SOFTWARE ARCHITECTURE FOUNDATIONS:

StYLeS & pAtTErNs

ThoughtWorks®

NEAL FORD

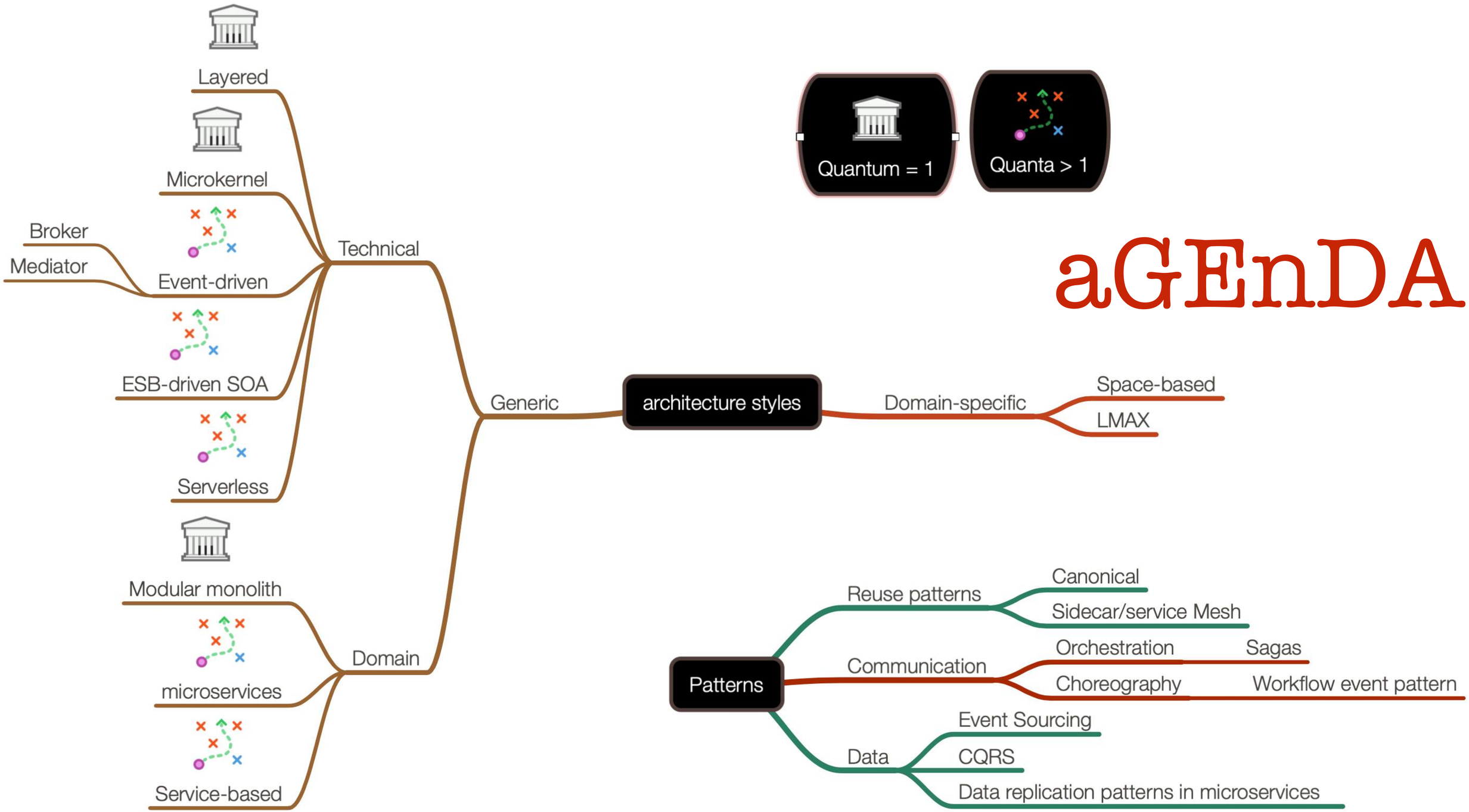
Director / Software Architect / Meme Wrangler

@neal4d

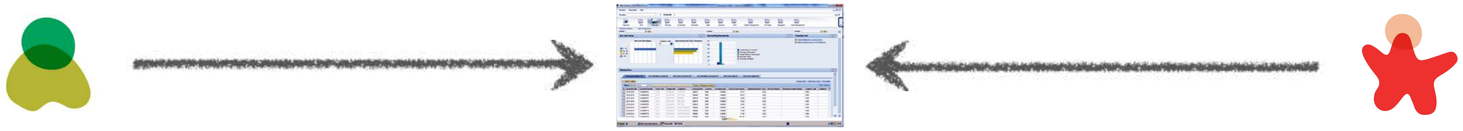
http://nealford.com

2020-01-14

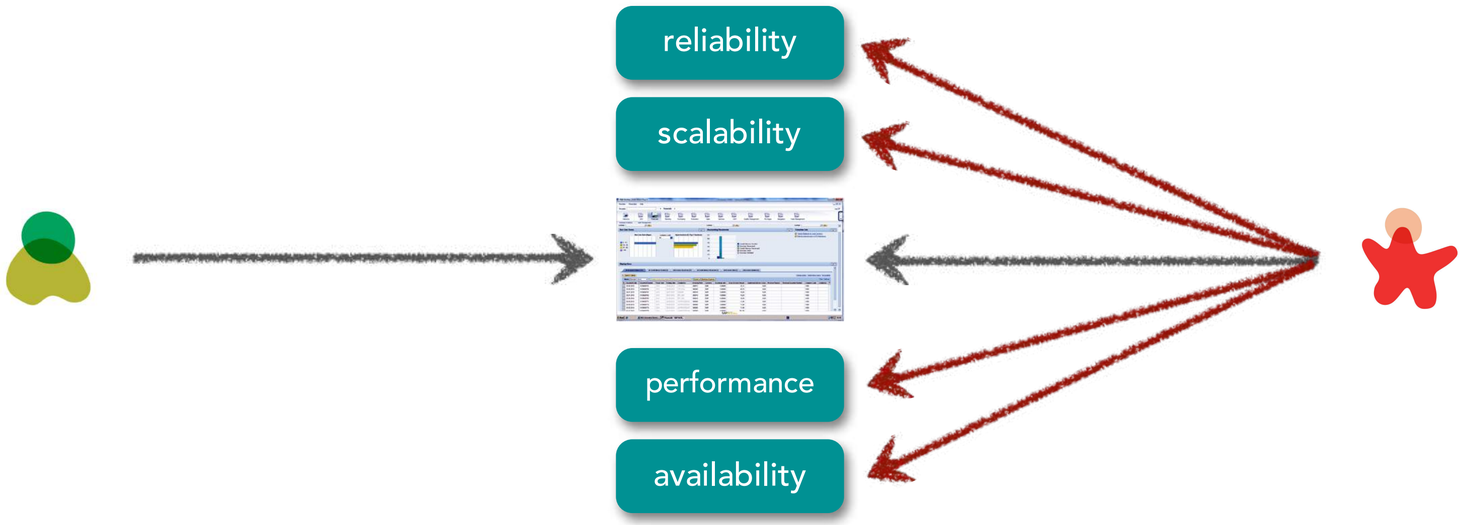
aGEnDA


Quantum = 1
Quanta > 1
architecture styles
Generic
Technical
Layered
Microkernel
Event-driven
Broker
Mediator
ESB-driven SOA
Serverless
Domain
Modular monolith
microservices
Service-based
Domain-specific
Space-based
LMAX
Patterns
Reuse patterns
Canonical
Sidecar/service Mesh
Communication
Orchestration
Sagas
Choreography
Workflow event pattern
Data
Event Sourcing
CQRS
Data replication patterns in microservices

Architecture Characteristics



Architecture Characteristics



Architecture Characteristics

1. synergistic

2. ill-defined

3. voluminous

Architecture Characteristics

accessibility
accountability
accuracy
adaptability
administrability
affordability
agility
auditability
autonomy
availability
compatibility
composability
configurability
correctness
credibility
customizability
debugability
degradability
determinability
demonstrability
dependability
deployability
discoverability
distributability
durability
effectiveness
efficiency

evolvability
extensibility
failure transparency
fault-tolerance
fidelity
flexibility
inspectability
installability
integrity
interchangeability
interoperability
learnability
maintainability
manageability
mobility
modifiability
modularity
operability
orthogonality
portability
precision
predictability
process capabilities
producibility
provability
recoverability
relevance

repeatability
reproducibility
resilience
responsiveness
reusability
robustness
safety
scalability
seamlessness
self-sustainability
serviceability
supportability
securability
simplicity
stability
standards compliance
survivability
sustainability
tailorability
testability
timeliness
traceability
transparency
ubiquity
understandability
upgradability
usability

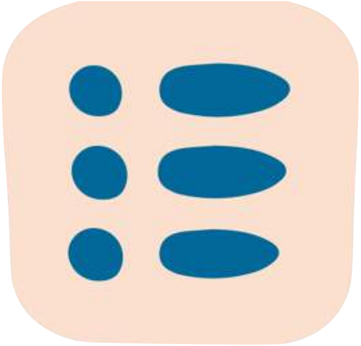
https://en.wikipedia.org/wiki/List_of_system_quality_attributes

Architecture Characteristics

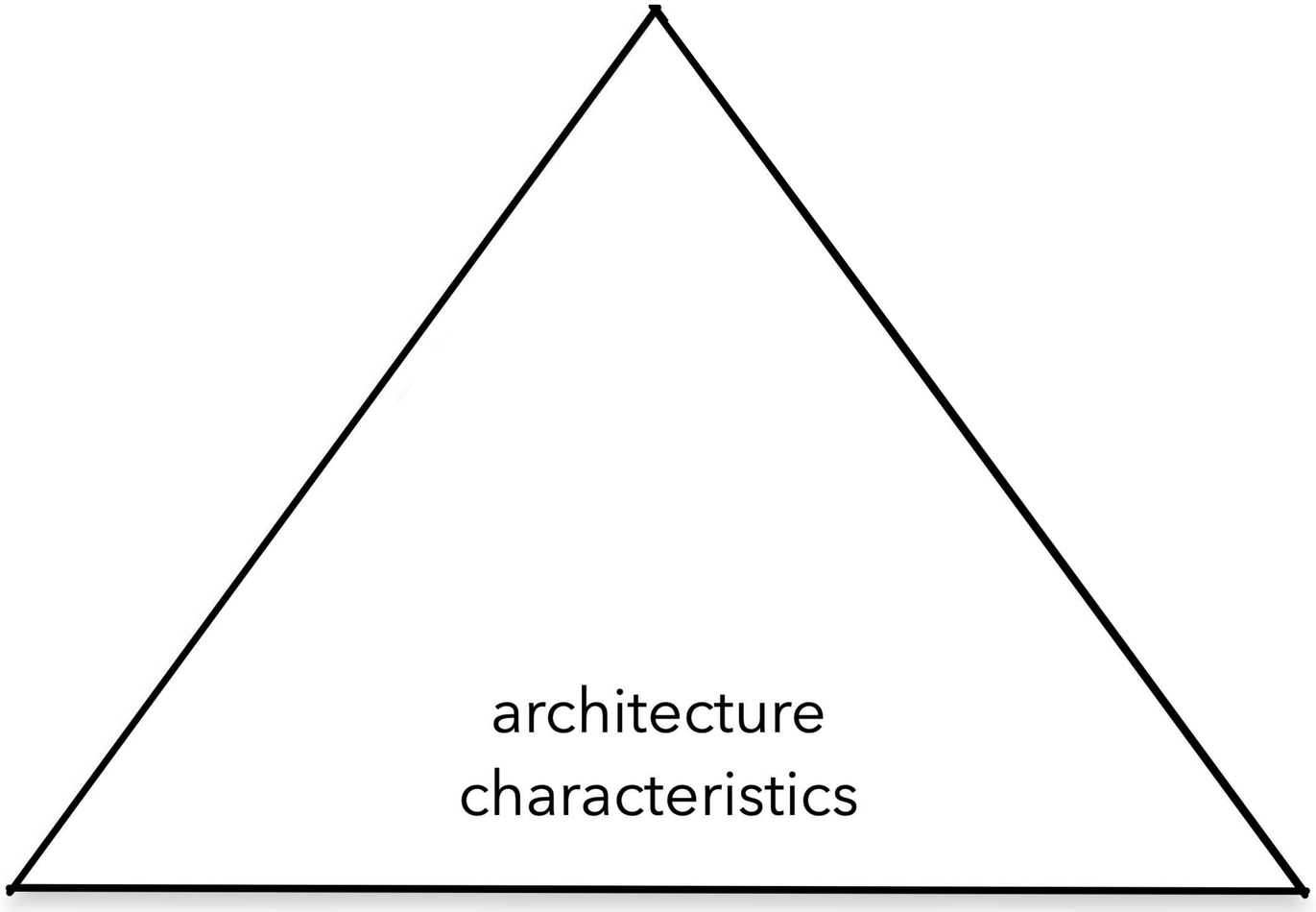
1. synergistic

2. ill-defined

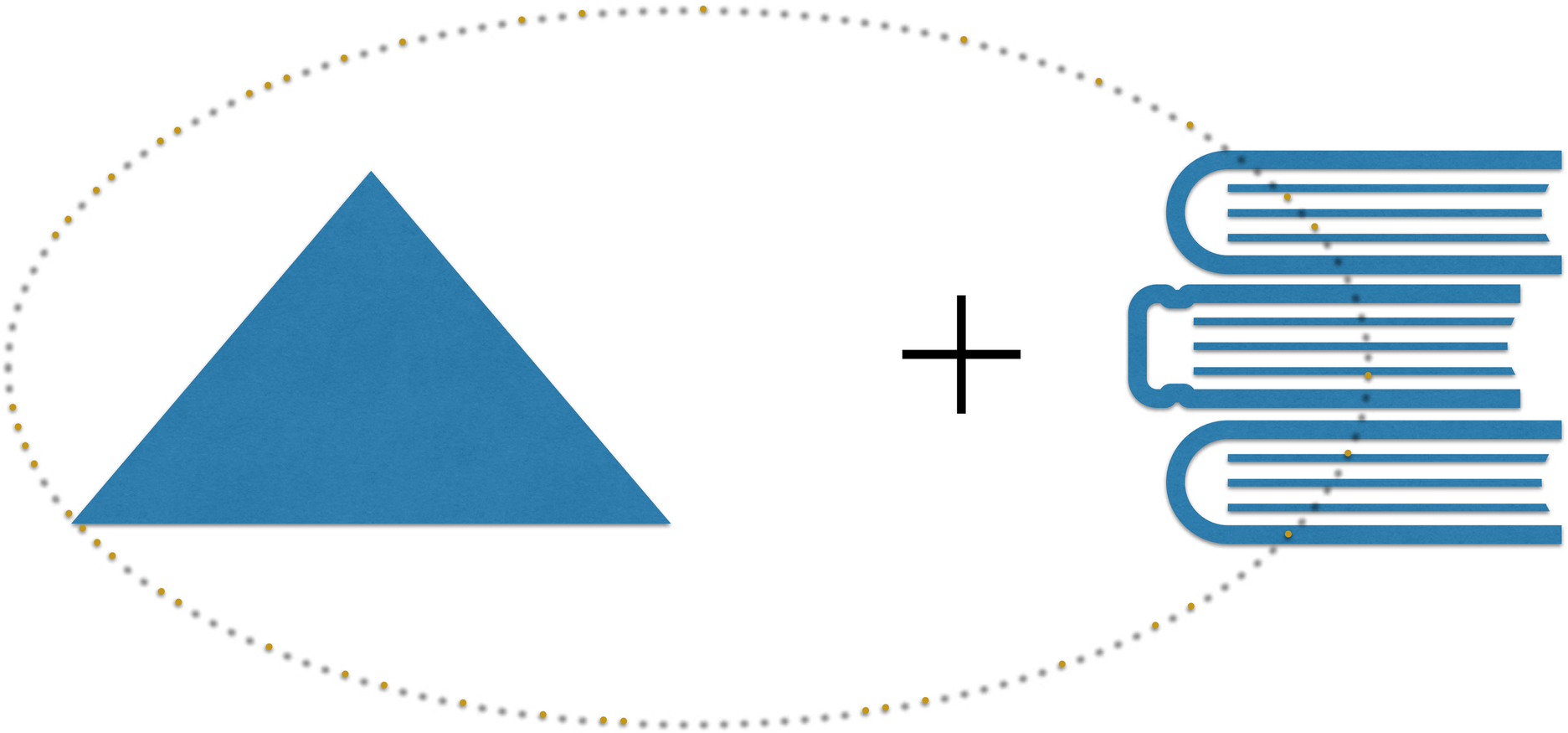
4. numerous categories



3. voluminous

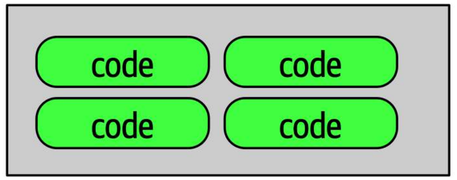

architecture
characteristics

architecture design

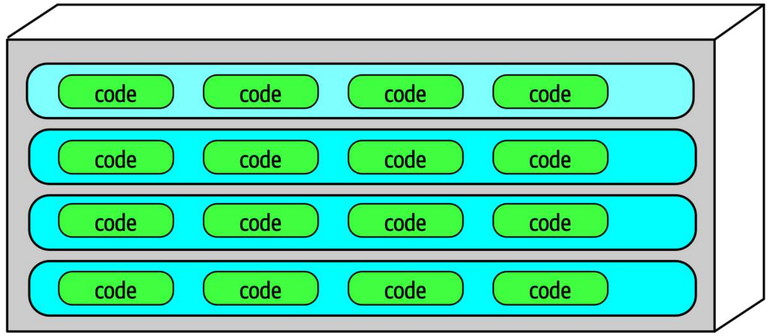


architecture styles

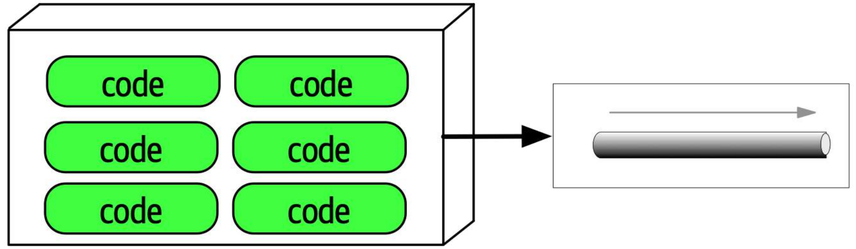
components as building blocks



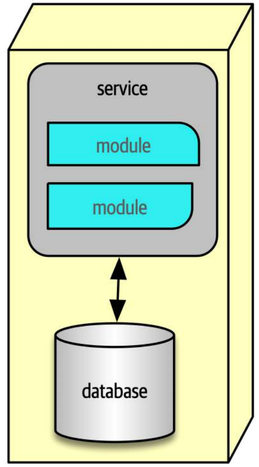
wrapper of related code



layer or subsystem



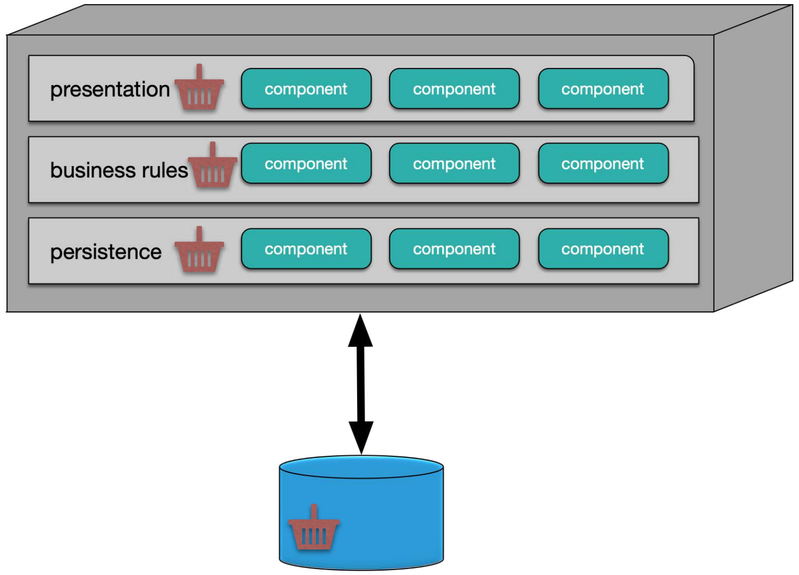
event processor



distributed service

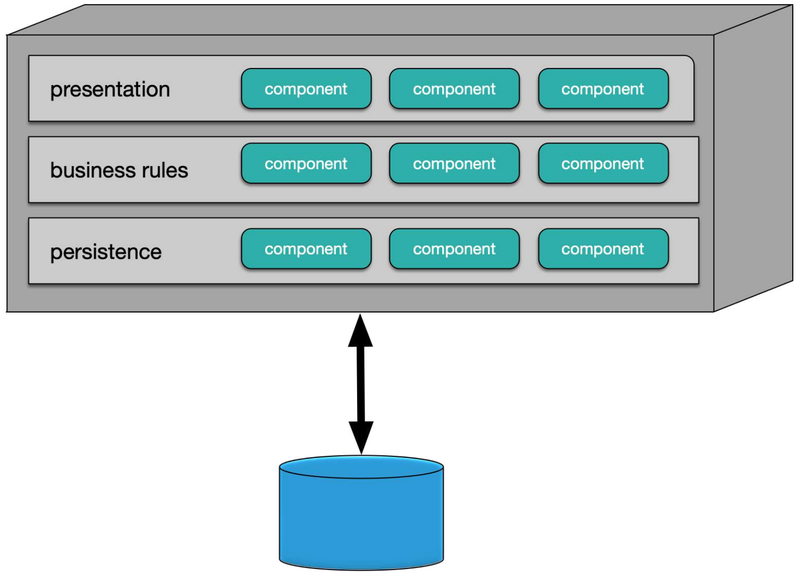
Top Level Partitioning

technical partitioning

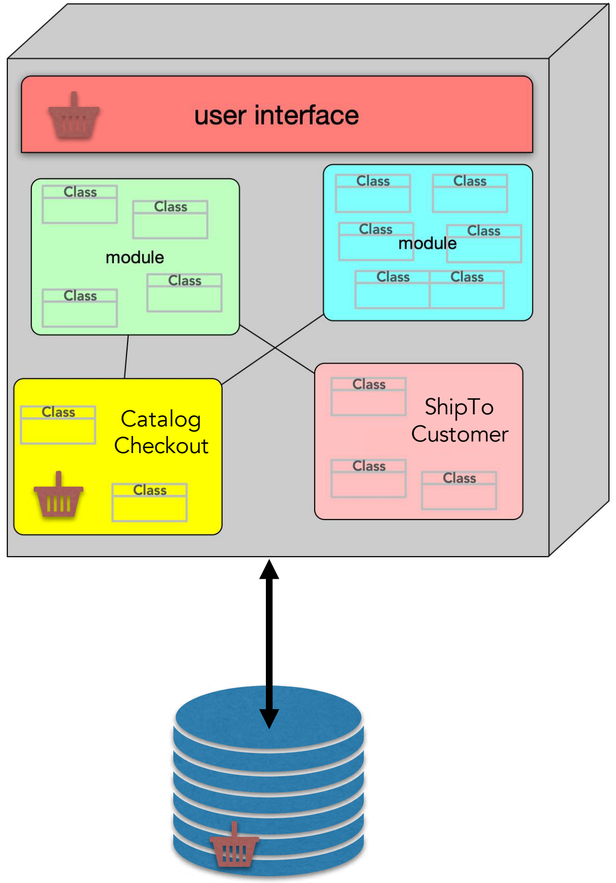


Top Level Partitioning

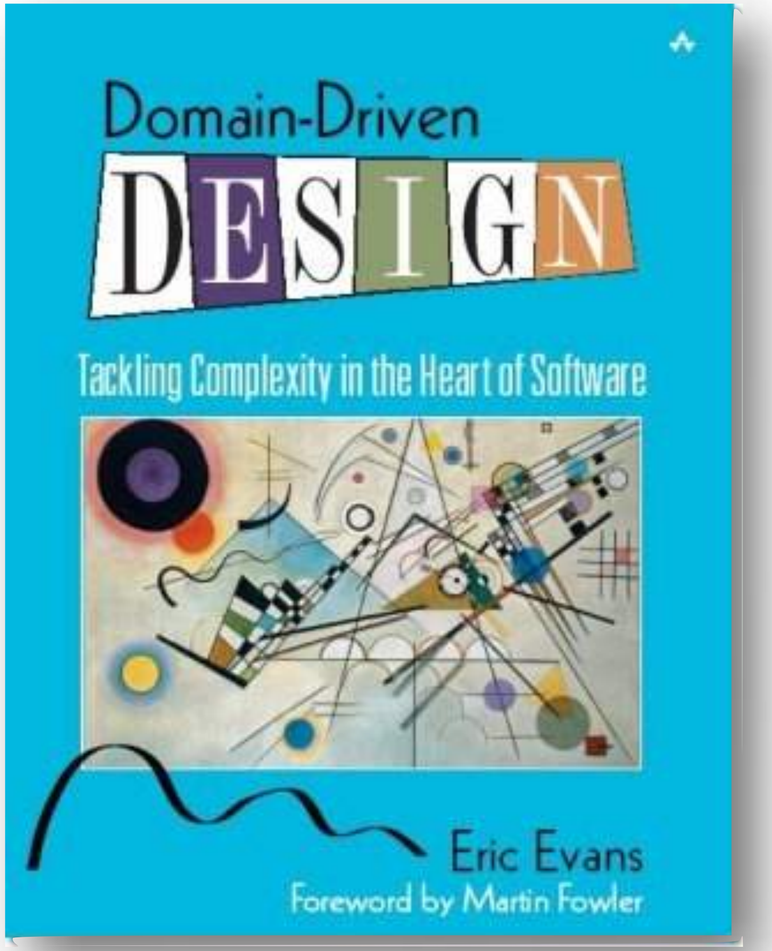
technical partitioning

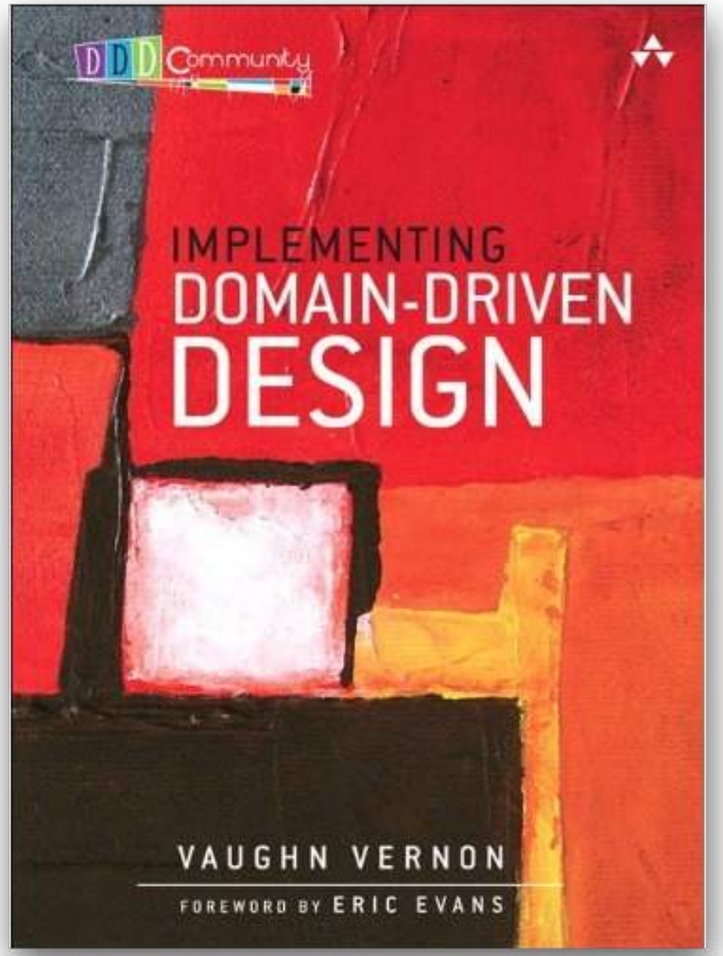


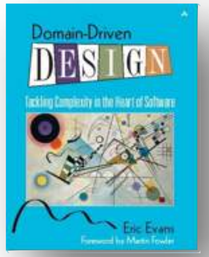
domain partitioning



Domain Driven Design

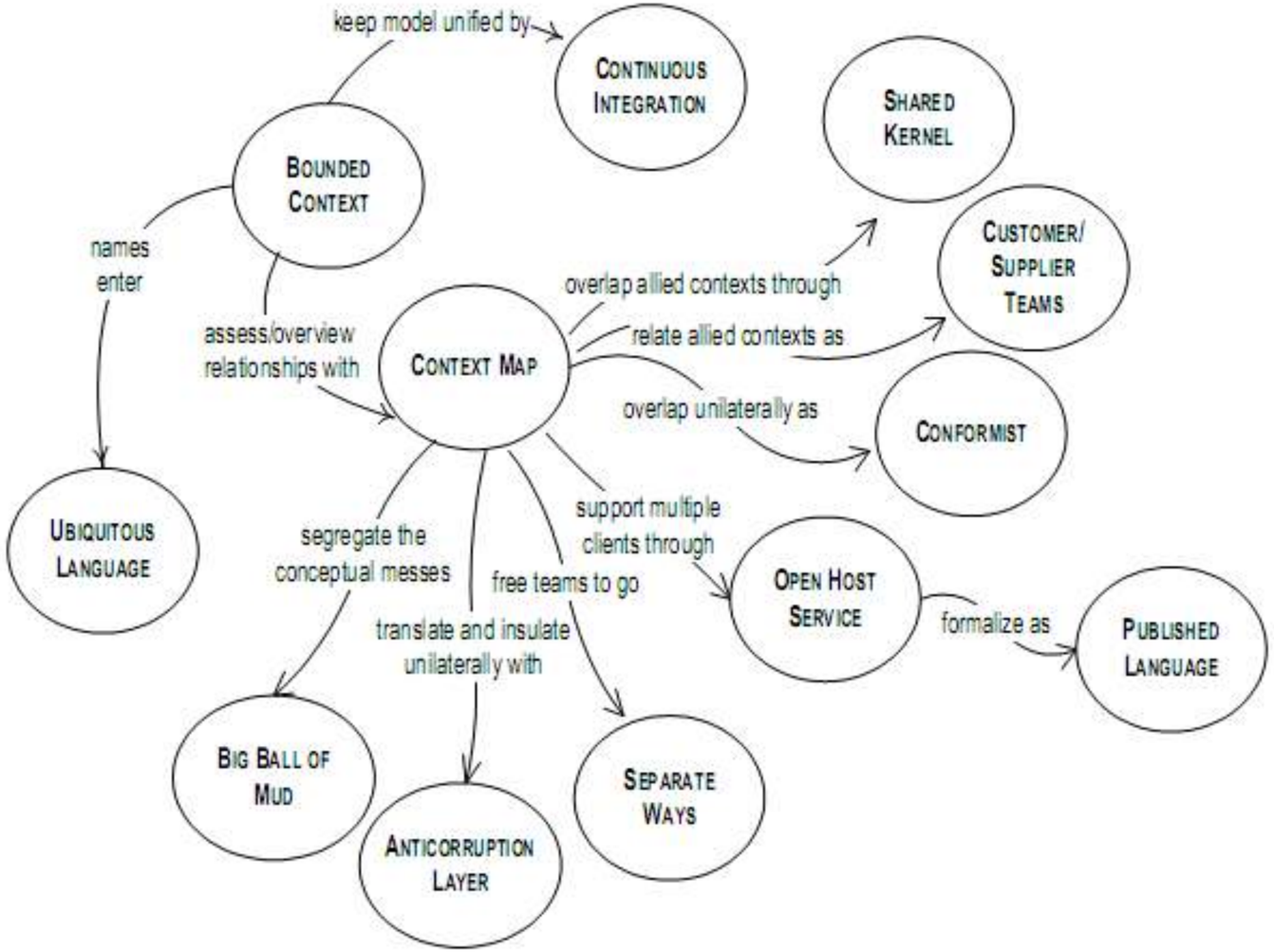




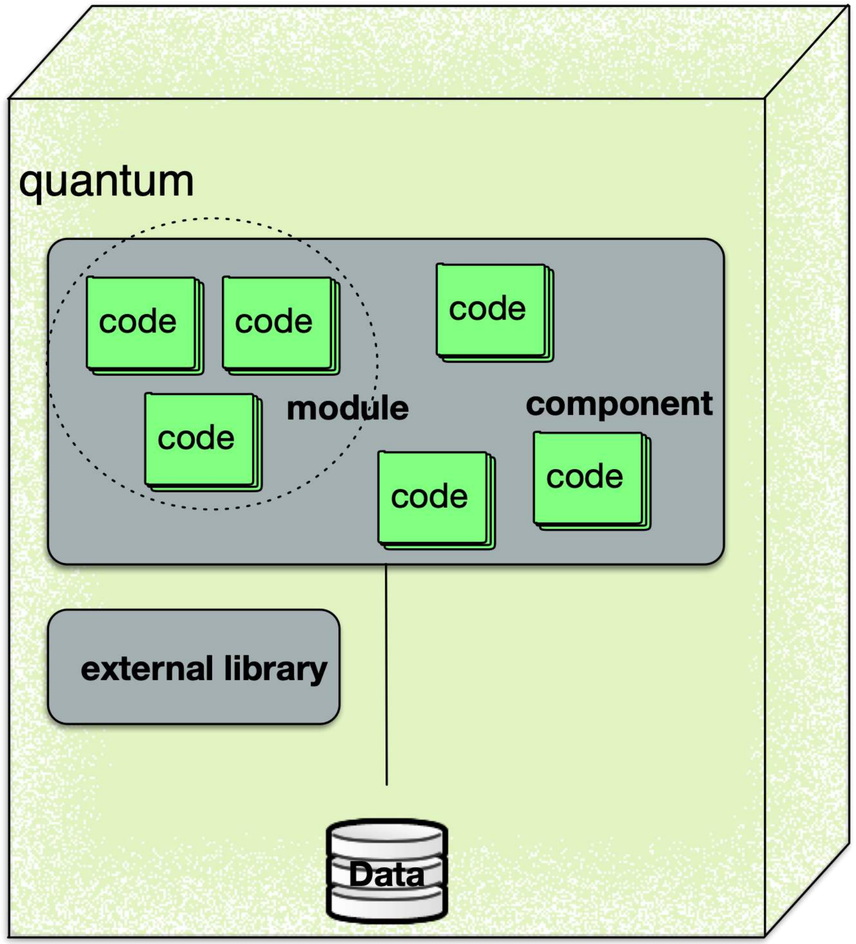


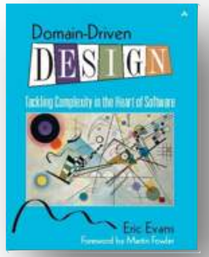
Bounded Context

Maintaining Model Integrity



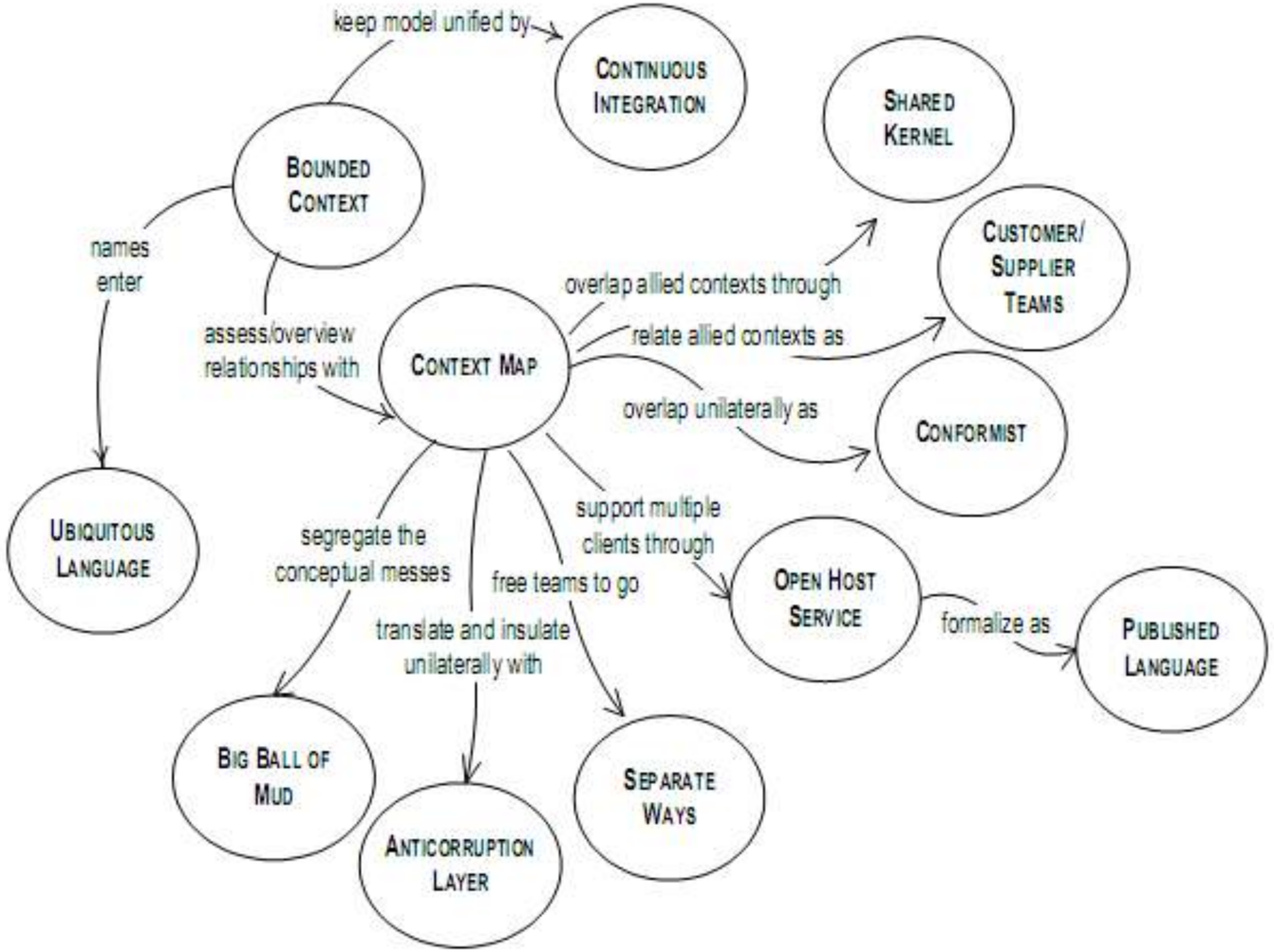
Architectural Quantum





Bounded Context

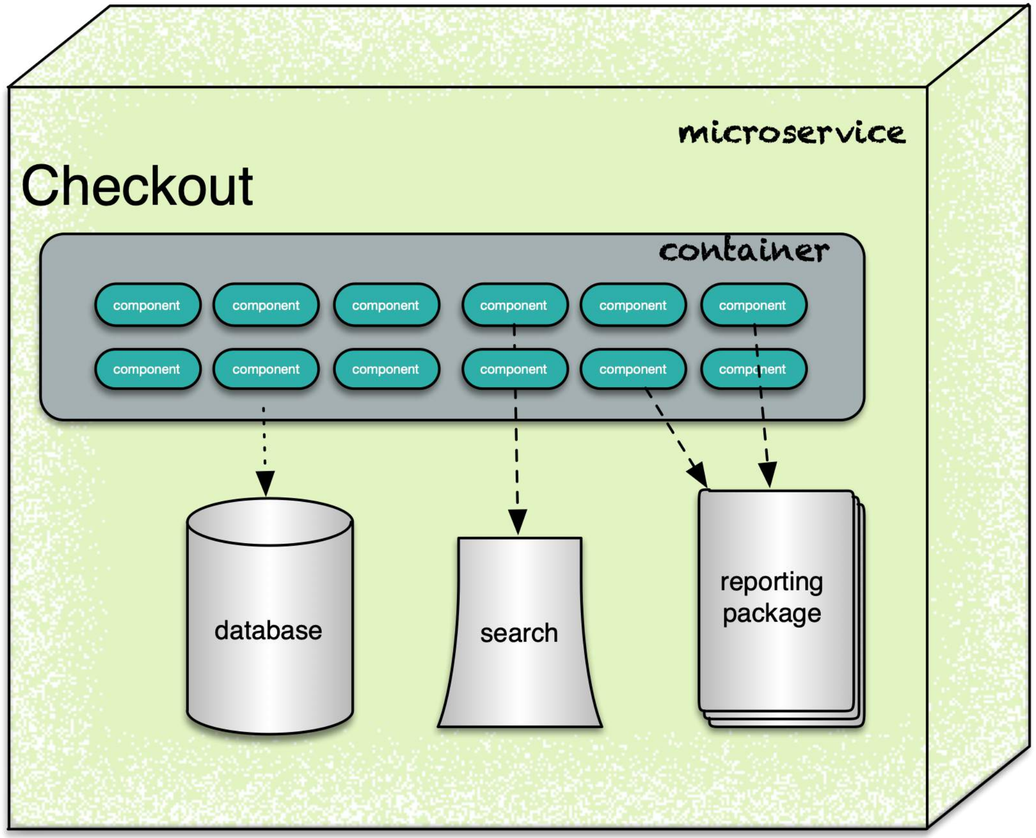
Maintaining Model Integrity



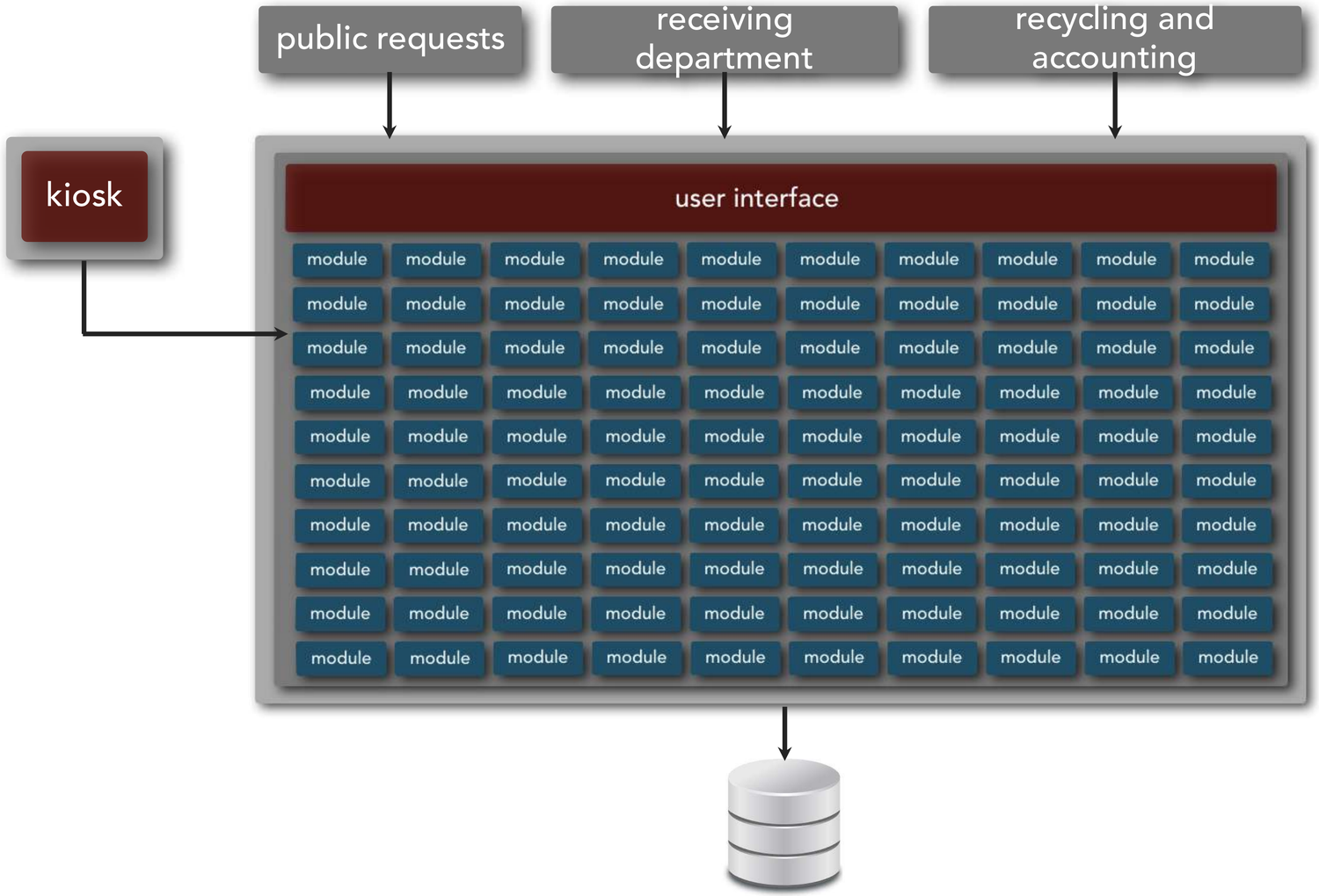
Architectural Quantum

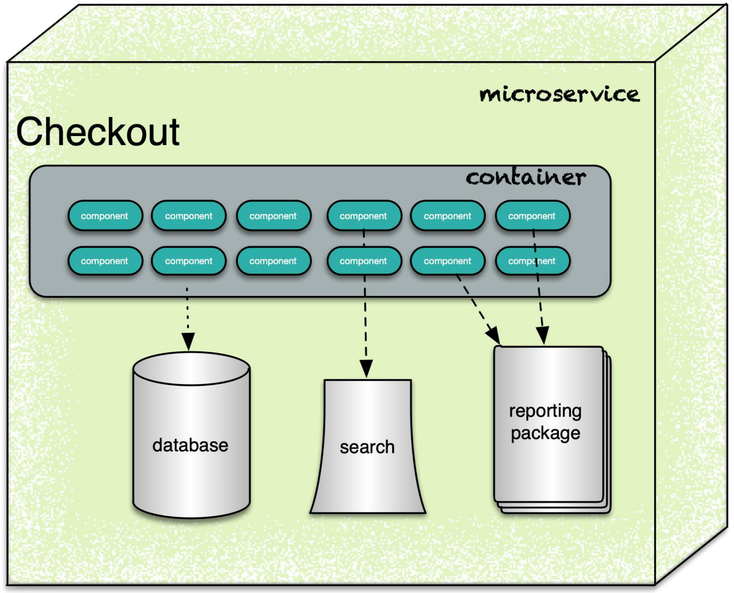
independently deployable artifact with high functional cohesion and synchronous quantum coupling

Independently deployable

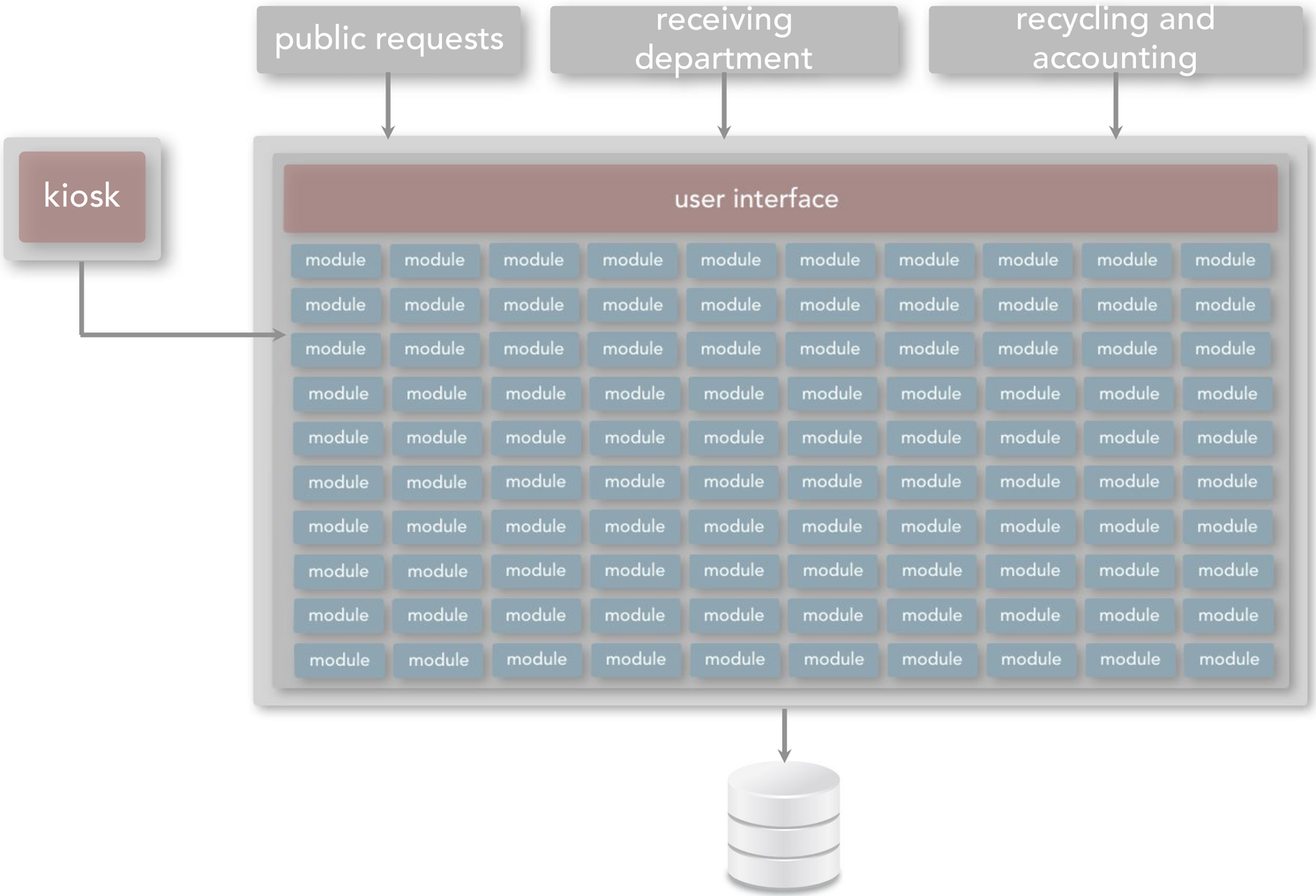


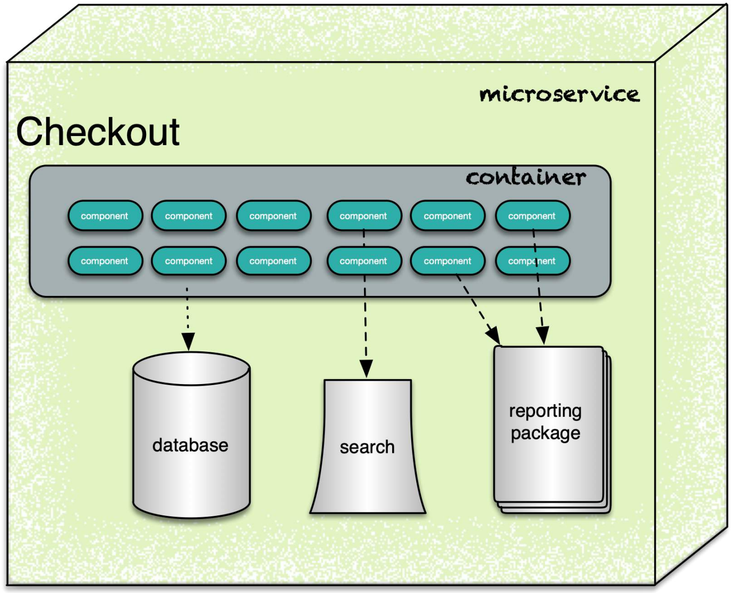
High functional cohesion



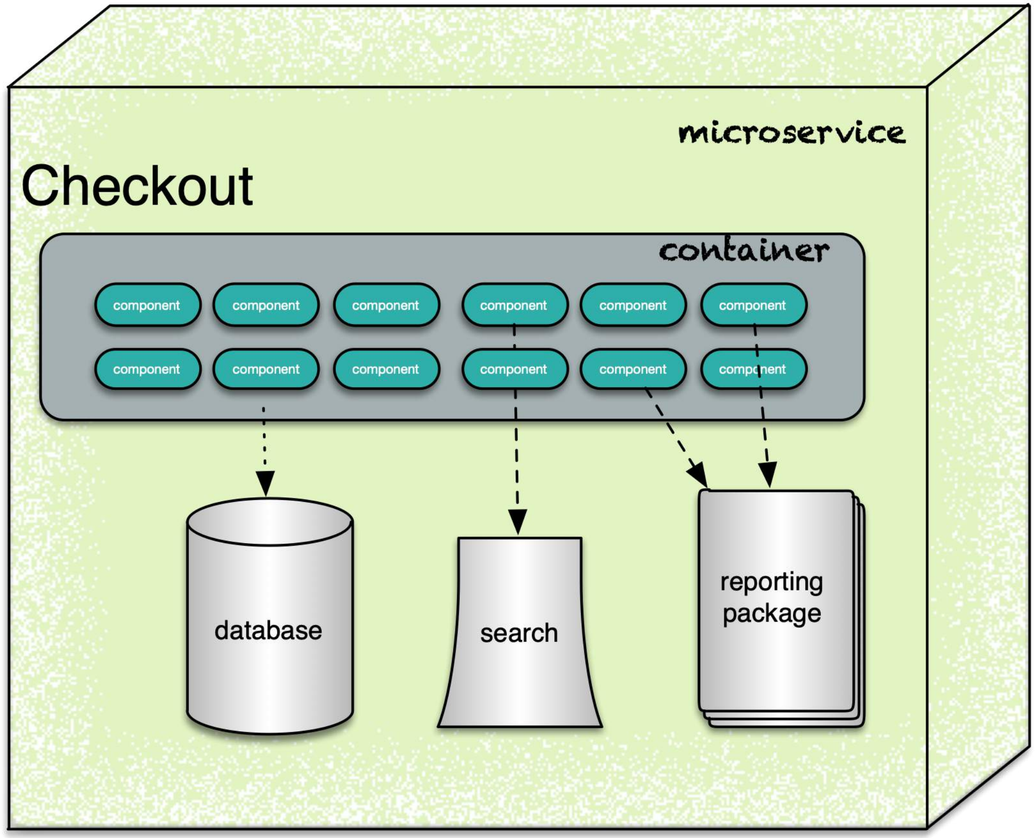


High functional cohesion

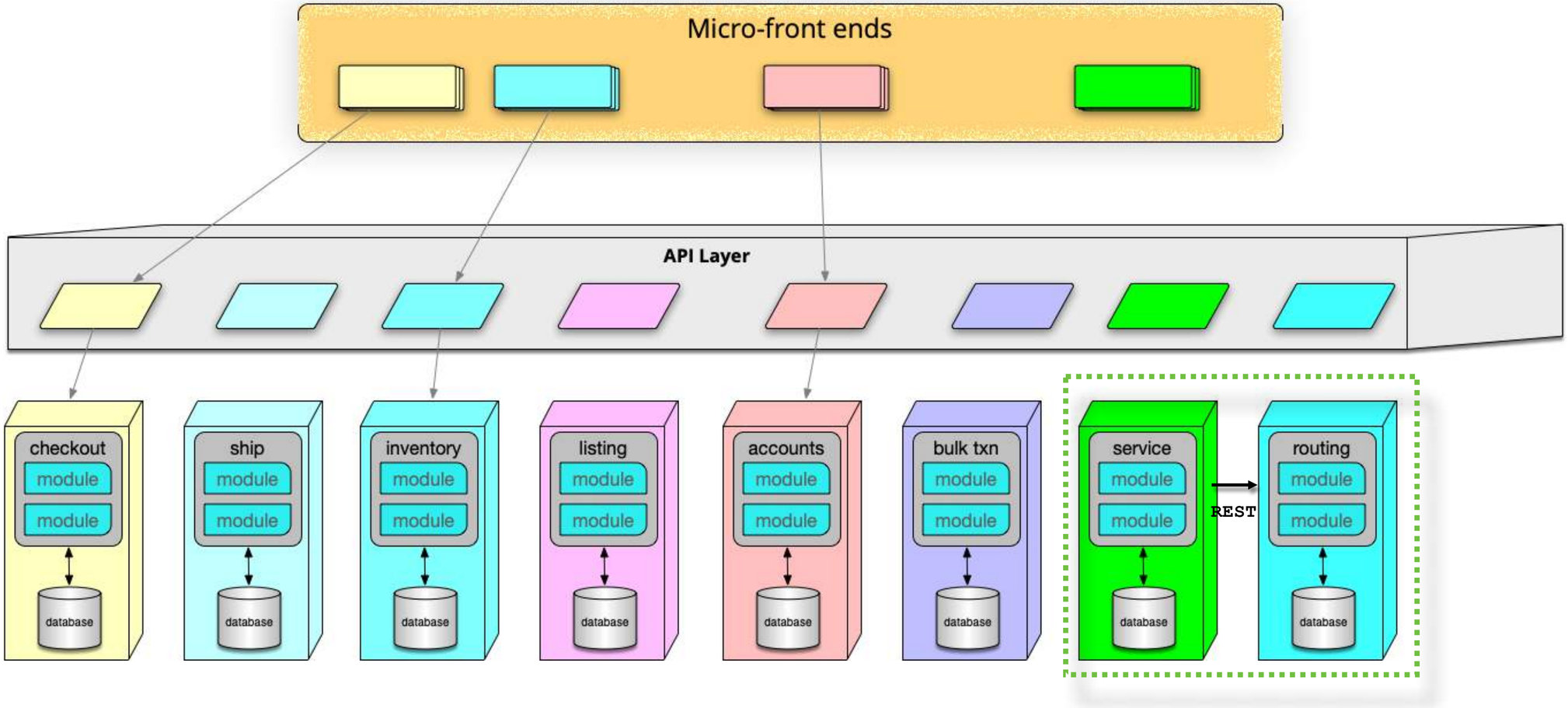




Synchronous connascence

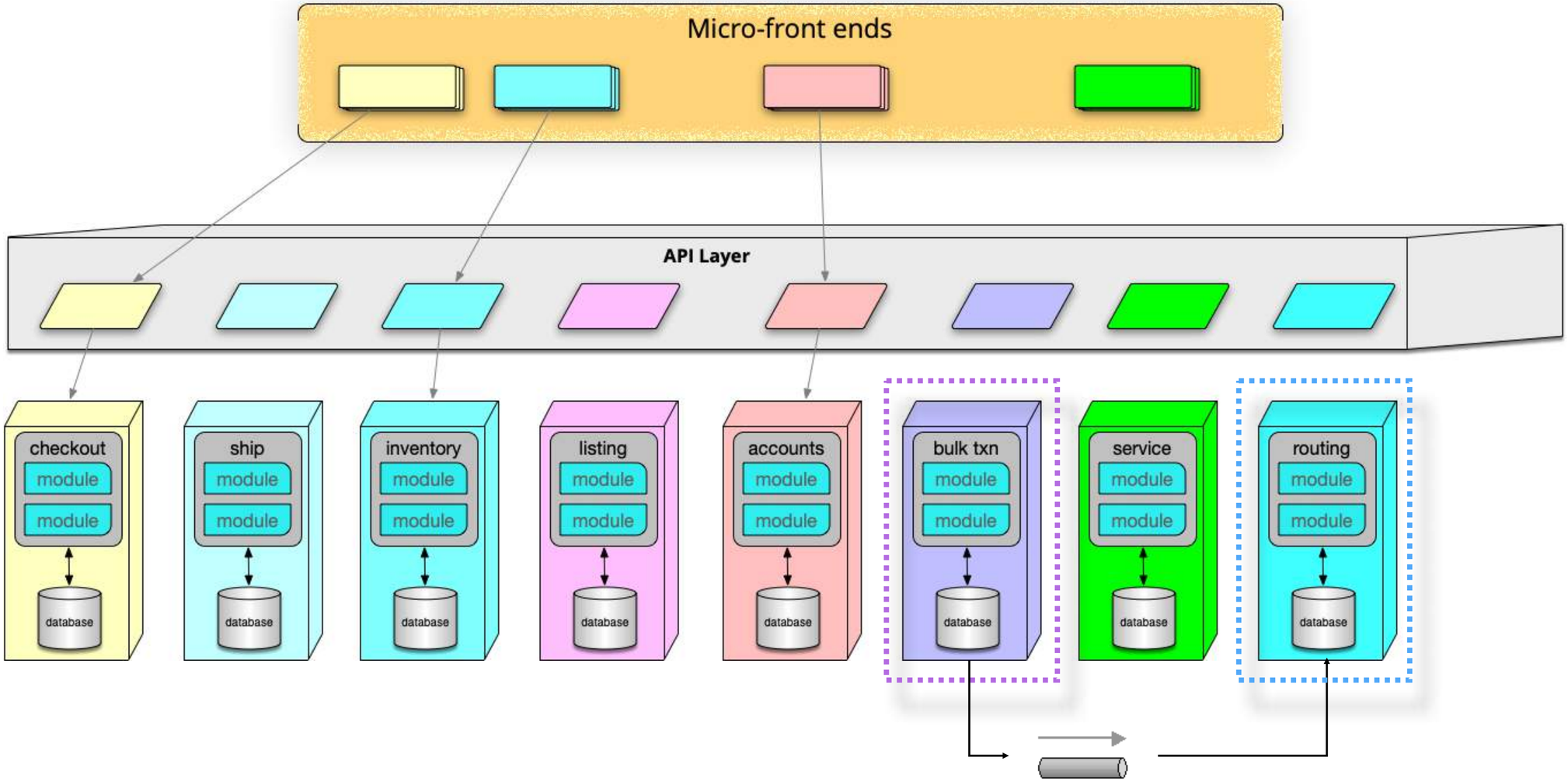


The Value of Asynchronicity



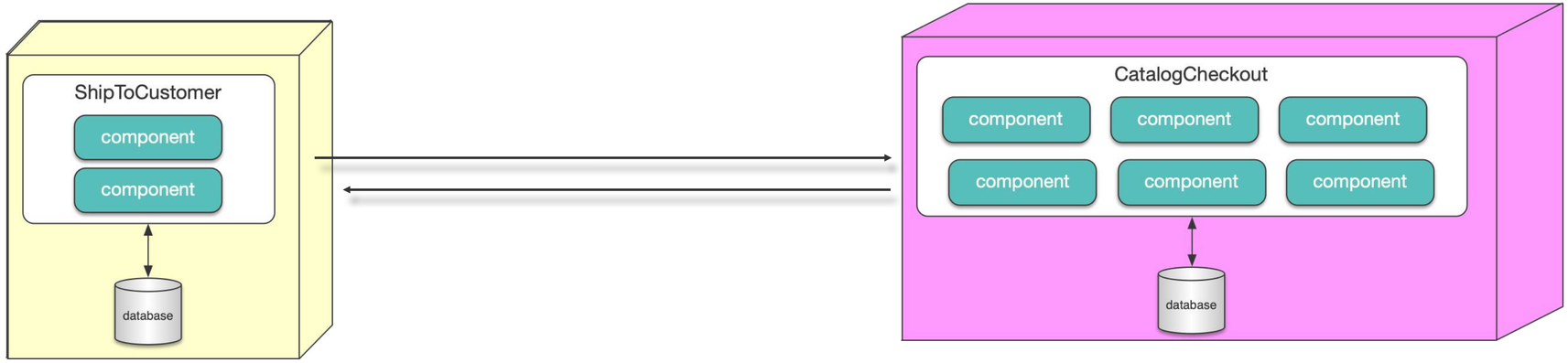
synchronous

The Value of Asynchronicity



asynchronous

Synchronous Coupling

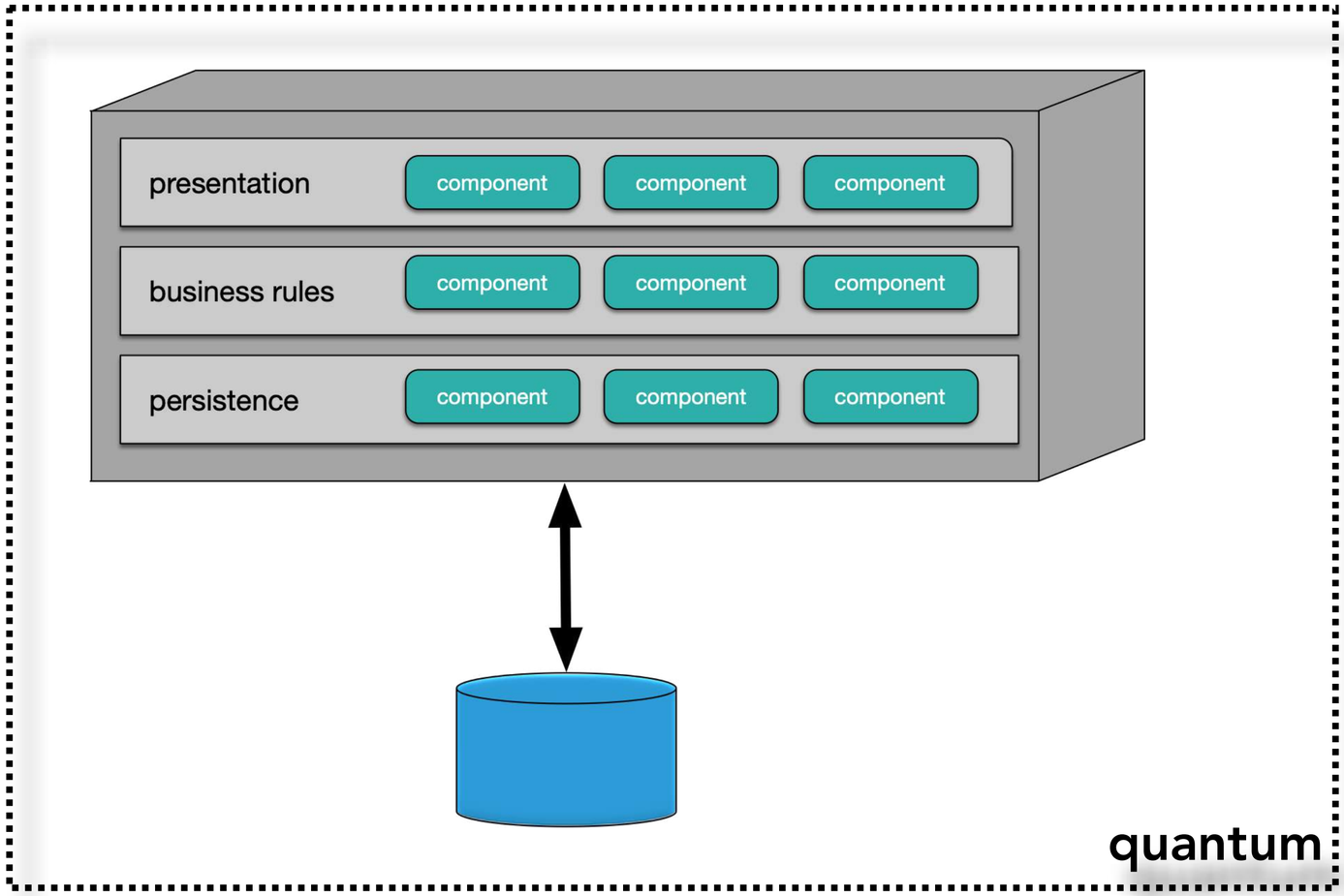


Synchronous calls expand the scope of quanta.

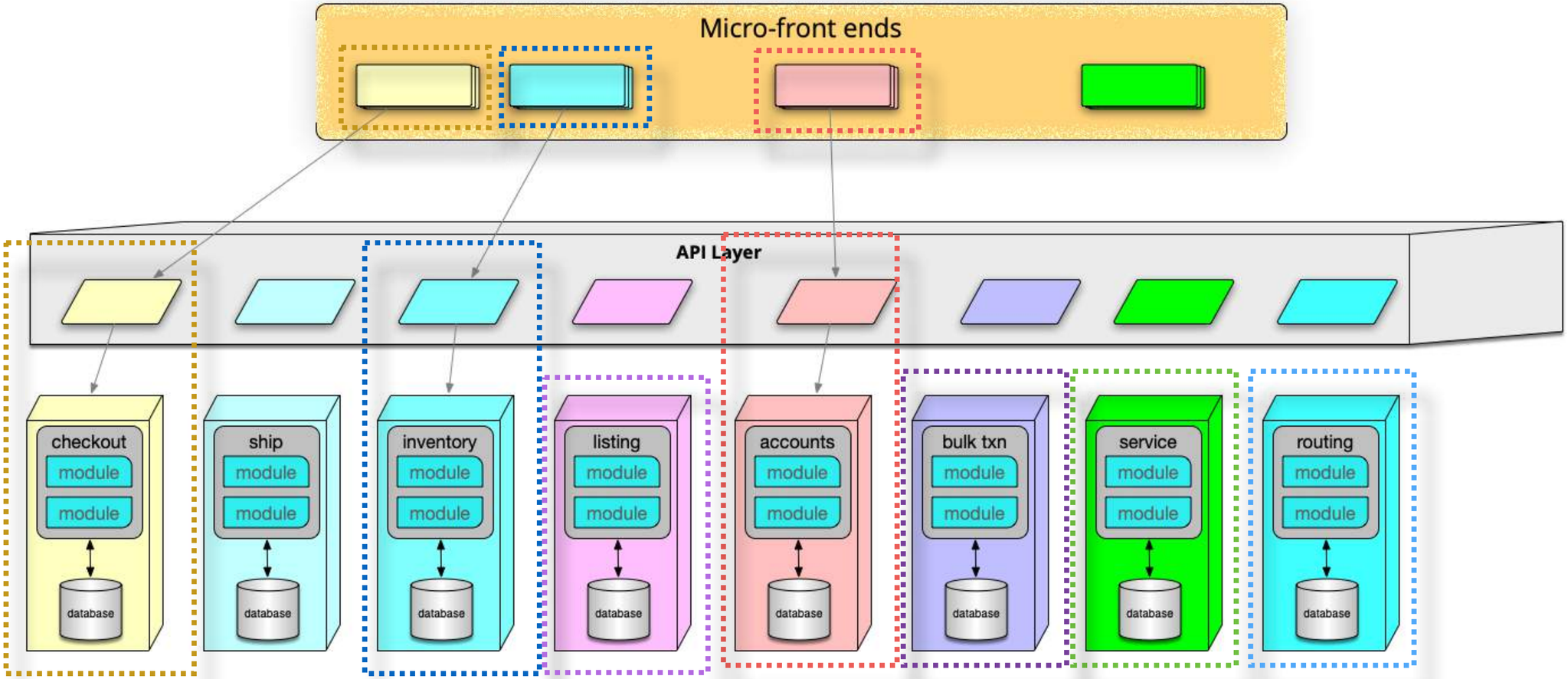
Why Quantum?

- Operational view of architecture
- Holistic view of architecture
  - Databases
  - User interface
- Useful for structural analysis
- ***Where architecture characteristics live***

Tradition Architecture Characteristics Axiom

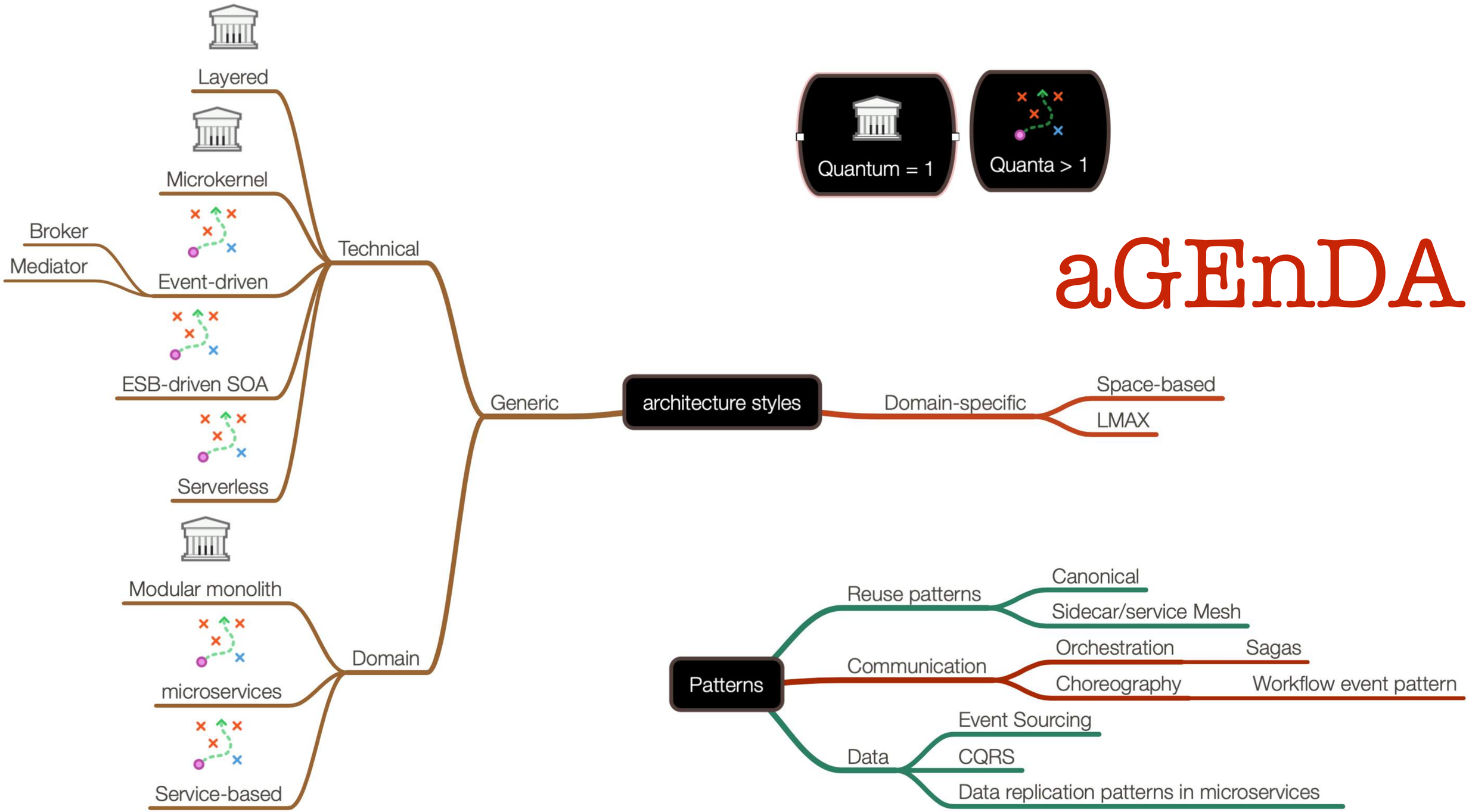


~~Traditional~~ Architecture Characteristics ~~Axiom~~

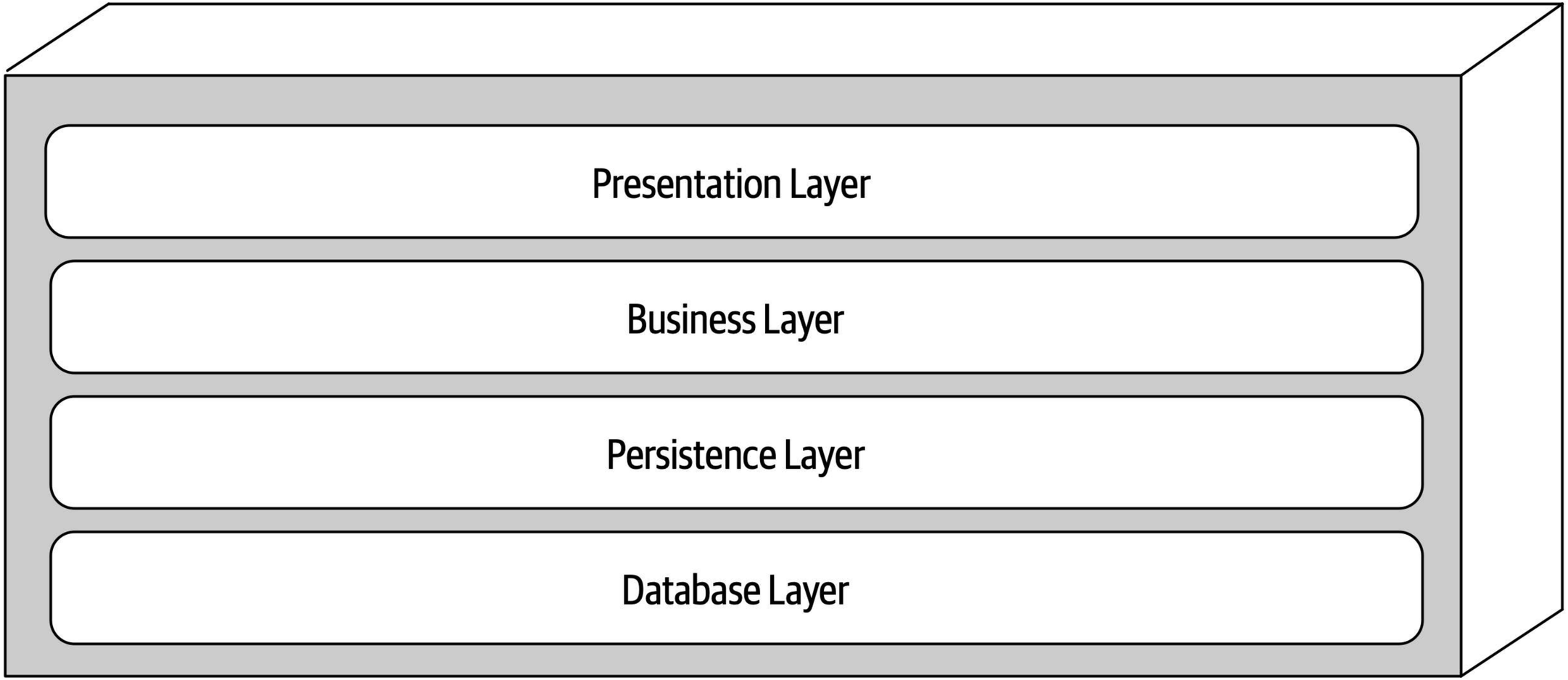


quanta

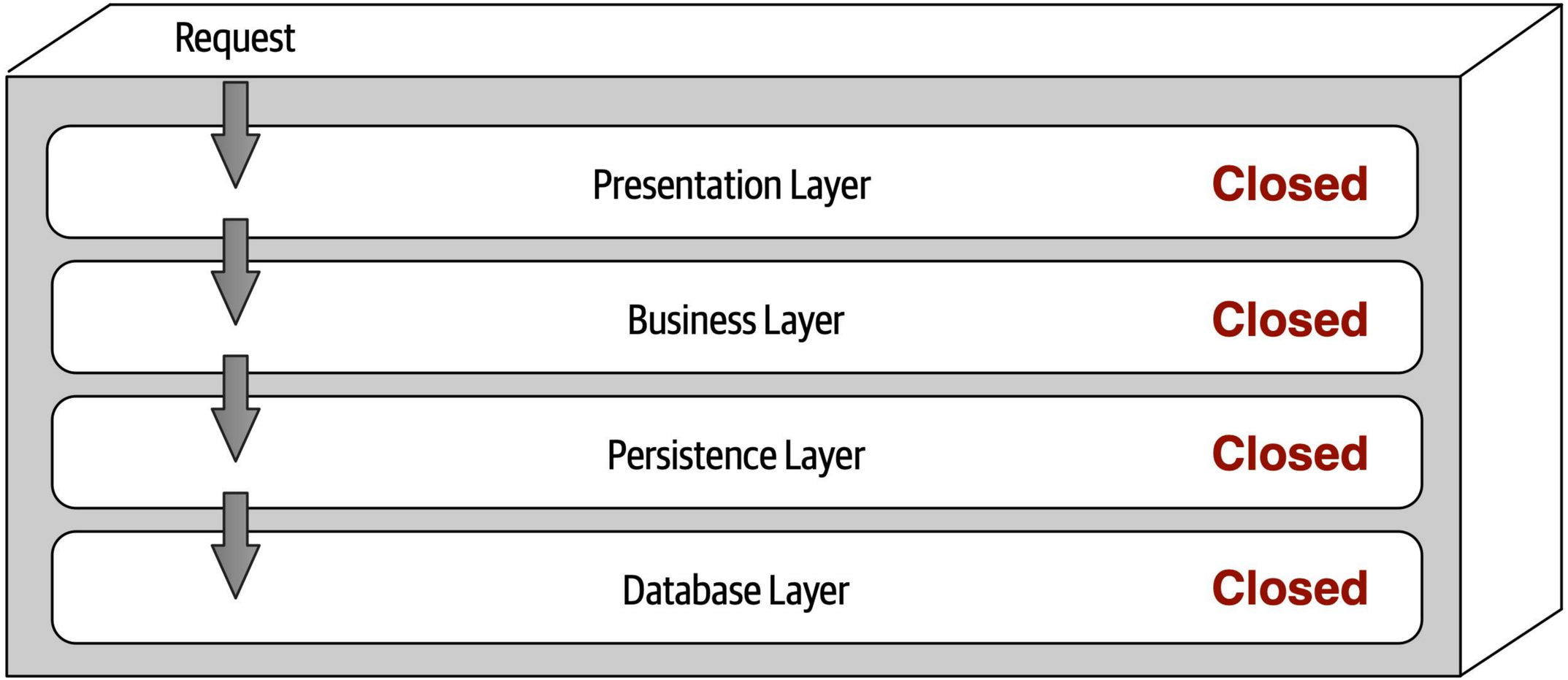
aGEnDA


Quantum = 1
Quanta > 1
architecture styles
Generic
Technical
Layered
Microkernel
Event-driven
Broker
Mediator
ESB-driven SOA
Serverless
Domain
Modular monolith
microservices
Service-based
Domain-specific
Space-based
LMAX
Patterns
Reuse patterns
Canonical
Sidecar/service Mesh
Communication
Orchestration
Sagas
Choreography
Workflow event pattern
Data
Event Sourcing
CQRS
Data replication patterns in microservices

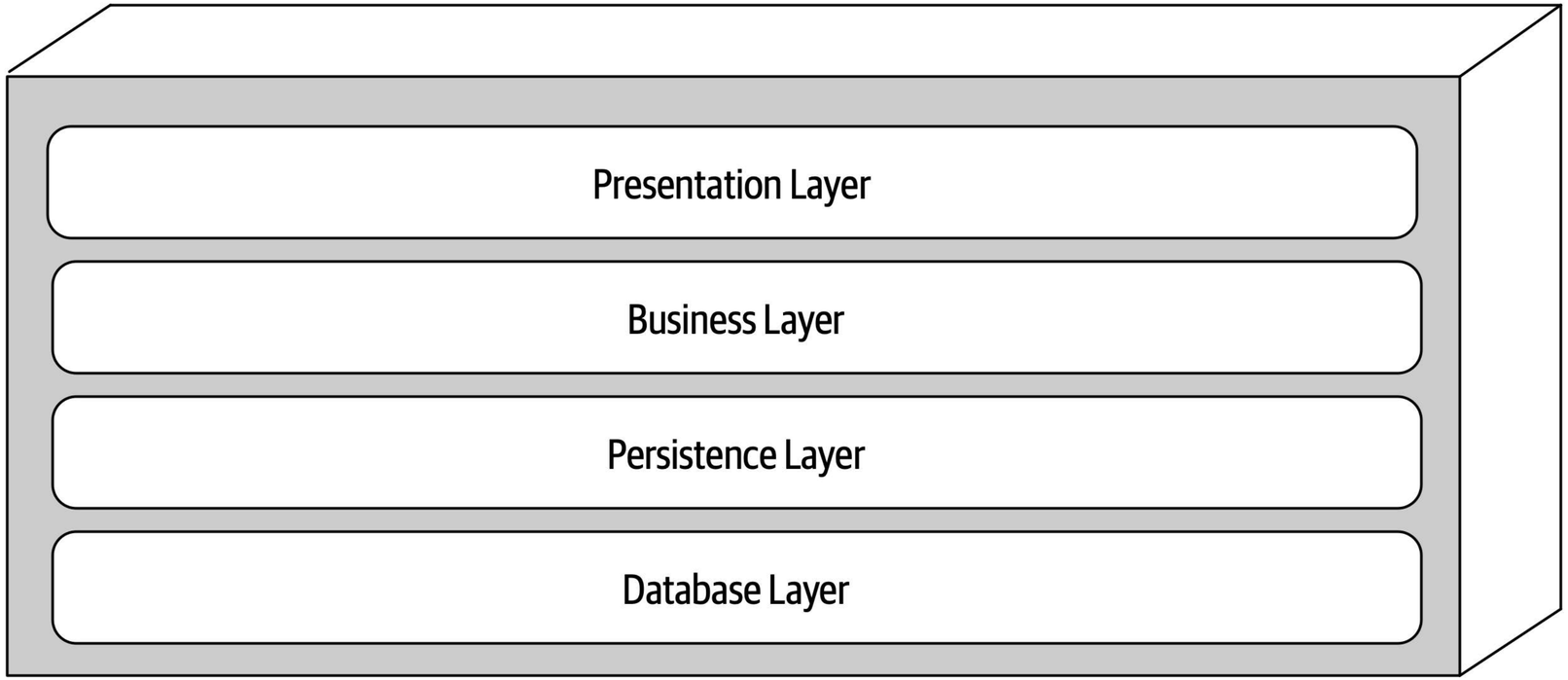
Layered Architecture



Layered Architecture



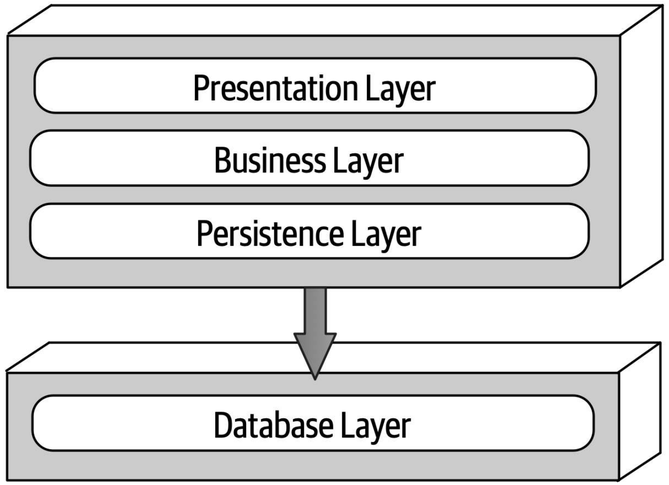
Layered Architecture

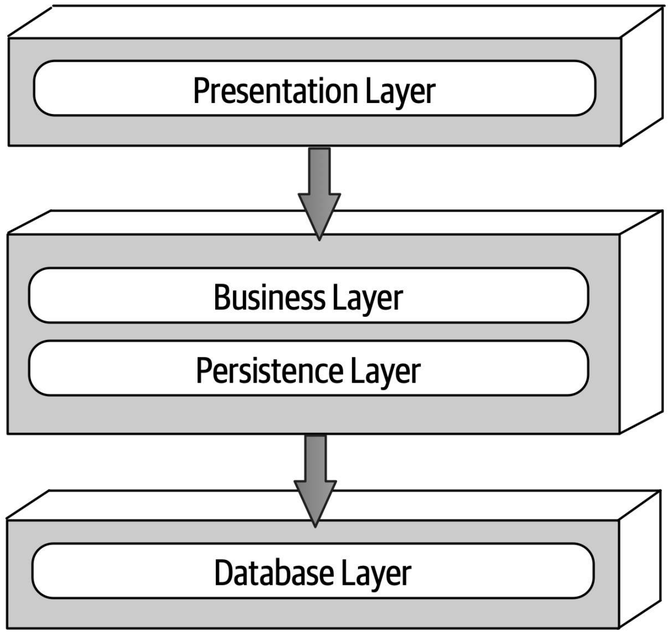


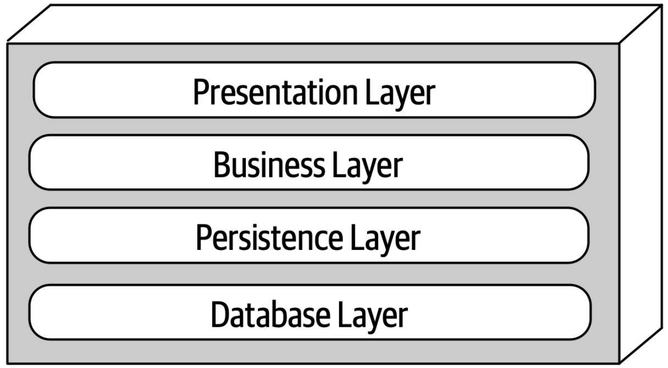
separation of concerns

Layered Architecture

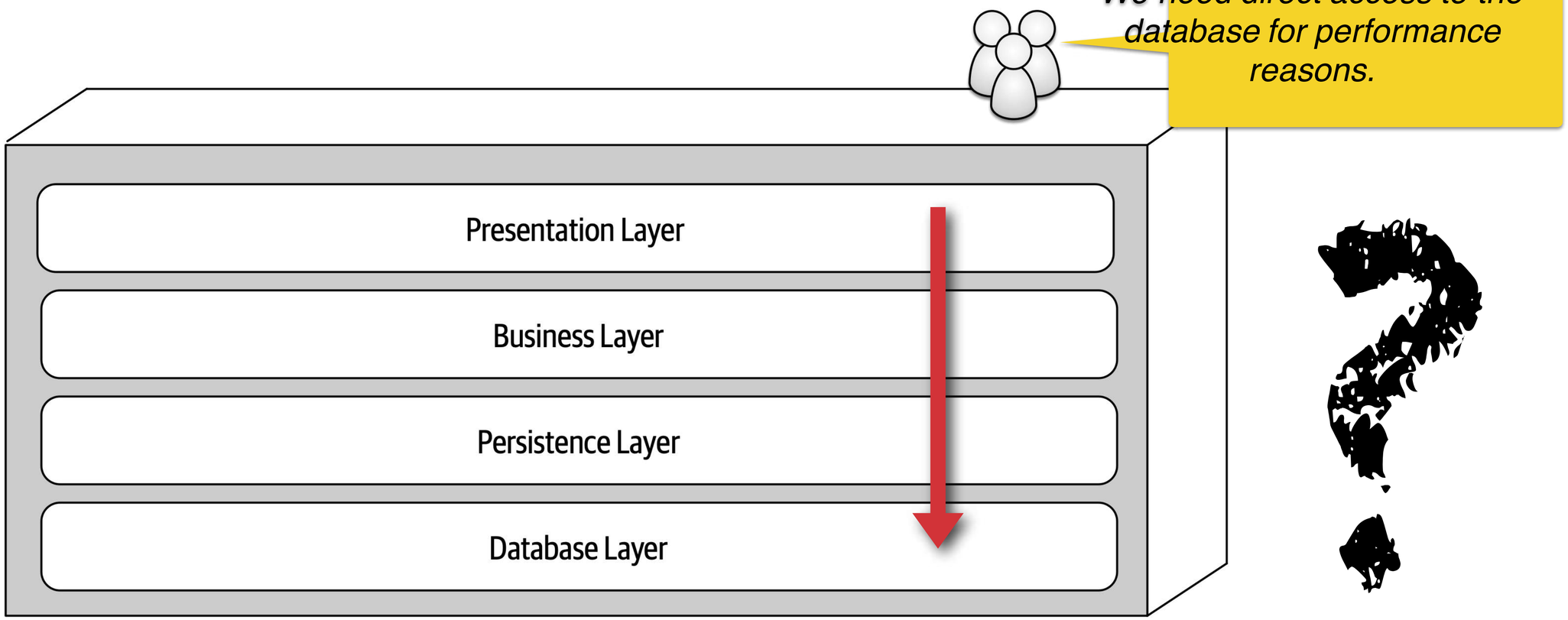
hybrids and variants

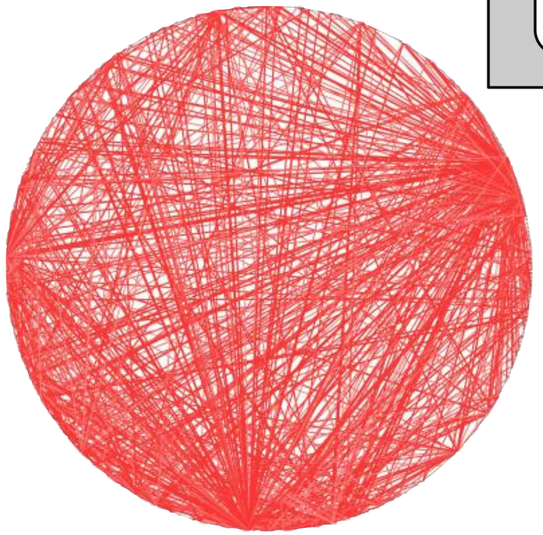






Pattern Governance





ArchUnit

https://www.archunit.org/

```java
public class LayerDependencyRulesTest {
    private JavaClasses classes;

    @Before
    public void setUp() throws Exception {
        classes = new ClassFileImporter().importPackagesOf(ClassViolatingCodingRules.class);
    }

    @Test
    public void services_should_not_access_controllers() {
        noClasses().that().resideInAPackage("..service..")
                .should().accessClassesThat().resideInAPackage("..controller..").check(classes);
    }

    @Test
    public void persistence_should_not_access_services() {
        noClasses().that().resideInAPackage("..persistence..")
                .should().accessClassesThat().resideInAPackage("..service..").check(classes);
    }

    @Test
    public void services_should_only_be_accessed_by_controllers_or_other_services() {
        classes().that().resideInAPackage("..service..")
                .should().onlyBeAccessed().byAnyPackage("..controller..", "..service..").check(classes);
    }
}
```

layer dependency

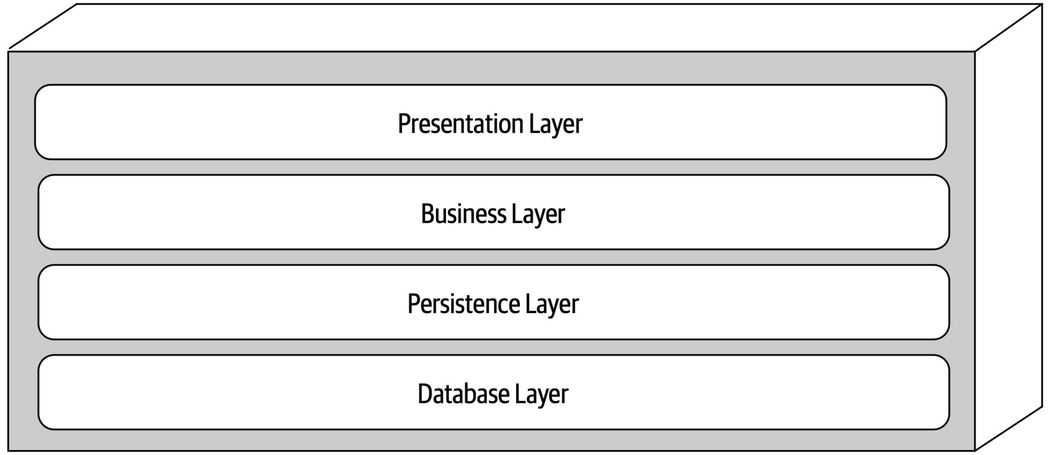
NetArchTest

https://github.com/BenMorris/NetArchTest/

```
// Controllers should not directly reference repositories
var result = Types.InCurrentDomain()
    .That()
    .ResideInNamespace("NetArchTest.SampleLibrary.Presentation")
    .ShouldNot()
    .HaveDependencyOn("NetArchTest.SampleLibrary.Data")
    .GetResult().IsSuccessful;
```

