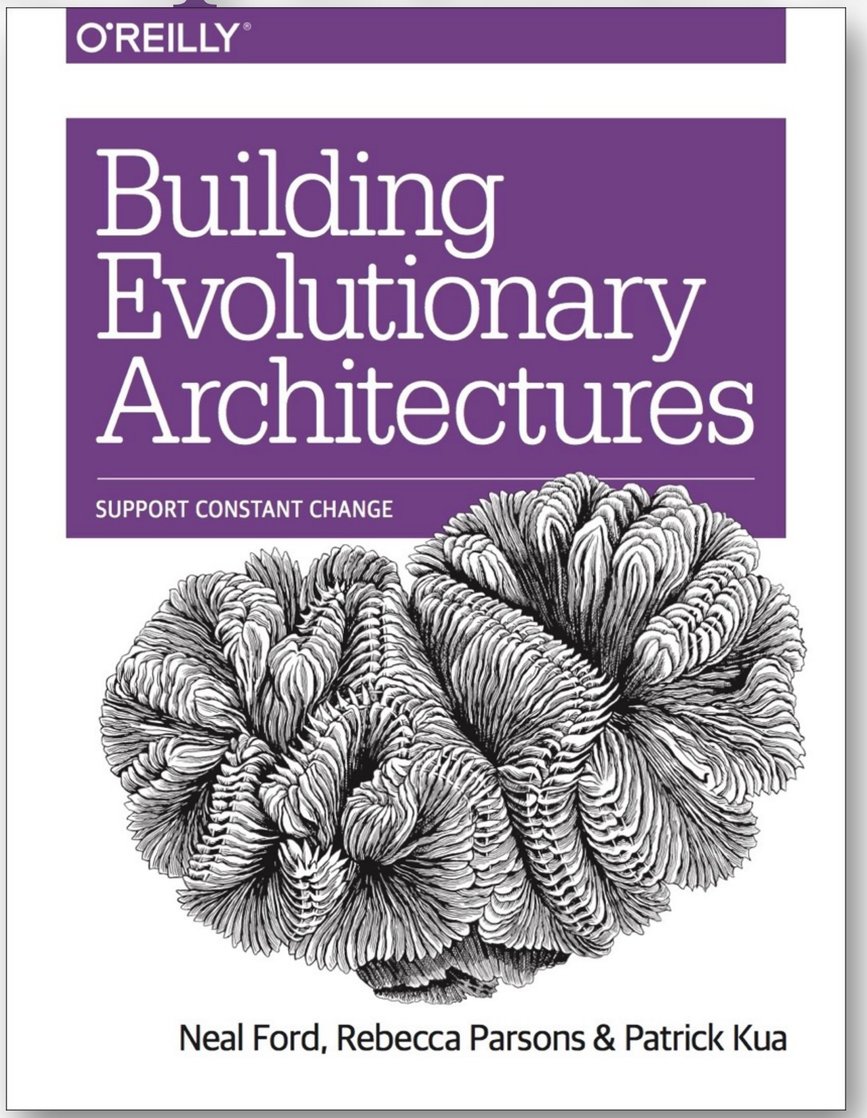
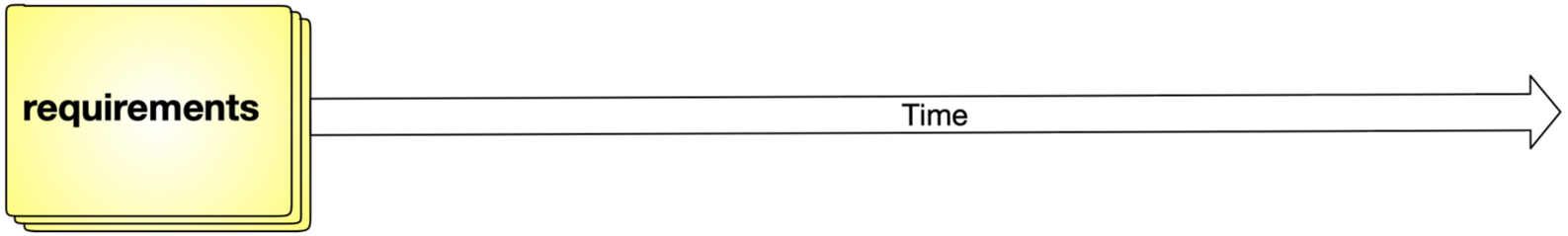
iMpLemENtiNg eVoLuTiONaRy ARcHitECtuREs

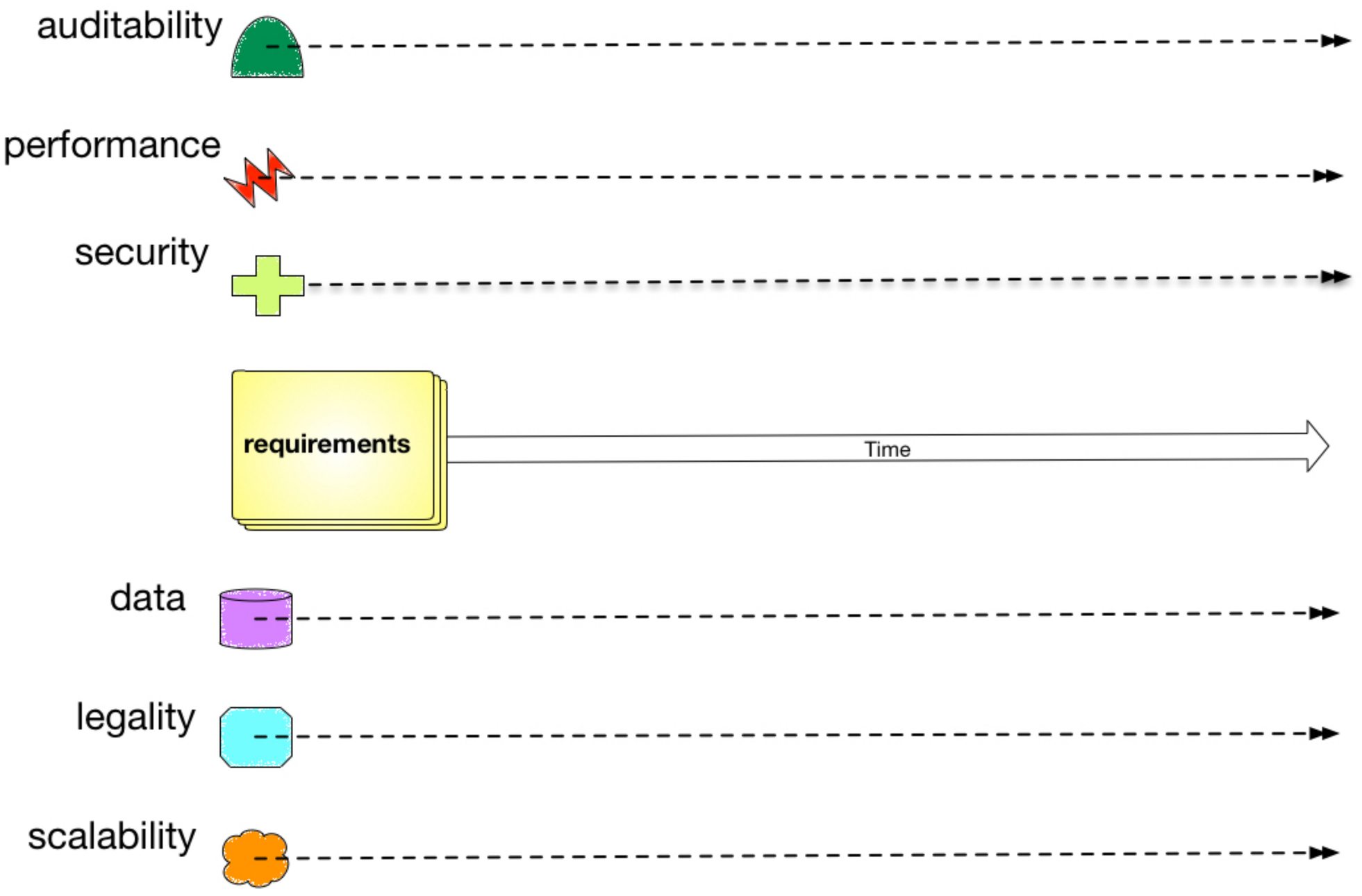


@neal4d
nealford.com

@rebeccaparsons

@patkua


requirements
Time


auditability
performance
security
requirements
Time
data
legality
scalability

Architecture Characteristics

1. synergistic

2. ill-defined

3. voluminous

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

reliability
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

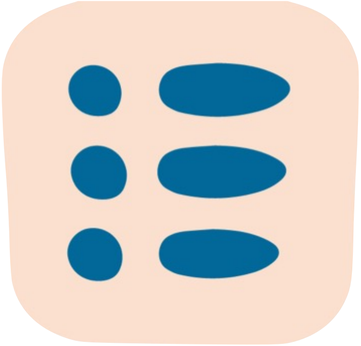
sampling

Architecture Characteristics

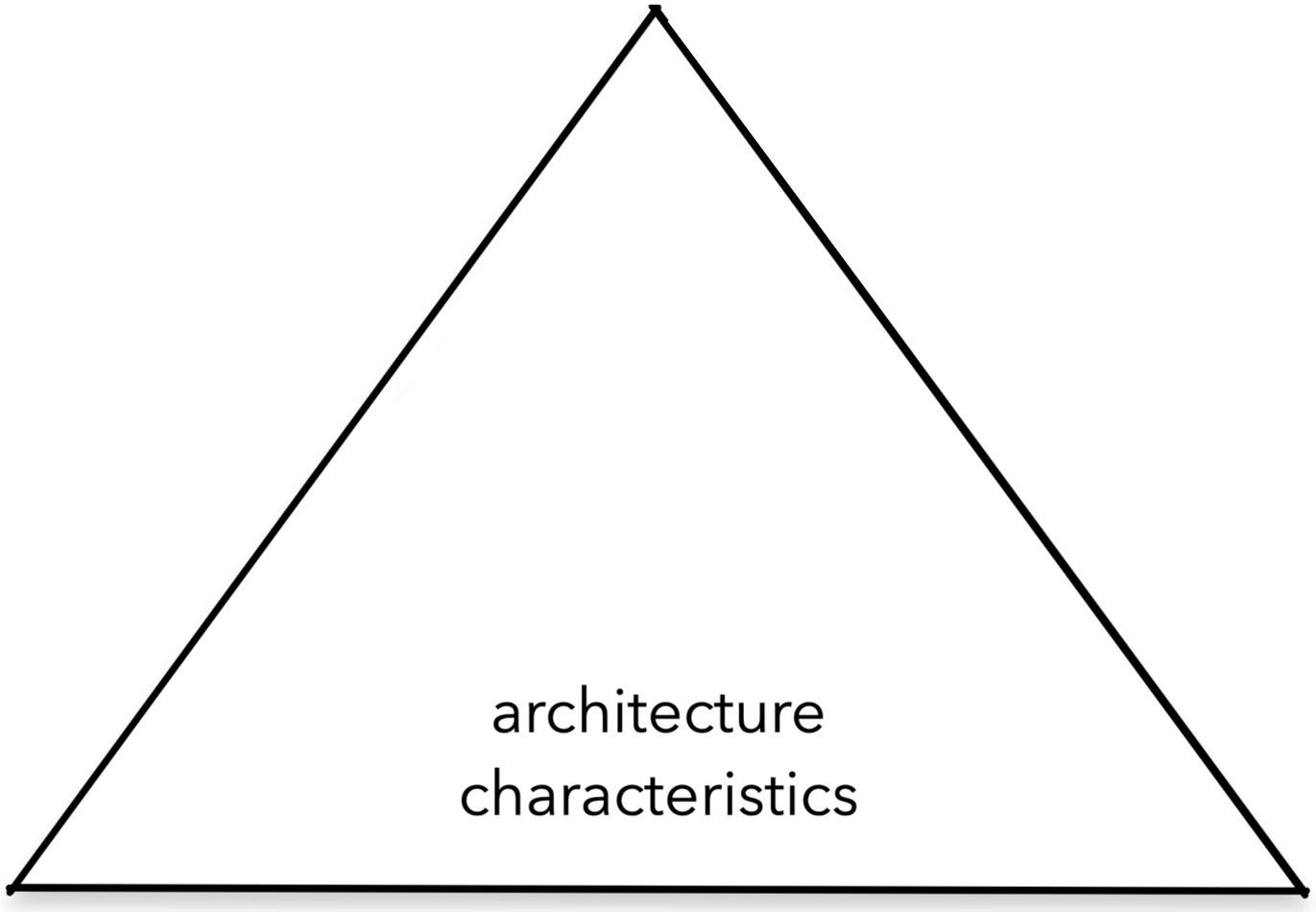
1. synergistic

2. ill-defined

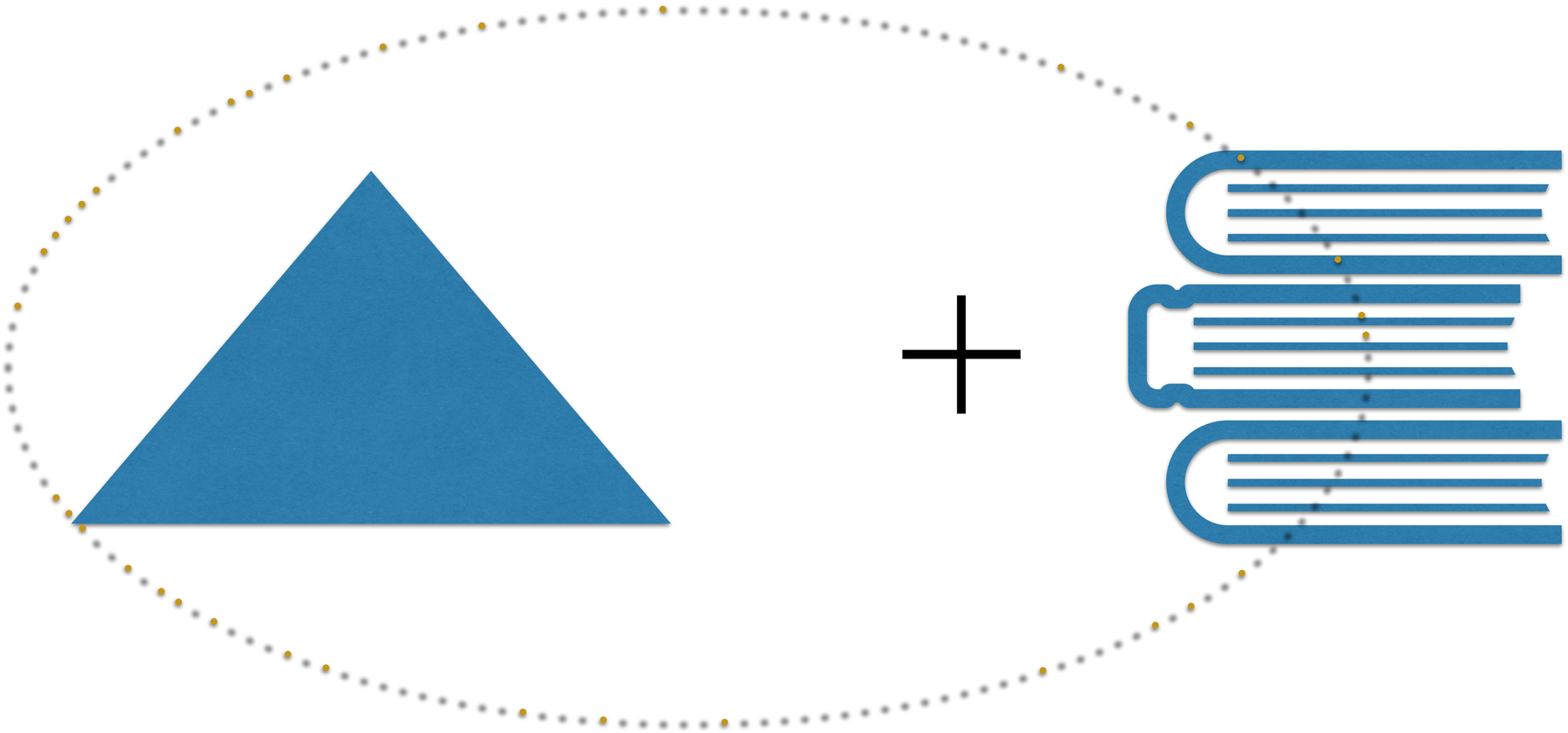
4. numerous categories



3. voluminous


architecture
characteristics

Architecture + Design



architecture styles

accessibility
accountability
accuracy
adaptability
administrability
affordability
agility
auditability

reliability
extensibility
failure transparency
fault-tolerance
fidelity
flexibility
inspectability
installability

repeatability
reproducibility
resilience
responsiveness
reusability
robustness
safety
scalability

evolvability

debugability
degradability
determinability
demonstrability
dependability
deployability
discoverability
distributability
durability
effectiveness
efficiency

modularity
operability
orthogonality
portability
precision
predictability
process capabilities
producibility
provability
recoverability
relevance

survivability
sustainability
tailorability
testability
timeliness
traceability
transparency
ubiquity
understandability
upgradability
usability

accessibility
accountability
accuracy
adaptability
administrability
affordability
agility
auditability

reliability
extensibility
failure transparency
fault-tolerance
fidelity
flexibility
inspectability
installability

repeatability
reproducibility
resilience
responsiveness
reusability
robustness
safety
scalability

evolvability

debugability
degradability
determinability
demonstrability
dependability
deployability
discoverability
distributability
durability
effectiveness
efficiency

modularity
operability
orthogonality
portability
precision
predictability
process capabilities
producibility
provability
recoverability
relevance

survivability
sustainability
tailorability
testability
timeliness
traceability
transparency
ubiquity
understandability
upgradability
usability

cHAnGe

technology
driven

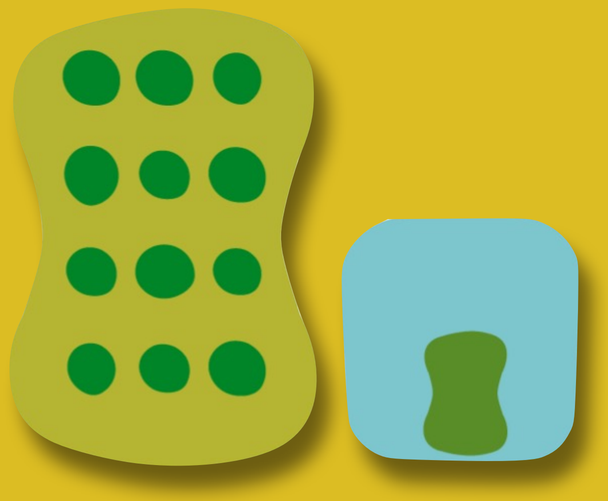
technology driven

business driven

technology driven

business driven

technology
driven

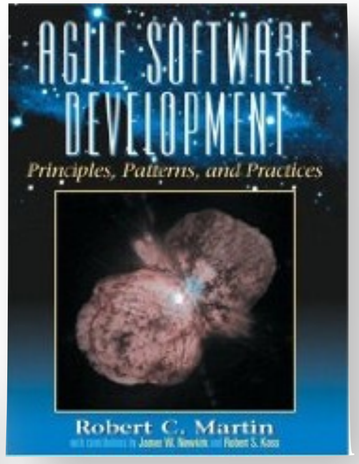


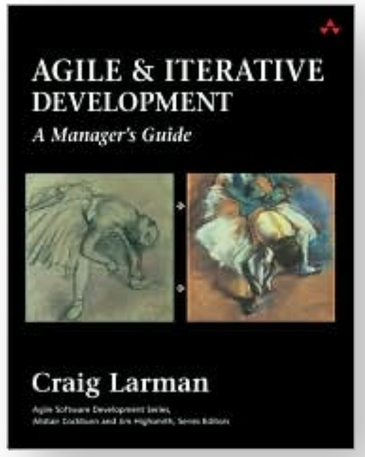
business driven

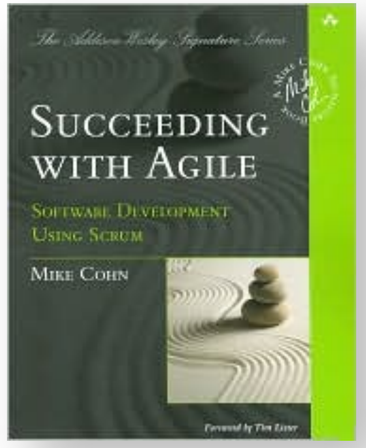
technology driven

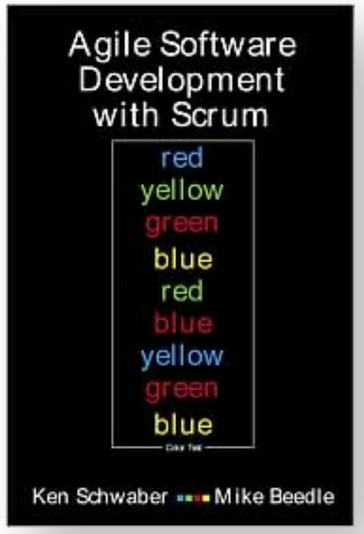
business driven

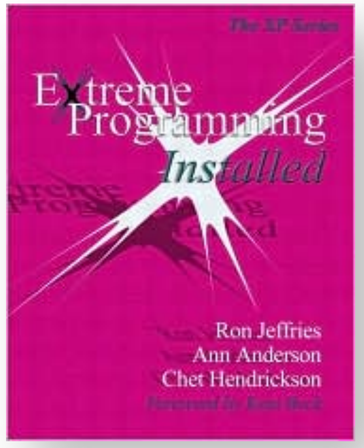
business driven

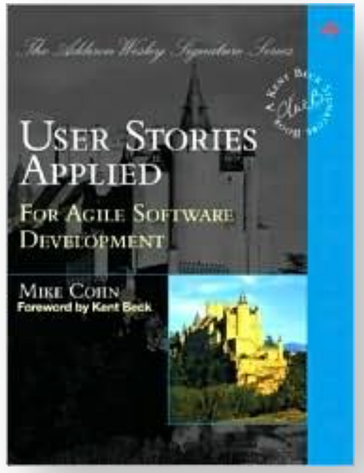


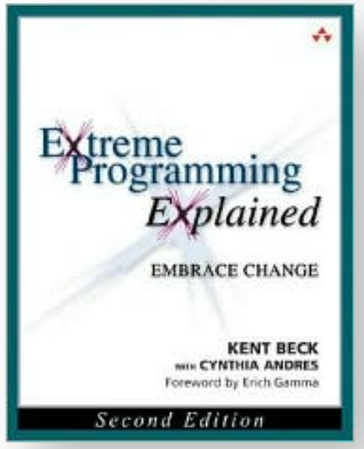


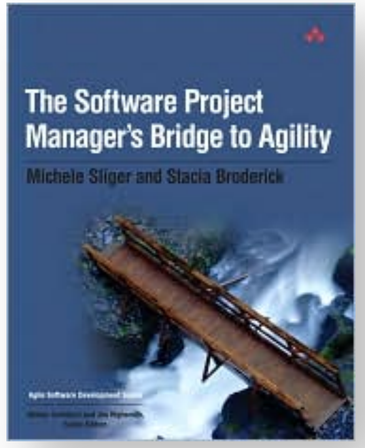












technology driven

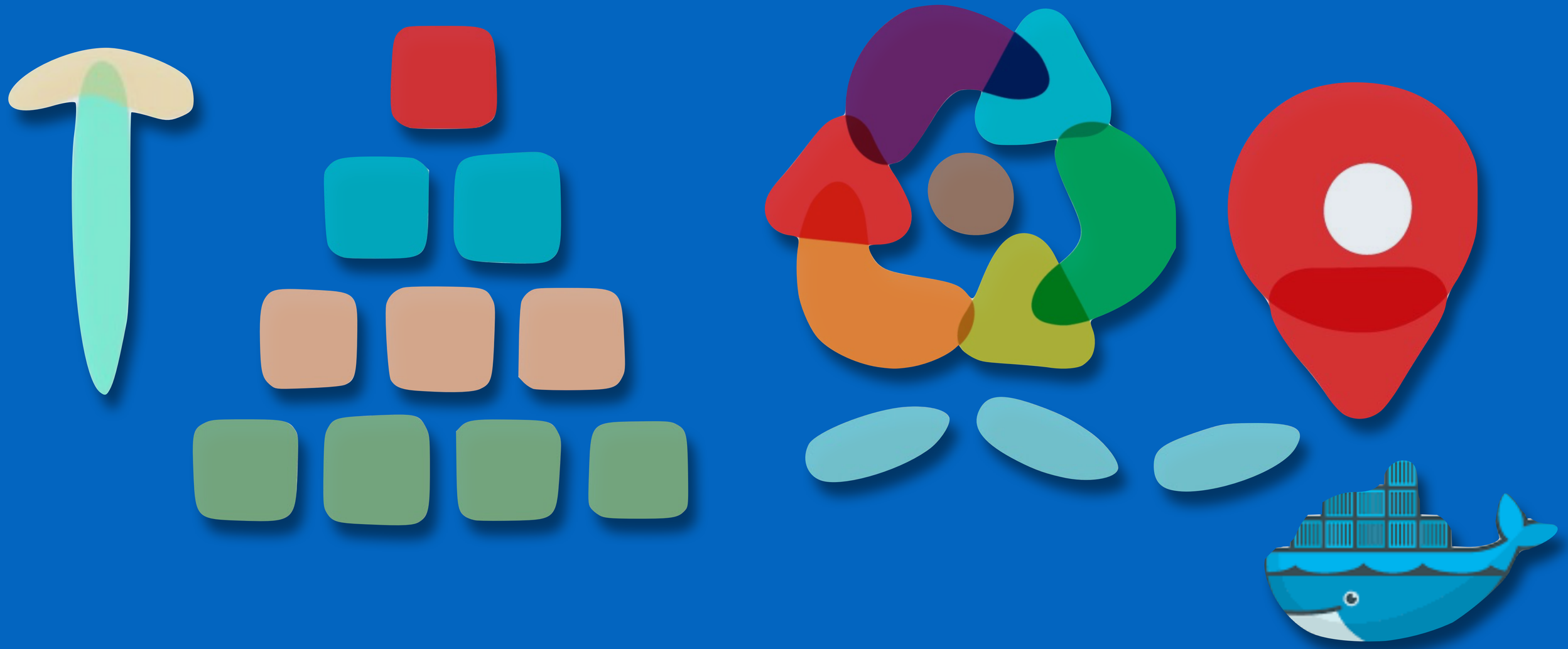
business driven

technology driven

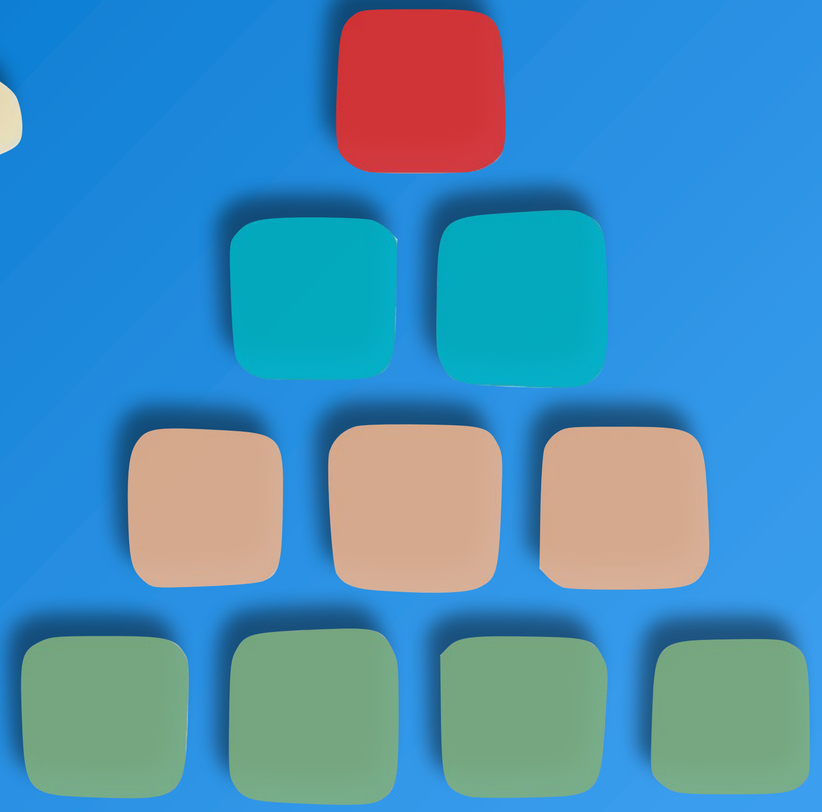
business driven

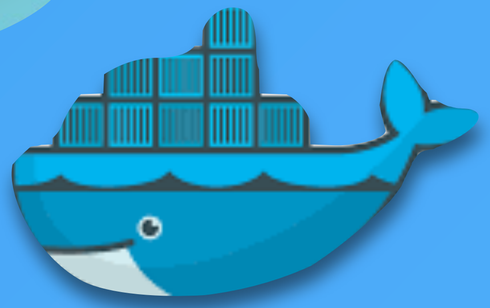
everything chAnGEs
all the time!

dYNaMic eqUiLiBRiuM



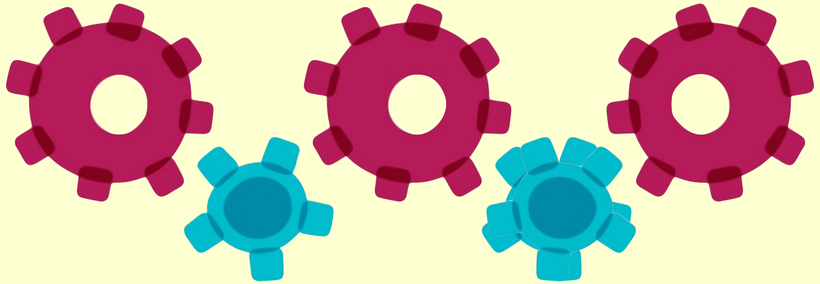
dYNaMic eqUiLiBRiuM



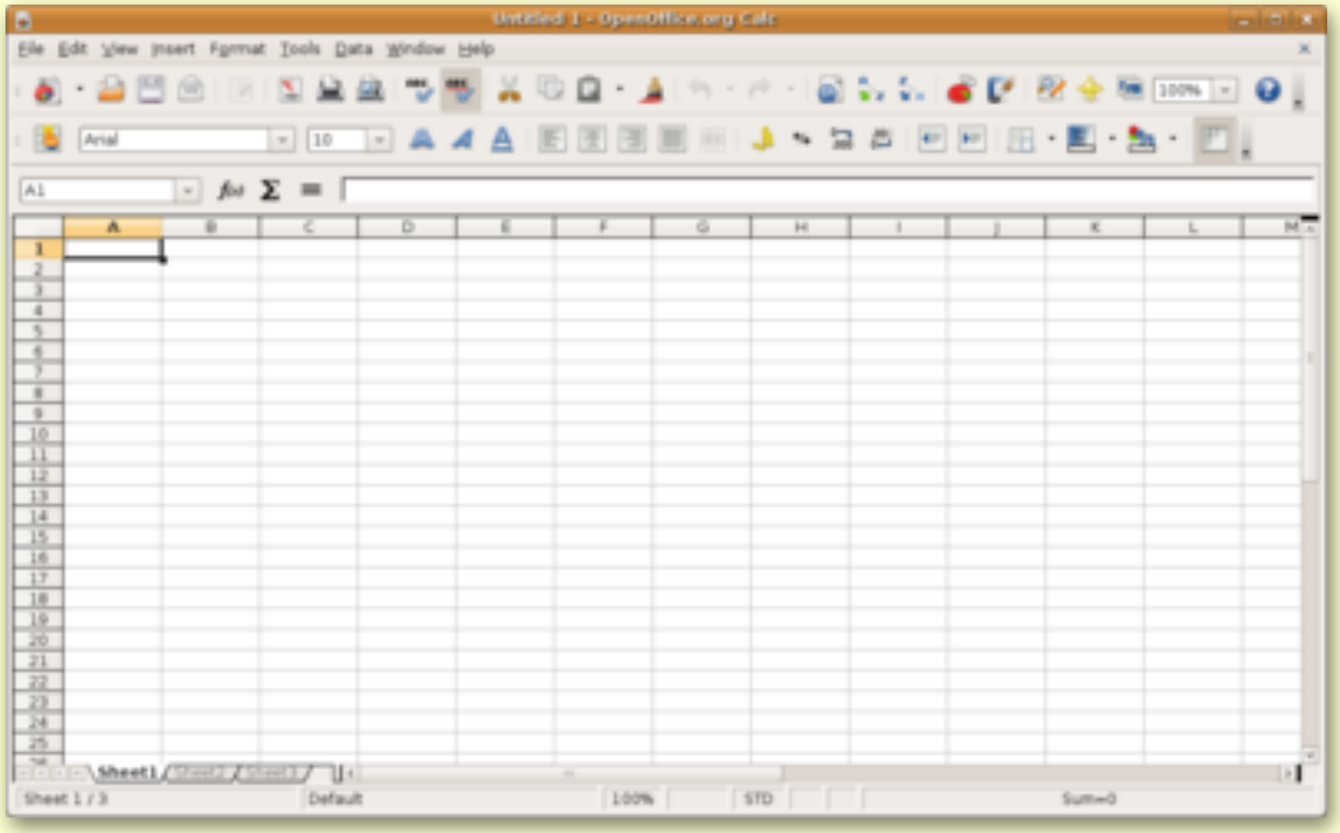


How is long term planning possible when things constantly change in unEXpECtEd ways?

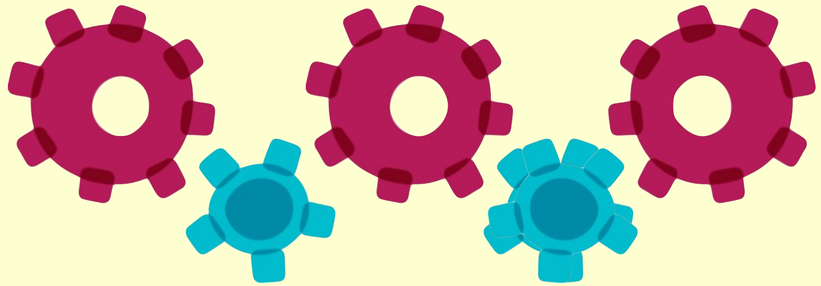
Penultima⬆☒e



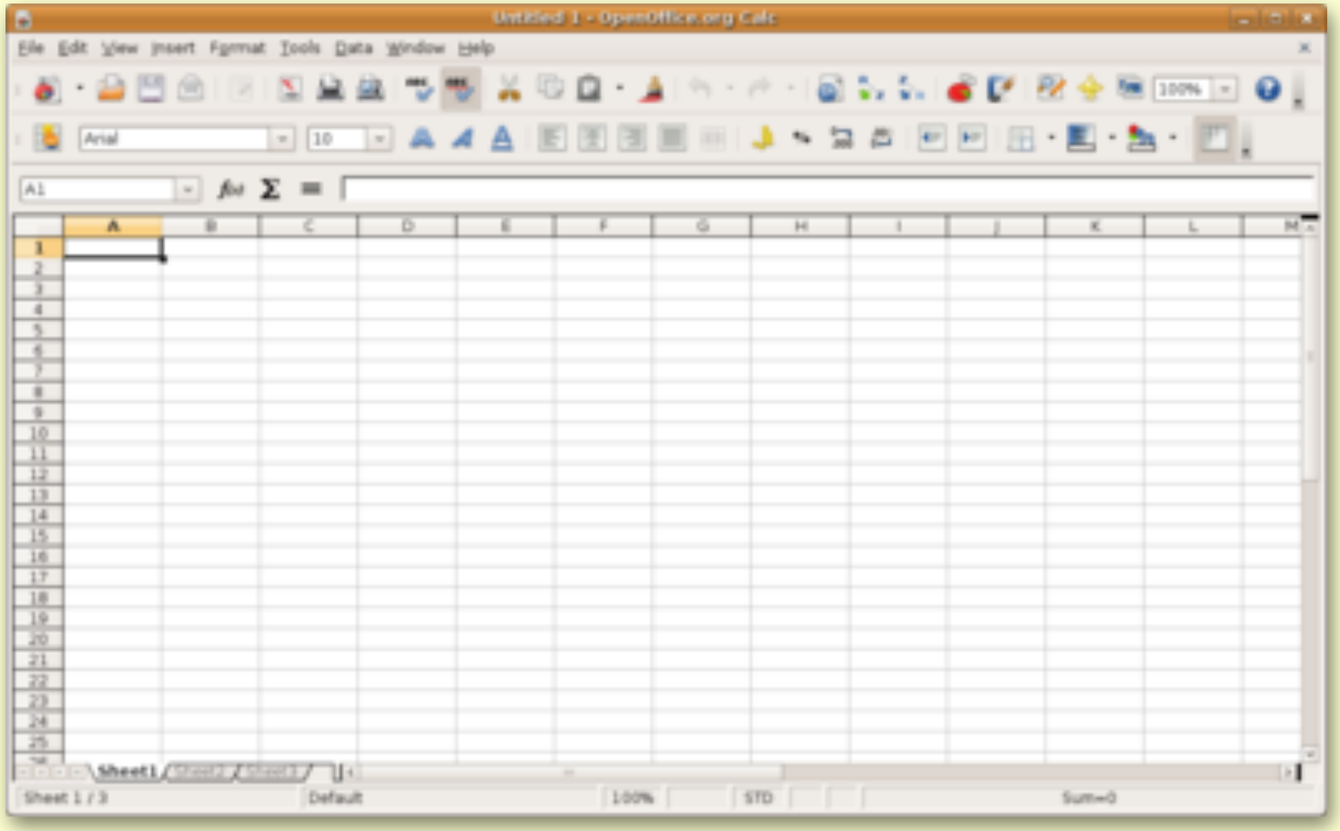
EA Spreadsheet



Penultima ⬆ ☒e

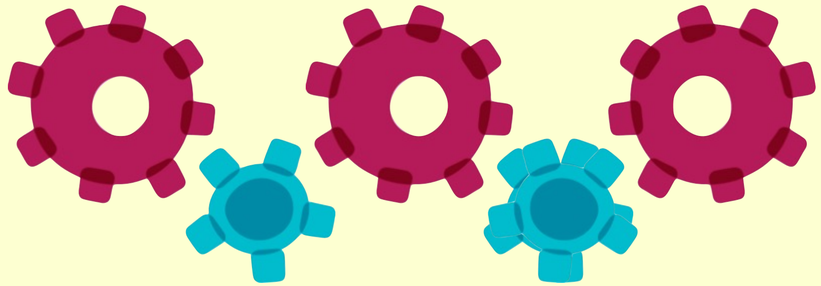


EA Spreadsheet

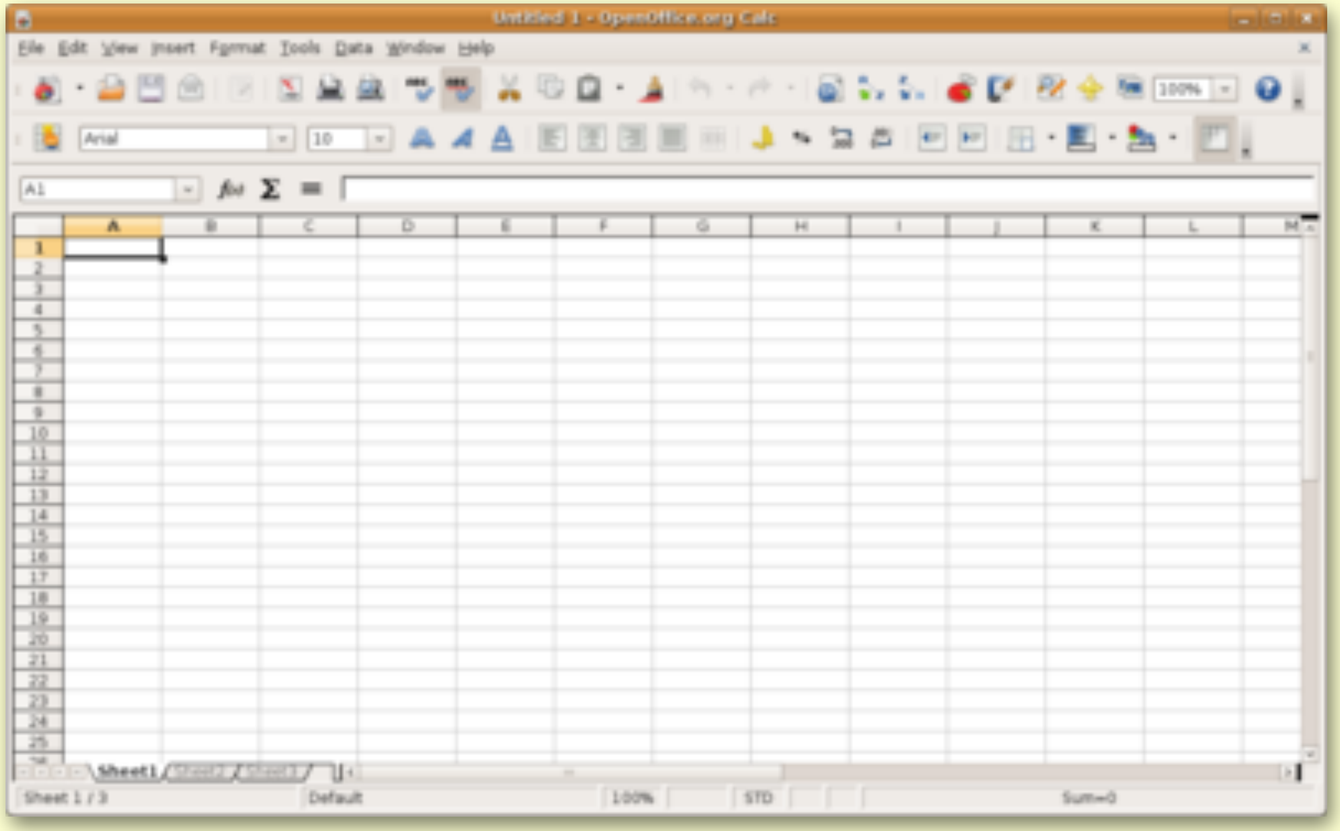


✔ definition

Penultima ⬆ ☒e



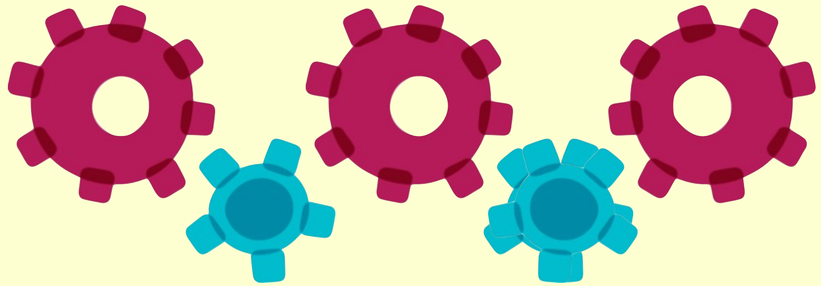
EA Spreadsheet



✔ definition

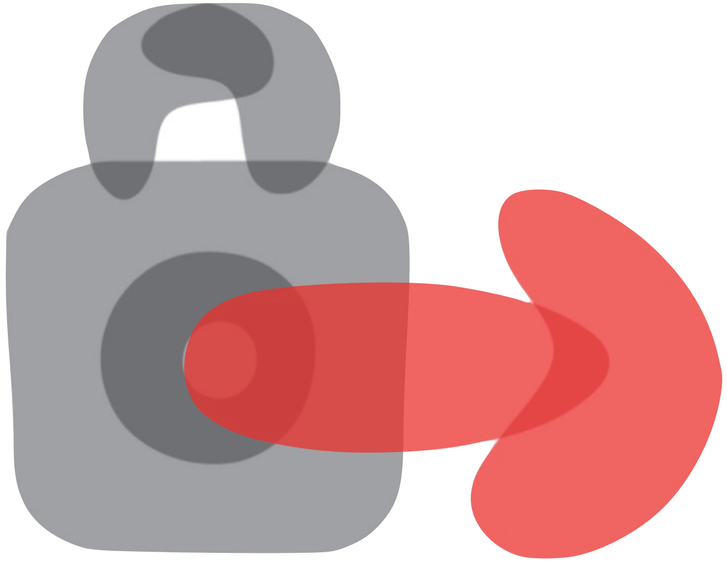
! **verification**

Penultima ⬆ ☒e



Once I've built an architecture, how can I prevent it from gradually dEgRAdiNg over time?

governance



Evolutionary Architecture

Continuous Architecture?

Evolutionary Architecture

Continuous Architecture?

Incremental Architecture?

Evolutionary Architecture

Continuous Architecture?

Incremental Architecture?

Agile Architecture?

Evolutionary Architecture

Continuous Architecture?

Incremental Architecture?

Agile Architecture?

Adaptable Architecture?

Evolutionary Architecture

Continuous Architecture?

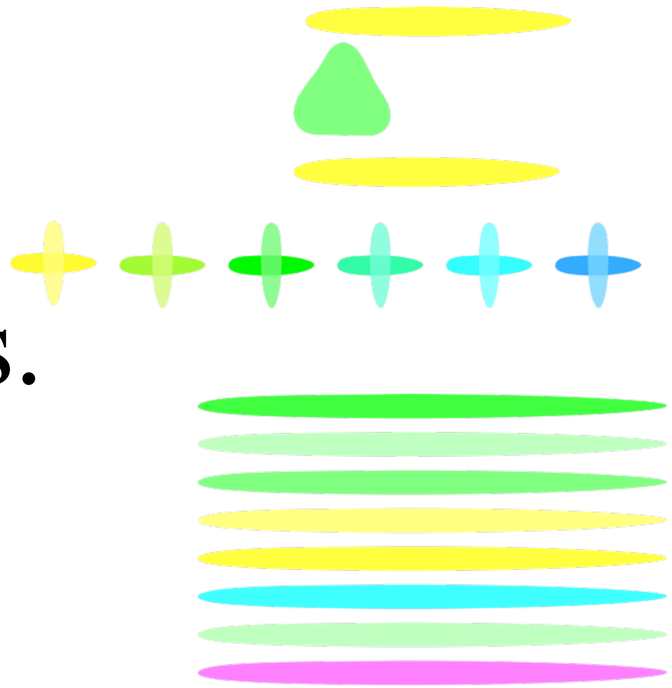
Incremental Architecture?

Agile Architecture?

Adaptable Architecture?

Evolutionary Architecture

An evolutionary architecture supports
guided,
incremental change
across multiple dimensions.



Evolutionary Architecture

An evolutionary architecture supports

`guided`
incremental change
across multiple dimensions.

guided

evolutionary computing fitness function:

a particular type of objective function that is used to summarize…how close a given design solution is to achieving the set aims.

Traveling Salesperson Problem

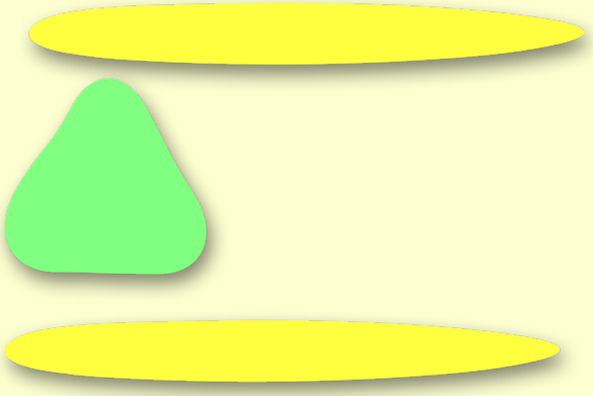
fitness function = length of route

guided

<u>architectural</u> *<u>fitness function</u>*:

An architectural fitness function provides an objective integrity assessment of some architectural characteristic(s).

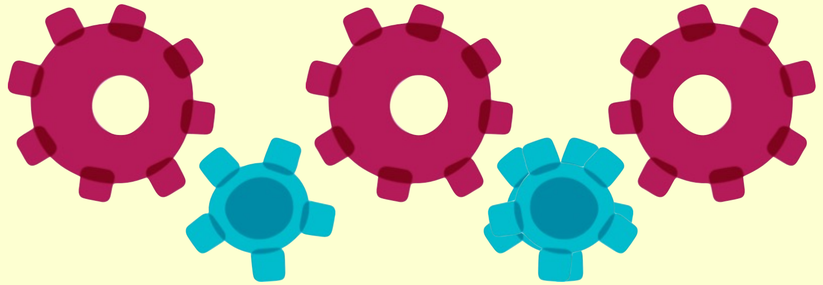
EA Spreadsheet



✓ definition

✓ `verification`

Penultima⬆⊠e



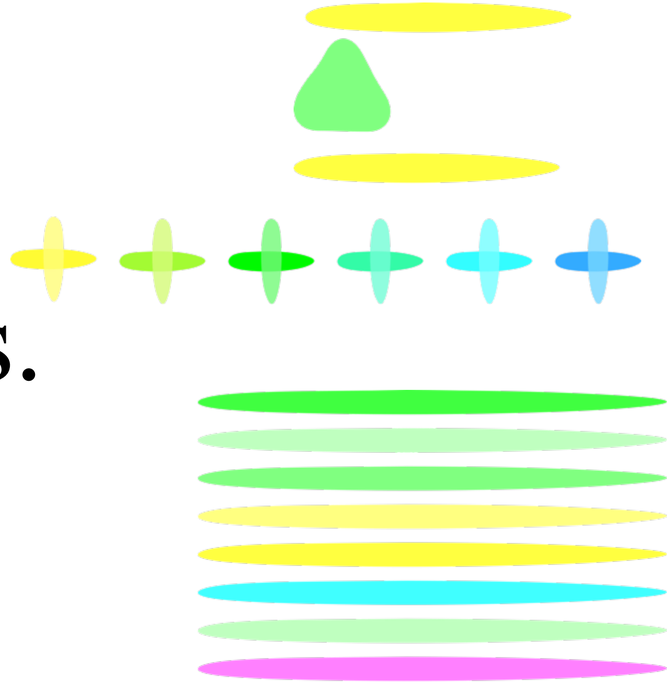
Evolutionary Architecture

An evolutionary architecture supports

`guided`

incremental change

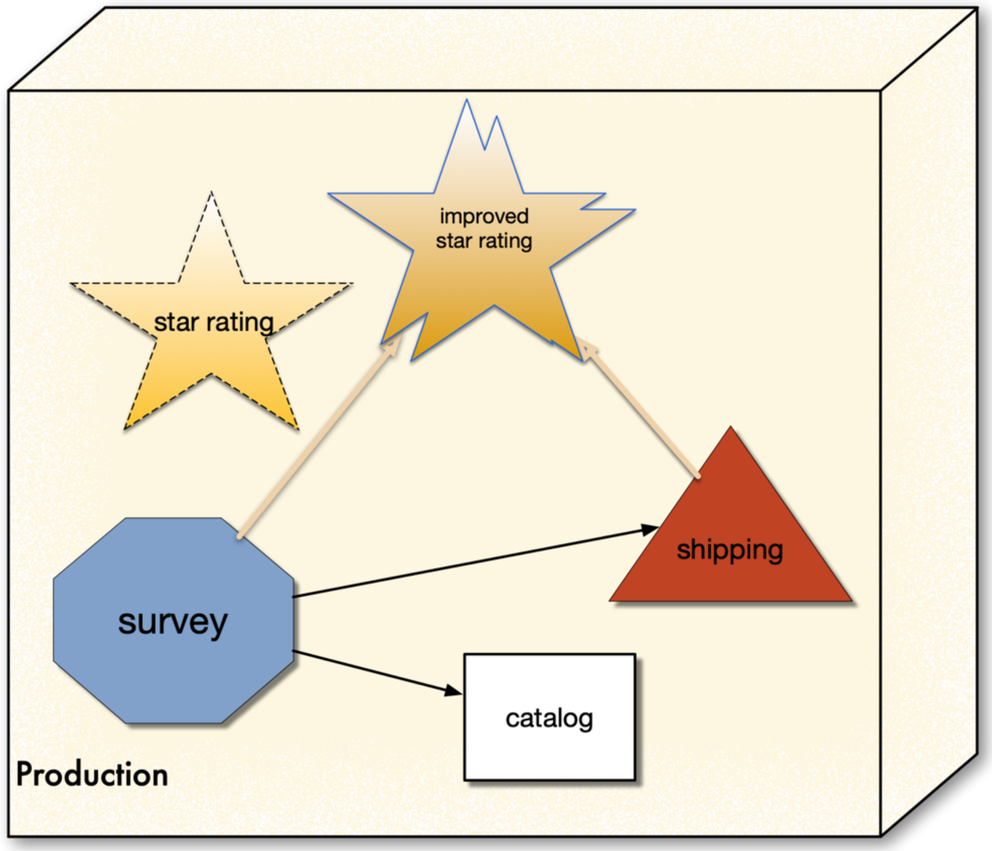
across multiple dimensions.

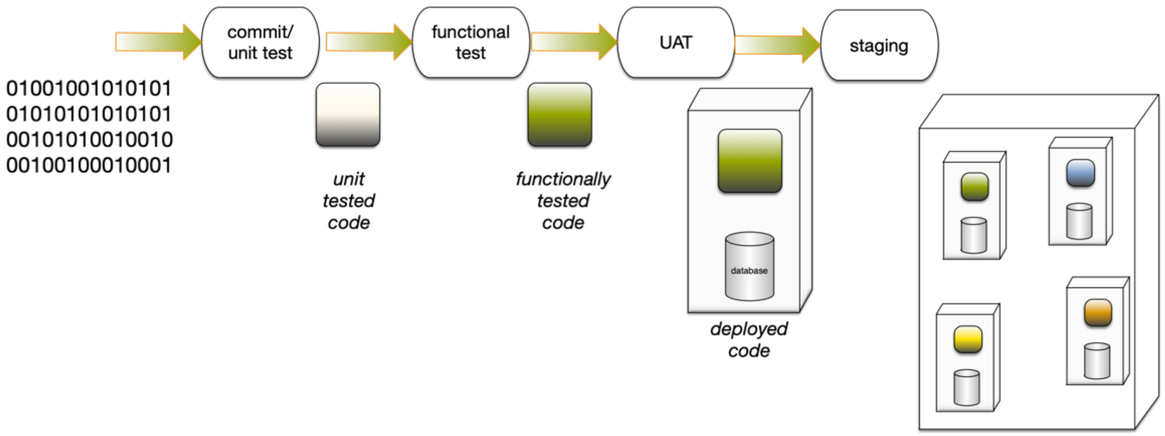


Evolutionary Architecture

An evolutionary architecture supports
guided,
incremental change
across multiple dimensions.

incremental





Evolutionary Architecture

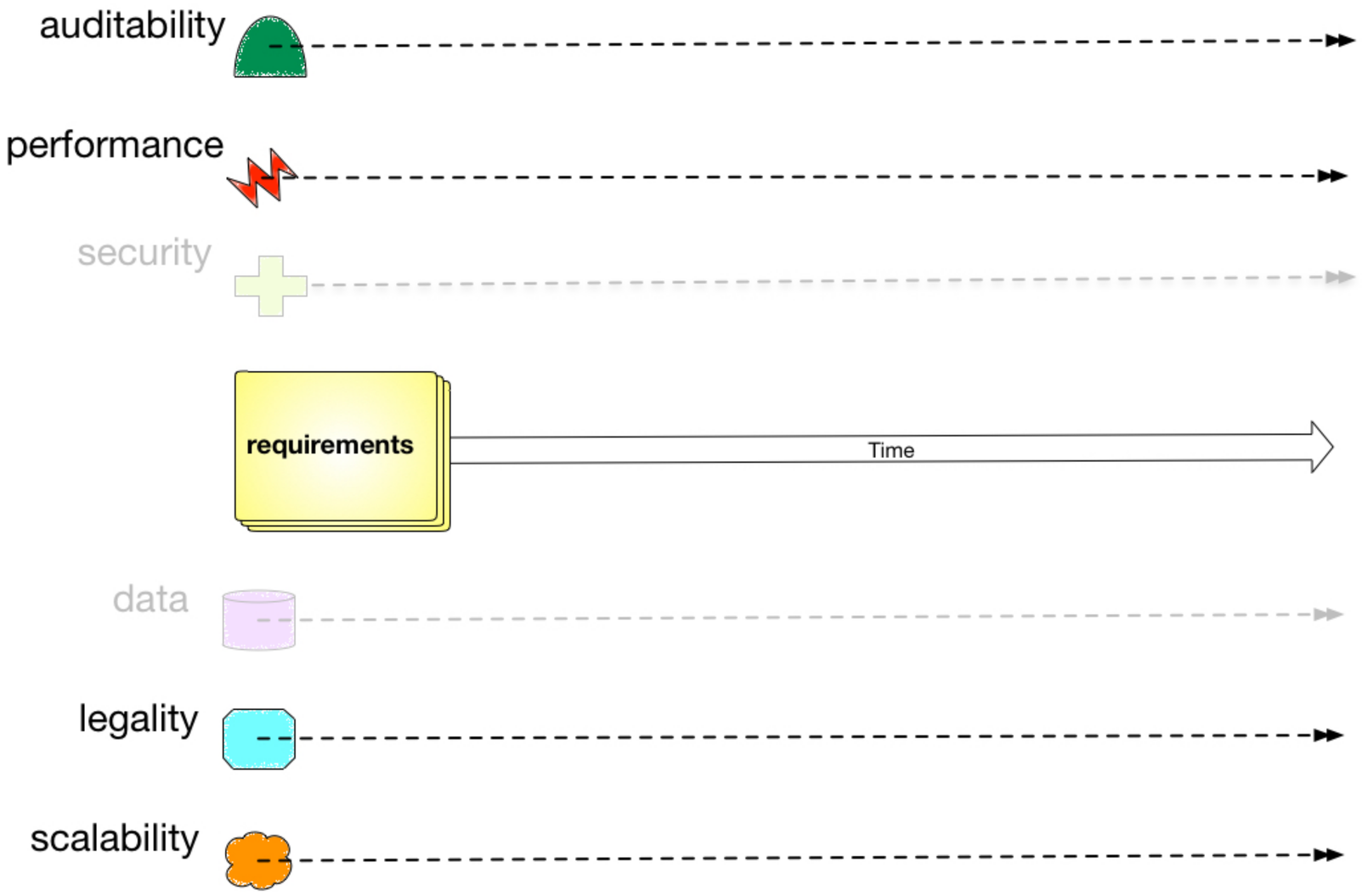
An evolutionary architecture supports
guided,
incremental change
across multiple dimensions.

`incremental`

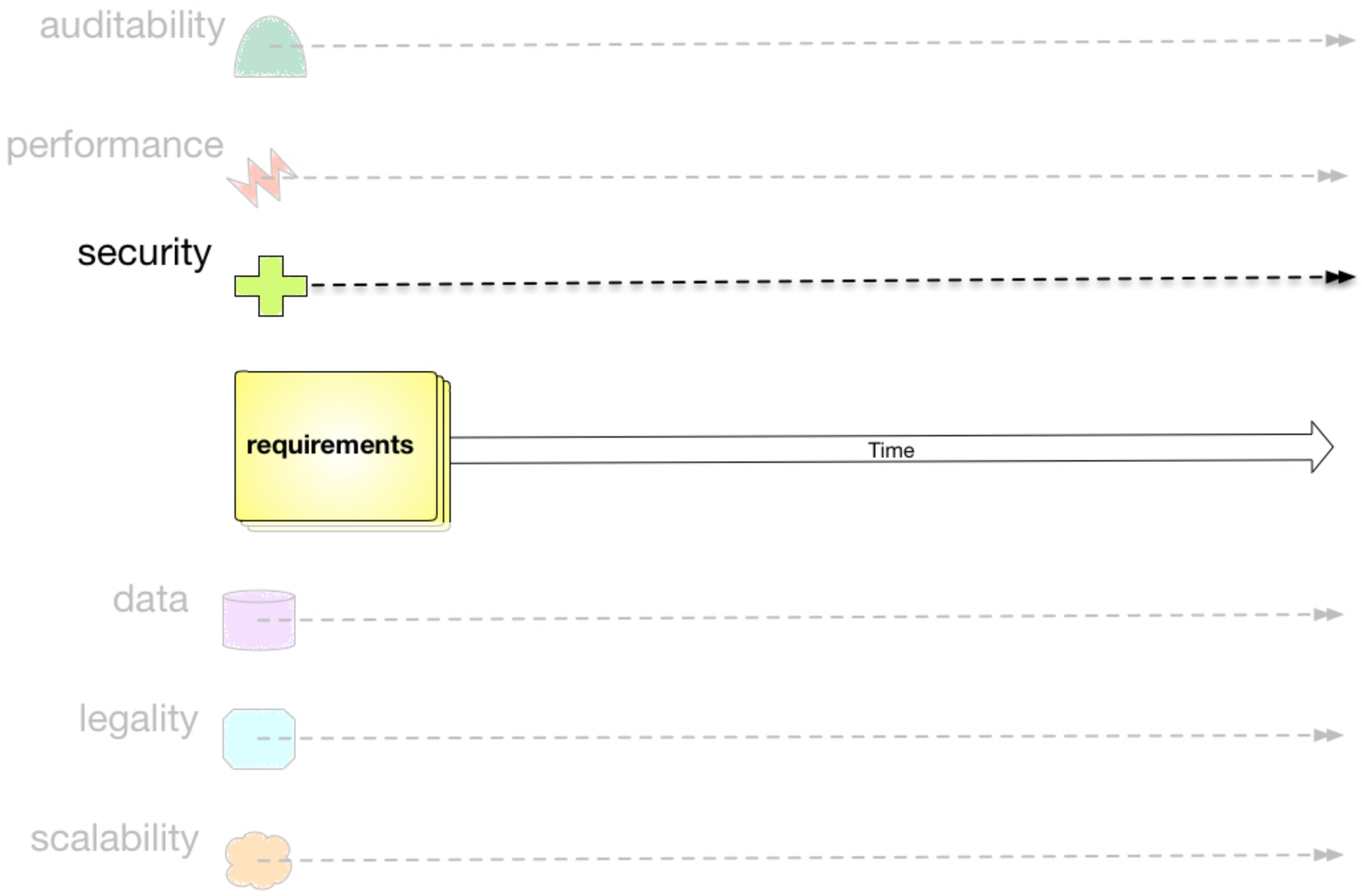
Evolutionary Architecture

An evolutionary architecture supports
guided,
incremental change
across **multiple dimensions**

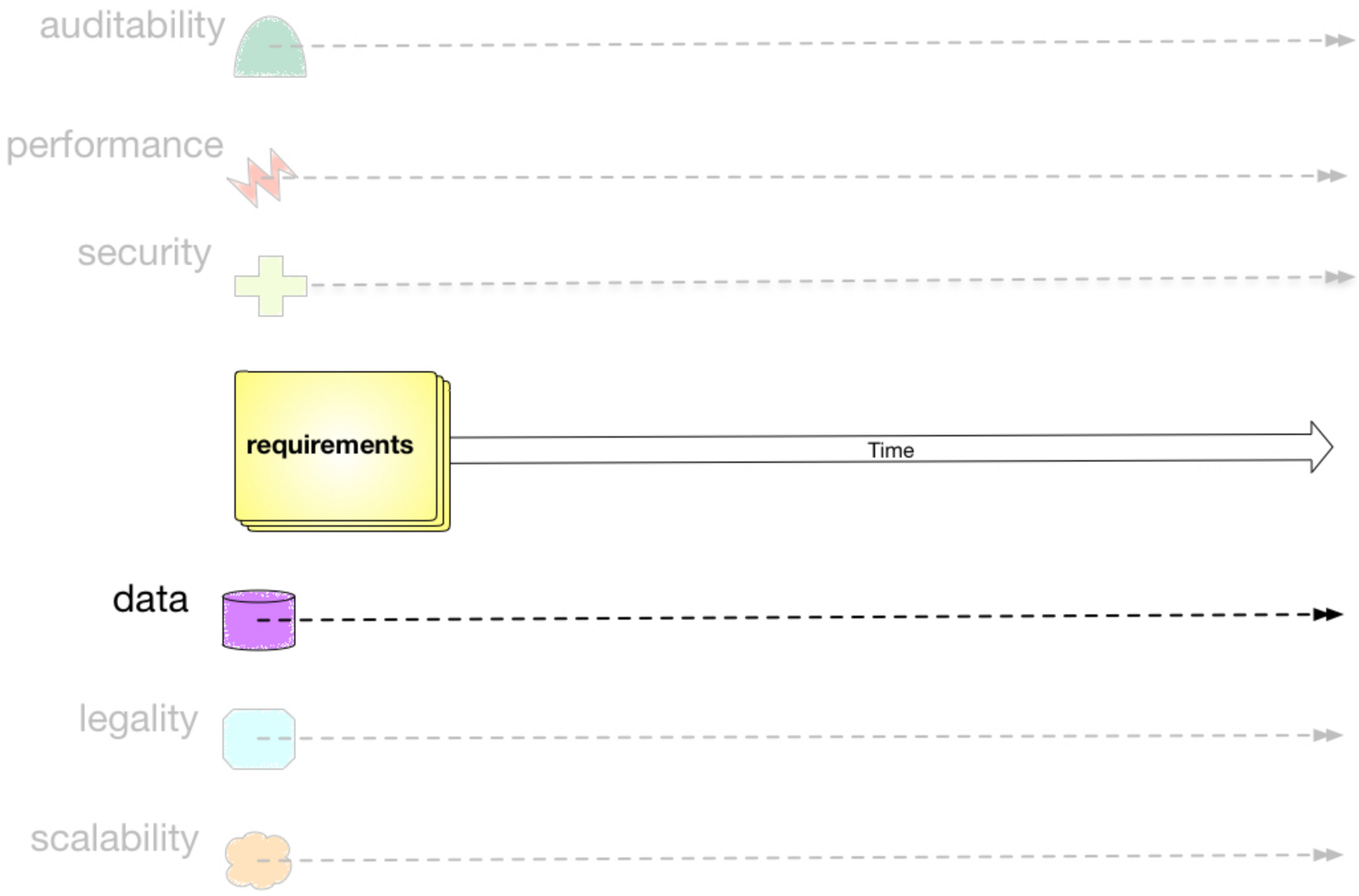
multiple dimensions



multiple dimensions



multiple dimensions

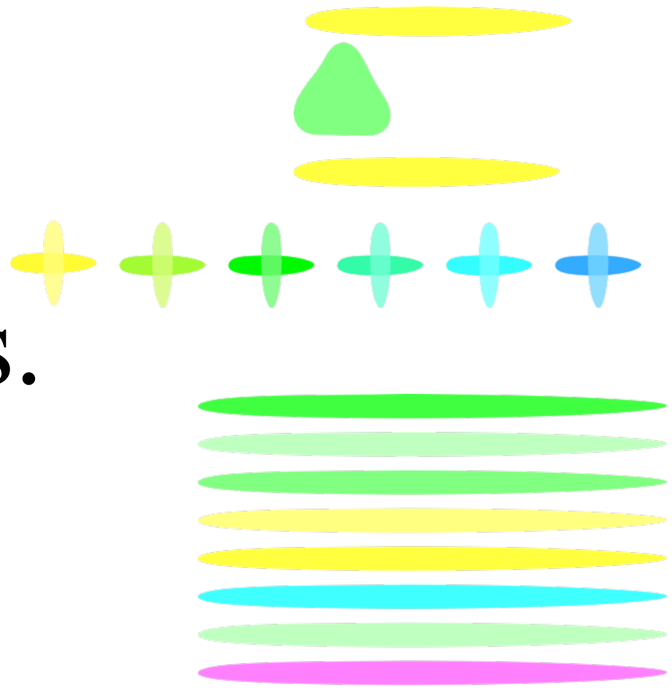


Evolutionary Architecture

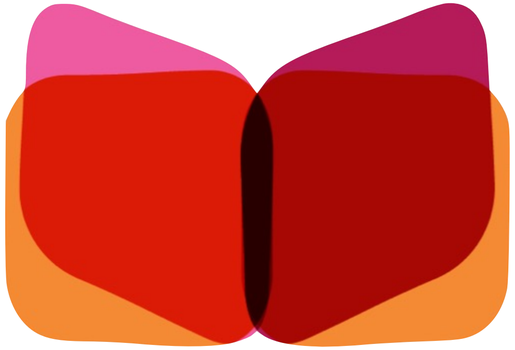
An evolutionary architecture supports
guided,
incremental change
across **multiple dimensions**

Evolutionary Architecture

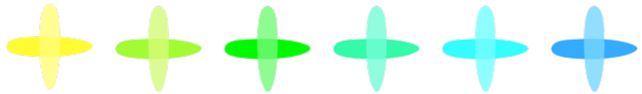
An evolutionary architecture supports
guided,
incremental change
across multiple dimensions.



Agenda Part 1

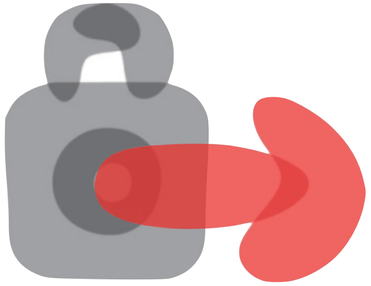


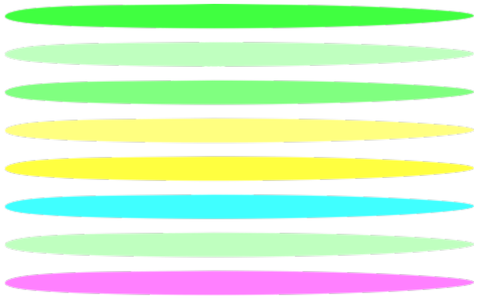
Definition



Incremental Change

Guided Change via Fitness Functions

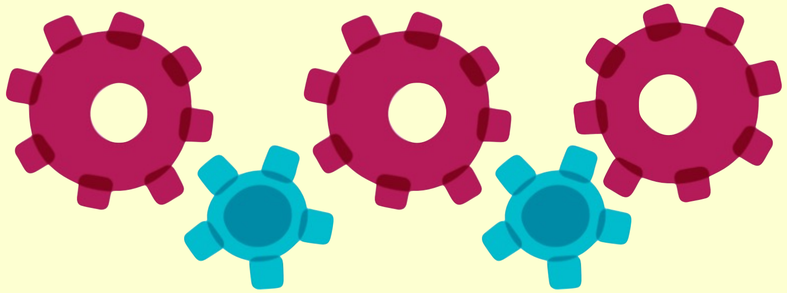




Structuring architecture for evolution

Agenda Part 2

The Evolution of Penultima⬆⊠e



Agenda Part 2

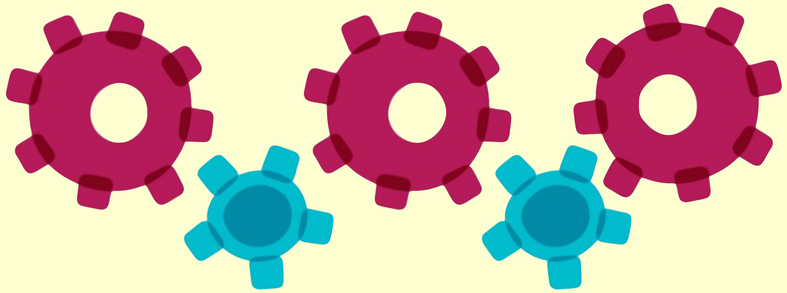
fitness functions

architecture migration

The Evolution of

Penultima⬆⊠e

experimentation



automating governance

hypothesis/data driven development

Evolutionary Architecture

An evolutionary architecture supports

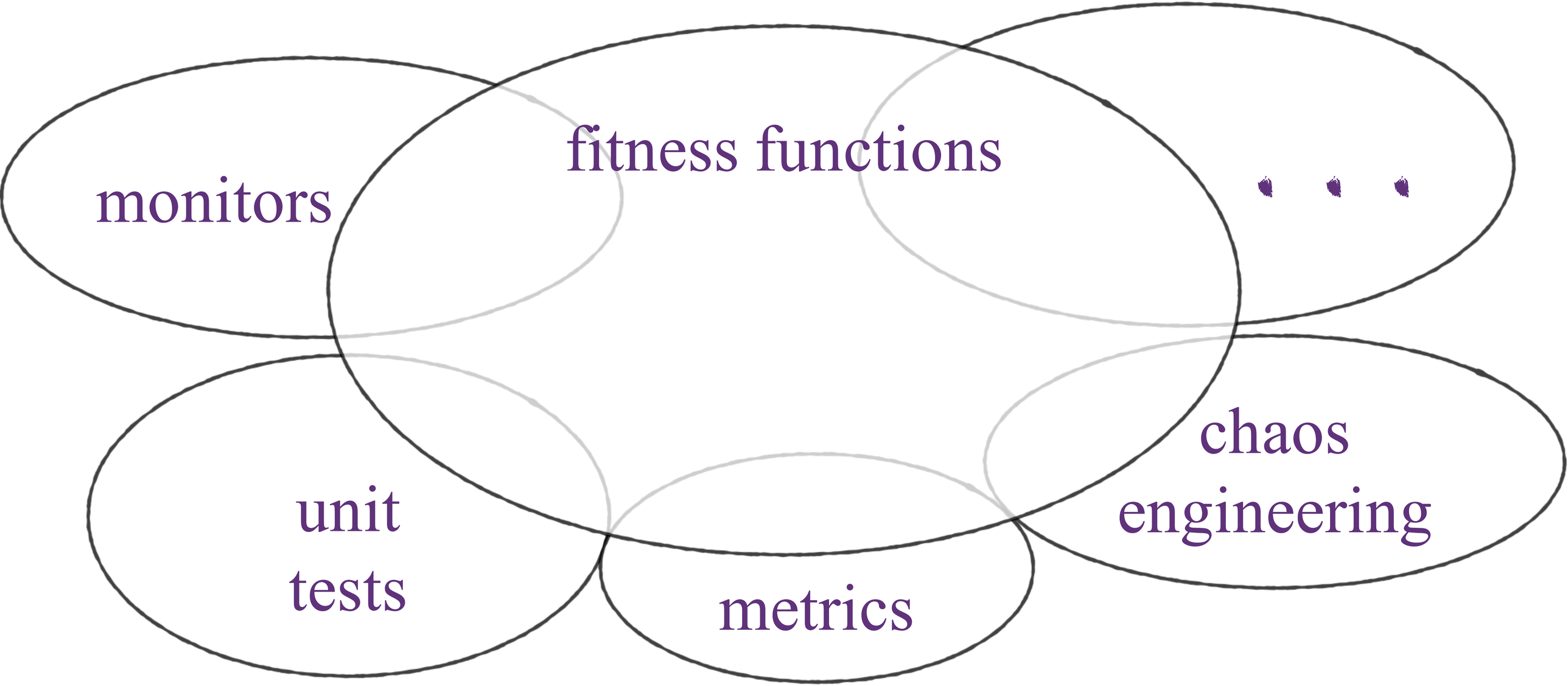
`guided`
incremental change
across multiple dimensions.

guided

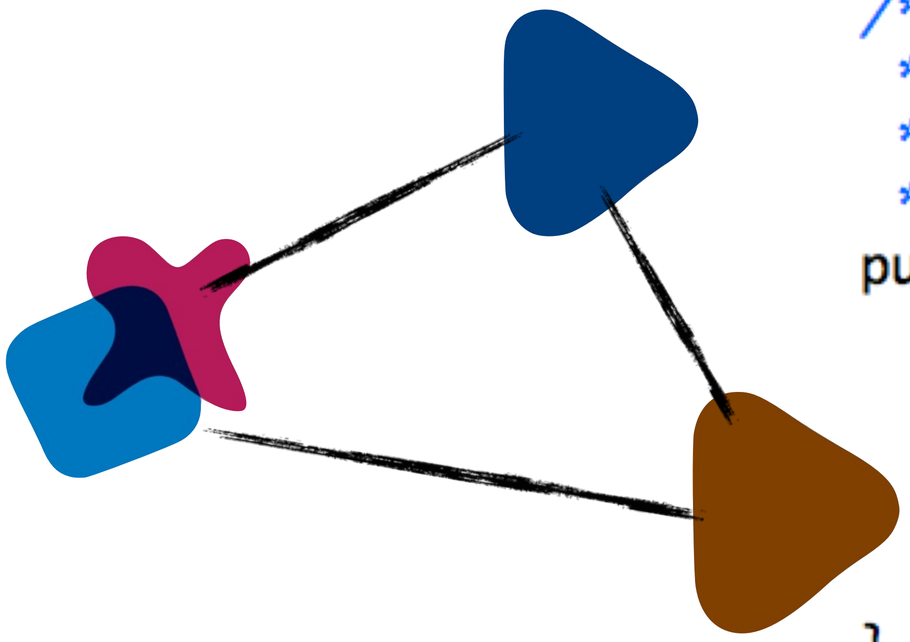
<u>architectural</u> *<u>fitness function</u>*:

An architectural fitness function provides an objective integrity assessment of some architectural characteristic(s).

Fitness Functions



Cyclic Dependency Function



```
/**
 * Tests that a package dependency cycle does not
 * exist for any of the analyzed packages.
 */
public void testAllPackages() {

    Collection packages = jdepend.analyze();

    assertEquals("Cycles exist",
                 false, jdepend.containsCycles());
}
```

clarkware.com/software/JDepend.html

guided

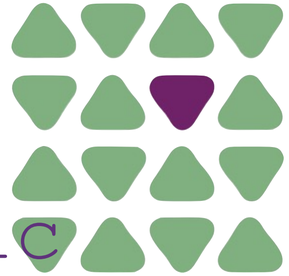
<u>architectural</u> *<u>fitness function</u>*:

An architectural fitness function provides an objective integrity assessment of some architectural characteristic(s).

Categories: Scope

atomic

run against a singular context and exercise one particular aspect of the architecture.



holistic

run against a shared context and exercise a combination of architectural aspects such as security and scalability

Categories: Invocation

triggered

run based on a particular event:
— developer executing a unit test
— deployment pipeline running tests
— timed task

continuous

executes constant verification of architectural aspect(s)

Categories: Return

`static`

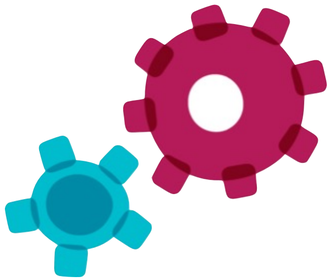
have a fixed result, such as the binary pass/fail of a unit test.

`dynamic`

rely on a shifting definition based on extra context.

Categories: Automation

`automated`



tests and other verification mechanism that run without human interaction.

`manual`

must involve at least one human.

