{Nano|Micro|Mini}-Services? Modularization for Sustainable Systems

Stefan Tilkov | innoQ
stefan.tilkov@innoq.com
@stilkov

microXchg 2015 – The Microservices Conference in Berlin

Thursday, 12 February 2015 at 08:30 - Friday, 13 February 2015 at 17:30 (CET)
Berlin, Germany

microxchg 2015

http://microxchg.io

1. Reviewing architectures

Generic Architecture Review Results

Building features takes too long

Technical debt is well-known and not addressed

Deployment is way too complicated and slow

Architectural quality has degraded

Scalability has reached its limit

"-ility" problems abound

Replacement would be way too expensive

Any architecture's quality is inversely proportional to the number of bottlenecks limiting its evolution, development, and operations

«Insert Obligatory Conway Reference Here»

Conway’s Law

Organization → Architecture

“Organizations which design systems are constrained to produce systems which are copies of the communication structures of these organizations.” – M.E. Conway

Reversal 1

Organization ← Architecture

Any particular architecture approach constraints organizational options – i.e. makes some organizational models simple and others hard to implement.

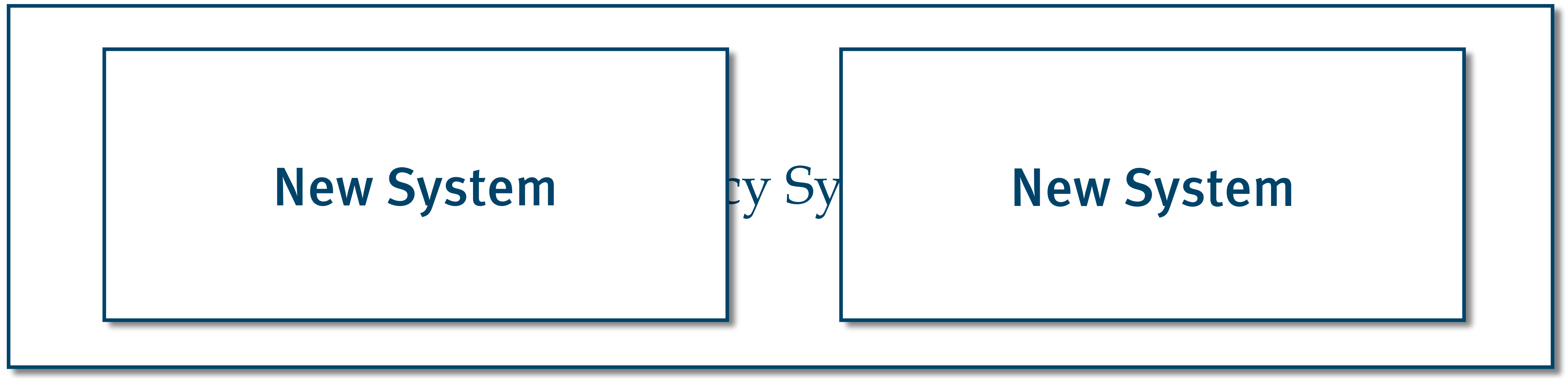
Reversal 2

Organization ← Architecture

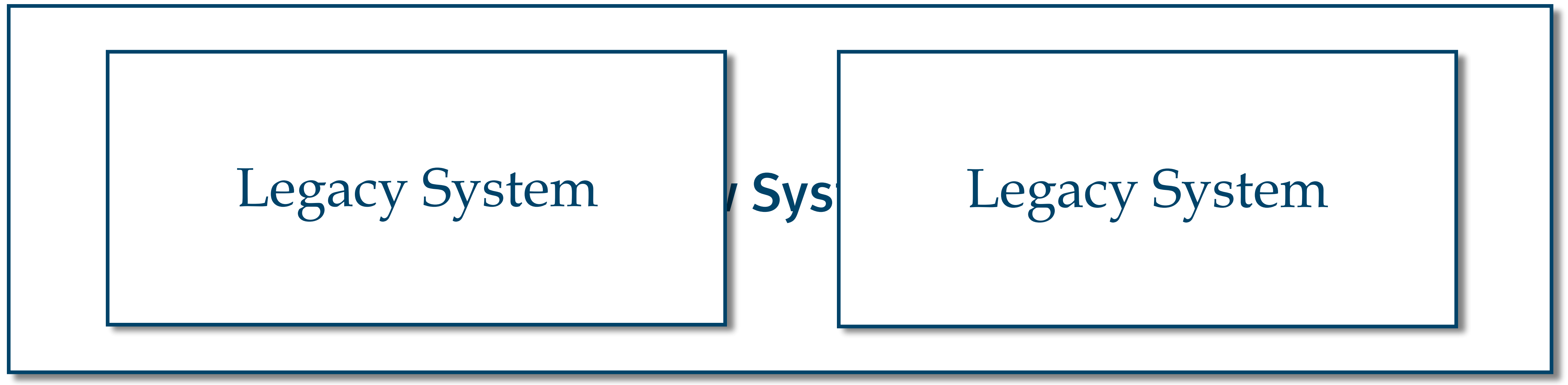
Choosing a particular architecture can be a means of optimizing for a desired organizational structure.

2. System boundaries

Modularization



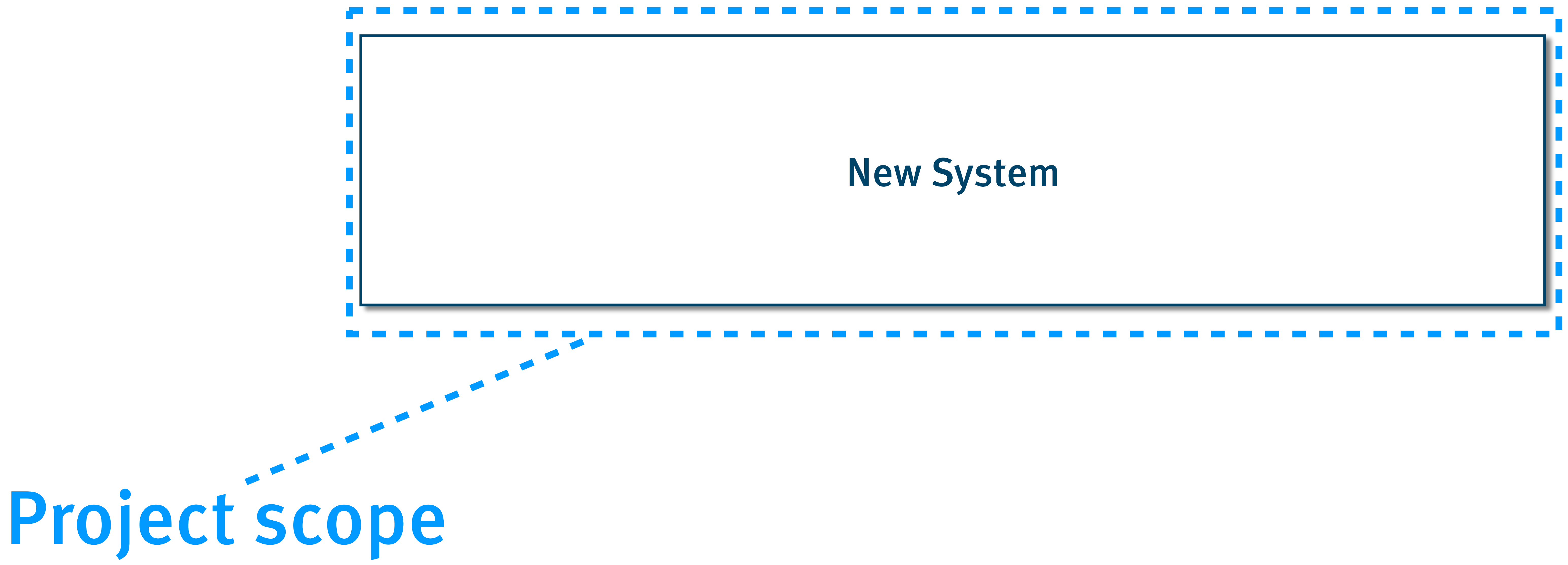
Consolidation



Modernization

Legacy System

Greenfield



1 Project = 1 System?

Size	Modularization
1-50 LOC	single file
50-500 LOC	few files, few functions
500-1000 LOC	Library, class hierarchy
1000-2000 LOC	Framework + application
>2000 LOC	multiple applications

System Characteristics

Separate (redundant) persistence

Internal, separate logic

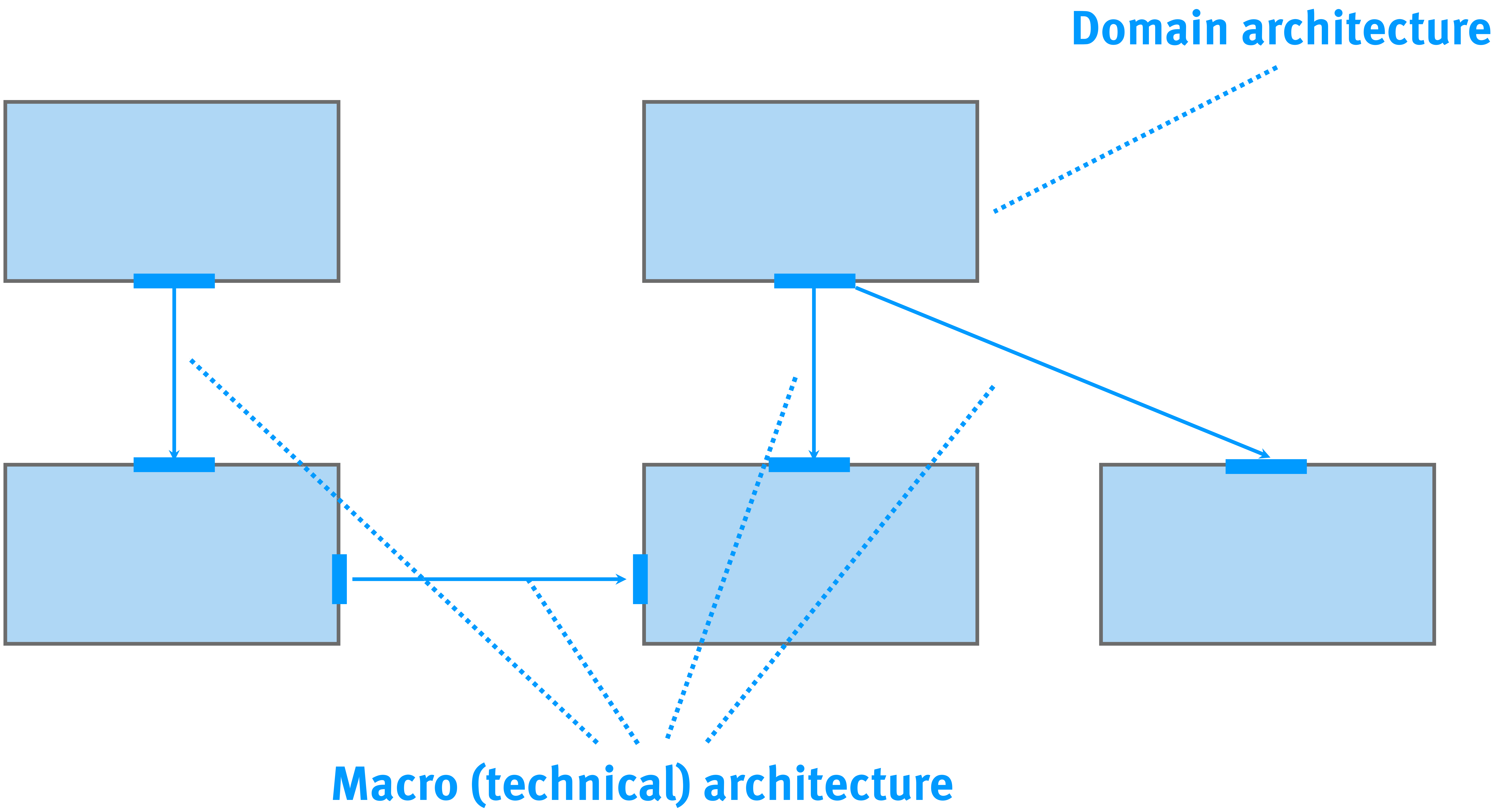
Domain models & implementation strategies

