Georgia Tech

DATABASE SYSTEM IMPLEMENTATION

GT 4420/6422 // SPRING 2019 // @JOY_ARULRAJ

LECTURE #1: COURSE INTRODUCTION & HISTORY OF DATABASE SYSTEMS

CREATING THE NEXT®

WHY YOU SHOULD TAKE THIS COURSE

DBMS developers are in demand and there are many challenging unsolved problems in data management and processing.

If you are good enough to write code for a DBMS, then you can write code on almost anything else.

TODAY'S AGENDA

Course Outline & Logistics

History of Database Systems

COURSE OBJECTIVES

Learn about modern practices in database internals and systems programming.

Students will become proficient in:

→ Writing correct + performant code
→ Proper documentation + testing
→ Code reviews
→ Working on a large systems programming project

COURSE TOPICS

The internals of single node systems for in-memory databases. We will ignore distributed deployment problems.

We will cover state-of-the-art topics.
This is **not** a course on classical DBMSs.

COURSE TOPICS

Storage Models, Compression
Logging & Recovery Methods
Indexing
Networking Protocols
Concurrency Control
Query Optimization, Execution, Compilation
Parallel Join Algorithms
New Hardware (NVM, FPGA, GPU)

BACKGROUND

I assume that you have already taken an intro course on databases (e.g., GT 4400).

We will discuss modern variations of classical algorithms that are designed for today's hardware.

Things that we will **not** cover:
SQL, Serializability Theory, Relational Algebra, Basic Algorithms + Data Structures.

BACKGROUND

All programming assignments will be written in C++11. Be prepared to debug, profile, and test a multi-threaded program.

Homework 1 will help get you caught up with C++. If you haven't encountered C++ before and are a Java programmer, you will need to pick C++ yourself.

COURSE LOGISTICS

Course Policies + Schedule:

→ Refer to https://www.cc.gatech.edu/~jarulraj/courses/4420-s19/ web page.

Academic Honesty:

→ Refer to Georgia Tech Academic Honor Code.
→ If you're not sure, ask me.
→ I'm serious. Don't plagiarize.
→ Don't forget that the person you would be cheating the most is yourself.

OFFICE HOURS

Before class in my office:

→ Tue/Thu: 2:00 – 3:00 PM

→ Klaus Advanced Computing Building 3324

Things that we can talk about:

→ Ideas for research projects

→ Paper clarifications/discussion

→ Tips for Tinder/Bumble

TEACHING ASSISTANTS

Prashanth Dintyala

→ MS Computer Science

→ Software engineer @ ThoughtWorks

Sonia Matthew

→ MS Computer Science

→ Software engineer @ PayPal

COURSE RUBRIC

1. Project
2. Homeworks
3. Exams
4. Reading Reviews

1. PROJECT – OUTLINE

The main component of this course will be an original research project.

Students will organize into groups and choose to implement a project that is:

→ Relevant to the topics discussed in class.

→ Requires a significant programming effort from all team members.

1. PROJECT – OUTLINE

You don't have to pick a topic until midway through the course.

We will provide sample project topics.

This project can be a conversation starter in job interviews.

1. PROJECT – DELIVERABLES

Proposal
Project Update
Code Reviews
Final Presentation
Code Drop

1. PROJECT – PROPOSAL

Five minute presentation to the class that discusses the high-level topic.

Each proposal must discuss:

→ What is the problem being addressed by the project?
→ Why is this problem important?
→ How will the team solve this problem?

1. PROJECT – STATUS UPDATE

Five minute presentation to update the class about the current status of your project.

Each presentation should include:

→ Current development status.
→ Whether anything in your plan has changed.
→ Any thing that surprised you.

1. PROJECT – CODE REVIEWS

Each group will be paired with another group and provide feedback on their code at least two times during the semester.

Grading will be based on participation.

1. PROJECT – FINAL PRESENTATION

10 minute presentation on the final status of your project during the scheduled final exam.

You'll want to include any performance measurements or benchmarking numbers for your implementation.

Demos are always hot too…

1. PROJECT – CODE DROP

A project is **not** considered complete until:

→ All comments from code review are addressed.

→ The project includes test cases that correctly verify that implementation is correct.

→ The group provides documentation in both the source code and in separate Markdown files.

2. HOMEWORKS – OUTLINE

Homeworks will be mostly problem sets and programming assignments to familiarize you with the internals of database management systems.

We will use Gradescope for giving you immediate feedback on programming assignments and Piazza for providing clarifications regarding problem sets.

This student guide provides information on how to use Gradescope.

2. HOMEWORKS – OUTLINE

We will provide you with test cases and scripts for the programming assignments.

We will share the grading rubric for problem sets via Gradescope.

If you have not yet received an invite from Gradescope, you can use the entry code that will be shared on Piazza.

2. HOMEWORKS – HW #0

HW#0 is released today on Gradescope.

Hand in **one page** with the following information:

→ Digital picture (ideally 2x2 inches of face)
→ Name (last name, first name)
→ More details on Gradescope

2. HOMEWORKS – HW #0

The purpose of this assignment is to help me:

→ know more about your background for tailoring the course, and

→ recognize you in class

HW #0 is due on next Tuesday Jan 15th.

PLAGIARISM WARNING

These programming assignments must be all of your own code.

You may **not** copy source code from other students or the web.

Plagiarism will **not** be tolerated.

See Georgia Tech Academic Honor Code for additional information.

3. EXAMS – MID-TERM EXAM

Written long-form examination on the mandatory readings and topics discussed in class. Closed notes.

Exam will be given near the end of February.

3. EXAMS – FINAL EXAM

Take home exam. Written long-form examination on the mandatory readings and topics discussed in class.

Will be given out on the last day of class in this room.

4. READING REVIEWS – OUTLINE

One mandatory review per week().

You can skip **three** reviews during the semester.

You must submit a synopsis **before** class:

→ Overview of the main idea (one paragraph).
→ Strengths of the paper (three sentences).
→ Weaknesses of the paper (three sentences).
→ Reflections on the paper (one paragraph).

4. READING REVIEWS – OUTLINE

Submissions will be done via Gradescope

No reading reviews due this week.

First reading review will be due on Thursday next week (Jan 17th).

PLAGIARISM WARNING

Each review must be your own writing.

You may **not** copy text from the papers or other sources that you find on the web.

Plagiarism will **not** be tolerated.
See Georgia Tech Academic Honor Code for additional information.

GRADE BREAKDOWN

Project (30%)

Homeworks (30%)

Exams (30%)

Reading Reviews (10%)

COURSE MAILING LIST

On-line Discussion through Piazza:

https://piazza.com/gatech/spring2019/cs44206422a

If you have a technical question about the projects, please use Piazza.

→ Don't email me or TAs directly.

All non-project questions should be sent to me.

HISTORY OF DATABASES

WHAT GOES AROUND COMES AROUND
Readings in DB Systems, 4th Edition, 2006.

HISTORY REPEATS ITSELF

Old database issues are still relevant today.

The “SQL vs. NoSQL” debate is reminiscent of “Relational vs. CODASYL” debate.