Categories of Fitness Functions

temporal

architects may want to build a time component into assessing fitness

Categories of Fitness Functions

domain-specific

Some architectures have specific concerns, such as special security or regulatory requirements

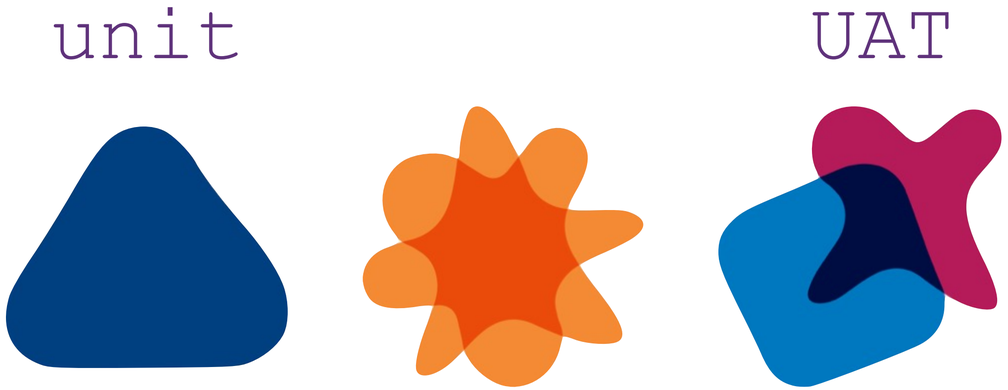
Categories of Fitness Functions

architectural
characteristic

Categories of Fitness Functions

architectural characteristic

problem domain



Categories of Fitness Functions

architects will define most fitness functions at project inception as they elucidate the characteristics of the architecture…

Categories:Origin

intentional

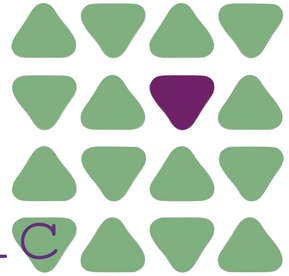
architects will define most fitness functions at project inception as they elucidate the characteristics of the architecture…

emergent

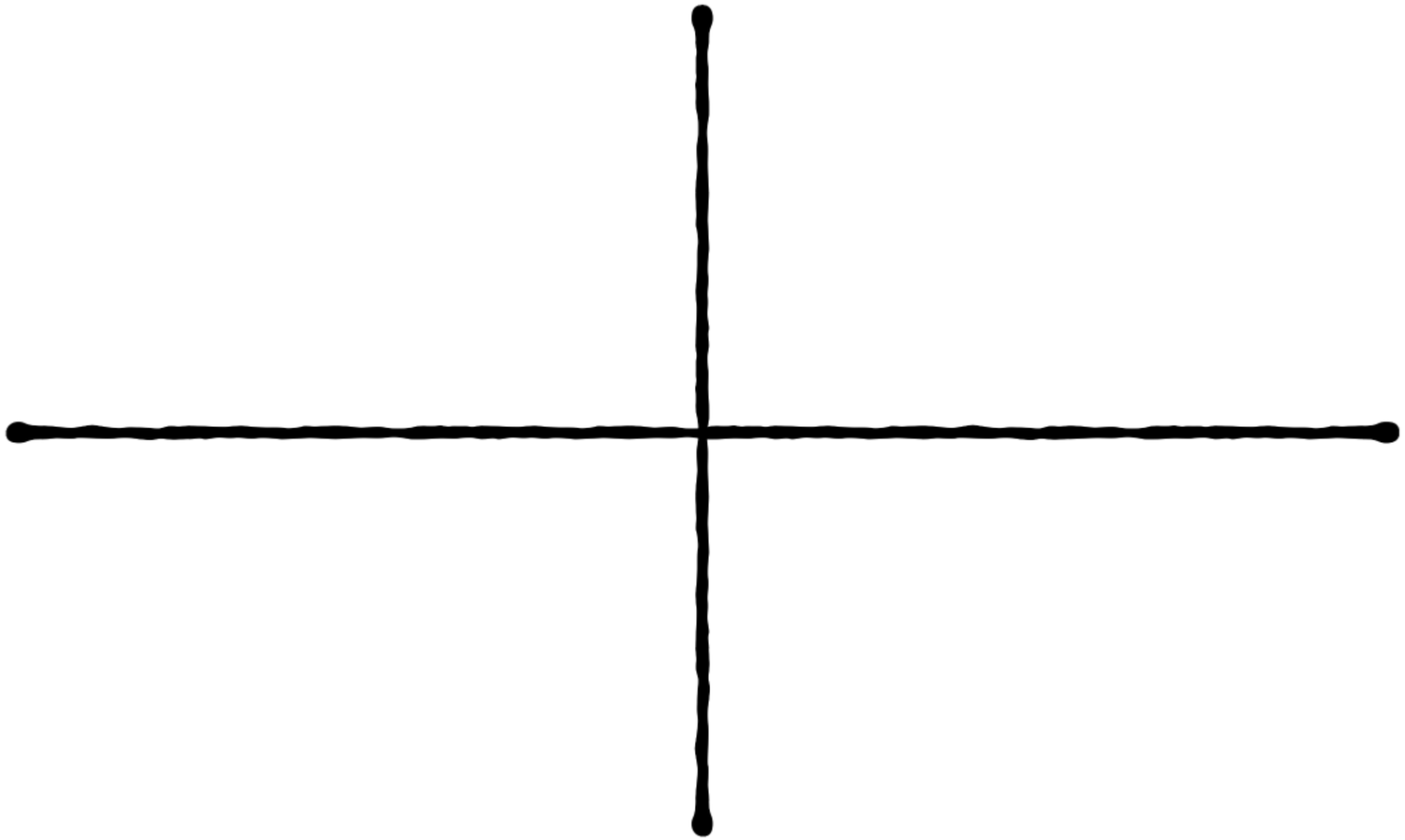
…some fitness functions will emerge during development of the system

Fitness Function

atomic



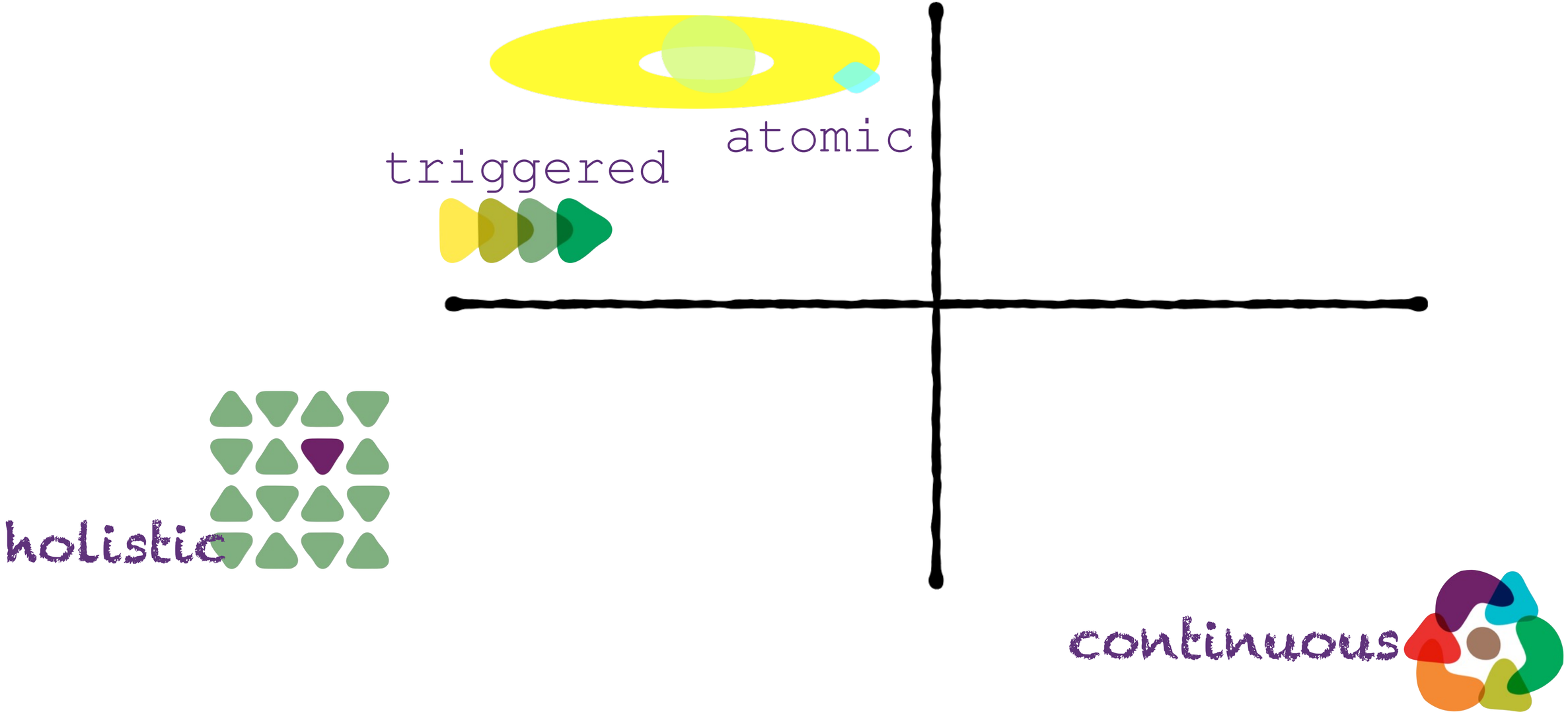
holistic



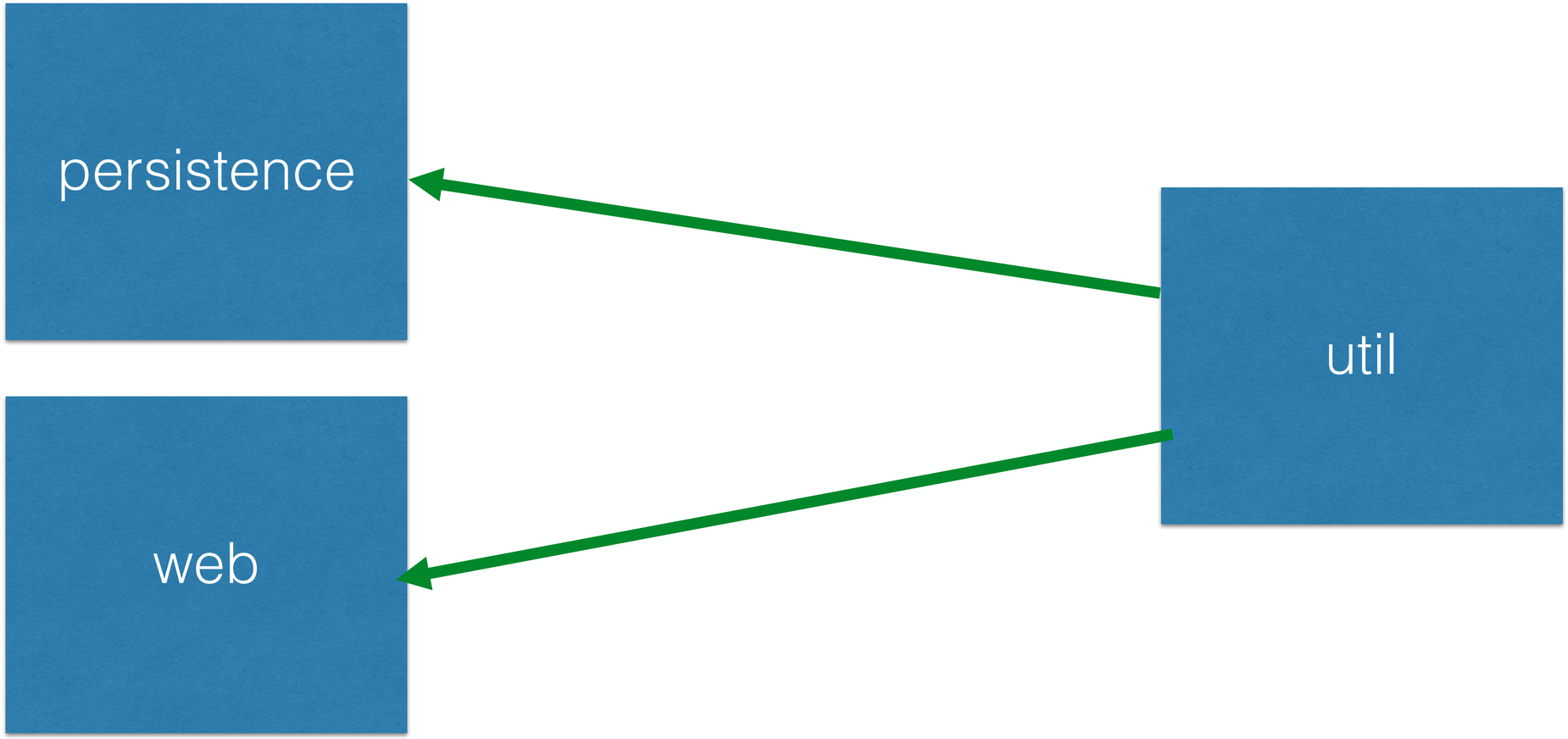
triggered

continuous

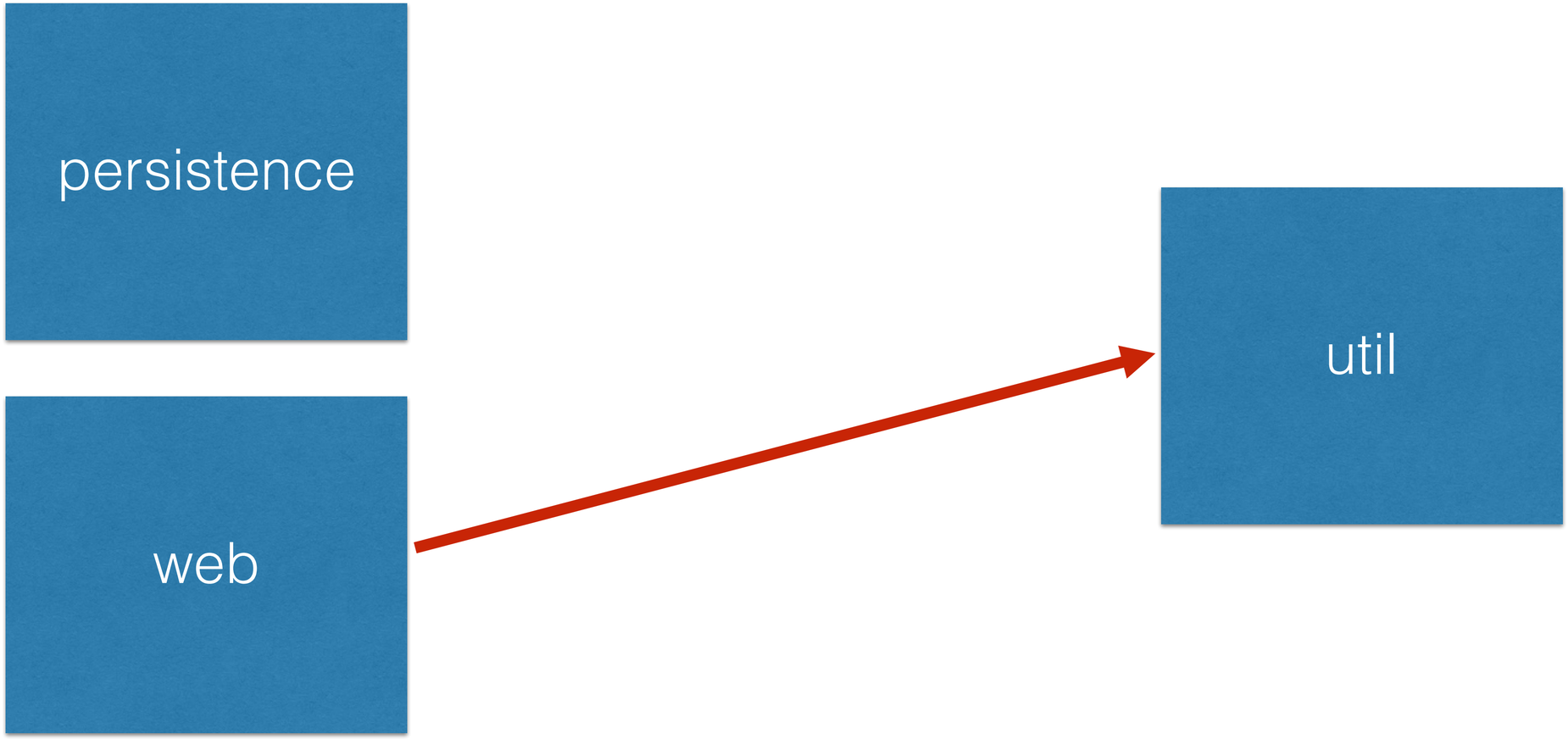
Fitness Function



Directionality of Imports



Directionality of Imports

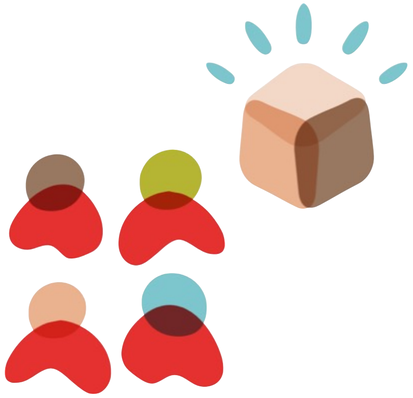


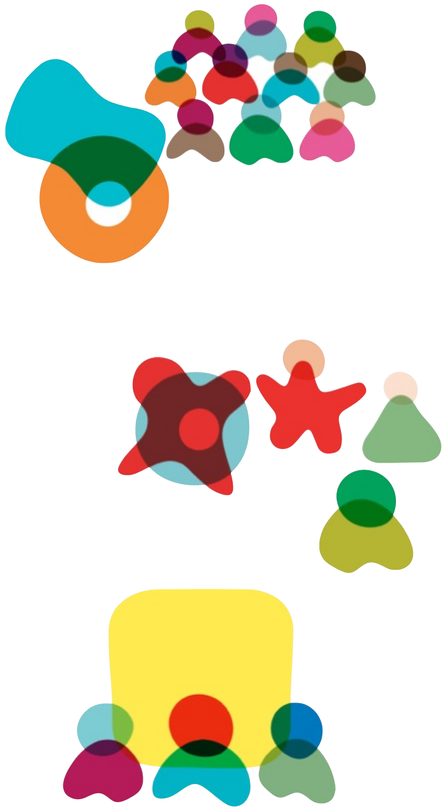
packages/namespaces

Coupling Fitness Function

```
public void testMatch() {
    DependencyConstraint constraint = new DependencyConstraint();

    JavaPackage persistence = constraint.addPackage("com.xyz.persistence");
    JavaPackage web = constraint.addPackage("com.xyz.web");
    JavaPackage util = constraint.addPackage("com.xyz.util");

    persistence.dependsUpon(util);
    web.dependsUpon(util);

    jdepend.analyze();

    assertEquals("Dependency mismatch",
            true, jdepend.dependencyMatch(constraint));
    }
```

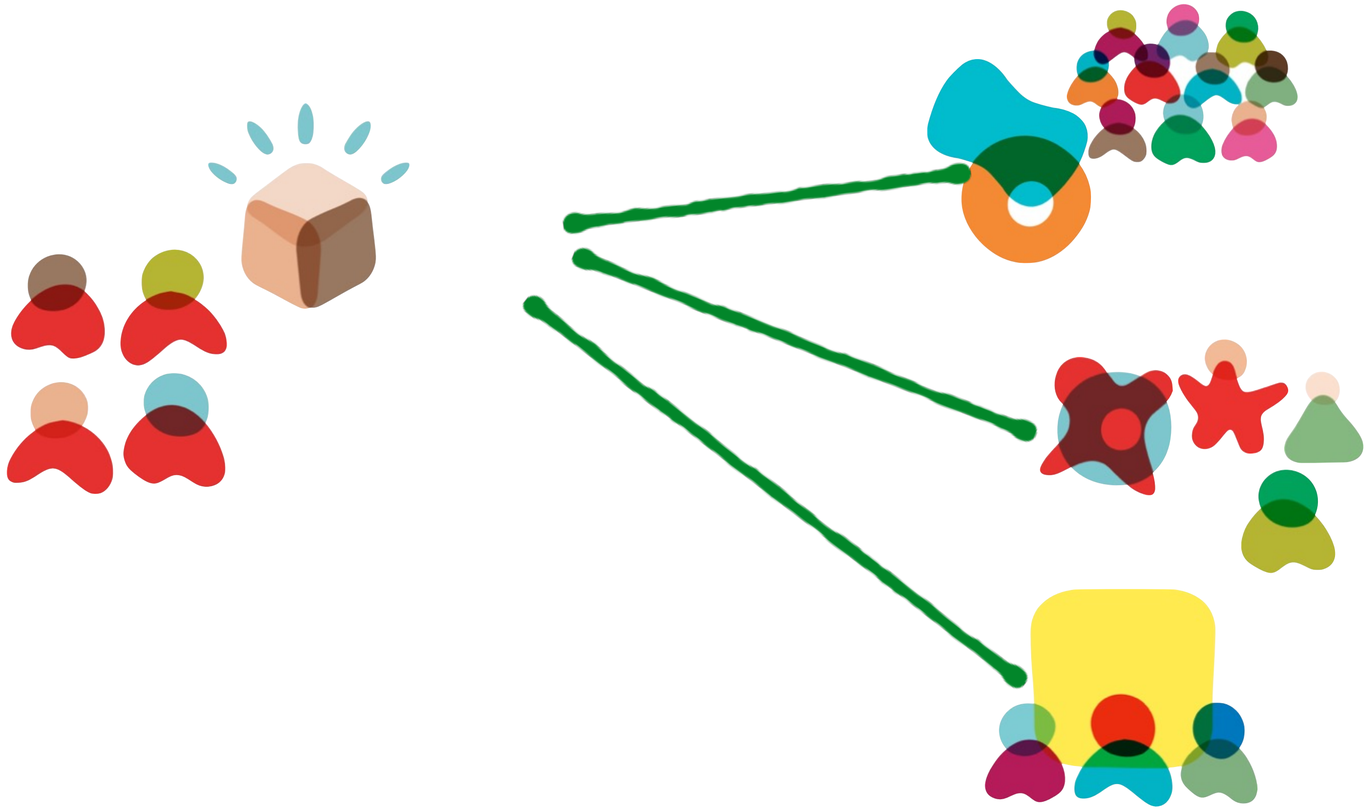
Consumer Driven Contracts





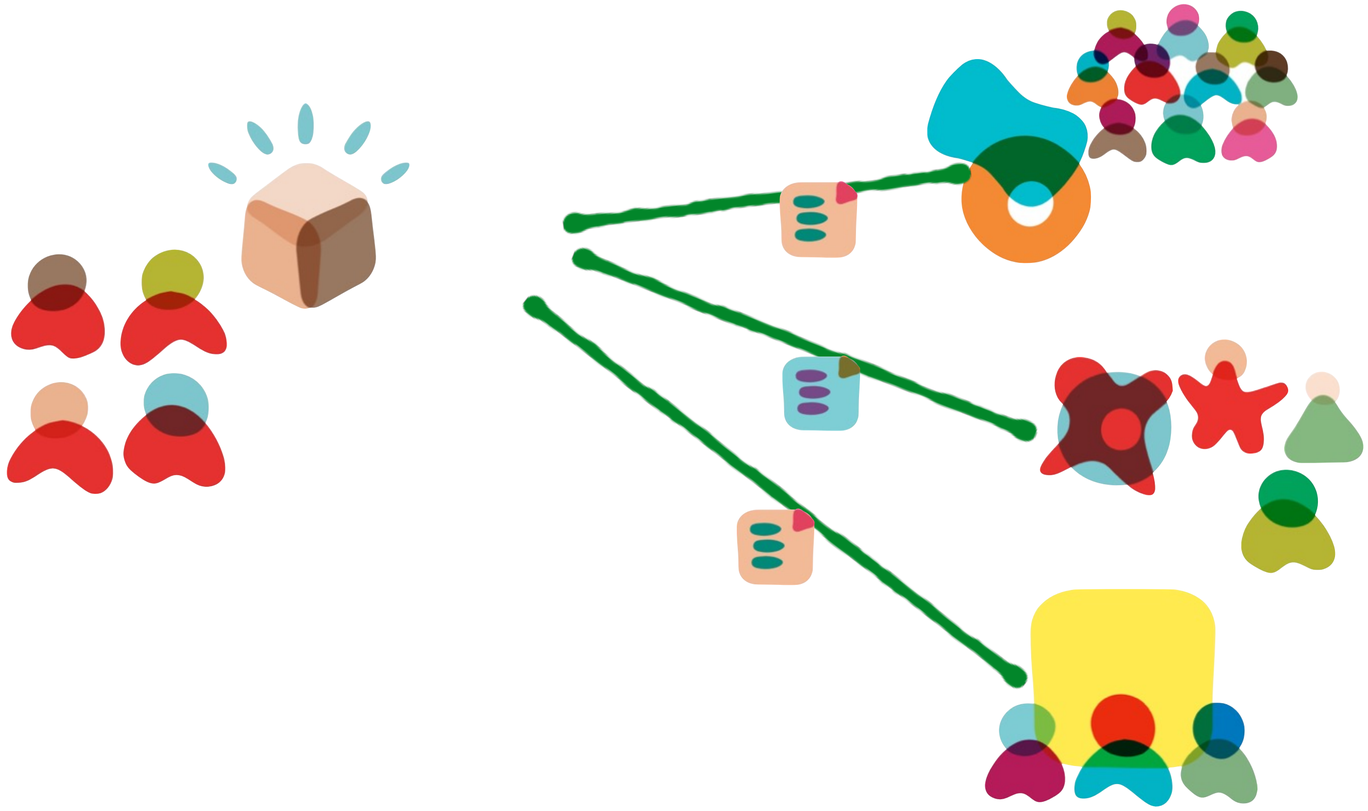
martinfowler.com/articles/consumerDrivenContracts.html

Consumer Driven Contracts



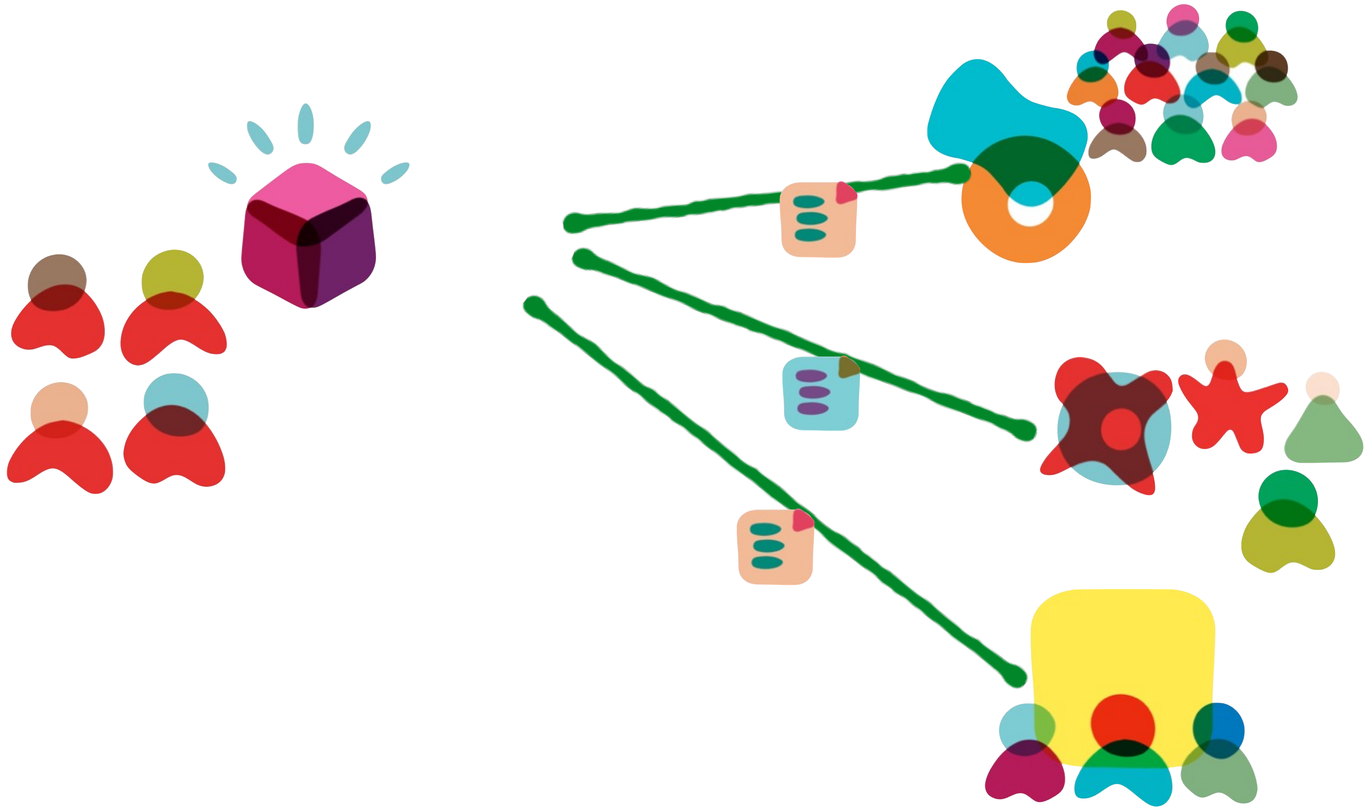
martinfowler.com/articles/consumerDrivenContracts.html

Consumer Driven Contracts



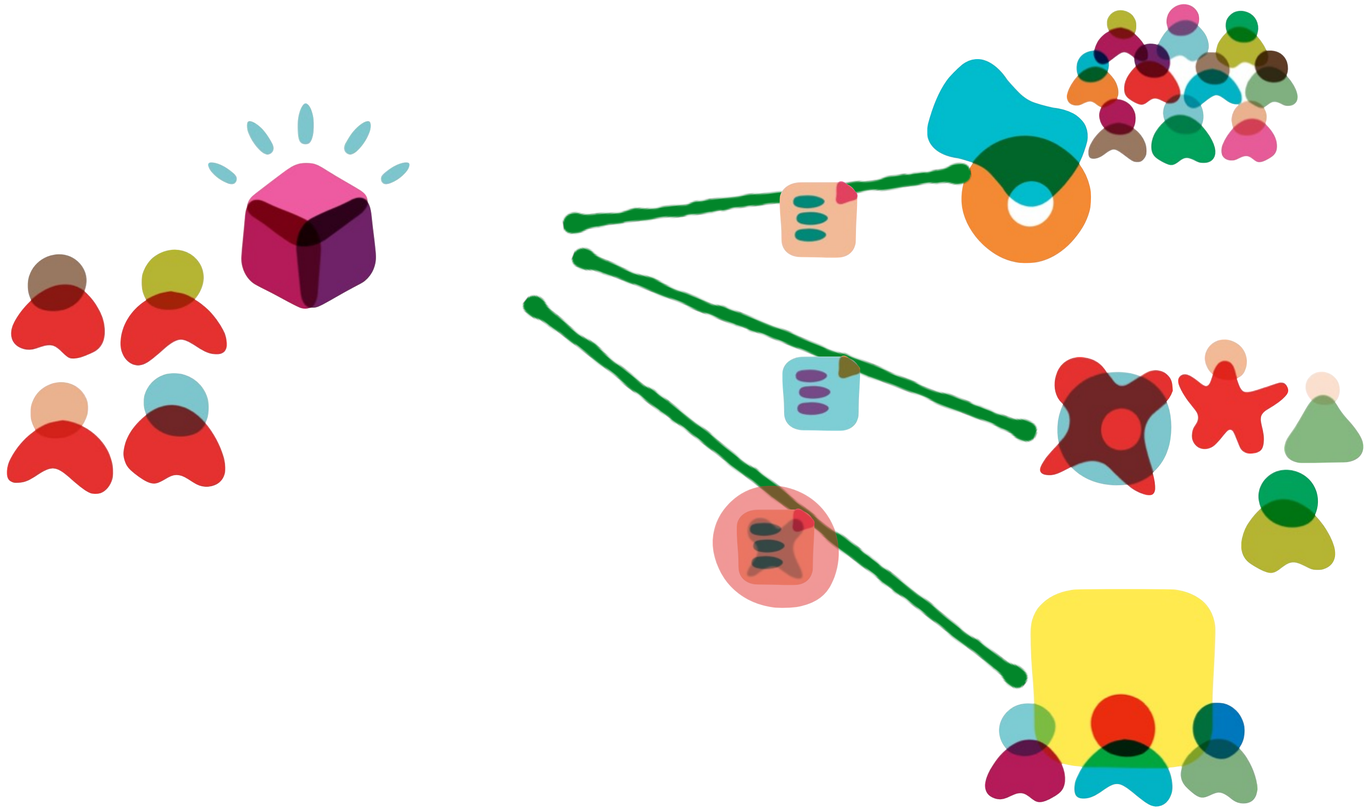
martinfowler.com/articles/consumerDrivenContracts.html

Consumer Driven Contracts



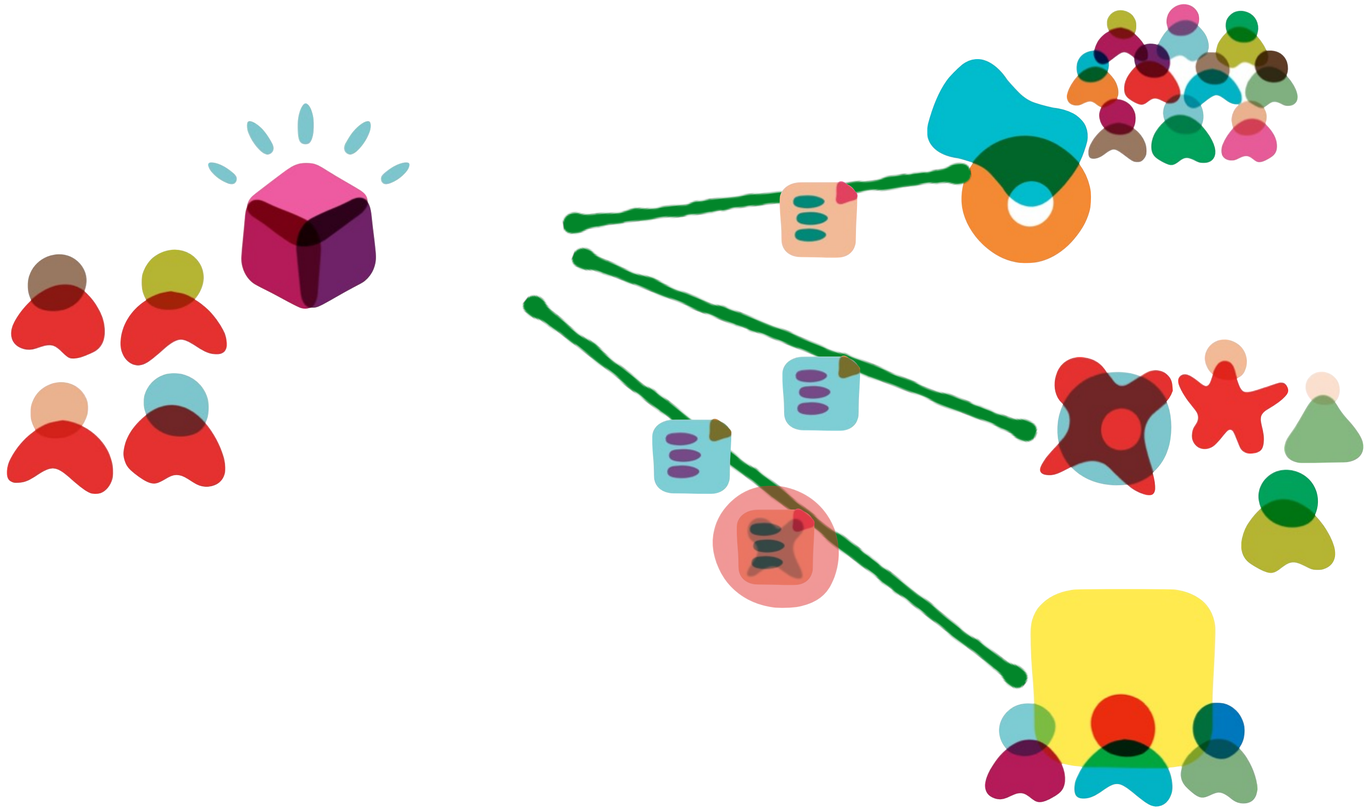
martinfowler.com/articles/consumerDrivenContracts.html

Consumer Driven Contracts



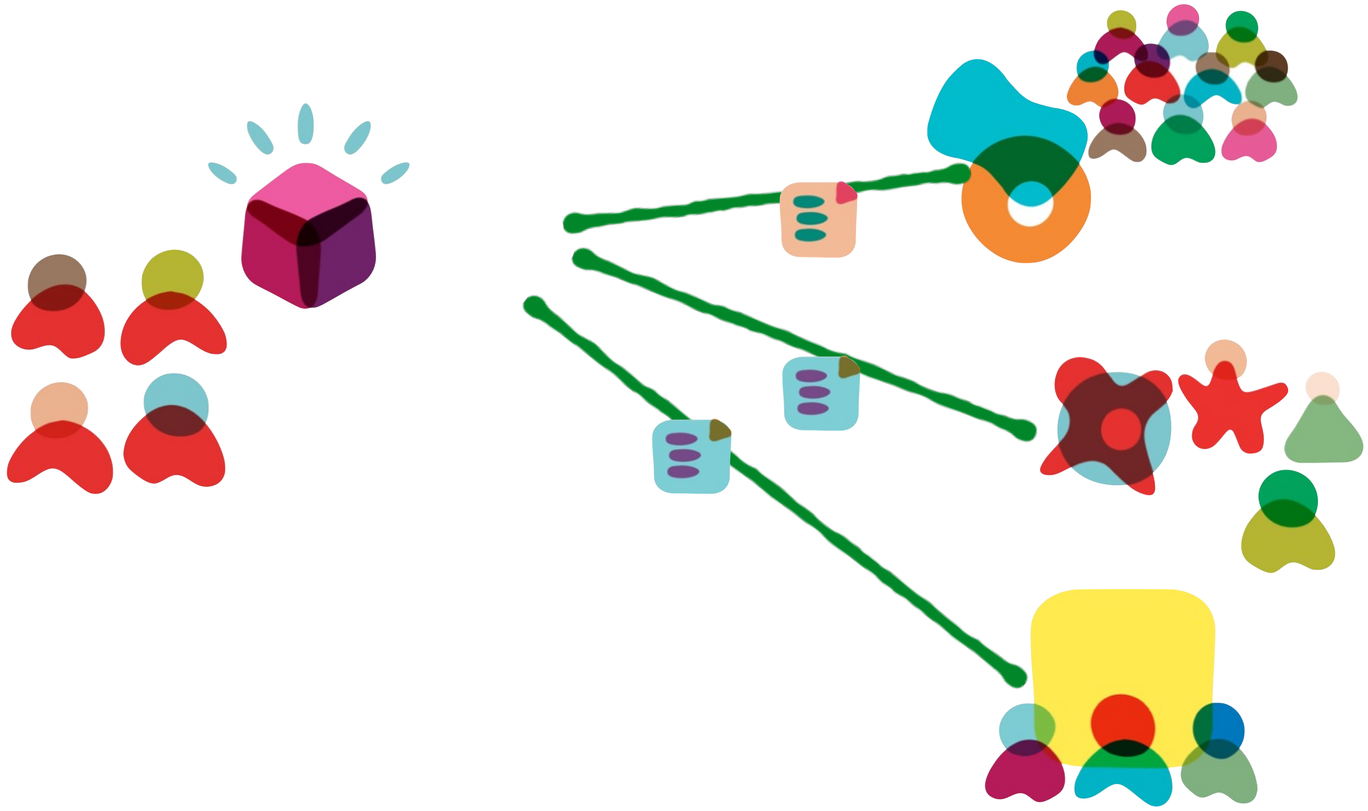
martinfowler.com/articles/consumerDrivenContracts.html

Consumer Driven Contracts



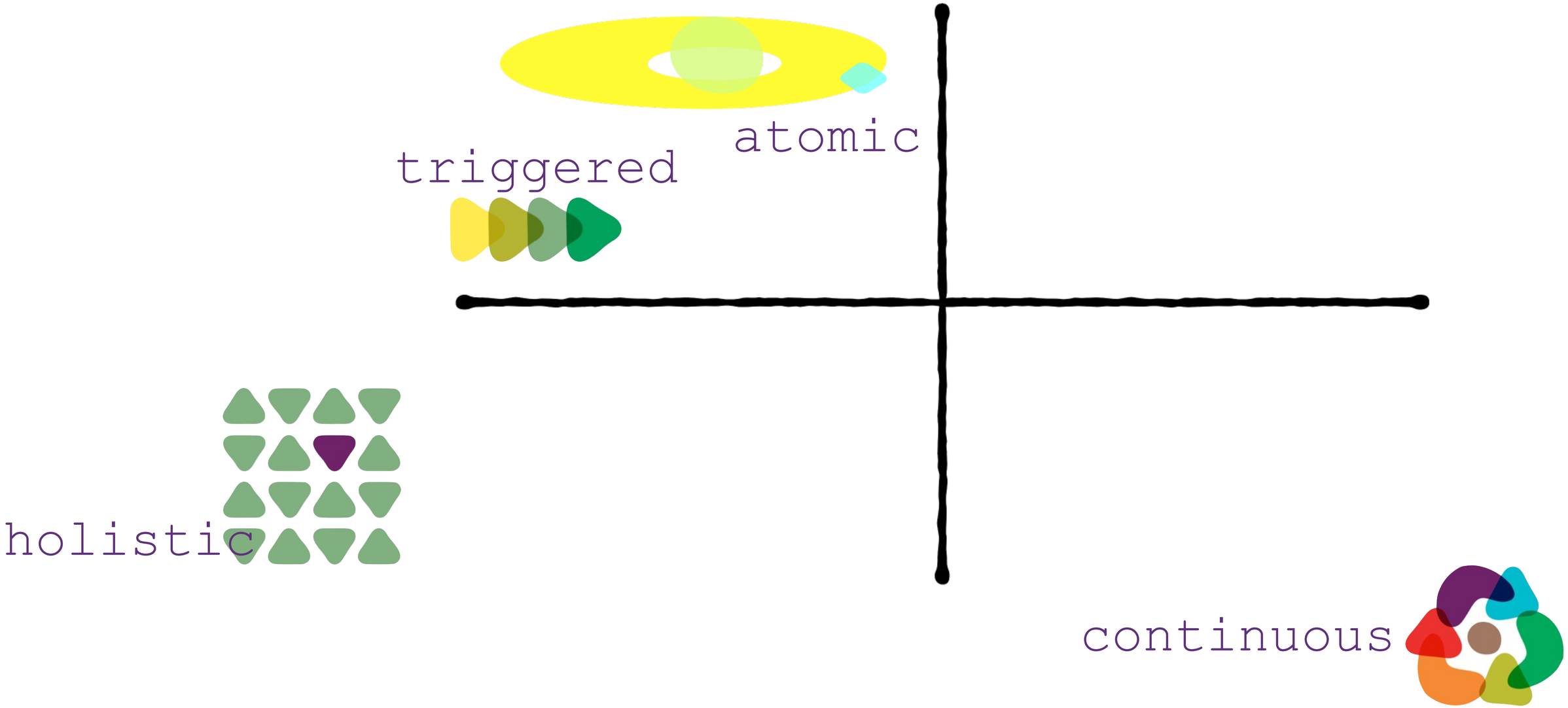
martinfowler.com/articles/consumerDrivenContracts.html

Consumer Driven Contracts



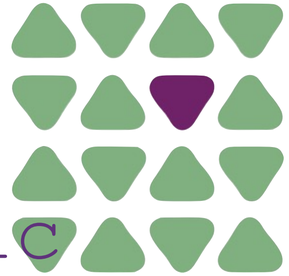
martinfowler.com/articles/consumerDrivenContracts.html

Fitness Function

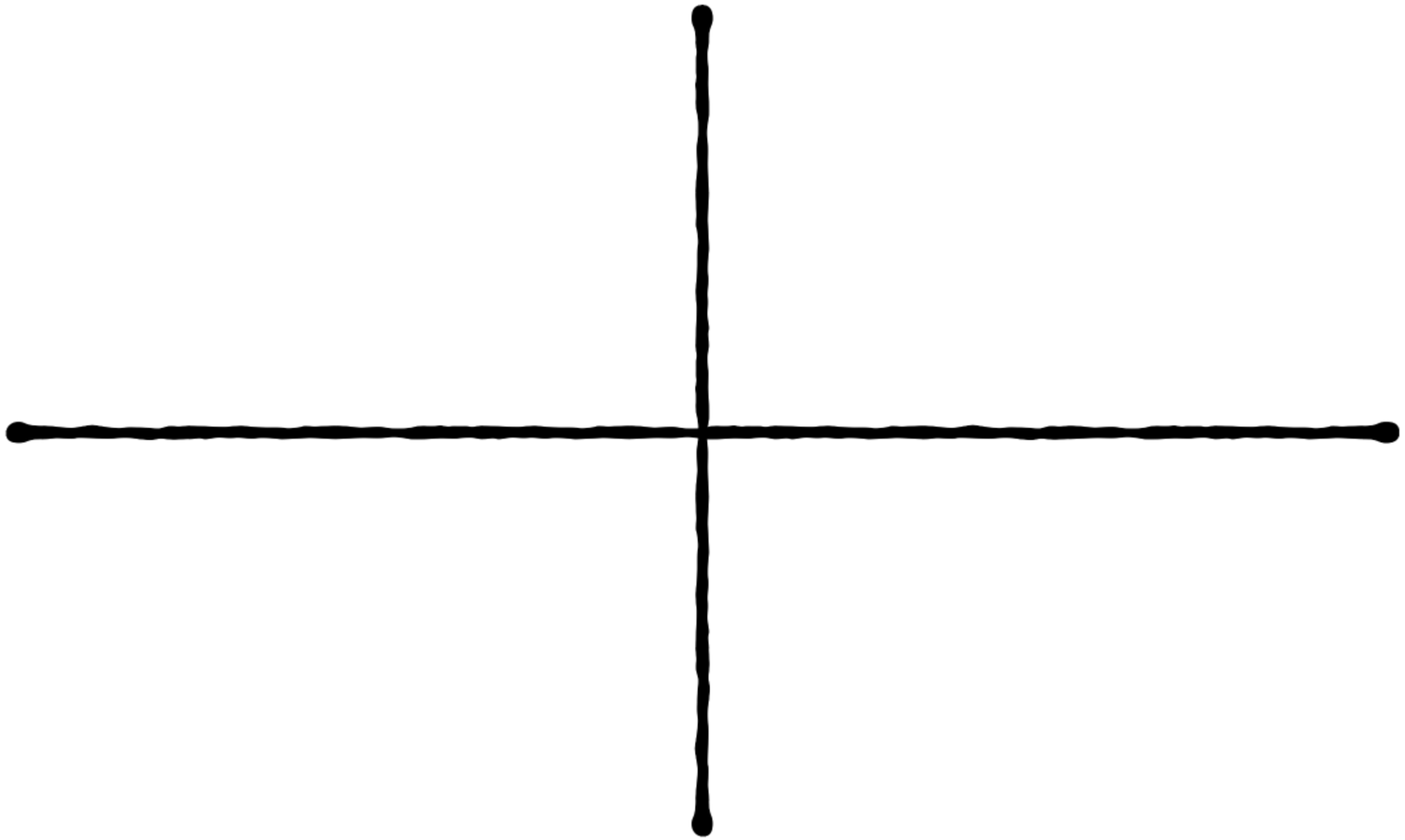


Fitness Function

atomic



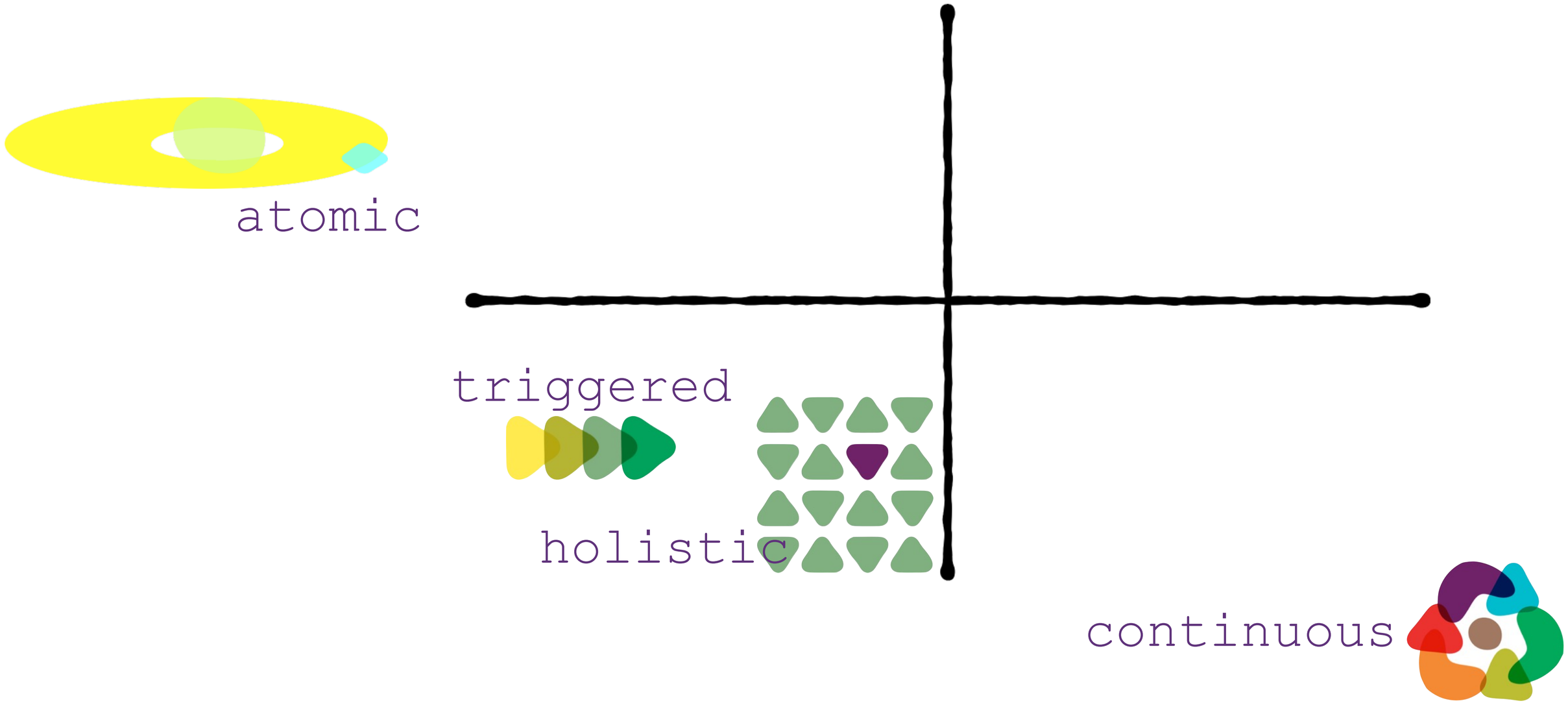
holistic



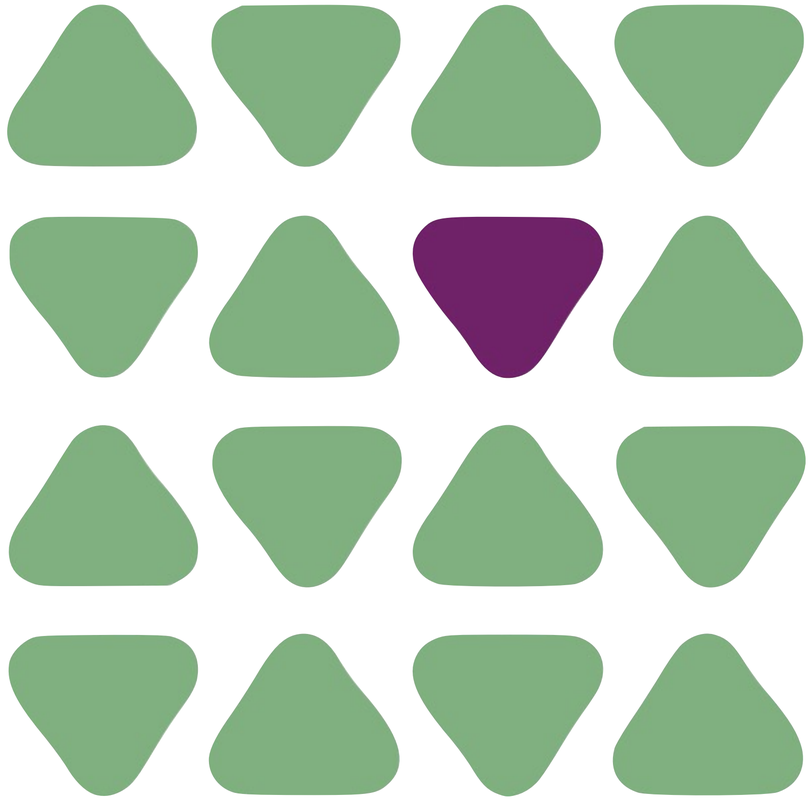
triggered

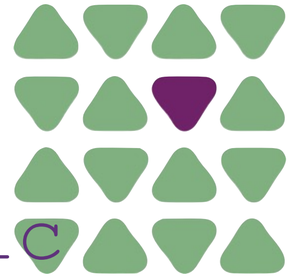
continuous

Fitness Function



triggered

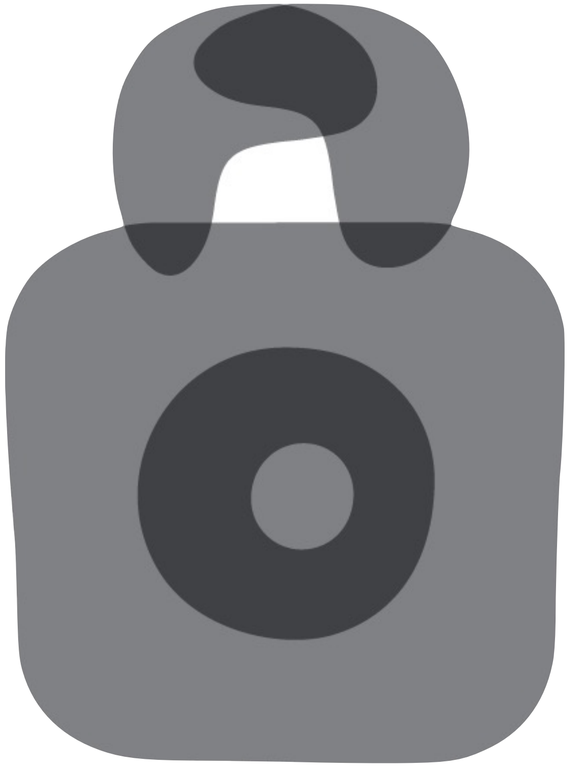




holistic

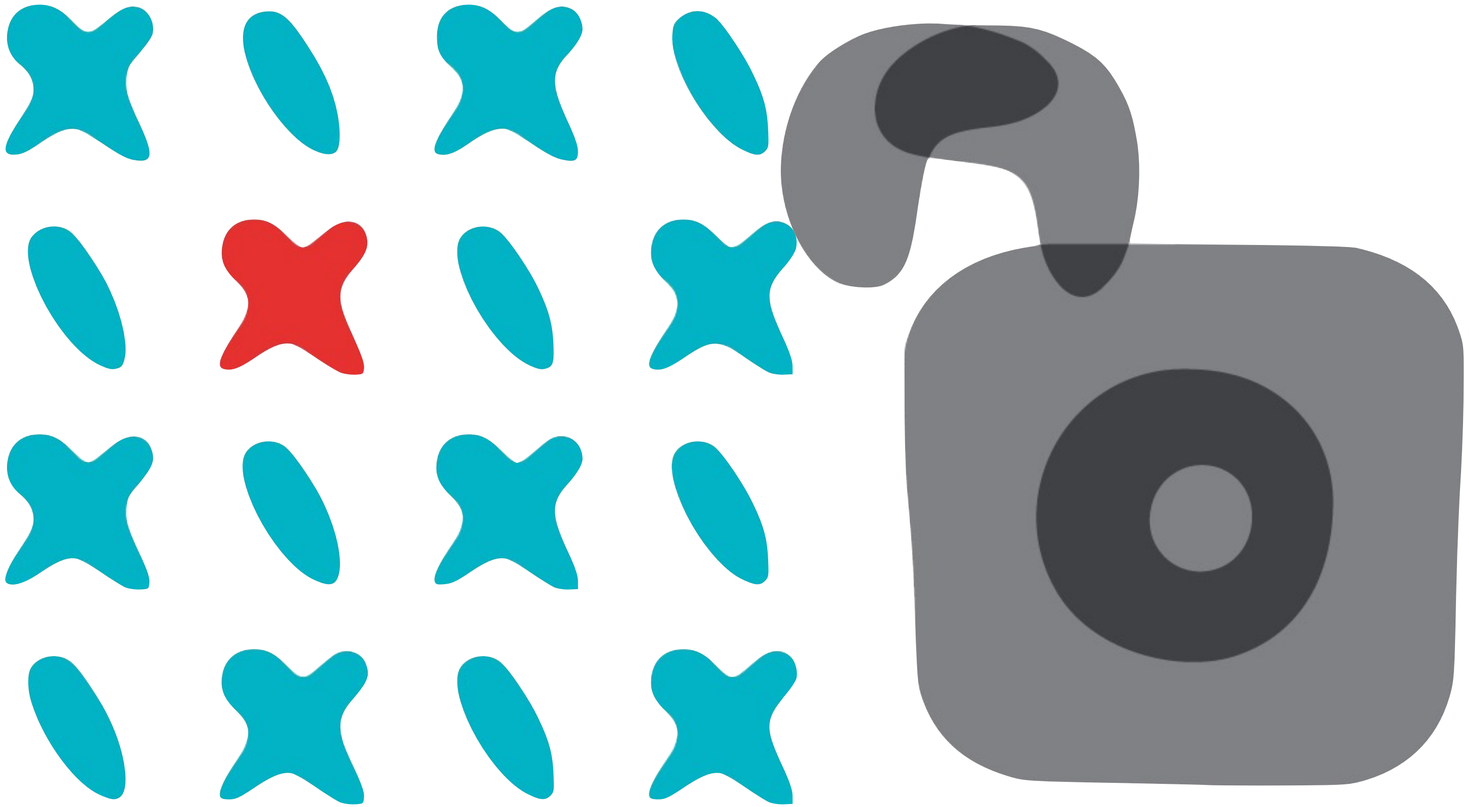
triggered

holistic

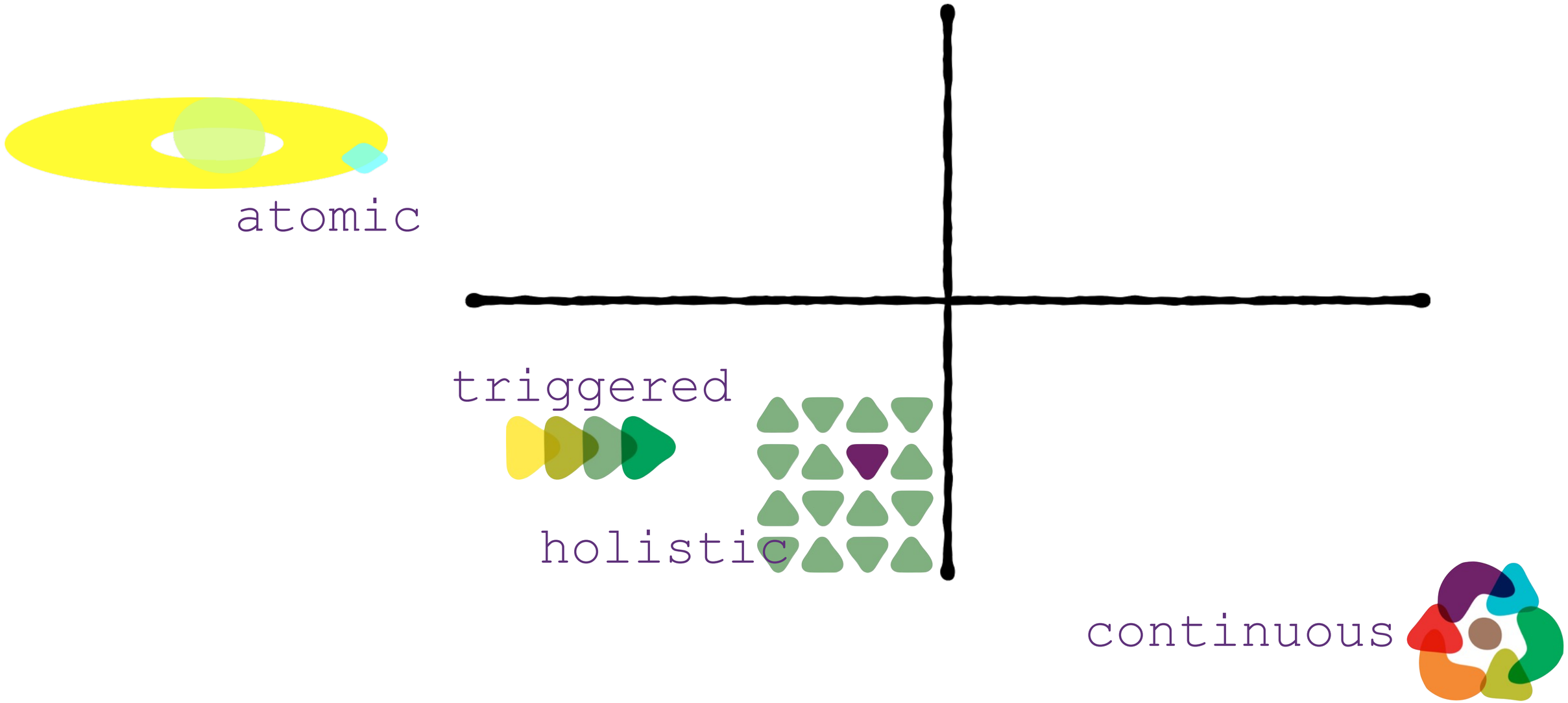


triggered

holistic

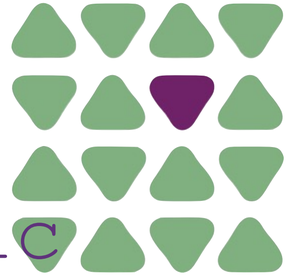


Fitness Function

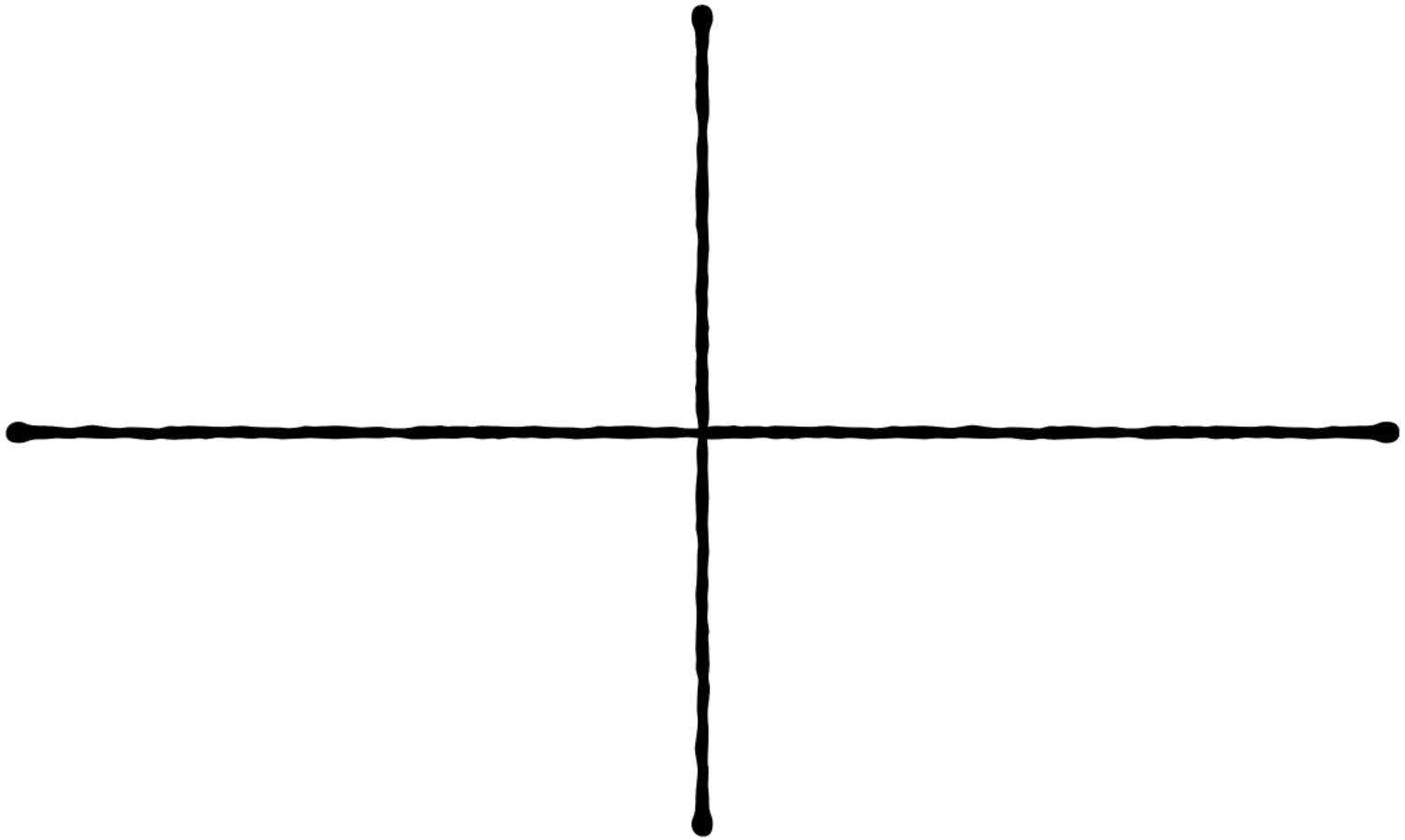


Fitness Function

atomic



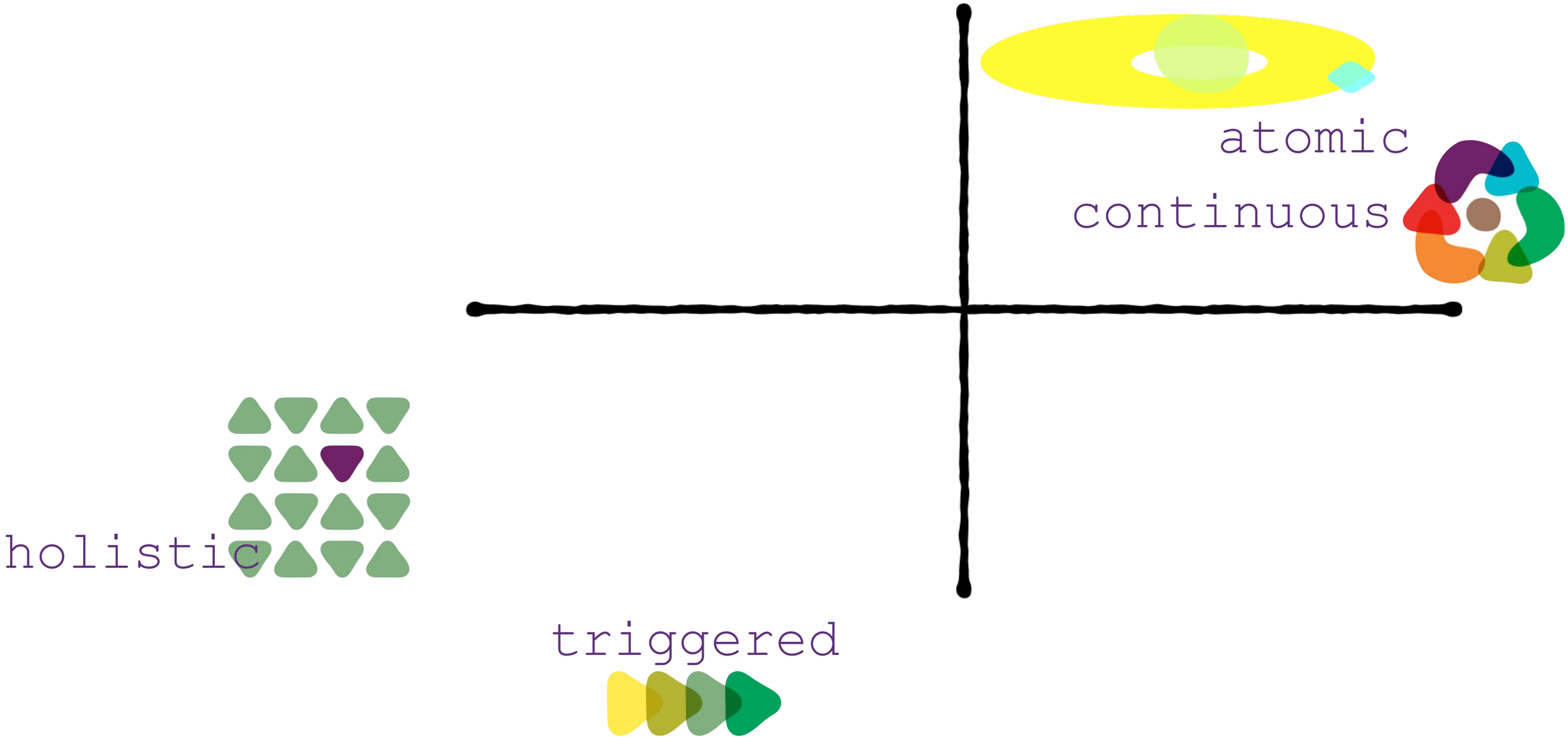
holistic



triggered

continuous

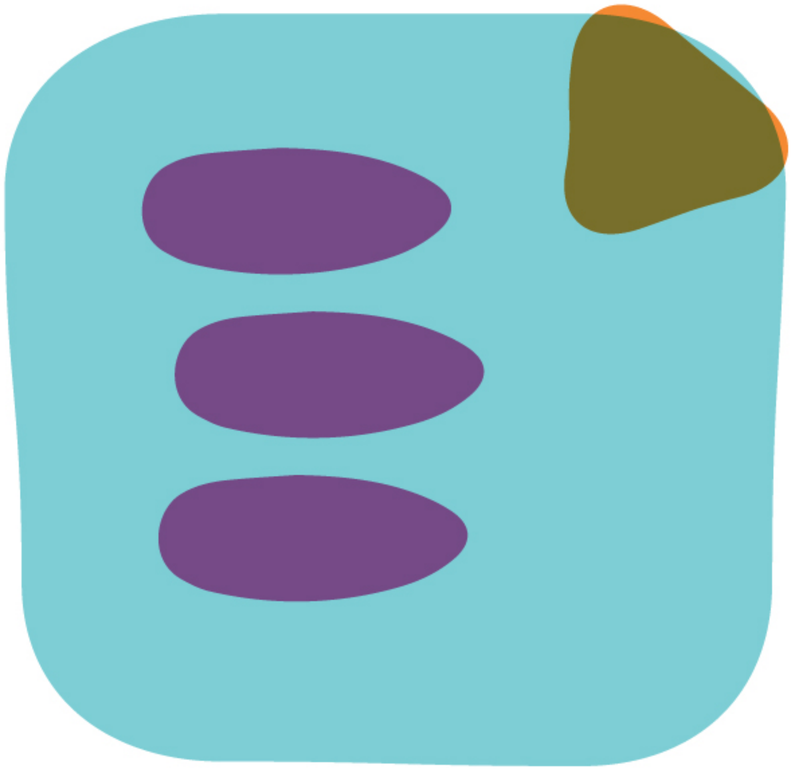
Fitness Function



atomic

continuous

monitoring

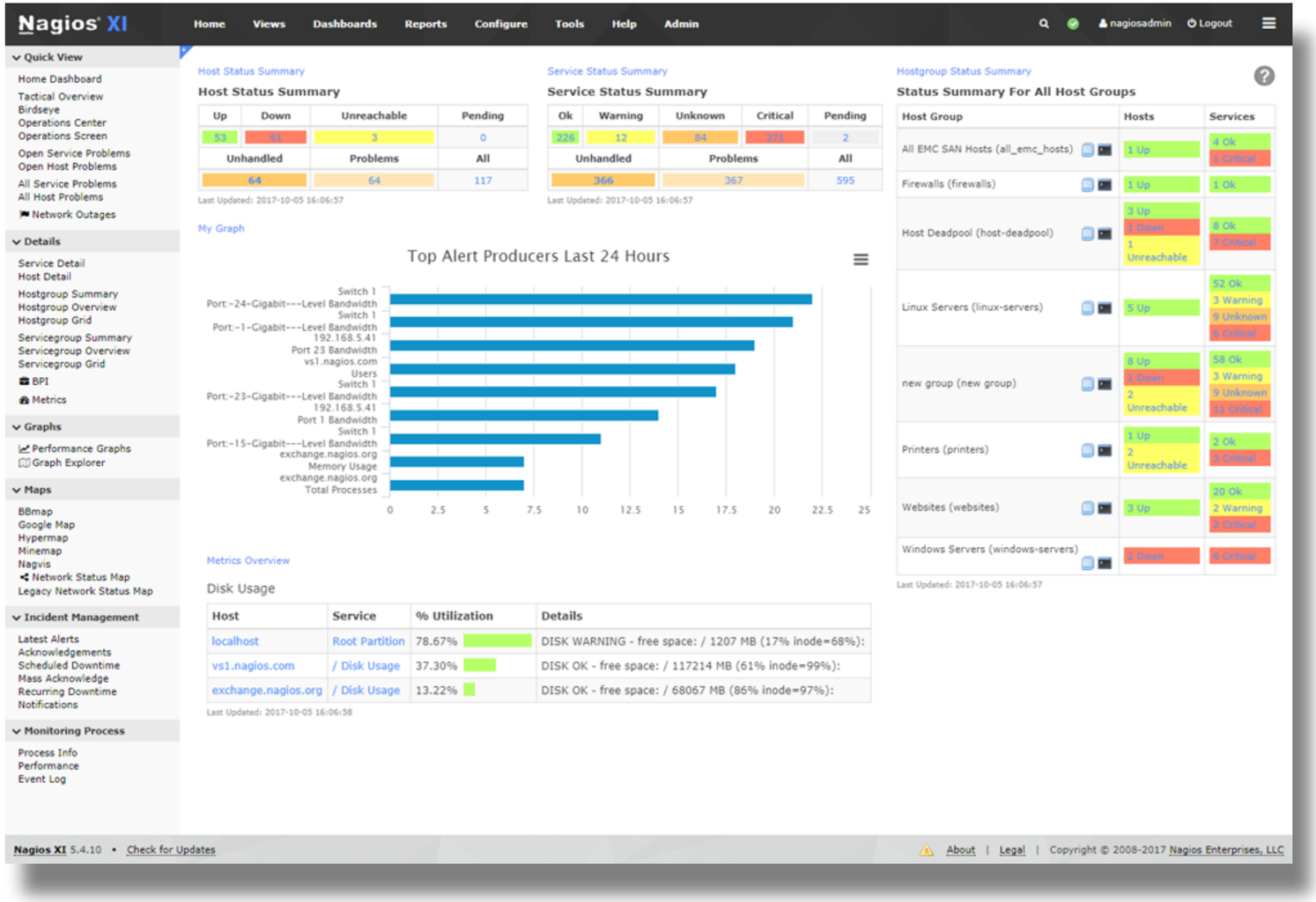


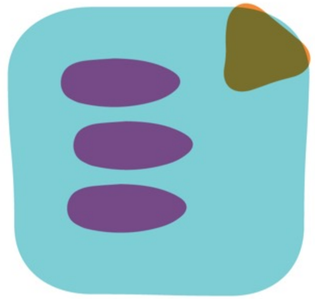
logging

atomic

continuous

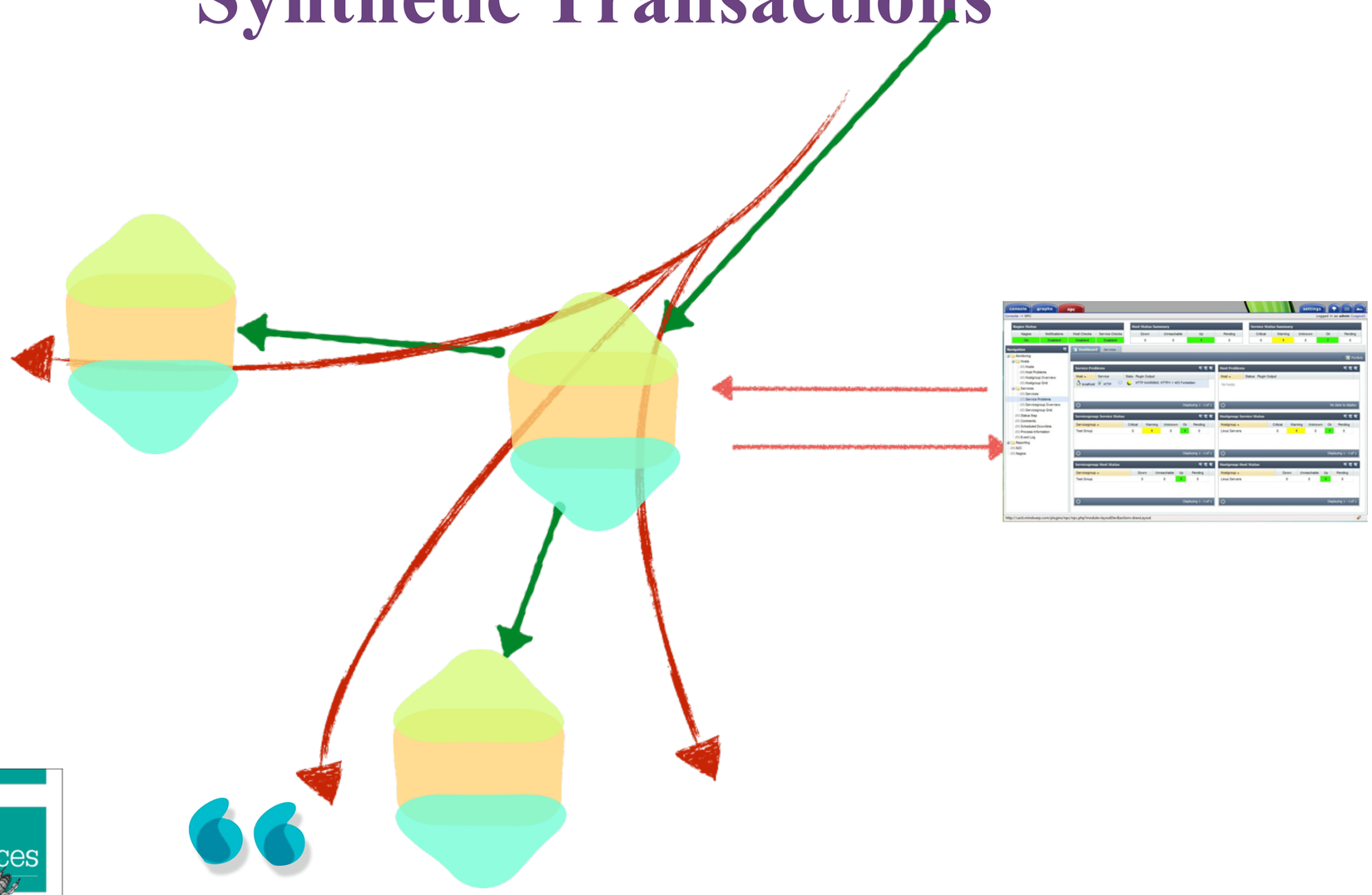
monitoring





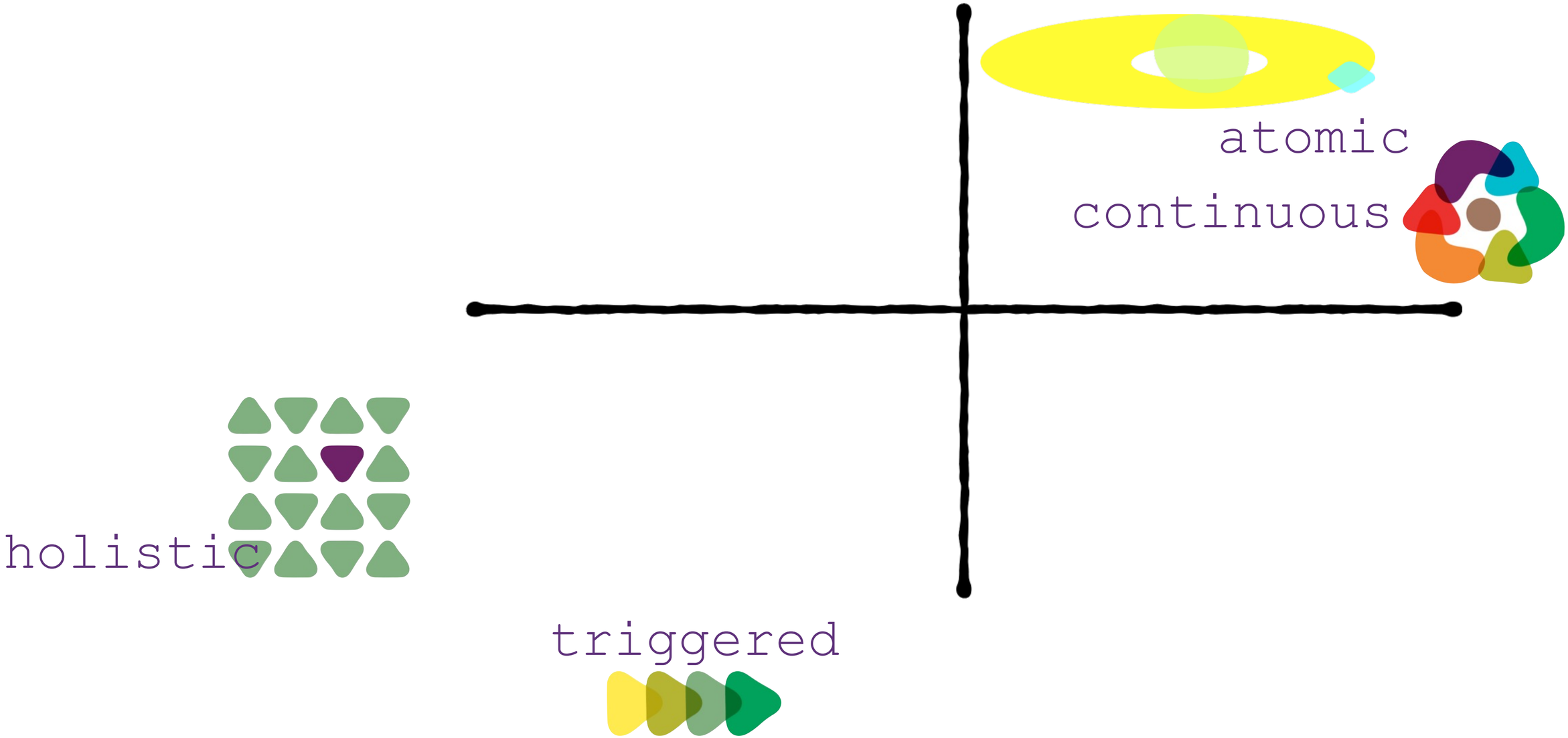
logging

Synthetic Transactions



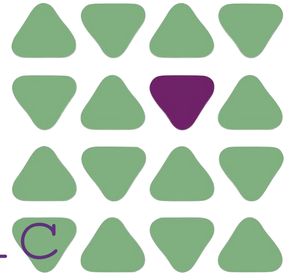
Use synthetic transactions to test production systems.

Fitness Function

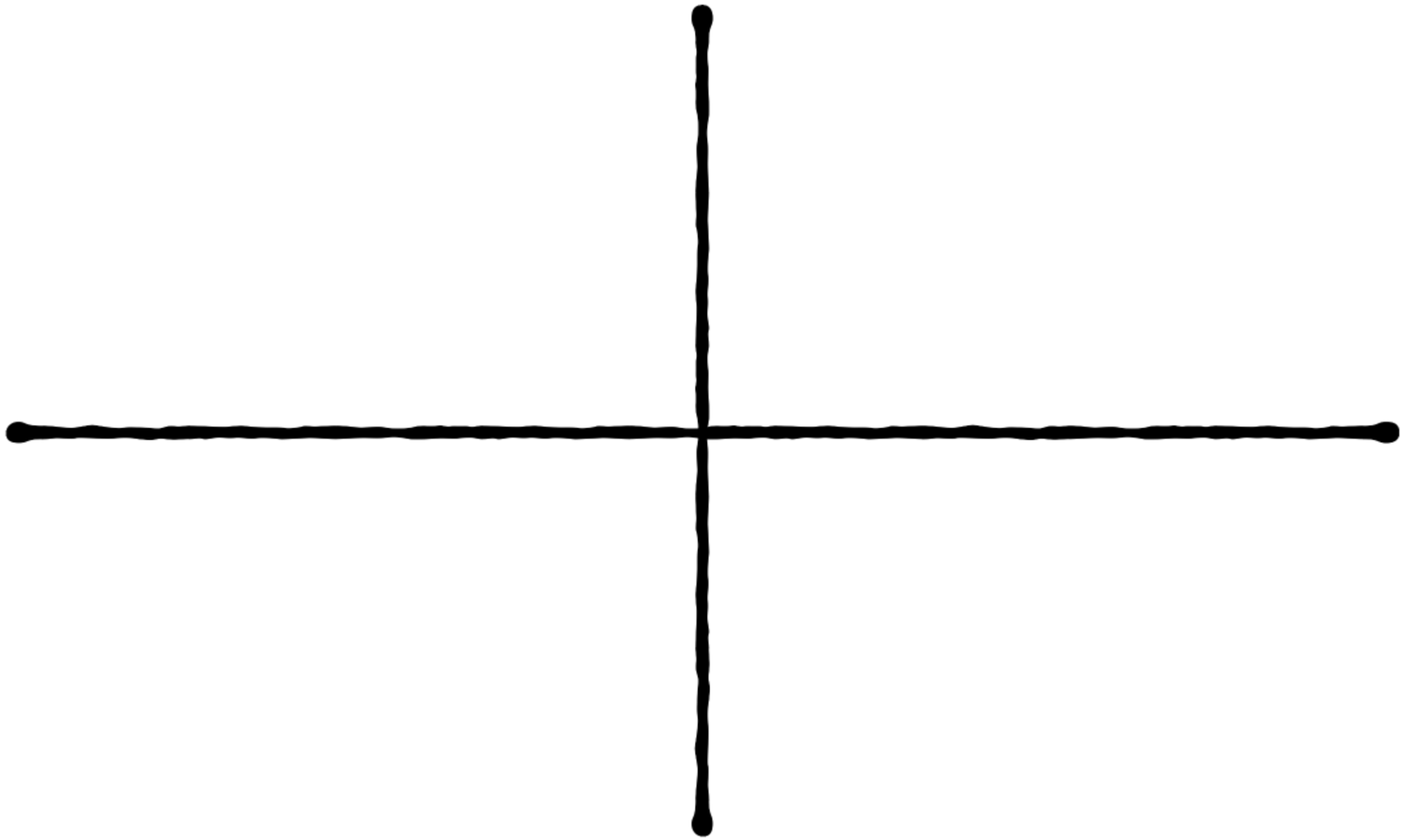


Fitness Function

atomic



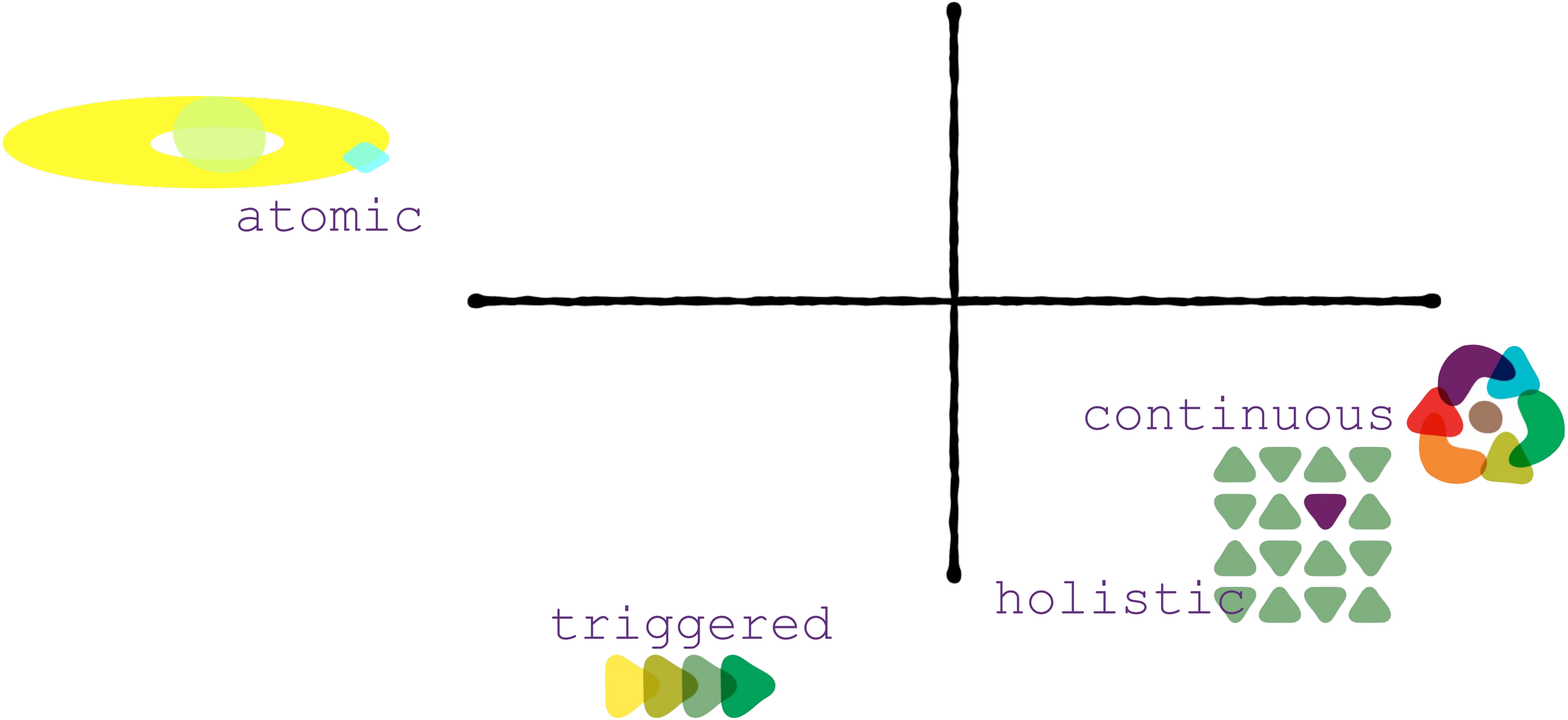
holistic



triggered

continuous

Fitness Function



amazon
web services
CHAOS
MONKEY

amazon
web services
CHAOS
MONKEY

SIMIAN
ARMY

chaos monkey

SIMIAN
ARMY
chaos gorilla

latency monkey

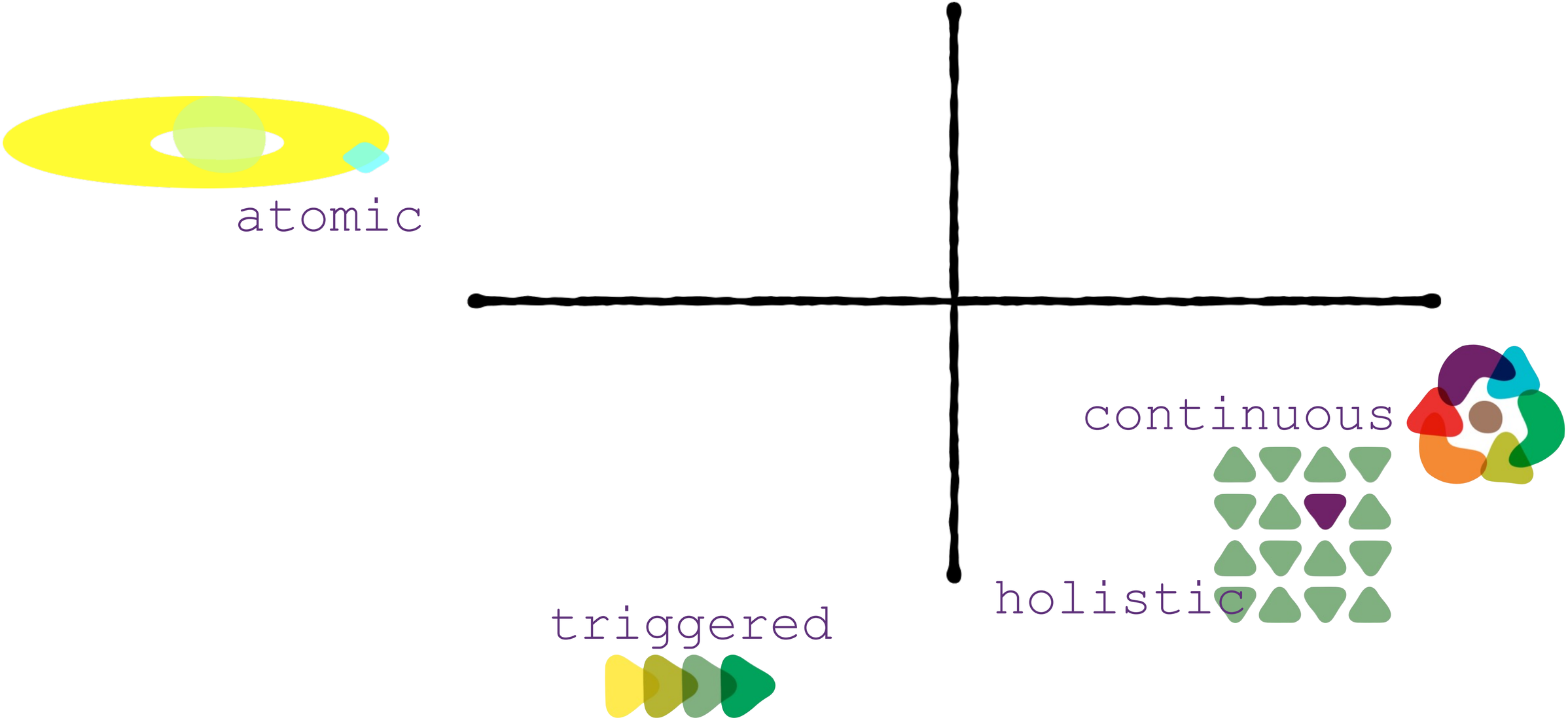
doctor monkey

janitor monkey

conformity monkey
janitor monkey
SIMIAN
ARMY

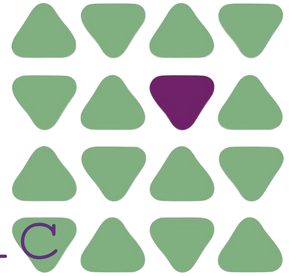
security monkey
conformity monkey
janitor monkey
SIMIAN
ARMY

Fitness Function

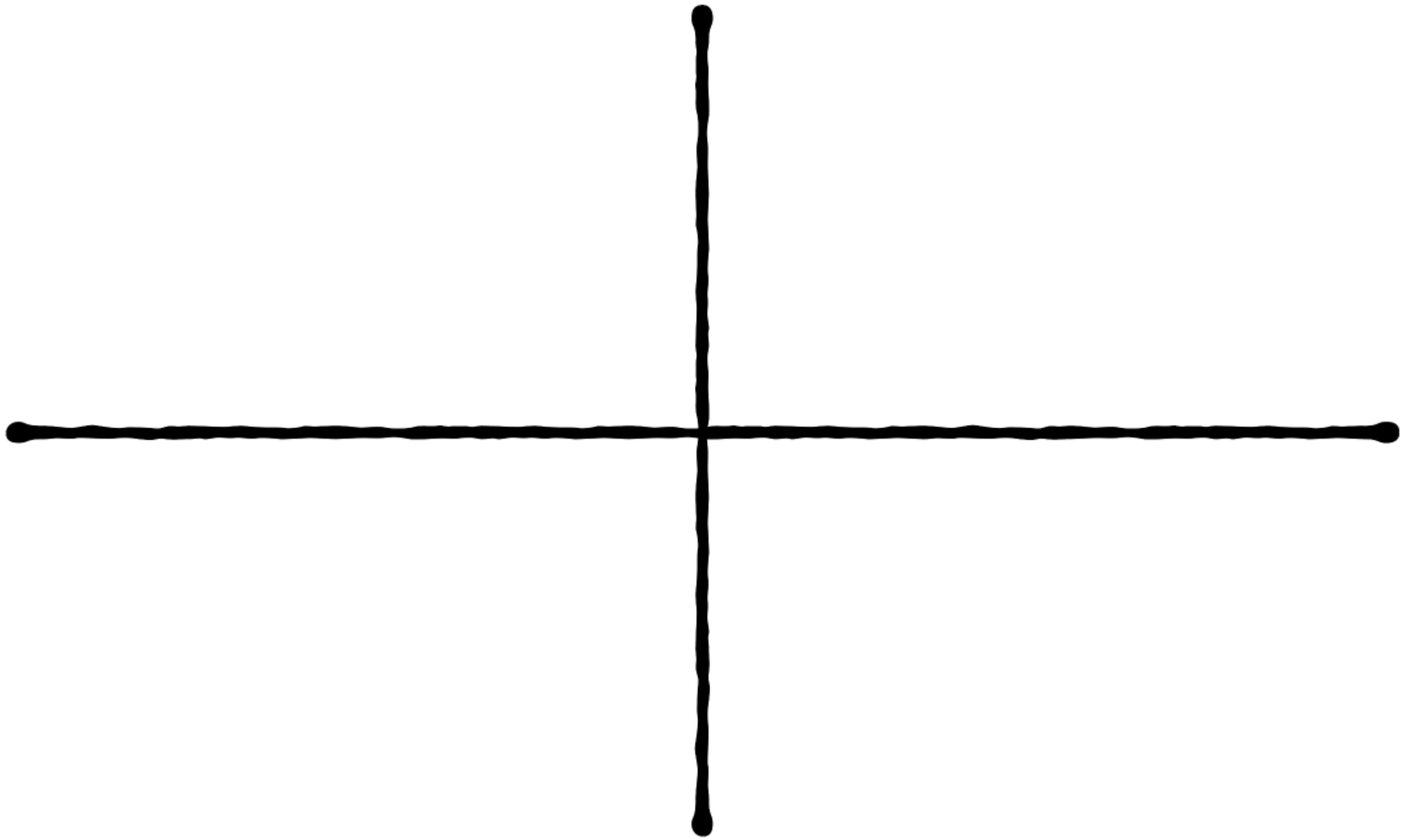


Fitness Function

atomic



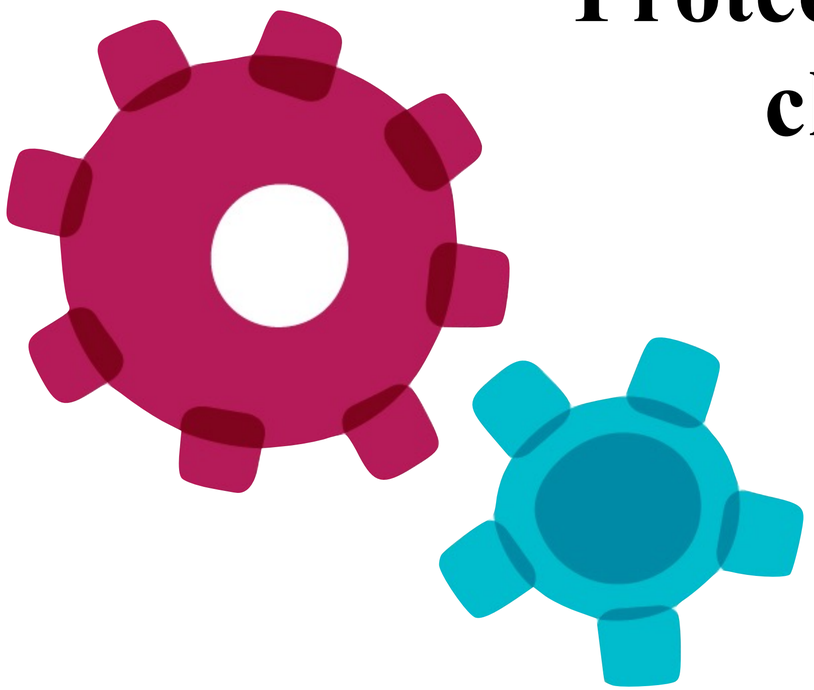
holistic



triggered

continuous

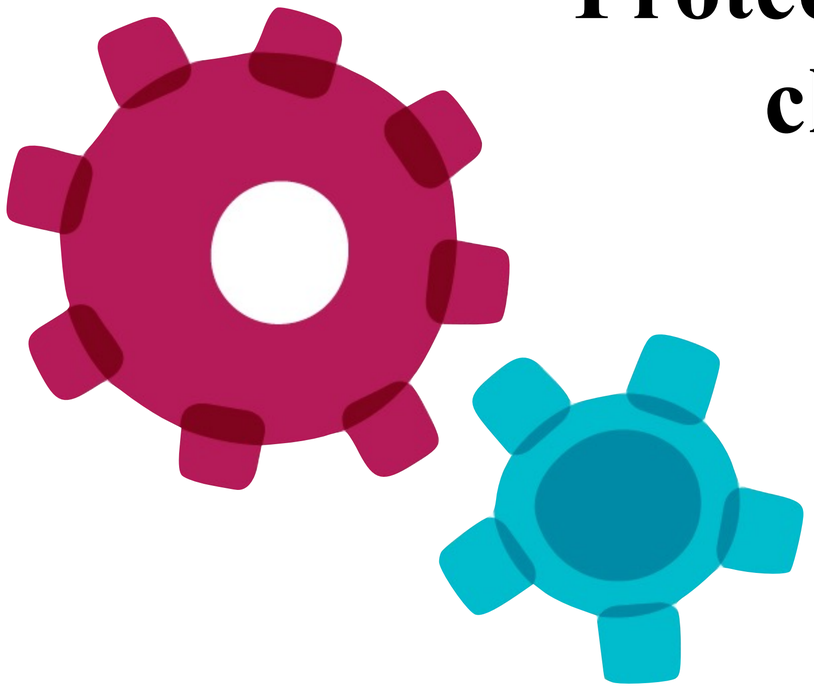
Implementing Fitness Functions



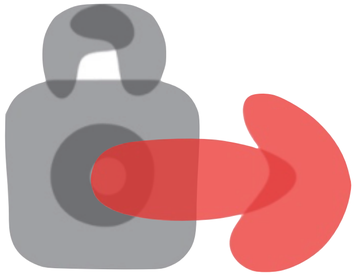
Protecting architectural characteristics

Implementing Fitness Functions

Protecting architectural characteristics



Automating governance



maintainable?

