
AWS
re:Invent
NOV. 29 – DEC. 3, 2021 | LAS VEGAS, NV

DAT207-S

SPONSORED BY MONGODB

Is relational the new COBOL?

Mark Porter
Chief Technology Officer
MongoDB

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Today's Journey Together

Intro and thesis

The first golden age of data

Amdahl's bloodless coup

The perfect storm

Pay me now or pay me later?

So, is relational the new COBOL or not?

Today's Journey Together

Intro and thesis

The first golden age of data

Amdahl's bloodless coup

The perfect storm

Pay me now or pay me later?

So, is relational the new COBOL or not?

43 years of nerd, data, systems background



- Developer
- Designer
- Executive
- DBA
- Founder
- General Manager
- Consumer CTO
- Board Advisor
- Board Member
- **Chief Technology Officer of MongoDB**

’84-’88	Department of Education, Caltech, NASA	USE
’88-’01	ORACLE, Oracle Media Server	BUILD
’11-’13	News Corp, Bill & Melinda Gates foundation	USE
‘13-’18	aws, AWS Database Migration Service (AWS DMS)	BUILD
’18-’20	Grab, GrabFood, GrabCar, GrabExpress, MariaDB	USE
’19-now	MongoDB.	BUILD

An example of one technology obsoleting another



New York 1900



New York 1911

Technology marches on relentlessly

Cars replaced horses

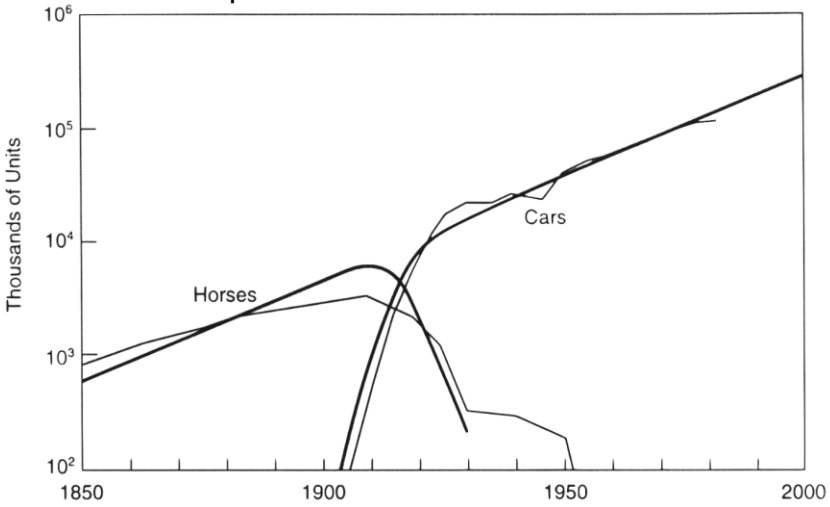
Internal combustion replaced steam

Steam replaced water wheels

Water wheels replaced oxen

And it can happen fast

US Replacement of Horses with Cars



After Nebojsa Nakicenovic (1997).
Technological Change As A Learning Process.
Induced Technology Workshop, IIASA.
26–27 June 1997.

But enough about horses . . .



QUESTIONS

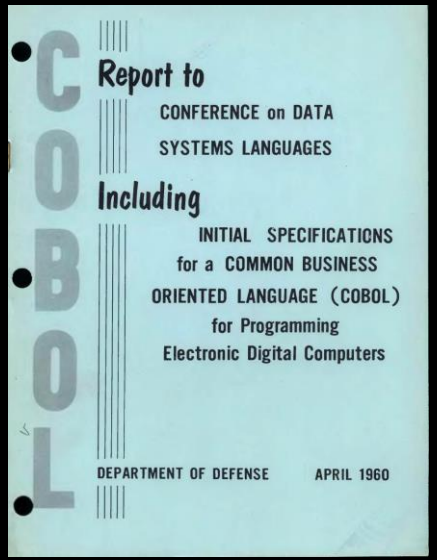
What causes technology changeovers?

Why do some technologies hang around?

Can we predict these two things?

WHAT IS COBOL?

COmmon Business Oriented Language

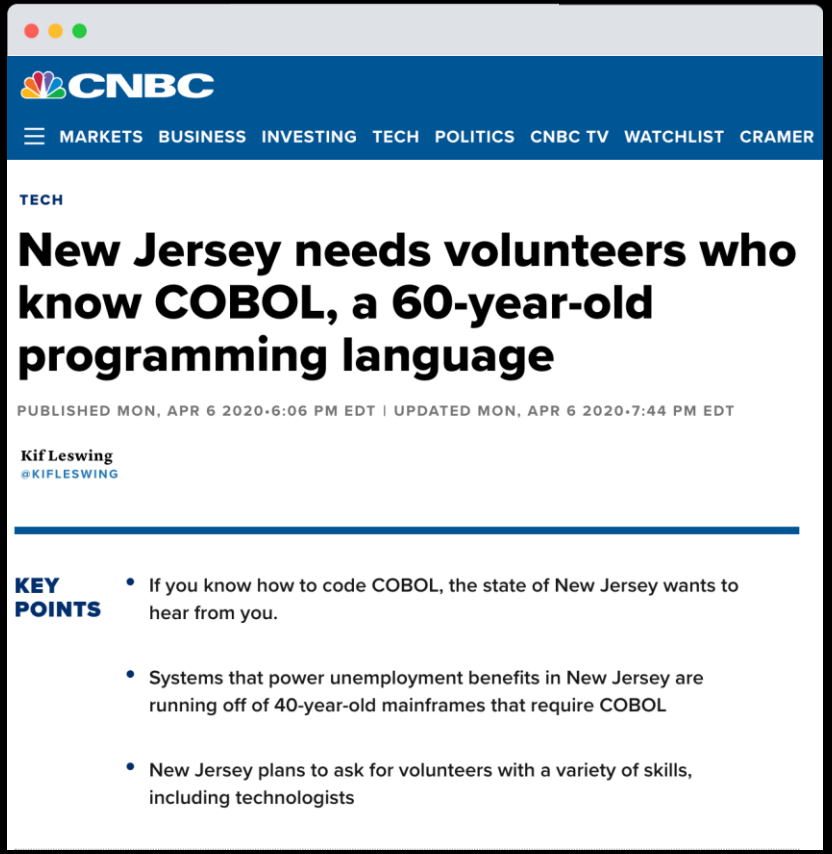


1959 – DOD Sponsored the creation of a portable business programming language

DOD forced COBOL's use leading to widespread adoption

COBOL is *still used widely* in applications deployed on mainframe computers, especially for transaction processing

Urgent announcement



CNBC

MARKETS BUSINESS INVESTING TECH POLITICS CNBC TV WATCHLIST CRAMER

TECH

New Jersey needs volunteers who know COBOL, a 60-year-old programming language

PUBLISHED MON, APR 6 2020•6:06 PM EDT | UPDATED MON, APR 6 2020•7:44 PM EDT

Kif Leswing
@KIFLESWING

KEY POINTS

- If you know how to code COBOL, the state of New Jersey wants to hear from you.
- Systems that power unemployment benefits in New Jersey are running off of 40-year-old mainframes that require COBOL
- New Jersey plans to ask for volunteers with a variety of skills, including technologists

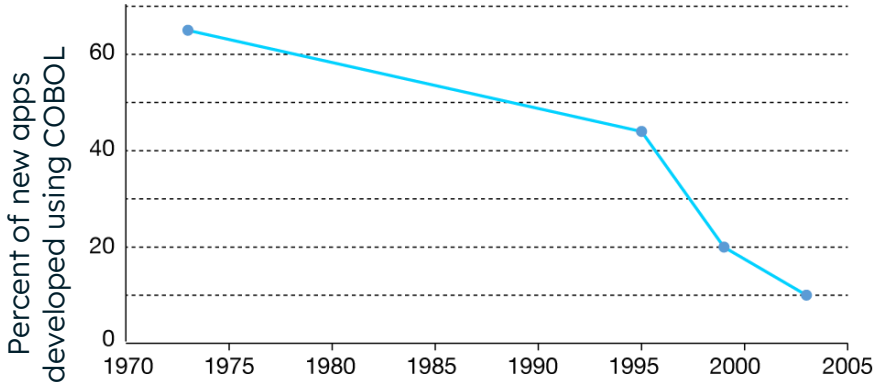
DEMAND

Still widely used



220B lines of COBOL code
1–2M active COBOL programmers

But not for new development



80–100B lines of new code (in all languages)/year
23–28M active developers

CLUMSY CODE

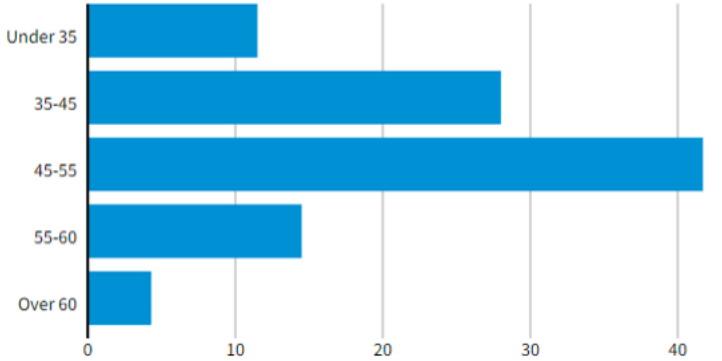
Hello World!

```
//COBUCLG  JOB (001),'COBOL SERVERLESS TEST',
//           CLASS=A,MSGCLASS=A,MSGLEVEL=(1,1)
//BASETEST EXEC COBUCLG
//COB.SYSIN DD *
 00000* VALIDATION OF BASE COBOL INSTALL
 01000 IDENTIFICATION DIVISION.
 01100 PROGRAM-ID. 'HELLO'.
 02000 ENVIRONMENT DIVISION.
 02100 CONFIGURATION SECTION.
 02110 SOURCE-COMPUTER.  GNULINUX.
 02120 OBJECT-COMPUTER.  HERCULES.
 02200 SPECIAL-NAMES.
 02210     CONSOLE IS CONSL.
 03000 DATA DIVISION.
 04000 PROCEDURE DIVISION.
 04100 00-MAIN.
 04110     DISPLAY 'HELLO, WORLD' UPON CONSL.
 04900     STOP RUN.
//LKED.SYSLIB DD DSNAME=SYS1.COBLIB,DISP=SHR
//           DD DSNAME=SYS1.LINKLIB,DISP=SHR
//GO.SYSPRINT DD SYSOUT=A
```

New graduates have moved on

AVERAGE AGE OF DEVELOPERS

On average, COBOL programmers are most likely to be between 45–55 years old





DISCLAIMER

I kind of like COBOL.
I have loved relational.
I've used both.



