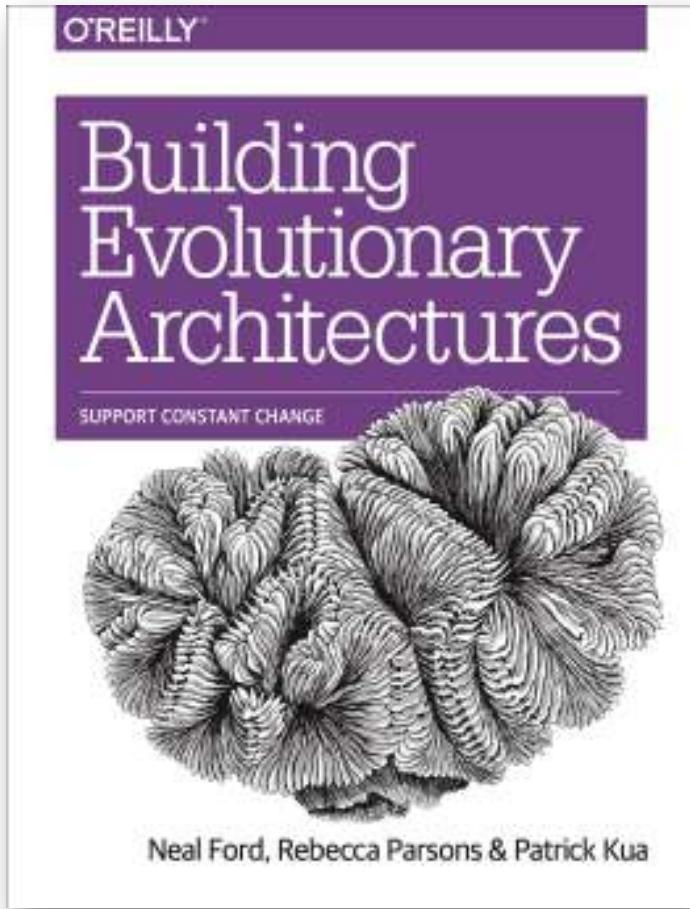


Building Evolutionary Architectures

SUPPORT CONSTANT CHANGE



 @neal4d
nealford.com



 @rebeccaparsons



 @patkua



Agenda

definition

incremental change

fitness functions

evolutionary architectures

fitness function katas

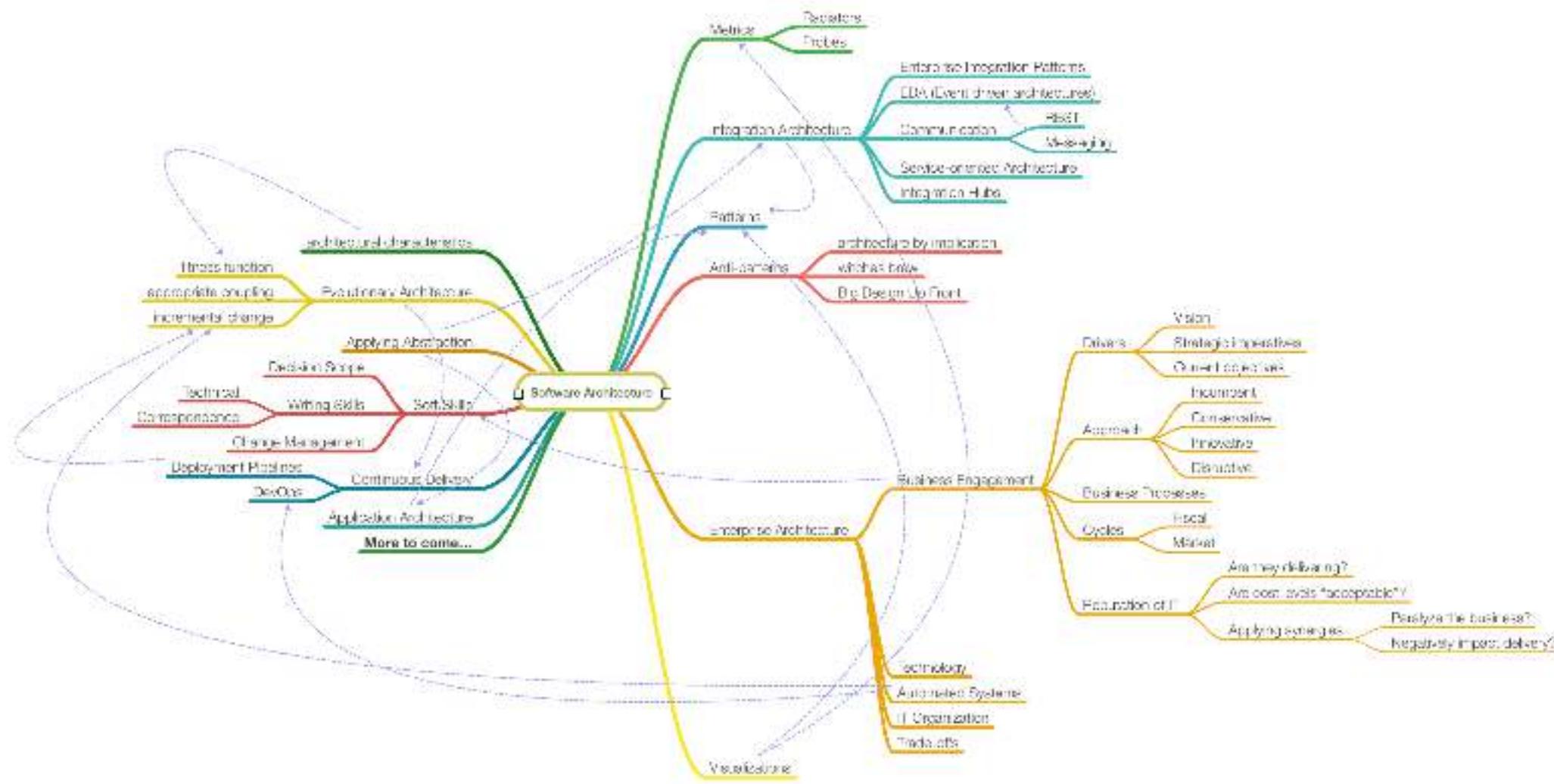
evolutionary data

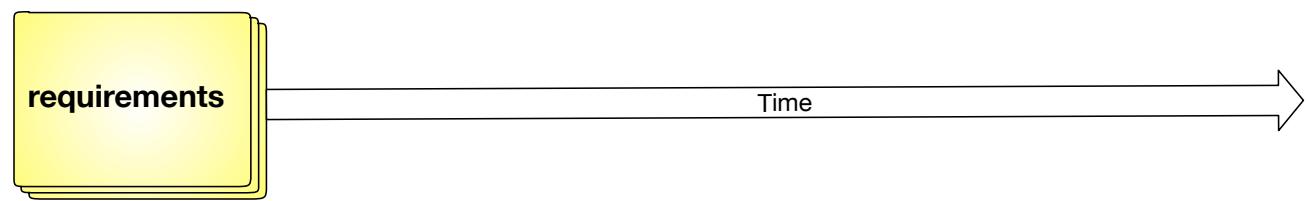
implementing

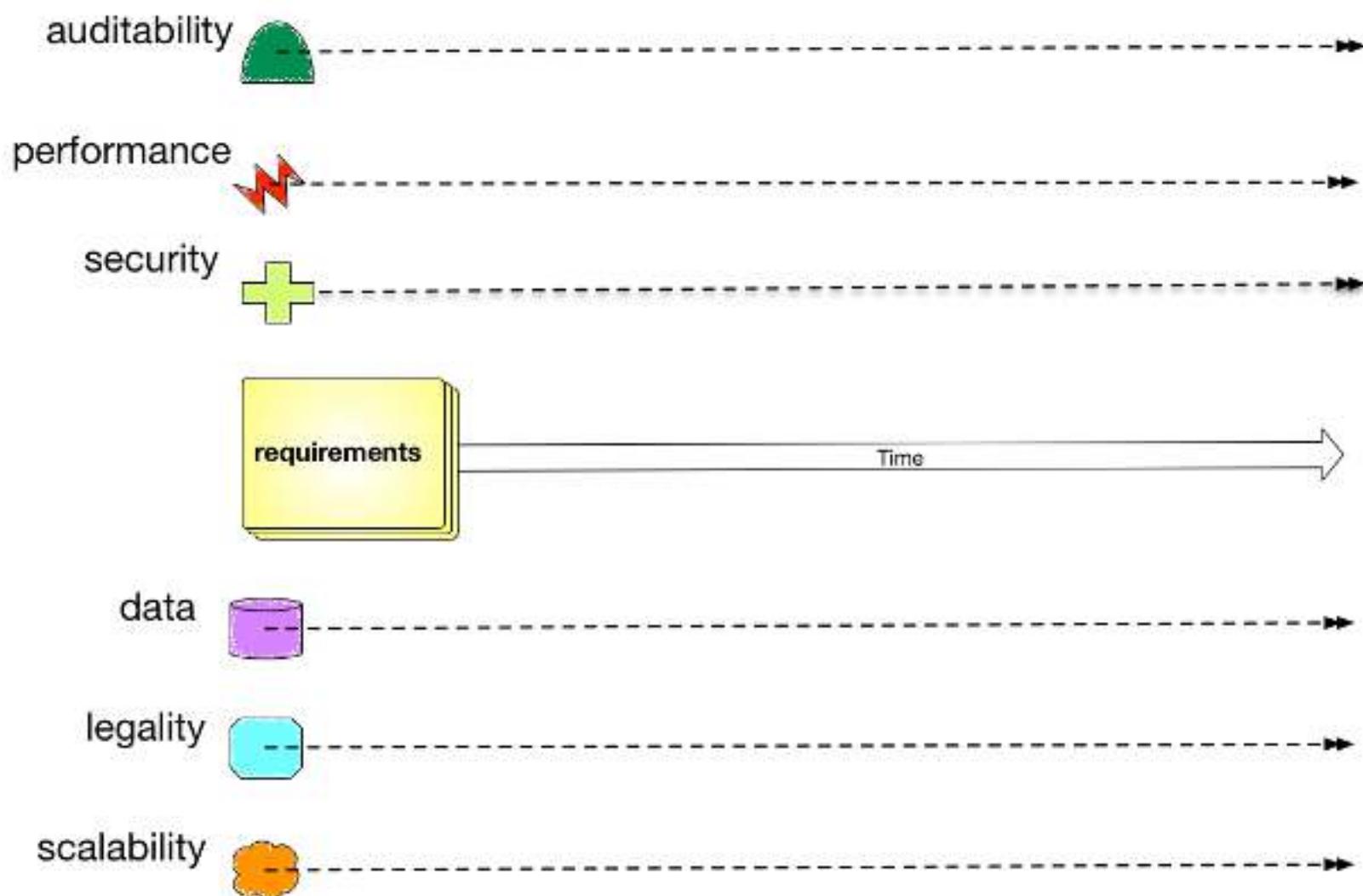
Day 1

Day 2

What is Software Architecture?





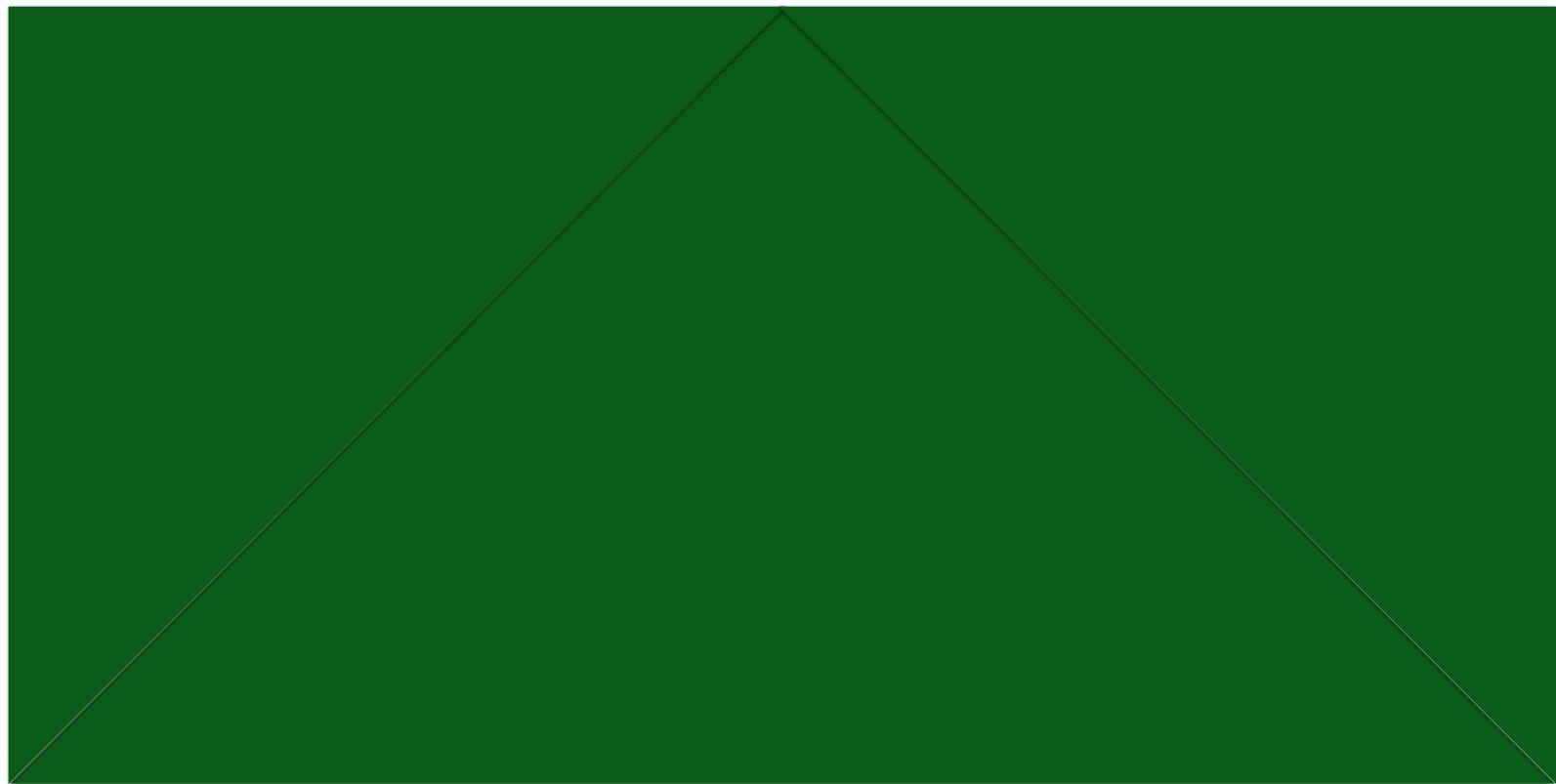




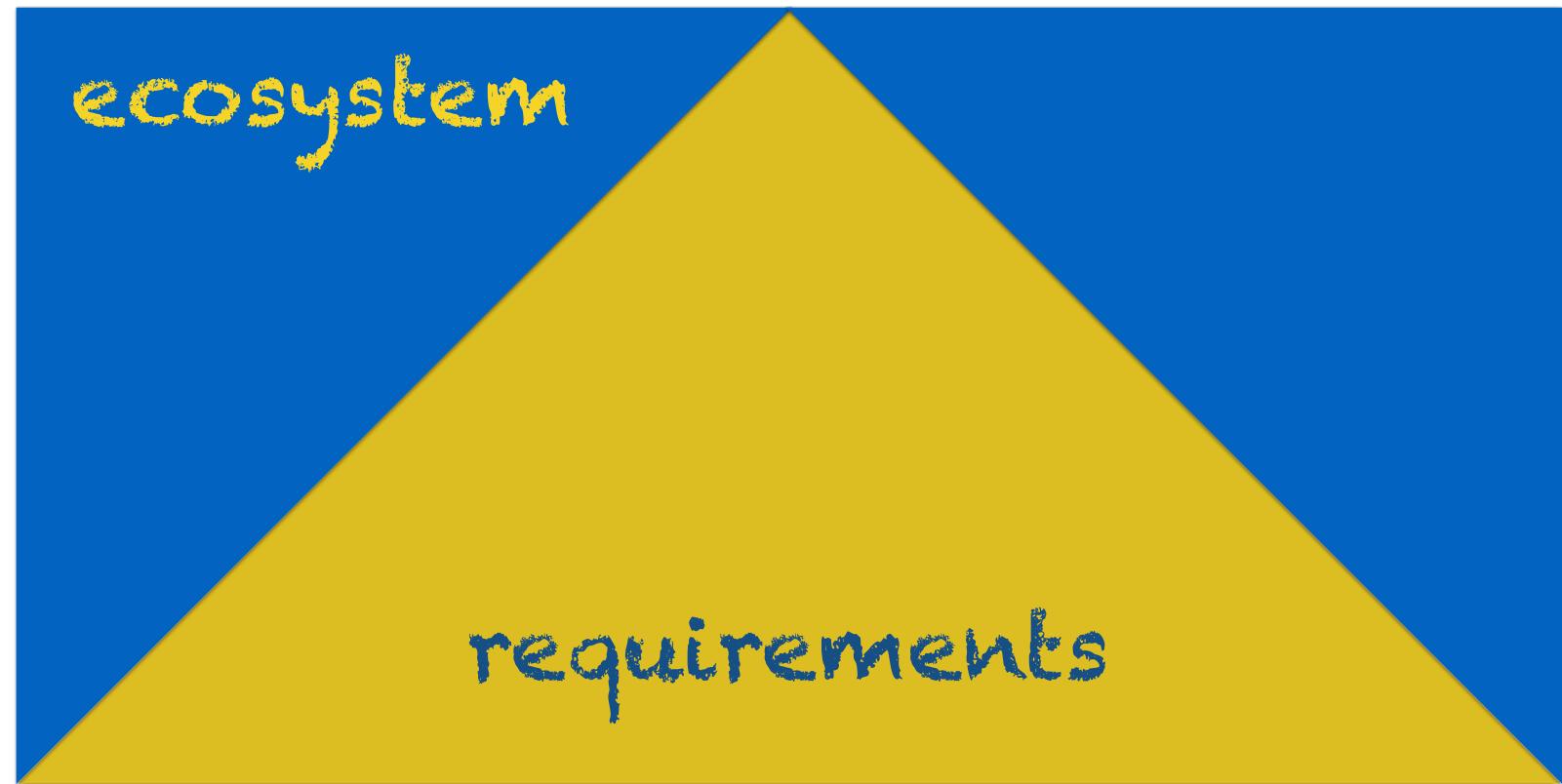
accessibility	reliability	repeatability
accountability	extensibility	reproducibility
accuracy	failure transparency	resilience
adaptability	fault-tolerance	responsiveness
administrability	fidelity	reusability
affordability	flexibility	robustness
agility	inspectability	safety
auditability	installability	scalability
autonomy	integrity	seamlessness
availability	interchangeability	self-sustainability
compatibility	interoperability	serviceability
composability	learnability	supportability
configurability	maintainability	securability
correctness	manageability	simplicity
credibility	mobility	stability
customizability	modifiability	standards compliance
debugability	modularity	survivability
degradability	operability	sustainability
determinability	orthogonality	tailorability
demonstrability	portability	testability
dependability	precision	timeliness
deployability	predictability	traceability
discoverability	process capabilities	transparency
distributability	productivity	ubiquity
durability	provability	understandability
effectiveness	recoverability	upgradability
efficiency	relevance	usability

evolvability

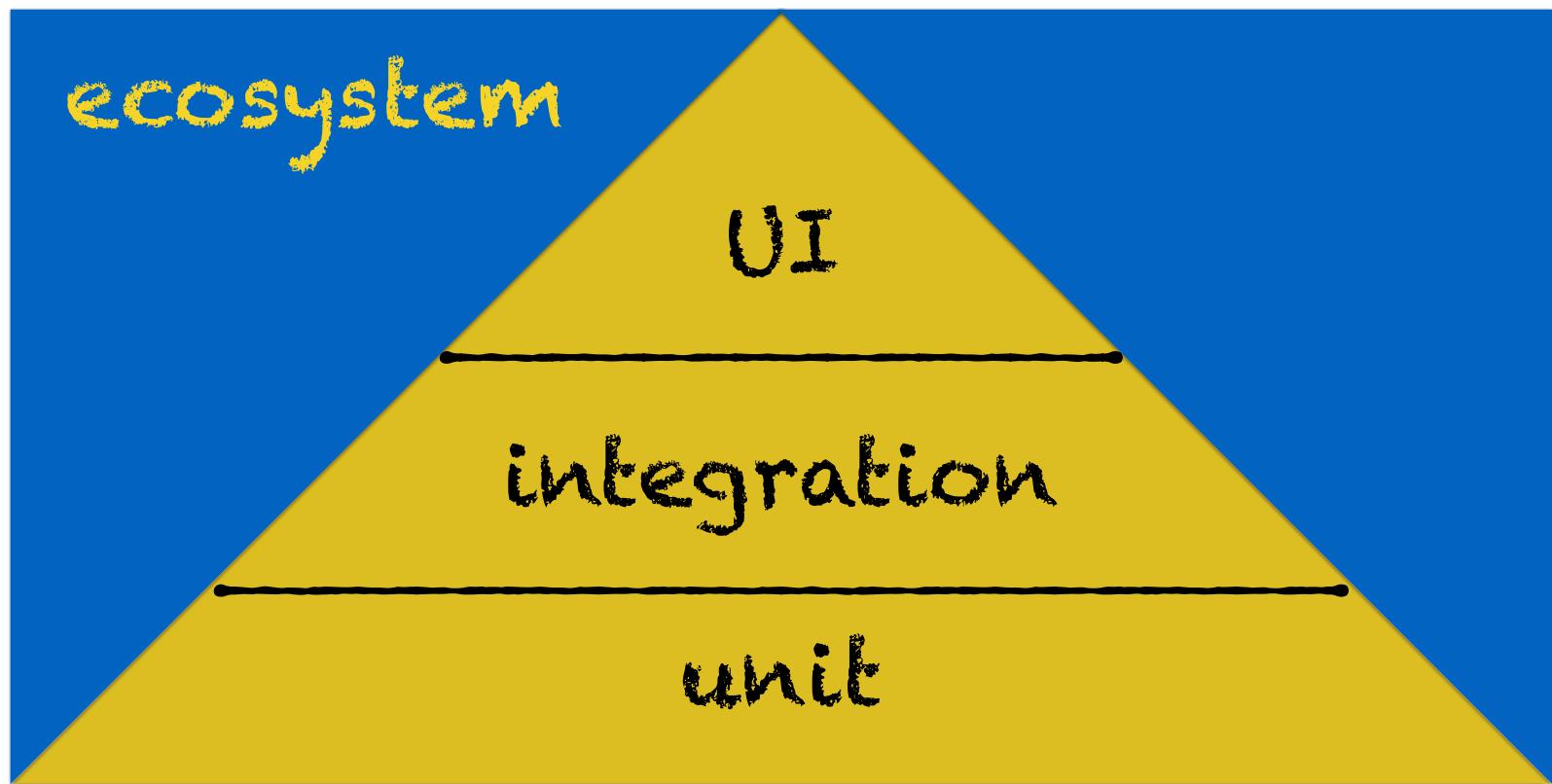
Change



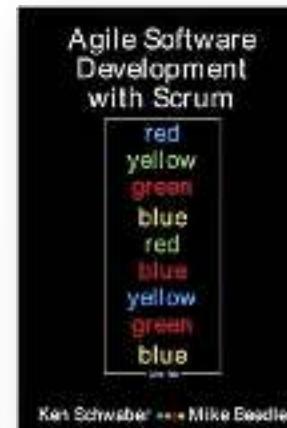
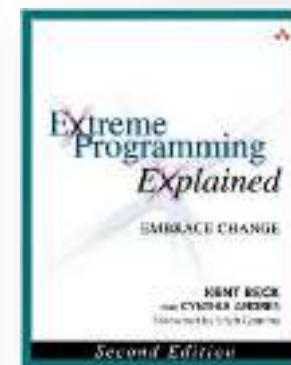
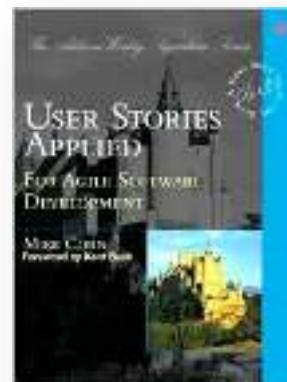
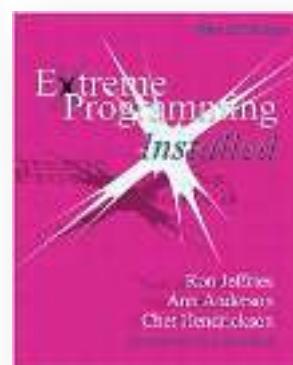
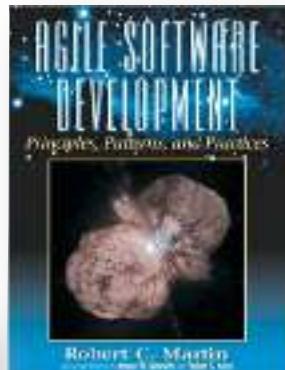
Change



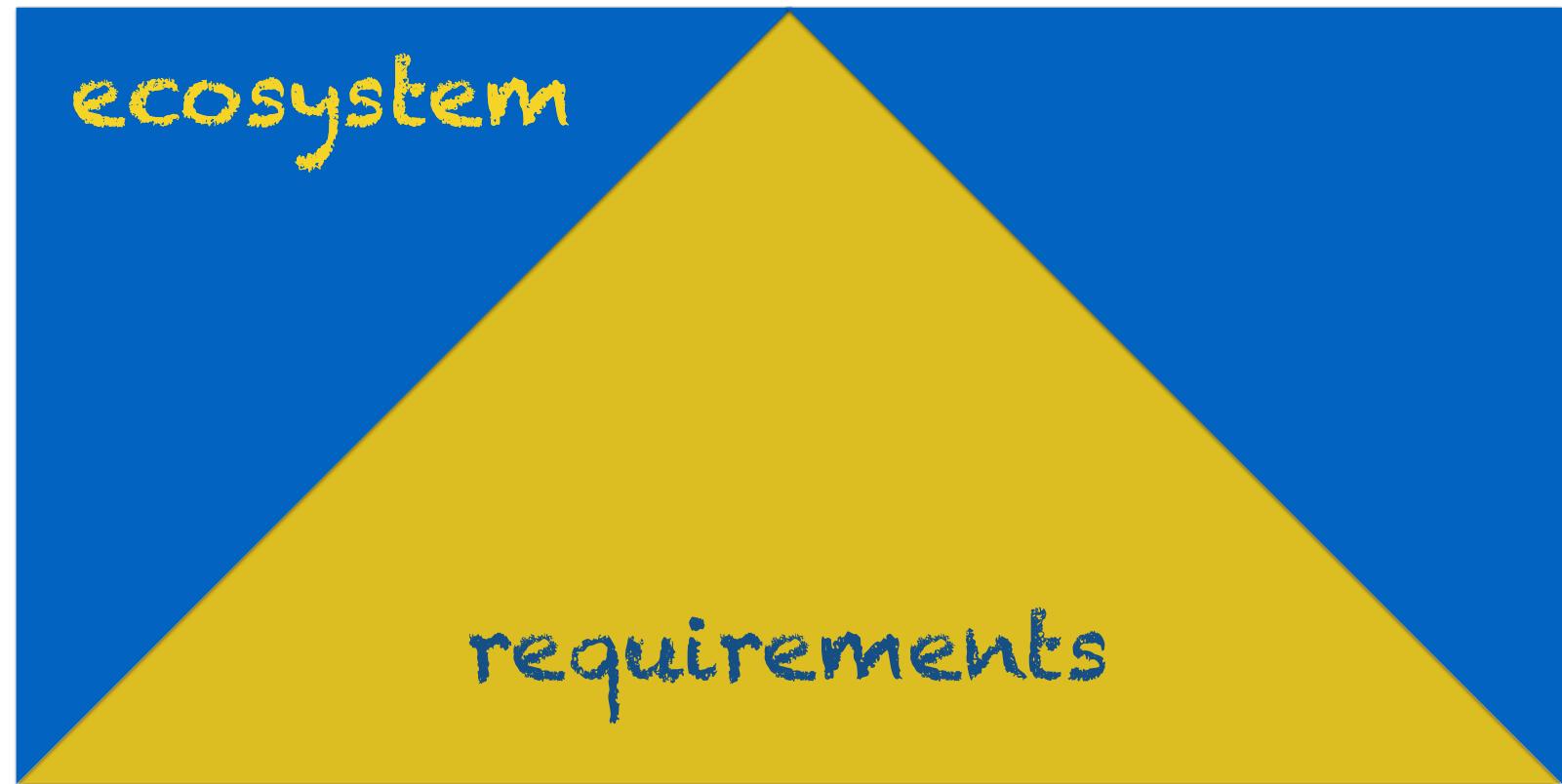
Change



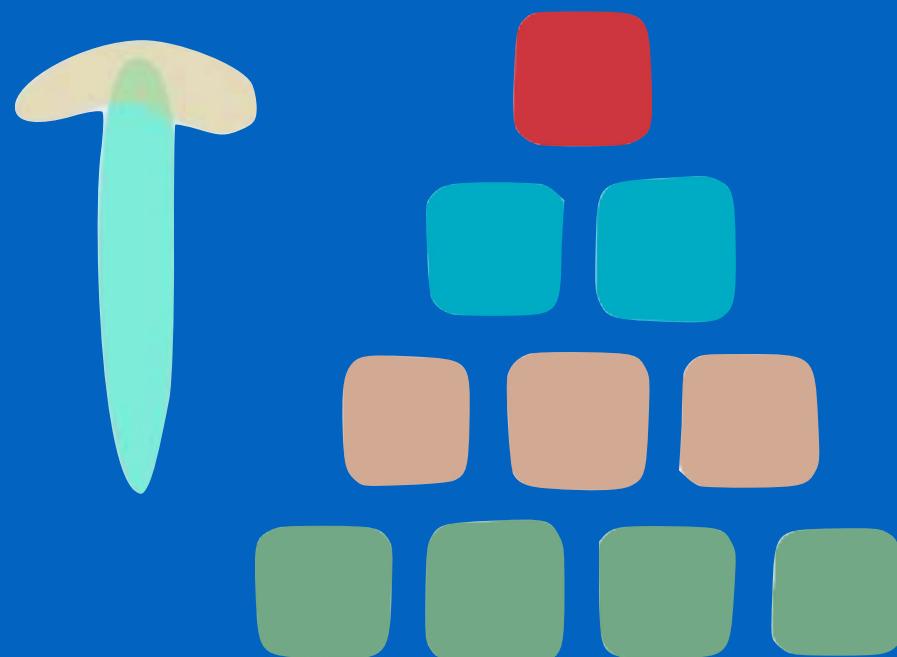
Business Change



Change



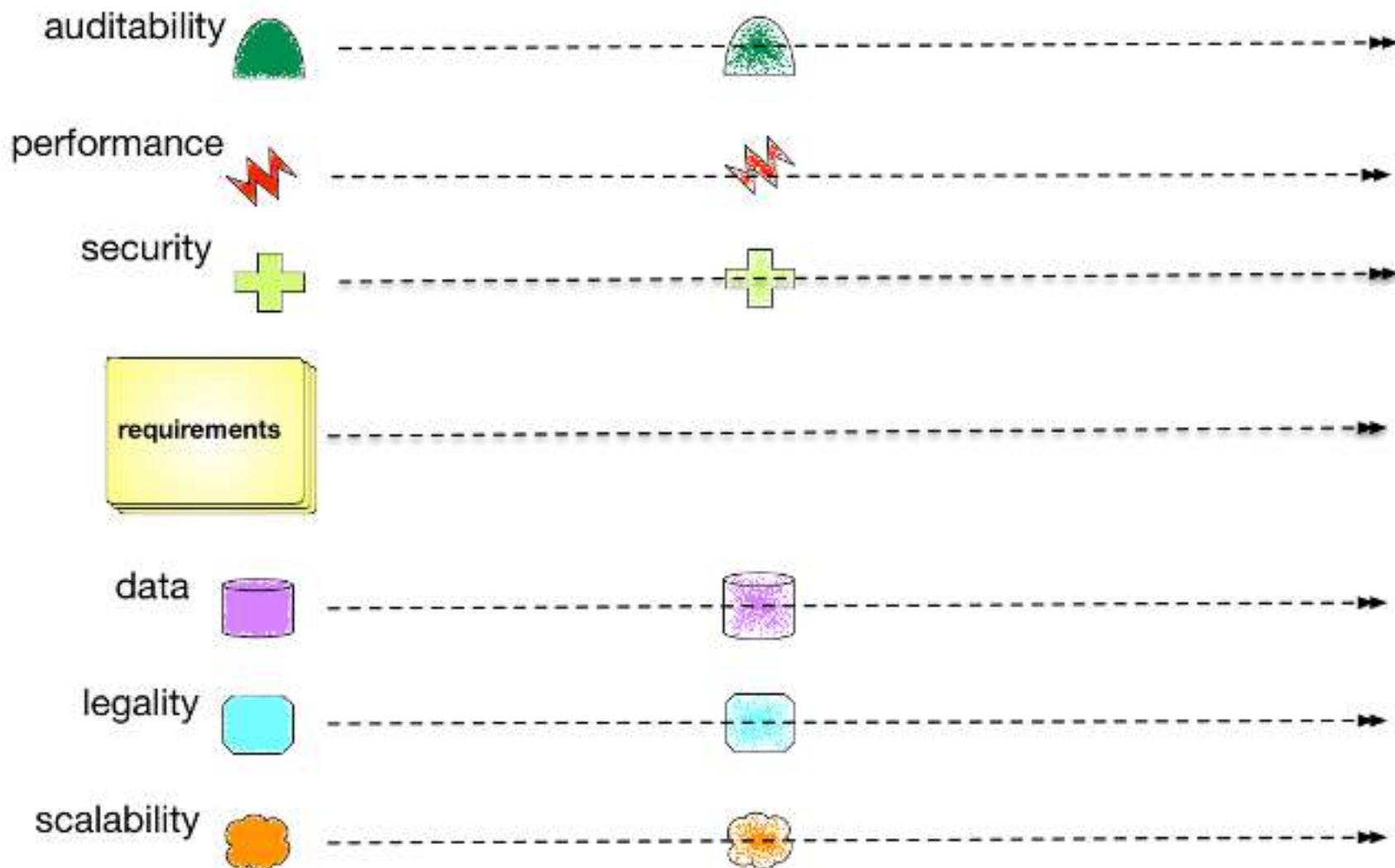
Dynamic Equilibrium





Dynamic Equilibrium

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



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

everything changes
all the time!

agile

continual

emergent

Why evolutionary?

reactive

incremental

Why evolutionary?

adaptable?

Evolutionary Architecture

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



Evolutionary Architecture

An evolutionary architecture supports
guided
incremental change across multiple dimensions.





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.

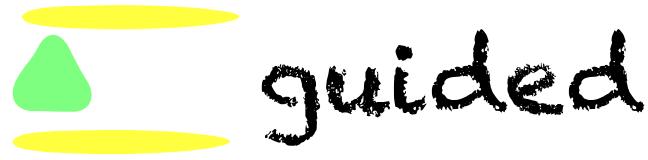


guided





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.

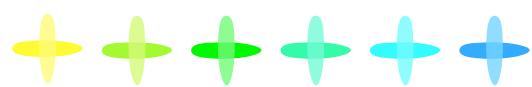


An architectural fitness function provides an objective integrity assessment of some architectural characteristic(s).

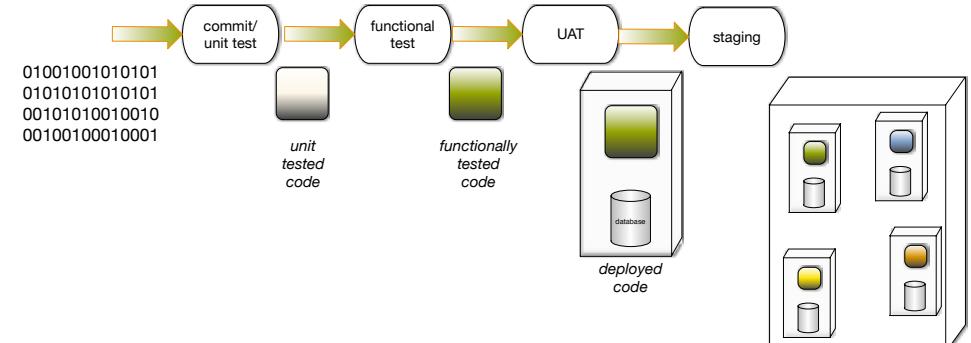
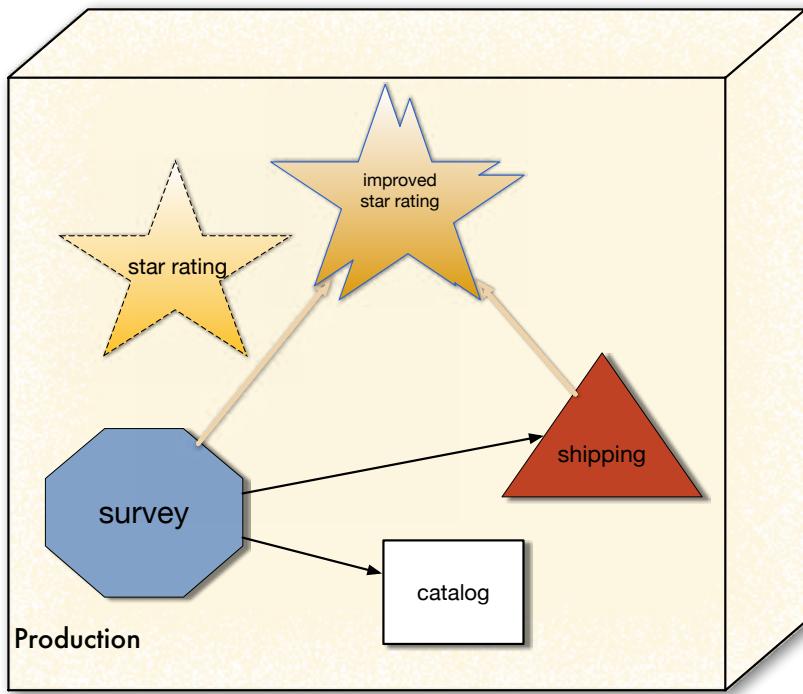
Evolutionary Architecture

An evolutionary architecture supports
guided
incremental change
across multiple dimensions.





incremental

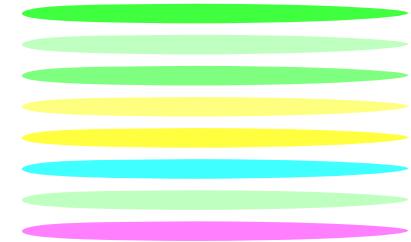


Evolutionary Architecture

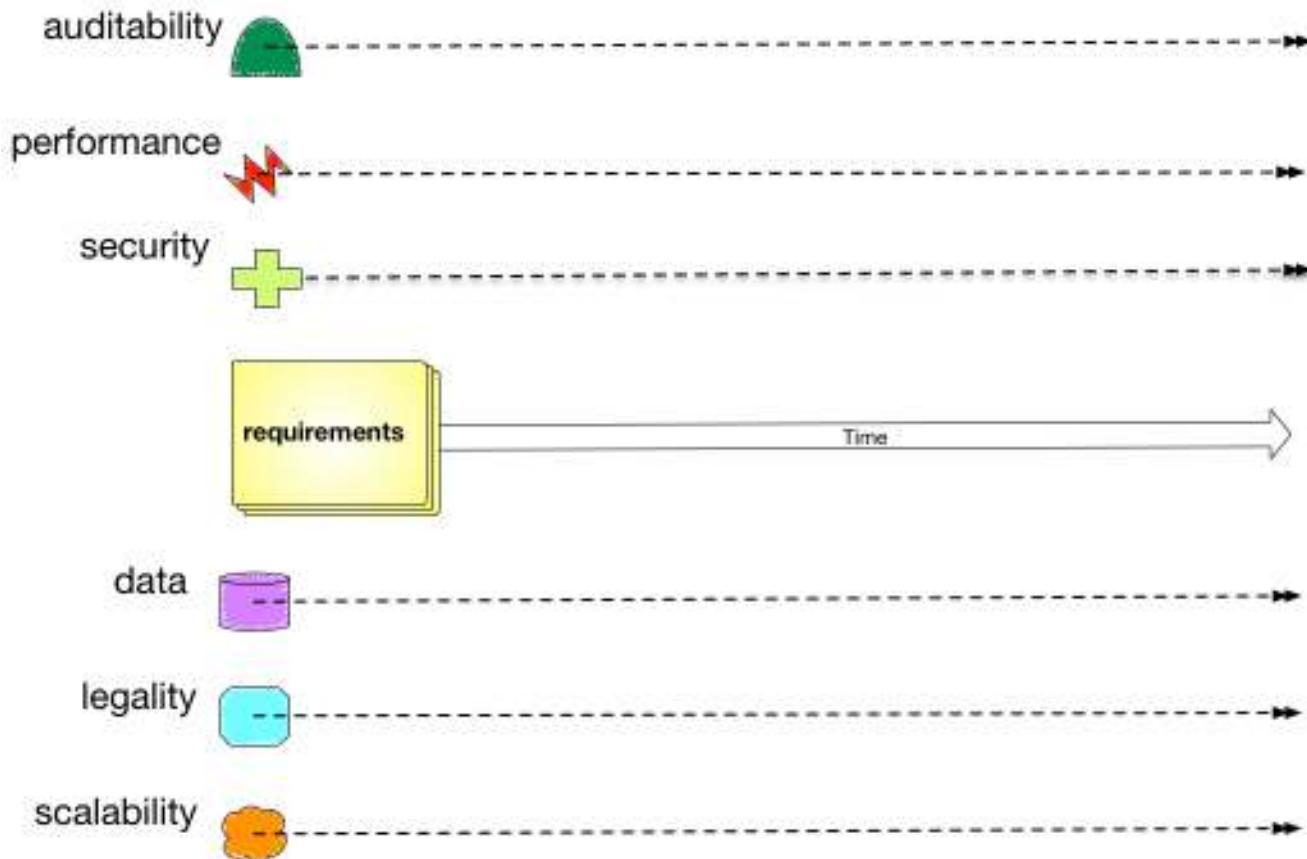
An evolutionary architecture supports
guided,

incremental change + + + + +
across multiple dimensions





multiple dimensions



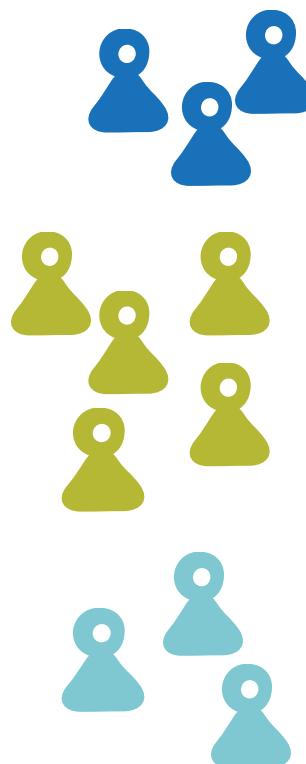
Evolutionary Architecture

An evolutionary architecture supports
guided,
incremental change

across **multiple dimensions**



Incidentally Coupled Teams



user interface

server-side

DBA

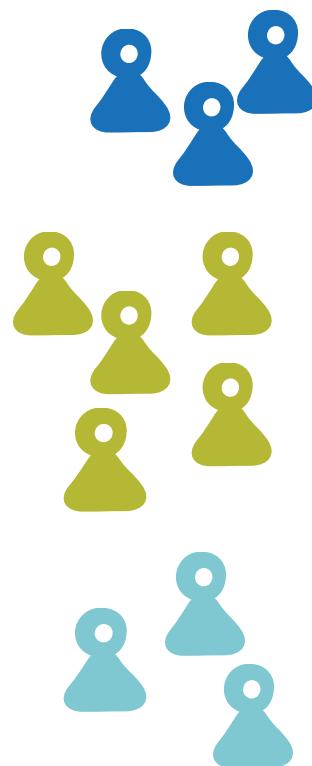
Conway's Law

“organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations”

Melvin Conway, 1968

en.wikipedia.org/wiki/Conway%27s_law

Incidentally Coupled Teams



user interface

server-side

DBA

Autonomous Teams



Orders



Shipping

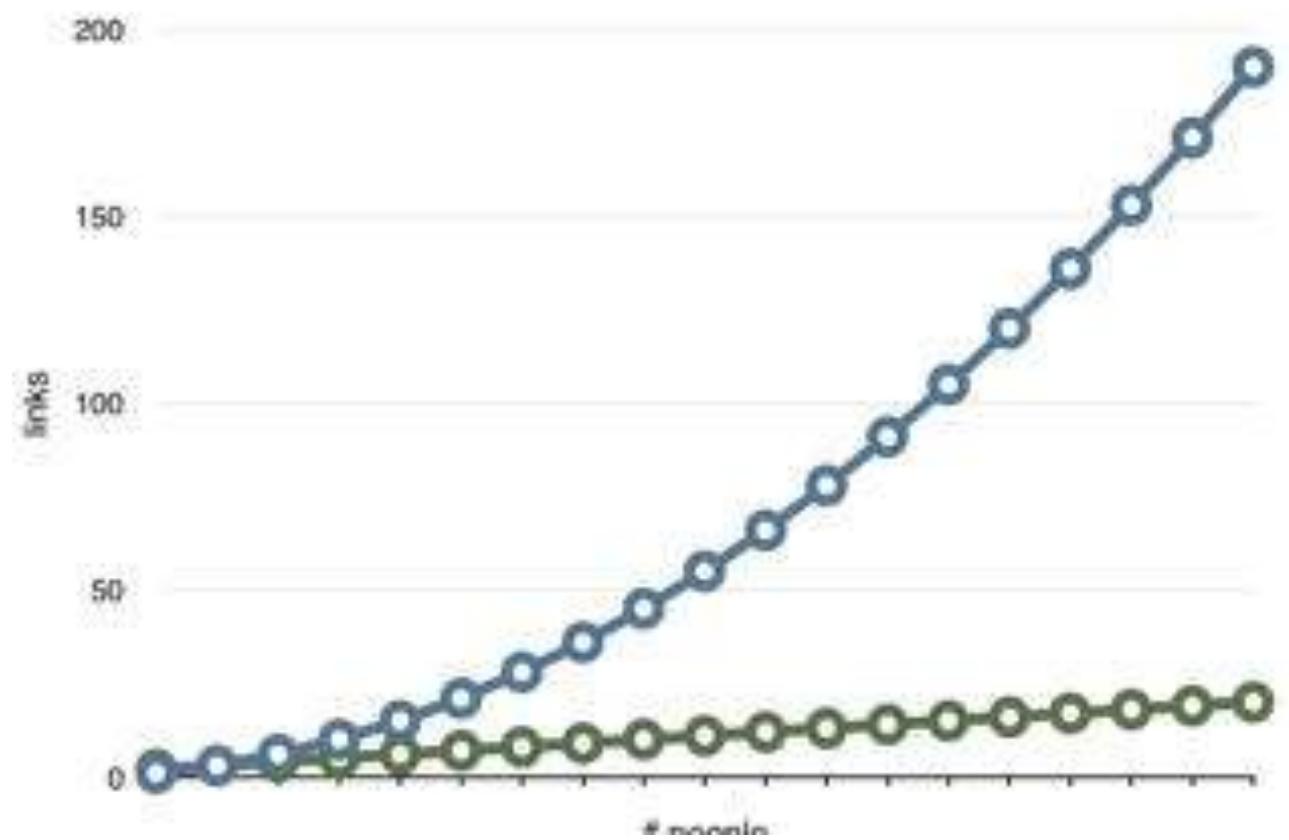
Inverse Conway Maneuver

Catalog

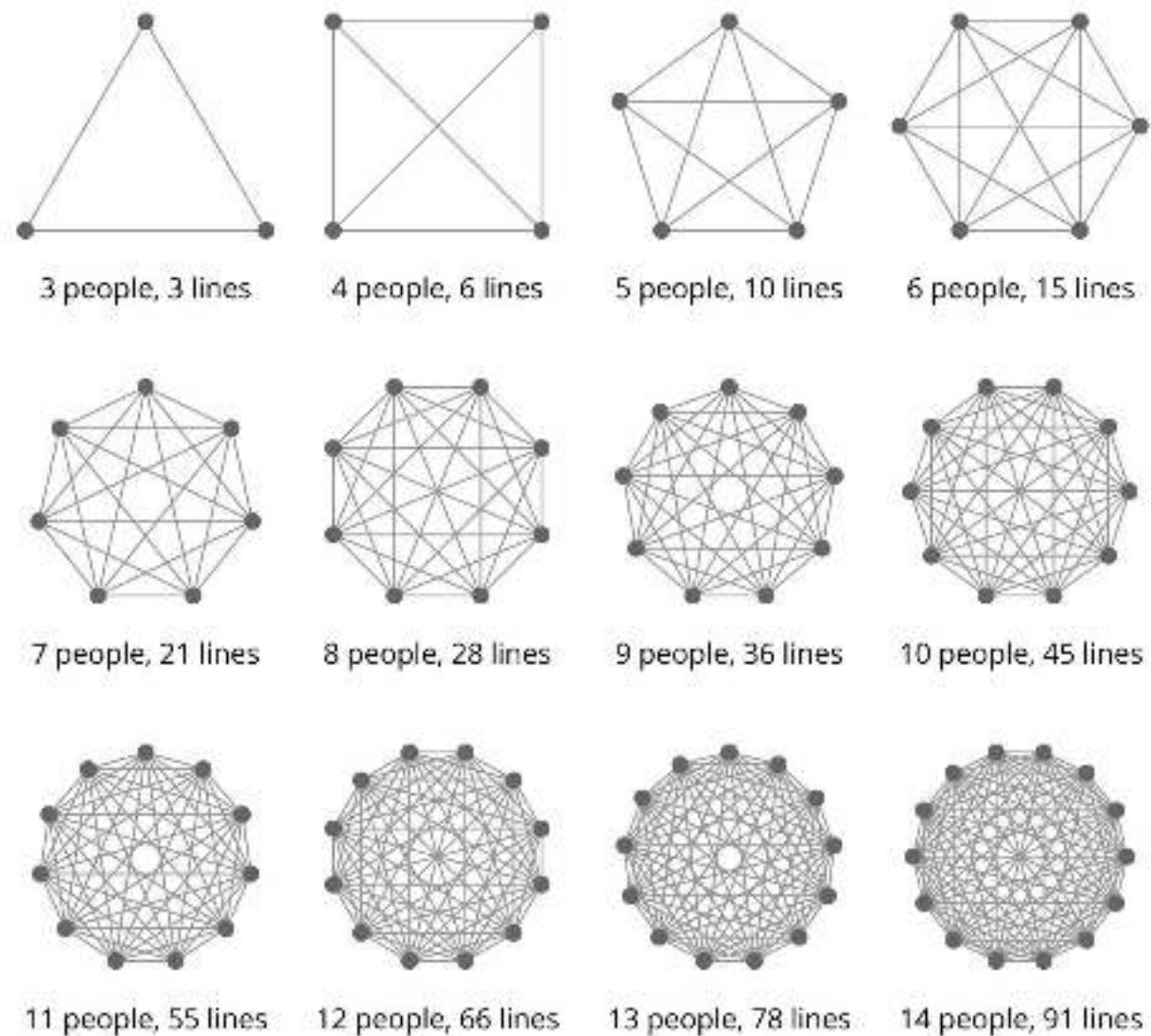


Low Efferent Coupling between Teams

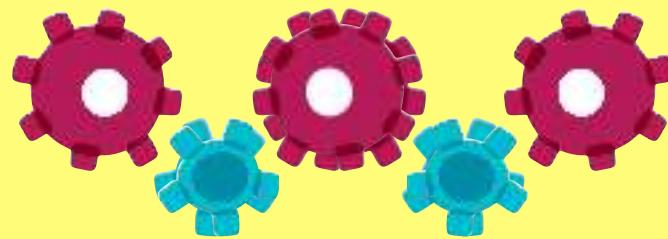
$$\frac{n(n-1)}{2}$$

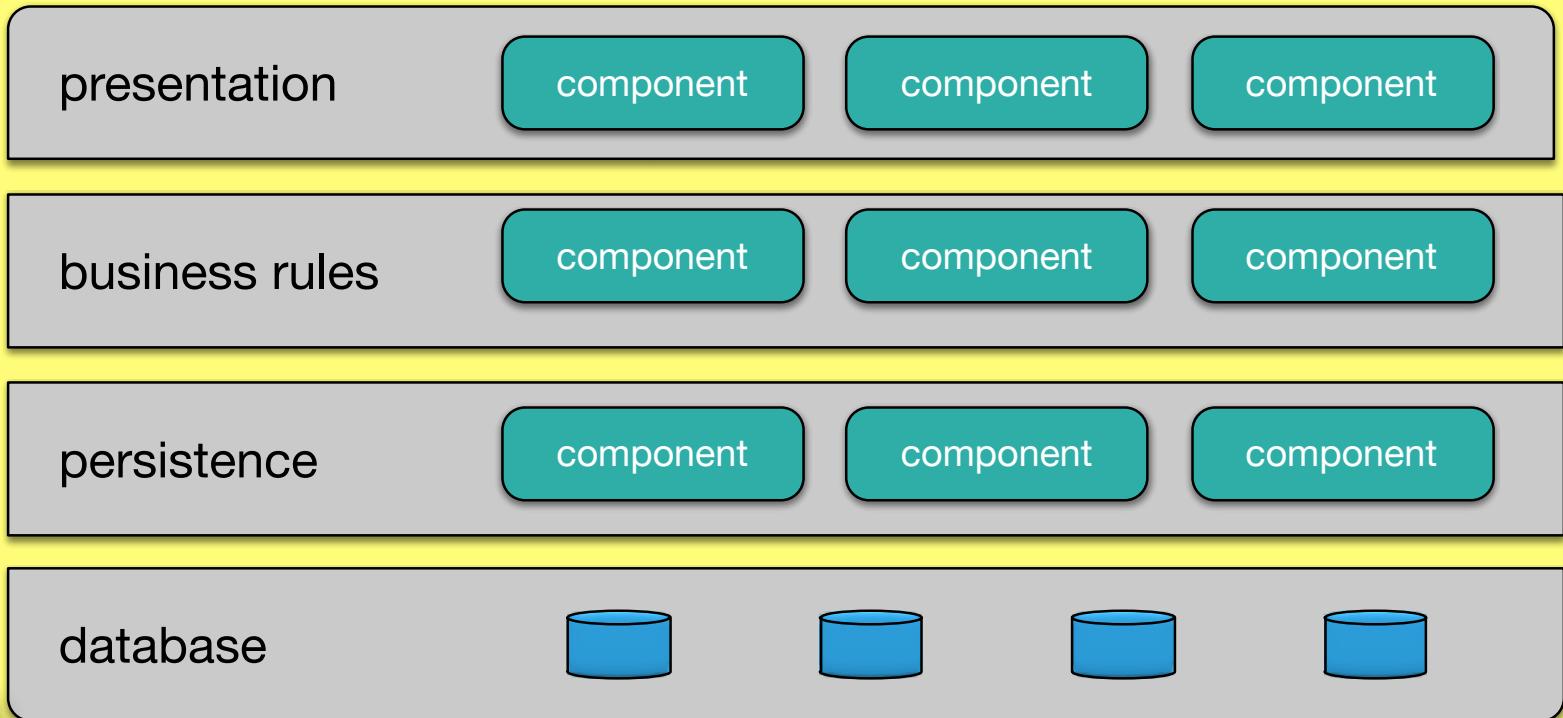
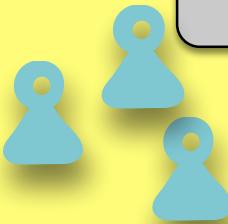


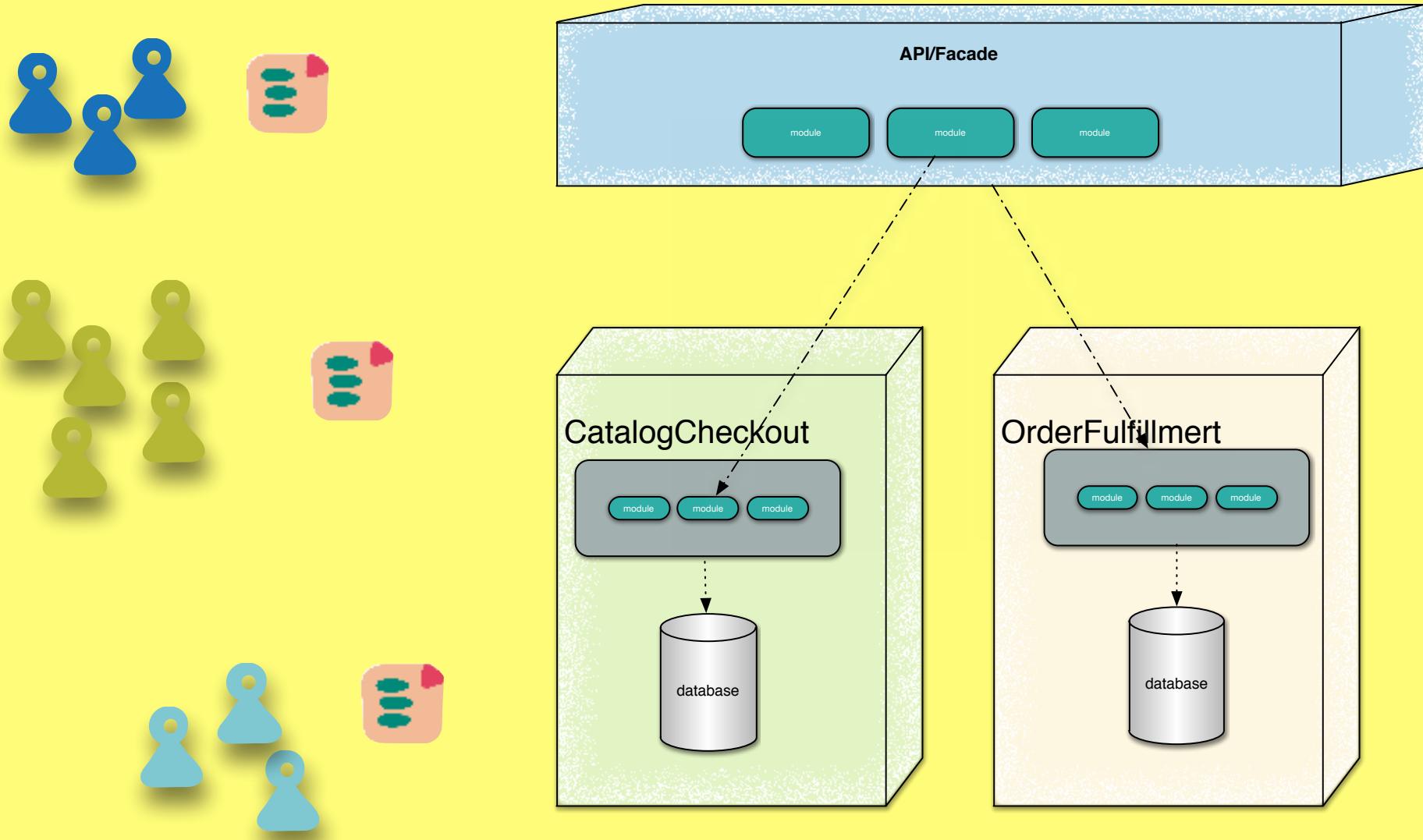
Low Efferent Coupling between Teams

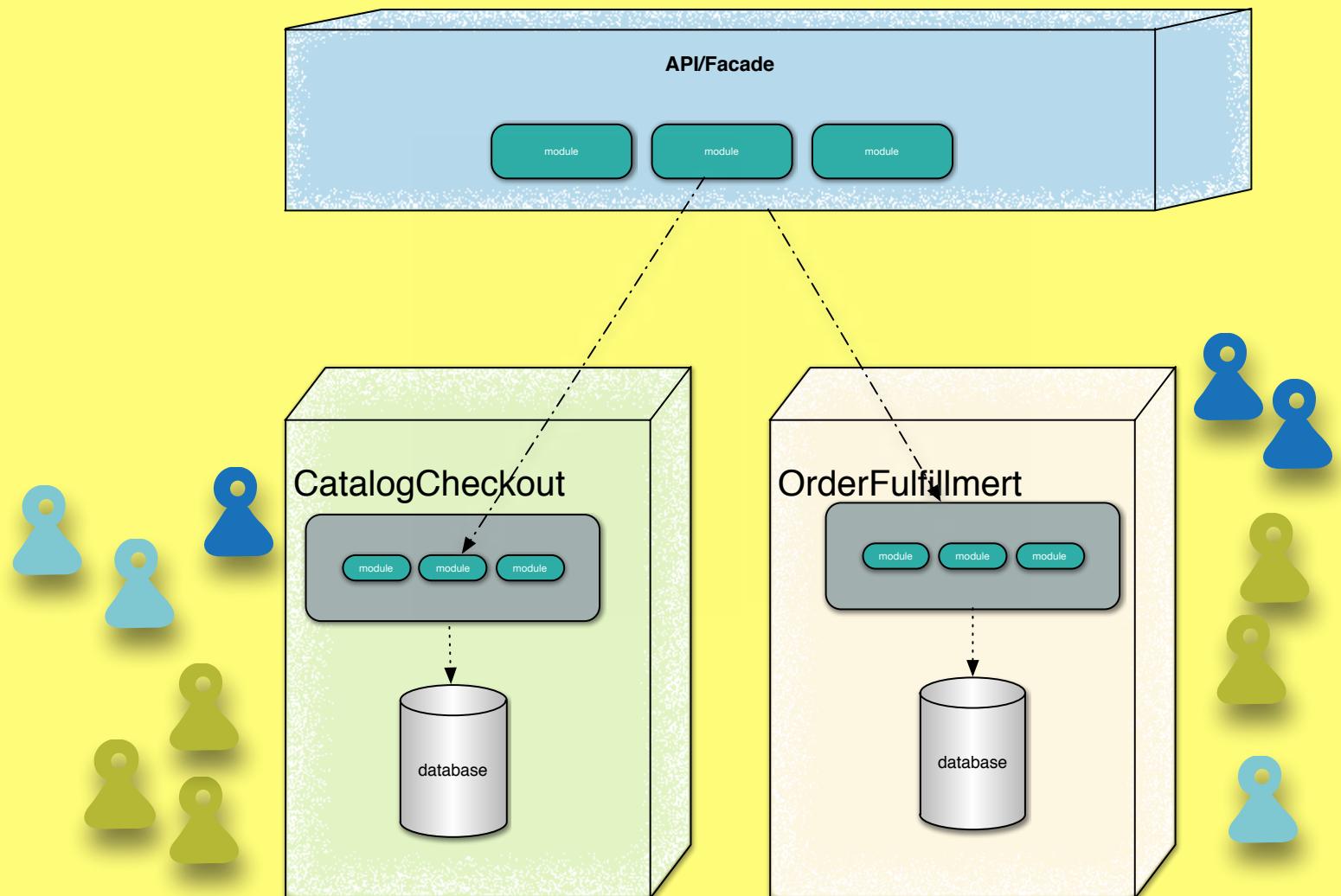


Penultima↑e



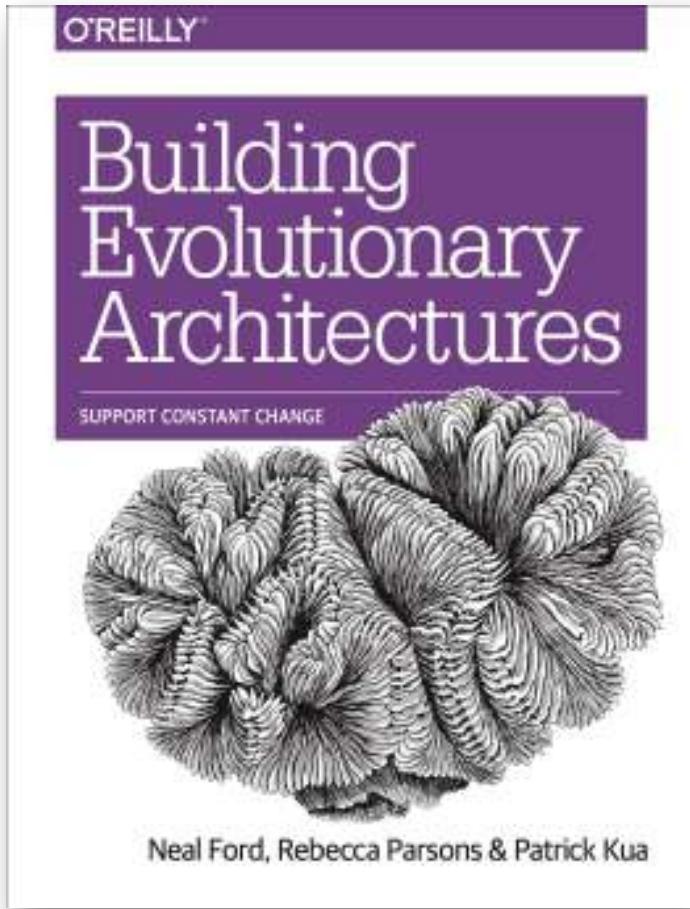






Building Evolutionary Architectures

FITNESS FUNCTIONS



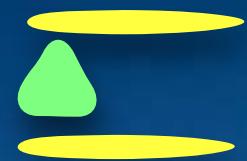
 @neal4d
nealford.com



Evolutionary Architecture

An evolutionary architecture supports
guided
incremental change across multiple dimensions.





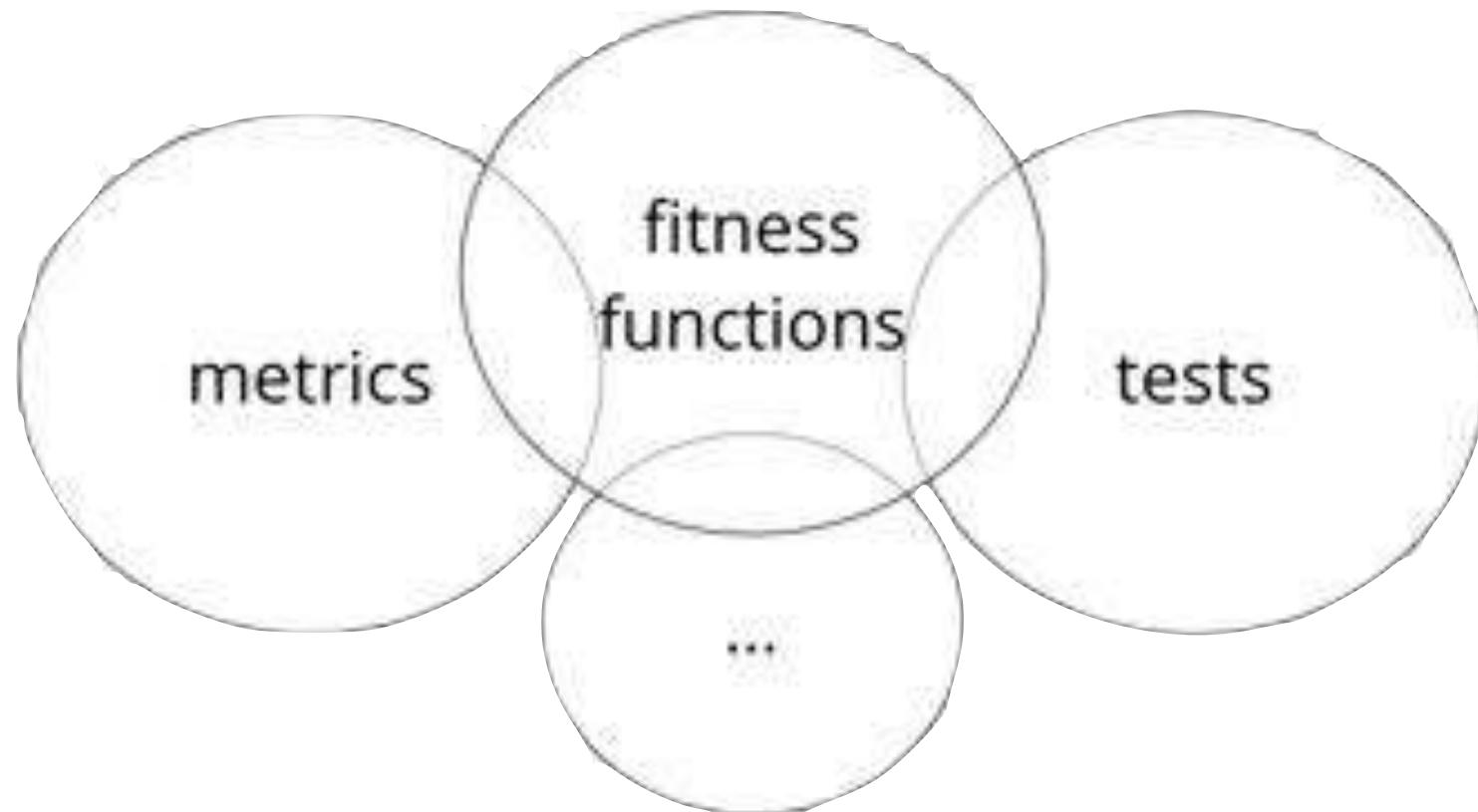
guided

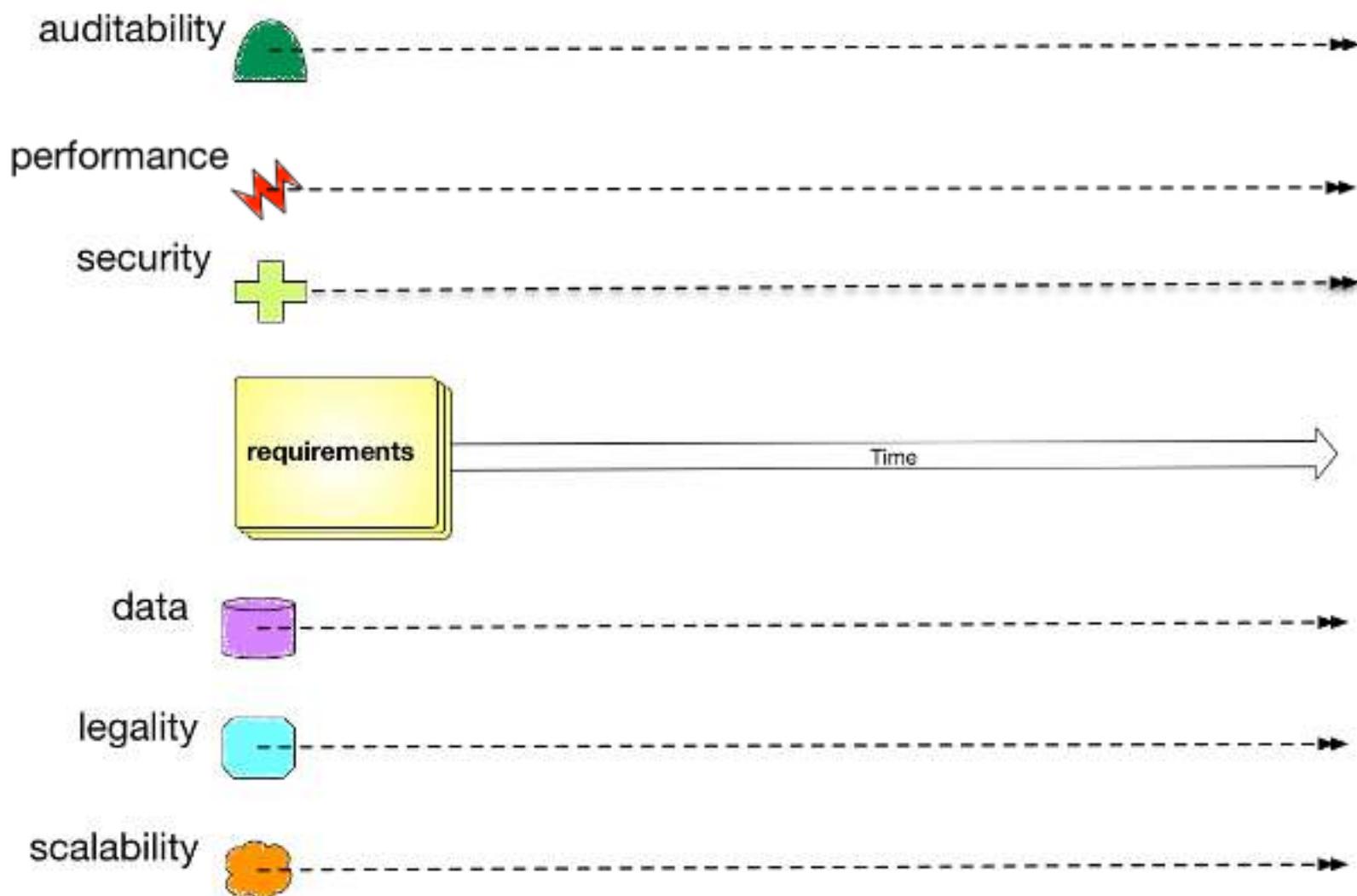




An architectural fitness function provides an objective integrity assessment of some architectural characteristic(s).

Fitness Functions





Performance

performance



tests?

monitors?

metrics?

