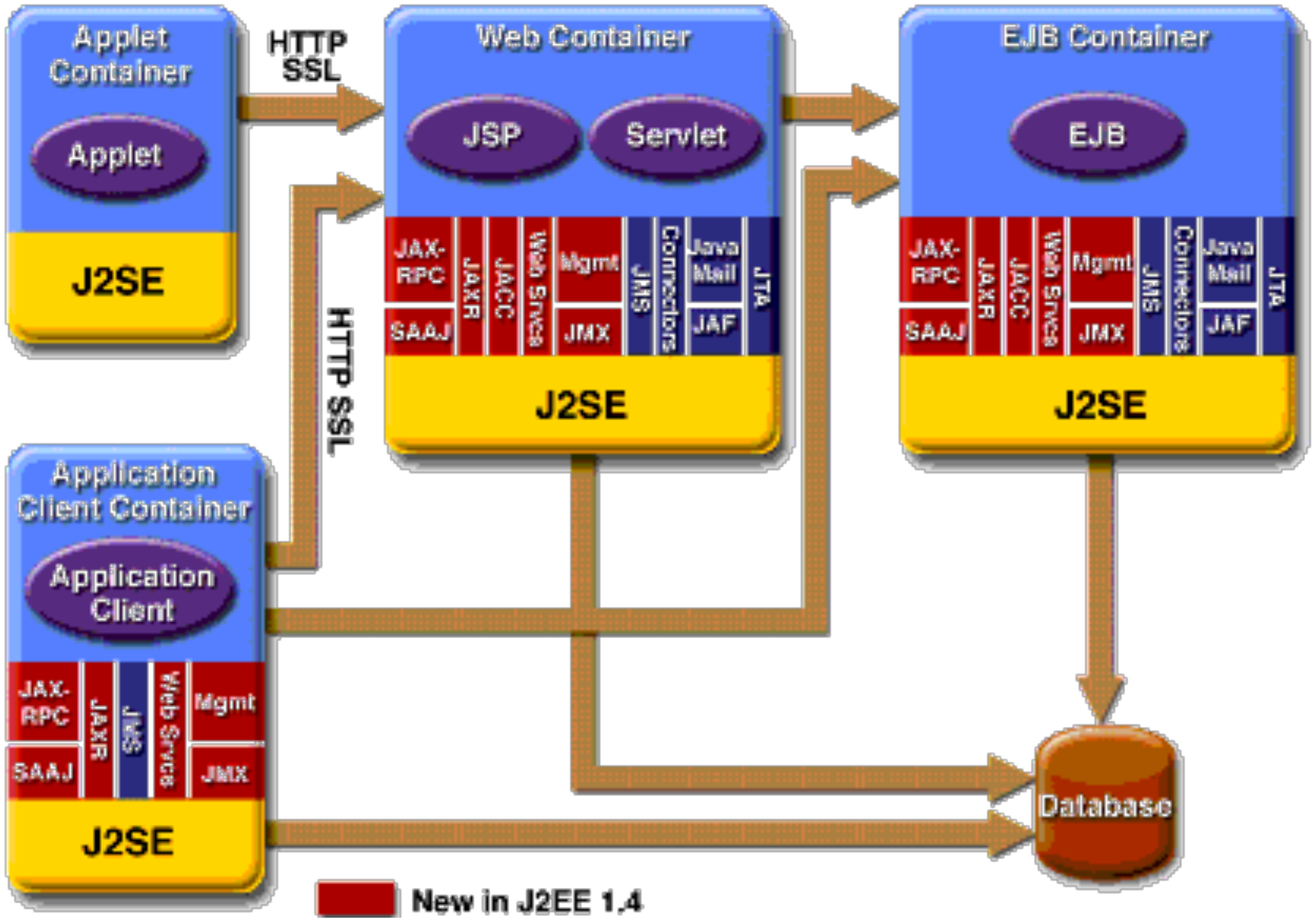
#Architecture201x

Stefan Tilkov | innoQ

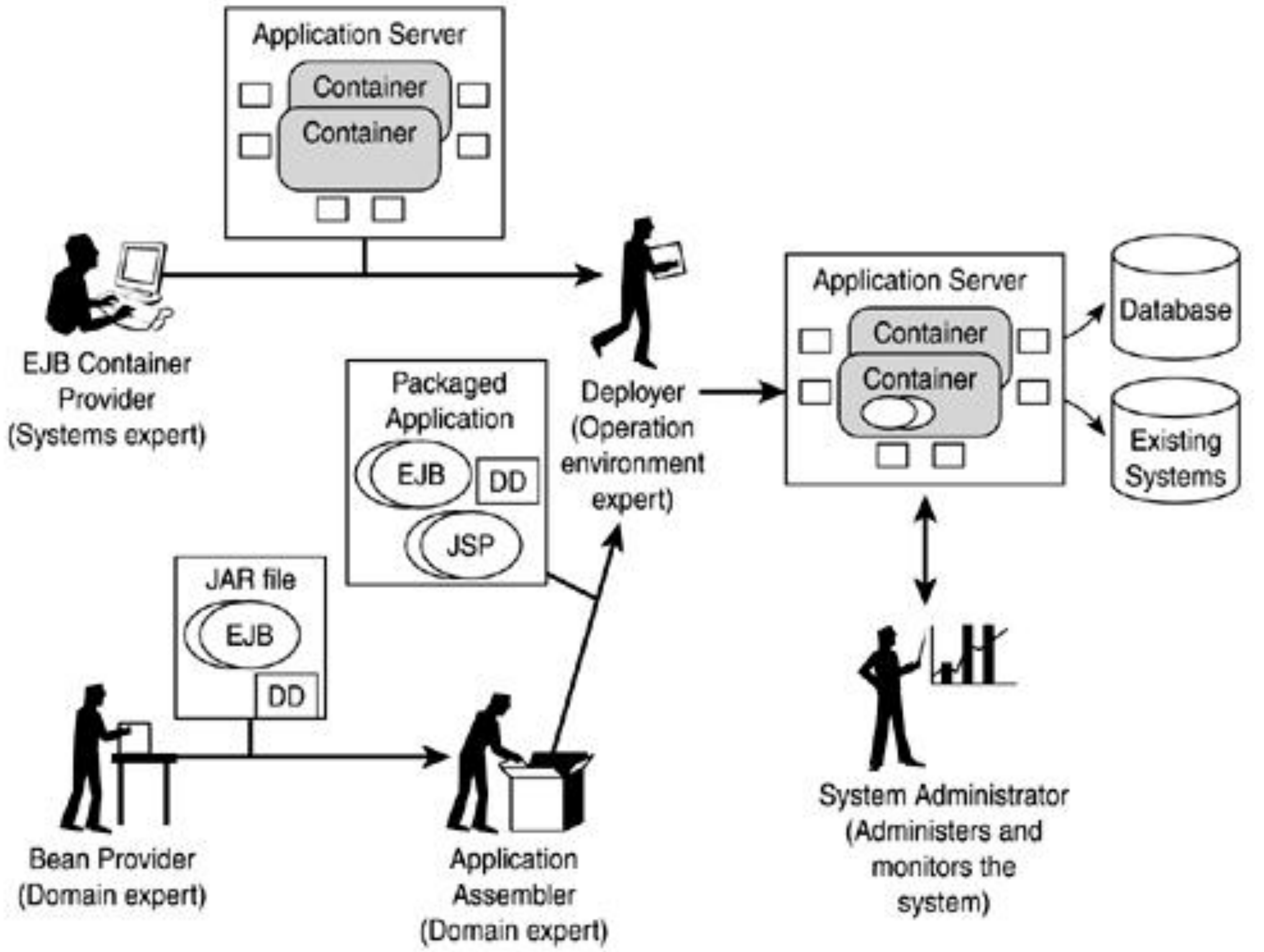
stefan.tilkov@innoq.com

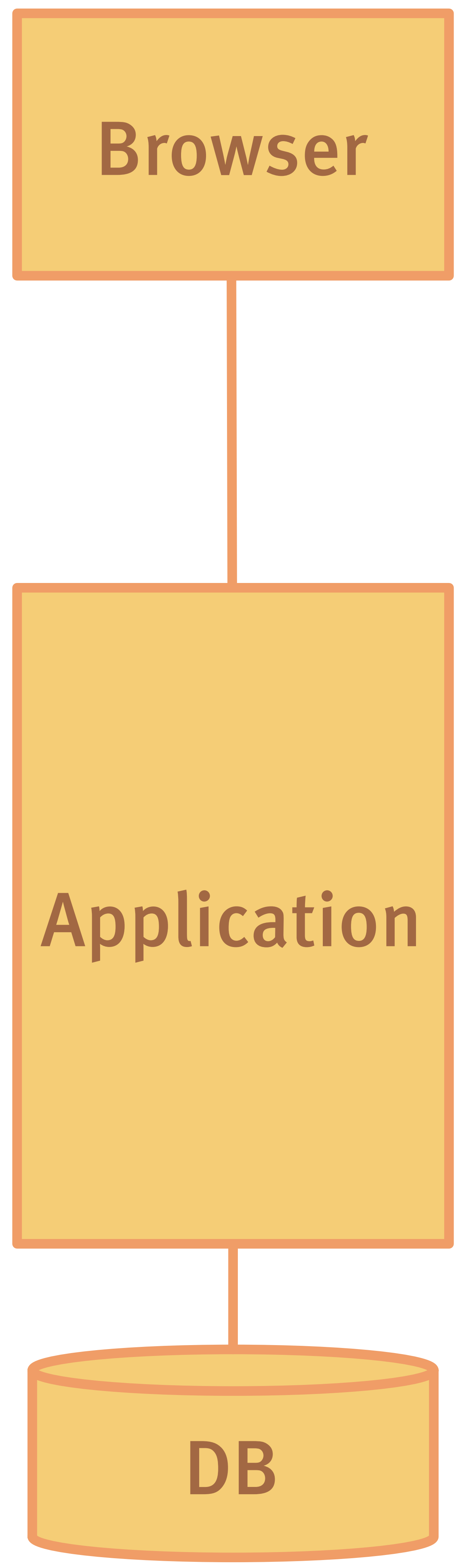
@stilkov

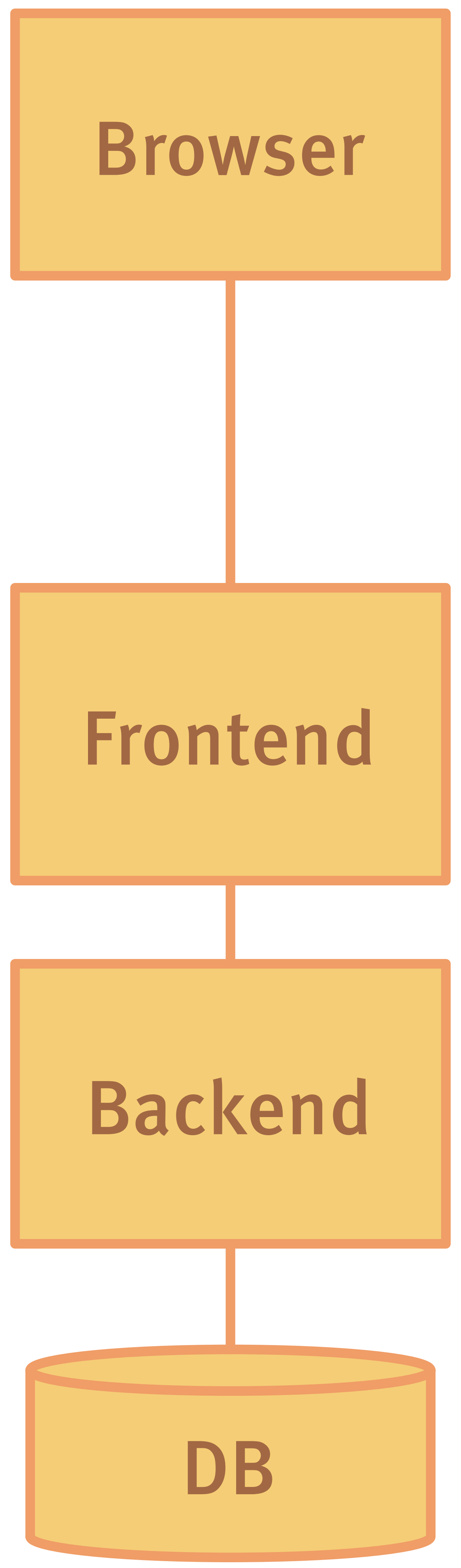
Let’s start with the enterprise



The J2EE(TM) 1.4 Tutorial, http://docs.oracle.com/cd/E17802_01/j2ee/j2ee/1.4/docs/tutorial-update2/doc/Overview7.html


Application Server
Container
Container
EJB Container Provider (Systems expert)
Packaged Application
EJB
DD
JSP
Deployer (Operation environment expert)
Application Server
Container
Container
Database
Existing Systems
JAR file
EJB
DD
Bean Provider (Domain expert)
Application Assembler (Domain expert)
System Administrator (Administers and monitors the system)


Browser
Application
DB


Browser
Frontend
Backend
DB

Assumptions to be challenged

One single system

One single environment

Predictable load

Clear & distinct roles

Planned releases

Built because they have to be

Somewhat Limited Agility

Increased Desaster Potential

1 2 3

Cut Things into Pieces

Small, lightweight, focused apps

My favorite programmer’s story

Task: Read a file of text, determine the *n* most frequently used words, and print out a sorted list of those words along with their frequencies.