OBSERVATION 1

COBOL
is a technology that has lived for many years – but we've moved past it being the most productive way to develop and deploy apps.



THESIS

Relational databases are a technology that will live for many years – but we've moved past them being the most productive way to develop and deploy apps.

Today's Journey Together

Intro and thesis

The first golden age of data

Amdahl's bloodless coup

The perfect storm

Pay me now or pay me later?

So, is relational the new COBOL or not?

Today's Journey Together

Intro and thesis

The first golden age of data

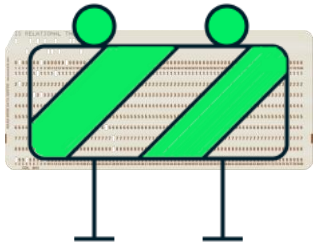
Amdahl's bloodless coup

The perfect storm

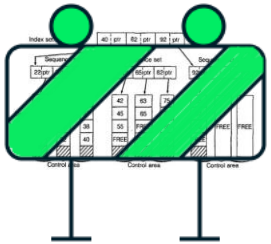
Pay me now or pay me later?

So, is relational the new COBOL or not?

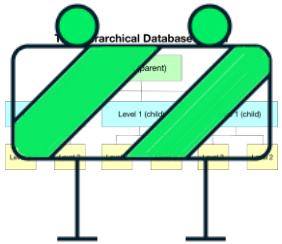
Relational wasn't even close to the beginning . . .



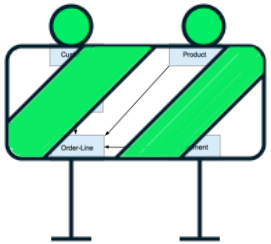
Hollerith cards



ISAM/ VSAM



Hierarchical models



Network models



Frustration

Relational technology was revolutionary (in 1970)



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 r ons (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

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

Volume 13 / Number 6 / June, 1970

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 incompleteness which are expected to become troublesome even in nondeductive systems.

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

stored ordering. Those application programs which take advantage of the stored ordering of a file are likely to fail to operate correctly if for some reason it becomes necessary to replace that ordering by a different one. Similar remarks hold for a stored ordering implemented by means of pointers.

It is unnecessary to single out any system as an example, because all the well-known information systems that are marketed today fail to make a clear distinction between order of presentation on the one hand and stored ordering on the other. Significant implementation problems must be solved to provide this kind of independence.

1.2.2. *Indexing Dependence.* In the context of formatted data, an index is usually thought of as a purely performance-oriented component of the data representation. It tends to improve response to queries and updates and, at the same time, slow down response to insertions and deletions. From an informational standpoint, an index is a redundant component of the data representation. If a system uses indices at all and if it is to perform well in an environment with changing patterns of activity on the data bank, an ability to create and destroy indices from time to time will probably be necessary. The question then arises: Can application programs and terminal activities remain invariant as indices come and go?

Present formatted data systems take widely different approaches to indexing. TDMS [7] unconditionally provides indexing on all attributes. The presently released version of IDS [5] provides the user with a choice for each file: a choice between no indexing at all (the hierarchic sequential organization) or indexing on the primary key only (the hierarchic indexed sequential organization). In neither case is the user's application logic dependent on the existence of the unconditionally provided indices. IDS [6], however, permits the file designers to select attributes to be indexed and to incorporate indices into the file structure by means of additional chains. Application programs taking advantage of the performance benefit of these indexing chains must refer to those chains by name. Such programs do not operate correctly if these chains are later removed.

1.2.3. *Access Path Dependence.* Many of the existing formatted data systems provide users with tree-structured files or slightly more general network models of the data. Application programs developed to work with these systems tend to be logically impaired if the trees or networks are changed in structure. A simple example follows.

Suppose the data bank contains information about parts and projects. For each part, the part number, part name, part description, quantity-on-hand, and quantity-on-order are recorded. For each project, the project number, project name, project description are recorded. Whenever a project makes use of a certain part, the quantity of that part committed to the given project is also recorded. Suppose that the system requires the user or file designer to declare or define the data in terms of tree structures. Then, any one of the hierarchical structures may be adopted for the information mentioned above (see Structures 1–5).

Structure 1. Projects Subordinate to Parts

File	Segment	Fields
F	PART	part # part name part description quantity-on-hand quantity-on-order
	PROJECT	project # project name project description quantity committed

Structure 2. Parts Subordinate to Projects

File	Segment	Fields
F	PROJECT	project # project name project description
	PART	part # part name part description quantity-on-hand quantity-on-order quantity committed

Structure 3. Parts and Projects as Peers Connected Relationship Subordinate to Projects

File	Segment	Fields
F	PART	part # part name part description quantity-on-hand quantity-on-order
G	PROJECT	project # project name project description
	PART	part # quantity committed

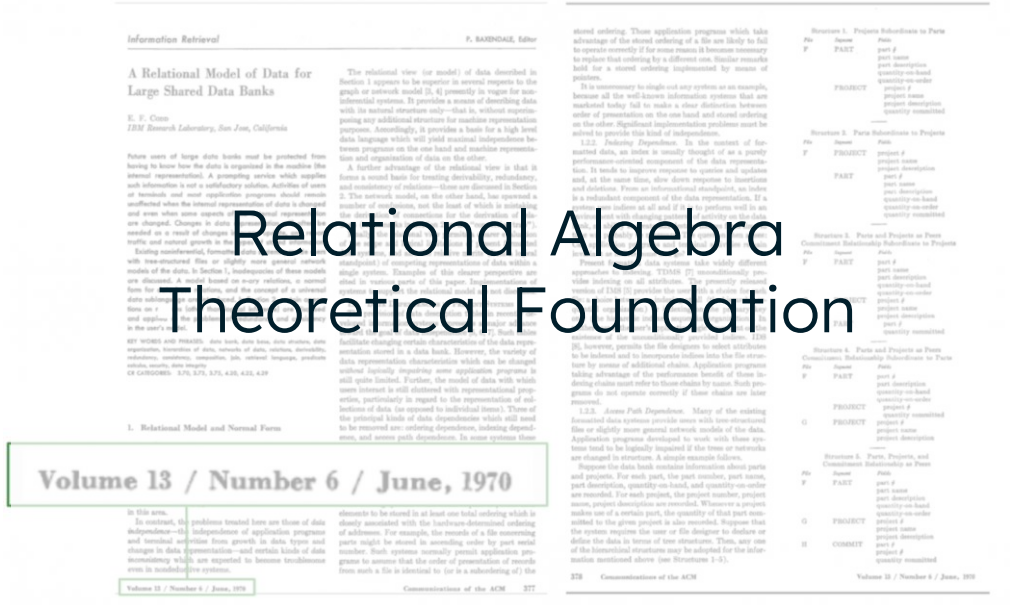
Structure 4. Parts and Projects as Peers Connected Relationship Subordinate to Parts

File	Segment	Fields
F	PART	part # part description quantity-on-hand quantity-on-order
	PROJECT	project # quantity committed
G	PROJECT	project # project name project description

Structure 5. Parts, Projects, and Commitment Relationship as Peers

File	Segment	Fields
F	PART	part # part name part description quantity-on-hand quantity-on-order
G	PROJECT	project # project name project description
H	COMMIT	part # project # quantity committed

Relational technology was revolutionary (in 1970)



Information Retrieval — P. BAXENDALE, Editor

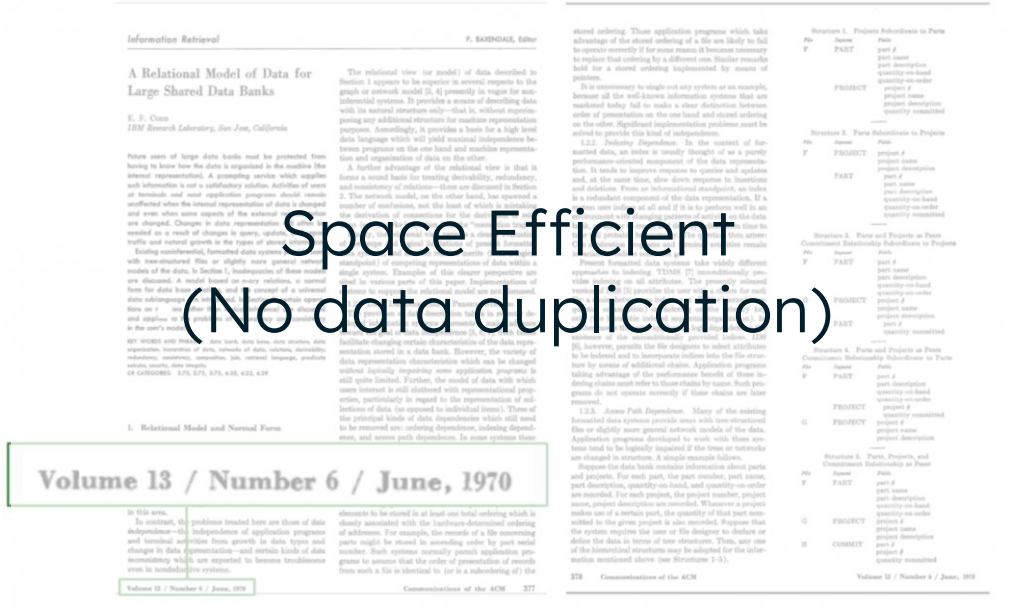
A Relational Model of Data for Large Shared Data Banks

