ICET LAB

CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

DAT490 Introduction

Dr. Philipp Leitner

philipp.leitner@chalmers.se

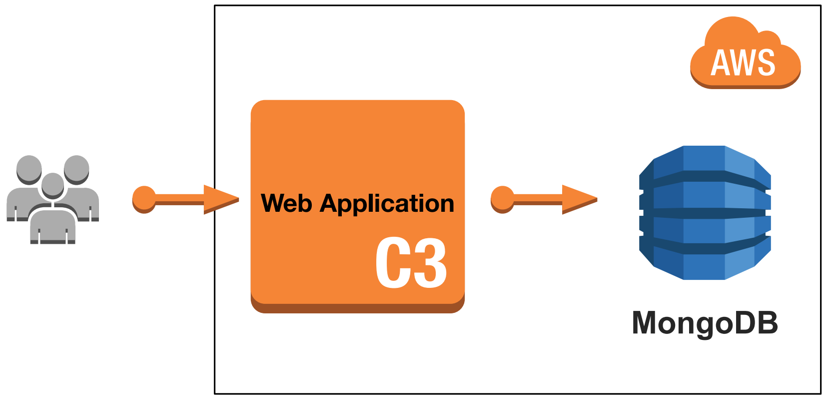
@xLeitix

LECTURE 1

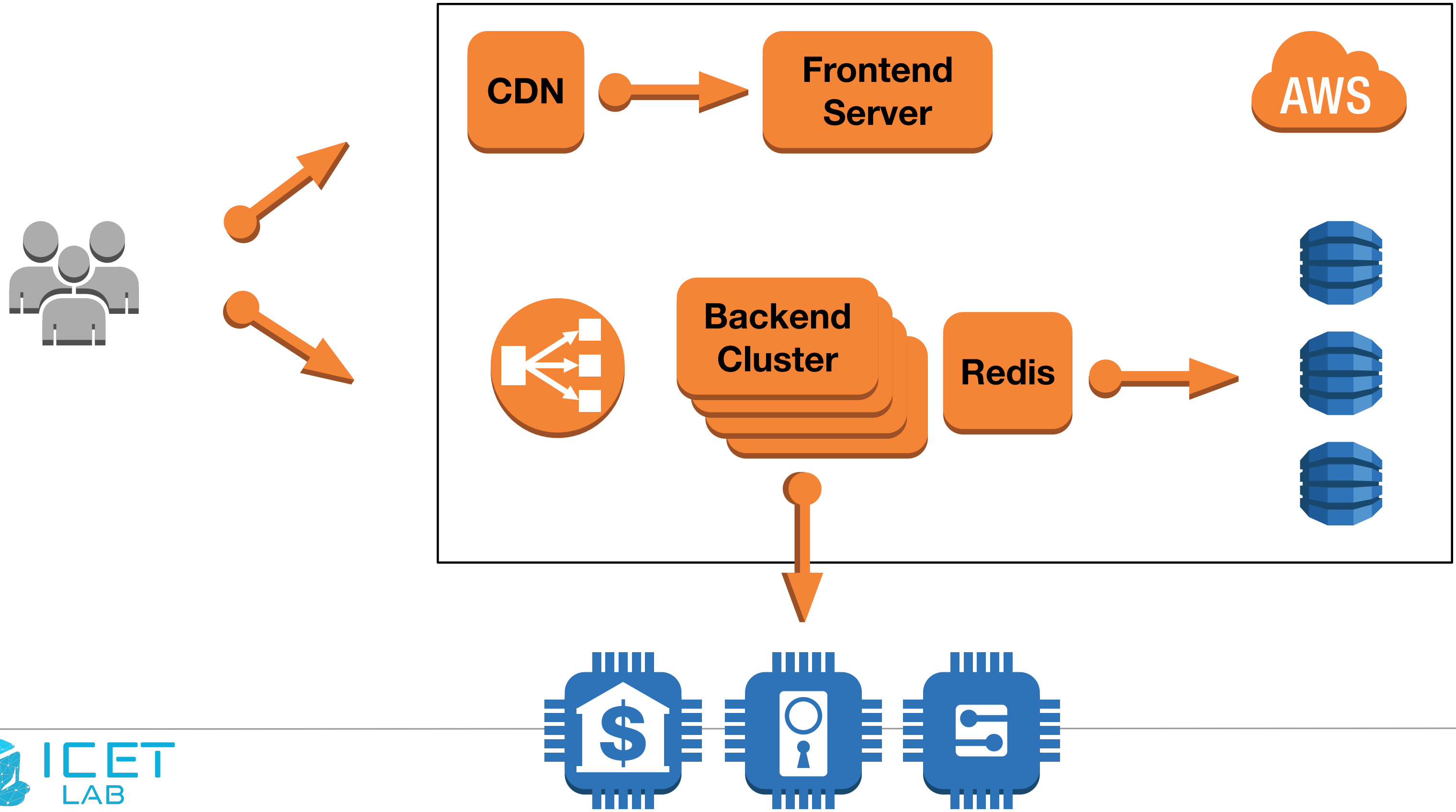
Covers …

A gentle introduction to Architecting for Scale
Scalability and Availability

Architectural Evolution of a Web App
Stage 1 - a three-tier web application



Architectural Evolution of a Web App
Stage 2 - architecting for (reasonable) scale