Donald Knuth

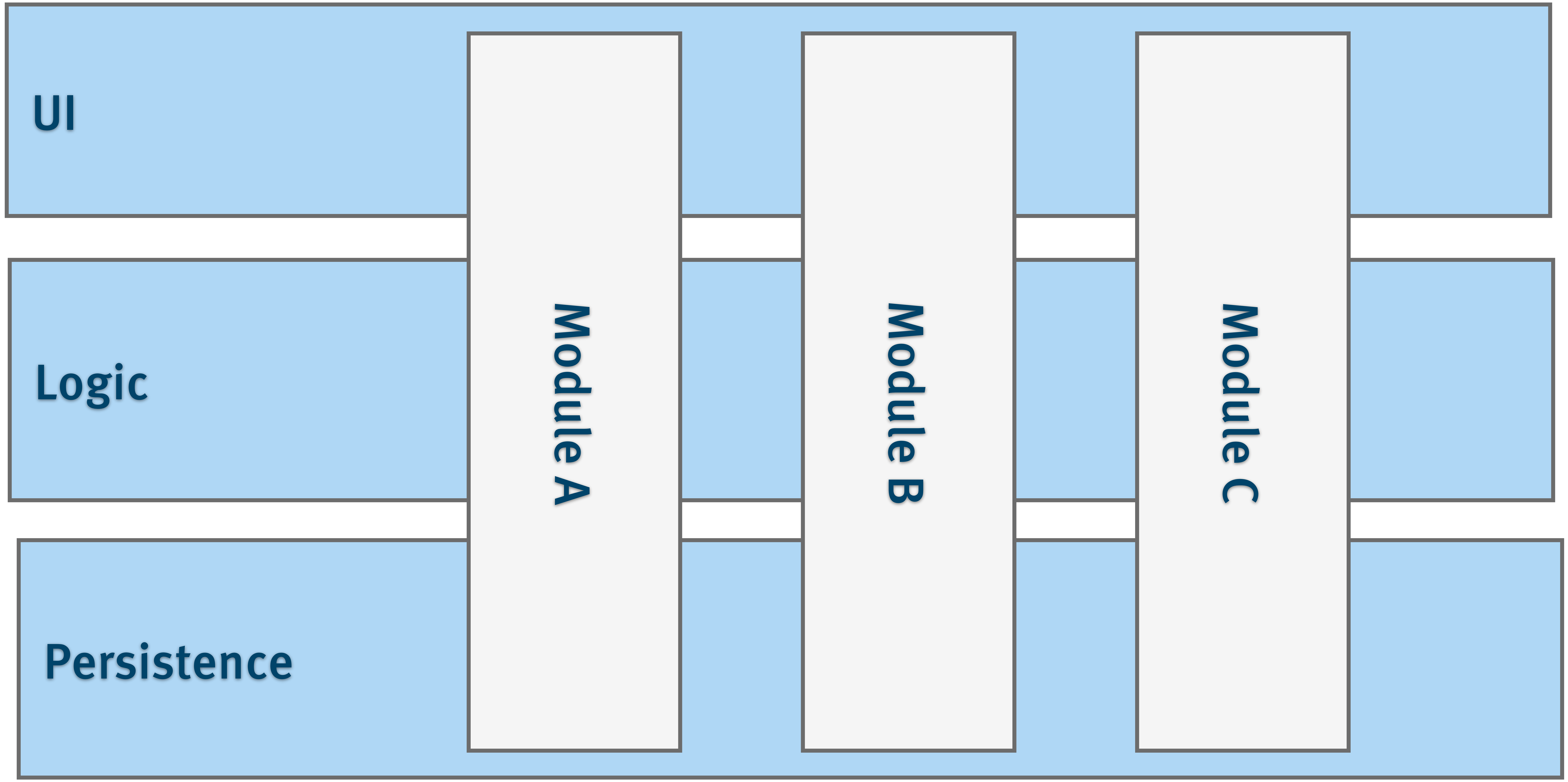
10-page literal Pascal program, including innovative new data structure

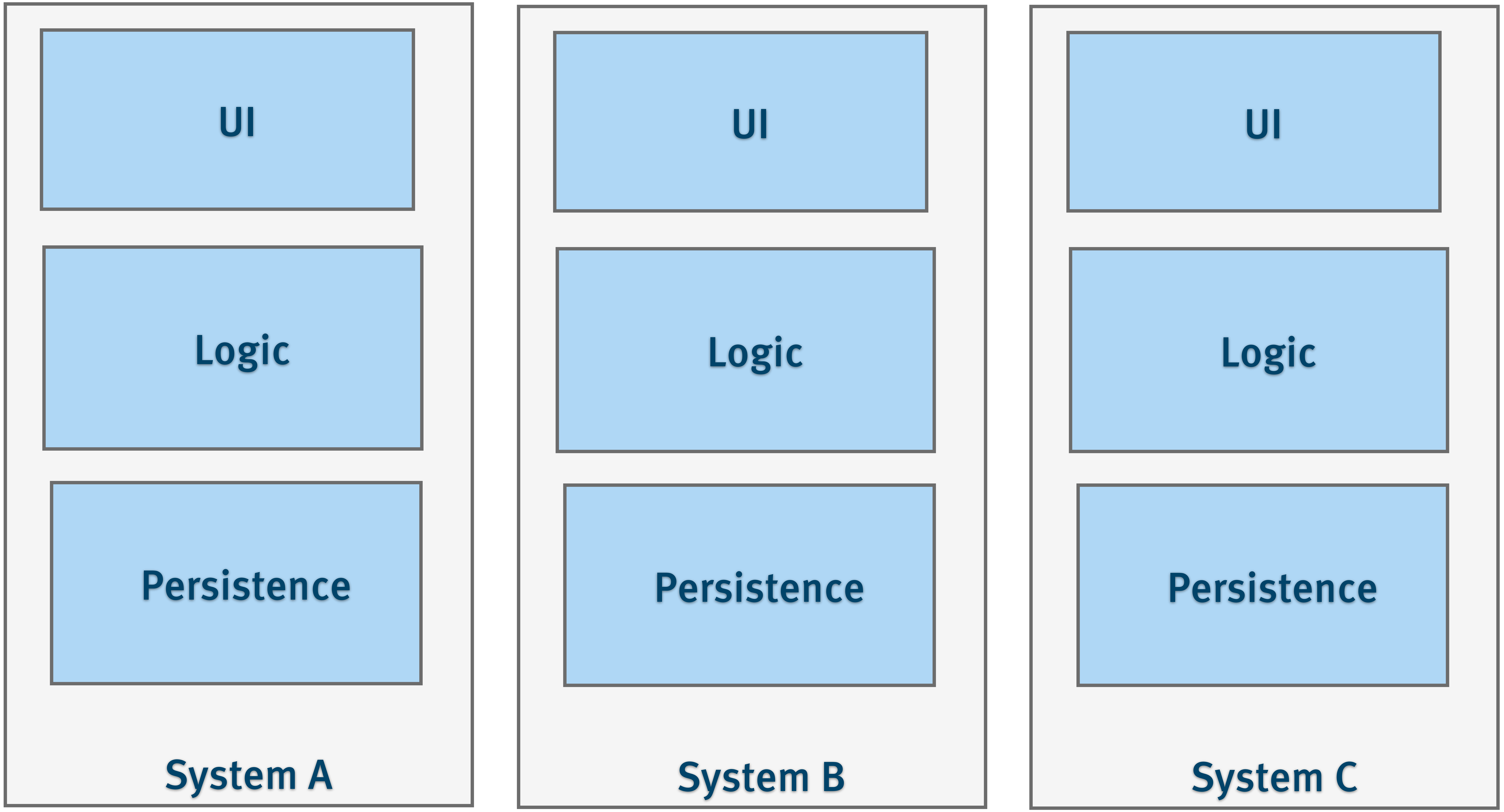
Doug McIlroy

```
tr -cs A-Za-z '\n' |
tr A-Z a-z |
sort |
uniq -c |
sort -rn |
sed ${1}q
```

Dr. Drang, http://www.leancrew.com/all-this/2011/12/more-shell-less-egg/

Small, lightweight, focused apps


UI
Logic
Persistence
Module A
Module B
Module C


UI
Logic
Persistence
System A
UI
Logic
Persistence
System B
UI
Logic
Persistence
System C

Assumptions to be challenged

Large systems with a single environment

Separation internal/external

Predictable non-functional requirements

Clear & distinct roles

Planned releases

Built because they have to be

http://12factor.net

The Twelve-Factor App

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing Services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

VIII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity
Keep development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

App characteristics

Separate, runnable process

Accessible via standard ports & protocols

Shared-nothing model

Horizontal scaling

Fast startup & recovery

MICROSERVICES
MICROSERVICES EVERYWHERE!
memegenerator.net

Microservice Characteristics

small

each running in its own process

lightweight communicating mechanisms (often HTTP)

built around business capabilities

independently deployable

mininum of centralized management

may be written in different programming languages

may use different data storage technologies

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

System Characteristics

Separate (redundant) persistence

Internal, separate logic

Domain models & implementation strategies

Separate UI

Separate development & evolution

Limited interaction with other systems

Autonomous deployment and operations

In search for a name ...

Executable component

Sovereign system

Bounded system

Small enough system

System

Autonomous system

Self-contained system

Large enough system

Cohesive system

Logical node

Domain unit

Independent system

Self-sufficient component

Small system

Full-stack service

Not-so-micro-service

Self-Contained System (SCS)

SCS Characteristics

Autonomous web application

Owned by one team

No sync remote calls

Service API optional

Includes data and logic

No shared UI

No or pull-based code sharing only

	SCS	App	Microservice
Size (kLoC)	1-50	0.5-10	0.1-?
State	Self-contained	External	Self-contained
# per Logical System	5-25	>50	>100
Communication between units	No (if possible)	?	Yes
UI	Included	Included	External (?)
UI Integration	Yes (web-based)	?	?

