Site Reliability engineering / Emergency Procedures

Context: Launching new features vs reliability

- Product teams want to move fast
  - Deploy as many features as possible
  - Deploy as fast as possible
- Reliability teams want to be stable
  - Focus on reliability
  - Break as little as possible

Goal of SREs is to enable the product teams to move as fast as practically possible without impacting users.

This talk

Users are having issues with your service

- What does “having issues with your service” mean ?
- How do we know if something went wrong ?
- What went wrong ?
- What action to take ?
- How to prevent these from occurring in future ?

Availability

Definition varies

- % Uptime
- % Requests served

Path towards high availability

- Add redundancy (N+2)
  - More replicas - webservers, database
  - Load Balancers
- Write more tests : integration tests
- More humans to respond to pagers
- Maintain checklist of possible bugs, and run through the list for each release
- Reduce release velocity
  - Release one feature at a time
  - Release a binary only after exhaustive automated & human testing

Quiz:
I want my service availability to be … ?

- ❑ 97%
- ❑ 99.9%
- ❑ 99.999%
- ❑ 100%

Right answer

It depends.

Trade-off between launch velocity and system stability.

- A feature with **low availability** would be unusable - always broken
- A feature with very **high availability** would be unusable - very boring, features missing

Deciding on right availability target

- Infrastructure (shared service) vs user-facing service
- Mature Product vs New Product
- Paid vs Free Product
- High Revenue impact vs Low

Basically, it’s a very logical cost-benefit analysis.

Service Level Indicators (SLI), Service Level Objectives (SLO)

Service Level Indicators

What are the metrics that matter ?

Latency

Throughput : Number of requests processed per second

Error Rate : % of requests answered

It is important that these metrics are clearly thought-about and mutually agreed upon.

Service Level Objectives (SLOs)

What are the target values for your SLIs

Latency - 300ms.

Often, it is **hard to define SLOs** for some metrics.
In those cases:

- It is important to have a **hard upper limit**
- Enables abnormal behavior detection
- Gives a guiding benchmark to teams

Expected downtime

99% = 4 days of downtime a year

99.9% = 9 hours of downtime

99.99% = 1 hour of downtime

99.999% = 5 min of downtime

Error Budget

Ex: Target Availability 99.9%

Practically, the service is available 99.99% time.

The service has gained some error budget to launch new features faster now !

Likewise, vice-versa.

Reliable only up to the defined limit, no more

- Sometimes, systems may give false impression of being over-reliable
- Leads other teams to build with false assumptions

Extreme measure : Planned Downtime

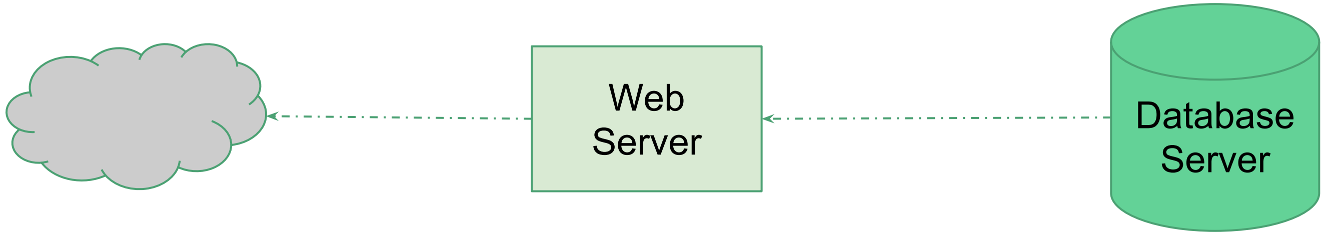
Google's Forced Downtime

Google's internal lock service is called Chubby; an open source clone is called Zookeeper. Chubby's uptime was so perfect that engineers designing systems that relied on Chubby starting writing code that assumed Chubby could not go down. This led to cascading failures when Chubby did have an outage.

To solve this problem, Google management decreed that if Chubby had zero downtime in a given month, it would be taken down intentionally for five minutes. This would assure that error handling code was exercised regularly.

