innoQ

MicroServices

a practical overview

Martin Eigenbrodt | martin.eigenbrodt@innoq.com

Alexander Heusingfeld | alexander.heusingfeld@innoq.com

innoQ

Reviewing architectures

Generic Architecture Review Results

Generic Architecture Review Results

Building features
takes too long

Generic Architecture Review Results

Building features takes too long

Technical debt is well-known and not addressed

Generic Architecture Review Results

Building features takes too long

Technical debt is well-known and not addressed

Deployment is way too complicated and slow

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

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

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

“-ility” problems abound

Scalability has reached its limit

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

“-ility” problems abound

Scalability has reached its limit

Replacement would be way too expensive

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

— Stefan Tilkov

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**

System boundaries


Project scope

Project scope

1 Project = 1 System?

Size

Modularization

Size	Modularization
1-50 LOC	single file

Size	Modularization
1-50 LOC	single file
50-500 LOC	few files, few functions

Size	Modularization
1-50 LOC	single file
50-500 LOC	few files, few functions
500-1000 LOC	Library, class hierarchy

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

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

System Characteristics

Separate (redundant) persistence

System Characteristics

Separate (redundant) persistence

Internal, separate logic

System Characteristics

Separate (redundant) persistence

Internal, separate logic

Domain models & implementation strategies

System Characteristics

Separate (redundant) persistence

Internal, separate logic

Domain models & implementation strategies

Separate UI

System Characteristics

Separate (redundant) persistence

Internal, separate logic

Domain models & implementation strategies

Separate UI

Separate development & evolution

System Characteristics

Separate (redundant) persistence

Internal, separate logic

Domain models & implementation strategies

Separate UI

Separate development & evolution

Limited interaction with other systems

System Characteristics

Separate (redundant) persistence

Internal, separate logic

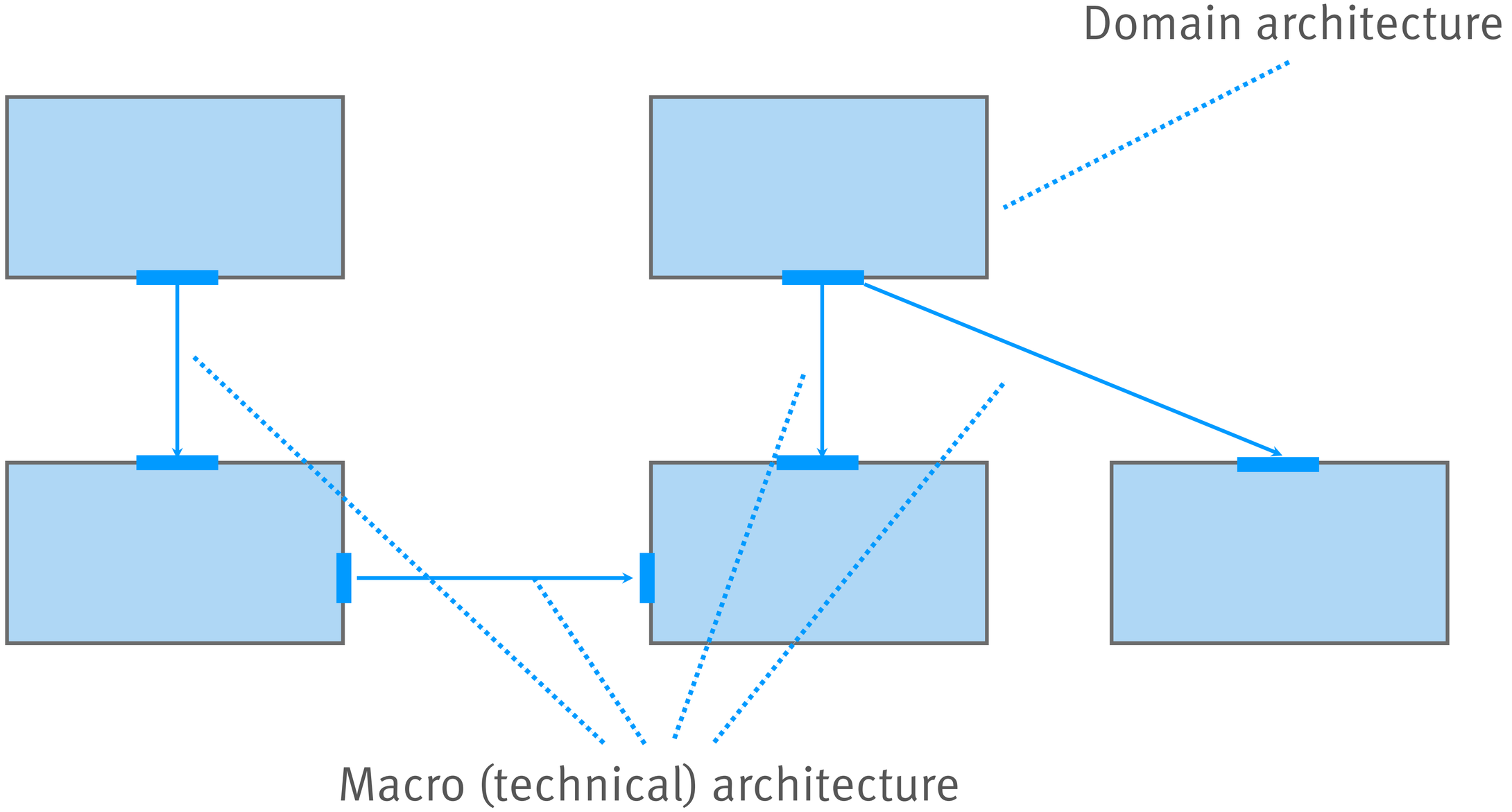
Domain models & implementation strategies