Separate UI

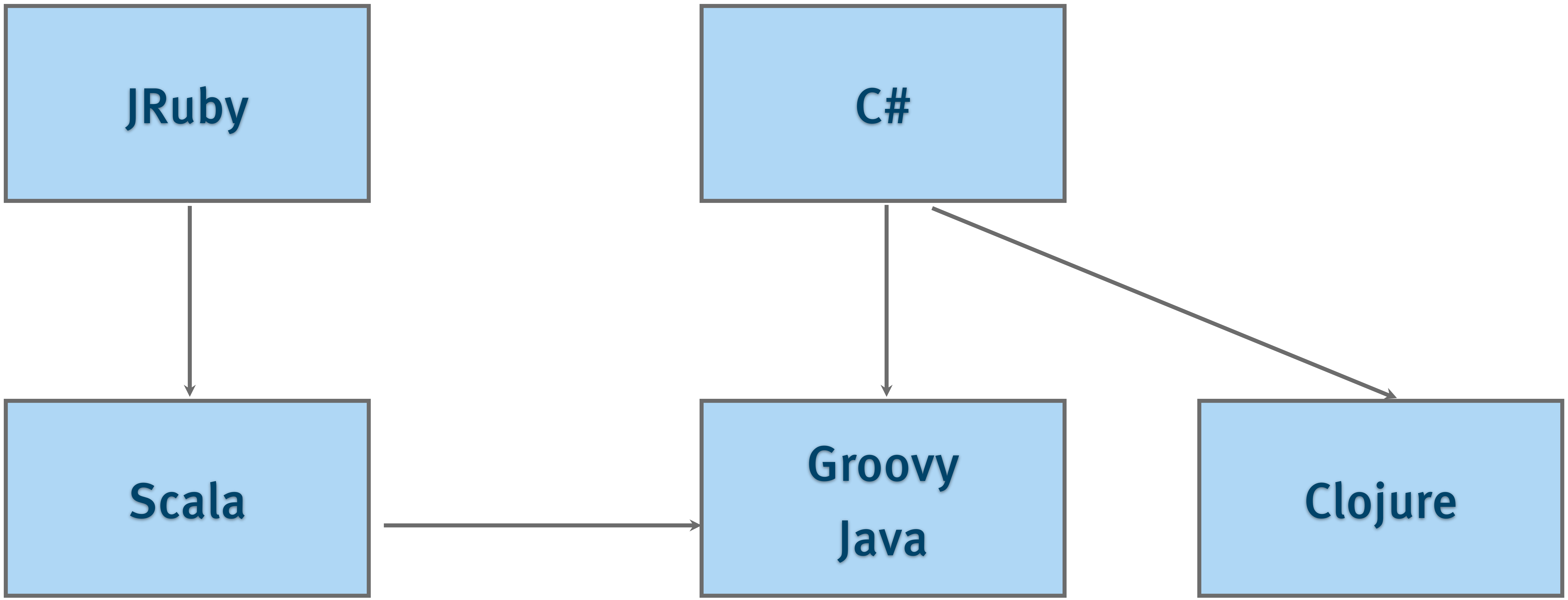
Separate development & evolution

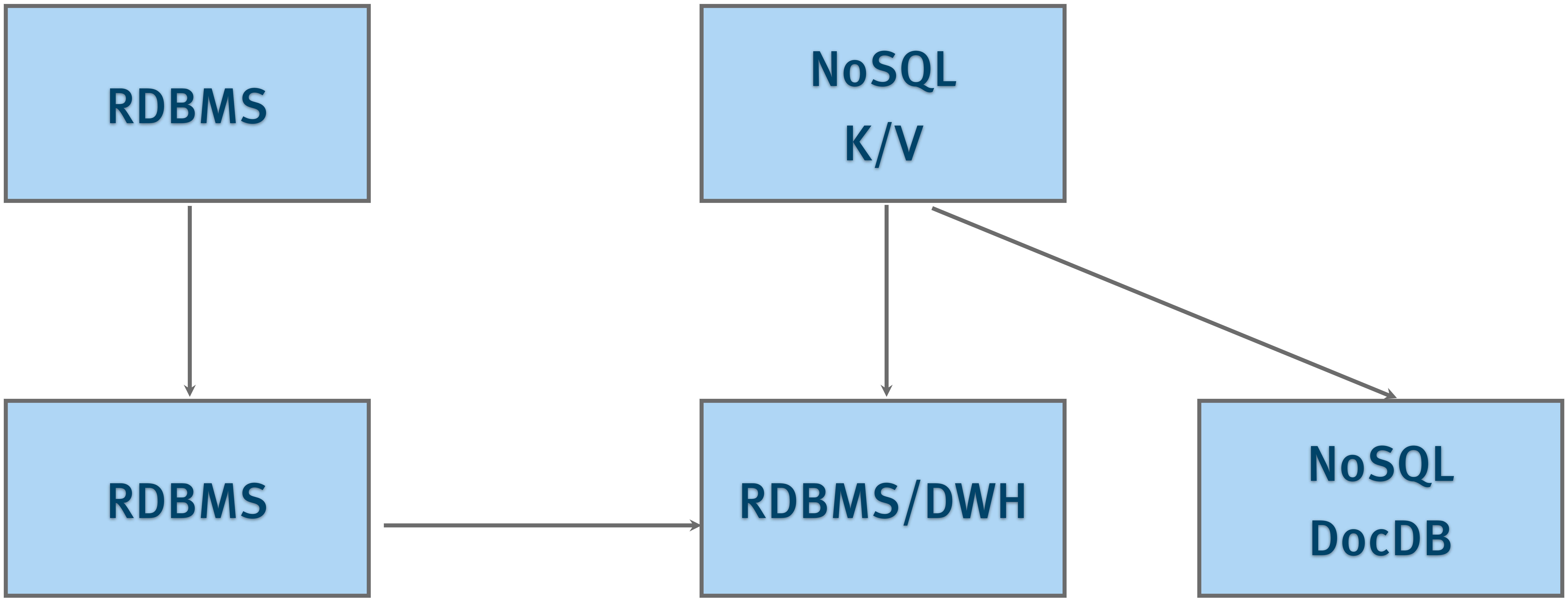
Limited interaction with other systems

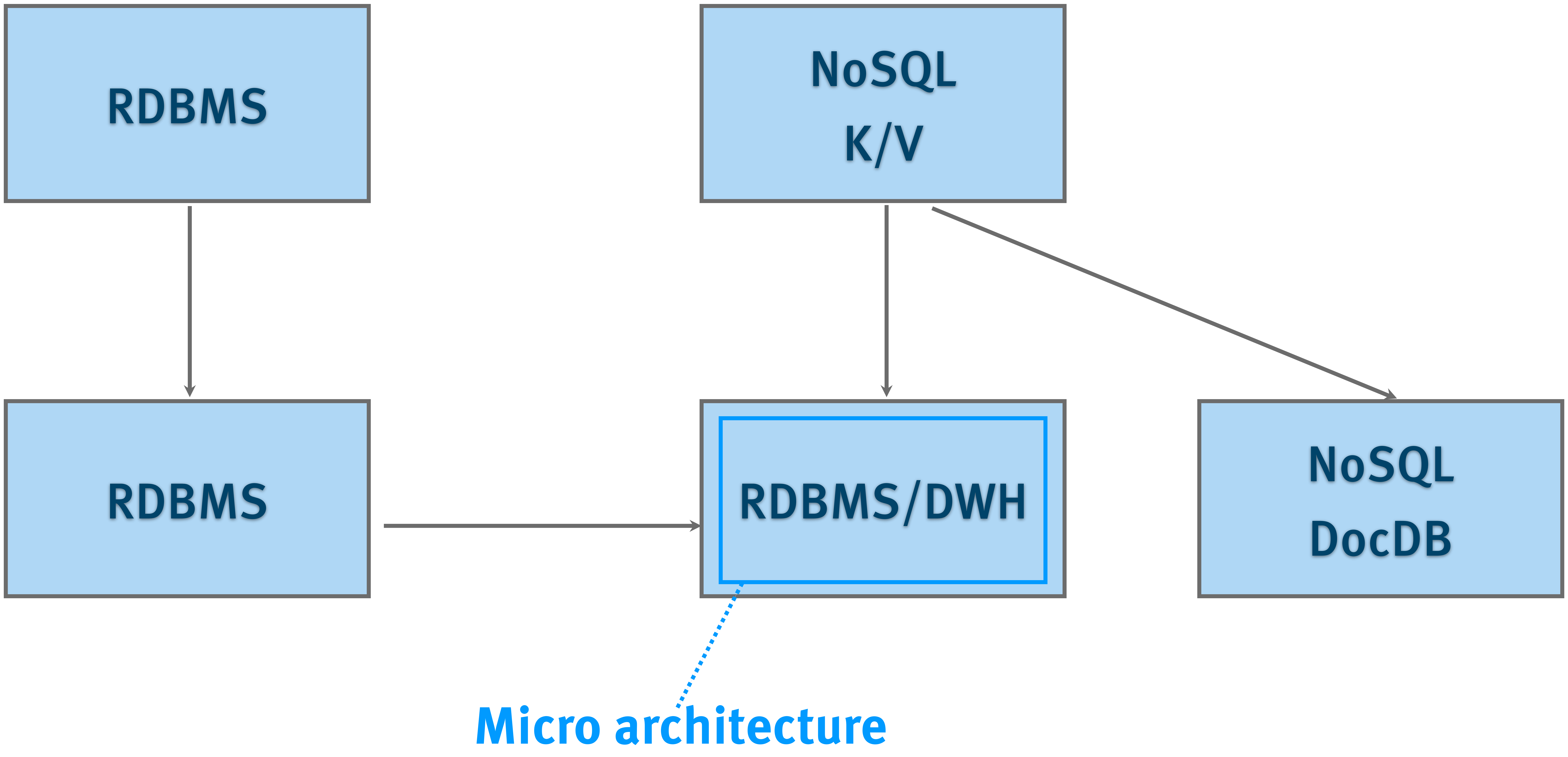
Autonomous deployment and operations

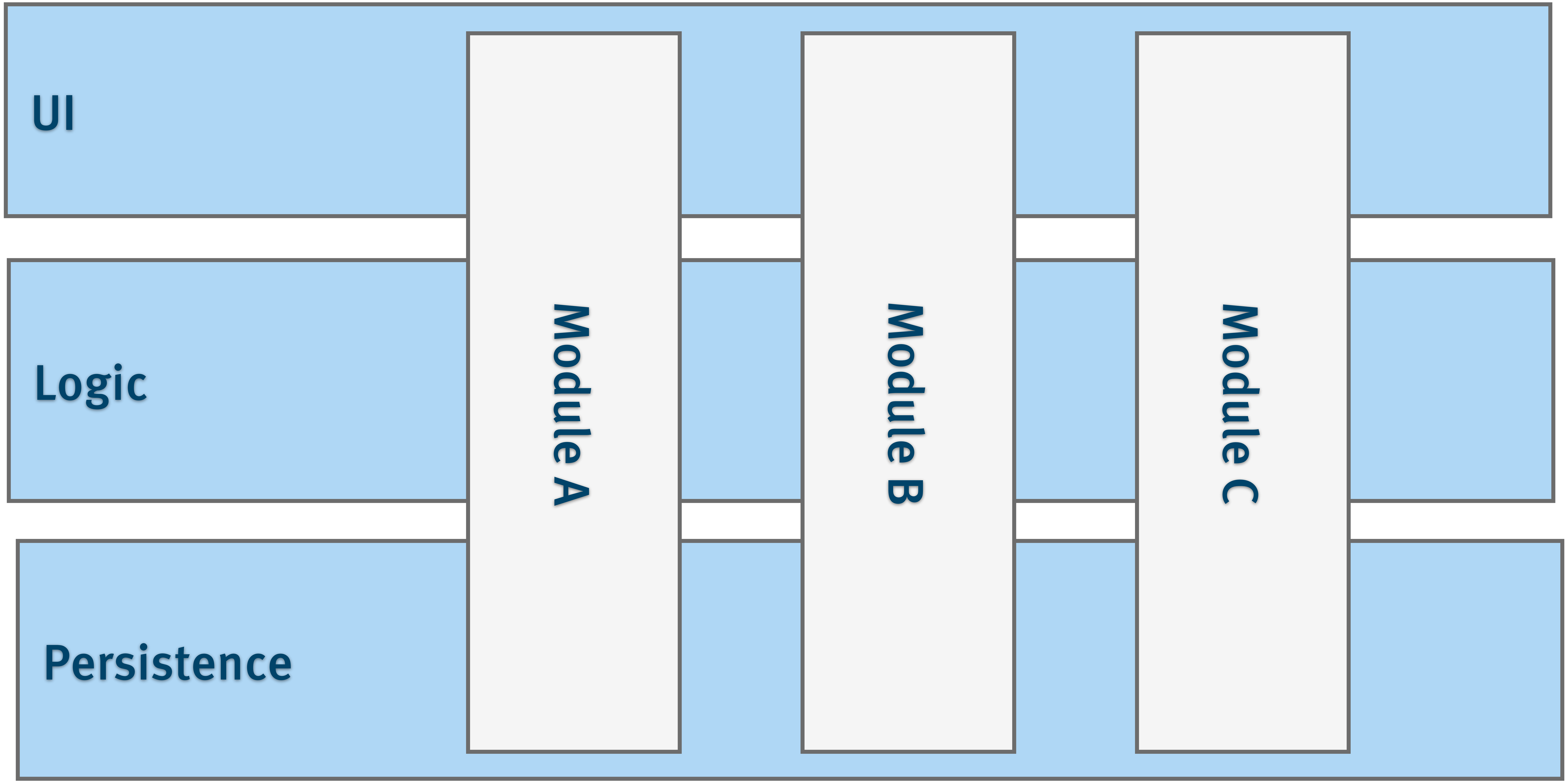


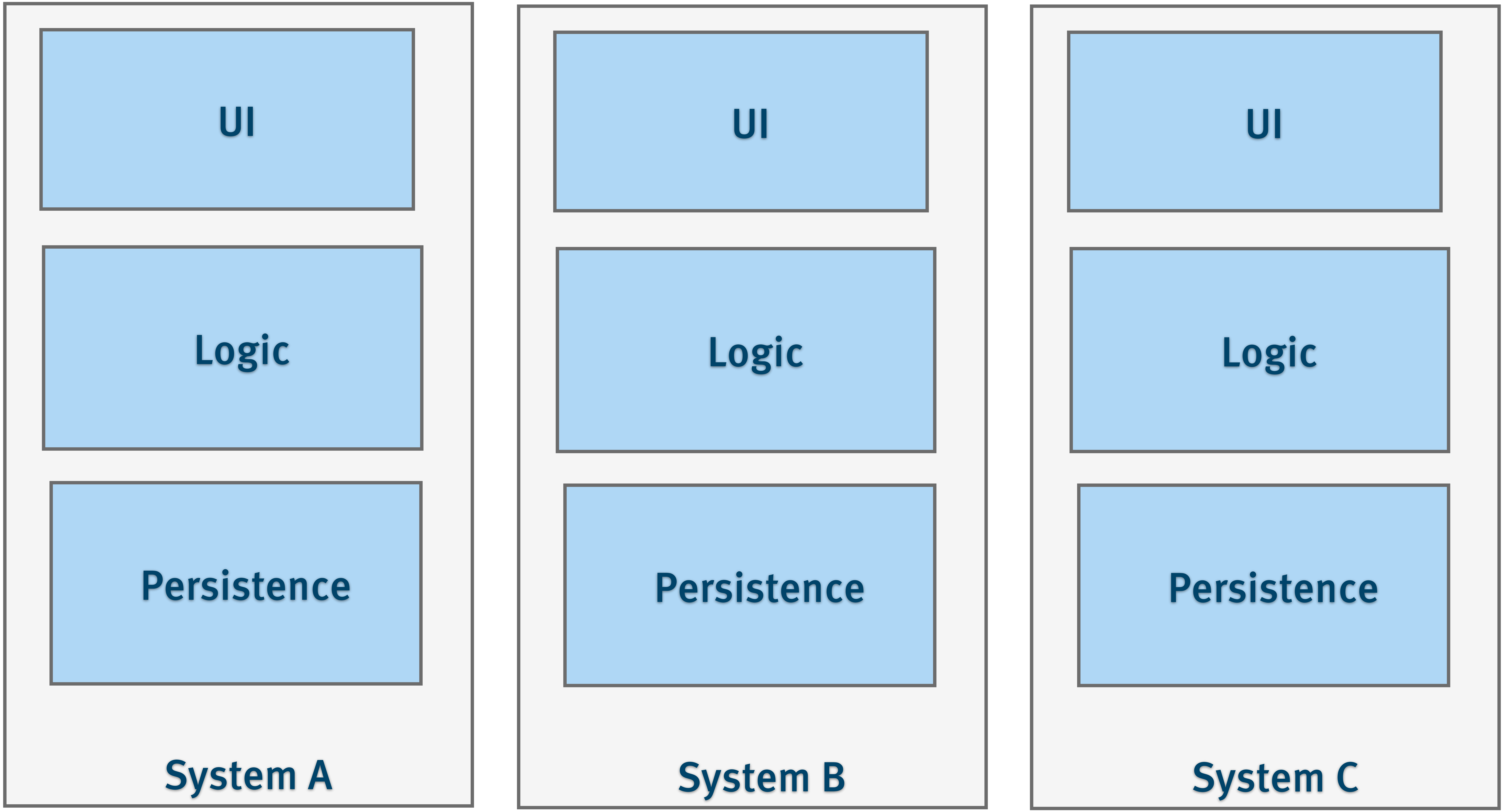
Domain architecture
Macro (technical) architecture


JRuby
C#
Scala
Groovy
Java
Clojure


RDBMS
NoSQL
K/V
RDBMS
RDBMS/DWH
NoSQL
DocDB


RDBMS
NoSQL
K/V
RDBMS
RDBMS/DWH
NoSQL
DocDB
Micro architecture


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

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)	?	?

But why?

