Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

# Open Application Standard Platform

## The next generation of Software Engineering

Jörg Hohwiller

Darmstadt, January 15th, 2016

People matter, results count.

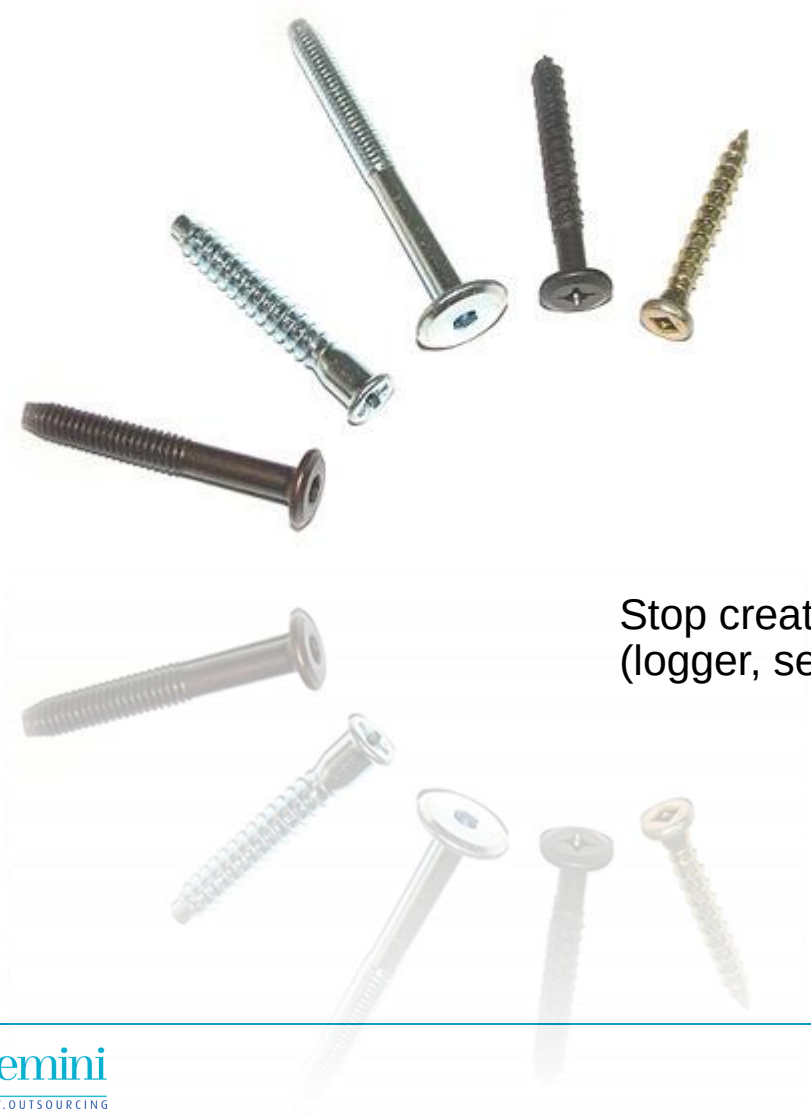
# Agenda

- Motivation (Why OASP?)
- What is OASP?
- Reference Architecture
- Why Open-Source?
- Licensing
- OASP at Universities

# Agenda

- **Motivation (Why OASP?)**
- What is OASP?
- Reference Architecture
- Why Open-Source?
- Licensing
- OASP at Universities

# Use standards and stop reinventing the wheel.

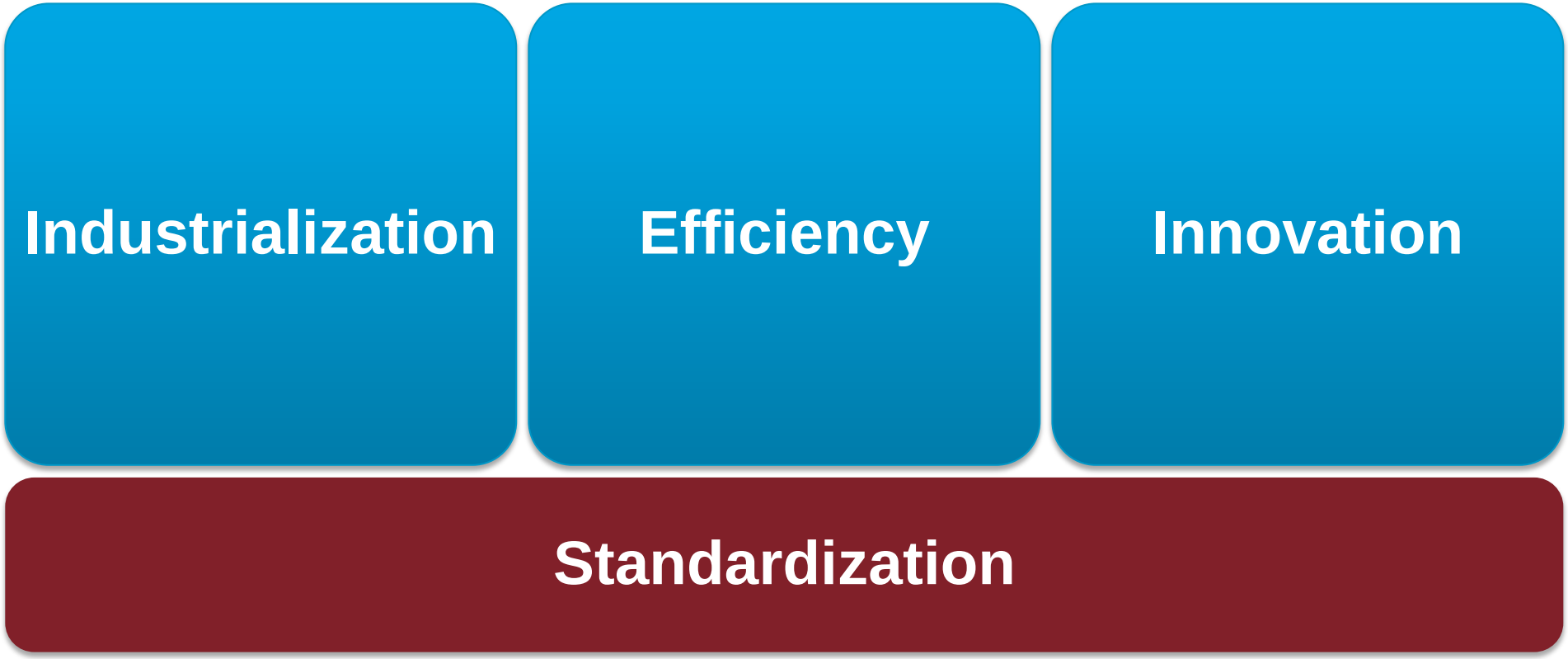


Decisions taking time in our projects

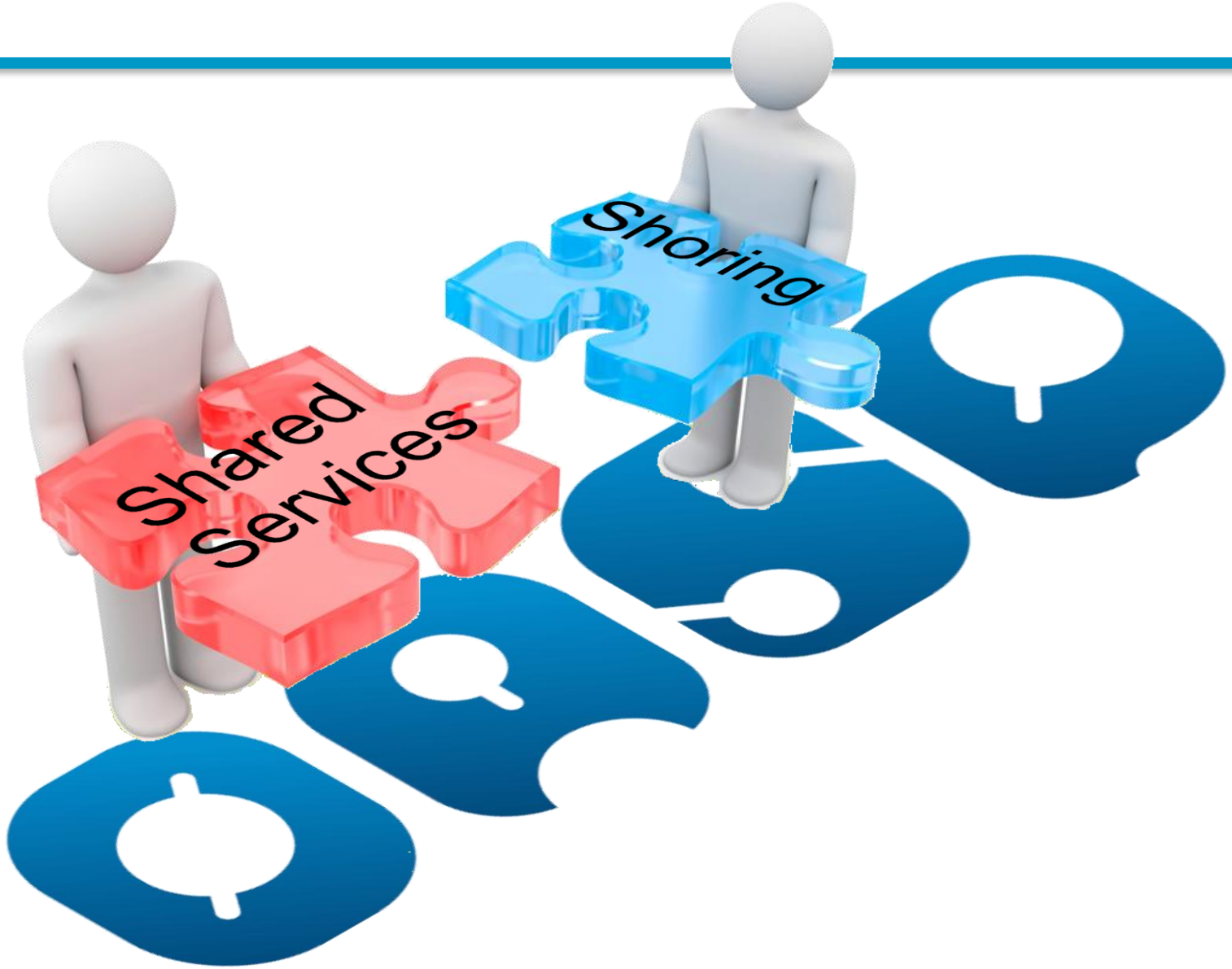
- Technical Architecture
- Technology Stack
- Code Style
- Software Engineering Process
- Building Blocks
- Integration of technical components

