Activation Energy: Technology Landscapes and Forces of Adoption

Fall 2023

Introduction

Hi

- I'm an industrial programmer, not researcher!
- Here by invitation, mainly to tell undergrads strange history stories in a different talk.
- This talk is more reflecting on own experience, to provoke conversation and speculate a bit about the future.

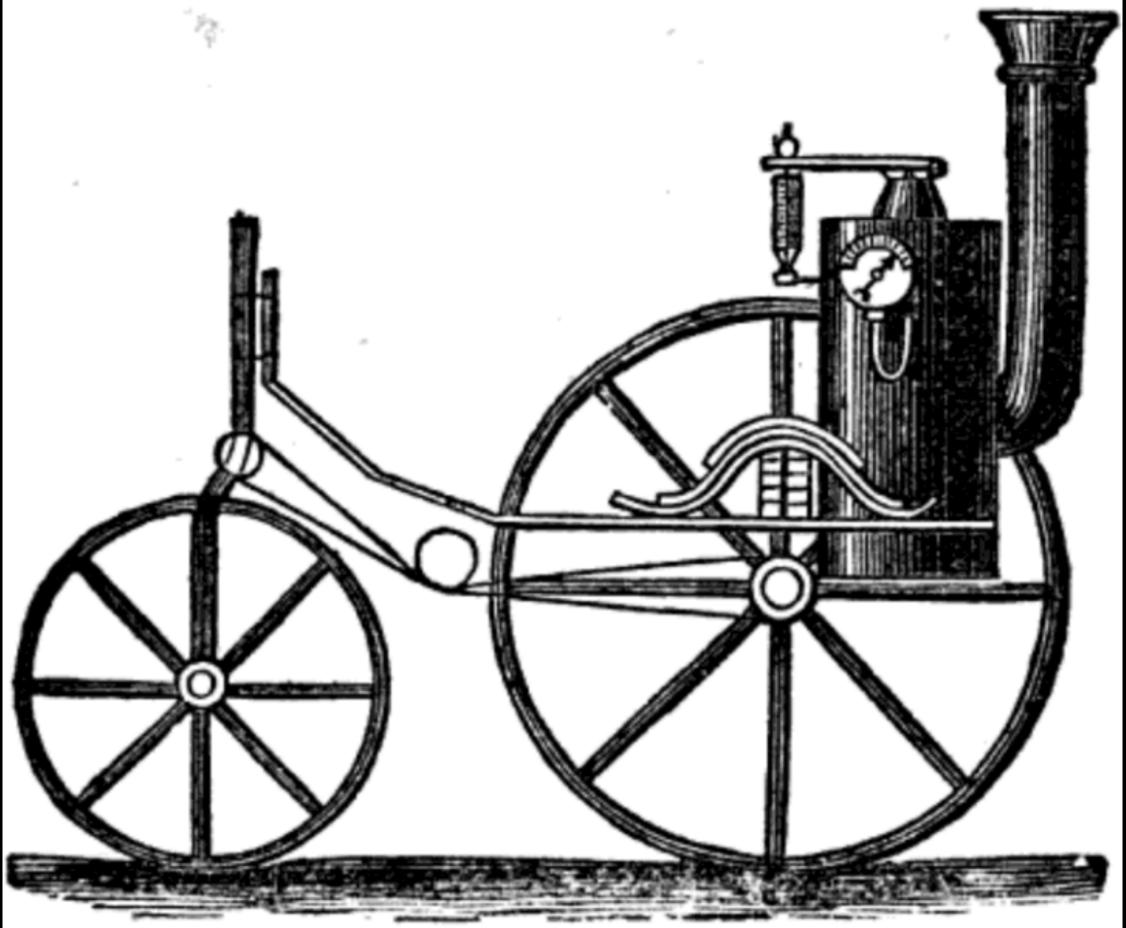
The Author in 1984 learning how to get out of vi

Technology Adoption

- I've had good fortune to have front-row seating during two technology adoption events in programming
    - Distributed version control
    - Memory-safe systems PLs
- I gather this is something every researcher wants to have happen to their work!