Isolation

(Independent) Scalability

Localized decisions

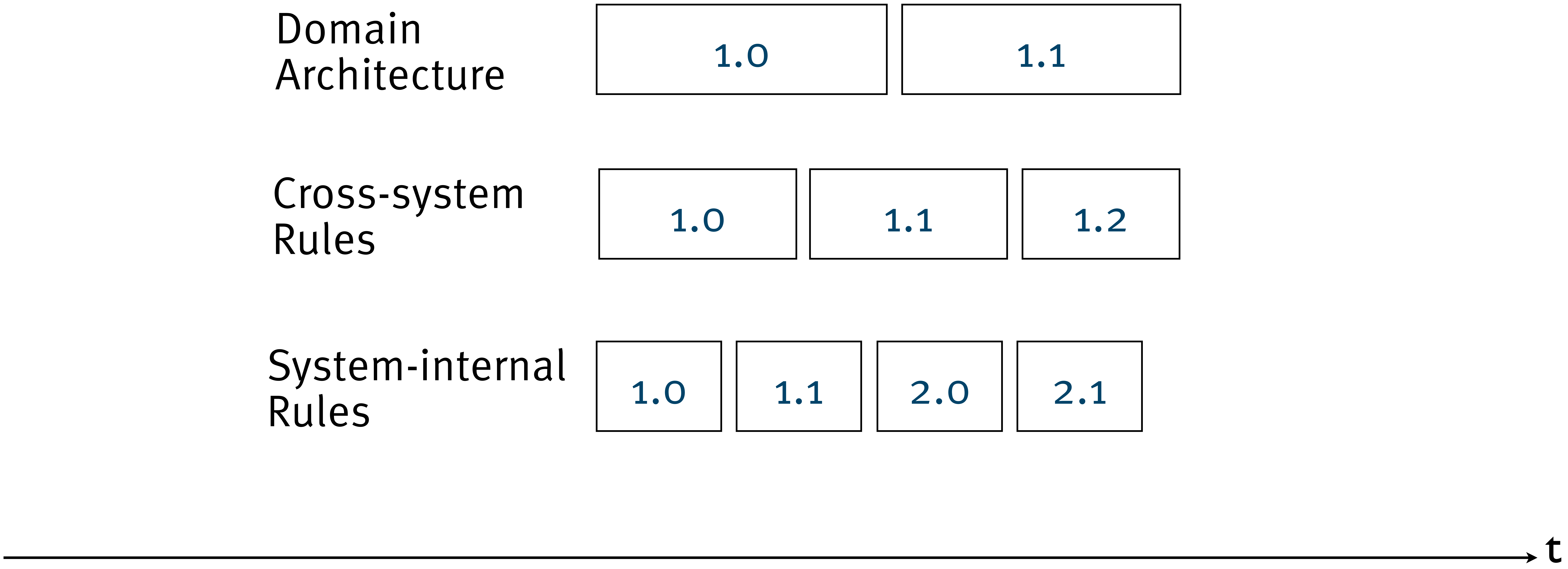
Replaceability

Playground effect

Afraid of chaos?

Necessary Rules & Guidelines

Cross-system	**System-internal**
Responsibilities	Programming languages
UI integration	Development tools
Communication protocols	Frameworks
Data formats	Process/Workflow control
Redundant data	Persistence
BI interfaces	Design patterns
Logging, Monitoring	Coding guidelines



Domain Architecture
1.0
1.1
Cross-system Rules
1.0
1.1
1.2
System-internal Rules
1.0
1.1
2.0
2.1
t

