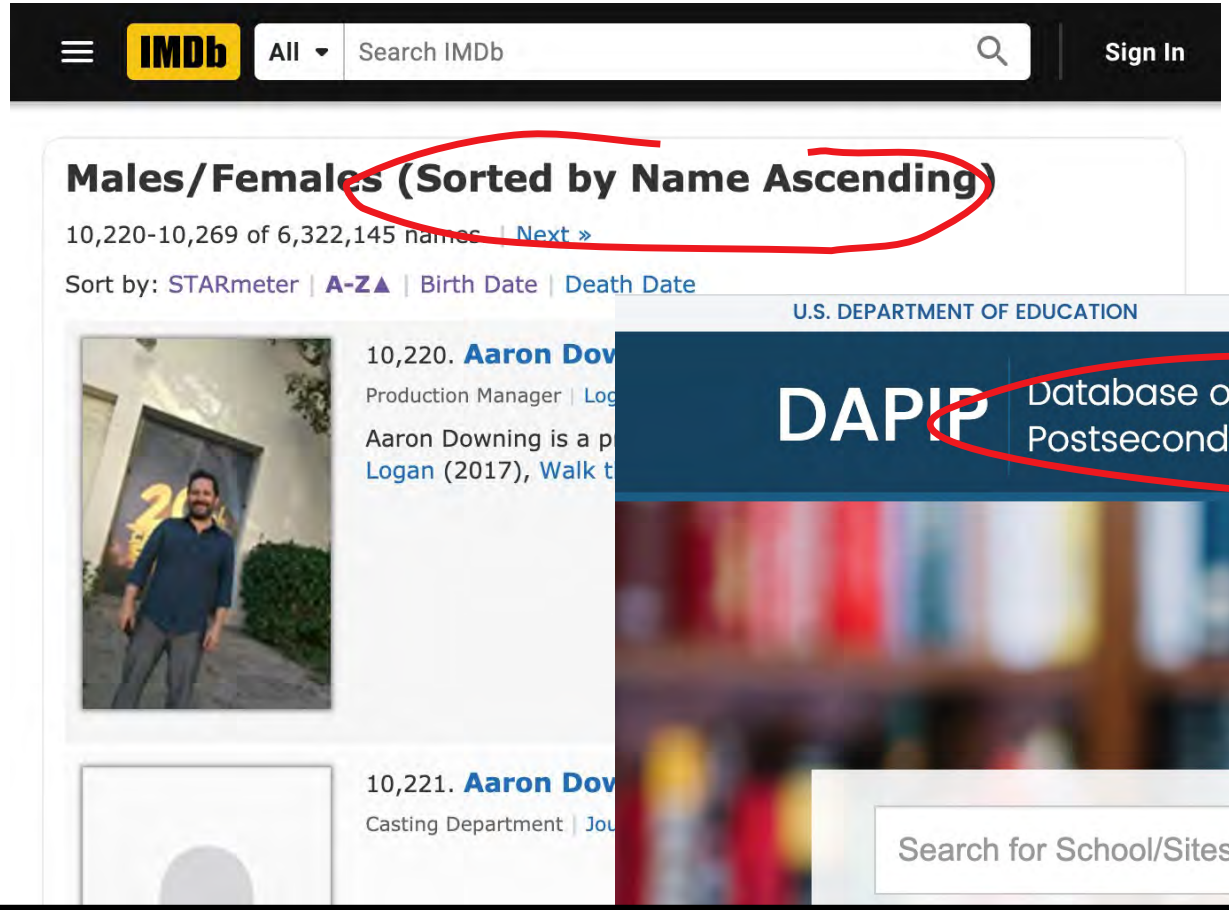
“Search” can return only what’s been “stored”

