Week 1: Introduction & Logistics

18-749: Reliable Distributed Systems

Priya Narasimhan
Professor
Electrical & Computer Engineering Department
Carnegie Mellon University

Overview of Lecture

- Introduction to the course
  - Logistics, grading criteria, policies
- Course support
  - Canvas, Slack
- Expectations
- Brief introduction to enterprise technologies

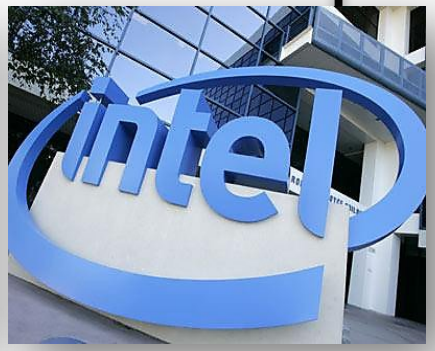
India

Professor, Carnegie Mellon

CEO & Founder, YinzCam, Inc.

Zambia

PhD, UC Santa Barbara
CTO, Eternal Systems



Director, Intel Labs
Pittsburgh

Director, Intel Science
& Tech Center in
Embedded Computing

YINZCAM

My Industry/Teaching Experience

- Industry experience
  - Director of Intel Labs Pittsburgh, 2010-11
  - Director of CMU's Intel Science and Technology Center in Embedded Computing, 2012 onwards
  - Director of CyLab Mobility Center, 2007-2009
  - Industrial experience
    - CTO and Vice-President of Engineering, Eternal Systems, Inc.
    - Founder and CEO, YinzCam, Inc.
    - Established an industrial fault-tolerance standard
- My teaching
  - 18-349: Fall 2002/2003/2005/2007/2008/2009/2011/2012/2013
  - 18-549: Embedded Systems Design (Spring 2007/2008/2009/2010/2011/2012/2013)
  - 18-738: Sports Technology (2009/2015/2016/2017/2018/2019/2020/2021/2022/2023/2024)
  - 18-749: Building Reliable Distributed Systems (2015/16/17/18/19, 2020/21/22/23/24)

Director, AthTetech Lab

Development of sports technologies. Use of AI in sports.

My Research Experience

- Ph.D. in distributed transparent fault-tolerance — became the Fault-Tolerant CORBA Standard
- Advise Ph.D. students in cloud computing, failure diagnosis, large-scale analytics, mobile edge computing, sports technology
  - Lancaster Best Ph.D. Dissertation Award, Alfred Sloan Fellow, NSF Career Award, Carnegie Science Award, Heinz History Award, CIT Teaching Award, Lutron Spira Teaching Award
- 150+ research publications in IEEE, ACM, USENIX conferences and journals
  - Showed that we could do data-driven failure diagnosis for production Hadoop clusters without requiring any training data (peer-comparison algorithm)—Best paper award
  - Showed that we could improve latency in processing large amounts of video at the edge through crowd-sourced processing on mobile devices—Best paper award
  - Developed algorithms for data-driven instant replays without human intervention
- Athletech Lab: Focused on sensors, systems, distributed computing to do with esports

Course Logistics

- Slack for all discussions and announcements
  - #always-learning — use for sharing incidents, failures, useful material
  - #lectures — use for sharing lectures or asking questions about lectures
  - #the-project — use for discussing the project (starting next week)
  - #exams — use for discussing quizzes and exams
  - #syllabus-schedule — syllabus, lecture schedule, project milestones
- Google Drive for all lectures (electronic form)
- Canvas for all grading
- Lectures (once a week)
  - Mondays, 5-650pm ET (with a break in between)
  - Project Milestones: Wednesdays, no meeting needed

How to reach me

- If you have any questions
  - Send me a DM (direct message) on Slack
  - I will check the course Slack daily, even on non-teaching days

- For any and all emergencies
  - My cellphone number is 412-600-1165

Exams

- Open-book, open-notes
  - Bring printed copies of your lecture slides with you
  - Bring any printed materials you like
- Mid-term exam (100 minutes)
  - Held in regular class hours
  - Covers all the material taught from the beginning until the mid-term
- Final exam (100 minutes)
  - Held in regular class hours
  - Covers all the material taught from the beginning until the final exam

Cheating and Plagiarism

- Your code may be (and will be) compared against code submitted by other groups and/or code submitted by groups in prior years
- CMU cheating policies
  - Please read and understand http://www.cmu.edu/policies/documents/Cheating.html
- Cheating penalty
  - An “R” grade in the course and notification to the department
- Mid-term exam, final exam indicate individual ability
  - Any collaboration in these aspects will be considered cheating
- Copyright warning
  - Please do not post or otherwise redistribute materials distributed in class

Grading Criteria

- Breakdown of grading
  - Midterm exam: 30% of your grade
  - Final exam: 40% of your grade
  - Project: 25% of your grade due to team contributions (five-person project) + 5% due to peer review

- Late Policy
  - Last day to turn in everything is the last day of class
  - 10% of the specific category of grade per-day penalty (including Saturday and Sunday)

The Project (30% of your Grade)

- Build an industrial-grade fault-tolerant system
  - Build fault detection via heartbeats
  - Build two styes of replication (active and passive)
  - Build a lightweight totally-ordered protocol for consensus
  - Build checkpointing and logging
  - Build state transfer and recovery
  - Measure the effectiveness of different configurations of fault-tolerance
  - Implement this in Java or C/C++
  - Multiple processes running on a single machine (and communicating via TCP/IP)

- Project approach
  - Recitation: we will spend 50 minutes describing the project, milestones, etc.