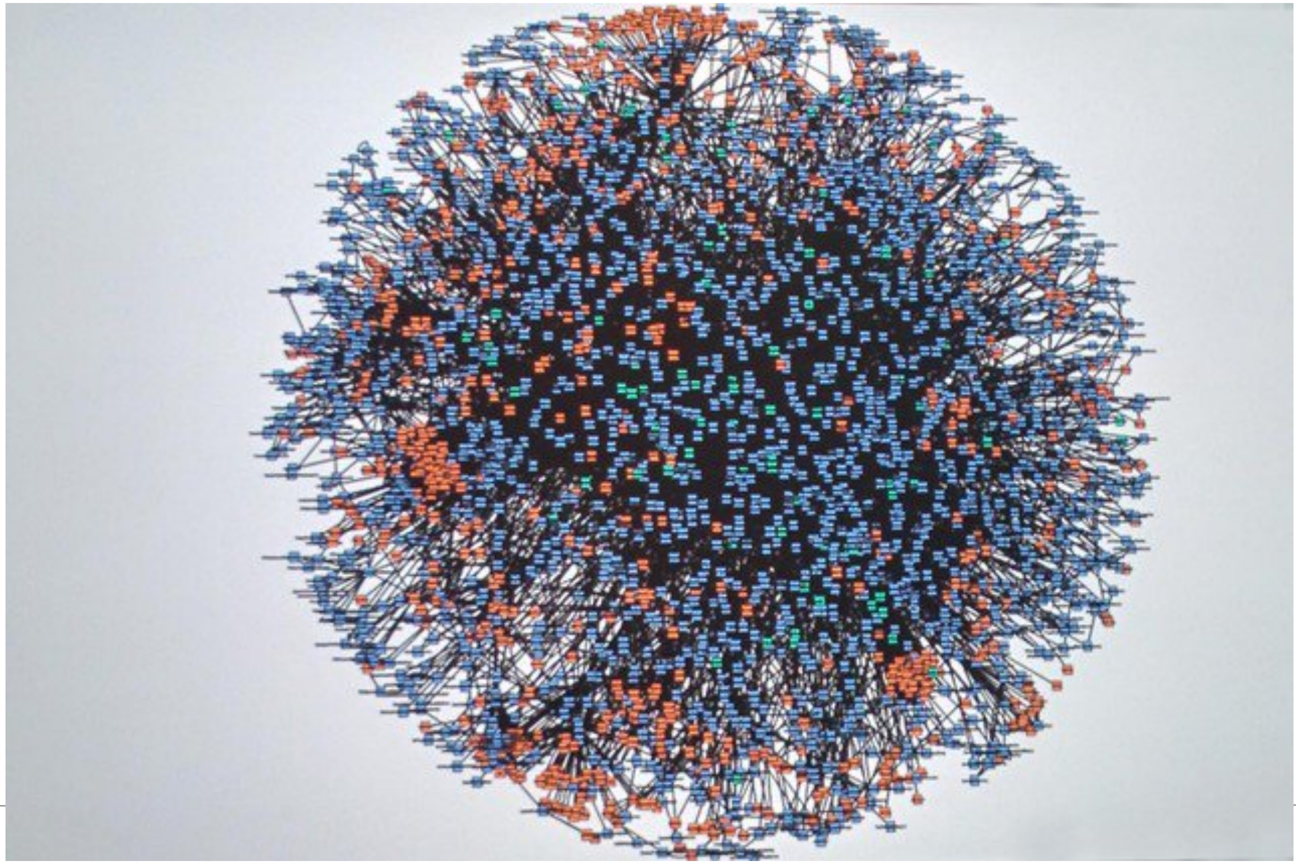


CHALMERS
UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG

....

Architectural Evolution of a Web App

Stage n - microservices



Source: amazon.com

What drives this evolution?

What drives this evolution?