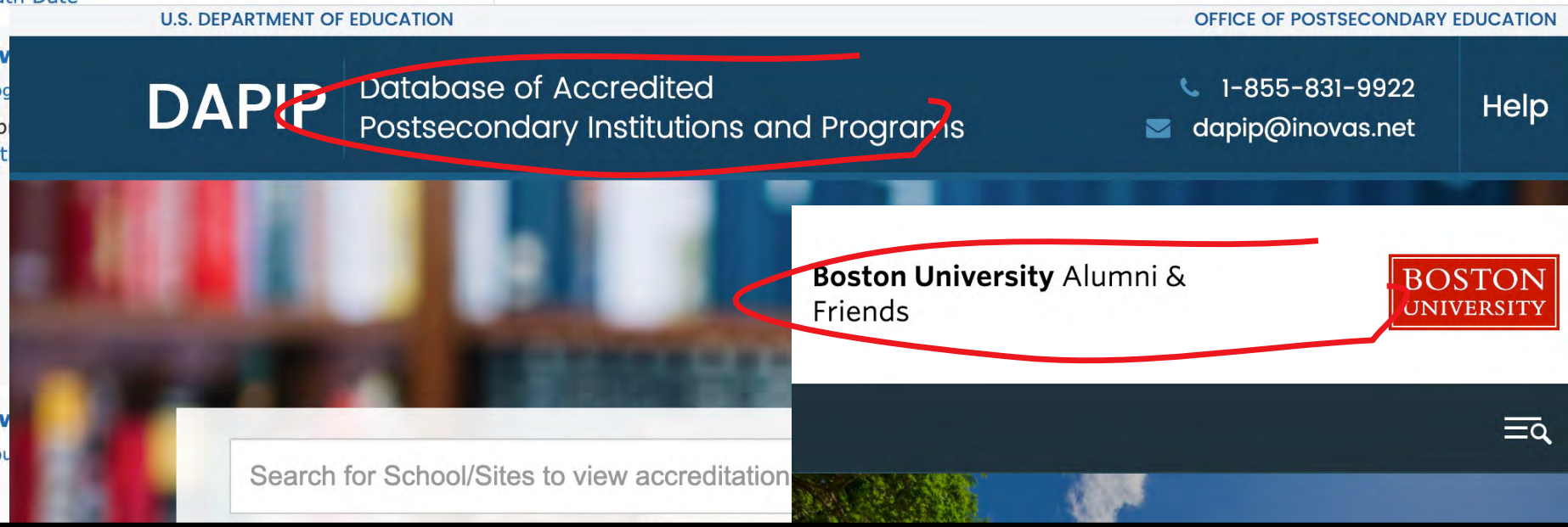
e.g., best match at Google:

A “Database Query” Approach

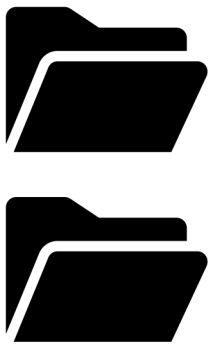
where can we find accurate data for actors, universities, and majors?

A "Database Query" Approach





"Actors" JOIN "Accredited Universities" JOIN "Alumni"
WHERE Actors.Name=Alumni.Name AND Alumni.Major=CS

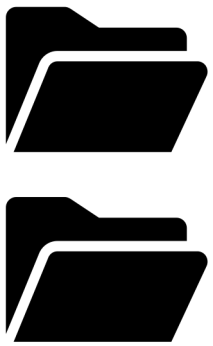


Is a File System a DBMS?

Thought Experiment 1:

- You and your project partner are editing the same file.
- You both save it at the same time.
- Whose changes survive?

A) Yours **B) Partner's** **C) Both** **D) Neither** **E) ???**



Is a File System a DBMS?

Thought Experiment 1:

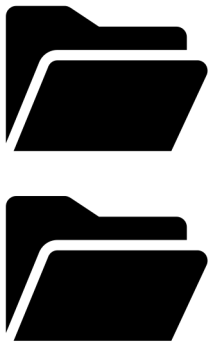
- You and your project partner are editing the same file.
- You both save it at the same time.
- Whose changes survive?

A) Yours B) Partner's C) Both D) Neither E) ???

Thought Experiment 2:

- You're updating a file.
- The power goes out.
- Which of your changes survive?

