the road to cloud native applications

Fabien Hermenier

Université Nice Sophia Antipolis

cloud ready applications

single-tiered
monolithic
hardware specific

cloud native applications

leverage cloud services
scalable
reliable

Agenda

Phase 1 | **Phase 2** | **Phase 3**

Follow the lecture

Profit

Develop Cloud-Native Applications

Cloud Architecture Patterns

O'REILLY®

Bill Wilder

Architecting for the Cloud: Best Practices

January 2011

Jinesh Varia
jvaria@amazon.com

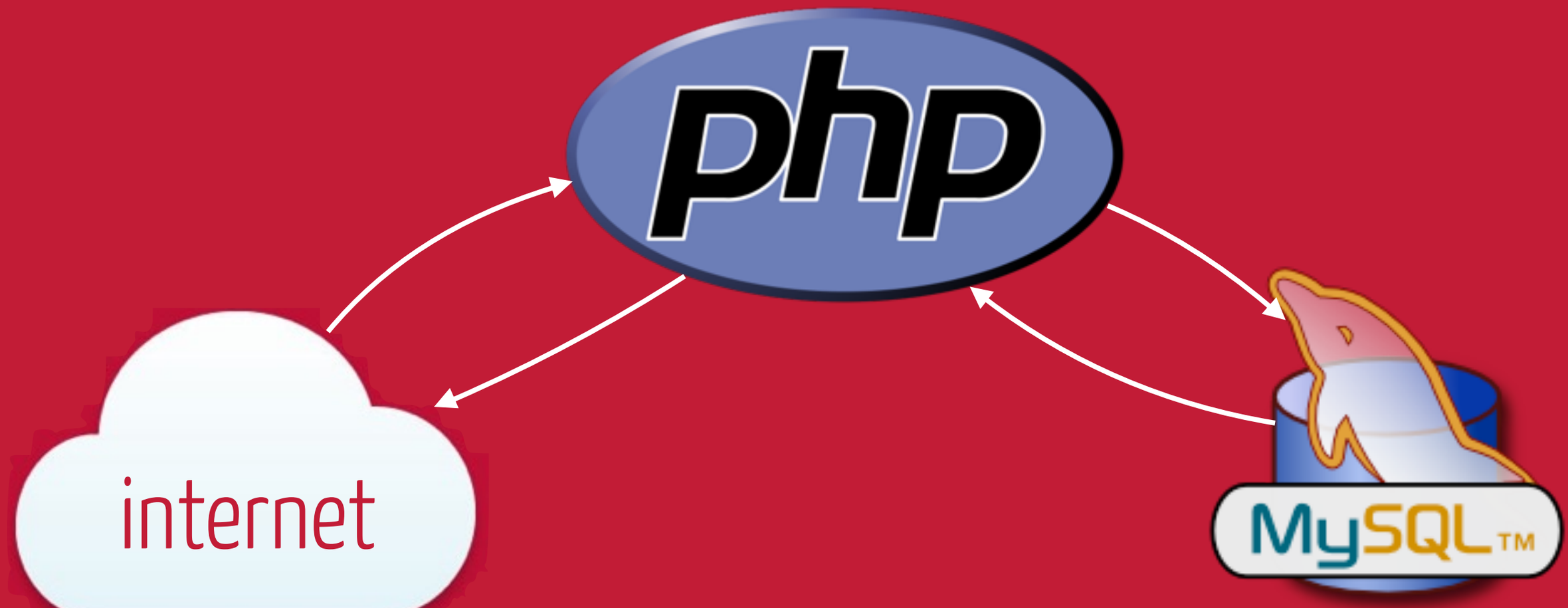
live and prosper wo. downtime

upgrade

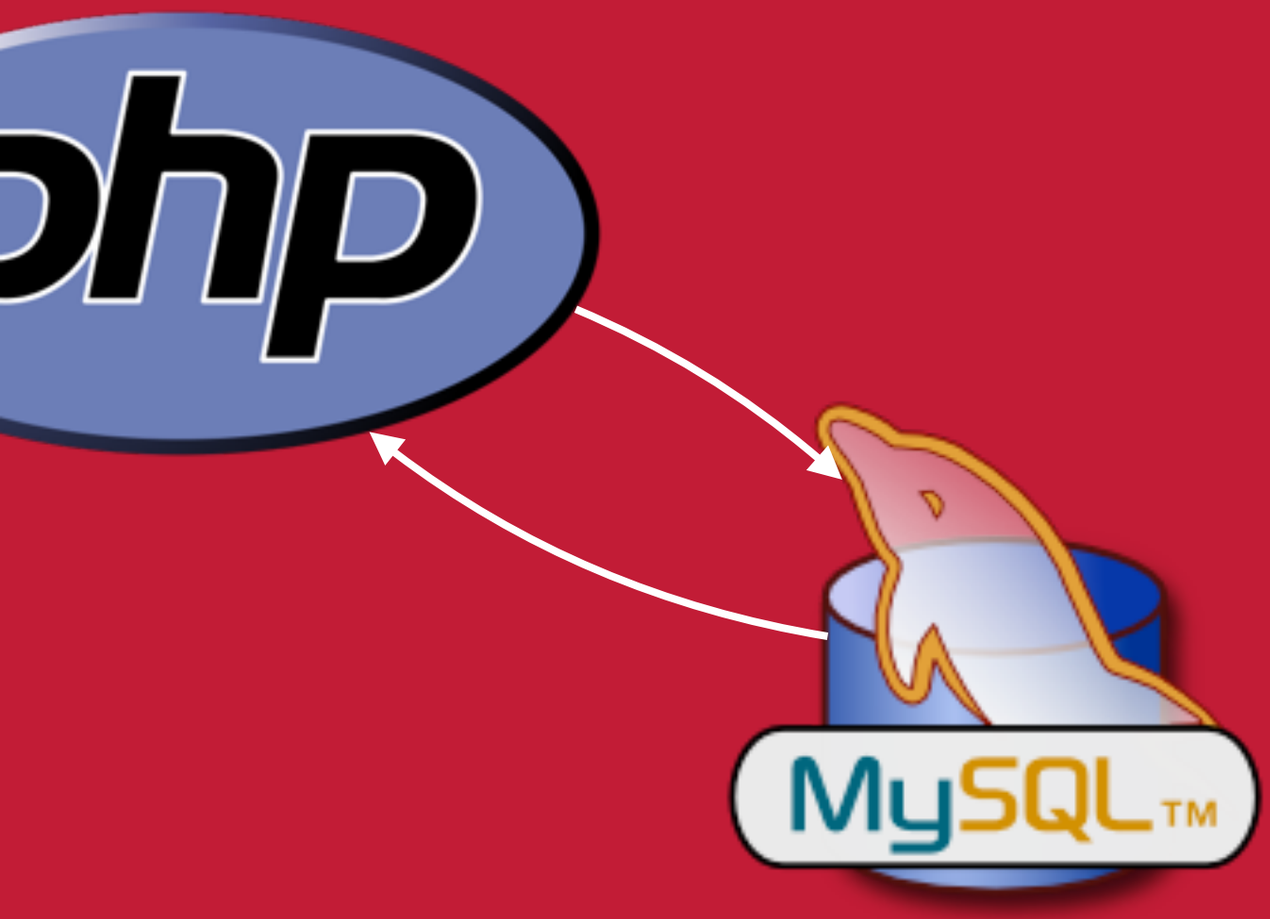
non-blocking, async communications

scale

prehistorical times

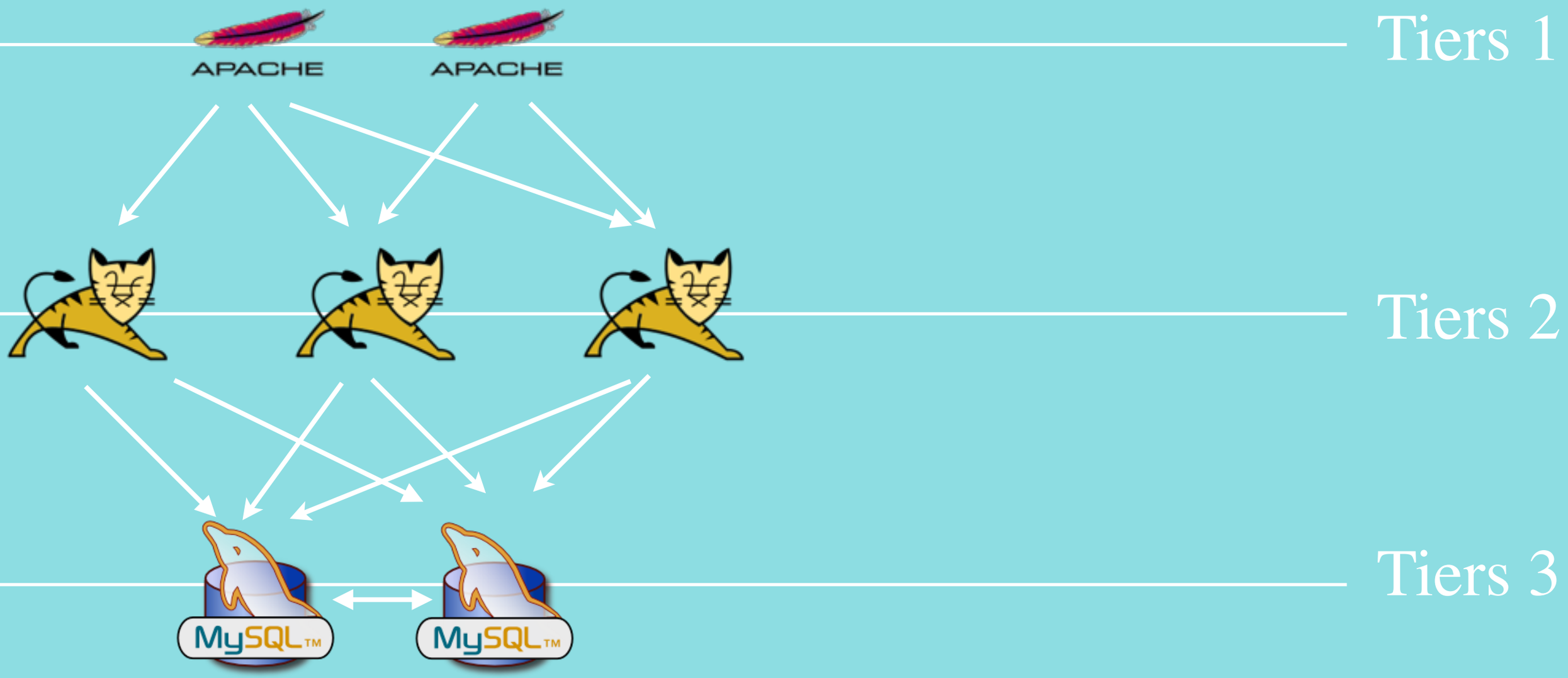


prehistorical times



2 **S**ingle **P**oint **o**f **F**ailure
limited scalability
not scalable online
not upgradable online

