GitLab 2017 Post-Mortem

Derek Rayside, P.Eng., Ph.D.

Post-mortem report by GitLab:
https://about.gitlab.com/blog/2017/02/10/postmortem-of-database-outage-of-january-31/

A few famous post-mortems …

Quebec City Bridge 1907

Collapsed before use

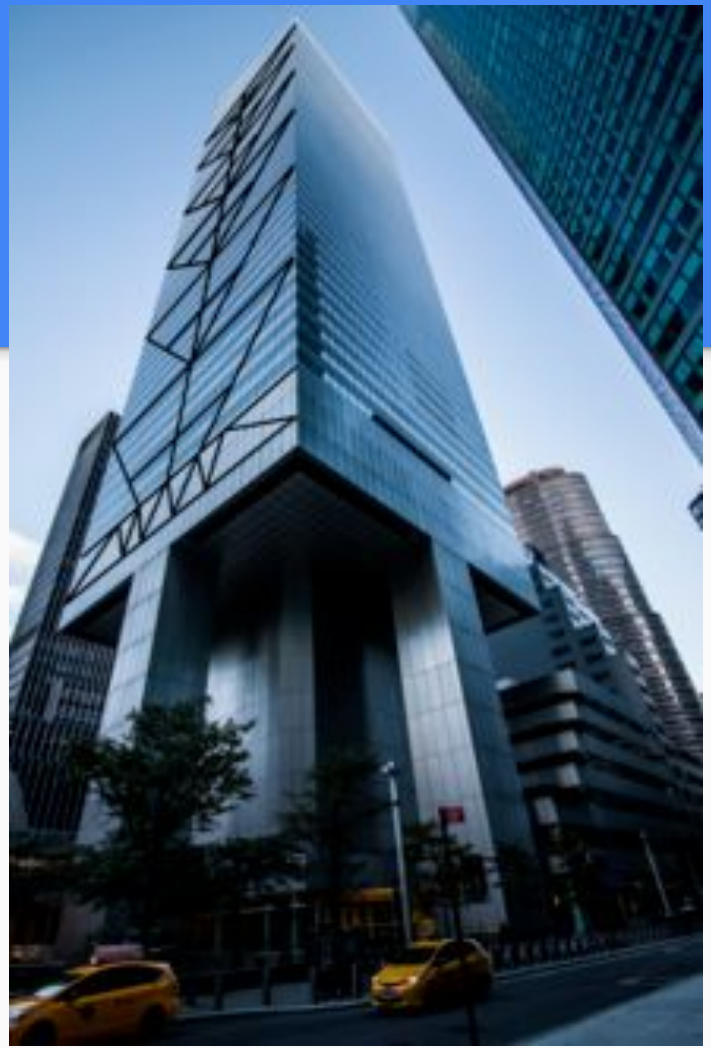
Iron ring origin

Citicorp Building NYC 1978

Building did not fall down!

Design was fixed after building was built.