A) All B) None C) All Since last save D) ???



Is a File System a DBMS?

Not really!

Thought Experiment 1:

- You and your project partner are editing the same file.
- You both save it at the same time.
- Whose changes survive?

A) Yours **B) Partner's** **C) Both** **D) Neither** **E) ???**

Thought Experiment 2:

- You're updating a file.
- The power goes out.
- Which of your changes survive?

A) All **B) None** **C) All Since last save** **D) ???**

Are Social Media a DBMS?

Not really!

Is the data structured & typed?

Does it offer well-defined queries?

Does it offer properties like “durability” and “consistency”?

Facebook is a data-driven company that uses several database systems (>10) for different use-cases (internal or external).

Why take this class?

computation to *information*

corporate, personal (web), science (big data)

database systems *everywhere*

data-driven world, data companies

DBMS: much of CS as a practical discipline

languages, theory, OS, logic, architecture, HW

CS460 in a nutshell

model
data representation model

query
query languages – ad hoc queries

access (concurrently multiple reads/writes)
ensure *transactional* semantics

store (reliably)
maintain *consistency/semantics* in *failures*

A “free taste” of the class

data modeling

query languages

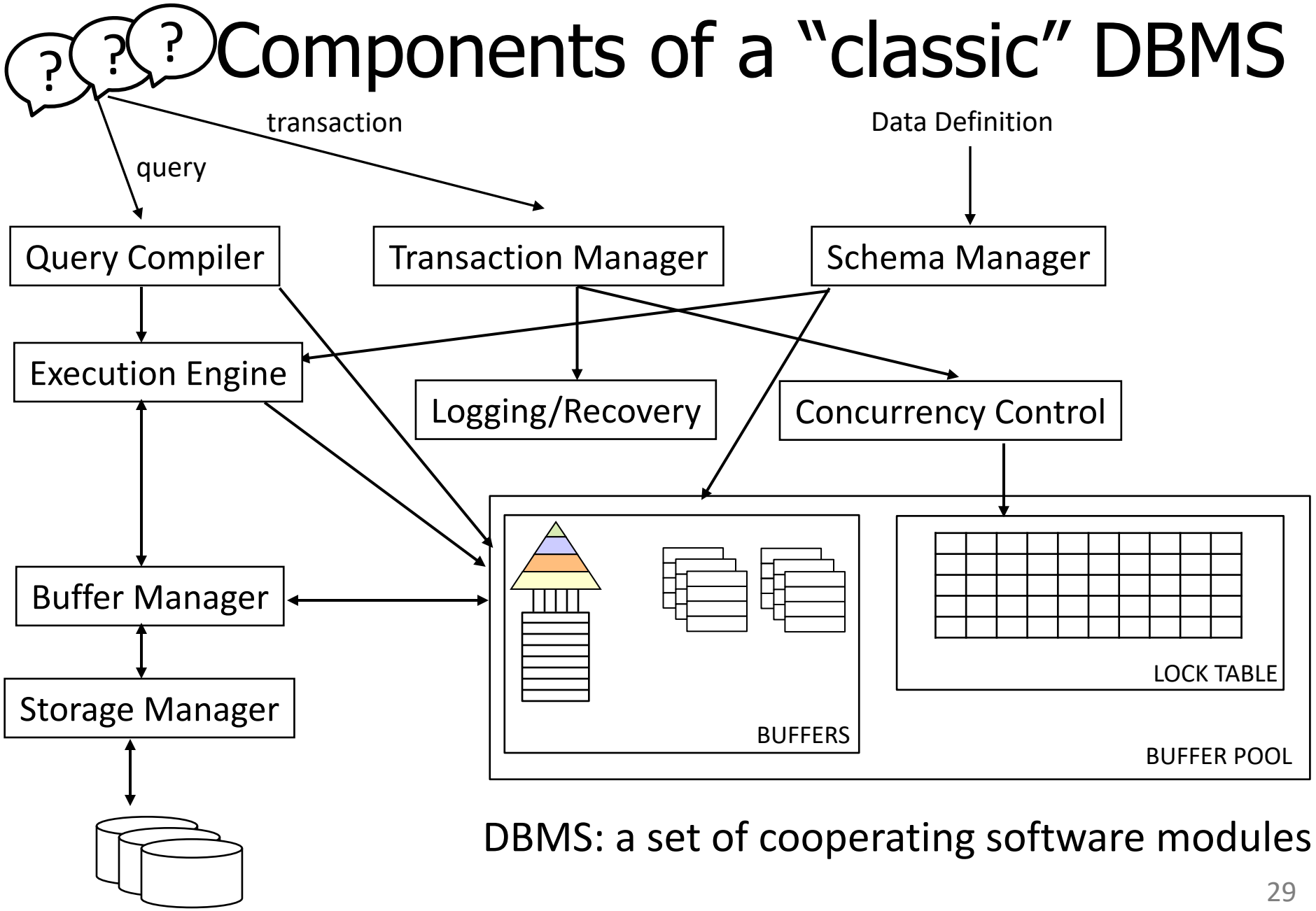
concurrent, fault-tolerant data management