Stop creating your own screws
(logger, service-framework, historization, etc.)

# Why do we need Standardization?



# Standardization is the basis for industrialization.



# With OASP we massively increase efficiency.

Reuse of

- Code
- Knowledge
- Documentation
- Tools

# With OASP we can reduce many risks and ensure high quality solutions.

**Approach is proven**

- Architecture, technology-stack, integration, …

**IT-security**

- Safe of vulnerabilities
- App is safe if standard is followed
- Tools can identify violations and vulnerabilities
- Early feedback via IDE integration

**Non functional requirements**

- Can be verified upfront

**Licensing**

- Technology stack is verified by legal department

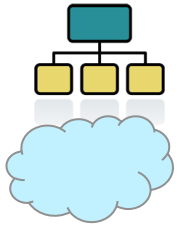
# Agenda

- Motivation (Why OASP?)
- **What is OASP?**
- Reference Architecture
- Why Open-Source?
- Licensing
- OASP at Universities

# OASP is available as Open-Source (ASL2.0) and offers many key features for building modern applications.

**Modular & flexible**

**Multi-Channel & -Platform**

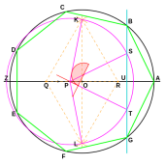


**SOA & Cloud**

**Secure (OWASP Top10 & more)**

**Open and free**

**Optimized tools**



**Pattern-based**

**Simple**

**Solid**


**Consolidated state-of-the-art approaches**

# OASP is divided by Technology and Client/Server.

| | OASP4J | OASP4JS | OASP4NET |
|---|---|---|---|
| **Client** | ▪ Currently not addressed<br>▪ JavaFx planned | ▪ AngularJS<br>▪ Best-Practices<br>▪ Modules<br>▪ Sample Application<br>▪ Application Template | ▪ Not planned |
| **Server** | ▪ JEE-Standards<br>▪ Best-Practices<br>▪ Modules<br>▪ Sample Application<br>▪ Application Template | ▪ Not planned (node.js) | ▪ In Progress |

# OASP provides reusable modules for typical projects

| | |
|---|---|
| oasp4j-basic | Basic Infrastructure |
| oasp4j-batch | Extensions to JSR352/spring-batch |
| oasp4j-beanmapping | Minimal shim for Dozer (or orika) |
| oasp4j-configuration | Constants + helper for configuration |
| oasp4j-jpa-envers | Historization/Auditing Support |
| oasp4j-jpa | DAO base-classes for CRUD (for JPA) |
| oasp4j-logging | Small enhancements for logback |
| oasp4j-rest | Helpers for REST and JSON |
| oasp4j-security | Extension for spring-security |
| oasp4j-test | Testing-Infrastructure (based on JUnit) |
| oasp4j-web | Helper for Servlets und Filters |
| pom.xml | |

JavaScript

| | |
|---|---|
| oasp-i18n | Internationalization |
| oasp-security | CSRF-Token, (re-)login, etc. |
| oasp-ui | Extensions to AngularUI |
| oasp-validation | Validation Support (based on valdr) |
| oasp.less | |
| oasp.module.js | |

# Framework vs. Patterns

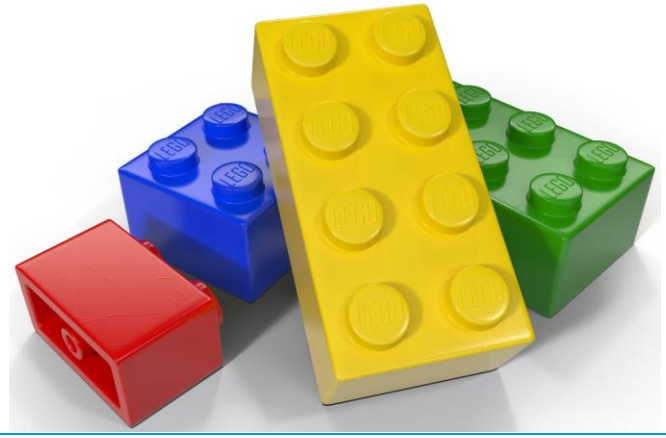
***OASP is no framework but follows a flexible pattern-based approach***

**Framework-based Approach**

- Often many dependencies on framework
- Some frameworks dominate your programming (Interfaces to implement, Classes to extend, Control-Flow, etc.)
- Powerful if your product follows framework-design
- Otherwise you might get trapped in the framework and you are programming „against“ the framework

**Pattern-based Approach**

- Patterns tell you how to solve a problem
- Compatible patterns lead to components composable like lego bricks
- Boilerplate-Code can also be generated to gain same efficiency as with framework
- You are free to customize to your needs



# Documentation vs. Code

***OASP has a strong focus on documentation***

**OASP Documentation**

- Maintained as collaborative wiki
- Available also as generated PDF per release
- Structured by layers and topics
- Guide with patterns and examples for each topic
- Available for free (Creative Commons License)

## Architecture Overview

### Layers

- Client Layer (GUI)
- Service Layer
- Logic Layer
- Data Access Layer
- Batch Layer

### Guides

- Logging
- Configuration
- Dependency Injection
- Exception Handling
- Internationalization (I18N)

## Dependency Injection

Dependency injection is one of the most important design patterns and is a key principle to a modular and component based architecture. The Java Standard for dependency injection is javax.inject (JSR330) that we use in combination with JSR250.

There are many frameworks which support this standard including all recent Java EE application servers. We recommend to use Spring (a.k.a. springframework) that we use in our example application. However, the modules we provide typically just rely on JSR330 and can be used with any compliant container.

### Example Bean

Here you can see the implementation of an example bean using JSR330 and JSR250:

```
@Named
public class MyBeanImpl implements MyBean {
  private MyOtherBean myOtherBean;
  @Inject
  public void setMyOtherBean(MyOtherBean myOtherBean) {
    this.myOtherBean = myOtherBean;
  }
  @PostConstruct
  public void init() {
    // initialization if required (otherwise omit this method)
  }
  @PreDestroy
  public void dispose() {
    // shutdown bean, free resources if required (otherwise omit this method)
  }
}
```