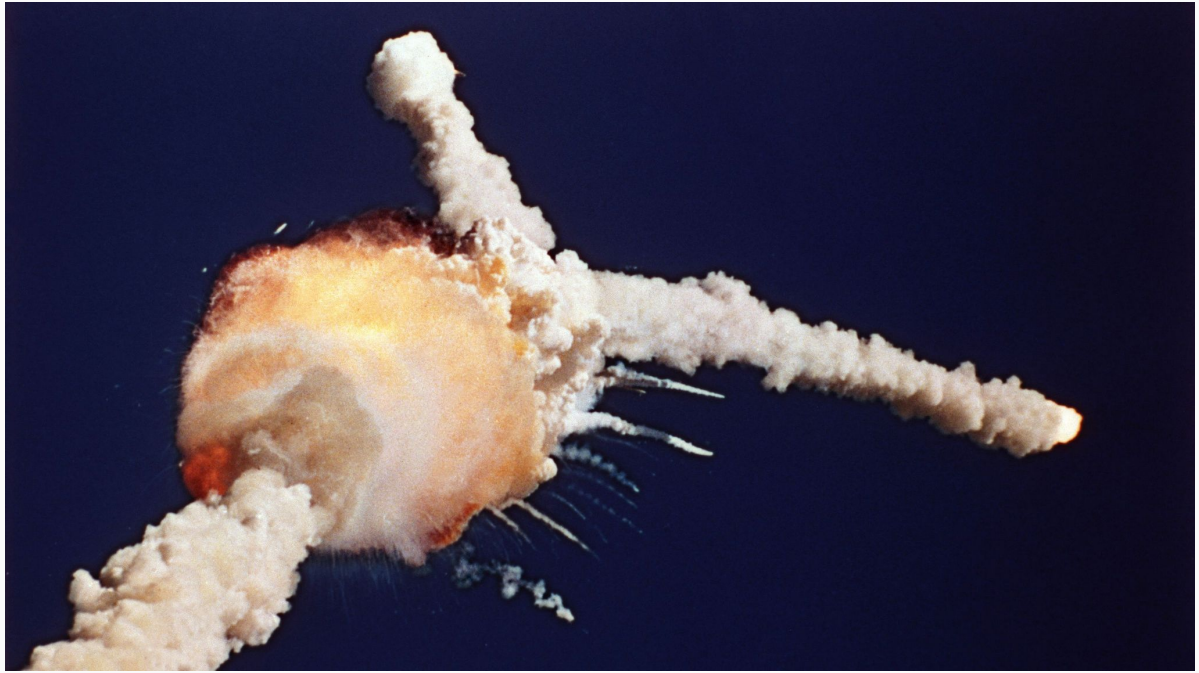
Original design would have failed in high wind.



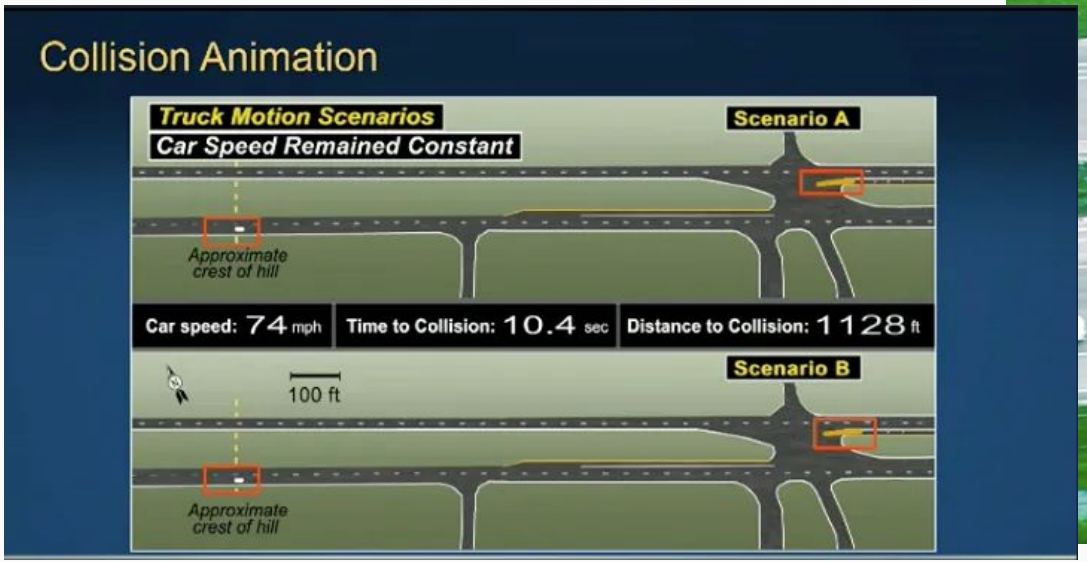
Space Shuttle Challenger 1986

O-rings froze

Fuel leaked



Tesla Autopilot 2019



What do we ***learn*** from post-mortems?

What do we learn from post-mortems?