Cyclomatic complexity < 50 for all projects

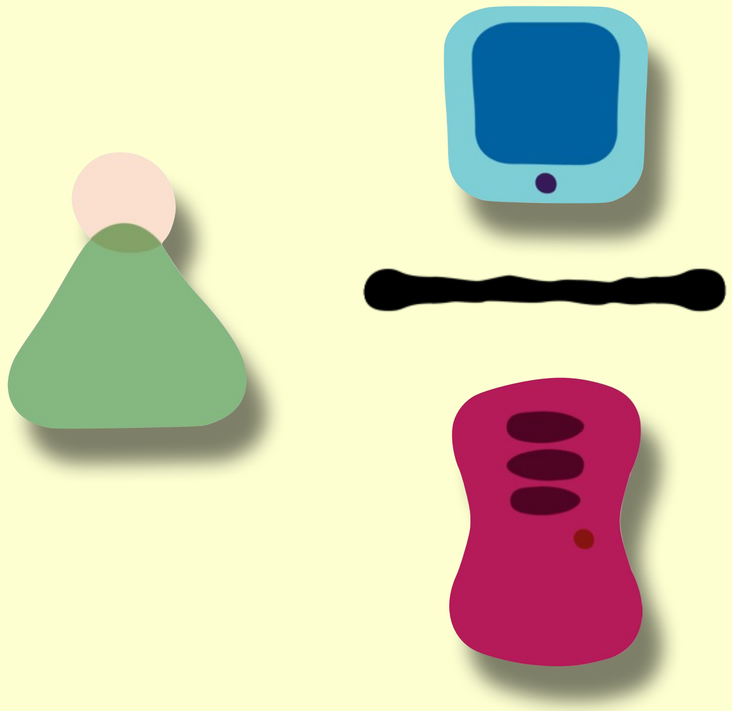
Naming conventions

immutability

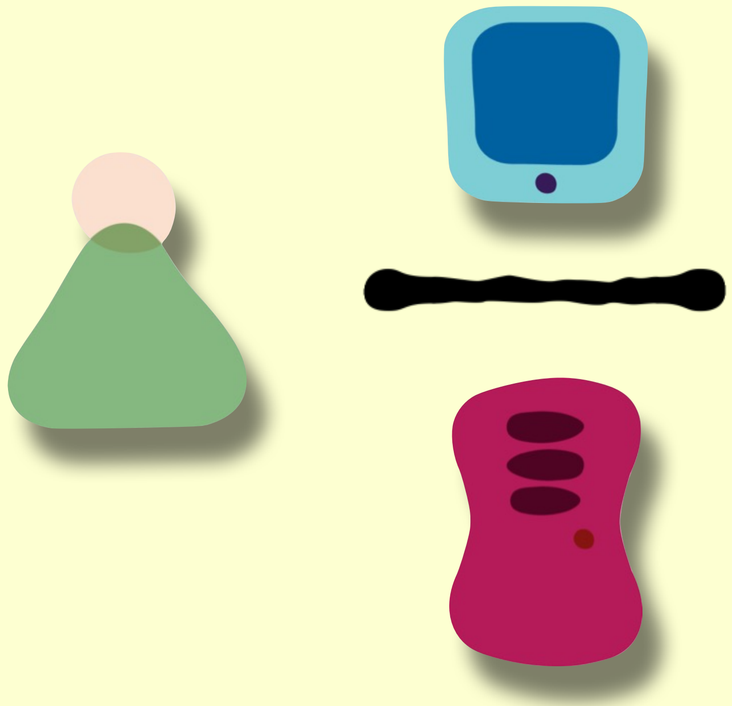
maintainable?

Controlled afferent/efferent coupling

Governing Code Quality

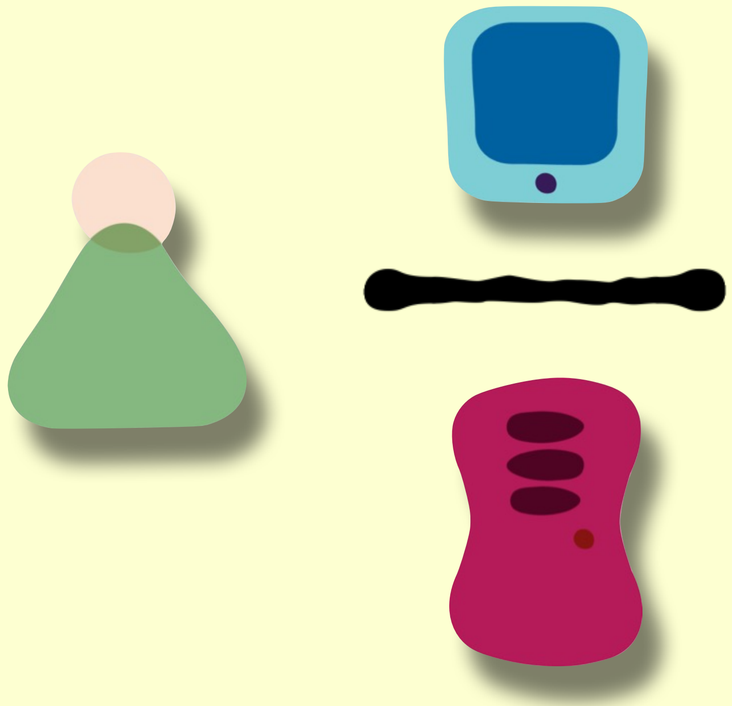


Governing Code Quality

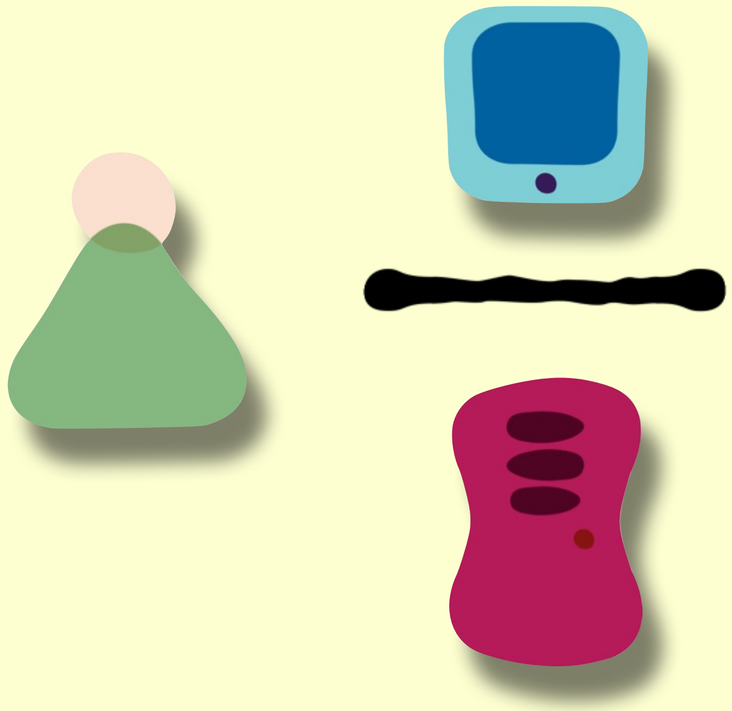


```
if state == “AL” then
    doSomethingForAL();
else if state == “GA” then
    doSomethingForGA();
else if …
```

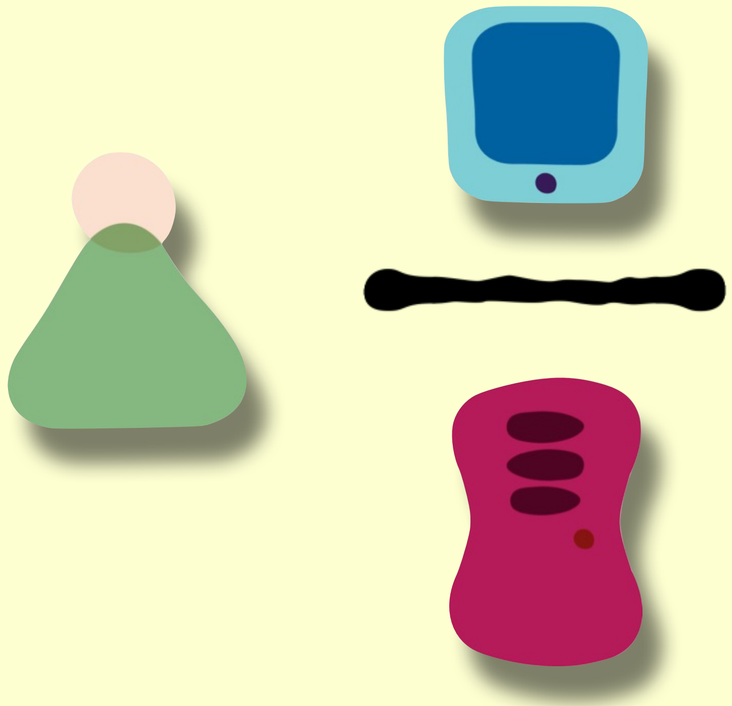
Governing Code Quality



Governing Code Quality

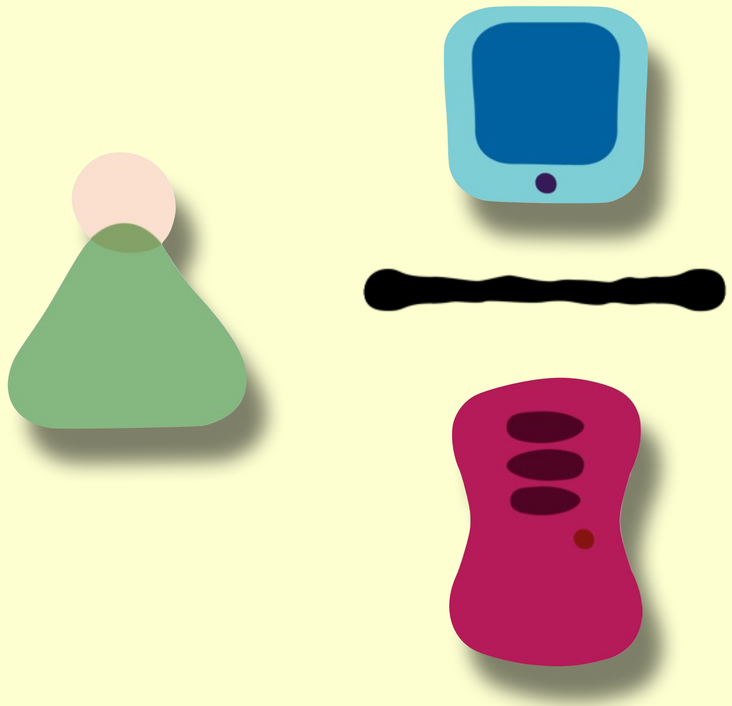


Governing Code Quality



```
if state == “AL” then
    doSomethingForAL();
else if state == “GA” then
    doSomethingForGA();
else if …
```

Governing Code Quality



```
if state ==          then
    doSome          AL();
else if s          ” then
    doS          );
else i
```

Governing Code Quality

Strategy Design Pattern

Safari File Edit View History Bookmarks Develop Window Help Sun 10:30

blog.jdriven.com

Apple iCloud Yahoo Bing Google Wikipedia Facebook Twitter LinkedIn The Weather Channel Yelp TripAdvisor

jdriven

← Previous Next →

Search

Implementing architectural fitness functions using Gradle, JUnit and code-assert

Posted on October 6, 2017 by Rob Brinkman Tweet

Architectural fitness functions

Inspired by Neal Ford's presentation at our Change is the Only constant event I started experimenting with architectural fitness functions. An architectural fitness function provides an objective integrity assessment of some architectural characteristic(s).

If you want to take a deeper dive into evolutionary architectures including fitness functions take look at Neals book: Building Evolutionary Architectures: Support Constant Change.

Neal's slides contained an example of verifying package dependencies from a Unit Test using JDepend.

Verifying code modularity

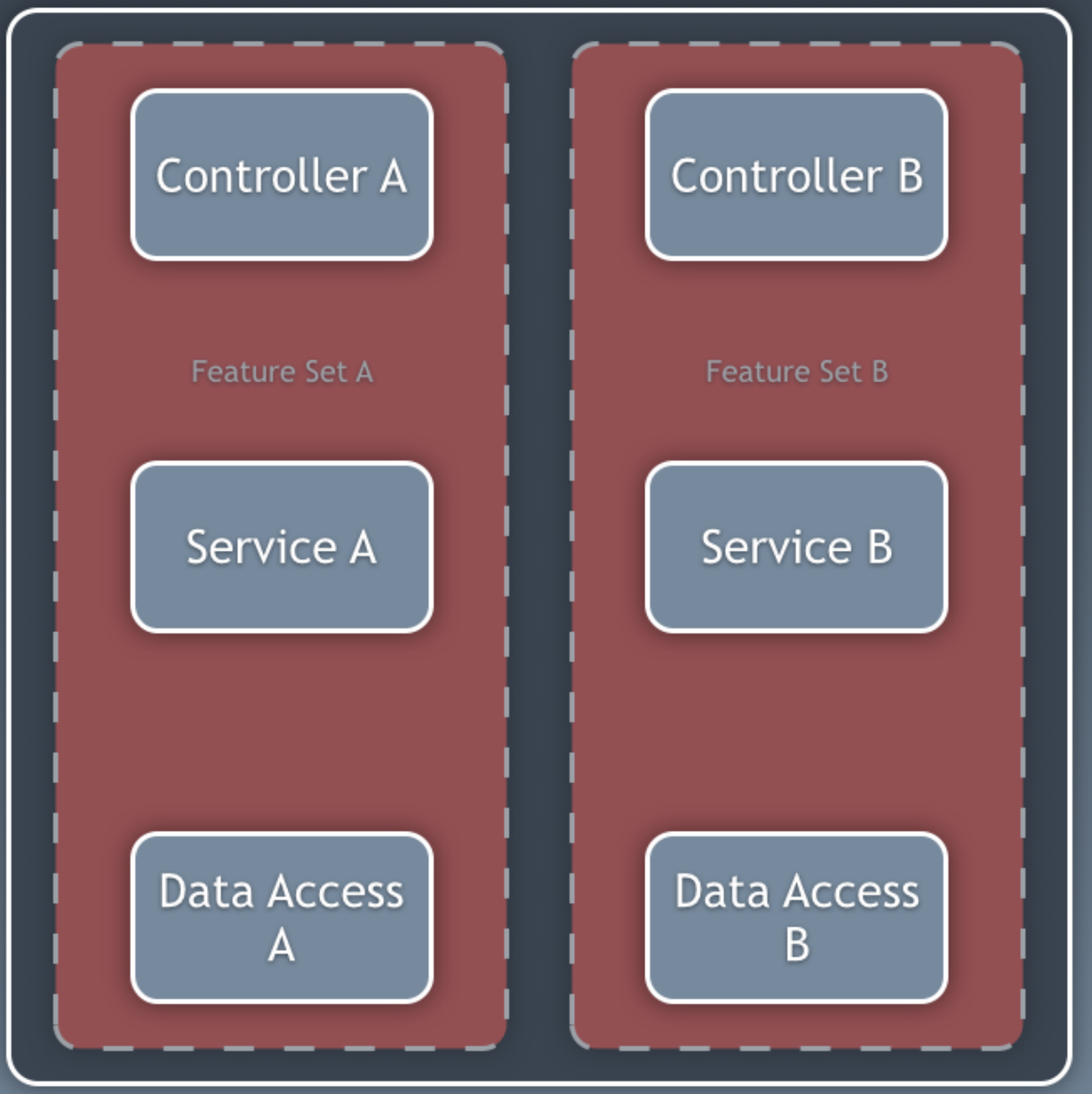
In this blog post we'll elaborate on that approach and create a Unit Test that verifies that our code complies to the chosen packaging strategies using an

JDriven
- blog.jdriven.com
- www.jdriven.com

Featured Posts
- Spring Sweets: Add (Extra) Build Information To Info Endpoint
- Angular2 and Spring Boot: Getting Started
- Het ontstaan van de passie voor het moderne maken
- Securing your application landscape with Spring Cloud Security – Part 1
- Spicy Spring : Dynamically create your own BeanDefinition

Recent Posts
- Awesome Asciidoctor: Grouping Floating Images
- Awesome Asciidoctor: Using Tab Separated Data In A Table
- Implementing architectural fitness functions using Gradle, JUnit and code-assert
- 6 Steps to help you debug your application

https://blog.jdriven.com/2017/10/implementing-architectural-fitness-functions-using-gradle-junit-code-assert/



Package by feature (vertical slicing)

```java
public class VerifyPackageByFeatureTest {

    @Test
    public void verifyPackageByFeature() {

        /// Create an analyzer config for the package we'd like to verify
        AnalyzerConfig analyzerConfig = GradleAnalyzerConfig.gradle().main("com.jdriven.fitness.packaging.by.feature");

        // Dependency Rules for Packaging By Feature
        // NOTE: the classname should match the packagename
        class ComJdrivenFitnessPackagingByFeature extends DependencyRuler {

            // Rules for feature child packages
            // NOTE: they should match the name of the sub packages
            DependencyRule a, b;

            @Override
            public void defineRules() {
                // Our App classes depends on all subpackages because it constructs all of them
                base().mayUse(base().allSub());
            }
        }

        // All dependencies are forbidden, except the ones defined in ComJdrivenFitnessPackagingByFeature
        // java, org, net packages may be used freely
        DependencyRules rules = DependencyRules.denyAll()
                .withRelativeRules(new ComJdrivenFitnessPackagingByFeature())
                .withExternals("java.*", "org.*", "net.*");

        DependencyResult result = new DependencyAnalyzer(analyzerConfig).rules(rules).analyze();
        assertThat(result, matchesRulesExactly());
    }
}
```

```java
public class ControllerA {

    private final ServiceA serviceA;
    private final ServiceB serviceB;

    public ControllerA(ServiceA serviceA, ServiceB serviceB) {
        this.serviceA = serviceA;
        this.serviceB = serviceB;
    }
}
```

```
java.lang.AssertionError:
 Expected: Comply with rules
 but: DENIED com.jdriven.fitness.packaging.by.feature.a -&gt;
 com.jdriven.fitness.packaging.by.feature.b (by com.jdriven.fitness.packaging.by.feature.a.ControllerA)
```

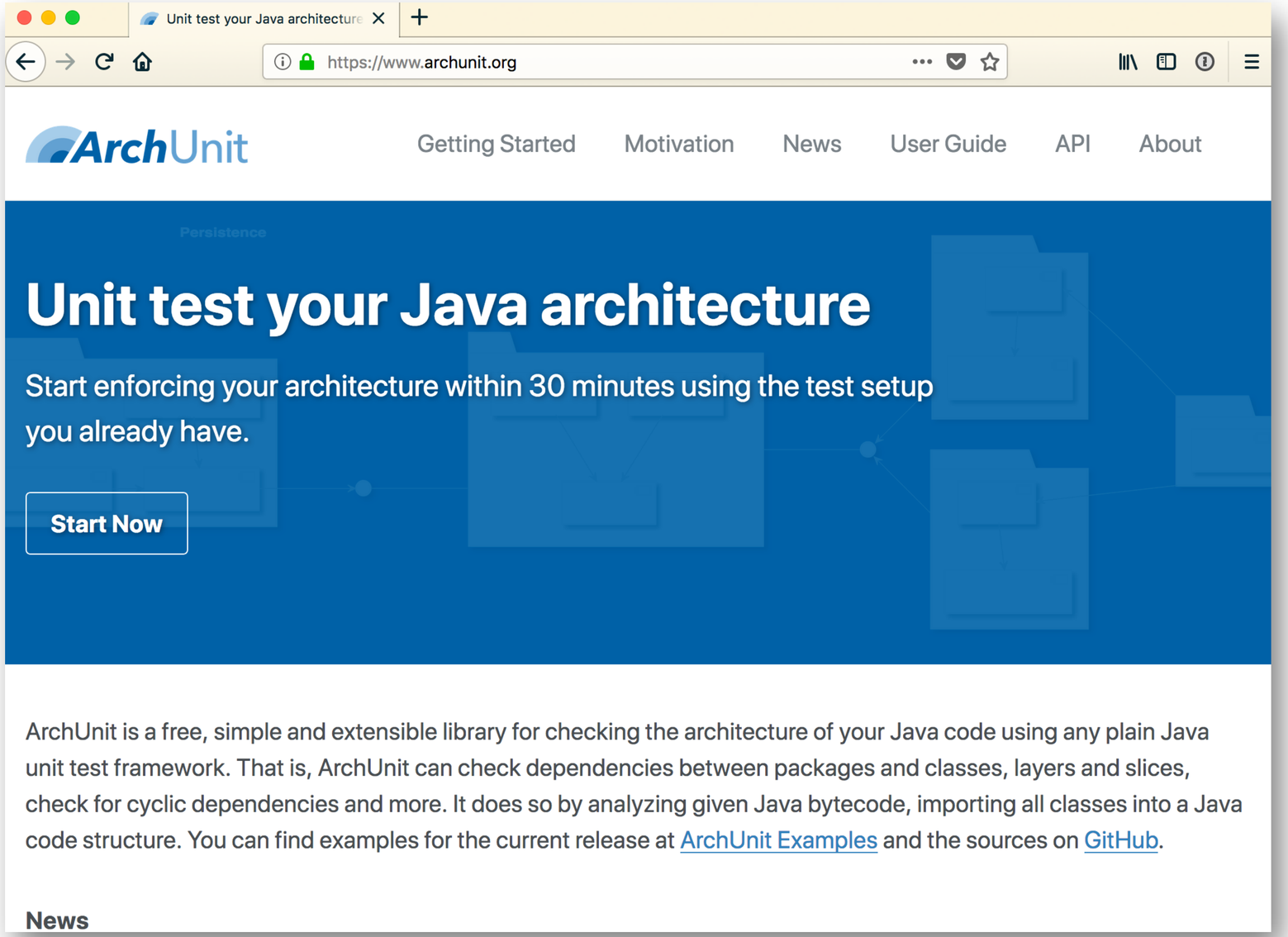
```java
public class ControllerA {

    private final ServiceA serviceA;
    private final ServiceB serviceB;

    public ControllerA(ServiceA serviceA, ServiceB serviceB) {
        this.serviceA = serviceA;
        this.serviceB = serviceB;
    }
}
```

```
java.lang.AssertionError:
 Expected: Comply with rules
 but: DENIED com.jdriven.fitness.packaging.by.feature.a -&gt;
 com.jdriven.fitness.packaging.by.feature.b (by com.jdriven.fitness.packaging.by.feature.a.ControllerA)
```

```java
// Allow package / module a to acces b
a.mayUse(b);
```

ArchUnit

https://www.archunit.org/



ArchUnit

https://www.archunit.org/

```java
import static com.tngtech.archunit.lang.syntax.ArchRuleDefinition.noClasses;
import static com.tngtech.archunit.library.GeneralCodingRules.ACCESS_STANDARD_STREAMS;
import static com.tngtech.archunit.library.GeneralCodingRules.NO_CLASSES_SHOULD_ACCESS_STANDARD_STREAMS;
import static com.tngtech.archunit.library.GeneralCodingRules.NO_CLASSES_SHOULD_THROW_GENERIC_EXCEPTIONS;
import static com.tngtech.archunit.library.GeneralCodingRules.NO_CLASSES_SHOULD_USE_JAVA_UTIL_LOGGING;

public class CodingRulesTest {
    private JavaClasses classes;

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

    @Test
    public void classes_should_not_access_standard_streams_defined_by_hand() {
        noClasses().should(ACCESS_STANDARD_STREAMS).check(classes);
    }

    @Test
    public void classes_should_not_access_standard_streams_from_library() {
        NO_CLASSES_SHOULD_ACCESS_STANDARD_STREAMS.check(classes);
    }

    @Test
    public void classes_should_not_throw_generic_exceptions() {
        NO_CLASSES_SHOULD_THROW_GENERIC_EXCEPTIONS.check(classes);
    }

    @Test
    public void classes_should_not_use_java_util_logging() {
        NO_CLASSES_SHOULD_USE_JAVA_UTIL_LOGGING.check(classes);
    }
}
```

coding rules

ArchUnit

https://www.archunit.org/

```
@Test
public void classes_should_not_access_standard_streams_from_library() {
    NO_CLASSES_SHOULD_ACCESS_STANDARD_STREAMS.check(classes);
}

@Test
public void classes_should_not_throw_generic_exceptions() {
    NO_CLASSES_SHOULD_THROW_GENERIC_EXCEPTIONS.check(classes);
}

@Test
public void classes_should_not_use_java_util_logging() {
    NO_CLASSES_SHOULD_USE_JAVA_UTIL_LOGGING.check(classes);
}
```

coding rules

ArchUnit

https://www.archunit.org/

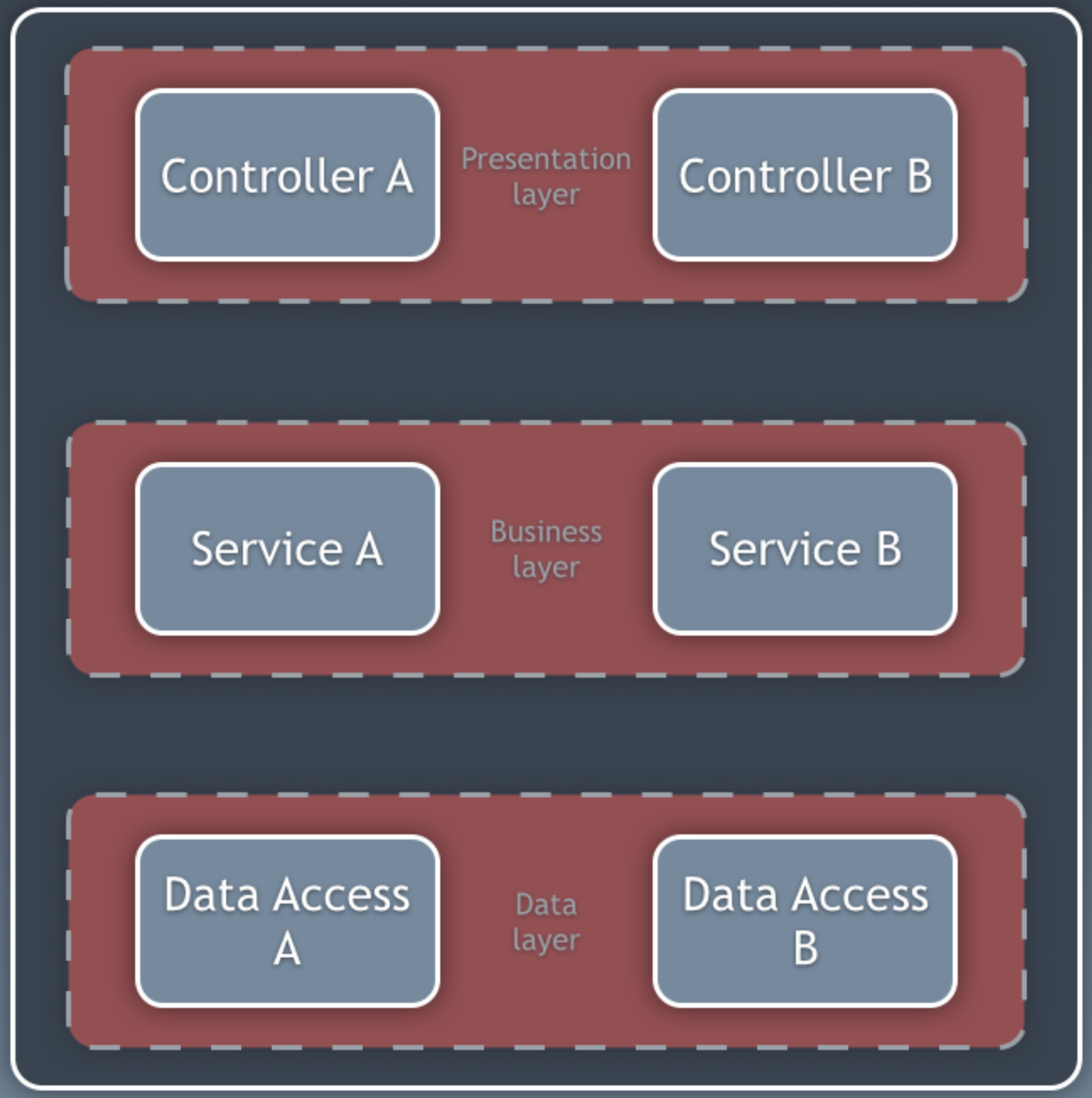
```java
public class InterfaceRules {

    @Test
    public void interfaces_should_not_have_names_ending_with_the_word_interface() {
        JavaClasses classes = new ClassFileImporter().importClasses(
                SomeBusinessInterface.class,
                SomeDao.class
        );

        noClasses().that().areInterfaces().should().haveNameMatching(".*Interface").check(classes);
    }

    @Test
    public void interfaces_should_not_have_simple_class_names_ending_with_the_word_interface() {
        JavaClasses classes = new ClassFileImporter().importClasses(
                SomeBusinessInterface.class,
                SomeDao.class
        );

        noClasses().that().areInterfaces().should().haveSimpleNameContaining("Interface").check(classes);
    }

    @Test
    public void interfaces_must_not_be_placed_in_implementation_packages() {
        JavaClasses classes = new ClassFileImporter().importPackagesOf(SomeInterfacePlacedInTheWrongPackage.class);

        noClasses().that().resideInAPackage("..impl..").should().beInterfaces().check(classes);
    }
}
```

interface rules

ArchUnit

https://www.archunit.org/

```
public class InterfaceRules {

    @Test
    public void interfaces_should_not_have_names_ending_with_the_word_interface() {
        JavaClasses classes = new ClassFileImporter().importClasses(
                SomeBusinessInterface.class,
                SomeDao.class
        );

        noClasses().that().areInterfaces().should().haveNameMatching(".*Interface")
    }

    @Test
    public void interfaces_should_not_have_simple_class_names_ending_with_the_word
```



Package by layer (horizontal slicing)

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

```
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);
    }
```

ArchUnit

https://www.archunit.org/

```
@Test
public void third_party_class_should_only_be_instantiated_via_workaround() {
    classes().should(notCreateProblematicClassesOutsideOfWorkaroundFactory()
            .as(THIRD_PARTY_CLASS_RULE_TEXT))
            .check(classes);
}

private ArchCondition<JavaClass> notCreateProblematicClassesOutsideOfWorkaroundFactory() {
    DescribedPredicate<JavaCall<?>> constructorCallOfThirdPartyClass =
            target(is(constructor())).and(targetOwner(is(assignableTo(ThirdPartyClassWithProblem.class))));

    DescribedPredicate<JavaCall<?>> notFromWithinThirdPartyClass =
            originOwner(is(not(assignableTo(ThirdPartyClassWithProblem.class)))).forSubType();

    DescribedPredicate<JavaCall<?>> notFromWorkaroundFactory =
            originOwner(is(not(equivalentTo(ThirdPartyClassWorkaroundFactory.class)))).forSubType();

    DescribedPredicate<JavaCall<?>> targetIsIllegalConstructorOfThirdPartyClass =
            constructorCallOfThirdPartyClass.
                    and(notFromWithinThirdPartyClass).
                    and(notFromWorkaroundFactory);

    return never(callCodeUnitWhere(targetIsIllegalConstructorOfThirdPartyClass));
}
```

governance

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;
```

NetArchTest

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

```
// Only classes in the data namespace can have a dependency on System.Data
result = Types.InCurrentDomain()
    .That().HaveDependencyOn("System.Data")
    .And().ResideInNamespace(("ArchTest"))
    .Should().ResideInNamespace(("NetArchTest.SampleLibrary.Data"))
    .GetResult().IsSuccessful;
```

NetArchTest

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

```
// All the classes in the data namespace should implement IRepository
result = Types.InCurrentDomain()
    .That().ResideInNamespace(("NetArchTest.SampleLibrary.Data"))
    .And().AreClasses()
    .Should().ImplementInterface(typeof(IRepository<>))
    .GetResult().IsSuccessful;

// Classes that implement IRepository should have the suffix "Repository"
result = Types.InCurrentDomain()
    .That().ResideInNamespace(("NetArchTest.SampleLibrary.Data"))
    .And().AreClasses()
    .Should().HaveNameEndingWith("Repository")
    .GetResult().IsSuccessful;

// Classes that implement IRepository must reside in the Data namespace
result = Types.InCurrentDomain()
    .That().ImplementInterface(typeof(IRepository<>))
    .Should().ResideInNamespace(("NetArchTest.SampleLibrary.Data"))
    .GetResult().IsSuccessful;