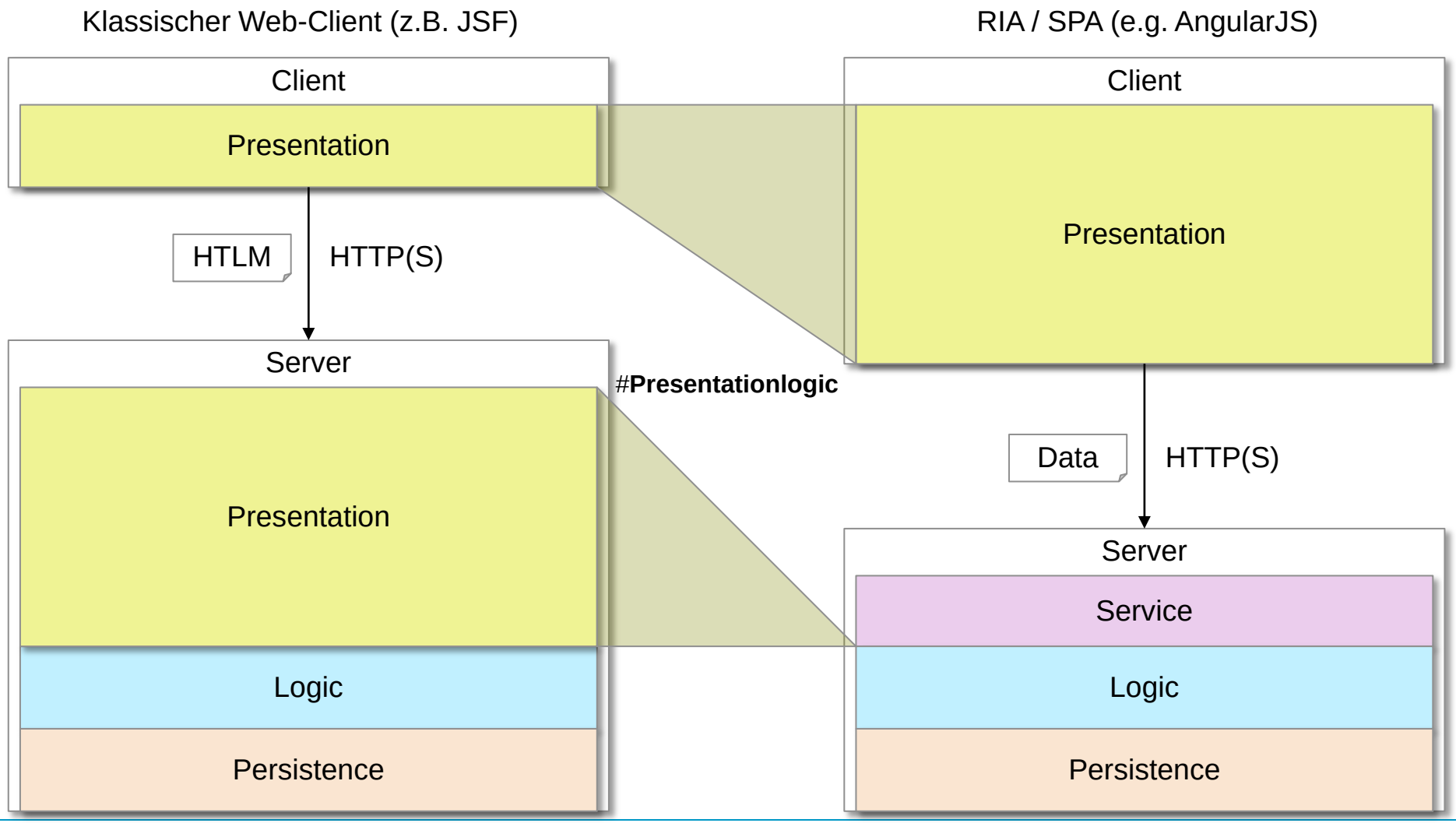
It depends on MyOtherBean that should be the interface of an other component that is injected into the setter because of the @Inject annotation. To make this work there must be exactly one

# Agenda

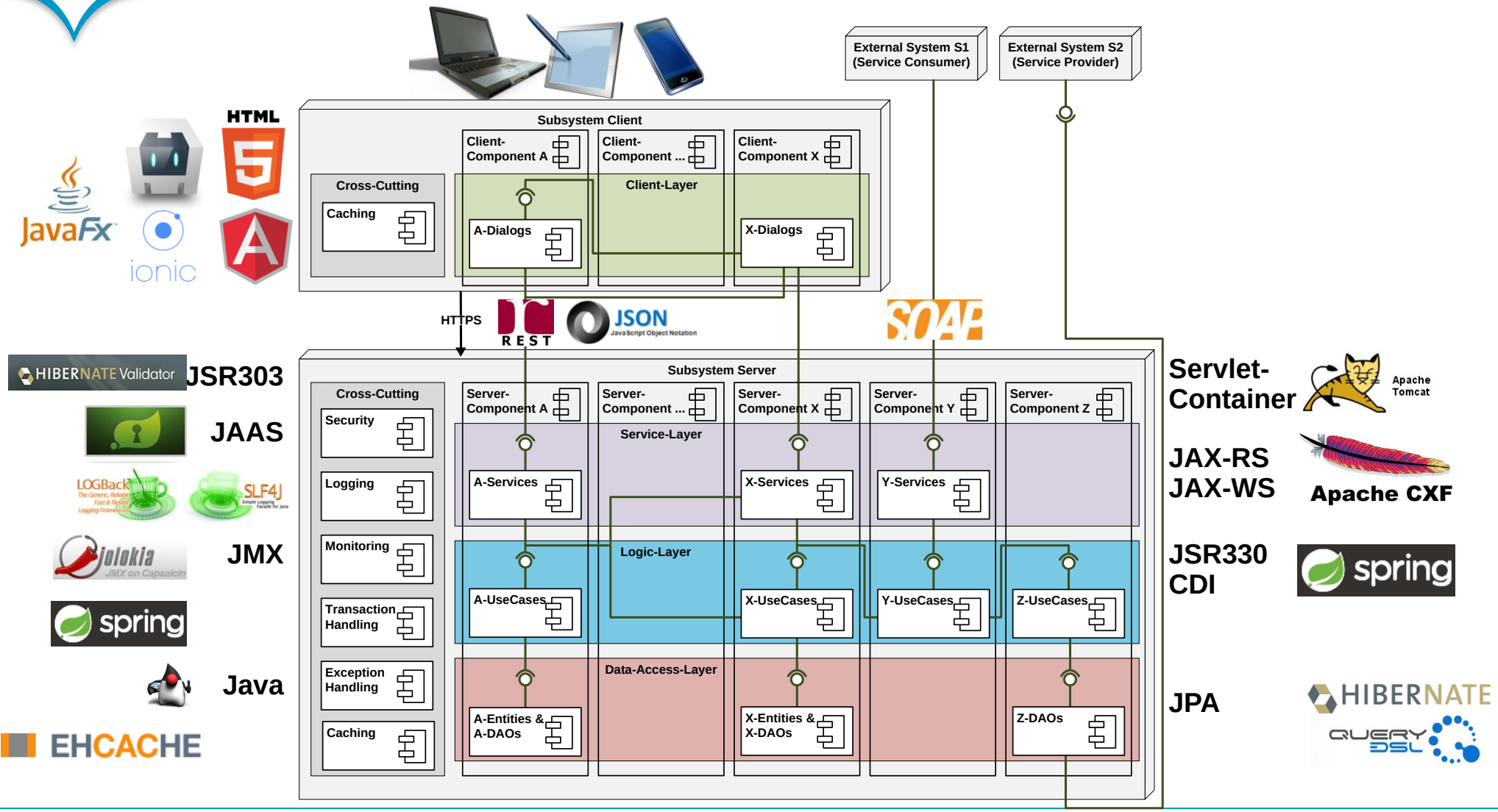
- Motivation (Why OASP?)
- What is OASP?
- **Reference Architecture**
- Why Open-Source?
- Licensing
- OASP at Universities