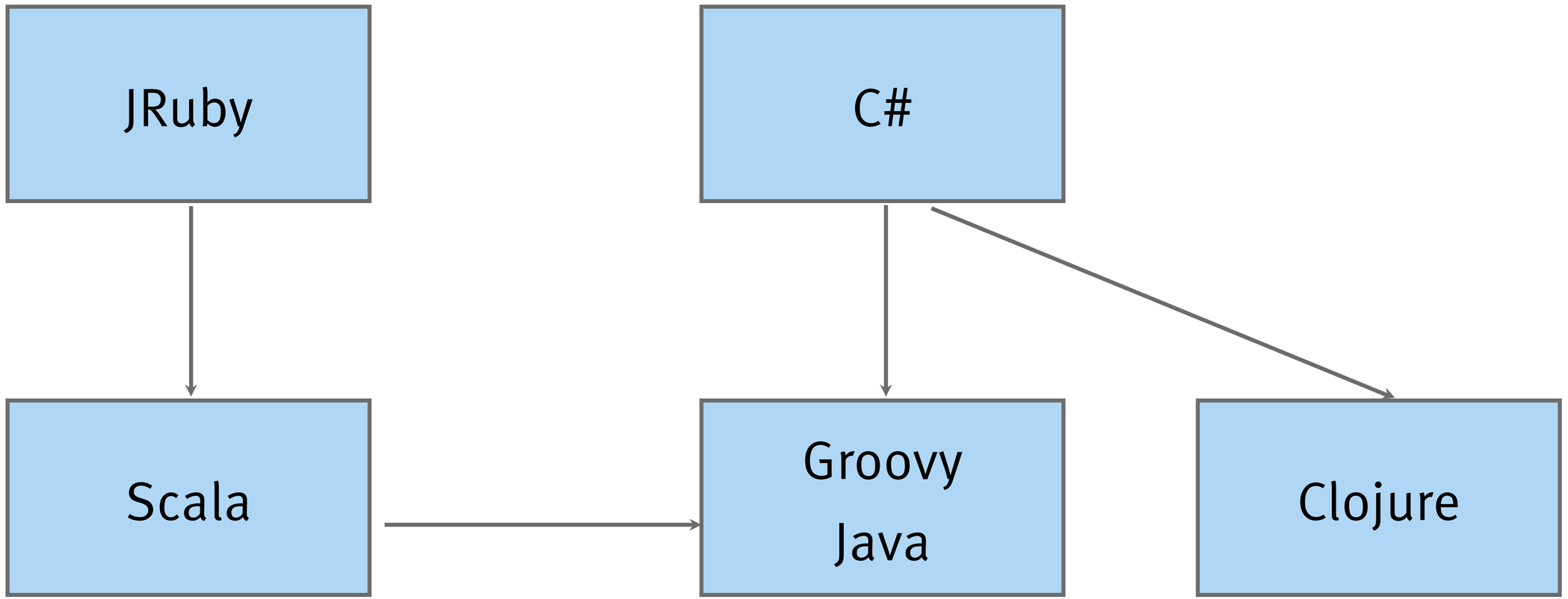
Separate UI

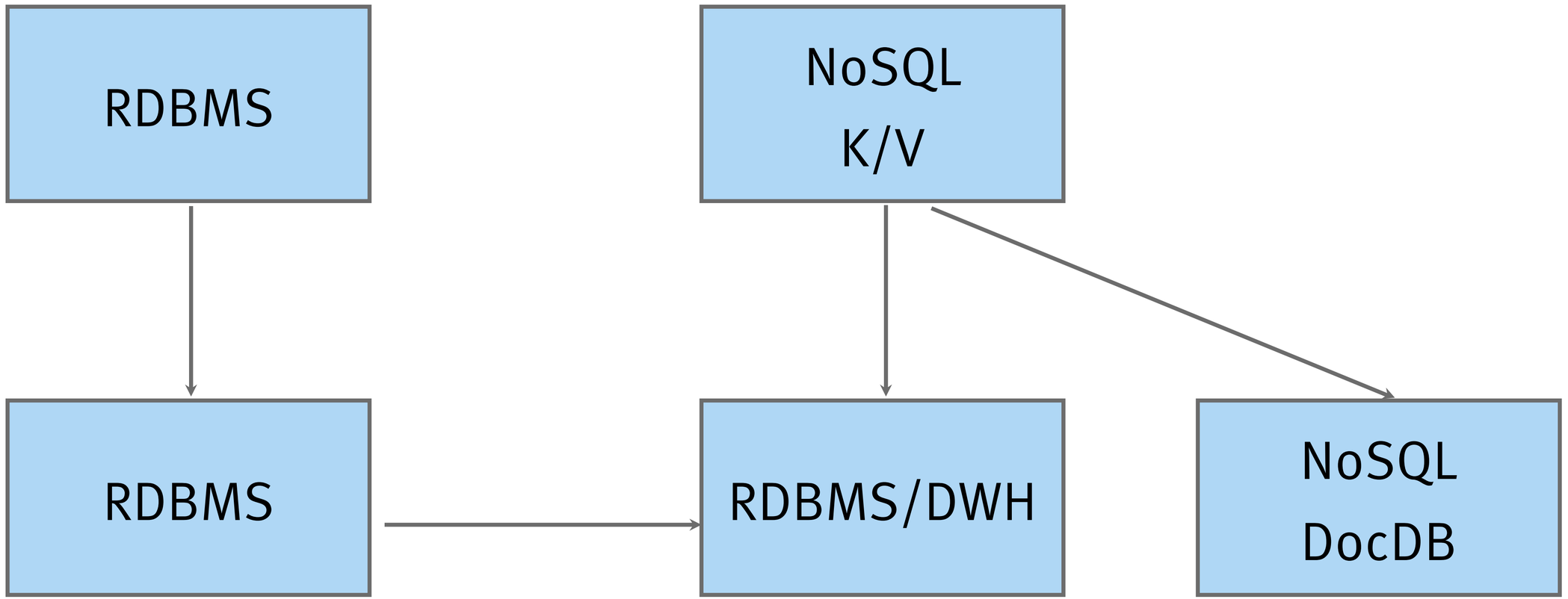
Separate development & evolution

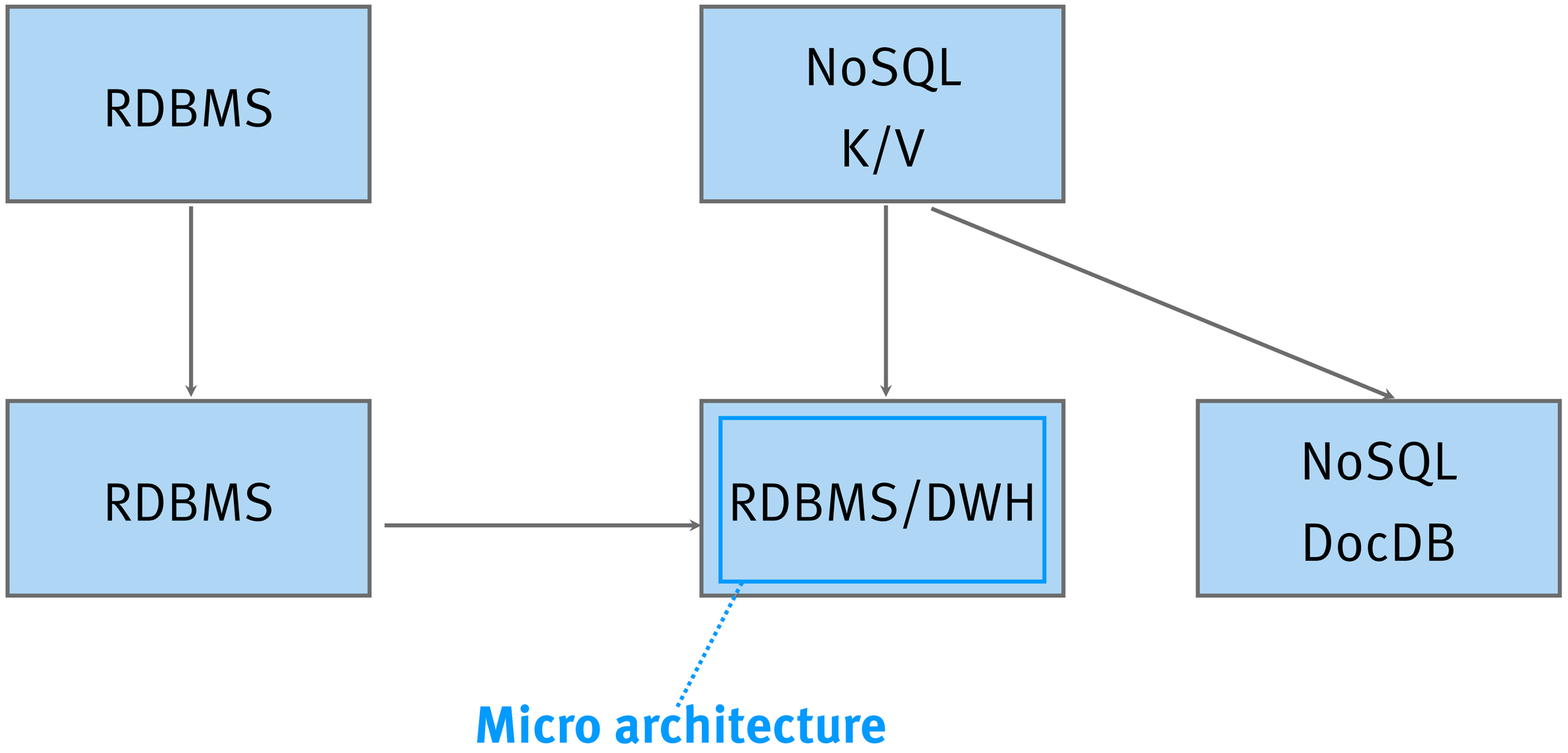
Limited interaction with other systems

Autonomous deployment and operations

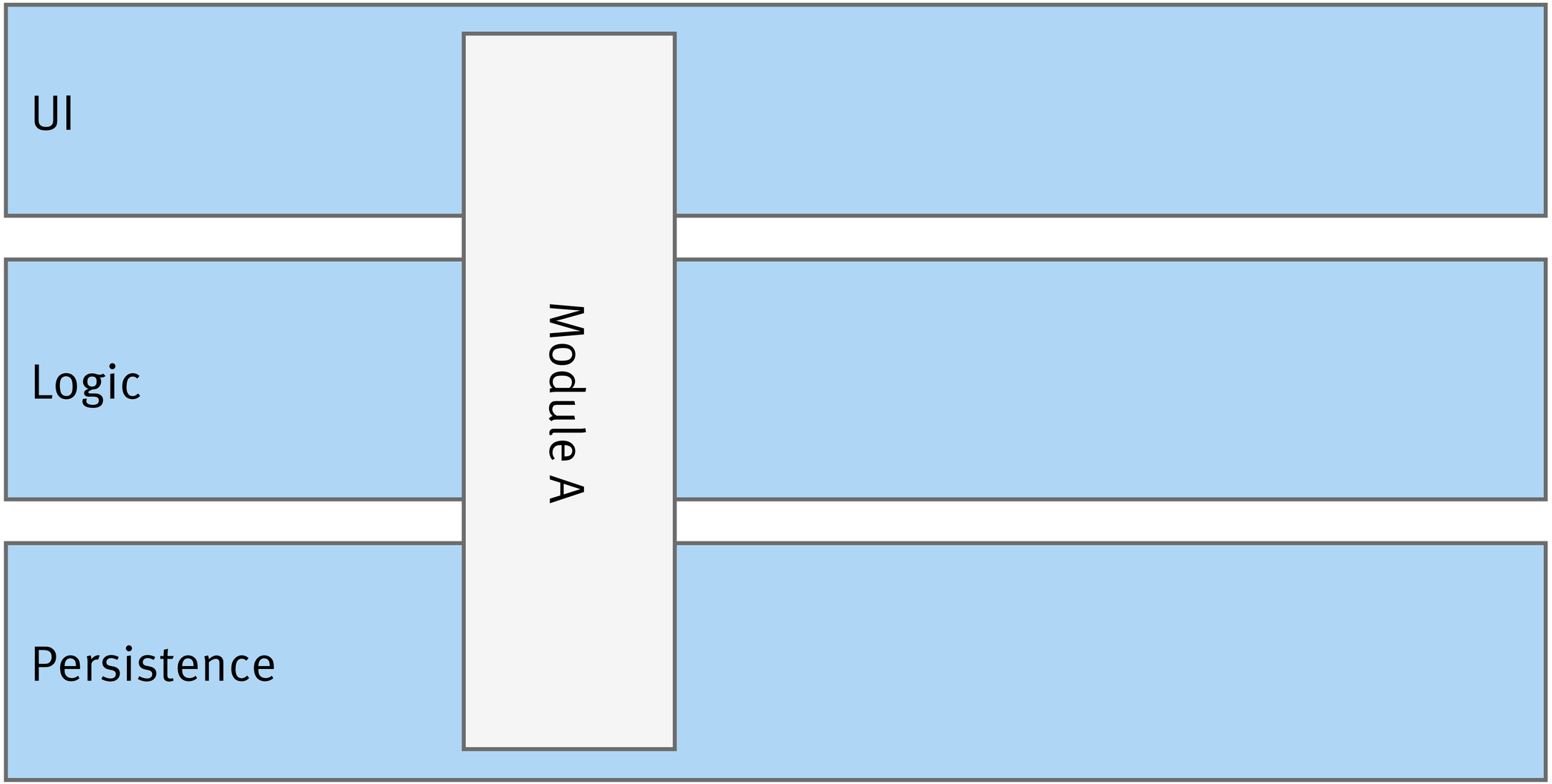

Domain architecture
Macro (technical) architecture

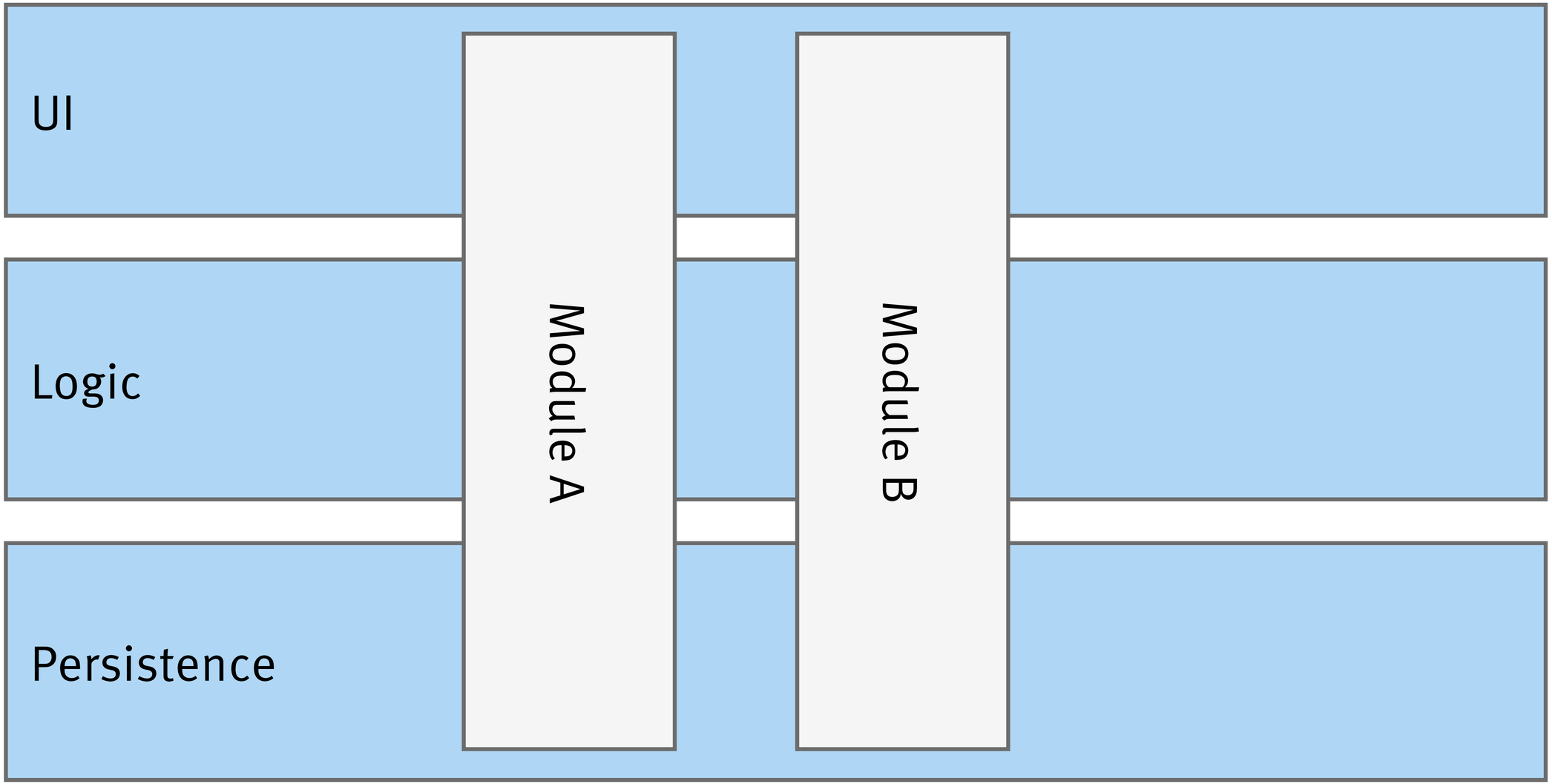

JRuby
C#
Scala
Groovy
Java
Clojure

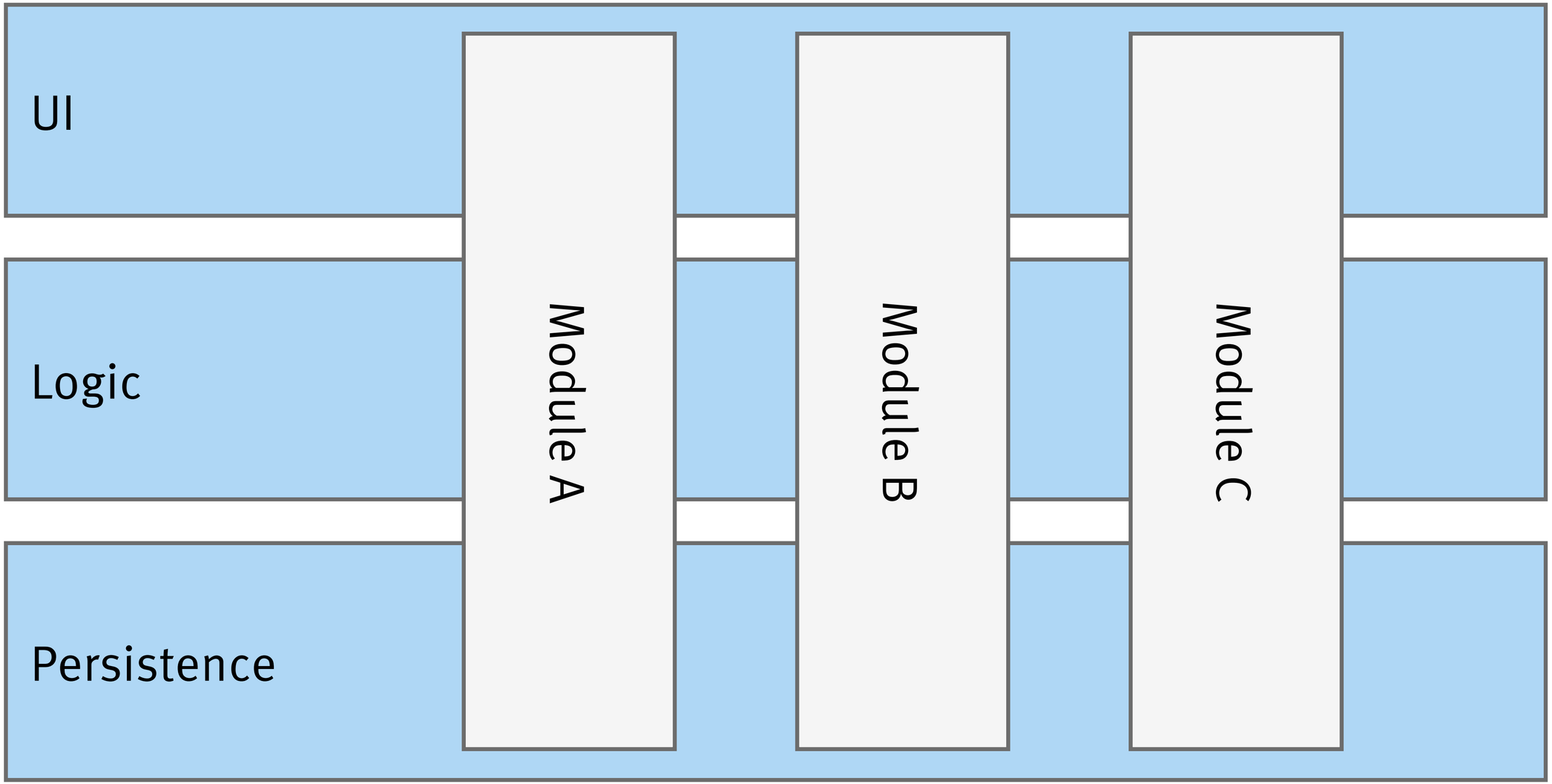

RDBMS
NoSQL
K/V
RDBMS
RDBMS/DWH
NoSQL
DocDB

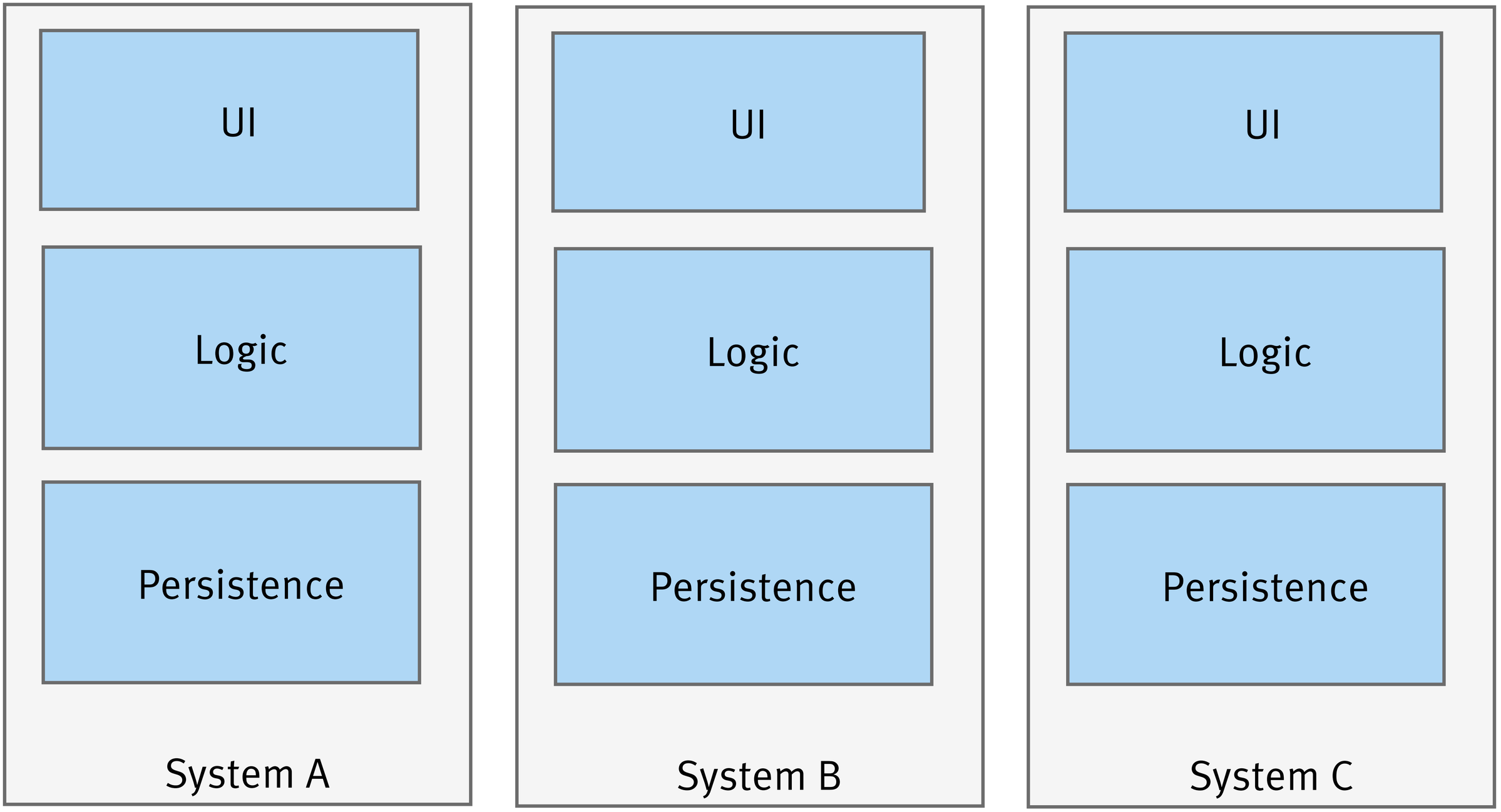

RDBMS
NoSQL
K/V
RDBMS
RDBMS/DWH
NoSQL
DocDB
Micro architecture


UI
Logic
Persistence


UI
Logic
Persistence
Module A


UI
Logic
Persistence
Module A
Module B


UI
Logic
Persistence
Module A
Module B
Module C


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

innoQ

Are this MicroServices?

MicroService Characteristics

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

MicroService Characteristics

small

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

MicroService Characteristics

small

each running in its own process

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

MicroService Characteristics

small

each running in its own process

lightweight communicating mechanisms (often HTTP)

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

MicroService Characteristics

small

each running in its own process

lightweight communicating mechanisms (often HTTP)

built around business capabilities

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

MicroService Characteristics

small

each running in its own process

lightweight communicating mechanisms (often HTTP)

built around business capabilities

independently deployable

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

MicroService Characteristics

small

each running in its own process

lightweight communicating mechanisms (often HTTP)