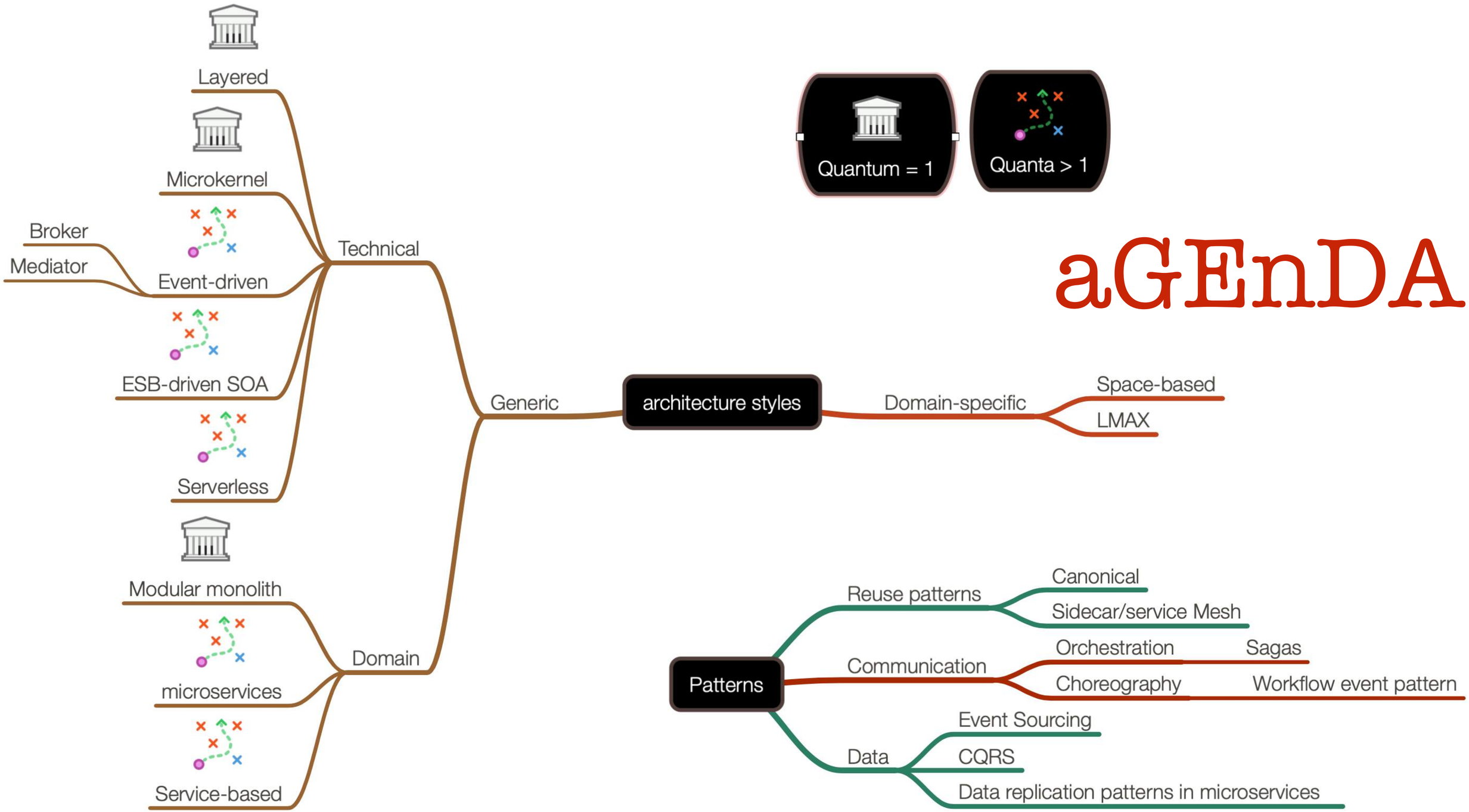
Layered Architecture

drivers



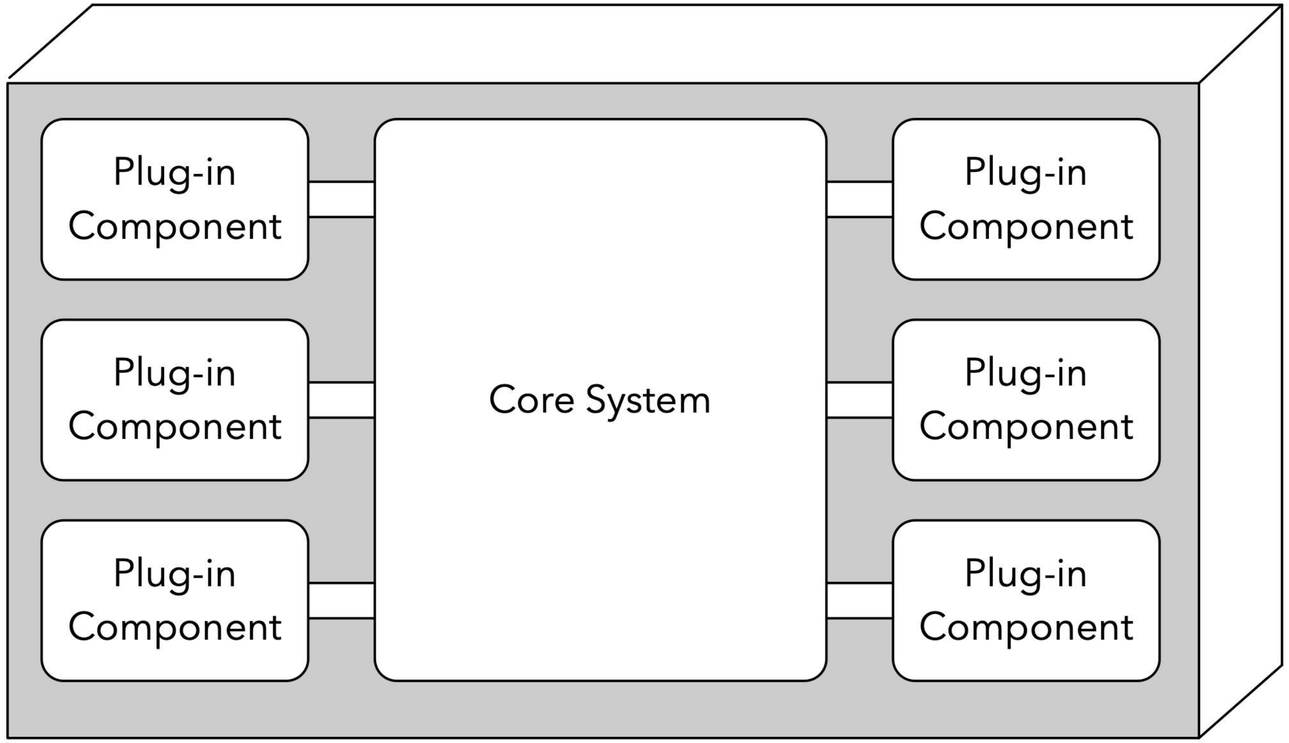
- ✓ simplicity
- ✓ cost

aGEnDA


Quantum = 1
Quanta > 1
architecture styles
Generic
Technical
Layered
Microkernel
Event-driven
Broker
Mediator
ESB-driven SOA
Serverless
Domain
Modular monolith
microservices
Service-based
Domain-specific
Space-based
LMAX
Patterns
Reuse patterns
Canonical
Sidecar/service Mesh
Communication
Orchestration
Sagas
Choreography
Workflow event pattern
Data
Event Sourcing
CQRS
Data replication patterns in microservices

Microkernel Architecture

(a.k.a. plug-in architecture pattern)



Microkernel Architecture

architectural components

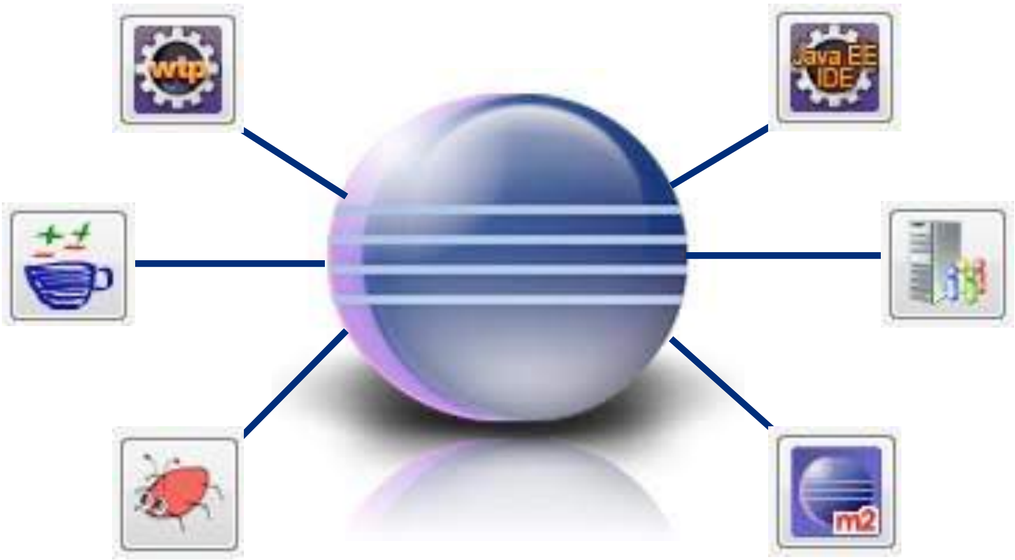
core system

- minimal functionality to run system
- general business rules and logic
- no custom processing

plug-in module

- standalone independent module
- specific additional rules or logic

Microkernel Architecture

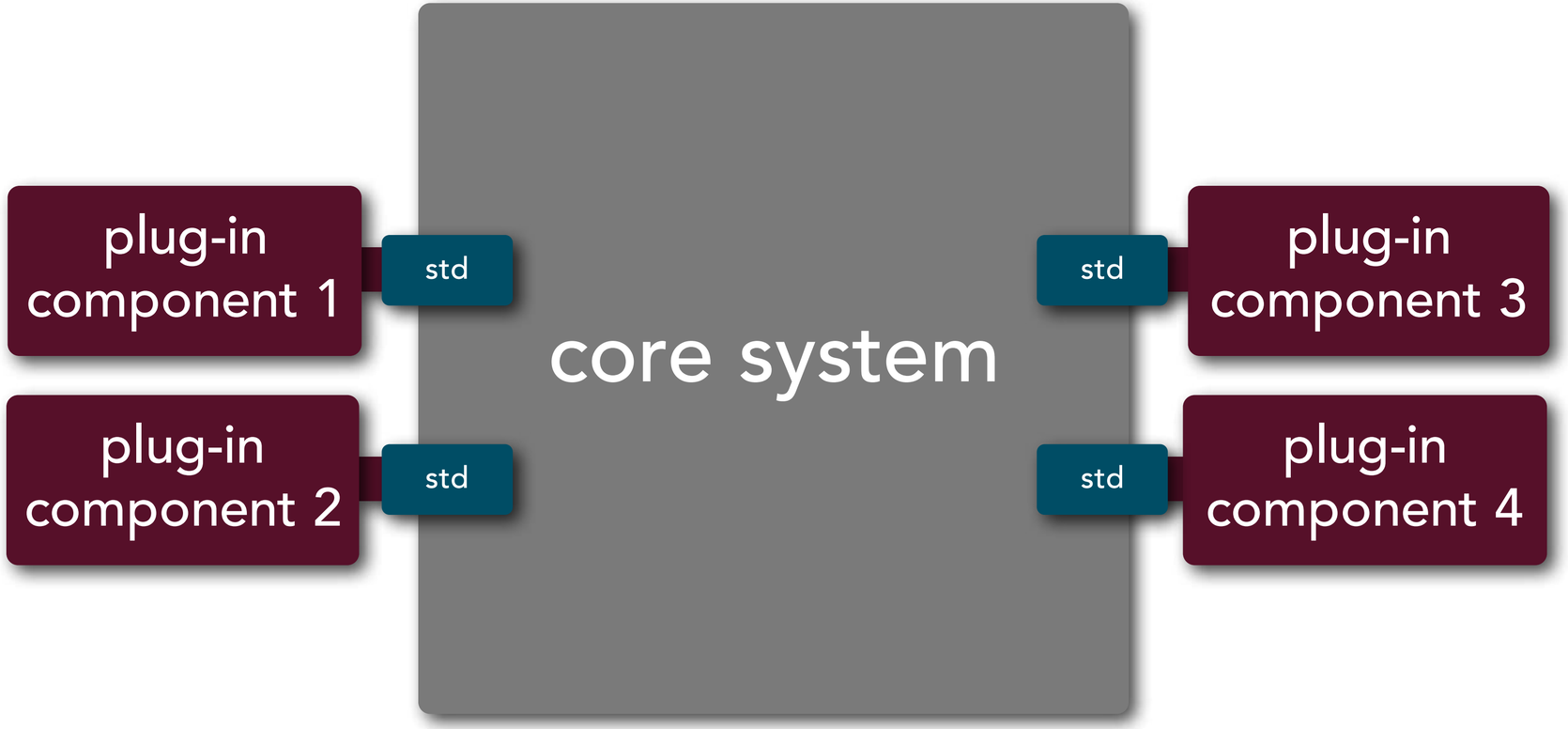


Microkernel Architecture

claims processing

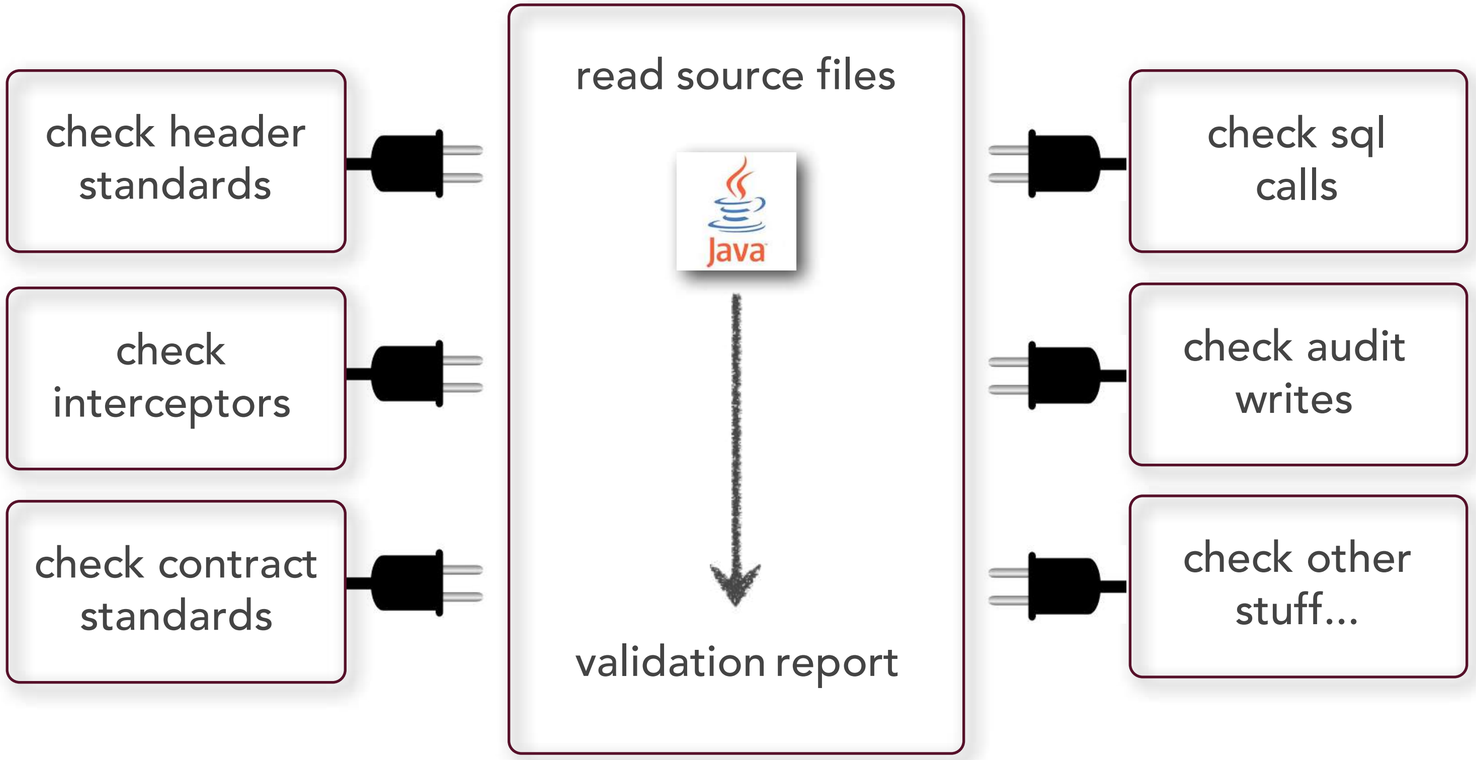
Microkernel Architecture

plug-in contracts



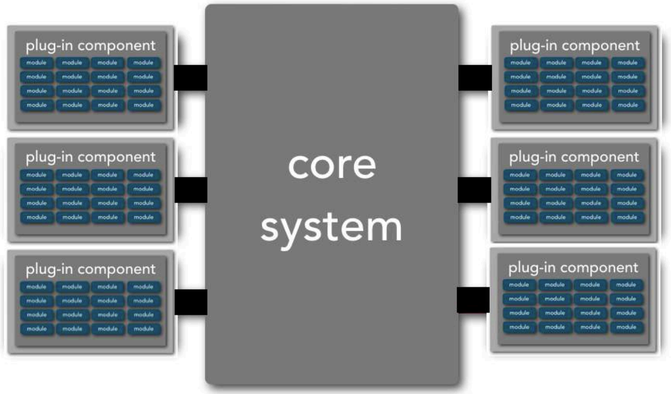
Microkernel Architecture

source validation tool



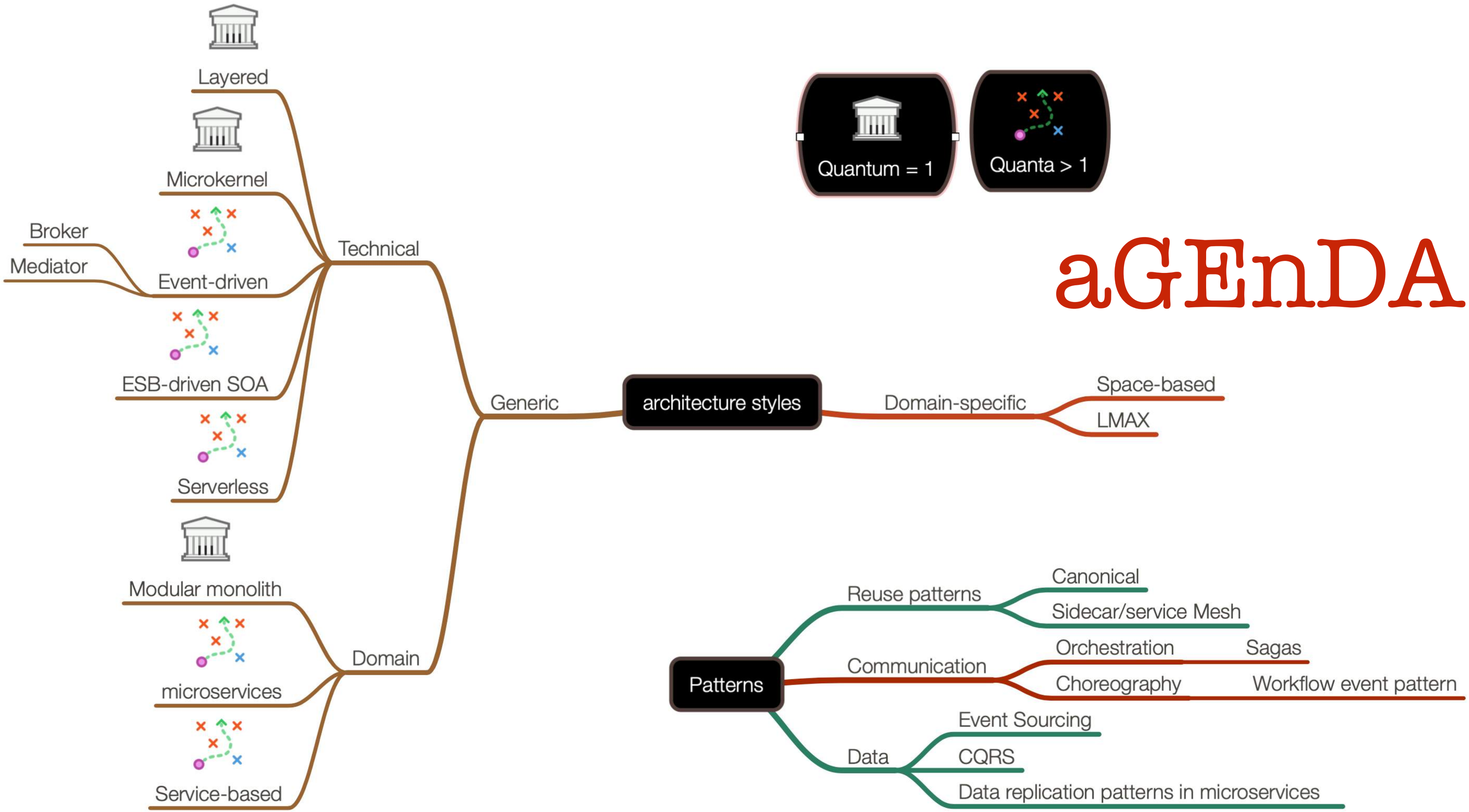
Microkernel Architecture

drivers

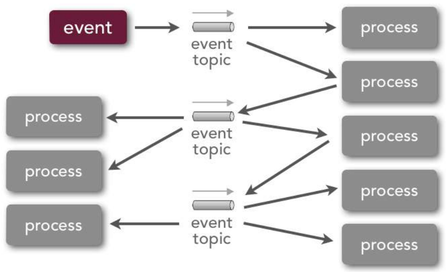


- ✓ modularity
- ✓ simplicity
- ✓ cost
- ✓ adaptability
- ✓ evolvability
- ✓ agility
- ✓ testability
- ✓ deployability

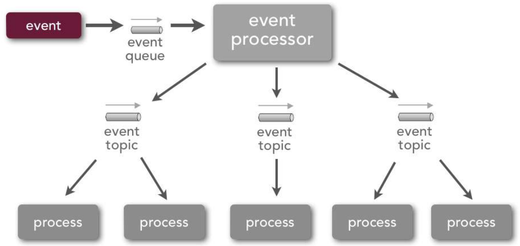
aGEnDA


Quantum = 1
Quanta > 1
architecture styles
Generic
Technical
Layered
Microkernel
Event-driven
Broker
Mediator
ESB-driven SOA
Serverless
Domain
Modular monolith
microservices
Service-based
Domain-specific
Space-based
LMAX
Patterns
Reuse patterns
Canonical
Sidecar/service Mesh
Communication
Orchestration
Sagas
Choreography
Workflow event pattern
Data
Event Sourcing
CQRS
Data replication patterns in microservices

Event-driven Architecture



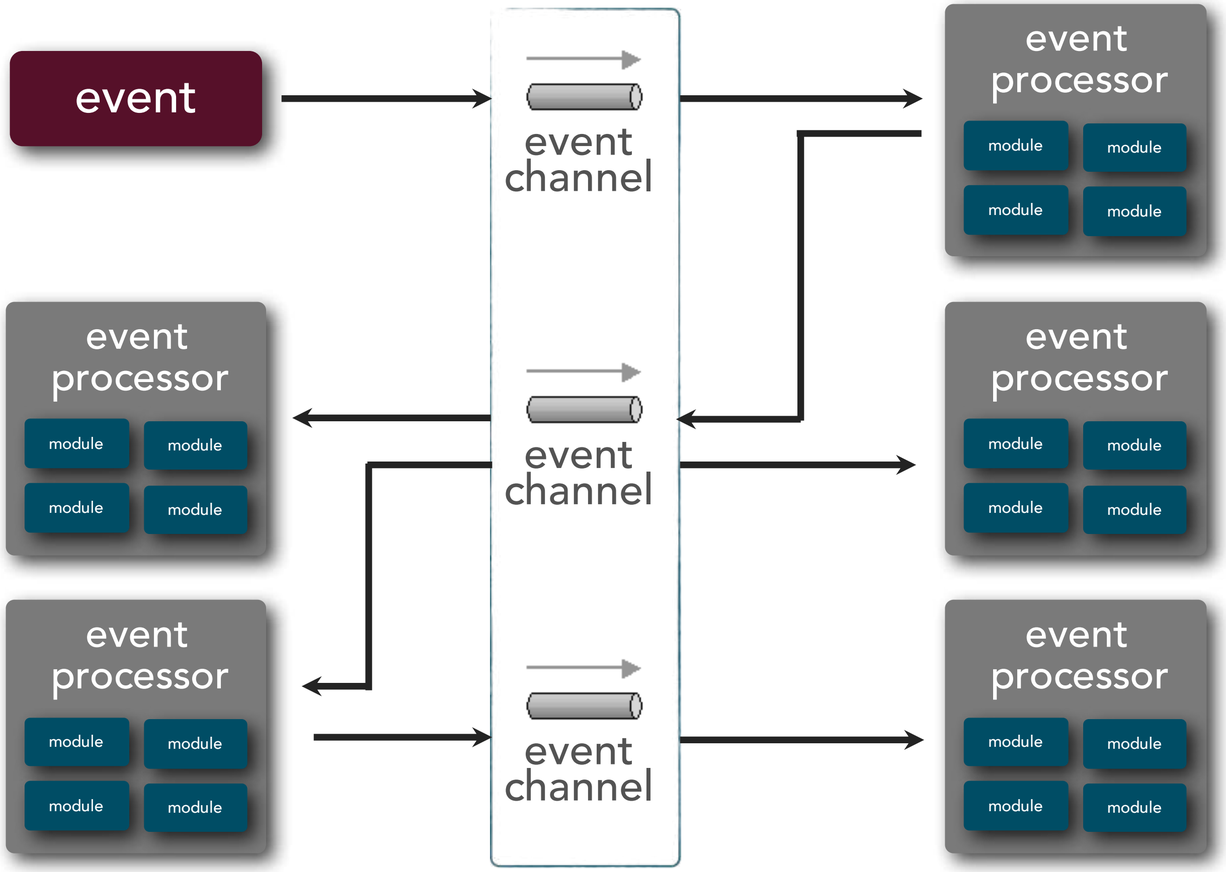
broker topology



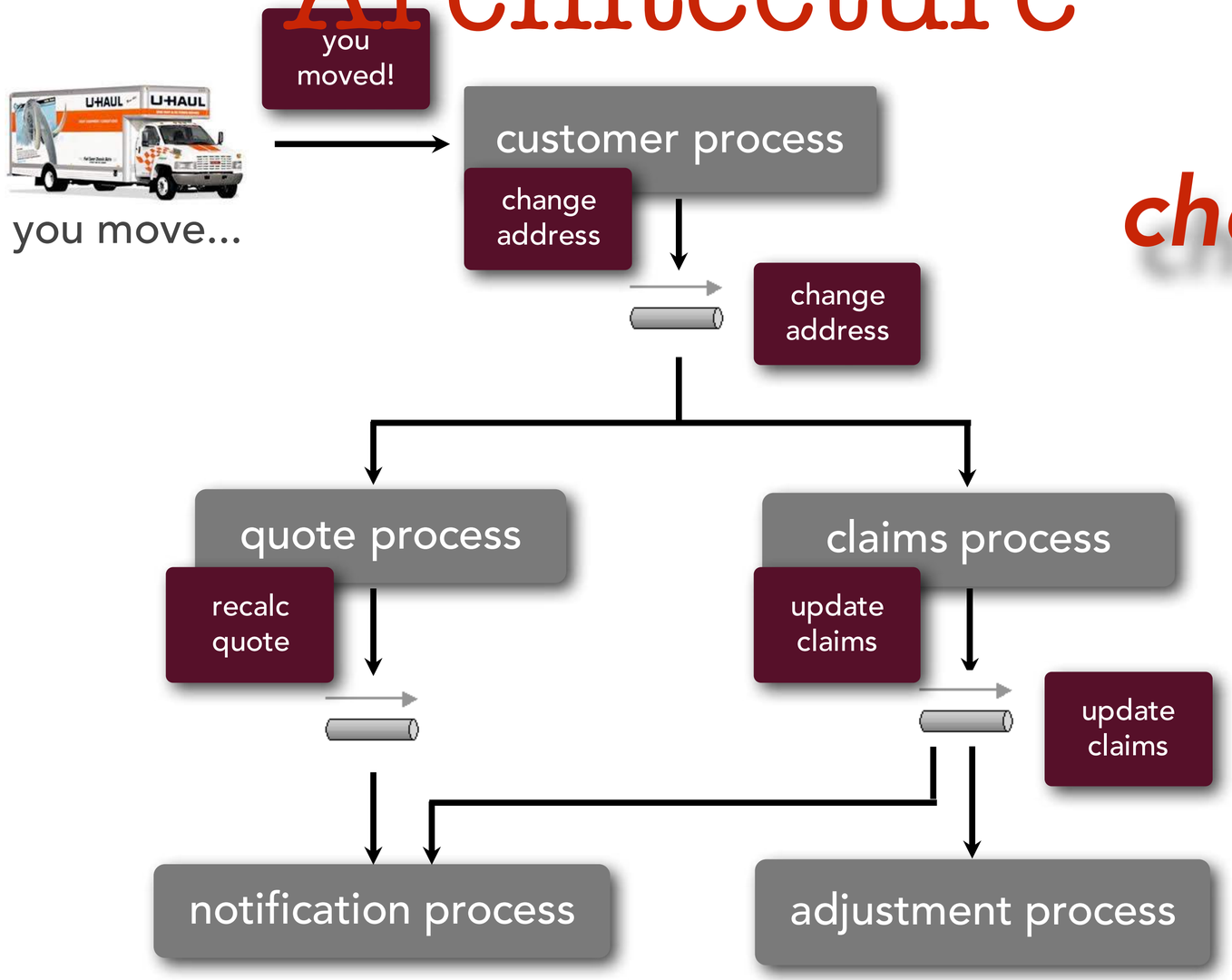
mediator topology

Event-driven Architecture

broker topology

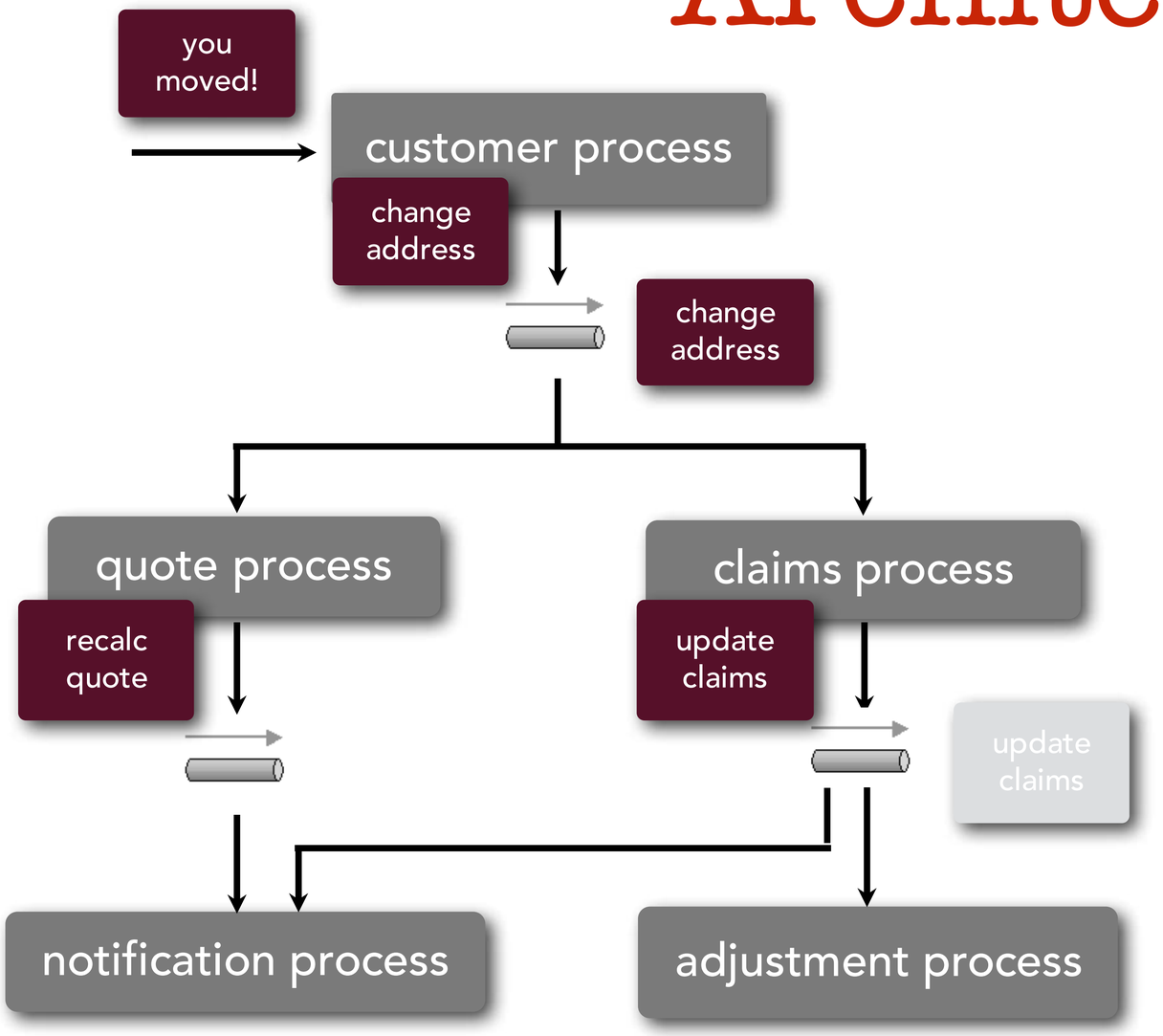


Event-driven Architecture



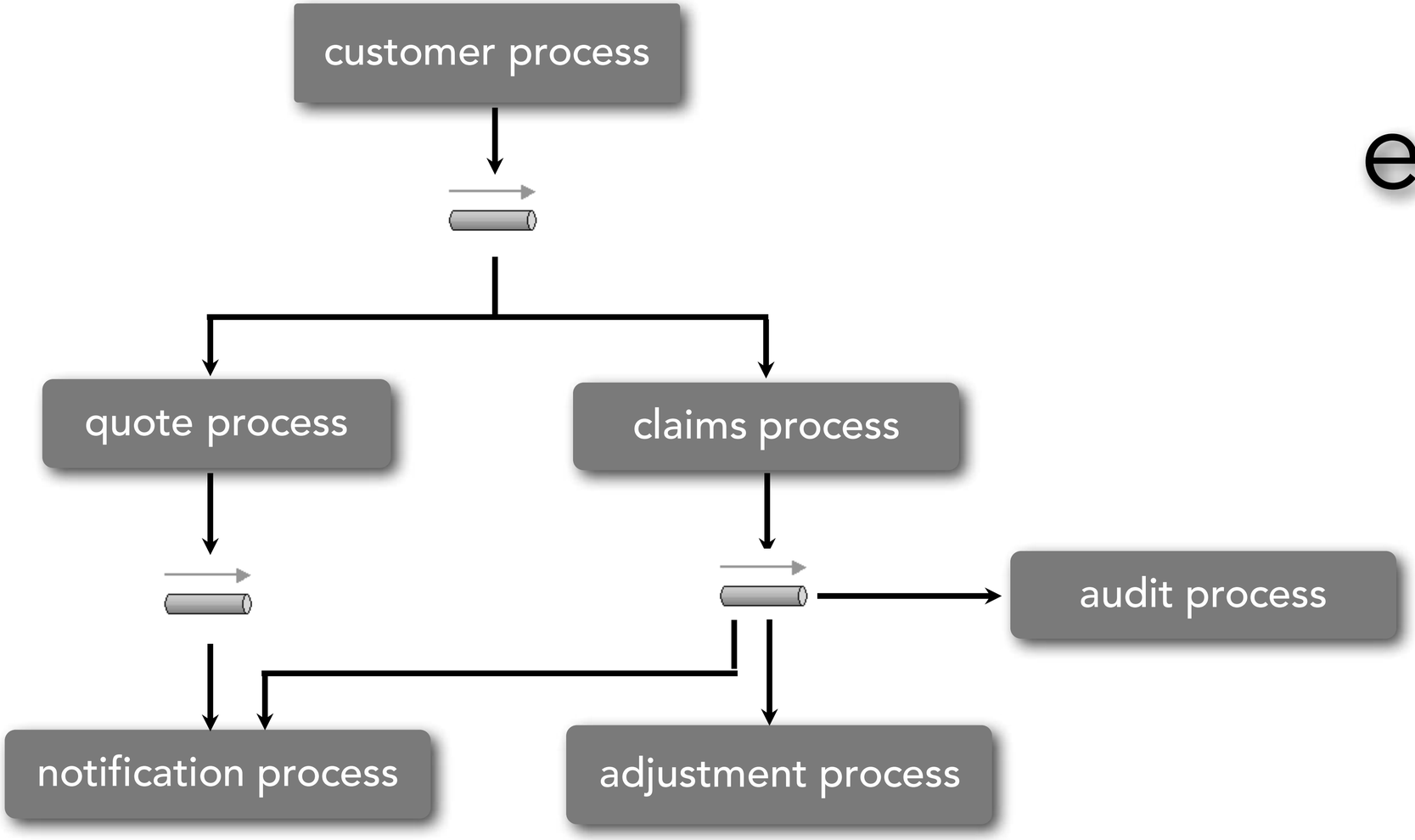
choreography

Event-driven Architecture



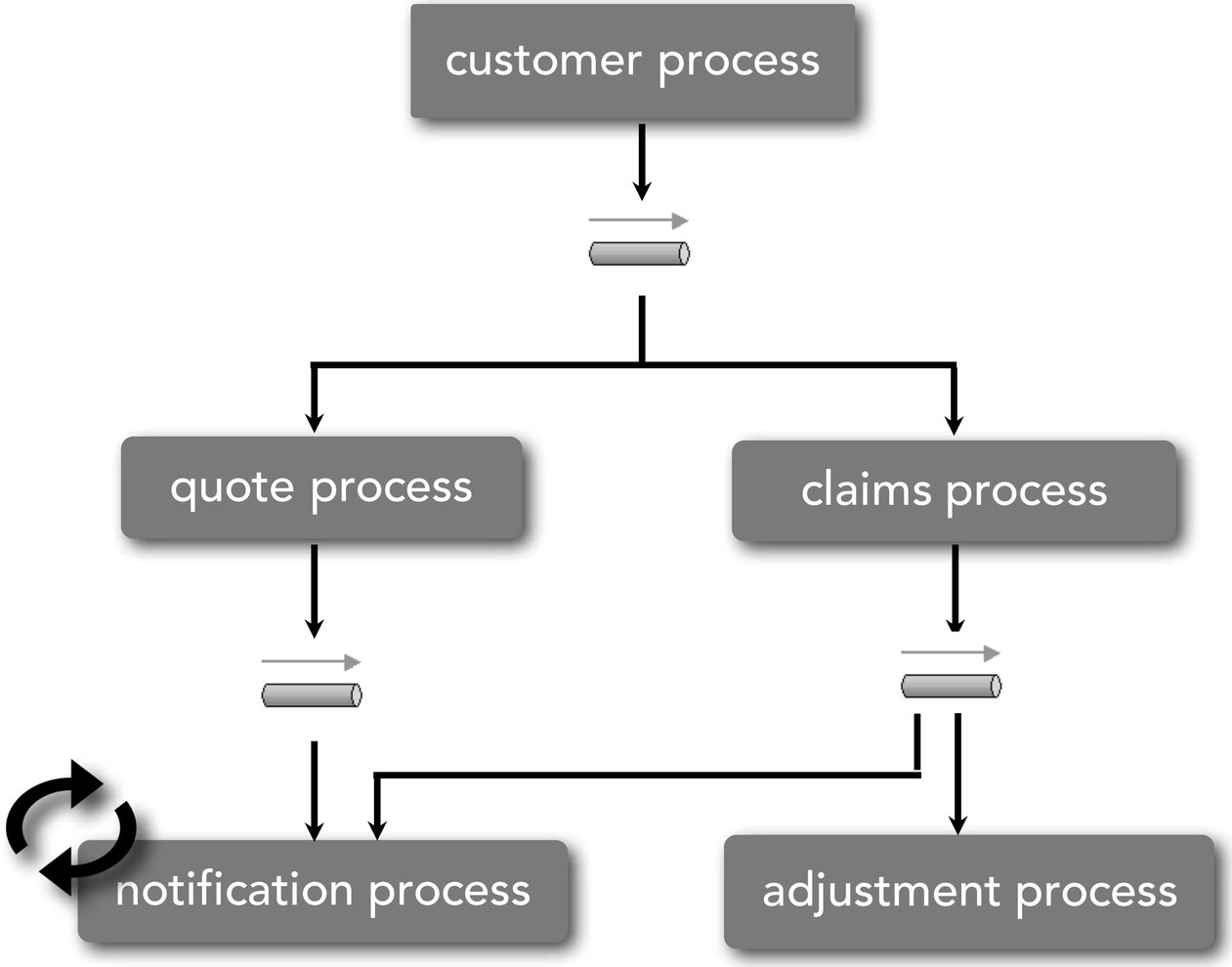
Error handling?

Event-driven Architecture



evolution

Event-driven Architecture

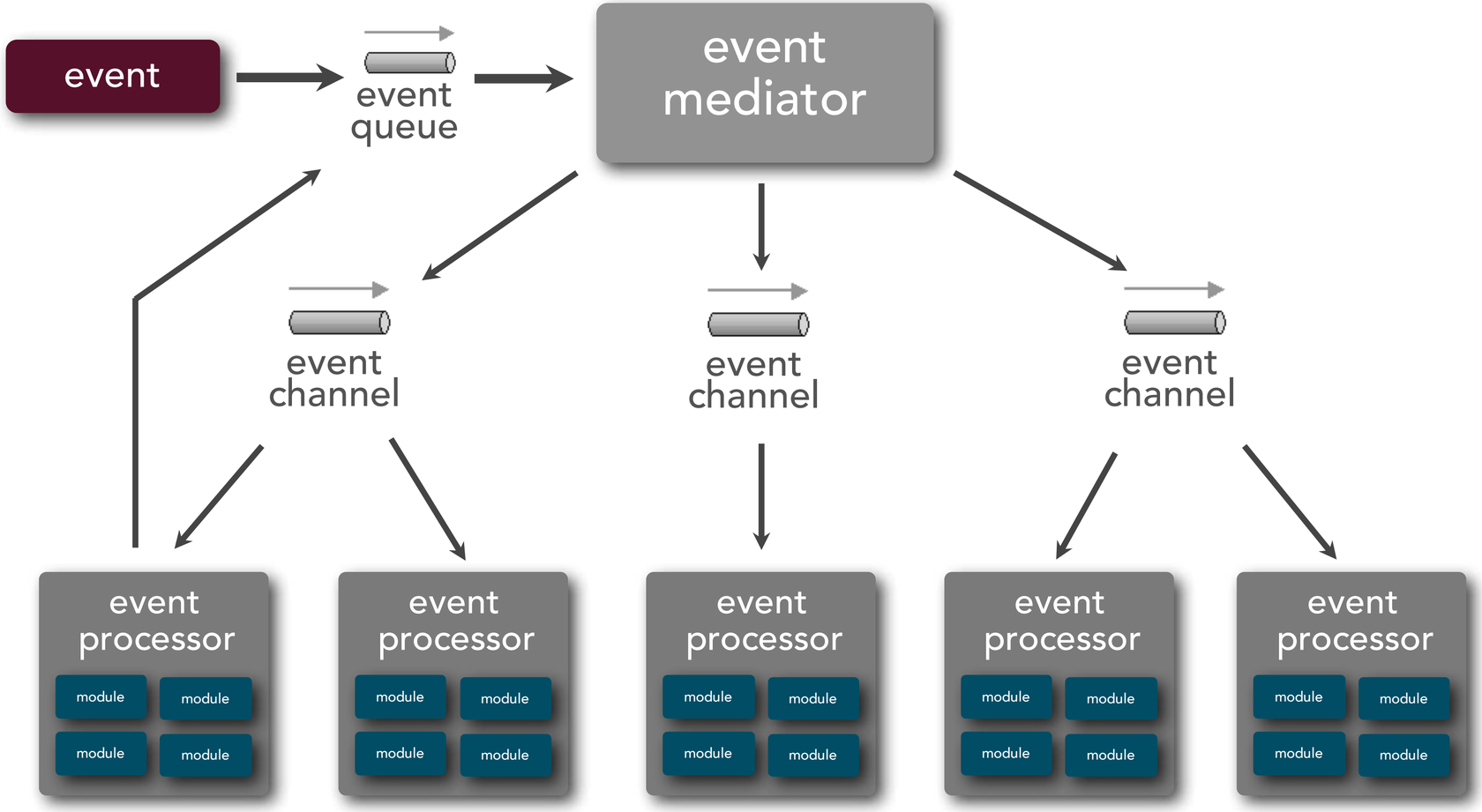


Unified notification message to customer?

In asynchronous systems, coordination happens at the end

Event-driven Architecture

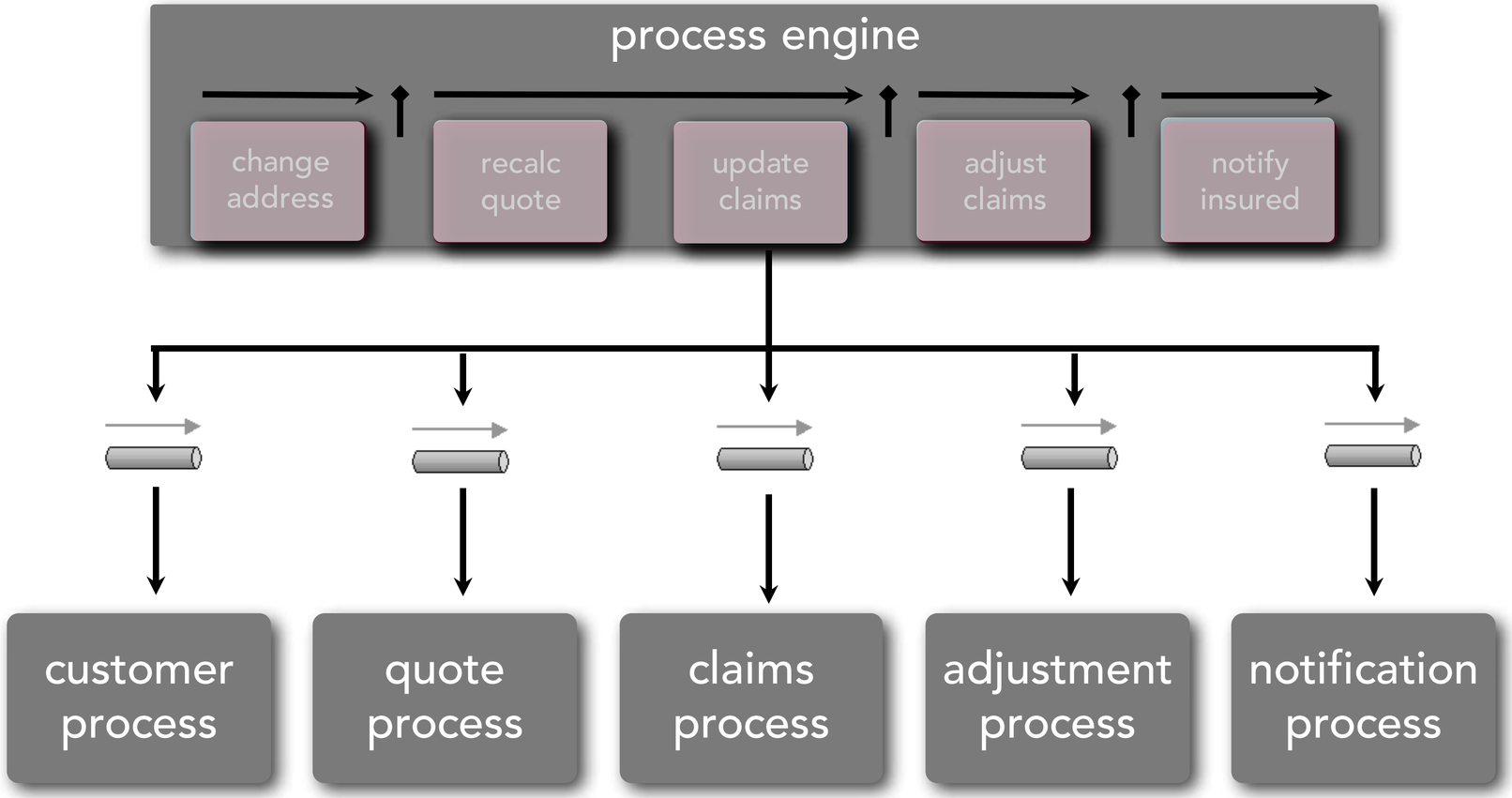
mediator topology



Event-driven Architecture

you move...

you moved

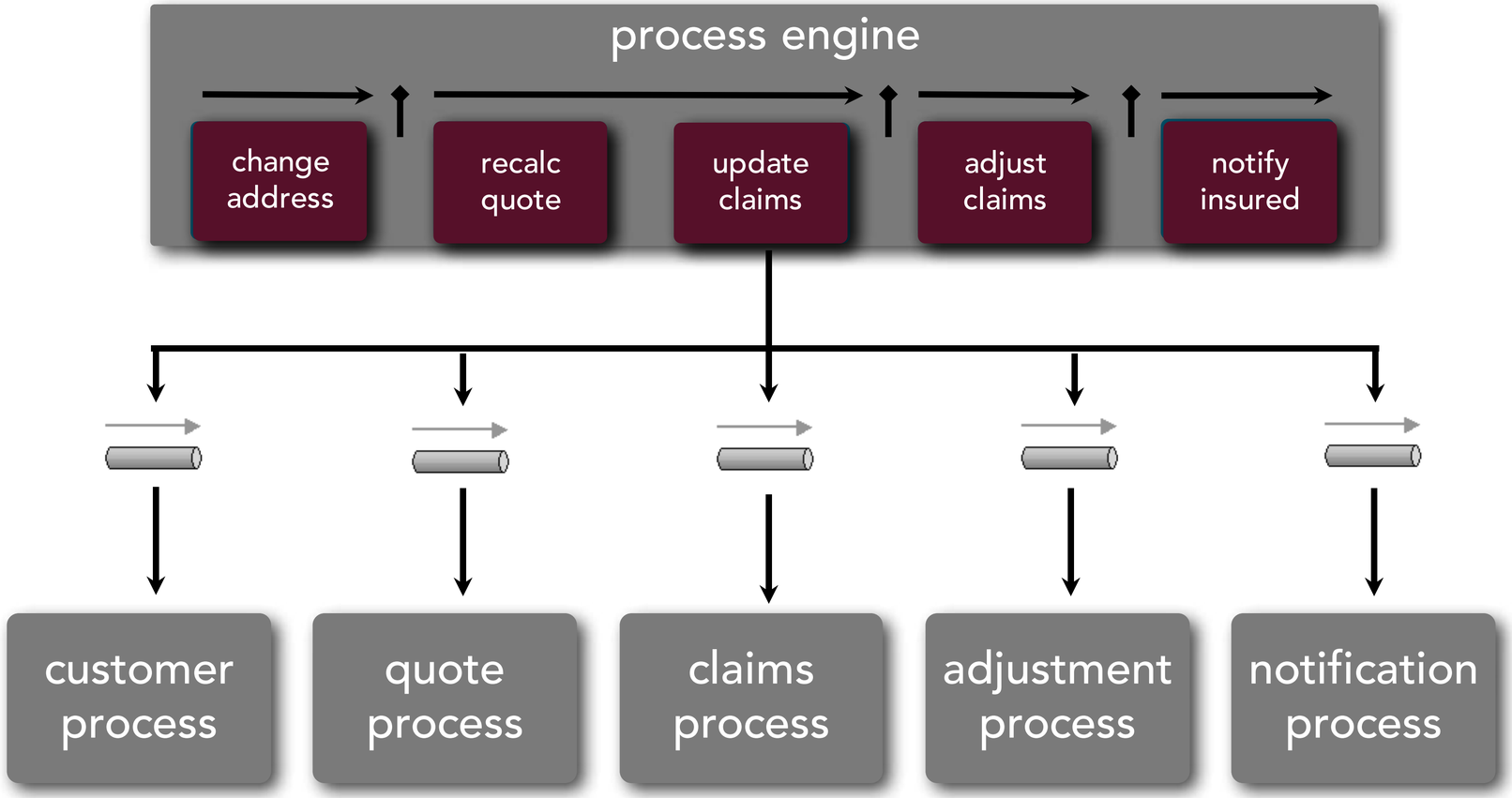


Event-driven Architecture

you move...

you moved

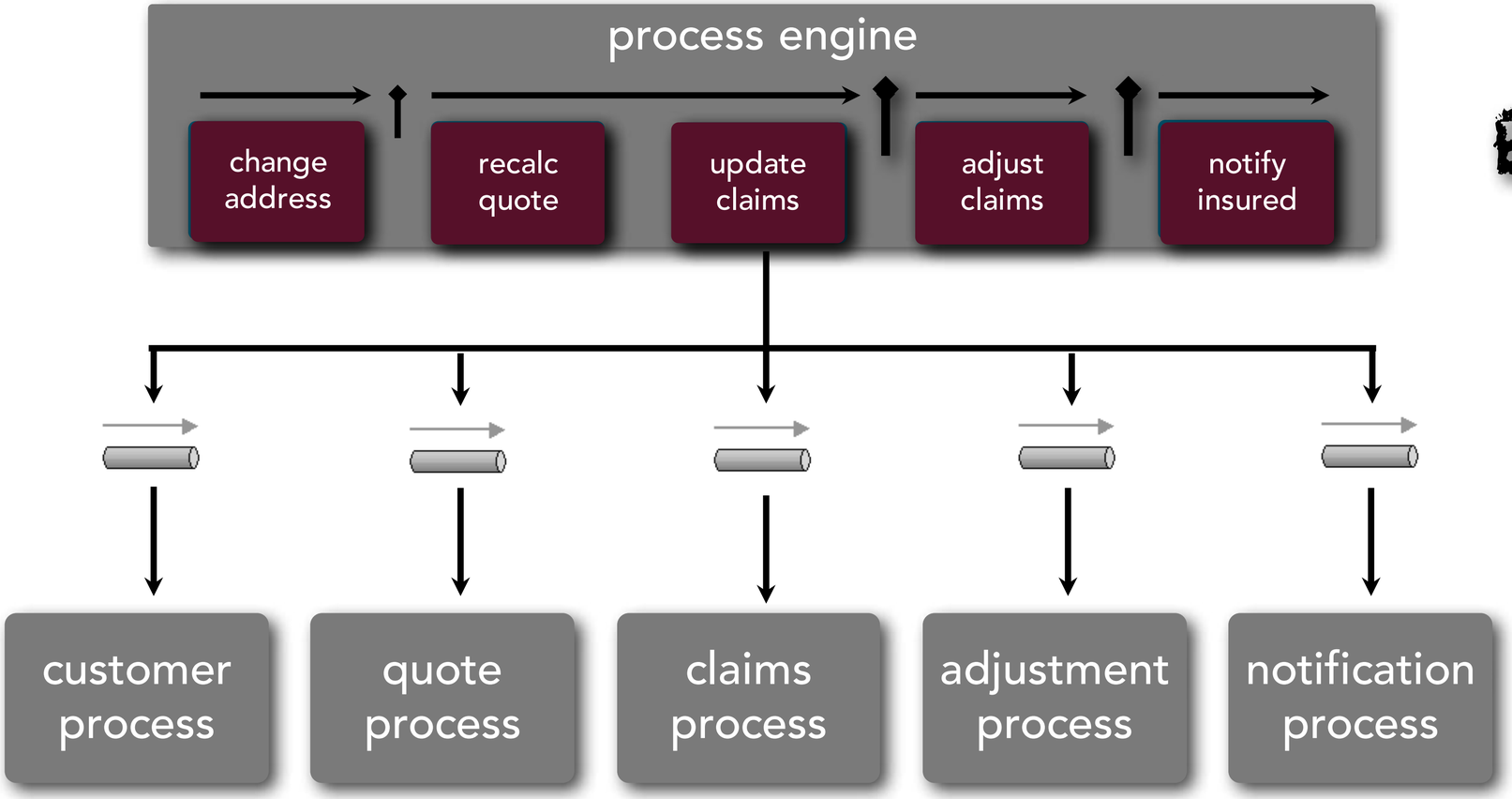
orchestration



Event-driven Architecture

you move...

you moved

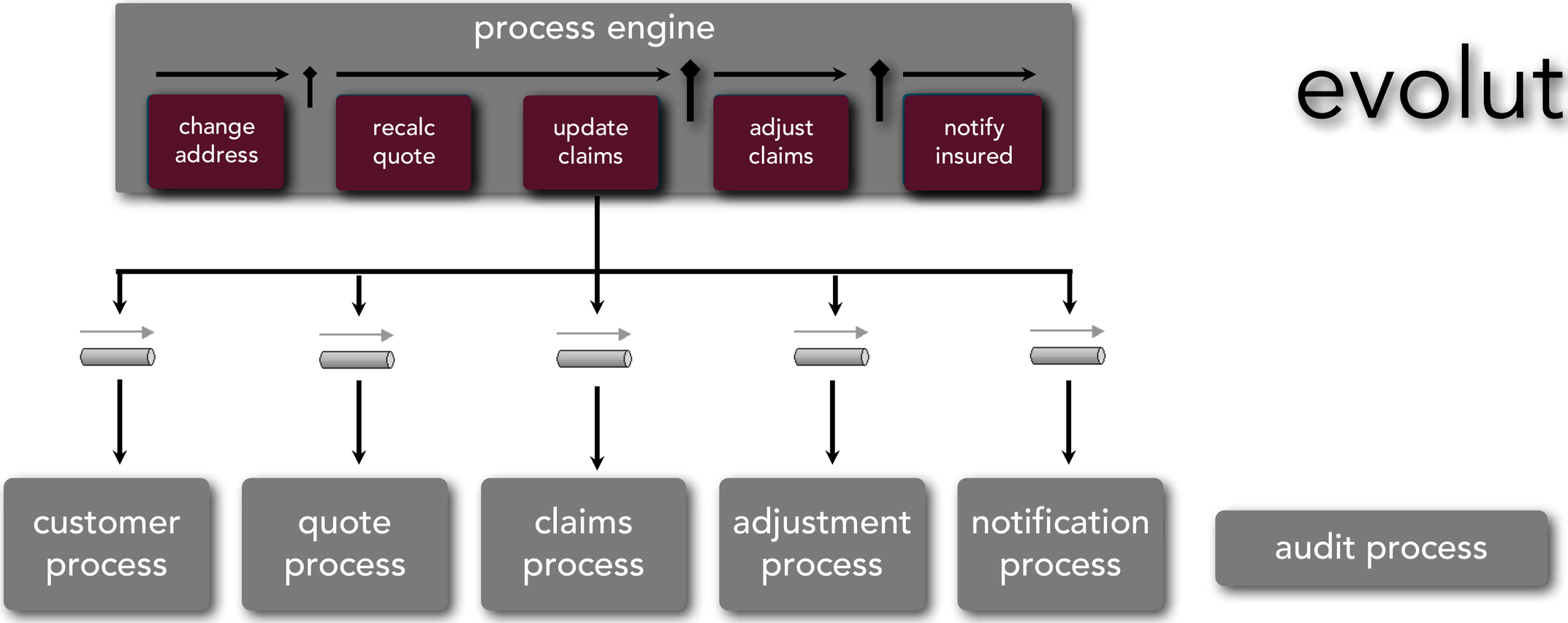


Error handling?

Event-driven Architecture

you move...

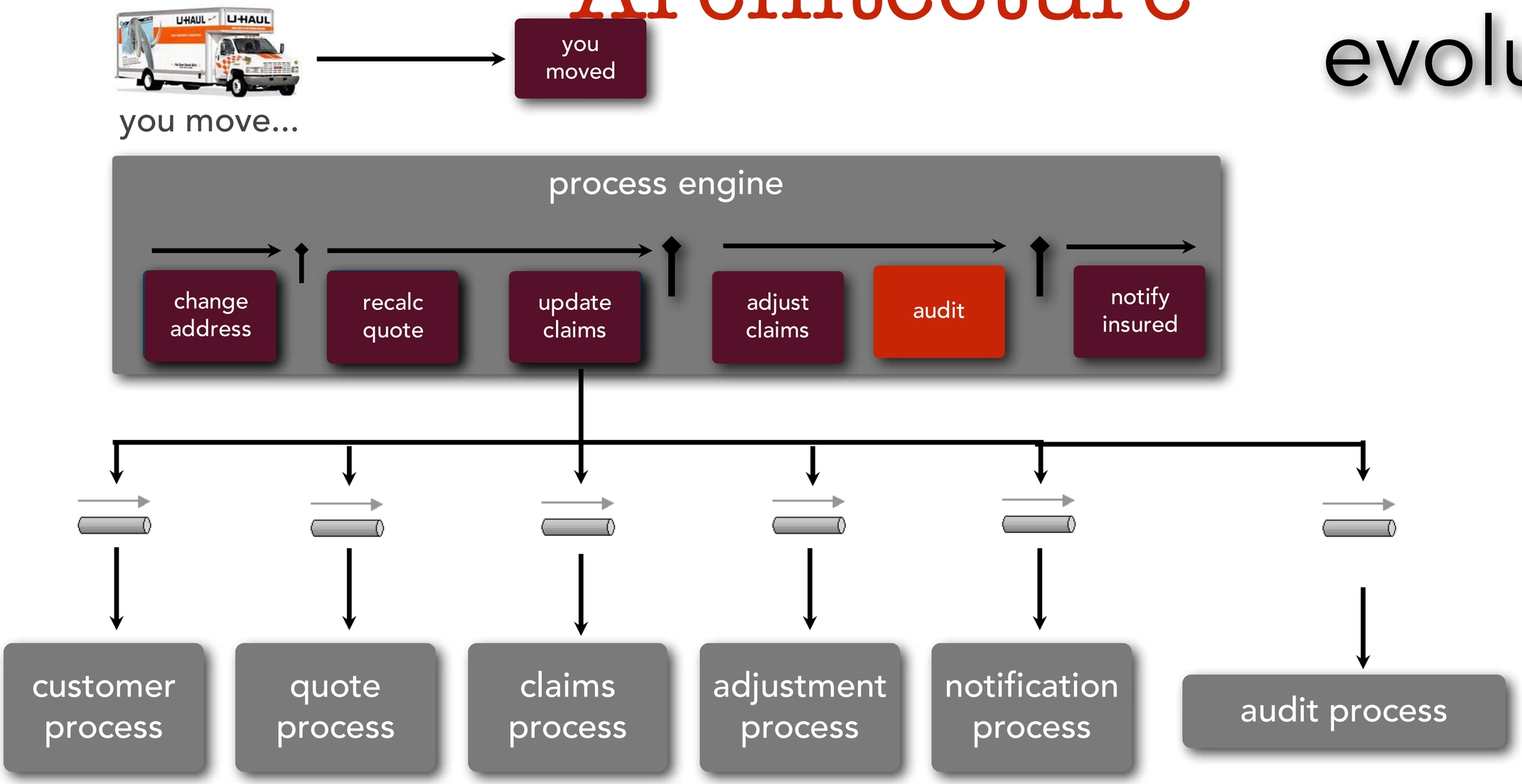
you moved



evolution

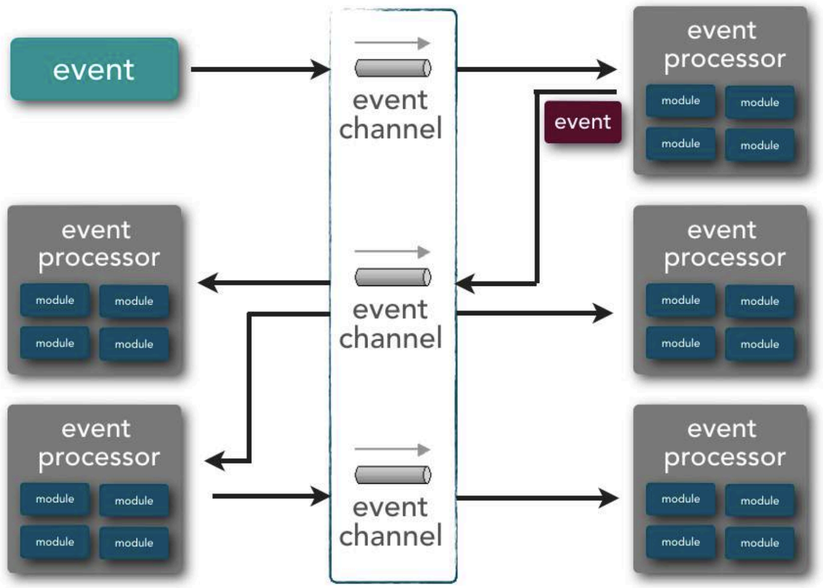
Event-driven Architecture

evolution



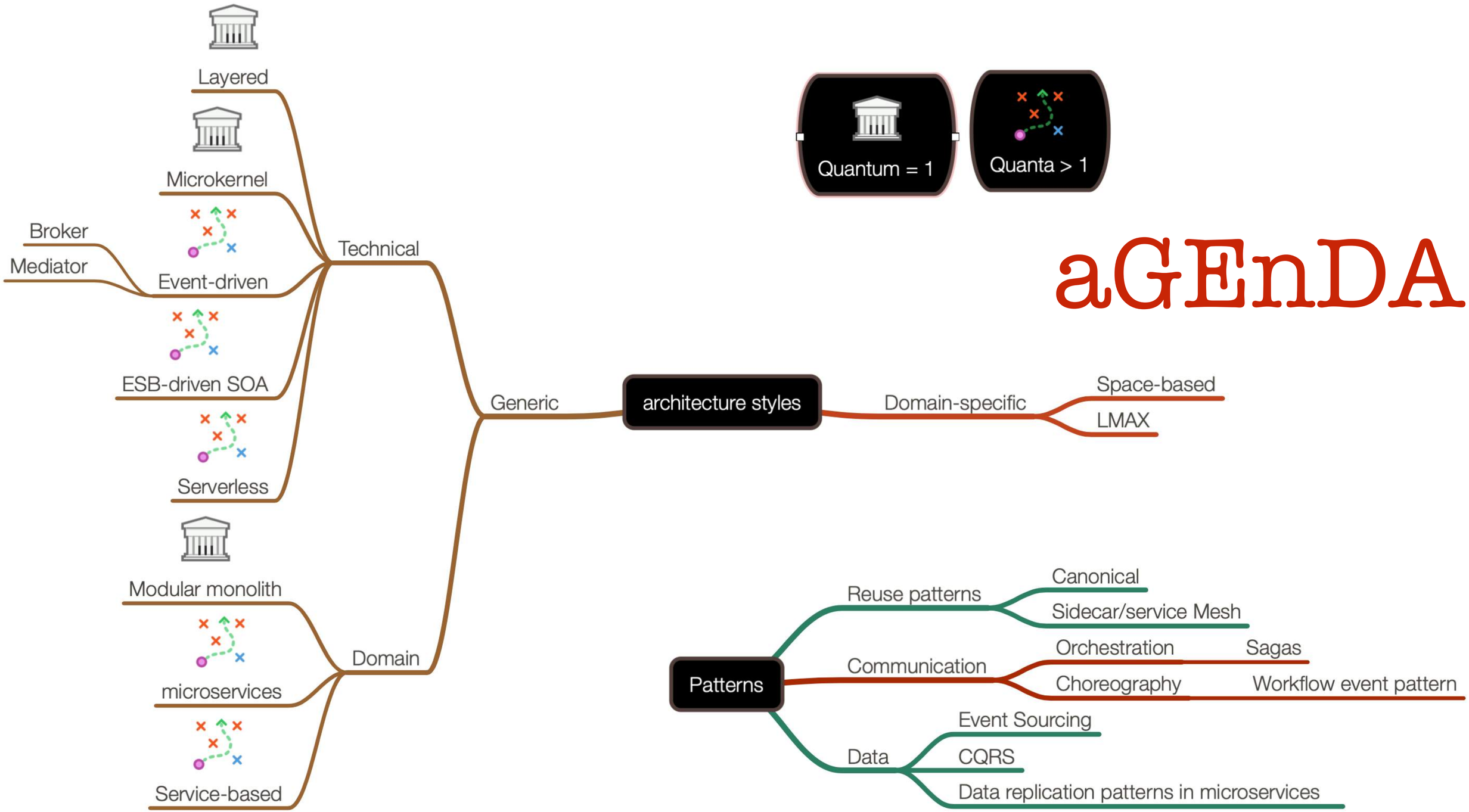
Event-driven Architecture

drivers

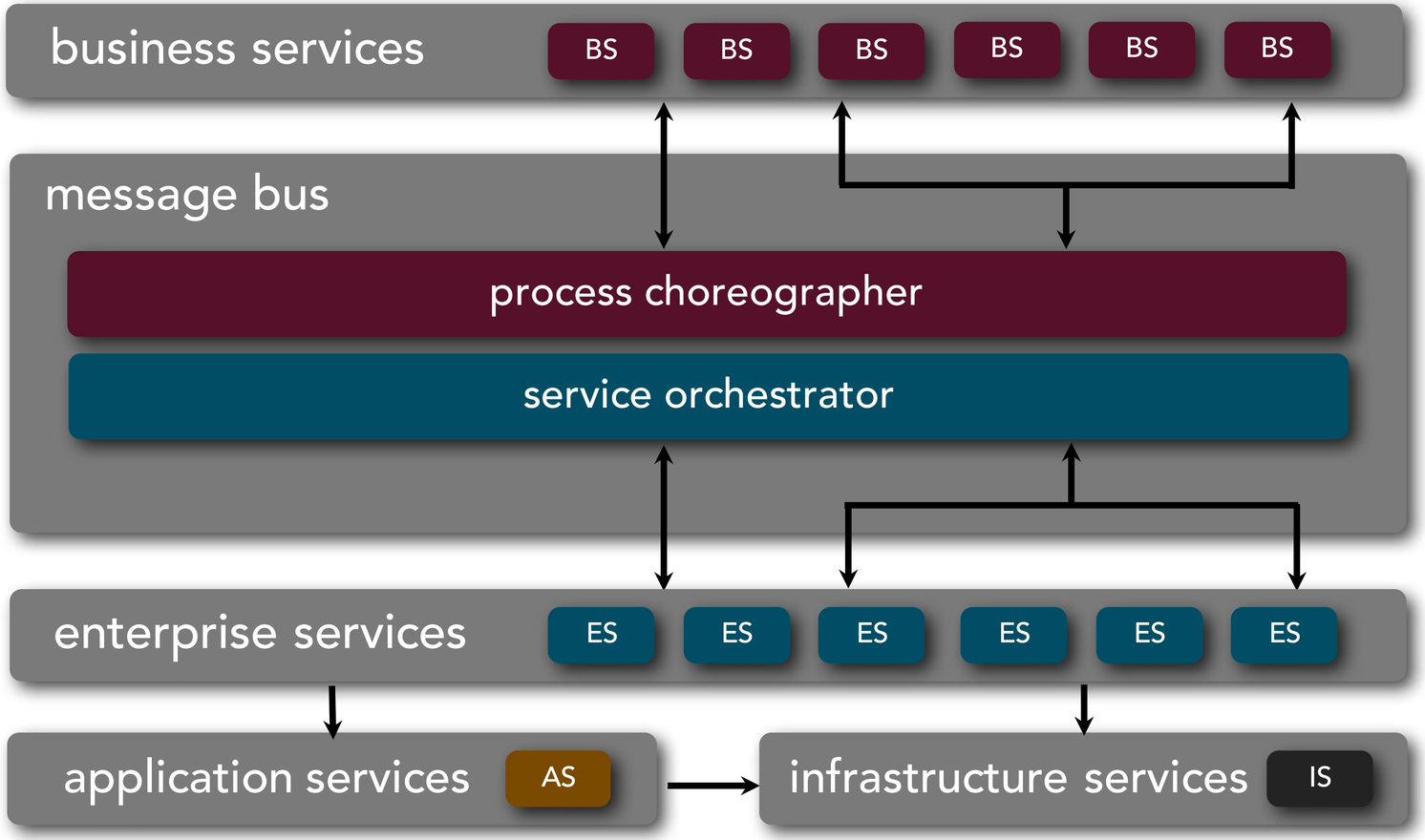


- ✓ modularity
- ✓ agility
- ✓ fault-tolerance
- ✓ scalability
- ✓ performance
- ✓ elasticity
- ✓ adaptability
- ✓ evolvability

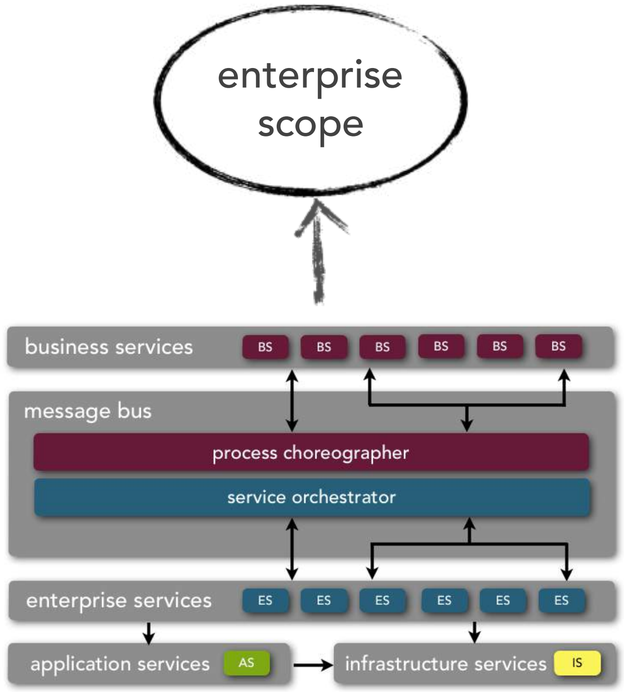
aGEnDA


Quantum = 1
Quanta > 1
architecture styles
Generic
Technical
Layered
Microkernel
Event-driven
Broker
Mediator
ESB-driven SOA
Serverless
Domain
Modular monolith
microservices
Service-based
Domain-specific
Space-based
LMAX
Patterns
Reuse patterns
Canonical
Sidecar/service Mesh
Communication
Orchestration
Sagas
Choreography
Workflow event pattern
Data
Event Sourcing
CQRS
Data replication patterns in microservices