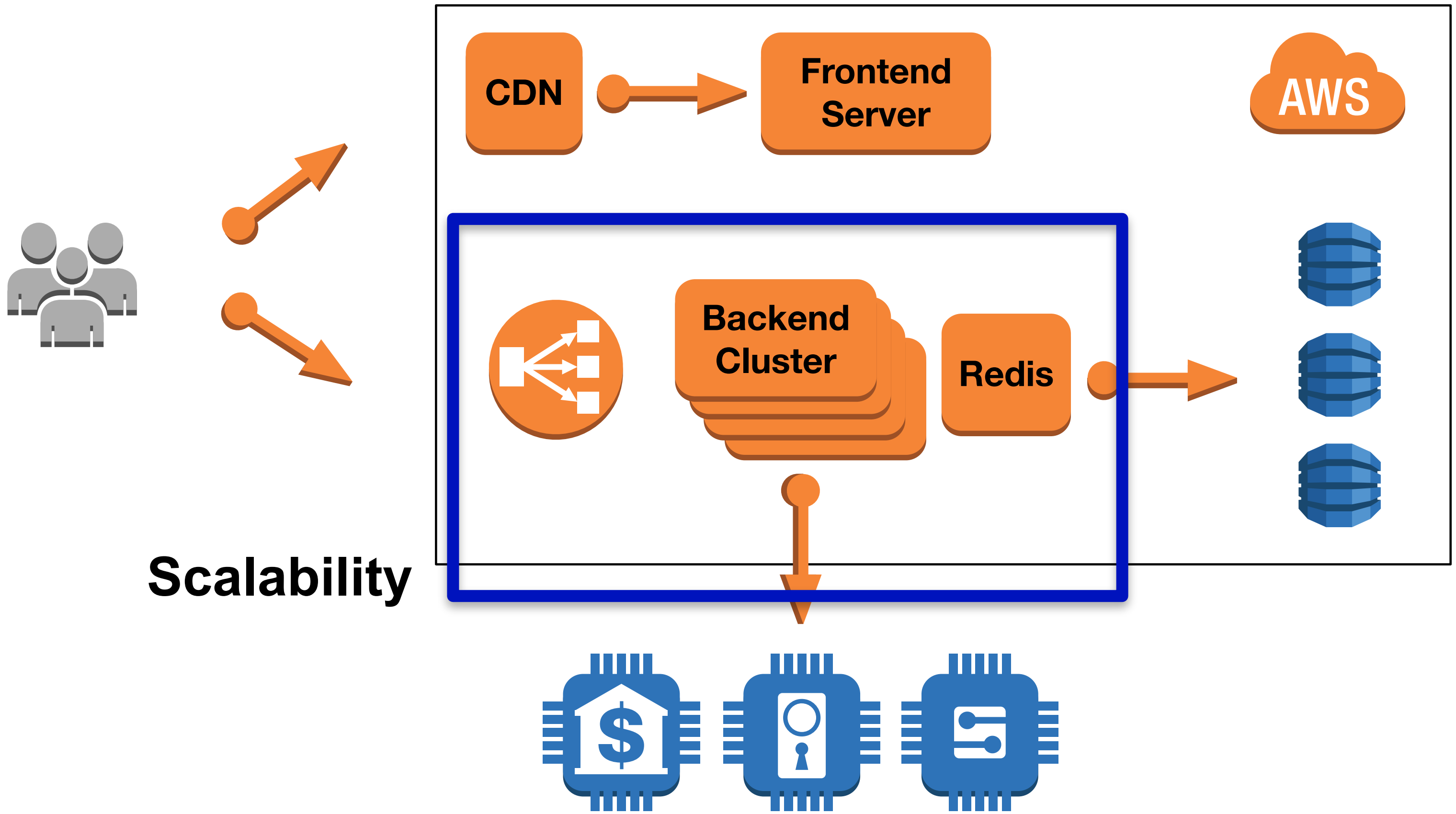
Support for new features (*of course*)

What drives this evolution?

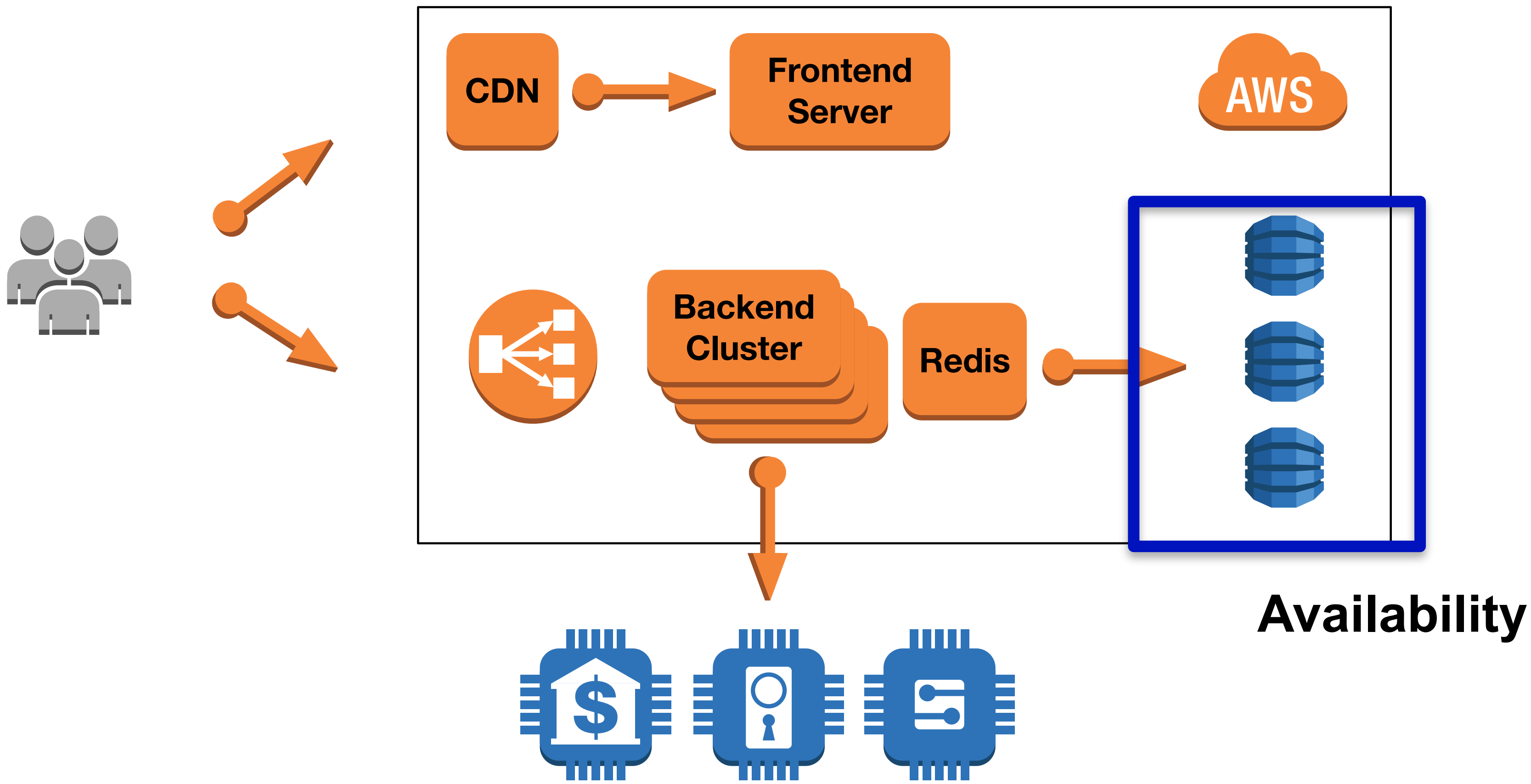
Support for new features (*of course*)