Simple process run model

Back to building servers

Closer to the metal

Isolation and independence

Polyglotism

Built for replacement,
not for re-use

amazon.com®

FAQ

Press Release

Customer Experience

User Manual

Werner Vogels, http://www.allthingsdistributed.com/2006/11/working_backwards.html

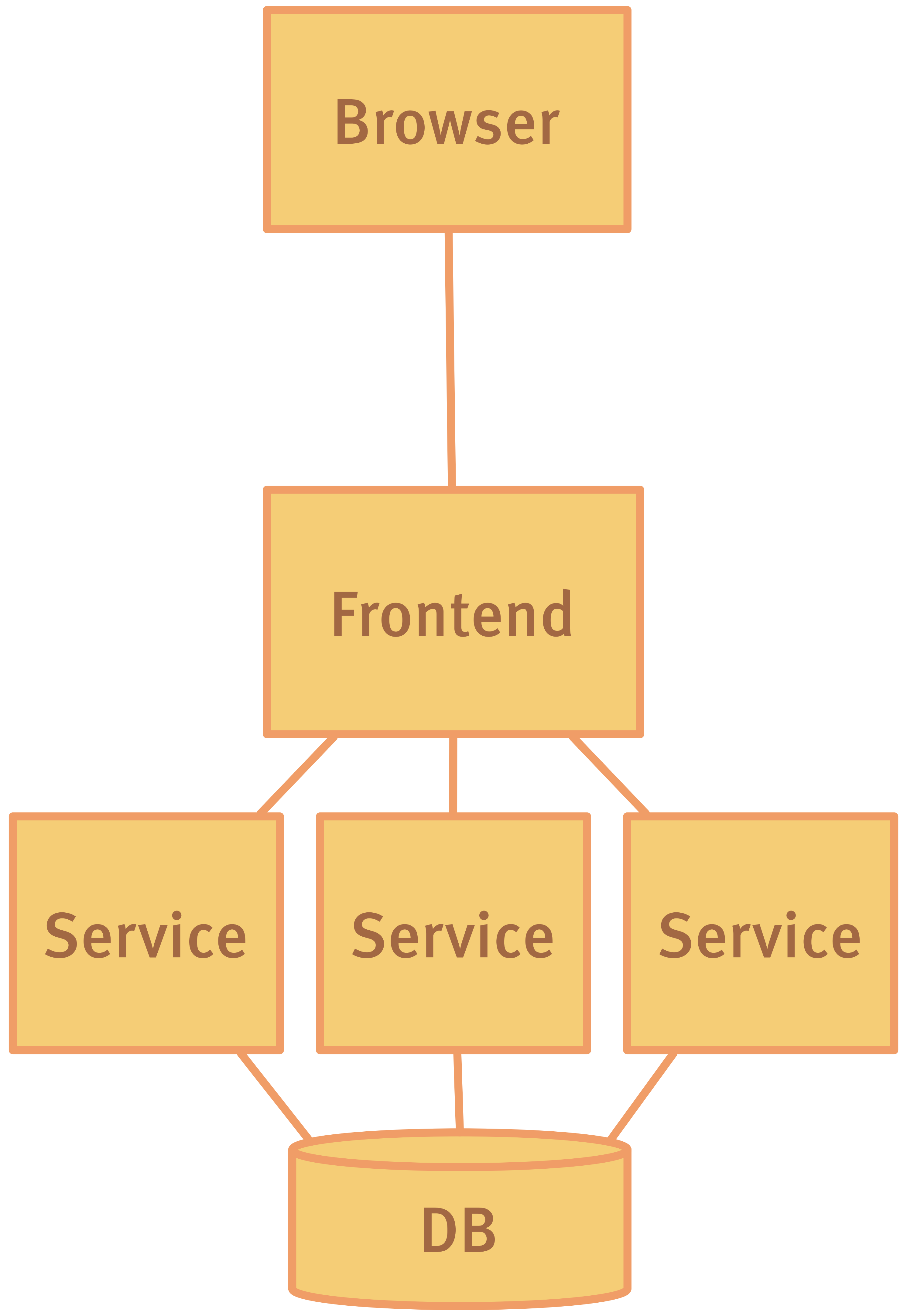
GROUPON®

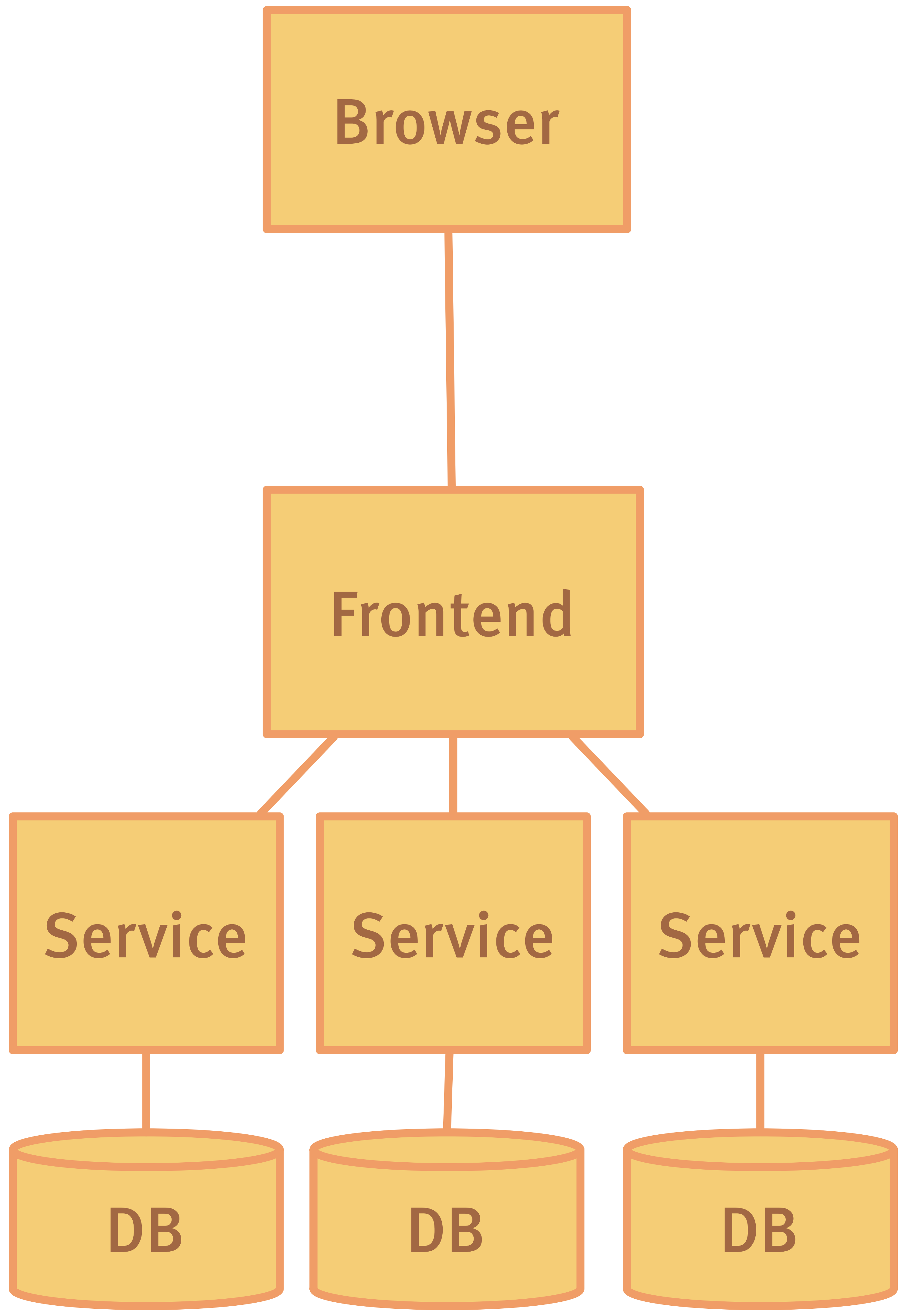
Dismantled monolith

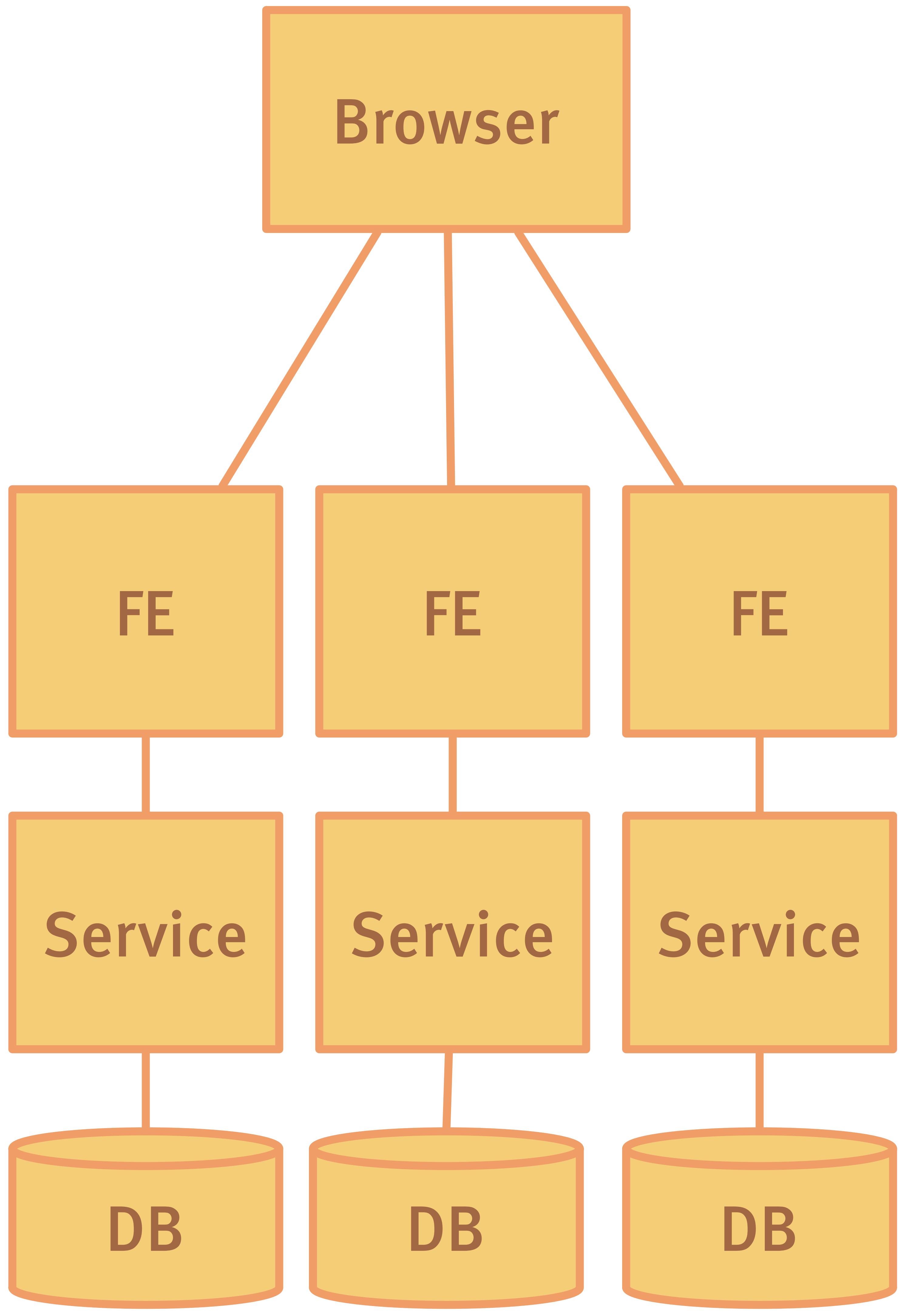
Backend & front-end services

(Re-Implementation in Node.js)