DBMS architecture

Coming in next class

Discussion on *database systems* *designs*

Components of a "classic" DBMS



DBMS: a set of cooperating software modules

Describing Data: Data Models

data model : a collection of concepts describing data

relational model is the most widely used model today

key concepts

relation : basically a table with rows and columns

schema : describes the columns (or fields) of each table

Schema of "University" Database

Students

***sid**: string, **name**: string, **login**: string, **age**: integer, **gpa**: real*

Courses

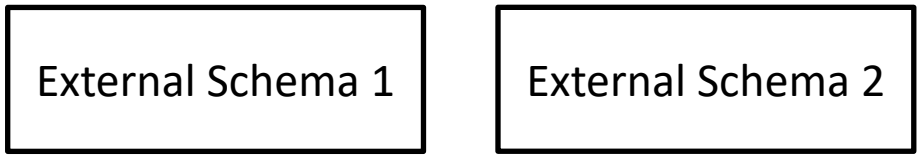
***cid**: string, **cname**: string, **credits**: integer*

Enrolled

***sid**: string, **cid**: string, **grade**: string*

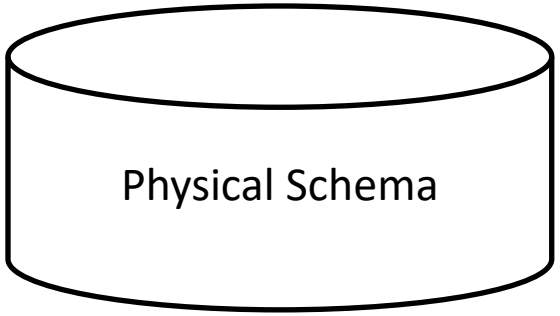
Levels of Abstraction

what the users *see*



what is the *data model*

how the data is *physically* stored
e.g., files, indexes



Schemas of "University" Database

Conceptual Schema

Students

***sid**: string, **name**: string, **login**: string, **age**: integer, **gpa**: real*

Courses

***cid**: string, **cname**: string, **credits**: integer*

Enrolled

***sid**: string, **cid**: string, **grade**: string*

Physical Schema

relations stored in heap files
indexes for sid/cid

External Schema

a "view" of data that can be derived from the existing data

*Course_Info (**cid**: string, **enrollment**:integer)*

*which **combines** information from Courses & Enrolled*

Data Independence

Abstraction offers “application independence”

Logical data independence

Protection from changes in *logical* structure of data

Physical data independence

Protection from changes in *physical* structure of data

Q: Why is this particularly important for DBMS?

Applications can treat DBMS as black boxes!

Queries

”Bring me all students with gpa more than 3.0”

“SELECT * FROM Students WHERE gpa>3.0”

SQL – a powerful *declarative* query language

treats DBMS as a black box

What if we have multiples accesses?