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

triggered



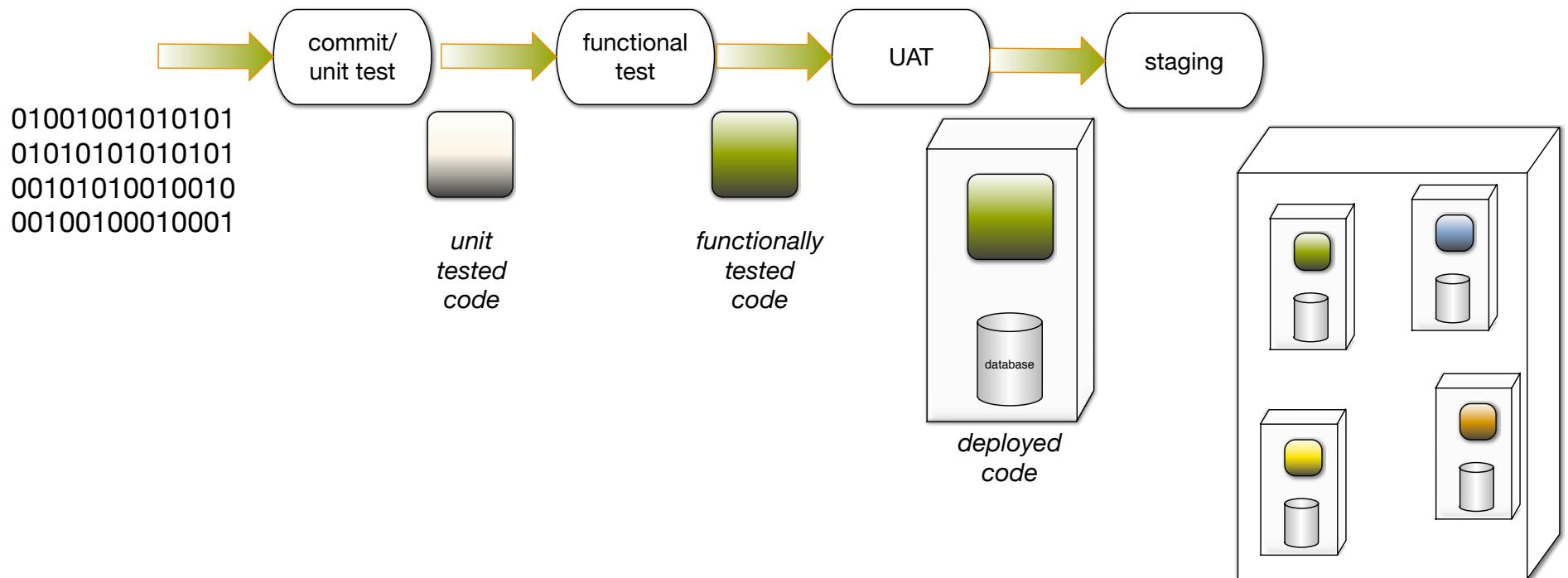
run based on a particular event, such as a developer executing a unit test, a deployment pipeline running unit tests, or a QA person performing exploratory testing.

continuous

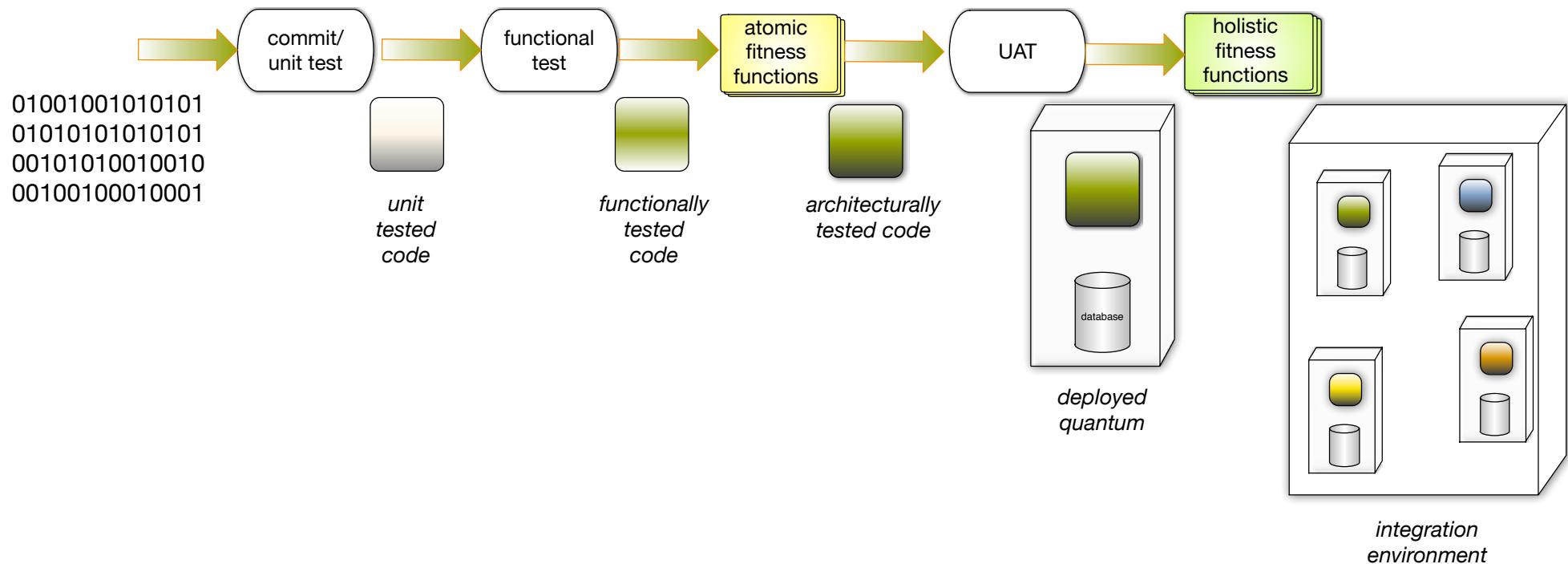


don't run on a schedule, but instead execute constant verification of architectural aspect(s) such as transaction speed.

Deployment Pipeline



Deployment Pipeline + Fitness Functions



Categories of Fitness Functions

static



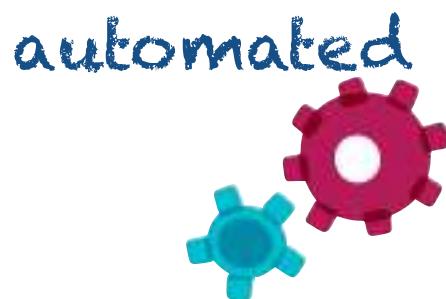
have a fixed result, such as the binary pass/fail of a unit test.

dynamic



rely on a shifting definition based on extra context. Some values may be contingent on circumstances, and most architects will accept lower performance metrics when operating at high scale.

Categories of Fitness Functions



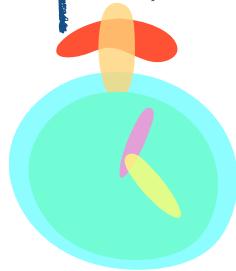
tests and other verification mechanism that run without human interaction.



must involve at least one human.

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

domain-specific



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

Categories of Fitness Functions



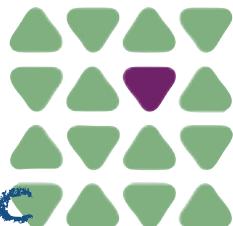
architects will define most fitness functions at project inception as they elucidate the characteristics of the architecture...



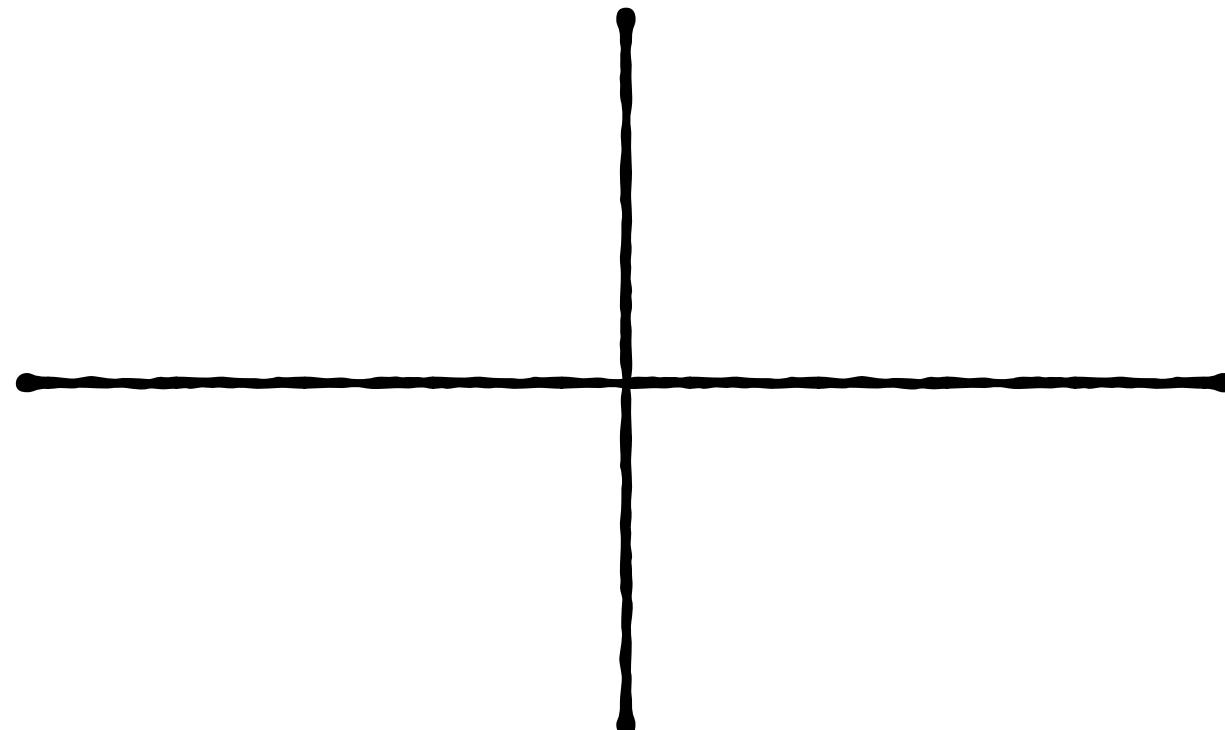
...some fitness functions will emerge during development of the system

Fitness Function

atomic



holistic



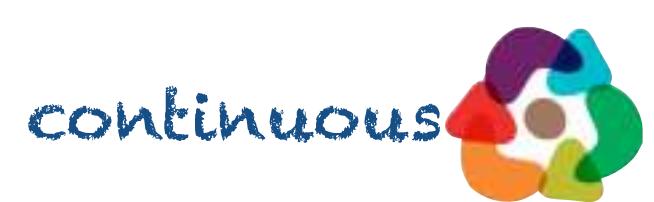
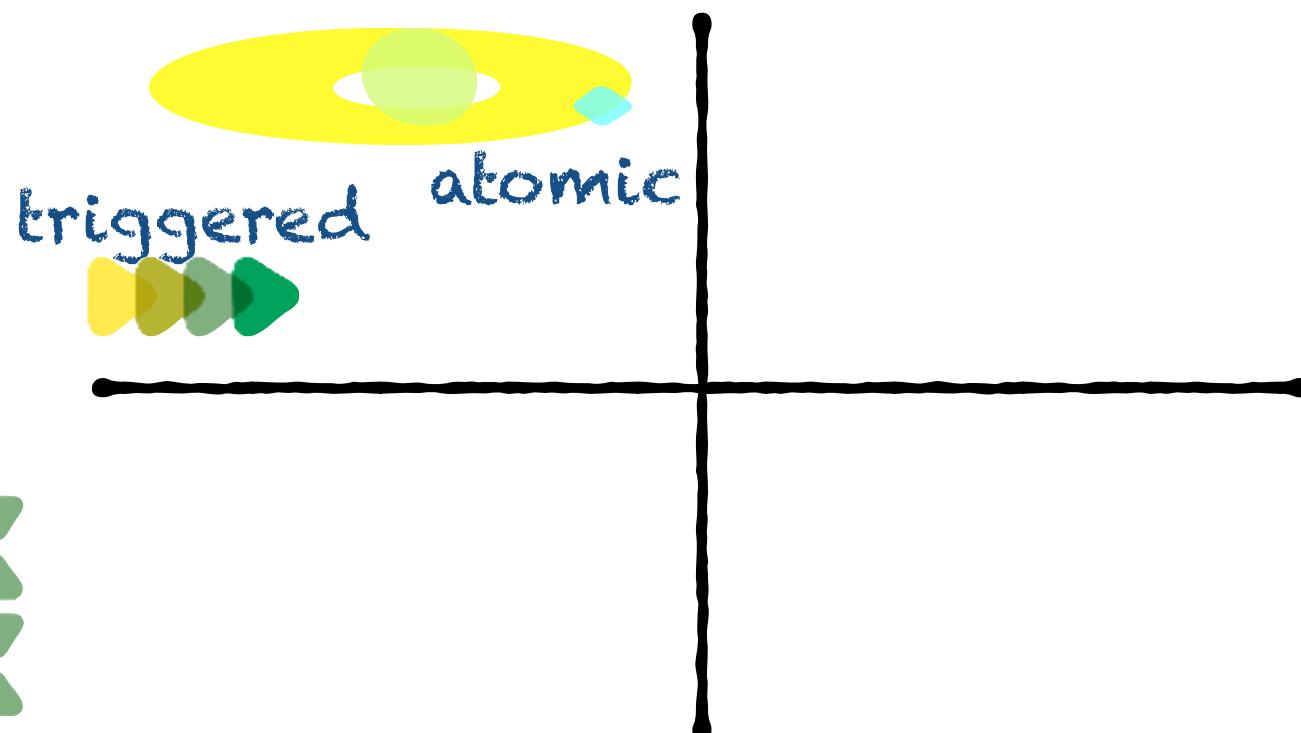
triggered



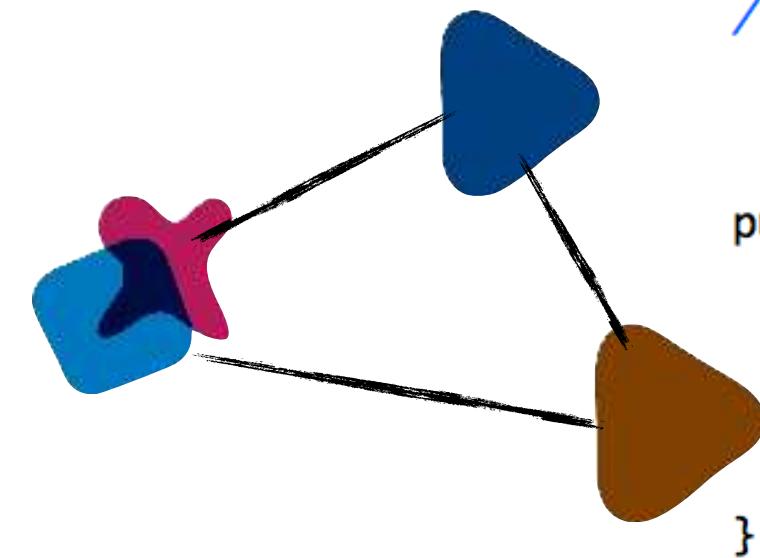
continuous



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

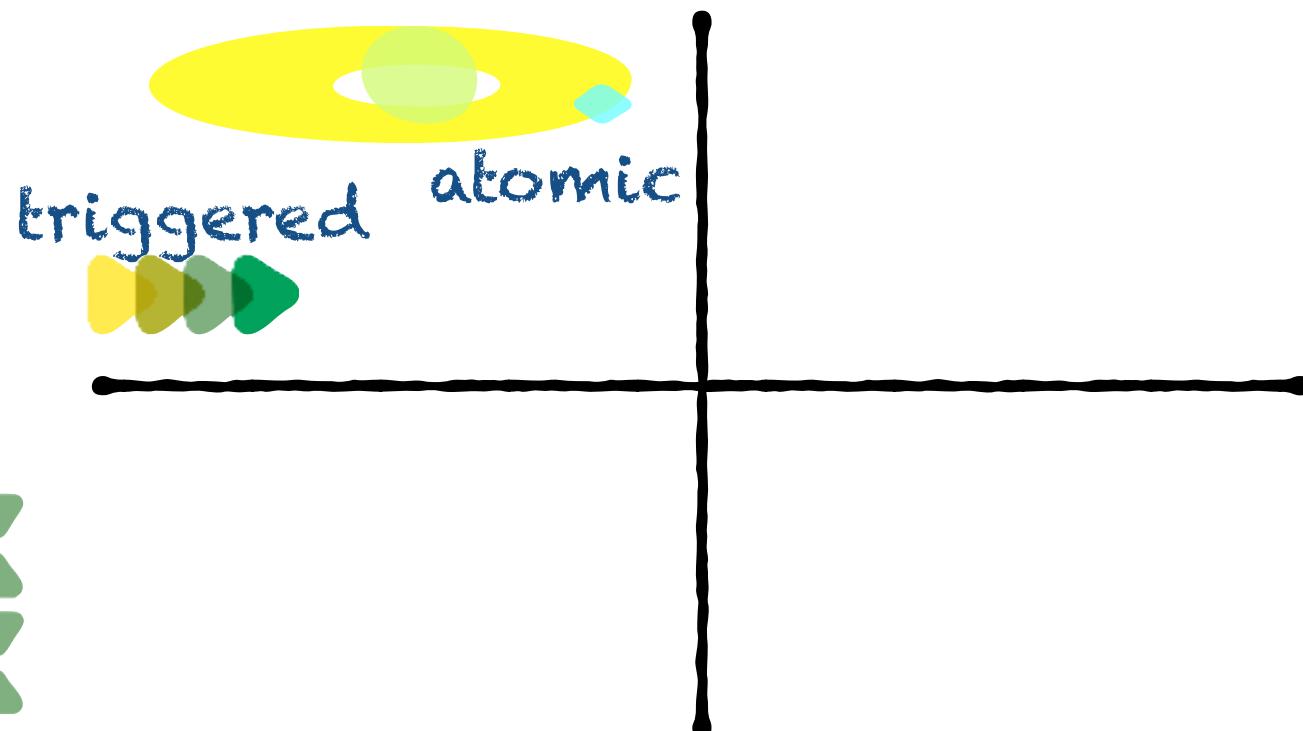


Coupling Fitness Function

```
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));  
}
```

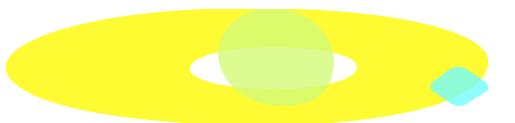


Fitness Function

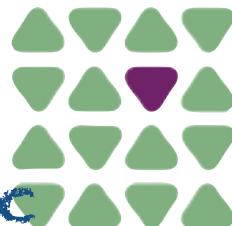
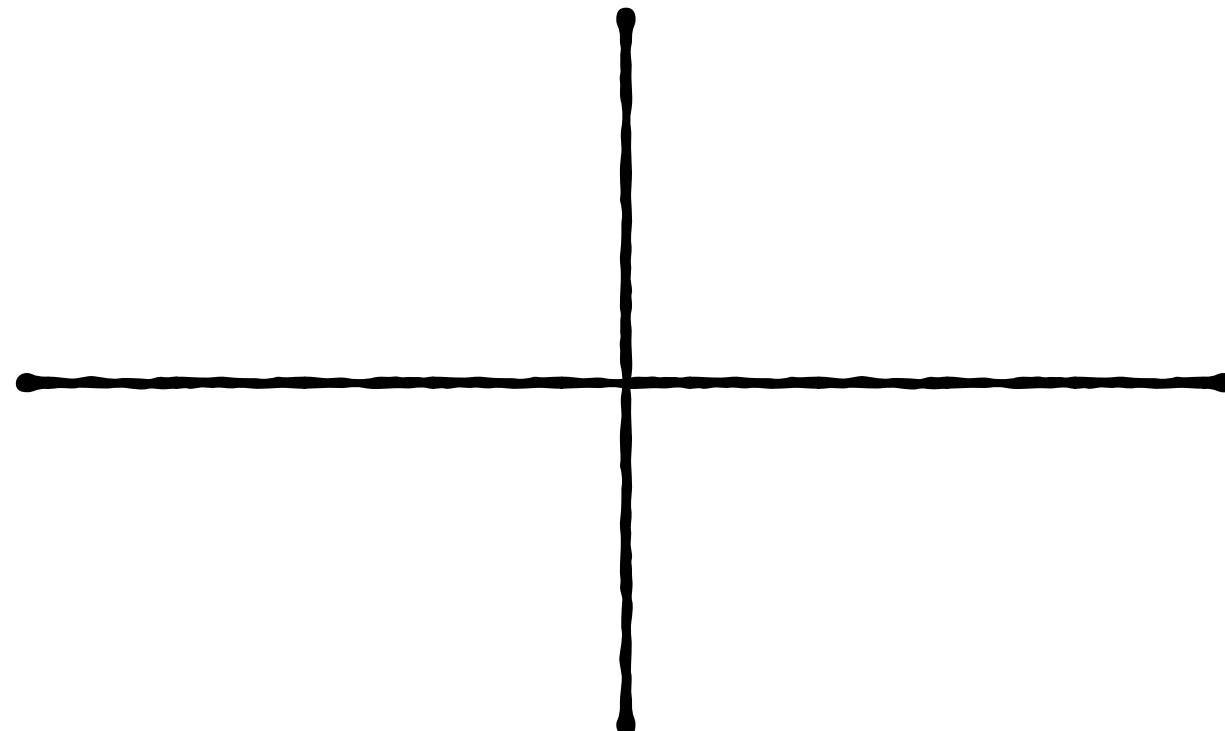


continuous

Fitness Function



atomic

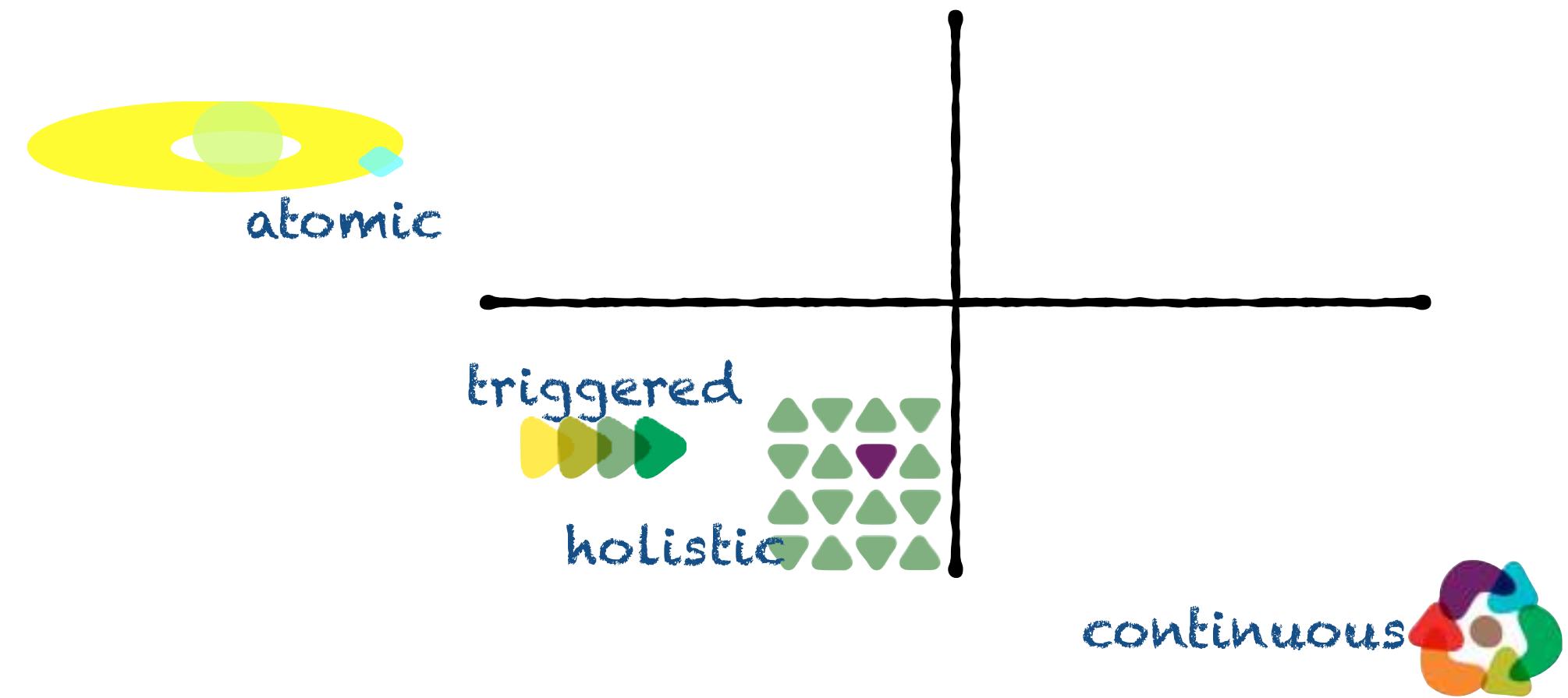


holistic

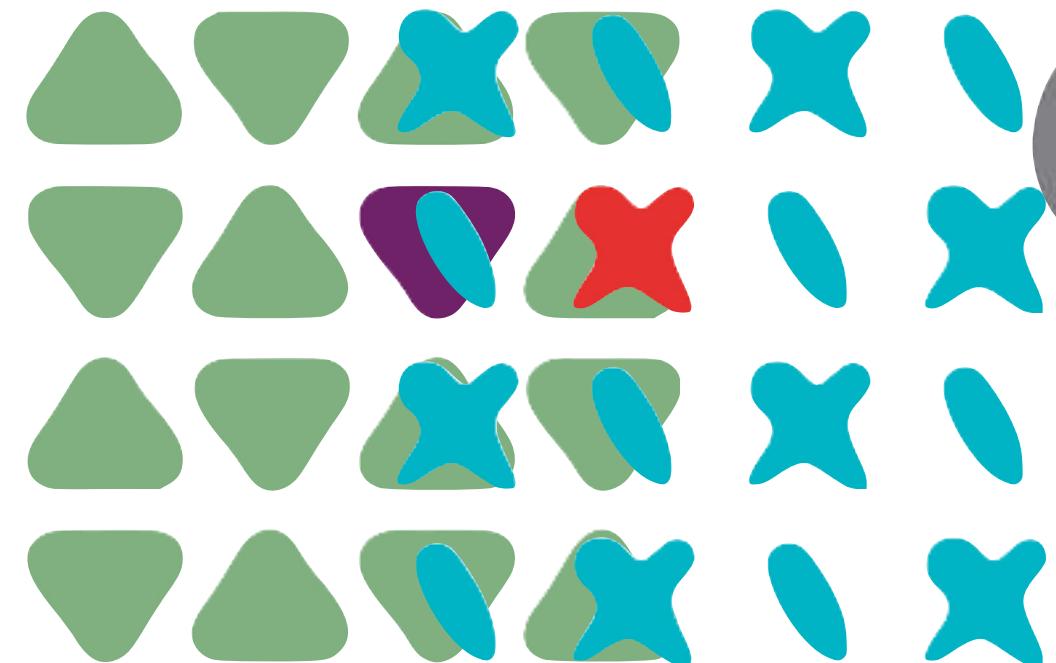


continuous

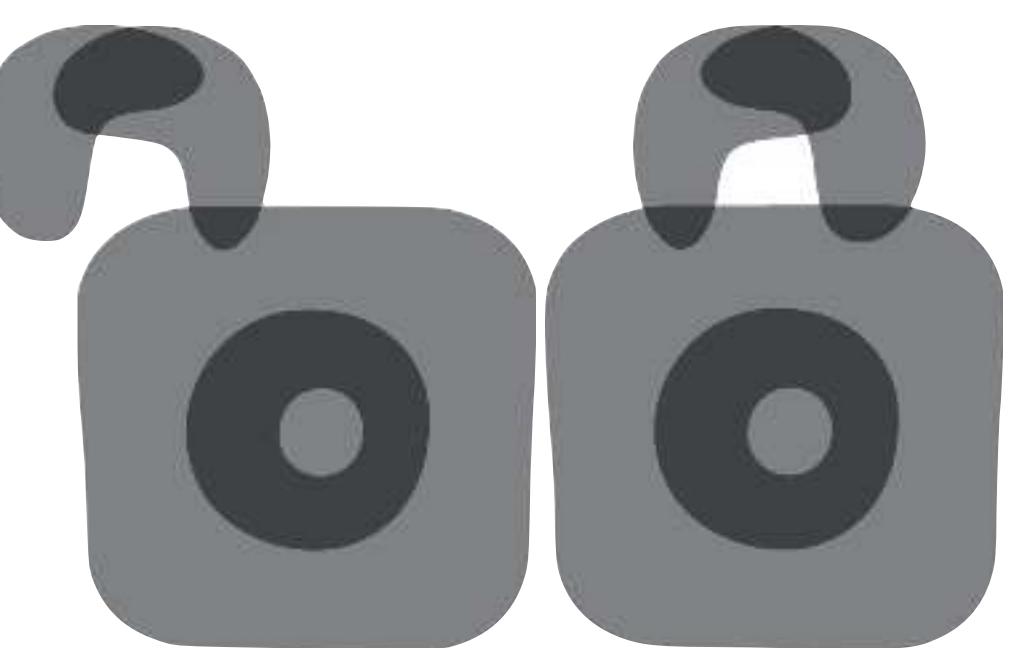
Fitness Function



triggered

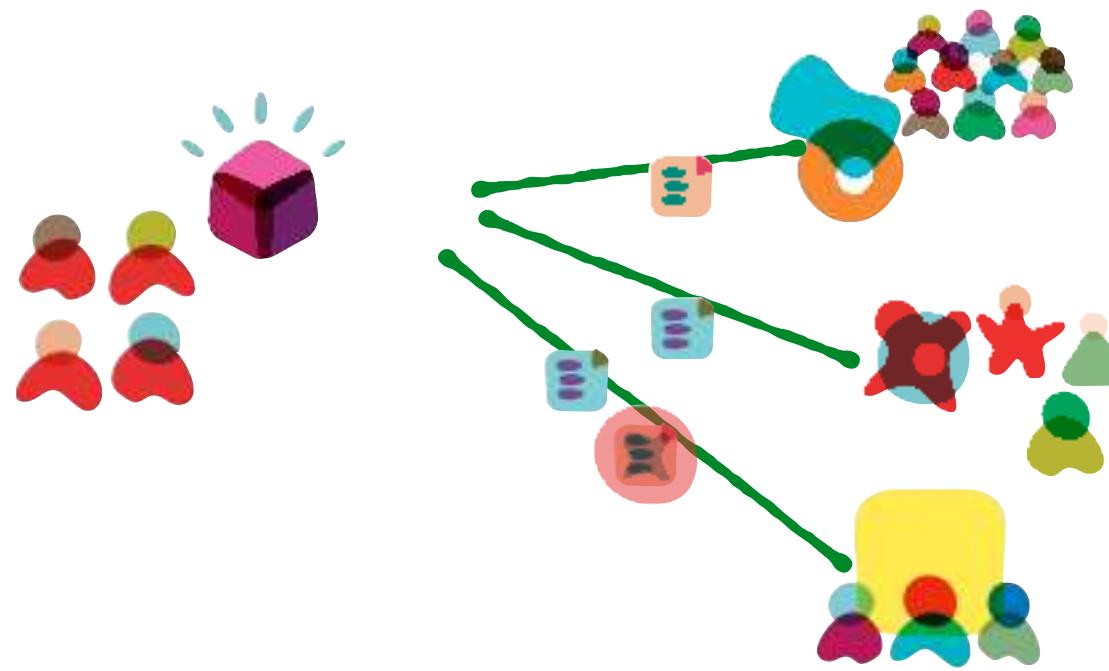


holistic

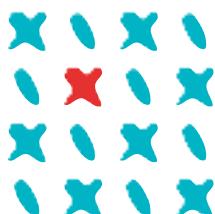


Holistic fitness functions must run in a specific (shared) context.

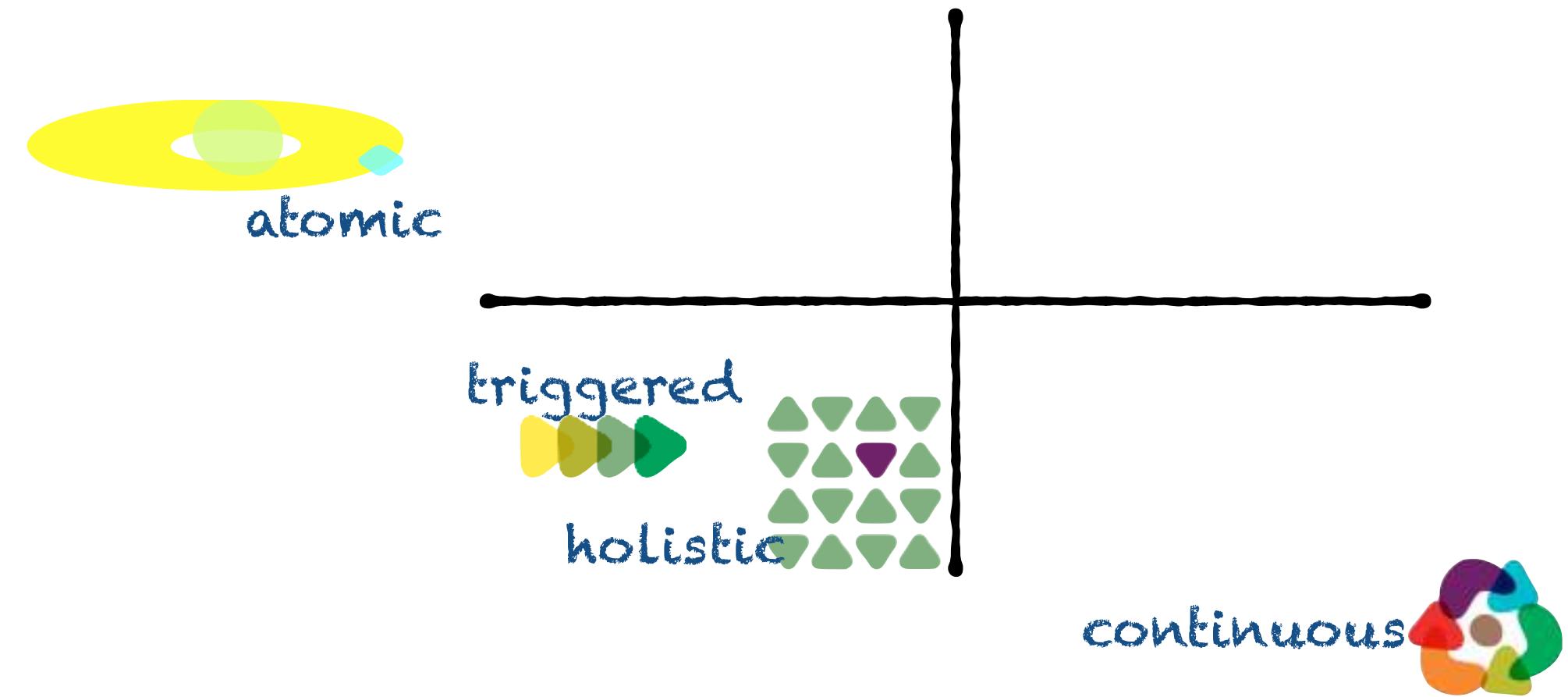
Consumer Driven Contracts



martinfowler.com/articles/consumerDrivenContracts.html

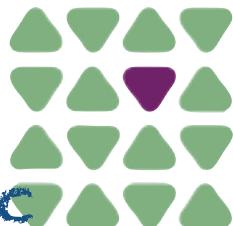


Fitness Function

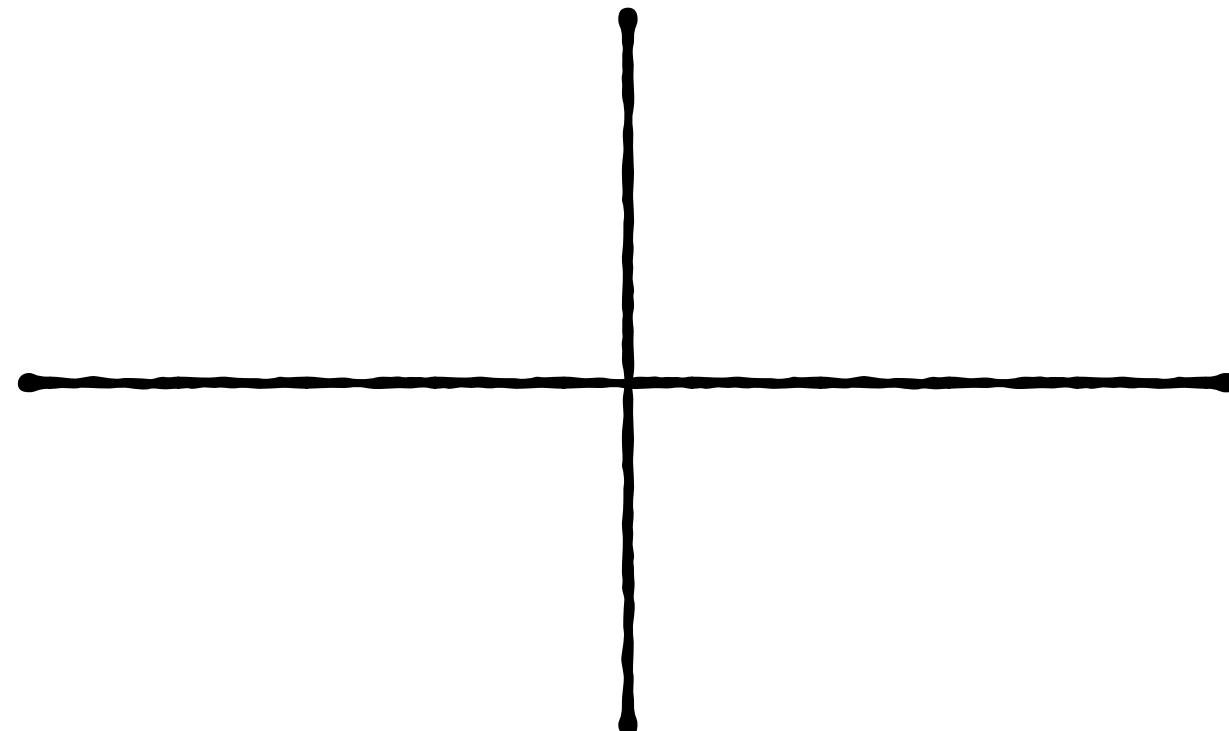


Fitness Function

atomic



holistic



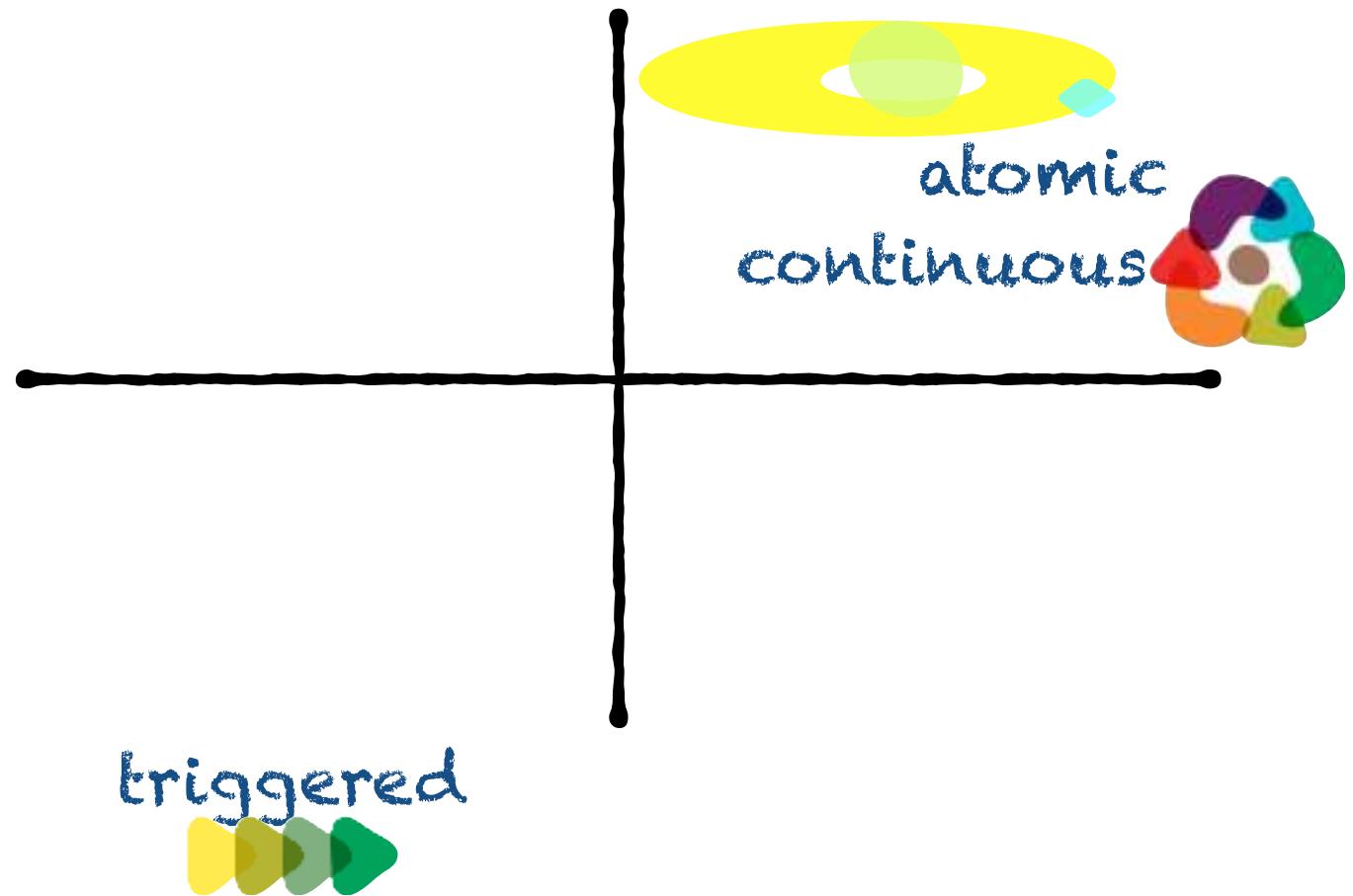
triggered



continuous



Fitness Function

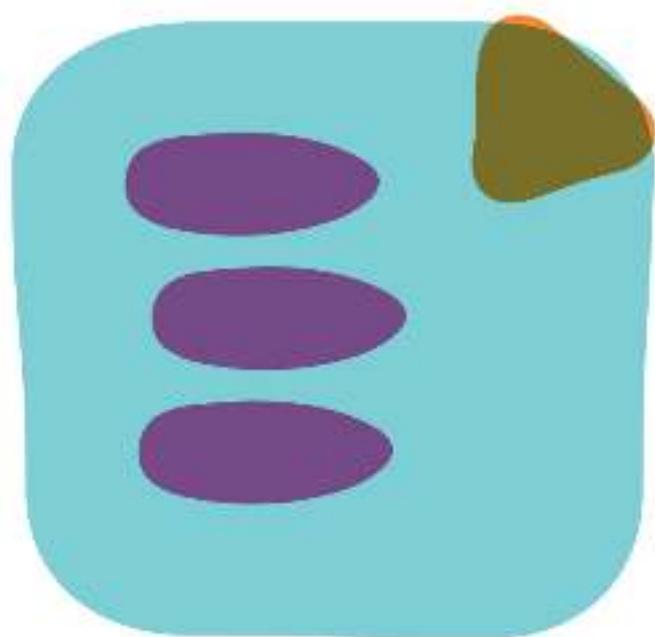




atomic

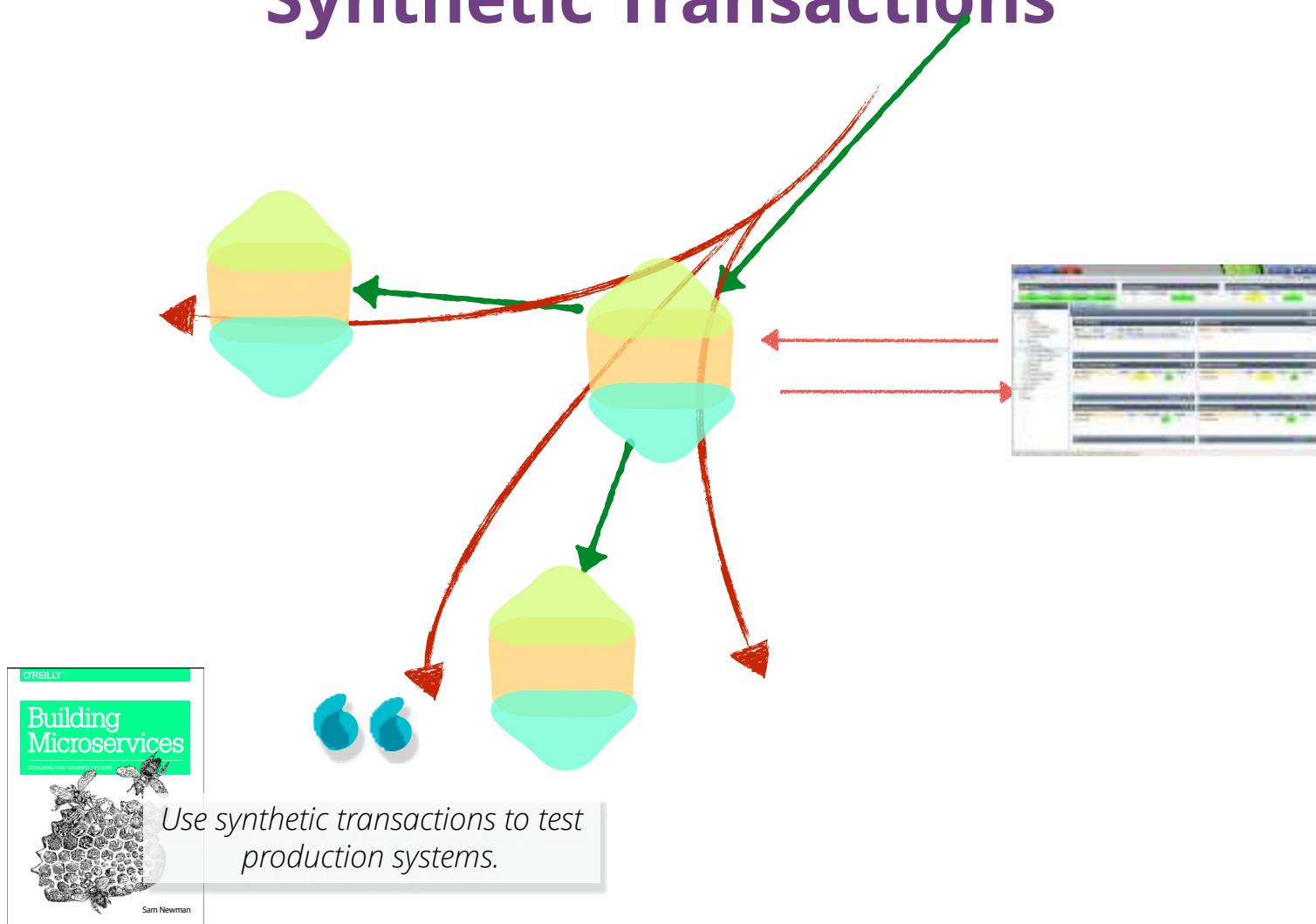


monitoring

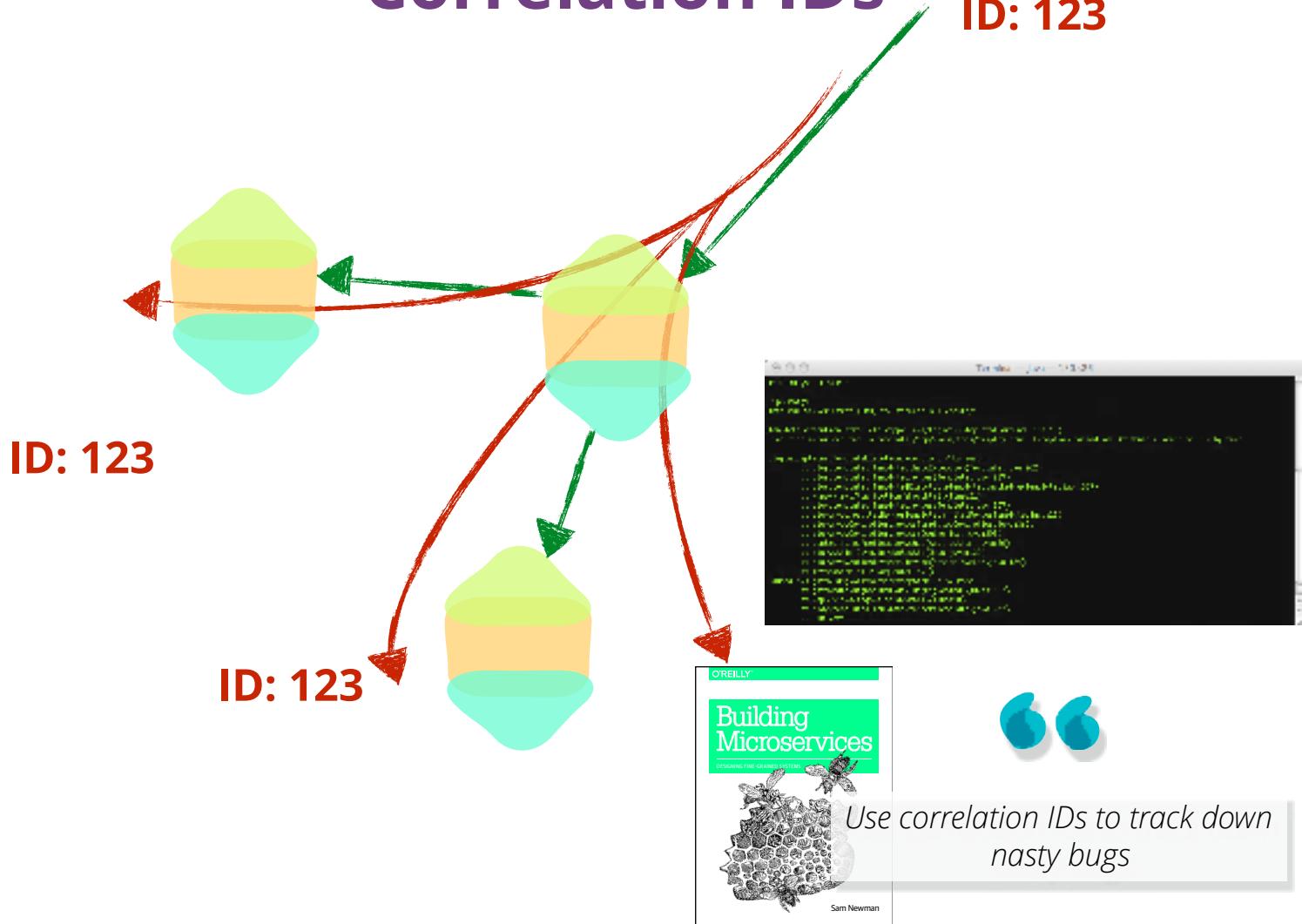


logging

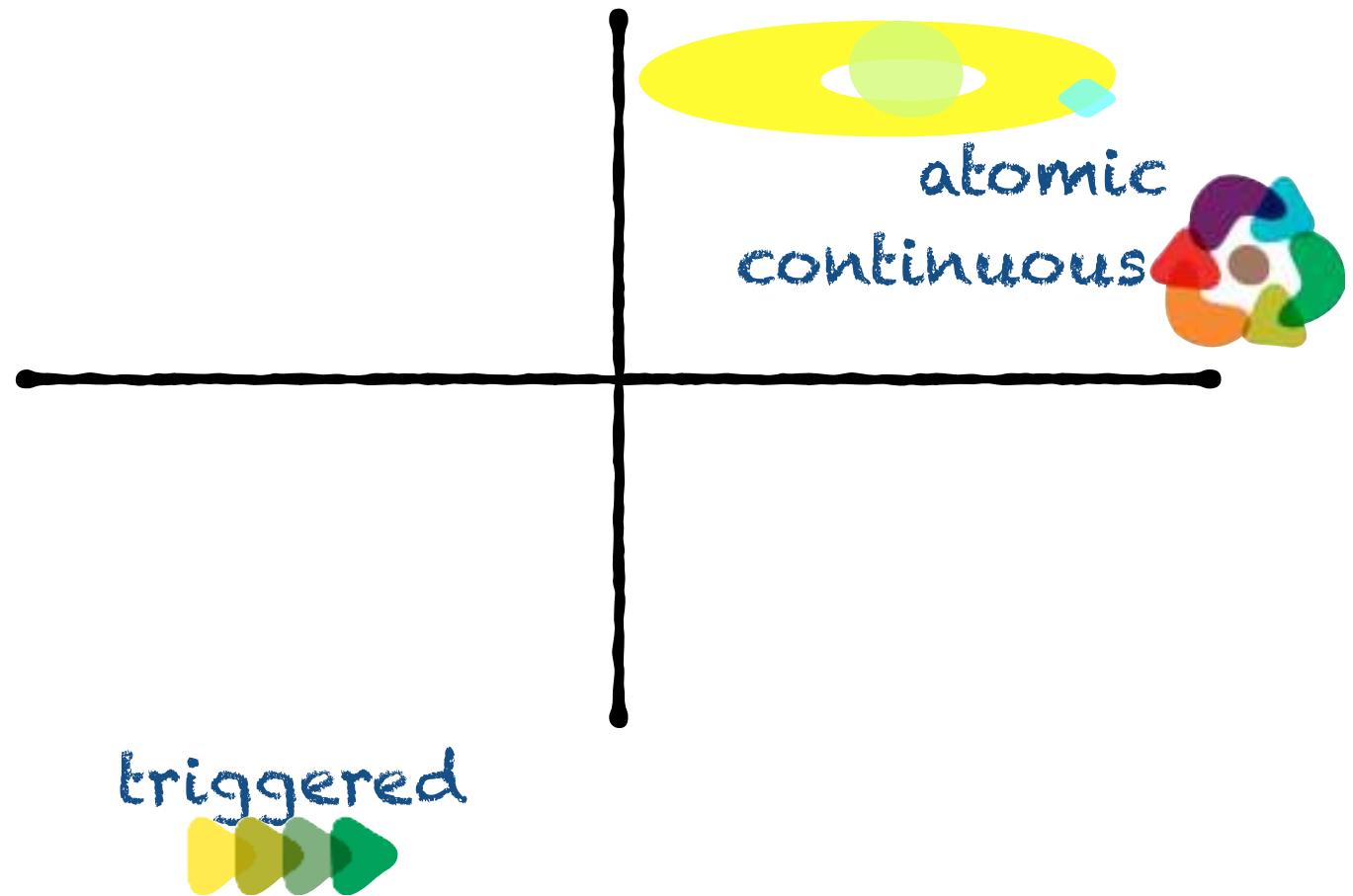
Synthetic Transactions



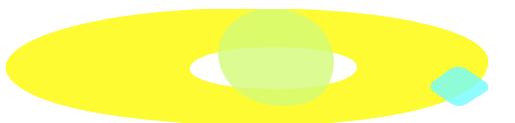
Correlation IDs



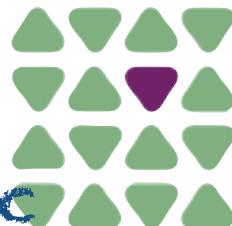
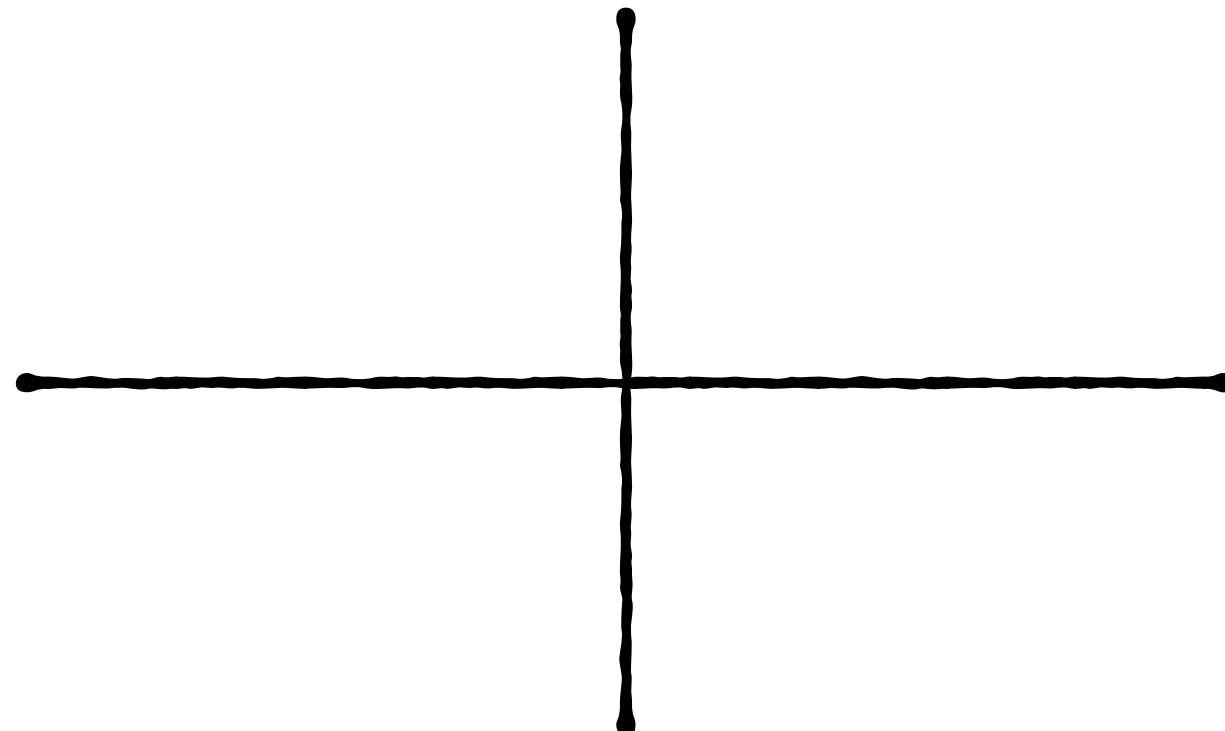
Fitness Function



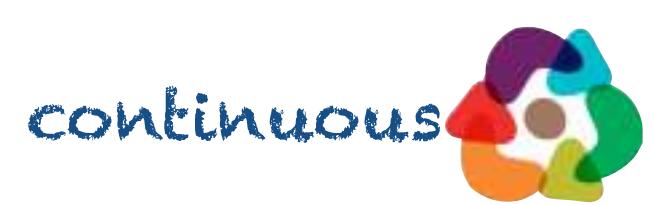
Fitness Function



atomic

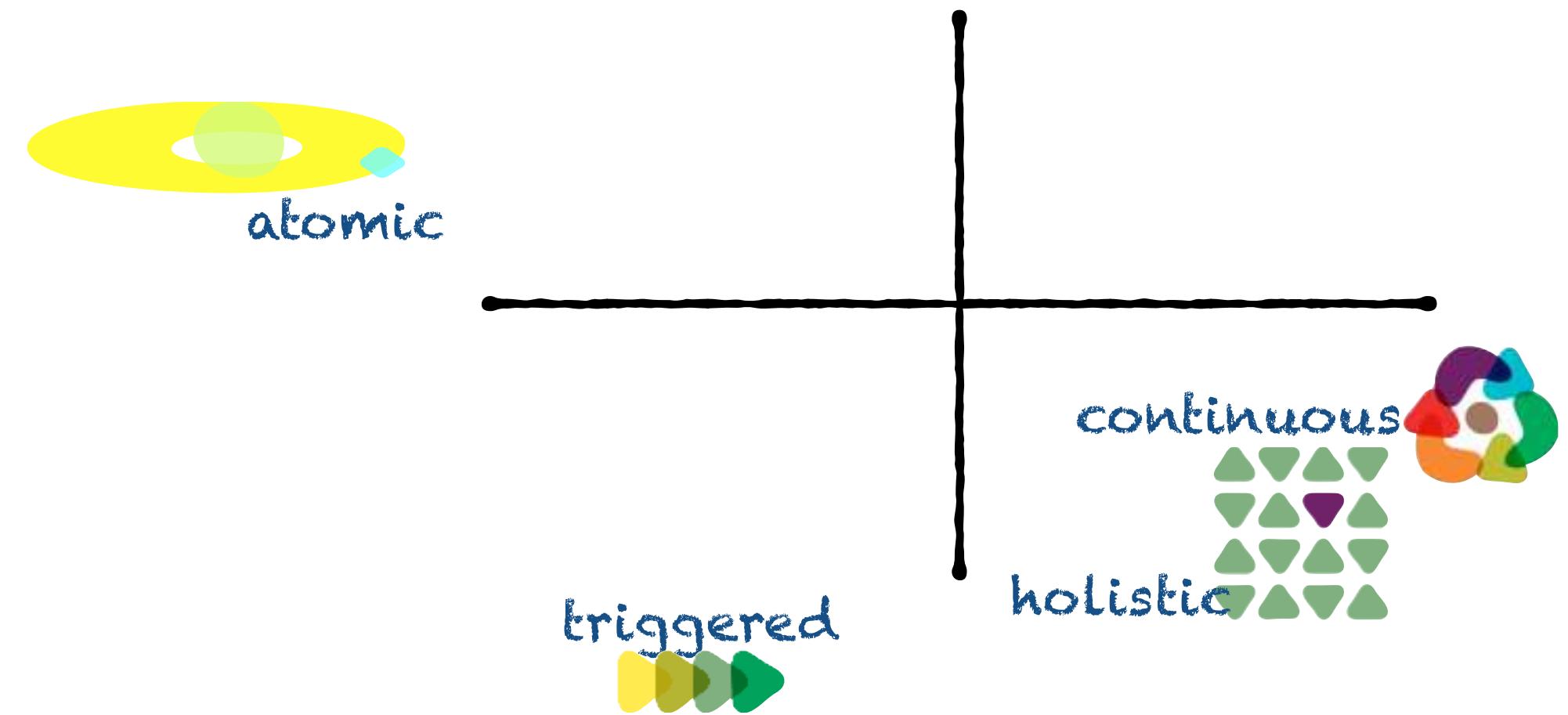


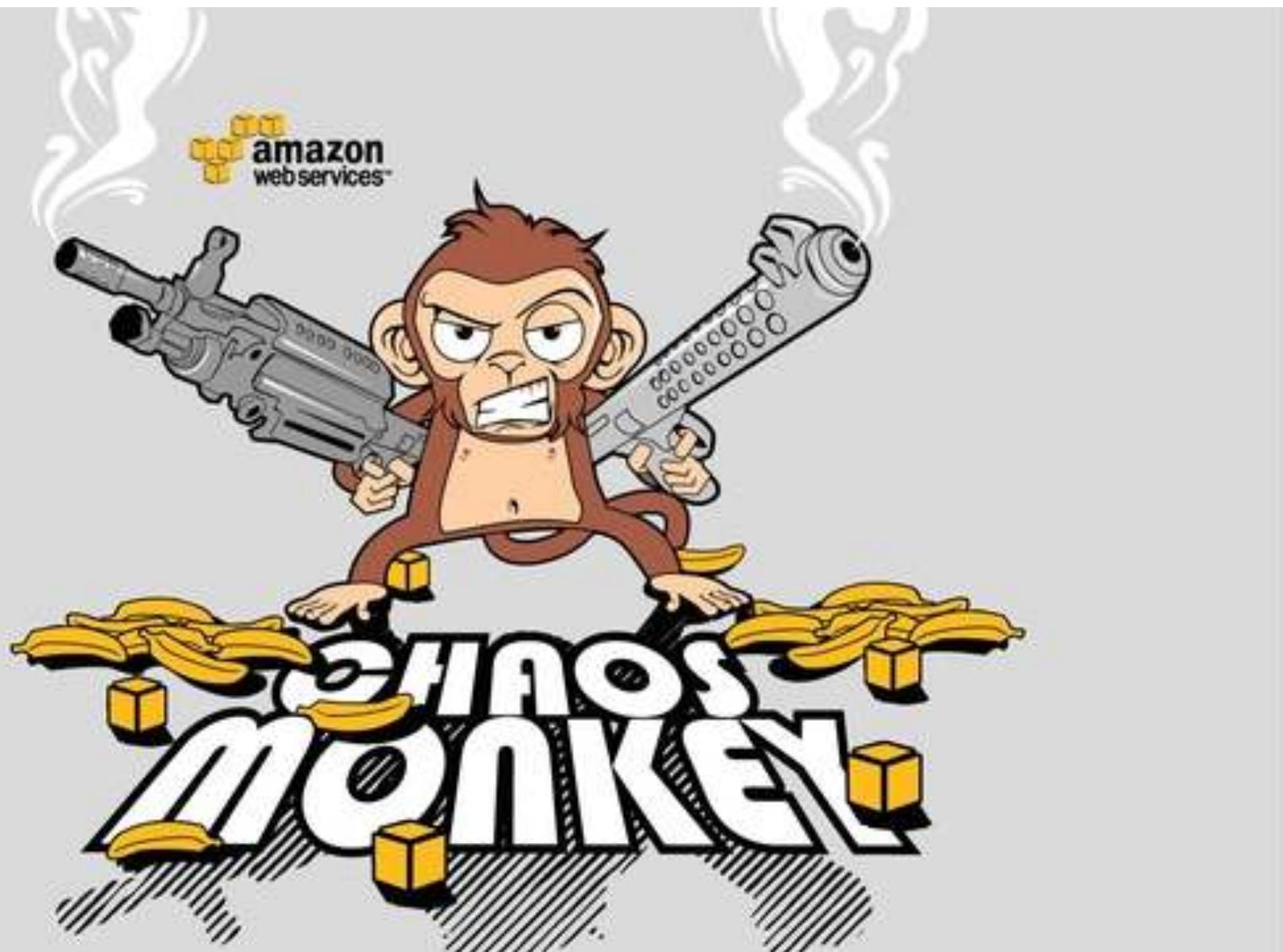
holistic



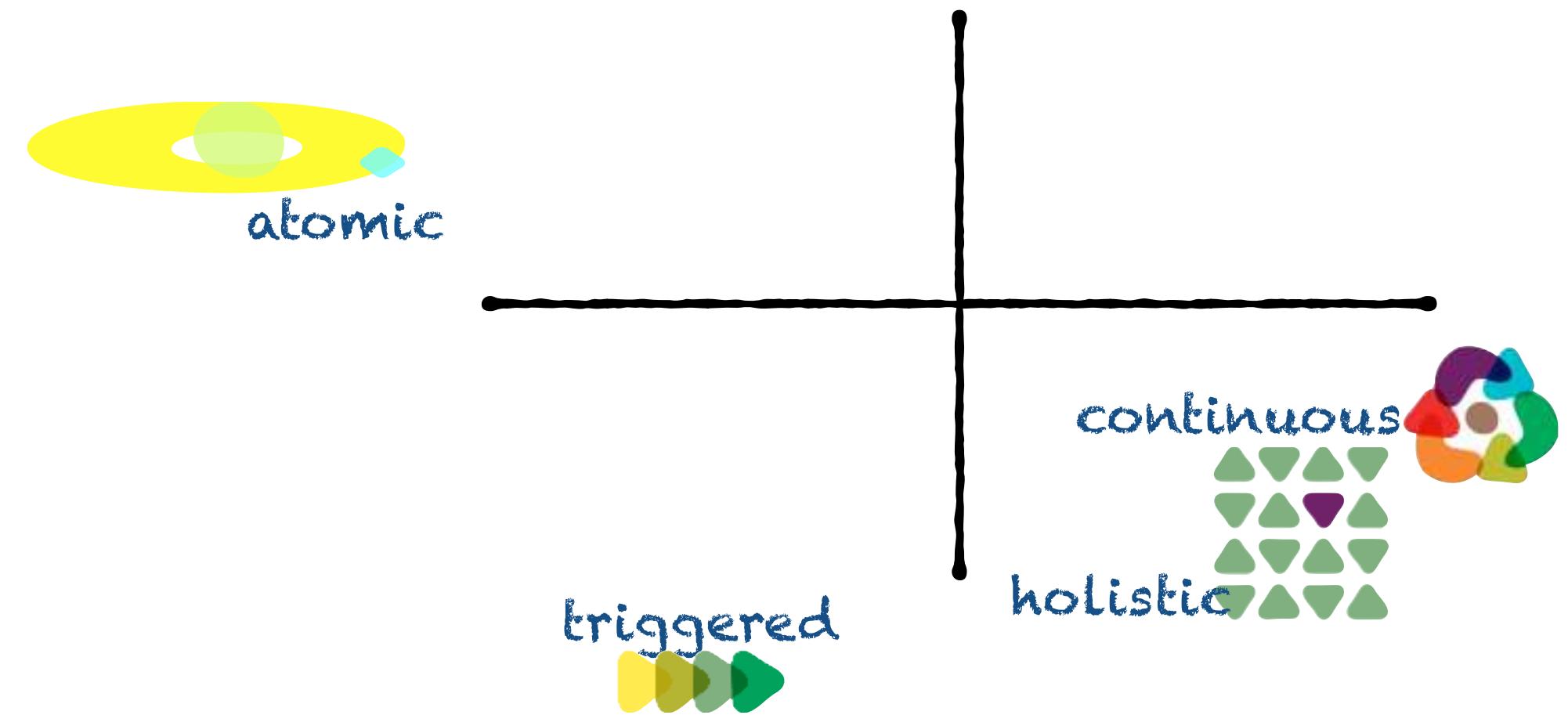
continuous

Fitness Function

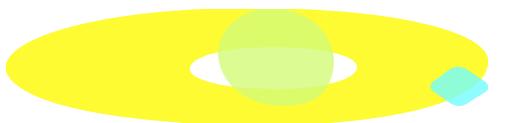




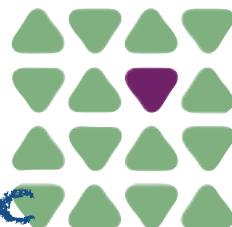
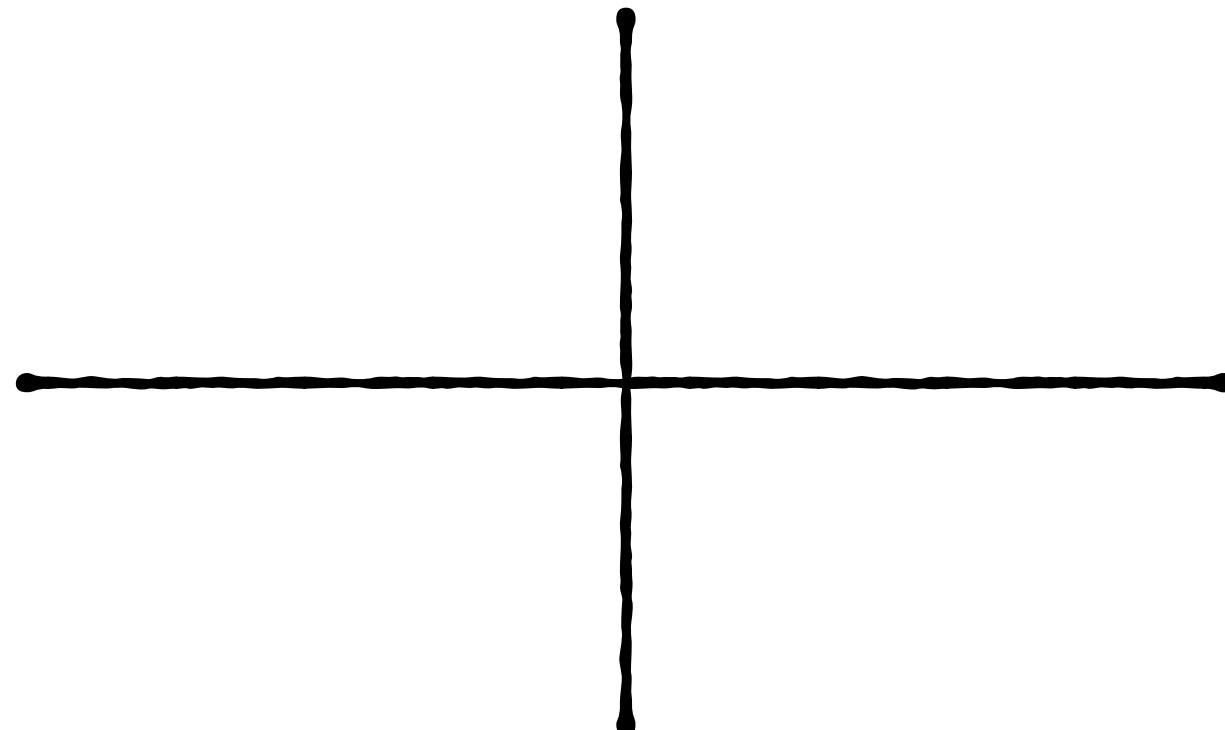
Fitness Function



Fitness Function



atomic

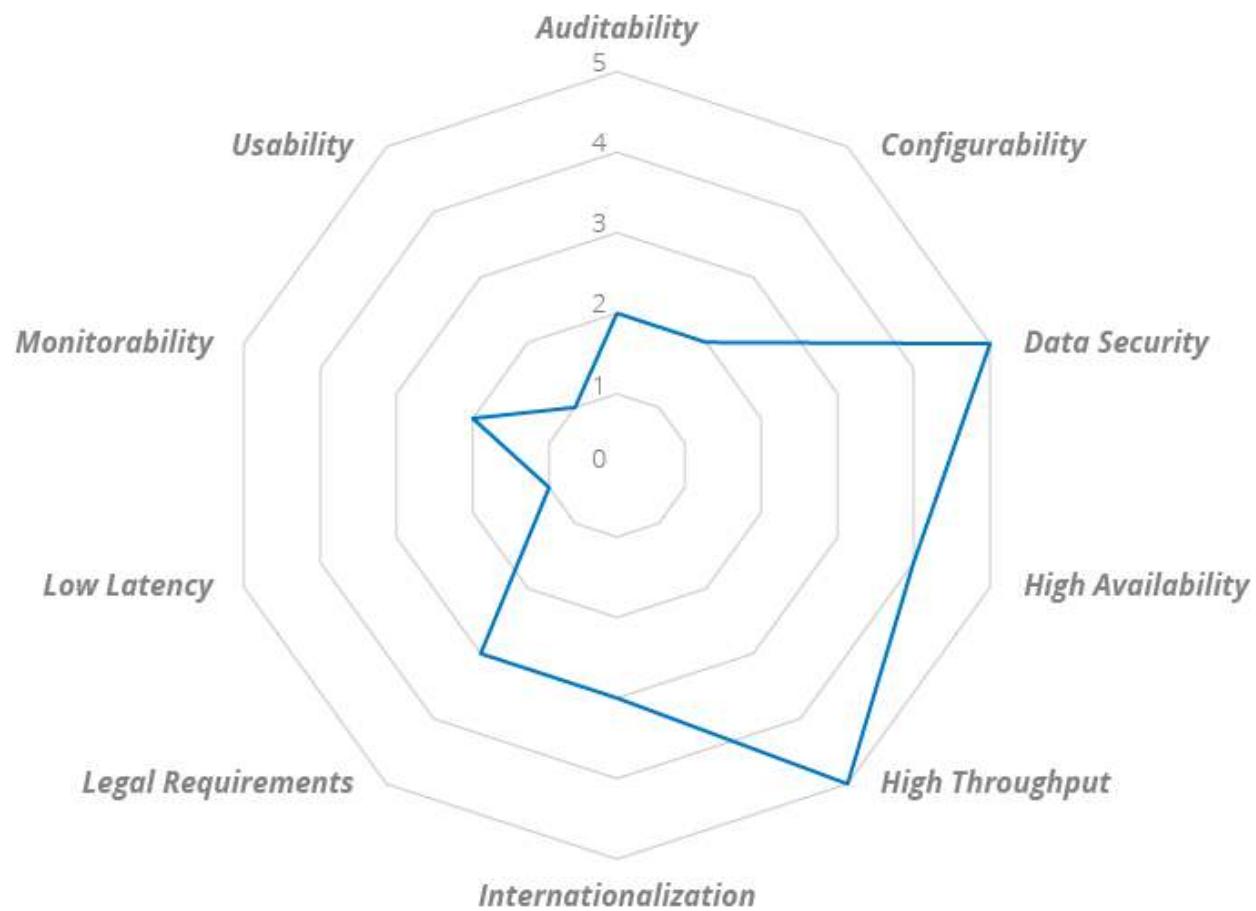


holistic

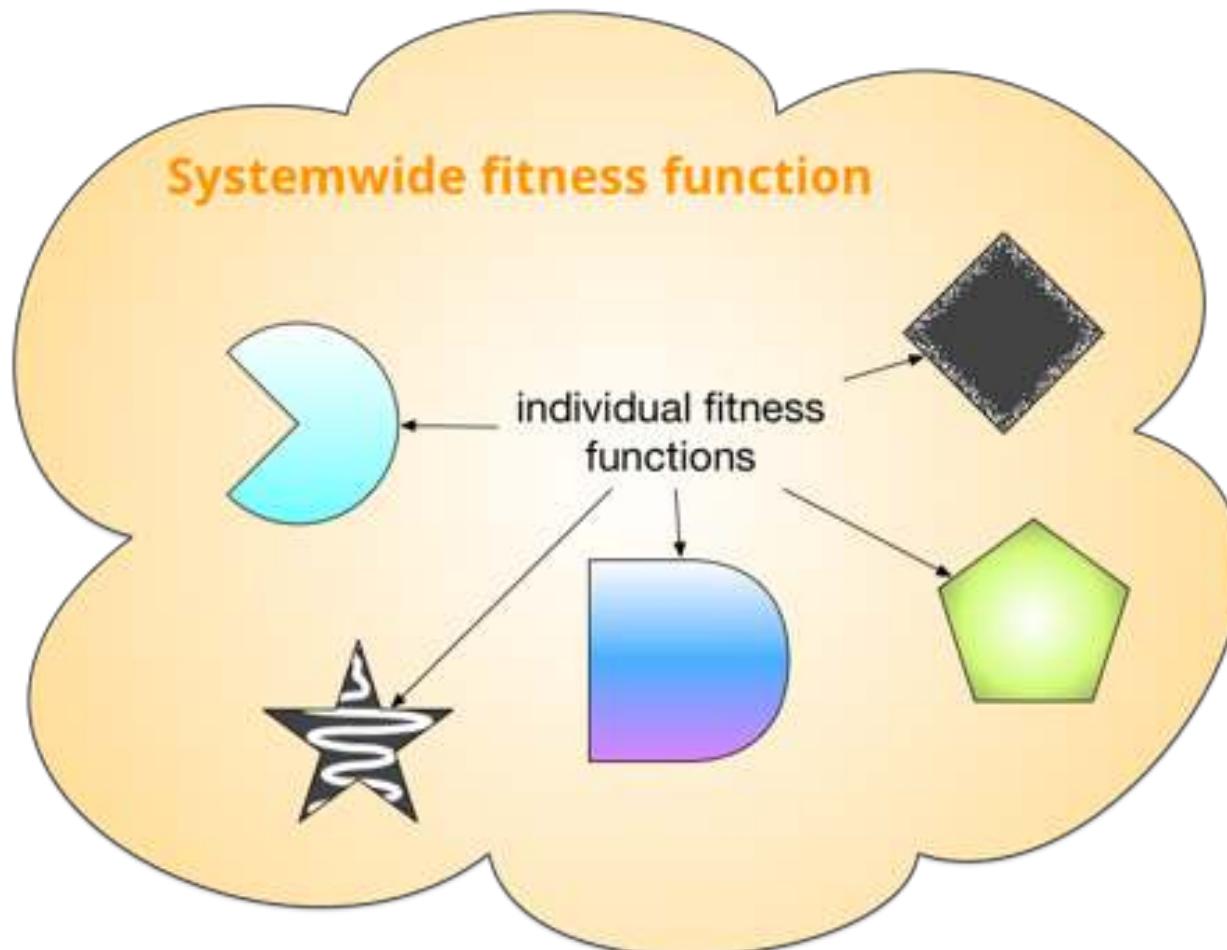


continuous

Fitness Function Fit



System-wide Fitness Function



**Identify Fitness Functions
Early**

Types of Fitness Functions

- **Key:** These dimensions are critical in making technology or design choices.
- **Relevant:** These dimensions need to be considered at a feature level, but are unlikely to guide architecture choices.
- **Not Relevant:** Design and technology choices are not impacted by these types of dimensions.



