CS 4440 A

Emerging Database Technologies

Lecture 1

01/12/26

Agenda

Course logistics and overview

A brief history of databases

- 1960s – 2020s

The essentials

Instructor: Kexin Rong

- Office: Klaus 3322

TAs:

- Yihao Mai
- Tianji Yang

How to reach us: cs4440-staff@groups.gatech.edu

- The above email reaches all of the course staff. You are strongly encouraged to use this, instead of emailing individual course staff.

The essentials

Course website: https://kexinrong.github.io/sp26-cs4440/

schedule and course material

Canvas/Gradescope : submitting assignments

Piazza: discussing course contents

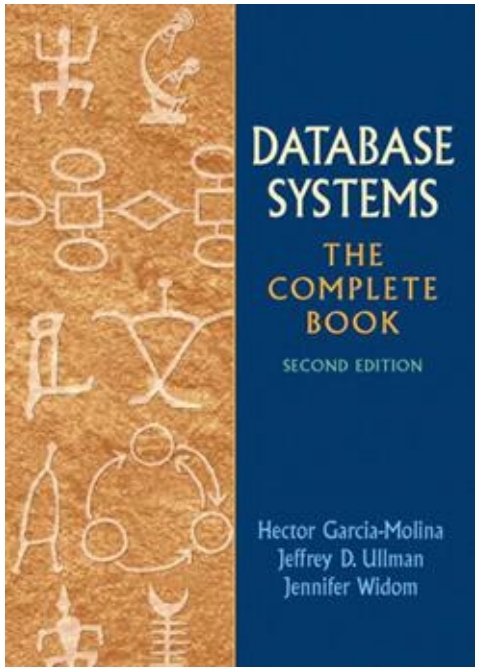
- https://piazza.com/gatech/spring2026/cs4440a/home

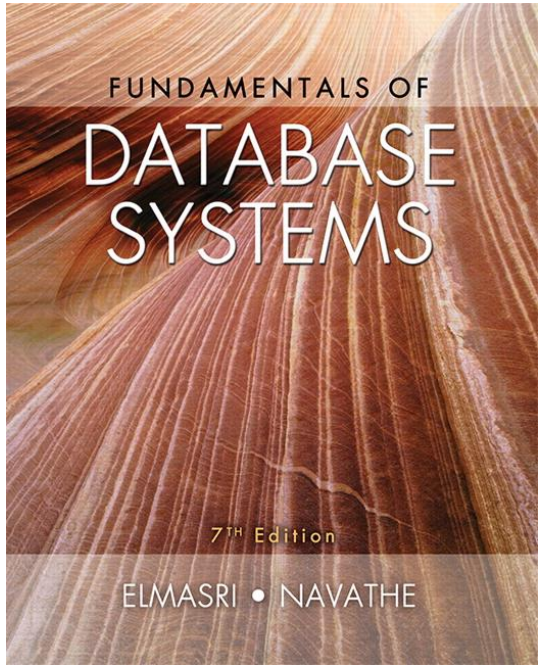
Email: special requests

OH: Starting next week. Time will be announced.

Course materials

- Textbooks:
  - Database Systems: The Complete Book (2nd edition)
  - Fundamentals of Database Systems
  - Can use interchangeably
- Both books have international versions and have PDFs searchable online





Course Learning Objectives

Learn about advanced and emerging database technologies beyond what is covered in CS4400.

Multiple ways to learn:

- Through lectures on database fundamentals
- Through surveying technologies in the wild
- Through hands-on implementations

Grading

Assignments – 58%

- Combination of individual and group assignments

Exams – 40%

- Midterm (in-class) – 20%
- Final (take-home) – 20%

Participation - 2%

- Based on attendance/participation for required lectures

https://kexinrong.github.io/sp26-cs4440/grading/

Assignments Overview

AS1: Technology Review (5%)
- Released this Wednesday

AS2: Failure Recovery (15%)
- Programming assignment

AS3: Technology Presentation (8%)
- Group-based

AS4: Postgres Parquet Extension (15%)
- Programming assignment

AS5: NL2SQL (15%)
- Open-ended assignment

Exams

Written tests based on material covered in lectures

- Exam 1: in-class (Feb 16) – 20%
- Exam 2: take-home (final's week) – 20%
  - focus on materials that are not covered by Exam 1

Attendance

I don't believe in mandatory attendance. But in the past we noticed…

- People who did not attend did worse ☹
- People who did not attend used more course resources ☹
- People who did not attend were less happy with the course ☹

This year's policy:

- **Required attendance** will be announced in advance for a few classes (guest lectures, student presentations).
  - Attendance at these required sessions counts toward your participation grade.
- Voluntary attendance for all other class sessions.