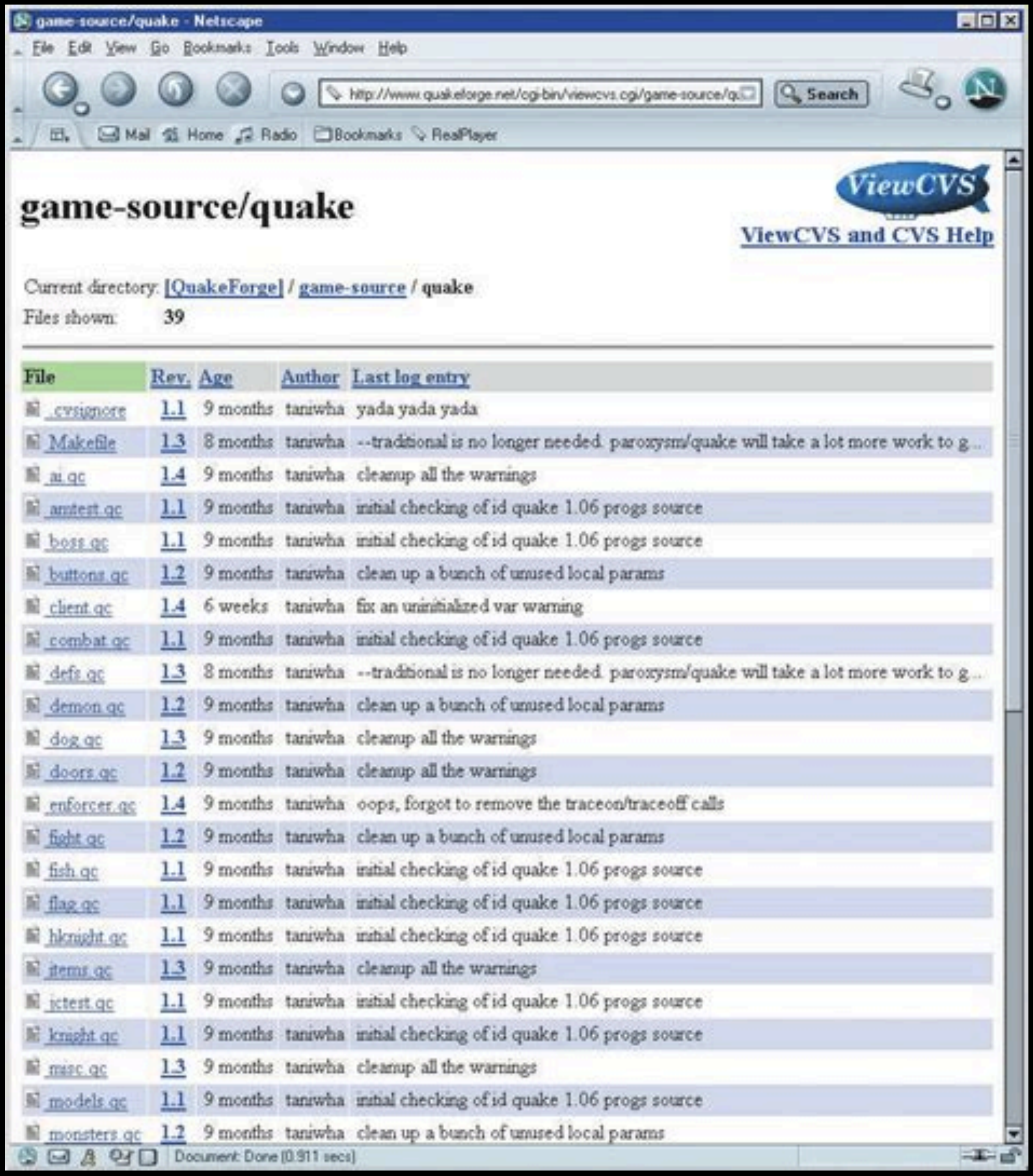
Conditions

- Each saw several attempts "before conditions were right"
- This is a talk about *conditions being right*, and not about anything intrinsic in the tech
  - Thesis: *tech maturity* only <u>one ingredient</u> in uptake
- At the end, will talk about application to .. databases?



Distributed Version Control

- For a long time (15 years?) *everyone* used CVS
- Flurry of activity in early 2000s
  - SVN, DCVS, CVSNT, OpenCM, BitKeeper, Arch, Bazaar, CodeVille, Monotone (mine), Hg, Git
- Git won, for several reasons



Memory-safe systems PLs

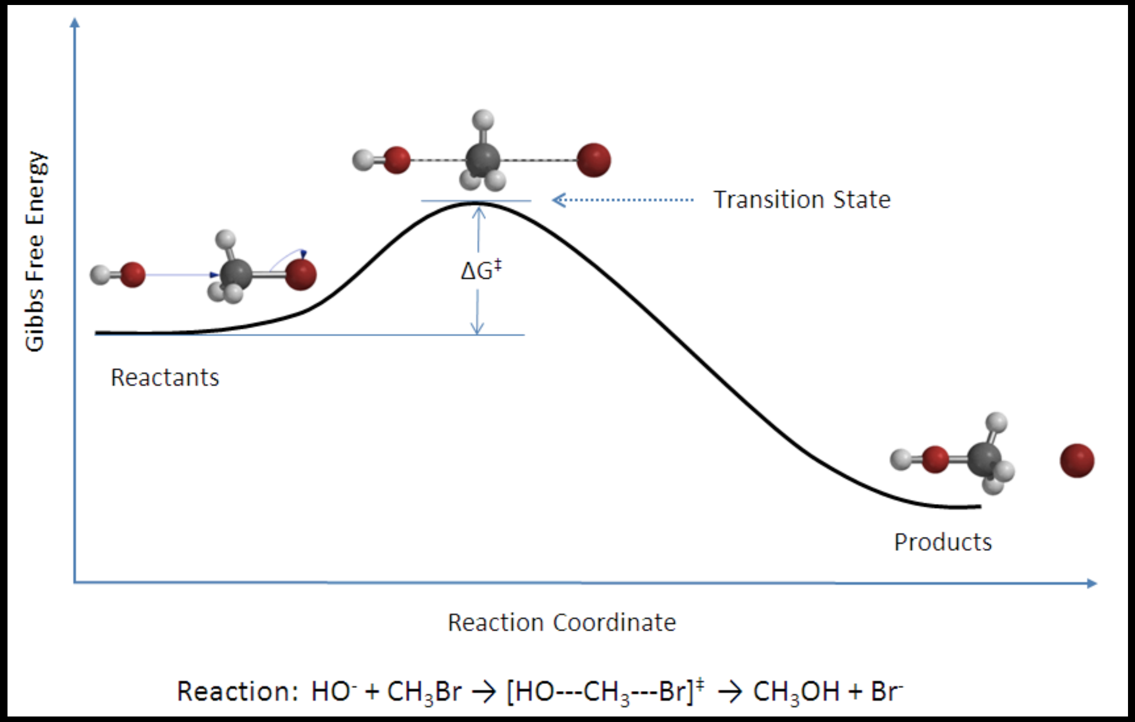
- For a *long time* (25 years?) *everyone* used C & C++; there was a "VM language" detour (Java & C#) but didn't unseat
- Flurry of activity in late 2000s / early 2010s
  - Cyclone, Nim, ParaSail, Go, Rust (mine), Swift, Clay, BitC
- I'm not going to declare any "winner" here yet