https://engineering.groupon.com/2013/misc/i-tier-dismantling-the-monoliths/


Browser
Frontend
Service
Service
Service
DB


Browser
Frontend
Service
Service
Service
DB
DB
DB


Browser
FE
FE
FE
Service
Service
Service
DB
DB
DB

Organization $\longleftrightarrow$ Architecture

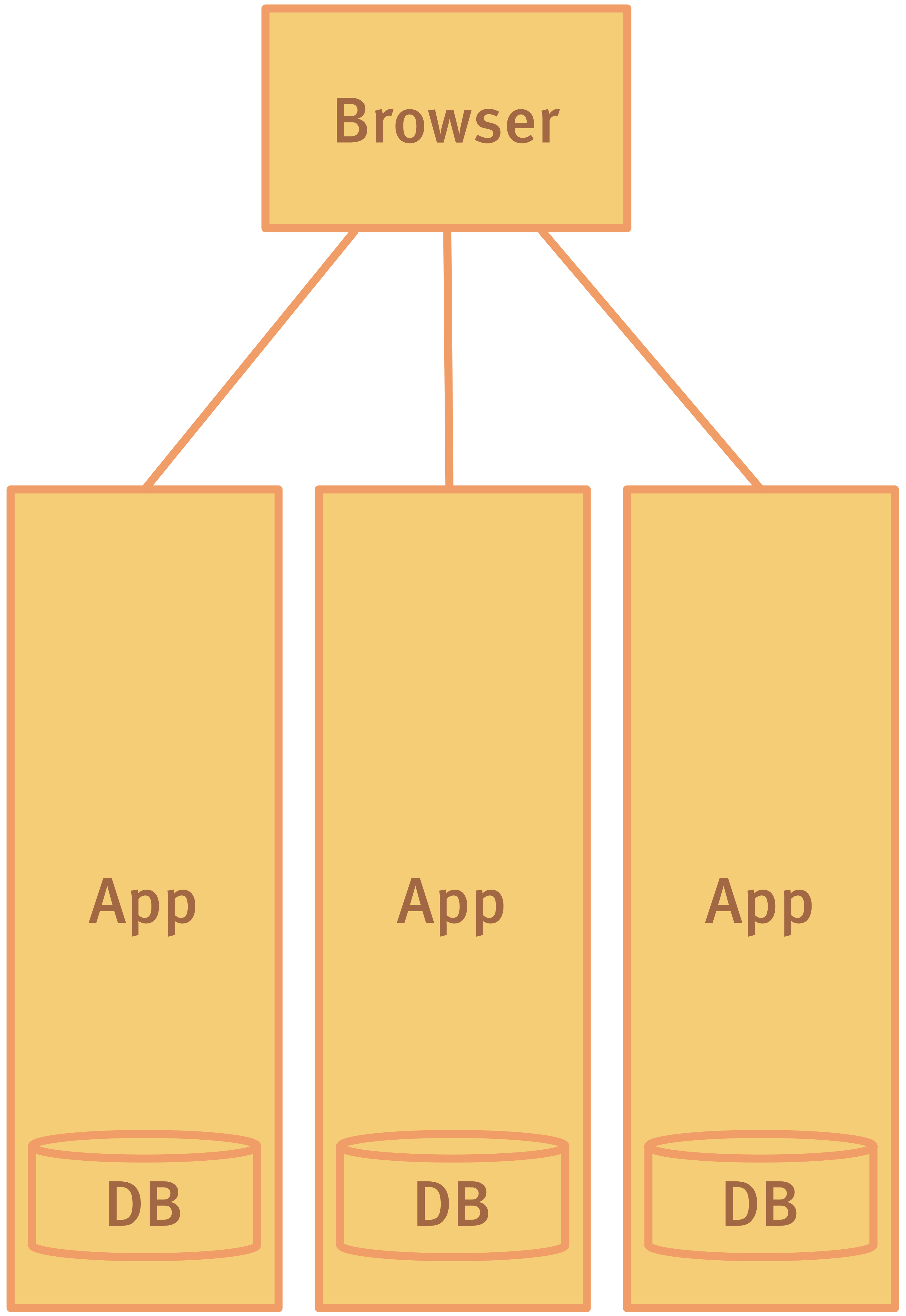
OTTO

Independent “Verticals”

REST-based macro architecture

Individual micro architecture

Kraus, Steinacker, Wegner: Teile und Herrsche – Kleine Systeme für große Architekturen, http://bit.ly/152cXbx


Browser
App
DB
App
DB
App
DB

amazon.com

Services as DNA

“Dogfooding”

Two-pizza rule

Steve Yegge, https://plus.google.com/110981030061712822816/posts/AaygmbzVeRq

Tools

Embedded Jetty

Play

Netty

vert.x

Modern Java EE containers

Akka

Node.js

DropWizard

Example Micro Architecture Stacks

Typesafe (Play, Akka), Java 7

Typesafe (Play, Akka), Scala

JRuby/Rails, Ruby/Sinatra, Passenger

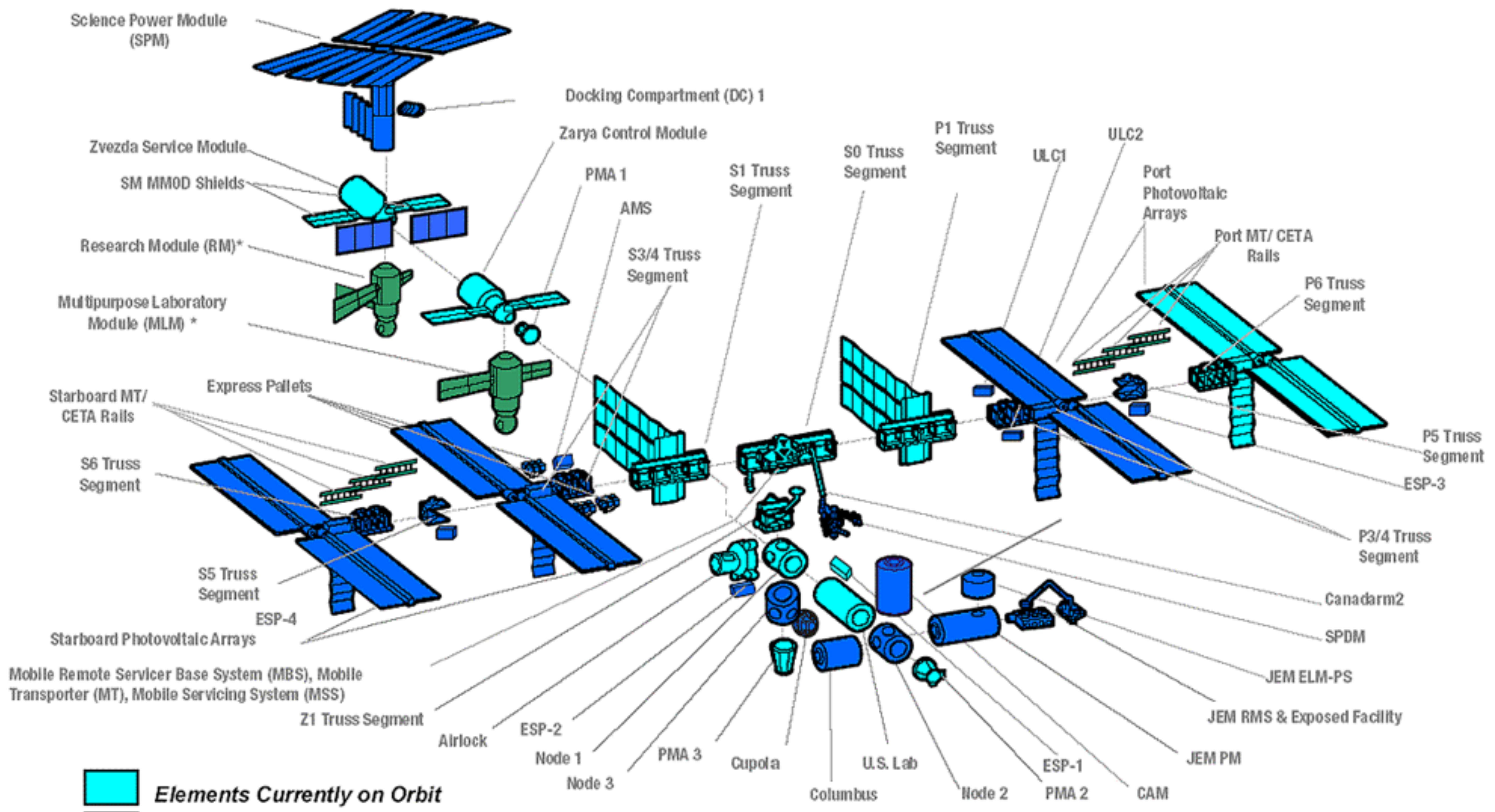
Play 2, Java 8, Spring 4, Spring Data, QueryDSL, Hystrix, Logback

Java 7, JAX-RS/Jersey, Jackson, Tomcat

Java 8, Jetty, Jersey 2.x, HalBuilder, Archaius, Ribbon, Eureka, Google OAuth2 Client Library

