Microservices.

Worauf es wirklich ankommt.

Leon Rosenberg
@dvayanu
Bed Con 2015


Hello...

Who am I

- Leon Rosenberg, Java Developer, Architect, OpenSource and DevOps Evangelist.
- 1997 Started programming with Java
- 2000 Started building portals
- 2007 Started MoSKito


FRIEND
SCOUT 24
PAYBACK
C-Date
PARSHIP.com
Find someone right for you
MOSKito
All you need

Was sind die typischen **Probleme** und wie löst man sie? Wie **baut** man **elastische** und **robuste** Microservices-Anwendungen, wie **monitored** man sie, und was passiert wenn es **kracht**.

So what are we talking about?

In short, the microservice **architectural style** is an approach to developing a single application as a suite of small **services**, each running in its own process and **communicating** with lightweight mechanisms, often an HTTP resource API

http://martinfowler.com/articles/microservices.html

A service-oriented architecture (SOA) is an **architectural pattern** in computer software design in which application components provide **services** to other components via a **communications** protocol, typically over a network. The principles of service-orientation are independent of any vendor, product or technology.

https://en.wikipedia.org/wiki/Service-oriented_architecture

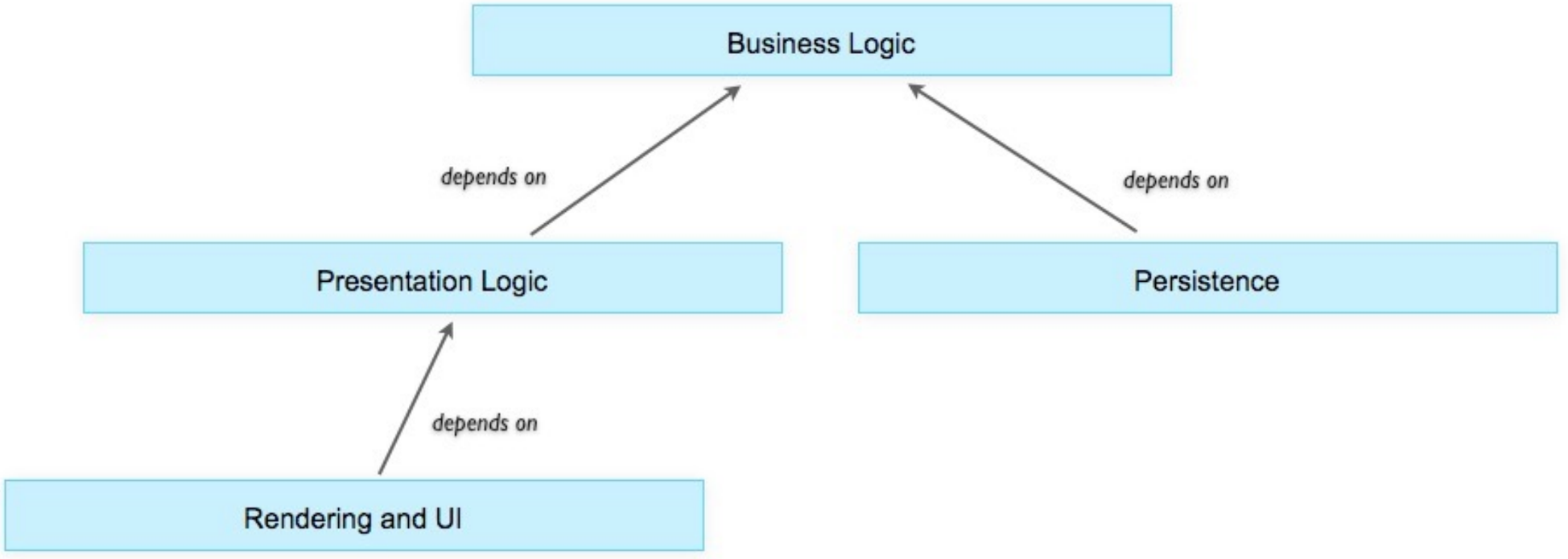
Microservices = SOA - ESB

Architecture

- Paradigms
- Communication
- Conventions

Paradigms

- Design by … (responsibility)
- Dumb vs. Smart Data
- Communication
- Trades


Business Logic
depends on
depends on
Presentation Logic
Persistence
depends on
Rendering and UI

Communication

- Synchronous vs Asynchronous
- 1:1, 1:n, n:m
- Direction
- Cycles

Problems

- Distributed transactions
- Too many calls (performance)
- Repetitions
- Communication overhead