built around business capabilities

independently deployable

mininum of centralized management

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

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

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

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

Self-Contained System (SCS)

SCS Characteristics

SCS Characteristics

Autonomous web application

SCS Characteristics

Autonomous web application

Owned by one team

SCS Characteristics

Autonomous web application

Owned by one team

sync remote calls discouraged

SCS Characteristics

Autonomous web application

Owned by one team

sync remote calls discouraged

Service API optional

SCS Characteristics

Autonomous web application

Owned by one team

sync remote calls discouraged

Service API optional

Includes data and logic

SCS Characteristics

Autonomous web application

Owned by one team

sync remote calls discouraged

Service API optional

Includes data and logic

No shared UI

SCS Characteristics

Autonomous web application

Owned by one team

sync remote calls discouraged

Service API optional

Includes data and logic

No shared UI

No or pull-based code sharing only

SCS

Microservice

	SCS	Microservice
Size (kLoC)	1-50	0.1-?

	SCS	Microservice
Size (kLoC)	1-50	0.1-?
State	Self-contained	Self-contained

	SCS	Microservice
Size (kLoC)	1-50	0.1-?
State	Self-contained	Self-contained
# per Logical System	5-25	>100

	SCS	Microservice
Size (kLoC)	1-50	0.1-?
State	Self-contained	Self-contained
# per Logical System	5-25	>100
Communication between units	No (if possible)	Yes

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

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

innoQ

But why?

innoQ

Isolation

(Independent) Scalability

innoQ

Localized decisions

Replaceability

Playground effect

innoQ

Afraid of chaos?

