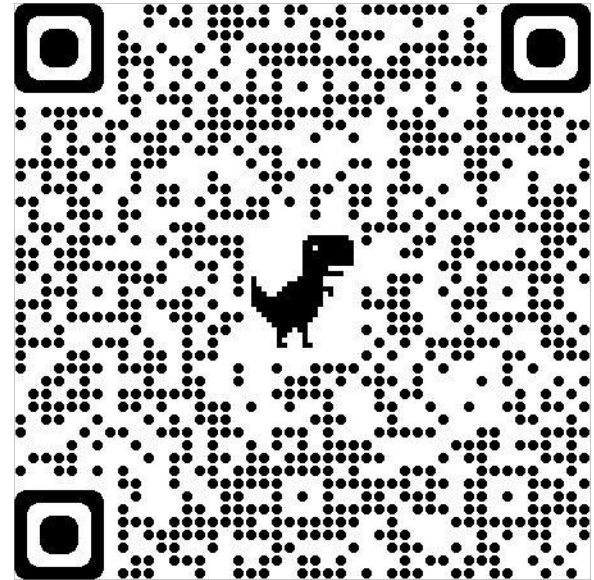


# CS130: Software Engineering

## Lecture 15: Monitoring, Documentation, & Postmortems

<https://bit.ly/3Lr3BSp>

- Tweet: What's the coolest thing you've learned so far? (could be anything)
- Tweet: Is there anything that you learned outside of class that's helped you in class?



# Assignment 8/9 thoughts

# Assignment 8/9...

- Persistent disks
- Project choice
  - Use CRUD?
  - Use your own CRUD?

# Other...

- Final exam
- Assignment 6 / 100

# Interlude: Testing log rotation

# From Assignment 4

- *For file logging, a new log file should be started daily at midnight, and when a day's log file size reaches 10 MB. (This is called log file rotation.)*
- Quite a few questions and problems with the implementation of this solution.

Assume your implementation went something like this:

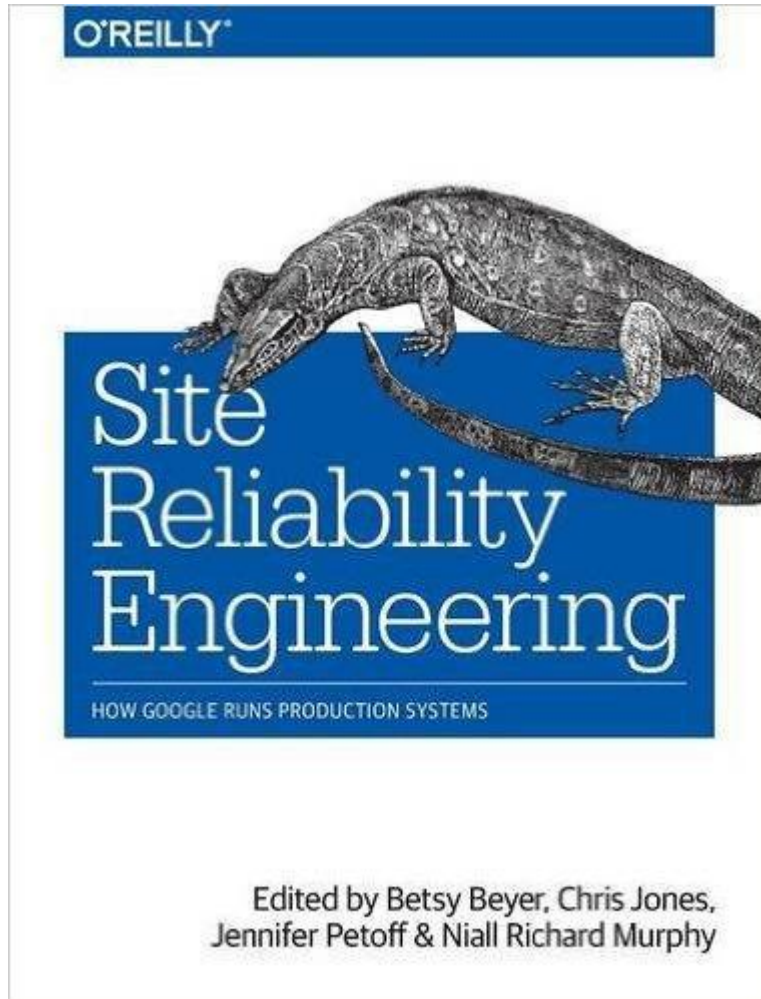
```
if ((log.timestamp() > midnight and last_log_timestamp < midnight) ||  
    (log.content.length() + current_log_size > 10000000)) {  
    log_file = rotateLogFile();  
}  
log_file.write(log);
```

How do you test this? How do you prove it works (to yourself or the grader)?

# Log Rotation Testability -- we now have tools to do better.

- Problem: embedding your logging code in the server too tightly
  - Hard to decouple to test
  - Need to wait 24 hours to see if it rolls the log
  - Need to write a huge amount of logs to see it roll.
  - ---> Hard to test and hard to see if it works.
- Need to configure the logger so it is testable!
  - Set the size and rotation time or interval with parameters
  - Set the parameters from the config file
  - The config file is not a toy! You almost certainly would want controls for this there anyway.
  - Want some kind of internal API to the logger that makes testing this functionality easy.
- Now testing is easy:
  - Make a config with an interval of 1 second and test rollover
  - Make a config with a size limit of 1 bytes and test rollover
  - If the class managing logging is separate, these can all be unit tests and managed without touching your server code at all.
  - Result is a clean module your server can invoke with confidence
  - Needs an API! The API needs to be thought through! (Is it multi-threaded? If not what bad thing happens? etc.)

# Monitoring overview



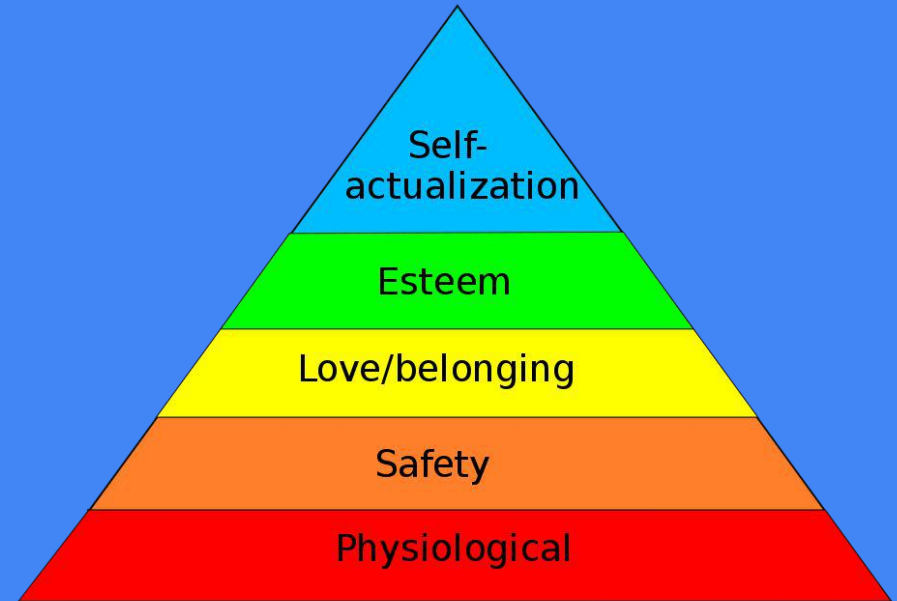
## The book

- Some content here derived from the book [Site Reliability Engineering](#)



# Pyramids 101

- Maslow's Hierarchy of Needs
- Self-actualization is the goal
- Physiological needs must be met
  - Food
  - Water
  - Shelter
  - Sleep

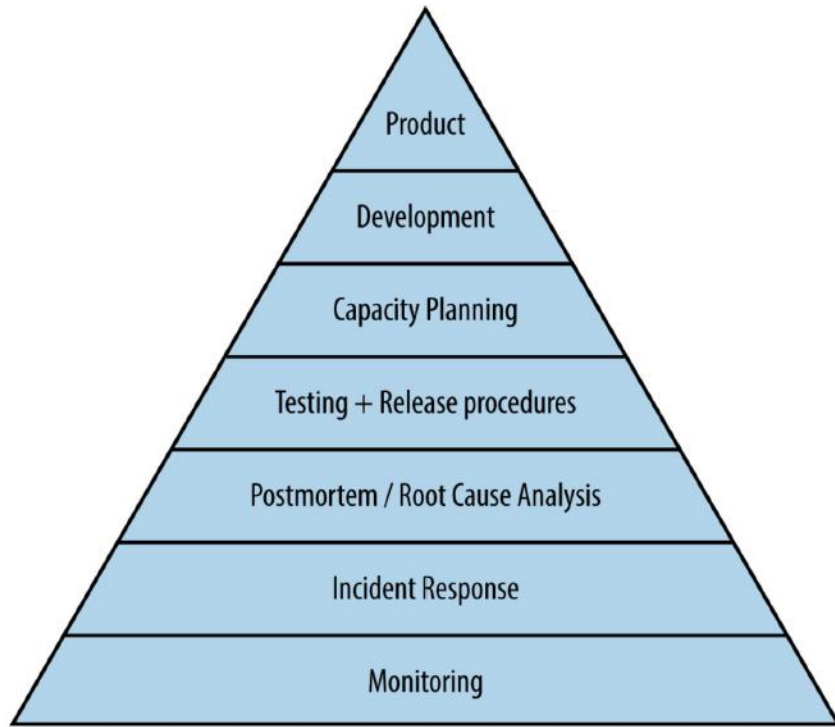


# Pyramids 101

- John Wooden's Pyramid of Success
- Competitive Greatness is the goal
- Foundations are most important
  - Industriousness
  - Friendship
  - Loyalty
  - Cooperation
  - Enthusiasm



# Pyramids 101



- Site Reliability Engineering's pyramid of productionization
- Successful product is the goal
- Monitoring is the foundation