Course Policy - IMPORTANT

Follow the Georgia Tech Honor Code!

Late policy: One automatic late day without penalty. Otherwise 10% deduction per 24 hours. Does not apply to group assignments and exams.

Makeup exam policy: NO makeup exam for midterm.

Generative AI policy: Clearly attribute AI-generated contents (e.g., direct quotes, different color text). Do not use generative AI tools to write code for you.

Details: https://kexinrong.github.io/sp26-cs4440/policy/

Course Outline

1. How can one use a DBMS (programmer's/designer's perspective)

- We will NOT teach SQL
- Design a good database (design theory)

2. How does a DBMS work (system's perspective, also for programmers for writing better queries)

- Transactions: concurrency control and recovery
- Physical design: storage and index
- Query processing and optimization

3. Beyond relational databases

- Parallel and distributed DBMS
- Map Reduce, Spark, NoSQL, NewSQL

A brief history of databases (1960s-2020s)

What Goes Around Comes Around.
Readings in DB Systems. 2006.

Acknowledgement: The following slides were adapted from Prof. Andy Pavlo (CMU)

Main takeaway: history repeats itself

Old database issues are still relevant today.

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

Someone invents a "SQL replacement" every decade. It then fails and/or SQL absorbs the key ideas into standards.

- The SQL vs. NoSQL debate is reminiscent of Relational vs. CODASYL debate from the 1970s.
- Spoiler: The relational model almost always wins.