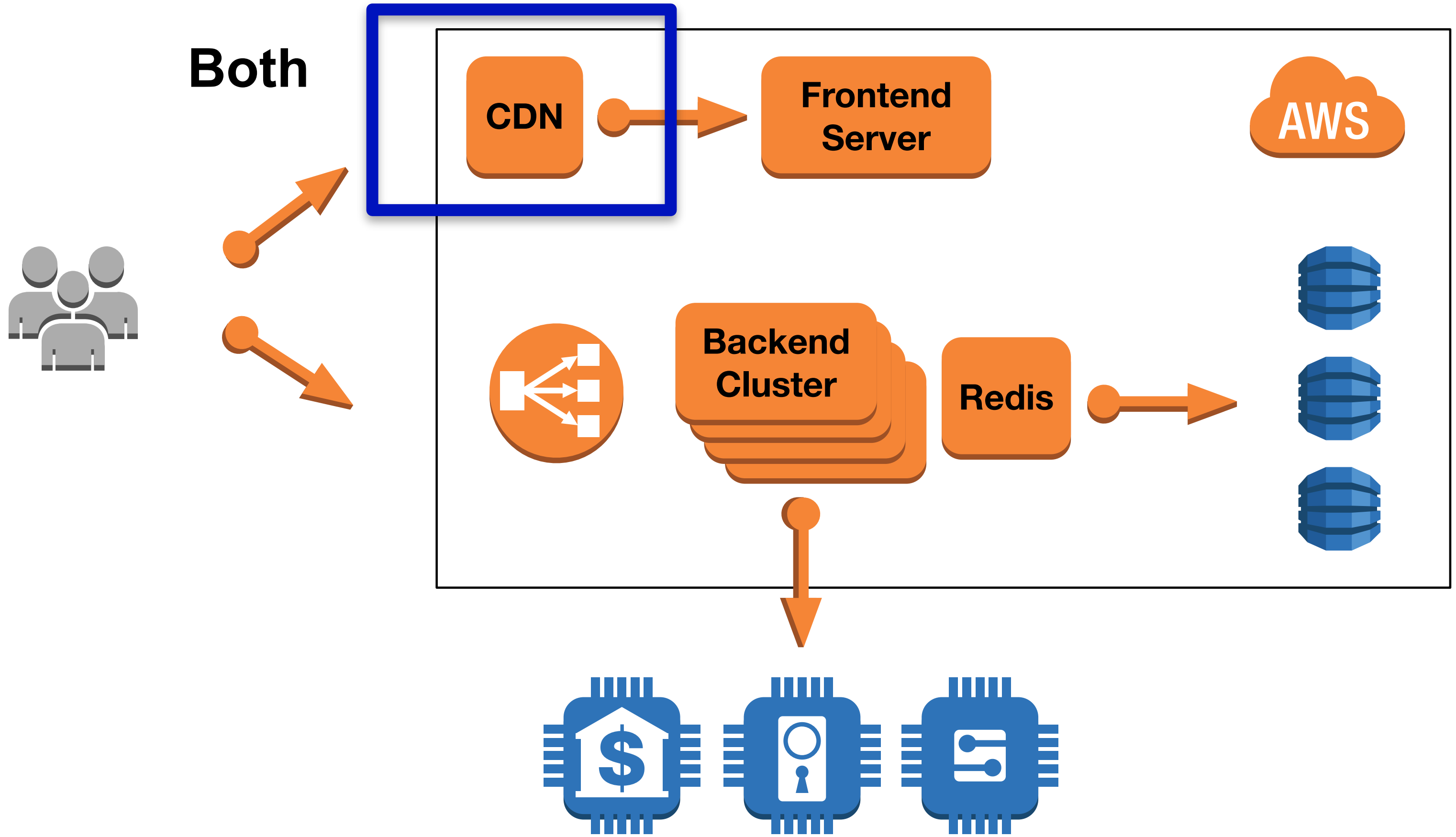
But also:

- **Scalability** (performance at scale, scaling development)
- **Availability** (resilience, survivability)



Scalability


CDN
Frontend Server
AWS
Backend Cluster
Redis
Availability


Both
CDN
Frontend Server
AWS
Backend Cluster
Redis

Architecting for Scale

"Transforming a (simple) application into a ***system*** *that can be used by a large number of concurrent users (****scalability****), that can be maintained and extended by a large team (****scalability of development****), that requires low or no downtimes (****availability****), and which survives intermittent failures (****resilience****)."*

Scalability (in terms of "number of concurrent users")

Scalability is the property of a system to handle a growing amount of work by adding resources to the system.