// All the service classes should be sealed
result = Types.InCurrentDomain()
    .That().ImplementInterface(typeof(IWidgetService))
    .Should().BeSealed()
    .GetResult().IsSuccessful;
```

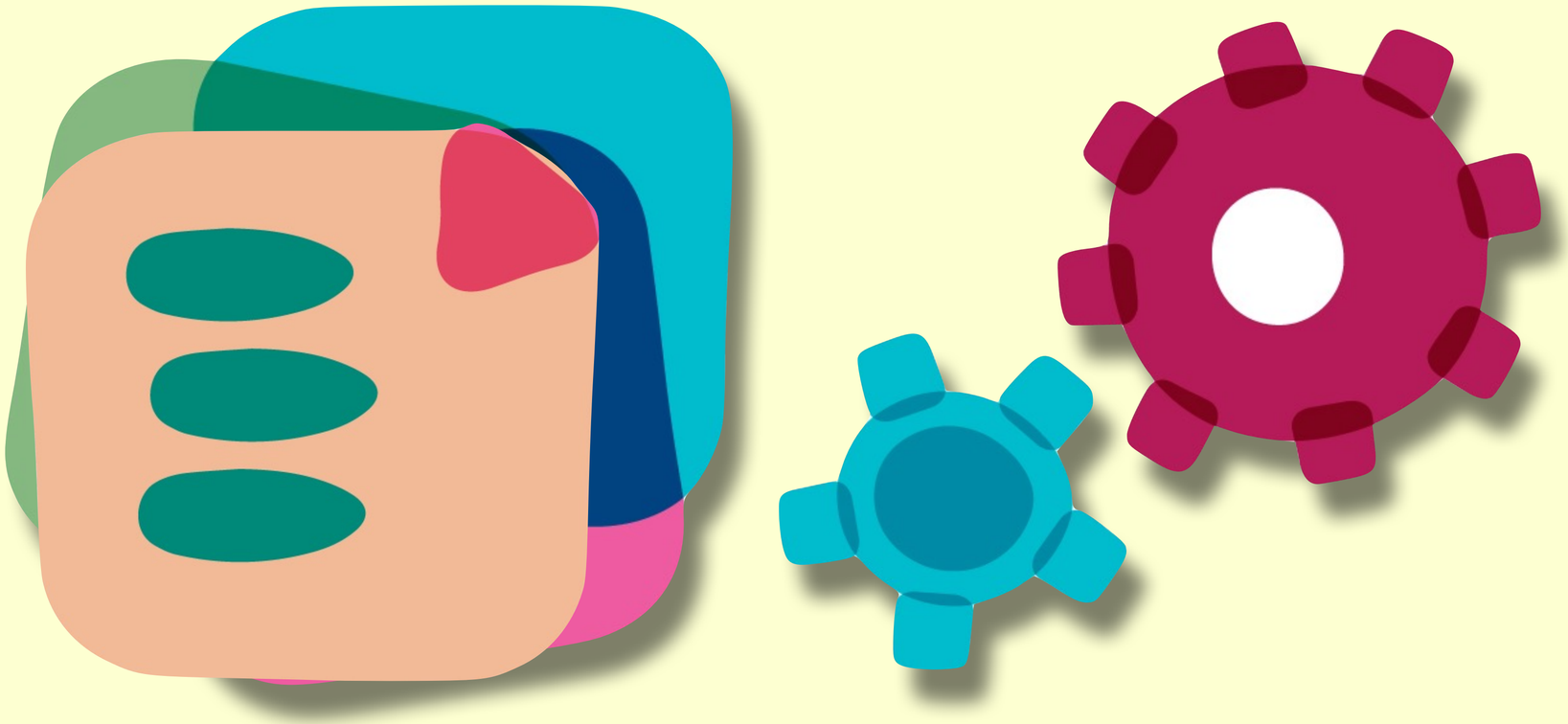
Legality of Open Source Libraries

Legality of Open Source Libraries

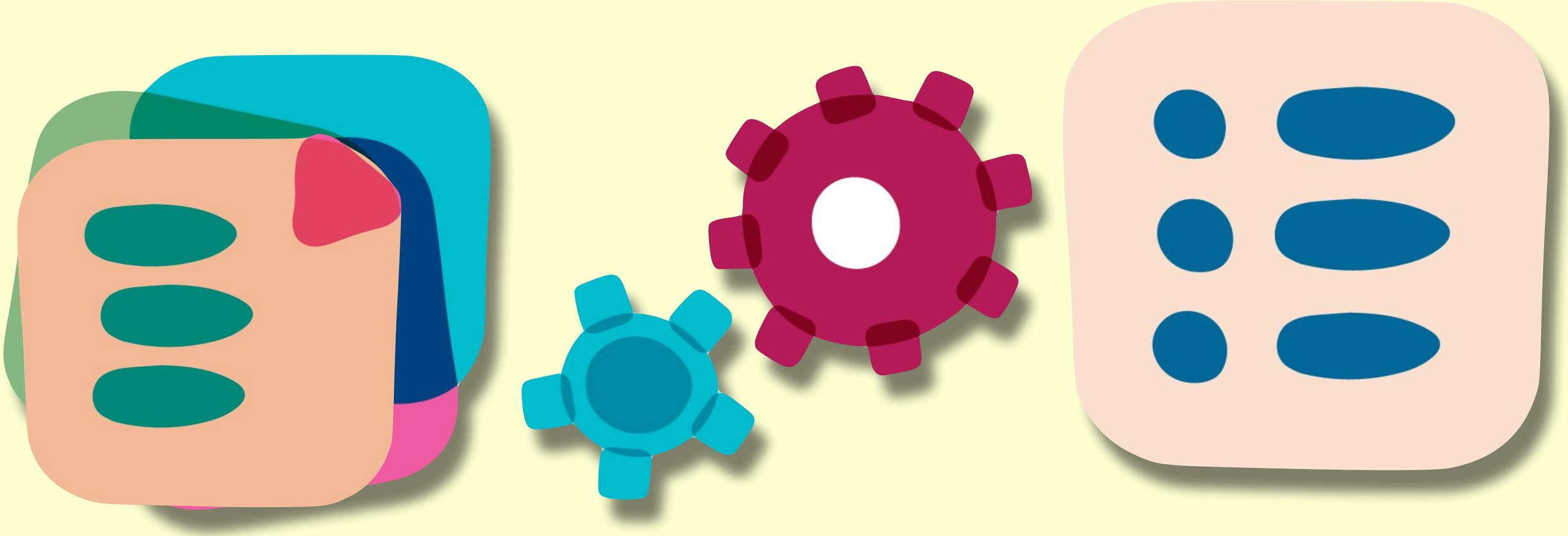
Penultima⬆⊠e

Legality of Open Source Libraries

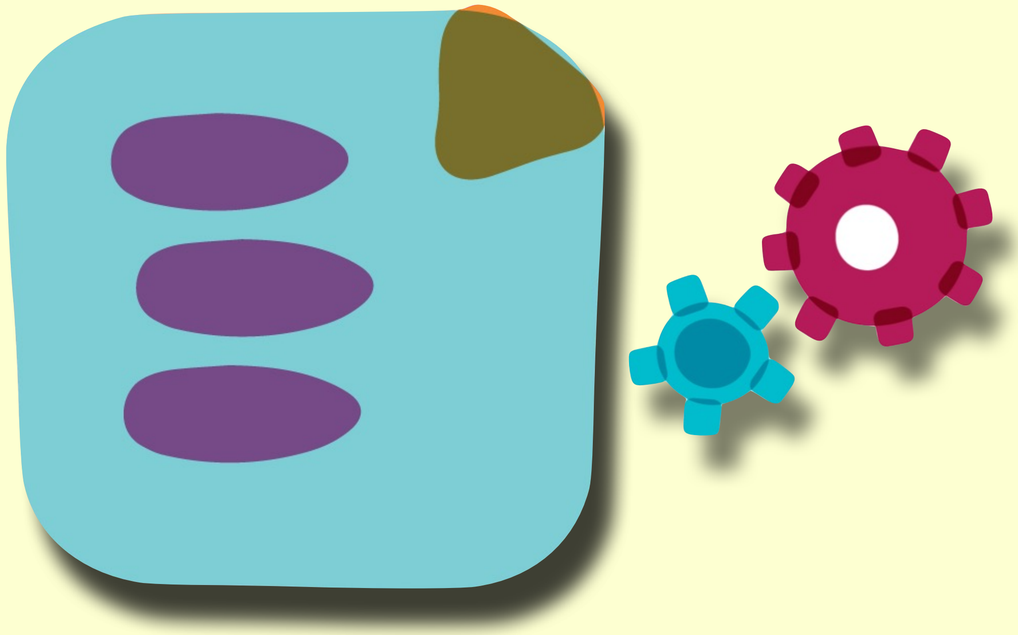
Legality of Open Source Libraries



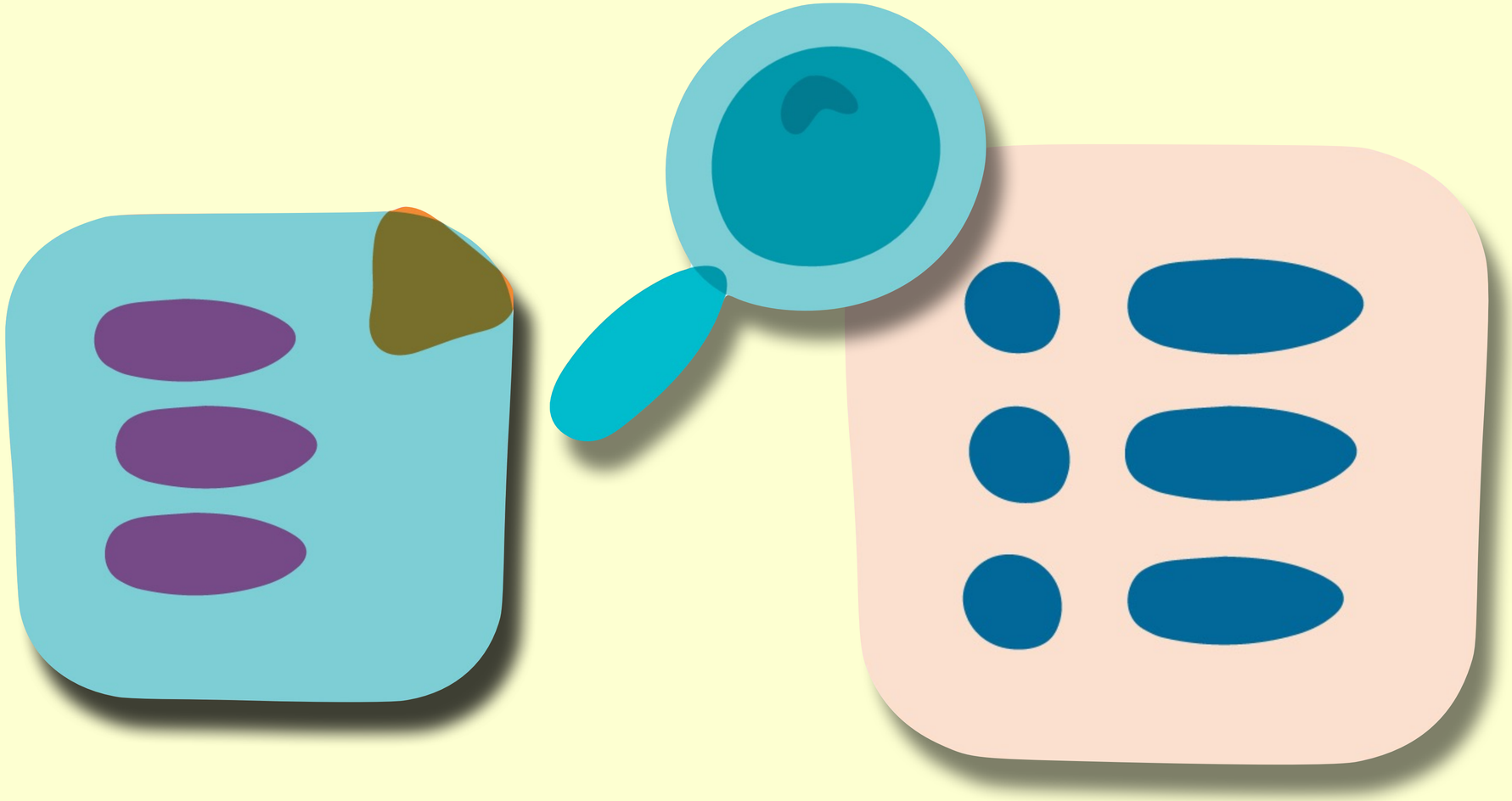
Legality of Open Source Libraries



Legality of Open Source Libraries

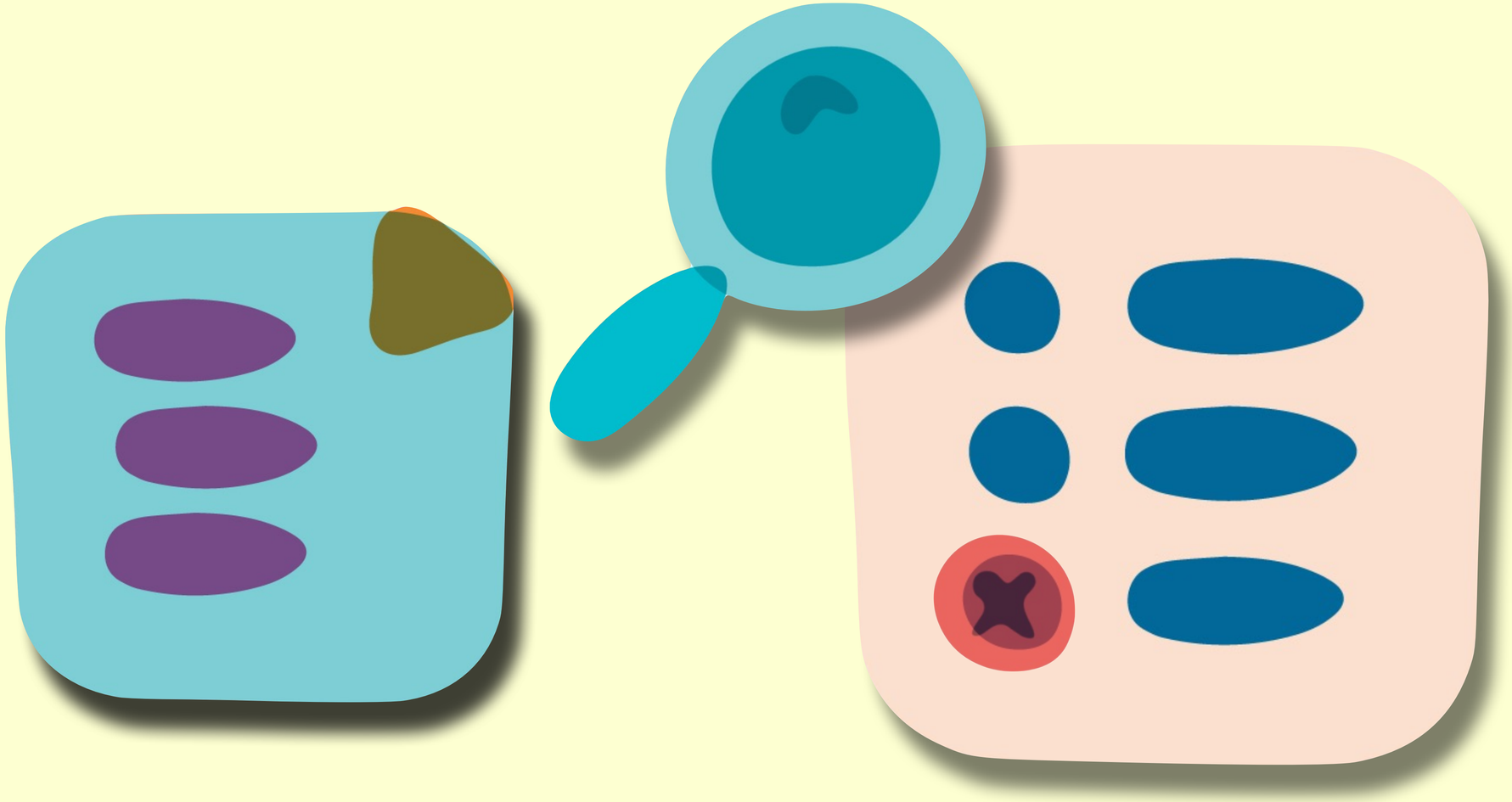


Legality of Open Source Libraries



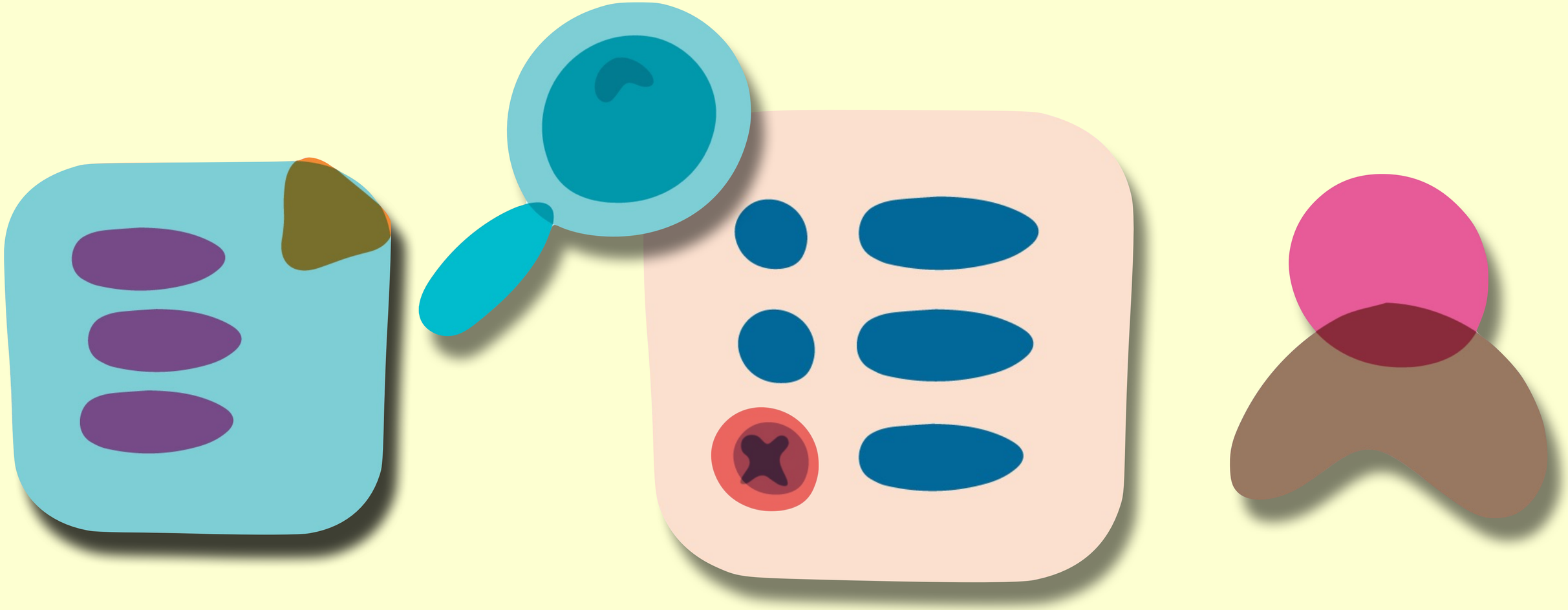
Penultima⬆⊠e

Legality of Open Source Libraries



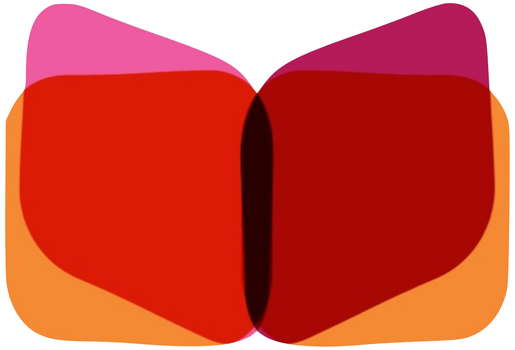
Penultima⬆⊠e

Legality of Open Source Libraries

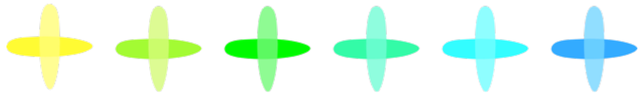


Penultima⬆⊠e

Agenda Part 1

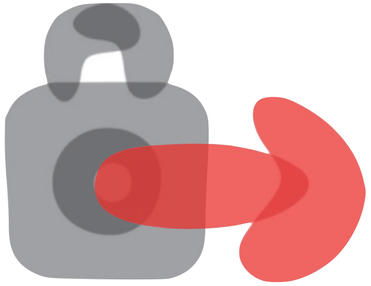


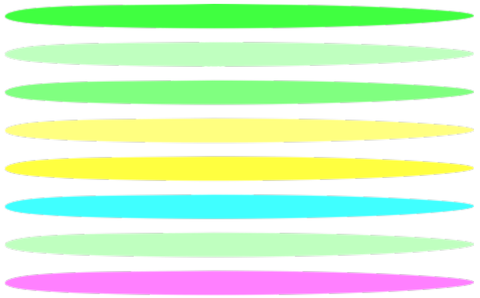
Definition



Incremental Change

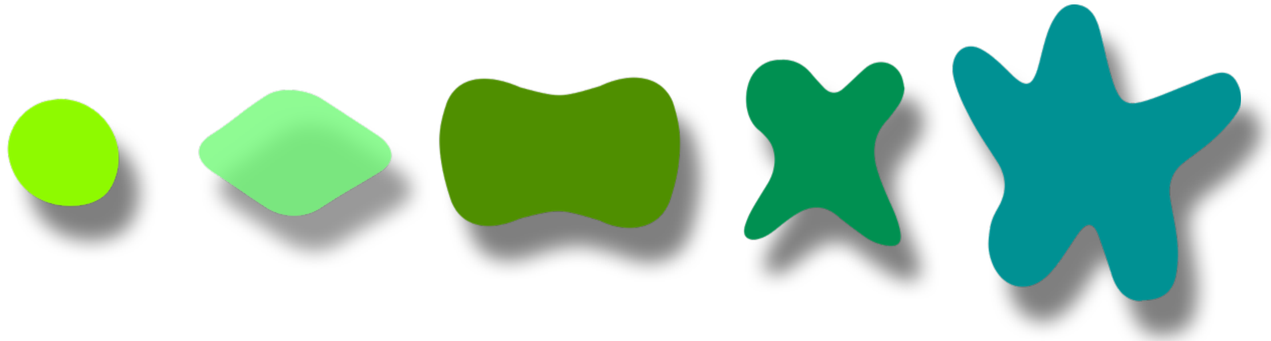
Guided Change via Fitness Functions



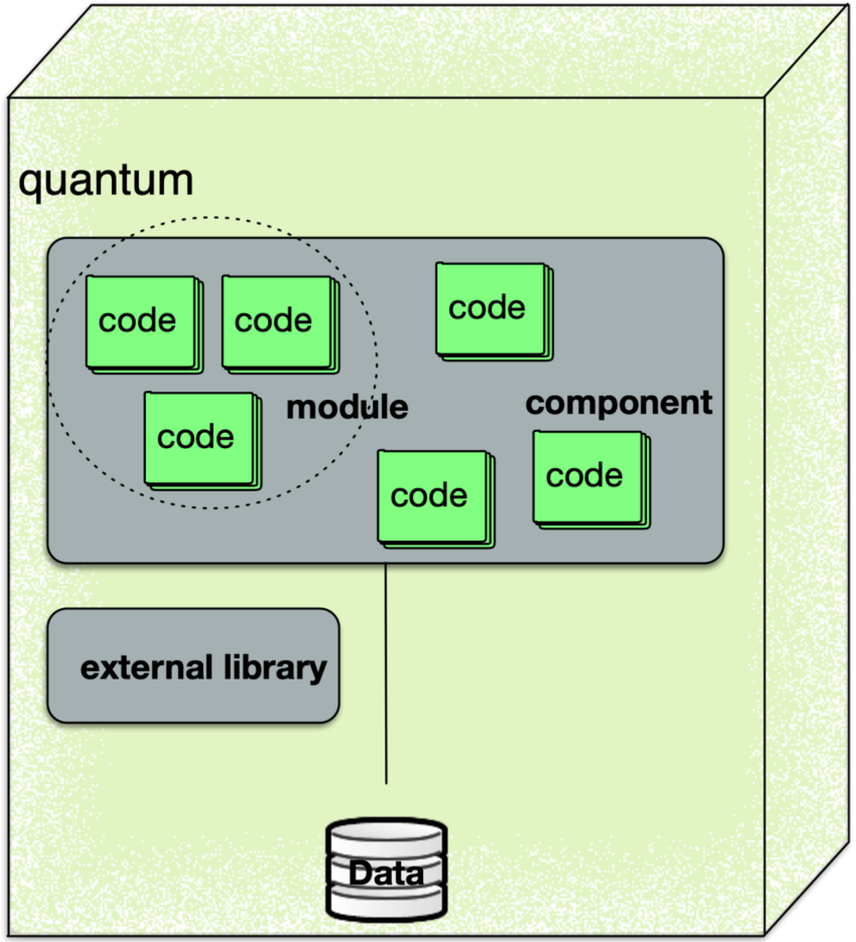


Structuring architecture for evolution

Two Big Ideas



fitness functions for evolvability



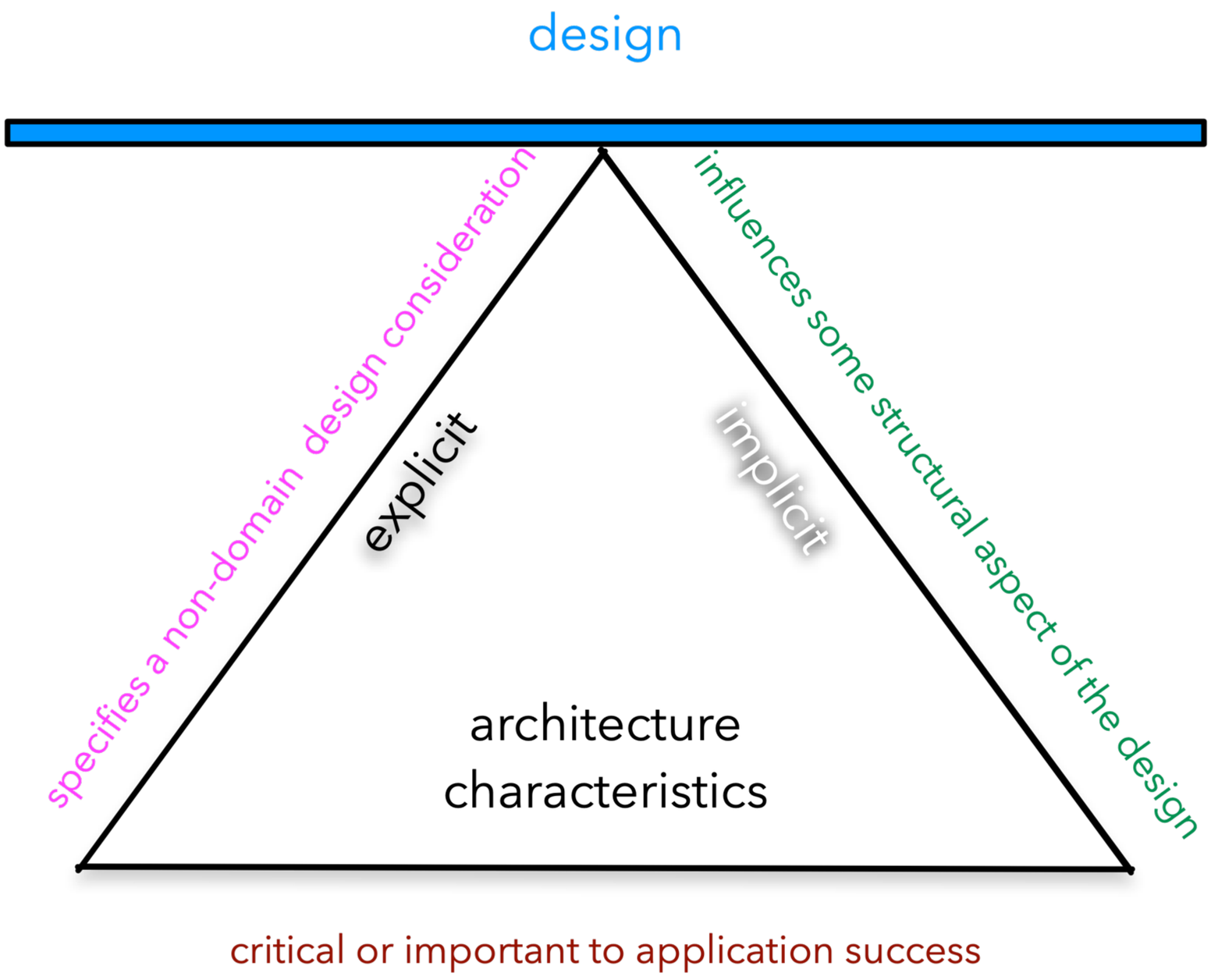
architectural quantum

O'REILLY®

Fundamentals of Software Architecture

An Engineering Approach

Mark Richards & Neal Ford

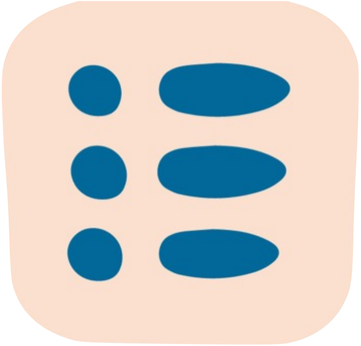

design
specifies a non-domain design consideration
influences some structural aspect of the design
explicit
implicit
architecture
characteristics
critical or important to application success

Architecture Characteristics

1. synergistic

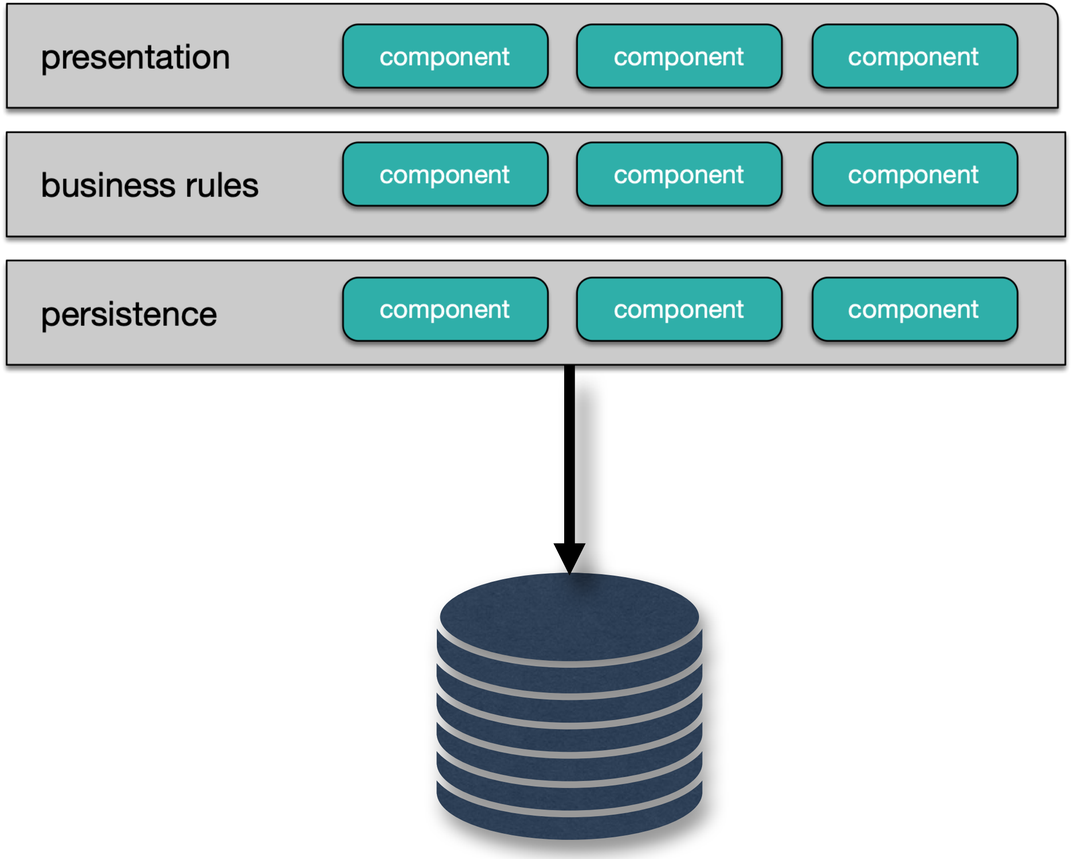
2. ill-defined

4. numerous categories

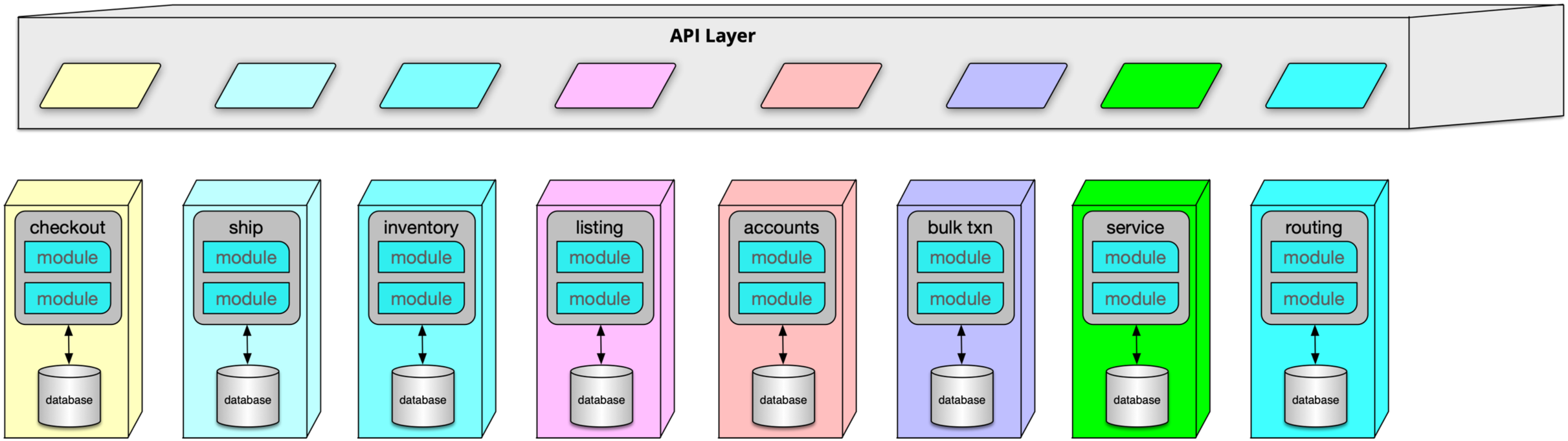


3. voluminous

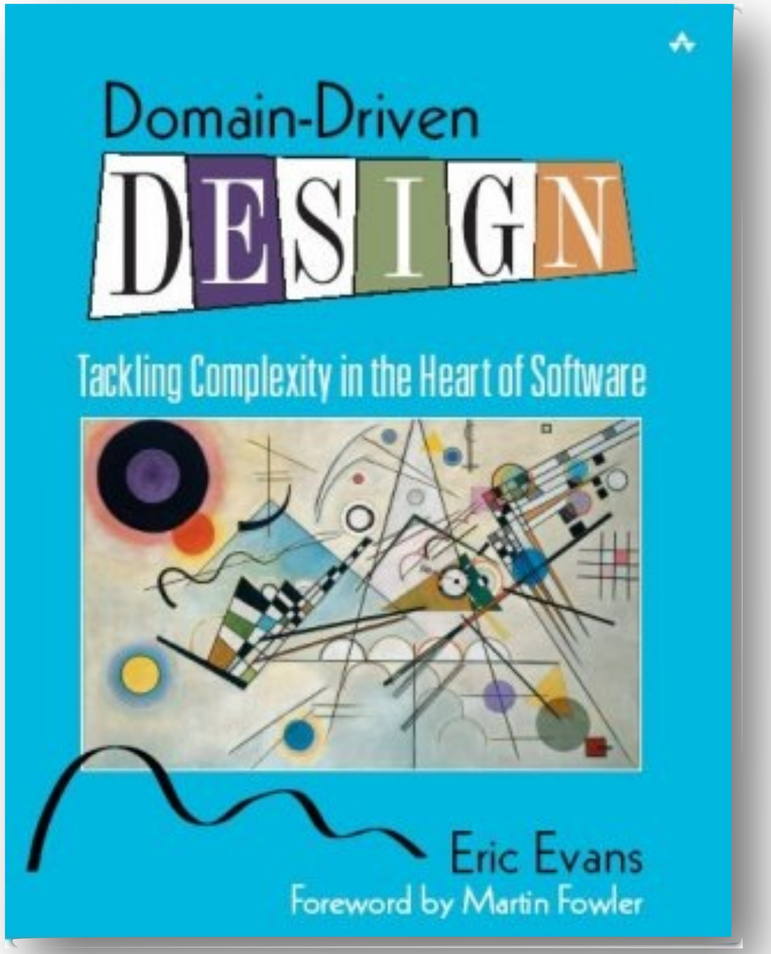
Old Axiom

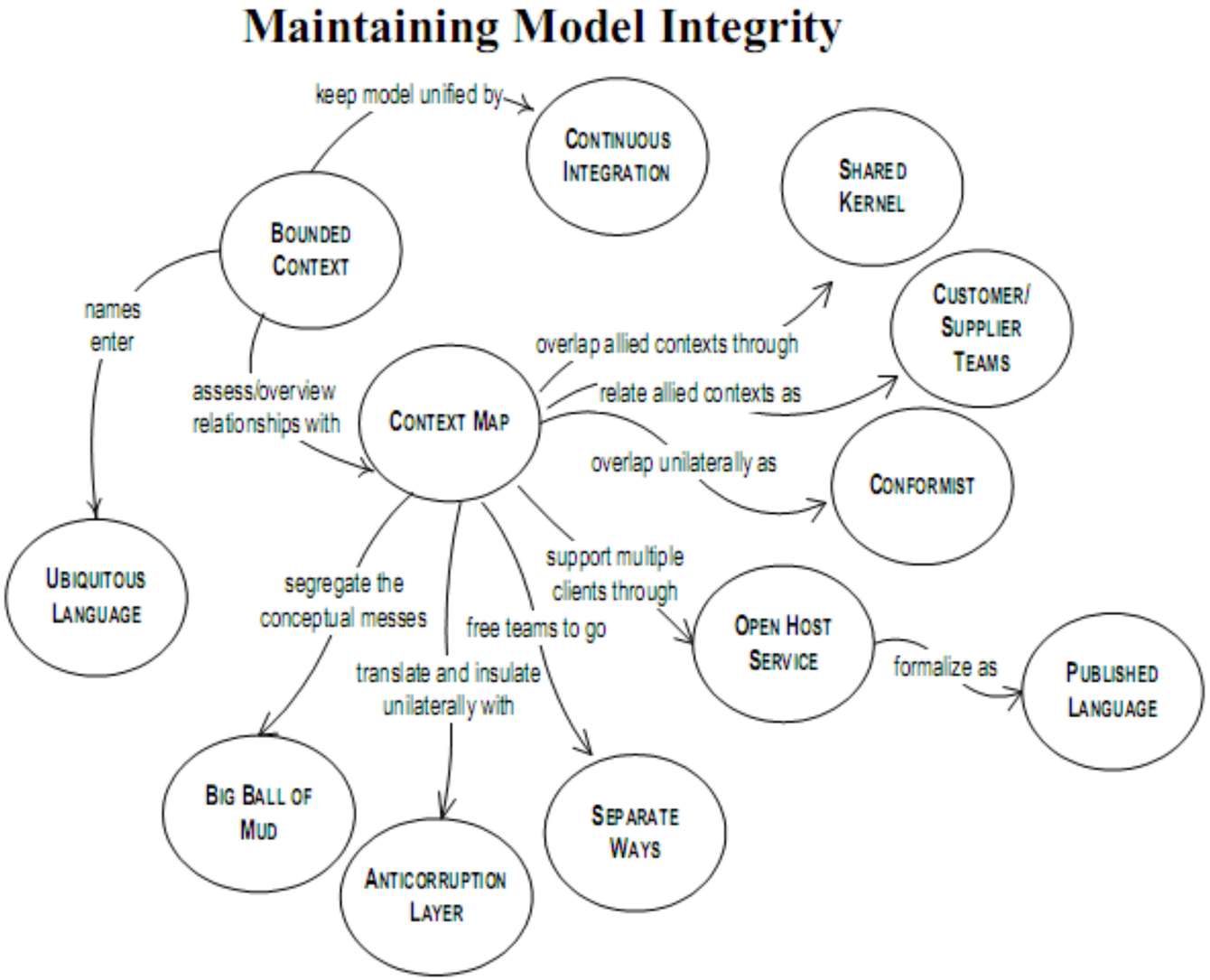

presentation
component
component
component
business rules
component
component
component
persistence
component
component
component

Current Reality



DDD => Architecture Quantum

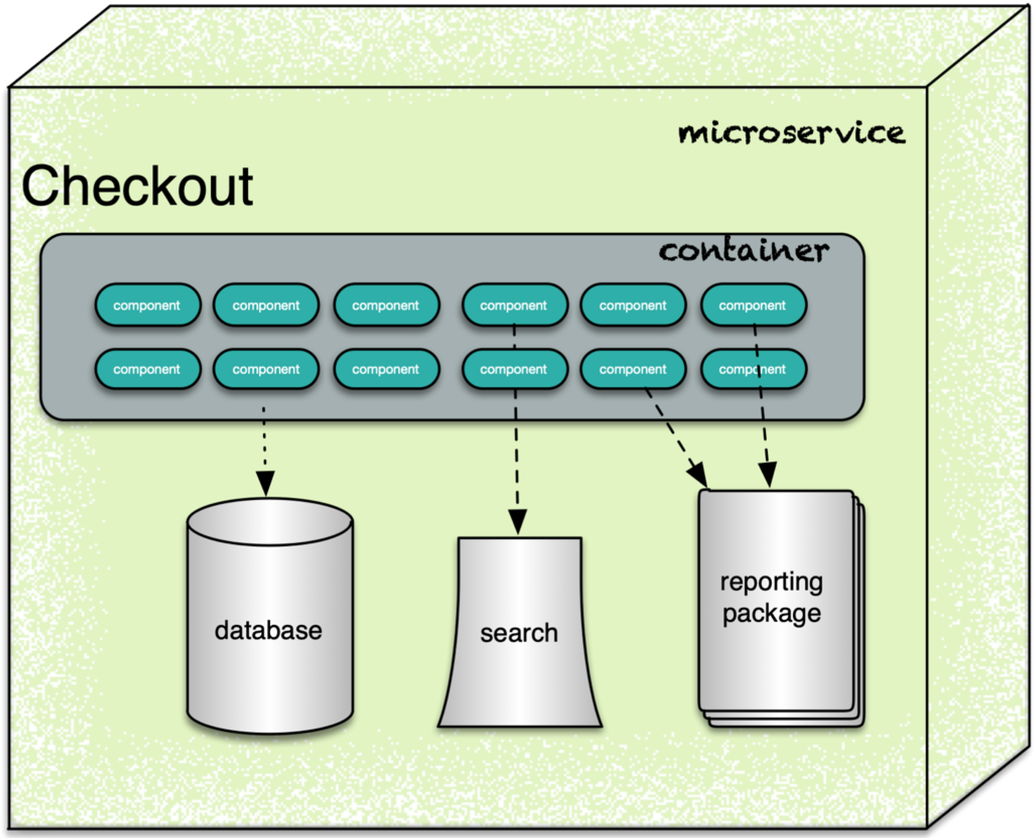




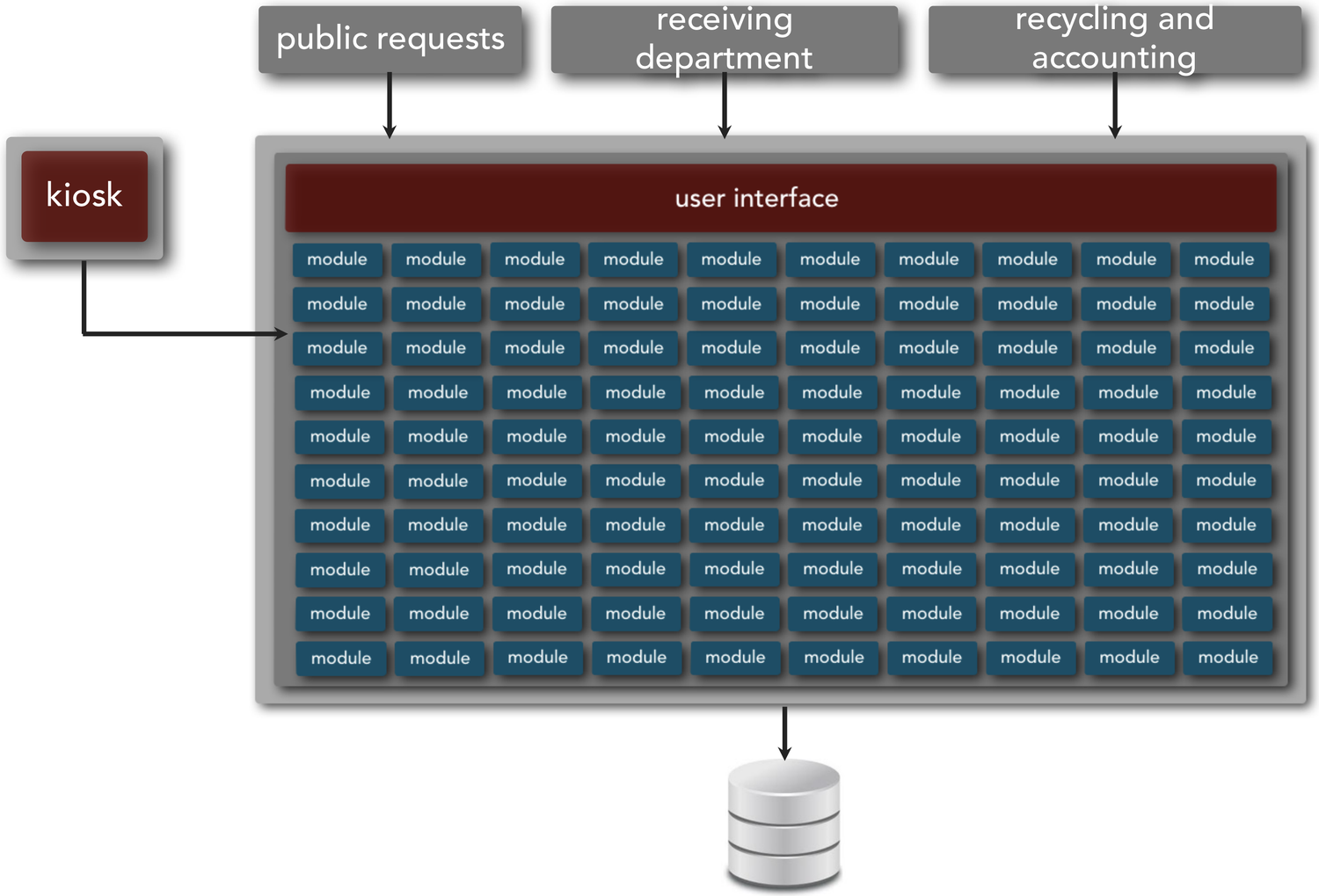
Architecture Quantum

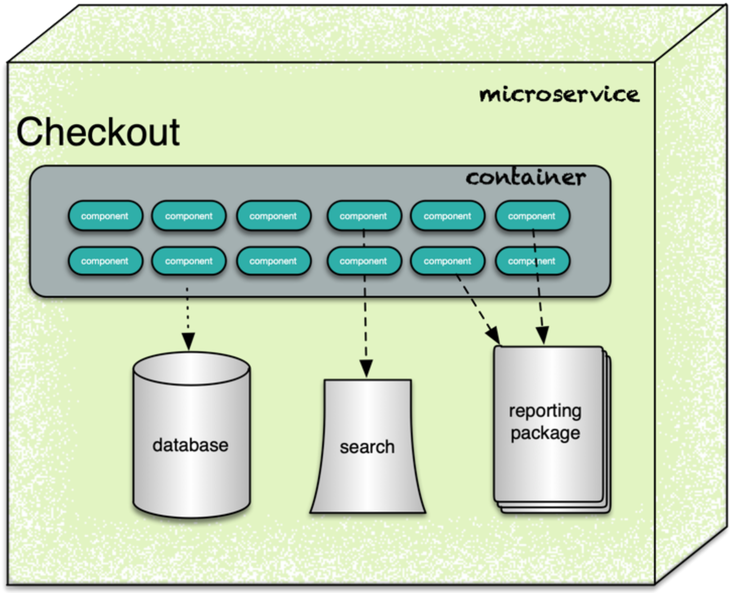
independently deployable artifact with high functional cohesion and synchronous quantum connascence

Independently deployable

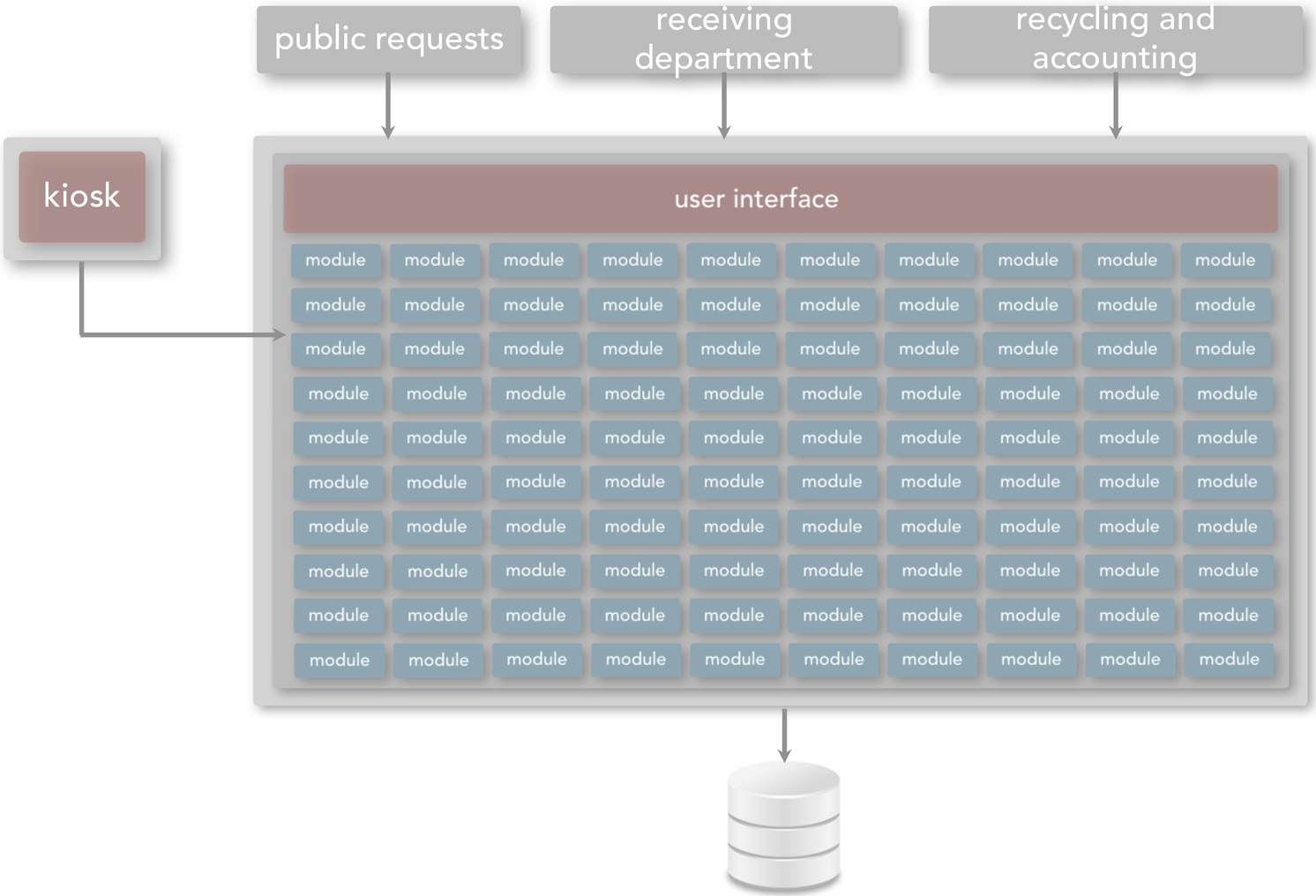


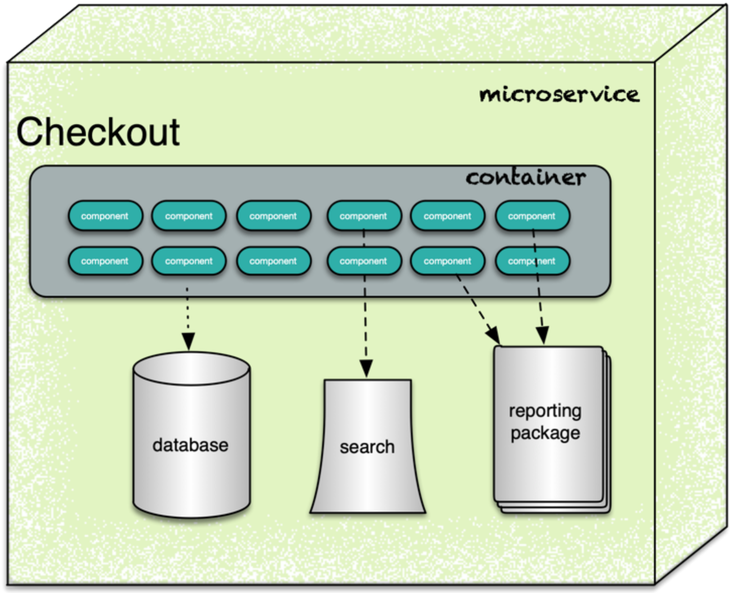
High functional cohesion



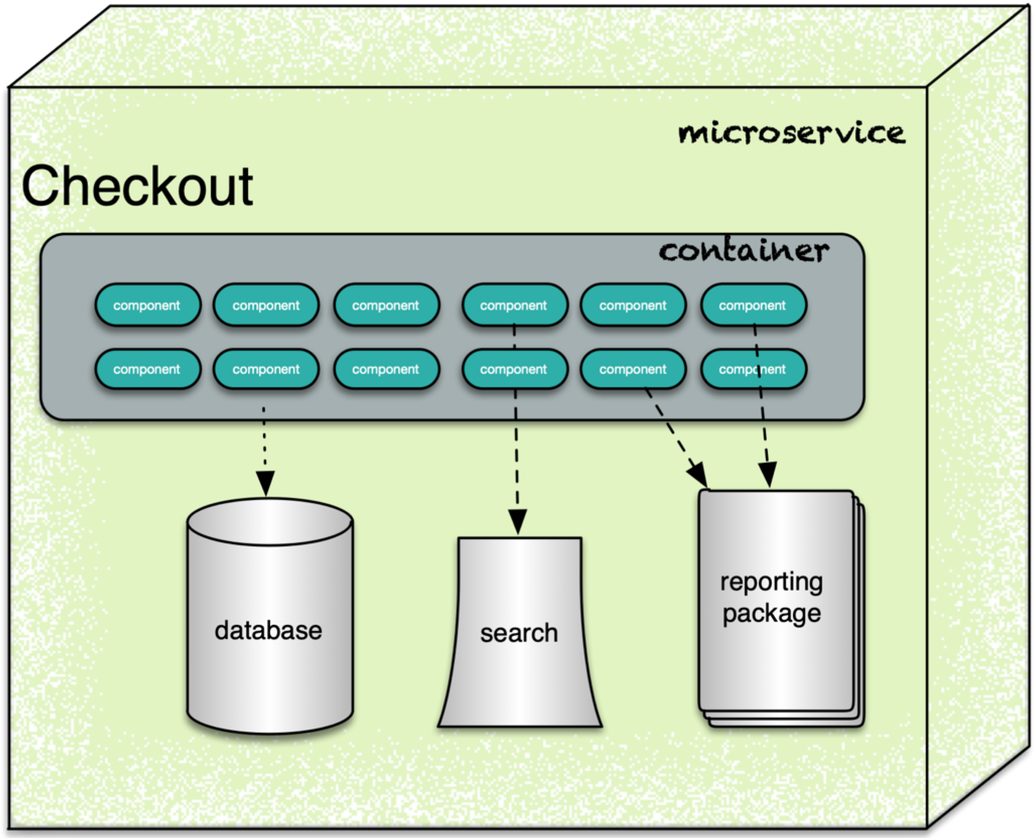


High functional cohesion

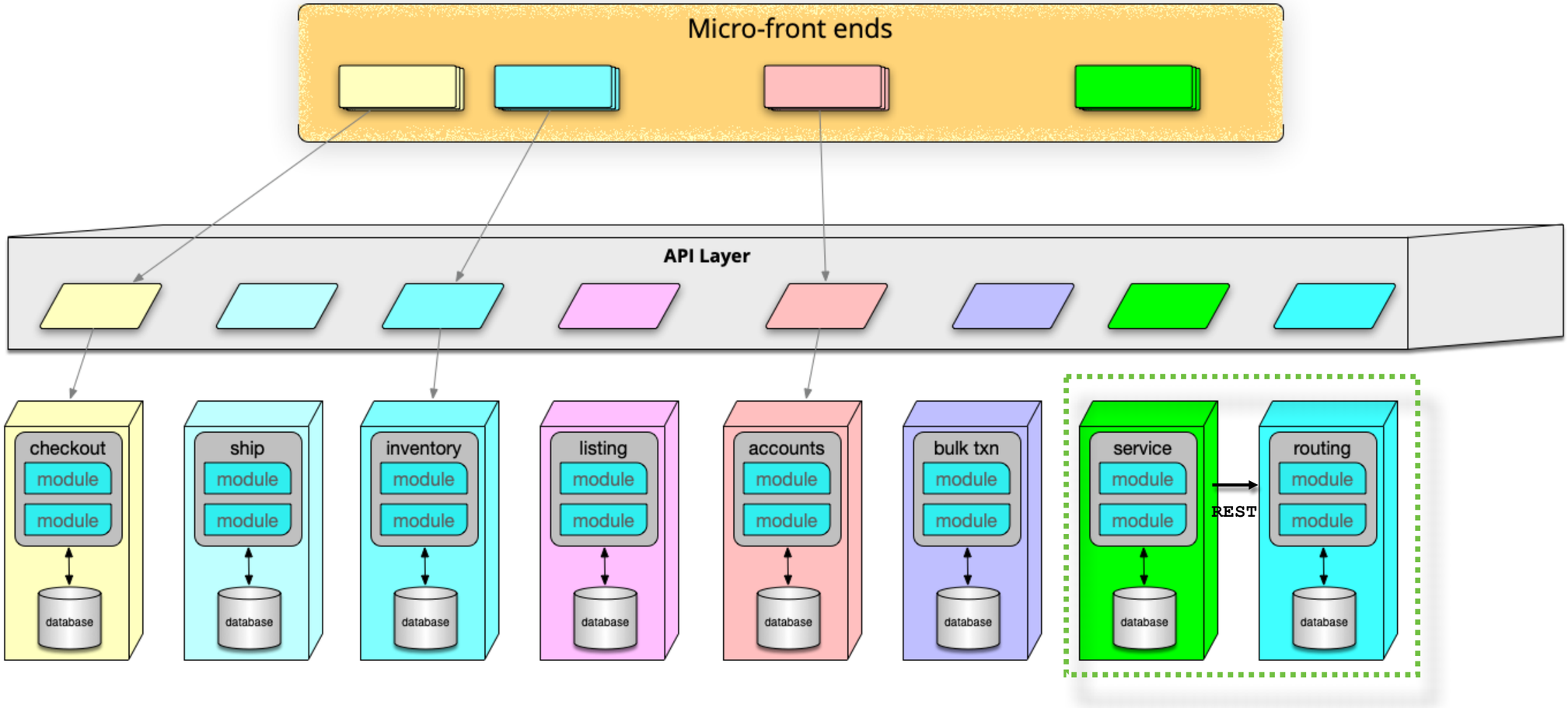




Synchronous connascence

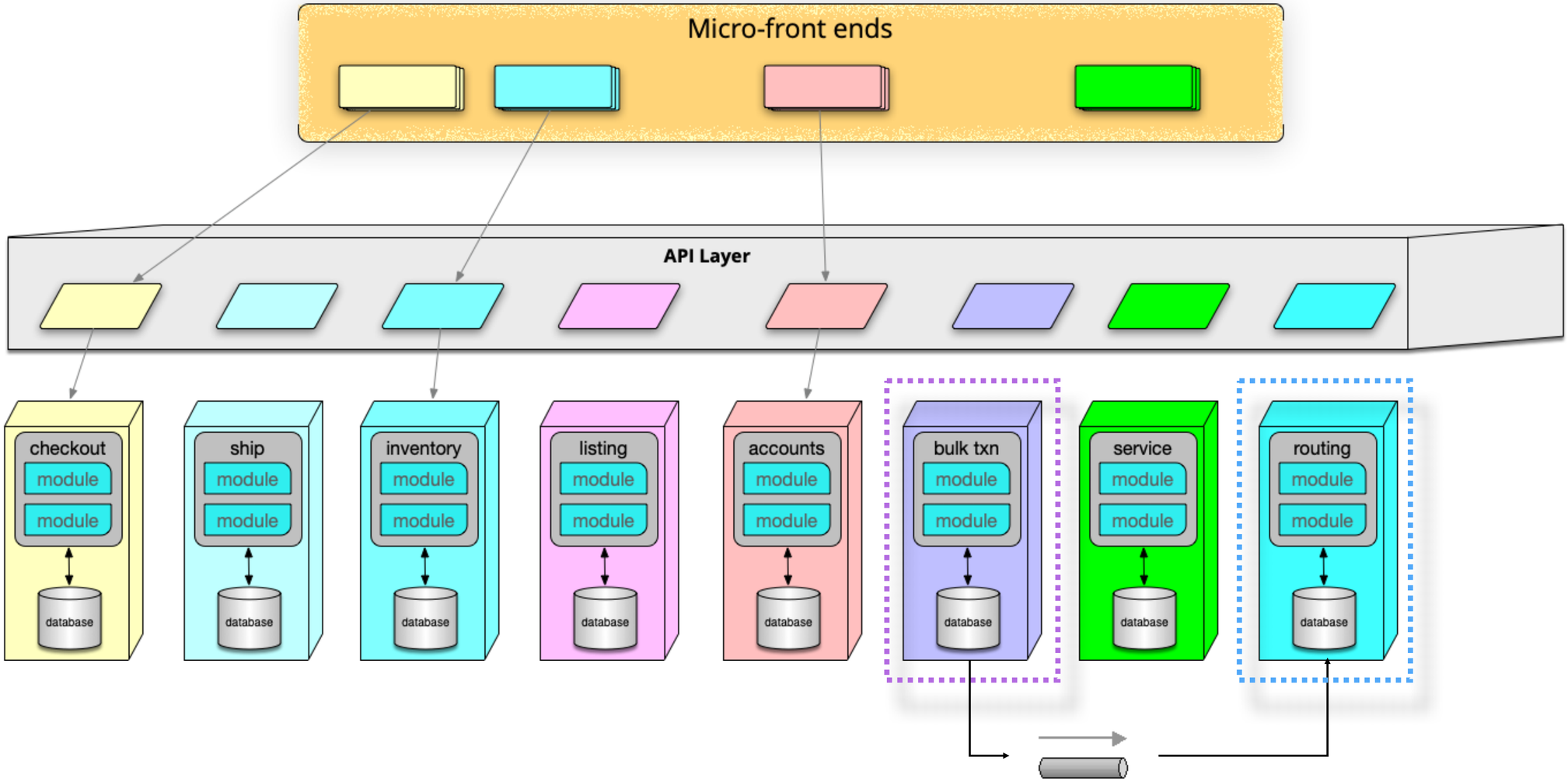


The Value of Asynchronicity



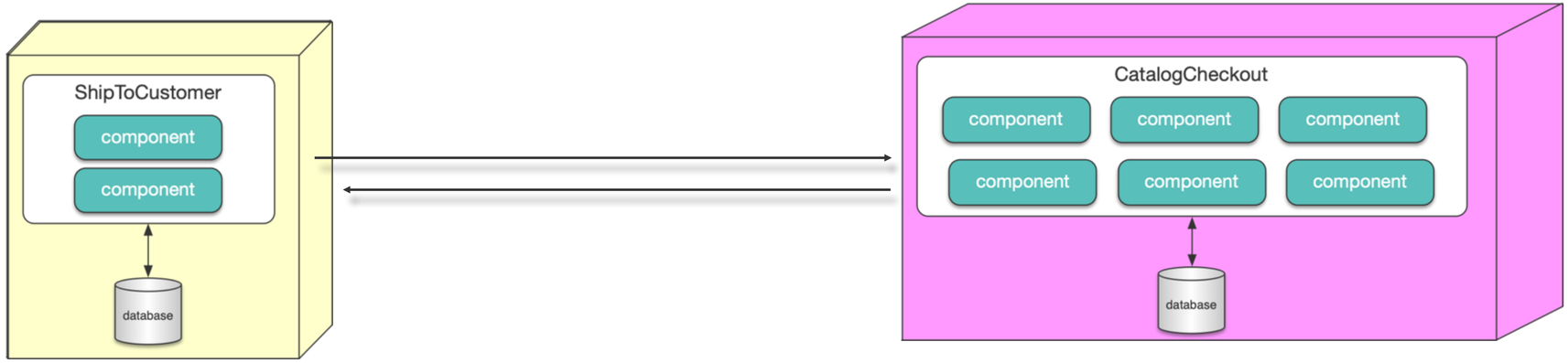
synchronous

The Value of Asynchronicity



asynchronous

Synchronous connascence

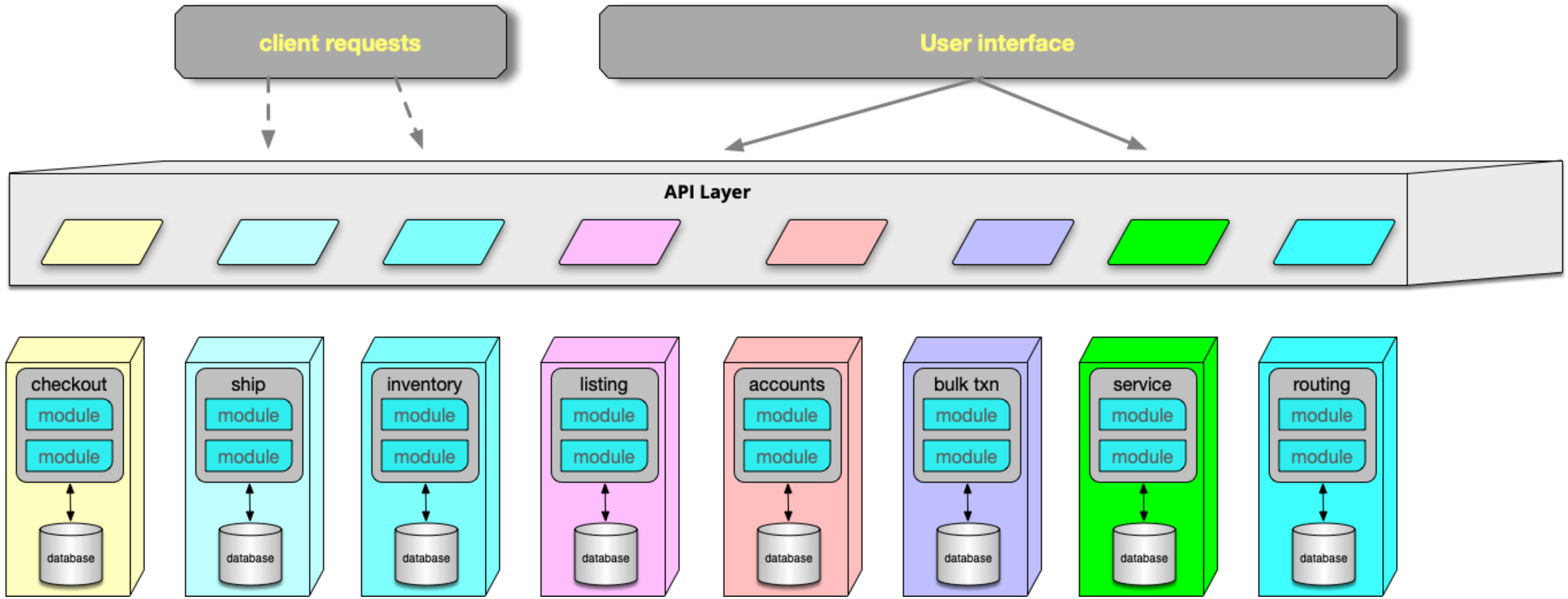


Synchronous calls expand the scope of quanta.

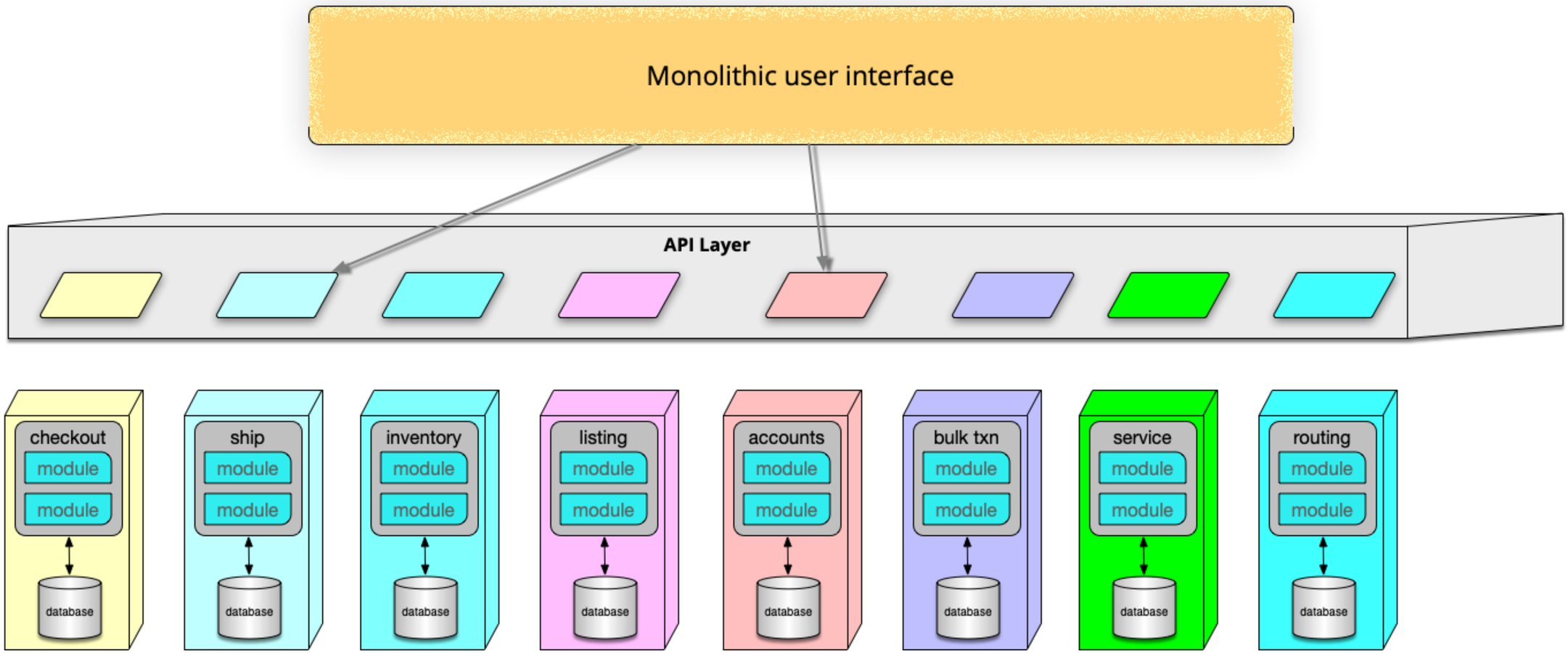
Why Quantum?

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

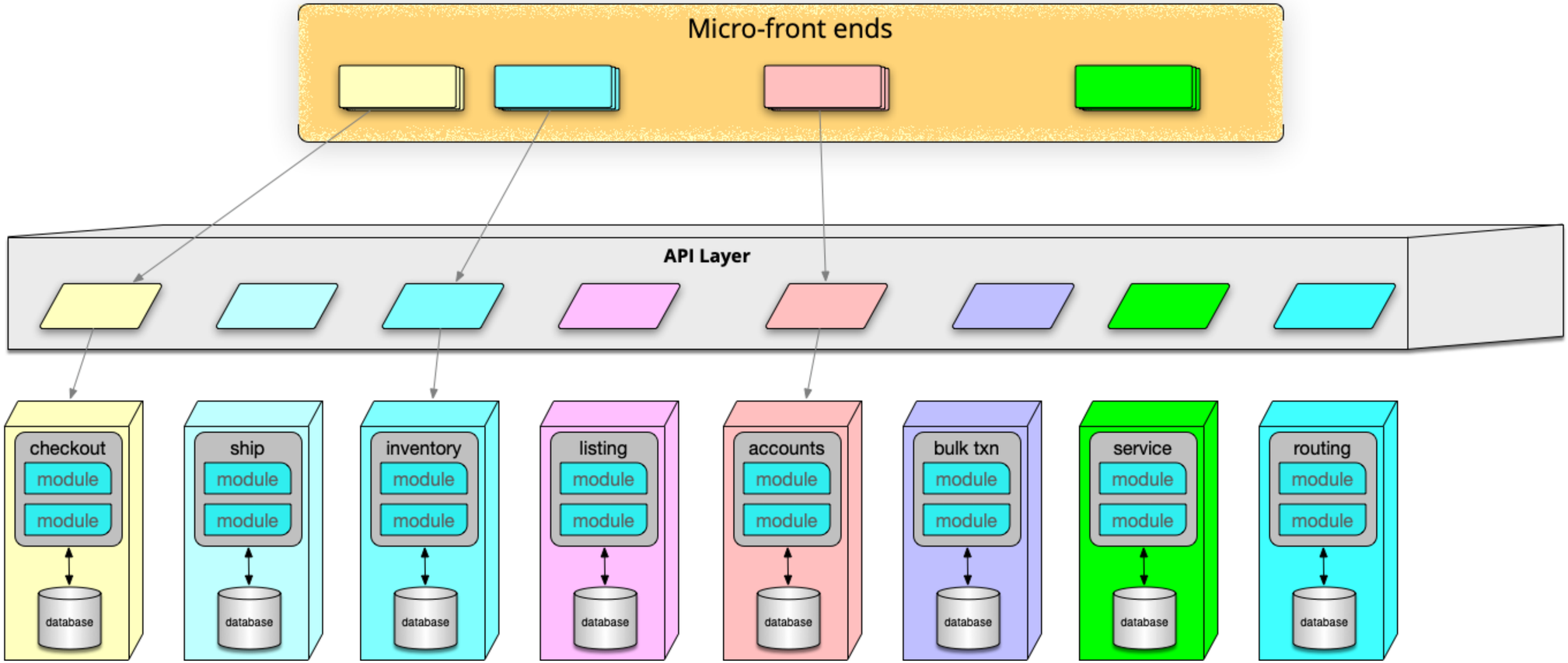
Microservices



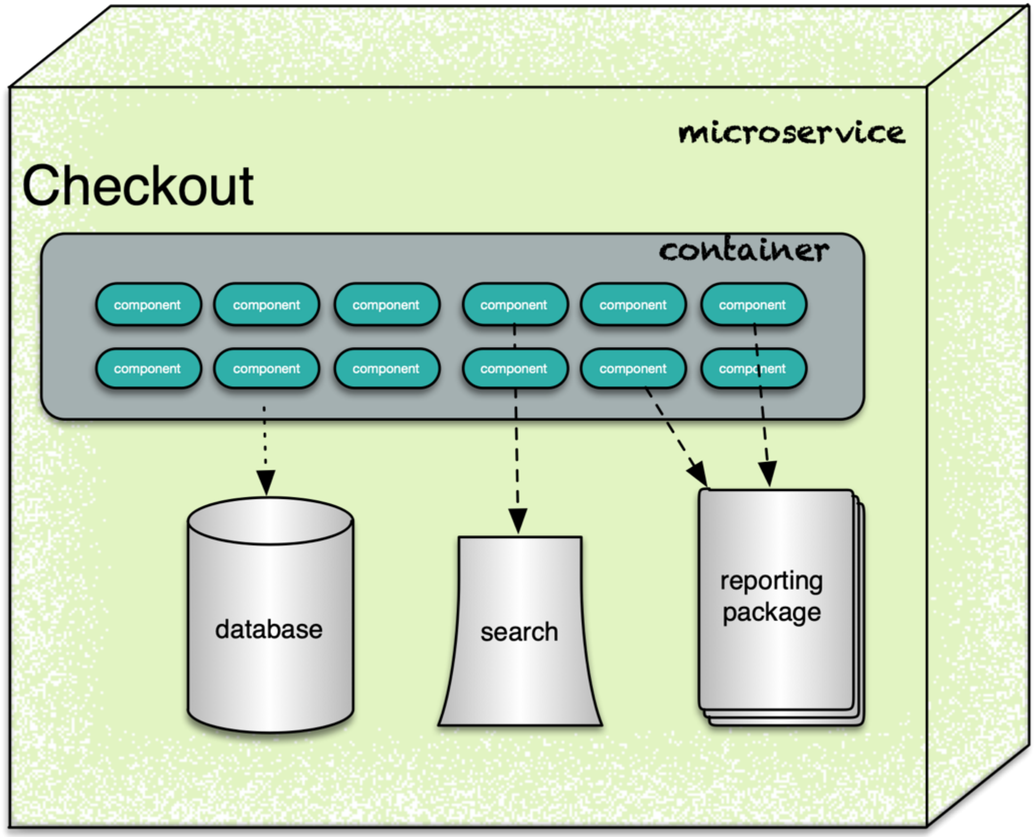
Microservices: Monolithic UI



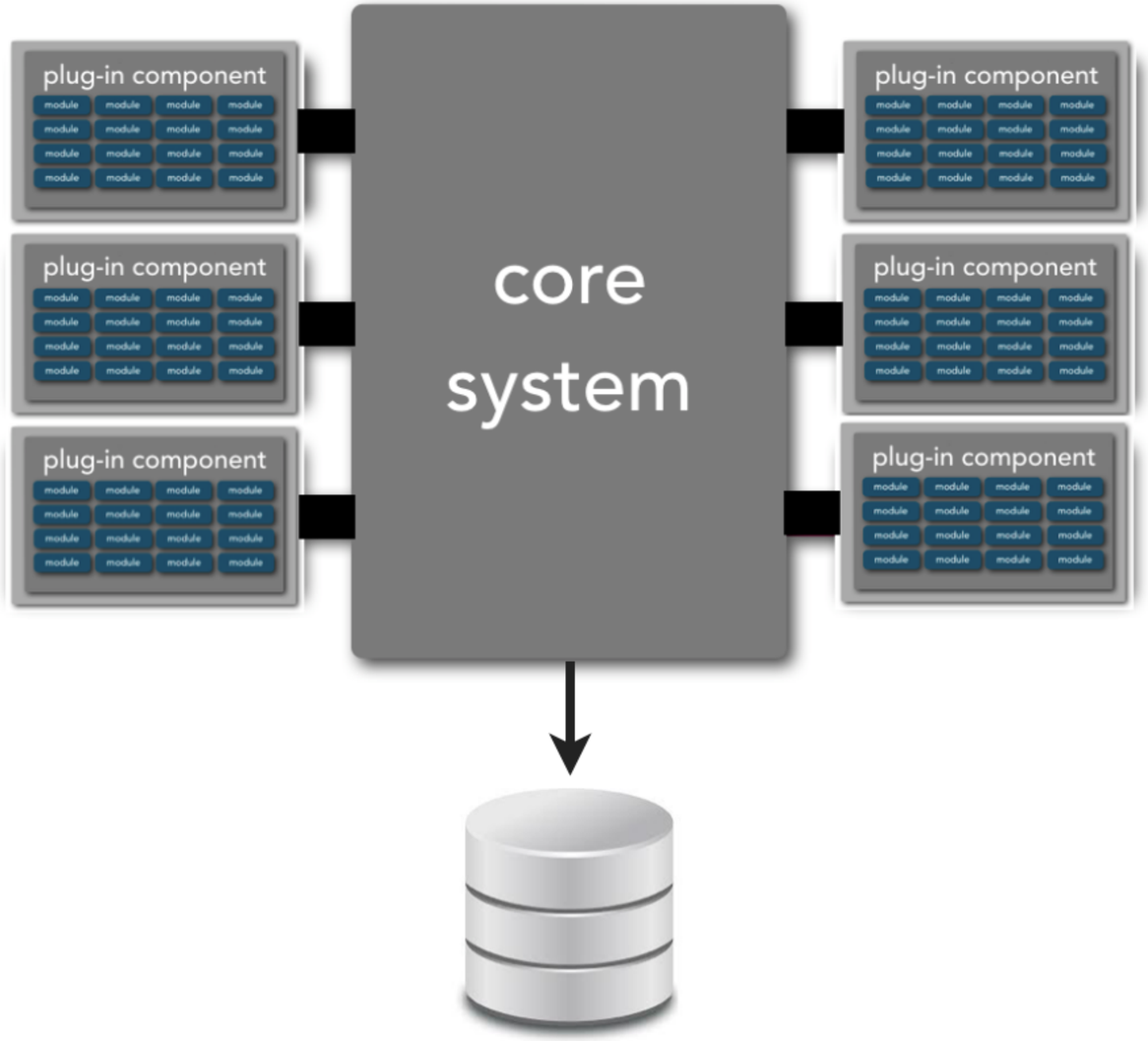
Microservices: Micro Front End



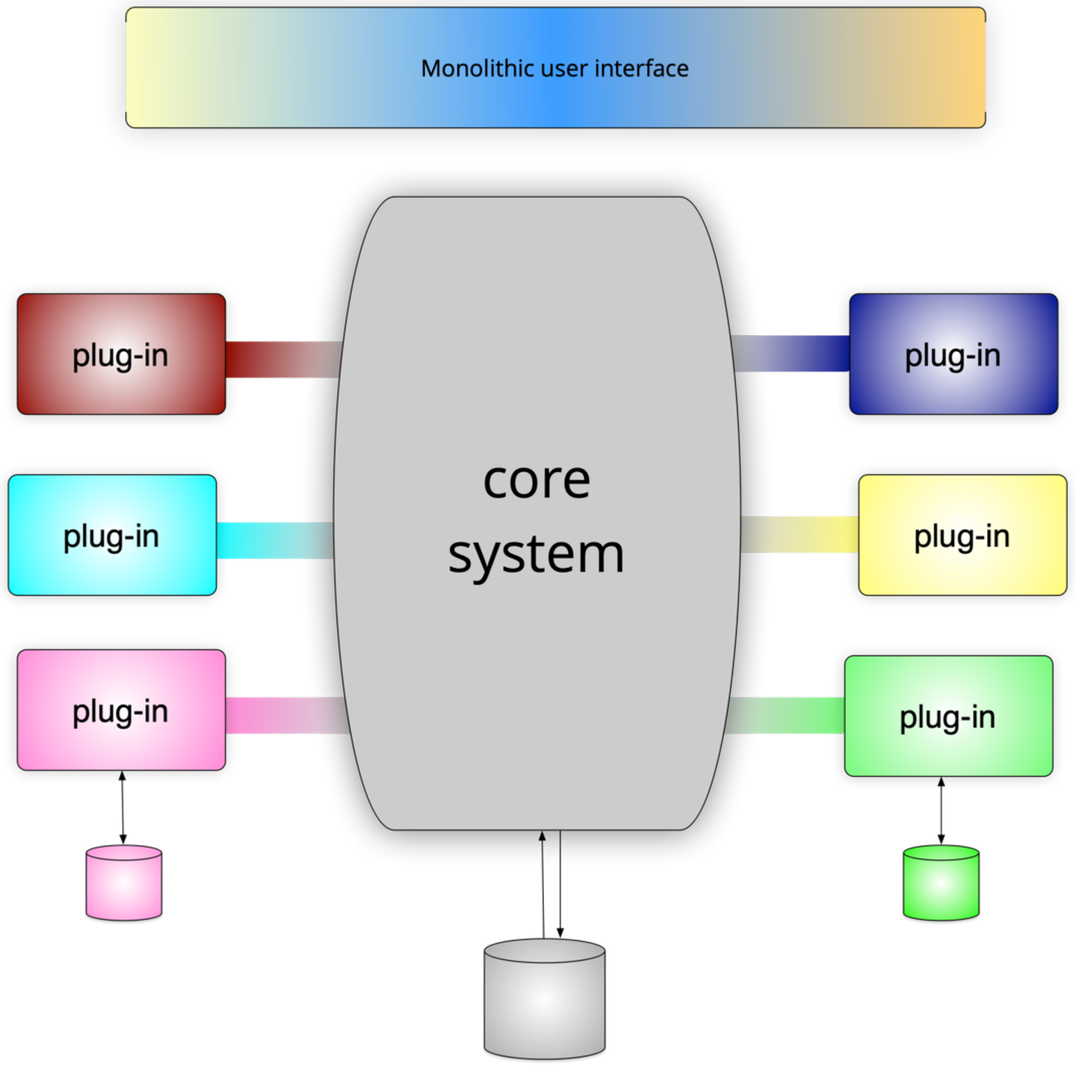
Quantum Examples



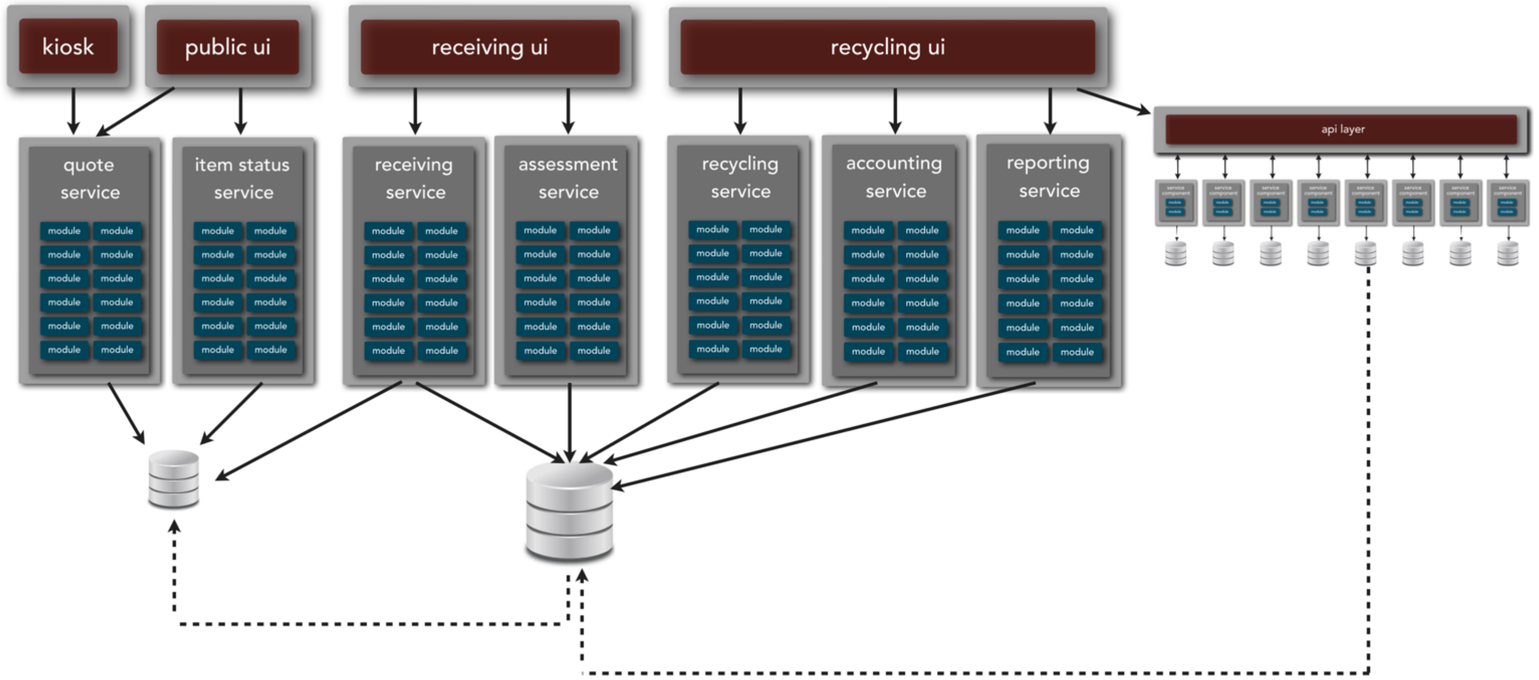
I. Microkernel w/ shared database



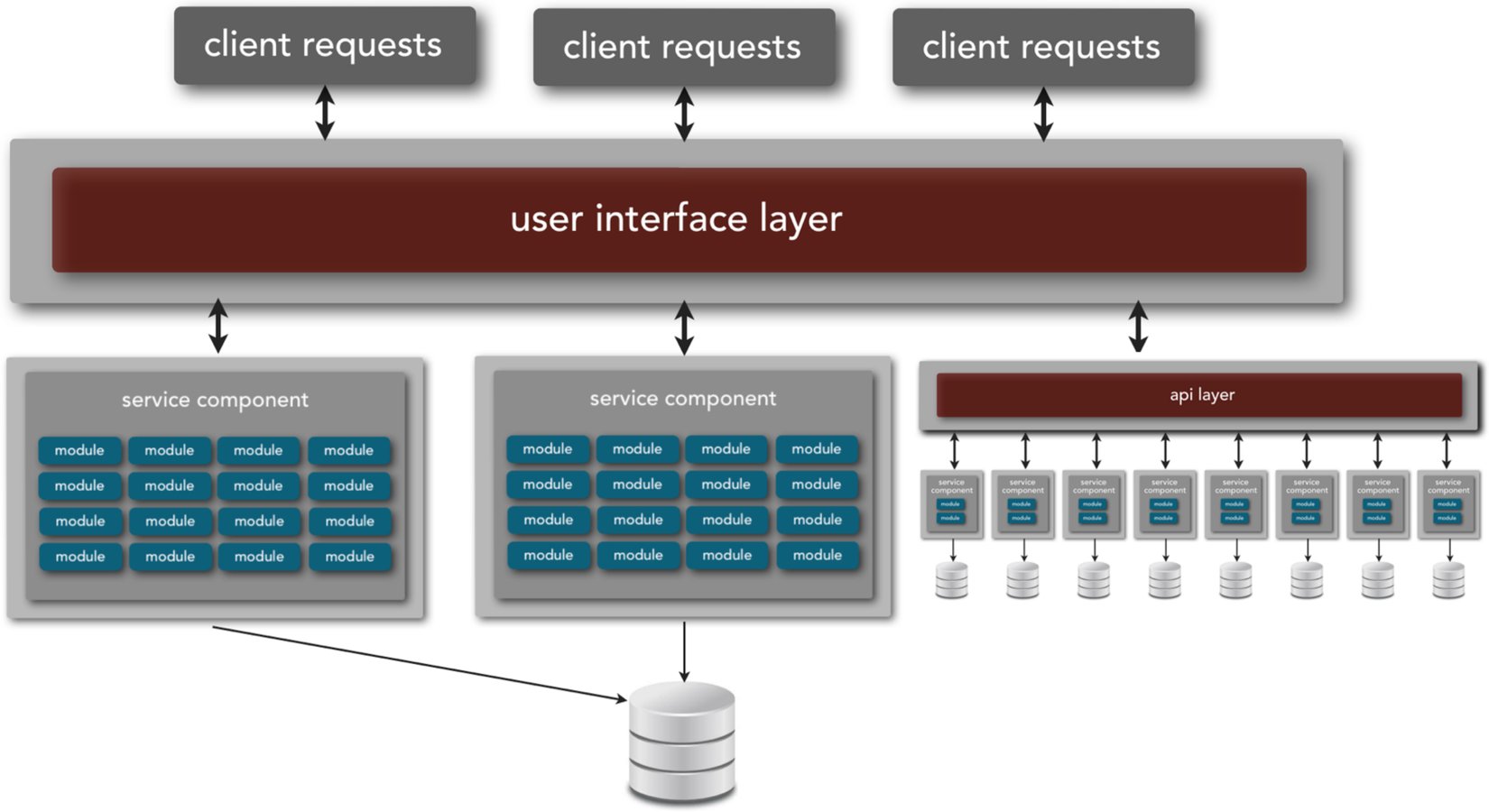
II. Microkernel w/ independent databases



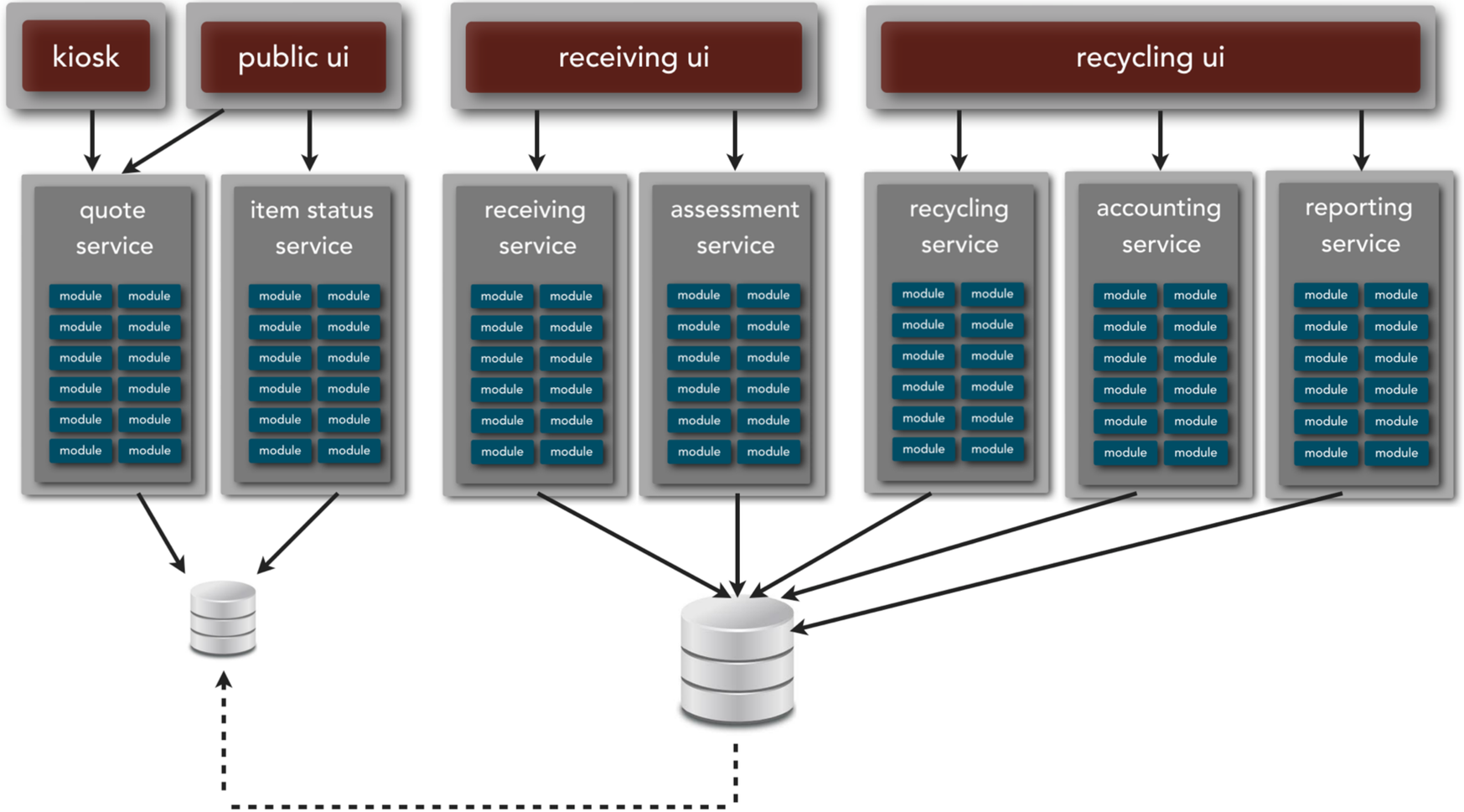
IV. Complex Hybrid



V. Complex Hybrid 2



VI. Complex Hybrid 3



Your Architectural Kata is...

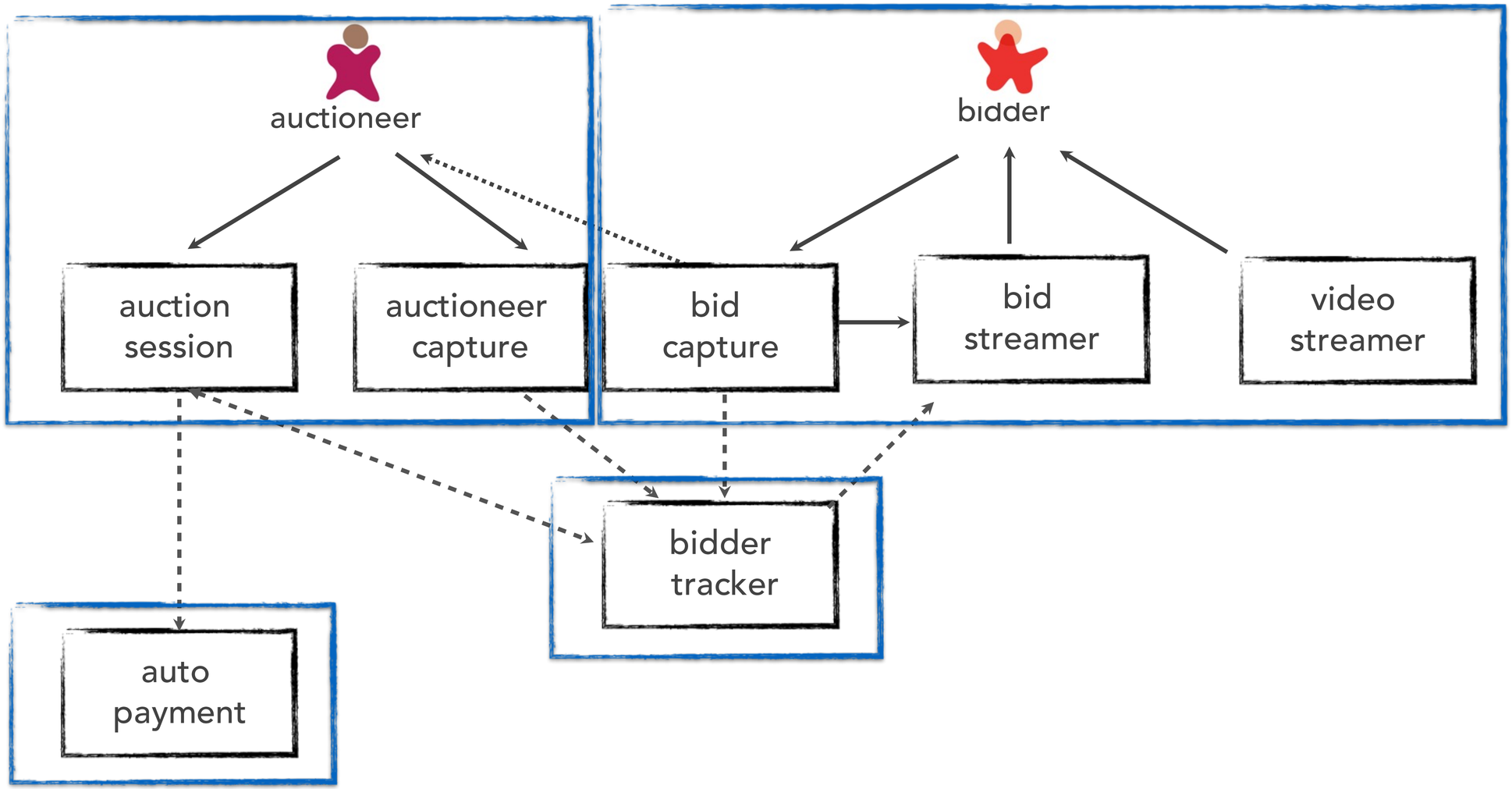
Going Going Gone!

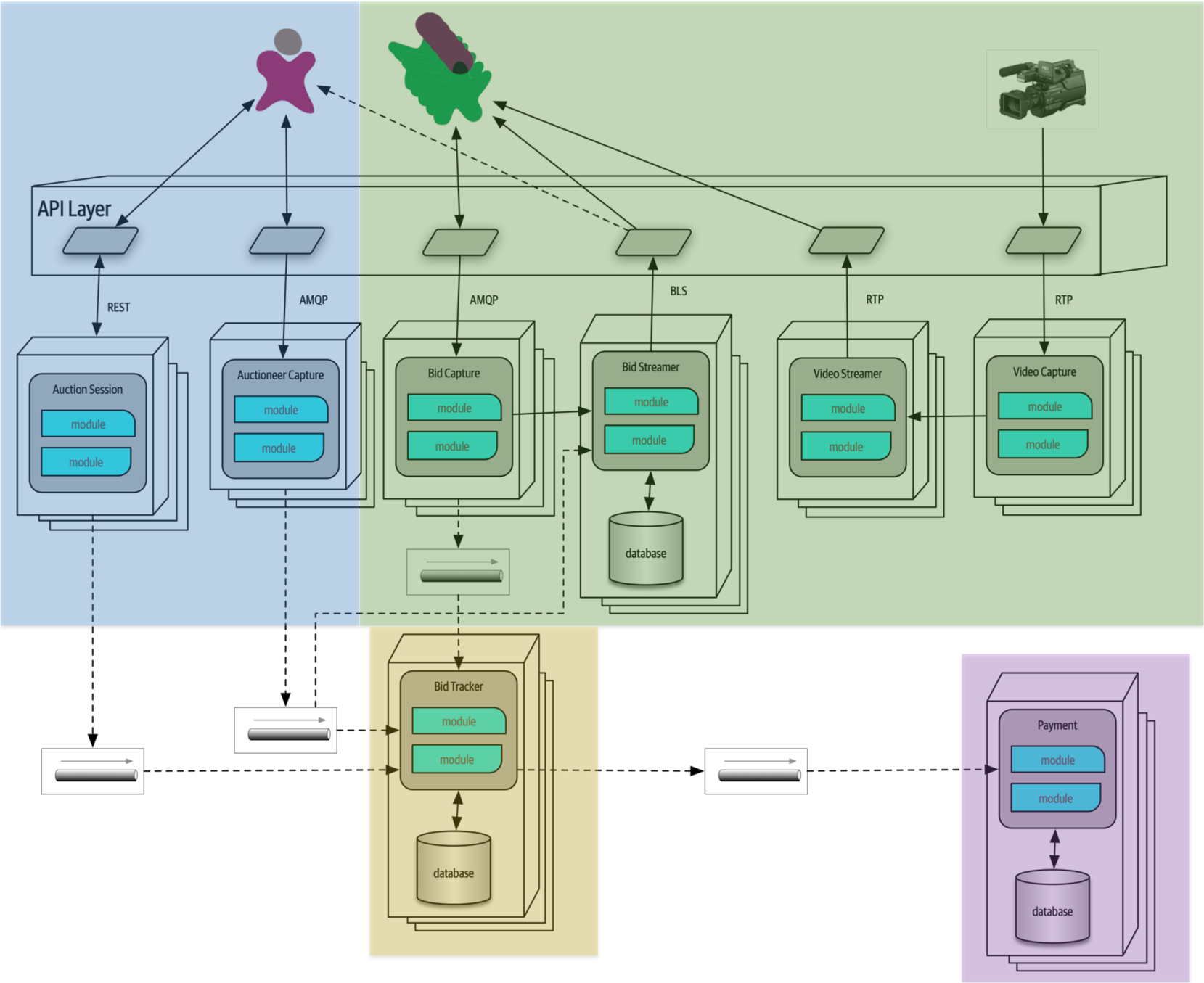
An auction company wants to take their auctions online to a nationwide scale--customers choose the auction to participate in, wait until the auction begins, then bid during the live auction as if they were there in the room, with the auctioneer.

- **Users**: scale up to hundreds of participants (per auction), potentially up to thousands of participants, and as many simultaneous auctions as possible
- **Requirements**:
  - bidders can see a live video stream of the auction and see all bids as they occur
  - auctions must be as real-time as possible
  - both online and live bids must be received in the order in which they are placed
  - bidders register with credit card; system automatically charges card if bidder wins
  - participants must be tracked via a reputation index
- **Additional Context:**
  - auction company is expanding aggressively by merging with smaller competitors
  - if nationwide auction is a success, replicate the model overseas
  - budget is not constrained--this is a strategic direction
  - company just exited a lawsuit where they settled a suit alleging fraud

Your Architectural Kata is...

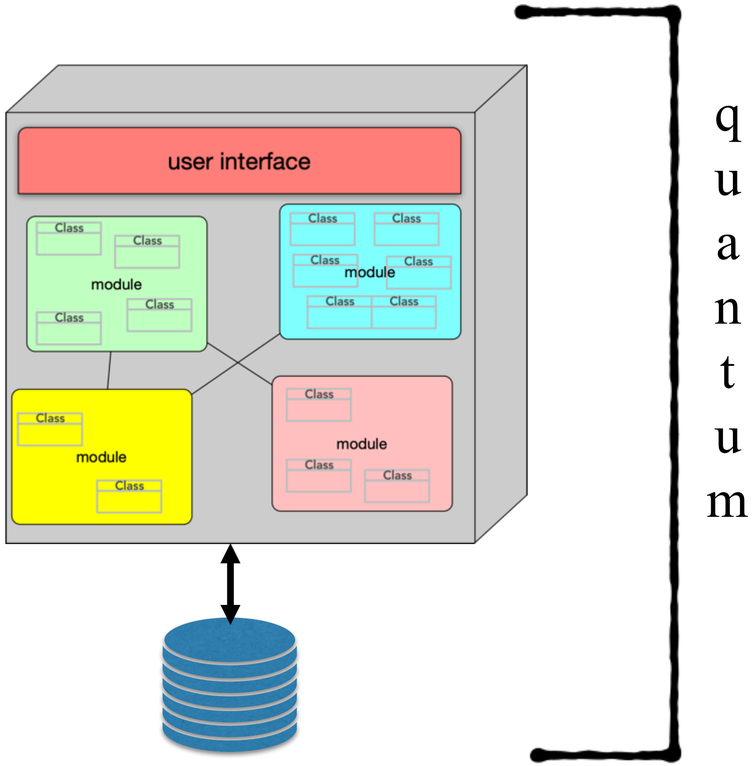
Going Going Gone!

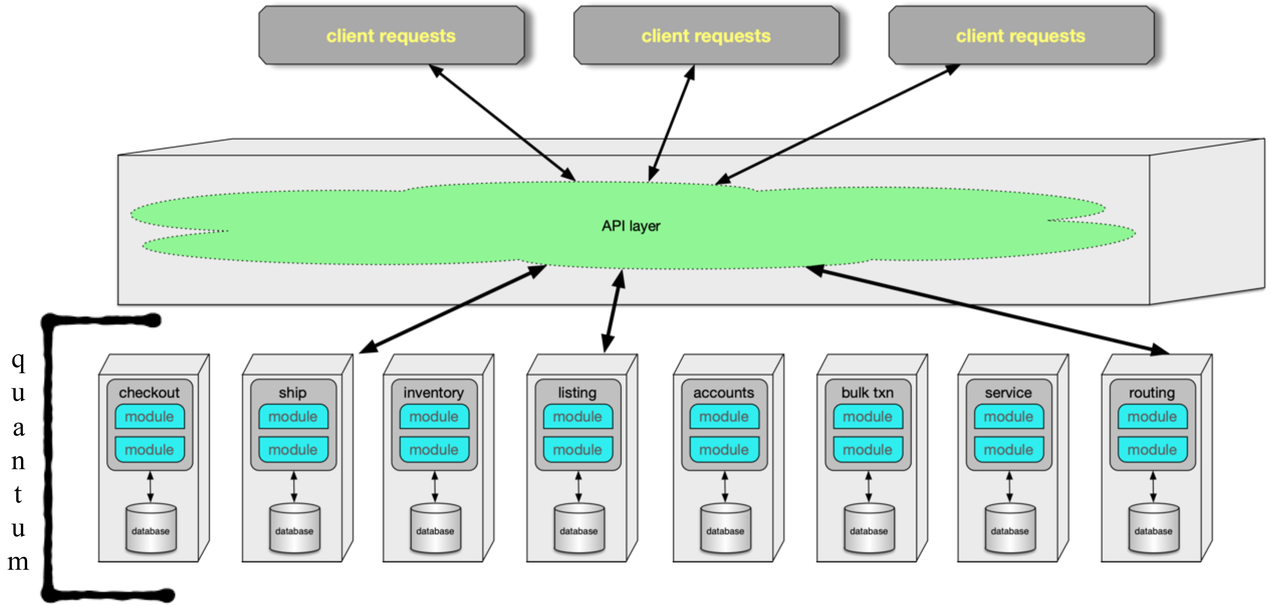



API Layer
REST
AMQP
AMQP
BLS
RTP
RTP
Auction Session
module
module
Auctioneer Capture
module
module
Bid Capture
module
module
Bid Streamer
module
module
database
Video Streamer
module
module
Video Capture
module
module
Bid Tracker
module
module
database
Payment
module
module
database

Why Quantum?

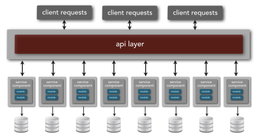
The quantum is where architectural characteristics live.



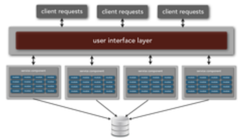


Architectural Patterns

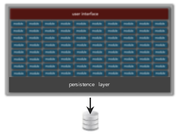
Architectural Patterns



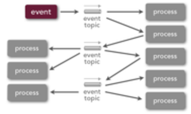
microservices architecture



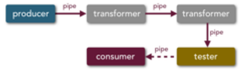
service-based architecture



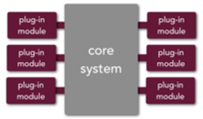
layered architecture



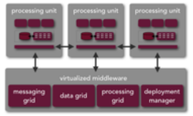
event-driven architecture



pipeline architecture

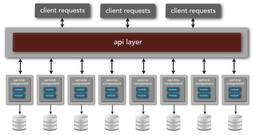


microkernel architecture

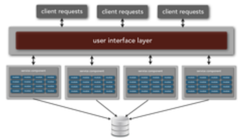


space-based architecture

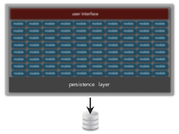
Architectural Patterns



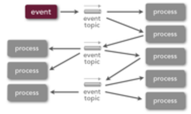
microservices architecture



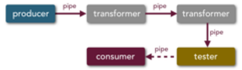
service-based architecture



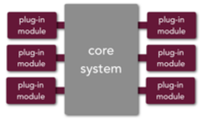
layered architecture



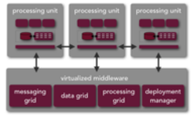
event-driven architecture



pipeline architecture



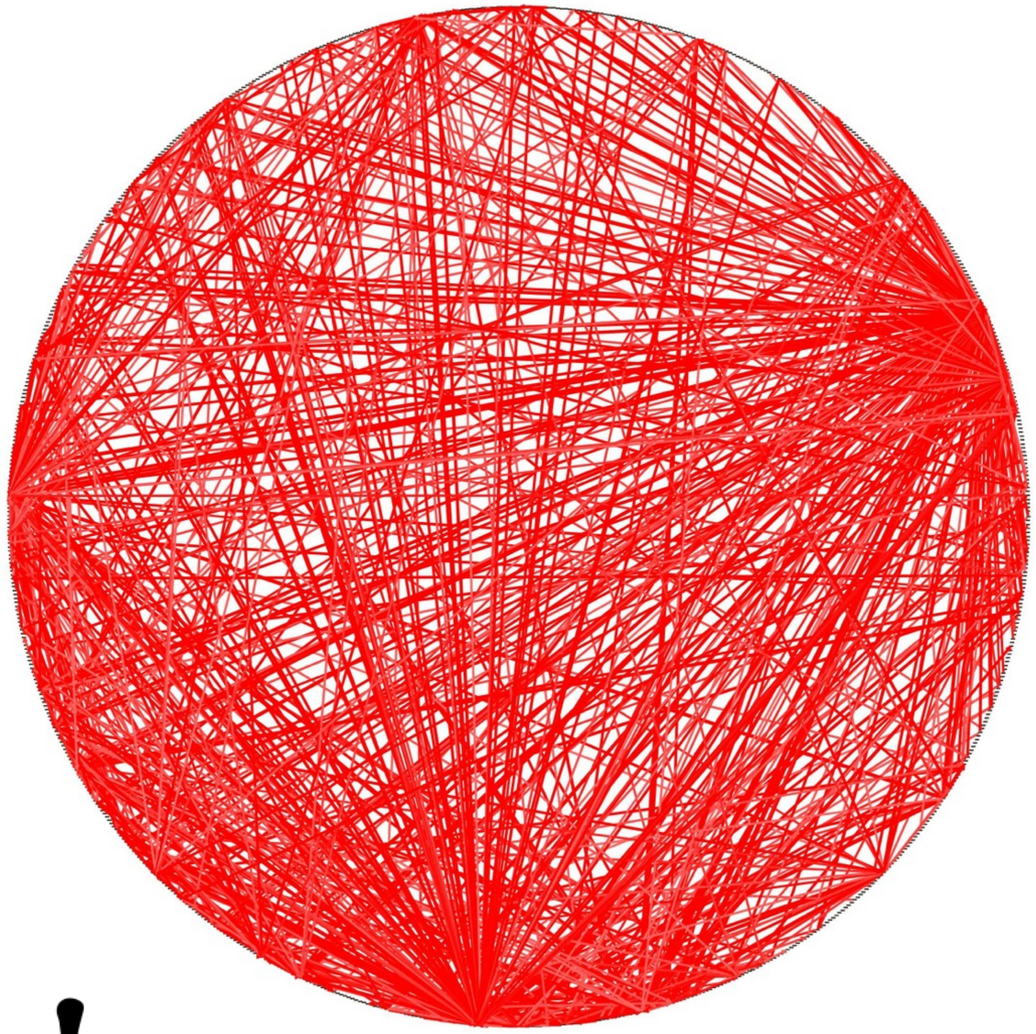
microkernel architecture



space-based architecture

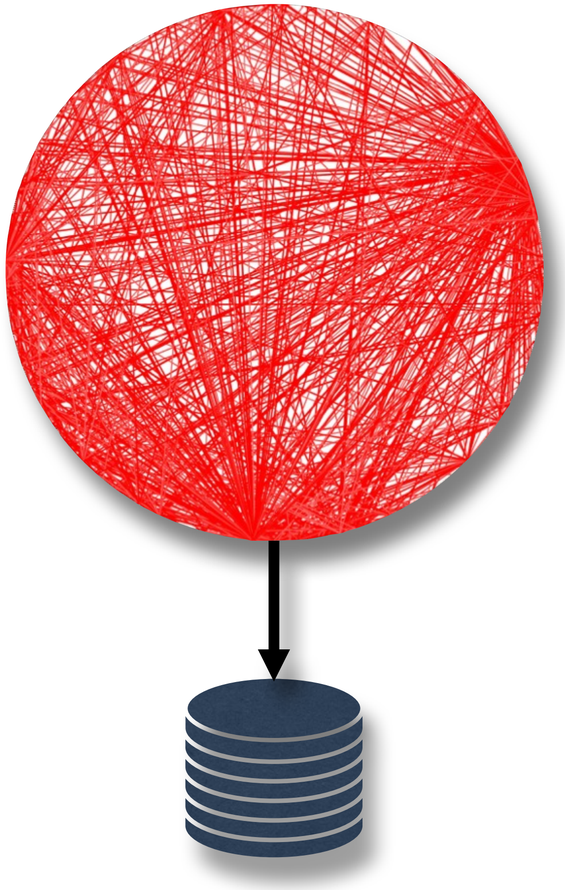
Patterns encapsulate a well-known set of architectural characteristics.

Quantum: Large to Small

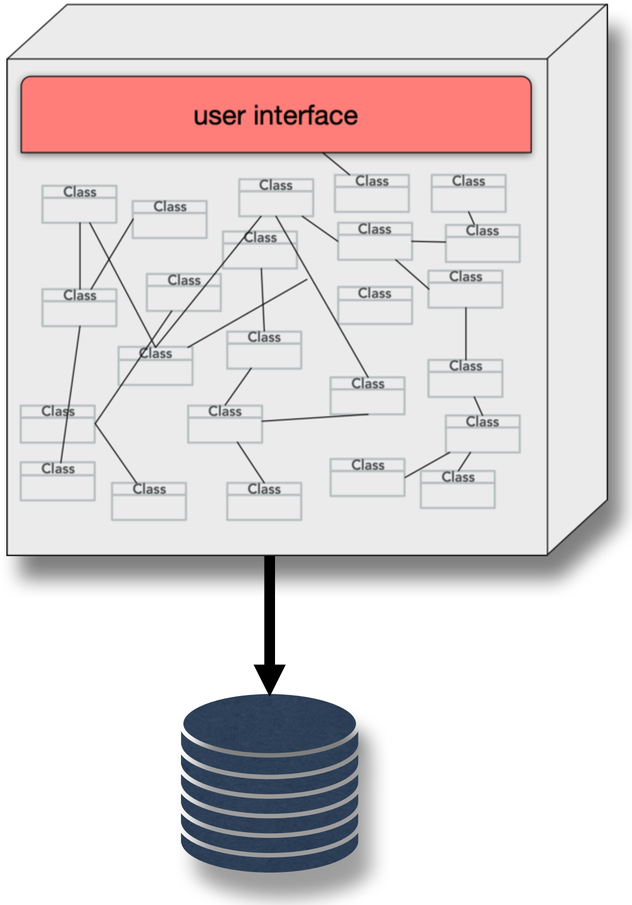


Monoliths

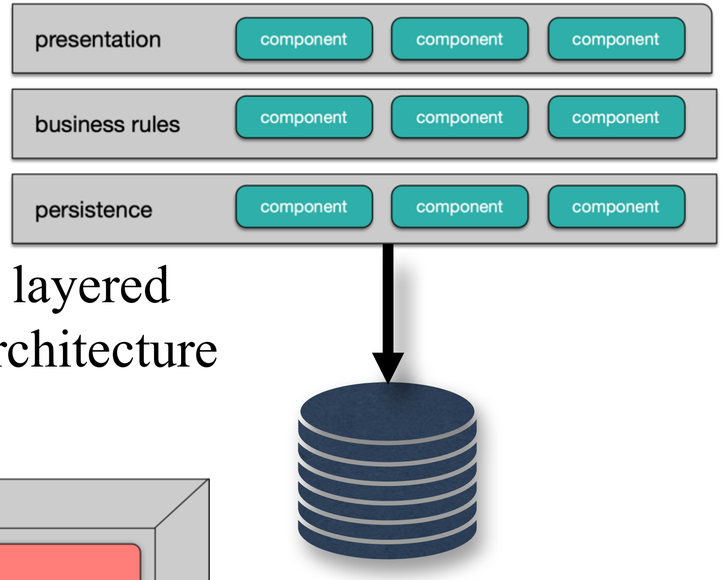
Monoliths



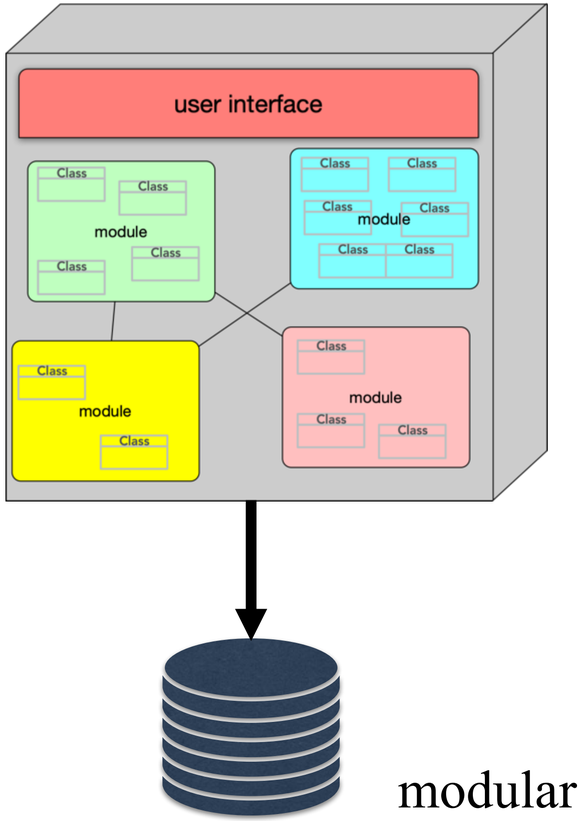
"Big Ball of Mud"
https://en.wikipedia.org/wiki/Big_ball_of_mud



unstructured monolith



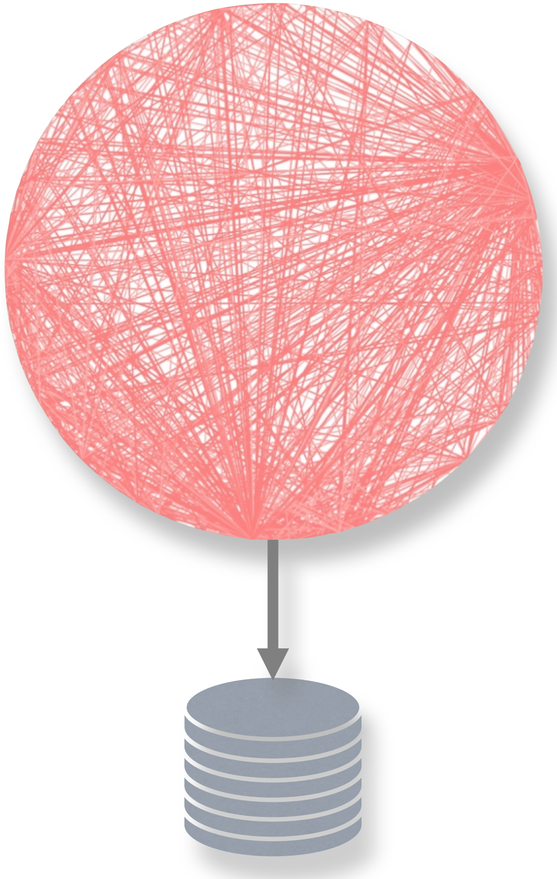
layered architecture



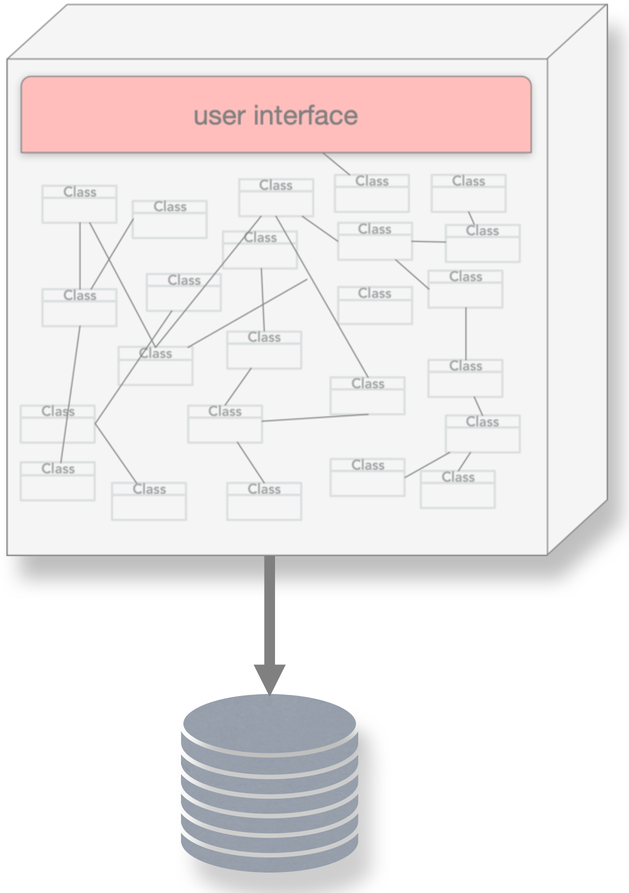
modular monolith

http://www.codingthearchitecture.com/presentations/sa2015-modular-monoliths

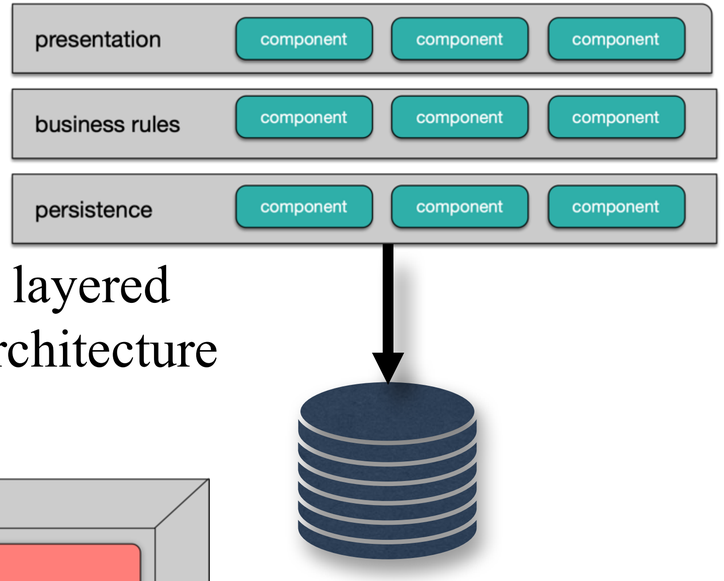