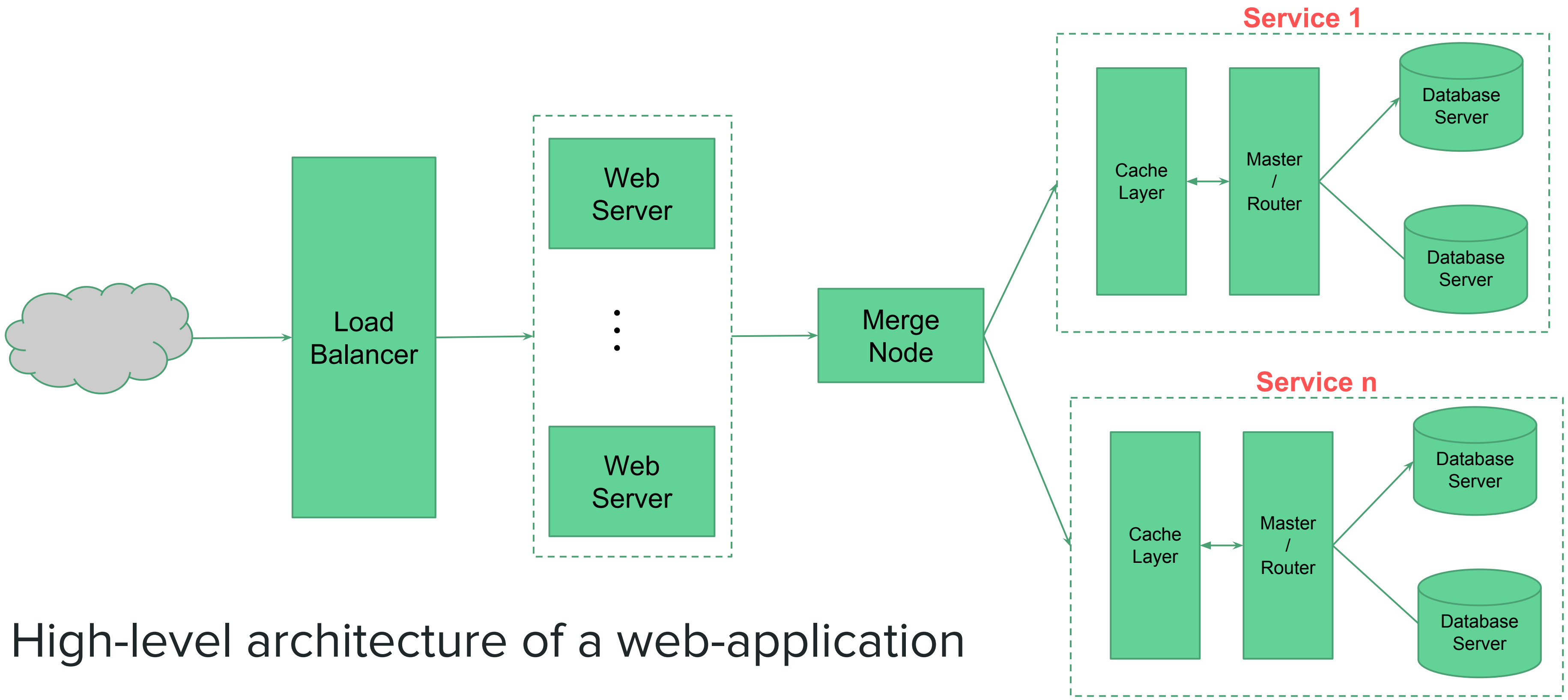
Abnormality Detection

It would have been easy if ...



A simple web application

But most often in reality



High-level architecture of a web-application

Monitoring

Without monitoring ...

- No way to know if my system is working, it is like **flying in the dark**
- Predicting a future issue, is not at all possible

Good monitoring

- Enables analysis of long-term trends
- Enables comparative analysis
    - 1 Machine with 32GB RAM vs 2 machines with 16 GM RAM each
- Enables conducting ad hoc analysis - latency shot up, what else changed ?

Good monitoring

- Never requires a human to interpret an alert
  - Humans add latency
  - Alert fatigue
  - Ignore, if alarms are usually false
- 3 kinds of monitoring output
  - **Alert** : True alarm - red alert. Human needs to take action.
  - **Ticket** : Alarm - yellow alert. Human needs to take action eventually
  - **Logging** : Not a alarm - no action needed.