Keep knowledge of key and relevant fitness functions alive by posting the results of executing fitness functions somewhere visible or in a shared space so that developers remember to consider them in day-to-day coding.

Review Fitness Functions

- Reviewing existing fitness functions
- Checking the relevancy of the current fitness functions
- Determining change in the scale or magnitude of each fitness function
- Deciding if there are better approaches for measuring or testing the system's fitness functions
- Discovering new fitness functions that the system might need to support

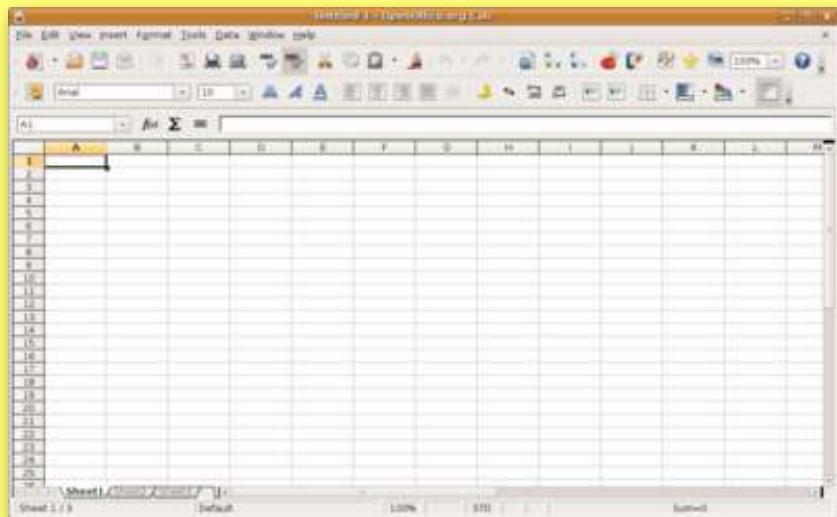


**Review your fitness
functions at least
once a year.**

Guided Evolution



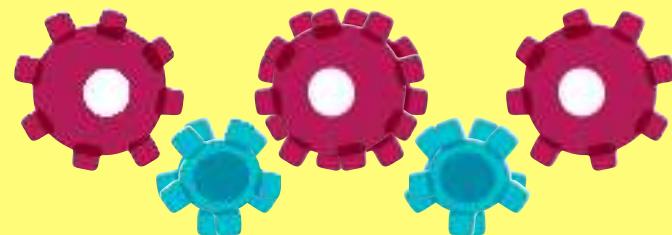
EA Spreadsheet



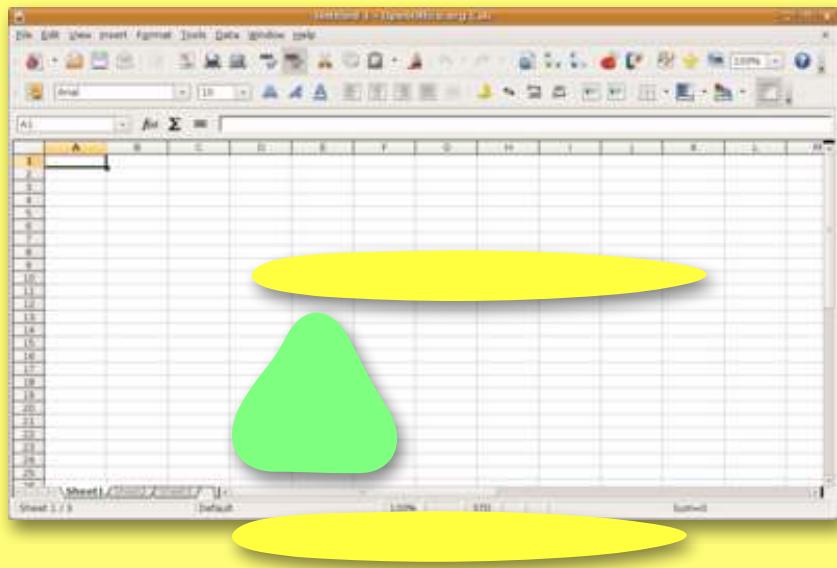
✓ definition

! verification

Penultima ↑ e



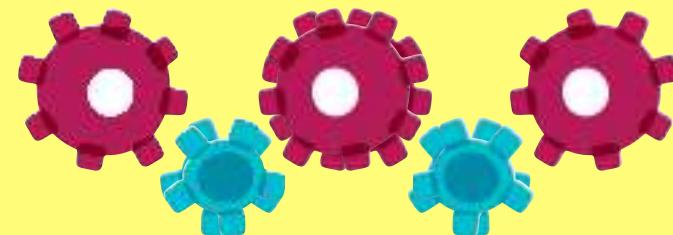
EA Spreadsheet



✓ definition

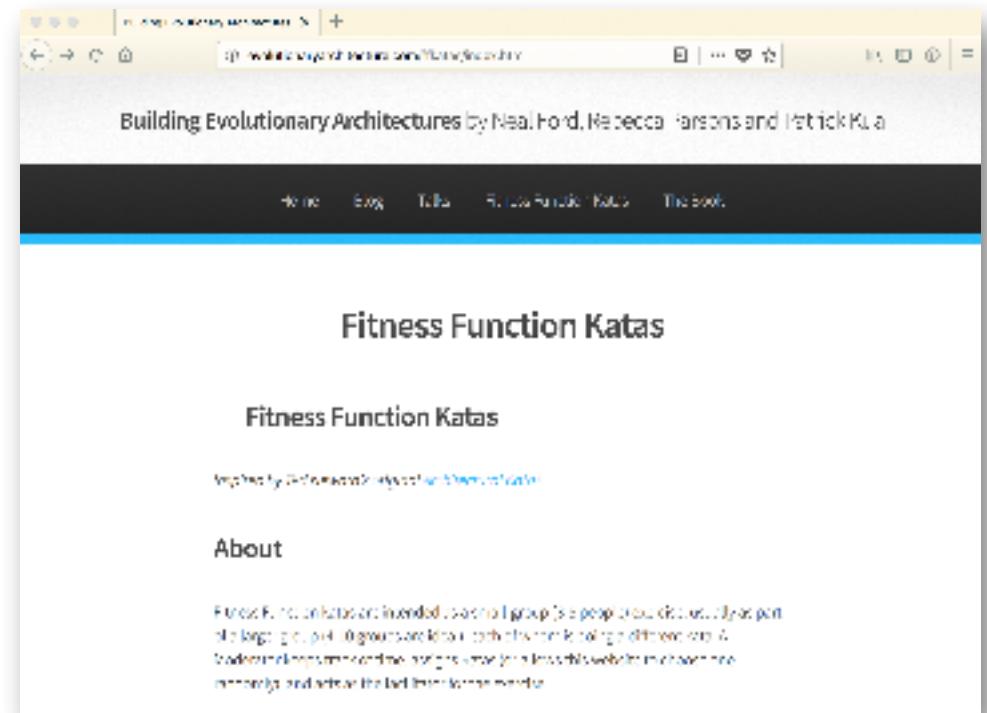
✗ verification

Penultima ↑ e



Fitness Function Katas

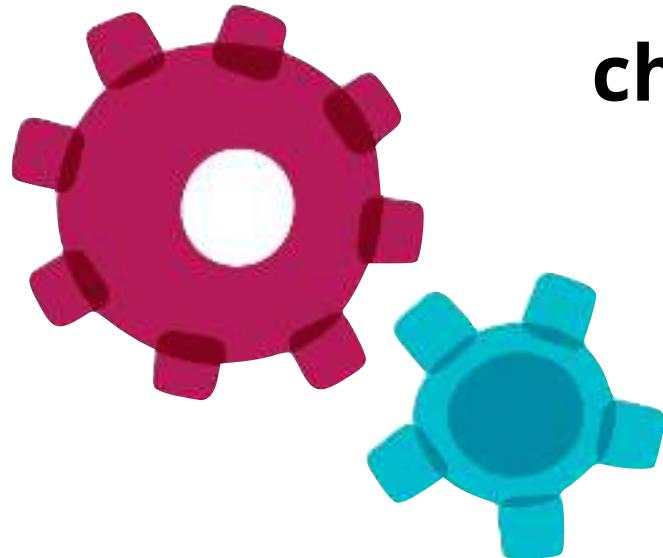
Round 1



<http://evolutionaryarchitecture.com/ffkatas/>

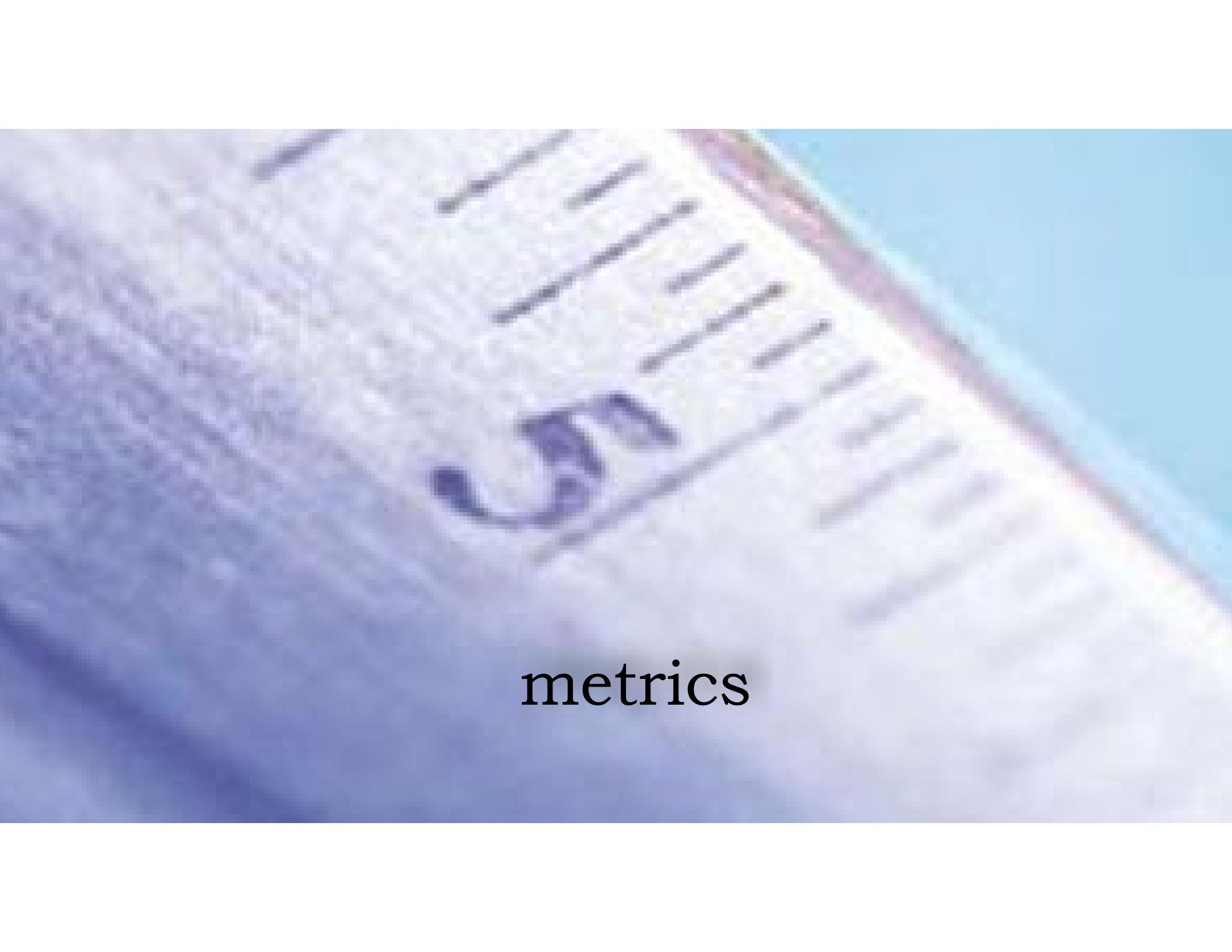
Implementing Fitness Functions

Protecting architectural
characteristics



Automating governance

maintainable?

The background of the slide features a faint, slightly blurred image of a document or ledger page. The document has several horizontal rows of text and some vertical columns. A small, dark logo or seal is visible in the center-left area of the document.

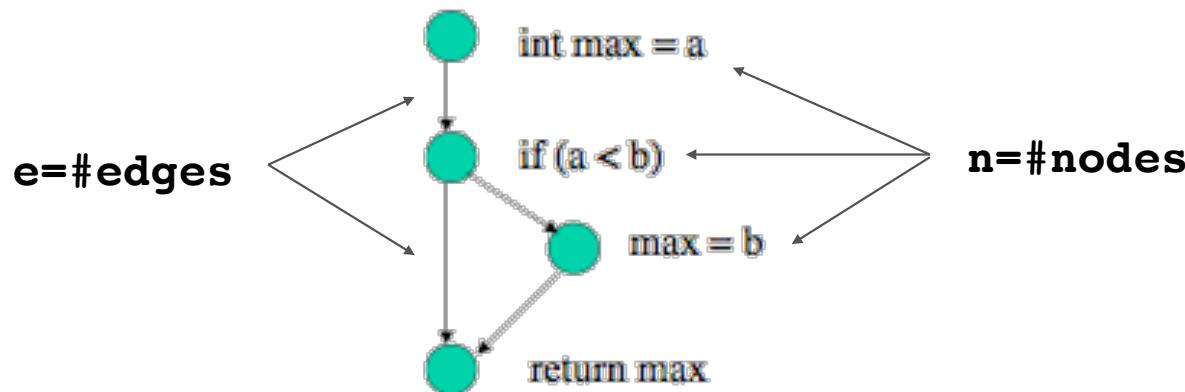
metrics

metrics and structural decay

cyclomatic complexity

provides a numeric value representing the complexity
of a function or method

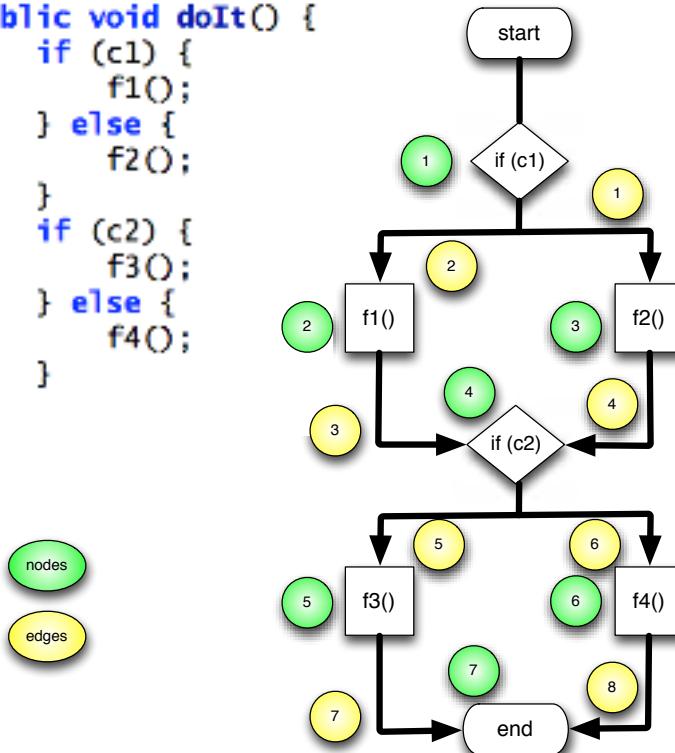
$$V(G) = e - n + 2$$



metrics and structural decay

cyclomatic complexity

```
public void doIt() {  
    if (c1) {  
        f1();  
    } else {  
        f2();  
    }  
    if (c2) {  
        f3();  
    } else {  
        f4();  
    }  
}
```



$$V(G) = e - n + 2$$

$$V(G) = 8 - 7 + 2 = 3$$

metrics and structural decay

core metrics

- ✓ number of classes per package
- ✓ number of lines of source code per package
- ✓ percent comments (range: 8-20)
- ✓ max complexity (1+num_paths thru method; range: 2-8)
- ✓ average complexity (range: 2.0 - 4.0)

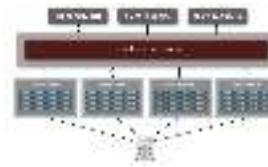
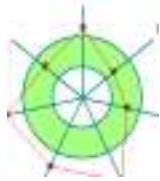
metrics and structural decay

Chidamber & Kemerer Metrics

- ✓ DIT (depth of inheritance tree)
- ✓ WMC (weighted methods/class; sum of CC)
- ✓ CE (efferent coupling count)
- ✓ CA (afferent coupling count)

metrics and structural decay

architecture characteristics mapping



component size

modularity

complexity / WMC

maintainability

coupling (CE, CA, CT)

testability

inheritance depth (DIT)

availability

percent comments

deployment

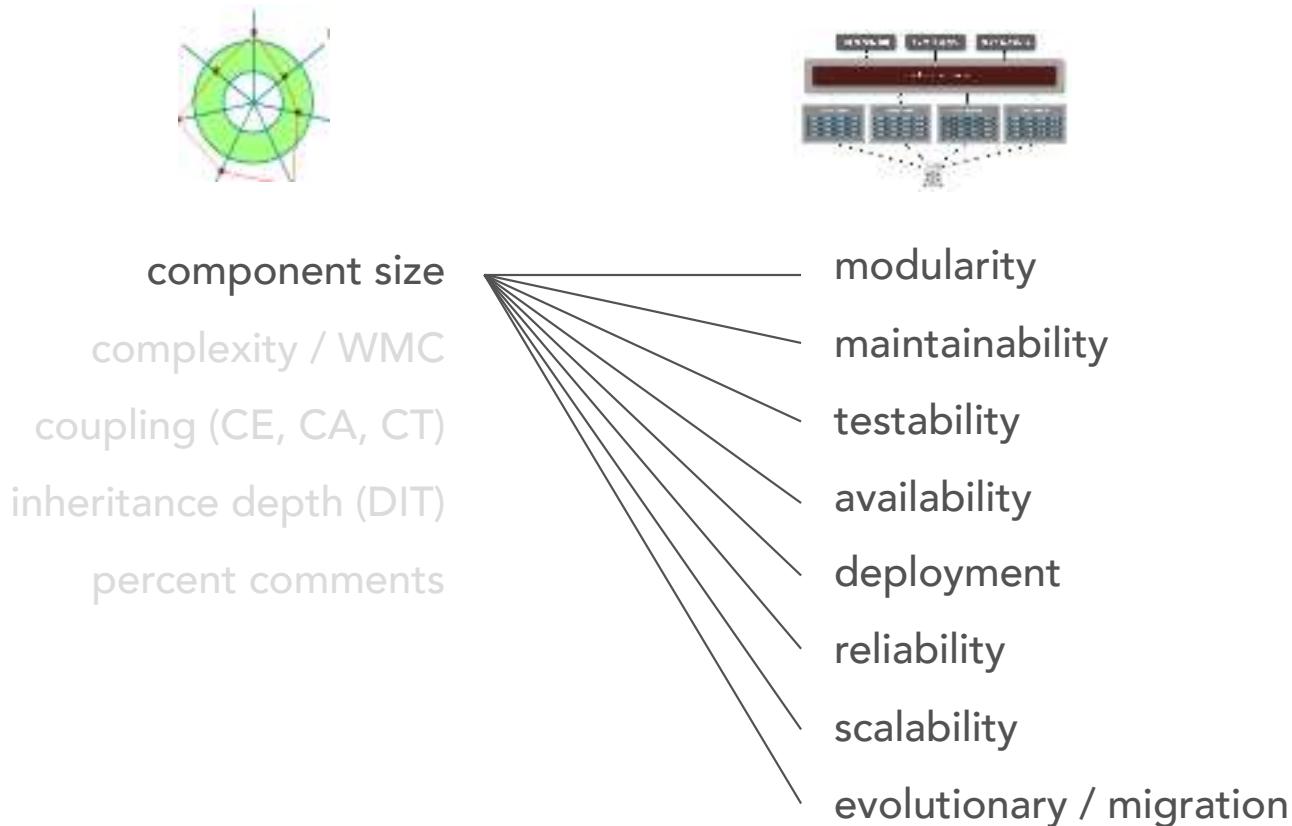
reliability

scalability

evolutionary / migration

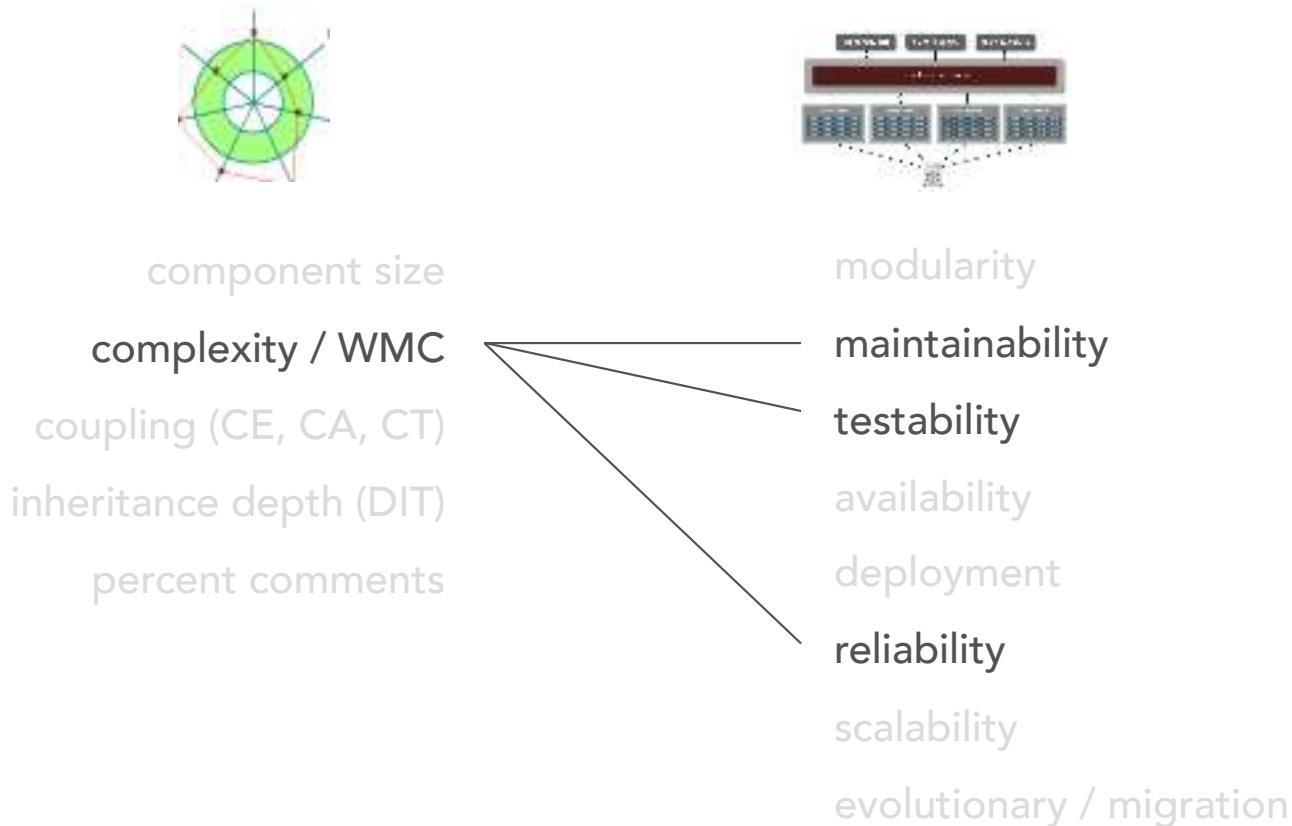
metrics and structural decay

architecture characteristics mapping



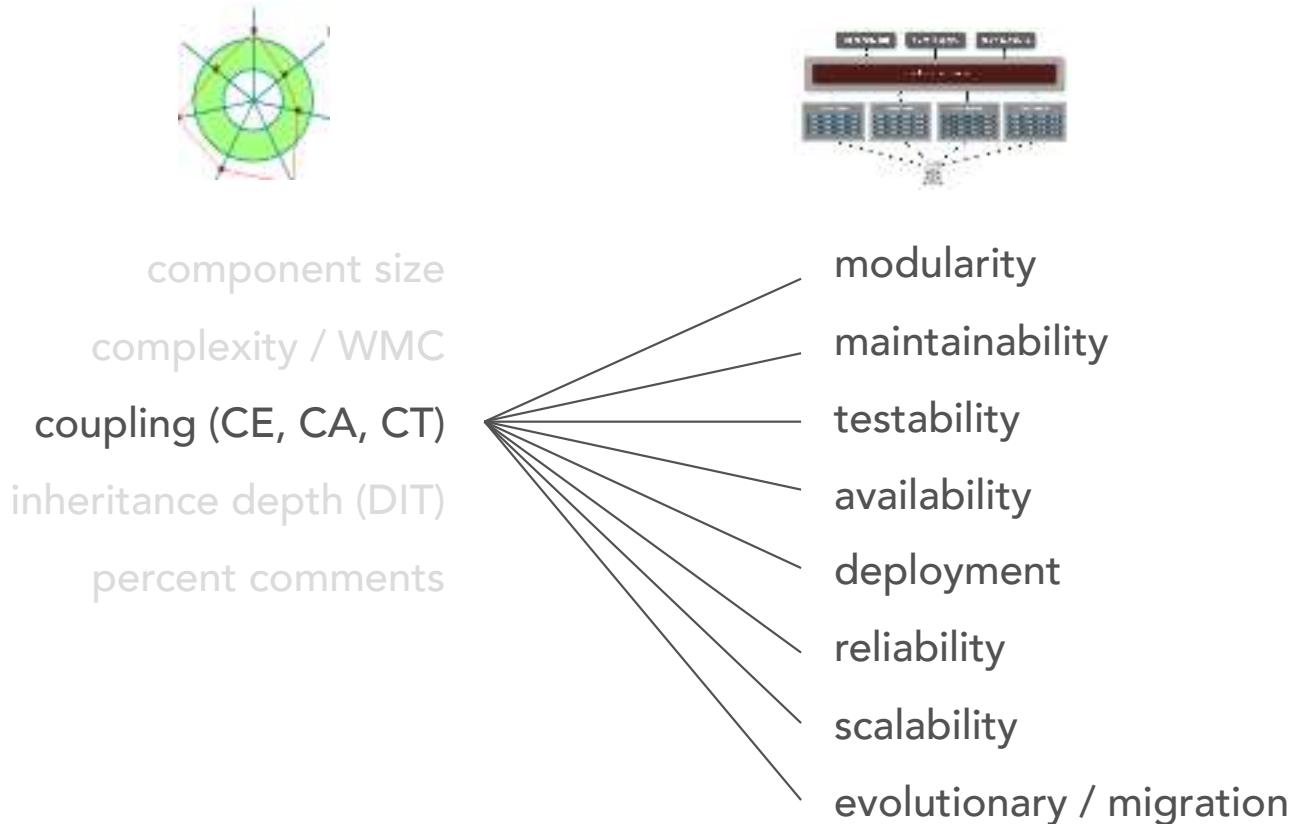
metrics and structural decay

architecture characteristics mapping



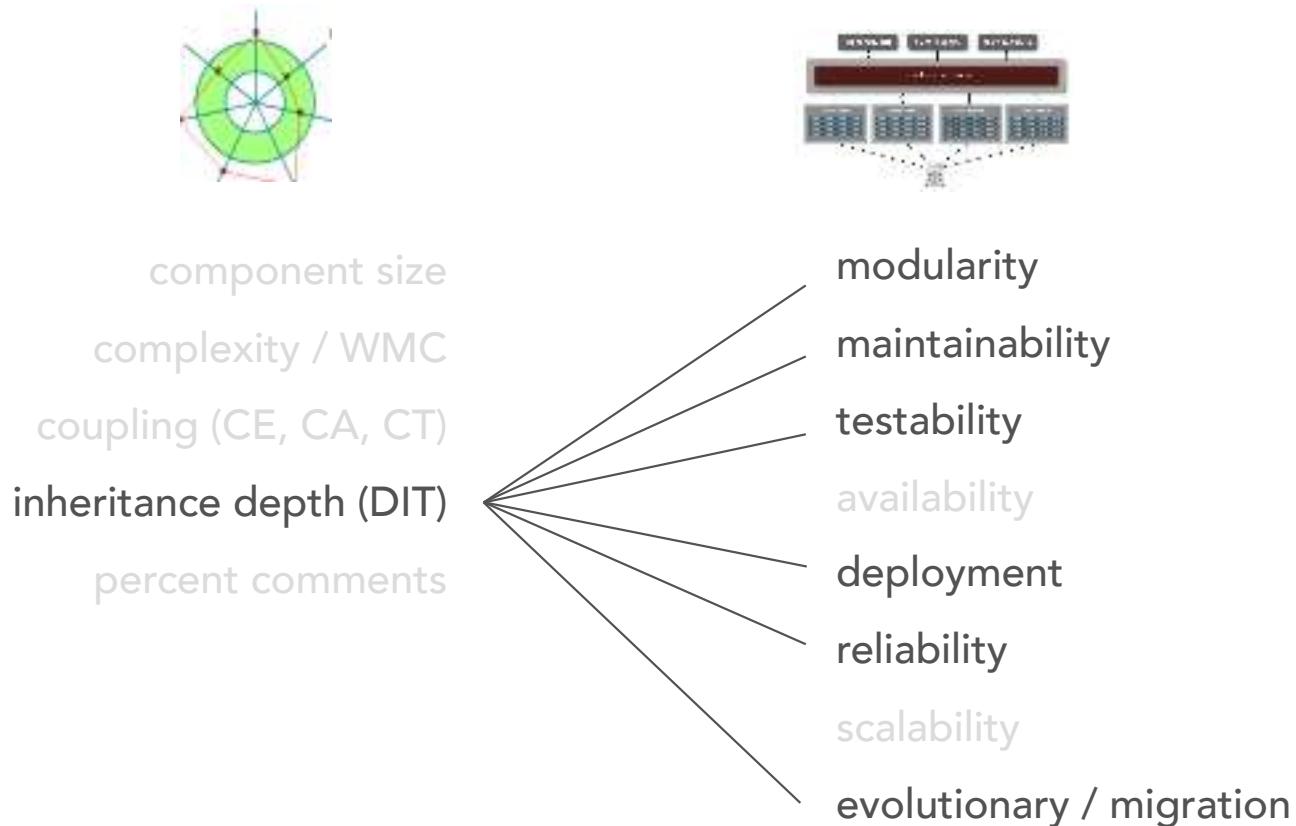
metrics and structural decay

architecture characteristics mapping



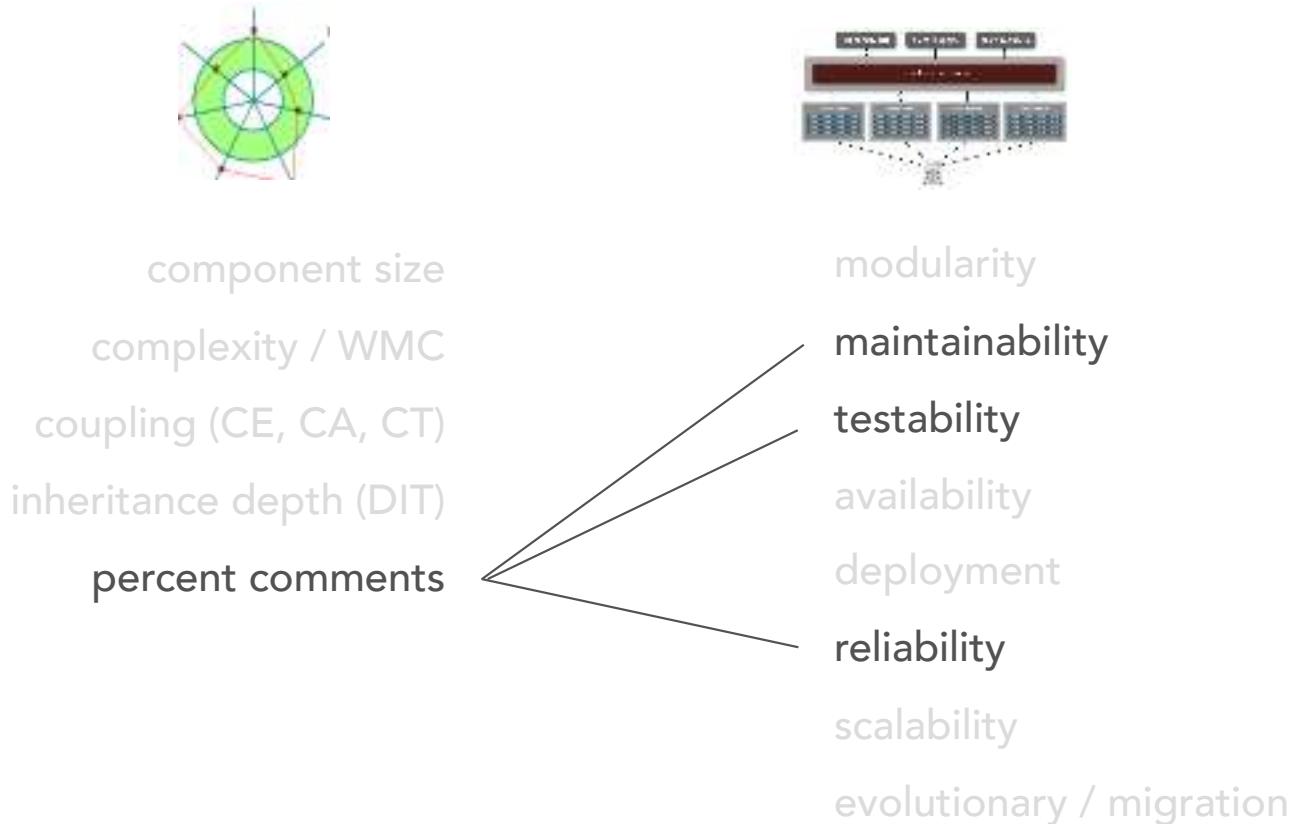
metrics and structural decay

architecture characteristics mapping

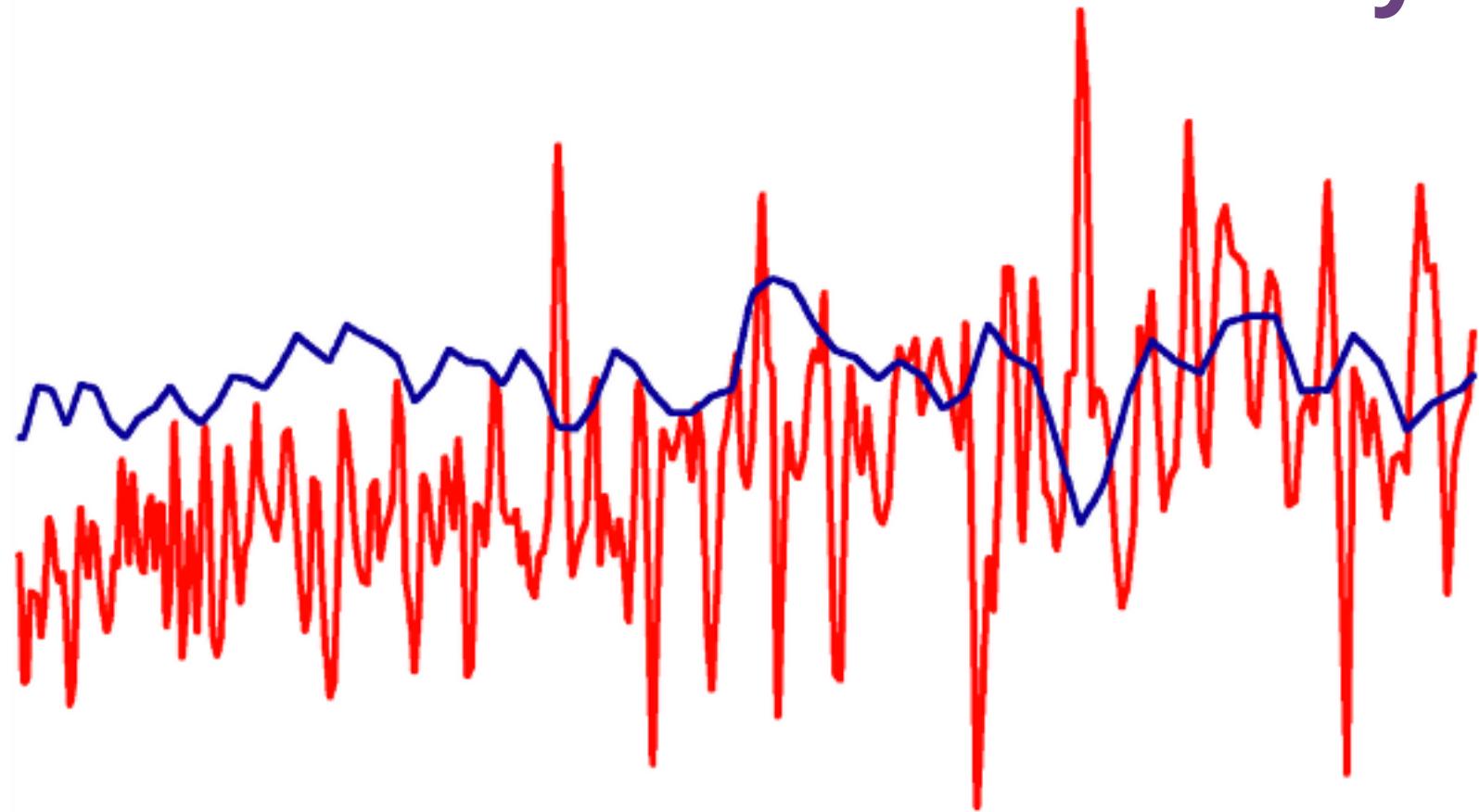


metrics and structural decay

architecture characteristics mapping



metrics and structural decay



architecture and structural decay

structural decay symptoms



complex and
error-prone
deployments



hard for new
people to learn
code base



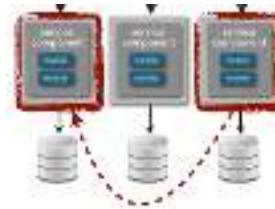
code changes
take longer than
expected

architecture and structural decay

general structural decay indicators



static
coupling



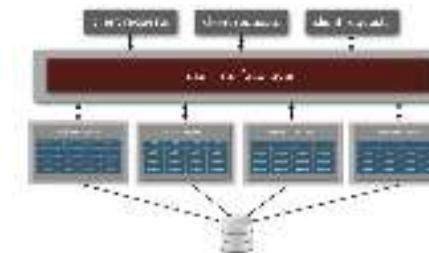
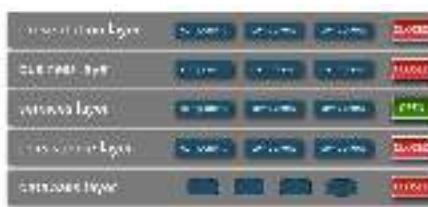
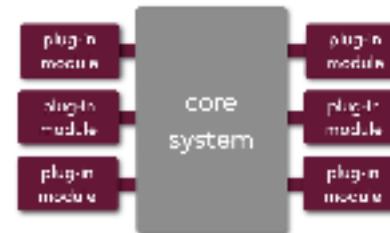
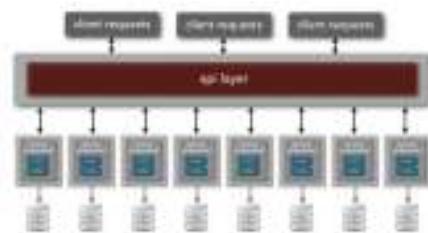
temporal
coupling



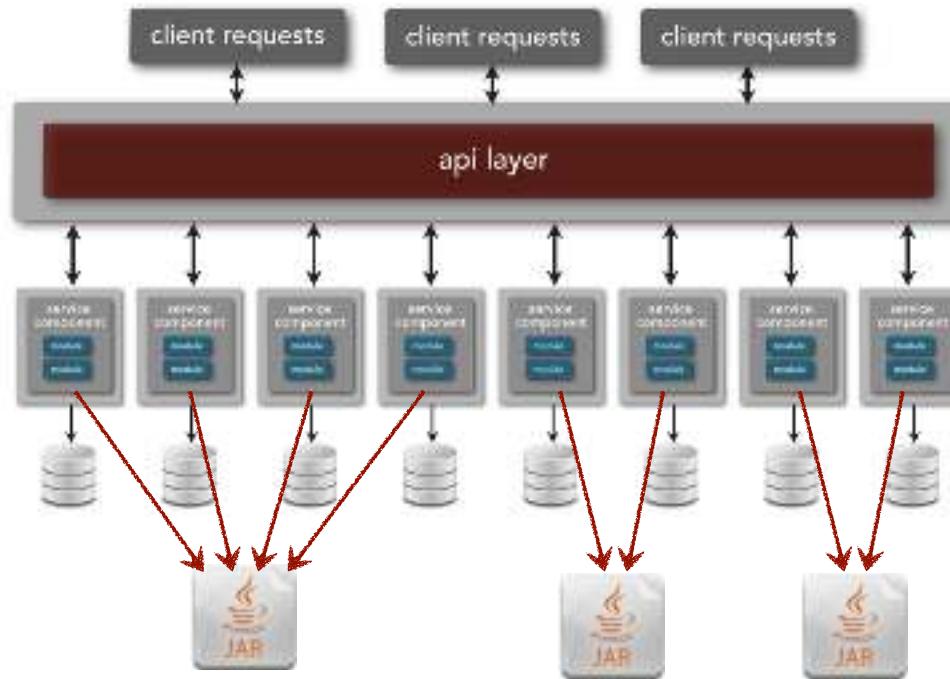
component
size

architecture and structural decay

pattern-specific decay indicators

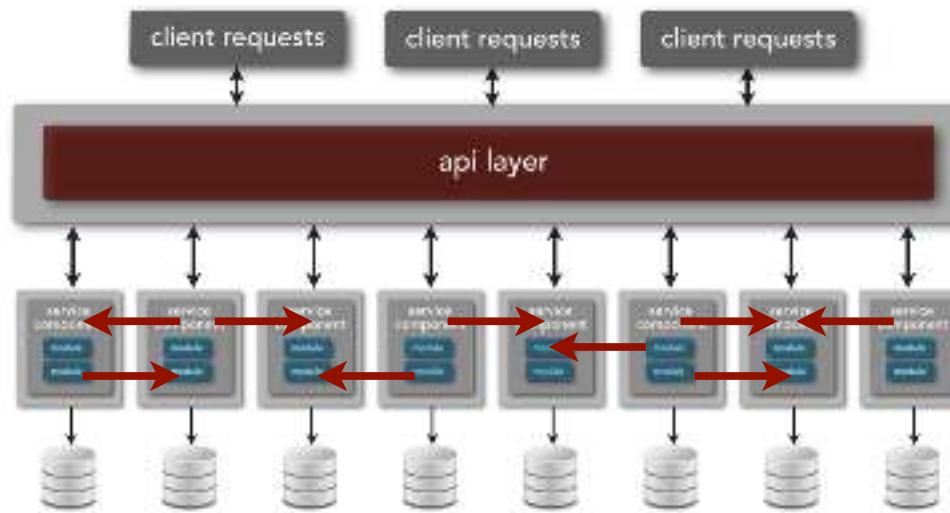


architecture and structural decay



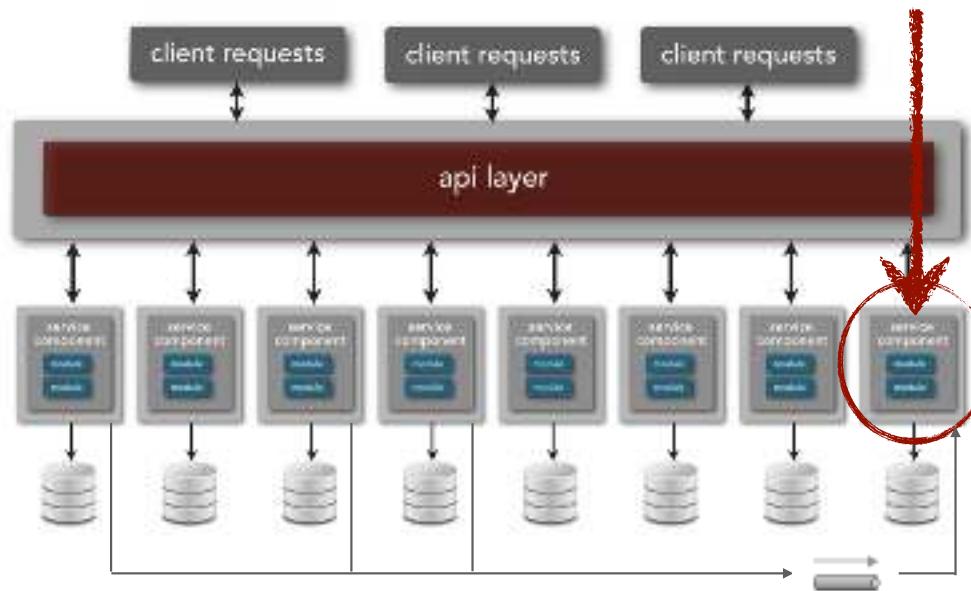
code dependencies between services
(shared libraries)

architecture and structural decay



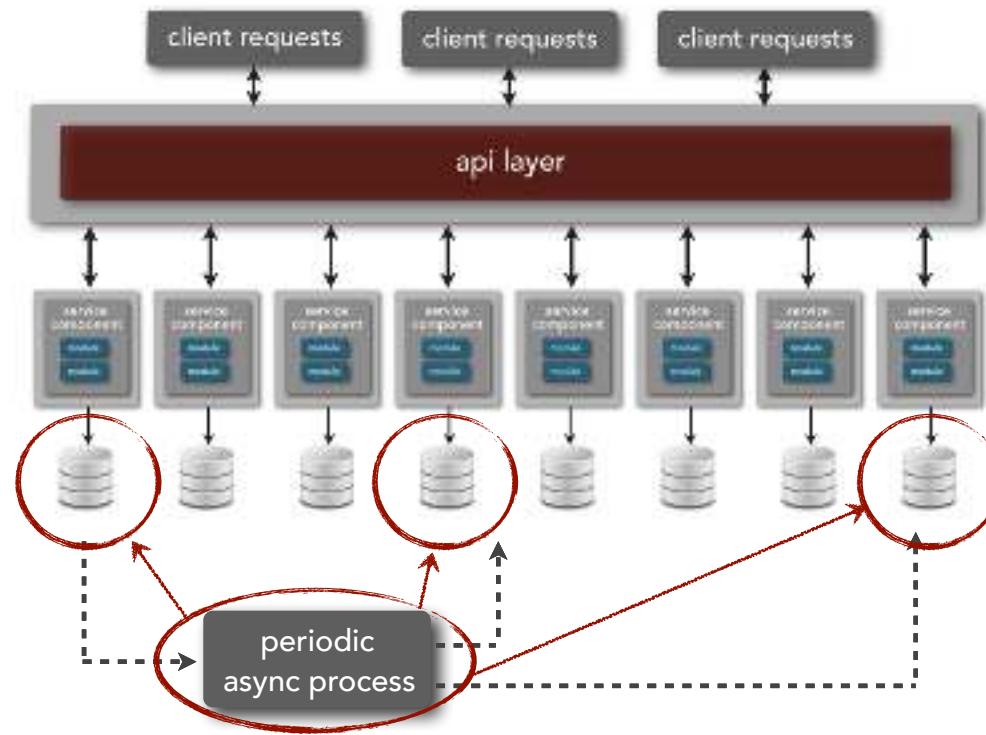
too much inter-service communication

architecture and structural decay



too many aggregation requests

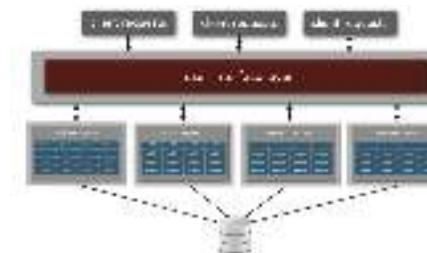
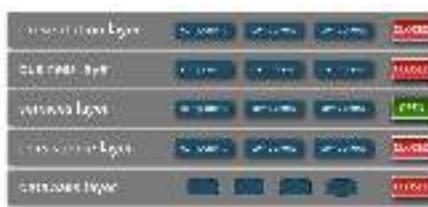
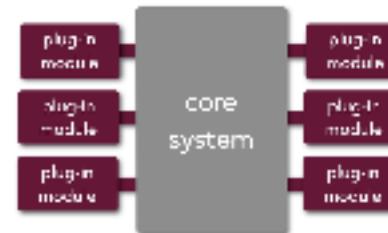
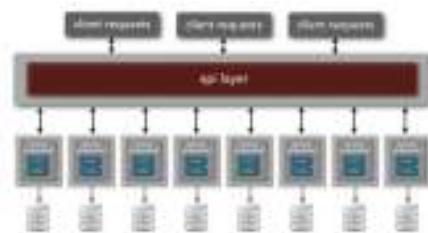
architecture and structural decay



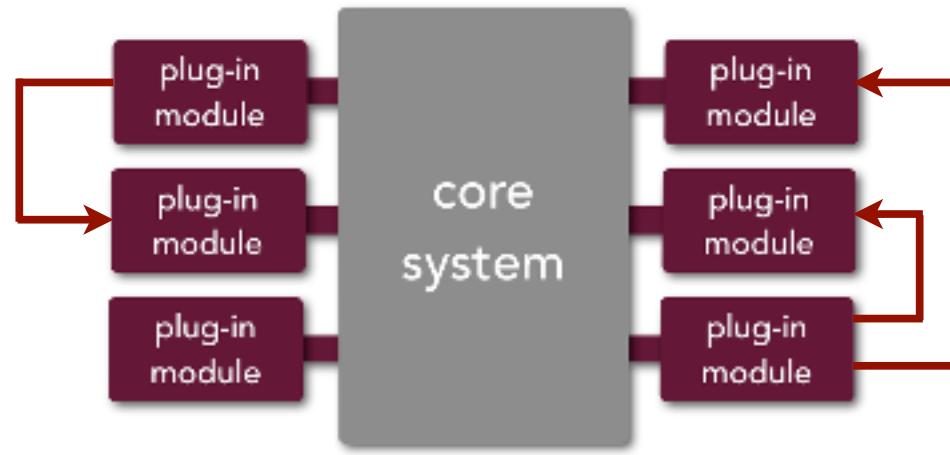
database coupling

architecture and structural decay

pattern-specific decay indicators

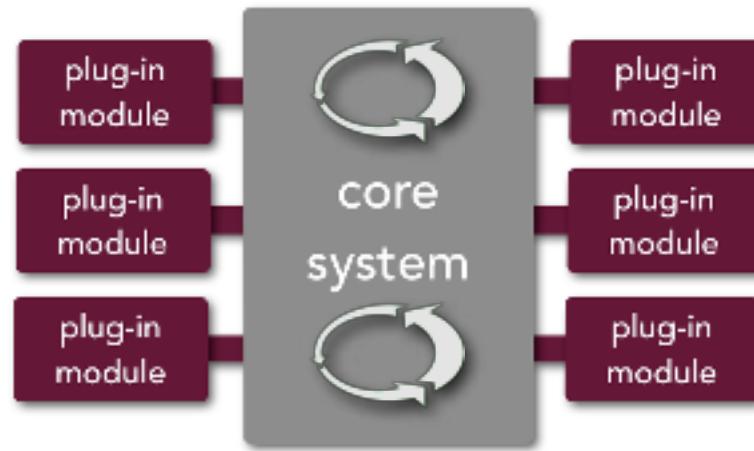


architecture and structural decay



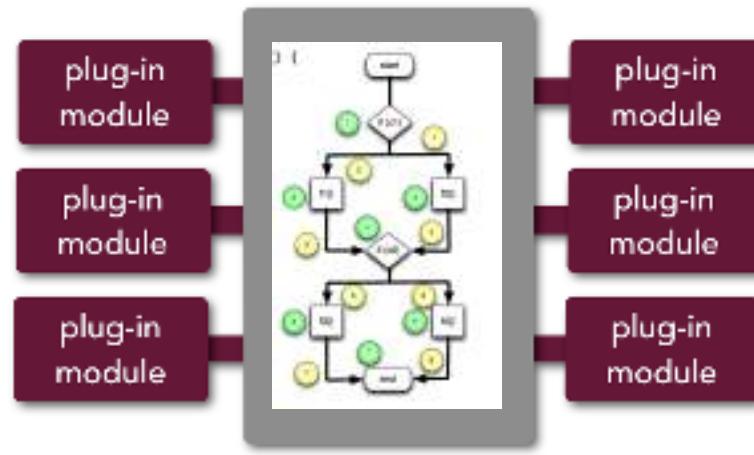
plug-in dependencies

architecture and structural decay



volatility in core system

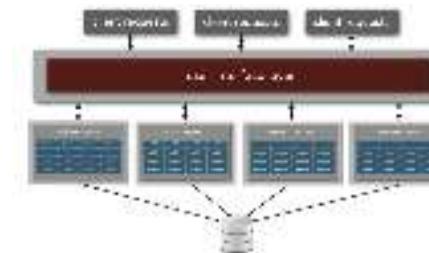
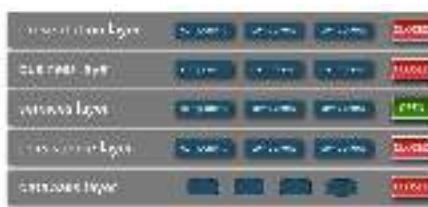
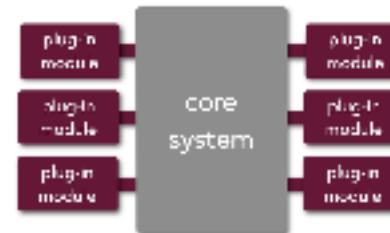
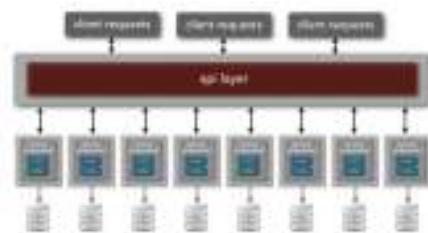
architecture and structural decay



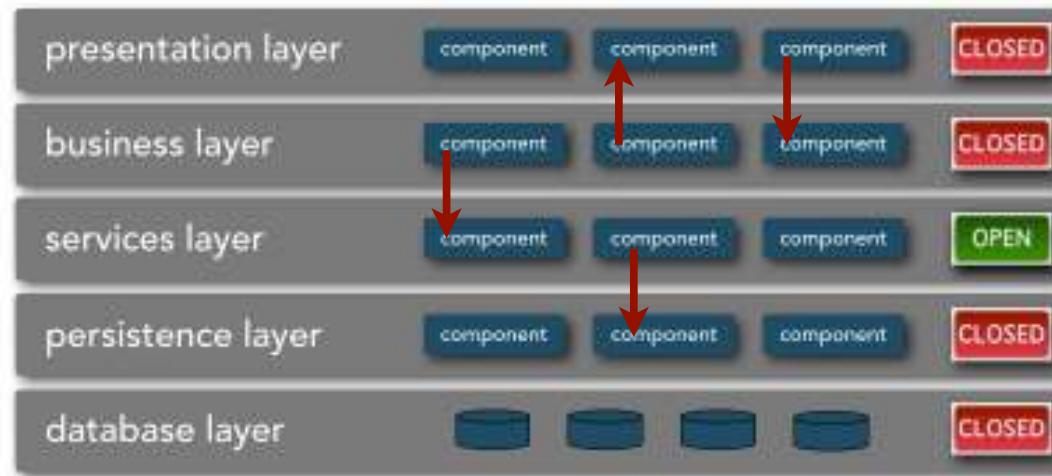
high complexity in core system

architecture and structural decay

pattern-specific decay indicators

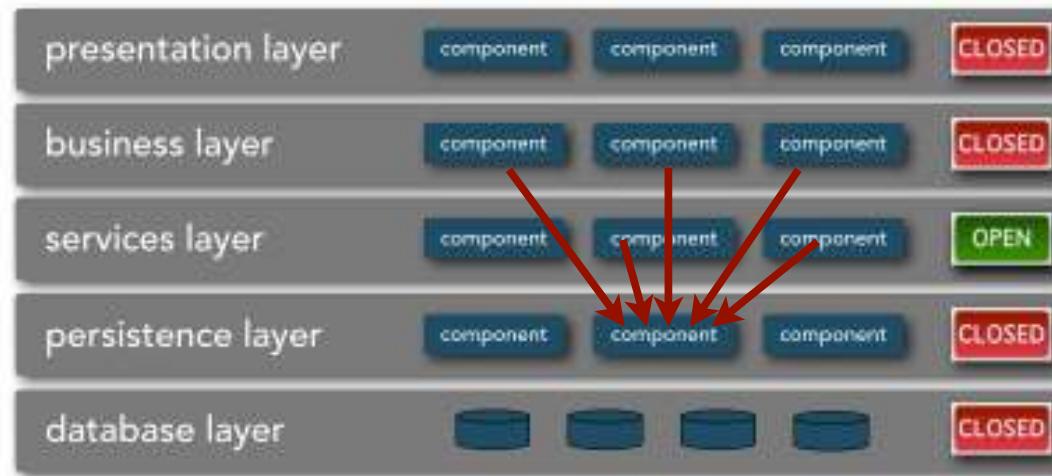


architecture and structural decay



static cross-domain dependencies

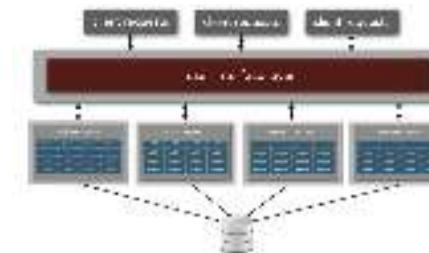
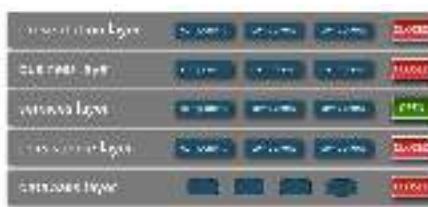
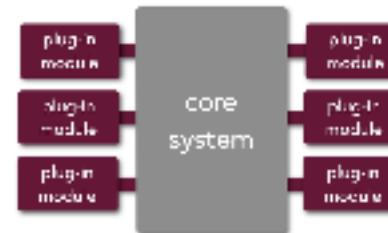
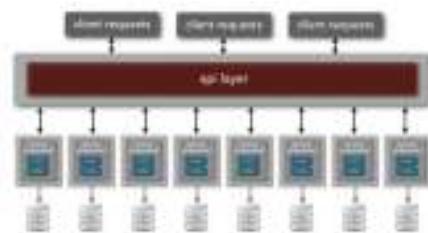
architecture and structural decay



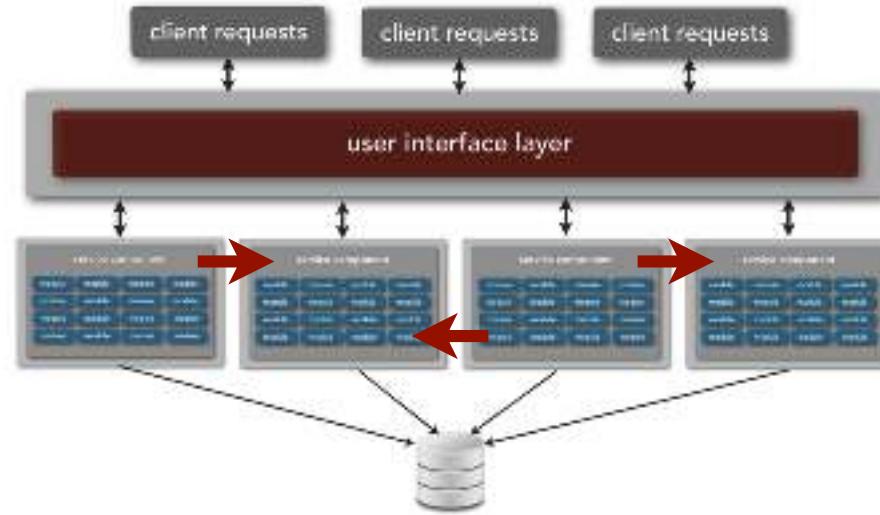
shared infrastructure components

architecture and structural decay

pattern-specific decay indicators

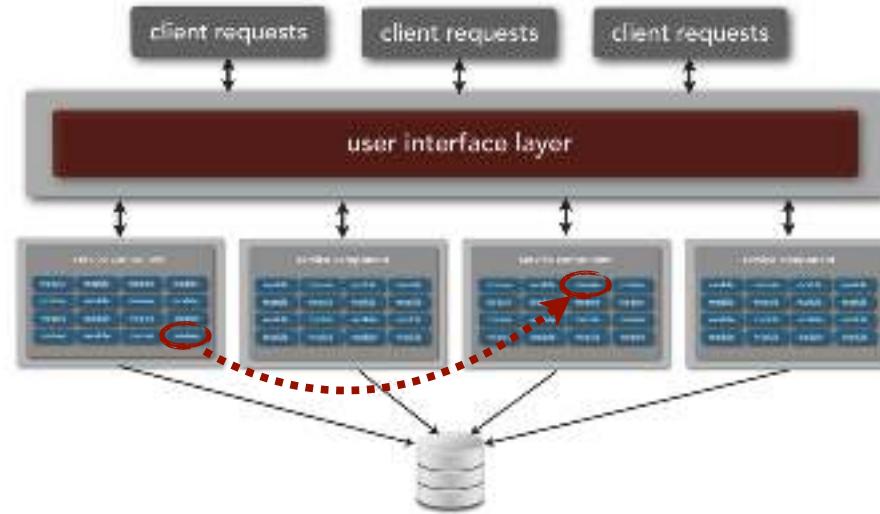


architecture and structural decay



inter-service communication

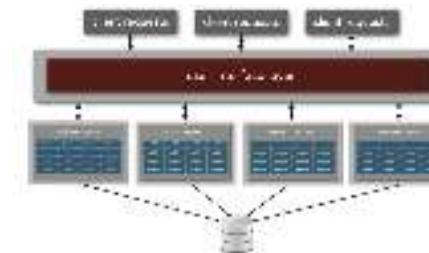
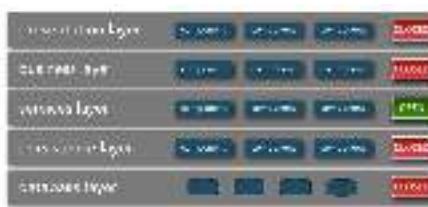
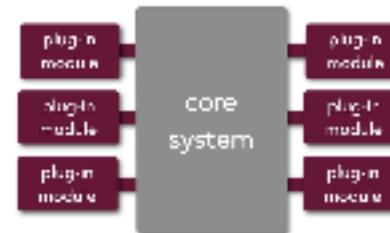
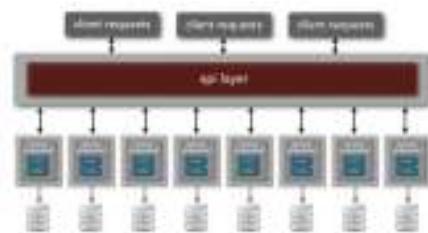
architecture and structural decay



temporal component coupling

architecture and structural decay

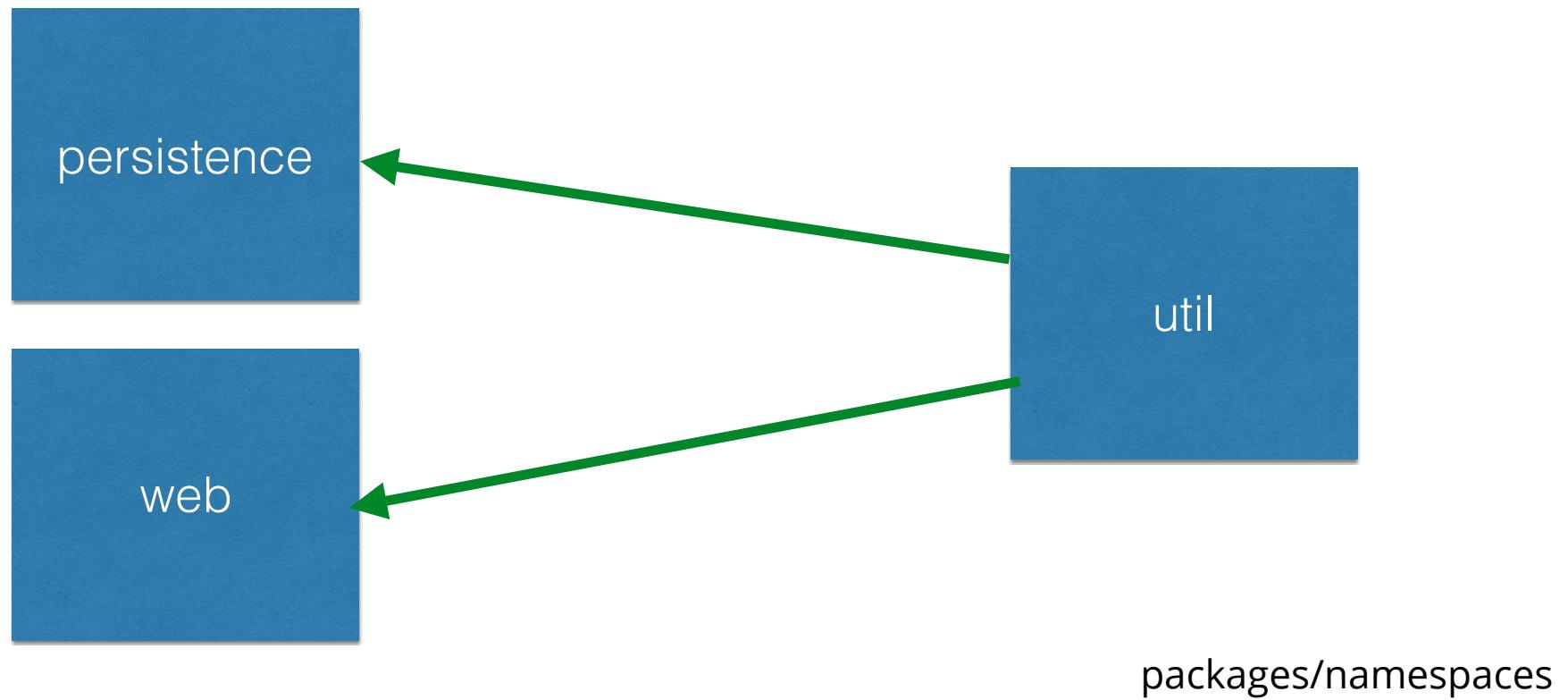
pattern-specific decay indicators



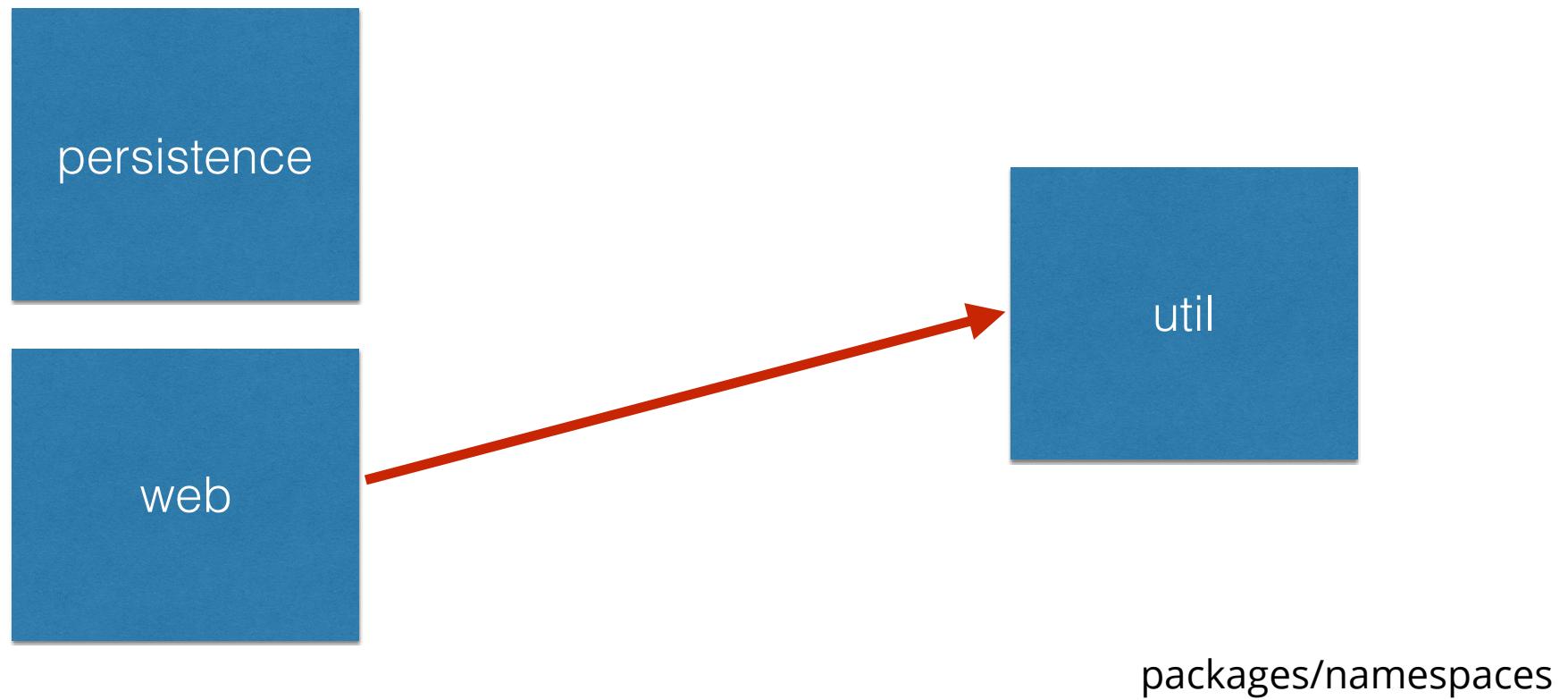
Testable

Many architecture characteristics are evaluable via testing.