Metric for scalability:

Speedup

how much does my system become faster when adding 1 new unit of computing resources?

$$Speedup = \frac{N_{orig}\, t_{new}}{t_{orig}\, N_{new}}$$

$$e.g.$$

$$Speedup = \frac{1 * 80}{100 * 2} = 0.4$$

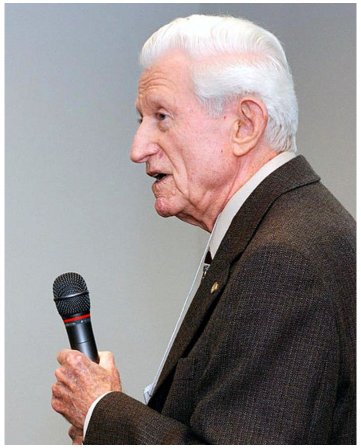
Speedup

Speedup is in [0;1]

Special cases:

- Speedup = 0
  - no gain, e.g., adding more CPUs to a single-threaded program
- Speedup = 1
  - Twice the resources, half the time (perfect speedup)
  - “Embarrassingly parallel” applications

Amdahl's Law

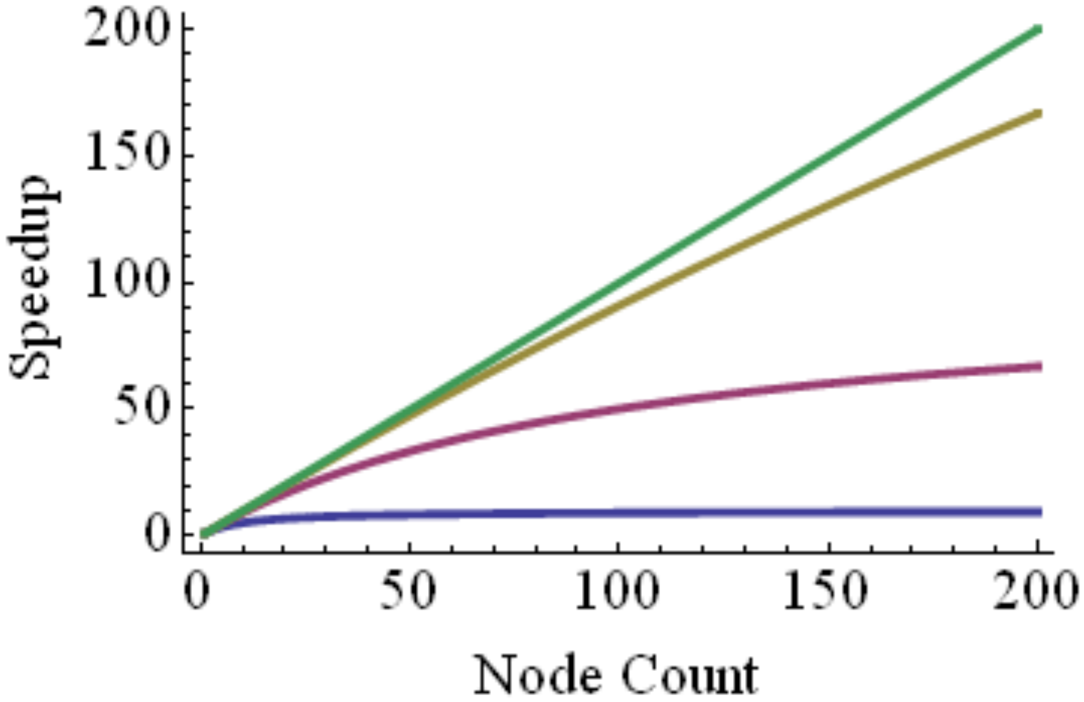


$$t_{total} = t_{serial} + \frac{t_{parallel}}{N}$$

$$Speedup = \frac{1}{(1 + (N-1)a)},\ with\ a = \frac{t_{serial}}{(t_{serial} + t_{parallel})}$$

For *a* > 0, speedup converges against 0 (quickly)