Innovation alone didn't drive either event

- If you look into proclaimed "technical innovations" in any of these projects, you'll see stuff lying around for *decades*
    - Content addressing and linked timestamps predated Monotone by 10+ years
    - Linear Lisp, Clean, Cyclone predated Rust by 10+ years
- Just "slow tech transfer"?

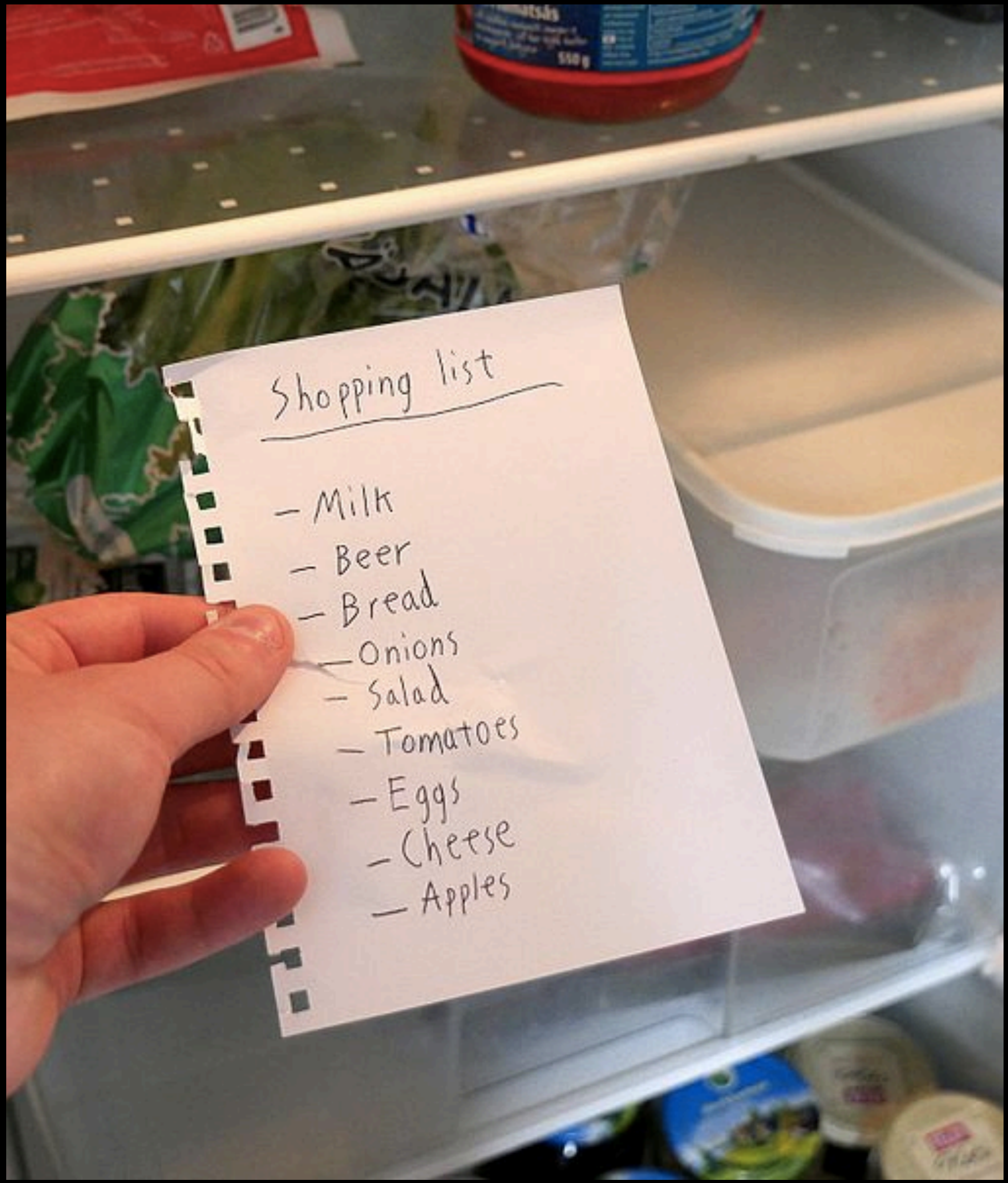
Another model

- Chemistry analogy: activation energy
- Technology sits in stable state due to barriers (people, processes)
- Conditions dictate change of state:
    - Pressure to change raised ("forcing")
    - Barriers to change lowered ("enabling")
- Many people sense this happening and throw their hats in the ring!
- Innovation happened *before*



Consequences of model

- Many factors force & enable
    - Make a list of several things wrong with current systems, consider fixing many at once
    - Don't neglect the enabling: what's preventing change? did any old barriers change?
- Grab bag of Other Stuff will "come along for the ride"
    - Some "technical upgrades", some "downgrades"
    - Just accept this will happen