n-tiered apps to the rescue

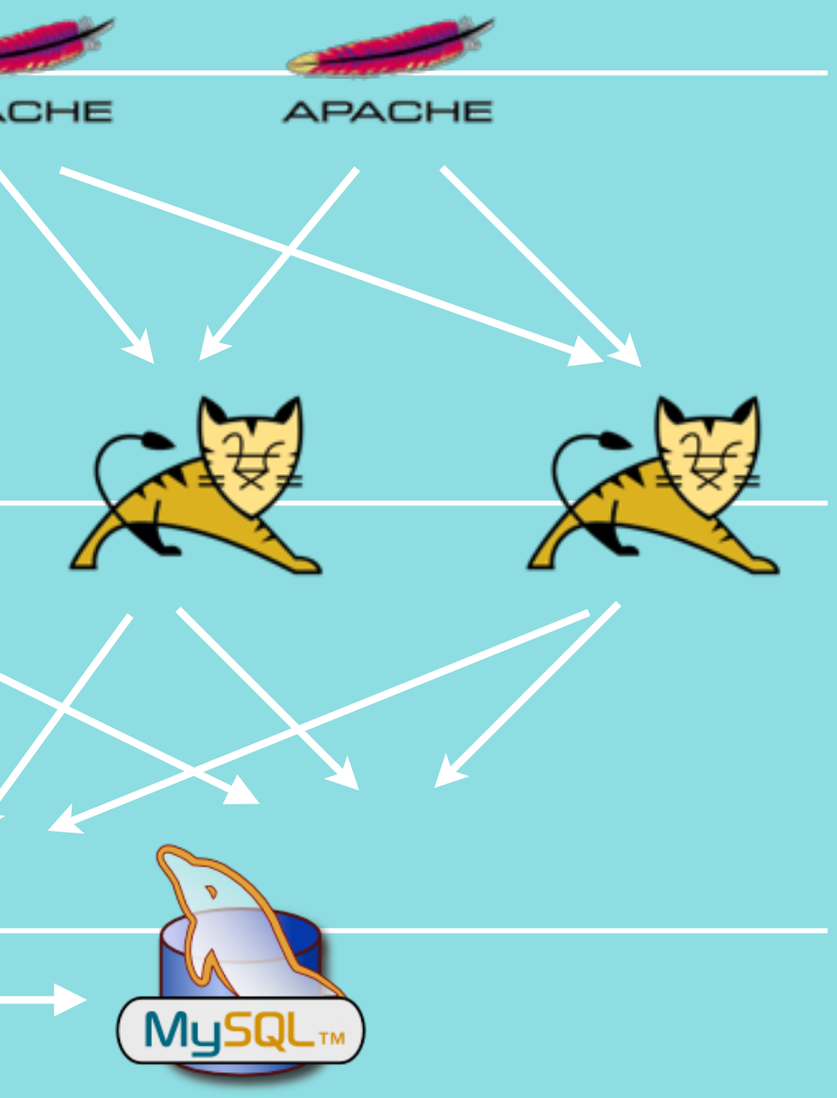


Tiers 1

Tiers 2

Tiers 3

n-tiered apps to the rescue



Tiers 1

Tiers 2

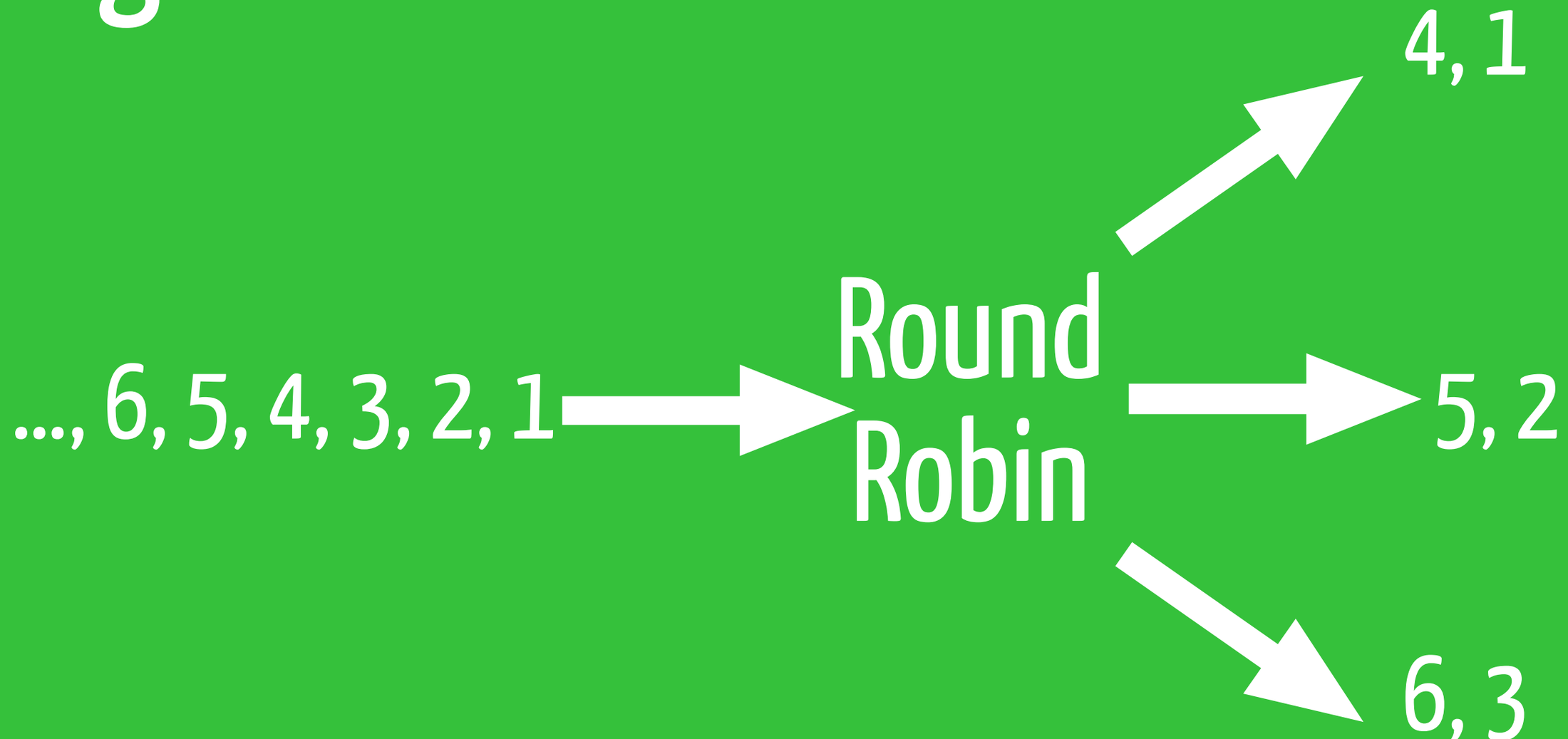
Tiers 3

load balancing
horizontal scalability
upgradable online

spread incoming requests
to backend services

synchronous communications

possible load balancing algorithms



nice on homogeneous nodes

possible load balancing algorithms

6, 5, 3, 1

..., 6, 5, 4, 3, 2, 1 → weighted →

4, 2

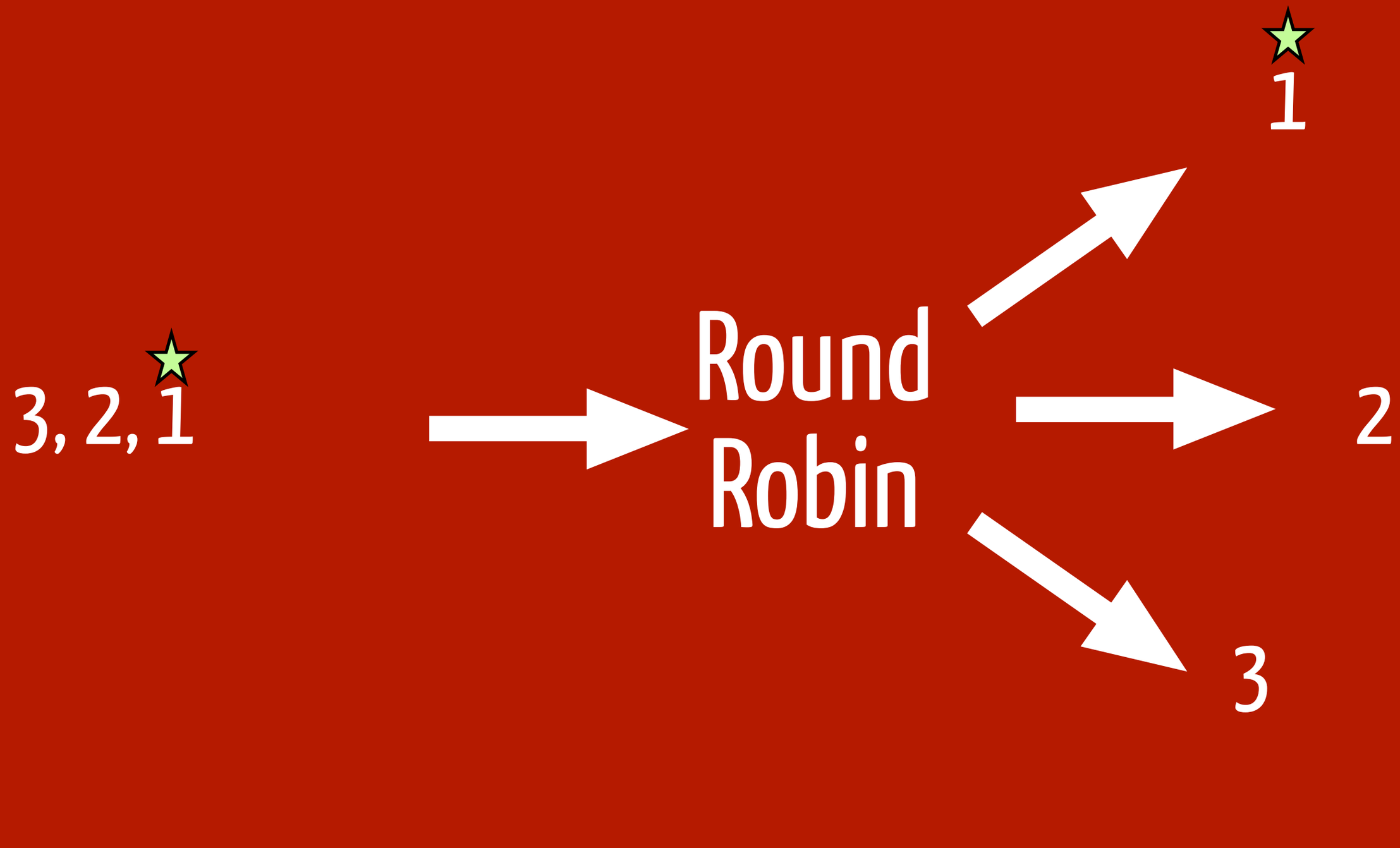
weight the nodes or the requests
depending on their size

“spread incoming requests
to backend services

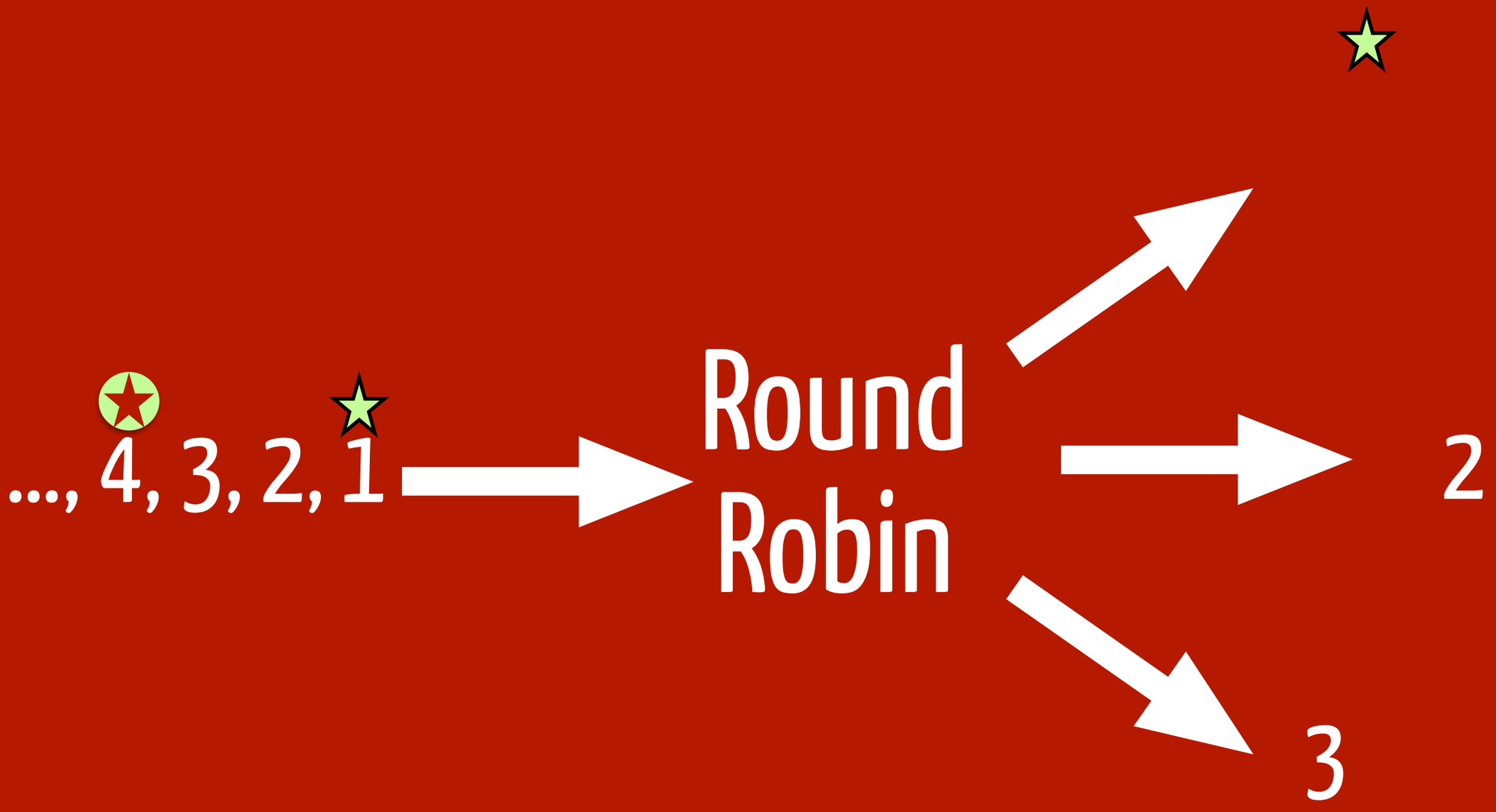
synchronous communications”

is everything balanceable ?

