

Reliable Software Systems (CS 598 TXU)

Spring 2019

Tianyin Xu

Course Info

- **Tianyin Xu**
 - Web: <https://tianyin.github.io/>
 - Email: tyxu@illinois.edu
 - Office hours: Tu/Th 5:00 – 6:00 pm
4108 Siebel
- **No TA**
- **Course web page (w/ tentative schedule)**
 - <https://cs598txu-uiuc.github.io/spring19/>
- **You should be on Piazza already.**

\$ Whoami

- I'm working on software and system reliability.
- I graduated from UC San Diego in 2017.
 - I worked on hardening cloud and datacenter systems against misconfigurations.
- I worked at Facebook on dealing with datacenter-level failures before joining UIUC.
 - I gained 20 lbs eating free food there.
- I applied twice for grad school.
 - I failed the first time.

\$ Whoareyou

- What are you (or will you be) working on?
- What you want to learn from the class?
- Anything you'd like to share.
 - What's the coolest thing you did in new year holiday?

Goals and Non-goals

- **Goals**

- Explore range of current problems and tensions in software/system reliability
- Understand how to identify reliability issues in your own research and how to address them
- Figure out if software/system reliability is an area of interest for you
- Get feet wet in software/system reliability research (mini research project)

- **Non-goals**

- Review of basic OS and distributed system concepts
 - Read a textbook or take CS 423 and 425

Goals and Non-goals (cont.)

This is a course of

“discussing research” + “doing research”

(you will be evaluated based on how well you discuss papers and how good your research is)

The work in this class

- **Reading**
 - 4 papers per week and 2 reviews
 - 10% of your grade is reviews
- **Discussion in class**
 - You will lead the discussion of 1 paper.
 - 10% of your grade is participation
- **Project**
 - This is the main purpose of the class (80% of your final score)

No midterm or final exam.

Readings

- **There is no textbook for this class.**
 - We will read a bunch of papers and articles.
- **We will see two types of papers.**
 - one from industry about state-of-the-art practices
 - one from academia about novel ideas/proposals
- **You are required to read two papers before the class by answering and asking questions.**
 - **You are required to submit them to me.**
 - The review forms will be sent to you via Piazza.
 - **due 11:59pm Mon/Wed**

How to read a research paper

1. What are **motivations** for this work?
2. What is the proposed **solution**?
3. What is the work's **evaluation** of the proposed solution?
4. What is your analysis of the identified problem, idea and evaluation?
5. What are the **contributions**?
6. What are **future directions** for this research?
7. What questions are you left with?
8. What is your take-away message from this paper?

Topics we will be covering

- Cloud systems and availability
- Understanding root causes
- Bug detection
- Software testing
- Chaos engineering
- Load tests and drain tests
- Monitoring
- Recovery
- Tracing
- Diagnosis
- **I'm open to more topics... got any?**

Discussion

- **You will be leading the discussion of one paper**
 - The slides will be available for you.
 - Feel free to add / modify or do your own slides.
 - Better to pick papers related to your project or interests.
 - I will send out the presentation signup form via Piazza.

Projects

- A **research** project related to the broad definition of software/system reliability.
 - In a group of **2**
 - If you can't find partner(s) we can try to help you.
 - If you have strong reasons to do a project in a team of more than 2 students, talk to me.
- **Please try to form groups this week.**
 - Send me an email by the end of this week identifying who is in your group

Most projects will fall into the category of:

- Most projects will fall into the category of:
 - **Analysis**: evaluate the reliability of a system of interest
 - **Study**: study a type of faults/errors/issues, discuss the possible ramifications, mitigations, etc.
 - **Measurement**: measure some aspect of reliability related data, characterize it, explore its limits, etc.
 - **Design/Implementation**: design and/or build a new system that addresses a problem in a new way
- My 2 cents about doing study-based projects.
 - It's not a number game.