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

Volume 13 / Number 6 / June, 1970

Relational Algebra
Theoretical Foundation

Relational technology was revolutionary (in 1970)



Information Retrieval — P. BAXENDALE, Editor

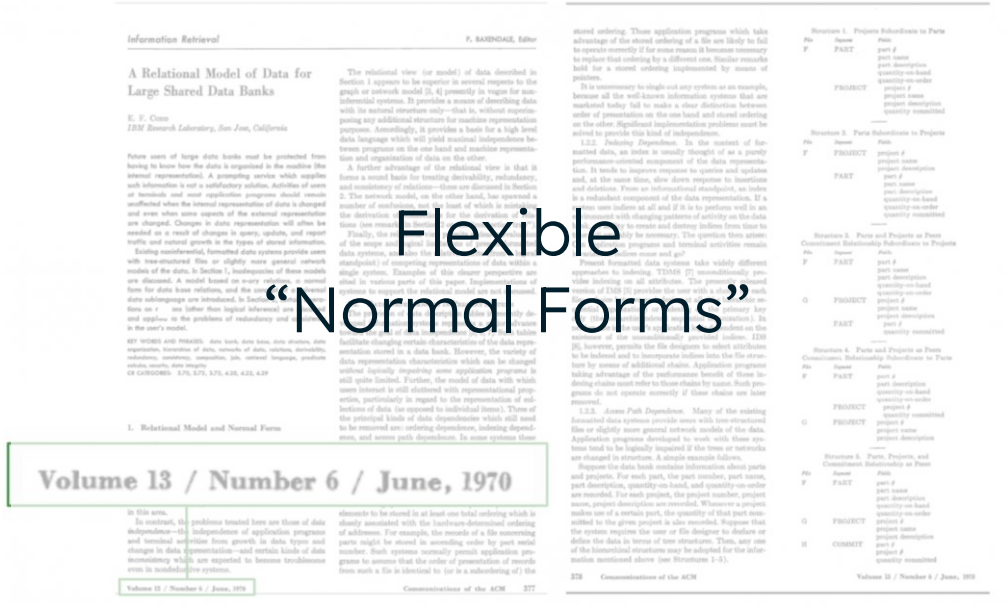
A Relational Model of Data for Large Shared Data Banks

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

Volume 13 / Number 6 / June, 1970

Space Efficient
(No data duplication)

Relational technology was revolutionary (in 1970)



Information Retrieval — P. BAXENDALE, Editor

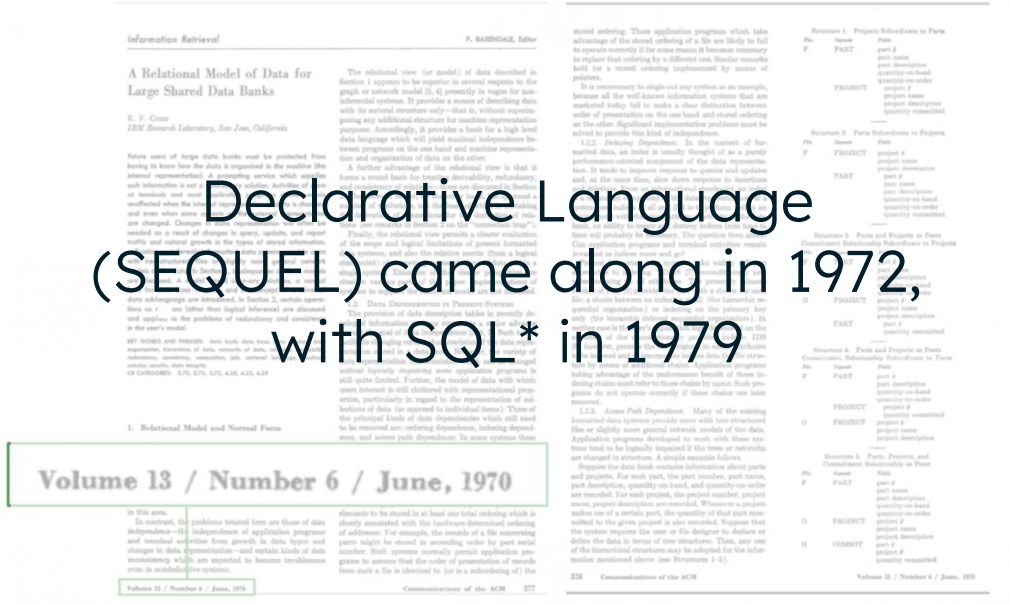
A Relational Model of Data for Large Shared Data Banks

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

Volume 13 / Number 6 / June, 1970

Flexible
“Normal Forms”

Relational technology was revolutionary (in 1970)



Information Retrieval

P. BAXENDALE, Editor

A Relational Model of Data for Large Shared Data Banks

E. F. Codd

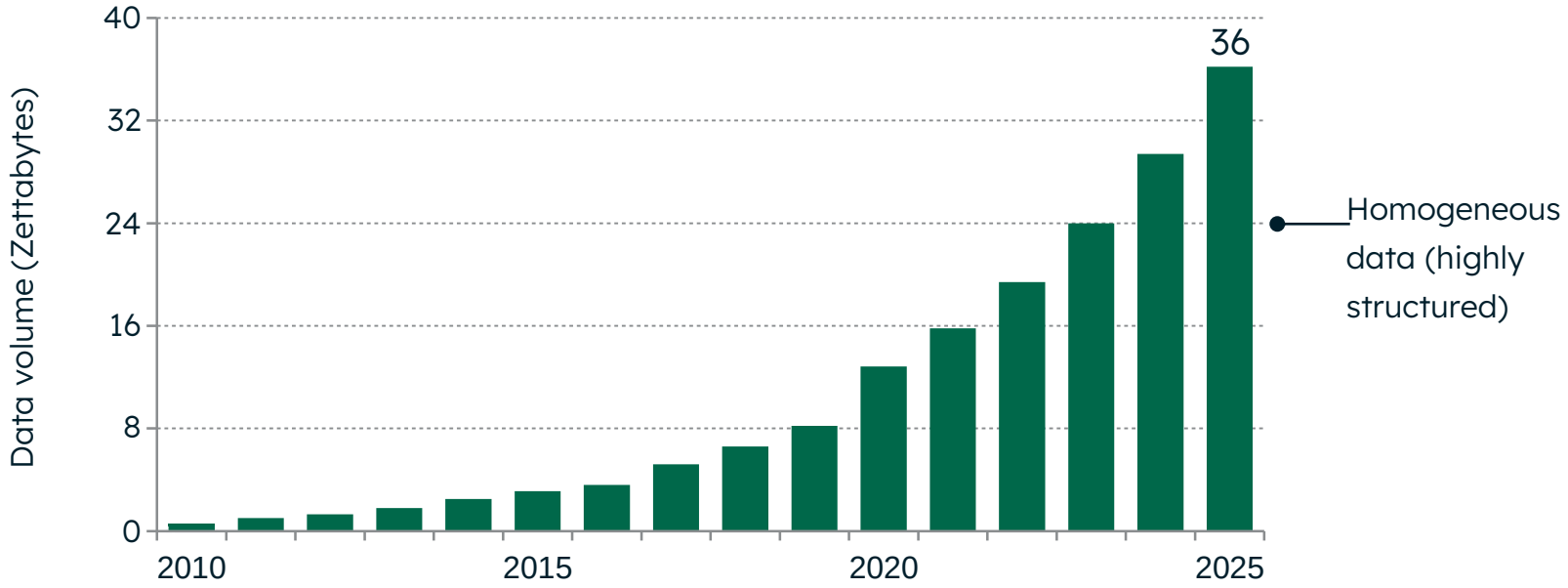
IBM Research Laboratory, San Jose, California