1960s - IDS

- Integrated Data Store
- 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

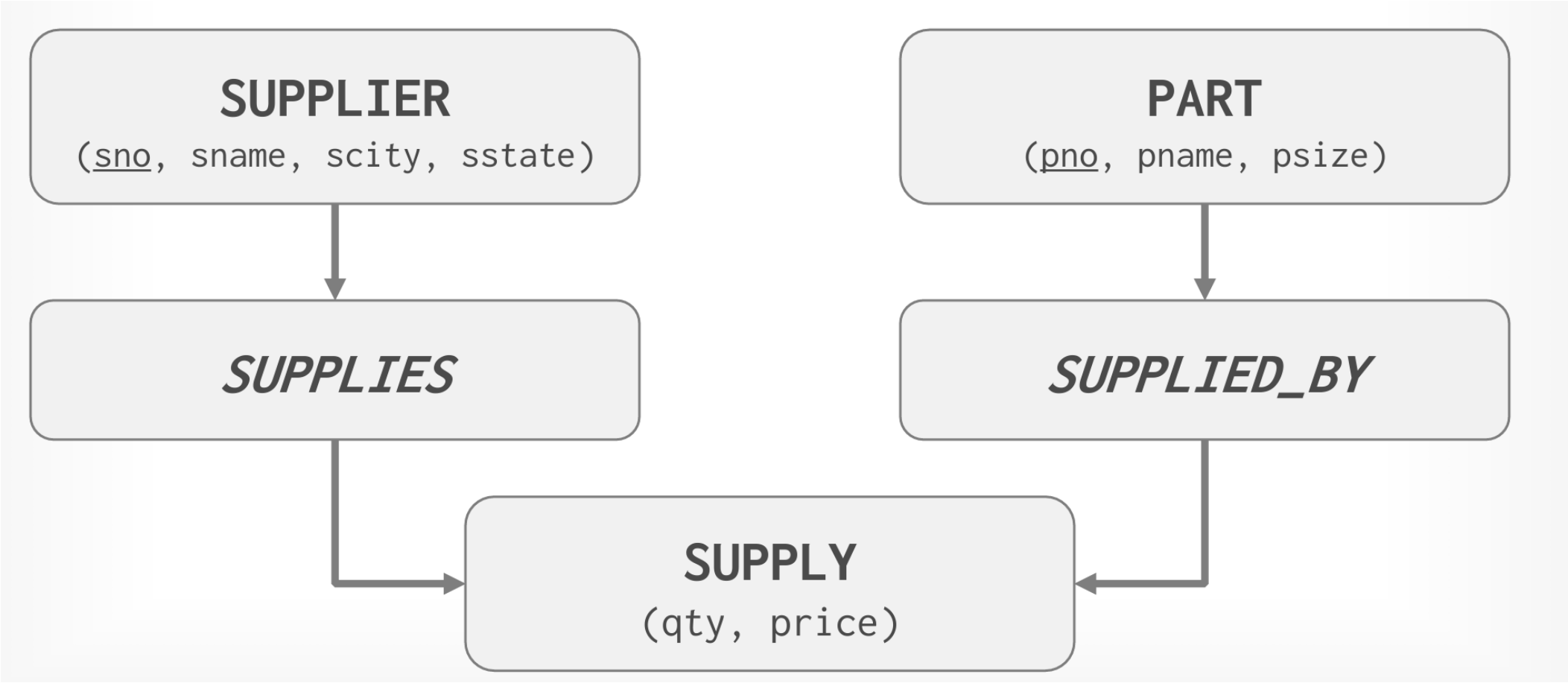
Turing Award 1973

- 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.

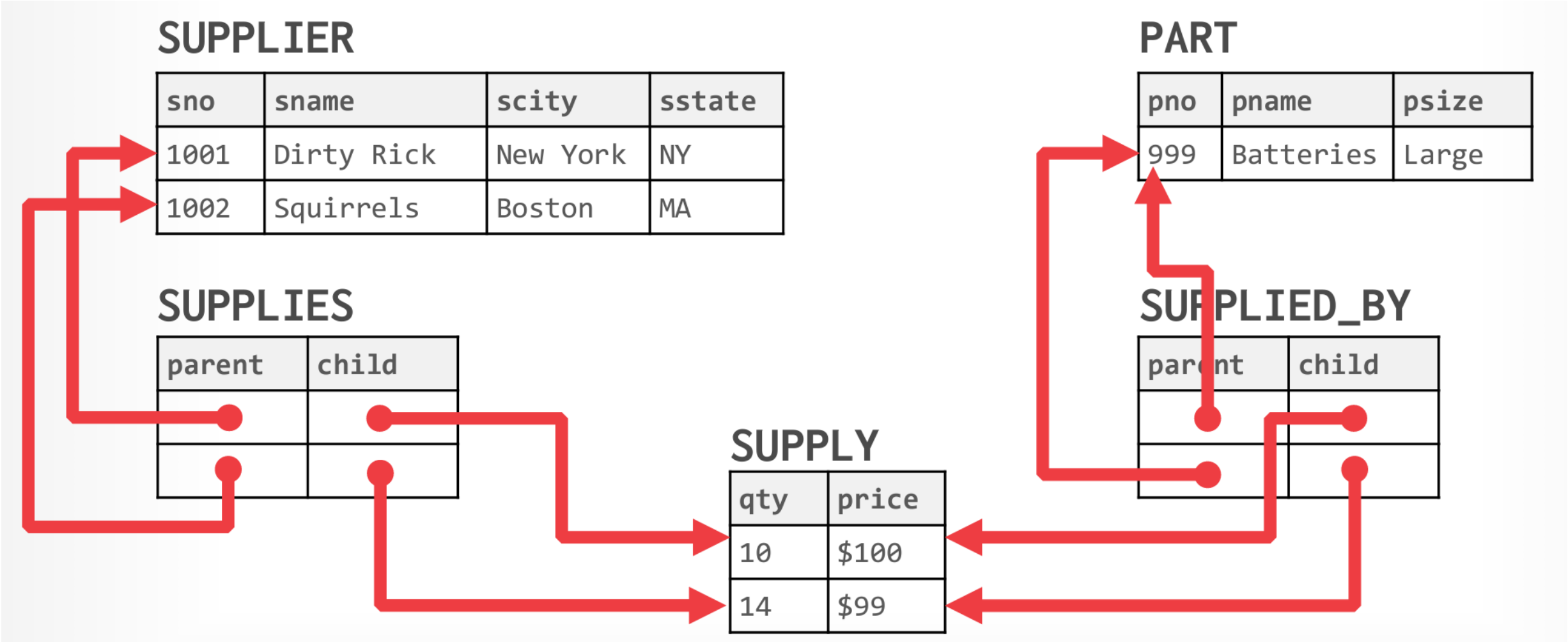
- Bachman also worked at Culliane Database Systems in the 1970s to help build **IDMS**.