Testable



Testable



Testable

```
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();

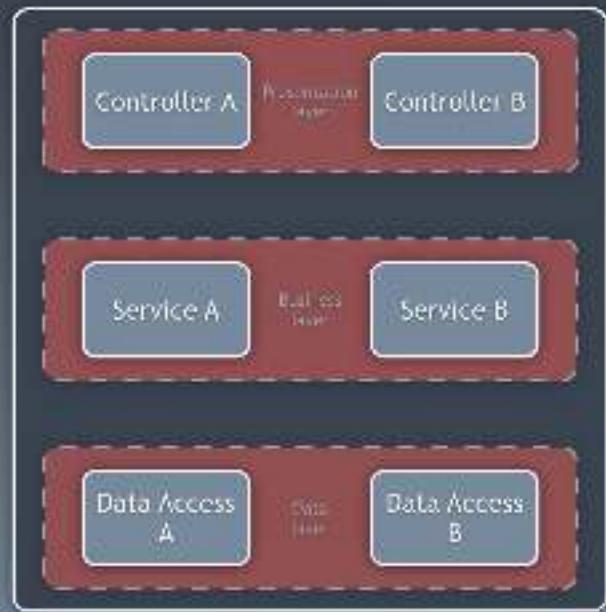
    assertEquals("Dependency mismatch",
                true, jdepend.dependencyMatch(constraint));
}
```

<https://github.com/clarkware/jdepend/>

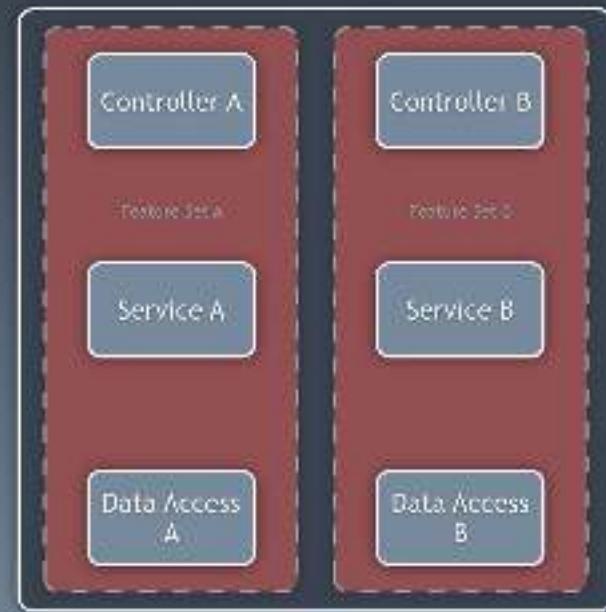
The screenshot shows a web browser window with the following details:

- Address Bar:** blog.jdriven.com
- Title Bar:** Implementing architectural fitness functions using Gradle, JUnit and code-assert
- Header:** jdriven
- Content Area:**
 - Section:** Implementing architectural fitness functions using Gradle, JUnit and code-assert
 - Text:** Posted on October 6, 2017 by Bob Jansen
 - Text:** Inspired by Neal Ford's presentation at our [Change is the Only Constant](#), I started experimenting with architectural fitness functions. An architectural fitness function provides an objective, integrity assessment of some architectural characteristic(s).
 - Text:** If you want to take a deeper dive into evolutionary architectures, including fitness functions, take a look at Neal's book: [Building Evolutionary Architectures: Supporting Change](#).
 - Text:** Neal's [slides](#) contained an example of verifying package dependencies from a Unit test using [JDepend](#).
 - Section:** Verifying code modularity
 - Text:** In this blog post we'll elaborate on that approach and create a Unit test that
- Right Sidebar:**
 - Search input field
 - Links:
 - » Driver
 - [blog.jdriven.com](#)
 - [search.jdriven.com](#)
 - Tag cloud:
 - Evolutionary Design
 - Spring Boot, Add Health
 - Full Information Triad
 - Dropwizard
 - Angular and Spring Boot
 - Getting Started
 - Unit testing via code coverage and mutation testing
 - Spring Boot Application
 - Dependency with Spring Cloud
 - Security – Part 1
 - Secure Boot – Dynamically control your code
 - Base64
 - Recent Posts:
 - « Action - Android
 - « Grouping Docker images
 - « Available Audiobooks – Using the Super cool Book API
 - « Implementing健忘性
 - « fitness functions using Gherkin
 - « JUnit 5 and code coverage
 - « Testing in the presence of external services

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



Package by layer (horizontal slicing)



Package by feature (vertical slicing)

```
public class VerifyPackageByLayerTest {

    @Test
    public void verifyPackageByLayer() {

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

        // Dependency rules for Packaging by Layer
        // NOTE: the classname should match the packagename
        class ComJdrivenFitnessPackagingByLayer extends DependencyRuler {

            // Rules for layer child packages
            // NOTE: they should match the name of the sub packages
            DependencyRule controller, service, repository;

            @Override
            public void defineRules() {
                // Our App classes depends on all subpackages because it constructs all of them
                base().mayUse(base().allSub());
                // Controllers may use Services
                controller.mayUse(service);
                // Services may use Repositories
                service.mayUse(repository);
            }
        }

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

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

```
public class ControllerA {  
    private final ServiceA serviceA;  
    private final RepositoryA repositoryA;  
  
    public ControllerA(ServiceA serviceA, RepositoryA repositoryA) {  
        this.serviceA = serviceA;  
        this.repositoryA = repositoryA;  
    }  
}
```

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

```
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());  
    }  
}
```

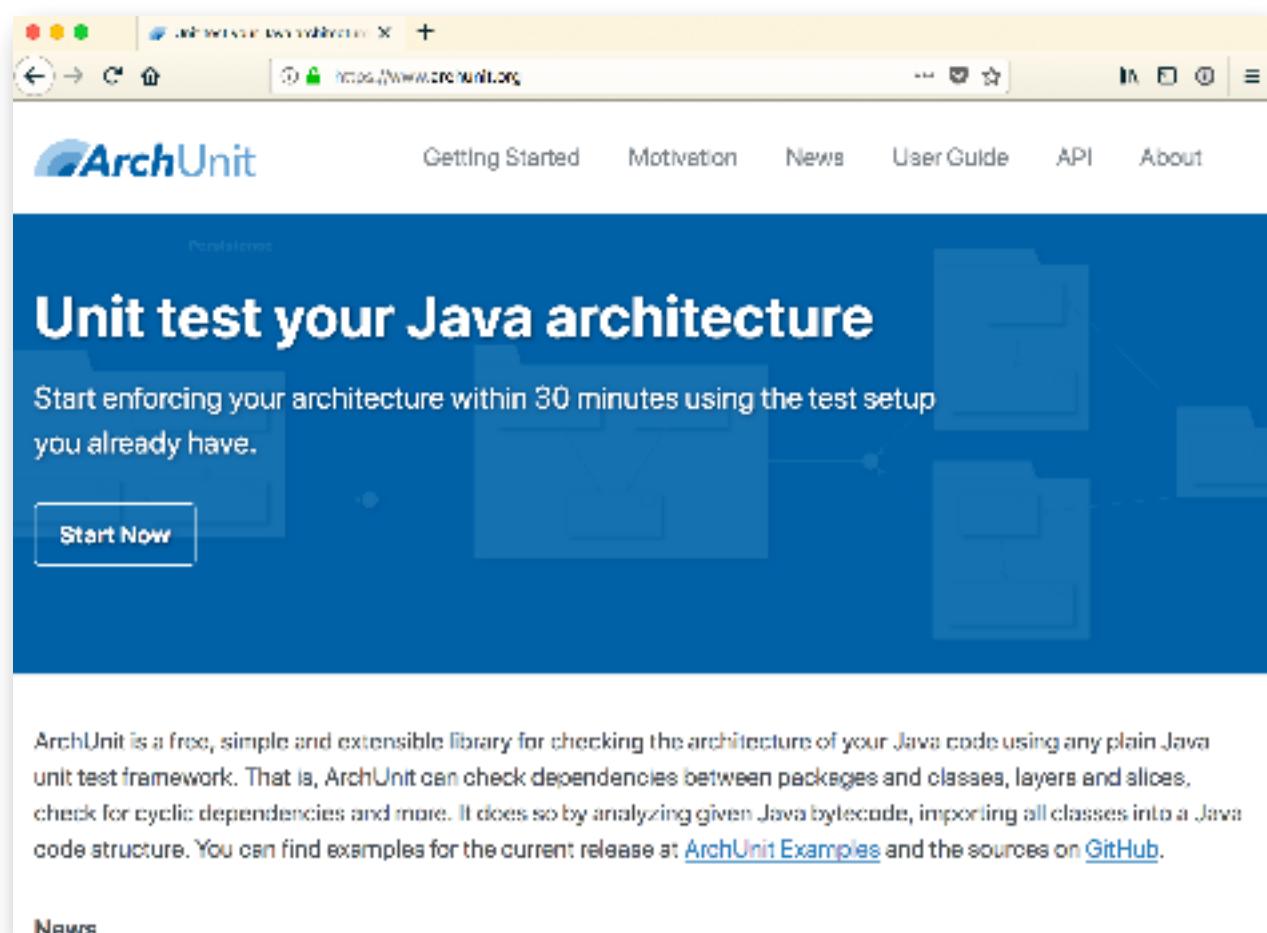
```
public class ControllerA {  
    private final ServiceA serviceA;  
    private final ServiceB serviceB;  
  
    public ControllerA(ServiceA serviceA, ServiceB serviceB) {  
        this.serviceA = serviceA;  
        this.serviceB = serviceB;  
    }  
}
```

```
java.lang.AssertionError:  
Expected: Comply with rules  
but: DENIED com.jdriven.fitness.packaging.by.feature.a ->  
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

```
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/>

```
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 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

ArchUnit

<https://www.archunit.org/>

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

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

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

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

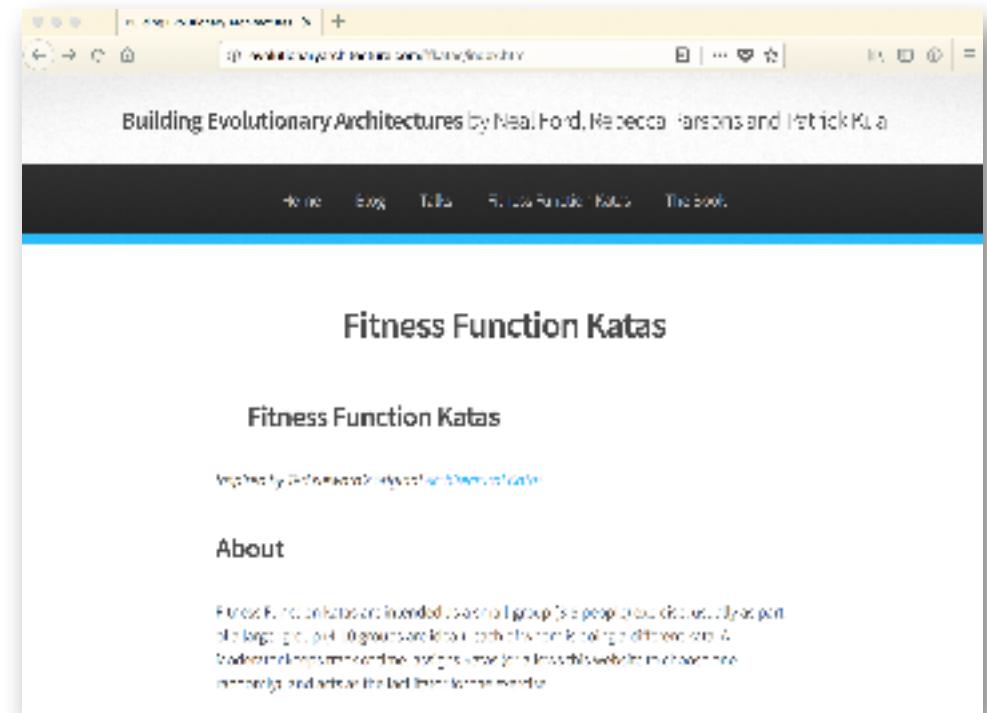
    DescribedPredicate<JavaCall<?>> targetIsIllegalConstructorOfThirdPartyClass =
        constructorCallOfThirdPartyClass.
            and(notFromWithinThirdPartyClass).
            and(notFromWorkaroundFactory);

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

governance

Fitness Function Katas

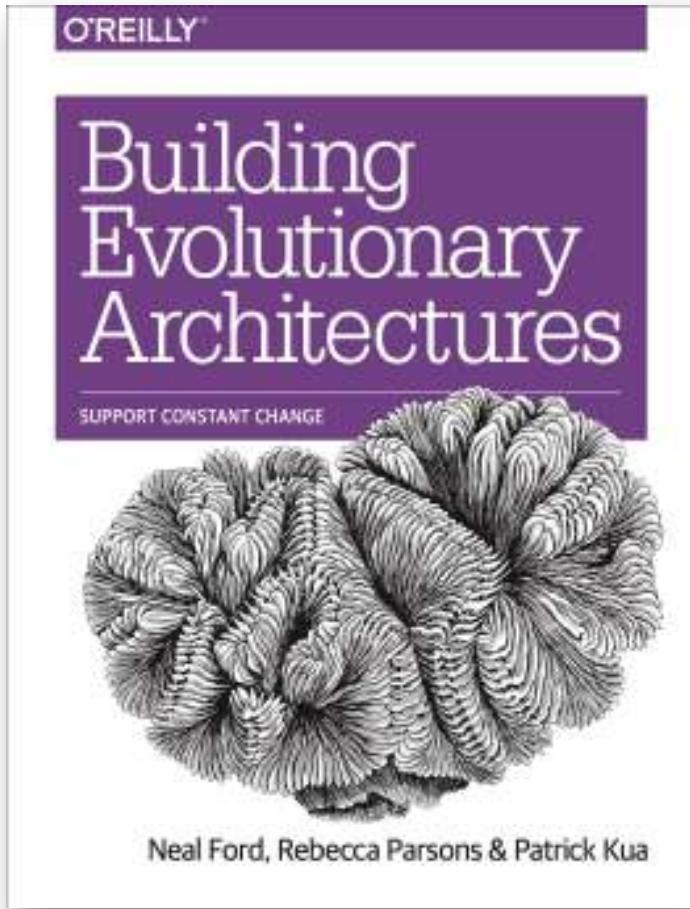
Round 2



<http://evolutionaryarchitecture.com/ffkatas/>

Building Evolutionary Architectures

SUPPORT CONSTANT CHANGE



 @neal4d
nealford.com



 @rebeccaparsons



 @patkua



Evolutionary Architecture

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



Deployment Pipelines



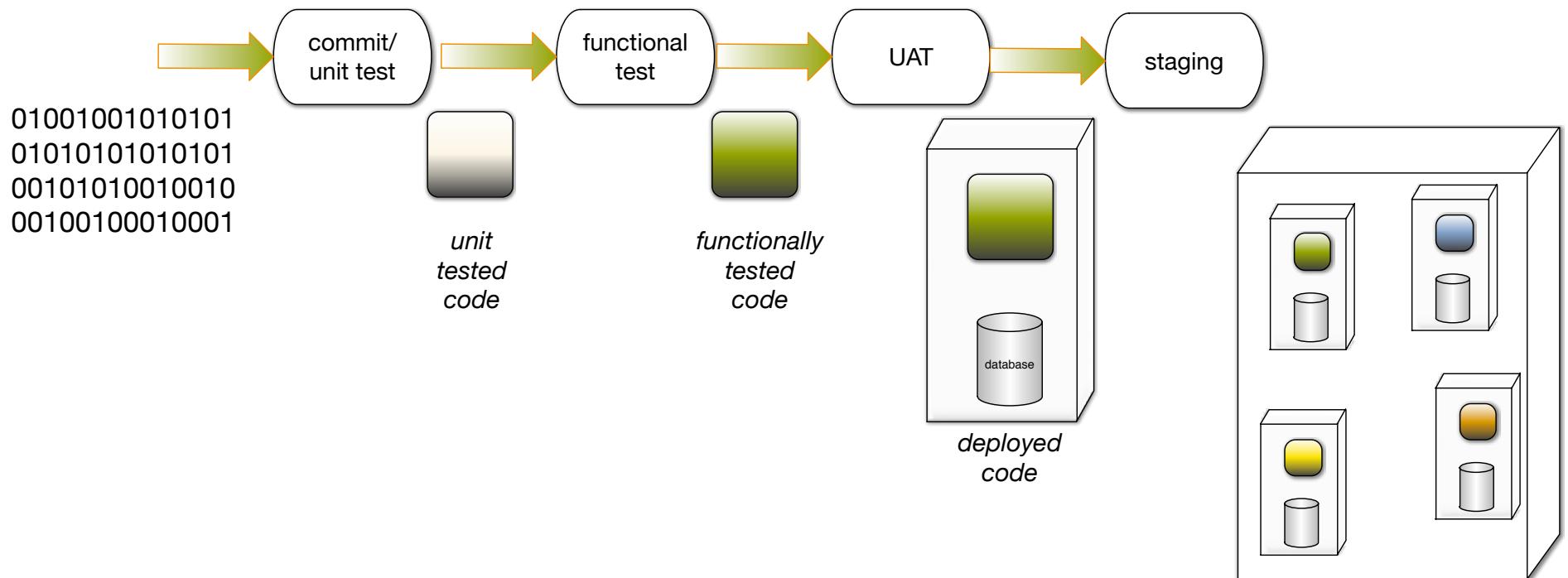
Continuous Integration

Fast, automated feedback on
the correctness of your
application every time there
is a change to code

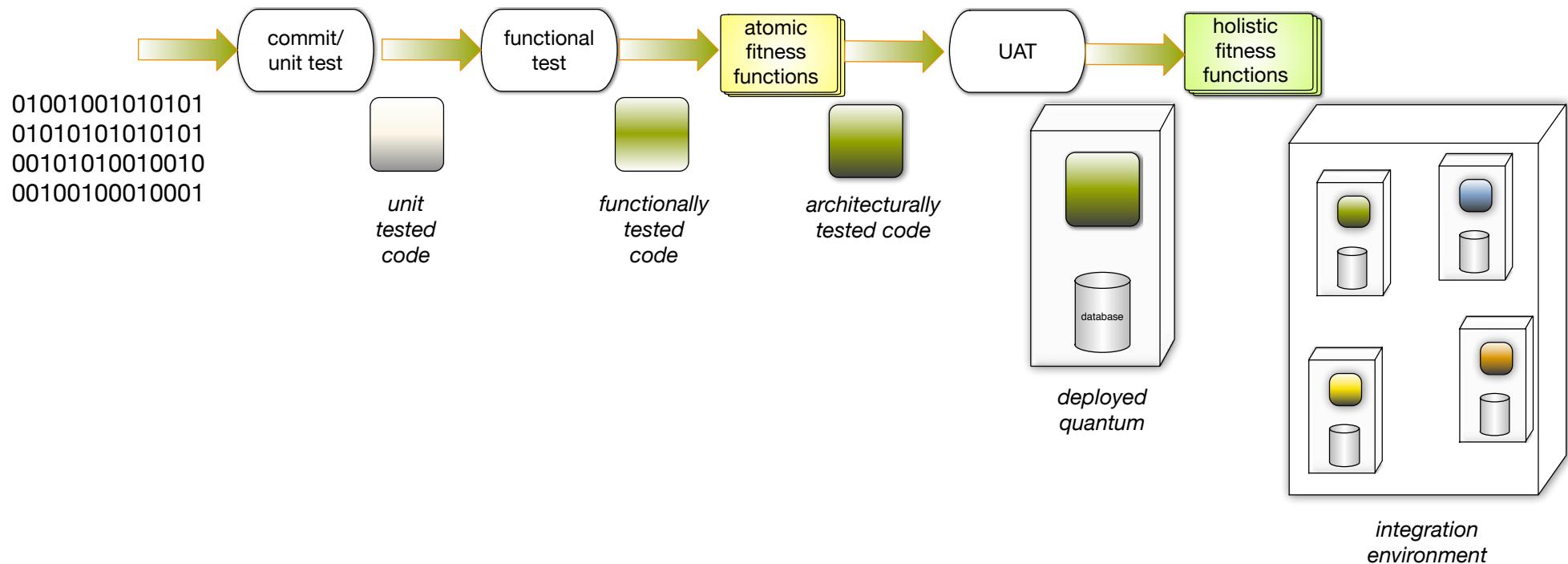
Deployment Pipeline

Fast, automated feedback
on the **production readiness**
of your application every
time there is a change — to
code, infrastructure or
configuration

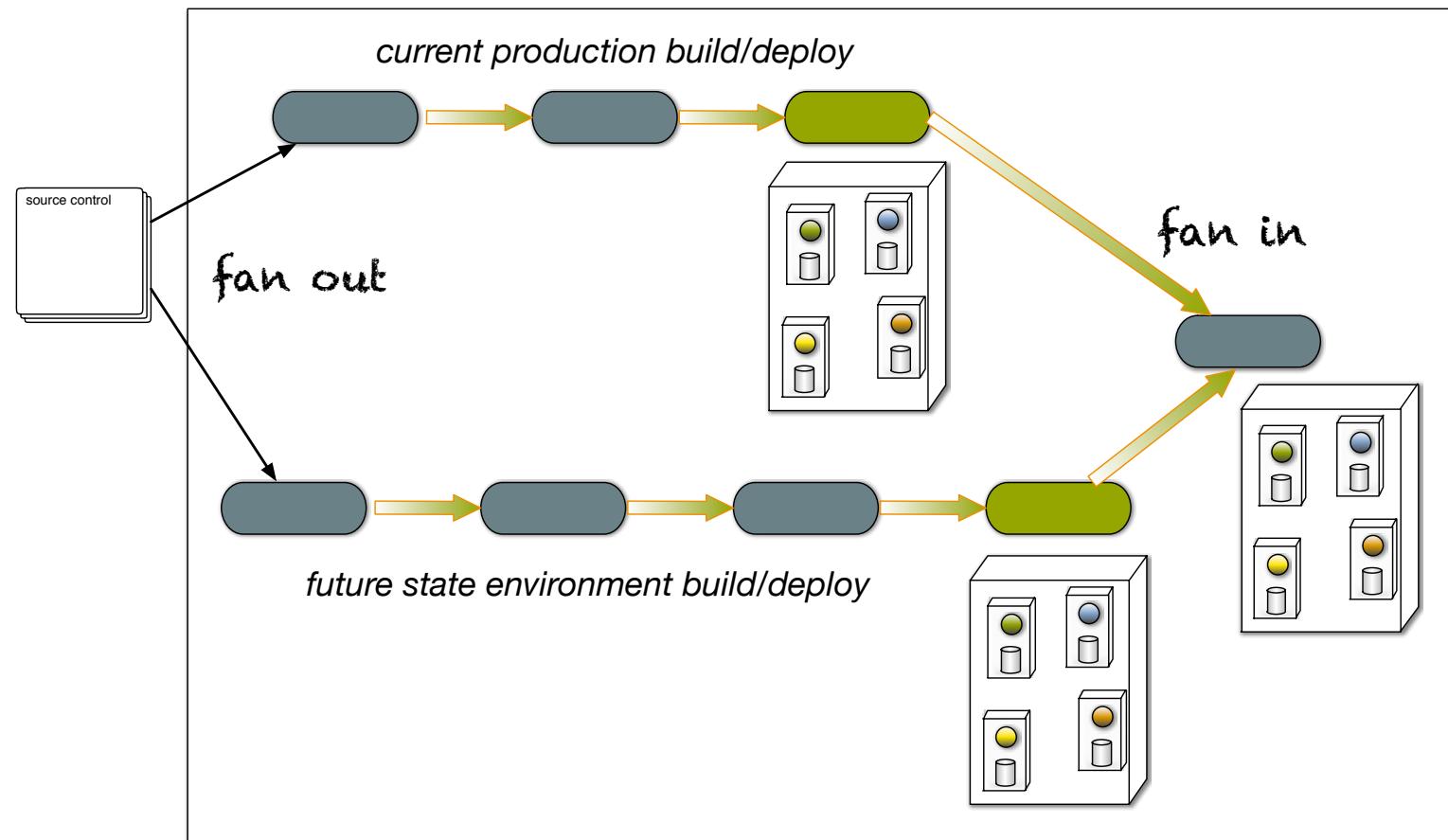
Deployment Pipeline



Deployment Pipeline + Fitness Functions



Deployment Pipeline Fan in/out



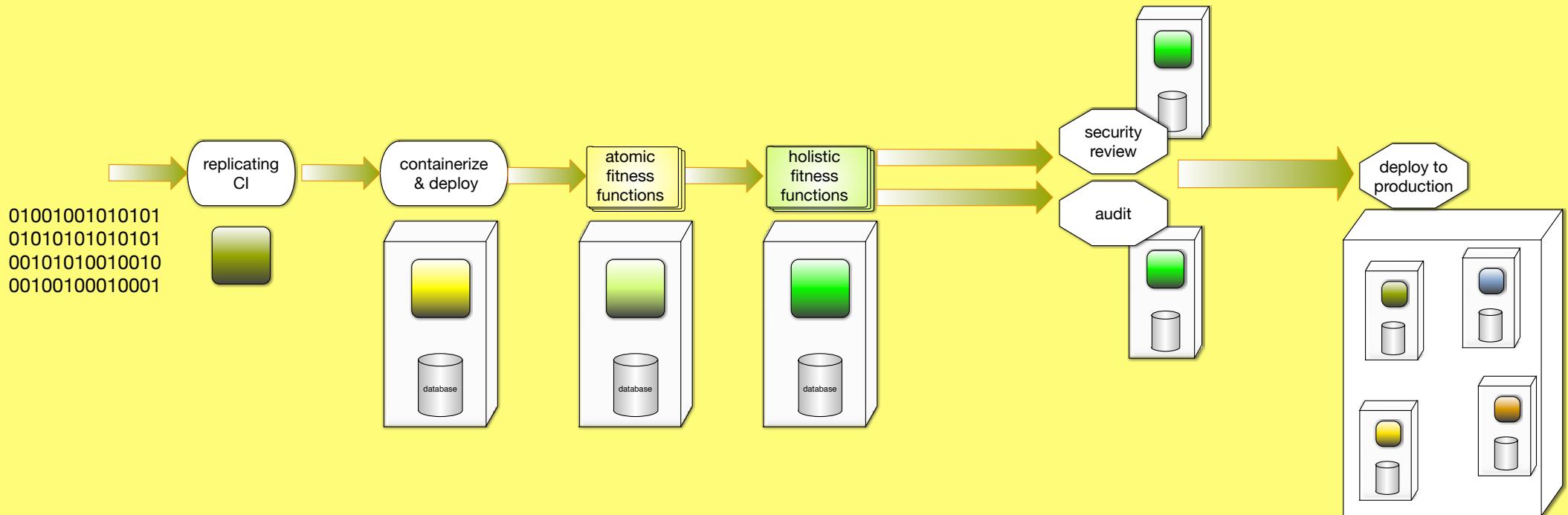
Adding fitness functions to Invoicing Application

- **Scalability:** While performance isn't a big concern for PenultimateWidgets, they handle invoicing details for several resellers, so the invoicing service must maintain availability service-level agreements.
- **Integration with other services:** Several other services in the PenultimateWidgets ecosystem use invoicing.
- **Security:** Invoicing means money, and security is always an ongoing concern.
- **Auditability:** Some state regulations require that changes to taxation code be verified by an independant accountant.

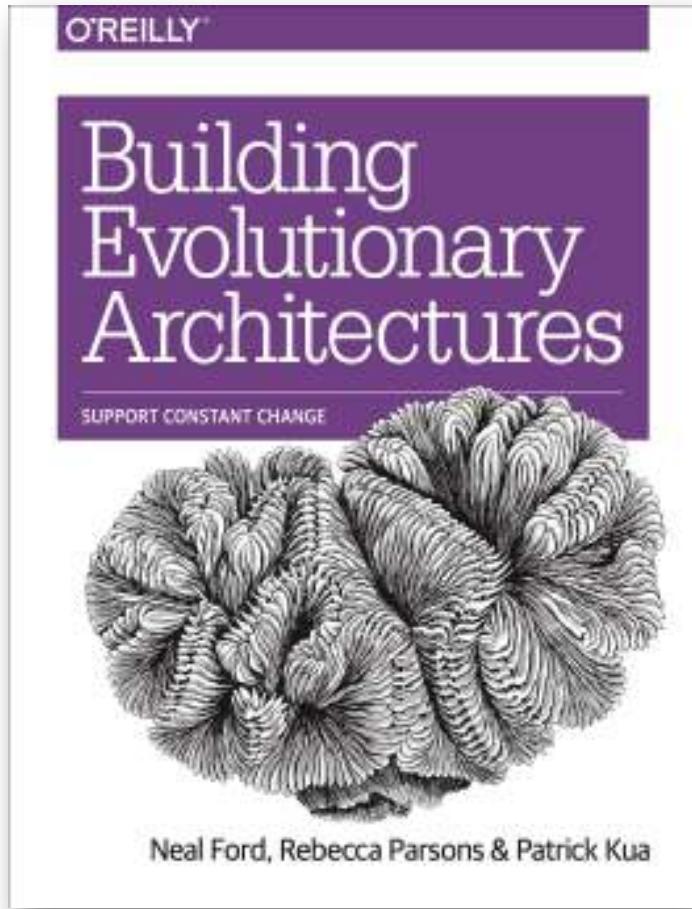




Deployment Pipeline



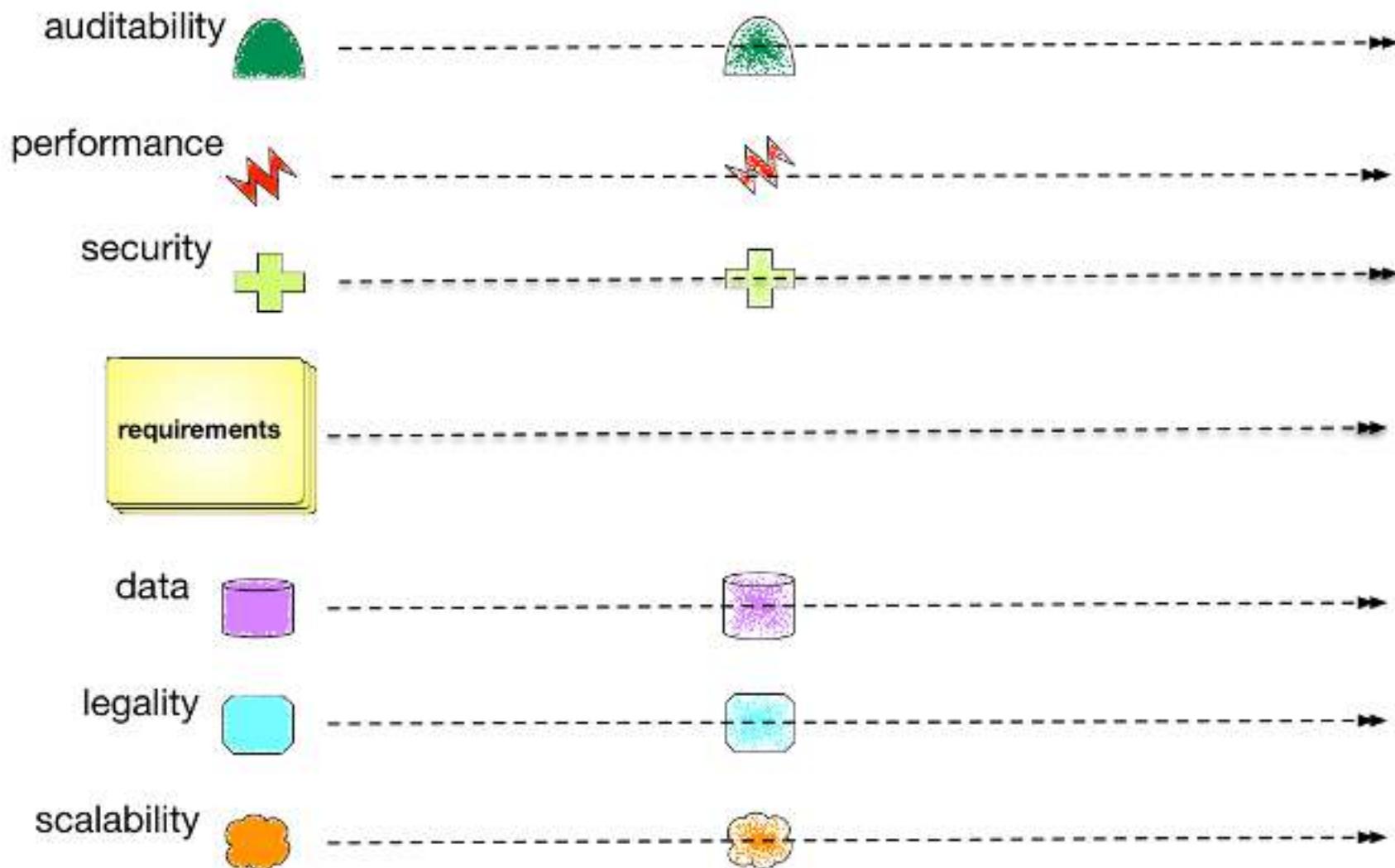
Building Evolutionary Architectures



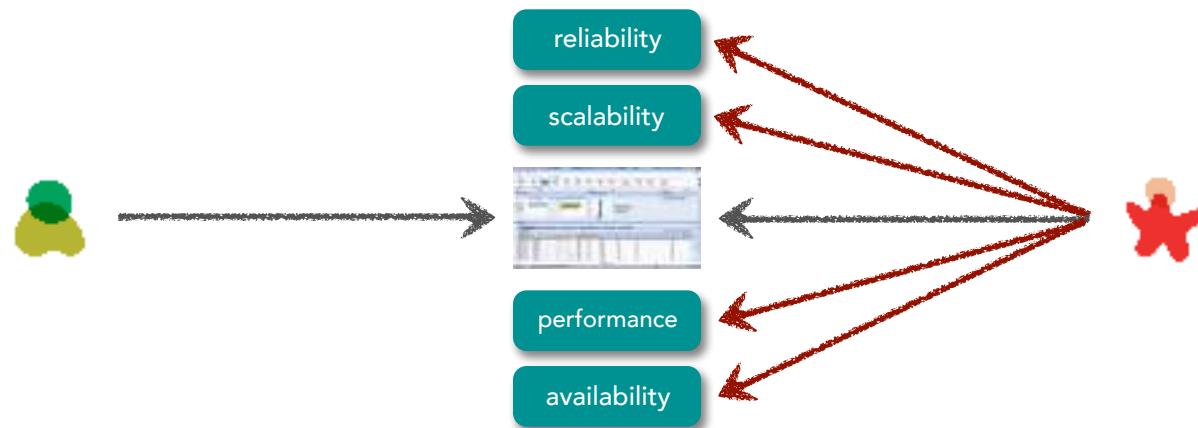
 @neal4d
nealford.com

EVOLUTIONARY
ARCHITECTURES





Architecture Characteristics



Architecture Characteristics

accessibility	evolvability	repeatability
accountability	extensibility	reproducibility
accuracy	failure transparency	resilience
adaptability	fault-tolerance	responsiveness
administrability	fidelity	reusability
affordability	flexibility	robustness
agility	inspectability	safety
auditability	installability	scalability
autonomy	integrity	seamlessness
availability	interchangeability	self-sustainability
compatibility	interoperability	serviceability
composability	learnability	supportability
configurability	maintainability	securability
correctness	manageability	simplicity
credibility	mobility	stability
customizability	modifiability	standards compliance
debugability	modularity	survivability
degradability	operability	sustainability
determinability	orthogonality	tailorability
demonstrability	portability	testability
dependability	precision	timeliness
deployability	predictability	traceability
discoverability	process capabilities	transparency
distributability	producibility	ubiquity
durability	provability	understandability
effectiveness	recoverability	upgradability
efficiency	relevance	usability
	reliability	

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

Architecture Characteristics



"our business is constantly changing
to meet new demands of the
marketplace"



extensibility, maintainability, agility,
modularity

Architecture Characteristics



"due to new regulatory requirements,
it is imperative that we complete end-
of-day processing in time"



performance, scalability, availability,
reliability

Architecture Characteristics



"we need faster time to market to remain competitive"

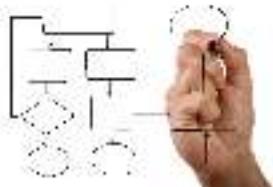


maintainability, agility, modularity,
deployability, testability

Architecture Characteristics



"our plan is to engage heavily in mergers and acquisitions in the next three years"

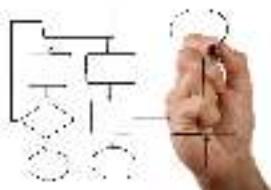


scalability, extensibility, openness,
standards-based, agility, modularity

Architecture Characteristics



“we have a very tight timeframe and budget for this project”

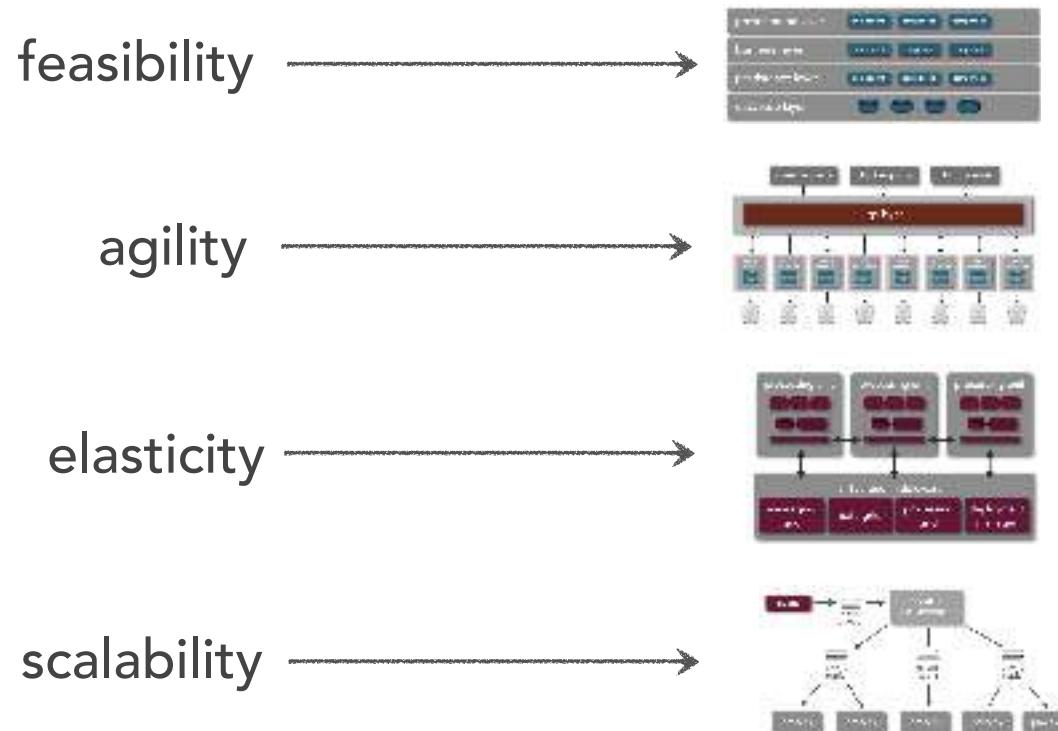


feasibility

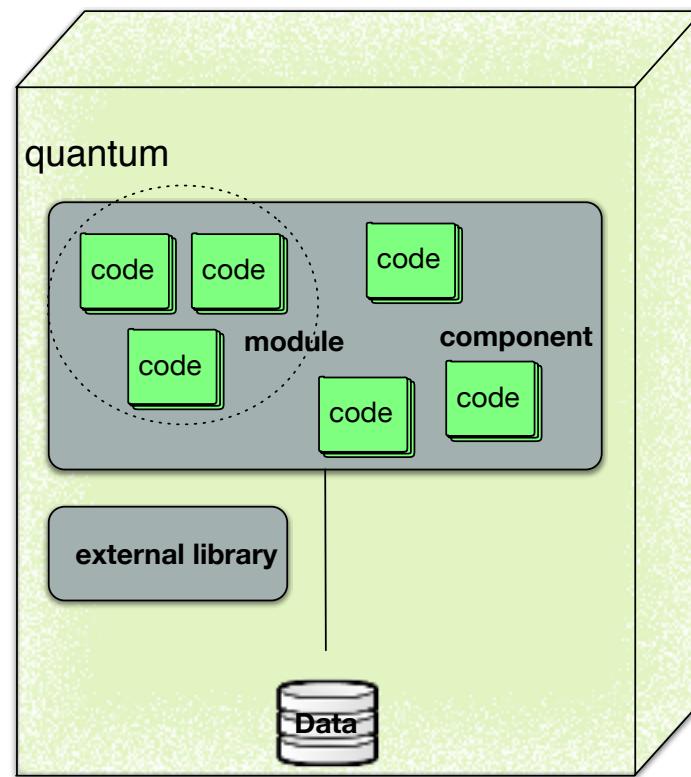




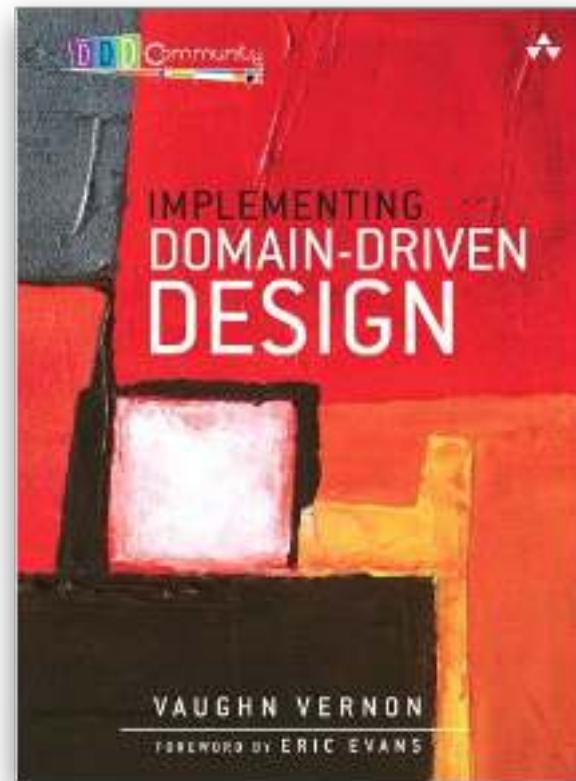
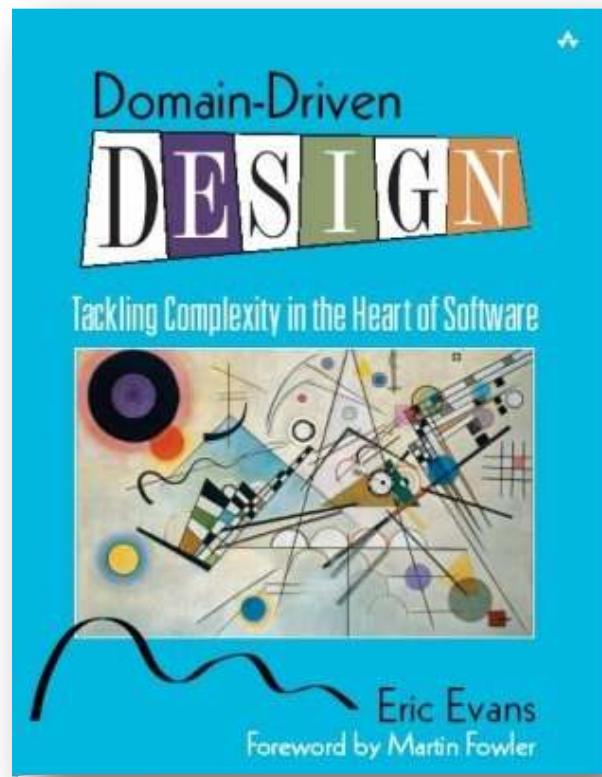
Architecture Characteristics



Modularity



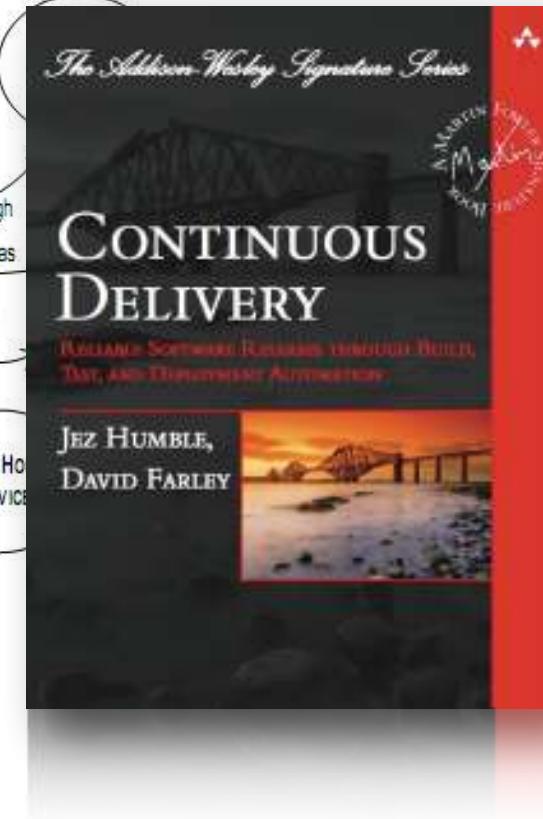
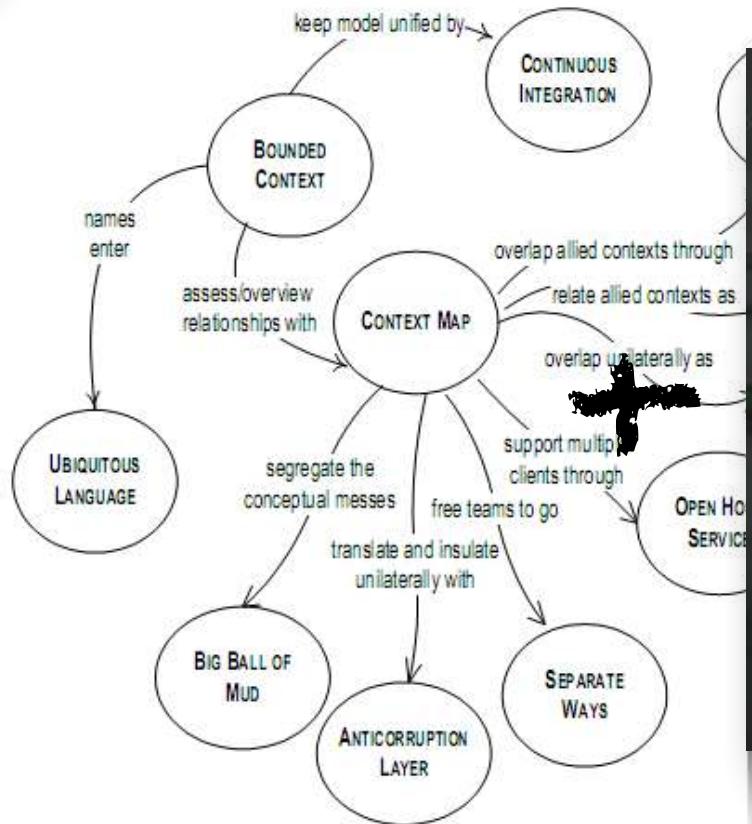
Domain Driven Design



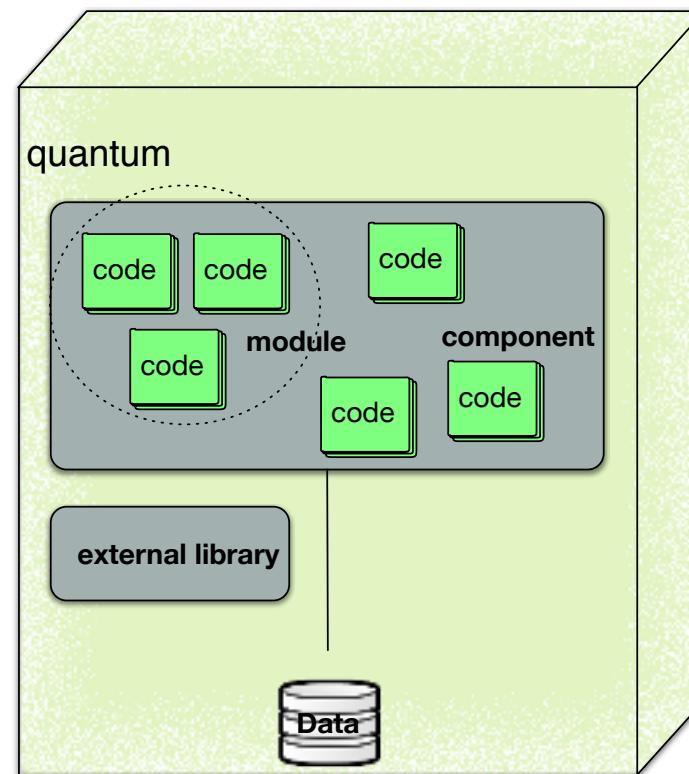


Bounded Context

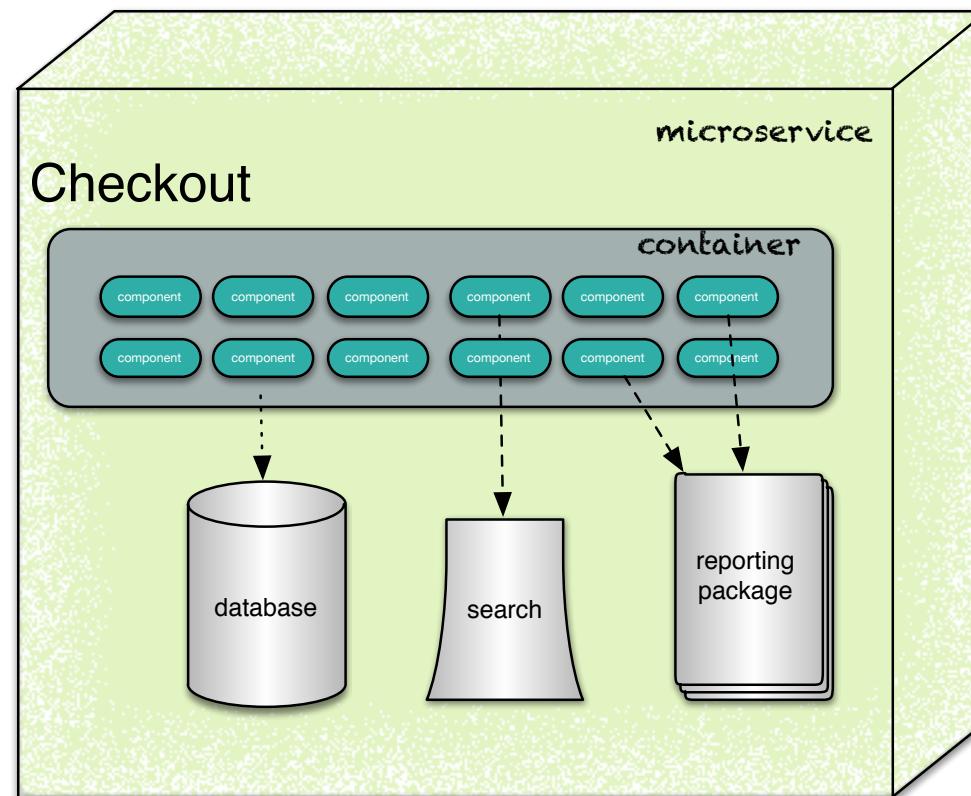
Maintaining Model Integrity



Architectural Quantum

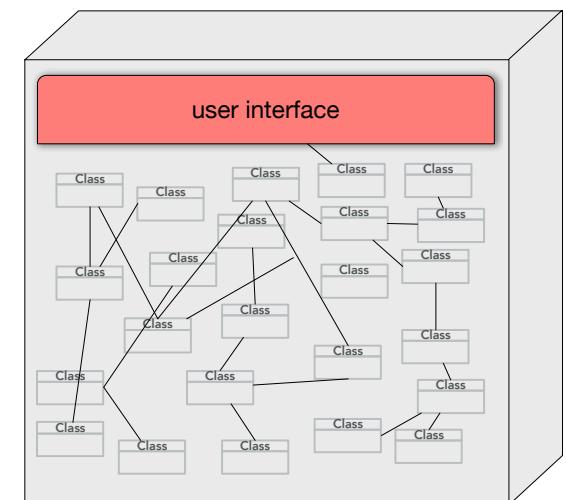
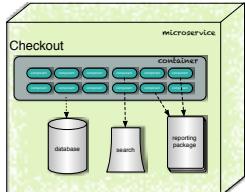


Microservices Quantum



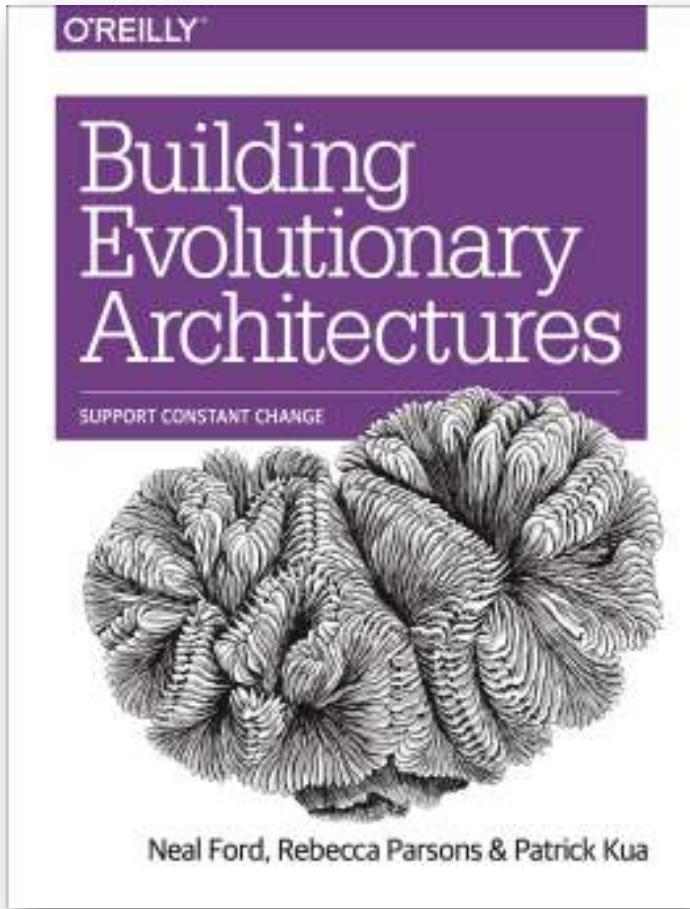
Architectural Quantum

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



Building Evolutionary Architectures

EVOLVABILITY OF
ARCHITECTURAL STYLES



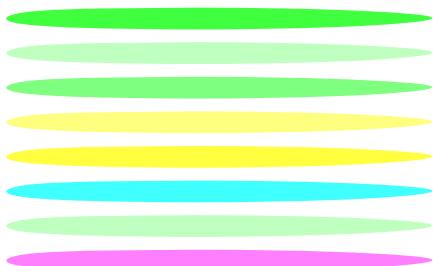
For Each Pattern:



incremental change



guided change via fitness functions

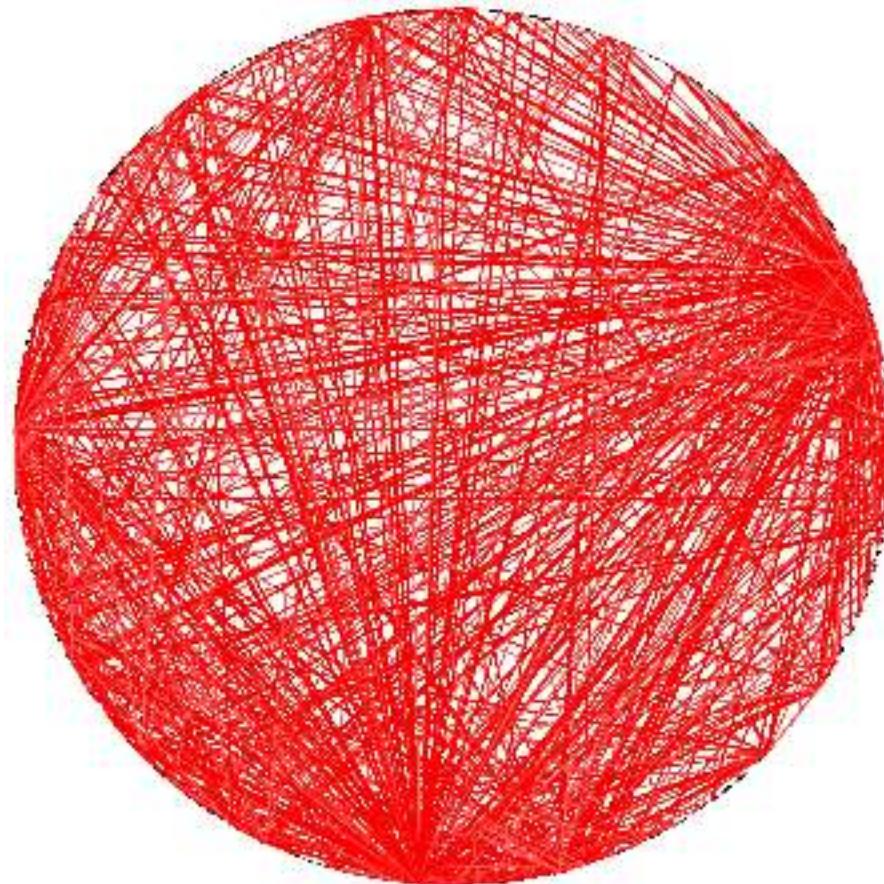


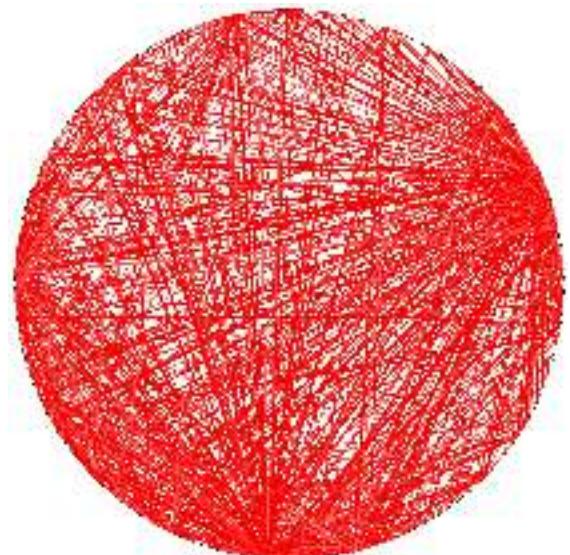
appropriate coupling



architectural quantum

Big Ball of Mud

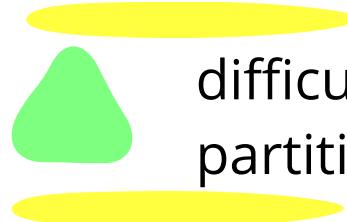




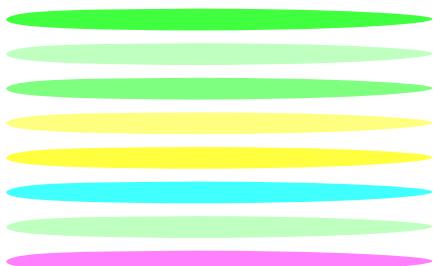
Big Ball of Mud



Rippling side effects for any change



difficult because no clearly defined
partitioning exists

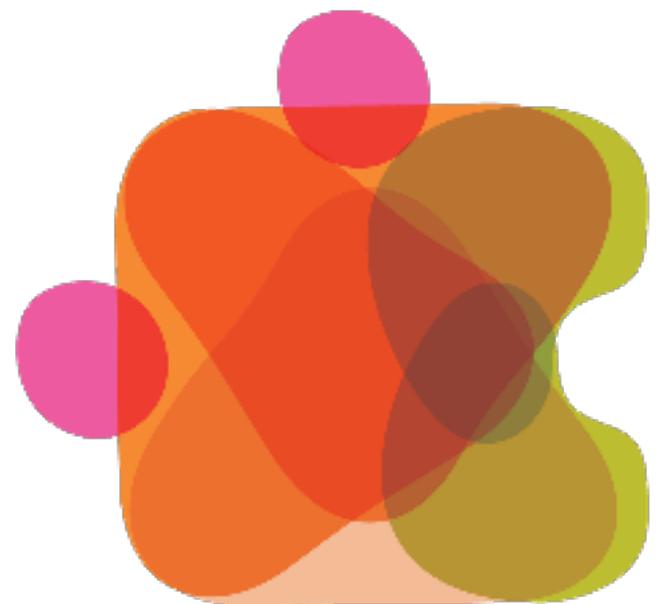


Epitome of **in**appropriate coupling

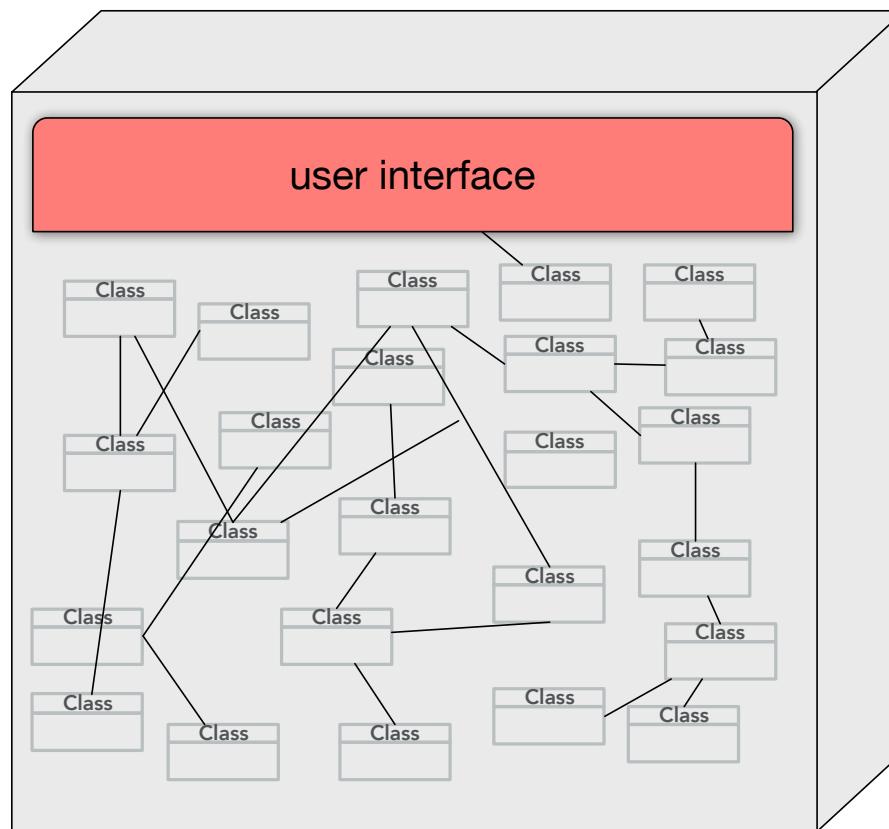


the entire system

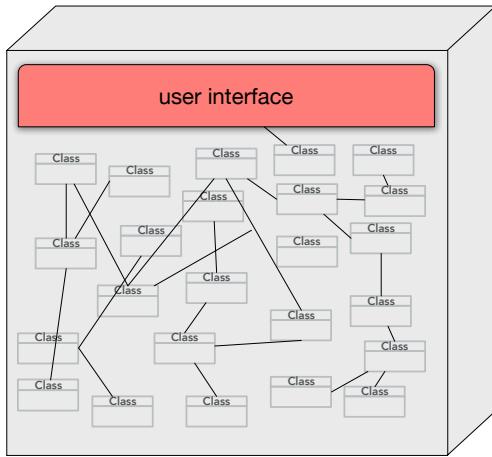
Monoliths



Unstructured Monoliths



Unstructured Monoliths

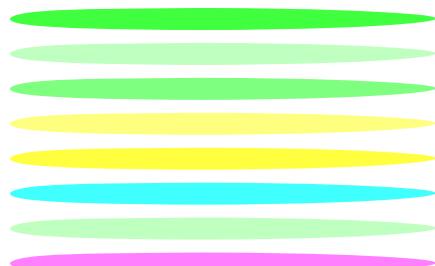


the entire system

⊕ ⊕ ⊕ ⊕ ⊕ ⊕ Large quantum size hinders
incremental change because high coupling requires
deploying large chunks of the application.