Necessary Rules & Guidelines

Necessary Rules & Guidelines

Cross-system

Responsibilities

UI integration

Communication protocols

Data formats

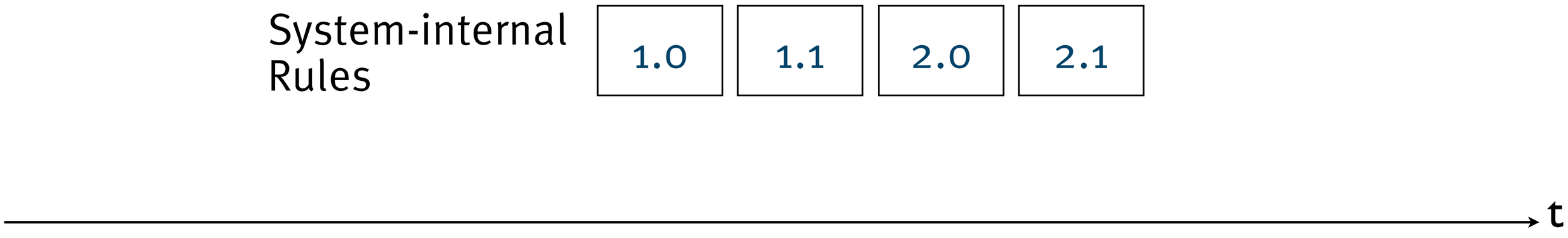
Redundant data

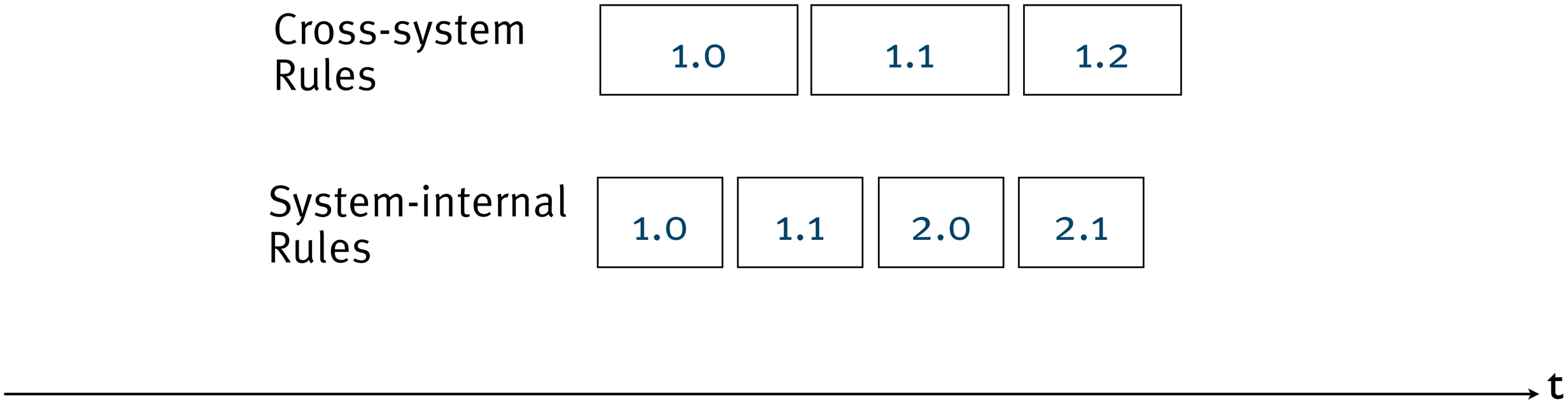
BI interfaces

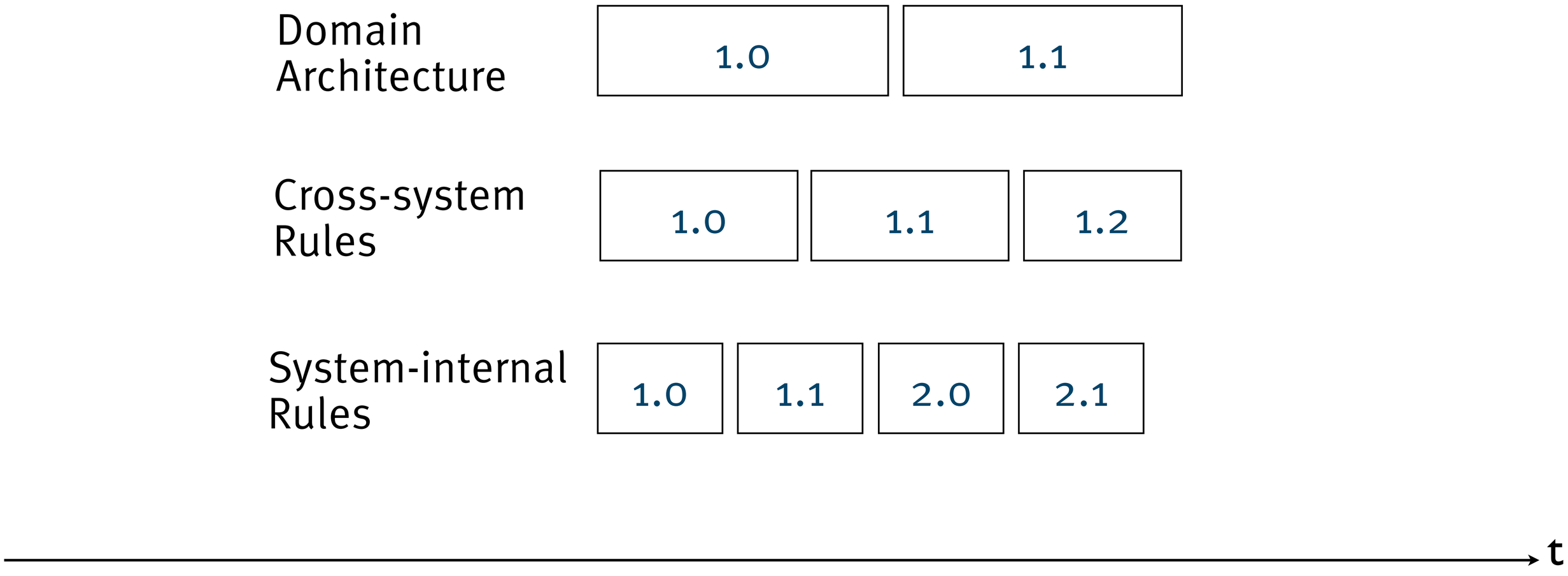
Logging, Monitoring

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


System-internal
Rules
1.0
1.1
2.0
2.1
t


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


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

5. Real-World Examples

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

1. Write Press Release

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

1. Write Press Release

2. Write FAQ

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

1. Write Press Release

2. Write FAQ

3. Describe Customer Experience

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

1. Write Press Release
2. Write FAQ
3. Describe Customer Experience
4. Write User Manual

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

1. Write Press Release

2. Write FAQ

3. Describe Customer Experience

4. Write User Manual

...only then start building it!

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

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

Independent “Verticals”

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

Independent “Verticals”

REST-based macro architecture

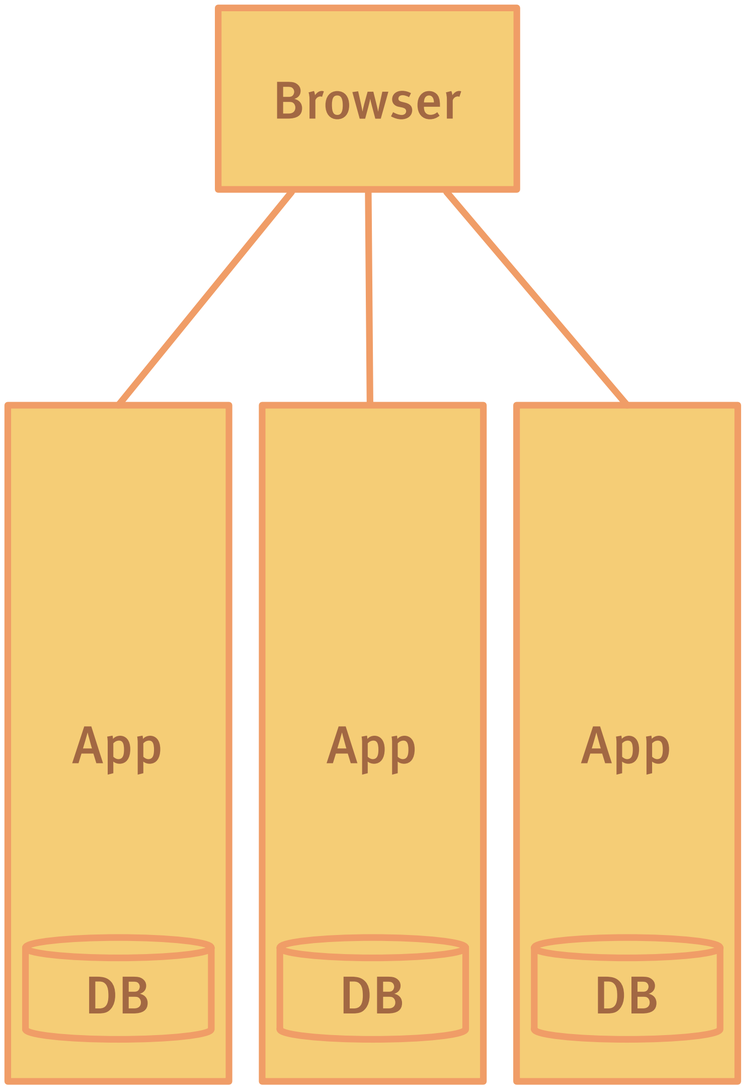
Kraus, Steinacker, Wegner: Teile und Herrsche – Kleine Systeme für große Architekturen, http://bit.ly/152cXbx