Good Alerts

- **Alert is a sign that something is actually broken**

Bad alert : Average latency is 125ms.

Good alert: Average latency is breaking latency SLOs

- **Alerts MUST be actionable.**

- **Alert for symptoms, not causes**

What = Symptom, Why = (Possible) Cause

“What” versus “Why” is an important distinction between good monitoring with maximum signal and minimal noise.

Types of monitoring

Black box monitoring

- Mimics a system user
- Shows how your users view your system

White box monitoring

- Viewing your system from the inside, knowing very well how it functions
- For e.g each service keeps counters of all backend service failures, latencies

Black-box monitoring example

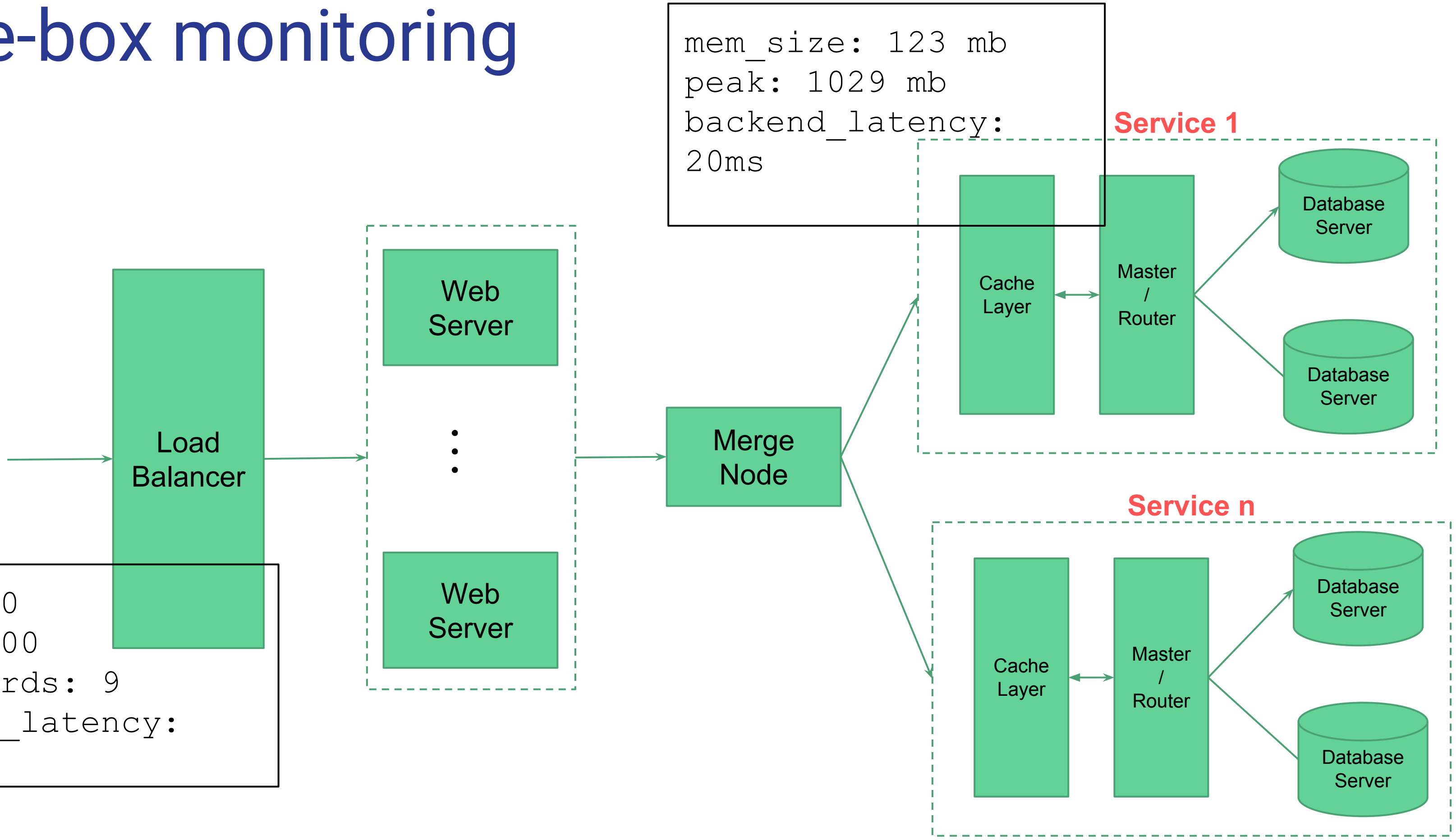
```
Request

GET /mysite.html HTTP/1.1
Host: www.example.com

Response
HTTP/1.1 200 OK
Date: 01 Jan 1900
Content Type: text/html

<html> … </html>
```