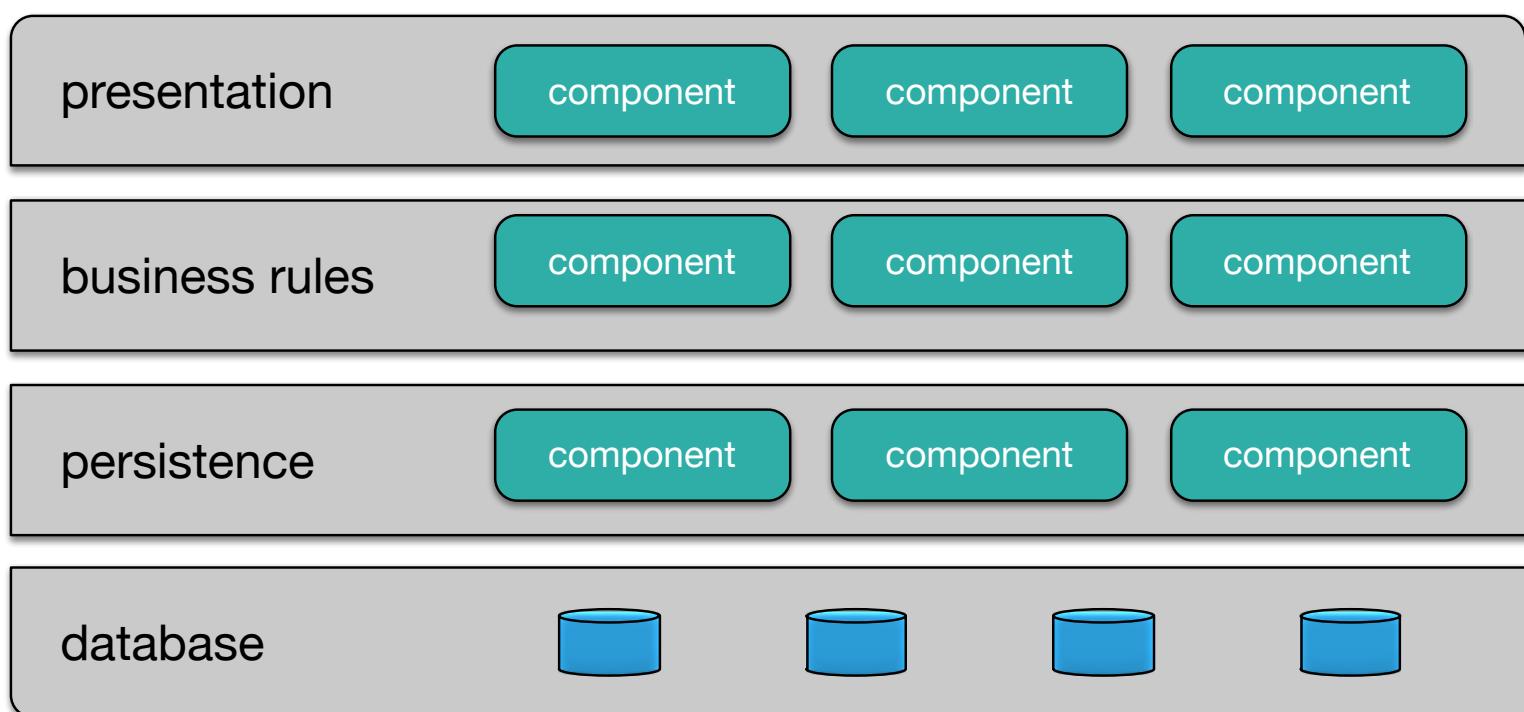


difficult but not impossible

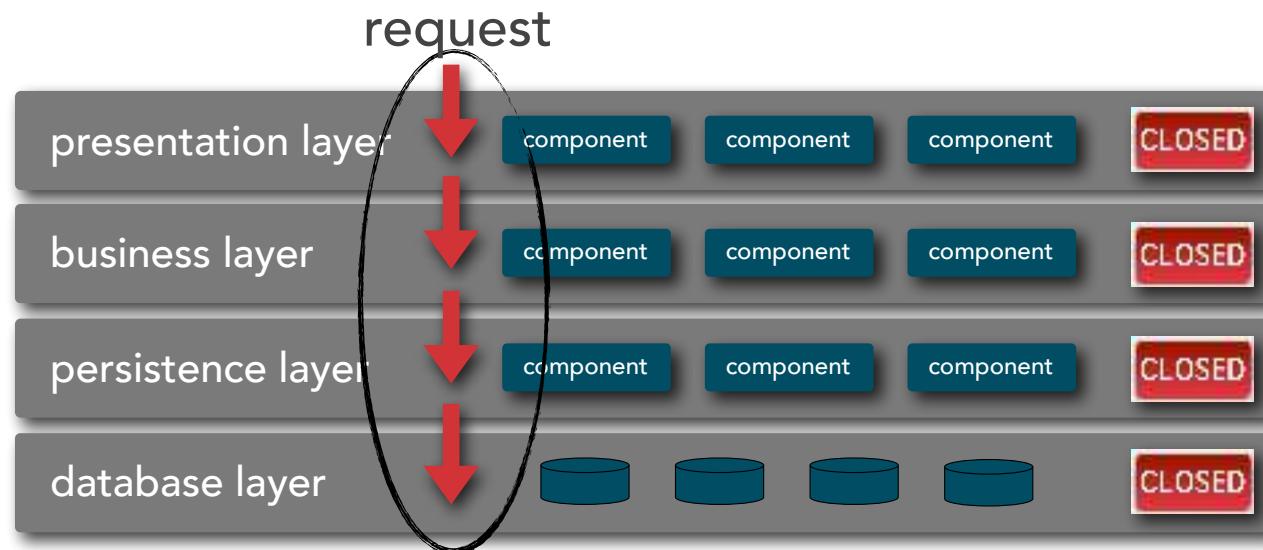


functionally almost as bad as the Big Ball of Mud

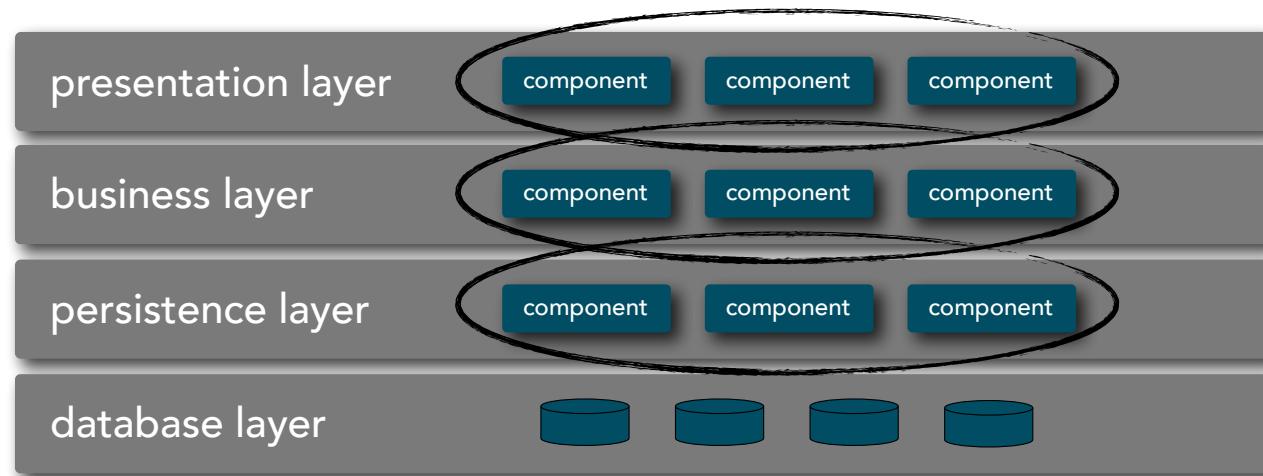
Layered Monolith



Layered Monolith



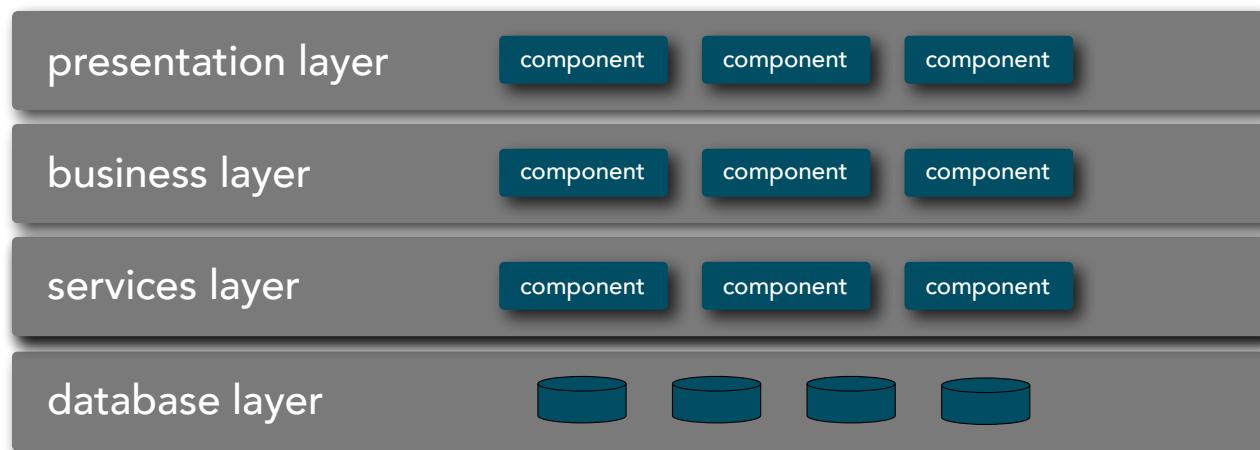
Layered Monolith



separation of concerns

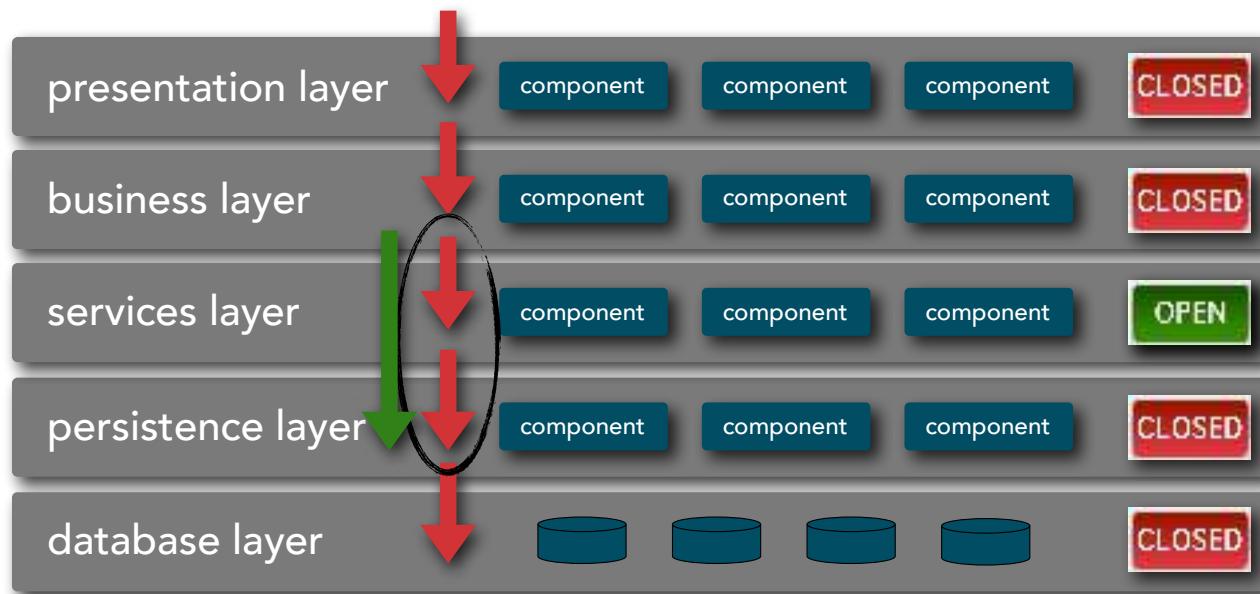
Layered Monolith

hybrids and variants

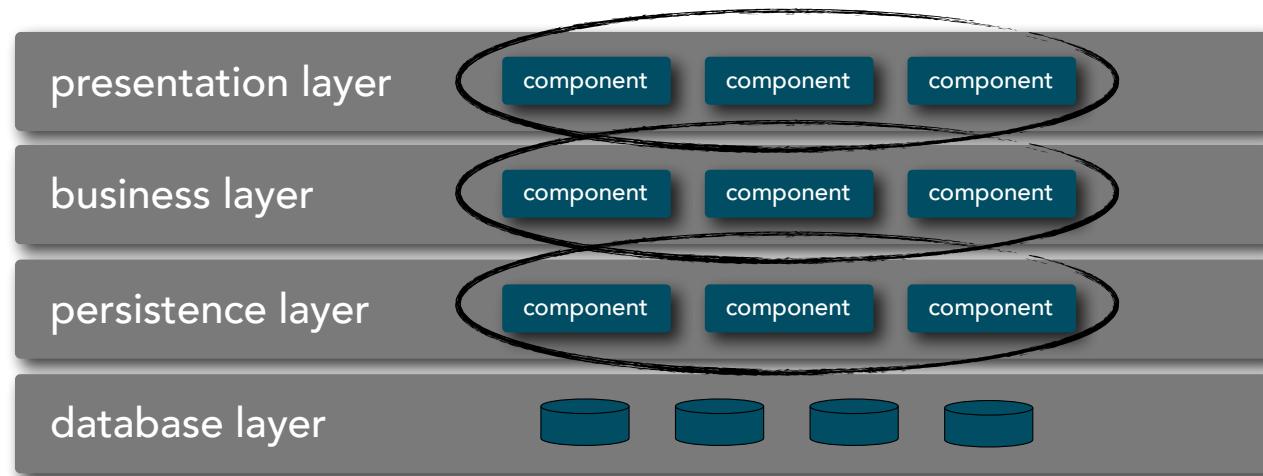


Layered Monolith

hybrids and variants

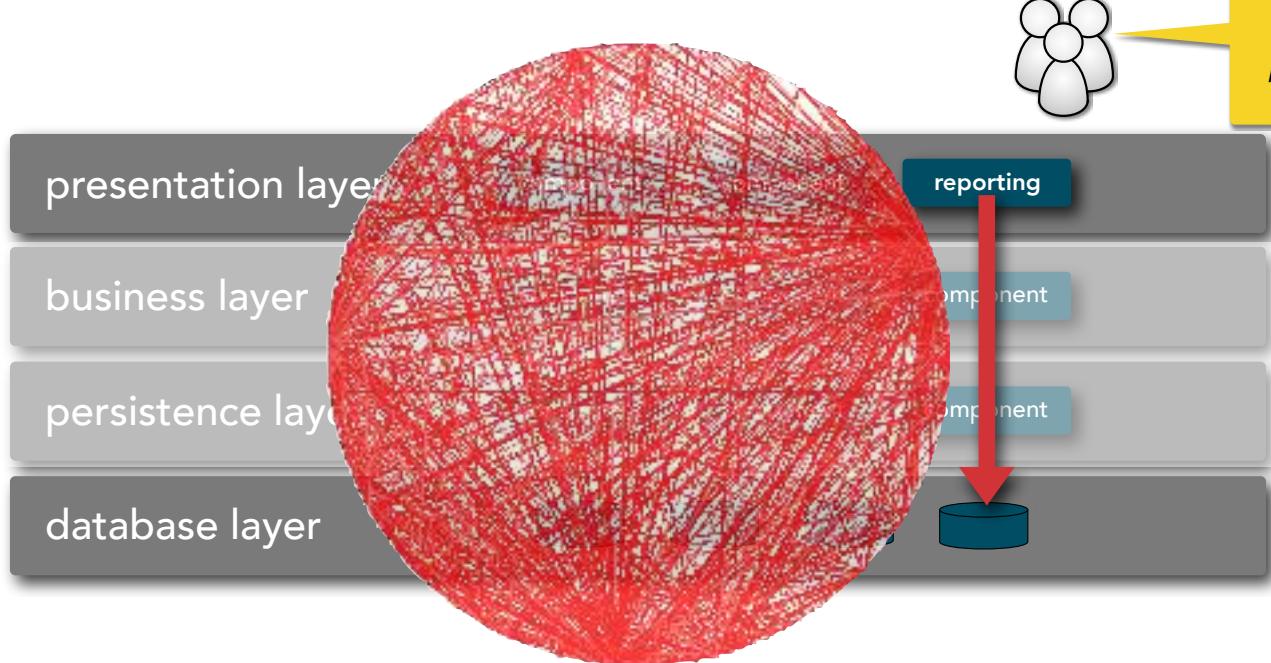


Layered Monolith

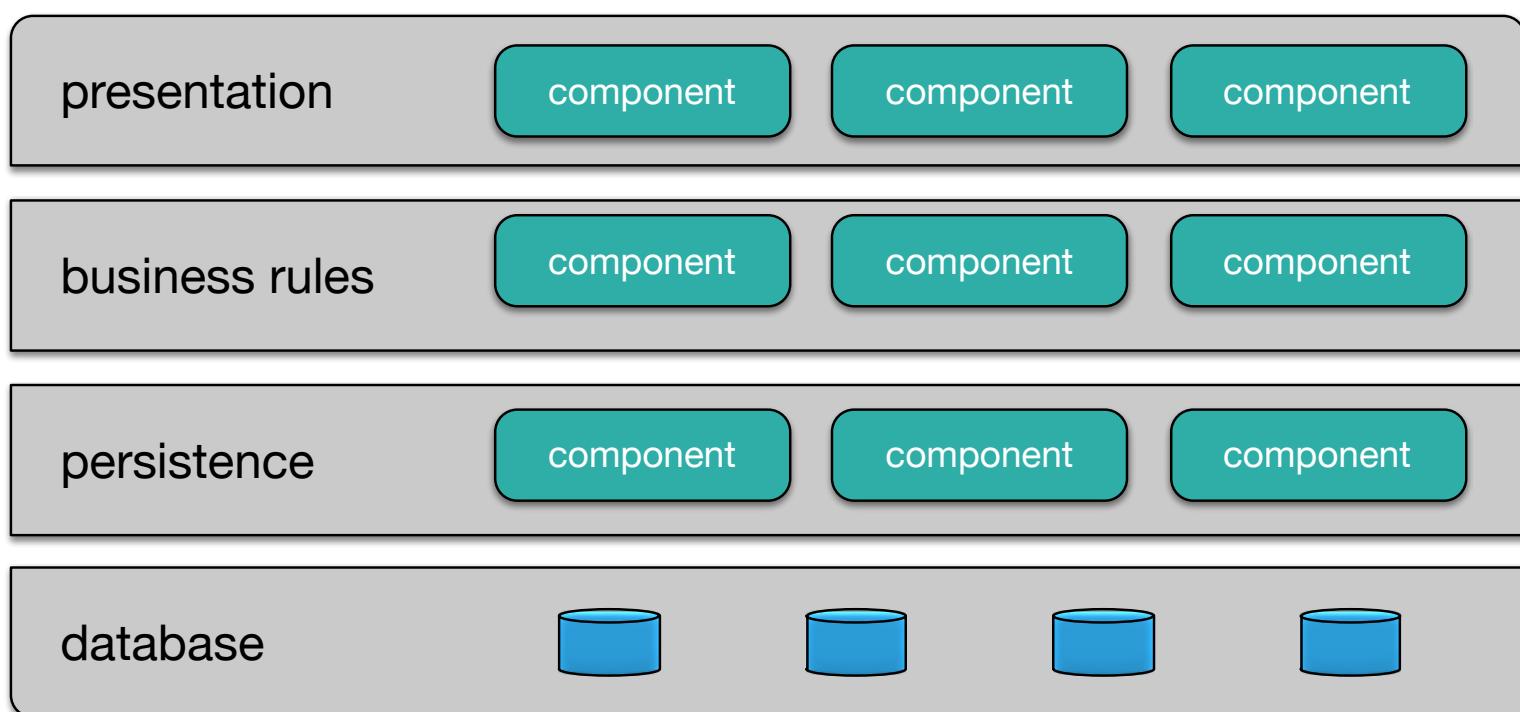


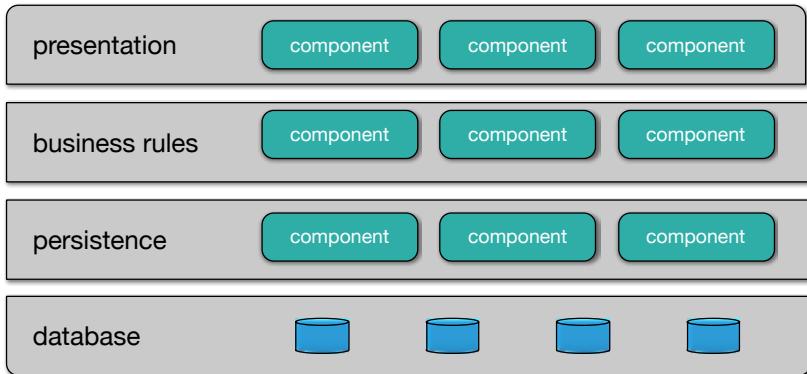
separation of concerns

Pattern Governance

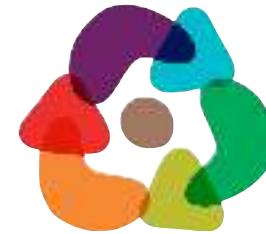


Layered Monolith





Layered Monolith



the entire system

⊕ ⊕ ⊕ ⊕ ⊕ ⊕ Selectively easy based on partitioning

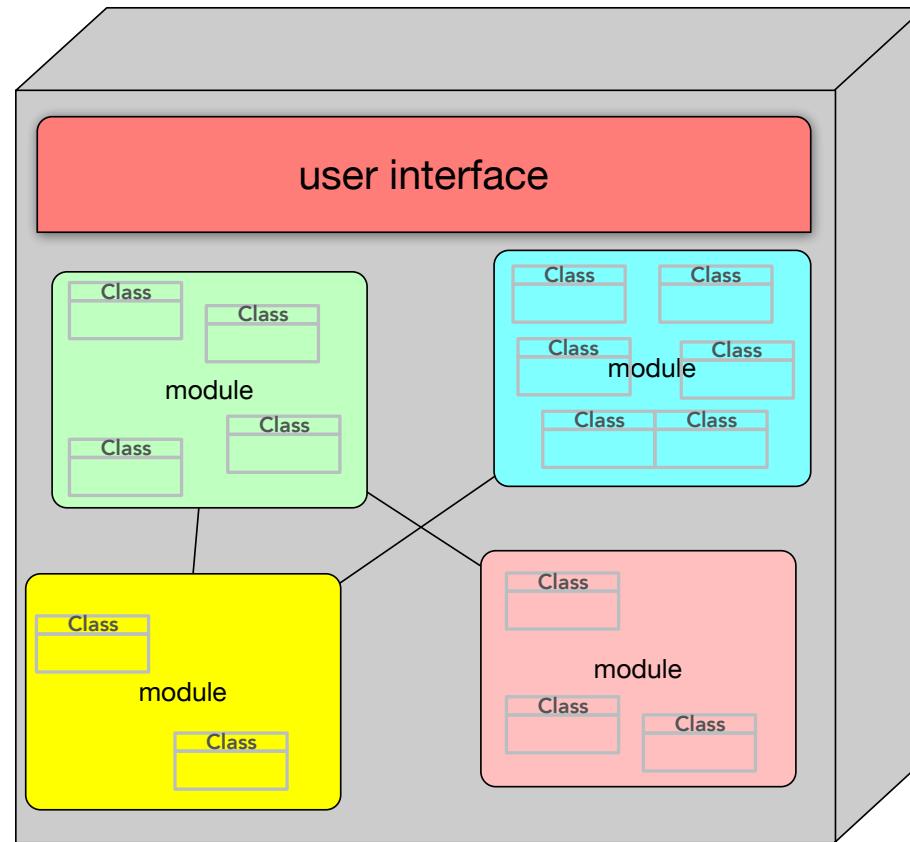


easier because structure is more apparent

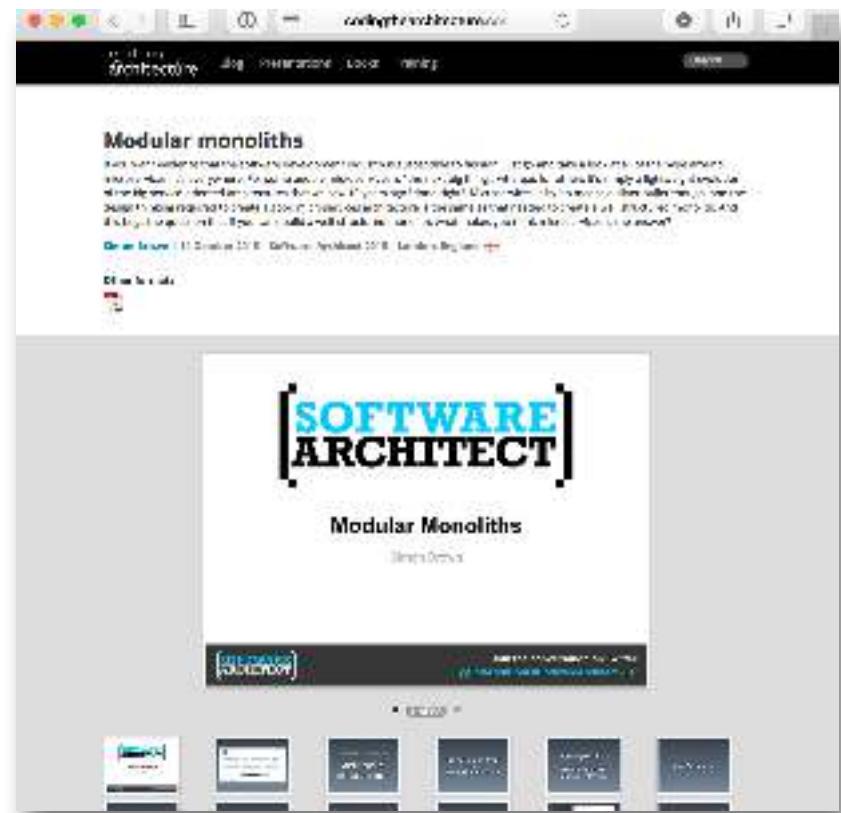
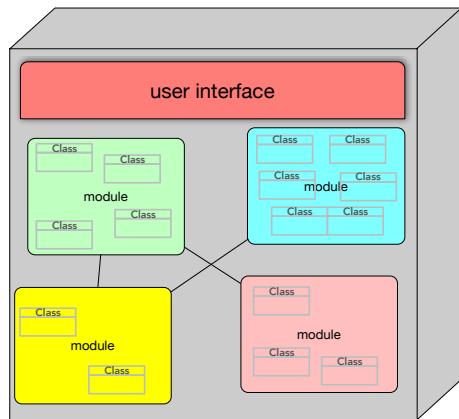


- good technical architecture partitioning
- more difficult for orthogonal concerns

Modular Monoliths



Modular Monoliths



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

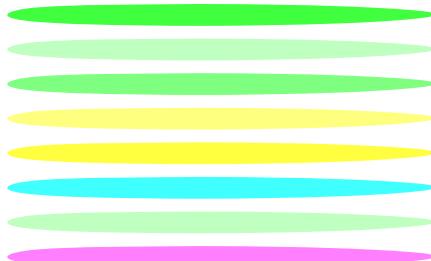
Modular Monoliths

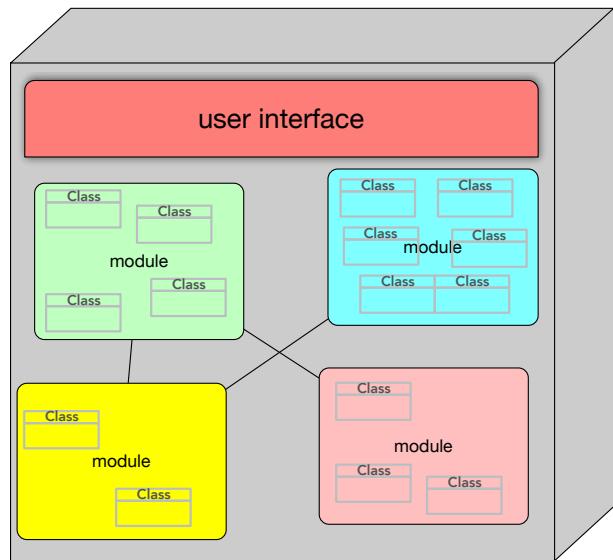


the entire system, with selective better granularity

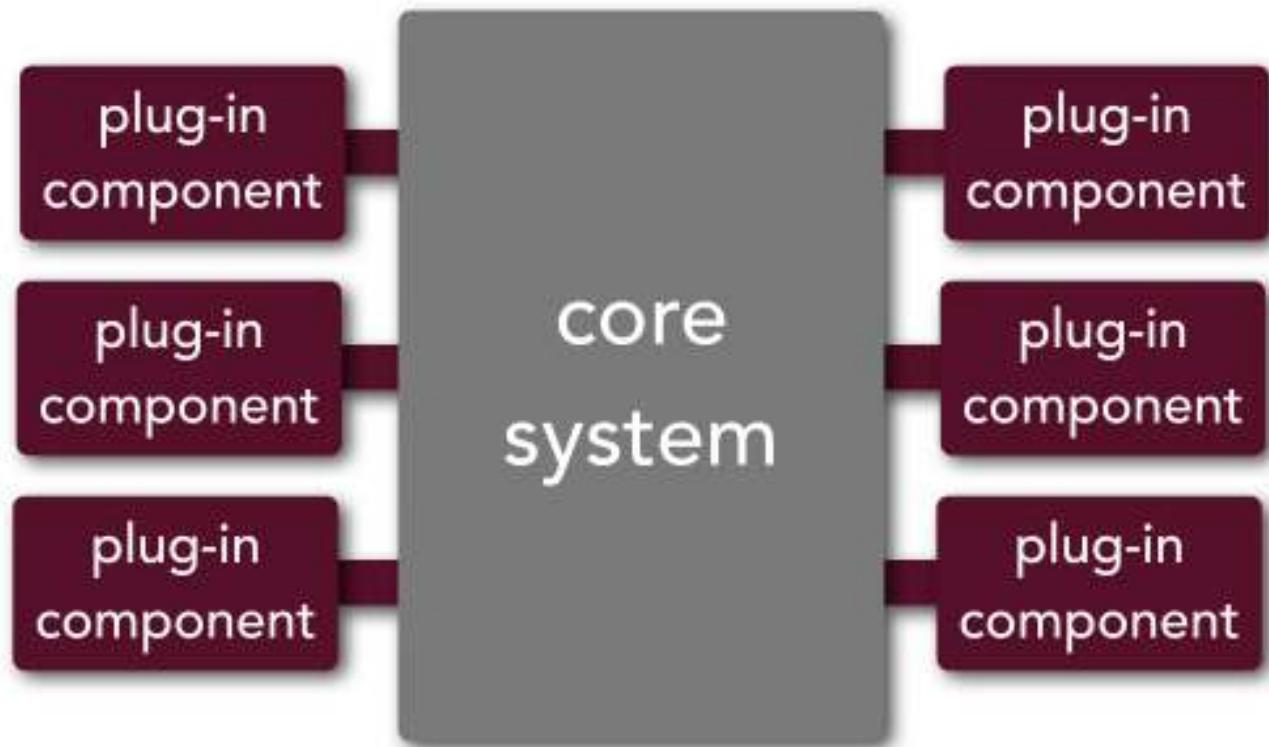
the degree of deployability 
of the components determines the rate of
incremental change.

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

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

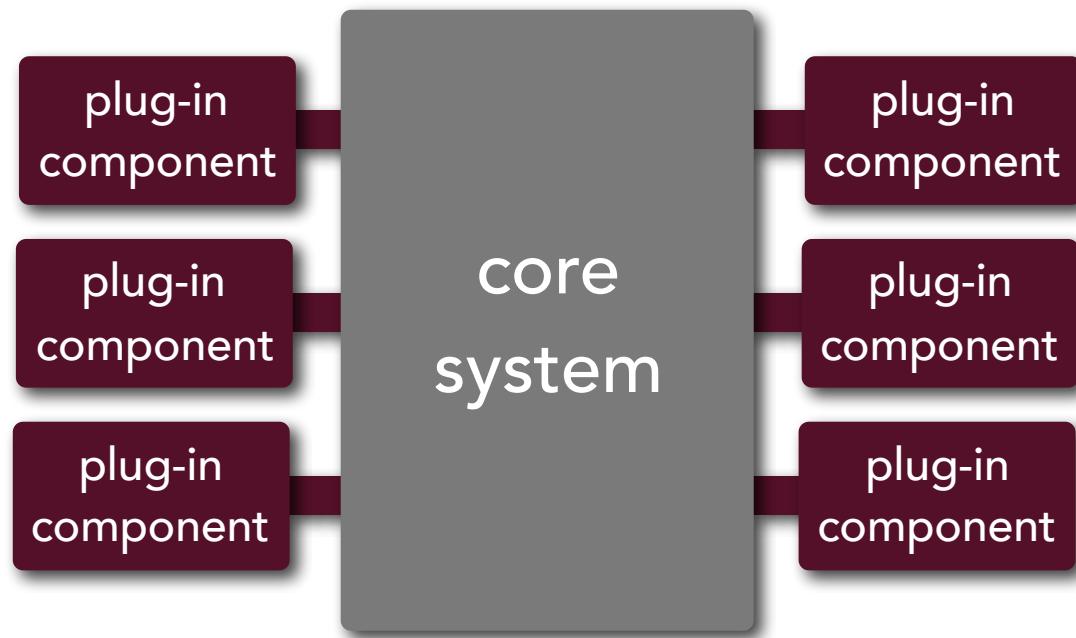


Microkernel



microkernel architecture

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



microkernel architecture

architectural components

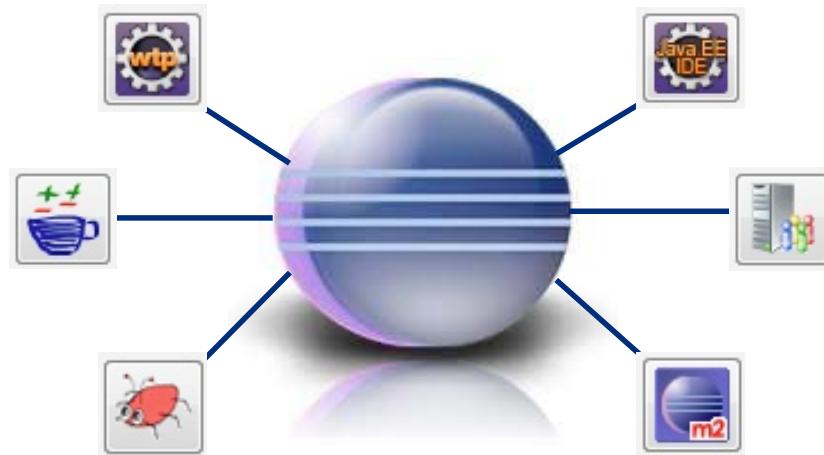


minimal functionality to run system
general business rules and logic
no custom processing



standalone independent module
specific additional rules or logic

microkernel architecture



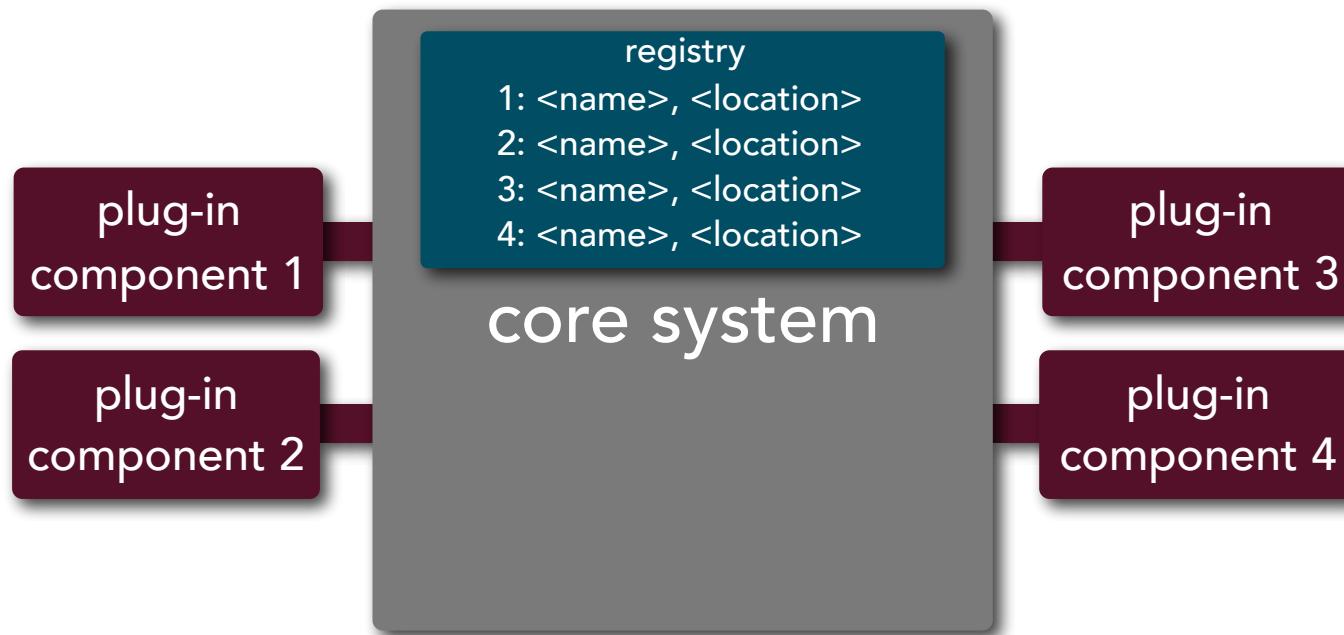
microkernel architecture

claims processing



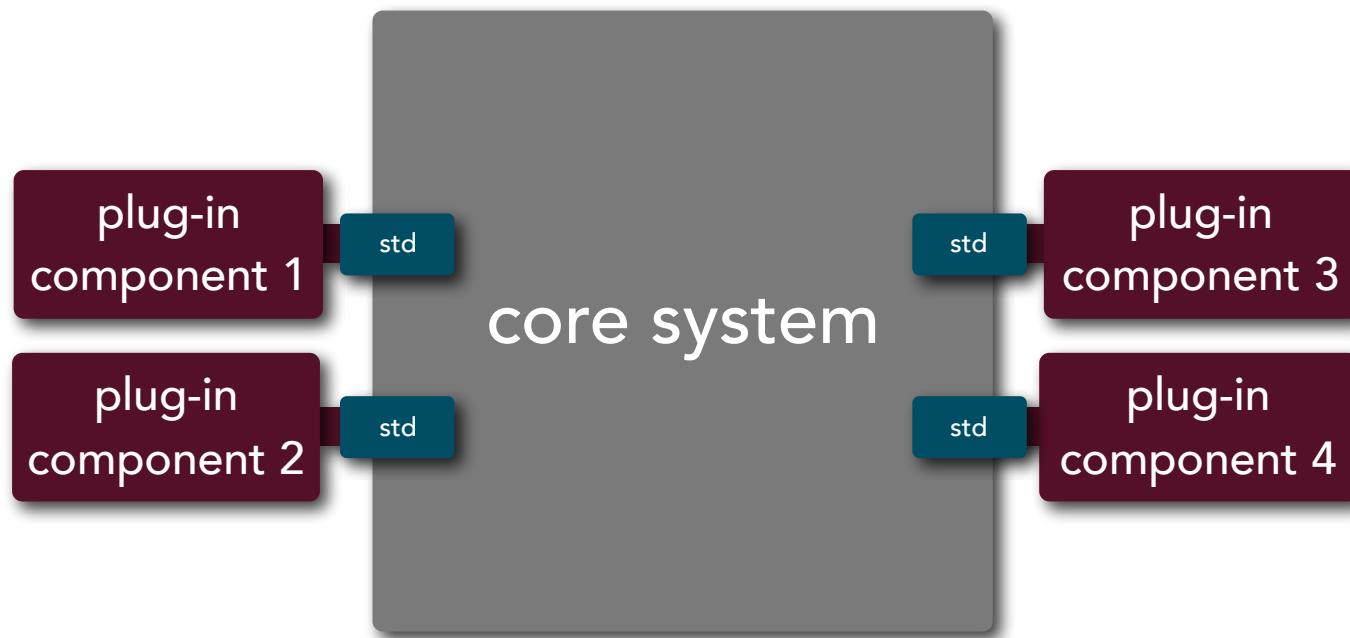
microkernel architecture

registry



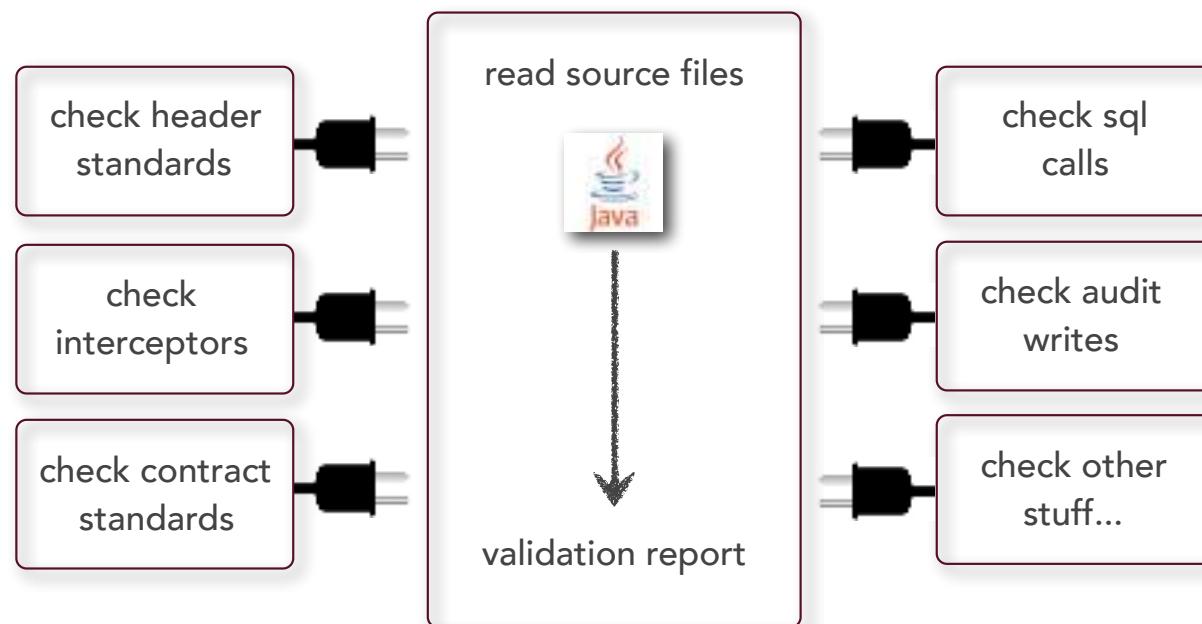
microkernel architecture

plug-in contracts

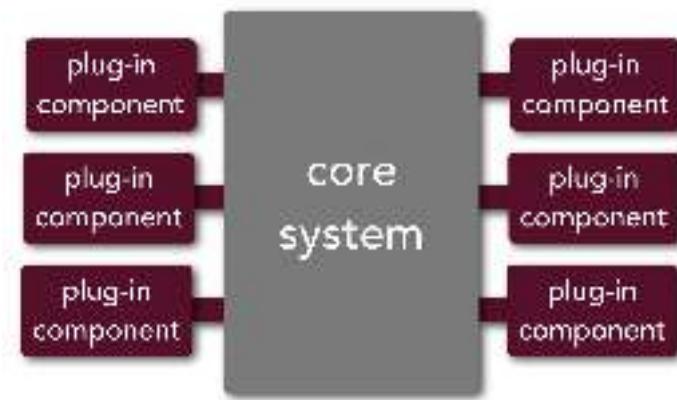


microkernel architecture

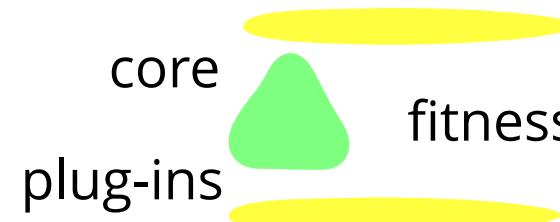
source validation tool



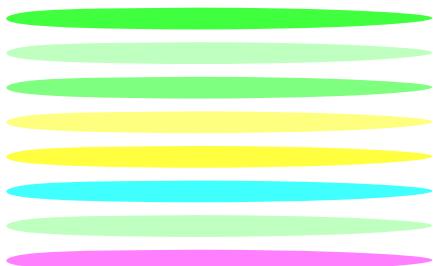
Microkernel



most changes via plug-ins



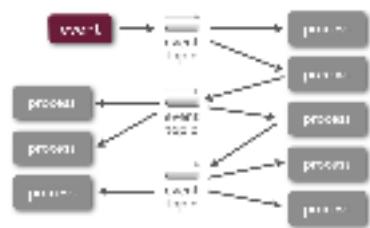
fitness functions for integration tests



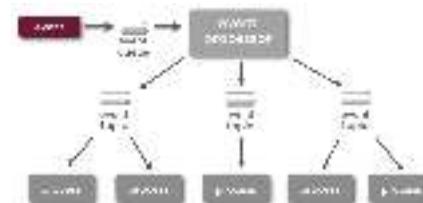
coupling well defined by architectural pattern

beware the Last 10% Trap

Event-driven Architectures

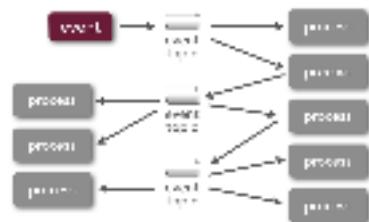


broker topology

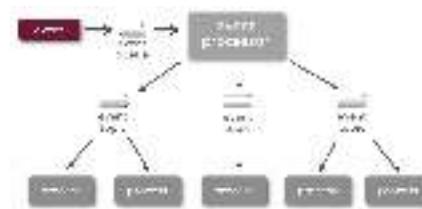


mediator topology

event-driven architecture



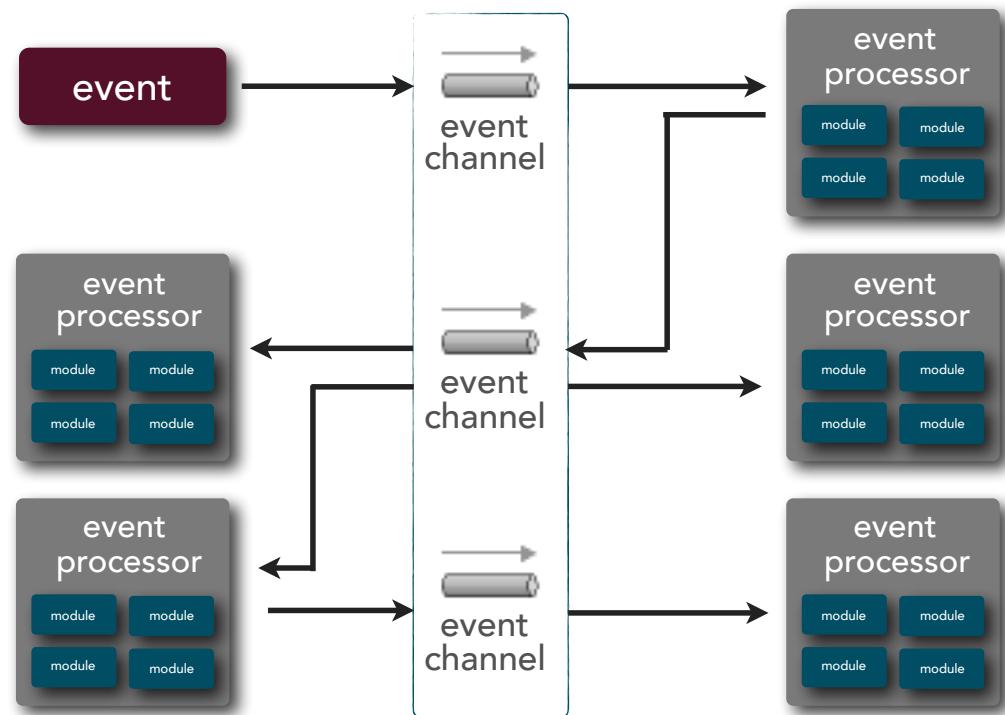
broker topology



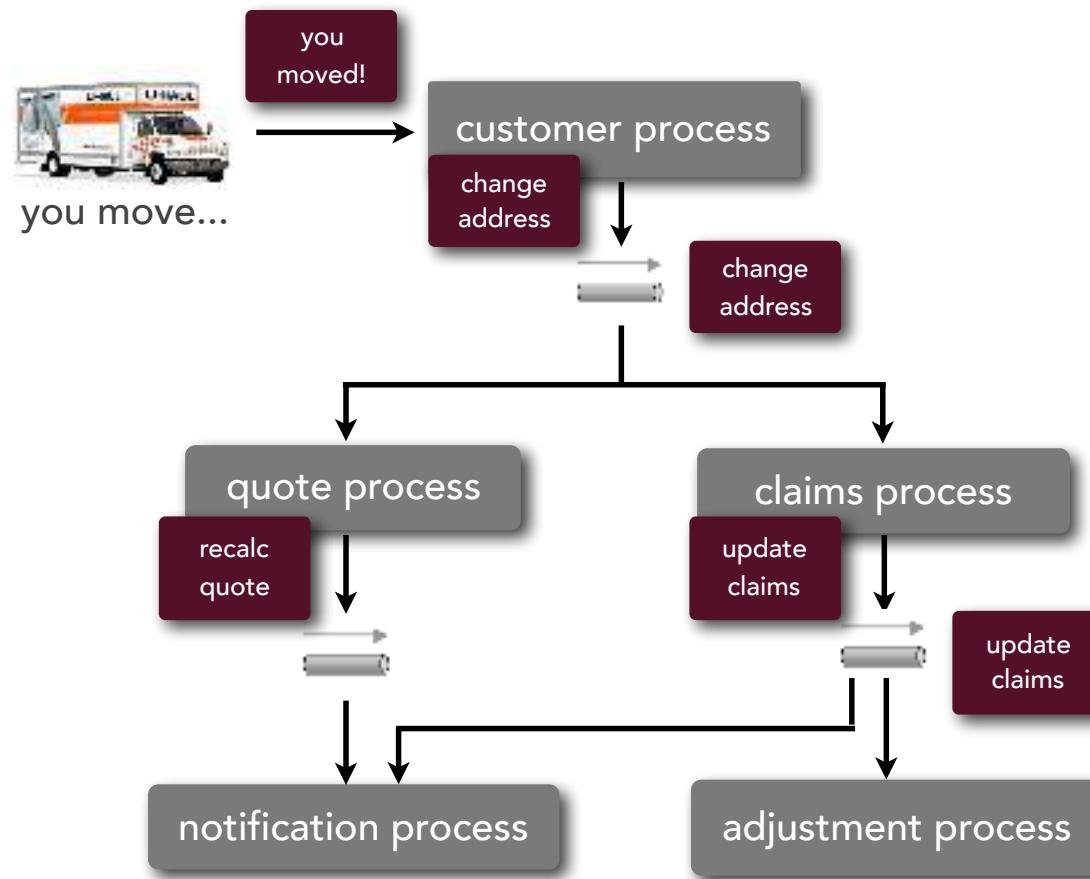
mediator topology

event-driven architecture

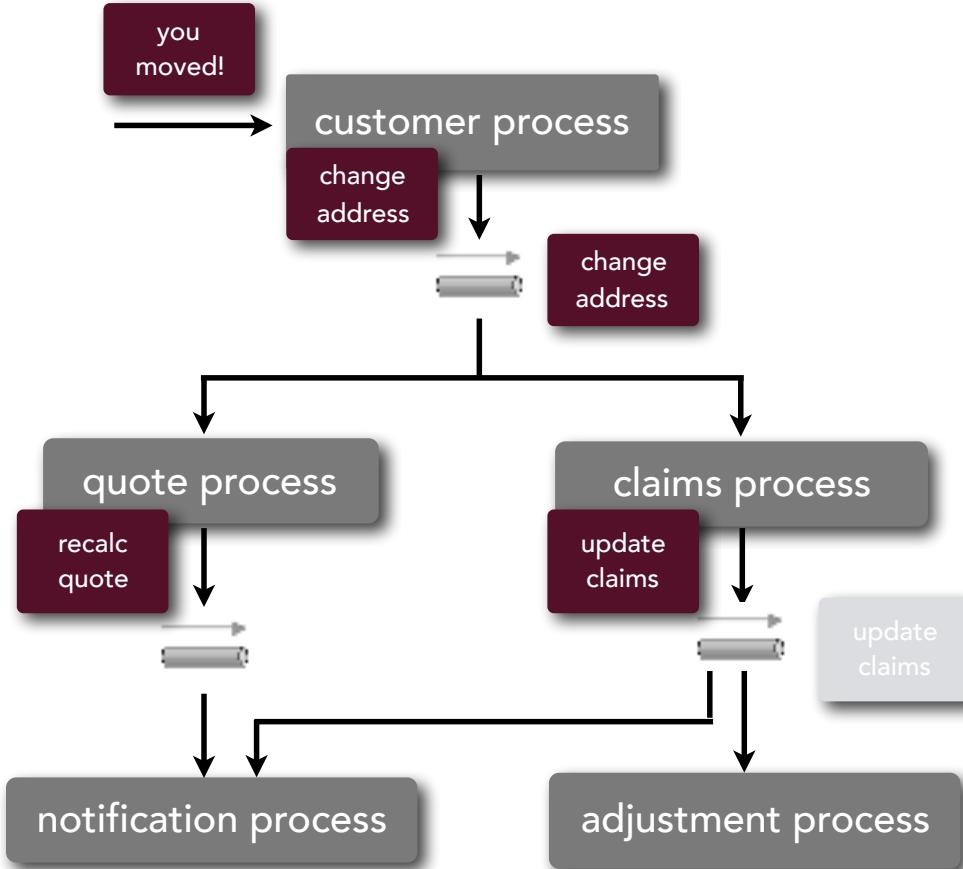
broker topology



event-driven architecture

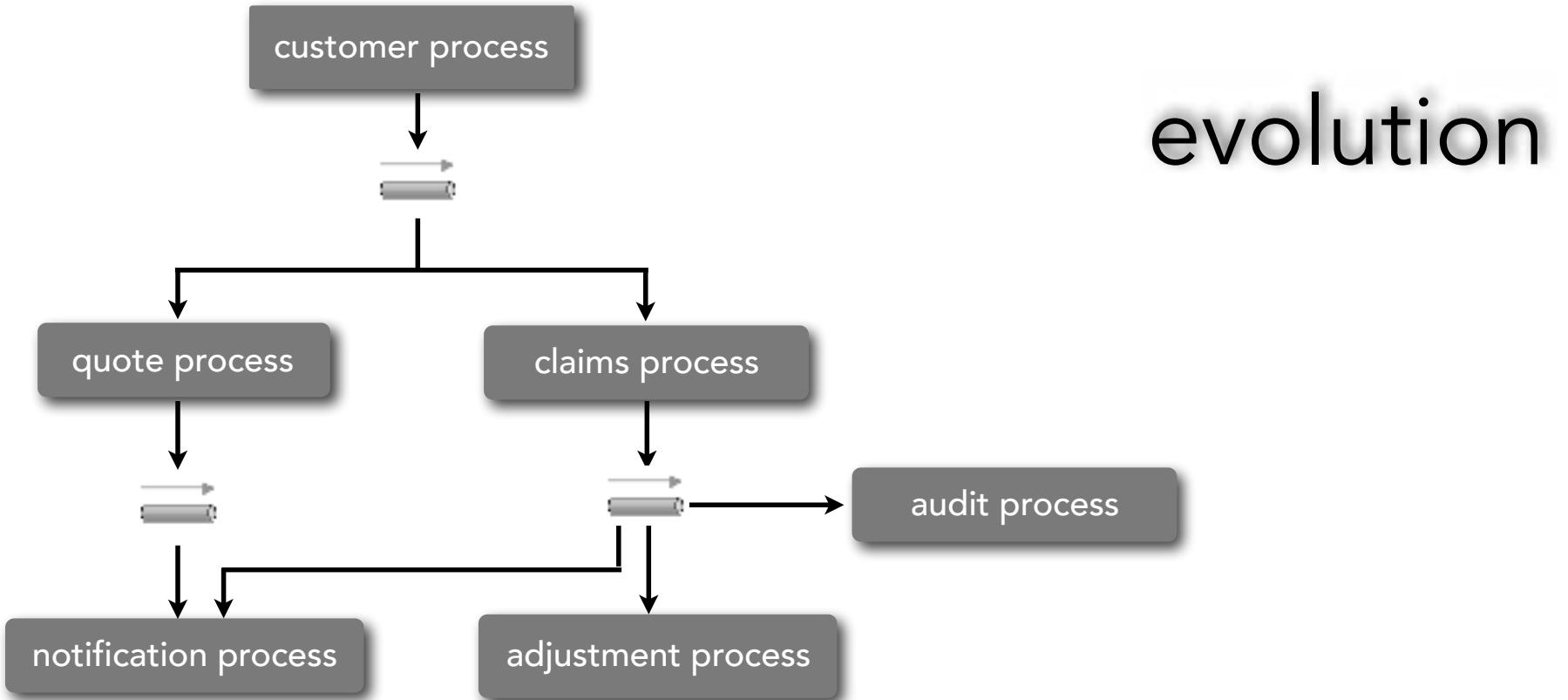


event-driven architecture

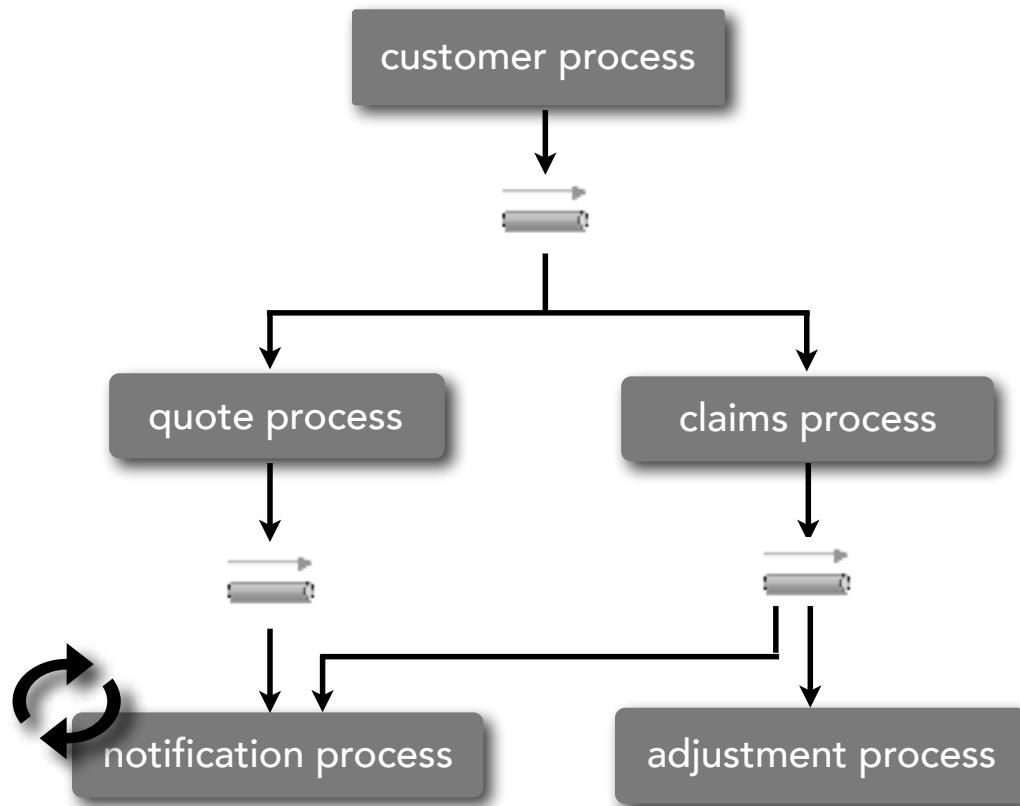


Error handling?

event-driven architecture



event-driven architecture

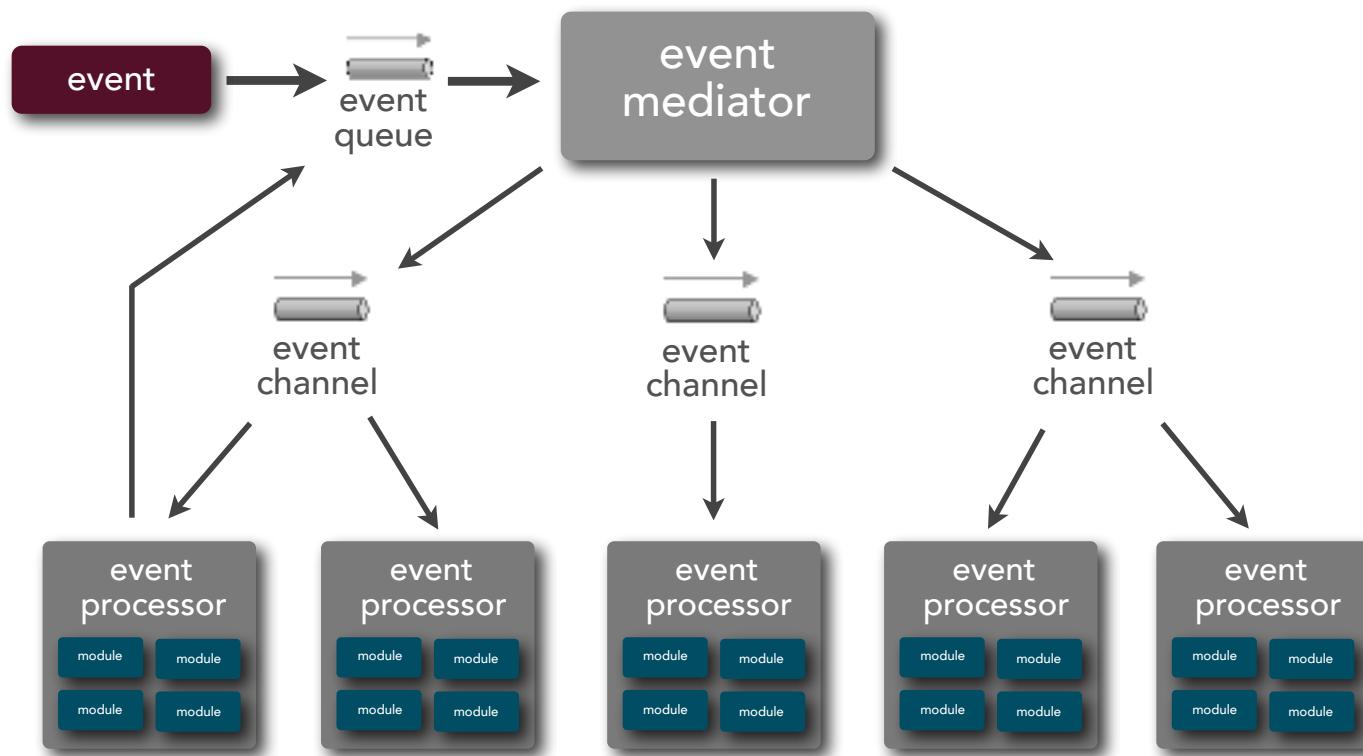


Unified notification
message to customer?

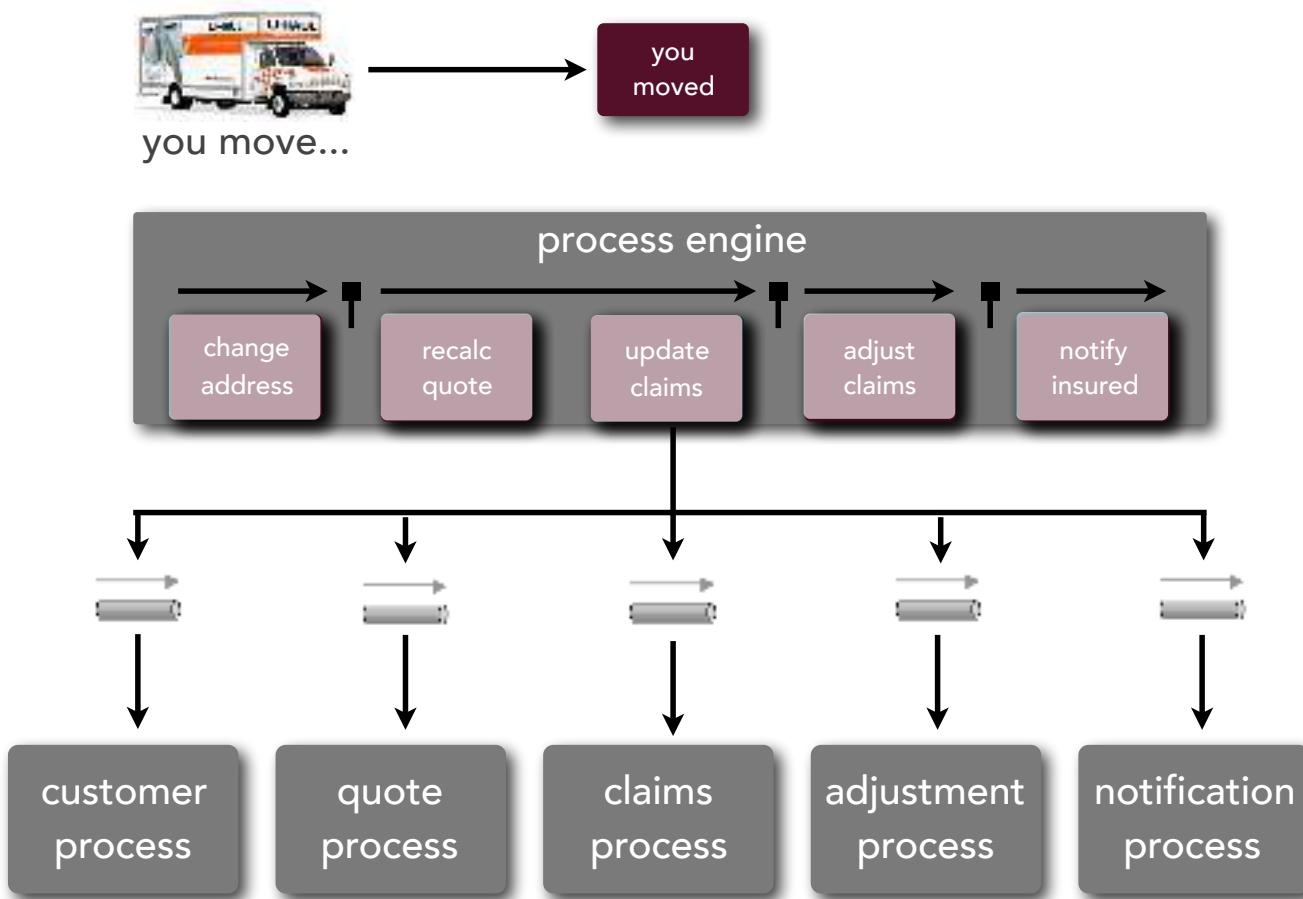
In asynchronous systems,
coordination happens at the end

event-driven architecture

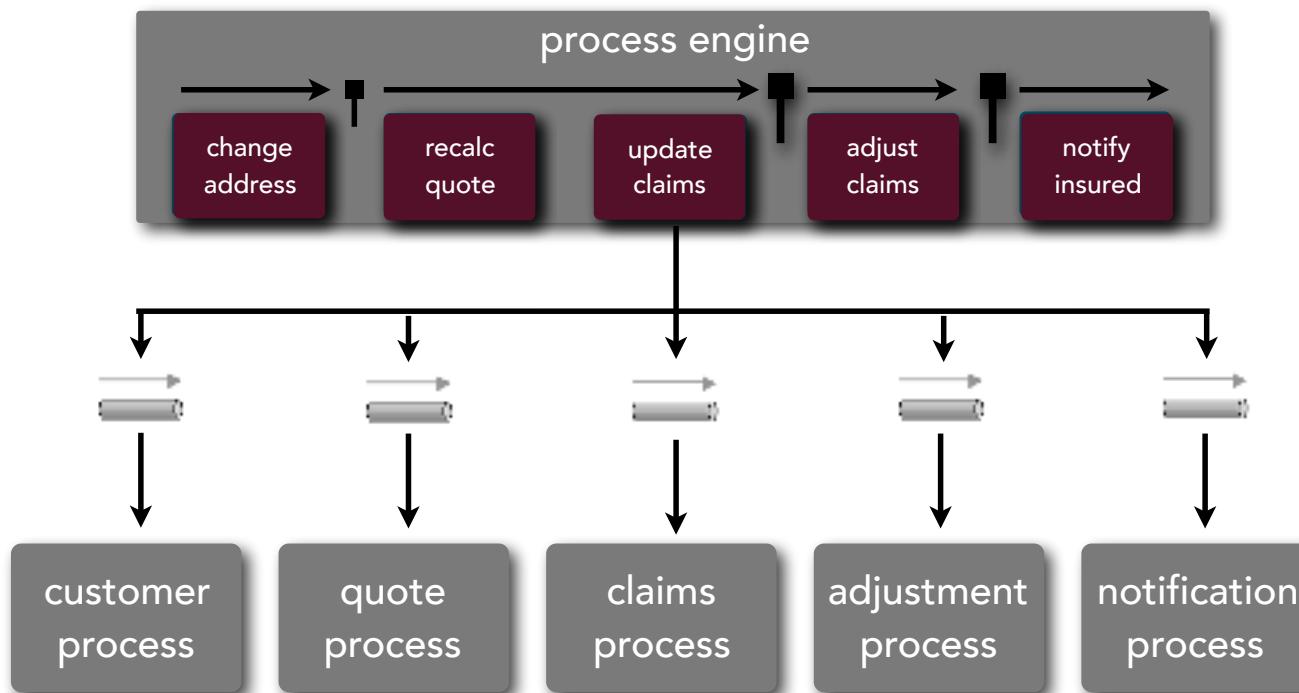
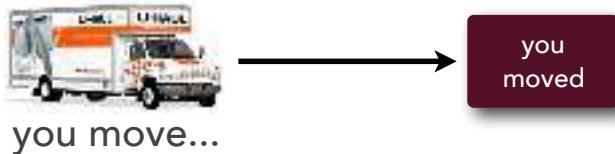
mediator topology



event-driven architecture

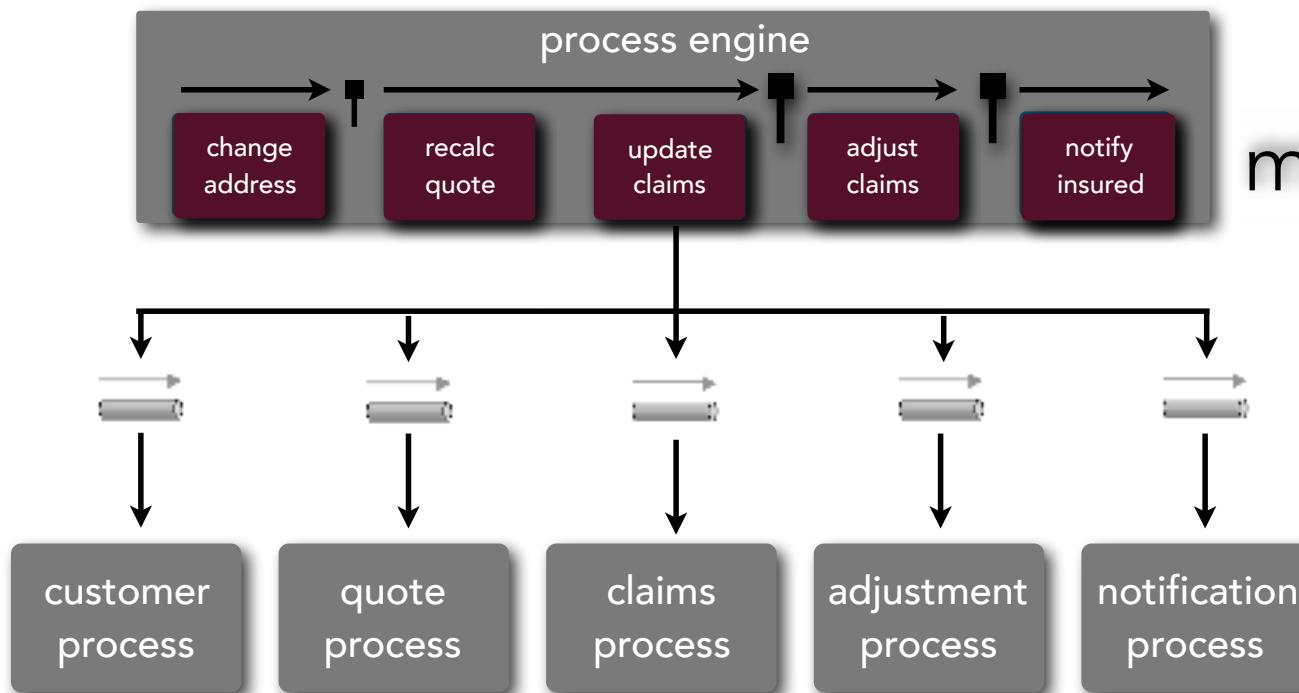
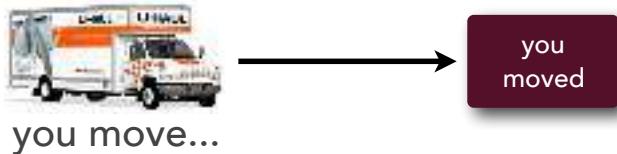


event-driven architecture



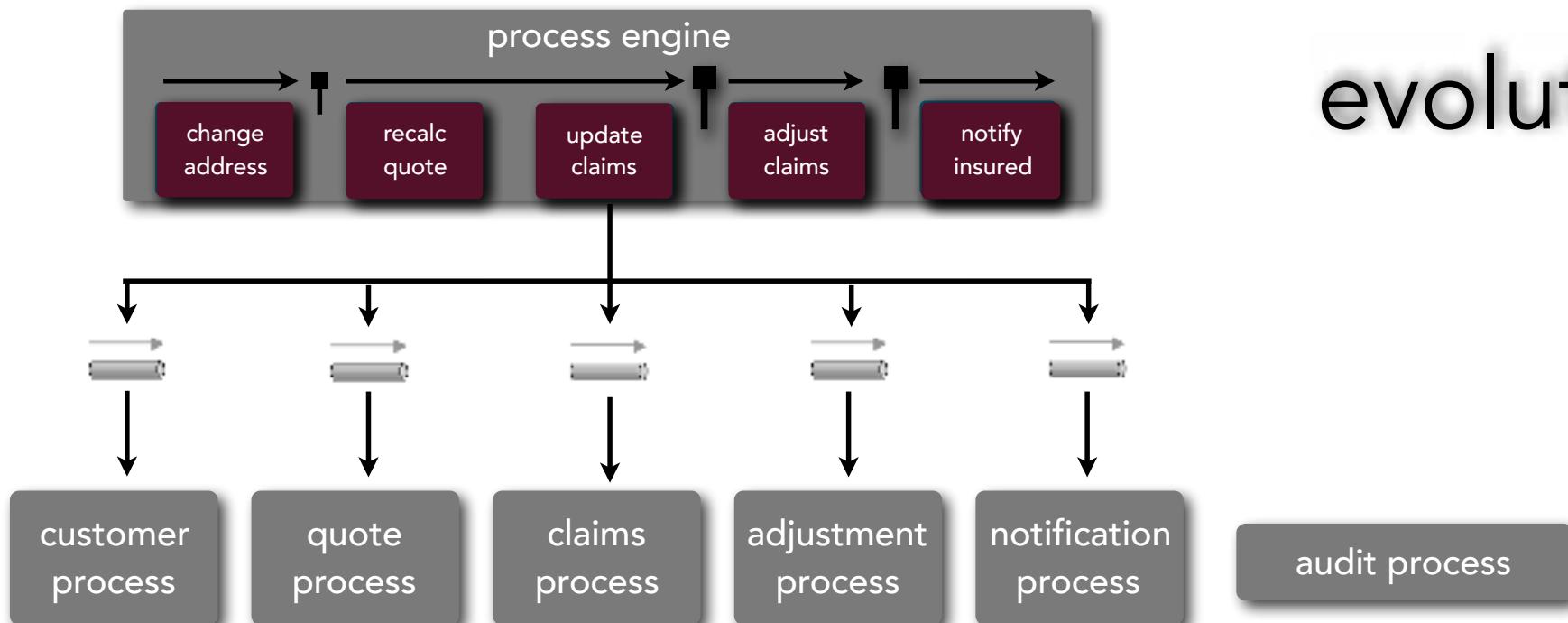
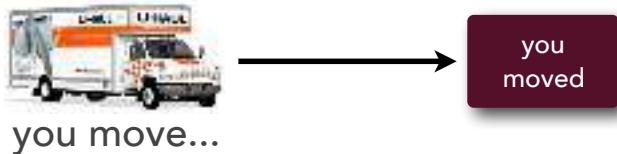
Error handling?

event-driven architecture



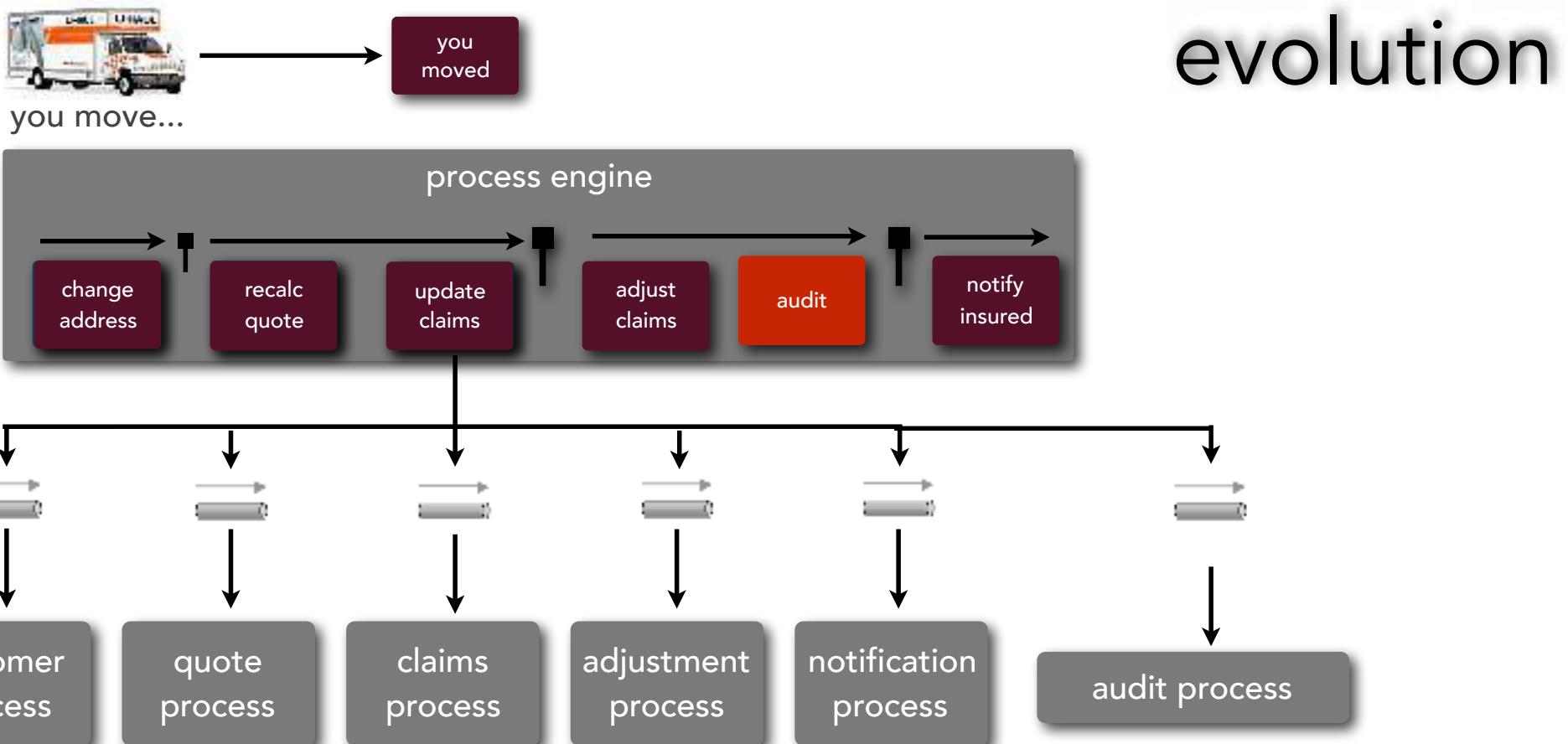
Unified notification message to customer?