Expectations

- You are expected to
  - Attend lectures
  - Know how to program in Java or C/C++
  - Get the project done by the deadlines specified
  - Participate in the project-management and presentation of the demos
  - Participate in the meetings with course staff regarding the project
- Learning material in this course requires participation
  - Questions are welcome and appreciated
  - If you see something that is a problem, you need to tell us right away
- Feedback
  - Talk to us if you have any concerns regarding anything related to the course
  - You can also provide anonymous feedback anytime

Getting Help

- Course workload
  - We expect an “average” CMU student will put in 12 hours/week
- If you see yourself exceed the number of hours greatly, then
  - You might need to brush up on some background knowledge
  - You might be approaching the project in the wrong way

Learning Objectives

- High-level goals
- Understand the scientific principles and concepts behind fault tolerance in enterprise systems
- Obtain hands-on experience in fault tolerance
- Understand basic and advanced fault-tolerance concepts
- Understand, and be able to discuss and communicate intelligently about
  - Replication, consistency, nondeterminism, fault models
  - Architectures for high-availability, protocols for reliability
  - How fault-tolerance is used in industry today
  - How reliability is implemented at Netflix, Google, Amazon, Facebook, eBay
  - Forensics of major failure incidents in industry

Questions we will seek to answer

- We are focusing on fault-tolerance in enterprise systems
- Enterprise systems have databases, servers, cloud computing, storage, etc.
- How is fault-tolerance implemented?
- How are faults detected?
- What are the best practices in industry?
- How is fault-tolerance enabling new architectures, and challenging the way companies are run?
- What are the academic approaches vs. the industry approaches?

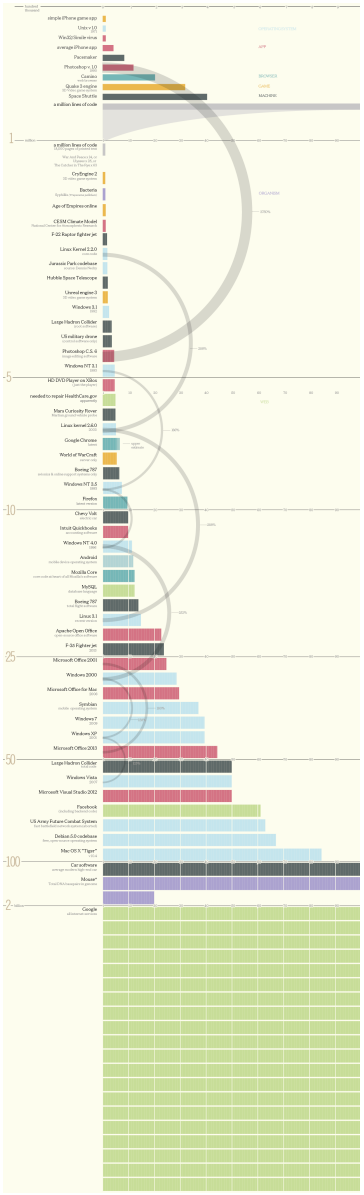
Sample Schedule (2023)

WK	DATE	LENGTH	LECTURE TOPIC	PROJECT MILESTONES	GRADING MILESTONE	GRADE %
1	8/28/2023	100 mins	**Course Overview, Expectations, Introduction**			
	8/30/2023			Start forming project groups.		
2	9/4/2023	100 mins	*No lecture -- Labor Day holiday*			
	9/6/2023			Start implementation of your client-server application with your group.		
3	9/11/2023	100 mins	**Fault Detection, Asynchronous Distributed Systems**			
	9/13/2023					
4	9/18/2023	100 mins	**Consensus, Total Order, Virtual Synchrony**			
	9/20/2023			**MILESTONE 1:** Client-server application, configurable fault-detection.	☑	5%
5	9/25/2023	100 mins	**Consistent Replication, Checkpointing, Recovery (Part 1)**			
	9/27/2023					
6	10/2/2023	100 mins	**Consistent Replication, Checkpointing, Recovery (Part 2)**			
	10/4/2023					
7	10/9/2023	100 mins	Mid-Term Exam (via Canvas)		☑	30%
	10/11/2023			**MILESTONE 2:** Active replication with single fault, and single failover.	☑	5%
8	10/16/2023	100 mins	*No lecture -- Fall Break holiday*			
	10/18/2023					
9	10/23/2023	100 mins	**Case Studies--Postmortems of Real-World Incidents**			
	10/25/2023			**MILESTONE 3:** Passive replication with single fault, primary failover.	☑	5%
10	10/30/2023	100 mins	**Network Partitioning, CAP Theorem**			
	11/1/2023					
11	11/6/2023	100 mins	**ACID, BASE, Eventual Consistency**			
	11/8/2023					
12	11/13/2023	100 mins	**Case Studies--Architectures of Netflix, Facebook, Google, Yahoo!**			
	11/15/2023			**MILESTONE 4:** Active & passive replication with recovery, checkpointing.	☑	5%
13	11/20/2023	100 mins	*No lecture -- Thanksgiving Week holiday*			
	11/22/2023					
14	11/27/2023	100 mins	**Case Studies--Architectures of Netflix, Facebook, Google, Yahoo!**			
	11/29/2023					
15	12/4/2023	100 mins	Final Exam (via Canvas)		☑	40%
	12/6/2023			**MILESTONE 5:** Entire Fault-Tolerance system for active & passive replication.	☑	10%
				TOTAL FOR GRADE =		100%

WHY SHOULD WE CARE?

Because these systems are everywhere!