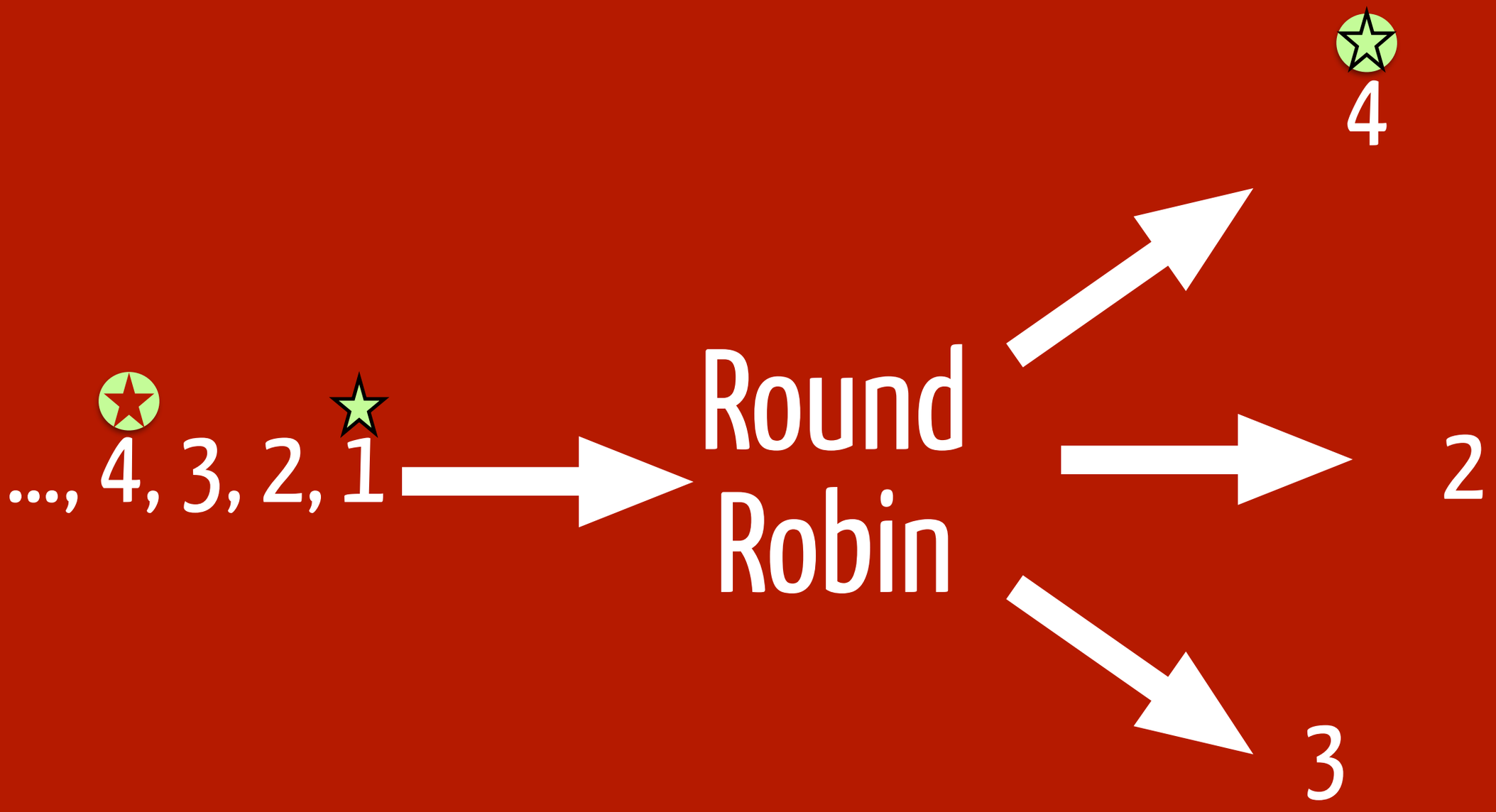
load balancing stateful stuff



load balancing stateful stuff



load balancing stateful stuff



load balancing stateful stuff

achievement unlocked

4

…, 4, 3, 2, 1

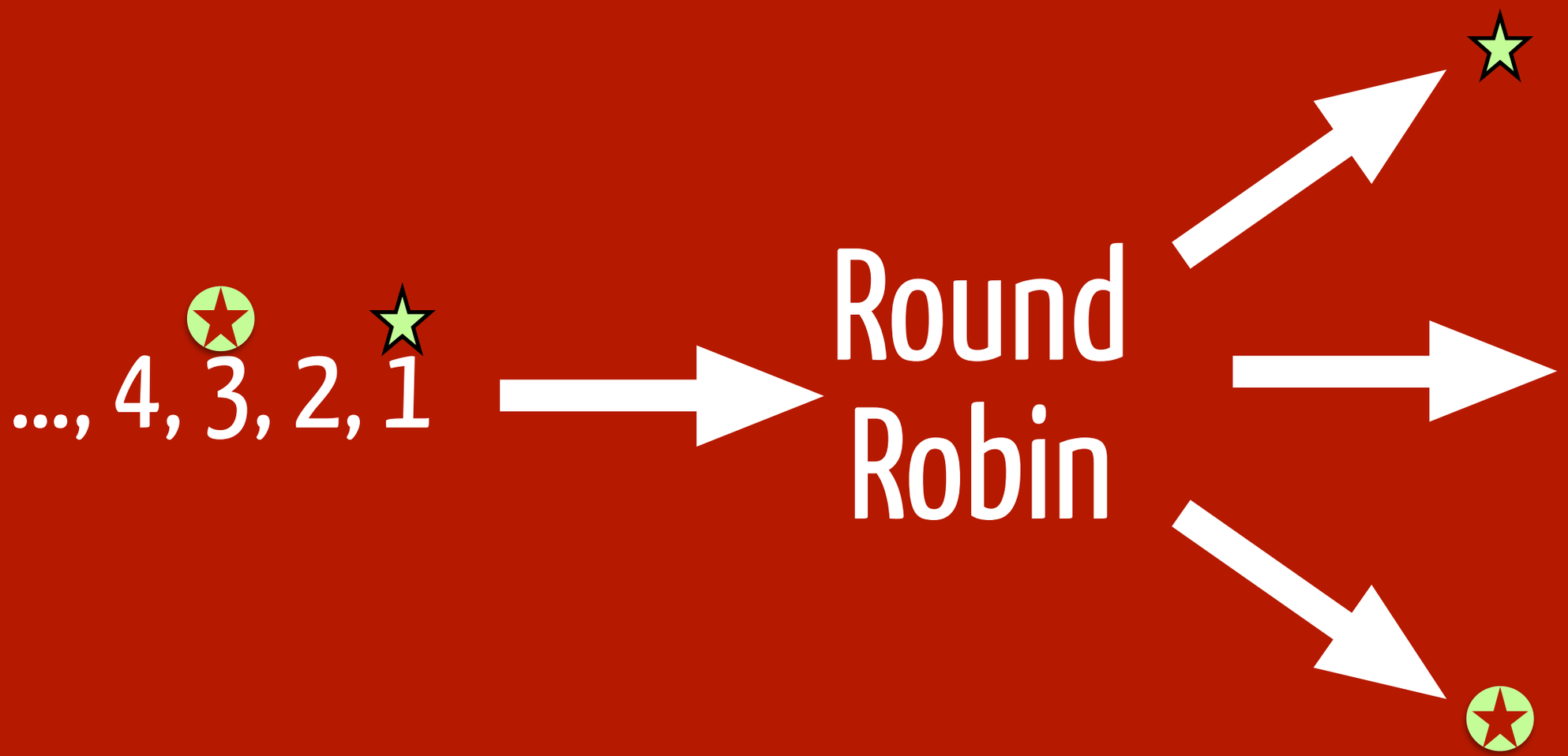
Round Robin

4

2

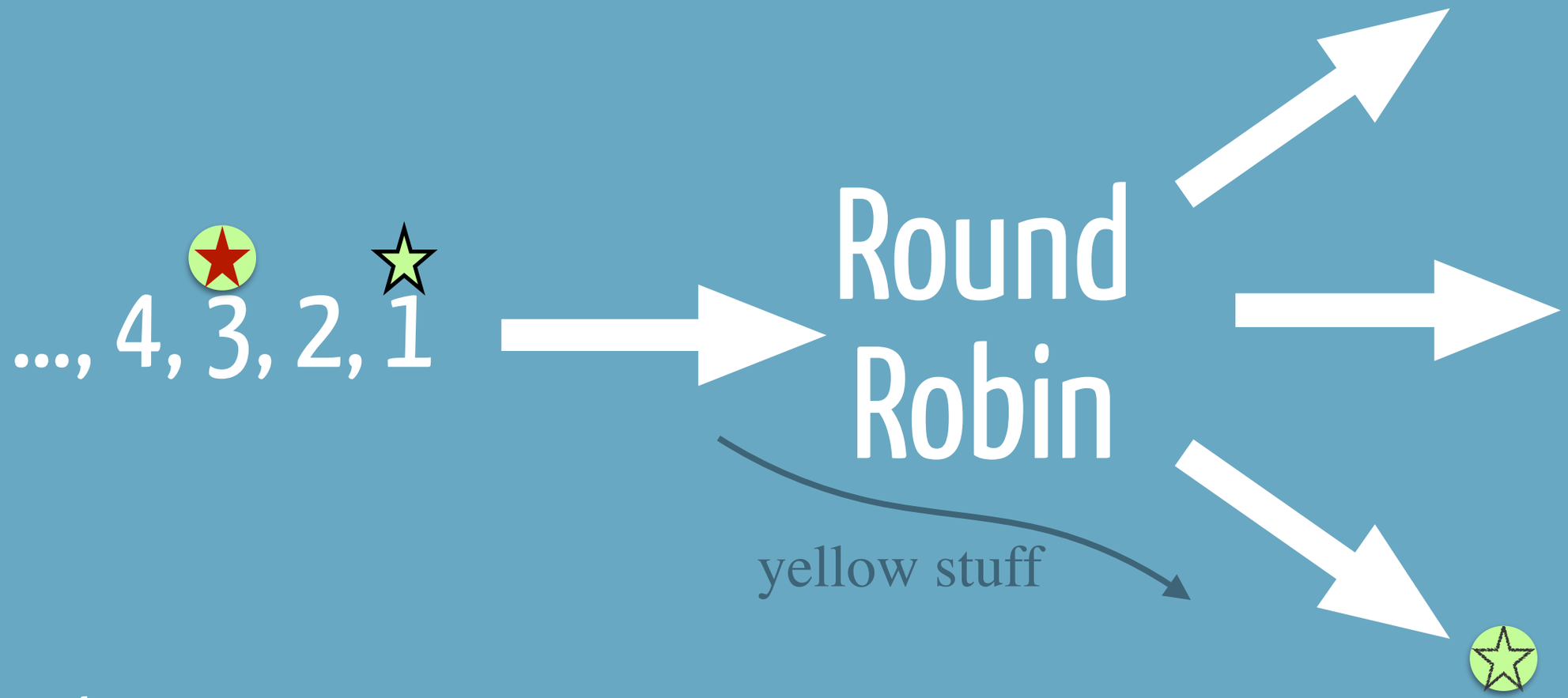
3

load balancing stateful stuff



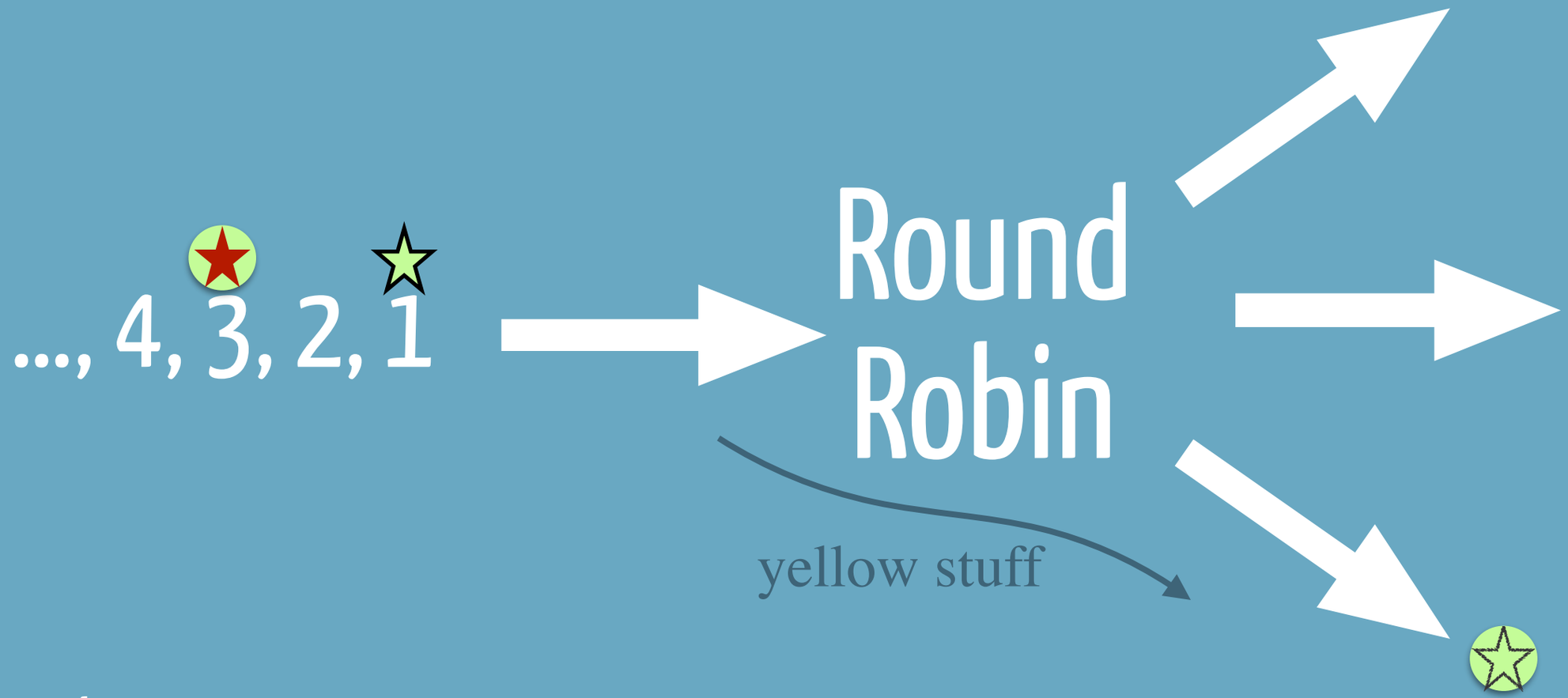
load balancing stateful stuff

load balancing stateful stuff + sticky sessions



IP based

load balancing stateful stuff + sticky sessions



IP based

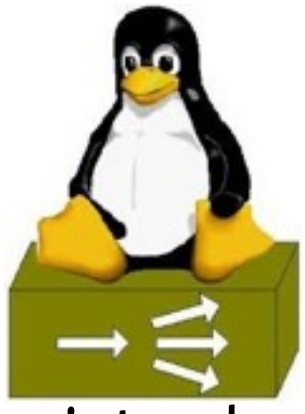
not resilient to node failures

amazon
elastic load balancing

NGINX

APACHE mod_proxy_balancer

tc lb



linux virtual server

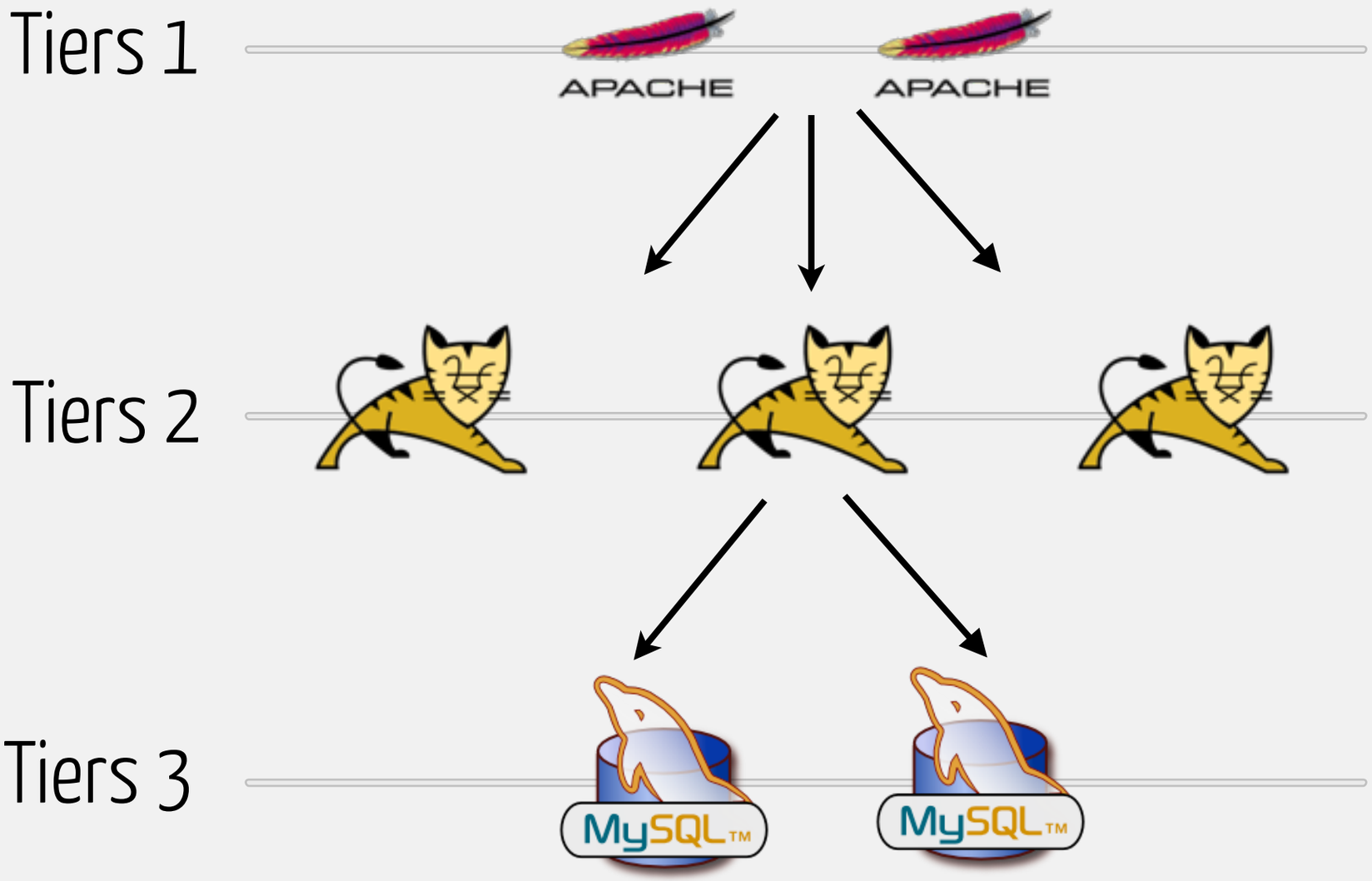
galera

MariaDB

scaling

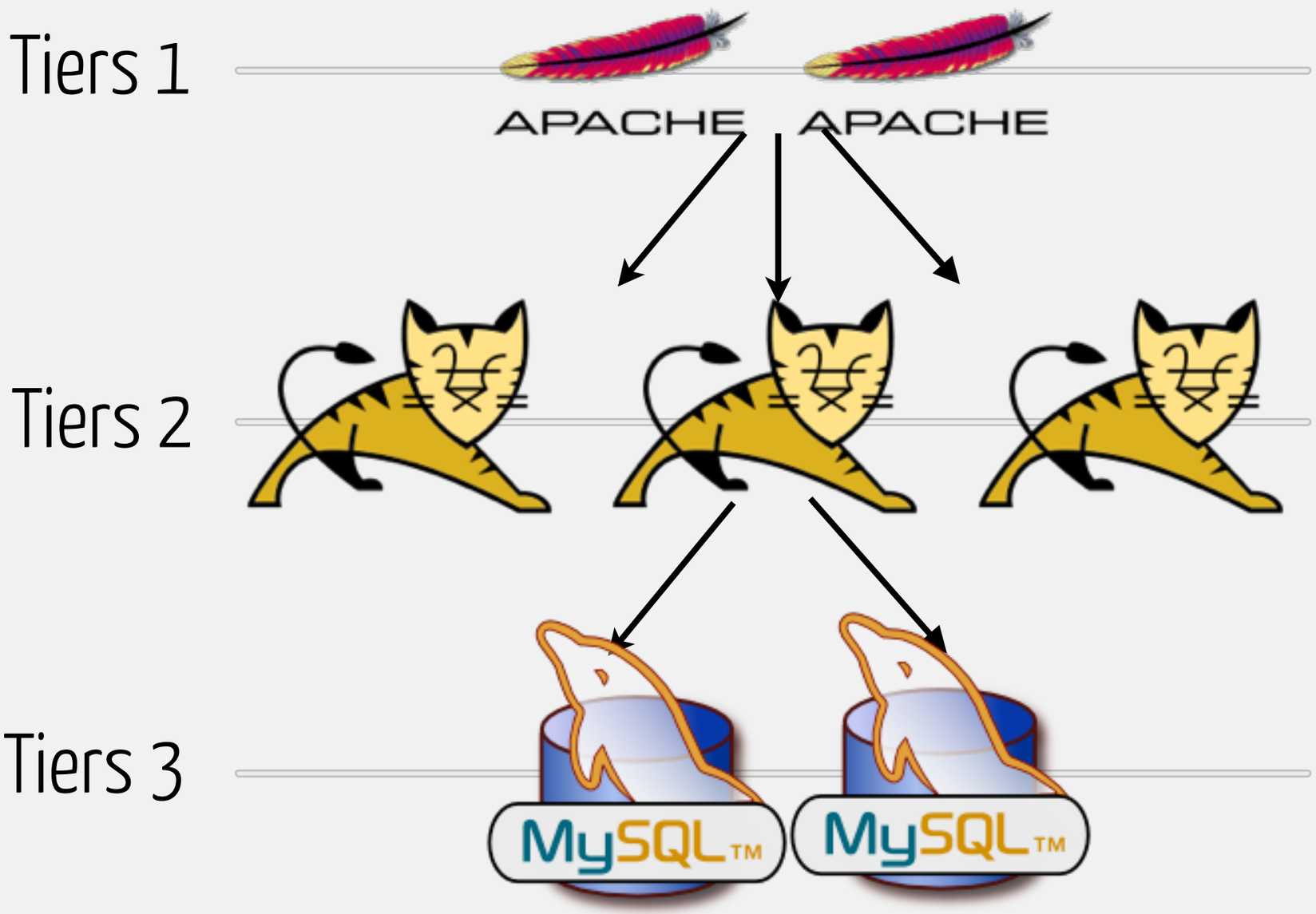
scale up / down

vertical sclaing



scale up / down

vertical sclaing



historical method (more powerful hardware)