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
App
App
DB
DB
DB

2 years in total

Scaled to >100 people

Finished in budget

Finished in quality

Minimum Viable Product

Finished before schedule: October, 24th – 4 months early


NETFLIX

REST APIs

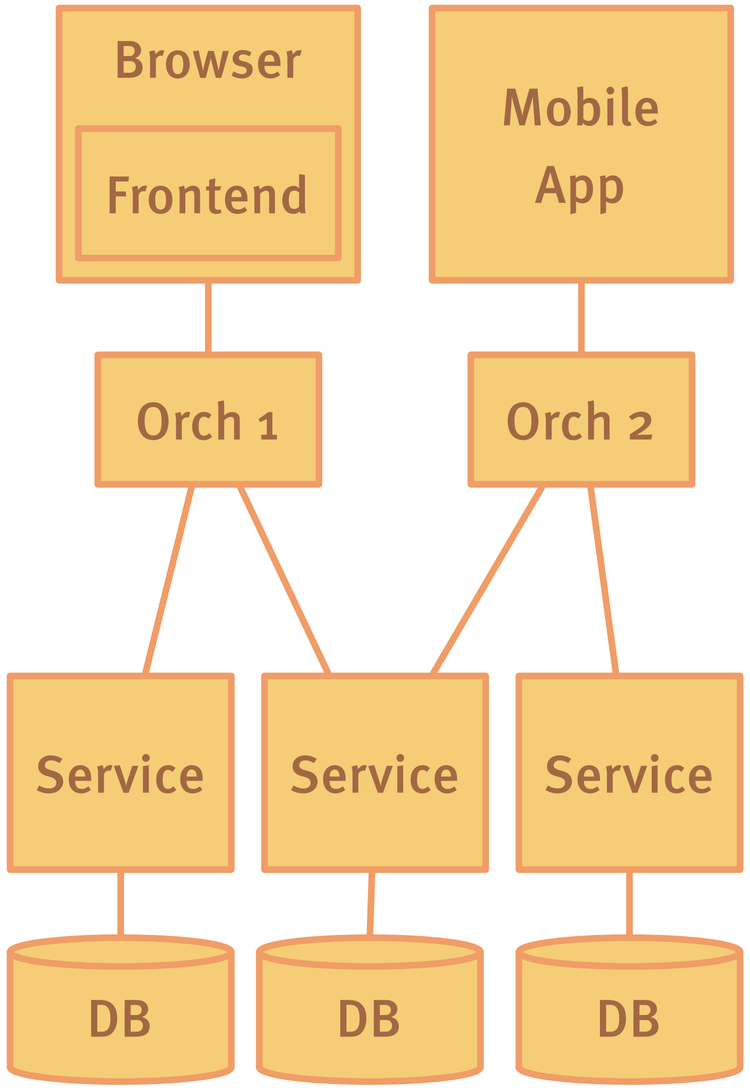
REST APIs

Client-specific orchestration

REST APIs

Client-specific orchestration

Streaming architecture


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

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

6. Challenges

Organization

Cross-system

Responsibilities

UI integration

Communication protocols

Data formats

Redundant data

BI interfaces

Logging, Monitoring

Cross-system

Responsibilities

UI integration

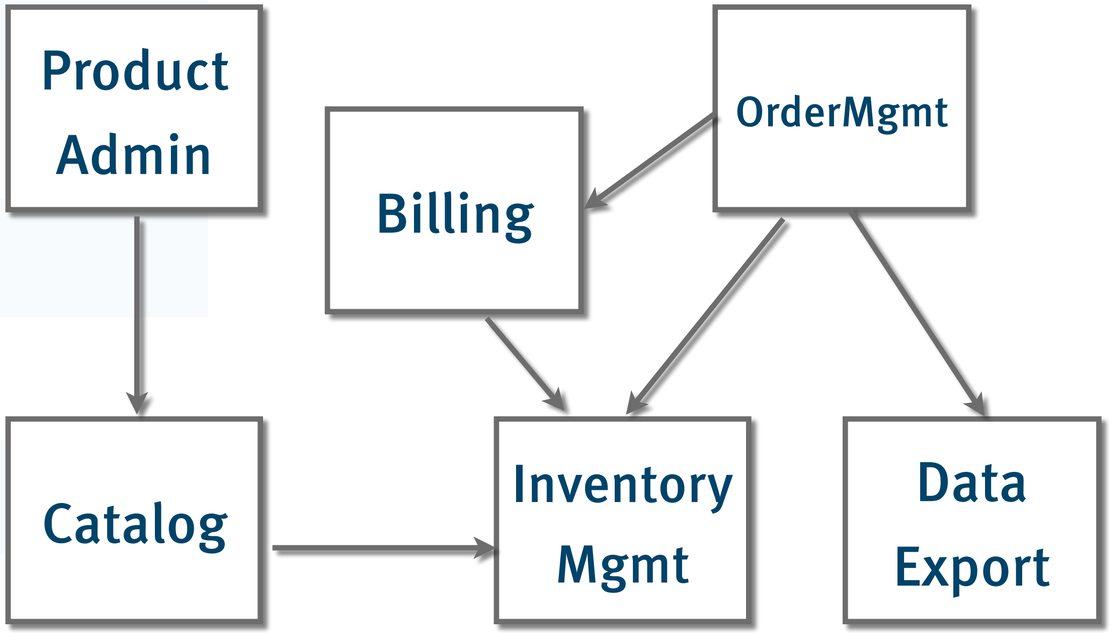
Communication protocols

Data formats

Redundant data

BI interfaces

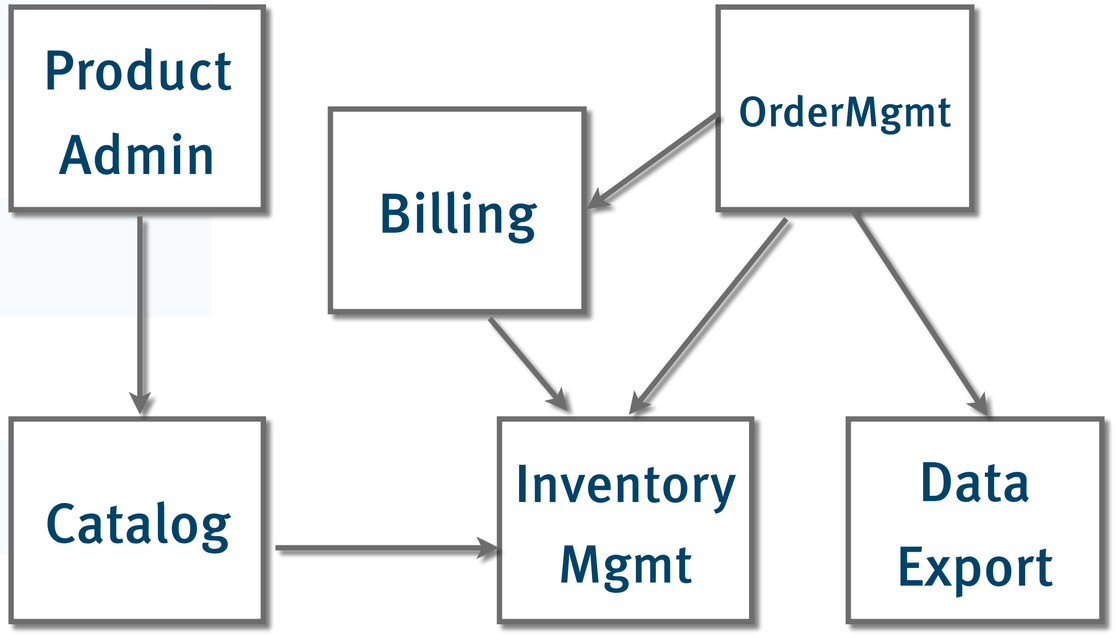
Logging, Monitoring



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

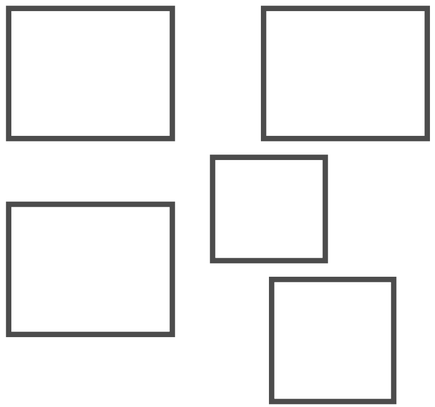
innoQ

Operations

Dev

Ops

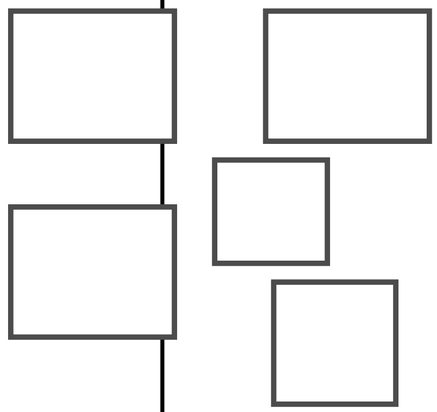
Dev



Ops

Dev

Ops



Dev

Ops

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

innoQ

Migration

Preconditions

Preconditions

High business value

Preconditions

High business value

Very high cost of change

Preconditions

High business value

Very high cost of change

Very slow “time to market”

Preconditions

High business value

Very high cost of change

Very slow “time to market”

Huge backlog of feature requests

Preconditions

High business value

Very high cost of change

Very slow “time to market”

Huge backlog of feature requests

Problem awareness

Preconditions

High business value

Very high cost of change

Very slow “time to market”

Huge backlog of feature requests

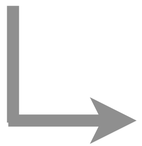
Problem awareness

Strong management support

Close for change

more patterns at **http://aim42.org**

Close for change

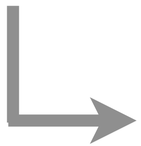


Enable integrateability

(auth/auth, navigation)

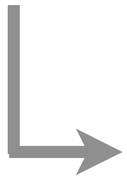
more patterns at **http://aim42.org**

Close for change



Enable integrateability

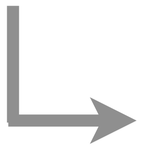
(auth/auth, navigation)



Create new system
for new features

more patterns at **http://aim42.org**

Close for change



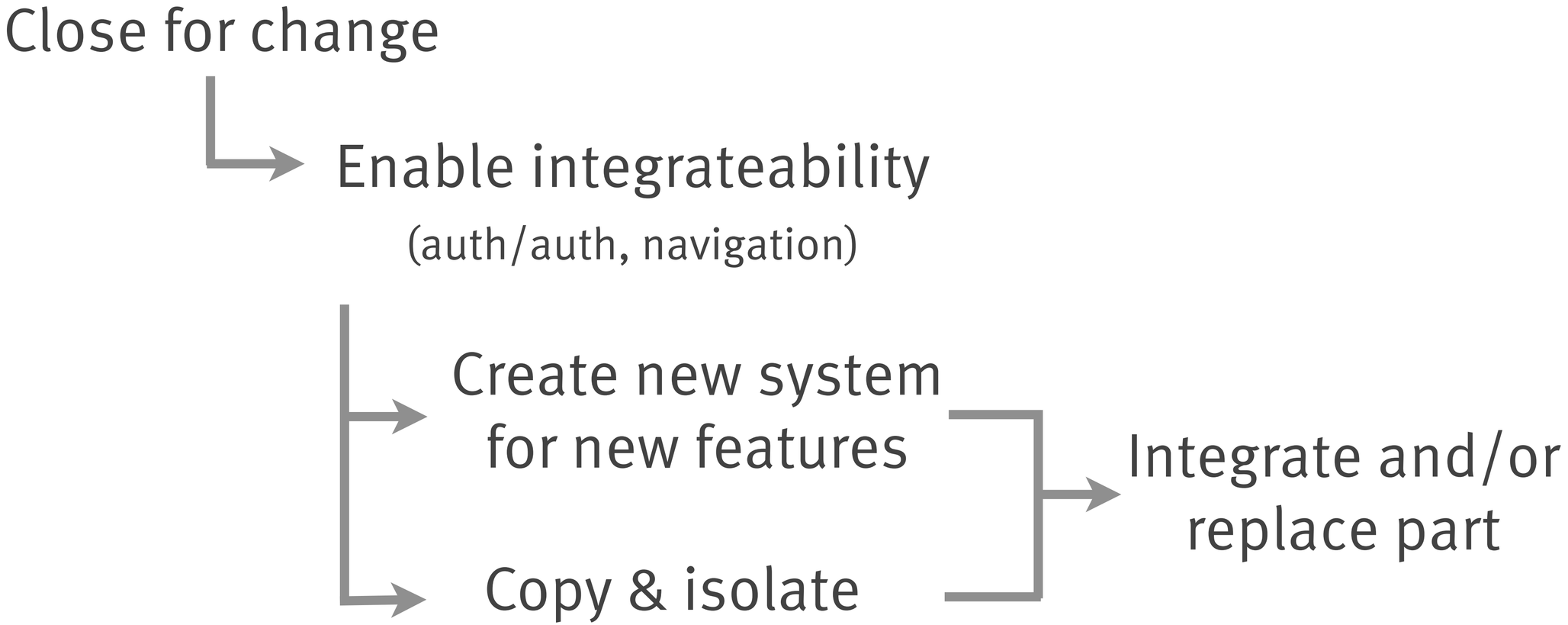
Enable integrateability

(auth/auth, navigation)

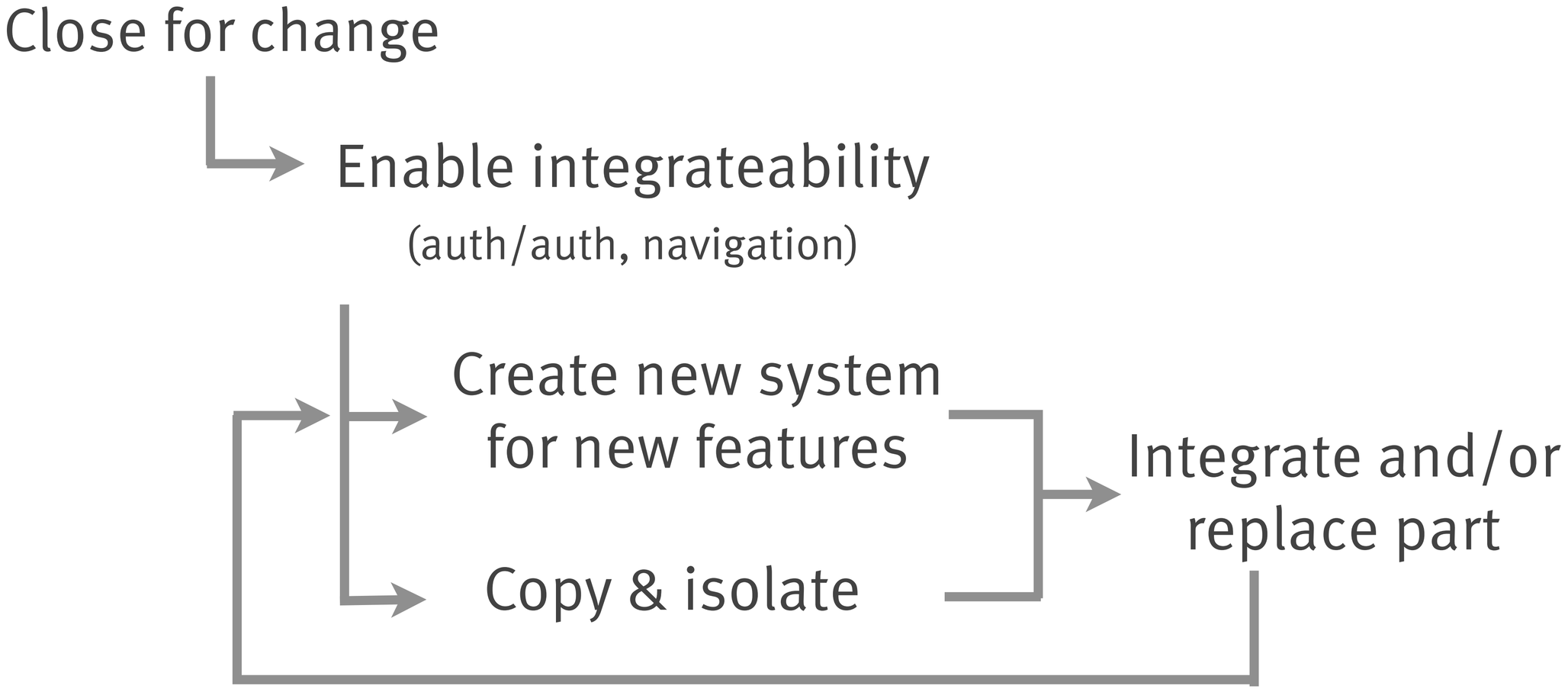
Create new system
for new features

Copy & isolate

more patterns at **http://aim42.org**



more patterns at **http://aim42.org**



more patterns at **http://aim42.org**

Integration

Integration

- distributed configuration

Integration

- distributed configuration
- service registration & discovery

Integration

- distributed configuration
- service registration & discovery
- resilience

Integration

- distributed configuration
- service registration & discovery
- resilience
- simple deployment & operations

