CS 6530:
Advanced Database Systems
Fall 2024

Prashant Pandey

prashant.pandey@utah.edu

SCHOOL OF COMPUTING
UNIVERSITY OF UTAH

no smartphones

no laptop

Why?

there is enough evidence that laptops and phones slow you down

Ask questions

… and answer my questions.

Our main **goal** is to have **interesting discussions** that will help to gradually understand the material

(it's ok if not everything is clear, as long as you have questions!)

Today's agenda

- Course logistics overview
- A brief history of databases

I want you to speak up!
[and you can always interrupt me]

Why you should take this course

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

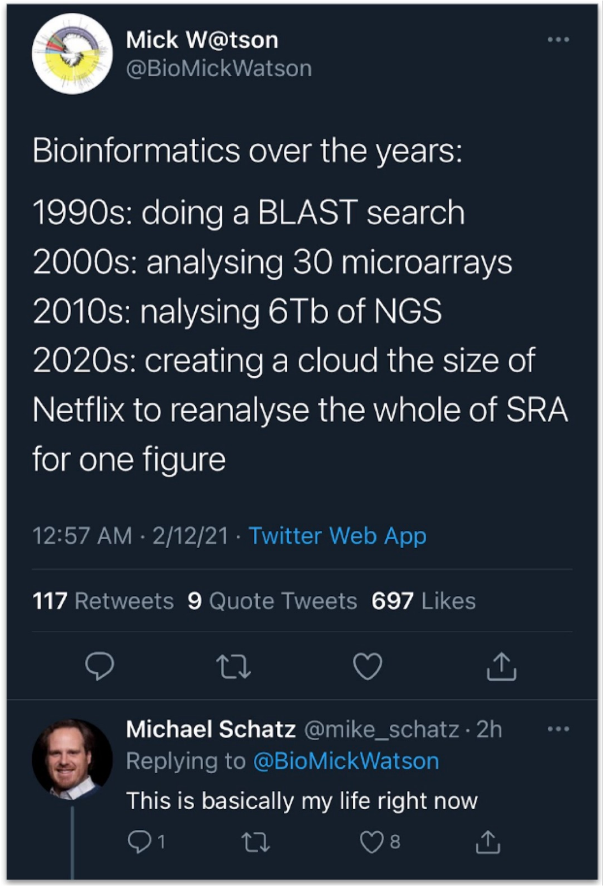
- If you are good enough to write code for a DBMS, then you can write code for almost anything else.

Data is the new oil!

But oil has to be refined and extracted to be usable

Our job is to develop refinement machinery to extract *information* from *data*!

Modern data challenges



Mick W@tson
@BioMickWatson

Bioinformatics over the years:

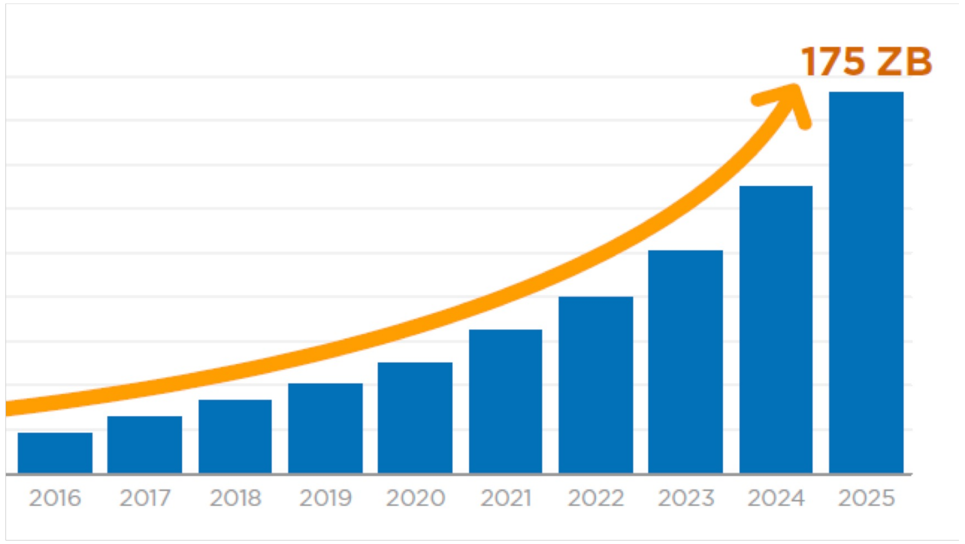
1990s: doing a BLAST search
2000s: analysing 30 microarrays
2010s: nalysing 6Tb of NGS
2020s: creating a cloud the size of Netflix to reanalyse the whole of SRA for one figure

12:57 AM · 2/12/21 · Twitter Web App

117 Retweets 9 Quote Tweets 697 Likes

Michael Schatz @mike_schatz · 2h
Replying to @BioMickWatson
This is basically my life right now

1 8



How many 0s in a Zettabyte?

Course objectives

- Learn about modern practices in database internals and systems programming.
- Next-generation challenges in data systems.

- Students will become proficient in:
  - Writing high-performance and concurrent code
  - Using tools to debug performance hot spots
  - Working on a large code base
  - Modern data system internals

Course topics

- The internals of modern single-node data systems.
- We will **<u>not</u>** discuss distributed systems.

- We will cover state-of-the-art topics in large-scale data management.
- This is **<u>not</u>** a course on classical DBMS.