mostly cold approaches

easy to implement coldly

hardware bounded

does not address reliability

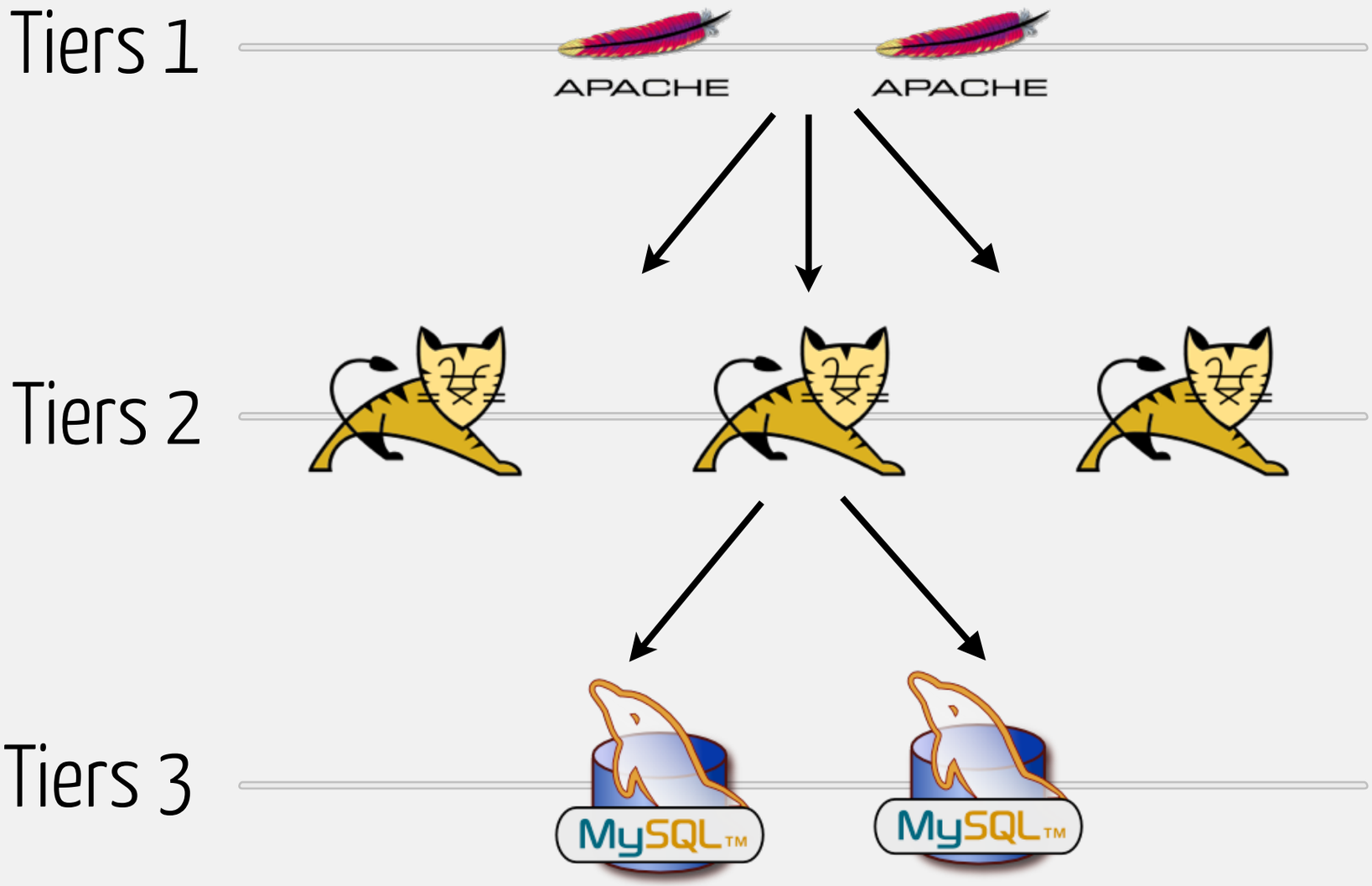
vertical scalin

oVirt support for hot plug CPU

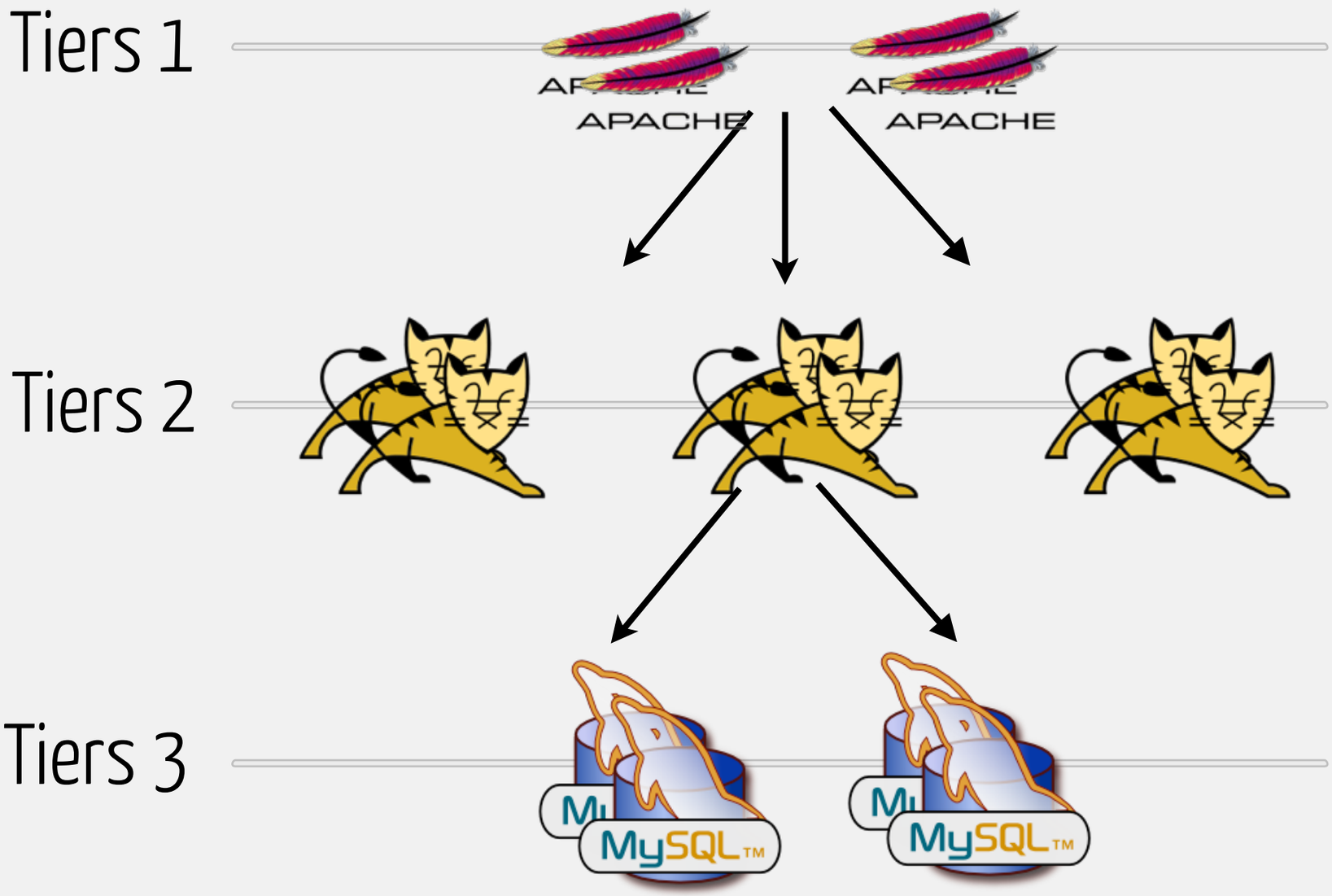
Guest OS Support Matrix

OS	Version	Arch	Plug	Unplug
Red Hat Enterprise Linux 6.3		x86	+	-
Red Hat Enterprise Linux 6.5		x86	+	+
Microsoft Windows Server 2003	All	x86	-	-
Microsoft Windows Server 2003	All	x64	-	-
Microsoft Windows Server 2008	All x86	-	-	
Microsoft Windows Server 2008	Standard, Enterprise	x64	Reboot Required	Reboot Required
Microsoft Windows Server 2008	Datacenter	x64	+	?
Microsoft Windows Server 2008 R2	All	x86	-	-
Microsoft Windows Server 2008 R2	Standard, Enterprise	x64	Reboot Required	Reboot Required
Microsoft Windows Server 2008 R2	Datacenter	x64	+	?
Microsoft Windows Server 2012	All	x64	+	?
Microsoft Windows Server 2012 R2	All	x64	+	?
Microsoft Windows 7	All	x86	-	-
Microsoft Windows 7	Starter, Home, Home Premium, Professional	x64	Reboot Required	Reboot Required
Microsoft Windows 7	Enterprise, Ultimate	x64	+	?
Microsoft Windows 8.x	All	x86	+	?
Microsoft Windows 8.x	All	x64	+	?

what about the runtime ?



horizontal scaling



horizontal scaling

horizontal scaling

not application agnostic — require a load balancer / synchronisation — scale to the infinite in theory — support node failure

Elasticity

optimize performance in live through
scaling requests

static elasticity

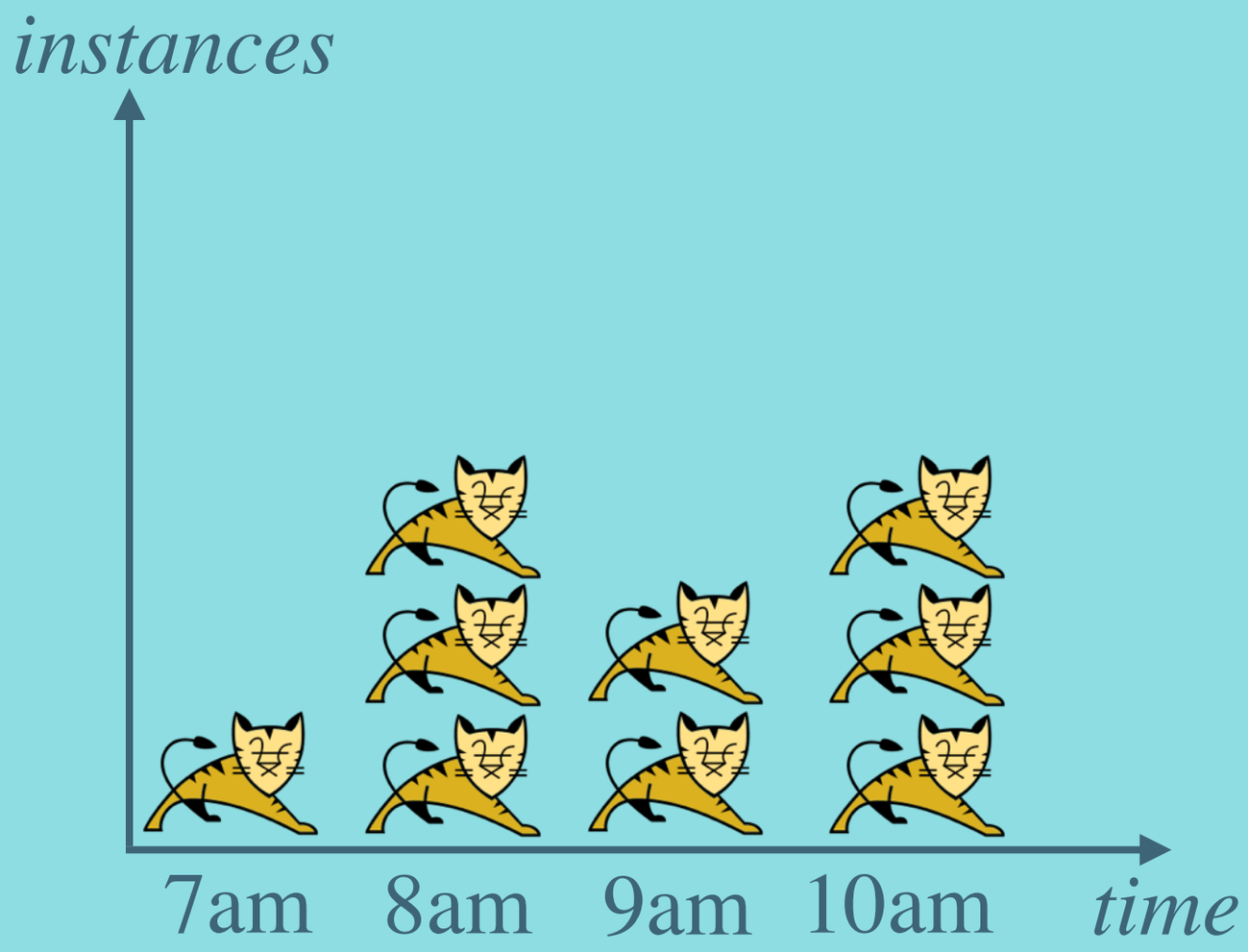
initiated by the administrator

not feedback based

error prone
under/over-estimations

static elasticity

initiated by the administrator



not feedback based

error prone
under/over-estimations

time-driven only ?

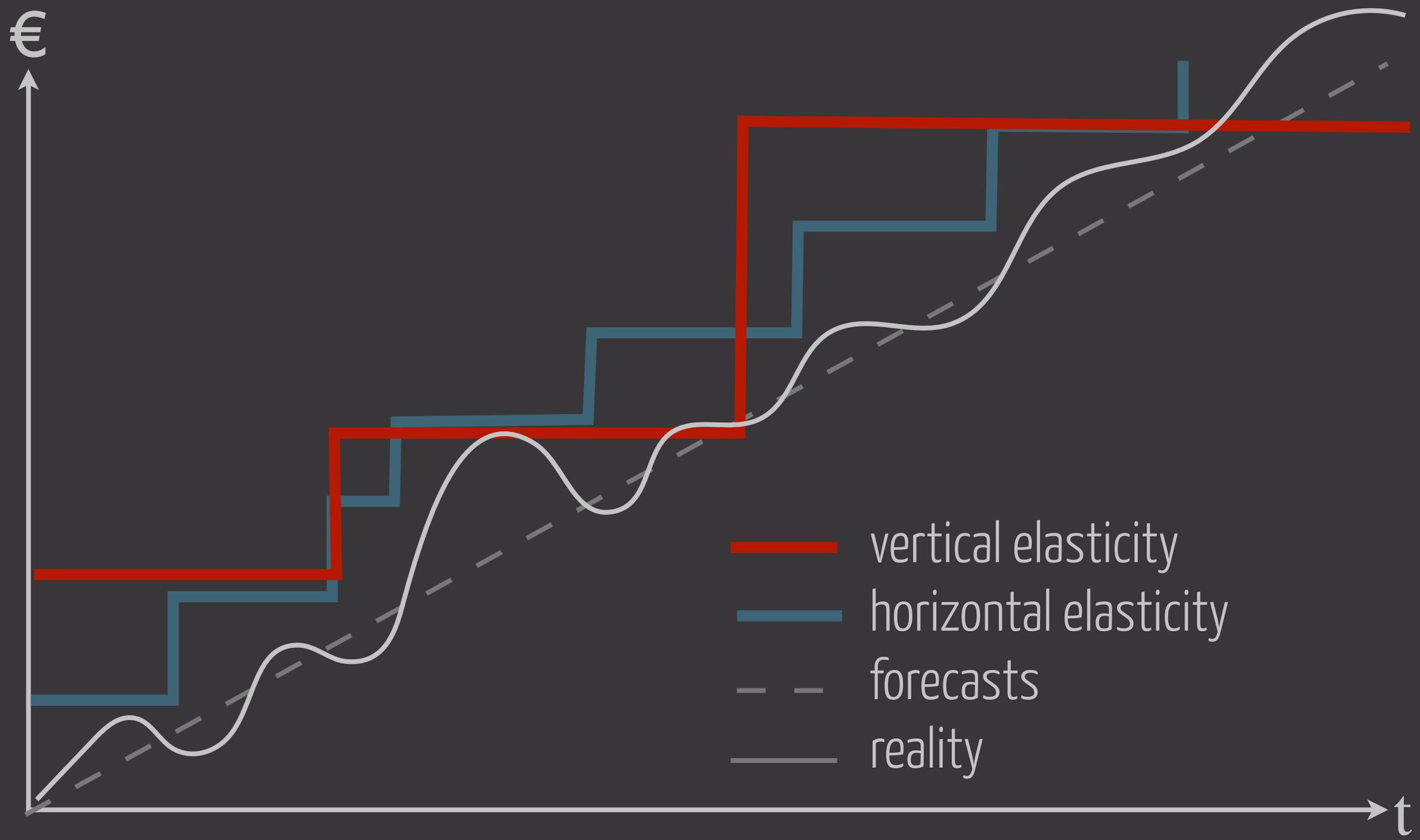
latency-aware elasticity

getting a VM takes up to 5 minutes

think in terms of trends

spare space just in case

static elasticity



dynamic elasticity

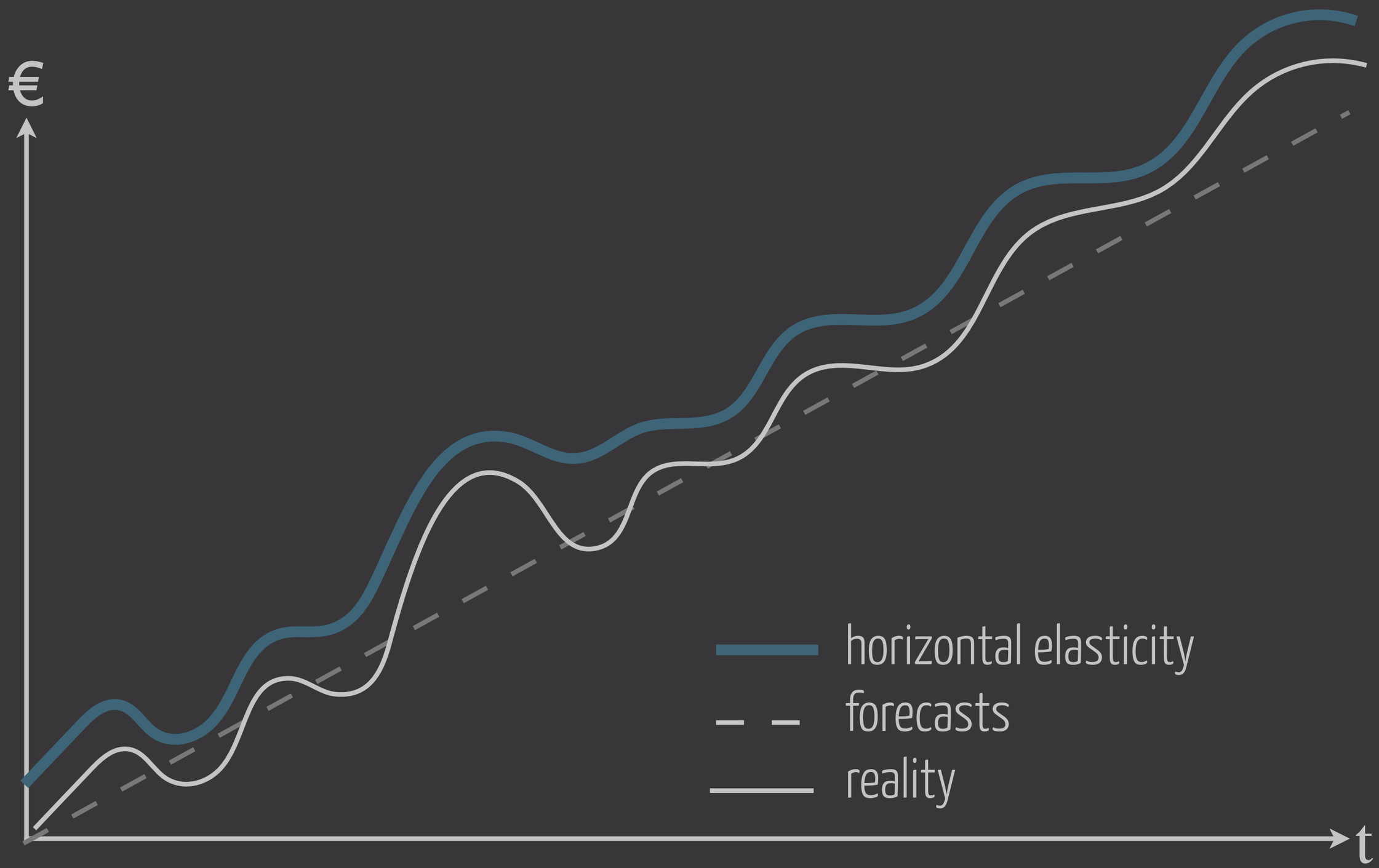
initiated by the app itself

rule based system

feedback from monitoring data

implemented inside/outside the app

dynamic elasticity



amazon web services™

Filter: Filter Auto Scaling groups... 1 to 2 of 2 Auto Scaling Groups

Name	Launch Configuration	Instances	Desired	Min	Max	Availability Zones	Default C
awseb-e-dnma...	awseb-e-dnmaa76xme-...	1	1	1	4	eu-west-1a, eu-west-1b, eu-...	360
WWWELB	my	1	1	1	10	eu-west-1b	100