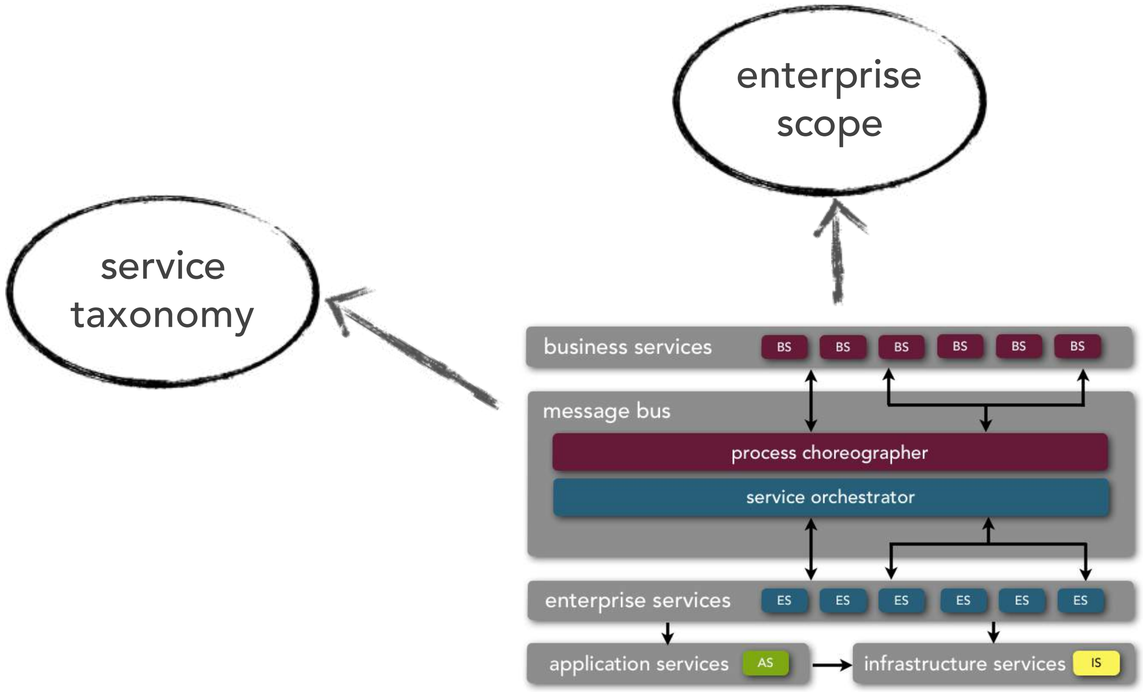
ESB-driven SOA



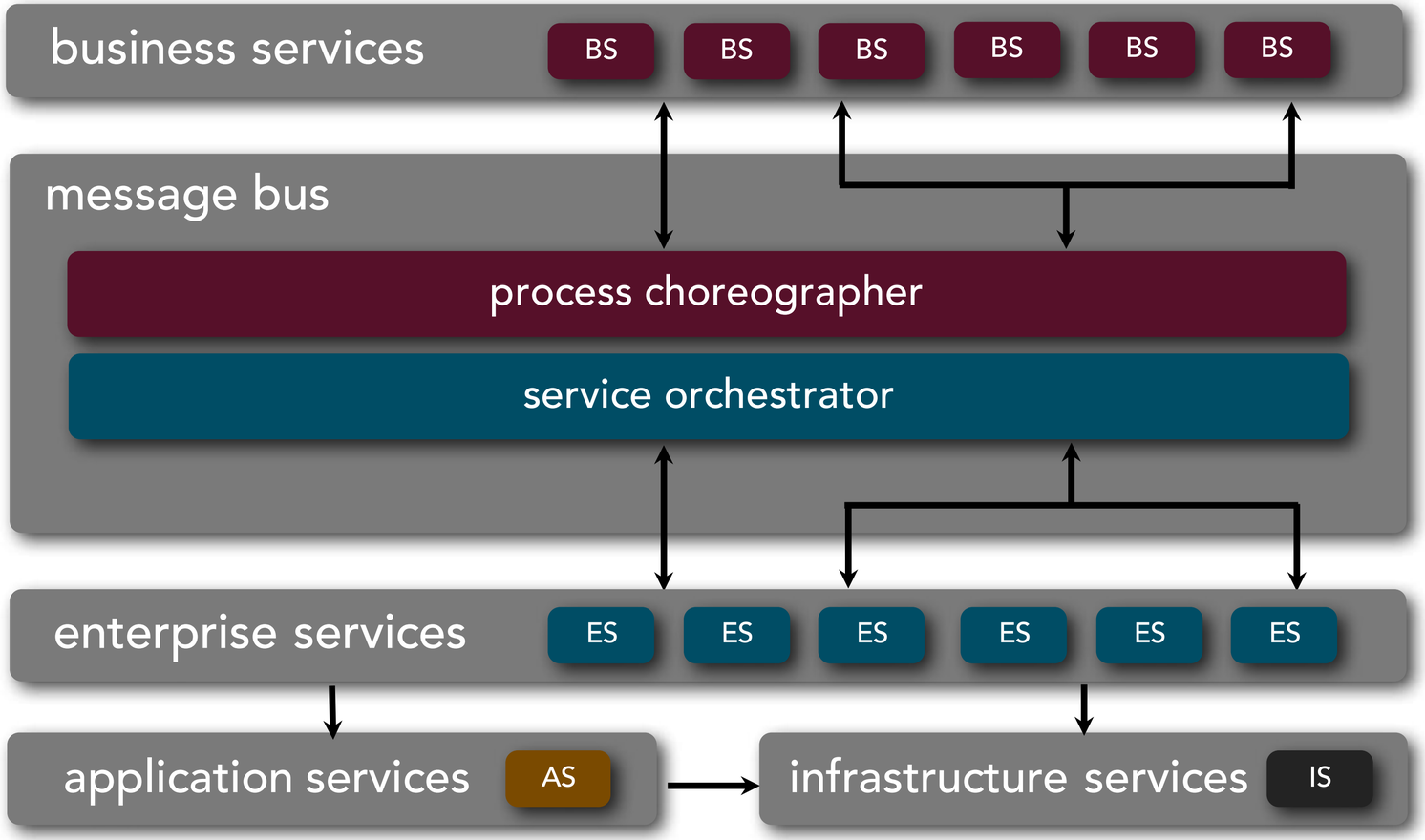
ESB-driven SOA



ESB-driven SOA

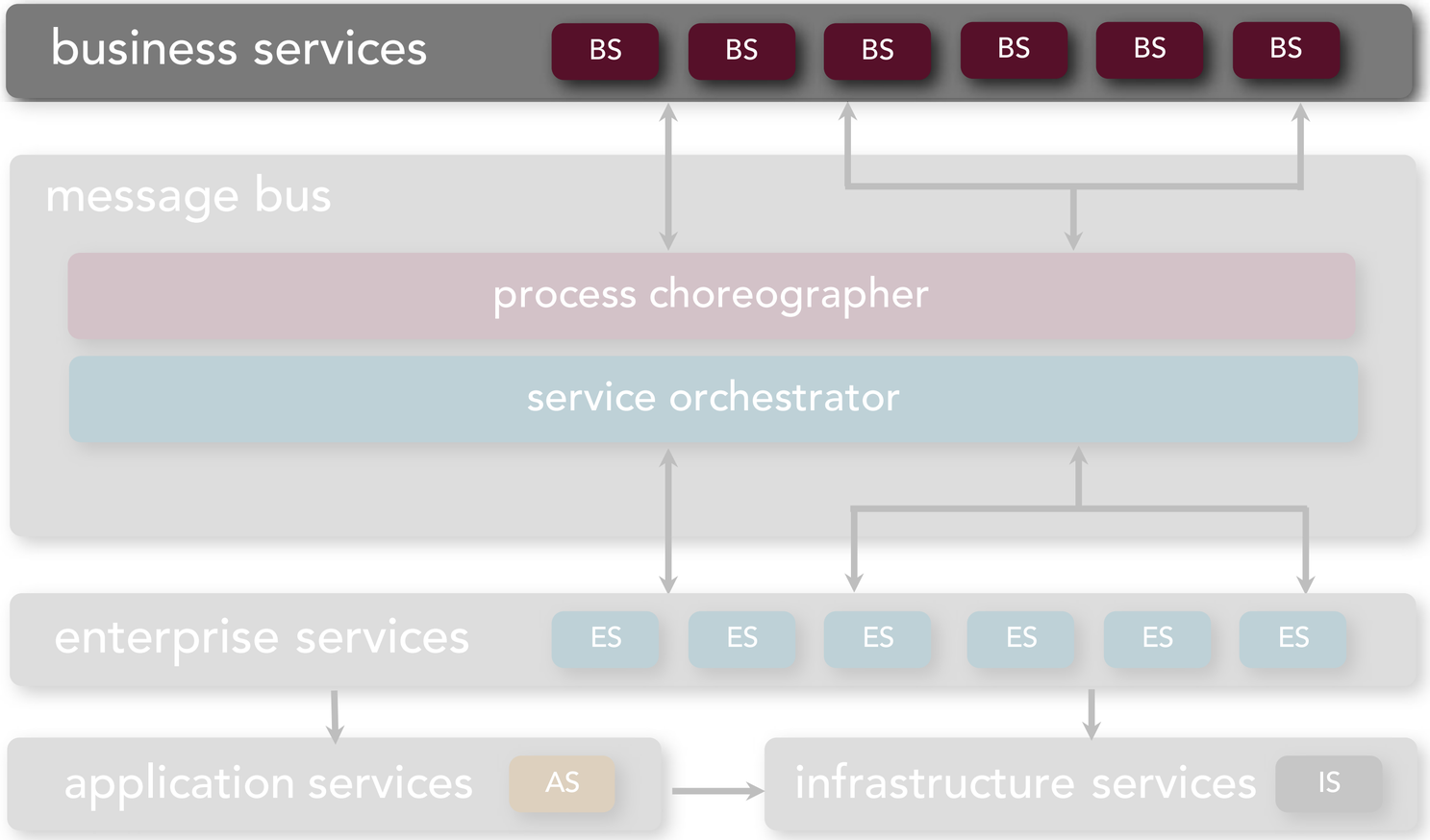


ESB-driven SOA



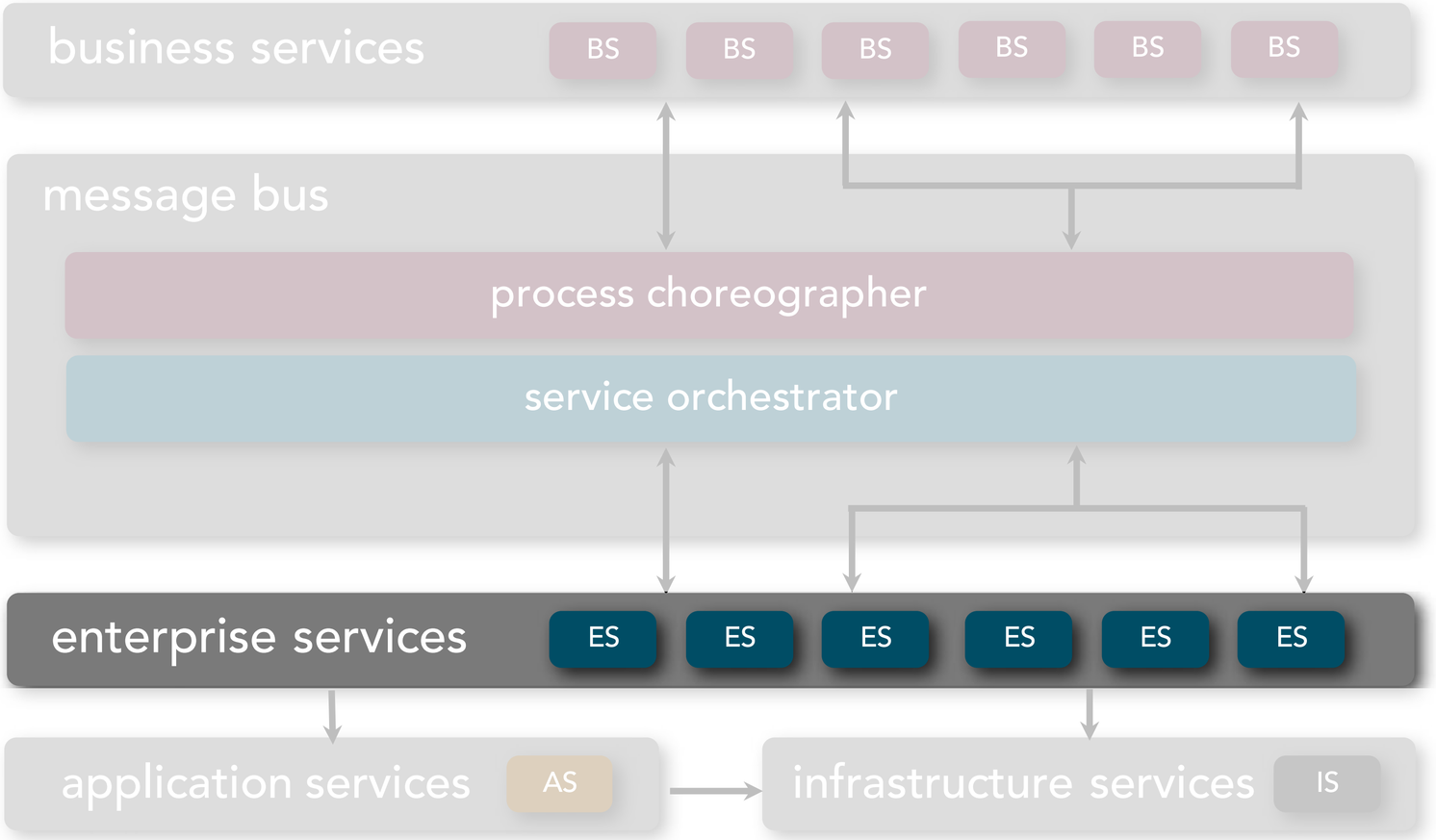
ESB-driven SOA

business services



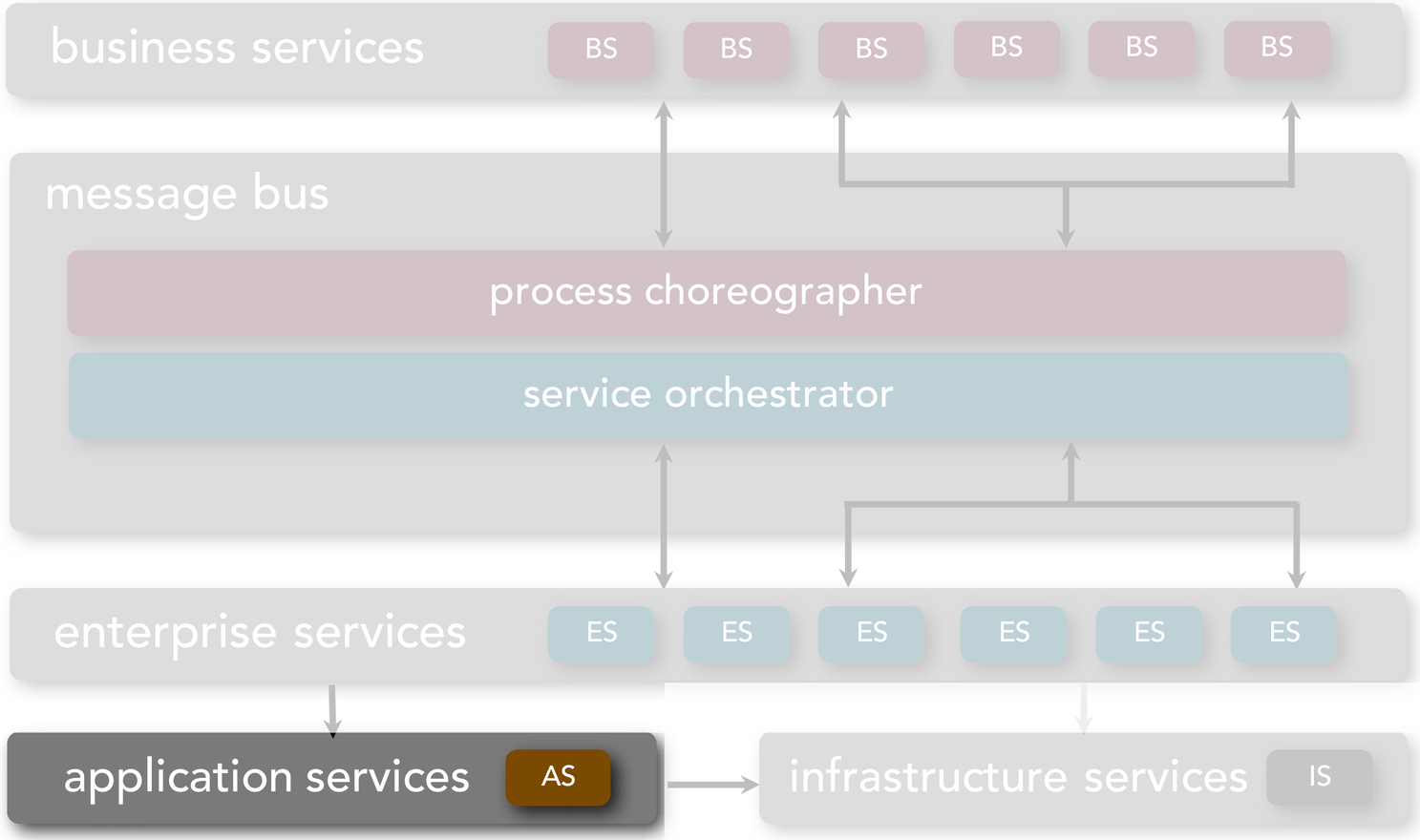
ESB-driven SOA

enterprise services



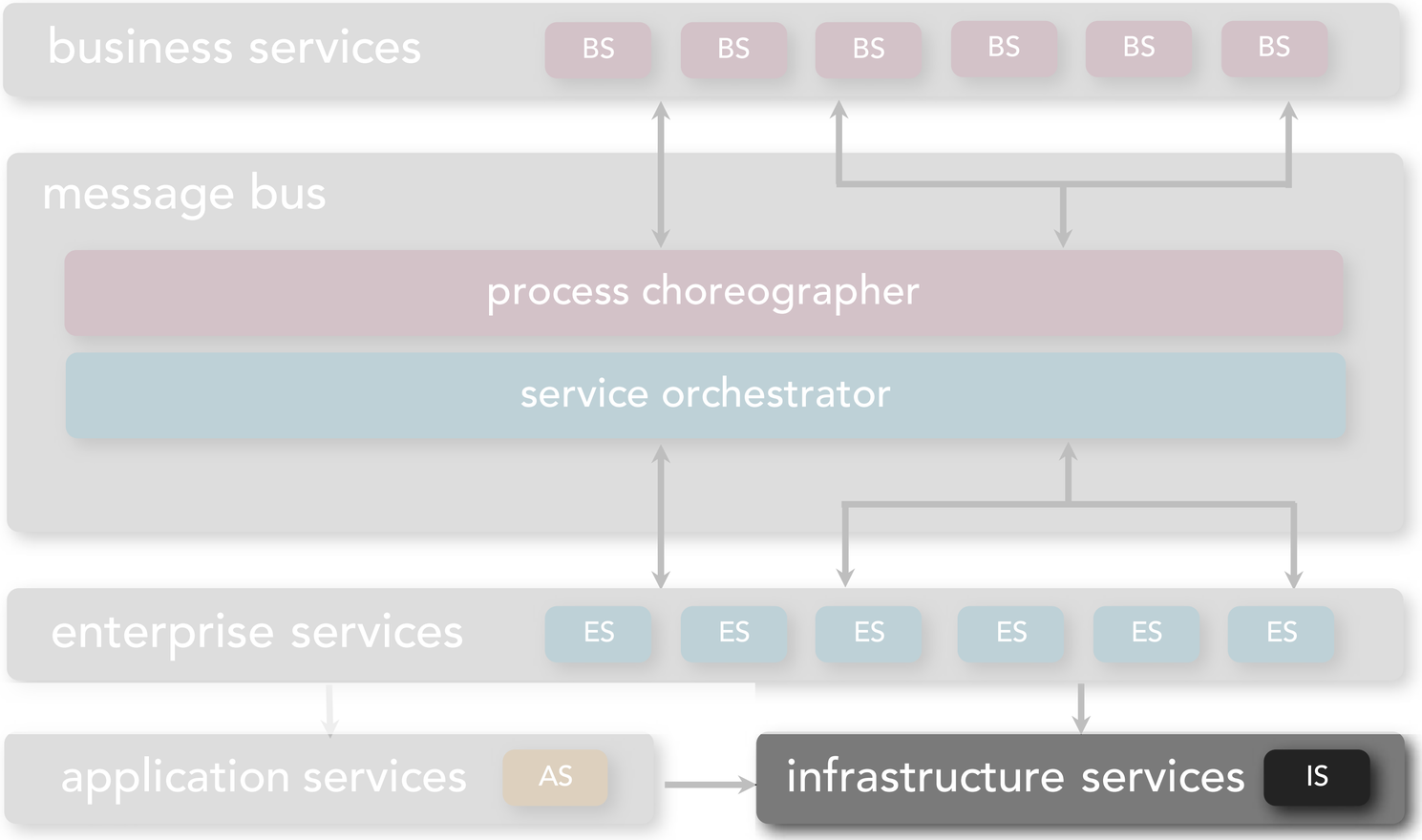
ESB-driven SOA

application services

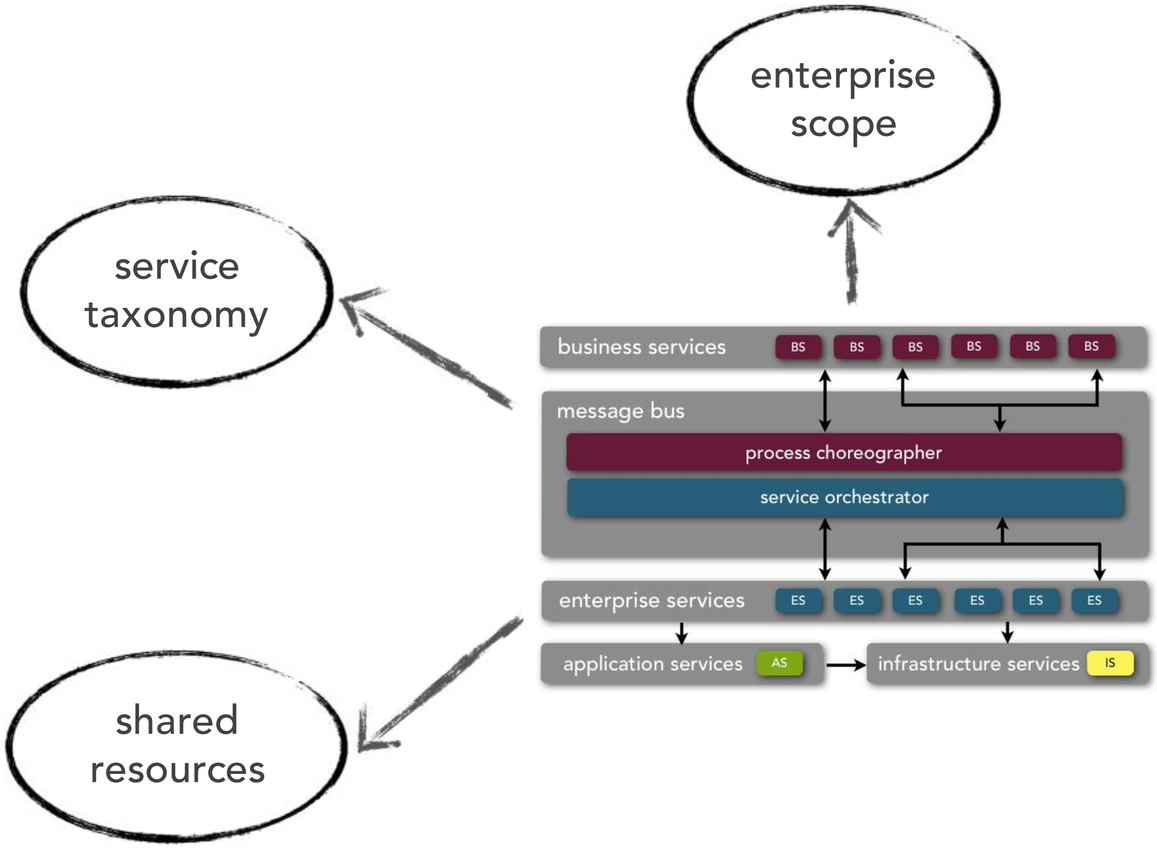


ESB-driven SOA

infrastructure services

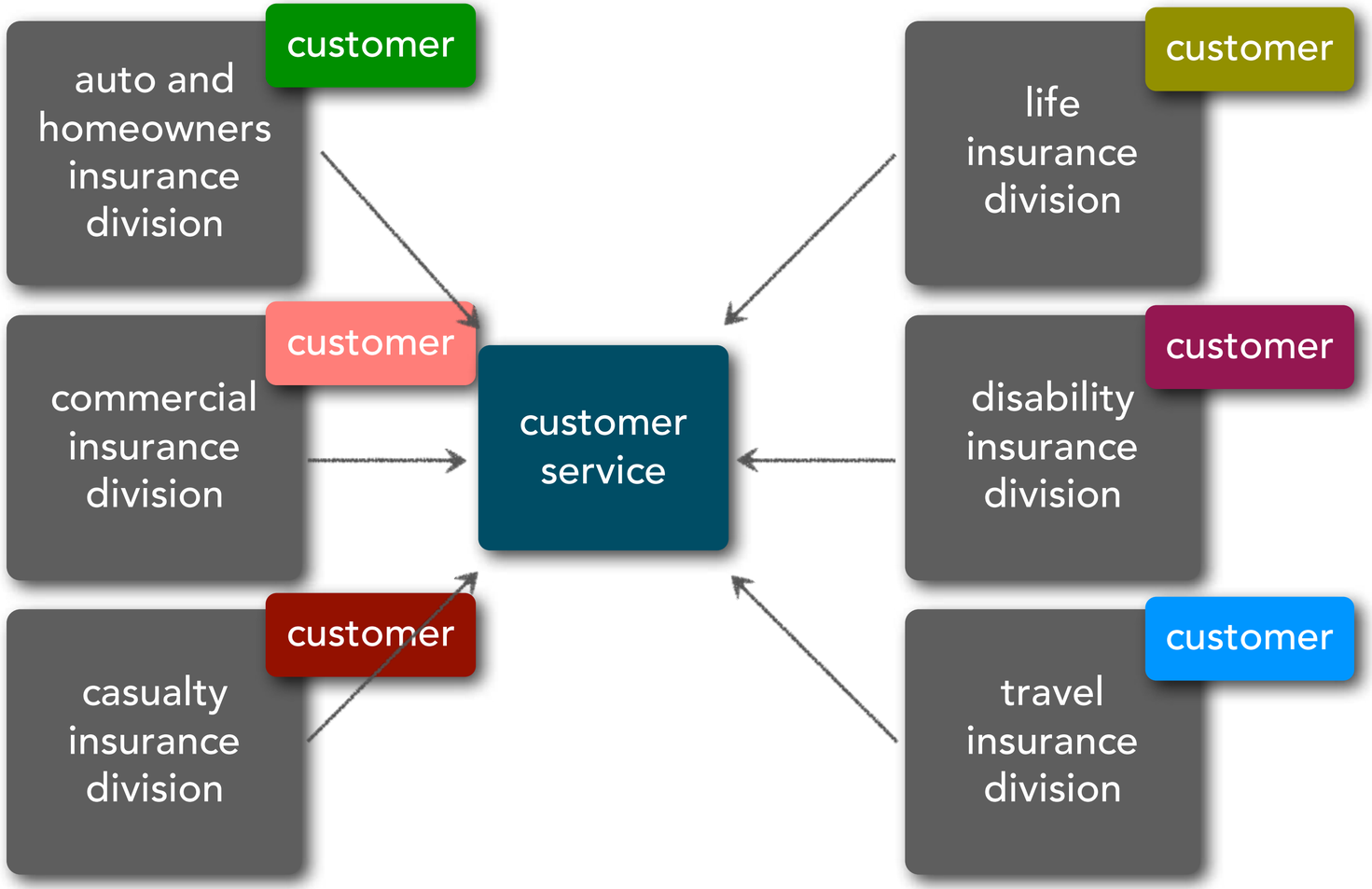


ESB-driven SOA

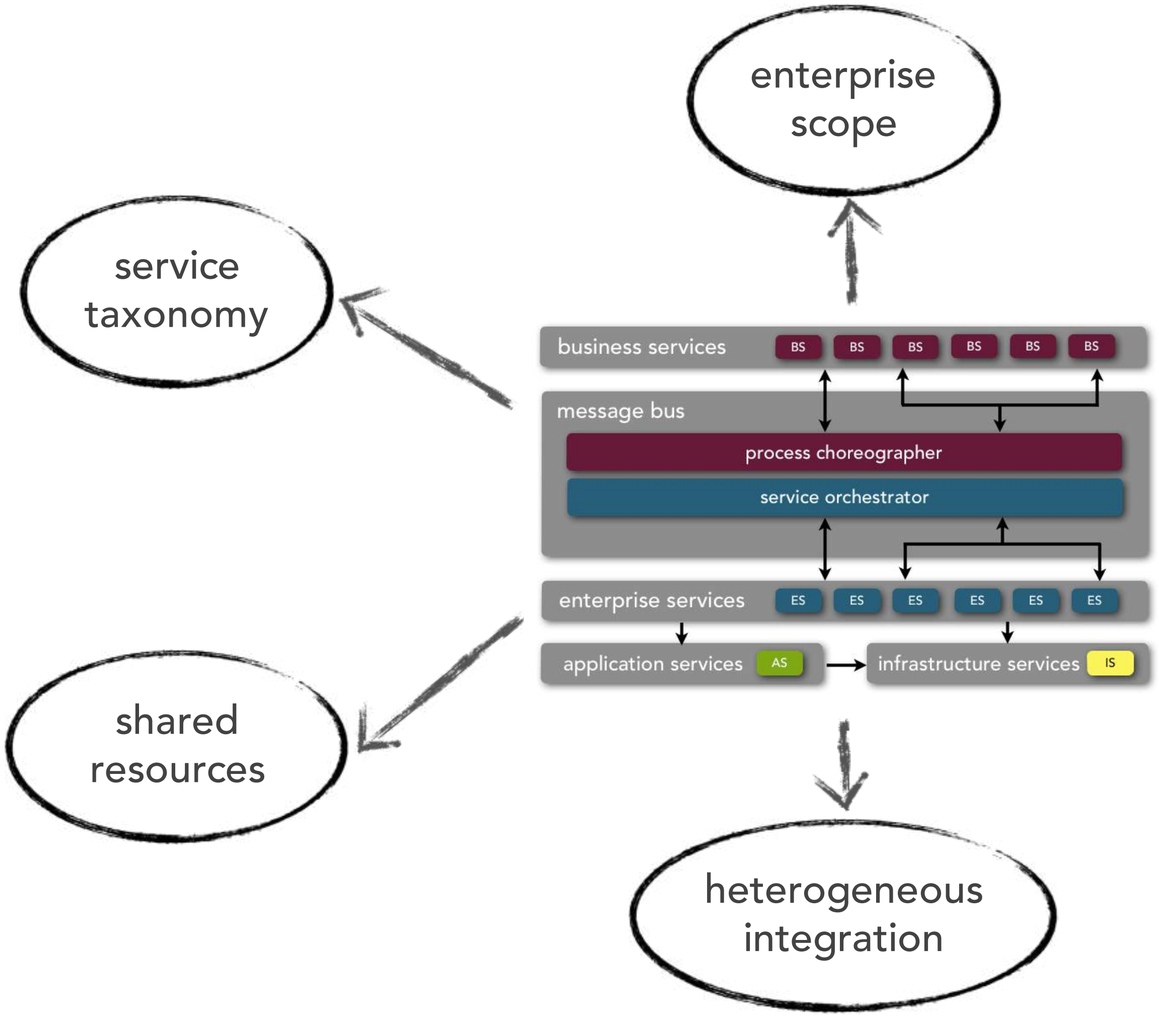


ESB-driven SOA

shared resources

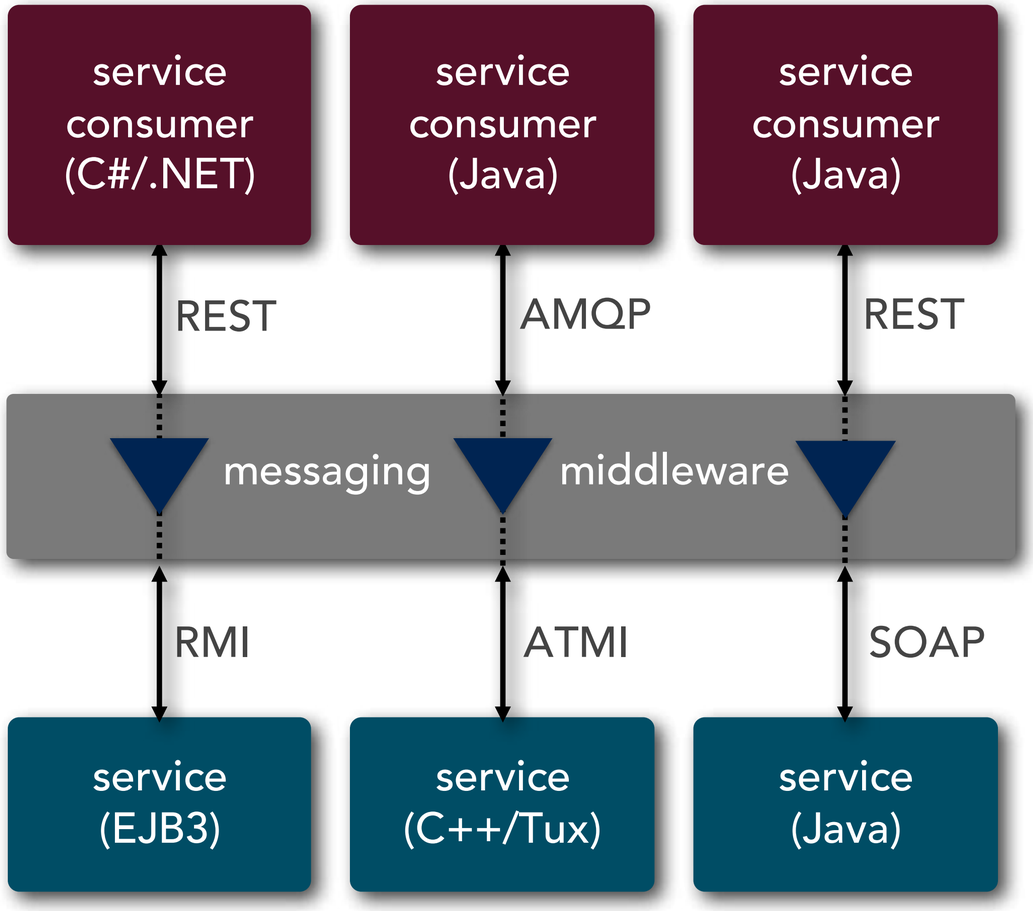


ESB-driven SOA

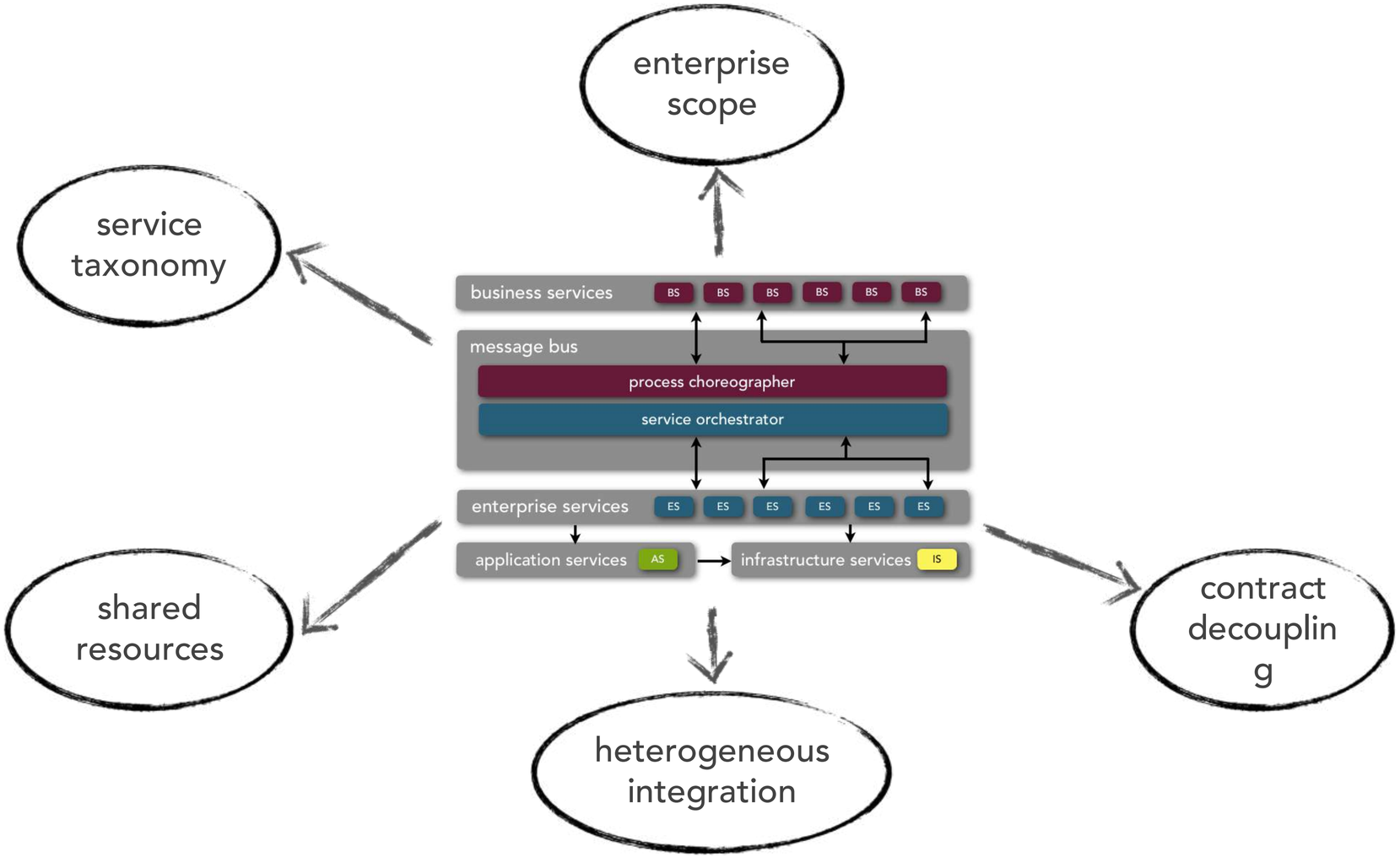


ESB-driven SOA

protocol-agnostic heterogeneous interoperability

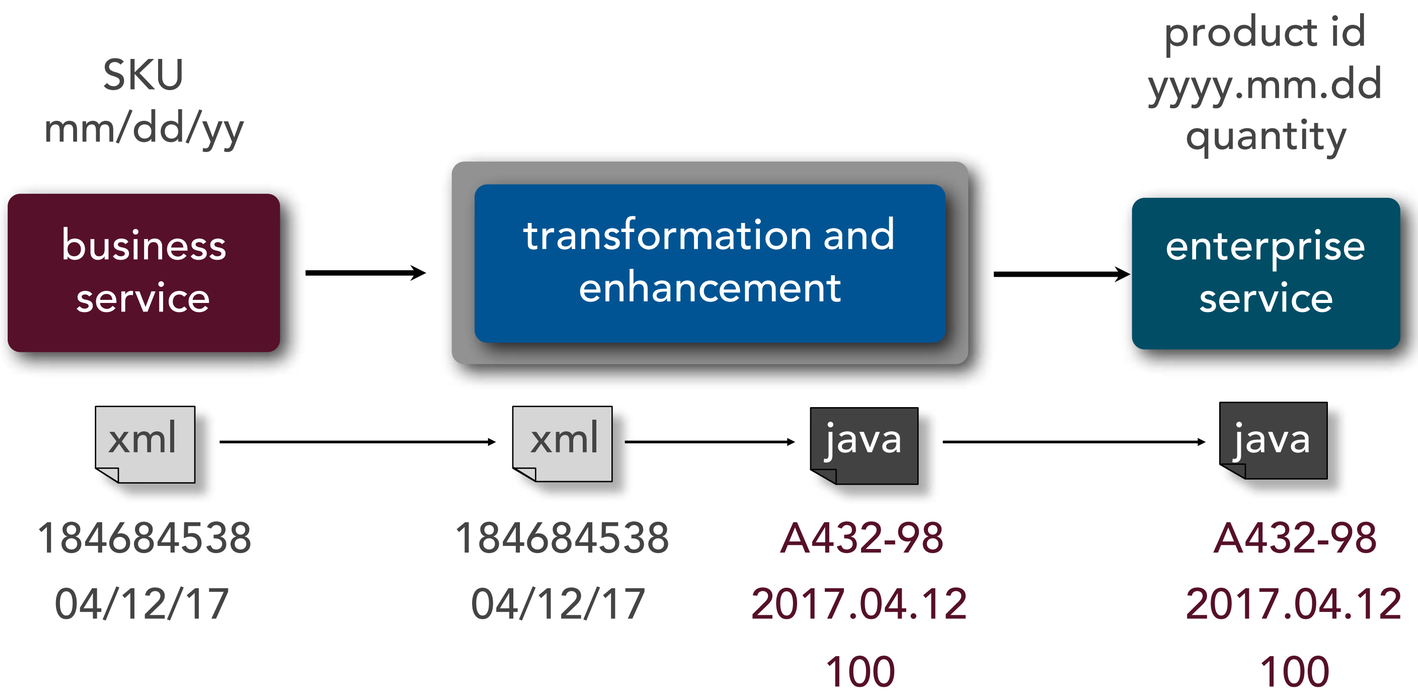


ESB-driven SOA

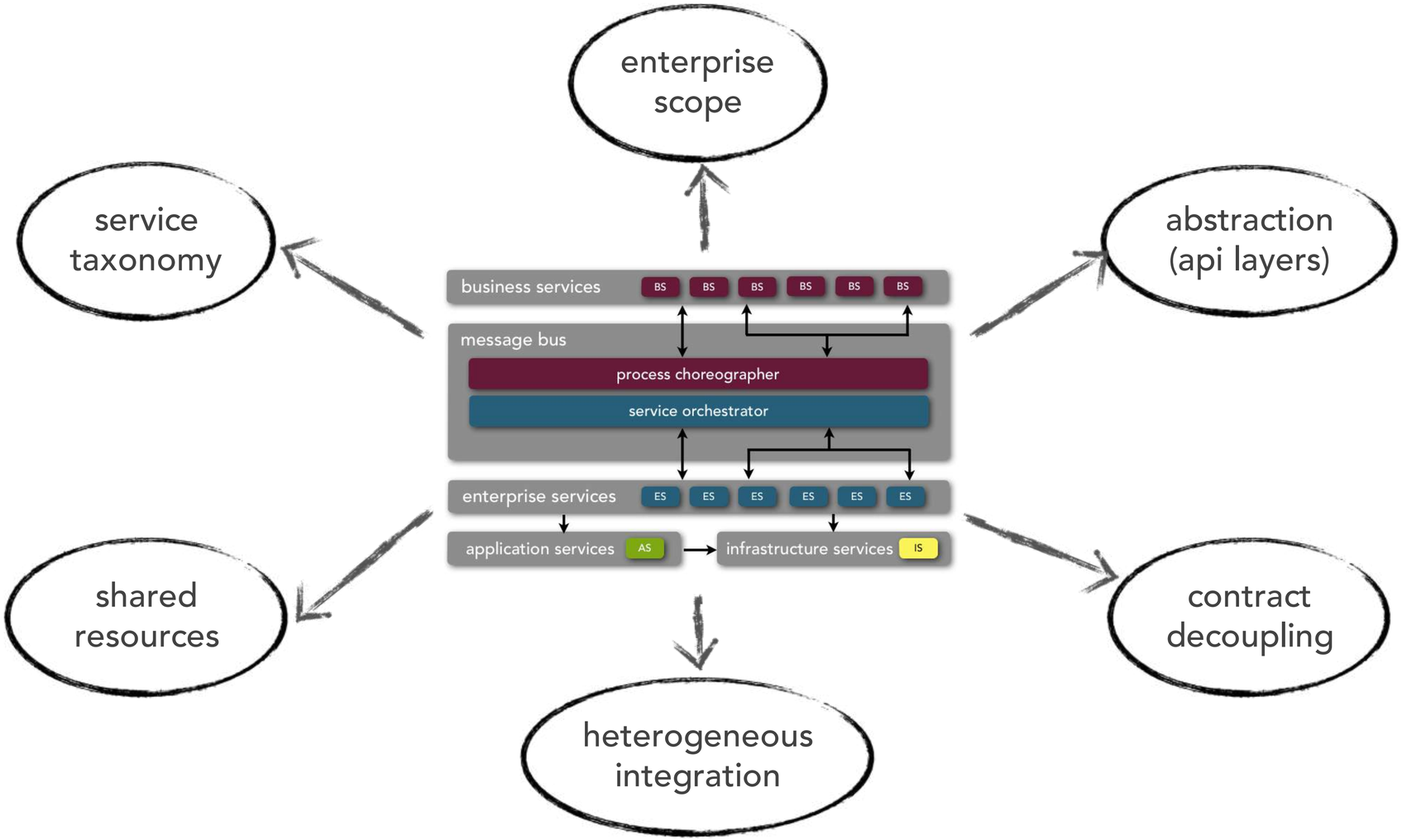


ESB-driven SOA

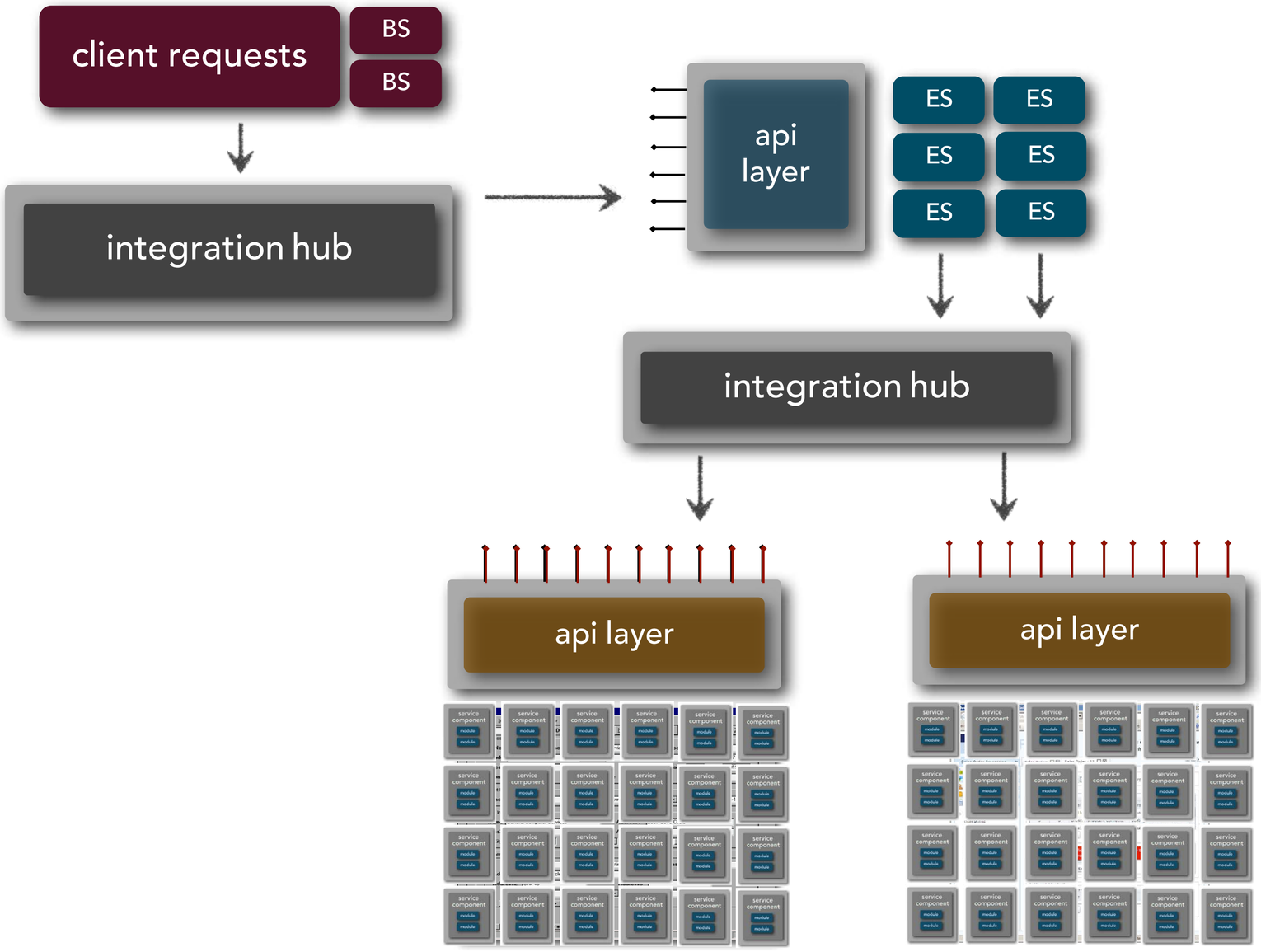
contract decoupling



service-oriented architecture

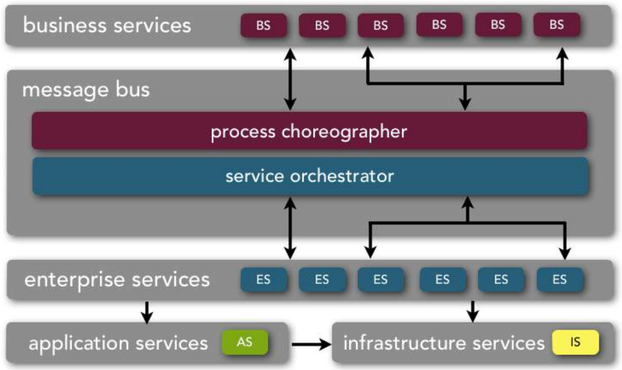


ESB-driven SOA



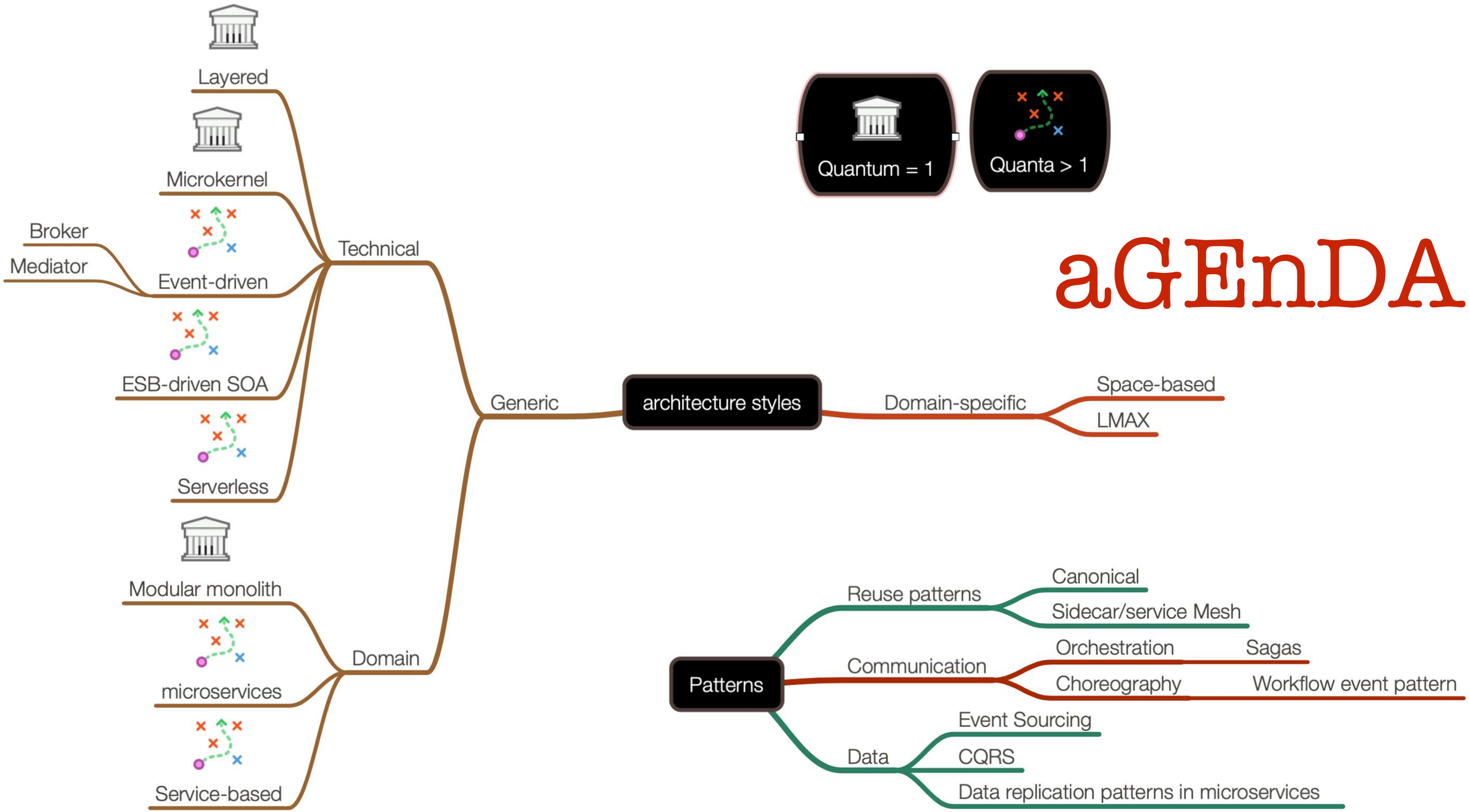
ESB-driven SOA

drivers

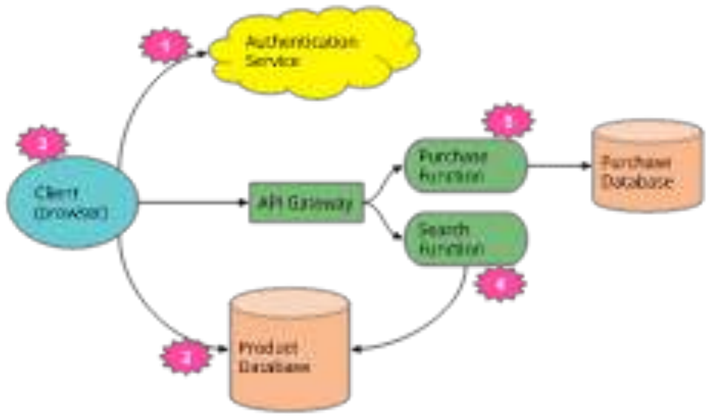


- ✓ integration
- ✓ integration
- ✓ orchestration

aGEnDA


Quantum = 1
Quanta > 1
architecture styles
Generic
Technical
Layered
Microkernel
Event-driven
Broker
Mediator
ESB-driven SOA
Serverless
Domain
Modular monolith
microservices
Service-based
Domain-specific
Space-based
LMAX
Patterns
Reuse patterns
Canonical
Sidecar/service Mesh
Communication
Orchestration
Sagas
Choreography
Workflow event pattern
Data
Event Sourcing
CQRS
Data replication patterns in microservices

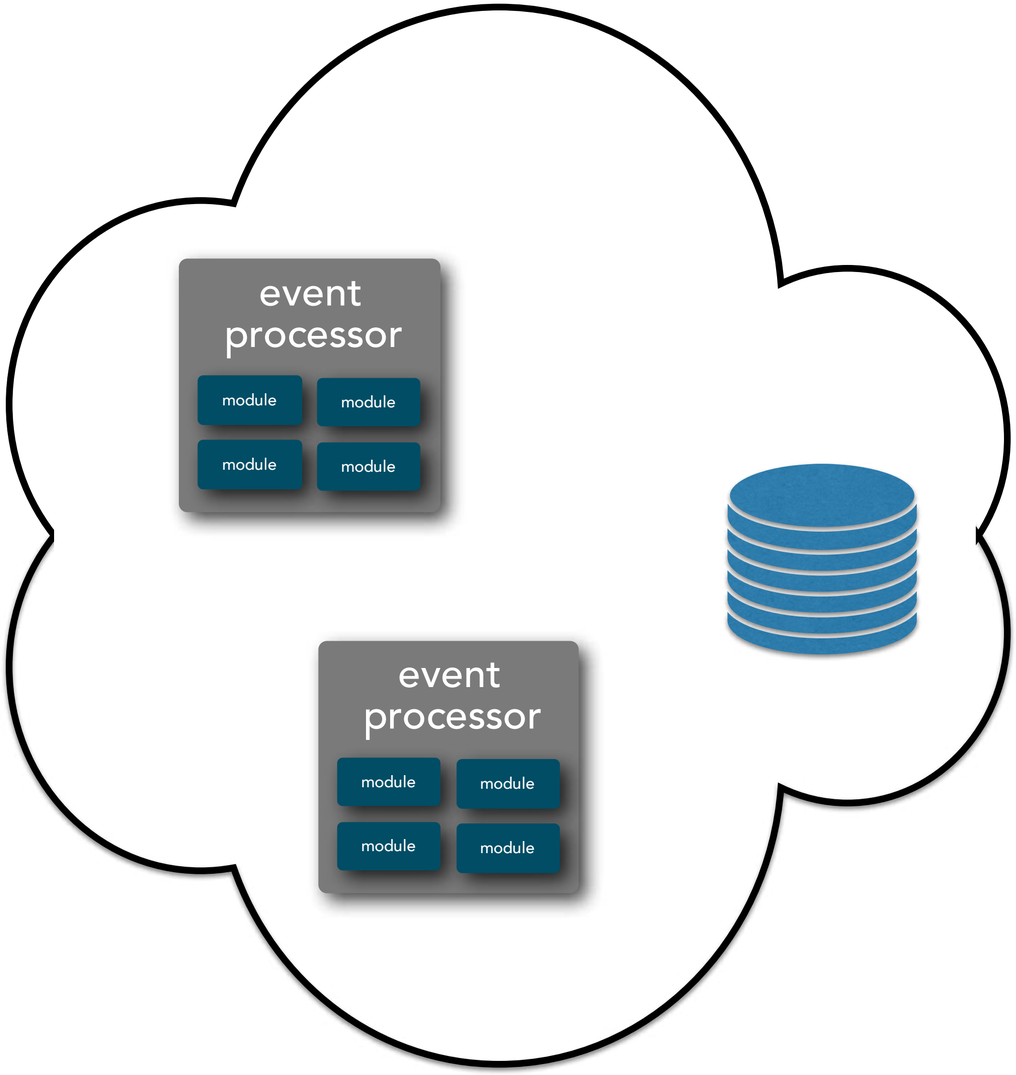
Serverless Architectures

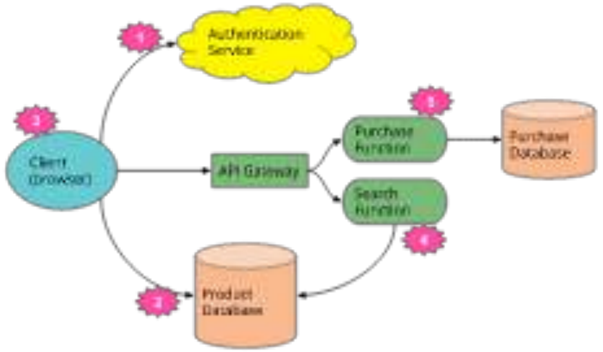


BaaS

FaaS

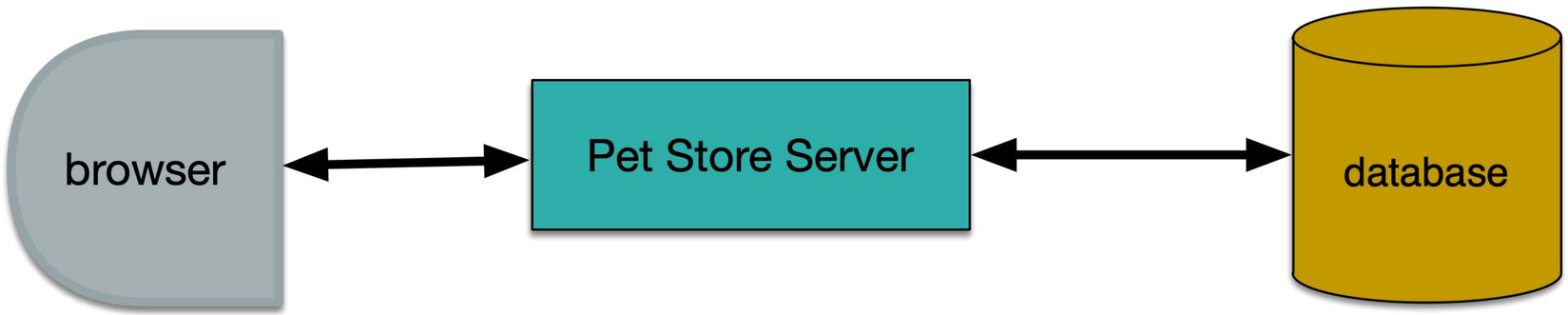
Serverless


event processor
module
module
module
module
event processor
module
module
module
module

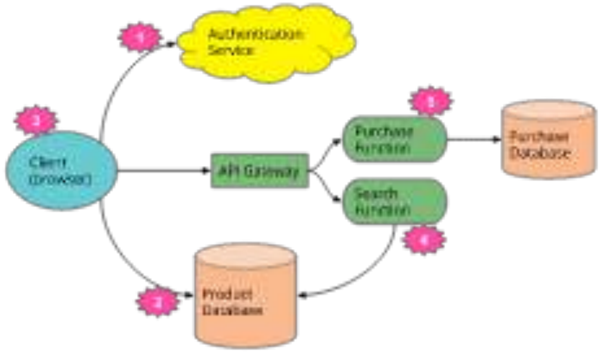


BaaS

(Backend as a Service)

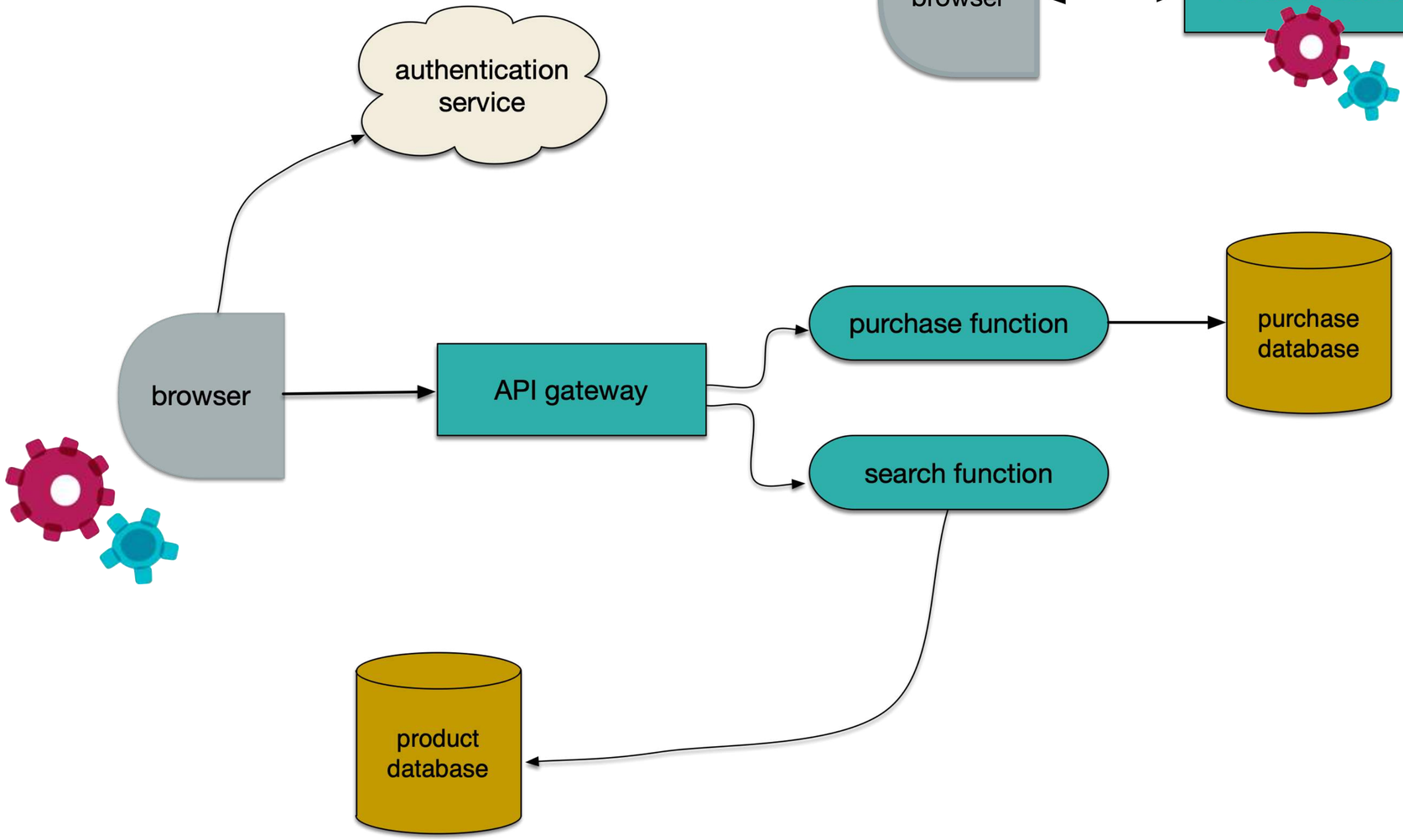


traditional client/server architecture

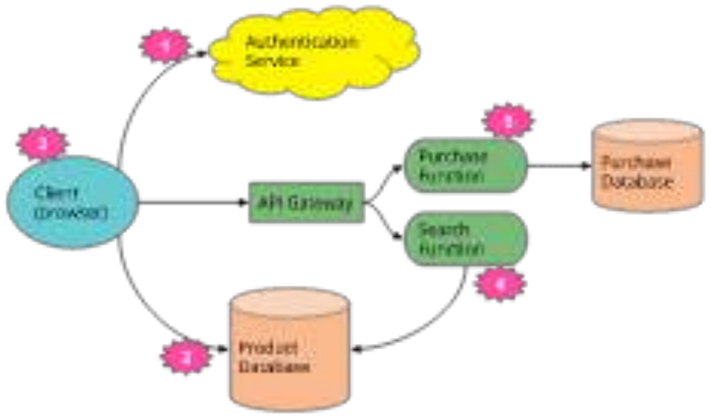

Authentication Service
Client (browser)
API Gateway
Purchase Function
Search Function
Purchase Database
Product Database

BaaS

browser
Pet Store Server
database


authentication service
browser
API gateway
purchase function
purchase database
search function
product database

Serverless Architectures

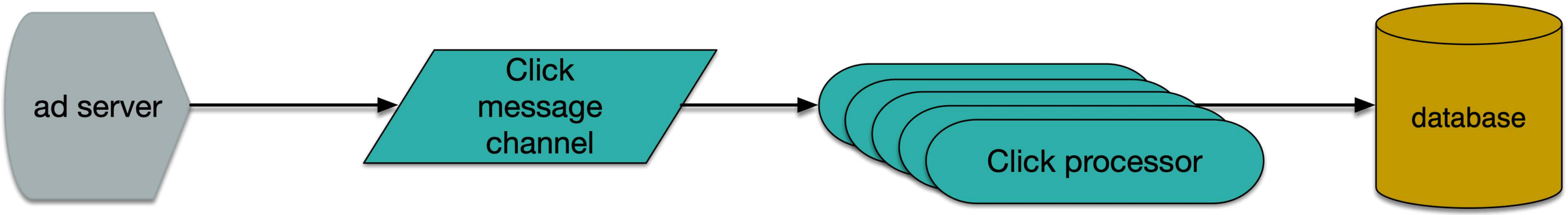


BaaS

FaaS

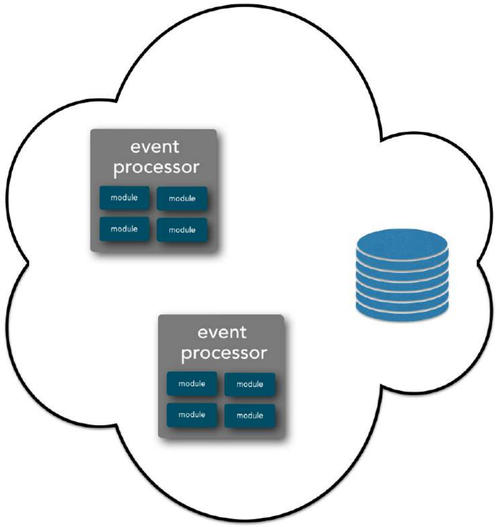
FaaS

(Functions as a Service)



Serverless

drivers

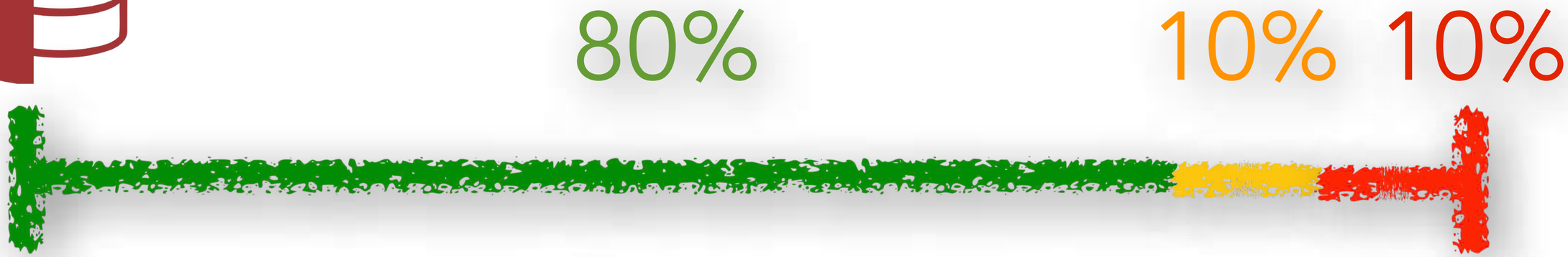


- ✓ agility
- ✓ lack of (apparent) infrastructure
- ✓ discrete functionality

Last 10% Trap

Last 10% Trap

"Users always want 100% of what they want (& are never satisfied with less)."

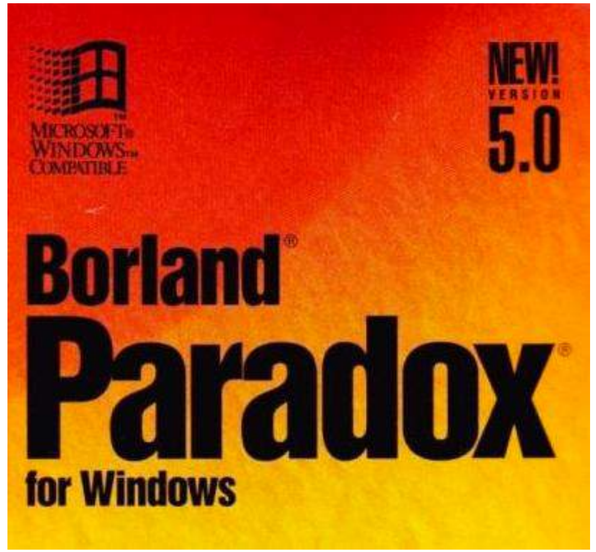


what the user wants

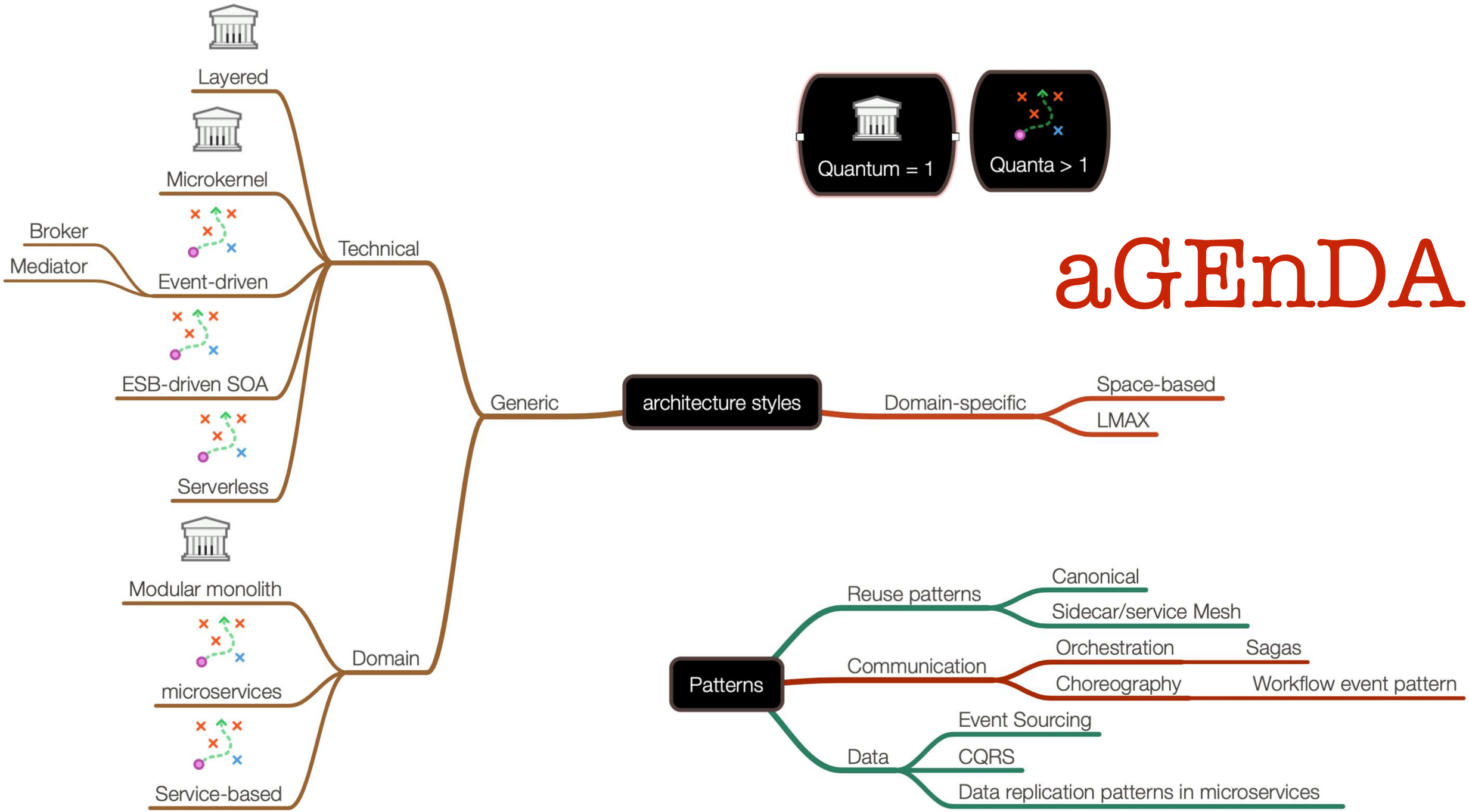
What happened to the 4GLs?

dBASE™

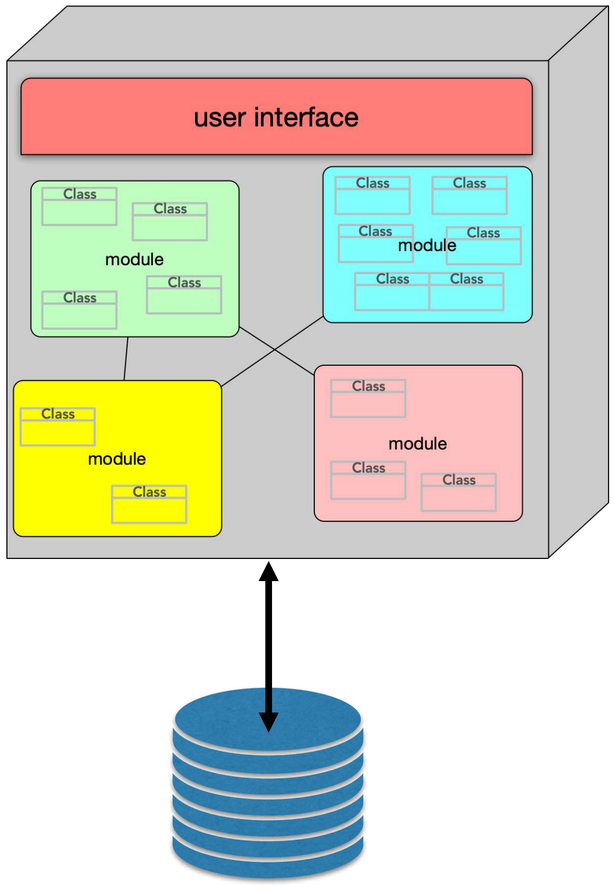
Microsoft® Visual FoxPro®



aGEnDA

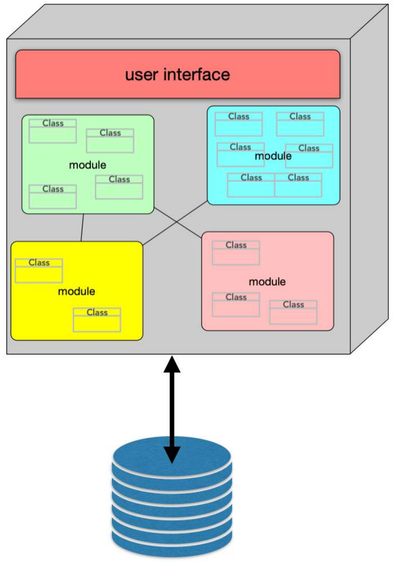

Quantum = 1
Quanta > 1
architecture styles
Generic
Technical
Layered
Microkernel
Event-driven
Broker
Mediator
ESB-driven SOA
Serverless
Domain
Modular monolith
microservices
Service-based
Domain-specific
Space-based
LMAX
Patterns
Reuse patterns
Canonical
Sidecar/service Mesh
Communication
Orchestration
Sagas
Choreography
Workflow event pattern
Data
Event Sourcing
CQRS
Data replication patterns in microservices

Modular Monolith



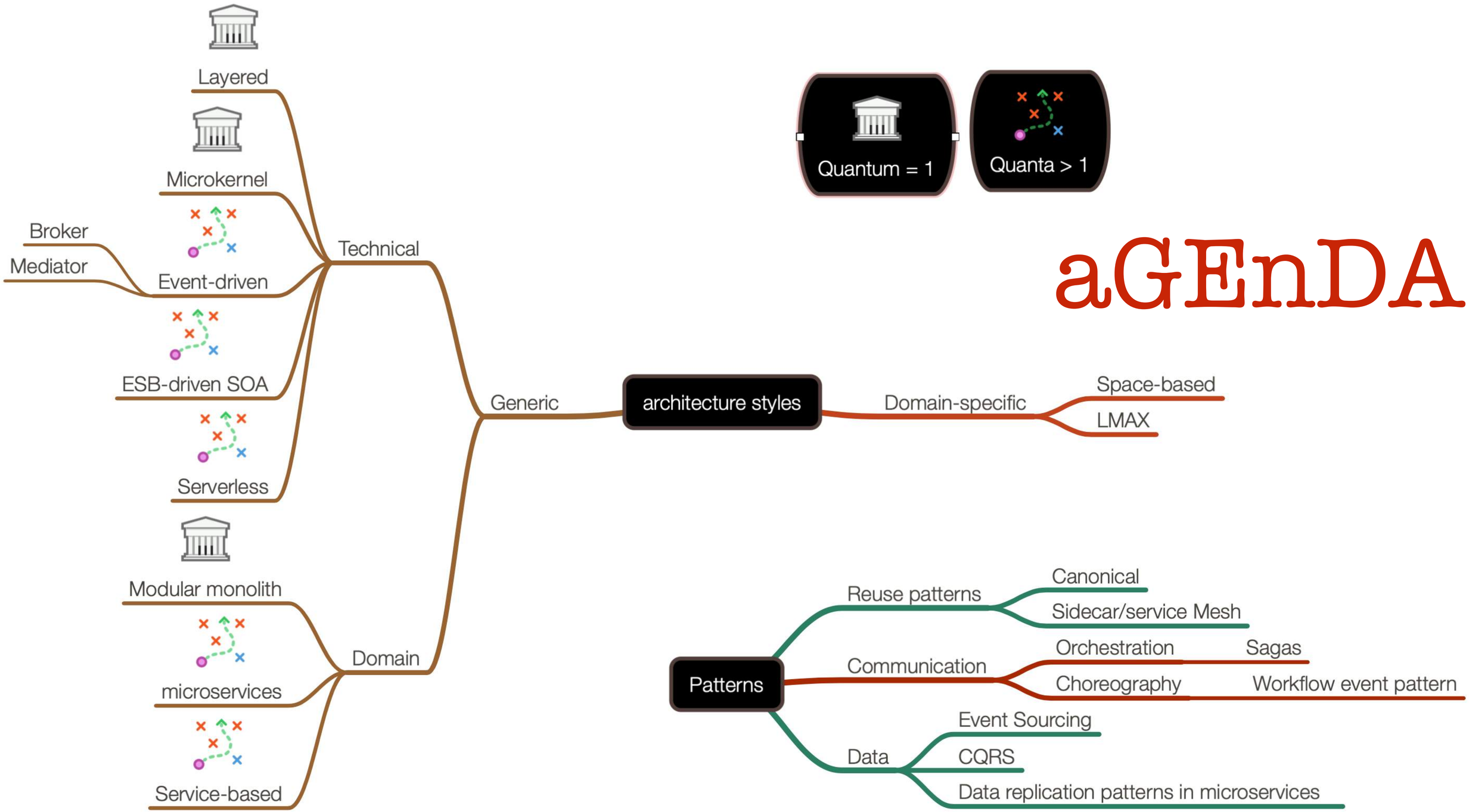
Modular Monolith

drivers

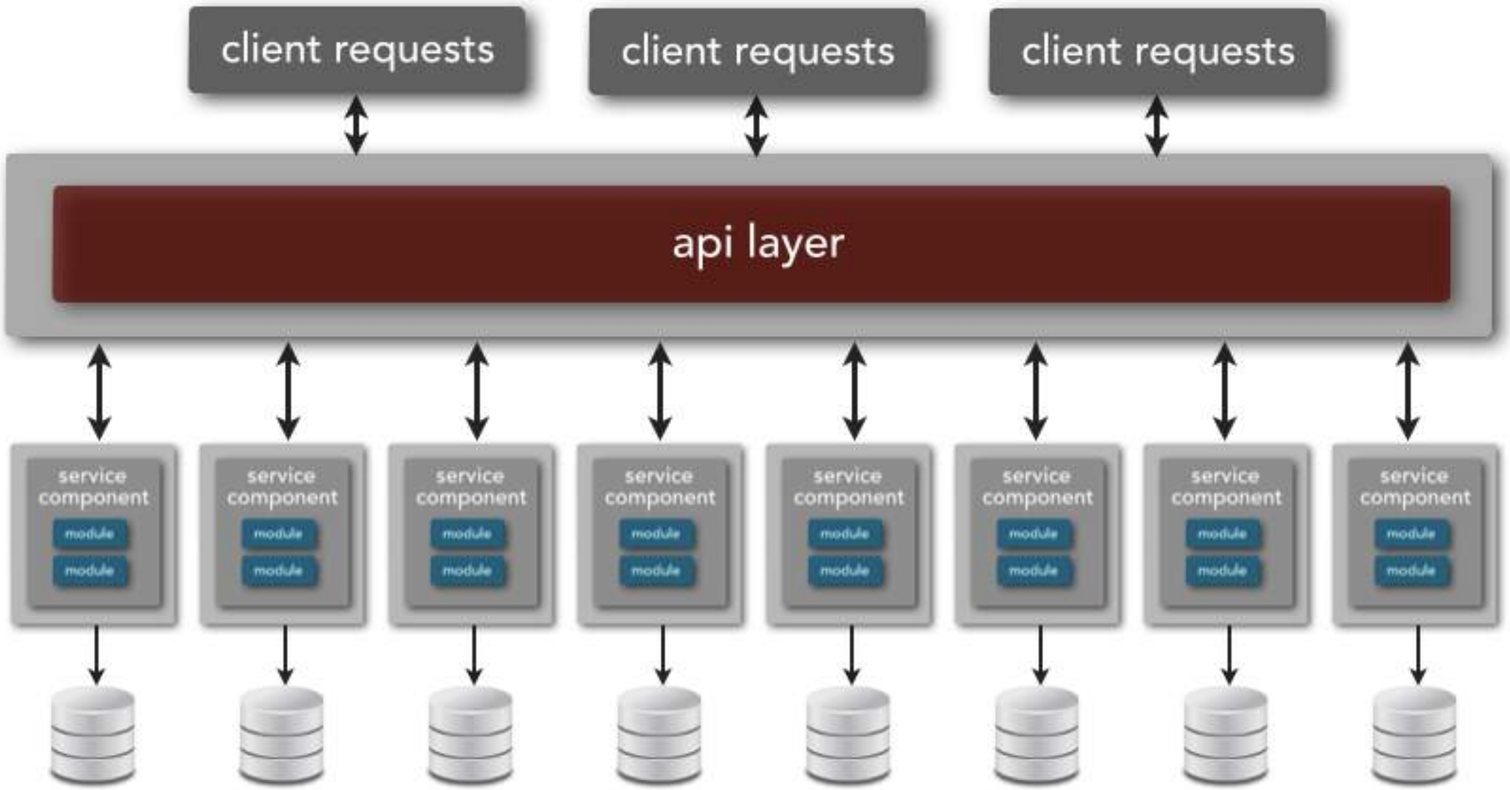


- ✓ agility
- ✓ inverse Conway Maneuver
- ✓ modularity
- ✓ deployability
- ✓ restructurability

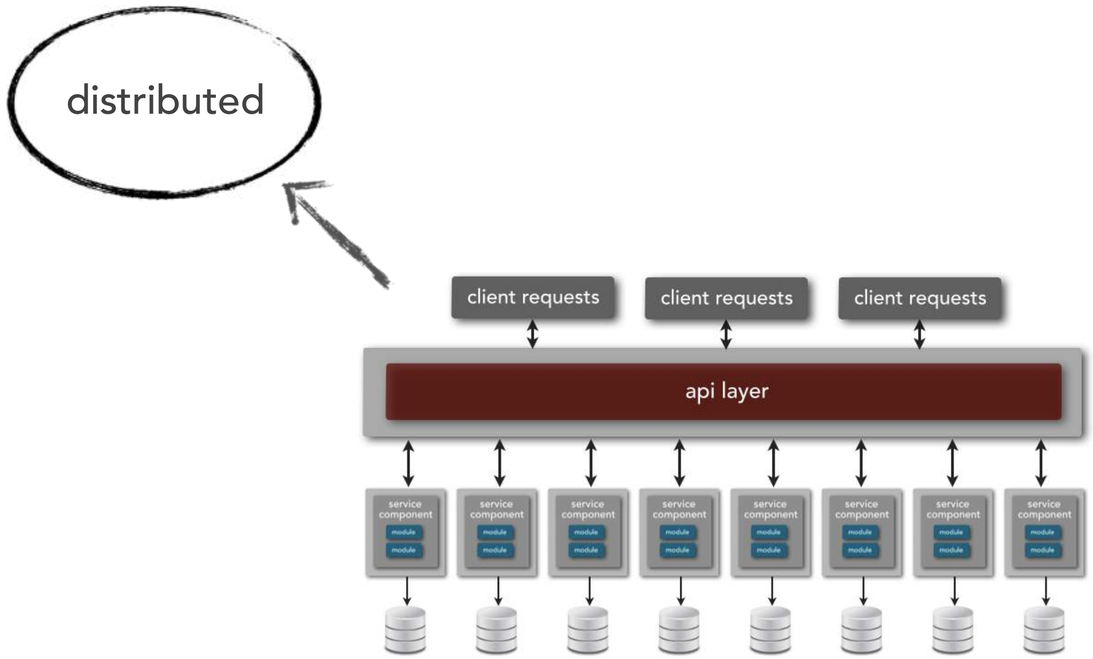
aGEnDA


Quantum = 1
Quanta > 1
architecture styles
Generic
Technical
Layered
Microkernel
Event-driven
Broker
Mediator
ESB-driven SOA
Serverless
Domain
Modular monolith
microservices
Service-based
Domain-specific
Space-based
LMAX
Patterns
Reuse patterns
Canonical
Sidecar/service Mesh
Communication
Orchestration
Sagas
Choreography
Workflow event pattern
Data
Event Sourcing
CQRS
Data replication patterns in microservices

Microservices Architecture

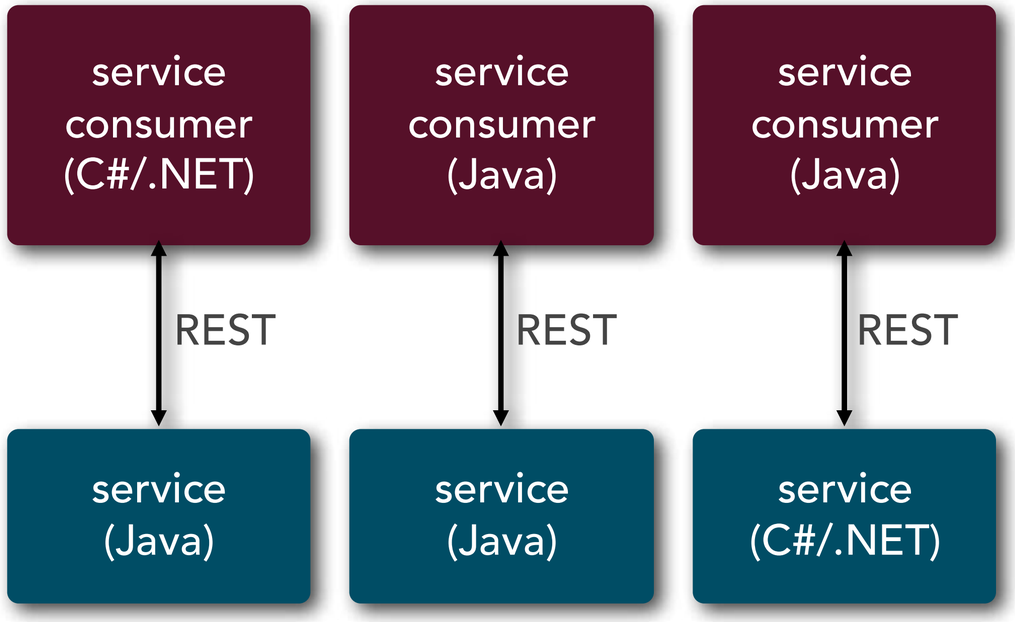


Microservices Architecture

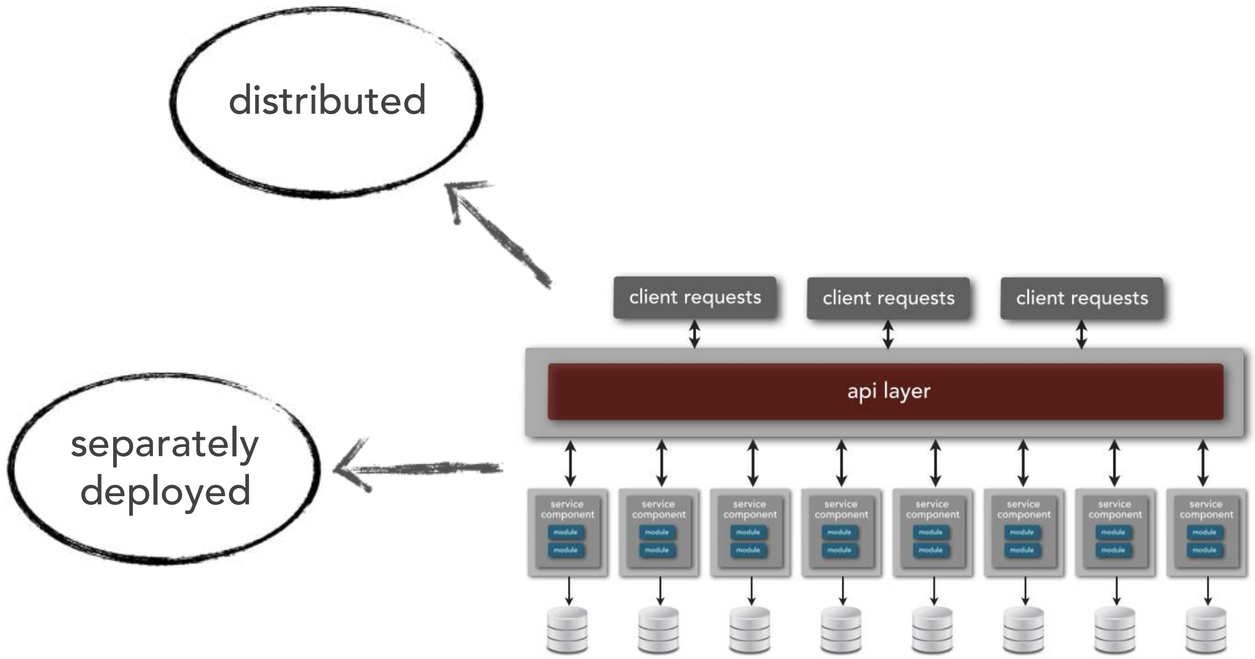


Microservices Architecture

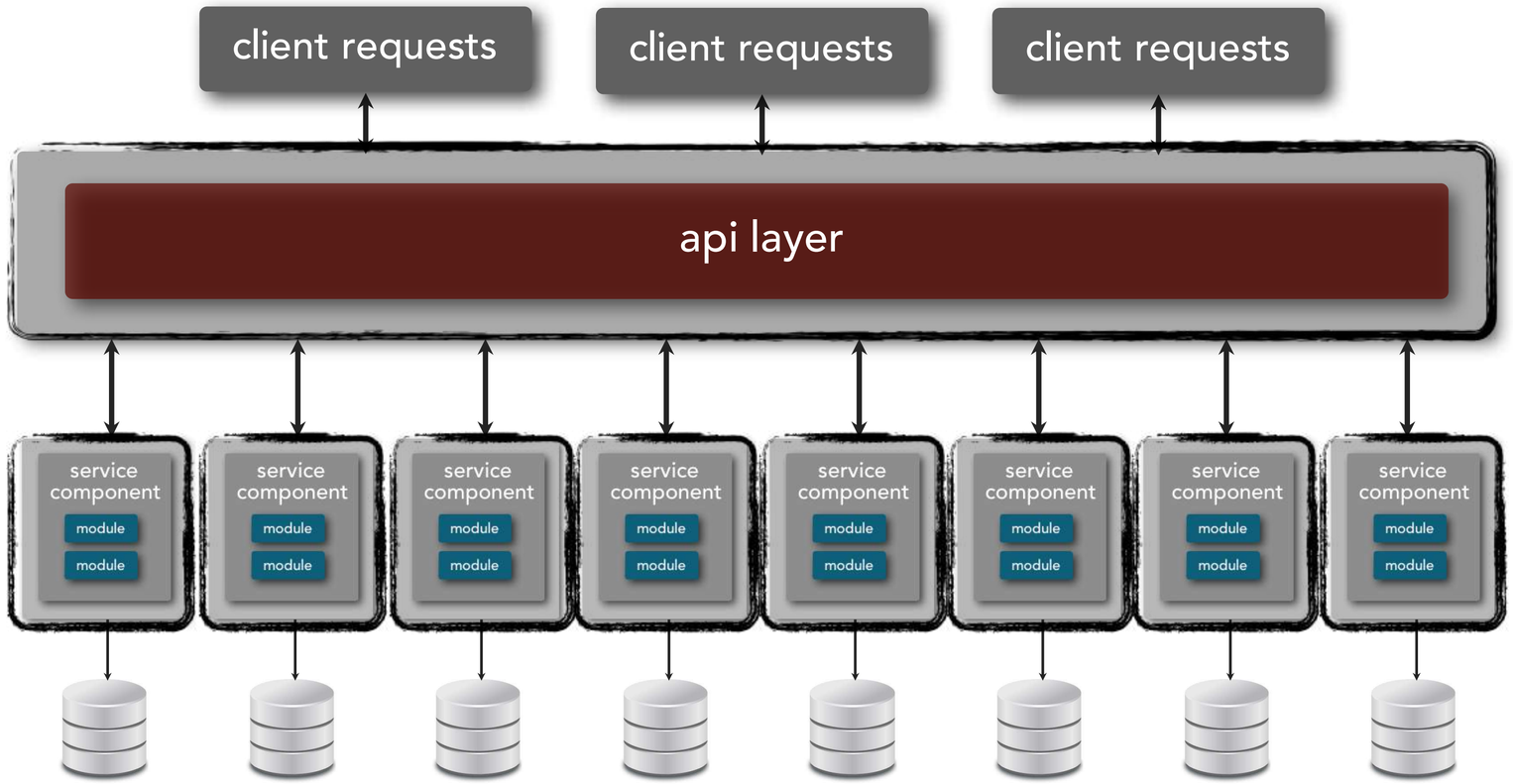
protocol-aware heterogeneous interoperability



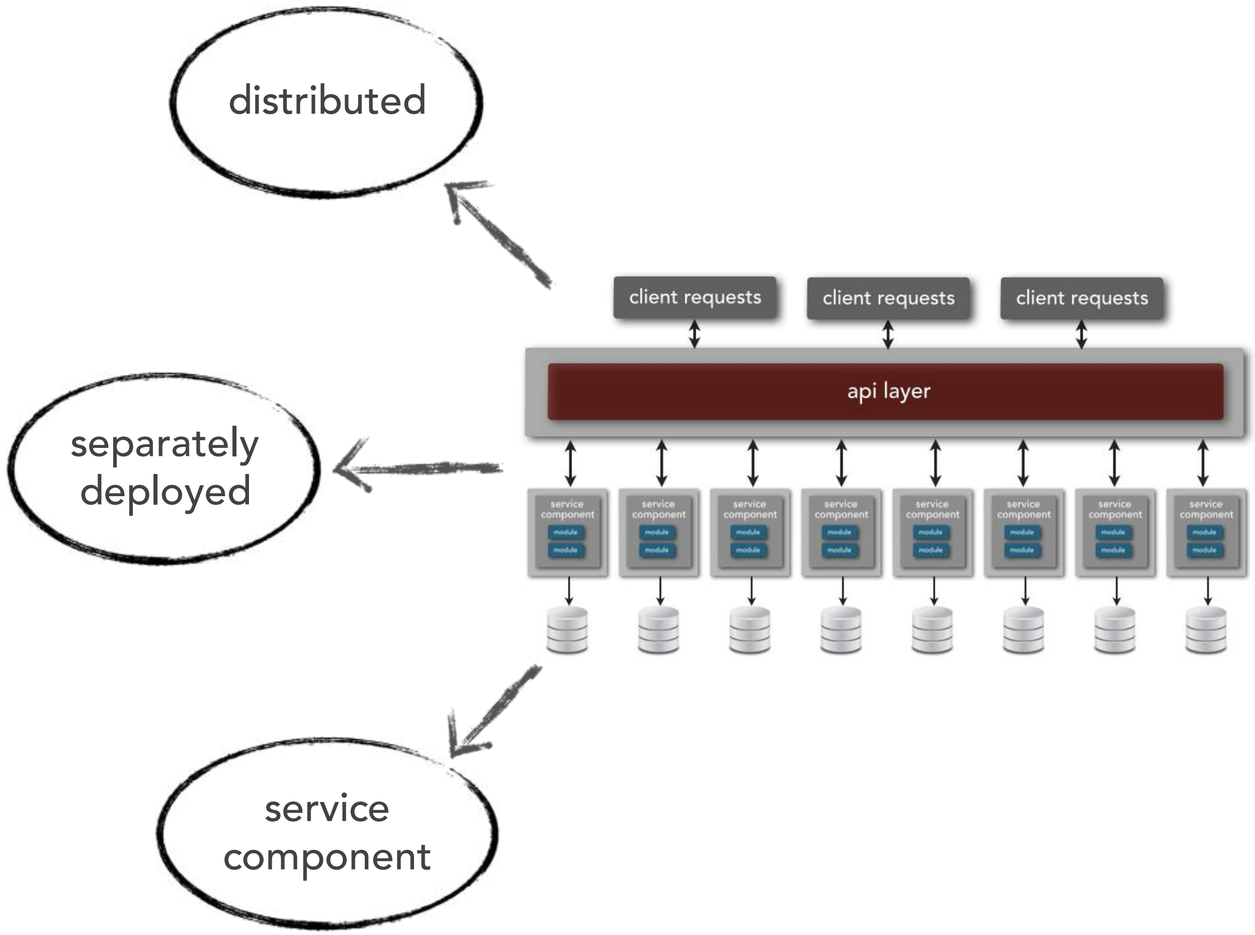
Microservices Architecture



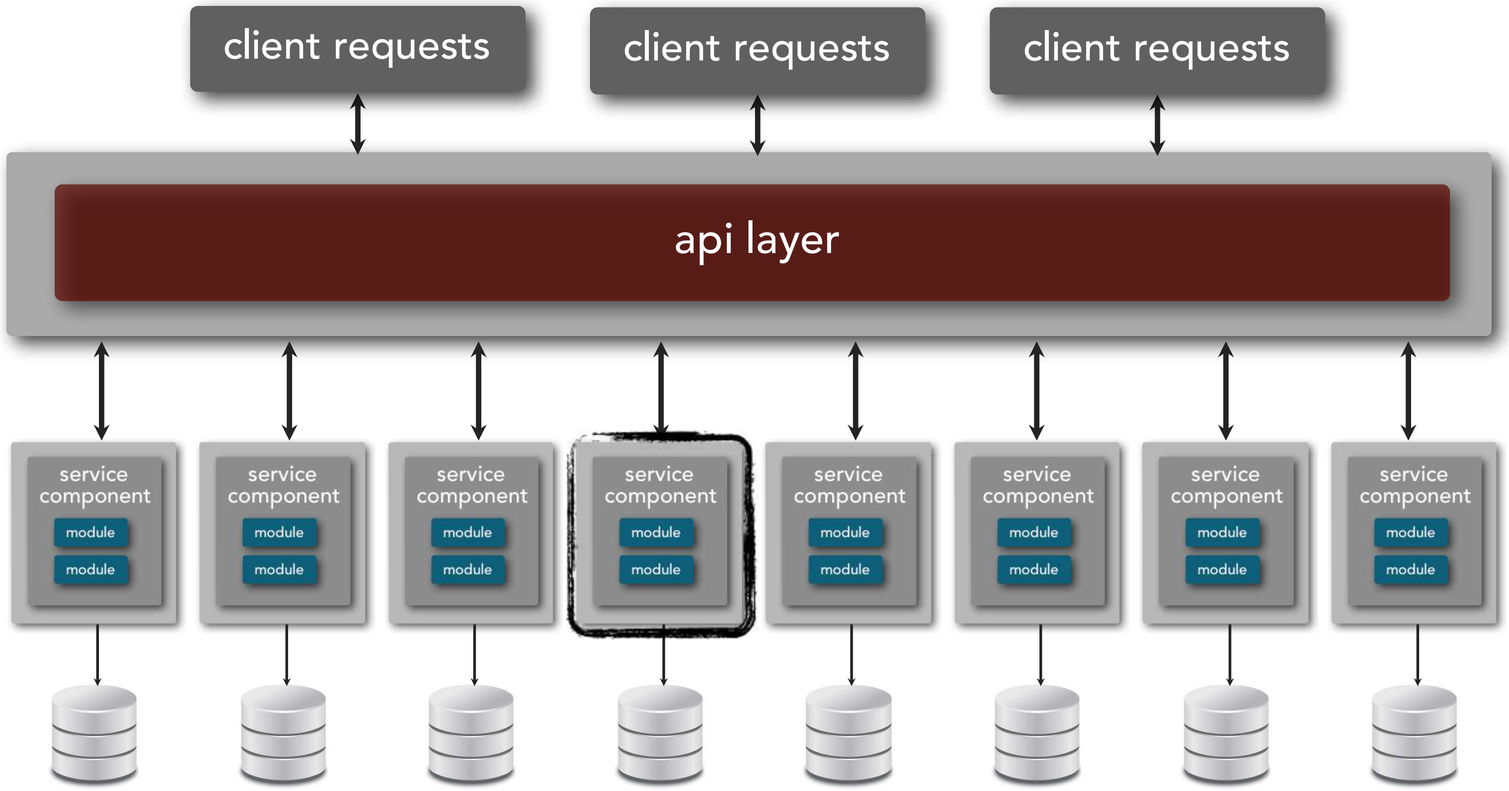
Microservices Architecture



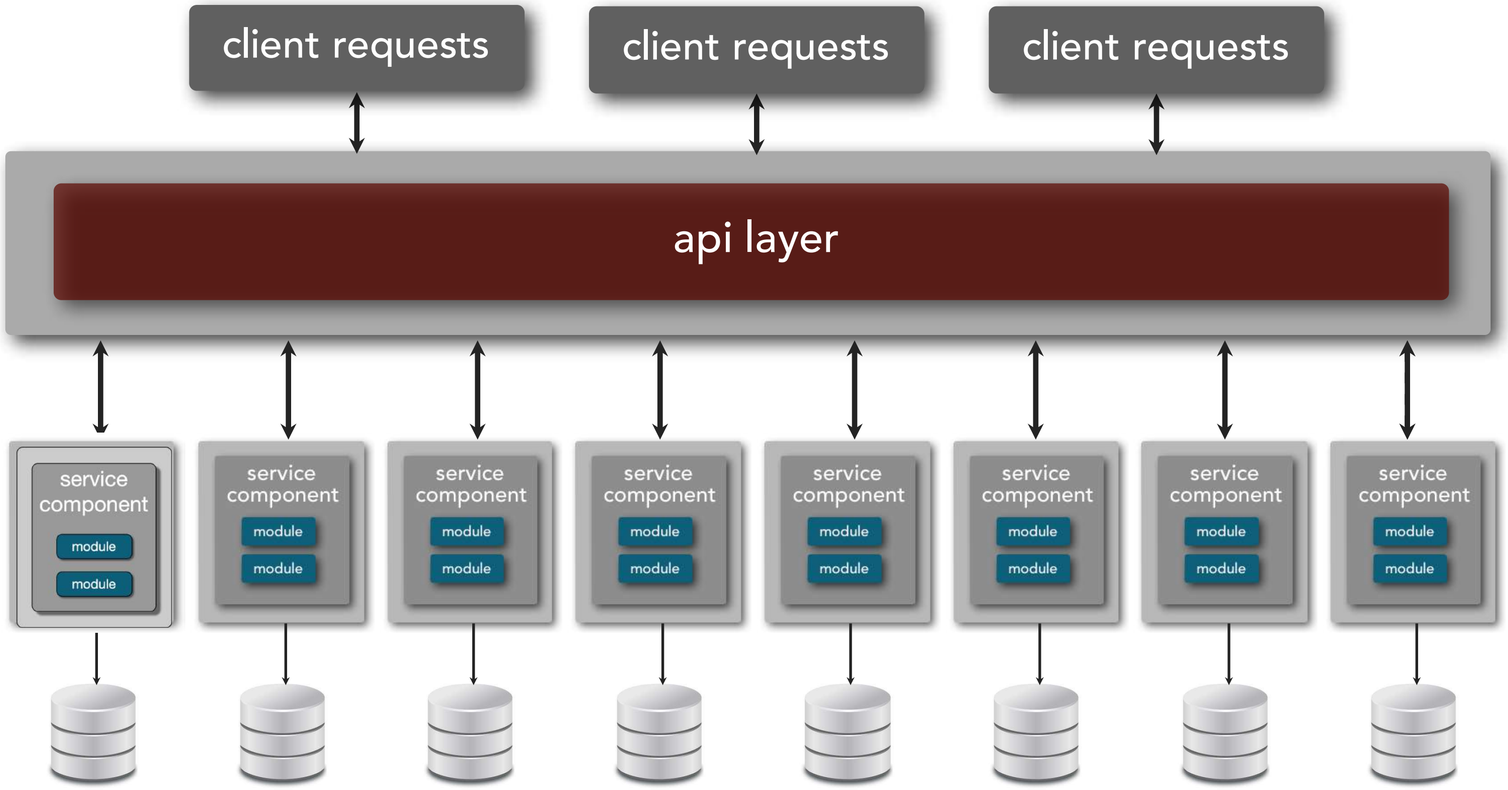
Microservices Architecture



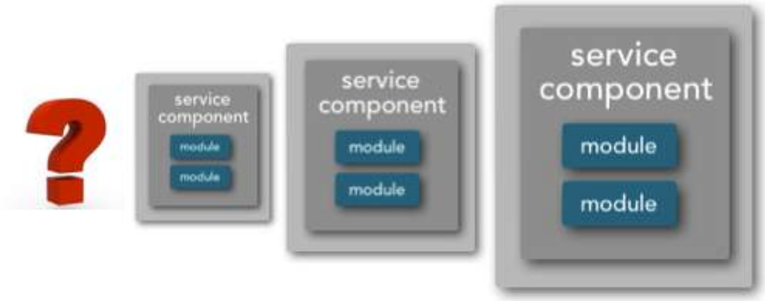
Microservices Architecture



Microservices Architecture

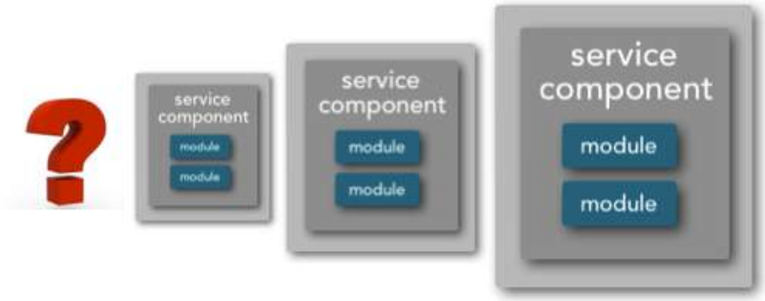


Microservices Architecture

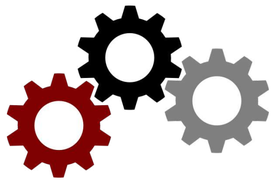


what is the right size for a microservice?

Microservices Architecture

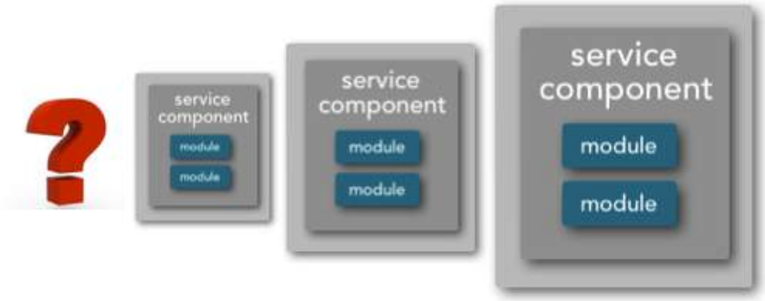


what is the right size for a microservice?

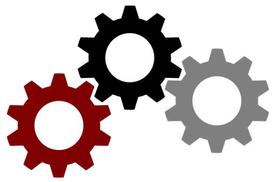


purpose

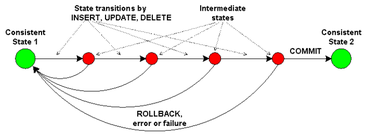
Microservices Architecture



what is the right size for a microservice?

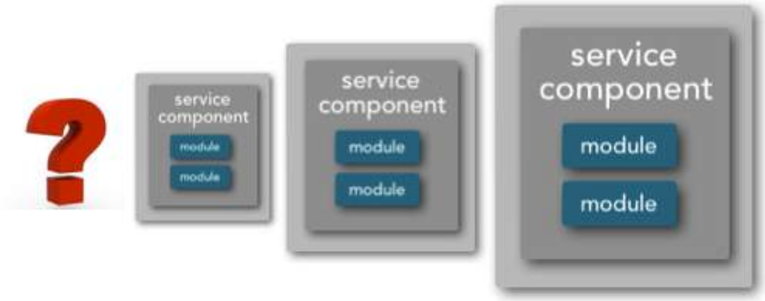


purpose

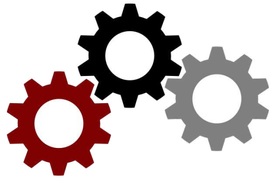


transactions

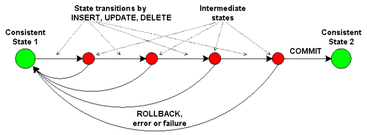
Microservices Architecture



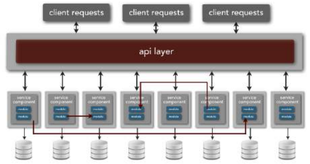
what is the right size for a microservice?



purpose

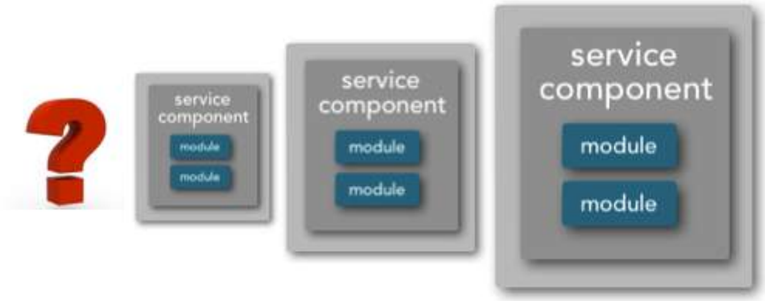


transactions

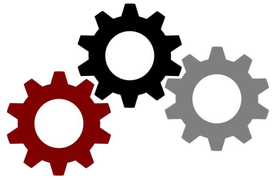


choreography

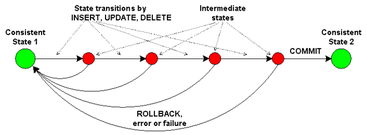
Microservices Architecture



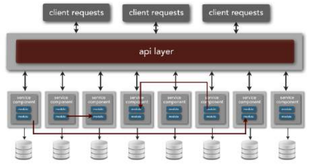
what is the right size for a microservice?



purpose



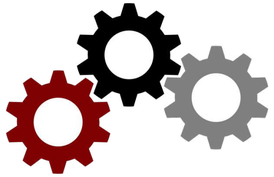
transactions



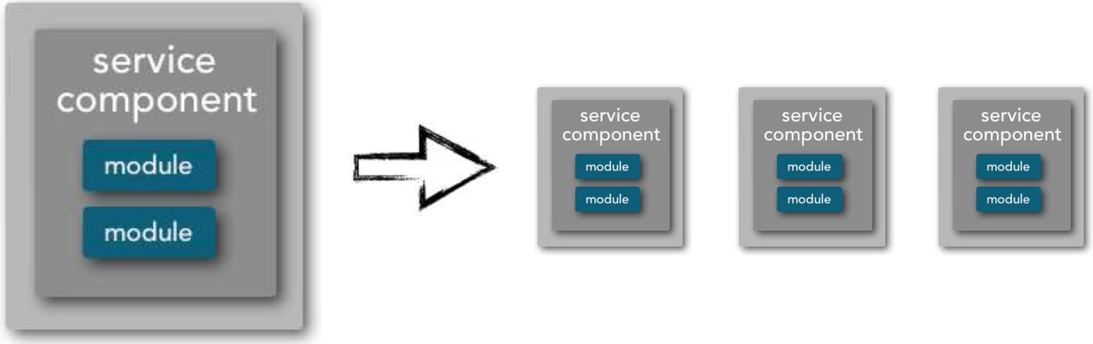
choreography

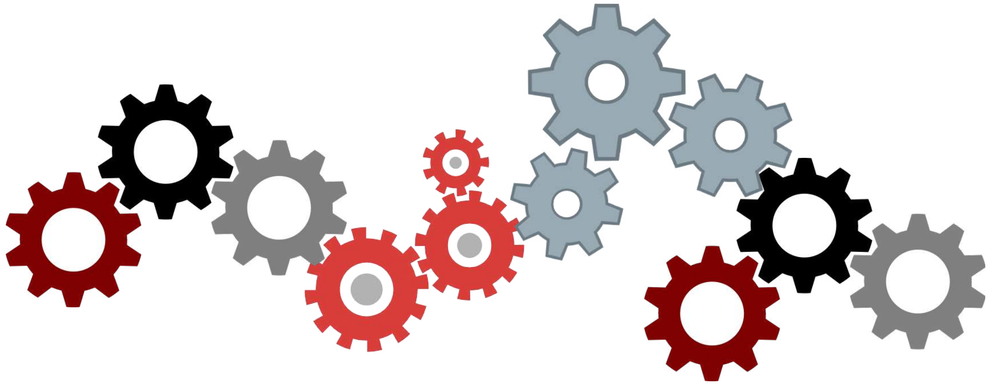
"Microservice" is a ***label***, not a ***description***.