event-driven architecture

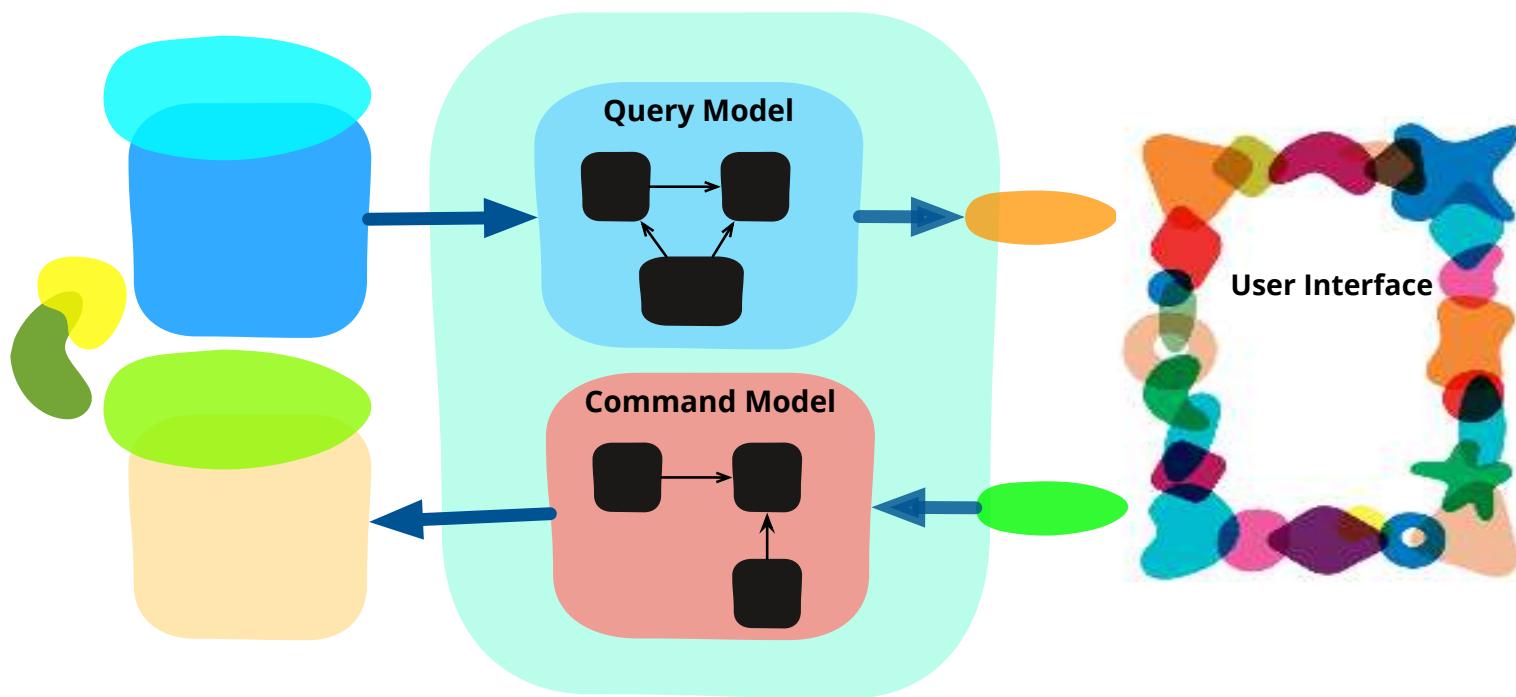


evolution

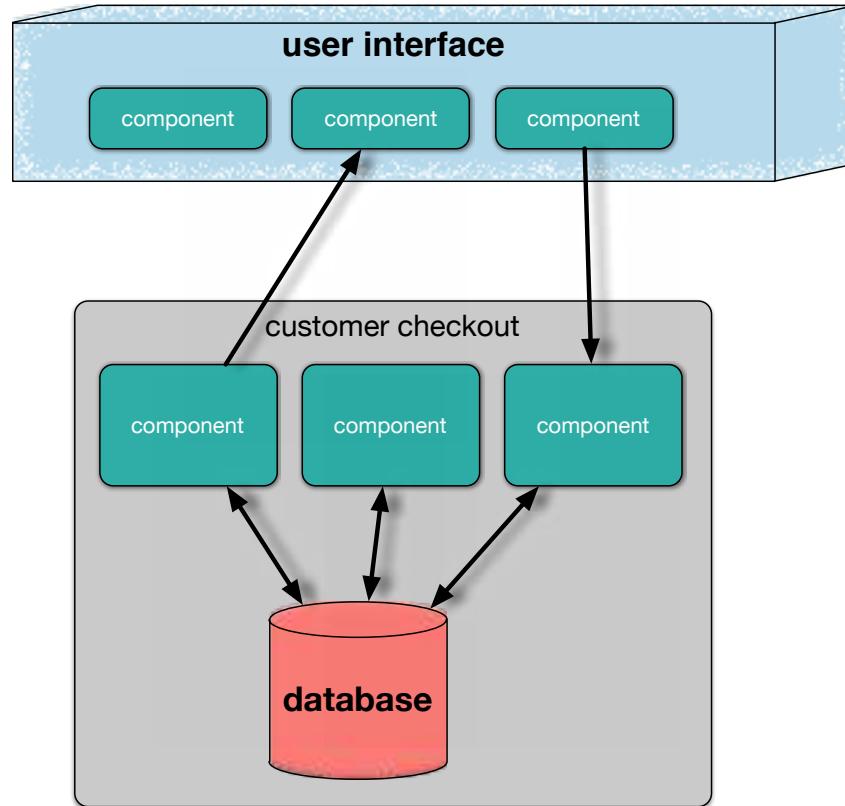
event-driven architecture



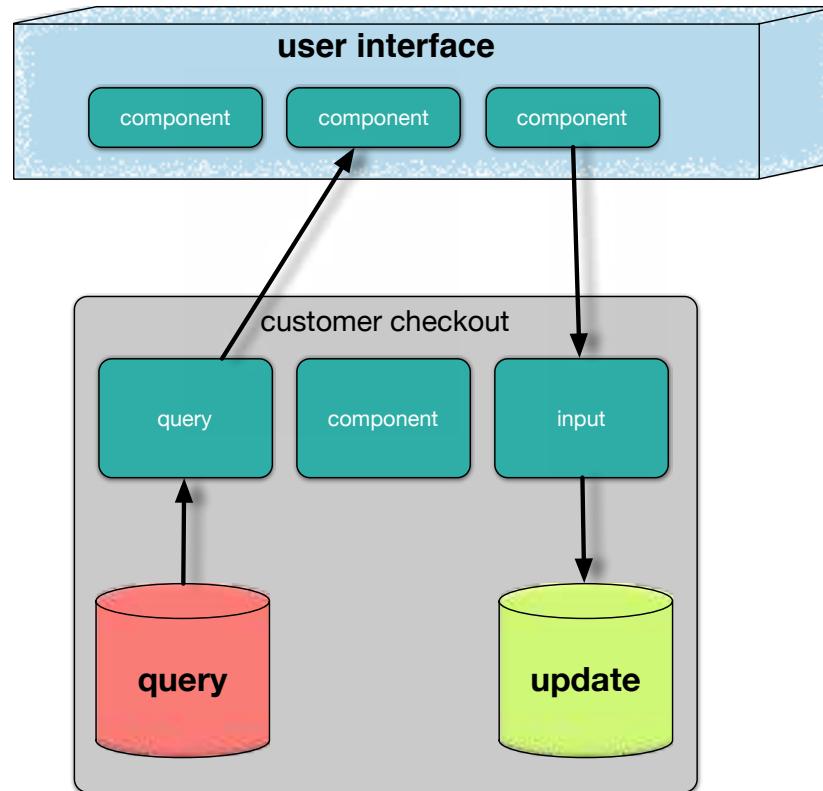
CQRS



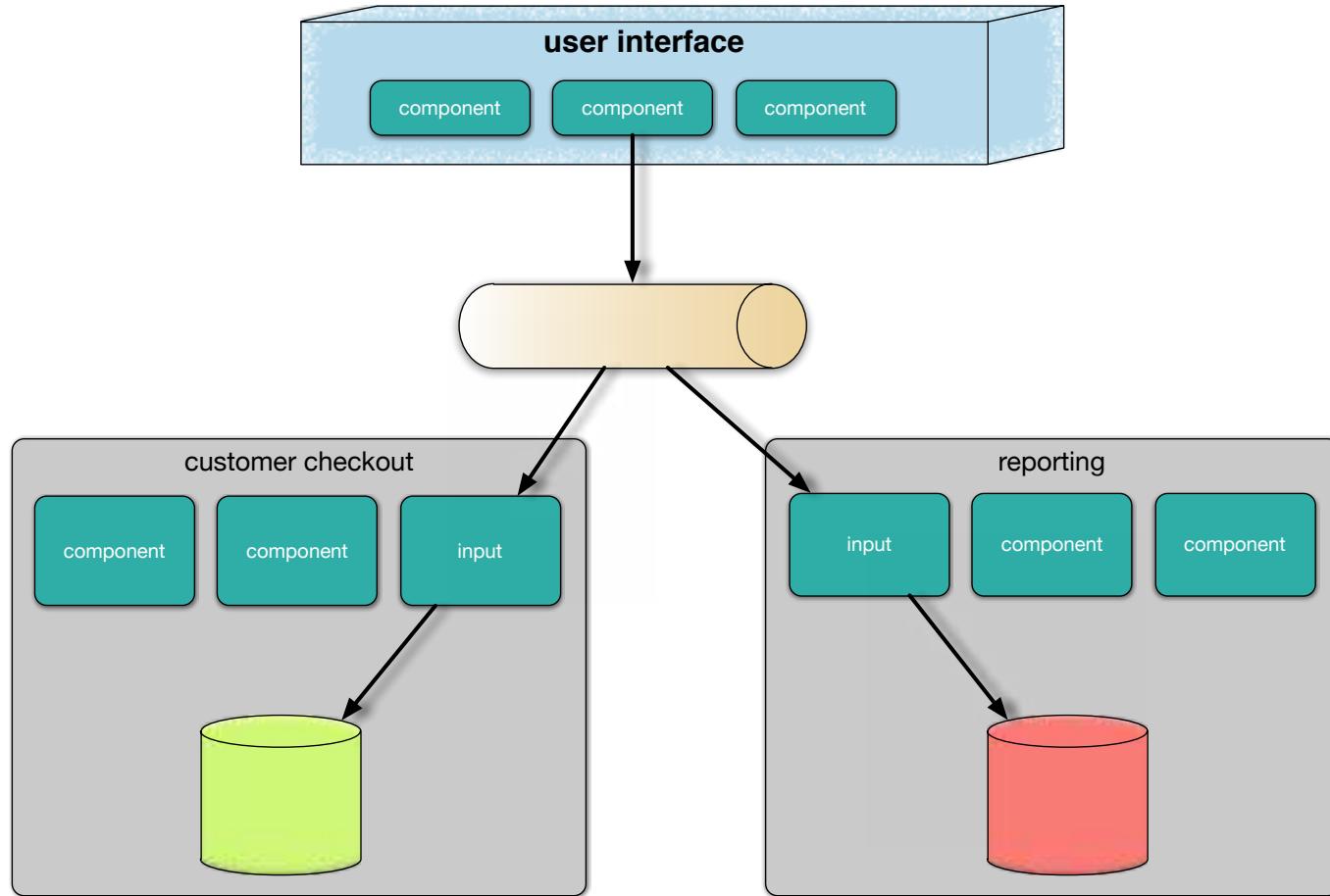
<http://codebetter.com/gregyoung/2010/02/16/cqrs-task-based-uis-event-sourcing-agh/>



traditional



CQRS Command Query Responsibility Separation



eventual consistency, events, & reports

eventual consistency



"Building reliable distributed systems at a worldwide scale demands trade-offs between consistency and availability."

http://www.allthingsdistributed.com/2008/12/eventually_consistent.html

CQRS natural fits

task-based user interface

meshes well with event sourcing

eventual consistency

consistency or availability
(but never both)

complex or granular domains

Conway's Law

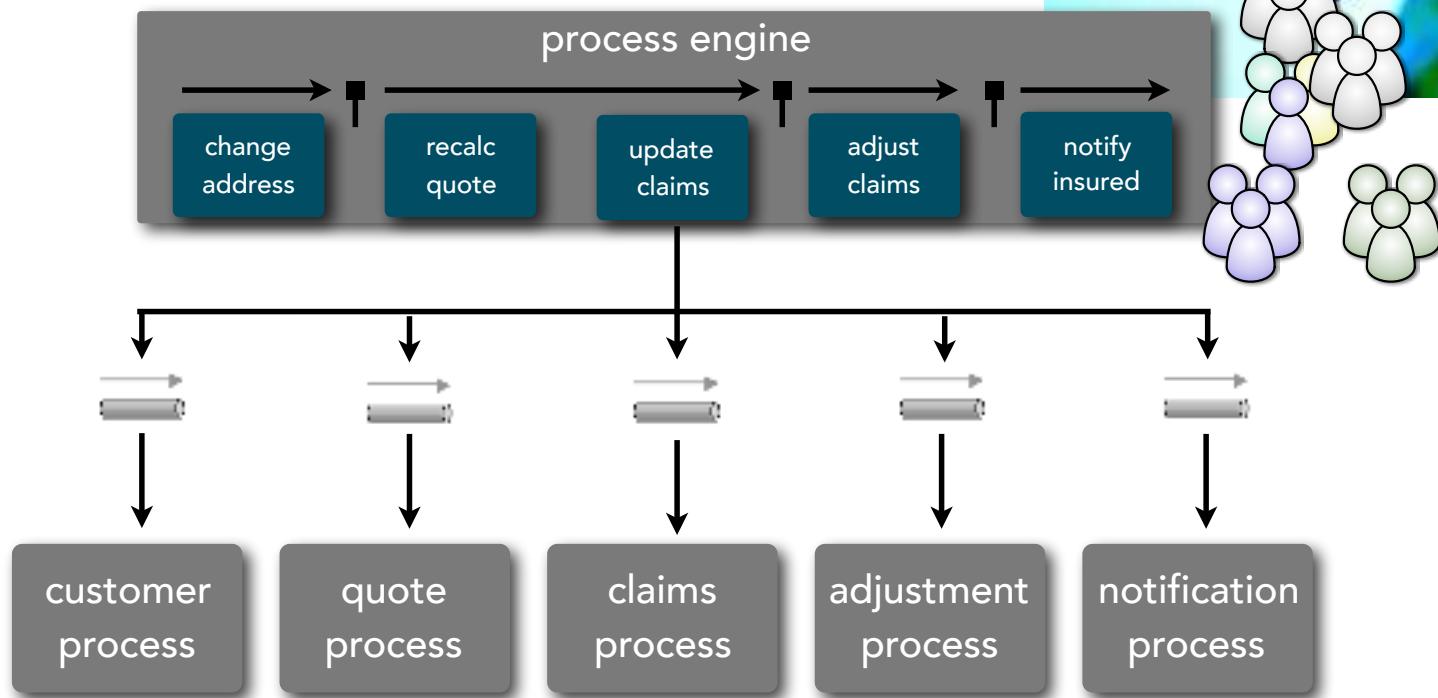
“organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations”

Melvin Conway, 1968

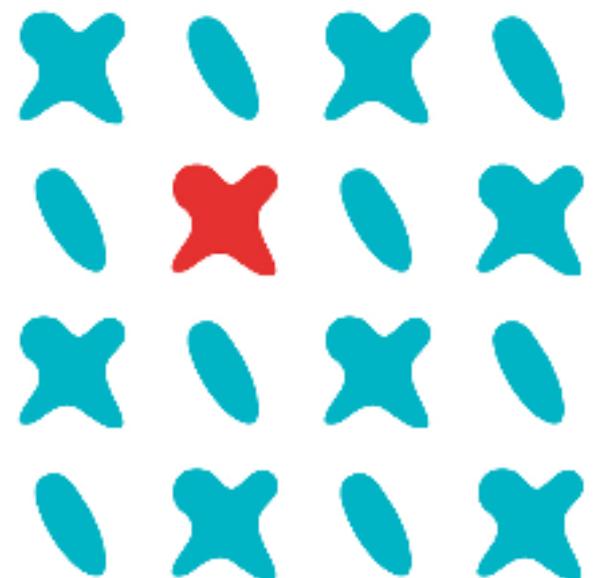
en.wikipedia.org/wiki/Conway%27s_law



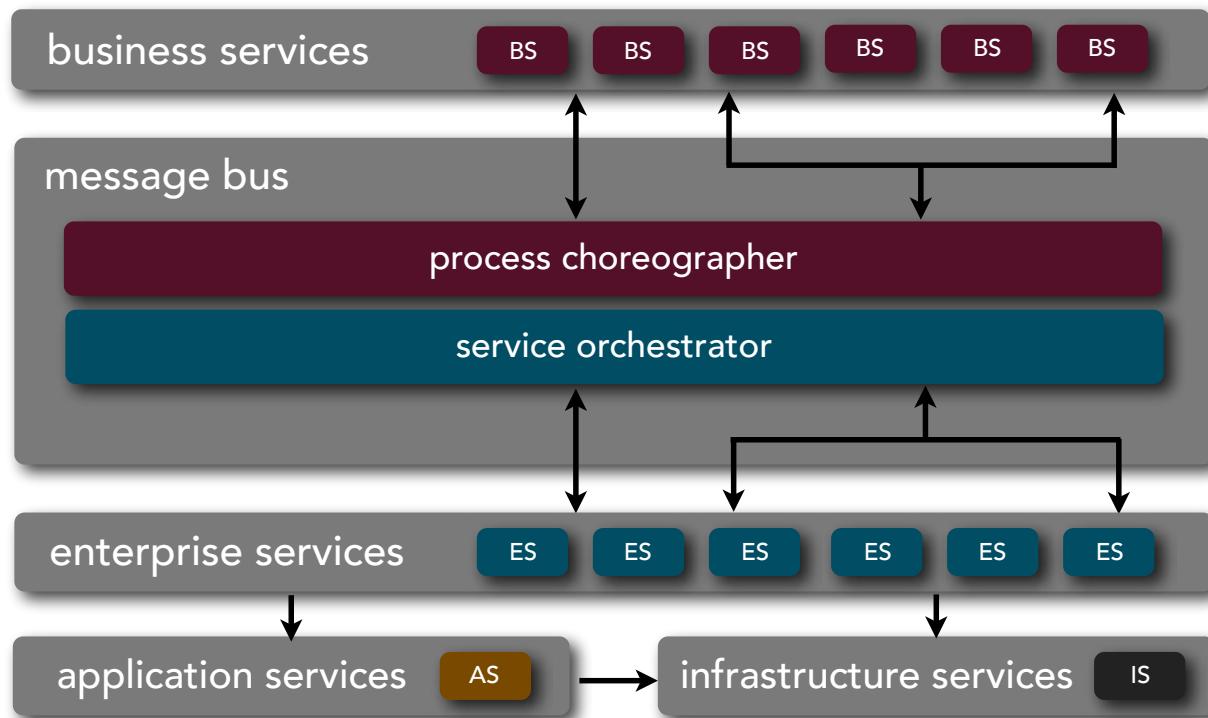
event-driven architecture



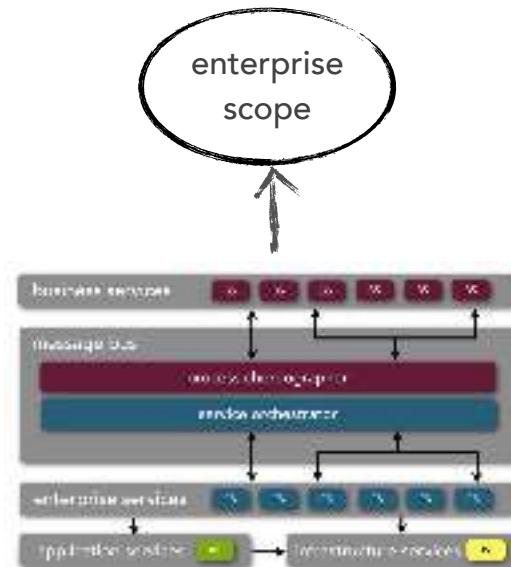
Service-oriented Architectures



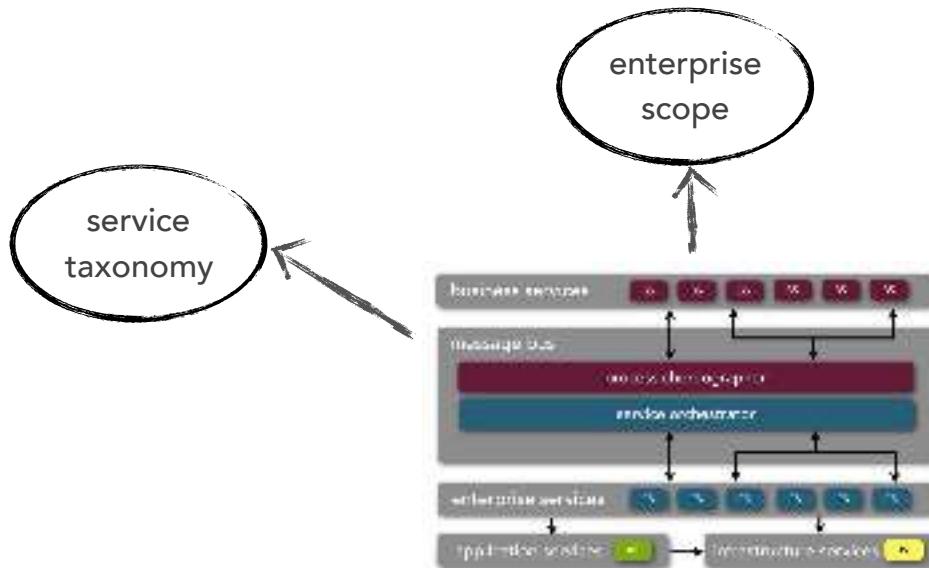
ESB-driven SOA



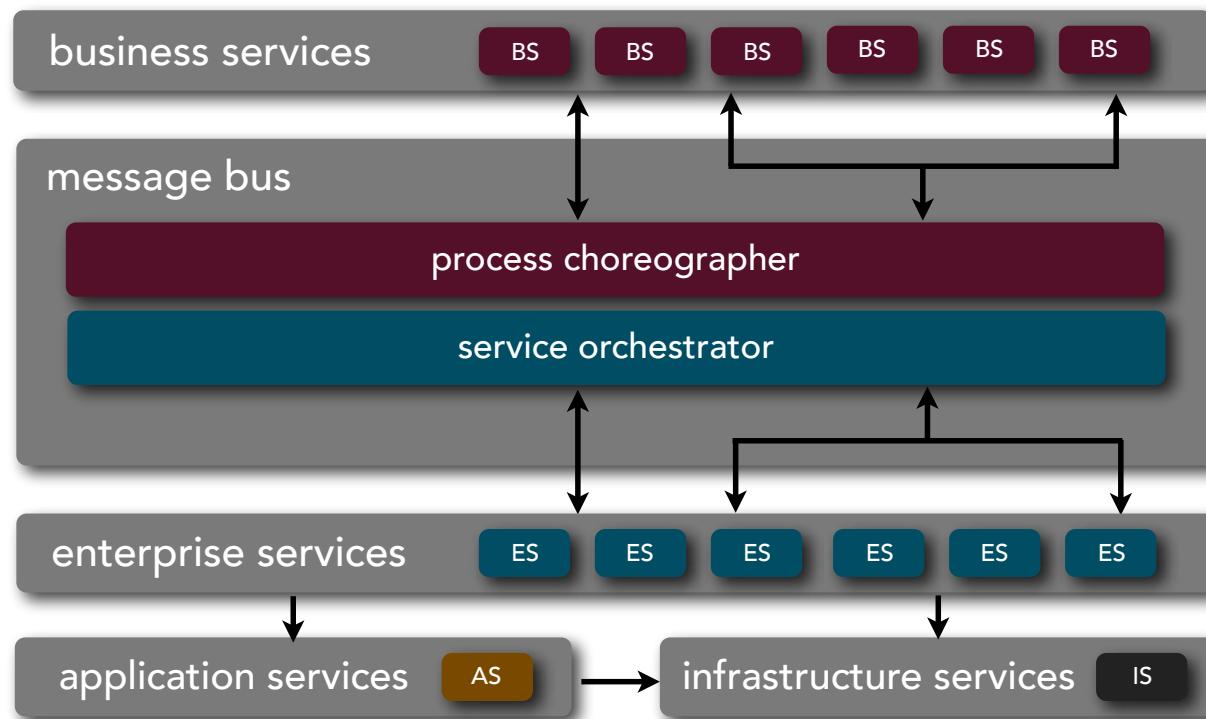
ESB-driven SOA



ESB-driven SOA

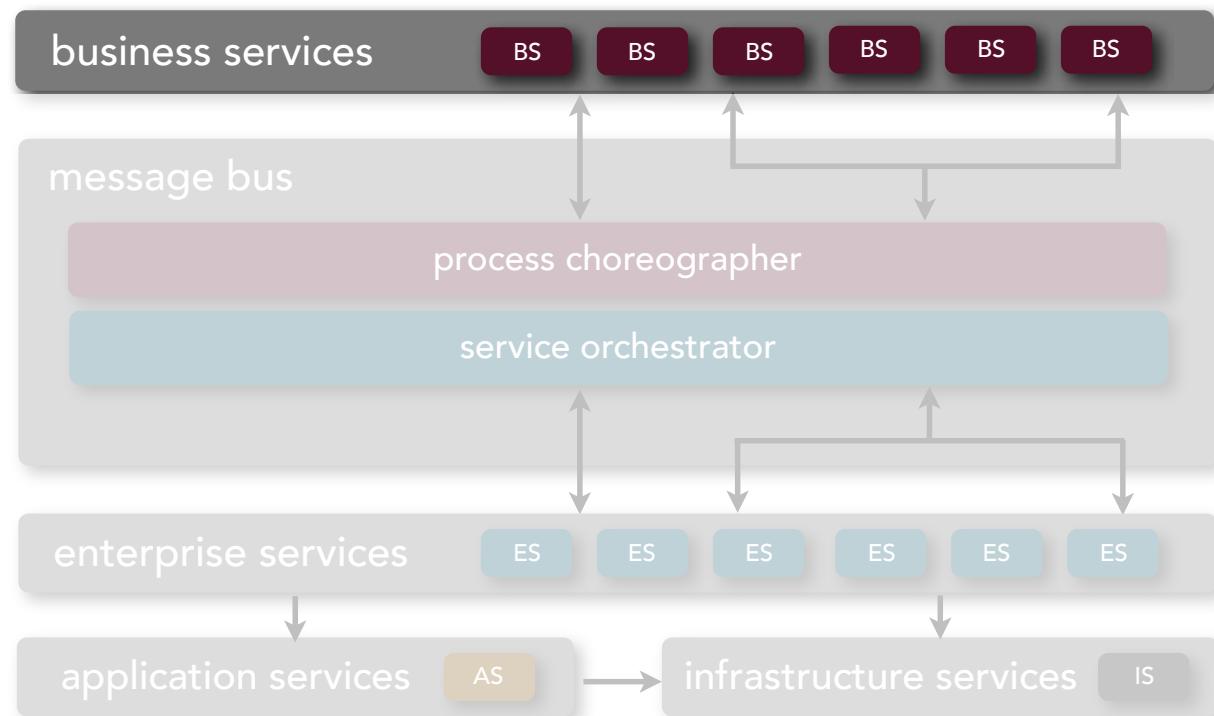


ESB-driven SOA



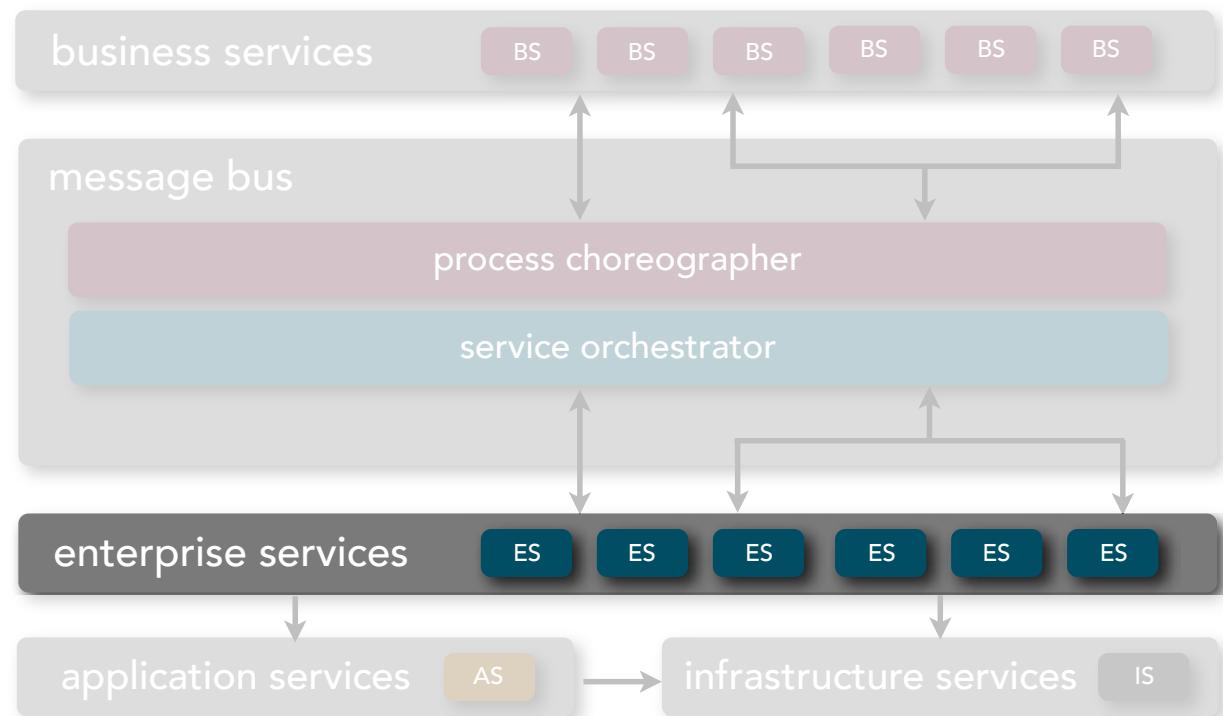
ESB-driven SOA

business services



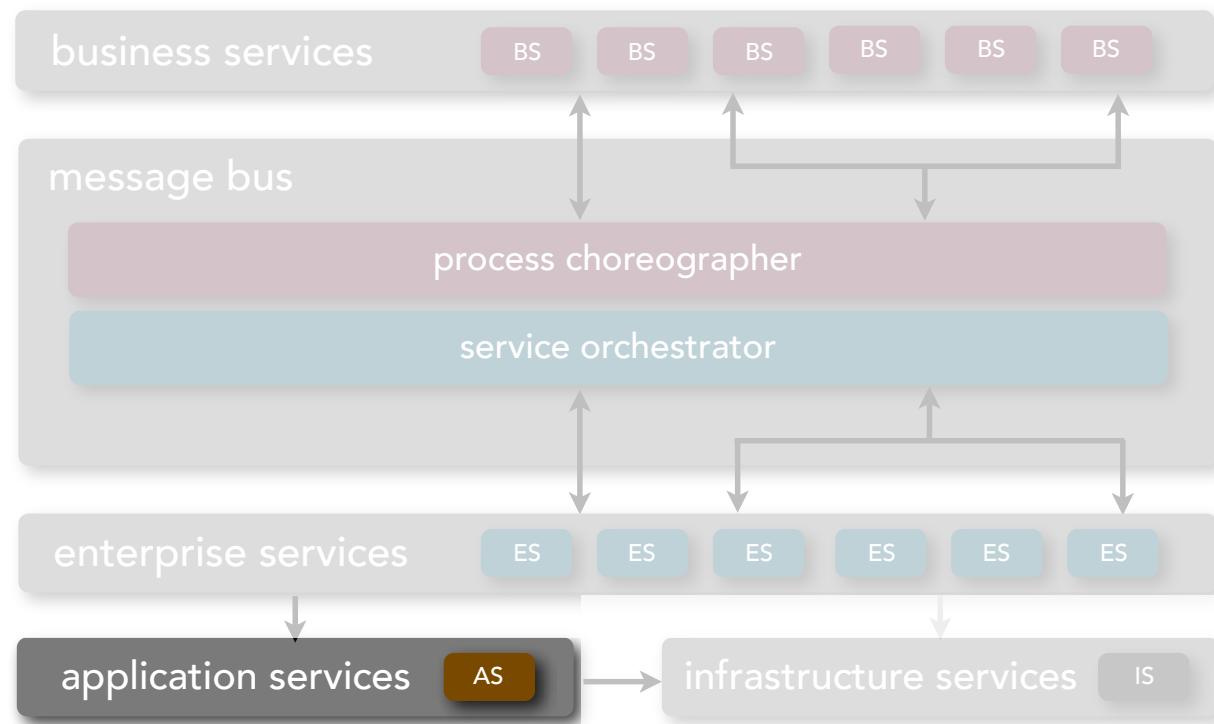
ESB-driven SOA

enterprise services



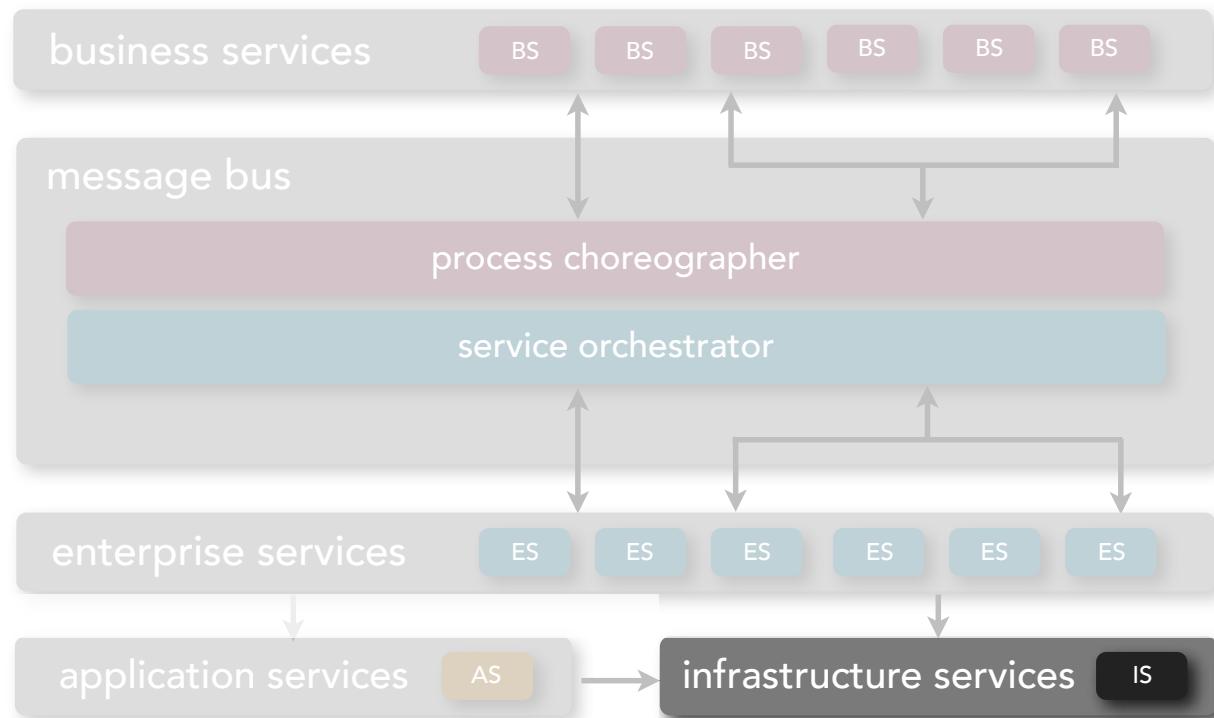
ESB-driven SOA

application services

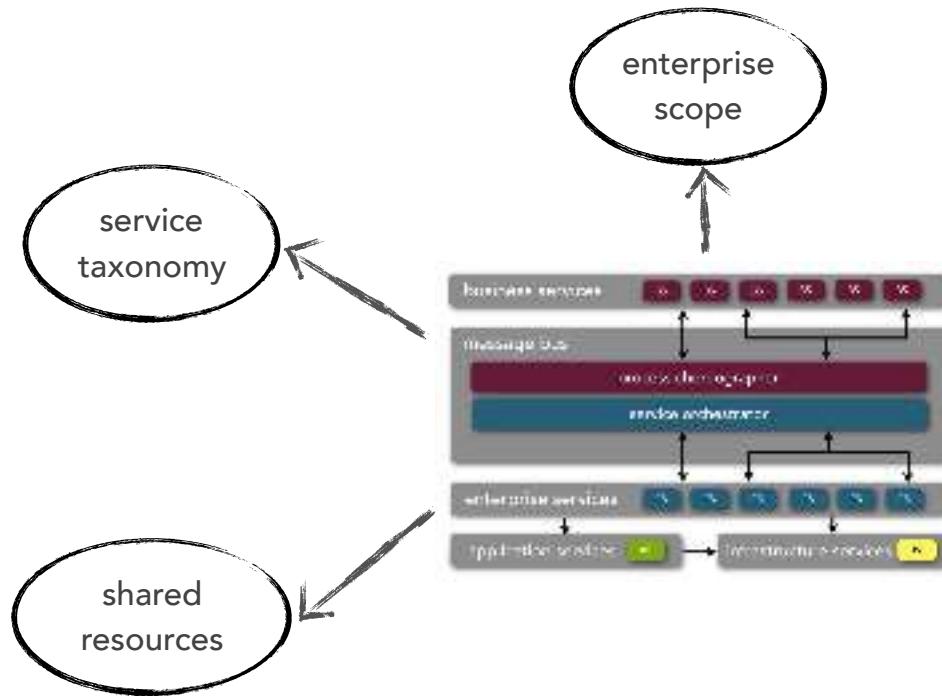


ESB-driven SOA

infrastructure services

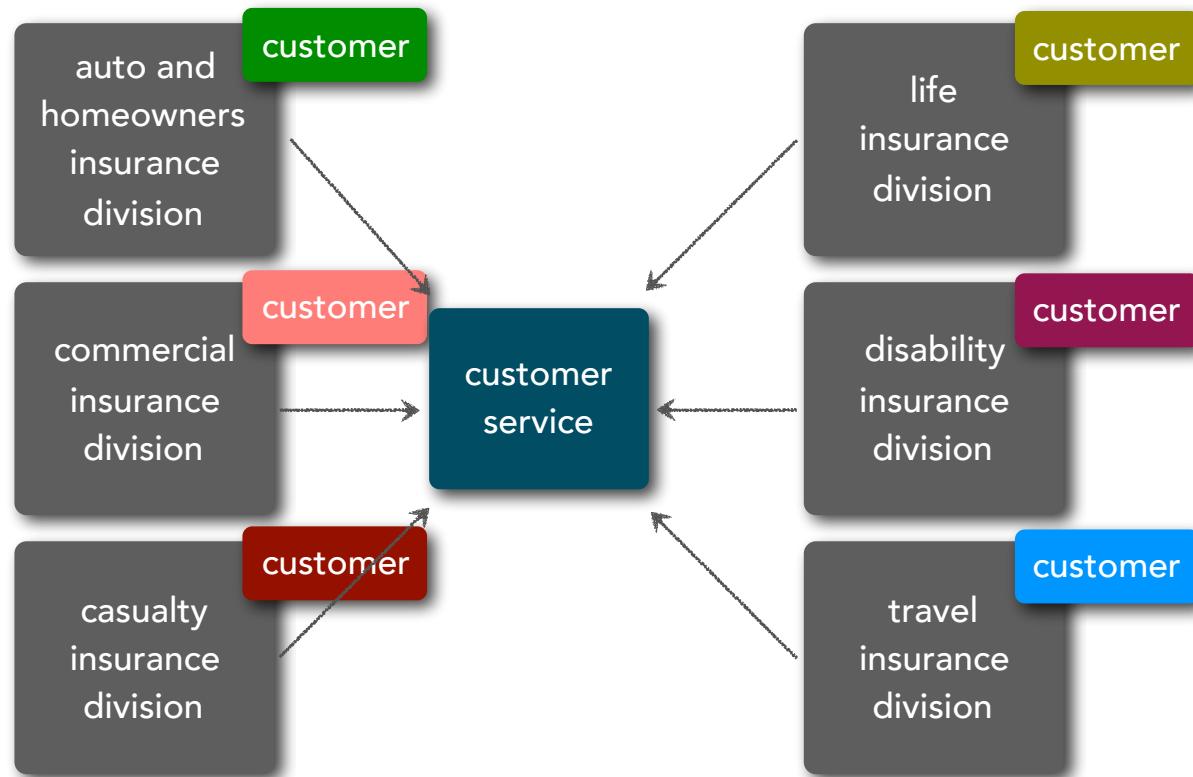


ESB-driven SOA

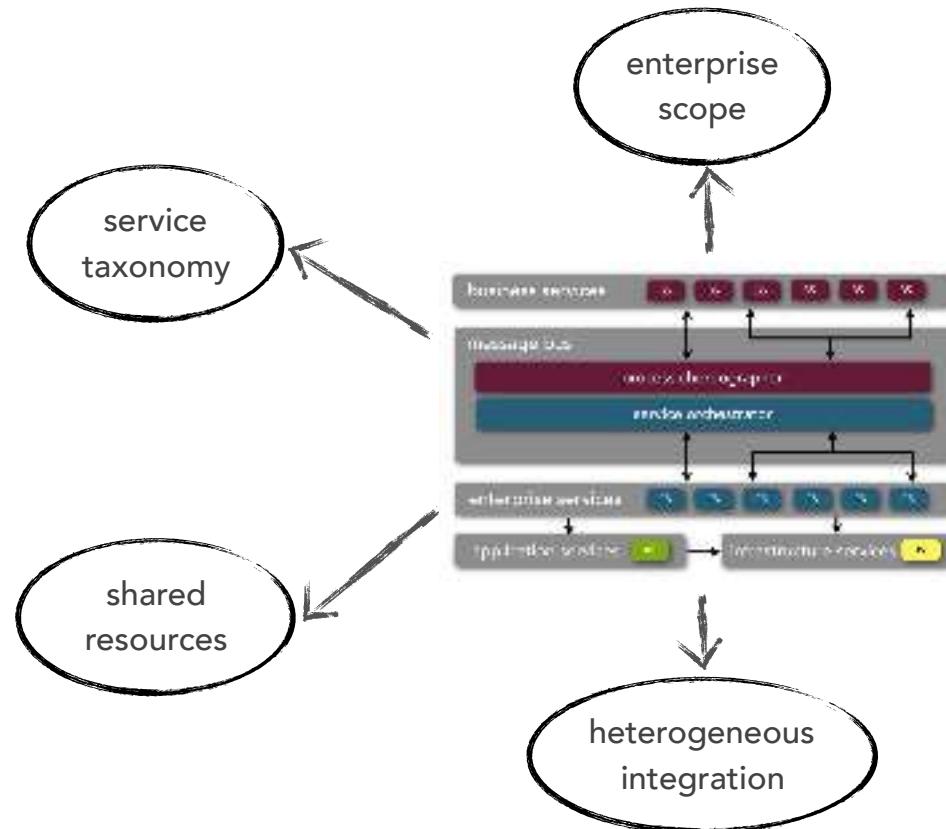


ESB-driven SOA

shared resources

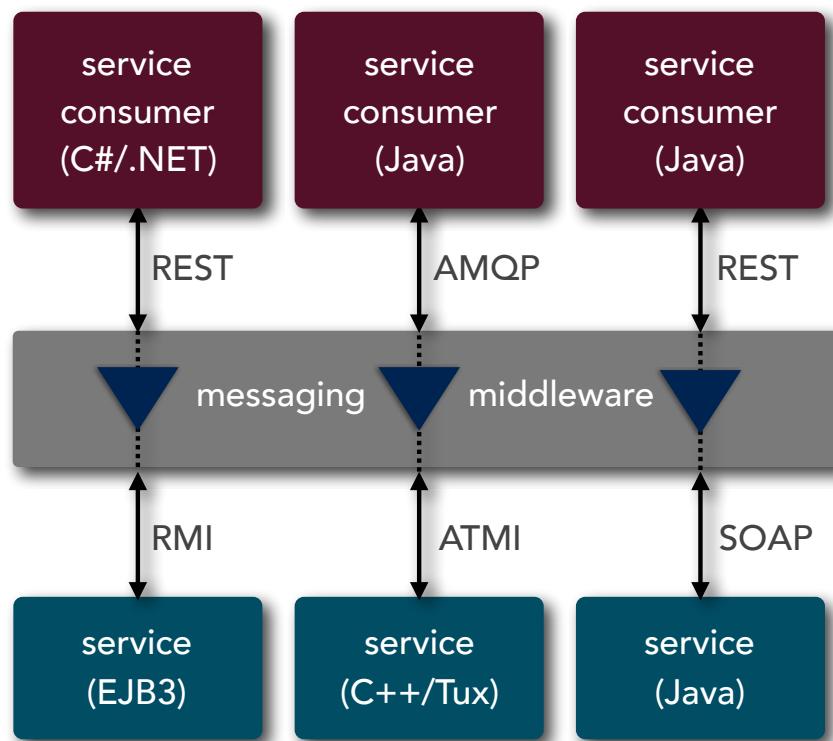


ESB-driven SOA

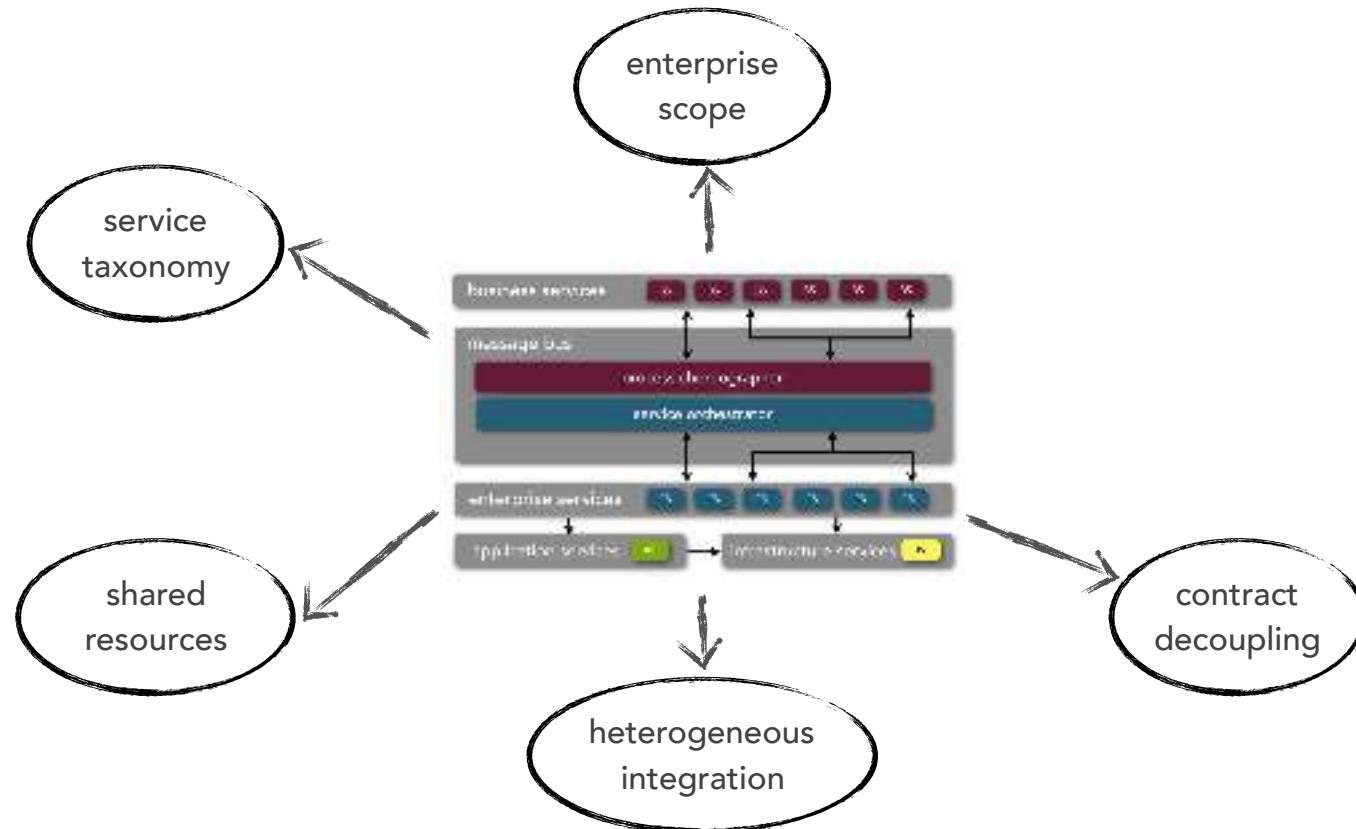


ESB-driven SOA

protocol-agnostic heterogeneous interoperability

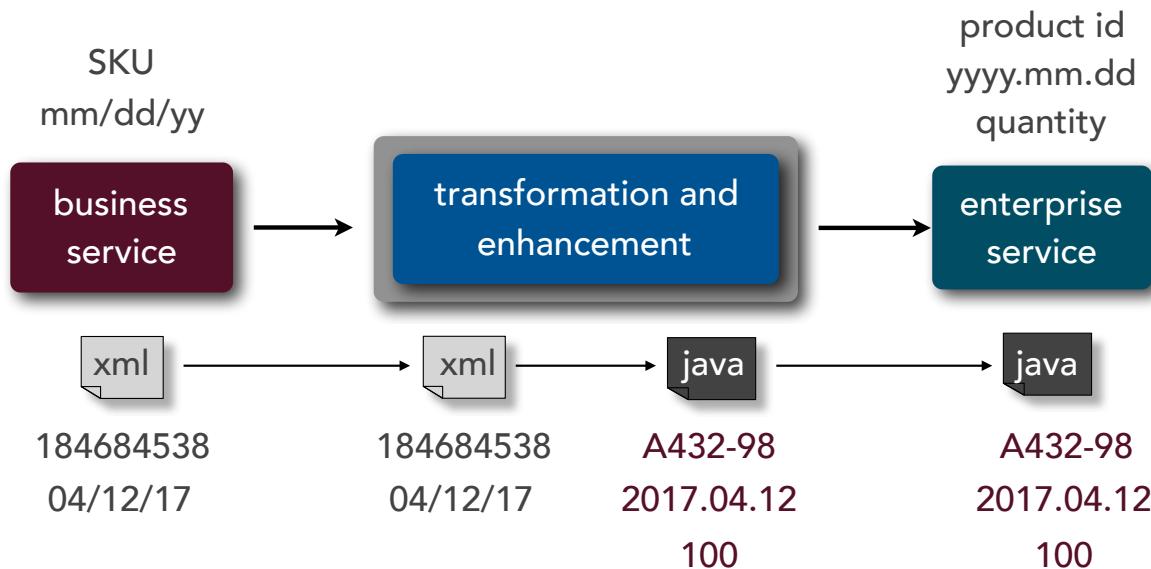


ESB-driven SOA

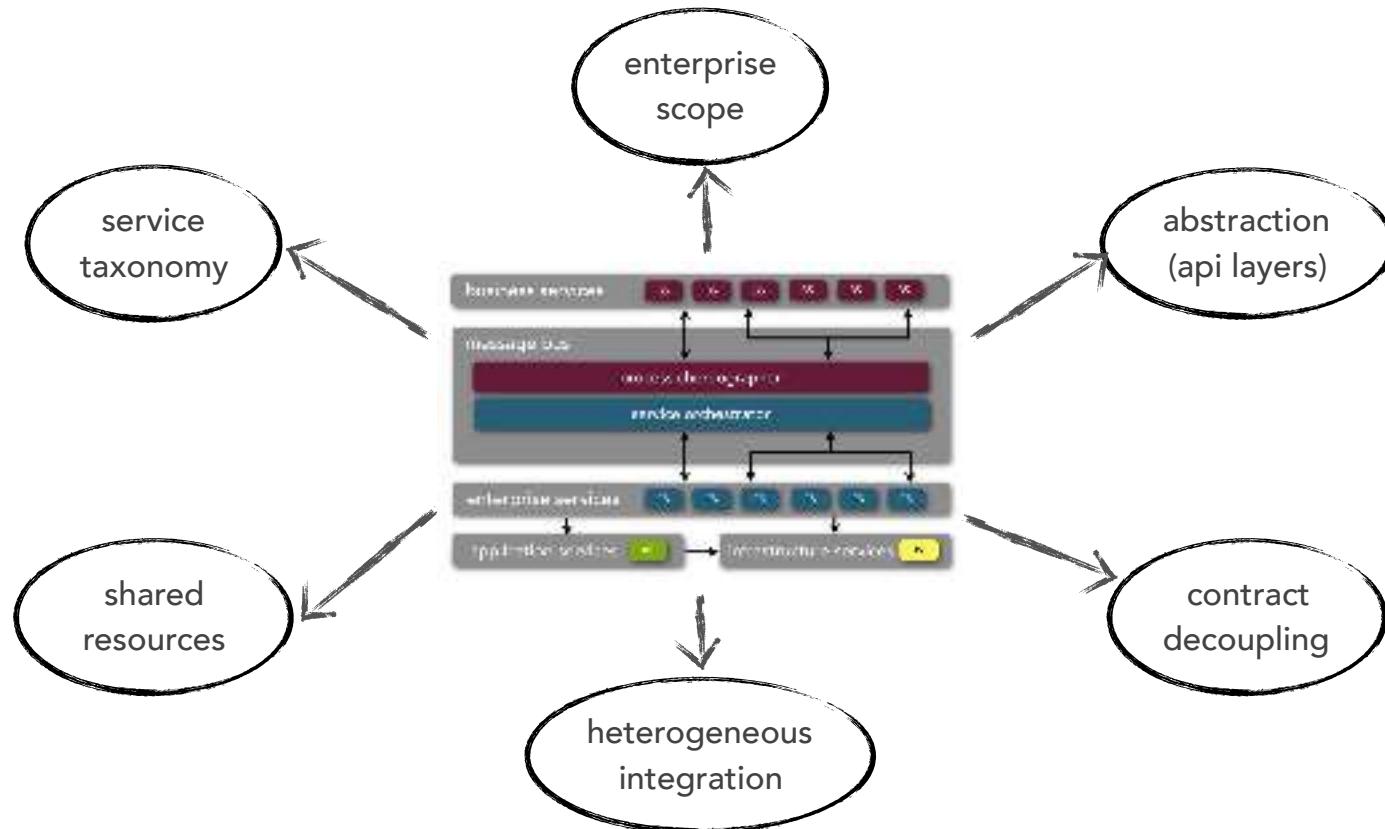


ESB-driven SOA

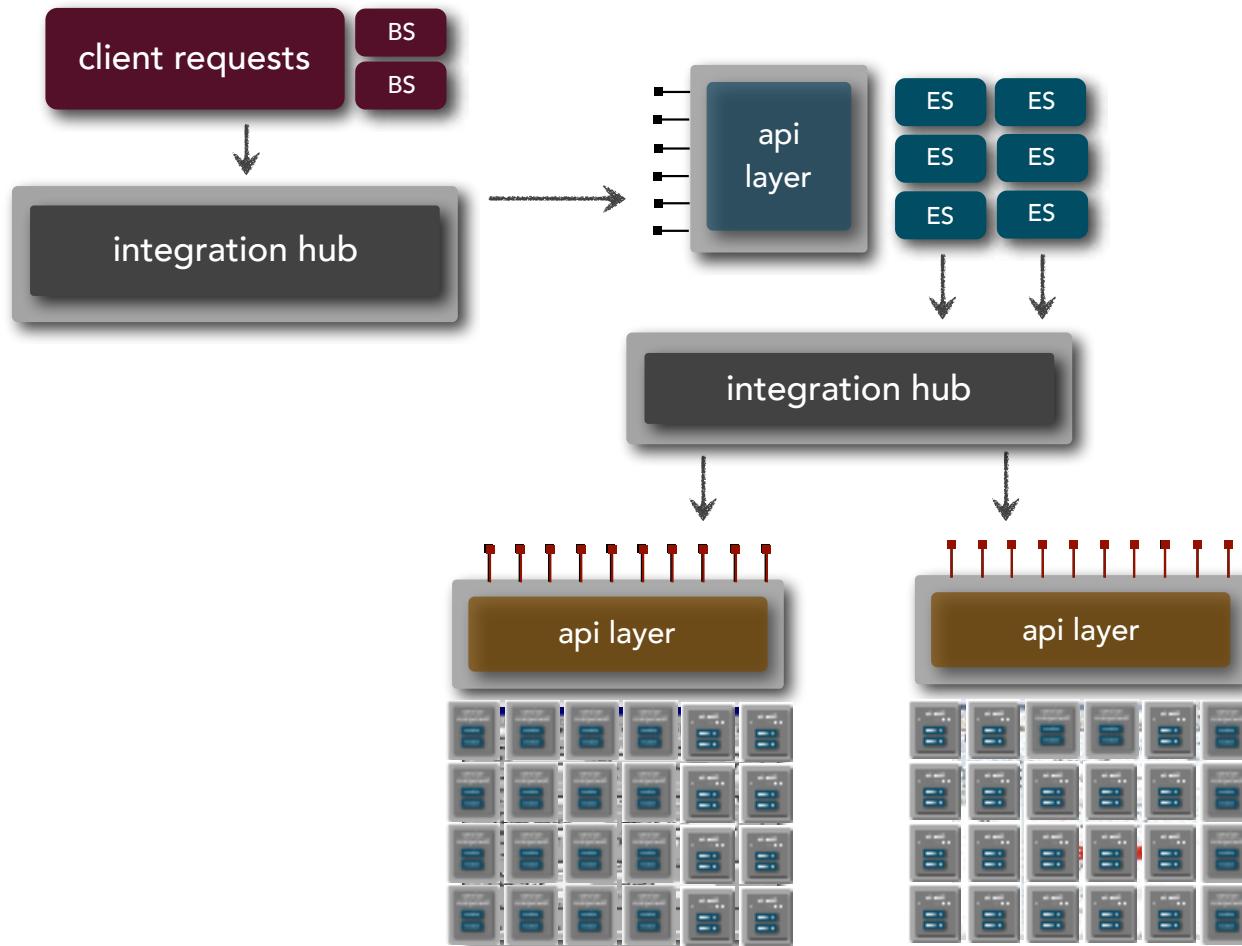
contract decoupling



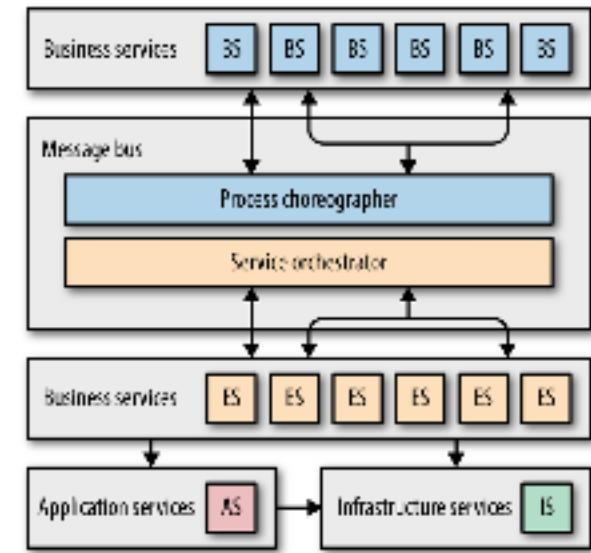
ESB-driven SOA



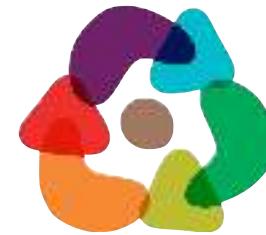
ESB-driven SOA



ESB-driven SOA

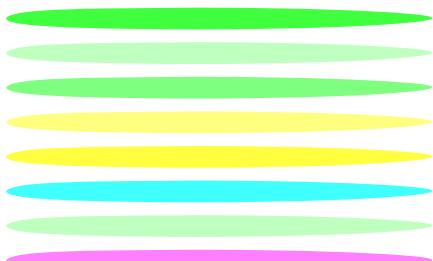


virtually impossible



the entire system

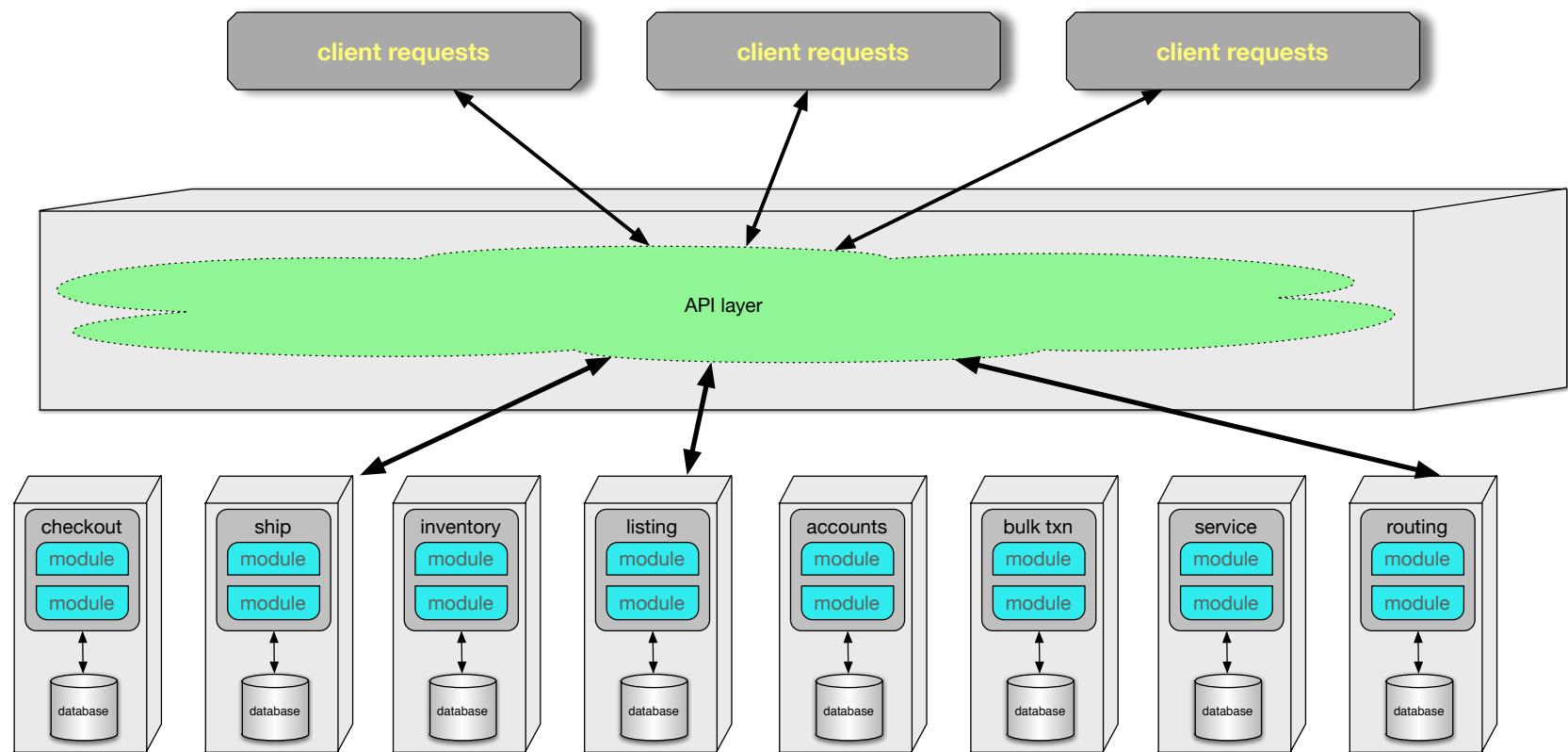
extremely difficult



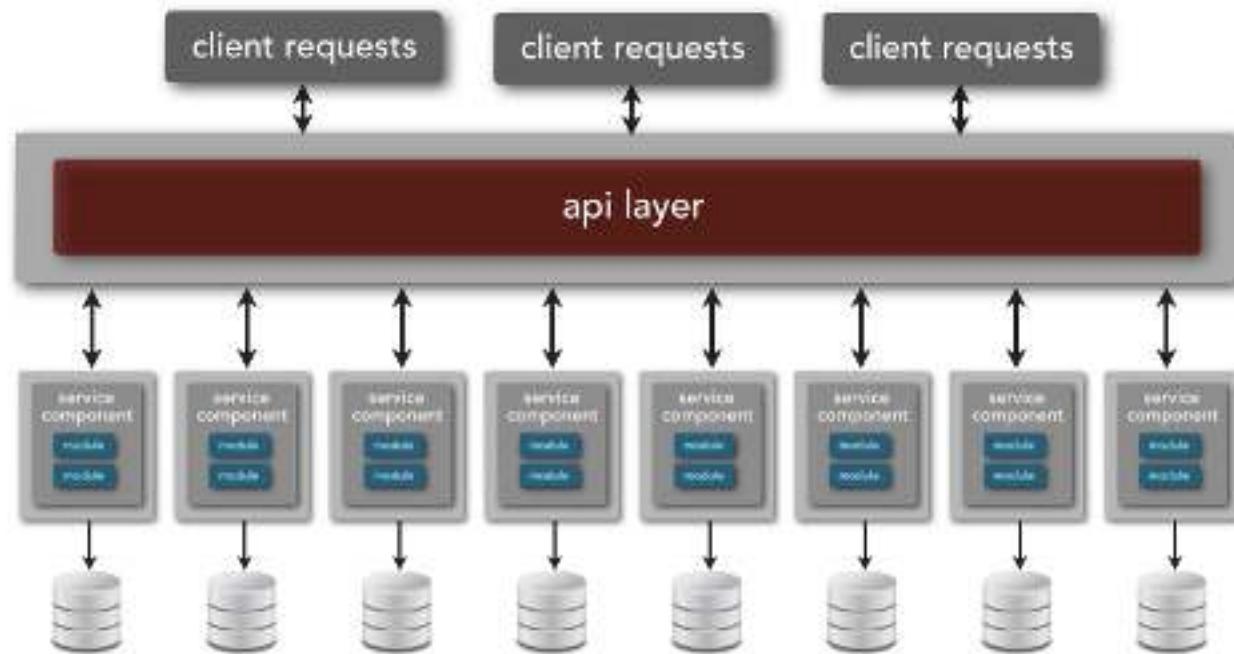
extravagant taxonomy makes sense if reuse is primary goal

extremely coupled in inappropriate ways.