# Production terminology

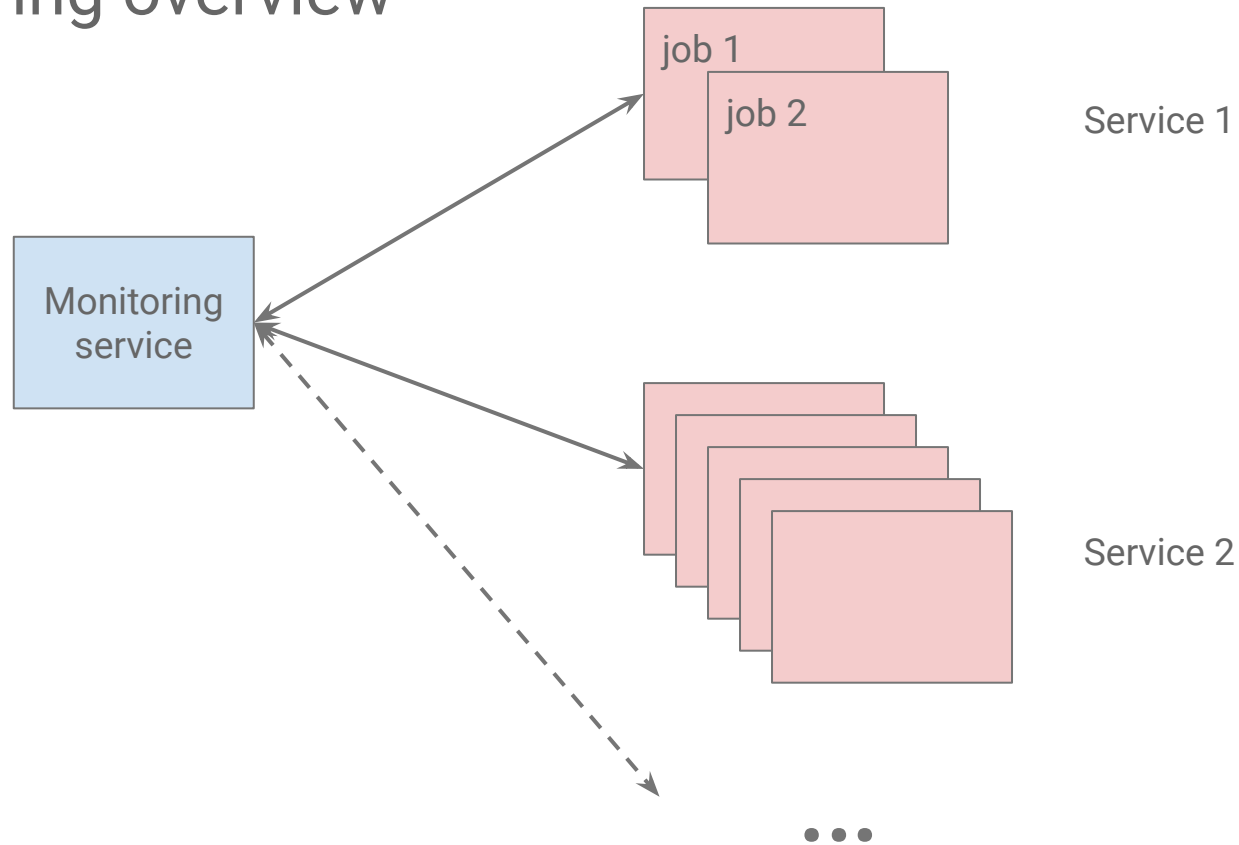
## Job

- A process running on a machine
- Instance of a Service
- Docker container

## Service

- Collection of identical jobs
- Definition of a job
- Docker image

# Monitoring overview



# Time series

<b>Time</b>	<b>t0</b>									
Running?	Y									

# Time series

Time	t0	t1	t2	t3	...					
Running?	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

# Time series

Service 1

<b>Time</b>	<b>t0</b>	<b>t1</b>	<b>t2</b>	<b>t3</b>	<b>...</b>						
<b>Running?</b>	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	

Service 2

<b>Time</b>	<b>t0</b>	<b>t1</b>	<b>t2</b>	<b>t3</b>	<b>...</b>						
<b>Running?</b>	Y	Y	Y	Y	Y	Y	N	N	N	N	



# Black-box monitoring

- OS-level monitoring
- Agnostic of application internal state
  - No integration with application
- Basic metrics:
  - CPU usage
  - Disk space
  - Network bandwidth
  - Process running

# White-box monitoring

- Application-level monitoring
- Exposes application internal state
  - Parsing logs or URLs
- Advanced metrics:
  - Number of requests
  - Response codes
  - Exceptions
  - Latency to remote services

# Implementing Server Monitoring

# Exposing server data

Recall from previous lecture on Threading and Concurrency:

```
class Counter {
public:
    void Increment(const string name, int by);
    map<string, int>* GetSnapshot(); // this is new
private:
    map<string, int> counters_;
    std::mutex counter_mutex_;
}
```

# Exposing server data

```
location /monitoring MonitoringHandler { ... }
```

```
class MonitoringHandler : public RequestHandler {  
    unique_ptr<Reply> HandleRequest(const Request& request) {  
        map<string, int>* counters =  
            Singleton<Counter>::get()->GetSnapshot();  
        std::stringstream body;  
        for (const auto& kv : *counters) {  
            body << kv.first << ": " << kv.second << std::endl;  
        }  
        // Set ContentType text/plain, set the body, etc.  
    }  
}
```

# Exposing server data

Elsewhere...

```
Singleton<Counter>::get()->IncrementBy("requests", 1);  
Singleton<Counter>::get()->IncrementBy("errors", 1);  
Singleton<Counter>::get()->IncrementBy("bytes_transferred", 1234);  
... etc.
```

# Viewing server data

Fetch `http://<url>:<port>/monitoring`:

```
requests: 1253
errors: 7
bytes_transferred: 1937585
...
```

With some work, you could also support map-valued counters:

```
http_responses: {
    200: 25
    404: 0
    500: 1
}
```

# Integration with monitoring

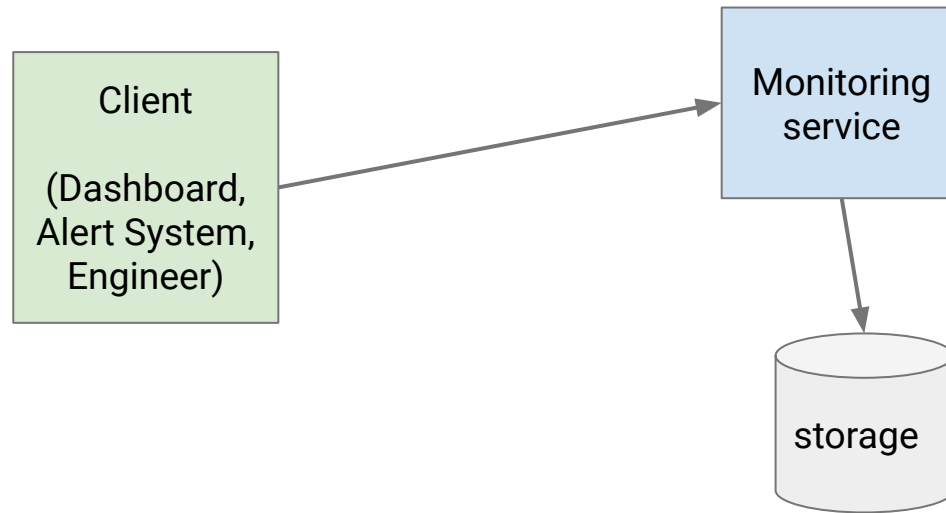
Monitoring process fetches and parses `http://<url>:<port>/monitoring`

Time	t0	t1	t2	t3	...					
Running?	Y	Y	Y	Y	Y	Y	N	N	N	N
requests	10	21	27	35	49	54				
errors	0	0	1	1	1	3				
...										

# Using Monitoring: Dashboards



# Other parts of the monitoring service



# Client interface

- A query language for retrieving time series and transforming them
- For example, since we know the count of requests and errors:
  - Compute requests per second for the whole service
  - Compute fraction of requests that are errors
- Important clients:
  - A graphing interface / dashboard
  - Alerts



Monitoring Overview

Resources

Alerting

Uptime Checks

Groups

Dashboards

Debug

Trace

Logging

Error Reporting

Google Cloud Platform

Uptime Checks / Uptime BETA

TIME 1h 6h 1d 1w 1m 6w custom

Create Alerting Policy

## Uptime Check latency

by checker location, instance id (mean) 1 min interval (mean)



checker_location	instance_id	Value
eur-belgium	instance-1	431.86ms
sa-brazil-sao_paulo	instance-1	635.43ms
usa-iowa	instance-1	306.79ms
usa-oregon	instance-1	206.30ms

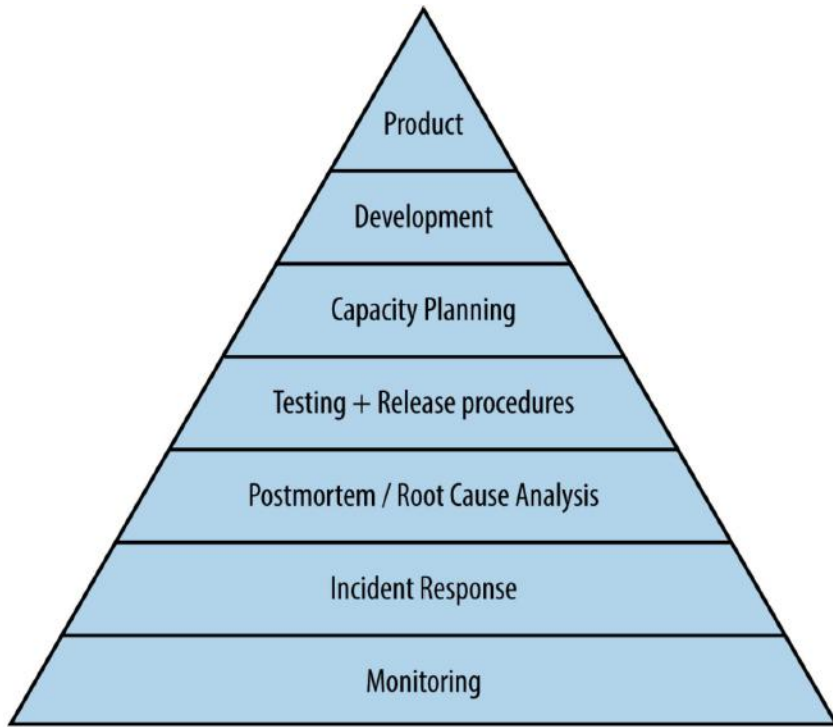
Uptime 100.000%

Location results All locations passed

Check config instance-1/ok

# Using Monitoring: Alerts

# Incident response



- Monitoring allows for detection of failures
  - Server crash
  - Failures in remote services
  - Errors being served to users
- How do you know these are happening?
  - Stare at dashboard?
  - Alerting!

# Alerts

## Components of an alert

- Condition: fraction of non-200 responses > 5%
- Time period: 2 minutes
- Severity: page the person on-call
- Message: "Too many non-200 HTTP responses"

# Alert severities

Depending on how urgent the alert is, we can do different things:

- File a bug
- Send an email
- Page during regular work hours
- Page 24x7





# Service Level Objectives (SLO)

# Service Level Objectives (SLO)

What's our goal for the uptime of our service?

- "5 9s" → 99.999% available
  - 5 minutes of downtime per year
- 99%
  - 3.6 days of downtime per year

In reality, measured by requests, not by time

How quickly will we respond to incidents

- Within 1 minute?
- Within 30 minutes?
- Within a day?
- What about nights and weekends?

# Implications of a strict SLO

- Need more people oncall
- If possible, want engineers in different timezones
- Need to consider long-term work/life balance
- More training and practice for people oncall

# Minimizing downtime

Example SLO: 10 minutes of downtime per week

If we achieve 5 minutes of downtime, is that better?

# What causes downtime?

# What causes downtime?

Change!

- New features
- Changing configuration
- Releases
- Growth of user base
- etc.

# SLOs are a risk budget

Too little downtime also has risks:

- Strain on on-call team
- Too little time spent developing features  
Instead, being on-call and working to reduce risk.
- Clients assume your service is too reliable.  
Lack of error handling when service is down.

Don't try to minimize risk at all costs.

Instead, figure out the right amount of risk for the business to take.