Amdahl’s Law

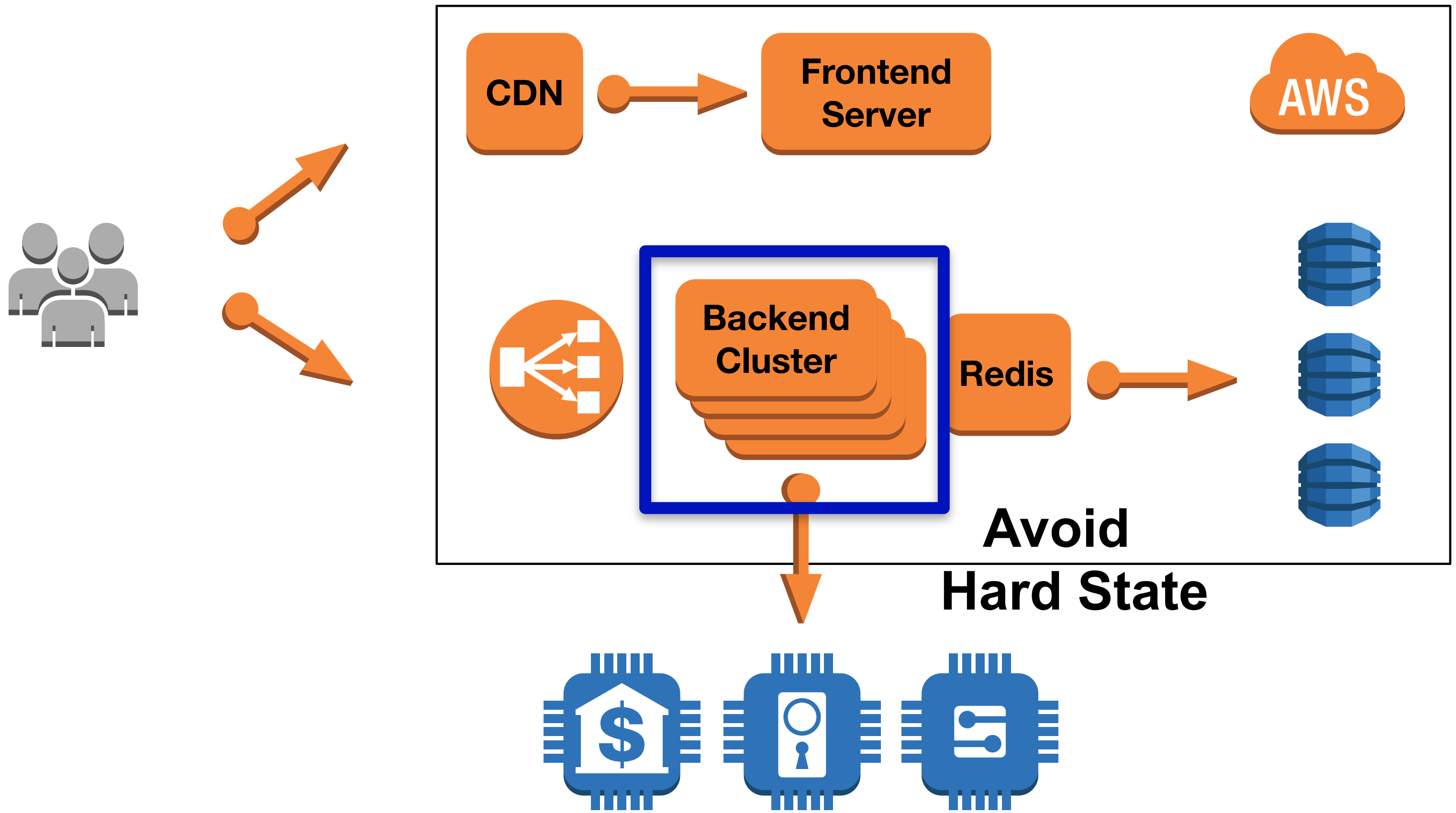


Building Embarrassingly Parallel Web Apps

Implication:

- **State (another word for "serial processing required") is the archenemy of architecting for scale**

- **Most architectural guidelines for building scalable Web applications have some "statelessness" requirement:**
  - **e.g., the REST architectural style, Heroku's 12-Factor App, etc.**


CDN
Frontend Server
AWS
Backend Cluster
Redis
Avoid Hard State

Performance

Page Load Time
Time from request sent to page completely loaded

Response Time
Time from request sent to response received

Network Latency
Time request / response takes for transmission through the network

Throughput
Number of requests that can be processed per time unit

Error Rate
How frequently to requests transiently fail?

For all these:
Average / 95-percentile / 99-percentile

Performance vs. Scalability in Practice

In Web development practice, the relationship between performance and scalability is … complicated.

- **… looking at Amdahl's law, we would expect that twice as many backend instances would lead to half the response time**
  - ***Spoiler: this is never how it actually works***

Availability

Availability is the probability of a system to be online (“available”) at any point in time

$$av(S) = 1 - \frac{downtime}{uptime}$$

Availability

Common measure for availability: x-nines,

"2-nines" (0.99) … about 8 hours offline per month
"3-nines" (0.999) … about 44 minutes offline per month
"5-nines" (0.99999) … about half a minute offline per month

Service Level Agreements (SLAs)

Service providers often contractually guarantee a certain availability (and sometimes response time)

E.g., Google:

> “
>
> Bigtable instances with a multi-cluster routing policy across three or more regions are now covered by a 99.999% monthly uptime percentage under the new SLA. Bigtable supports 99.99% monthly uptime percentage for all instances with a multi-cluster routing policy across less than three regions and 99.9% monthly uptime percentage for all instances with a Single-Cluster routing policy.

Service Level Agreements (SLAs)

Important questions for SLAs:

How is the metric specified (exactly), and who measures it?

Especially, what's the aggregation time window?

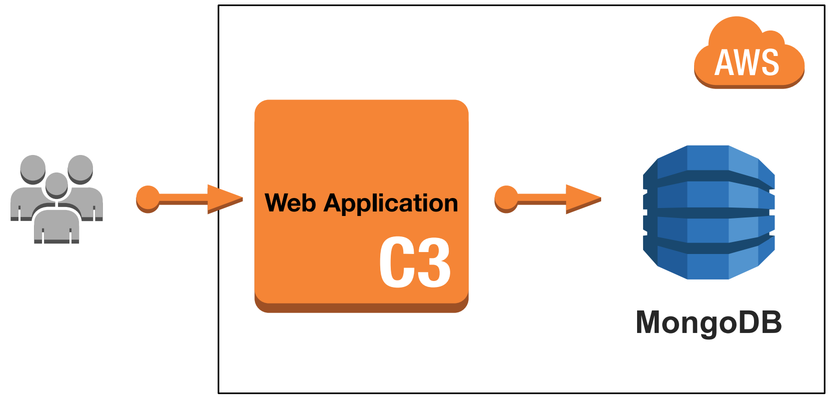
What's the compensation for SLA violations?

Do scheduled maintenance windows count?

Etc.