Sometimes changes bring technical "downgrades"

- Minicomputers to micros
- Desktop software to web
- Mice and keyboards to touch
- Static to dynamic PL designs
- Strong to eventual consistency
- These are not necessarily *bad* but they are "downgrades" in the sense of *removing existing tech* because the new state has different requirements

Distributed Version Control

DVC forcing conditions

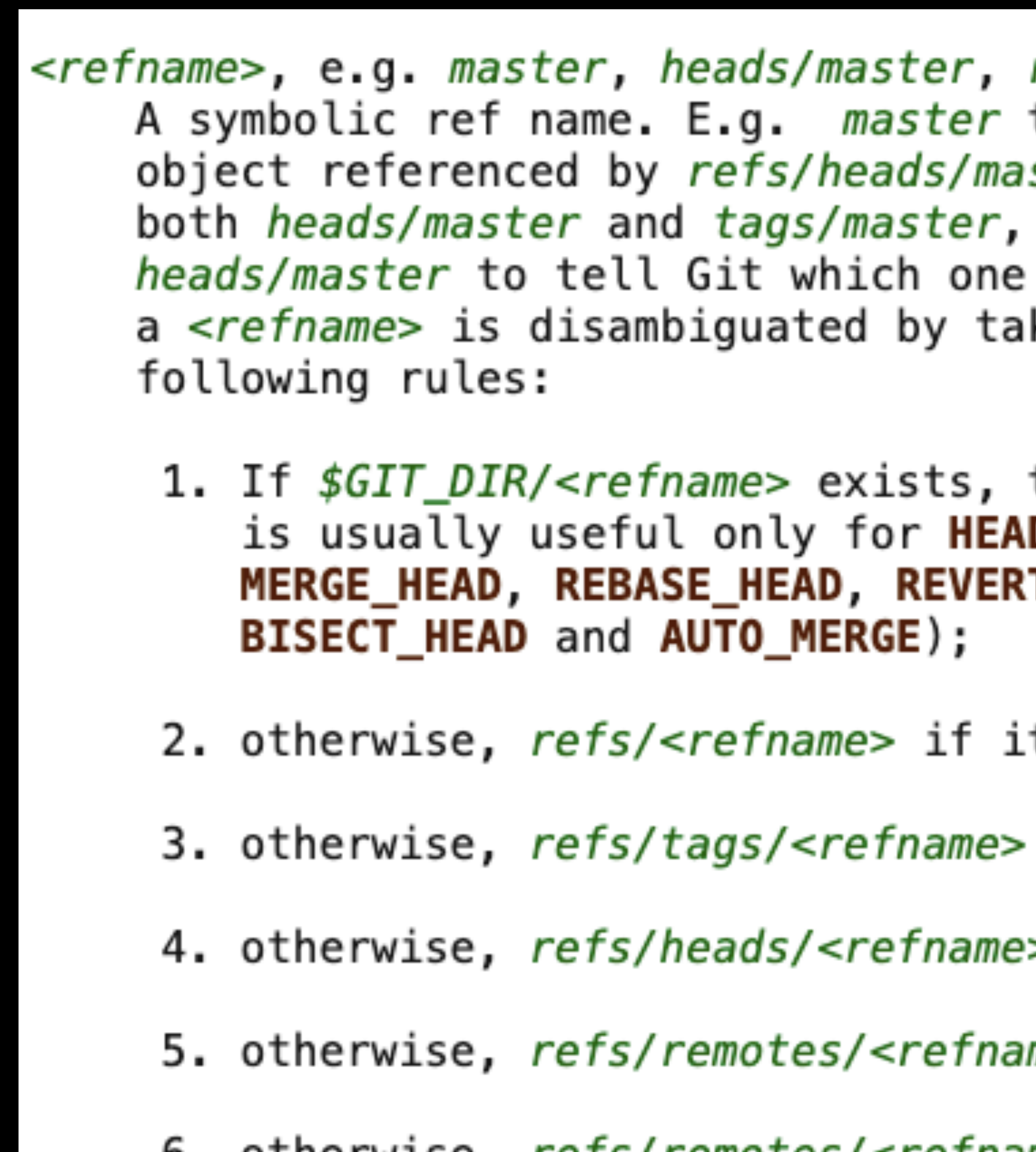
- CVS was inadequate in *many ways*
  - Non-atomic commits
  - Synchronous online "updates" that clobber workspace
  - No offline actions at all
  - Branching slow and fragile
  - Didn't remember last merge
  - No ability to fork, admin is gatekeeper to project history
  - Renames, binary data, etc. etc.

DVC enabling conditions

- Disks big enough and networks fast enough to replicate whole repo to clients
- Servers obtainable enough for users to host their own repos
- Widespread cryptography to play around with new models of collaboration and trust (SSH, PGP, SHA-1)

DVC technical upgrades and downgrades

- Upgrades:
  - Content addressing (venti)
  - Linked timestamps
  - Binary diffing (rsync, xdelta)
  - Atomicity, renames, better merges
- Downgrades:
  - Weakened confidentiality control
    - Every replica gets everything!
  - Weakened integrity control
    - Every replica claims truth!
  - UI got extremely complex
    - 3 possible meanings of any git ref?!