Project Timeline

- **End of Week 3: Submit project proposal**
 - A well-defined research problem and feasible solutions.
 - Show the feasibility by concrete examples, datasets, and tools for system building.
 - **I encourage you try me the idea before deciding on the project.**
- **End of Week 7: Submit Checkpoint 1 report**
 - Show your system/tool prototype and preliminary results.
 - Your prototypes should be able to work with your motivating examples.
- **End of Week 11: Submit Checkpoint 2 report**
 - (At this point, you are expected to build your system/tool and start evaluation)
 - Describe the detailed evaluation plan in your report.
- **4/25 and 4/30 (in class): Final project demo (15 min)**
- **5/1 Fri 23:59pm: Submit final project report (6 pages)**

Exploring your project **NOW!**

- **Initial project proposals due 2/1 (one page)**

- What you plan to do
- Why is it interesting
- How you'll do it (feasibility)
- What you're not sure about



Problem Statement

Project Grading

- **A to A+:** significant results and publishable work;
- **A- to A:** strong results and a clear roadmap to publishable work;
- **B+ to A-:** interesting results but quite far from being significant;
- **B to B+:** a good exploration but leads to nothing;
- **B- to B:** some efforts of exploration but no conclusion.

(You should feel free to explore and fail)

- **Hope: sufficiently interesting to be real papers**
 - at least something you are proud of talking about

Where do we publish?

- OSDI/SOSP
- ASPLOS (PL and arch)
- NSDI (networked systems)
- FAST (file and storage systems)
- EuroSys (European)
- SOCC* (Cloud systems)
- USENIX ATC* (everything)
- (related) ICSE/FSE, CHI, MobiSys, SIGCOMM*, IMC, PLDI*

* I never (or failed to) publish there.

Things to think about

- **Pick good problems**
 - Why is this problem interesting or will become interesting?
 - Look at what others are doing:
 - Academic conferences: OSDI/SOSP, NSDI, EuroSys, SOCC, ATC
 - Engineering blogs and postmortems
- **Pick problems that are achievable.**
 - What resources would you need to investigate the problem? (ask if you're serious)
- **Think about how to evaluate your work**

Sample ideas and directions

- I will post a number of sample ideas on Piazza.
 - **This is not a list you must pick from!**
 - Just examples to give you ideas and make sure you understand how broad the scope is.

Questions about the project?

- I'm always here to help
 - use me well (but don't abuse me)
- Systems research requires no genius.
 - It requires understanding and experiences.

What is “reliability”?

- Merriam-Webster online dictionary:
 - **Reliability:** The quality or state of being **reliable**
 - **Reliable:** suitable or fit to be **relied** on
 - **Rely:** to be **dependent**
 - **Dependent:** **relying** on another for support
- “The ability of a system or component to perform its required functions under stated conditions for a specified period of time”

What is reliability?


- Availability, reliability, safety, security

Availability	System is available for use at any time.
Reliability	The system operates correctly and is trustworthy.
Safety	The system does not injure people or damage the environment.
Security	The system prevents unauthorized intrusions.

What is reliability?

- **Most of CS is about providing functionality**

- User interface
- Software design
- Algorithms
- Operating systems/networking
- Compilers/PL
- Vision/graphics
- Microarchitecture
- VLSI/CAD



**There are reliability
problems in all of these
domains**

- **Reliability is **not** about functionality**

- It is about how the embodiment of functionality behaves in the presence of errors and failures

Reliability is like security... somewhat

- Threat model
- Zero day
- Bugs and misconfigurations
- Fail static versus default deny
 - If you don't know what to do, do the least harmful thing
- Need to mitigate before you know what's going on
- Both conflict with velocity
- Defense in depth

* Mogul, Thinking about availability in large-scale infrastructures, HotOS, 2017.

Why reliability techniques matter?

- We are living in an unreliable world.
 - but we desire highly available services.
- Hardware breaks.
 - Assume you could start with super reliable servers (MTBF of 30 years)
 - Build a computing system with 10 thousand of those
 - Watch one fail per day
- -> Building fault-tolerance into the software.