- Inadequate team organization
- Inadequate team processes
- Inadequate training
- Inadequate communication
- Inadequate load models/expectations
- Inadequate technology/materials
- Inadequate data management & backup

Rarely individual mistakes.

Often multiple interacting issues.

GitLab 2017 Database Outage

GitLab Outage, January 31, 2017

- Repos and wikis were down, but not lost
- Lost 7 hours of customer database data
  - projects, comments, user accounts, issues and snippets
  - About 5,000 projects, 5,000 comments and 700 new user accounts

Architecture and Backup Process

- Production db replication:
  - primary + secondary
  - live replication process from primary to secondary
  - failover capability to secondary in case primary fails
- Backup:
  - pg_dump to AWS S3 every 24 hours
  - LVM snapshot from production to staging every 24 hours
  - server disk snapshots every 24 hours

3 things at the same time …

1. Trying to add load balancer to db engine in staging environment
2. Bot/spam attack (increased db load)
3. Admin mistake to delete employee account (added to db load)

Timeline

- 17:20 UTC
  - snapshot of prod DB to staging environment
  - working on DB upgrade in staging
- 19:00 UTC
  - spam/bot attack increases load on prod DB
  - background job erroneously removing employee account increases load on prod DB
- 23:00 UTC
  - prod DB starts to fail due to load …
- 23:30 UTC … attempting recovery …

Attempting replication as primary DB melts

- pg_basebackup didn't work …
  - max_wal_senders bad value
  - max_connections bad value
- pg_basebackup behaviour undocumented …
  - Not in GitLab staff runbook …
  - Not in PostGRES vendor documentation …
- mistake made while trying to restart the DB replication process …
  - Loss of 300 GB of production data

Both primary and secondary prod DBs are now broken … need backups …

Broken backup recovery procedures …

- pg_dump to AWS S3 hadn’t been working for a while …
  - The script ran, but just produced an error with no real output …
  - The cronjob error emails about this weren’t being delivered …
  - Nobody had manually checked that this db backup script wasn’t working …
- Azure disk snapshots …
  - Only used for NFS servers …
  - Not used for db servers …
- LVM snapshots …
  - These were working. Remember the snapshot to staging?
  - But staging was on cheap, slow storage: 18 hours to copy back to production
  - Reconfigure webhooks

Publication & Transparency

- Thousands of customers affected
- Twitter
- Livestream YouTube of recovery efforts (peak of 5000 viewers)
- Public Google Doc with working notes
- Post-mortem report https://about.gitlab.com/blog/2017/02/10/postmortem-of-database-outage-of-january-31/

What did we observe?

- Unexpected load from two sources
- Live replication process got overwhelmed
- Mode confusion: didn’t see difference between primary + secondary
- Team playbooks were missing important information
- Backup procedure wasn’t working + notifications weren’t working
- Restore procedure was untested — team wasn’t organized to do this

Improving Teamwork

- **Organization**:
  - who owns backup + restore?
  - new position to be responsible for data durability
- **Processes**:
  - Scripts (instructions for machines)
  - Playbooks (instructions for people)
- **Mode Confusion:**
  - clearer indicator of current environment in PS1 prompt variable

Conway's Law 1968:
A software's architecture reflects the social structure of the organization that created it.

Modern corollary: change the social structure to align with the software architecture you want to create.

Other Architectural + Process Improvements

- Load balancing
- Connection pooling
- Observability (Prometheus)
- Automatic testing of recovery procedures

Previous architecture/approach was designed to mitigate machine failure.

New architecture/approach has more focus on load, observability, durability, recovery.

Consider when making architectural decisions

- What is the rate of change in the data?
- What is the rate of change of the code? Of the libraries?
- How long will the system operate before replacement?
- Load: expectations, calculations, testing.
- Chaos testing
- Observability of the system (like feedback control!)
- Social structure of the organization(s) creating the system(s)

A Software Engineering Definition ~1970
Multi-person development of multi-version programs.

A Software Engineering Definition ~2020
Programming integrated over time.