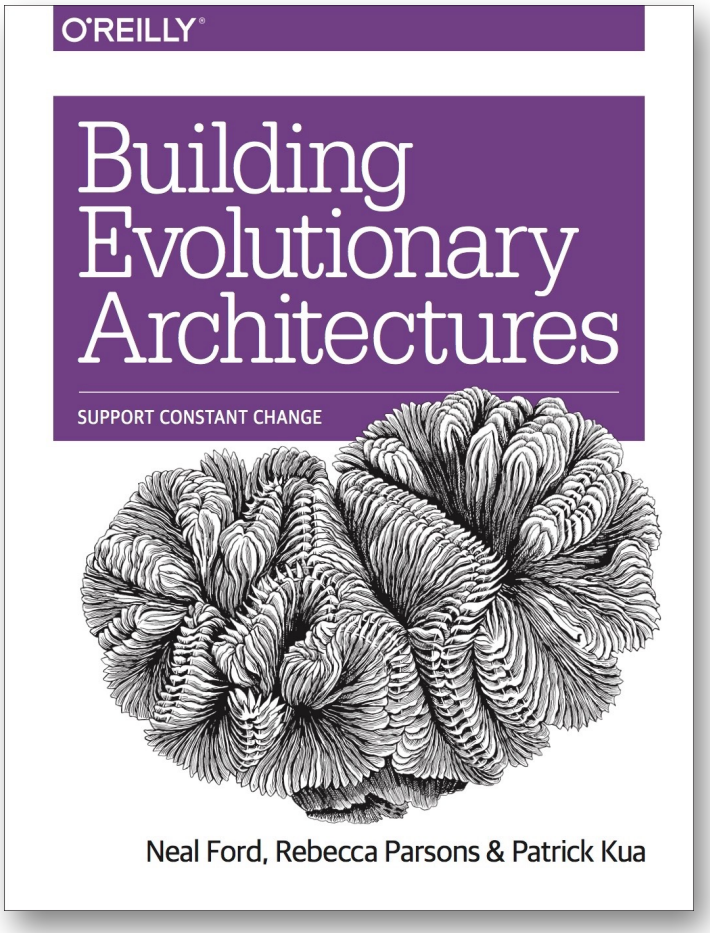
iMpLemENtiNg eVoLuTiONaRy ARcHitECtuREs

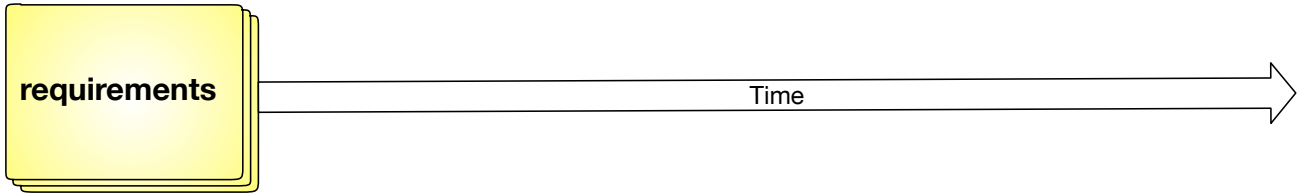


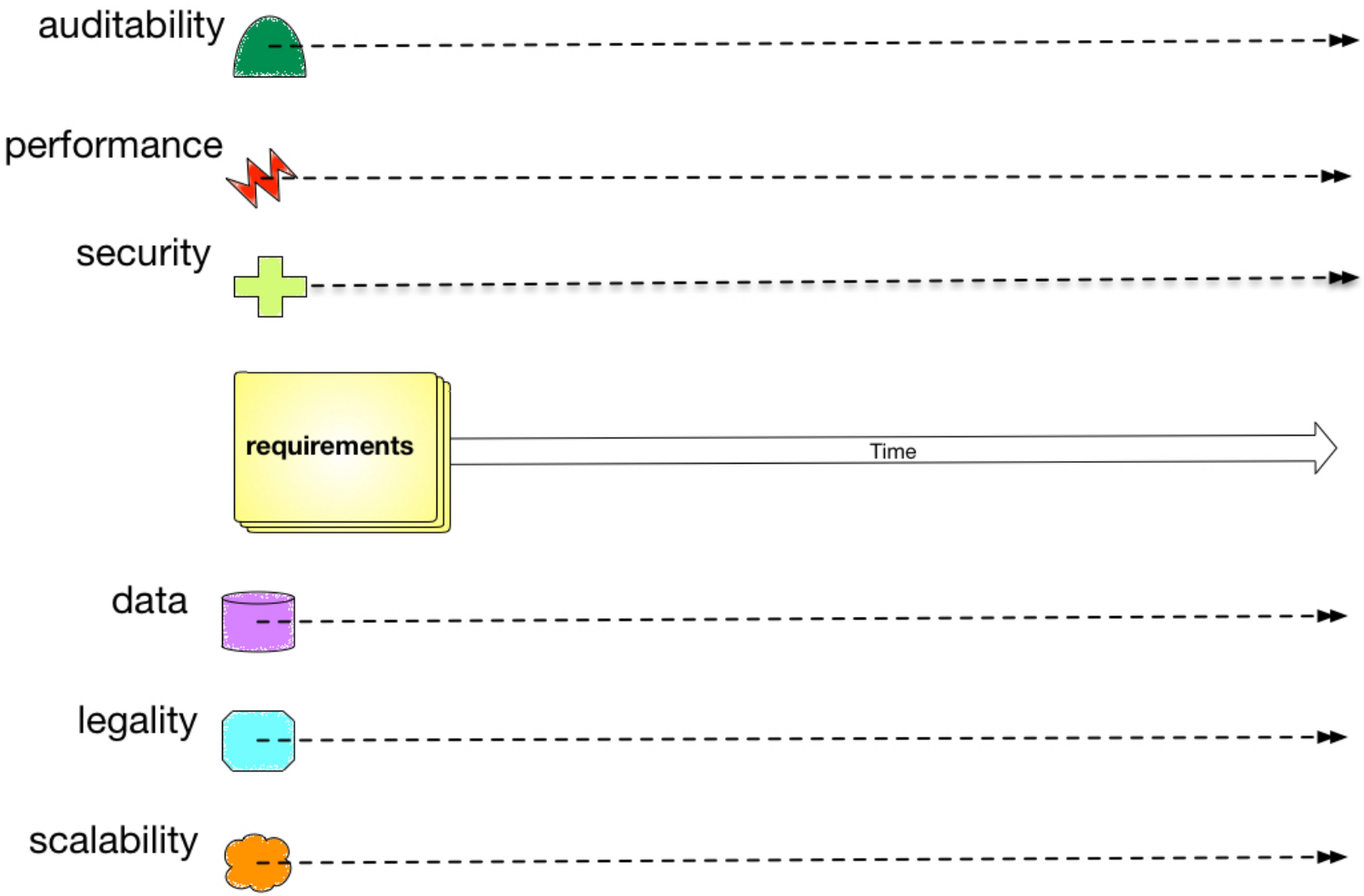
SUPPORT CONSTANT CHANGE

@neal4d
nealford.com

@rebeccaparsons

@patkua


requirements
Time


auditability
performance
security
requirements
Time
data
legality
scalability

ility

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

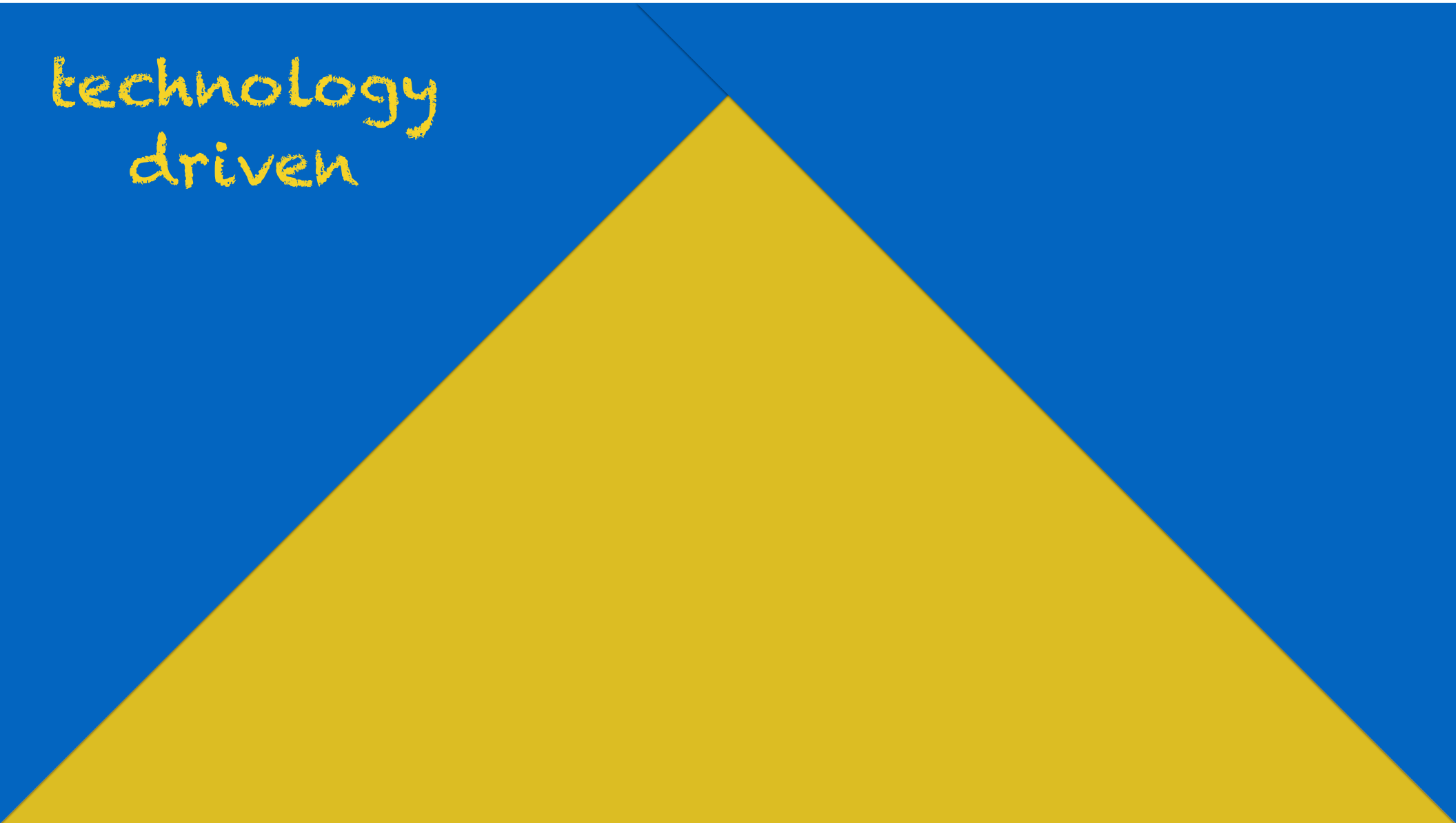
reliability
extensibility
failure transparency
fault-tolerance
fidelity
flexibility
inspectability
installability
integrity
interchangeability
interoperability
learnability
maintainability
manageability
mobility
modifiability
modularity
operability
orthogonality
portability
precision
predictability
process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

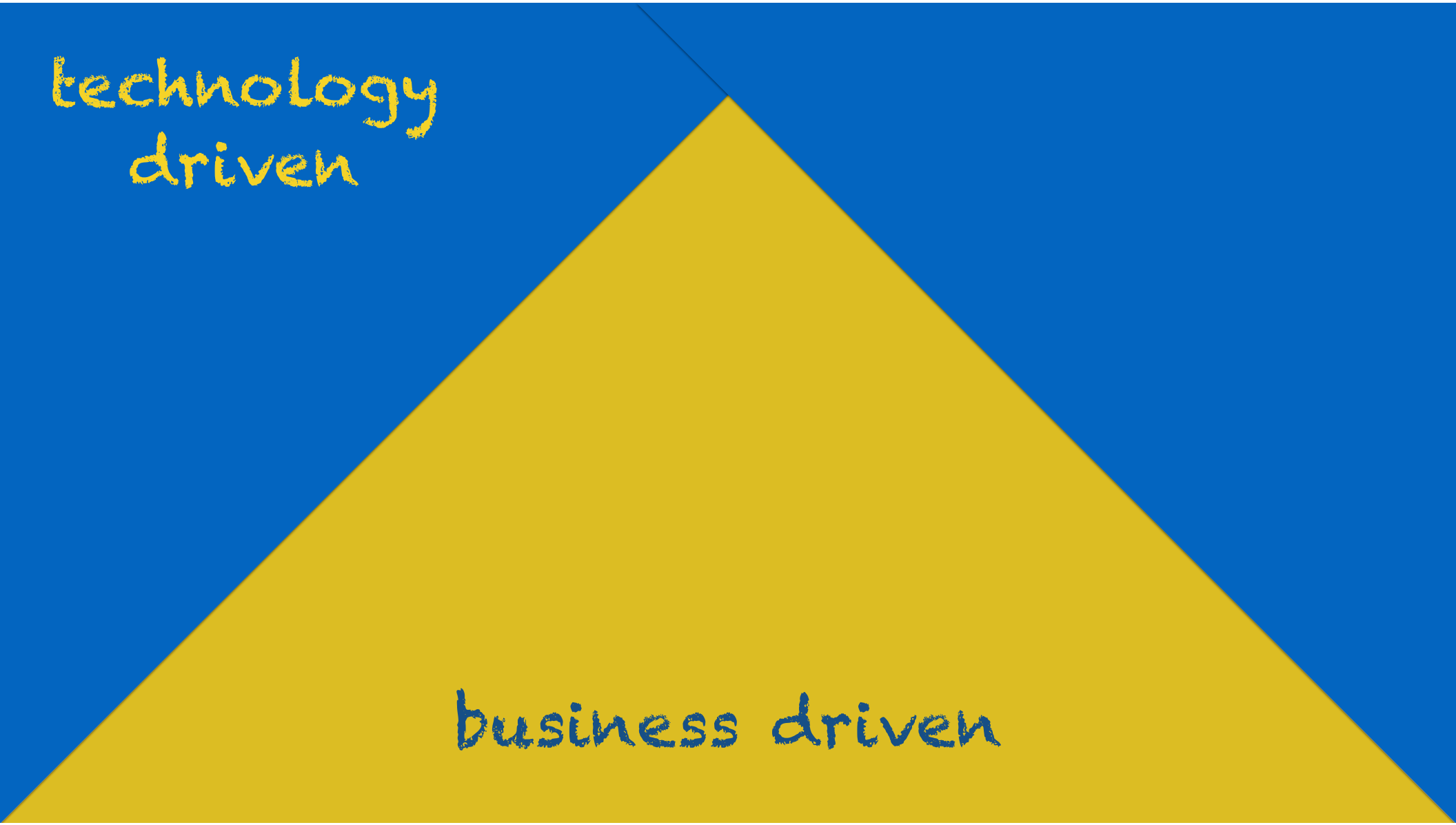
repeatability
reproducibility
resilience
responsiveness
reusability
robustness
safety
scalability
seamlessness
self-sustainability
serviceability
supportability
securability
simplicity
stability
standards compliance
survivability
sustainability
tailorability
testability
timeliness
traceability
transparency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

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

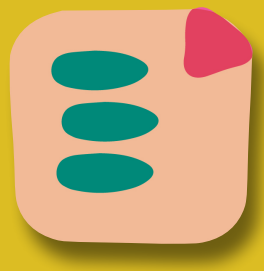

accessibility
accountability
accuracy
adaptability
administrability
affordability
agility
auditability
autonomy
reliability
extensibility
failure transparency
fault-tolerance
fidelity
flexibility
inspectability
installability
integrity
repeatability
reproducibility
resilience
responsiveness
reusability
robustness
safety
scalability
seamlessness
evolvability
degradability
determinability
demonstrability
dependability
deployability
discoverability
distributability
durability
effectiveness
efficiency
operability
orthogonality
portability
precision
predictability
process capabilities
producibility
provability
recoverability
relevance
sustainability
tailorability
testability
timeliness
traceability
transparency
ubiquity
understandability
upgradability
usability


cHAnGe


technology
driven


technology
driven
business driven

technology driven

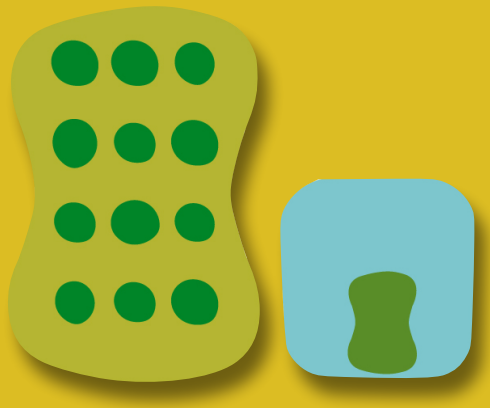


business driven

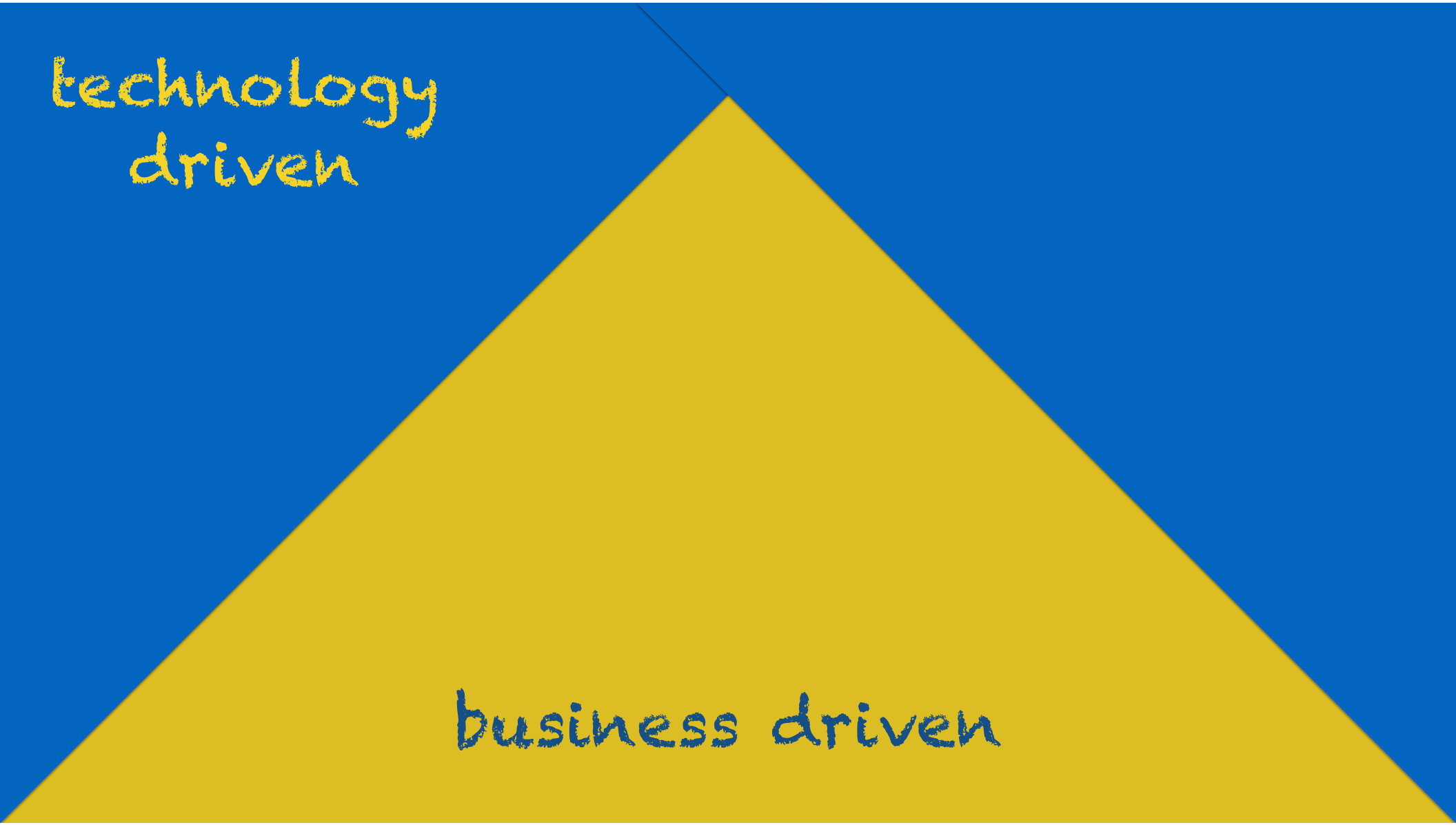
technology driven

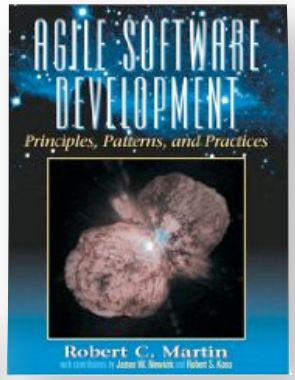
business driven

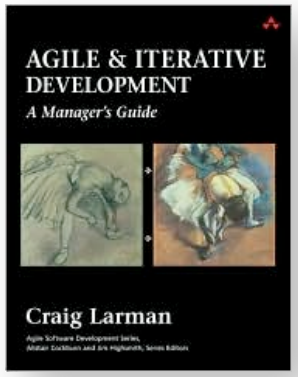
technology driven

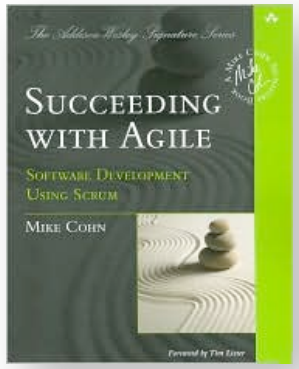


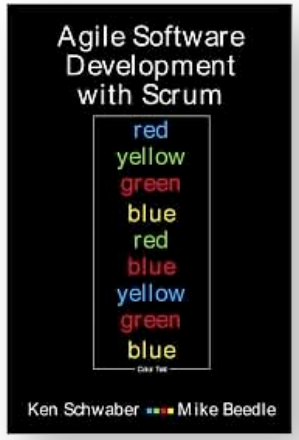
business driven


technology
driven
business driven

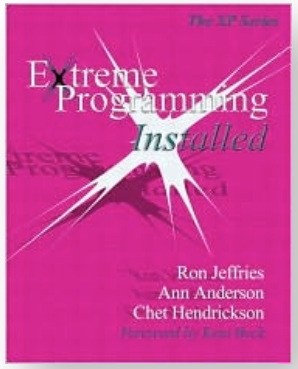

AGILE SOFTWARE DEVELOPMENT
Principles, Patterns, and Practices
Robert C. Martin

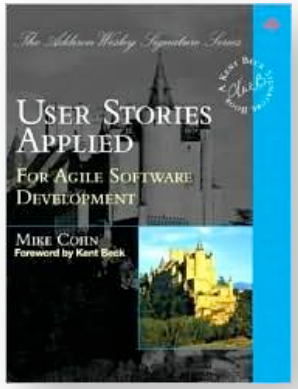

AGILE & ITERATIVE DEVELOPMENT
A Manager's Guide
Craig Larman

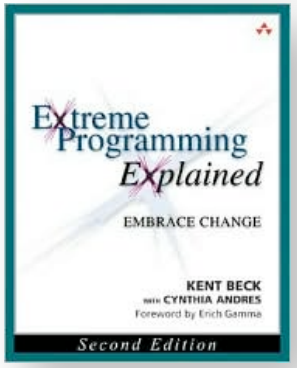

SUCCEEDING WITH AGILE
SOFTWARE DEVELOPMENT USING SCRUM
MIKE COHN

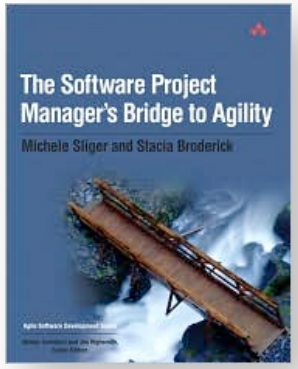

Agile Software Development with Scrum
red
yellow
green
blue
red
blue
yellow
green
blue
Ken Schwaber
Mike Beedle

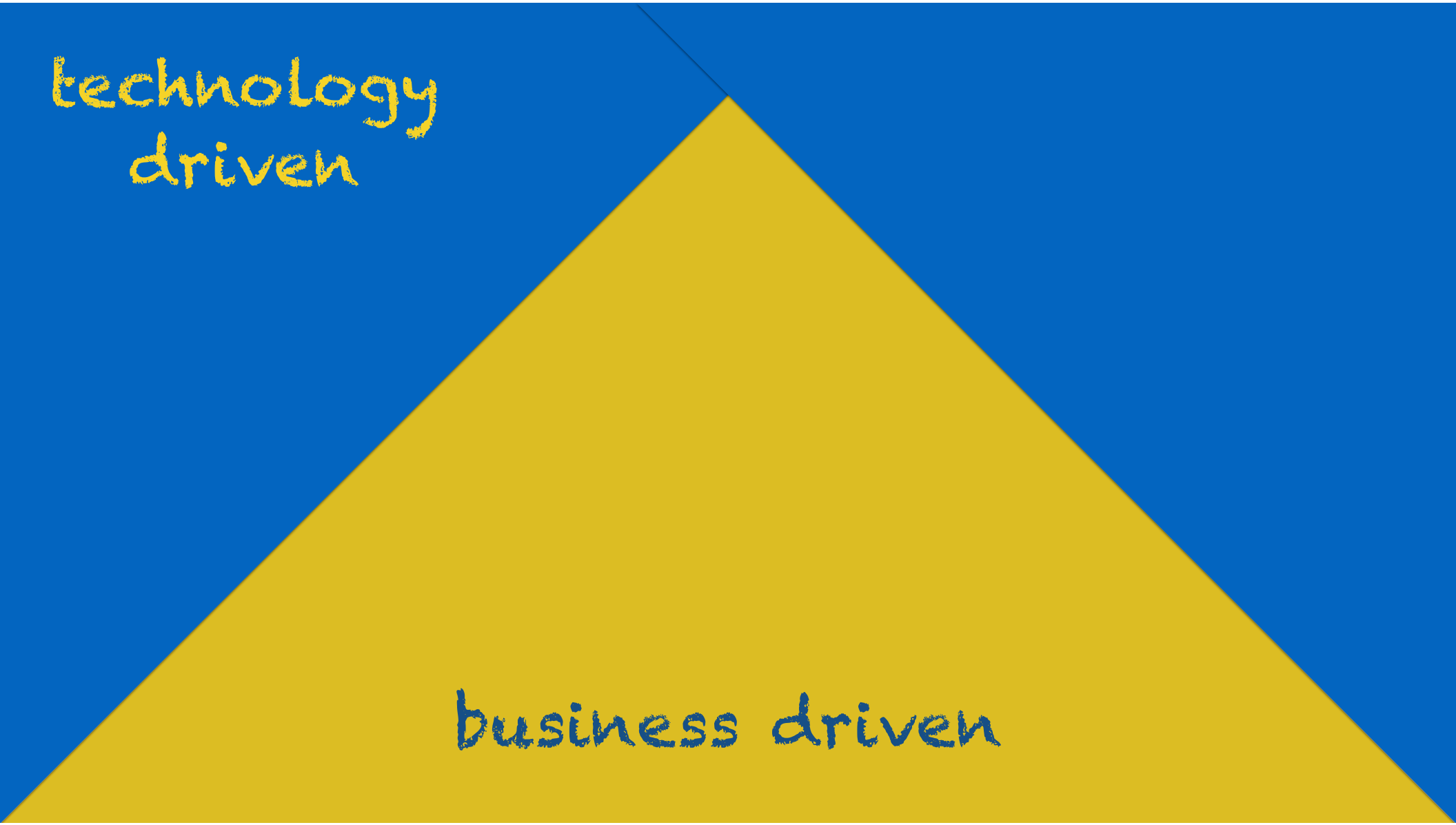

AGILE PROJECT MANAGEMENT WITH SCRUM


Extreme Programming Installed
Ron Jeffries
Ann Anderson
Chet Hendrickson


USER STORIES APPLIED
FOR AGILE SOFTWARE DEVELOPMENT
MIKE COHN
Foreword by Kent Beck

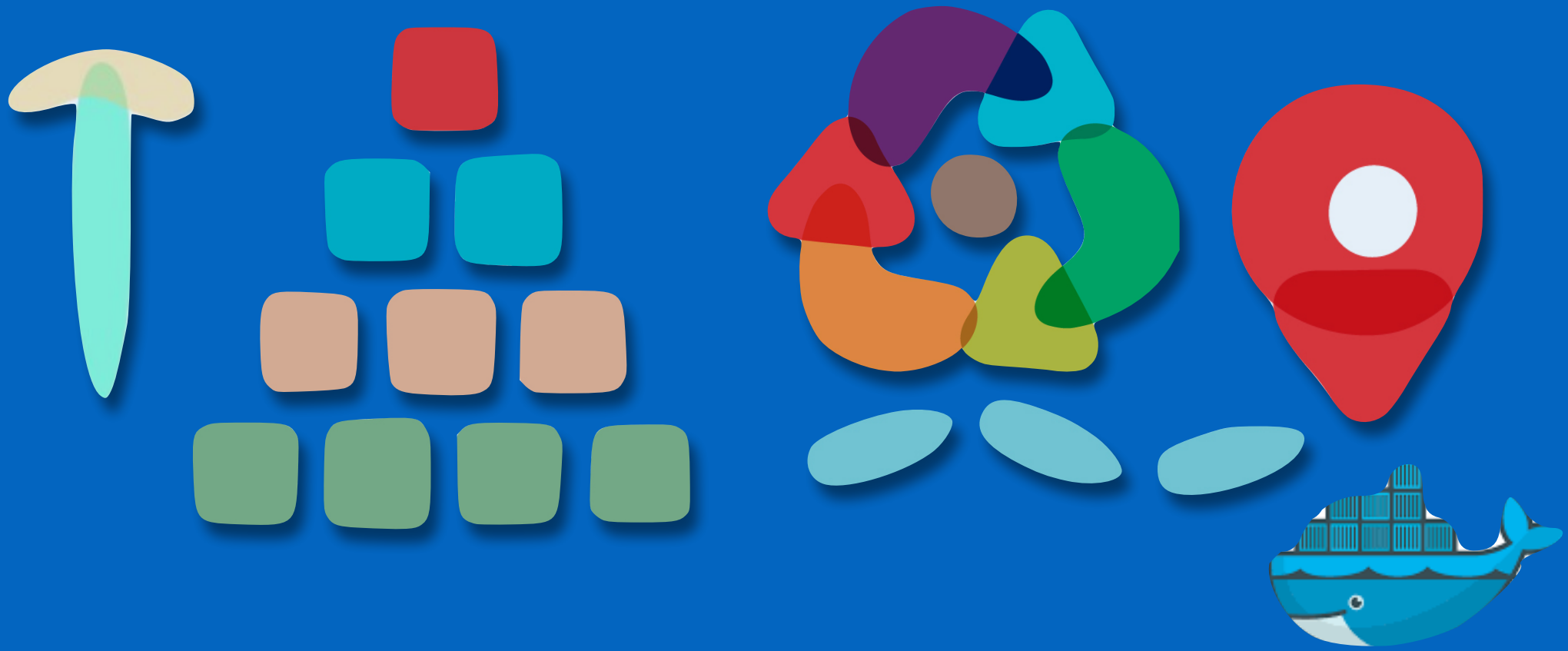

Extreme Programming Explained
EMBRACE CHANGE
KENT BECK
with CYNTHIA ANDRES
Foreword by Erich Gamma
Second Edition


The Software Project Manager's Bridge to Agility
Michele Sliger and Stacia Broderick

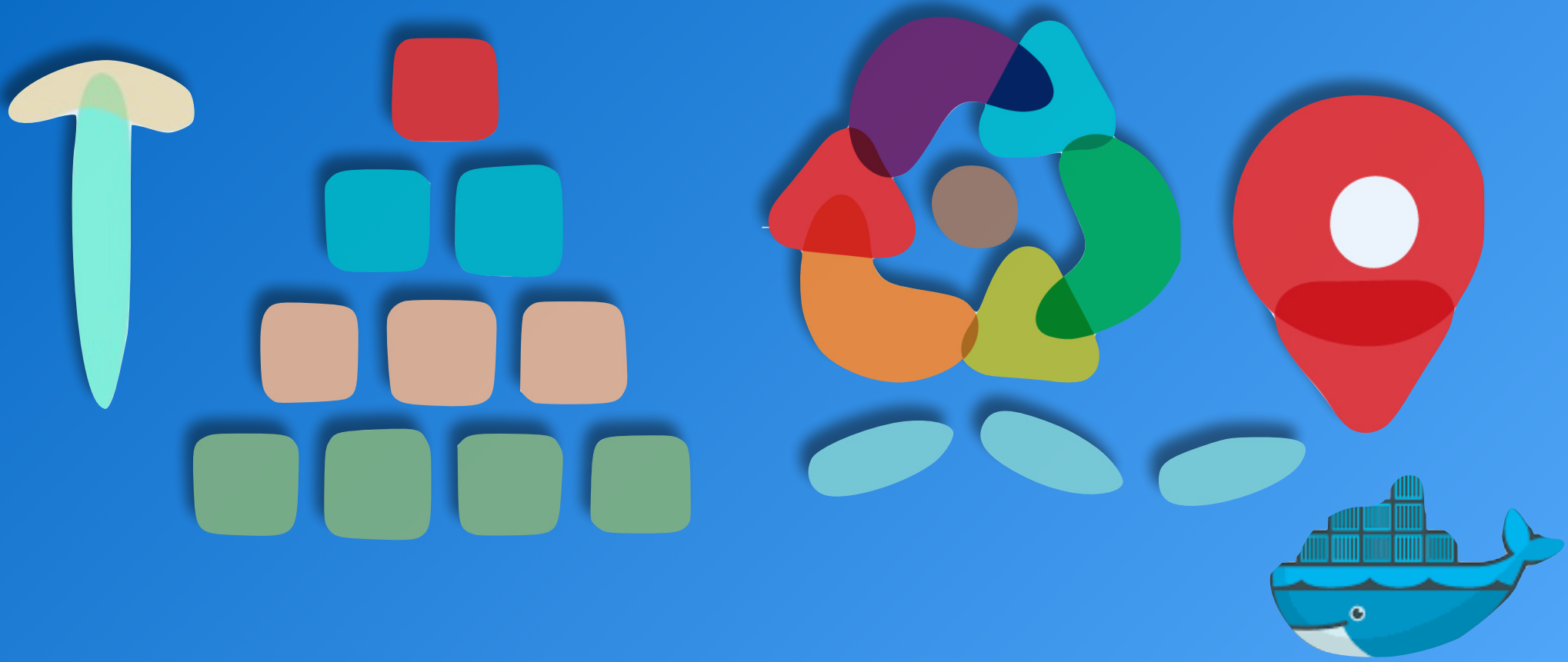

technology
driven
business driven

everything chAnGEs
all the time!

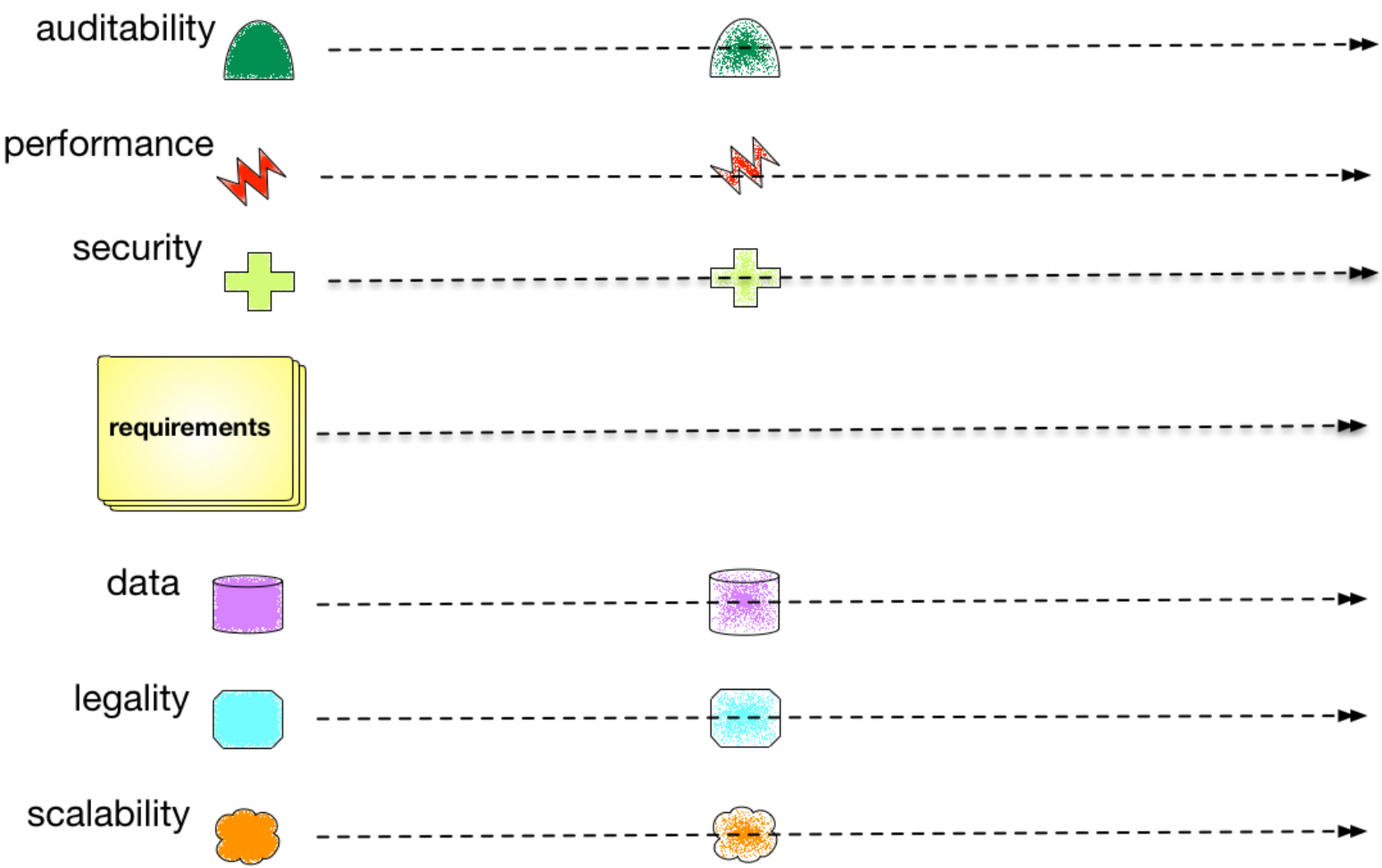
dYNaMic eqUiLiBRiuM



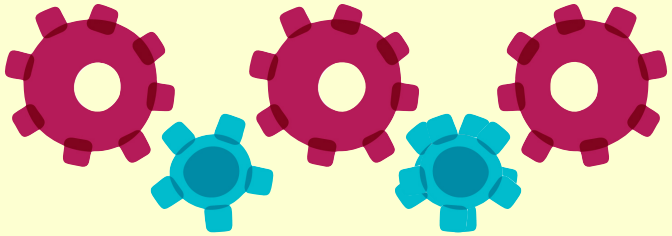
dYNaMic eqUiLiBRiuM



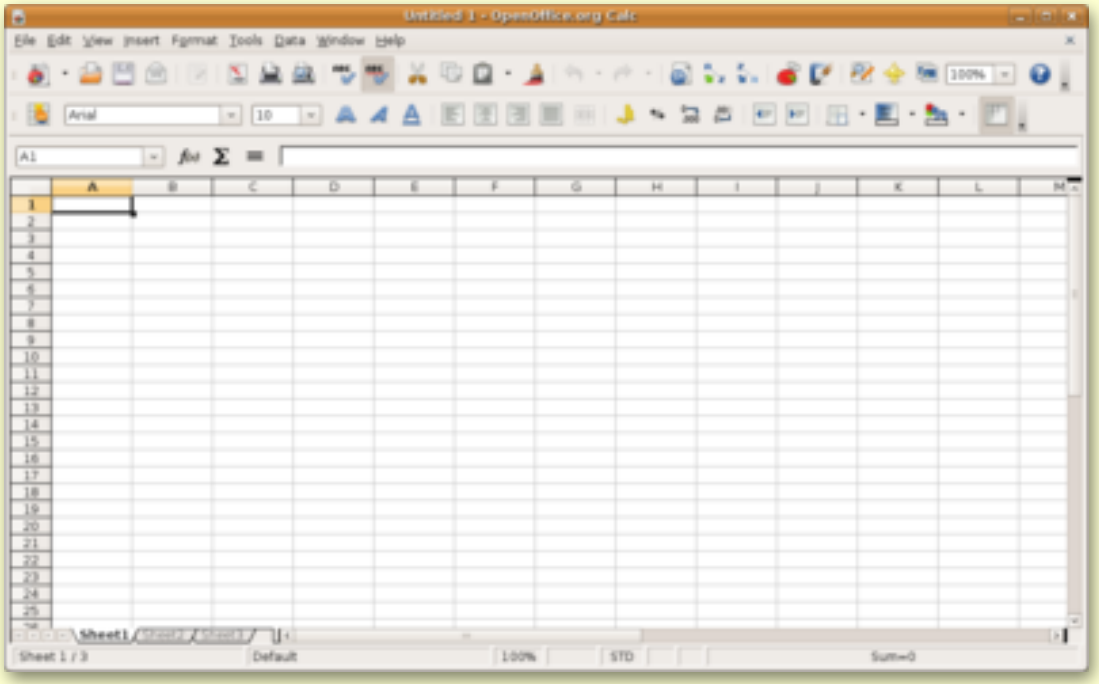
How is long term planning possible when things constantly change in unEXpECtEd **ways?**


auditability
performance
security
requirements
data
legality
scalability

Penultima e

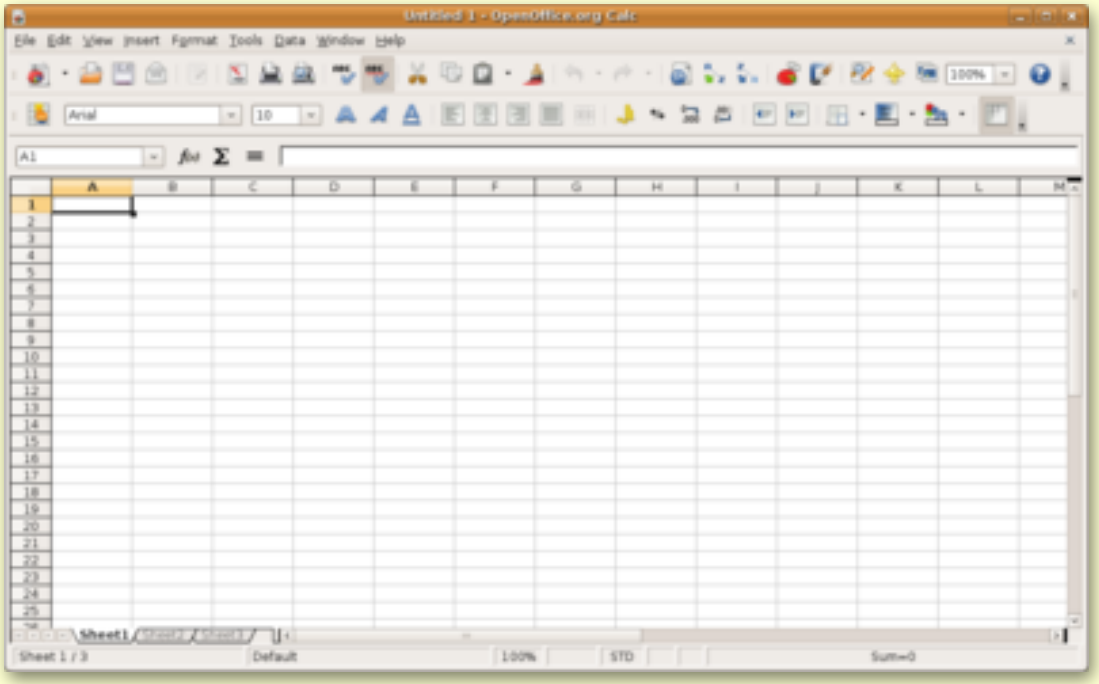


EA Spreadsheet



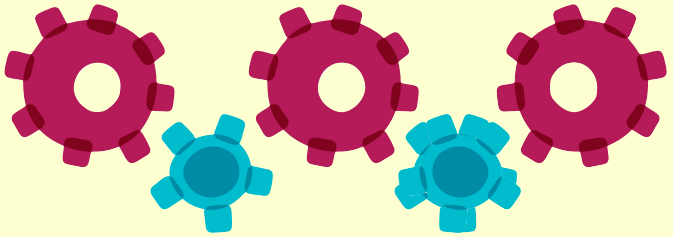
Penultimate

EA Spreadsheet

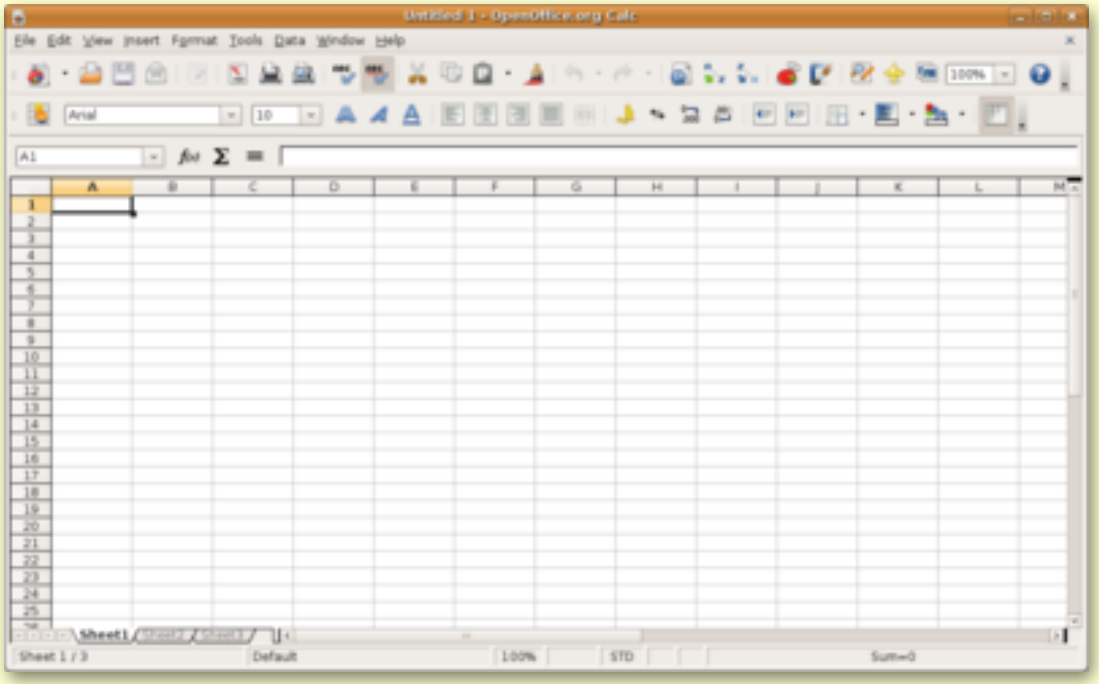


✓ definition

Penultima⬆e



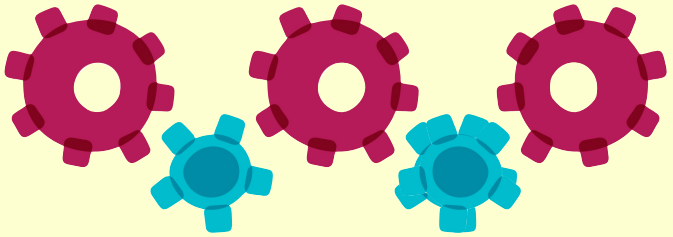
EA Spreadsheet



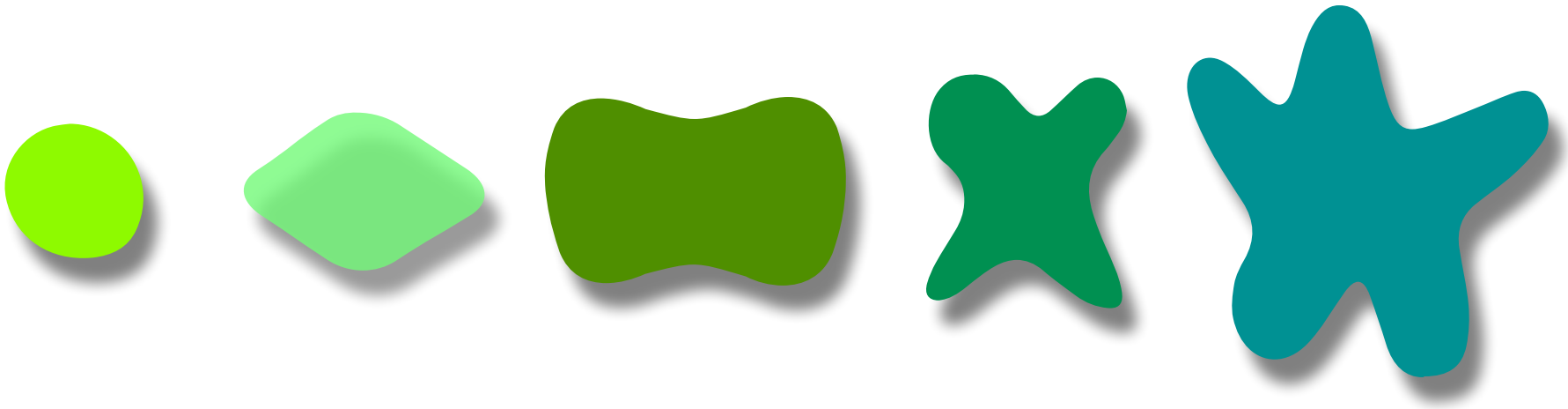
✓ definition

! verification

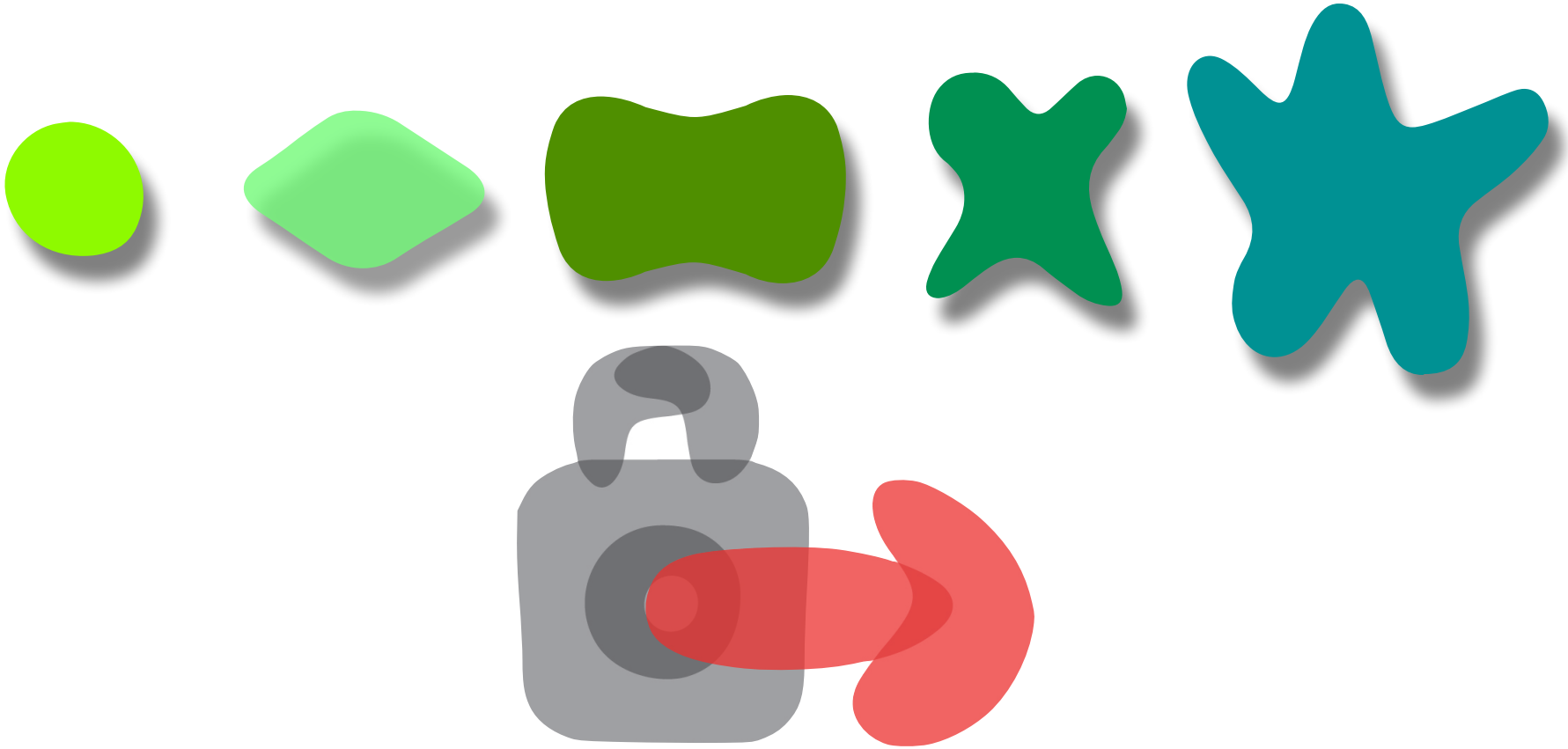
Penultima⬆e



sEcONd-ORdeR eFfEcT

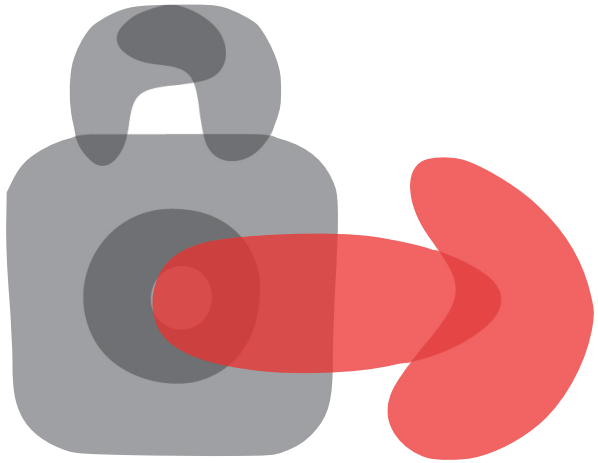


sEcONd-ORdeR eFfEcT



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

governance



Evolutionary Architecture

Continuous Architecture?