Concurrency Control

multiple users/apps

Challenges

how frequent access to slow medium

how to keep CPU busy

how to avoid *short jobs* waiting behind *long ones*

e.g., ATM withdrawal while summing all *balances*

interleaving actions of *different* programs

Concurrency Control

Problems with *interleaving* actions of diff. programs

Bill

Move 100 from savings to checking

Alice

Bad interleaving:

Savings –= 100

Print balances

Checking += 100

Printout is missing 100$!

Concurrency Control

Problems with *interleaving* actions of diff. programs

Bill

Move 100 from savings to checking

Alice

What is a correct interleaving?

- Savings –= 100
- Checking += 100
- Print balances

How to achieve this interleaving?

Scheduling Transactions

Transactions: atomic sequences of **R**eads & **W**rites

$T_{Bill}=\{R1_{Savings}, R1_{Checking}, W1_{Savings}, W1_{Checking}\}$
$T_{Alice}=\{R2_{Savings}, R2_{Checking}\}$

How to avoid previous problems?

Scheduling Transactions

All interleaved executions equivalent to a *serial*

All actions of a transaction executed *as a whole*

Time

$R1_{Savings}$, $R1_{Checking}$, $W1_{Savings}$, $W1_{Checking}$, $R2_{Savings}$, $R2_{Checking}$

$R2_{Savings}$, $R2_{Checking}$, $R1_{Savings}$, $R1_{Checking}$, $W1_{Savings}$, $W1_{Checking}$

~~$R1_{Savings}$, $R1_{Checking}$, $W1_{Savings}$, $R2_{Savings}$, $R2_{Checking}$, $W1_{Checking}$~~

$R1_{Savings}$, $R1_{Checking}$, $R2_{Savings}$, $R2_{Checking}$, $W1_{Savings}$, $W1_{Checking}$

How to achieve one of these?