Initial goals

Simplicity

Speed

Easy development

Maximum productivity

Long-term goals

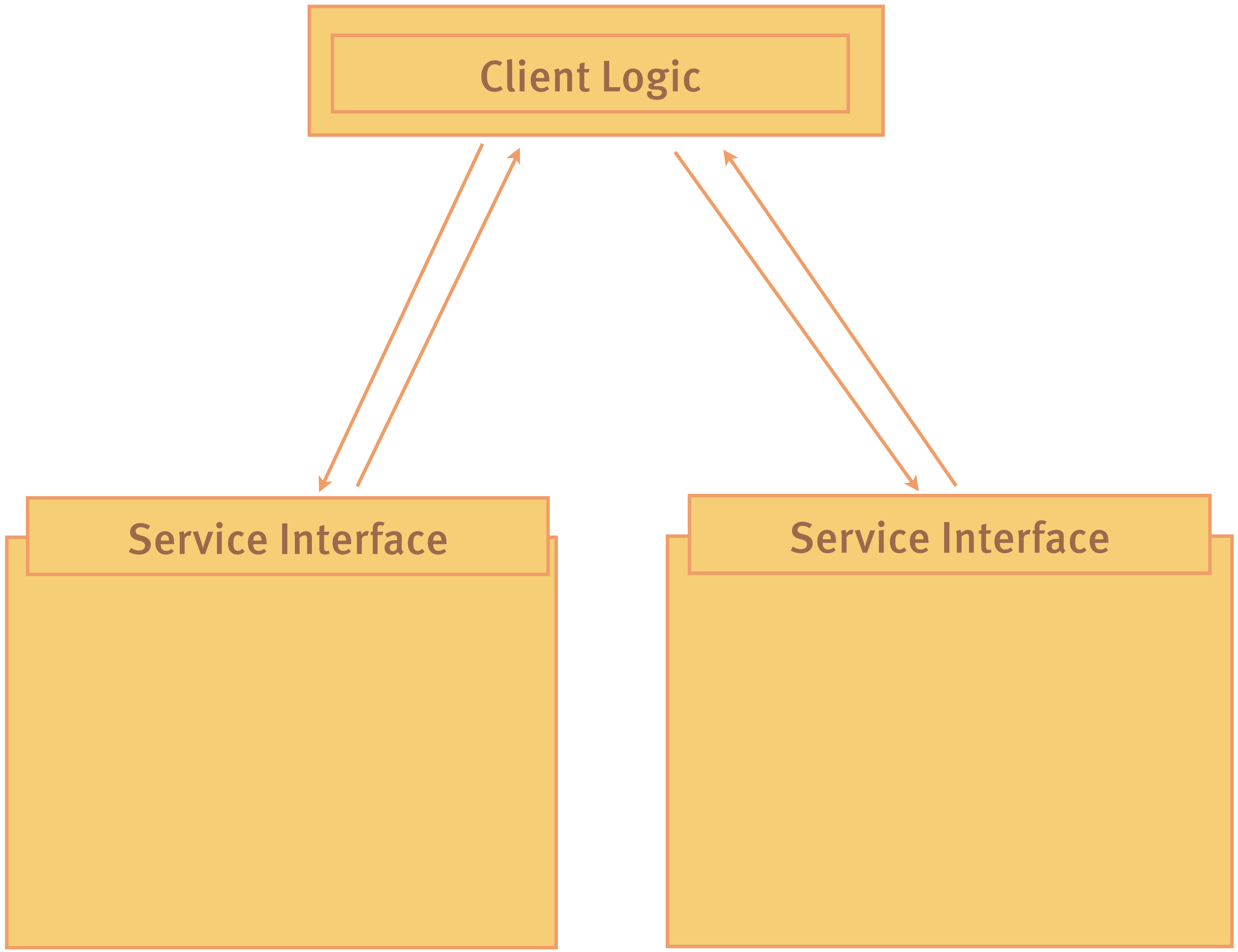
Stability

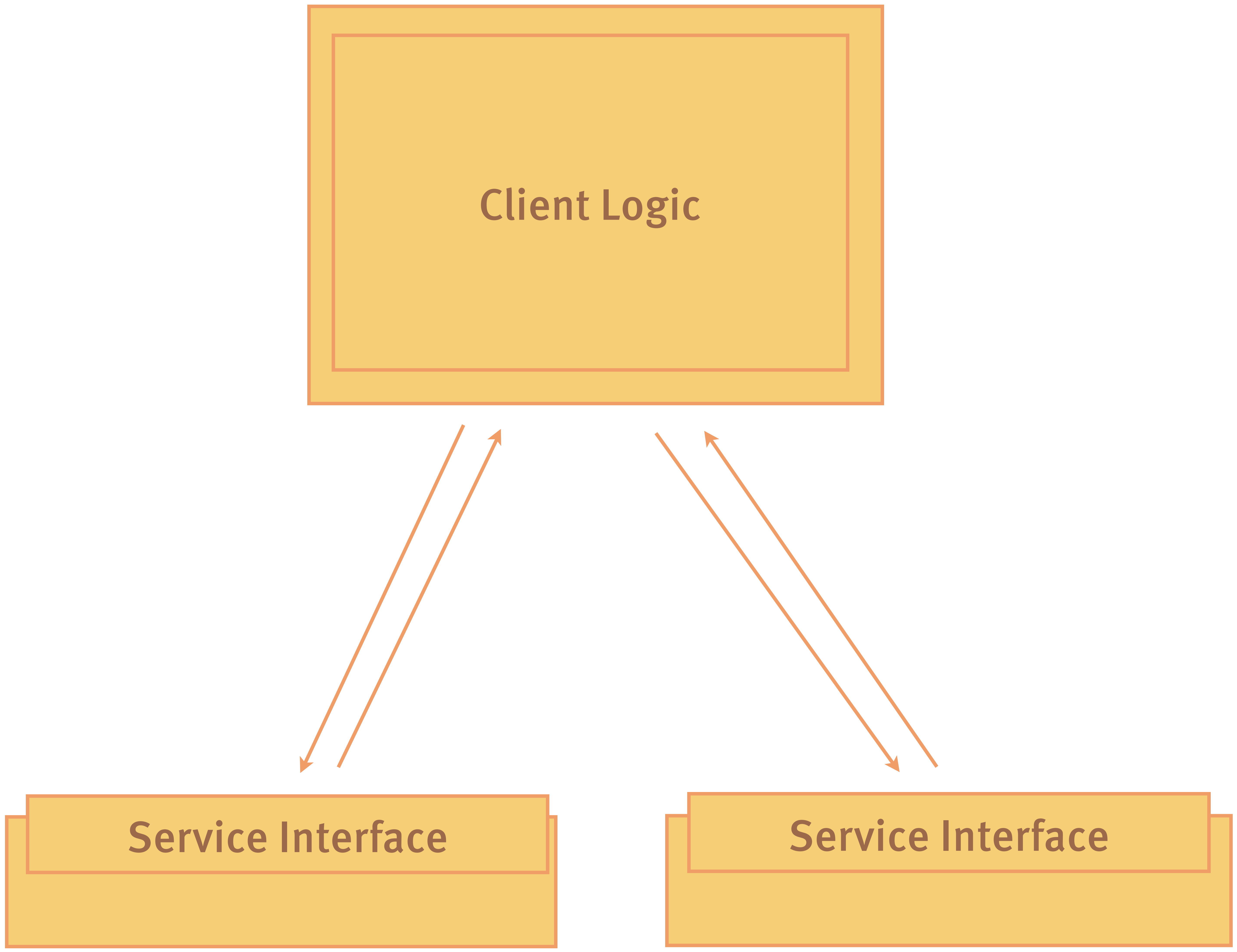
Scalability

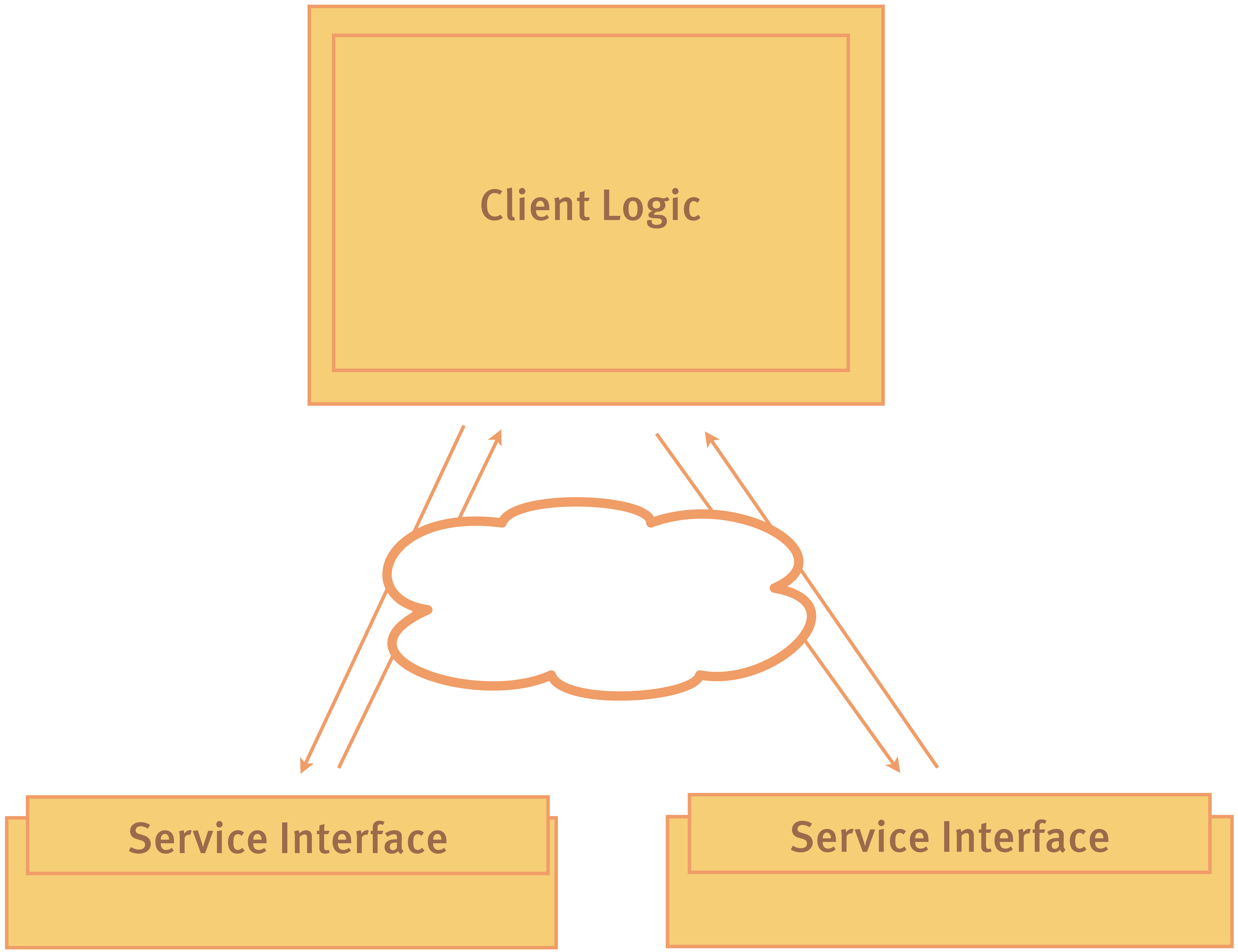
Maintainability

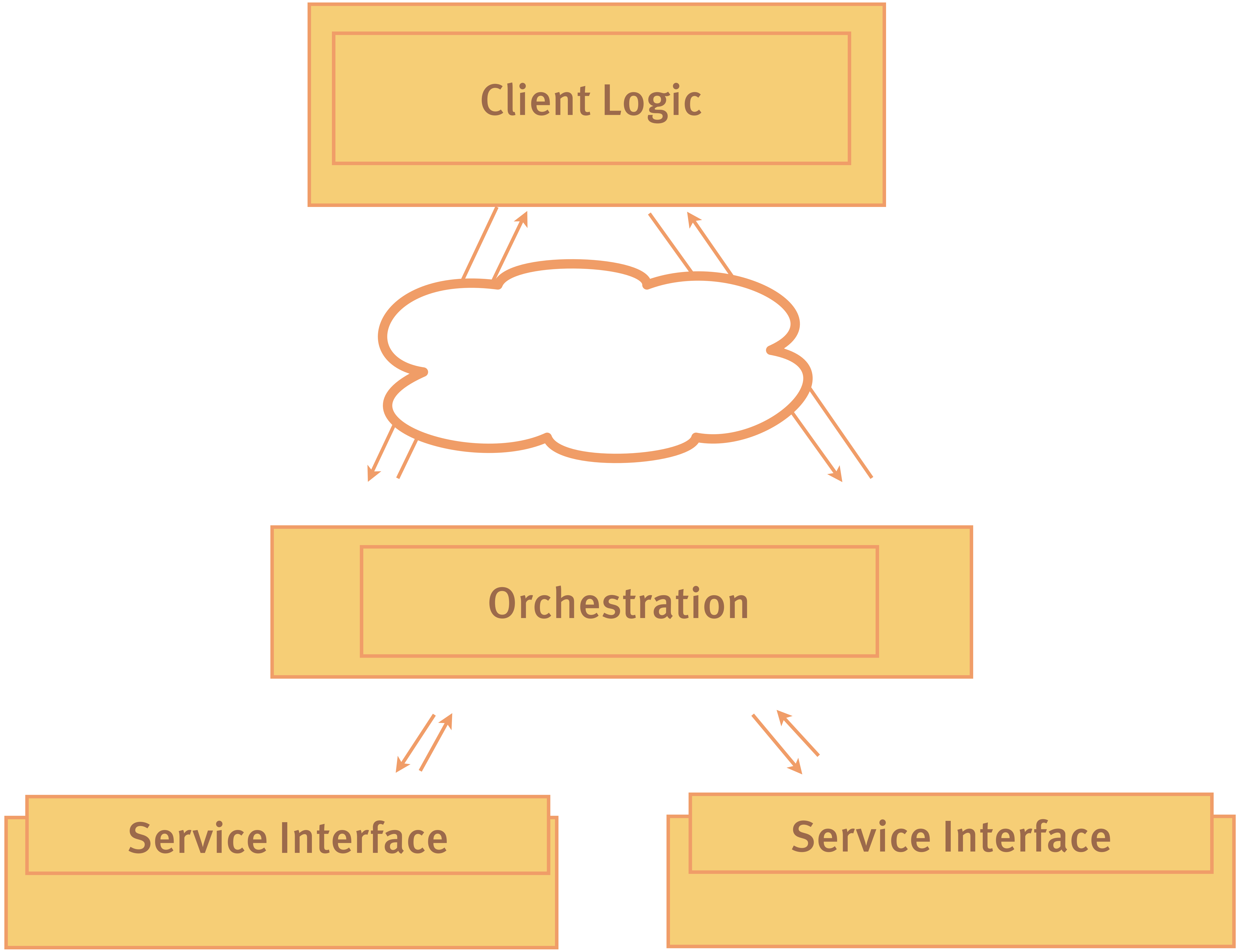
Decoupling

4. … putting pieces together

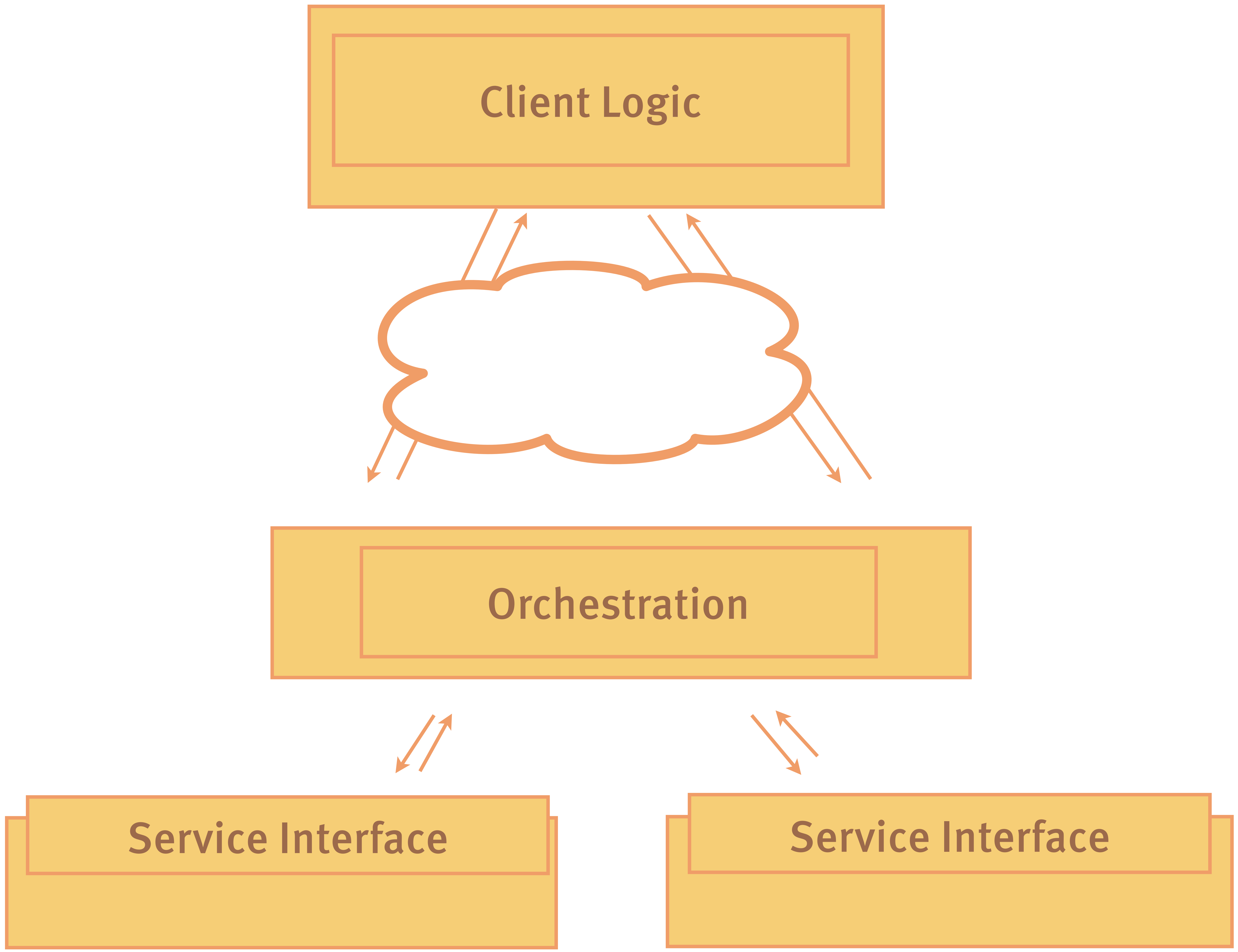

Client Logic
Service Interface
Service Interface

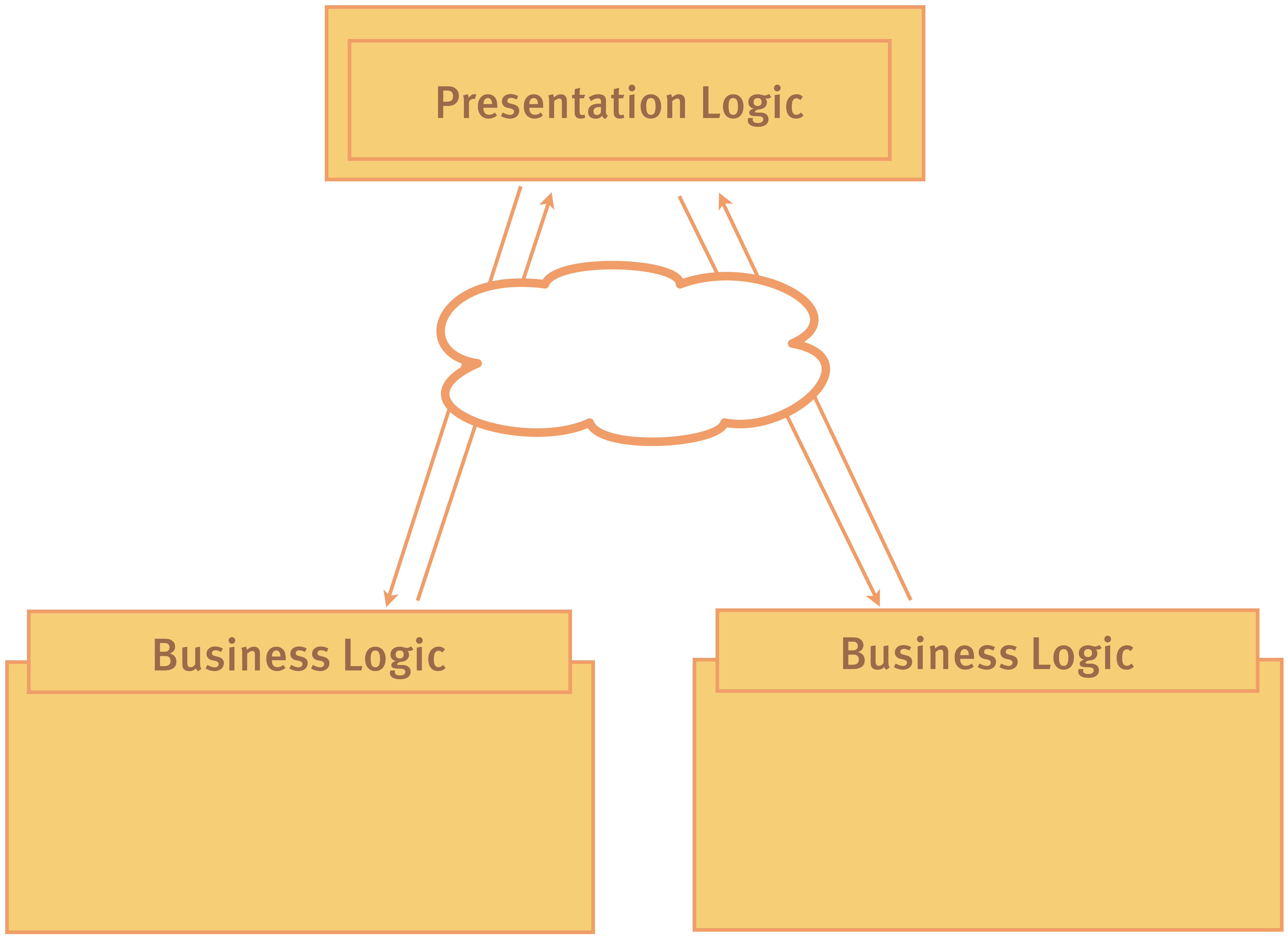

Client Logic