Course topics

- In-memory Indexing
- Concurrency control
- Data storage and File organization
- Key-value stores
- Logging and recovery
- Query optimization
- Parallel join and external sorting
- Data systems on modern hardware
- Learned indexes and ML for Databases
- Vector Databases

Background

- I assume you have already taken undergrad Database course (e.g., CS 5530) or similar.
- You are comfortable in writing **concurrent C/C++ code**.
- We will discuss modern variations to classical data structures and algorithms that are designed for today's hardware.

- Things that we will **not** cover:

SQL, Relational Algebra, Serialization, Basic Algorithms and Data Structures

Course logistics

- Course policies + Schedule
  - Website: https://users.cs.utah.edu/~pandey/courses/cs6530/fall24/index.html
- Academic honesty
  - Refer to SoC policy on academic conduct.
  - If you are not sure, ask me.
  - **I am serious. DO NO PLAGIARISE.**

What is plagiarism

- Listening while someone dictates a solution.
- Basing your solution on any other written solution.
- Copying another student's code or <u>sharing</u> your code with any other student.
- Searching for solution online (e.g., stack overflow, Github, Github Copilot, ChatGPT).

What is collaboration

- Asking questions on Canvas discussions.
- Working <u>together</u> to find a good approach for solving a problem.
  - Students with similar understanding of the material.
- A high-level discussion of solution strategy.
- If you collaborate with other students, **<u>declare</u>** it upfront
  - Put names of the collaborator at the start of the project report.

Office hours

- Before class in my office
  - Tu/Th 9:30 AM – 10:30 AM
  - WEB 2686
- Things that we can talk about:
  - Issues on implementing projects
  - Paper clarification/discussions
  - Getting involved in a research project
  - How to get a database/systems dev job

Teaching assistant/mentor

- TA: Yuvaraj Chesetti
  - Office hours Wednesday 12Noon --- 2PM WEB 2780
  - 2nd year PhD student
  - Research on Data Management
    - Hash tables
    - Filters
    - Learned indexes

Instructor

- Previous:
  - Research Scientist, VMware Research
  - Postdoc: CMU/UC Berkeley
  - PhD: Stony Brook University
- Research:
  - Data systems
  - Storage systems & graph processing
  - Computational biology

Somewhere in Patagonia, Chile

Course rubric

- Reading assignments
- Programming projects
- Final exam
- Class participation

Reading assignments

- Pick five papers from the reading list.
- Write a one-paragraph synopsis of each of the five papers.
- There will be five deadlines throughout the semester.
- Synopsis:
  - Overview of the main idea (Three sentences).
  - Main finding/takeaway of the paper (One sentence).
  - System used and how it was modified (One sentence).
  - Workloads evaluated (One sentence).

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 SoC policy on academic conduct for additional information.

Programming projects

- Do all development on your local machine.
  - The initial code for projects builds on linux.
  - We will provide configuration/build files.
- Do all benchmarking using Cade clusters.
  - Cade setup instructions are available in Canvas.
  - We will provide further details later in semester.

Projects

- We will provide you with test cases and scripts for the first programming
  - We will teach you how to profile a system using a tool

Project #1 will be done individually.

Project #2, #3 will be done in a group of **three**.

Project #3

- We will provide a default project topic.
  - Will have multiple milestones.

- A group can also choose a project that is:
  - Relevant to the materials discussed in class.
  - Requires a significant programming effort from <u>all</u> team members.
  - Unique (i.e., two groups cannot pick same idea).
  - Approved by me.

Plagiarism warning

- These projects must be all of your own code.
- You may **<u>not</u>** copy source code from other groups or the web.
- Plagiarism will **<u>not</u>** be tolerated.
  See SoC policy on academic conduct for additional information.

Grade breakdown

- Project #1 15%
- Project #2 25%
- Project #3 30%
- Paper reports 10%
- Final exam (take home) 10%
- Class participation 10%

More logistics

- Prashant traveling on 08/22
- Lecture 08/22: Yuvaraj Chesetti
  - Introduction to project #1
  - Tutorial on tools for profiling and build system
  - Atomics and memory consistency

Course mailing list

- Online discussion through Canvas
  - Use Canvas Discussion
- If you have a technical question about the projects, please use Canvas
  - Don’t email me or TAs directly
- All non-project questions should be sent to me.

A brief history of databases

Acknowledgement: Slides taken from Prof. Andy Pavlo, CMU

History repeats itself

- Old database issues are still relevant today.

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

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

1960s – IDS

- **I**ntegrated **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.

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.