<refname>, e.g. *master*, *heads/master*, *r*
A symbolic ref name. E.g. *master* t
object referenced by *refs/heads/mas*
both *heads/master* and *tags/master*,
heads/master to tell Git which one
a *<refname>* is disambiguated by tak
following rules:

1. If *$GIT_DIR/<refname>* exists, t
   is usually useful only for **HEAD**
   MERGE_HEAD, **REBASE_HEAD**, **REVERT**
   BISECT_HEAD and **AUTO_MERGE**);
2. otherwise, *refs/<refname>* if it
3. otherwise, *refs/tags/<refname>*
4. otherwise, *refs/heads/<refname>*
5. otherwise, *refs/remotes/<refnam*
6. otherwise, *refs/remotes/<refnam*

Memory-Safe Systems Programming Languages

Memory-Safe Systems PL forcing conditions

- C++ memory unsafety causing constant security exploits
- Much worse with threads, and suddenly CPUs are multicore
- Nightmare build systems, using 3rd party packages hard
- Illegible template errors
- Younger devs avoiding entirely

Memory-Safe Systems PL enabling conditions

- LLVM, LLVM, LLVM
- Wealthy industrial benefactors from dotcom & mobile booms
- Free academic publications: Citeseer and ArXiv
- Accessible new books on type systems and compilers (Pierce, Appel)

Memory-Safe Systems PL technical upgrades and downgrades

- Upgrades:
  - GC, RC, affine types or at least *some* discipline for general memory safety
  - Sometimes also data-race freedom
  - *FP-style* tools for generic code (protocols, typeclasses, existentials)
  - Integrated build, test & packaging
- Downgrades:
  - Often new fussy static rules (lifetimes?!)
  - Mostly single "reference implementations"
  - *OO-style* tools for generic code (overloading, specialization, inheritance)

```
Compiling url v2.2.1
Compiling console v0.14.1
Compiling env_logger v0.8.3
Compiling globset v0.4.6
Compiling rand_chacha v0.3.0
Compiling rand_pcg v0.2.1
Compiling rand_chacha v0.2.2
Compiling chrono v0.4.19
Compiling alpm-utils v0.6.2
Compiling indicatif v0.16.0
Compiling rand v0.8.3
Compiling rand v0.7.3
Compiling tempfile v3.2.0
Compiling phf_generator v0.8.0
Compiling phf_codegen v0.8.0
Compiling string_cache_codegen v0.5.1
Compiling selectors v0.22.0
Compiling markup5ever v0.10.1
Compiling tokio-macros v1.1.0
Compiling futures-macro v0.3.14
Compiling pin-project-internal v1.0.7
Compiling phf_macros v0.8.0
Compiling cssparser v0.27.2
Compiling html5ever v0.25.1
Compiling cssparser-macros v0.6.0
Compiling derive_more v0.99.13
Compiling smart-default v0.6.0
Compiling phf v0.8.0
Compiling futures-util v0.3.14
Compiling tokio-util v0.6.6
Compiling tokio-native-tls v0.3.0
Compiling async-compression v0.3.8
Compiling futures-executor v0.3.14
Compiling pin-project v1.0.7
Compiling h2 v0.3.3
Compiling futures v0.3.14
Compiling aur-fetch v0.9.1
Compiling serde_urlencoded v0.7.0
Compiling string_cache v0.8.1
Compiling hyper v0.14.7
Compiling kuchiki v0.8.1
Compiling hyper-tls v0.5.0
Compiling reqwest v0.11.3
Compiling raur v5.0.1
Compiling aur-depends v0.14.3
Compiling paru v1.6.1 (/home/kousekip/.cache/paru/clone/paru/src/paru-1.6.1)
 Building [========================> ] 240/241: paru(bin)
```

Next-Generation Databases!
(and maybe IFC)

Databases

- Thesis: pressure building for a technology adoption event in databases (or "data systems")
    - Pure speculation on my part
- Largely same structure since 1970s, but now with WAN web and mobile clients interacting with DB via manual glue code
    - System full of annoyances!
    - Biggest shift was "NoSQL", which *removed* features!

Rank			DBMS	D
Nov 2023	Oct 2023	Nov 2022		
1.	1.	1.	Oracle	Re
2.	2.	2.	MySQL	Re
3.	3.	3.	Microsoft SQL Server	Re
4.	4.	4.	PostgreSQL	Re
5.	5.	5.	MongoDB	Do
6.	6.	6.	Redis	Ke
7.	7.	7.	Elasticsearch	Se
8.	8.	8.	IBM Db2	Re
9.	9.	↑ 10.	SQLite	Re
10.	10.	↓ 9.	Microsoft Access	Re

Database forcing conditions

- Fragile replication and backup
- Bad versioning, incrementalism
- Poor built-in query languages
- Impedance mismatches, low integration
  - Code/DB data model (ORMs)
  - WAN/DB (auth, caching)
  - Repetitive manual UIs for CRUD
  - Schema migration & reflection
- Siloing, lack of federation, schema interop
- Increasing data regulations (residency, retention, privacy, deletion)