Microservices Architecture



purpose

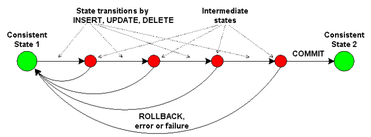




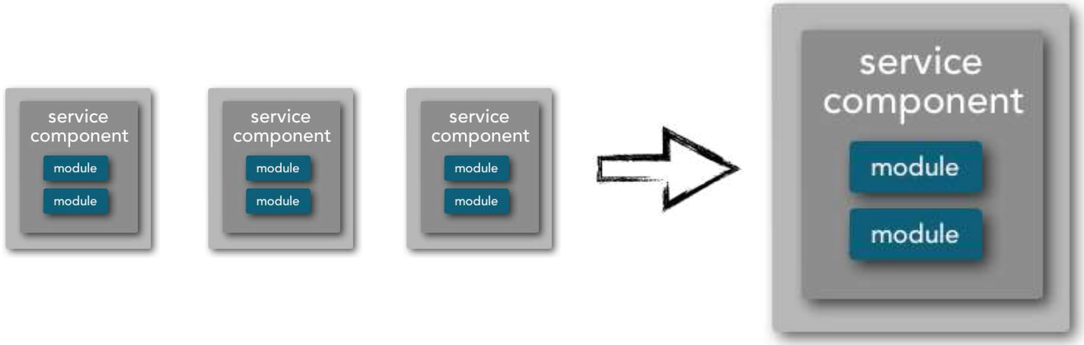
service scope and function

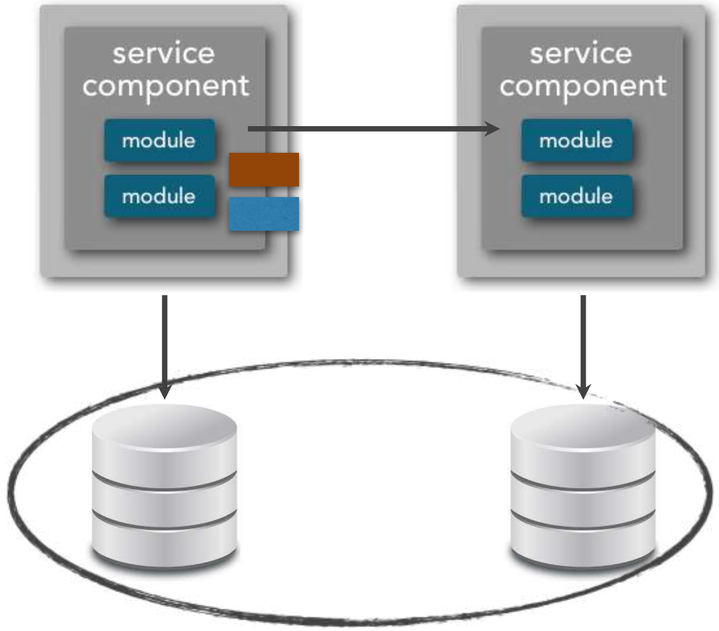
(single-purpose function)

Microservices Architecture



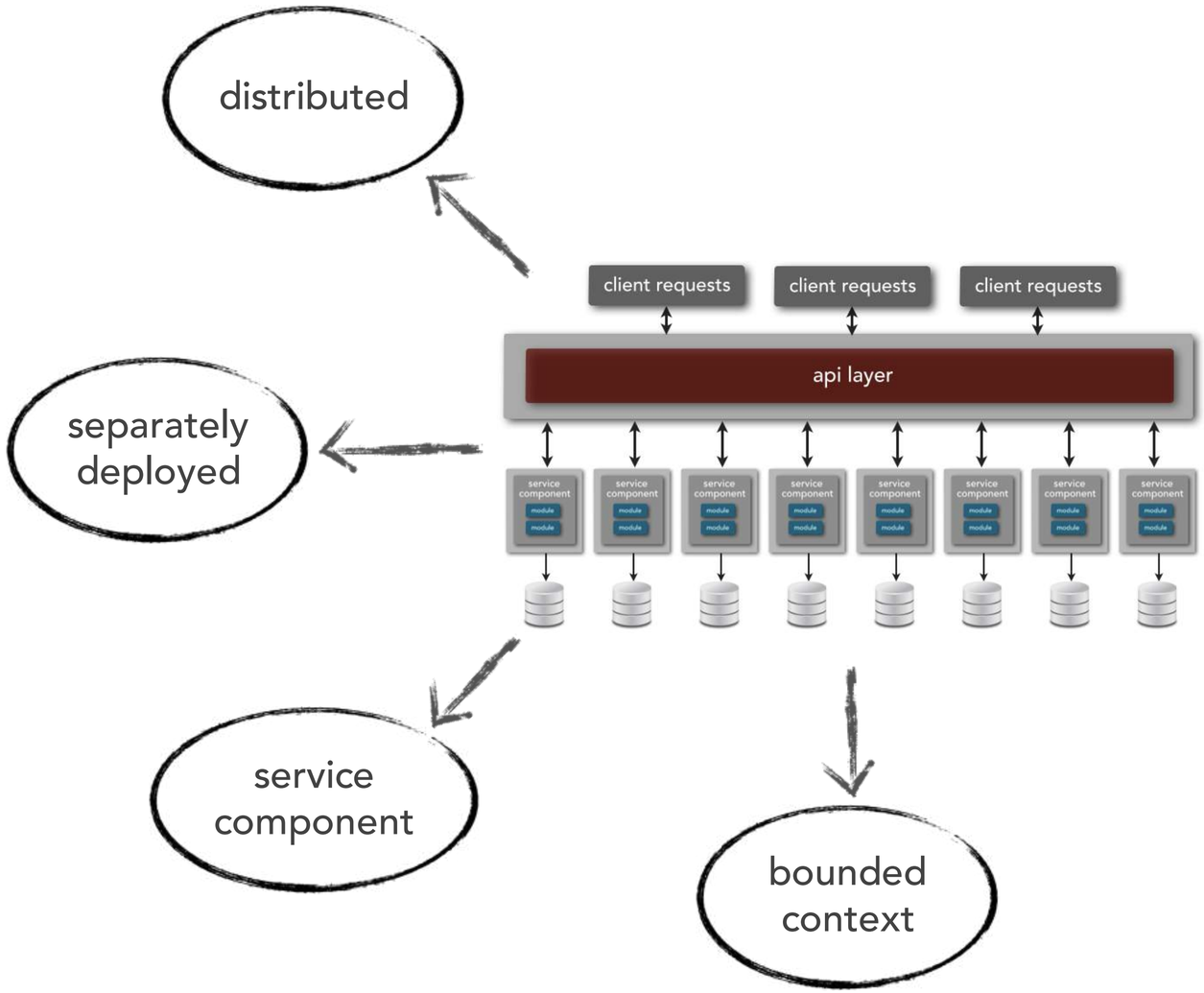
transactions



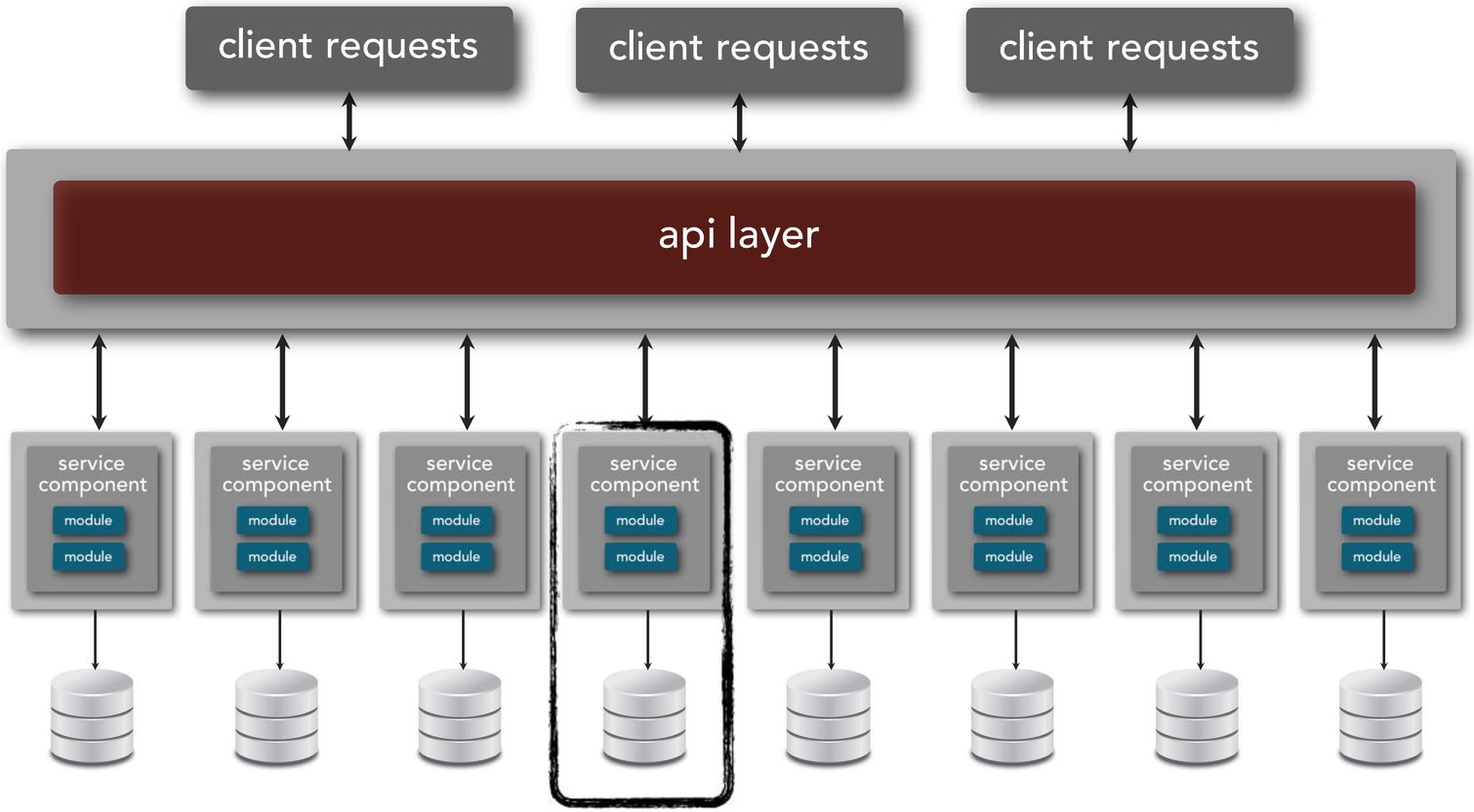


no acid
transaction

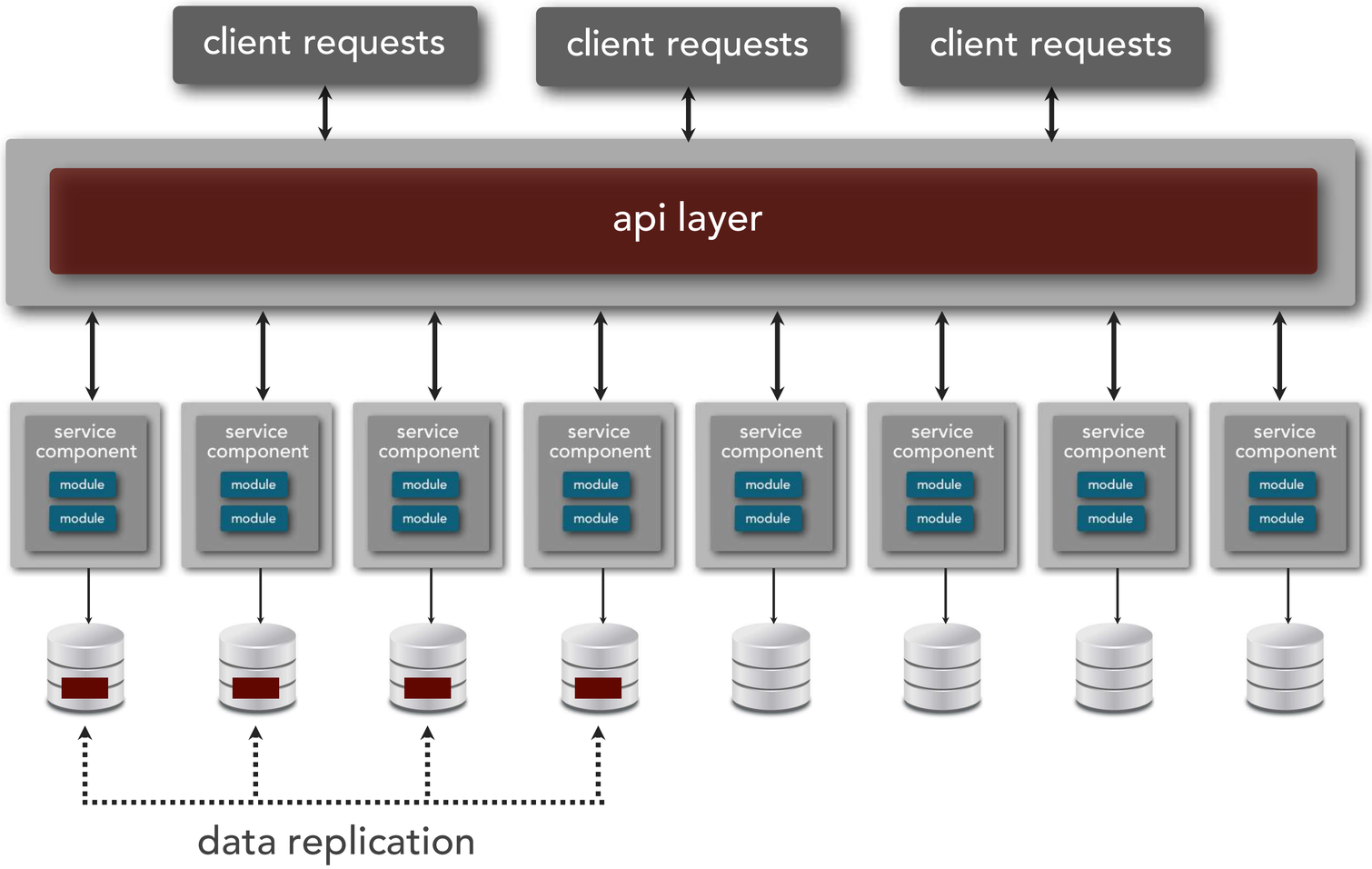
Microservices Architecture



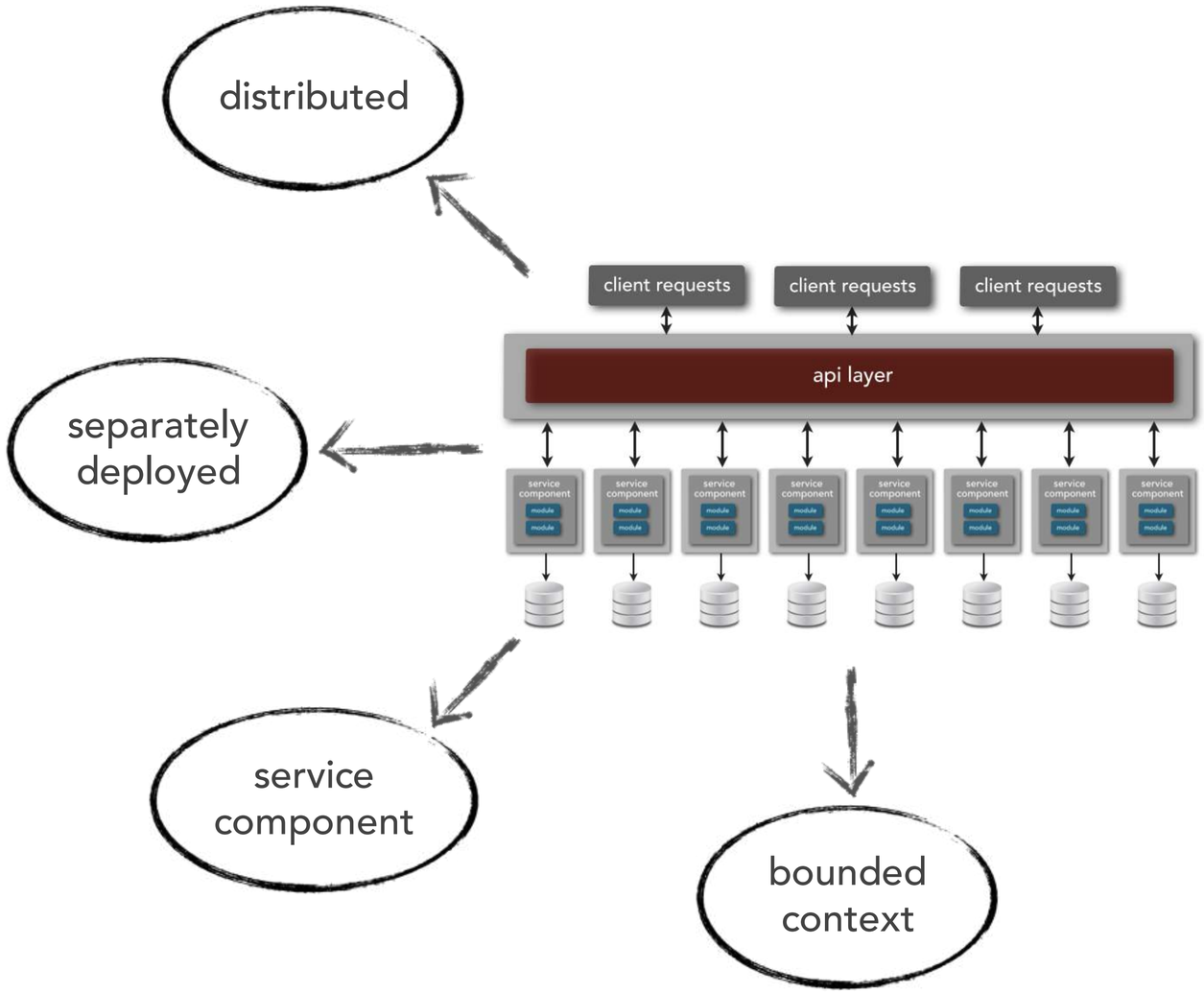
Microservices Architecture



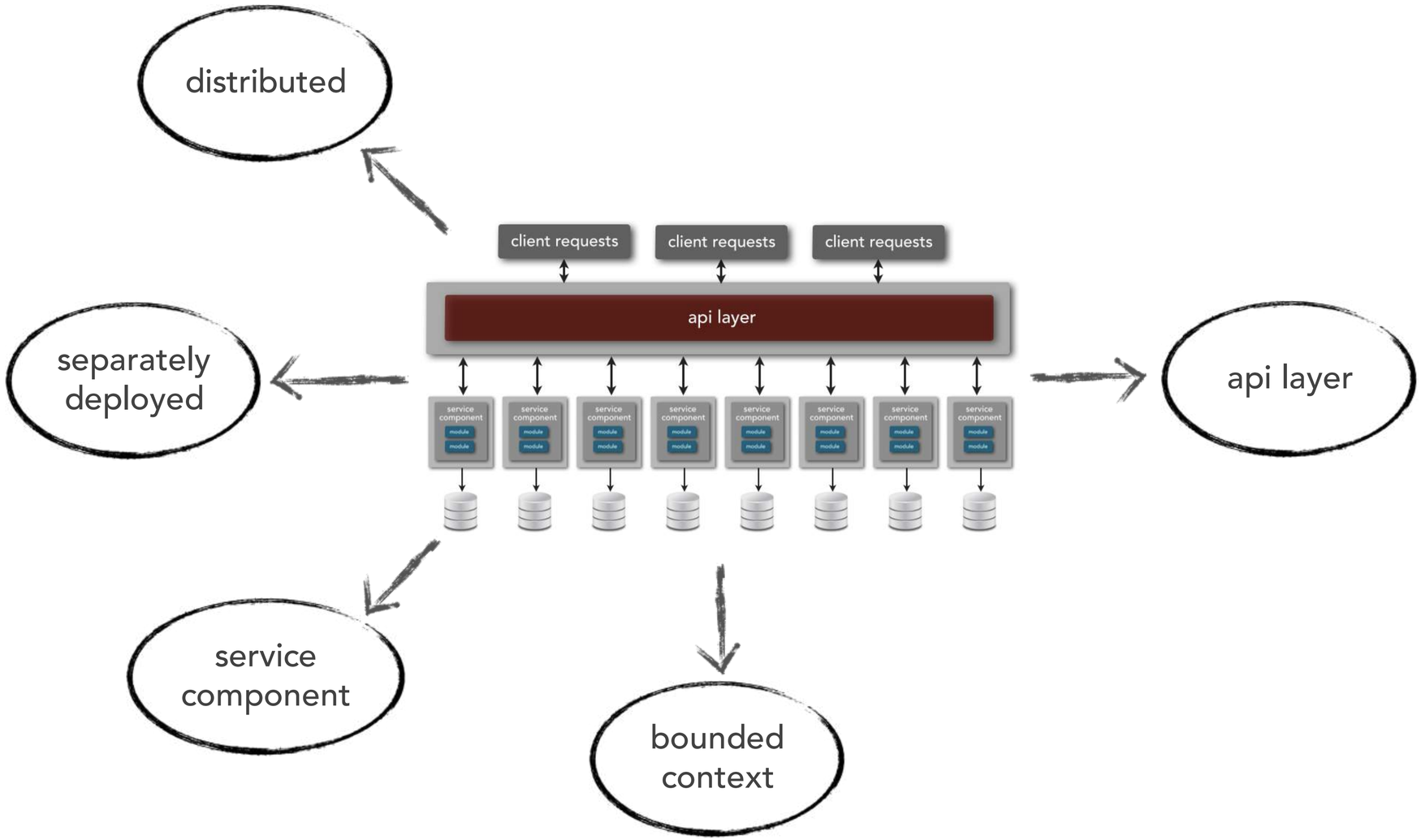
Microservices Architecture



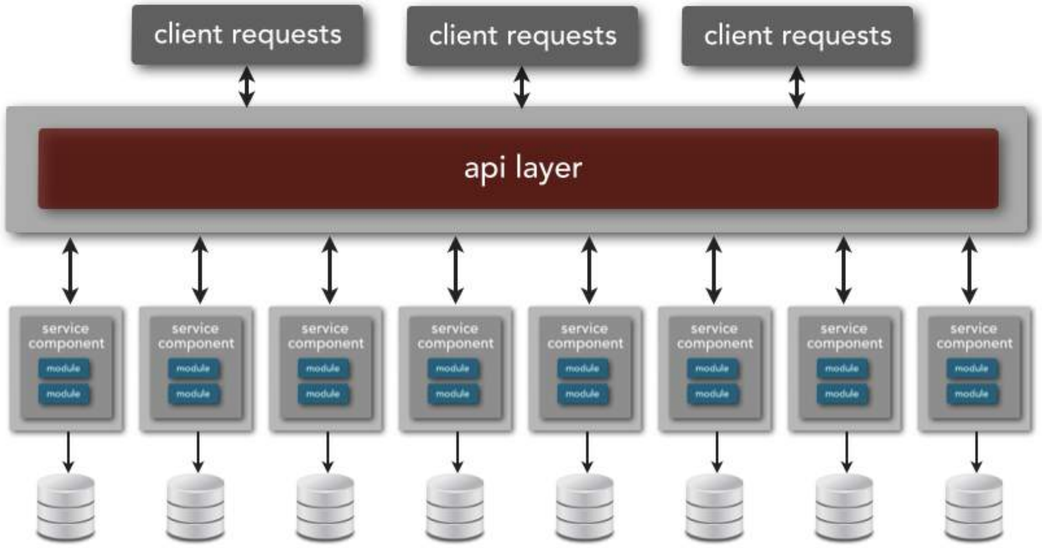
microservices architecture



Microservices Architecture

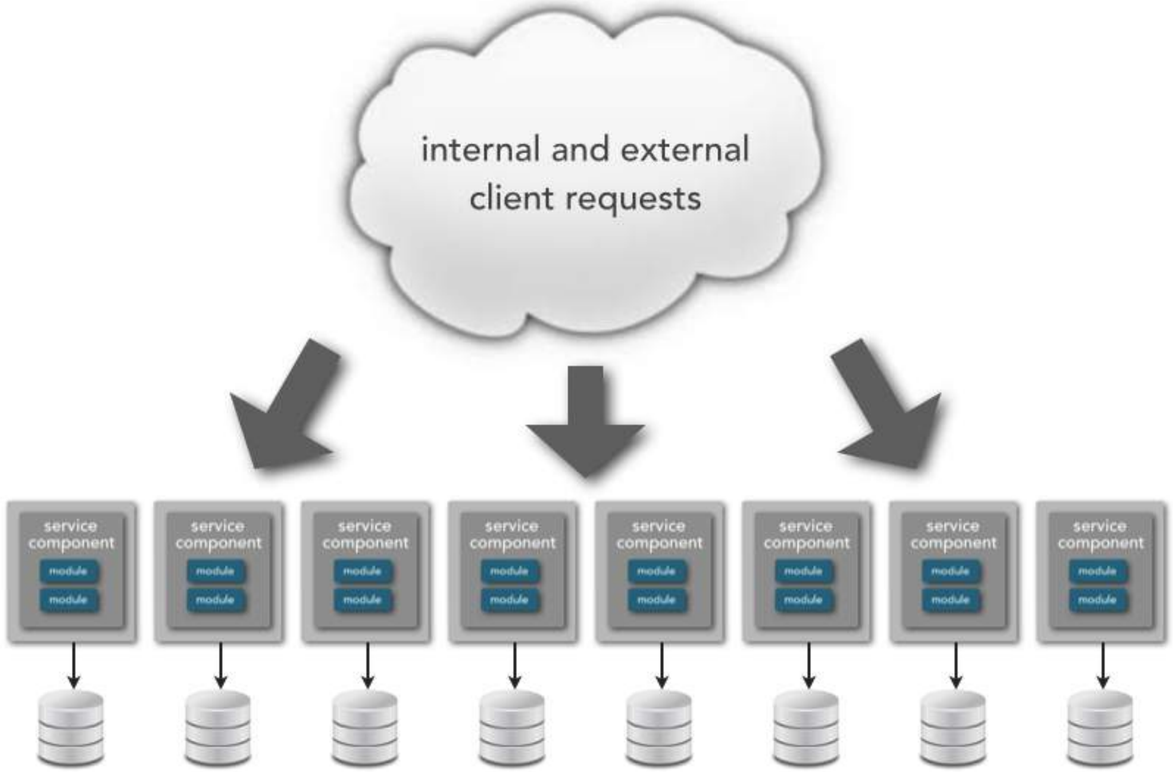


API Layer

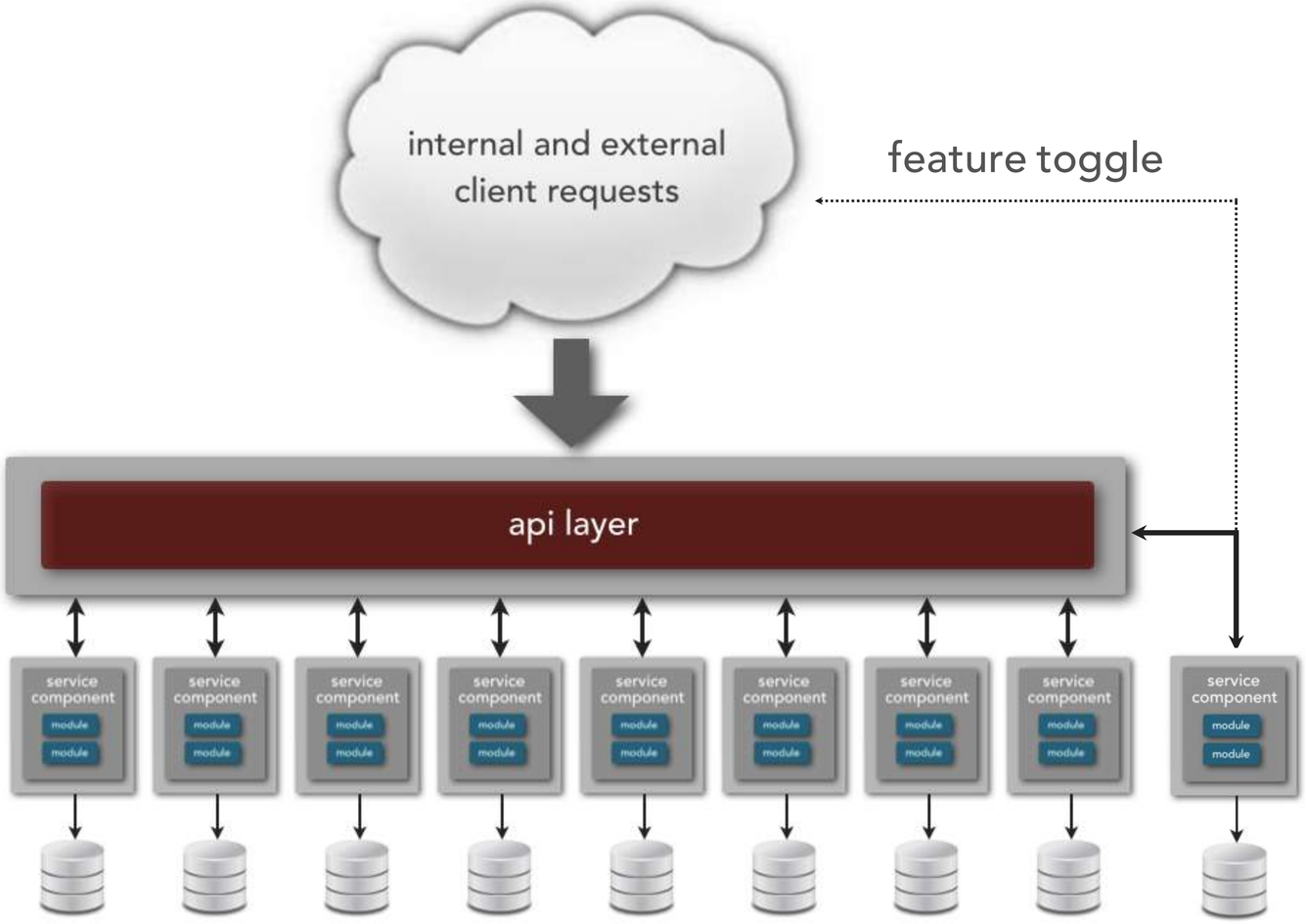


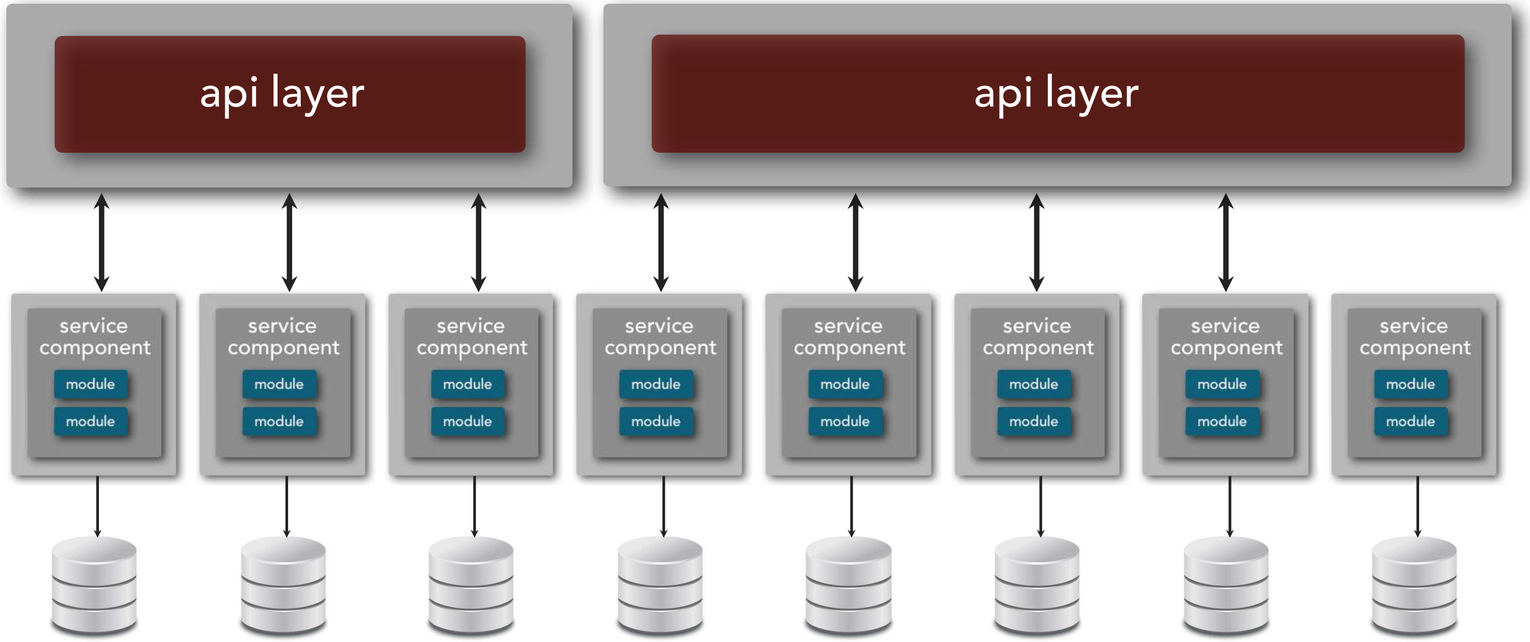
hides the actual endpoint of the service, exposing only those services available for public consumption

API Layer

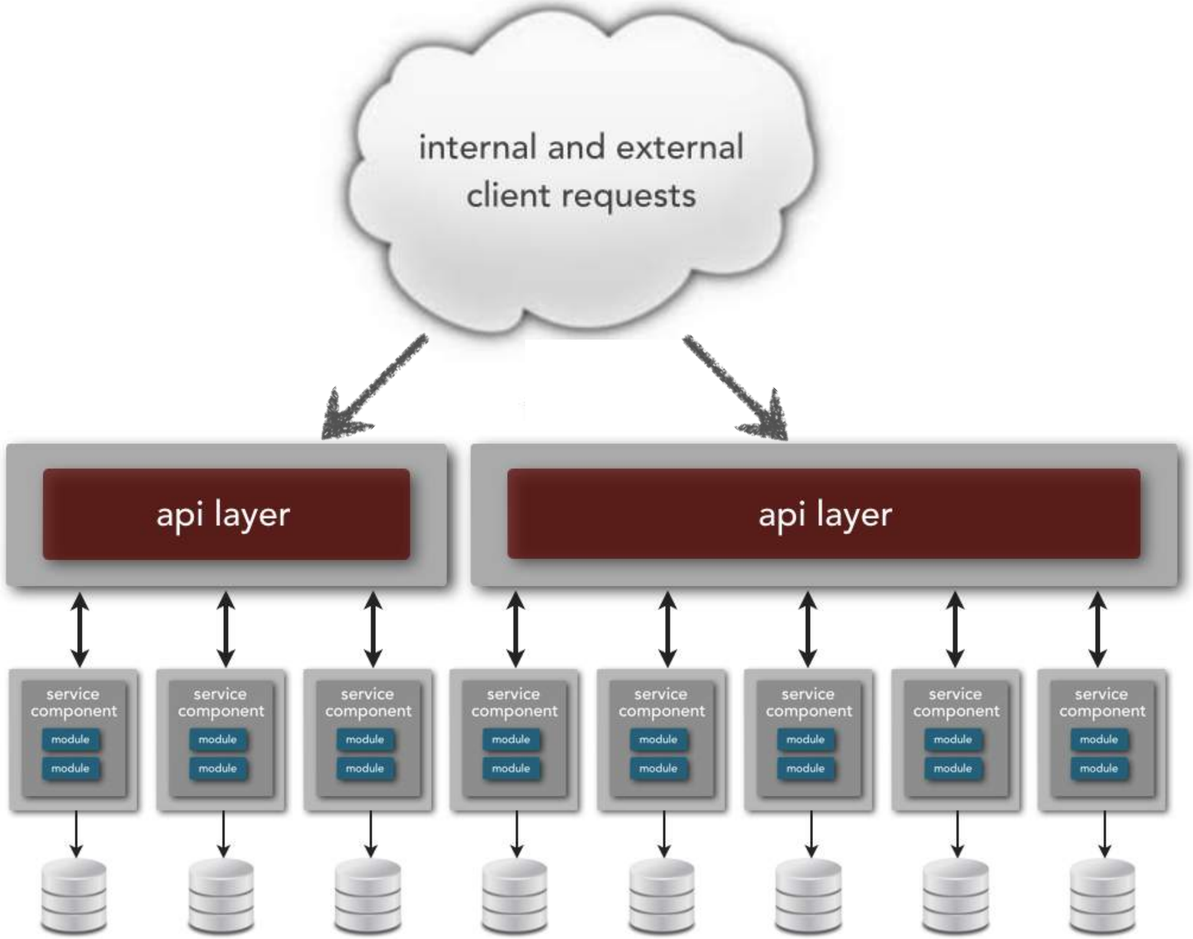

internal and external
client requests
service component
module
module

API Layer

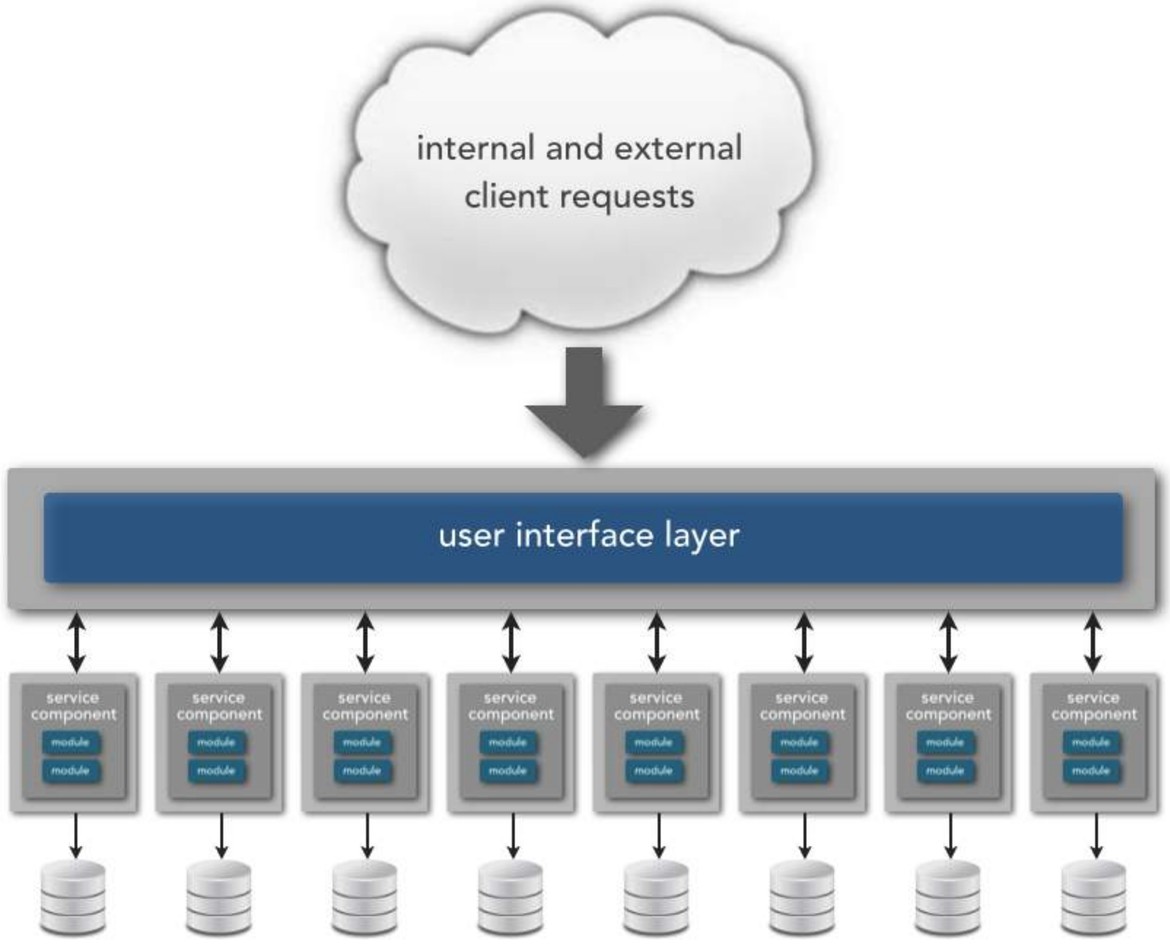

internal and external
client requests
feature toggle
api layer
service component
module
module


api layer
api layer
service component
module
module
service component
module
module
service component
module
module
service component
module
module
service component
module
module
service component
module
module
service component
module
module
service component
module
module

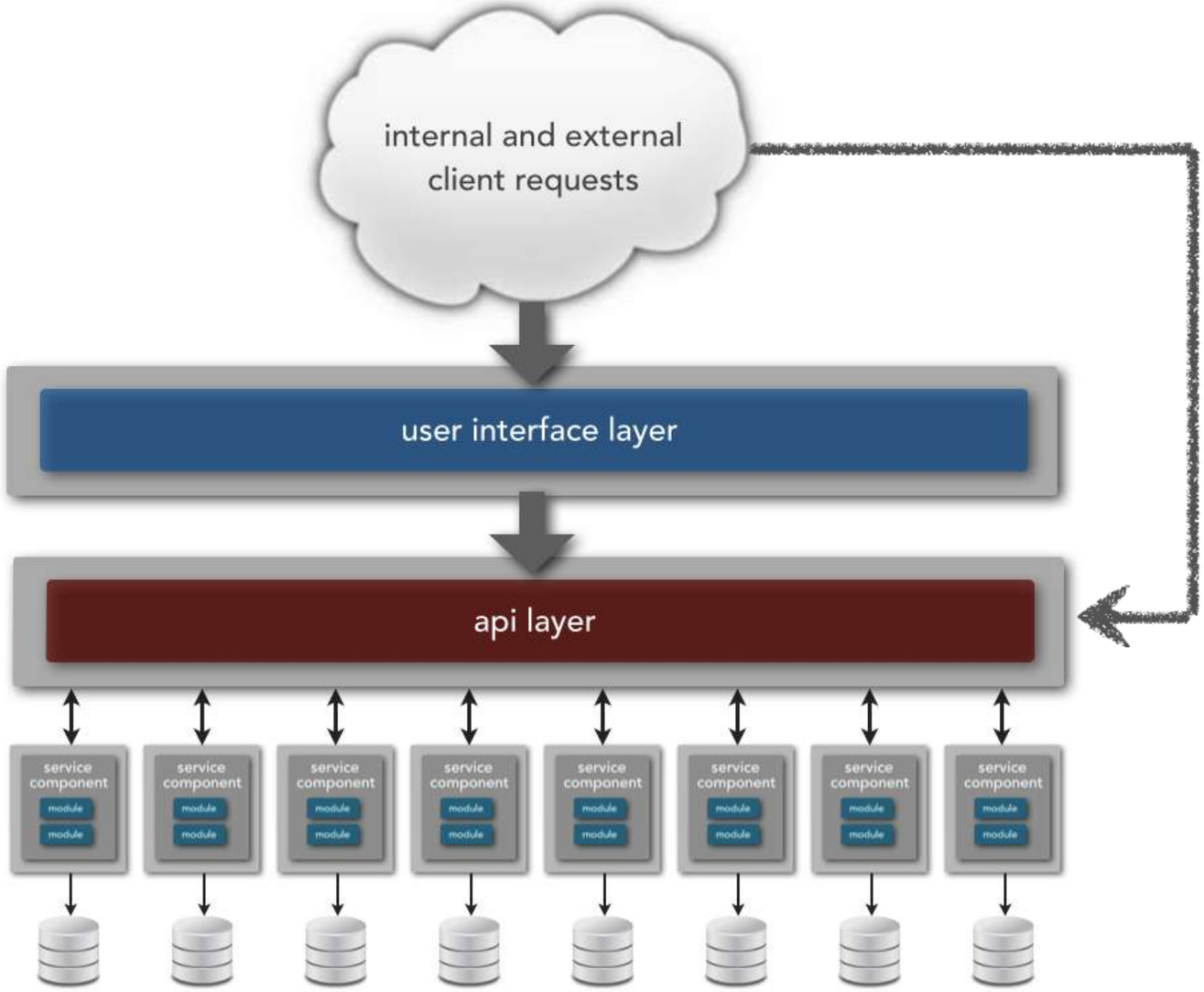
API Layer


internal and external
client requests
api layer
api layer
service component
module
module

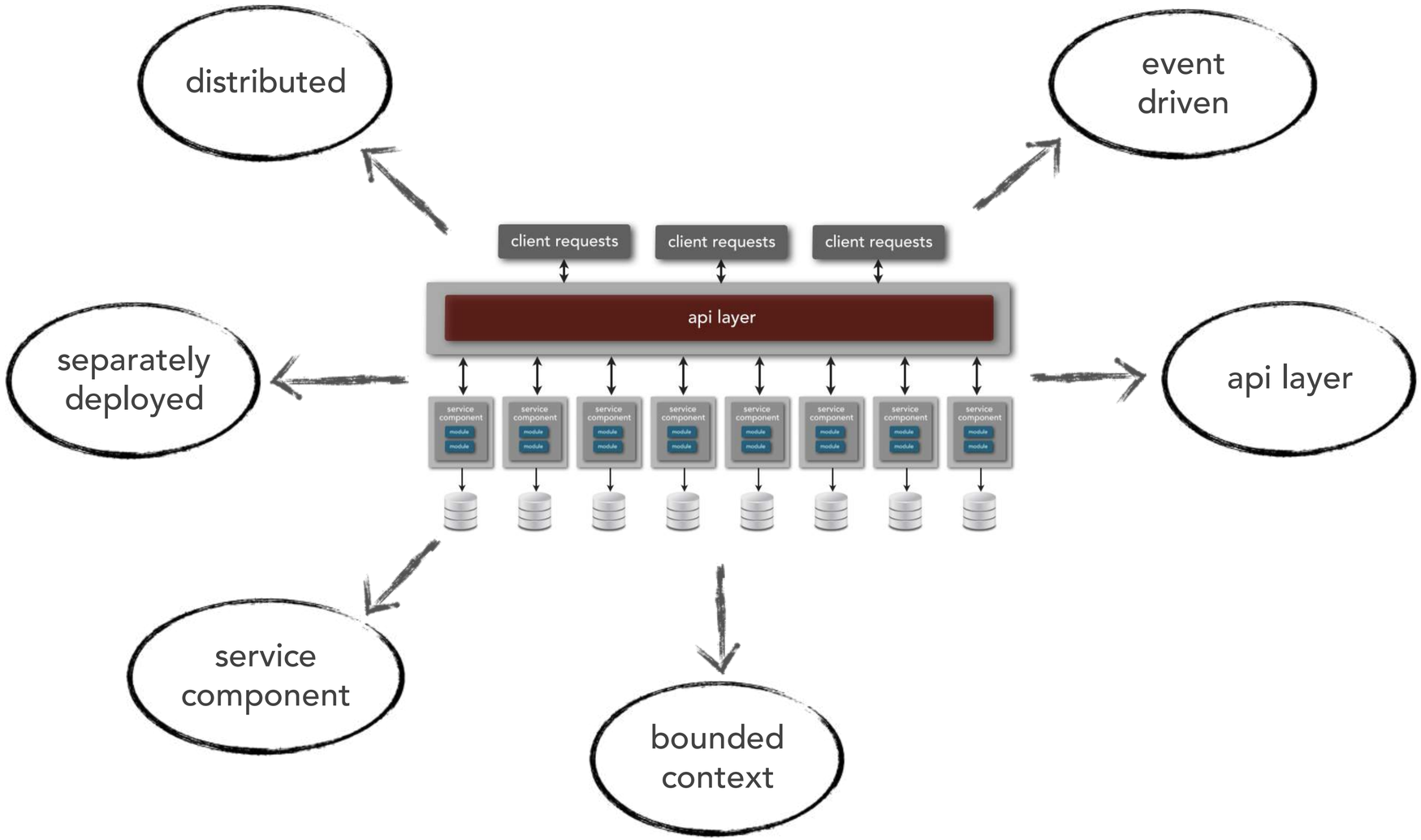
API Layer


internal and external
client requests
user interface layer
service component
module
module

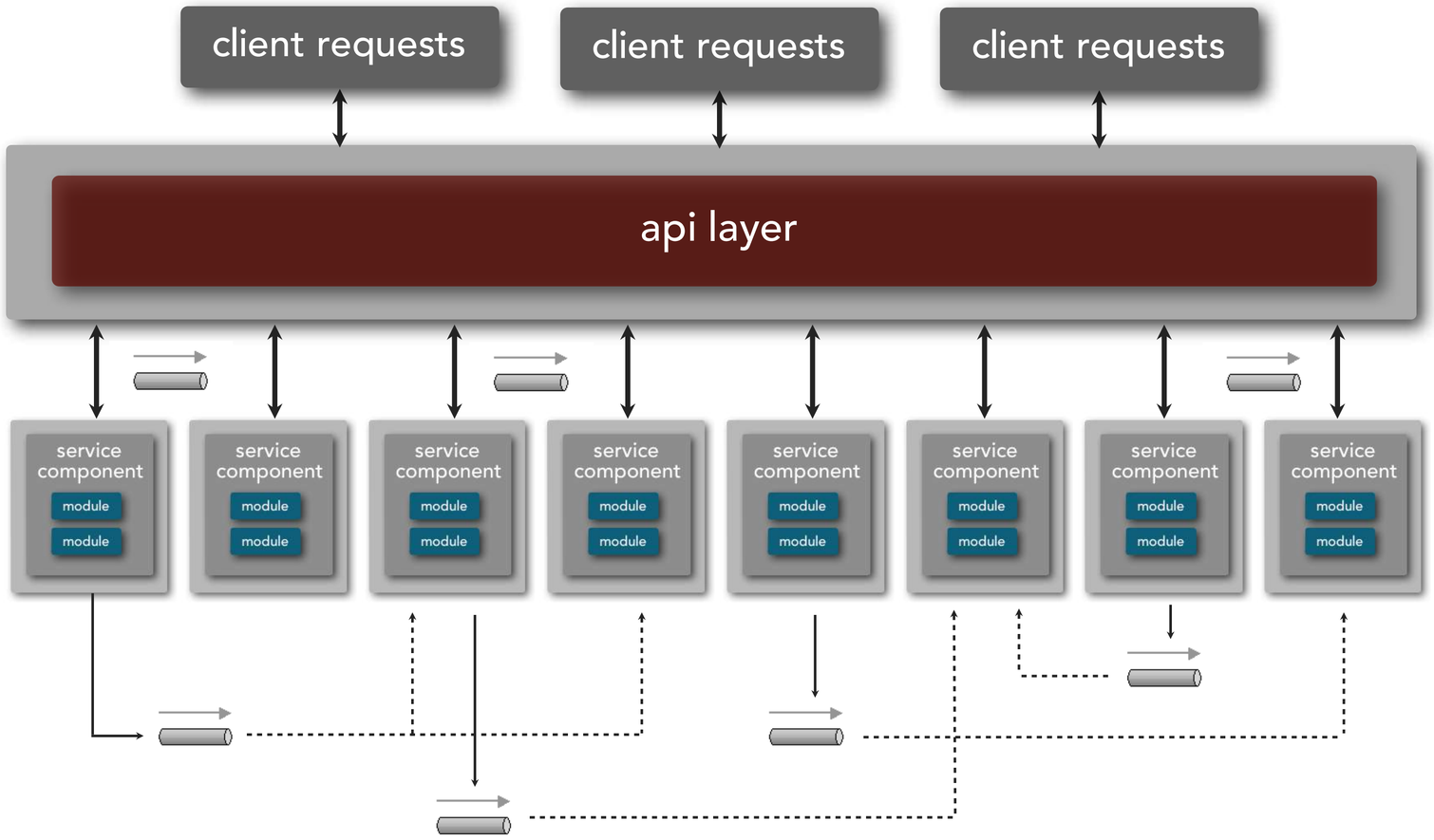
Microservices Architecture



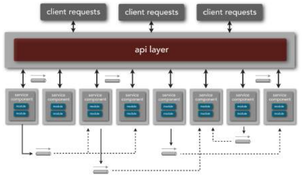
microservices architecture



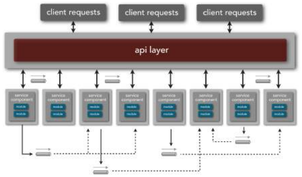
Microservices Architecture



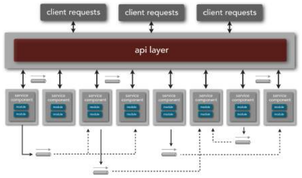
Microservices Architecture



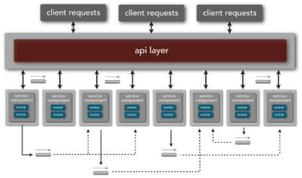
reporting techniques



eventual consistency patterns



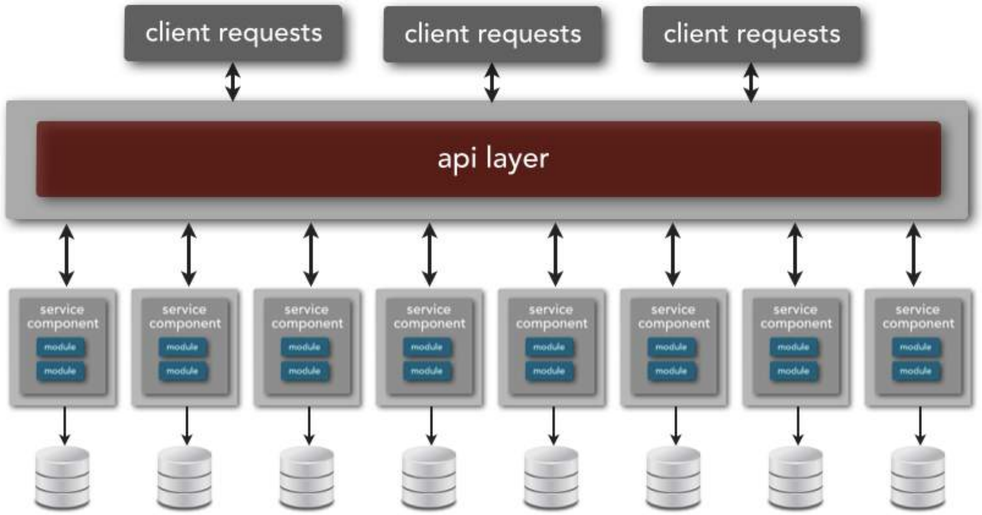
performance tuning



reactive architecture patterns

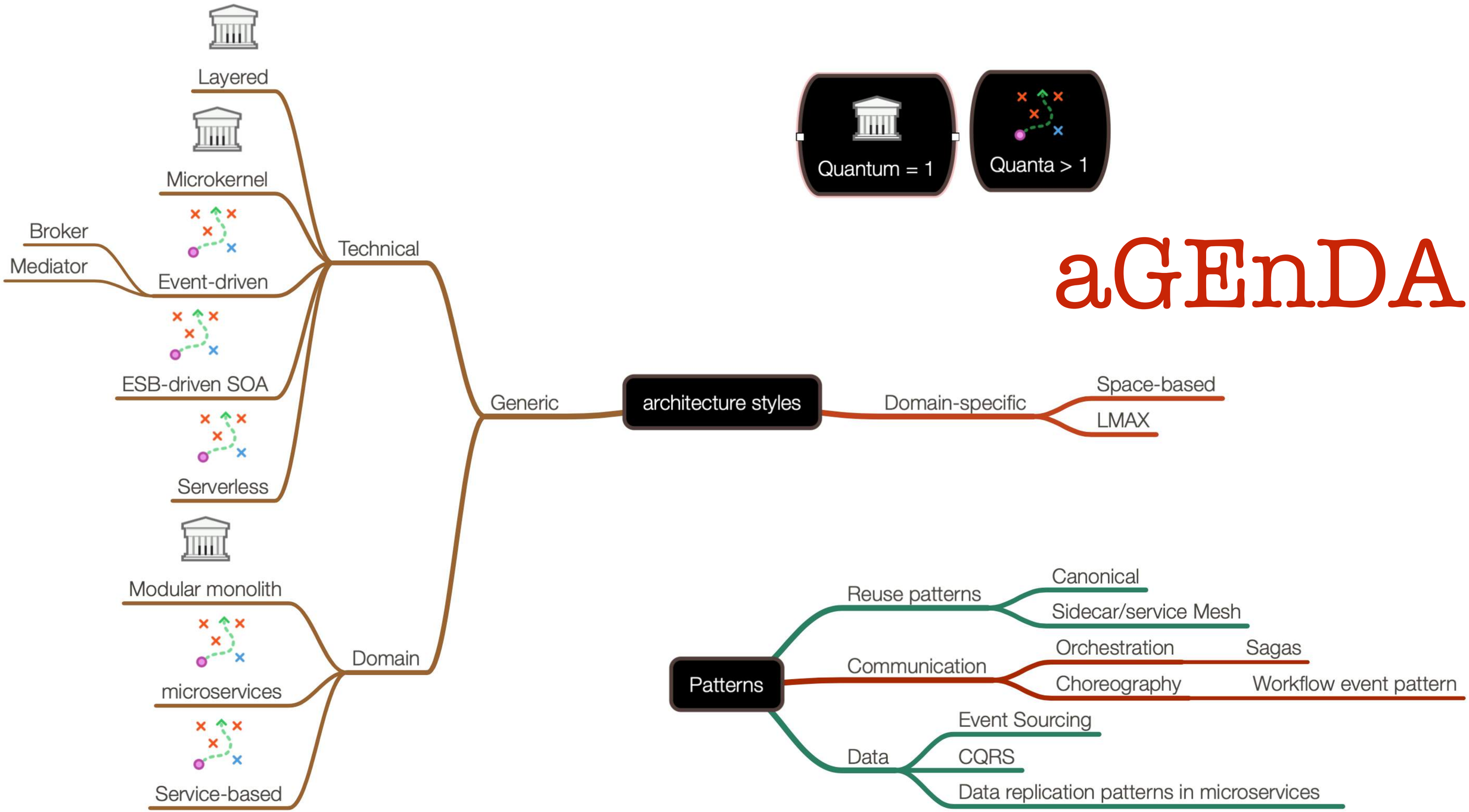
Microservices Architecture

drivers

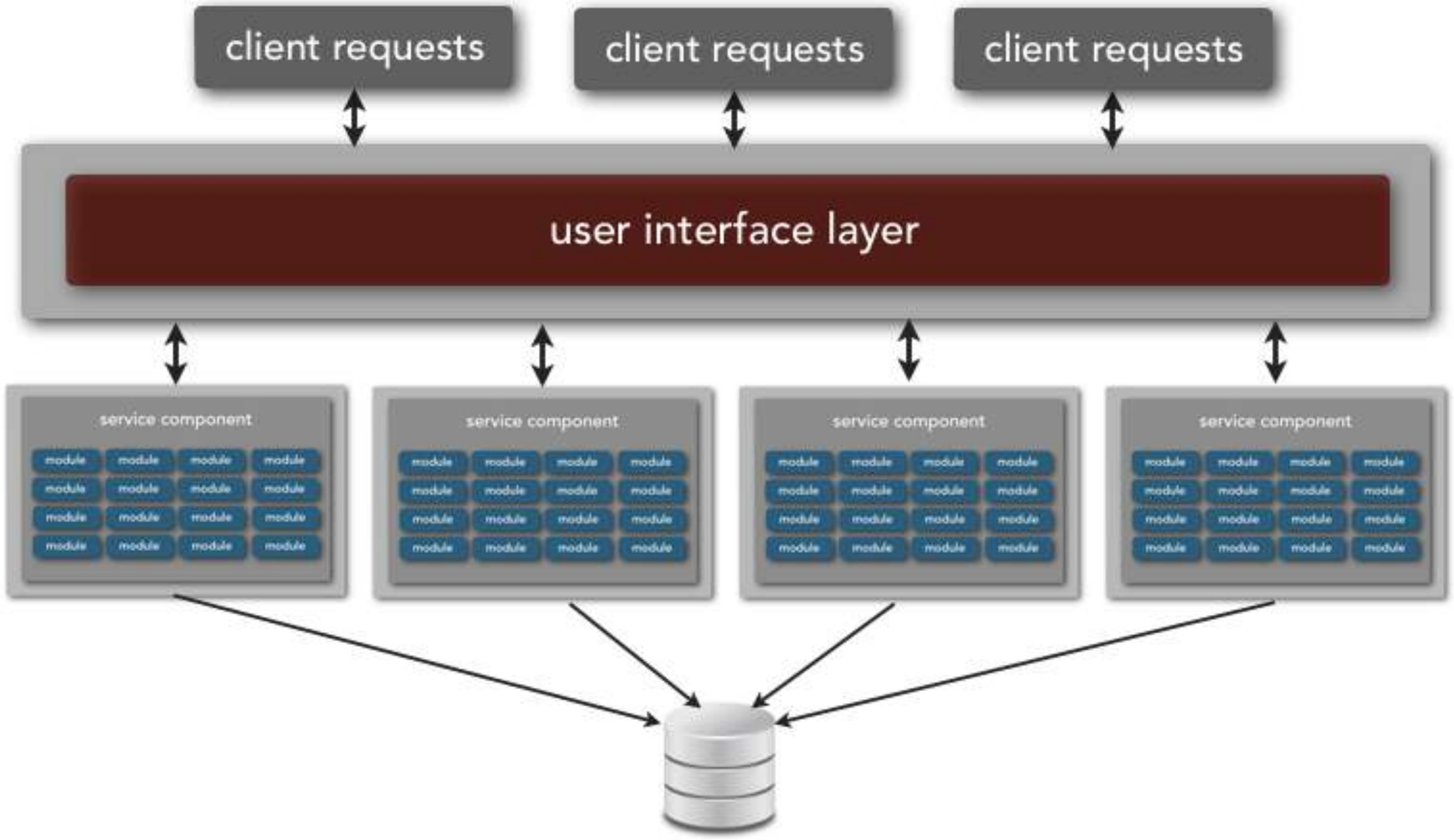


- ✓ modularity
- ✓ agility
- ✓ fault-tolerance
- ✓ scalability
- ✓ testability
- ✓ deployability
- ✓ evolvability

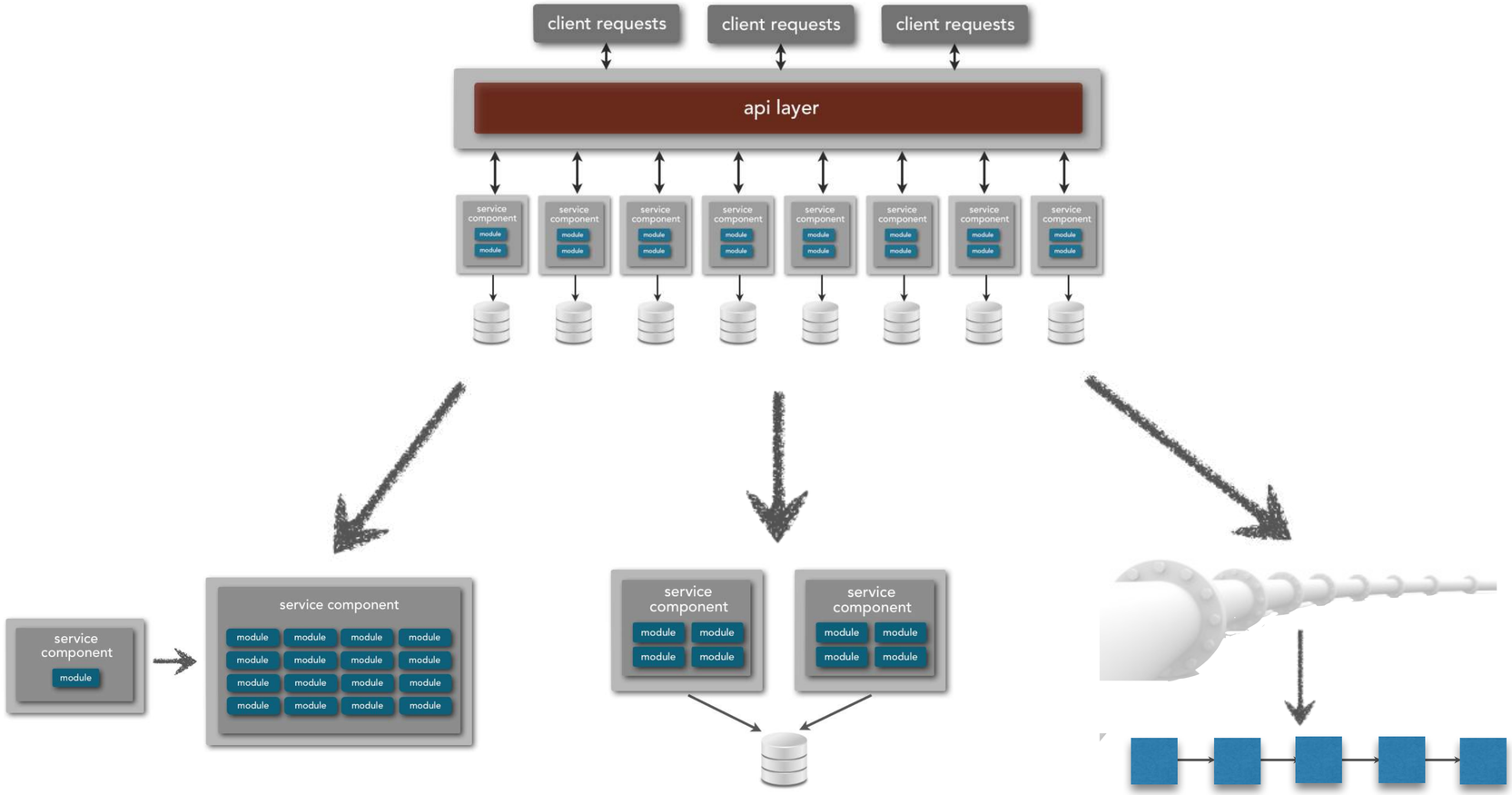
aGEnDA


Quantum = 1
Quanta > 1
architecture styles
Generic
Technical
Layered
Microkernel
Event-driven
Broker
Mediator
ESB-driven SOA
Serverless
Domain
Modular monolith
microservices
Service-based
Domain-specific
Space-based
LMAX
Patterns
Reuse patterns
Canonical
Sidecar/service Mesh
Communication
Orchestration
Sagas
Choreography
Workflow event pattern
Data
Event Sourcing
CQRS
Data replication patterns in microservices

Service-based Architecture



Service-based Architecture



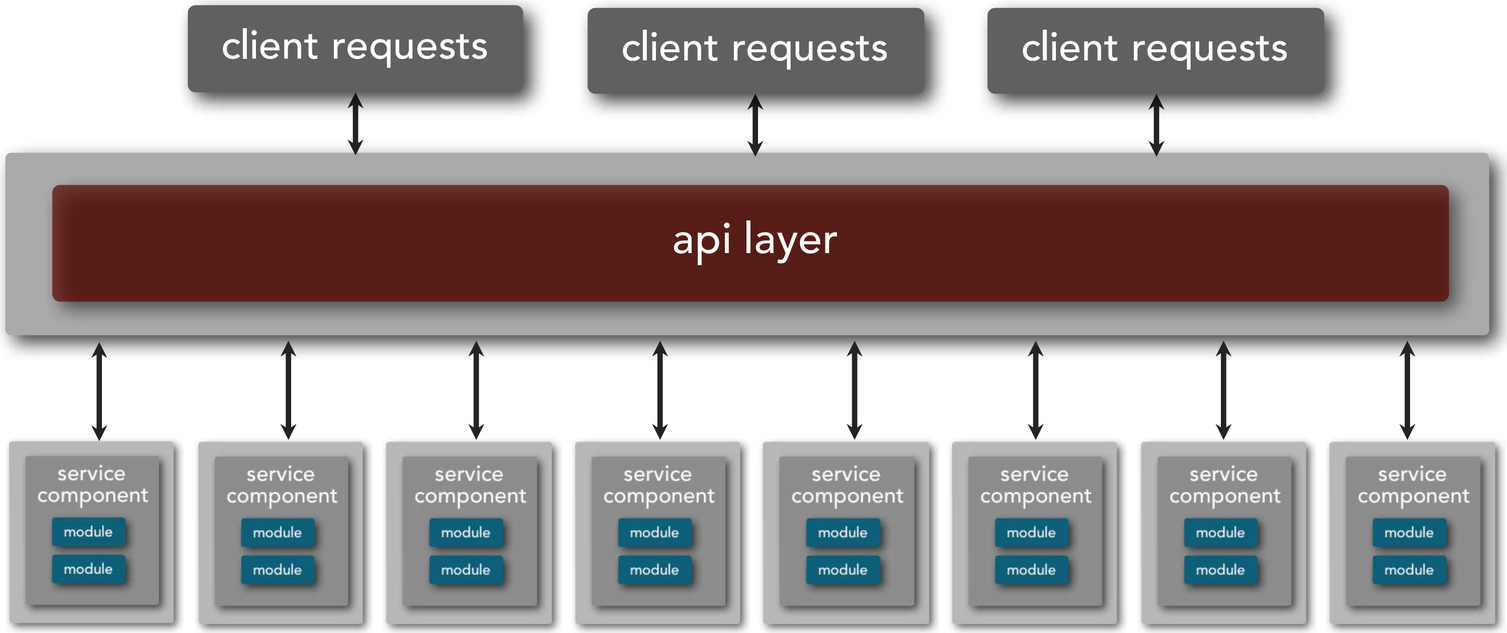
service granularity

database scope

deployment pipeline

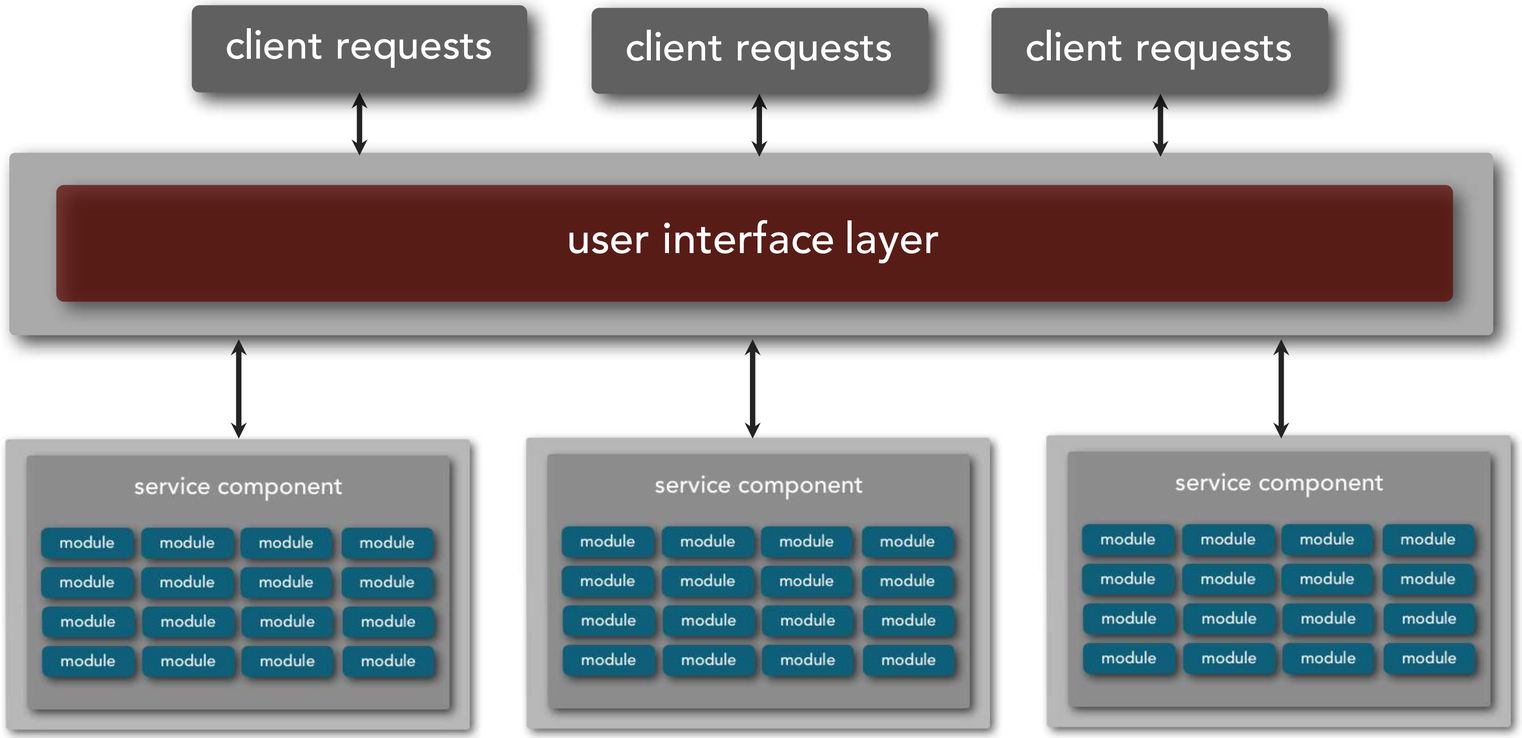
Service-based Architecture

service granularity



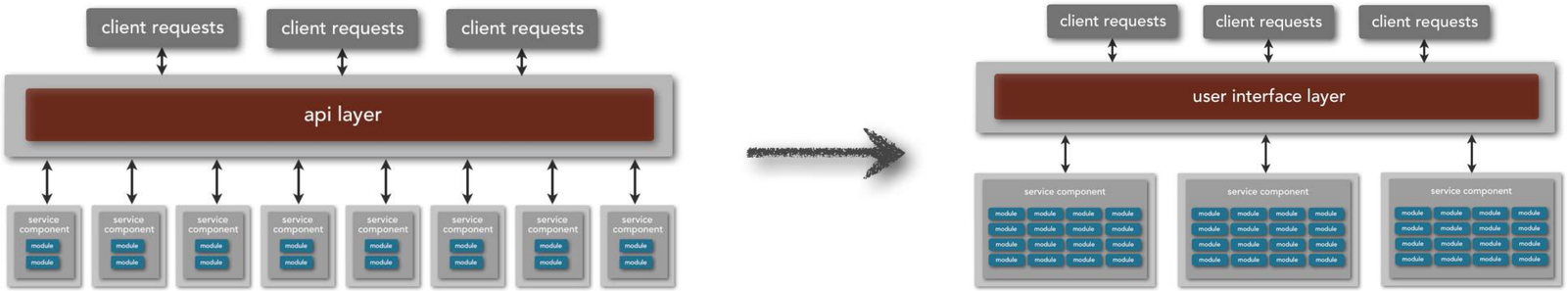
Service-based Architecture

service granularity



Service-based Architecture

service granularity



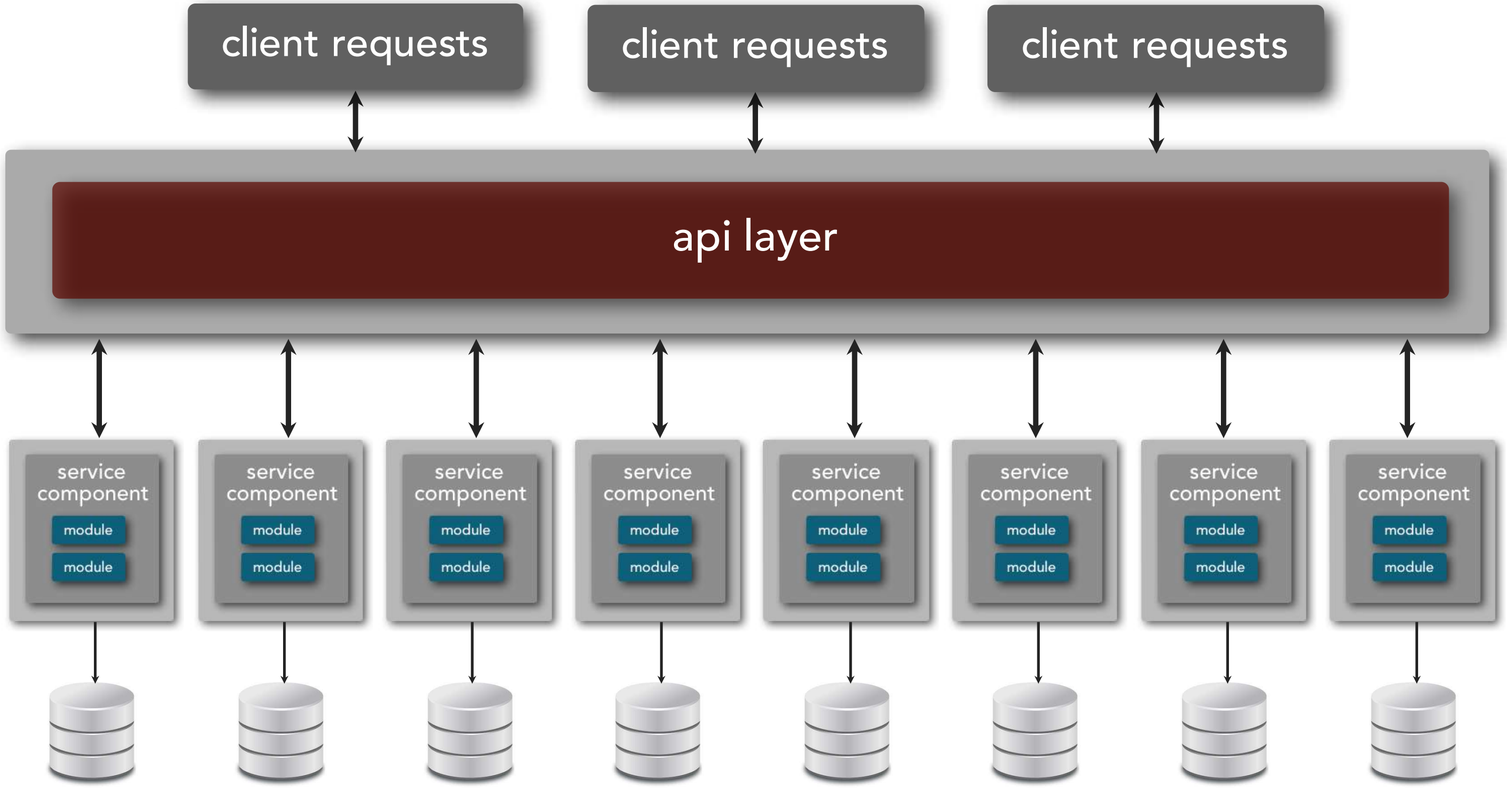
tradeoffs

- performance
- robustness
- domain scope

- service development
- service testing
- deployment pipeline

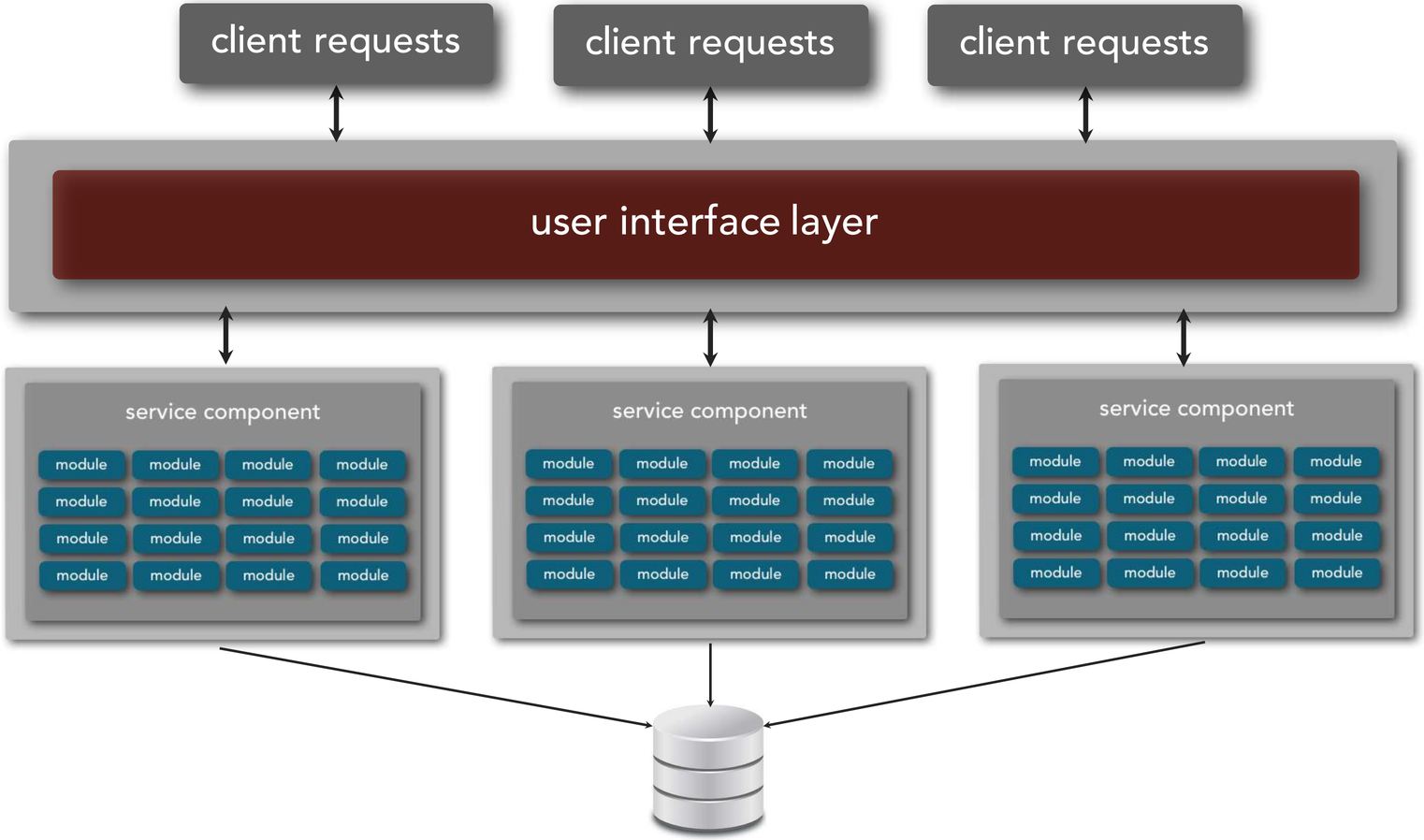
Service-based Architecture

database scope



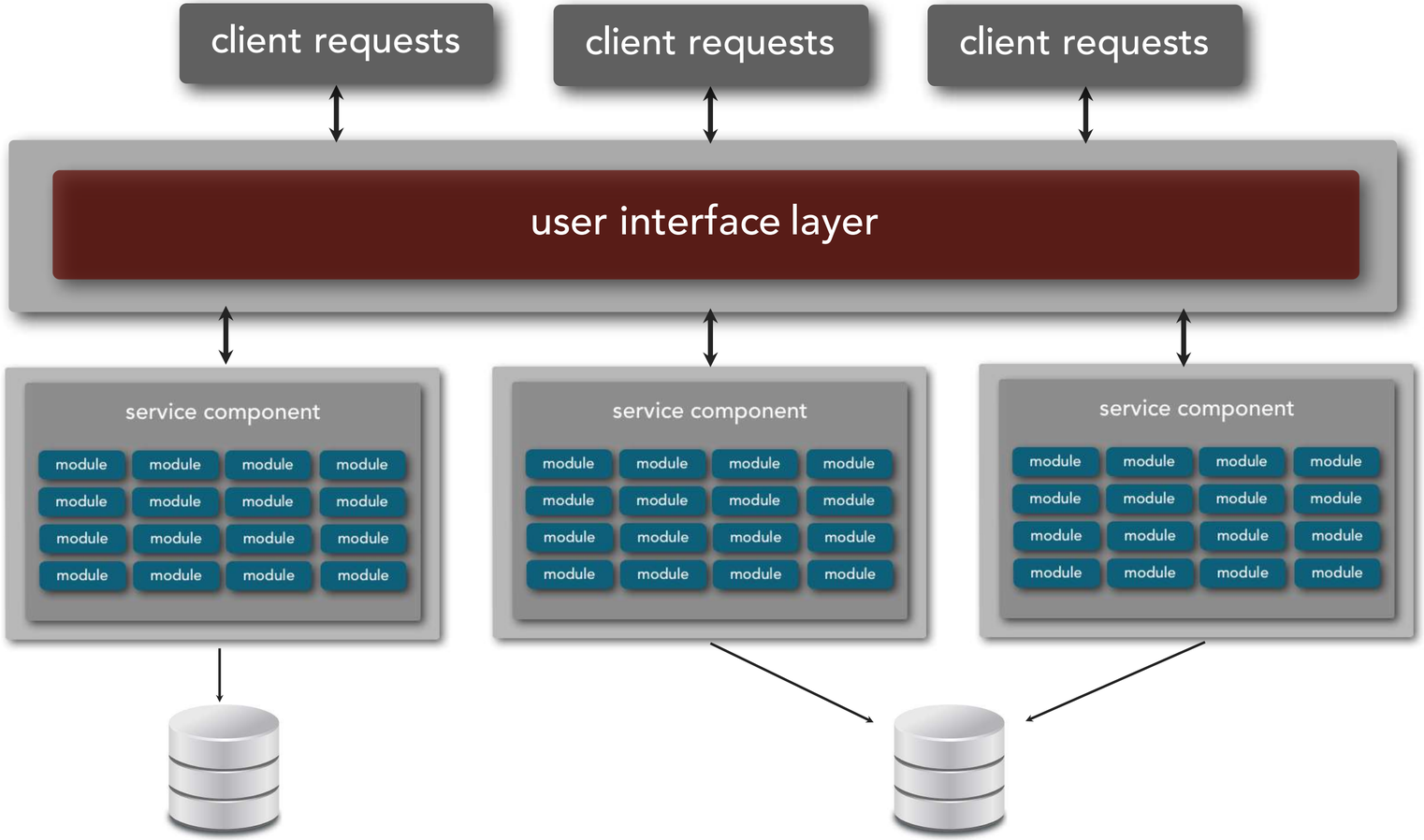
Service-based Architecture

database scope



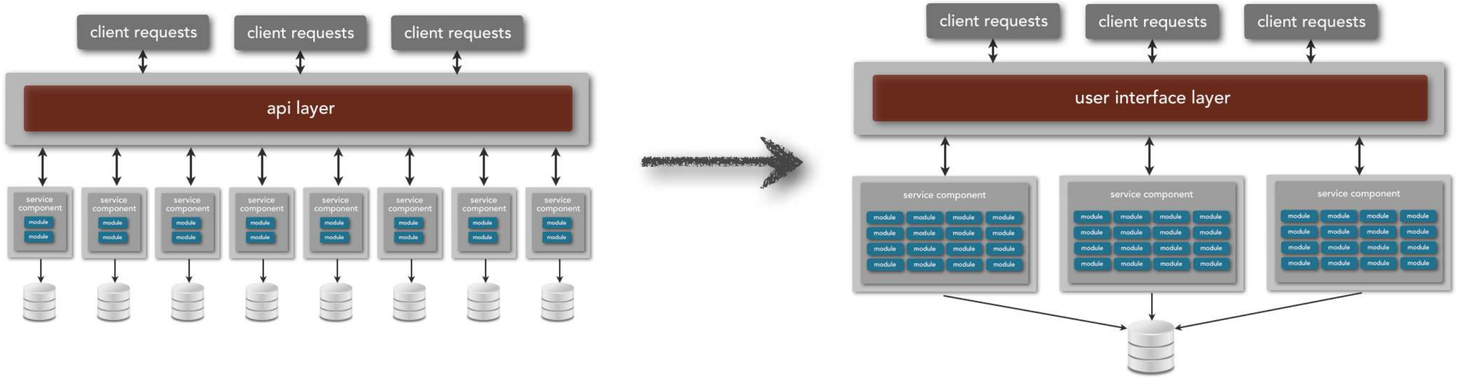
Service-based Architecture

database scope



Service-based Architecture

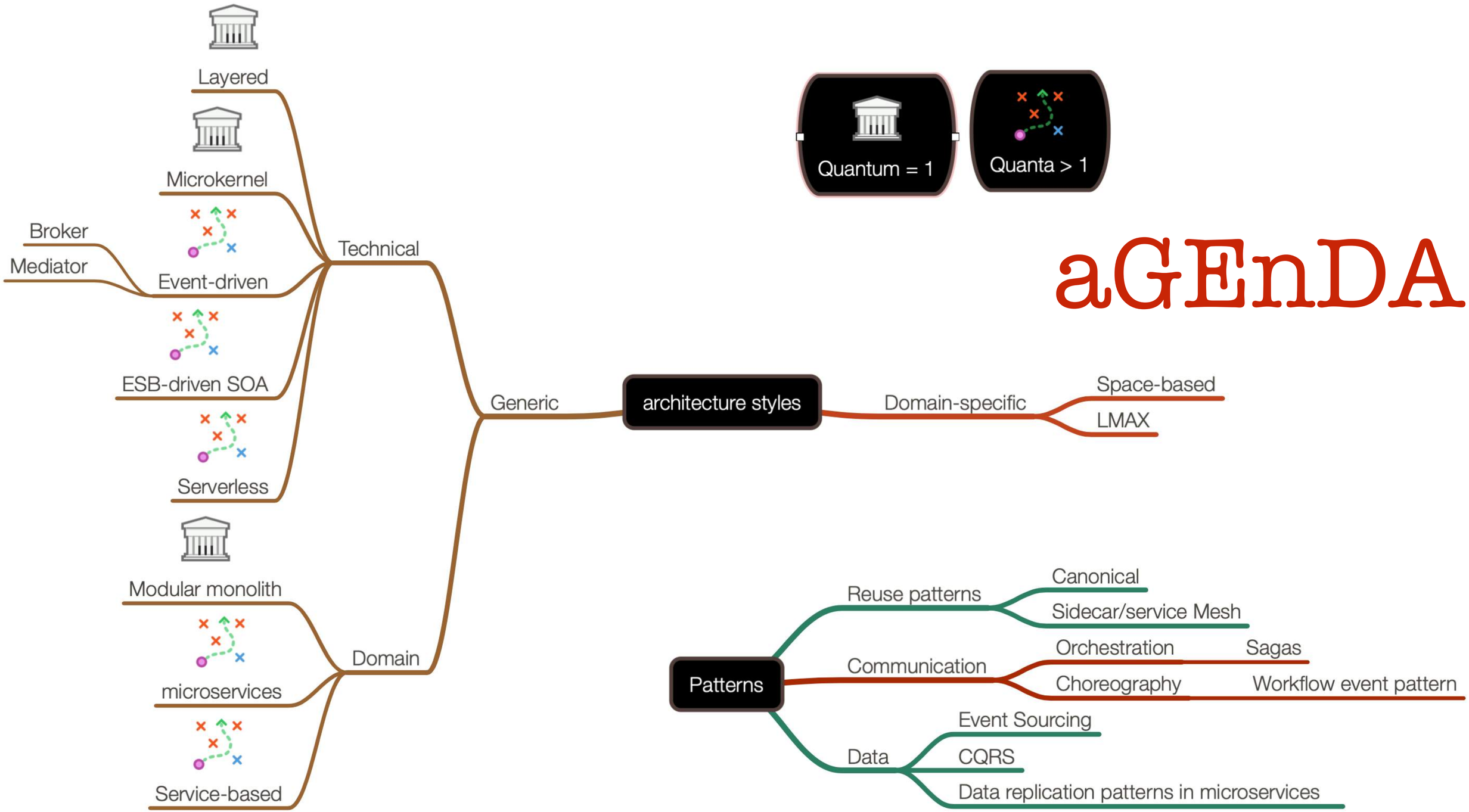
database scope



tradeoffs

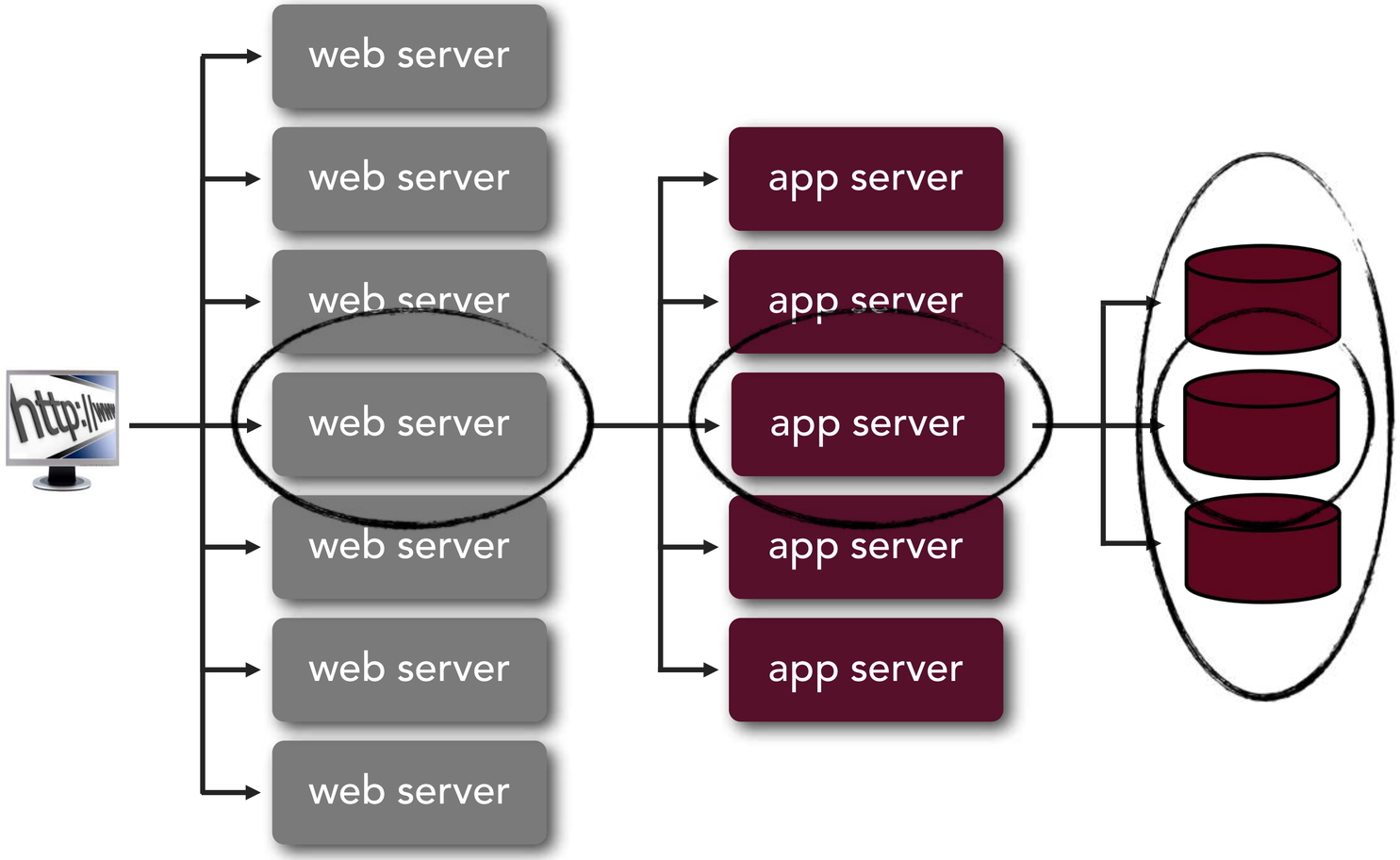
- performance
- feasibility
- service coupling
- schema changes

aGEnDA

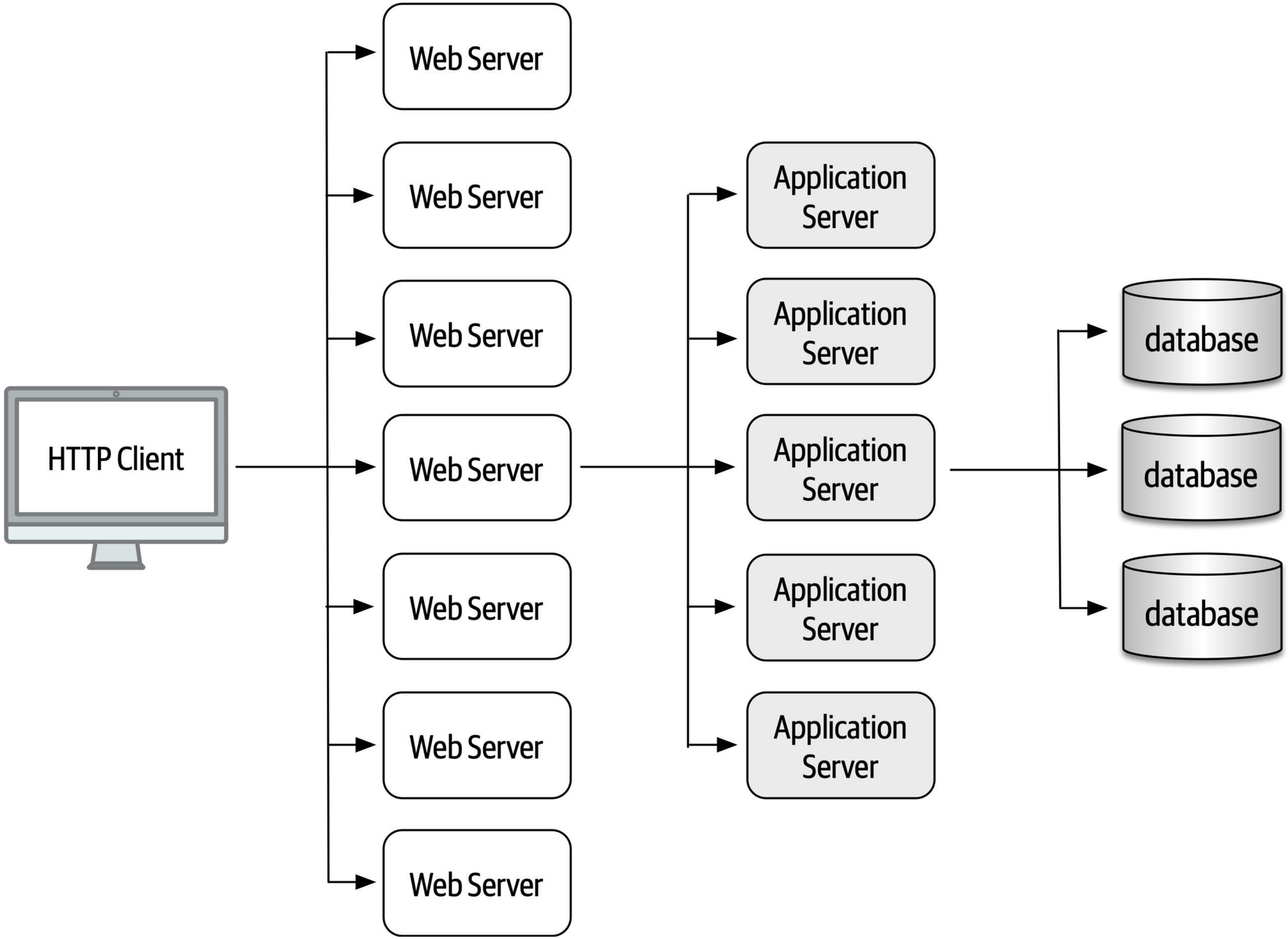

Quantum = 1
Quanta > 1
architecture styles
Generic
Technical
Layered
Microkernel
Event-driven
Broker
Mediator
ESB-driven SOA
Serverless
Domain
Modular monolith
microservices
Service-based
Domain-specific
Space-based
LMAX
Patterns
Reuse patterns
Canonical
Sidecar/service Mesh
Communication
Orchestration
Sagas
Choreography
Workflow event pattern
Data
Event Sourcing
CQRS
Data replication patterns in microservices

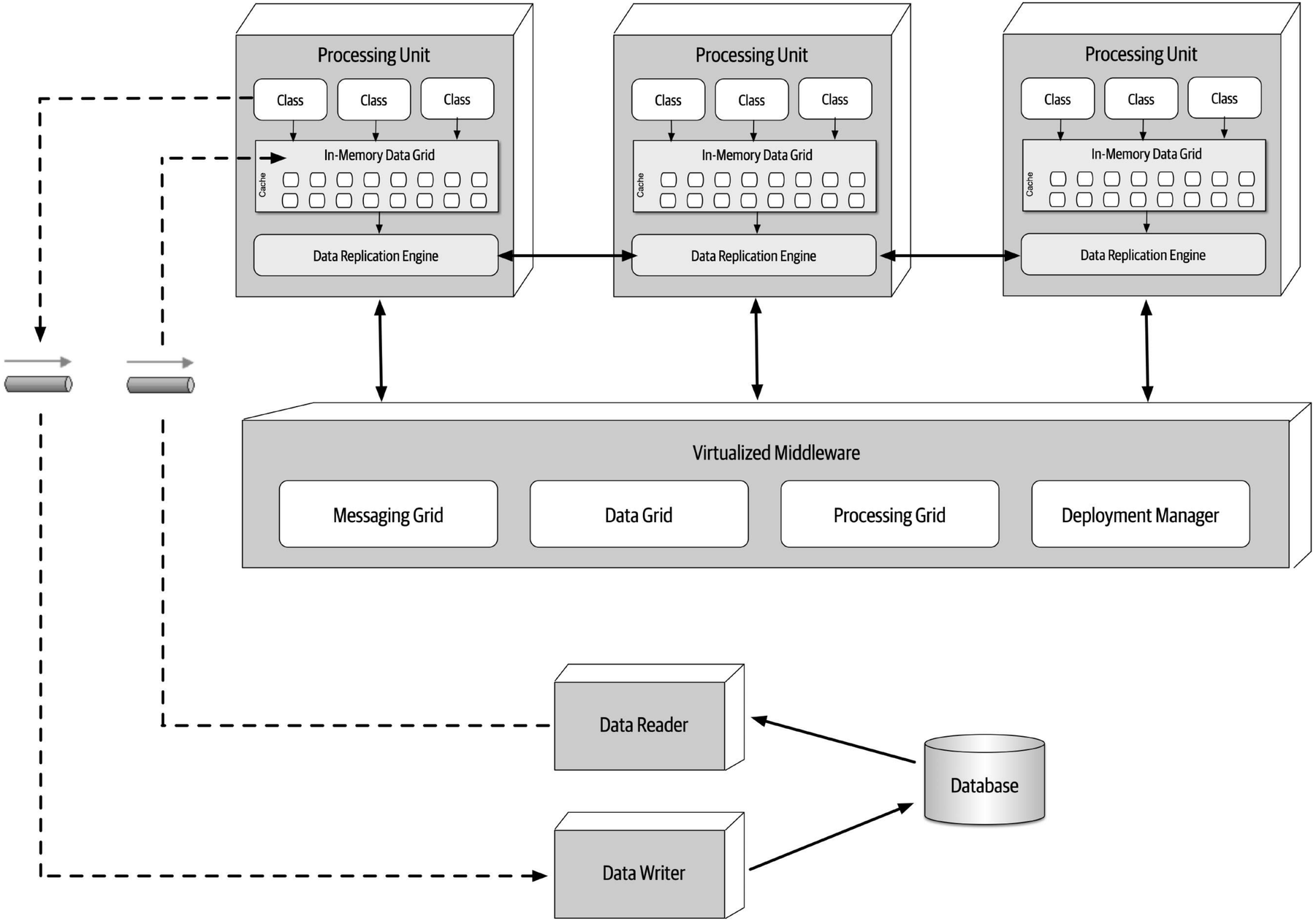
Space-based Architecture

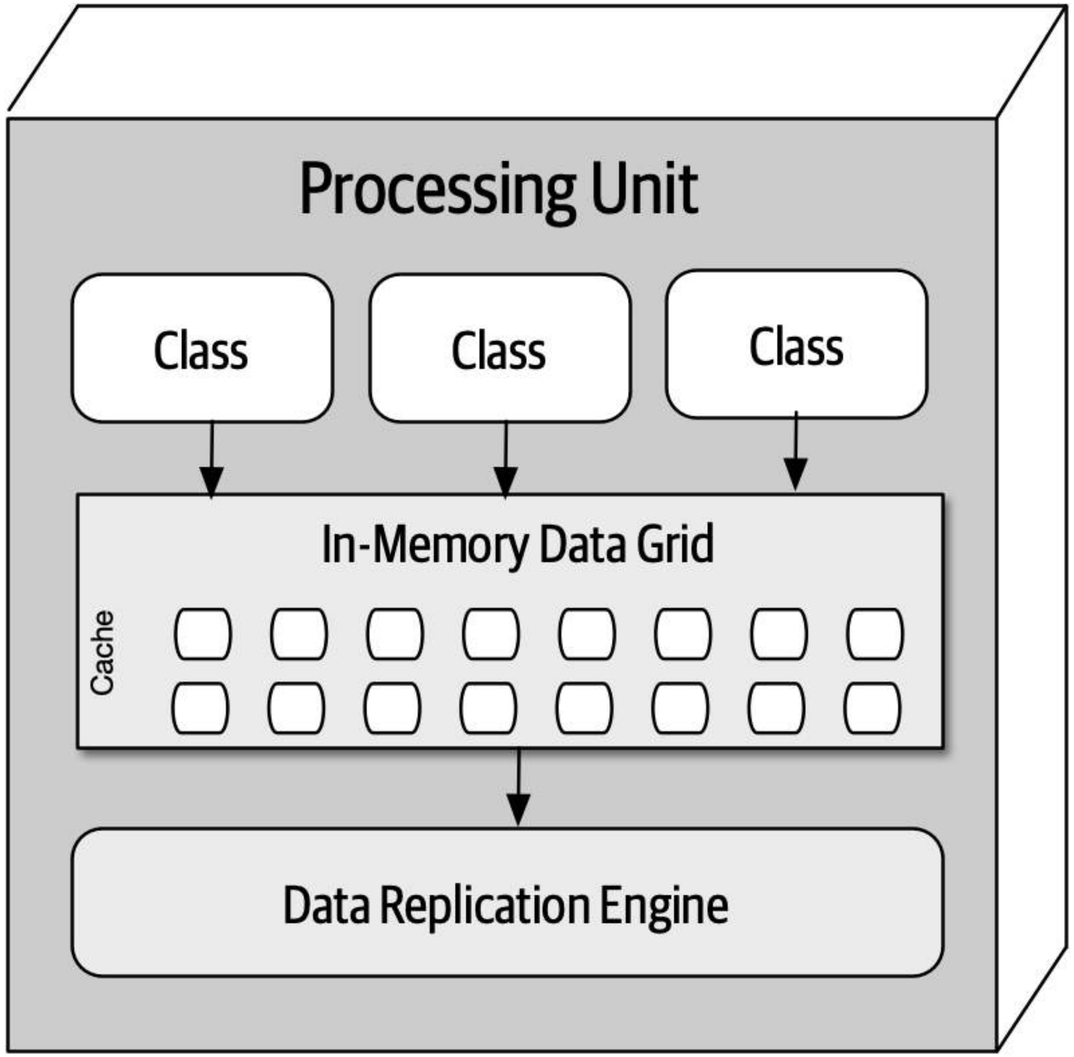
let's talk about scalability for a moment...