Volume 13 / Number 6 / June, 1970

Declarative Language (SEQUEL) came along in 1972, with SQL* in 1979

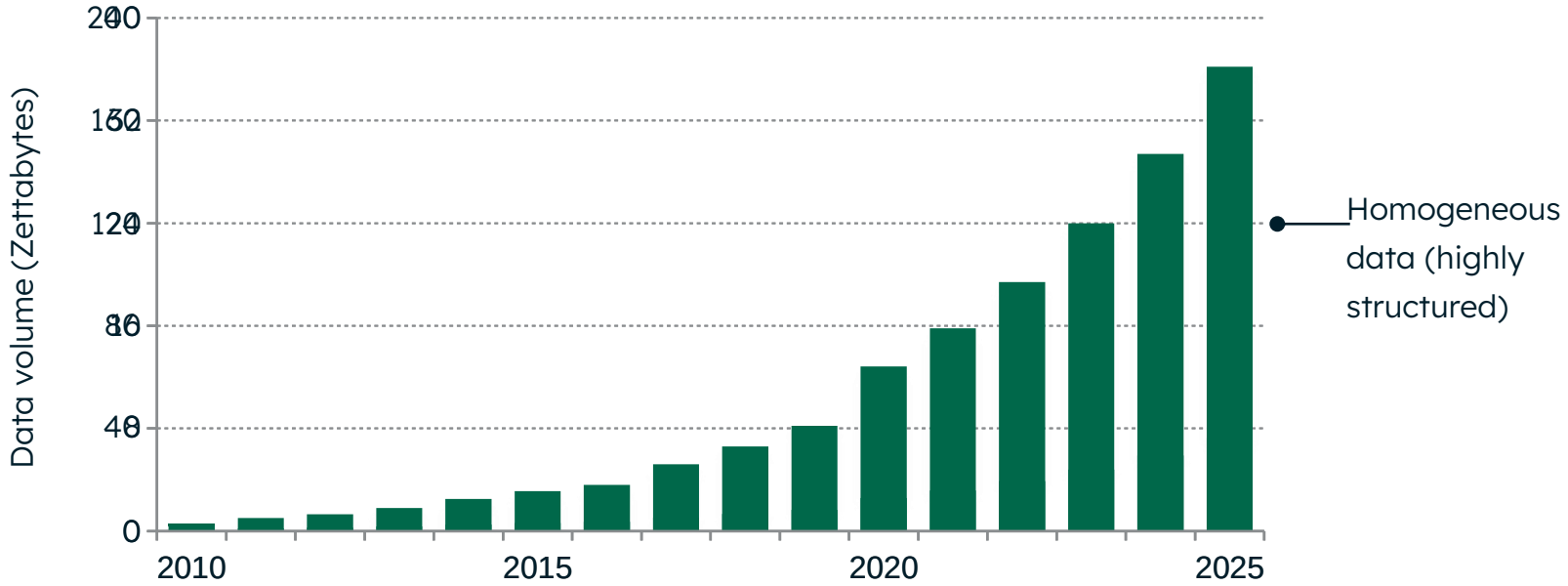
Now we had our first way to store data in a universal way

Structured data has been exploding for 50 years . . .



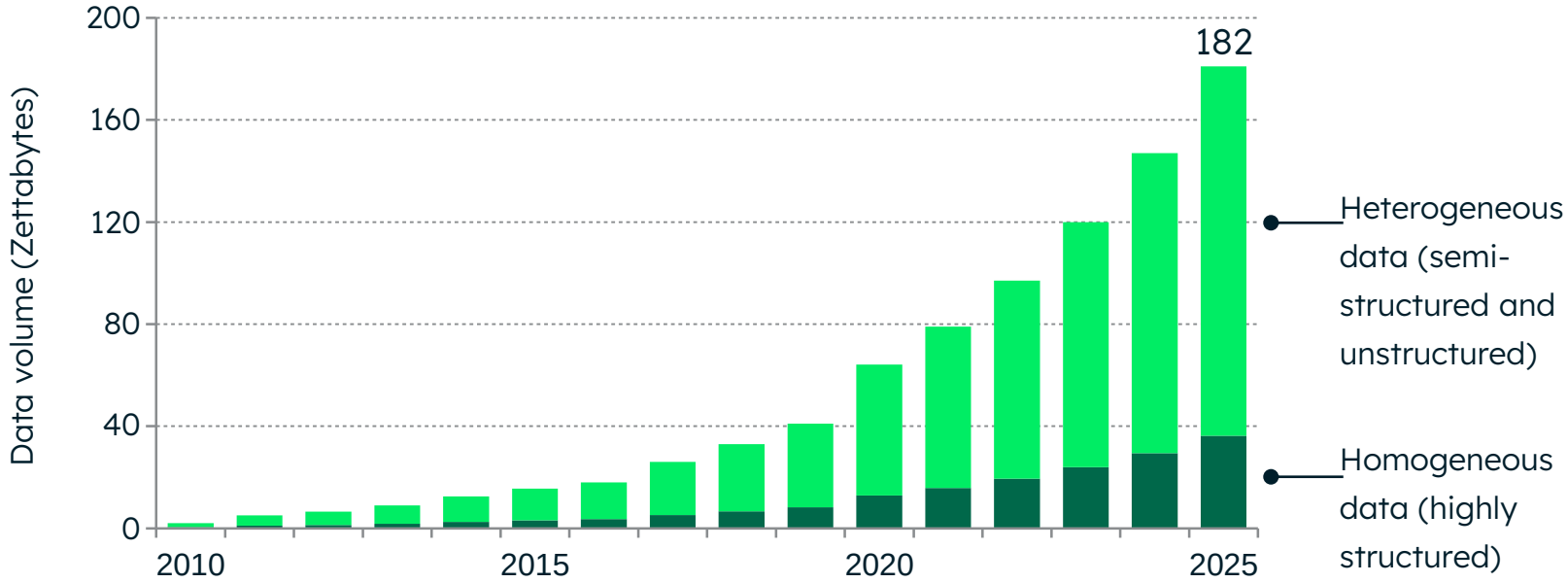
Source of data: IDC, Seagate, Statista, bit.ly/3cBztoA

Not just more data – More *kinds* of data



Source of data: IDC, Seagate, Statista, bit.ly/3cBztoA

Not just more data – More *kinds* of data



Source of data: IDC, Seagate, Statista, bit.ly/3cBztoA

Today's Journey Together

Intro and thesis

The first golden age of data

Amdahl's bloodless coup

The perfect storm

Pay me now or pay me later?

So, is relational the new COBOL or not?

Today's Journey Together

Intro and thesis

The first golden age of data

Amdahl's bloodless coup

The perfect storm

Pay me now or pay me later?

So, is relational the new COBOL or not?

In the beginning, there was Moore's Law

> "The number of transistors in a dense integrated circuit (IC) doubles about every two years.

1965



GORDON MOORE
Co-Founder of Intel

It sure looks like a “law” . . .

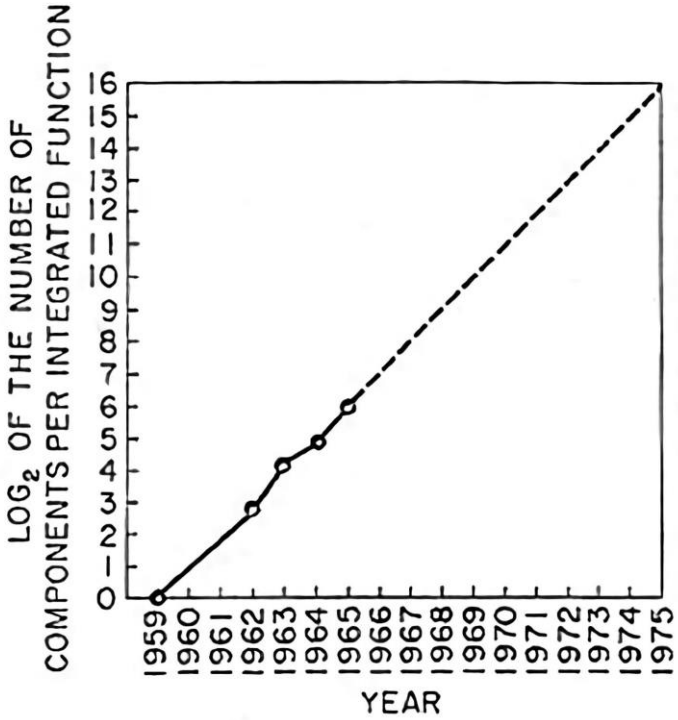
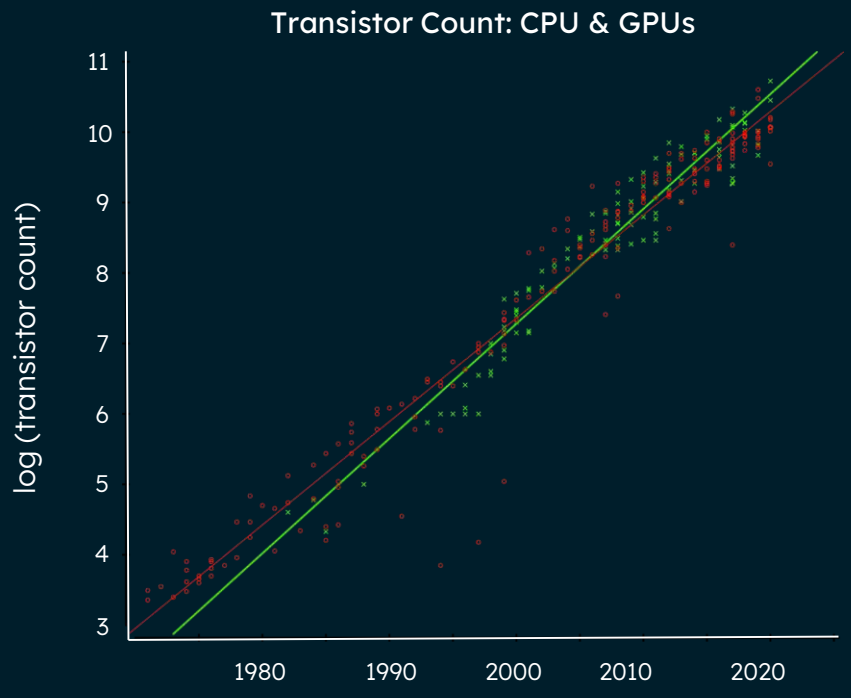


Fig. 2 Number of components per integrated function for minimum cost per component extrapolated vs time.