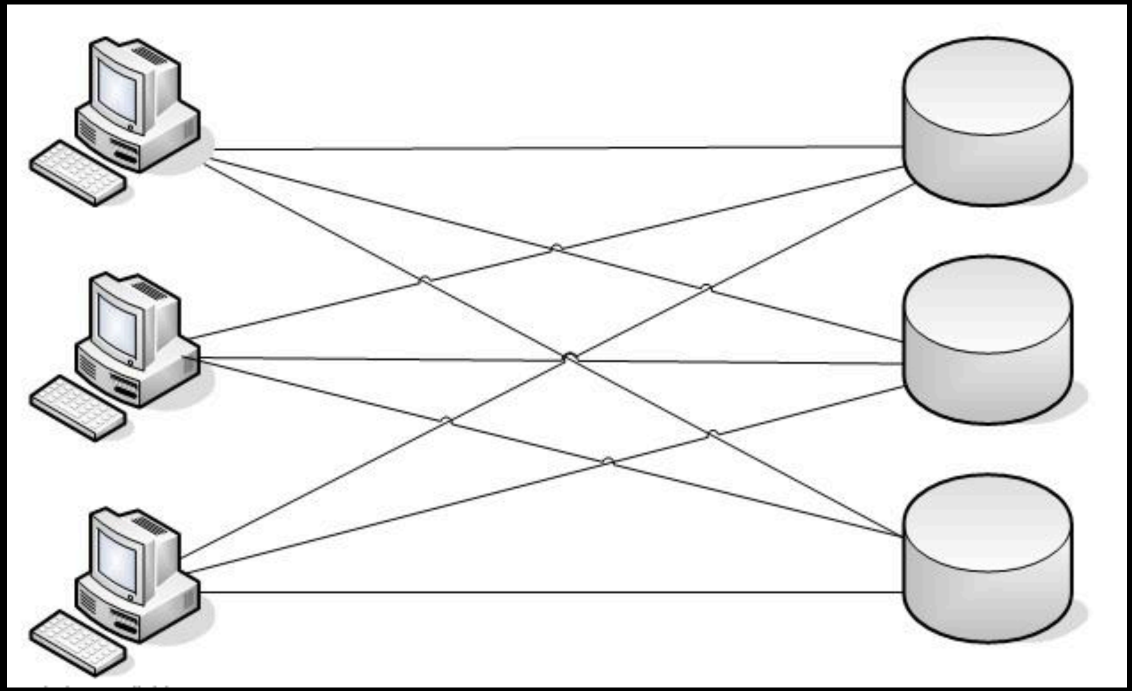
```
from sqlalchemy import *
from sqlalchemy.ext.declarative import declarative_
from sqlalchemy.orm import relation, sessionmaker

Base = declarative_base()

class Movie(Base):
    __tablename__ = "movies"

    id = Column(Integer, primary_key=True)
    title = Column(String(255), nullable=False)
    year = Column(Integer)
    directed_by = Column(Integer, ForeignKey("direc

    director = relation("Director", backref="movies

    def __init__(self, title=None, year=None):
        self.title = title
        self.year = year

    def __repr__(self):
        return "Movie(%r, %r, %r)" % (self.title, s

class Director(Base):
    __tablename__ = "directors"

    id = Column(Integer, primary_key=True)
    name = Column(String(50), nullable=False, uniq
```

Database enabling conditions

- Dramatic single-node perf improvement
  - NVMe, io_uring, large memories, multicore, GPUs
  - Vectorized interpreters (VectorWise)
- Commodity columnar formats (Parquet, ORC, Arrow)
- Commodity cloud object storage (S3)
- Theory improvements
  - Deterministic DB protocols (Calvin)
  - Differential dataflow, IVM, "Datalog 2.0"
  - Commodity machine learning
    - Text, vector search, schema matching
- Stable set of "native UI" targets
- Accessible new books on databases and distributed systems (Petrov, Kleppmann)
- Possibly also Rust :)

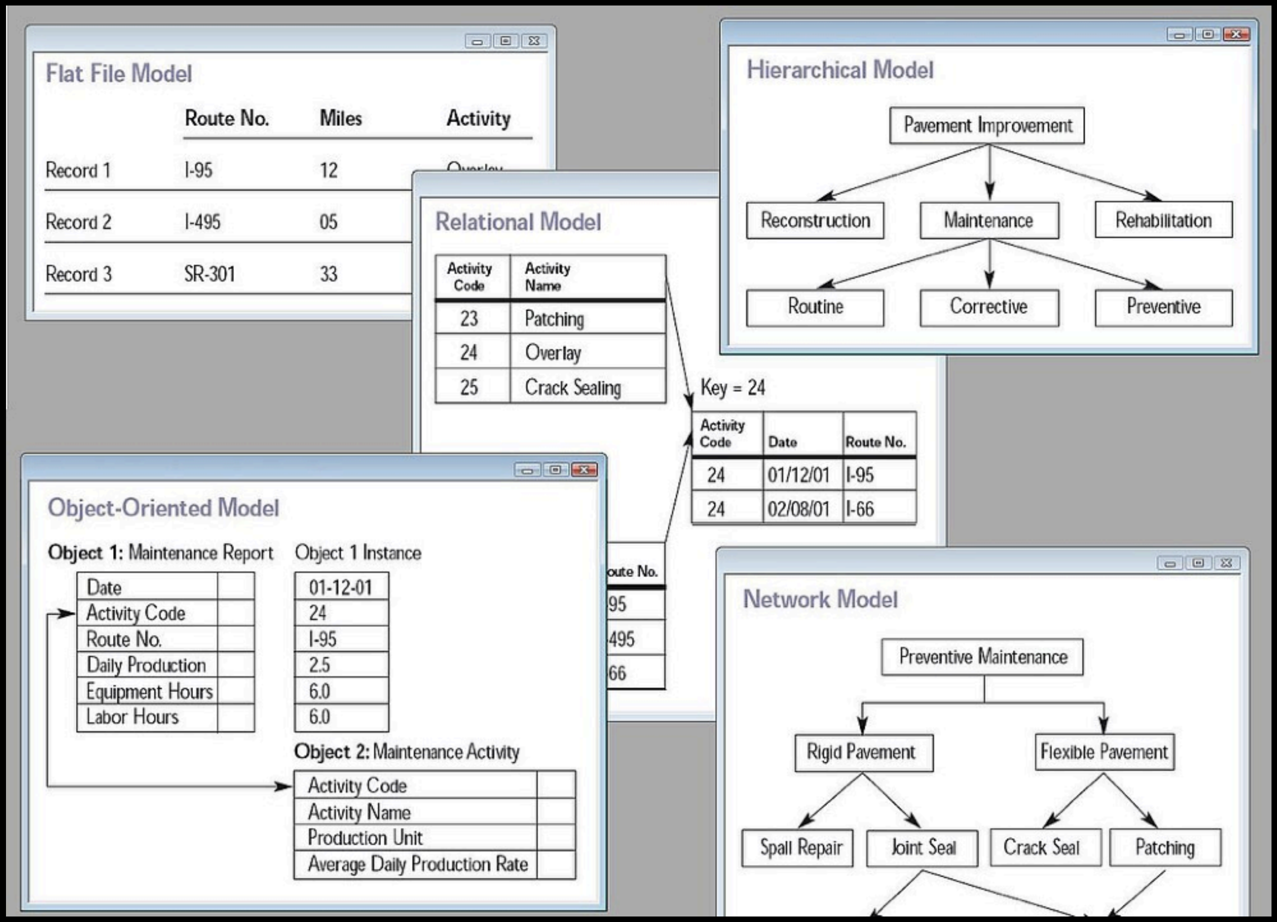
(Plausible) Database technical upgrades and downgrades