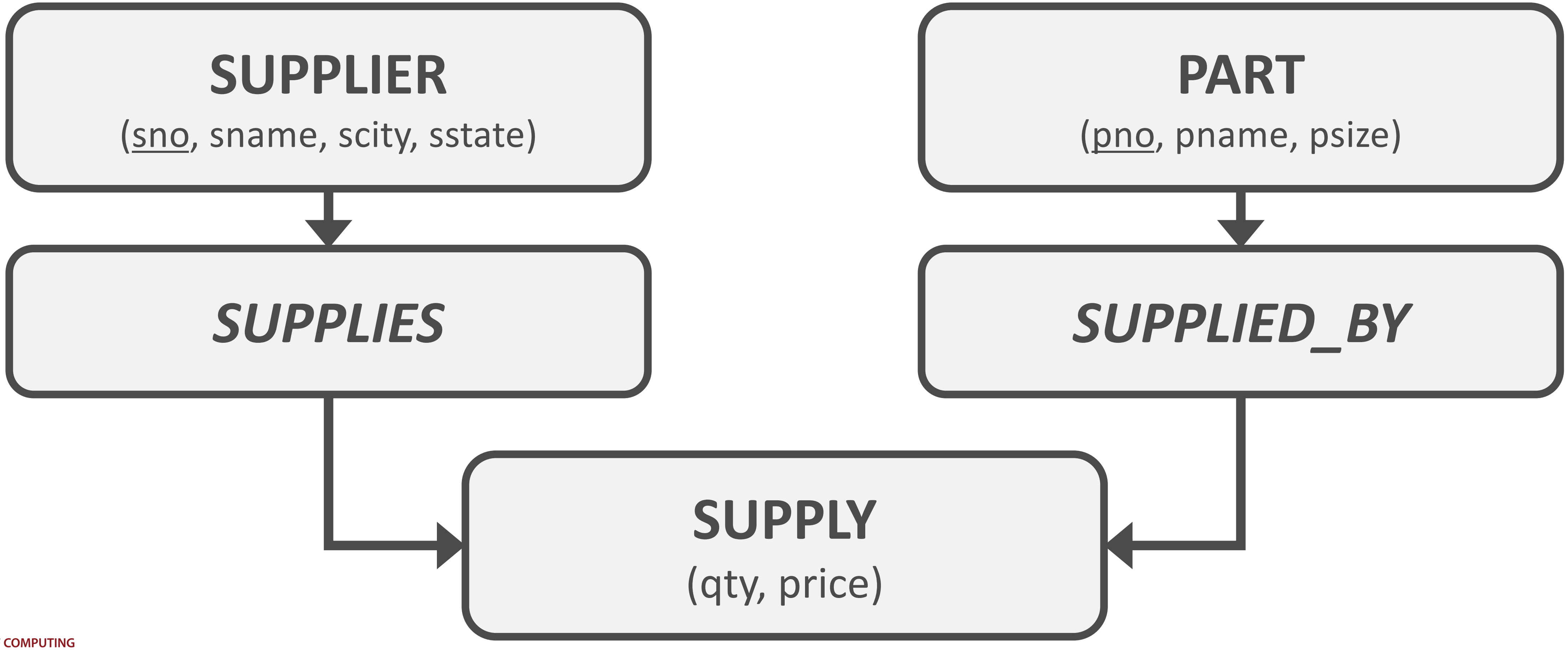
Bachman

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

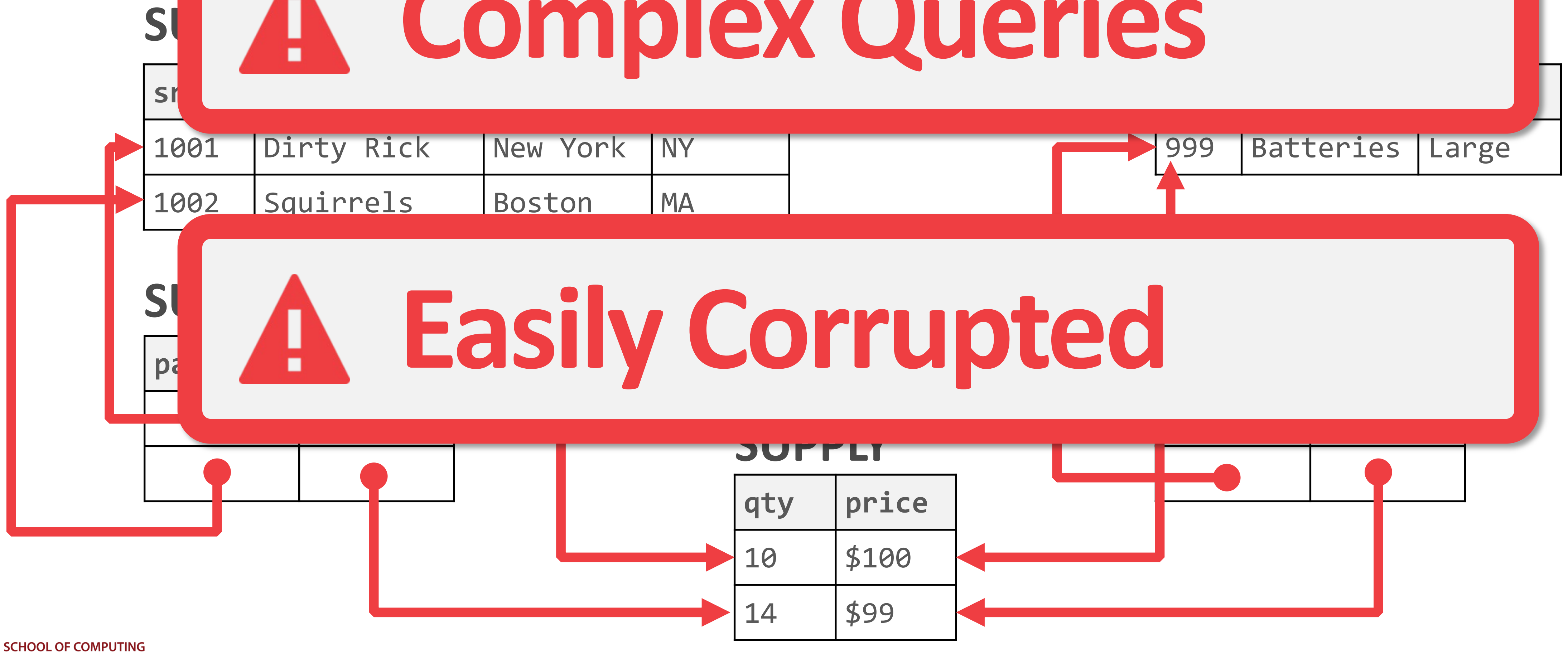
Turing award 1973

Network data model

Schema



Network data model

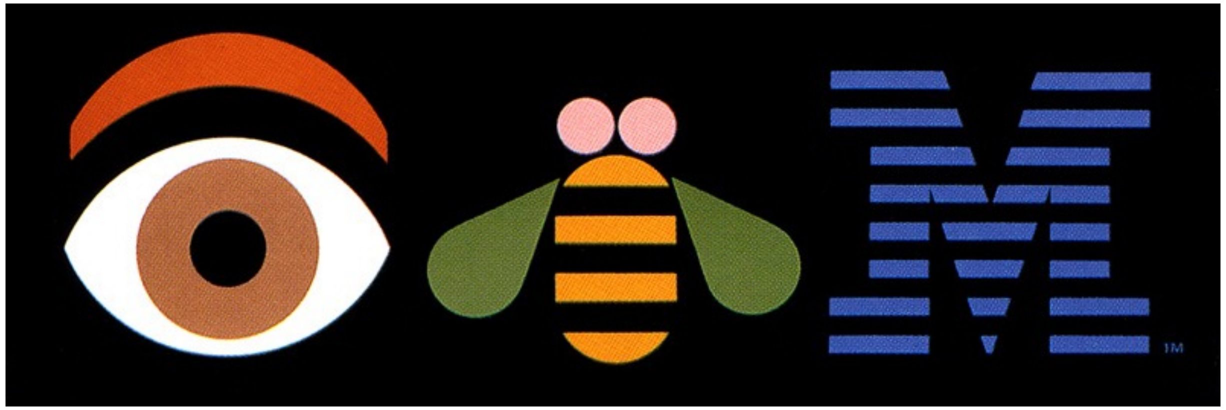


Complex Queries

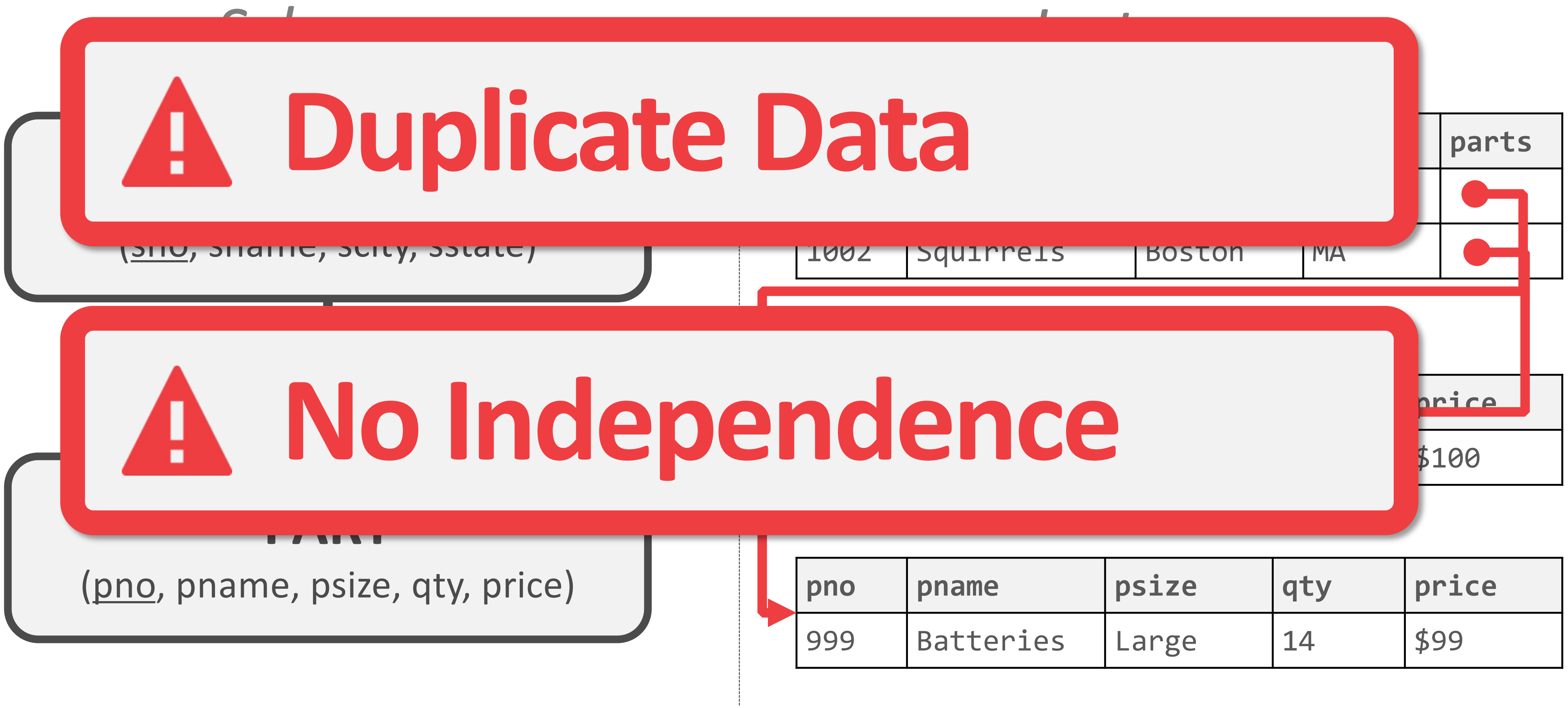
Easily Corrupted

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



⚠ Duplicate Data

⚠ No Independence