The joys of real hardware

- **Typical first year for a new cluster:**

- ~0.5 overheating (power down most machines in <5 mins, ~1-2 days to recover)
- ~1 PDU failure (~500-1000 machines suddenly disappear, ~6 hours to come back)
- ~1 rack-move (plenty of warning, ~500-1000 machines powered down, ~6 hours)
- ~1 network rewiring (rolling ~5% of machines down over 2-day span)
- ~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
- ~5 racks go wonky (40-80 machines see 50% packet loss)
- ~8 network maintenances (4 might cause ~30-minute random connectivity losses)
- ~12 router reloads (takes out DNS and external vips for a couple minutes)
- ~3 router failures (have to immediately pull traffic for an hour)
- ~dozens of minor 30-second blips for DNS
- ~1000 individual machine failures
- ~thousands of hard drive failures
- slow disks, bad memory, misconfigured machines, flaky machines, etc.
- Long distance links: wild dogs, sharks, dead horses, drunken hunters, etc.

Why reliability techniques matter (cont.)?

- **Software has (many) bugs.**
- **Some bugs could be hard to find.**
 - concurrency bugs
 - not exposed in every simple run
 - manifested in a larger scale
 - latent bugs
 - only manifested under certain circumstances
- **-> Prioritize critical bugs and correctness properties**
- **-> Recover from bugs**
 - revert buggy code changes
 - remove the triggering conditions
- **-> Formal verification**

Consequences of system failures

- **Corrupted:** committed data that are impossible to regenerate, are lost, or corrupted
- **Unreachable:** service is down or otherwise unreachable by the users
- **Degraded:** service is available but in some degraded mode
- **Masked:** faults occur but are hidden from users by the fault-tolerant software/hardware mechanisms
 - Why is this still bad?