- Upgrades:
  - Provenance, data-policy compliance
  - Code in DB; typed, compositional PLs
  - Standard system-provided CRUD UIs
  - Federation, pub/sub, WAN clients
  - IVM and versioning
  - Online hot replicas & continuous backups
- Downgrades:
  - Interactive transactions, dependent queries
  - Large menu of isolation levels, complex concurrency control for peak performance
  - ARIES, complex durability protocols



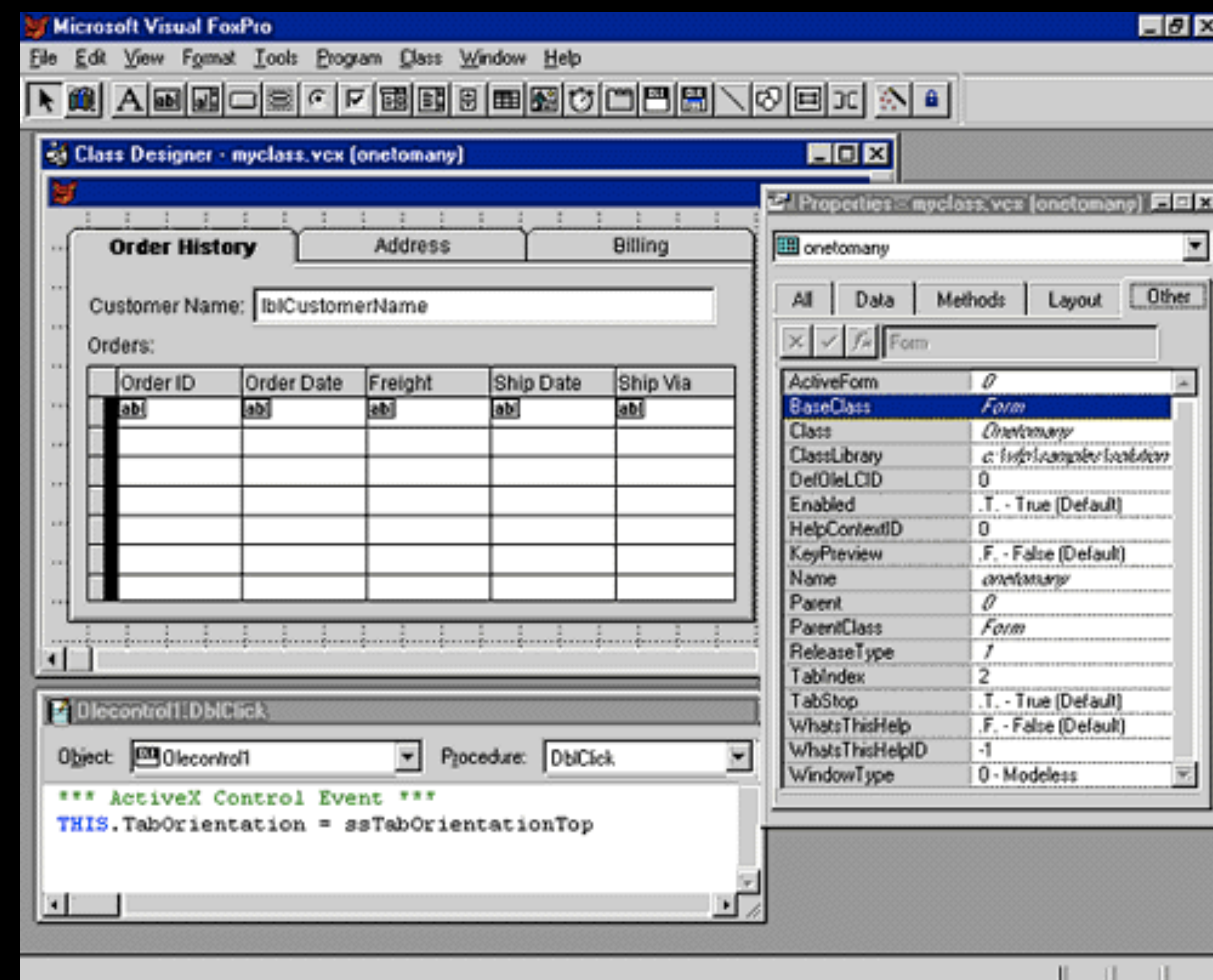
Surely we have enough databases already?