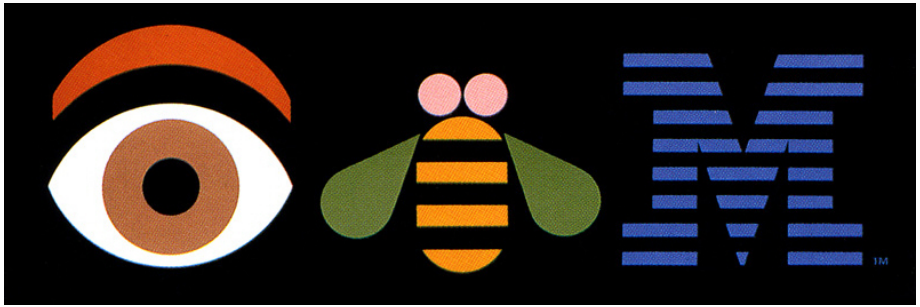
Many of the ideas in today’s database systems are not new.

1960S – IBM IMS

Information **M**anagement **S**ystem

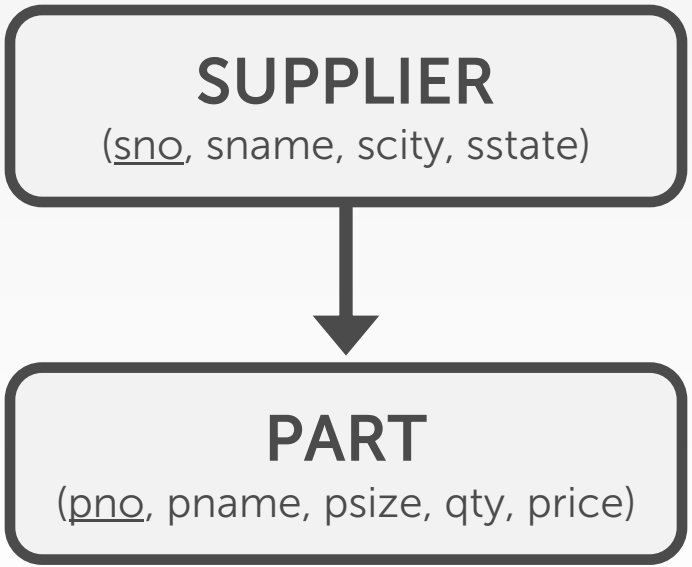
Early database system developed to keep track of purchase orders for Apollo moon mission.

→ Hierarchical data model.
→ Programmer-defined physical storage format.
→ Tuple-at-a-time queries.



HIERARCHICAL DATA MODEL

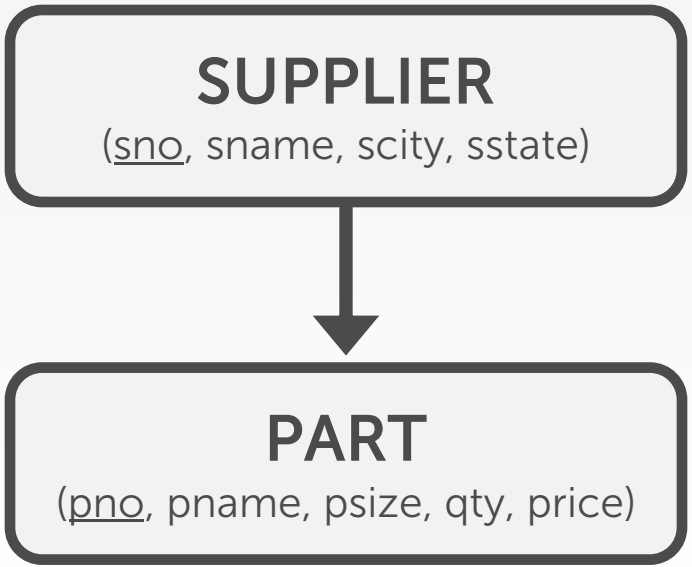
Schema



Instance

HIERARCHICAL DATA MODEL

Schema



Instance

sno	sname	scity	sstate	parts
1001	Dirty Rick	New York	NY	
1002	Squirrels	Boston	MA	

HIERARCHICAL DATA MODEL

Schema

SUPPLIER
(sno, sname, scity, sstate)

PART
(pno, pname, psize, qty, price)

Instance

sno	sname	scity	sstate	parts
1001	Dirty Rick	New York	NY	
1002	Squirrels	Boston	MA	

pno	pname	psize	qty	price
999	Batteries	Large	10	$100

HIERARCHICAL DATA MODEL

Schema

SUPPLIER
(sno, sname, scity, sstate)

PART
(pno, pname, psize, qty, price)

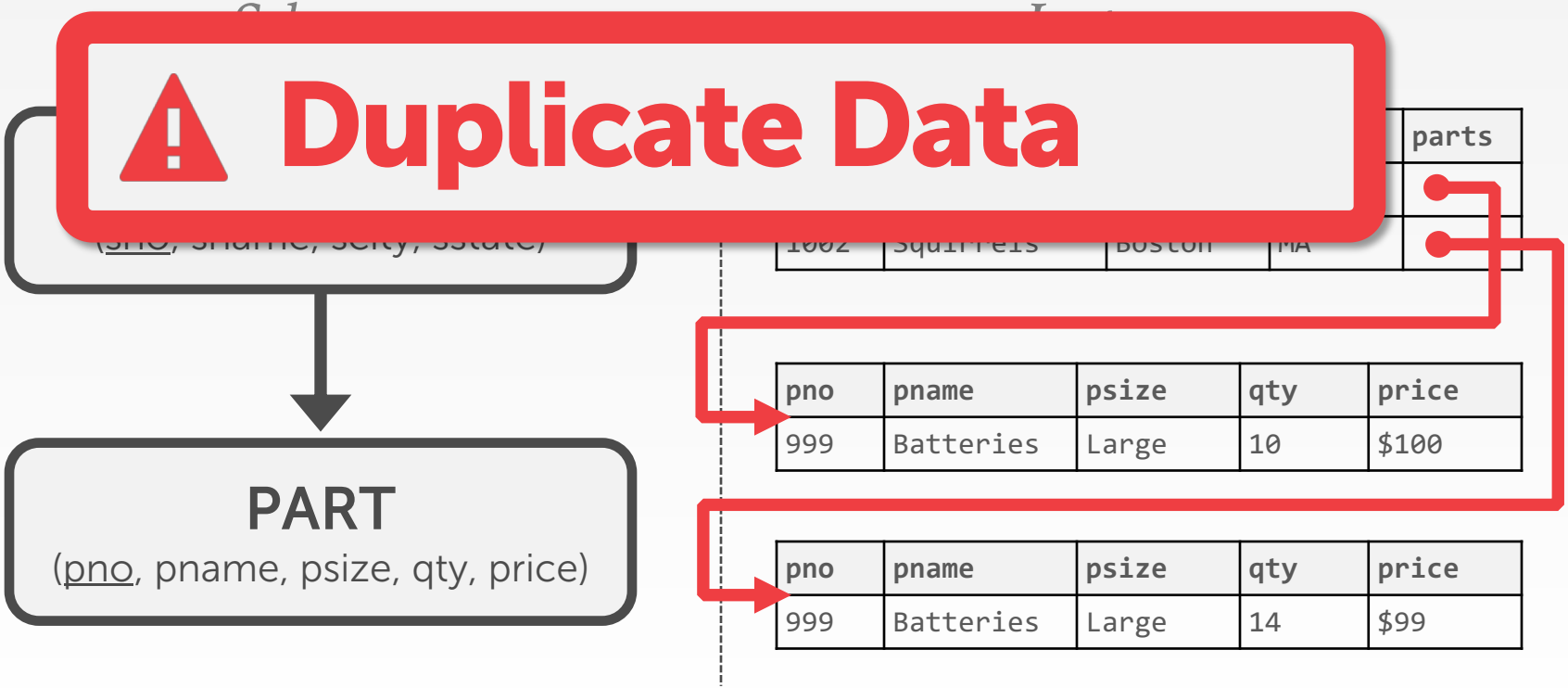
Instance

sno	sname	scity	sstate	parts
1001	Dirty Rick	New York	NY	
1002	Squirrels	Boston	MA	