# SLOs affect development practices and priorities

## Strict SLO

- Testing
- Deployment
- Monitoring
- Change management
- On-call

## Loose SLO

- Features
- Prototyping
- Major architectural improvements
- Trying out crazy ideas



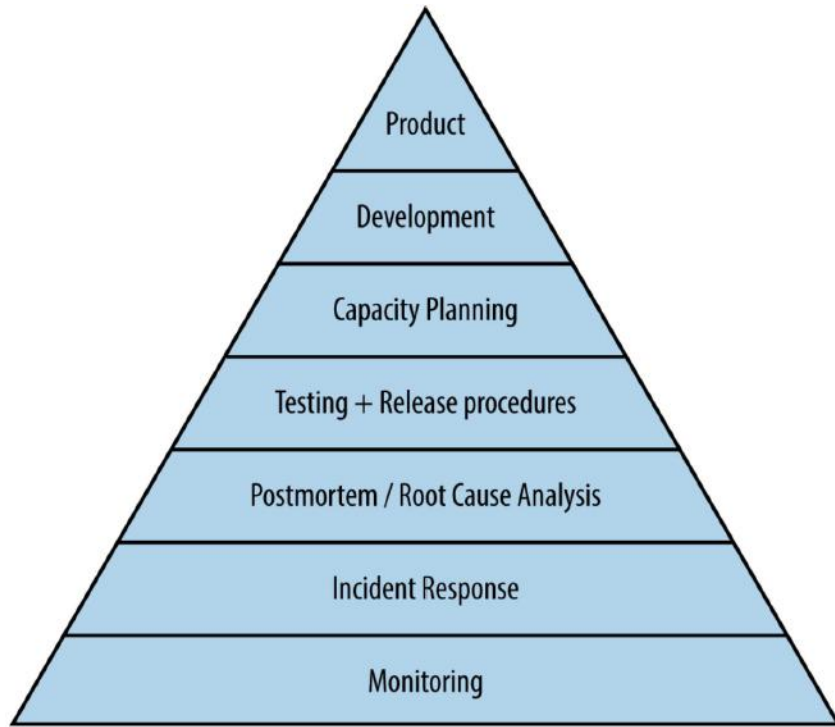
# Minimizing toil

Toil defined:

- Manual
- Repetitive
- Automatable
- Tactical
- No enduring value
- $O(n)$  with service growth

We tend to associate testing, deployment, etc. with toil, but they don't have to be.

# Pyramids 101



- Successful product
- Application development
- Capacity planning
- Testing and release procedures
- Postmortem / root cause analysis
- Incident response

... all depend on monitoring

# Documentation

# Starting on a new project...

1. Read requirements docs
2. Read design docs
3. Read API docs
4. Read code

# Product requirements documents (PRD)

- Why?
- What?

# Technical design docs (DD)

- Why?
- What?
- How?

# DNS Prefetching

- Why? (The problem)

DNS resolution is slow

- What? (The solution)

Resolve addresses before user clicks link.

- How? (The implementation details)

# Product requirements documents (PRD)

- Vision
- Motivation
- Goals
- Requirements
- Success criteria

# Technical design docs (DD)

- Objective
- Background
- Requirements
- Detailed design
- Alternatives considered

# Good requirements are...

- Unambiguous
  - Complete
  - Verifiable
  - Consistent
  - Modifiable
  - Traceable
  - Usable for the lifetime of the product (implementation, maintenance)
- Use strong language like “shall” and “will”
  - Avoid ambiguous language like “could” and “may”
  - Quantify verification criteria, not just “be good”



# Technical design docs: extra credit

- Security
  - Privacy
  - Testing plan
  - Work estimate (how long?)
- Performance - latency, throughput, scalability
  - Migration strategy
  - Reliability - SLA
  - Deployment and launch plan

# API docs

- How to use the code
- What are the classes?
- What do the methods do?

Example of good API docs:

<https://docs.oracle.com/javase/8/docs/api/java/lang/String.html>

# JavaDoc

- Generate API docs from special comments
- `/** ... */`

```
/**
 * Returns a new string that is a substring of this string. The
 * substring begins at the specified <code>beginIndex</code>
 * and extends to the character at index <code>endIndex -
 * 1</code>. Thus the length of the substring is
 * <code>endIndex-beginIndex</code>.
 * <p>
 * Examples:
 * <blockquote><pre>
 * "hamburger".substring(4, 8) returns "urge"
 * "smiles".substring(1, 5) returns "mile"
 * </pre></blockquote>
 *
 * @param    beginIndex    the beginning index, inclusive.
 * @param    endIndex      the ending index, exclusive.
 * @return    the specified substring.
 * @exception IndexOutOfBoundsException if the
 *           <code>beginIndex</code> is negative, or
 *           <code>endIndex</code> is larger than the length
 *           of this <code>String</code> object, or
 *           <code>beginIndex</code> is larger than
 *           <code>endIndex</code>.
 */
public String substring(int beginIndex, int endIndex) {
    ...
}
```

# Quick survey

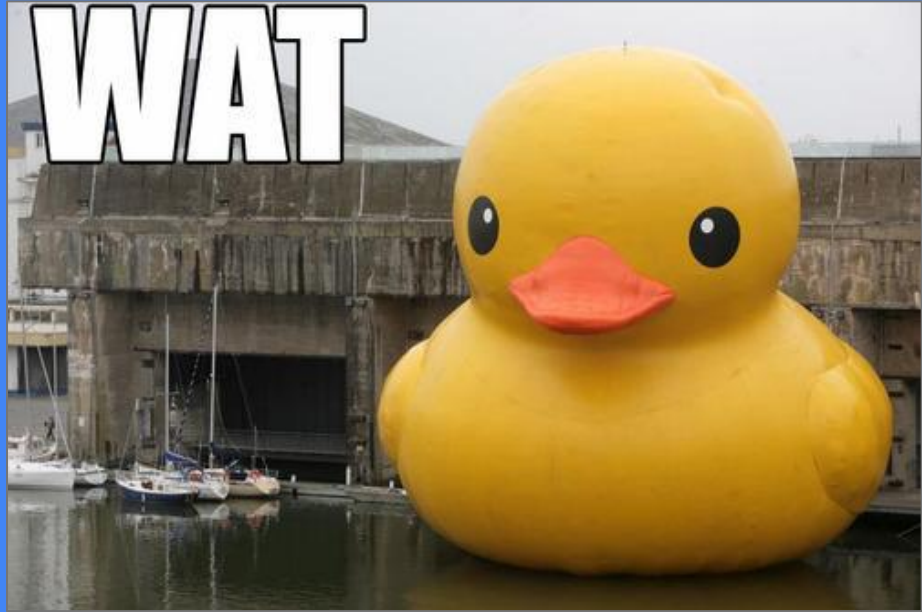
- How many people have written code for this class?
- How many people have written design docs for this class?
- How many people have written API docs for this class?

# READ THE SOURCE LUKE



"The source code is  
the documentation"

```
initServletSystem(  
    true, false, 1500, true, true);
```



# "The source code is the documentation"

Better...

```
initServletSystem(true, // Use authentication
                  false, // Don't enable the cache
                  1500, // 1500 ms initialization timeout
                  true, // Continue startup on servlet init error
                  true); // Enable logging
```

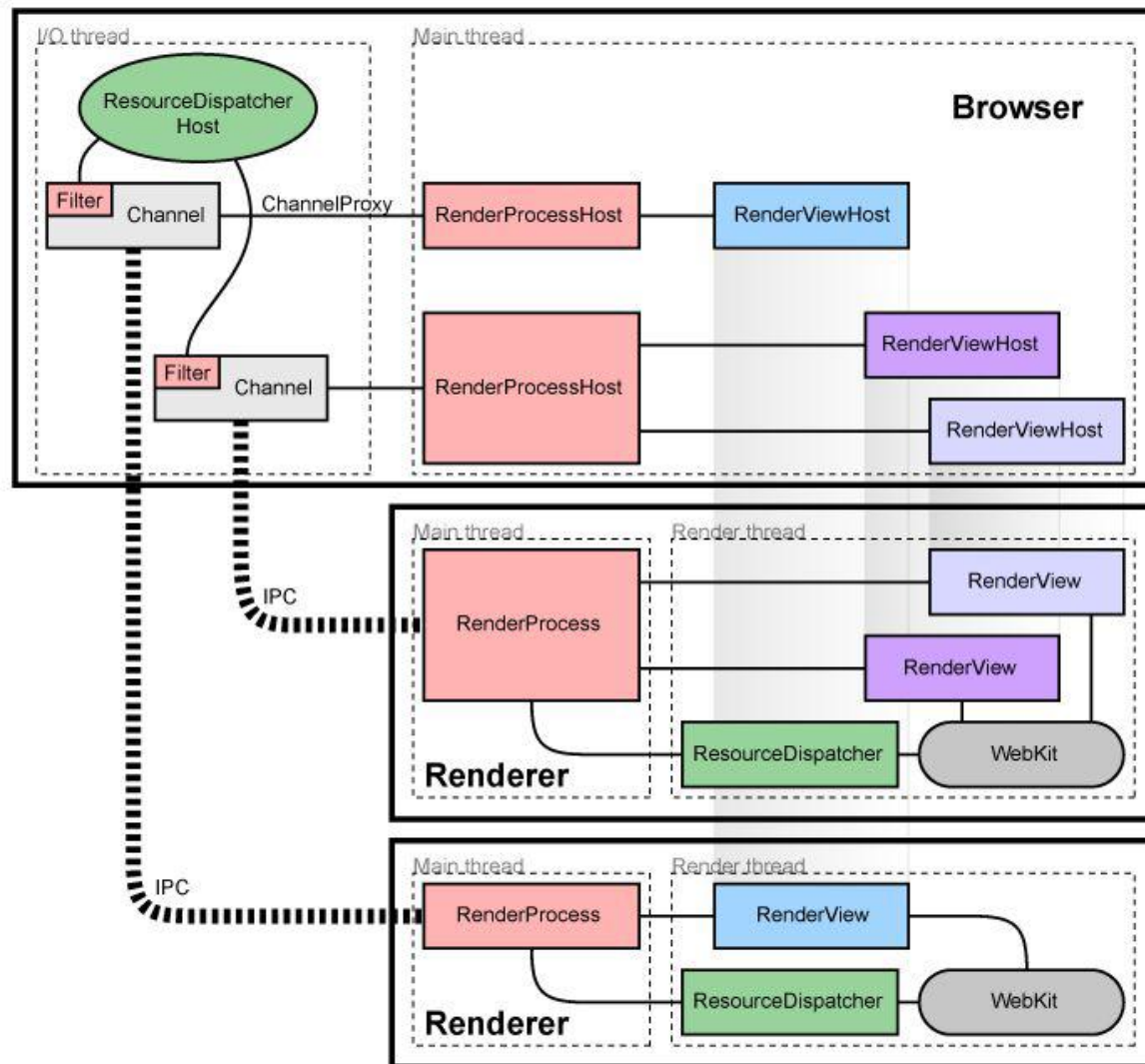
# "The source code is the documentation"