White-box monitoring

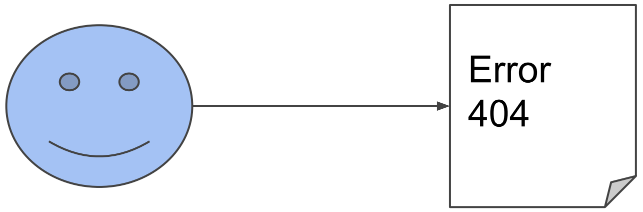


```
mem_size: 123 mb
peak: 1029 mb
backend_latency:
20ms
```

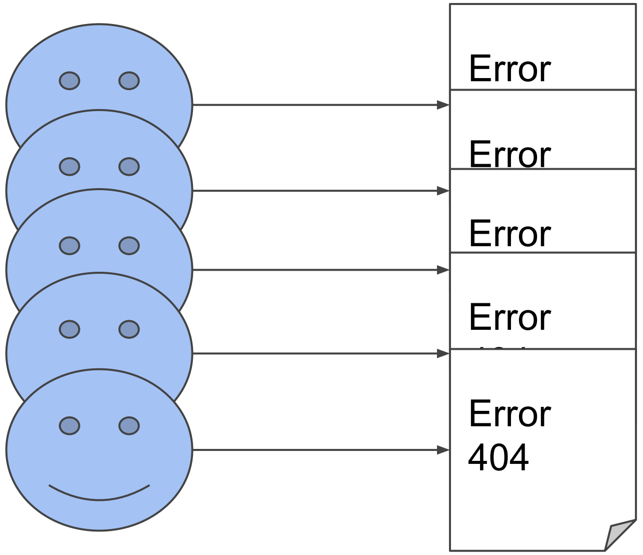
```
qps: 100
peak: 500
num_shards: 9
backend_latency:
200ms
```