Bachman

Network data model - *schema*

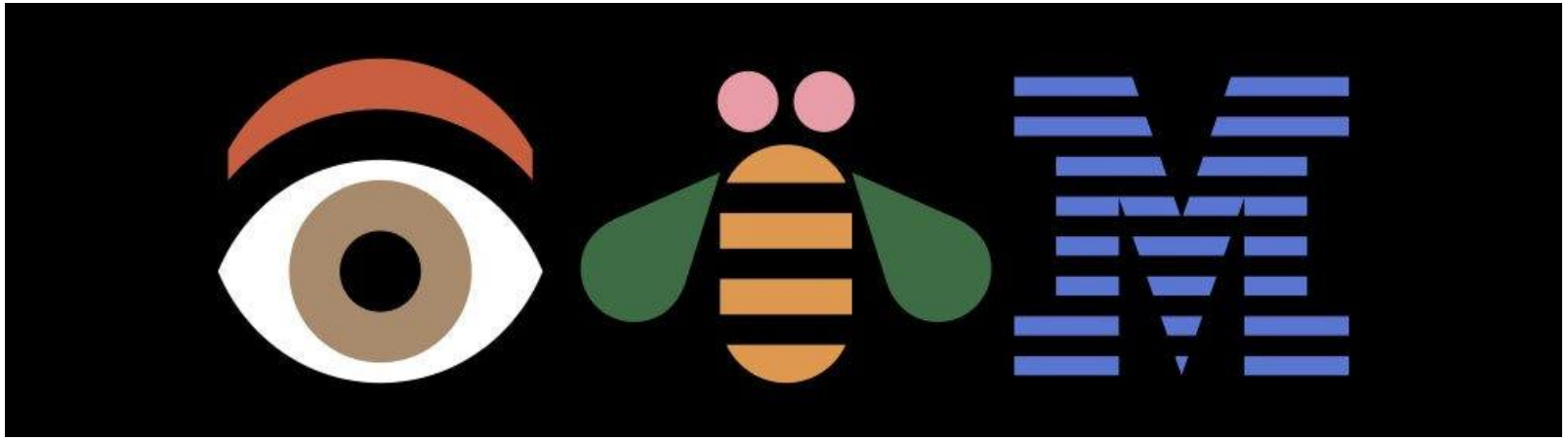


Network data model - *instance*

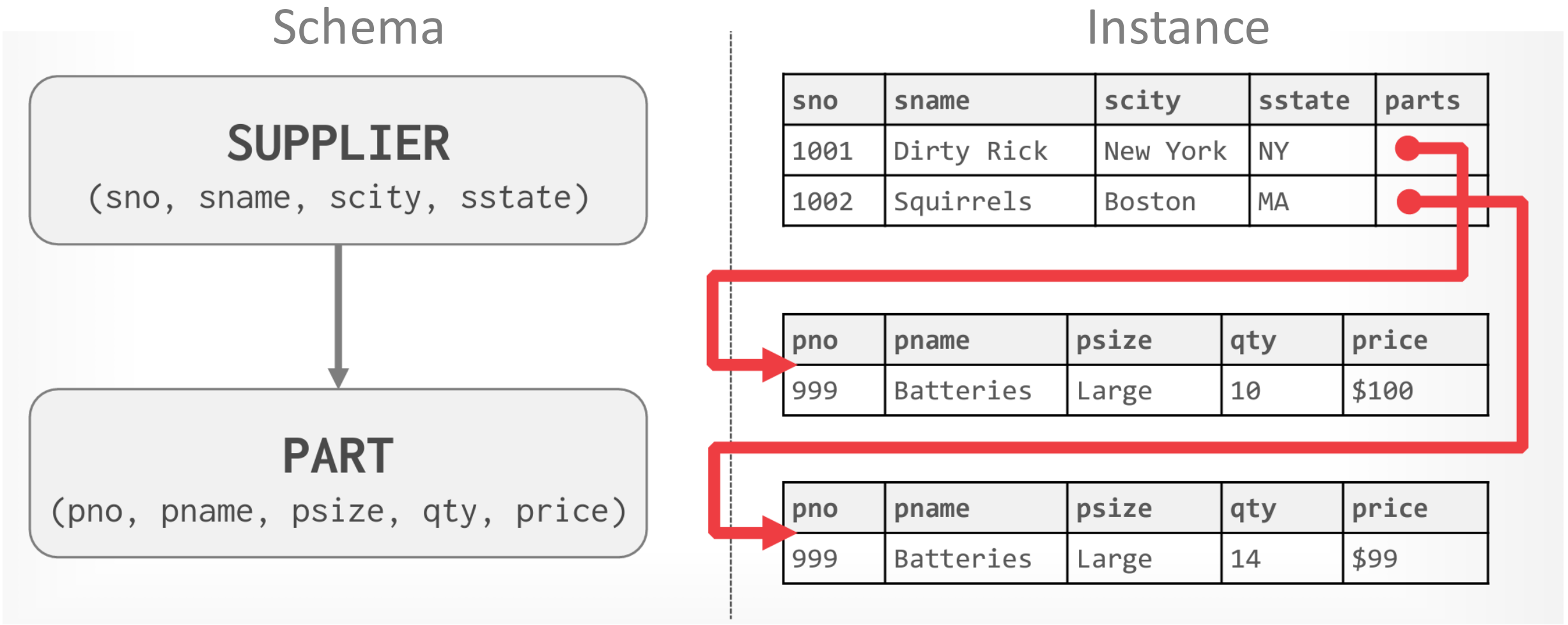


1960s – IBM IMS

- Information Management System
- 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

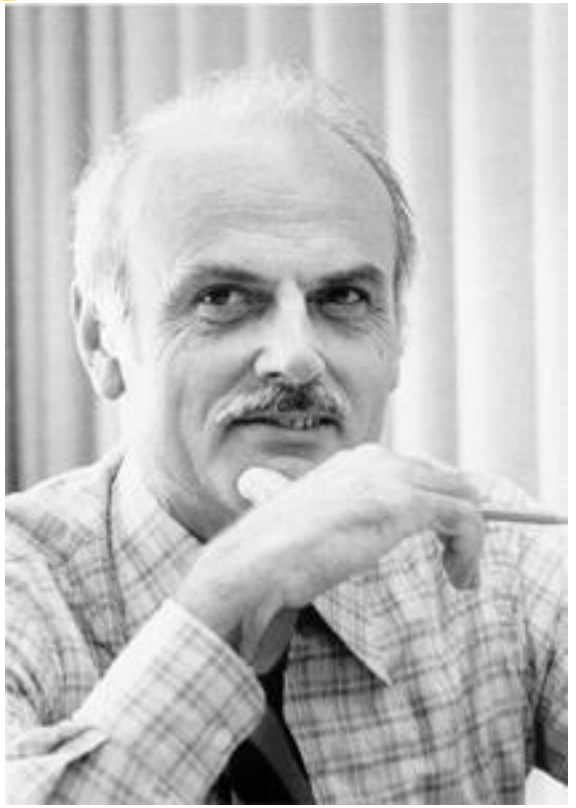


1970s - Relational data model

- 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.

- Database abstraction to avoid this maintenance:
  - Store database in simple data structures.
  - Access data through set-at-a-time high-level language.
  - Physical storage left up to implementation.

Turing Award 1981



Codd

From Jim Gray

DERIVABILITY, REDUNDANCY AND CONSISTENCY OF RELATIONS
STORED IN LARGE DATA BANKS

E. F. Codd
Research Division
San Jose, California

ABSTRACT: The large, integrated data banks of the future will contain many relations of various degrees in stored form. It will not be unusual for this set of stored relations to be redundant. Two types of redundancy are defined and discussed. One type may be employed to improve accessibility of certain kinds of information which happen to be in great demand. When either type of redundancy exists, those responsible for control of the data bank should know about it and have some means of detecting any "logical" inconsistencies in the total set of stored relations. Consistency checking might be helpful in tracking down unauthorized (and possibly fraudulent) changes in the data bank contents.

RJ 599(# 12343) August 19, 1969

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").