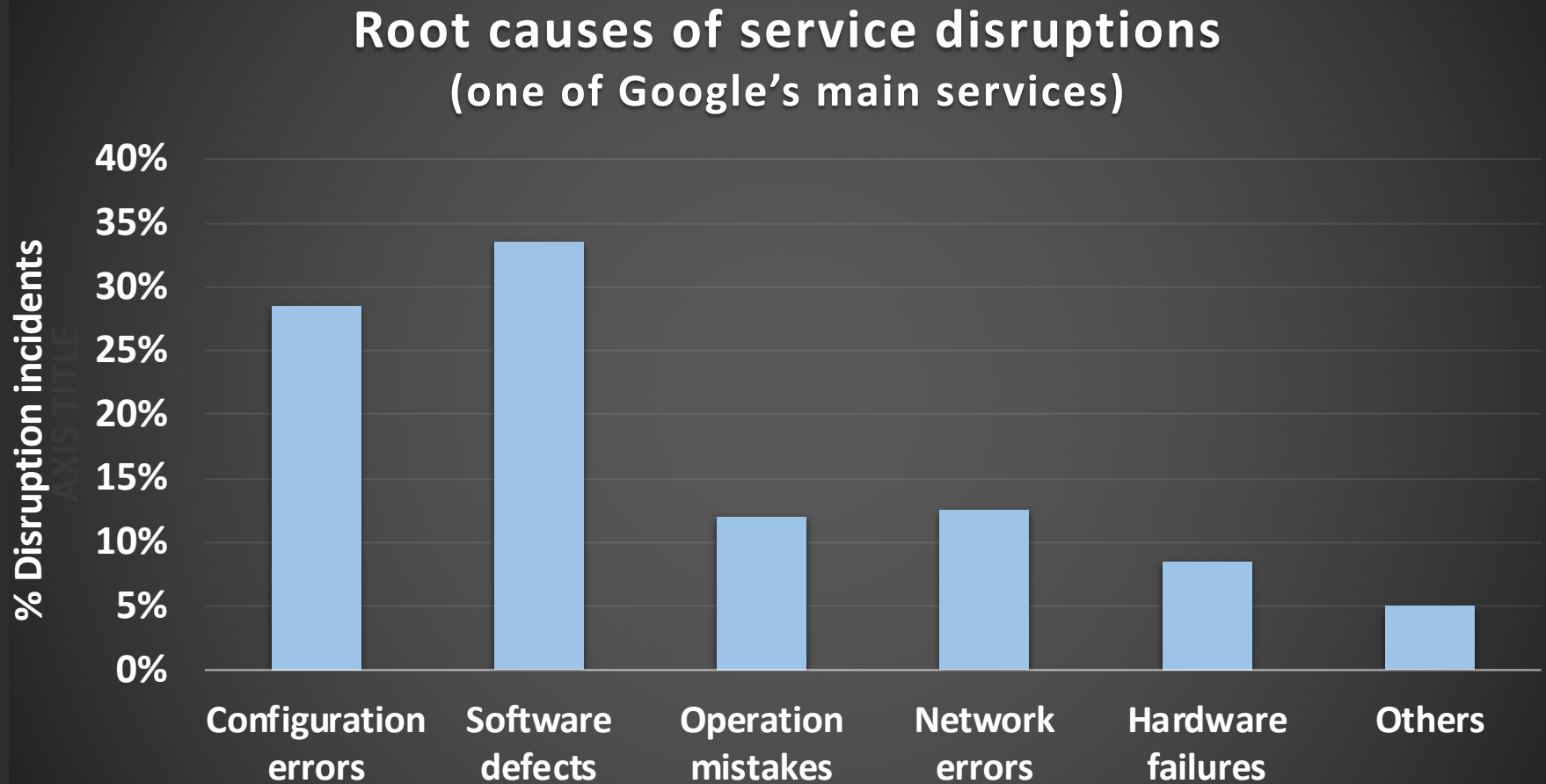
Redundancy for fault tolerance

- **Fault tolerance “is just” redundancy.**
 - Replication for component failures
 - space
 - Timeout and retry for message loss
 - time
 - Write ahead log and data pages
 - representation
 - Error correcting code
 - mathematical
 - K-version programming

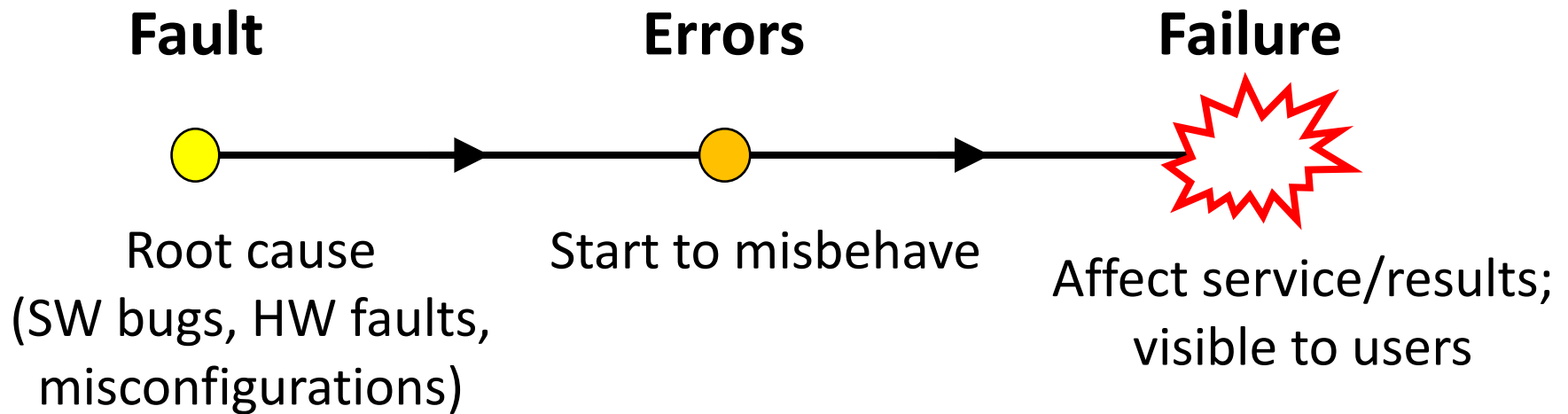
Why do systems still fail? (despite all the redundancies)

- Not enough redundancies.
 - failure of entire datacenter
- Redundancies are also affected.
 - corrupted
- Redundancies are faulty (but unnoticed).
 - error handling code is often buggy.
- (Often Hidden) SPOF
 - e.g., failure recovery chain