- Distributed software systems are everywhere
  - Mission-critical software: defense, air traffic control
  - Financial software: stock purchases, travel reservations
  - Medical software: pacemaker, health monitoring
- Bugs in software have financial and life-threatening consequences
- Imperative to find and fix problems before they cost too dearly
- Software is one of the most complex artifacts
  - 0.5 million lines of code in a pacemaker
  - 2 million lines of code in Netscape browser
  - 2 million lines of code in Microsoft Word (27,000 lines in first version)
  - 7 million lines of code in a Boeing 777
  - 40 million lines of code in the space station
  - The more distributed the software, the higher the complexity



By the Lines of Code …

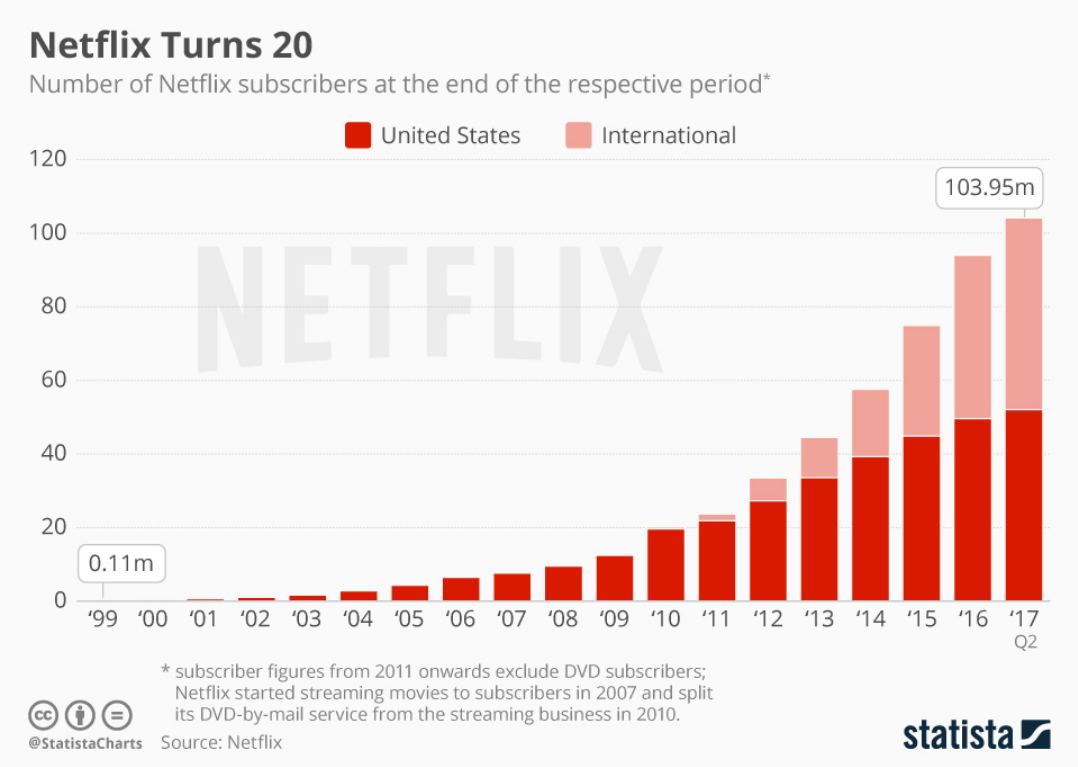
- U.S. military drone control software: 3.5 million.
- Boeing 787: 6.5 million in avionics and online support.
- Google Chrome (browser): 6.7 million (upper estimate).
- A Chevy Volt: 10 million.
- Android OS: 12-15 million lines.
- Large Hadron Collider: 50 million lines.
- Facebook: 62 million (not including backend)
- Cloud-connected car: 100 million
- All Google services: A whopping 2 billion

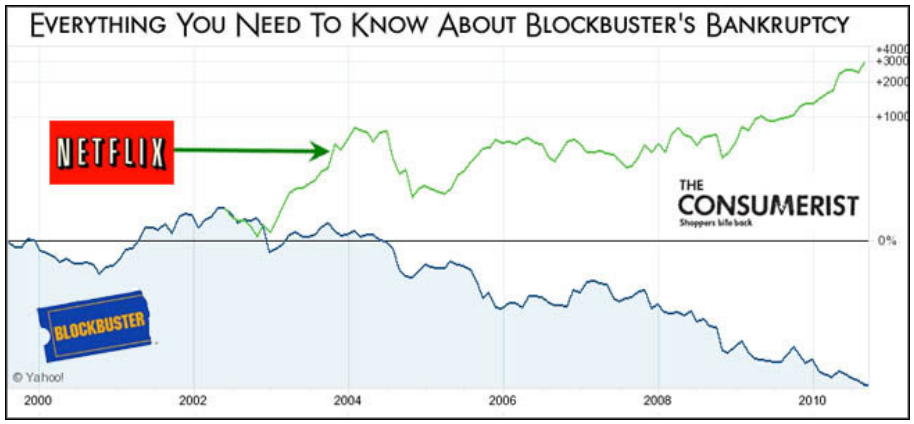
Source: https://www.informationisbeautiful.net/visualizations/million-lines-of-code/

Case Study: Netflix

- Brick-and-mortar store can hold only up to 2000 titles
  - Stores stock popular titles, brand-new releases
  - Business model: DVD Availability Window, steep rental fees, steeper late fees
- Netflix's original business model
  - Store 120k+ titles in 50+ shipping hubs (recommend movies in the long-tail)
  - Find ways to match a consumer's interest with low-cost movie rental
  - Invested in the consumer-recommendation algorithm (crowdsourced, Netflix prize)
- 1999, Netflix launched DVD rentals by mail
  - Unlimited rentals for low monthly fee, no due dates, no late/shipping fees
  - Eliminated Blockbuster, Hollywood Video, Movie Gallery after 8 years