Monoliths



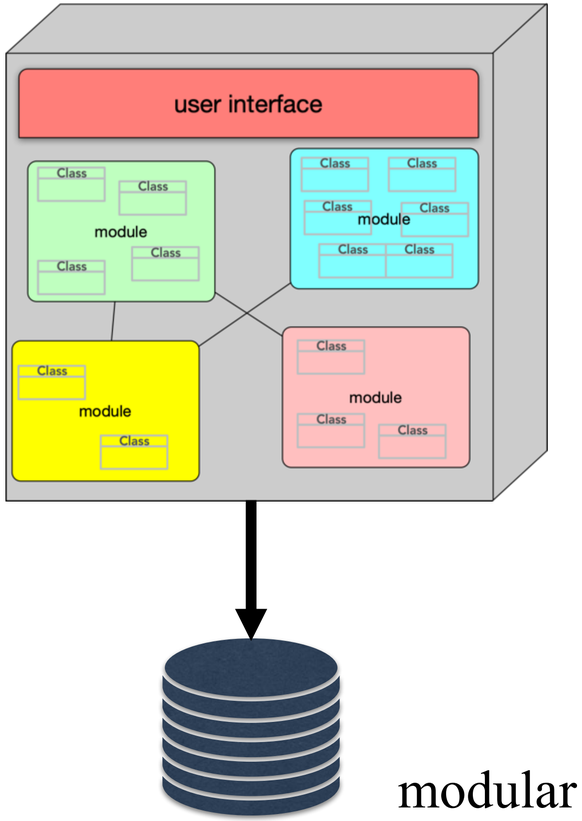
"Big Ball of Mud"
https://en.wikipedia.org/wiki/Big_ball_of_mud



unstructured monolith

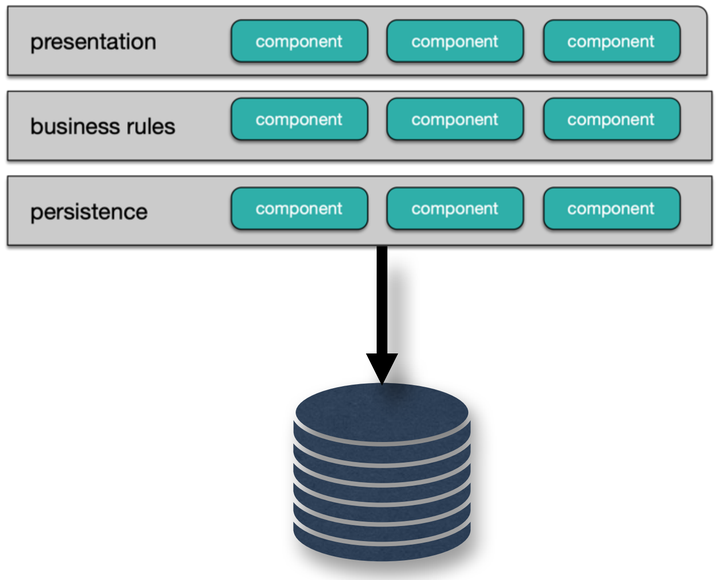


layered architecture

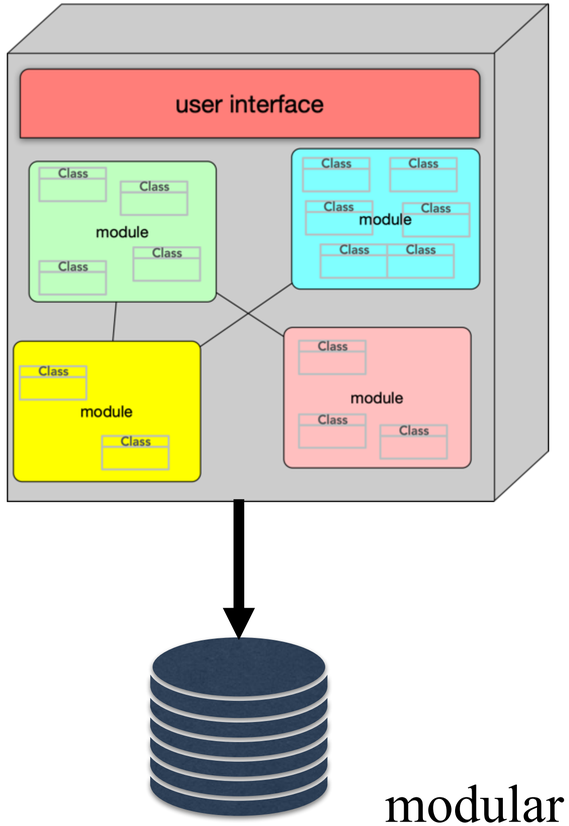


modular monolith
http://www.codingthearchitecture.com/presentations/sa2015-modular-monoliths

Monoliths



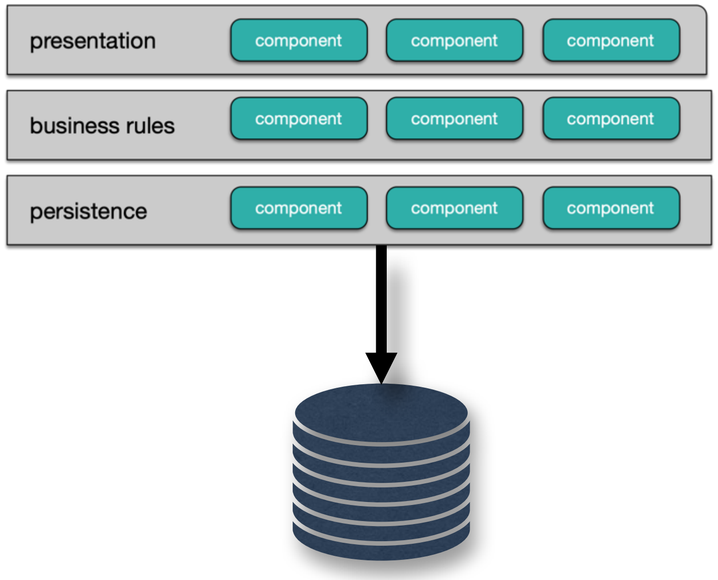
layered architecture



modular monolith

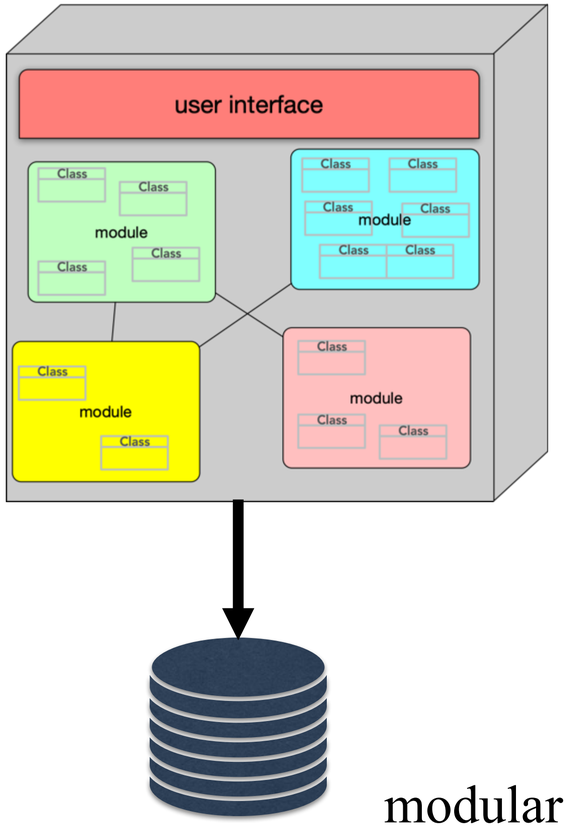
Monoliths

technical partitioning

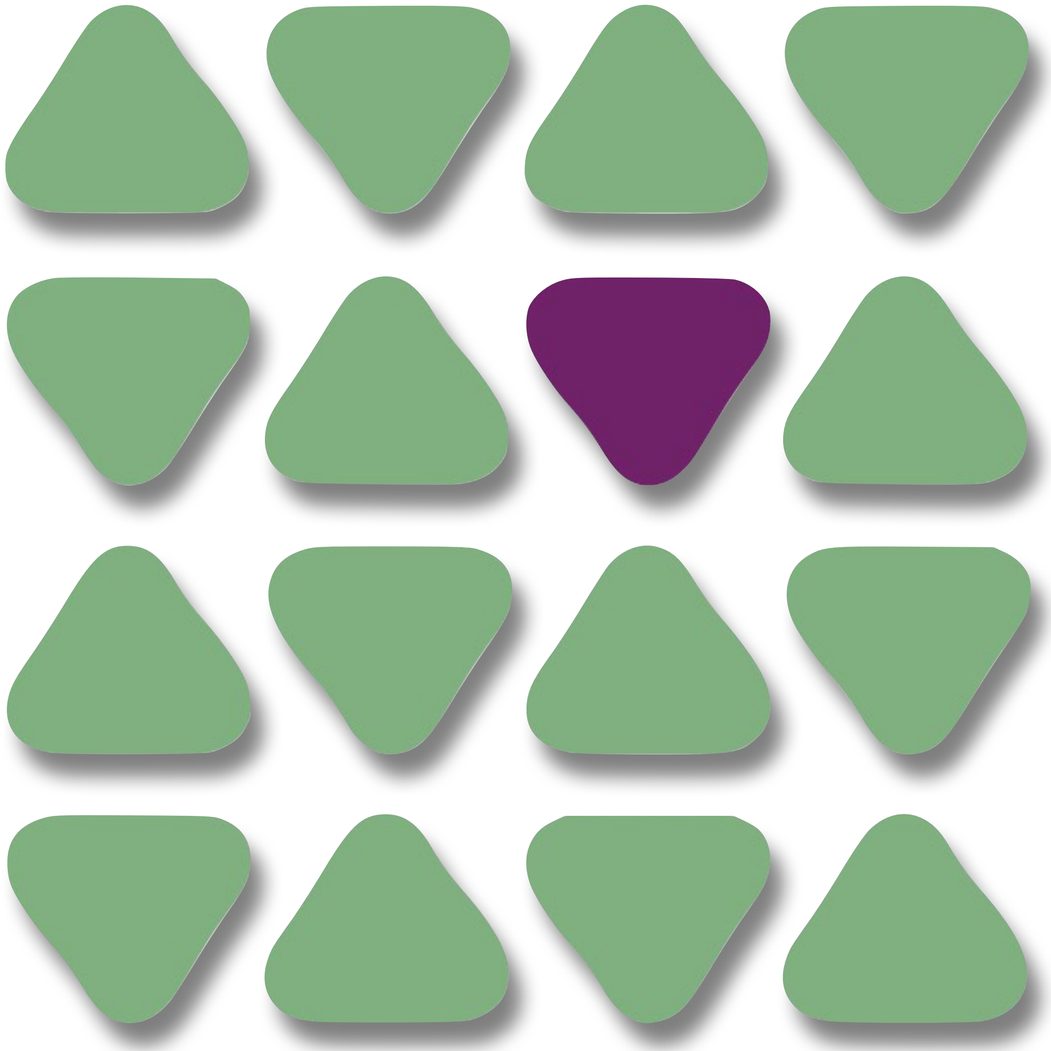


layered architecture

domain partitioning

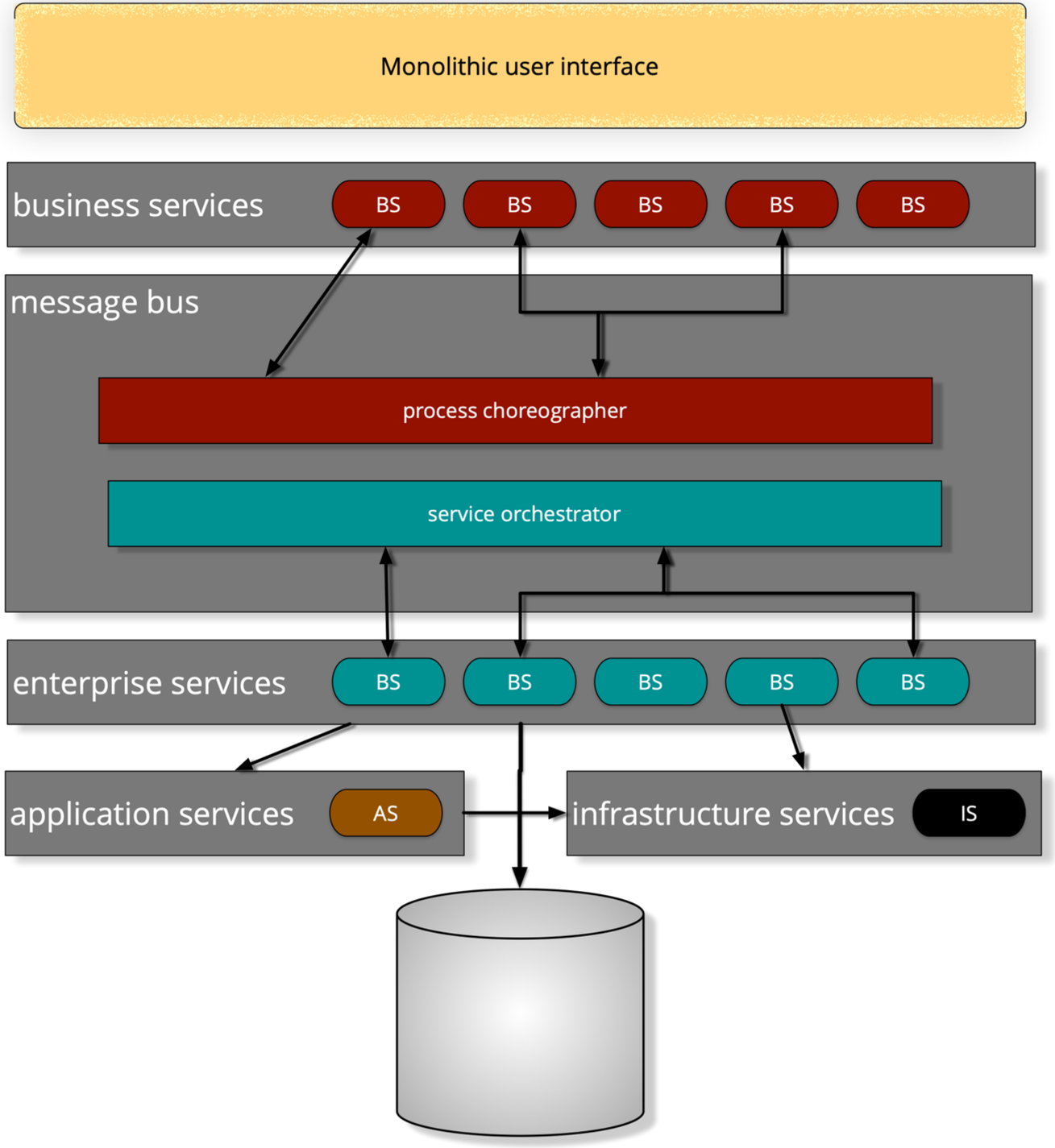


modular monolith

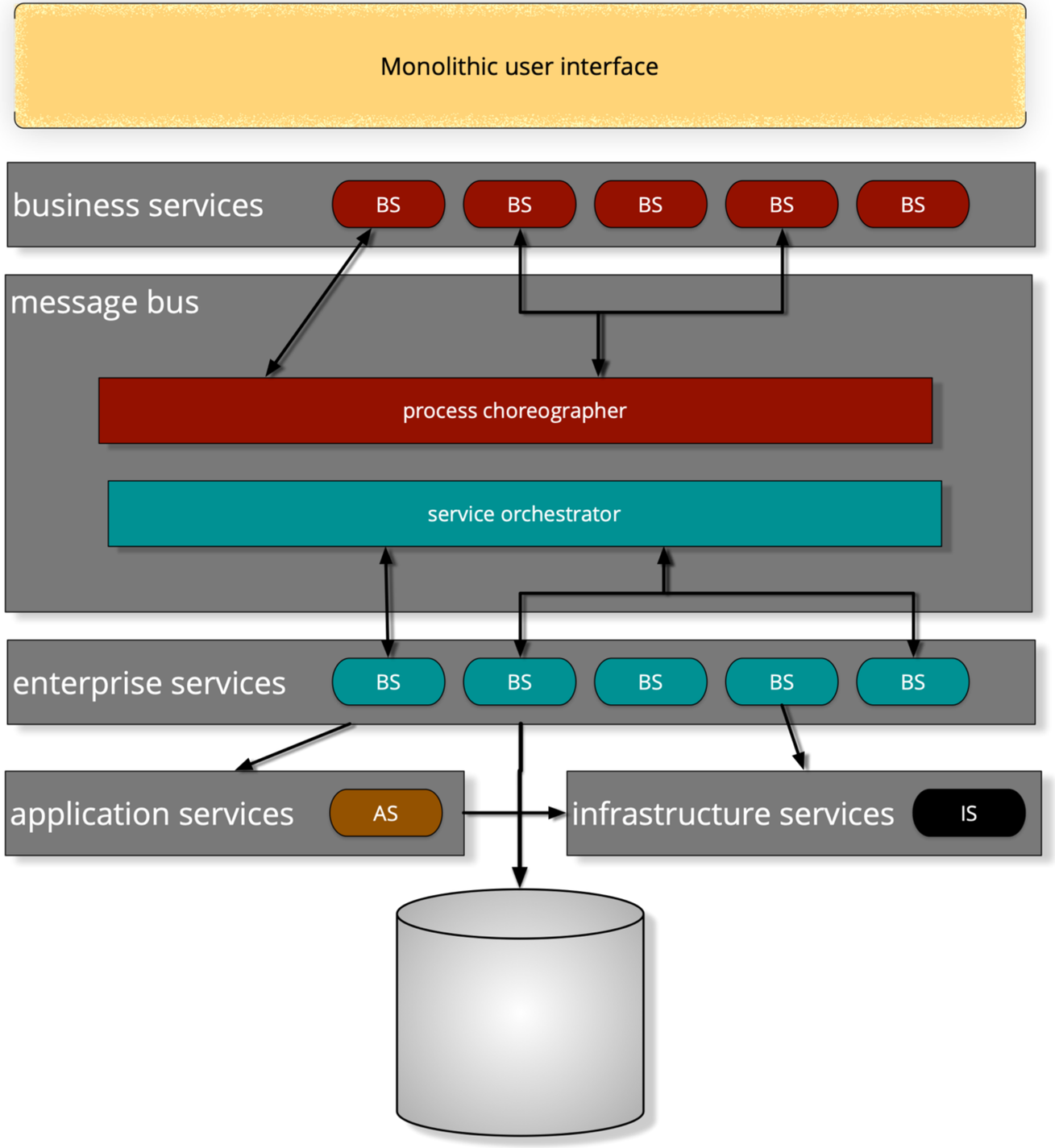


distributed systems

ESB-driven SOA

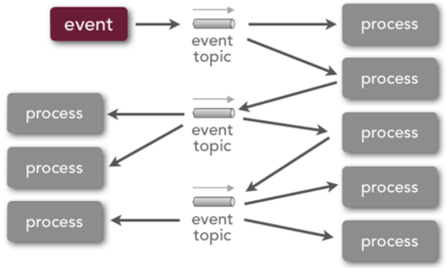


ESB-driven SOA

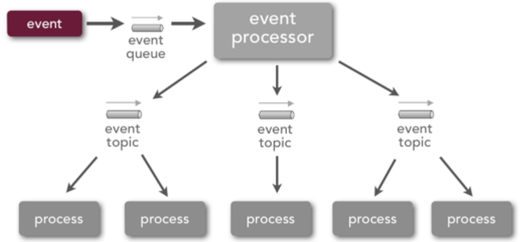


technical partitioning

Event-driven Architectures

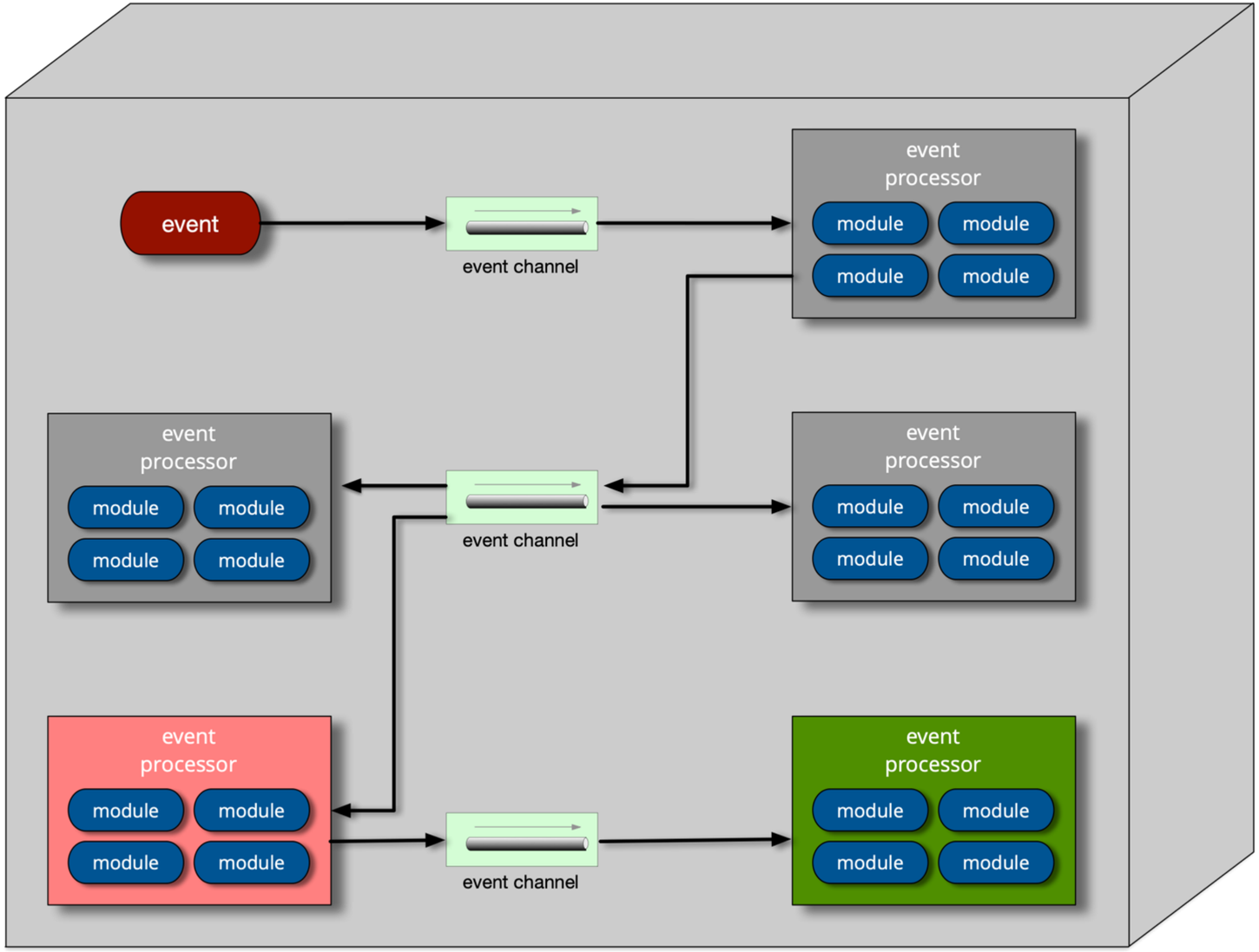


broker topology

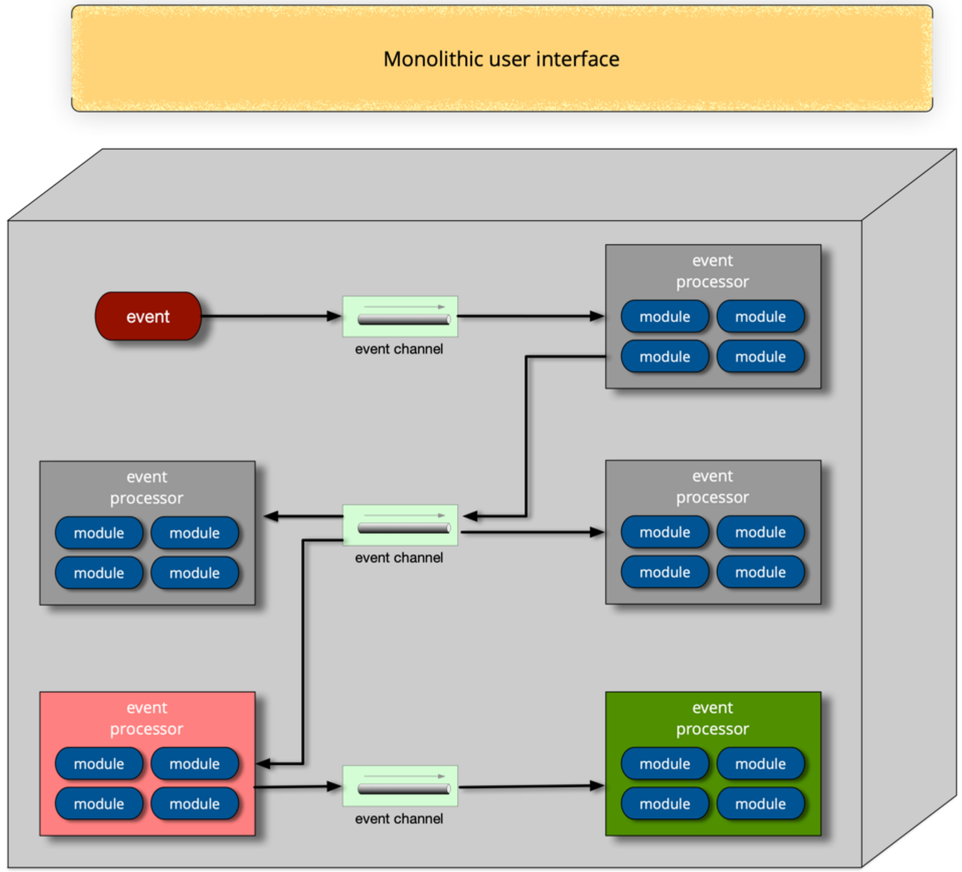


mediator topology

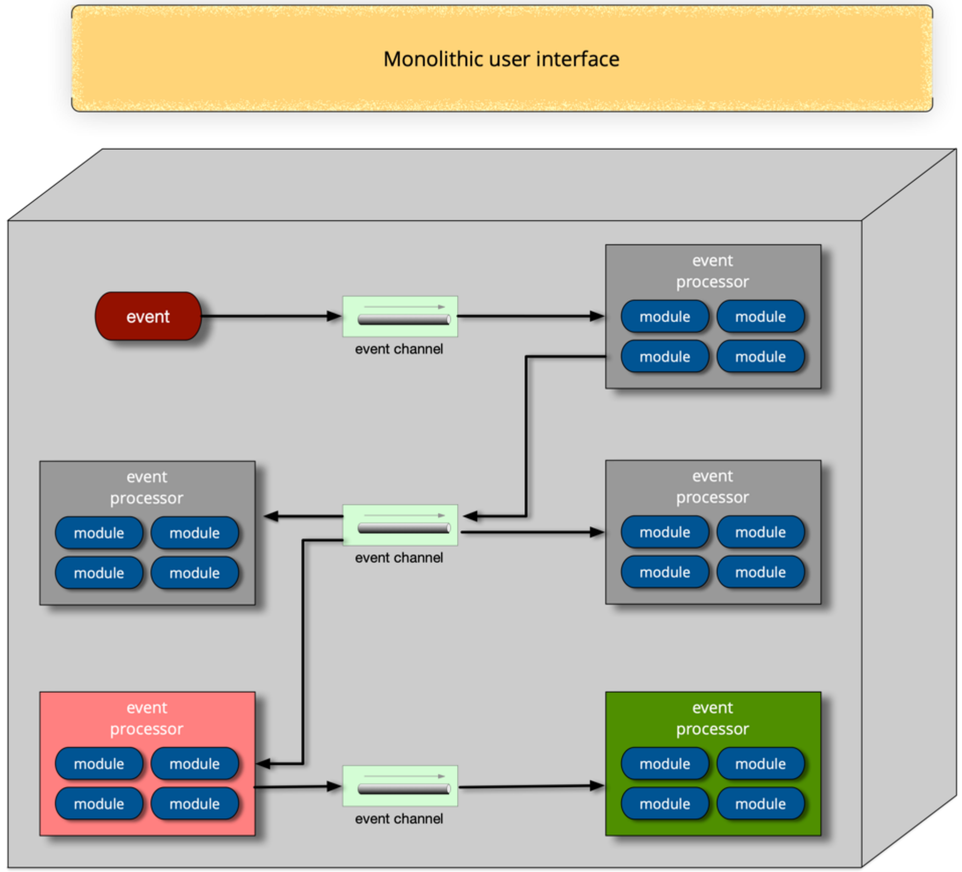
Broker Base Architecture

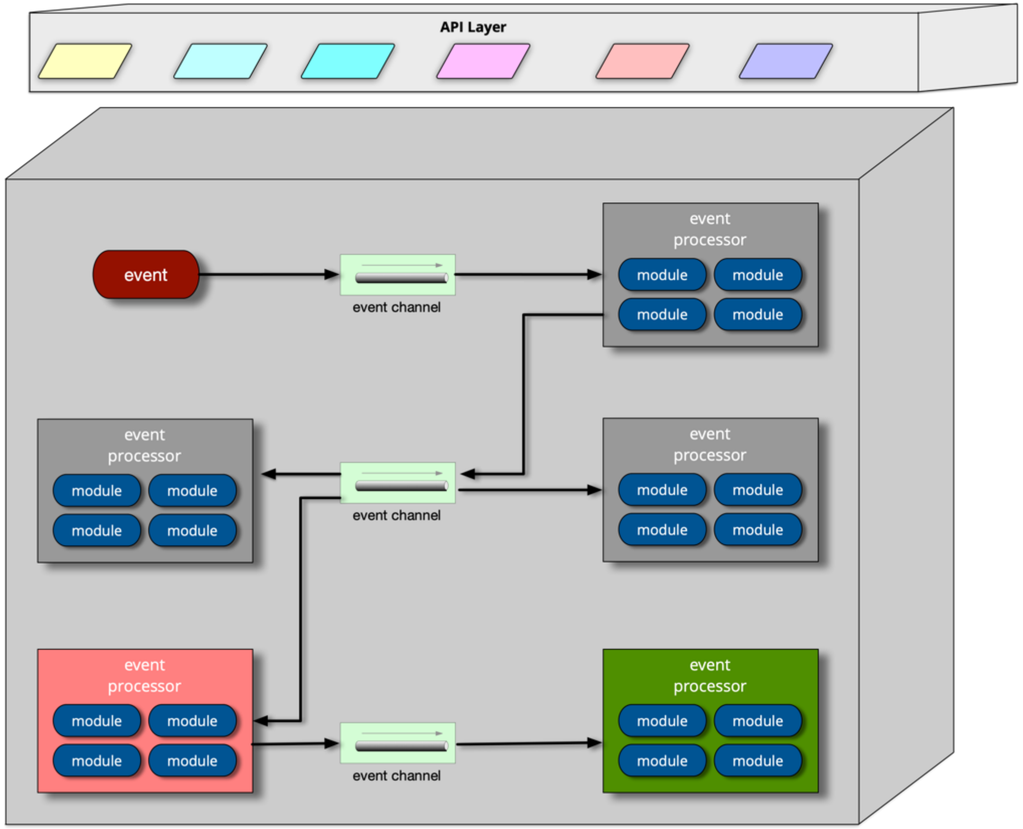


Broker: UI

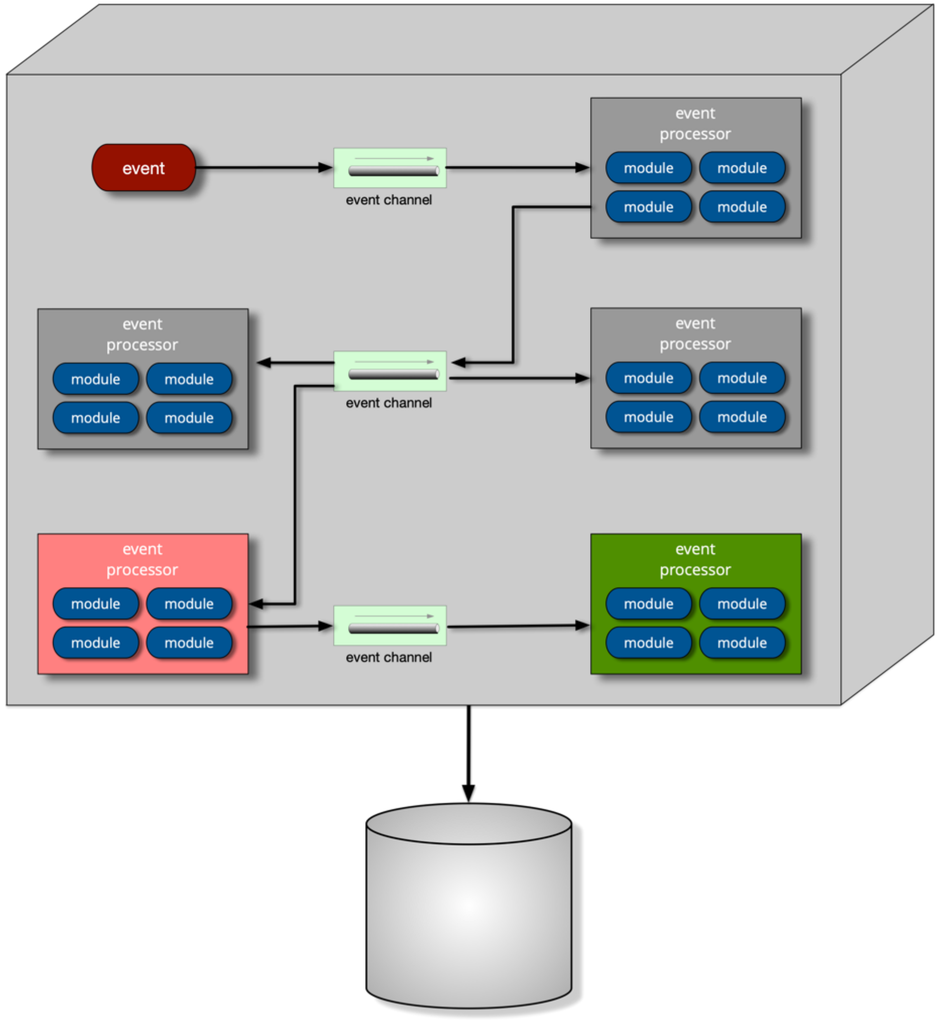

Monolithic user interface
event
event channel
event processor
module
module
module
module
event processor
module
module
module
module
event channel
event processor
module
module
module
module
event processor
module
module
module
module
event channel
event processor
module
module
module
module

Broker: UI

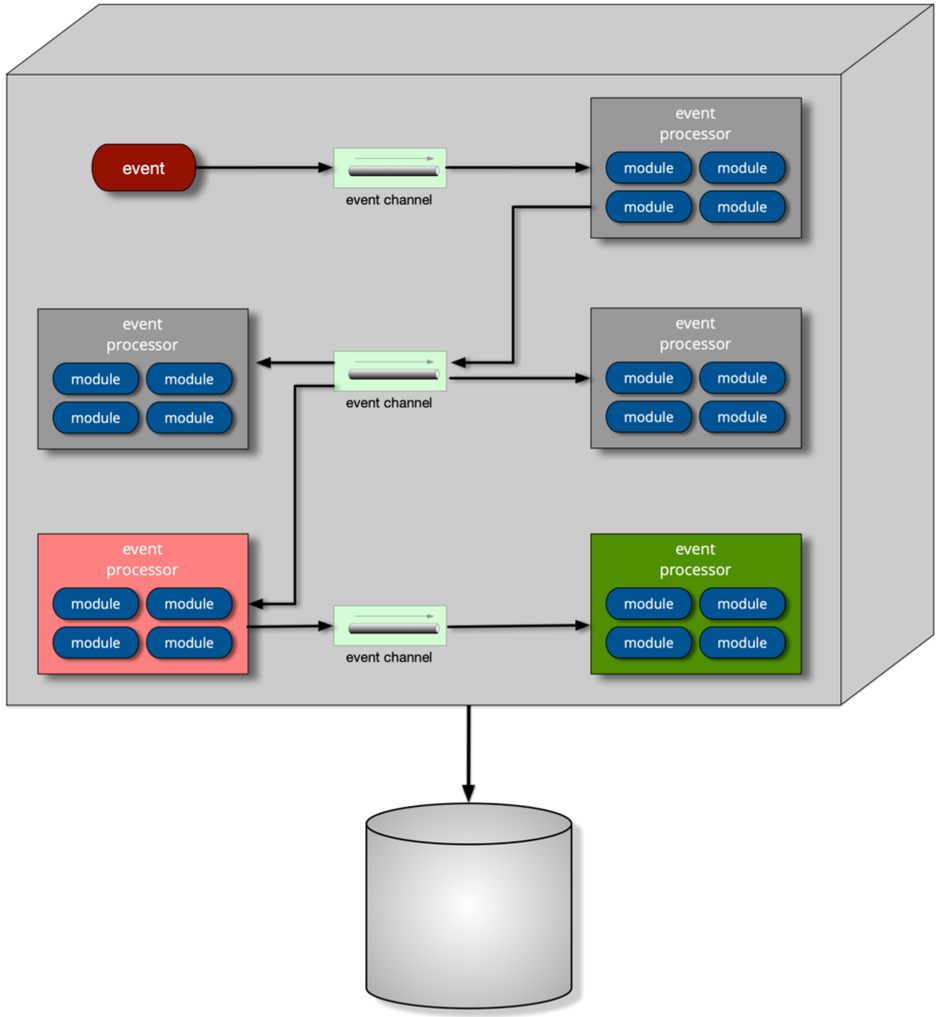

Monolithic user interface
event
event channel
event processor
module
module
module
module
event processor
module
module
module
module
event channel
event processor
module
module
module
module
event processor
module
module
module
module
event channel
event processor
module
module
module
module

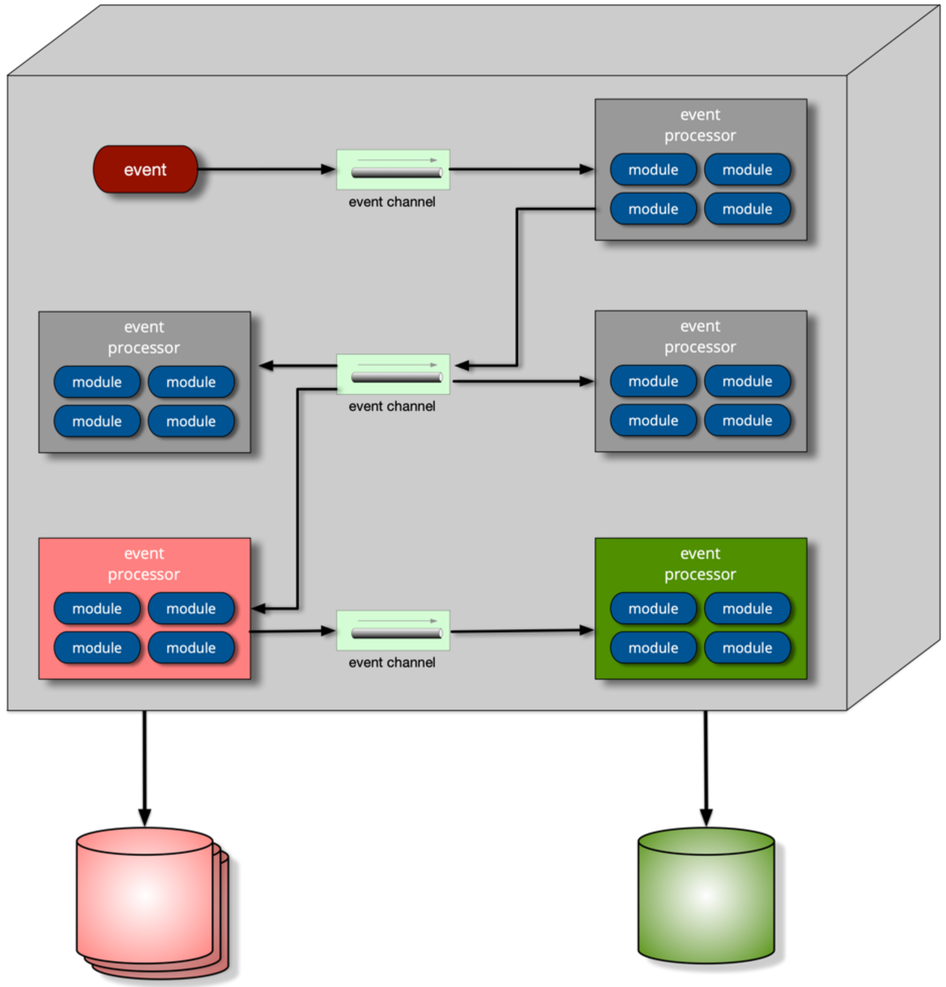

API Layer
event
event channel
event processor
module
module
module
module
event processor
module
module
module
module
event channel
event processor
module
module
module
module
event processor
module
module
module
module
event channel
event processor
module
module
module
module

Broker: Data

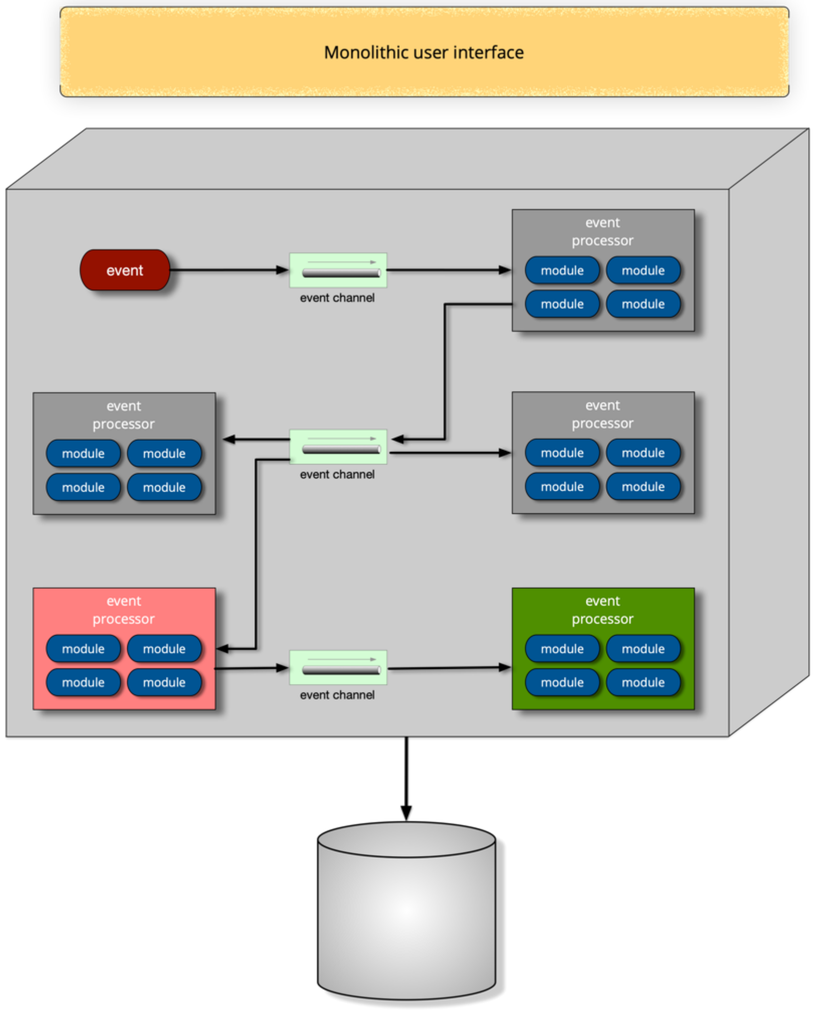

event
event channel
event processor
module
module
module
module
event processor
module
module
module
module
event channel
event processor
module
module
module
module
event processor
module
module
module
module
event channel
event processor
module
module
module
module
event channel

Broker: Data

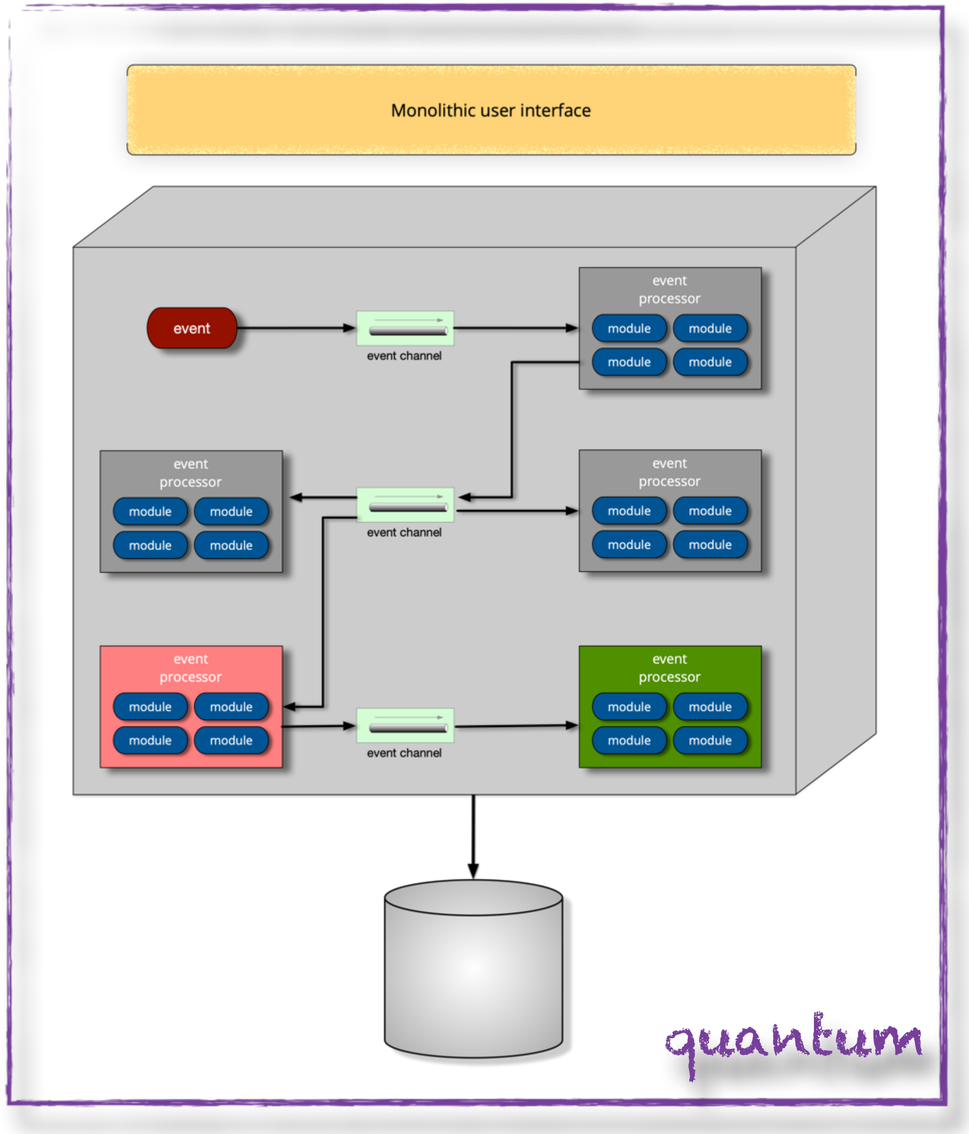

event
event channel
event processor
module
module
module
module
event channel
event processor
module
module
module
module
event channel
event processor
module
module
module
module
event channel
event processor
module
module
module
module


event
event channel
event processor
module
module
module
module
event channel
event processor
module
module
module
module
event channel
event processor
module
module
module
module
event channel
event processor
module
module
module
module

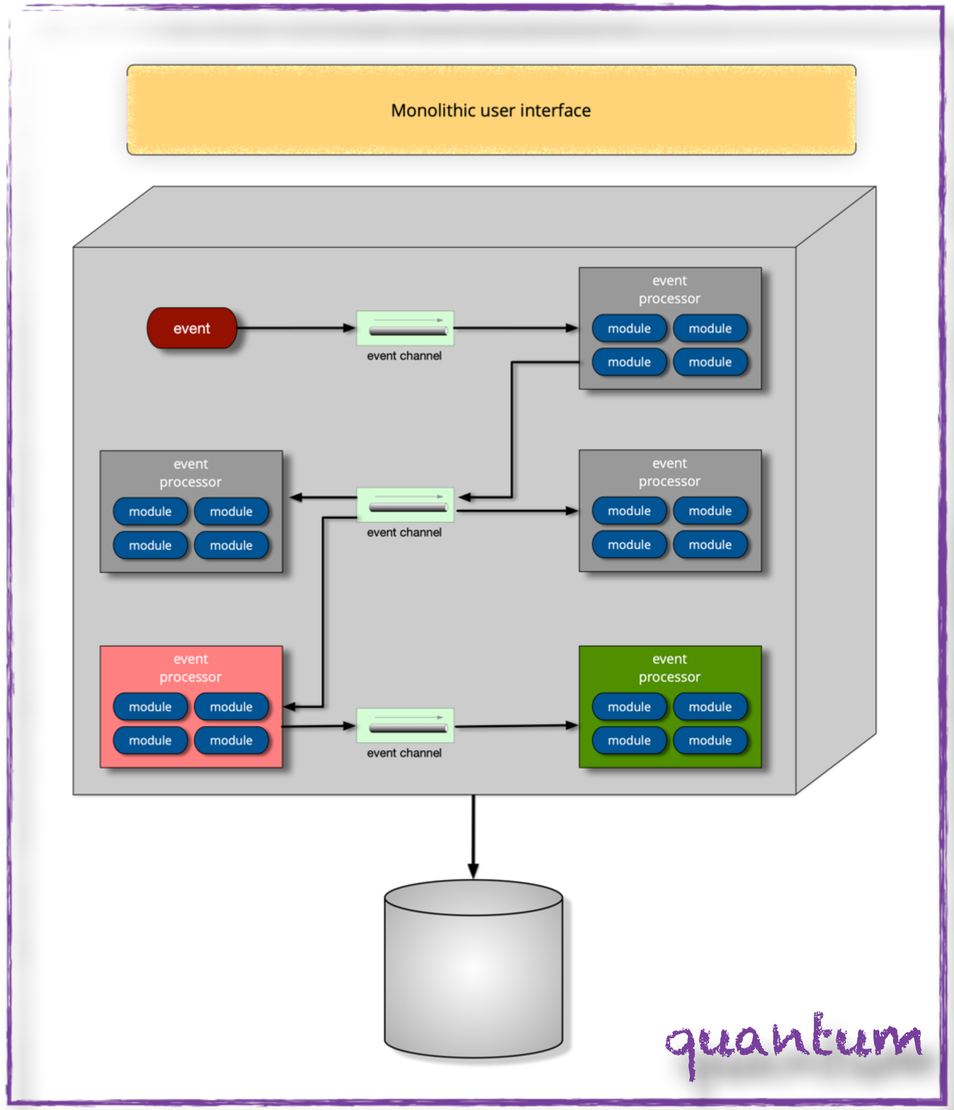
Broker: Quanta

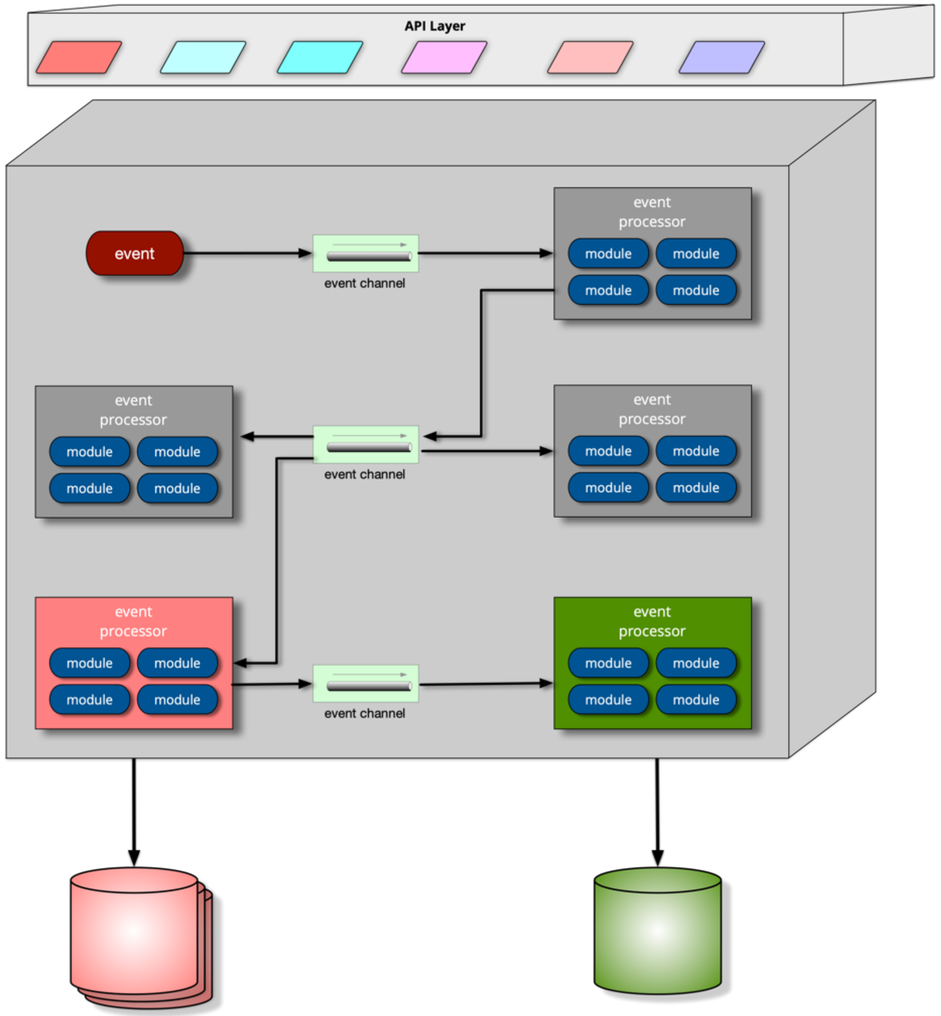


Broker: Quanta

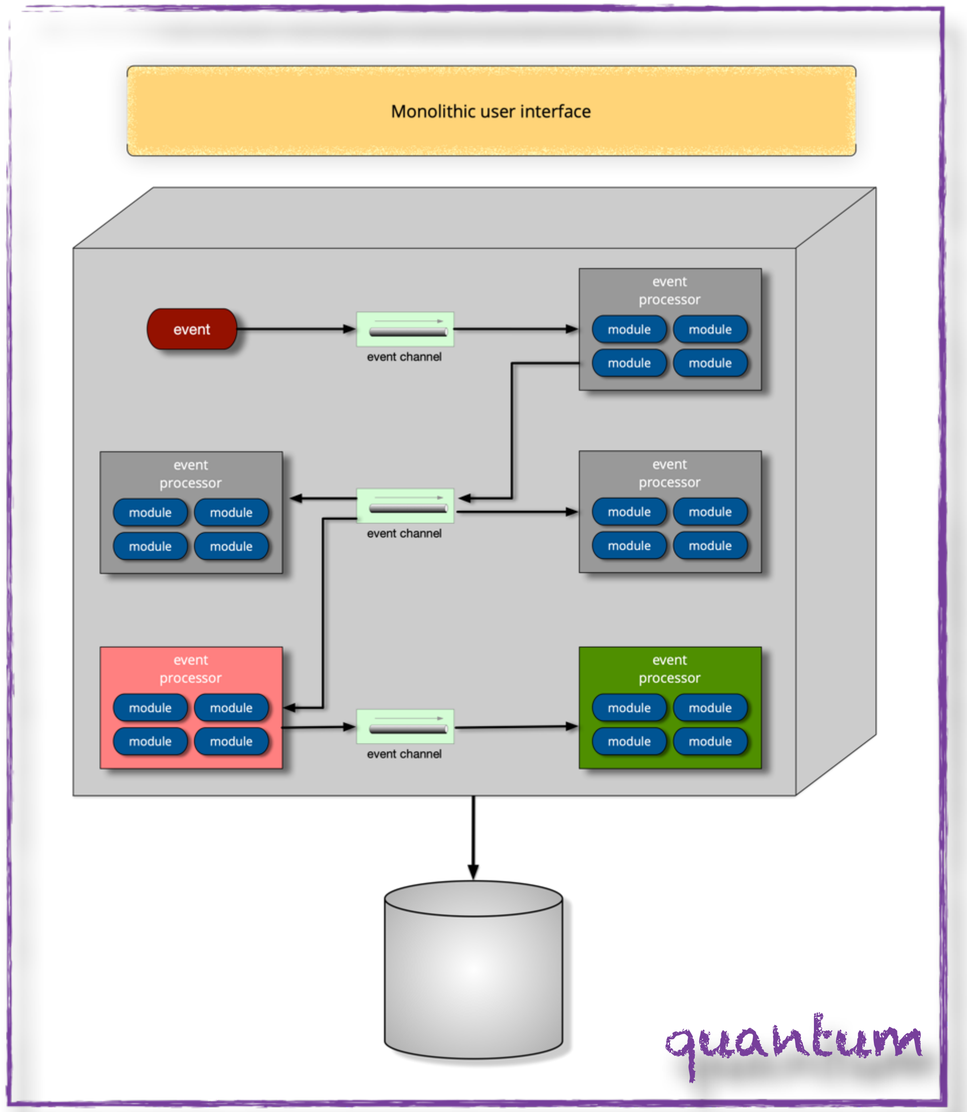


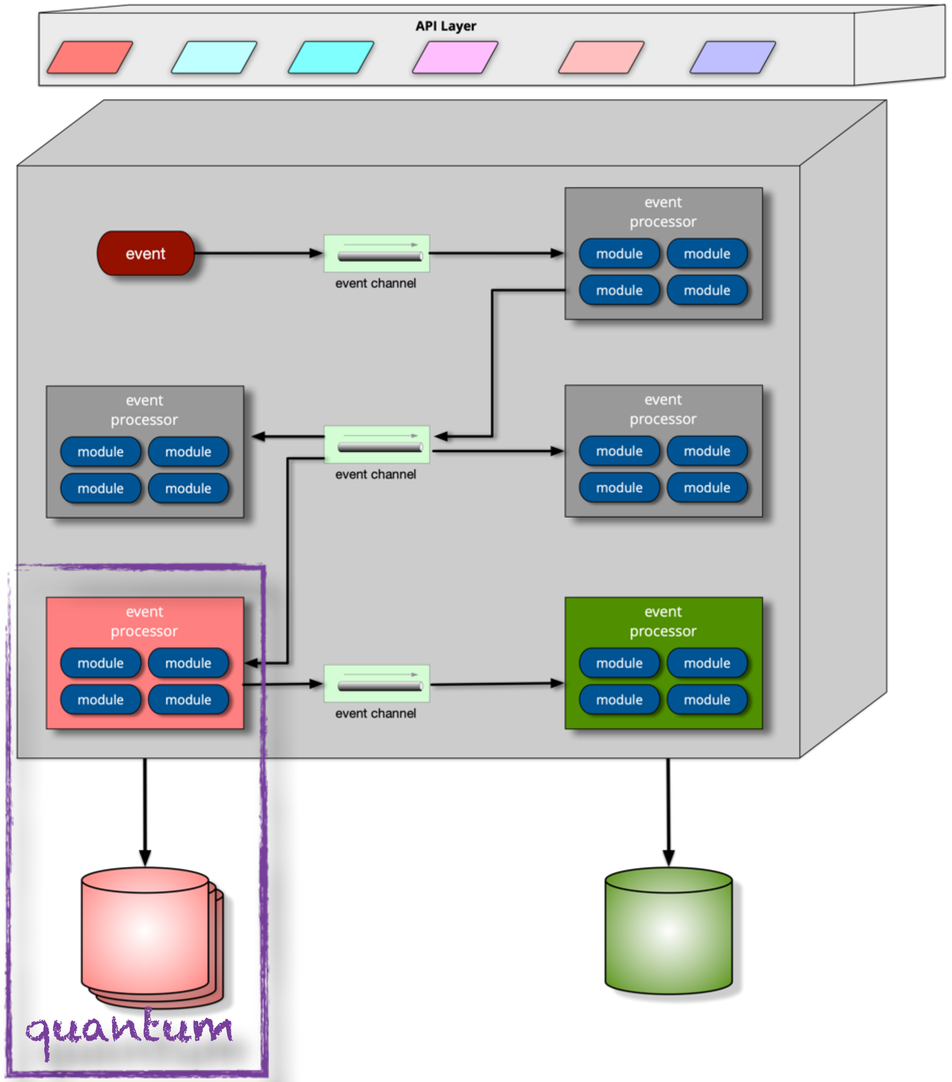
Broker: Quanta



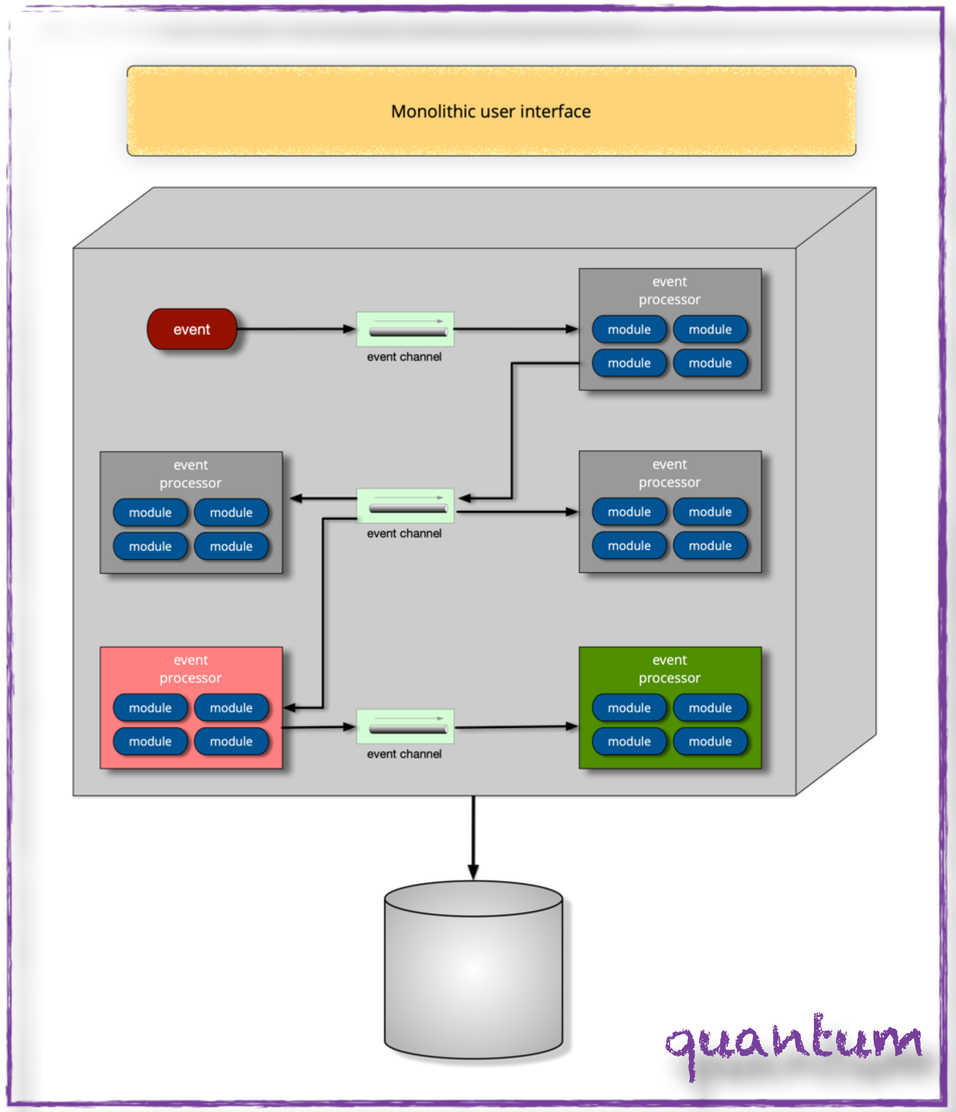


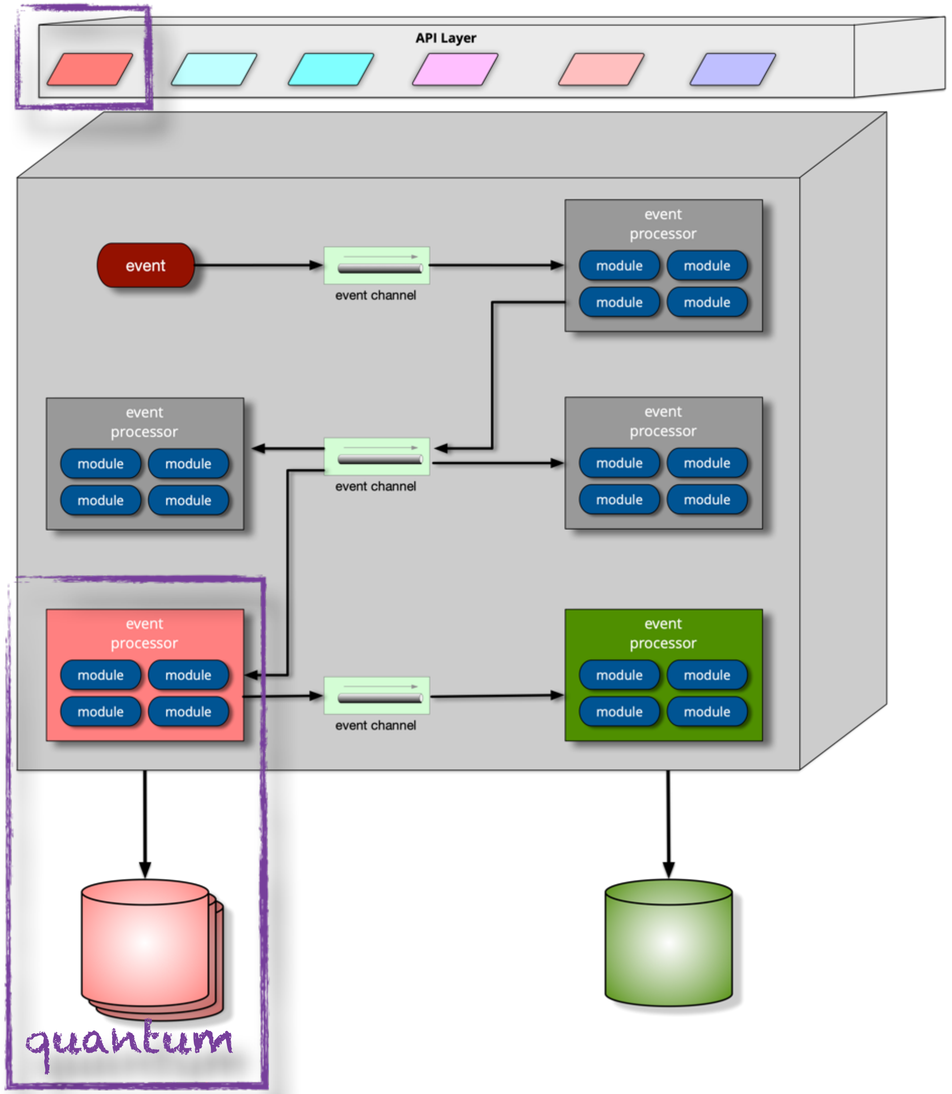
Broker: Quanta



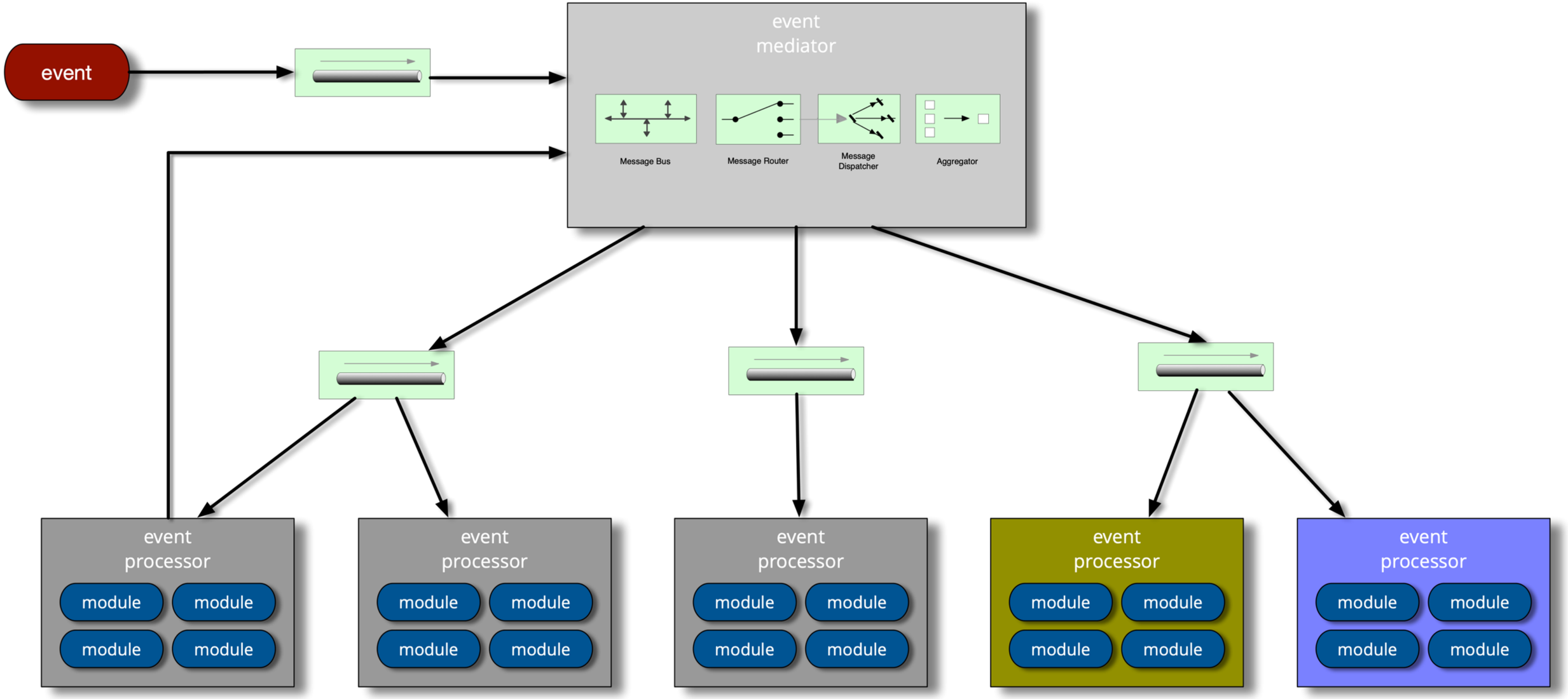


Broker: Quanta

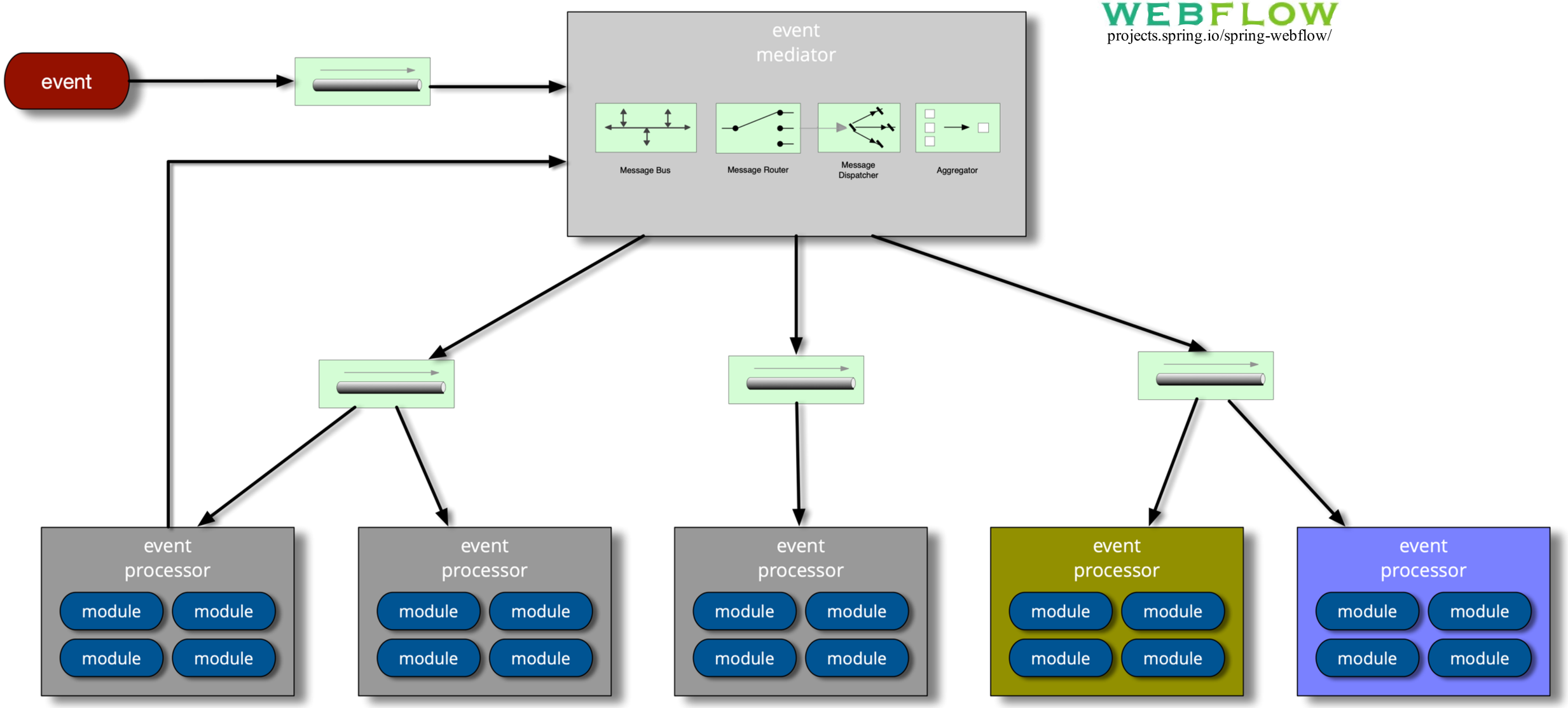




Meditator EDA

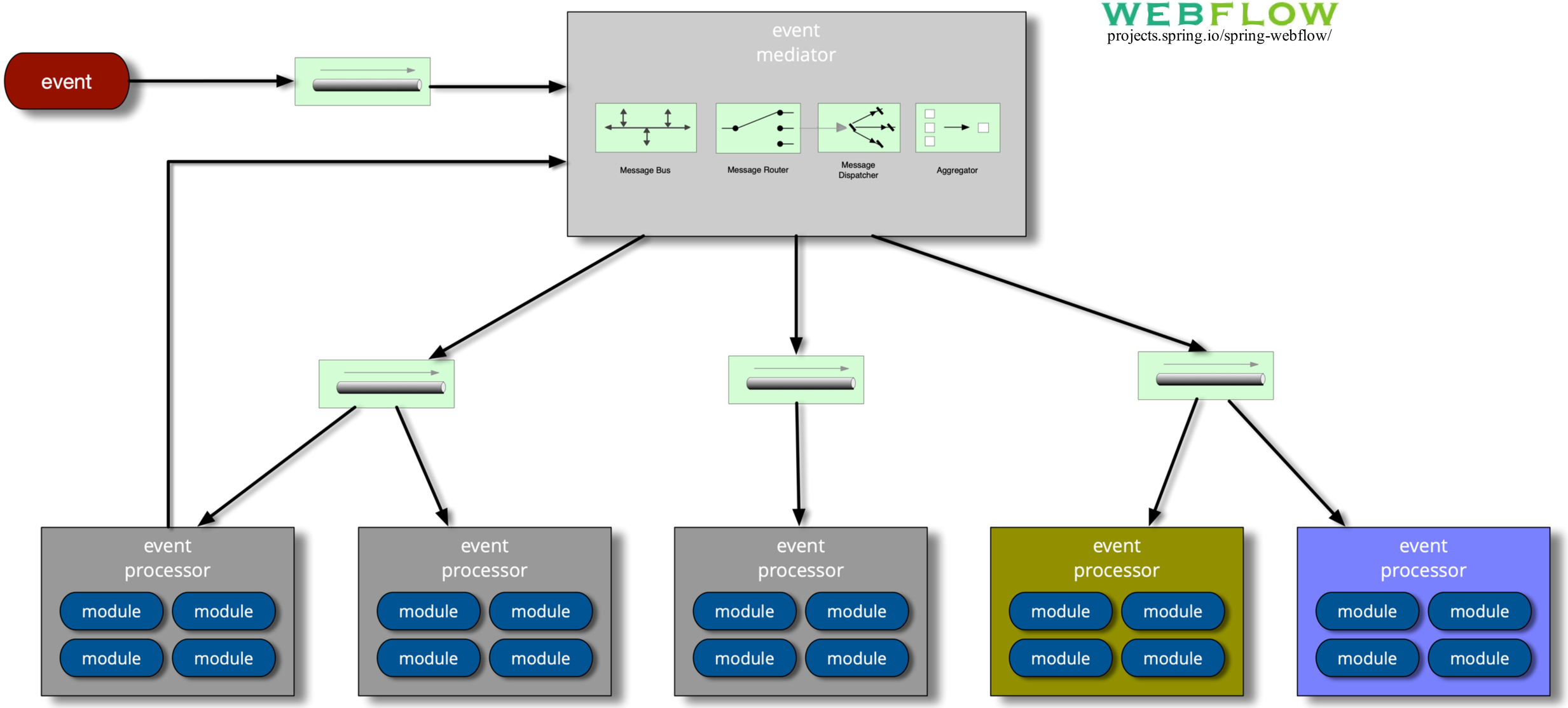


Meditator EDA



Spring WEBFLOW
projects.spring.io/spring-webflow/

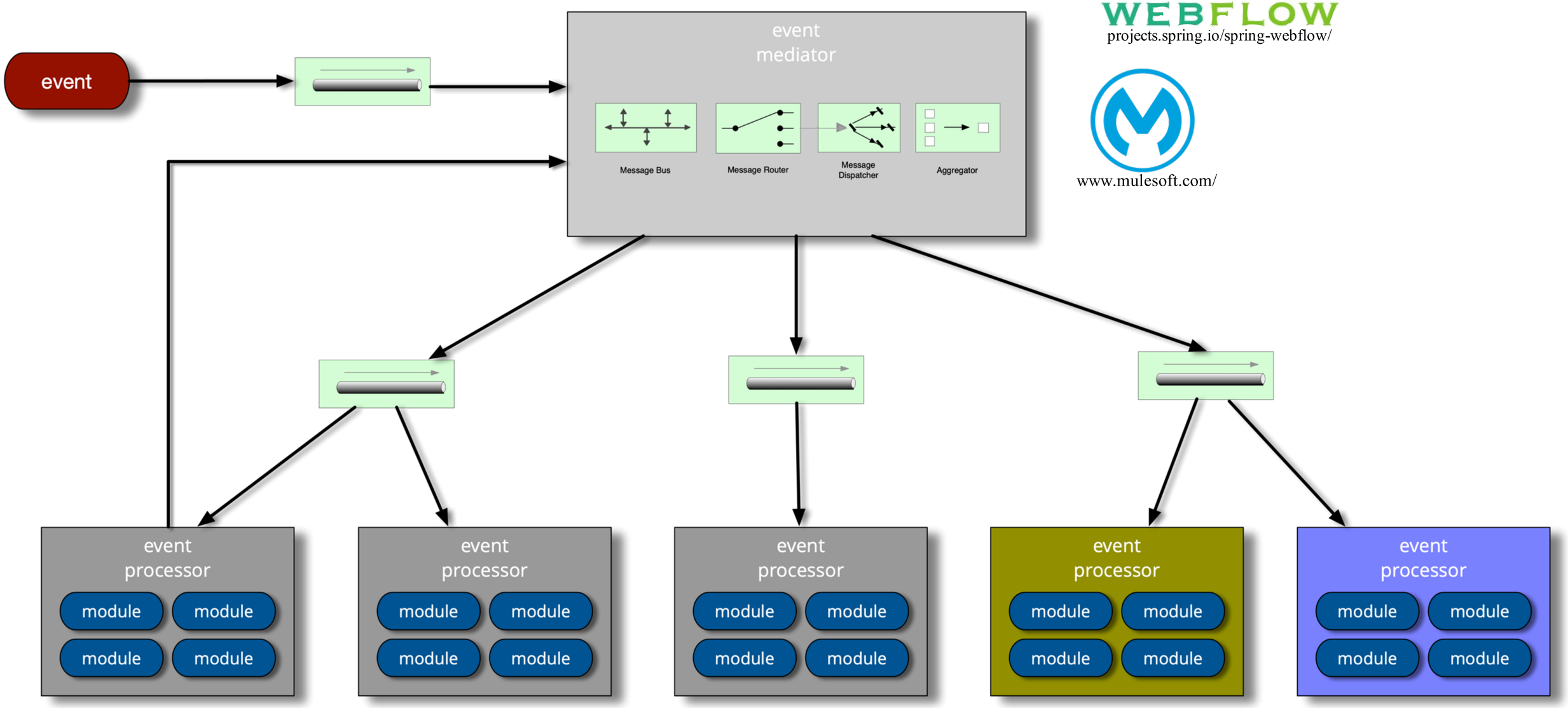
Meditator EDA



WEBFLOW
projects.spring.io/spring-webflow/

APACHE Camel
camel.apache.org/

Meditator EDA

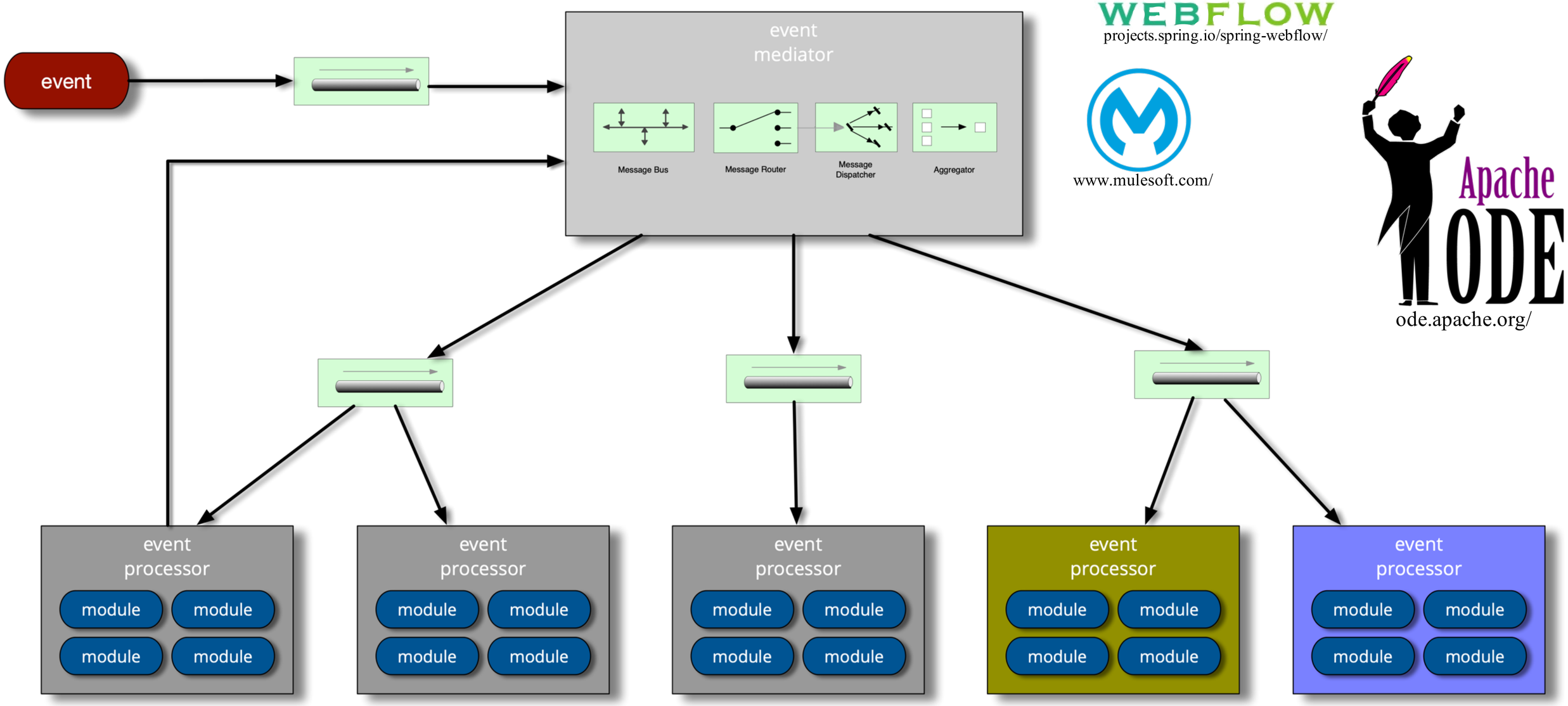


Spring
WEBFLOW
projects.spring.io/spring-webflow/

APACHE
Camel
camel.apache.org/

www.mulesoft.com/

Meditator EDA



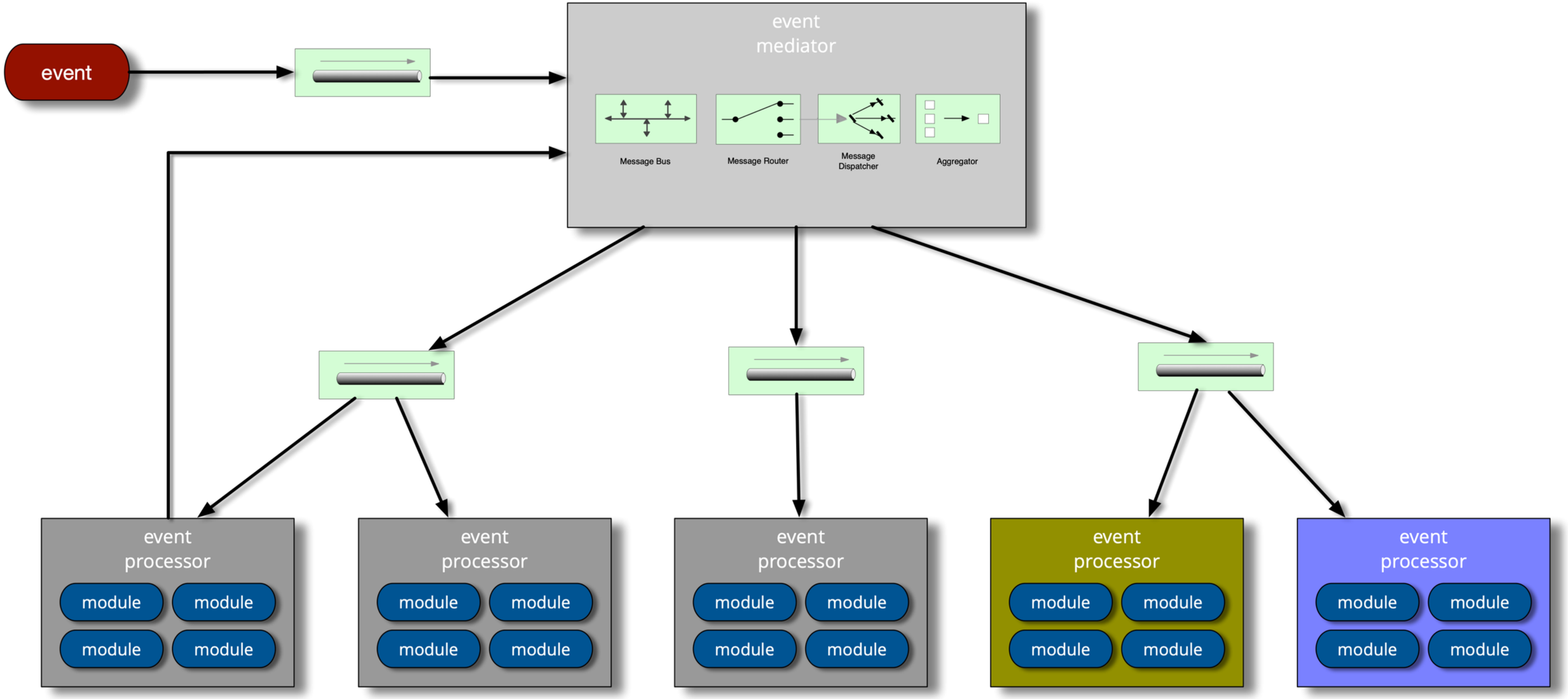
Spring WEBFLOW
projects.spring.io/spring-webflow/

APACHE Camel
camel.apache.org/

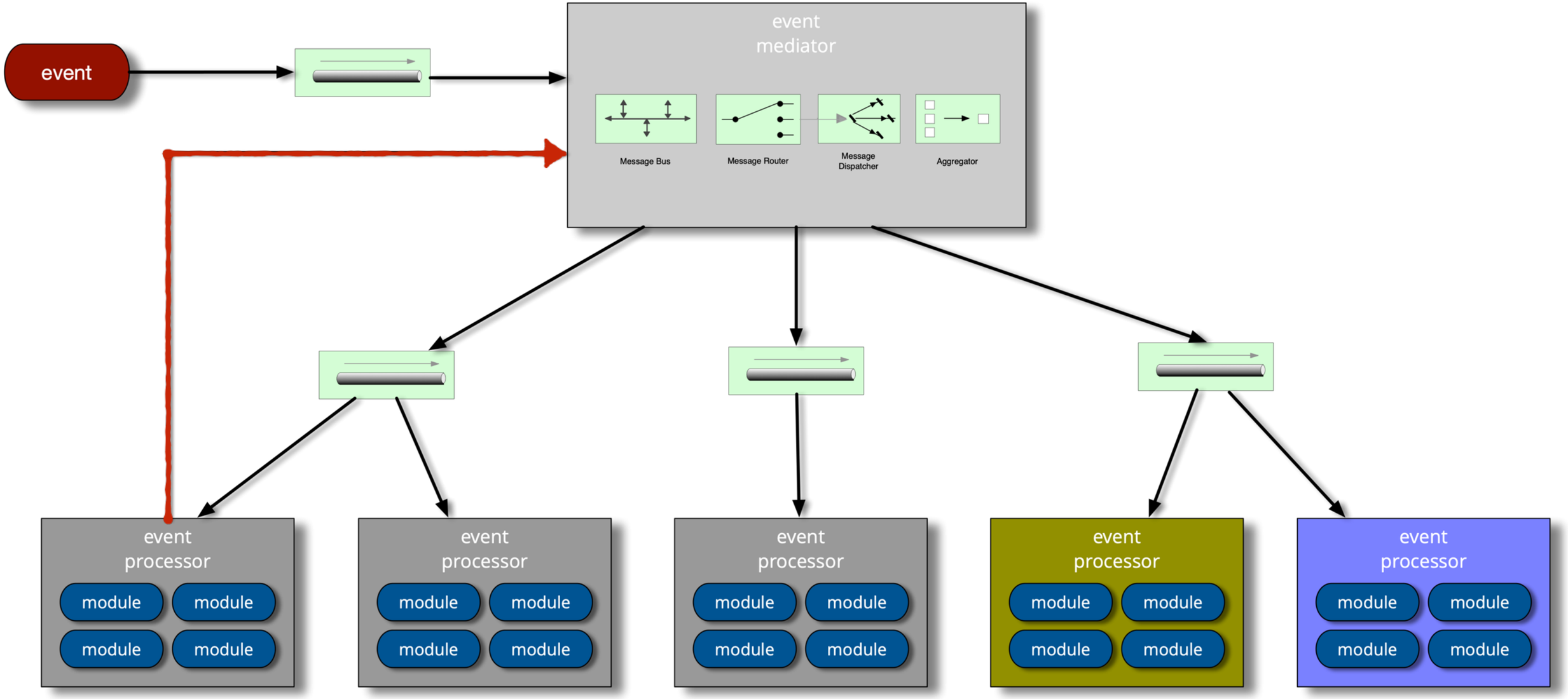
www.mulesoft.com/

Apache ODE
ode.apache.org/

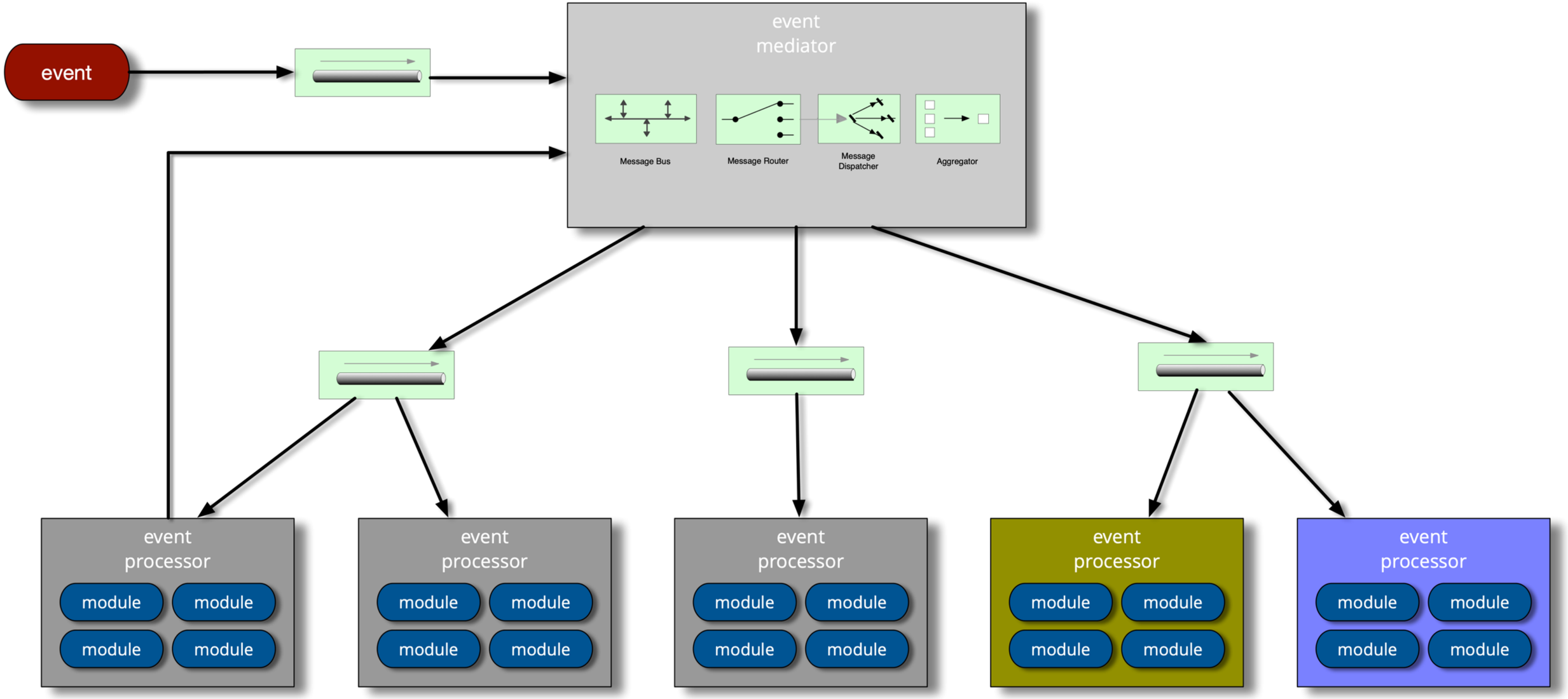
Meditator EDA



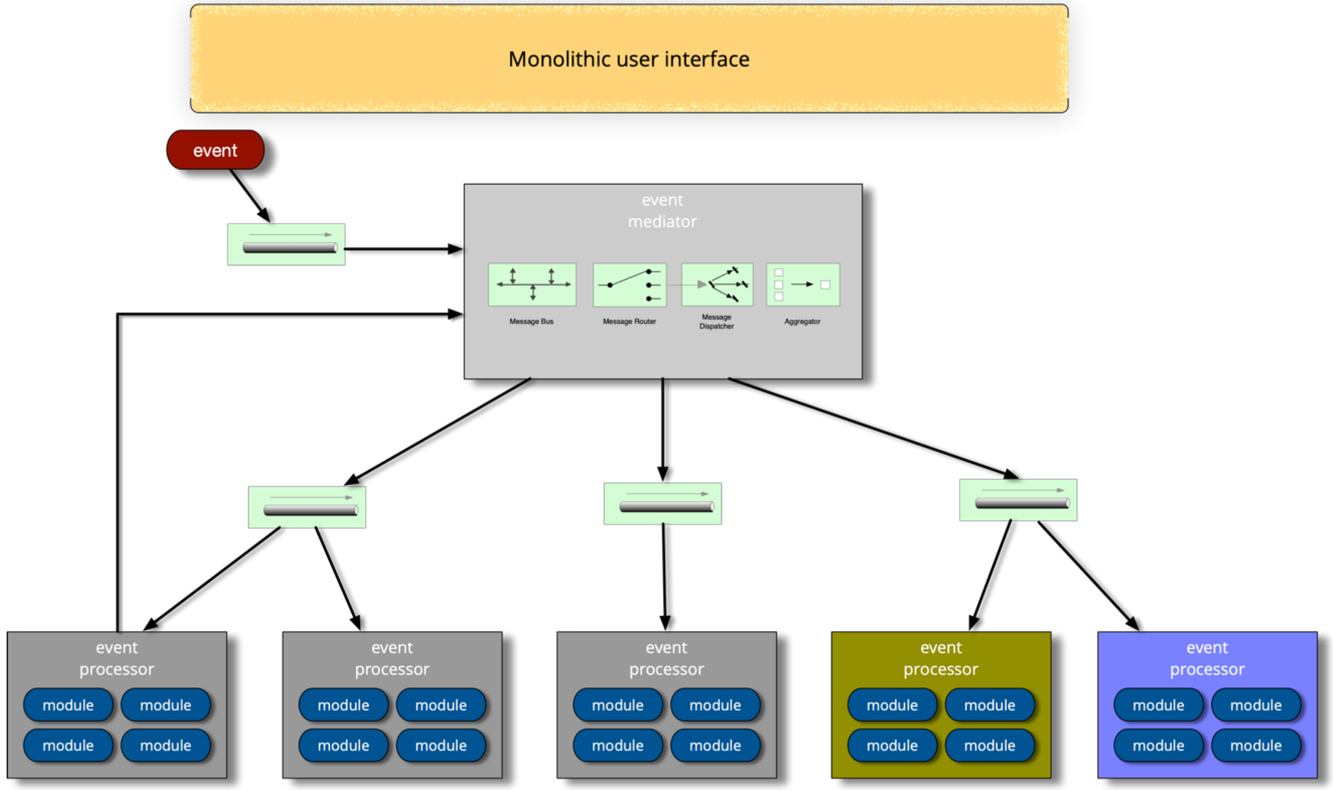
Meditator EDA



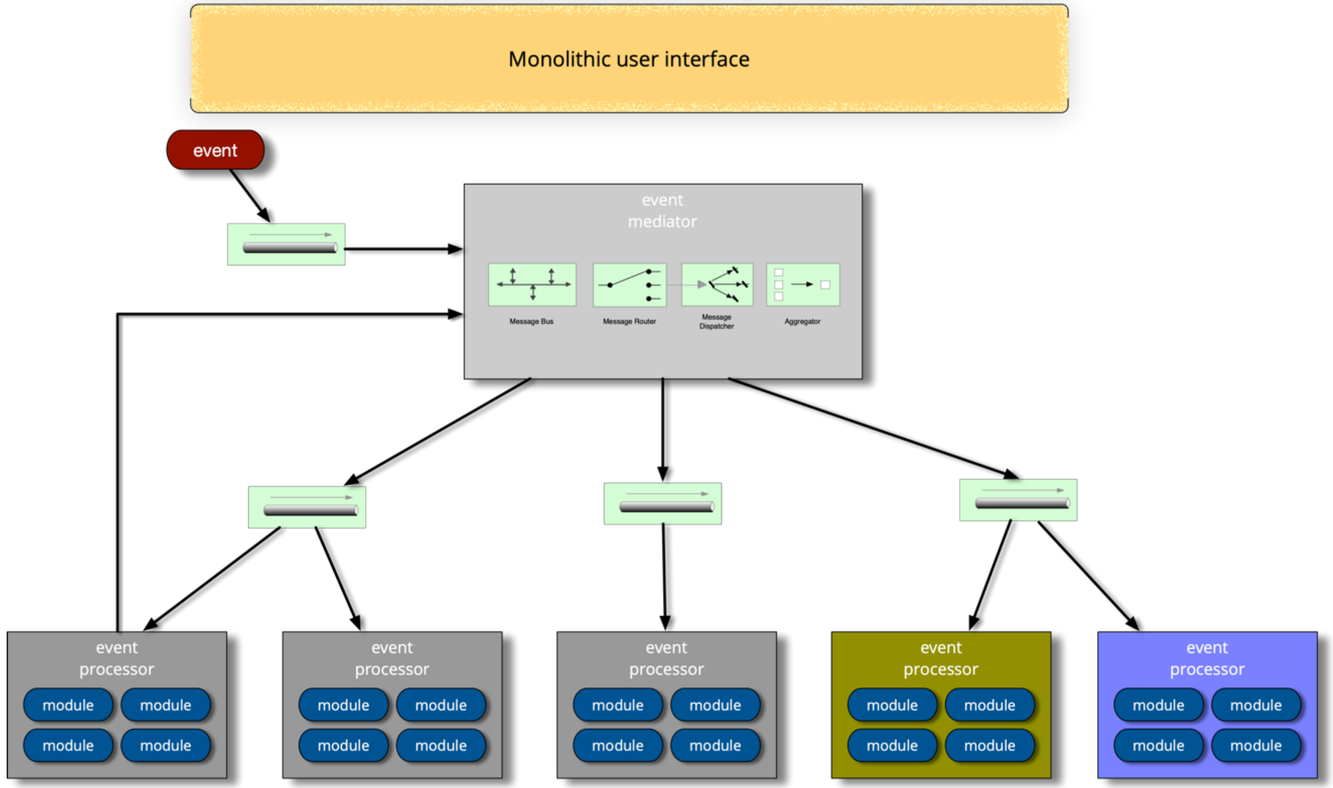
Mediator Quanta

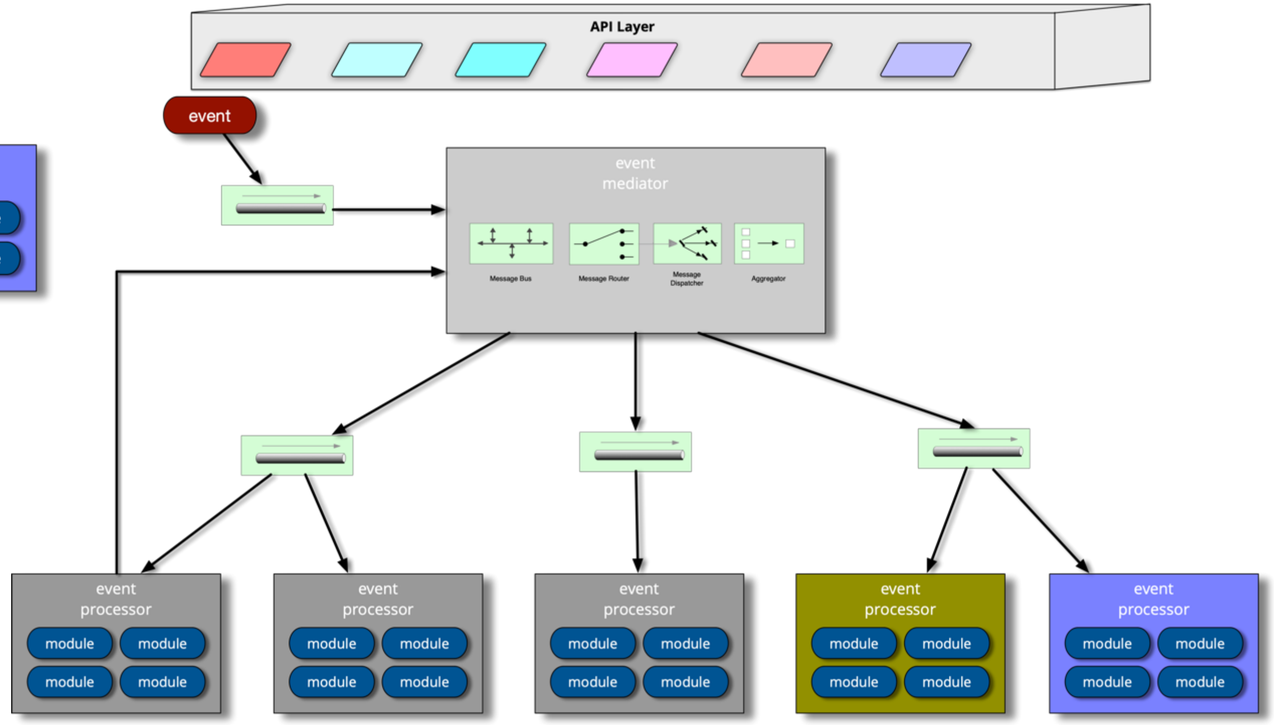


Mediator: UI

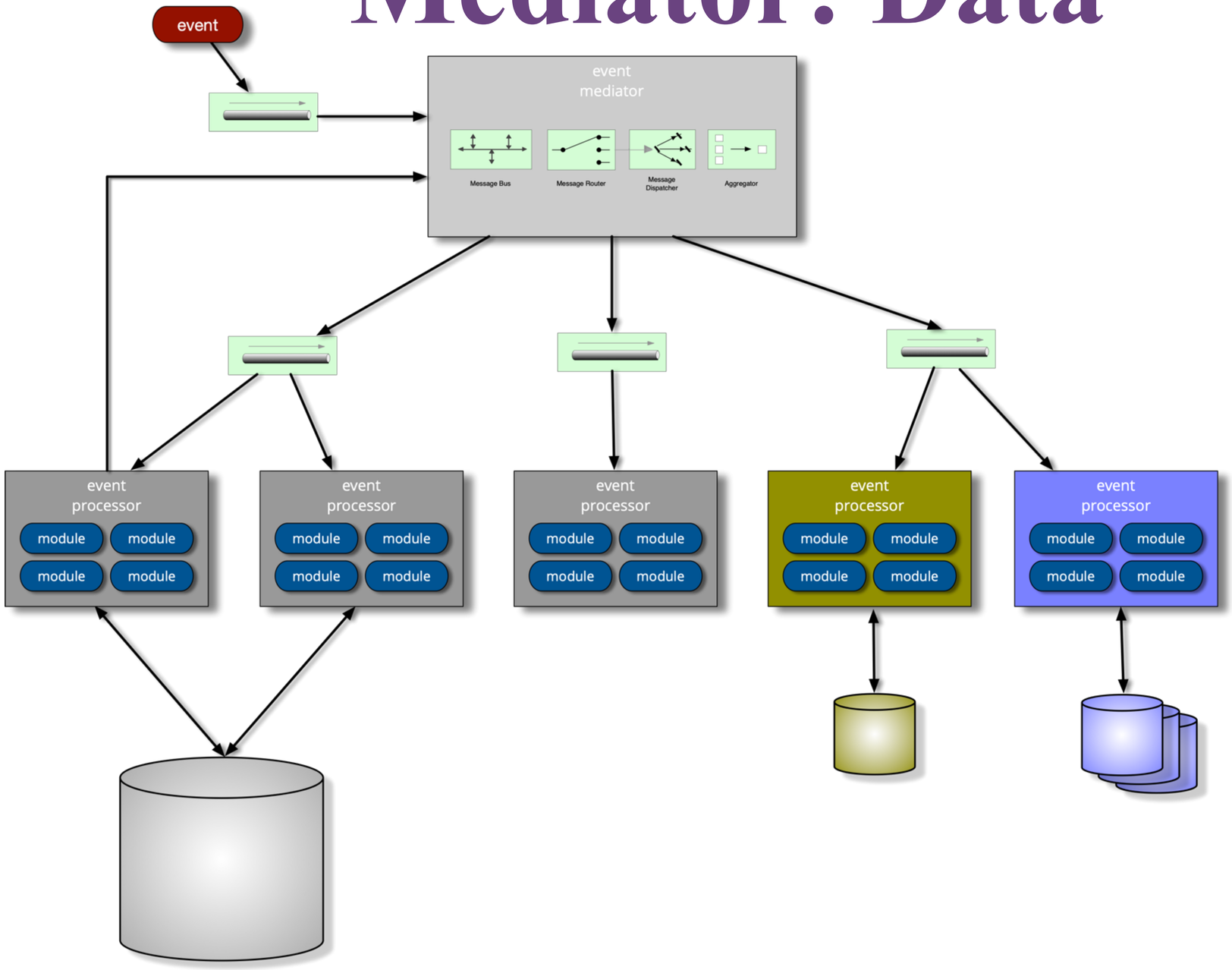

Monolithic user interface
event
event
mediator
Message Bus
Message Router
Message
Dispatcher
Aggregator
event
processor
module
module
module
module
event
processor
module
module
module
module
event
processor
module
module
module
module
event
processor
module
module
module
module
event
processor
module
module
module
module

Mediator: UI

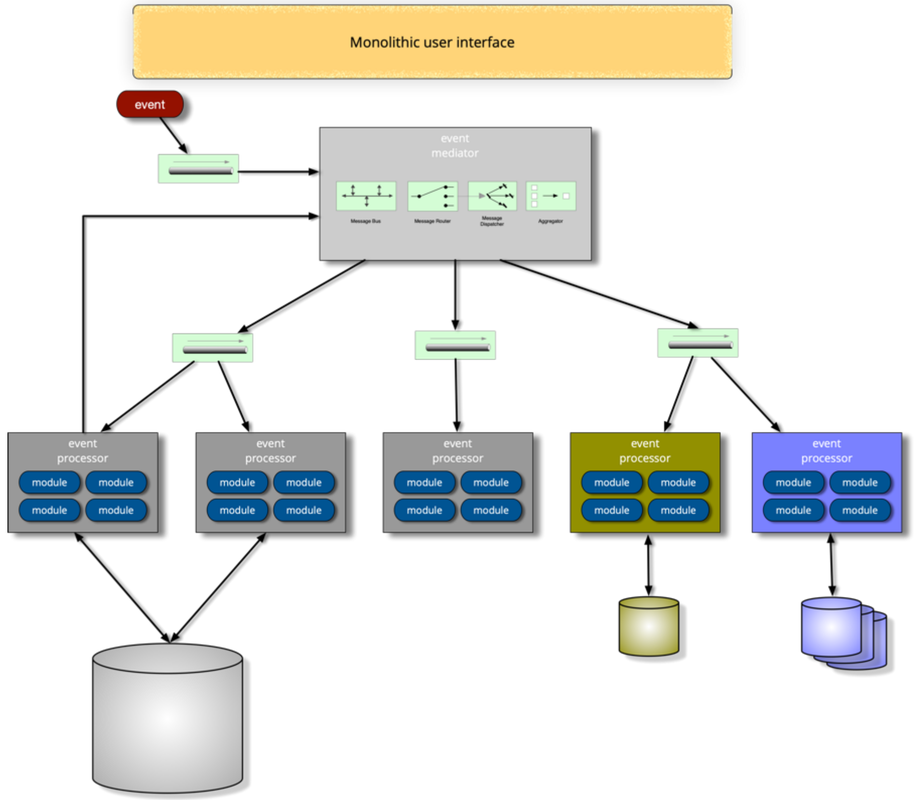

Monolithic user interface
event
event
mediator
Message Bus
Message Router
Message
Dispatcher
Aggregator
event
processor
module
module
module
module
event
processor
module
module
module
module
event
processor
module
module
module
module
event
processor
module
module
module
module
event
processor
module
module
module
module


API Layer
event
event
mediator
Message Bus
Message Router
Message
Dispatcher
Aggregator
event
processor
module
module
module
module
event
processor
module
module
module
module
event
processor
module
module
module
module
event
processor
module
module
module
module
event
processor
module
module
module
module

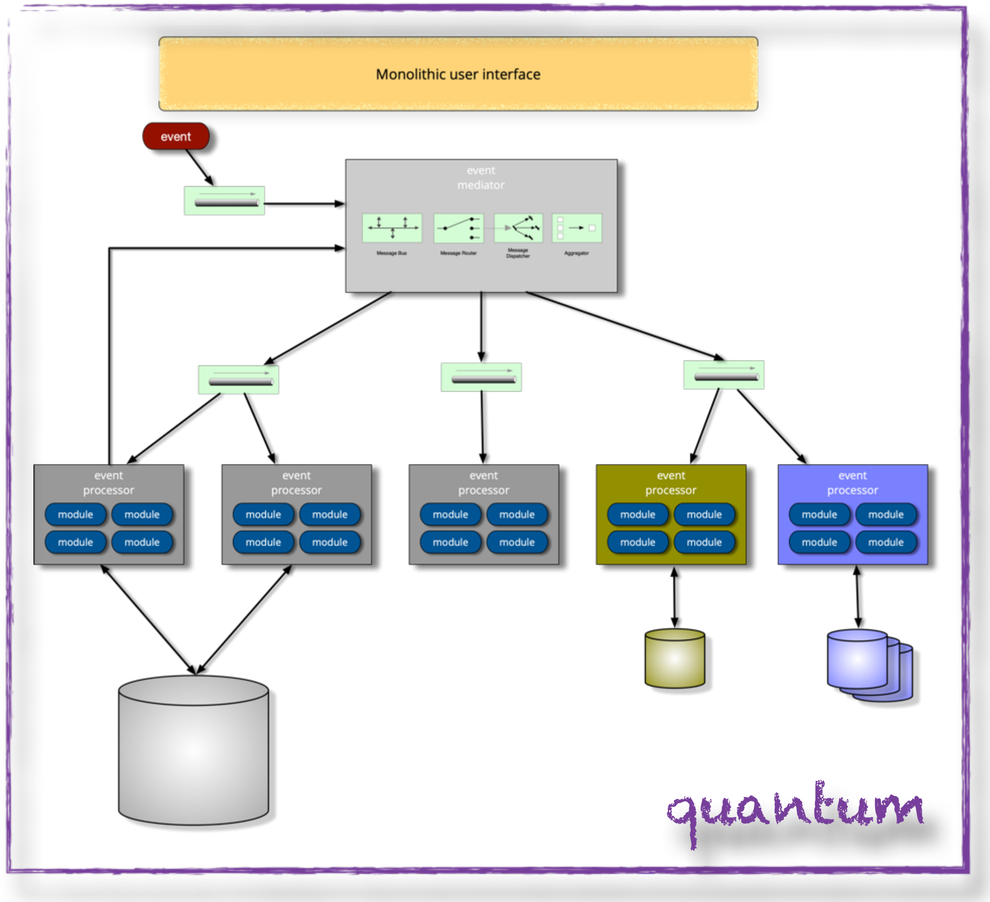
Mediator: Data



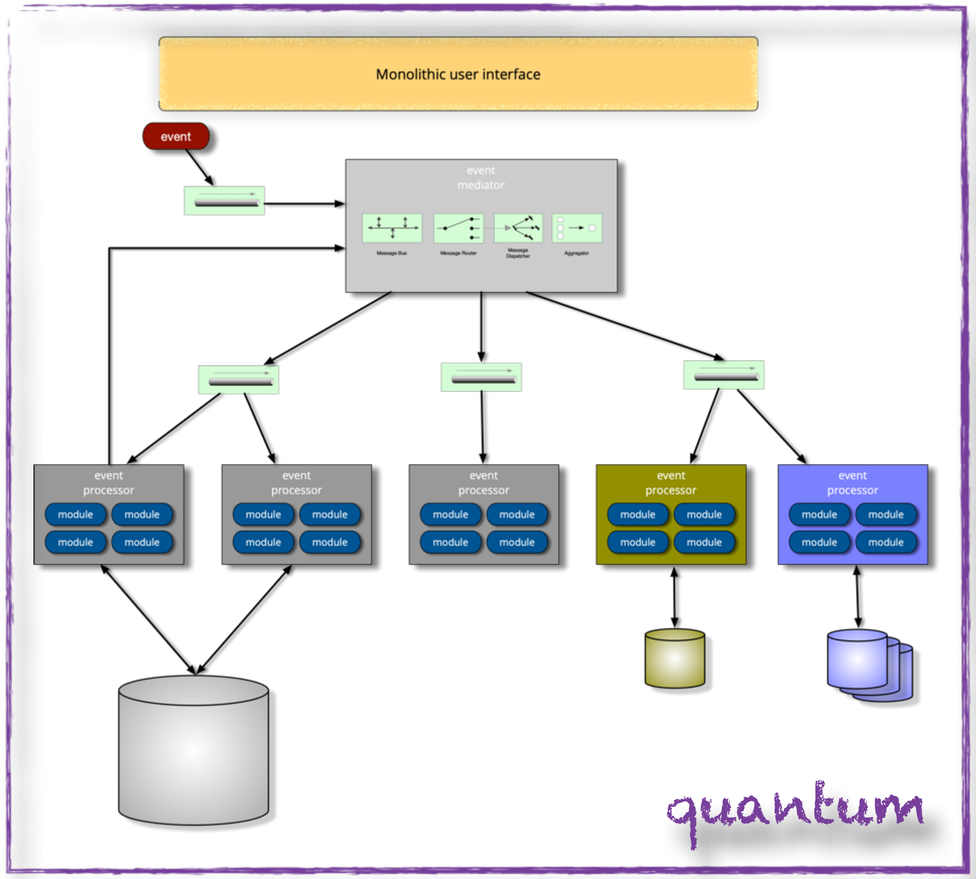
Mediator Quanta

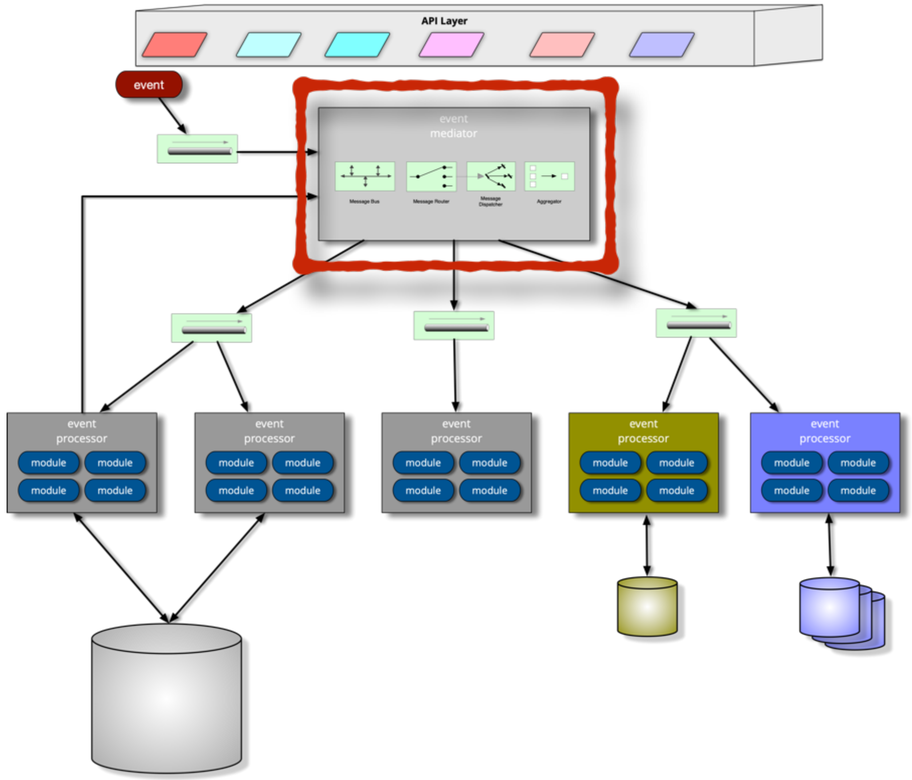


Mediator Quanta

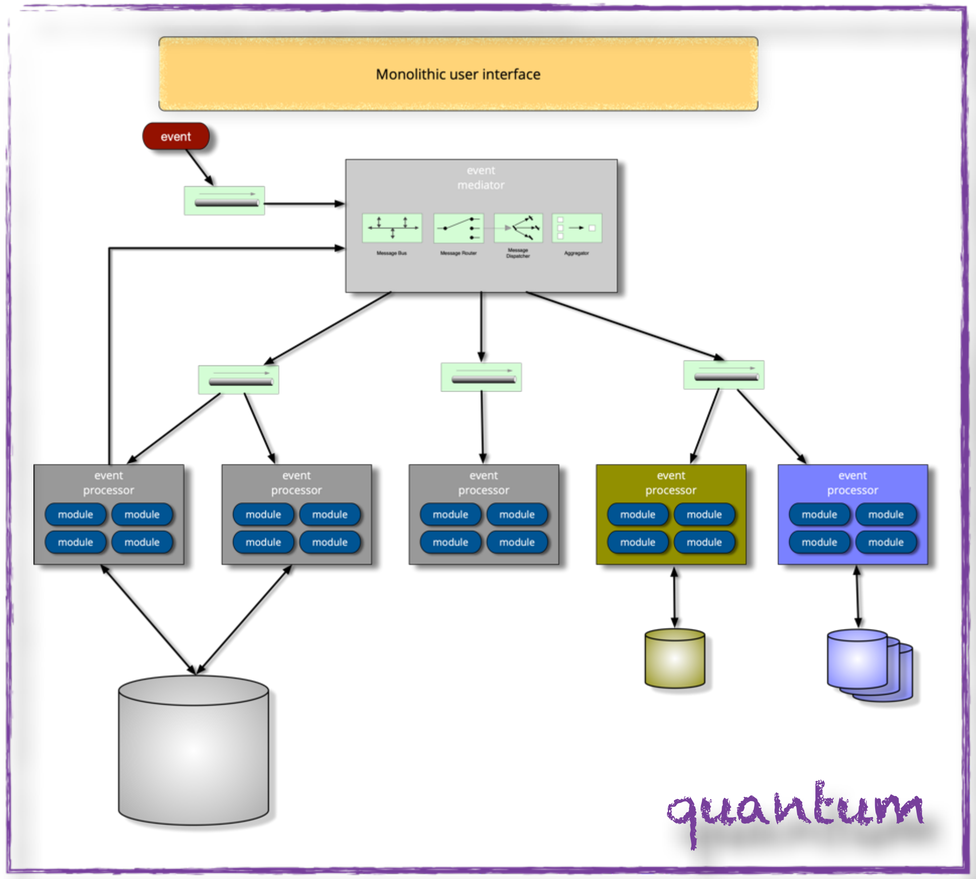


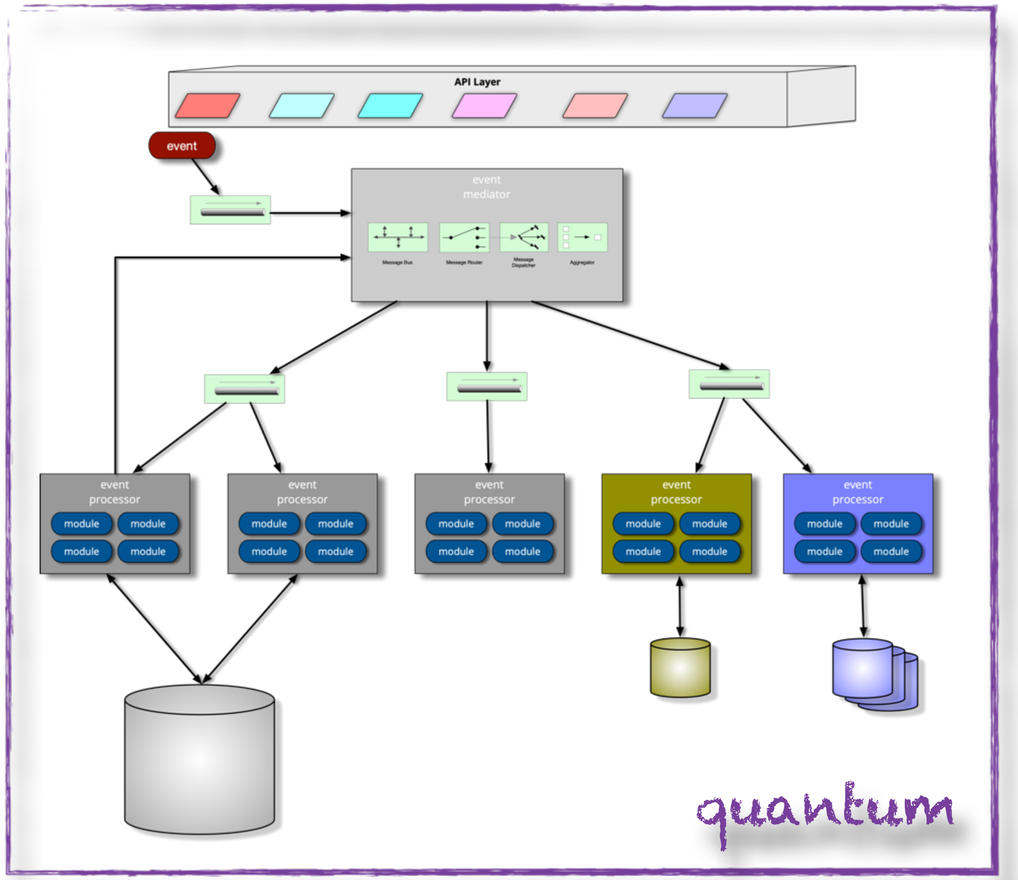
Mediator Quanta



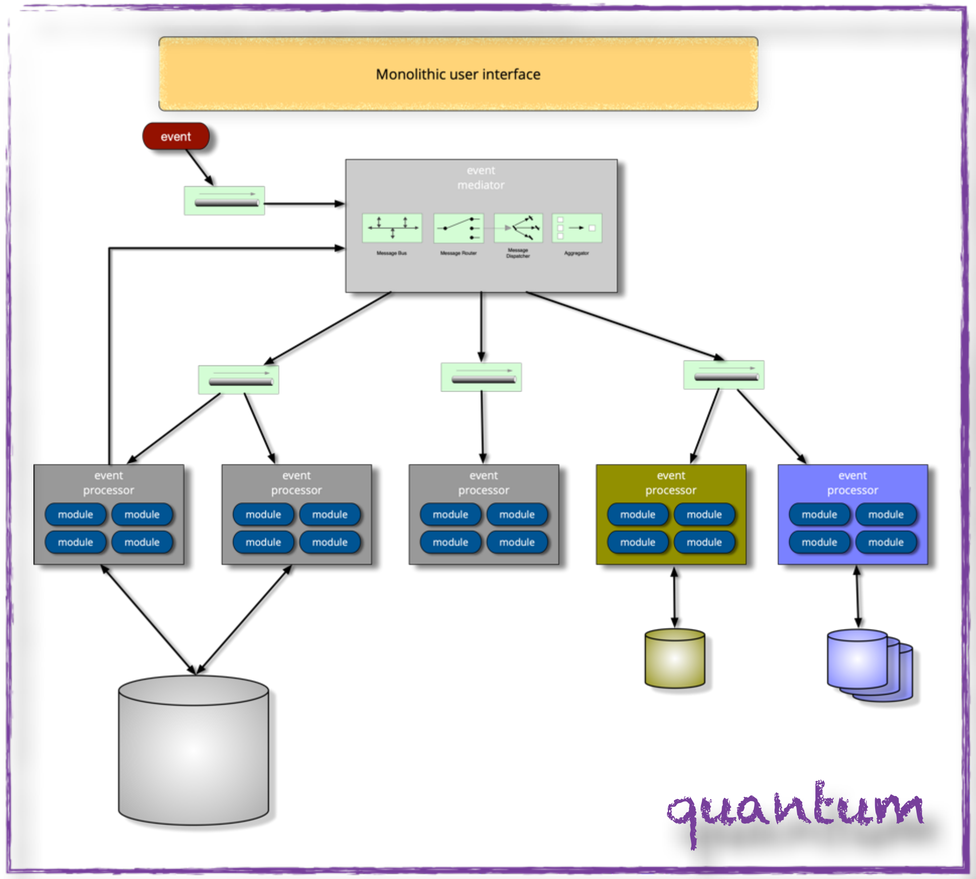


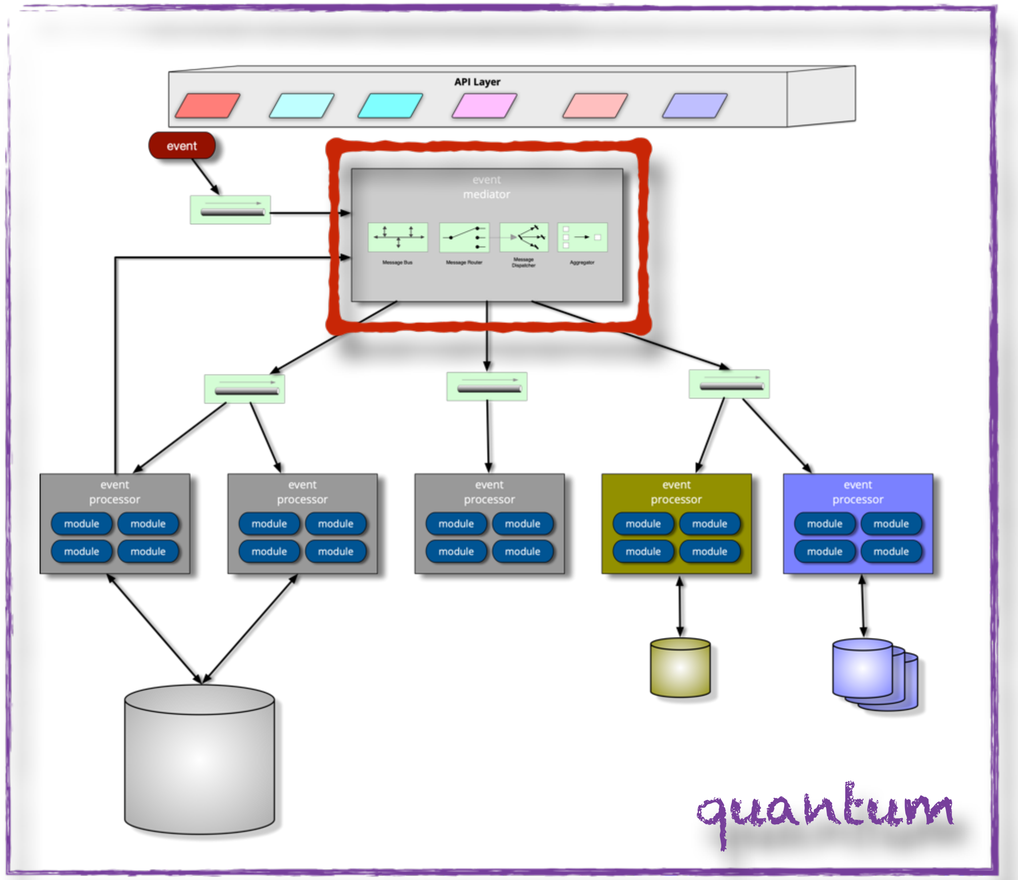
Mediator Quanta



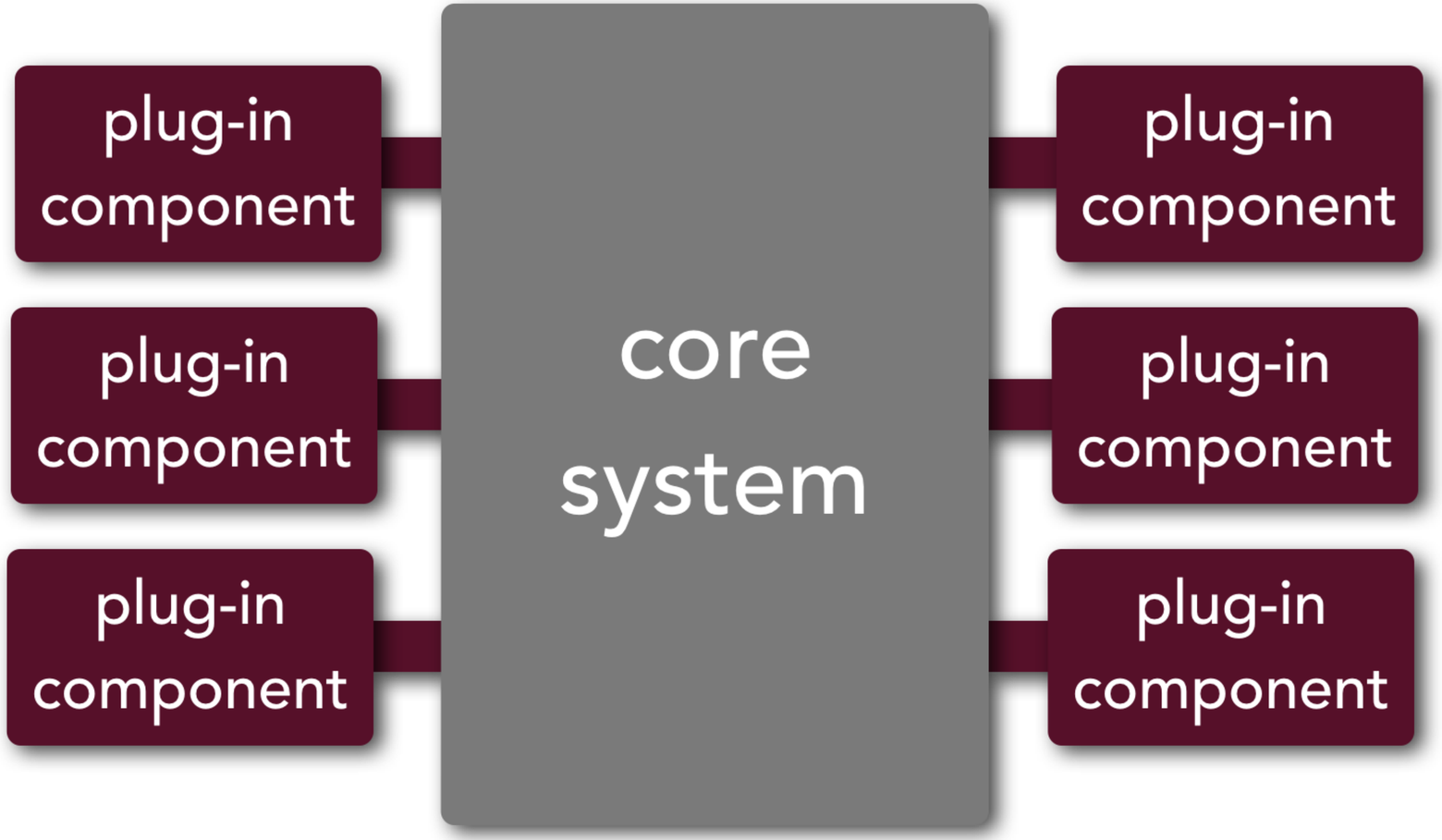


Mediator Quanta

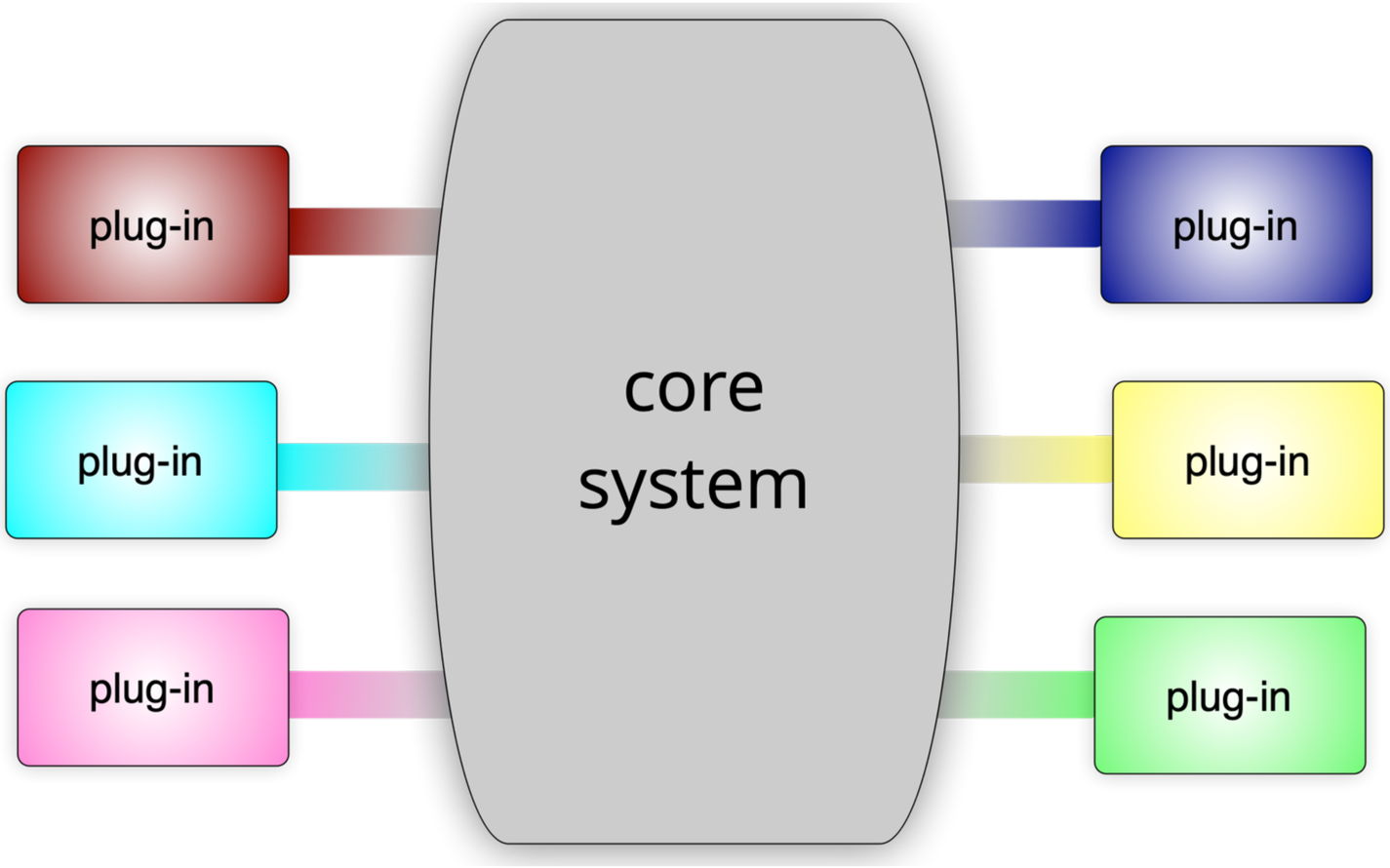




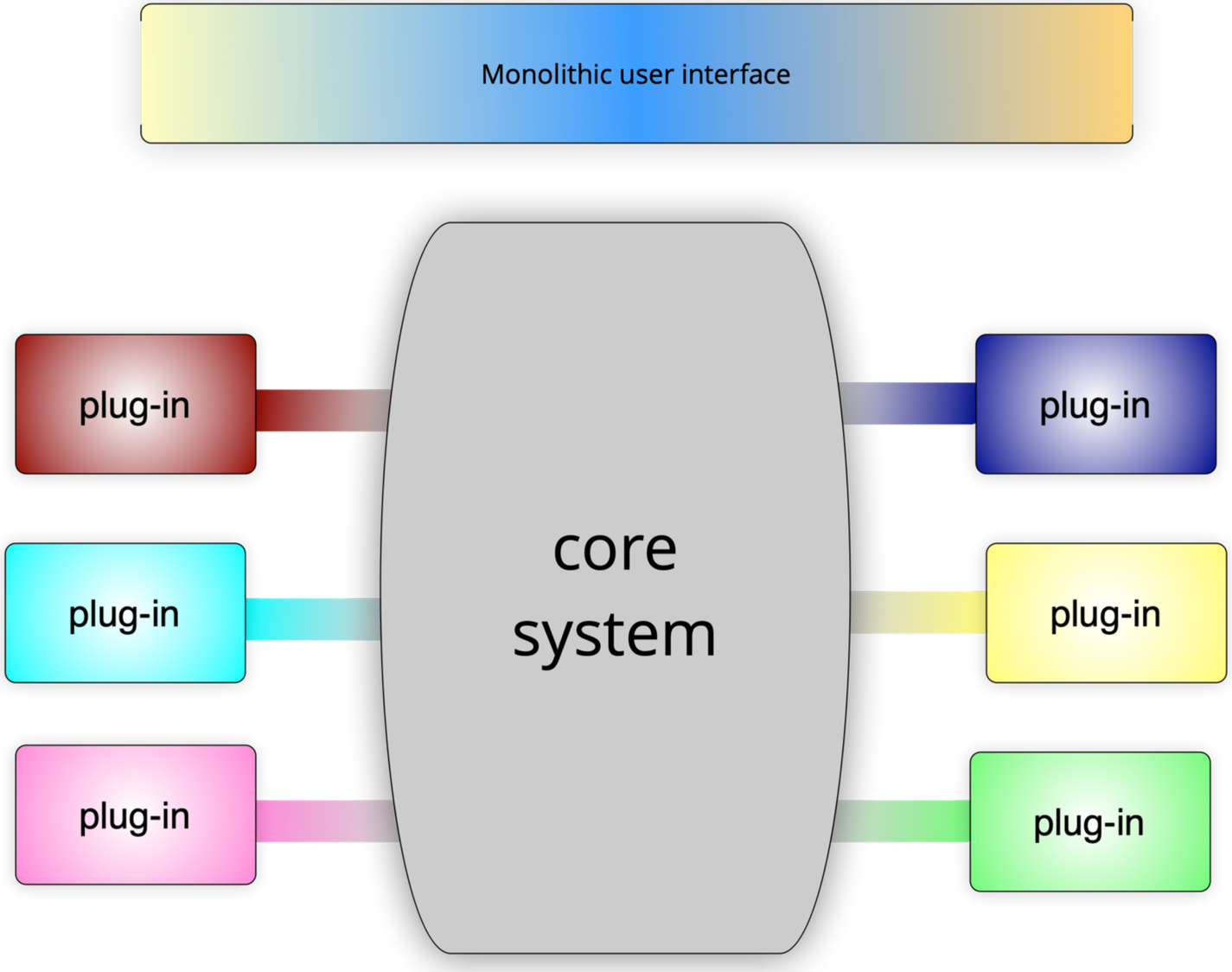
Microkernel



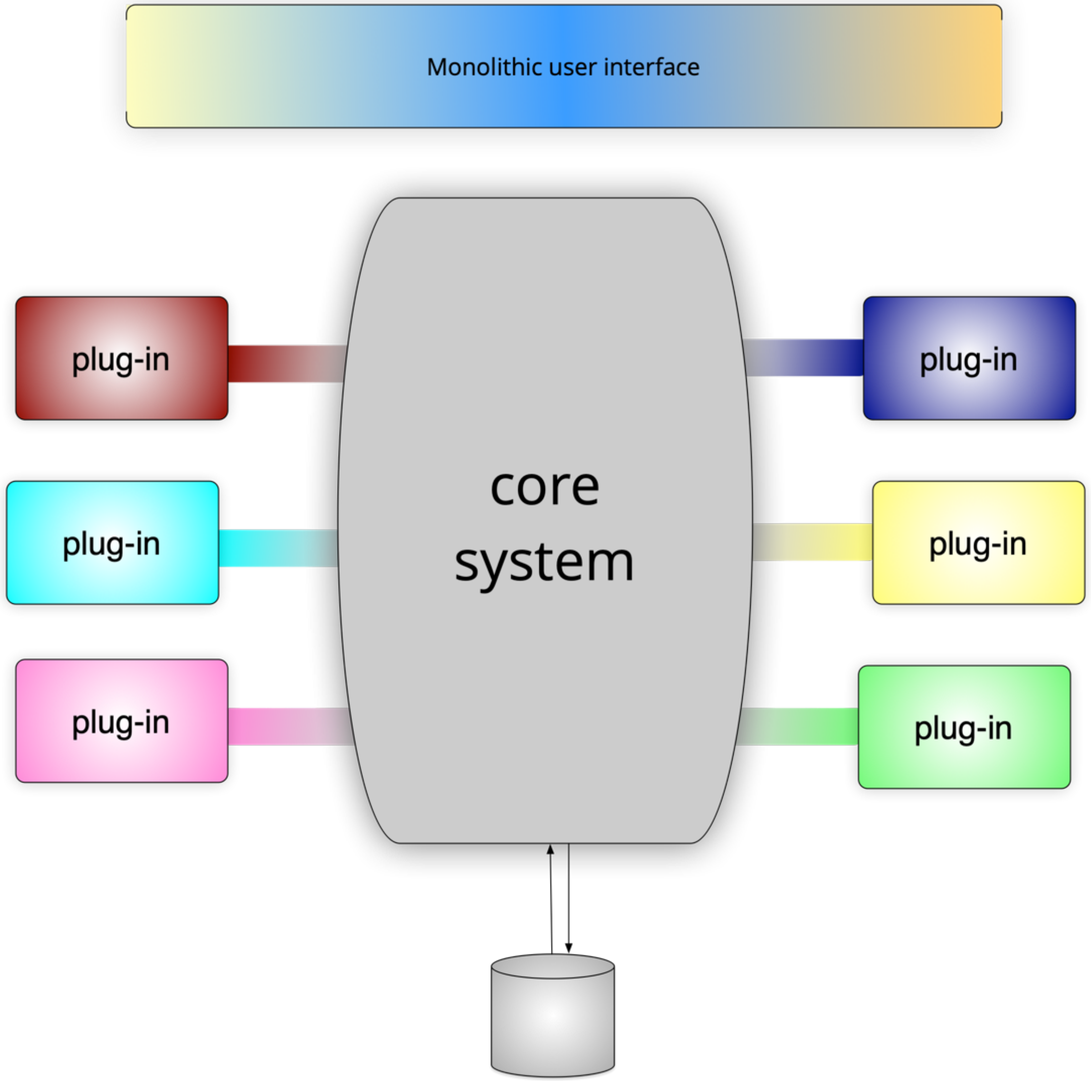
Microkernel Quanta

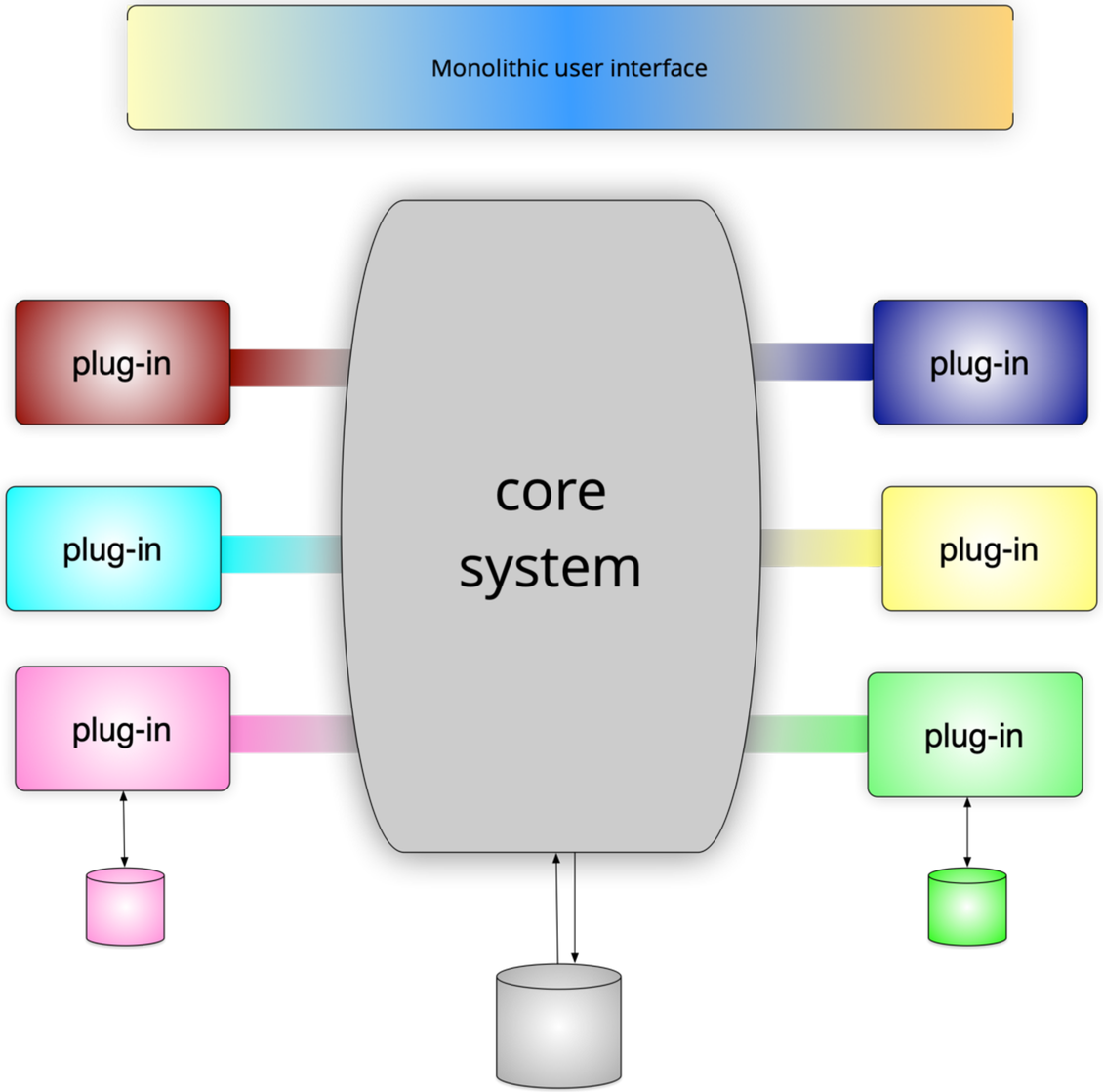


Microkernel: UI

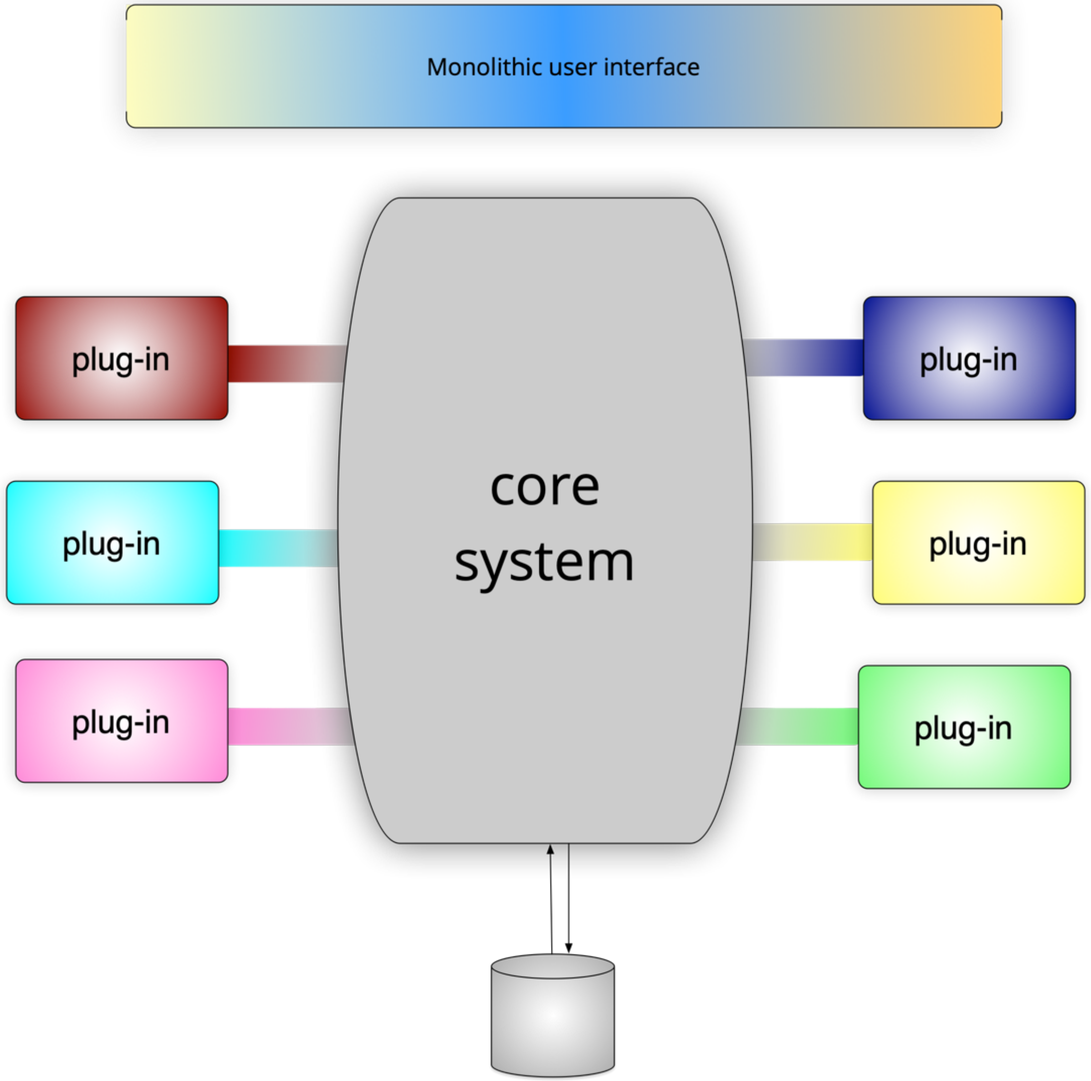


Microkernel: Data

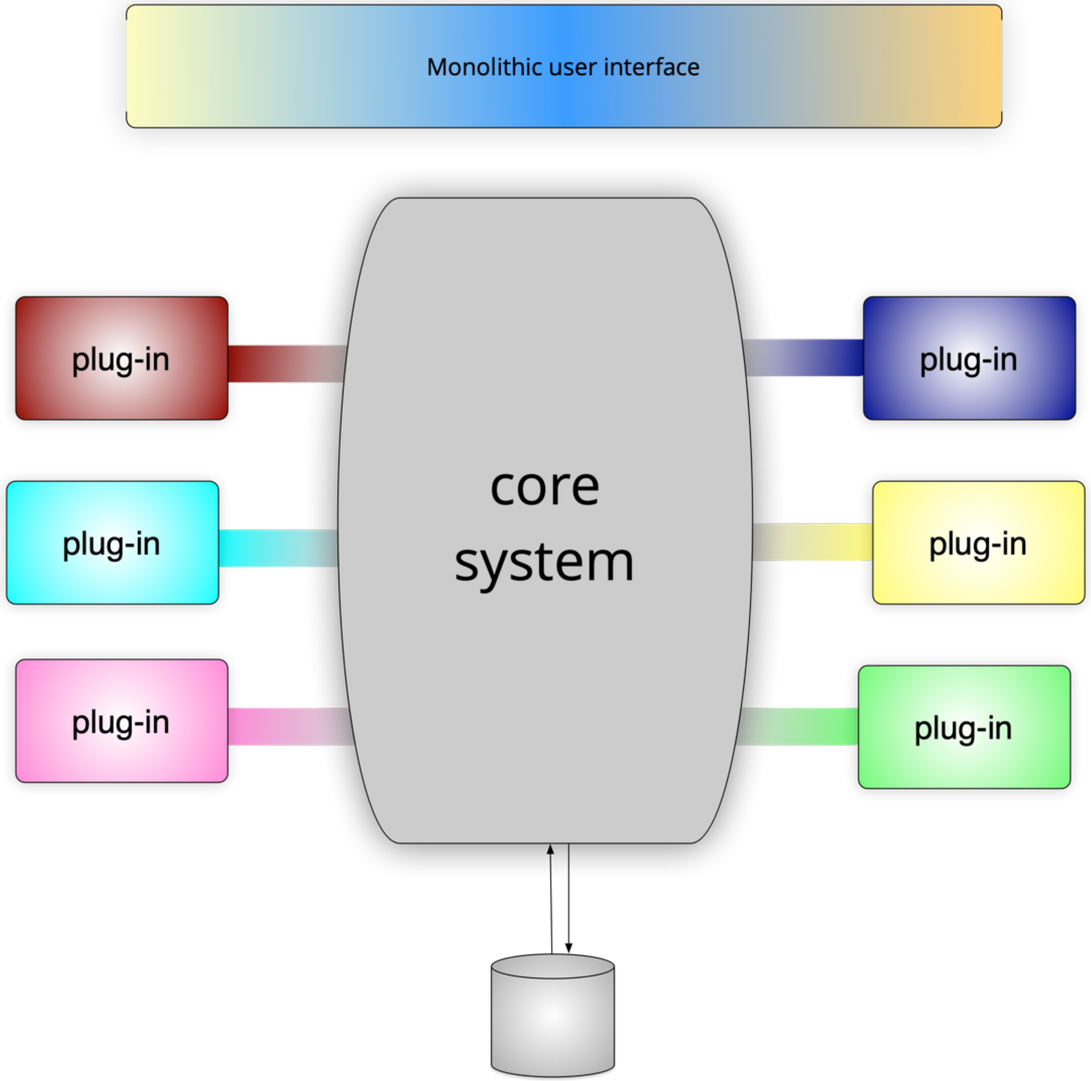


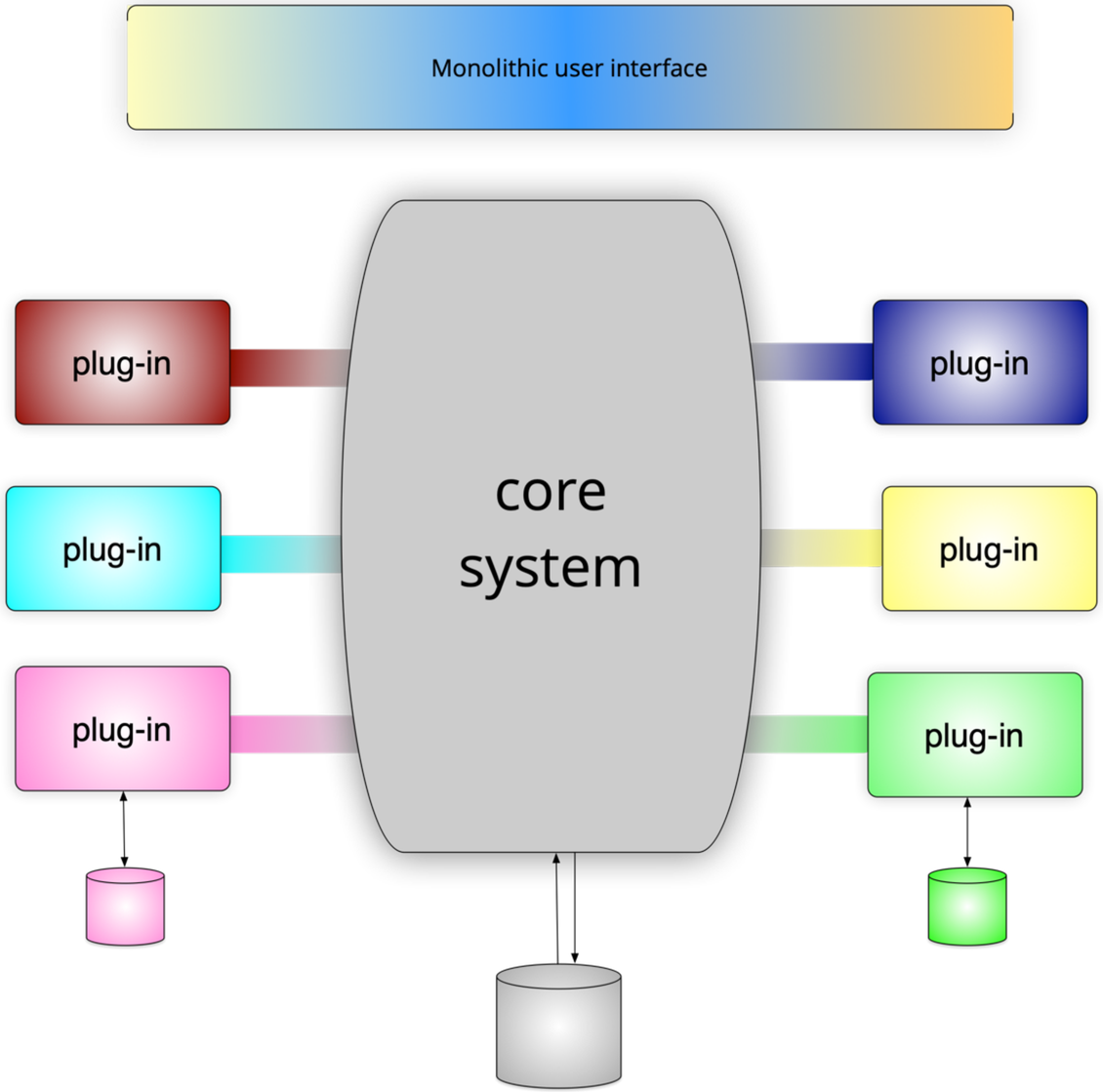


Microkernel: Data

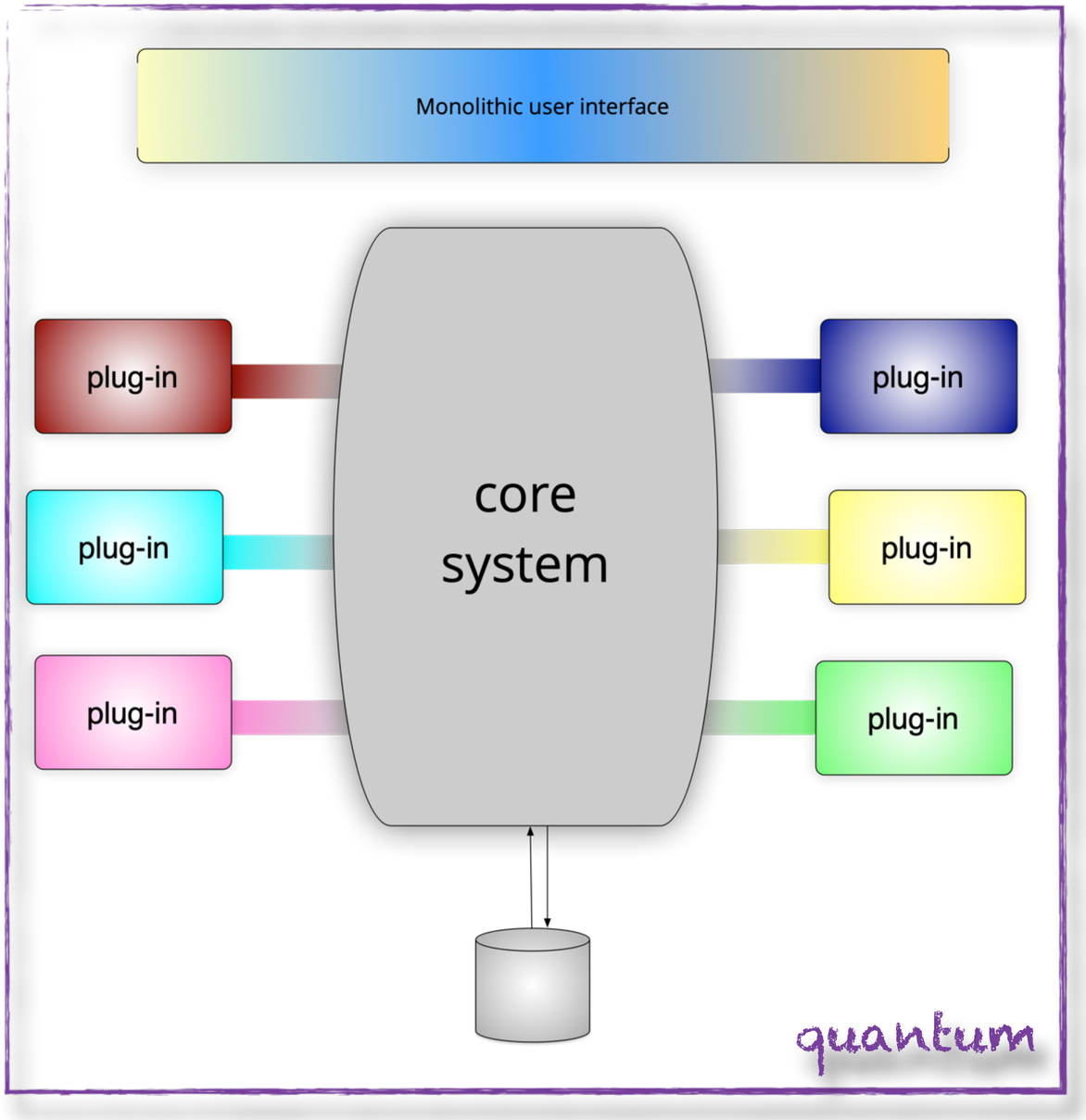


Microkernel: Data

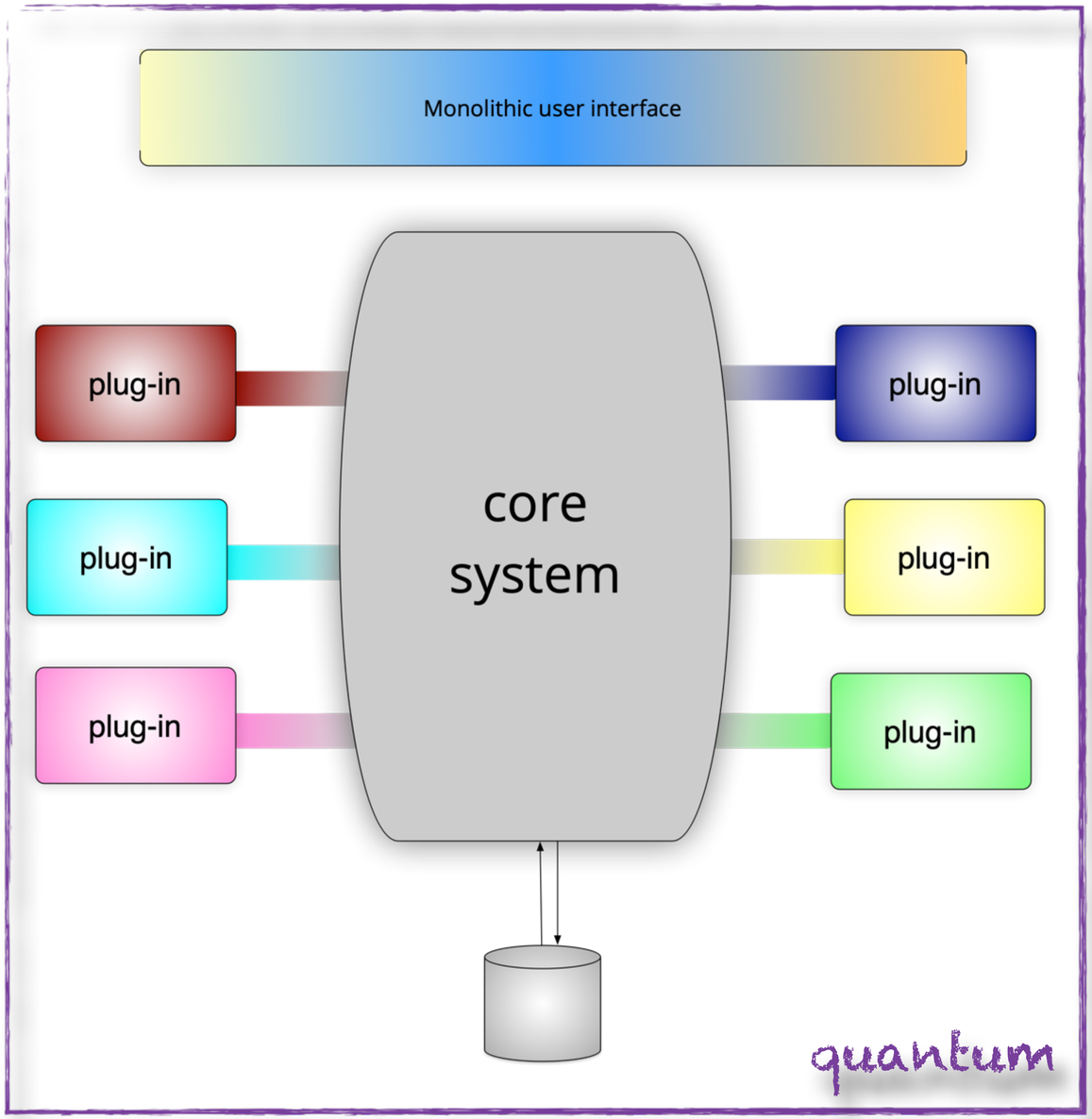


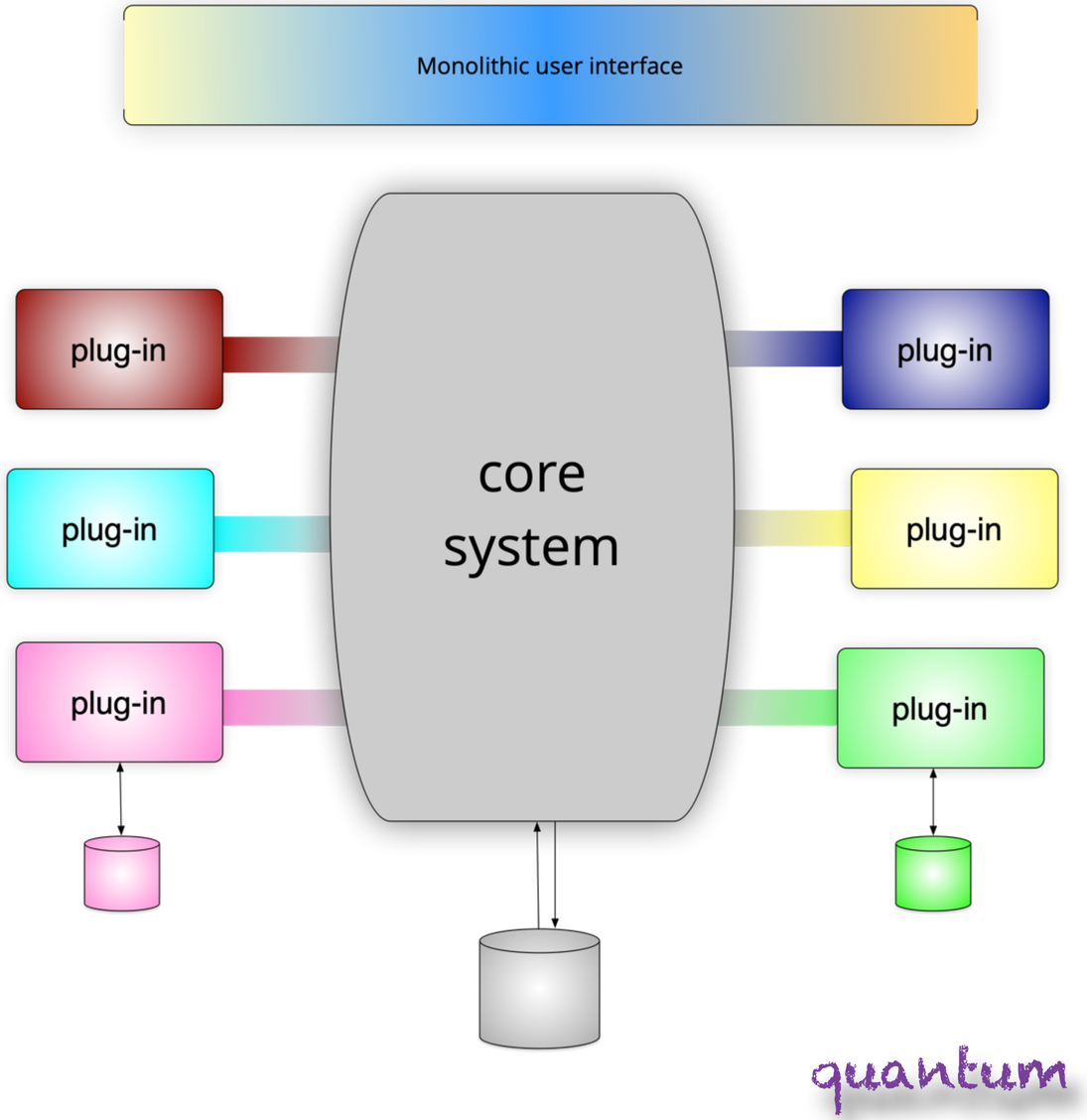


Microkernel: Quanta

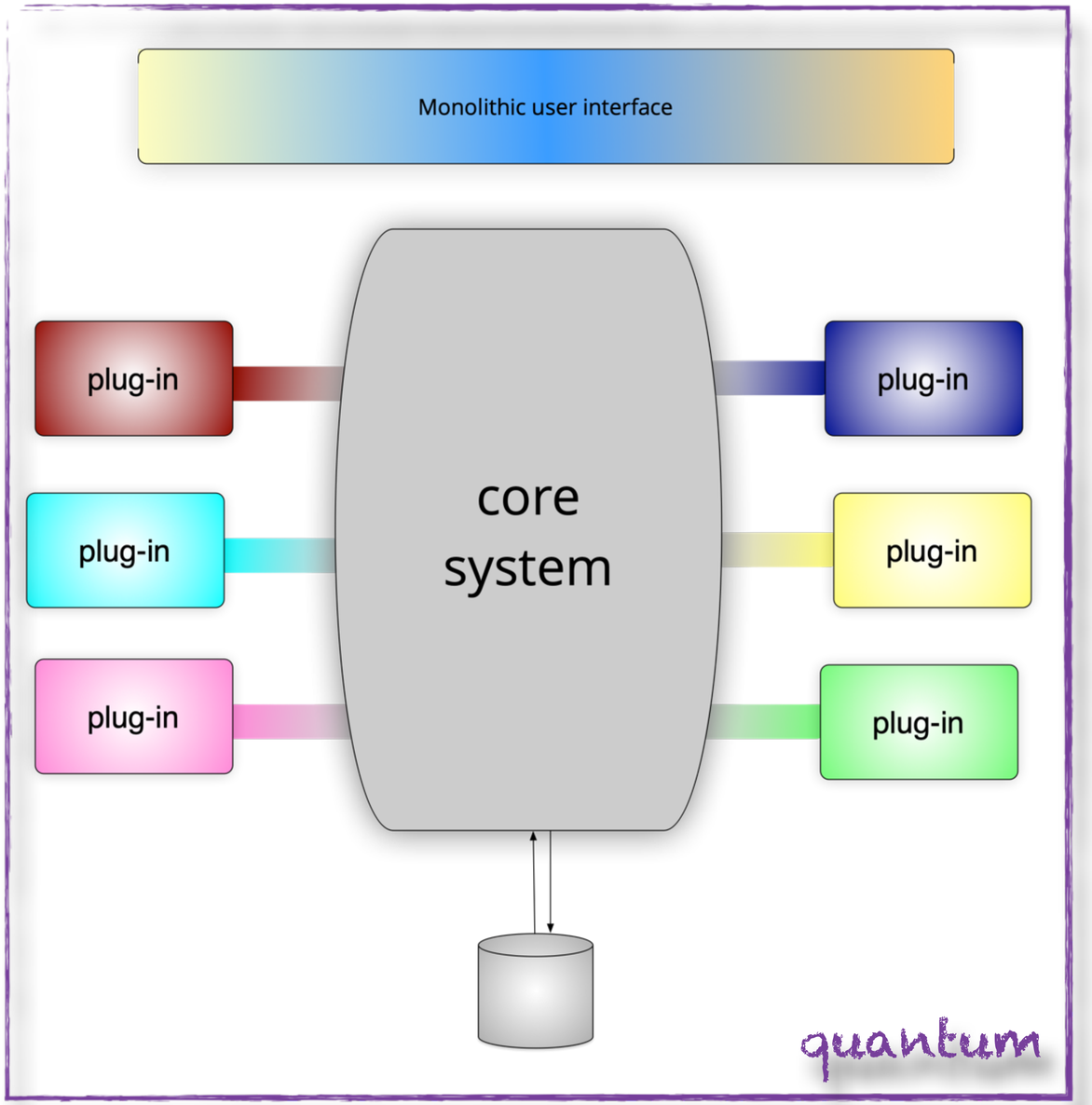


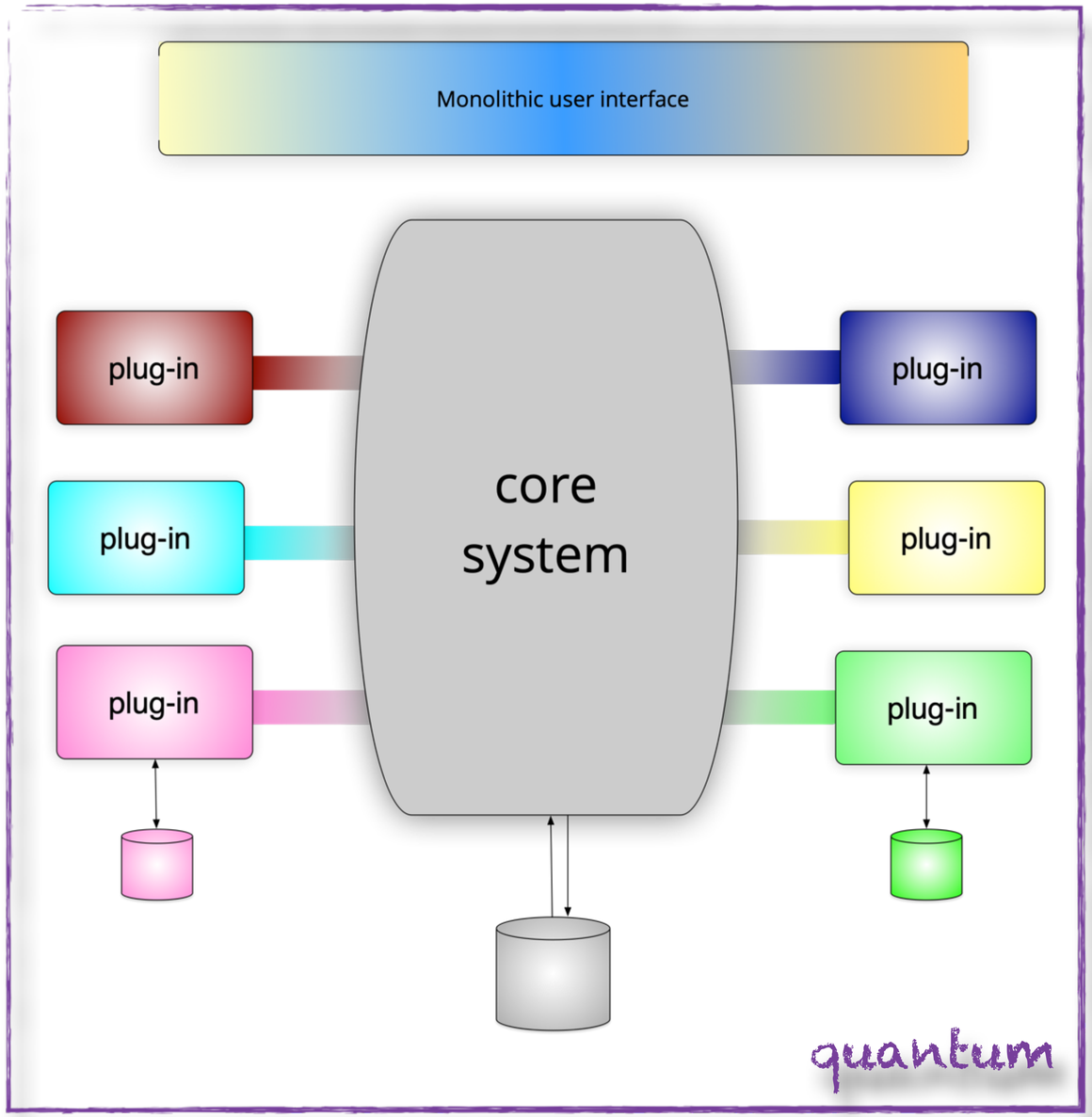
Microkernel: Quanta



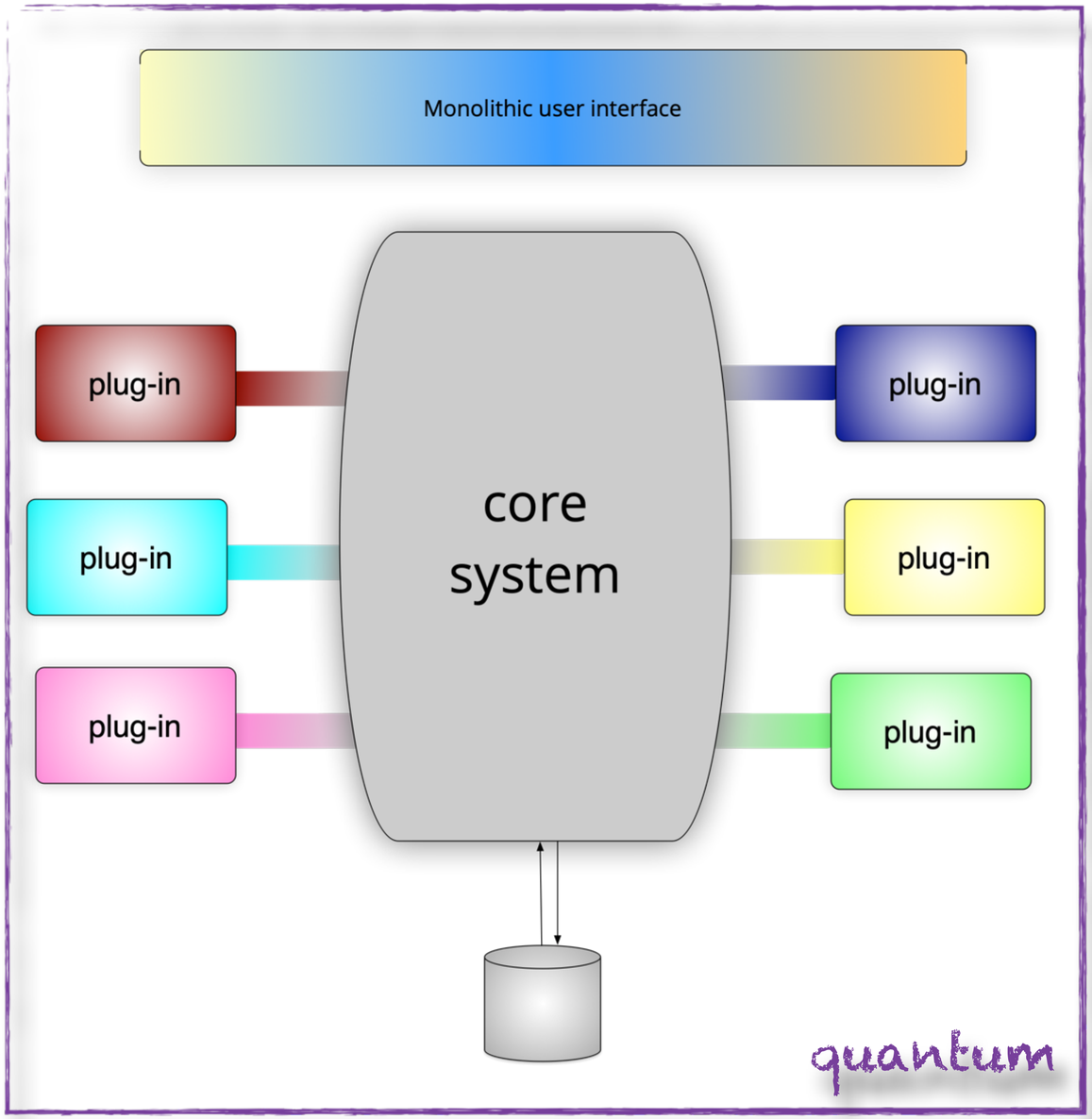


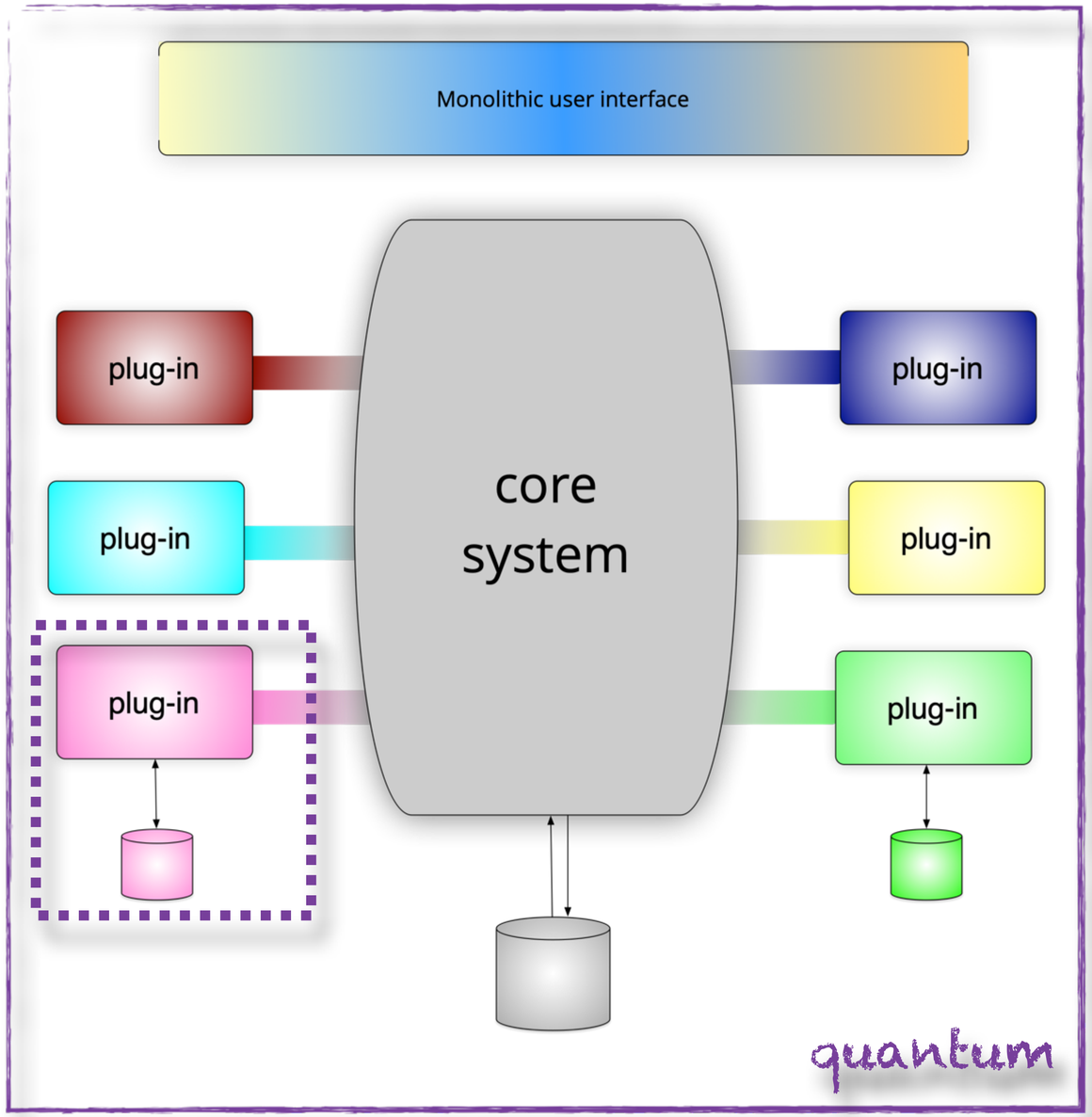
Microkernel: Quanta



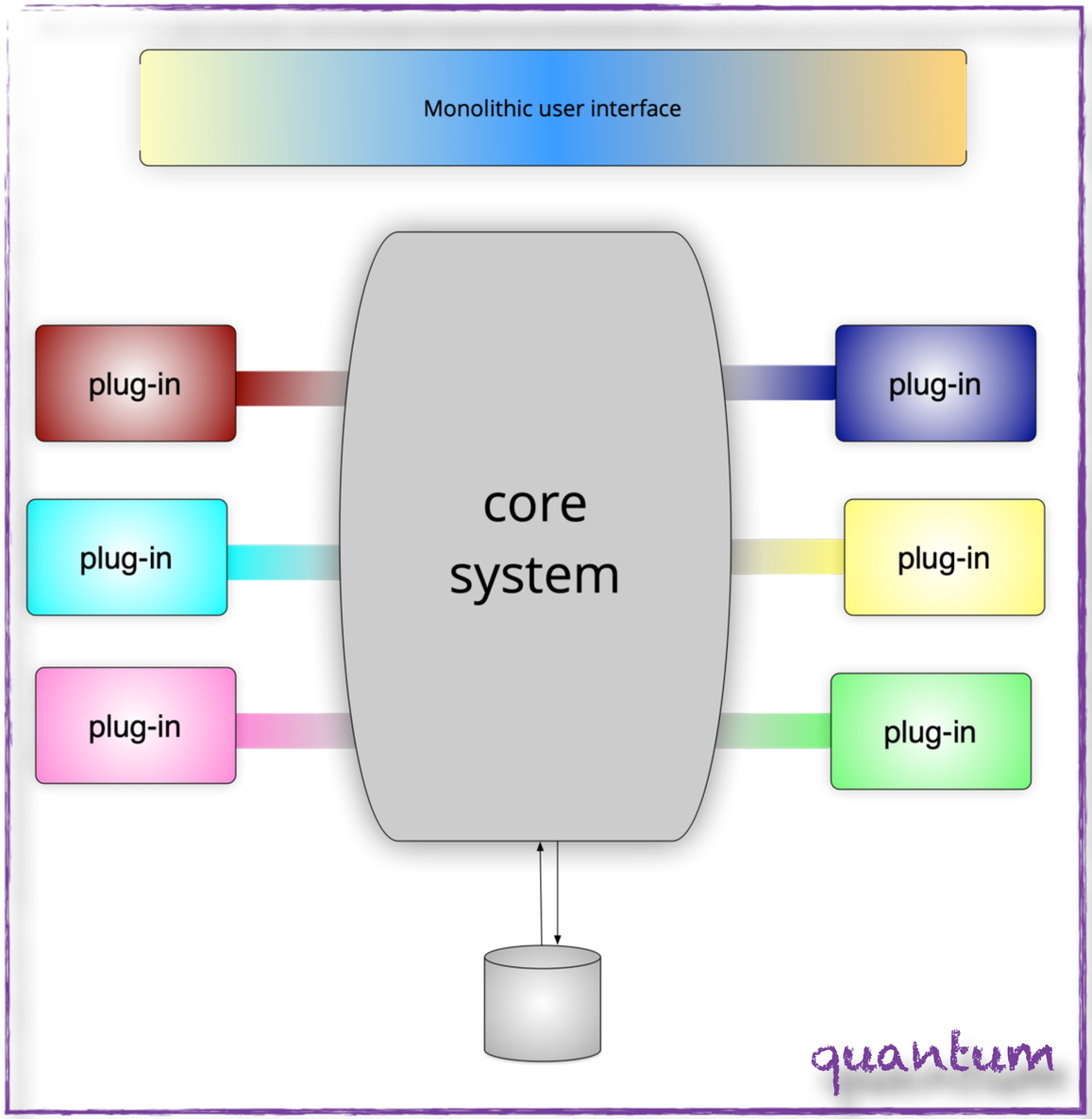


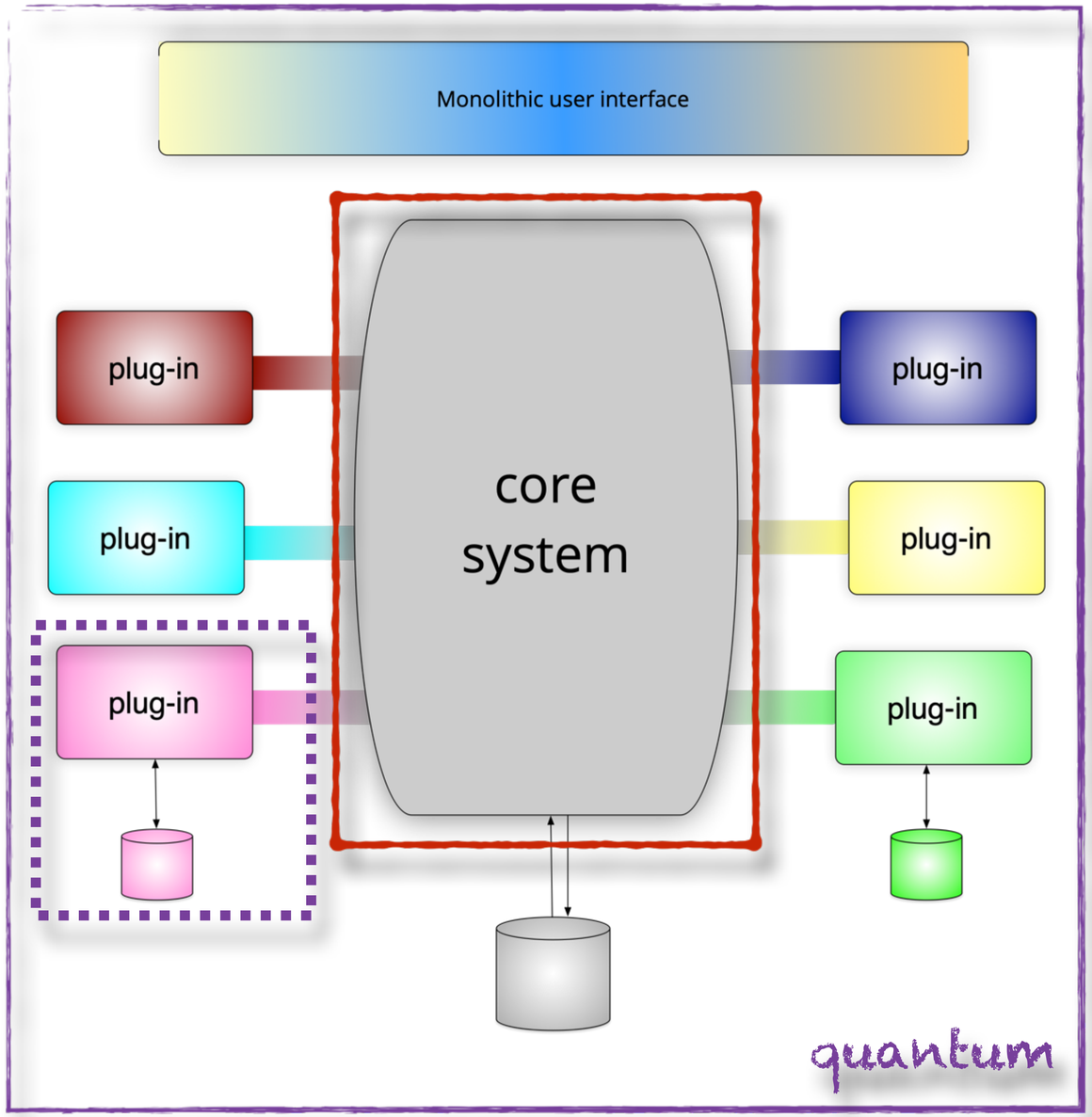
Microkernel: Quanta



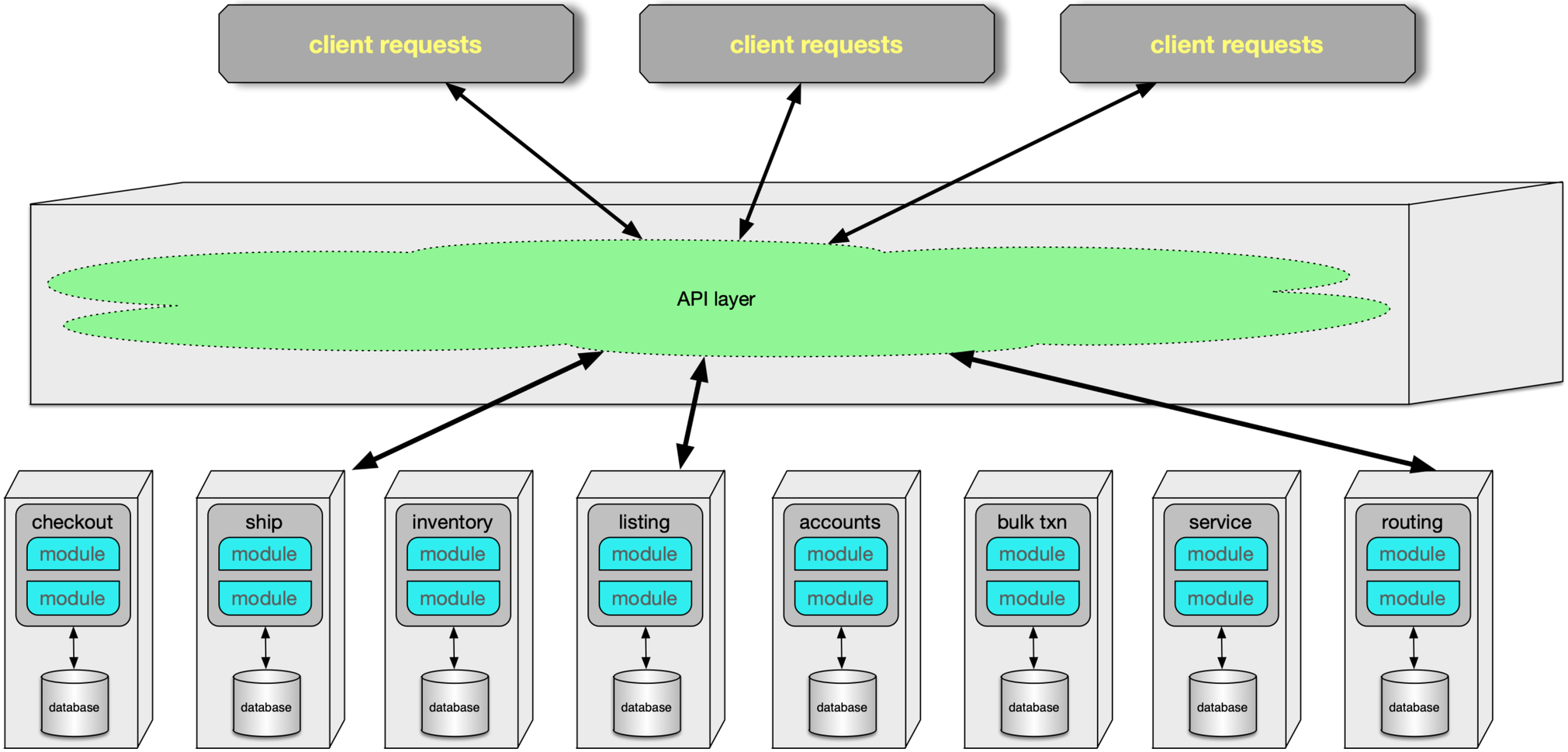


Microkernel: Quanta



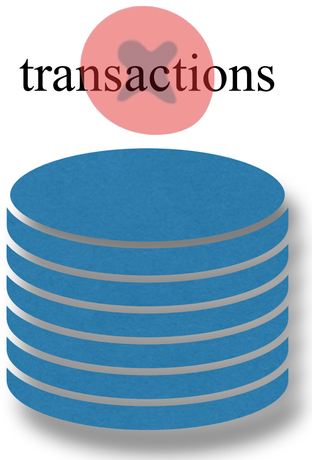


Microservices

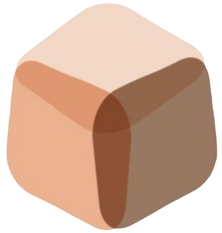


What makes microservices so evolvable?

extremely loose coupling

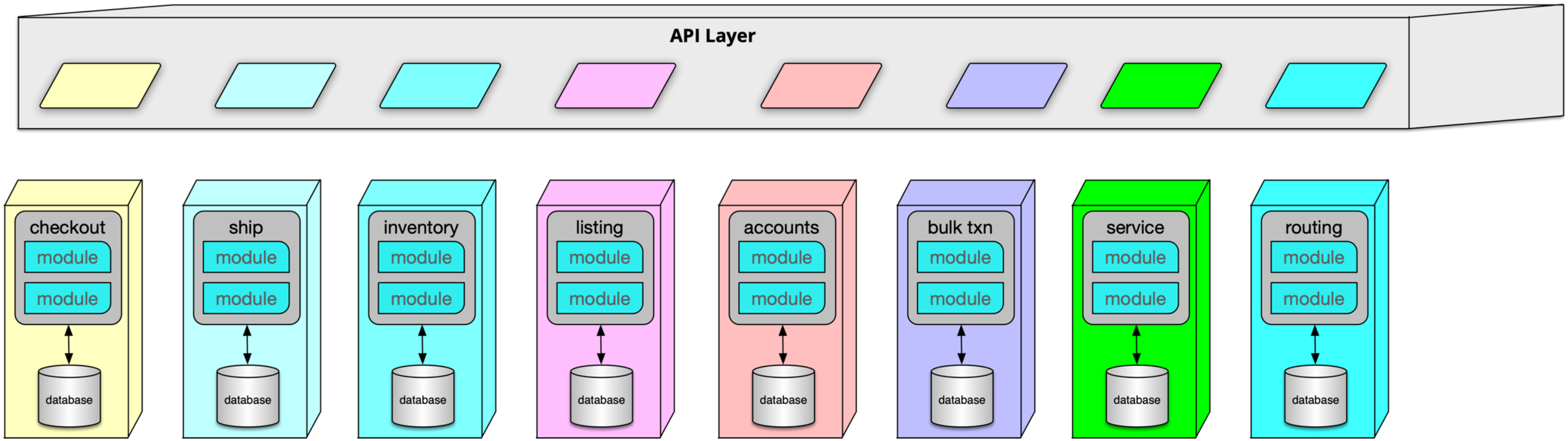


