Reliability of Cloud-Scale Systems (CS 598)

Fall 2018

Tianyin Xu

Reliability of Cloud-Scale Systems
Reliability of Large-Scale Systems

(CS 598)

Fall 2018

Tianyin Xu

Reliability of Cloud-Scale Systems
Reliability of Large-Scale Systems
Reliability of Computer Systems (CS 598)

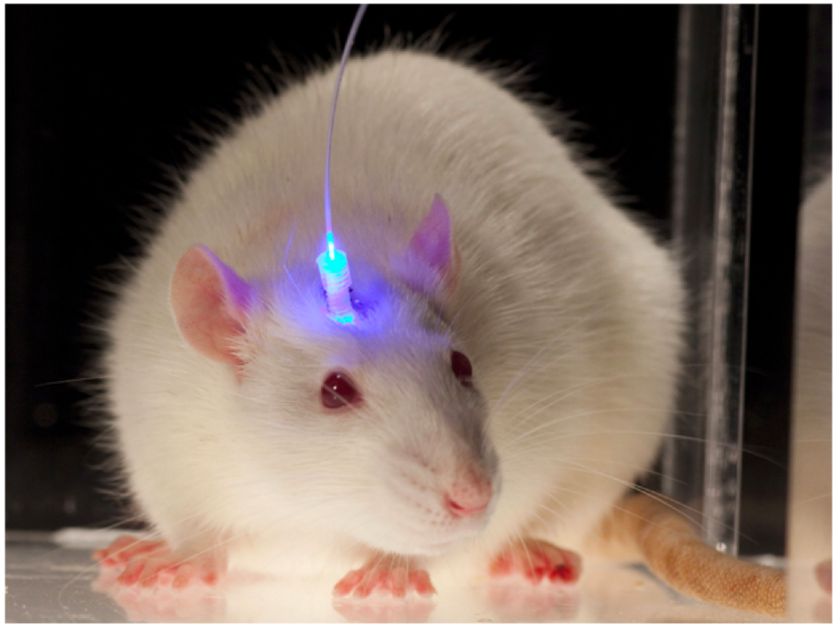
Fall 2018

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
- **TA: Ranvit Bommineni**
  - bommine2@illinois.edu
- **Course web page (w/ in-progress schedule)**
  - https://tianyin.github.io/cs598-fa18/

This is a class in "progress."



You

Me

Let’s know each other.

- Who are you?
- 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 the summer?

About me

- 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 configuration errors.
- 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.

Goals and non-goals

- **Goals**
  - Explore range of current problems and tensions in software and 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

Readings

- **There is no textbook for this class.**
  - We will read a bunch of papers and articles.

- **We will discuss two papers in each class.**
  - one from industry about state-of-the-art practices
  - one from academia about novel ideas/proposals
  - (there are topics where this does not apply)

- **You are required to read both of the two papers before the class.**
  - **You are required to write reviews for one of them.**
  - 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?

W. Griswold, How to Read an Engineering Research Paper, https://cseweb.ucsd.edu/~wgg/CSE210/howtoread.html

Topics we will be covering

- Understanding failure root causes
- Observability
- Troubleshooting
- Failure recovery
- Finding bugs
- Testing production systems
- Reliability auditing
- Formal verification
- **I'm open to more topics… got any?**

The work in this class

- **Reading**
  - 4 papers per week and 2 paper reviews
  - 10% of your grade is reviews
- **Discussion in class**
  - The papers and concepts we have covered
  - 10% of your grade is participation
- **Project**
  - This is the main purpose of the class (80% of grade)

No other homework, midterm, or final.

Projects

- **Some kind of research project related to software or system reliability**
  - Best in a group of 2-3
  - If you can't find partner(s) we can try to help you
- **Please try to form groups next Wed (9/5)**
  - Send me an email by 9/5 identifying who is in your group
- **Initial project proposals due 9/14 (one page)**
  - What you plan to do
  - Why is it interesting
  - How you'll do it
  - What you're not sure about

Problem Statement

Projects (cont.)