The Netflix Story

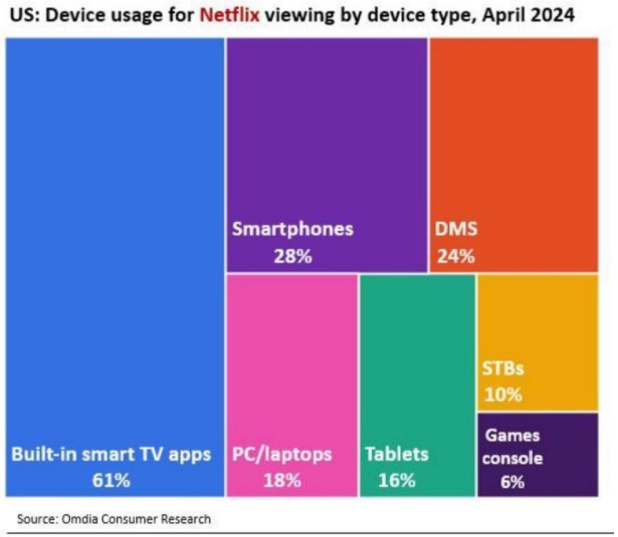




Netflix: DVD-by-Mail to DVD-by-Streaming

- 2008, Netflix saw the opportunity in streaming to home
  - Were spending up to $600M/yr on postage alone
  - Streaming a movie (via deals with CDNs) is 1/100th the cost of shipping a DVD
  - Easier to move into international markets (DVD theft is a crime that requires police intervention; hard to do internationally)
- Netflix streaming service today
  - 301 million+ subscribers in the U.S. and Canada
  - Uses approx 35% of U.S. peak downstream bandwidth
  - 1st or 2nd largest CDN user in the U.S.
  - Growing subscriber base through
    - Partnering up with device makers (game consoles, mobile devices)
    - Strategically decided not to build their own device

Your device. Your way.



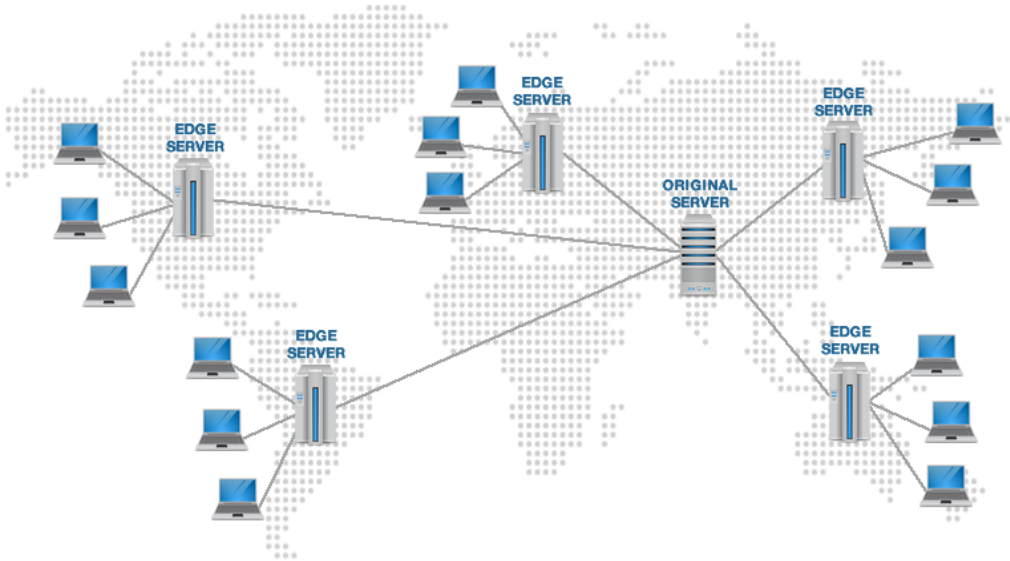
Copyright © 2024. All rights reserved. Informa Tech

Netflix's Cloud-based IT architecture

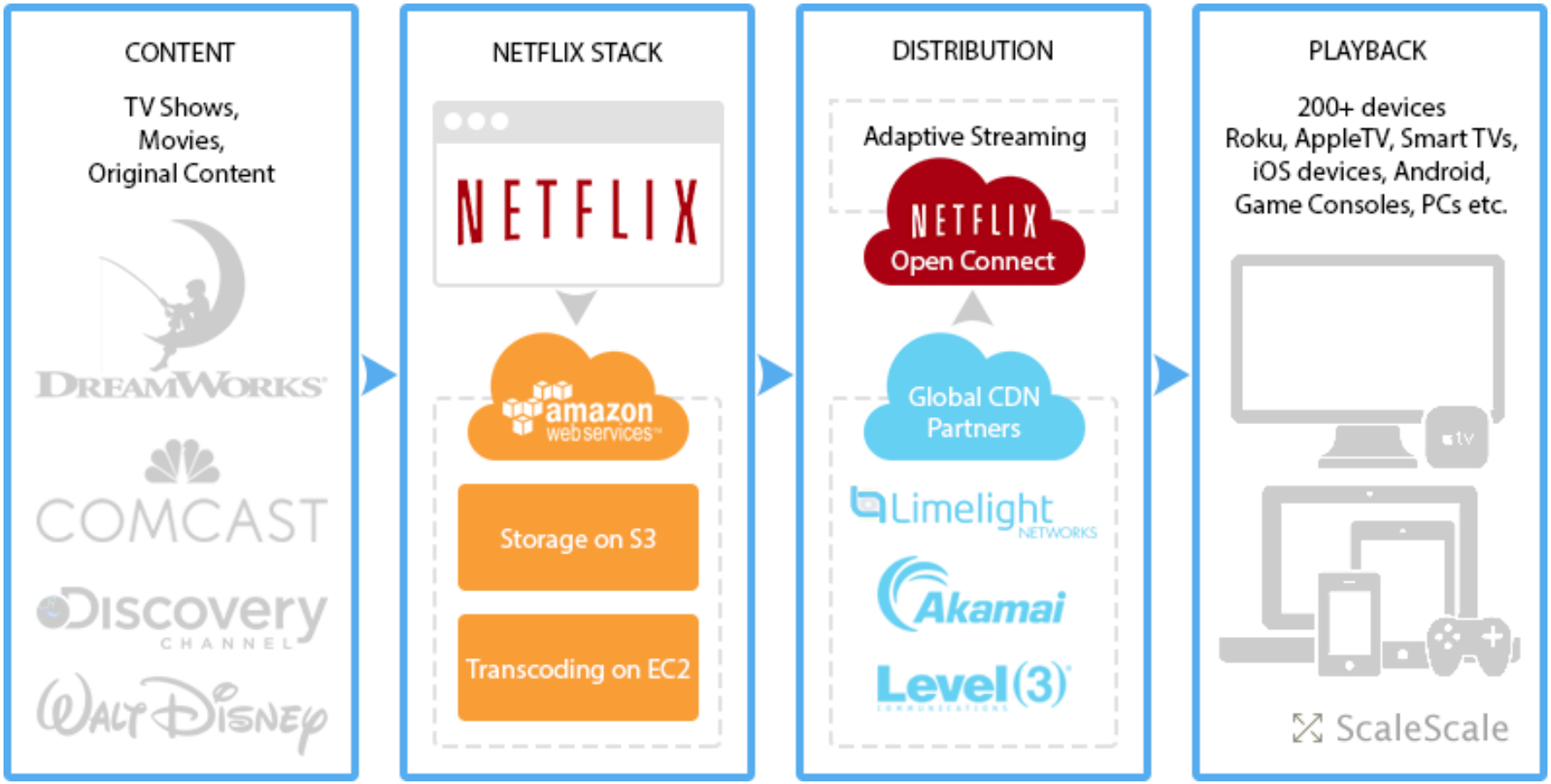
- Netflix had their own data center until 2008
  - Experienced downtime, single point of failure
  - Approached cooling, power, space, traffic, capacity limits
  - 15 IT ops staff, didn't want to worry about scaling out
  - Two alternatives: build more data centers, outsource IT capacity planning and focus on core competency
- Partnered with Amazon Web Services (AWS), 90% of traffic via AWS, except video streaming bits via CDNs
  - Large encoding farms of raw source files from studios, adaptation to different screen resolutions, encryption/decryption, video output to CDNs
  - Migrated back-end support, one type of device at a time, to cloud
  - Big-data: Billions of SimpleDB database statements/day, huge rental history
- Off-the-shelf AWS capability supplemented with custom
  - Data replication and persistent storage
  - Load-balancing and configuration