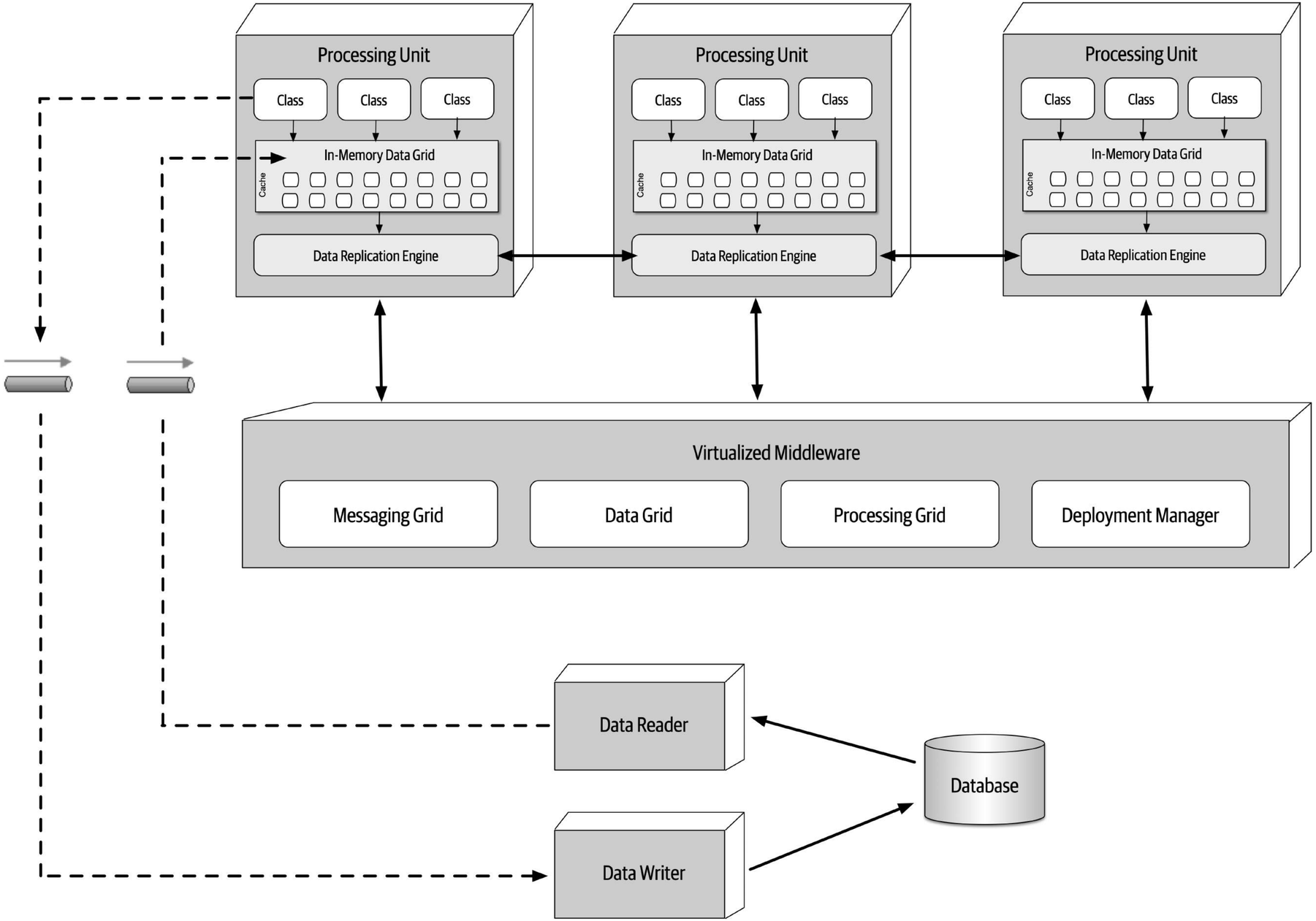


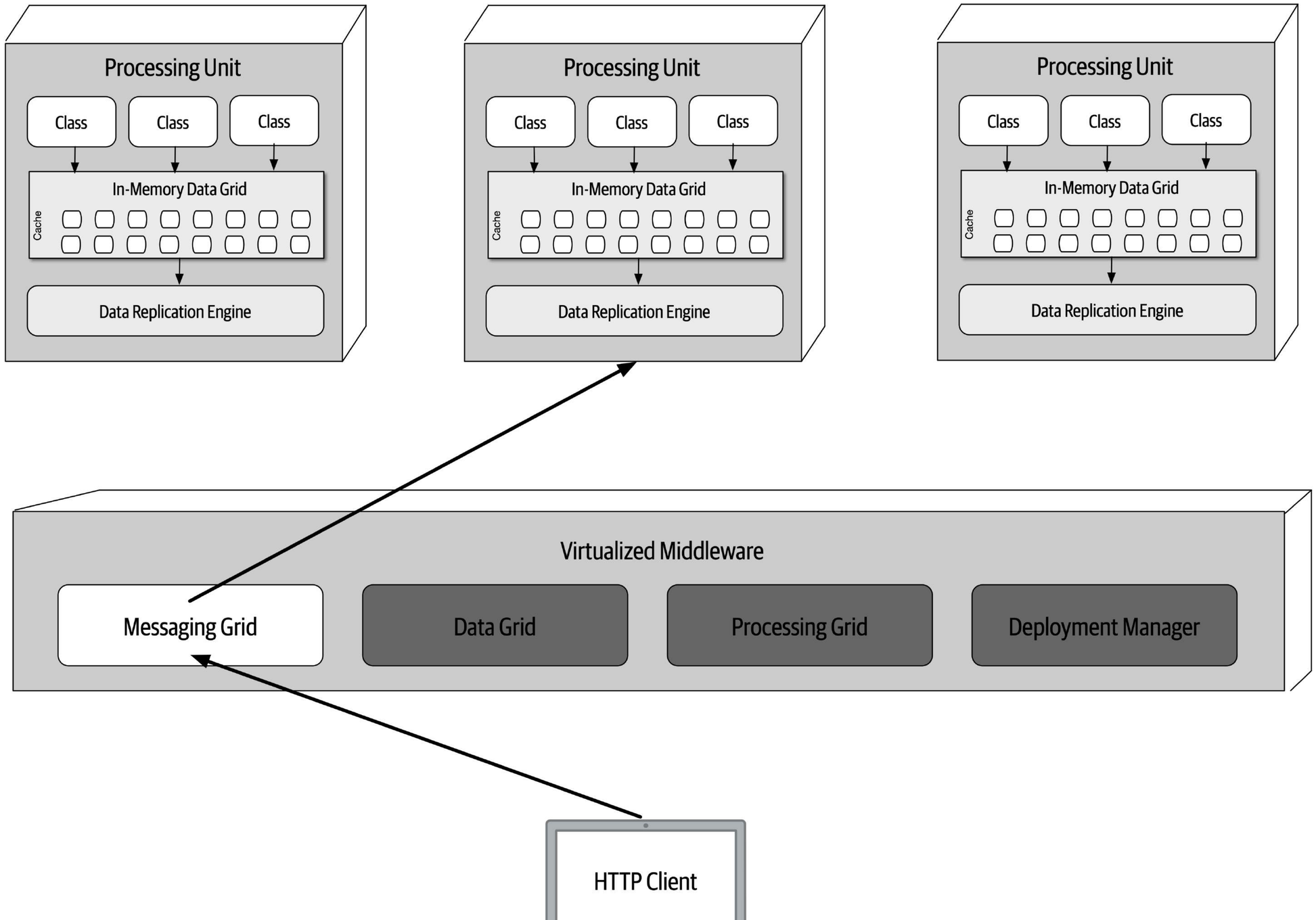
space-based architecture

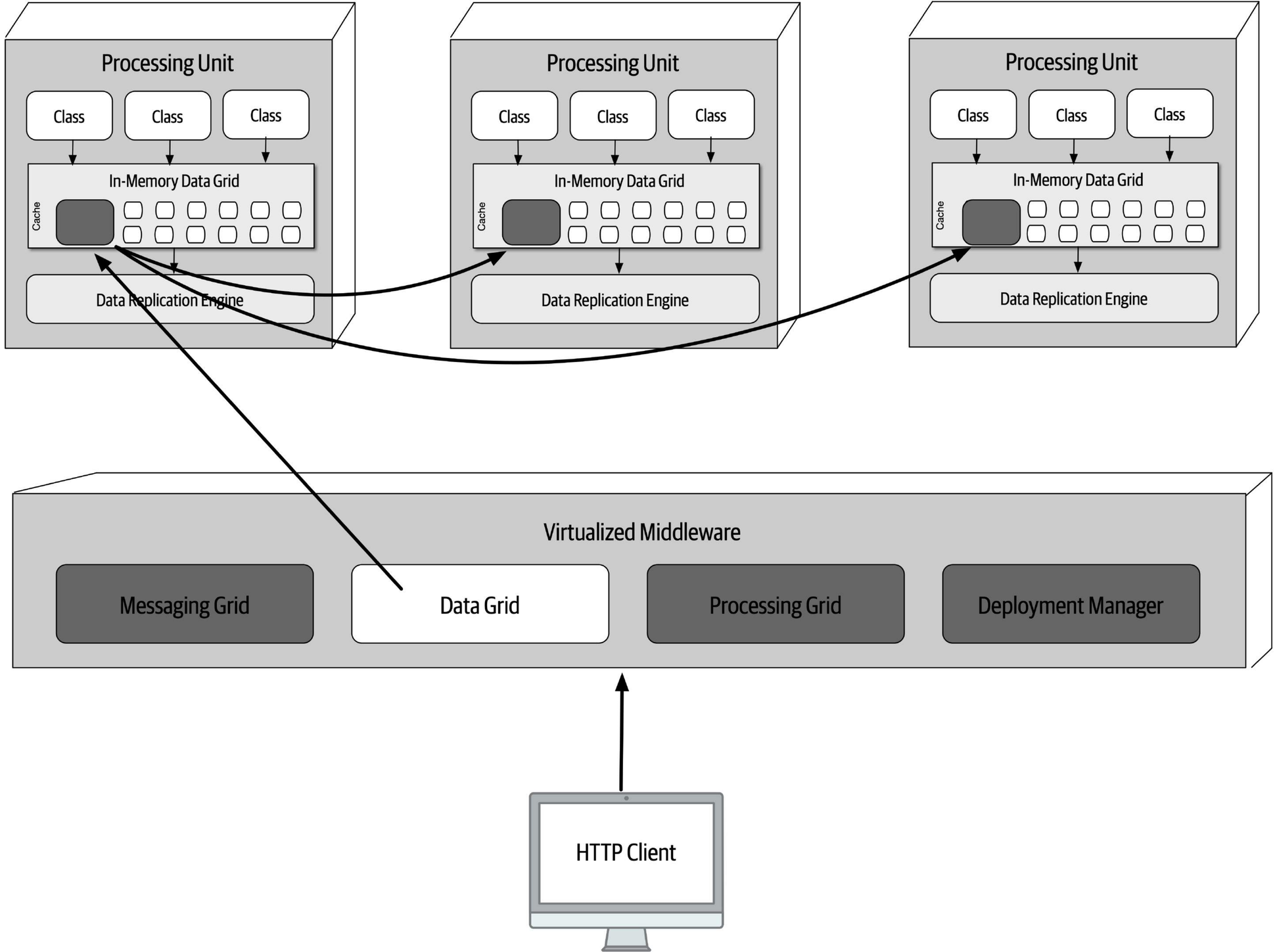


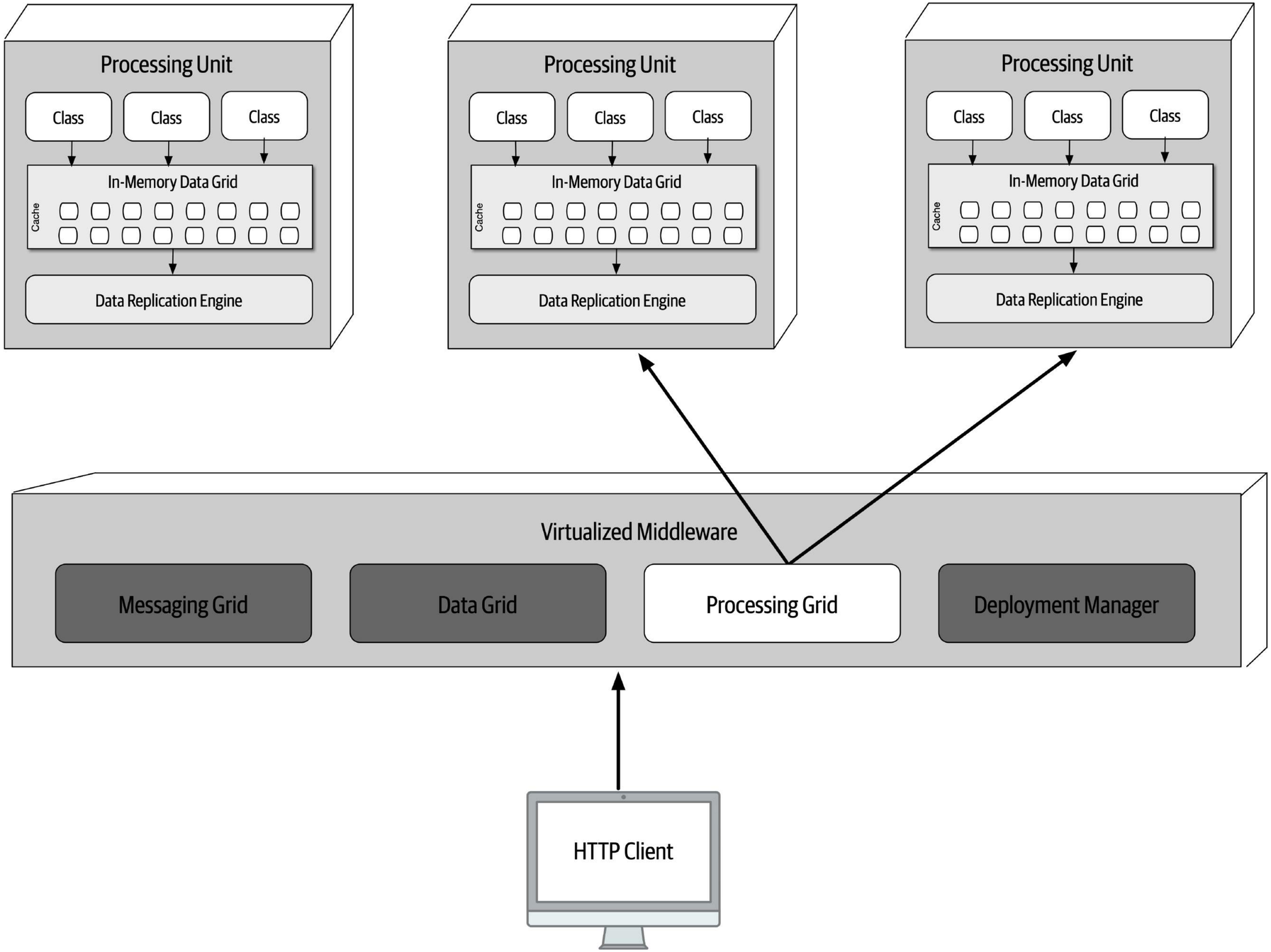

Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Virtualized Middleware
Messaging Grid
Data Grid
Processing Grid
Deployment Manager
Data Reader
Database
Data Writer

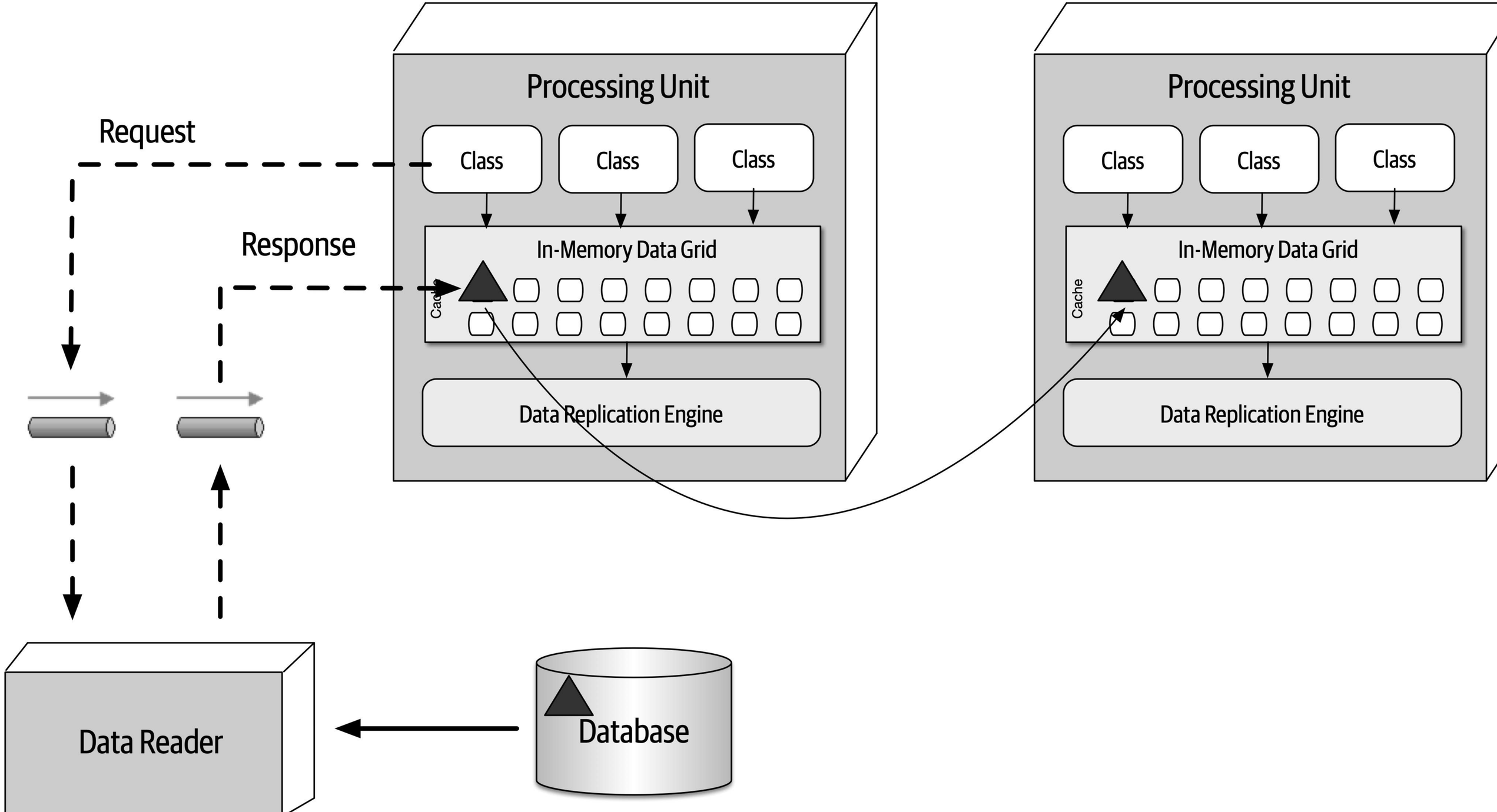

Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine

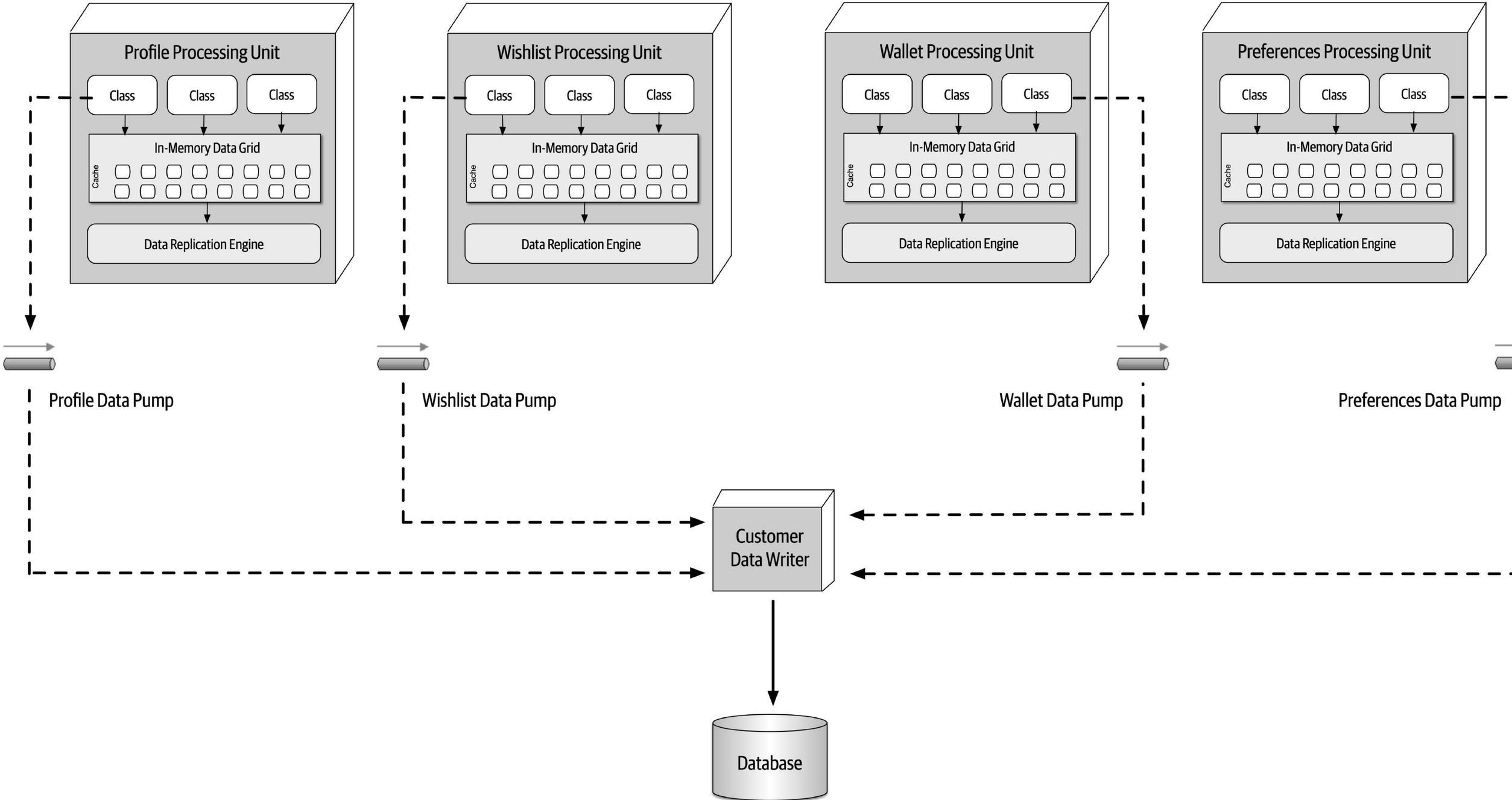

Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Virtualized Middleware
Messaging Grid
Data Grid
Processing Grid
Deployment Manager
Data Reader
Database
Data Writer

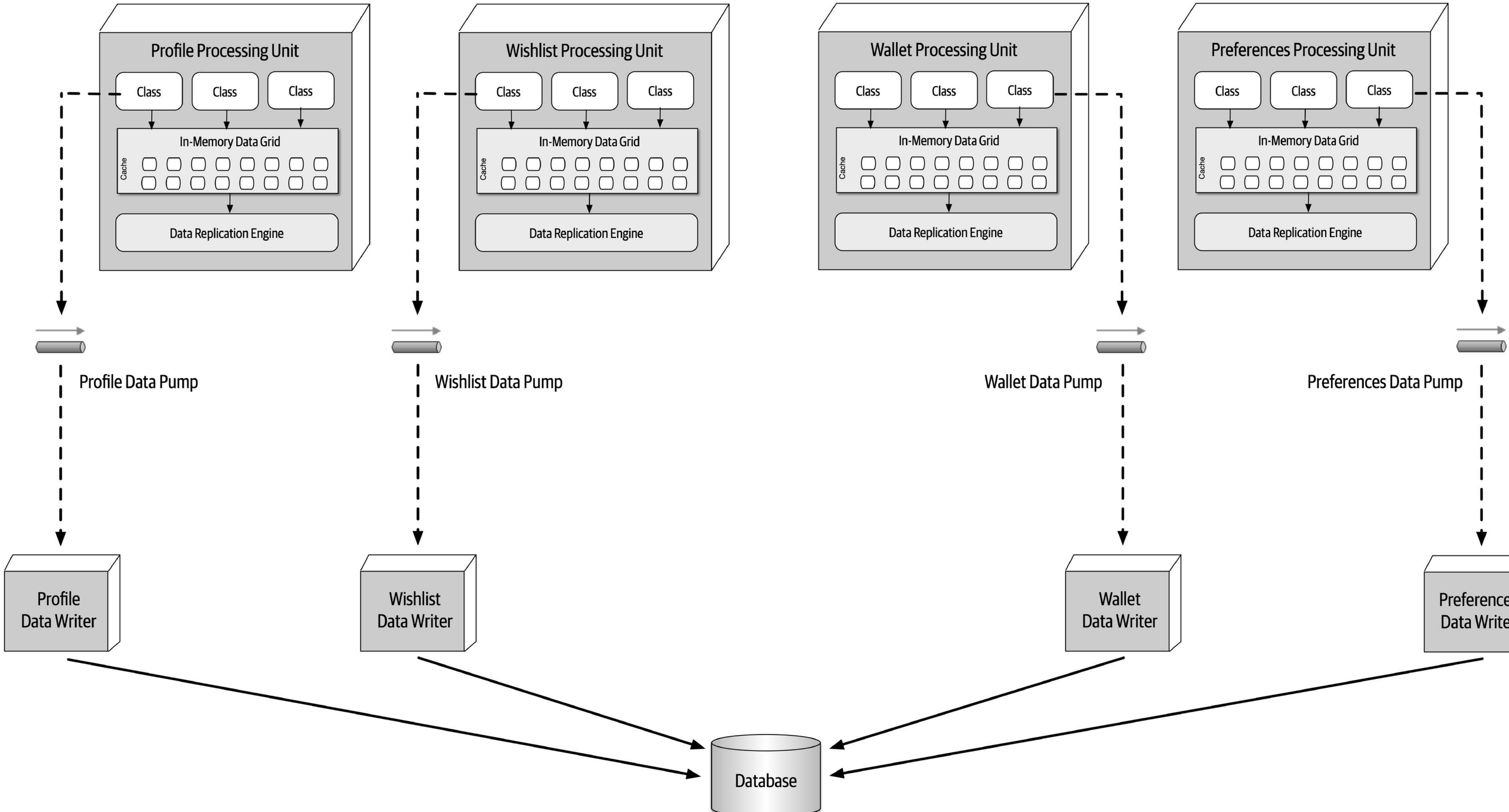

Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Virtualized Middleware
Messaging Grid
Data Grid
Processing Grid
Deployment Manager
HTTP Client

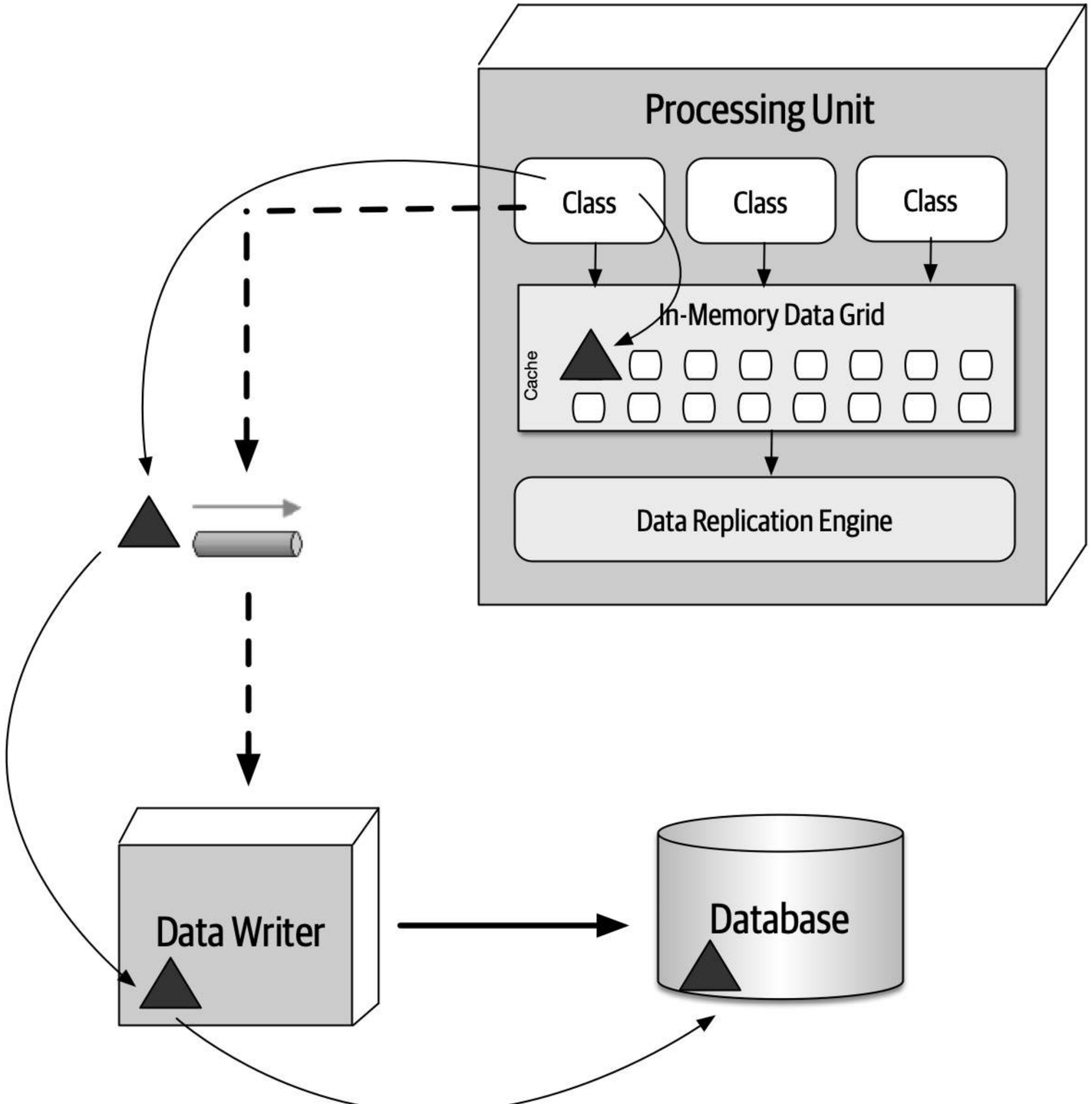

Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Virtualized Middleware
Messaging Grid
Data Grid
Processing Grid
Deployment Manager
HTTP Client

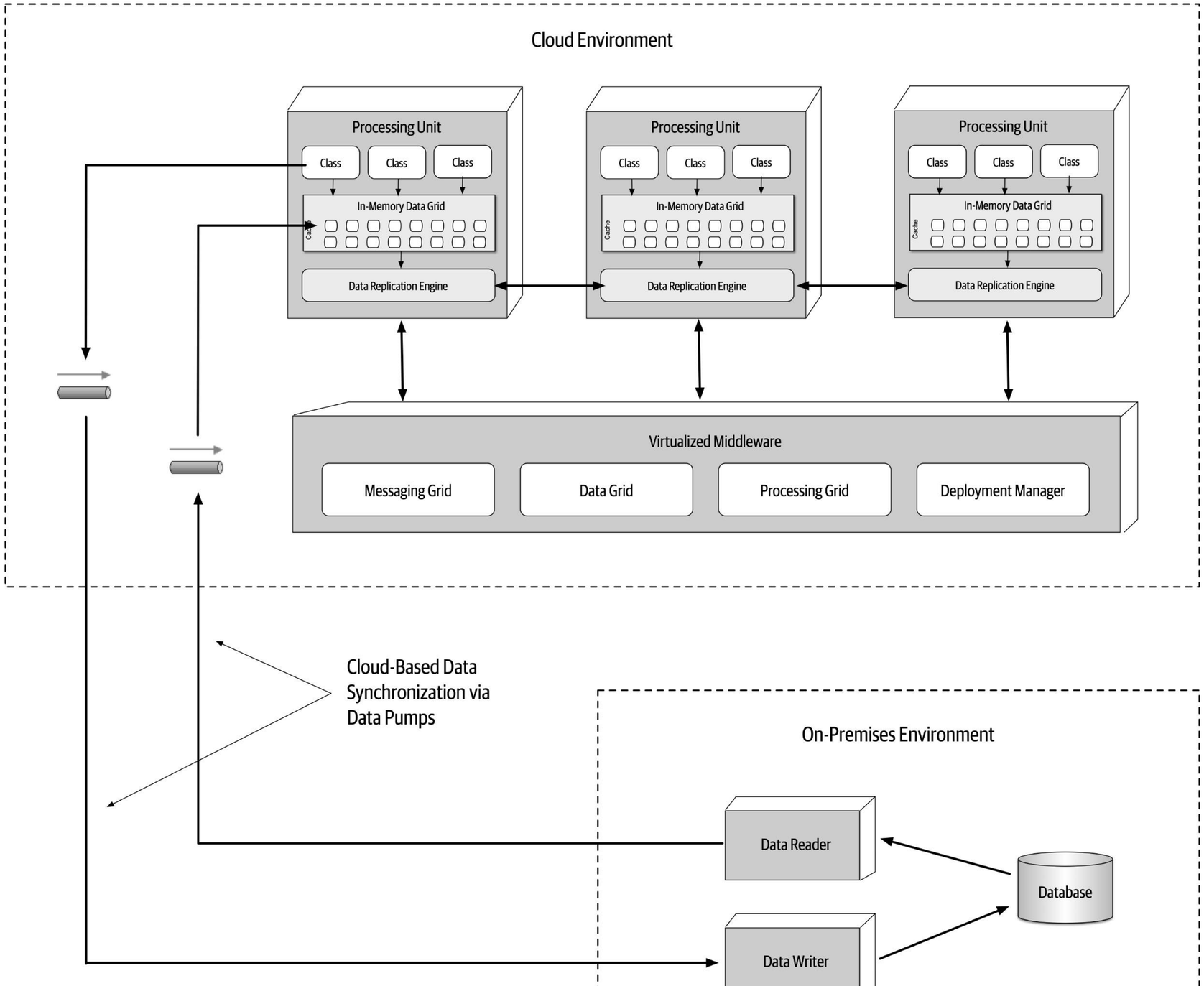

Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Virtualized Middleware
Messaging Grid
Data Grid
Processing Grid
Deployment Manager
HTTP Client

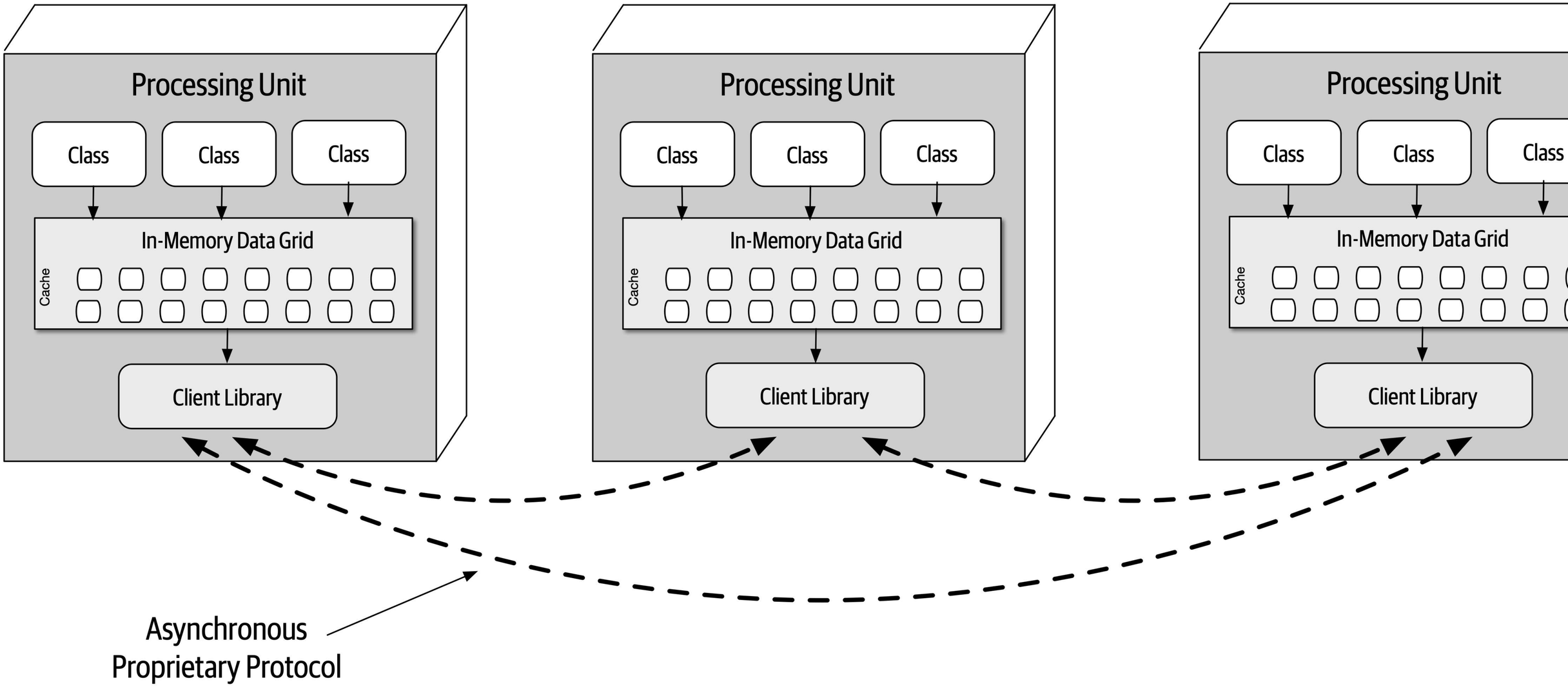

Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Request
Response
Data Reader
Database

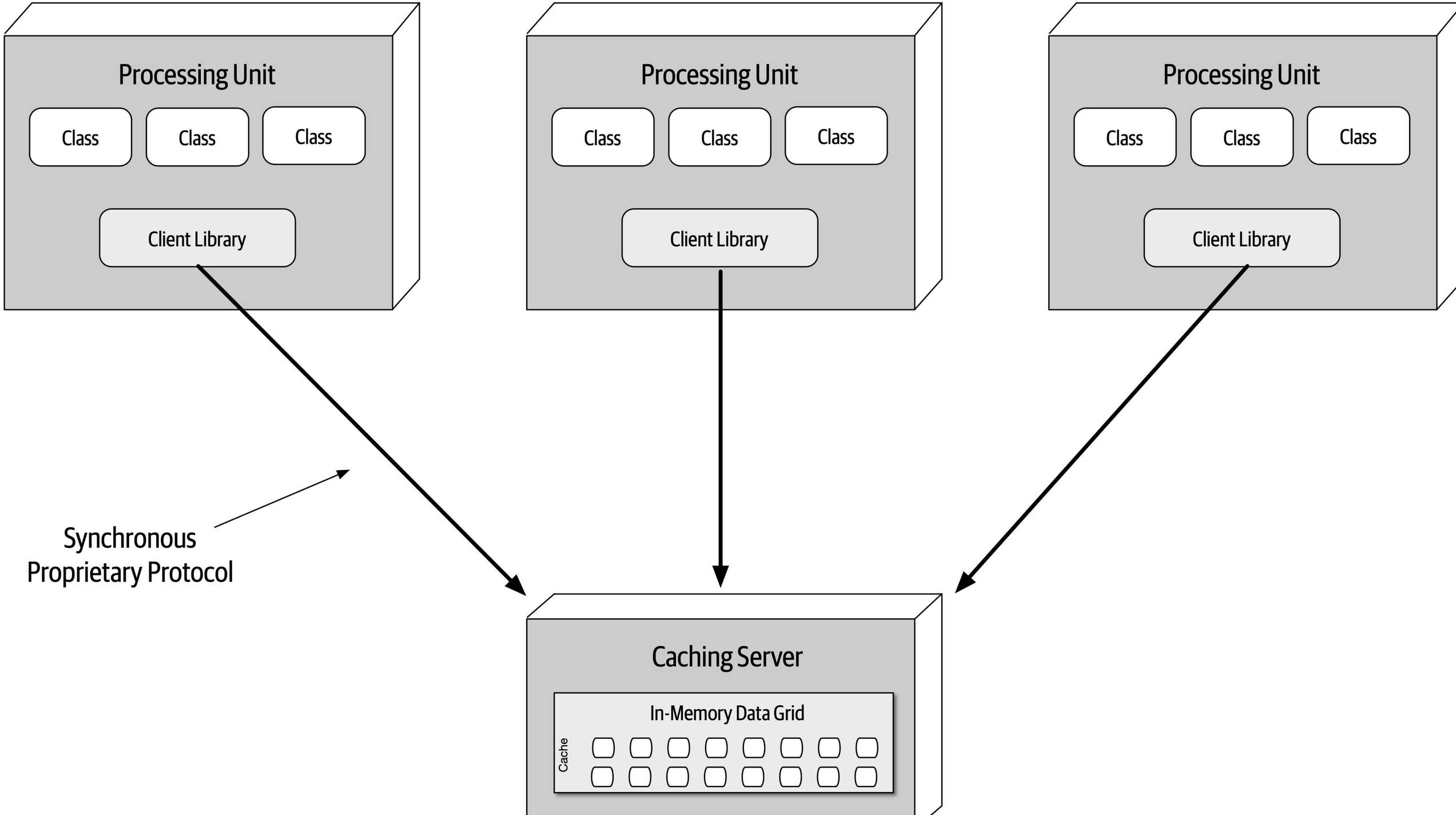

Profile Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Wishlist Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Wallet Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Preferences Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Profile Data Pump
Wishlist Data Pump
Wallet Data Pump
Preferences Data Pump
Customer Data Writer
Database

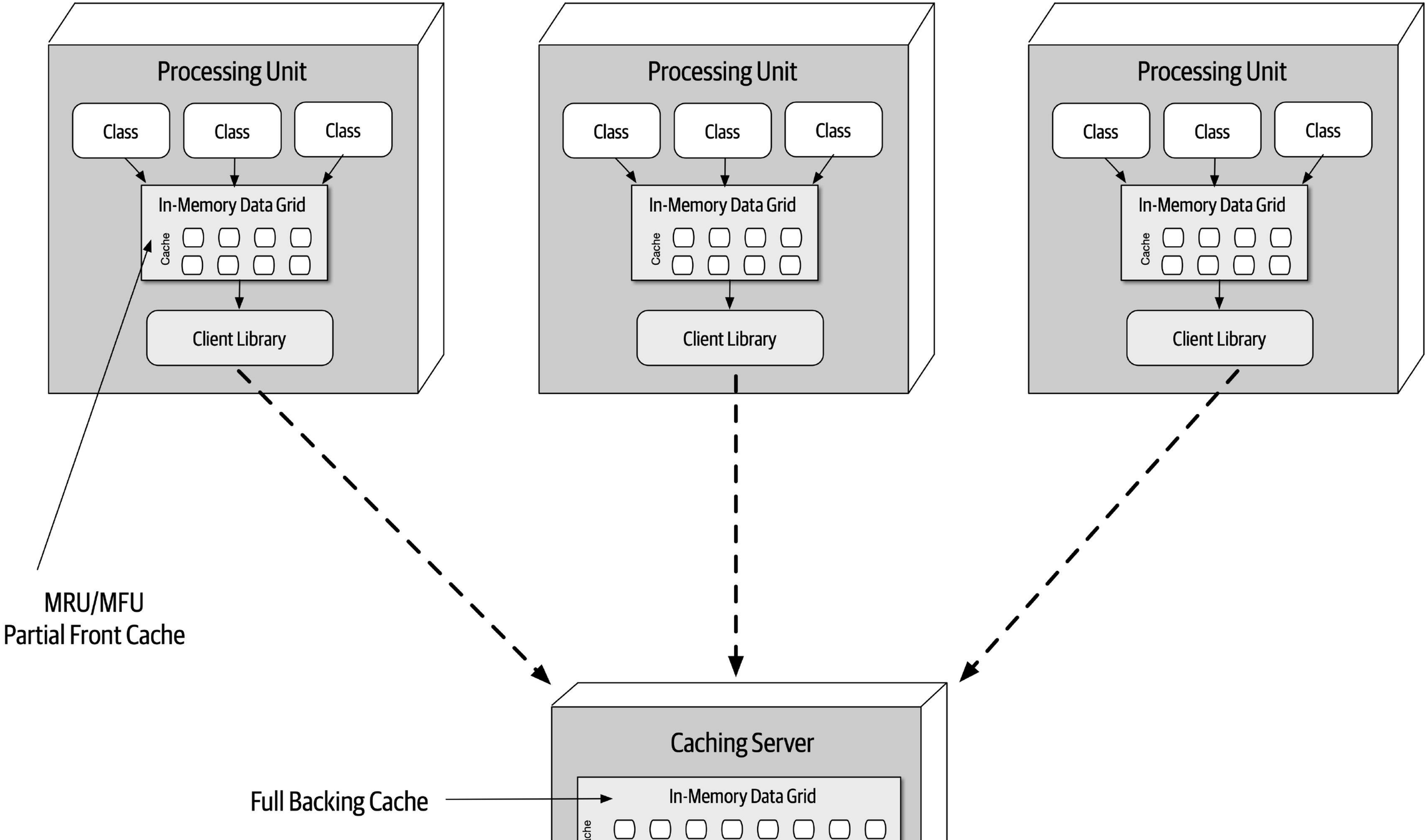

Profile Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Wishlist Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Wallet Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Preferences Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Profile Data Pump
Wishlist Data Pump
Wallet Data Pump
Preferences Data Pump
Profile Data Writer
Wishlist Data Writer
Wallet Data Writer
Preference Data Write
Database


Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Data Writer
Database


Cloud Environment
Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Data Replication Engine
Virtualized Middleware
Messaging Grid
Data Grid
Processing Grid
Deployment Manager
Cloud-Based Data Synchronization via Data Pumps
On-Premises Environment
Data Reader
Database
Data Writer


Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Client Library
Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Client Library
Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Client Library
Asynchronous
Proprietary Protocol

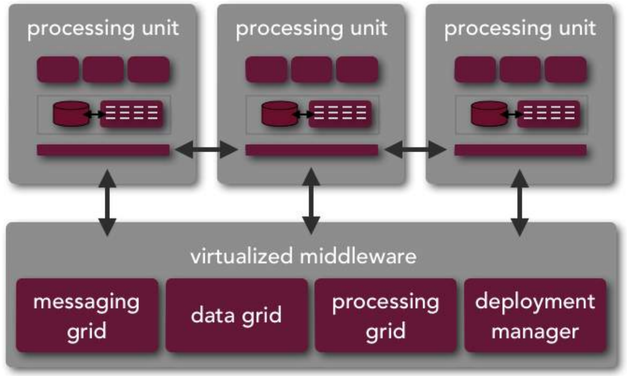

Processing Unit
Class
Class
Class
Client Library
Processing Unit
Class
Class
Class
Client Library
Processing Unit
Class
Class
Class
Client Library
Synchronous
Proprietary Protocol
Caching Server
In-Memory Data Grid
Cache


Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Client Library
Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Client Library
Processing Unit
Class
Class
Class
In-Memory Data Grid
Cache
Client Library
MRU/MFU
Partial Front Cache
Caching Server
Full Backing Cache
In-Memory Data Grid
Cache

Space-based Architecture

product implementations

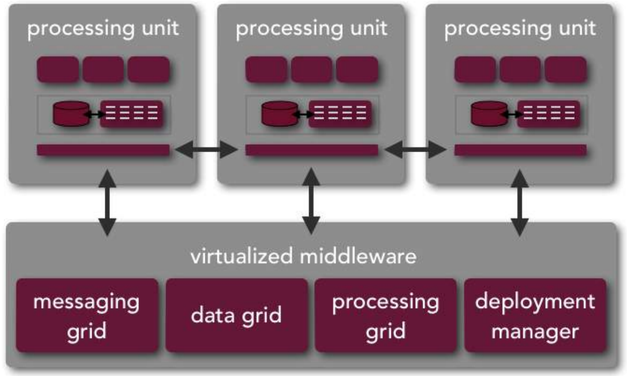
- javaspaces
- gigaspaces
- ibm object grid
- gemfire
- ncache
- oracle coherence



Space-based Architecture

it's all about variable scalability...

good for applications that have variable load or inconsistent peak times

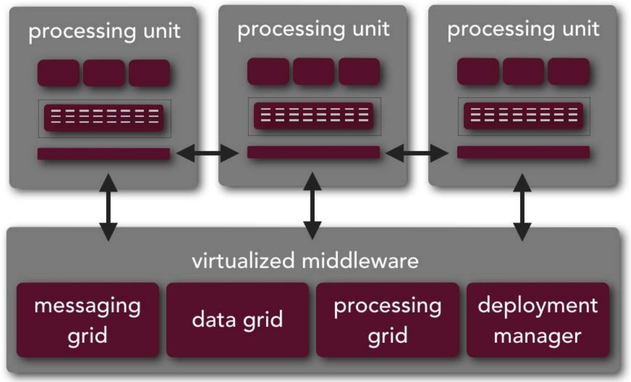


not a good fit for traditional large-scale relational database systems

relatively complex and expensive pattern to implement

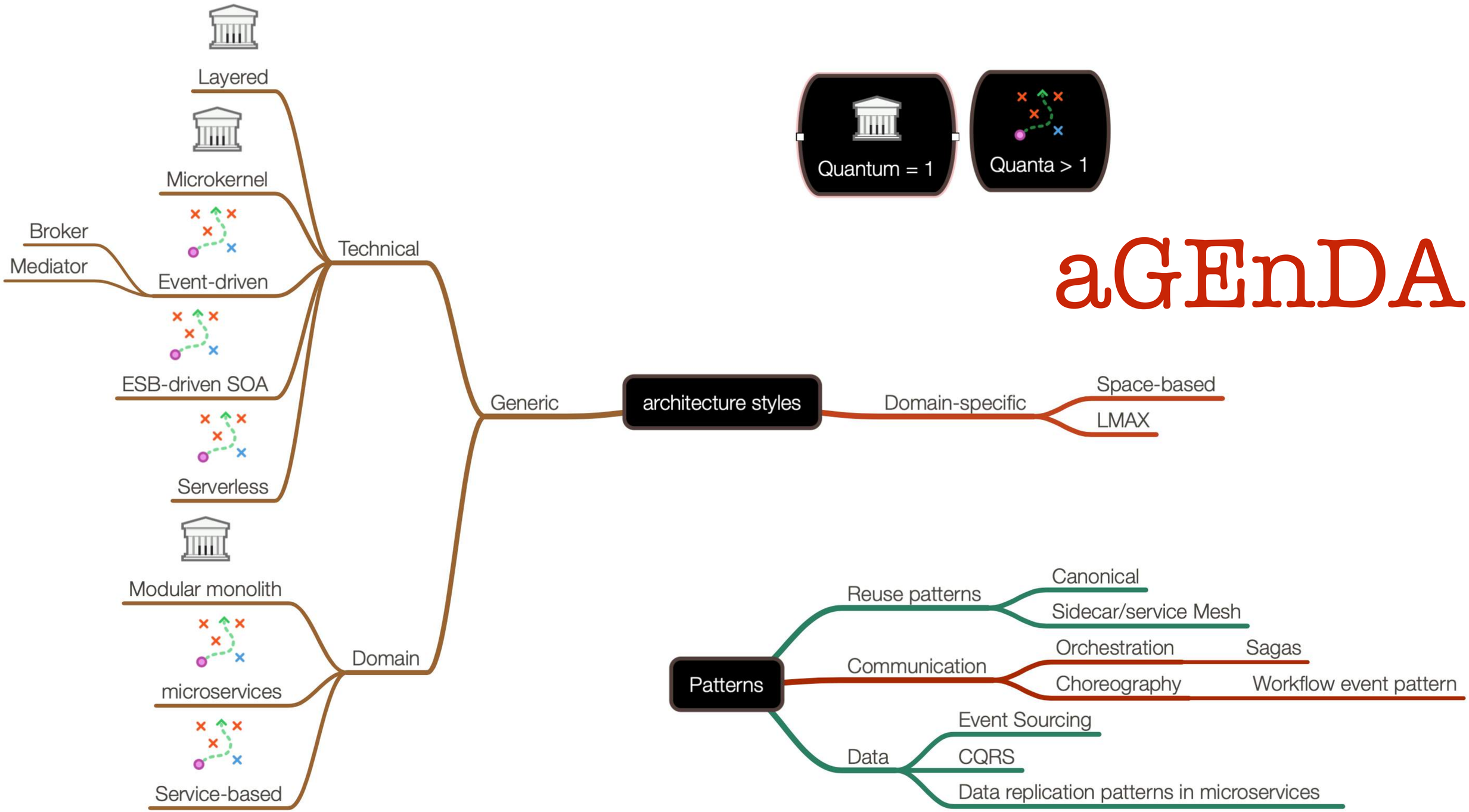
Space-based Architecture

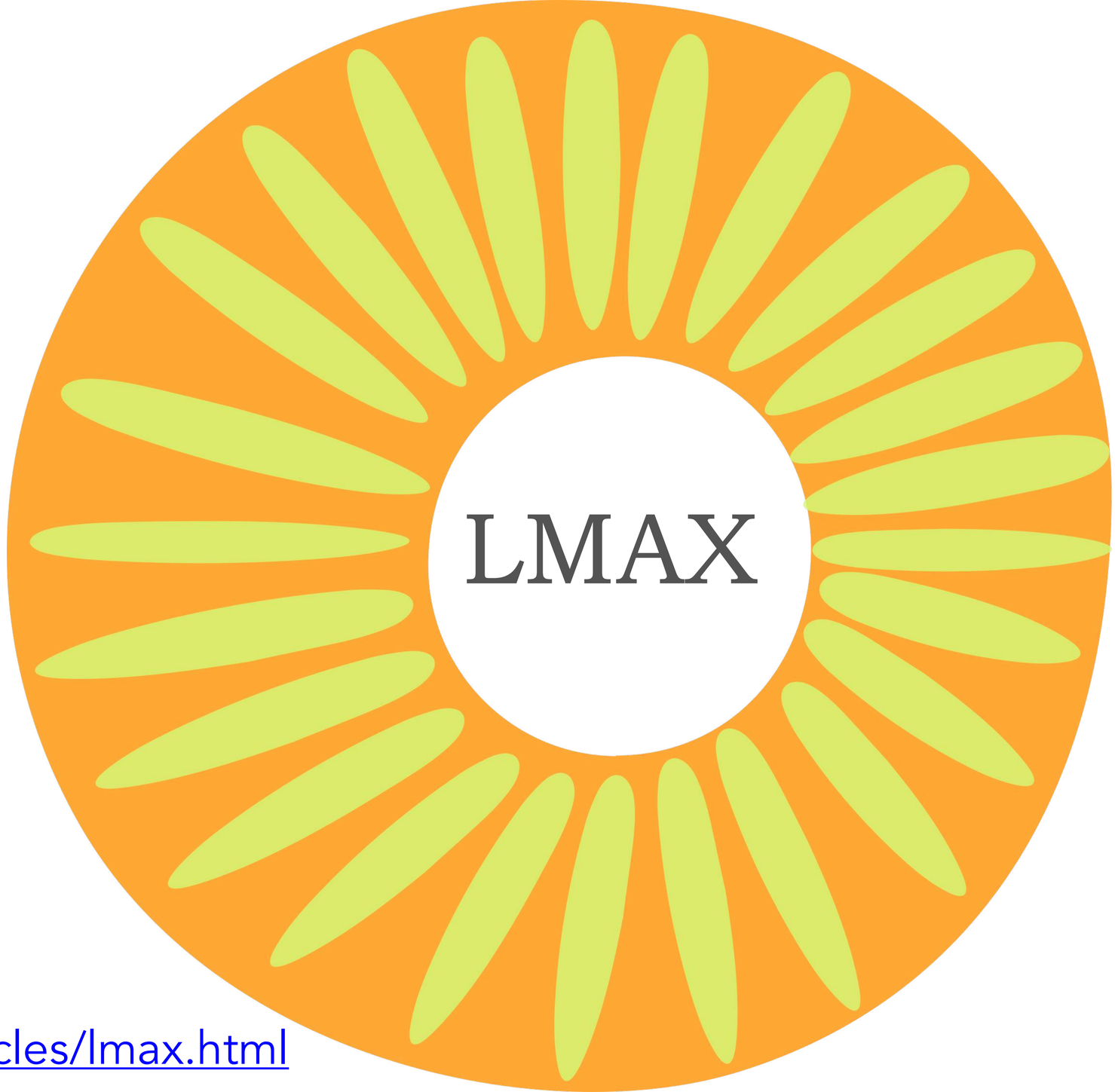
drivers



- ✓ scalability
- ✓ elasticity
- ✓ performance

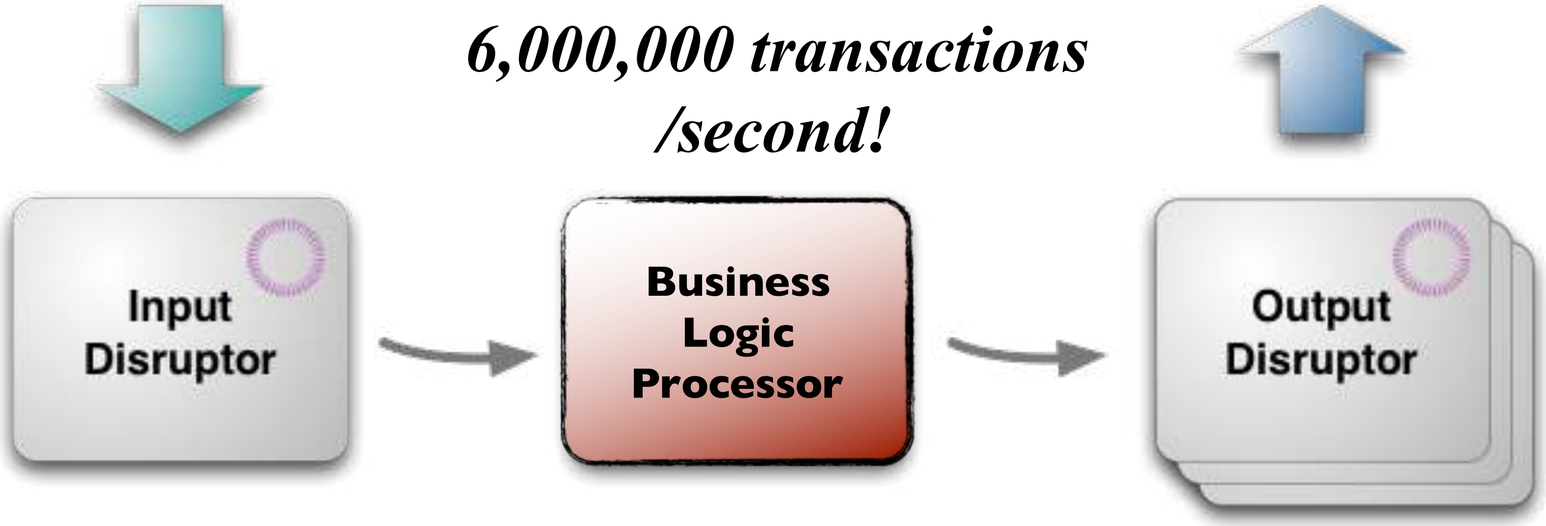
aGEnDA


Quantum = 1
Quanta > 1
architecture styles
Generic
Technical
Layered
Microkernel
Event-driven
Broker
Mediator
ESB-driven SOA
Serverless
Domain
Modular monolith
microservices
Service-based
Domain-specific
Space-based
LMAX
Patterns
Reuse patterns
Canonical
Sidecar/service Mesh
Communication
Orchestration
Sagas
Choreography
Workflow event pattern
Data
Event Sourcing
CQRS
Data replication patterns in microservices



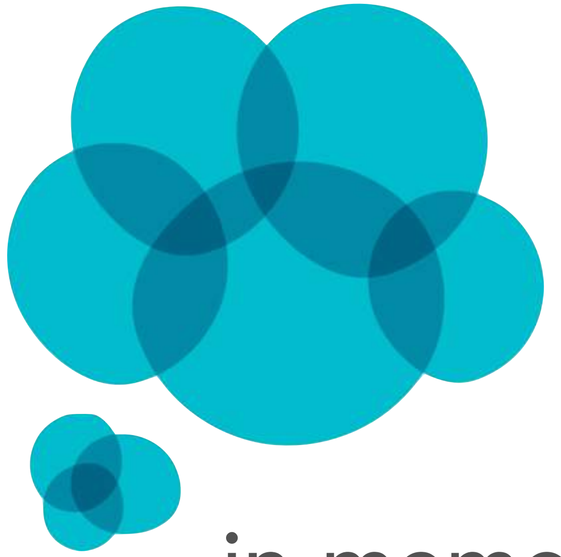
http://martinfowler.com/articles/lmax.html

Overall Structure



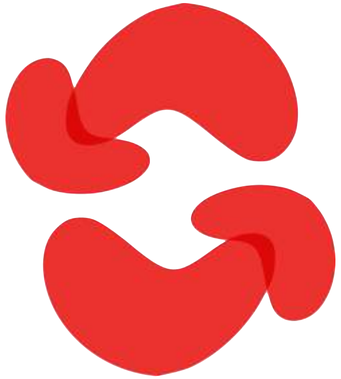
Business Logic Processor

Business Logic Processor

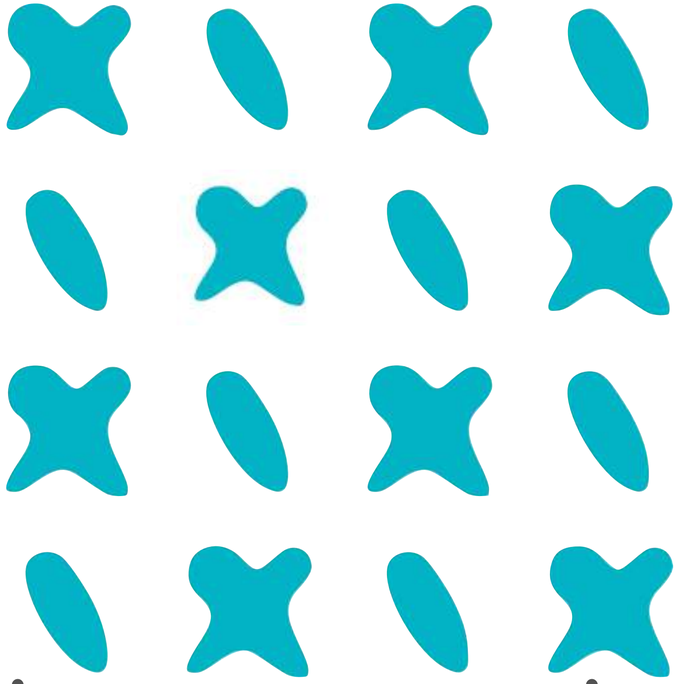


in-memory

snapshots



(full restart—JVM + snapshots— < 1 min)

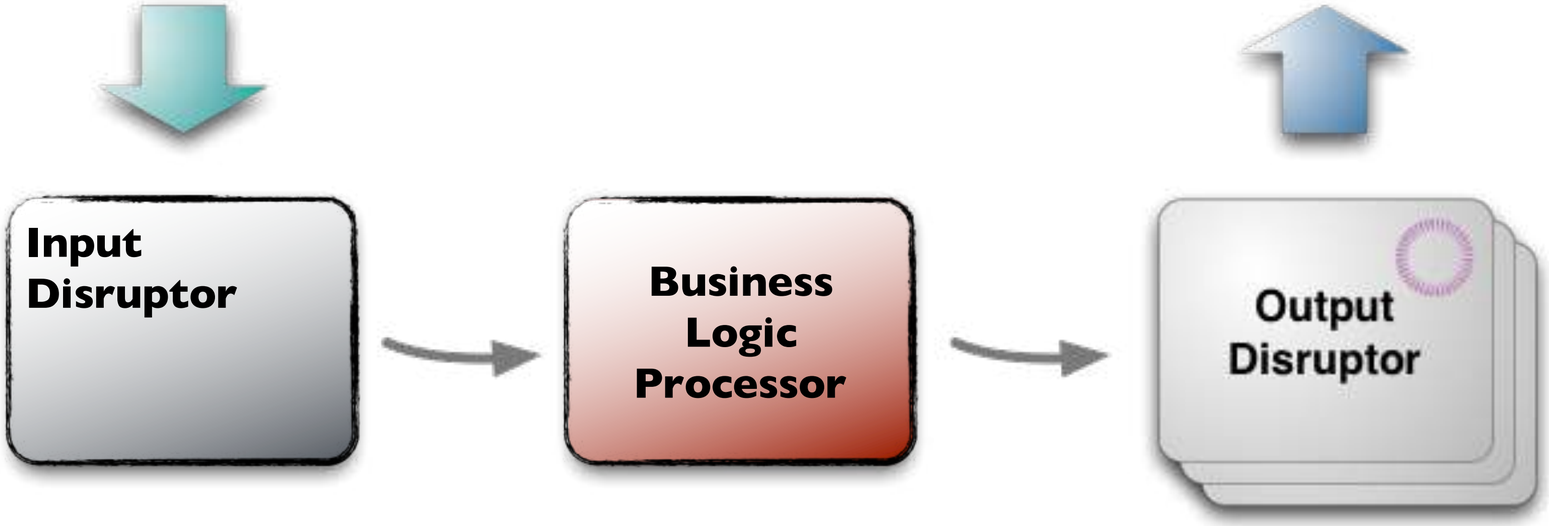


multiple instances running

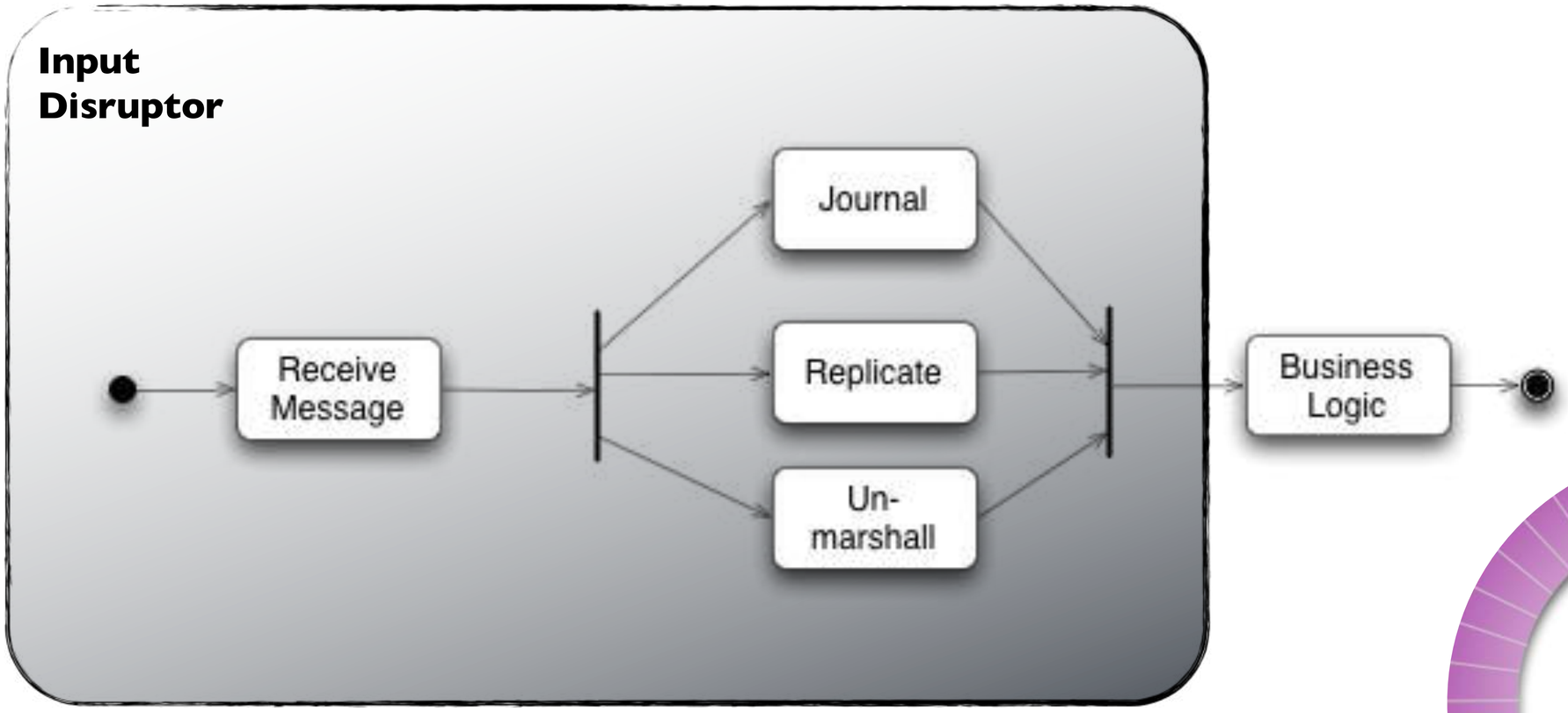
each event processed by multiple processors

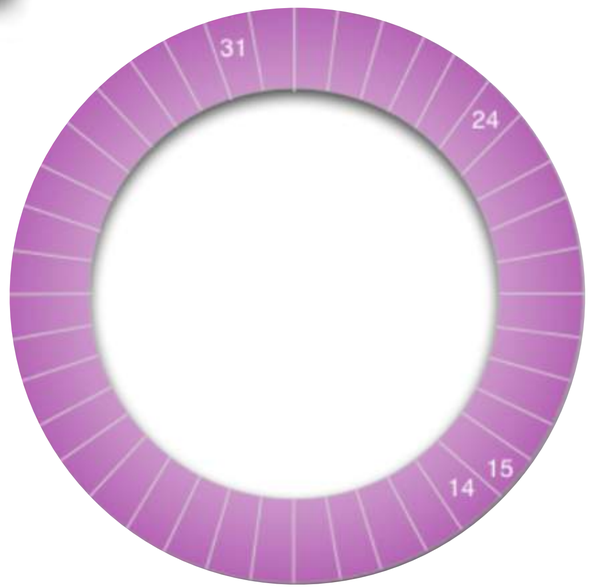
but only one result used

Overall Structure



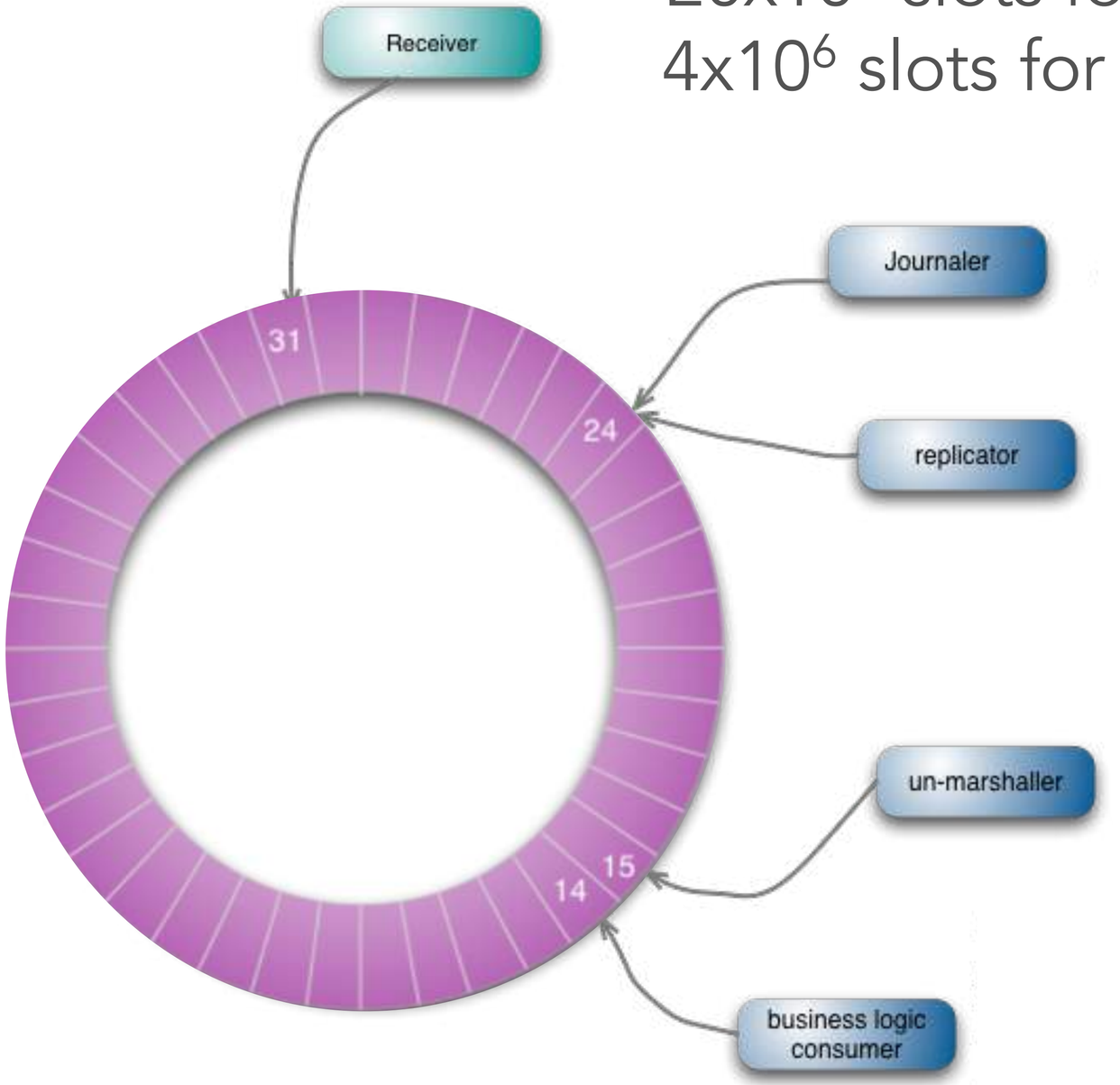
Input Disruptor



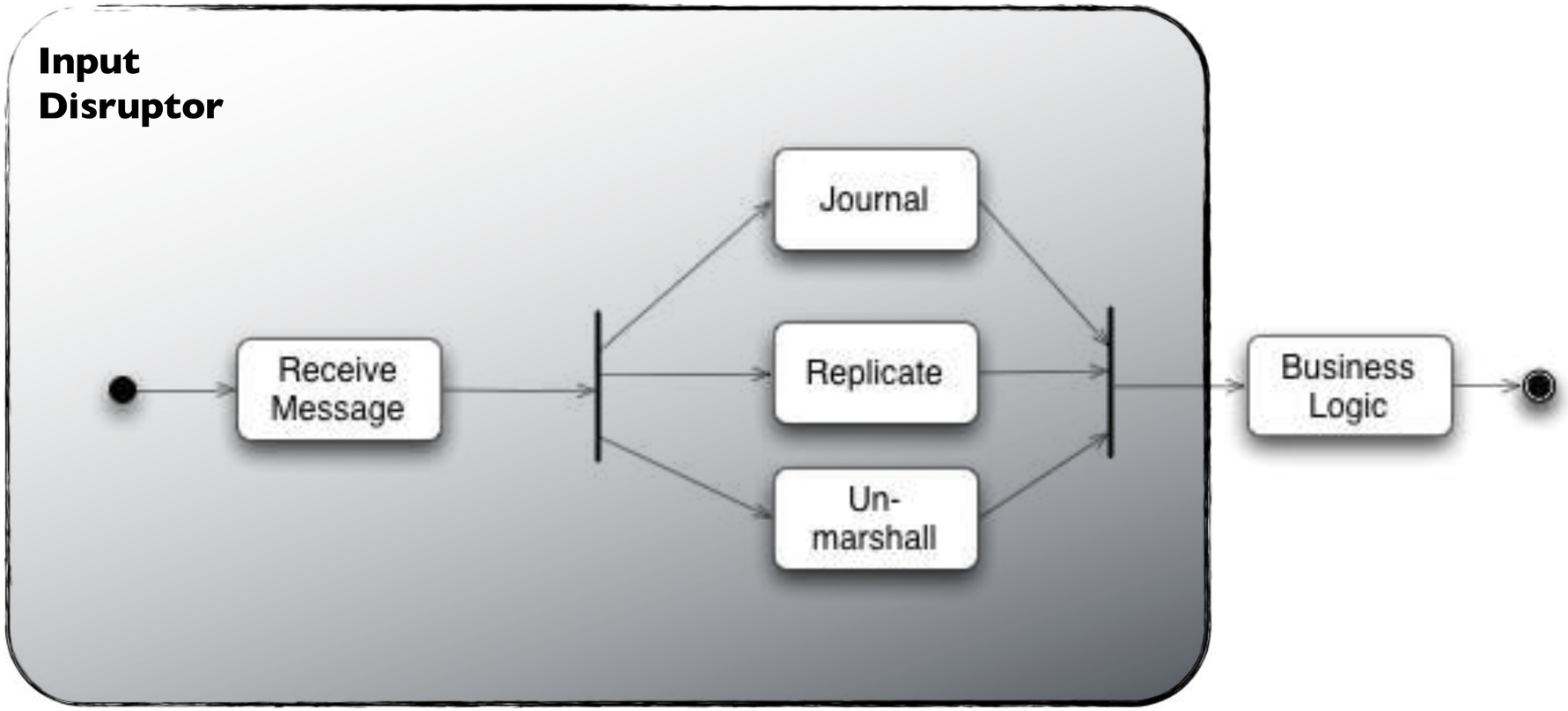


custom concurrency component

$20x10^6$ slots for input buffer
$4x10^6$ slots for output buffer

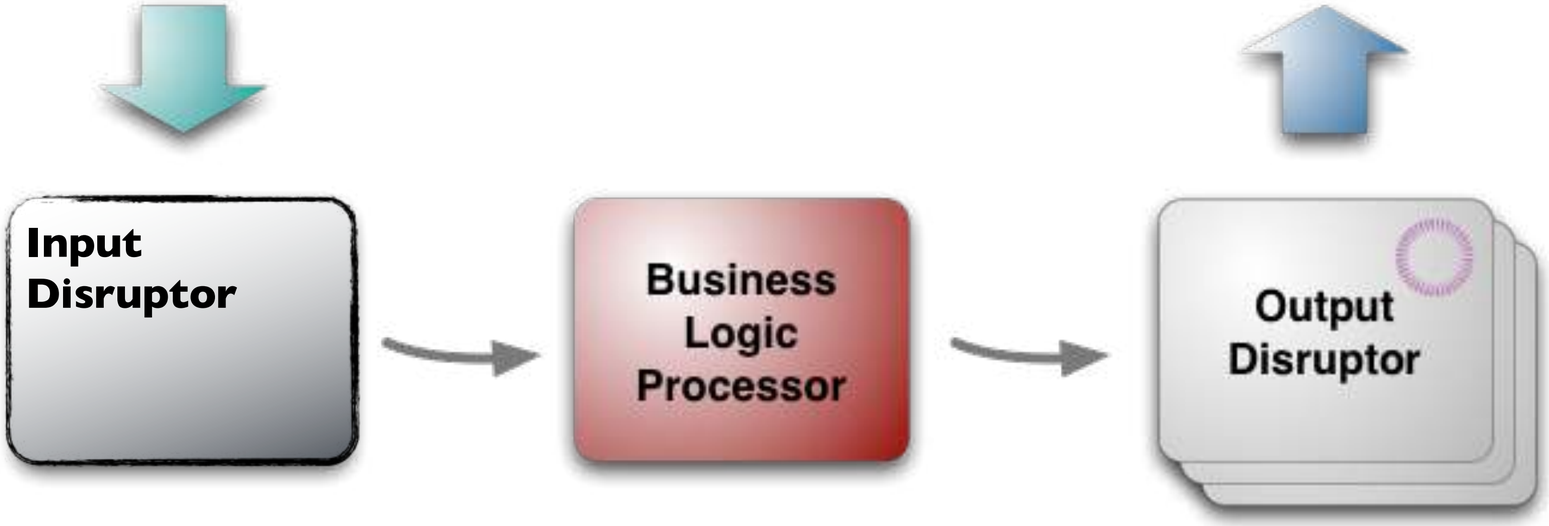


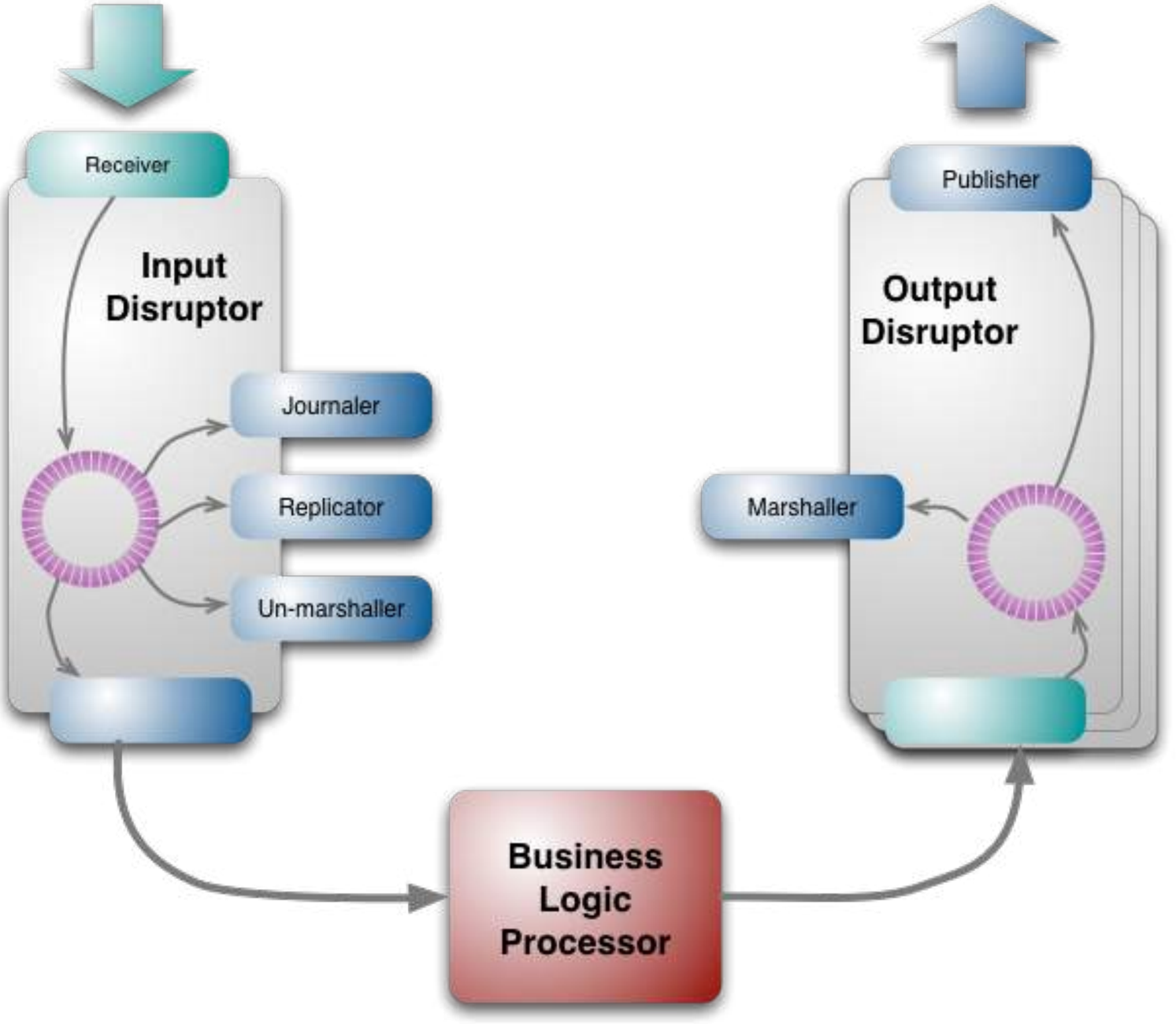
Input Disruptor



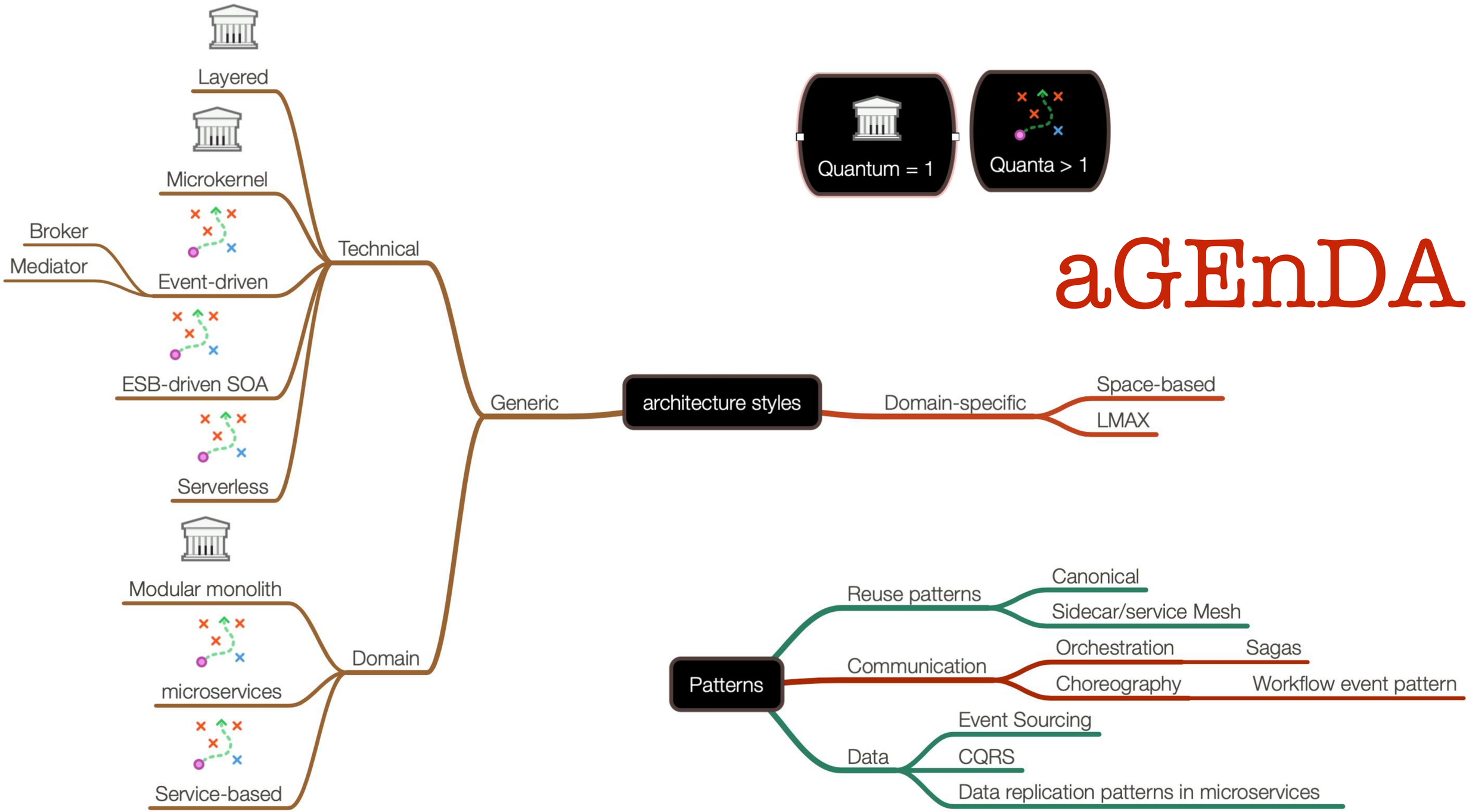
custom concurrency component

Overall Structure




Receiver
Input Disruptor
Journaler
Replicator
Un-marshaller
Publisher
Output Disruptor
Marshaller
Business Logic Processor