- **Checkpoint report #1: due 10/12 (one page)**
  - Describe your progress
  - Explain any changes you make to your proposal (if any)
  - Examples or cases
  - Concrete plans of what you will need accomplish in the remaining weeks
- **Checkpoint report #2: due 11/16 (one page)**
  - Similar as #1
  - You are required to include primary results at CP #2.
- **Ultimately 6 pages and a short talk (10-15mins)**
- **Hope: sufficiently interesting to be real papers**
  - at least something you are proud of talking about

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

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

Random ideas

- On the class Web page
  - **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

$\Bigg\}$ **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

Why does it 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
- Fault-tolerant software is inevitable.

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 does it 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**

Service-level 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?

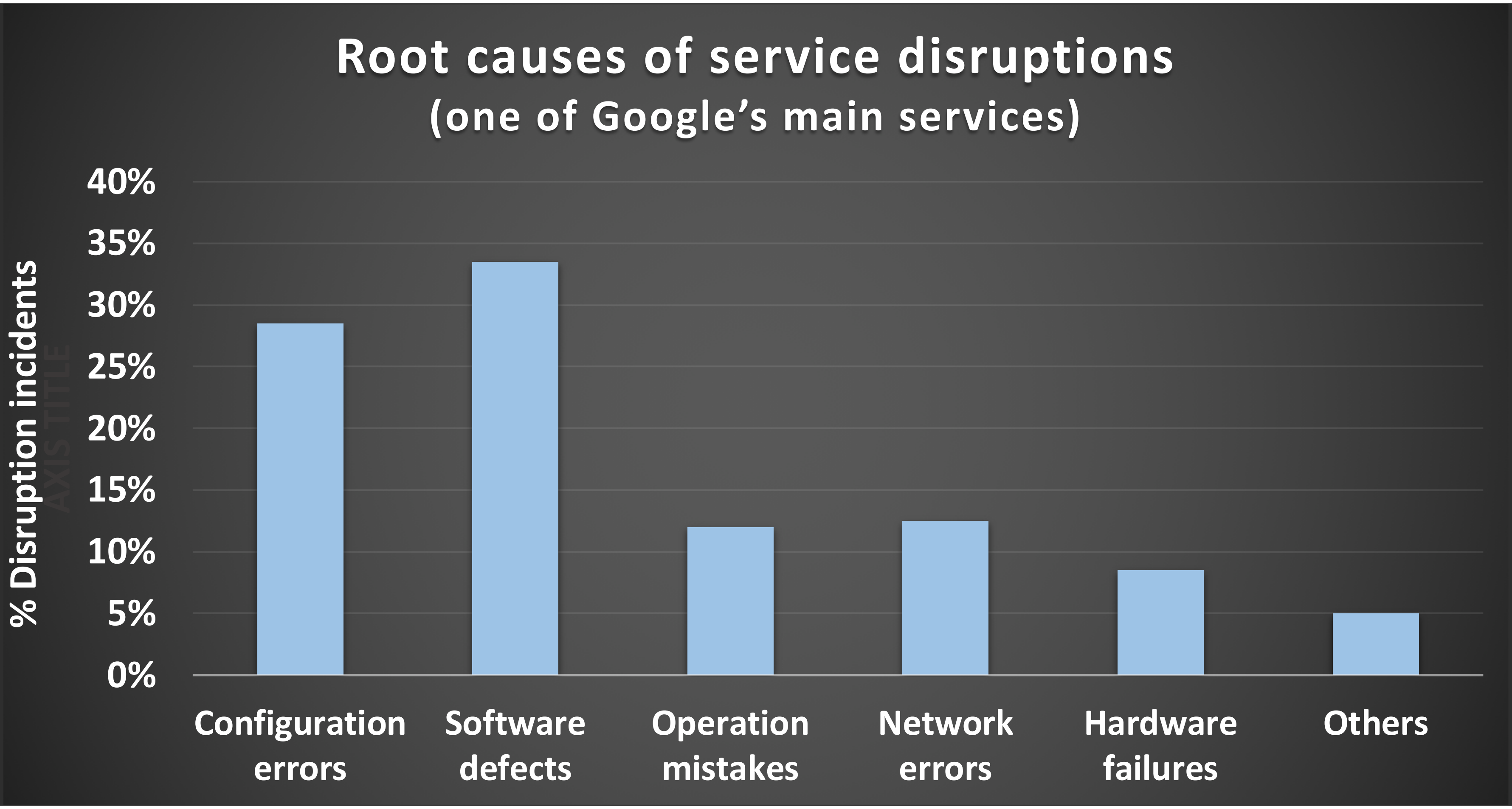
Redundancy for fault tolerance

- **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?



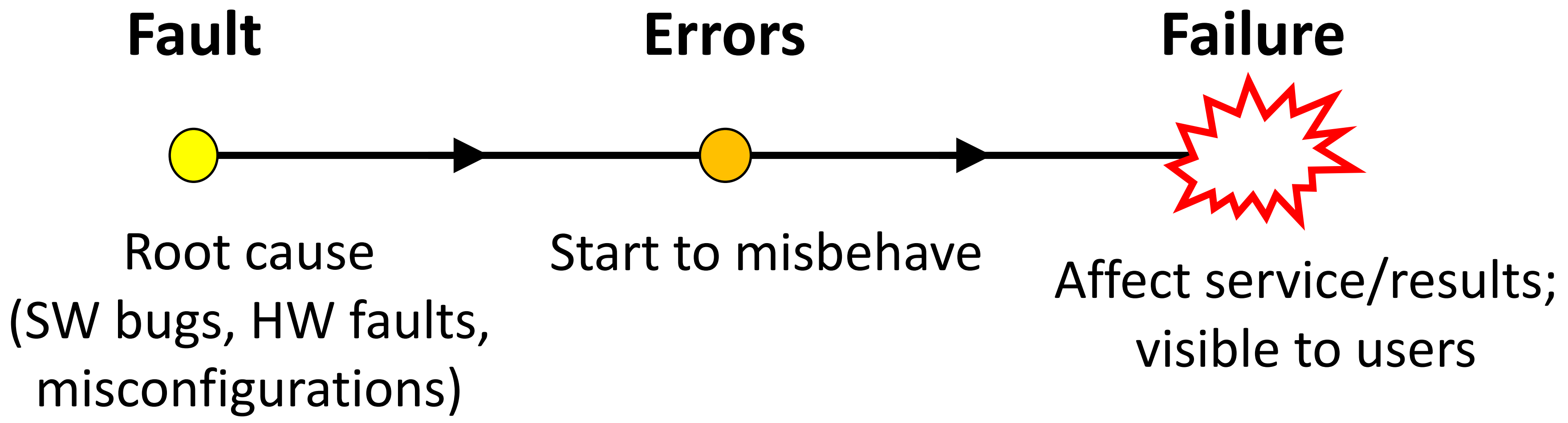
L. A. Barroso, J. Clidaras and U. Hölzle, The Datacenter as a Computer: An Introduction to the Design of Warehouse Scale Machines (2nd Edition), Morgan & Claypool Publishers, 2013.