pno	pname	psize	qty	price
999	Batteries	Large	14	$99

MOD

natician

w devel

riting II

time th

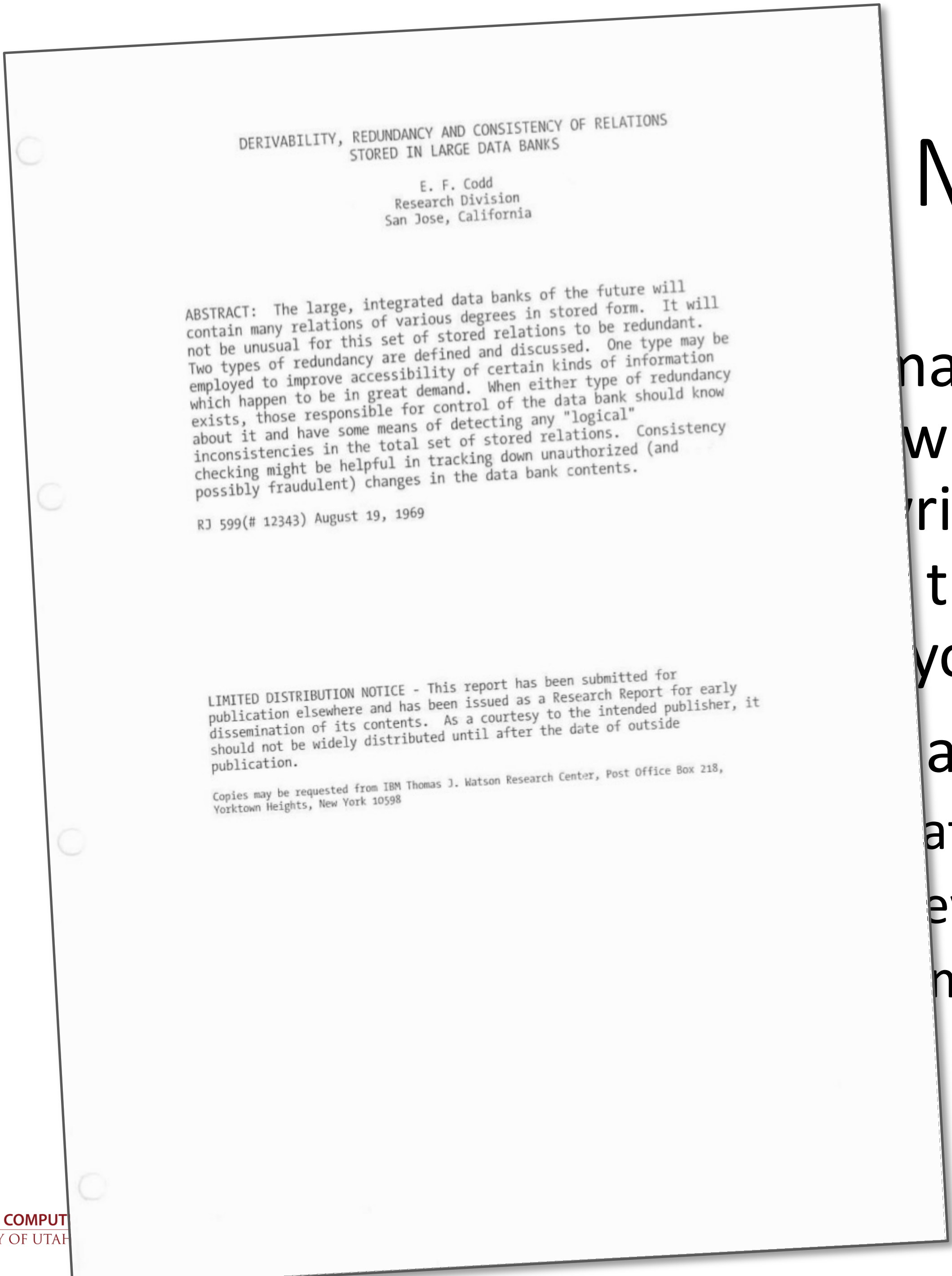
yout ch

avoid

ata stru

evel lan

mplem



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

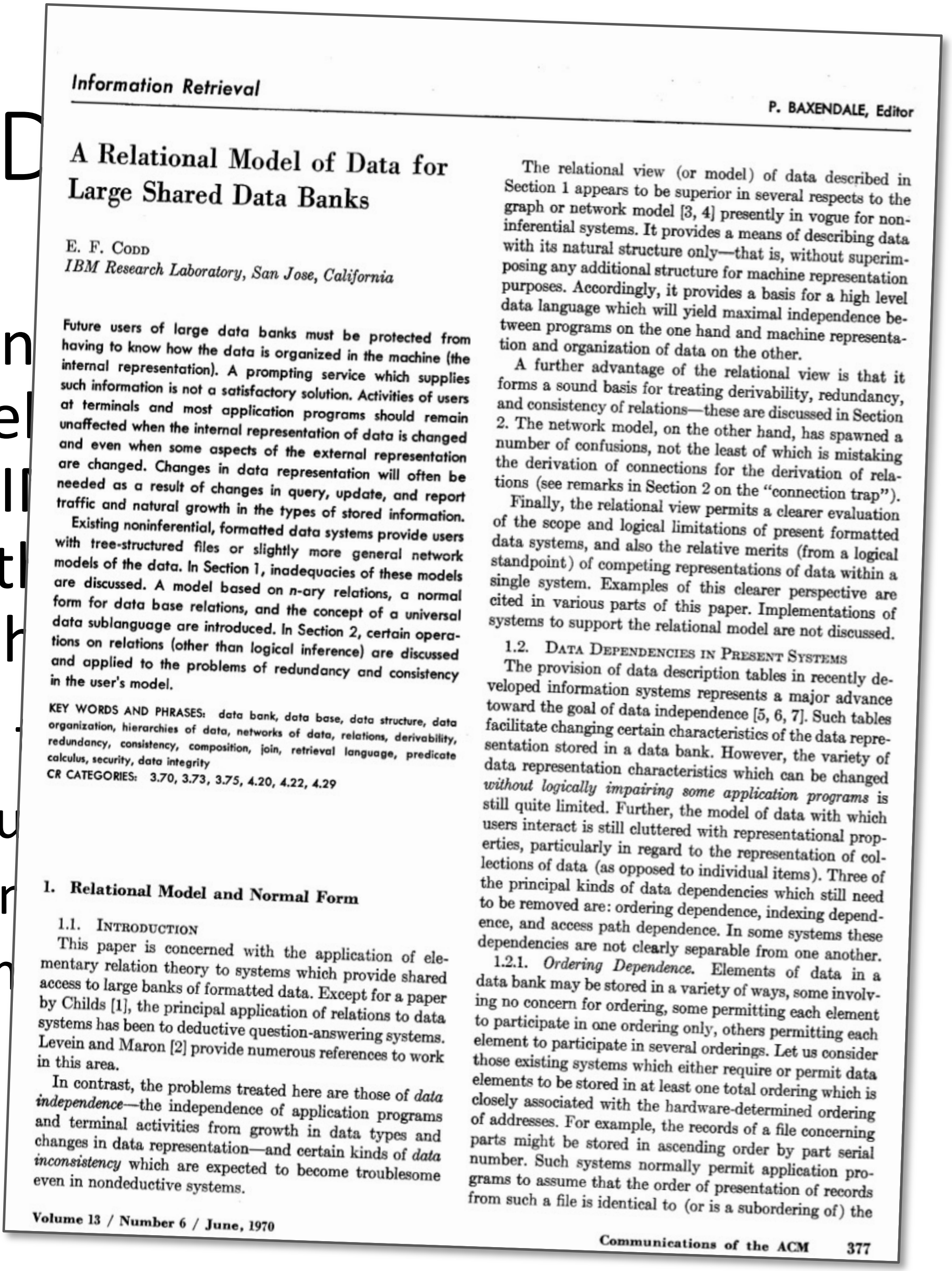
E. F. Codd
Research Division
San Jose, California

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