Distributed transactions

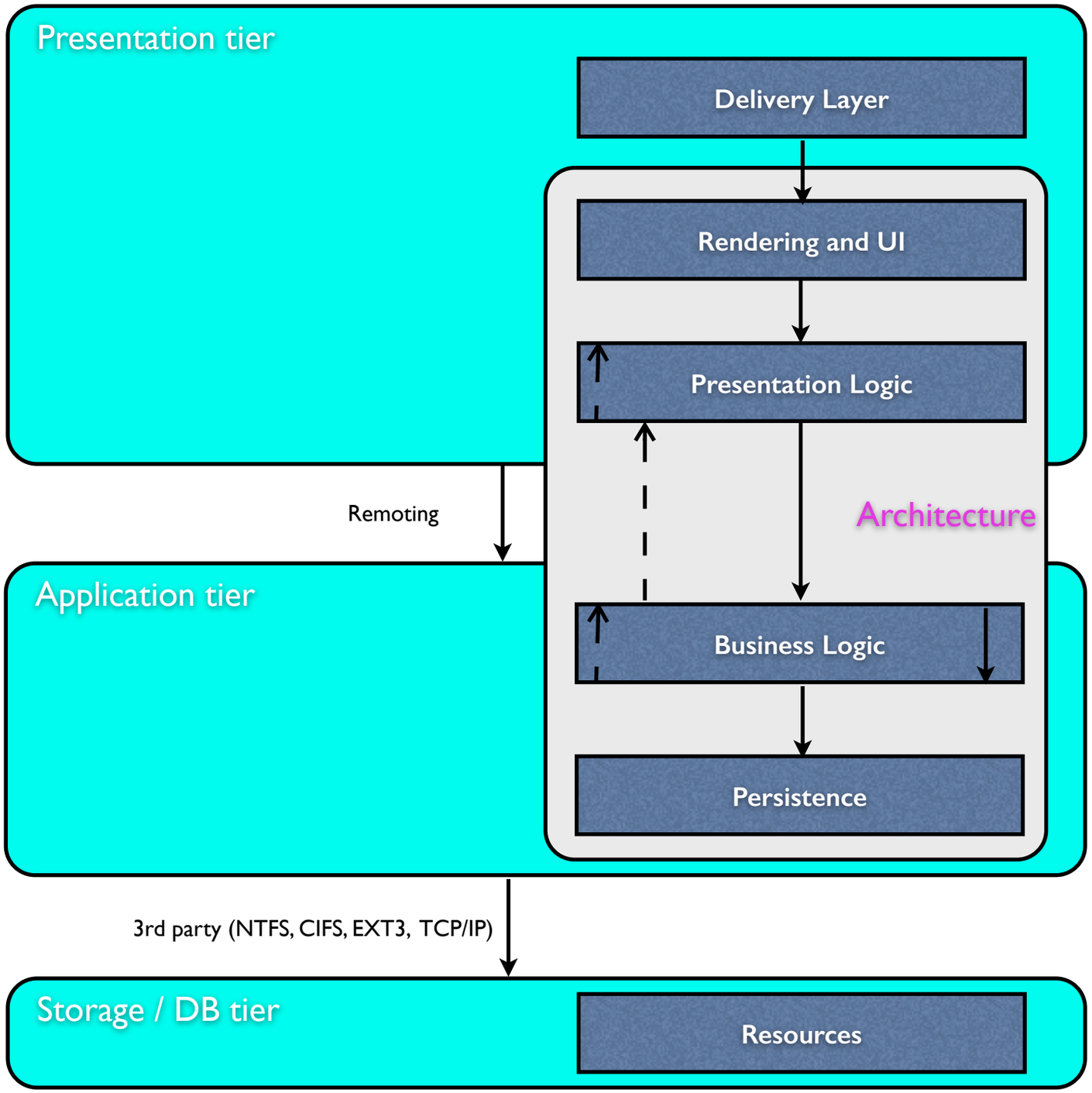
- Manual rollback.
- Special services (OTS).
- Allow it (order of modification).
- Consistency checks.
- Handle it when you need to.

Too many calls

- Combine calls.
- Execute calls in parallel.

Repetition

- Frontend User != Service User.
- Same steps are repeated over and over again.
- Separate business and presentation logic.
- Provide a service like client-side API for frontend, Presentation API.



loadbalancer, apache, squid

spring-mvc/struts/…

api

services, processes

DAOs, Exporter, Importer, FS-Writer

Postgresql, Mongo, FS

Caches

- Object cache.
- Expiry/Proxy/Client-side cache.
- Query cache.
- Negative cache.
- Partial cache.
- Index.

Just one service?

- Single point of failure
- Bottleneck
- Generally considered extremely uncool

Multiple Instances

- Failing strategy
- Routing

Failing

- Fail fast.
- Retry once/twice/…
- Failover to next node (and return or stay).
- Failover for xxx seconds.

Routing / Balancing

- Round-Robin
- Sharding
- Sticky

Combinations

- Round-Robin / Repeat once
- Failover for 60 seconds and return
- Mod 3 - Sharded with Repeat twice and failover to next node