The case of configuration errors

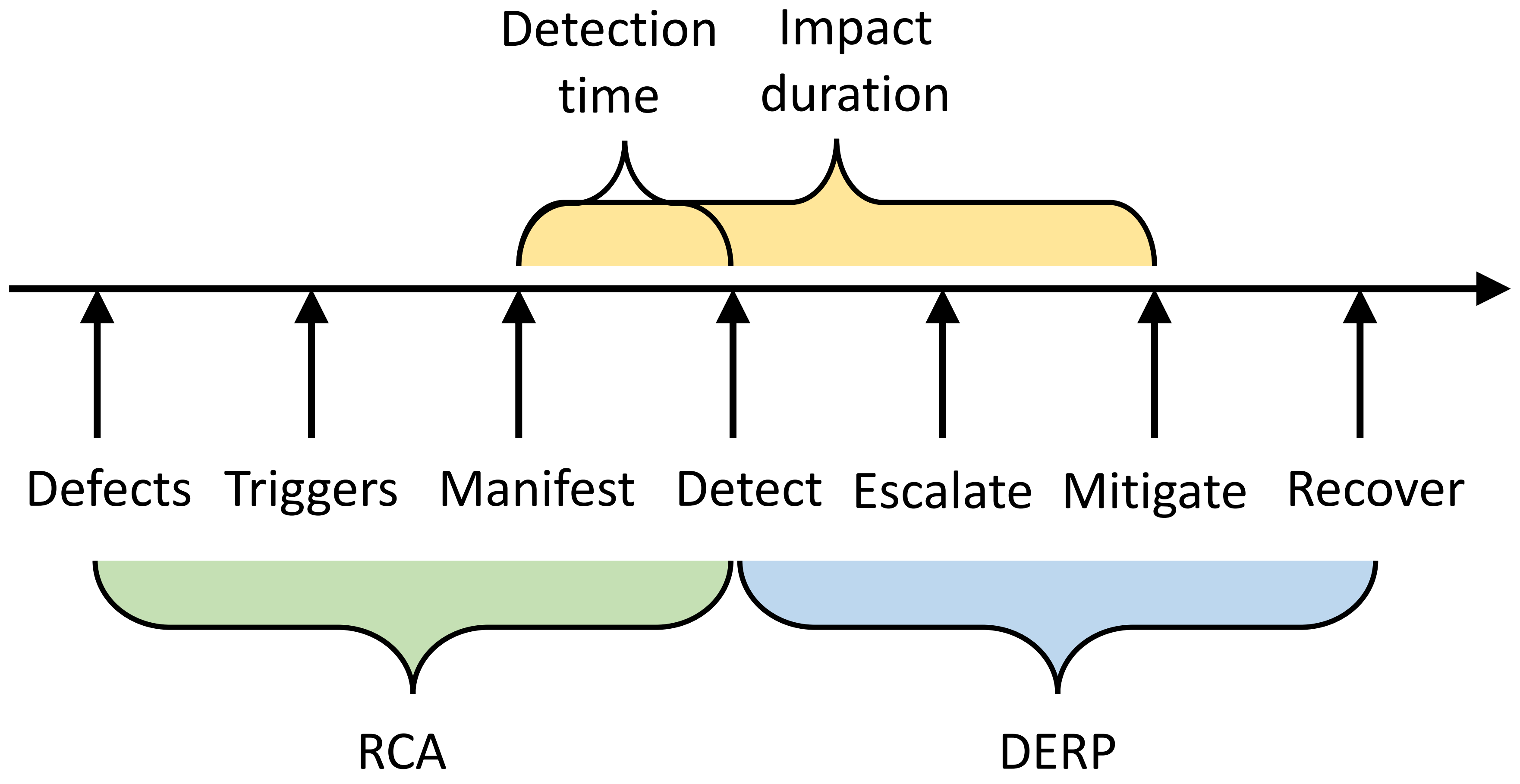
- Configuration errors are often epidemic.
  - propagated to a large number of software instances
  - invalidate traditional fault tolerance

- Fault tolerance & recovery are often misconfigured.
  - "Faulty **failover** configurations turned a 10 minute outage into a 2.5 hour ordeal."
  - "Misconfigured **backup** DNS (used upon attacks) made LinkedIn inaccessible for half a day."
  - "Misconfigured data **protection** allowed a bug to wipe out 10% of the storage nodes."