Netflix's Cloud-based IT architecture

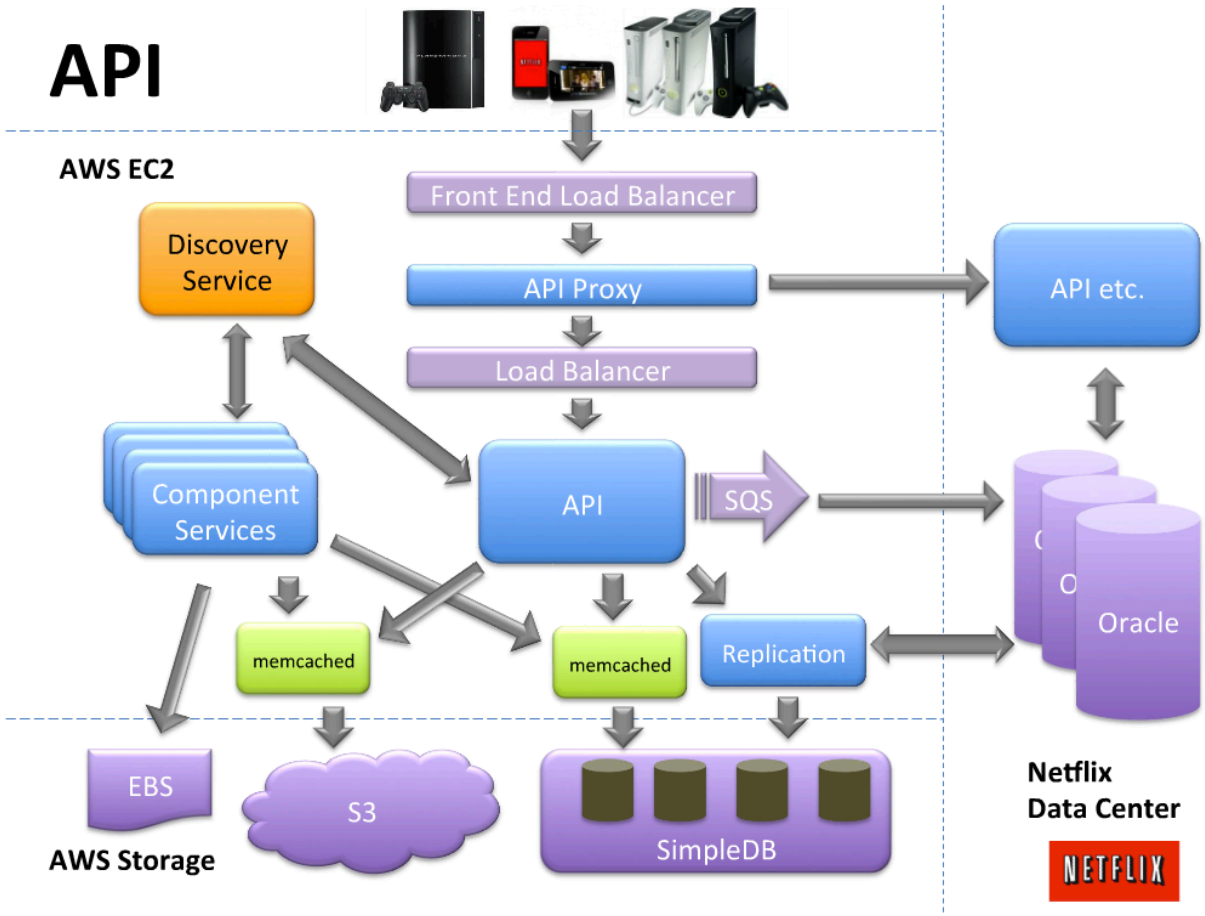
- 100s of micro services
- 1000s of daily production changes
- 10,000s of instances
- 100,000s of customer interactions
- 1,000,000s of customers
- 1,000,000,000s of metrics
- 1,000,000,000,000 hours streamed
- 10s of operations engineers