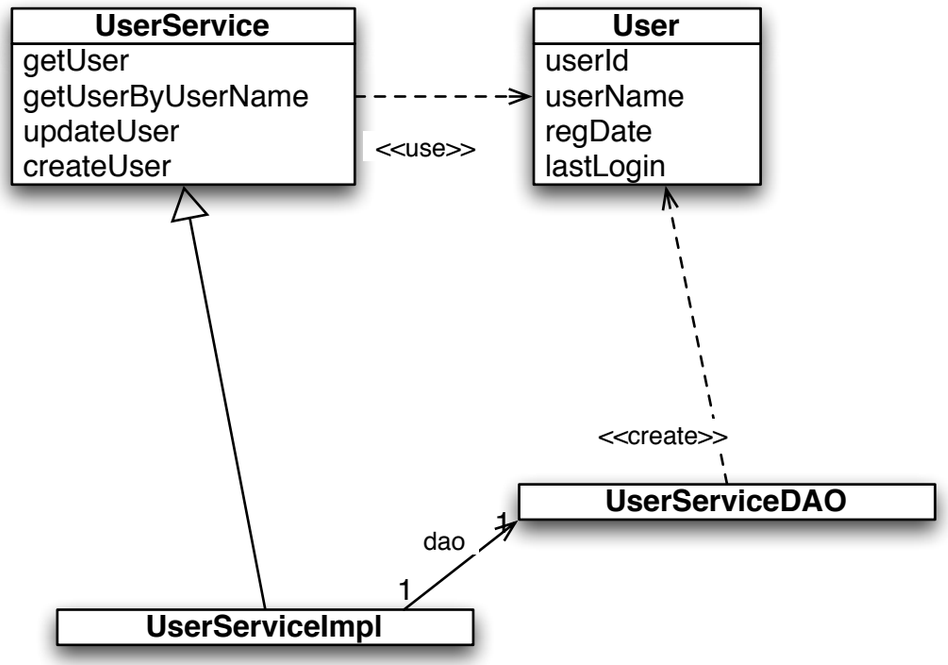
Non-Mod-able

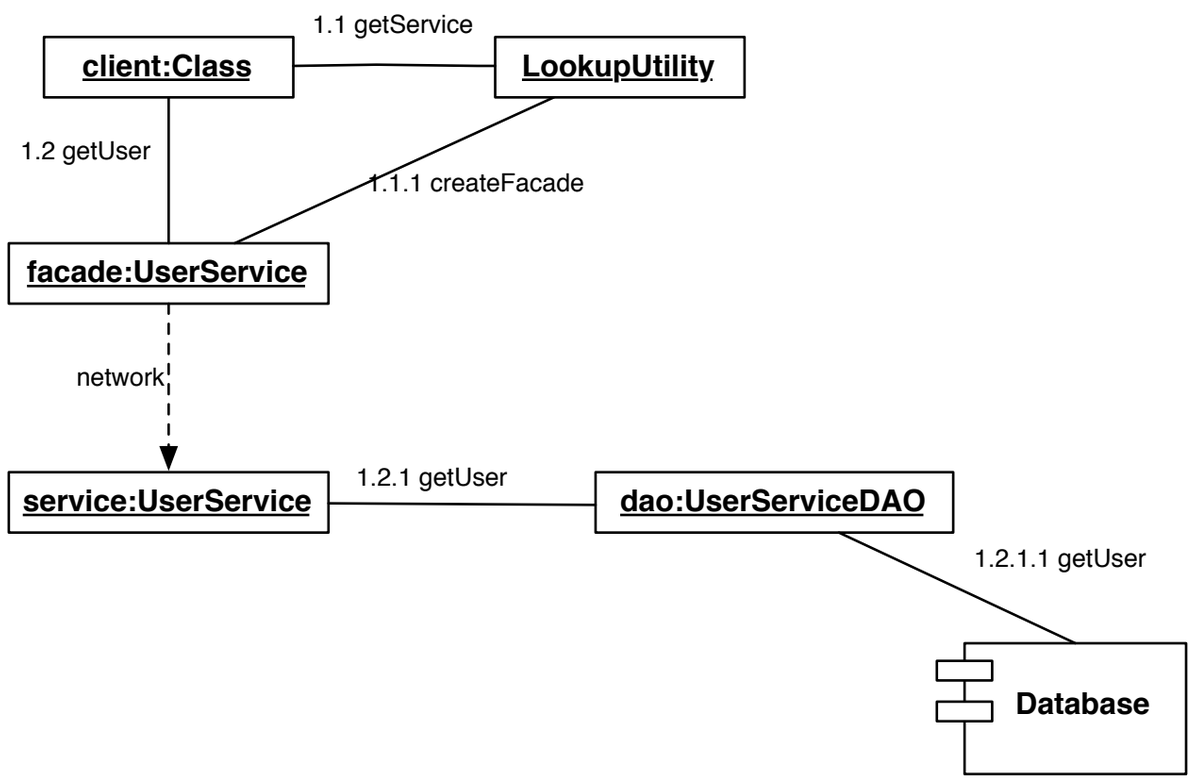
- Problem: Who creates new data?
- Do-what-I-did.
- Separate data segments.
- Proxy - Service.

Example

- Assume we have a User Object we need upon each request at least once, but up to several hundreds (mailbox, favorite lists etc), with assumed mid value of 20.
- Assume we have an incoming traffic of 1000 requests per second.