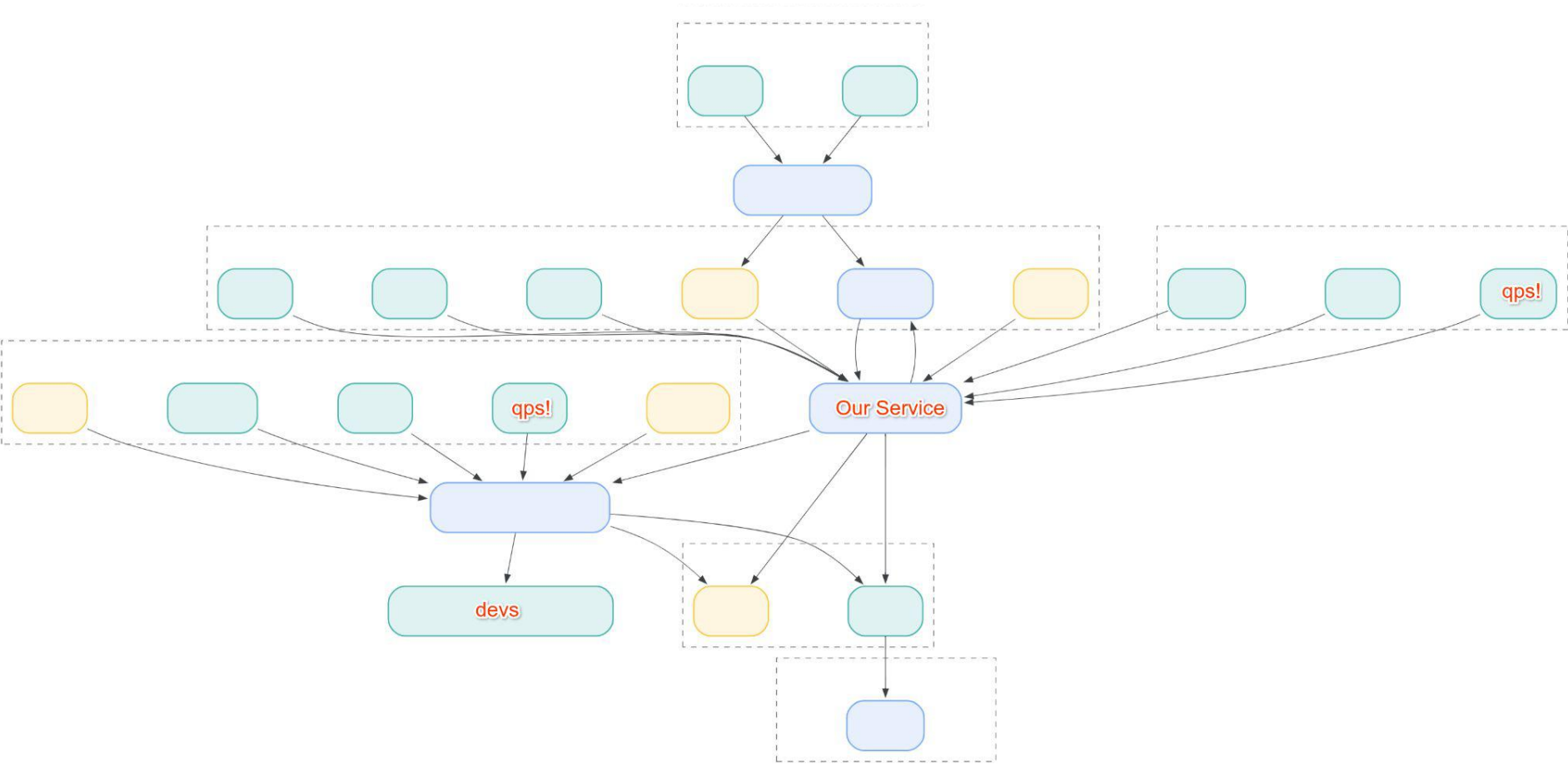
Even better...

```
initServletSystem(ServletAuth.ENABLED,  
                  ServletCache.DISABLED,  
                  ServletSystem.INIT_TIMEOUT_1500_MS,  
                  ServletSystem.CONTINUE_ON_ERROR,  
                  Logging.ENABLED);
```



This is just the beginning...

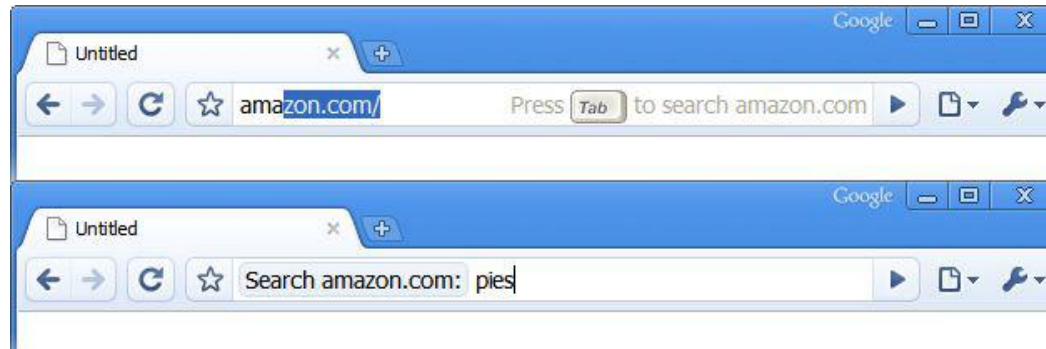




# Use cases....

## Tab to Search

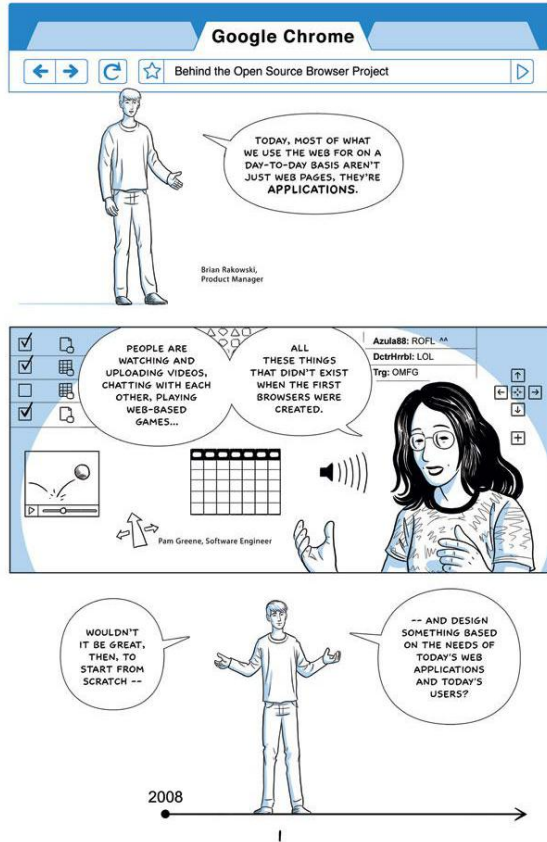
If a user is partway through typing something that auto-completes to a keyword, we allow the user to press **Tab** to jump to the end of the auto-completion and begin their keyword search term input. This is powerful when combined with our automatic keyword creation - a user who does a search on amazon.com in the course of their normal browsing will be presented with the option of pressing **Tab** to do an Amazon search the next time they type something that auto-completes to amazon.com, leading to the ability to type 'ama [tab to complete] pies'. We call this functionality 'tab to search'.



# And more...

- User stories
- UI mocks
- Tutorials
- End-user documentation
- Contributor docs (how to be a team member)
- FAQs

# Even cartoons!



# Postmortems

*“Cherish your system-failures”*

—*The Systems Bible* by John Gall



# A long ago outage



How a typo took down  
S3, the backbone of the  
internet

The Verge · 2 days ago



Amazon S3 outage  
spotlights disaster  
recovery tradeoffs

SearchAWS.com - TechTarget · ...



Employee Error to  
Blame For Huge  
Amazon S3 Outage This  
Week

DSLReports · 1 day ago

# Dissecting a recent postmortem

## Summary of the Amazon S3 Service Disruption in the Northern Virginia (US-EAST-1) Region

We'd like to give you some additional information about the service disruption that occurred in the Northern Virginia (US-EAST-1) Region on the morning of February 28th. The Amazon Simple Storage Service (S3) team was debugging an issue causing the S3 billing system to progress more slowly than expected. At 9:37AM PST, an authorized S3 team member using an established playbook executed a command which was intended to remove a small number of servers for one of the S3 subsystems that is used by the S3 billing process. Unfortunately, one of the inputs to the command was entered incorrectly and a larger set of servers was removed than intended. The servers that were inadvertently removed supported two other S3 subsystems. One of these subsystems, the index subsystem, manages the metadata and location information of all S3 objects in the region. This subsystem is necessary to serve all GET, LIST, PUT, and DELETE requests. The second subsystem, the placement subsystem, manages allocation of new storage and requires the index subsystem to be functioning properly to correctly operate. The placement subsystem is used during PUT requests to allocate storage for new objects. Removing a significant portion of the capacity caused each of these systems to require a full restart. While these subsystems were being restarted, S3 was unable to service requests. Other AWS services in the US-EAST-1 Region that rely on S3 for storage, including the S3 console, Amazon Elastic Compute Cloud (EC2) new instance launches, Amazon Elastic Block Store (EBS) volumes (when data was needed from a S3 snapshot), and AWS Lambda were also impacted while the S3 APIs were unavailable.

S3 subsystems are designed to support the removal or failure of significant capacity with little or no customer impact. We build our systems with the assumption that things will occasionally fail, and we rely on the ability to remove and replace capacity as one of our core operational processes. While this is an operation that we have relied on to maintain our systems since the launch of S3, we have not completely restarted the index subsystem or the placement subsystem in our larger regions for many years. S3 has experienced massive growth over the last several years and the process of restarting these services and running the necessary safety checks to validate the integrity of the metadata took longer than expected. The index subsystem was the first of the two affected subsystems that needed to be restarted. By 12:26PM PST, the index subsystem had activated enough capacity to begin servicing S3 GET, LIST, and DELETE requests. By 1:18PM PST, the index subsystem was fully recovered and GET, LIST, and DELETE APIs were functioning normally. The S3 PUT API also required the placement subsystem. The placement subsystem began recovery when the index subsystem was functional and finished recovery at 1:54PM PST. At this point, S3 was operating normally. Other AWS services that were impacted by this event began recovering. Some of these services had accumulated a backlog of work during the S3 disruption and required additional time to fully recover.

We are making several changes as a result of this operational event. While removal of capacity is a key operational practice, in this instance, the tool used allowed too much capacity to be removed too quickly. We have modified this tool to remove capacity more slowly and added safeguards to prevent capacity from being removed when it will take any subsystem below its minimum required capacity level. This will prevent an incorrect input from triggering a similar event in the future. We are also auditing our other operational tools to ensure we have similar safety checks. We will also make changes to improve the recovery time of key S3 subsystems. We employ multiple techniques to allow our services to recover from any failure quickly. One of the most important involves breaking services into small partitions which we call cells. By factoring services into cells, engineering teams can assess and thoroughly test recovery processes of even the largest service or subsystem. As S3 has scaled, the team has done considerable work to refactor parts of the service into smaller cells to reduce blast radius and improve recovery. During this event, the recovery time of the index subsystem still took longer than we expected. The S3 team had planned further partitioning of the index subsystem later this year. We are reprioritizing that work to begin immediately.

From the beginning of this event until 11:37AM PST, we were unable to update the individual services' status on the AWS Service Health Dashboard (SHD) because of a dependency the SHD administration console has on Amazon S3. Instead, we used the AWS Twitter feed (@AWScloud) and SHD banner text to communicate status until we were able to update the individual services' status on the SHD. We understand that the SHD provides important visibility to our customers during operational events and we have changed the SHD administration console to run across multiple AWS regions.

Finally, we want to apologize for the impact this event caused for our customers. While we are proud of our long track record of availability with Amazon S3, we know how critical this service is to our customers, their applications and end users, and their businesses. We will do everything we can to learn from this event and use it to improve our availability even further.