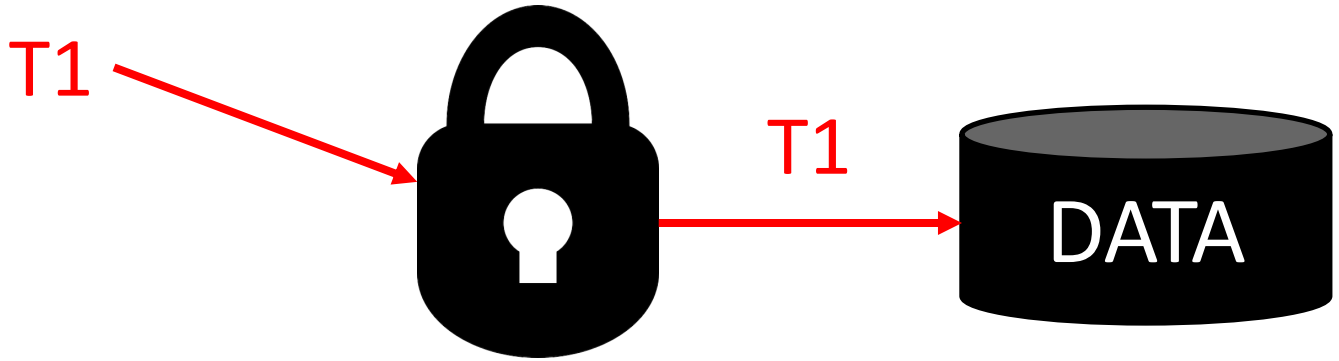
Locking

before an object is accessed a lock is requested

Locking

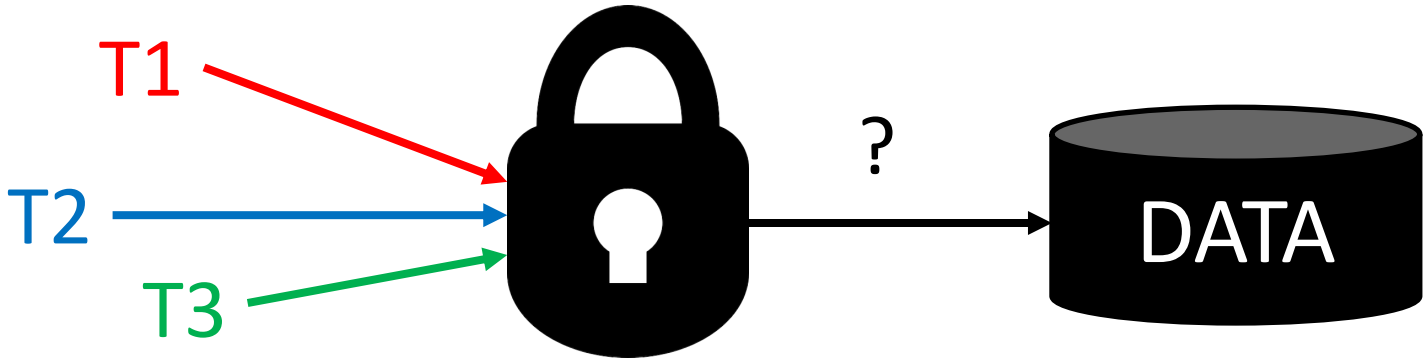
before an object is accessed a lock is requested

Locking



before an object is accessed a lock is requested

Locking



locks are held until the end of the transaction

[this is only one way to do this, called "strict two-phase locking"]

Locking

$T_1=\{R1_{Savings}, R1_{Checking}, W1_{Savings}, W1_{Checking}\}$
$T_2=\{R2_{Savings}, R2_{Checking}\}$

Both should lock *Savings* and *Checking*

What happens:
if T1 locks Savings & Checking ?
T2 has to wait
if T1 locks Savings & T2 locks Checking ?
we have a deadlock

How to solve deadlocks?

we need a mechanism to *undo*

also when a transaction is *incomplete*

e.g., due to a crash

what can be an undo mechanism?

log every action before it is applied!

Transactional Semantics

Transaction: one execution of a user program

multiple executions → multiple transactions

Every transaction:

Logging → ***A**tomic*

***C**onsistent*

***I**solated*

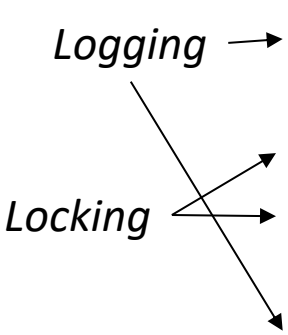
***D**urable*

Transactional Semantics

Transaction: one execution of a user program

multiple executions → multiple transactions

Every transaction:



***A**tomic* *"executed entirely or not at all"*

***C**onsistent* *"leaves DB in a consistent state"*

***I**solated* *"as if it is executed alone"*

***D**urable* *"once completed is never lost"*

Who else needs transactions?

lots of data

lots of users

frequent updates

background game analytics

Scaling games to epic proportions,

by W. White, A. Demers, C. Koch, J. Gehrke and R. Rajagopalan

ACM SIGMOD International Conference on Management of Data, 2007

Only “classic” DBMS?

No, there is much more!

NoSQL & Key-Value Stores: No transactions, focus on queries

Graph Stores

Querying raw data without loading/integrating costs

Database queries in large datacenters

New hardware and storage devices

… many exciting open problems!

Course Summary

We will learn how to:

- **Model data** and design good databases

  ER Model, Relational

- **Query data** with SQL
- **Store & manage** data

  Bits to Files to Disks, Storage Layouts, Indexes, Sorting

- **Reason** about query performance

  Query evaluation & optimization

- **Update** data

  Transactions, logging, ACID properties

https://bu-disc.github.io/CS460/

Next time in …

CS 460: Introduction to Database Systems

Database Systems Architectures

Class administrativia

Class project administrativia

https://bu-disc.github.io/CS460/

Additional Accommodations

If you require additional accommodations please contact the Disability & Access Services office at aslods@bu.edu or 617-353-3658 to make an appointment with a DAS representative to determine which are the appropriate accommodations for your case.

Please be aware that accommodations cannot be enacted retroactively, making timeliness a critical aspect for their provision.

You can optionally choose to disclose this information to the instructor.