Naive approach

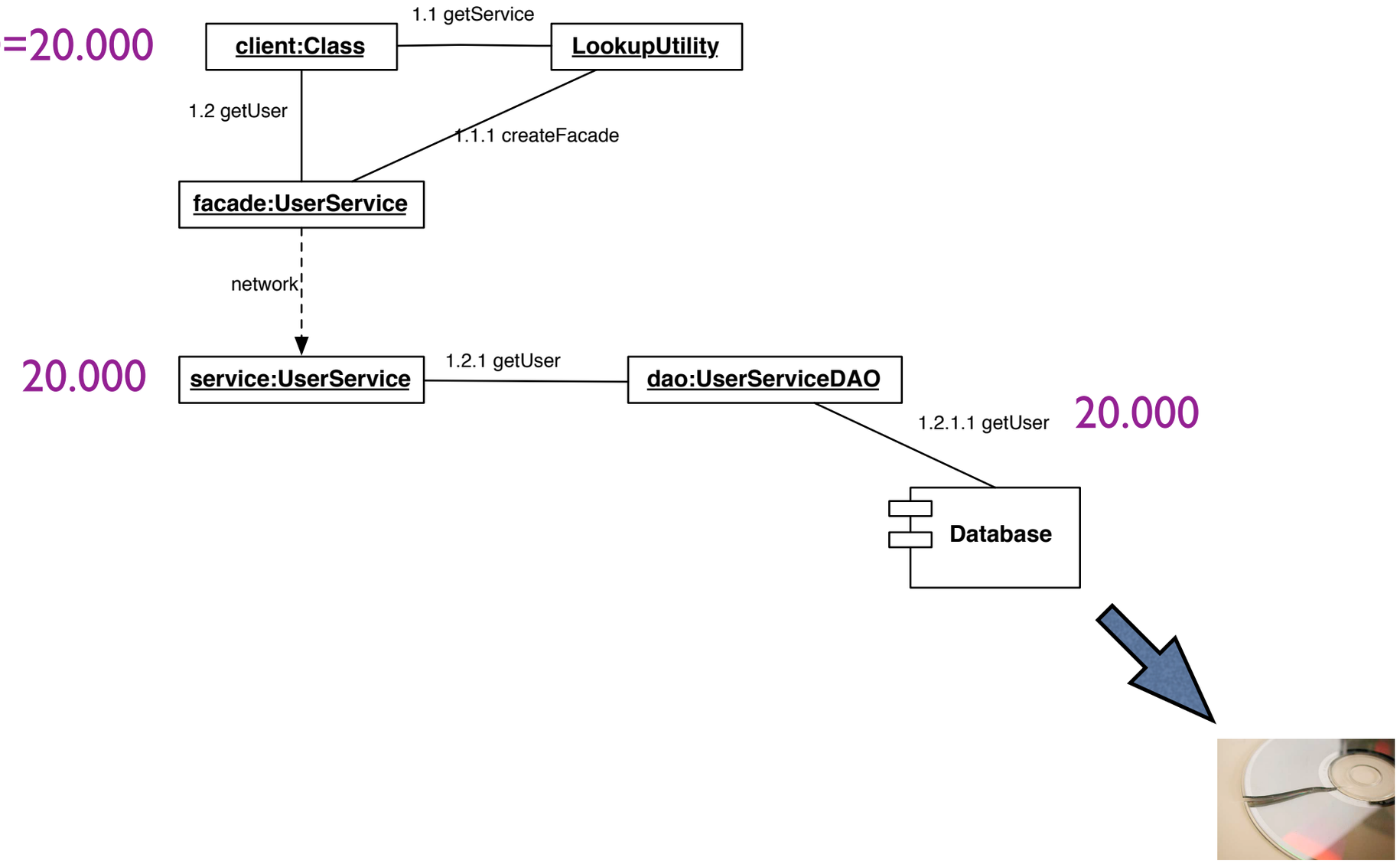



client:Class
1.1 getService
LookupUtility
1.2 getUser
1.1.1 createFacade
facade:UserService
network
service:UserService
1.2.1 getUser
dao:UserServiceDAO
1.2.1.1 getUser
Database