Netflix's Cloud-based IT architecture



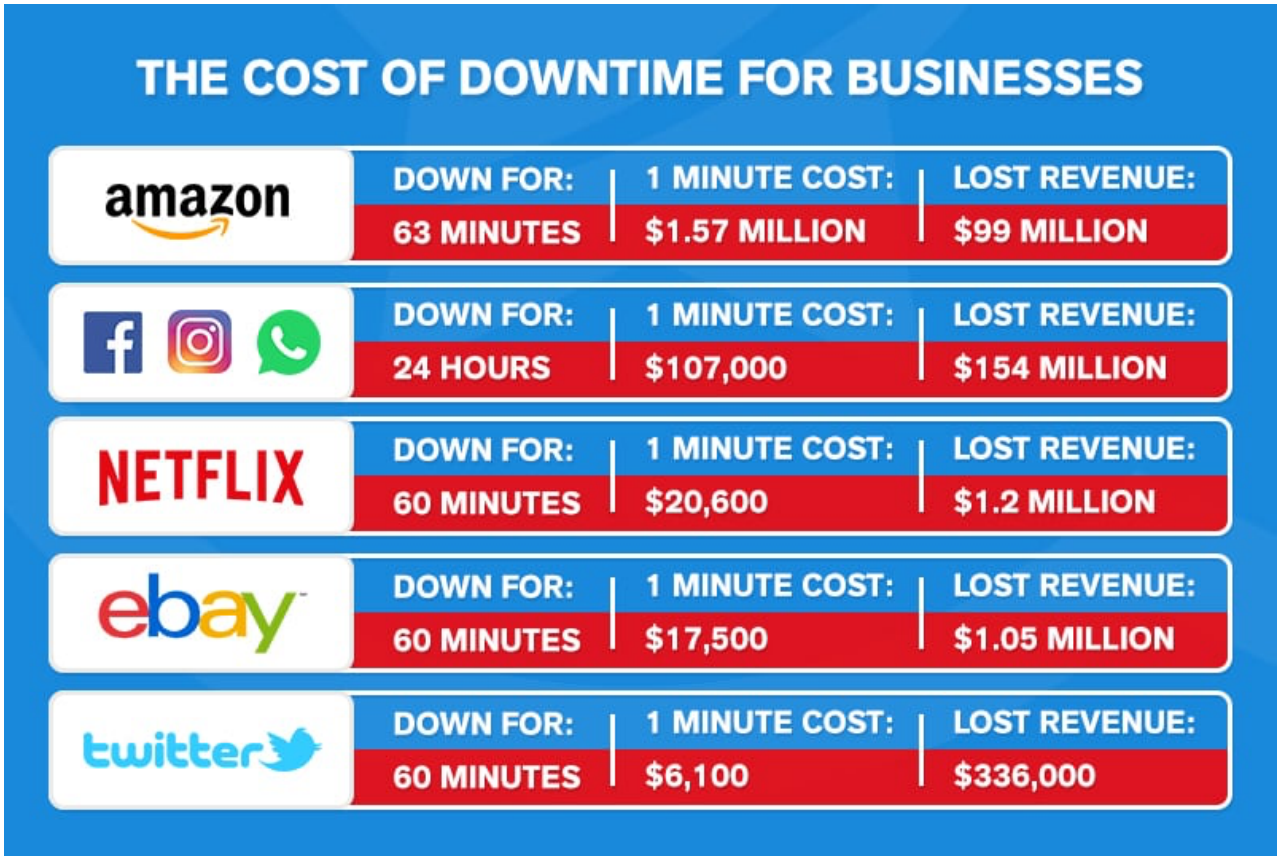
Netflix's Cloud-based IT architecture

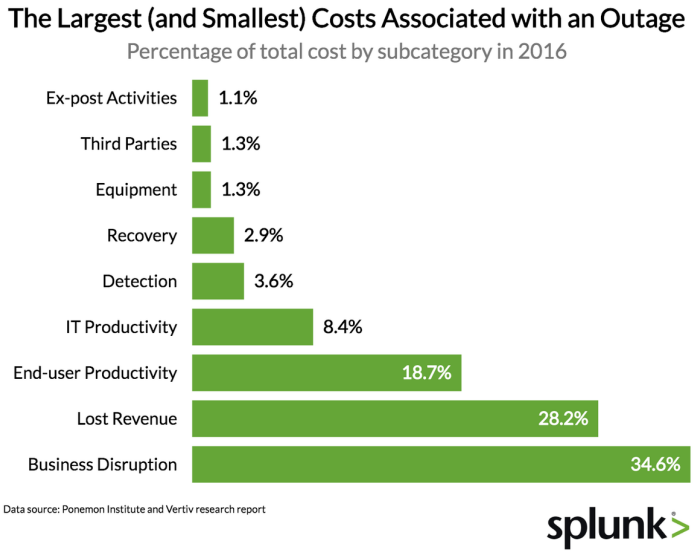


The Point of these "Big Ideas"

- They are all large complex distributed systems that are built to survive failure
- They all have some aspect of reliability built into them
  - Netflix: Reliability to ensure low-latency streaming of video
- Reliability encompasses many aspects
  - Things may fail, and there needs to be a way to detect failures
  - When one thing fails, another thing else needs to take over
  - Users should not see things fail (perceived continuity of service)
  - Failed things need to be able to recover and re-integrate
  - When things fail, data may get lost or data may get inconsistent
  - There needs to be a way to prevent the failure from happening again