ISS Technical Configuration

Endorsed by ISS Heads of Agency on July 23, 2004



🟦 *Elements Currently on Orbit*

🟦 *Elements Pending US Shuttle Launch*

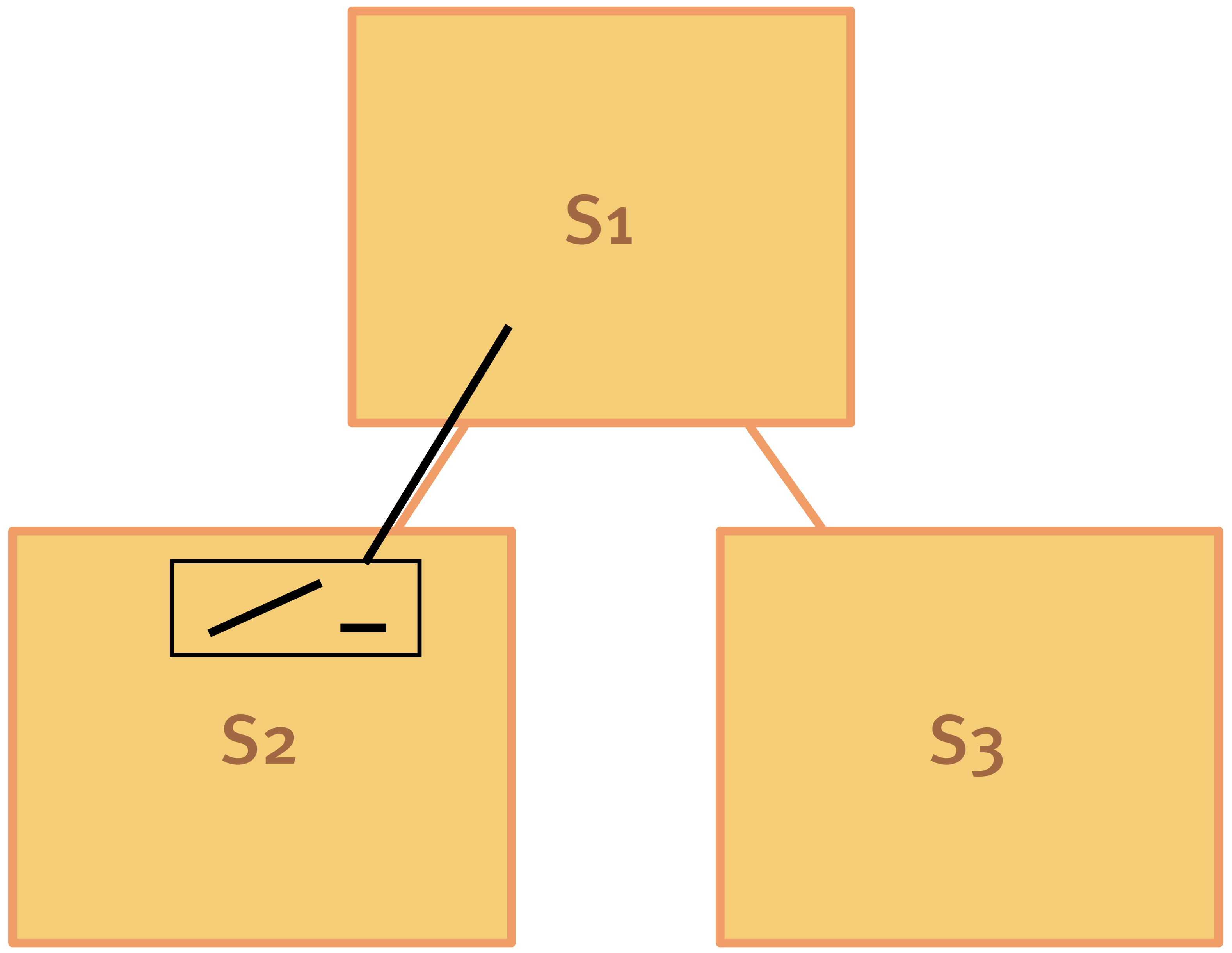
🟩 *Elements Pending Russian Launch*

•RM and MLM are included in Russian plans and launched on Russian vehicles

1 2 3

Integrate pieces to form a whole

Robust systems
Unreliable networks


S1
S2
S3

Tools

Akka

Hystrix

Finagle

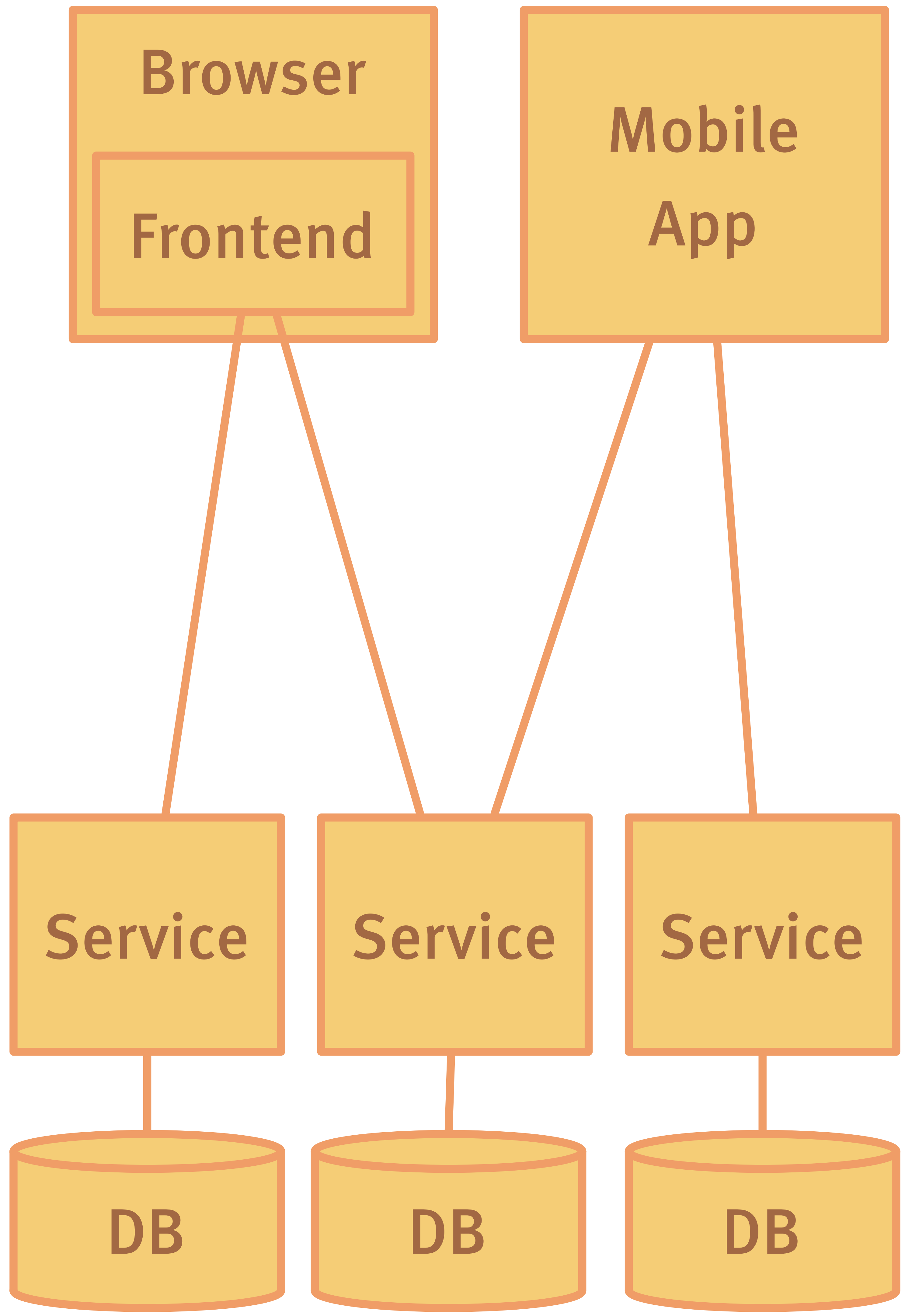
Smart aggregation

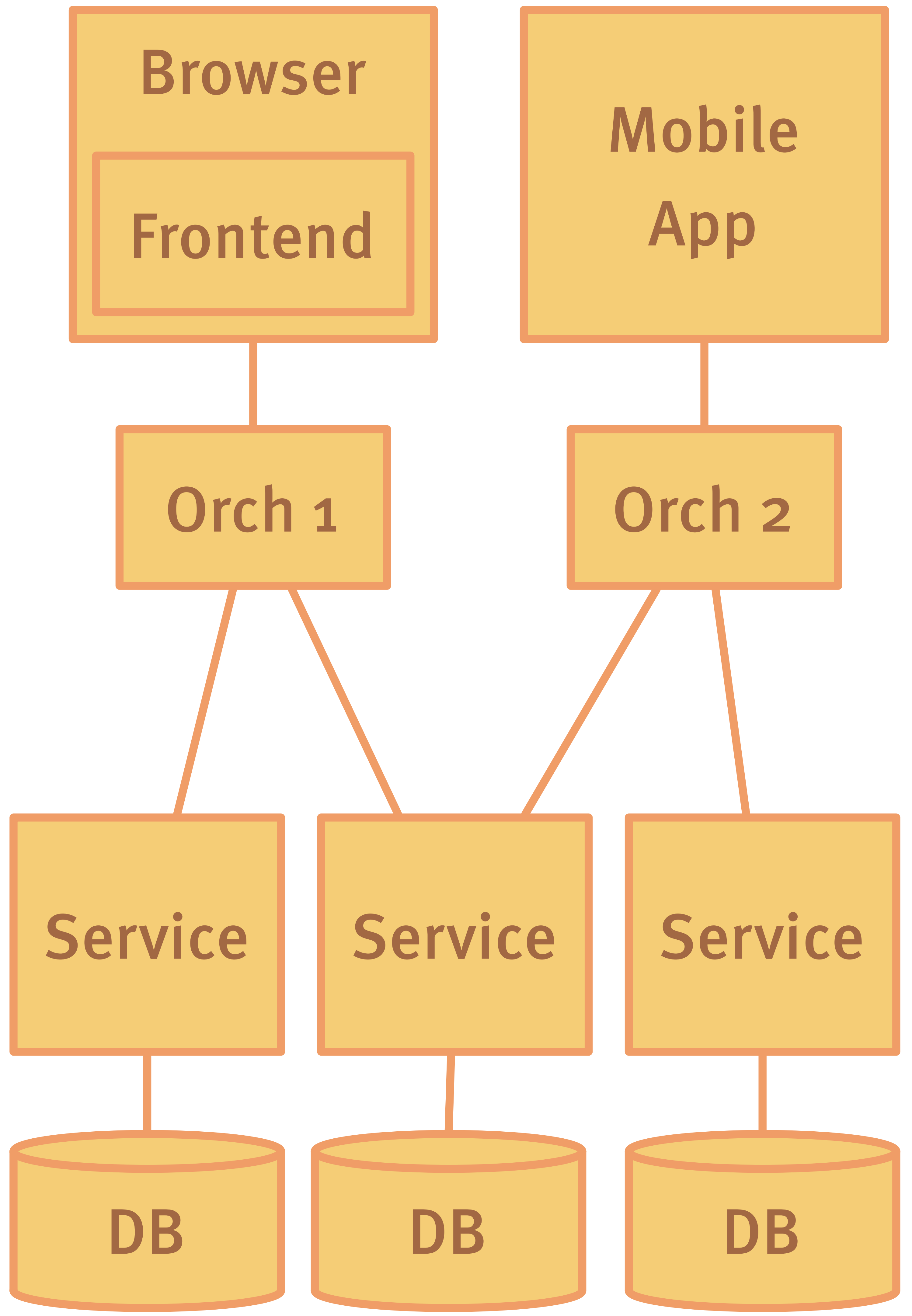