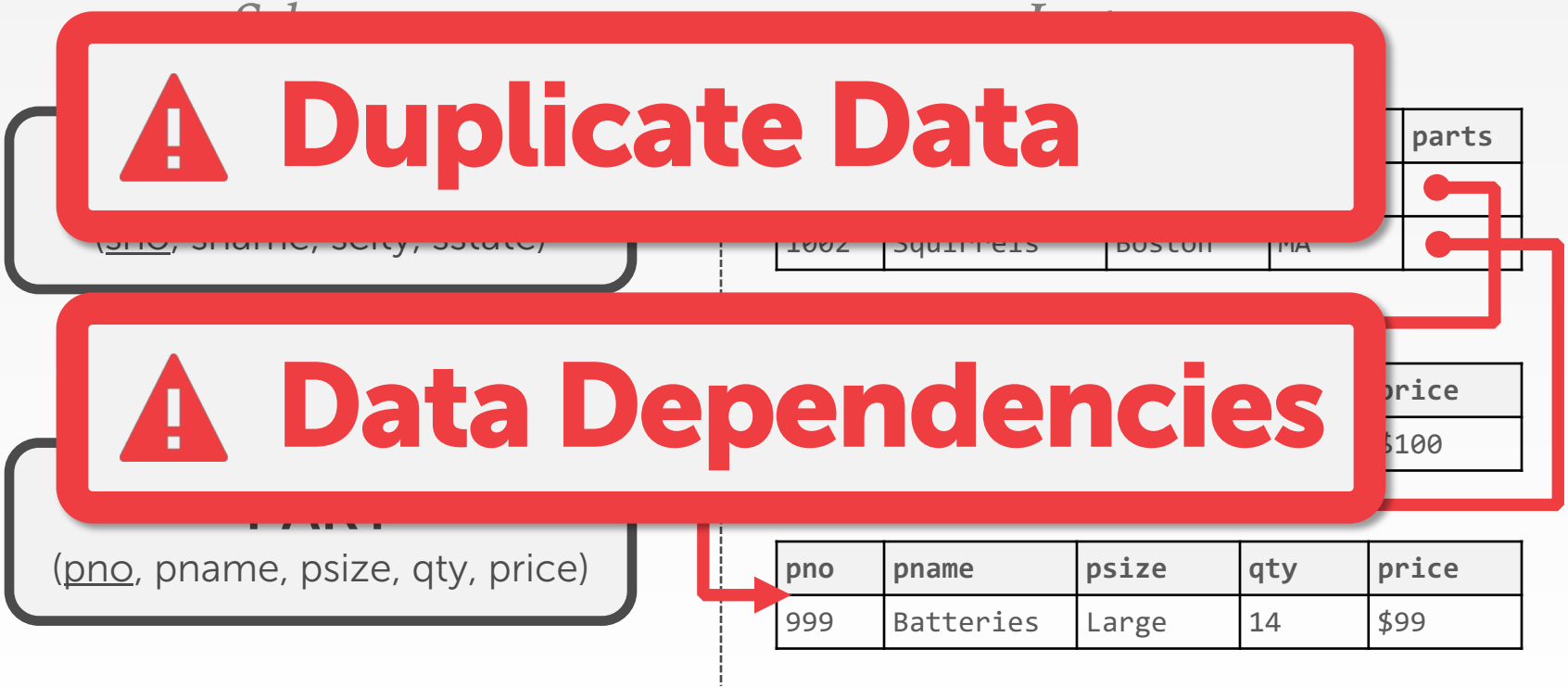
pno	pname	psize	qty	price
999	Batteries	Large	10	$100

pno	pname	psize	qty	price
999	Batteries	Large	14	$99

HIERARCHICAL DATA MODEL



HIERARCHICAL DATA MODEL



HIERARCHICAL DATA MODEL

Advantages

→ **No need to reinvent the wheel** for every application

→ **Logical data independence:** New record types may be added as the logical requirements of an application may change over time.

HIERARCHICAL DATA MODEL

Limitations

→ **Tree structured data models** are very restrictive

→ **No physical data independence**: Cannot freely change storage organizations to tune a database application because there is no guarantee that the applications will continue to run

→ **Optimization:** A tuple-at-a-time user interface forces the programmer to do manual query optimization, and this is often hard.

1960s – IDS

Integrated **D**ata **S**tore

Developed internally at GE in the early 1960s.

GE sold their computing division to Honeywell in 1969.

One of the first DBMSs:

→ Network data model.

→ Tuple-at-a-time queries.

Honeywell

1960s – CODASYL

COBOL people got together and proposed a standard for how programs will access a database. Lead by Charles Bachman.

→ Network data model.

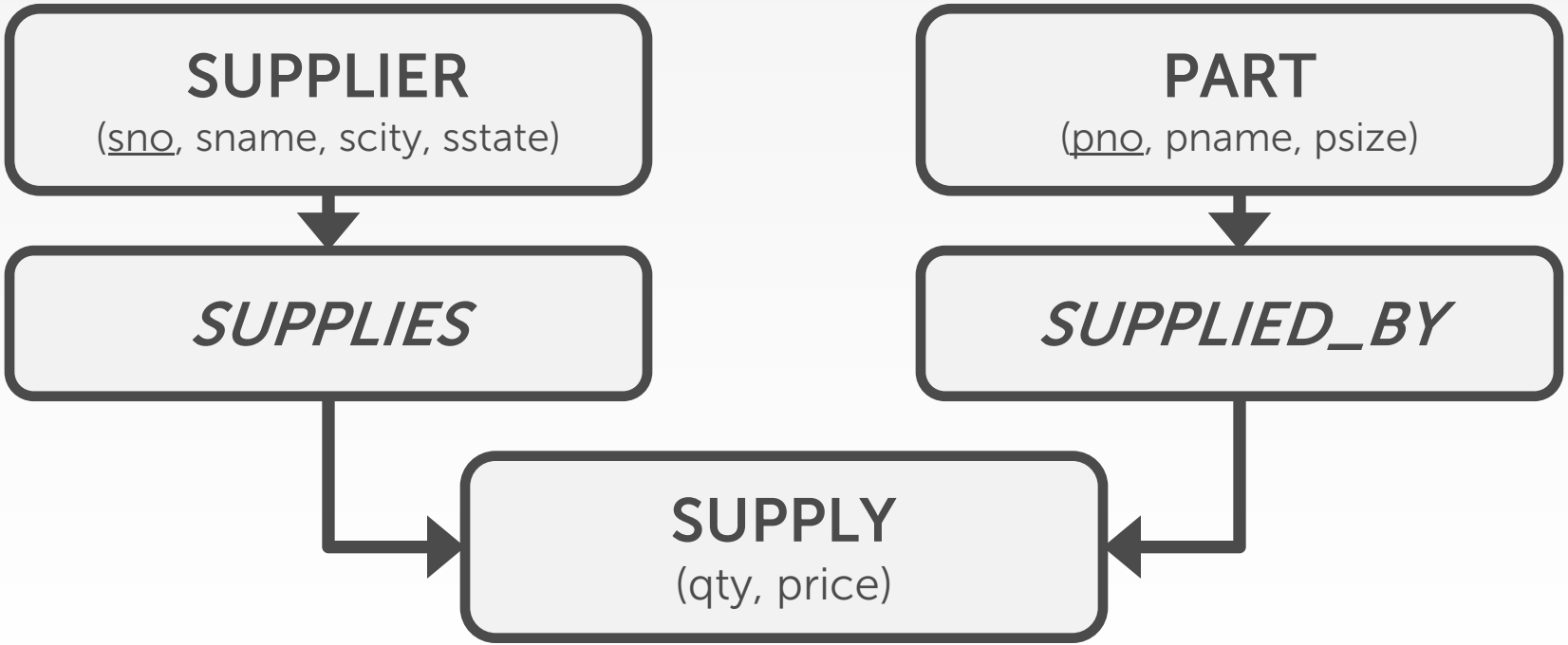
→ Tuple-at-a-time queries.