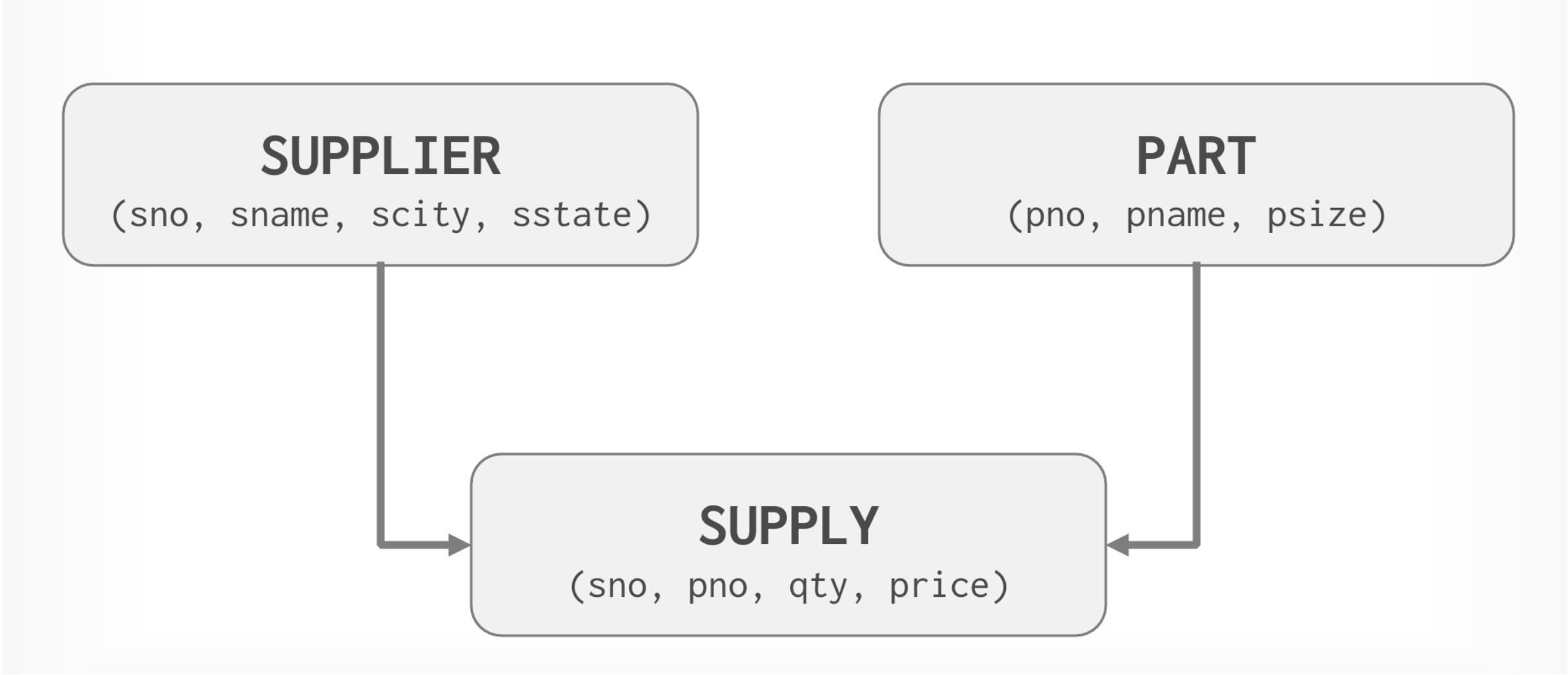
Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted data systems, and also the relative merits (from a logical standpoint) of competing representations of data within a single system. Examples of this clearer perspective are cited in various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