NETFLIX

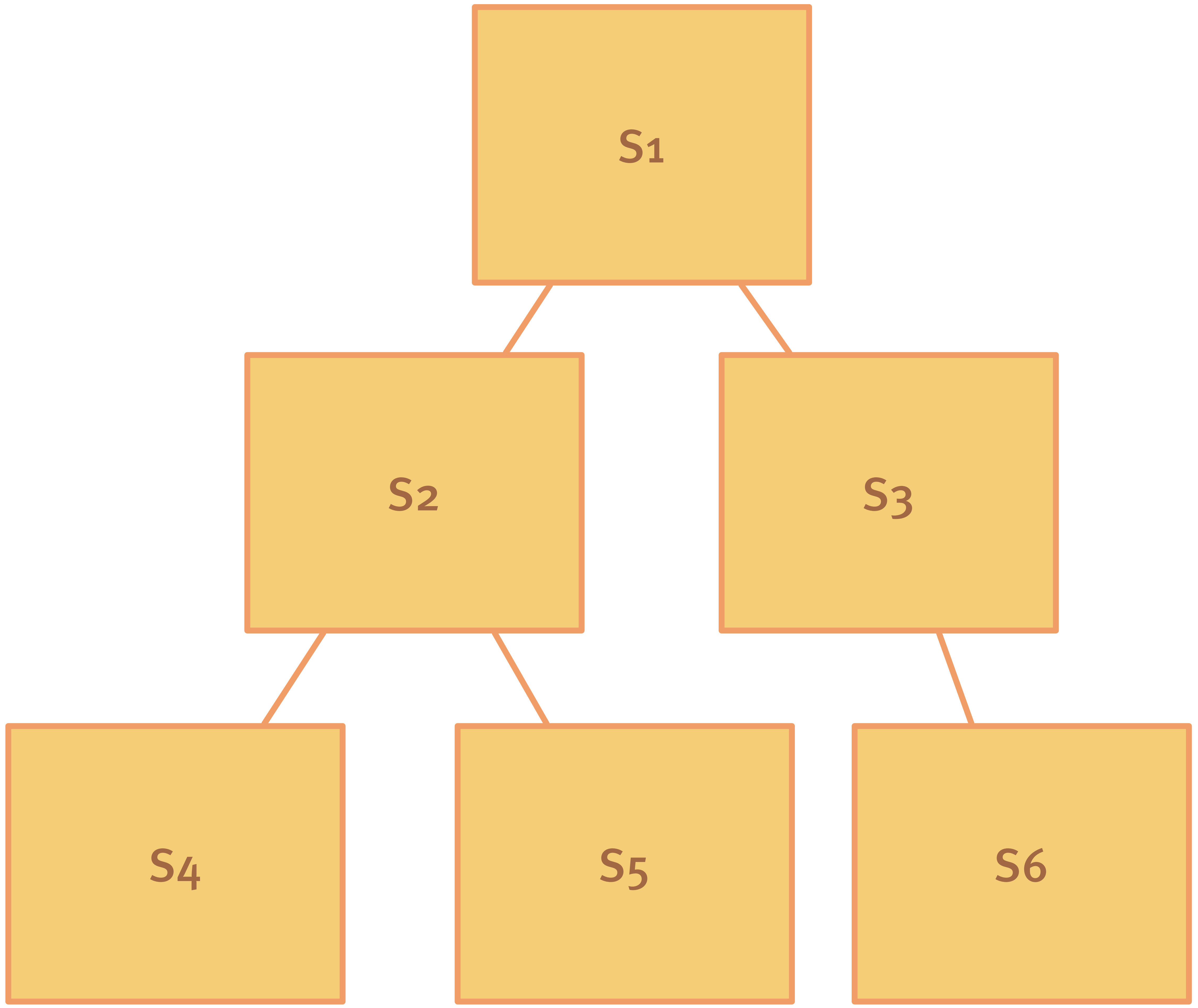
REST APIs

Client-specific orchestration

Streaming architecture


Browser
Frontend
Mobile App
Service
Service
Service
DB
DB
DB


Browser
Frontend
Mobile App
Orch 1
Orch 2
Service
Service
Service
DB
DB
DB


S1
S2
S3
S4
S5
S6

Tools

Storm

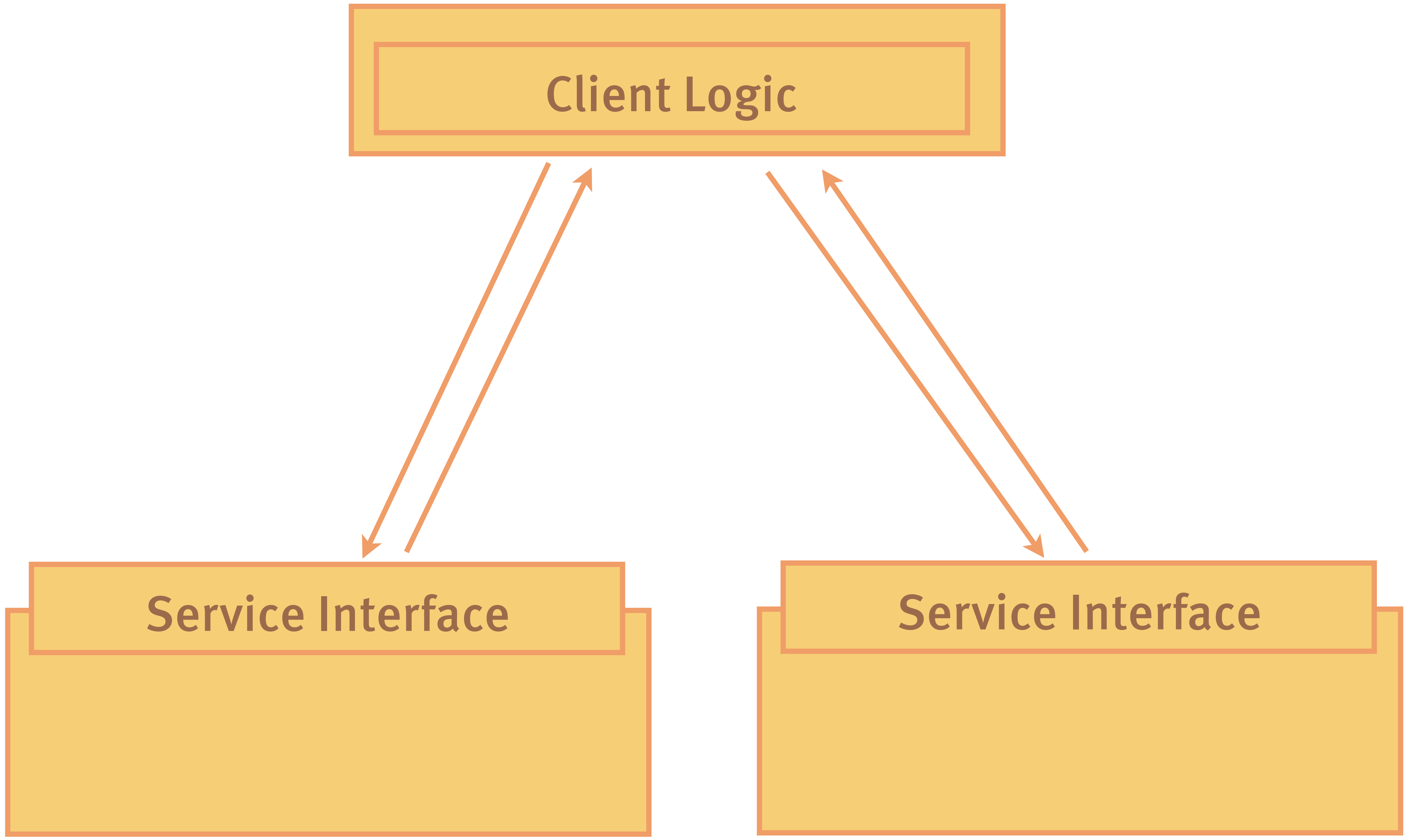
Rx

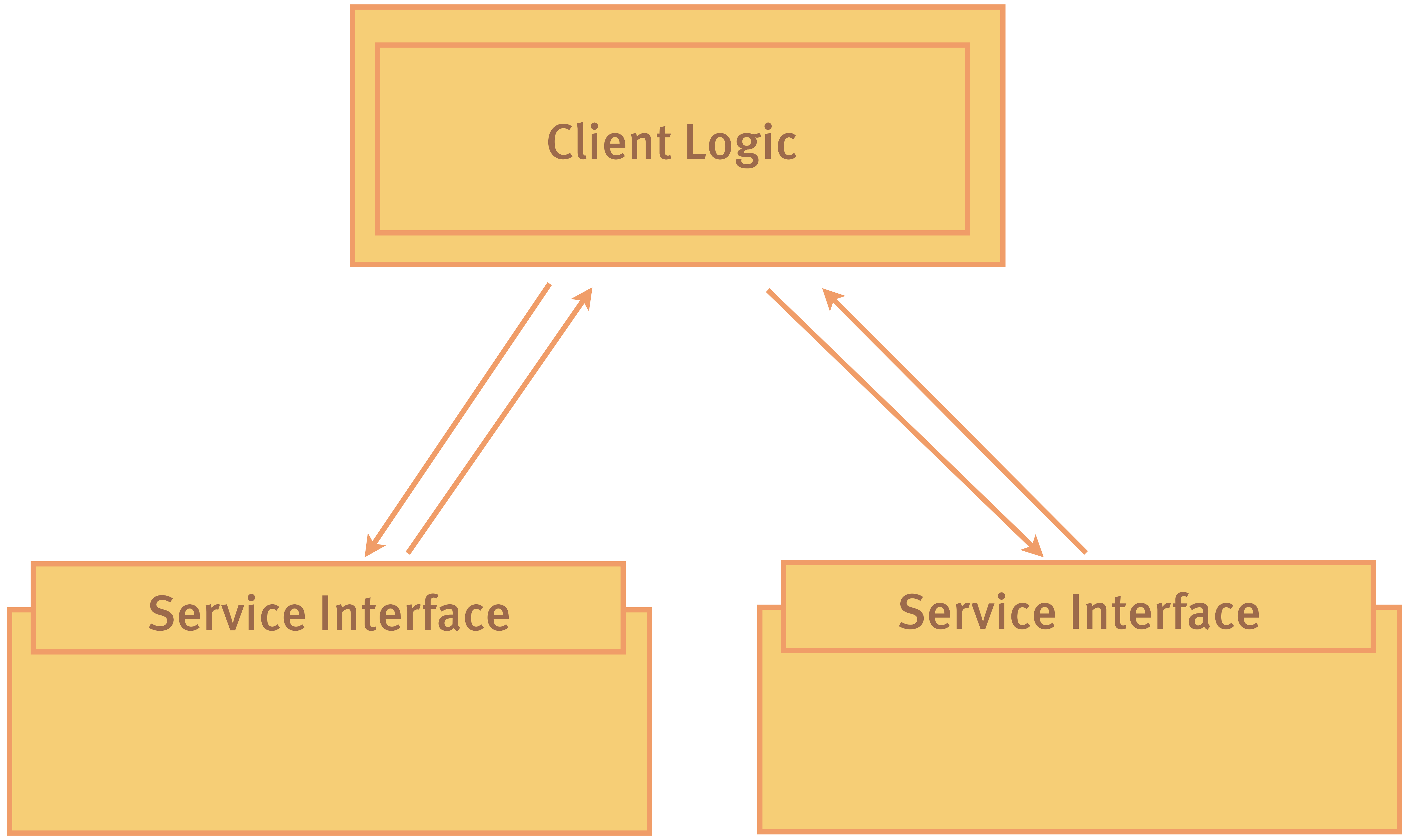
ql.io

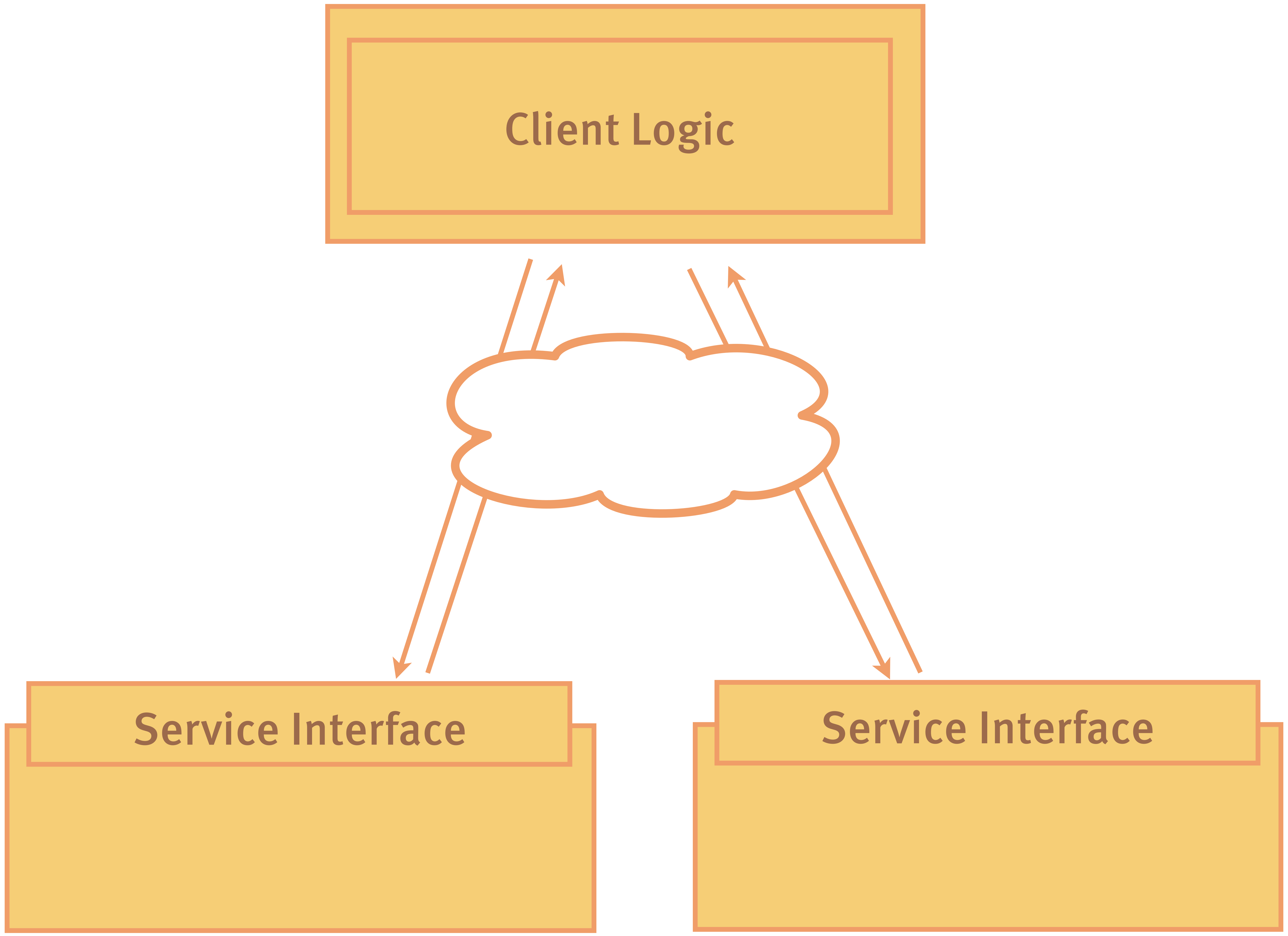
spray

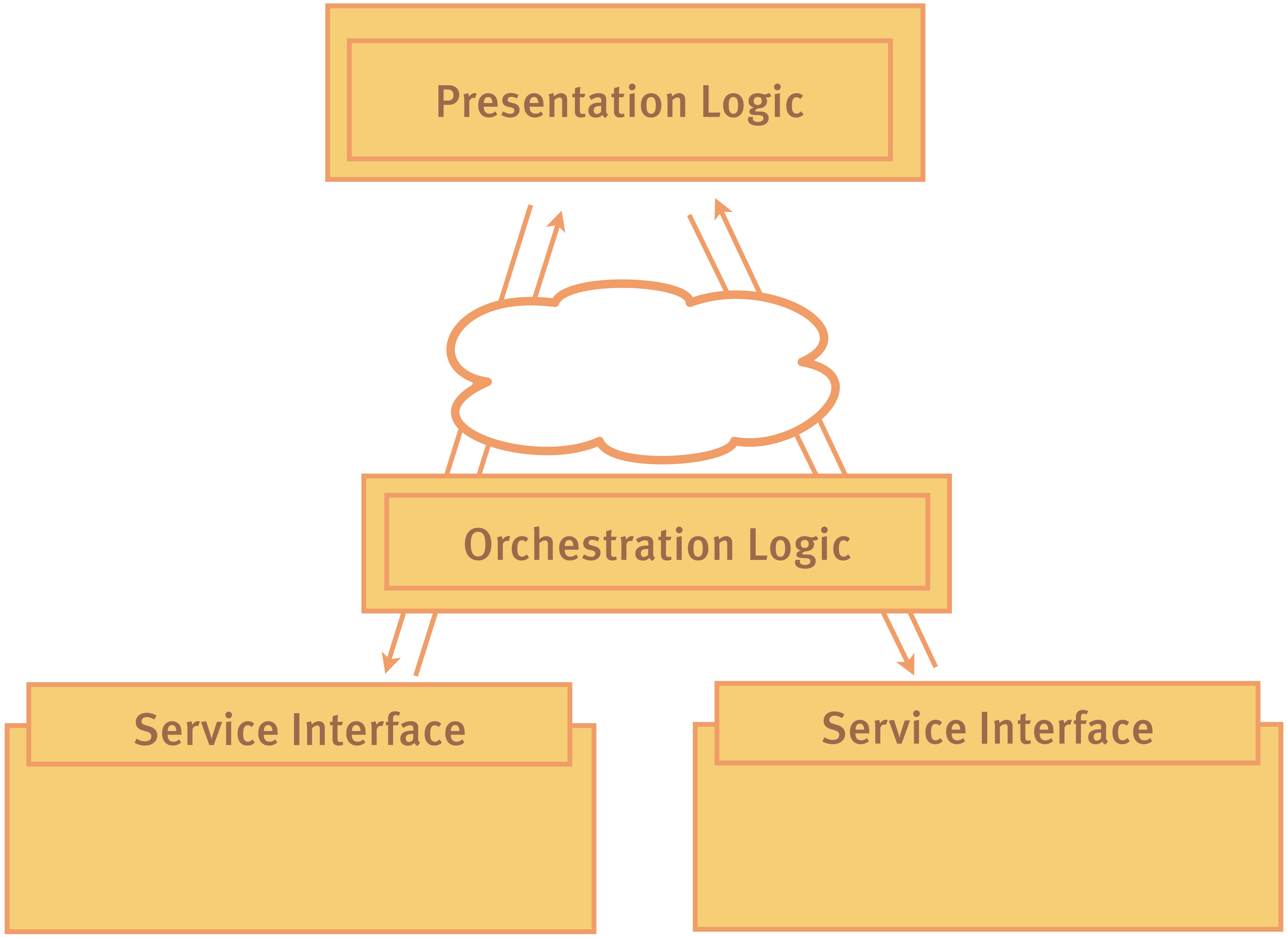
Play

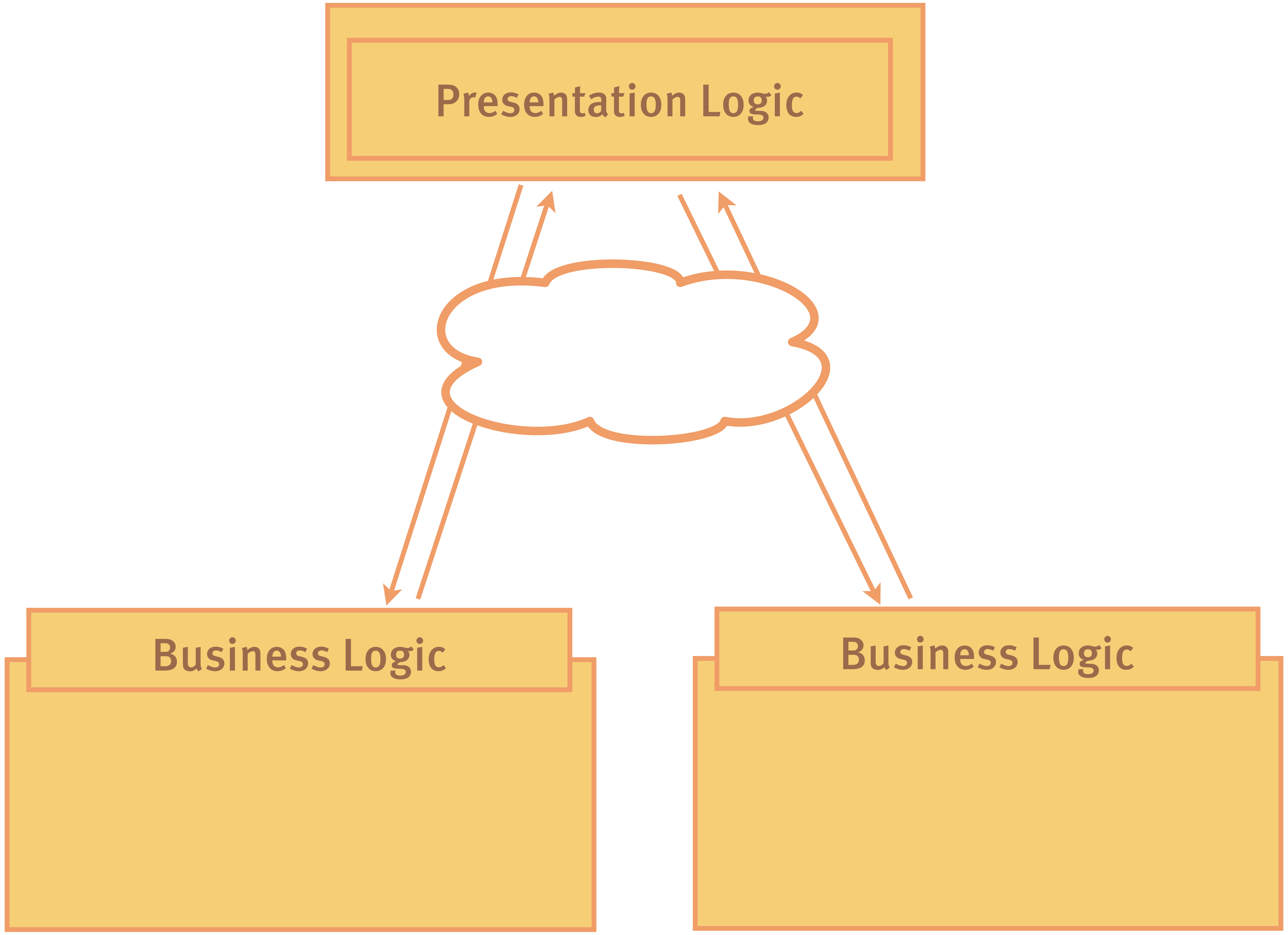