Bachman

Bachman also worked at Culliane Database Systems in the 1970s to help build **IDMS**.

NETWORK DATA MODEL

Schema

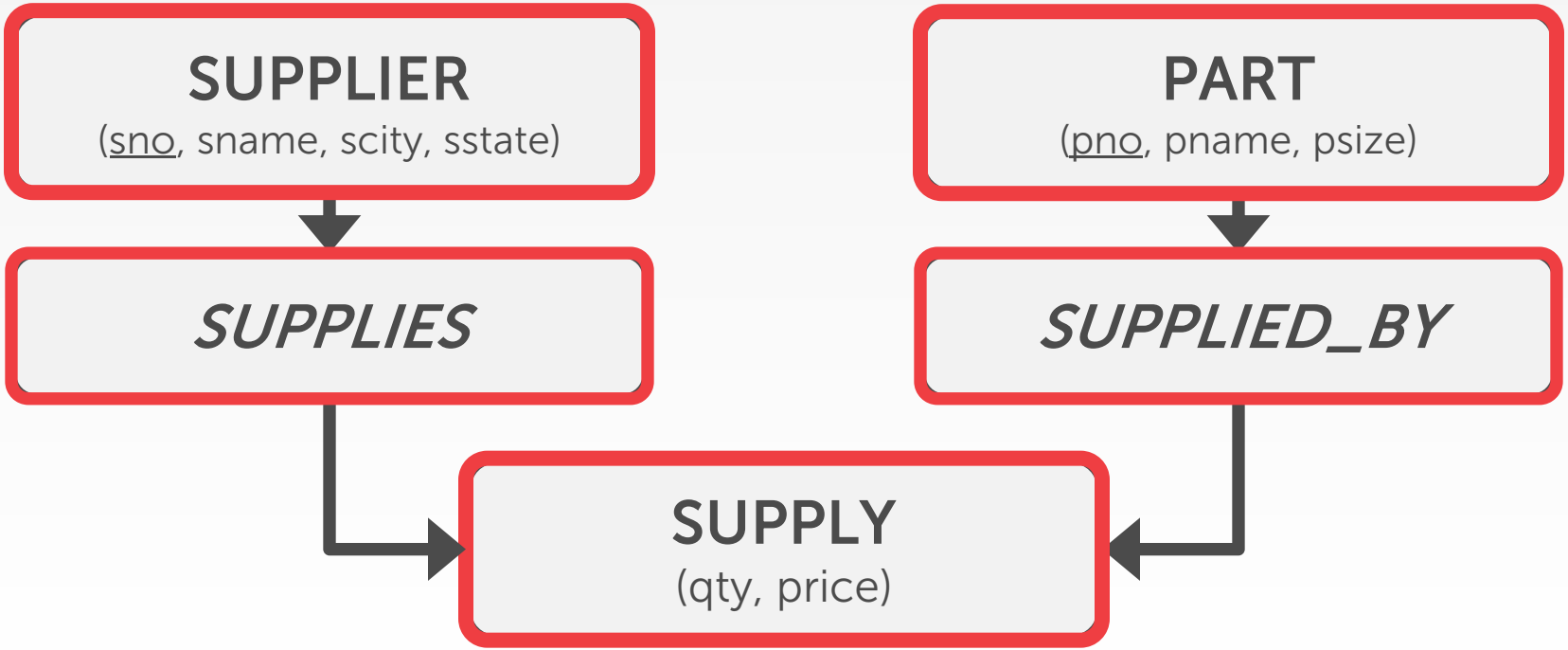


NETWORK DATA MODEL

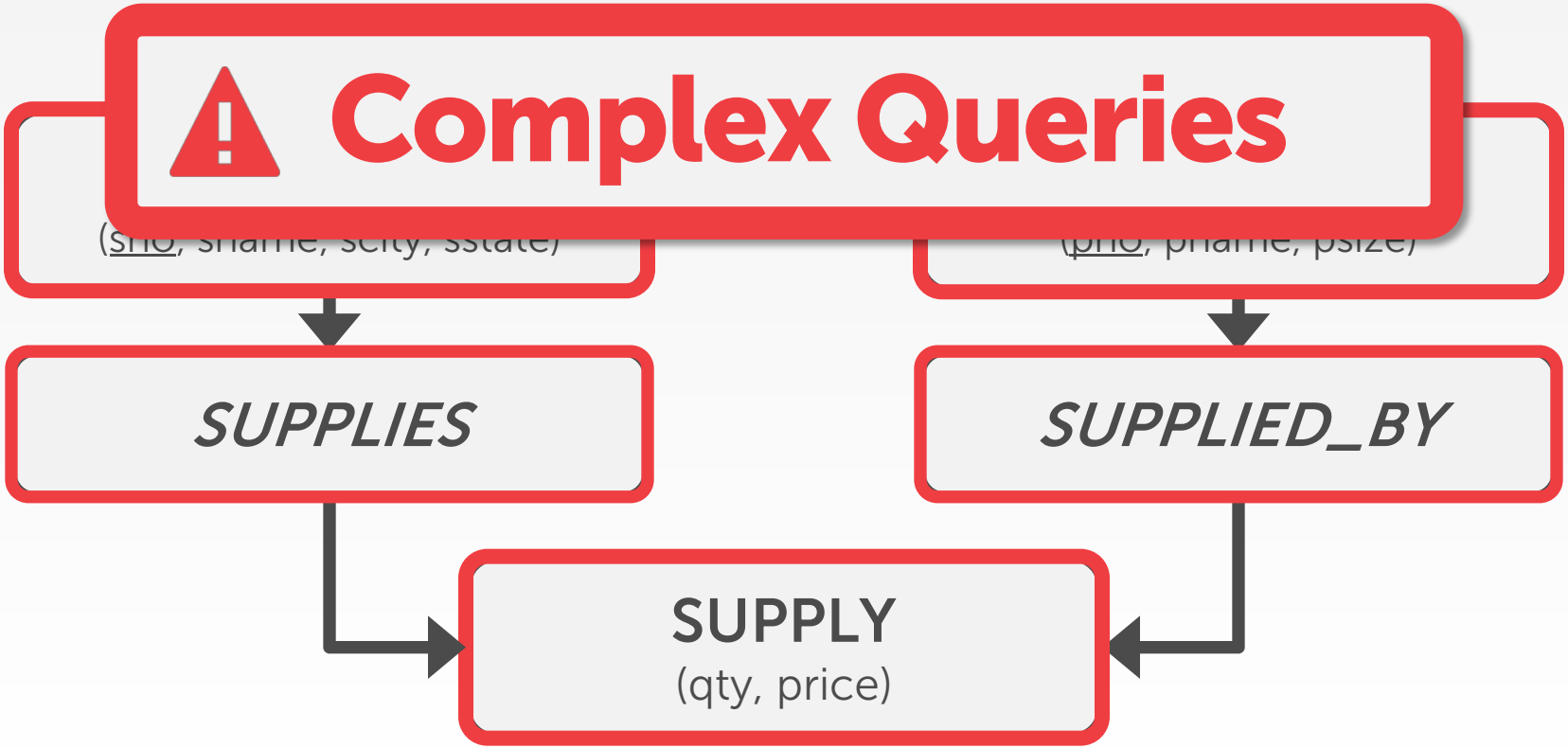
Schema

NETWORK DATA MODEL

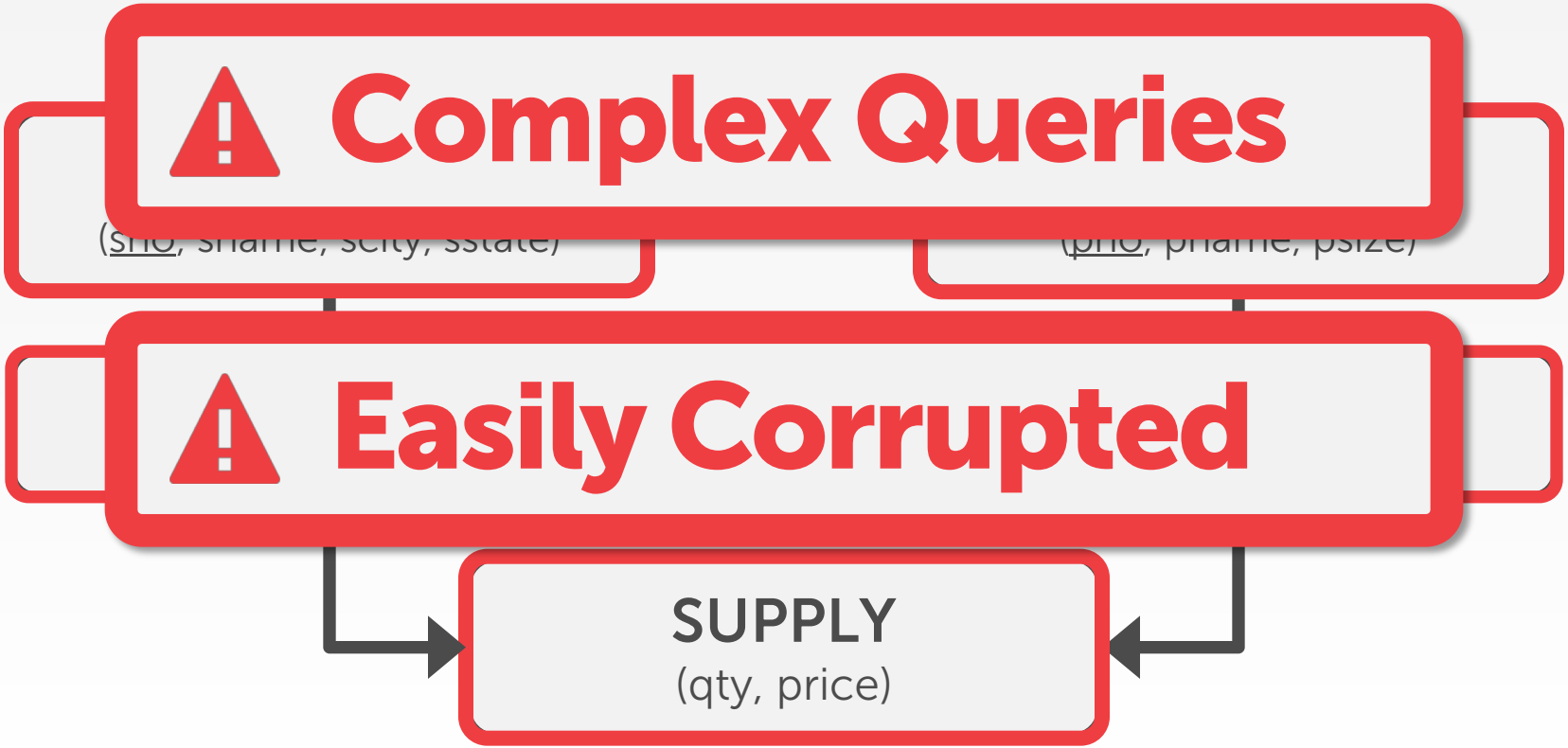
Schema



NETWORK DATA MODEL



NETWORK DATA MODEL



NETWORK DATA MODEL

Advantages

→ **Graph structured data models** are less restrictive

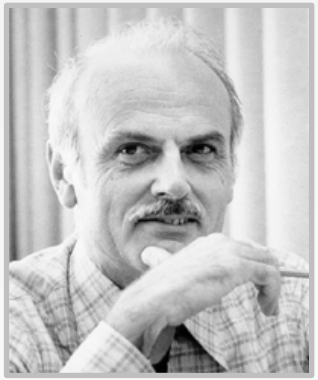
Limitations

→ **Poorer physical and logical data independence**: Cannot freely change storage organizations or change application schema

→ **Slow loading and recovery:** Data is typically stored in one large network. This much larger object had to be bulk-loaded all at once, leading to very long load times.

1970s – RELATIONAL MODEL

Ted Codd was a mathematician working at IBM Research. He saw developers spending their time rewriting IMS and Codasyl programs every time the database's schema or layout changed.