eventual consistency

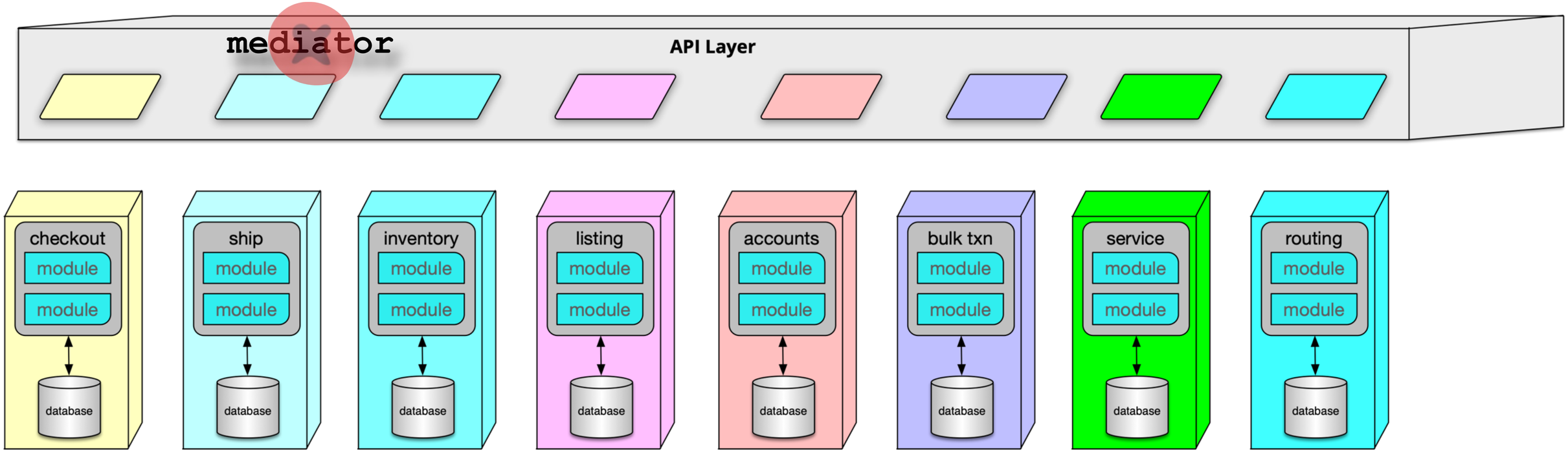


small quantum

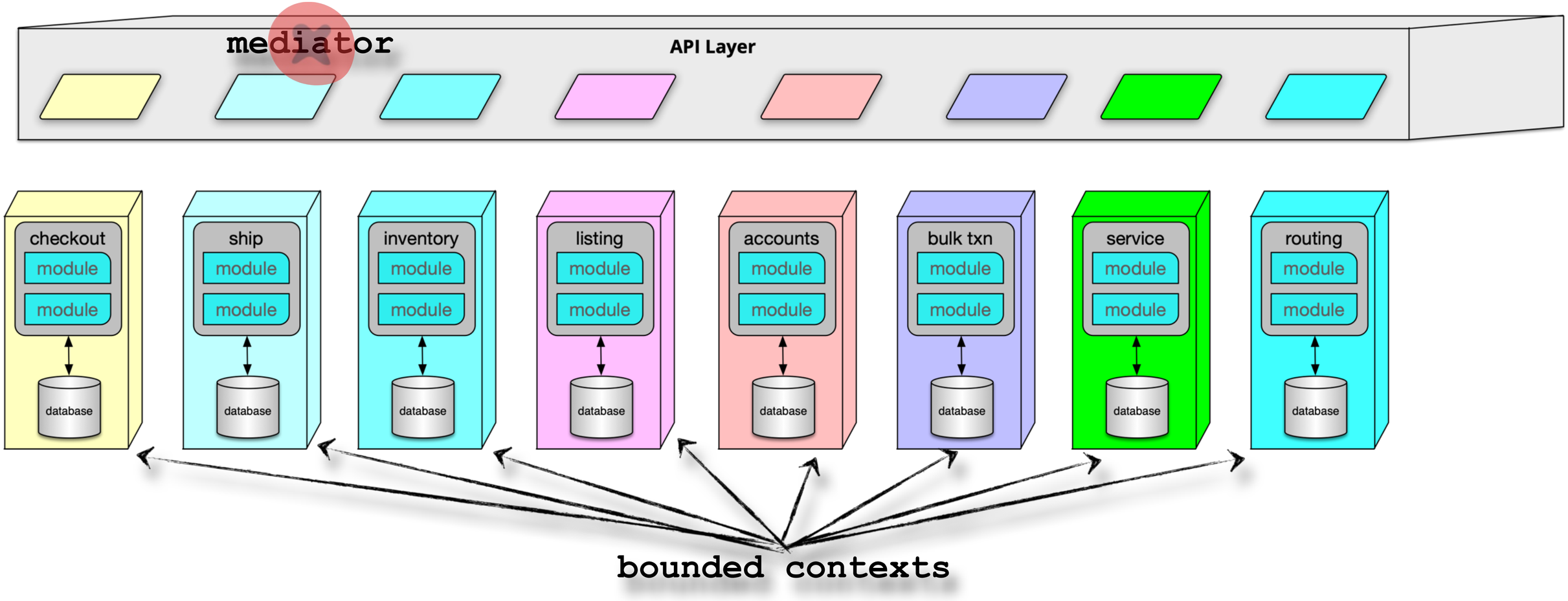
Microservices



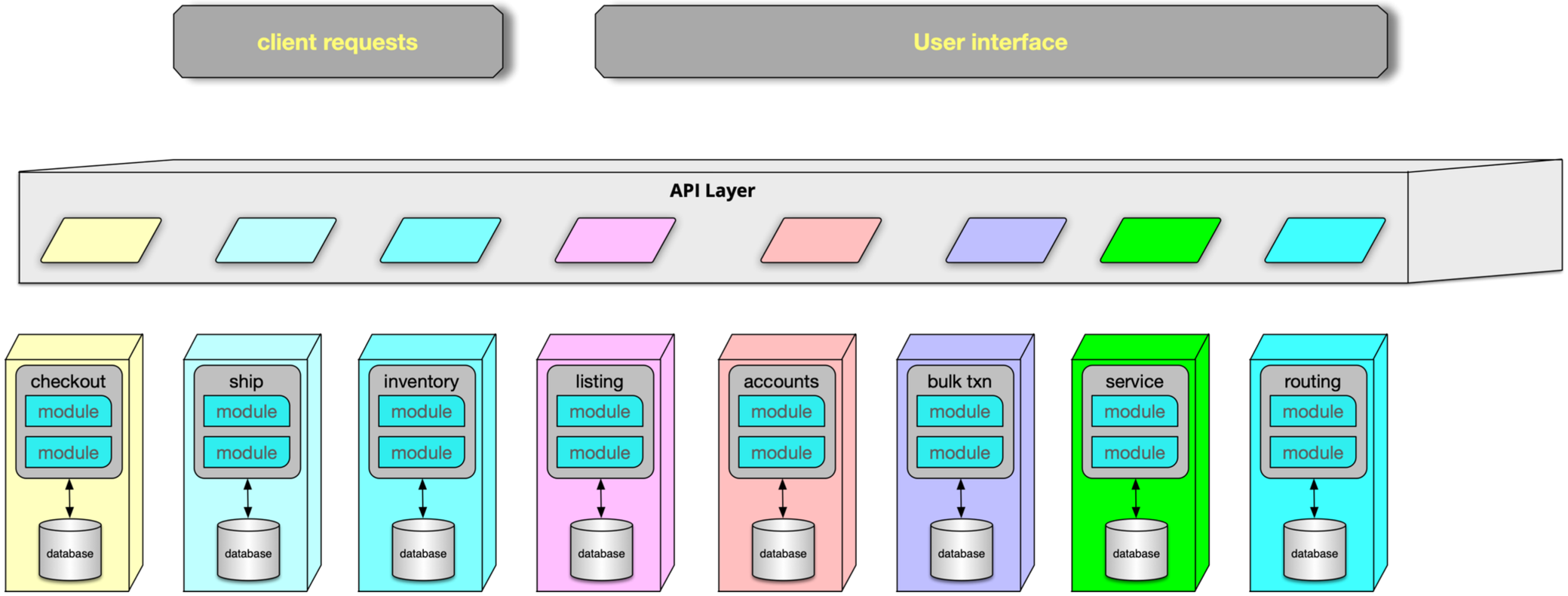
Microservices



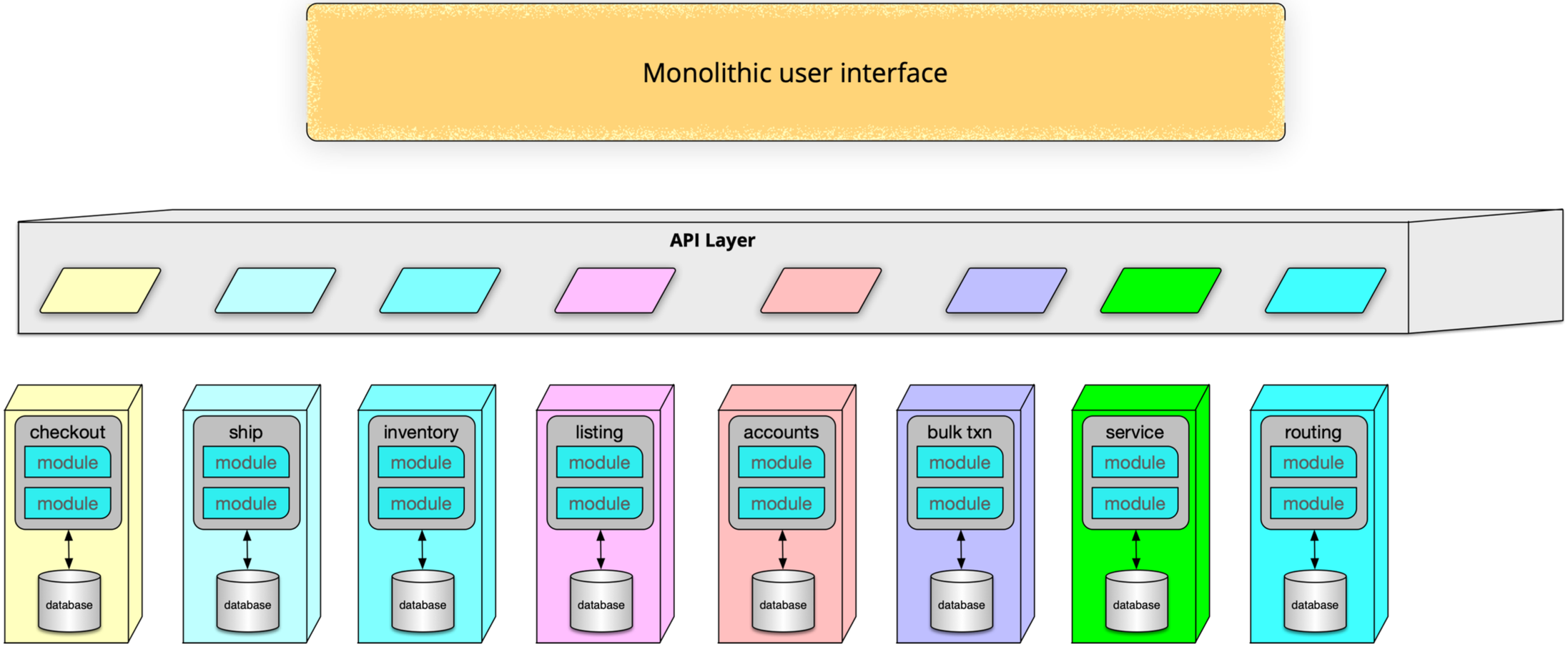
Microservices



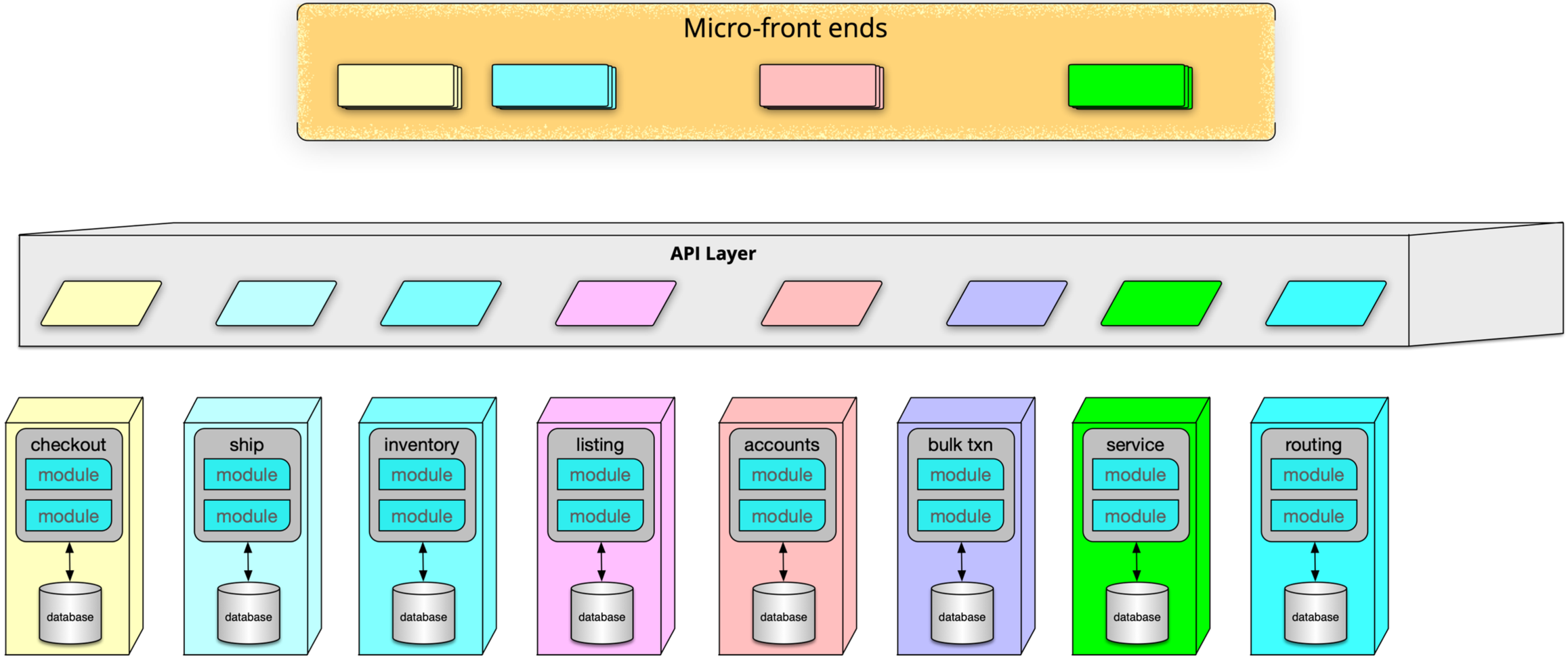
Microservices



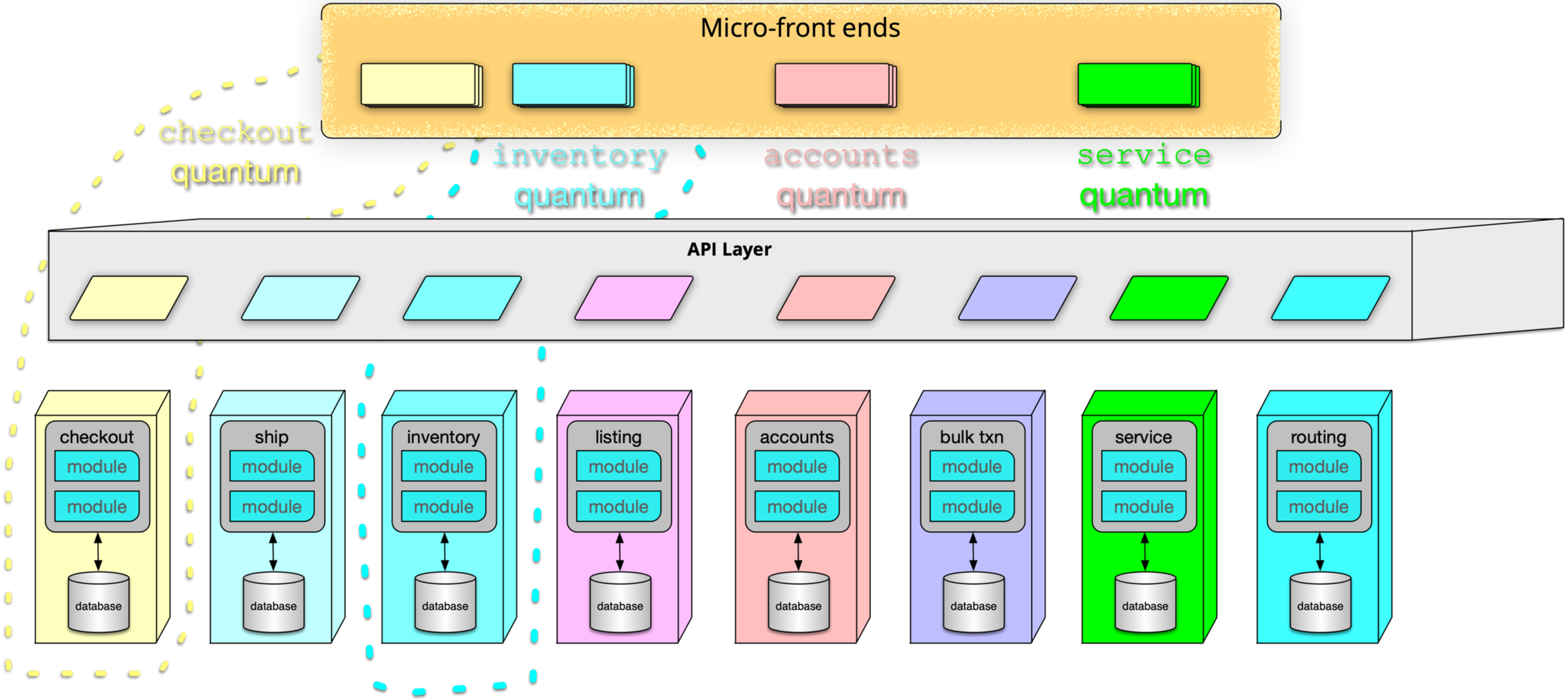
Microservices: Monolithic UI



Microservices

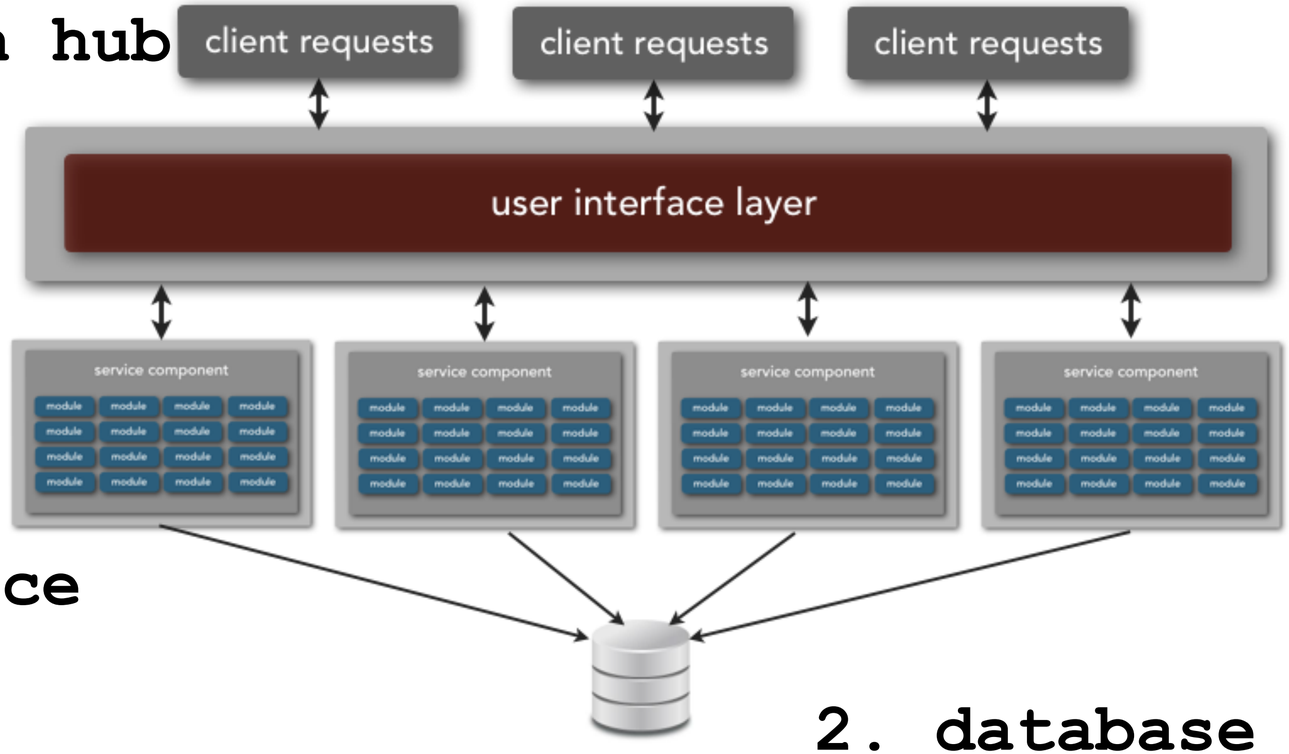


Microservices



Service-based Architectures

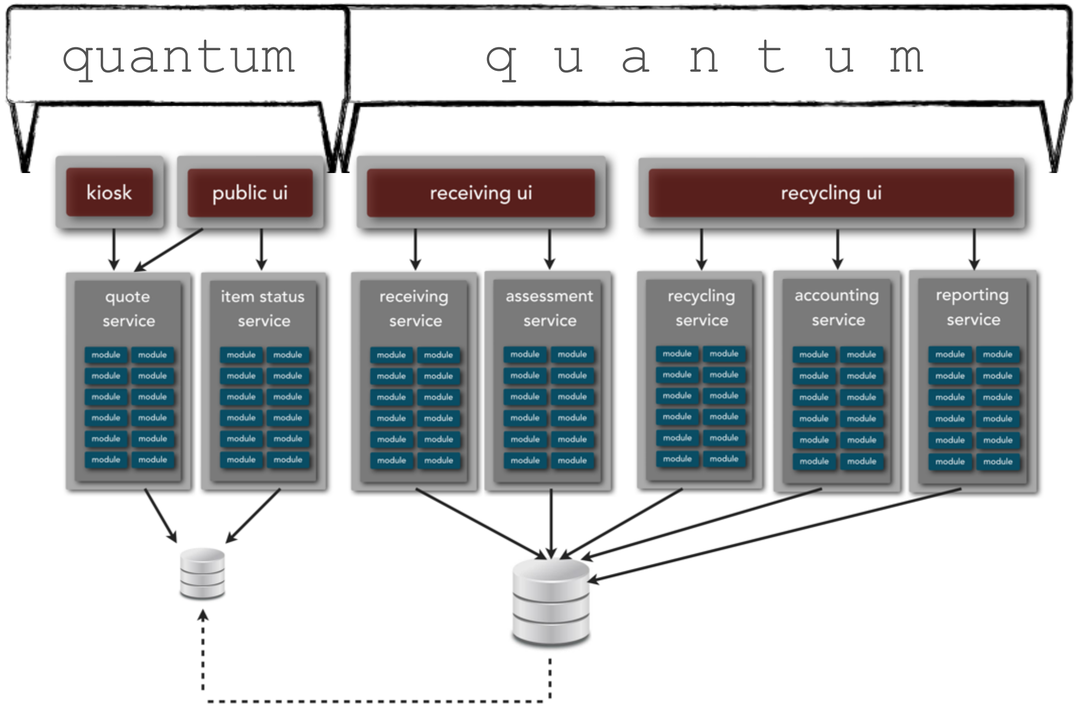
3. use of service bus
   as integration hub



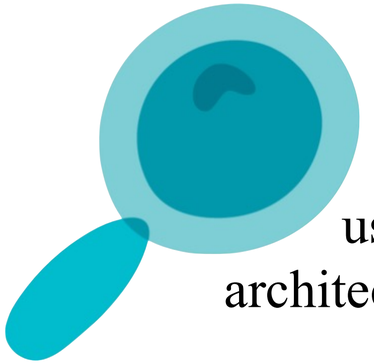
1. Larger service
   granularity

2. database scope

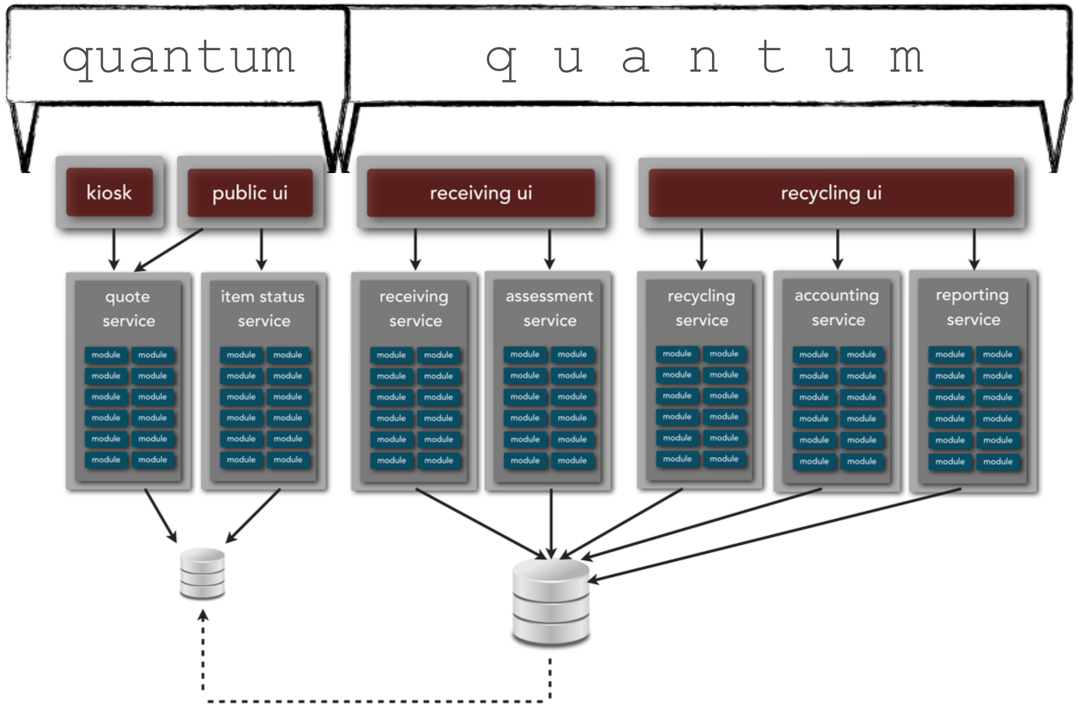
Reducing Quanta Size



Reducing Quanta Size



useful for
architectural analysis

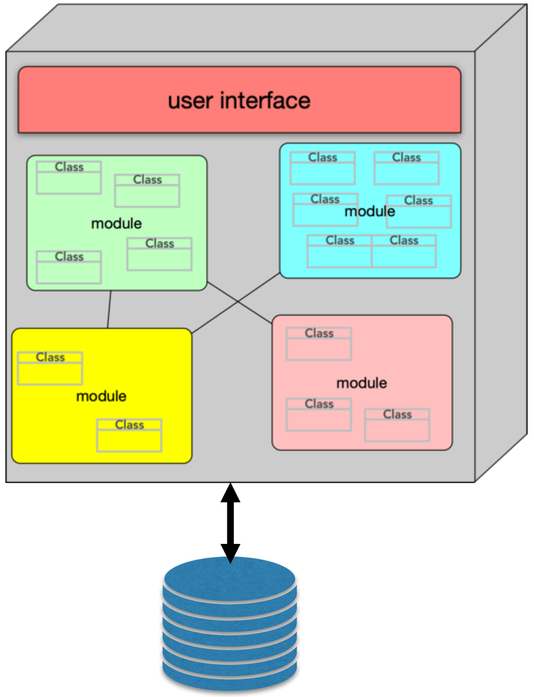


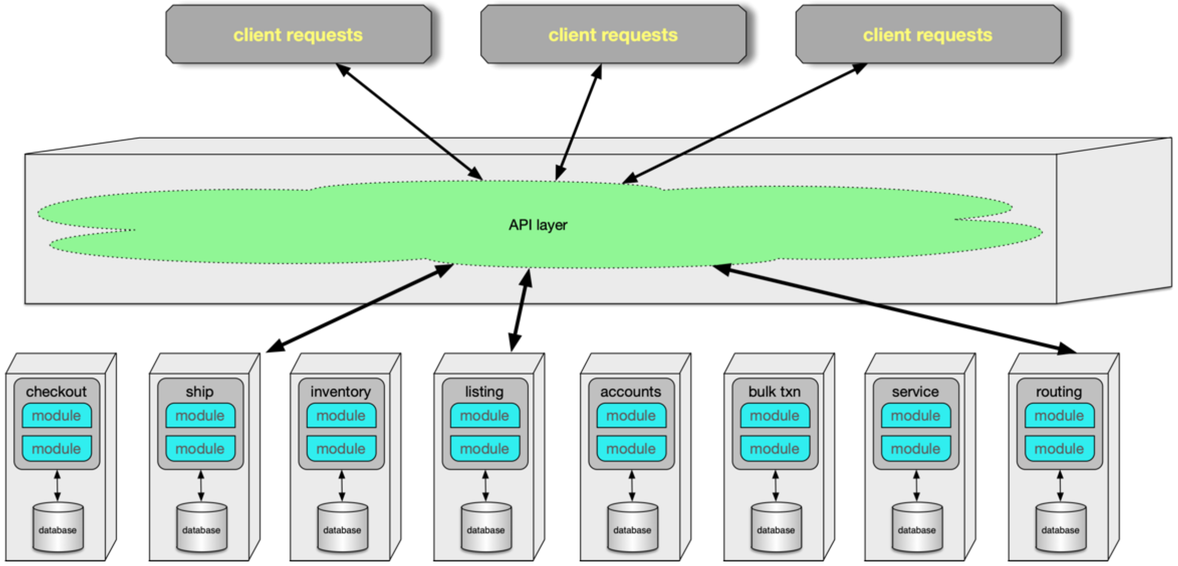
helps analyze
coupling

Smaller Quanta Size

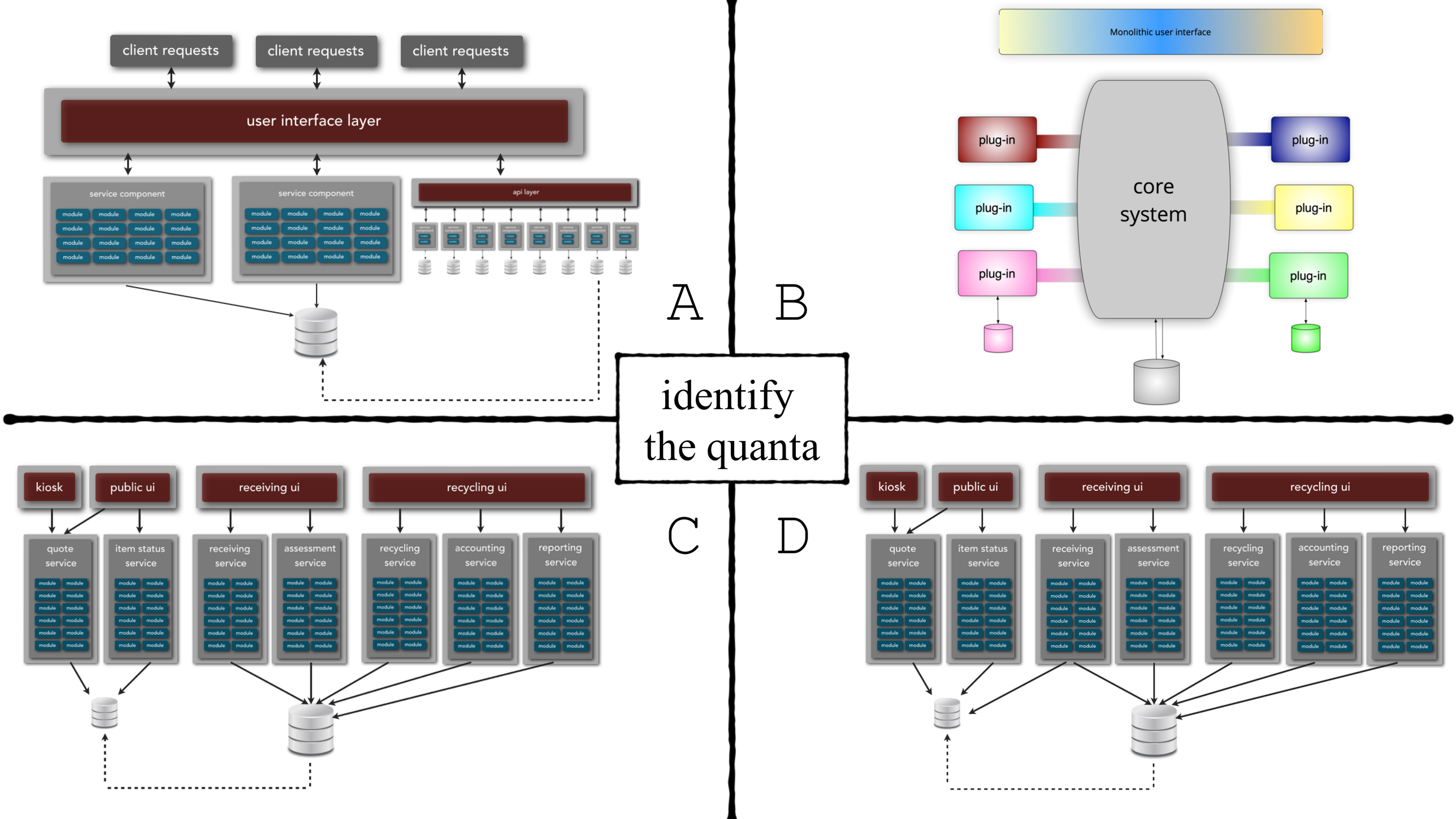
≙

Evolutionary

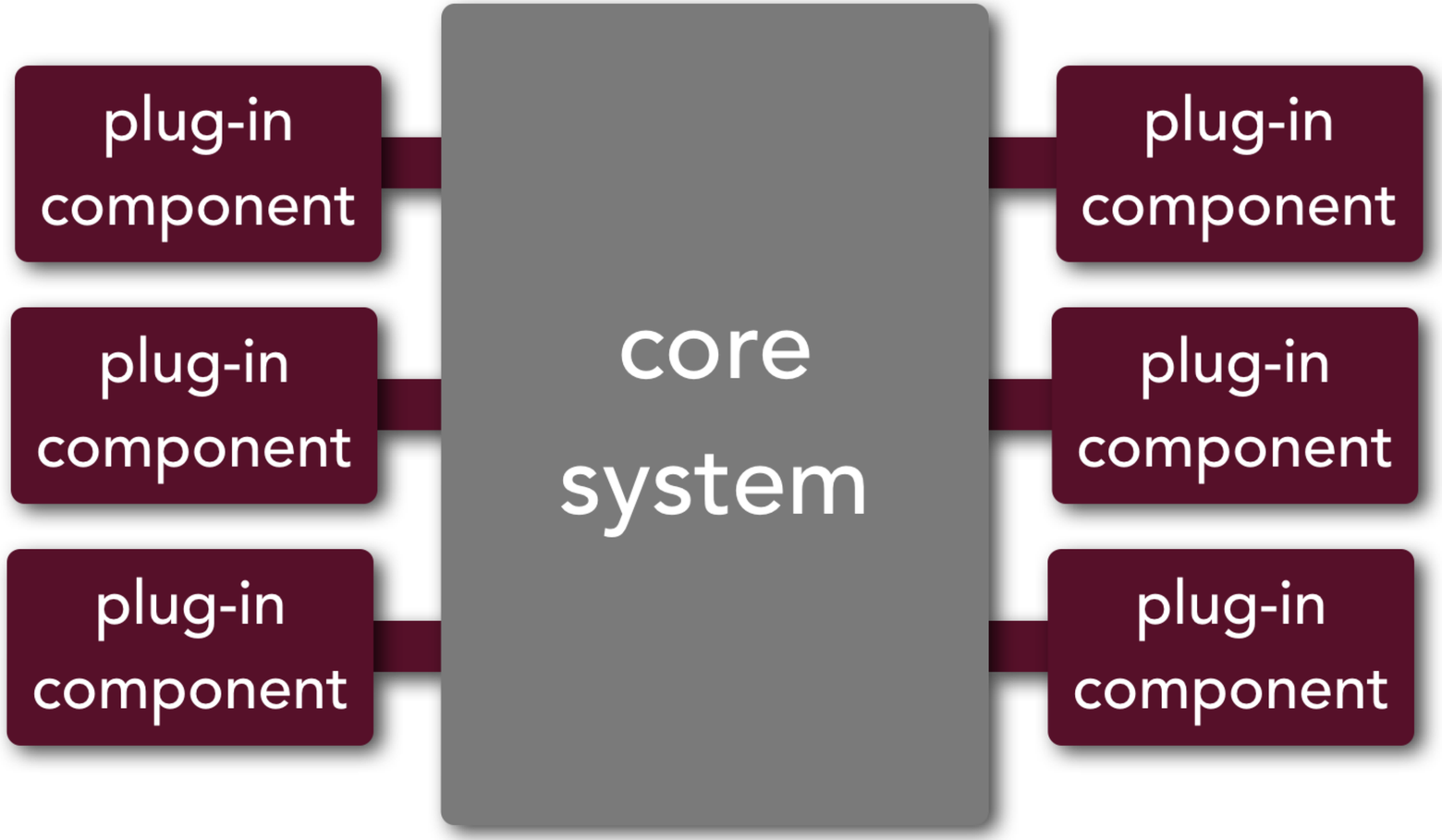




identify the quanta


client requests
client requests
client requests
user interface layer
service component
service component
api layer
module
A
Monolithic user interface
plug-in
plug-in
plug-in
plug-in
plug-in
plug-in
core system
B
identify the quanta
kiosk
public ui
receiving ui
recycling ui
quote service
item status service
receiving service
assessment service
recycling service
accounting service
reporting service
C
D

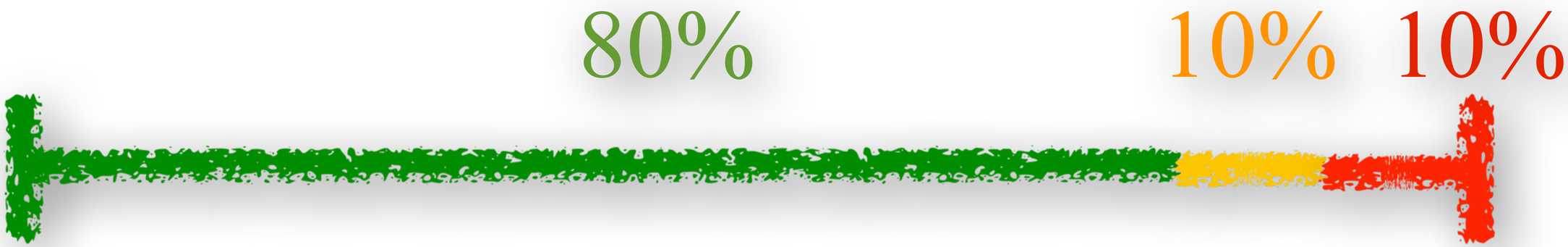
Microkernel Redux



Last 10% Trap

Last 10% Trap

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



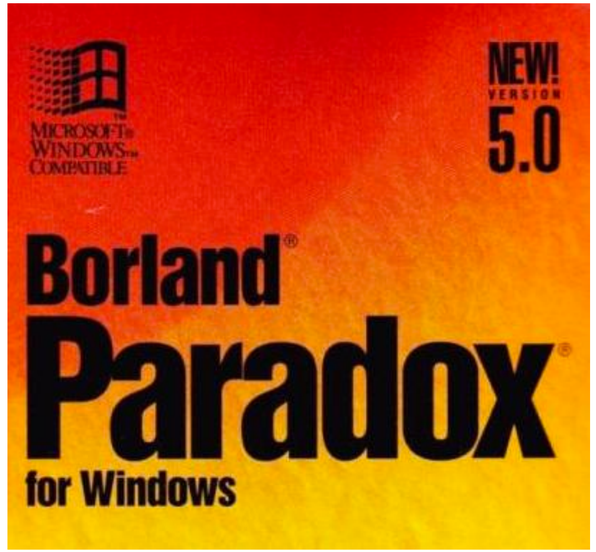
what the user wants

What Happened to the 4GLs?

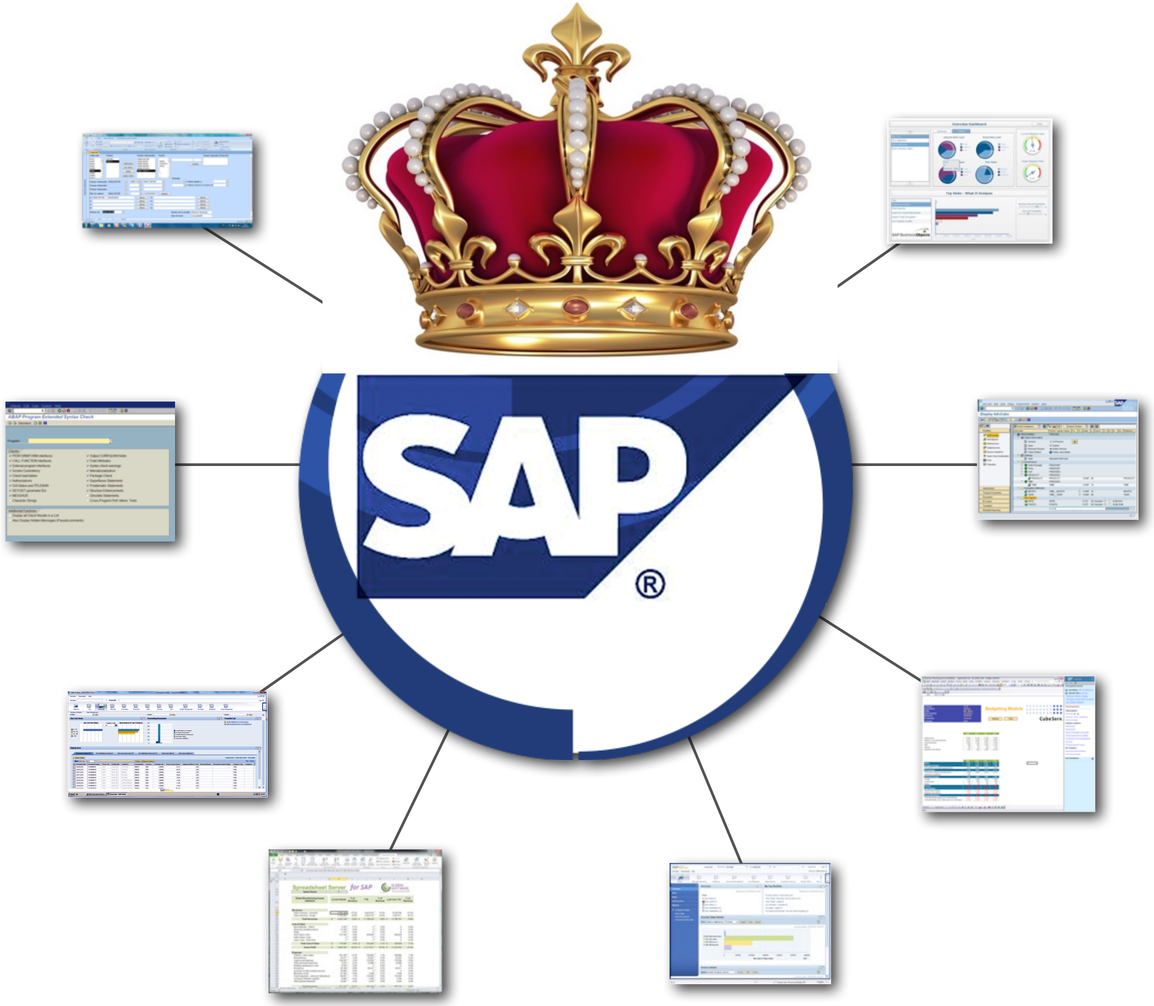
What Happened to the 4GLs?

dBASE™

Microsoft® Visual FoxPro®

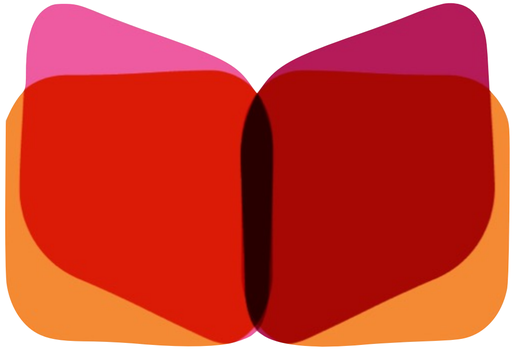


see also: Vendor King

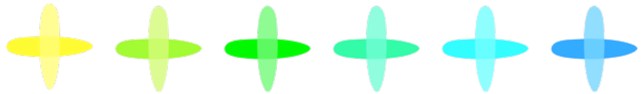


Be careful of the *Last 10% Trap* when choosing tools & frameworks.

Agenda Part 1

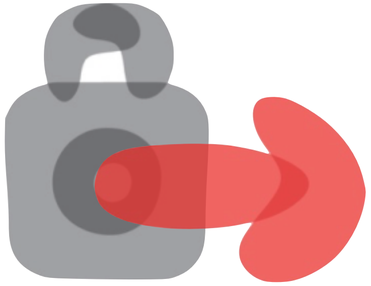


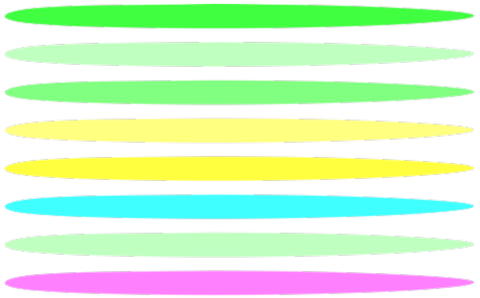
Definition



Incremental Change

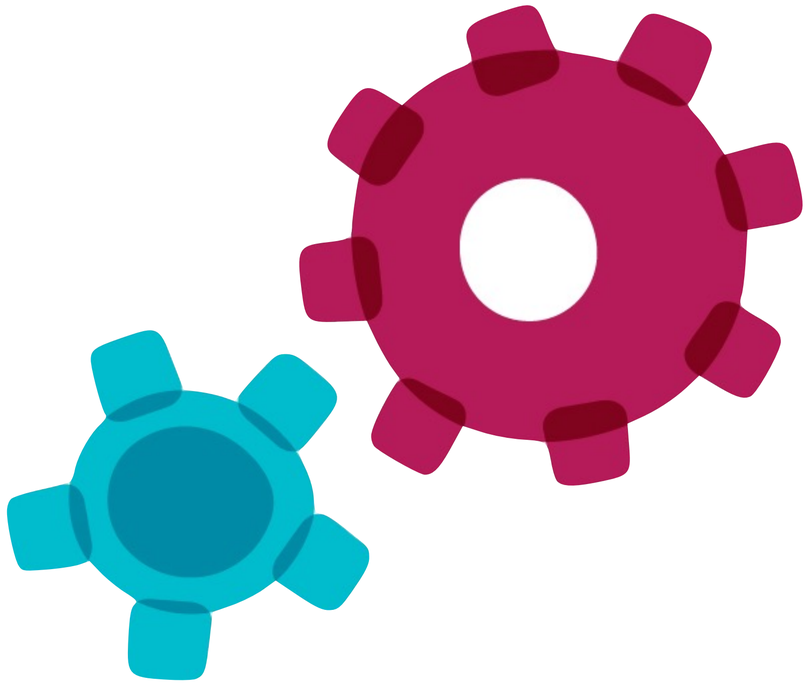
Guided Change via Fitness Functions





Structuring architecture for evolution

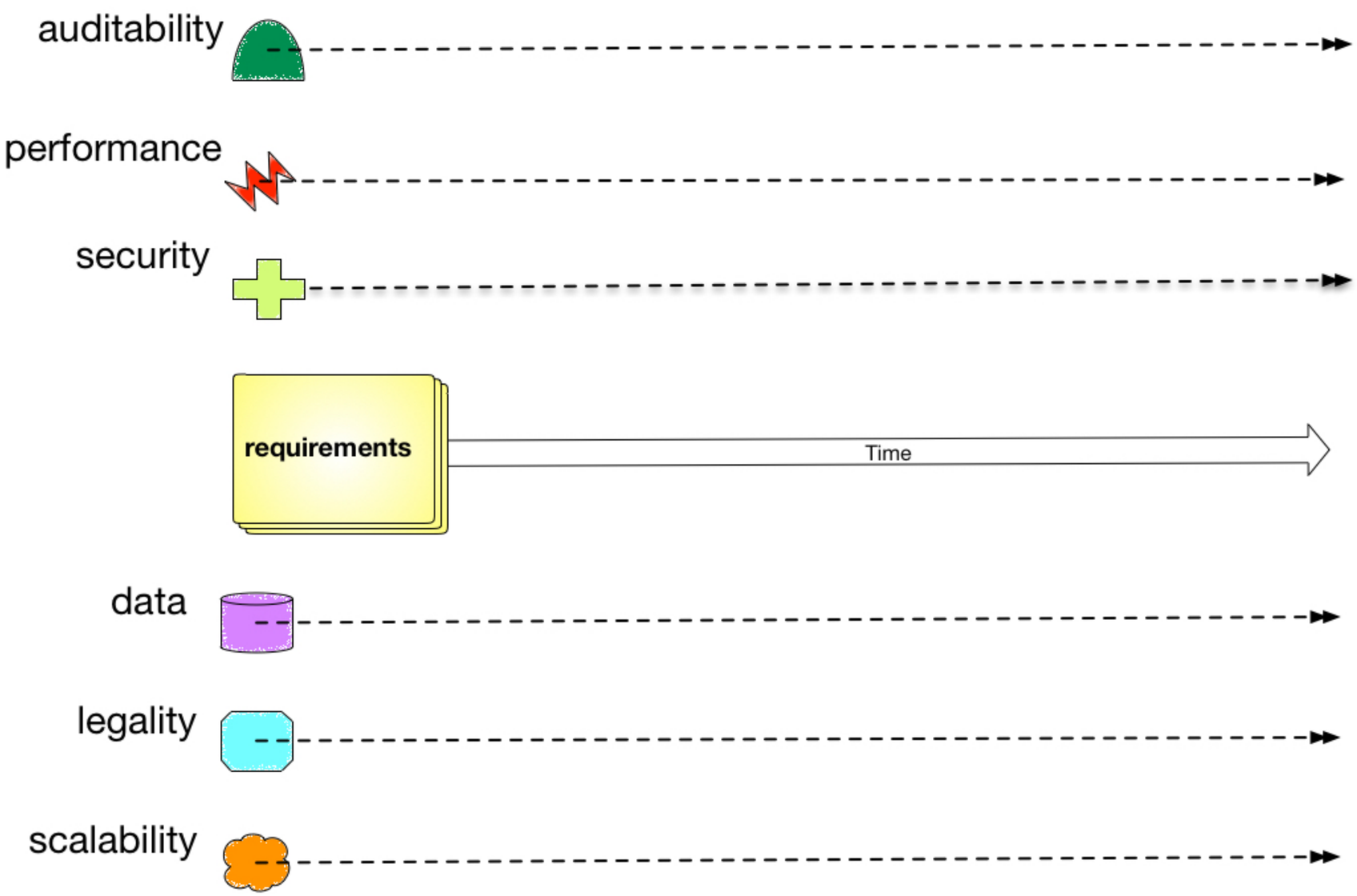
Putting evolutionary architecture into Practice



Mechanics

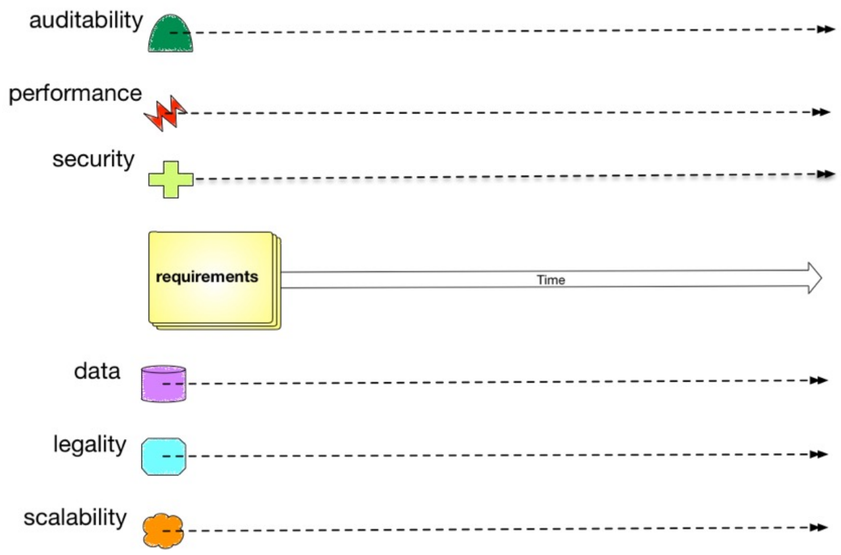
1. Identify dimensions

1. Identify dimensions

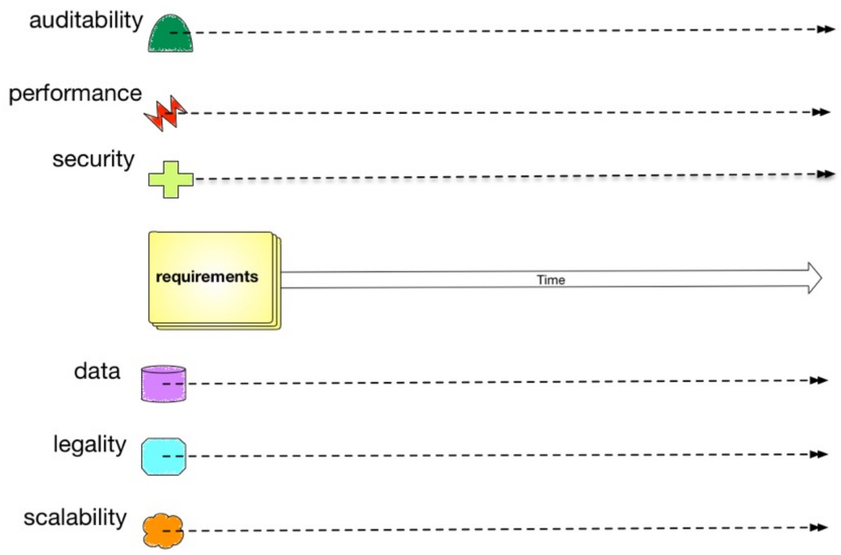


1. Identify dimensions

prioritization



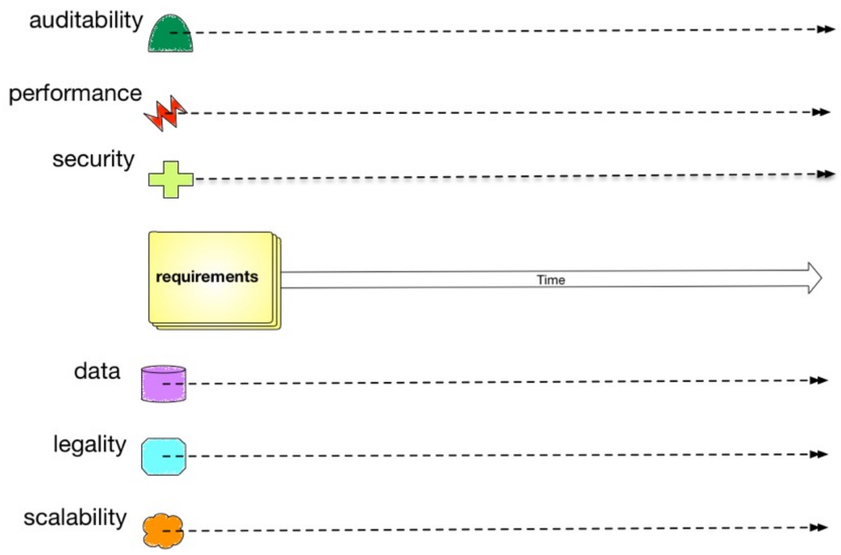
1. Identify dimensions



prioritization

cost to implement & maintain

1. Identify dimensions



prioritization

cost to implement & maintain

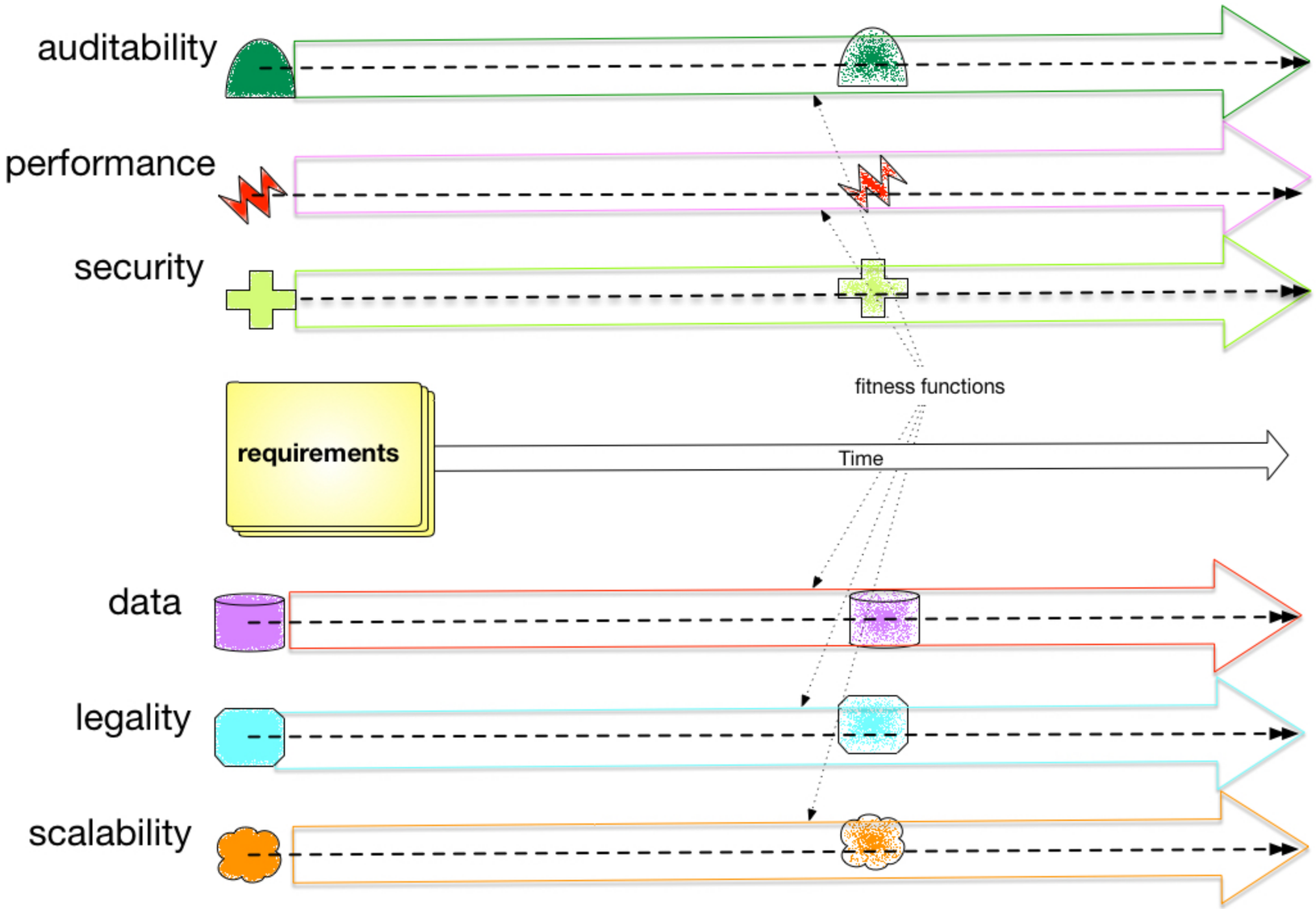
change variable “fix it” cost to constant maintenance cost

Mechanics

1. Identify dimensions

2. Define fitness function(s)

2. Define Fitness Function(s)



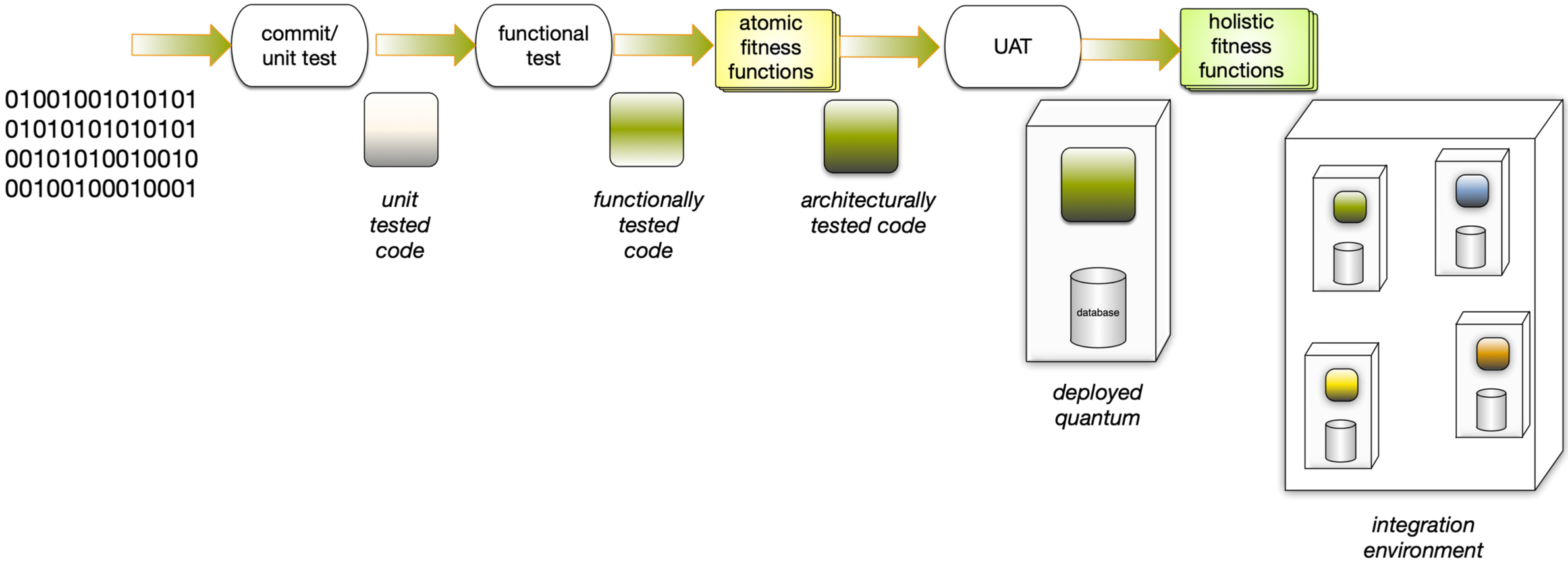
Mechanics

1. Identify dimensions

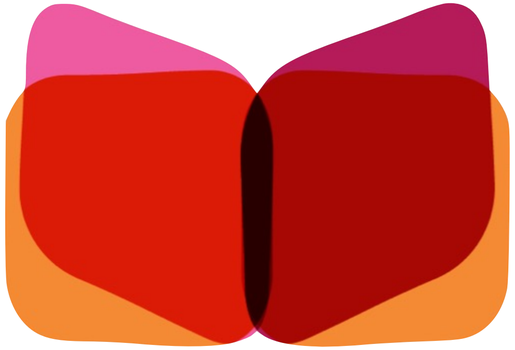
2. Define fitness function(s)

3. Use deployment pipelines and/or continuous fitness functions

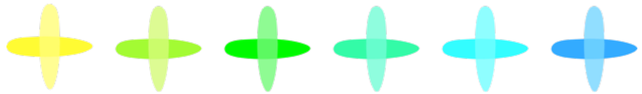
3. Use deployment pipelines and/or continuous fitness functions



Agenda Part 1

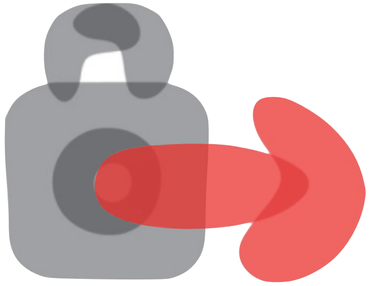


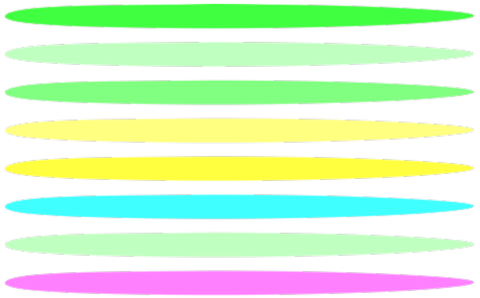
Definition



Incremental Change

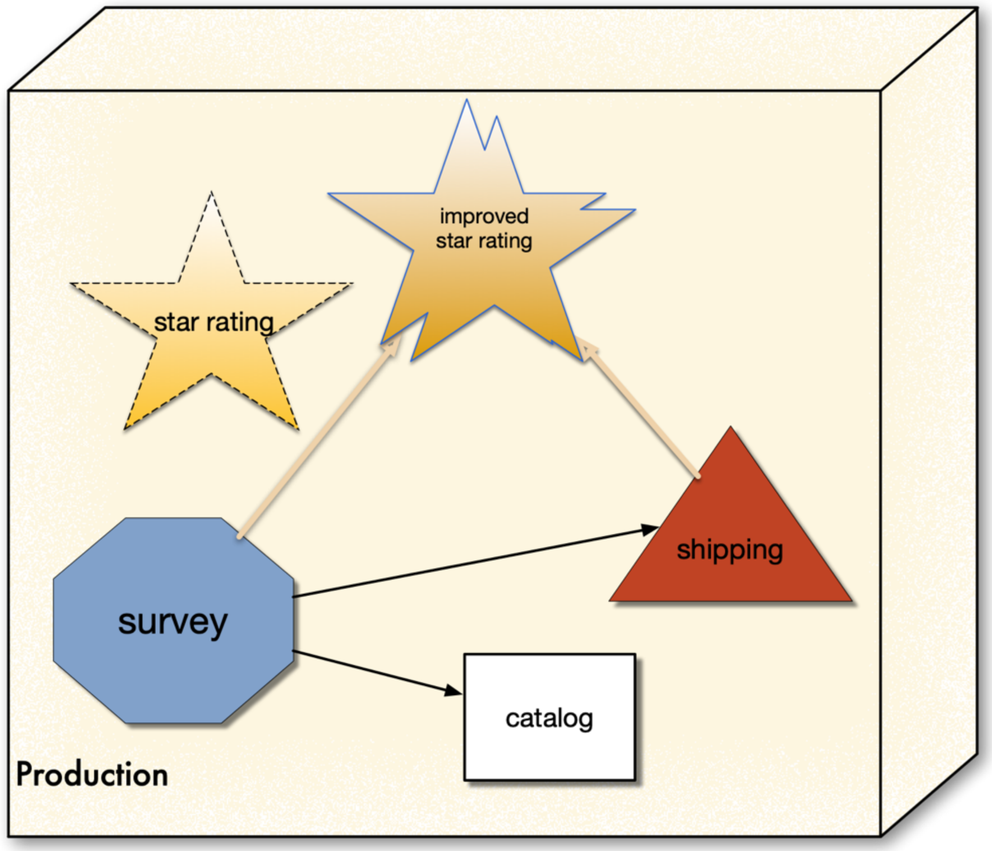
Guided Change via Fitness Functions

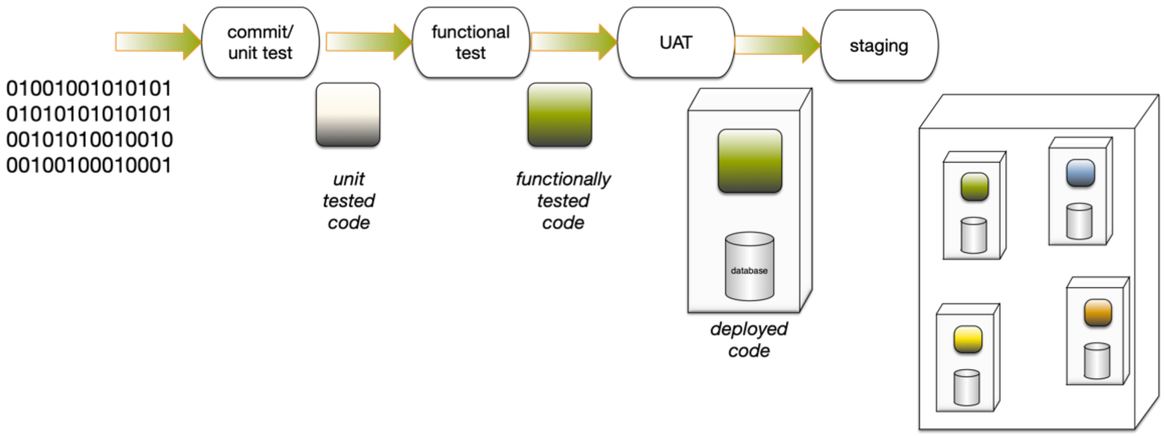




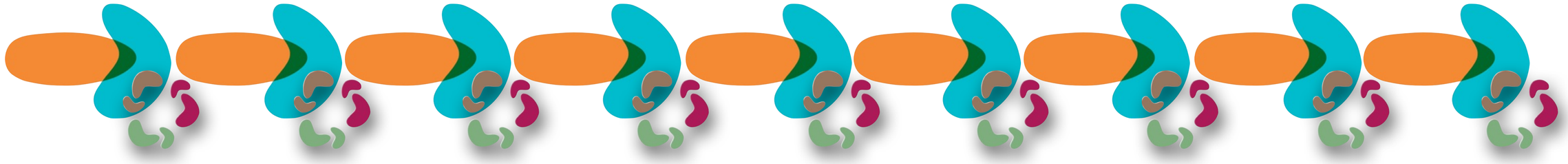
Structuring architecture for evolution

incremental

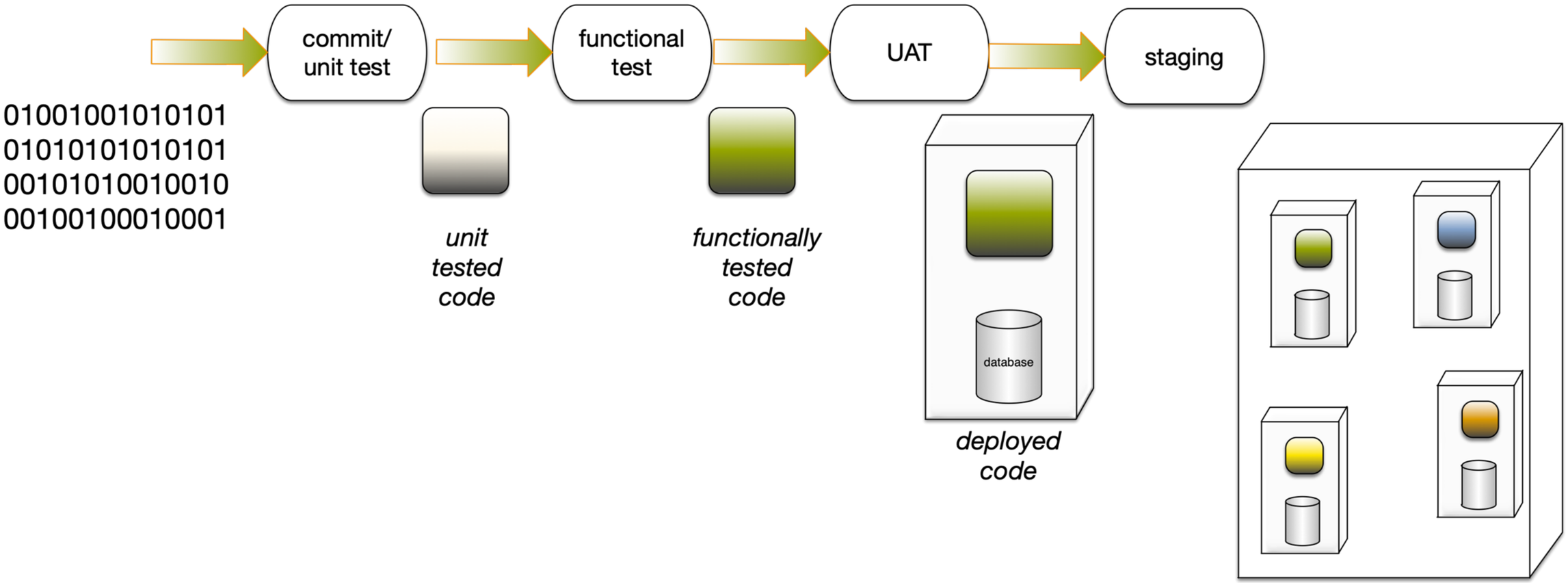




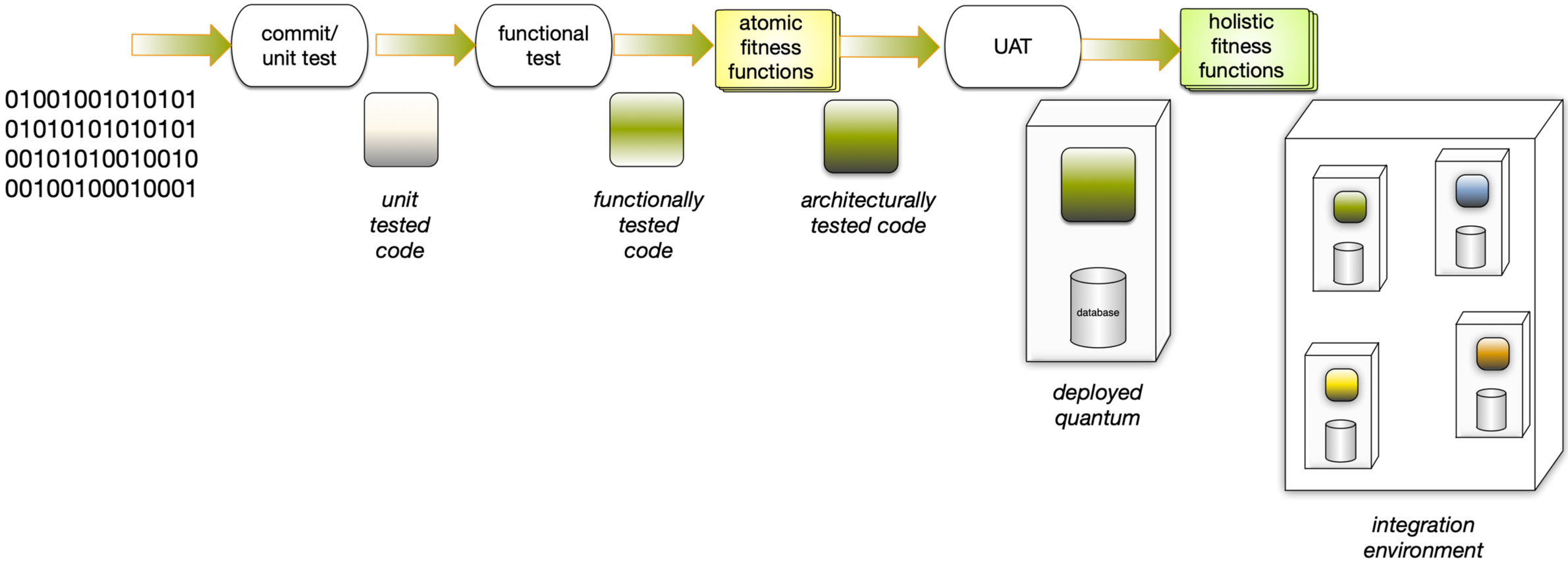
Deployment Pipelines



Deployment Pipeline

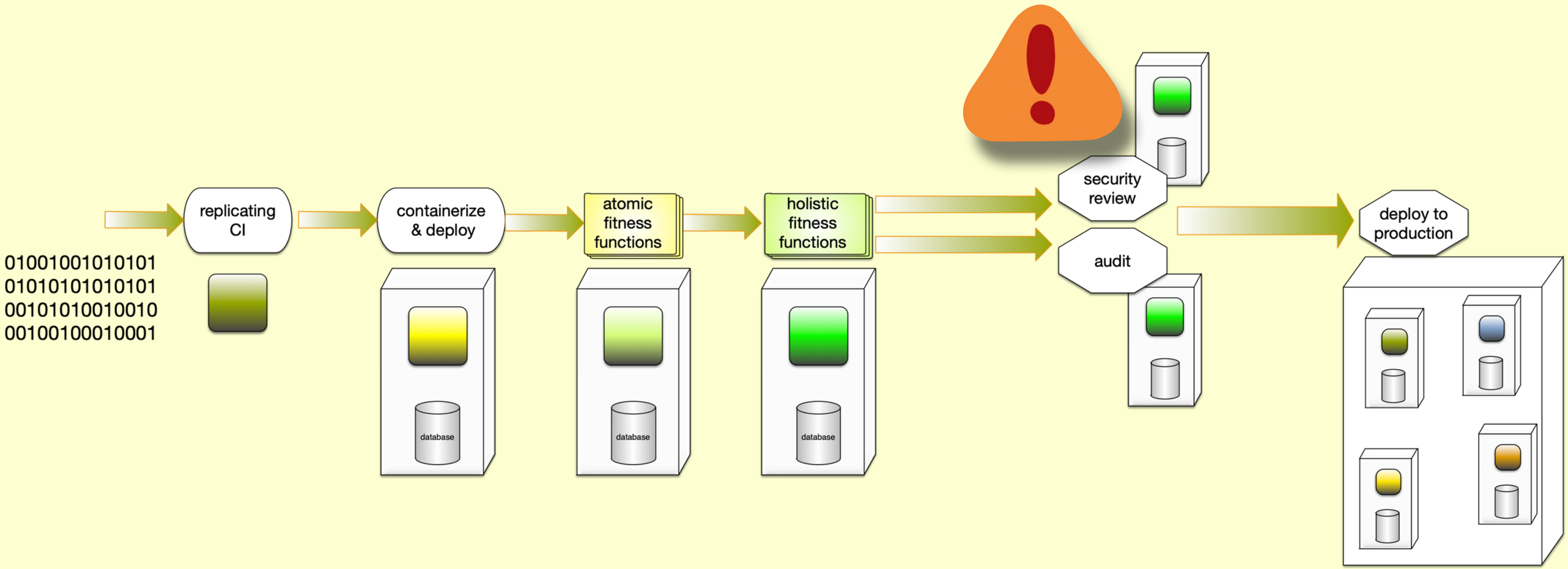


Deployment Pipeline + Fitness Functions



Penultima⬆⊠e

Deployment Pipeline

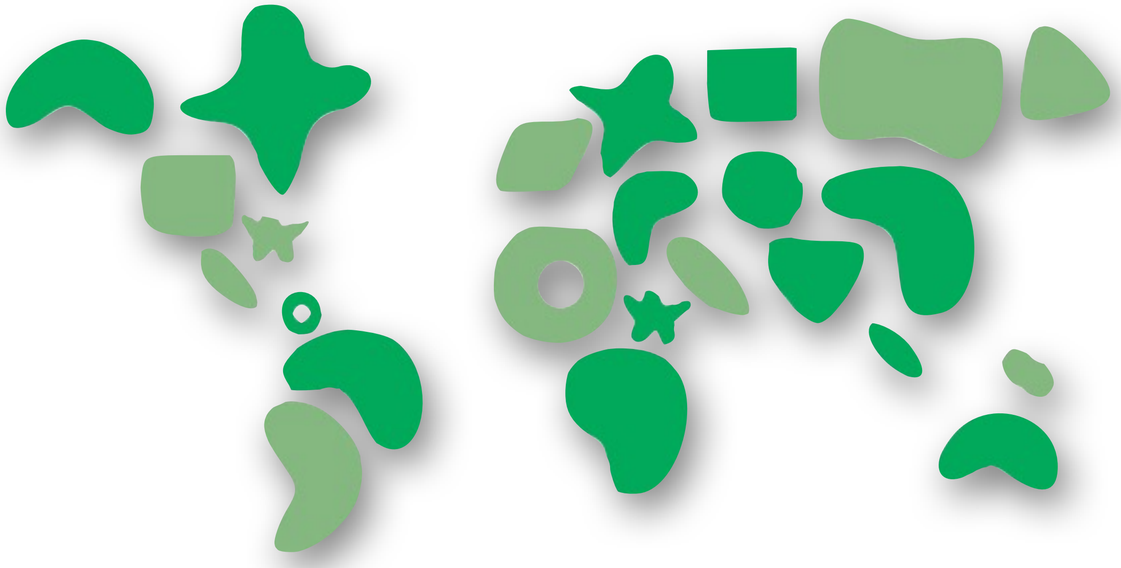


Mechanics

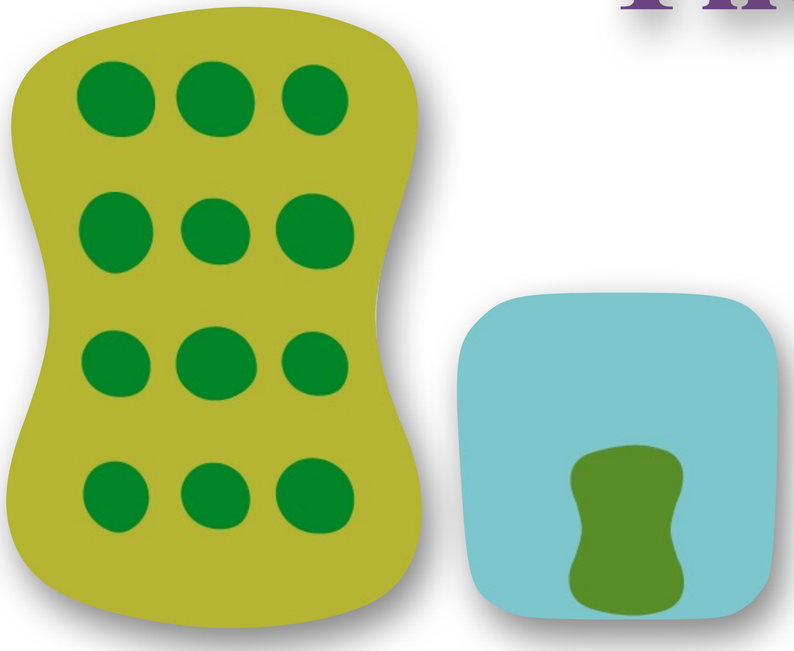
1. Identify dimensions

2. Define fitness function(s)

3. Use deployment pipelines and/or continuous fitness functions



The Business Case



barryoreilly.com

BARRY O'REILLY

experimental by design, free thinker, curious, i use a lot of post-its

PRESENTATIONS LEAN ENTERPRISE CONSULTING ABOUT HOME

OCT 21 2013
10 COMMENTS

AGILE, CONTINUOUS DELIVERY, INNOVATION, LEAN, LEAN ENTERPRISE

LINK

HOW TO IMPLEMENT HYPOTHESIS-DRIVEN DEVELOPMENT →

Remember back to the time when we were in high school science class. Our teachers had a framework for helping us learn – an experimental approach based on the best available evidence at hand. We were asked to make observations about the world around us, then attempt to form an explanation or hypothesis to explain what we had observed. We then tested this hypothesis by predicting an outcome based on our theory that would be achieved in a controlled experiment – if the outcome was achieved, we had proven our theory to be correct.

We could then apply this learning to inform and test other hypotheses by constructing more sophisticated experiments, and tuning, evolving or abandoning any hypothesis as we made further observations from the results we achieved.

Experimentation is the foundation of the scientific method, which is a systematic means of exploring the world around us. Although some experiments take place in laboratories, it is possible to perform an experiment anywhere, at any time, even in software development.

Practicing Hypothesis-Driven Development*[1]* is thinking about the development of new ideas, products and services – even organizational change – as a series of experiments to determine whether an expected outcome will be achieved. The process is iterated upon until a desirable outcome is obtained or the idea is determined to be not viable.

We need to change our mindset to view our proposed solution to a problem statement as a hypothesis, especially in new product or service development – the market we are targeting, how a business model will work, how code will execute and even how the customer will use it.

We do not do projects anymore, only experiments. Customer discovery and Lean Startup strategies are designed to test assumptions about customers. Quality Assurance is testing system behavior against defined specifications. The experimental principle also applies in Test-Driven Development – we write the test first, then use the test to validate that our code is correct, and succeed if the code passes the test. Ultimately, product or service development is a process to test a hypothesis about system behaviour in the environment or market it is developed for.

The key outcome of an experimental approach is measurable evidence and learning. Learning is the information we have gained from conducting the experiment. Did what we expect to occur actually happen? If not, what did and how does that inform what we should do next?

In order to learn we need use the scientific method for investigating phenomena, acquiring new knowledge, and correcting and integrating previous knowledge back into our thinking.