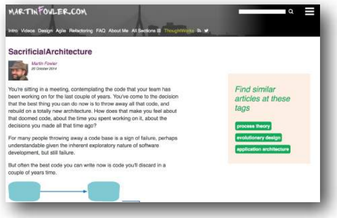
aGEnDA

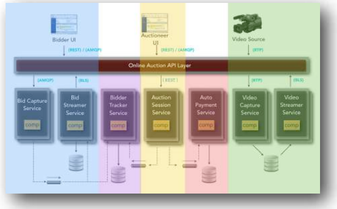

Quantum = 1
Quanta > 1
architecture styles
Generic
Technical
Layered
Microkernel
Event-driven
Broker
Mediator
ESB-driven SOA
Serverless
Domain
Modular monolith
microservices
Service-based
Domain-specific
Space-based
LMAX
Patterns
Reuse patterns
Canonical
Sidecar/service Mesh
Communication
Orchestration
Sagas
Choreography
Workflow event pattern
Data
Event Sourcing
CQRS
Data replication patterns in microservices

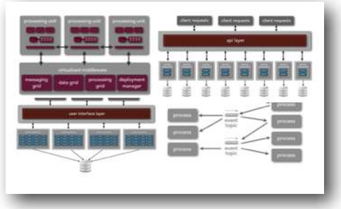
Tradeoffs of Distributed Architecture

tradeoffs of distributed architecture

the good

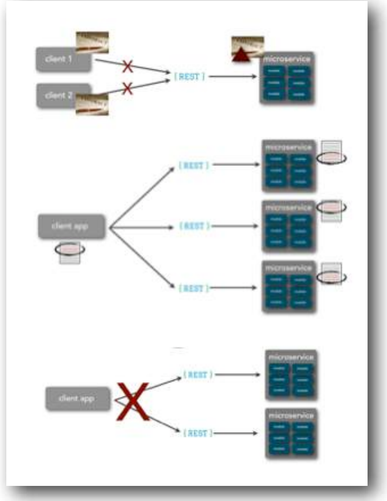






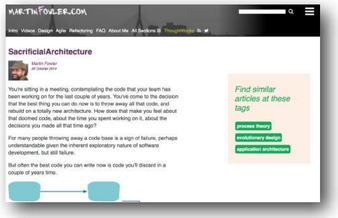
Agility
Scalability
Deployability
Fault Tolerance

the bad



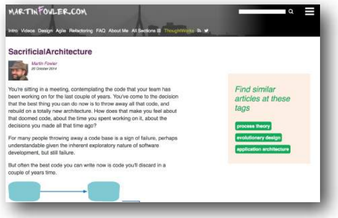
tradeoffs of distributed architecture

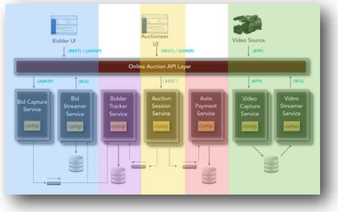
the good



tradeoffs of distributed architecture

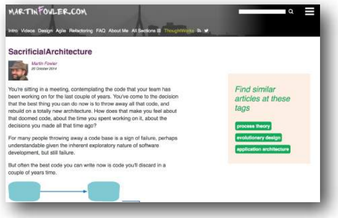
the good

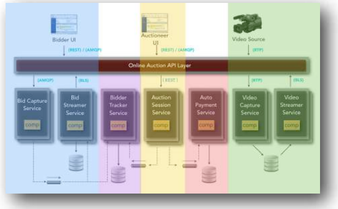


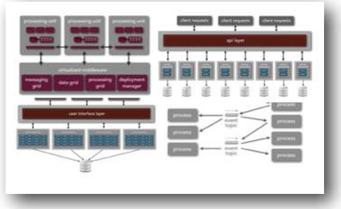


tradeoffs of distributed architecture

the good

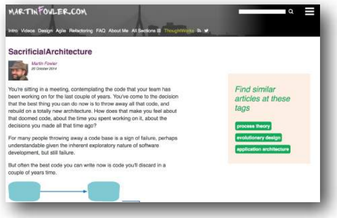


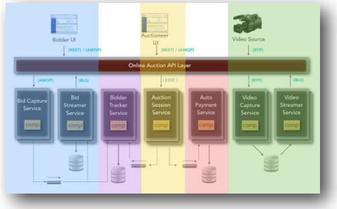


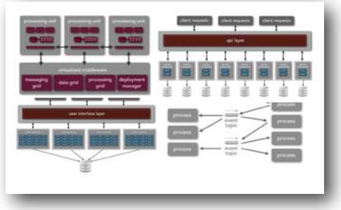


tradeoffs of distributed architecture

the good



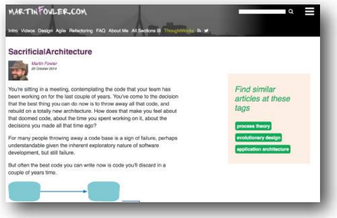


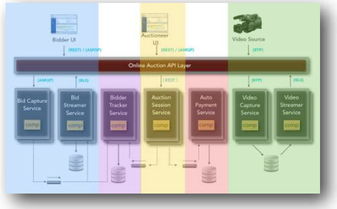


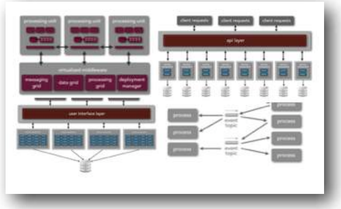
Agility
Scalability
Deployability
Fault Tolerance

tradeoffs of distributed architecture

the good





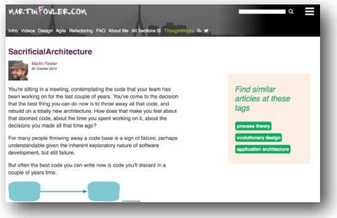


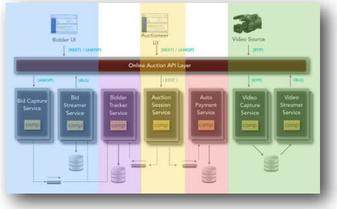
Agility
Scalability
Deployability
Fault Tolerance

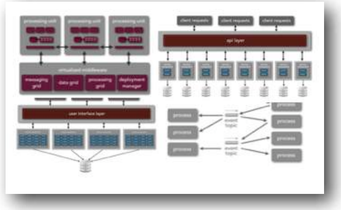
the bad

tradeoffs of distributed architecture

the good

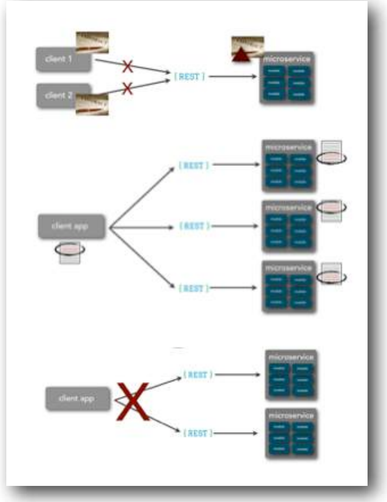






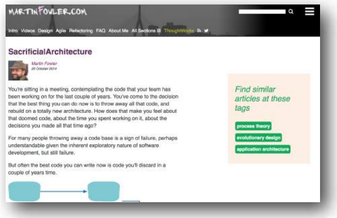
Agility
Scalability
Deployability
Fault Tolerance

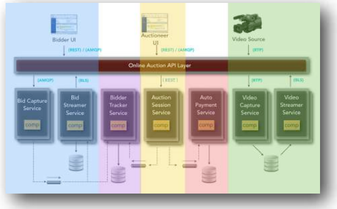
the bad

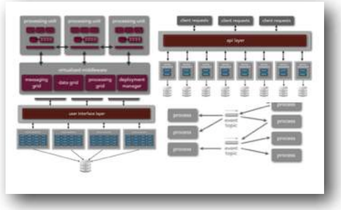


tradeoffs of distributed architecture

the good

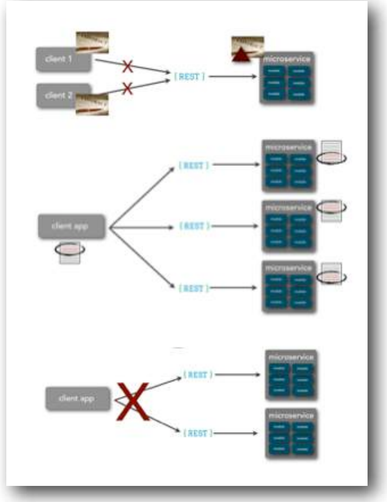






Agility
Scalability
Deployability
Fault Tolerance

the bad



en.wikipedia.org

Create account Log in

Article Talk Read Edit View history Search

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF

Fallacies of distributed computing

From Wikipedia, the free encyclopedia

The **Fallacies of Distributed Computing** are a set of assumptions that L Peter Deutsch and others at Sun Microsystems originally asserted programmers new to distributed applications invariably make. These assumptions ultimately prove false, resulting either in the failure of the system, a substantial reduction in system scope, or in large, unplanned expenses required to redesign the system to meet its original goals.[*citation needed*]

Contents [hide]

1 The fallacies
2 Effects of the fallacies
3 History
4 See also
5 References
6 External links

The fallacies [edit]

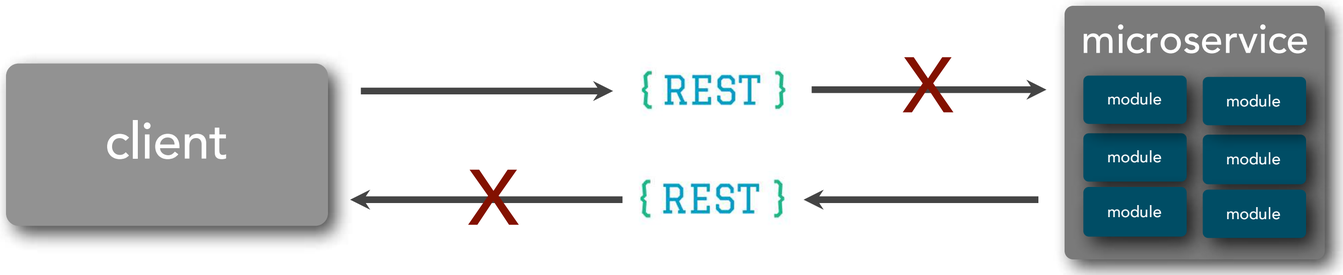
The fallacies are summarized below:[1]

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator

en.wikipedia.org/wiki/Fallacies_of_distributed_computing

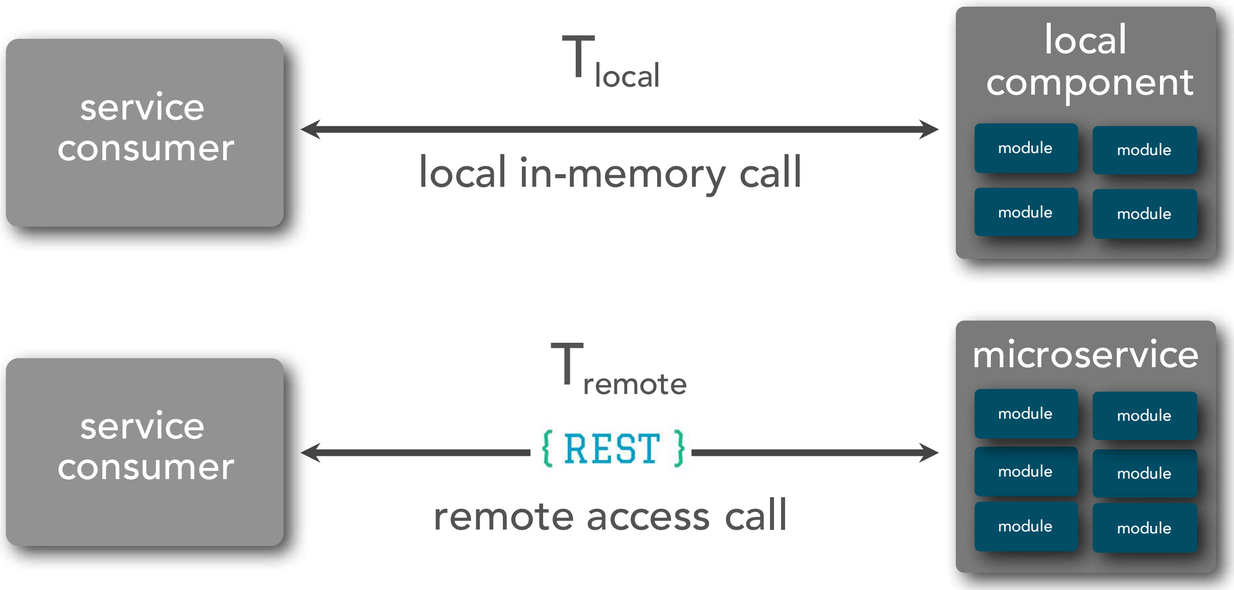
tradeoffs of distributed architecture

1. the network is reliable



tradeoffs of distributed architecture

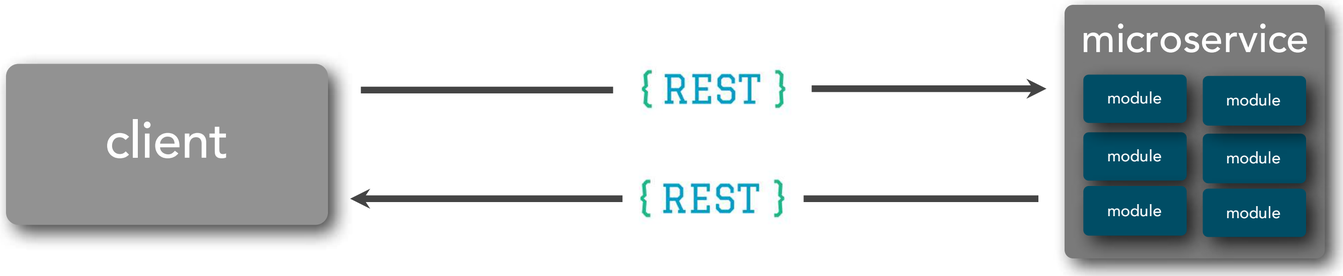
2. latency is zero

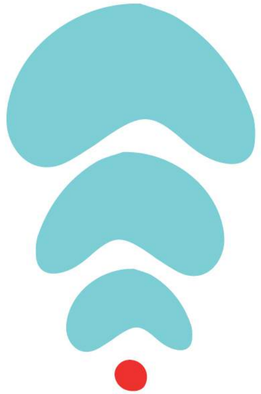


$T_{remote} > T_{local}$

tradeoffs of distributed architecture

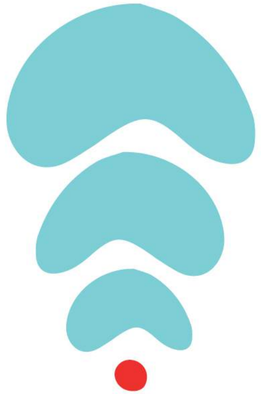
3. bandwidth is infinite





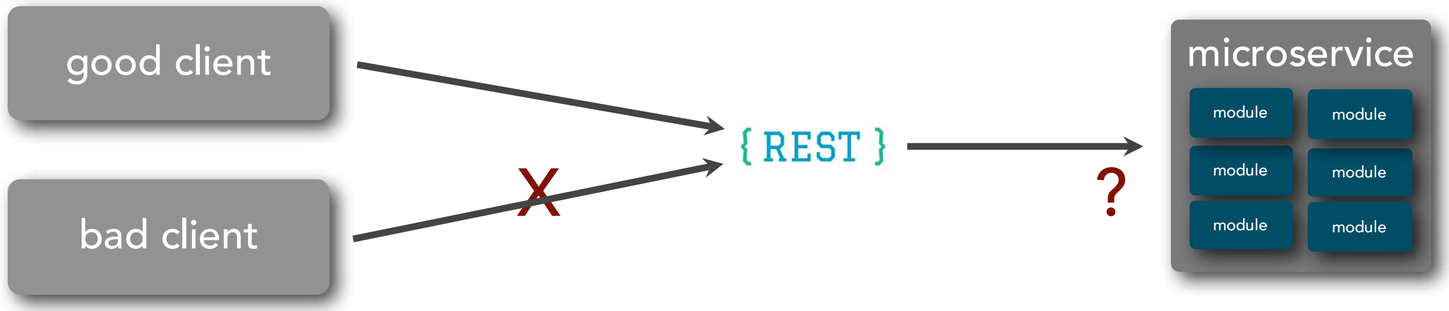
tradeoffs of distributed architecture

3. bandwidth is infinite



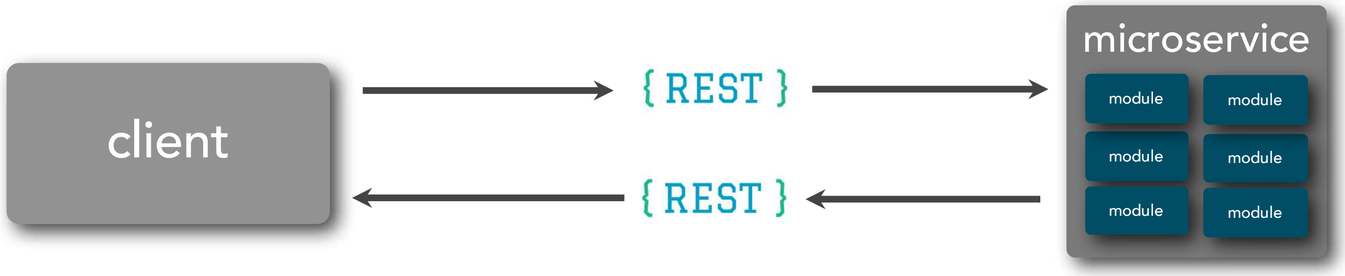
tradeoffs of distributed architecture

4. the network is secure



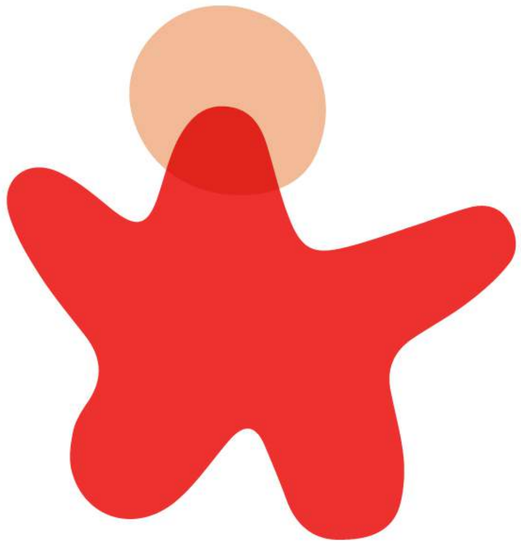
tradeoffs of distributed architecture

5. topology doesn't change



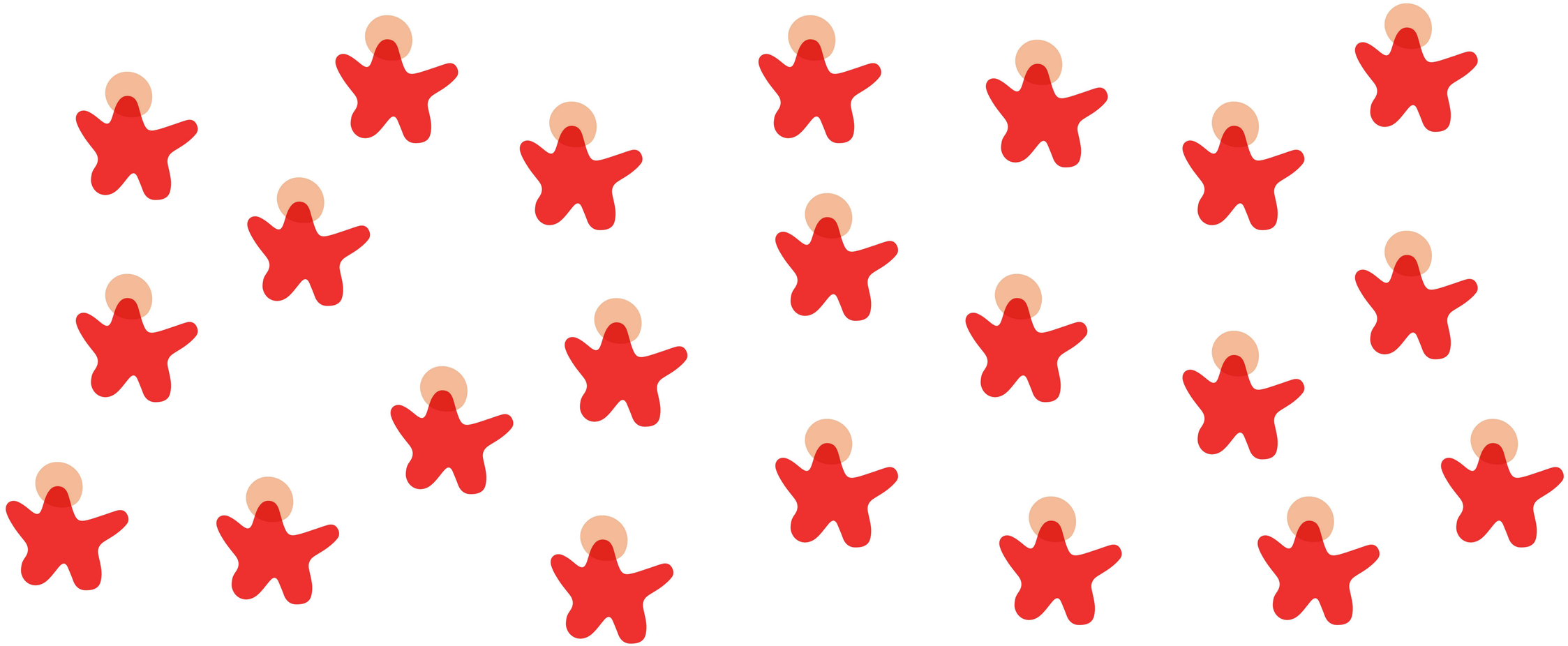
tradeoffs of distributed architecture

6. there is one administrator



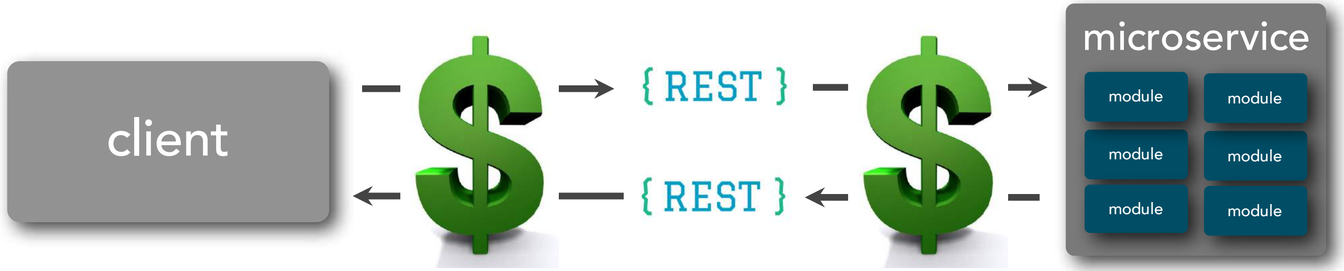
tradeoffs of distributed architecture

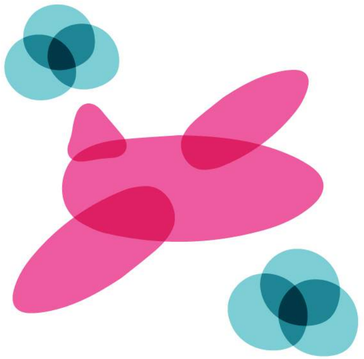
6. there is one administrator



tradeoffs of distributed architecture

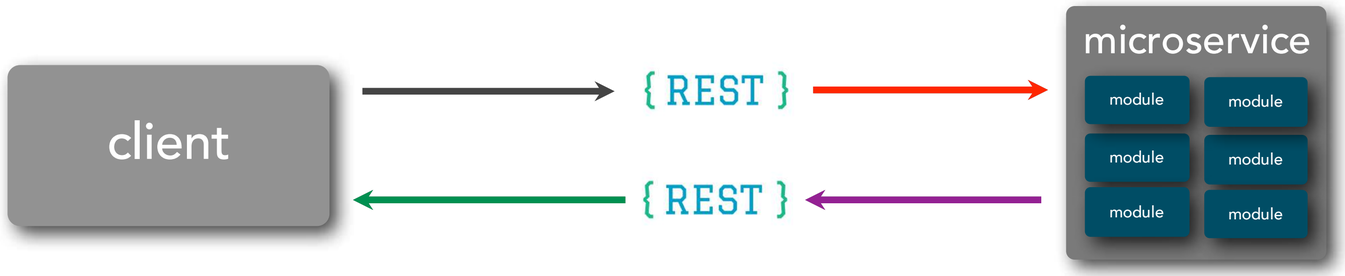
7. transport cost is zero





tradeoffs of distributed architecture

8. the network is homogeneous



en.wikipedia.org

Create account Log in

Article Talk Read Edit View history Search

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF

Fallacies of distributed computing

From Wikipedia, the free encyclopedia

The **Fallacies of Distributed Computing** are a set of assumptions that L Peter Deutsch and others at Sun Microsystems originally asserted programmers new to distributed applications invariably make. These assumptions ultimately prove false, resulting either in the failure of the system, a substantial reduction in system scope, or in large, unplanned expenses required to redesign the system to meet its original goals.[*citation needed*]

Contents [hide]

1 The fallacies
2 Effects of the fallacies
3 History
4 See also
5 References
6 External links

The fallacies [edit]

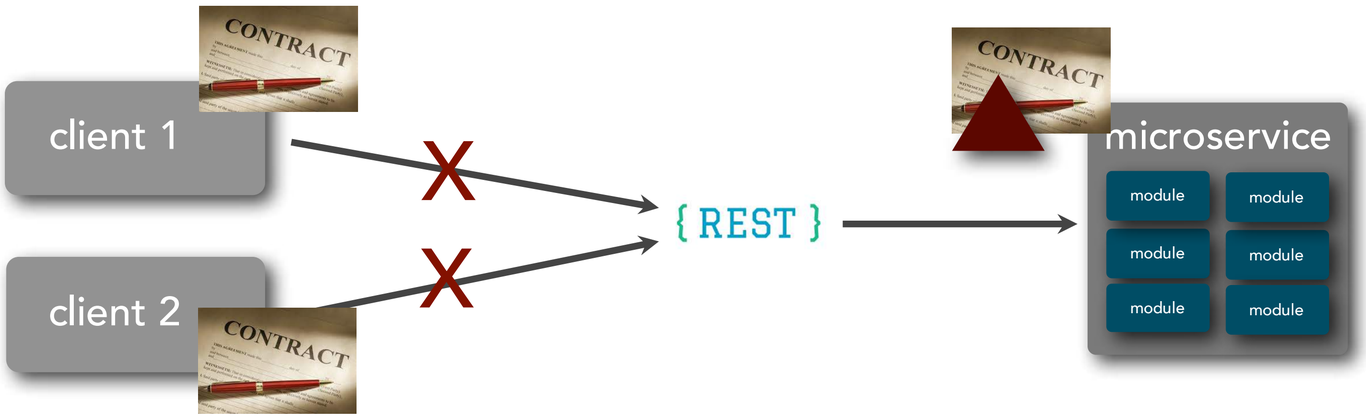
The fallacies are summarized below:[1]

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator

en.wikipedia.org/wiki/Fallacies_of_distributed_computing

tradeoffs of distributed architecture

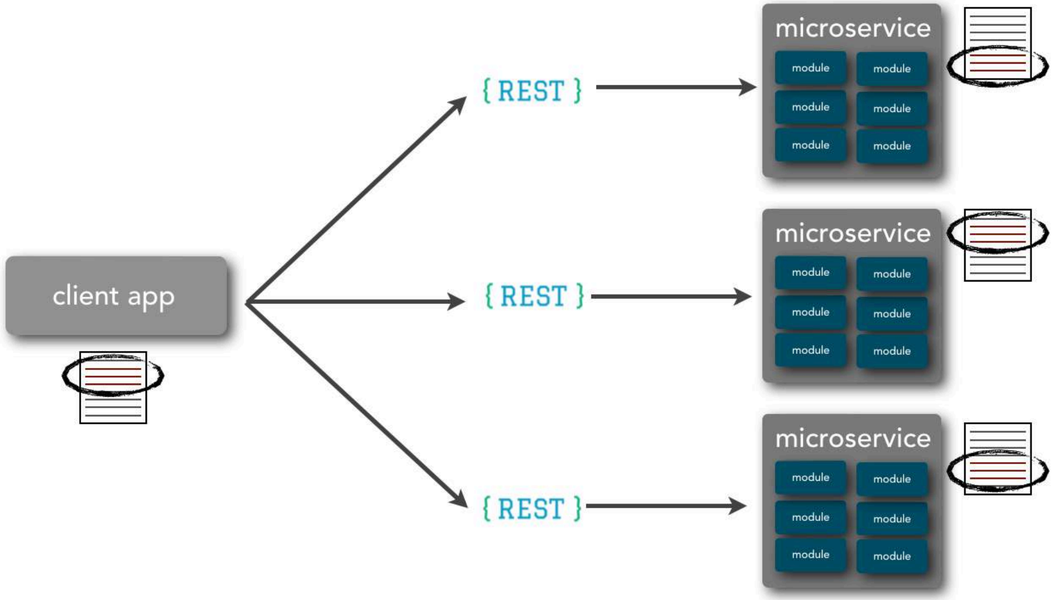
contract creation, maintenance, versioning,
and coordination



https://www.developertoarchitect.com/lessons/lesson41.html

tradeoffs of distributed architecture

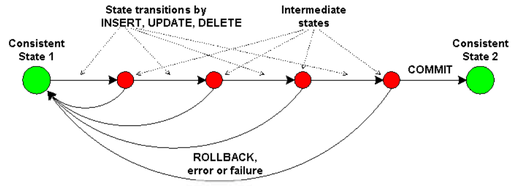
distributed logging facilities to provide a holistic view of a transaction

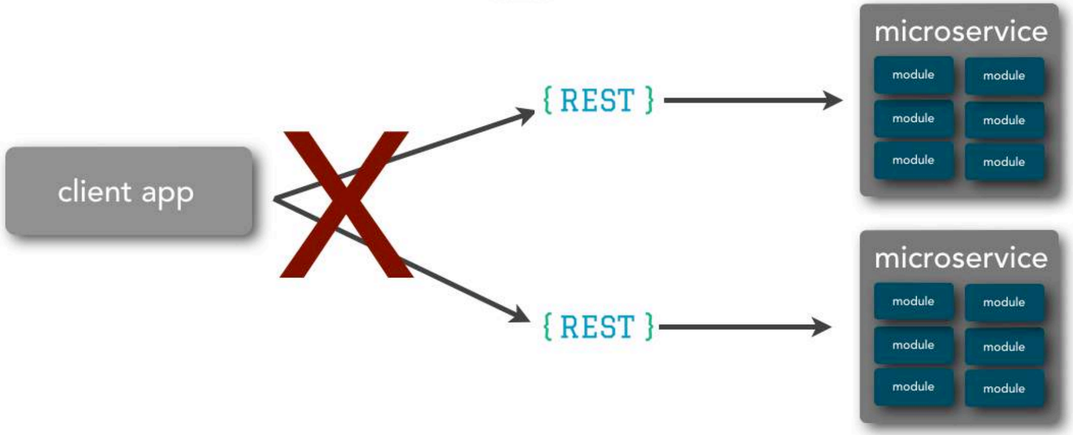


https://www.developertoarchitect.com/lessons/lesson4.html

tradeoffs of distributed architecture

distributed transaction management

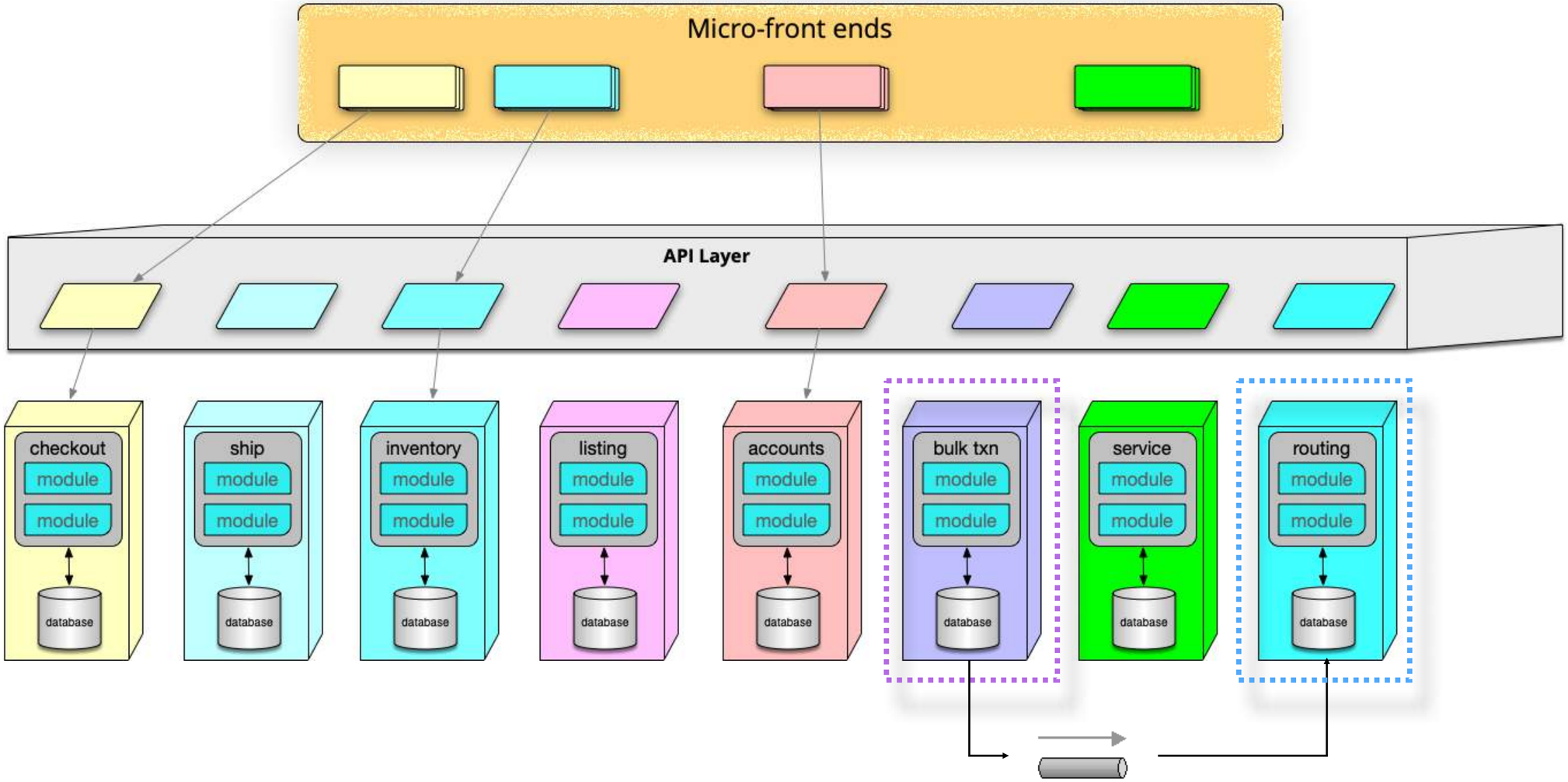




https://www.developertoarchitect.com/lessons/lesson53.html

The Value of Asynchronicity

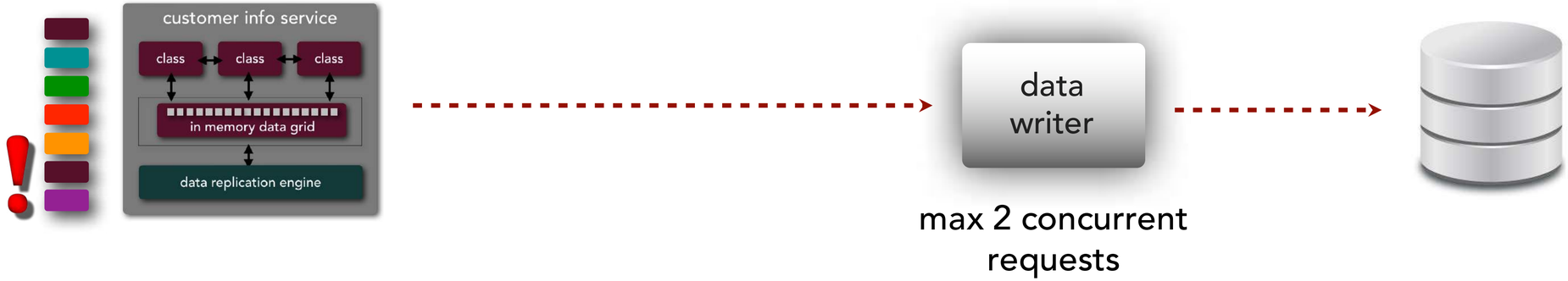
the value of asynchronicity



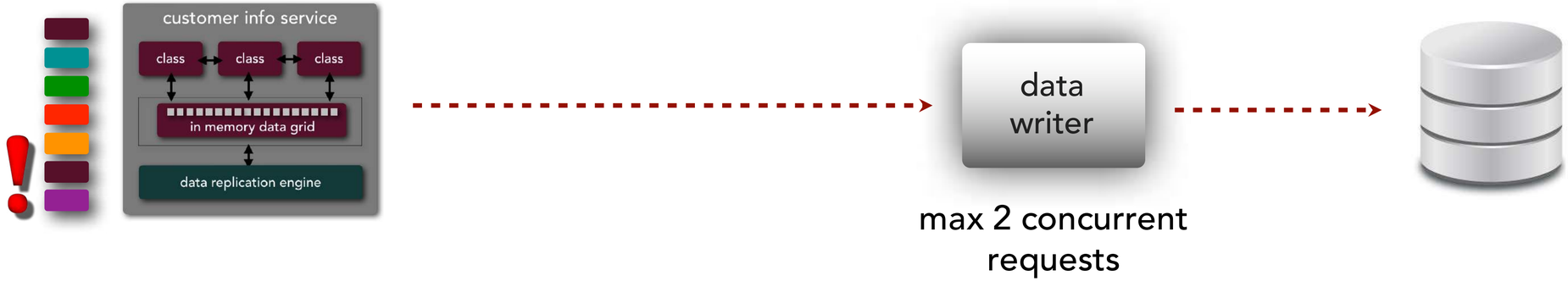
asynchronous

Leveraging Queues For Back Pressure

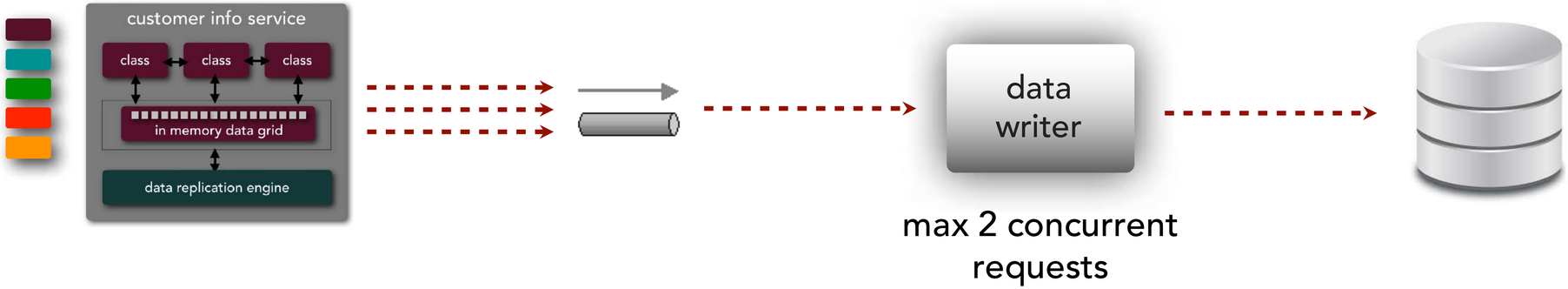
leveraging queues for back pressure



leveraging queues for back pressure

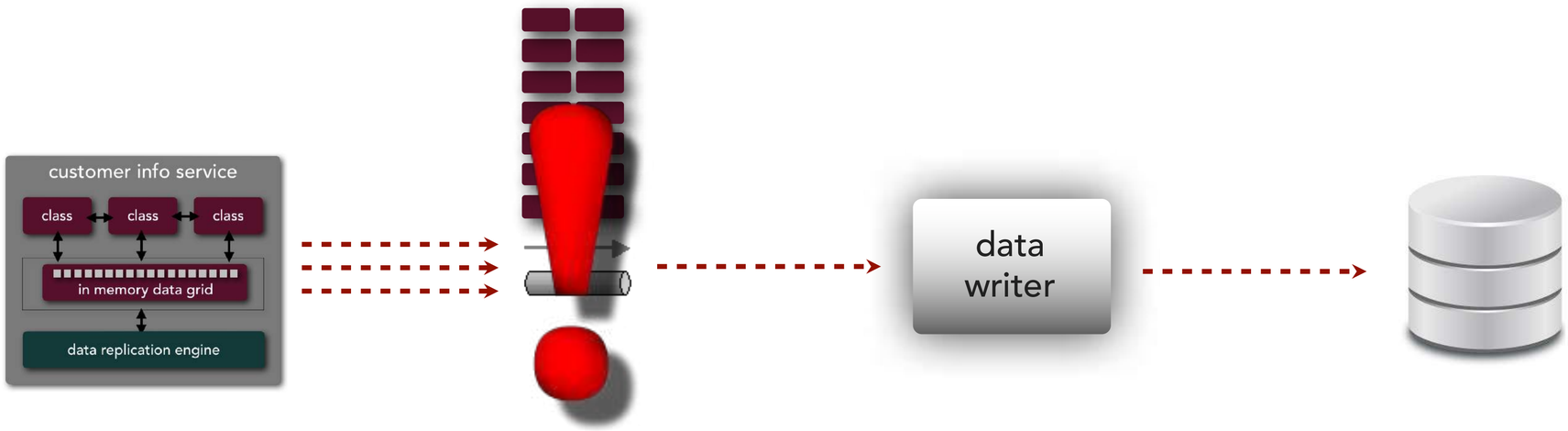


leveraging queues for back pressure



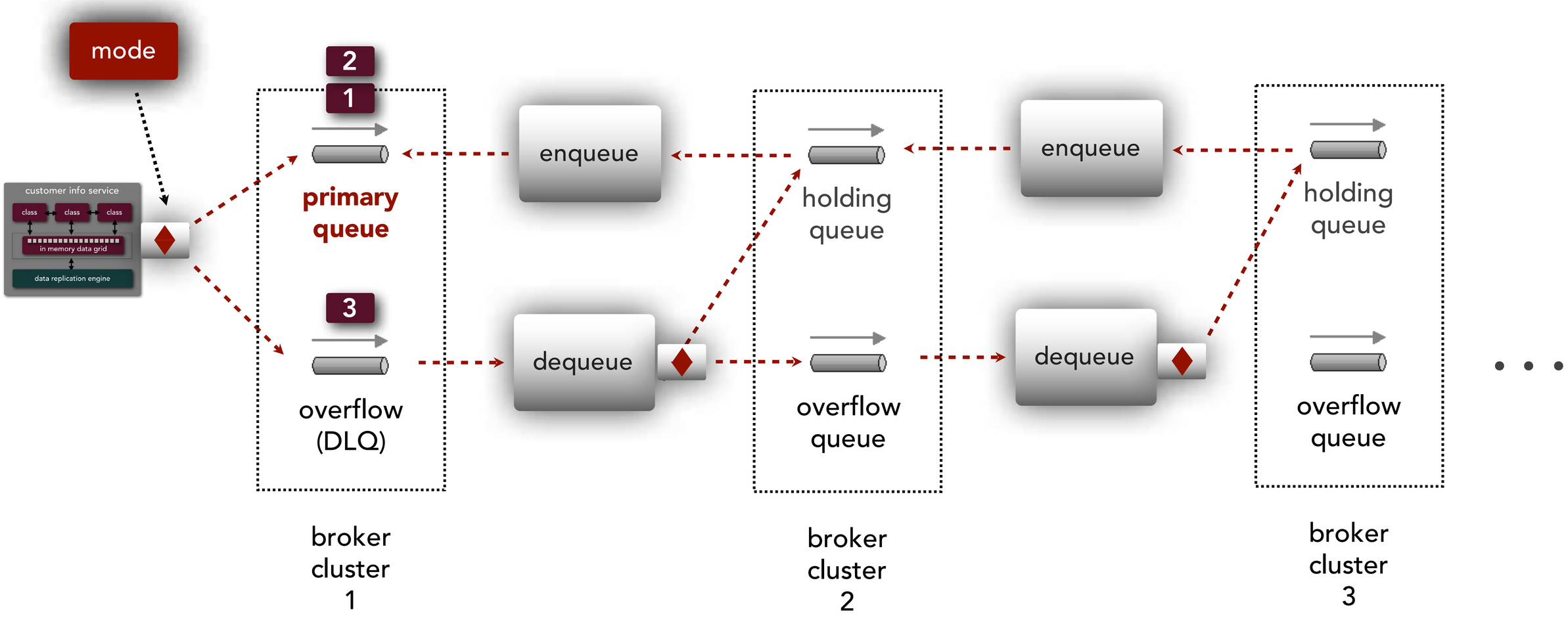
leveraging queues for back pressure

managing queue overflow



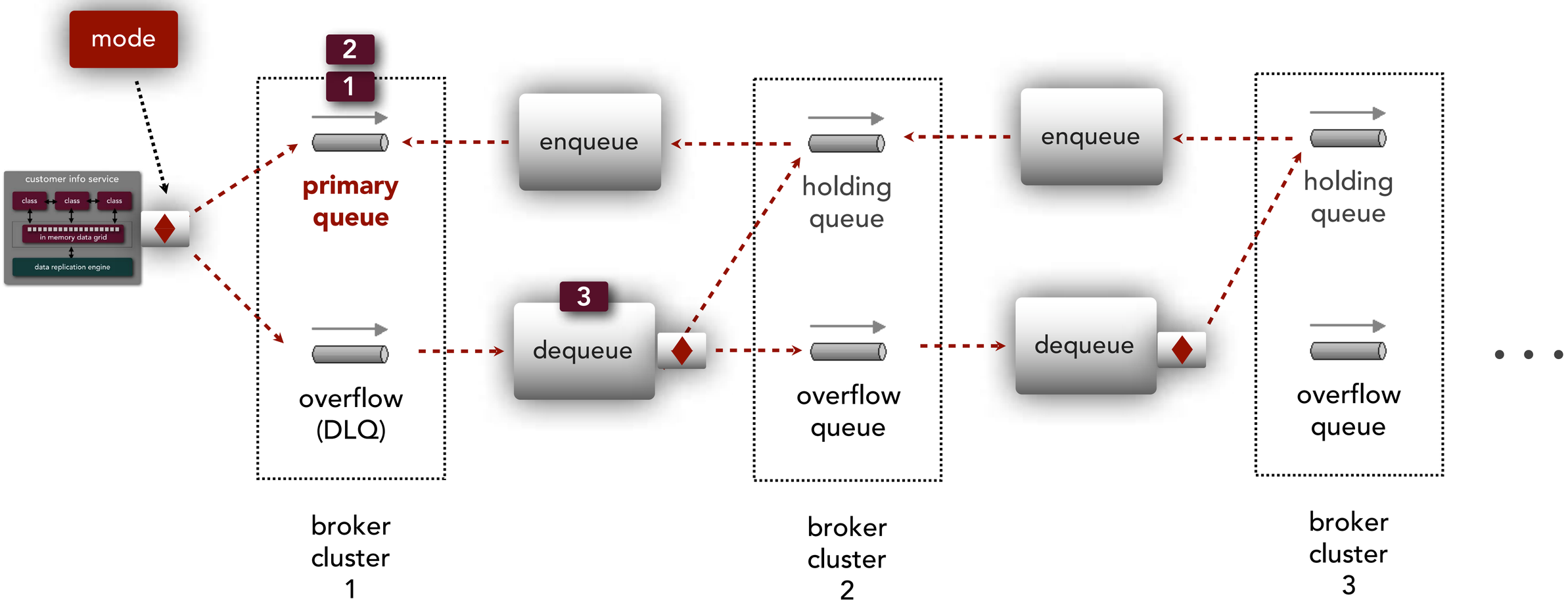
leveraging queues for back pressure

generic overflow pattern (message order preserved)



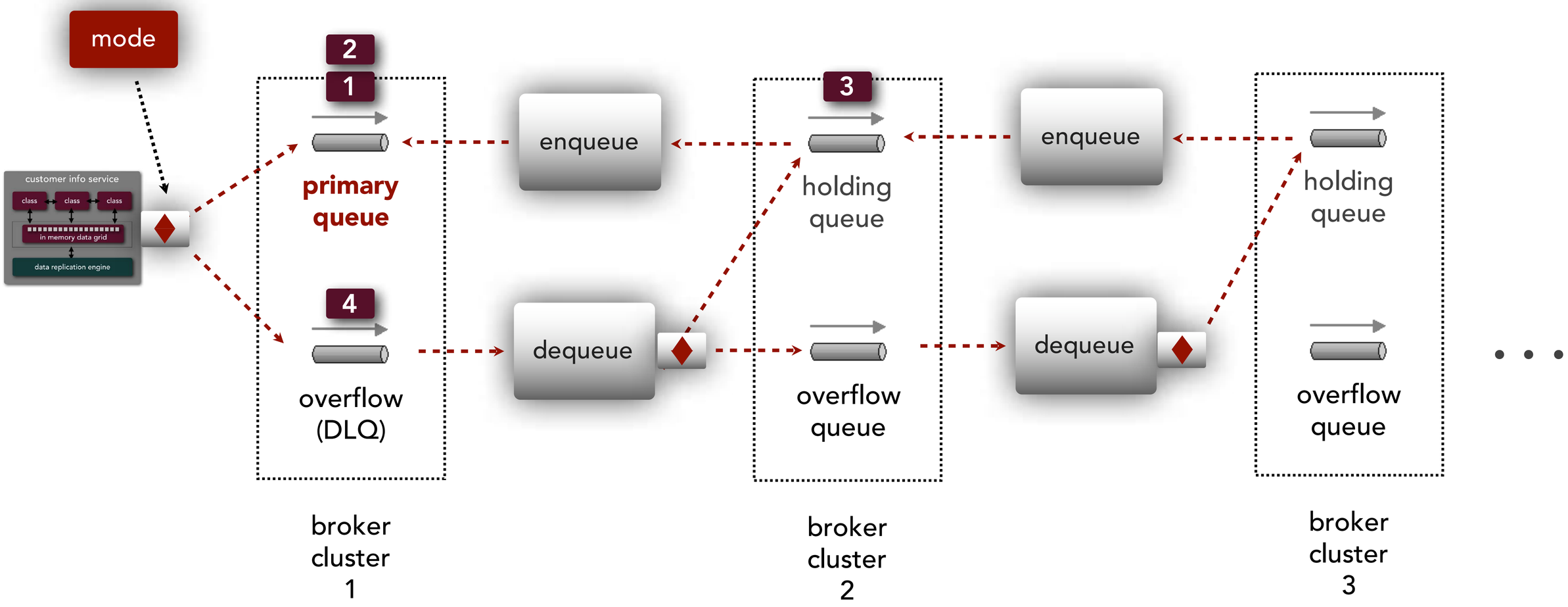
leveraging queues for back pressure

generic overflow pattern (message order preserved)



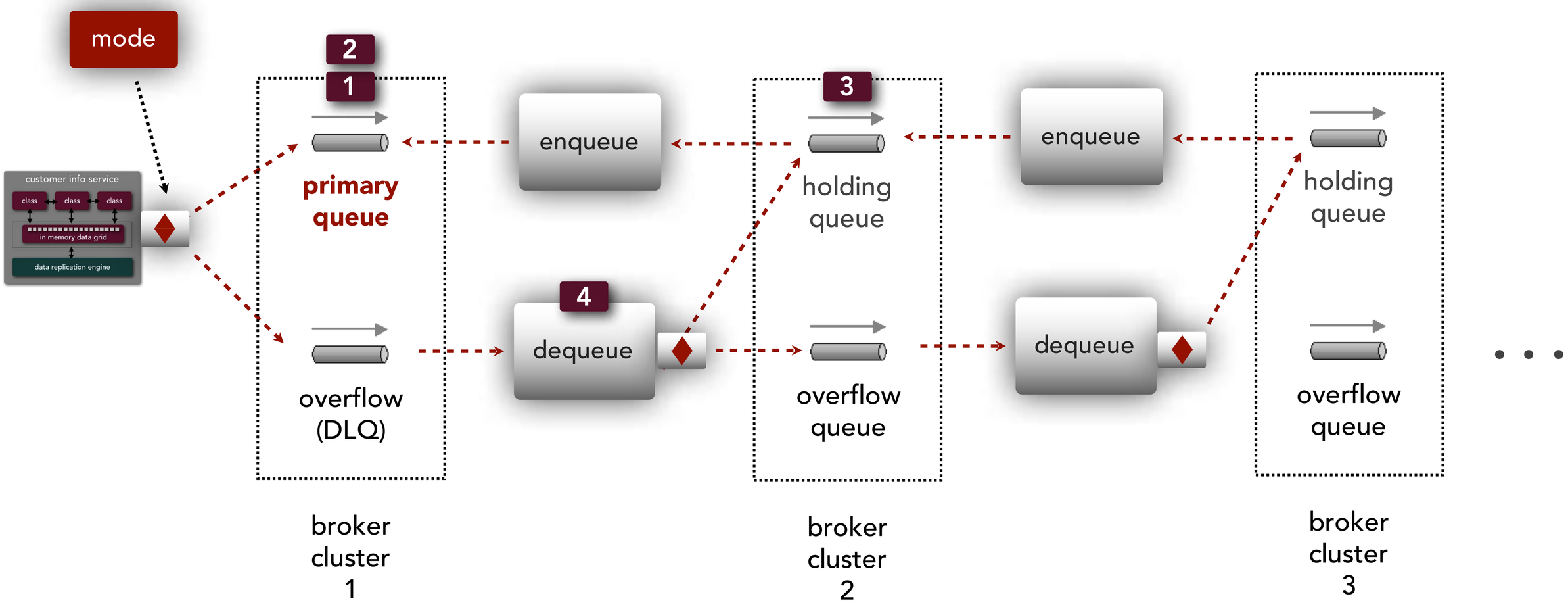
leveraging queues for back pressure

generic overflow pattern (message order preserved)



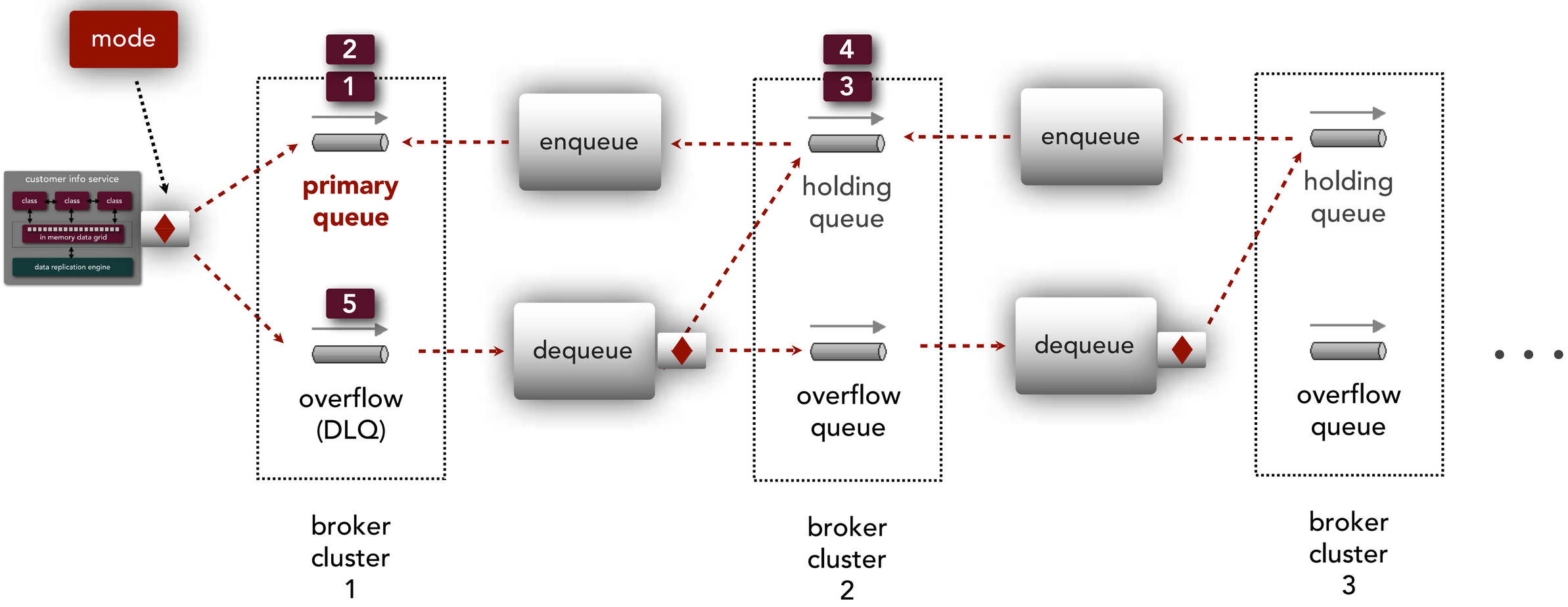
leveraging queues for back pressure

generic overflow pattern (message order preserved)



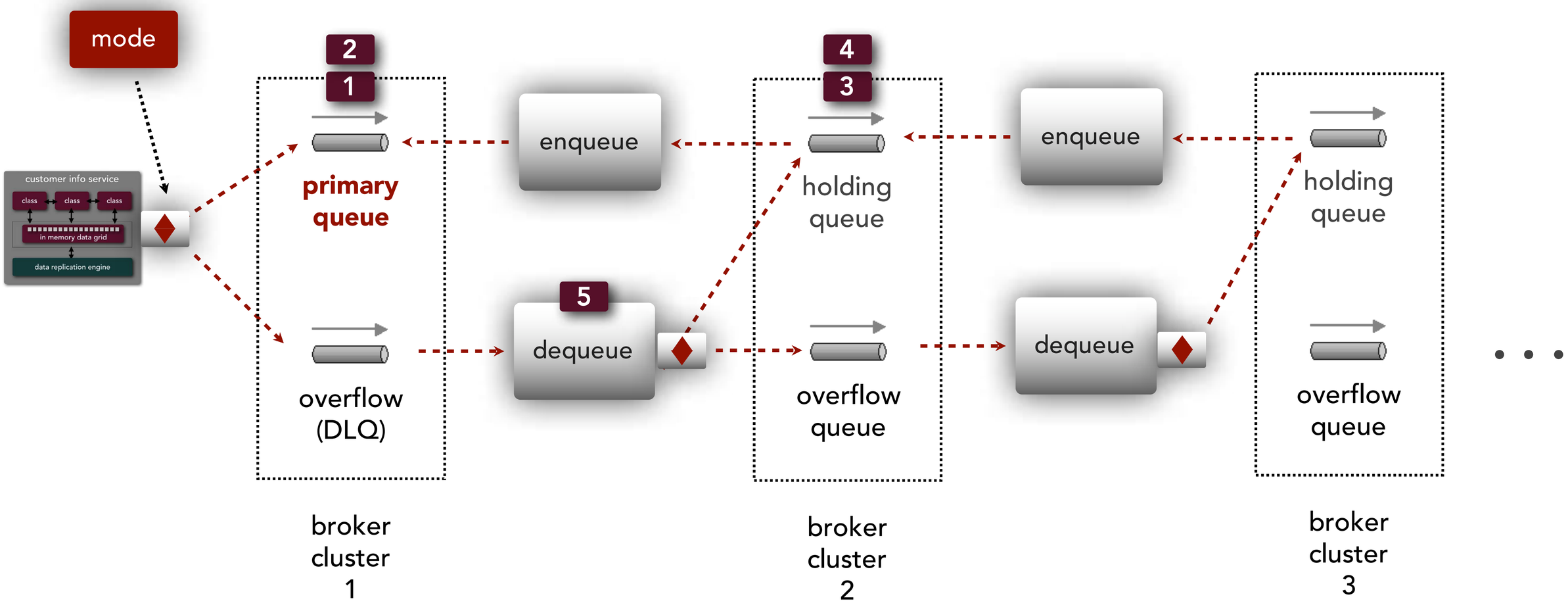
leveraging queues for back pressure

generic overflow pattern (message order preserved)



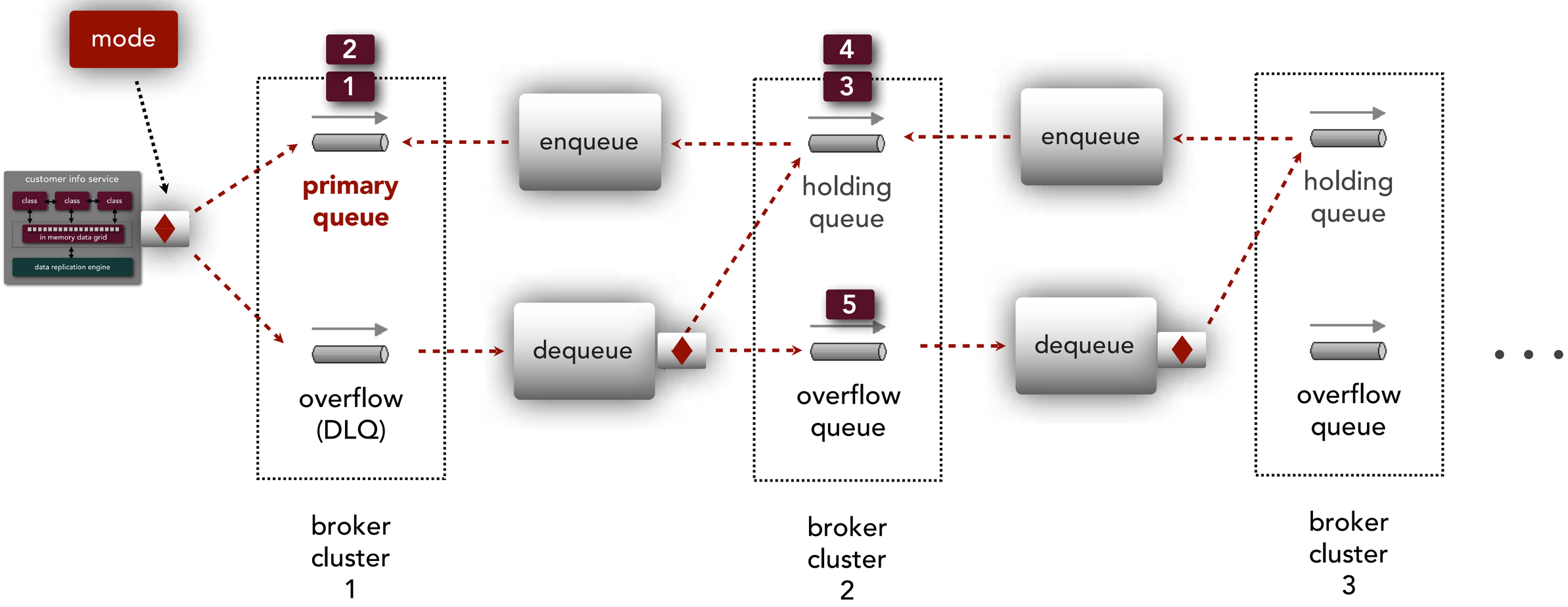
leveraging queues for back pressure

generic overflow pattern (message order preserved)



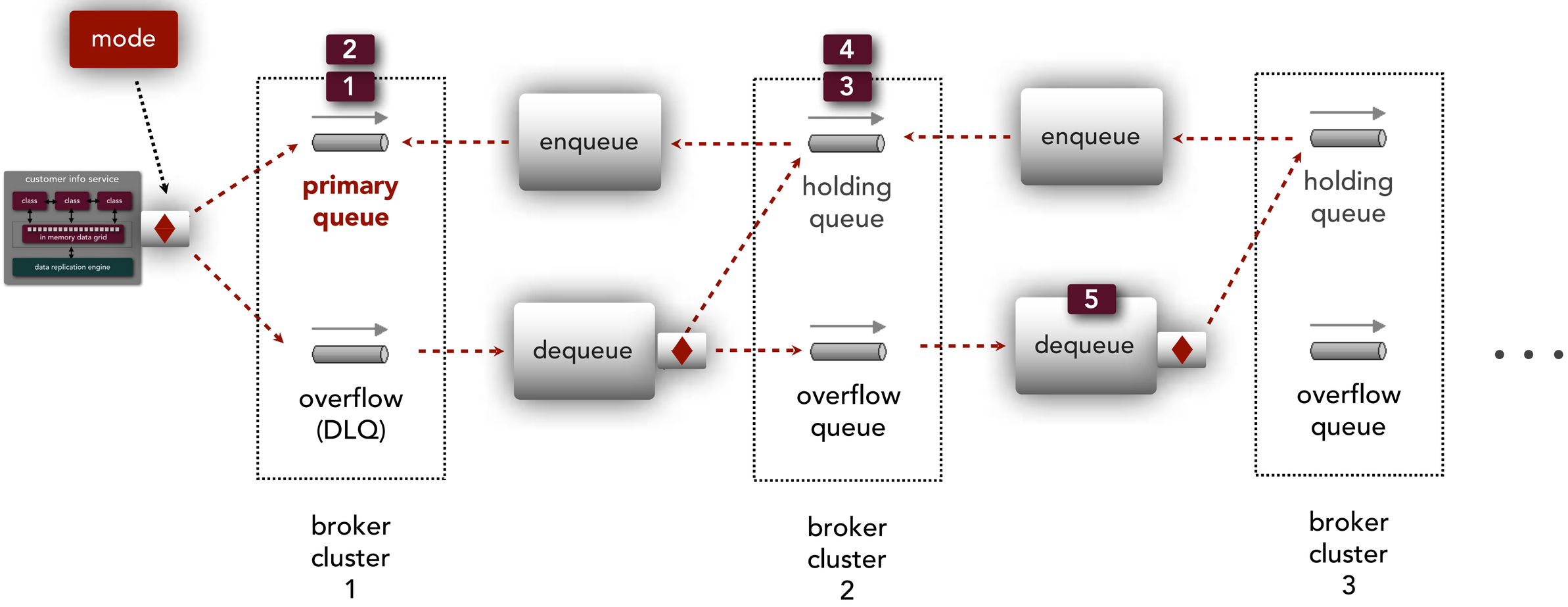
leveraging queues for back pressure

generic overflow pattern (message order preserved)



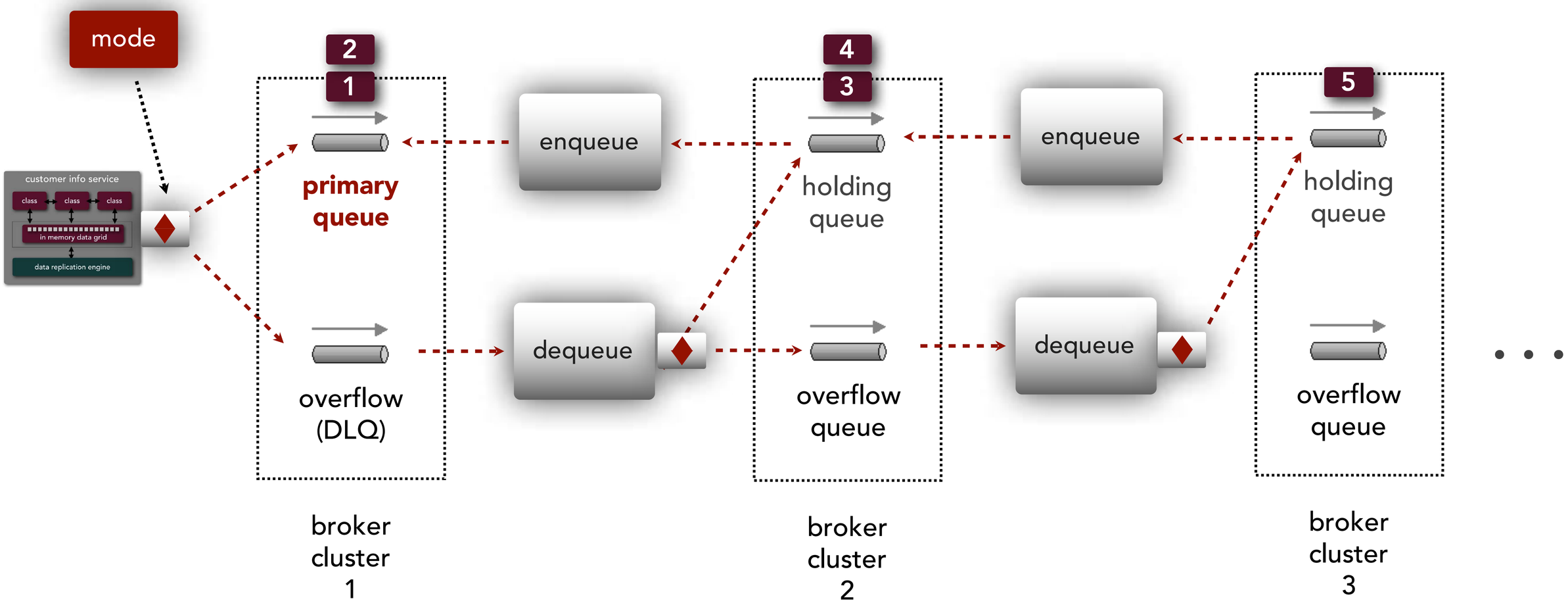
leveraging queues for back pressure

generic overflow pattern (message order preserved)



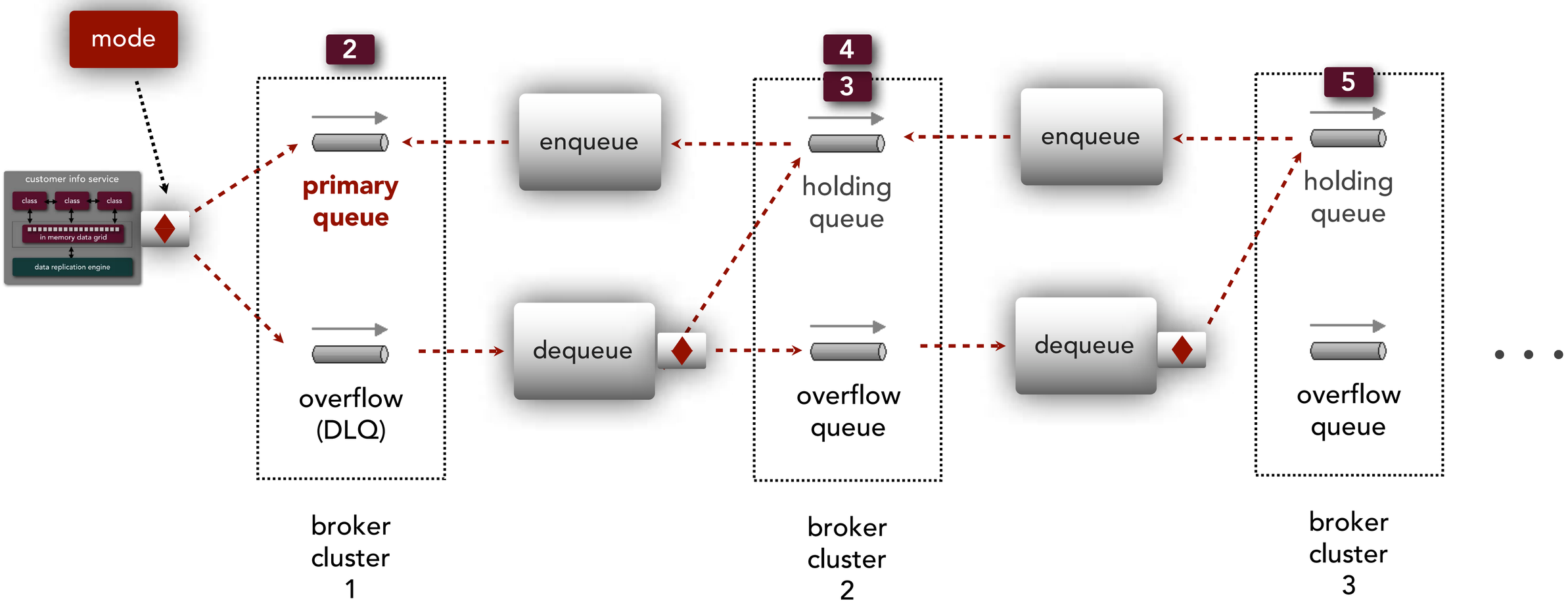
leveraging queues for back pressure

generic overflow pattern (message order preserved)



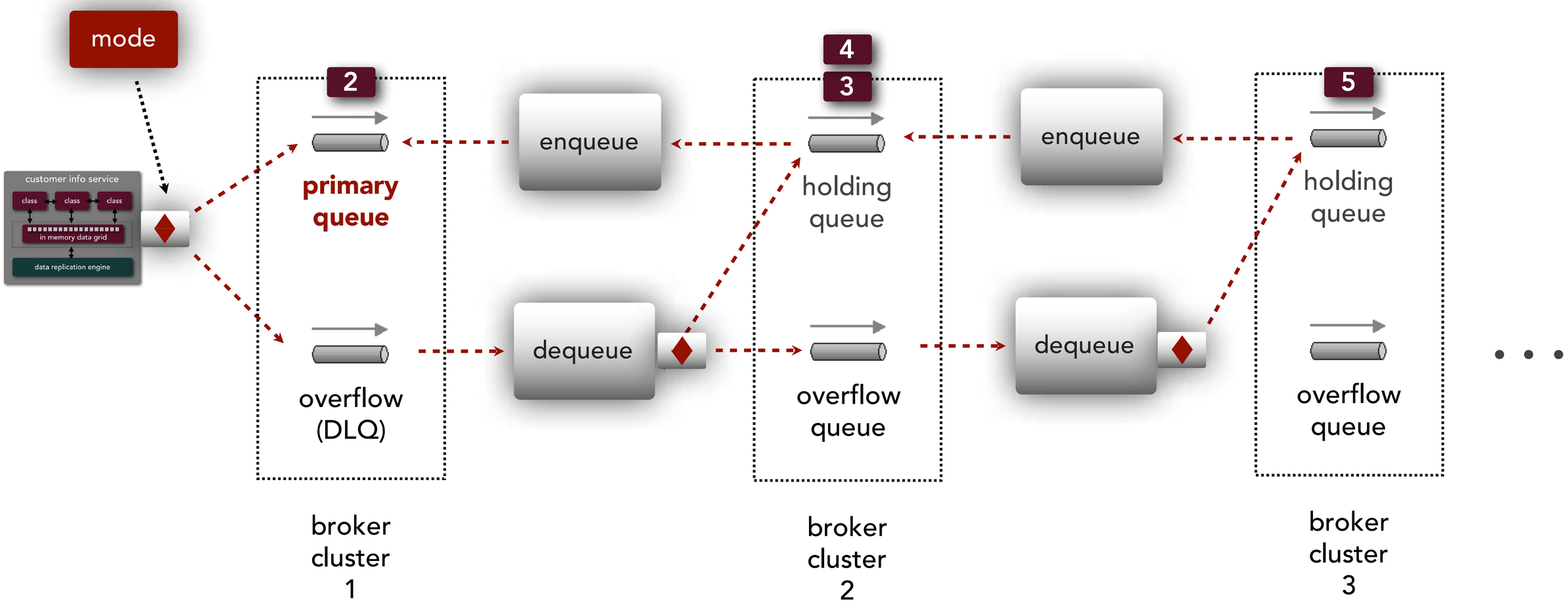
leveraging queues for back pressure

generic overflow pattern (message order preserved)



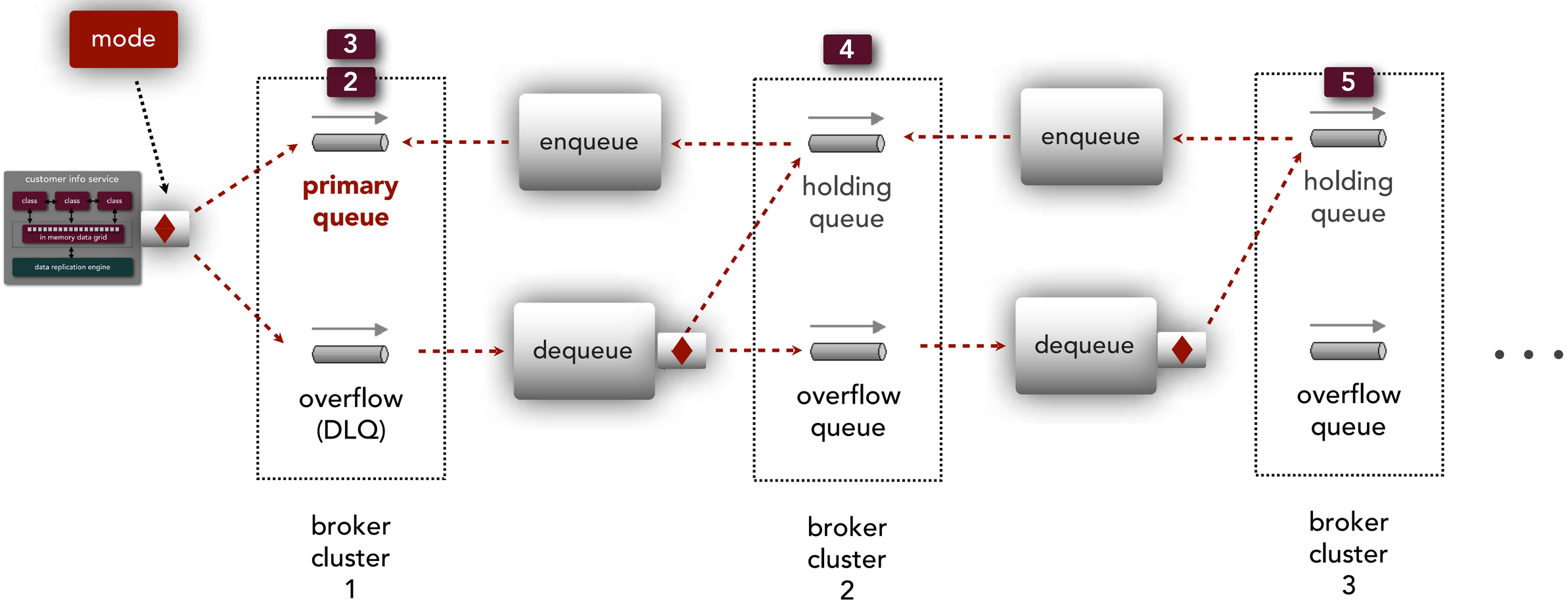
leveraging queues for back pressure

generic overflow pattern (message order preserved)



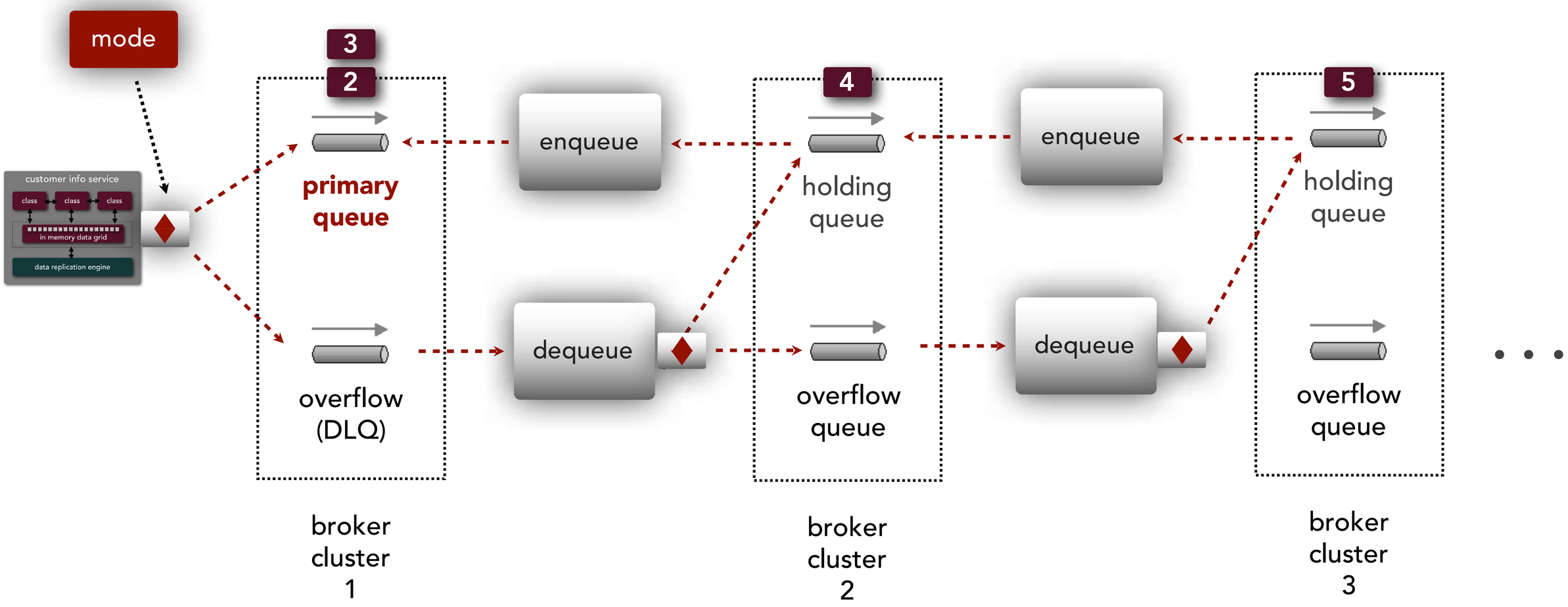
leveraging queues for back pressure

generic overflow pattern (message order preserved)



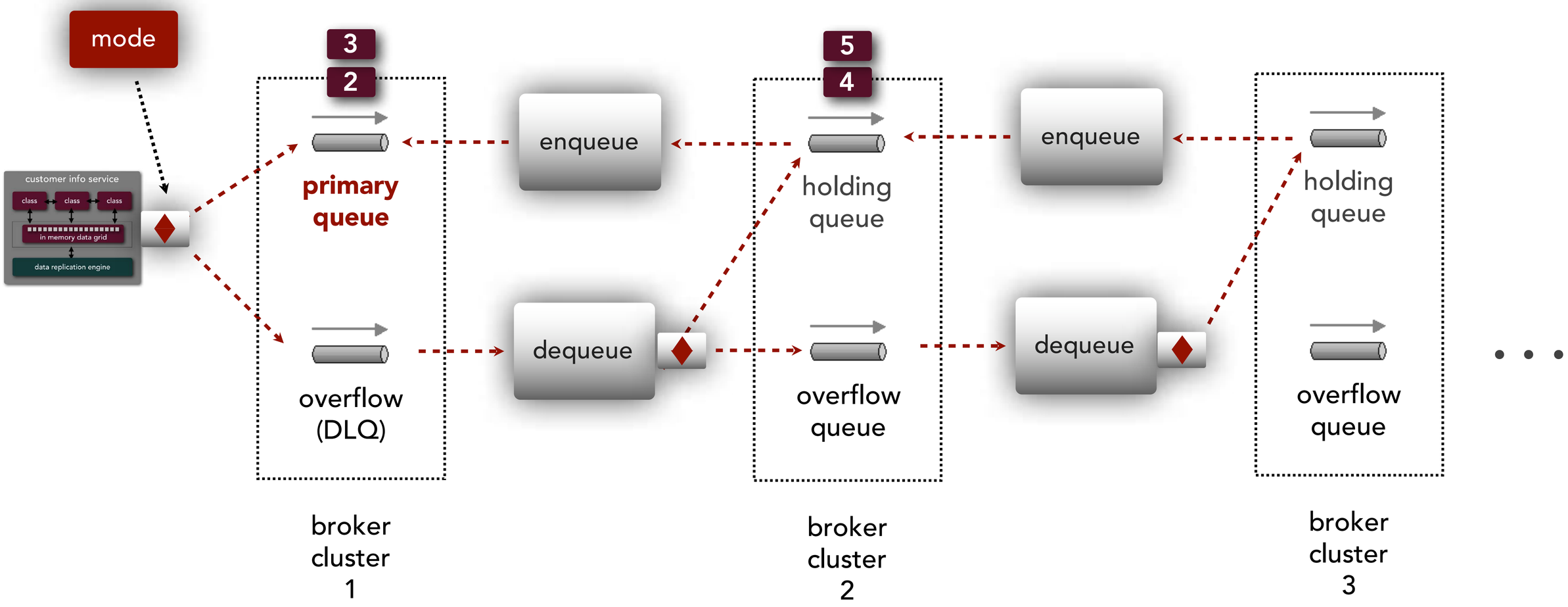
leveraging queues for back pressure

generic overflow pattern (message order preserved)



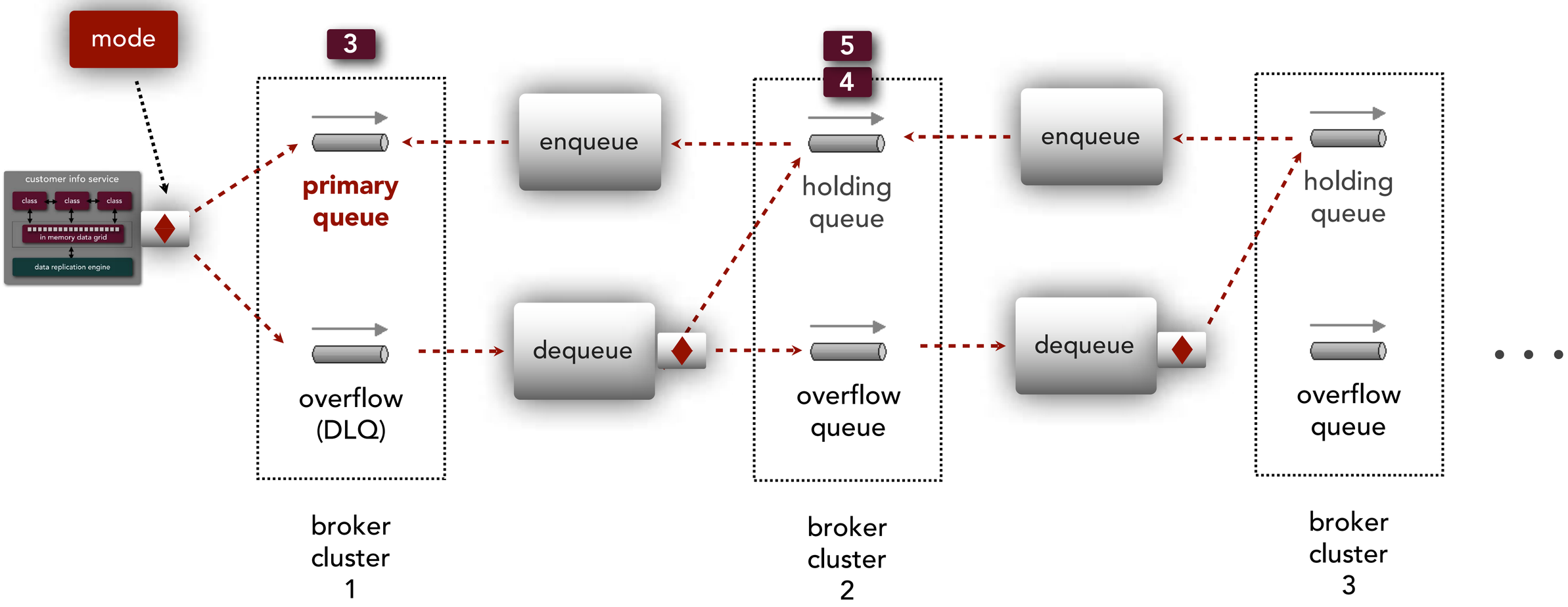
leveraging queues for back pressure

generic overflow pattern (message order preserved)



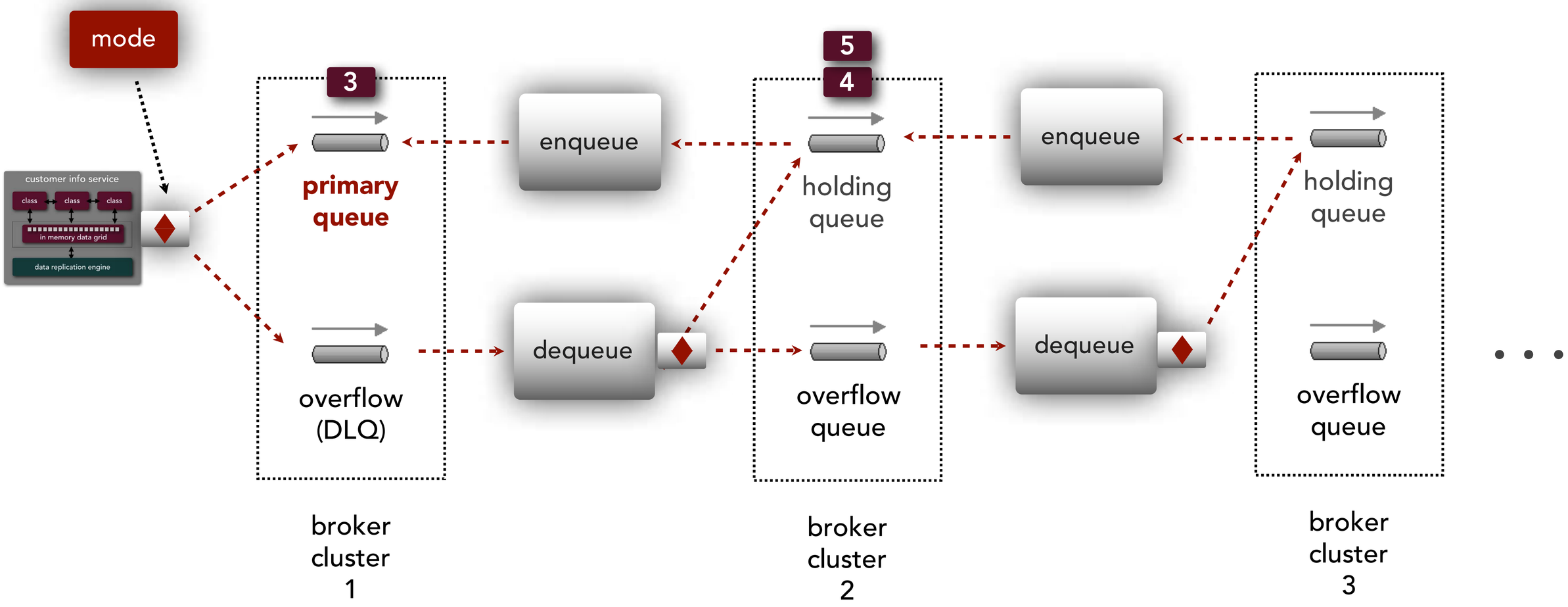
leveraging queues for back pressure

generic overflow pattern (message order preserved)



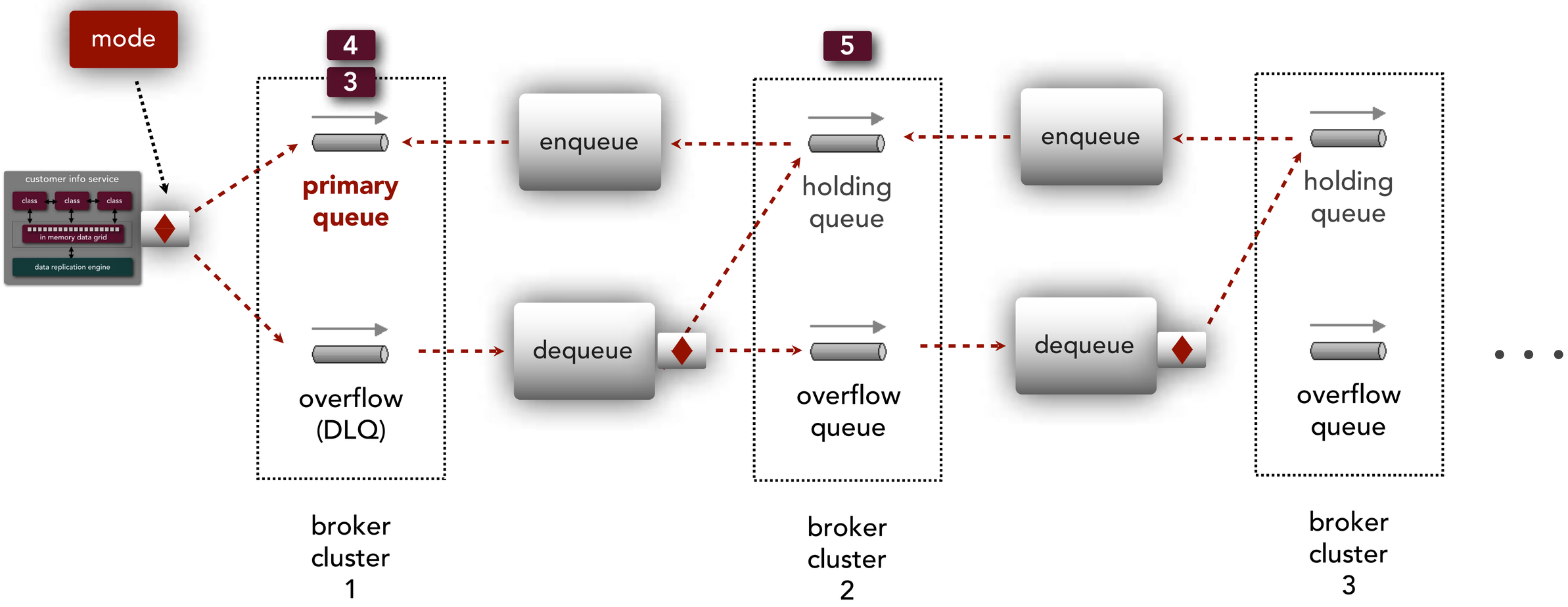
leveraging queues for back pressure

generic overflow pattern (message order preserved)



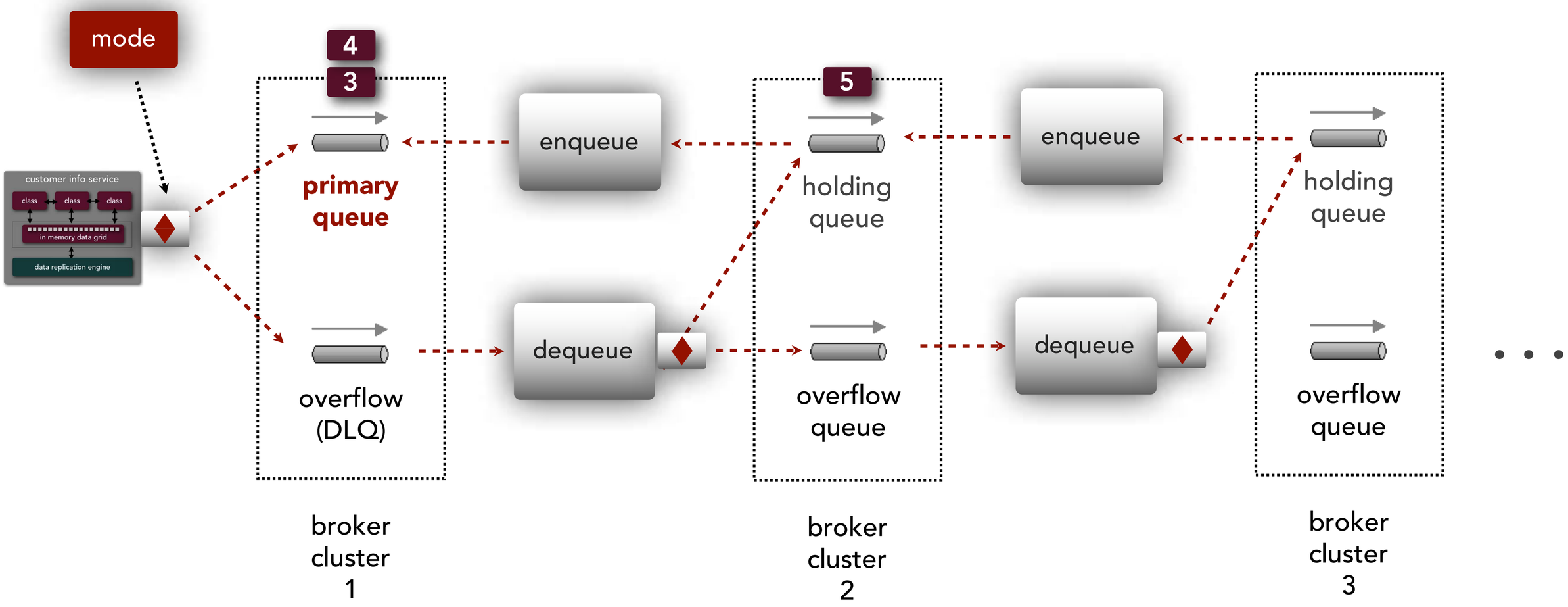
leveraging queues for back pressure

generic overflow pattern (message order preserved)