# From classical web-client to RIA/SPA

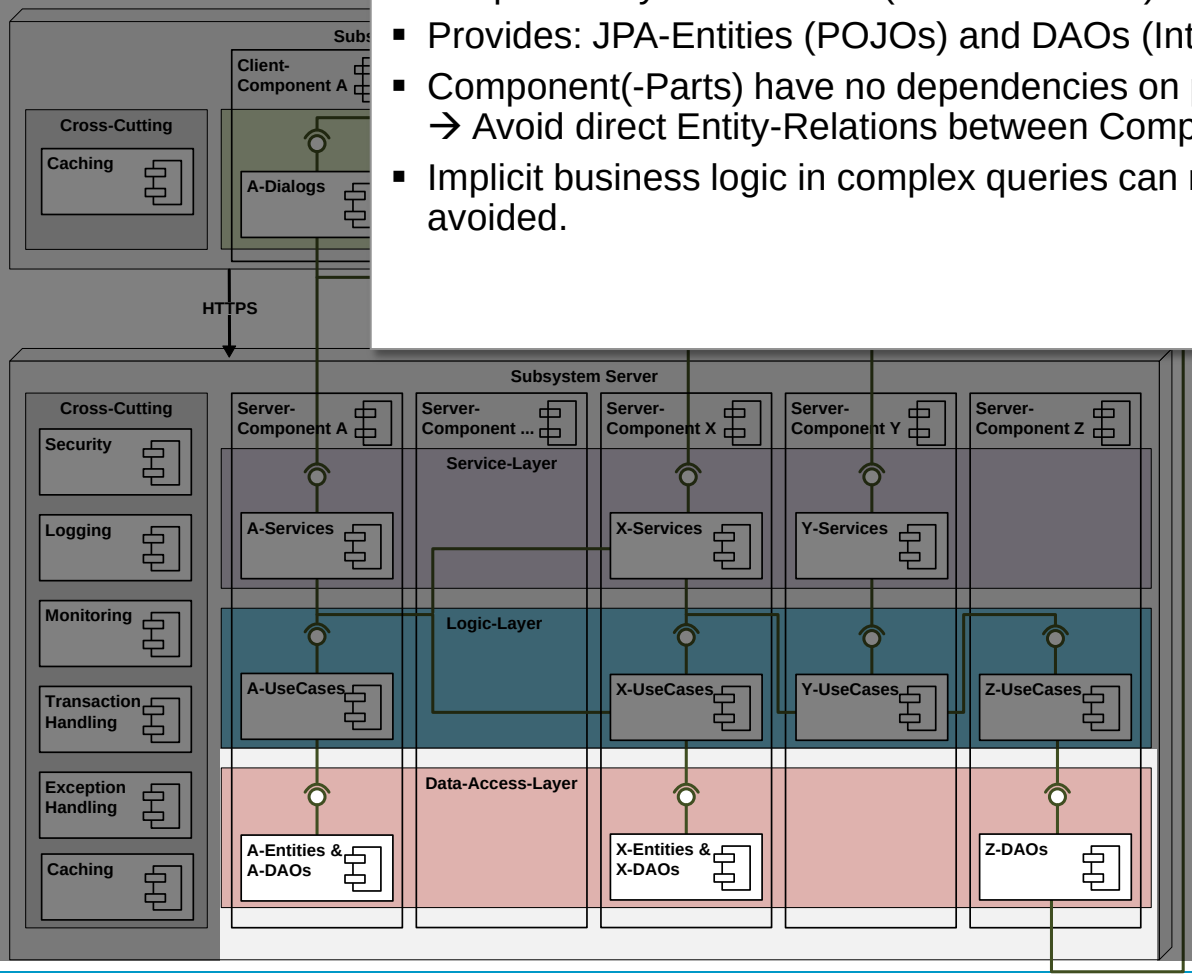
(Rich Internet Application / Single Page Application)



# OASP defines a technical reference architecture that is state-of-the-art and based our profound project experience.



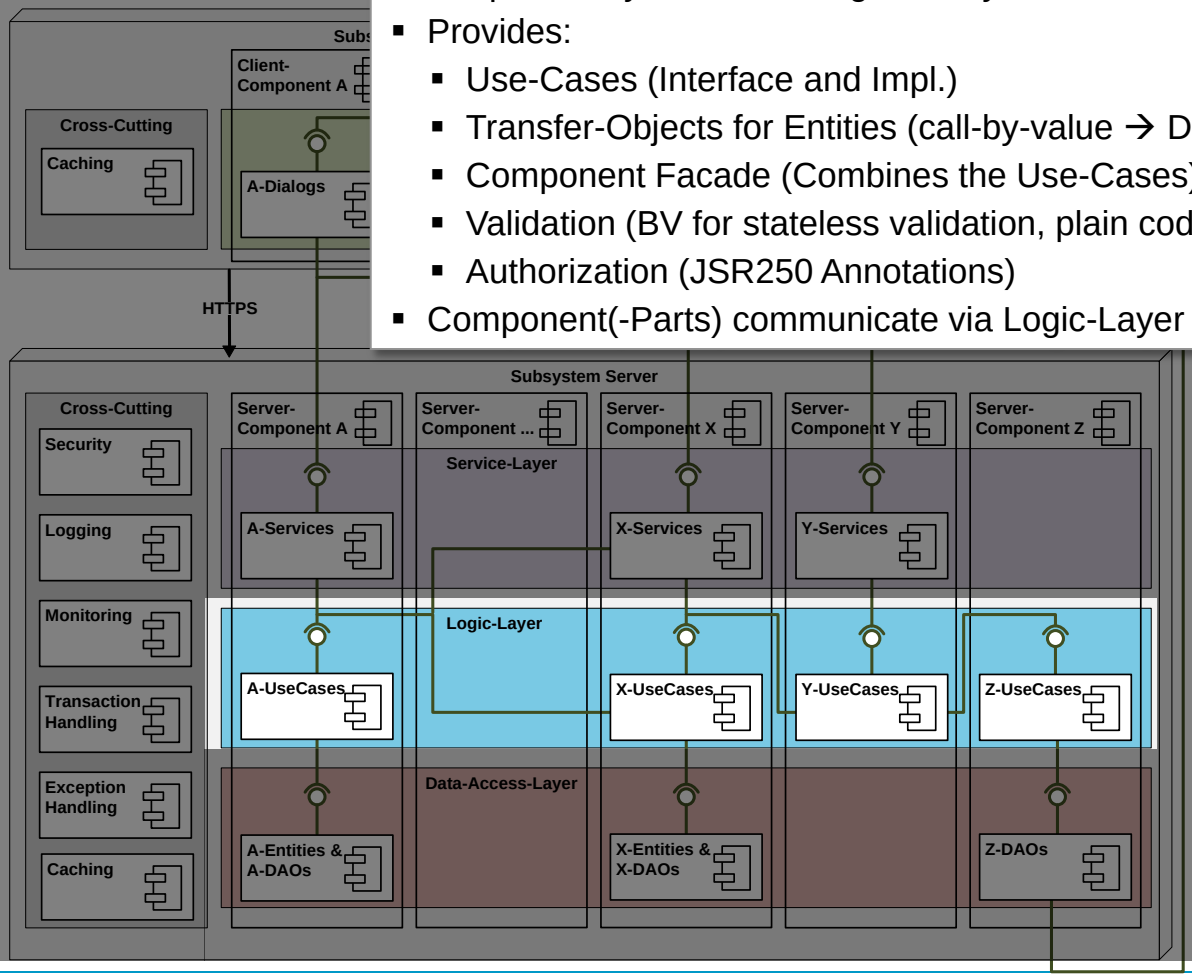
# Data-Access Layer: (JPA-)Entities and DAOs.




## Persistence Layer

- Responsibility: Persistence (CRUD & More.) & ext. access
- Provides: JPA-Entities (POJOs) and DAOs (Interfaces and Impl.)
- Component(-Parts) have no dependencies on persistence layer.
  → Avoid direct Entity-Relations between Components
- Implicit business logic in complex queries can not always be avoided.

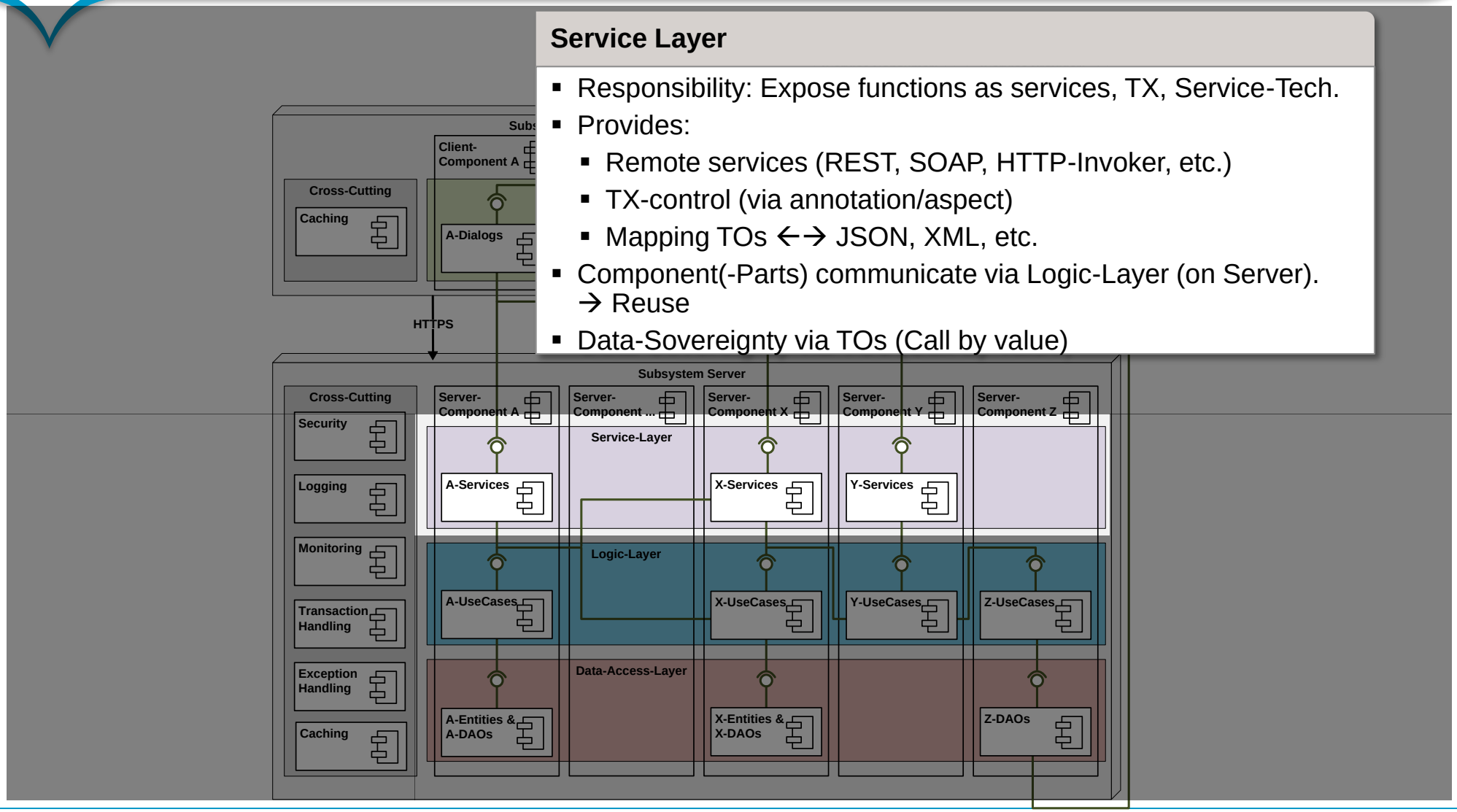
# Logic Layer: Business Logic with Use-Cases, TOs and Component-Interface.