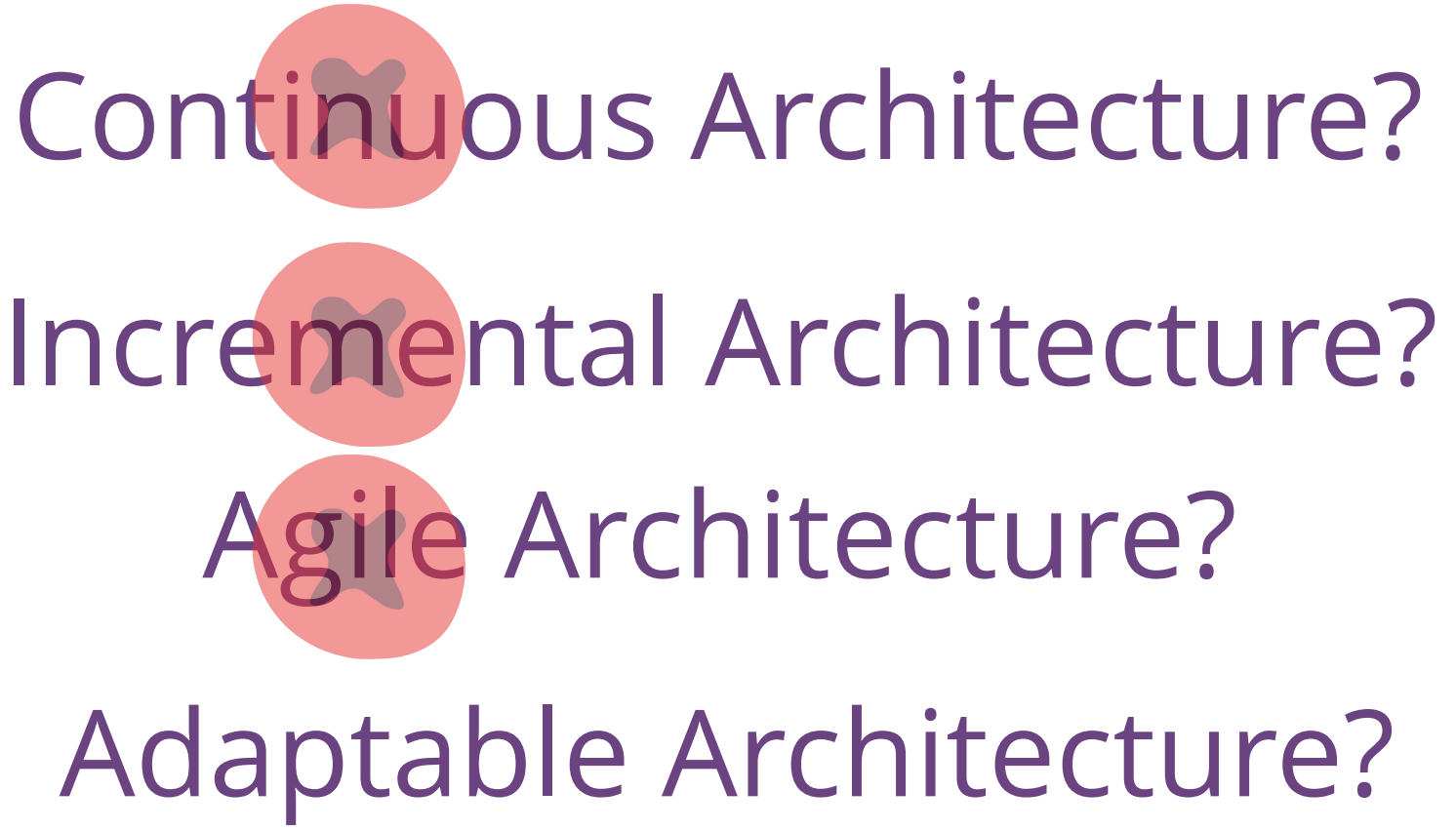
Continuous Architecture?

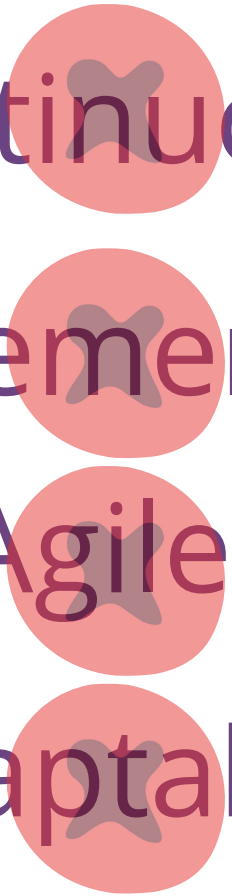
Incremental Architecture?

Continuous Architecture?

Incremental Architecture?

Agile Architecture?


Continuous Architecture?
Incremental Architecture?
Agile Architecture?
Adaptable Architecture?


Continuous Architecture?
Incremental Architecture?
Agile Architecture?
Adaptable Architecture?
Evolutionary 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.

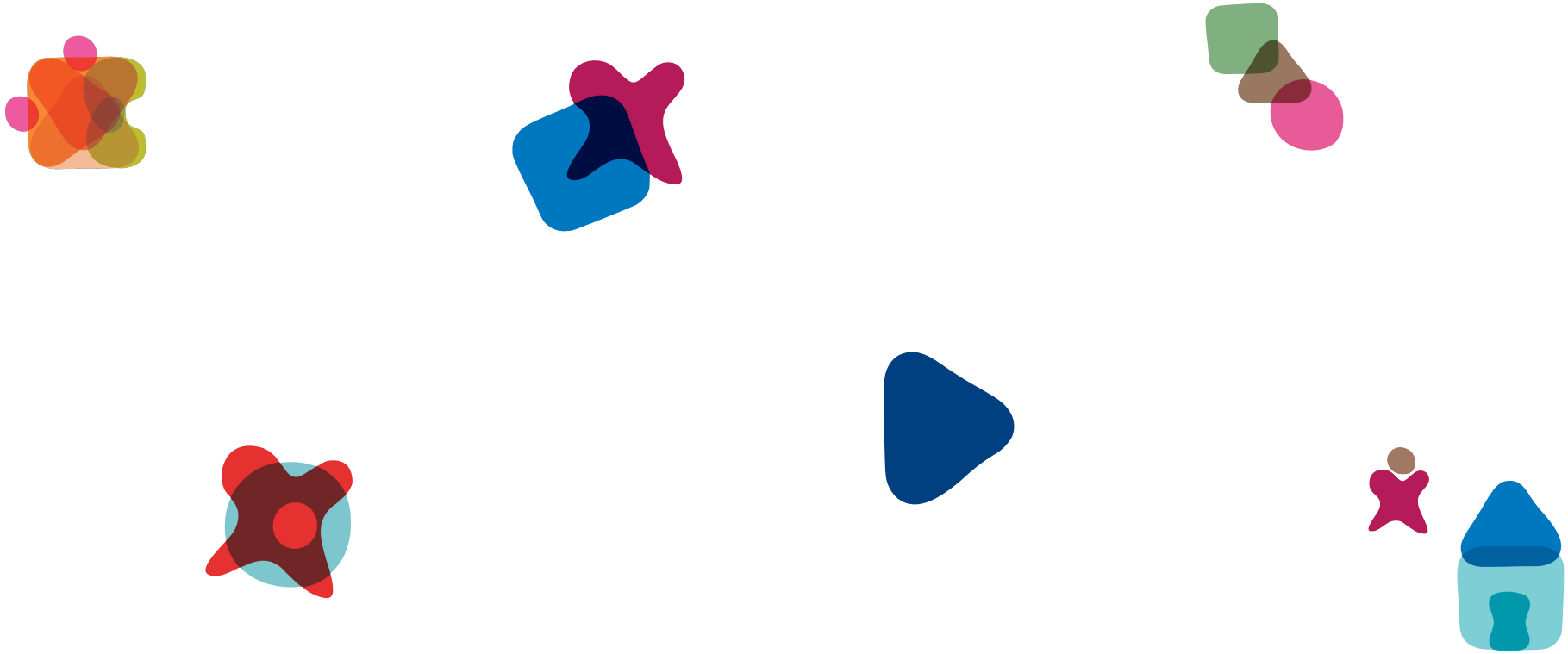
Traveling Salesman Problem







Traveling Salesman Problem



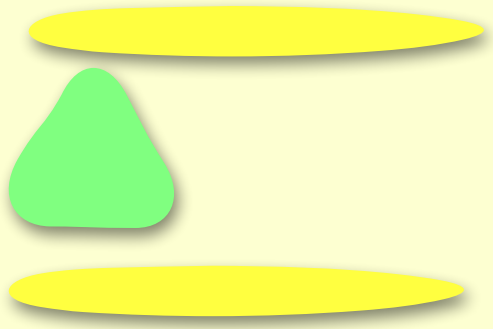
fitness function = length of route


guided

***architectural** fitness function*:

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

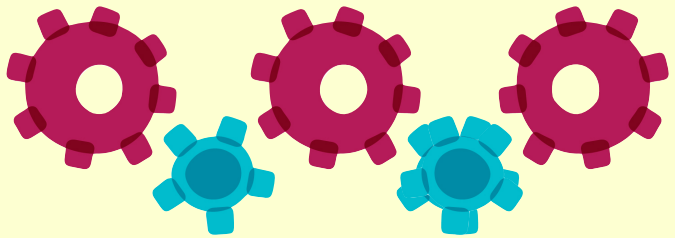
EA Spreadsheet



✓ definition

✓ verification

Penultima⬆e



Evolutionary Architecture

An evolutionary architecture supports

guided

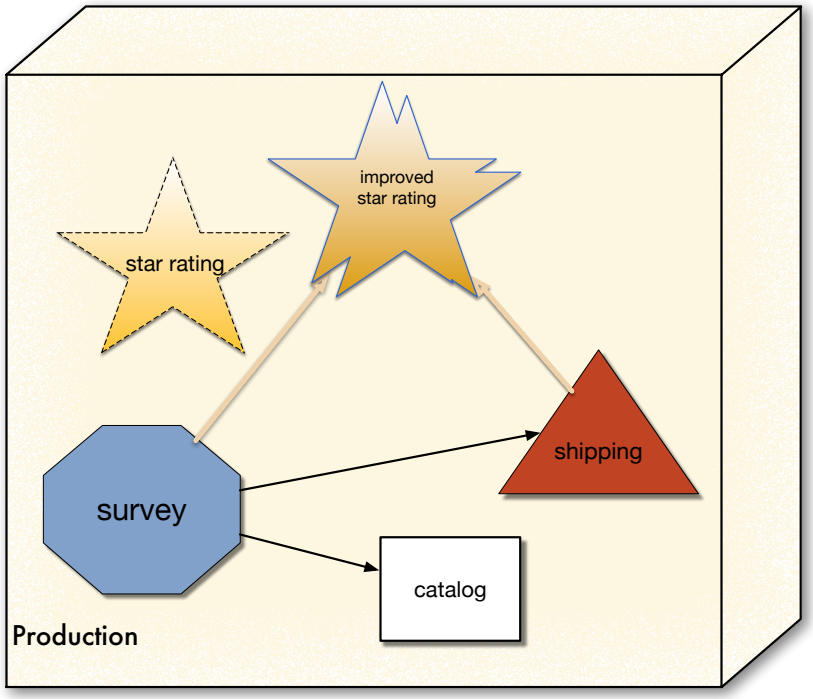
incremental change

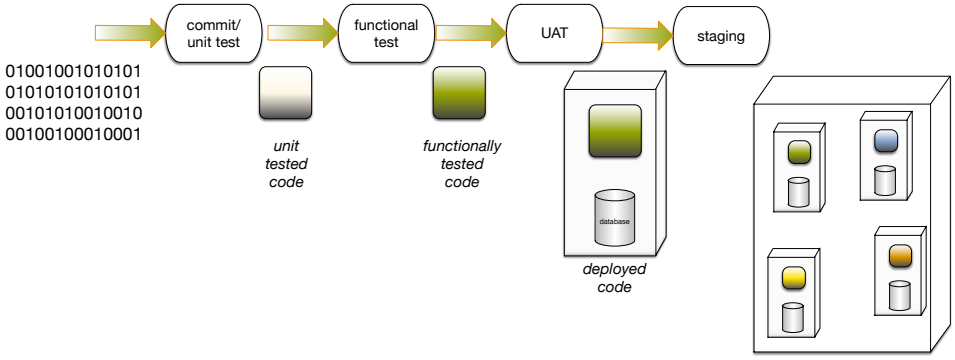
across multiple dimensions.

Evolutionary Architecture

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

incremental





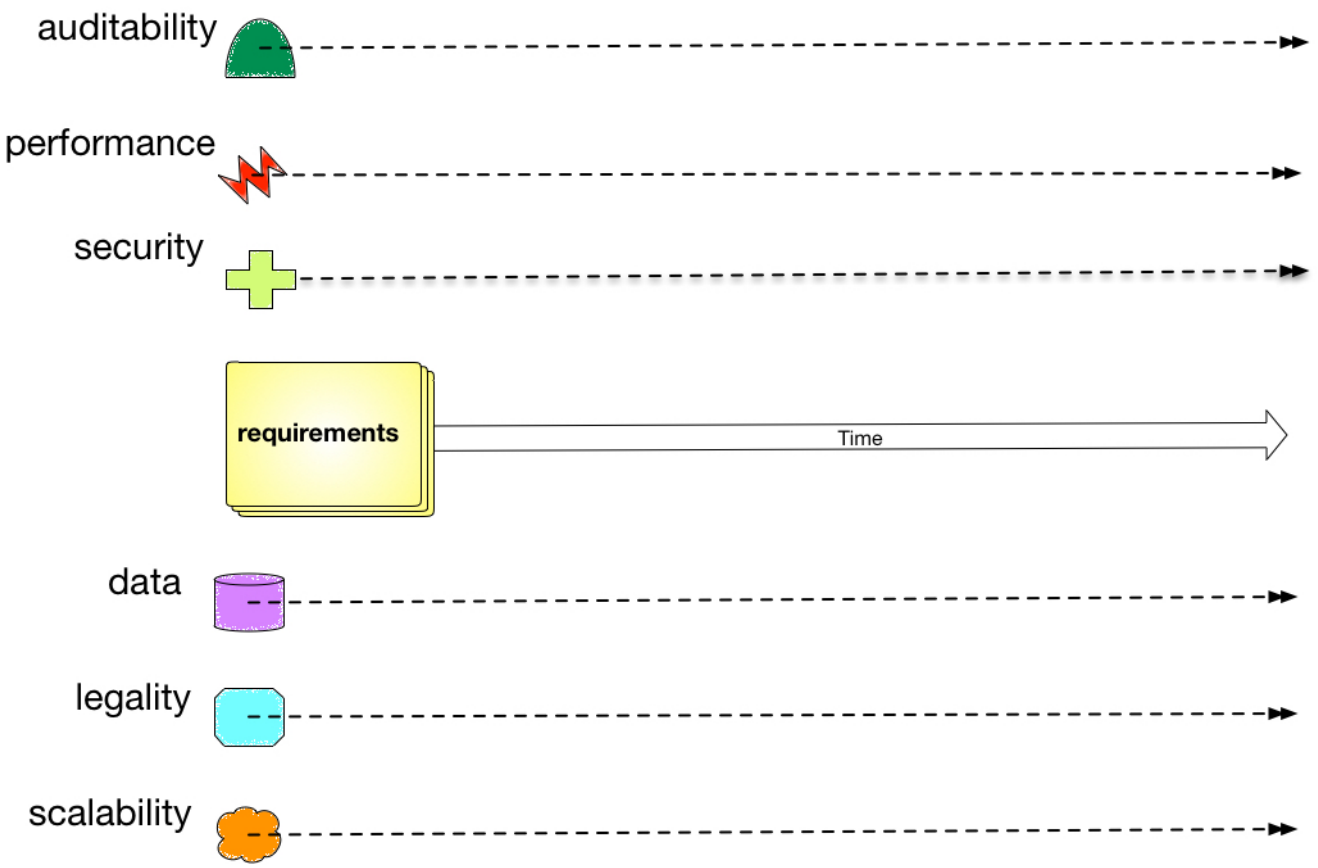
Evolutionary Architecture

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

Evolutionary Architecture

An evolutionary architecture supports
guided,
incremental change
across multiple dimensions

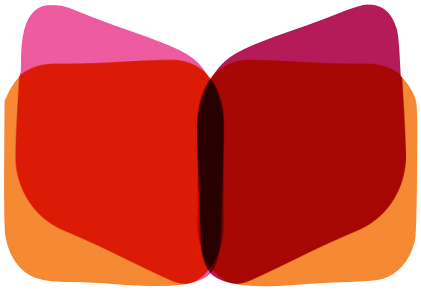
multiple dimensions



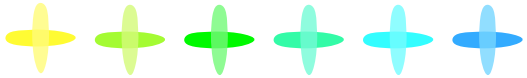
Evolutionary Architecture

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

Agenda Part 1

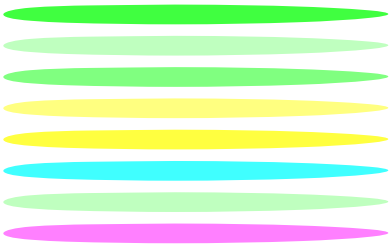


Definition



Incremental Change

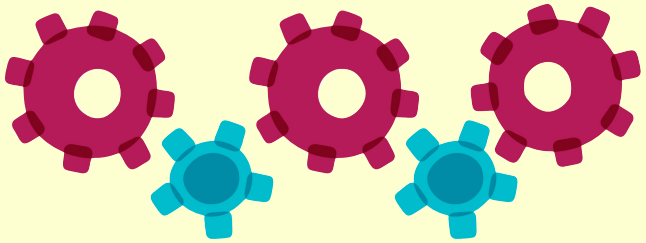
Guided Change via Fitness Functions



Structuring architecture for evolution

Agenda Part 2

The Evolution of Penultimate

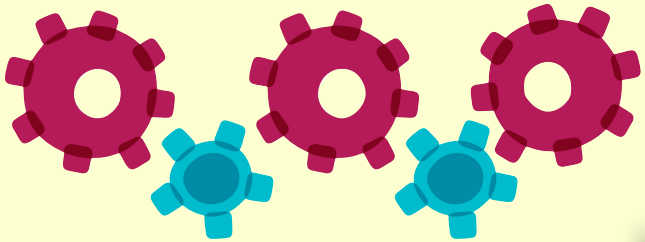


Agenda Part 2

fitness functions

architecture migration

The Evolution of Penultimate



experimentation

automating governance

hypothesis/data driven development

Evolutionary Architecture

An evolutionary architecture supports

guided

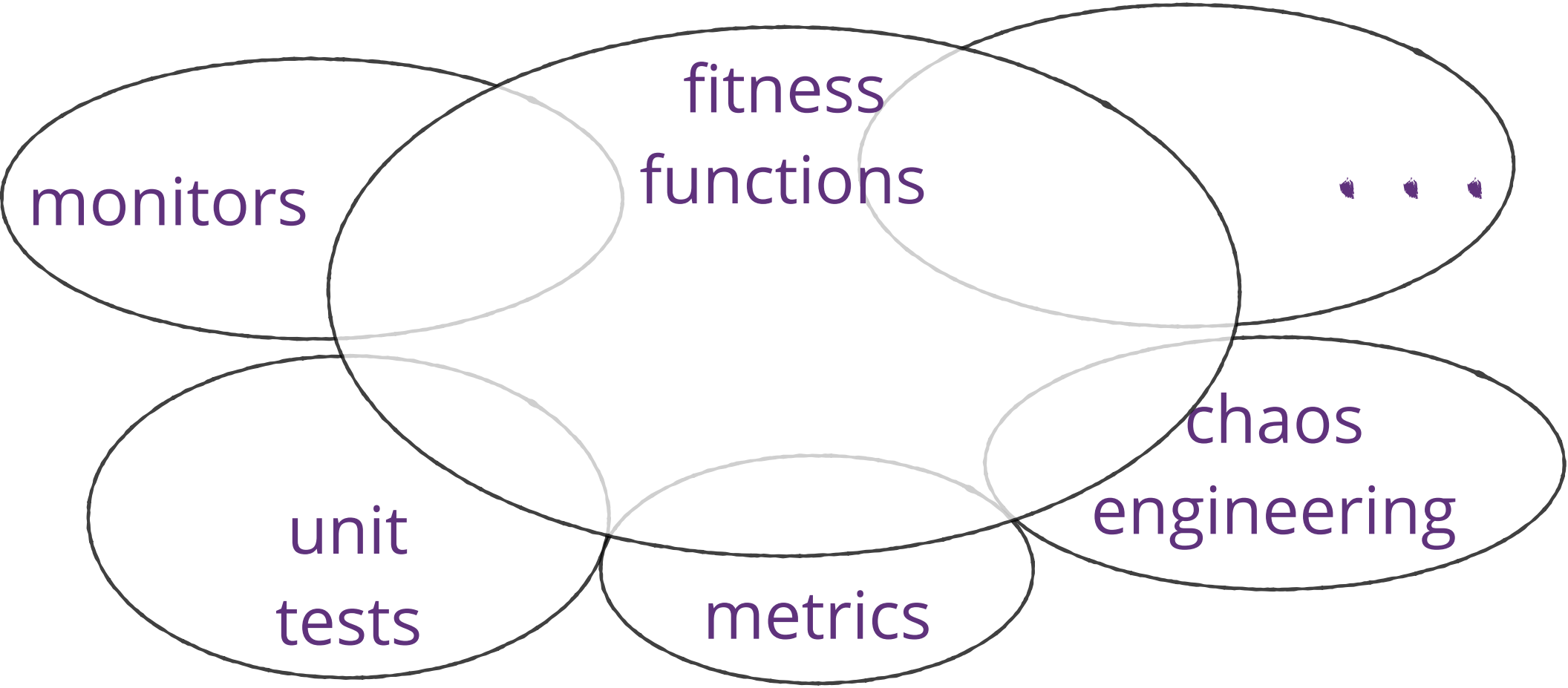
incremental change

across multiple dimensions.

***architectural** fitness function*:

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

Fitness Functions



Categories of Fitness Functions

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

Categories of Fitness Functions

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

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

Categories of Fitness Functions

run based on a particular event:

— developer executing a unit test

— deployment pipeline running tests

— timed task

Categories of Fitness Functions

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

executes constant verification of architectural aspect(s)

Categories of Fitness Functions

static 

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

Categories of Fitness Functions

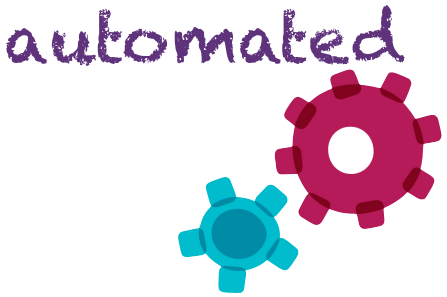
static 

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

dynamic 

rely on a shifting definition based on extra context.

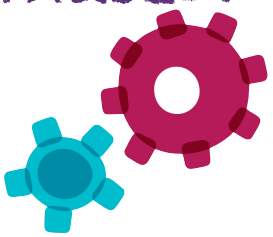
Categories of Fitness Functions



tests and other verification mechanism that run without human interaction.

Categories of Fitness Functions

automated



tests and other verification mechanism that run without human interaction.

manual

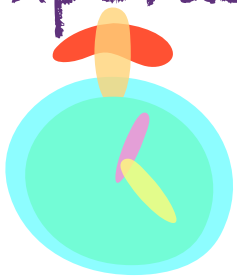
must involve at least one human.

Categories of Fitness Functions

architects may want to build a time component into assessing fitness

Categories of Fitness Functions

temporal

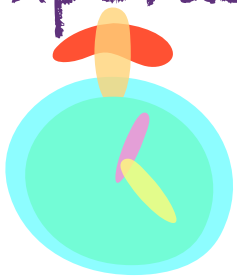


architects may want to build a time component into assessing fitness

break on upgrade

Categories of Fitness Functions

temporal



architects may want to build a time component into assessing fitness

break on upgrade

overdue library update

Categories of Fitness Functions

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

Categories of Fitness Functions

architectural characteristic

Categories of Fitness Functions

architectural characteristic

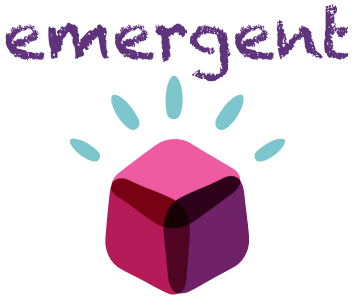
problem domain

Categories of Fitness Functions

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

Categories of Fitness Functions

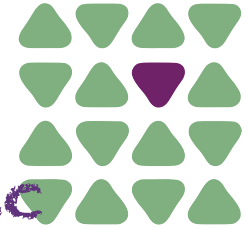
architects will define most fitness functions at project inception as they elucidate the characteristics of the architecture...



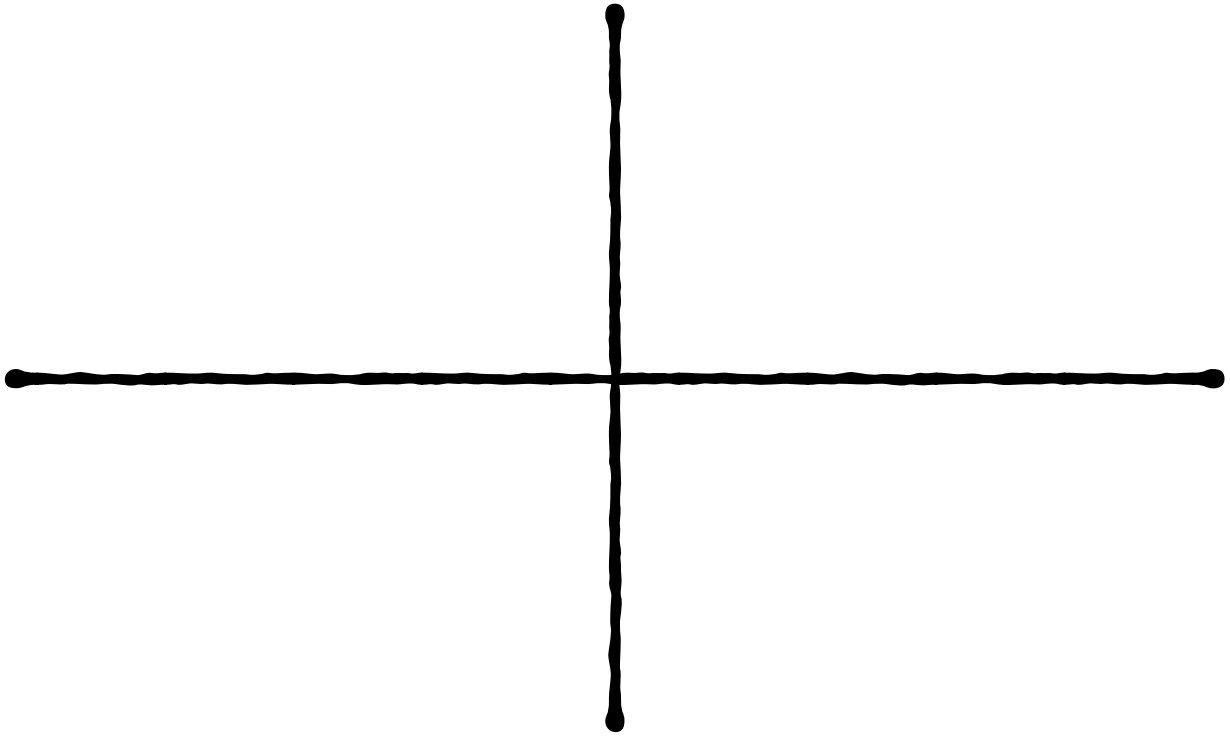
...some fitness functions will emerge during development of the system

Fitness Function

atomic



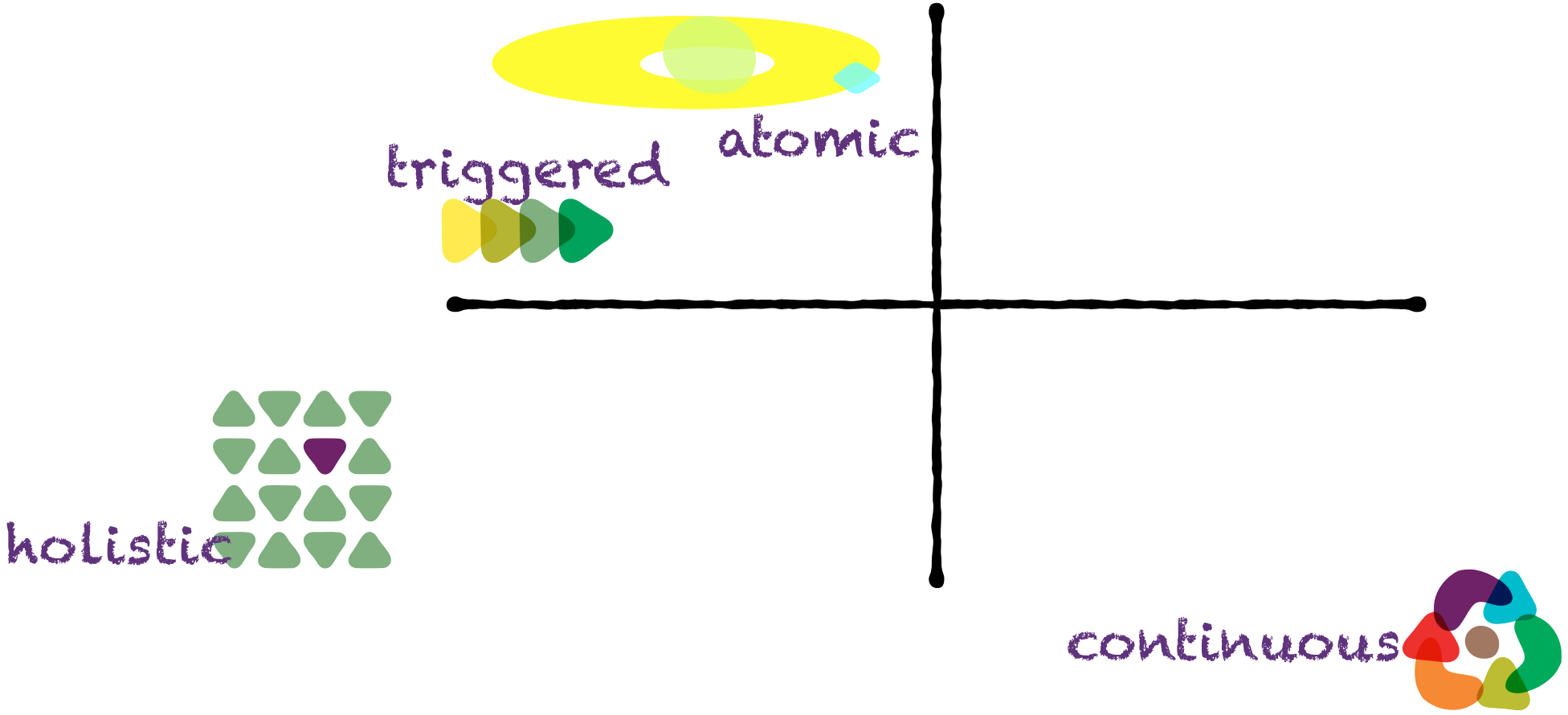
holistic



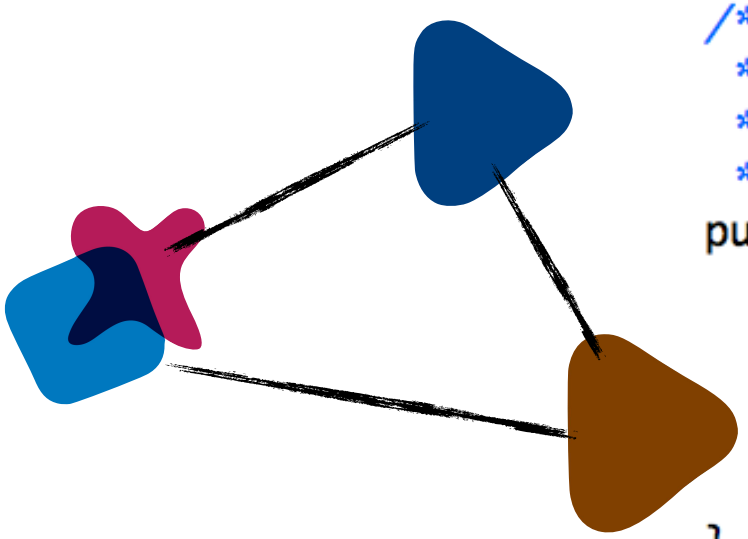
triggered

continuous

Fitness Function



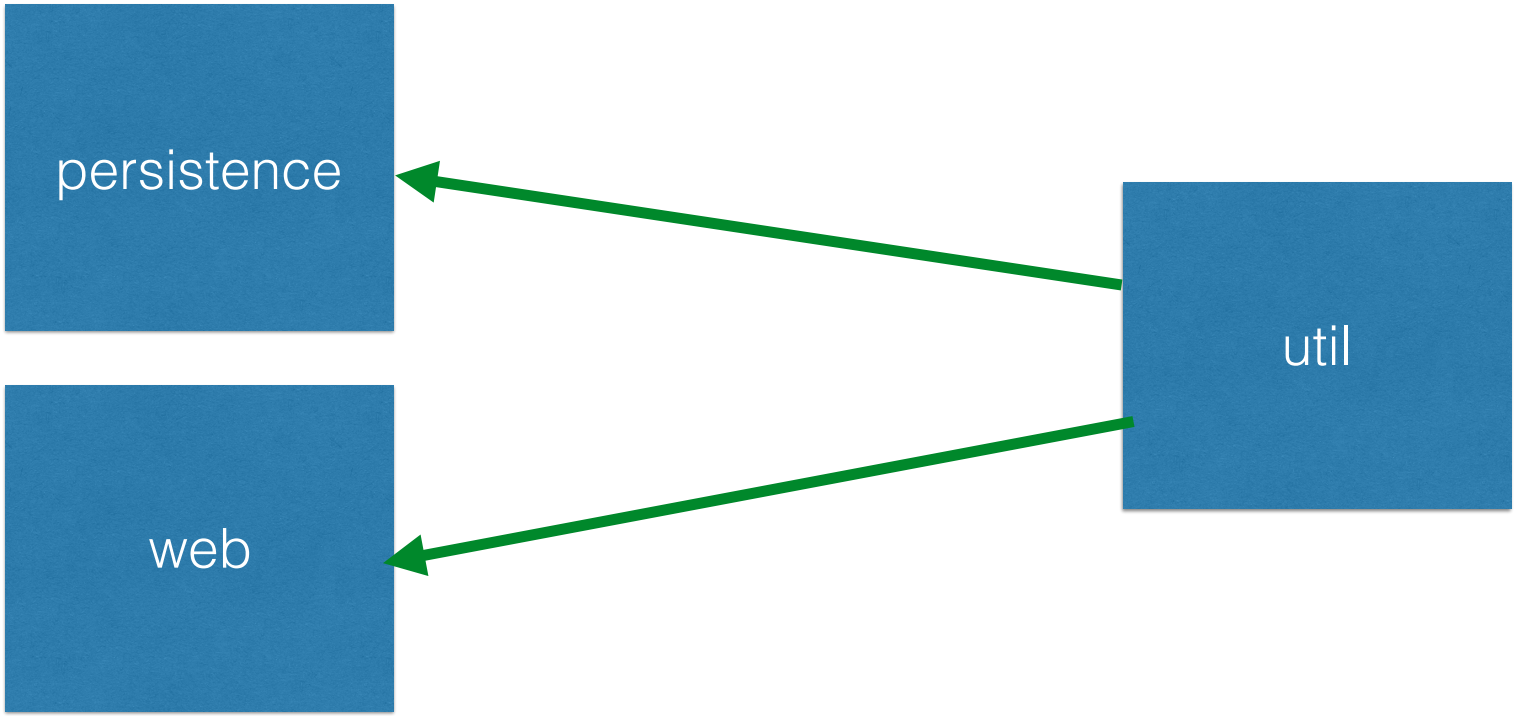
Cyclic Dependency Function



```
/**
 * 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());
}
```