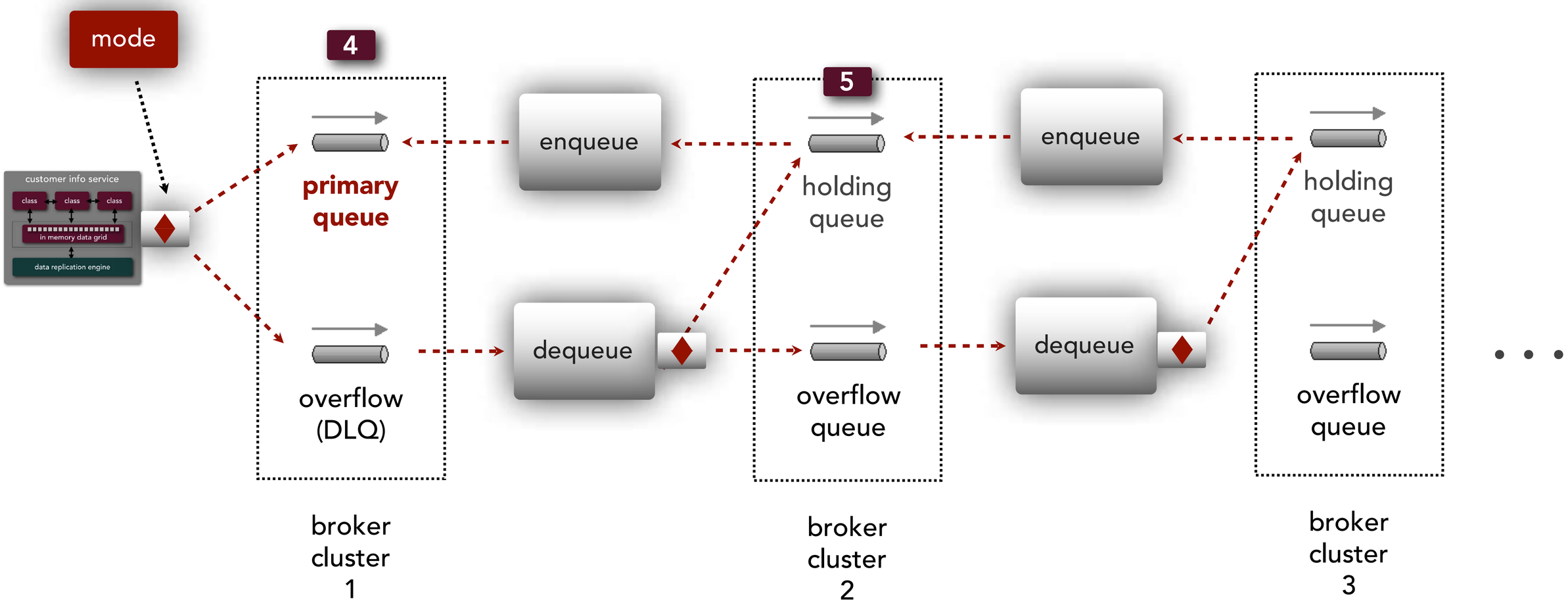
leveraging queues for back pressure

generic overflow pattern (message order preserved)



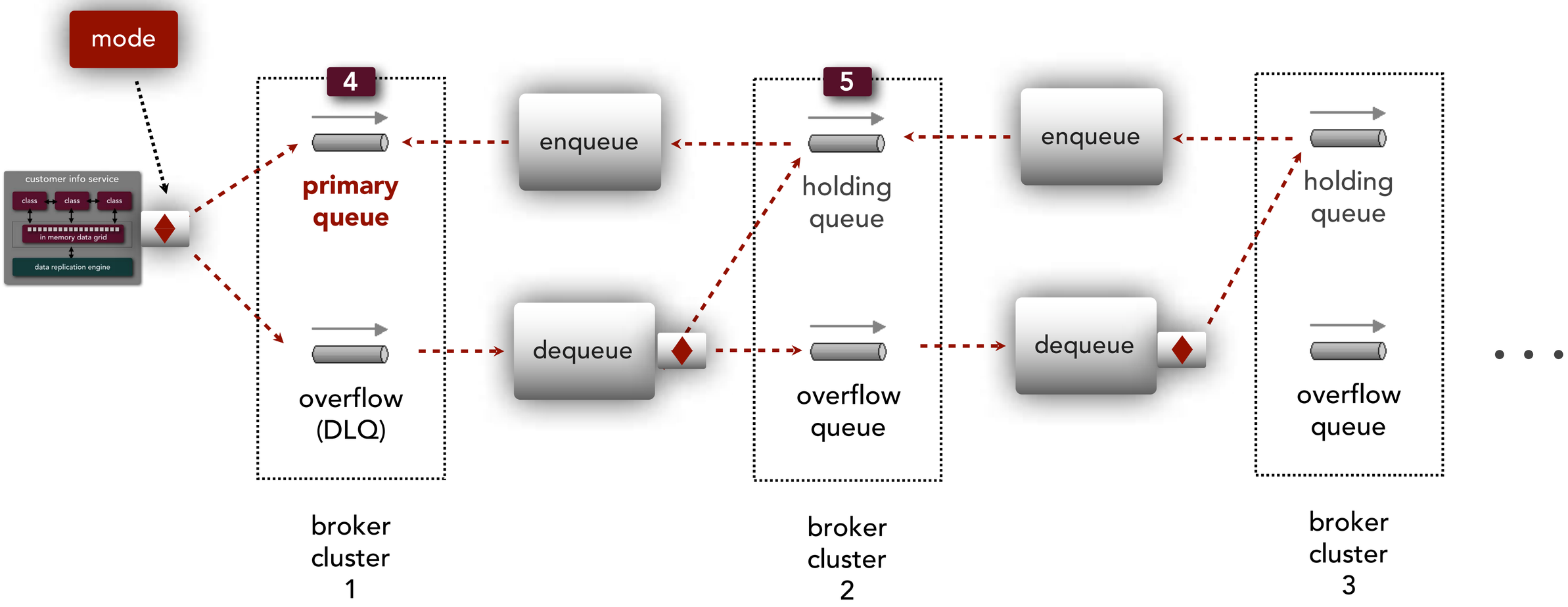
leveraging queues for back pressure

generic overflow pattern (message order preserved)



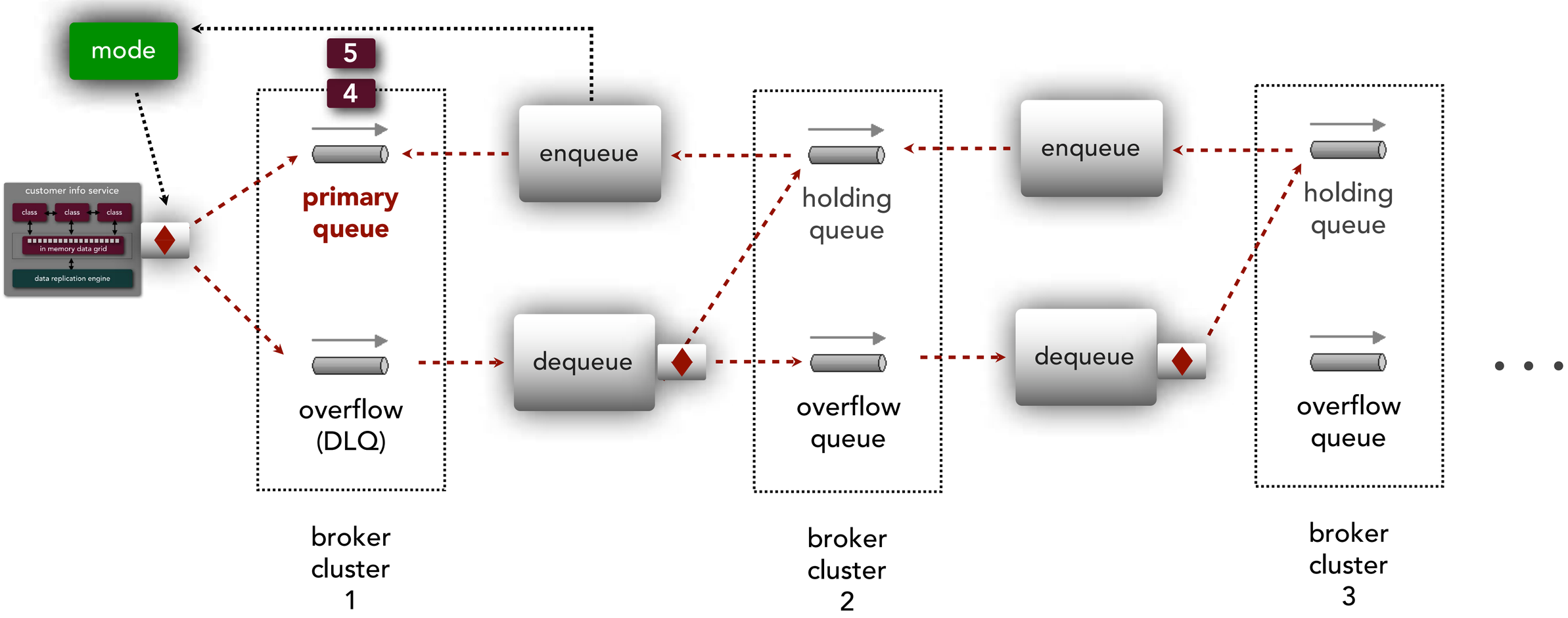
leveraging queues for back pressure

generic overflow pattern (message order preserved)



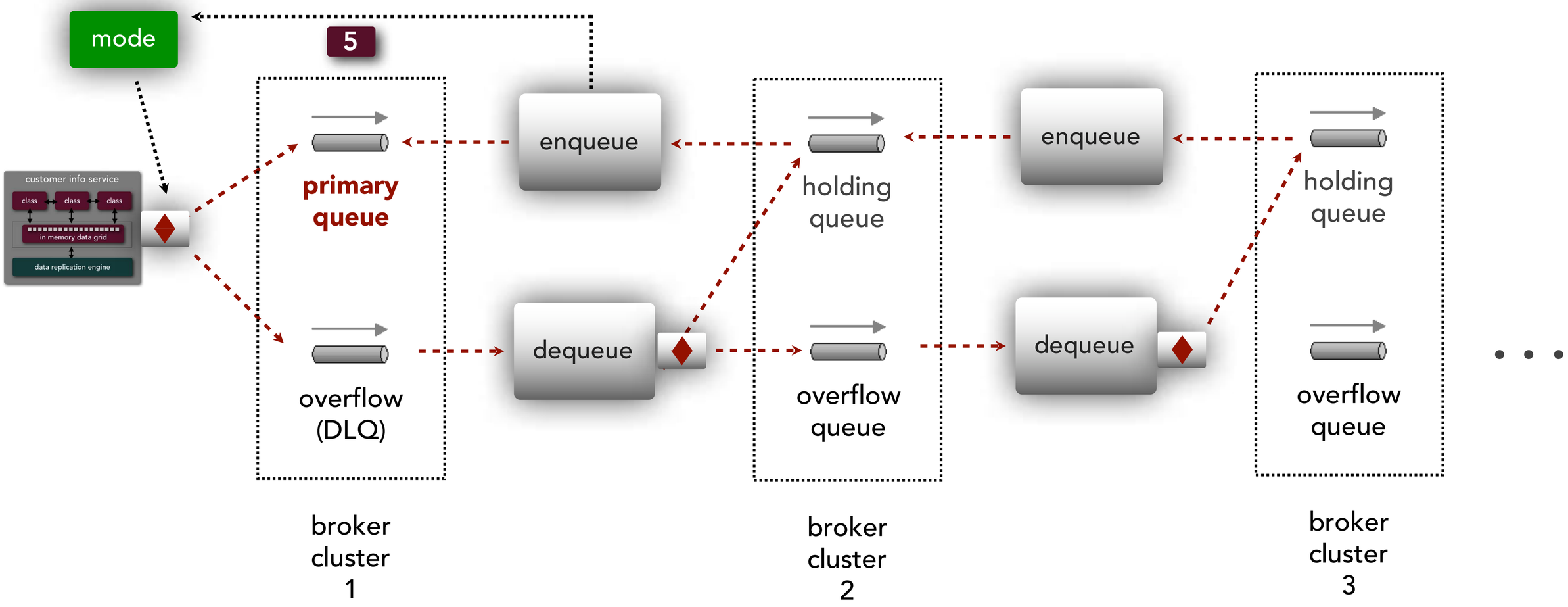
leveraging queues for back pressure

generic overflow pattern (message order preserved)



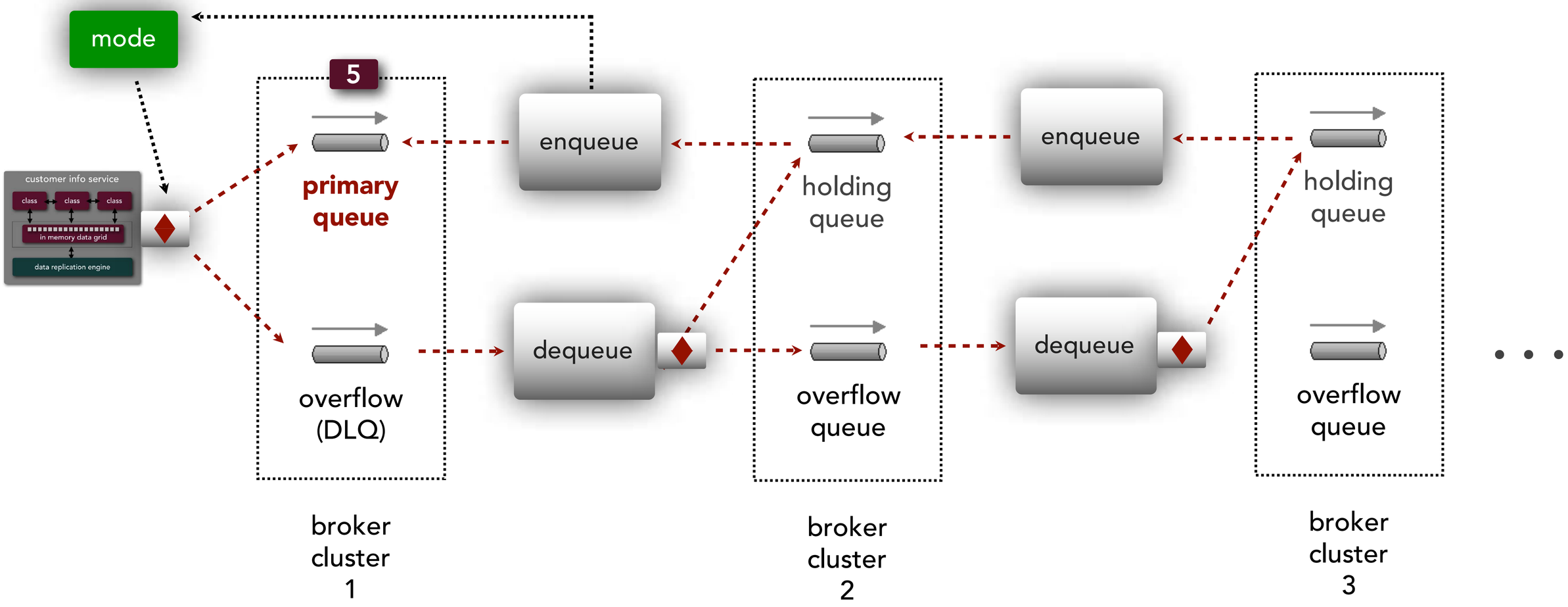
leveraging queues for back pressure

generic overflow pattern (message order preserved)



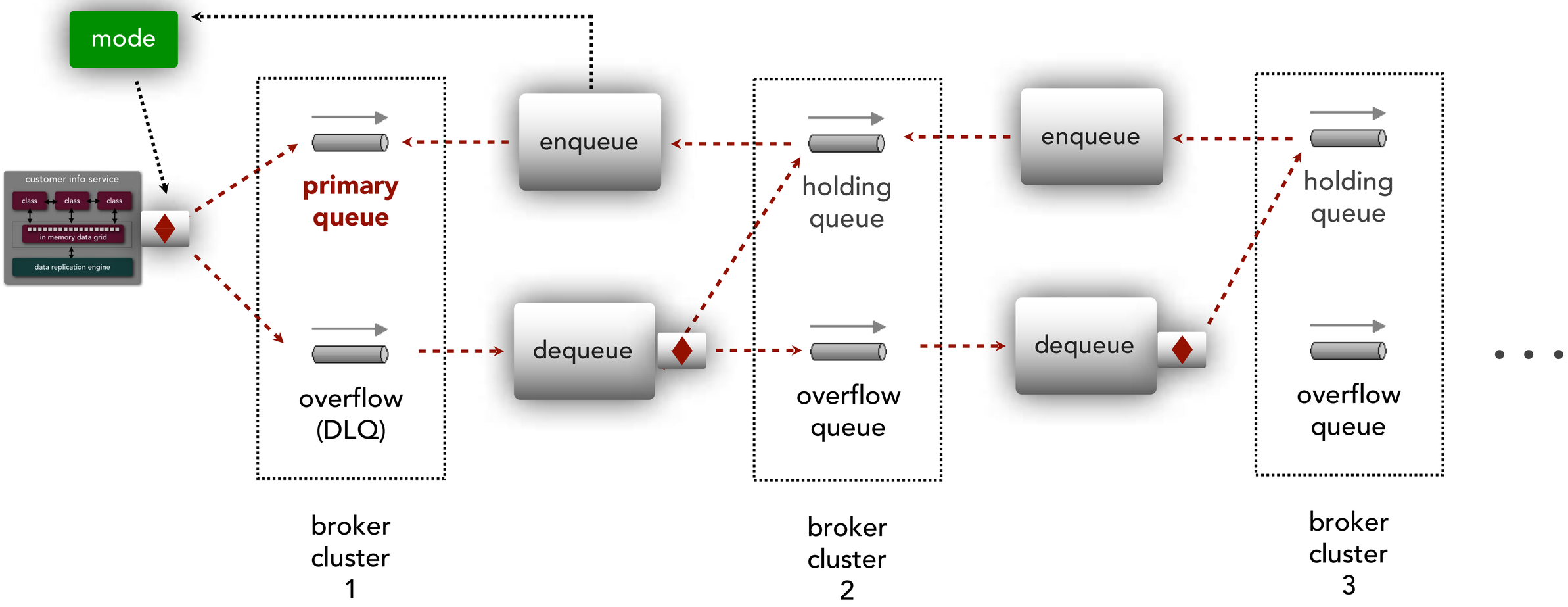
leveraging queues for back pressure

generic overflow pattern (message order preserved)



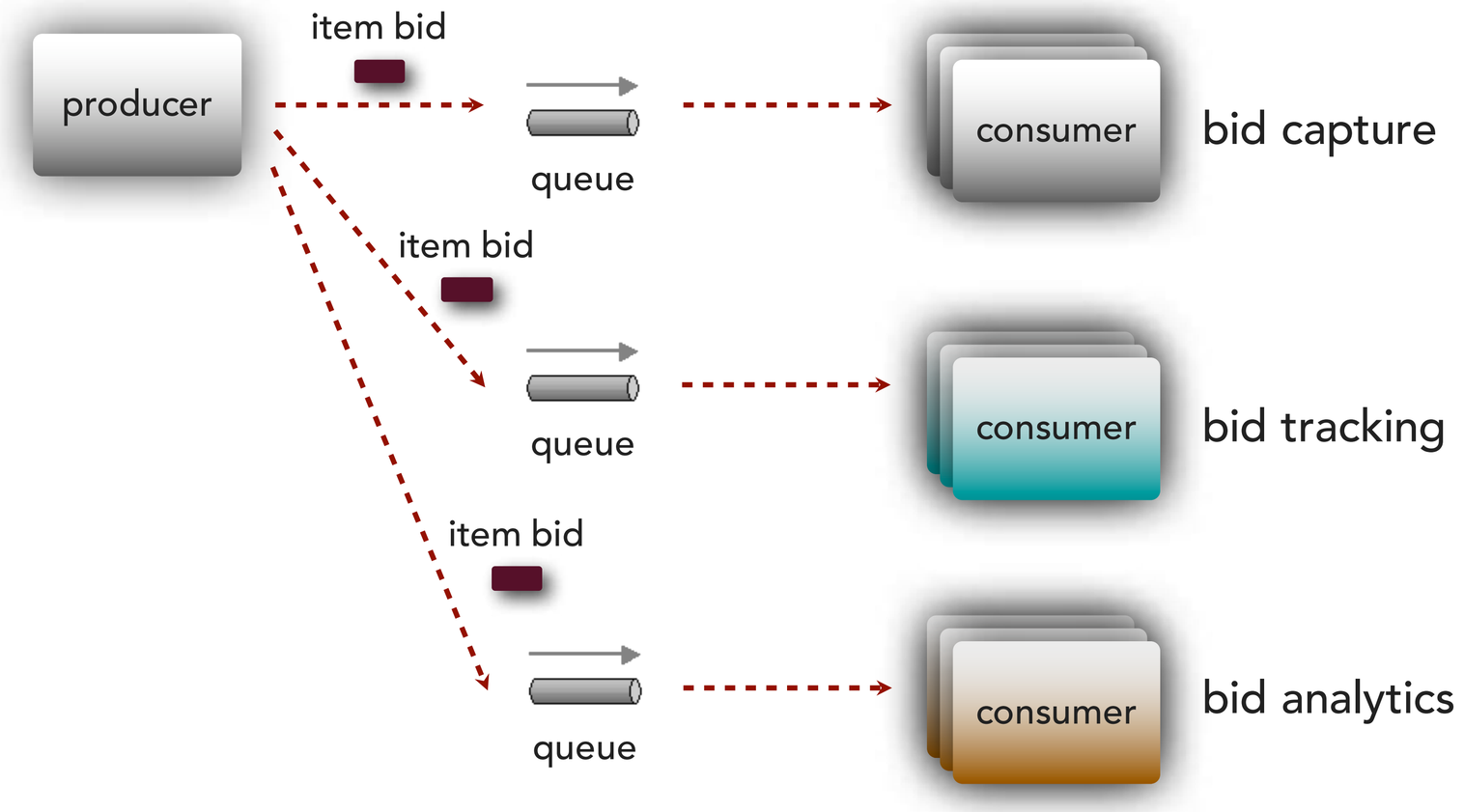
leveraging queues for back pressure

generic overflow pattern (message order preserved)

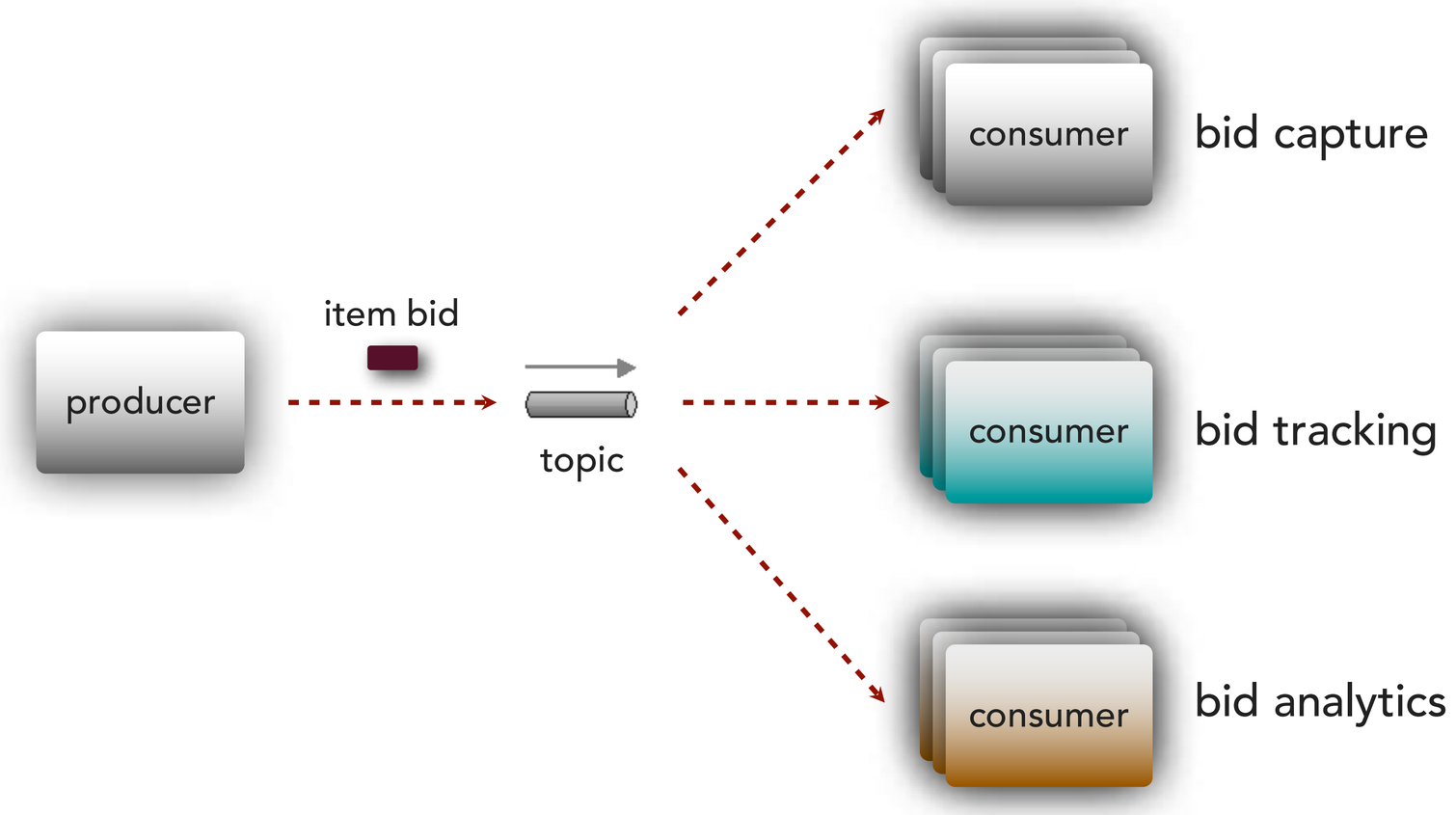


Leveraging Pub/Sub For Extensibility

leveraging pub/sub for extensibility

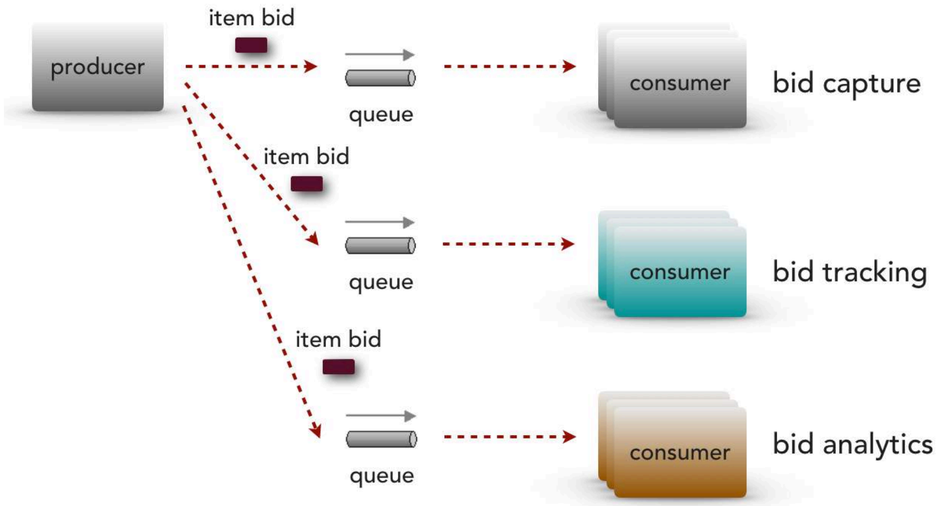


leveraging pub/sub for extensibility

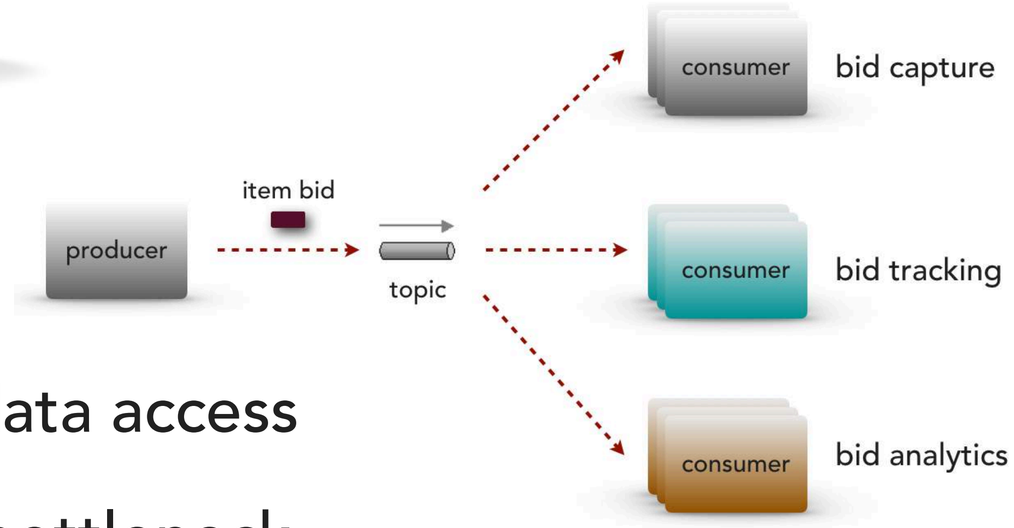


leveraging pub/sub for extensibility

point-to-point



publish-and-subscribe



1: security / data access

2: capacity / bottleneck

3: monitoring / queue depth

4: auto-scaling capabilities

Orchestration vs. Choreography

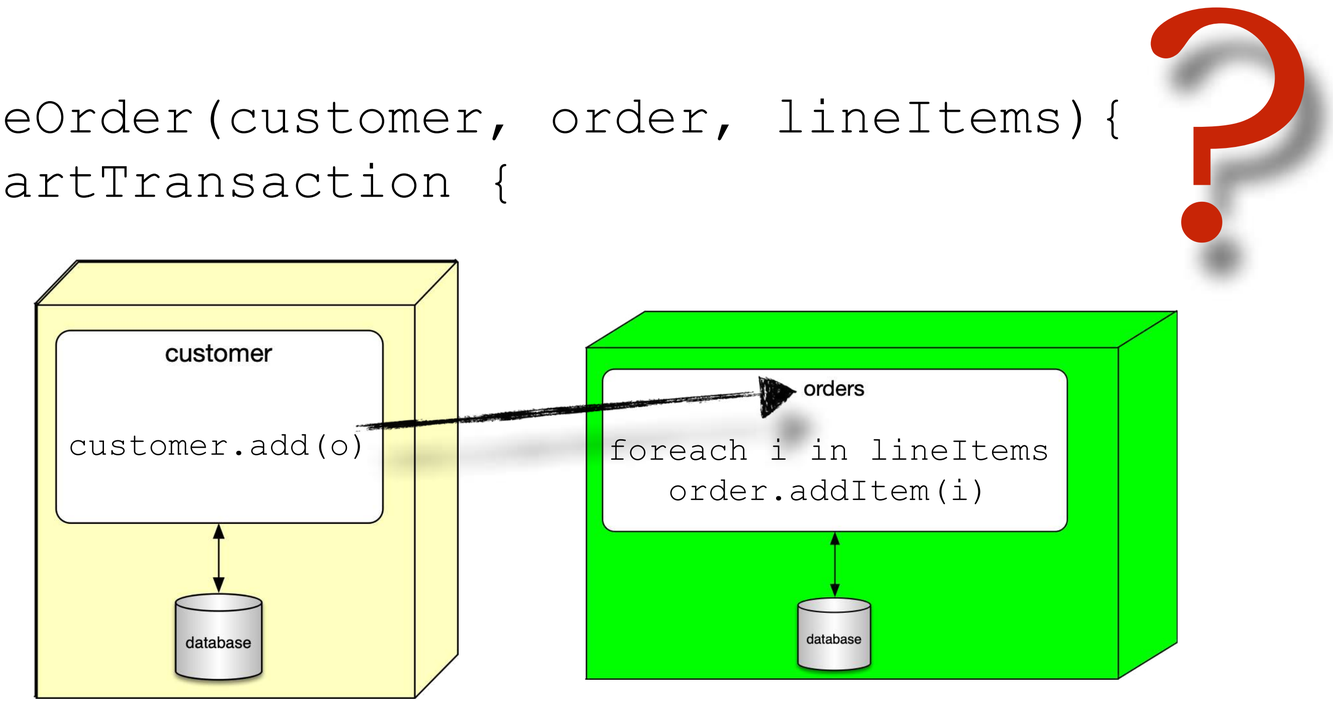
print exclude 2

the origin of orchestration

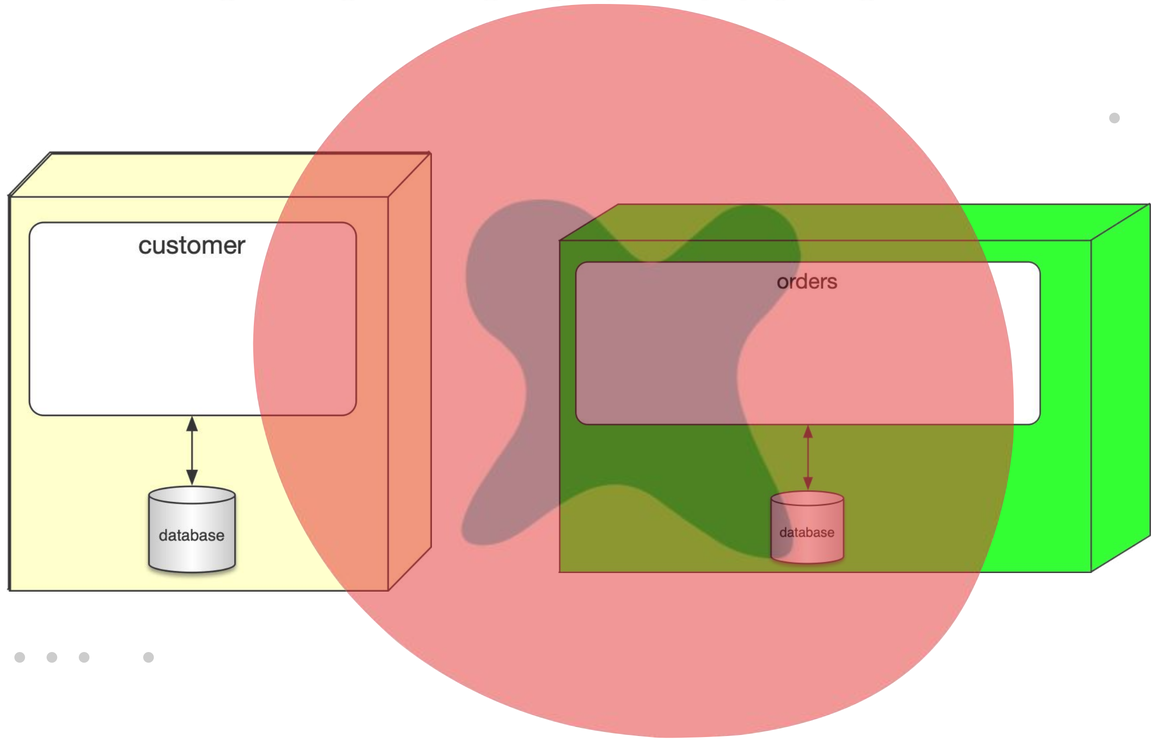
```
placeOrder(customer, order, lineItems){
  startTransaction {

    foreach i in lineItems
       order.addItem(i)
     customer.add(o)
  }
}
```

the origin of orchestration

```
placeOrder(customer, order, lineItems){
  startTransaction {

  }
}
```

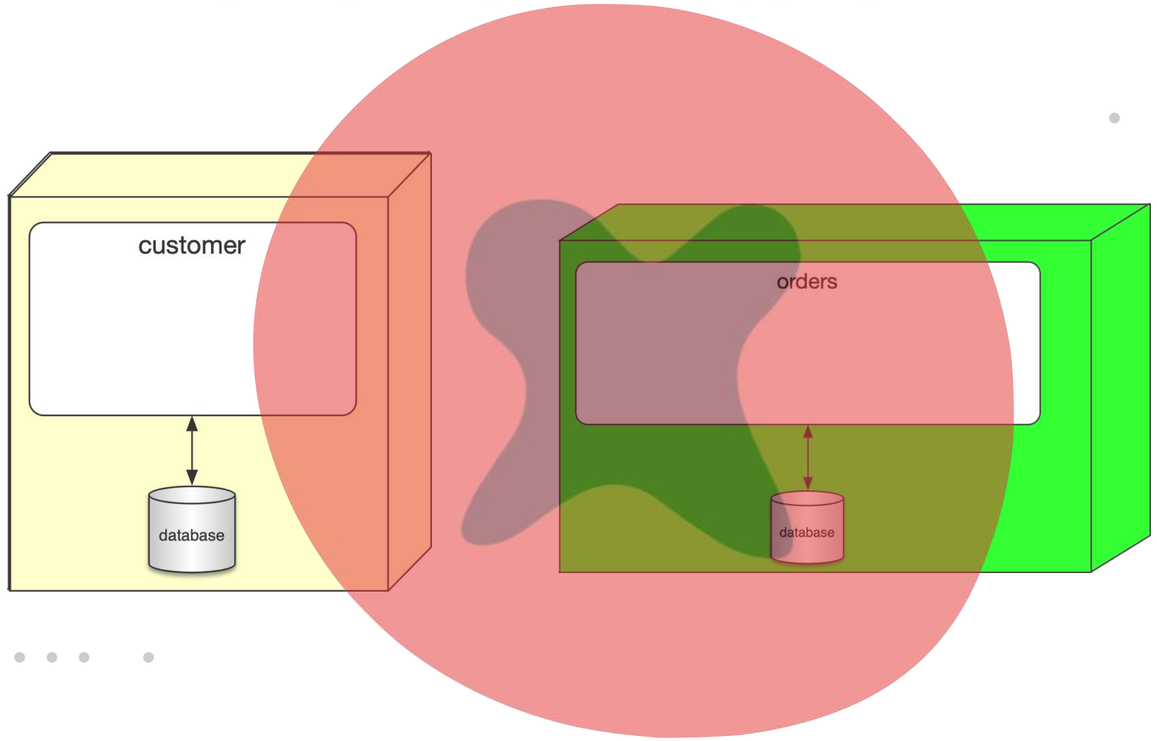


orchestration in microservices



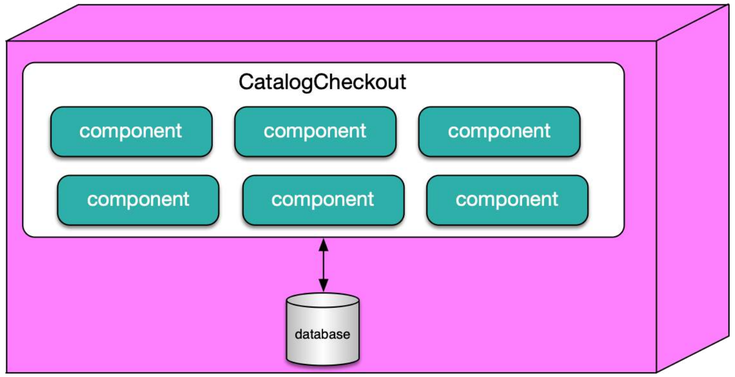
The best orchestration is no orchestration.

orchestration in microservices

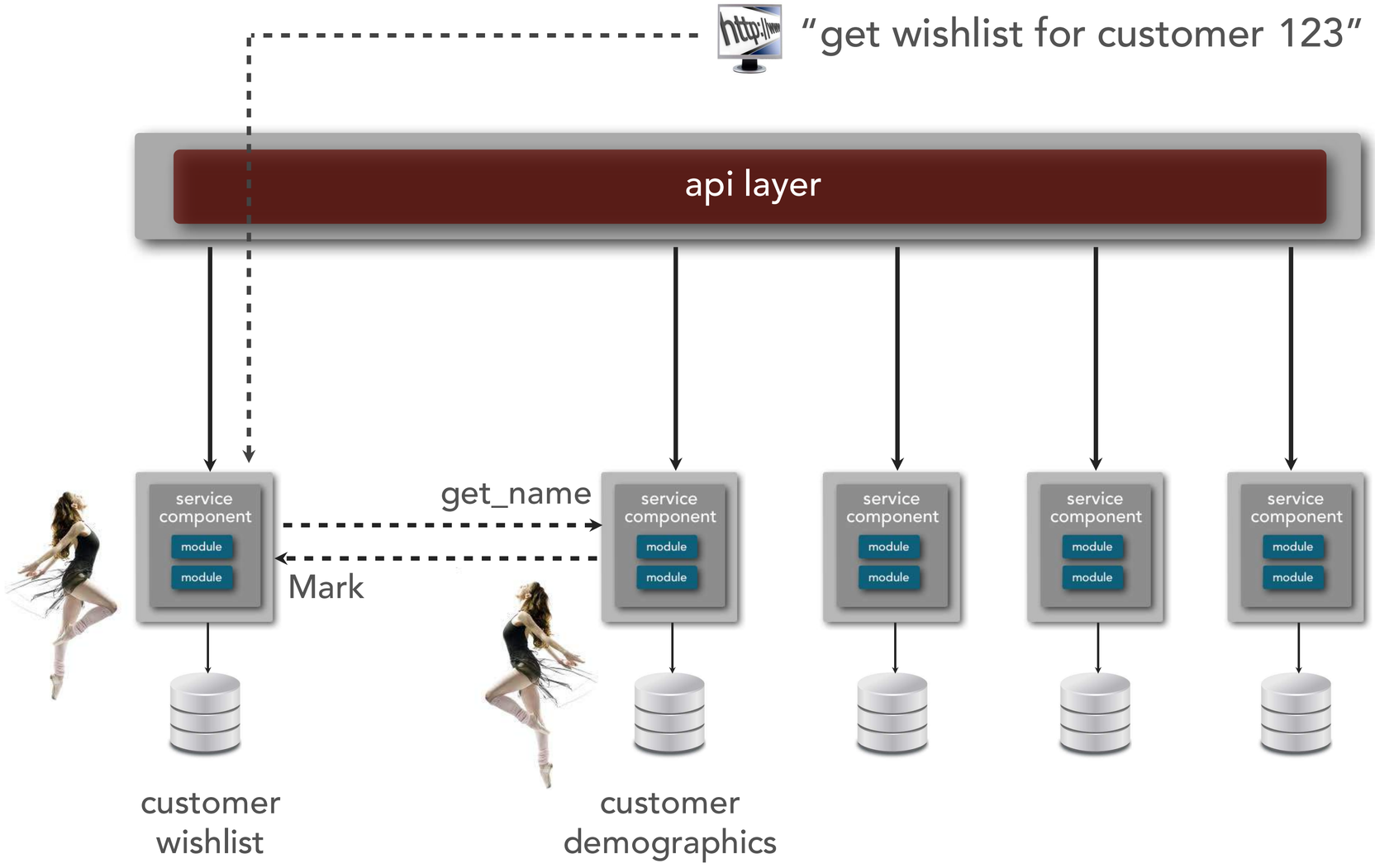


The best orchestration is no orchestration.

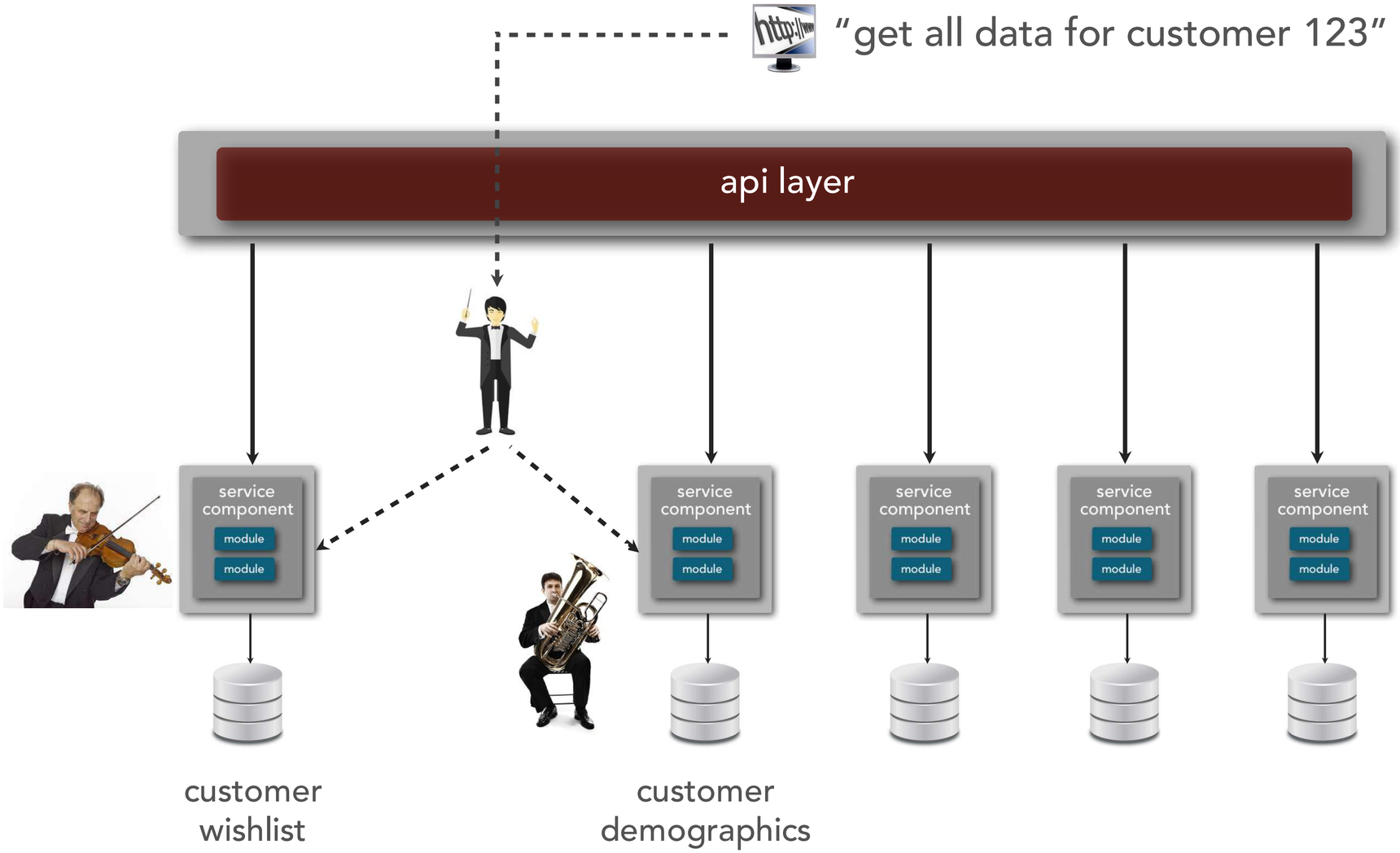
Fix your granularity instead.



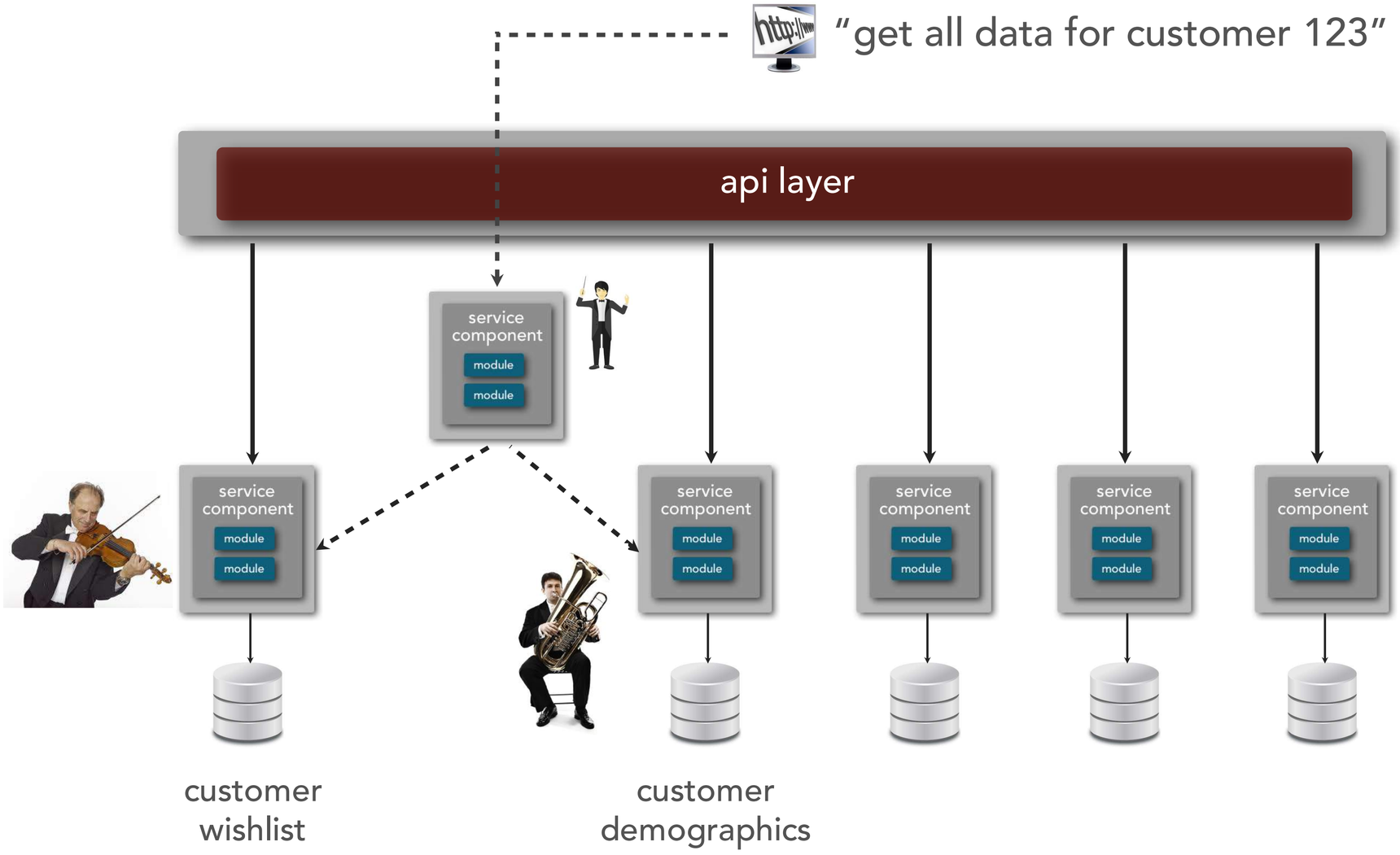
orchestration vs. choreography



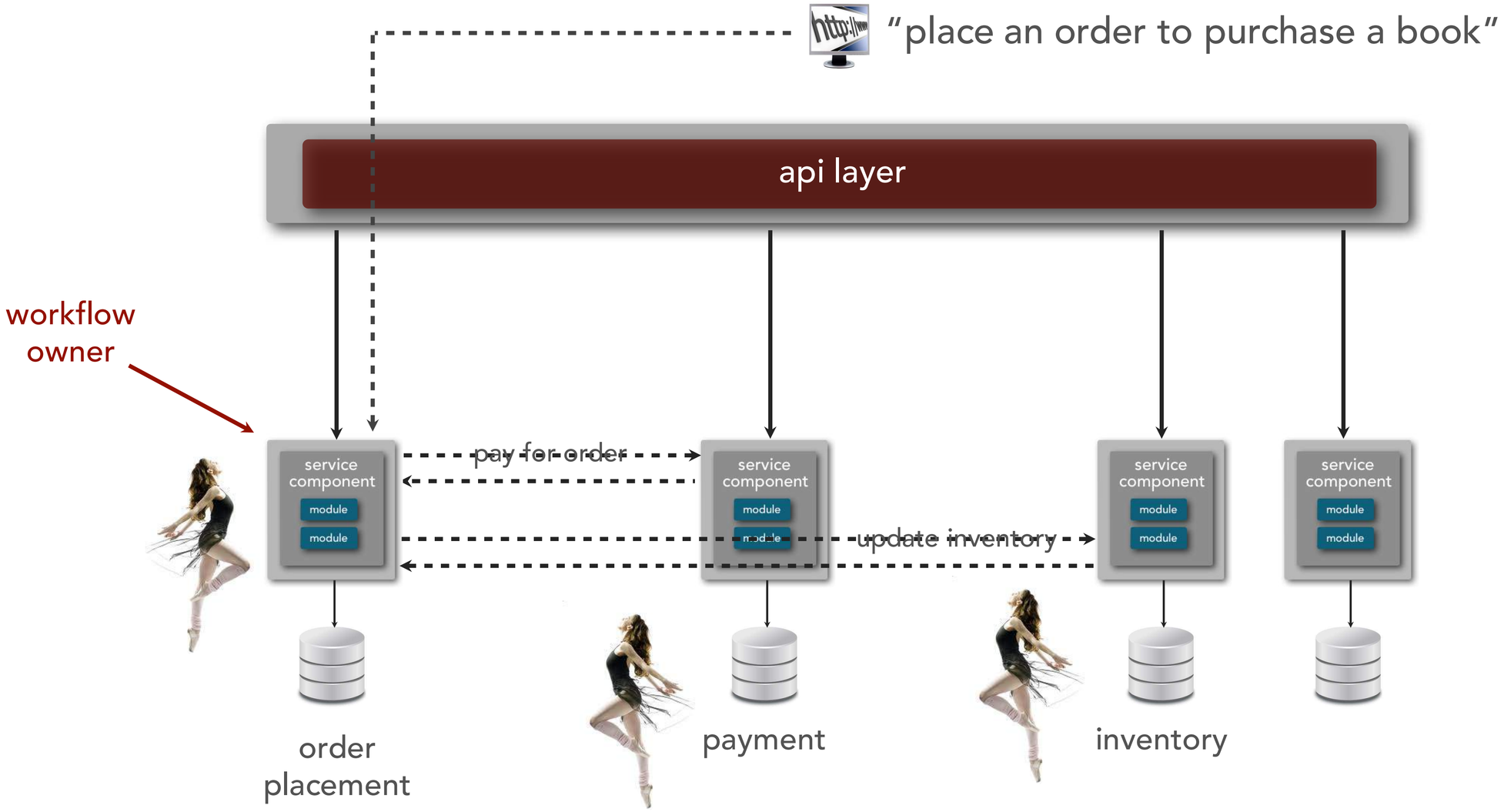
orchestration vs. choreography



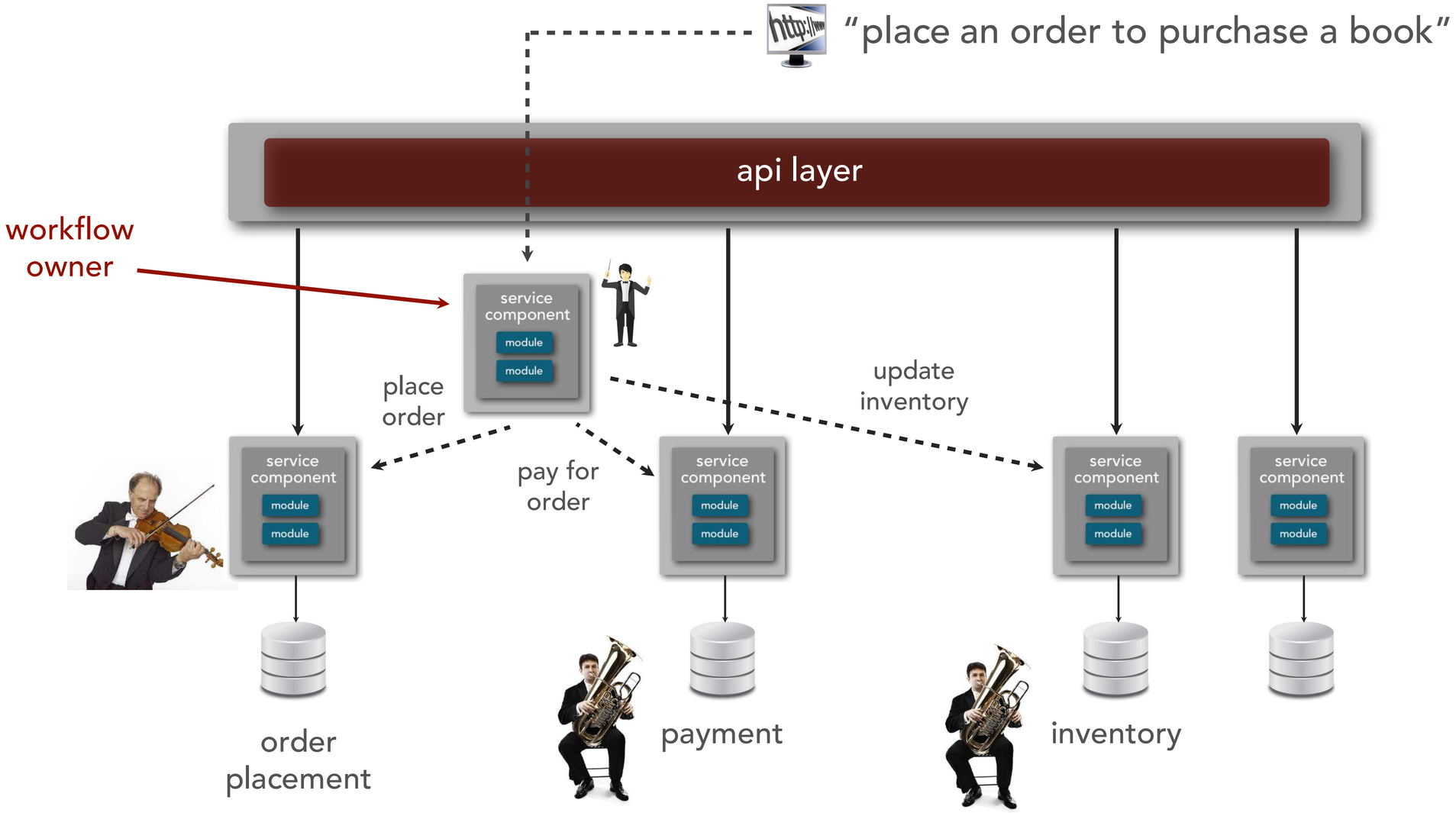
orchestration vs. choreography



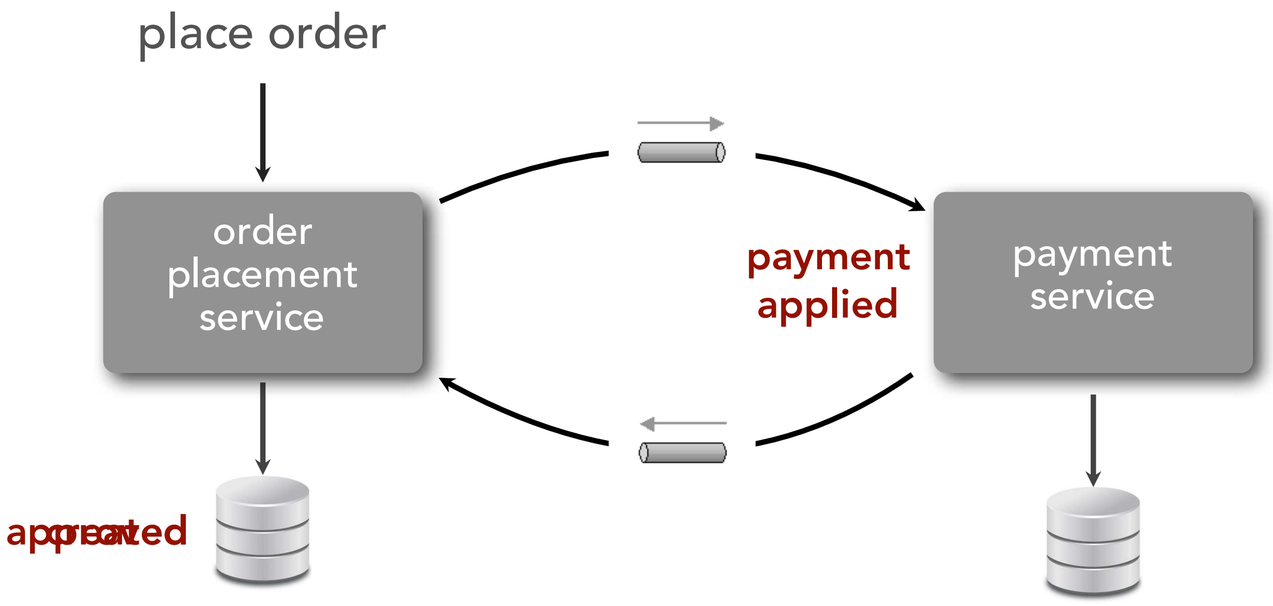
orchestration vs. choreography



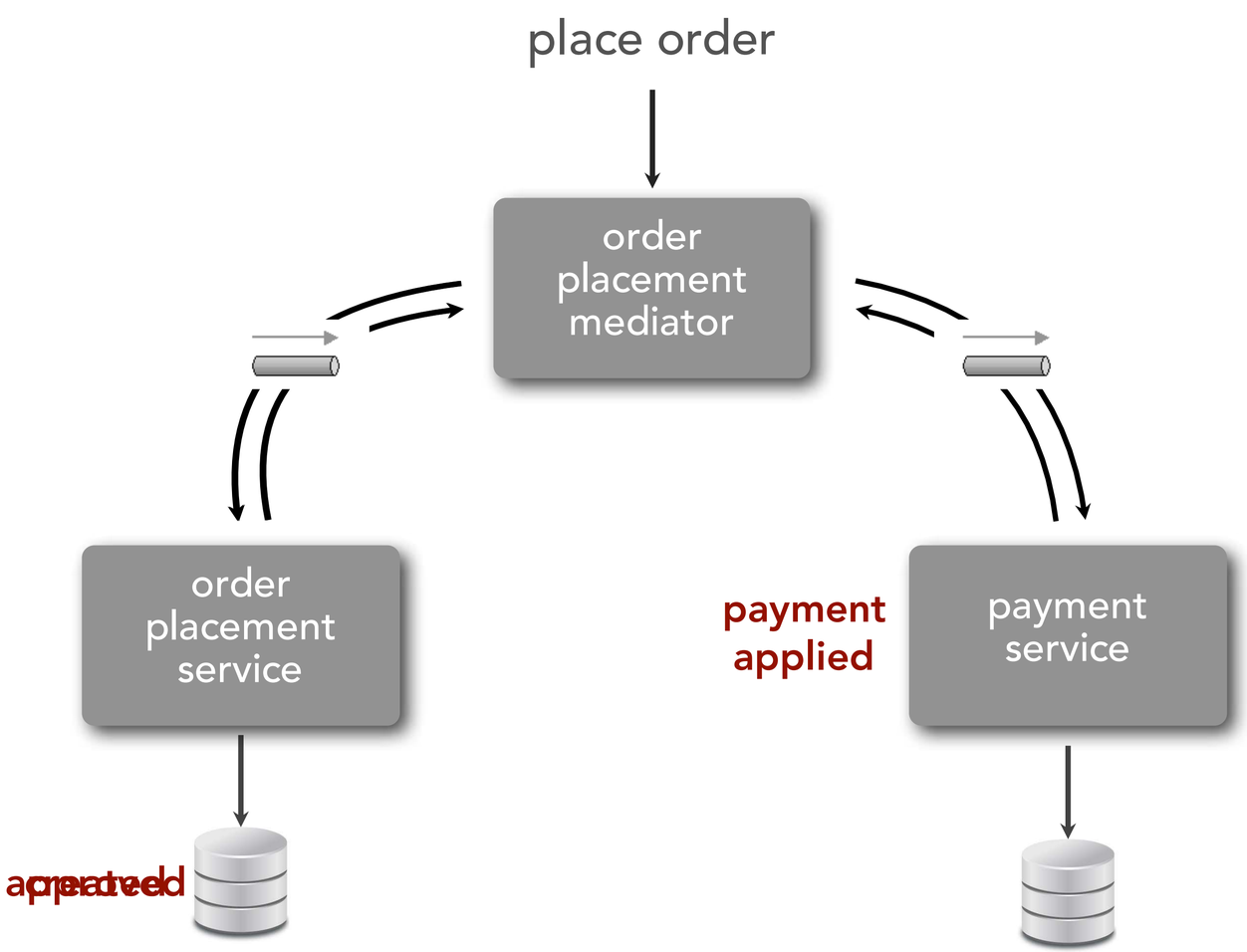
orchestration vs. choreography



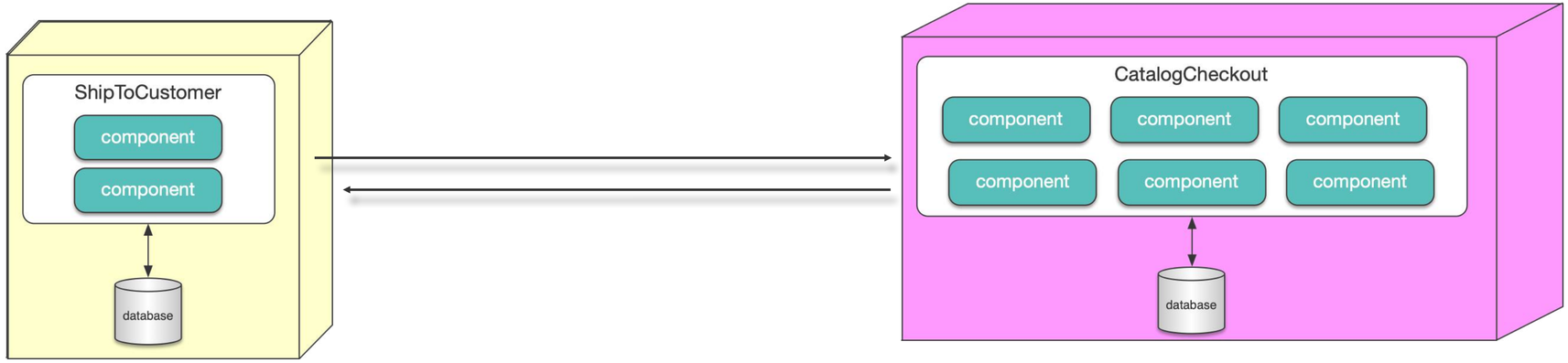
orchestration vs. choreography



orchestration vs. choreography



orchestration in microservices



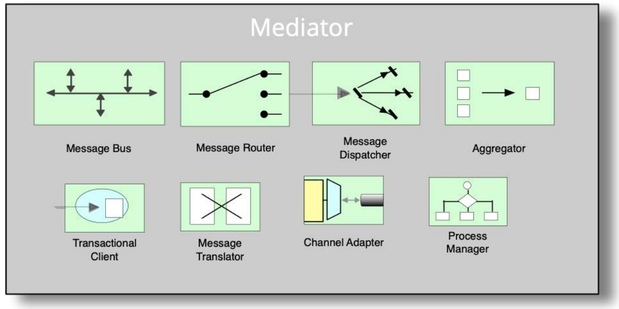
Synchronous calls expand the scope of quanta.

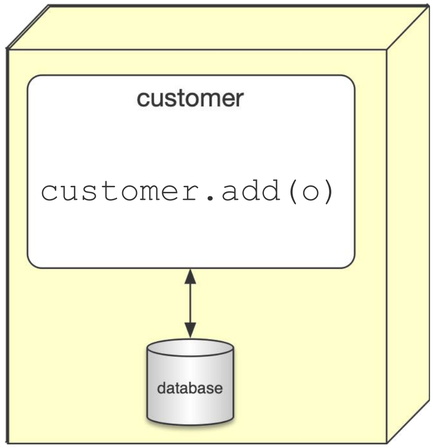
sagas

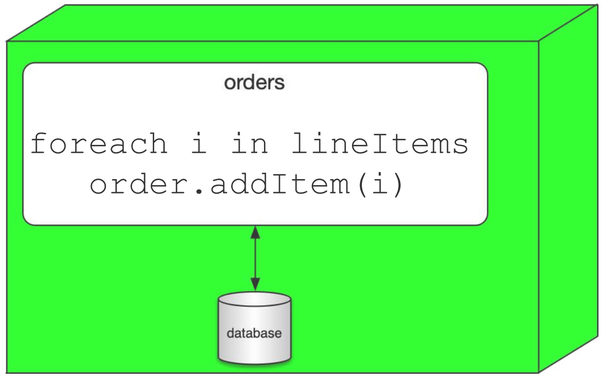
```
placeOrder(customer, order, lineItems){
  startTransaction {

    foreach i in lineItems
       order.addItem(i)
     customer.add(o)
  }
}
```

sagas

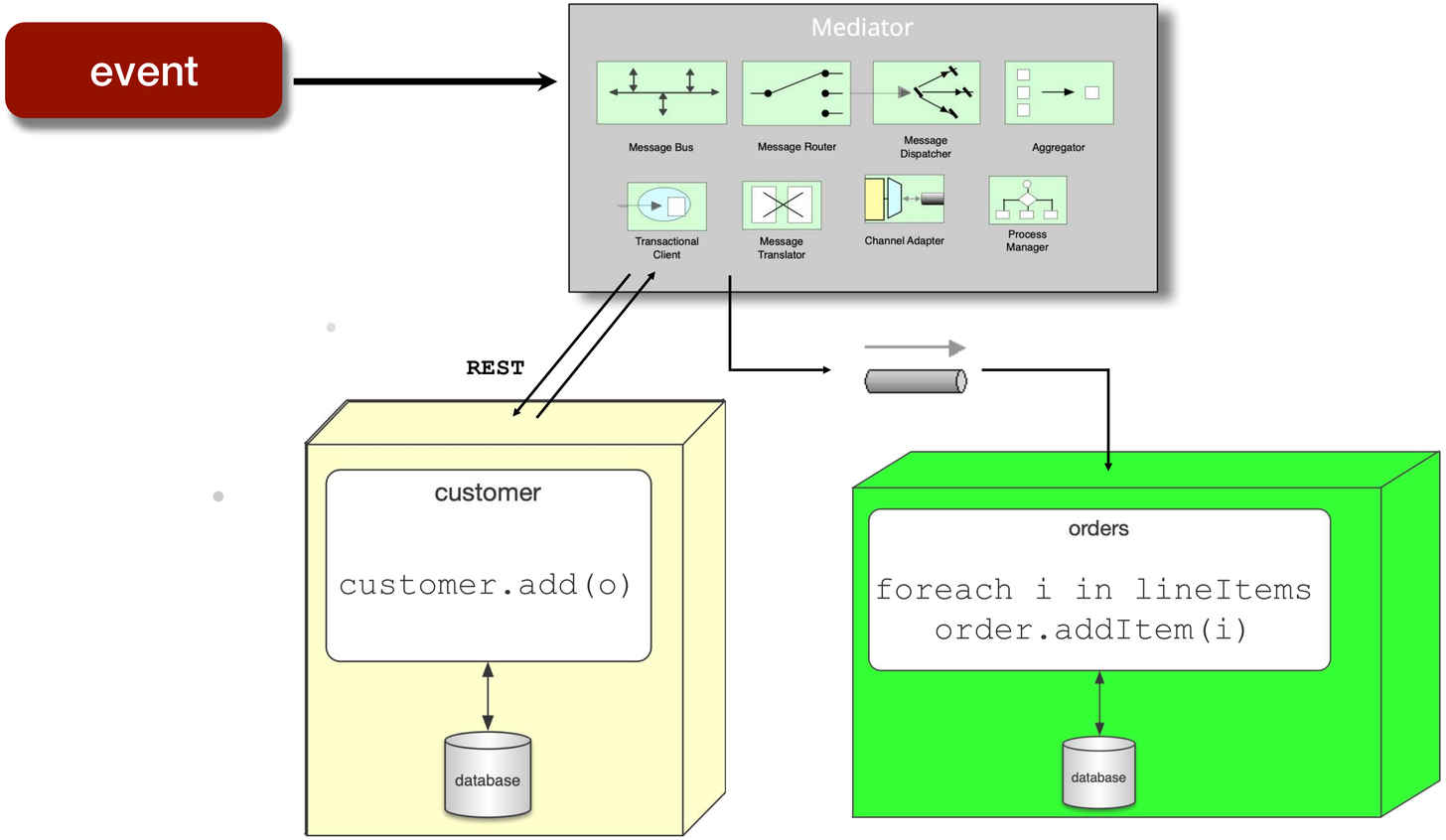
mediator
event driven architecture





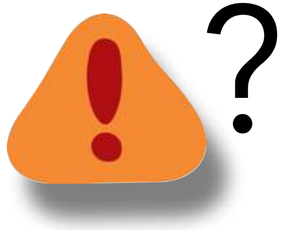


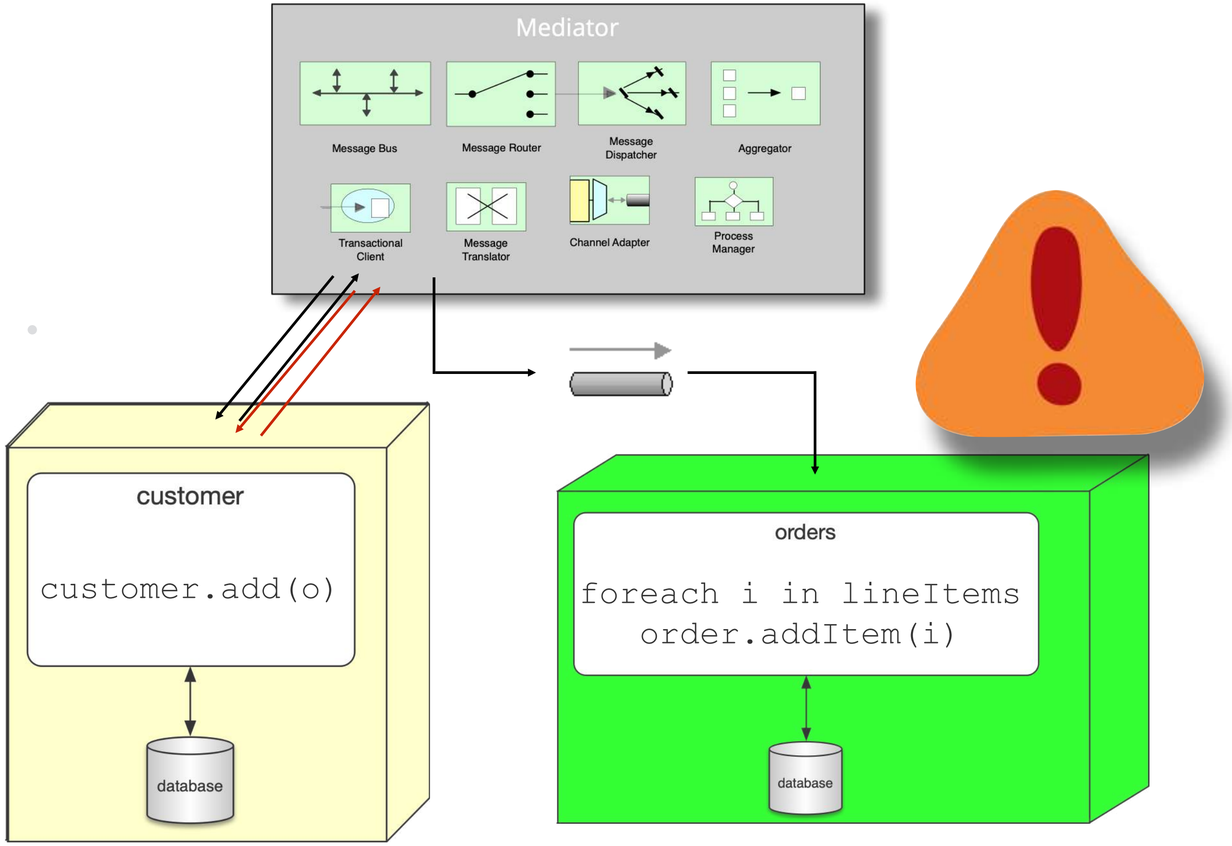
orchestration via sagas



orchestration via sagas

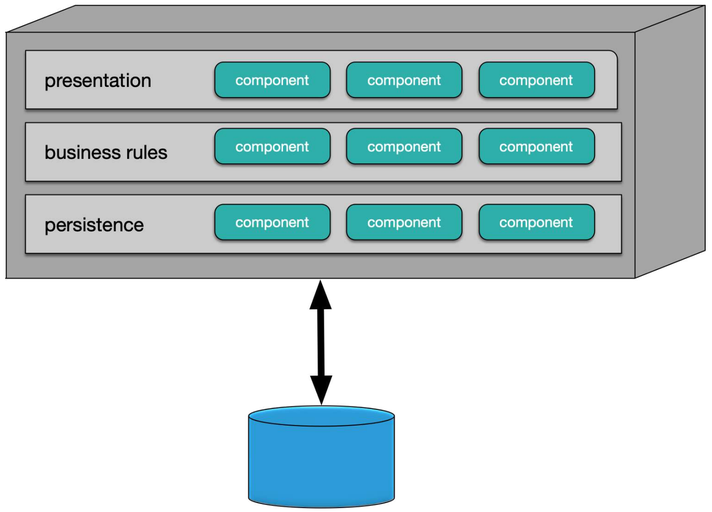
Error Conditions?

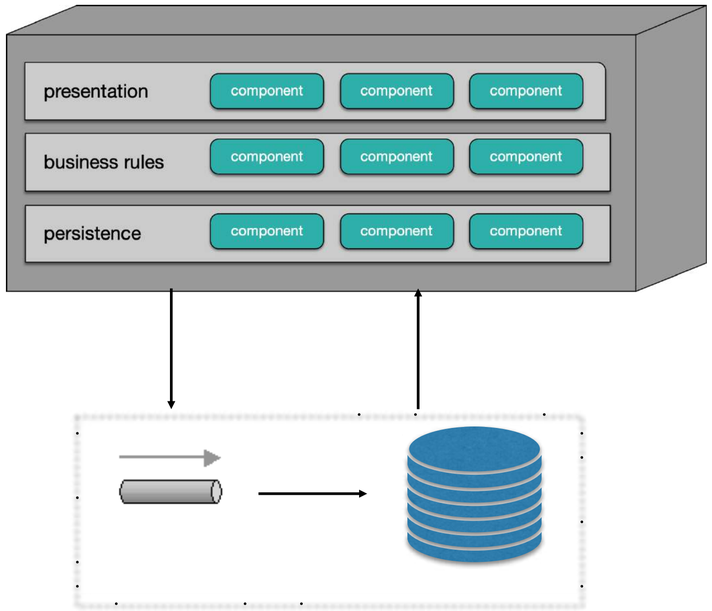




compensating transactions

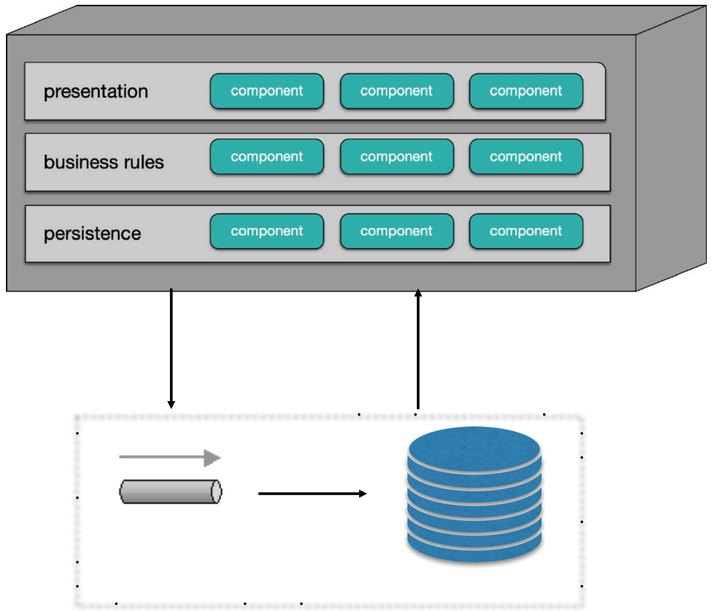
Event Sourcing



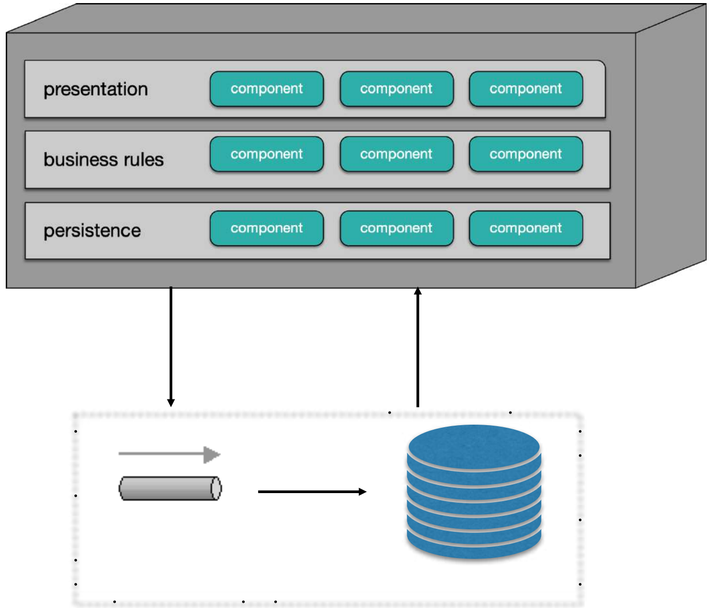


Event Sourcing

Online order	Traditional Storage	Event Sourcing
Select pizza	Order 123: Selected pizza	T1: Selected pizza
Select cola	Order 123: Selected pizza, cola	T1: Selected pizza T2: Selected cola
Select ice cream	Order 123: Selected pizza, cola, ice cream	T1: Selected pizza T2: Selected cola T3: Selected ice cream
Deselect ice cream	Order 123: Selected pizza, cola	T1: Selected pizza T2: Selected cola T3: Selected ice cream T4: Deselected ice cream
Confirm order	Order 123: Ordered pizza, cola	T1: Selected pizza T2: Selected cola T3: Selected ice cream T4: Deselected ice cream T5: Confirmed order

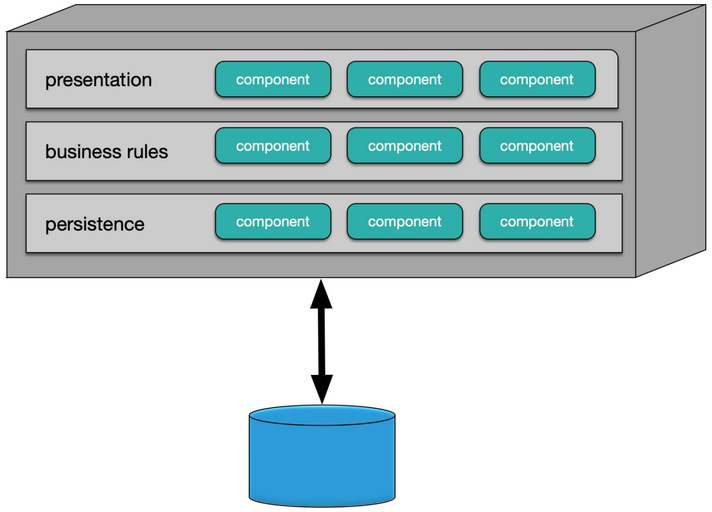


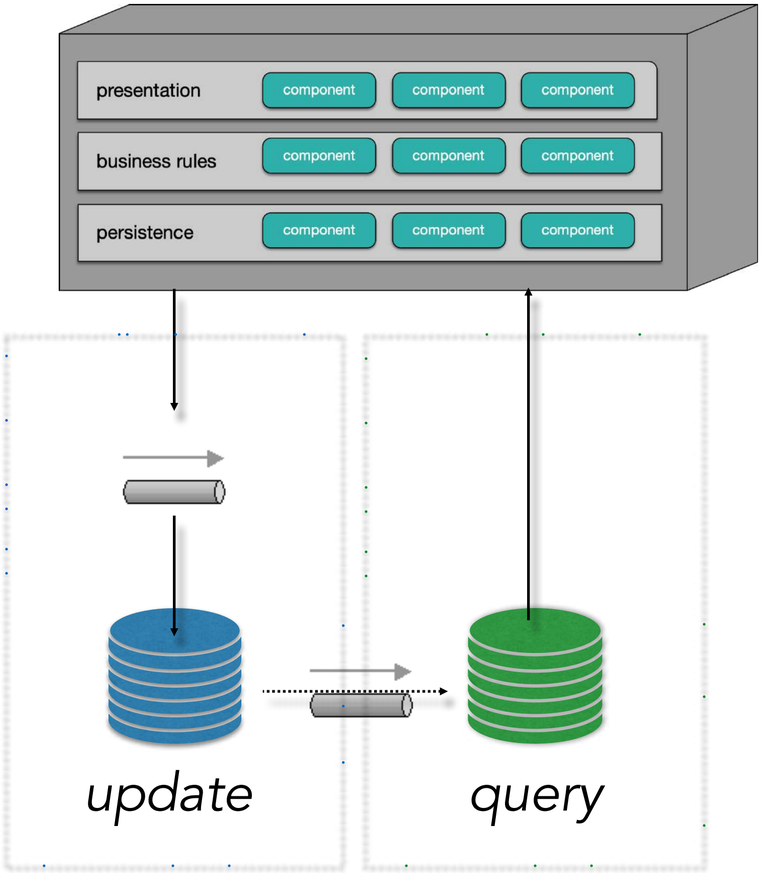
Event Sourcing



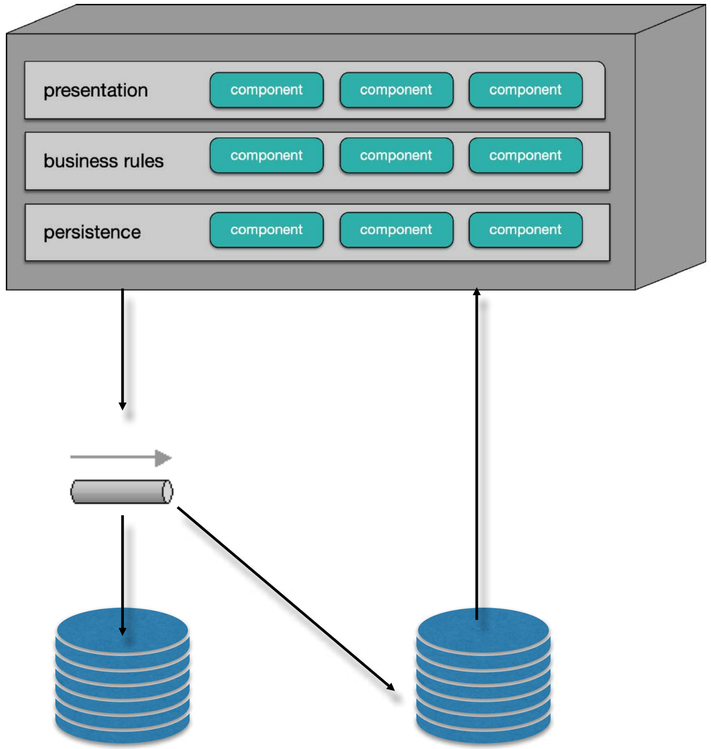
- Complete rebuild
- Temporal Query
- Event Replay for error correction/testing
- Upcasting

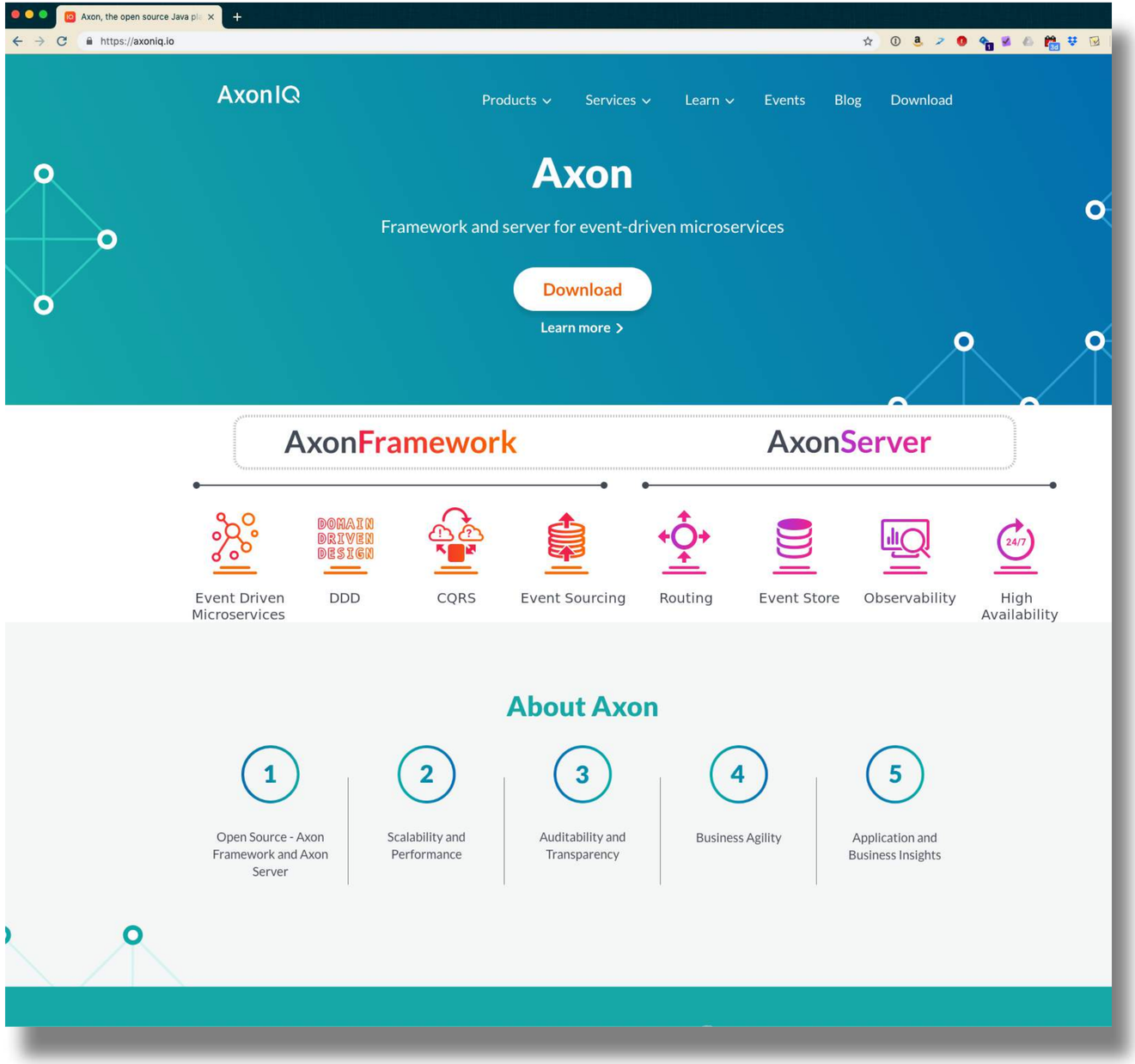
CQRS


presentation
component
component
component
business rules
component
component
component
persistence
component
component
component


presentation
component
component
component
business rules
component
component
component
persistence
component
component
component
update
query

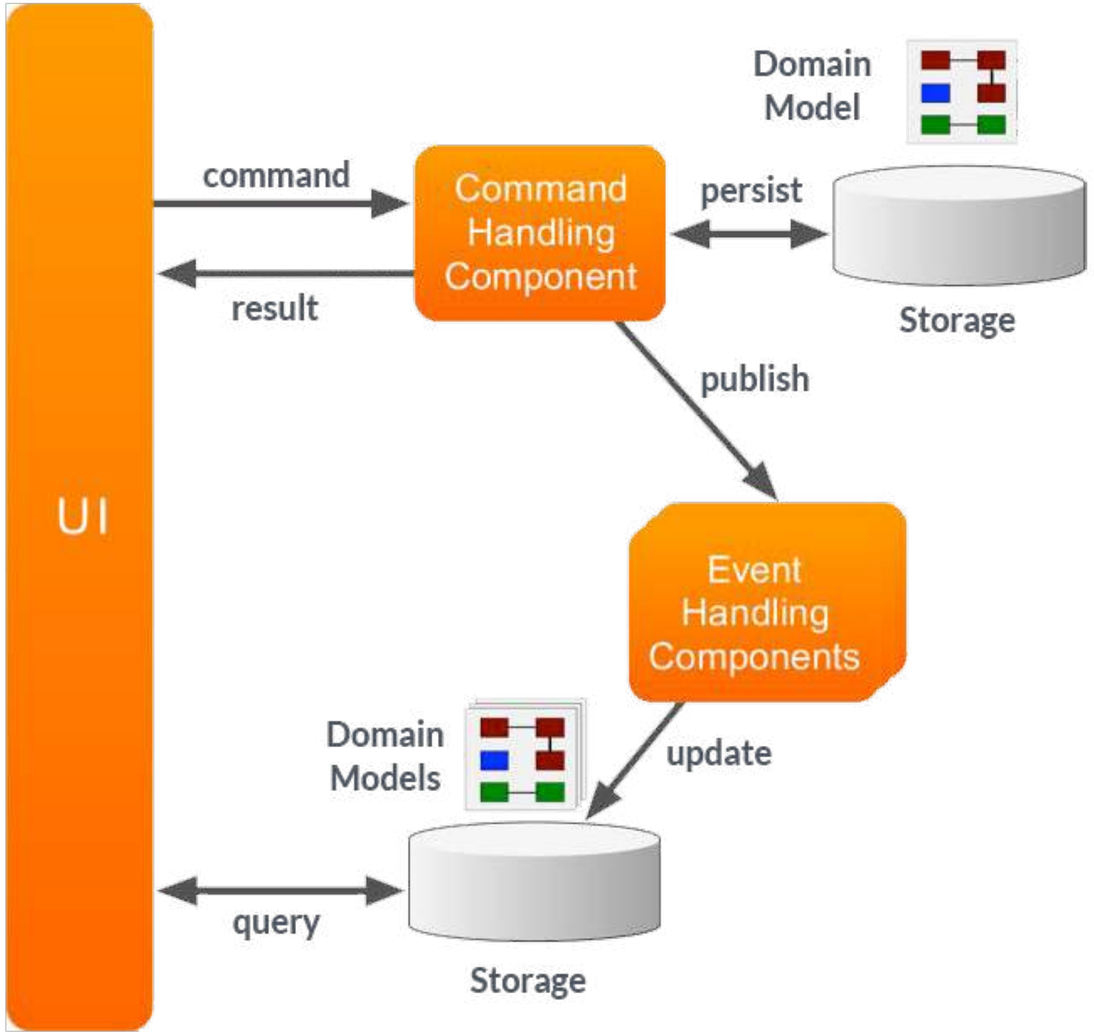
CQRS

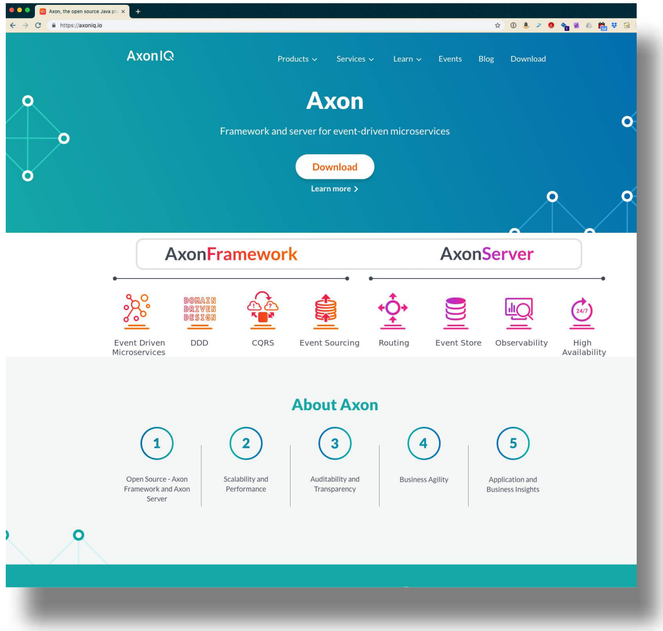

presentation
component
component
component
business rules
component
component
component
persistence
component
component
component



https://axoniq.io/

CQRS


UI
command
result
Command Handling Component
persist
Domain Model
Storage
publish
Event Handling Components
update
Domain Models
query
Storage