## Logic Layer

- Responsibility: Business Logic, Entity ←→ TO, Authorization
- Provides:
  - Use-Cases (Interface and Impl.)
  - Transfer-Objects for Entities (call-by-value → Data-Sovereignty)
  - Component Facade (Combines the Use-Cases)
  - Validation (BV for stateless validation, plain code otherwise)
  - Authorization (JSR250 Annotations)
- Component(-Parts) communicate via Logic-Layer (on Server)

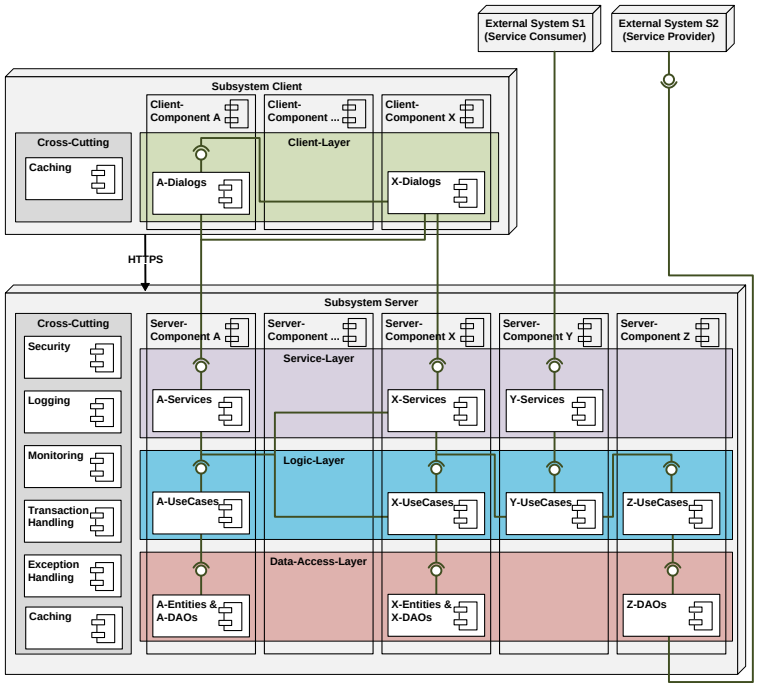
# Service Layer: Expose Functionality as Service to other Consumers.

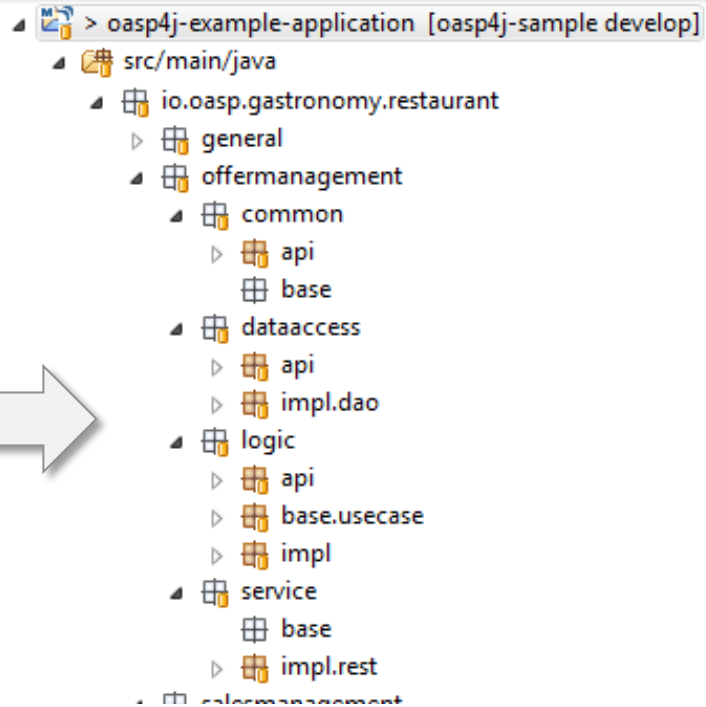



## Service Layer

- Responsibility: Expose functions as services, TX, Service-Tech.
- Provides:
  - Remote services (REST, SOAP, HTTP-Invoker, etc.)
  - TX-control (via annotation/aspect)
  - Mapping TOs ←→ JSON, XML, etc.
- Component(-Parts) communicate via Logic-Layer (on Server). → Reuse
- Data-Sovereignty via TOs (Call by value)