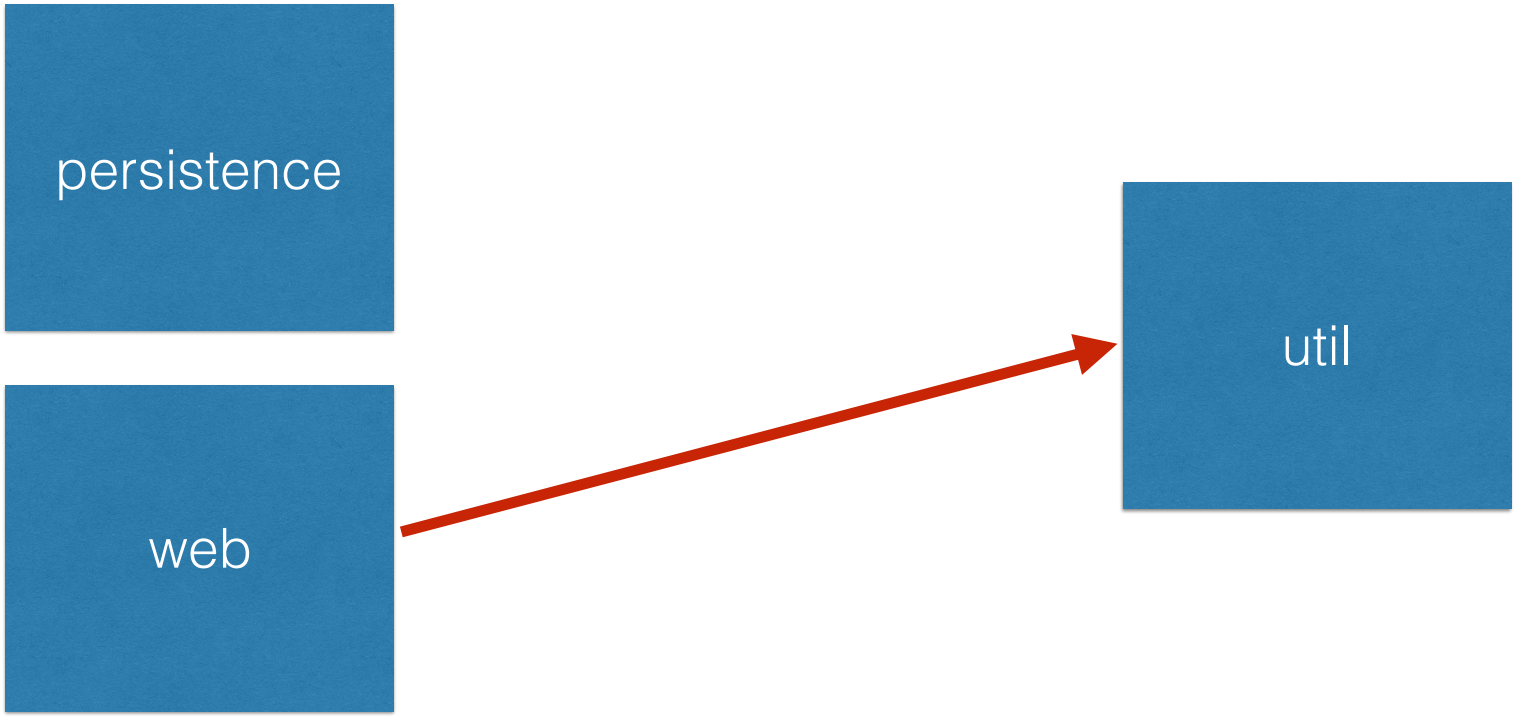
clarkware.com/software/JDepend.html

Directionality of Imports



packages/namespaces

Directionality of Imports

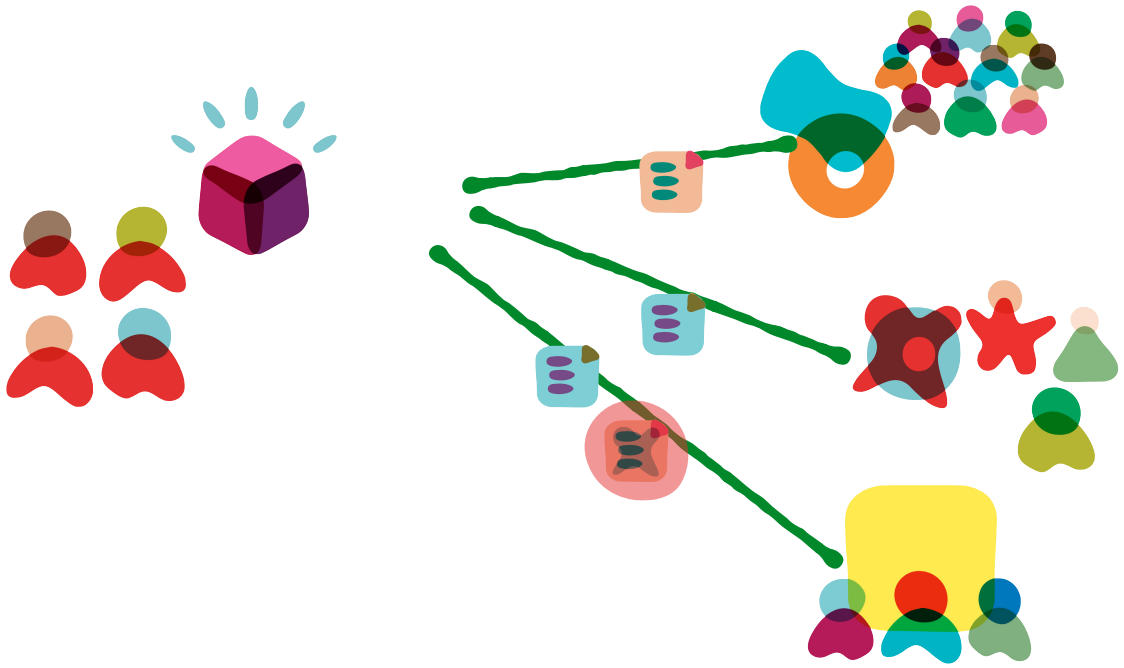


packages/namespaces

Coupling Fitness Function

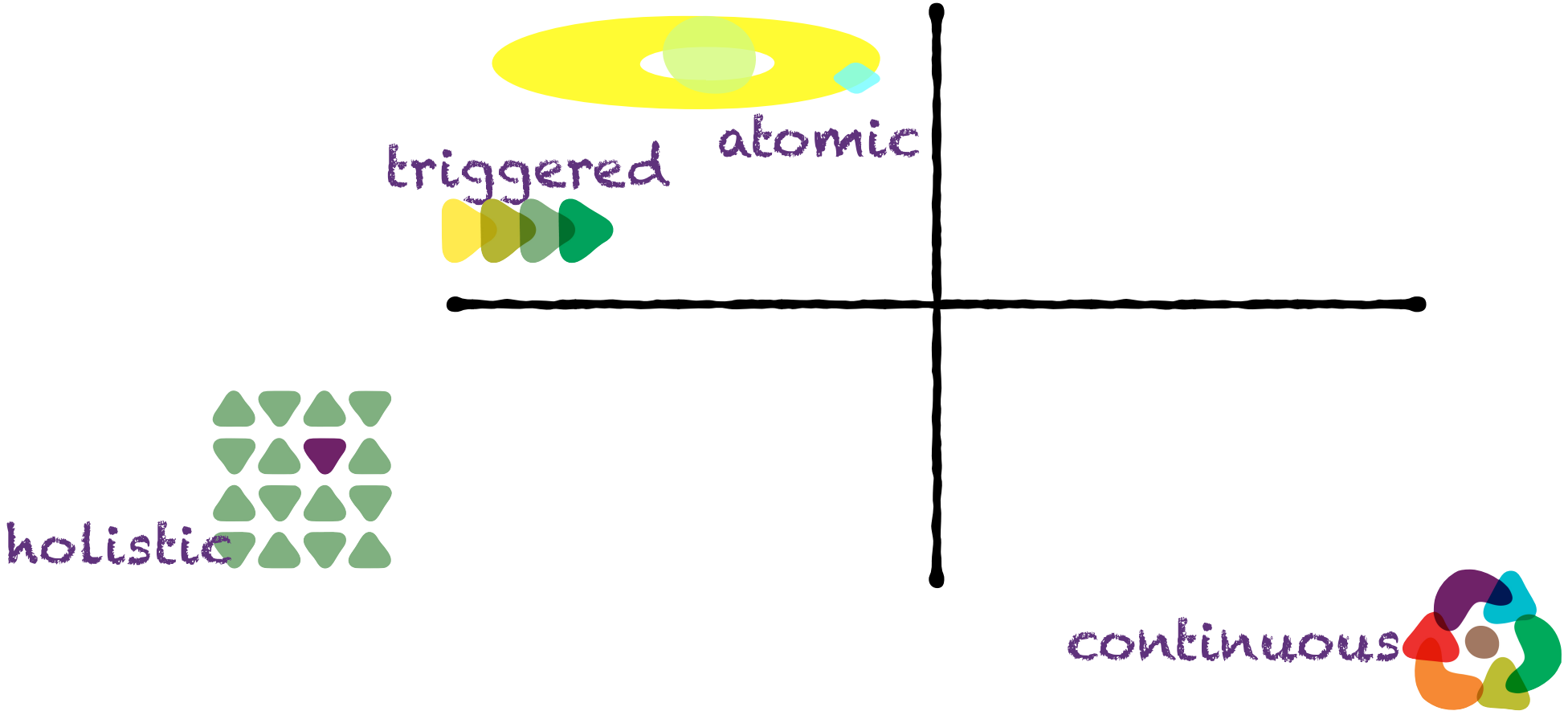
```
public void testMatch() {
    DependencyConstraint constraint = new DependencyConstraint();

    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



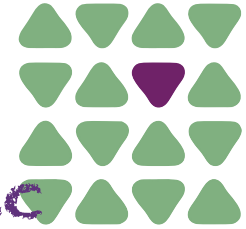
martinfowler.com/articles/consumerDrivenContracts.html

Fitness Function

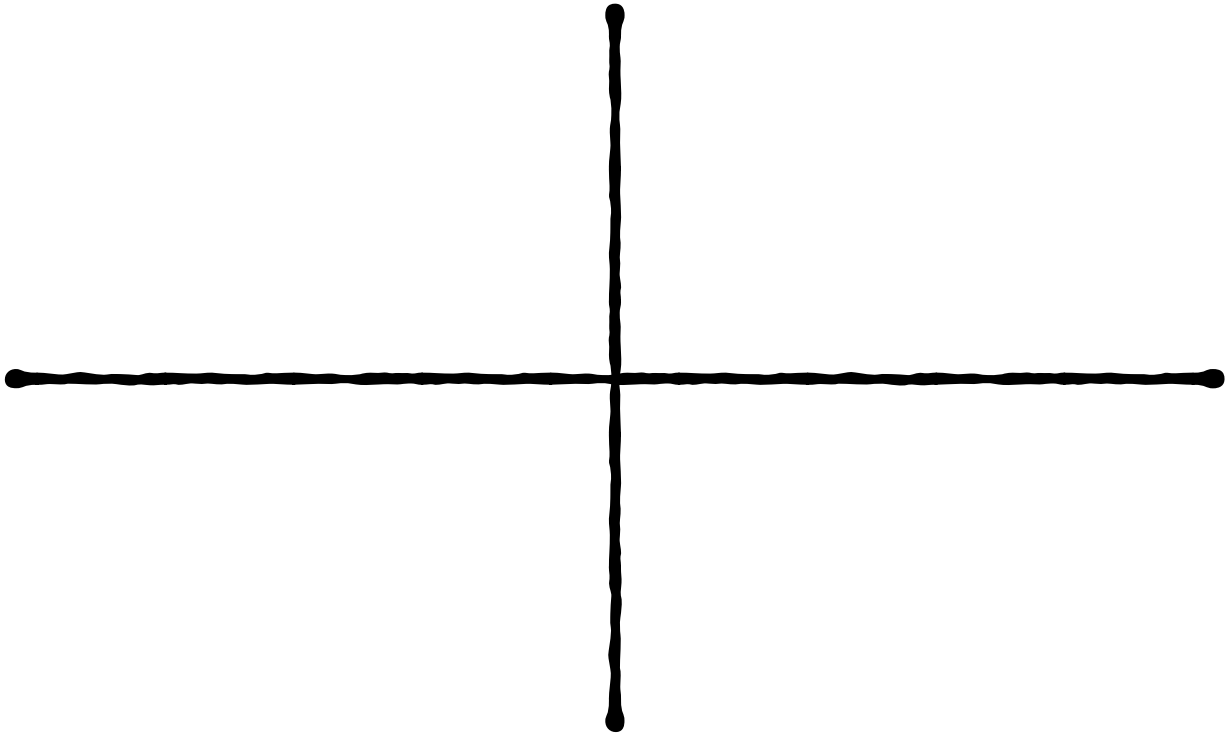


Fitness Function

atomic



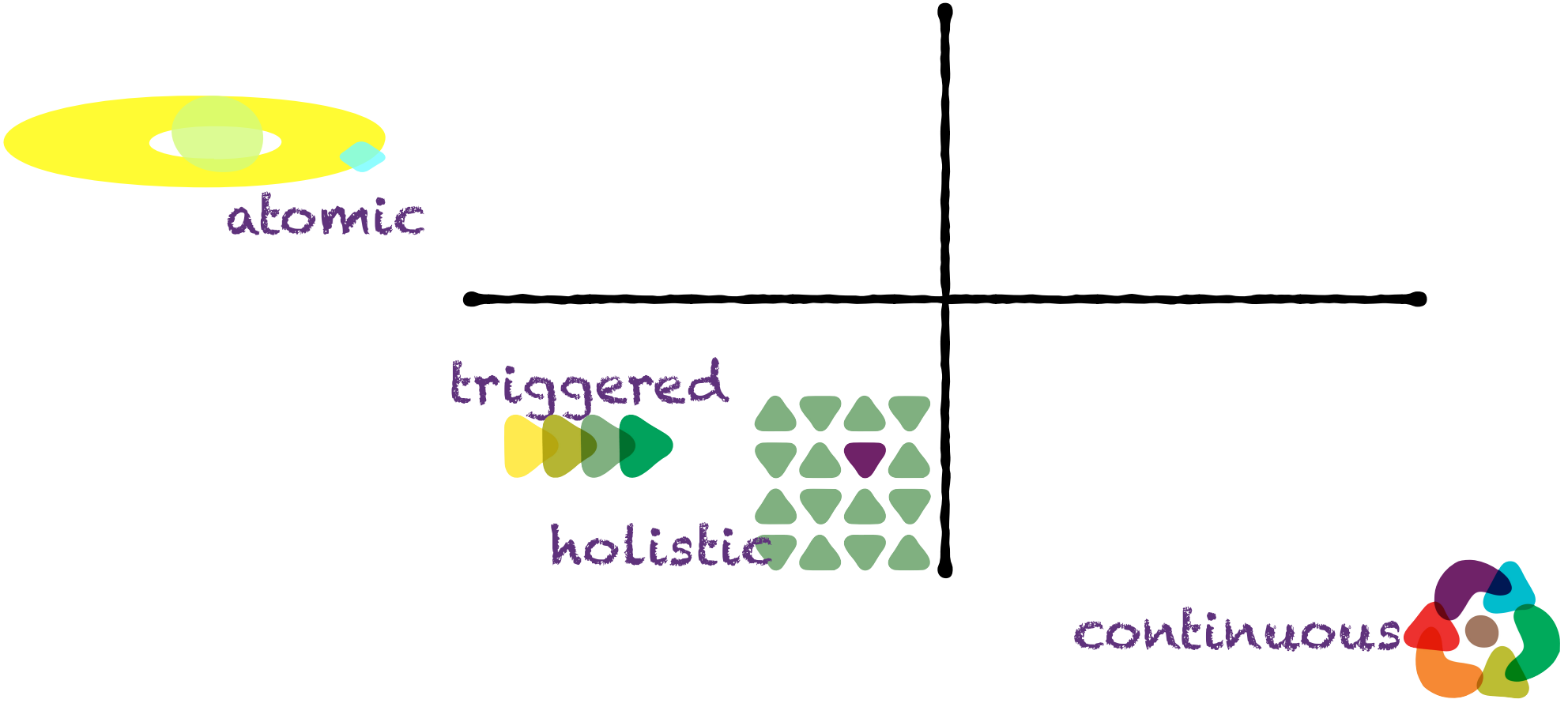
holistic



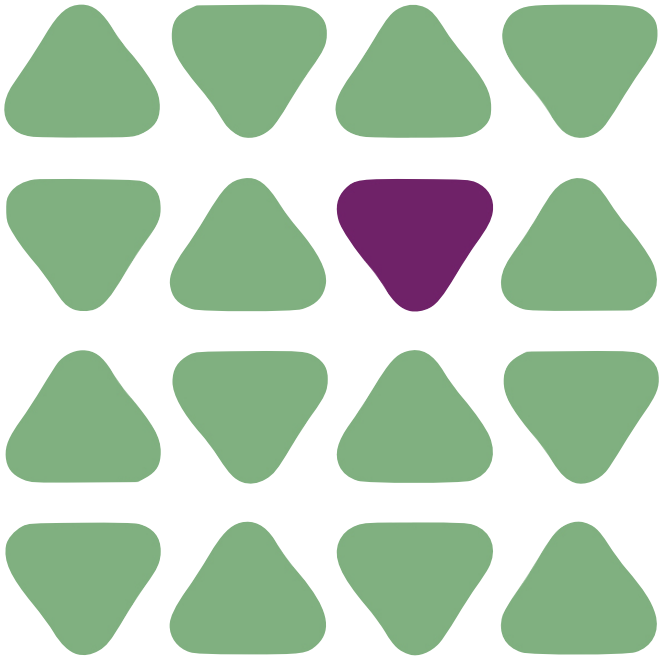
triggered

continuous

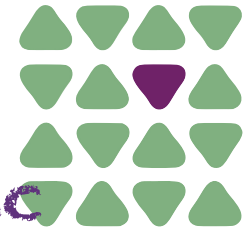
Fitness Function



triggered

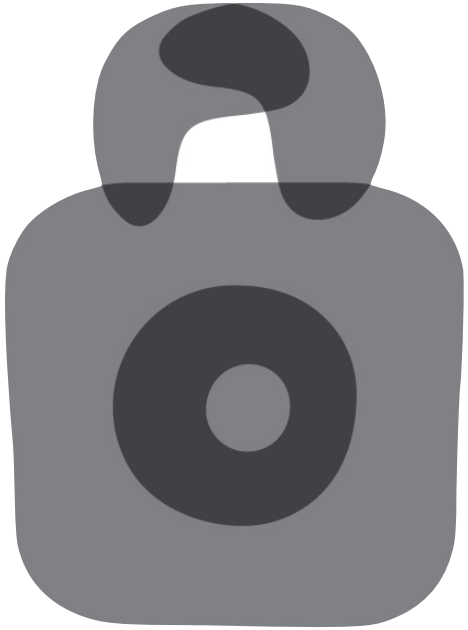


holistic



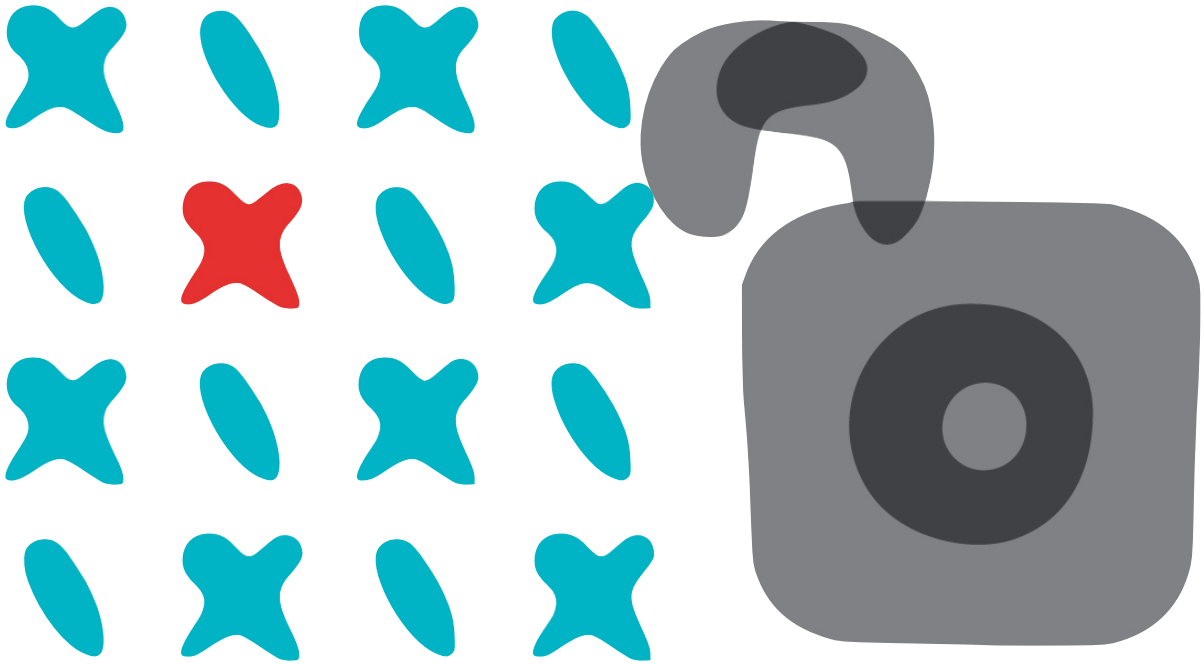
triggered

holistic

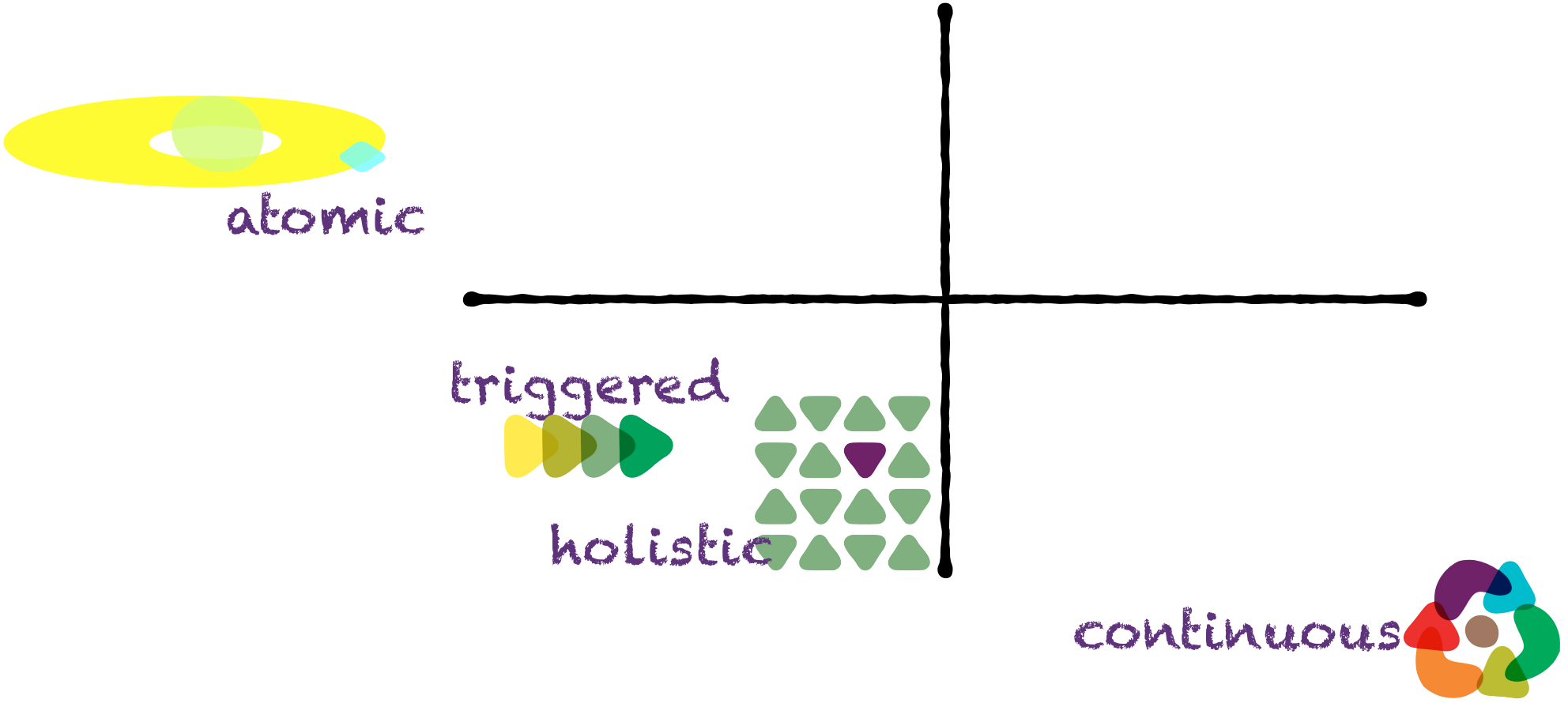


triggered

holistic

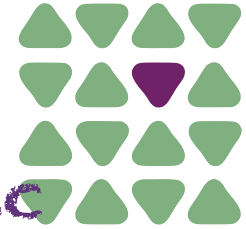


Fitness Function

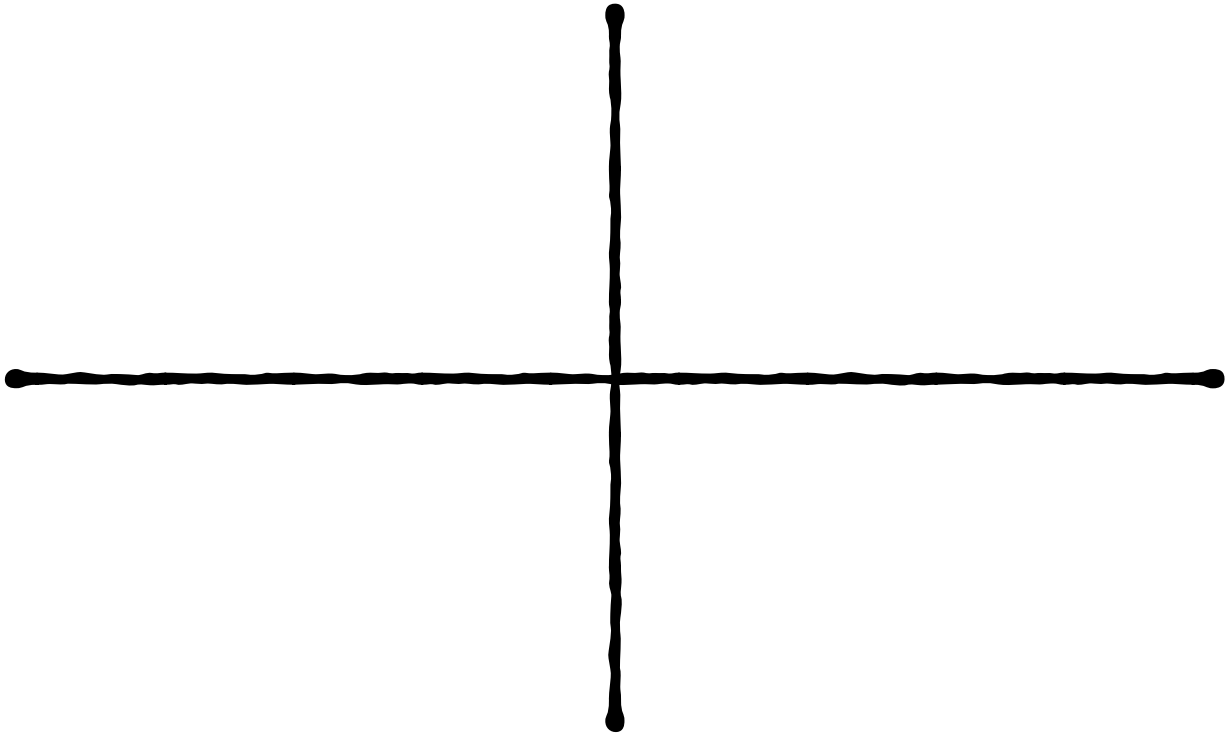


Fitness Function

atomic



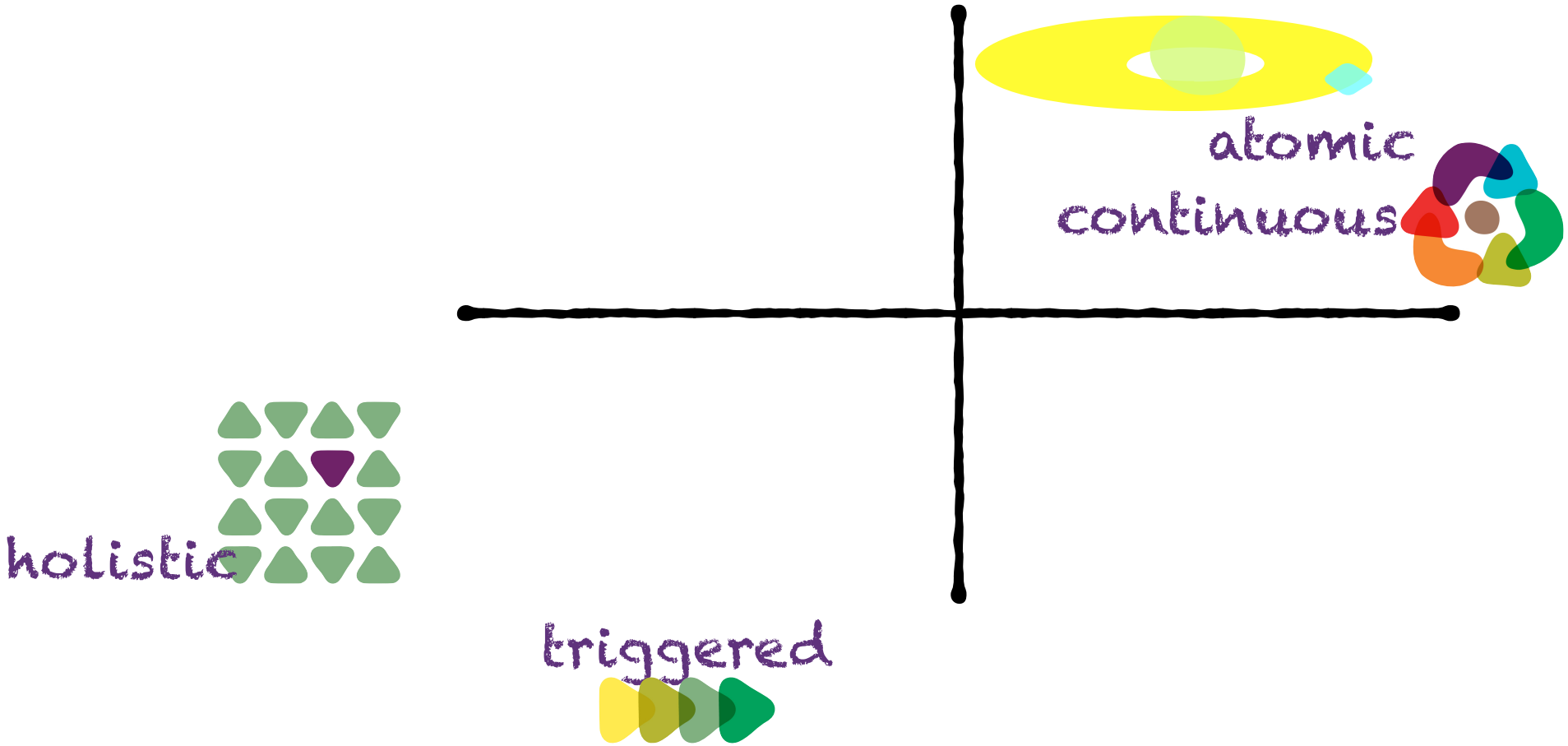
holistic



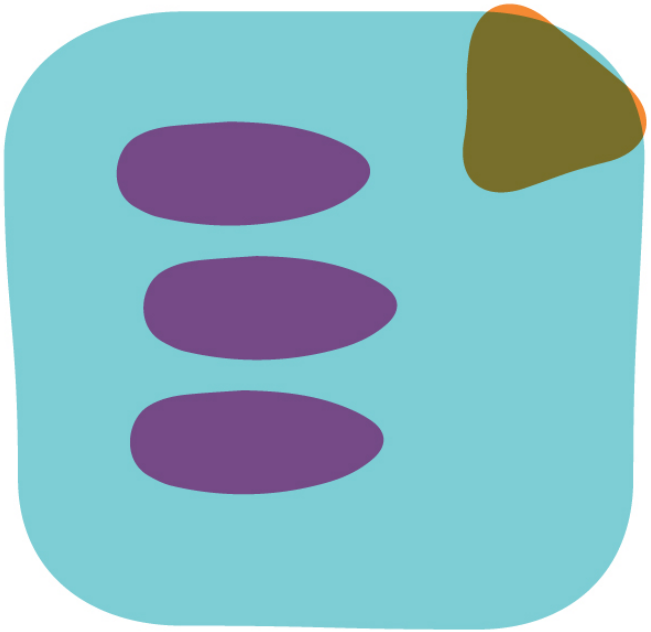
triggered

continuous

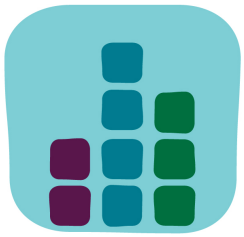
Fitness Function



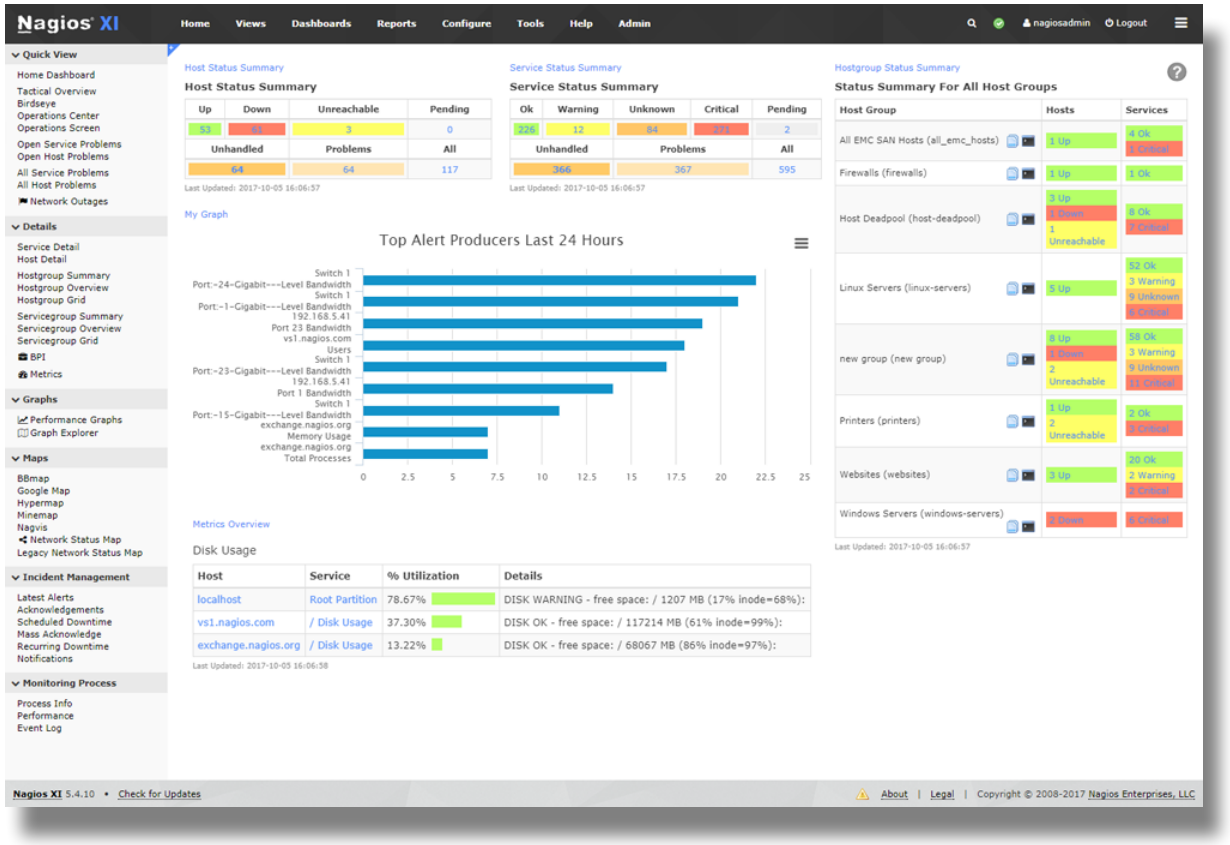
monitoring

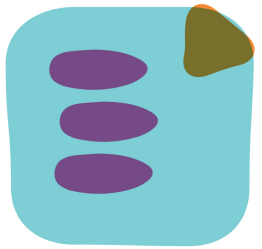


logging



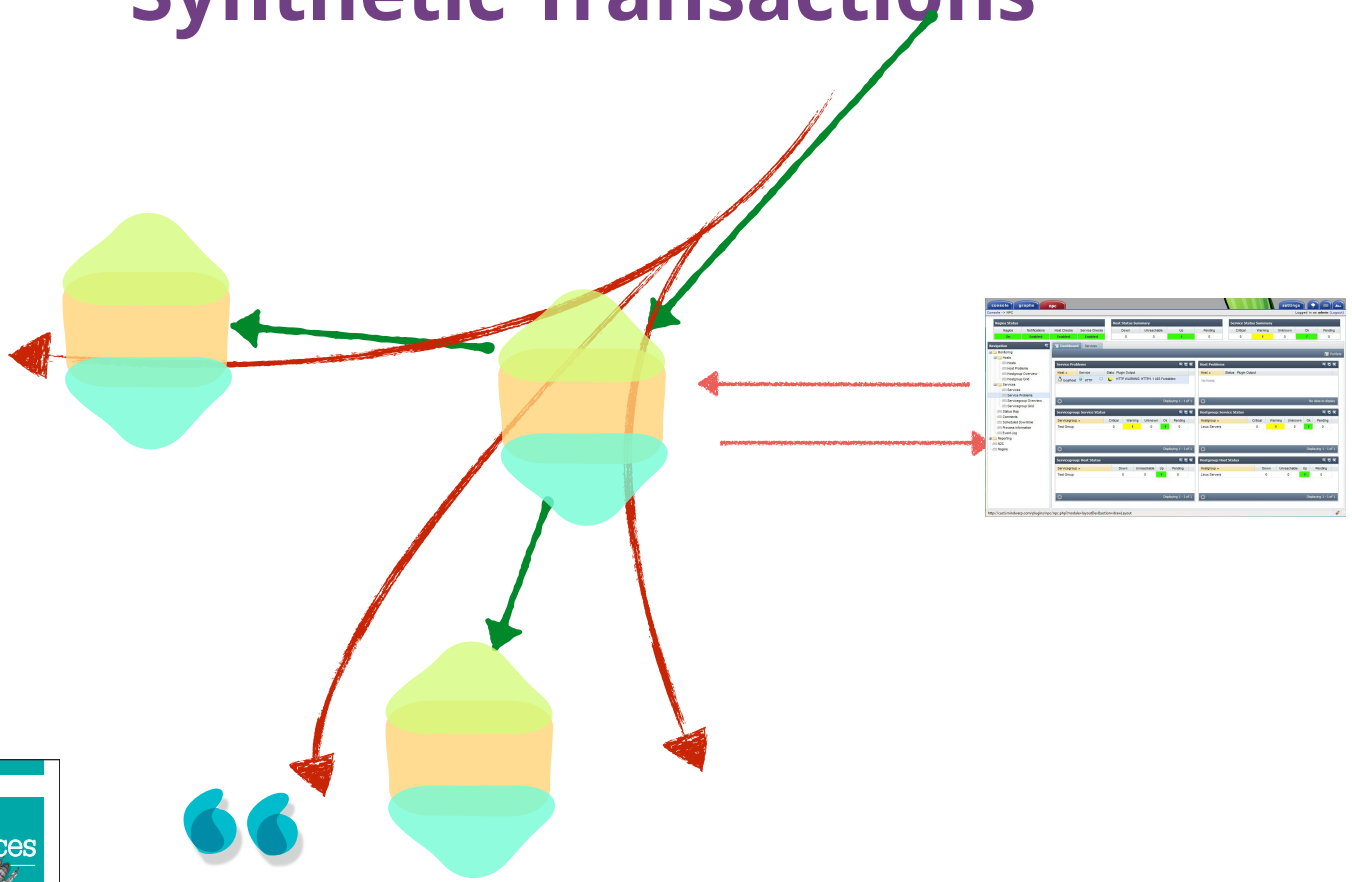
monitoring





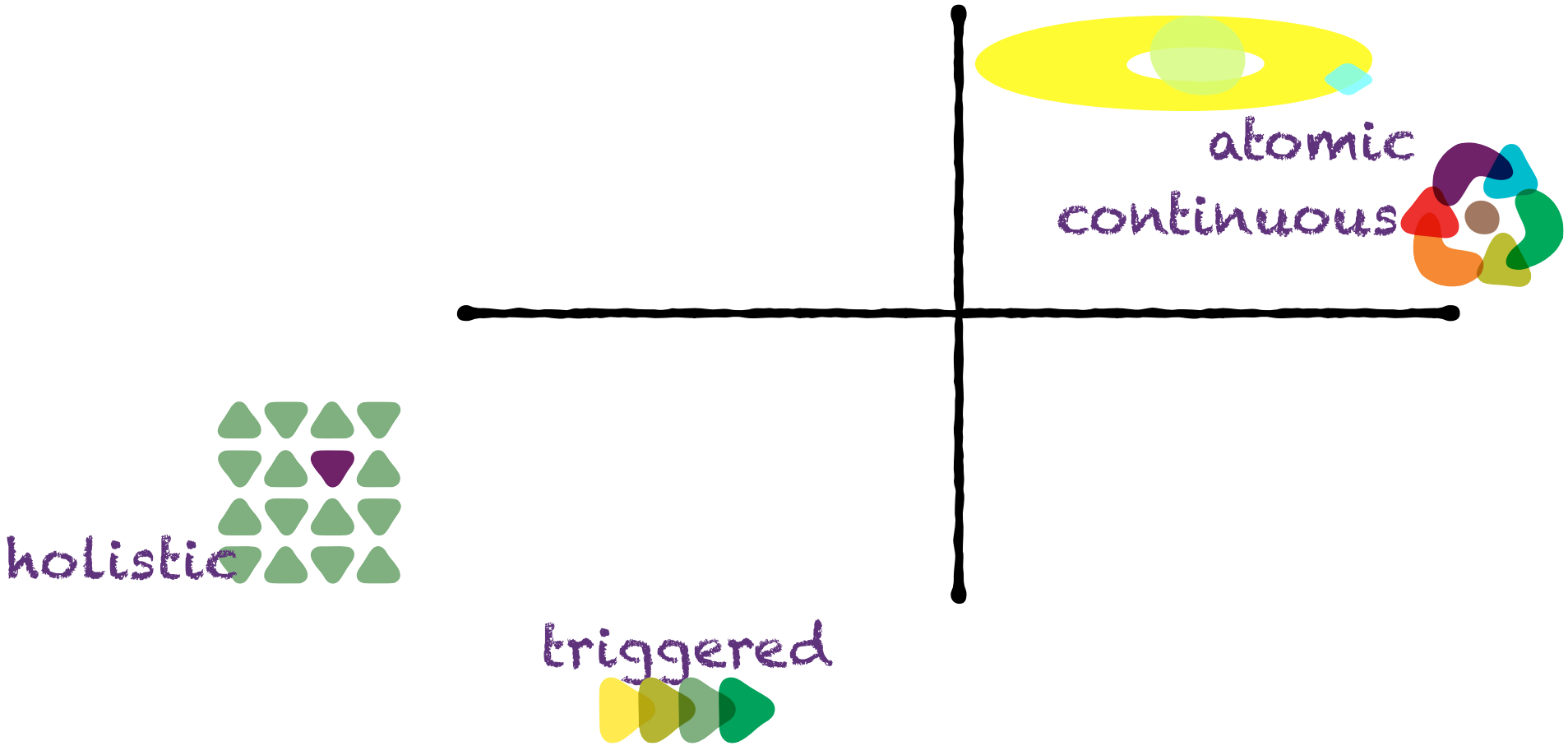
logging

Synthetic Transactions



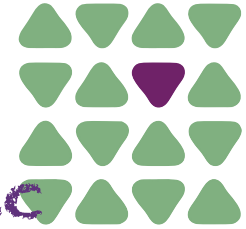
Use synthetic transactions to test production systems.

Fitness Function

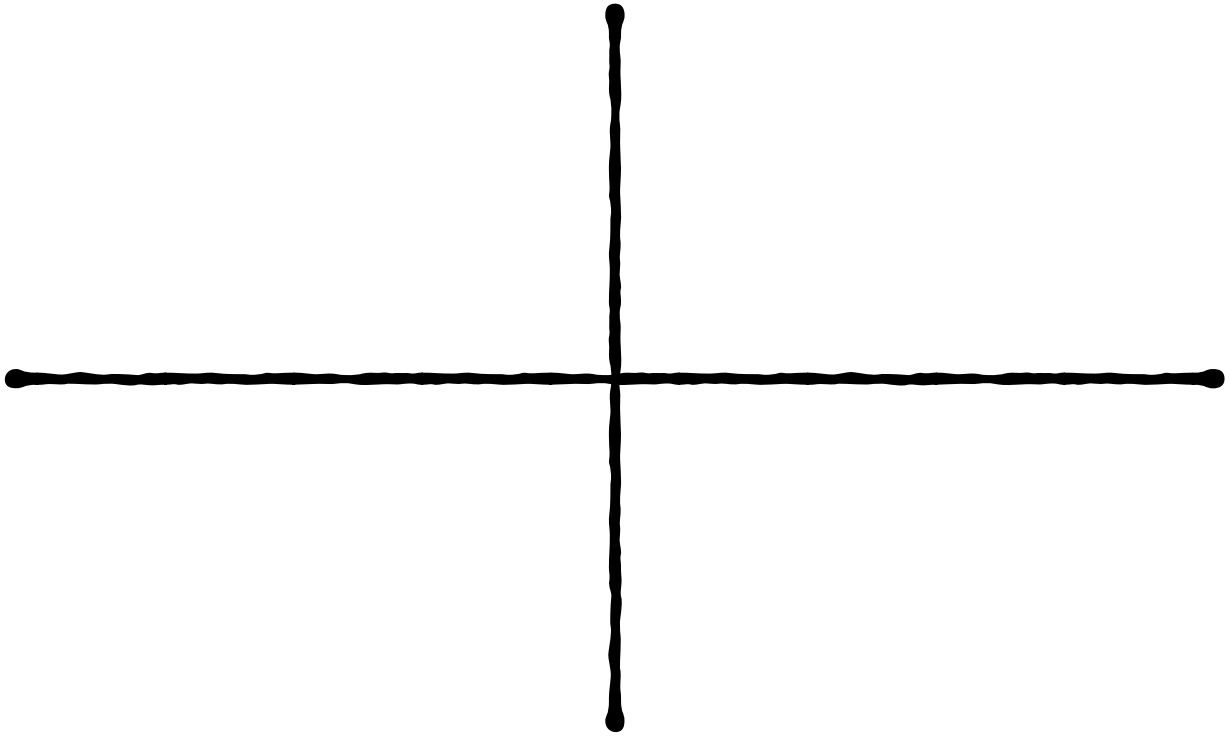


Fitness Function

atomic



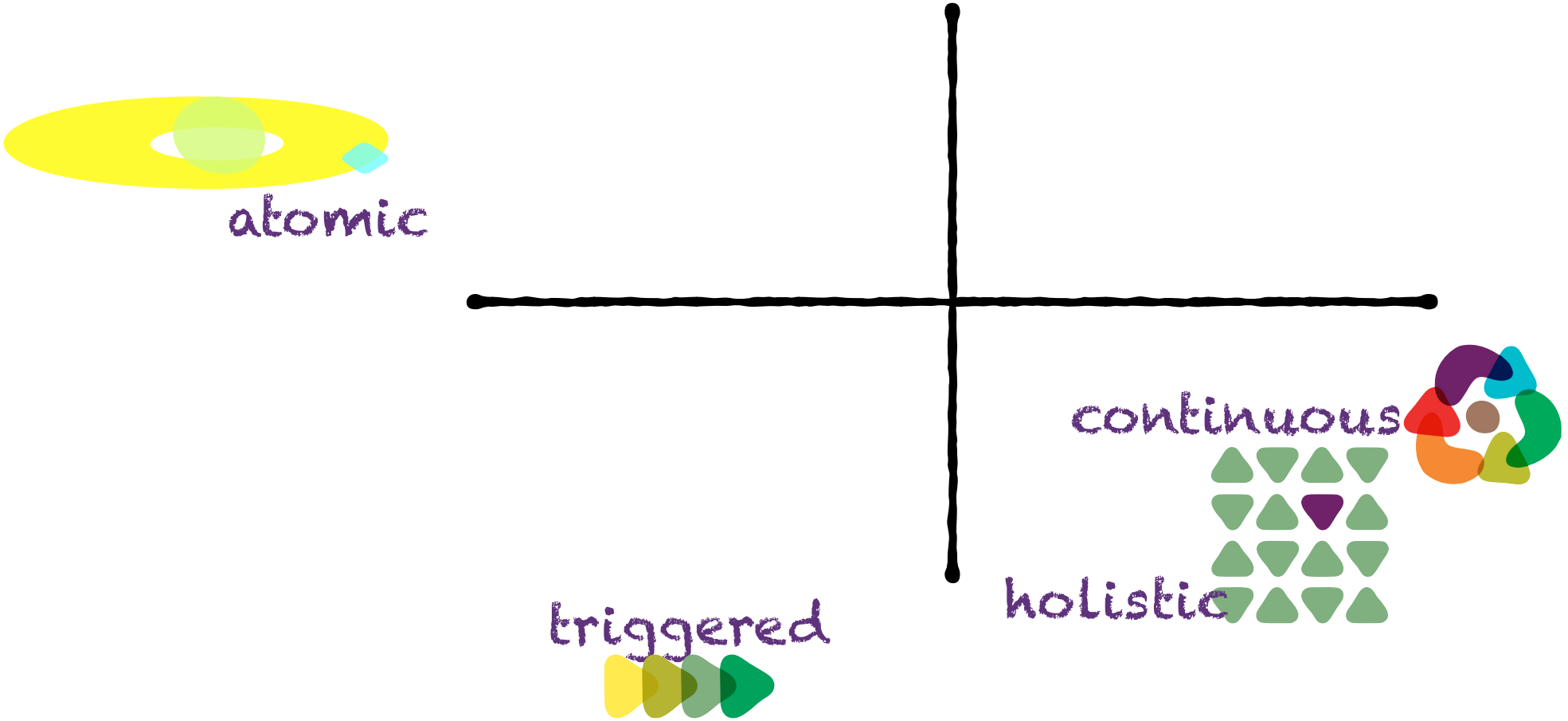
holistic



triggered

continuous

Fitness Function

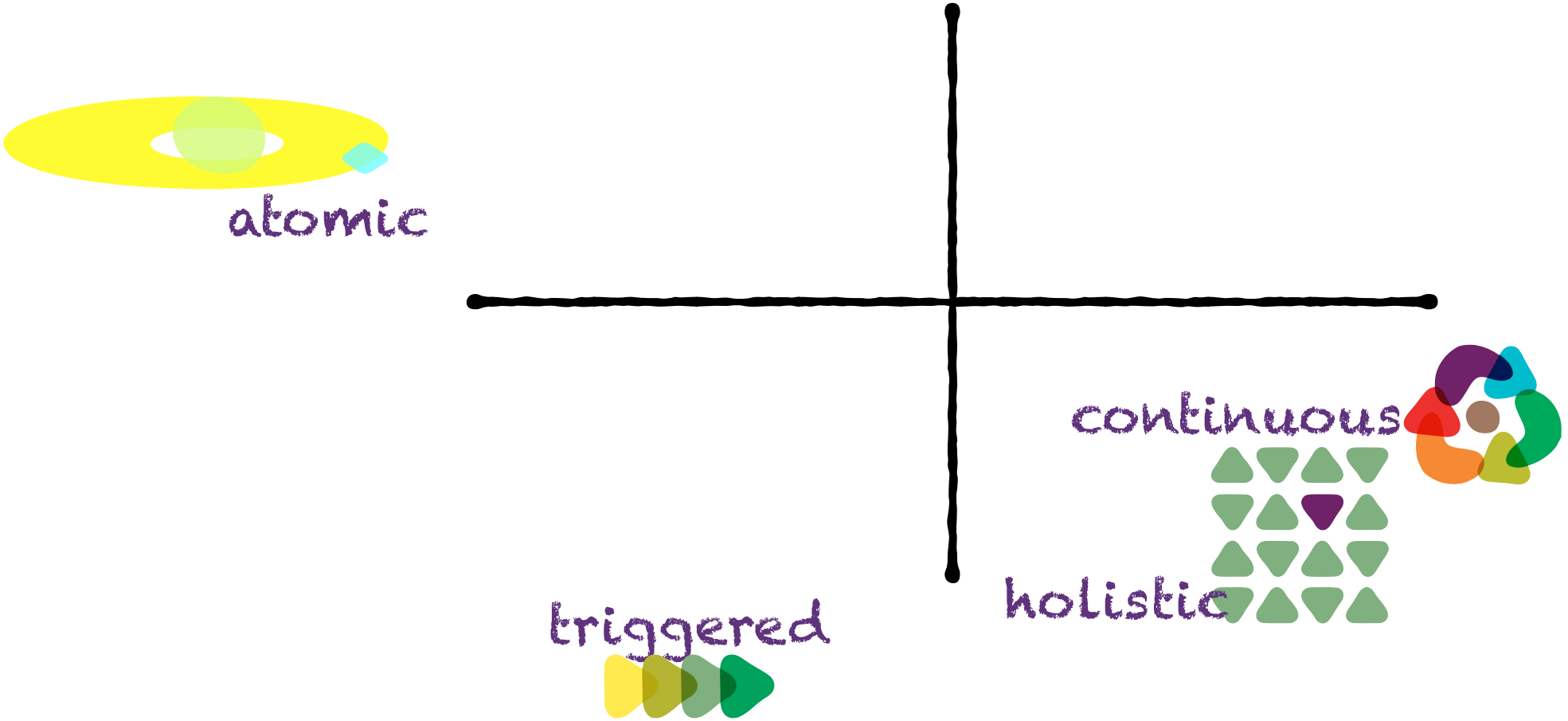



amazon
web services
CHAOS
MONKEY


amazon
web services
CHAOS
MONKEY

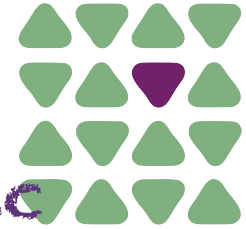

SIMIAN
ARMY

Fitness Function

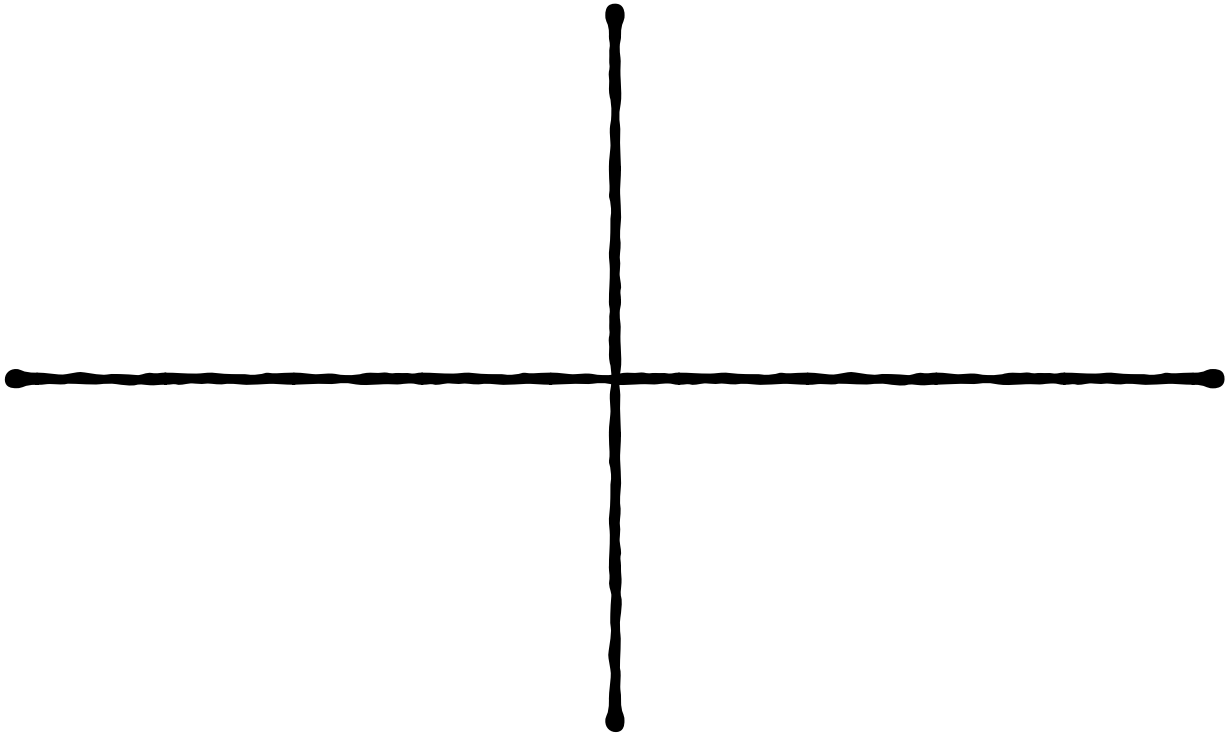


Fitness Function

atomic



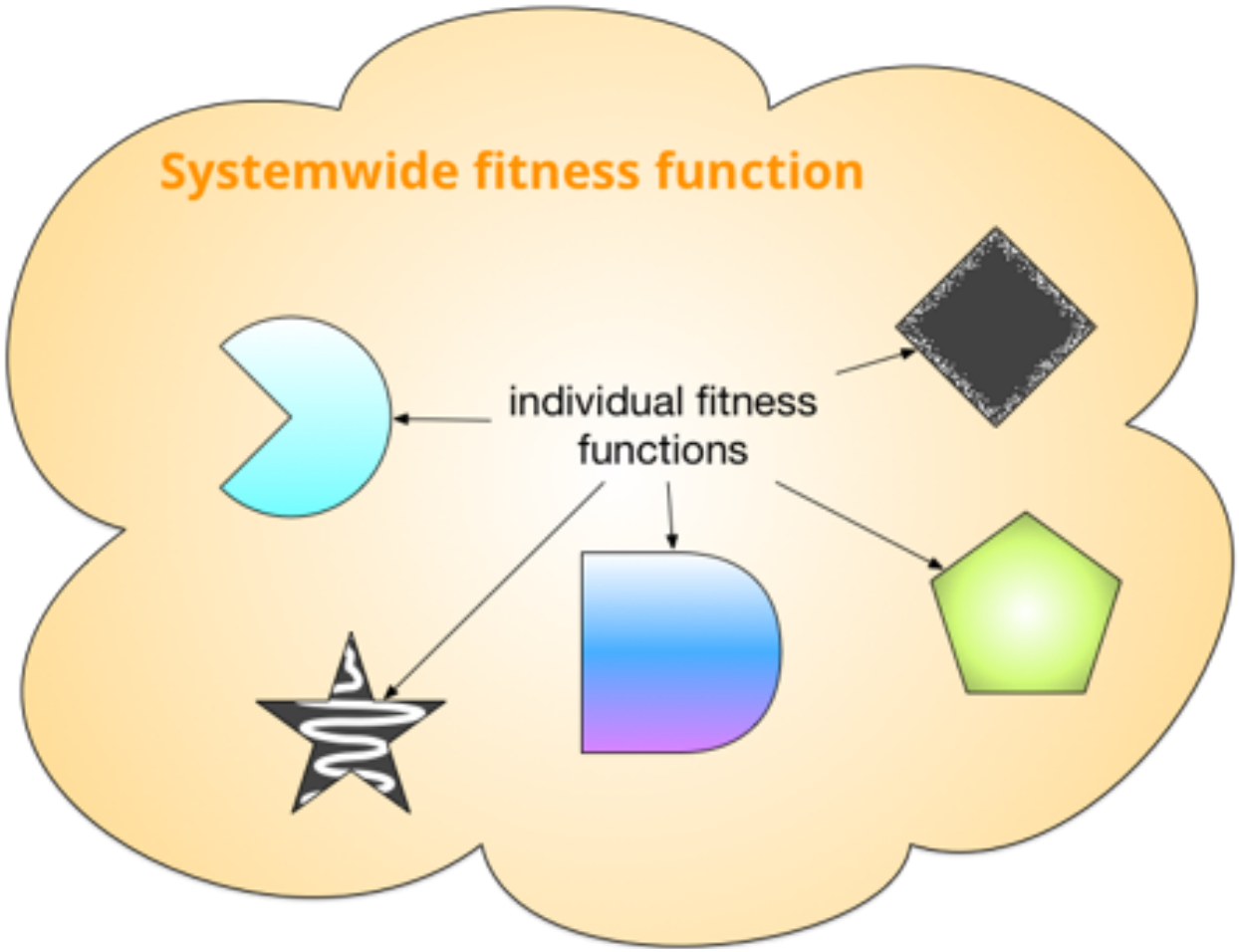
holistic



triggered

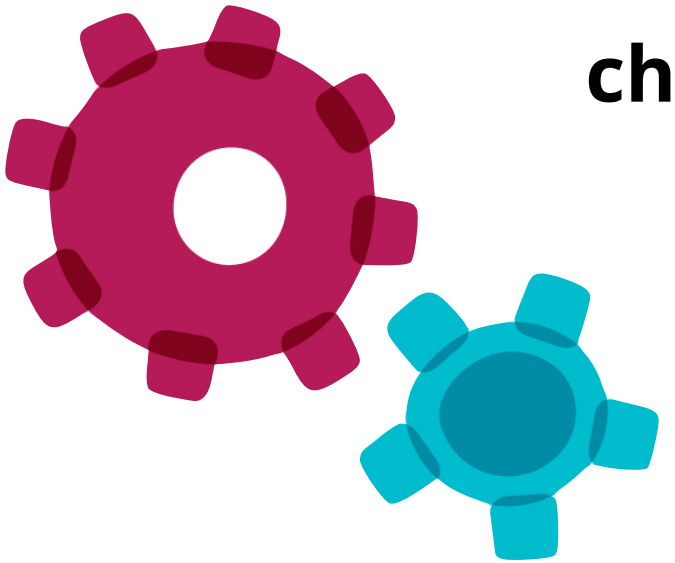
continuous

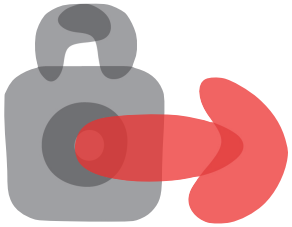
System-wide Fitness Function



Implementing Fitness Functions

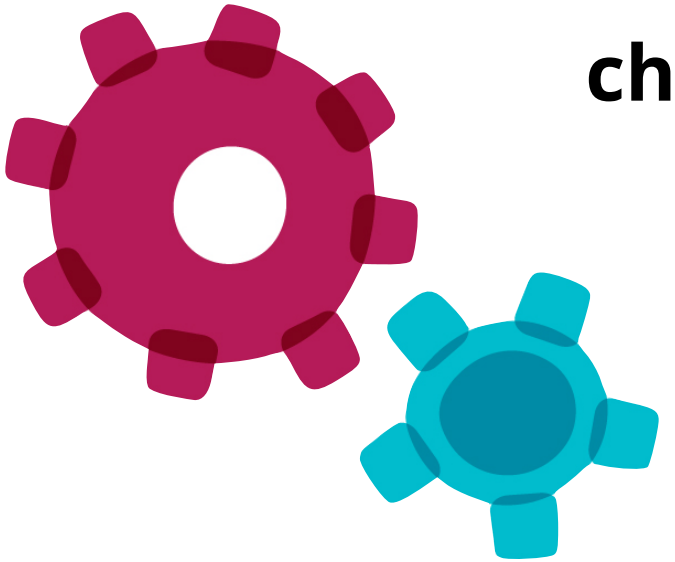
Protecting architectural characteristics



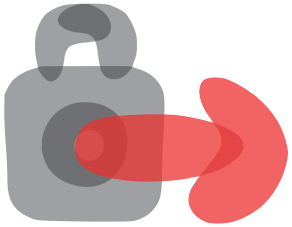


Implementing Fitness Functions

Protecting architectural characteristics



Automating governance



maintainable?

Cyclomatic complexity < 50 for all projects

Naming conventions

Postel's Law

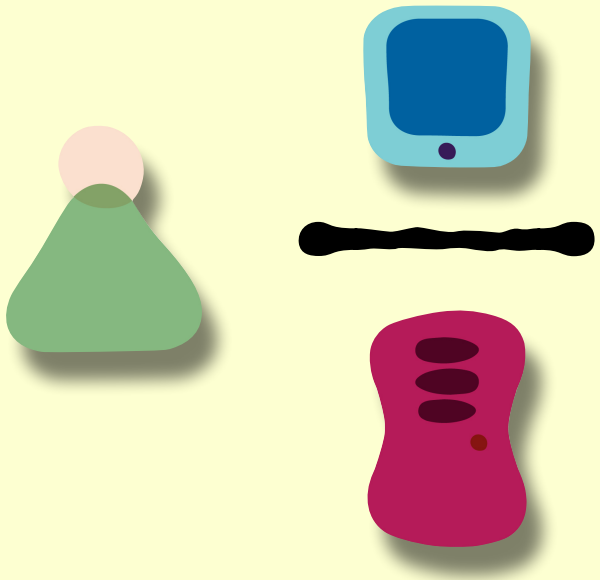
maintainable?

(incoming/outgoing)

Controlled afferent/efferent coupling

Penultimate

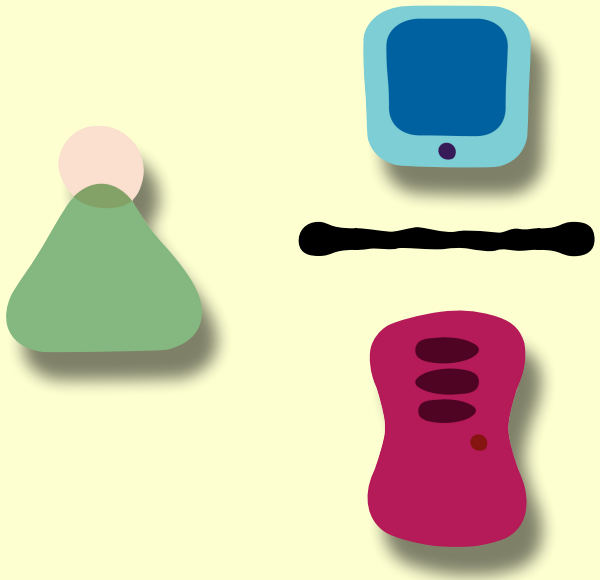
Governing Code Quality



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

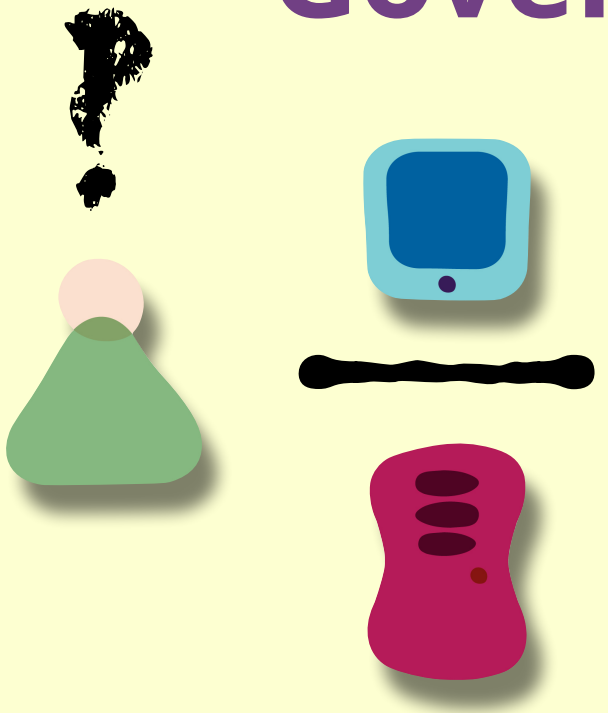
Penultimate

Governing Code Quality



Penultimate

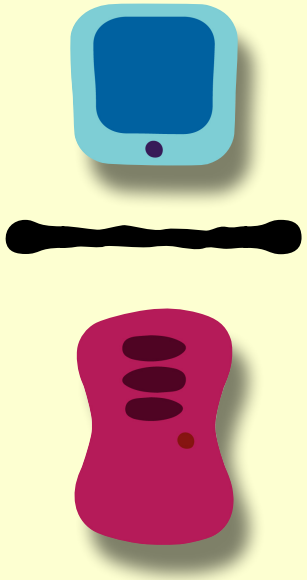
Governing Code Quality



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

Penultimate

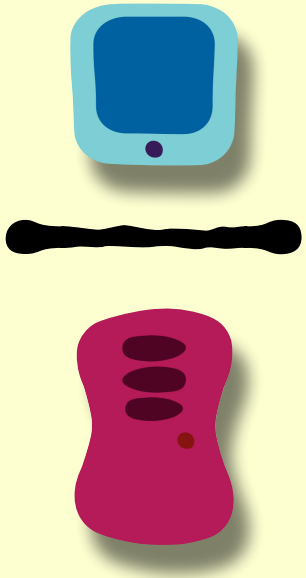
Governing Code Quality



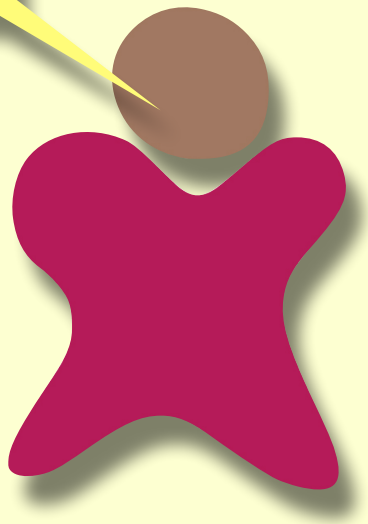
```
if state ==          then
     doSome        AL();
else if s          A" then
     doSo          ();
else if
```

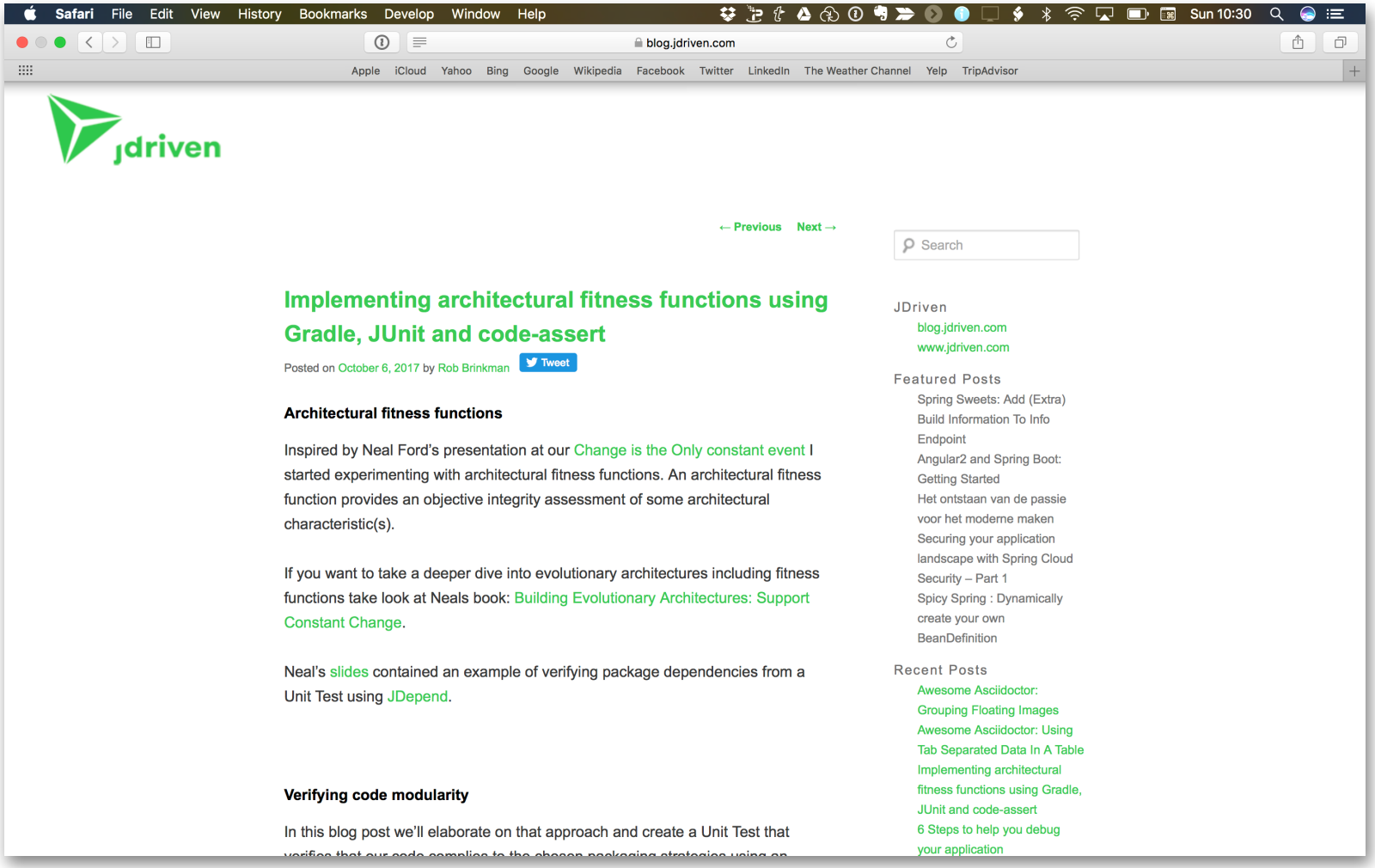
Penultimate

Governing Code Quality

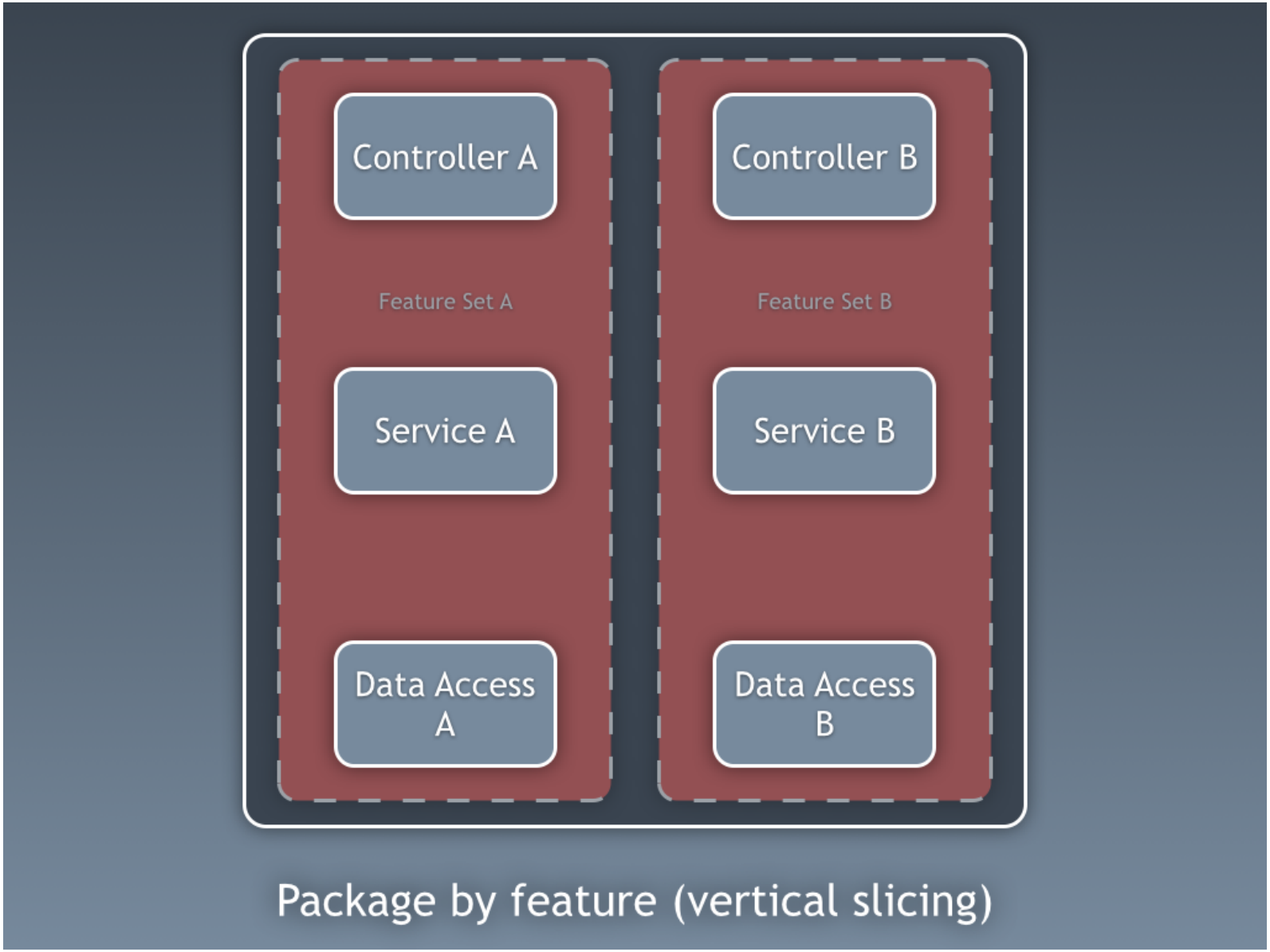


Strategy Design Pattern





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


Controller A
Controller B
Feature Set A
Feature Set B
Service A
Service B
Data Access A
Data Access B
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)
```