LIMITED DISTRIBUTION NOTICE - This report has been submitted for publication elsewhere and has been issued as a Research Report for early dissemination of its contents. As a courtesy to the intended publisher, it should not be widely distributed until after the date of outside publication.

Copies may be requested from IBM Thomas J. Watson Research Center, Post Office Box 218, Yorktown Heights, New York 10598



Information Retrieval

P. BAXENDALE, Editor

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

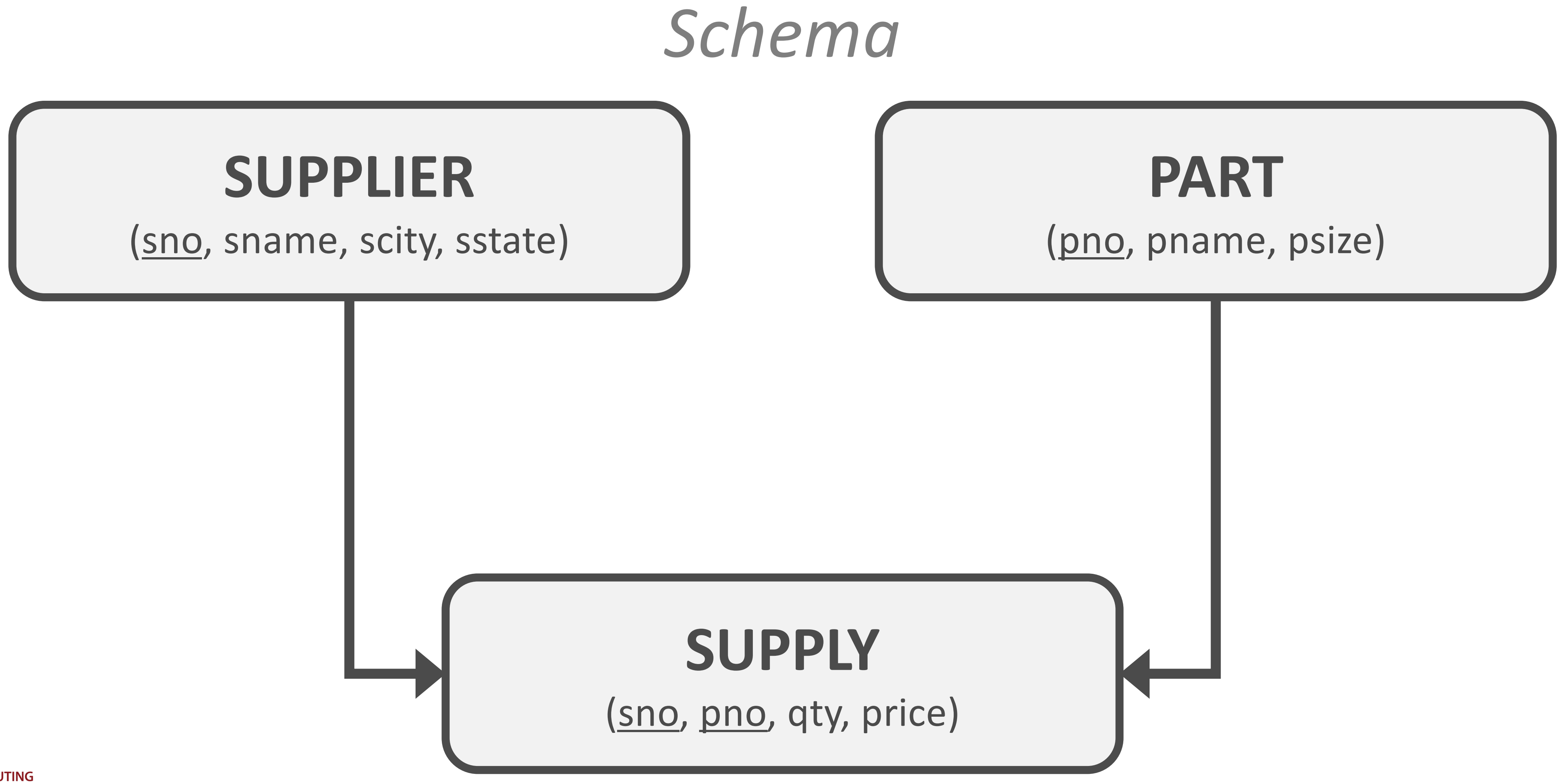
The provision of data description tables in recently developed information systems represents a major advance toward the goal of data independence [5, 6, 7]. Such tables facilitate changing certain characteristics of the data representation stored in a data bank. However, the variety of data representation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