Decrease Group Size

Actions

Execute policy when: awsec2-WWWELB-High-CPU-Utilization
breaches the alarm threshold: CPUUtilization < 20 for 300 seconds
for the metric dimensions AutoScalingGroupName = WWWELB

Take the action: Remove 1 instances

And then wait: 100 seconds before allowing another scaling activity

Increase Group Size

Actions

Execute policy when: awsec2-WWWELB-CPU-Utilization
breaches the alarm threshold: CPUUtilization >= 40 for 300 seconds
for the metric dimensions AutoScalingGroupName = WWWELB

Take the action: Add 1 instances

And then wait: 100 seconds before allowing another scaling activity

dynamic elasticity

scale where its matter

monitor each tier to indentify the bottlenecks

scale out apache/tomcat/mysql ?

cost model for elasticity

service cost

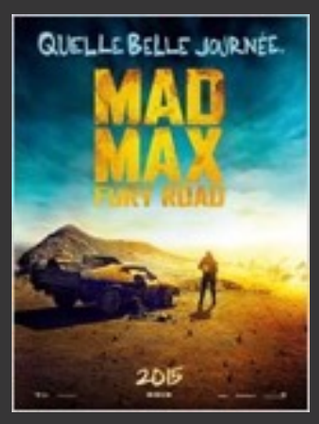
instance cost (hourly based)

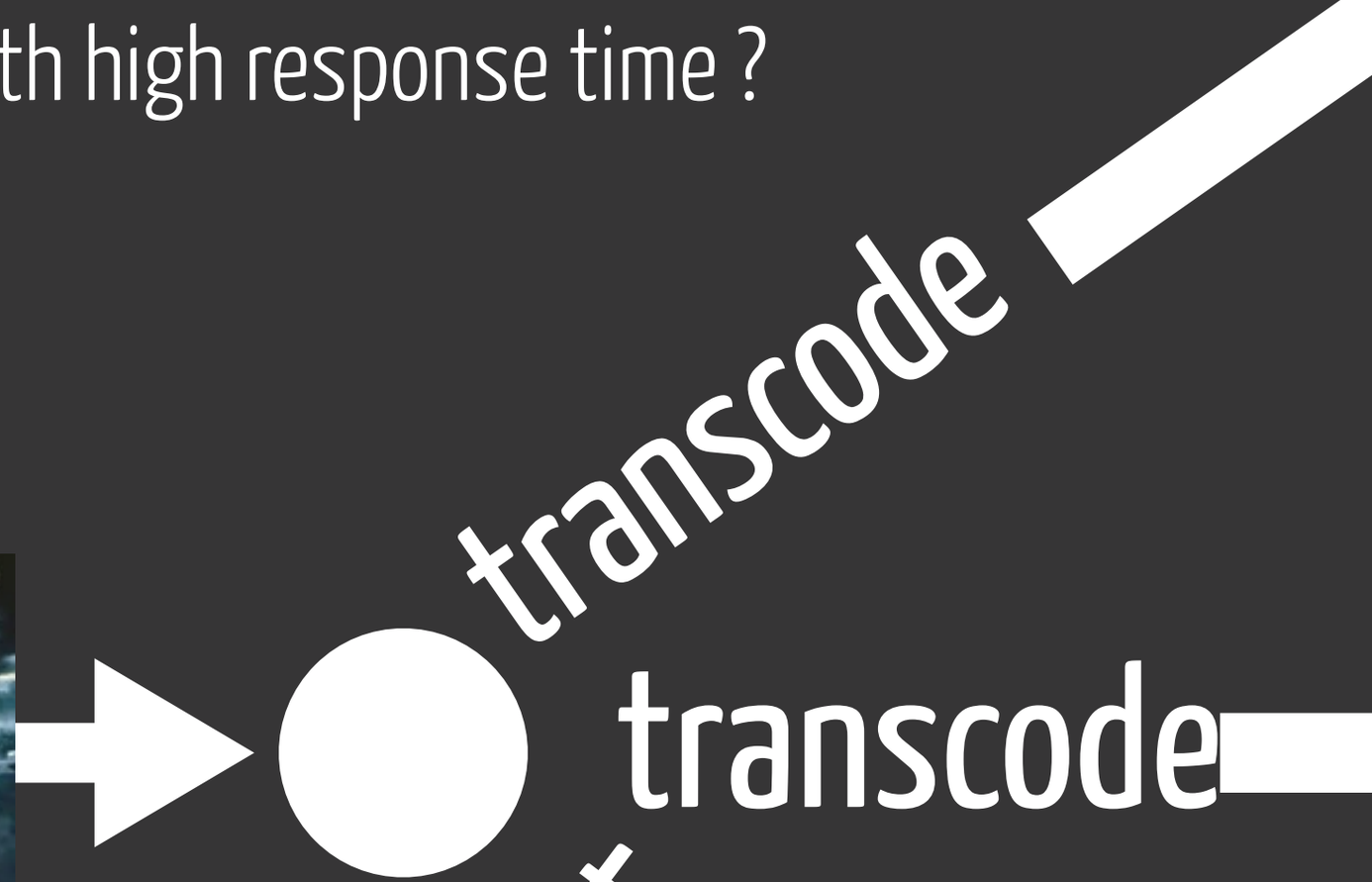
/!\ don't scale too often

load balancing

requests cannot stall in a load balancer

consequences with high response time ?

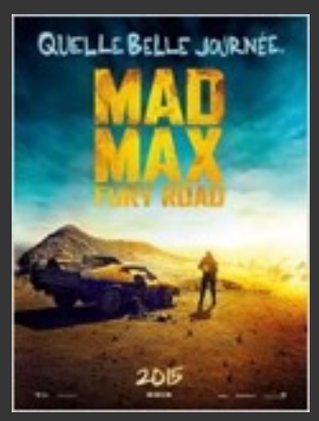


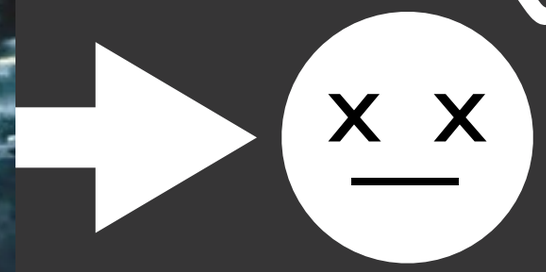


load balancing

requests cannot stall in a load balancer

consequences with high response time ?



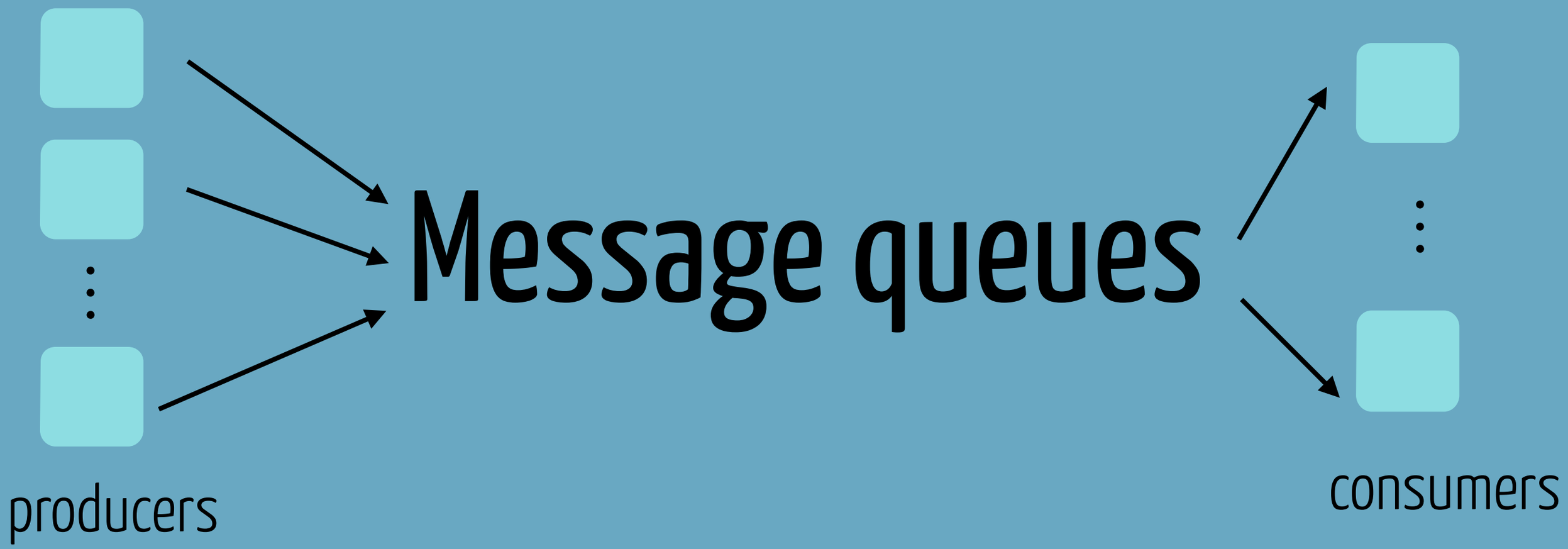


transcode

transcode

trans

asynchronous communication protocol
to transfer data



benefits

shift time consuming tasks to workers

loose coupling

queues are reliable

durable data

fault tolerant

eventual consistency

queues are scalable

workers

internal components

Programming model for workers

3 basic operations: dequeue — process — delete

1. ?
2. ?
3. ?

Programming model for workers

1. dequeue/delete
2. process

Programming model for workers

1. dequeue/delete
2. process

message lost

Programming model for workers

1. dequeue
2. process
3. delete

« at least 1 one read » ?

1. dequeue
2. process
3. delete

what is there is a node failure
while processing

Programming model for workers

invisibility window

messages are not definitely dequeued

hidden for a period (configurable)

removed once deleted

a timeout makes the message visible again

Programming model for workers

1. dequeue
2. process
3. delete

invisibility window

realize eventual consistency

1. dequeue
2. process
3. delete

what if
processing time > invisibility window

Programming model for workers

1. dequeue
2. process
3. delete

eventual consistency makes possible to process a message twice.

Take care !

billing model for queues

#request

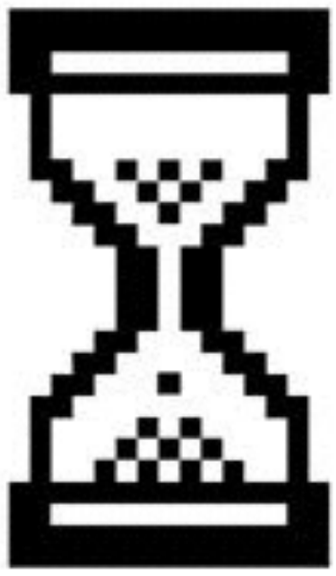
#Data transfered

/!\ pull mode

Amazon SQS
Simple Queuing System

dealing with latency

latency over business



>

€

Google search

+500 ms   -20% traffic

+100 ms 

-1% $$

Latency numbers

L1 cache reference ..... 0.5 ns
Branch mispredict ..... 5 ns
L2 cache reference ..... 7 ns
Mutex lock/unlock ..... 25 ns
Main memory reference ..... 100 ns
Compress 1K bytes with Zippy ..... 3,000 ns = 3 µs
Send 2K bytes over 1 Gbps network ..... 20,000 ns = 20 µs
SSD random read ..... 150,000 ns = 150 µs
Read 1 MB sequentially from memory ..... 250,000 ns = 250 µs
Round trip within same datacenter ..... 500,000 ns = 0.5 ms
Read 1 MB sequentially from SSD* ..... 1,000,000 ns = 1 ms
Disk seek ..... 10,000,000 ns = 10 ms
Read 1 MB sequentially from disk .... 20,000,000 ns = 20 ms
Send packet CA->Netherlands->CA .... 150,000,000 ns = 150 ms

colocate

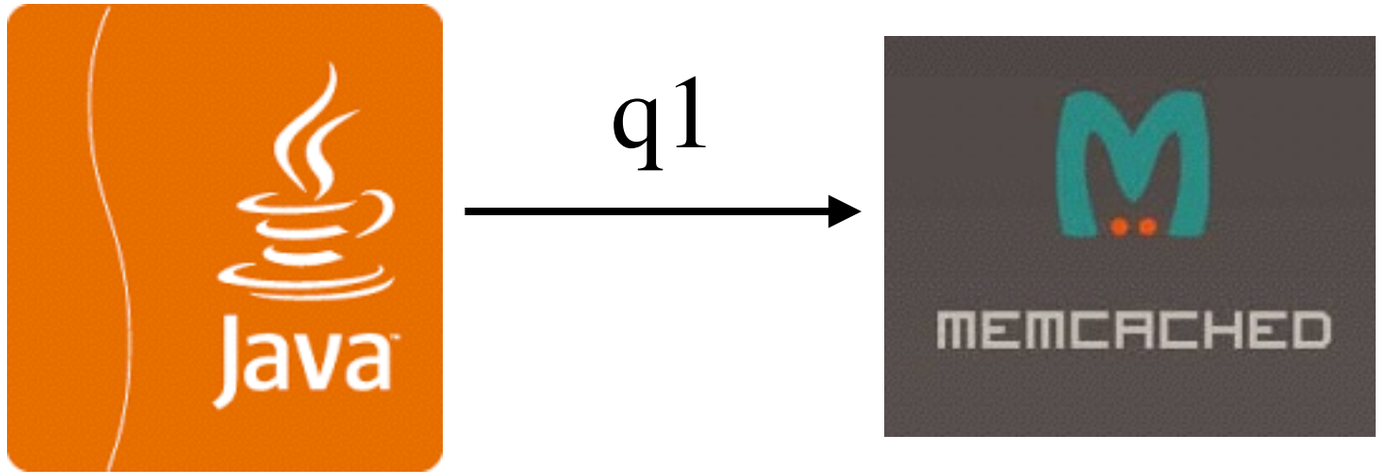
same process, server, rack, datacenter

1 Gb/sec

reduce latency
reduce data transfer costs

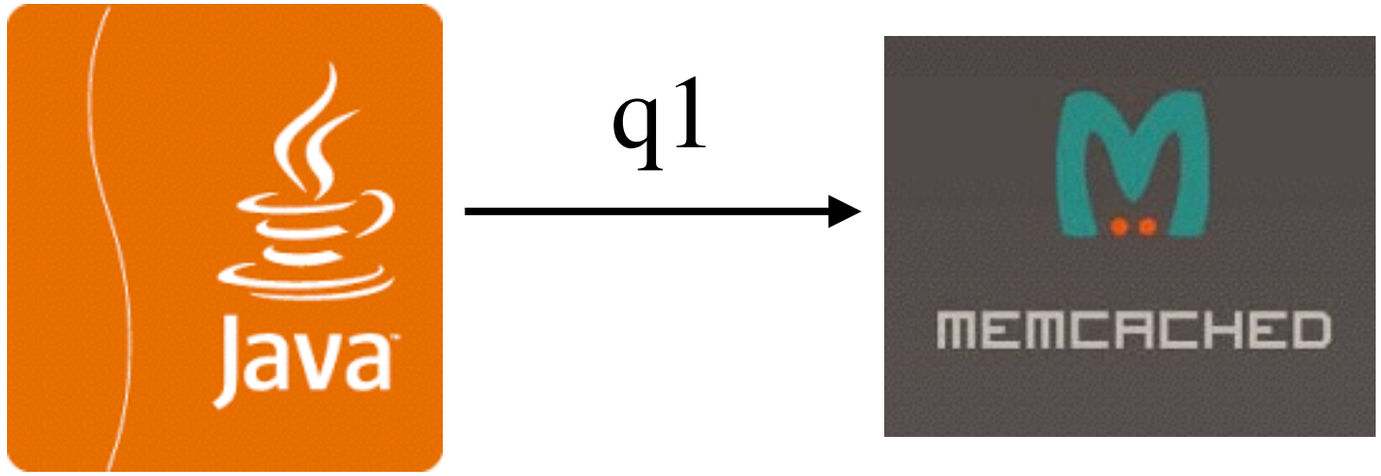
cache

to store data closely
and speed up future accesses



cache

to store data closely
and speed up future accesses



q1 ?