# OASP Guidelines map Architecture to Code.

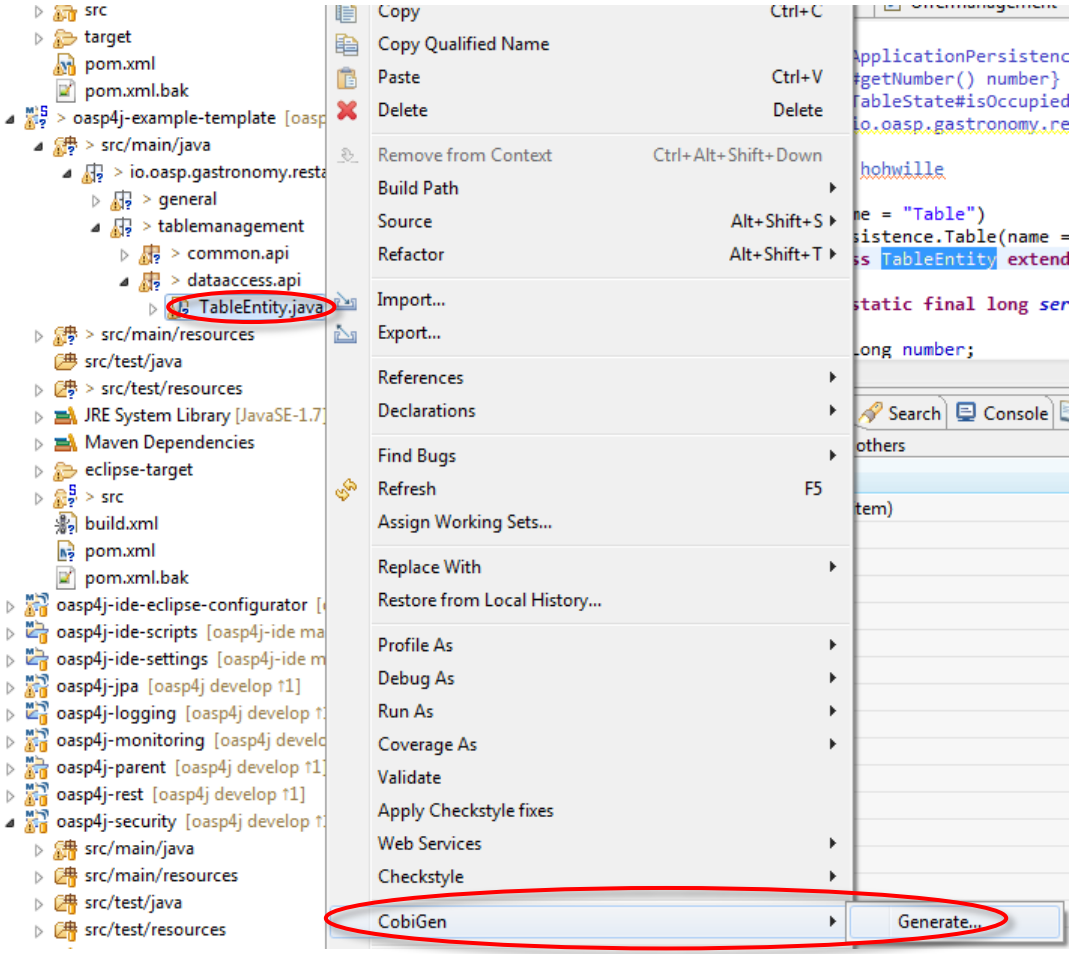


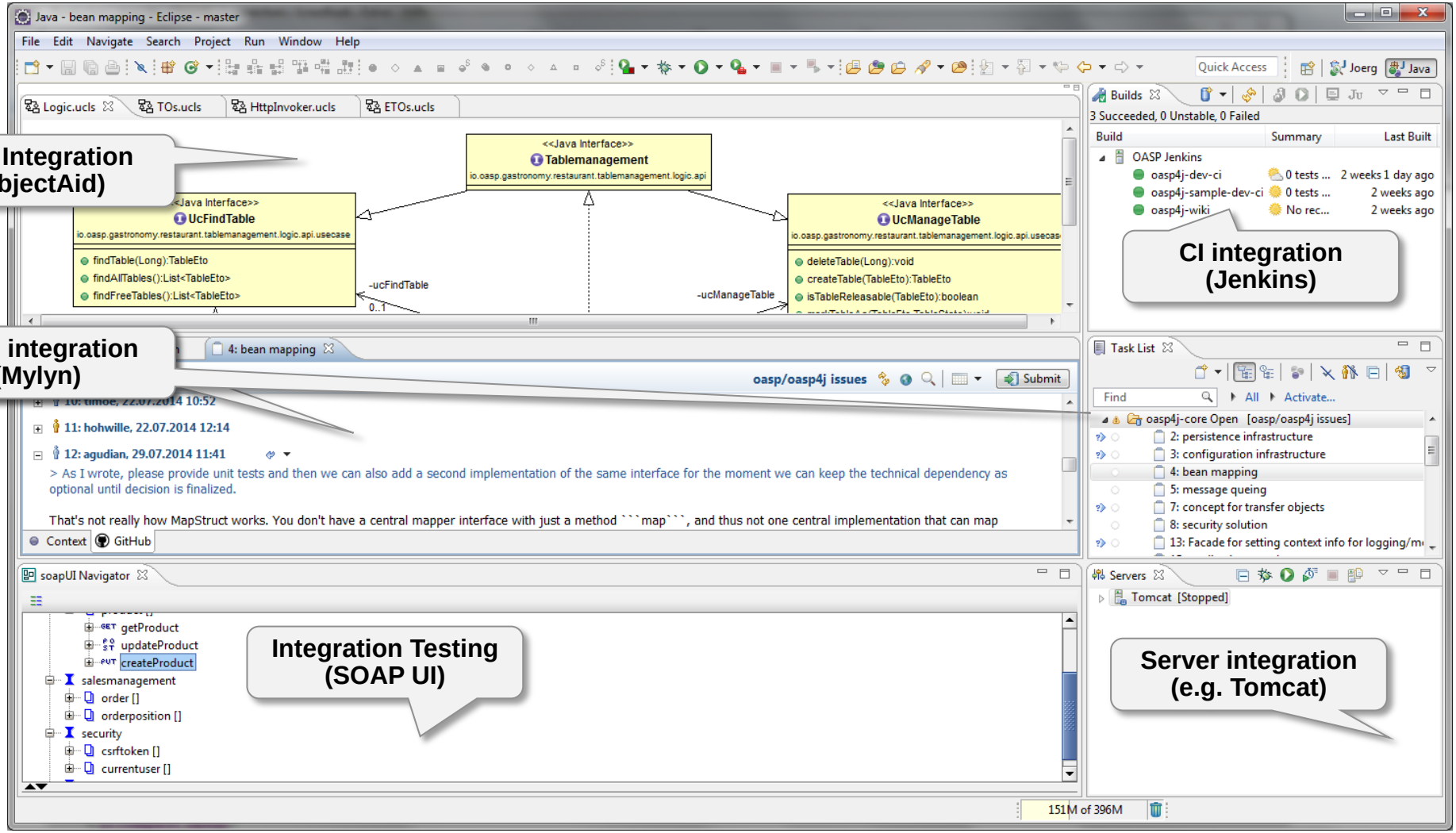

## Benefits

- Brings orientation and structure into the Code
- Tools can “understand” structure and architecture
  - High-level refactoring (work in progress)
  - Architecture validation → early feedback of violations
  - Code-generation → OASP templates ready to use

# OASP provides an incremental code generator with ready-to-use templates based on OASP-standard.

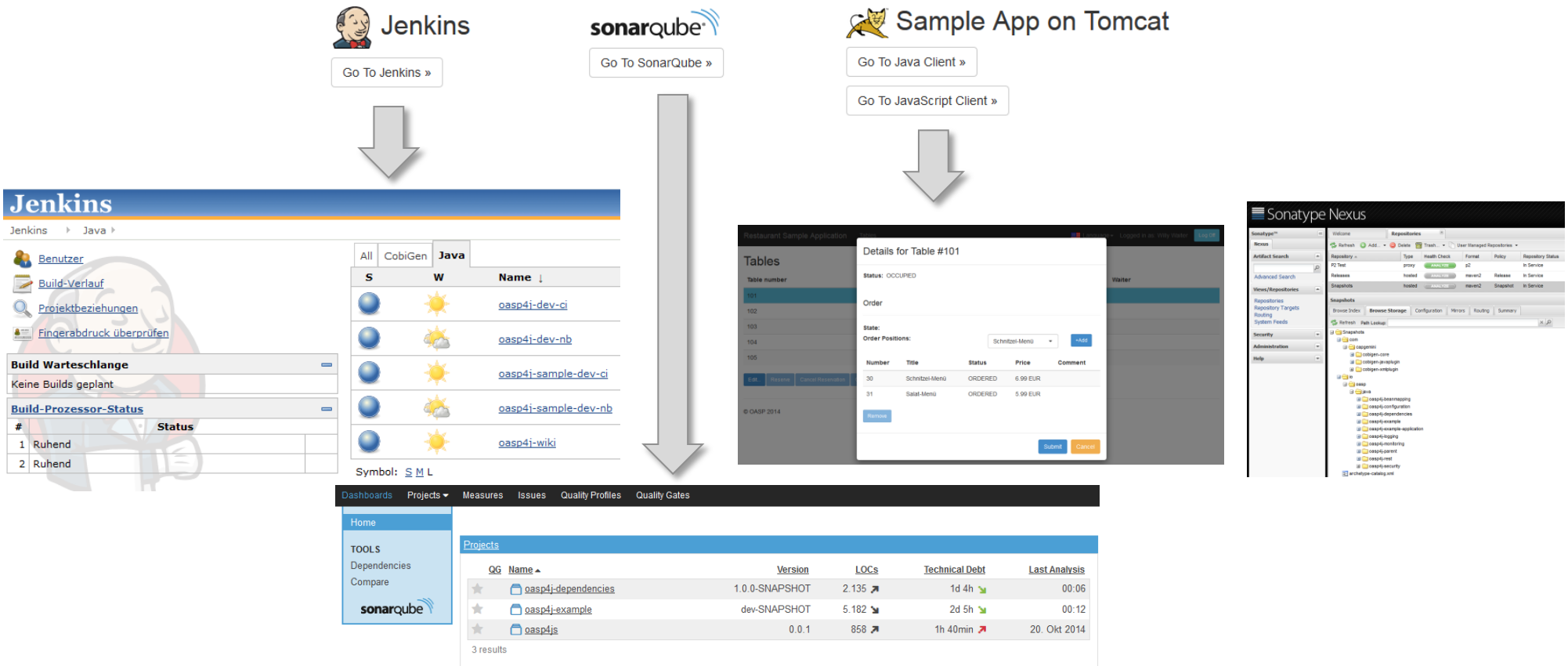



# OASP offers complete IDE solution integrated with all tools.



# OASP is available in the cloud and uses established tools for continuous-integration and quality-measurement.

Open Application Standard Platform - Continuous Integration



# Agenda

- Motivation (Why OASP?)
- What is OASP?
- Reference Architecture
- **Why Open-Source?**
- Licensing
- OASP at Universities