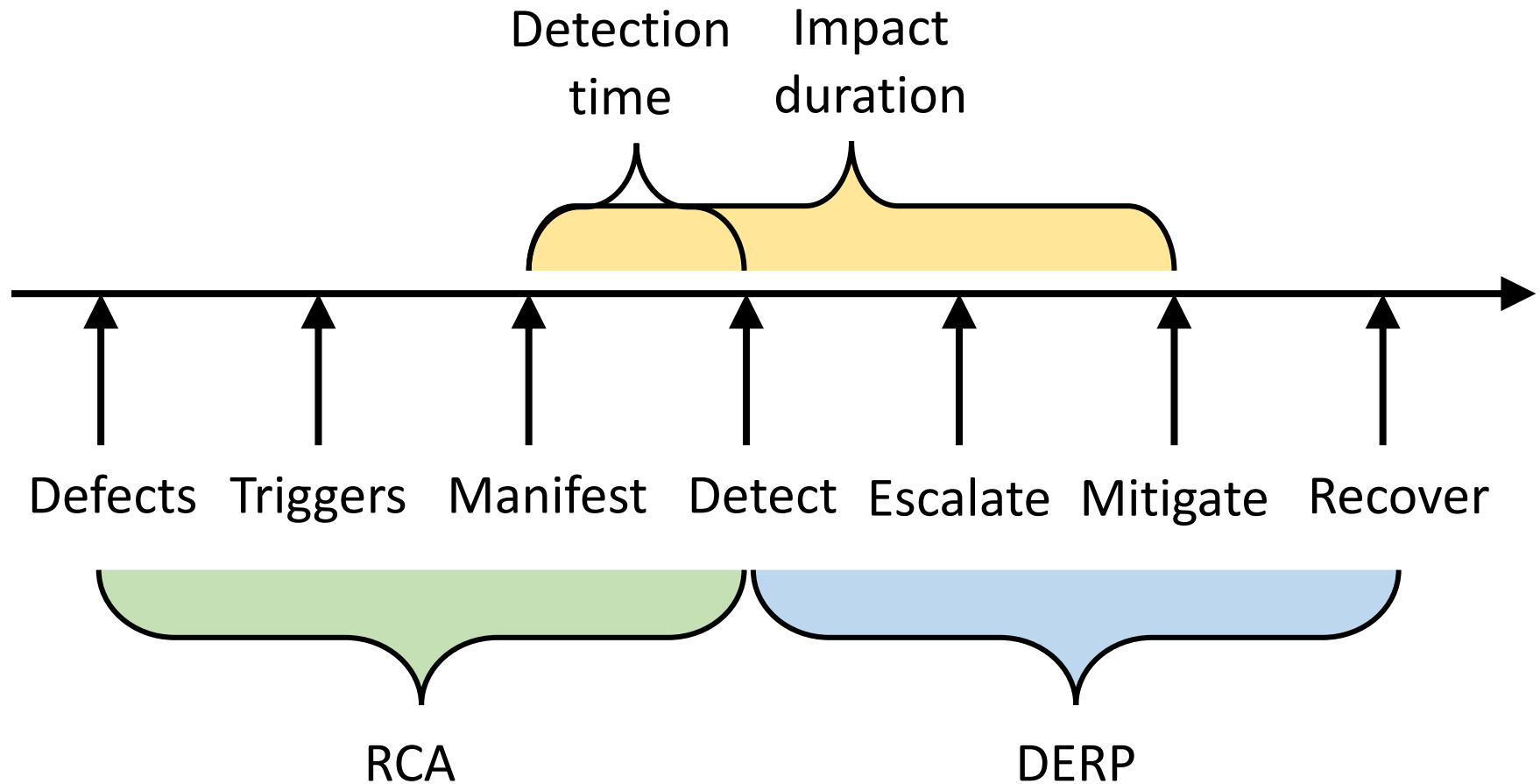
How do systems fail in the field?