# Summary

“Amazon S3 Service Disruption in the Northern Virginia (US-EAST-1) Region”

# Impact

- “At 9:37AM PST...a larger set of servers was removed than intended”
- “Supported...the index subsystem...necessary to serve all GET, LIST, PUT, and DELETE requests ... [and] the placement subsystem... used during PUT requests”
- “S3 console, Amazon Elastic Compute Cloud (EC2) new instance launches, Amazon Elastic Block Store (EBS) volumes (when data was needed from a S3 snapshot), and AWS Lambda were also impacted while the S3 APIs were unavailable”
- The final S3 system “finished recovery at 1:54PM PST”
- Some other services “had accumulated a backlog of work during the S3 disruption and required additional time to fully recover.”

## Even more impact

“From the beginning of this event until 11:37AM PST, we were unable to update the individual services’ status on the AWS Service Health Dashboard (SHD) because of a dependency the SHD administration console has on Amazon S3. Instead, we used the **AWS Twitter feed** (@AWSCloud) and SHD banner text to communicate status until we were able to update the individual services’ status on the SHD.”

# Root cause

“An authorized S3 team member using an established playbook executed a command which was intended to remove a small number of servers for one of the S3 subsystems that is used by the S3 billing process. Unfortunately, one of the inputs to the command was entered incorrectly and a larger set of servers was removed than intended.”

# Timeline

- 9:37AM: Bad command entered, beginning of outage
- x:xxAM: Discovery from monitoring
- x:xxAM: Internal activities
- 11:37AM: Service Health Dashboard restored
- 12:26PM: Index subsystem has enough capacity to begin servicing S3 GET, LIST, and DELETE requests
- 1:18PM: Index subsystem restored and GET, LIST, and DELETE APIs were functioning normally
- 1:54PM: Placement subsystem finished recovery. End of PUT API outage
- x:xxPM: End of dependent service outage

# Analysis

## What went well:

- Twitter as backup to health dashboard
- (Other items probably omitted in this public disclosure.)

## Poorly:

- Able to take down the service with one command.
- Took a long time to restart systems.
- Didn't know it would take so long to restart.
- Health dashboard depends on S3, single-homed.
- ...



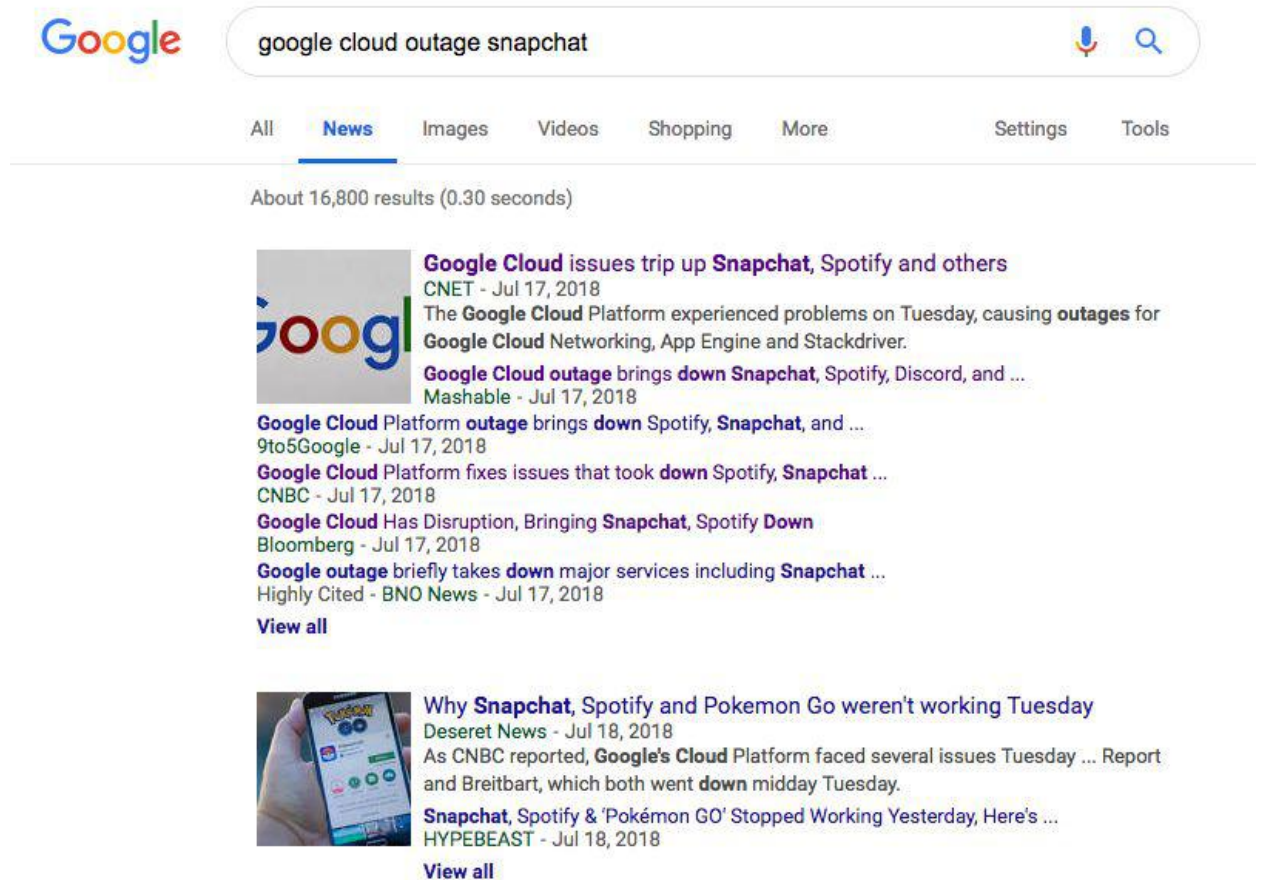
# Action items

- “The tool used allowed too much capacity to be removed too quickly. We have modified this tool to remove capacity more slowly and added safeguards to prevent capacity from being removed when it will take any subsystem below its minimum required capacity level.”
- “auditing our other operational tools to ensure we have similar safety checks.”
- “make changes to improve the recovery time of key S3 subsystems”
- “changed the SHD administration console to run across multiple AWS regions”

What is *not* in this postmortem?

“Unfortunately, one of the inputs to the command was entered incorrectly and a larger set of servers was removed than intended.”

# Bonus round




The screenshot shows a Google search interface with the query "google cloud outage snapchat". The search results are filtered by "News". The first result is from CNET, dated July 17, 2018, titled "Google Cloud issues trip up Snapchat, Spotify and others". The snippet mentions that the Google Cloud Platform experienced problems on Tuesday, causing outages for Google Cloud Networking, App Engine, and Stackdriver. Below this, there are several other news snippets from various sources like Mashable, 9to5Google, CNBC, Bloomberg, and BNO News, all reporting on the Google Cloud outage and its impact on services like Snapchat, Spotify, and Pokémon Go. A "View all" link is present at the bottom of the first set of results.

Google

google cloud outage snapchat

All News Images Videos Shopping More Settings Tools

About 16,800 results (0.30 seconds)

 **Google Cloud** issues trip up **Snapchat**, Spotify and others  
CNET - Jul 17, 2018  
The **Google Cloud** Platform experienced problems on Tuesday, causing **outages** for **Google Cloud** Networking, App Engine and Stackdriver.  
**Google Cloud outage** brings **down Snapchat**, Spotify, Discord, and ...  
Mashable - Jul 17, 2018


**Google Cloud** Platform **outage** brings **down** Spotify, **Snapchat**, and ...  
9to5Google - Jul 17, 2018

**Google Cloud** Platform fixes issues that took **down** Spotify, **Snapchat** ...  
CNBC - Jul 17, 2018

**Google Cloud** Has Disruption, Bringing **Snapchat**, Spotify **Down**  
Bloomberg - Jul 17, 2018

**Google outage** briefly takes **down** major services including **Snapchat** ...  
Highly Cited - BNO News - Jul 17, 2018

[View all](#)

 **Why Snapchat, Spotify and Pokemon Go weren't working Tuesday**  
Deseret News - Jul 18, 2018  
As CNBC reported, **Google's Cloud** Platform faced several issues Tuesday ... Report and Breitbart, which both went **down** midday Tuesday.  
**Snapchat**, Spotify & 'Pokémon GO' Stopped Working Yesterday, Here's ...  
HYPEBEAST - Jul 18, 2018

[View all](#)

<https://status.cloud.google.com/incident/cloud-networking/18012>

# Bringing it all together

- Title
- Summary
- Impact
- Root Cause
- Timeline (including beginning and ending of outage)
- What went well/poorly
- Action items

# Why are so many things broken?

# Things were always broken

- Complex systems are operating in multiple failure modes all the time.
- If you've ever stepped through your code to find a bug, you've probably noticed other bugs.
- The good news: Fixing all the things can help make the system more robust.

# Consider the cost of prevention

- Cost > benefit?
- Sometimes preventing errors is too expensive





# Small, informal postmortems

- Get into the postmortem mindset
- After anything that goes wrong, spend a moment to think, what went well, what would have gone better?
- In particular, “how could this have been prevented or detected automatically?”

# Asking others for postmortems

- When there is a customer-facing outage
- If another team is asking you to do something above and beyond what you normally do

# Retrospectives

- Just like a postmortem, but no outage
- What went well, what could go better?
- Usually just fix 1 or 2 things
- If you find they aren't helping, stop doing them
- A good way to deal with prior overzealous postmortem action items
  - “Is this step/system/activity still useful?”

# Invariants

# Invariants

Things that are always true...

# Invariants

Things that are always true...

- Conservation of Energy

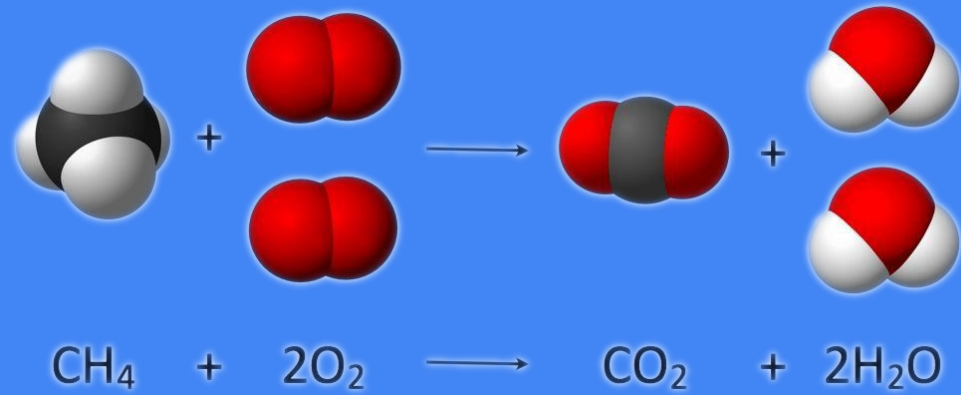
$$\text{Kinetic}_{\text{init}} + \text{Potential}_{\text{init}} = \text{Kinetic}_{\text{final}} + \text{Potential}_{\text{final}}$$



# Invariants

Things that are always true...