As the software development industry continues to mature, we now have an opportunity to leverage improved capabilities such as Continuous Design and Delivery to maximize our potential to learn quickly what works and what does not. By taking an experimental approach to information discovery, we can more rapidly test our solutions against the problems we have identified in the products or services we are attempting to build. With the goal to optimize our effectiveness of solving the right problems, over simply becoming a feature factory by continually building solutions.

The steps of the scientific method are to:

- Make observations
- Formulate a hypothesis
- Design an experiment to test the hypothesis

Follow

barryoreilly.com/2013/10/21/how-to-implement-hypothesis-driven-development/

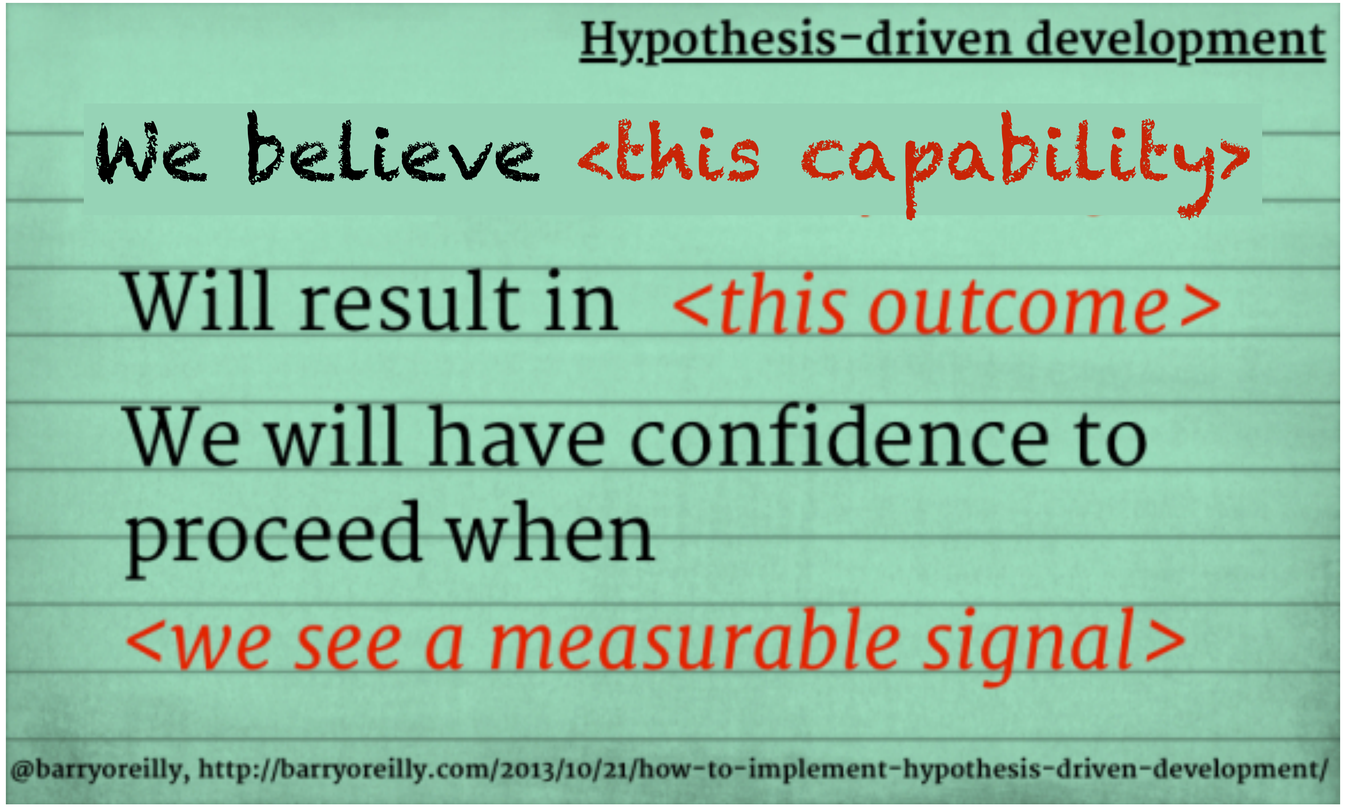
Requirements

As A.... <role>
I Want... <goal/desire>
So That... <receive benefit>

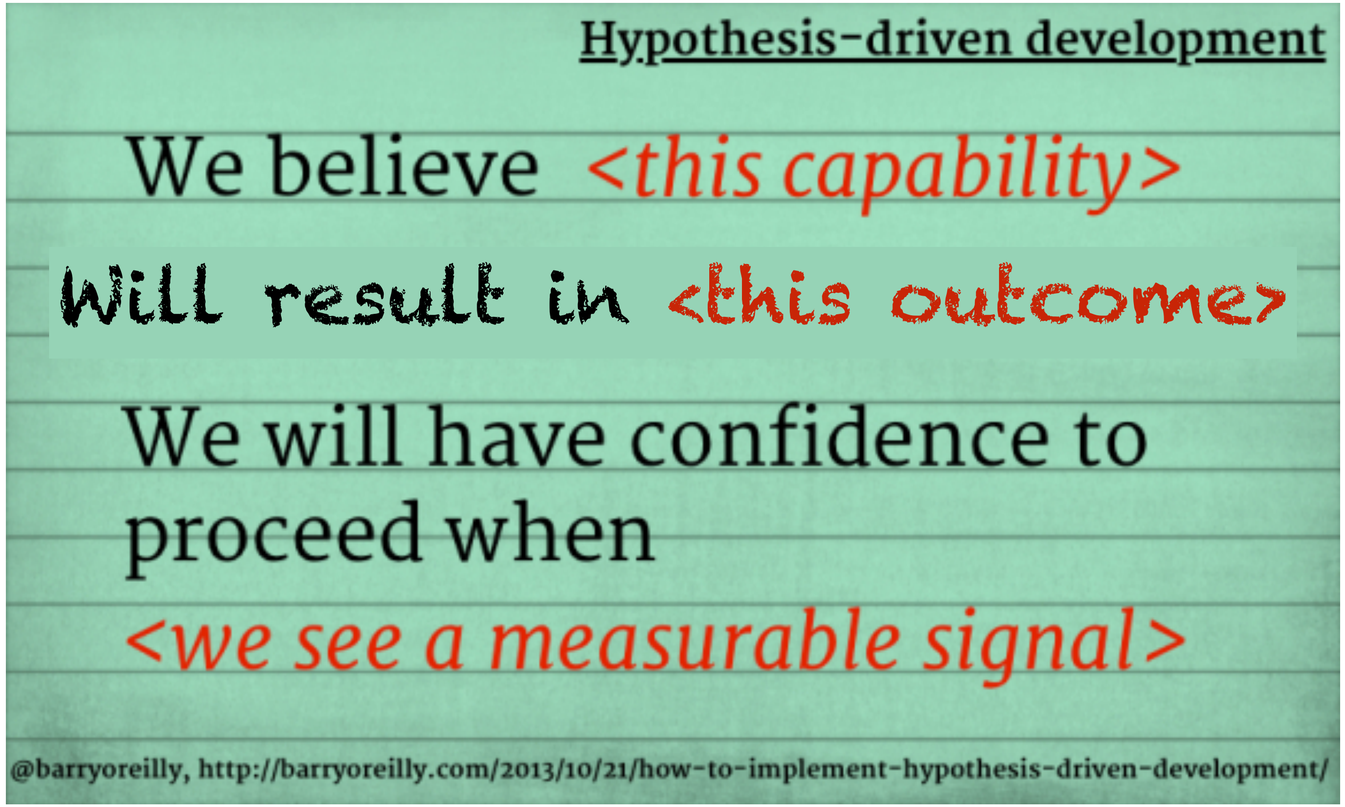
Behavior-Driven Development

In Order To... <receive benefit>
As A... <role>
I Want... <goal/desire>

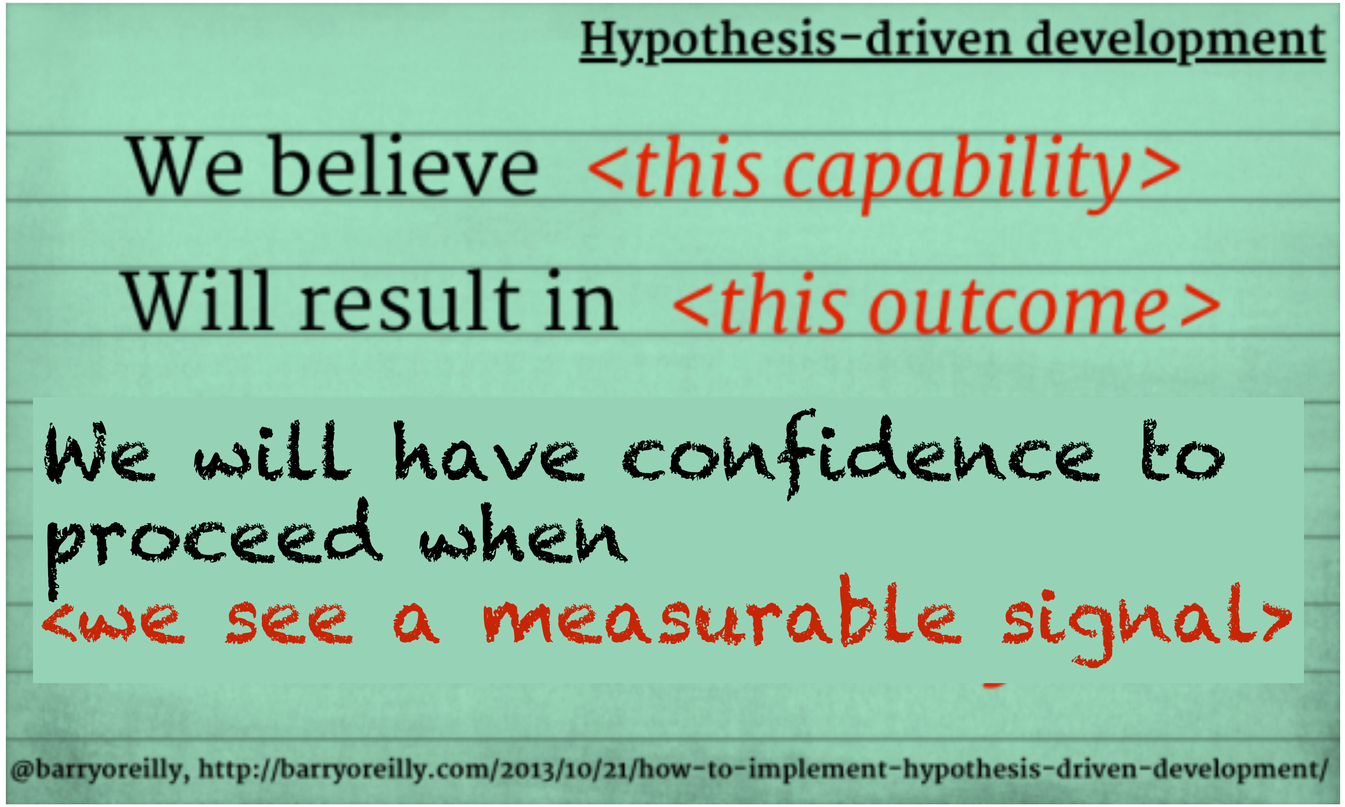
Hypothesis-driven Development



Hypothesis-driven Development



Hypothesis-driven Development



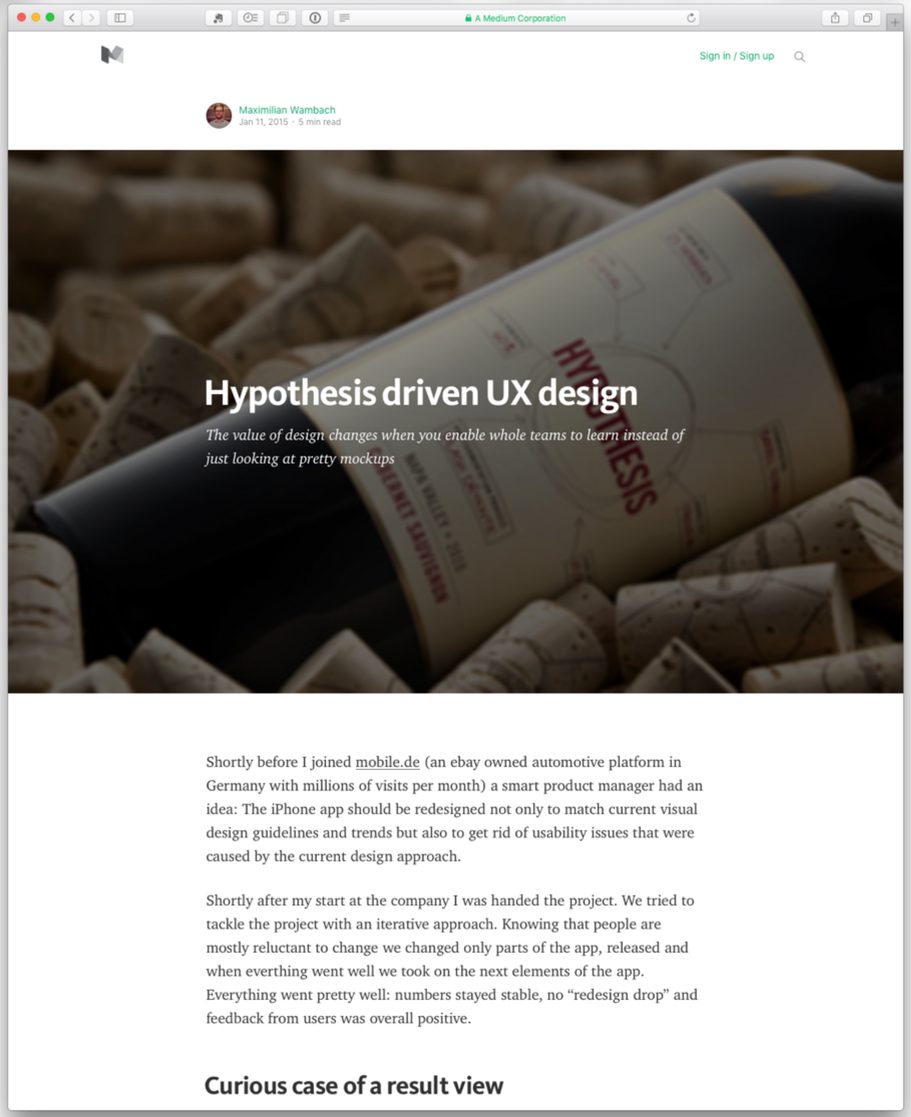
Business Story

We Believe That increasing the size of hotel images on the booking page

Will Result In improved customer engagement and conversion

We Will Have Confidence To Proceed When we see a 5% increase in customers who review hotel images who then proceed to book in 48 hours.

Hypothesis Driven UX



A Medium Corporation

Sign in / Sign up

Maximilian Wambach
Jan 11, 2015 · 5 min read

Hypothesis driven UX design

The value of design changes when you enable whole teams to learn instead of just looking at pretty mockups

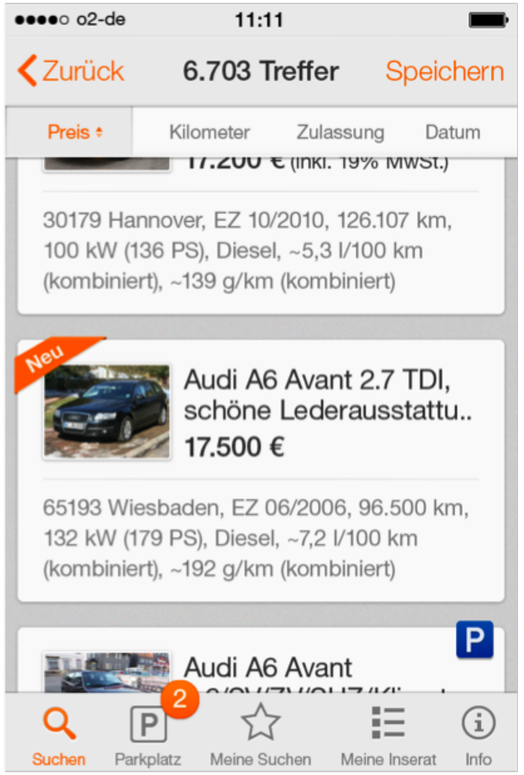
Shortly before I joined mobile.de (an ebay owned automotive platform in Germany with millions of visits per month) a smart product manager had an idea: The iPhone app should be redesigned not only to match current visual design guidelines and trends but also to get rid of usability issues that were caused by the current design approach.

Shortly after my start at the company I was handed the project. We tried to tackle the project with an iterative approach. Knowing that people are mostly reluctant to change we changed only parts of the app, released and when everthing went well we took on the next elements of the app. Everything went pretty well: numbers stayed stable, no “redesign drop” and feedback from users was overall positive.

Curious case of a result view

https://medium.com/@mwambach1/hypotheses-driven-ux-design-c75fbf3ce7cc#.gk3dpip81

Hypothesis Driven UX



Numbers went down by too many %

Three Hypotheses

More Listings

If we provide more listings on the screen then we can provide better comparability and offer more diversity on our platform because users like to compare a lot of listings on the result page

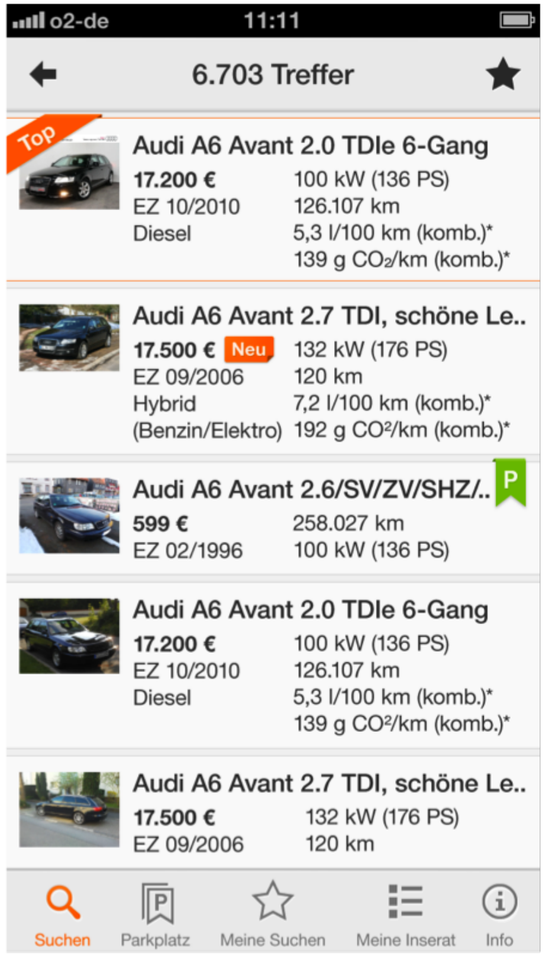
Better Structure

If we provide more structure to our listings then we achieve a better scanability because the user is able to scan the relevant information quicker

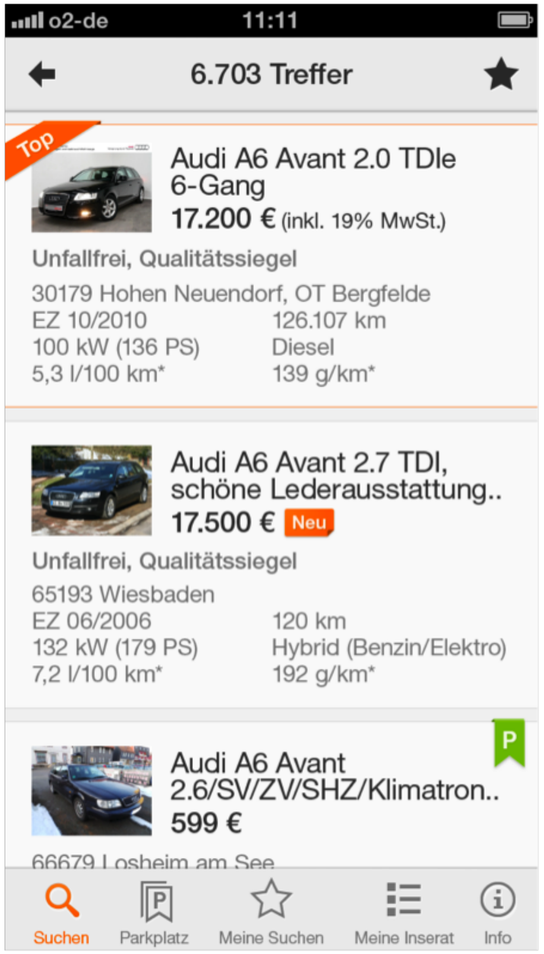
Better Prioritization

If we prioritize information according to user needs then we achieve better guidance because the user can see all relevant information at a glance

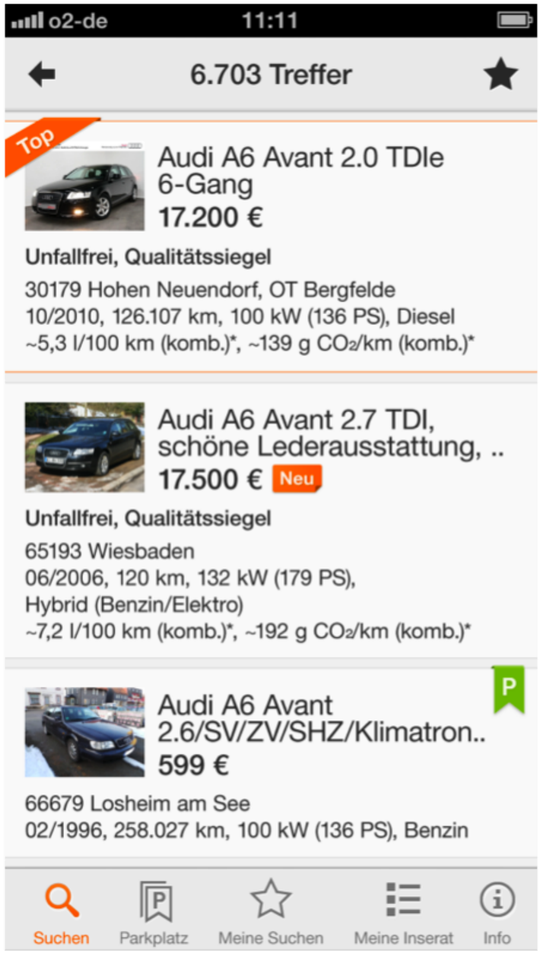
Experiments to Perform



More Listings

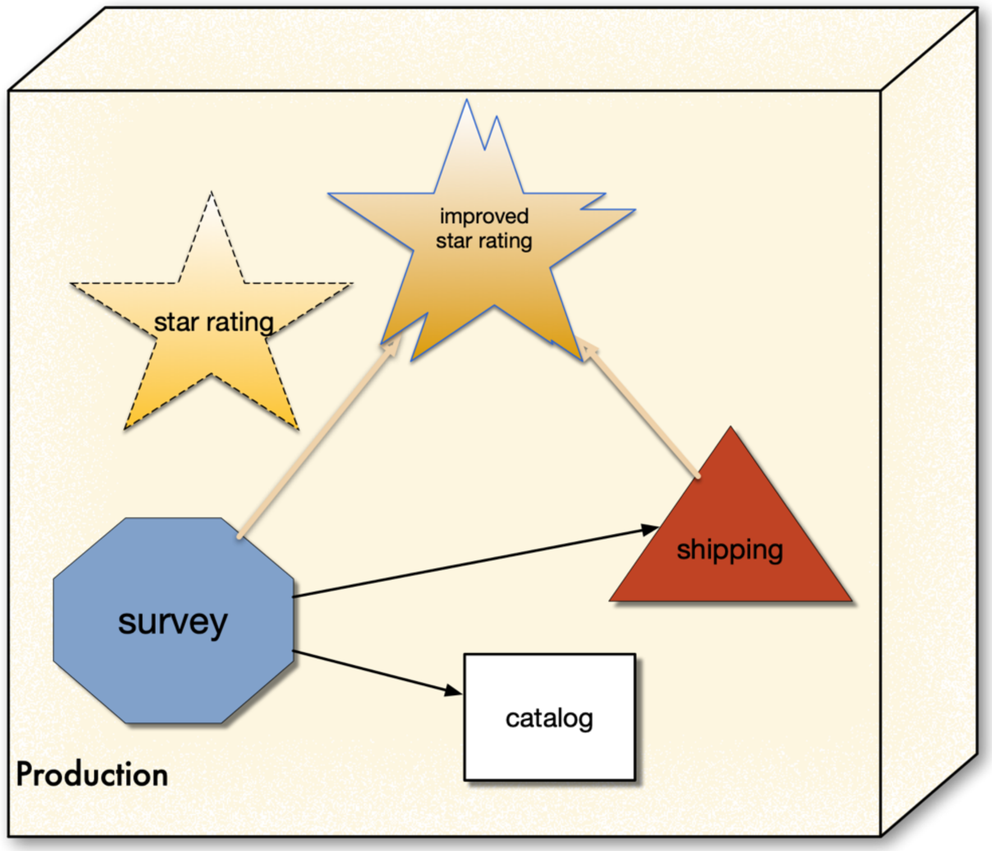


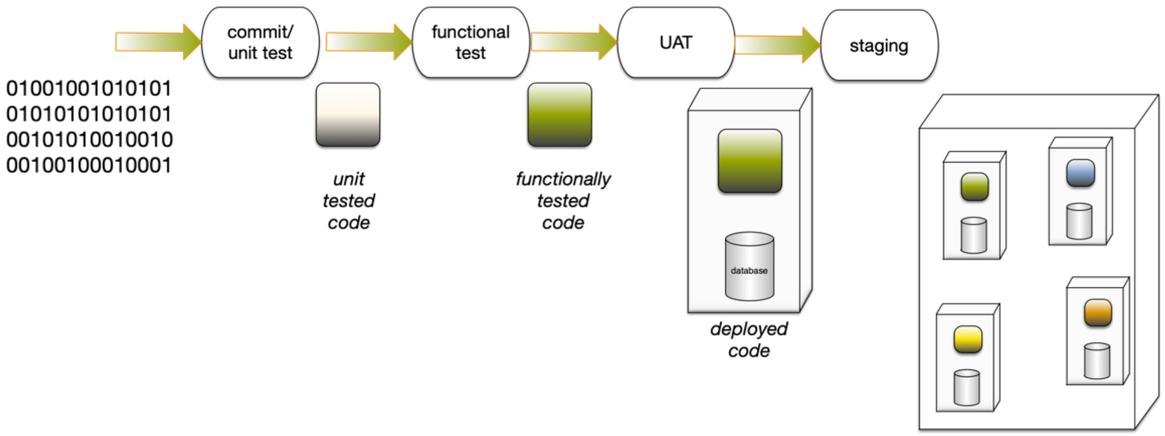
Better Structure



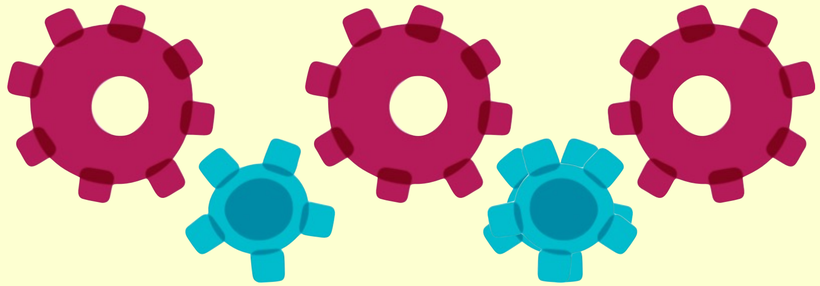
Better Prioritization

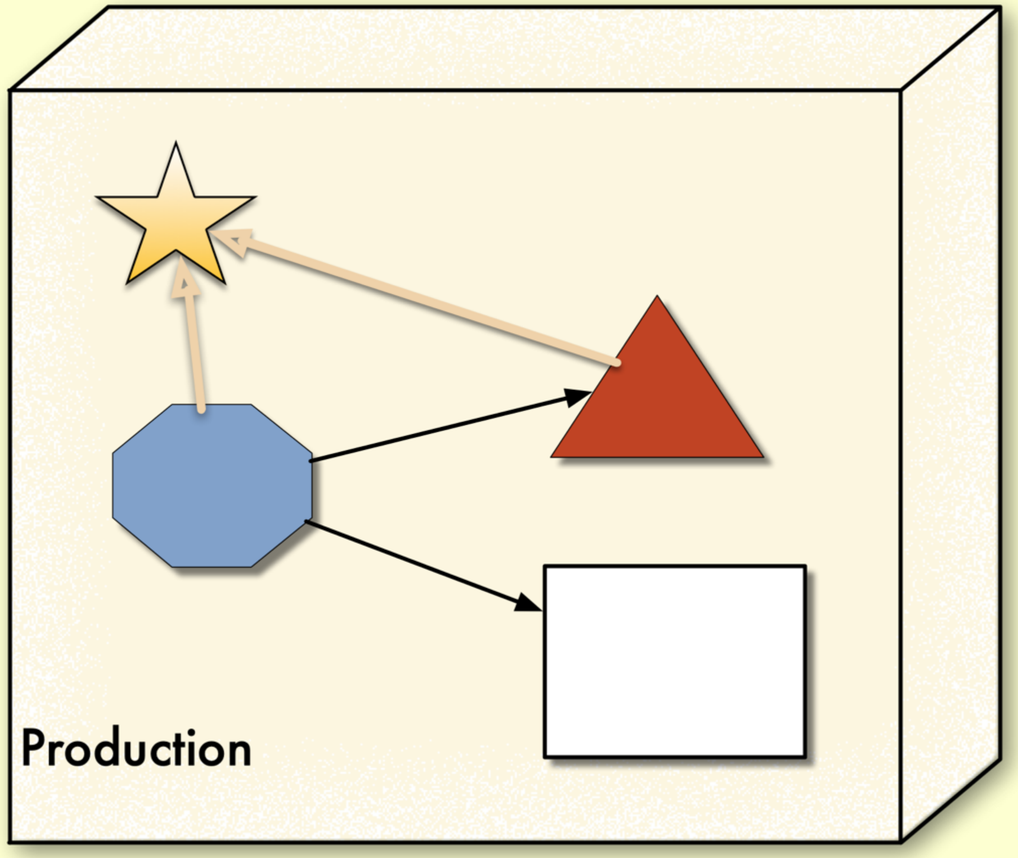
incremental



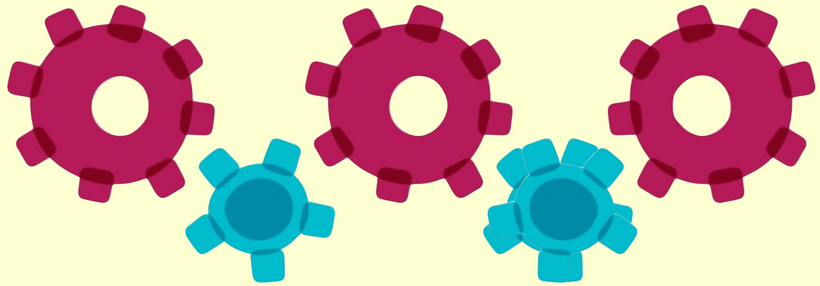


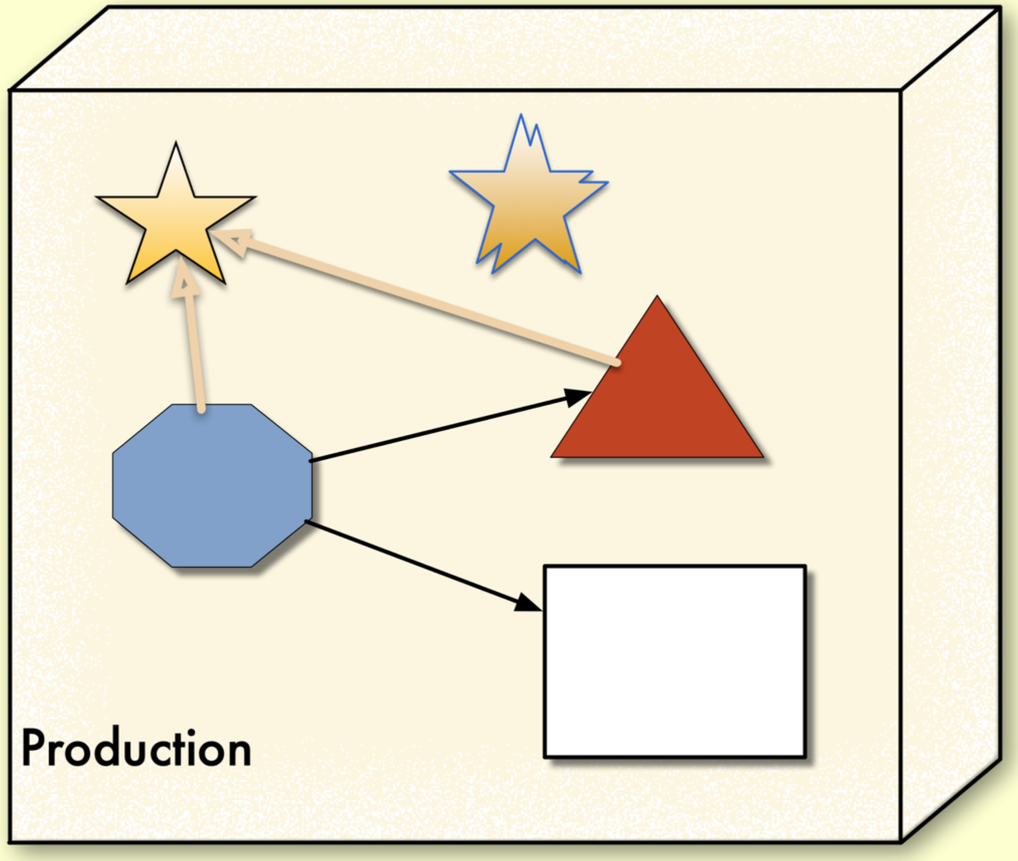
Penultima⬆☒e



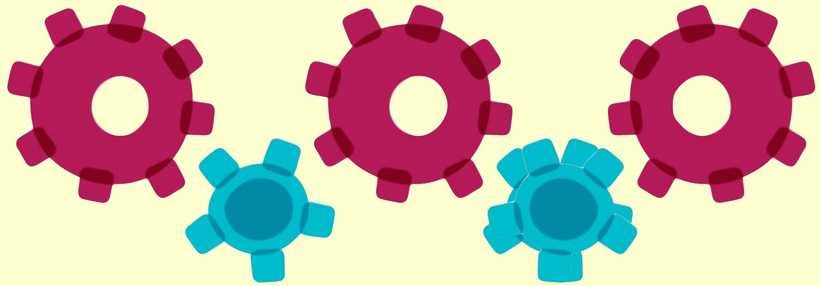


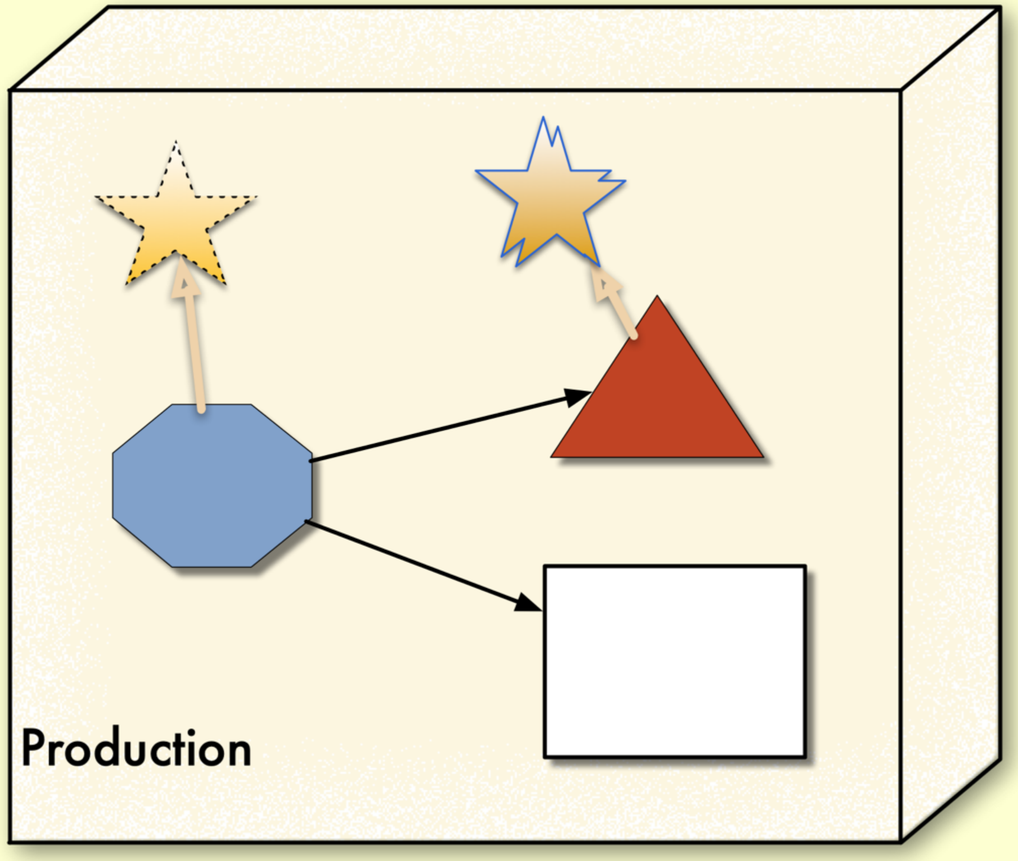
Penultima⬆⊠e



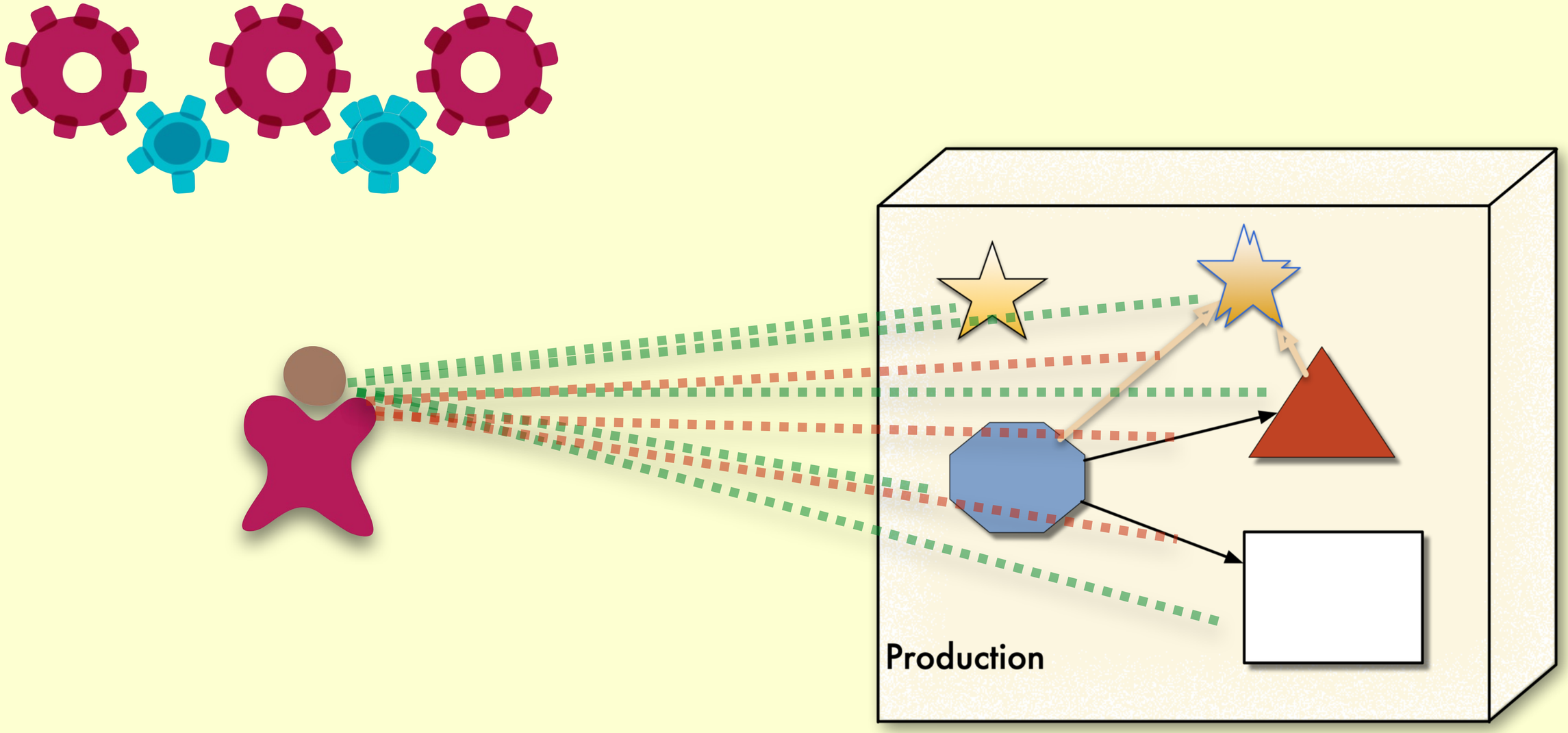


Penultima⬆⊠e

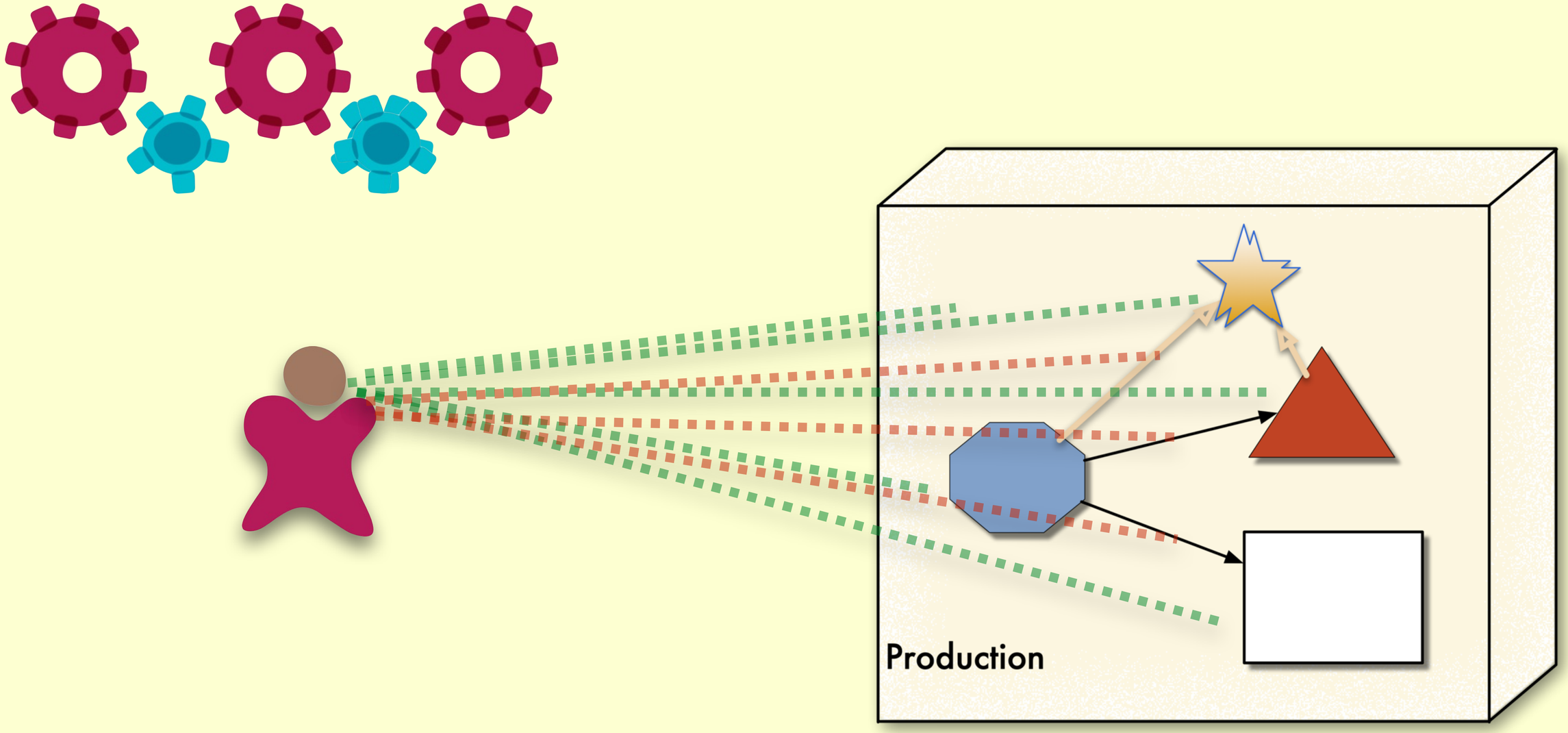




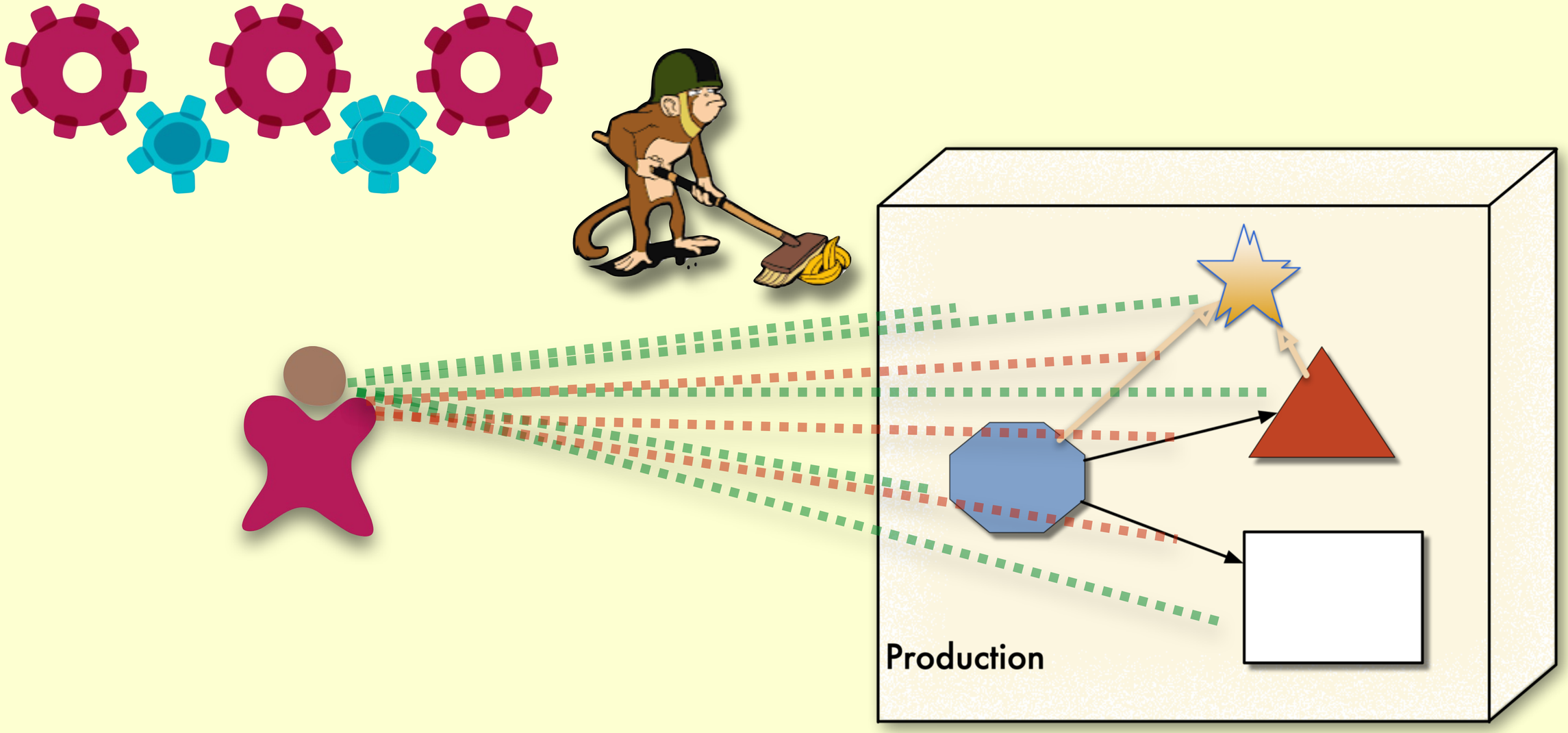
Penultima⬆⊠e

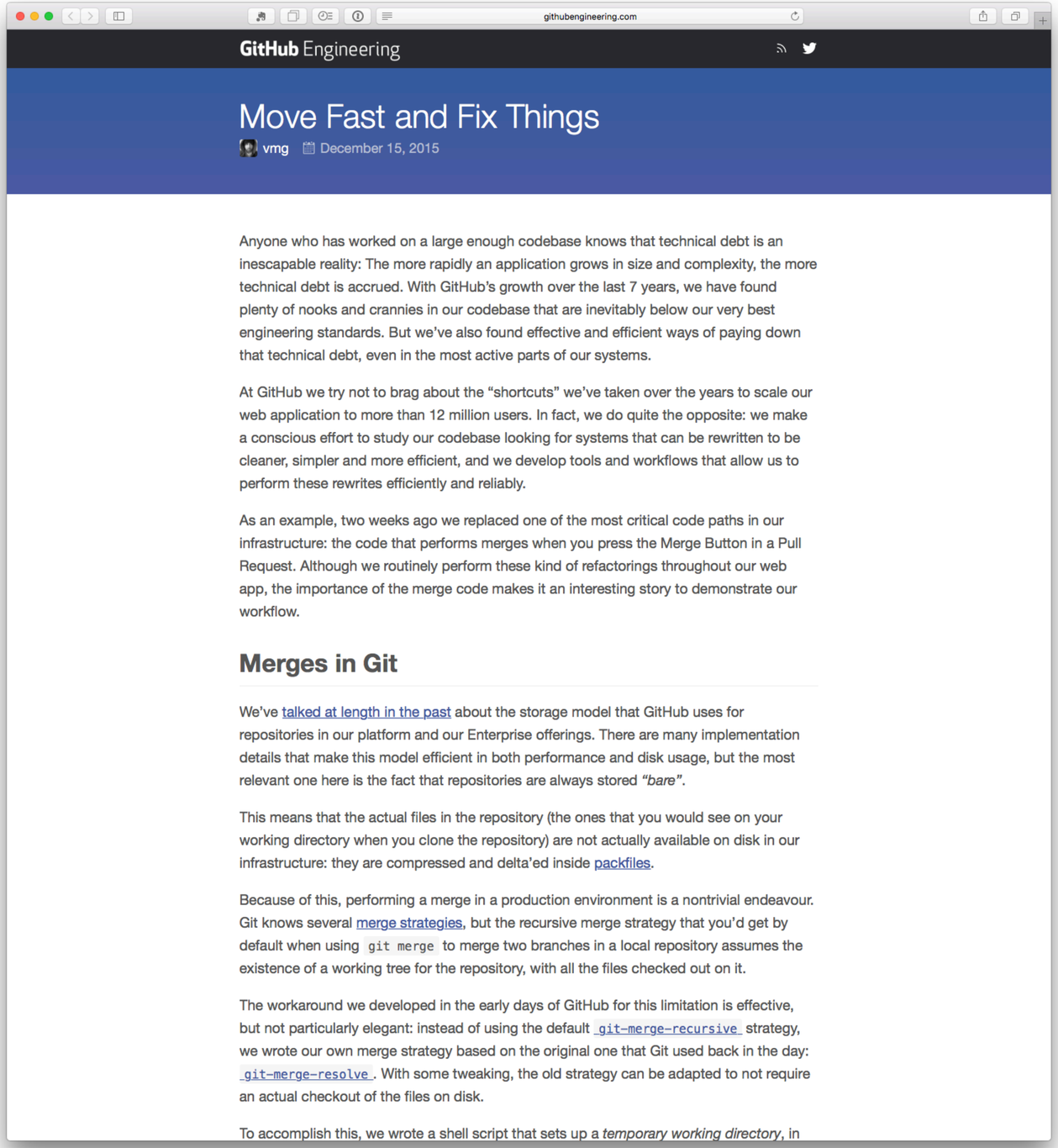


Penultima⬆⊠e



Penultima⬆⊠e





GitHub Engineering

Move Fast and Fix Things

vmg December 15, 2015

Anyone who has worked on a large enough codebase knows that technical debt is an inescapable reality: The more rapidly an application grows in size and complexity, the more technical debt is accrued. With GitHub's growth over the last 7 years, we have found plenty of nooks and crannies in our codebase that are inevitably below our very best engineering standards. But we've also found effective and efficient ways of paying down that technical debt, even in the most active parts of our systems.

At GitHub we try not to brag about the "shortcuts" we've taken over the years to scale our web application to more than 12 million users. In fact, we do quite the opposite: we make a conscious effort to study our codebase looking for systems that can be rewritten to be cleaner, simpler and more efficient, and we develop tools and workflows that allow us to perform these rewrites efficiently and reliably.

As an example, two weeks ago we replaced one of the most critical code paths in our infrastructure: the code that performs merges when you press the Merge Button in a Pull Request. Although we routinely perform these kind of refactorings throughout our web app, the importance of the merge code makes it an interesting story to demonstrate our workflow.

Merges in Git

We've talked at length in the past about the storage model that GitHub uses for repositories in our platform and our Enterprise offerings. There are many implementation details that make this model efficient in both performance and disk usage, but the most relevant one here is the fact that repositories are always stored *"bare"*.

This means that the actual files in the repository (the ones that you would see on your working directory when you clone the repository) are not actually available on disk in our infrastructure: they are compressed and delta'ed inside packfiles.

Because of this, performing a merge in a production environment is a nontrivial endeavour. Git knows several merge strategies, but the recursive merge strategy that you'd get by default when using `git merge` to merge two branches in a local repository assumes the existence of a working tree for the repository, with all the files checked out on it.

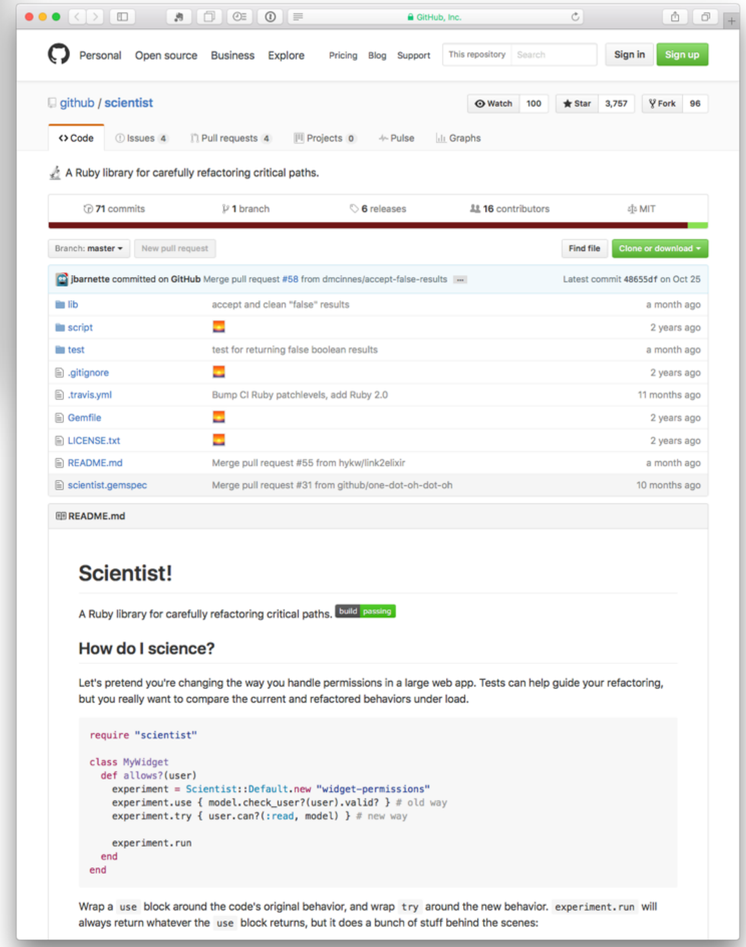
The workaround we developed in the early days of GitHub for this limitation is effective, but not particularly elegant: instead of using the default `git-merge-recursive` strategy, we wrote our own merge strategy based on the original one that Git used back in the day: `git-merge-resolve`. With some tweaking, the old strategy can be adapted to not require an actual checkout of the files on disk.

To accomplish this, we wrote a shell script that sets up a *temporary working directory*, in

Move Fast & Fix Things