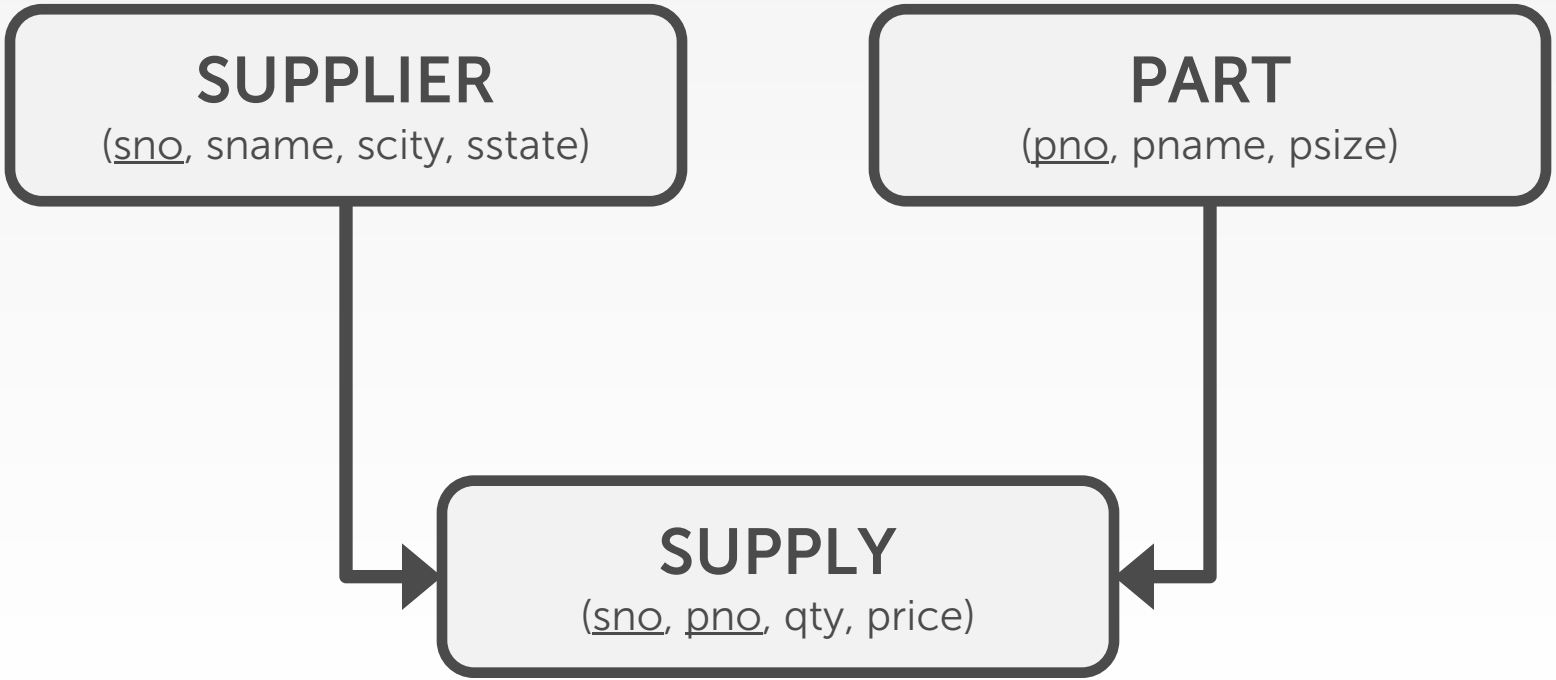
Codd

Database abstraction to avoid this maintenance:

→ Store database in simple data structures.
→ Access data through high-level declarative language.
→ Physical storage left up to implementation.

RELATIONAL DATA MODEL

Schema



RELATIONAL DATA MODEL

Schema

RELATIONAL DATA MODEL

Schema

RELATIONAL DATA MODEL

Schema

A Relational Model of Data for Large Shared Data Banks

E. F. CODD
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on *n*-ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the user's model.

KEY WORDS AND PHRASES: data bank, data base, data structure, data organization, hierarchies of data, networks of data, relations, derivability, redundancy, consistency, composition, join, retrieval language, predicate calculus, security, data integrity
CR CATEGORIES: 3.70, 3.73, 3.75, 4.20, 4.22, 4.29

1. Relational Model and Normal Form

1.1. Introduction

This paper is concerned with the application of elementary relation theory to systems which provide shared access to large banks of formatted data. Except for a paper by Childs [1], the principal application of relations to data systems has been to deductive question-answering systems. Levein and Maron [2] provide numerous references to work in this area.

In contrast, the problems treated here are those of *data independence*—the independence of application programs and terminal activities from growth in data types and changes in data representation—and certain kinds of *data inconsistency* which are expected to become troublesome even in nondeductive systems.

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for noninferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the "connection trap").

Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted data systems, and also the relative merits (from a logical standpoint) of competing representations of data within a single system. Examples of this clearer perspective are cited in various parts of this paper. Implementations of systems to support the relational model are not discussed.

1.2. Data Dependencies in Present Systems

The provision of data description tables in recently developed information systems represents a major advance toward the goal of data independence [5, 6, 7]. Such tables facilitate changing certain characteristics of the data representation stored in a data bank. However, the variety of data representation characteristics which can be changed *without logically impairing some application programs* is still quite limited. Further, the model of data with which users interact is still cluttered with representational properties, particularly in regard to the representation of collections of data (as opposed to individual items). Three of the principal kinds of data dependencies which still need to be removed are: ordering dependence, indexing dependence, and access path dependence. In some systems these dependencies are not clearly separable from one another.

1.2.1. *Ordering Dependence.* Elements of data in a data bank may be stored in a variety of ways, some involving no concern for ordering, some permitting each element to participate in one ordering only, others permitting each element to participate in several orderings. Let us consider those existing systems which either require or permit data elements to be stored in at least one total ordering which is closely associated with the hardware-determined ordering of addresses. For example, the records of a file concerning parts might be stored in ascending order by part serial number. Such systems normally permit application programs to assume that the order of presentation of records from such a file is identical to (or is a subordering of) the

RELATIONAL DATA MODEL

Advantages

→ **Set-a-time languages** are good, regardless of the data model, since they offer physical data independence

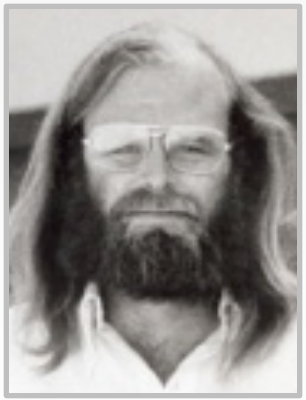
→ **Logical data independence** is easier with a simple data model than with a complex one.

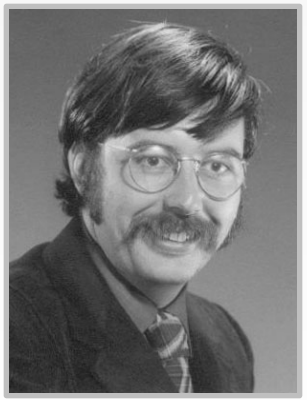
→ **Query optimizers** can beat all but the best tuple-at-a-time DBMS application programmers.

1970s – RELATIONAL MODEL

Early implementations of relational DBMS:

→ **System R** – IBM Research
→ **INGRES** – U.C. Berkeley
→ **Oracle** – Larry Ellison


Gray


Stonebraker


Ellison

1980s – RELATIONAL MODEL

The relational model wins.
→ IBM comes out with DB2 in 1983.
→ "SEQUEL" becomes the standard (SQL).

Many new "enterprise" DBMSs
but Oracle wins marketplace.

1980s – OBJECT-ORIENTED DATABASES

Avoid "relational-object impedance mismatch" by tightly coupling objects and database.