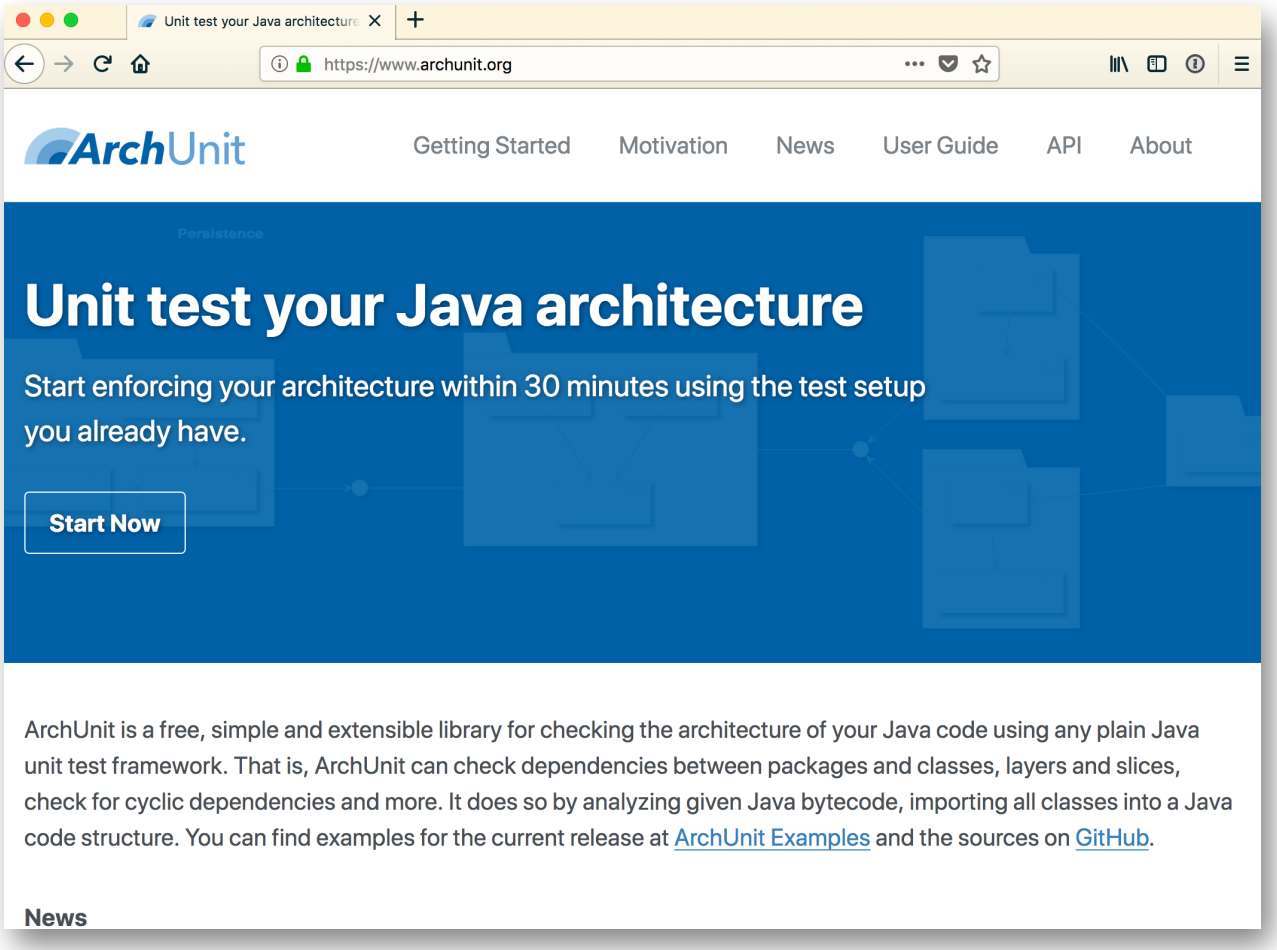
```java
public class ControllerA {

    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)
```

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

ArchUnit

https://www.archunit.org/



ArchUnit

https://www.archunit.org/

coding rules

```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);
    }
}
```

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/

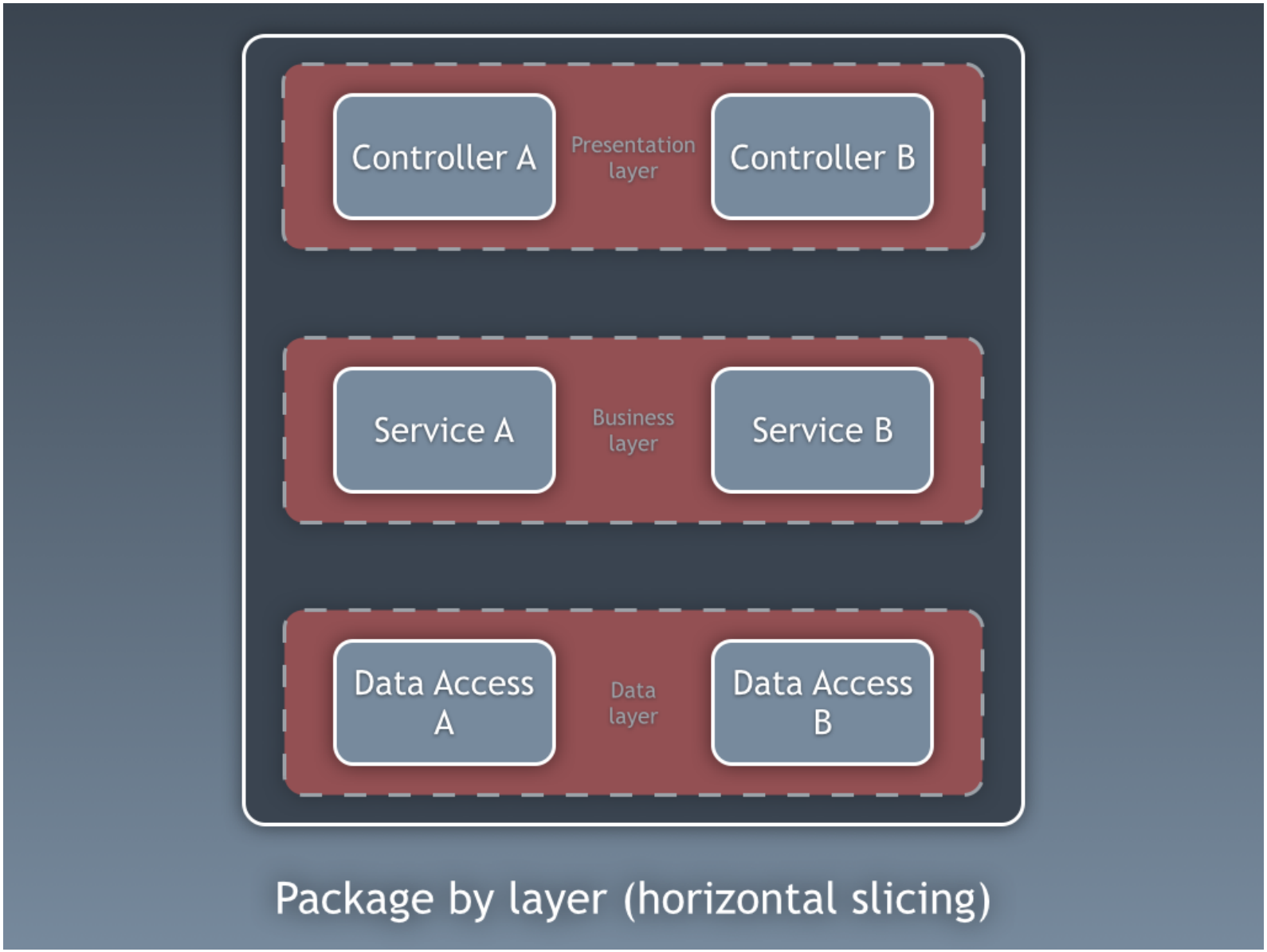
```java
public class InterfaceRules {

    @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