1.2. Data Dependencies in Present Systems

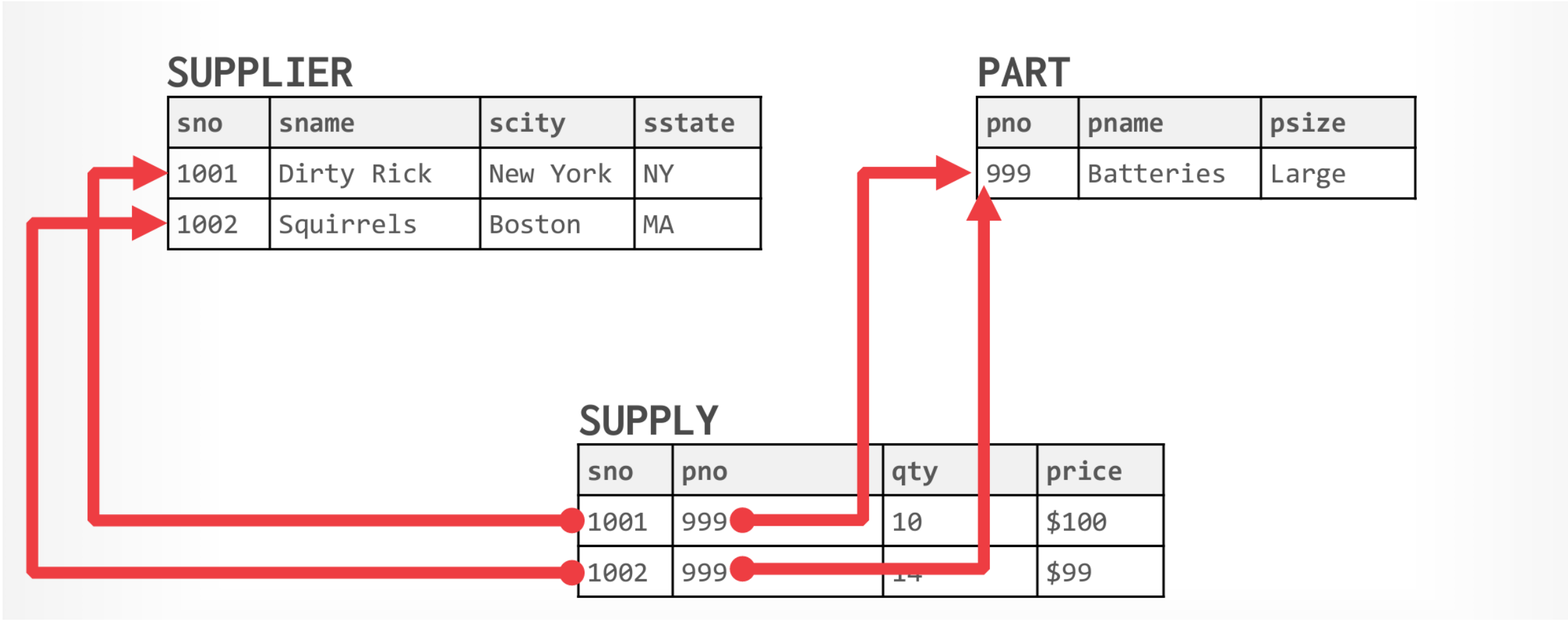
The provision of data description tables in recently developed information systems represents a major advance toward the goal of data independence [5, 6, 7]. Such tables facilitate changing certain characteristics of the data representation stored in a data bank. However, the variety of data representation 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 - *schema*



Relational Data Model - *instance*



1970s – Relational Model

- Early implementations of relational DBMS:
  - Peterlee Relational Test Vehicle – IBM Research (UK)
  - System R – IBM Research
  - INGRES – U.C. Berkeley
  - Oracle – Larry Ellison