# Closed and framework-based approaches enforce a huge bottleneck effect

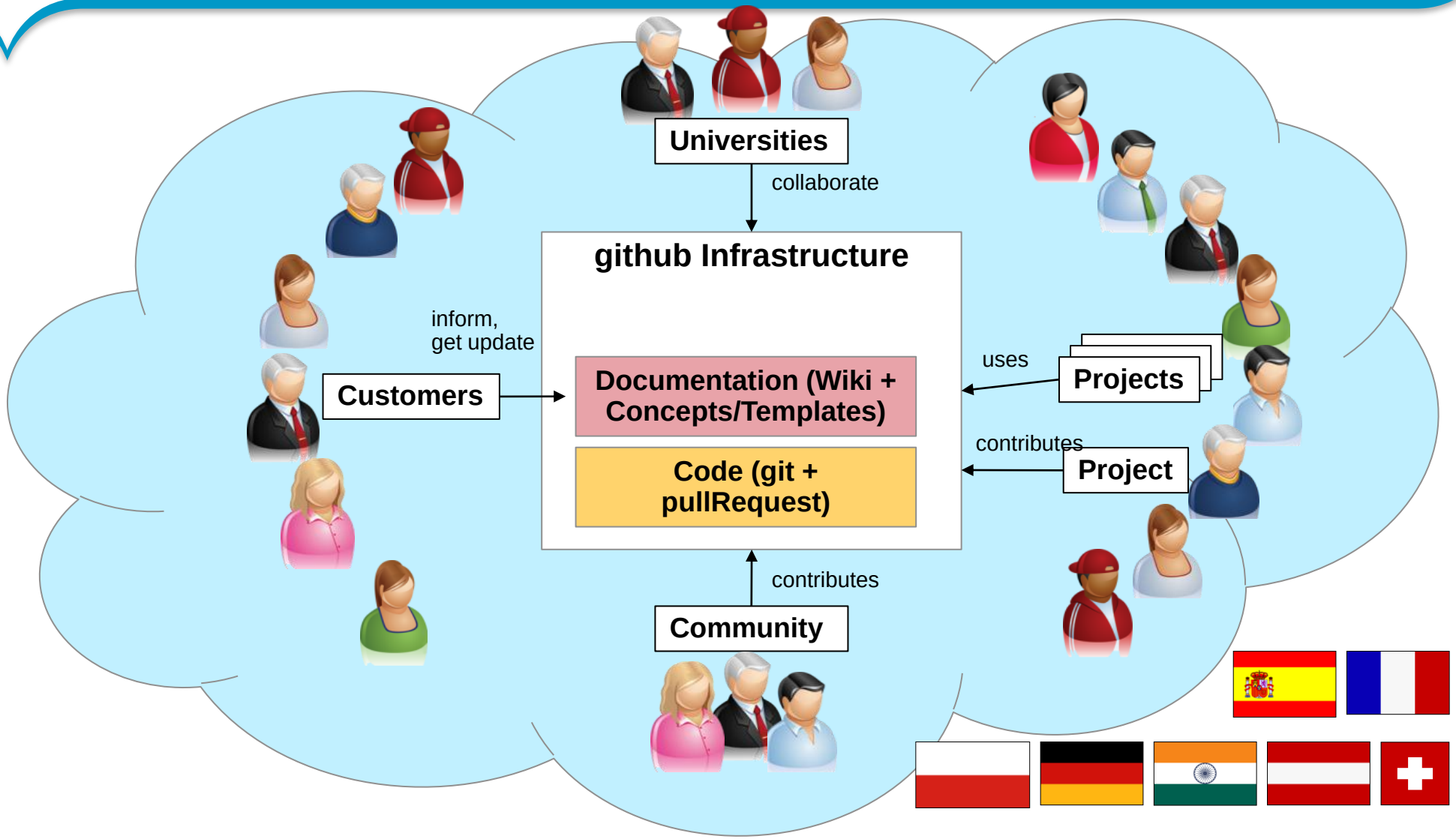
Historical approaches

- Huge frameworks
- High maintenance
- Experts as bottleneck
- Failed to compete with OSS

# What we learned from Open-Source

- Development of our standard shall be collaborative
- Enable users to bring in new ideas and improvements
- Make it grow and get better continuously with our projects (where we make the money)
- Success for collaboration requires great tools

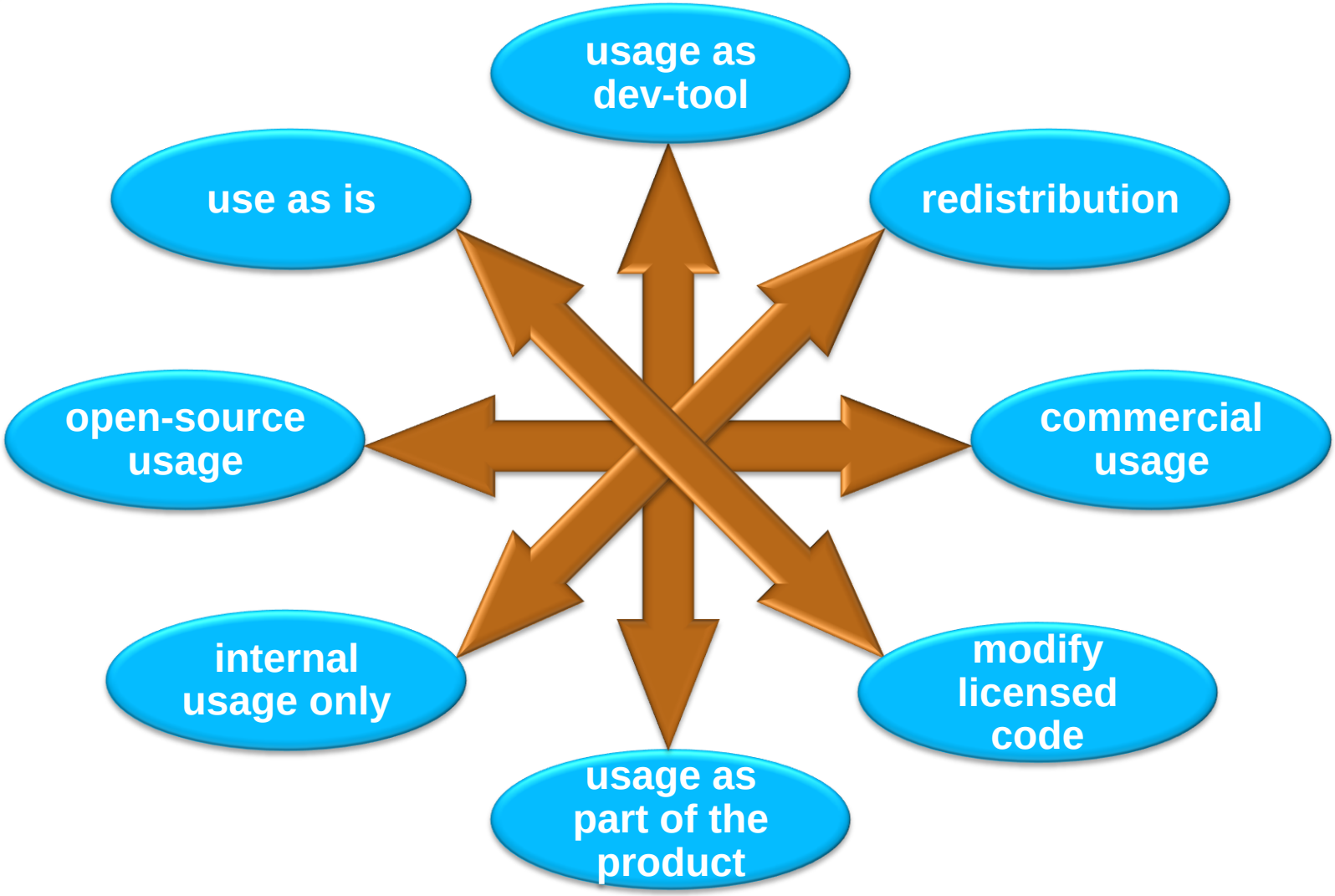
# OASP is based on a community model and uses github for easy development across entire Capgemini Group