AxonIQ
Products
Services
Learn
Events
Blog
Download
Axon
Framework and server for event-driven microservices
Download
Learn more
AxonFramework
AxonServer
Event Driven Microservices
DDD
CQRS
Event Sourcing
Routing
Event Store
Observability
High Availability
About Axon
1
Open Source - Axon Framework and Axon Server
2
Scalability and Performance
3
Auditability and Transparency
4
Business Agility
5
Application and Business Insights

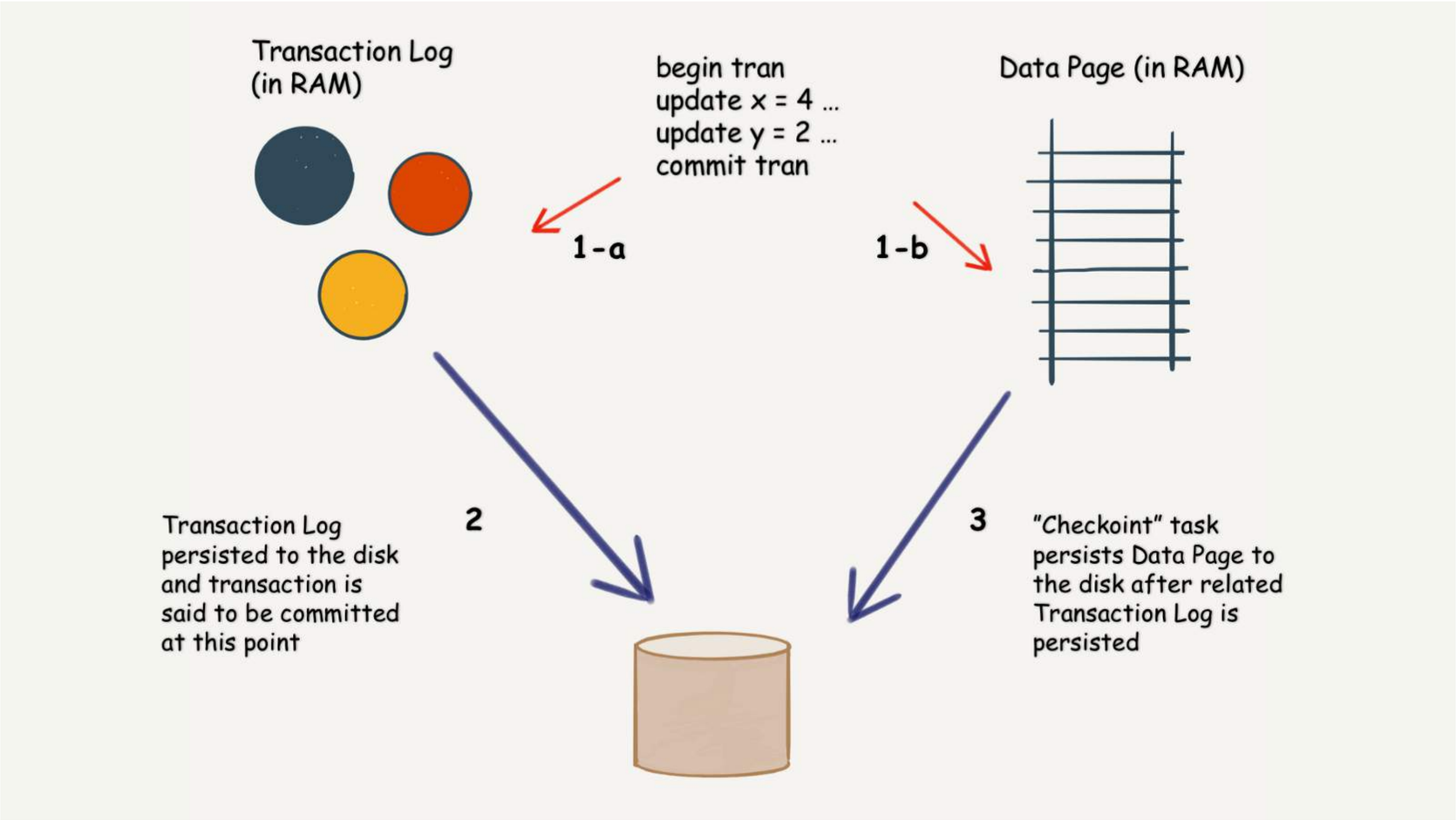
Data Consistency in Microservices

- Sagas and compensating transactions
- Write-ahead log

Write-ahead log

Write-ahead Log

– Write-ahead log



Data Consistency in Microservices

- Sagas and compensating transactions
- Write-ahead log
- Data reconciliation
- Data replication

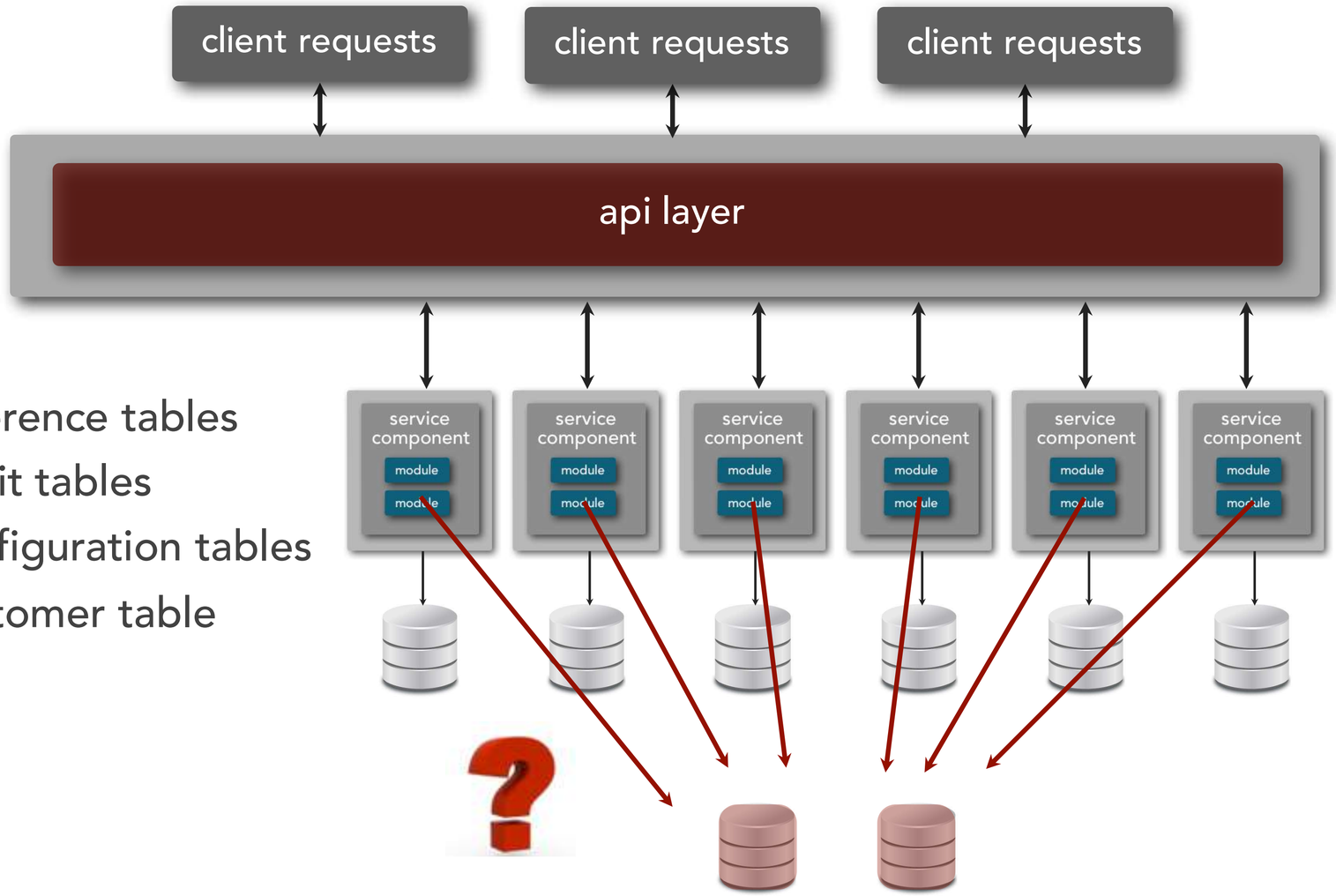
Data Replication

reference tables

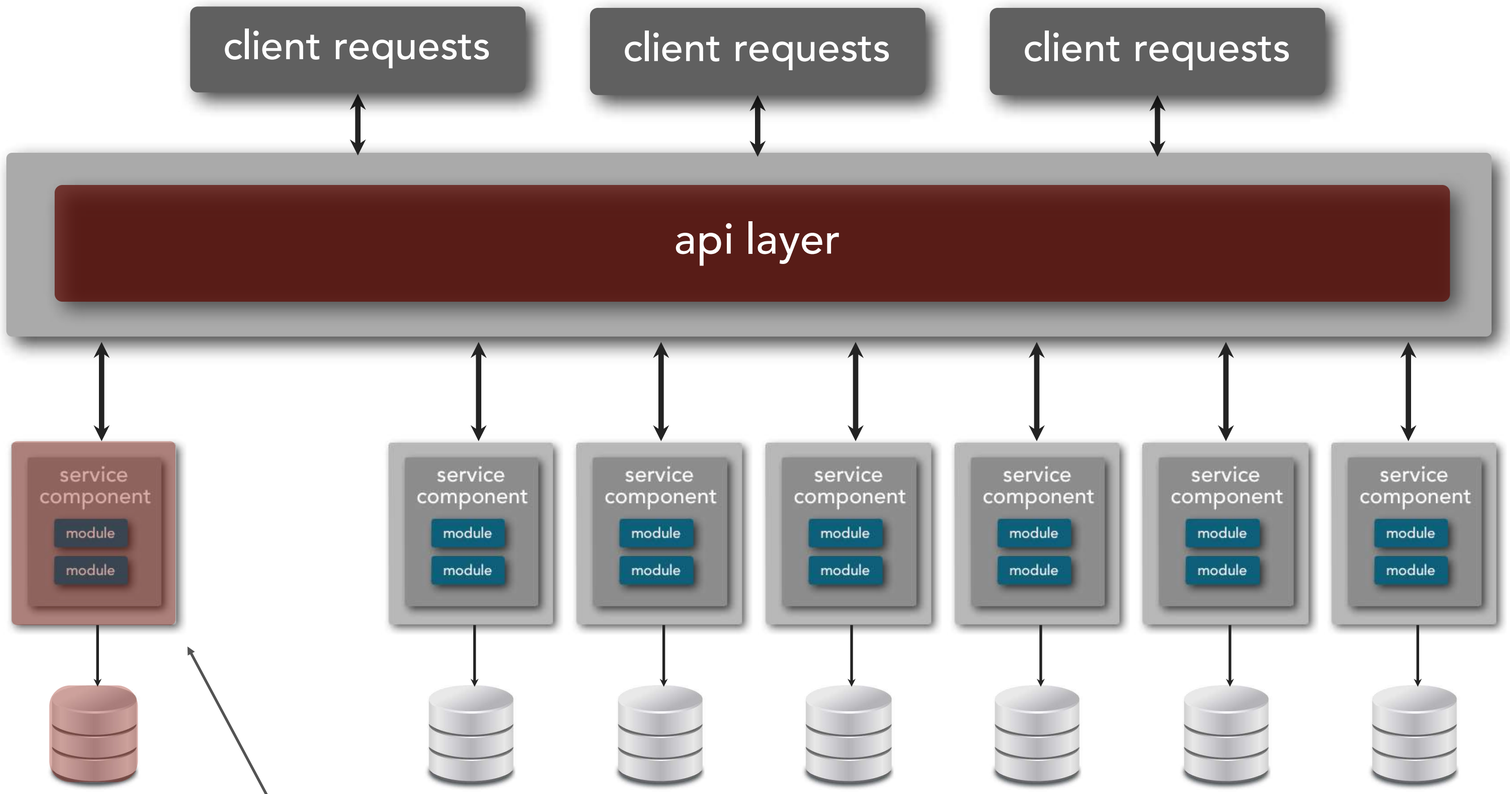
audit tables

configuration tables

customer table

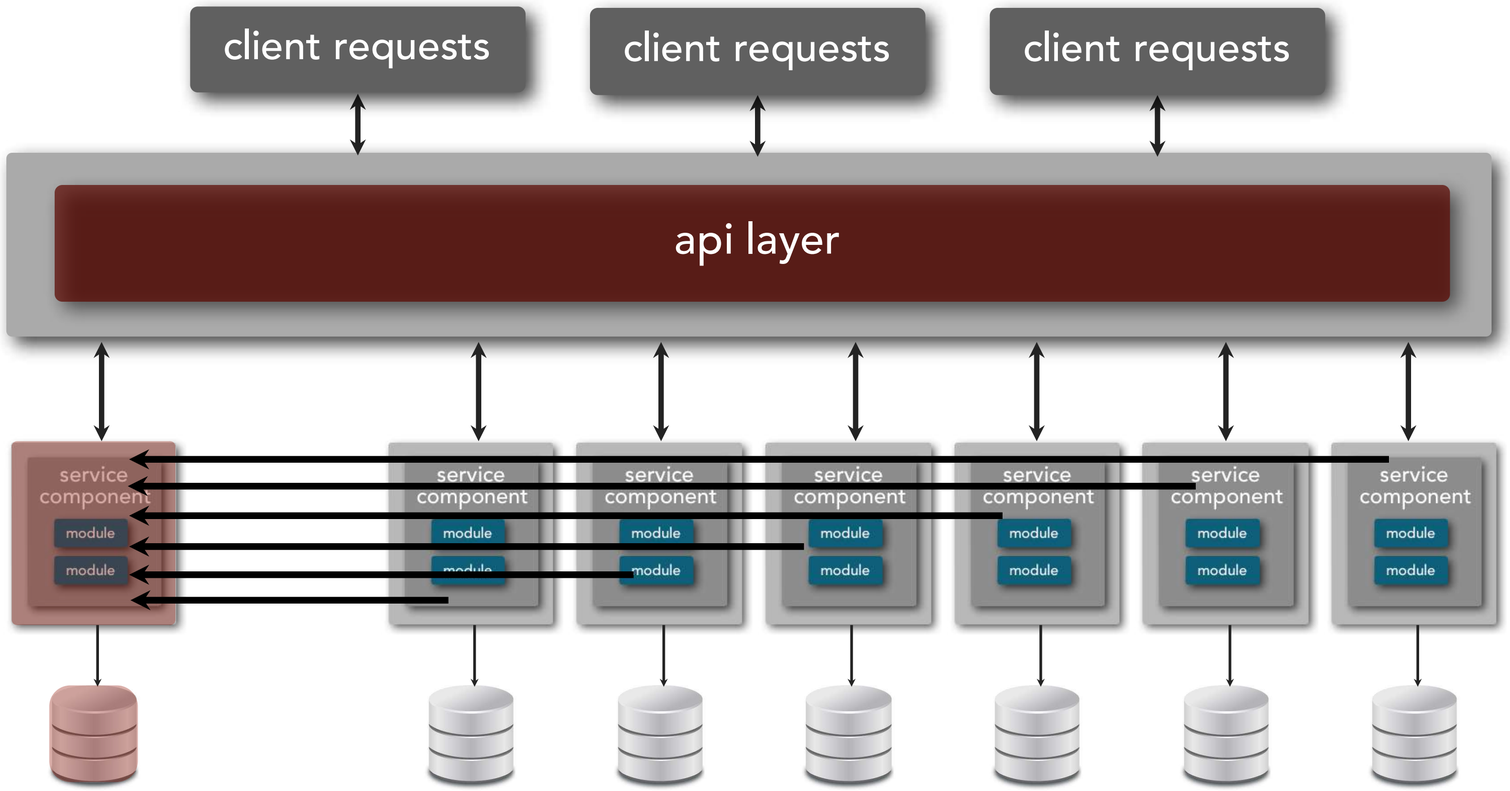


Data Replication



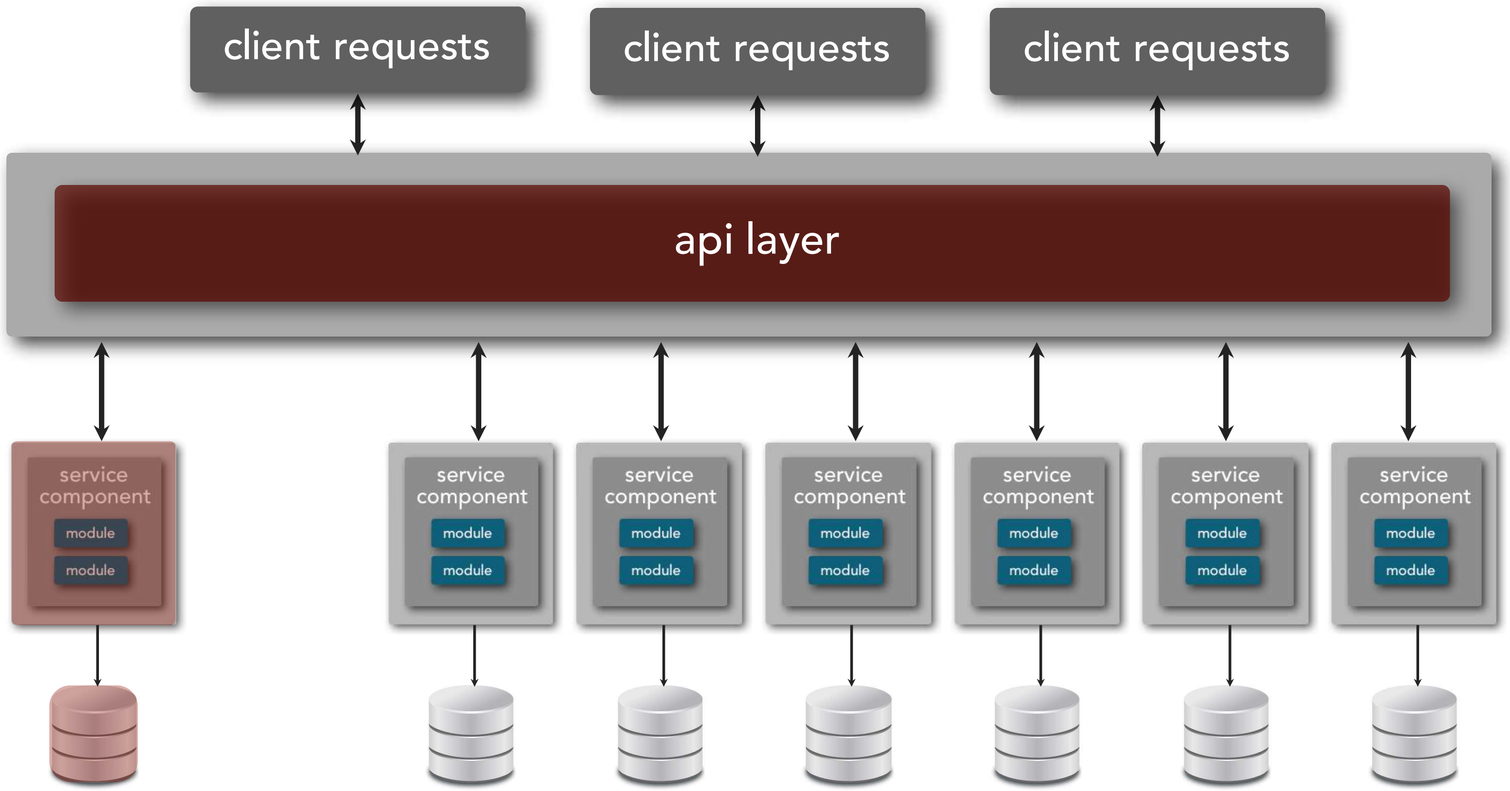
one service per common data domain
(ref, config, audit, etc.)

Data Replication

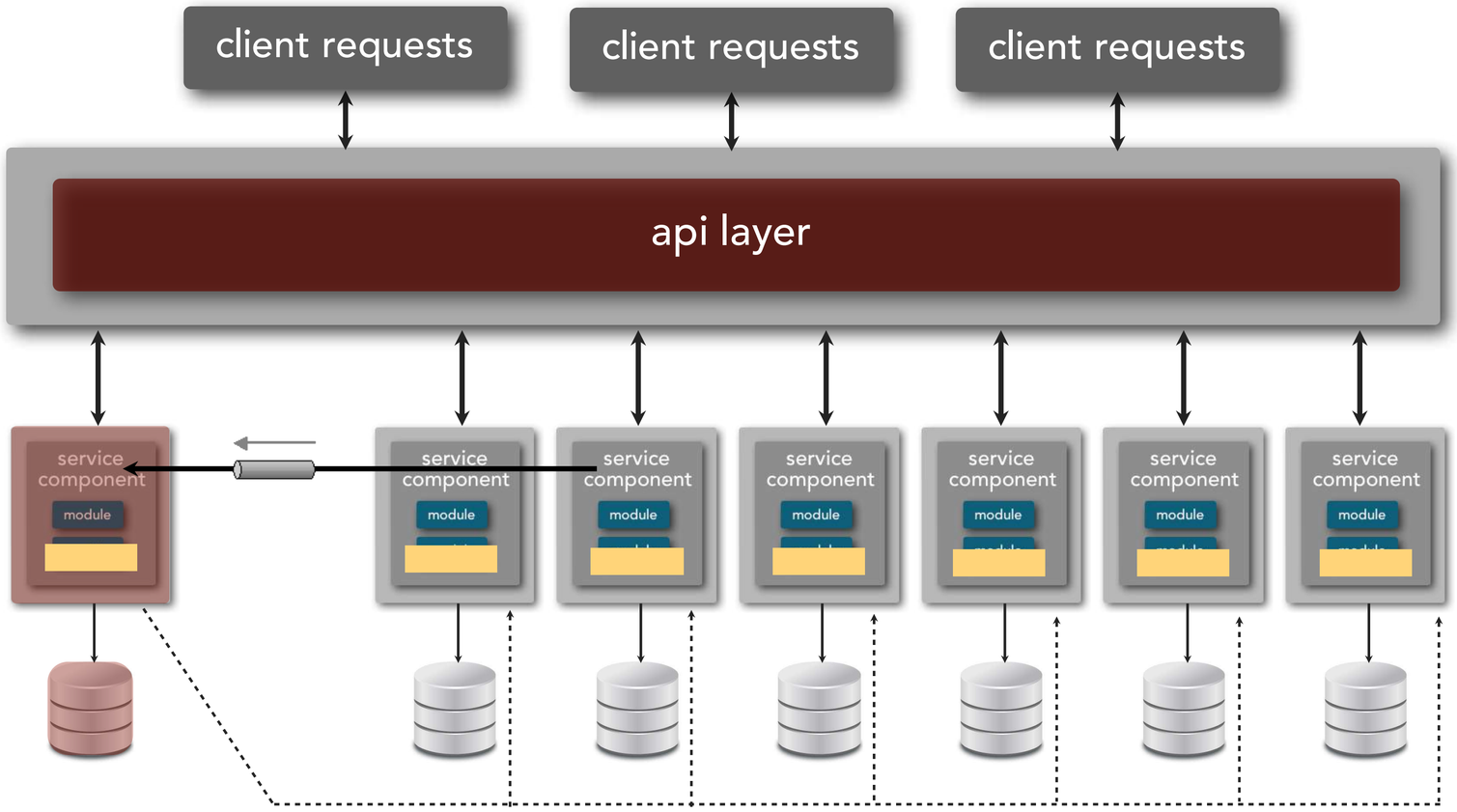


inter-service communication

Data Replication



Data Replication



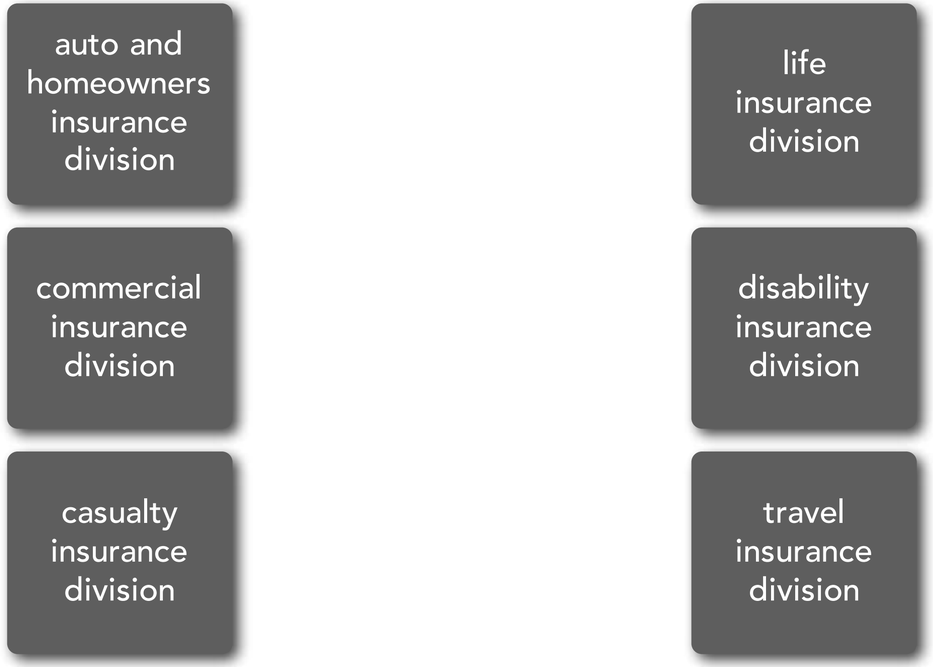
replicated cache

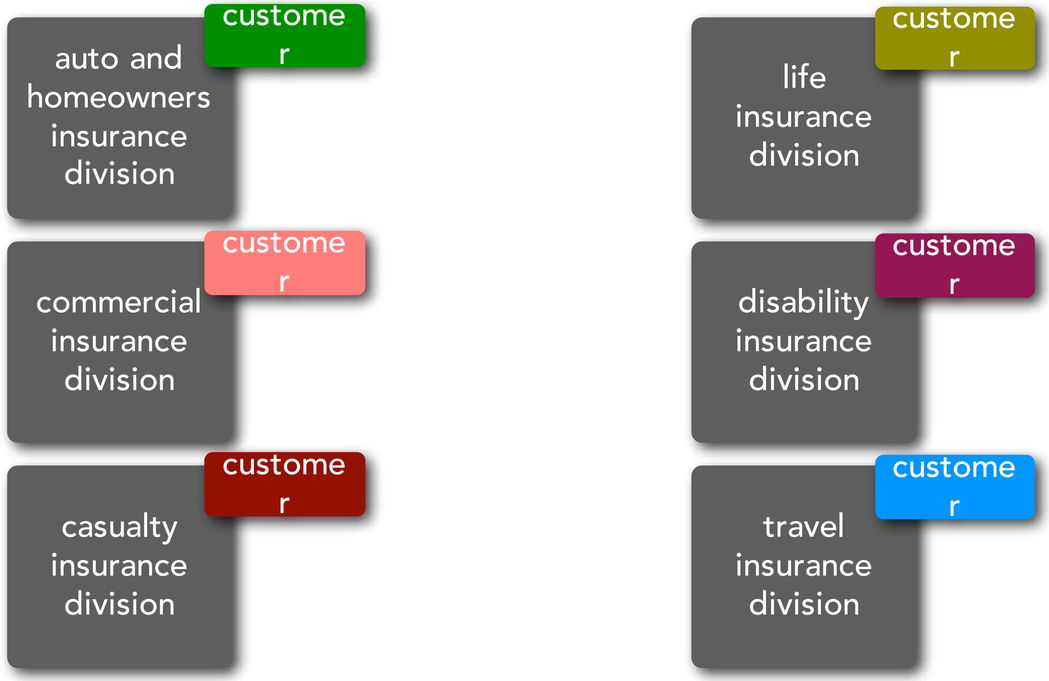
Data Consistency in Microservices

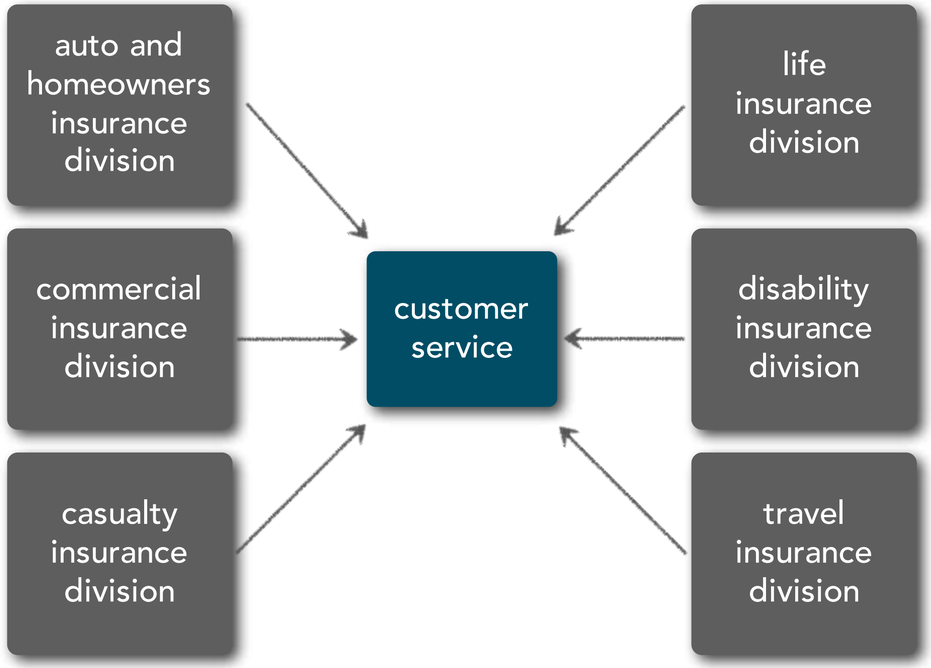
- Sagas and compensating transactions
- Write-ahead log
- Data reconciliation
- Data replication
- Events as a single source of truth

effective reuse

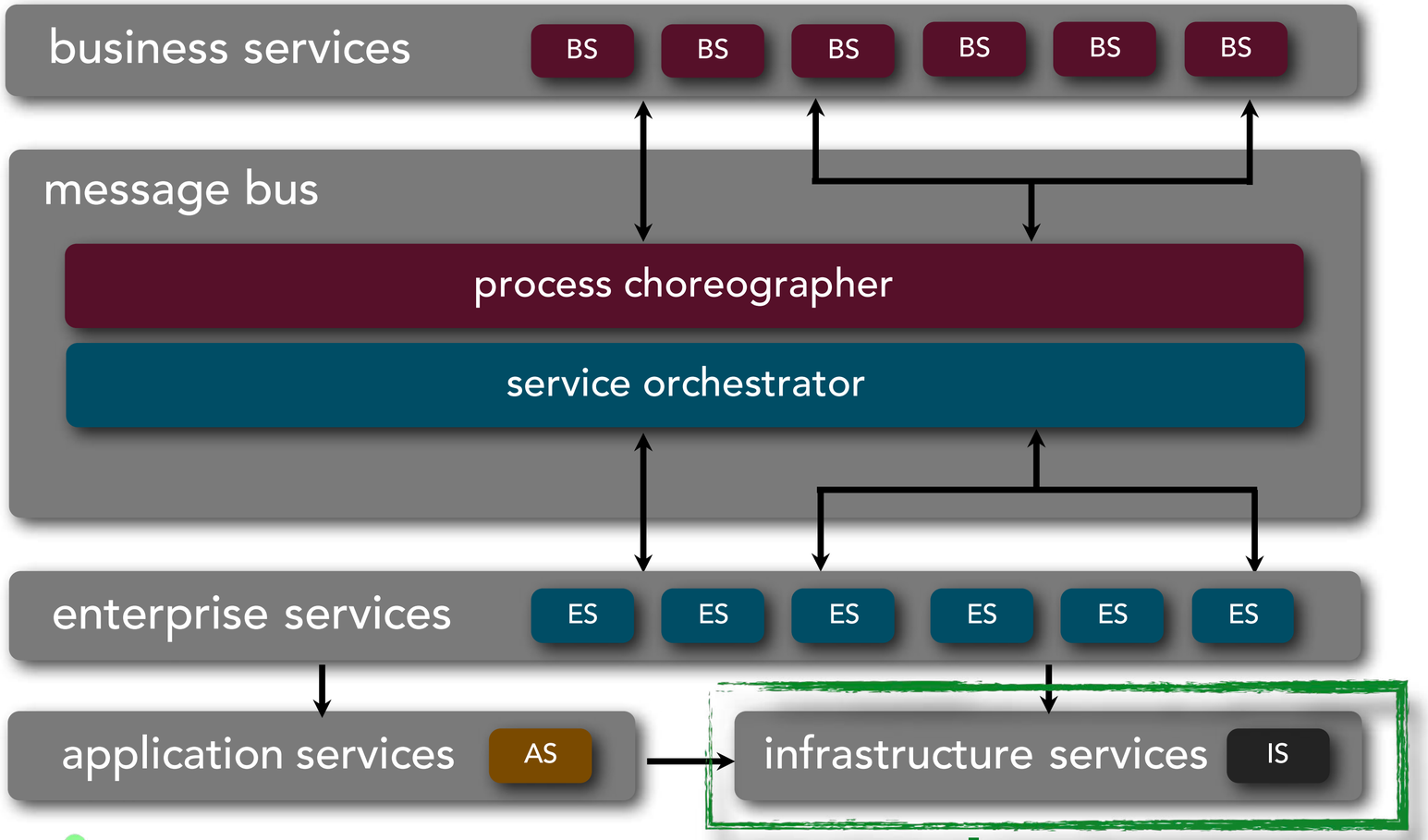
reuse in the SOA era




auto and homeowners insurance division
customer
commercial insurance division
customer
casualty insurance division
customer
life insurance division
customer
disability insurance division
customer
travel insurance division
customer

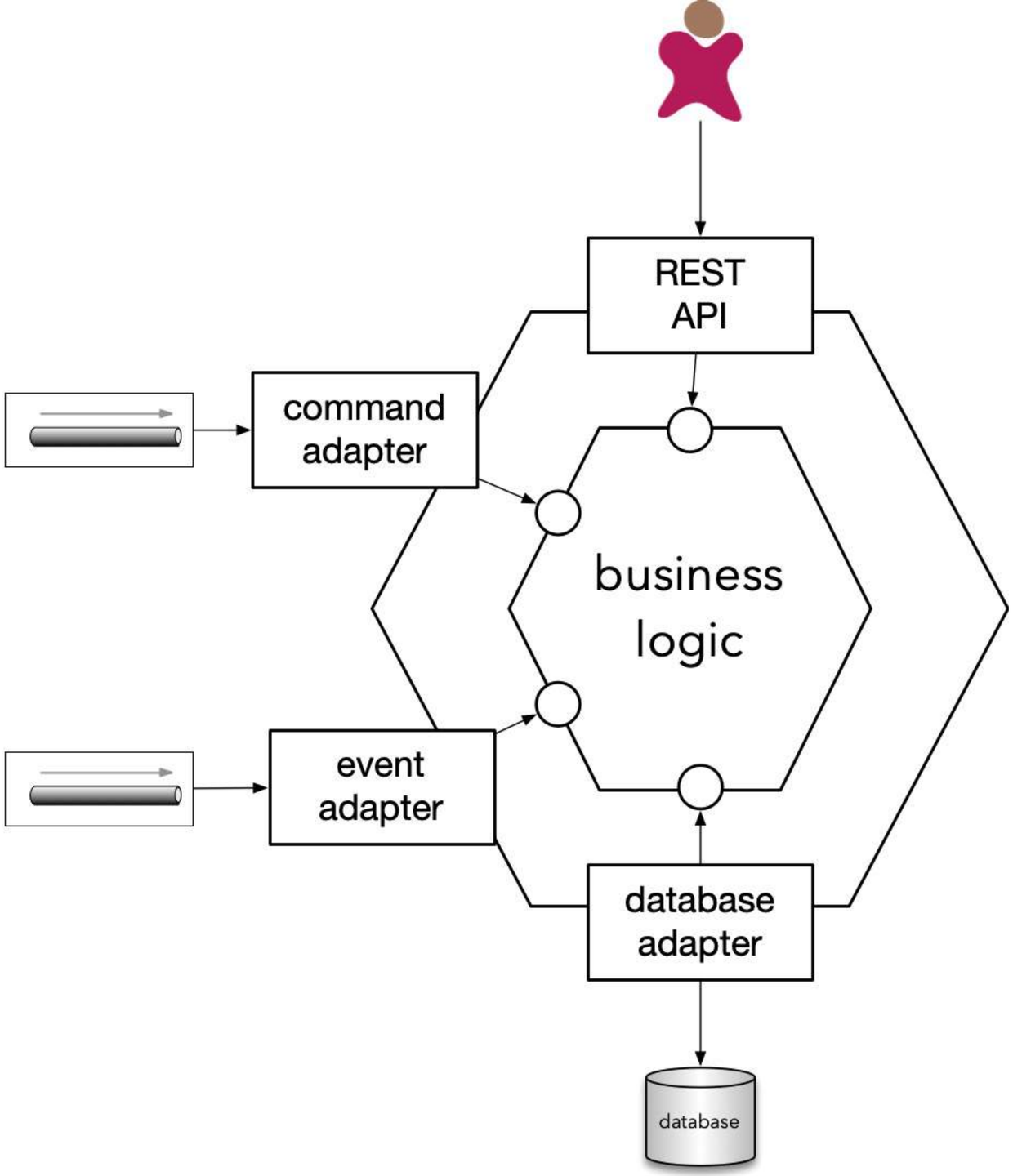

auto and homeowners insurance division
commercial insurance division
casualty insurance division
customer service
life insurance division
disability insurance division
travel insurance division

service-oriented architecture

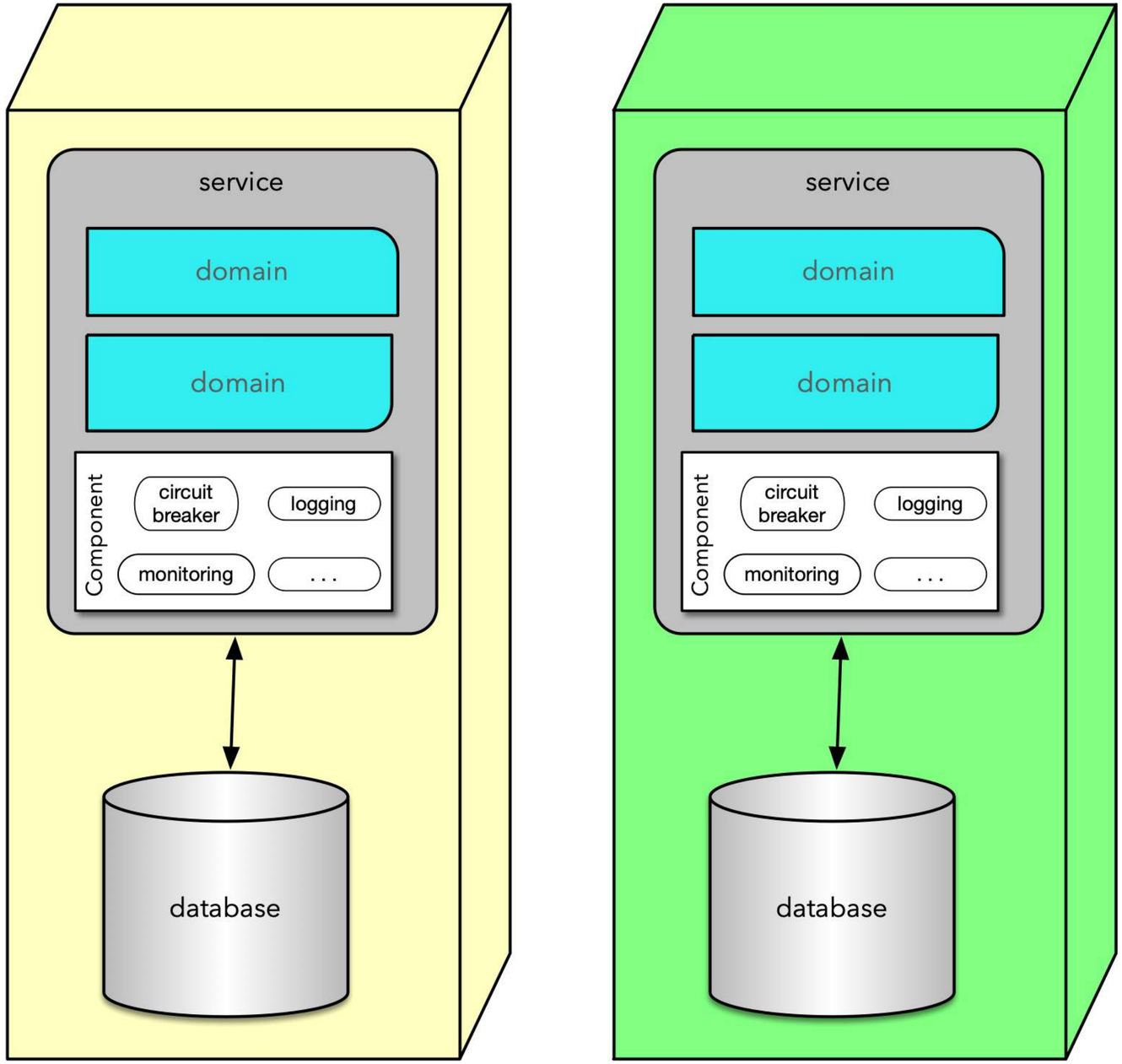


- maximize canonicality
- maximize reuse

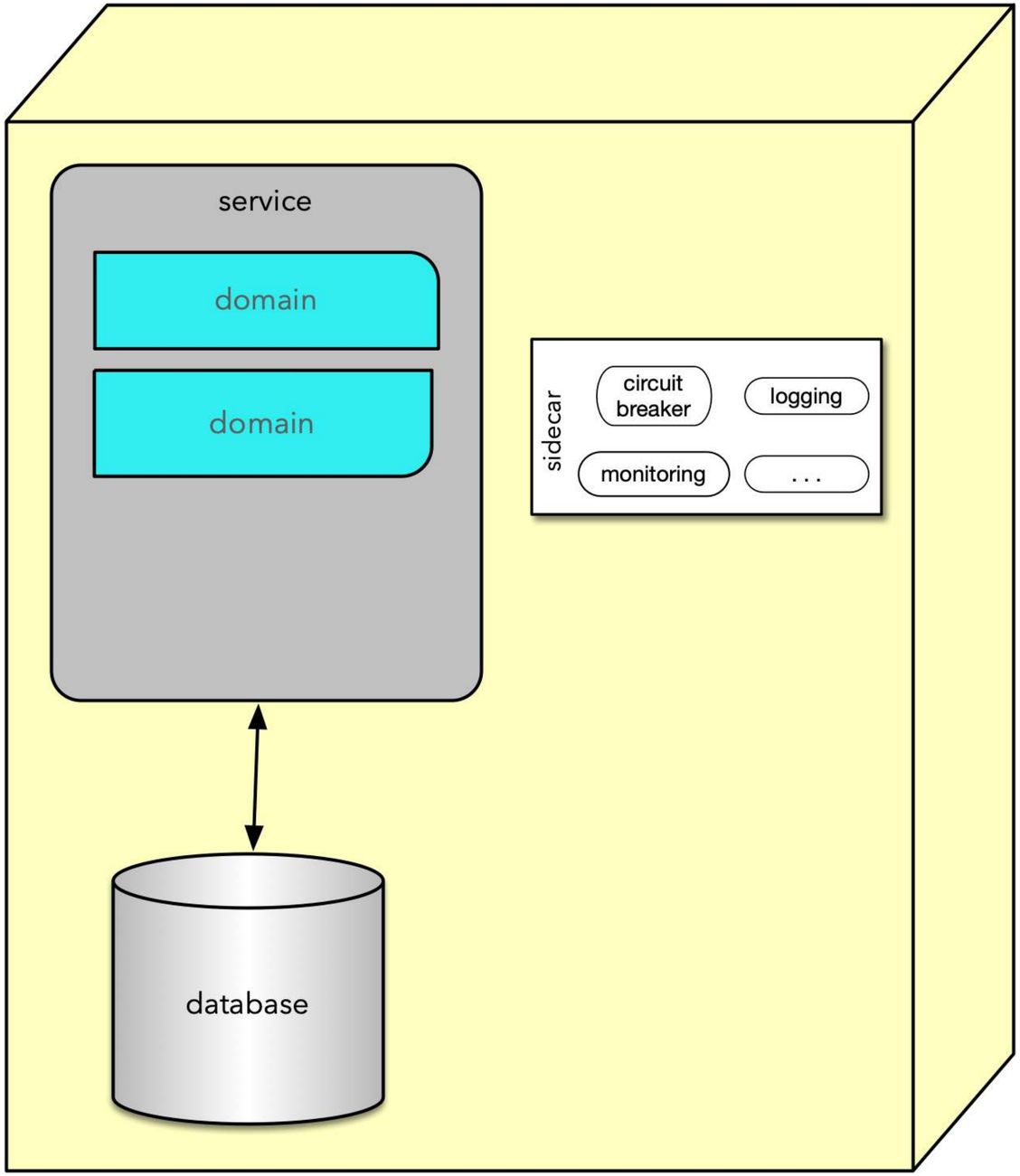
hexagonal pattern



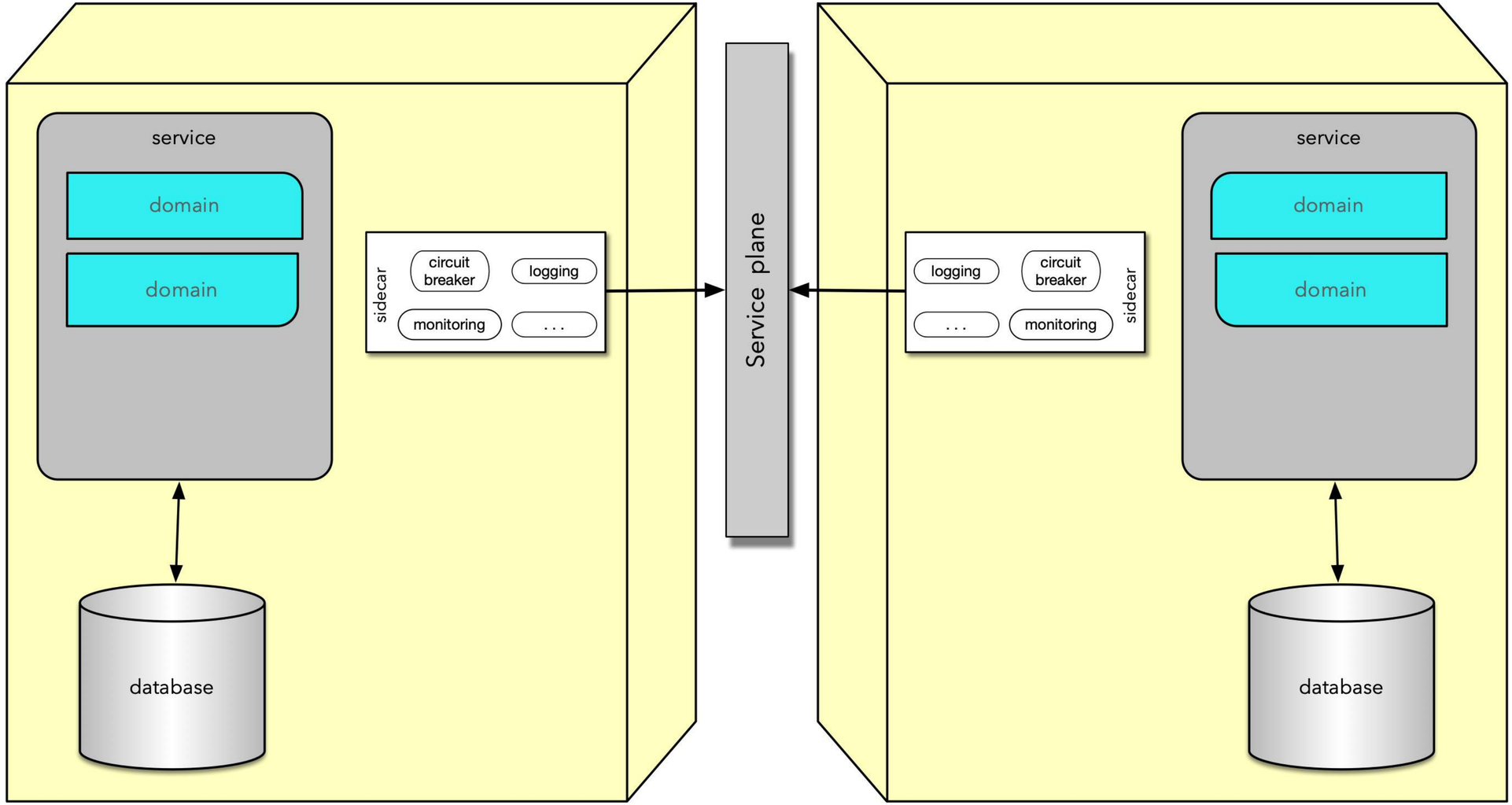
operational reuse



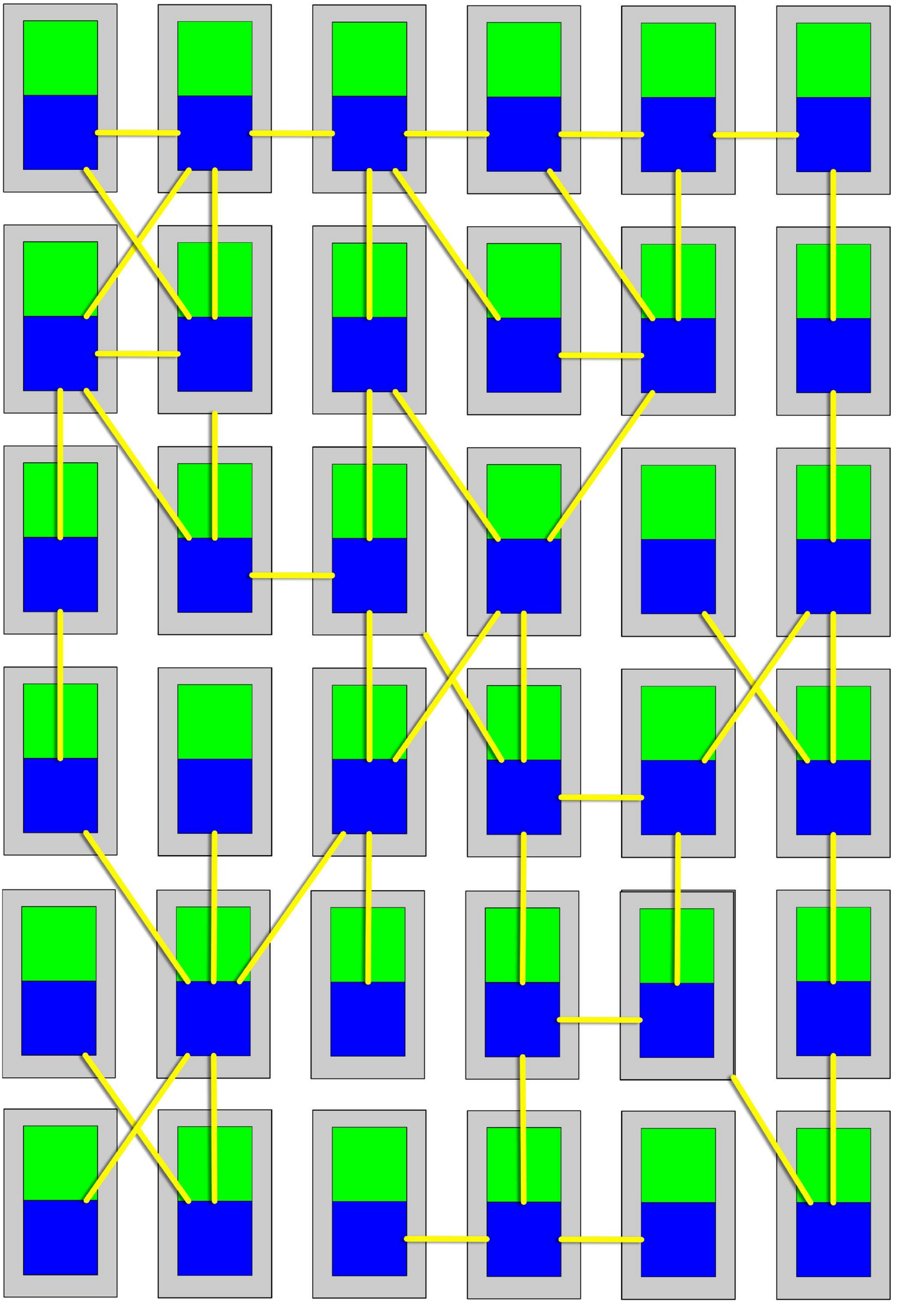
the sidecar pattern



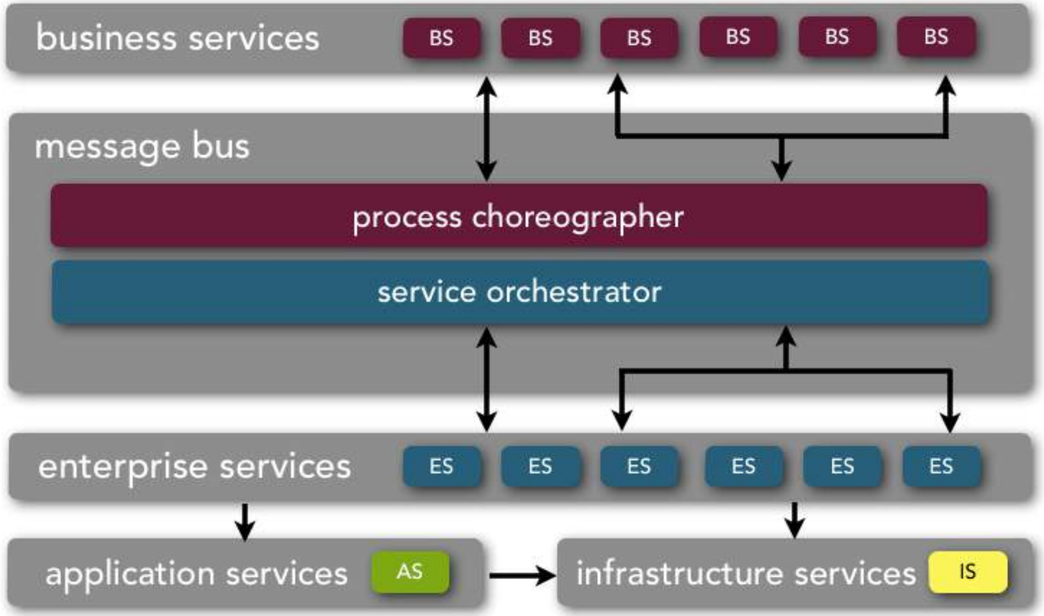
the sidecar pattern



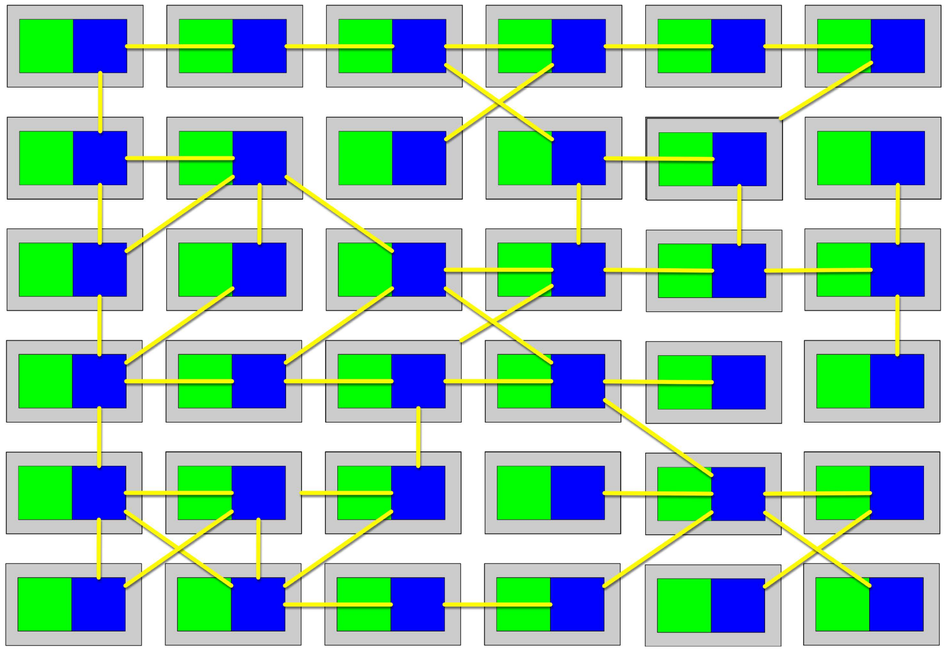
service mesh



two reuse solutions



SOA

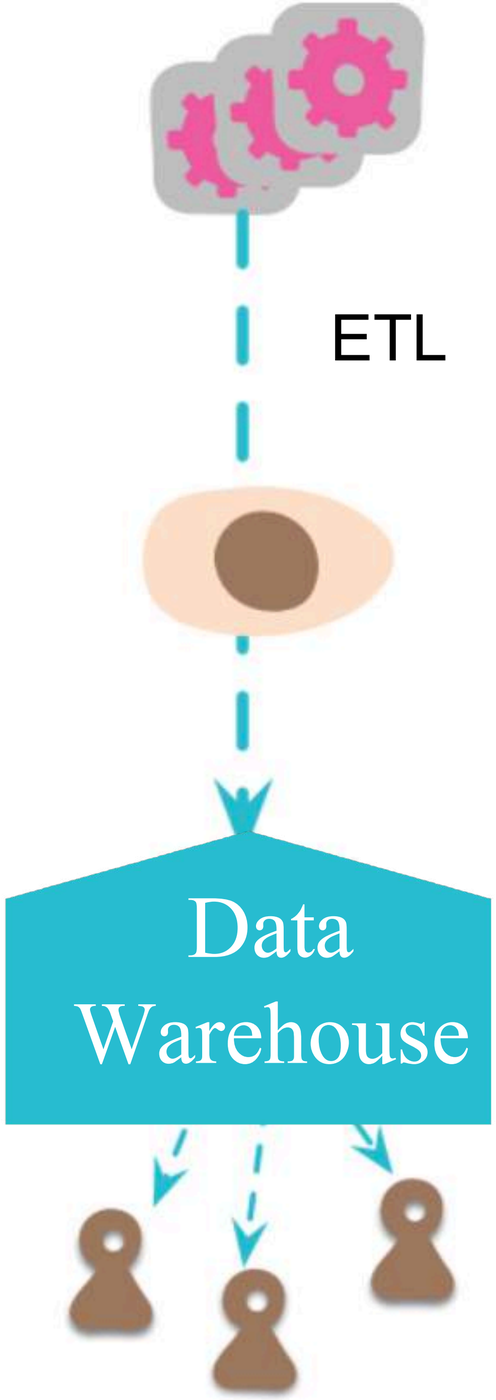


service mesh

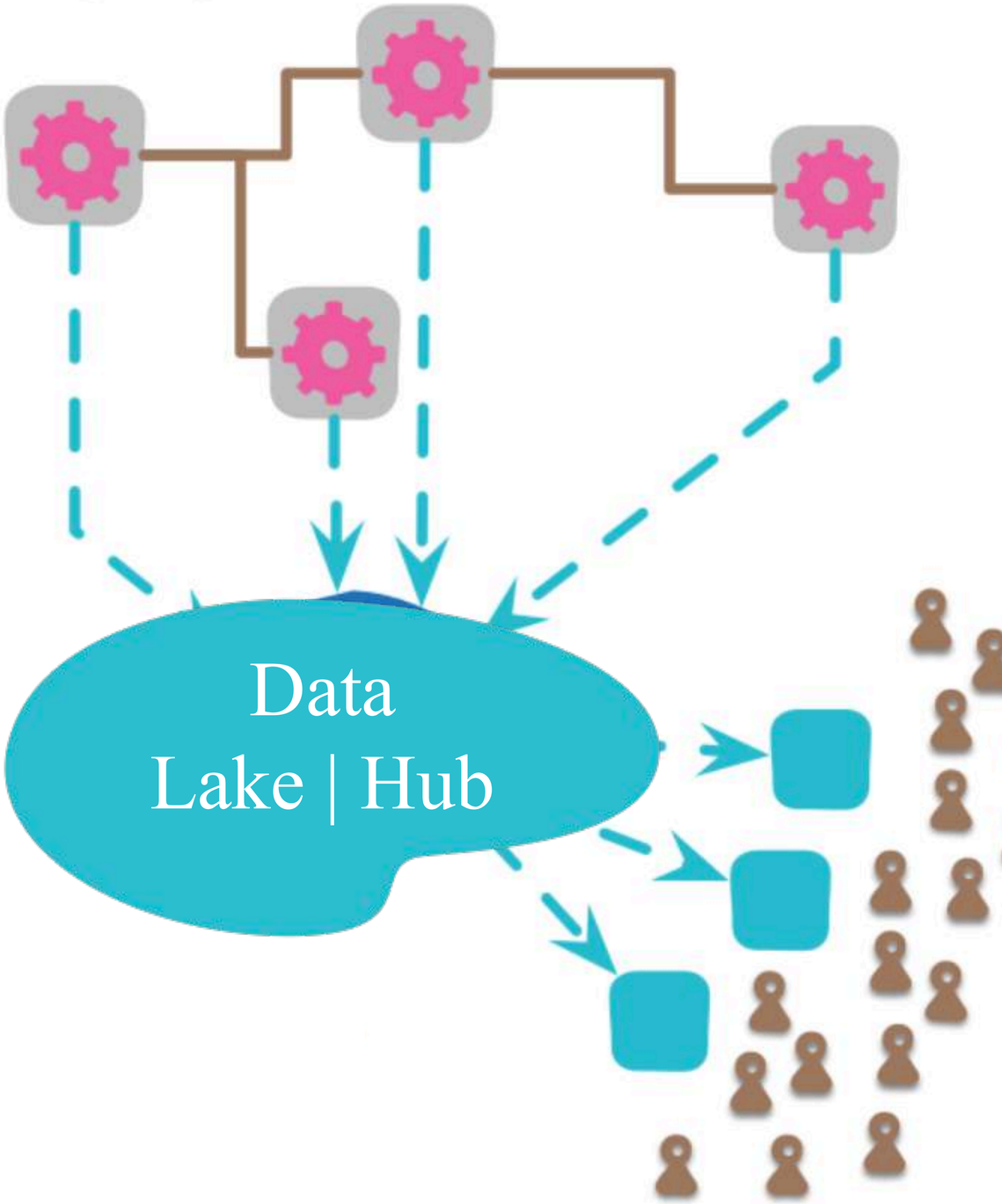
Data Mesh

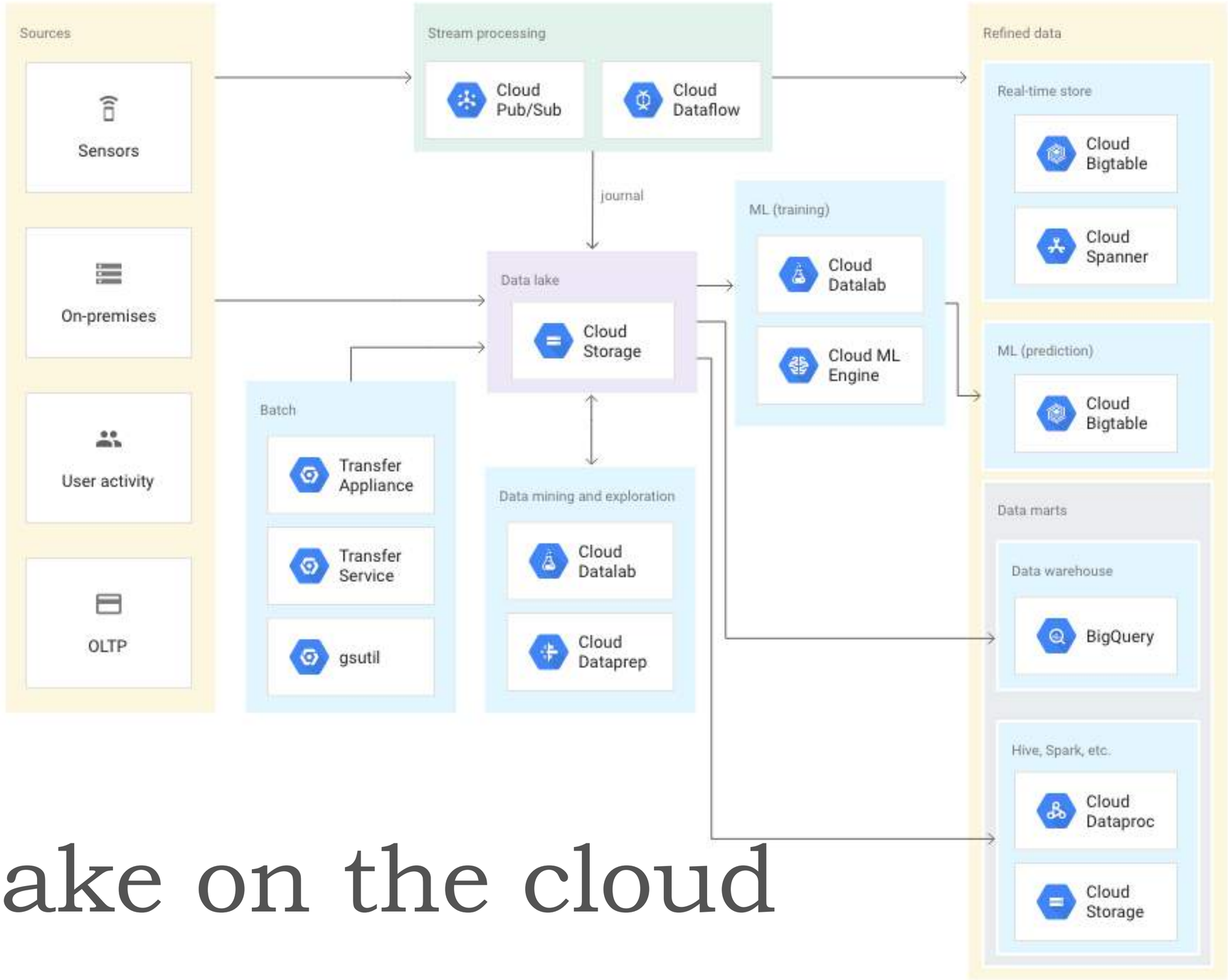
https://martinfowler.com/articles/data-monolith-to-mesh.html

data warehouse



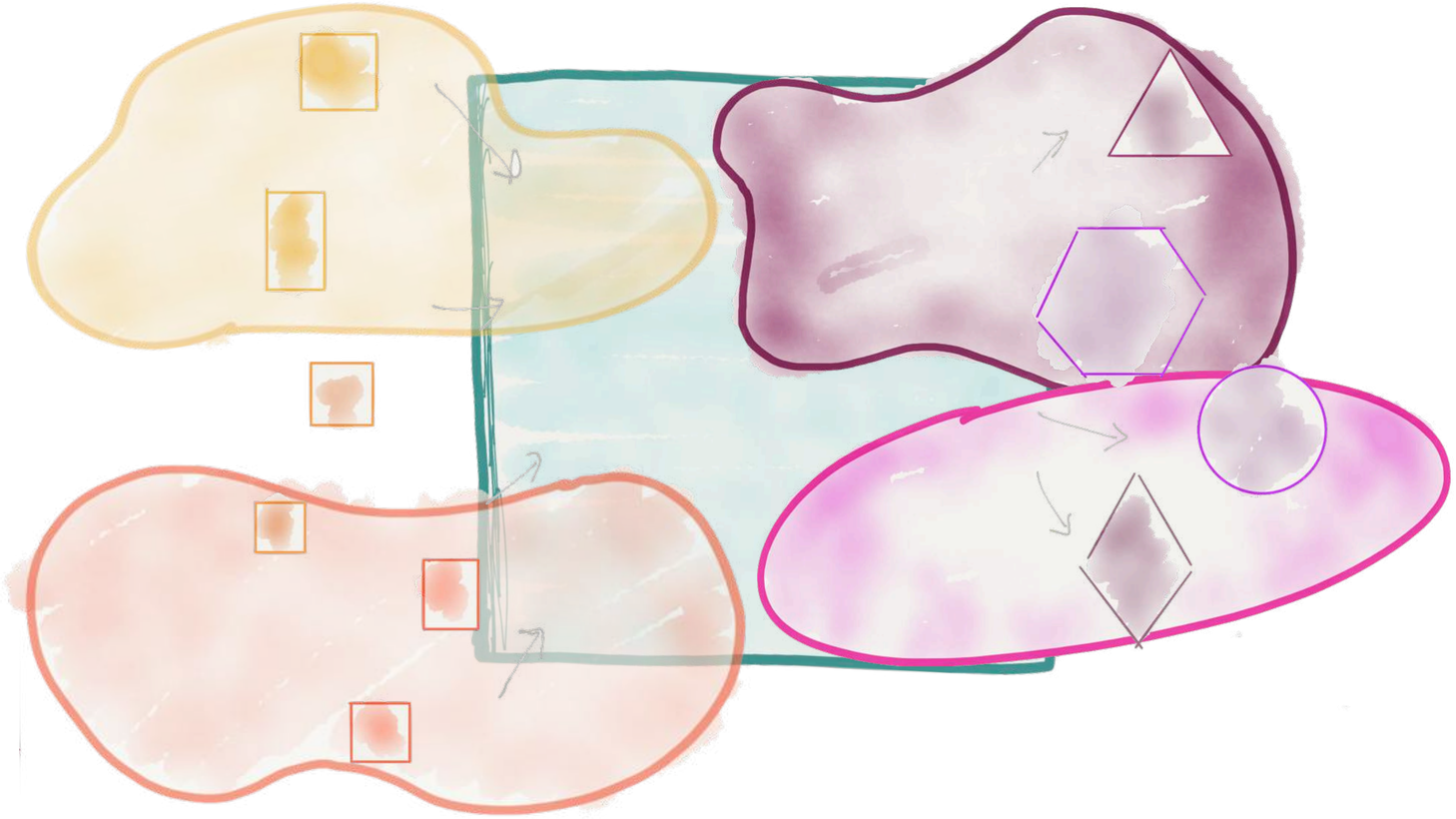
Data Lake



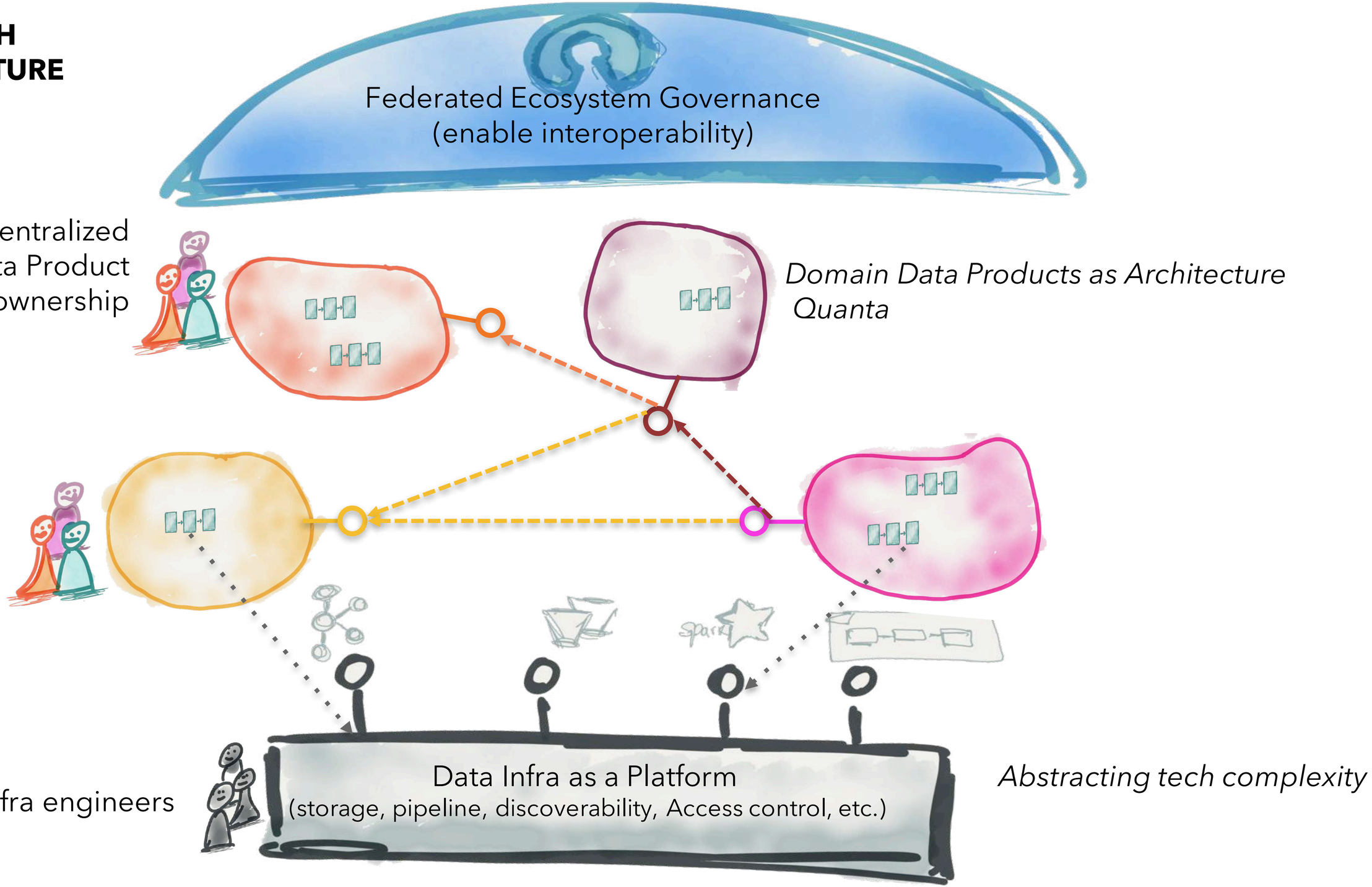


data lake on the cloud

Data Mesh



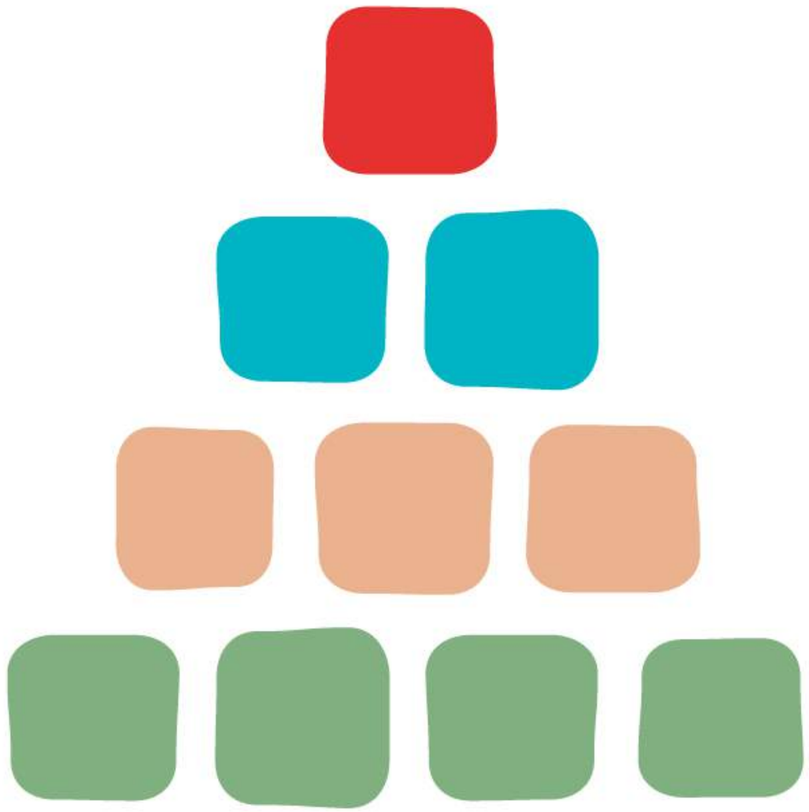
DATA MESH ARCHITECTURE



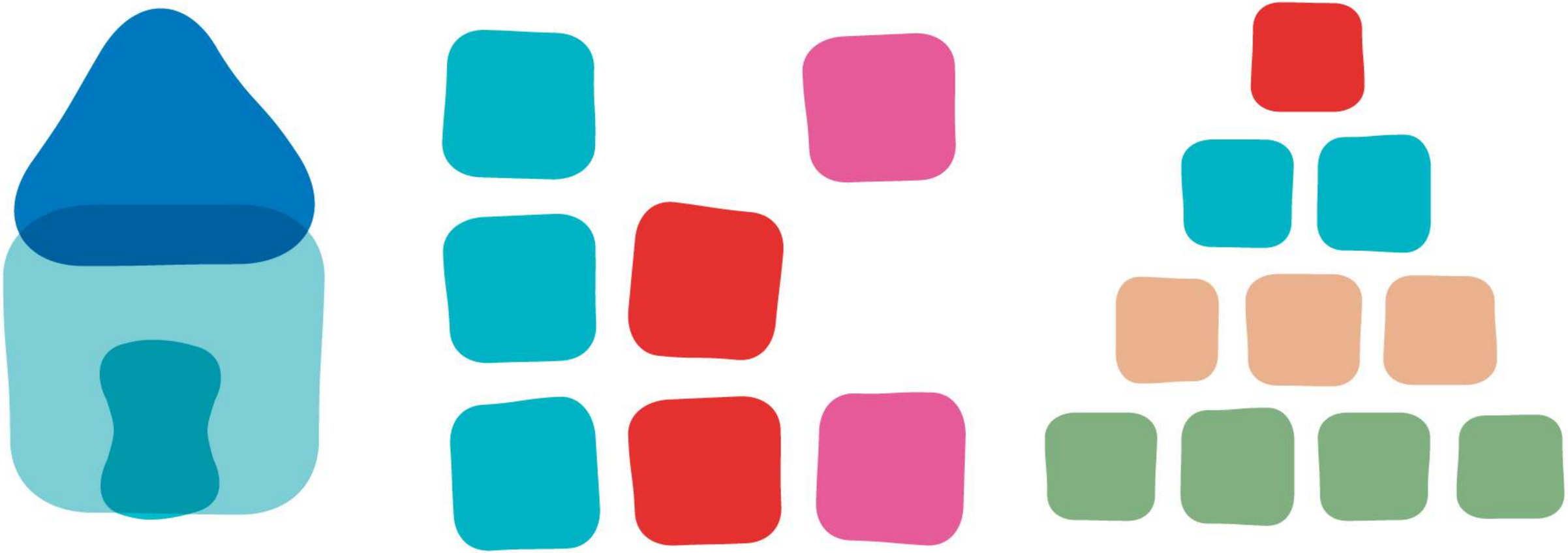
Data Mesh

https://martinfowler.com/articles/data-monolith-to-mesh.html

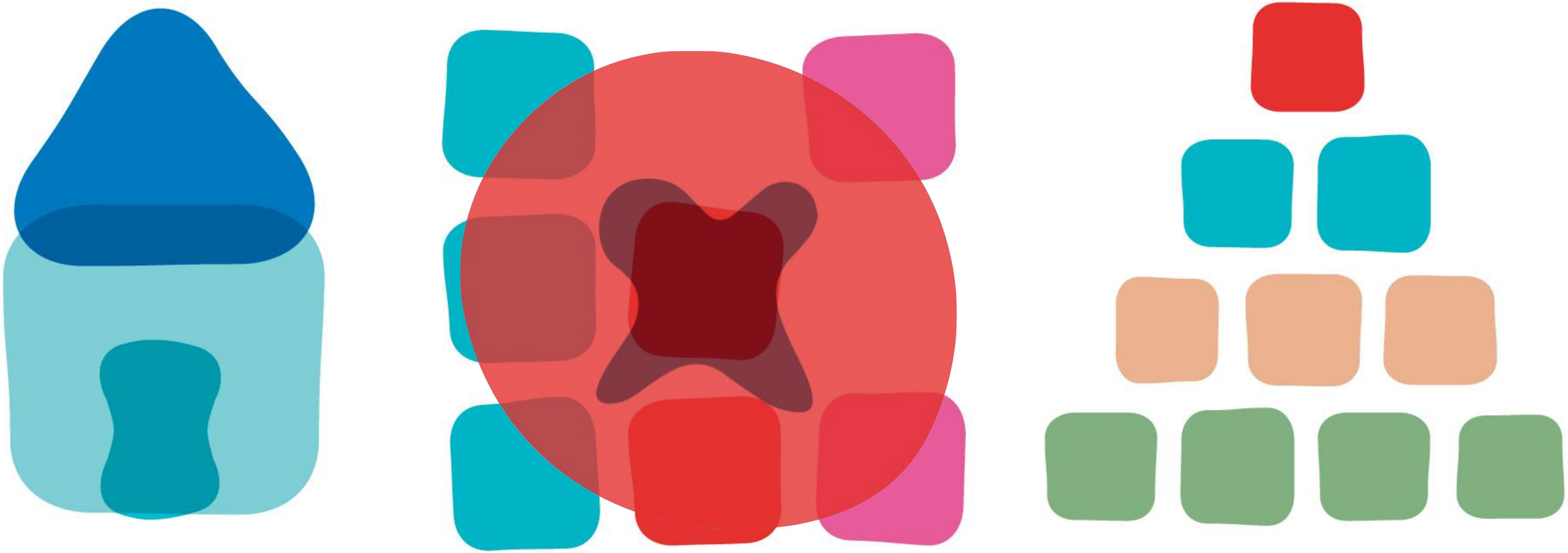
Domain/Architecture Isomorphism



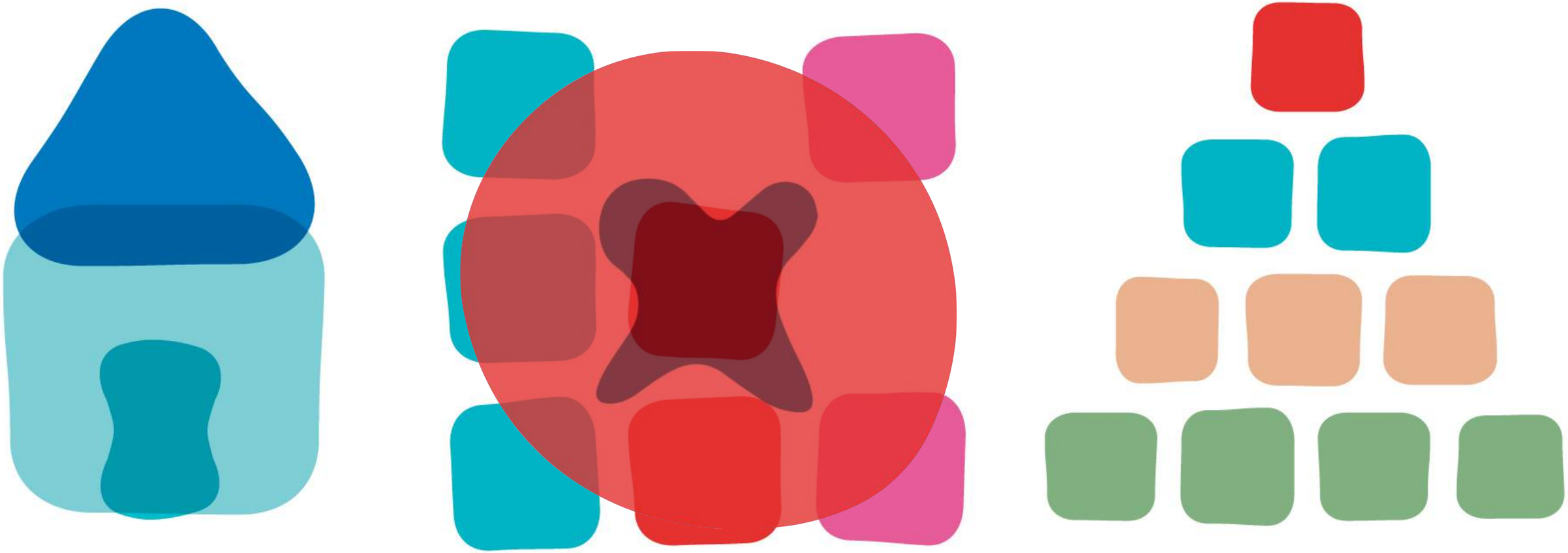
Domain/Architecture Isomorphism



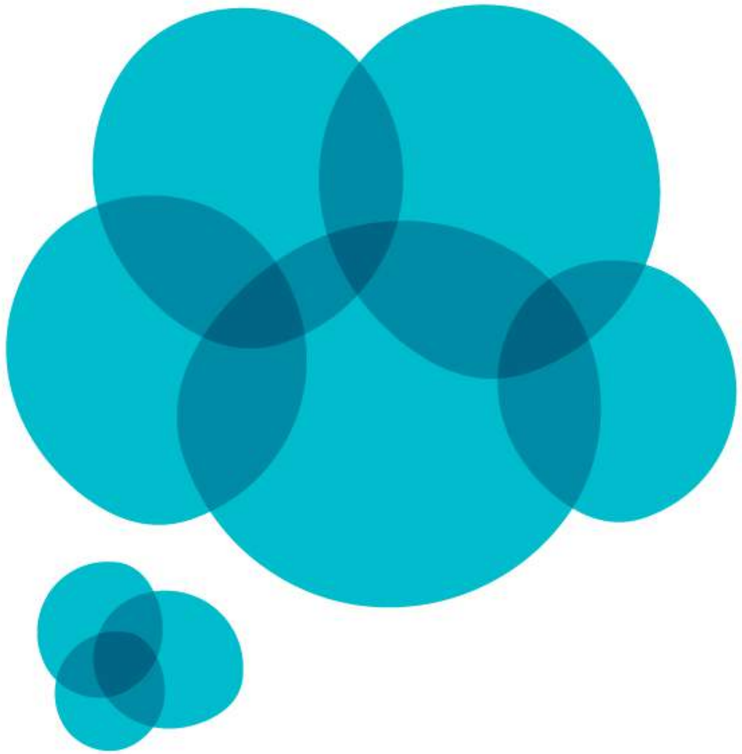
Domain/Architecture Isomorphism



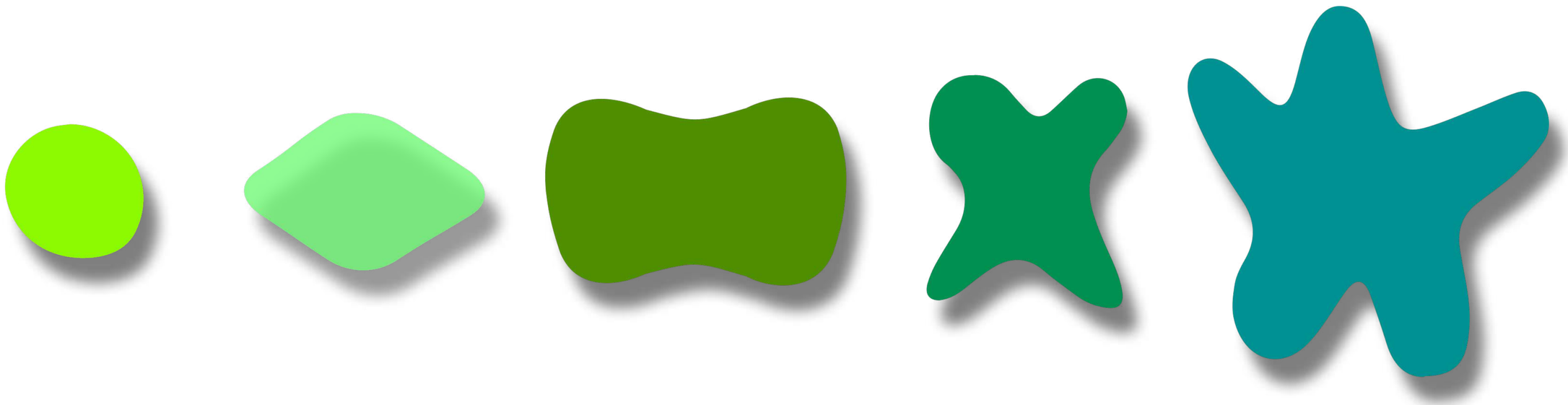
Domain/Architecture Isomorphism



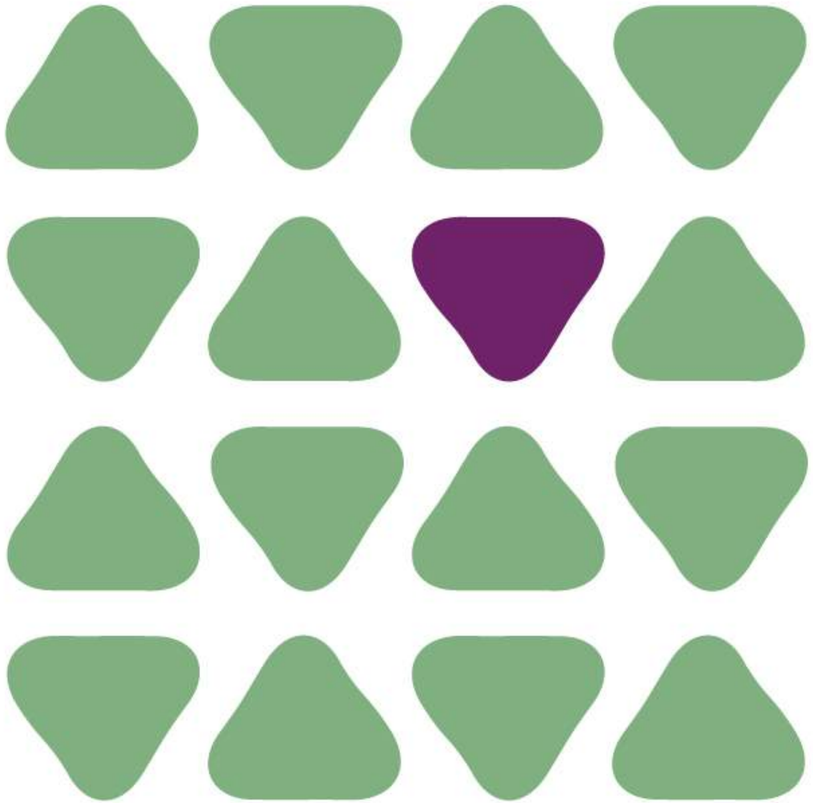
Understand Style Philosophy

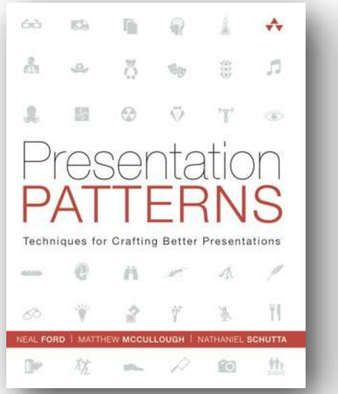


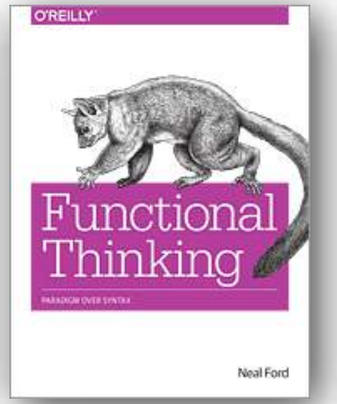
Hybrids Abound

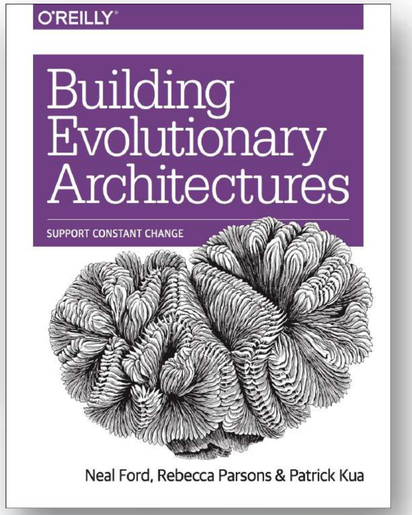


Distinguish Patterns from Tools









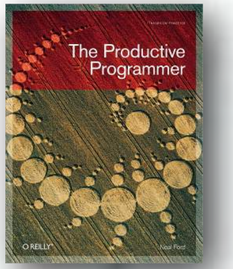
nealford.com

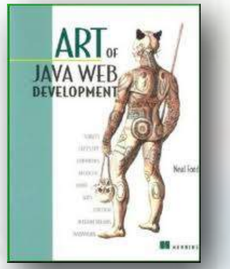
@neal4d

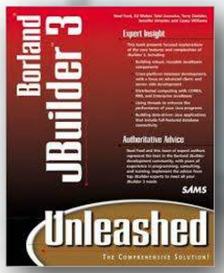
ThoughtWorks

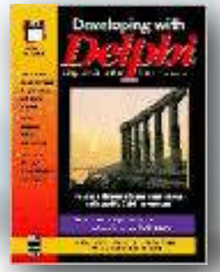
NEAL FORD

Director / Software Architect / Meme Wrangler

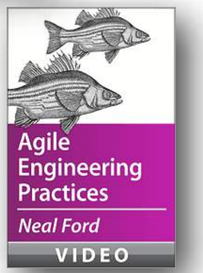


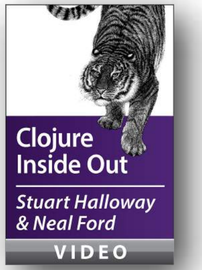


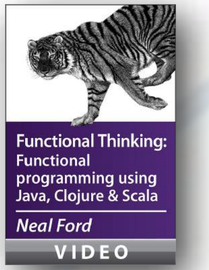




nealford.com/books

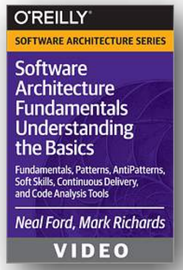
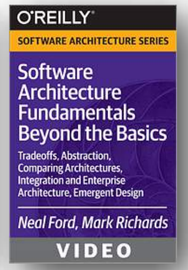
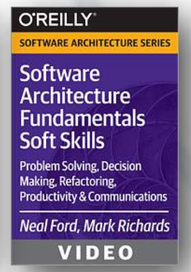
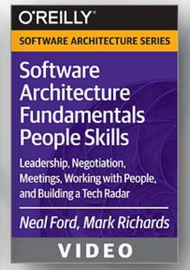
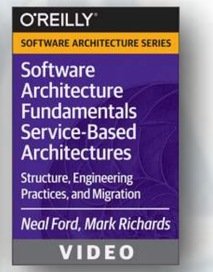
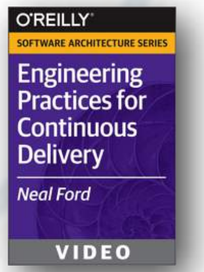






nealford.com/videos

O'REILLY®

SOFTWARE ARCHITECTURE SERIES

www.oreilly.com/software-architecture-video-training-series.html