```
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/

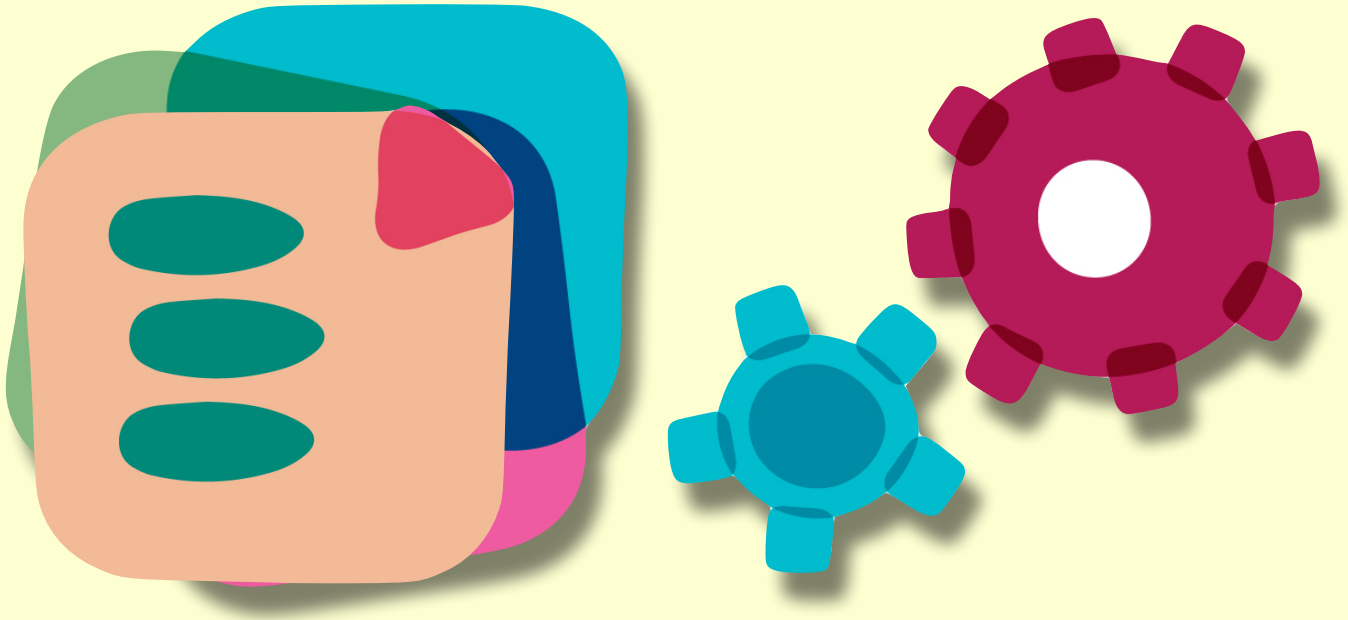
```java
@Test
public void third_party_class_should_only_be_instantiated_via_workaround() {
    classes().should(notCreateProblematicClassesOutsideOfWorkaroundFactory()
            .as(THIRD_PARTY_CLASS_RULE_TEXT))
            .check(classes);
}

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

Legality of Open Source Libraries

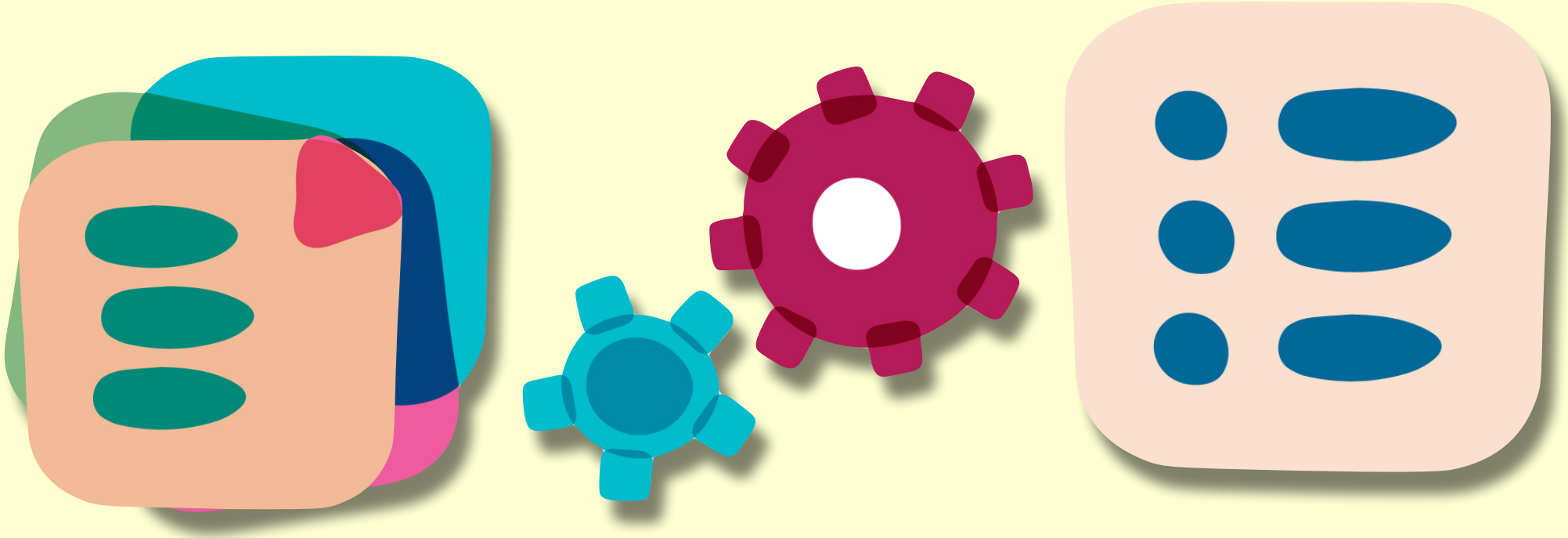
Penultimate

Legality of Open Source Libraries



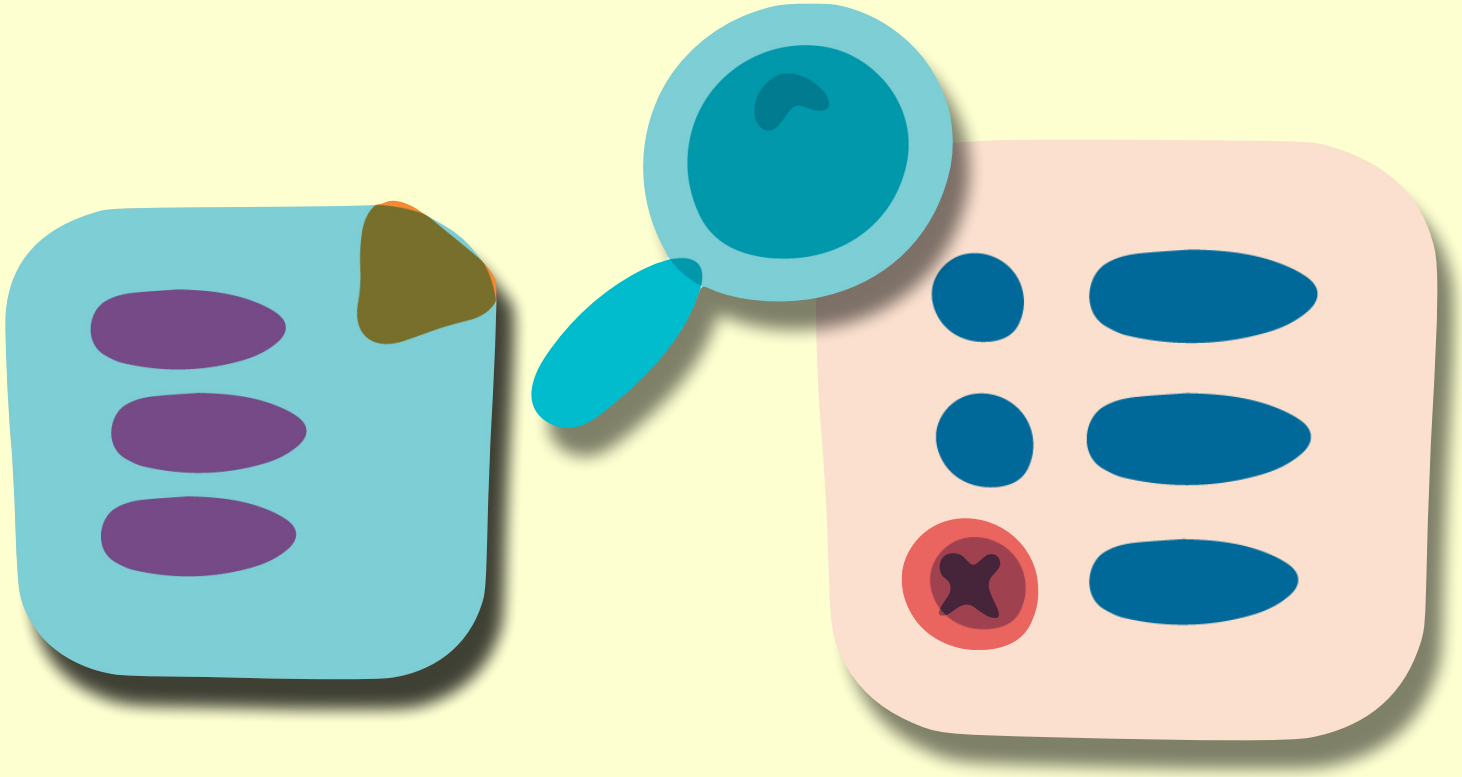
Penultimate

Legality of Open Source Libraries



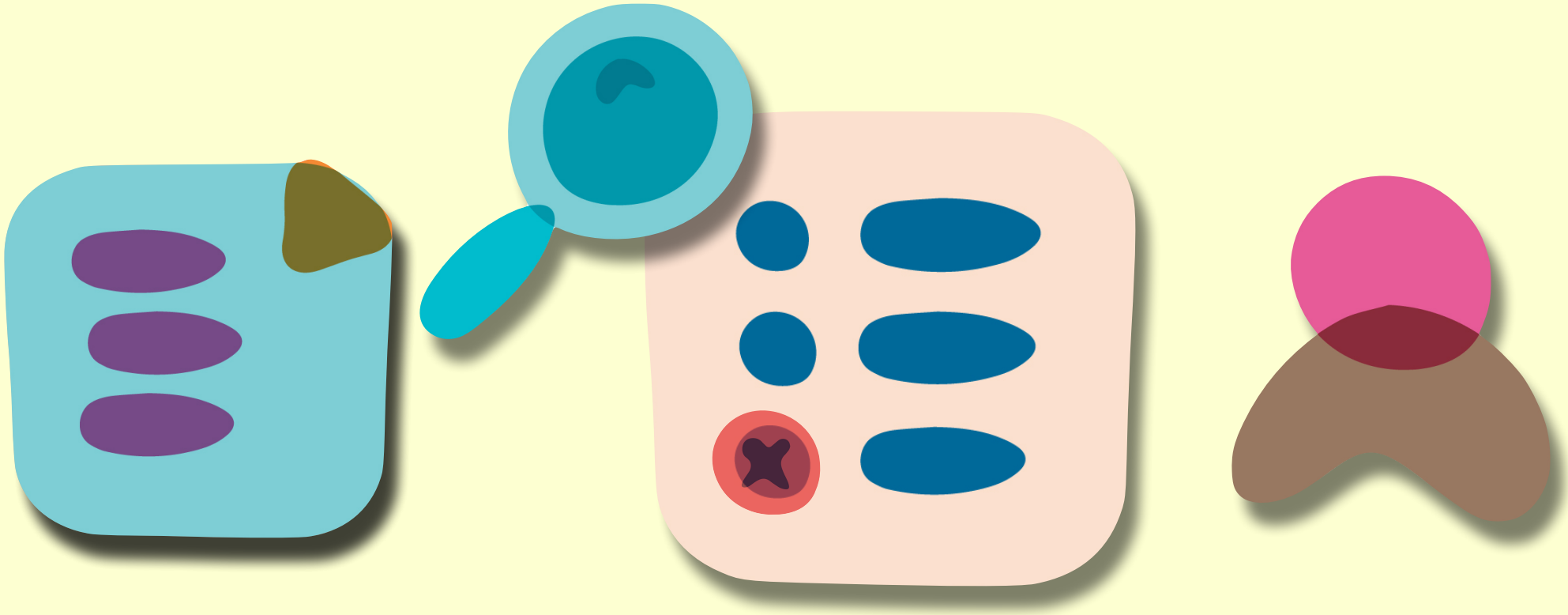
Penultimate

Legality of Open Source Libraries



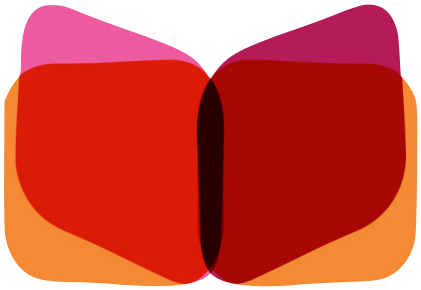
Penultimate

Legality of Open Source Libraries

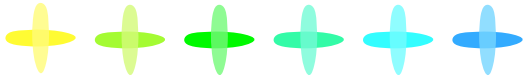


Penultimate

Agenda Part 1

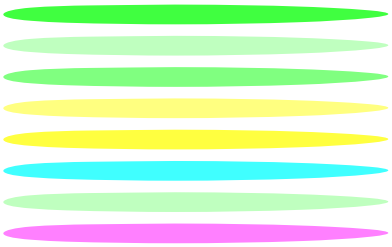


Definition



Incremental Change

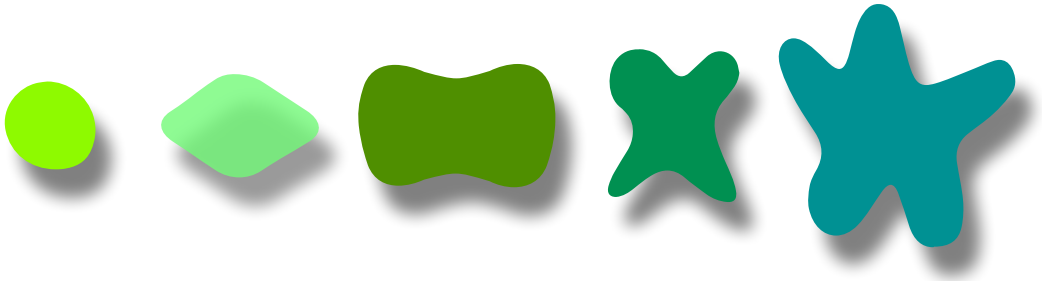
Guided Change via Fitness Functions



Structuring architecture for evolution

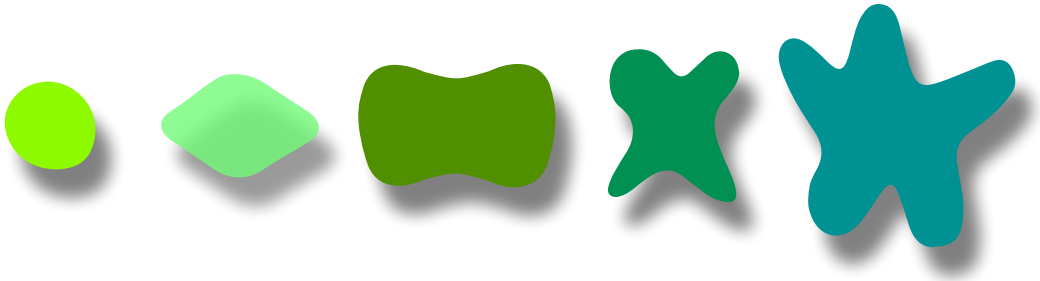
Two Big Ideas

Two Big Ideas

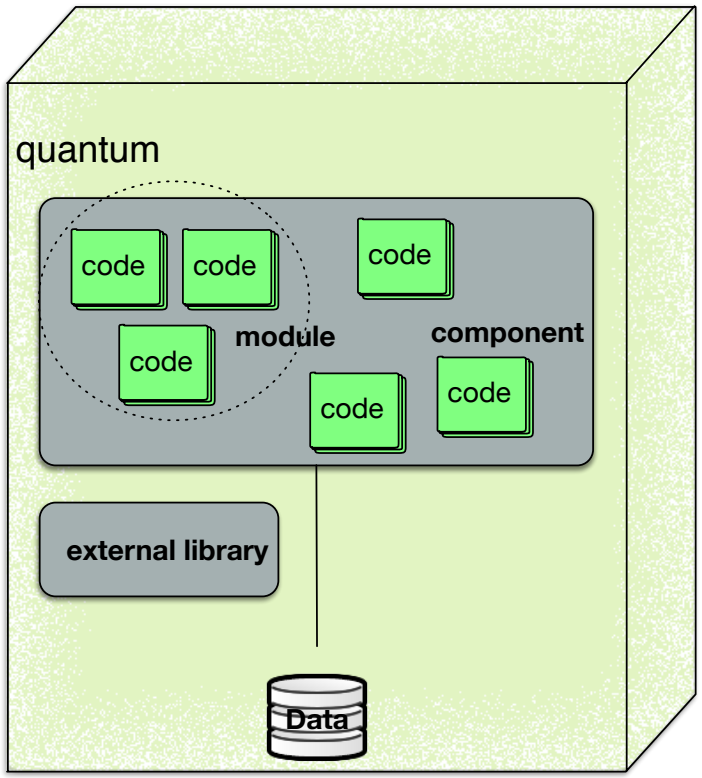


fitness functions for evolvability

Two Big Ideas

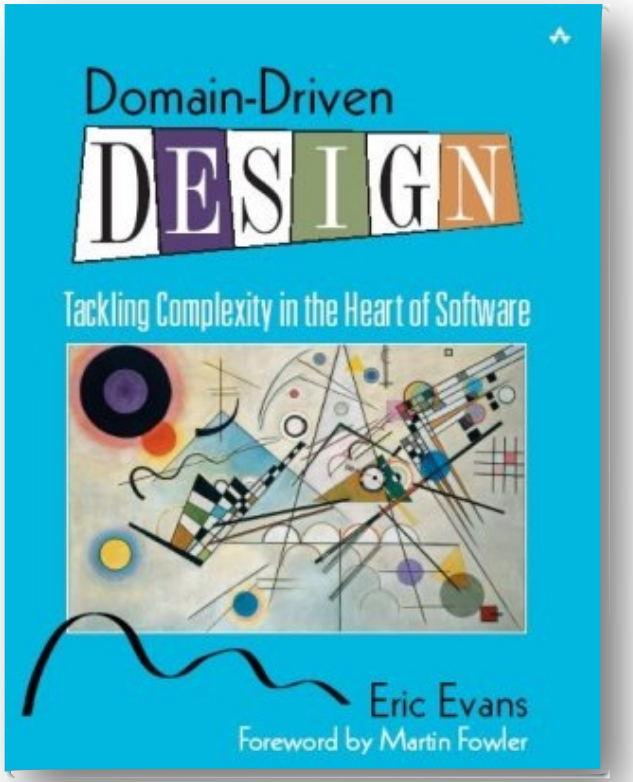


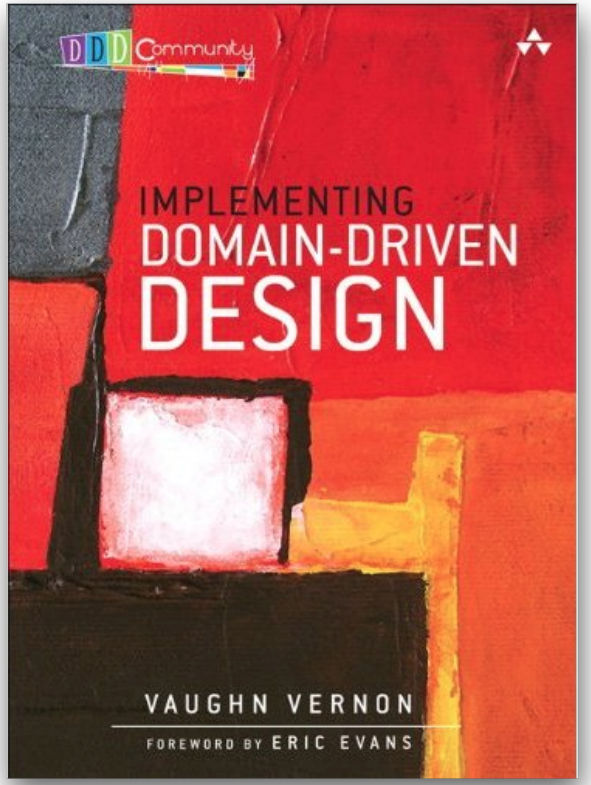
fitness functions for evolvability

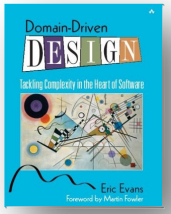


architectural quantum

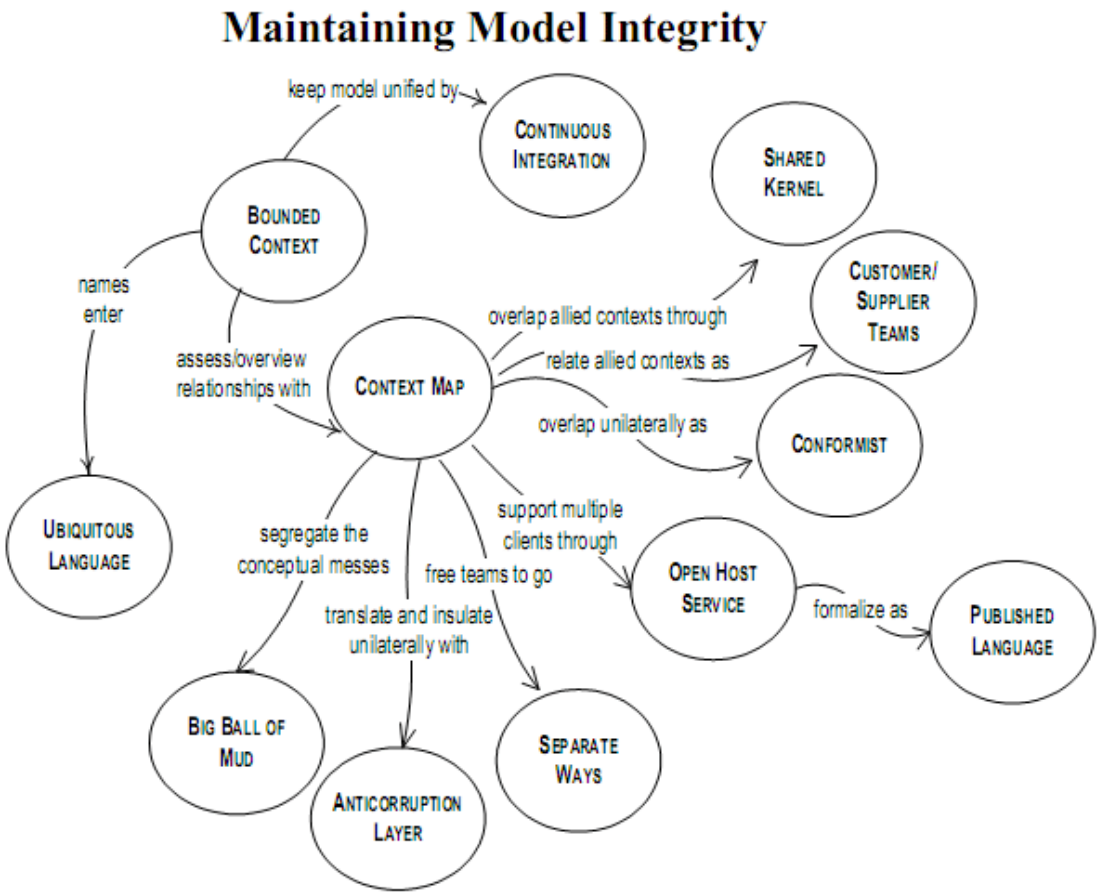
Domain Driven Design



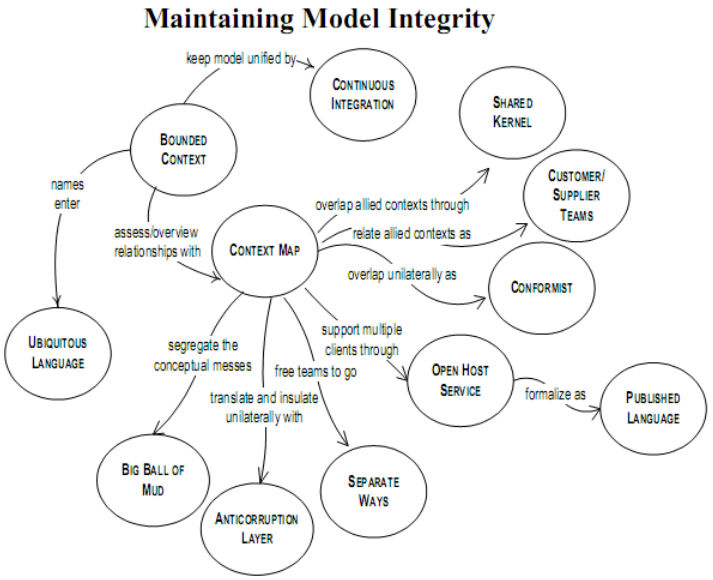




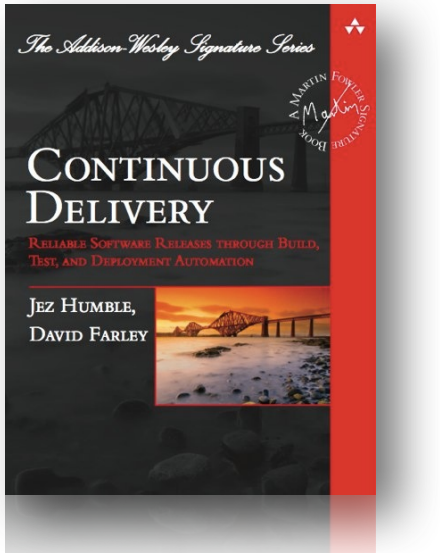
Bounded Context



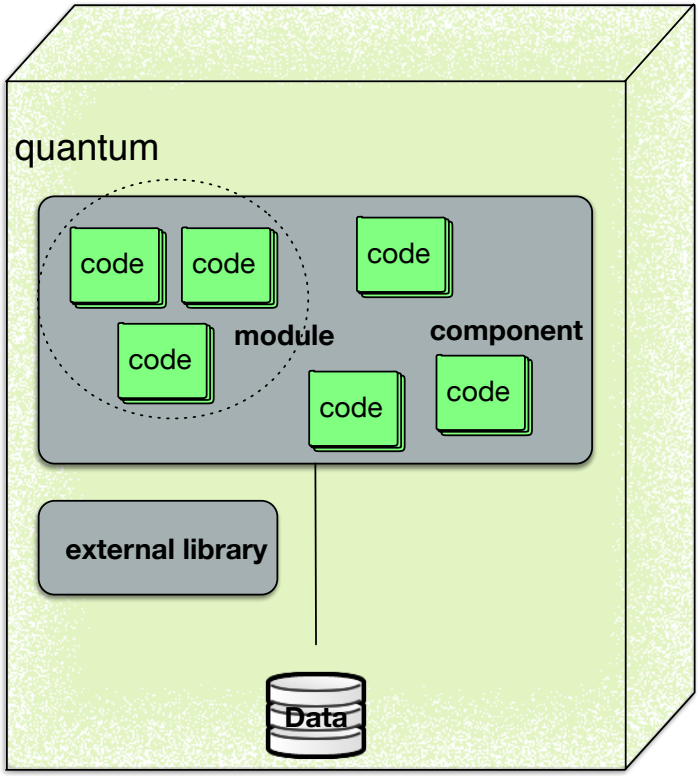
Bounded Context + Continuous Delivery = microservices



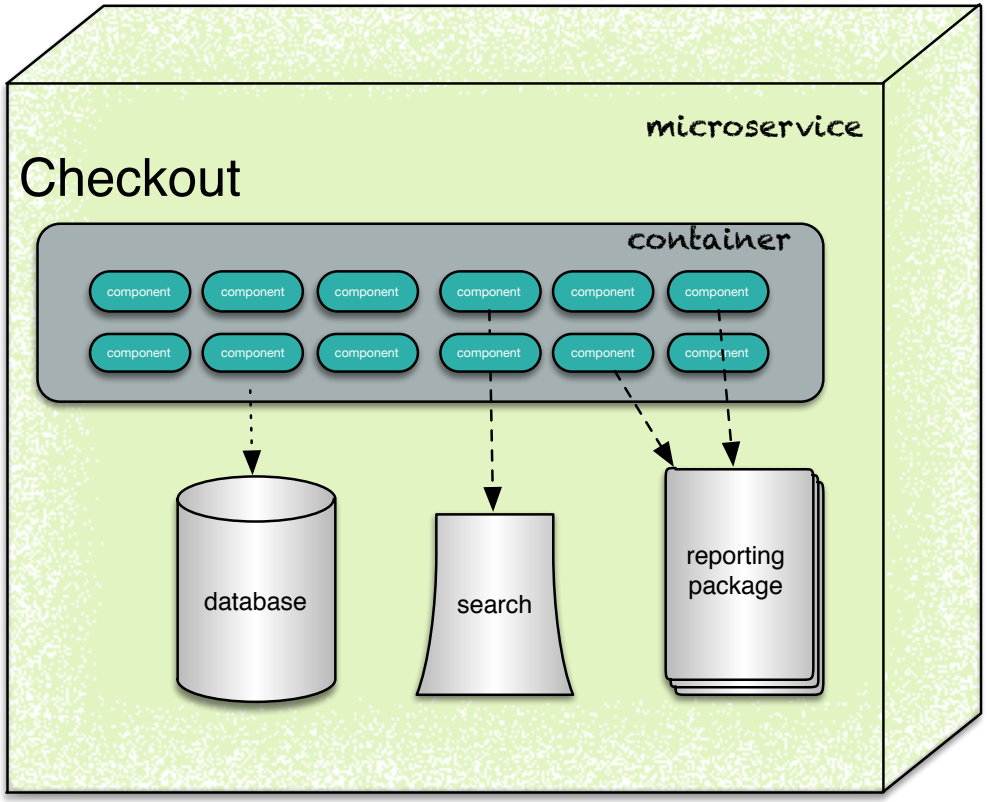
+



Architectural Quantum



Microservices Quantum

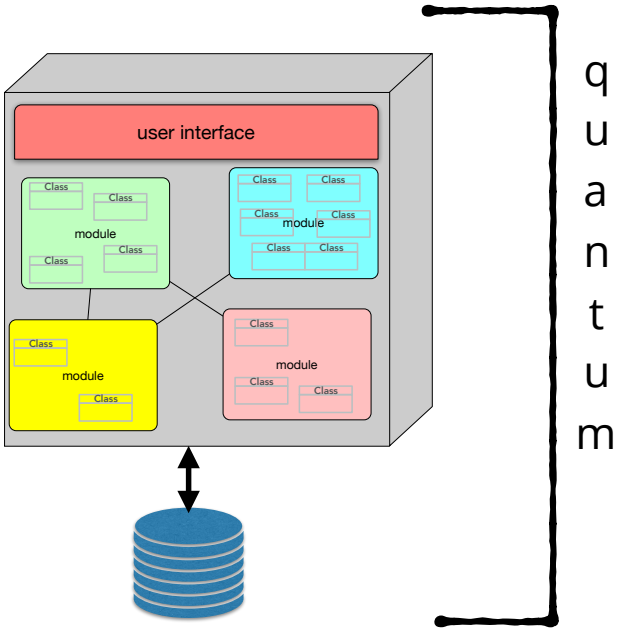


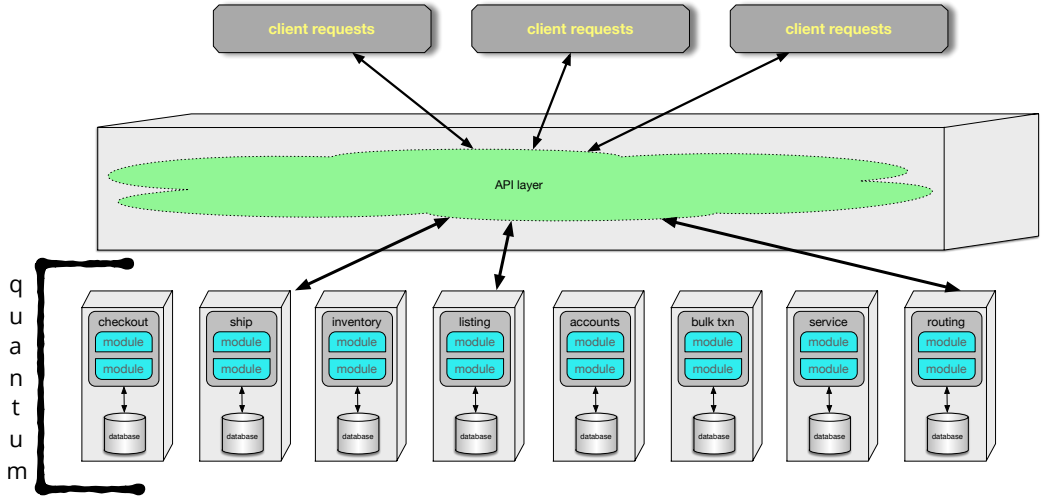
Architectural Quantum

An architectural quantum is an independently deployable component with high functional cohesion, which includes all the structural elements
required for the system to function properly.

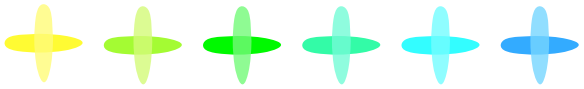
Why Quantum?

The quantum is where architectural characteristics live.



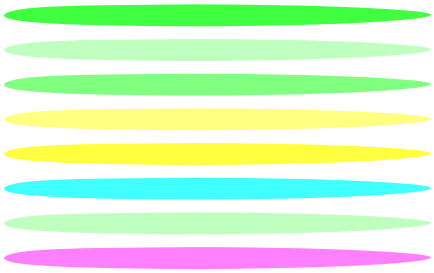


For Each Pattern:



incremental change

guided change via fitness functions

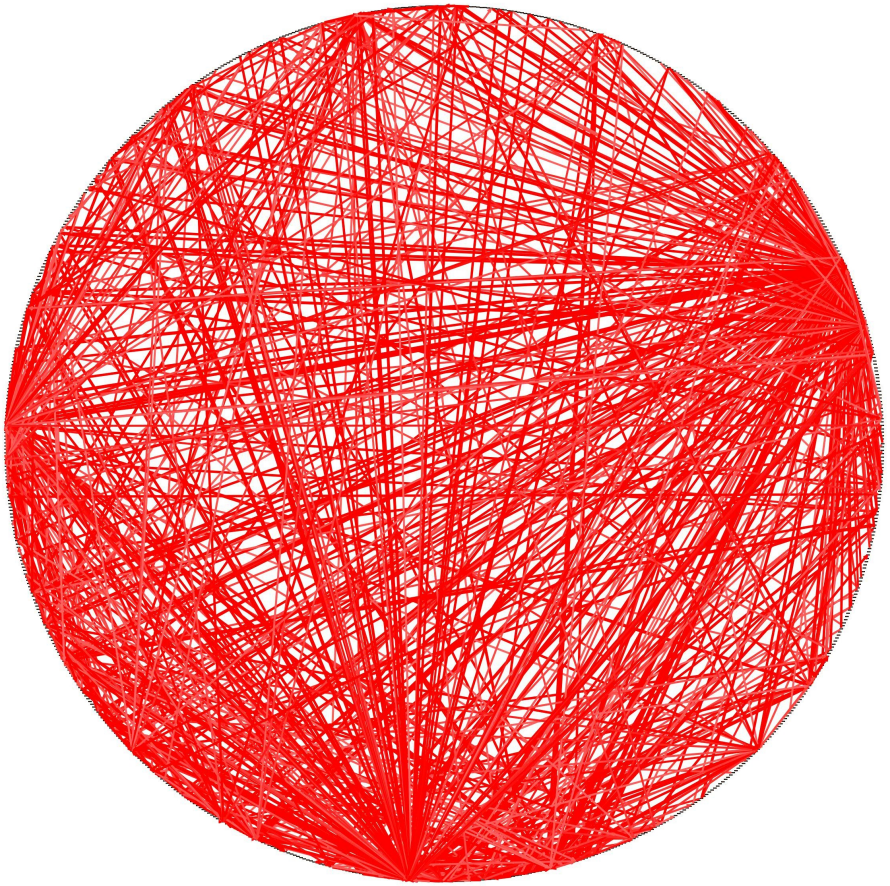


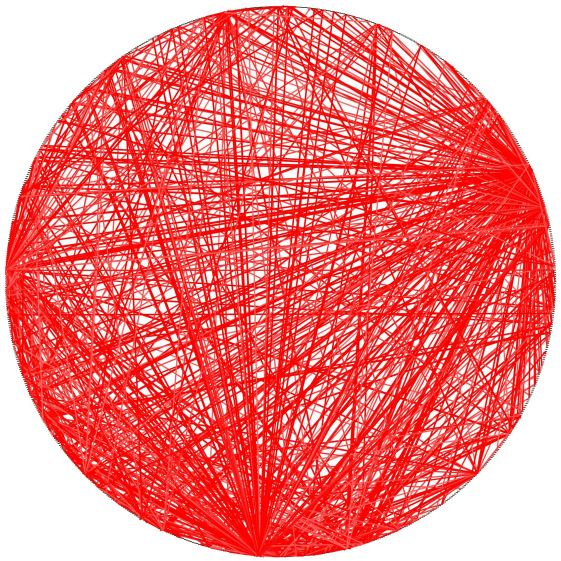
appropriate coupling

organized by:

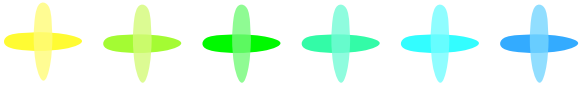
architectural quantum

Big Ball of Mud



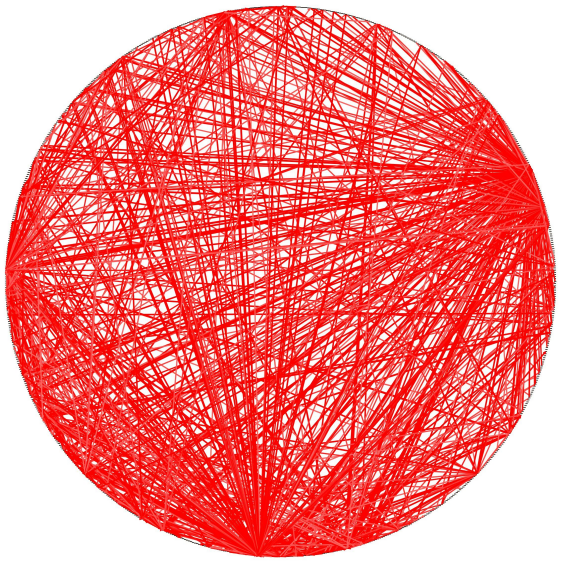


Big Ball of Mud

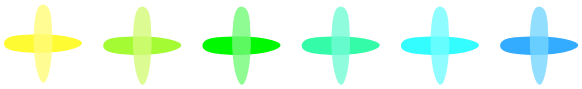


Rippling side effects for any change

the entire system



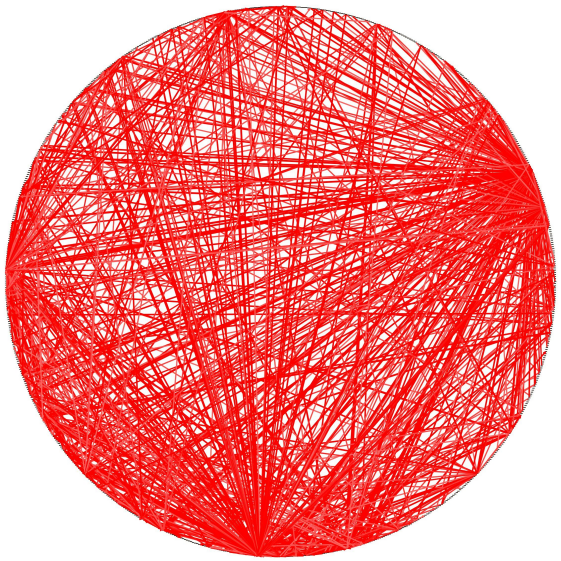
Big Ball of Mud



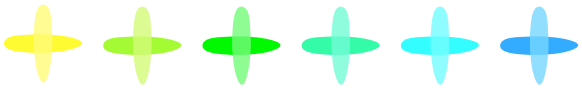
Rippling side effects for any change

difficult because no clearly defined partitioning exists

the entire system

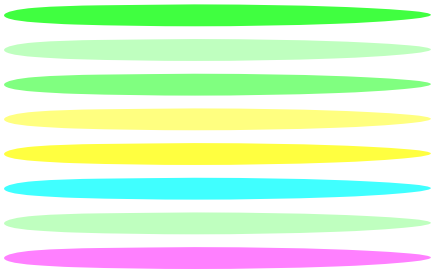


Big Ball of Mud



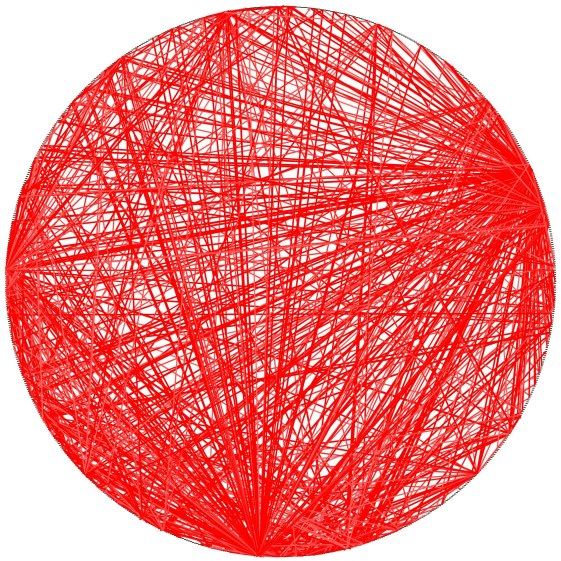
Rippling side effects for any change

difficult because no clearly defined partitioning exists

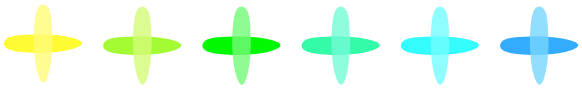


Epitome of **in**appropriate coupling

the entire system

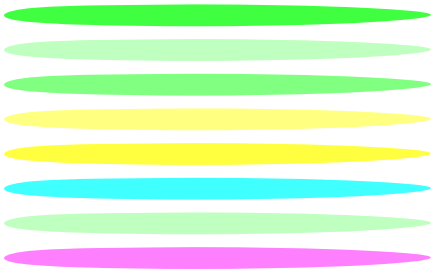


Big Ball of Mud



Rippling side effects for any change

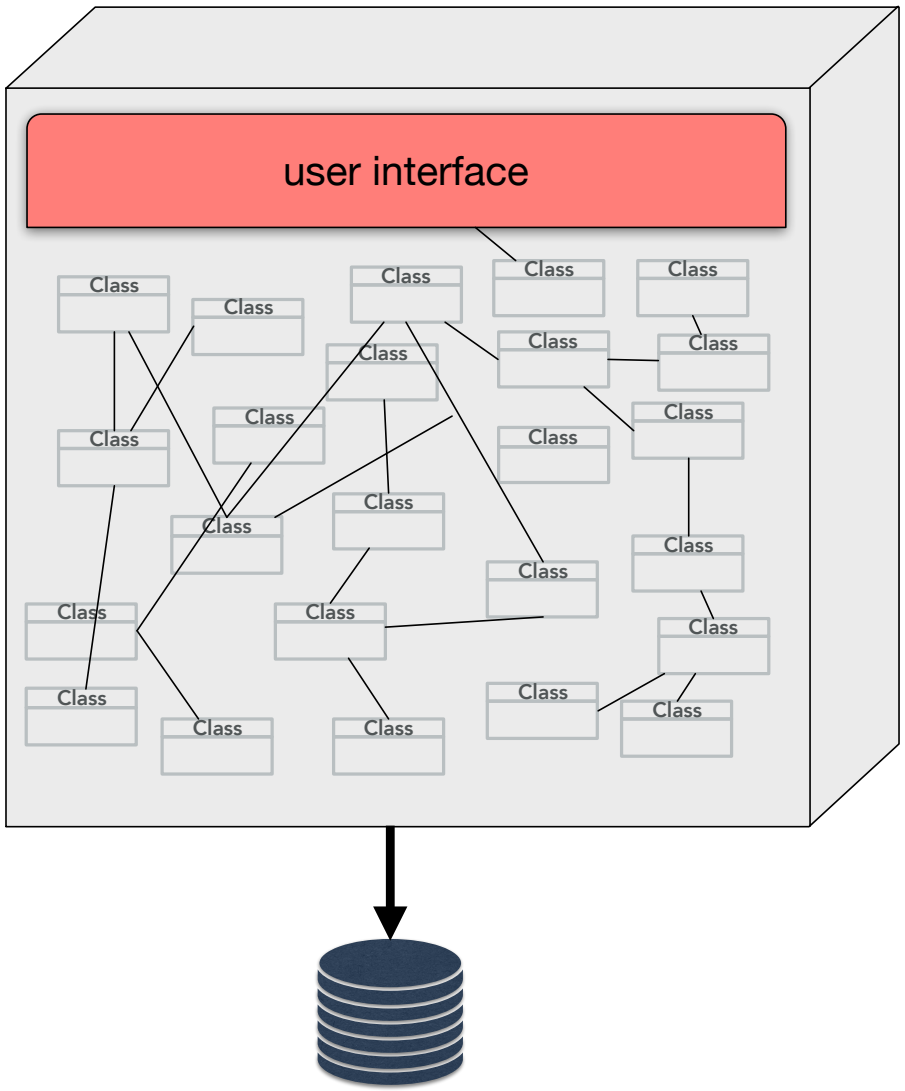
difficult because no clearly defined partitioning exists



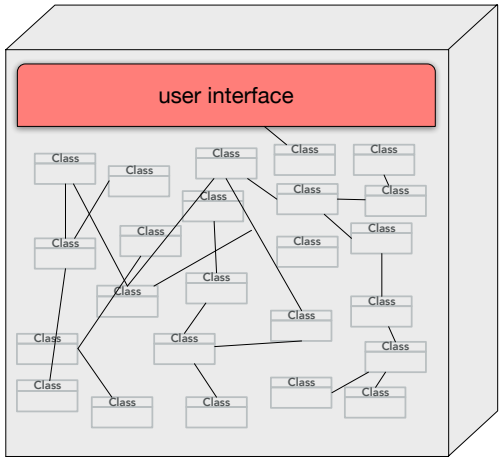
Epitome of **in**appropriate coupling

the entire system

Monoliths



Unstructured Monoliths

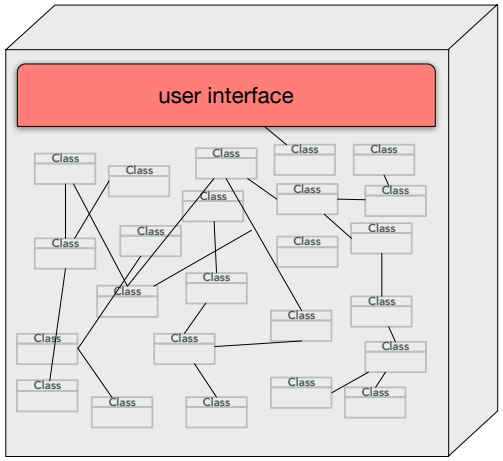


Unstructured Monoliths

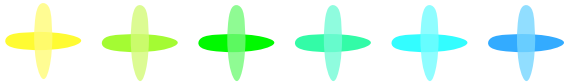
Large quantum size

hinders incremental change

the entire system



Unstructured Monoliths

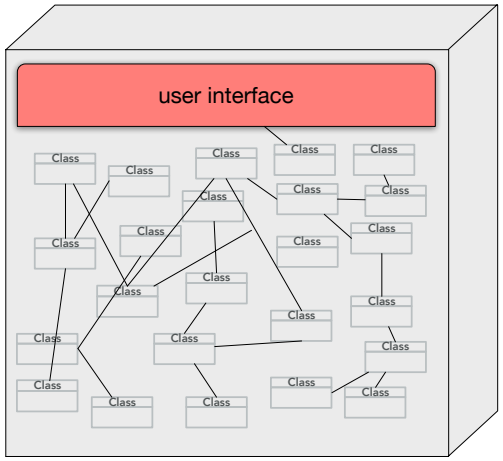


Large quantum size

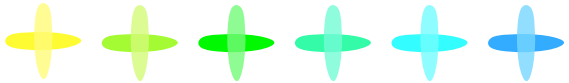
hinders incremental change

difficult but not impossible

the entire system



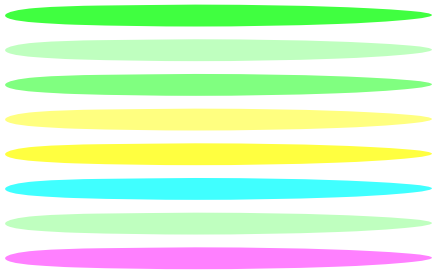
Unstructured Monoliths



Large quantum size

hinders incremental change

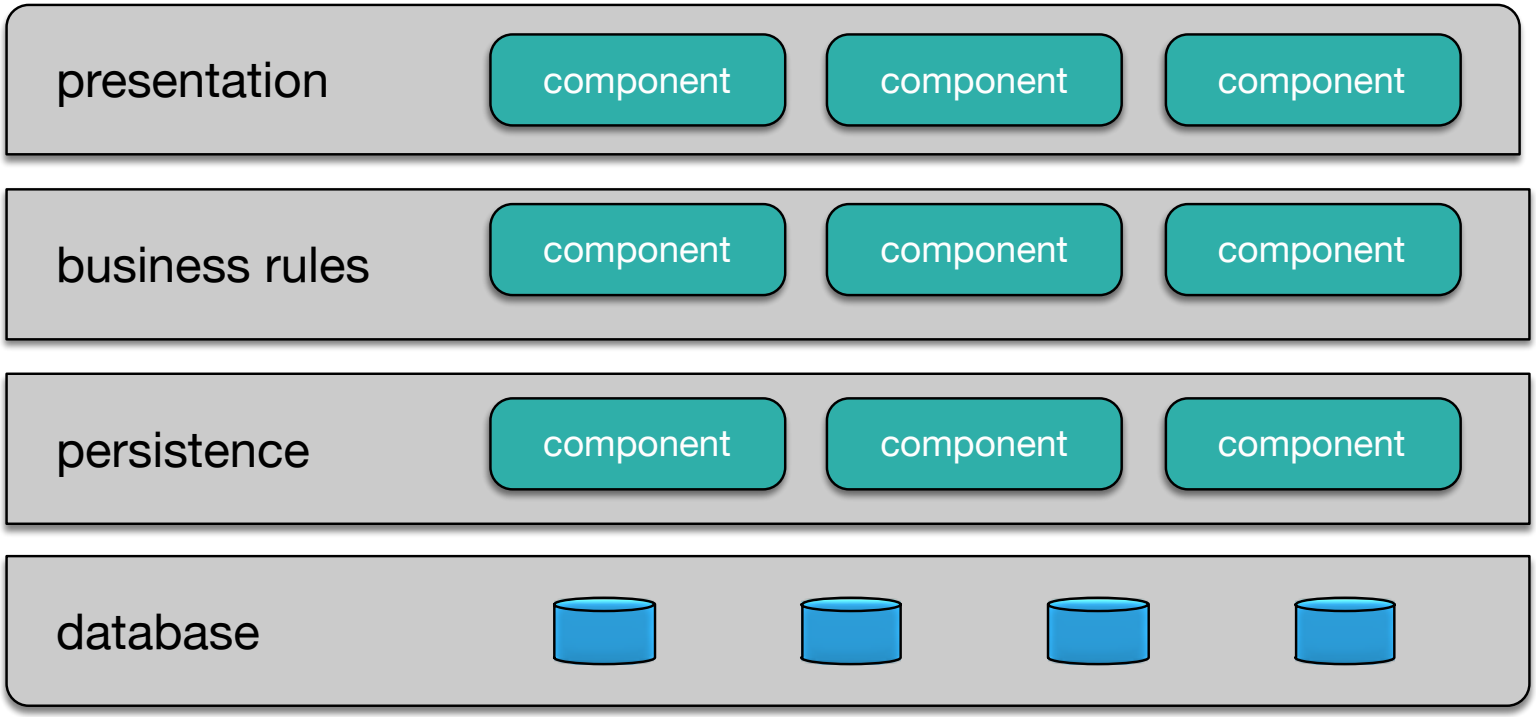
difficult but not impossible

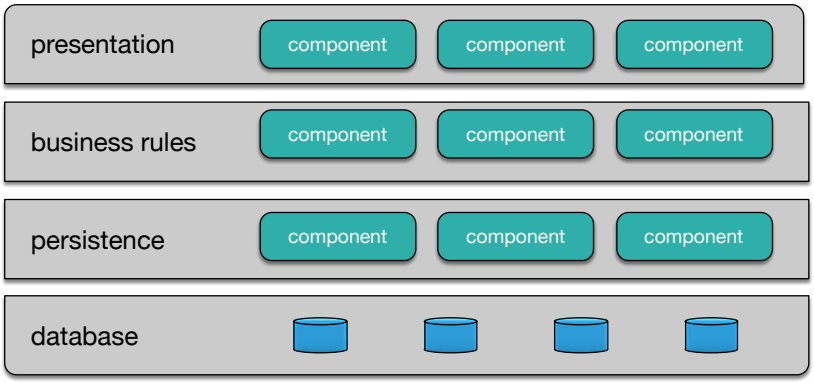


functionally almost as bad as the Big Ball of Mud

the entire system

Layered Monolith

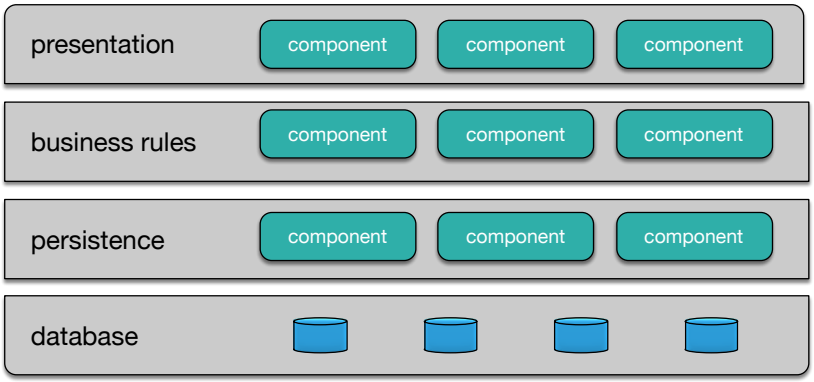




Layered Monolith

Selectively easy based on partitioning

the entire system

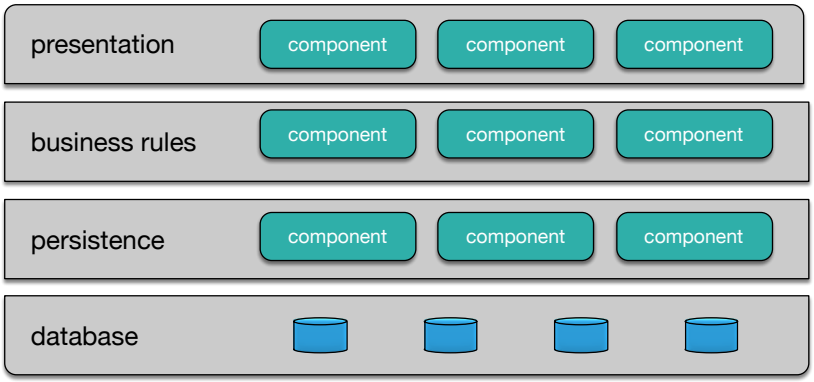


Layered Monolith

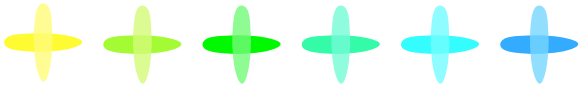
Selectively easy based on partitioning

easier because structure is more apparent

the entire system

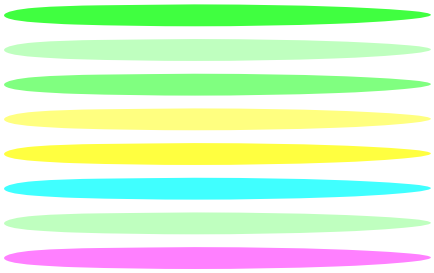


Layered Monolith



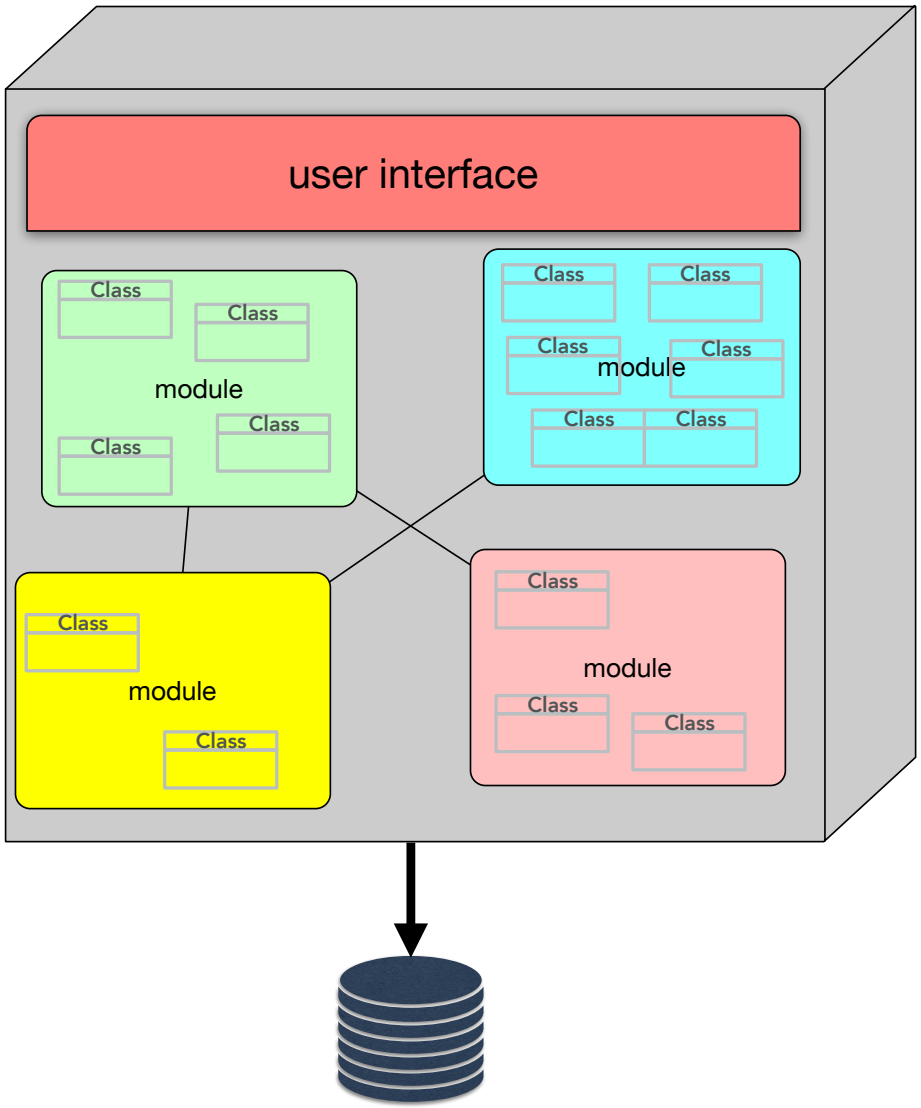
Selectively easy based on partitioning

easier because structure is more apparent

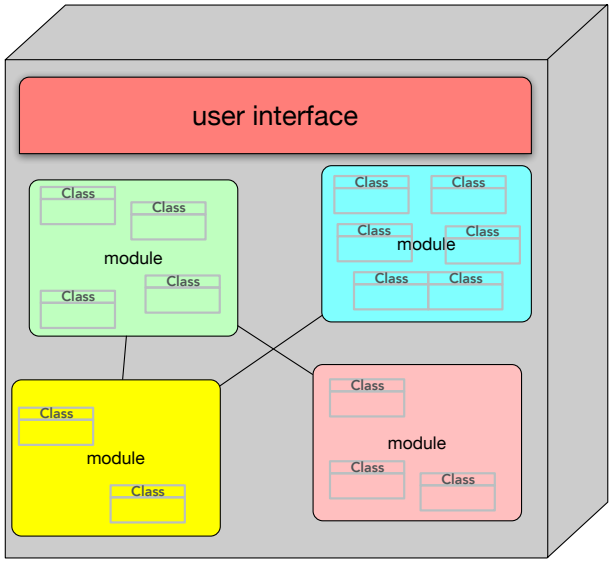


— good technical architecture partitioning
— more difficult for orthogonal concerns

the entire system

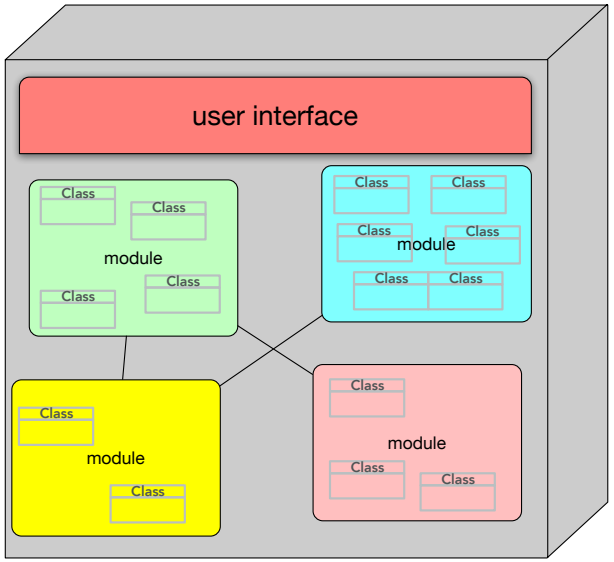


Modular Monoliths



Modular Monoliths

the entire system

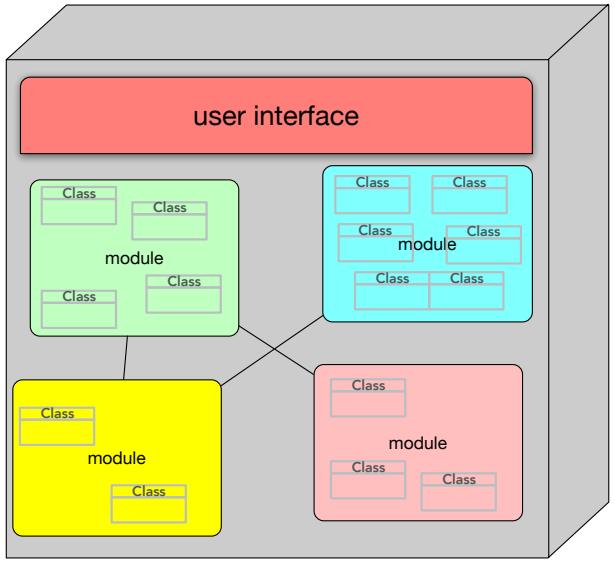


Modular Monoliths

the entire system, with selective better granularity

the degree of deployability of the components determines the rate of incremental change.

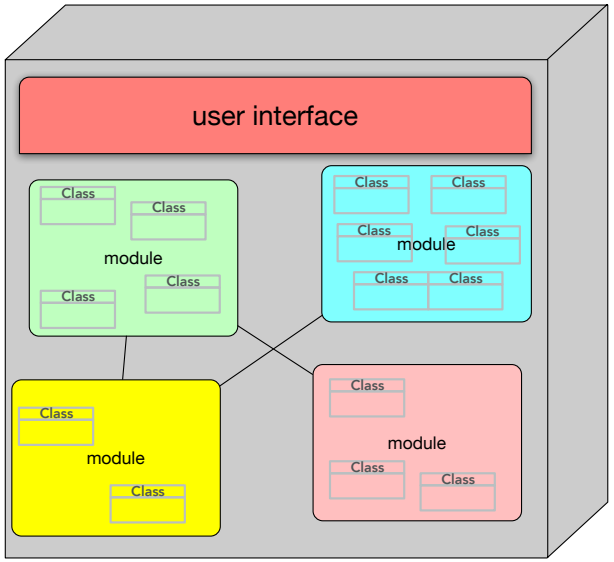
Modular Monoliths



the entire system, with selective better granularity

the degree of deployability of the components determines the rate of incremental change.

easier to design and implement in this architecture because of good separation of components



Modular Monoliths

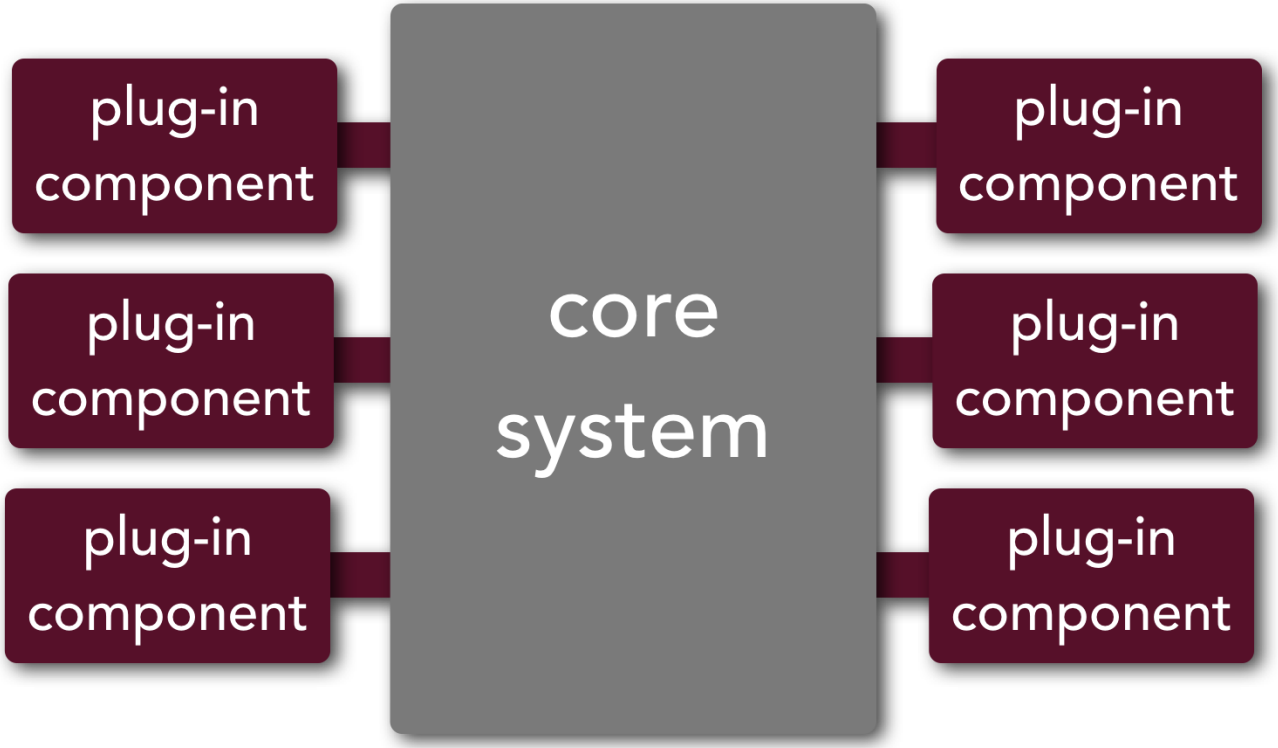
the degree of deployability of the components determines the rate of incremental change.

easier to design and implement in this architecture (good separation of components)

Each component is functionally cohesive, with good interfaces between them and low coupling.

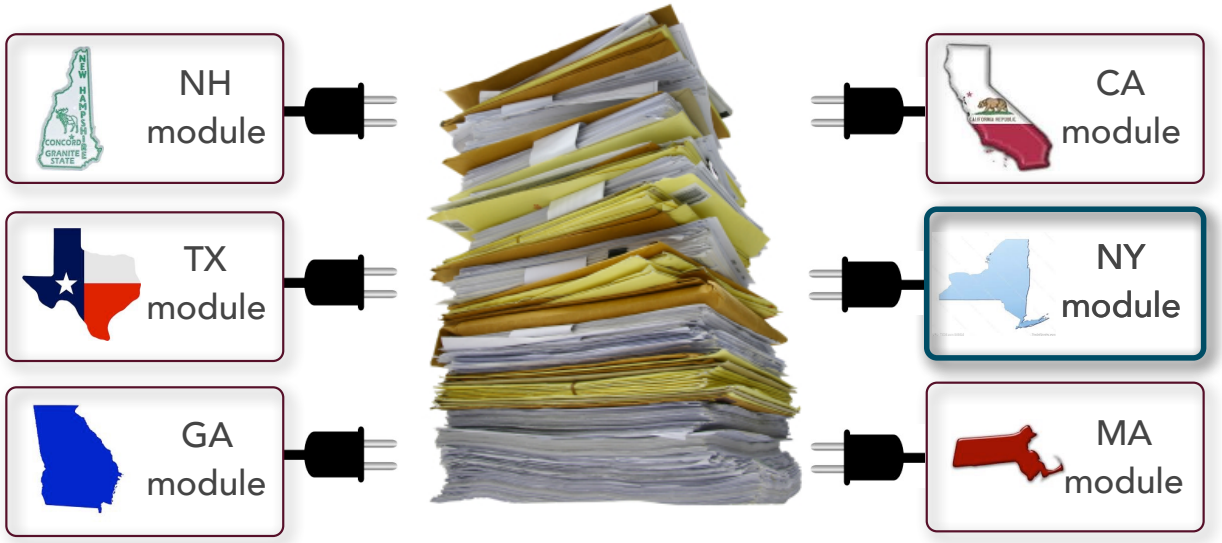
the entire system

Microkernel

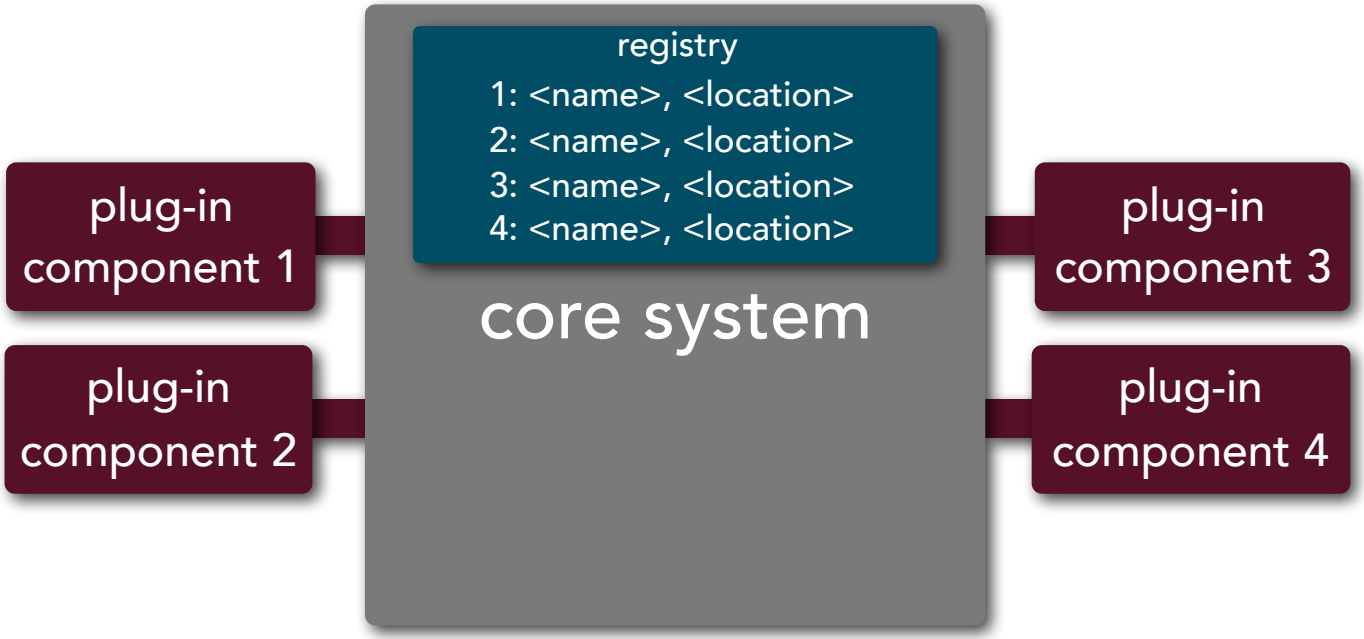


Microkernel

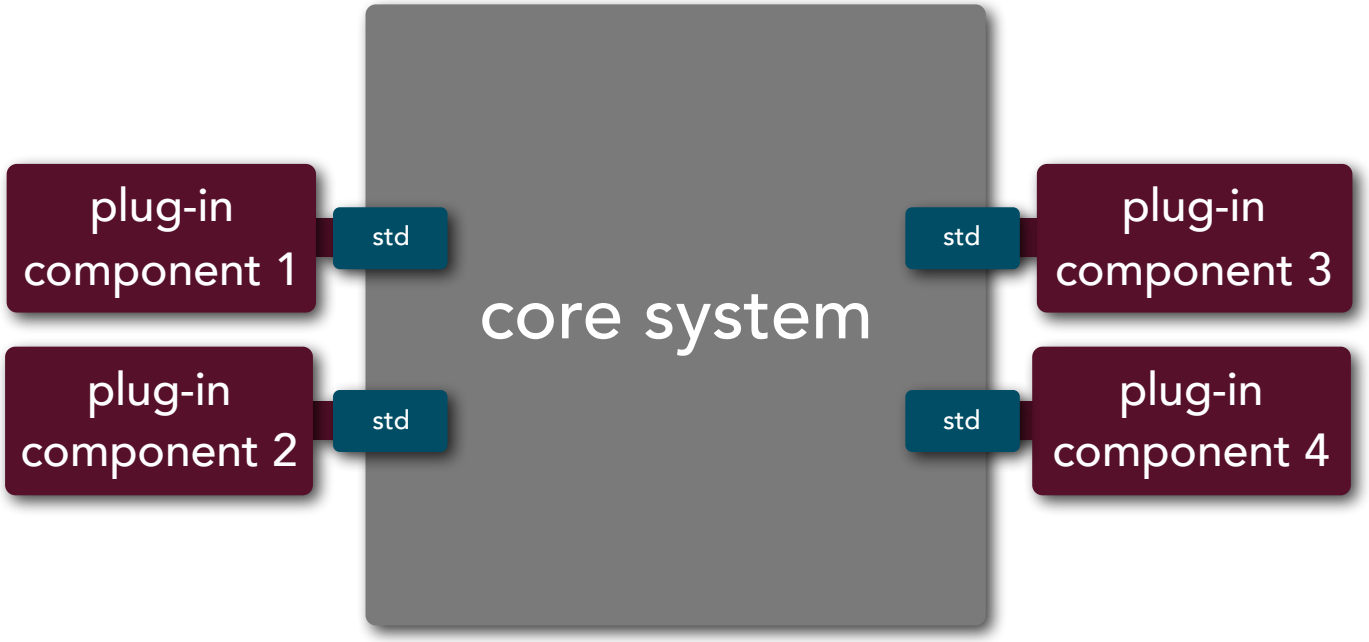
claims processing



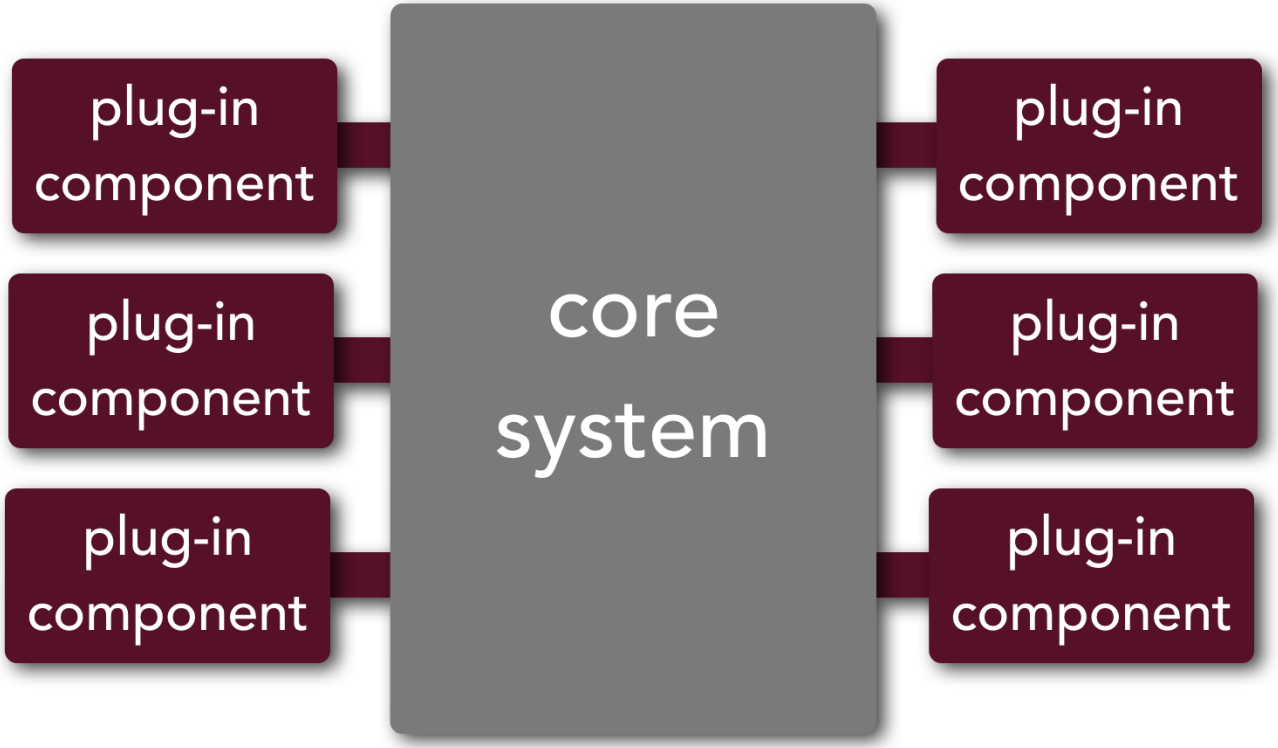
Microkernel Registry

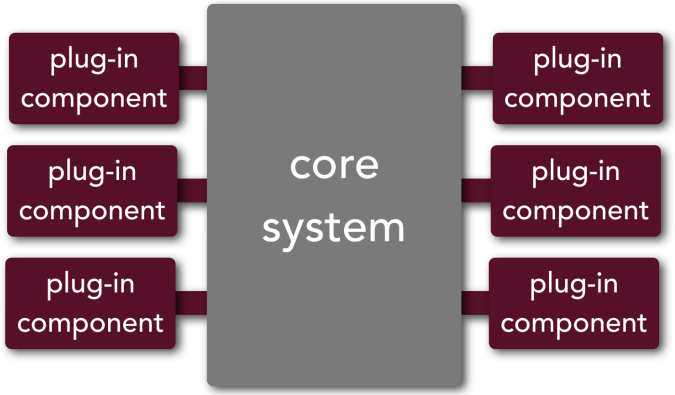


Contracts

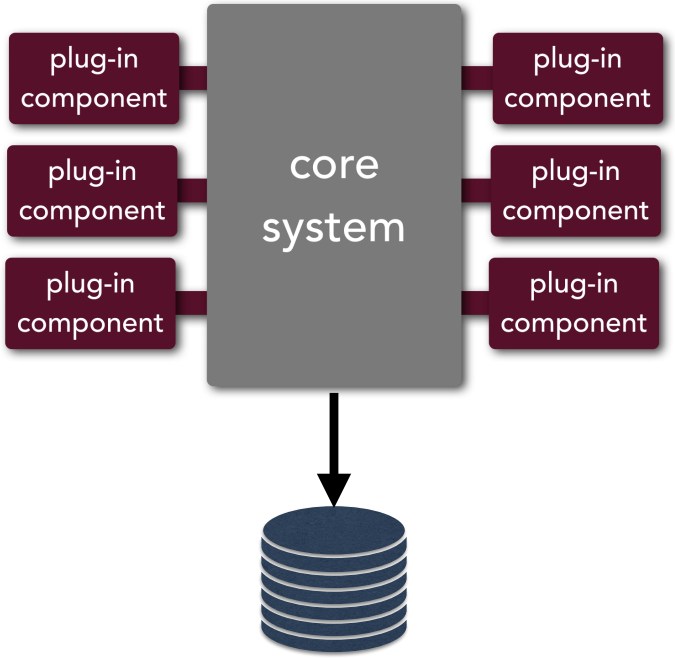

plug-in component 1
std