56 years later, Moore's Law is still going strong



Source of Data: Wikipedia, "Transistor Count"; Wikichip.org

Counting transistors is harder than it sounds

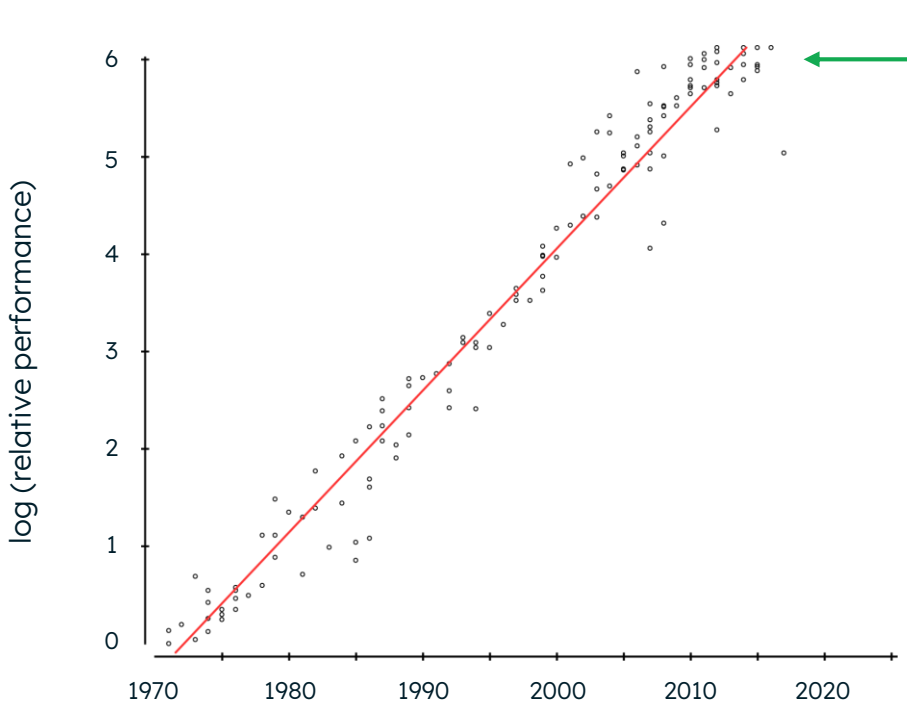
Counting ignores architecture and design (RISC, CISC, etc.)

Transistor density varies: L3 cache density 3x to 5x times higher than the logic density

Overall transistor density can vary by 20x! (recall Moore's Law is 2x)

The purpose of a chip drives how much of each kind of region is used

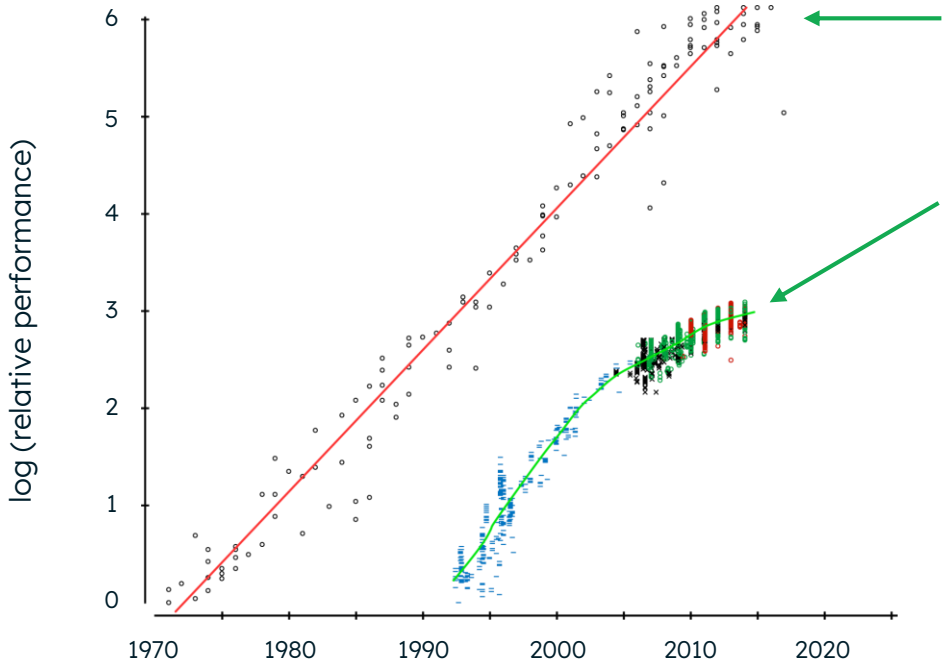
It’s the party that never ends! Or is it?



Source of Data: Wikipedia, cpudb.stanford.edu

1 Moore’s Law unabated (transistors per chip) regression trend line

Performance growth started to slow down (2004)

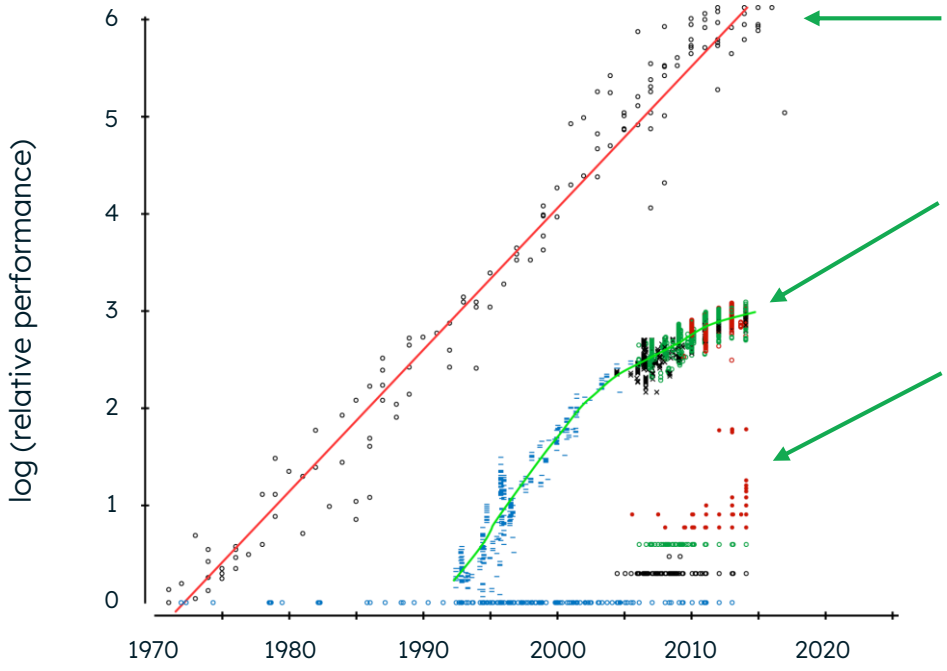


Source of Data: Wikipedia, cpudb.stanford.edu

1. Moore's Law unabated (transistors per chip) regression trend line

2. Performance improvement is slowing (SPECIntRate, normalized system throughput) Loess trend line

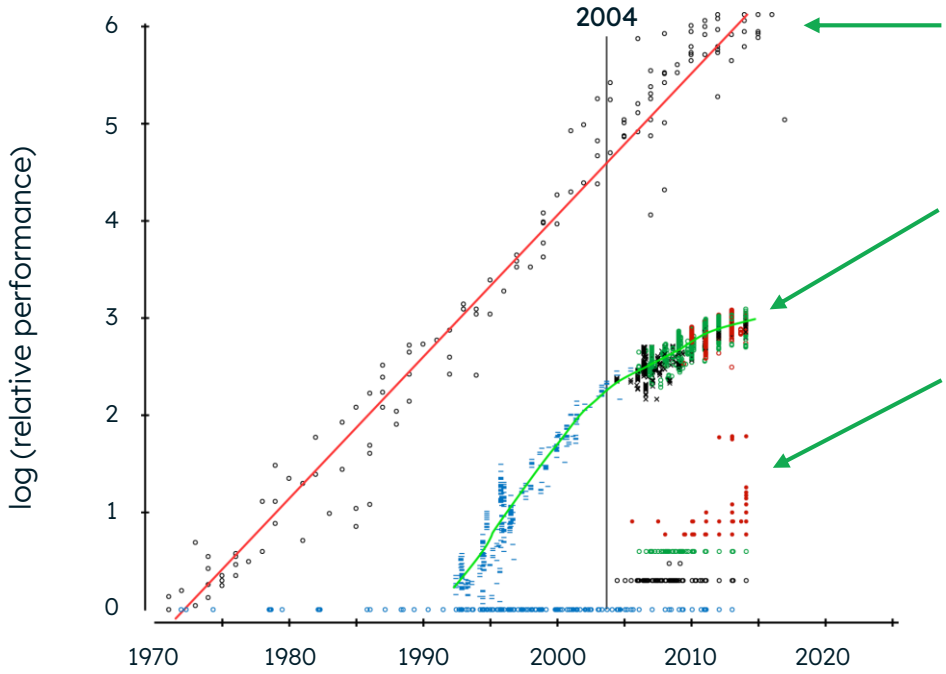
Adding more cores kept Moore's Law going



Source of Data: Wikipedia, cpudb.stanford.edu

1. Moore's Law unabated (transistors per chip) regression trend line
2. Performance improvement is slowing (SPECIntRate, normalized system throughput) Loess trend line
3. More cores means more transistors (cores per chip) Adding cores keeps Moore's Law going