```ruby
def create_merge_commit(base, head, author, commit_message)
  base = resolve_commit(base)
  head = resolve_commit(head)
  commit_message = Rugged.prettify_message(commit_message)

  merge_base = rugged.merge_base(base, head)
  return [nil, "already_merged"] if merge_base == head.oid

  ancestor_tree = merge_base && Rugged::Commit.lookup(rugged, merge_base).tree
  merge_options = {
    :fail_on_conflict => true,
    :skip_reuc => true,
    :no_recursive => true,
  }
  index = base.tree.merge(head.tree, ancestor_tree, merge_options)
  return [nil, "merge_conflict"] if (index.nil? || index.conflicts?)

  options = {
    :message    => commit_message,
    :committer  => author,
    :author     => author,
    :parents    => [base, head],
    :tree       => index.write_tree(rugged)
  }

  [Rugged::Commit.create(rugged, options), nil]
end
```

```ruby
def create_merge_commit(author, base, head, options = {})
  commit_message = options[:commit_message] || "Merge #{head} into #{base}"
  now = Time.current

  science "create_merge_commit" do |e|
    e.context :base => base.to_s, :head => head.to_s, :repo => repository.nwo
    e.use { create_merge_commit_git(author, now, base, head, commit_message) }
    e.try { create_merge_commit_rugged(author, now, base, head, commit_message)
  end
end
```



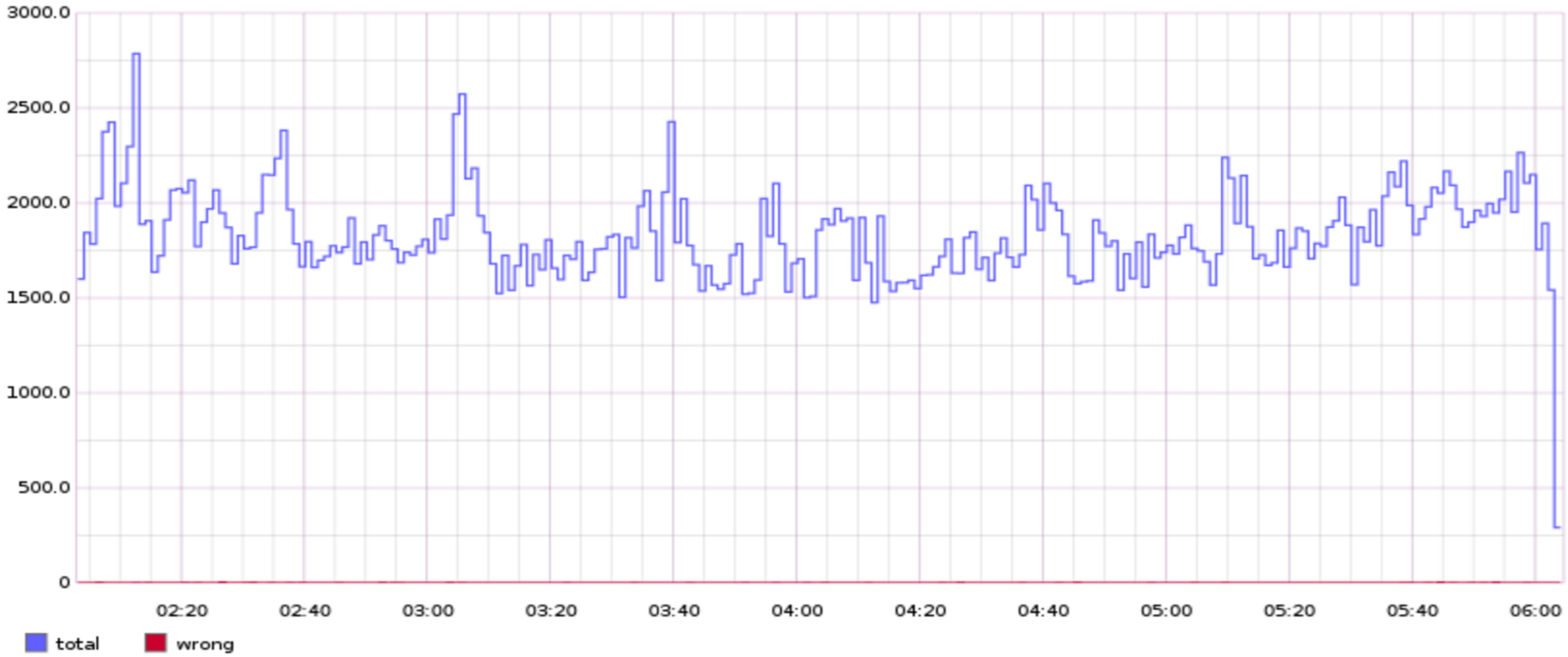
https://github.com/github/scientist

```
require "scientist"

class MyWidget
  def allows?(user)
    experiment = Scientist::Default.new "widget-permissions"
    experiment.use { model.check_user?(user).valid? } # old way
    experiment.try { user.can?(:read, model) } # new way

    experiment.run
  end
end
```

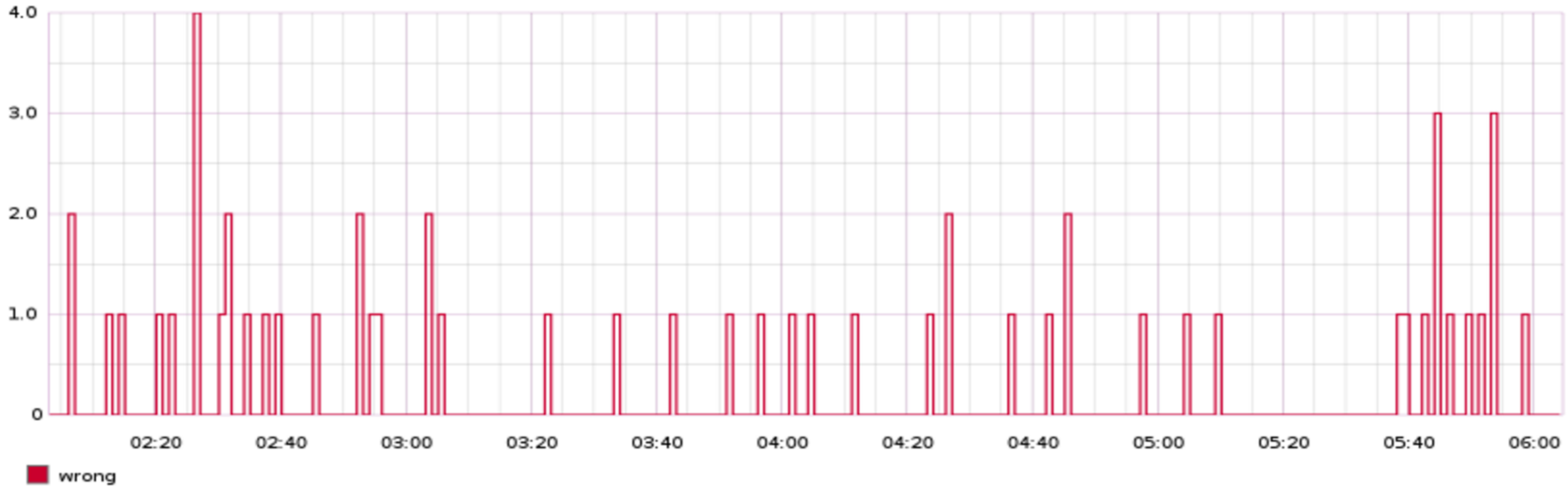
- It decides whether or not to run the try block,
- Randomizes the order in which use and try blocks are run,
- Measures the durations of all behaviors,
- Compares the result of try to the result of use,
- Swallows (but records) any exceptions raised in the try block
- Publishes all this information.

Accuracy

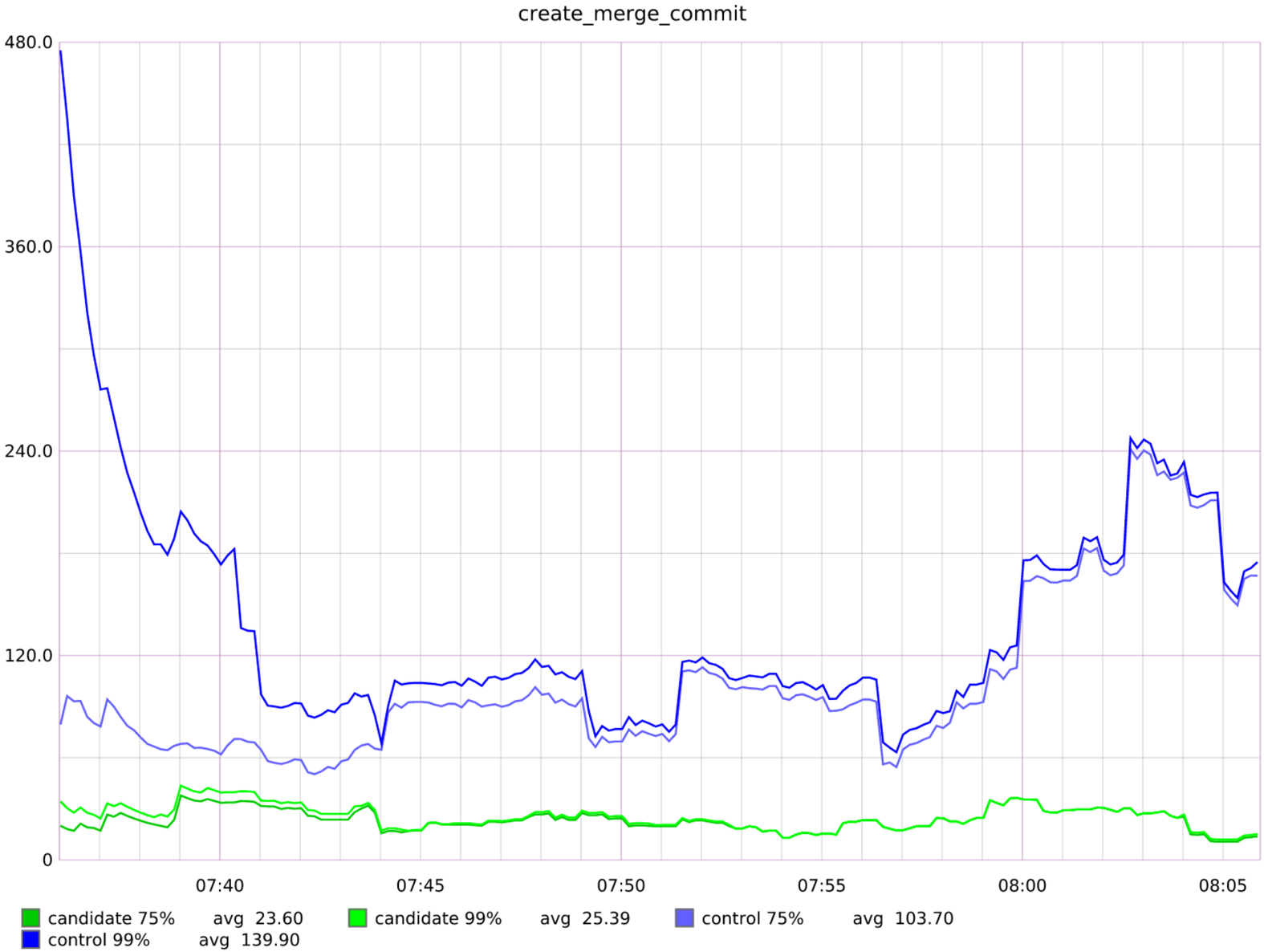
The number of times that the candidate and the control agree or disagree. View mismatches



The number of incorrect/ignored only.



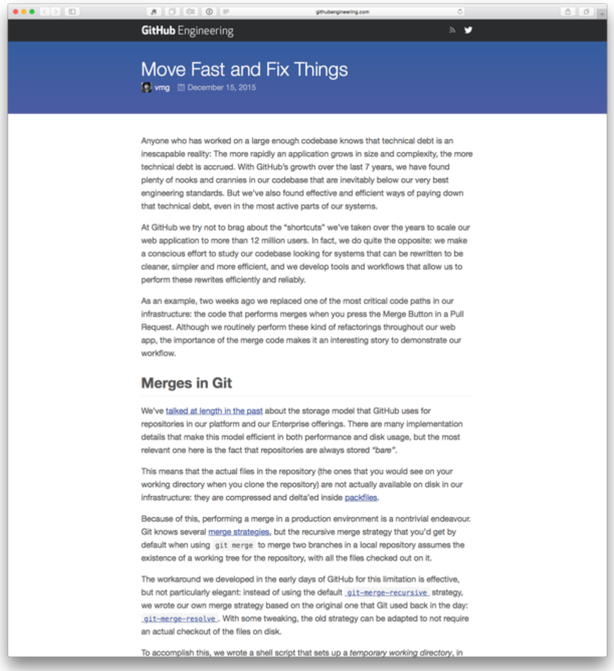
GitHub Engineering
Move Fast and Fix Things


create_merge_commit
480.0
360.0
240.0
120.0
0
07:40
07:45
07:50
07:55
08:00
08:05
candidate 75% avg 23.60
candidate 99% avg 25.39
control 75% avg 103.70
control 99% avg 139.90

4 days

24 hours/
no mismatches or slow cases

> 10,000,000
comparisons



GitHub Engineering

Move Fast and Fix Things

vmg December 15, 2015

Anyone who has worked on a large enough codebase knows that technical debt is an inescapable reality: The more rapidly an application grows in size and complexity, the more technical debt is accrued. With GitHub's growth over the last 7 years, we have found plenty of nooks and crannies in our codebase that are inevitably below our very best engineering standards. But we've also found effective and efficient ways of paying down that technical debt, even in the most active parts of our systems.

At GitHub we try not to brag about the "shortcuts" we've taken over the years to scale our web application to more than 12 million users. In fact, we do quite the opposite: we make a conscious effort to study our codebase looking for systems that can be rewritten to be cleaner, simpler and more efficient, and we develop tools and workflows that allow us to perform these rewrites efficiently and reliably.

As an example, two weeks ago we replaced one of the most critical code paths in our infrastructure: the code that performs merges when you press the Merge Button in a Pull Request. Although we routinely perform these kind of refactorings throughout our web app, the importance of the merge code makes it an interesting story to demonstrate our workflow.

Merges in Git

We've talked at length in the past about the storage model that GitHub uses for repositories in our platform and our Enterprise offerings. There are many implementation details that make this model efficient in both performance and disk usage, but the most relevant one here is the fact that repositories are always stored *"bare"*.

This means that the actual files in the repository (the ones that you would see on your working directory when you clone the repository) are not actually available on disk in our infrastructure: they are compressed and delta'ed inside packfiles.

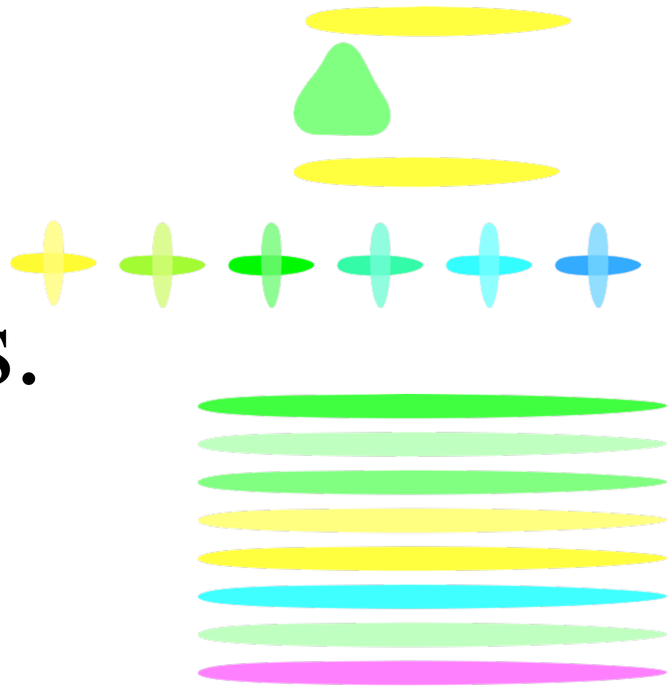
Because of this, performing a merge in a production environment is a nontrivial endeavour. Git knows several merge strategies, but the recursive merge strategy that you'd get by default when using `git merge` to merge two branches in a local repository assumes the existence of a working tree for the repository, with all the files checked out on it.

The workaround we developed in the early days of GitHub for this limitation is effective, but not particularly elegant: instead of using the default `git-merge-recursive` strategy, we wrote our own merge strategy based on the original one that Git used back in the day: `git-merge-resolve`. With some tweaking, the old strategy can be adapted to not require an actual checkout of the files on disk.

To accomplish this, we wrote a shell script that sets up a *temporary working directory*, in

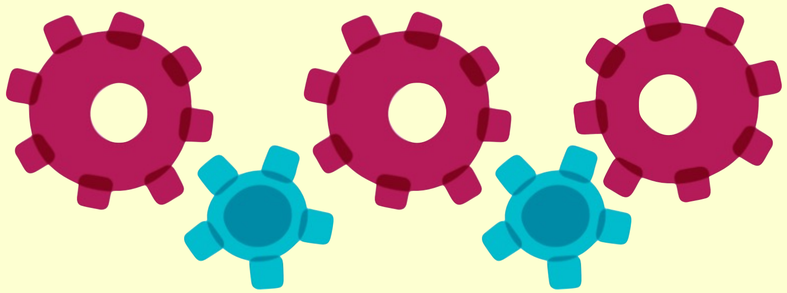
Evolutionary Architecture

An evolutionary architecture supports
guided,
incremental change
across multiple dimensions.



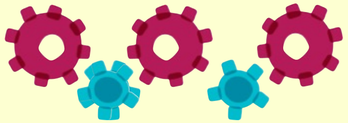
Agenda Part 2

The Evolution of Penultima⬆⊠e



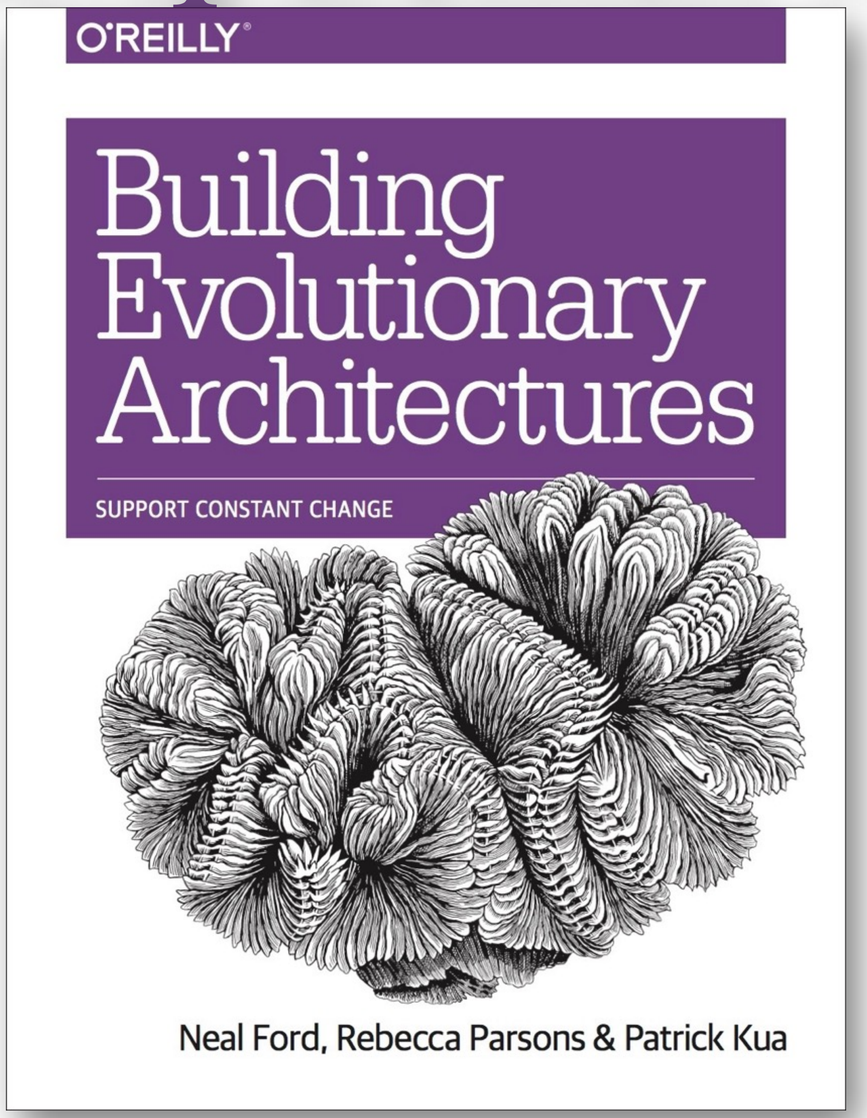
Agenda Part 2

Penultima⬆⊠e



A11yAllTheThings
AuditTheAccounting
AvoidComplexCode
BreakOnUpgrade
ConfigureSomeOfTheThingsAllOfTheTime
CycleTimeGuard
DebugAllTheThings
DegradeGracefully
DependOnDependencies
DeployAllTheThings
DeterministicDistributability
DireDeltas
DiscoverAllTheThings
DocSyncWithApi
ElasticityOfMicroservices
InstallAllTheThings
LegalityOfOpenSourceLibraries
MaintainTheMock
MaintainingGoodThroughput
MakeSureNewReallyReplacesOld
MonitorAllTheThings
NoMoreViewModels
ReplaceTheCruftyCore
RespondResponsibly
SellThePlatform
UpgradeAllTheThings
ZeroDaySecurity

iMpLemENtiNg eVoLuTiONaRy ARcHitECtuREs



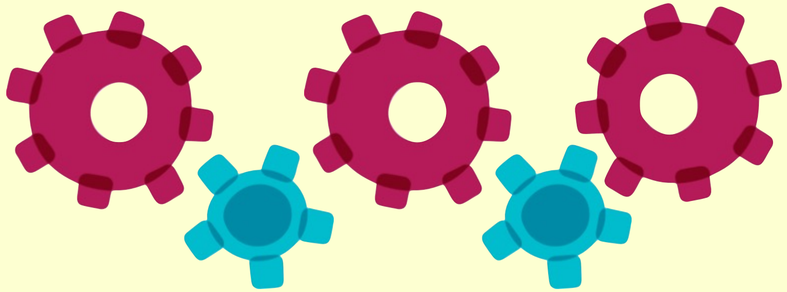
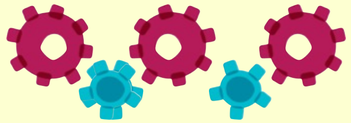
Day 1:
Theory & Practice

@neal4d
nealford.com

@rebeccaparsons

@patkua

Penultima⬆⊠e



Penultima⬆⊠e