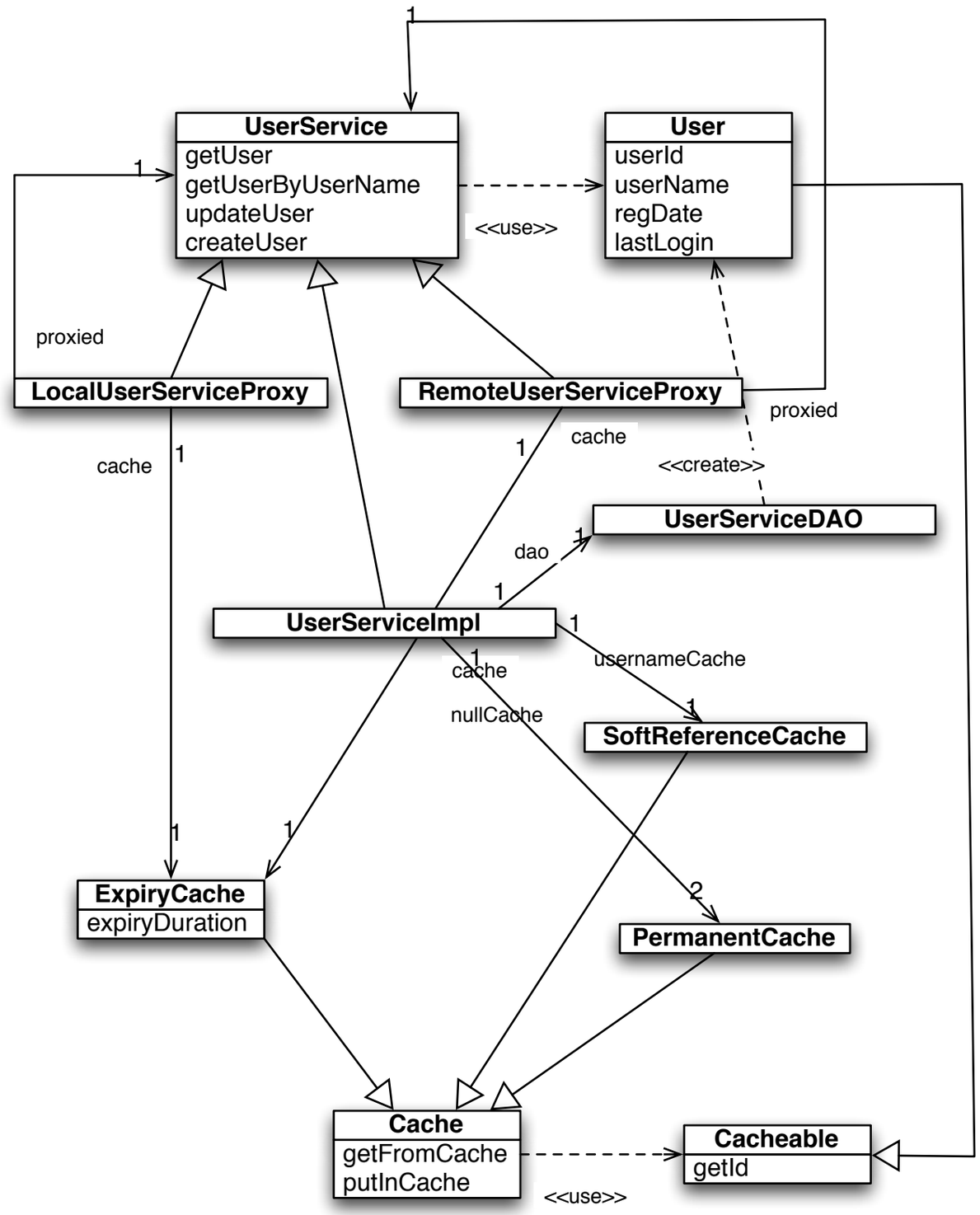
Naive approach

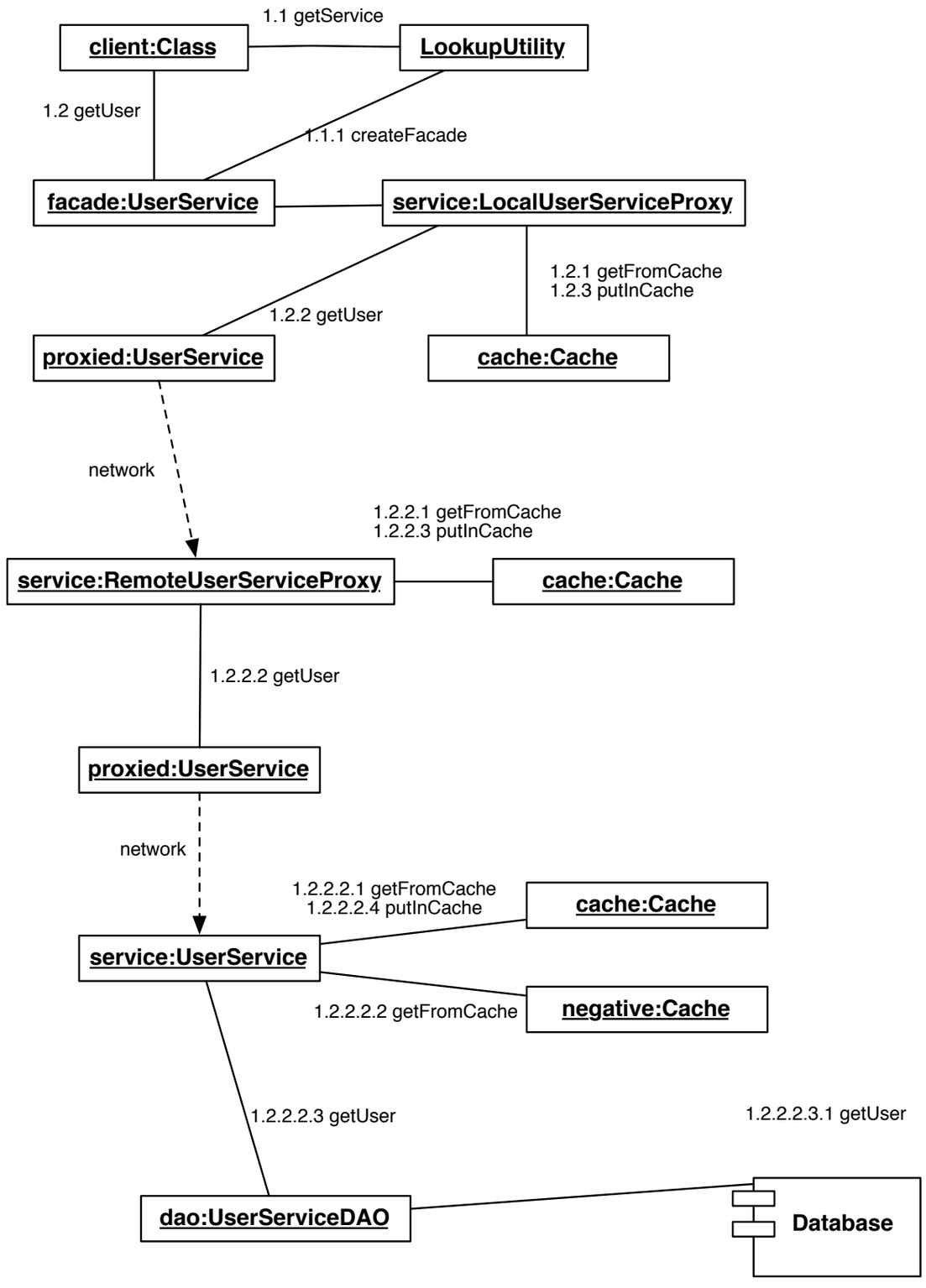
- The DB will have to handle 20.000 requests per second.
- Average response time must be 0,05 milliseconds.
- … Tricky …