In 2004, Amdahl's Law took over

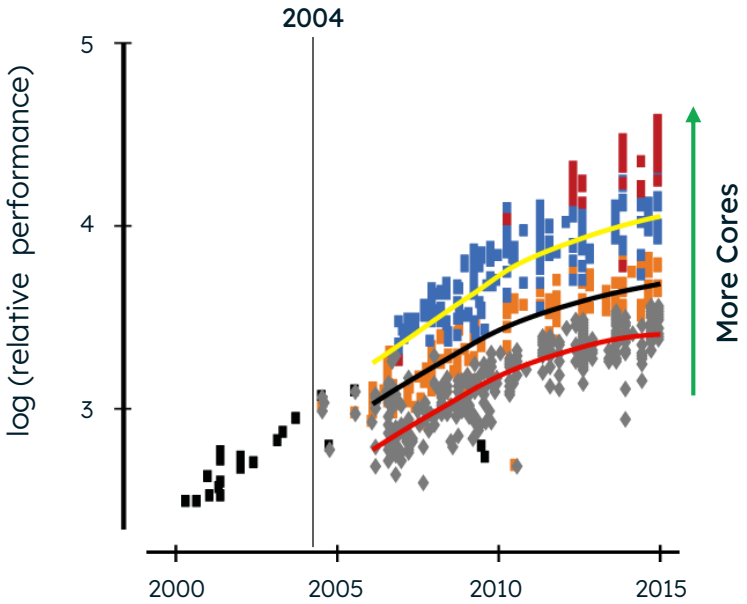


Source of Data: Wikipedia, cpudb.stanford.edu

1. Moore's Law unabated (transistors per chip) regression trend line
2. Performance improvement is slowing (SPECIntRate, normalized system throughput) Loess trend line
3. More cores means more transistors (cores per chip) Adding cores keeps Moore's Law going
4. Around 2004, introducing multi-core CPUs transitioned performance growth from scaling by Moore's Law to scaling by Amdahl's Law

Amdahl's Law incurs diminishing returns

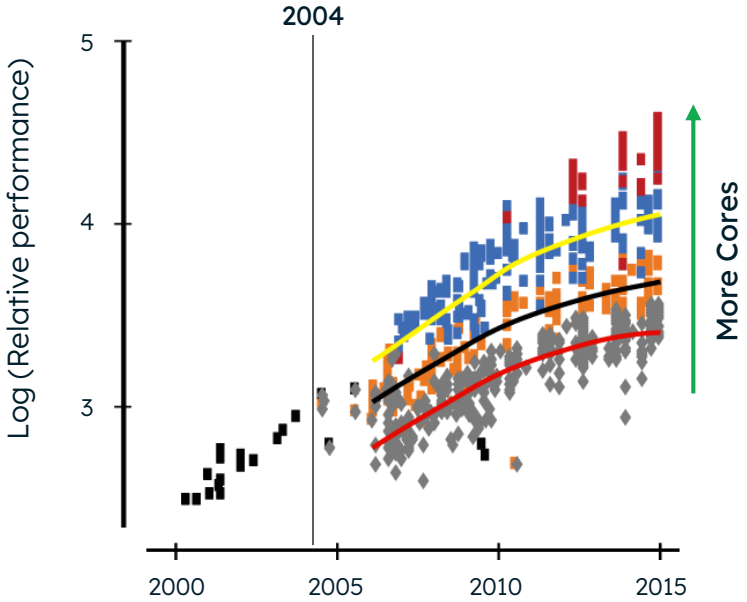
Each additional core contributes less additional speedup



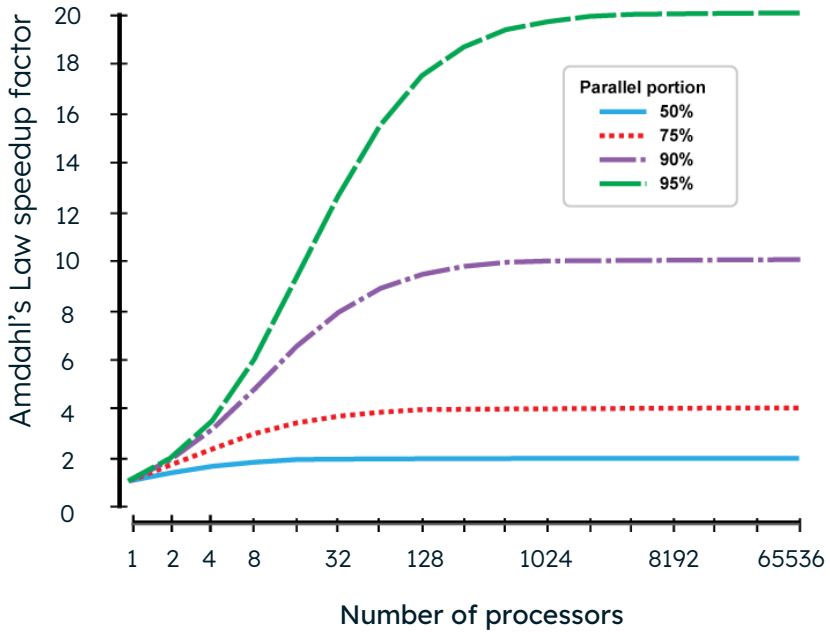
After Leiserson, et. al. *Science* 368, eaam9744 (2020)

Amdahl's Law incurs diminishing returns

Each additional core contributes less additional speedup



After Leiserson, et. al. *Science* 368, eaam9744 (2020)



Source: Wikipedia

Today's Journey Together

Intro and thesis

The first golden age of data

Amdahl's bloodless coup

The perfect storm

Pay me now or pay me later?

So, is relational the new COBOL or not?

Today's Journey Together

Intro and thesis

The first golden age of data

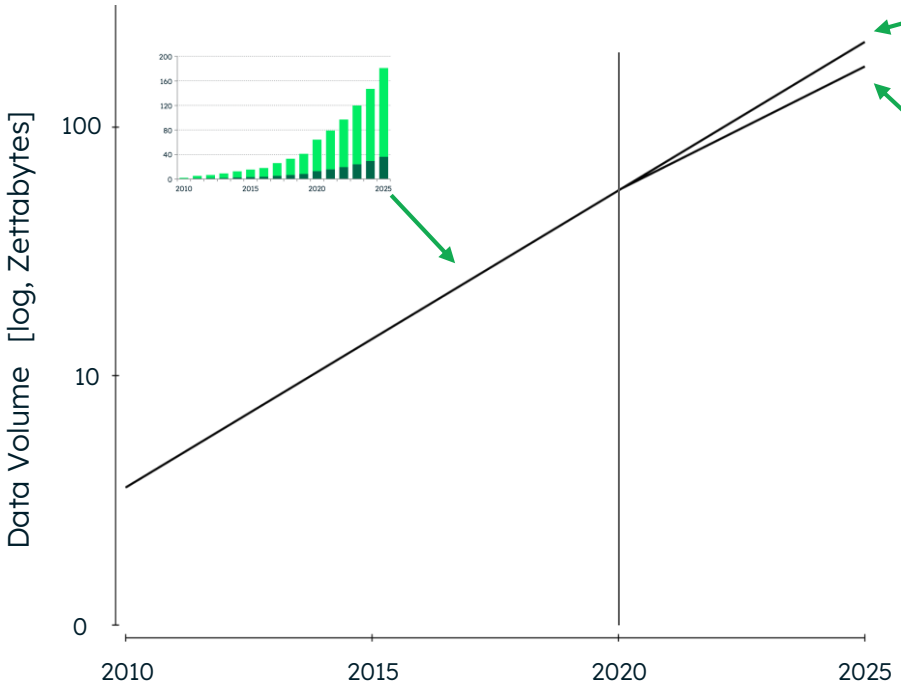
Amdahl's bloodless coup

The perfect storm

Pay me now or pay me later?

So, is relational the new COBOL or not?

Let’s look at that data explosion again

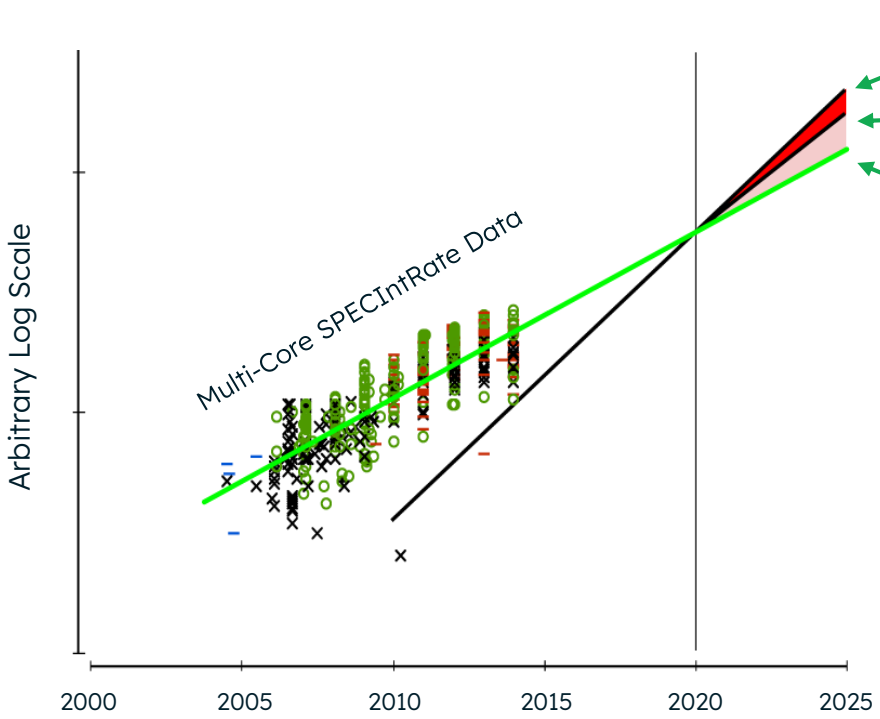


IDC & Statista pre-COVID-19 estimate 30%/yr predicts 181 Zettabytes by 2025 – that’s 181,000,000,000,000,000,000,000 Bytes

Revised post-COVID-19 estimate (2021) now 23%/yr, predicts 127 Zettabytes by 2025

Source of Data: IDC, Seagate, Statista

Data is flooding us faster than CPUs can pump it



IDC & Statista pre-COVID-19 estimate 30%/yr

Revised post-COVID-19 Estimate (2021) 23%/yr

System performance growth (≈ 21%/yr) (SPECIntRate)

Source of Data: cpudb.stanford.edu; IDC, Seagate, Statista


Data explosion
+
Moore's Law → Amdahl's Law
⇒
Growing performance gap

Today's Journey Together

Intro and thesis

The first golden age of data

Amdahl's bloodless coup

The perfect storm

Pay me now or pay me later?