- Conservation of Energy
- Conservation of Mass

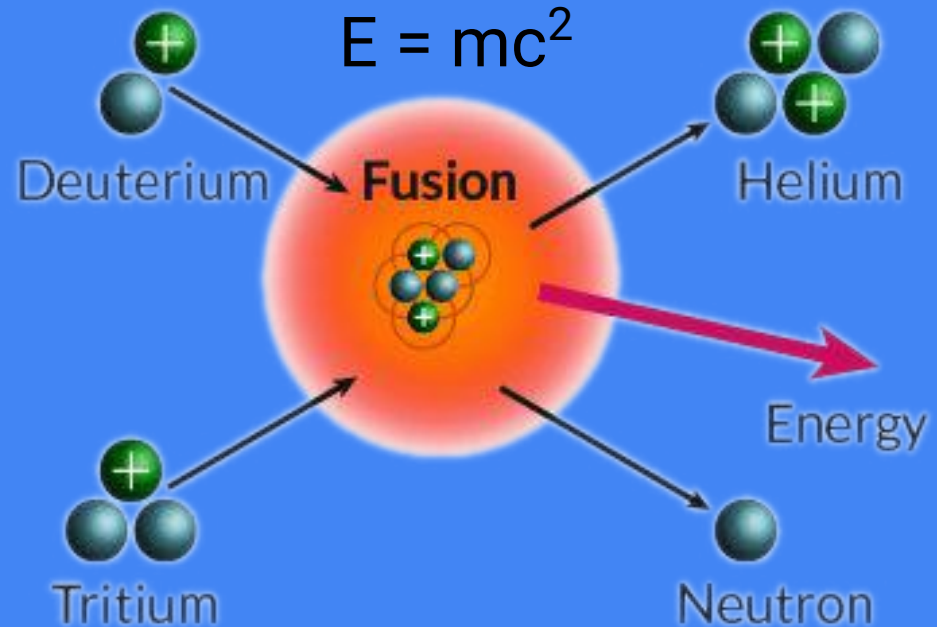


# Invariants

Things that are always true...

- Conservation of Energy
- Conservation of Mass

What if these invariants are violated?





# Invariants

Things that are always true...

- Conservation of Energy
- Conservation of Mass

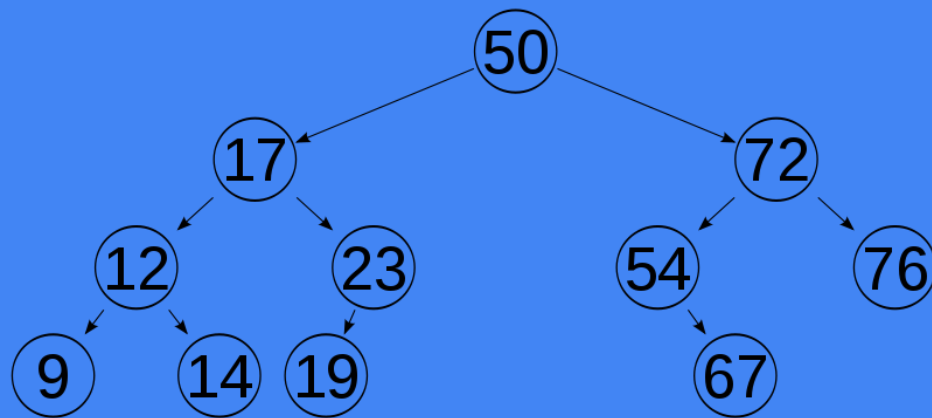


# Invariants

Things that are always true...

- Conservation of Energy
- Conservation of Mass
- Binary Search Property

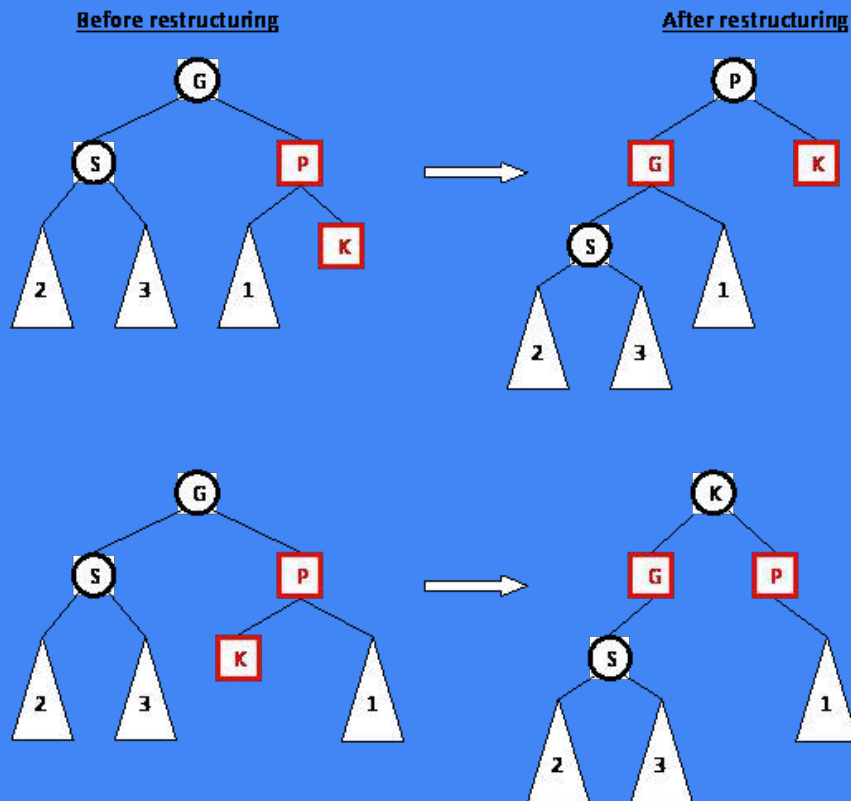
Left  $\leq$  self  $\leq$  Right



# Invariants

Things that are always true...

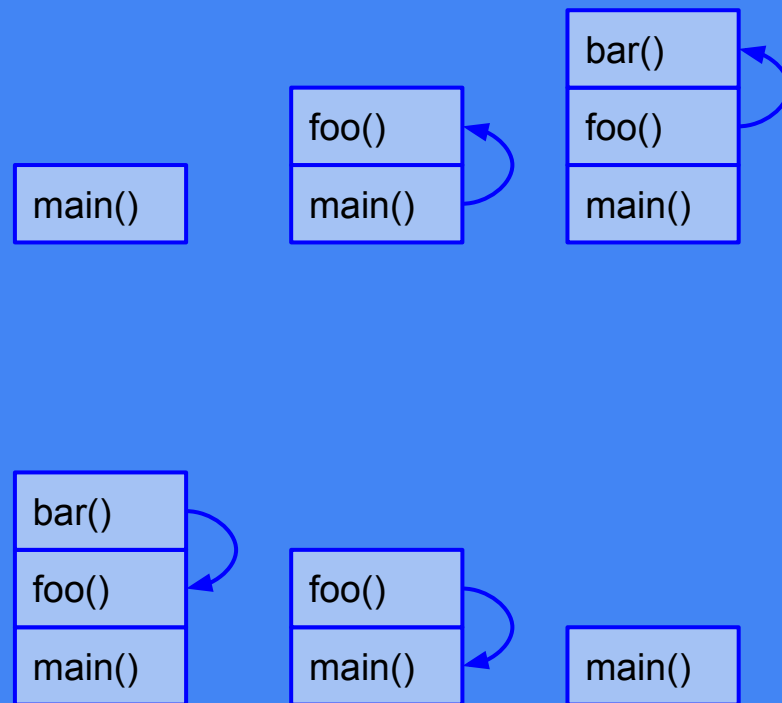
- Conservation of Energy
- Conservation of Mass
- Binary Search Property
- Red-Black Tree Properties
  - Root is black
  - Every leaf is black
  - If a node is red, then both its children are black
  - Every path from a given node to leaves traverses the same number of black nodes



# Invariants

Things that are always true...

- Conservation of Energy
- Conservation of Mass
- Binary Search Property
- Red-Black Tree Properties
- Stack Height
  - End of Program: restores height to 0
  - End of Function: restores to previous height

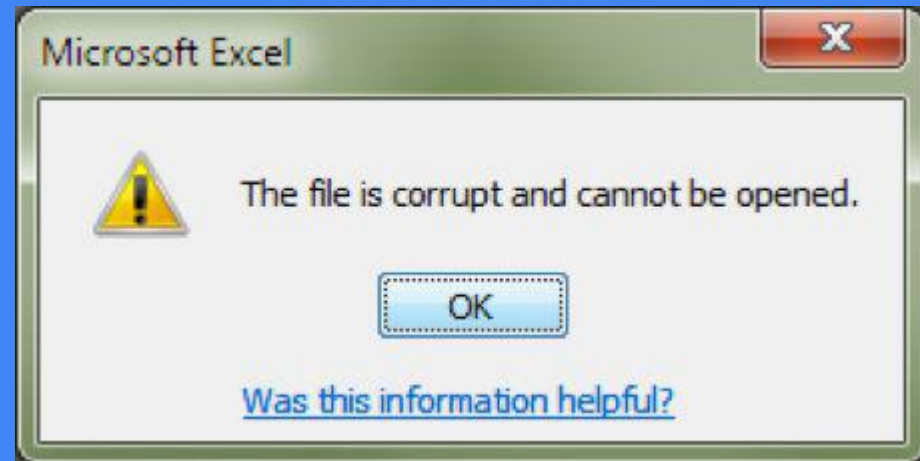


# Invariants

Things that are always true...

- Conservation of Energy
- Conservation of Mass
- Binary Search Property
- Red-Black Tree Properties
- Stack Height

## What if these invariants are violated?



A problem has been detected and Windows has been shut down to prevent damage to your computer.

**IRQL\_NOT\_LESS\_OR\_EQUAL**

An invariant or simply a failed precondition?

If this is the first time you've seen this Stop error screen, restart your computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical Information:

\*\*\* STOP: 0x00000001 (0x00000001, 0x00000001, 0x00000000, 0x00000000)

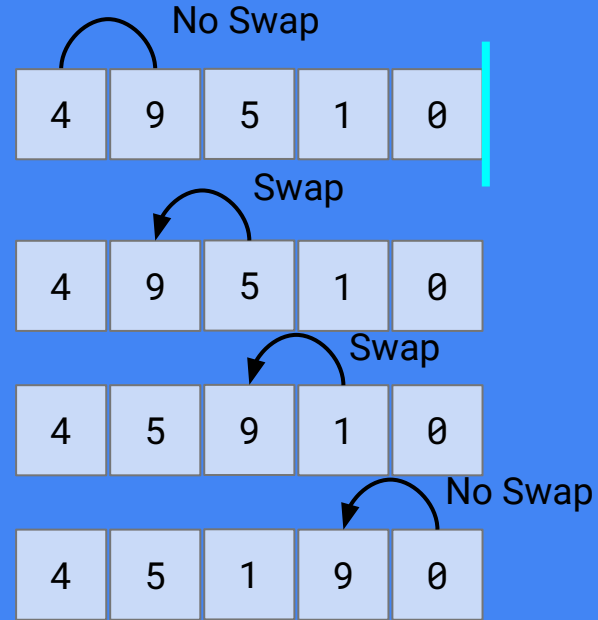
# Invariants

That you have used...

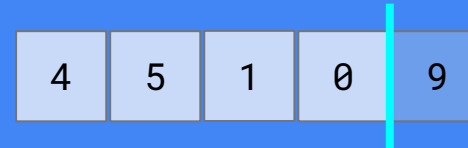
- Algorithm analysis (bubble sort)

After  $i$  passes, the last  $i$  elements are sorted. So after  $n$  passes all  $n$  elements are sorted.