Integration

- distributed configuration
- service registration & discovery
- resilience
- simple deployment & operations
- metrics

Integration Challenges

Macro- vs. Micro Architecture

innoQ

Frameworks

Dropwizard

Dropwizard

- Glue Code for well known libraries
- Java

Dropwizard libraries

Dropwizard libraries

- Jetty

Dropwizard libraries

- Jetty
- Jersey

Dropwizard libraries

- Jetty
- Jersey
- Metrics

Dropwizard libraries

- Jetty
- Jersey
- Metrics
- Jackson
- Guava
- Logback
- Hibernate Validator
- Apache Http Client
- JDBI
- Liquibase
- Freemarker & Mustache
- Joda

Spring Boot

and Spring Cloud

Spring Boot

Spring Boot

- convention over configuration approach

Spring Boot

- convention over configuration approach
- Java, Groovy or Scala

Spring Boot

- convention over configuration approach
- Java, Groovy or Scala
- self-contained jar or war

Spring Boot

- convention over configuration approach
- Java, Groovy or Scala
- self-contained jar or war
- tackles dependency-hell via pre-packaging

Spring Cloud

Spring Cloud

- common patterns in distributed systems

Spring Cloud

- common patterns in distributed systems
- umbrella project for cloud connectors

Spring Cloud

- common patterns in distributed systems
- umbrella project for cloud connectors
- build on top of Spring Boot

Spring Cloud

- common patterns in distributed systems
- umbrella project for cloud connectors
- build on top of Spring Boot
- config server for distributed configuration

Spring Cloud

- common patterns in distributed systems
- umbrella project for cloud connectors
- build on top of Spring Boot
- config server for distributed configuration
- annotations for service-discovery & resilience

Play 2

Play 2

- Java or Scala
- based on Akka
- strong async support

Routing

Dropwizard

```
@Path ("/cart")
public class ShoppingCartResource {

    @GET
    @Path("{id}")
    @Timed
    @Produces(MediaType.TEXT_HTML)
    public  ShoppingCartView shoppingCart(@PathParam("id") String cartId,
      @Context UriInfo uriInfo) {
      ImmutableList<String> urls = dao.findItemsForCartId(cartId);
      ImmutableList<Product> products = catalog.resolveProducts(urls);
      return new ShoppingCartView(products,
          uriInfo.getAbsolutePath().toString()
          );
    }

  ...
}
```

Spring Boot

```
@Controller
@RequestMapping(value = ORDERS_RESOURCE)
public class OrderController {

    private @Autowired CounterService counterService;
    private @Autowired OrderRepository repository;

   @RequestMapping(method = RequestMethod.POST)
    public String checkoutCart(@RequestParam(value = "products") List<String> productUris) {
       counterService.increment("checkouts.withproducts." + productUris.size());

      // save products as an order

      return "redirect:" ORDERS_RESOURCE + savedOrder.getId();
    }
}
```

Play

Play

```
# Routes
#  List of all books
GET     /                                   controllers.BooksController.books
# A specific Book
GET    /books/:id          controllers.BooksController.book(id:String)
```

Play

```
object BooksController extends Controller {

  def books = Action.async { implicit request =>
    val bestsellerFuture = Bestseller.getBestseller
    val booksFuture = Books.books
    for {
      bestseller <- bestsellerFuture
      books <- booksFuture
    } yield {
      Ok(views.html.books(books.values.toList, bestseller))
    }
  }
  ...
}
```

Configuration

Play - Typesafe Config

Play - Typesafe Config

- Config Library used by akka, play and others

Play - Typesafe Config

- Config Library used by akka, play and others
- HOCON - JSON Data Model + syntactic sugar

Play - Typesafe Config

- Config Library used by akka, play and others
- HOCON - JSON Data Model + syntactic sugar
- override via system property

Play - Typesafe Config

- Config Library used by akka, play and others
- HOCON - JSON Data Model + syntactic sugar
- override via system property
- rich merge and include possibilities

Spring Boot

```
@ComponentScan
@EnableAutoConfiguration
public class OrderApp {

    public static void main(String[] args) {
        SpringApplication.run(OrderApp.class, args);
    }
}
```

Spring Boot

```
@ComponentScan
@EnableAutoConfiguration
public class OrderApp {

    public static void main(String[] args) {
        SpringApplication.run(OrderApp.class, args);
    }
}
```

Spring Boot

```
@ComponentScan
@EnableAutoConfiguration
public class OrderApp {

    public static void main(String[] args) {
        SpringApplication.run(OrderApp.class, args);
    }
}
```

- opinionated preset

Spring Boot

```
@ComponentScan
@EnableAutoConfiguration
public class OrderApp {

    public static void main(String[] args) {
        SpringApplication.run(OrderApp.class, args);
    }
}
```

- opinionated preset
- HTTP resource “/configprops” shows all properties

Spring Boot

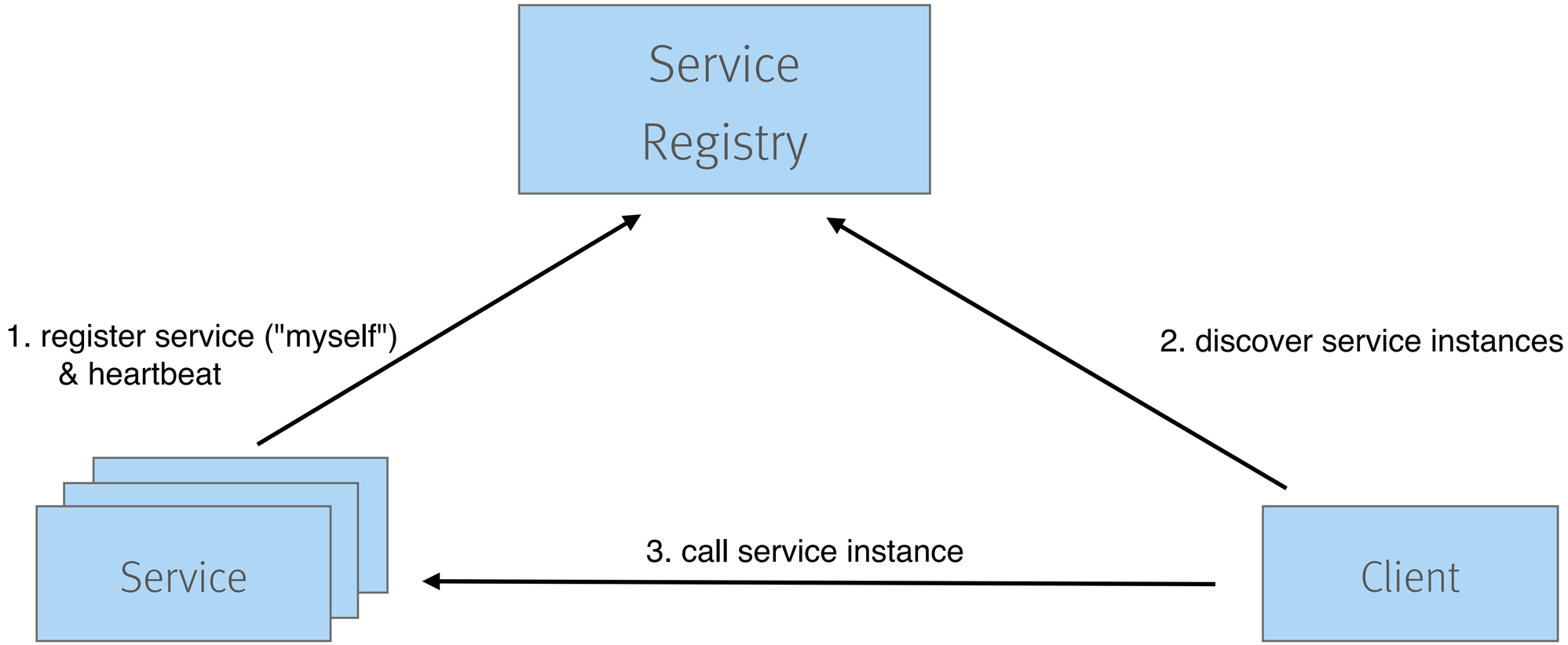
```
@ComponentScan
@EnableAutoConfiguration
public class OrderApp {

    public static void main(String[] args) {
        SpringApplication.run(OrderApp.class, args);
    }
}
```

- opinionated preset
- HTTP resource “/configprops” shows all properties
- overwrite via application.properties or CLI parameter

innoQ

Service Discovery

Service Discovery

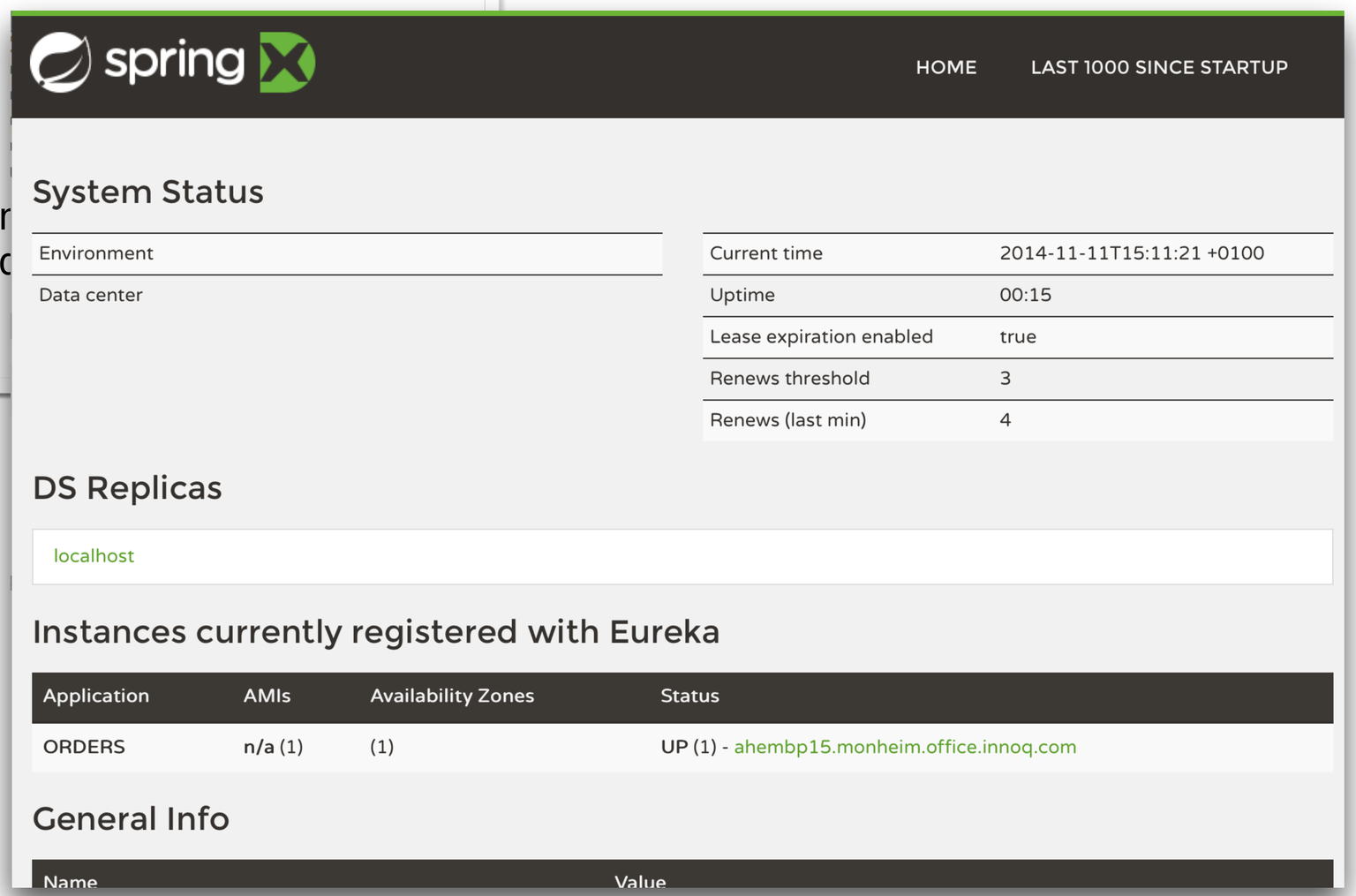


Spring Cloud

```
@ComponentScan
@EnableAutoConfiguration
@EnableEurekaClient
public class OrdersApp {

    public static void main(String[] args) {
        SpringApplication.run(OrdersApp.class, args);
    }
}
```