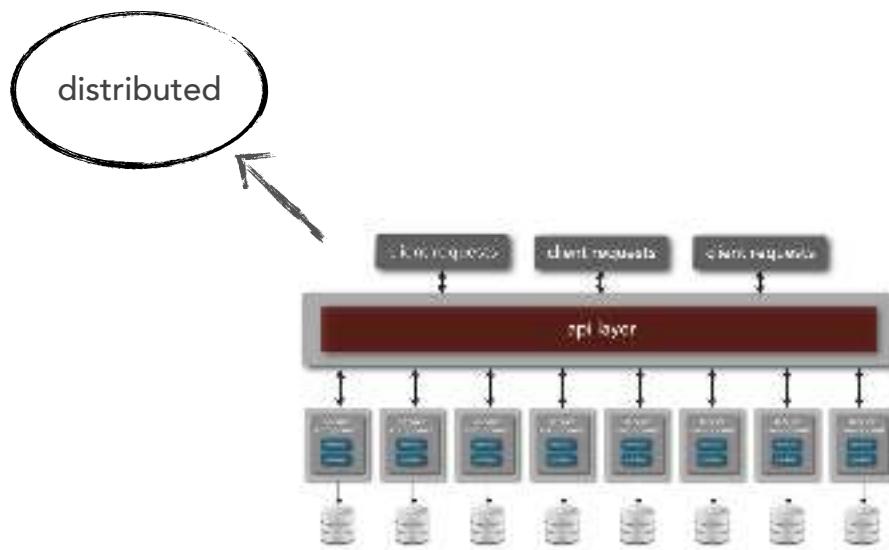
Microservices



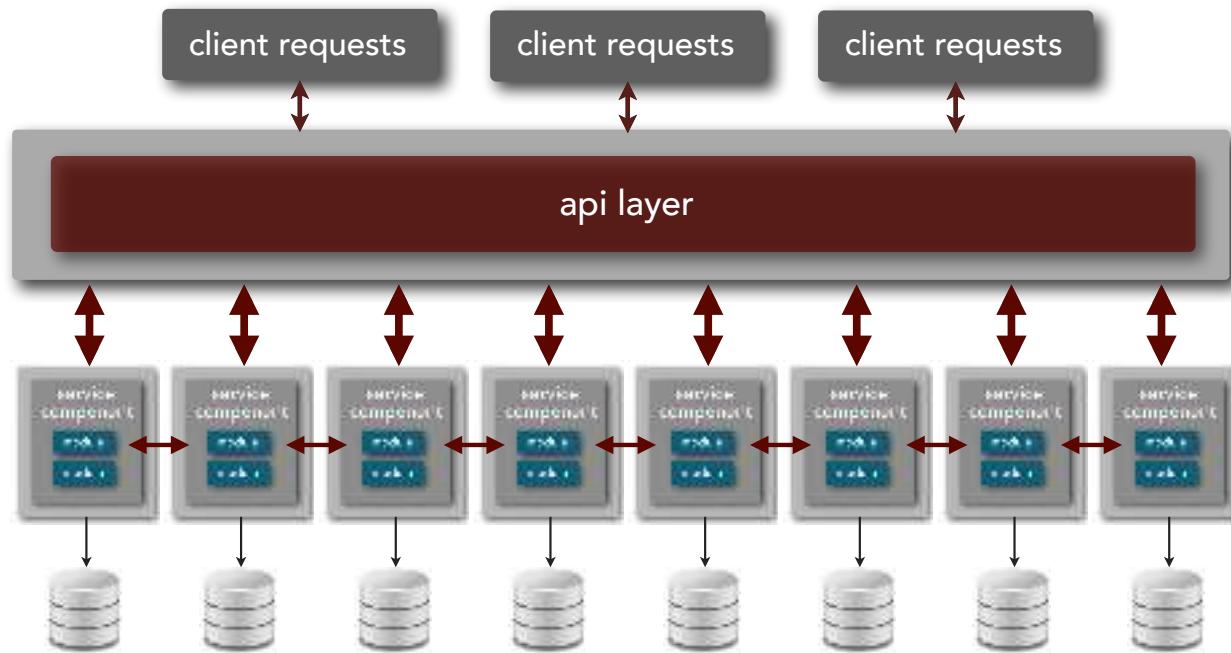
microservices architecture



microservices architecture

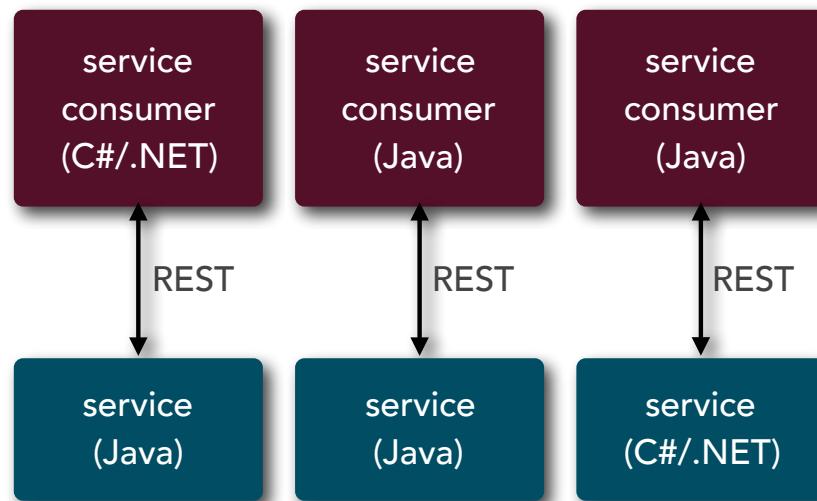


microservices architecture

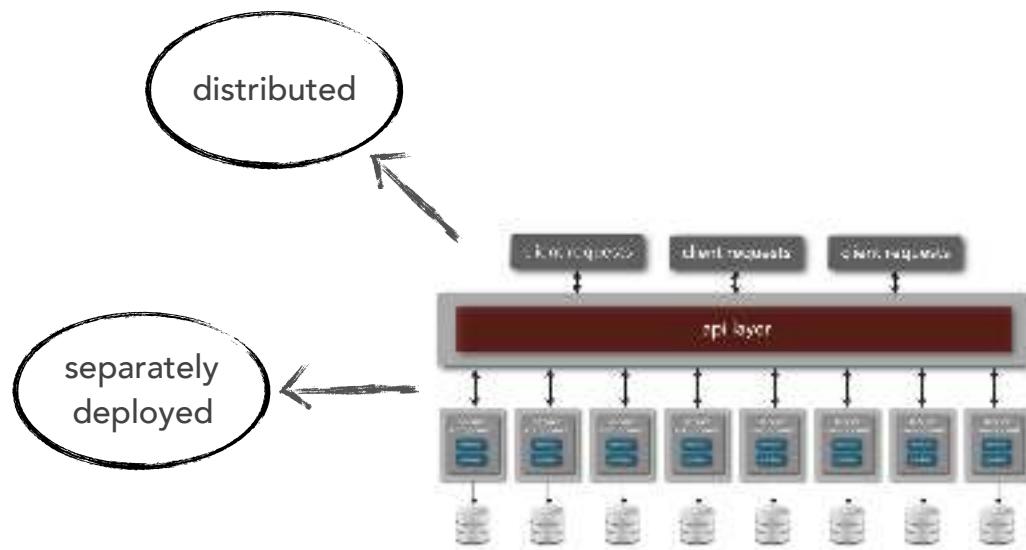


microservices architecture

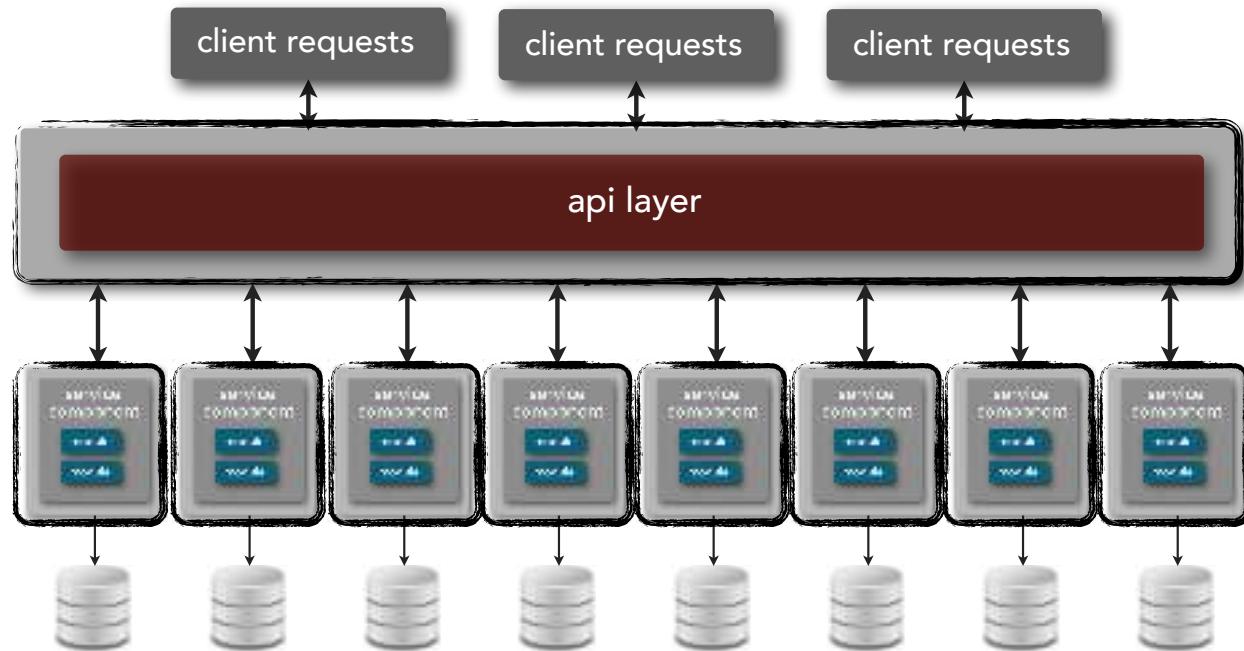
protocol-aware heterogeneous interoperability



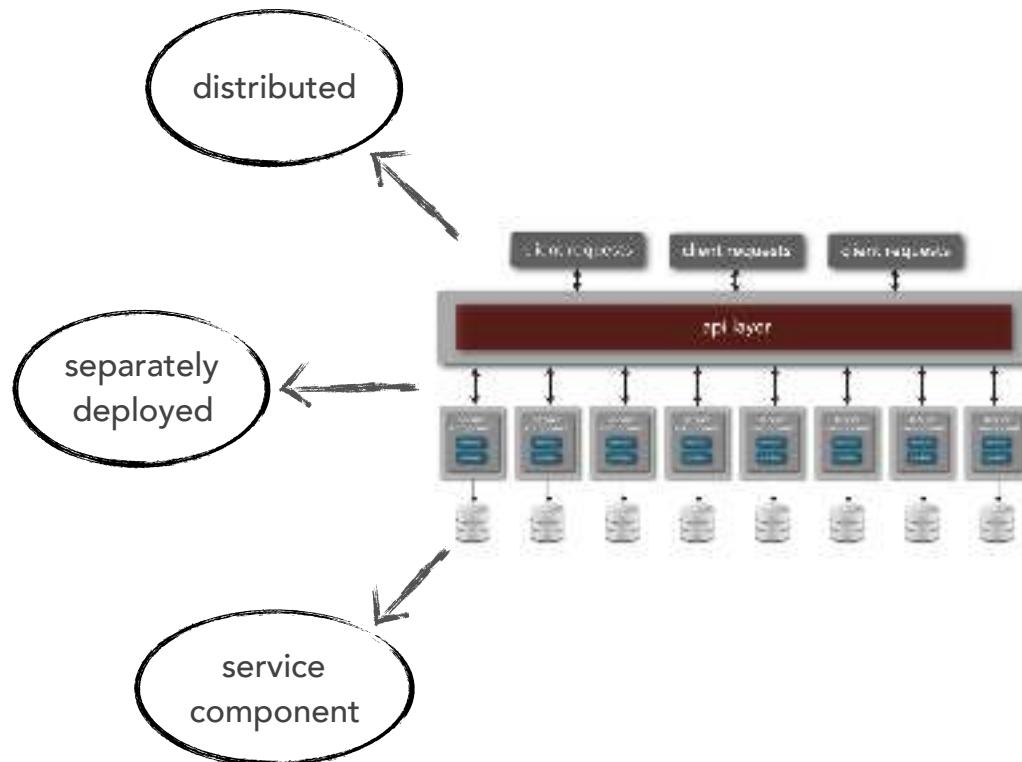
microservices architecture



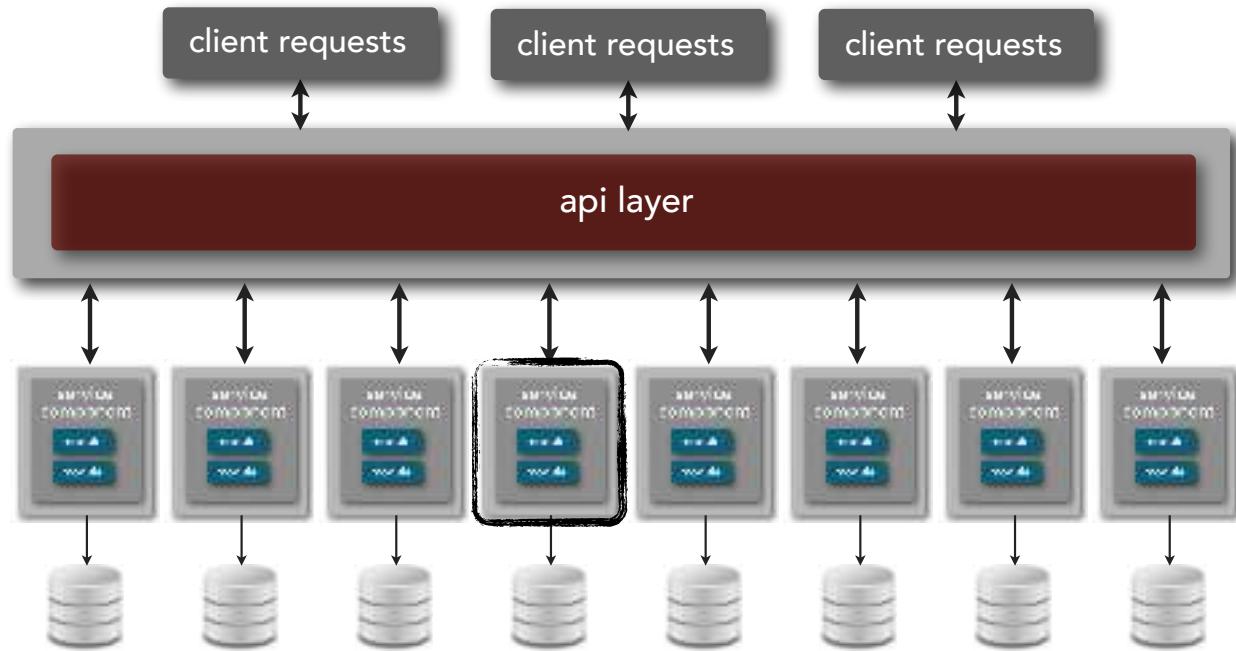
microservices architecture



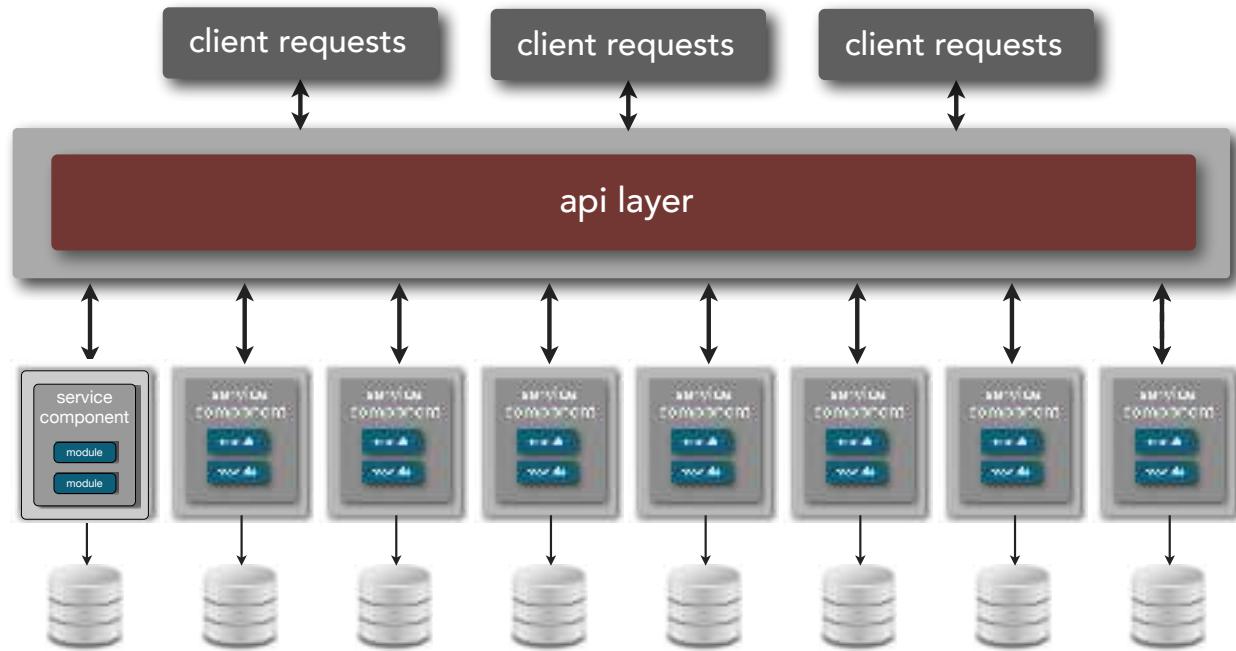
microservices architecture



microservices architecture



microservices architecture



service
component

module

module

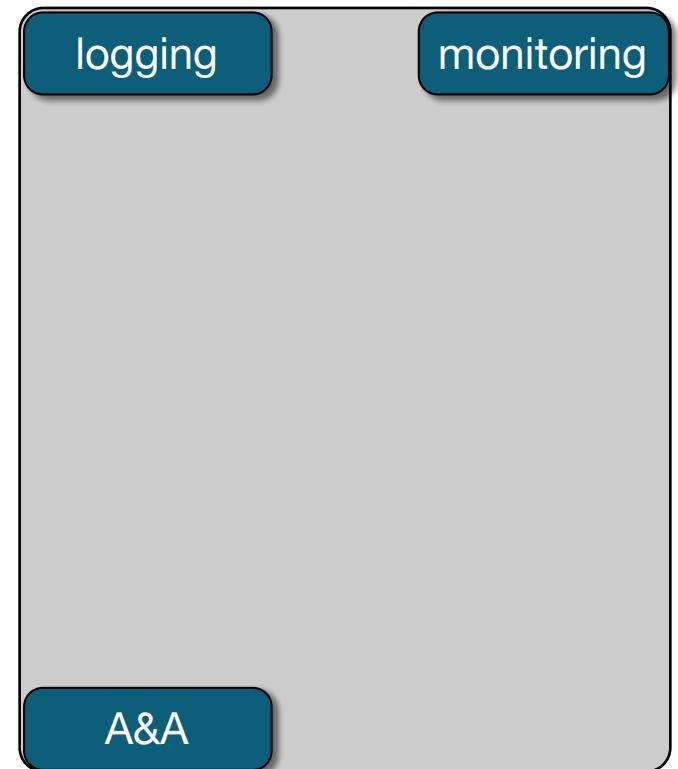
service templates



<https://projects.spring.io/spring-boot/>



<http://www.dropwizard.io/>



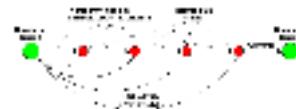
microservices architecture



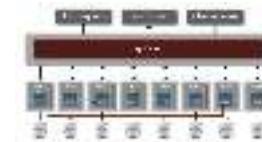
what is the right size for a microservice?



purpose



transactions



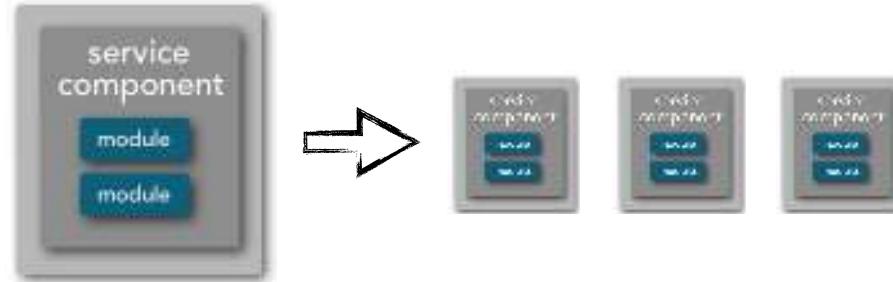
choreography

“Microservice” is a *label*, not a *description*.

microservices architecture

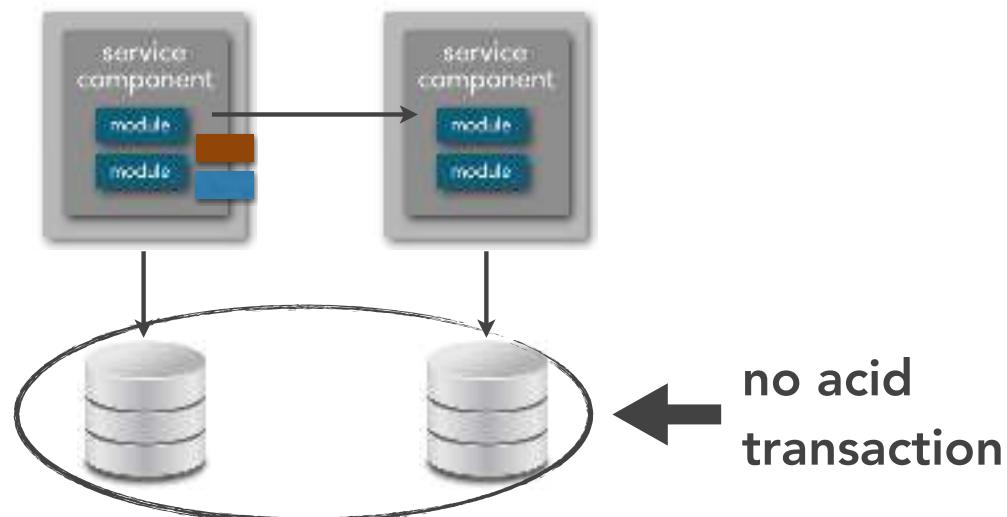


purpose



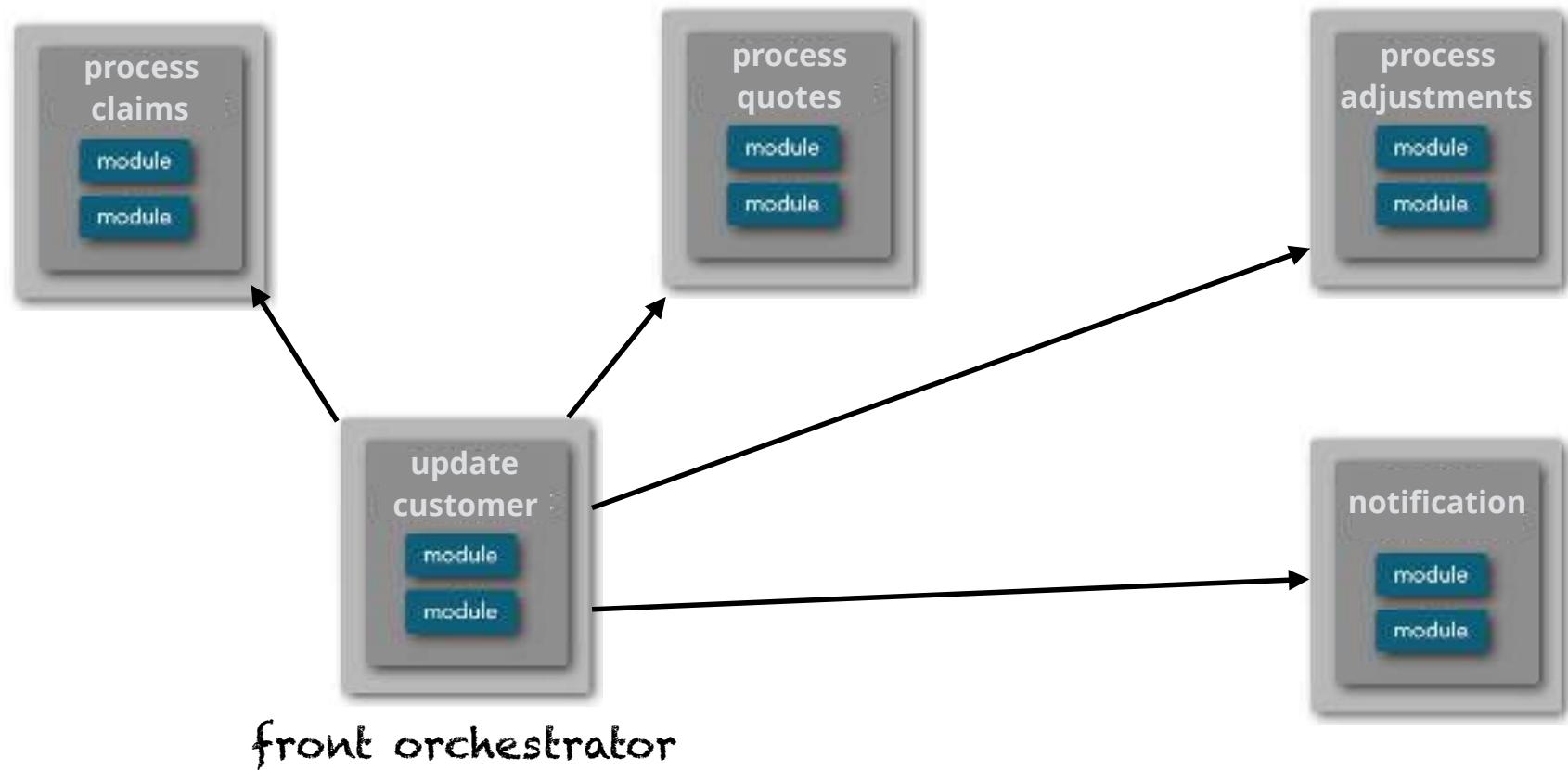
service scope and function
(single-purpose function)

microservices architecture



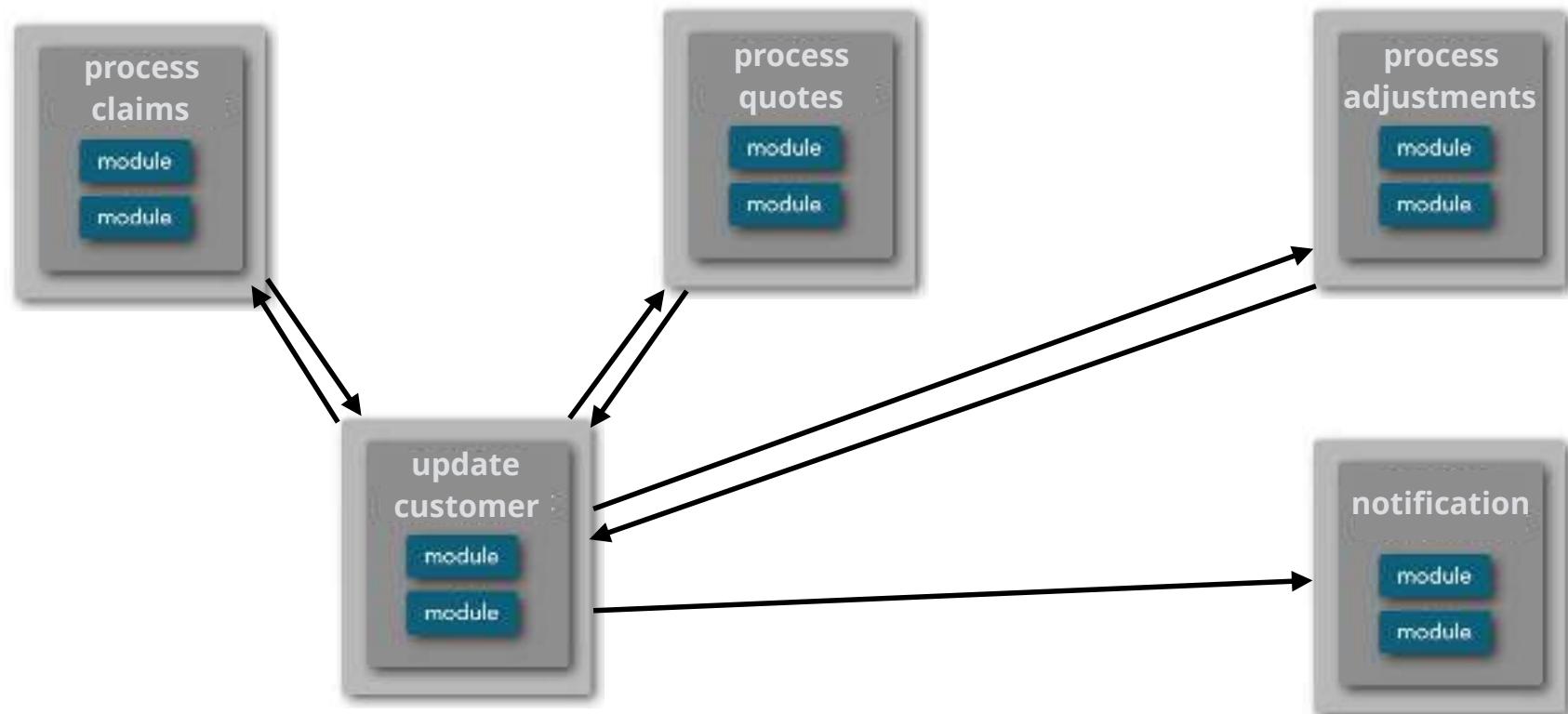
microservices architecture

service orchestration



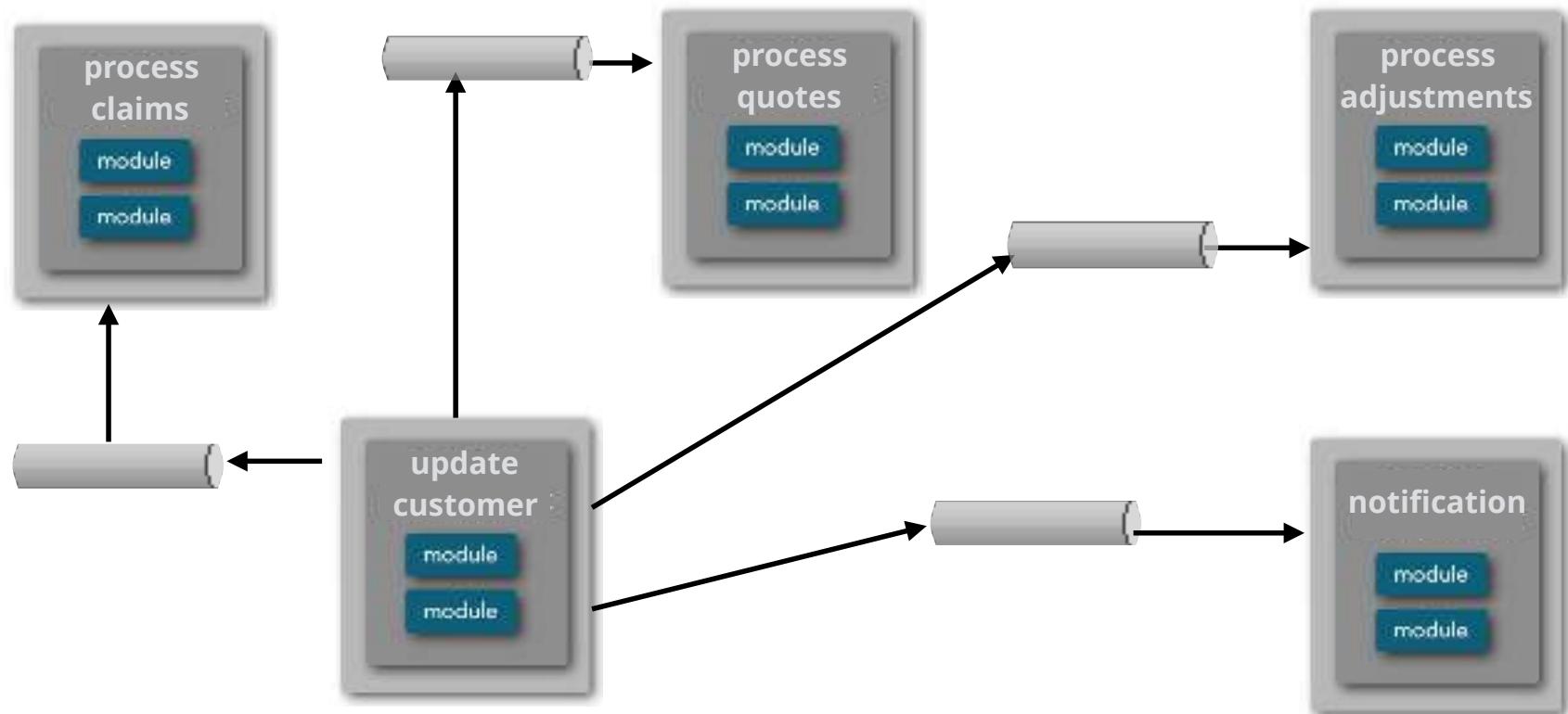
microservices architecture

service orchestration



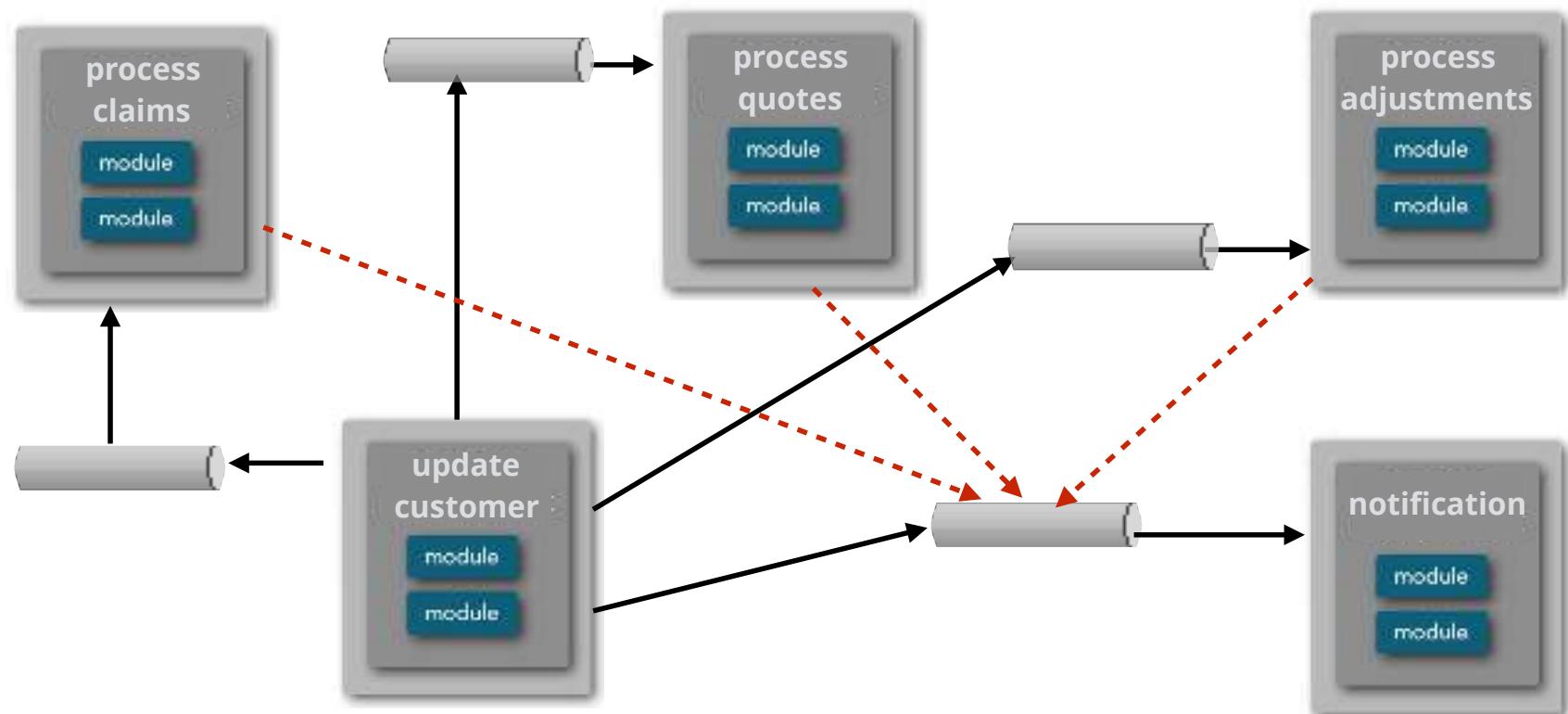
microservices architecture

service orchestration



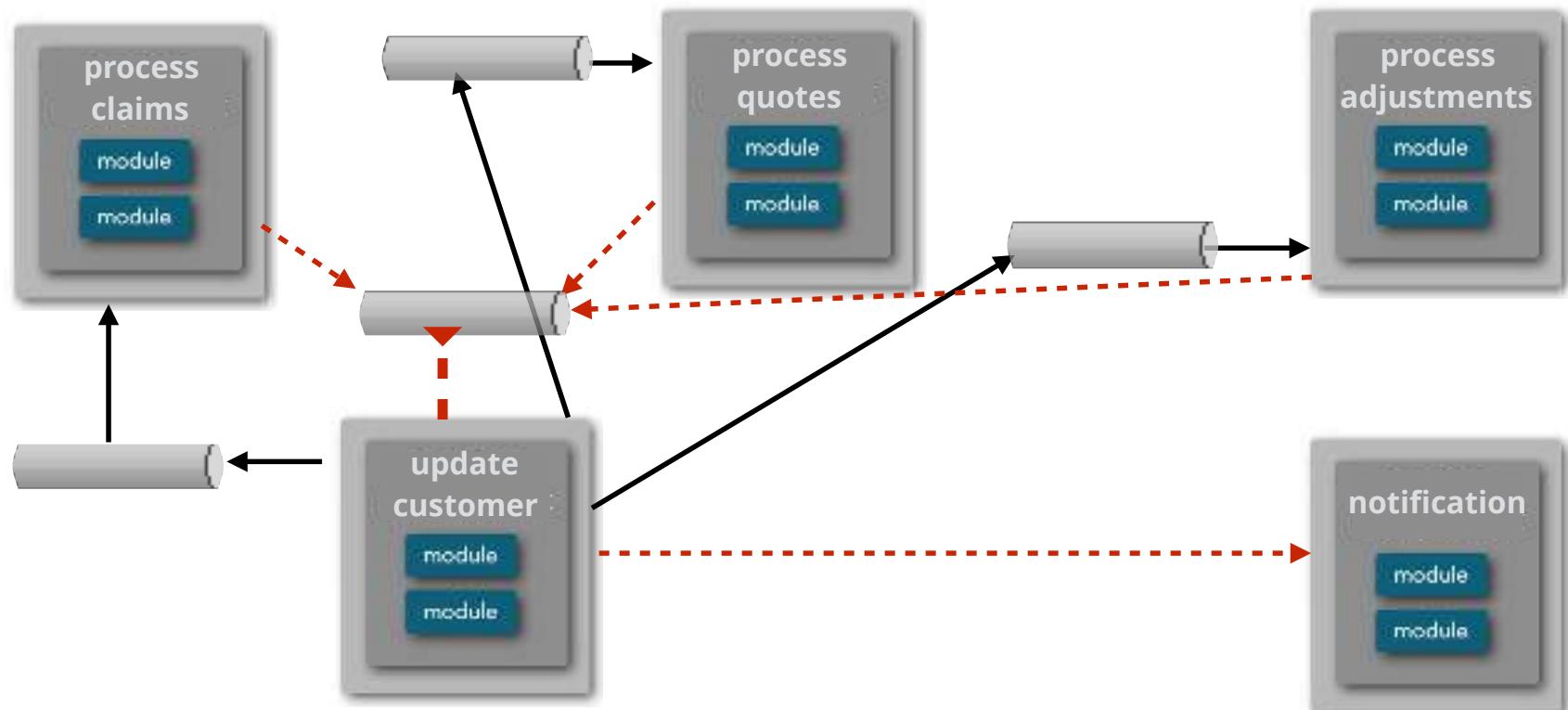
microservices architecture

service orchestration

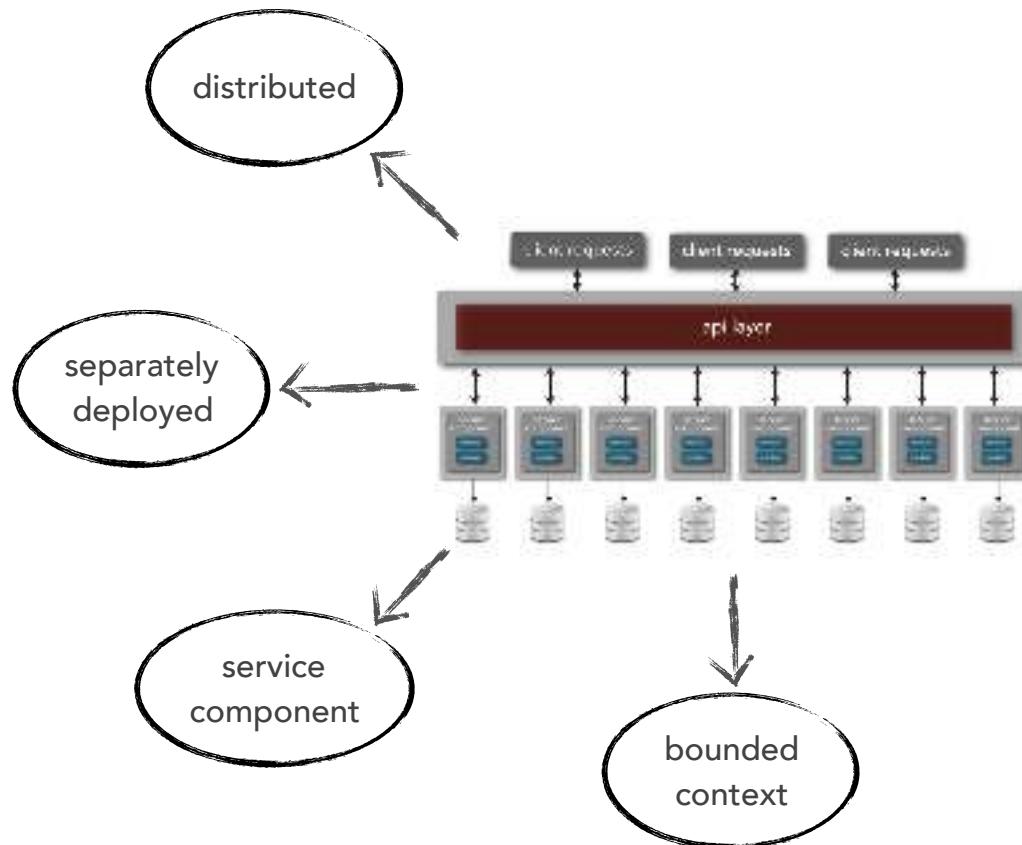


microservices architecture

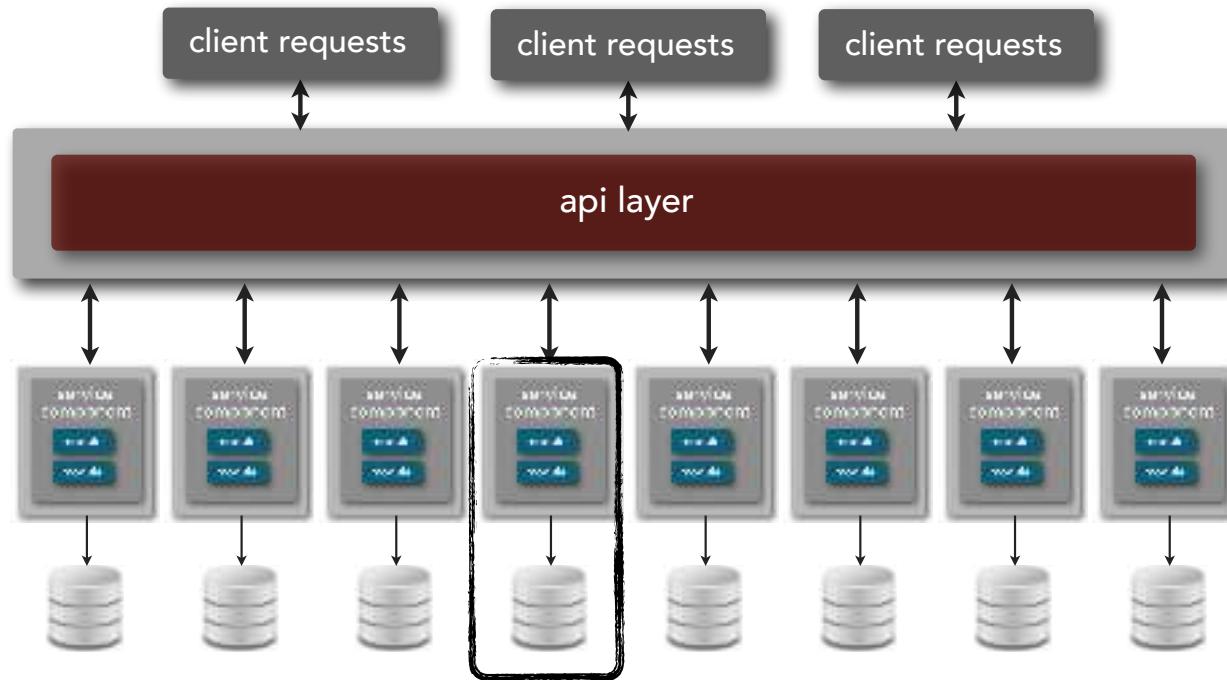
service orchestration



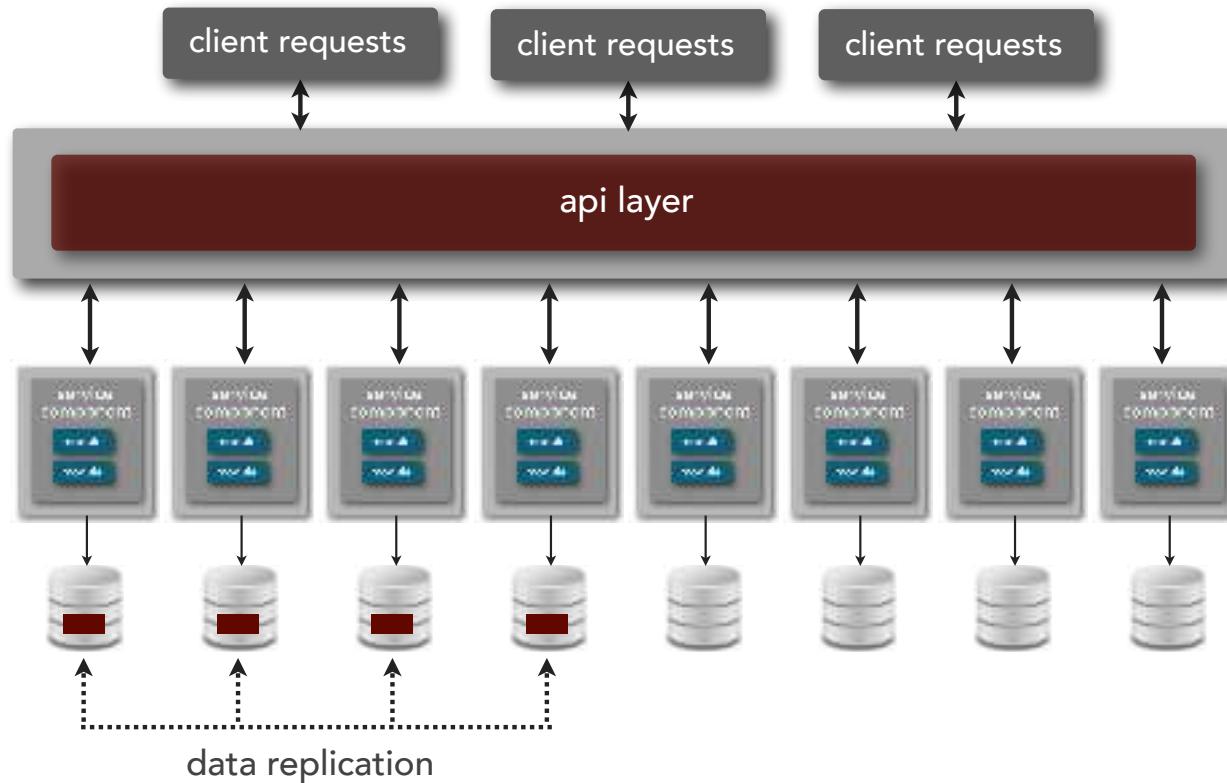
microservices architecture



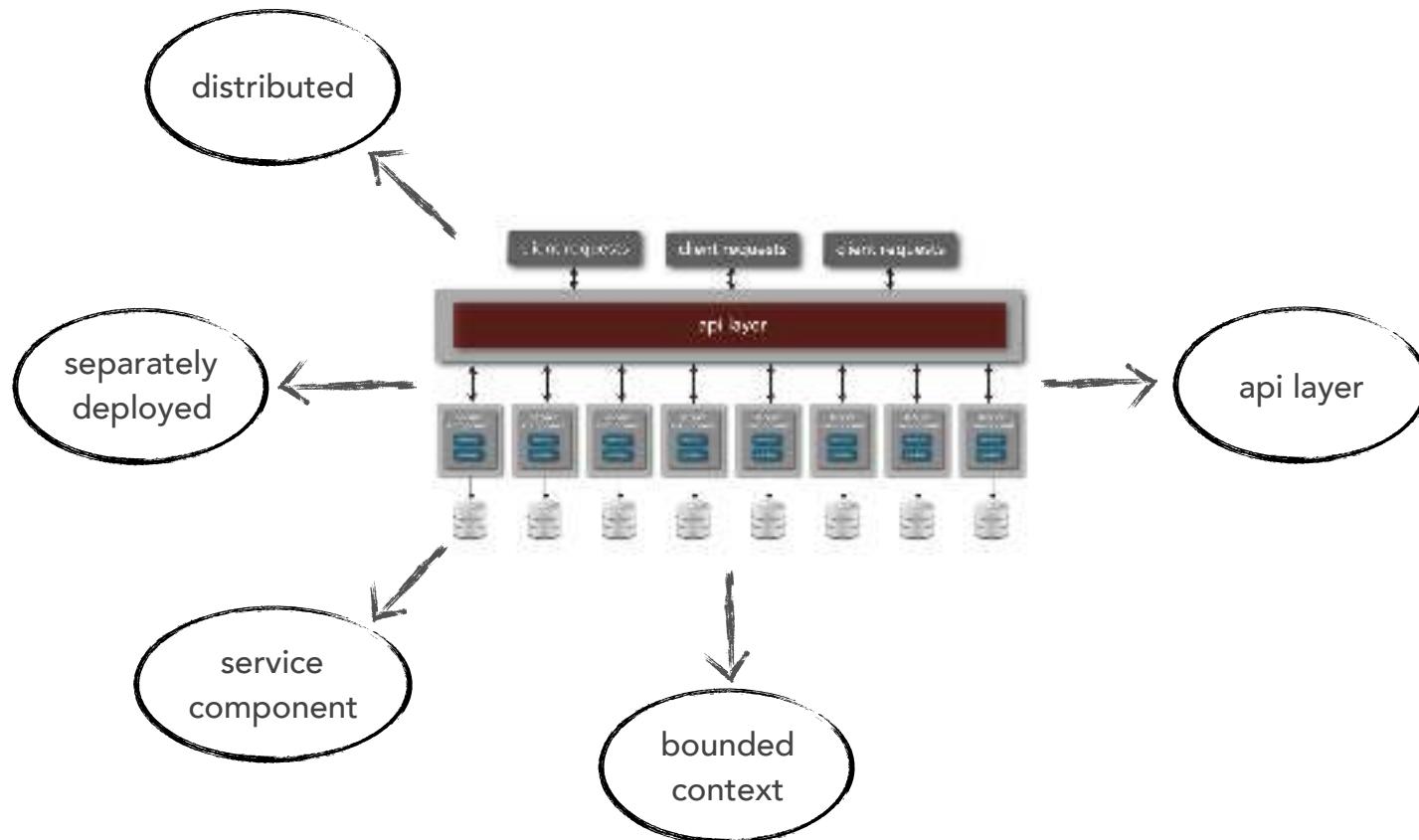
microservices architecture



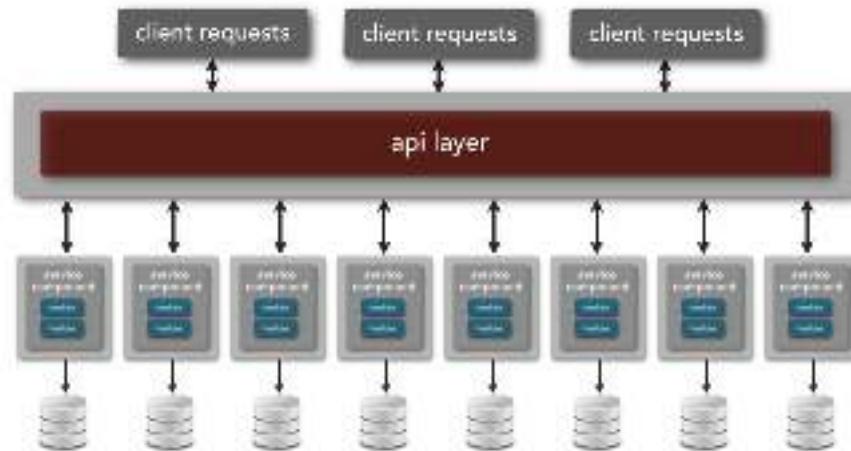
microservices architecture



microservices architecture

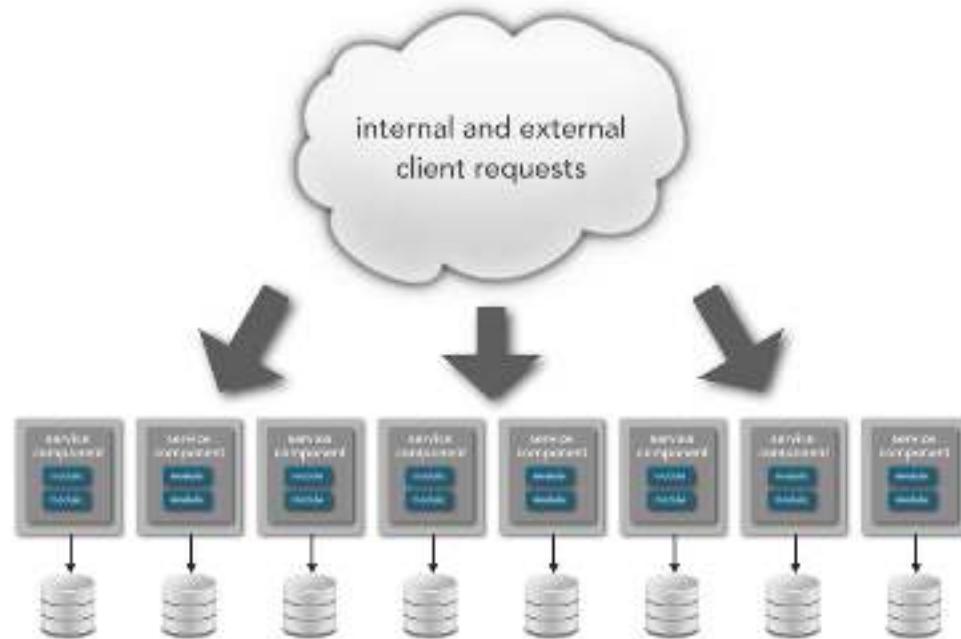


api layer

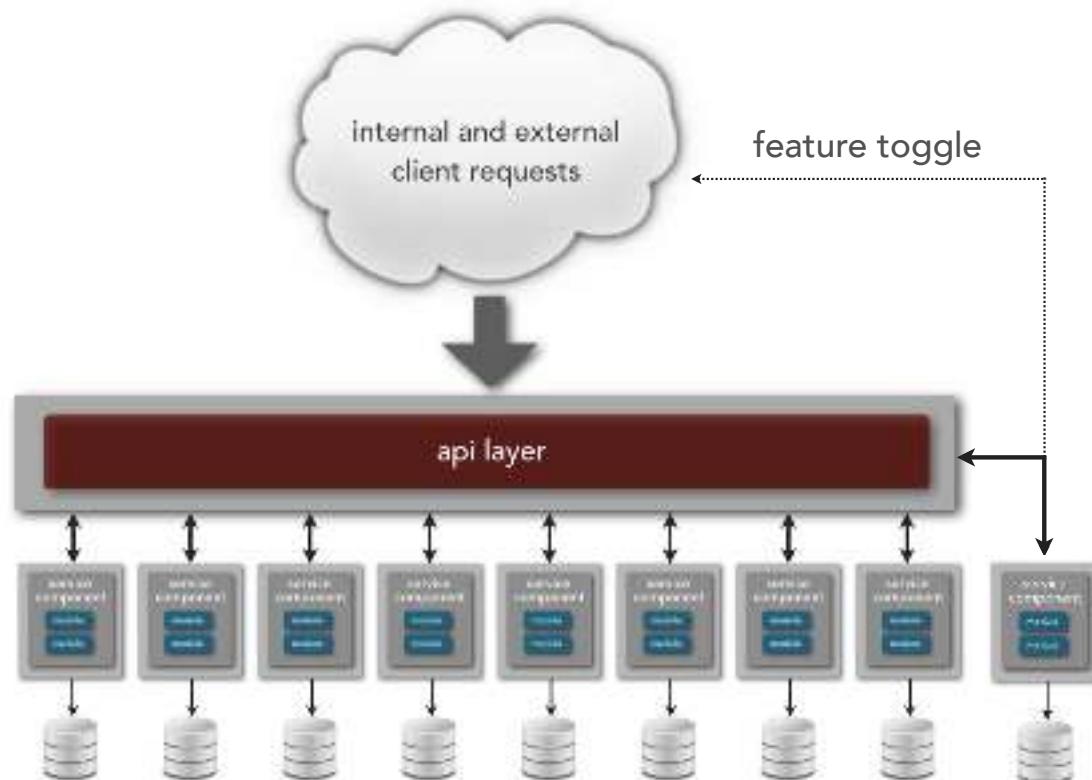


hides the actual endpoint of the service, exposing
only those services available for public consumption

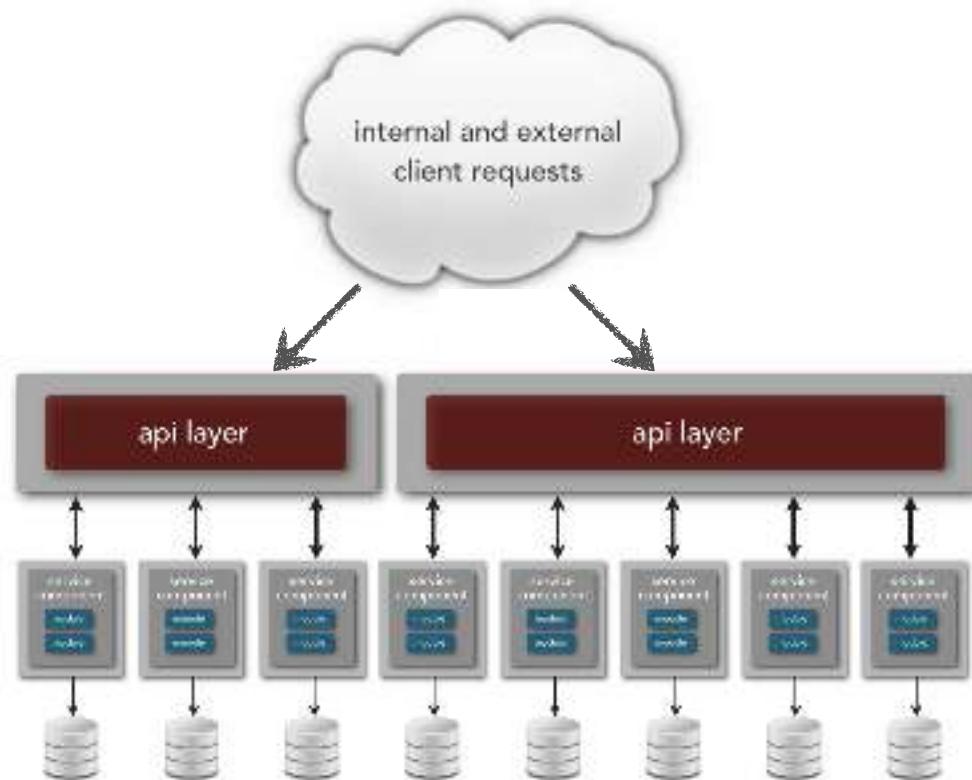
api layer



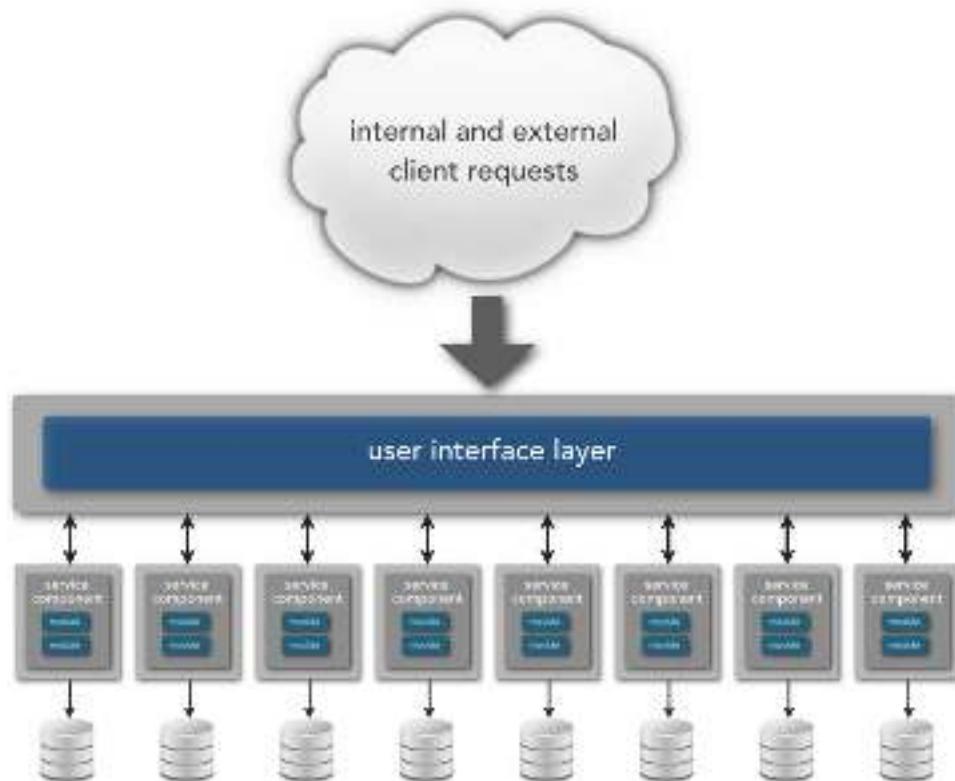
api layer



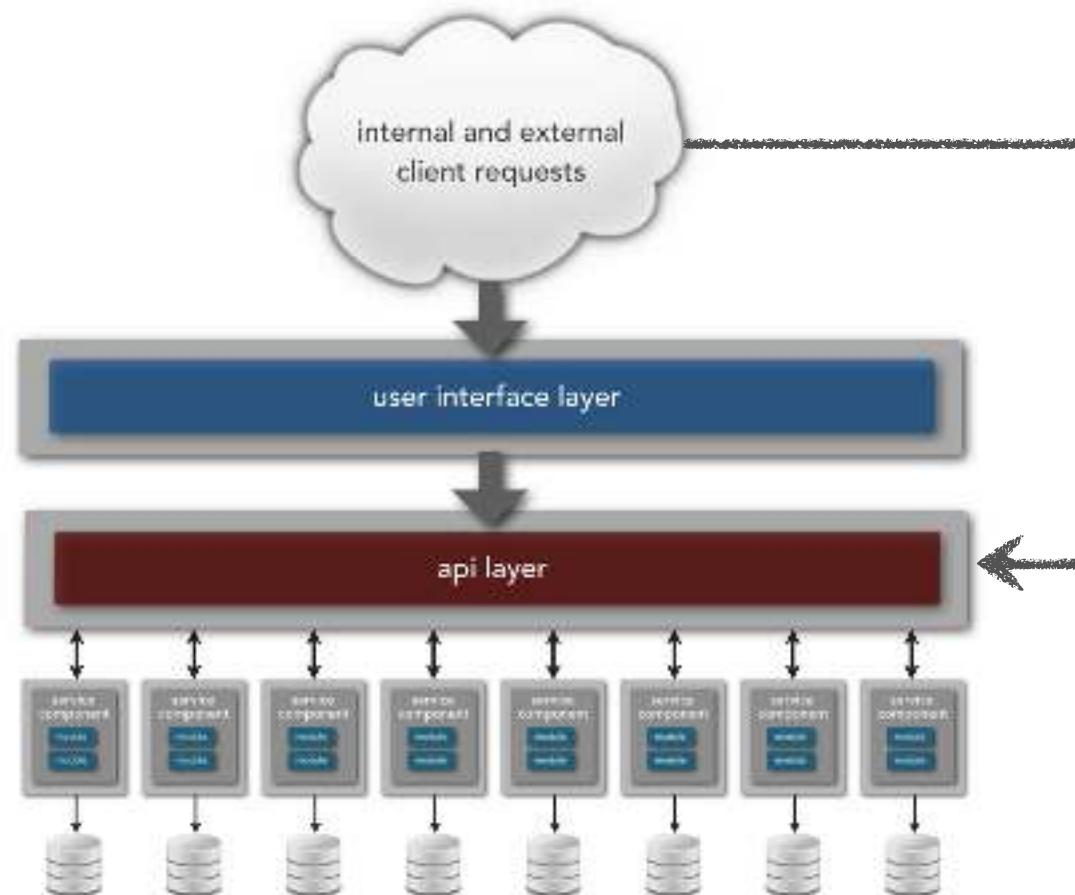
api layer



api layer

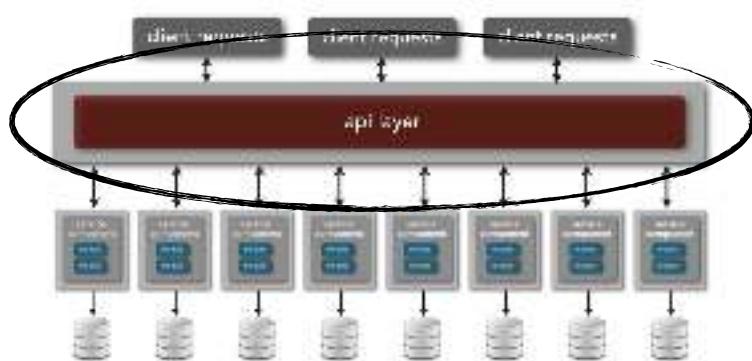


microservices architecture



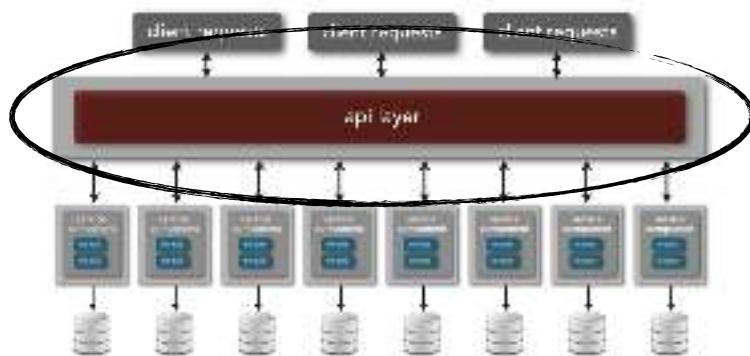
api layer

endpoint proxy



api layer

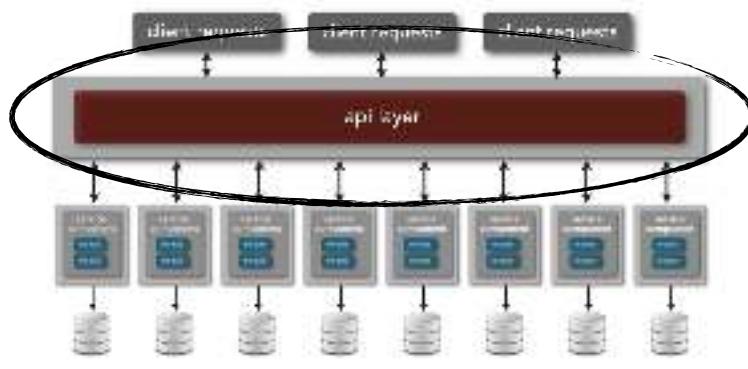
load balancer



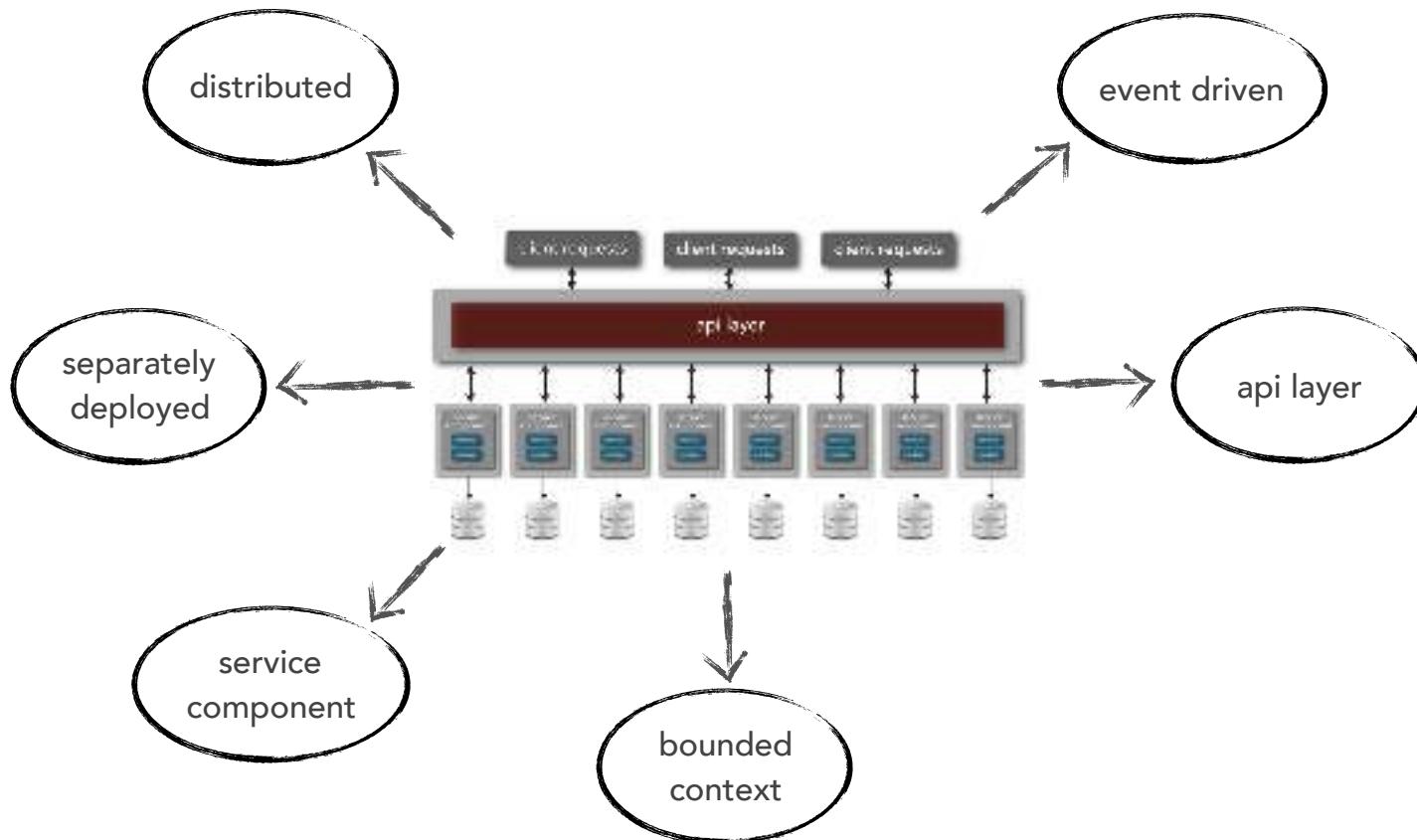
Citrix NetScaler

api layer

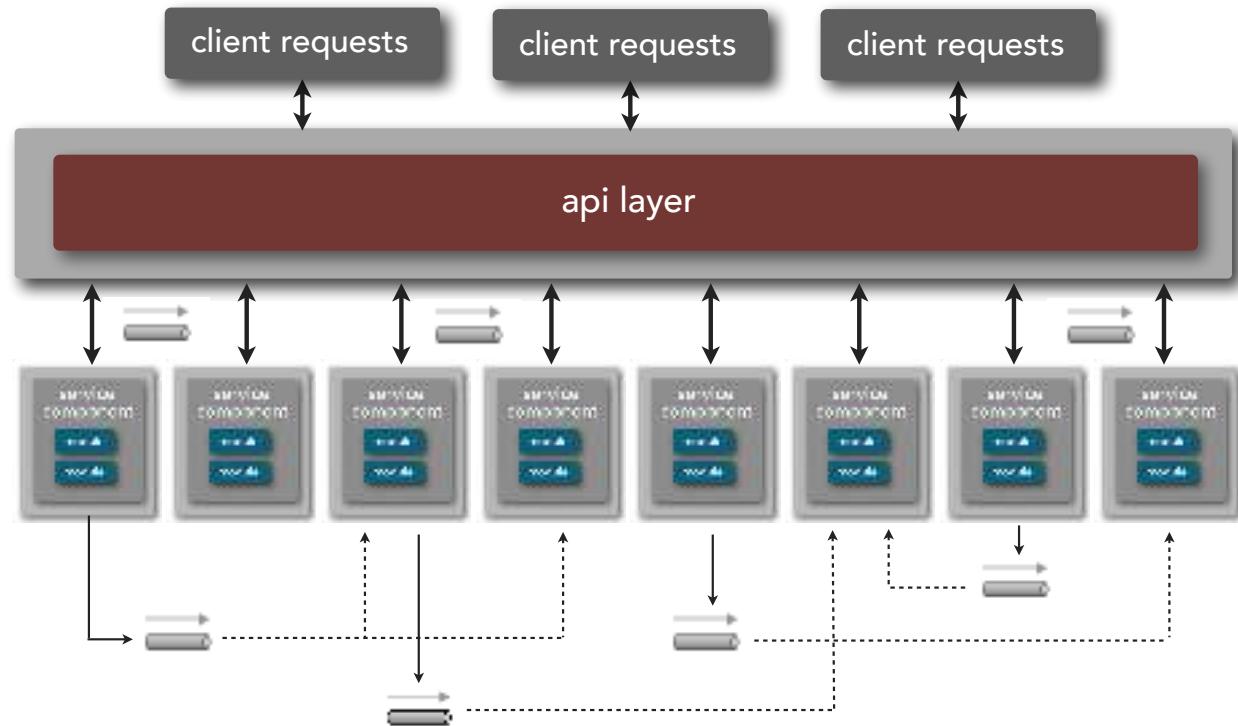
gateway (integration hub)