Redundancy

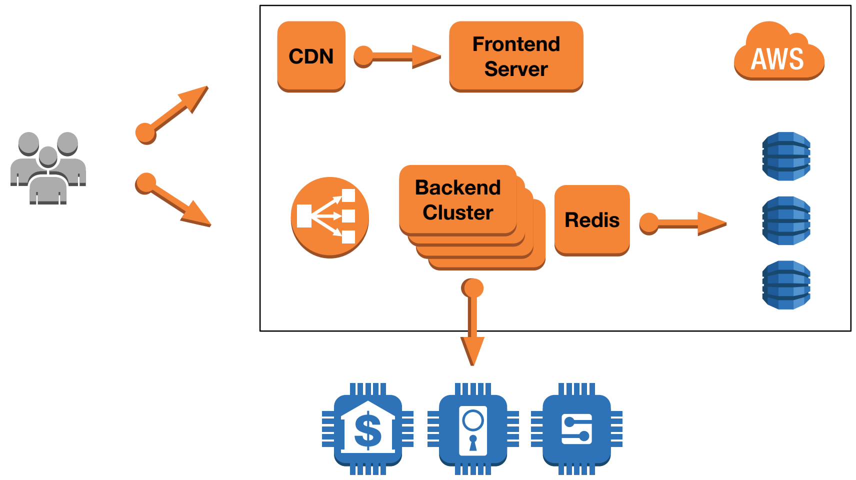
One (common) way to improve availability is redundancy
(Avoiding single points of failure)



Availability of our simple architecture is:

av = av(app) * av(db)

av = 0.99 * 0.99 = 0.98



Availability of our "scaled" architecture:

av = av(cdn) * av(any_backend) * av(any_db)

av ~ 1 (theoretically)

Co-Located Failures

In practice our availability will still be < 1

Mostly due to "co-located failures" (co-failures)

Availabilities of our services are in practice not independent

One service failing drastically increases the failure probability of other services

E.g., if all our services are in one AWS data center and this data center burns down, it matters little *how many* replicas we had

In practice availability of > 0.99999 is not consistently achievable

For the third time in less than a month **Amazon Web Services (AWS)** is down, not only affecting its own store, but third party services that host with them like Epic Games Store, Tinder, and Hulu among others.

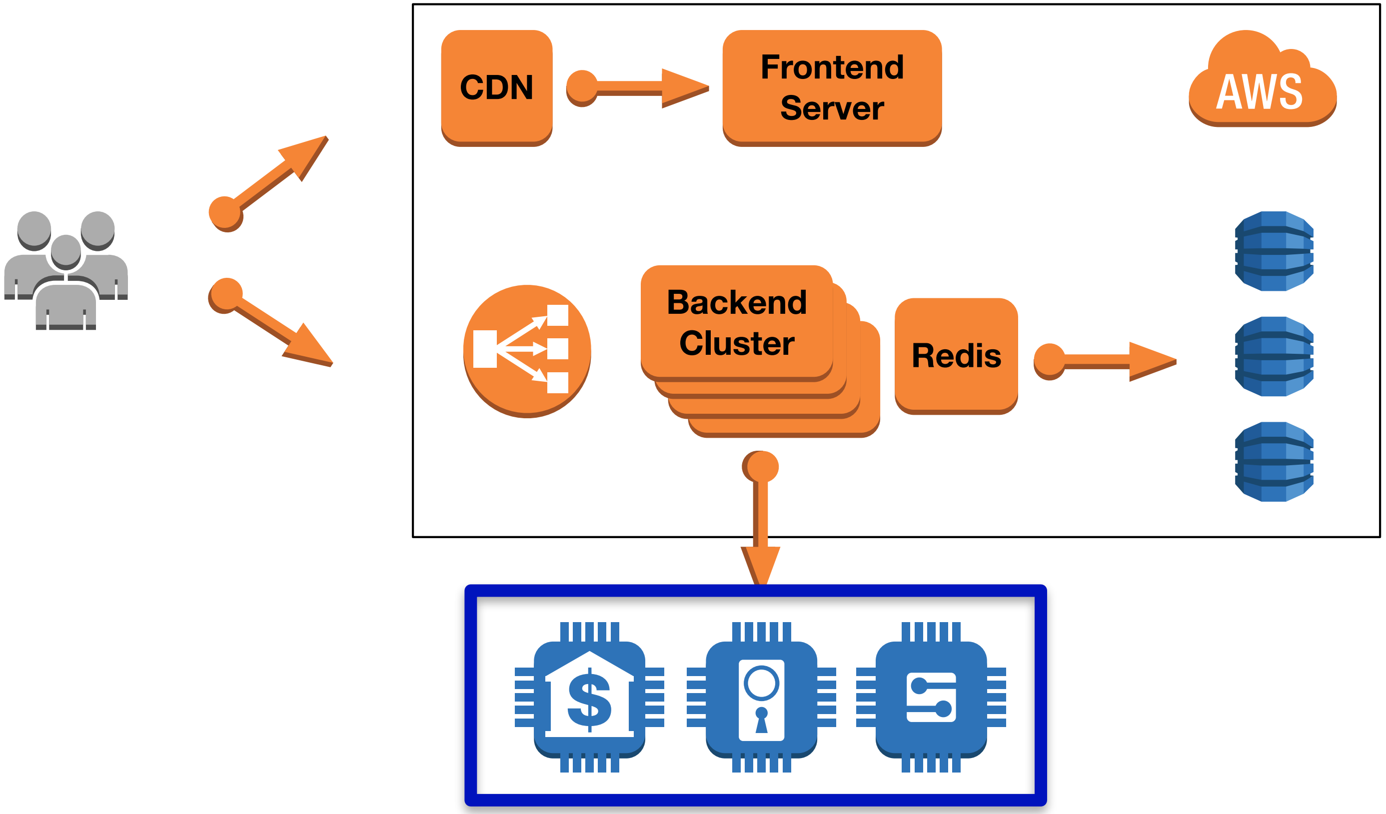
https://www.marca.com/en/lifestyle/us-news/2021/12/22/61c36eb5ca4741ee0f8b45ec.html