Few of these original DBMSs from the 1980s still exist today but many of the technologies exist in other forms (JSON, XML)

VERSANT ObjectStore MarkLogic™

OBJECT-ORIENTED MODEL

Application Code

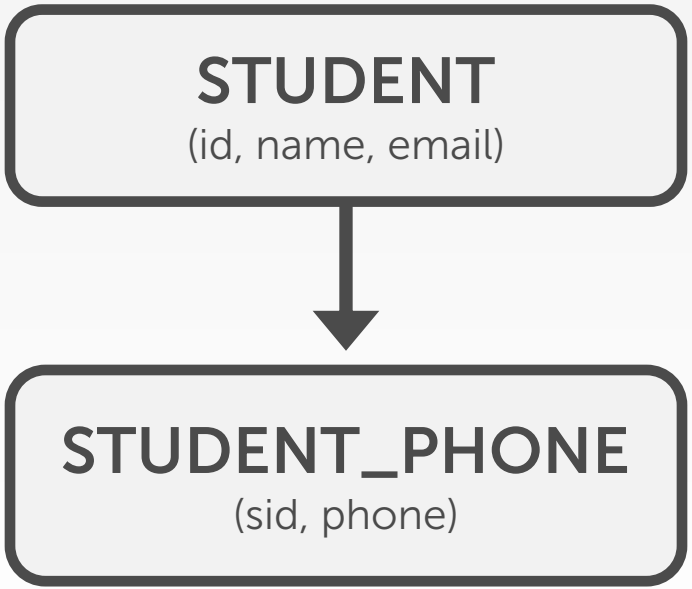
```
class Student {
   int id;
   String name;
   String email;
   String phone[];
}
```

OBJECT-ORIENTED MODEL

Application Code

```
class Student {
    int id;
    String name;
    String email;
    String phone[];
}
```

Relational Schema



OBJECT-ORIENTED MODEL

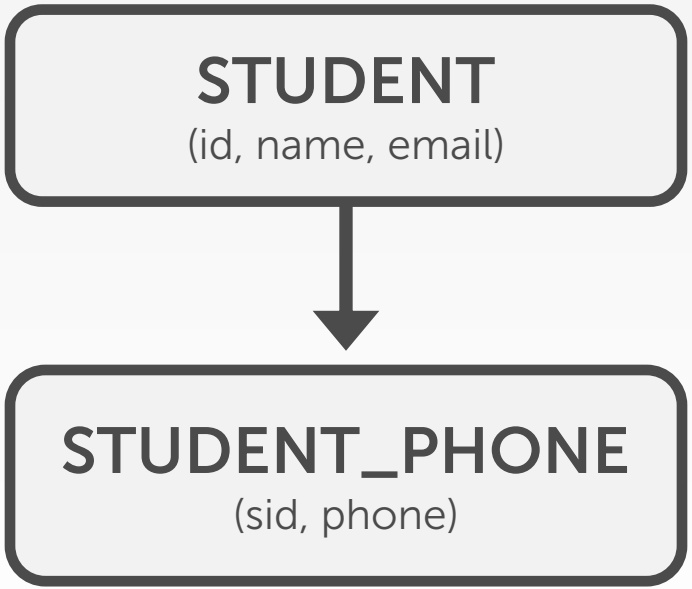
Application Code

```
class Student {
    int id;
    String name;
    String email;
    String phone[];
}
```

id	name	email
1001	M.O.P.	ante@up.com

sid	phone
1001	444-444-4444
1001	555-555-5555

Relational Schema



OBJECT-ORIENTED MODEL

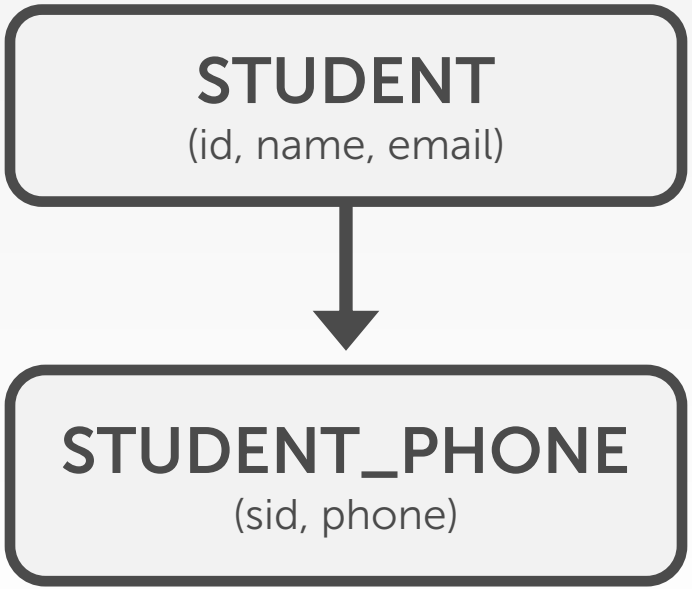
Application Code

```
class Student {
    int id;
    String name;
    String email;
    String phone[];
}
```

id	name	email
1001	M.O.P.	ante@up.com

sid	phone
1001	444-444-4444
1001	555-555-5555

Relational Schema



OBJECT-ORIENTED MODEL

Application Code

```
class Student {
   int id;
   String name;
   String email;
   String phone[];
}
```

OBJECT-ORIENTED MODEL

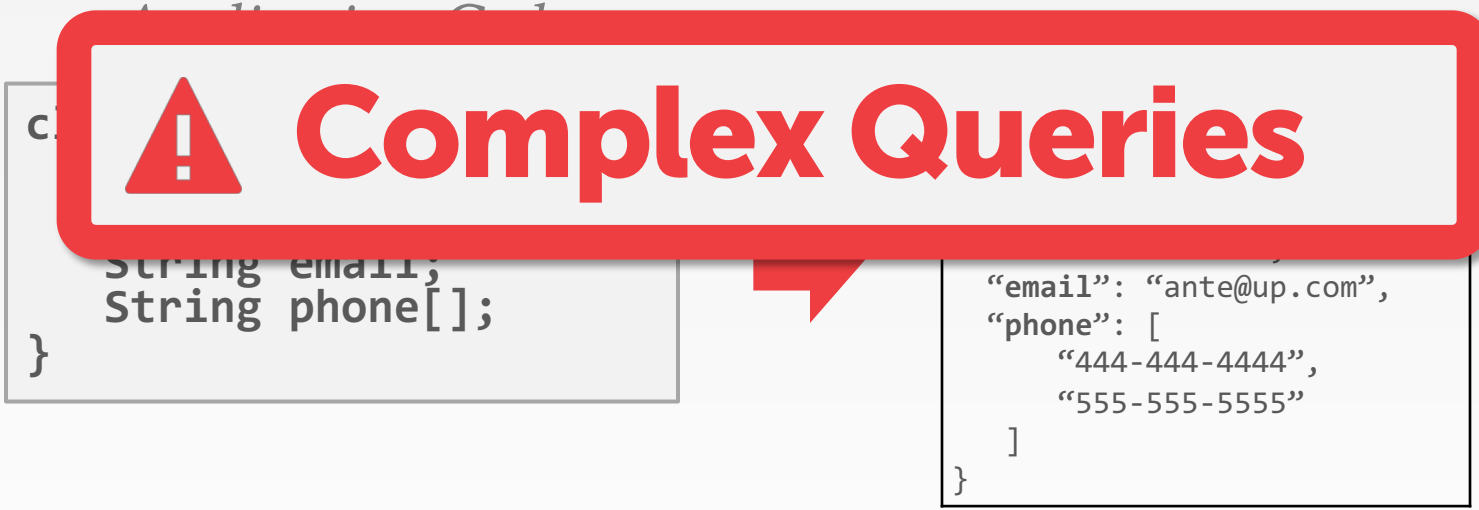
Application Code

```
class Student {
    int id;
    String name;
    String email;
    String phone[];
}
```

Student

```
{
  “id”: 1001,
  “name”: “M.O.P.”,
  “email”: “ante@up.com”,
  “phone”: [
      “444-444-4444”,
      “555-555-5555”
   ]
}
```

OBJECT-ORIENTED MODEL



OBJECT-ORIENTED MODEL

1990s – BORING DAYS

No major advancements in database systems or application workloads.