1000*20=20.000



Some optimization




client:Class
LookupUtility
1.1 getService
1.2 getUser
1.1.1 createFacade
facade:UserService
service:LocalUserServiceProxy
1.2.1 getFromCache
1.2.3 putInCache
1.2.2 getUser
proxied:UserService
cache:Cache
network
1.2.2.1 getFromCache
1.2.2.3 putInCache
service:RemoteUserServiceProxy
cache:Cache
1.2.2.2 getUser
proxied:UserService
network
1.2.2.2.1 getFromCache
1.2.2.2.4 putInCache
cache:Cache
service:UserService
1.2.2.2.2 getFromCache
negative:Cache
1.2.2.2.3 getUser
1.2.2.2.3.1 getUser
dao:UserServiceDAO
Database

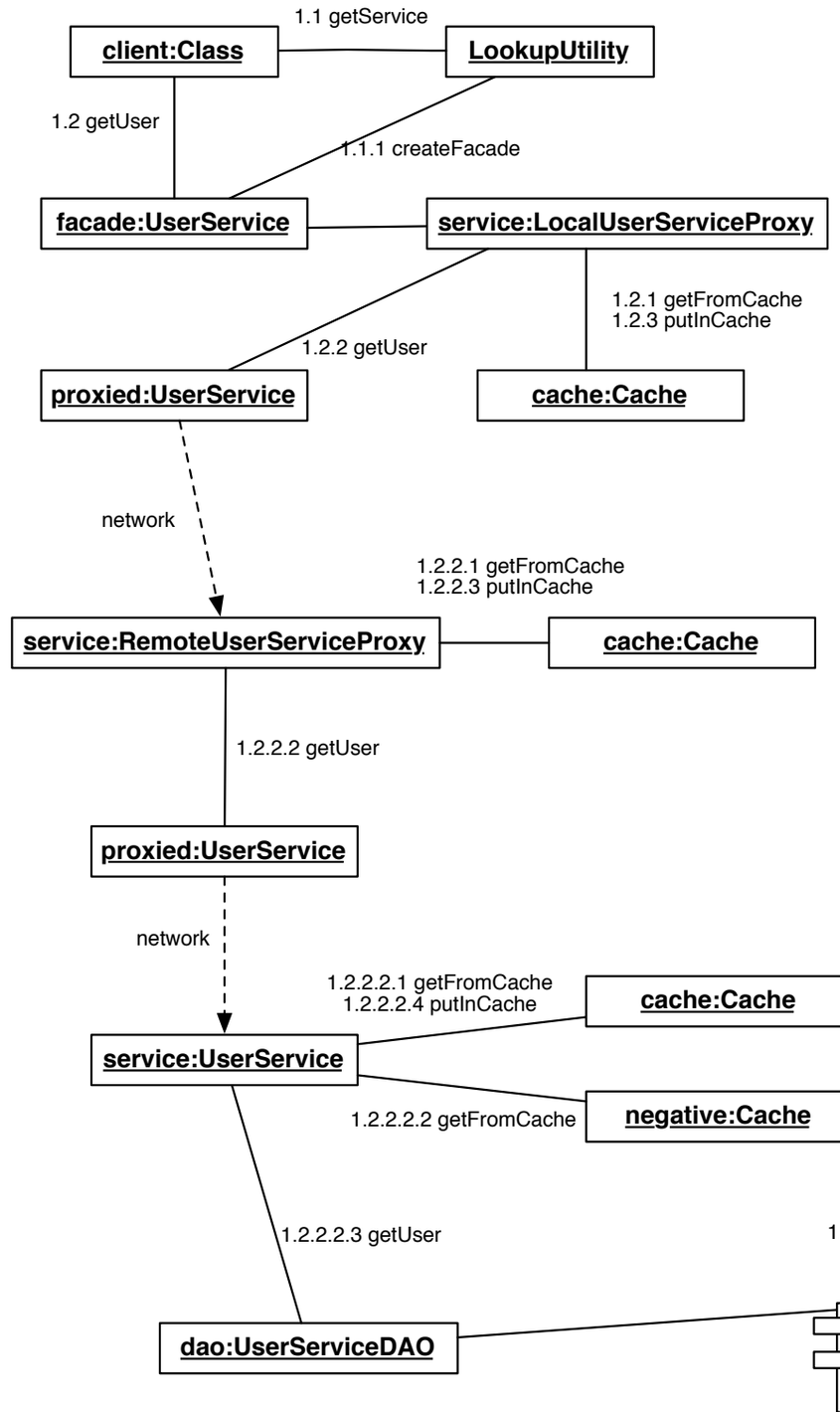
Optimized approach

- LocalServiceProxy can handle approx. 20% of the requests.
- With Mod 5, 5 Instances of RemoteServiceProxy will handle 16000/s requests or 3200/s each. They will cache away 90% of the requests.
- 1600 remaining requests per second will arrive at the UserService.