## 1st pass



## 2nd pass

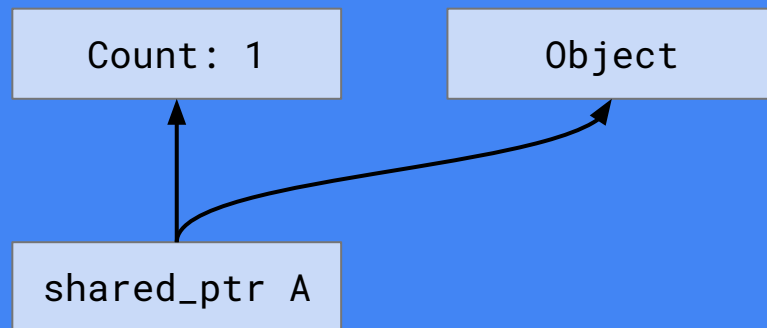


# Invariants

That you have used...

- Algorithm analysis (bubble sort)
- `shared_ptr<>`

The underlying object always exists as long as the reference count  $> 0$ .



Allocate object and a counter for it

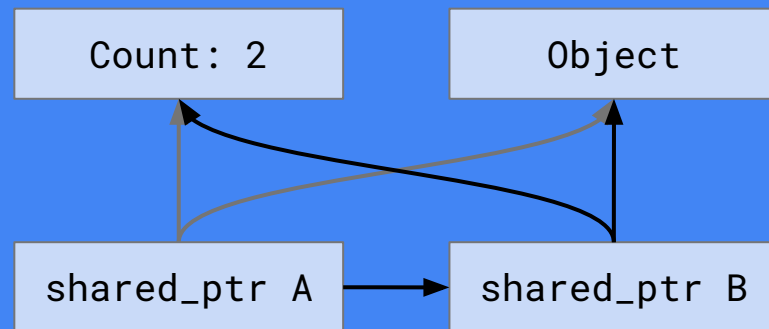


# Invariants

That you have used...

- Algorithm analysis (bubble sort)
- `shared_ptr<>`

The underlying object always exists as long as the reference count  $> 0$ .



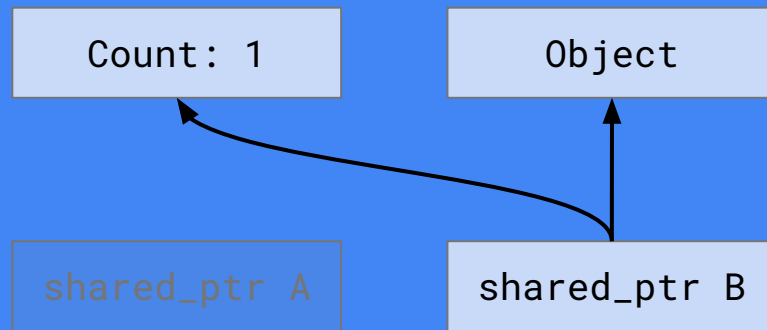
Copy the pointer, increment count

# Invariants

That you have used...

- Algorithm analysis (bubble sort)
- `shared_ptr<>`

The underlying object always exists as long as the reference count  $> 0$ .



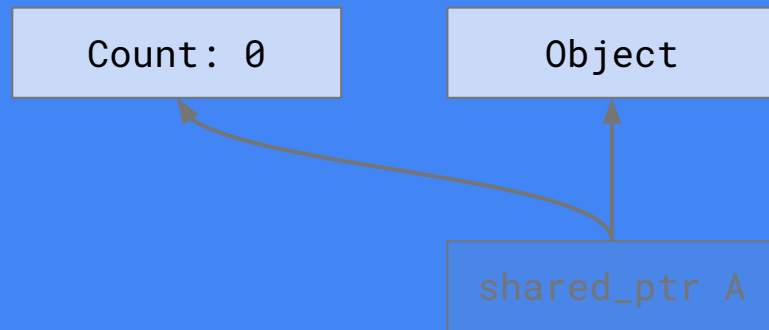
Deallocate ptr A, decrement count

# Invariants

That you have used...

- Algorithm analysis (bubble sort)
- `shared_ptr<>`

The underlying object always exists as long as the reference count  $> 0$ .



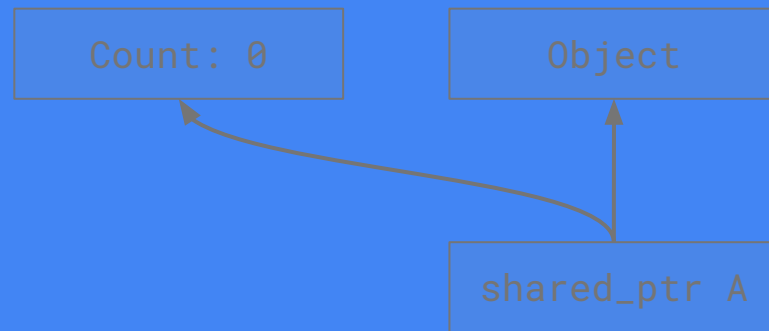
Deallocate ptr B, decrement count

# Invariants

That you have used...

- Algorithm analysis (bubble sort)
- `shared_ptr<>`

The underlying object always exists as long as the reference count  $> 0$ .



Count reaches 0, deallocate Object

# Invariants

That you have used...

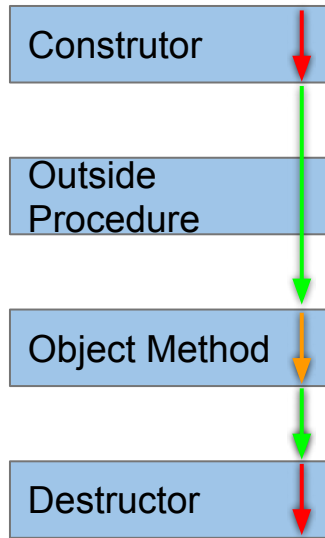
- Algorithm analysis (bubble sort)
- `shared_ptr<>`
- RAI

If socket exists, it is open.

Constructor acquires resource

Destructor relinquishes resource

```
struct Socket {  
    Socket() {  
        if (!open())  
            throw exception("Failed to open")  
    }  
    ~Socket() { close(); }  
}  
  
int main() {  
    {  
        // socket does not exist  
        Socket s;  
        // socket exists, and must also be open  
        // else would have failed construction  
        doSomething();  
        // socket still open  
    }  
    // socket no longer exists  
}
```



Invariant Holds

Invariant Temporarily Broken

Invariant Doesn't Hold

# Lifecycle Pattern

- *Public* methods do not break an object's invariant

OK, to break them internally  
but *must* restore them at completion

*Thread-safety* is the ability of an object to maintain its invariants in multi-threaded environment

# Application of Invariants

Compiler optimizations

# Code Optimization

Don't repeatedly run calculations that you only have to run once.

## Loop-invariant code motion

```
int i = 0;
while (i < n) {
    x = y + z;
    A[i] = 6 * i + x * x;
    ++i;
}
```



# Code Optimization

Don't repeatedly run calculations that you only have to run once.

Does the value of  $x$  ever change?

## Loop-invariant code motion

```
int i = 0;
while (i < n) {
    x = y + z;
    A[i] = 6 * i + x * x;
    ++i;
}
```

# Code Optimization

Don't repeatedly run calculations that you only have to run once.

Does the value of  $x$  ever change?

No. Let's move it out of the loop!

## Loop-invariant code motion

```
int i = 0;
while (i < n) {
    x = y + z;
    A[i] = 6 * i + x * x;
    ++i;
}
```

# Code Optimization

Don't repeatedly run calculations that you only have to run once.

Does the value of  $x$  ever change?

No. Let's move it out of the loop!

## Loop-invariant code motion

```
int i = 0;

x = y + z;

while (i < n) {
    A[i] = 6 * i + x * x;
    ++i;
}
```

# Code Optimization

Don't repeatedly run calculations that you only have to run once.

Does the value of  $x$  ever change?

No. Let's move it out of the loop!

What if the condition was false?

## Loop-invariant code motion

```
int i = 0;

x = y + z;

while (i < n) {

    A[i] = 6 * i + x * x;

    ++i;

}
```

# Code Optimization

Don't repeatedly run calculations that you only have to run once.

Does the value of  $x$  ever change?

No. Let's move it out of the loop!

What if the condition was false?

Loop would not have executed!  
 $x$  would not have changed value

## Loop-invariant code motion

```
int i = 0;

x = y + z;

while (i < n) {

    A[i] = 6 * i + x * x;

    ++i;

}
```

# Code Optimization

Don't repeatedly run calculations that you only have to run once.

Does the value of  $x$  ever change?

No. Let's move it out of the loop!

What if the condition was false?

Loop would not have executed!  
 $x$  would not have changed value

## Loop-invariant code motion

```
int i = 0;
if (i < 0) {
    x = y + z;
    do {
        A[i] = 6 * i + x * x;
        ++i;
    } while (i < n);
}
```

# Code Optimization

Don't repeatedly run calculations that you only have to run once.

Invariant:

Values in variables *before* optimization

==

Values in variables *after* optimization

## Loop-invariant code motion

```
int i = 0;
if (i < 0) {
    x = y + z;
    do {
        A[i] = 6 * i + x * x;
        ++i;
    } while (i < n);
}
```

# Code Optimization

Don't repeatedly run calculations that you only have to run once.

If  $x$  never changes

Then  $x * x$  also never changes

## Loop-invariant code motion

```
int i = 0;
if (i < 0) {
    x = y + z;
    do {
        A[i] = 6 * i + x * x;
        ++i;
    } while (i < n);
}
```



# Code Optimization

Don't repeatedly run calculations that you only have to run once.

If  $x$  never changes

Then  $x * x$  also never changes

## Loop-invariant code motion

```
int i = 0;
if (i < 0) {
    x = y + z;
    int const x2 = x * x;
    do {
        A[i] = 6 * i + x2;
        ++i;
    } while (i < n);
}
```

# Code Optimization

Replace expensive operation with  
*equivalent* but less expensive operation

## Strength Reduction

```
int i = 0;
if (i < 0) {
    x = y + z;
    int const x2 = x * x;
    do {
        A[i] = 6 * i + x2;
        ++i;
    } while (i < n);
}
```

# Code Optimization

Replace expensive operation with  
*equivalent* but less expensive operation

```
6 * i
= (4 + 2) * i
= 4 * i + 2 * i
= (i << 2) + (i << 1)
```

# Strength Reduction

```
int i = 0;
if (i < 0) {
    x = y + z;
    int const x2 = x * x;
    do {
        A[i] = 6 * i + x2;
        ++i;
    } while (i < n);
}
```

# Code Optimization

Replace expensive operation with  
*equivalent* but less expensive operation

```
6 * i
= (4 + 2) * i
= 4 * i + 2 * i
= (i << 2) + (i << 1)
```

# Strength Reduction

```
int i = 0;
if (i < 0) {
    x = y + z;
    int const x2 = x * x;
    do {
        int const i6 = i<<2 + i<<1;
        A[i] = i6 + x2;
        ++i;
    } while (i < n);
}
```

# Code Optimization

Replace expensive operation with  
*equivalent* but less expensive operation

$A[i] = A + \text{sizeof}(\text{Atype}) * i$

So could do another strength reduction  
Esp. if  $\text{sizeof}(\text{Atype})$  is power of 2

# Strength Reduction

```
int i = 0;
if (i < 0) {
    x = y + z;
    int const x2 = x * x;
    do {
        int const i6 = i<<2 + i<<1;
        A[i] = i6 + x2;
        ++i;
    } while (i < n);
}
```

# Code Optimization

Replace expensive operation with  
*equivalent* but less expensive operation

$A[i] = A + \text{sizeof}(\text{Atype}) * i$

So could do another strength reduction  
Esp. if  $\text{sizeof}(\text{Atype})$  is power of 2