Volume 13 / Number 6 / June, 1970

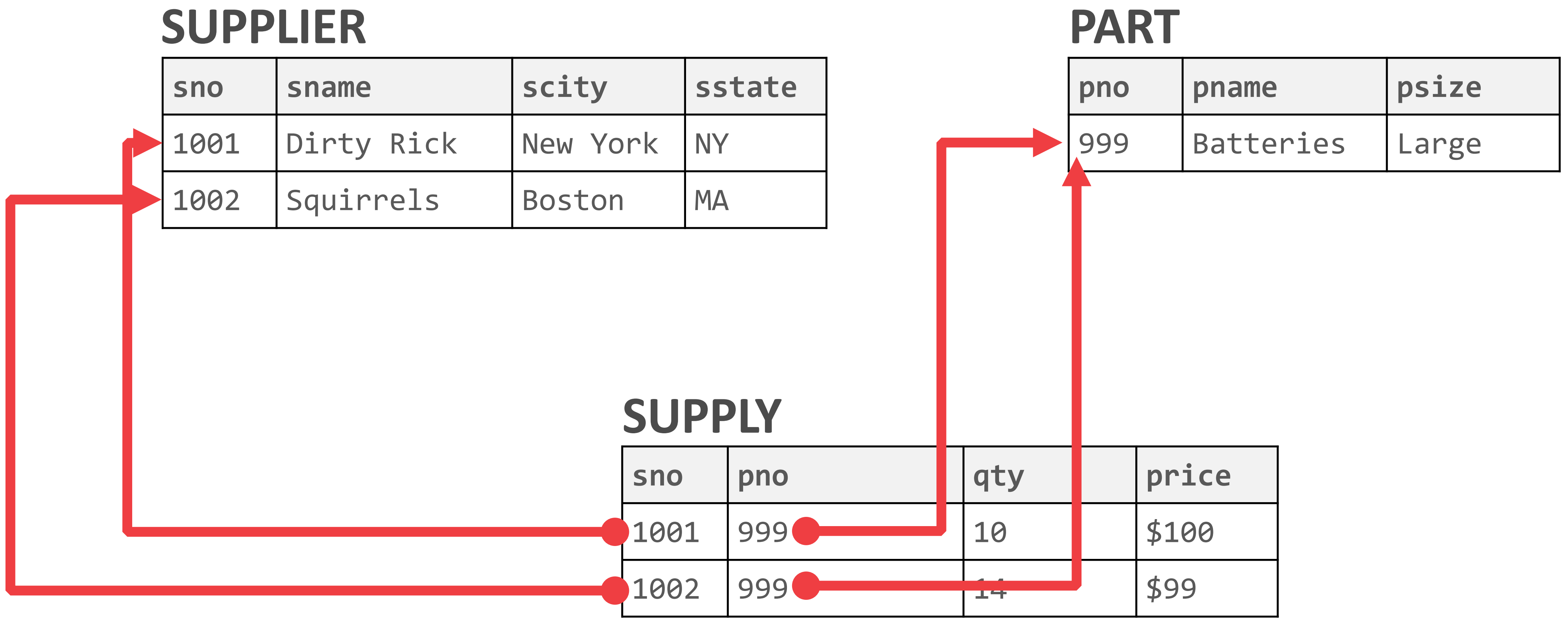
Communications of the ACM 377

Relational data model



Relational data model

Instance



SUPPLIER

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

PART

pno	pname	psize
999	Batteries	Large

SUPPLY

sno	pno	qty	price
1001	999	10	$100
1002	999	14	$99

SCHOOL OF COMPUTING
UNIVERSITY OF UTAH

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

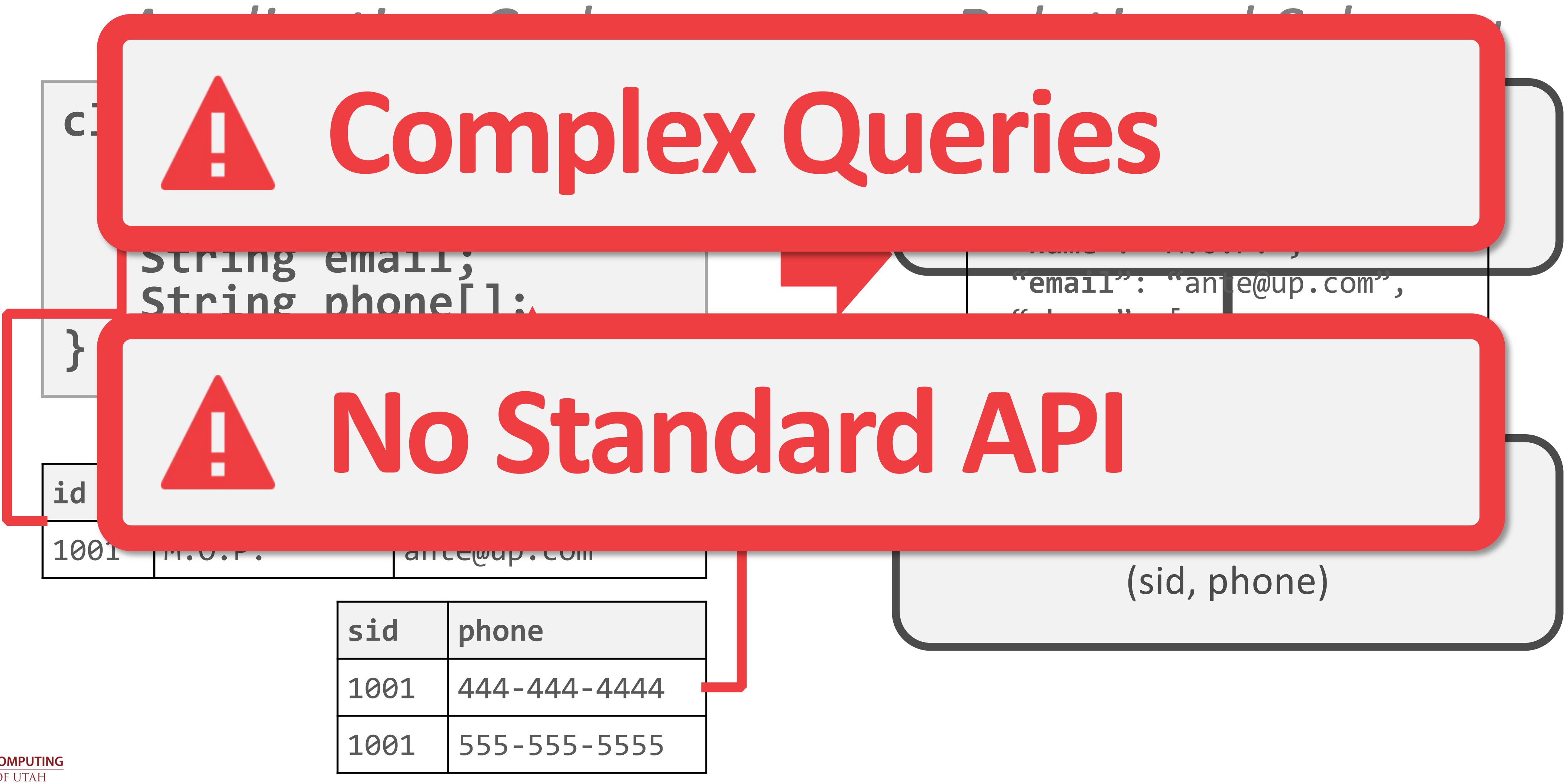
- The relational model wins.
  - IBM comes out with DB2 in 1983.
  - “SEQUEL” becomes the standard (SQL).
- Many new “enterprise” DBMSs but Oracle wins marketplace.
- Stonebraker creates Postgres.

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



⚠ Complex Queries

⚠ No Standard API

sid	phone
1001	444-444-4444
1001	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.

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 – 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

APACHE HBASE amazon DynamoDB mongoDB NOSQL redis RethinkDB Couchbase neo4j CouchDB relax cassandra riak AEROSPIKE RAVENDB

2010s – NewSQL

- Provide same performance for OLTP workloads as NoSQL DBMSs without giving up ACID:
  - Relational / SQL
  - Distributed
  - Usually closed-source

Cockroach LABS

dbShards H-Store Clustrix Comdb2

ScaleArc SAP HANA VOLTDB NUODB TiDB

Google Spanner MEMSQL HyPer YugaByte

2010s – Hybrid systems

- **H**ybrid **T**ransactional-**A**nalytical **P**rocessing.
- 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.

MEMSQL HyPer SNAPPY SAP HANA

JustOne DB splice MACHINE Peloton

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 – Stream processing

- Execute continuous queries on streams of tuples.
- Extend processing semantics to include notion of windows.

- Often used in combination of batch-oriented systems in a **lambda architecture** deployment.

kafka samza PULSAR

Flink APACHE STORM™ HERON

2010s – Graph systems

- Systems for storing and querying graph data.
- Their main advantage over other data models is to provide a graph-centric query API
  - Recent research 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.

M3 TIMESCALE influxdb

VICTORIA METRICS ClickHouse

2010s – SPECIALIZED SYSTEMS

- Embedded DBMSs
- Multi-Model DBMSs
- Blockchain DBMSs
- Hardware Acceleration