So, is relational the new COBOL or not?

Today's Journey Together

Intro and thesis

The first golden age of data

Amdahl's bloodless coup

The perfect storm

Pay me now or pay me later?

So, is relational the new COBOL or not?



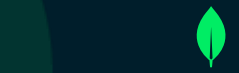
TRADE OFFS

Horses and Cars solve the same problem differently.

Cars do a better job at most things, are faster, run more efficiently, and are easier to use.

Relational and non-relational solve the same problem differently.

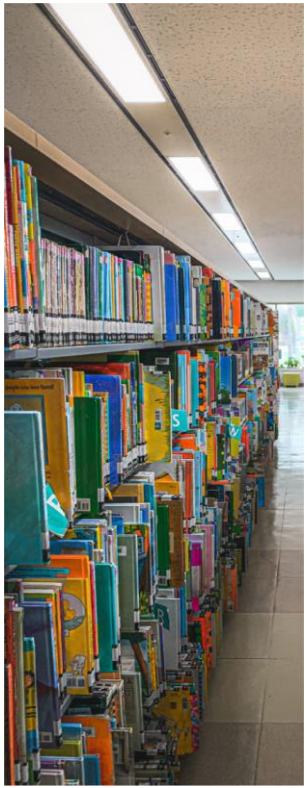
Does non-relational do a better job, faster, more efficiently, and easier?



Non-relational is more CPU, network, I/O efficient on storage and retrieval

Data model costs

Relational storage



Schema setup

Must know entire data model beforehand to design schema

vs.

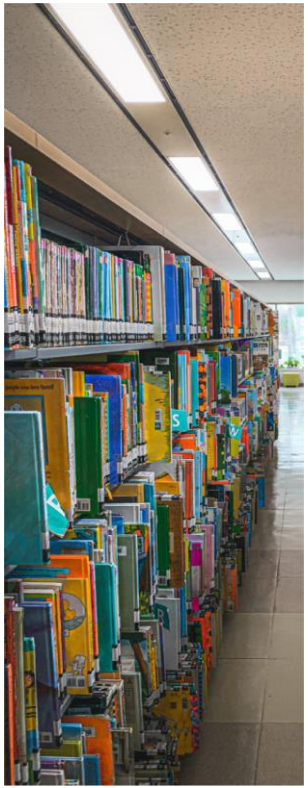
Only need to know enough to label for later retrieval & easier to change



Non-relational storage

Data model costs

Relational storage



Normalization overhead

Unpack everything & put each in its proper place at insert/append time

vs.

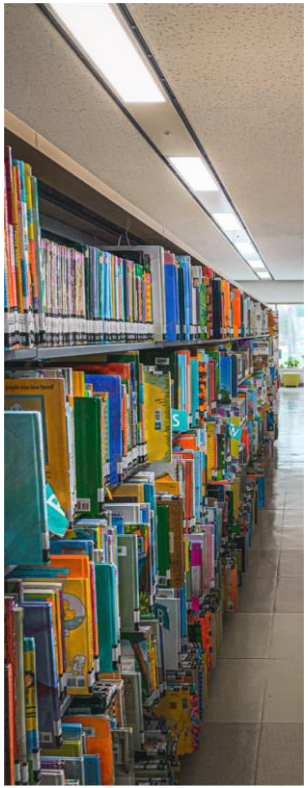
Fetch only what's needed at query time, unpack only what's needed



Non-relational storage

Lower unpacking costs

Relational storage



Gleaning data

Join together needed items from different sections (tables)

vs.

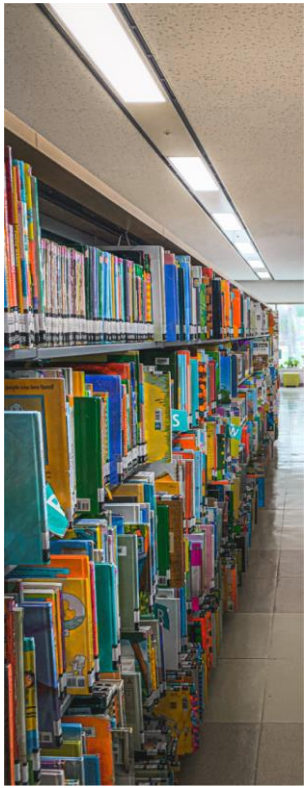
Items colocated so what we'll use is already persisted together



Non-relational storage

Lower unpacking costs

Relational storage



Normalize at use time

All data pre-normalized, so fields ready to assemble into tuple

vs.

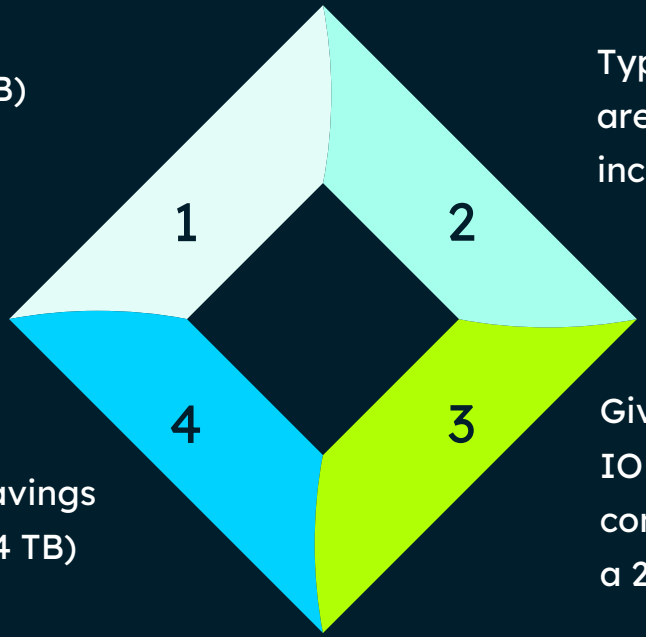
Once selected data are retrieved, normalization may be required



Non-relational storage

Huge data volumes amplify even the smallest costs

Enterprises have many terabytes (TB) of structured and unstructured data



Typical storage costs are $20,000/TB/year, including IOPs

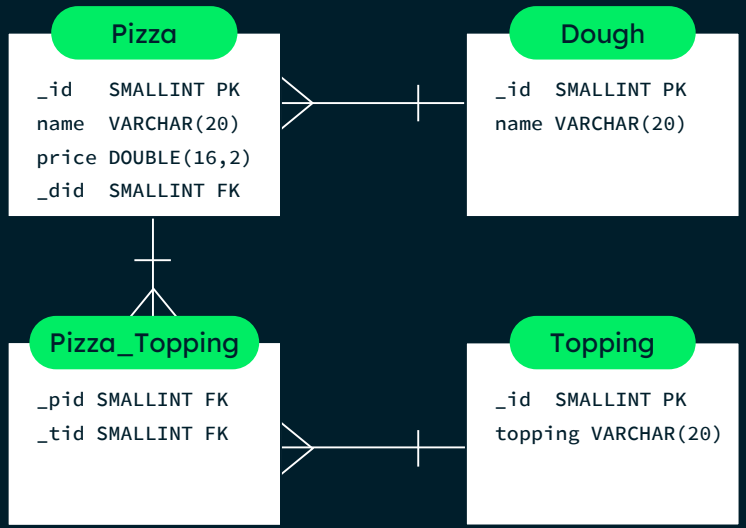
Given non-relational IOPS efficiency, conservatively assume a 25% savings on IOPS

That's $5M/year savings per petabyte (1,024 TB)

Your mileage will vary

Non-relational is easier to program against

Relational



Non-relational

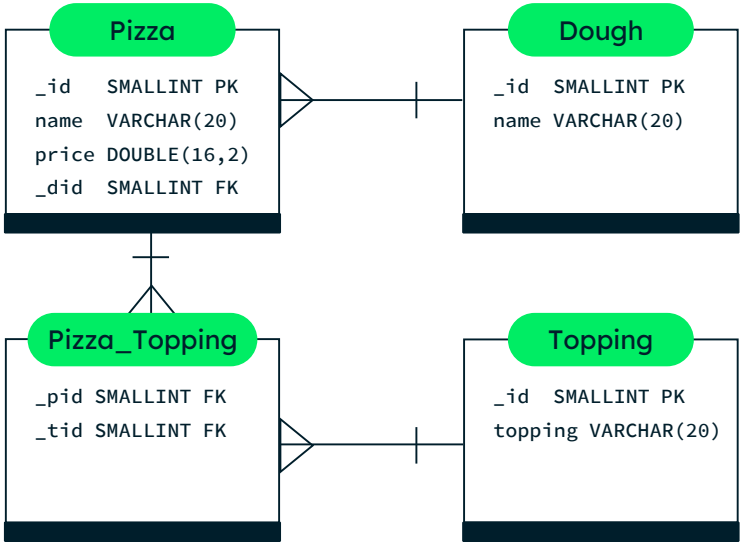


Code

```
var pizza = {
        _id: 5cf0029,
        name: "Margherita",
        price: 9.5,
        ingredients: {
        dough: "New York Style",
        toppings: ["tomatoes",
        "mozzarella"]
        }
};
```

OBJECT RELATIONAL MAPPER

Relational



Code

```
var pizza = {
          _id: 5cf0029,
          name: "Margherita",
          price: 9.5,
          ingredients: {
          dough: "New York Style",
          toppings: ["tomatoes",
          "mozzarella"]
          }
};
```