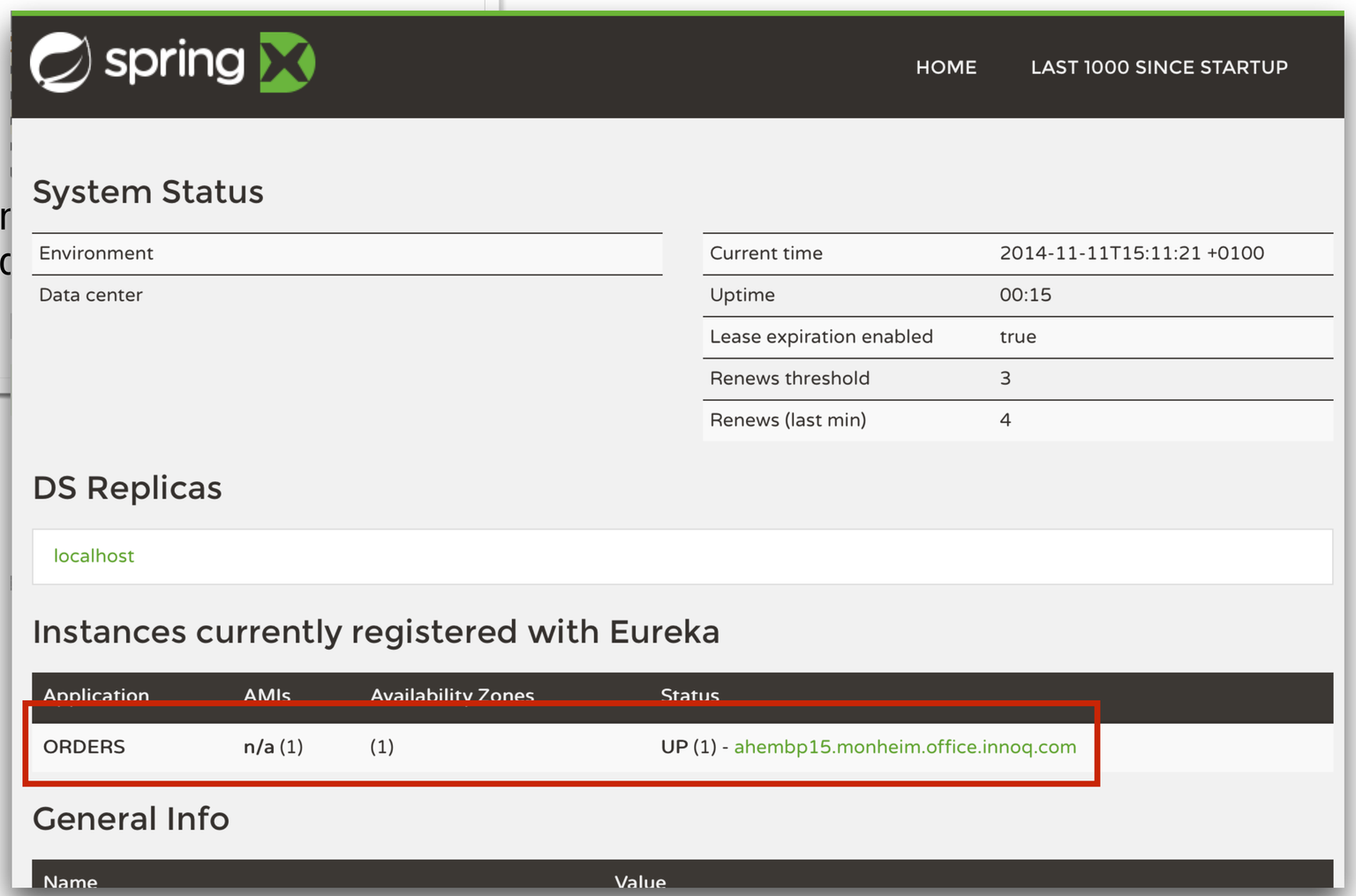
Spring Cloud

```
@ComponentScan
@EnableAutoConfiguration
@EnableEurekaClient
public class OrdersApp {

    public static void main(Strir
        SpringApplication.run(Orc
    }
}
```

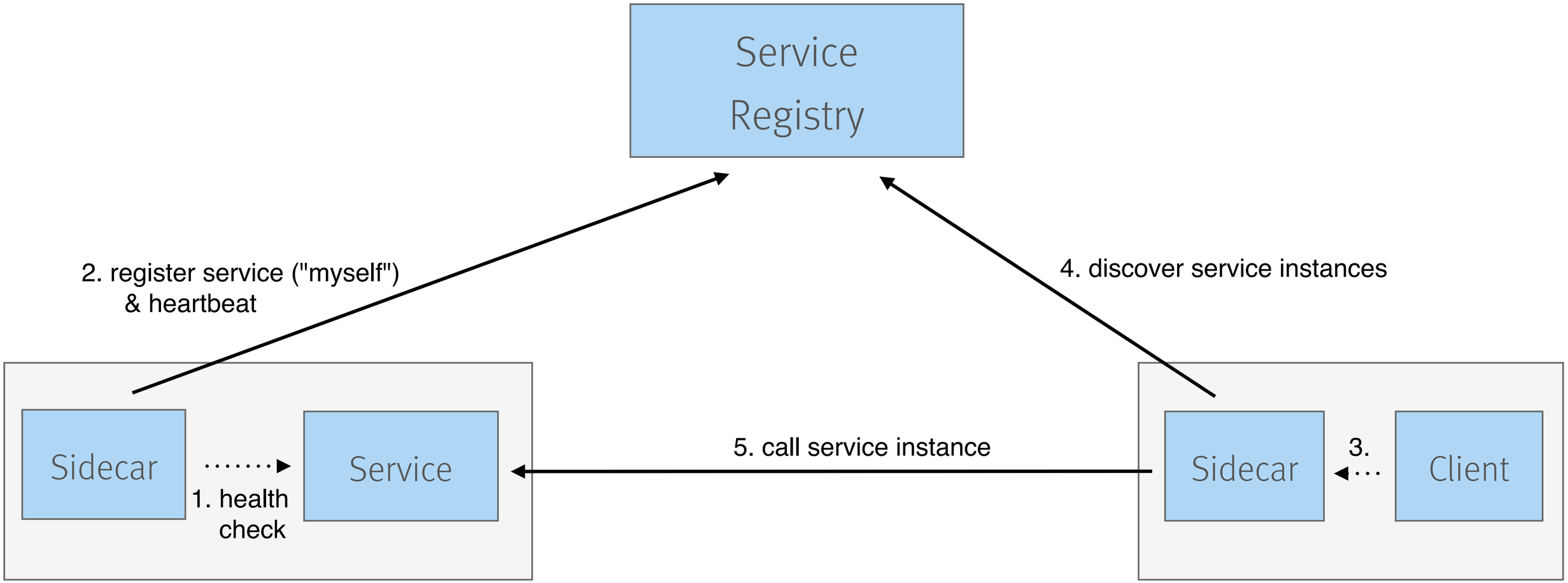


Spring Cloud

```
@ComponentScan
@EnableAutoConfiguration
@EnableEurekaClient
public class OrdersApp {

    public static void main(Strir
        SpringApplication.run(Orc
    }
}
```



Service Discovery with Sidecar

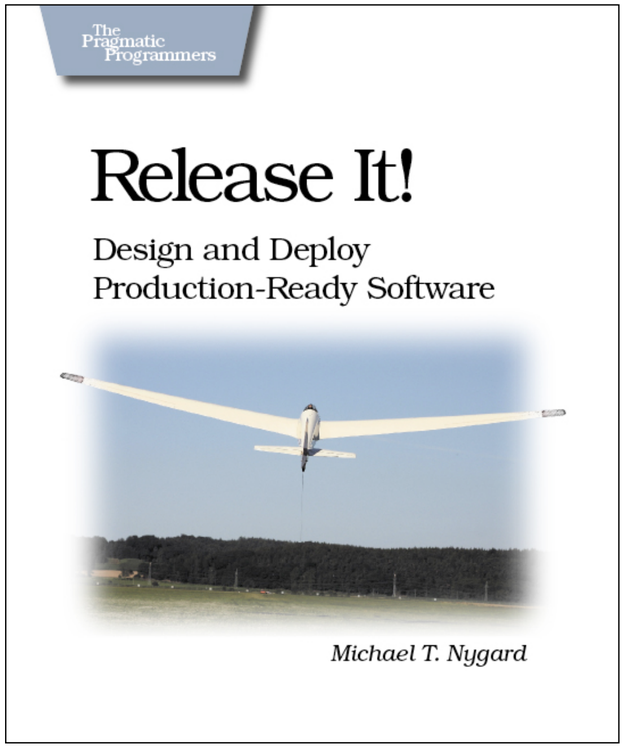


innoQ

Resilience

Resilience

- isolate Failure
- apply graceful degradation
- be responsive in case of failure



Request →  *closed* → service

Request ⇄  *open* service

Request ⇄  *half-open* ⇢ service

http://en.wikipedia.org/wiki/Circuit_breaker

HYSTRIX

DEFEND YOUR APP

- Provides Command-oriented Integration of Services
- Introduces Circuit Breaker, Bulkheads and Isolation
- Decouples from Service-dependencies
- Provides metrics-facility to protect from failures

Hystrix & Dropwizard

```
public class CommandInDropwizard extends TenacityCommand<Product> {


    @Override
    protected Product run() throws Exception {

      Product product = client.resource(url)
        .accept(MediaType.APPLICATION_JSON).get(Product.class);
     return product;
    }

    protected Product getFallback() {
        return FALLBACK_PRODUCT
    }
}
```

Hystrix & Dropwizard

```
public class CommandInDropwizard extends TenacityCommand<Product> {


    @Override
    protected Product run() throws Exception {

      Product product = client.resource(url)
        .accept(MediaType.APPLICATION_JSON).get(Product.class);
     return product;
    }

    protected Product getFallback() {
        return FALLBACK_PRODUCT
    }
}
```

Hystrix & Dropwizard

```
public class CommandInDropwizard extends TenacityCommand<Product> {


    @Override
    protected Product run() throws Exception {

      Product product = client.resource(url)
        .accept(MediaType.APPLICATION_JSON).get(Product.class);
     return product;
    }

    protected Product getFallback() {
        return FALLBACK_PRODUCT
    }
}
```