→ Microsoft forks Sybase and creates SQL Server.
→ MySQL is written as a replacement for mSQL.
→ Postgres gets SQL support.
→ SQLite started in early 2000.

2000s – INTERNET BOOM

All the big players were heavyweight and expensive. Open-source databases were missing important features.

Many companies wrote their own custom middleware to scale out database across single-node DBMS instances.

2000s – DATA WAREHOUSES

Rise of the special purpose OLAP DBMSs.
→ Distributed / Shared-Nothing
→ Relational / SQL
→ Usually closed-source.

Significant performance benefits from using Decomposition Storage Model (i.e., columnar)

2000s – NoSQL SYSTEMS

Focus on high-availability & high-scalability:

→ Schemaless (i.e., "Schema Last")
→ Non-relational data models (document, key/value, etc)
→ No ACID transactions
→ Custom APIs instead of SQL
→ Usually open-source

2010s – NewSQL

Provide same performance for OLTP workloads as NoSQL DBMSs without giving up ACID:

→ Relational / SQL
→ Distributed
→ Usually closed-source

2010s – HYBRID SYSTEMS

Hybrid Transactional-Analytical Processing.

Execute fast OLTP like a NewSQL system while also executing complex OLAP queries like a data warehouse system.

→ Distributed / Shared-Nothing
→ Relational / SQL
→ Mixed open/closed-source.

2010s – CLOUD SYSTEMS

First database-as-a-service (DBaaS) offerings were "containerized" versions of existing DBMSs.

There are new DBMSs that are designed from scratch explicitly for running in a cloud environment.

2010s – SPECIALIZED SYSTEMS

Shared-disk DBMSs

Embedded DBMSs

Times Series DBMS

Multi-Model DBMSs

Blockchain DBMSs

2010s – SPECIALIZED SYSTEMS

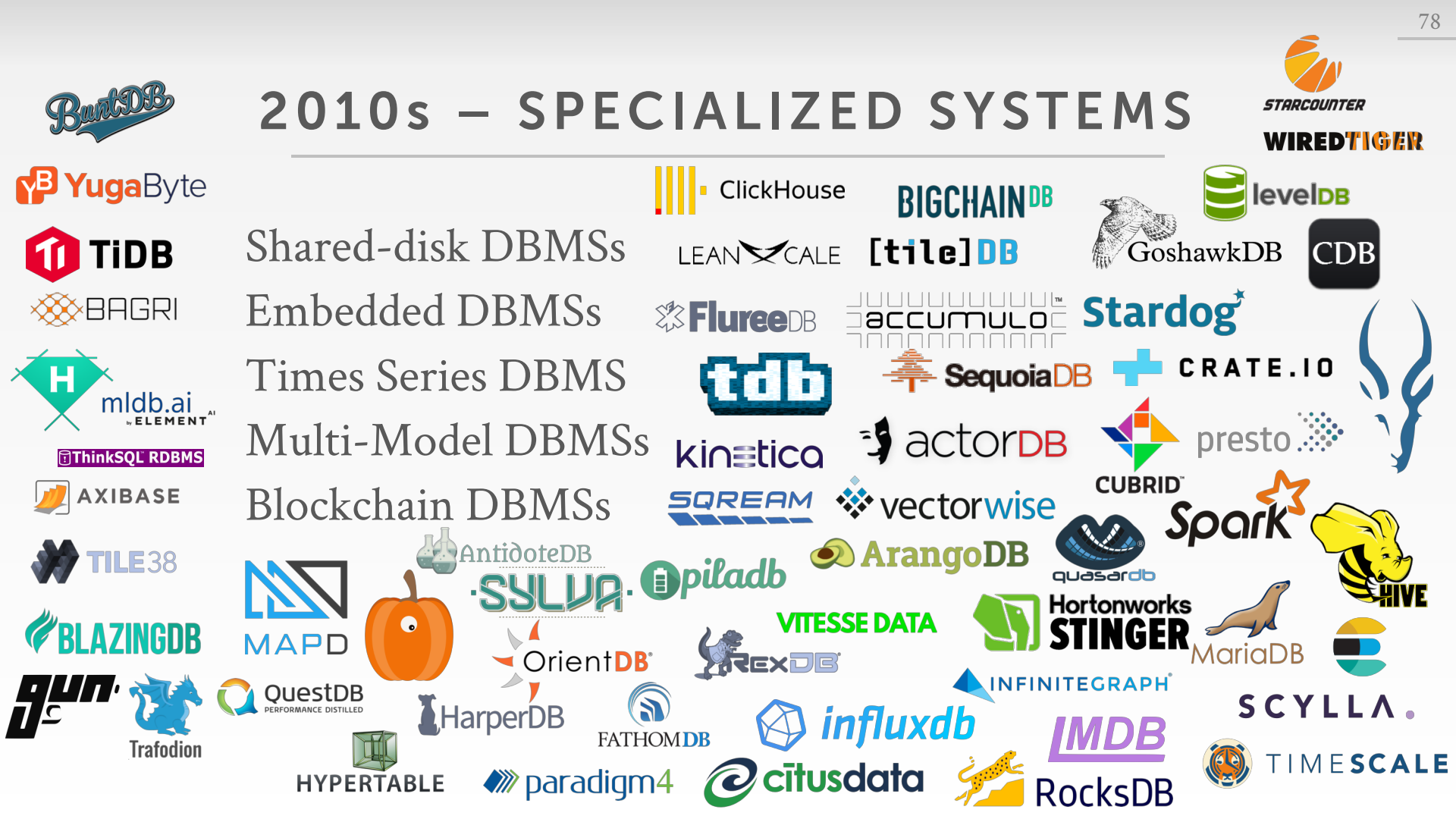
Shared-disk DBMSs

Embedded DBMSs

Times Series DBMS

Multi-Model DBMSs

Blockchain DBMSs



PARTING THOUGHTS

There are many innovations that come from both industry and academia:

→ Lots of ideas start in academia but few build complete DBMSs to verify them.
→ IBM was the vanguard during 1970-1980s but now there is no single trendsetter.
→ Oracle borrows ideas from anybody.

The relational model has won for operational databases.

NEXT CLASS

Disk vs. In-Memory DBMSs

Reminder: Homework 0 is due on Tuesday Jan 15th. Submit via Gradescope.