Life of an incident

10.59 PM



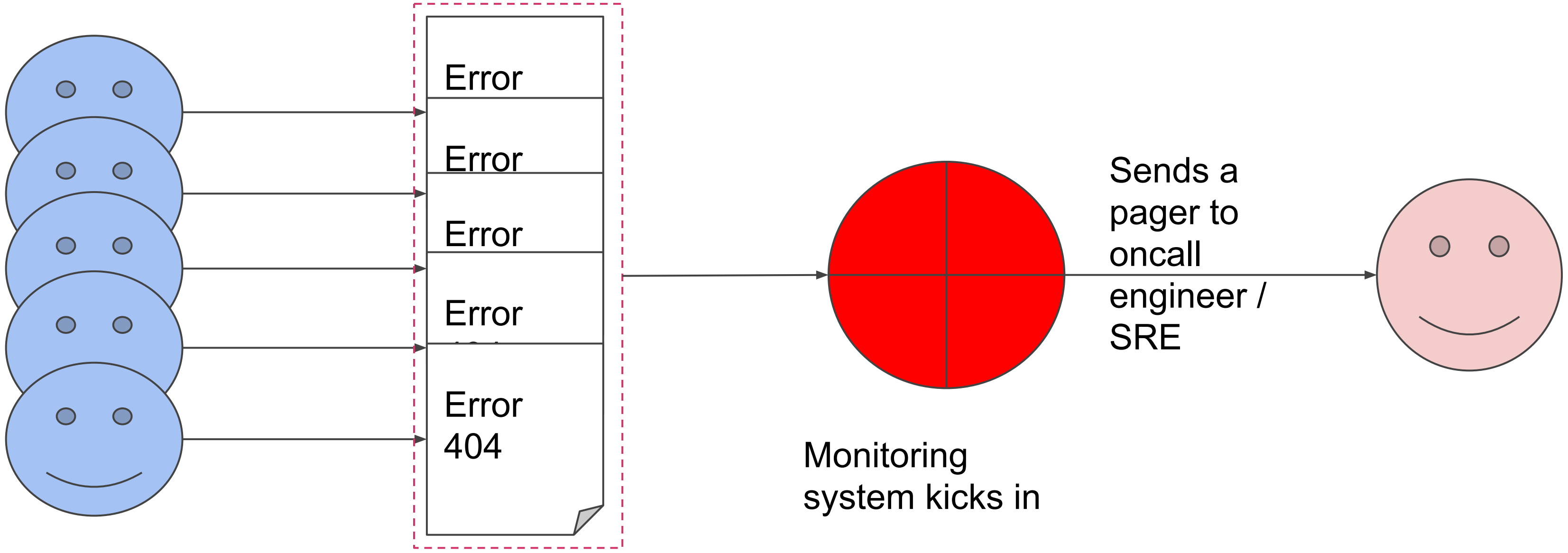
11.00 PM



10 users get 404 error in 1 minute.

11.00 PM

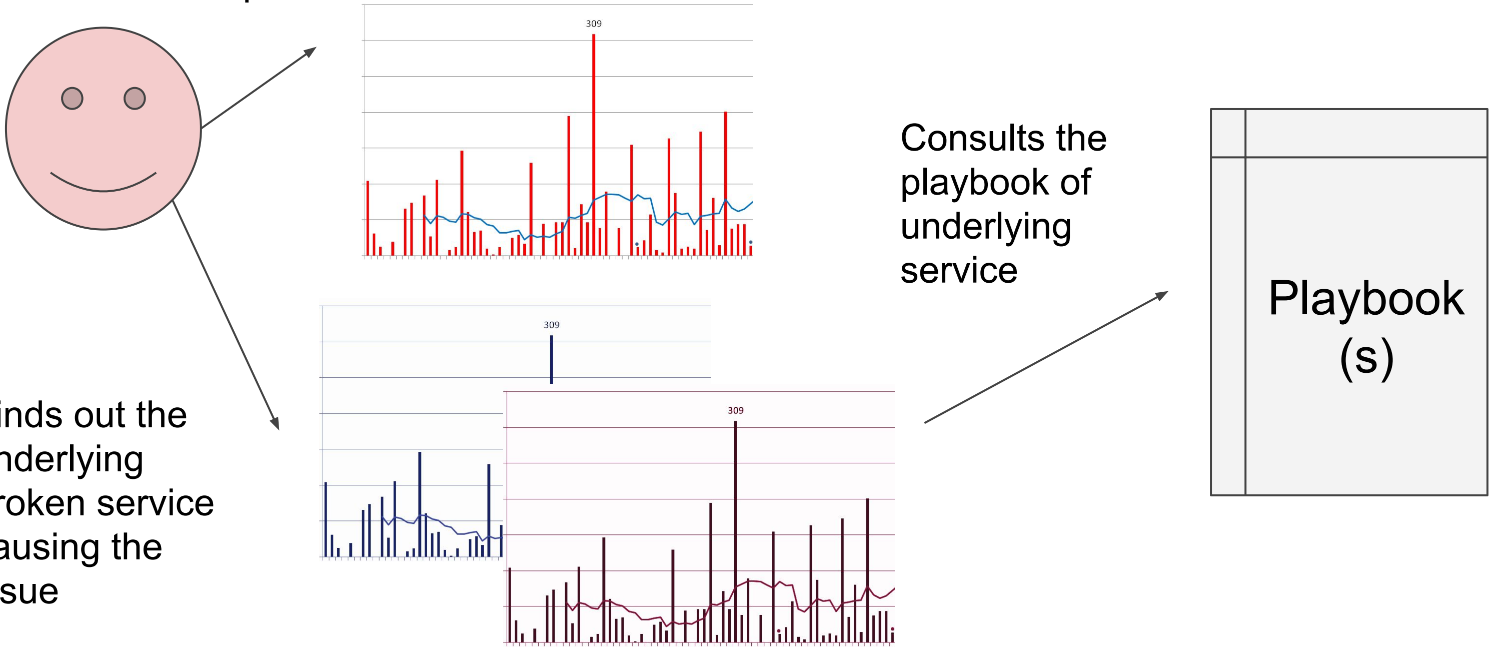
In this case, black box monitoring kicked off an alert.
This is after the fact alert.
Sometimes, white-box monitoring can help predict an issue is about to happen.