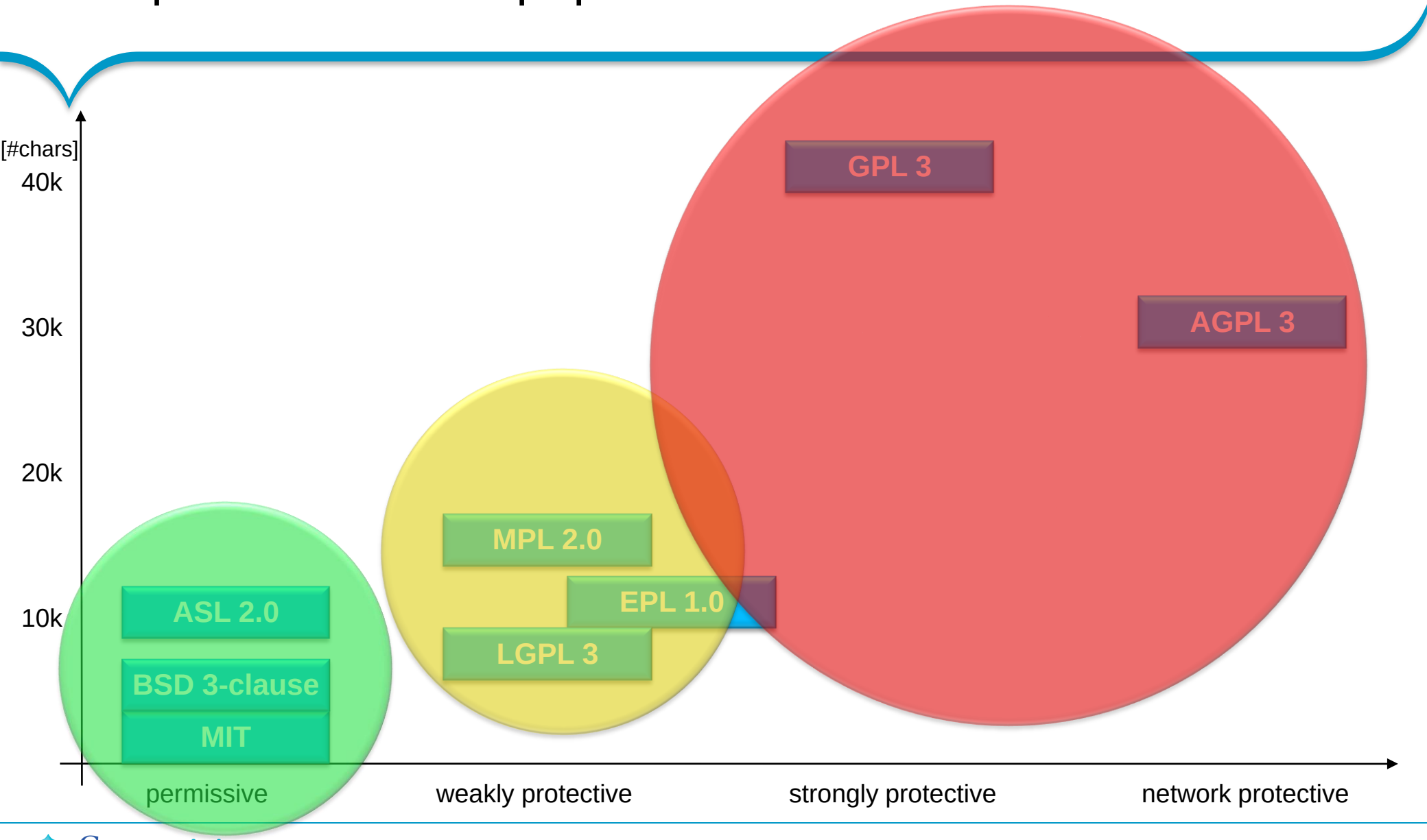
# Agenda

- Motivation (Why OASP?)
- What is OASP?
- Reference Architecture
- Why Open-Source?
- **Licensing**
- OASP at Universities

# The licensing of a software has multiple dimensions.



# Comparison of most popular OSS Licenses



# Aspects of License-Constraints and -Violations

## Constraints

- BSD:
  *"Redistributions ... must reproduce the above copyright notice, this list of conditions and the following disclaimer"*
- GPL:
  *"You may convey a covered work in object code form …, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways: …"*
- LPPL (<=1.2)
  *"You must not distribute the modified file with the filename of the original file."*

## Violations (Headlines)

- 2003: Court imposes SCO to pay a fine of **10 000 €** due to not withdrawing the statement that Linux contains stolen Unix code.
- 2004: Due to GPL violation Sitecom has to withdraw WLAN-Router from market and pay **100 000 €**
- 2013: Due to LGPL violation ZDF-Developer has to pay **15 000 €** to adhoc dataservice GmbH

# License Management is complicated

- Licenses can be incompatible
- Some OSS projects violate licenses (e.g. spring-data-neo4j [ASL2.0] but is derivate work of neo4j driver [GPL3])

Various challenges

- Venue (area of jurisdiction) can matter a lot
- Licenses constraints (e.g. filename clause of LPPL)
- OSS libraries can contain code from other OSS libraries with different license

- Only specialized lawyers can decide on non-standard cases (→IANAL)
- OASP has been verified by lawyer

# Agenda

- Motivation (Why OASP?)
- What is OASP?
- Reference Architecture
- Why Open-Source?
- Licensing
- **OASP at Universities**

# OASP can be used by universities for education and research.

**Education**

- Example on how to build large-scale business information systems
- Example for technology usage
- Example for industry tool-chains

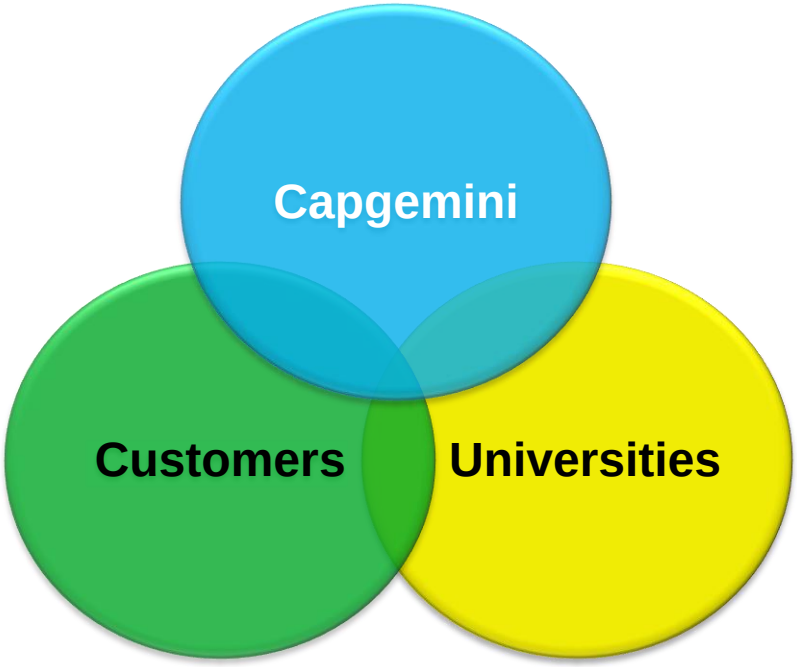
**Research**

- Includes an example application demonstrating the usage of OASP
- Can be used to evaluate software engineering techniques and new approaches in architecture, tooling and engineering processes.
- Open, flexible and easily adoptable

**University projects**

- Working students
- Bachelor- & Master-Thesis (SOA Dependency-Management, Architecture Validation, Security Violation Detection, etc.)
- TU-Darmstadt Projects with Capgemini (Enterprise-Security, Usability/UX)

# OASP is a Win-Win for all



Capgemini

CONSULTING.TECHNOLOGY.OUTSOURCING

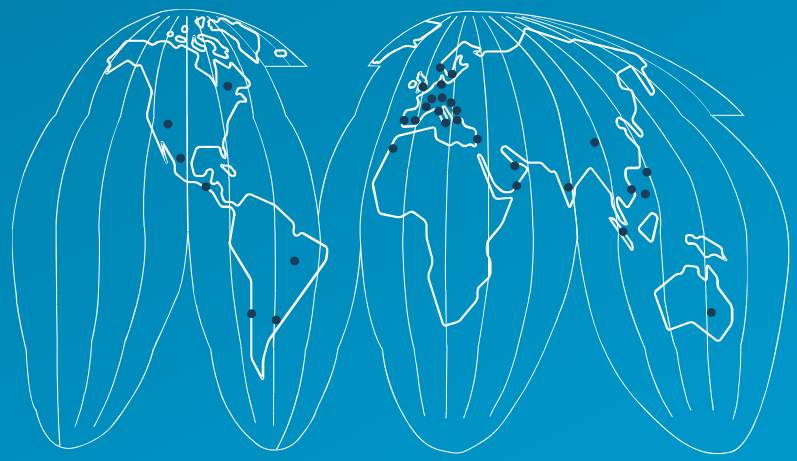
People matter, results count.

## About Capgemini

With more than 130,000 people in 44 countries, Capgemini is one of the world's foremost providers of consulting, technology and outsourcing services. The Group reported 2012 global revenues of EUR 10.3 billion.

Together with its clients, Capgemini creates and delivers business and technology solutions that fit their needs and drive the results they want. A deeply multicultural organization, Capgemini has developed its own way of working, the Collaborative Business Experience™, and draws on Rightshore®, its worldwide delivery model.

*Rightshore® is a trademark belonging to Capgemini*



**www.capgemini.com**

# Research and teaching depend on the exchange of ideas, a concept which is supported by explicit rules:



**Disclaimer**
Even though this presentation, including the cohesive results, has been provided to the best of our knowledge and as subject to diligent review, Capgemini and the authors disclaim all liability for its use.

**Please direct your inquiries to:**
Capgemini
Jörg Hohwiller
Berliner Straße 76, 63065 Offenbach, Germany
069 82901 261
joerg.hohwiller@capgemini.com