plug-in component 2
std
core system
std
plug-in component 3
std
plug-in component 4

Microkernel

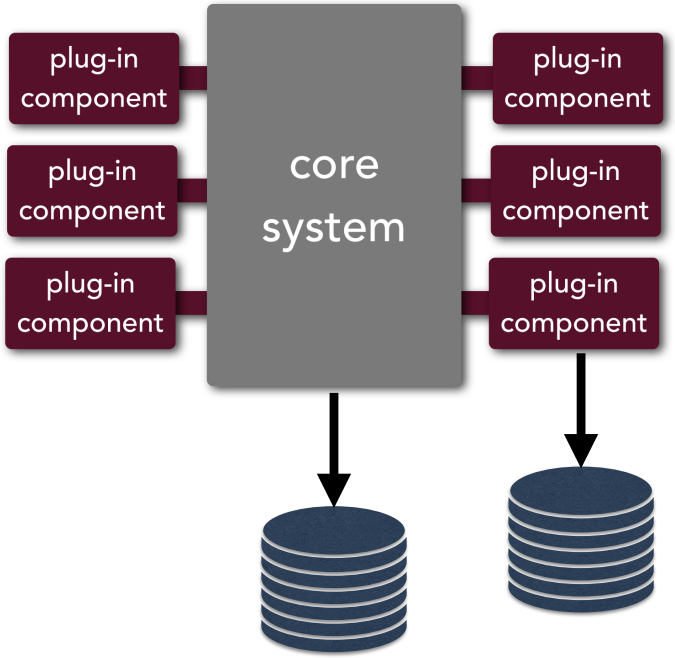




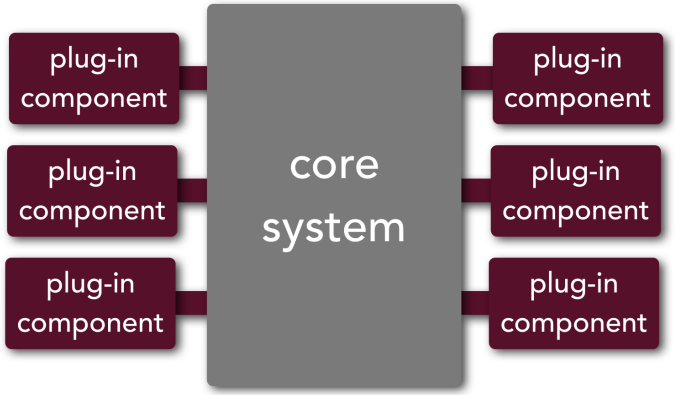
Microkernel



Microkernel

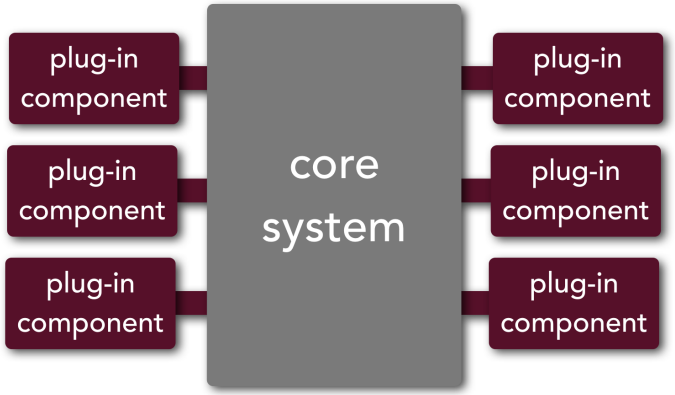


Microkernel

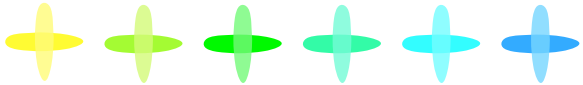


Microkernel

most changes via plug-ins

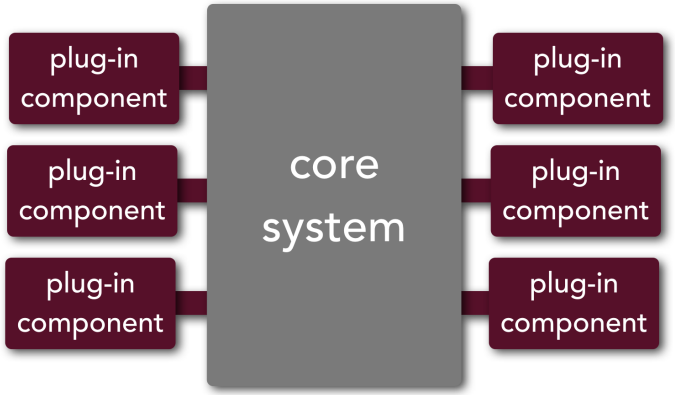


Microkernel

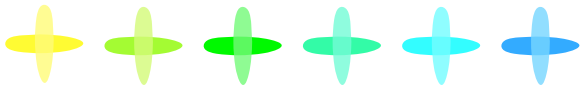


most changes via plug-ins

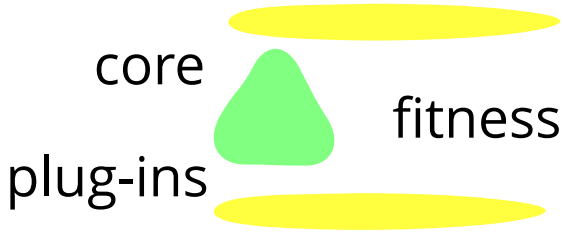
fitness functions for integration tests



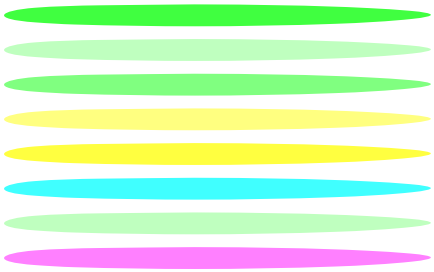
Microkernel



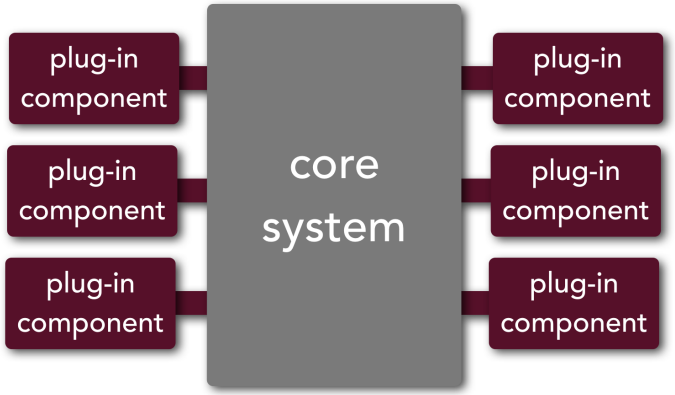
most changes via plug-ins



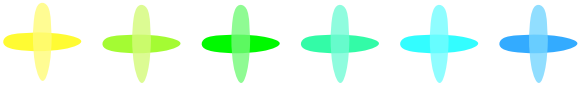
fitness functions for integration tests



coupling well defined by architectural pattern

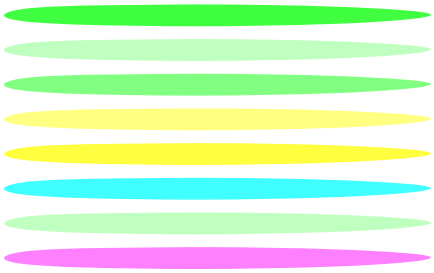


Microkernel



most changes via plug-ins

fitness functions for integration tests

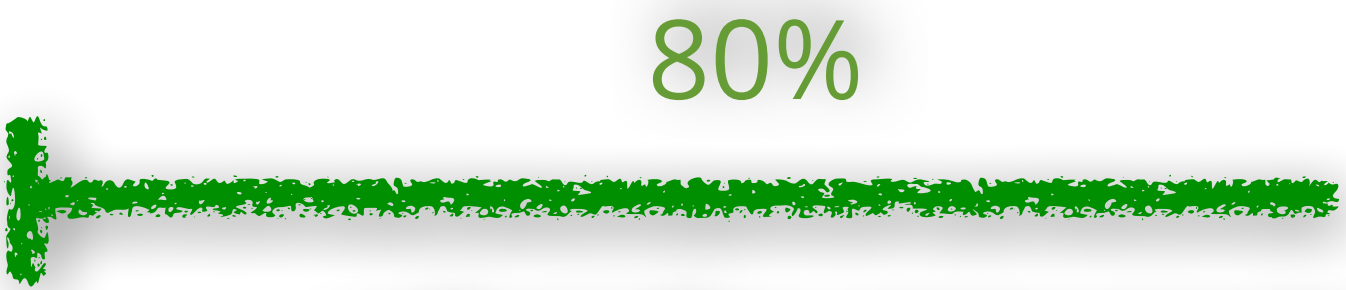


coupling well defined by architectural pattern

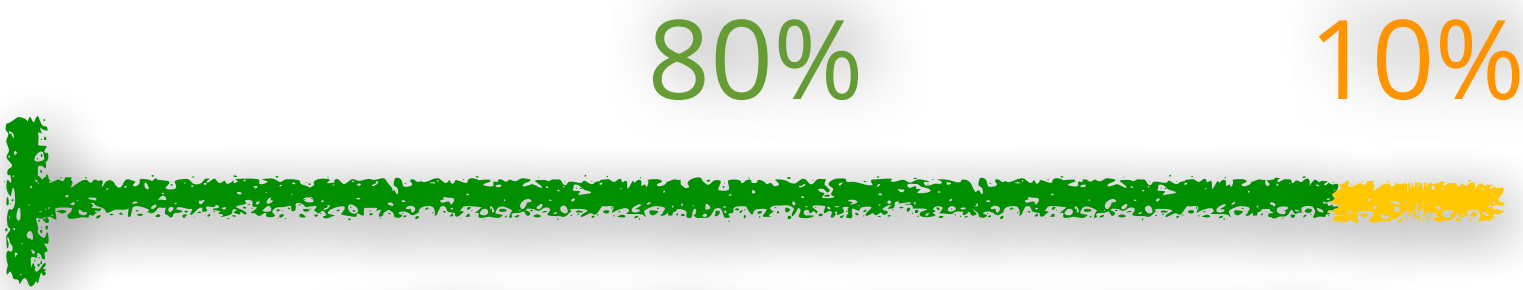
beware the Last 10% Trap

Last 10% Trap

Last 10% Trap



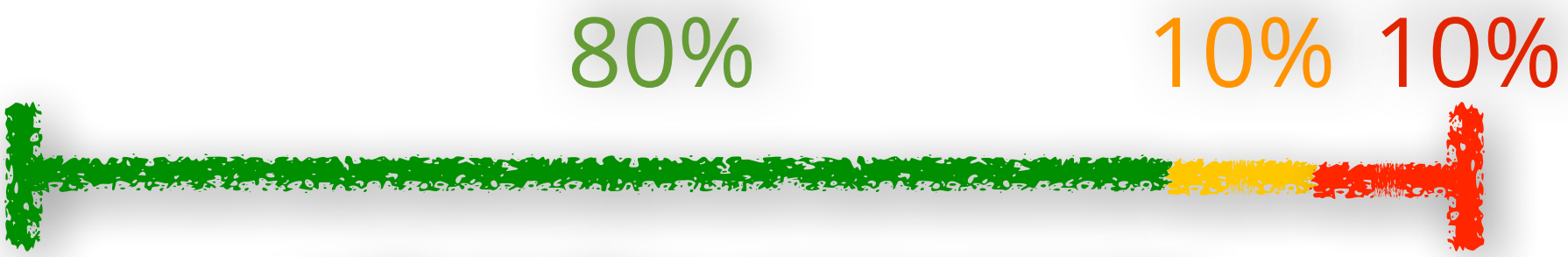
Last 10% Trap



what the user wants

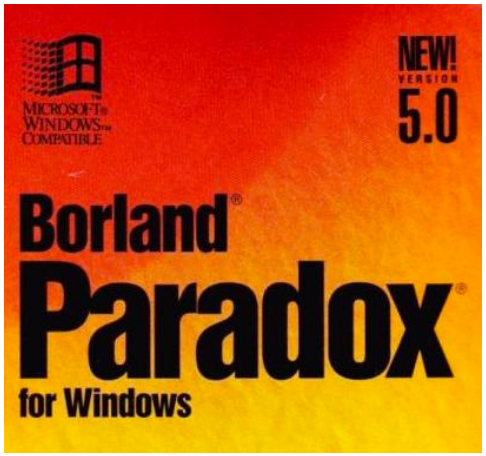
Last 10% Trap

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



what the user wants

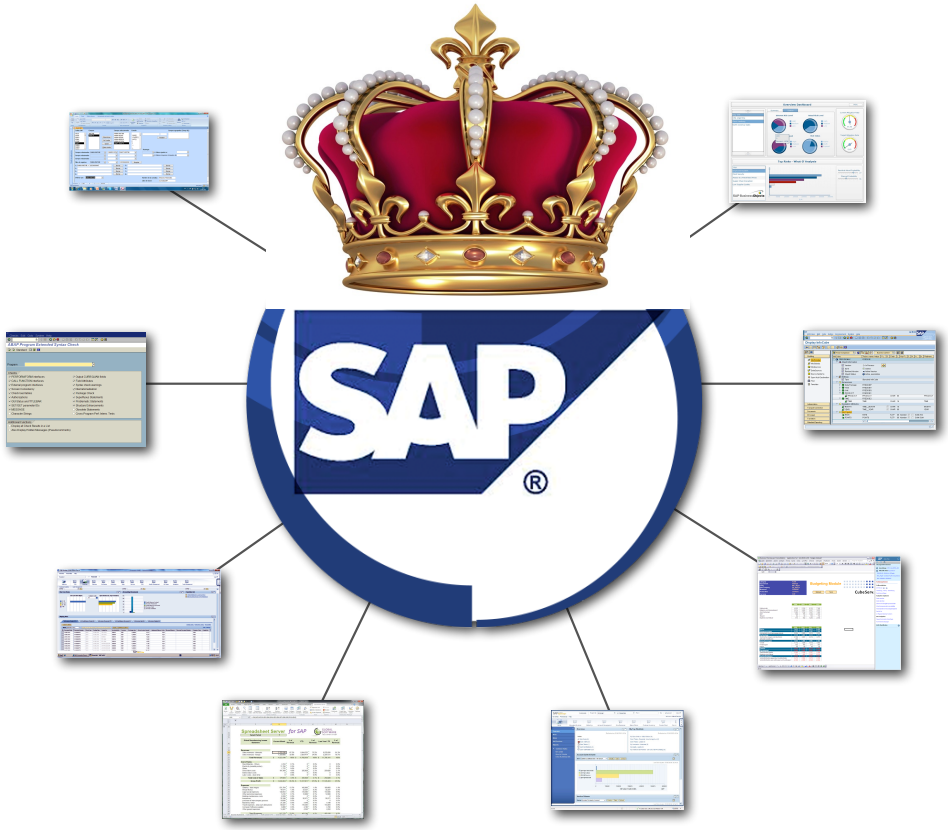
What Happened to the 4GLs?



What Happened to the 4GLs?

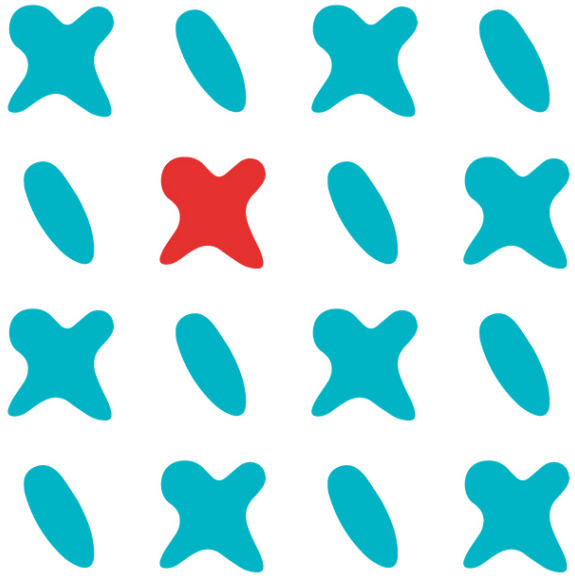
DSL

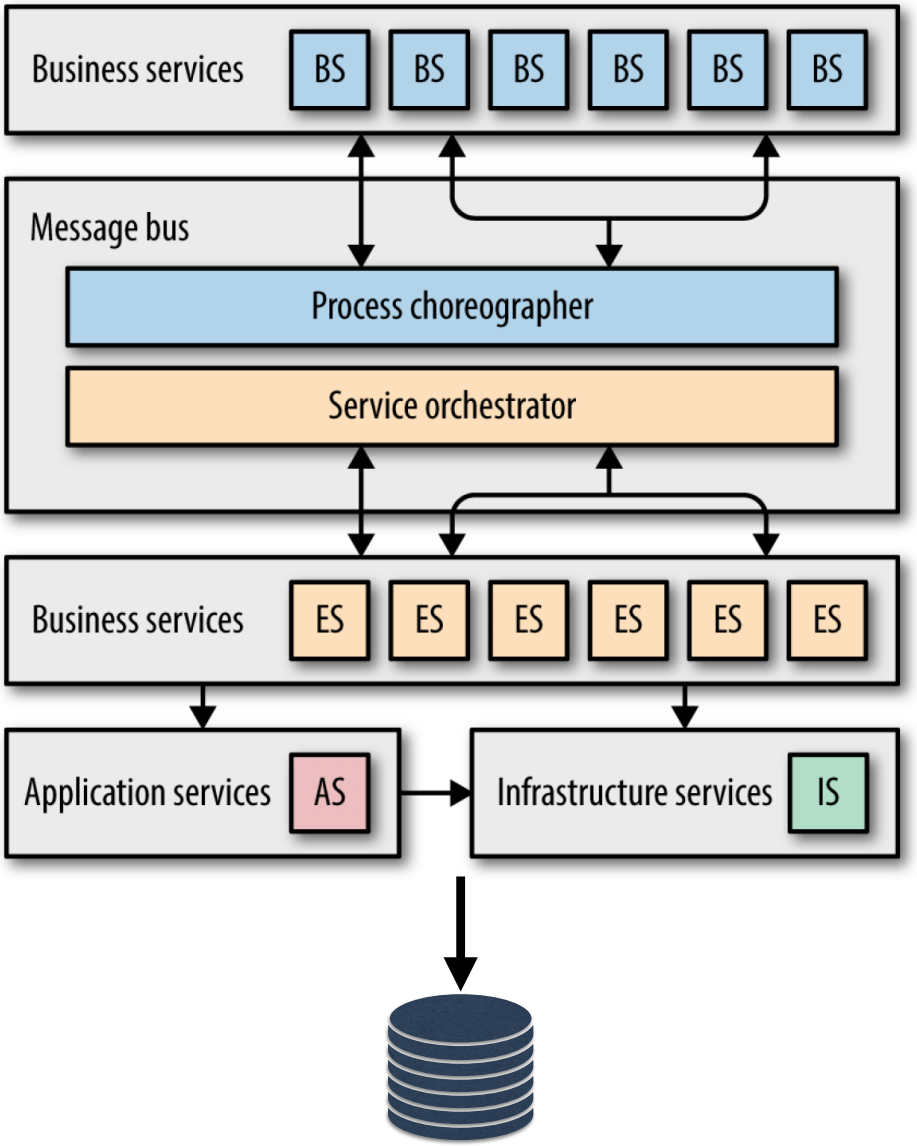
see also: Vendor King



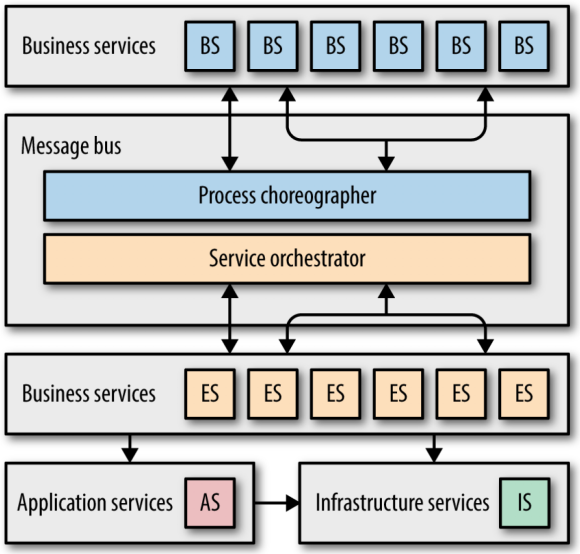
Be careful of the *Last 10% Trap* when choosing tools & frameworks.

Service-oriented Architectures



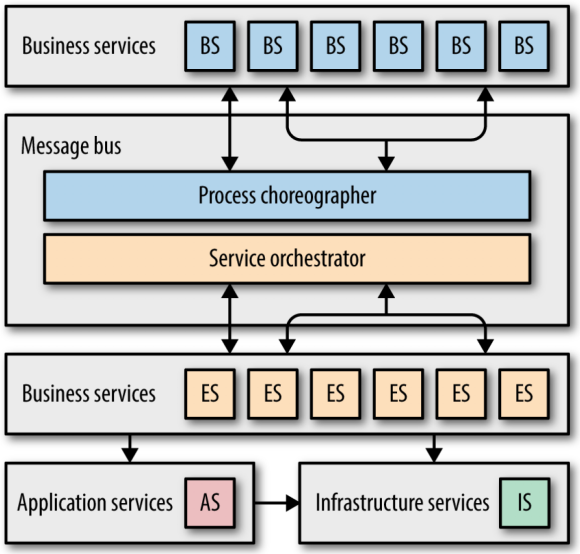


ESB-driven SOA



ESB-driven SOA

the entire system

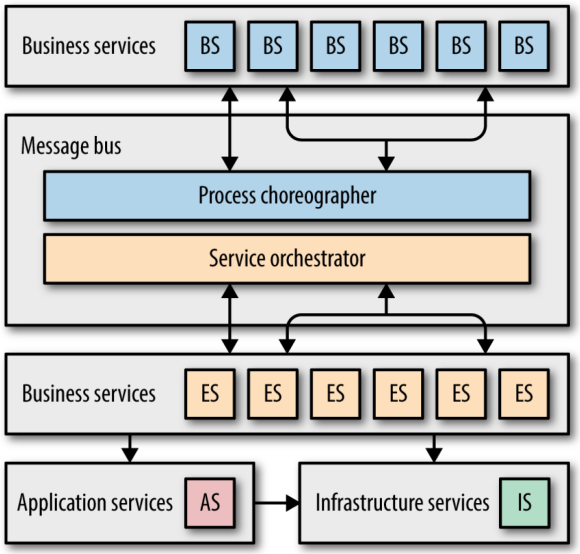


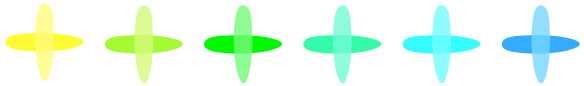
ESB-driven SOA

virtually impossible

the entire system

ESB-driven SOA



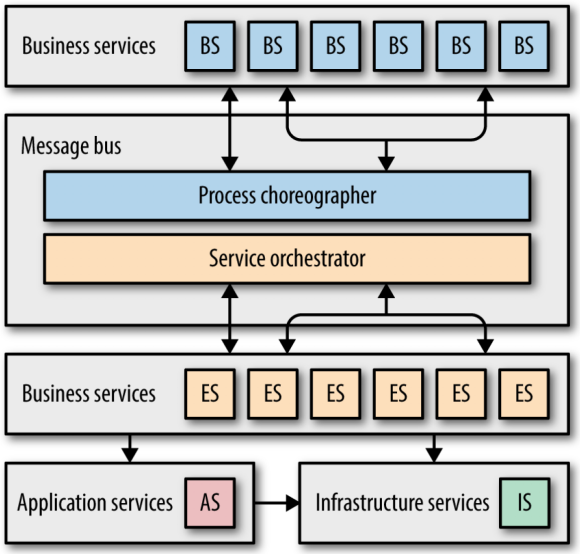


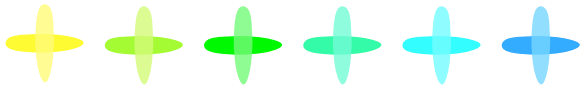
virtually impossible

the entire system

extremely difficult

ESB-driven SOA

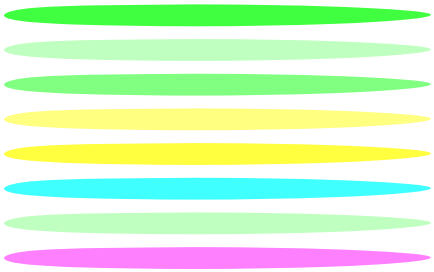




virtually impossible

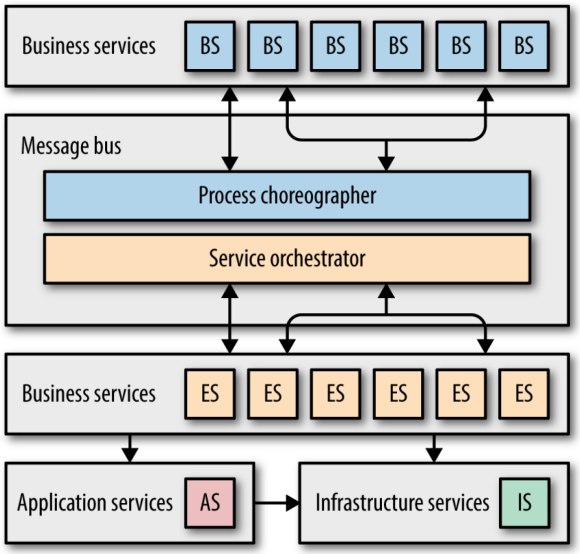
the entire system

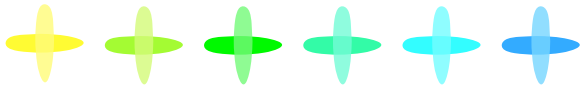
extremely difficult



extravagant taxonomy makes sense if reuse is primary goal

ESB-driven SOA

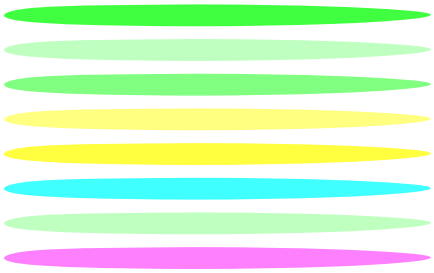




virtually impossible

the entire system

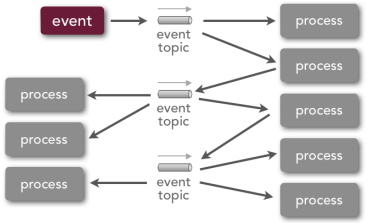
extremely difficult



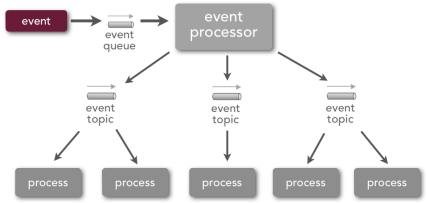
extravagant taxonomy makes sense if reuse is primary goal

extremely coupled in inappropriate ways.

Event-driven Architectures

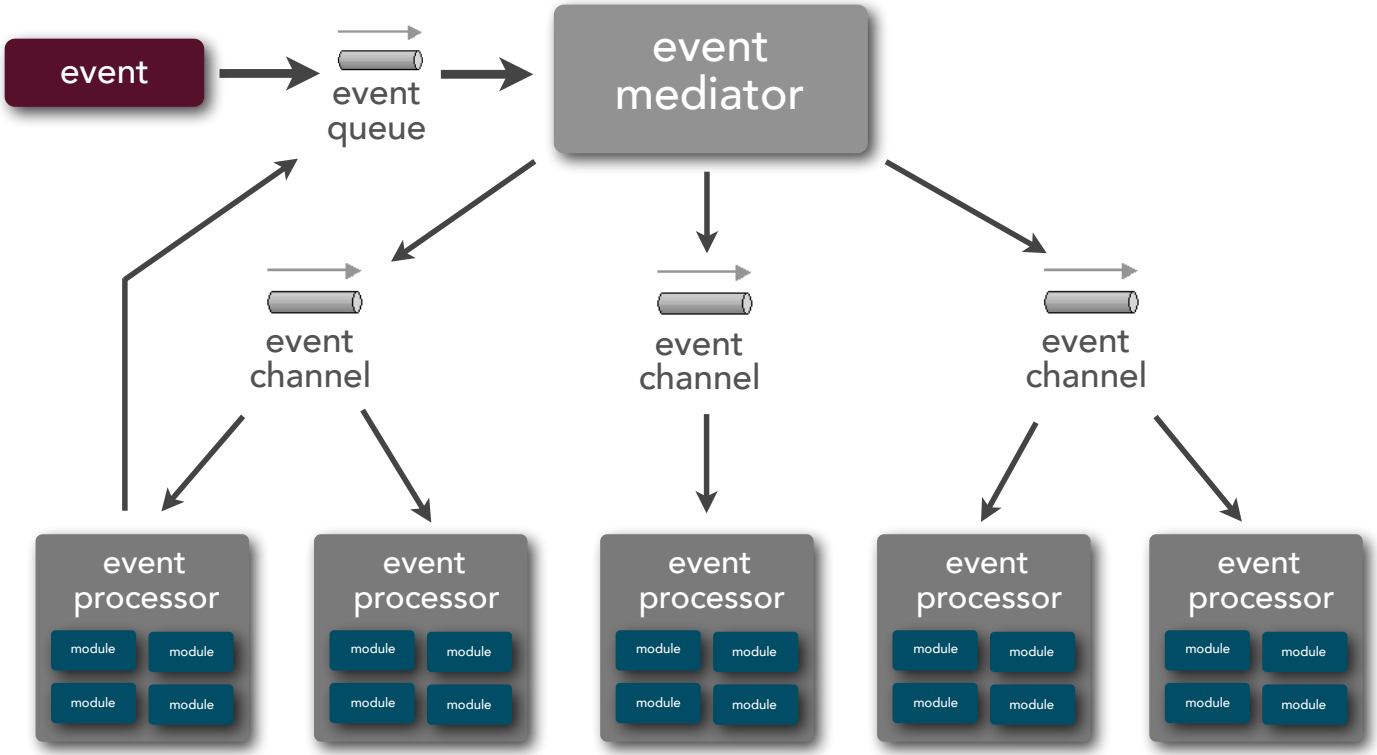


broker topology



mediator topology

Meditator EDA



Mediator Message Flow

you move...

you moved

process engine

change address

recalc quote

update claims

adjust claims

notify insured

customer process

quote process

claims process

adjustment process

notification process

you move...

you moved

process engine

change address

recalc quote

update claims

adjust claims

notify insured

customer process

quote process

claims process

adjustment process

notification process

Error handling?

you move...

you moved

process engine

change address

recalc quote

update claims

adjust claims

notify insured

customer process

quote process

claims process

adjustment process

notification process

Unified notification message to customer?

you move...

you moved

process engine

change address

recalc quote

update claims

adjust claims

notify insured

customer process

quote process

claims process

adjustment process

notification process

audit process

evolution

evolution

you move...

you moved

process engine

change address

recalc quote

update claims

adjust claims

audit

notify insured

customer process

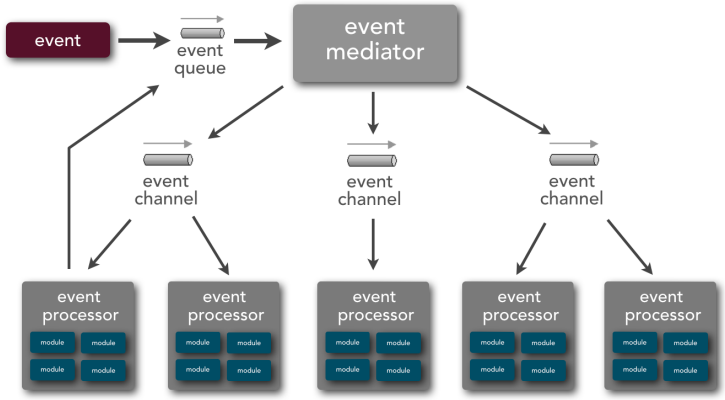
quote process

claims process

adjustment process

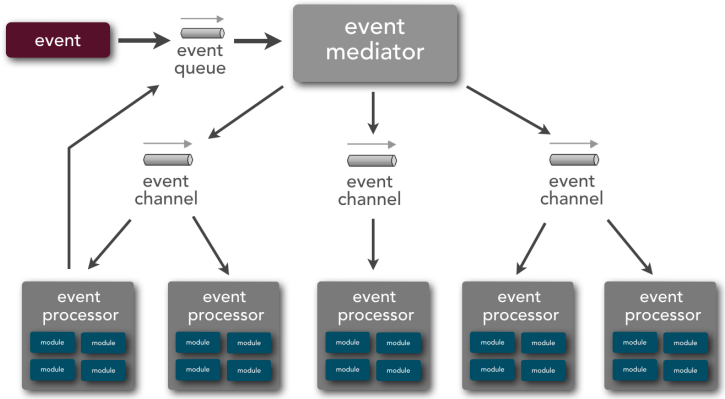
notification process

audit process



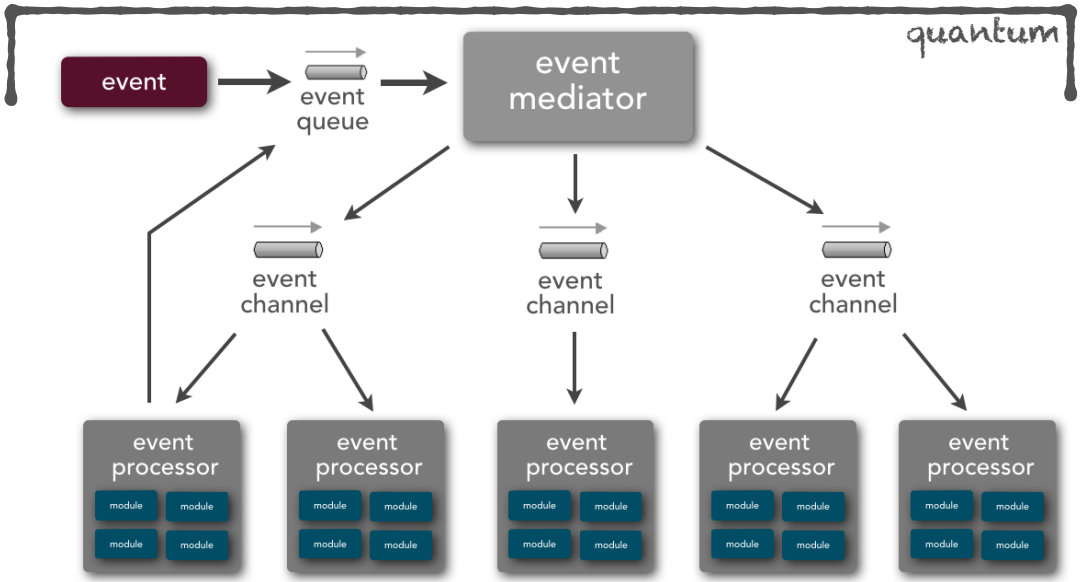
Mediator EDA

services + meditator

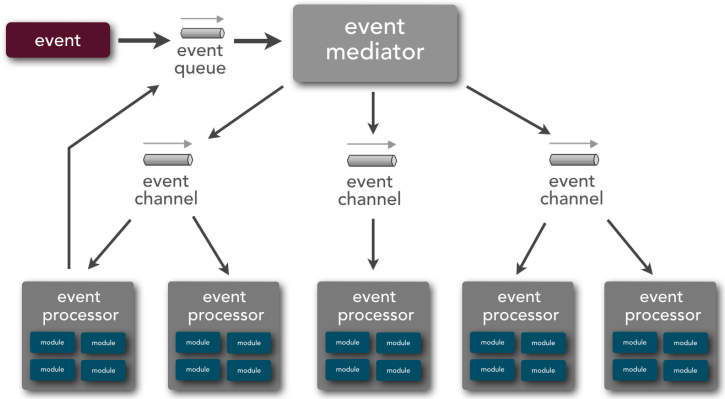


Mediator EDA

services + meditator



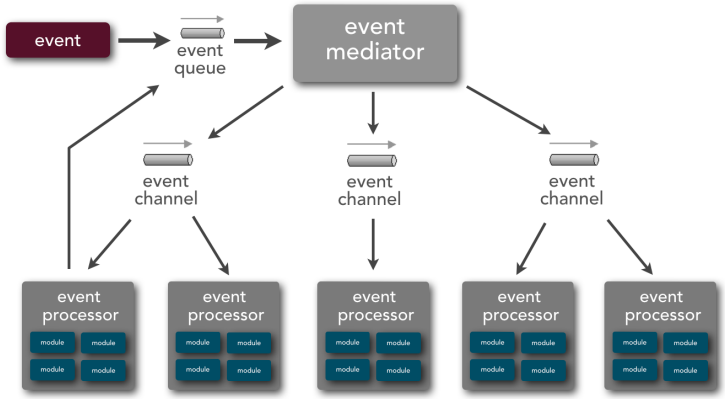
the entire system



Mediator EDA

services + meditator

similar to broker

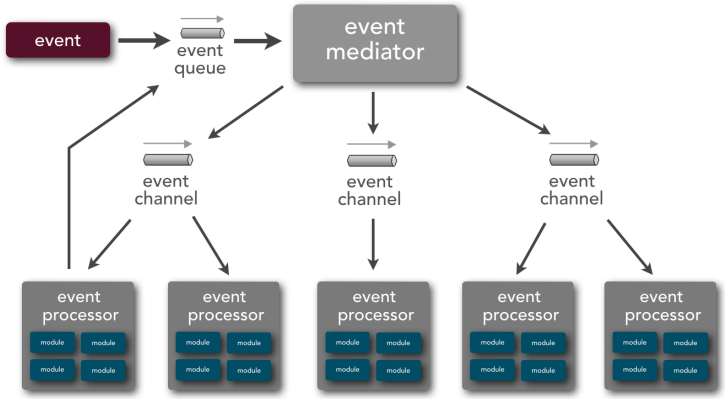


Mediator EDA

services + meditator

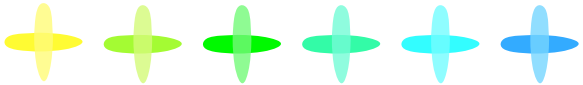
similar to broker

easier integration testing via mediator



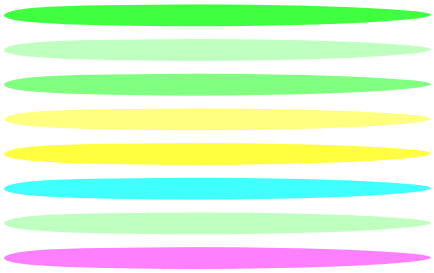
Mediator EDA

services + meditator



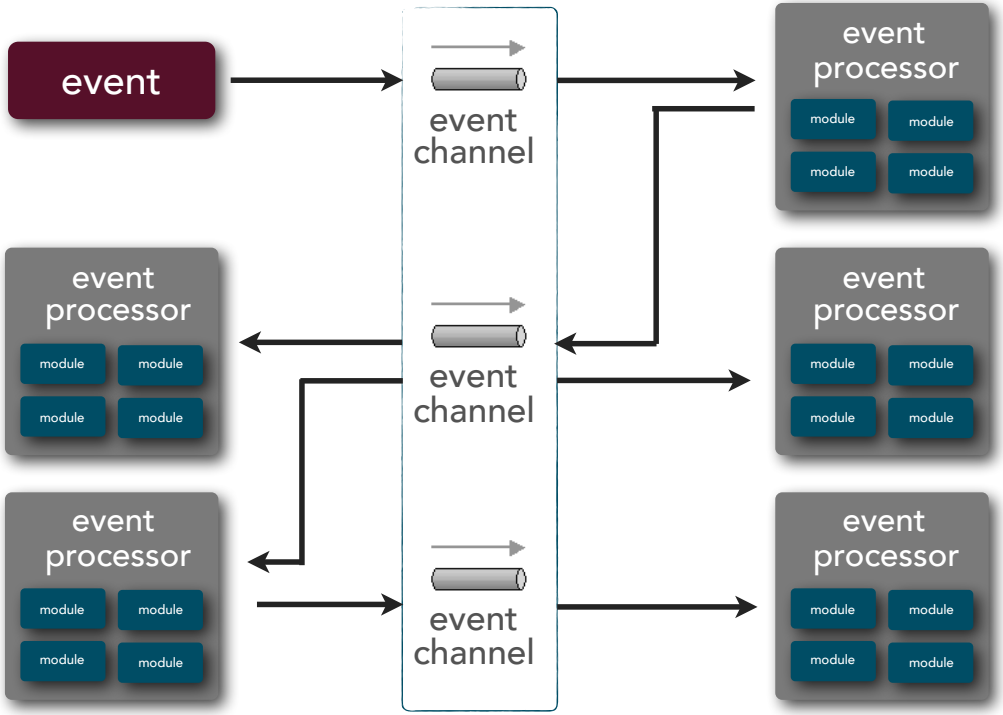
similar to broker

easier integration testing via mediator

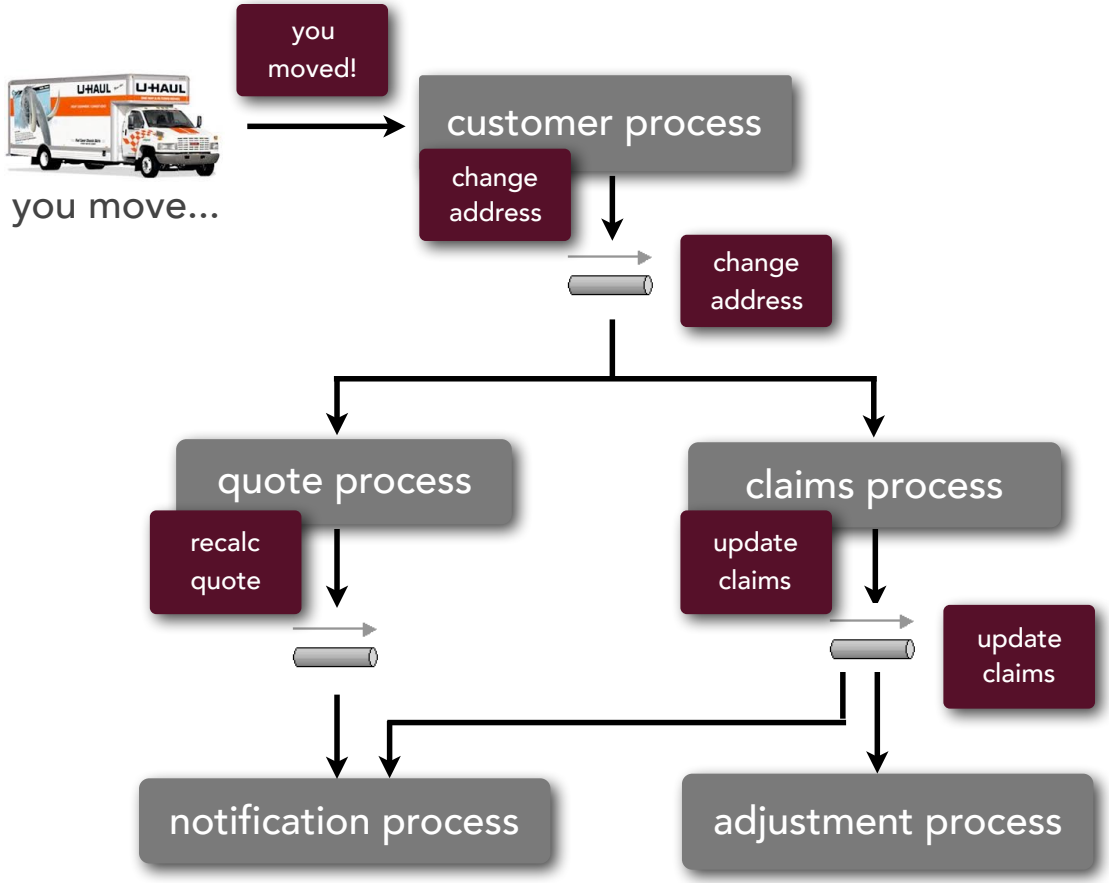


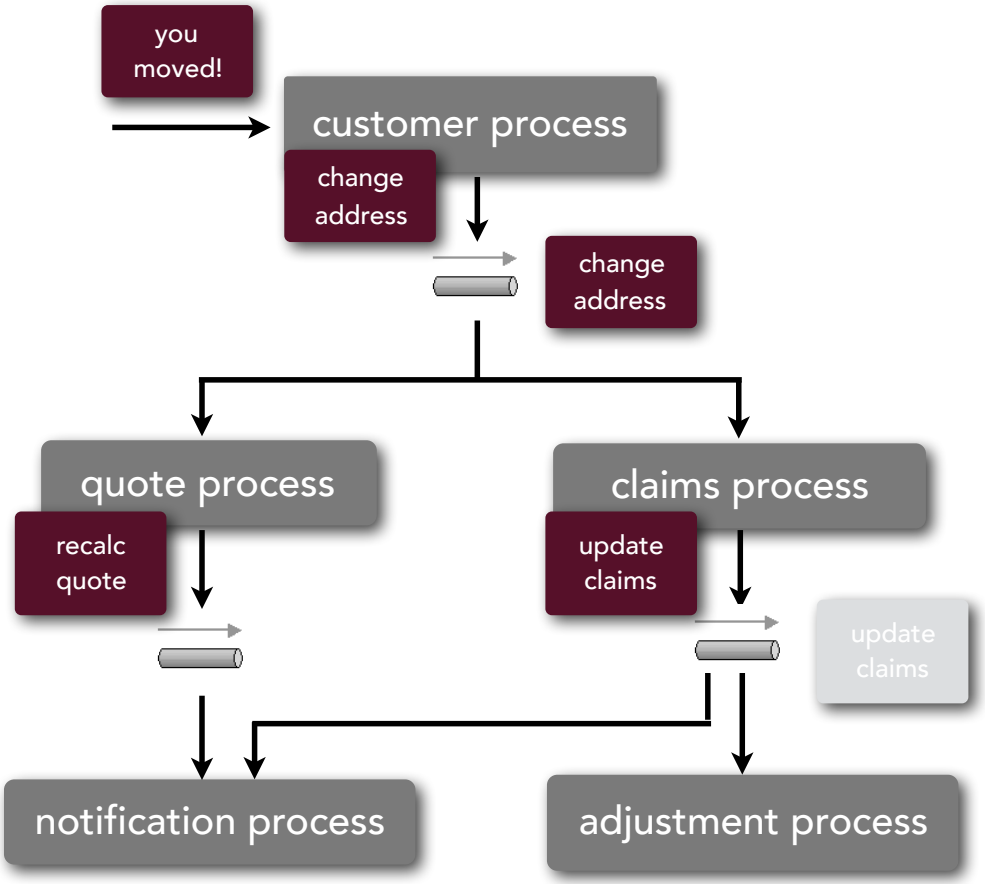
increased semantic coupling

Broker

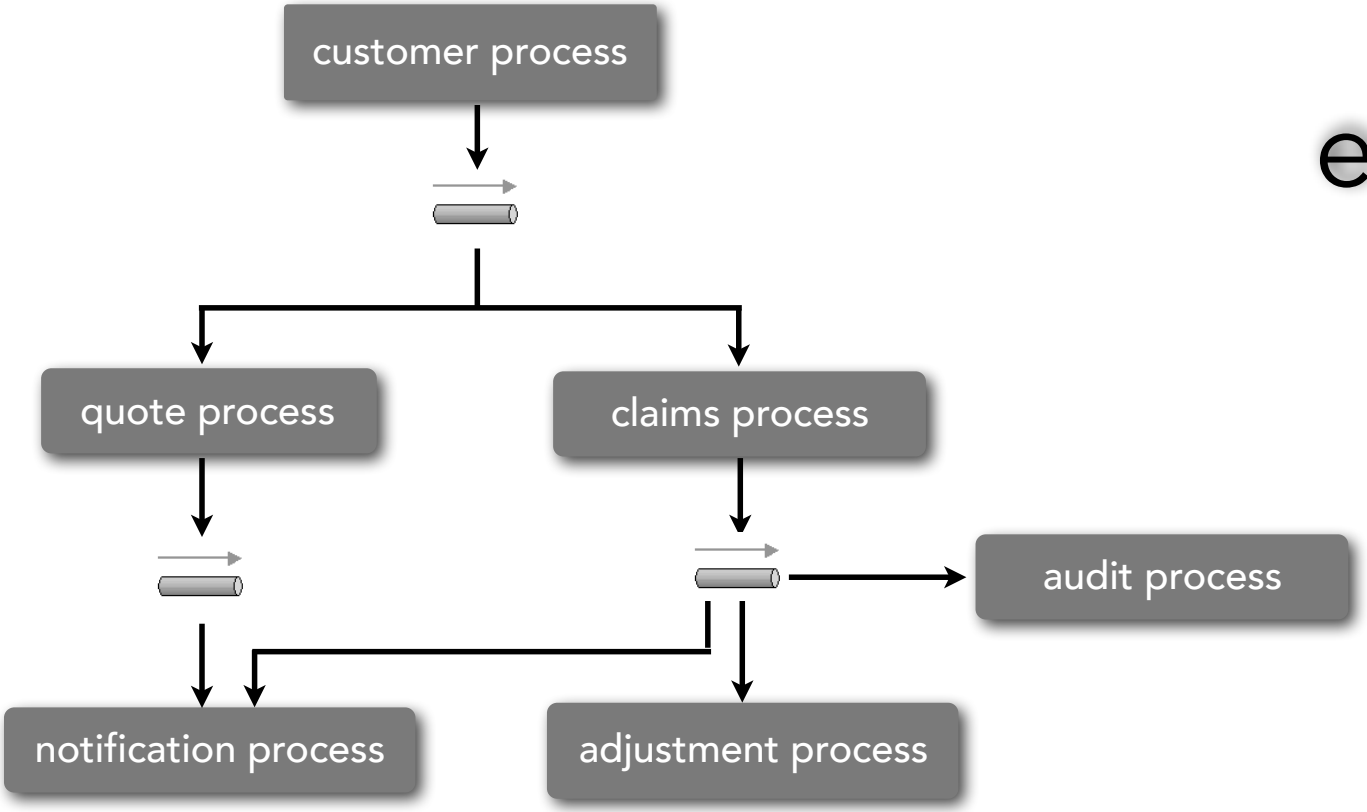

event
event channel
event channel
event channel
event processor
module
module
module
module
event processor
module
module
module
module
event processor
module
module
module
module
event processor
module
module
module
module
event processor
module
module
module
module

Broker Message Passing

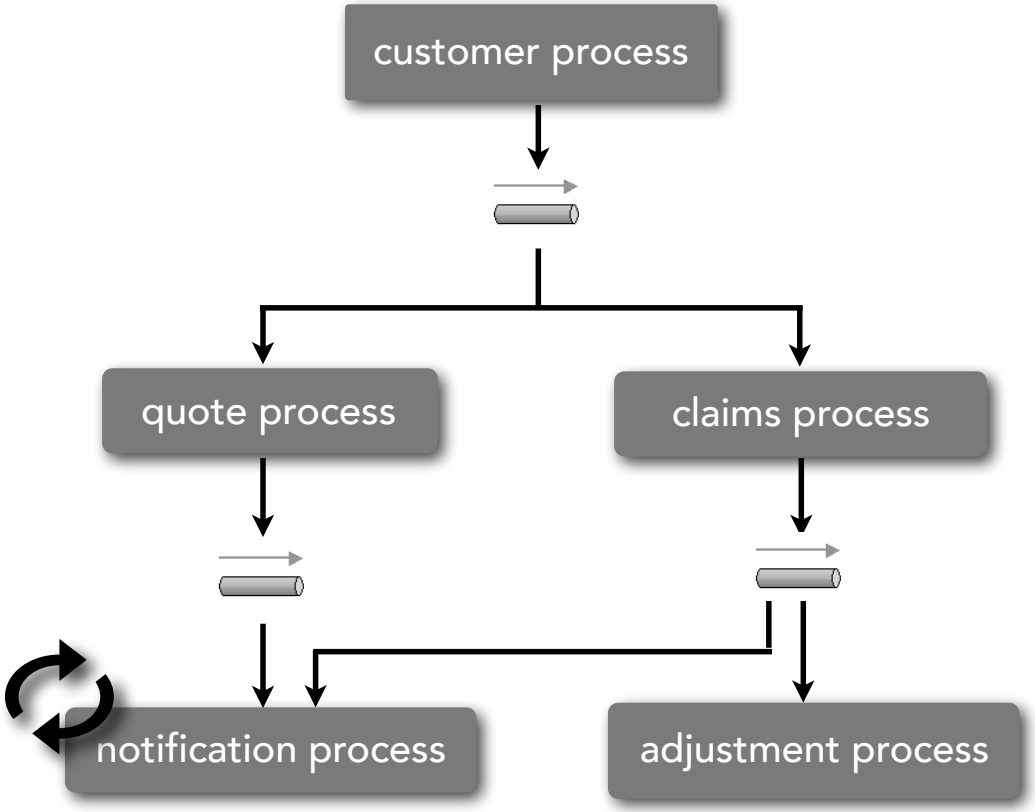




Error handling?

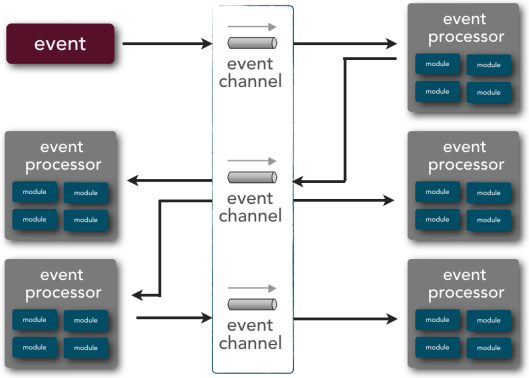

customer process
quote process
claims process
audit process
notification process
adjustment process

evolution



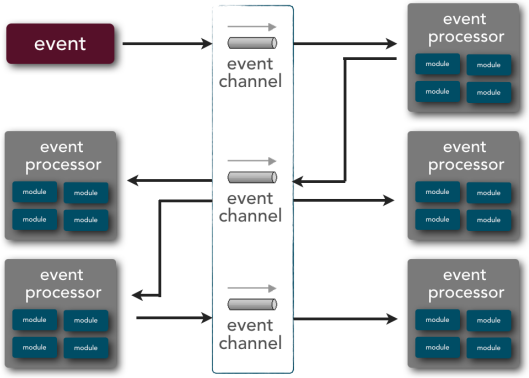
Unified notification message to customer?

In asynchronous systems, coordination happens at the end



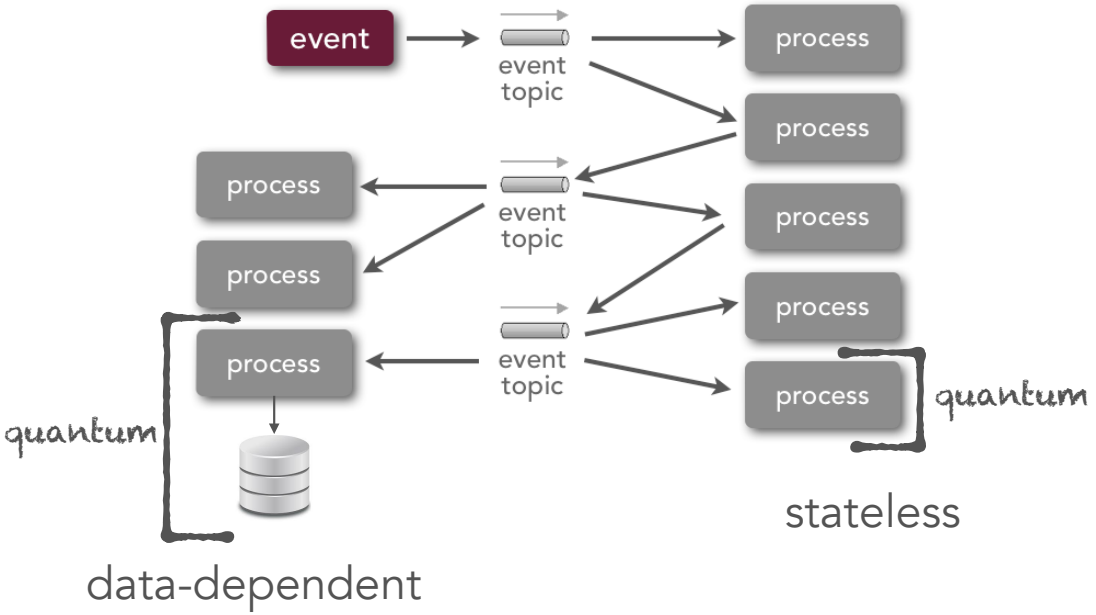
decoupled events

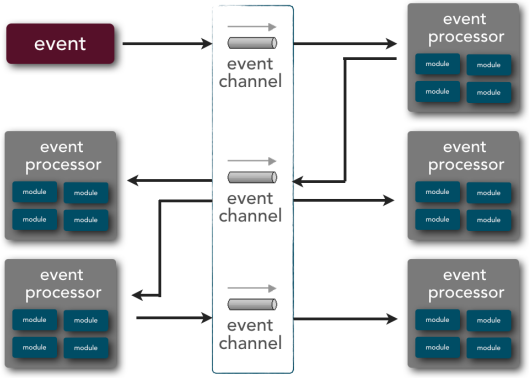
Broker EDA



decoupled events

Broker EDA



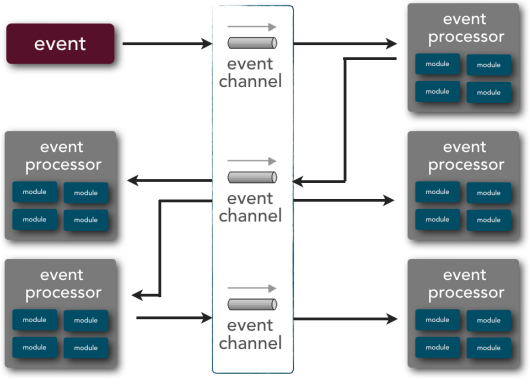


Broker EDA

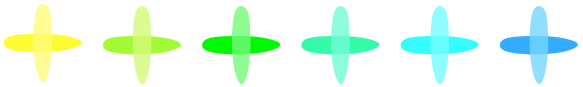
decoupled events

deployment pipeline integration challenging

Broker EDA



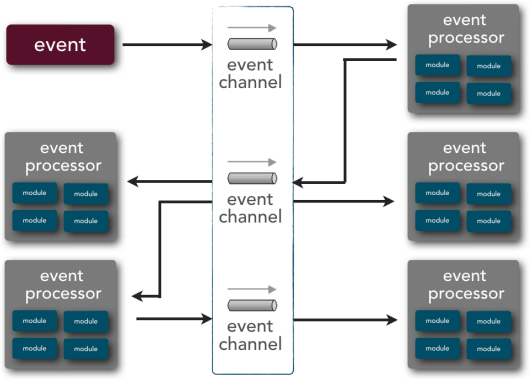
decoupled events



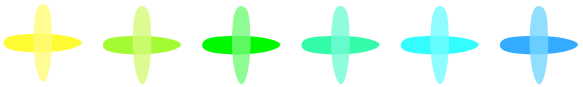
deployment pipeline integration challenging

easy atomic
difficult holistic

Broker EDA



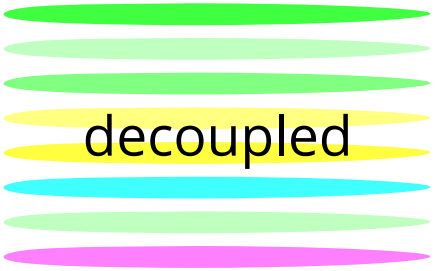
decoupled events



deployment pipeline integration challenging

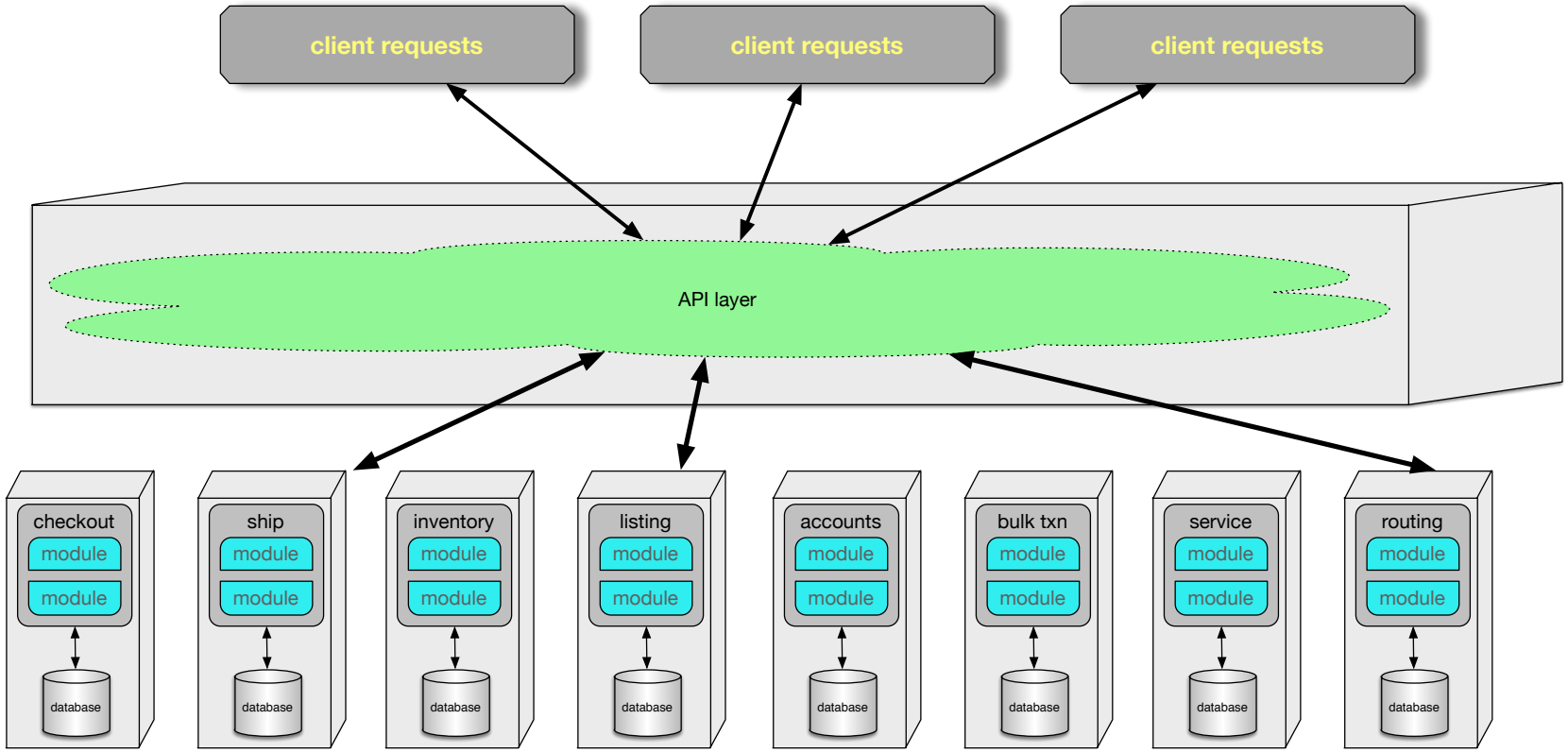
easy atomic
difficult holistic