Optimized approach (II)

- Permanent cache of the user service will be able to cache away 98% of the requests.
- NullUser Cache will cache away 1% of the original requests.
- Max 16 Requests per second will reach to the DB, demanding a response time of 62,5ms --> Piece of cake.
  And no changes in client code at all!

1000*20=20.000



4000 stop here

14400 stop here
in different instances

1568 stop here

16 stop here

16 make it to DB

Partytime !

Monitoring (APM)

- Who needs it anyway?

incoming request

Production

Loadbalancer (pair)

Static pool

Pix pool

guest pool

webgb01 webgb02

member pool

web01 web02 web03 ... web12

ExtAPI pool

Admin pool

Exporter

console

neofonie

omniture

attivio

user data

profile data

usage data

profile data

registry

business logic servers pool

biz01 biz02 biz03 biz04 ... biz09

biz00
hotstandby

Database (pair)

data01 data02

FileSystem Storage

Connector

heidelpay

clickandbuy

parship

attivio

neofonie

payment

payment

profiles

profiles

search

MoSKito Control
0.4-SNAPSHOT

Last refresh: 2013-11-13T17:43:40,655 Next refresh in 49 seconds Mute for 60 minutes Mute Settings

CATEGORY
- All Categories (3)
- biz (1)
- web (2)

WIDGETS
- Status
- TV
- Charts
- History

STATISTICS
- 0
- 0
- 0
- 3

TV

History

Timestamp	Name	Status change
2013-11-13T15:56:05,064	web01.prod	
2013-11-13T15:54:15,014	web02.prod	
2013-11-13T15:39:14,595	web02.prod	
2013-11-13T14:56:03,455	web02.prod	
2013-11-13T14:56:03,411	web01.prod	
2013-11-13T13:56:01,784	web02.prod	


Health?
Performance?
Architecture?

Top 5 things people are doing wrong with Application Performance Management

5

You don’t have any
Application Performance Management.
At all.

4

You measure room temperature to find out if the patient has fever.

3

You have APM, but you only look at it, when the system crashes, and switch it off when its alive.