Hystrix & Dropwizard

```
public class CommandInDropwizard extends TenacityCommand<Product> {


    @Override
    protected Product run() throws Exception {

      Product product = client.resource(url)
        .accept(MediaType.APPLICATION_JSON).get(Product.class);
     return product;
    }

    protected Product getFallback() {
        return FALLBACK_PRODUCT
    }
}
```

Hystrix & Dropwizard

```
public class CommandInDropwizard extends TenacityCommand<Product> {


    @Override
    protected Product run() throws Exception {

      Product product = client.resource(url)
        .accept(MediaType.APPLICATION_JSON).get(Product.class);
     return product;
    }

    protected Product getFallback() {
        return FALLBACK_PRODUCT
    }
}
```

Hystrix & Dropwizard

```
public class CommandInDropwizard extends TenacityCommand<Product> {


    @Override
    protected Product run() throws Exception {

      Product product = client.resource(url)
        .accept(MediaType.APPLICATION_JSON).get(Product.class);
     return product;
    }

    protected Product getFallback() {
        return FALLBACK_PRODUCT
    }
}
```

Hystrix & Dropwizard

```
ResolveProductCommand command = new ResolveProductCommand(client, url);
Product product = command.execute();
```

Spring Cloud Hystrix

```
@HystrixCommand(fallbackMethod = “fallbackProduct")
private Pair<String, ResponseEntity<Product>> resolveProduct(String productUri) {
    final RestTemplate restTemplate = new RestTemplate();
    return new Pair(productUri, restTemplate.getForEntity(productUri, Product.class));
}


private Pair<String, ResponseEntity<Product>> fallbackProduct(String productUri) {
    final Product product = new Product(productUri, null, BigDecimal.ZERO);
    final ResponseEntity<Product> response = new ResponseEntity<Product>(product, PARTIAL_CONTENT);
    return new Pair(productUri, response);
}
```

Spring Cloud Hystrix

auto-wrapped with command!

```
@HystrixCommand(fallbackMethod = “fallbackProduct")
private Pair<String, ResponseEntity<Product>> resolveProduct(String productUri) {
    final RestTemplate restTemplate = new RestTemplate();
    return new Pair(productUri, restTemplate.getForEntity(productUri, Product.class));
}


private Pair<String, ResponseEntity<Product>> fallbackProduct(String productUri) {
    final Product product = new Product(productUri, null, BigDecimal.ZERO);
    final ResponseEntity<Product> response = new ResponseEntity<Product>(product, PARTIAL_CONTENT);
    return new Pair(productUri, response);
}
```

Spring Cloud Hystrix

```
@HystrixCommand(fallbackMethod = “fallbackProduct")
private Pair<String, ResponseEntity<Product>> resolveProduct(String productUri) {
    final RestTemplate restTemplate = new RestTemplate();
    return new Pair(productUri, restTemplate.getForEntity(productUri, Product.class));
}


private Pair<String, ResponseEntity<Product>> fallbackProduct(String productUri) {
    final Product product = new Product(productUri, null, BigDecimal.ZERO);
    final ResponseEntity<Product> response = new ResponseEntity<Product>(product, PARTIAL_CONTENT);
    return new Pair(productUri, response);
}
```

Spring Cloud Hystrix

```
@HystrixCommand(fallbackMethod = “fallbackProduct")
private Pair<String, ResponseEntity<Product>> resolveProduct(String productUri) {
    final RestTemplate restTemplate = new RestTemplate();
    return new Pair(productUri, restTemplate.getForEntity(productUri, Product.class));
}


private Pair<String, ResponseEntity<Product>> fallbackProduct(String productUri) {
    final Product product = new Product(productUri, null, BigDecimal.ZERO);
    final ResponseEntity<Product> response = new ResponseEntity<Product>(product, PARTIAL_CONTENT);
    return new Pair(productUri, response);
}
```

Spring Cloud Hystrix

method reference!

```
@HystrixCommand(fallbackMethod = “fallbackProduct")
private Pair<String, ResponseEntity<Product>> resolveProduct(String productUri) {
    final RestTemplate restTemplate = new RestTemplate();
    return new Pair(productUri, restTemplate.getForEntity(productUri, Product.class));
}


private Pair<String, ResponseEntity<Product>> fallbackProduct(String productUri) {
    final Product product = new Product(productUri, null, BigDecimal.ZERO);
    final ResponseEntity<Product> response = new ResponseEntity<Product>(product, PARTIAL_CONTENT);
    return new Pair(productUri, response);
}
```

Play - Circuit Breaker

```
val apiUrl = "..."
val breaker =
  CircuitBreaker(Akka.system().scheduler,
    maxFailures = 5,
    callTimeout = 2.seconds,
    resetTimeout = 1.minute)

def getBestseller : Future[List[Bestseller]] = {
  breaker.withCircuitBreaker(
    WS.url(apiUrl).get.map {
      response => response.json.as[List[Bestseller]]
    }).recover { case e => List() }
}
```

Play - Circuit Breaker

```
val apiUrl = "..."
val breaker =
  CircuitBreaker(Akka.system().scheduler,
    maxFailures = 5,
    callTimeout = 2.seconds,
    resetTimeout = 1.minute)

def getBestseller : Future[List[Bestseller]] = {
  breaker.withCircuitBreaker(
    WS.url(apiUrl).get.map {
      response => response.json.as[List[Bestseller]]
    }).recover { case e => List() }
}
```

Play - Circuit Breaker

```
val apiUrl = "..."
val breaker =
  CircuitBreaker(Akka.system().scheduler,
    maxFailures = 5,
    callTimeout = 2.seconds,
    resetTimeout = 1.minute)

def getBestseller : Future[List[Bestseller]] = {
  breaker.withCircuitBreaker(
    WS.url(apiUrl).get.map {
      response => response.json.as[List[Bestseller]]
    }).recover { case e => List() }
}
```

Play - Circuit Breaker

```
val apiUrl = "..."
val breaker =
  CircuitBreaker(Akka.system().scheduler,
    maxFailures = 5,
    callTimeout = 2.seconds,
    resetTimeout = 1.minute)

def getBestseller : Future[List[Bestseller]] = {
  breaker.withCircuitBreaker(
    WS.url(apiUrl).get.map {
      response => response.json.as[List[Bestseller]]
    }) recover { case e => List() }
}
```

Play - Circuit Breaker

```
val apiUrl = "..."
val breaker =
  CircuitBreaker(Akka.system().scheduler,
    maxFailures = 5,
    callTimeout = 2.seconds,
    resetTimeout = 1.minute)

def getBestseller : Future[List[Bestseller]] = {
  breaker.withCircuitBreaker(
    WS.url(apiUrl).get.map {
      response => response.json.as[List[Bestseller]]
    }).recover { case e => List() }
}
```

Play - Circuit Breaker

```
val apiUrl = "..."
val breaker =
  CircuitBreaker(Akka.system().scheduler,
    maxFailures = 5,
    callTimeout = 2.seconds,
    resetTimeout = 1.minute)

def getBestseller : Future[List[Bestseller]] = {
  breaker.withCircuitBreaker(
    WS.url(apiUrl).get.map {
      response => response.json.as[List[Bestseller]]
    }).recover { case e => List() }
}
```

Play - Circuit Breaker

```
val apiUrl = "..."
val breaker =
  CircuitBreaker(Akka.system().scheduler,
    maxFailures = 5,
    callTimeout = 2.seconds,
    resetTimeout = 1.minute)

def getBestseller : Future[List[Bestseller]] = {
  breaker.withCircuitBreaker(
    WS.url(apiUrl).get.map {
      response => response.json.as[List[Bestseller]]
    }).recover { case e => List() }
}
```

Play - Circuit Breaker

```
val apiUrl = "..."
val breaker =
  CircuitBreaker(Akka.system().scheduler,
    maxFailures = 5,
    callTimeout = 2.seconds,
    resetTimeout = 1.minute)

def getBestseller : Future[List[Bestseller]] = {
  breaker.withCircuitBreaker(
    WS.url(apiUrl).get.map {
      response => response.json.as[List[Bestseller]]
    }).recover { case e => List() }
}
```

Play - Circuit Breaker

```
val apiUrl = "..."
val breaker =
  CircuitBreaker(Akka.system().scheduler,
    maxFailures = 5,
    callTimeout = 2.seconds,
    resetTimeout = 1.minute)

def getBestseller : Future[List[Bestseller]] = {
  breaker.withCircuitBreaker(
    WS.url(apiUrl).get.map {
      response => response.json.as[List[Bestseller]]
    }).recover { case e => List() }
}
```

innoQ

Deployment

Spring Boot - Packaging

```
./gradlew distZip
./gradlew build
```

Spring Boot - Packaging

```
./gradlew distZip
./gradlew build
```

ZIP + shell-script

Spring Boot - Packaging

```
./gradlew distZip
./gradlew build
```

ZIP + shell-script

executable JAR

Play - Packaging

```
sbt dist
sbt debian:packageBin
sbt rpm:packageBin
```

innoQ

Metrics

Dropwizard

- “Metrics” Integrated with Dropwizard
- @Timed on Resources
- HTTP Client is already instrumented
- JVM Data

```
"org.apache.http.client.HttpClient.cart.get-requests": {
    "count": 11,
    "max": 0.062107,
    "mean": 0.013355909090909092,
    "min": 0.00575000000000001,
    "p50": 0.009454,
    "p75": 0.010427,
    "p95": 0.062107,
    "p98": 0.062107,
    "p99": 0.062107,
    "p999": 0.062107,
    "stddev": 0.016285873488729705,
    "m15_rate": 0,
    "m1_rate": 0,
    "m5_rate": 0,
    "mean_rate": 2.9714422786532126,
    "duration_units": "seconds",
    "rate_units": "calls/second"
}
```

```
"cart.resources.ShoppingCartResource.shoppingCart": {
    "count": 22,
    "max": 0.136162,
    "mean": 0.01208109090909091,
    "min": 0.00093,
    "p50": 0.008174500000000001,
    "p75": 0.011782250000000001,
    "p95": 0.11783499999999976,
    "p98": 0.136162,
    "p99": 0.136162,
    "p999": 0.136162,
    "stddev": 0.028135302398214 26,
    "m15_rate": 1.8524577712890011,
    "m1_rate": 0.18057796798879996,
    "m5_rate": 1.315746847992022,
    "mean_rate": 0.133050618509084,
    "duration_units": "seconds",
    "rate_units": "calls/second"
}
```

Dropwizard Metrics

- Exposed over HTTP (as Json)
- Exposed as jmx
- Others available: stdout, csv, slf4j, ganglia, graphite

Spring Boot Metrics

- Spring Boot “actuator” module
- enables HTTP resources for metrics
- configurable via application.properties

http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#production-ready

Using a counter metric in your Java code...

```
counterService.increment("checkouts.withproducts." + productUris.size());
```

...will display it in the /metrics JSON

```
GET /metrics HTTP/1.1

{

    "counter.checkouts.withproducts.3": 4,
    ...
}
```

Summary

Summary

Explicitly design system boundaries

Summary

Explicitly design system boundaries

Modularize into independent, self-contained systems

Summary

Explicitly design system boundaries

Modularize into independent, self-contained systems

Separate micro and macro architectures

Summary

Explicitly design system boundaries

Modularize into independent, self-contained systems

Separate micro and macro architectures

MicroServices aren’t micro!

Summary

Explicitly design system boundaries

Modularize into independent, self-contained systems

Separate micro and macro architectures

MicroServices aren’t micro!

Strike a balance between control and decentralization

Thank you!

Questions?

Comments?

Martin Eigenbrodt | eigenbrodtm
martin.eigenbrodt@innoq.com

Alexander Heusingfeld | goldstift
alexander.heusingfeld@innoq.com

https://www.innoq.com/en/talks/2015/01/talk-microservices-on-the-jvm/

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