- *Many* address <u>some subset</u> of issues!
  - dbdb.io has 900+ DBs,
    db-engines.com has 400
- *Far fewer* addressing <u>structural issues</u> of the whole "data system"
  - Most treat DB as "separate part"
  - A few attempts that didn't stick:
    - "distributed objects"
    - "semantic web" / "linked data"
    - "web3"



Looking to The Past?

- My view: we took a bit of a wrong turn with the web?
  - Or at least .. the web only does <u>some things</u> well
- 80s-90s 4GLs allowed *simple* development of *end-to-end* apps
- DB, PL, UI (forms & tables) all co-designed, tightly integrated
  - Doing today would embrace WAN
  - No-code / Low-code systems are currently dabbling here
- Market: line-of-business and ERP apps



Information Flow Control (IFC)

- IFC hasn't really made it *on its own* (47 years since Denning!)
- It might come along for the ride, *if databases shift*
- And/or be basis of modelling:
  - Consistency, Availability, Retention, Residency, Provenance ... lots of stuff!
  - Cornell projects & alumni already explored several of these:
    - Fabric, Qimp, MixT, ...

Operating Systems — R.S. Gaines, Editor

A Lattice Model of Secure Information Flow

Dorothy E. Denning
Purdue University

This paper investigates mechanisms that guarantee secure information flow in a computer system. These mechanisms are examined within a mathematical framework suitable for formulating the requirements of secure information flow among security classes. The central component of the model is a lattice structure derived from the security classes and justified by the semantics of information flow. The lattice properties permit concise formulations of the security requirements of different existing systems and facilitate the construction of mechanisms that enforce security. The model provides a unifying view of all systems that restrict information flow, enables a classification of them according to security objectives, and suggests some new approaches. It also leads to the construction of automatic program certification mechanisms for verifying the secure flow of information through a program.

Key Words and Phrases: protection, security, information flow, security class, lattice, program certification

CR Categories: 4.35

Copyright © 1976, Association for Computing Machinery, Inc. General permission to republish, but not for profit, all or part of this material is granted, provided that ACM's copyright notice is given and that reference is made to the publication, to its date of issue, and to the fact that reprinting privileges were granted by permission of the Association for Computing Machinery.

A version of this paper was presented at the Fifth ACM Symposium on Operating Systems Principles, The University of Texas at Austin, November 19–21, 1975.

Work reported herein was supported in part by the National Science Foundation under grants GJ-43176 and GJ-41289 and by IBM under a fellowship. Author's present address: Computer Sciences Department, Purdue University, West Lafayette, IN 47907.

1. Introduction

The security mechanisms of most computer systems make no attempt to guarantee secure information flow. "Secure information flow," or simply "security," means here that no unauthorized flow of information is possible. In the common example of a government or military system, security requires that processes be unable to transfer data from files of higher security classifications to files (or users) of lower ones: not only must a user be prevented from directly reading a file whose security classification exceeds his own, but he must be inhibited from indirectly accessing such information by collaborating in arbitrarily ingenious ways with other users who have authority to access the information [19].

Most access control mechanisms are designed to control immediate access to objects without taking into account information flow paths implied by a given, outstanding collection of access rights. Contemporary access control mechanisms, such as are found in Multics [18, 20] or Hydra [24], have demonstrated their abilities to enforce the isolation of processes essential to the success of a multitask system. These systems rely primarily on assumptions of "trustworthiness" of processes for secure information flow among cooperating processes. Though it is mainly of theoretical interest, Harrison et al. [12] have recently demonstrated that in general it may be undecidable whether an access right to an object will "leak" to a process in a system whose access control mechanism is modeled by an access matrix [11, 15].

In our research into this problem, we sought to find suitable and viable restrictions according to which the security of a system would not only be decidable, but simply so. Our results show that suitable constraints do indeed exist, and moreover within the context of a richly structured model.

2. The Model

2.1 Description

An *information flow model FM* is defined by

$FM = \langle N, P, SC, \oplus, \rightarrow \rangle$.

$N = \{a, b, \ldots\}$ is a set of logical storage *objects* or information receptacles. Elements of N may be files, segments, or even program variables, depending on the level of detail under consideration. Each user of the system may also be regarded as an object. $P = \{p, q, \ldots\}$ is a set of *processes*. Processes are the active agents responsible for all information flow.

236 — Communications of the ACM — May 1976, Volume 19, Number 5

Or ... maybe not?

- I may be wrong about how tech adoption works
- I may be wrong about how ripe databases are for an overhaul
- This is just a hunch / talk idea
- Maybe I just read some database papers and books and got too excited!
- Please don't blame me for sending you on wild research goose-chase!

Fini