Web-native front-end integration

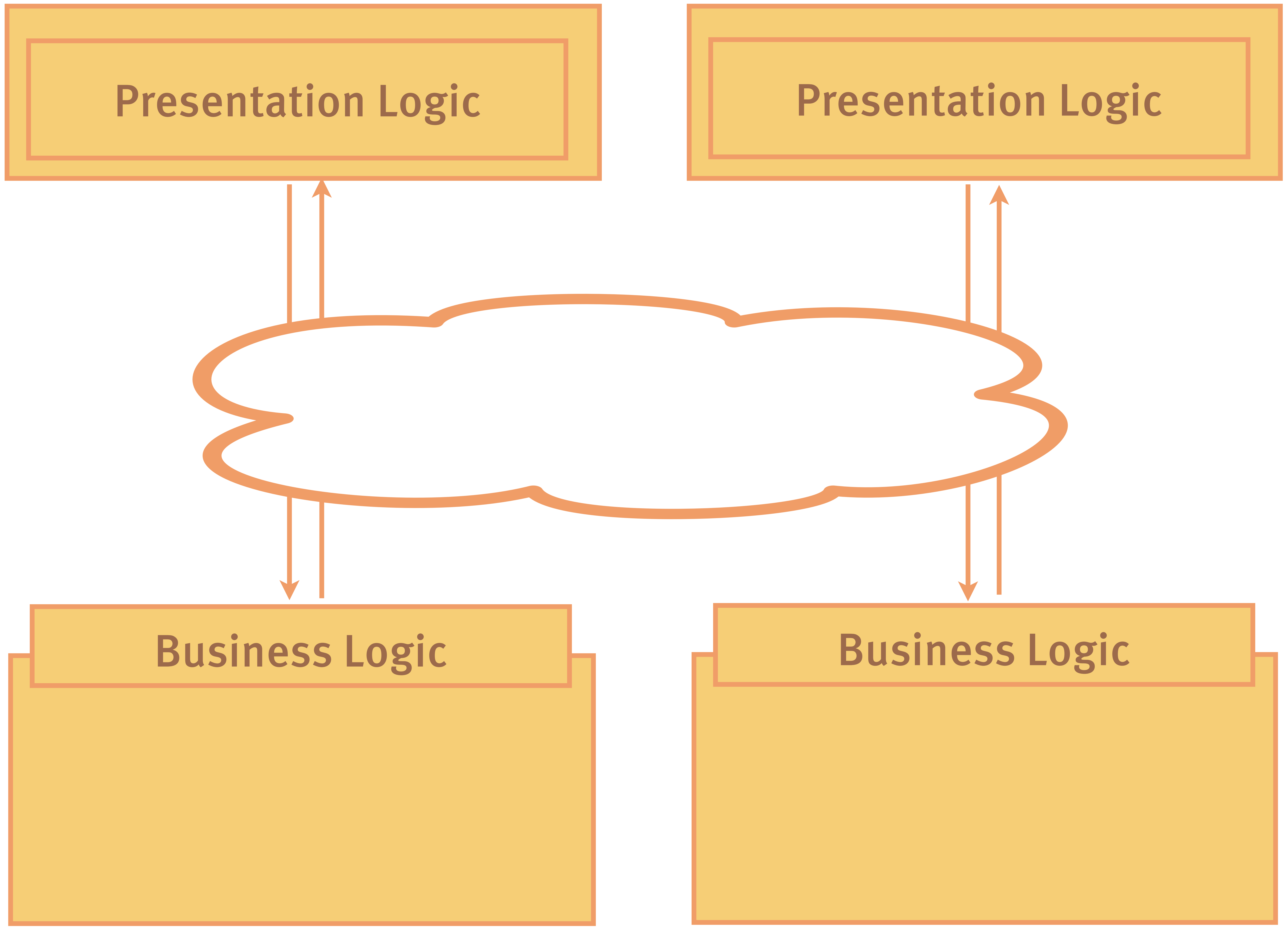

Client Logic
Service Interface
Service Interface


Client Logic
Service Interface
Service Interface


Client Logic
Service Interface
Service Interface


Presentation Logic
Orchestration Logic
Service Interface
Service Interface


Presentation Logic
Business Logic
Business Logic


Presentation Logic
Presentation Logic
Business Logic
Business Logic

GOV.UK

Simple semantic HTML

Open Data

Single domain – no portal

“Google as the homepage”