Error
Error
Error
Error
Error
404
Sends a pager to oncall engineer / SRE
Monitoring system kicks in

10 users get 404 error in 1 minute.

11.05 PM

White-box monitorning invariably plays a big role in figuring out the origin of the problem.
Also, a good central logging system used by ALL servers throughout the system.



Verifies the user impact

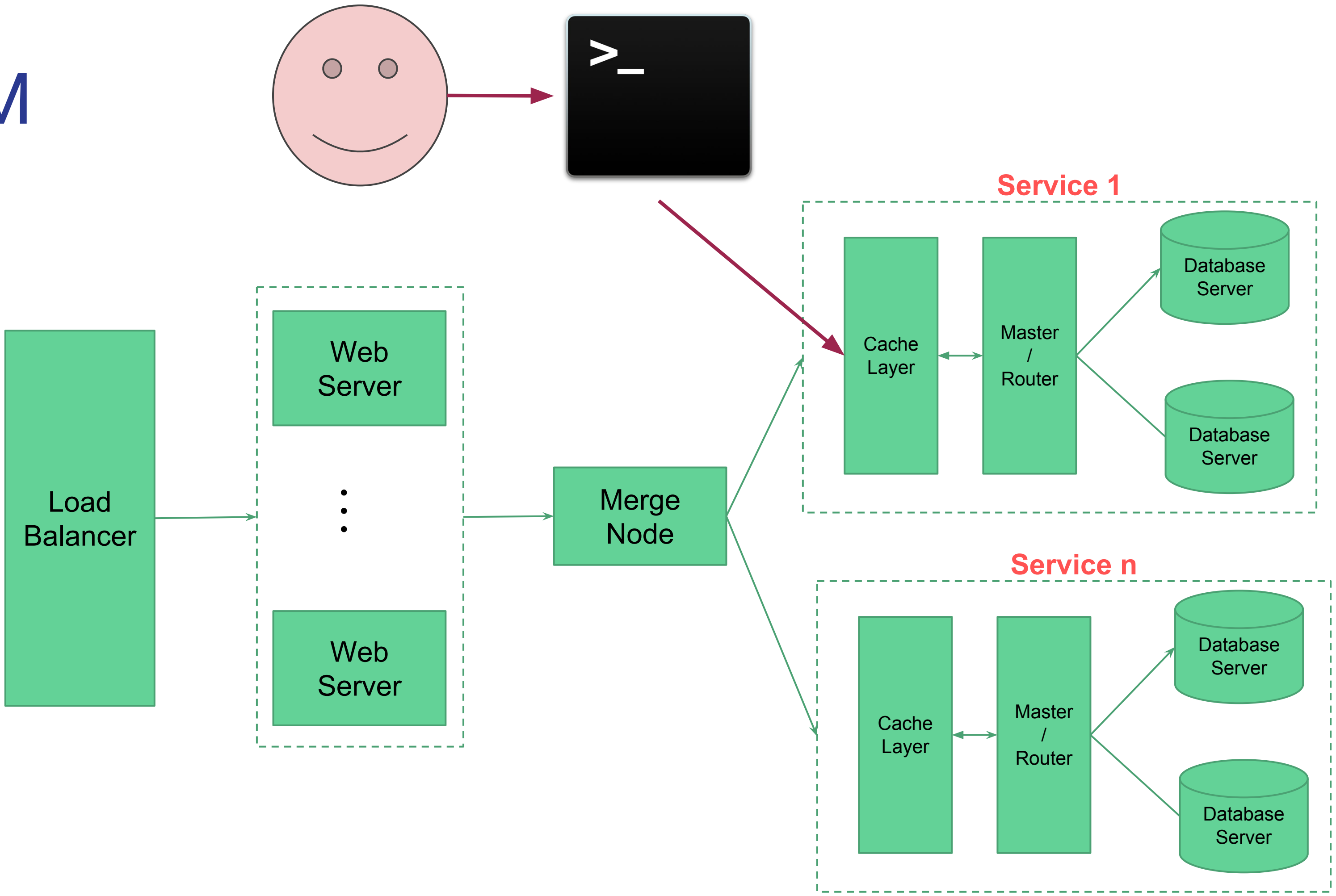
Finds out the underlying broken service causing the issue