Turing Award 1998

Gray

Turing Award 2015

Stonebraker

Ellison

1980s - Relational model

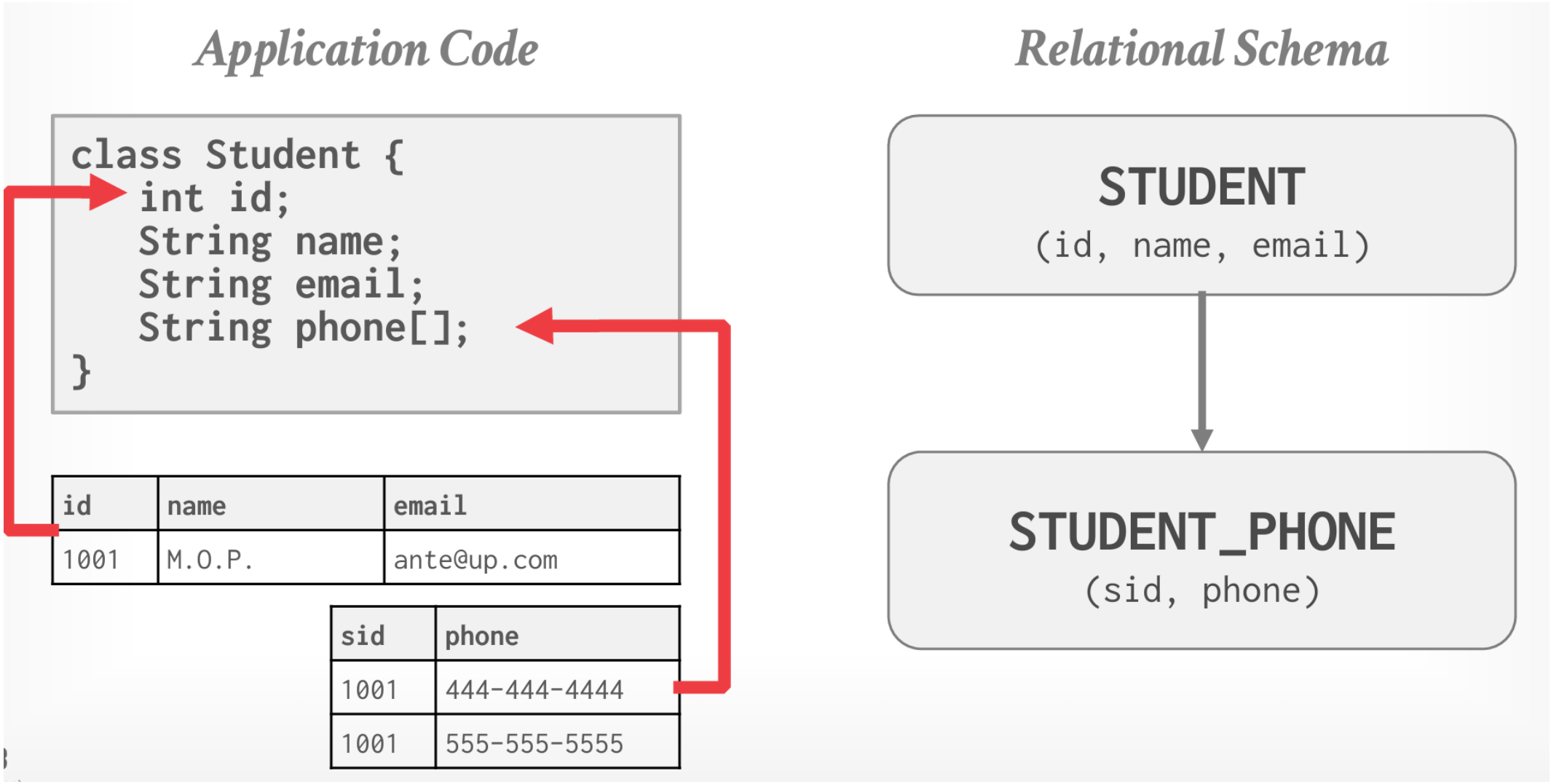
- The relational model wins.
  - IBM first releases SQL/DS in 1981.
  - IBM then turns out DB2 in 1983.
  - “SEQUEL” becomes the standard (SQL).
- Many new “enterprise” DBMSs but Oracle wins marketplace.
- Stonebraker creates Postgres as an “object-relational” DBMS

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)

Object-oriented Model



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”
   ]
}
```

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.
- Some DBMSs introduced pre-computed data cubes for faster analytics.

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 columnar data storage model.

2000s – MapReduce Systems

- Distributed programming and execution model for analyzing large data sets.
  - First proposed by Google (**MapReduce**).
  - Yahoo! created an open-source version (**Hadoop**).
  - Data model decided by user-written functions.
- People (eventually) realized this was a bad idea and added SQL on top of MR. That was a bad idea too.

MAPR-DB

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 Systems

- Provide same performance for OLTP workloads as NoSQL DBMSs without giving up ACID:
  - Relational / SQL
  - Distributed
- Almost all the first group of systems failed
- Second wave of “distributed SQL” systems are (potentially) doing better

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 - Shared-disk engines

- Instead of writing a custom storage manager, the DBMS leverages distributed storage.
  - Scale execution layer independently of storage.
  - Favors log-structured approaches.
- This is what most people think of when they talk about a data lake.

2010s - Graph systems

- Systems for storing and querying graph data.
  - Similar to the network data model (CODASYL)
- Their main advantage over other data models is to provide a graph centric query API
  - SQL:2023 is adding graph query syntax (SQL/PCG)
  - Recent research (2023) demonstrated that is unclear whether there is any benefit to using a graph-centric execution engine and storage manager.

2010s - Timeseries systems

- Specialized systems that are designed to store timeseries / event data.
- The design of these systems make deep assumptions about the distribution of data and workload query patterns.

2010s – Specialized Systems

- Embedded DBMSs
- Multi-Model DBMSs
- Hardware Acceleration
- Array / Matrix / Vector DBMSs


2010s — SPECIALIZED SYSTEMS
Embedded DBMSs
Multi-Model DBMSs
Blockchain DBMSs
Hardware Acceleration

Final Thoughts

- The demarcation lines of DBMS categories will continue to blur over time as specialized systems expand the scope of their domains.
  - Every NoSQL DBMS (except for Redis) now supports SQL

- The relational model and declarative query languages promote better data engineering.