Service Interface
Service Interface

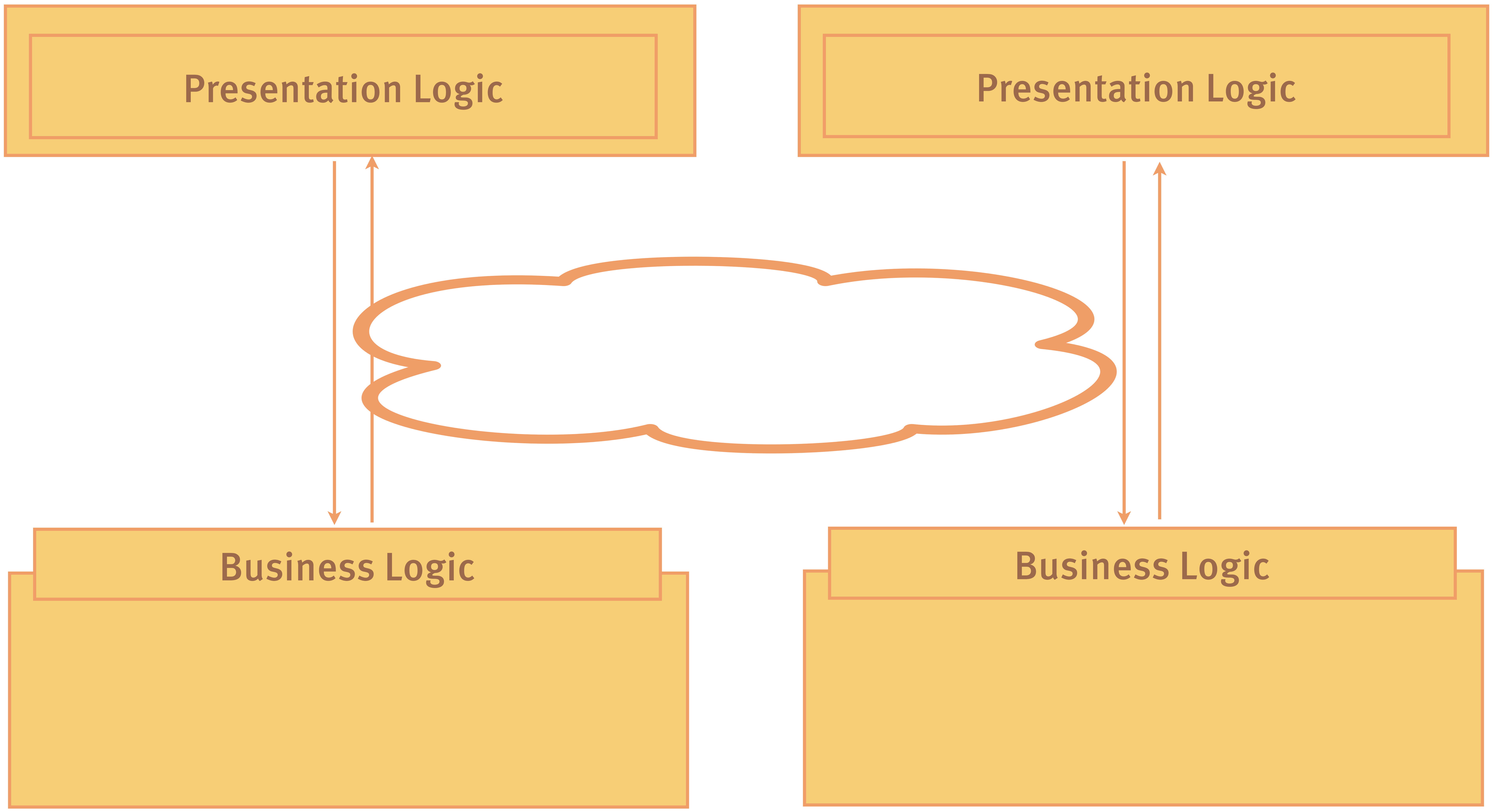

Client Logic
Service Interface
Service Interface



Client Logic
Orchestration
Service Interface
Service Interface


Client Logic
Orchestration
Service Interface
Service Interface

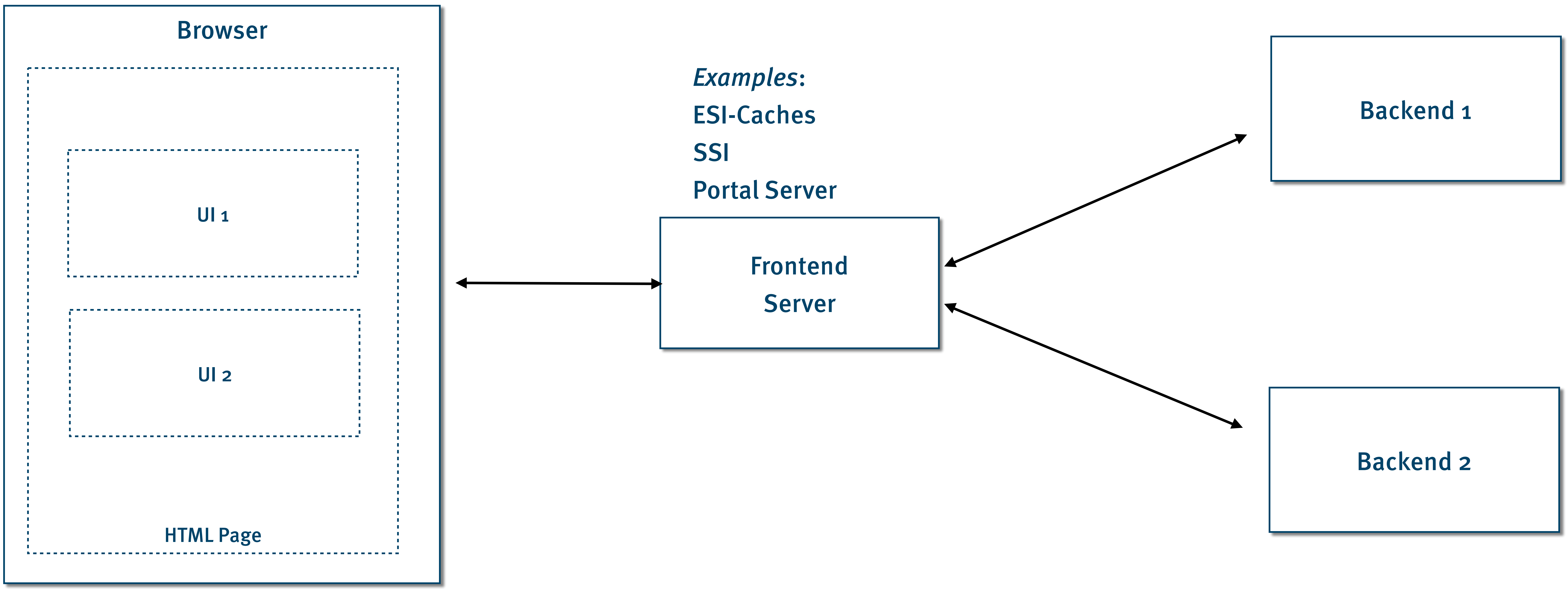

Presentation Logic
Business Logic
Business Logic



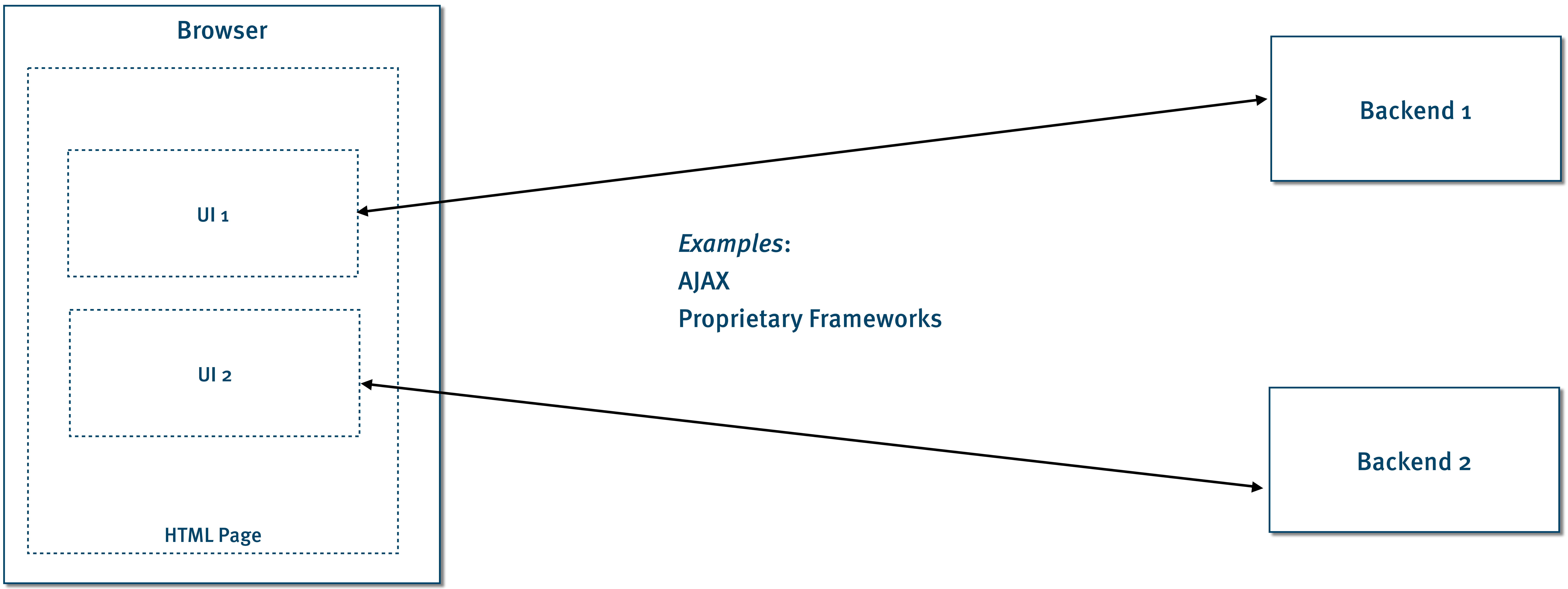
Presentation Logic
Presentation Logic
Business Logic
Business Logic