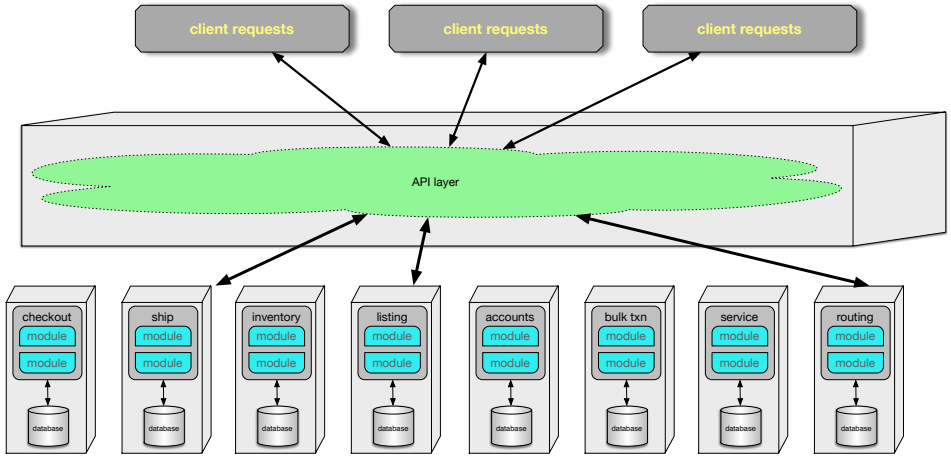
structurally decoupled



semantically decoupled

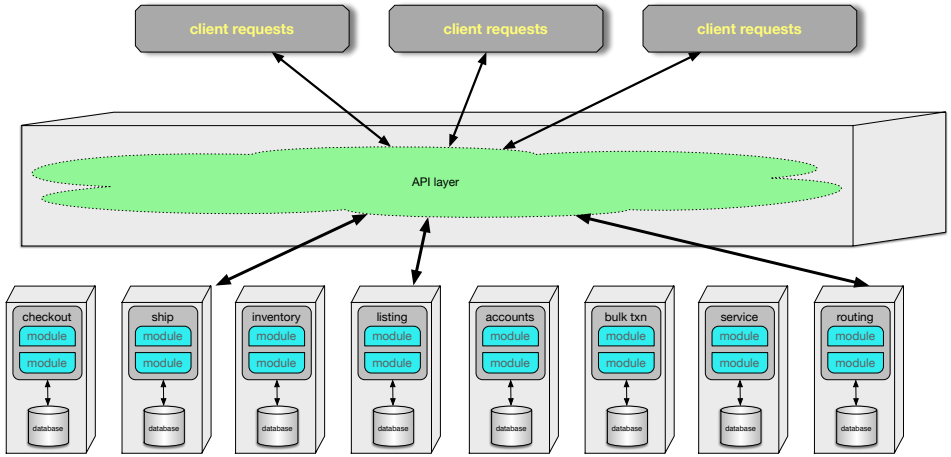
Microservices





Microservices

extremely fine grained

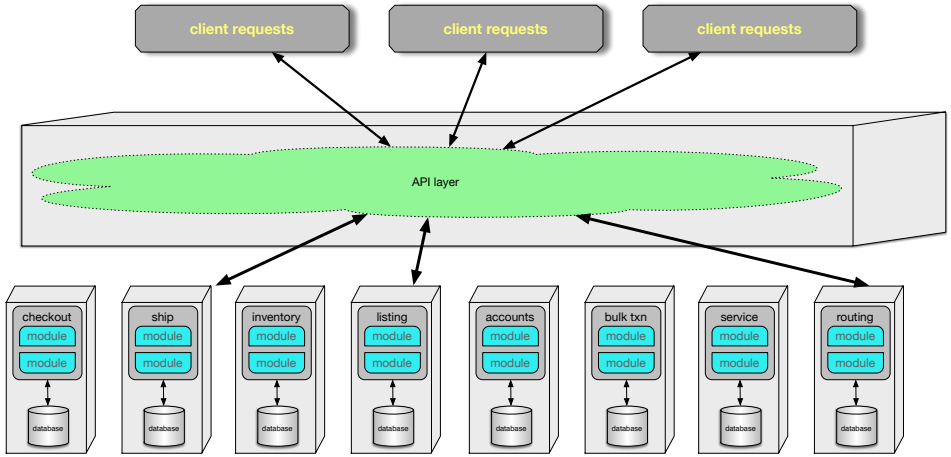


Microservices

extremely fine grained

deployment pipeline considered standard

21st century DevOps practices



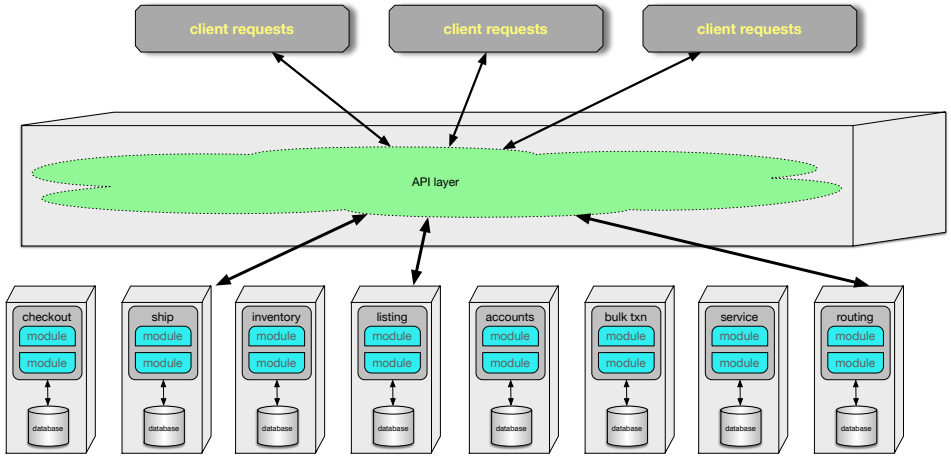
Microservices

extremely fine grained

deployment pipeline considered standard

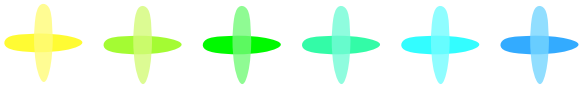
21st century DevOps practices

robust testing & fitness function culture



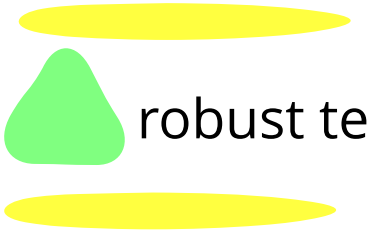
Microservices

extremely fine grained

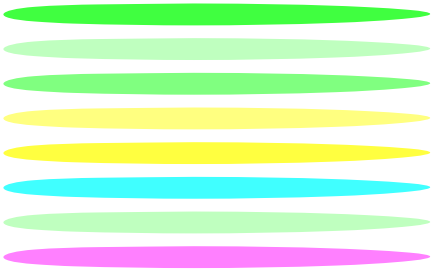


deployment pipeline considered standard

21st century DevOps practices

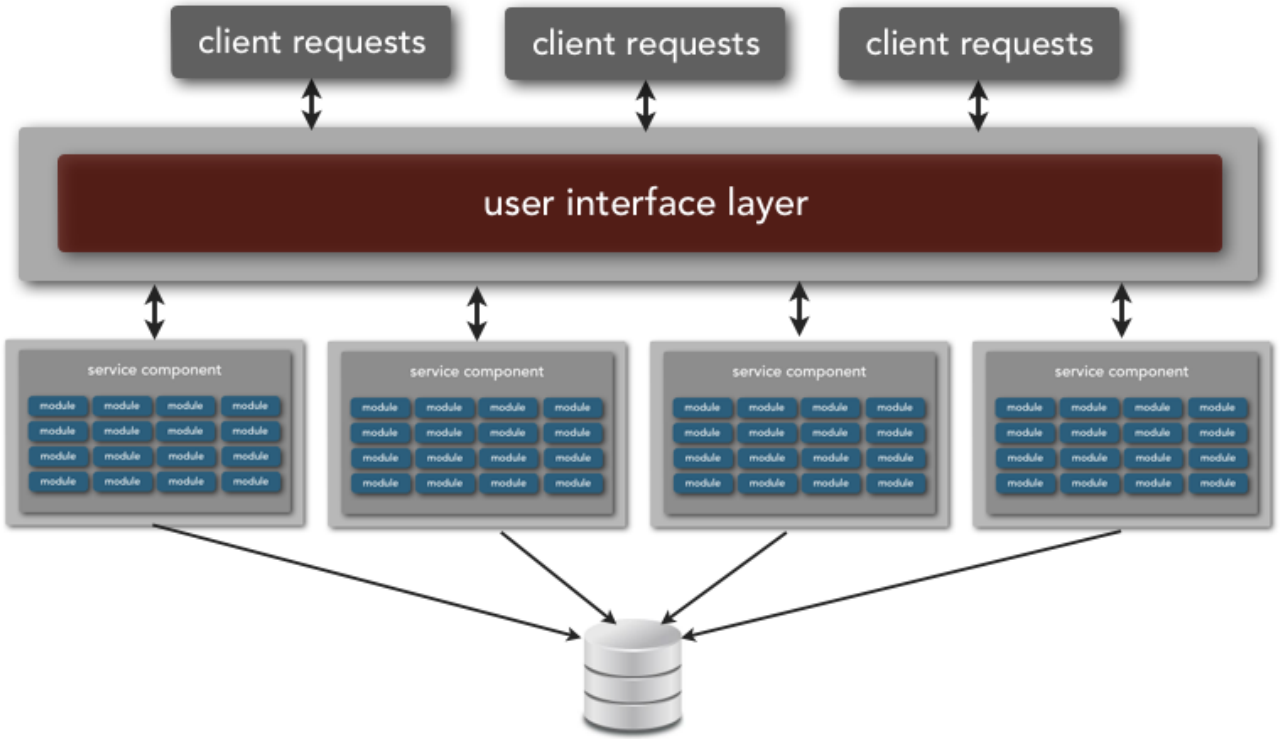


robust testing & fitness function culture



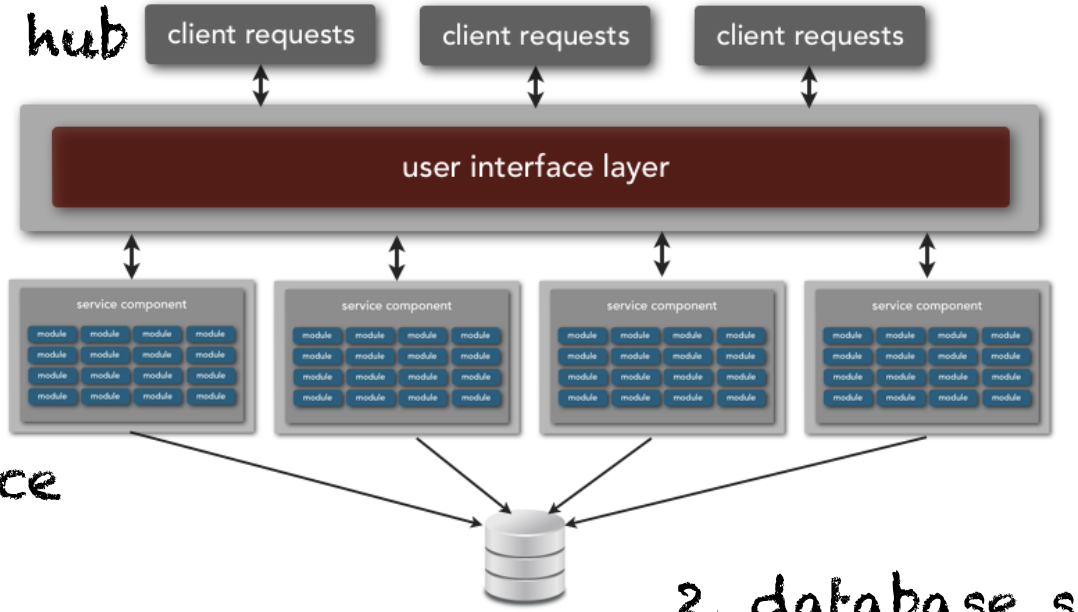
extremely decoupled fine-grained
quanta with well defined boundaries

Service-based Architectures



Service-based Architectures

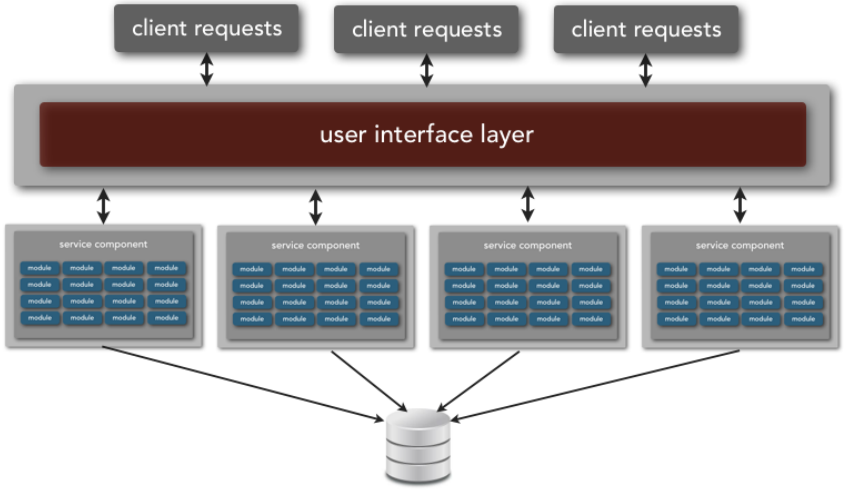
3. use of service bus as integration hub



1. Larger service granularity

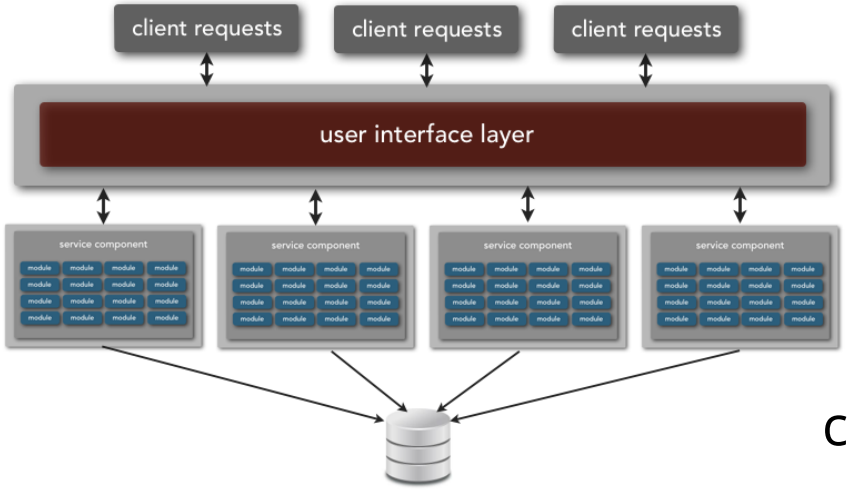
2. database scope

Service-based Architectures

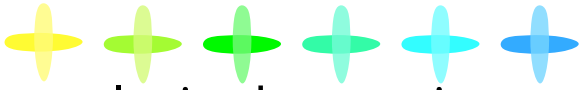


larger than microservices

Service-based Architectures

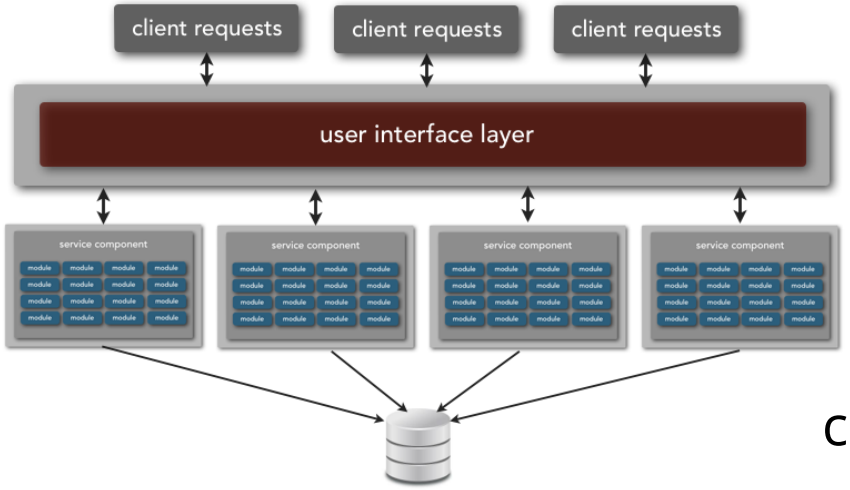


larger than microservices

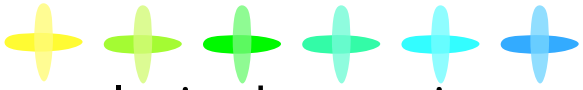


chunkier granularity harms incremental change

Service-based Architectures



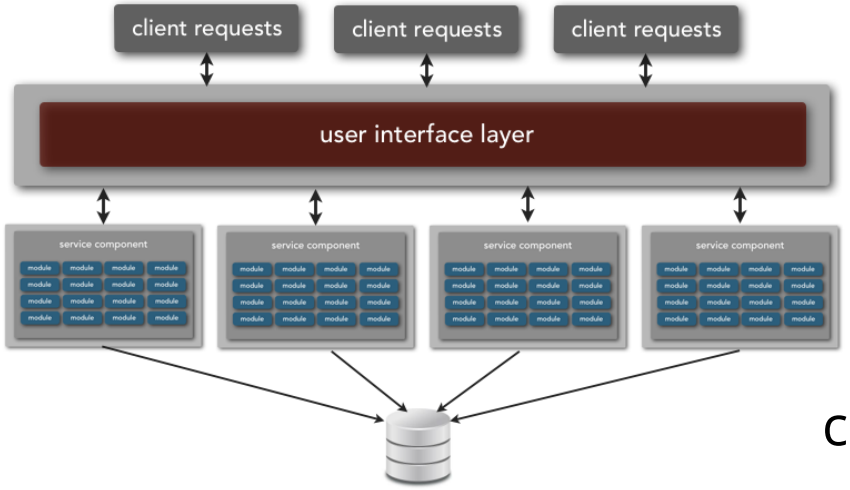
larger than microservices



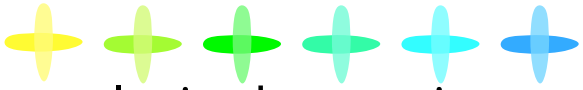
chunkier granularity harms incremental change

bigger scope of tests and other fitness functions

Service-based Architectures

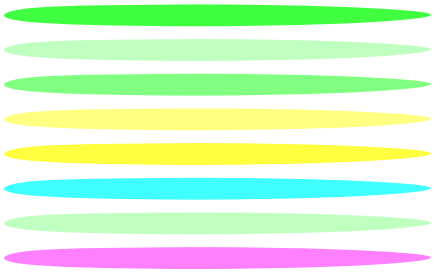


larger than microservices



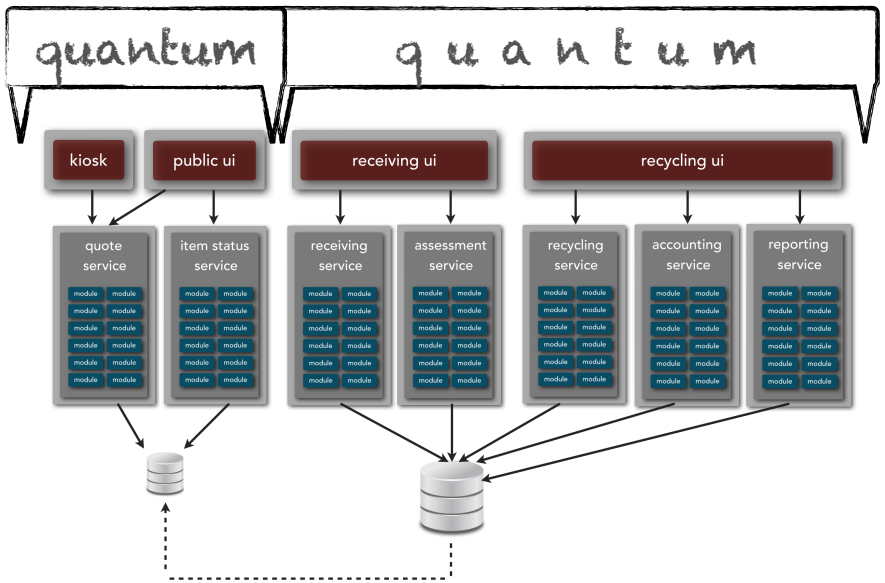
chunkier granularity harms incremental change

bigger scope of tests and other fitness functions

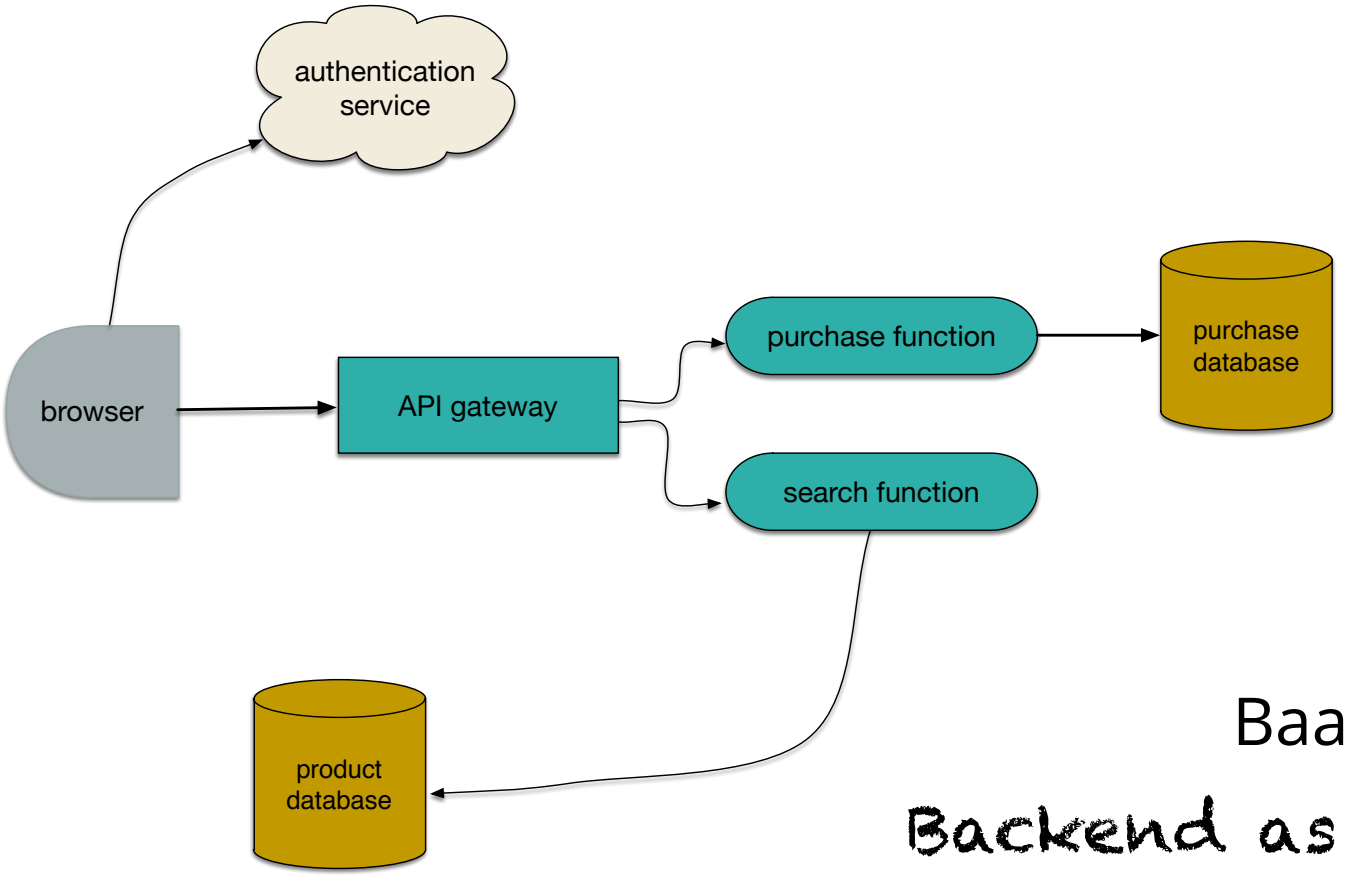


benefits from domain-centric architecture

Reducing Quanta Size



“Serverless” Architecture



BaaS

Backend as a Service

“Serverless” Architecture

FaaS

Function as a Service

API Gateway

Amazon Lambda

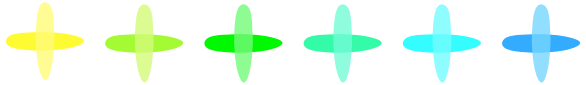
DynamoDB

“Serverless” Architecture

Database?

“Serverless” Architecture

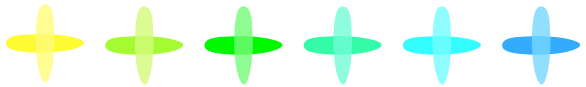
Database?



redeploy code

“Serverless” Architecture

Database?

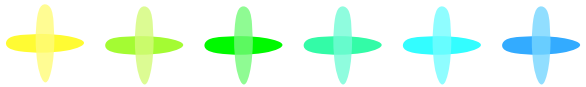


redeploy code

critical

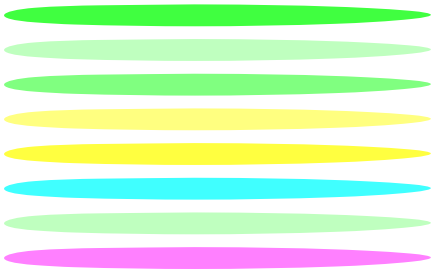
“Serverless” Architecture

Database?



redeploy code

critical

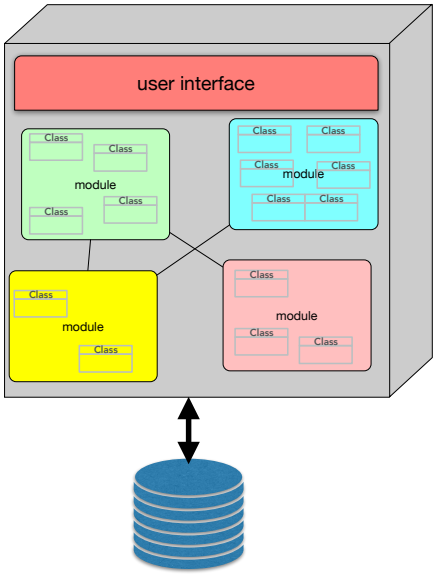


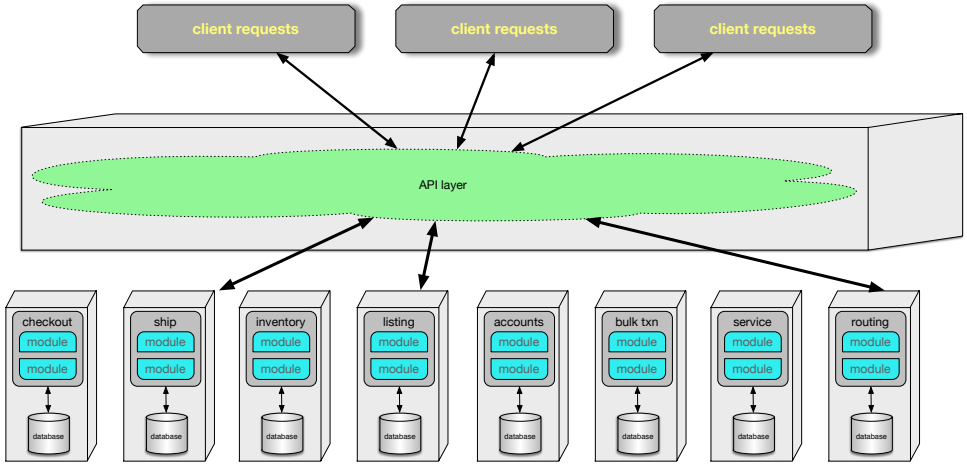
deployment pipeline integration challenging

Smaller Quanta Size

≙

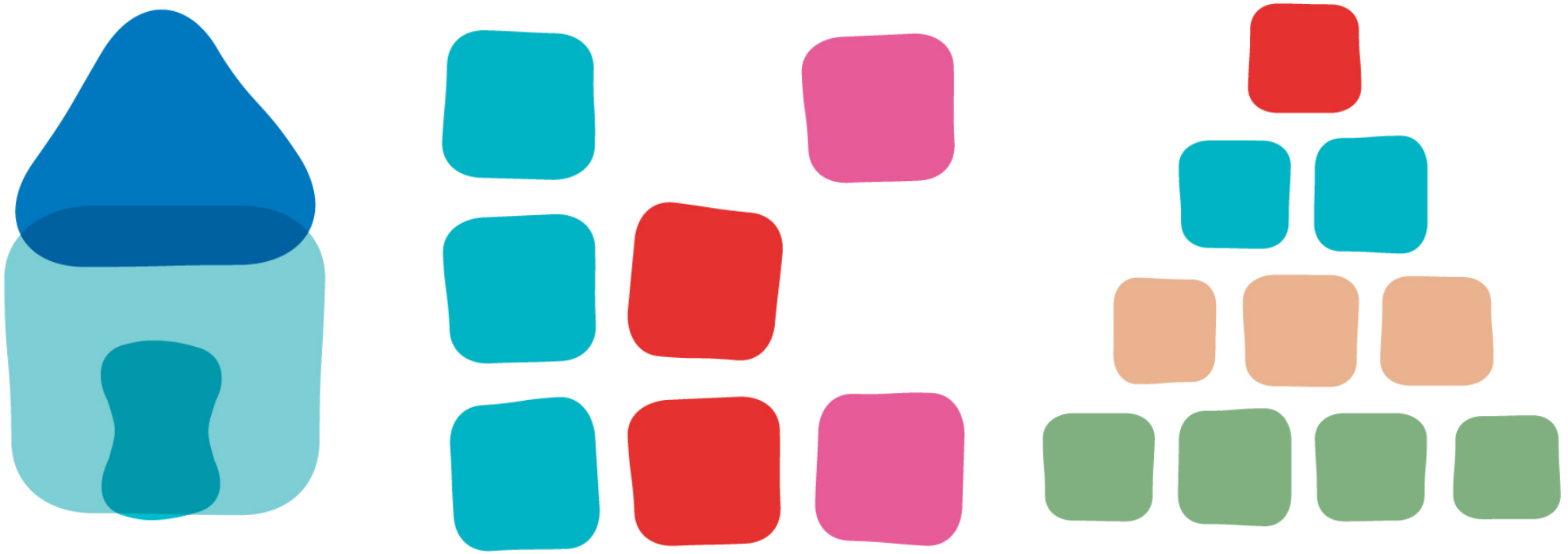
Evolutionary



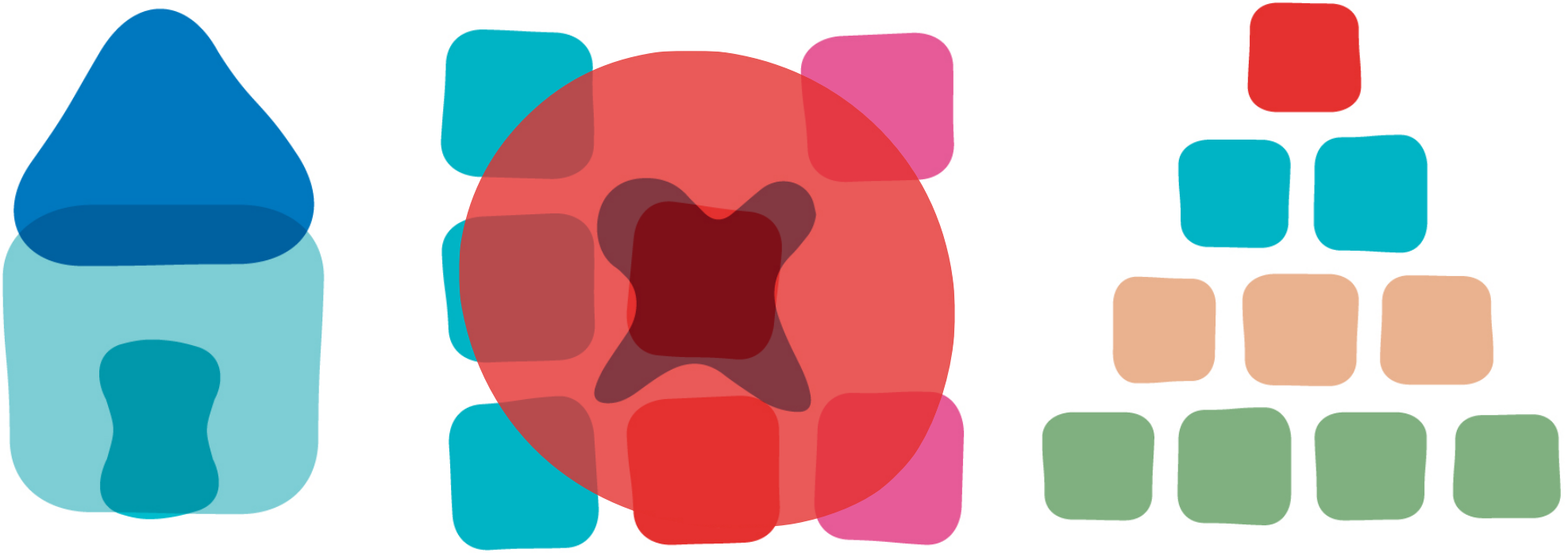


Domain/Architecture Isomorphism

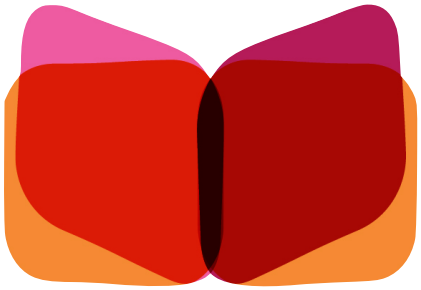
Domain/Architecture Isomorphism



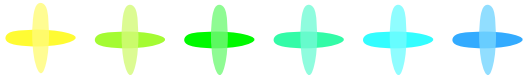
Domain/Architecture Isomorphism



Agenda Part 1

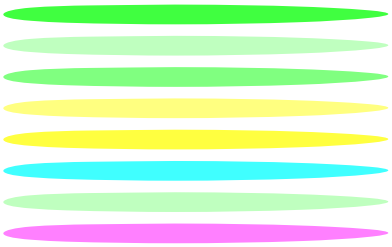


Definition



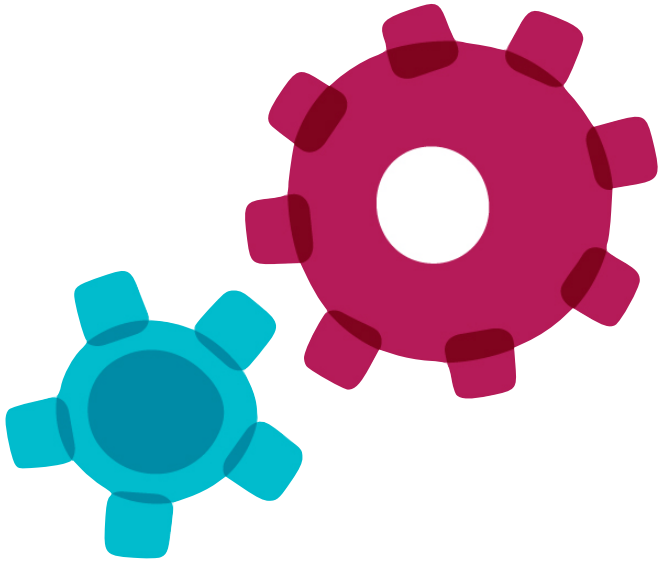
Incremental Change

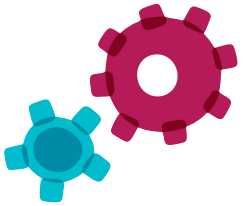
Guided Change via Fitness Functions



Structuring architecture for evolution

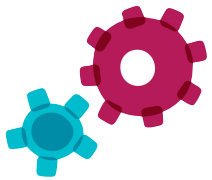
Putting evolutionary architecture into Practice



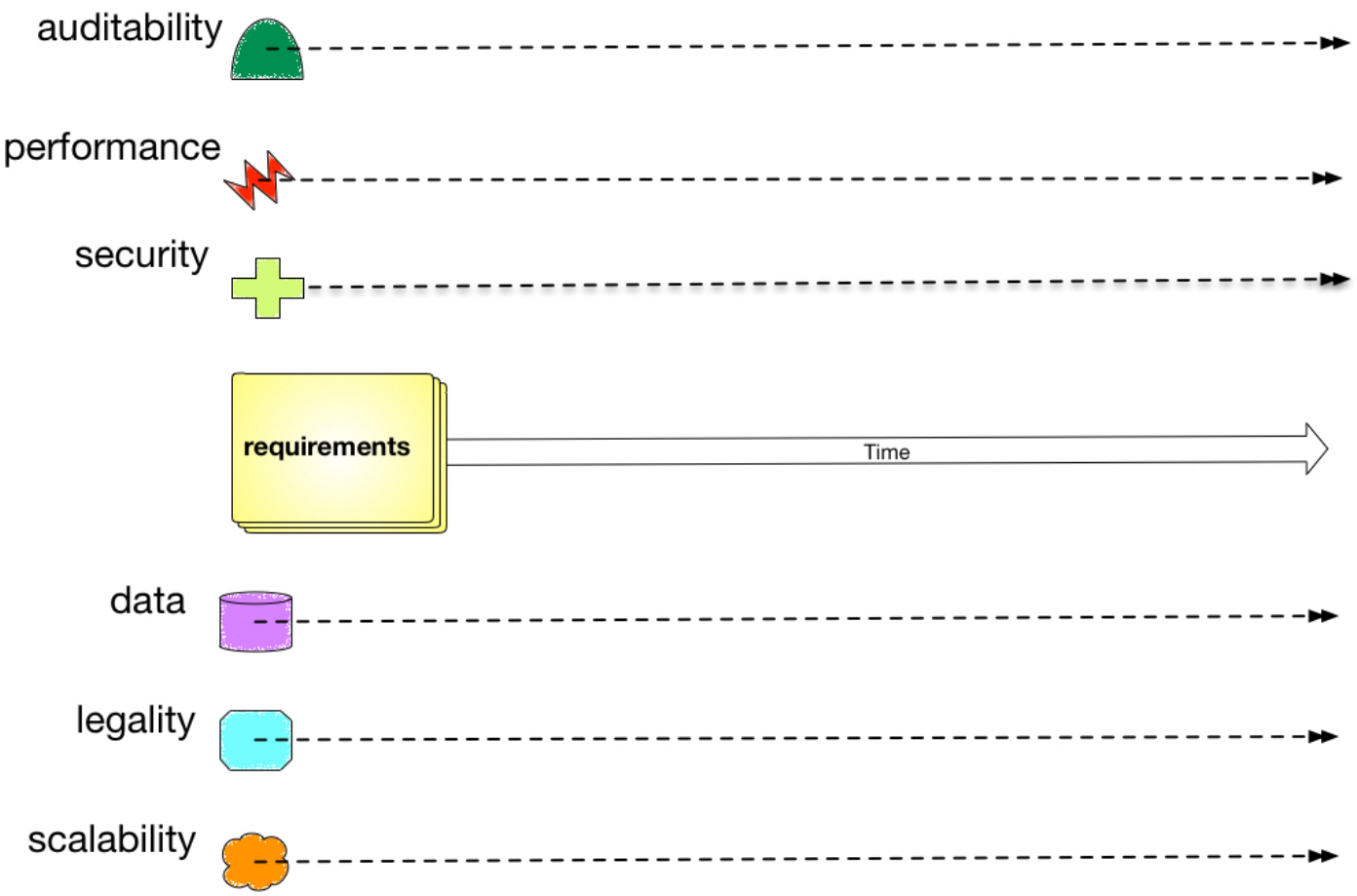


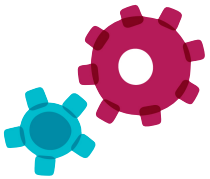
Mechanics

1. Identify dimensions



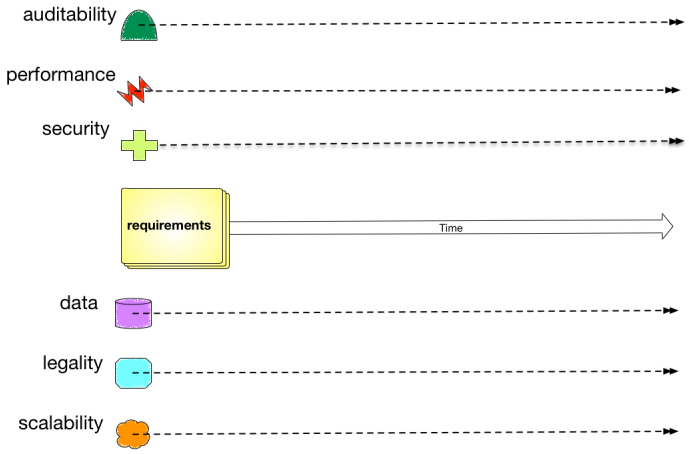
1. Identify dimensions

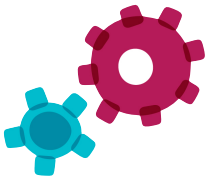




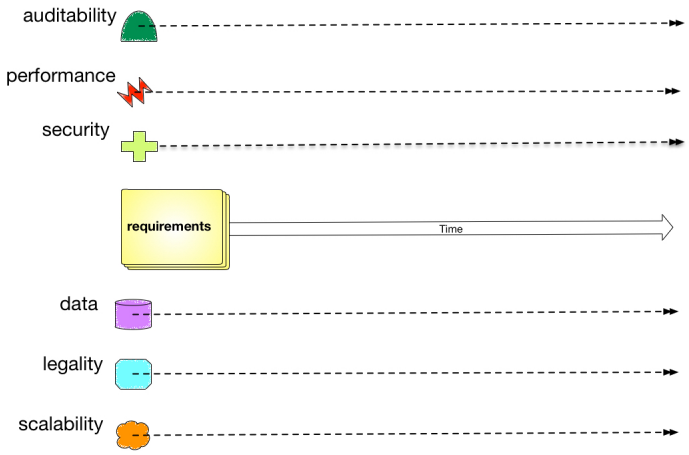
1. Identify dimensions

prioritization



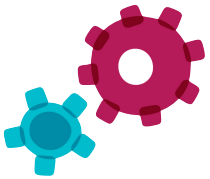


1. Identify dimensions

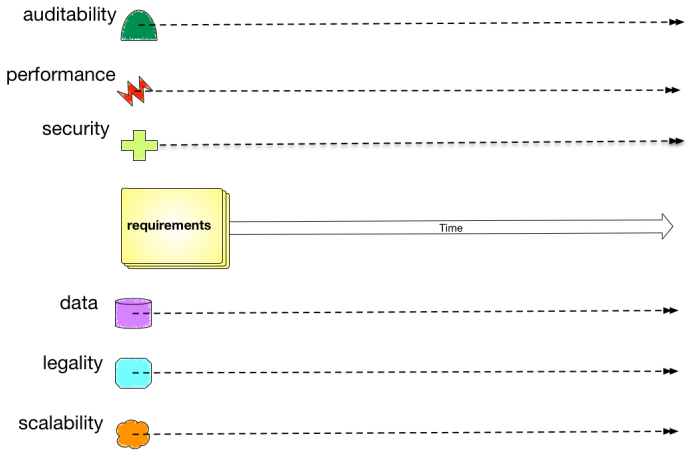


prioritization

cost to implement & maintain



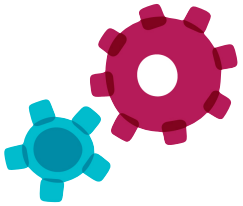
1. Identify dimensions



prioritization

cost to implement & maintain

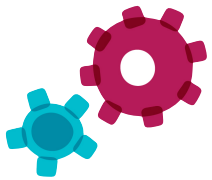
change variable “fit it” cost to constant maintenance cost



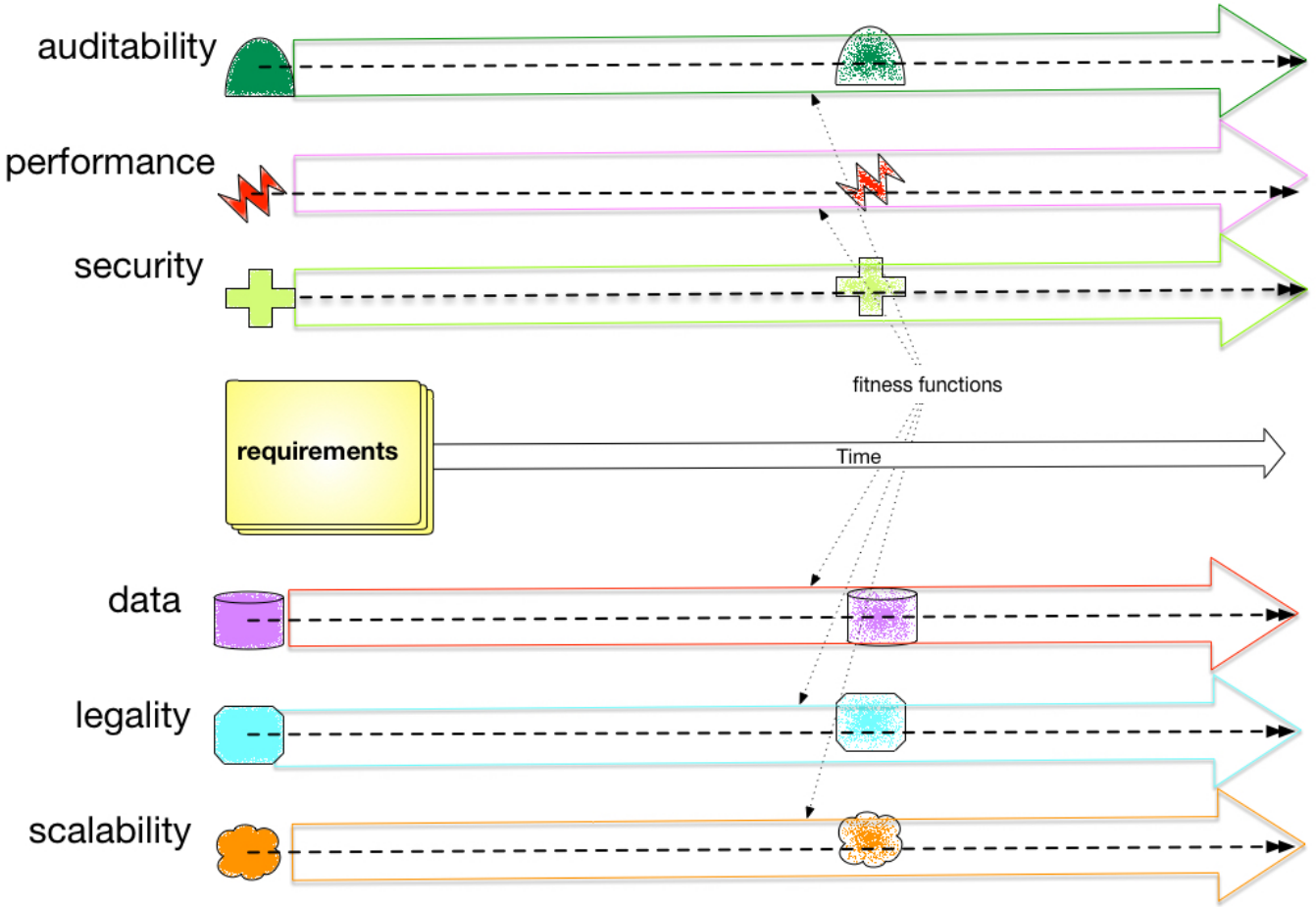
Mechanics

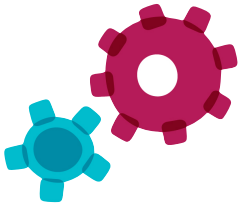
1. Identify dimensions

2. Define fitness function(s)



2. Define Fitness Function(s)



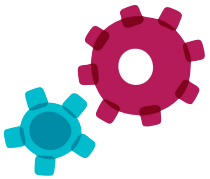


Mechanics

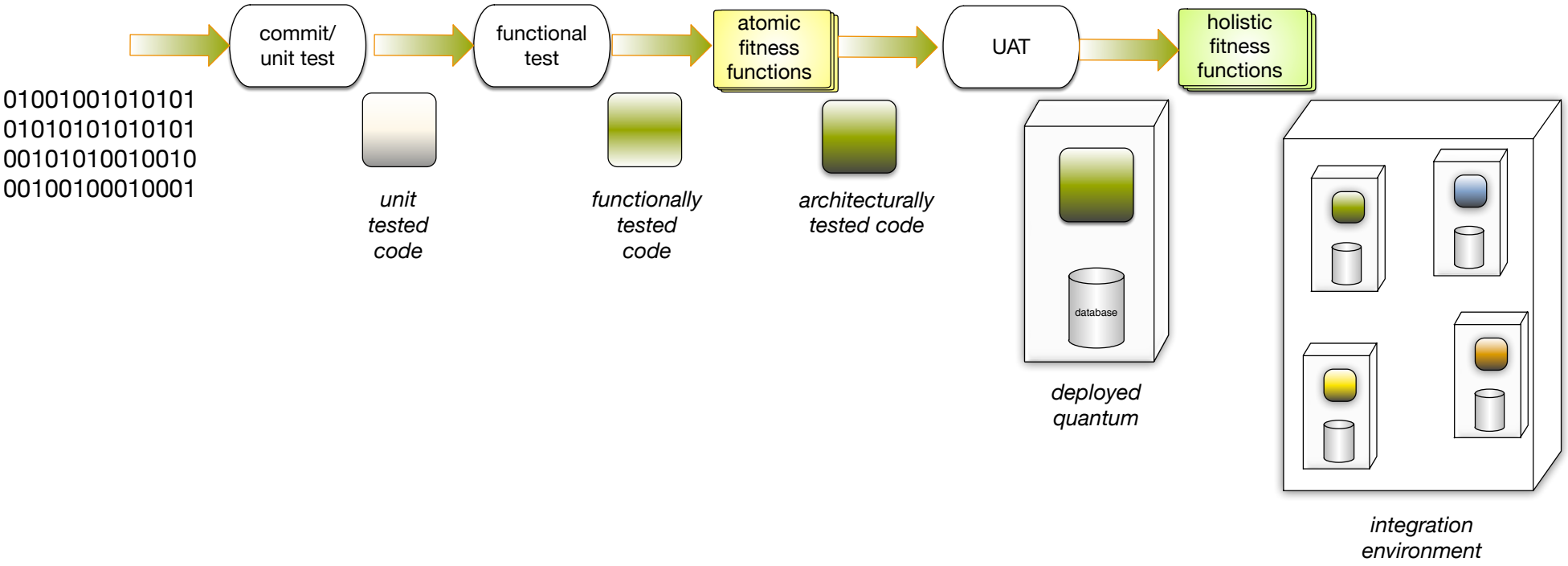
1. Identify dimensions

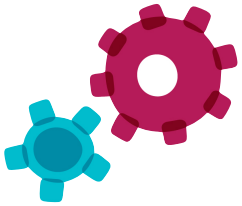
2. Define fitness function(s)

3. Use deployment pipelines and/or continuous fitness functions



3. Use deployment pipelines and/or continuous fitness functions



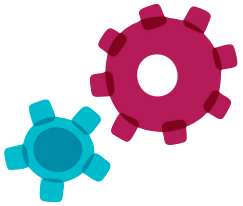


Mechanics

1. Identify dimensions

2. Define fitness function(s)

3. Use deployment pipelines and/or continuous fitness functions



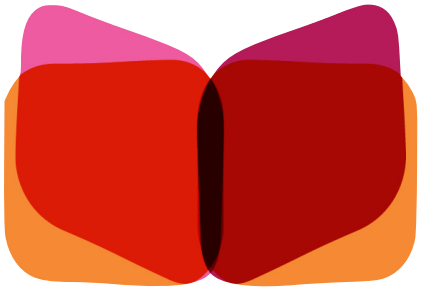
Mechanics

1. Identify dimensions

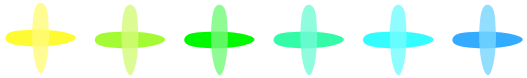
2. Define fitness function(s)

3. Use deployment pipelines and/or continuous fitness functions

Agenda Part 1

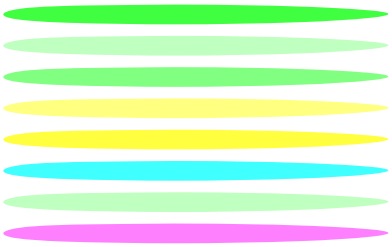


Definition



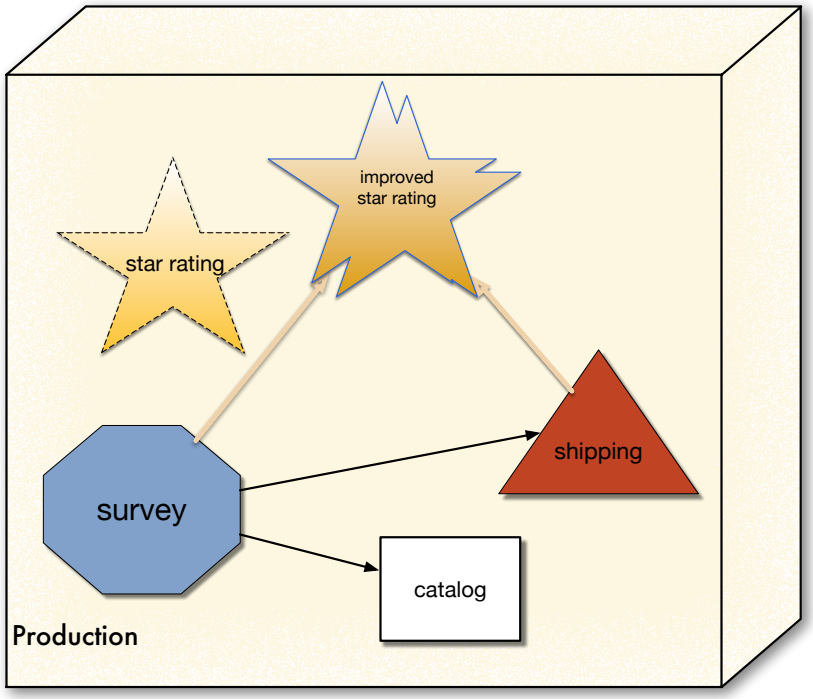
Incremental Change

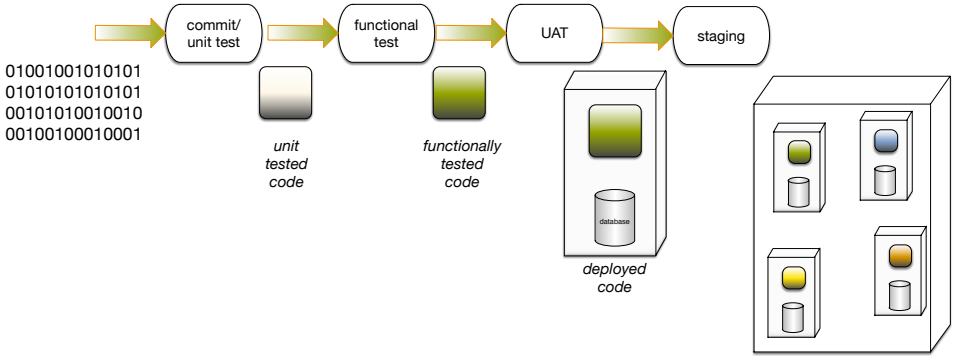
Guided Change via Fitness Functions



Structuring architecture for evolution

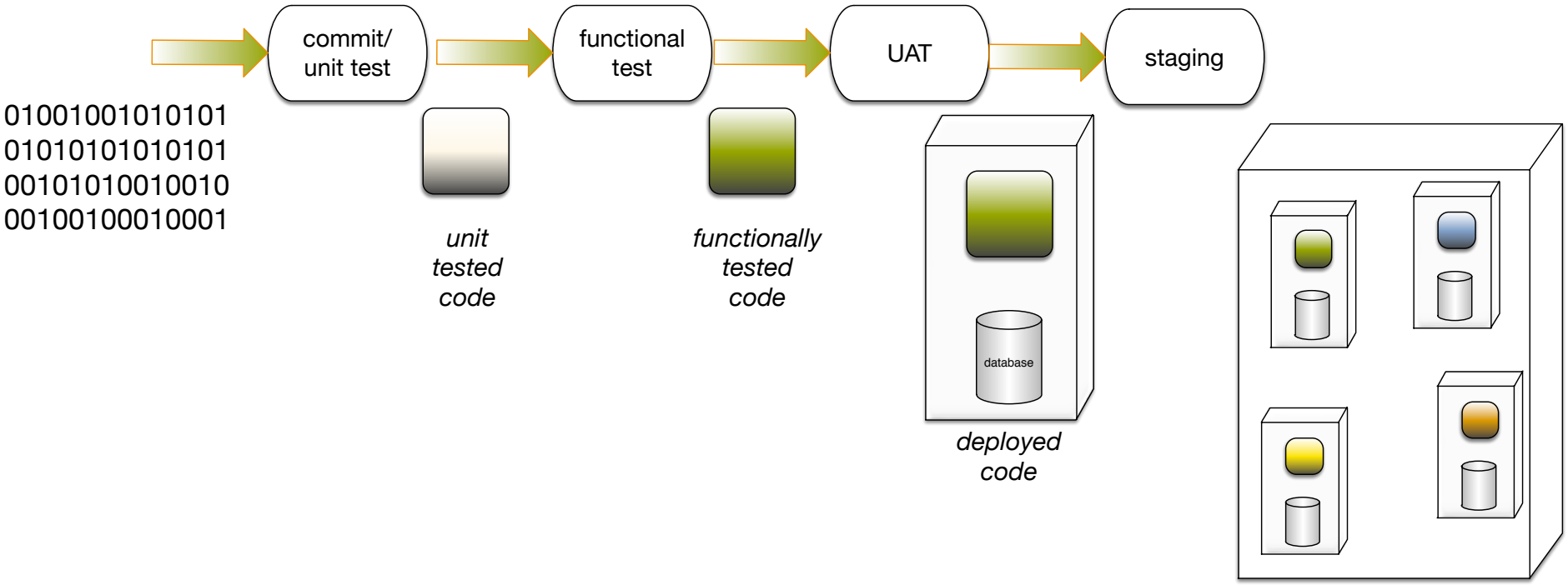
incremental



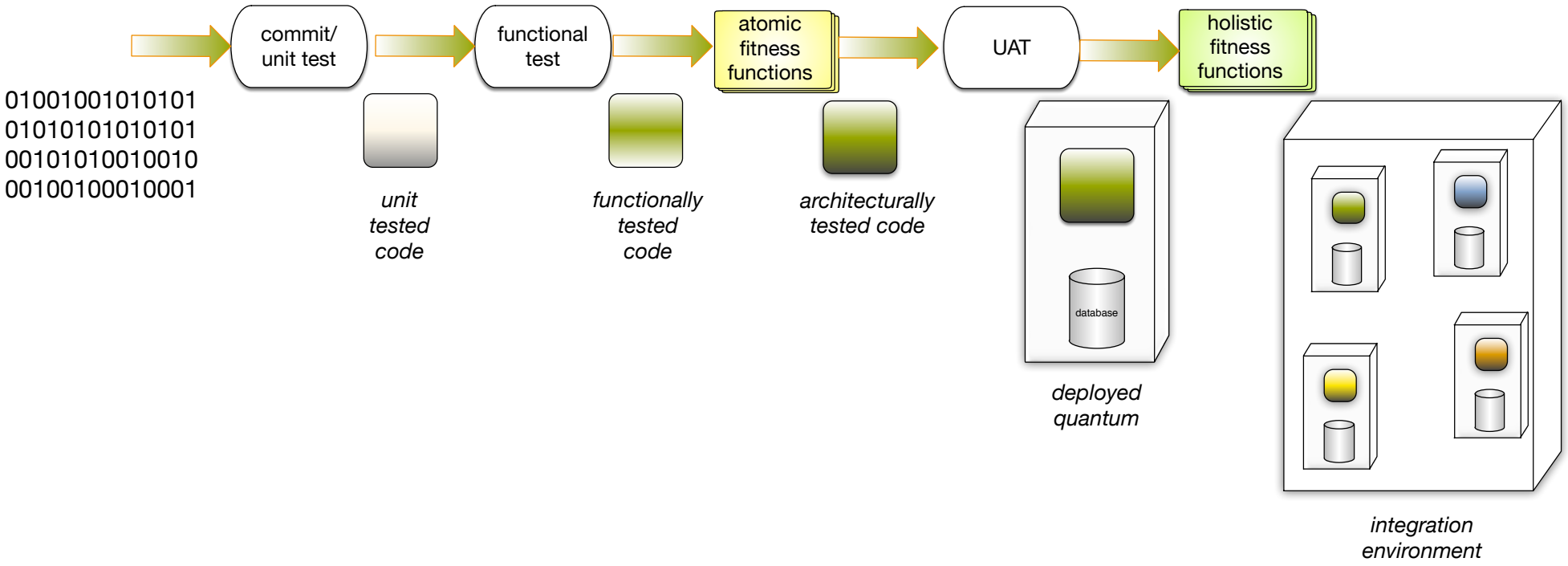


Deployment Pipelines

Deployment Pipeline

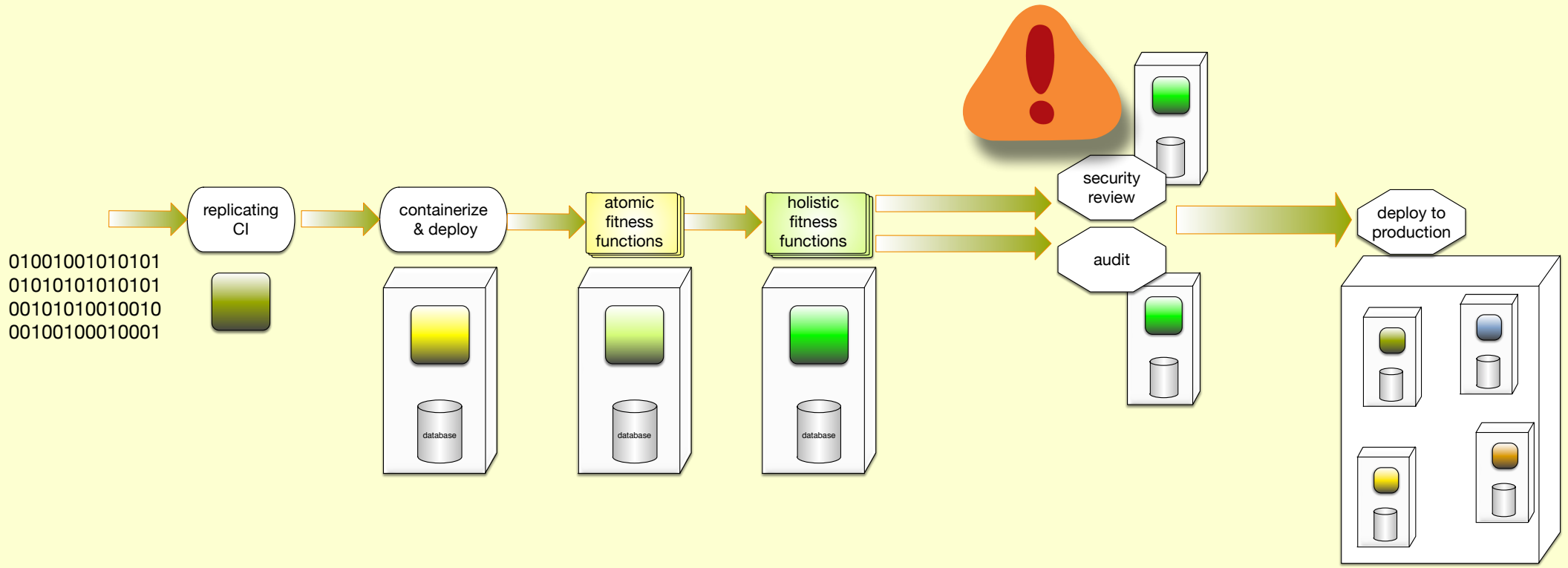


Deployment Pipeline + Fitness Functions

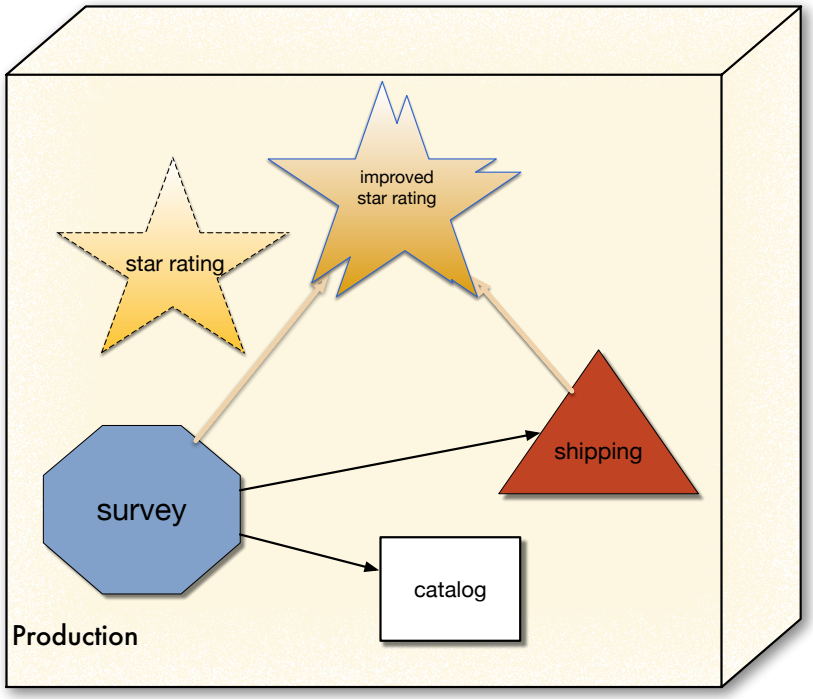


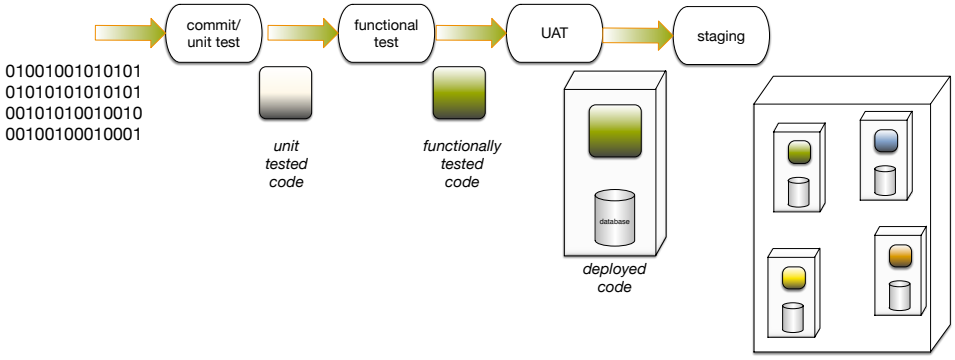
Penultimate

Deployment Pipeline

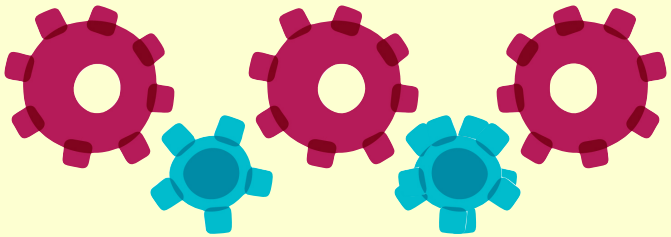


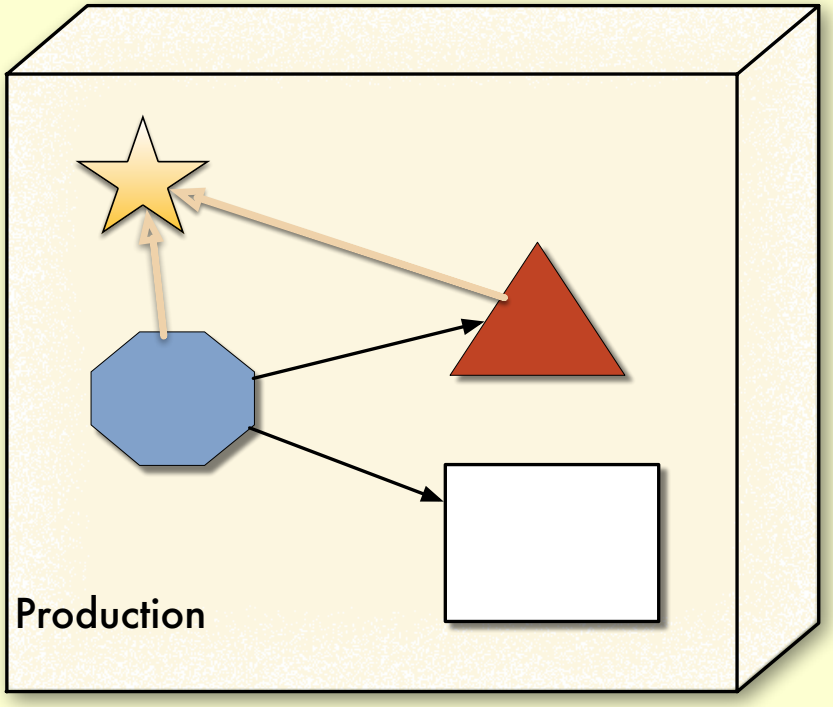
incremental



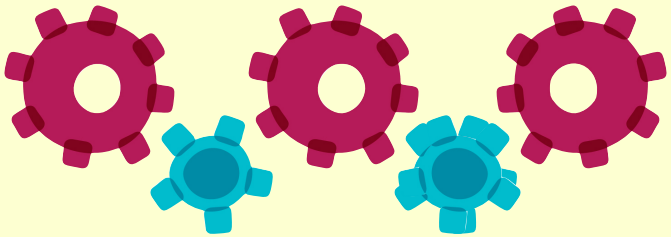


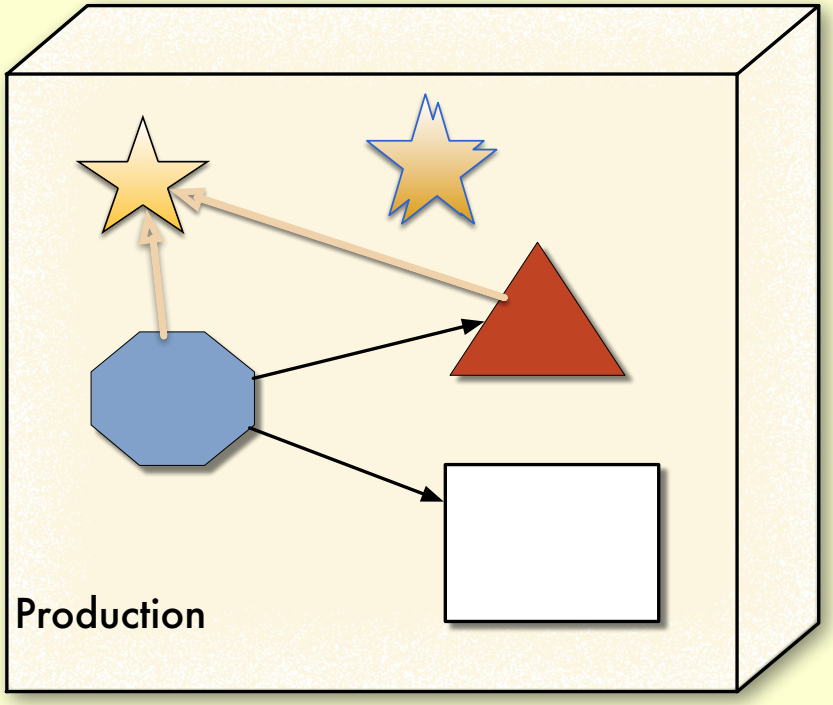
Penultimate



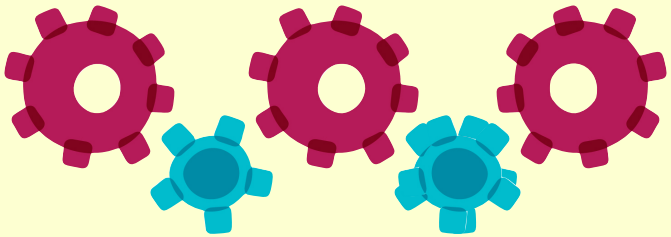


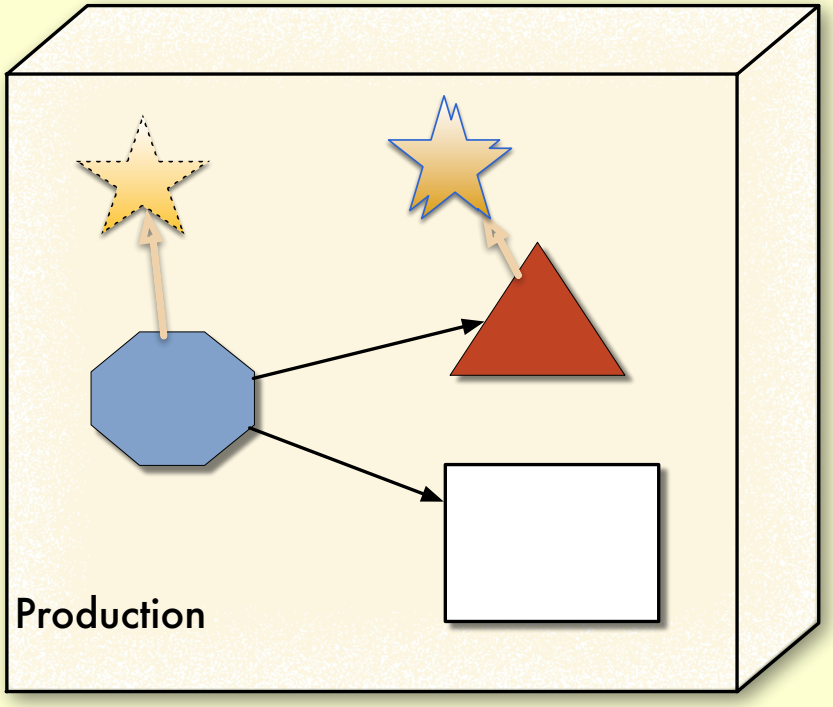
Penultimate



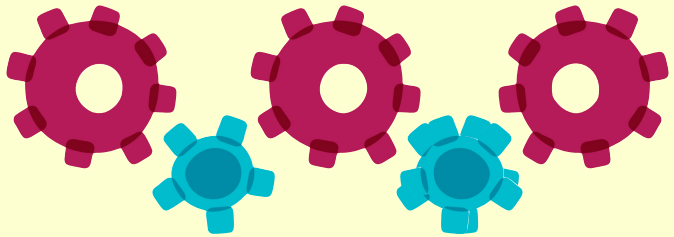


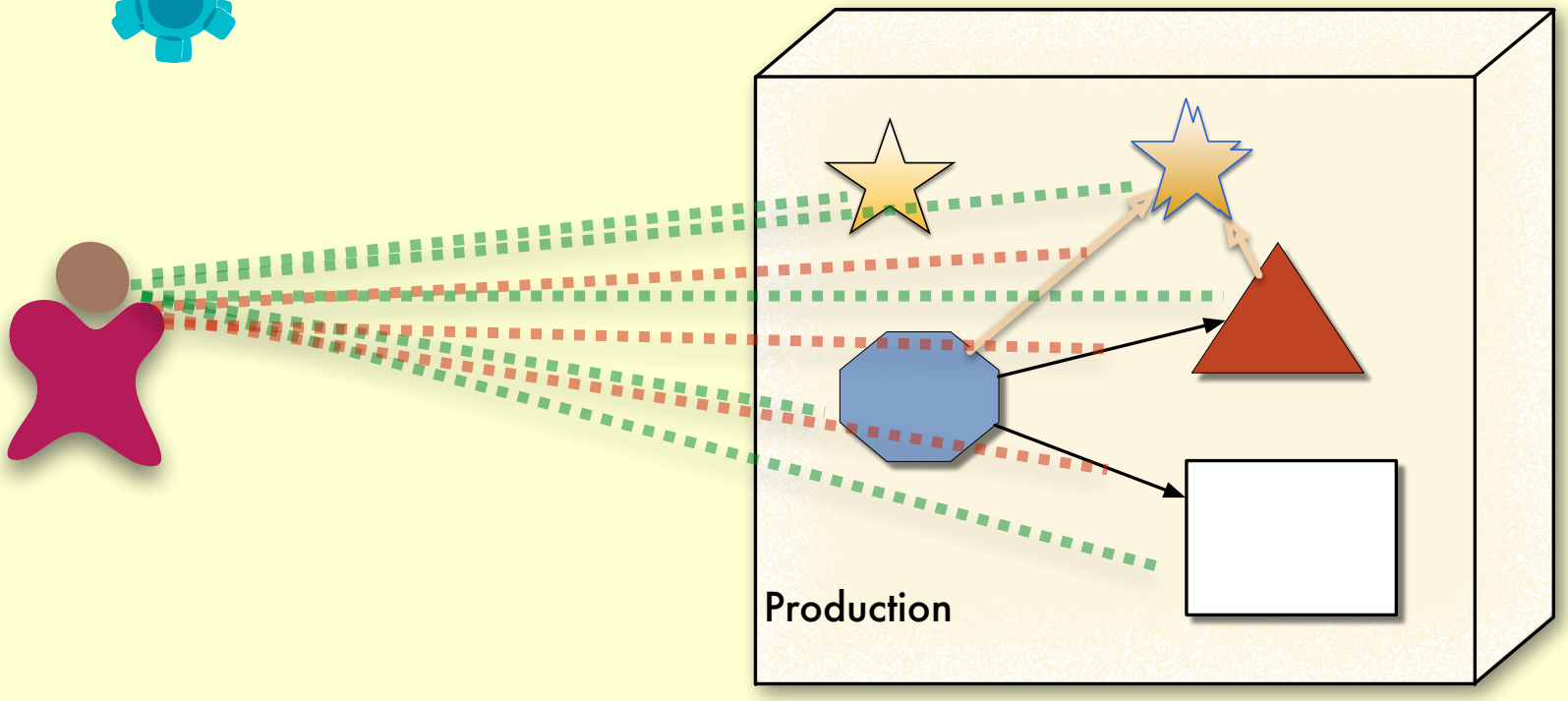
Penultimate



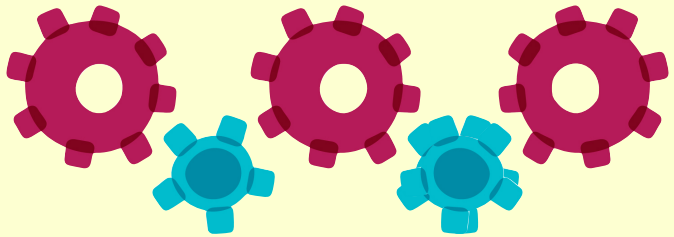


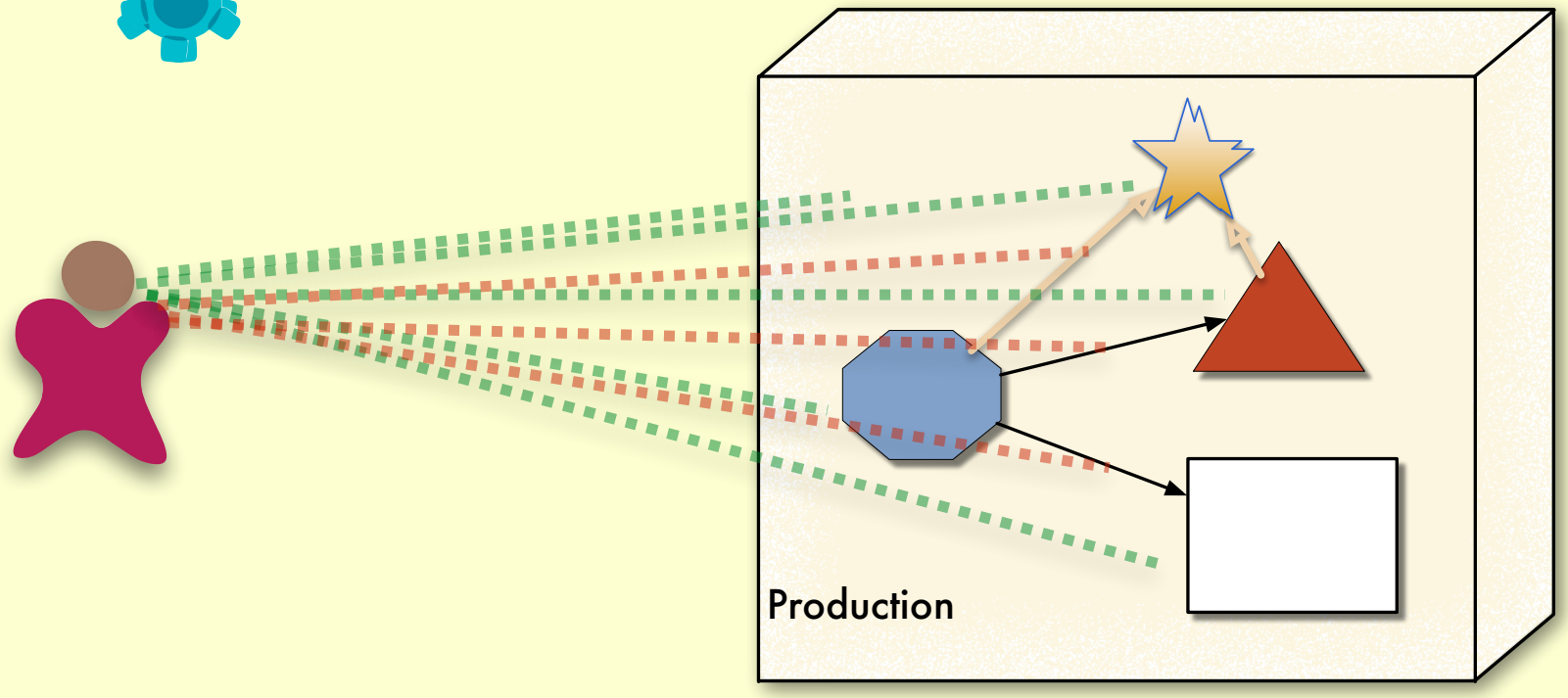
Penultimate

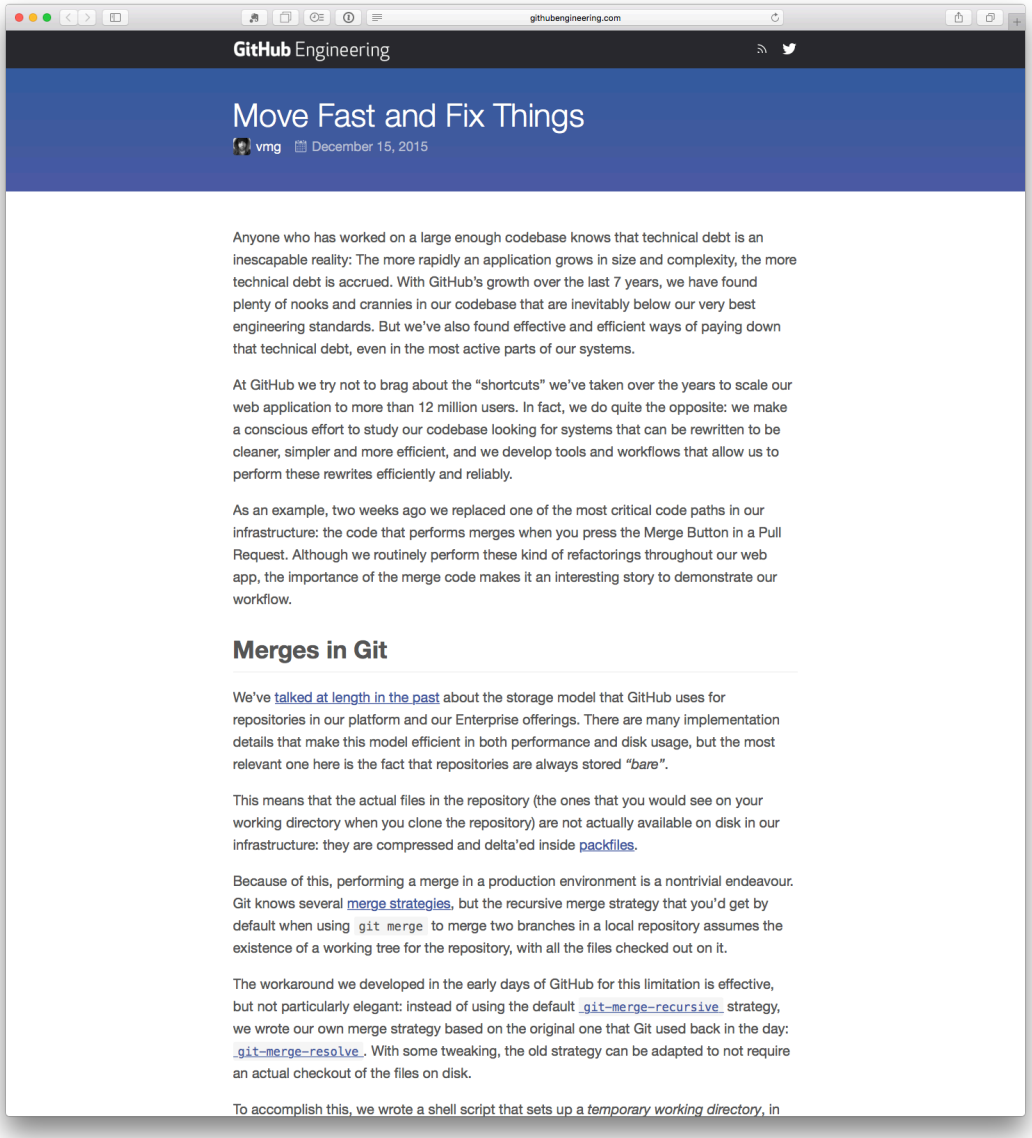




Penultimate



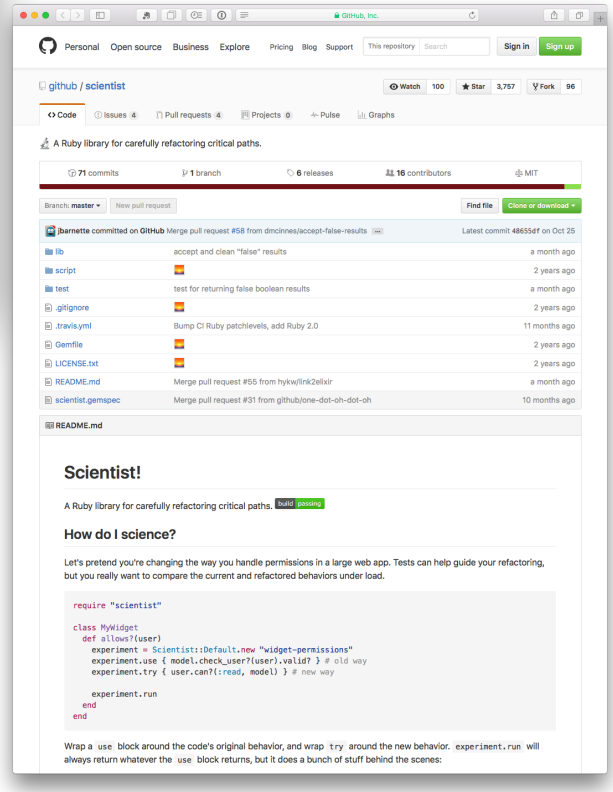




Move Fast & Fix Things