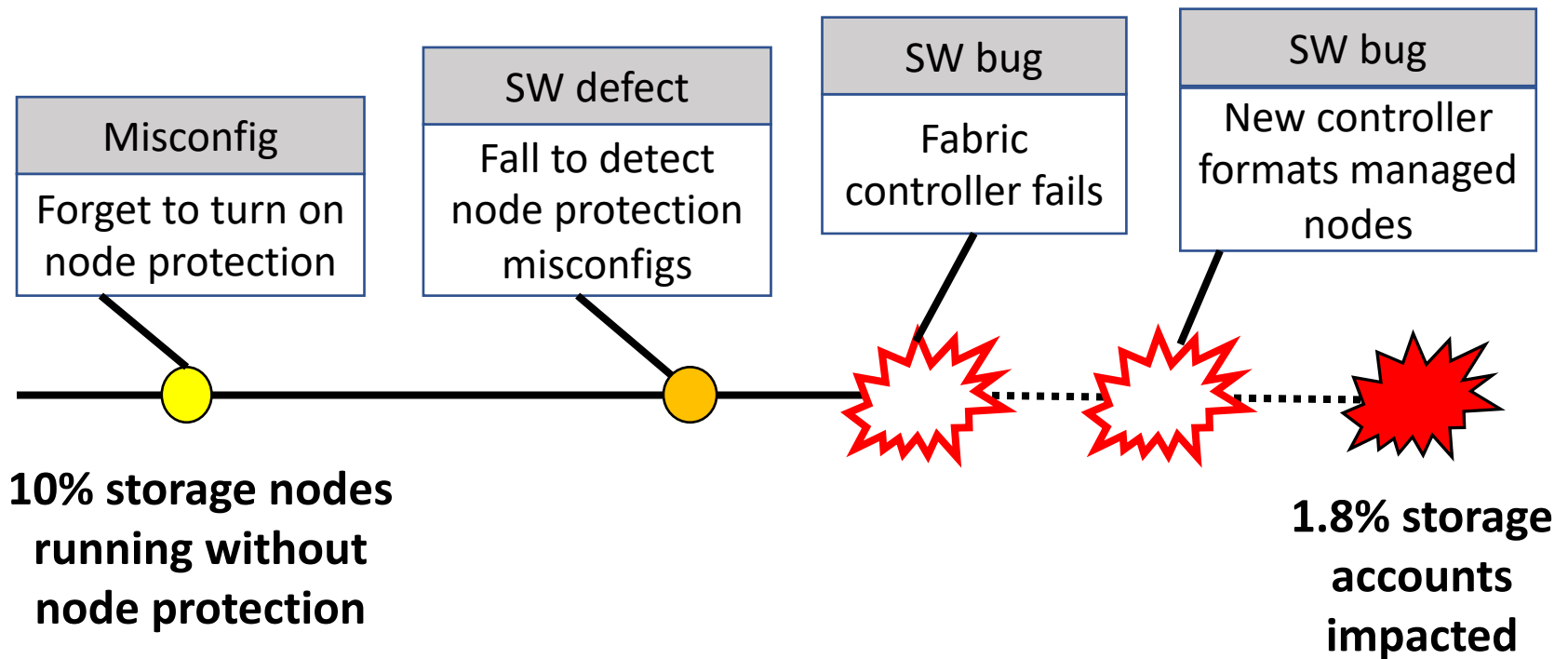
The fault-error-failure model



The event chain mode



Case Study: Microsoft Azure's 12/28/12 disruption in US South



<https://azure.microsoft.com/en-us/blog/details-of-the-december-28th-2012-windows-azure-storage-disruption-in-us-south/>

How to study reliability?

- **Understand how systems break in the past**
 - postmortem and SEV reviews
 - bug study and analysis
 - field failure data
- **Prevent similar problems in the future**
 - understand failure modes
 - examine deficiency of existing mechanisms
 - learn from your own or others' mistakes

How to study reliability?

- **Examine how systems behave in the presence of realistic faults and errors**
 - thought experiments
 - reliability review
 - fault injection
 - fuzzing
 - production experiments
 - DiRT and Game Day

How to evaluate new reliability methods/tools?

- Mathematical proofs
 - Formal verification
- Error injection
 - Inject errors and see how systems behave
- Evaluation with historical data
 - Misconfiguration detection
- Finding defects in existing systems/software
 - Bug detection

That's it for today

- **Questions?**
- **Next class:** Data Center, Systems and Availability
- **Watch:**
 - Dean, [Building Software Systems At Google and Lessons Learned](#)
- **Read**
 - Barroso et al., “[The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines](#)” (Chapter 1 and 7)
 - Gunawi et al., “[Why Does the Cloud Stop Computing? Lessons from Hundreds of Service Outages](#)”