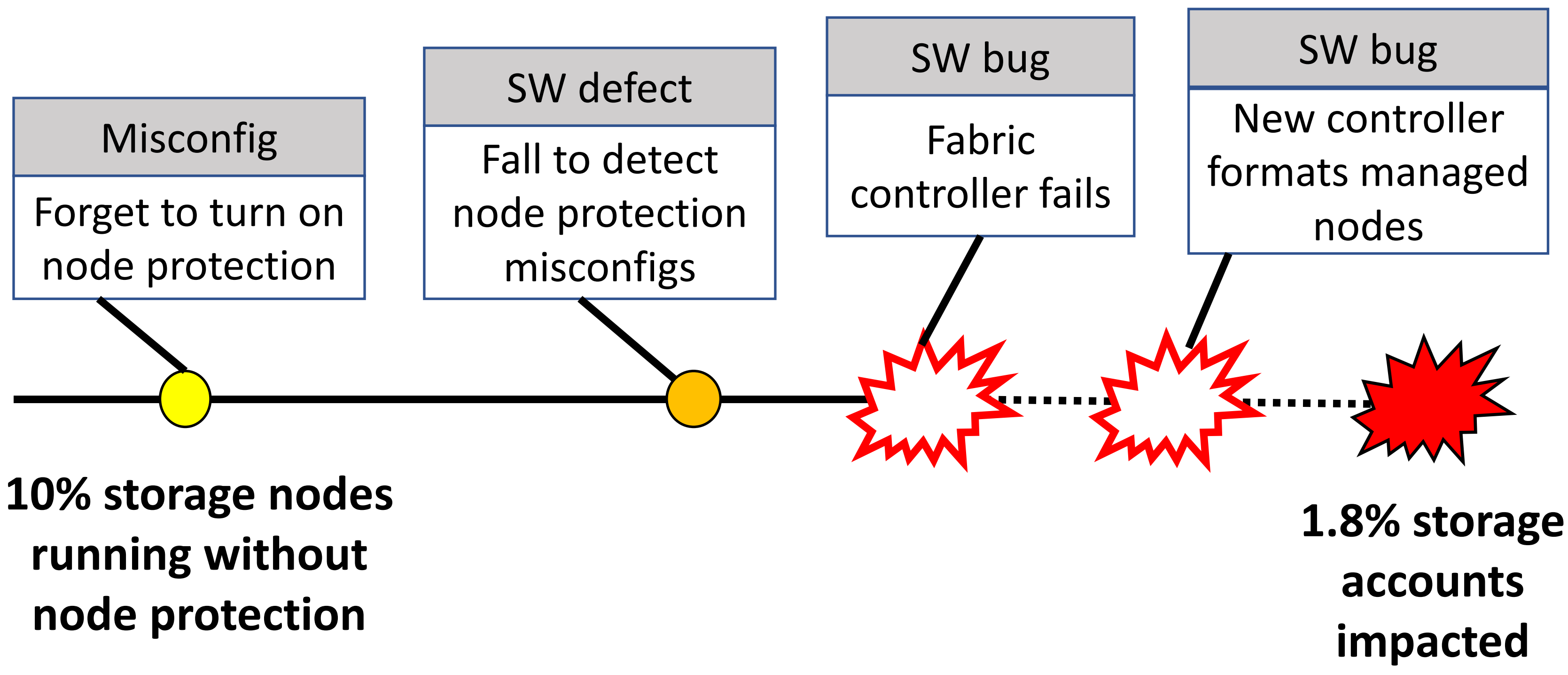
The fault-error-failure model


Fault
Errors
Failure
Root cause
(SW bugs, HW faults, misconfigurations)
Start to misbehave
Affect service/results; visible to users

The event chain mode



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


Misconfig
Forget to turn on node protection
SW defect
Fall to detect node protection misconfigs
SW bug
Fabric controller fails
SW bug
New controller formats managed nodes
10% storage nodes running without node protection
1.8% storage accounts impacted

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
- **Accept and tolerate the failures**
  - based on failure characteristics

How to study reliability?

- **Examine how systems behave in the presence of realistic faults and errors**
  - thought experiments
    - reliability review
  - fault injection
  - fuzzing
  - testing in production
    - 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: Availability

- Read:
  - Gunawi, et al’s “Why Does the Cloud Stop Computing? Lessons from Hundreds of Service Outages”
  - Treynor et al’s “ The Calculus of Service Availability”