```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
```

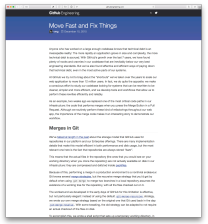
```ruby
def create_merge_commit(author, base, head, options = {})
  commit_message = options[:commit_message] || "Merge #{head} into #{base}"
  now = Time.current

  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

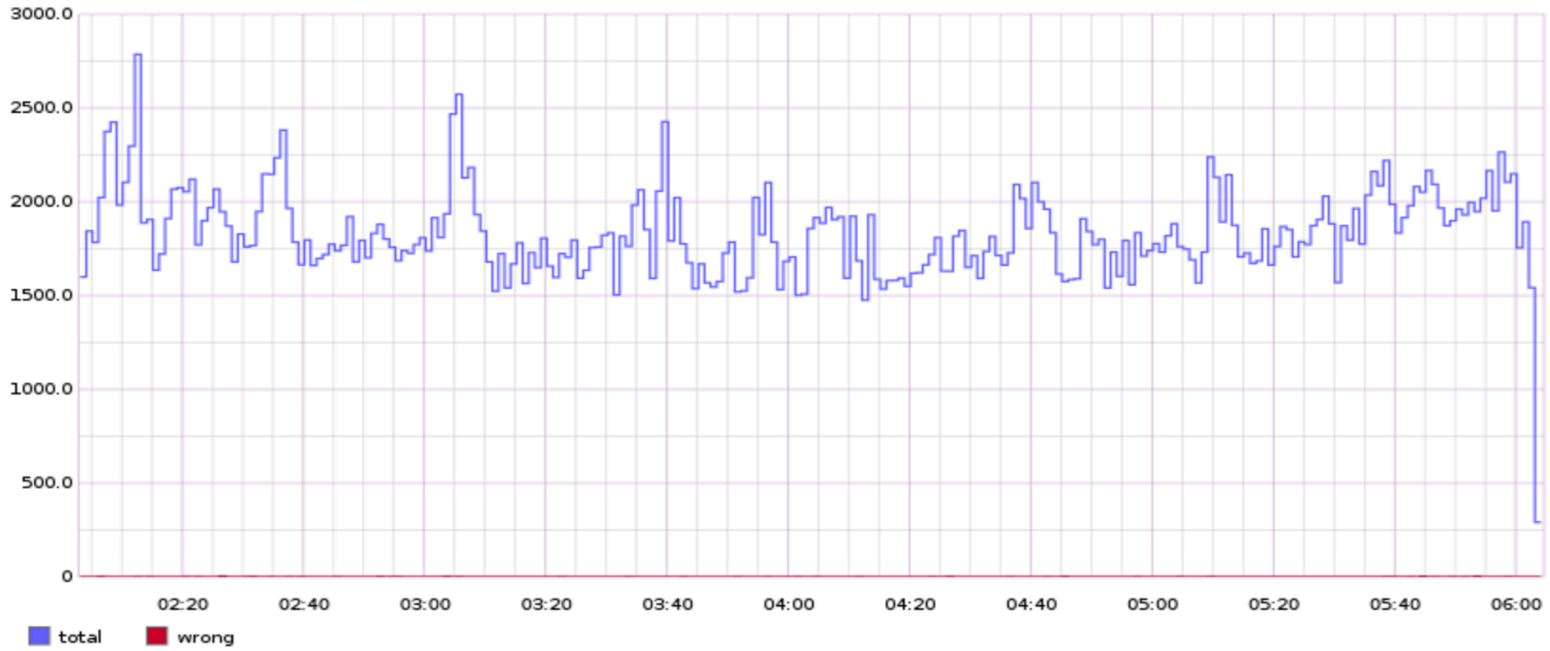

```ruby
require "scientist"

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
```

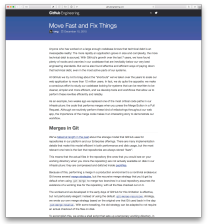
- It decides whether or not to run the try block,
- Randomizes the order in which use and try blocks are run,
- Measures the durations of all behaviors,
- Compares the result of try to the result of use,
- Swallows (but records) any exceptions raised in the try block
- Publishes all this information.



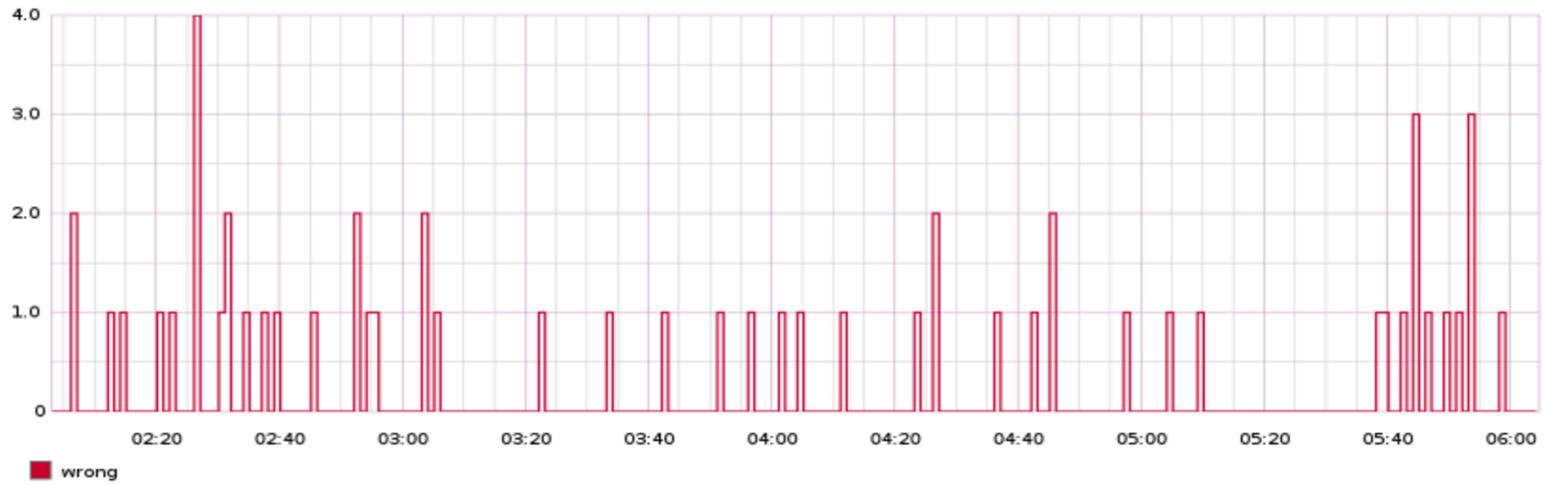
Accuracy

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

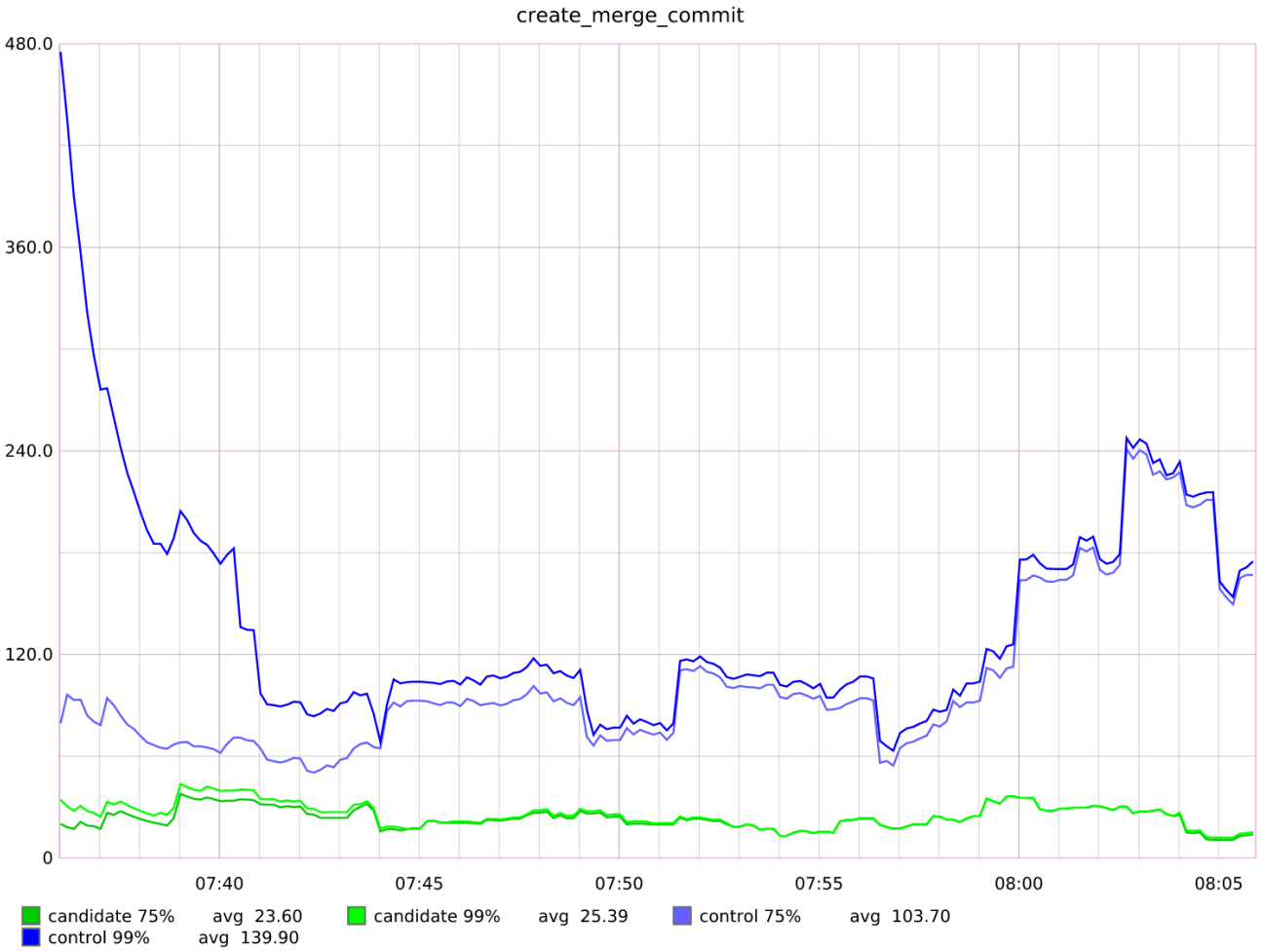




The number of incorrect/ignored only.



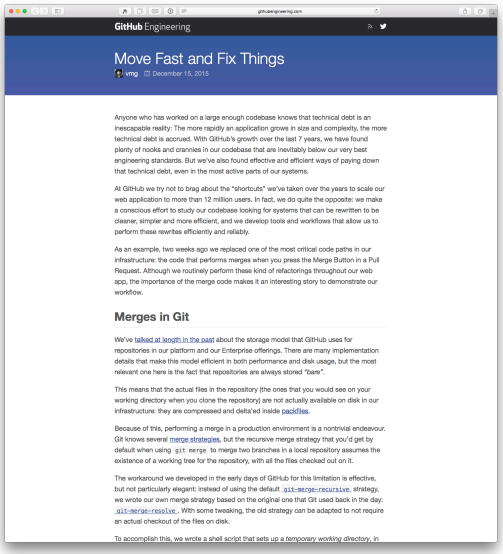

Move Fast and Fix Things


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

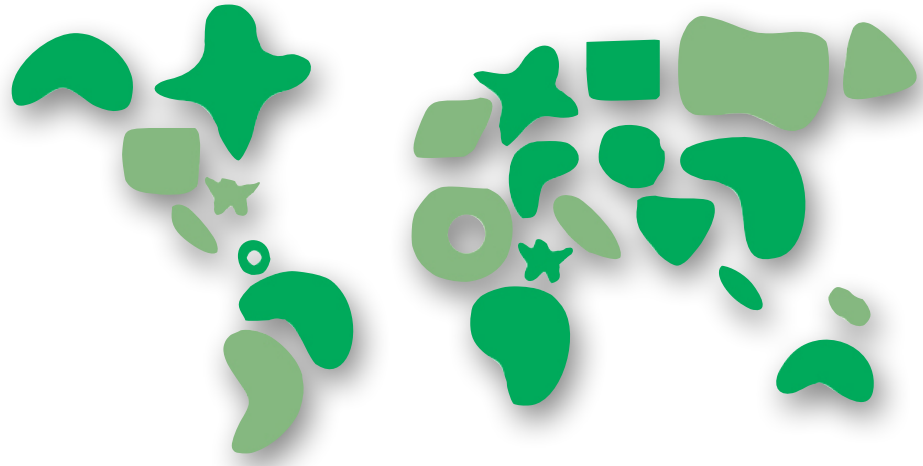
We've talked at length in the past about the storage model that GitHub uses for repositories in our platform and our Enterprise offerings. There are many implementation details that make this model efficient in both performance and disk usage, but the most relevant one here is the fact that repositories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

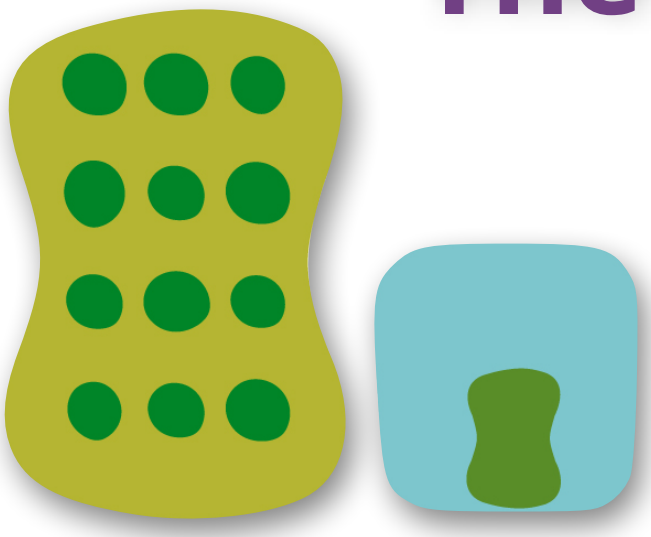
Because of this, performing a merge in a production environment is a nontrivial endeavour. Git knows several merge strategies, but the recursive merge strategy that you'd get by default when using `git merge` to merge two branches in a local repository assumes the existence of a working tree for the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

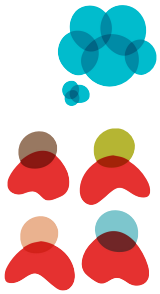
The workaround we developed in the early days of GitHub for this limitation is effective, but not particularly elegant: instead of using the default `git-merge-recursive` strategy, we wrote our own merge strategy based on the original one that Git used back in the day: `git-merge-resolve`. With some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

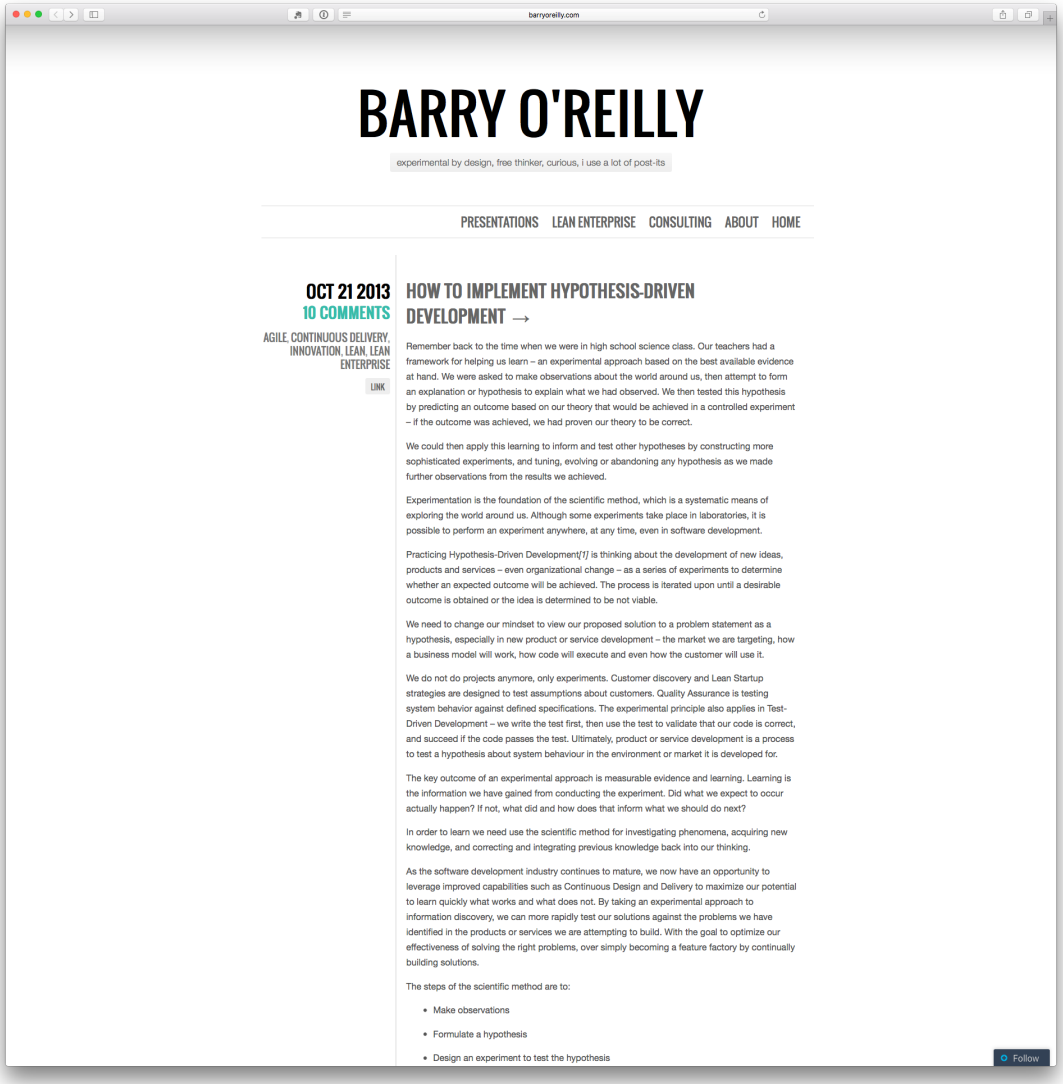
To accomplish this, we wrote a shell script that sets up a *temporary working directory*, in



The Business Case







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

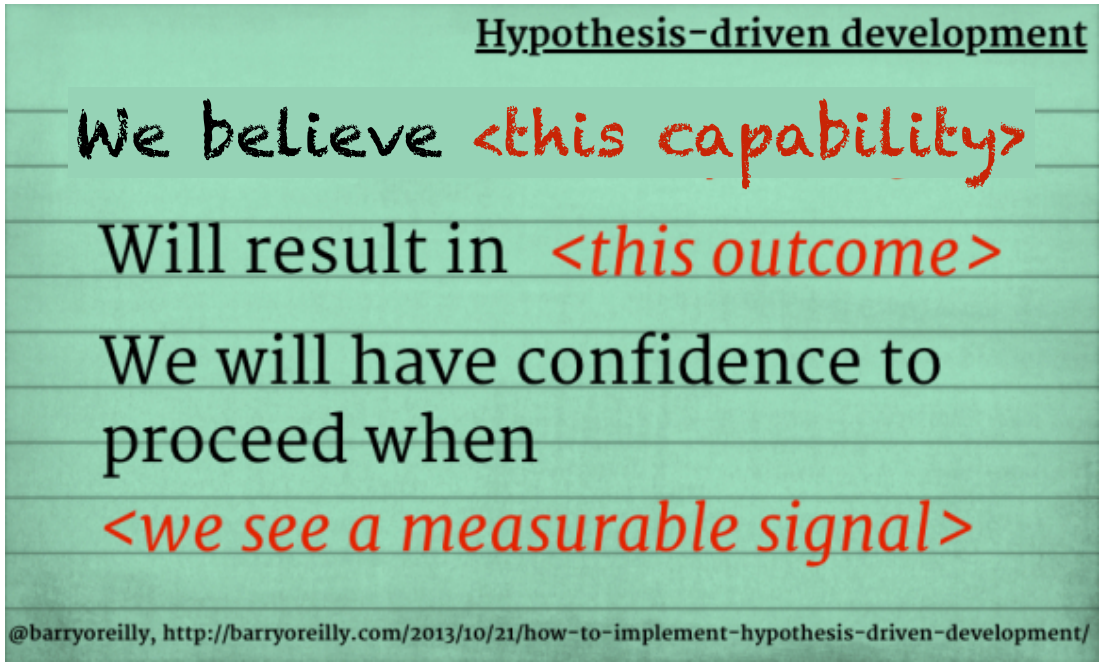
Requirements

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

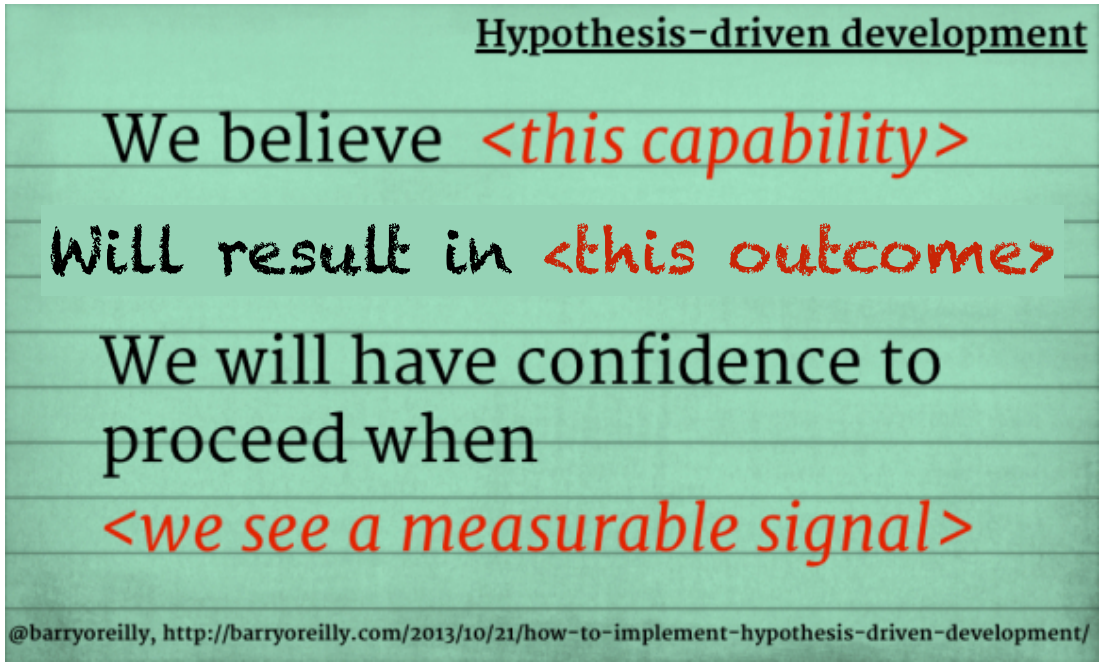
Behavior-Driven Development

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

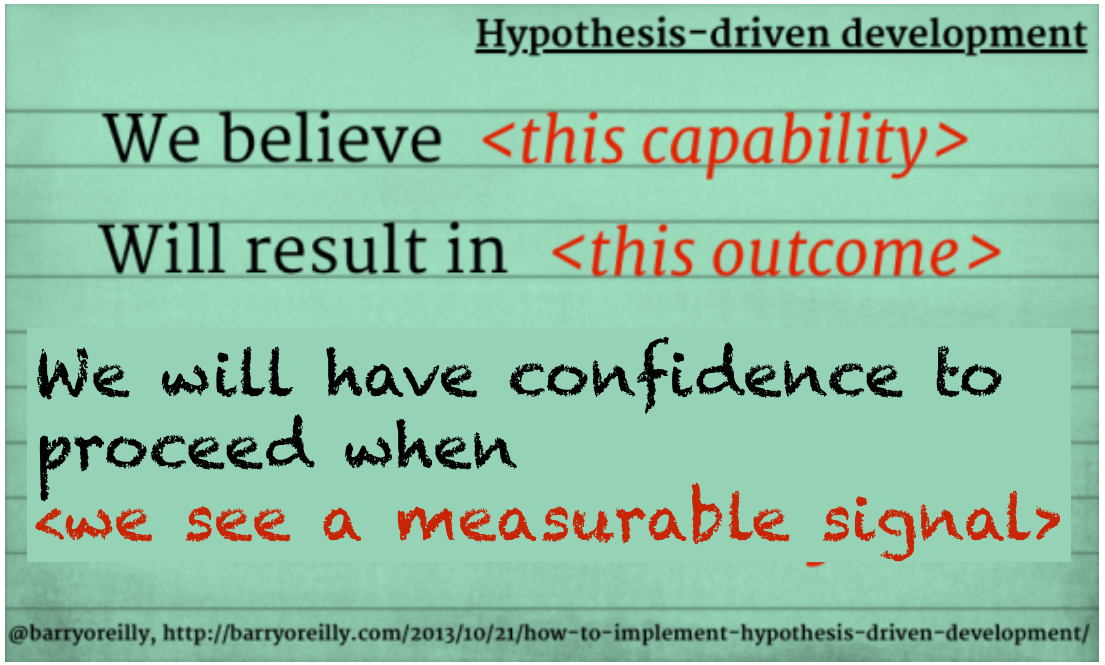
Hypothesis-driven Development



Hypothesis-driven Development



Hypothesis-driven Development



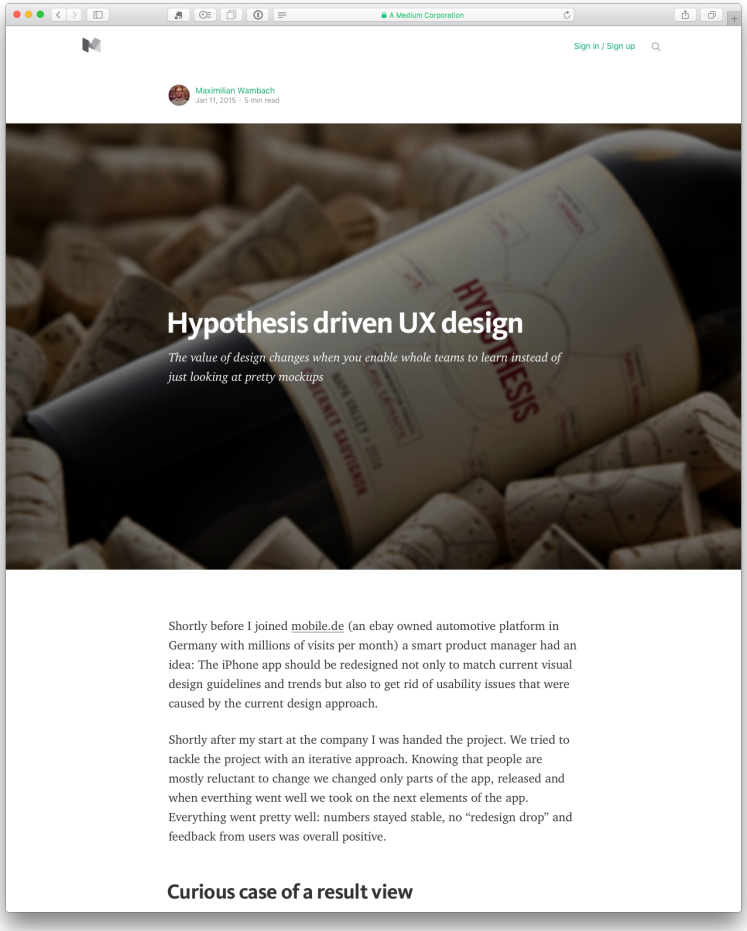
Business Story

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



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

Hypothesis Driven UX

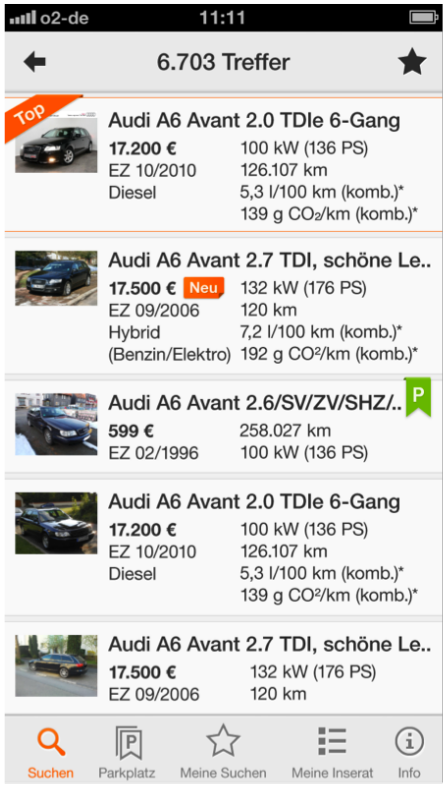
Three Hypotheses

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

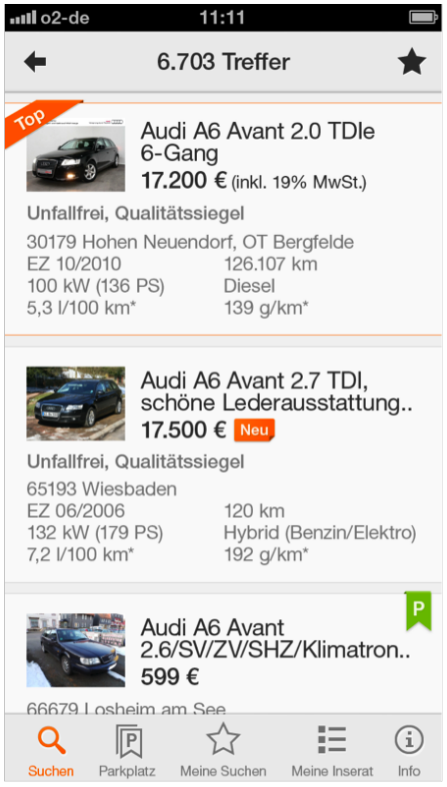
Better Structure
If we provide more structure to our listings then we achieve a better scanability because the user is able to scan the relevant information quicker

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

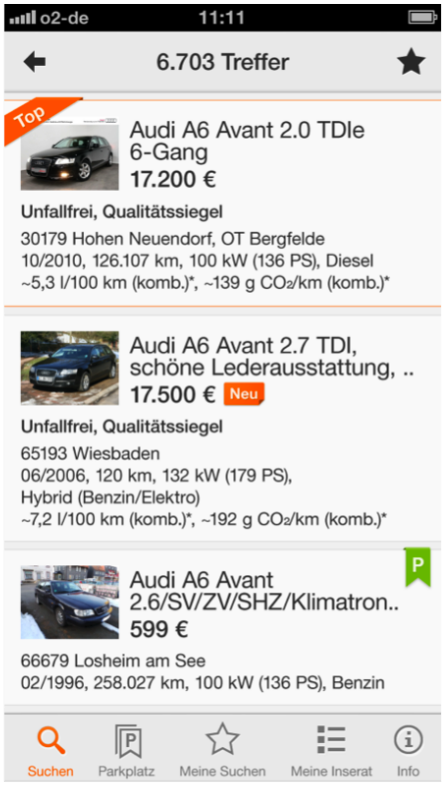
Experiments to Perform



More Listings



Better Structure



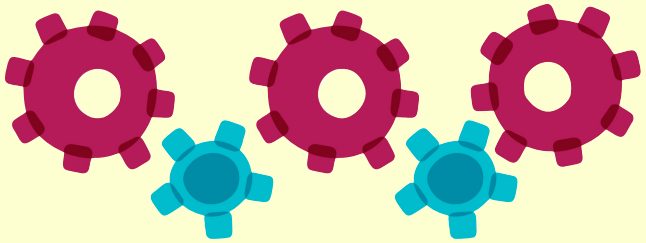
Better Prioritization

Evolutionary Architecture

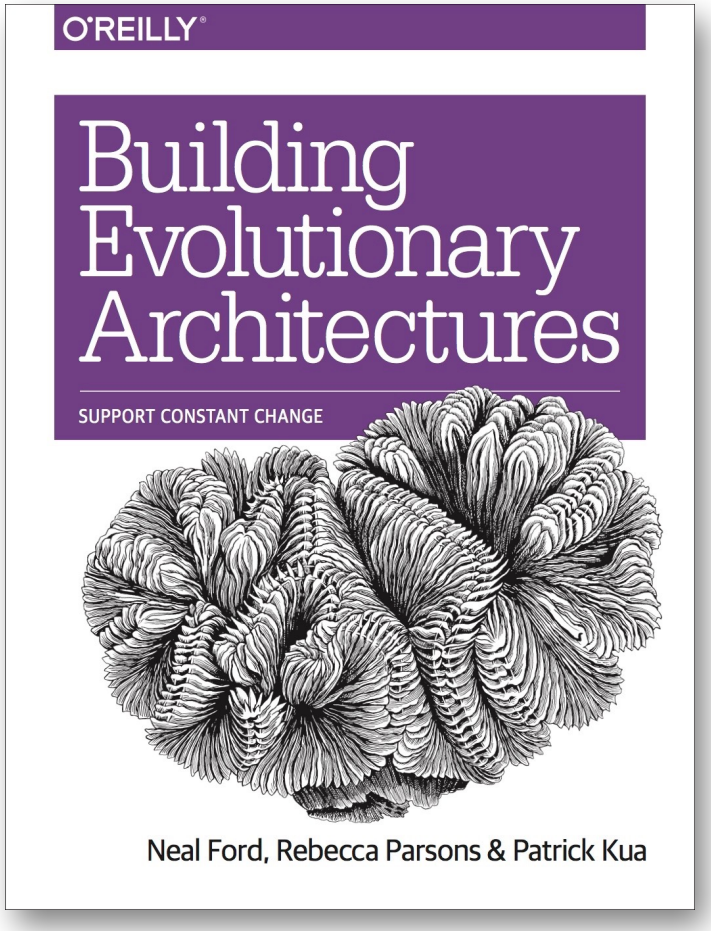
An evolutionary architecture supports
guided,
incremental change
across multiple dimensions.

Agenda Part 2

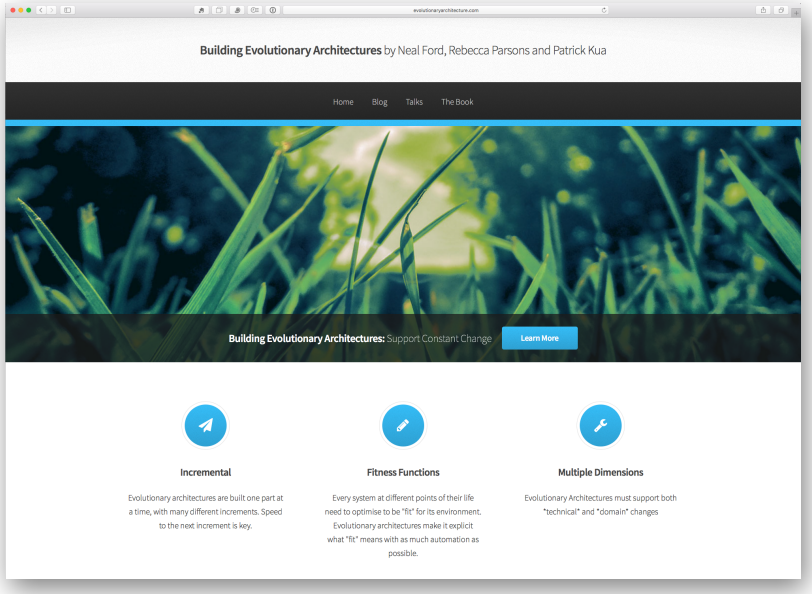
The Evolution of Penultimate



bUiLdiNG eVoLuTiONaRy ARcHitECtuREs



For more information:



http://evolutionaryarchitecture.com