Financial Impact of Downtime





Examples of Financial Impact

- July 2019, Twitter platform was down for ~1 hour:
  - Company stock fell by 1%, company lost $336,000
- May 2019: Facebook had a 24-hour service disruption
  - Company lost $154MM in revenue
- July 2018: Amazon had 1 hour of downtime on their Prime Day
  - Company lost $99MM in revenue
- June 2018: Netflix went down
  - Company lost $1.2MM in revenue

- 1999: eBay was down for 24 hours
  - Longest online downtime ever recorded (notable incident)
  - Company lost $5MM in sales

Amazon EC2 Failure

- April 21, 2011
- Outage at EC2's Northern Virginia data center
- Affected the websites of multiple companies
  - Foursquare
  - Quora
  - Hootsuite
  - Reddit
- Performance problems when using EC2
- Data loss in some cases

Amazon EC2 Failure

At 12:47 AM PDT on April 21st, a network change was performed as part of our normal AWS scaling activities in a single Availability Zone in the US East Region. The configuration change was to upgrade the capacity of the primary network. During the change, one of the standard steps is to shift traffic off of one of the redundant routers in the primary EBS network to allow the upgrade to happen. The traffic shift was executed incorrectly and rather than routing the traffic to the other router on the primary network, the traffic was routed onto the lower capacity redundant EBS network. For a portion of the EBS cluster in the affected Availability Zone, this meant that they did not have a functioning primary or secondary network because traffic was purposely shifted away from the primary network and the secondary network couldn't handle the traffic level it was receiving.

(Amazon Blog Post)

What is Dependability?

- Computing systems characterized by functionality, performance, cost
- Correct service
  - Delivered when the service implements the system function
- System failure
  - Event that occurs when the delivered service deviates from correct service
  - Transition from correct service to incorrect service
- Service recovery
  - Transition from incorrect service to correct service
- Service outage
  - Time interval during which incorrect service is delivered

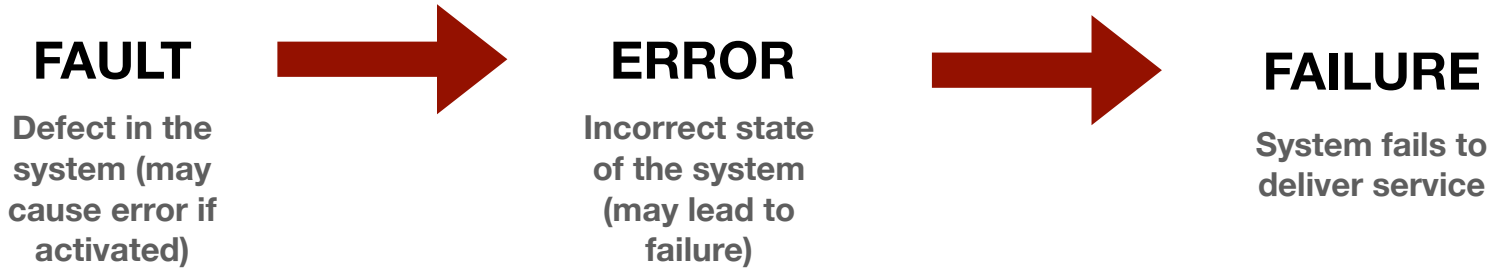
The Definition of Dependability

Dependability is the ability to deliver service that can justifiably be trusted.

For discussion in class: So, how should we achieve the “justifiably” part?

Terminology

- **Fault** = hypothesized case of an error
- **Error** = incorrect system state that may cause a subsequent failure
- **Failure** = what happens when an error reaches the service interface and alters the service
- Fault is active when it produces an error; else, dormant
- Error is detected when its presence is signaled by an exception; else, latent



We ♥ Errors

- **Error** = incorrect system state that may cause a subsequent failure
- Errors are detectible, errors can be seen, errors can be caught
  - For discussion in class: Why is this a good thing?

- The story of *mercaptan*: Pungent, smelly gas that is added to natural gas (which is colorless and odorless)

How Do We Achieve Dependability?

- Fault prevention – how to prevent the occurrence of faults?
  - Design your systems well
- Fault tolerance – how to deliver correct service in the presence of faults?
  - Error detection, recovery, logging, checkpointing
- Fault containment or removal – how to reduce the number/severity of faults?
  - Corrective & preventive maintenance
- Fault diagnosis — how do we know the root-cause so the fault won't happen again?
  - Monitoring, root-cause analysis
- Fault forecasting – how to estimate the future incidence of faults?
  - Simulation, modeling, prediction from process metrics (data from models)
  - Historical data, accelerated life testing, etc. (data from real systems)

What Kinds of Faults Are Interesting to Us?

- **Crash fault** – Something crashes and stops working
  - Process crash
  - Processor/machine crash
  - **Fail-stop fault or fail-silent fault**
  - For discussion in class: So, how should we handle this?
- **Message loss** – A message is not delivered
  - Communication/network fault
  - **Omission fault**
  - For discussion in class: So, how should we handle this?
- **Hang fault** – Something is hung and we can't get service from it
  - Process hang
  - Processor/machine hang
  - For discussion in class: So, how should we handle this?

✓ Will cover in this course

What Kinds of Faults Are Interesting to Us?

- **Network-partitioning fault** – Network splits a system into multiple disconnected pieces
  - Split-brain fault
  - Machines on different sides can't reach each other
  - For discussion in class: So, how should we handle this?
- **Timing fault** – Something fails to happen on time
  - A deadline is missed
  - For discussion in class: So, how should we handle this?
- **Design fault** – Something goes wrong, but nothing crashes or hangs
  - Algorithmic fault
  - For discussion in class: So, how should we handle this?

✓ Will cover in this course

Handling Design Faults

- N-version programming or design diversity
  - Make multiple software versions and let them vote

- If you have infinite money, then use all available techniques
  - Diverse implementations
  - Diverse specifications
  - Diverse testing/certification teams
  - For discussion in class: Do you see a flaw with this?