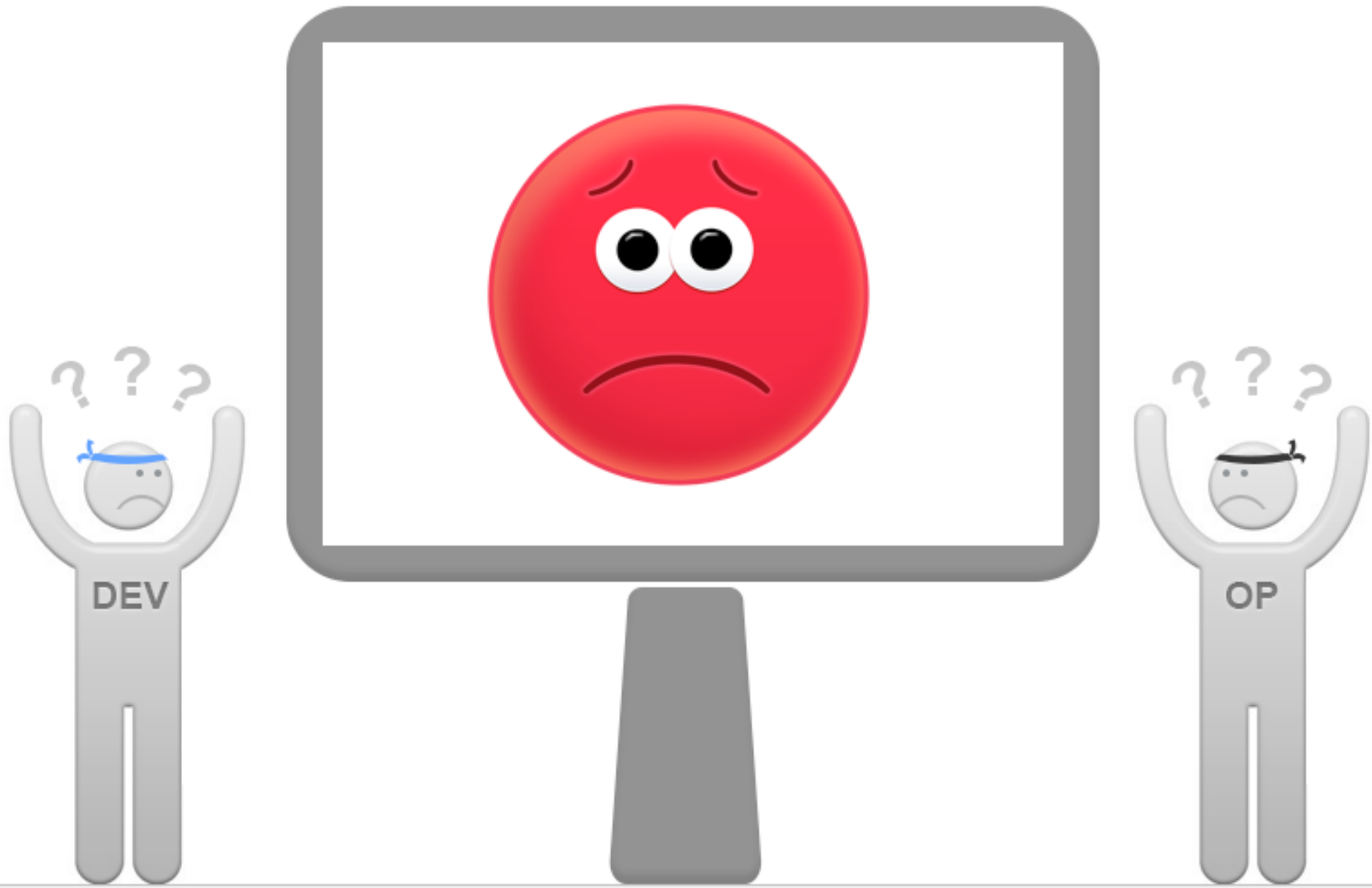
2

You don’t care about business key figures and don’t have any in your APM.

1

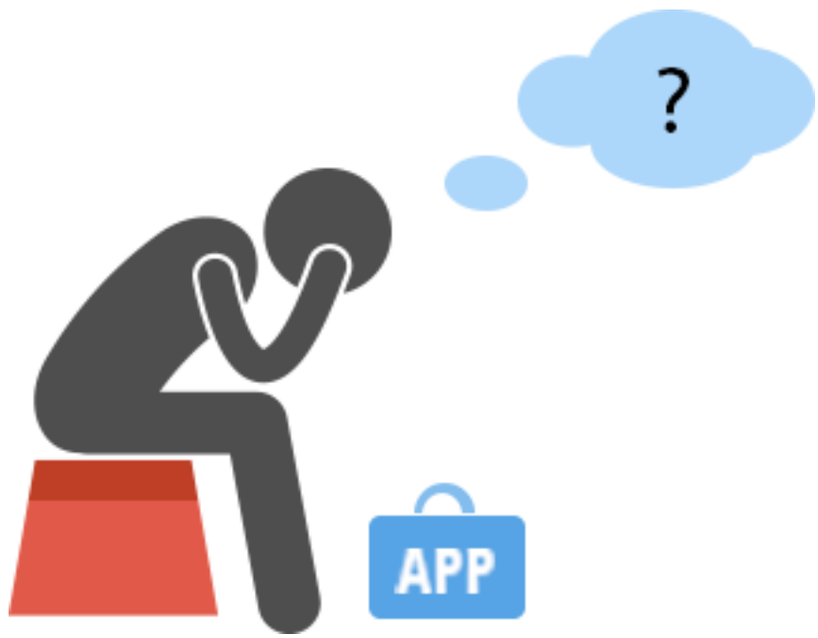
Everyone has it’s own
Application Performance Management.
And no-one speaks to each other.

und wenn es kracht?



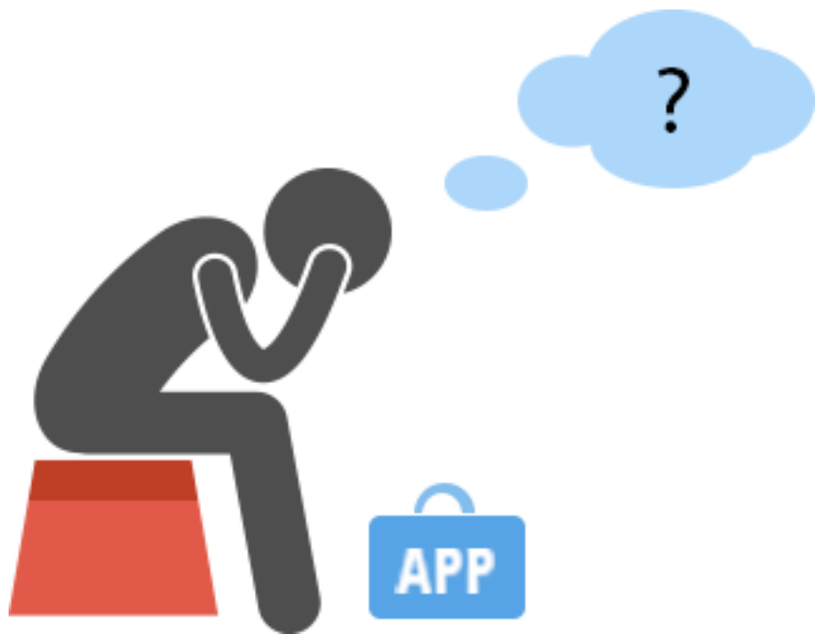
Oliver's First Rule of Concurrency

With enough concurrent requests any condition in code marked with „Can't happen" -
will happen.



Oliver’s Second Rule of Concurrency

After you fixed the „can’t happen“ part, and you are sure, that it „REALLY can’t happen now“ -
It will happen again.



a user will always

- Outsmart you.
- Find THE input data that crashes you.
- Hit F5.

So, what do I do?

- Accept possibility of failure.
- Handle failures fast.
- Minimize the effect.
- Build a chaos monkey!

Thank you

Tech Stack

MoSKito http://www.moskito.org https://github.com/anotheria/moskito

http://www.distributeme.org https://github.com/anotheria/distributeme

http://blog.anotheria.net/msk/the-complete-moskito-integration-guide-step-1/

Human Stack

http://leon-rosenberg.net @dvayanu http://www.speakerdeck.com/dvayanu