Consults the playbook of underlying service

Finds how to disable a particular feature

11.10 PM

Issue fixed



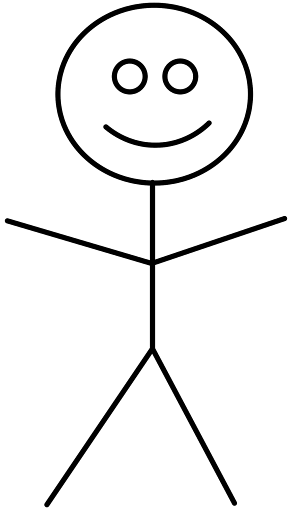
In summary

- **Playbooks**: guide to turning off features, feature flag names & descriptions
- **Mean time to failure** (MTTF)
- **Mean time to recovery** (MTTR)

Reliability = function (MTTF, MTTR)

- Statistically, having playbooks improves MTTR by 3x.

Culture of SREs



Avoid a culture of heros.

John is a hero.

John knows about the system more than others.

John can fix this issue in under a minute.

John doesn’t need to read docs to fix it.

But there is only ONE John.

And John needs to sleep, eat …

And John can solve only ONE issue at a time.

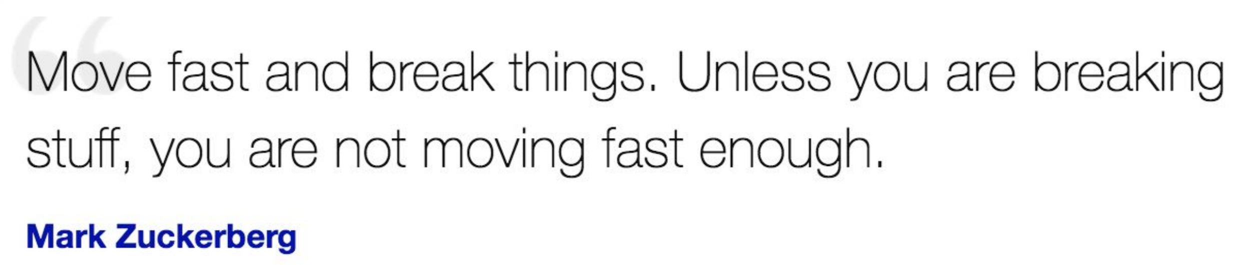
Preventing future hiccups

Release management : Progressive rollouts

- Canary servers
  - A few production instances serving traffic from new binary
- Canary for sufficient time
  - Give the new binary some time to experience all possible use cases
- Canary becomes prod on green
- Changes be backward compatible with 2-3 releases
  - Binaries can be safely rolled back

No-blame postmortem

- Teams write detailed postmortem after resolving outages
- Not attributed to any single person / team
- Postmortem is structured as:
  - What went wrong
  - What was the immediate fix
  - What is the long term fix
  - What can we do to prevent it occurring in the future ?



"Move fast and break things. Unless you are breaking stuff, you are not moving fast enough.

Mark Zuckerberg

Catastrophe Testing (Artificial)

Dedicated days to stress test the system

- Bring down a few machines
- Lose network connectivity

In summary

- Judiciously decide on what metrics to measure, and target values
- Monitor both user facing as well as internal metrics
- Alert only when necessary
- Plan for failure management - dashboards, playbooks and uniform logging
- Avoid a culture of heros
- Learn from failures

Questions ?