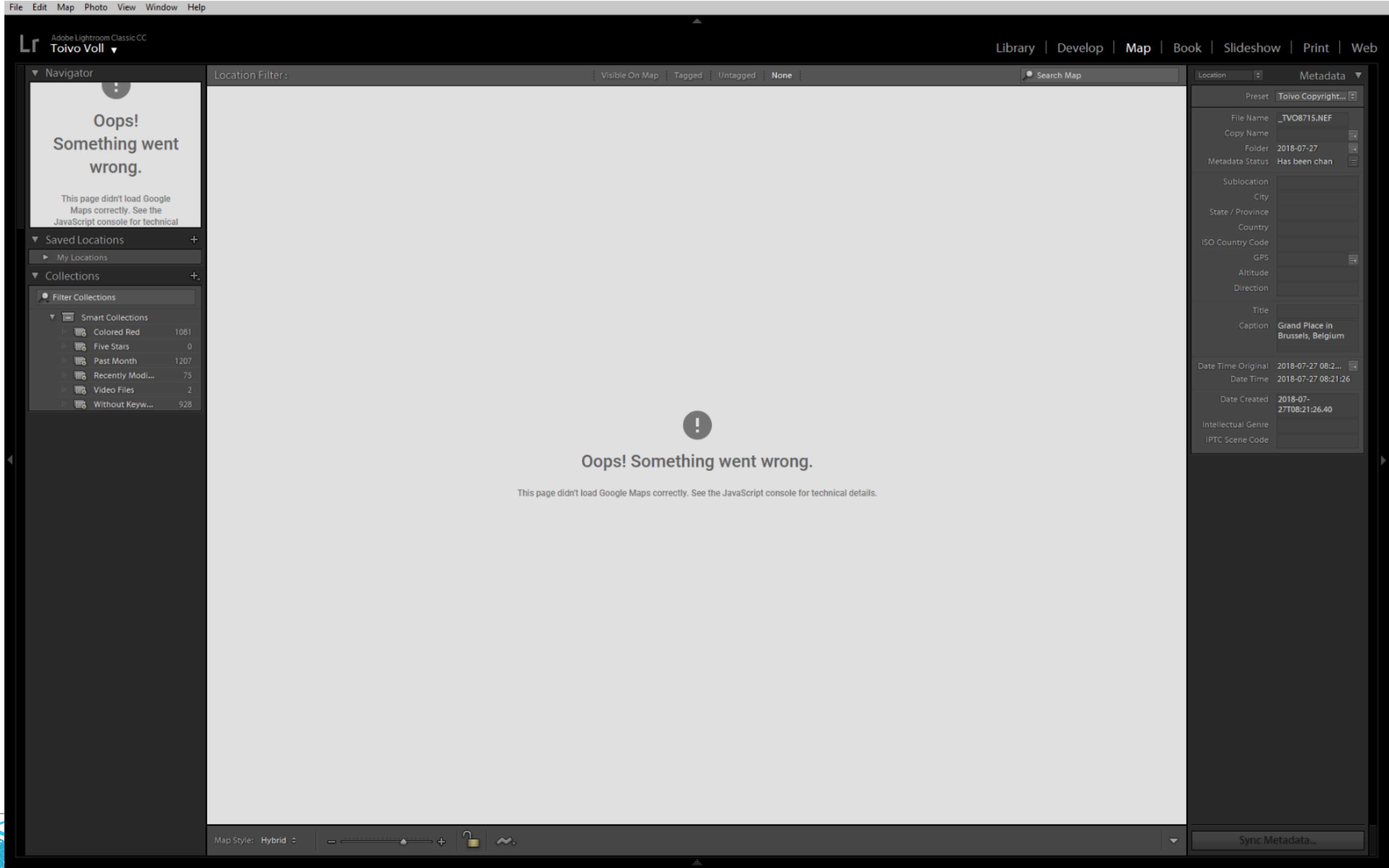
Resilience

Another way to improve our availability is resilience (or survivability)

Essentially:

- **Make your application react gracefully to optional components not being available**


CDN
Frontend Server
AWS
Backend Cluster
Redis


File Edit Map Photo View Window Help
Adobe Lightroom Classic CC
Toivo Voll
Library | Develop | Map | Book | Slideshow | Print | Web
Navigator
Oops! Something went wrong.
This page didn't load Google Maps correctly. See the JavaScript console for technical
Saved Locations
My Locations
Collections
Filter Collections
Smart Collections
Colored Red 1081
Five Stars 0
Past Month 1207
Recently Modi... 75
Video Files 2
Without Keyw... 928
Location Filter :
Visible On Map
Tagged
Untagged
None
Search Map
Oops! Something went wrong.
This page didn't load Google Maps correctly. See the JavaScript console for technical details.
Map Style: Hybrid
Metadata
Preset Toivo Copyright...
File Name _TVO8715.NEF
Copy Name
Folder 2018-07-27
Metadata Status Has been chan
Sublocation
City
State / Province
Country
ISO Country Code
GPS
Altitude
Direction
Title
Caption Grand Place in Brussels, Belgium
Date Time Original 2018-07-27 08:2...
Date Time 2018-07-27 08:21:26
Date Created 2018-07-27T08:21:26.40
Intellectual Genre
IPTC Scene Code
Sync Metadata...

Summary

Scalability and availability as driving the migration to scaled architectures