cache

to store data closely
and speed up future accesses

cache miss

cache

to store data closely
and speed up future accesses

cache miss

cache

to store data closely
and speed up future accesses

cache miss

q1: d1

cache

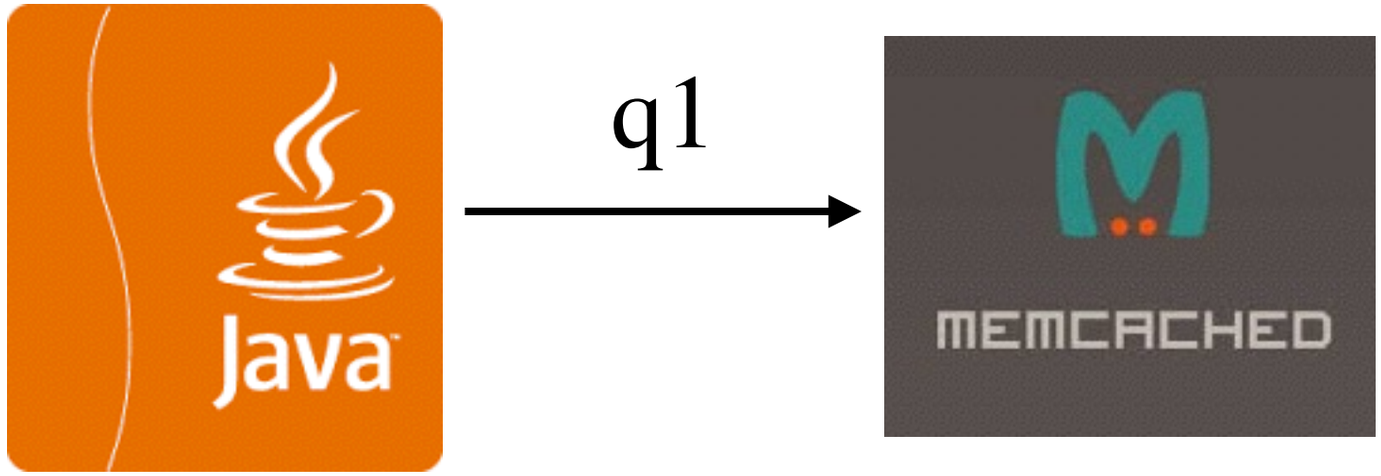
to store data closely
and speed up future accesses

cache miss

q1: d1

cache

to store data closely
and speed up future accesses

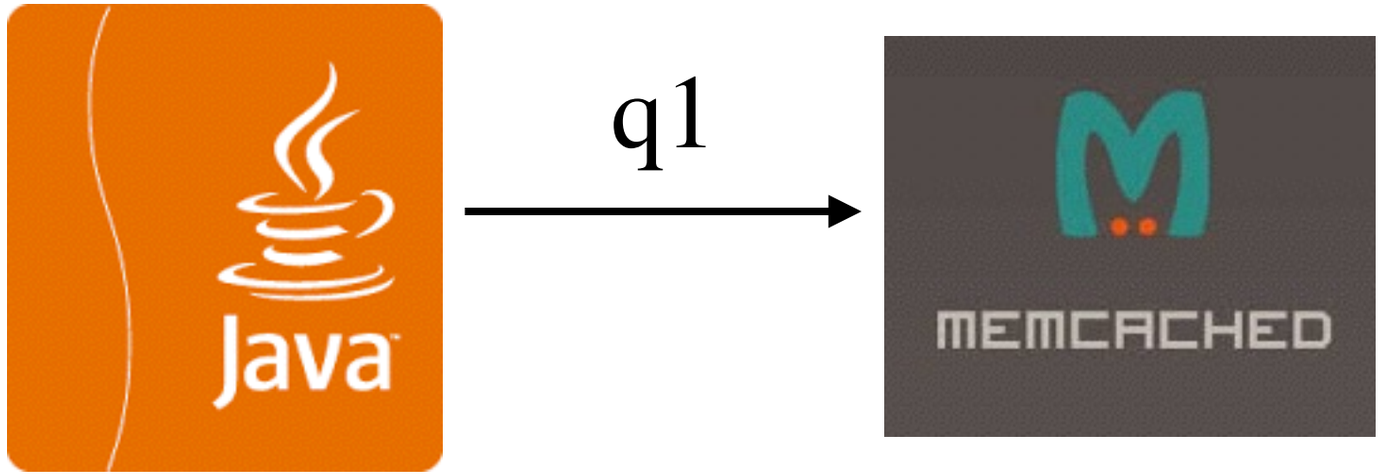


q1: d1

cache

to store data closely
and speed up future accesses

cache hit

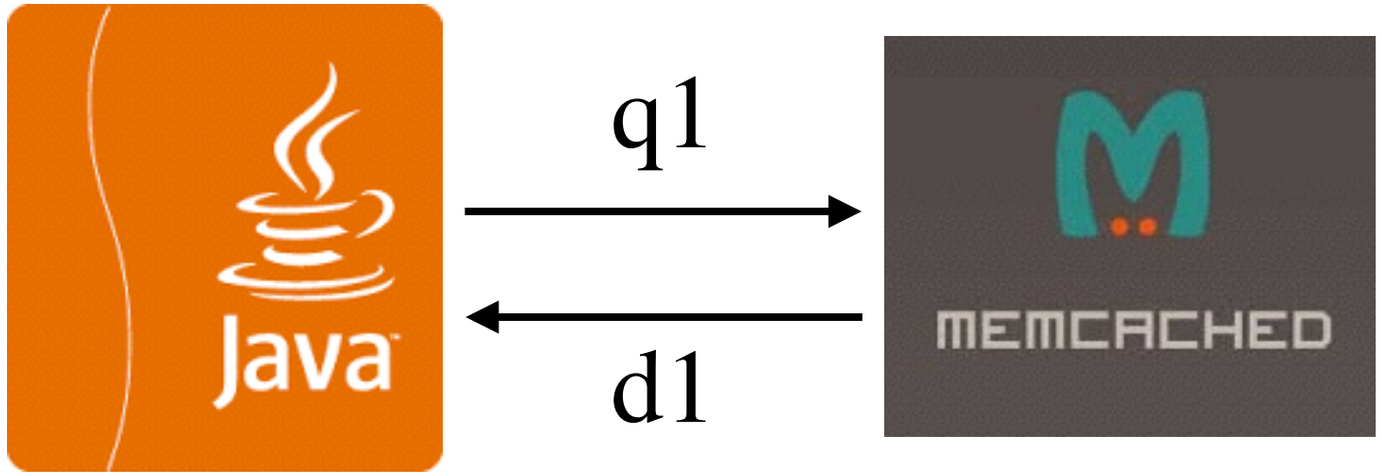


q1: d1

cache

to store data closely
and speed up future accesses

cache hit



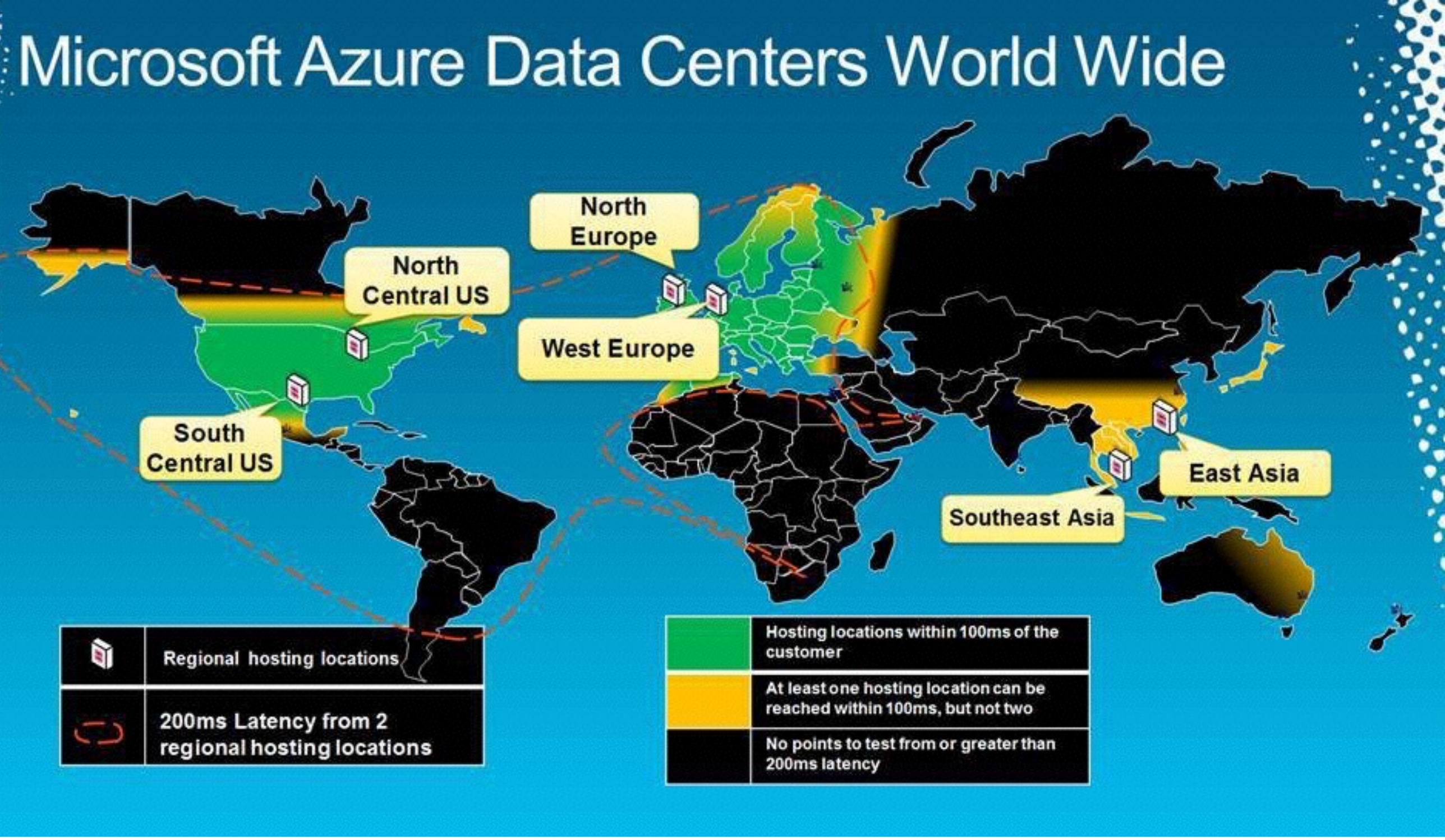
q1: d1

cache everything everywhere

« Cache is the new RAM »

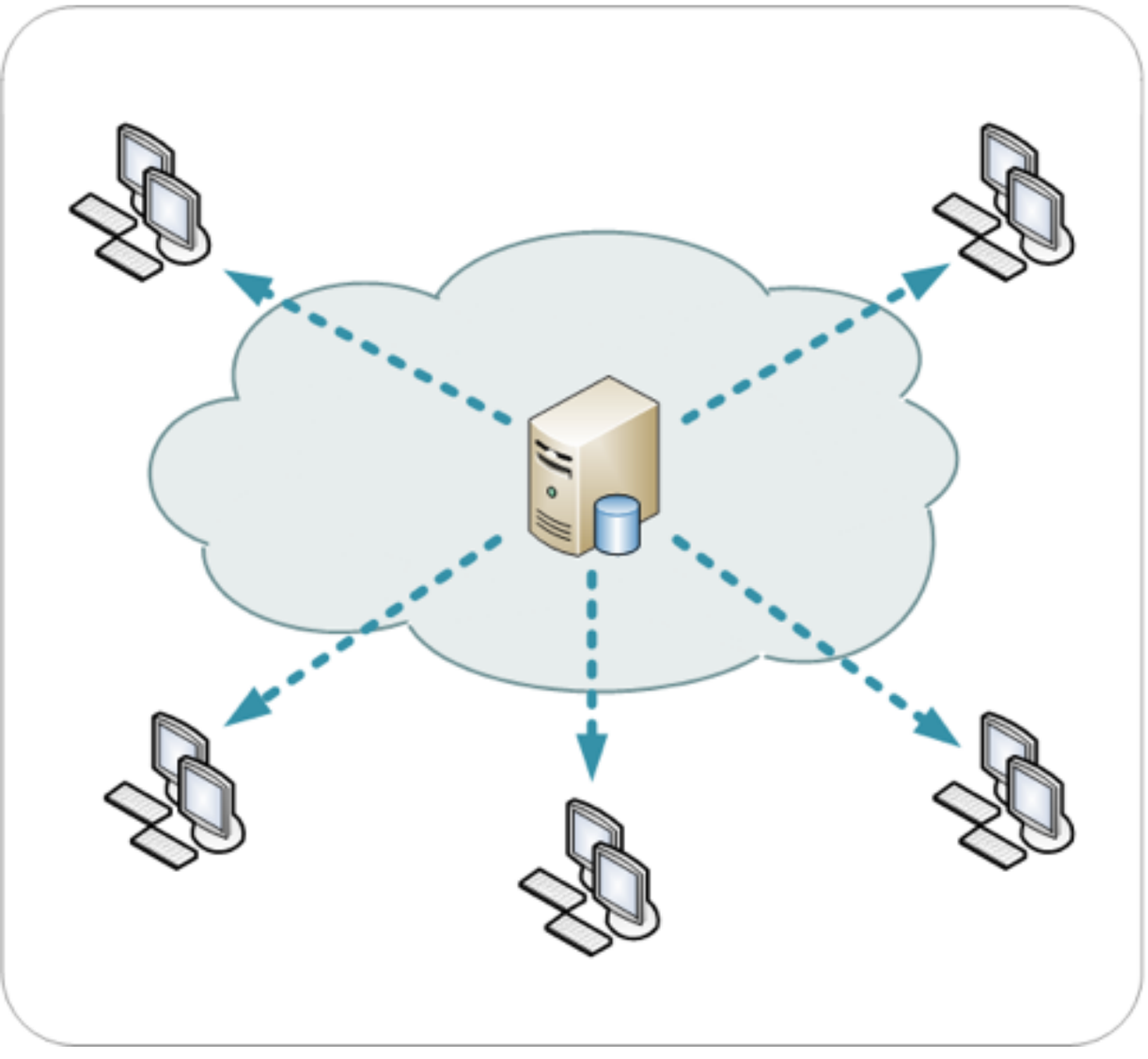
Getting close to clients

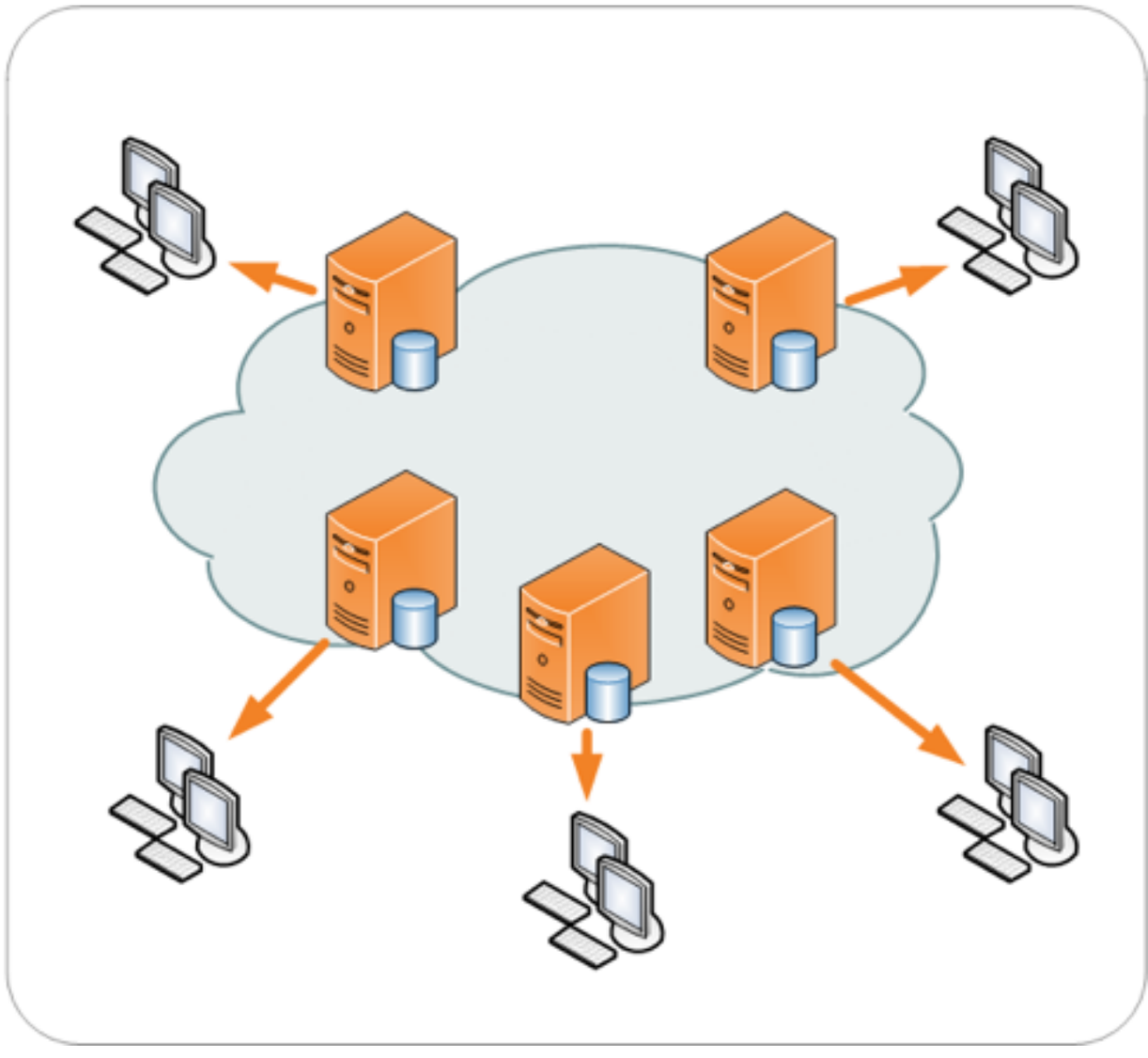
Microsoft Azure



Getting close to the clients

Content Delivery Networks (CDN)