Fault Masking

- Making it appear to the user that the fault never happened
- How?
  - Use enough redundancy in the system to allow recovery without explicit error detection
- What's good?
  - User is blissfully unaware of anything bad happening
  - Fault/failure transparency – great way to build systems!
- What's bad?
  - **Independent-failure assumption**
  - Does not deal adequately with design faults (e.g., core dump)

For discussion in class: How real are independent failures?

Reliability Engineering vs. Fault-Tolerance

- Reliability engineering suggests that either a system is safe or compromised
  - Focuses on preventing faults, and keeping defects out
- No system is ever fully fault-tolerant
  - Systems and components can fail, and are failing, daily
- Fault-tolerance focuses instead on how a system can
  - Continue to function, in the face of faults
  - Recover from a fault
  - Deals with the issue of consistency under faults

So, How Do We Do Fault-Tolerance?

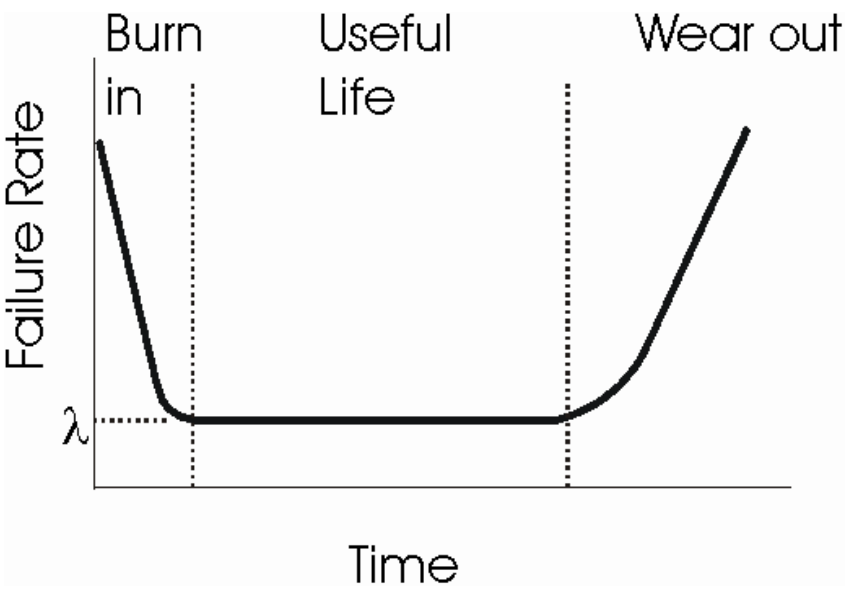
- Replication
  - The art of making copies, and keeping copies as copies
  - If one copy fails, another copy does the work

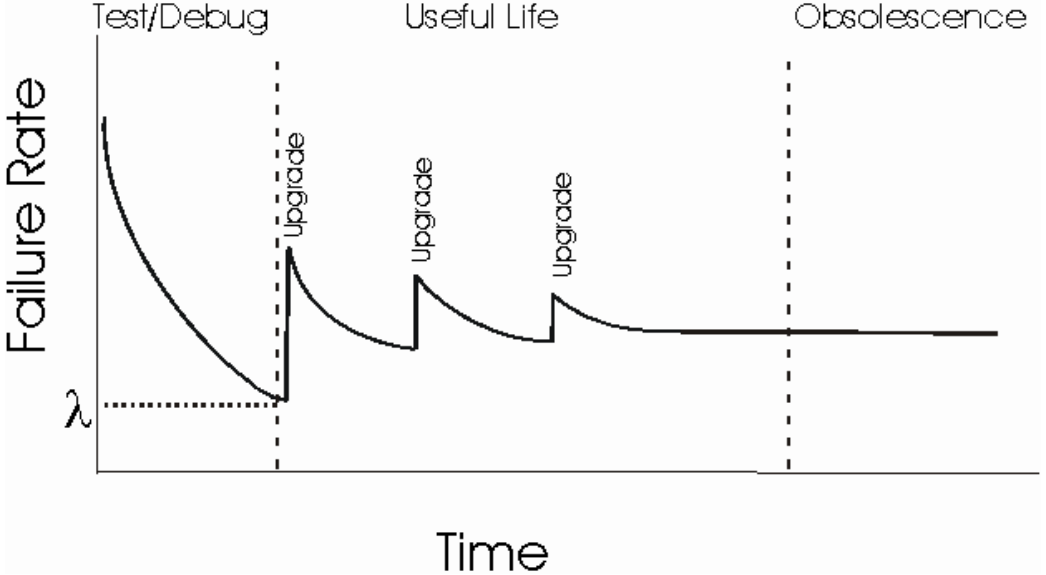
- For discussion in class: What's the challenge with this?
  - How do we create a copy?
  - How do we ensure that a copy remains a copy?
  - How do we ensure that a copy remains a copy, even when another copy fails?
  - How do we ensure that a copy remains a copy, even when it fails and then recovers again?

Fault-Tolerance is Recursive!

- The mechanisms that provide fault tolerance must be fault-tolerant also!
- Examples
  - Voter replication
  - Self-checking checkers
- Issue in fault tolerance
  - Who watches the watchers?

Hardware vs. Software Reliability





What Will You Learn in This Course?

- To take distributed software systems apart to find their weaknesses
  - How do we find a single point of failure?
- To learn about different types of reliability and fault-tolerance
- To learn the fundamental building blocks for fault-tolerance, e.g., consensus protocols
- To understand what industry does for fault-tolerance today
- To understand how today's large systems (Netflix, Google, Amazon) handle reliability
- (Indirectly) how to put distributed software systems together in the first place so that they work reliably and meet operational requirements

Summary

- Logistics of the course
  - Brief tour of the value of dependability
  - Policies
  - Grading criteria
- Why should you take this course?
- Welcome to 18-749 – good luck this semester!
- Enroll on Slack (we will use that for all course communication)
- Slides will be posted to Slack
- We will use Canvas for grading and for the exams