Document

```
{
  "_id": "5cf0029",
  "name": "Margherita",
  "price": "9.5",
  "ingredients" {
          dough: "New York Style",
          toppings: ["tomatoes",
          "mozzarella"]
  }
}
```

Code

```
var pizza = {
          _id: 5cf0029,
          name: "Margherita",
          size: "medium",
          price: 9.5,
          ingredients: {
          dough: "New York Style",
          toppings: ["tomatoes",
          "mozzarella"]
          }
};
```

Document

```
{
  "_id": "5cf0029",
  "name": "Margherita",
  "price": "9.5",
  "ingredients" {
          dough: "New York Style",
          toppings: ["tomatoes",
          "mozzarella"]
  }
}
```

Code

```
var pizza = {
          _id: 5cf0029,
          name: "Margherita",
          size: "medium",
          price: 9.5,
          ingredients: {
          dough: "New York Style",
          toppings: ["tomatoes",
          "mozzarella"]
          }
};
```

Document

```
{
  "_id": "5cf0029",
  "name": "Margherita",
  "size": "medium",
  "price": "9.5",
  "ingredients" {
          dough: "New York Style",
          toppings: ["tomatoes",
          "mozzarella"]
  }
}
```

Non-relational APIs are more intuitive than SQL

People tend to think about a processes moving toward a goal (imperative) rather than starting from a goal and working backward (declarative)

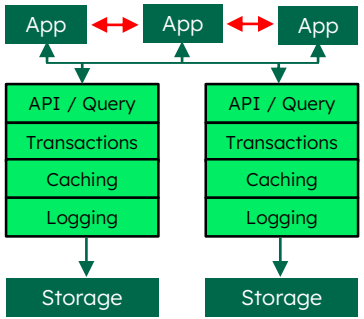
	Declarative Programming (SQL)	Imperative Programming (*.*)
Paradigm	Describe the desired end result, NOT all the intermediate work steps	Provide precise steps for computer to follow to get result
Comprehensibility	Code is characterized by a high level of abstraction. Complex/Sophisticated apps can be opaque	Flow to goal is natural, but structure and flow can become incomprehensible if not managed
Flexibility	Can be hard to take characteristics of individual applications into account during programming	Application variability managed by libraries, frameworks, and code reuse
Debugging	EXPLAIN and re-run, again and again	Single stepping/RAM inspection


Non-relational
scales easier

Scaling needs to be done at the right level of the stack

Scale via application

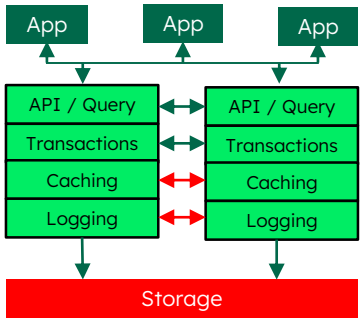
Coupled at the application layer



Applications must understand topology

Scale via shared storage

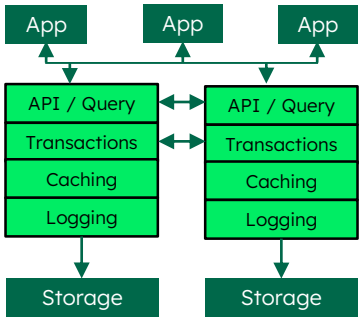
Coupled at all layers



Orchestrated state management

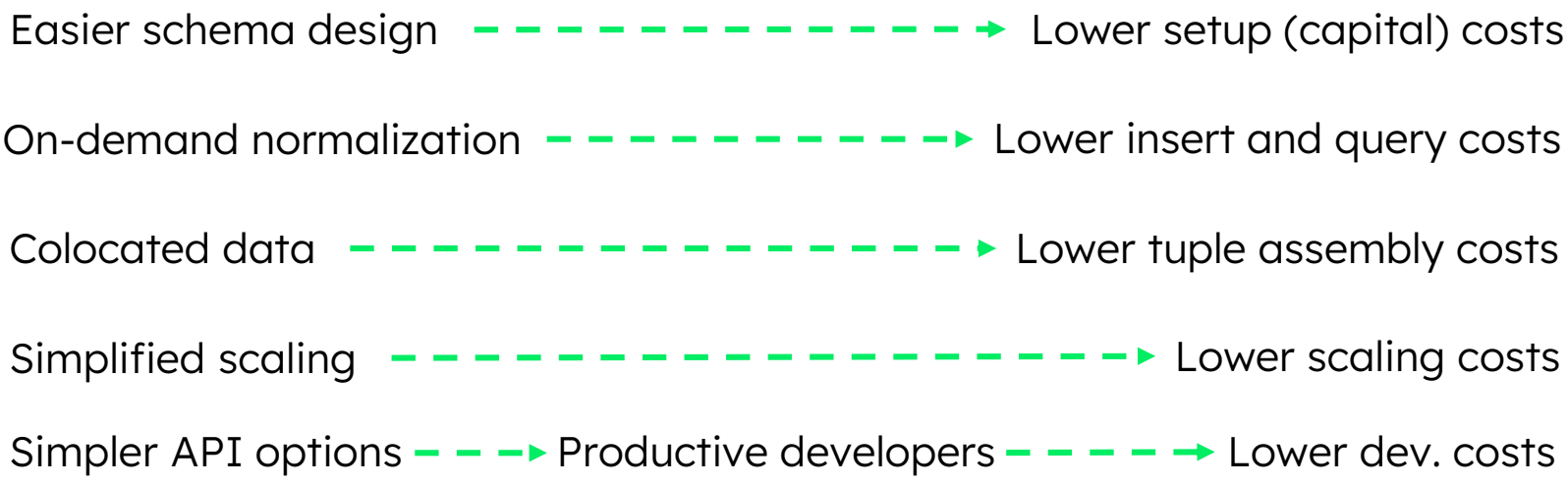
Scale via database

Coupled at logical layers only



Queries easily partition and deploy

Non-relational technology cost savings



Is relational the new COBOL?

Why do some technologies endure while others do not?



FADING OUT

Railroads (in US)

Lotus Notes

IBM Mainframes

Coal

FORTRAN, COBOL, PHP

GONE

Horses

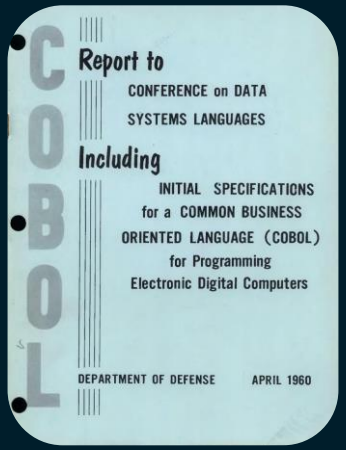
8-Track Tapes

Lotus 123

Blimps and Zeppelins

Slide Rulers

Cobol has endured in its niche



Many enterprises still use COBOL today

Because it still “just works”

Can’t justify the switching cost

Avoids perceived migration risk

Could relational technologies have similar characteristics?



Information Retrieval

A Relational Model of Data for Large Shared Data Banks

E. F. Codd

IBM Research Laboratory, San Jose, California

1. Relational Model and Normal Form

ume 13 / Number 6 / June, 1970

Developers are clever and can coerce heterogenous data to conform to relational's rules

Will it just work?

What will the cost difference be?

Will it be less than switching cost?

Relational longevity possible but not assured

The homogeneous / heterogeneous data mix likely to will remain



20% of a lot of data is still a lot of relational-friendly data

Possible futures for relational

Continue with relational

Possible futures for relational

Continue with relational

Relational for structured,
Non-relational for unstructured

Possible futures for relational

Continue with relational

Relational for structured,
Non-relational for unstructured

Migrate relational data when economically viable

Possible futures for relational

Continue with relational

Relational for structured,
Non-relational for unstructured

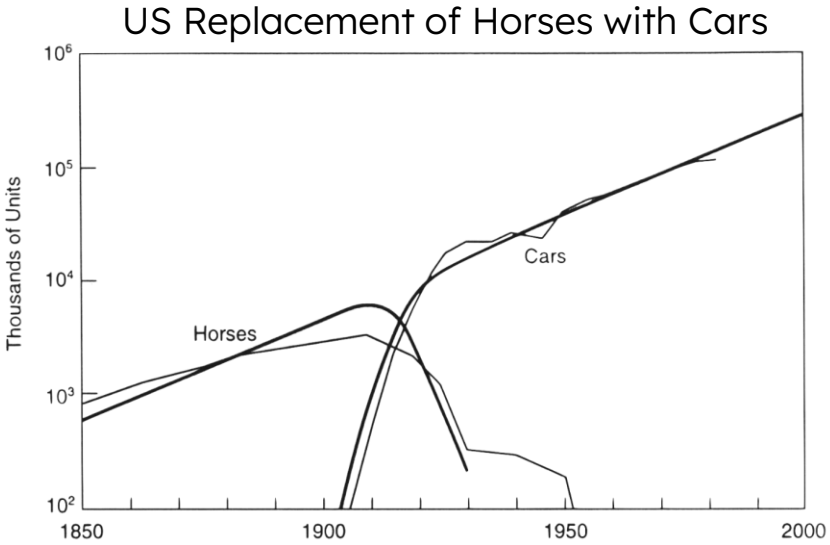
Migrate relational data
when economically viable

Non-relational platforms to offer SQL compatibility

Is relational
the next COBOL?

Economics will decide

Did they see it coming?



After Nebojsa Nakicenovic (1997).
Technological Change As A Learning Process.
Induced Technology Workshop, IIASA.
26-27 June 1997.

Thank you for your time.


Mark Porter


@marklovestech


marklovestech.com