Web-native front-end integration

Server-side integration

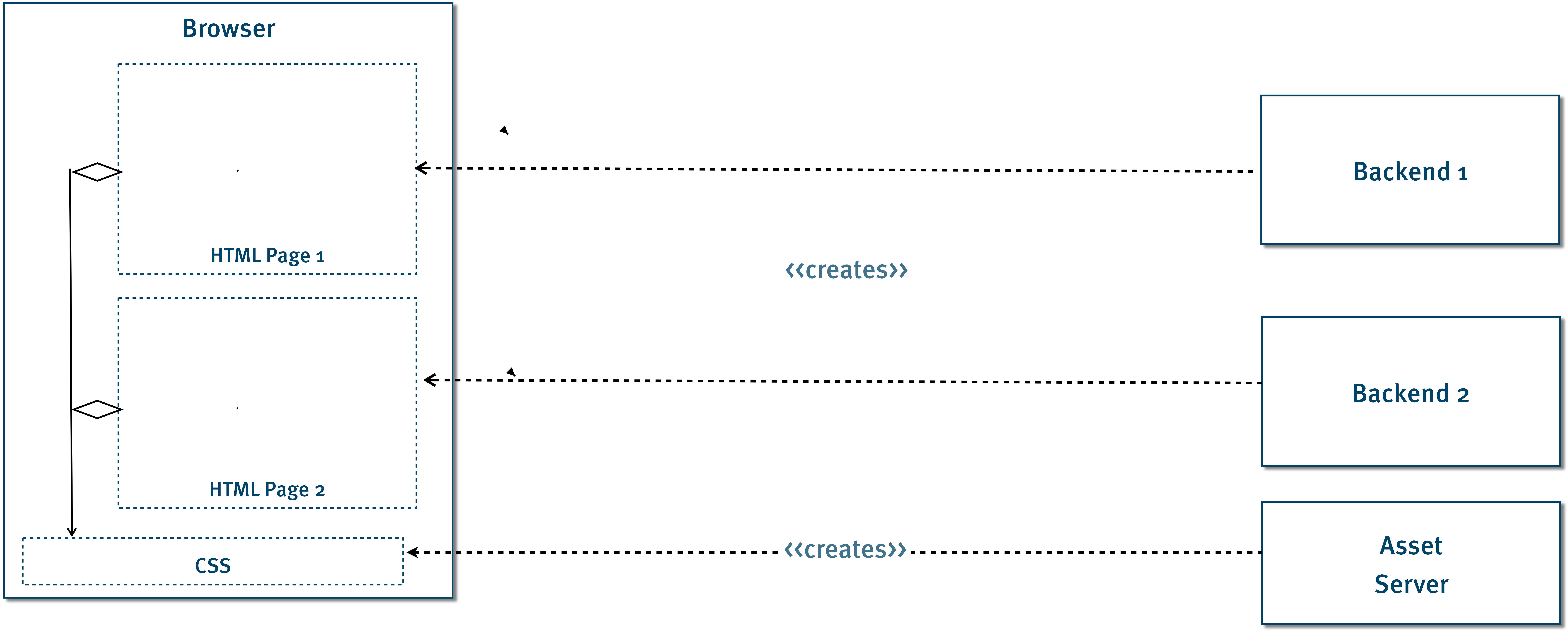


Client-side integration



Examples:
AJAX
Proprietary Frameworks

Links


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

Server-side integration options

Edge integration	ESI, Homegrown, (Portal server)
Backend call	RMI, RPC, REST, WS-*
Storage	DB replication, Feeds
Deployment	Build tools, Chef, Puppet, …, Asset pipeline
Development	Git/SVN submodules, Gems, Maven artifacts