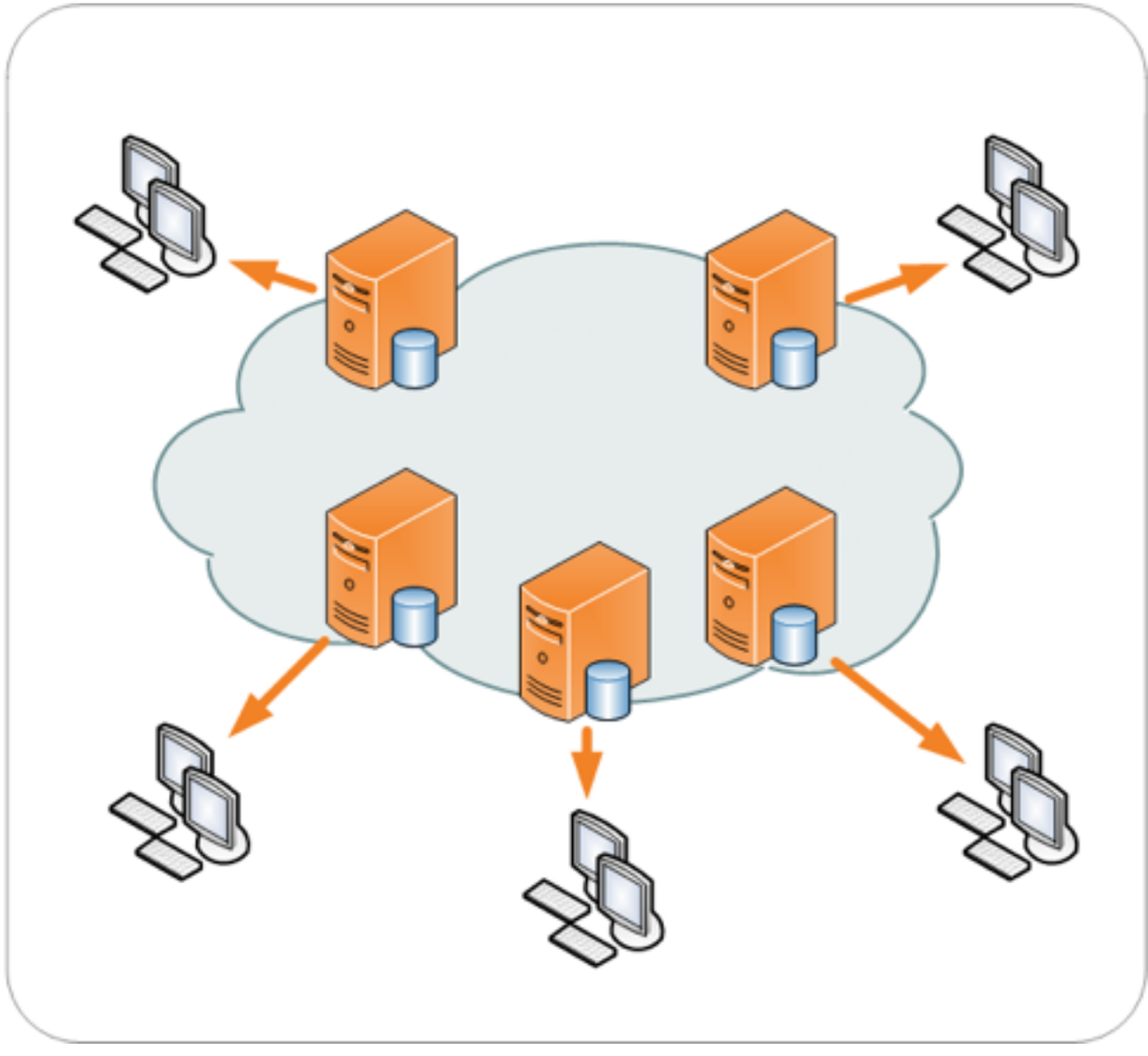
Content Delivery Networks (CDN)

available at network edges

data are injected into the CDN

the CDN spreads the data where it matters

user requests are redirected to the closest Point of Presence (PoP)




Deploy

Zero Downtime Deployment

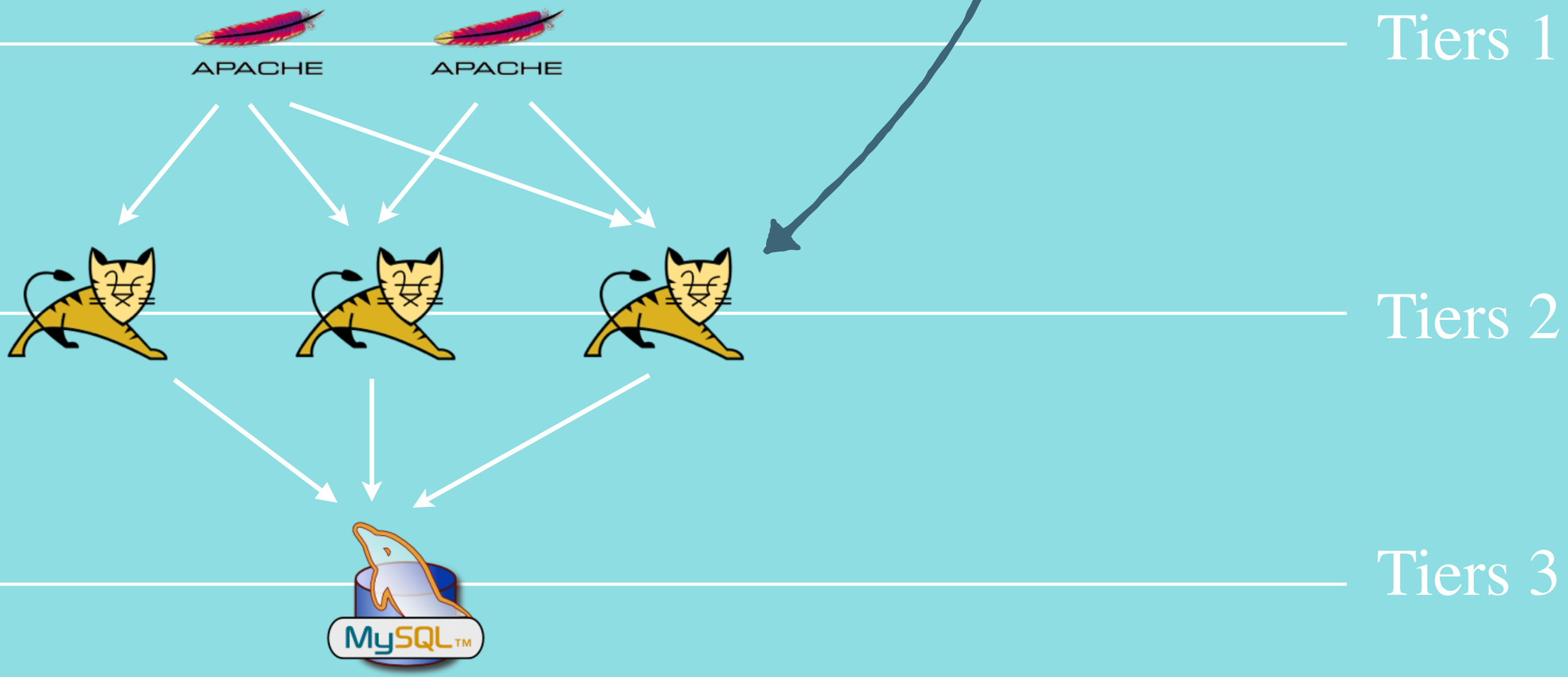
What to deploy

new features

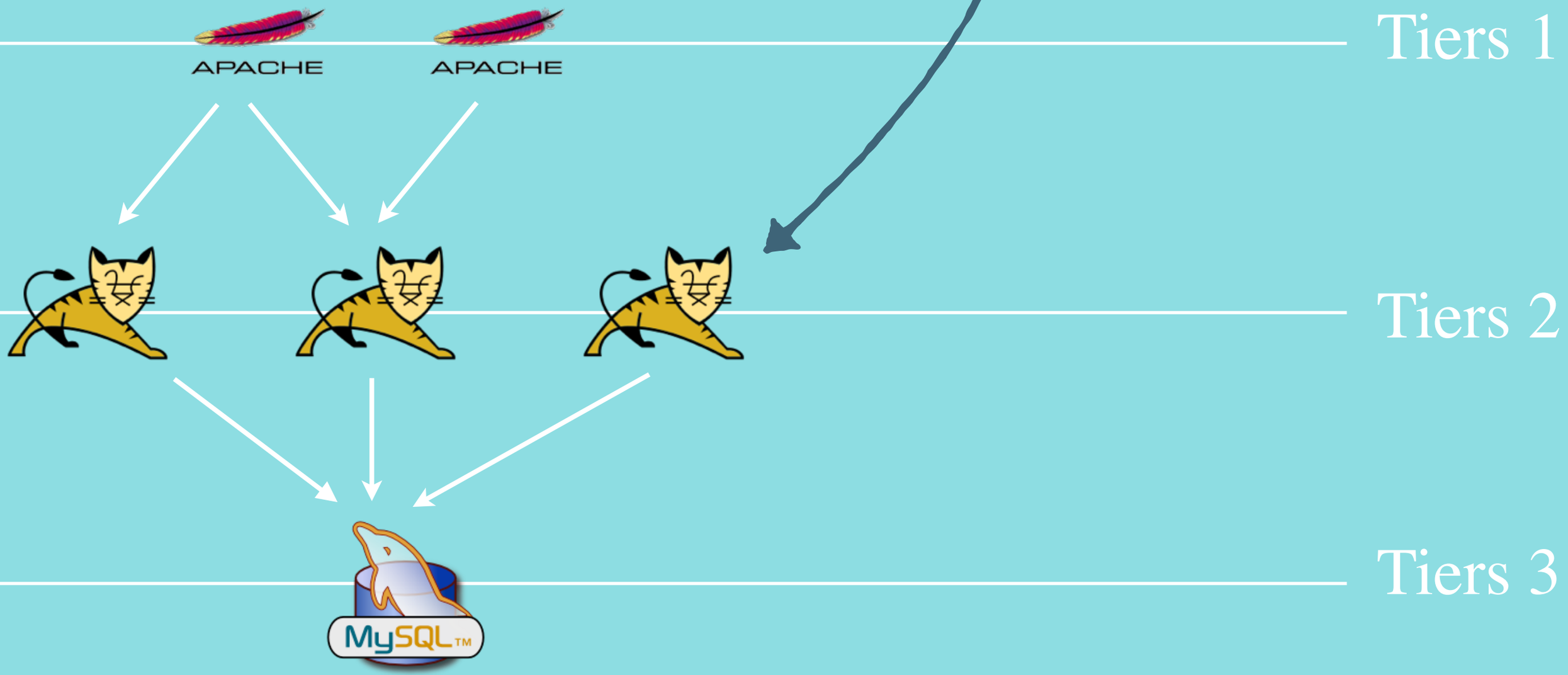
internal changes

fixes

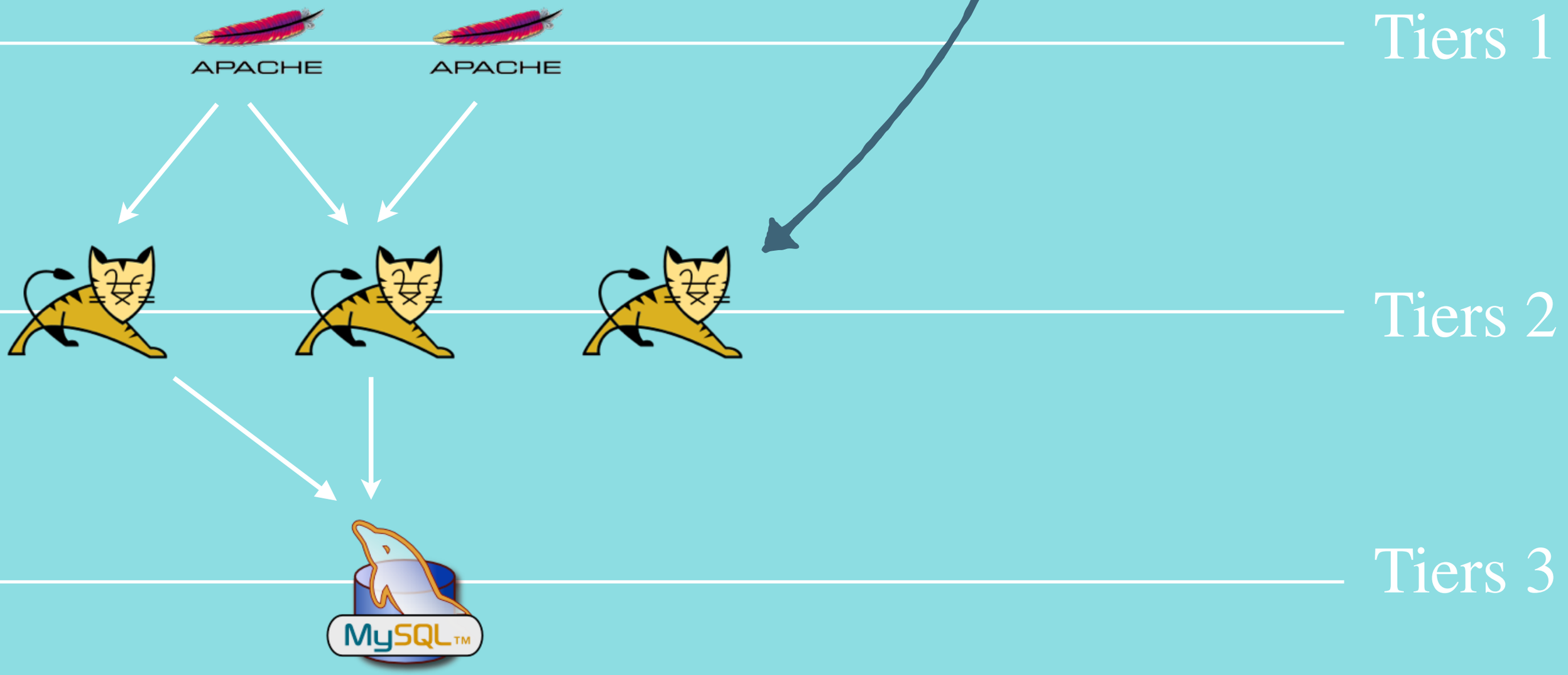
how-to upgrade gracefully ?



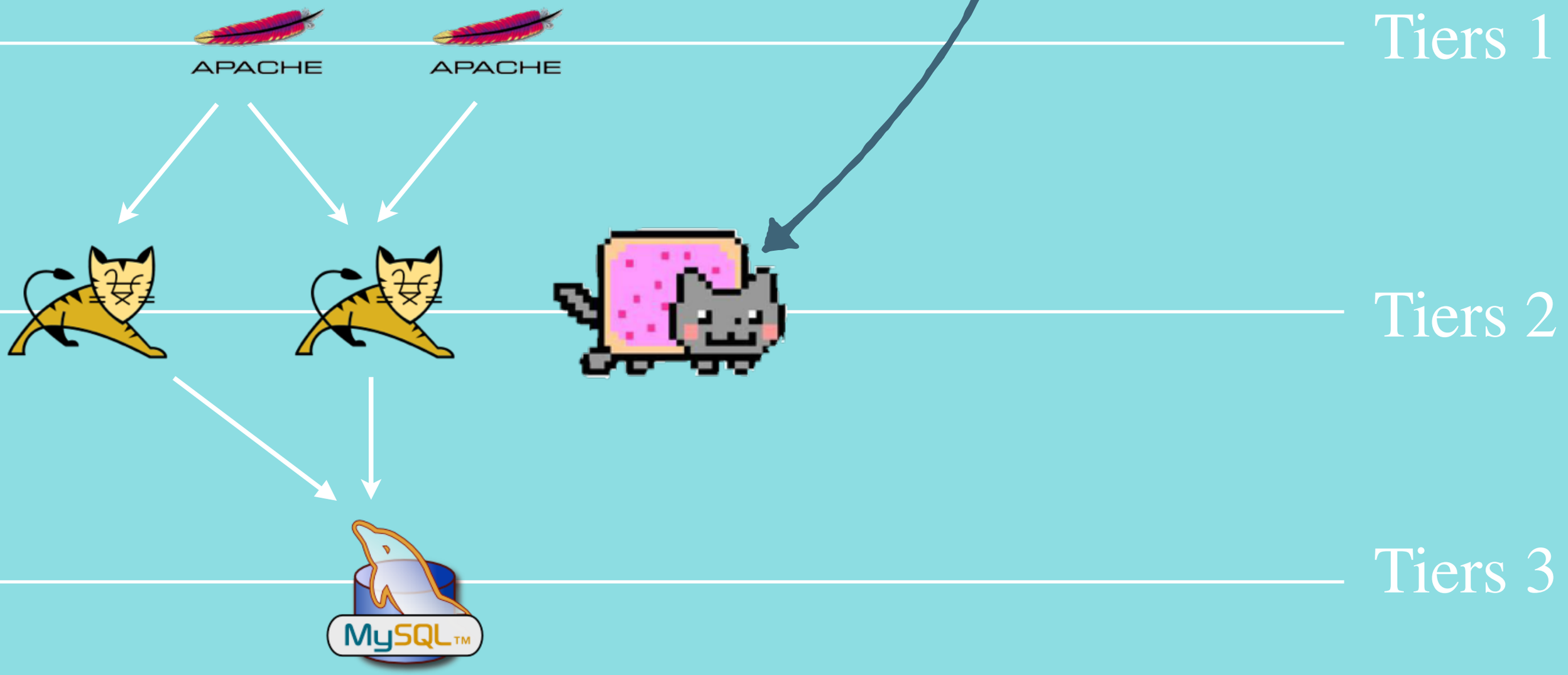
how-to upgrade gracefully ?