Polyglot environment

https://gds.blog.gov.uk/govuk-launch-colophon/

Tools & Approaches

MVC Web Frameworks

RESTful HTTP

ROCA

1 2 3

Change & run efficiently

Horizontal scaling

Virtualized operating system as container

Fully automated, repeatable deployment

Transparent monitoring

Etsy

Small changesets

Everyone deploys

Fast deploys

Change flags

Graphs/metrics

Fix fast/roll forward

Ross Snyder, http://www.slideshare.net/beamrider9/continuous-deployment-at-etsy-a-tale-of-two-approaches

NETFLIX

Fully cloud-based

Self-made PaaS

Simian Army

Adrian Cockcroft, http://www.infoq.com/presentations/Netflix-Architecture

Netflix Stack

Zuul	Edge Router
Eureka	Service Registry
Hystrix	Stability patterns
Ribbon	HTTP client on steroids
Karyon	Application blueprint
Archaius	Configuration
Asgard	Console
Servo	Annotation-based metrics
…	…

Many, many more at http://netflix.github.io

Simian Army

10-18 Monkey

Doctor Monkey

Conformity Monkey

Janitor Monkey

Chaos Monkey

Chaos Gorilla

Security Monkey

Latency Monkey

Tools

logstash

Packer

Vagrant

Metrics

Puppet

Zipkin

docker

Chef

KUKA
KUKA

Summary

Build smaller

Aggregate smartly

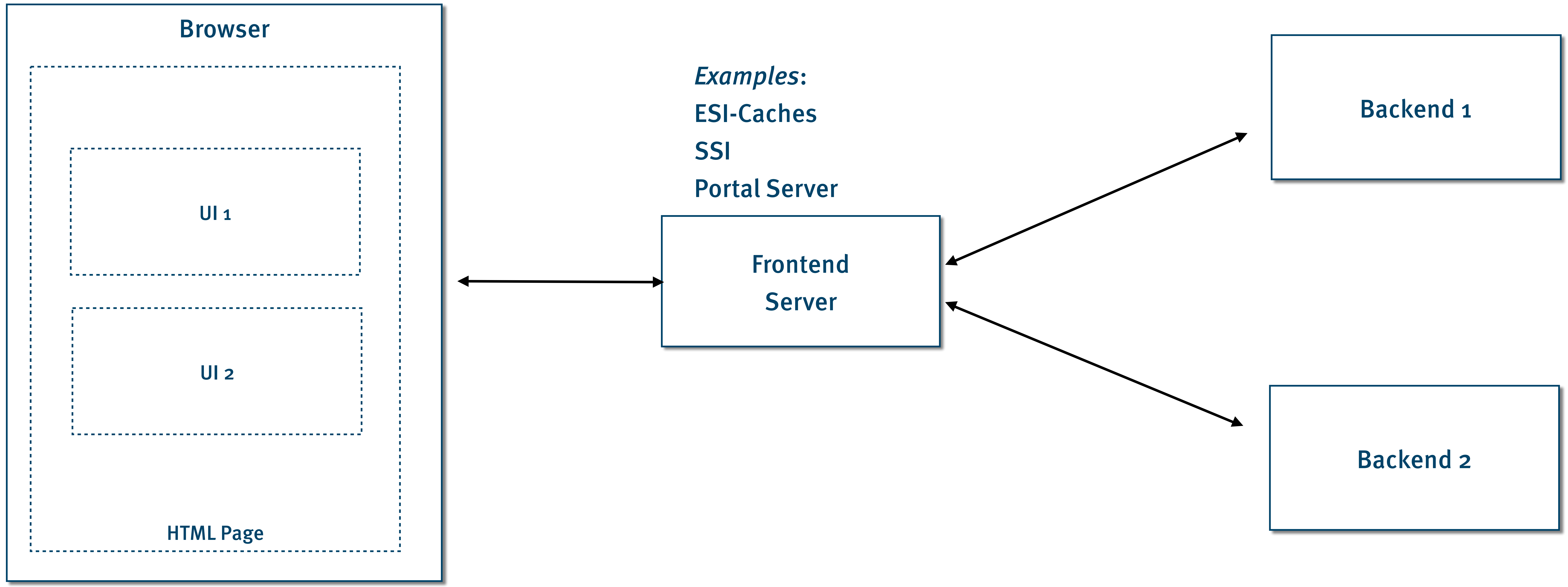
Merge run & change

Thank you!
Questions?
Comments?

Stefan Tilkov, @stilkov
stefan.tilkov@innoq.com
http://www.innoq.com/blog/st/
Phone: +49 170 471 2625

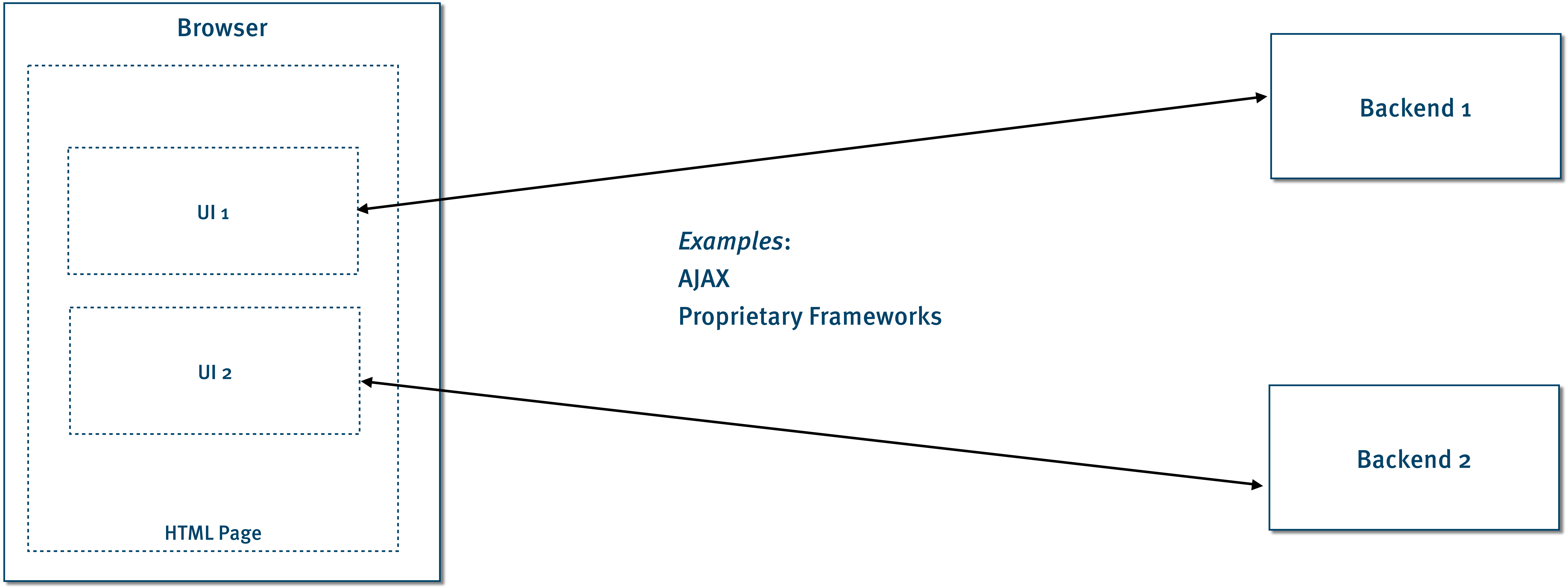
Backup

Server-side integration



Examples:
ESI-Caches
SSI
Portal Server

Client-side integration

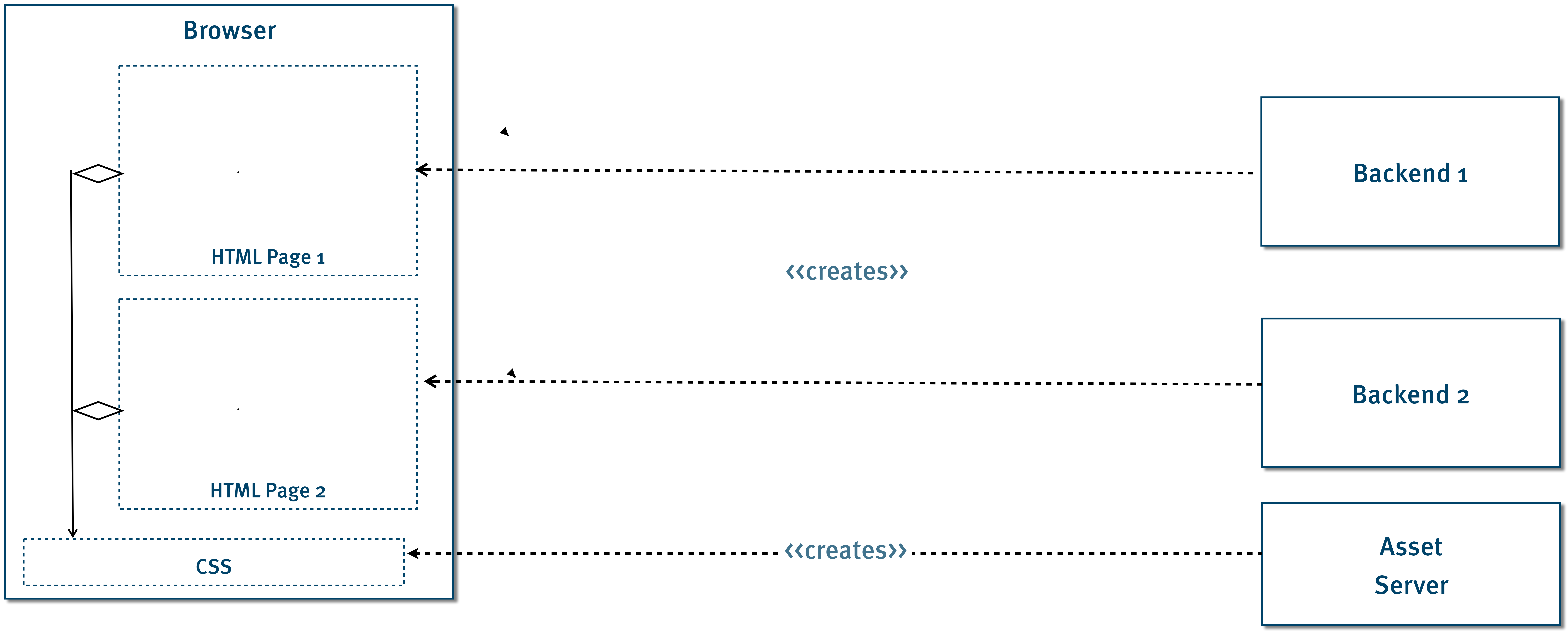


Examples:

AJAX

Proprietary Frameworks

Links


Browser
HTML Page 1
HTML Page 2
CSS
«creates»
«creates»
Backend 1
Backend 2
Asset Server