# Strength Reduction

```
int i = 0;
if (i < 0) {
    x = y + z;
    int const x2 = x * x;
    do {
        int const i6 = i<<2 + i<<1;
        A[i] = i6 + x2;
        ++i;
    } while (i < n);
}
```

# Code Optimization

Replace expensive operation with  
*equivalent* but less expensive operation

$A[i] = A + \text{sizeof}(\text{Atype}) * i$

So could do another strength reduction  
Esp. if  $\text{sizeof}(\text{Atype})$  is power of 2

But I only expand that for simplicity  
... to see the next optimization

# Strength Reduction

```
int i = 0;
if (i < 0) {
    x = y + z;
    int const x2 = x * x;
    do {
        int const i6 = i<<2 + i<<1;
        *(A + Asize*i) = i6 + x2;
        ++i;
    } while (i < n);
}
```

# Code Optimization

Replace expensive operation with  
*equivalent* but less expensive operation

$A[i] = A + \text{sizeof}(\text{Atype}) * i$

So could do another strength reduction  
Esp. if  $\text{sizeof}(\text{Atype})$  is power of 2

But I only expand that for simplicity  
... to see the next optimization

# Strength Reduction

```
int i = 0;
if (i < 0) {
    x = y + z;
    int const x2 = x * x;
    do {
        int const i6 = i<<2 + i<<1;
        *(A + Asize*i) = i6 + x2;
        ++i;
    } while (i < n);
}
```



# Code Optimization

Remove *unnecessary* variables

i6 moves in lock-step with i

## Induction Variable

```
int i = 0;
if (i < 0) {
    x = y + z;
    int const x2 = x * x;
    do {
        int const i6 = i<<2 + i<<1;
        *(A + Asize*i) = i6 + x2;
        ++i;
    } while (i < n);
}
```

# Code Optimization

Remove *unnecessary* variables

i6 moves in lock-step with i

## Induction Variable

```
int i = 0;
if (i < 0) {
    x = y + z;
    int const x2 = x * x;
    int i6 = i;
    do {
        *(A + Asize*i) = i6 + x2;
        i6 += 6;
        ++i;
    } while (i < n);
}
```

# Code Optimization

Remove *unnecessary* variables

i6 moves in lock-step with i

A[i] = increments a pointer  
by Asize each iteration

## Induction Variable

```
int i = 0;
if (i < 0) {
    x = y + z;
    int const x2 = x * x;
    int i6 = i;
    do {
        *(A + Asize*i) = i6 + x2;
        i6 += 6;
        ++i;
    } while (i < n);
}
```

# Code Optimization

Remove *unnecessary* variables

`i6` moves in lock-step with `i`

`A[i]` = increments a pointer  
by `Asize` each iteration

## Induction Variable

```
int i = 0;
if (i < 0) {
    x = y + z;
    int const x2 = x * x;
    int i6 = i;
    do {
        *A = i6 + x2;
        i6 += 6;
        A += Asize;
        ++i;
    } while (i < n);
```

# Code Optimization

Remove *unnecessary* variables

i6 moves in lock-step with i

A[i] = increments a pointer  
by Asize each iteration

Now i has become unnecessary

## Induction Variable

```
int i = 0;
if (i < 0) {
    x = y + z;
    int const x2 = x * x;
    int i6 = i;
    do {
        *A = i6 + x2;
        i6 += 6;
        A += Asize;
        ++i;
    } while (i < n);
```

# Code Optimization

Remove *unnecessary* variables

`i6` moves in lock-step with `i`

`A[i]` = increments a pointer  
by `Asize` each iteration

Now `i` has become unnecessary

Still keep `i6` because can't alter `i`  
*Can't affect outside scope*

## Induction Variable

```
int i = 0;
if (i < 0) {
    x = y + z;

    int const x2 = x * x;

    int i6 = i, const n6 = 6 * n;
    do {
        *A = i6 + x2;

        i6 += 6;

        A += Asize;
    } while (i6 < n6);
}
```

# Should I write this code?

```
int i = 0;
if (i < 0) {
    x = y + z;
    int const x2 = x * x;
    int i6 = i, const n6 = 6 * n;
    do {
        *A = i6 + x2;
        i6 += 6;
        A += Asize;
    } while (i6 < n6);
}
```

# Should I write this code?

**No!**

The compiler does this for me.

We'd rather have readability

## No! Unreadable code

```
int i = 0;
if (i < 0) {
    x = y + z;
    int const x2 = x * x;
    int i6 = i, const n6 = 6 * n;
    do {
        *A = i6 + x2;
        i6 += 6;
        A += Asize;
    } while (i6 < n6);
}
```



# Should I write this code?

**No!**

The compiler does this for me.

We'd rather have readability

One exception:

Performance-critical section

*AND* the compiler isn't optimizing

*AND* the code has been profiled

## No! Unreadable code

```
int i = 0;
if (i < 0) {
    x = y + z;
    int const x2 = x * x;
    int i6 = i, const n6 = 6 * n;
    do {
        *A = i6 + x2;
        i6 += 6;
        A += Asize;
    } while (i6 < n6);
}
```

# A Personal Tale (Eric's PhD)

From FlowCore to JitFlow:

Improving the speed of Information Flow in JavaScript.

# Goal: Track Information Flow

Tag values with a “label” indicating what contributed to the construct of the value

## Problem:

Track that assignments inside a loop depend on the condition

```
function stealpin() {  
    for (var i=0; i < 10000; ++i) {  
        if (i == pin) break;  
    }  
    return i;  
}
```

Returns the same value as the pin!

# Goal: Track Information Flow

## Problem:

Track that assignments inside  
a loop depend on the condition

## Technique:

1. Label on the program counter
2. Track entry/exit of loops on a stack of labels (think: function call stack)

```
function stealpin() {  
    for (var i=0; i < 10000; ++i) {  
        if (i == pin) break;  
    }  
    return i;  
}
```

Returns the same value as the pin!  
But also carries the same label!

## Technique:

Track entry/exit of loops on a stack  
(just like functions)

## Implementation:

Instrument new bytecodes:

- `dup_cflabel`
- `join_cflabel`
- `popj_cflabel`

```
function stealpin() {  
    for (var i=0; i < 10000; i++) {  
        if (i == pin) break;  
    }  
    return i;  
}
```

```
[00]  enter  
  
[02]  mov          r0, Int32: 0  
[05]  jmp          27(->32)  
  
[08]  resolve_global r1, pin(@id0)  
[13]  eq          r1, r0, r1  
  
[19]  jfalse      r1, 8(->27)  
  
[25]  jmp          16(->41)  
  
[30]  pre_inc     r0  
[32]  less       r1, r0, Int32: 10000  
  
[38]  loop_if_true r1, -31(->7)  
  
[44]  ret         r0
```

## Problem!

Sometimes inserted incorrectly!

**Solution:** Abstract interpreter

Compute the stack height at each bytecode.

## Invariants:

- End of function: height == 0
  - [44]
- Fall through matches height
  - [32, 41]
- Jump target matches height
  - [5->32, 19->27, 25->41, 38->7]
- Pop + Join have req. Height
  - [22]

```
function stealpin() {  
    for (var i=0; i < 10000; i++) {  
        if (i == pin) break;  
    }  
    return i;  
}
```

```
[00] enter  
[01] dup_cflabel  
[02] mov                r0, Int32: 0  
[05] jmp                27(->32)  
[07] dup_cflabel  
[08] resolve_global    r1, pin(@id0)  
[13] eq                 r1, r0, r1  
[17] join_cflabel      r1  
[19] jfalse             r1, 8(->27)  
[22] popj_cflabel      pop:1, join:2  
[25] jmp                16(->41)  
[27] popj_cflabel      pop:1, join:0  
[30] pre_inc           r0  
[32] less              r1, r0, Int32: 10000  
[36] join_cflabel      r1  
[38] loop_if_true      r1, -31(->7)  
[41] popj_cflabel      pop:1, join:0  
[44] ret               r0
```

# Database Invariants

# Database Invariants

- Every ad must have a campaign
- Every customer must have...
- Every click/view must have...
- Every log message must have...
- No record with X may have Y
- ....

Account	Debit	Credit
Cash	1,000	
Inventory		500
Cost of Goods	500	
Revenue		800
Deferred Revenue		200

SUM(Debit) == SUM(Credit)



# Invariants: Check Them!

- Write some SQL run it periodically.  
Expect that it should be empty, if not then it contains the violating data.
- Fire an alert, send an email  
Teams do this at Google.  
It catches problems quickly:
  - During integration tests  
(prevents bad code commit)
  - After a release  
(prevents it from getting worse)
  - Sidecar job created incorrect entries

Account	Debit	Credit
Cash	1,000	
Inventory		500
Cost of Goods	560	
Revenue		800
Deferred Revenue		200

Transaction Debit: \$1500

Unloading the Box

An error was encountered.  
The data is invalid.

OK

# Invariants: Check Them!

- SQL doc generator can auto-produce invariant descriptions
- Engineers can write code on those assumptions and remain safe
- **Don't go too far...**
  - **Can't usually fix upstream problems**

Is that field always filled out?

Where'd that NULL come from?

Why doesn't this ID join with the other table?

Are some records missing?

About 5% of these entries don't make sense!

Complex  
accessibility

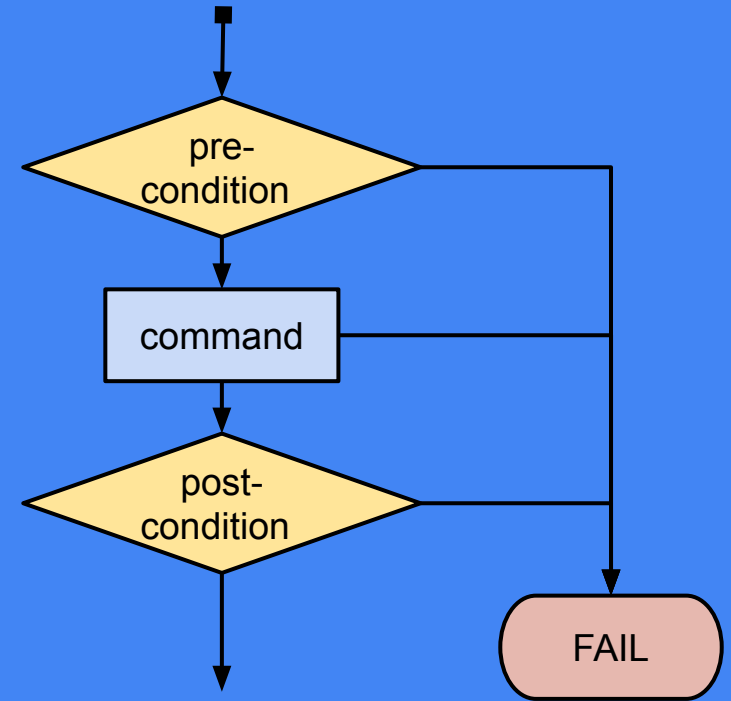
No pre-defined  
schema

Flexible  
analysis

**DATA LAKE**

# Lesson Learned:

- Invariants are system-wide
  - Core part of the system design
  - Document them!
- Relations that *always* hold true
- Enforce with Pre/Post-Conditions around every code block that touches the database
- Appear as assumptions *safely* made in many places of the application
  - not co-located
  - but are coordinated by design



<https://bit.ly/3wGfgav>

- Tweet: What automated monitoring would improve your life?
- Tweet: What invariant(s) does your webserver have?