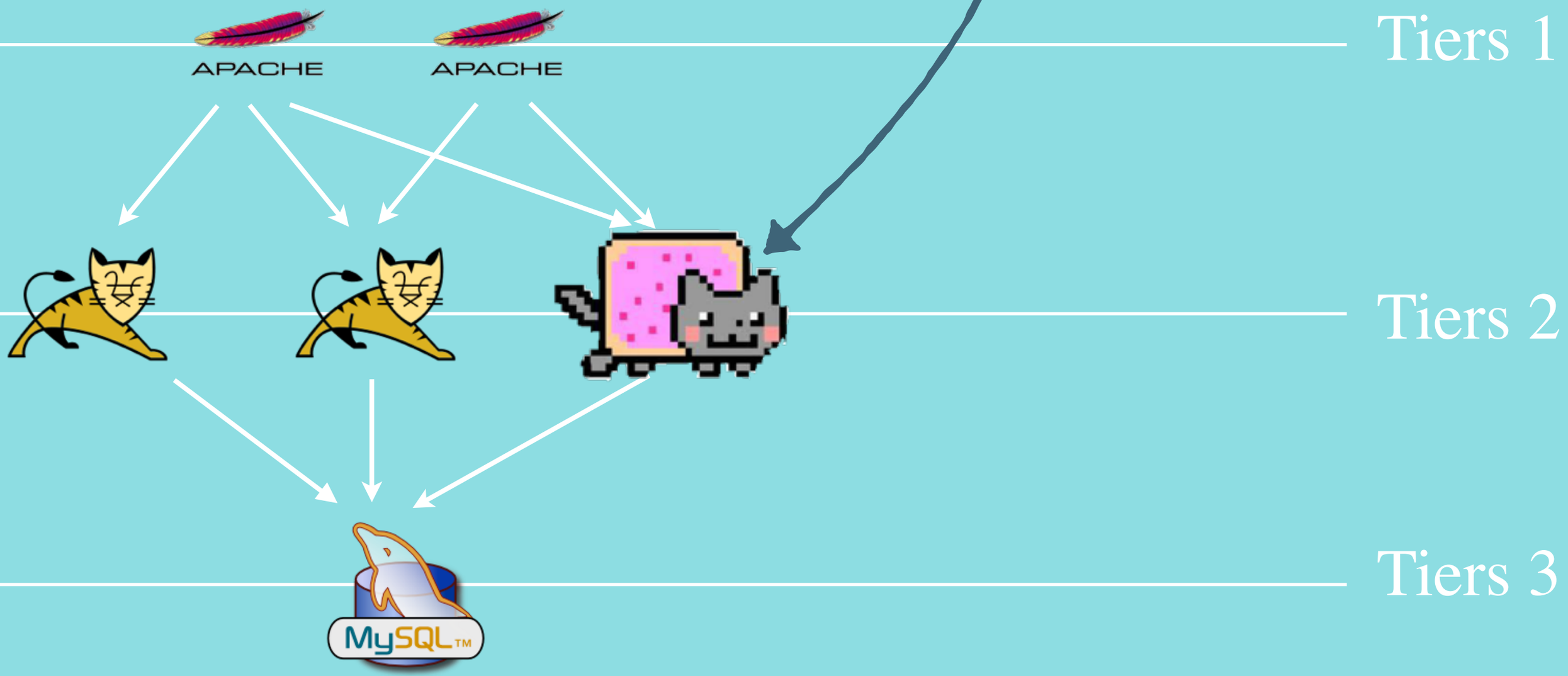
how-to upgrade gracefully ?



how-to upgrade gracefully ?



how-to upgrade gracefully ?



move fast & break nothing

a talk about code, teams & process by @holman

(that guy make good talks about github processes)

There is some code you can't break

permission - billing - upgrades - maintenance

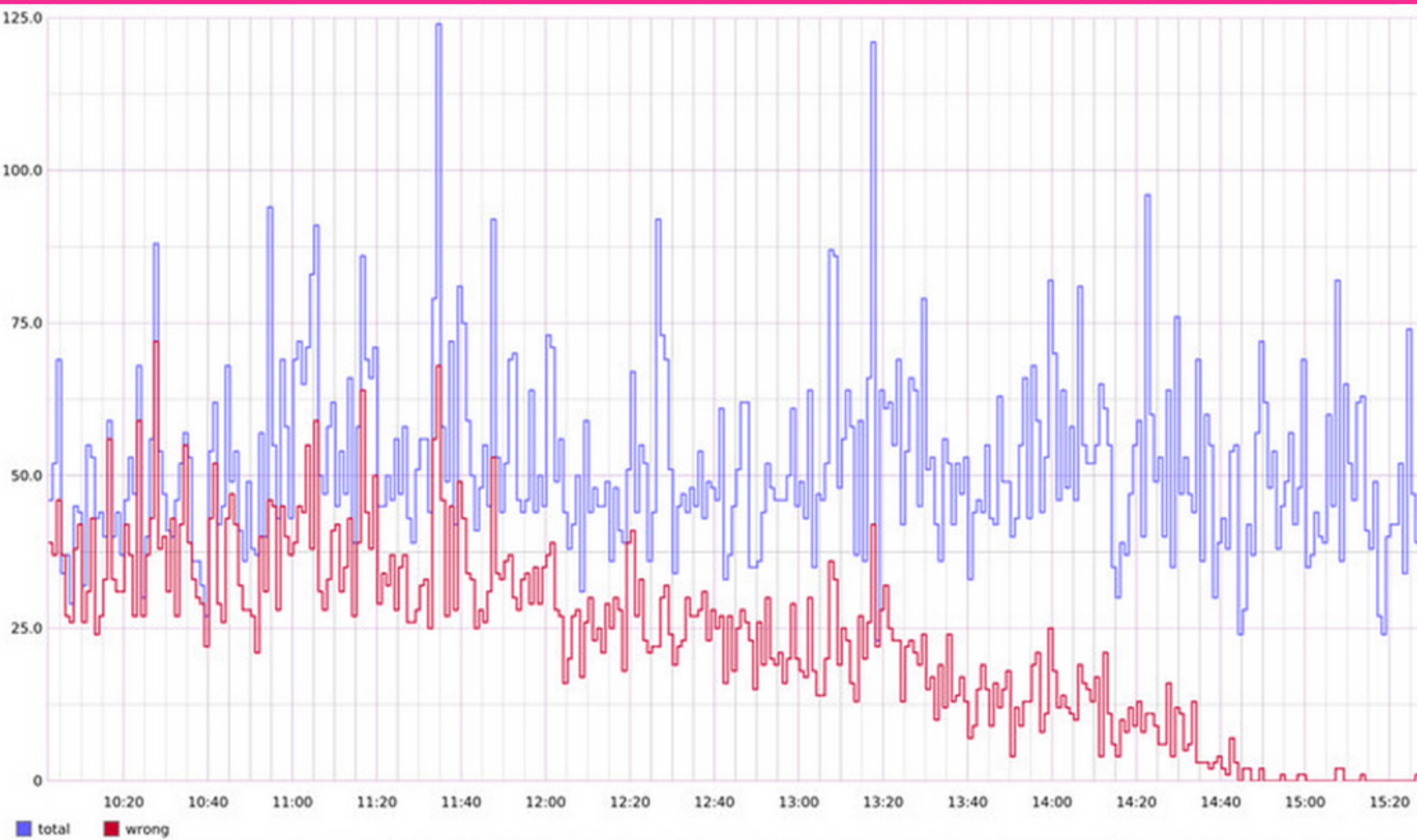
tests are not enough

new code can't change production code behavior

parallel paths

```
science « new auth » do |e|
    e.control{user.slow_auth}
    e.candidate{user.fast_auth}
end
```

monitor progress



at facebook

2 releases a day

the frontend is a standalone 1.5GB executable

deployment over bitorrent

no downtime

see Push: tech talk

at facebook

canary release

dev fenced to the next release
(6 private servers)

over 2% of the production servers

over 100% of the production servers

at facebook

decoupling

stateless sessions

backward & forward compatible UI

dark launches & feature flag

integration tests

test if images are ok

test if every files are in a cdn

test if css is clean

everything is automated

CI + grunt

tumblr.

(in 2012)

http://highscalability.com/

tumblr.

500 M page views a day

Peak rate of ~40k req/sec

1+ TB/day into Hadoop cluster

Posts: 50GB a day

Follower list updates: 2.7TB a day

Dashboard: 1M w/sec, 50K r/sec

tumblr. origin

simple LAMP application on rackspace
everything on a single server

backend service in C

memcache
CDN

HA-proxy
MySQL sharding for the blog

Redis for the dashboard

tumblr.

moved to high-concurrency oriented frameworks

JVM centric approach — for hiring and dev. speed
PHP just for request authentication and presentation
scale, finaggle — support from the big guys

HBase + Redis but still MySQL for bulk data

Redis for the dashboard notification

tumblr.

dynamic isolation of users into cells

standalone app with its database
once logged, a cell is assigned to the user,
populated with data
cell are populated for live events by a stream

isolation eases parallelism
small isolated components isolate failures

tumblr.

500 web servers
200 database servers
47 pools
30 shards
30 memcache servers
22 redis servers
15 varnish servers
25 haproxy nodes
8 nginx
14 job queue servers

RECAP

always

available

performant at any scale

cost

effective

Developing within a PaaS

PaaS to create and push a cloud application

support for multiple software stacks

Ruby, PHP, Pyhon, Java, Node.js, Docker, Go

deployment method

git, files

EC2, S3, SNS, CloudWatch, AutoScaling,
Elastic load balancers, SQS

a low-level HTTP API

official binding for popular languages

official eclipse plugin

tons of documentation, tutorial
on AWS website

runtimes on top of EC2

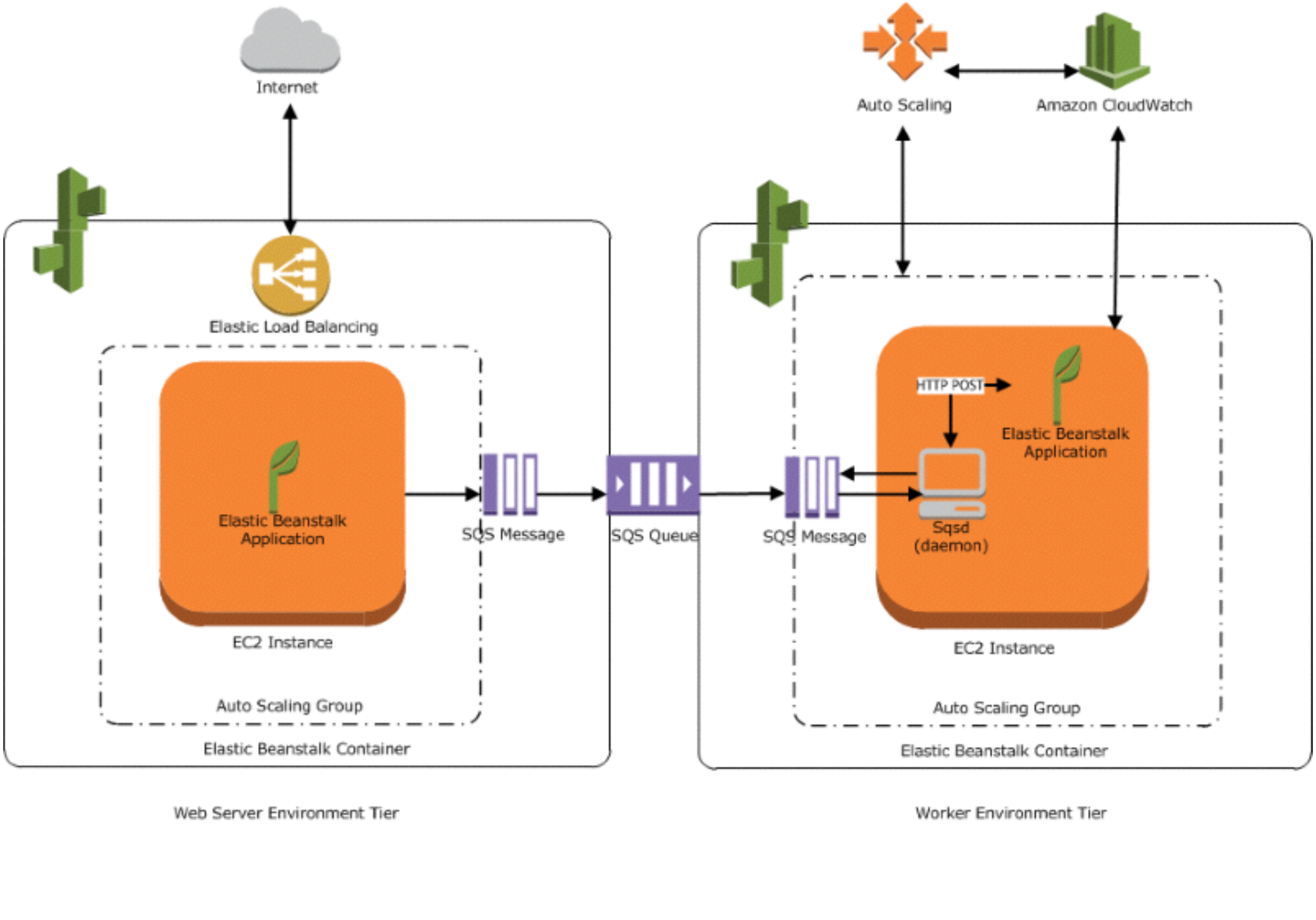
multiple environments
(web, worker, data)

deploy: env version + env configuration

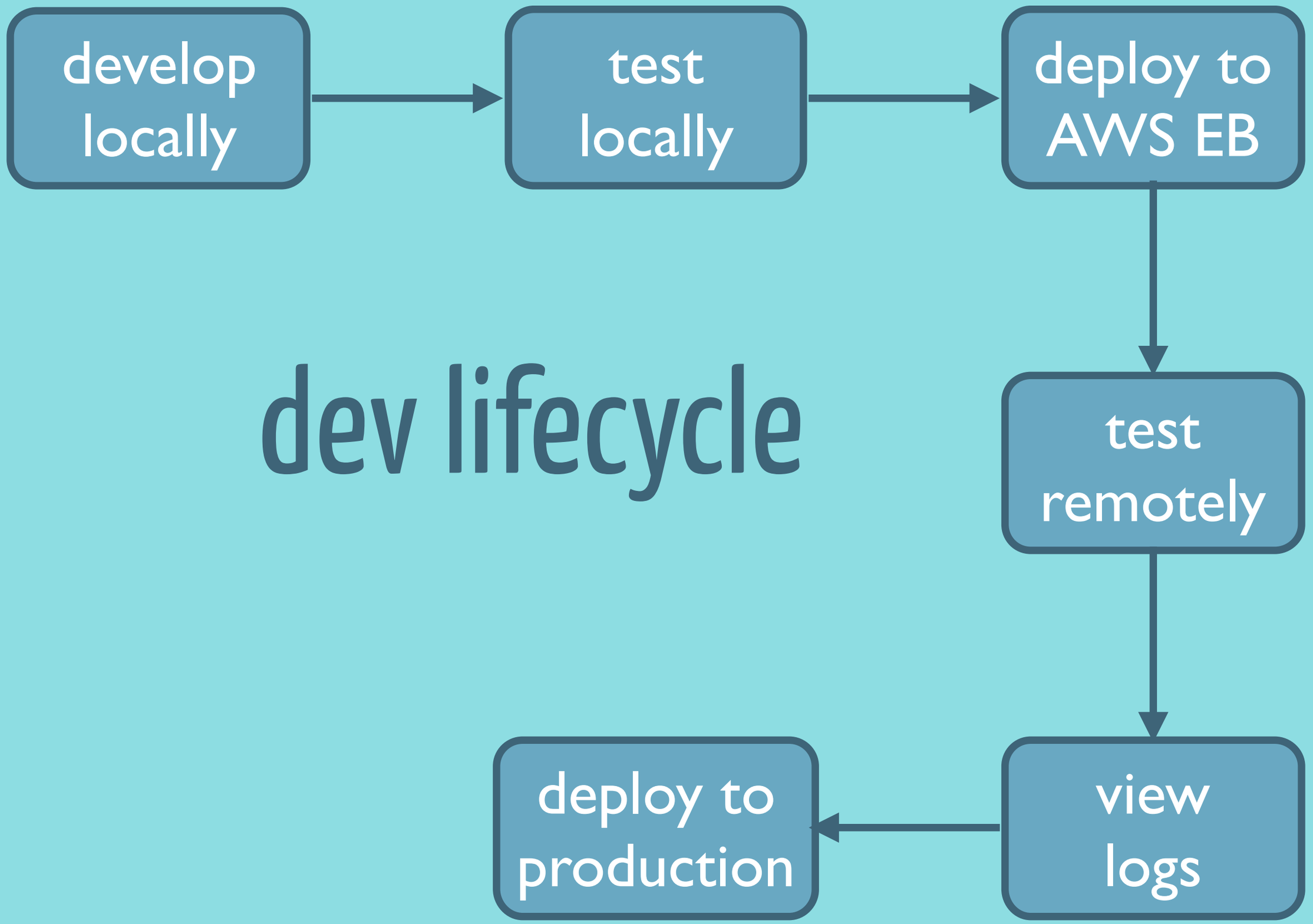
env version comes from S3

Elastic HTTP balancers
between the envs.

security, security, security,


Internet
Auto Scaling
Amazon CloudWatch
Elastic Load Balancing
HTTP POST
Elastic Beanstalk Application
Elastic Beanstalk Application
SQS Message
SQS Queue
SQS Message
Sqsd (daemon)
EC2 Instance
EC2 Instance
Auto Scaling Group
Auto Scaling Group
Elastic Beanstalk Container
Elastic Beanstalk Container
Web Server Environment Tier
Worker Environment Tier

dev lifecycle



Go online !!