Client-side integration options

Client call
- SPA-style
- JS Widgets

Replaced link
- Unobtrusive JS
- oEmbed
- ROCA-style

Link
- Magical integration concept

5. Challenges

Organization

Architecture Governance

Cross-system

Responsibilities

UI integration

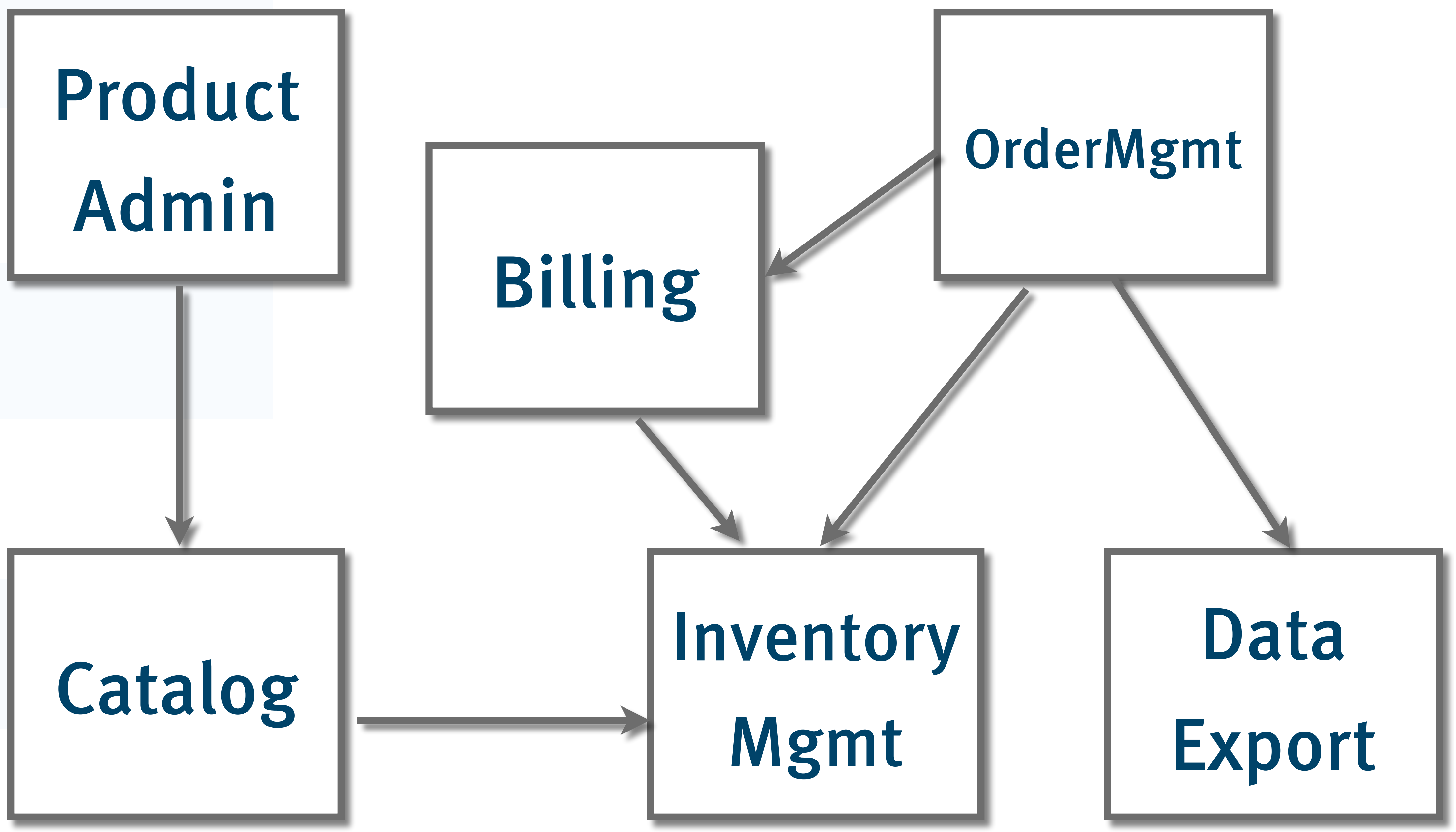
Communication protocols

Data formats

Redundant data

BI interfaces

Logging, Monitoring



Surprise: There ***is*** *a justification for someone to take care of the overall architecture*

Operations

System characteristics

Separate (redundant) persistence

Internal, separate logic

Domain models & implementation strategies

Separate UI

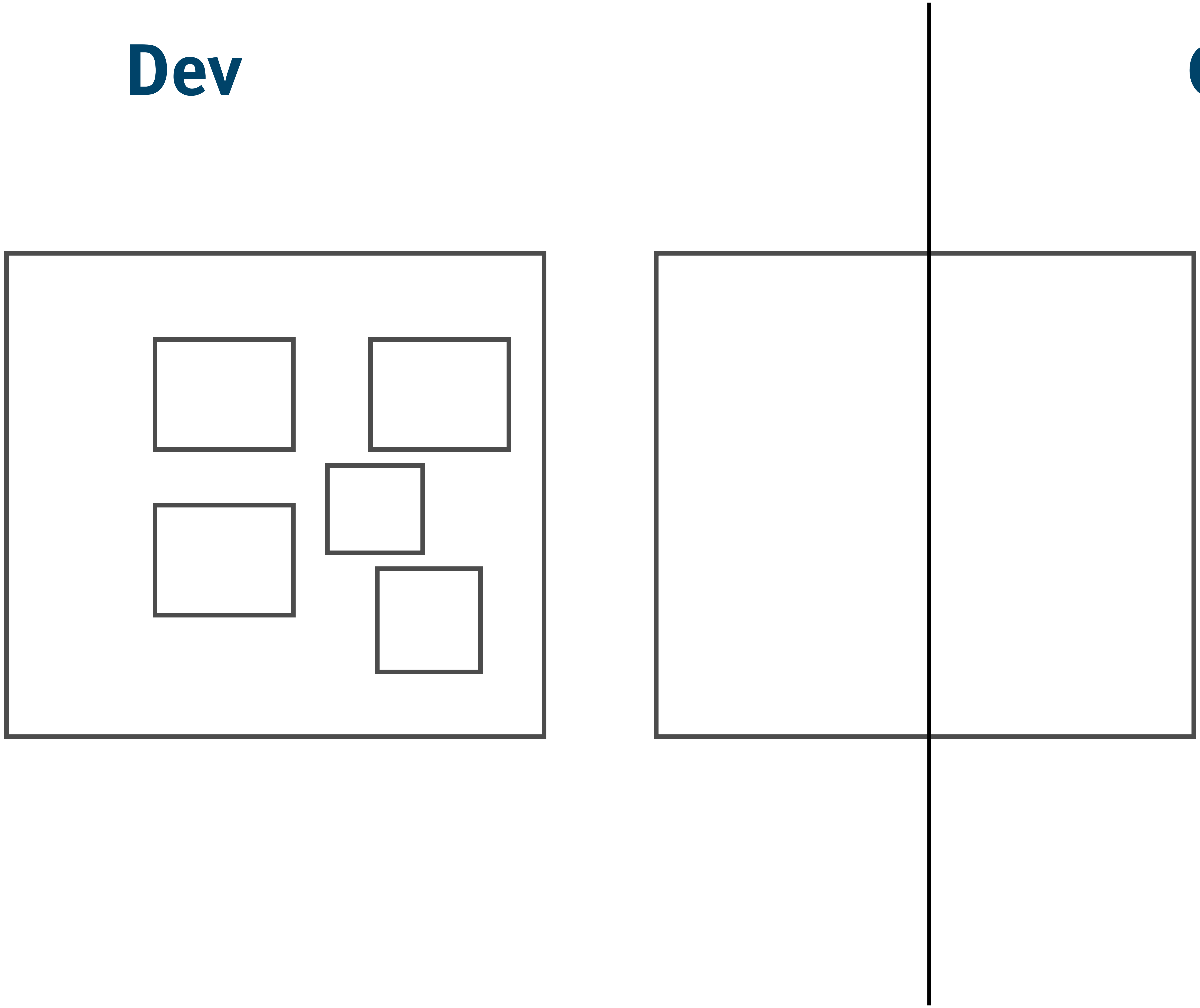
Separate development & evolution

Limited interaction with other systems

Autonomous deployment and operations

Dev

Ops



If systems are really separate, they need to be so from start to finish

Migration

Assumptions

High business value

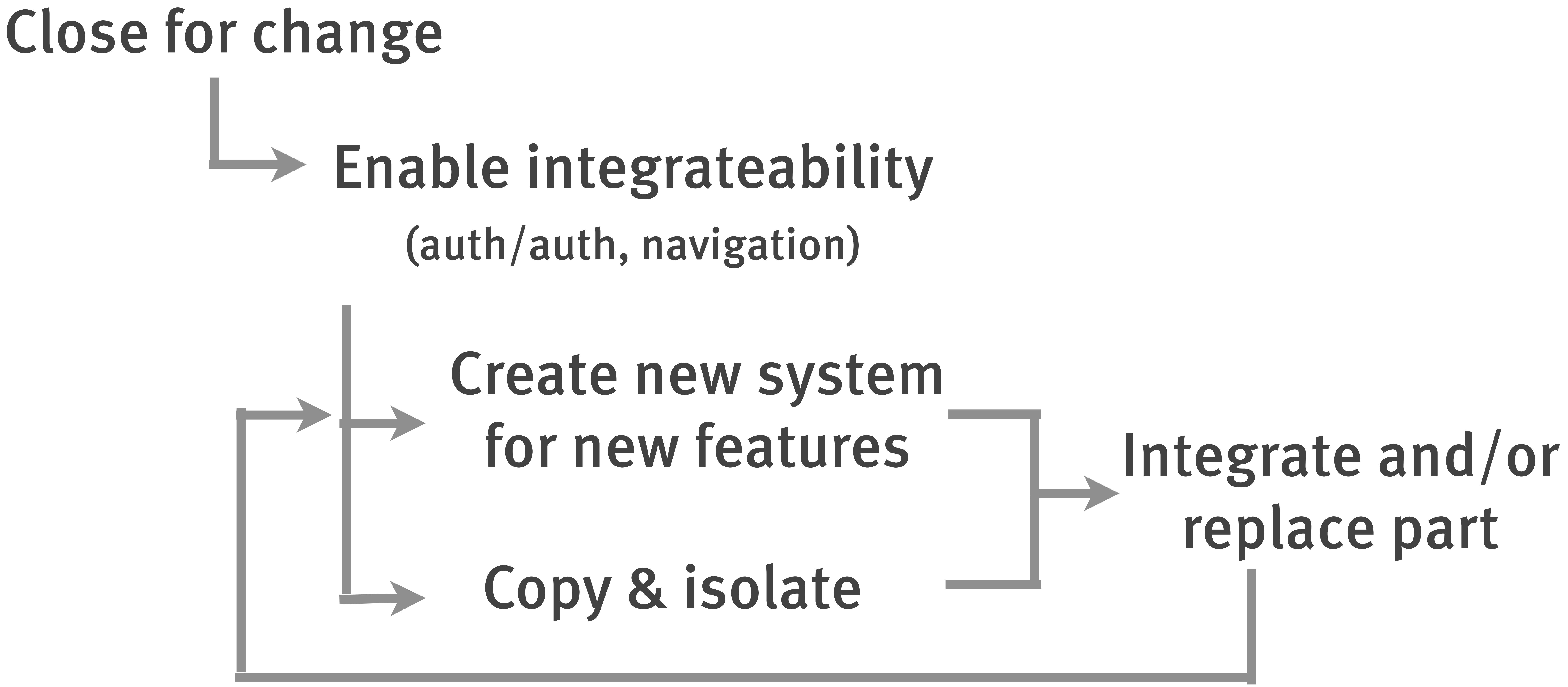
Very high cost of change

Very slow “time to market”

Huge backlog of feature requests

Problem awareness

Strong management support



more patterns at http://aim42.org

Summary

Explicitly design system boundaries

Modularize into independent, self-contained systems

Separate micro and macro architectures

Be aware of changing quality goals

Strike a balance between control and decentralization

Thank you!

Questions?

Comments?

Stefan Tilkov, @stilkov

stefan.tilkov@innoq.com

http://www.innoq.com/blog/st/

Phone: +49 170 471 2625

innoQ

www.innoq.com

innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim am Rhein
Germany
Phone: +49 2173 3366-0

Ohlauer Straße 43
10999 Berlin
Germany
Phone: +49 2173 3366-0

Robert-Bosch-Straße 7
64293 Darmstadt
Germany
Phone: +49 2173 3366-0

Radlkoferstraße 2
D-81373 München
Germany
Telefon +49 (0) 89 741185-270

innoQ Schweiz GmbH

Gewerbestr. 11
CH-6330 Cham
Switzerland
Phone: +41 41 743 0116