microservices architecture

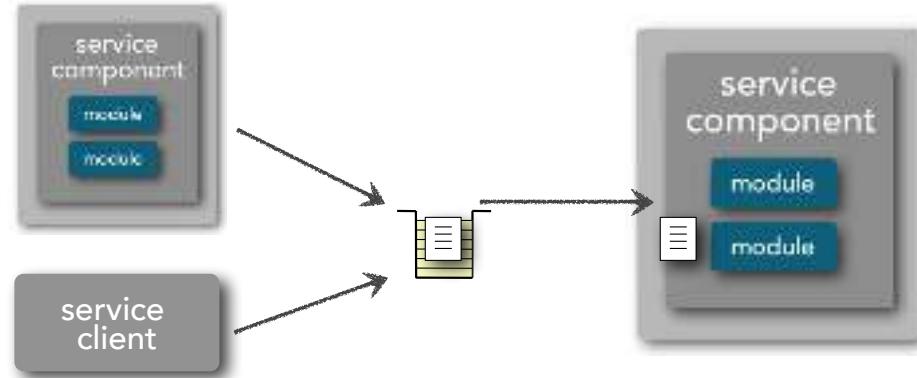


microservices architecture



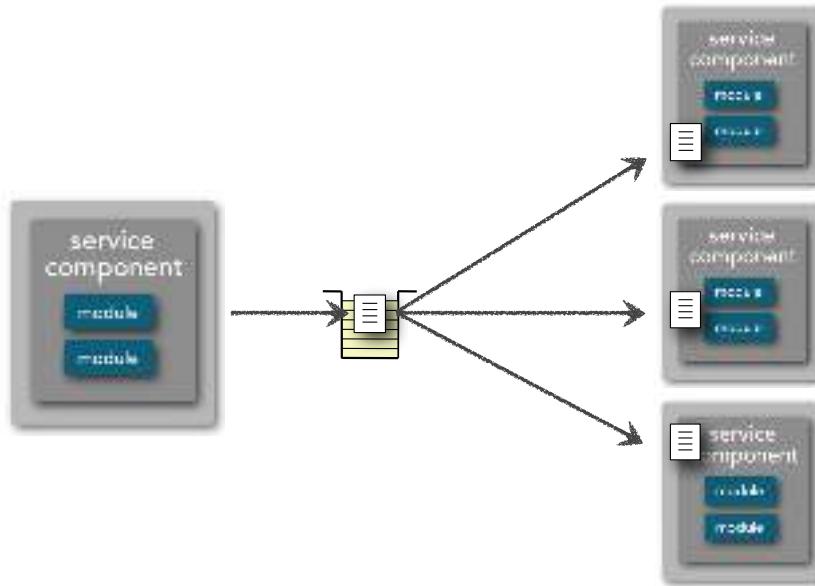
microservices architecture

asynchronous communications



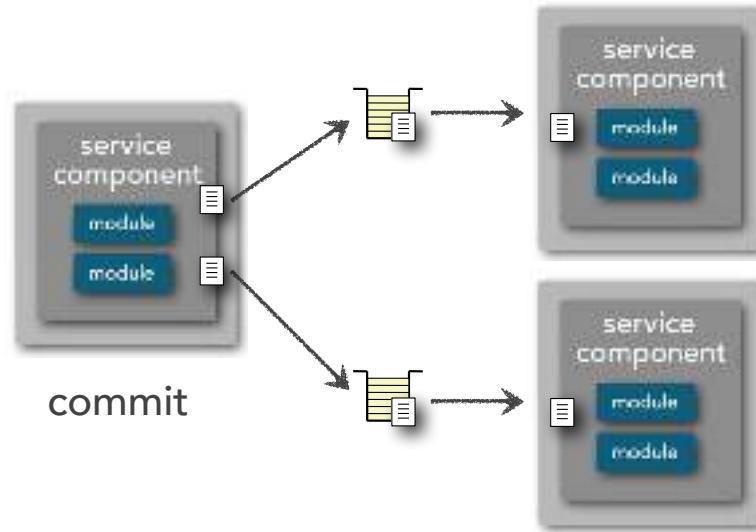
microservices architecture

broadcast capabilities



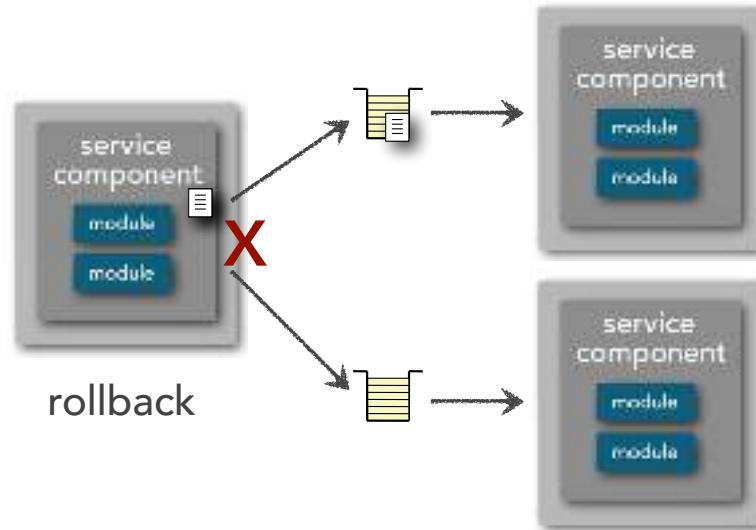
microservices architecture

transactional capabilities

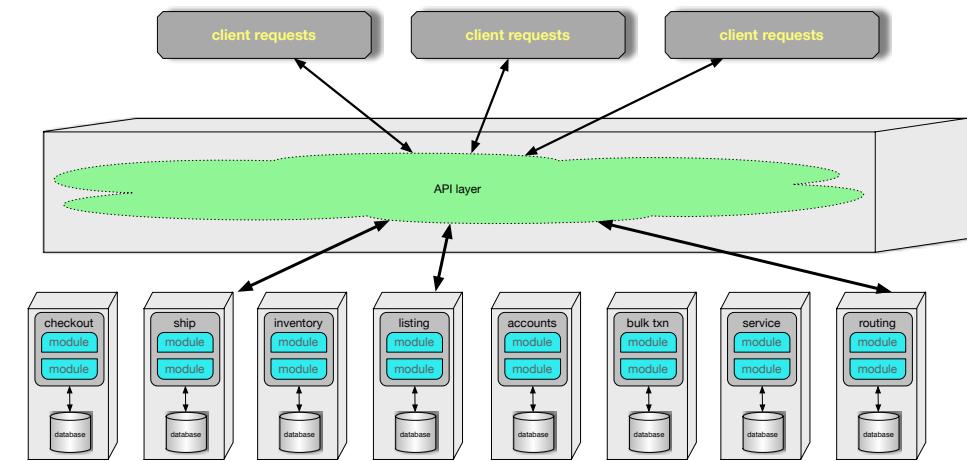


microservices architecture

transactional capabilities



Microservices

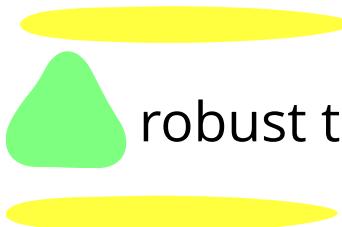
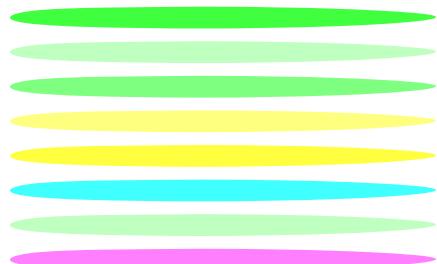


extremely fine grained



deployment pipeline considered standard

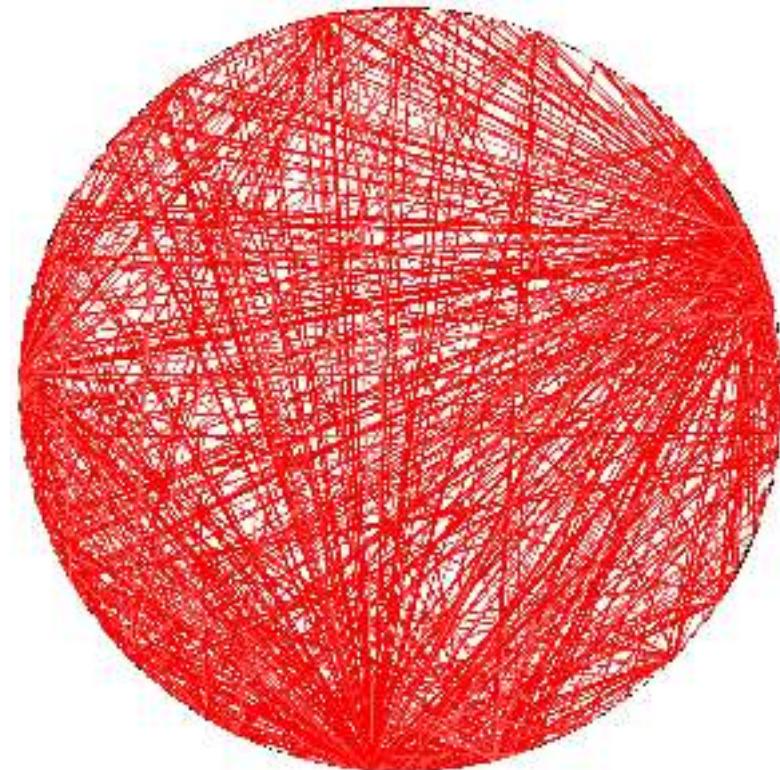
21st century DevOps practices



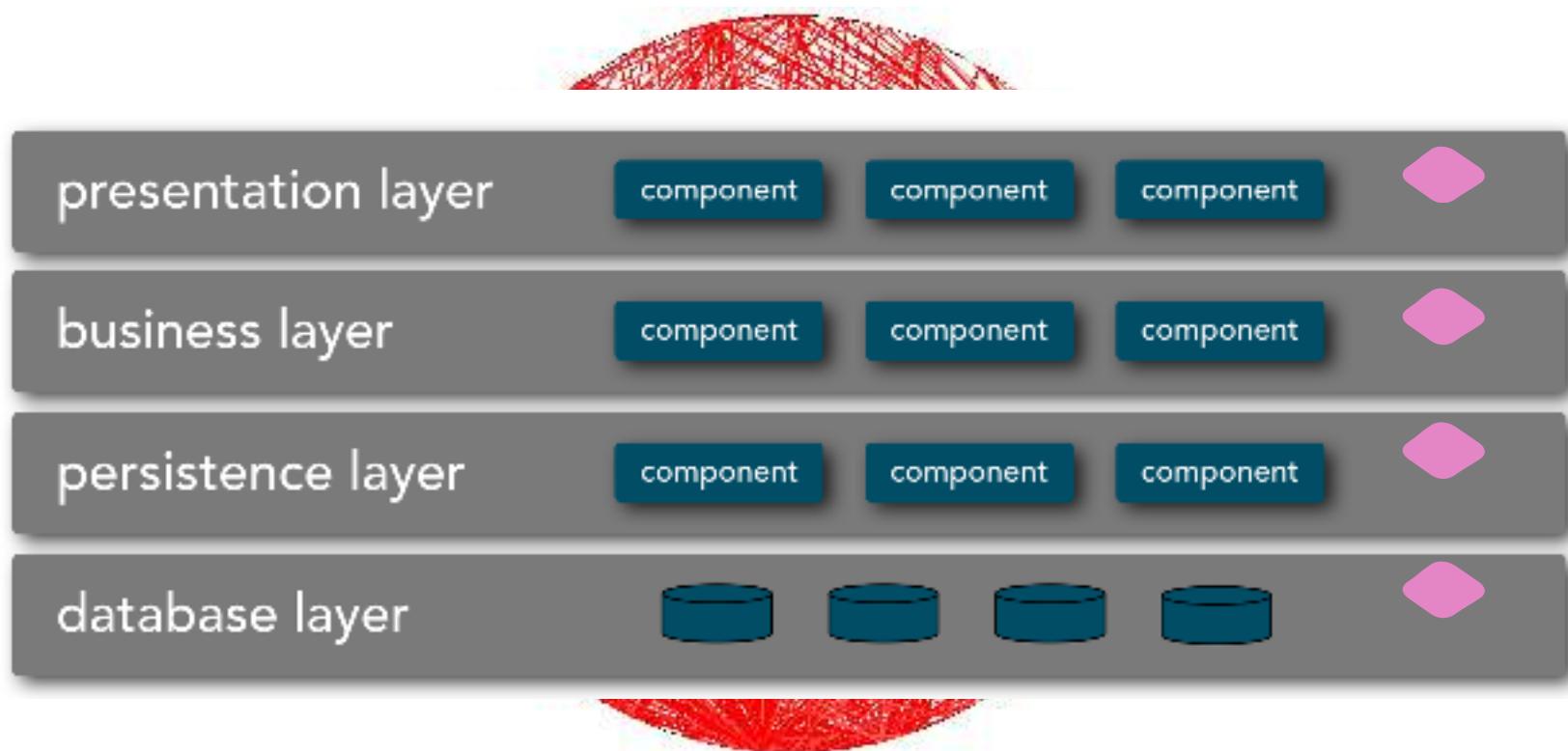
robust testing & fitness function culture

extremely decoupled fine-grained
quanta with well defined boundaries

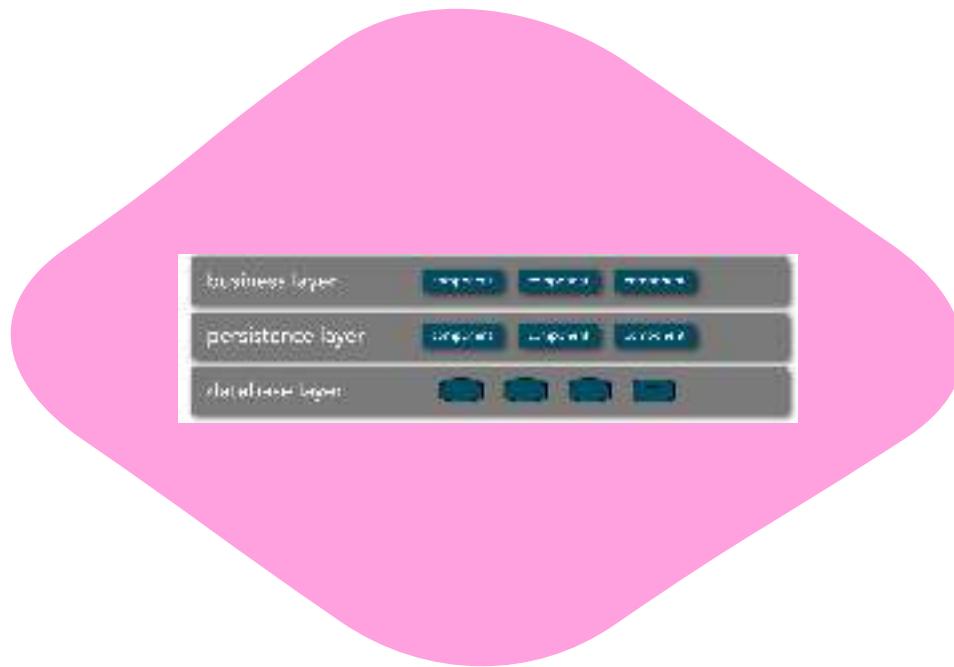
Shift to Domain-centric Architectures



Shift to Domain-centric Architectures

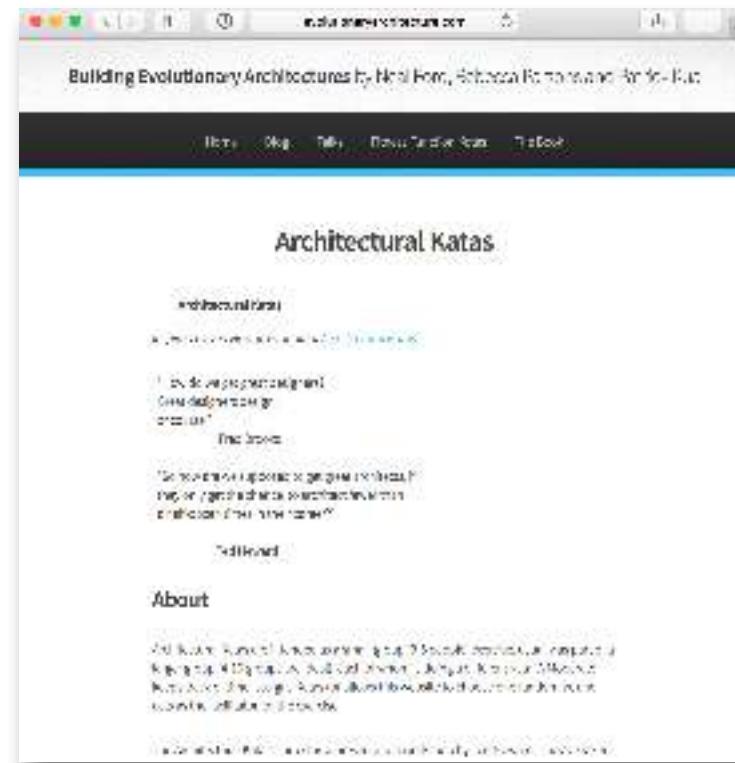


Shift to Domain-centric Architectures



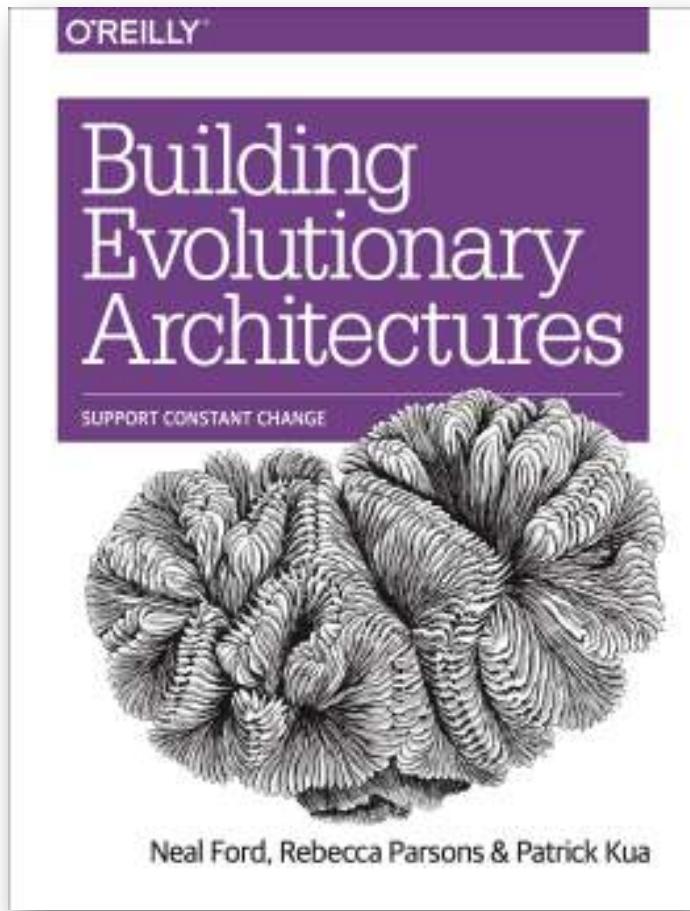
Architecture Katas

Round 2



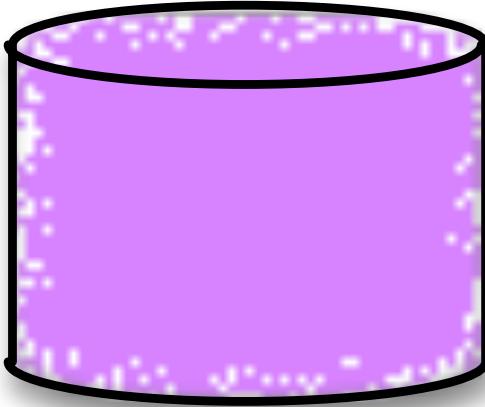
<http://evolutionaryarchitecture.com/ffkatas/>

Building Evolutionary Architectures



EVOLUTIONARY DATA





```
public class Test
{
    // Summary:
    // The main entry point for the application.
    // Command-line arguments are not supported.
    [STAThread]
    static void Main(string[] args)
    {
        // Logon
        EULoginManager licenseManager = new EULoginManager();
        // you can add the license here
        string networkLicense = "XXXXXXXX-XXXX-XXXX-XXXX-XXXX-XXXX";
        string password;
        password = "";
        licenseManager.Login(networkLicense, password);
        Console.WriteLine("Logged in.");
        var fullHome = @"C:\Documents and Settings\XXXXXX\Documents\My Pictures\Dark Castle 2003\St 17 Luigi's
EULogin\library\newfile.htm";
        EULogin library = new EULogin();
        // Set properties
        EULoginDay day = library.GetFirstByFullHome(fullHome);
        // Print properties to the console
        Console.WriteLine("Properties of {0}", fullHome);
        foreach (EUPProperty prop in day.Properties)
        {
            Console.WriteLine("Property: {0}, {1}, propValue: {2}");
        }
        // For the EUPProperty into a more friendly collection class,
        // it's a good idea for you to write a ToString() method than
        // would allow you to add and remove properties and sort them by
        // the EUPProperty type on the fly.
        AssemblyProperties = new AssemblyProperties();
        // Change the "CaptionDescription" property
        foreach (EUPProperty prop in AssemblyProperties)
        {
            if (prop.Name == "CaptionDescription")
            {
                prop.Value = "The logo congate to appear in a talk show. (Edited by Beyond TV Descriptivity)";
            }
        }
        // Create a new EUPProperty with the edited property
        EULoginDay newDay = new EULoginDay();
        newDay.Properties = (EUPPropertyList)AssemblyProperties;
        // This method will edit the recording
        library.EditRecord(newDay, null);
        // Print properties to the console and verify the changes
        Console.WriteLine("Edited properties of {0}", fullHome);
        foreach (EUPProperty prop in day.Properties)
        {
            Console.WriteLine("Property: {0}, {1}, propValue: {2}");
        }
        // From now you can use the edited fullHome to read from
        Console.WriteLine("Please say log to index...");
        library.ReadIndex();
        return;
    }
}
```

Data & code are both abstractions based on the real world.

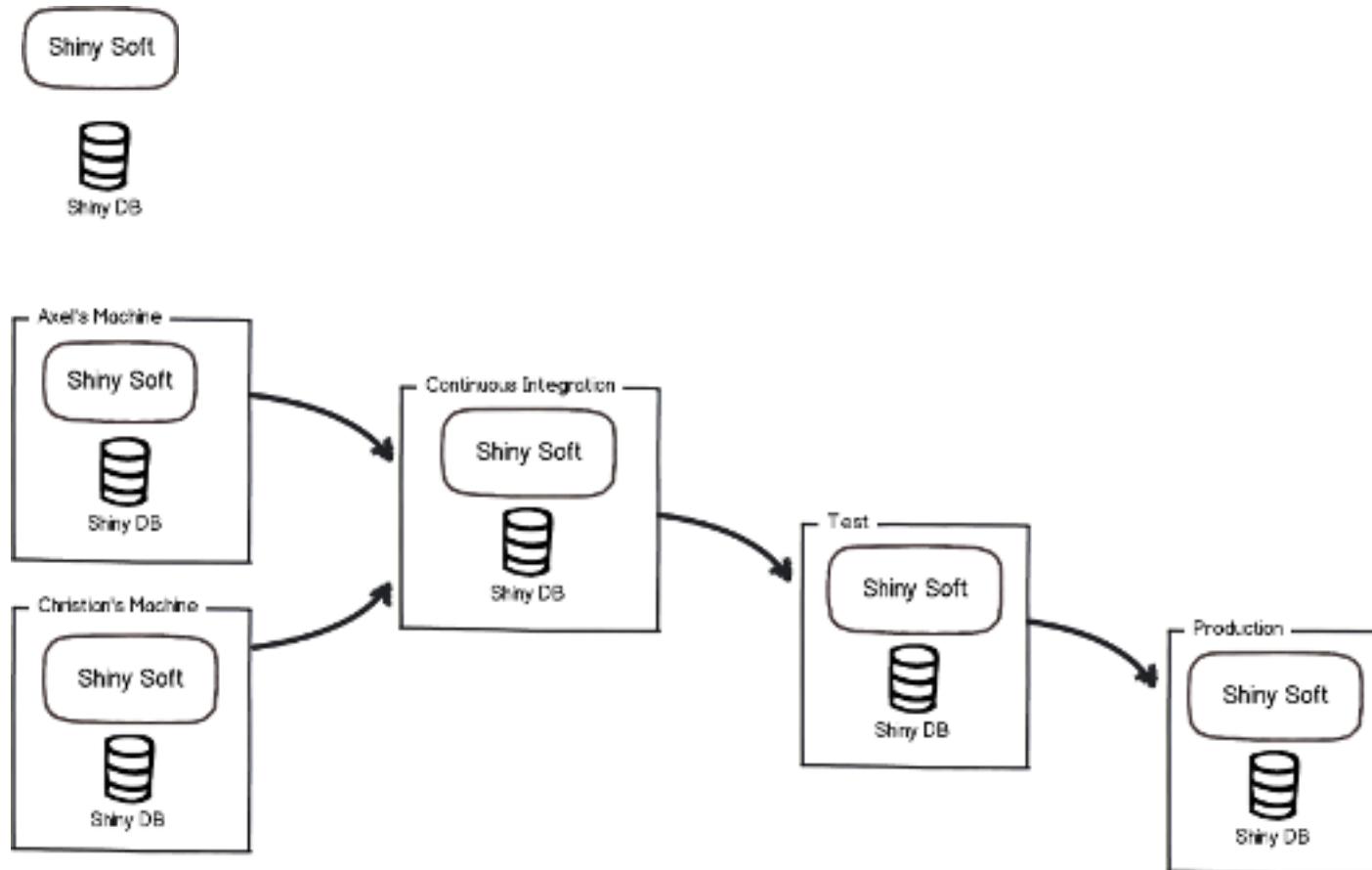


Data & code are symbiotic.

Evolving Schemas (like code)

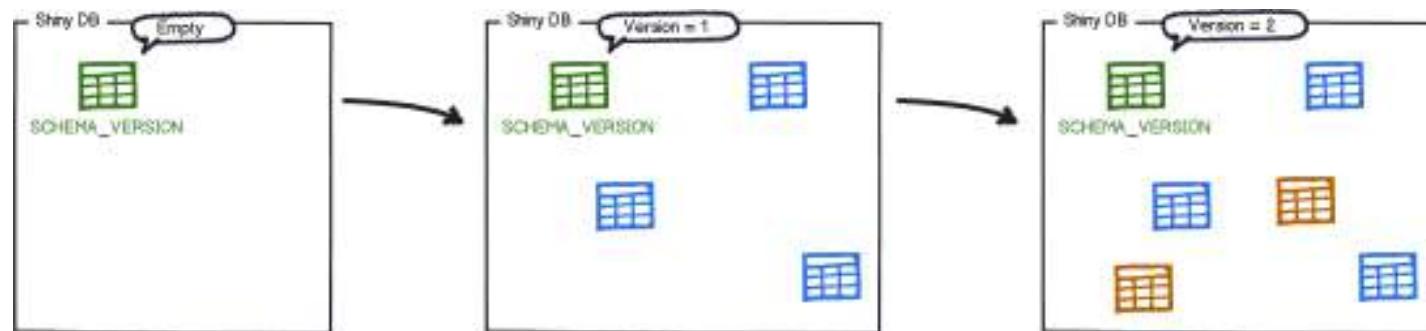
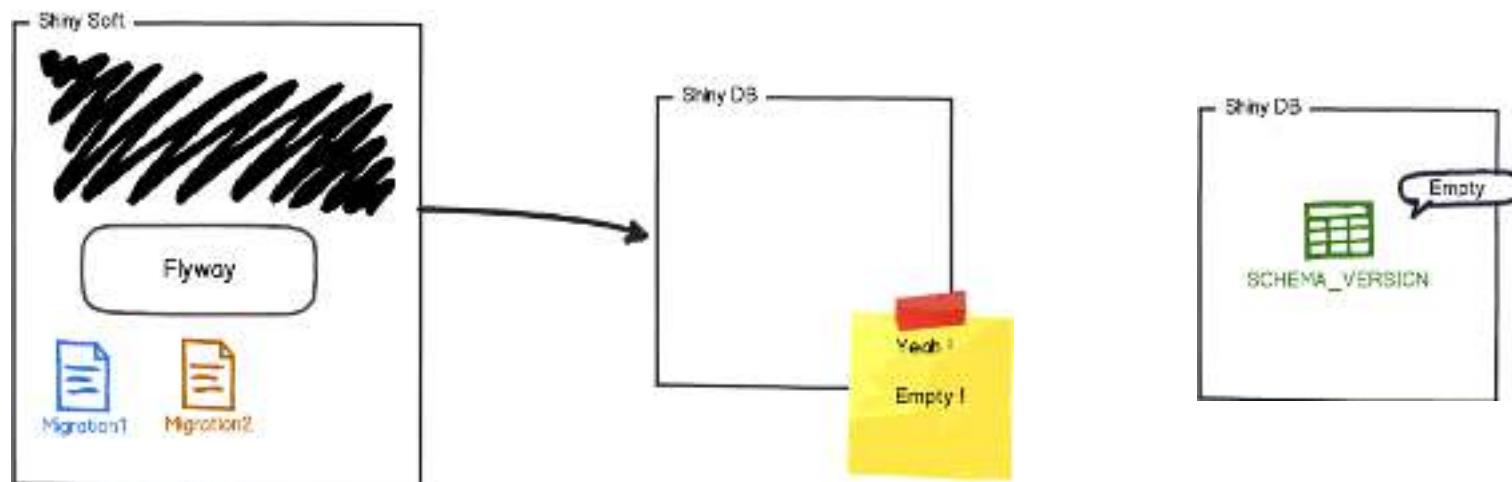
- Tested
- Versioned
- Incremental

DbDeploy Pattern

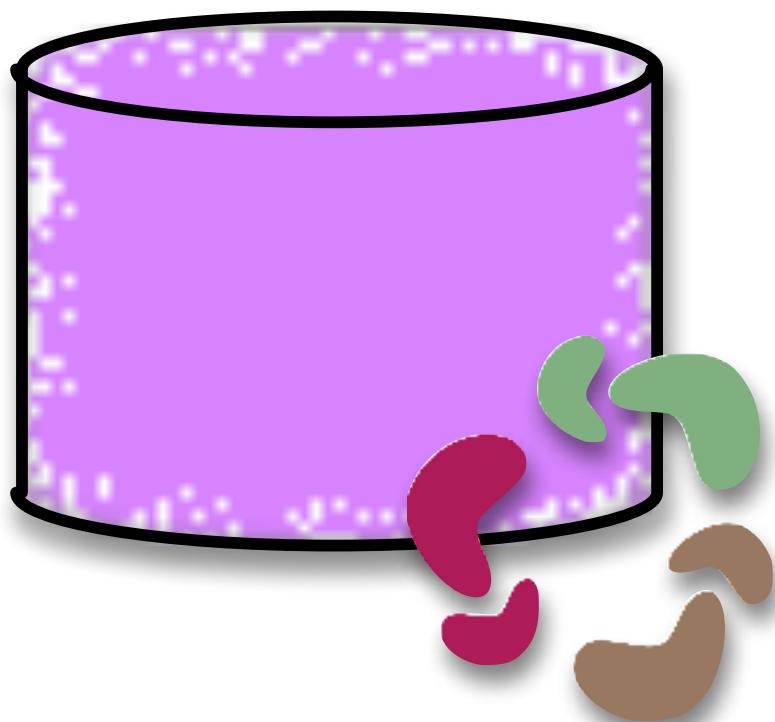


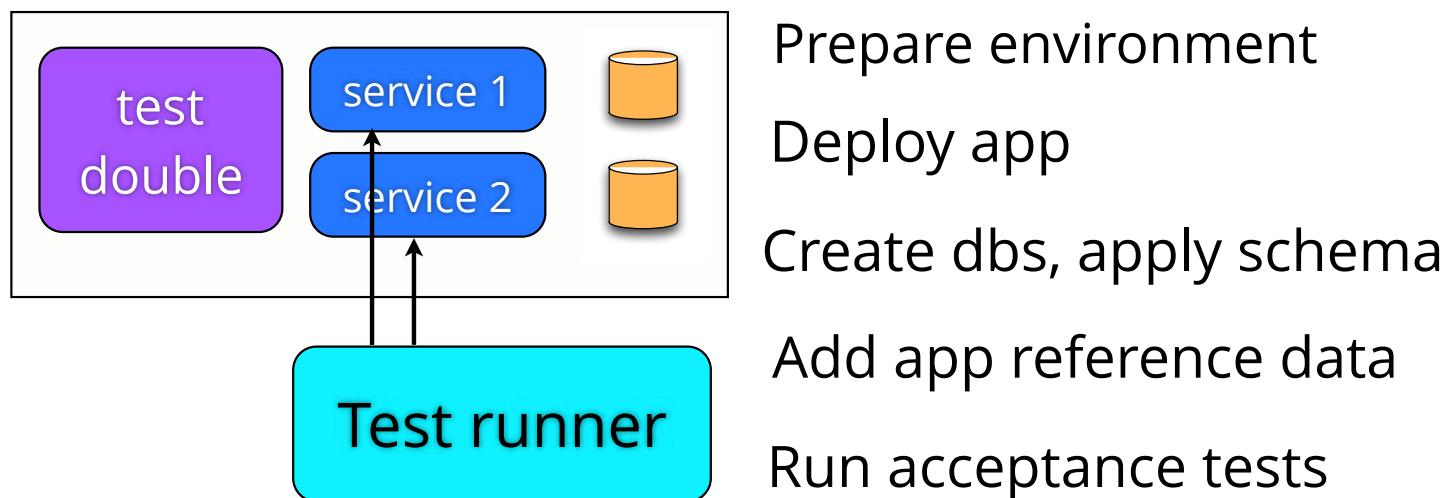
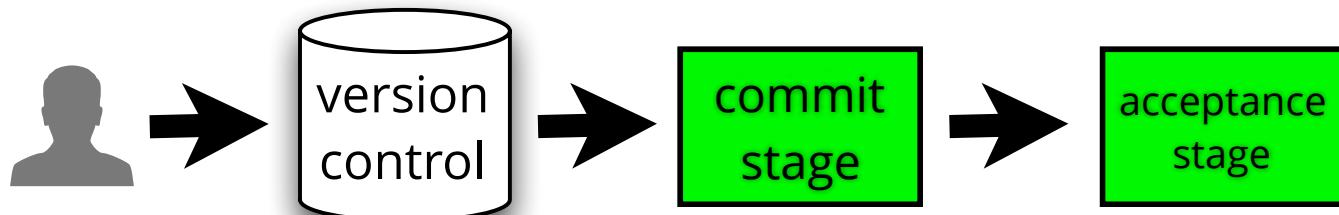


<http://flywaydb.org/>

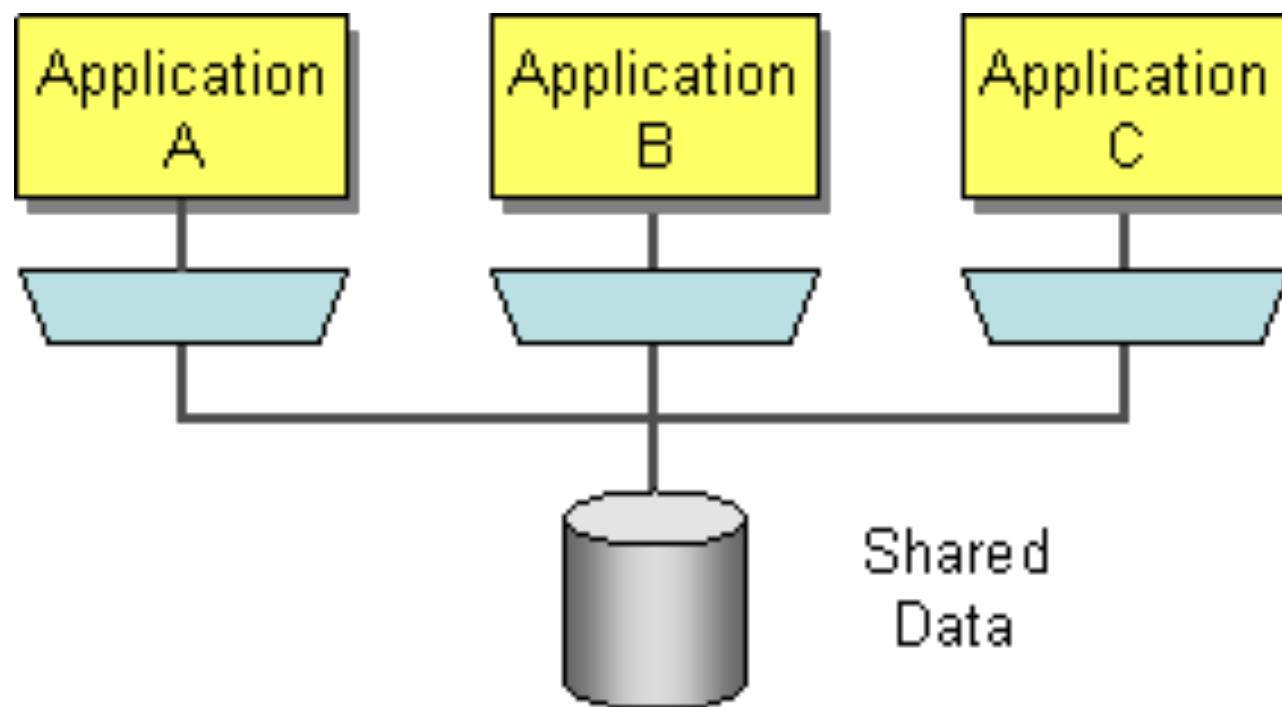


Continuous Integration for Databases

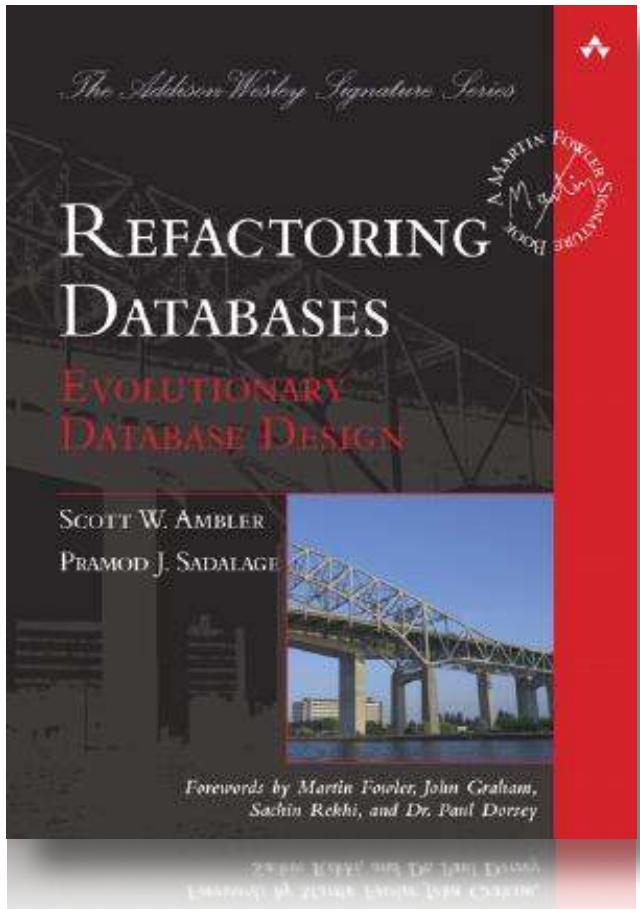




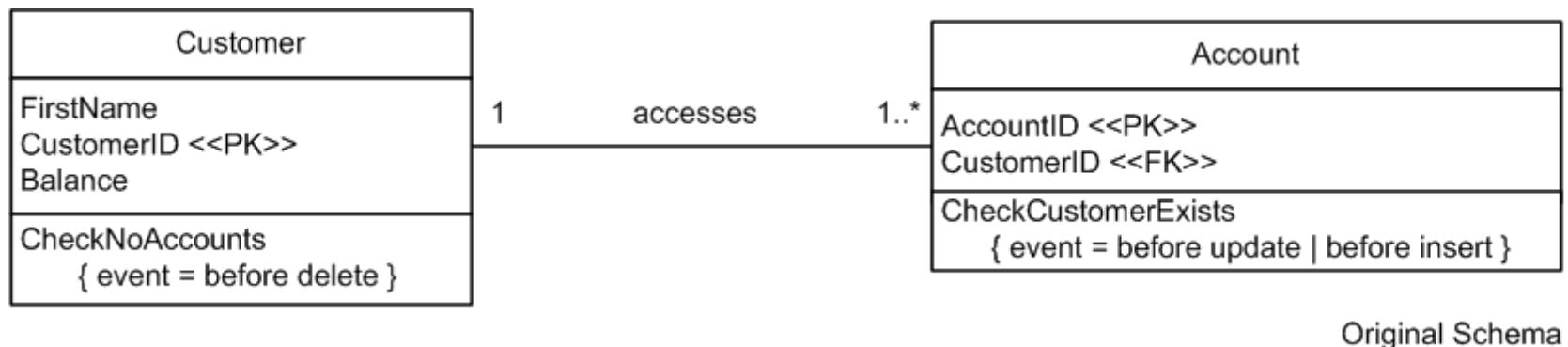
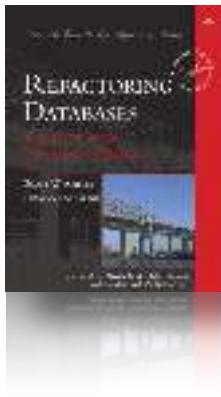
Shared-database Integration



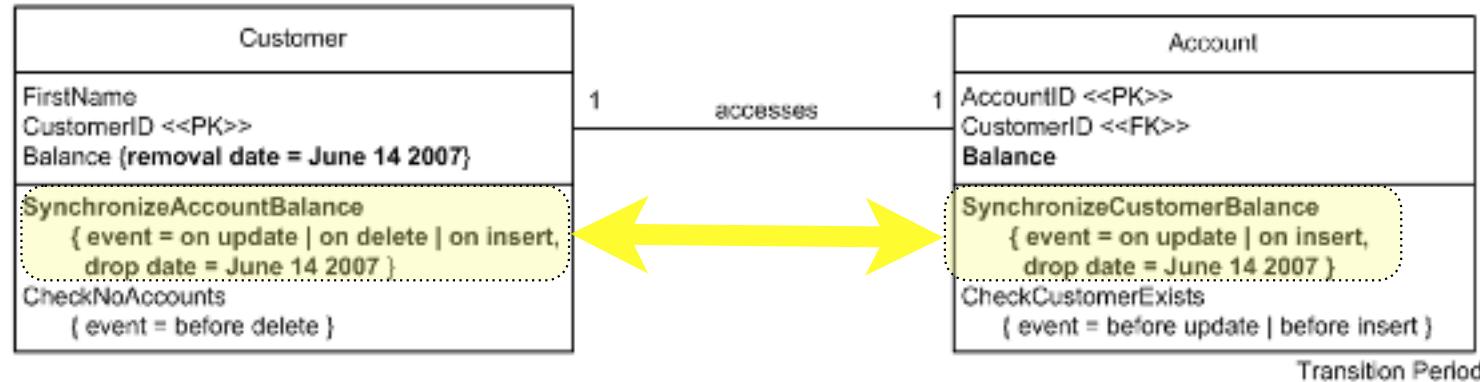
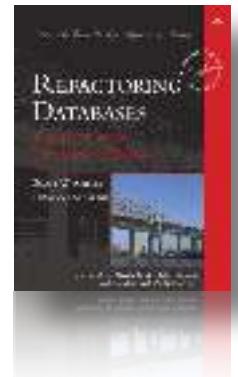
Refactoring Databases



Move Column Refactoring

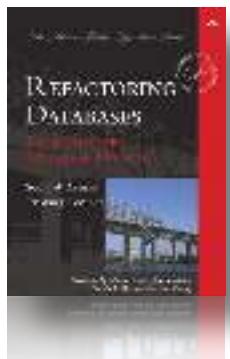
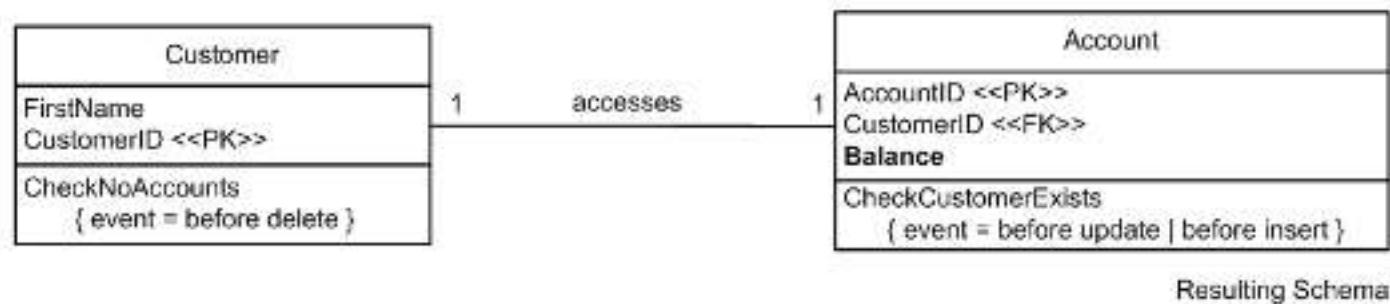


Transition Period

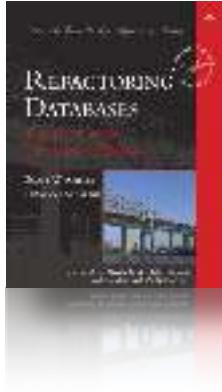


Move Column Refactoring

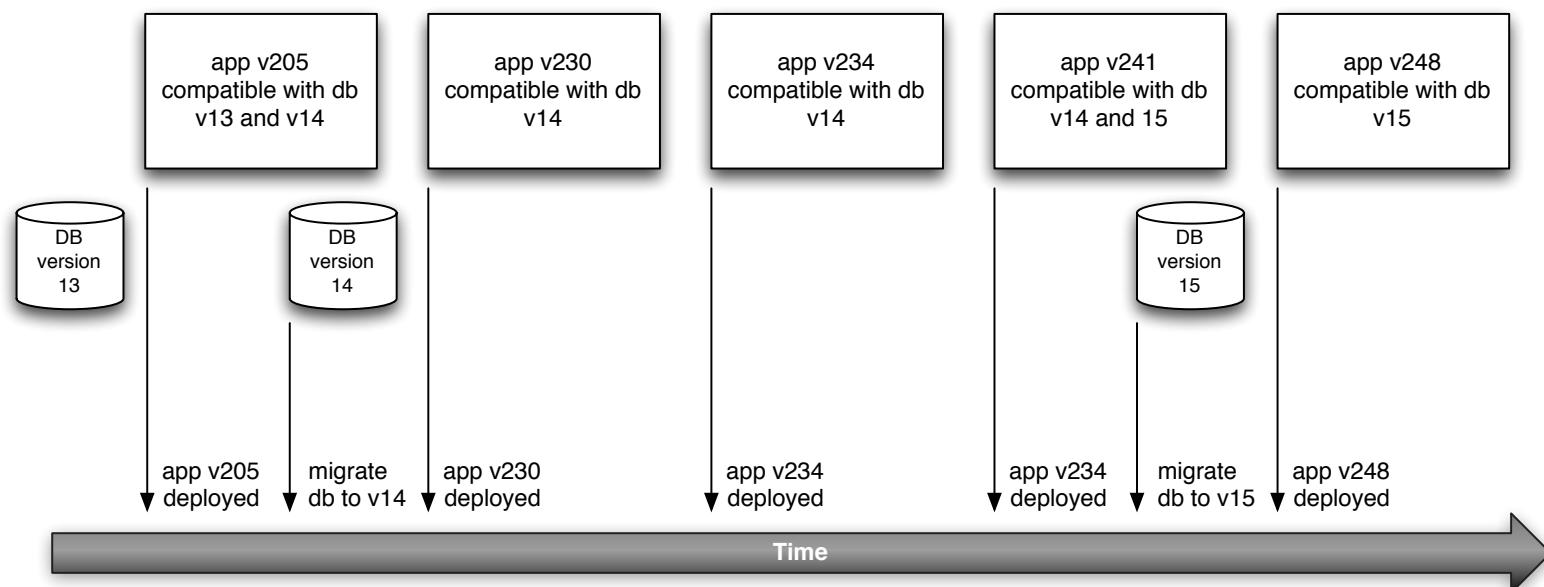
Ending Schema



Move Column Refactoring

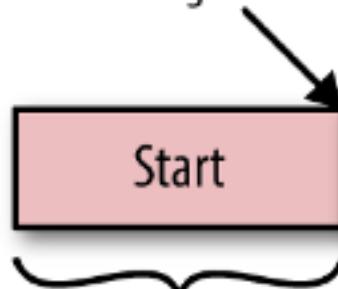


Decouple DB Updates: the Expand/contract Pattern

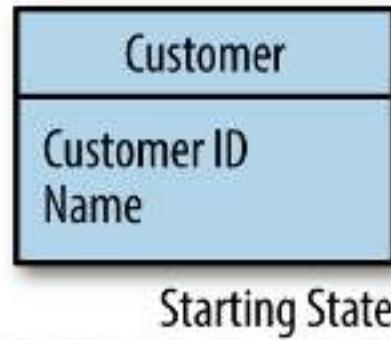


Expand/Contract Pattern

Deploy new changes,
migrate data, put in
scaffolding code



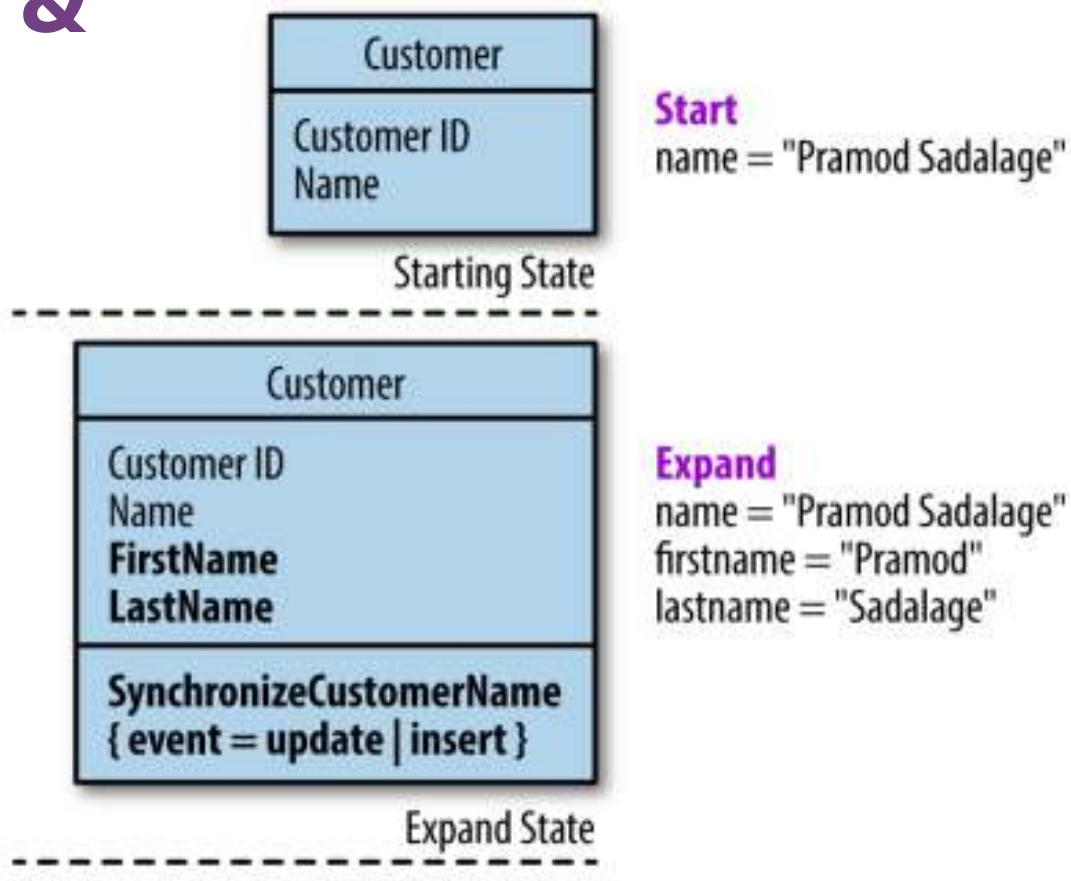
Change 'name' to 'firstname' & 'lastname'



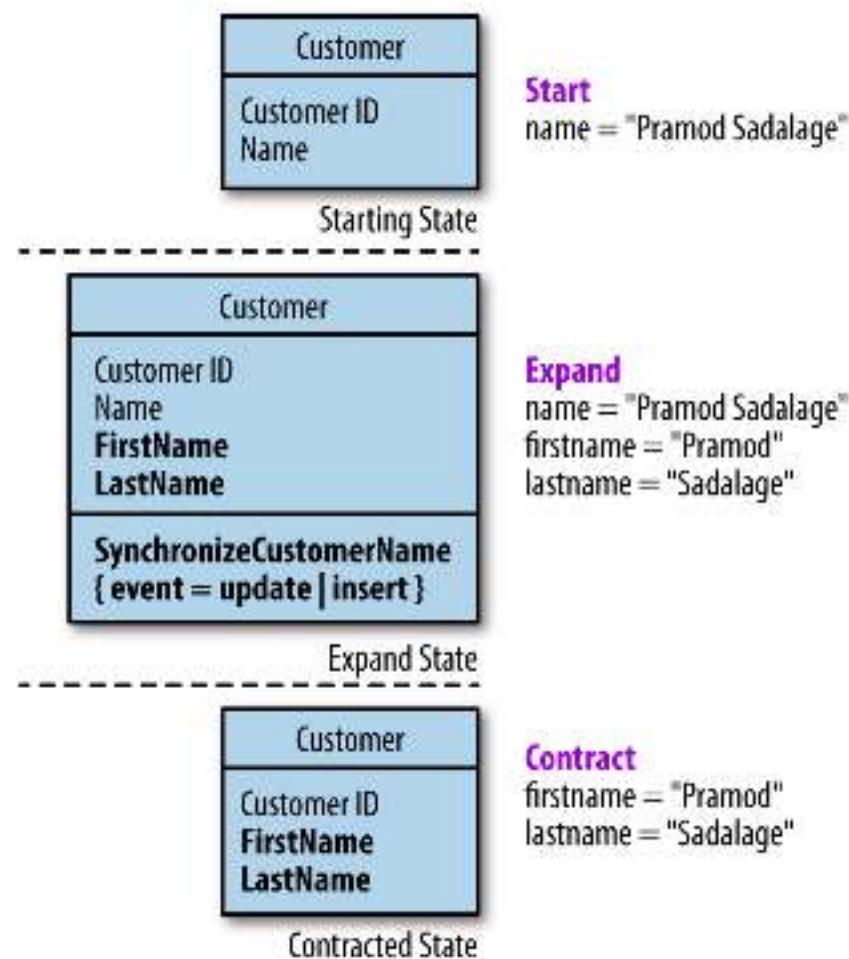
Start

name = "Pramod Sadalage"

Change 'name' to 'firstname' & 'lastname'



Change 'name' to 'firstname' & 'lastname'



#1: integration points, legacy data

```
ALTER TABLE customer ADD firstname VARCHAR2(60);
ALTER TABLE customer ADD lastname VARCHAR2(60);
ALTER TABLE customer DROP COLUMN name;
```

#2: integration points, legacy data

```
ALTER TABLE Customer ADD firstname VARCHAR2(60);
ALTER TABLE Customer ADD lastname VARCHAR2(60);
UPDATE Customer set firstname = extractfirstname (name);
UPDATE Customer set lastname = extractlastname (name);
ALTER TABLE customer DROP COLUMN name;
```

#3: integration points, legacy data

```
ALTER TABLE Customer ADD firstname VARCHAR2(60);
ALTER TABLE Customer ADD lastname VARCHAR2(60);

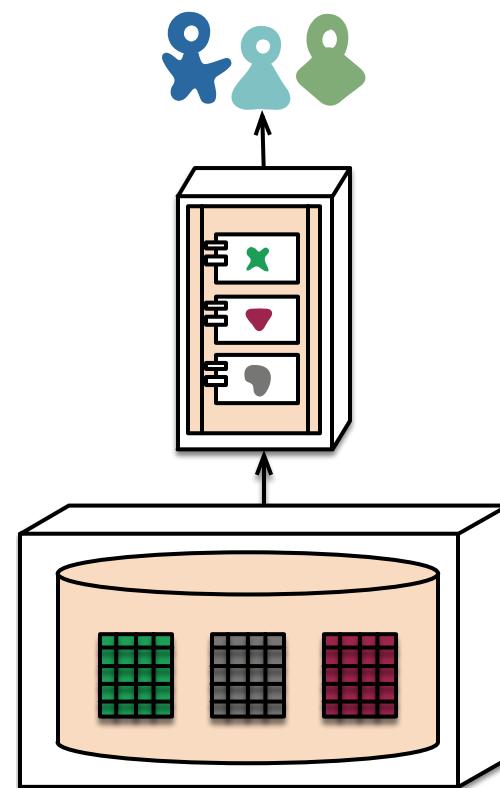
UPDATE Customer set firstname = extractfirstname (name);
UPDATE Customer set lastname = extractlastname (name);

CREATE OR REPLACE TRIGGER SynchronizeName
BEFORE INSERT OR UPDATE
ON Customer
REFERENCING OLD AS OLD NEW AS NEW
FOR EACH ROW
BEGIN

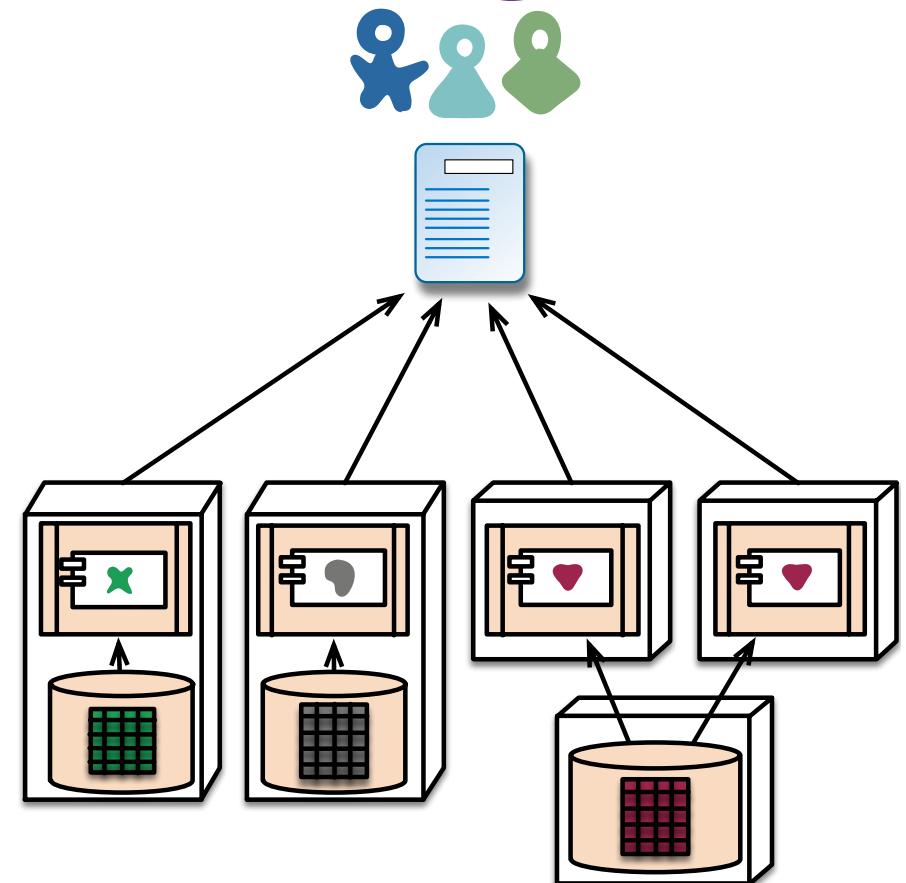
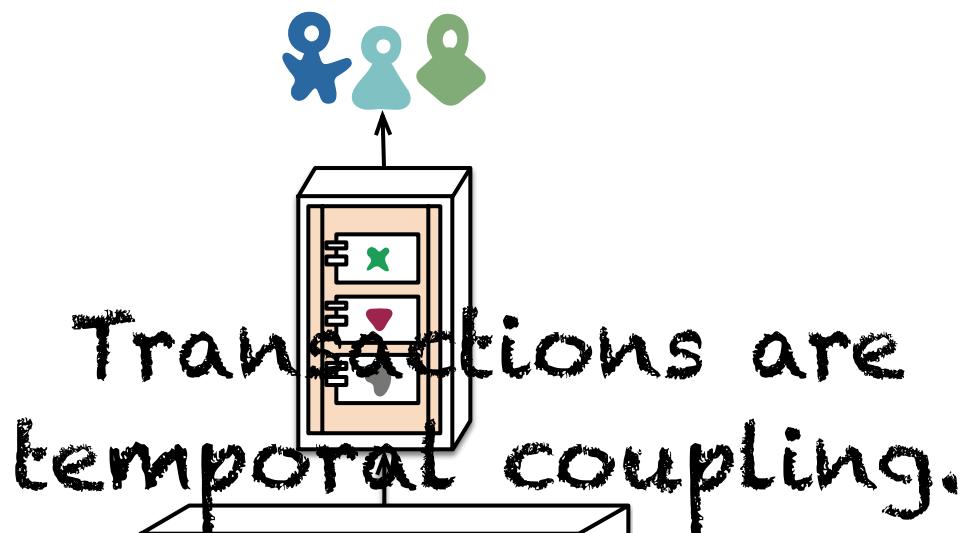
IF :NEW.Name IS NULL THEN
    :NEW.Name := :NEW.firstname||' '||:NEW.lastname;
END IF;
IF :NEW.name IS NOT NULL THEN
    :NEW.firstname := extractfirstname(:NEW.name);
    :NEW.lastname := extractlastname(:NEW.name);
END IF;
END;
```



Decentralized Data Management

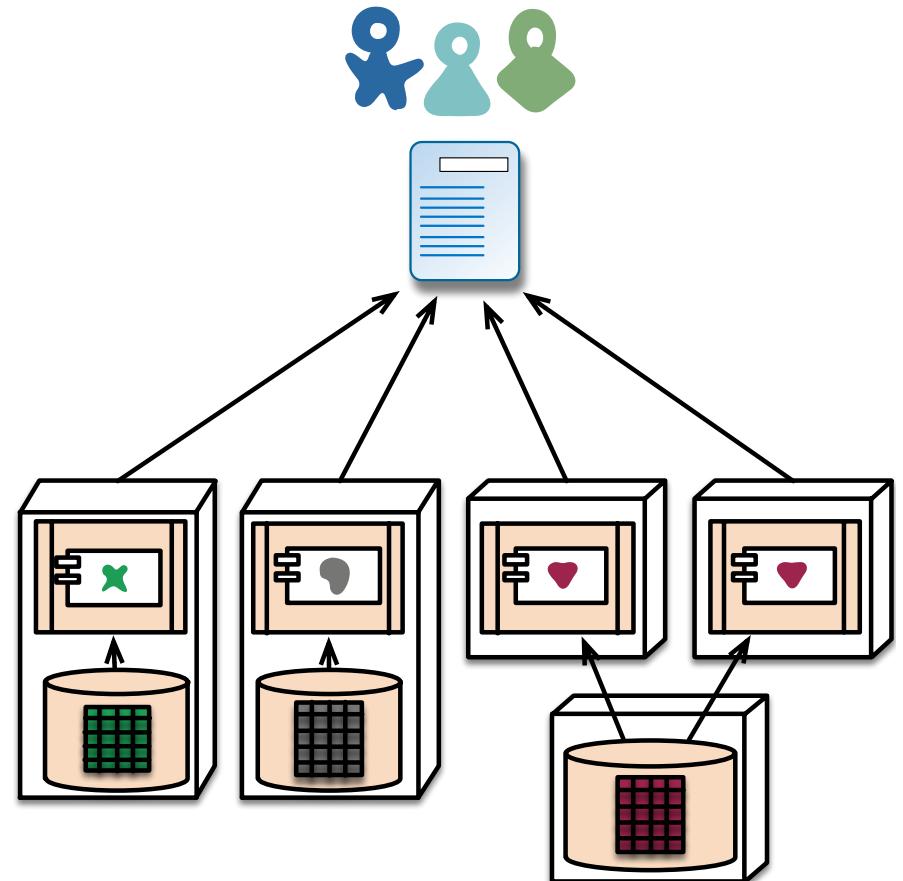


Decentralized Data Management



Decentralized Data Management

Limit
transactional
contexts.





**Database transactions
act as a strong nuclear
force, binding quanta
together.**

Age & Quality of Data





**Refusing to refactor schemas or
eliminate old data couples your
architecture to the past, which
is difficult to refactor.**

Penultima ↑ e



Evolving Routing

< >

<home> <catalog><item>

Penultima ↑ e



Evolving Routing

Routes



Penultima ↑ e

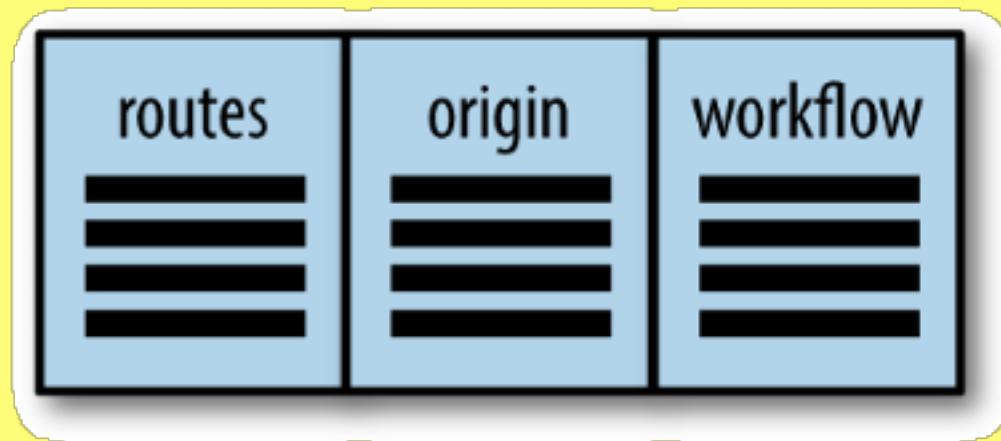


Evolving Routing

Routes



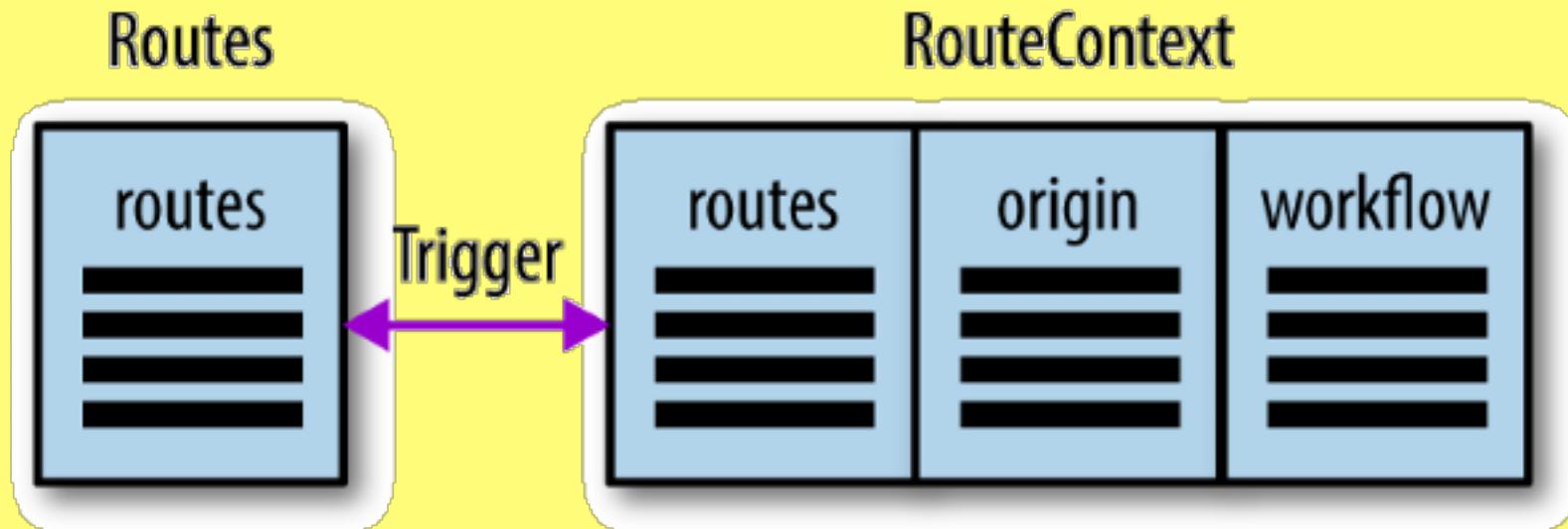
RouteContext



Penultima ↑ e



Evolving Routing



Penultima ↑ e

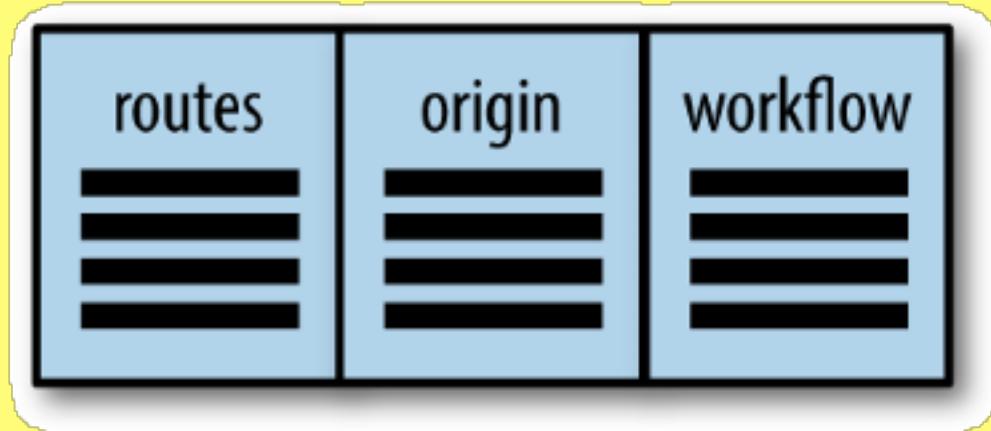


Evolving Routing

Routes

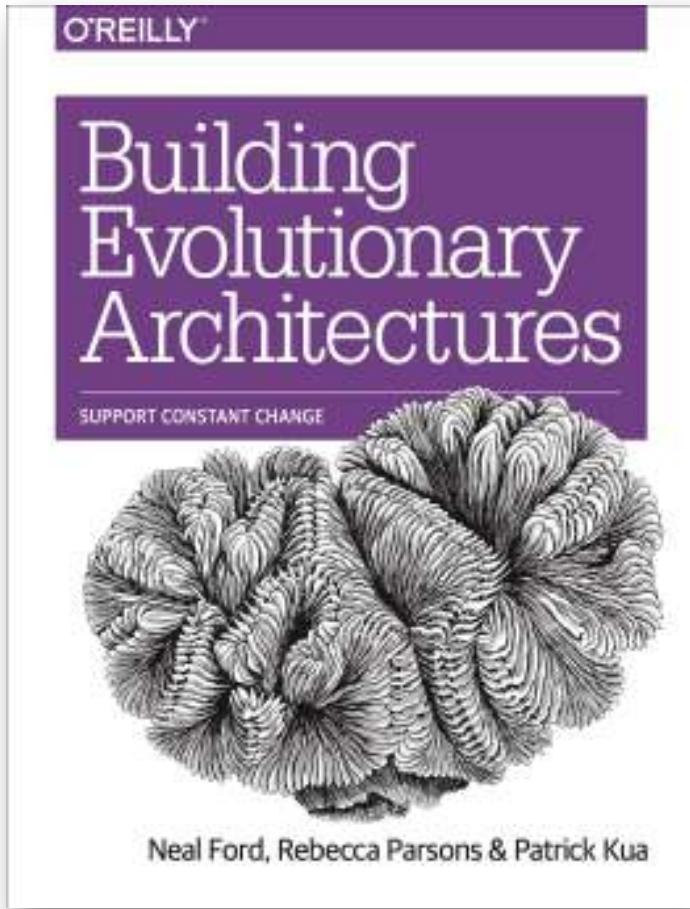


RouteContext



Building Evolutionary Architectures

BUILDING EVOLVABLE
ARCHITECTURES

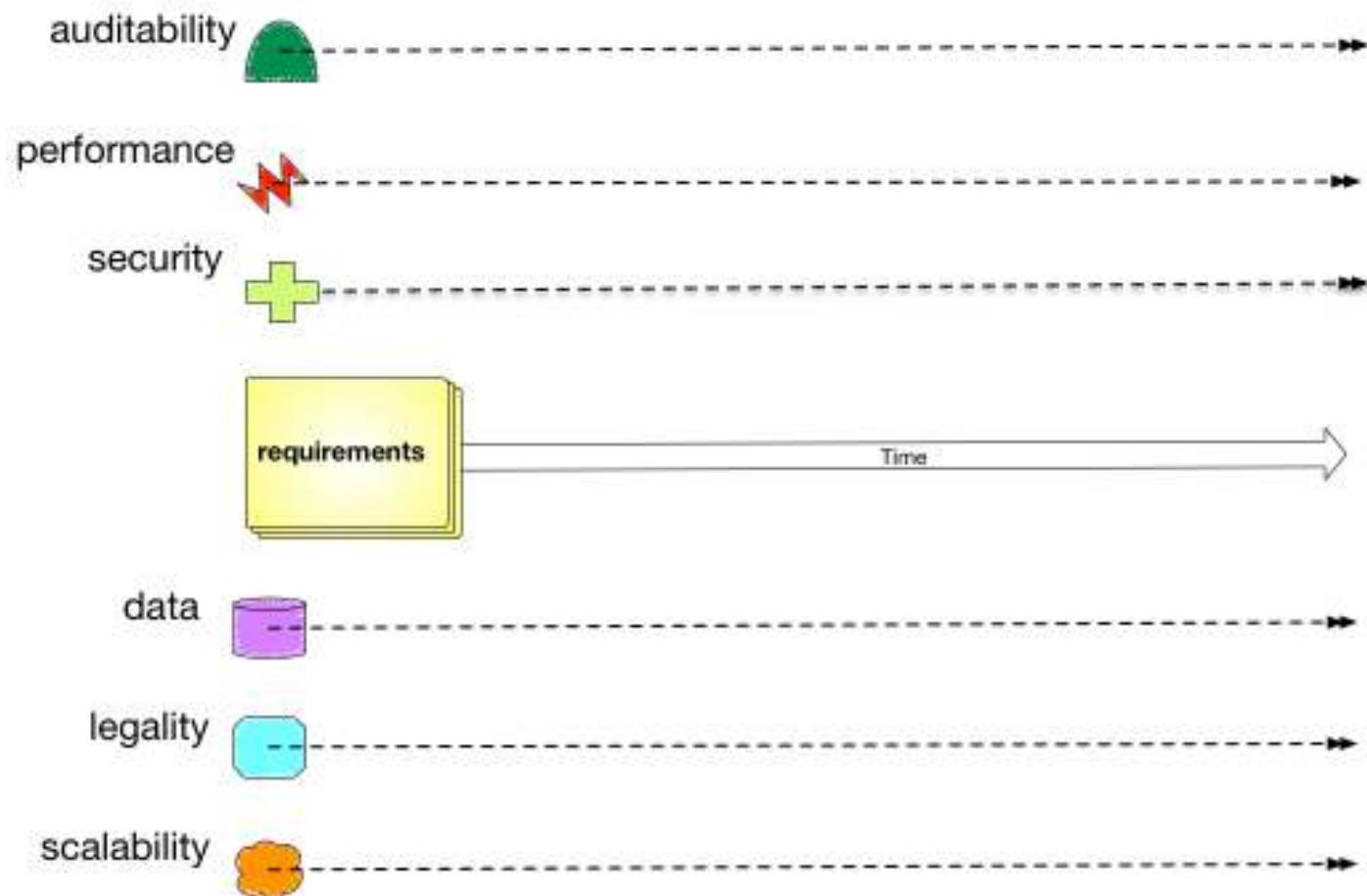




Mechanics

1. Identify dimensions affect by evolution

1. Identify dimensions affect by evolution



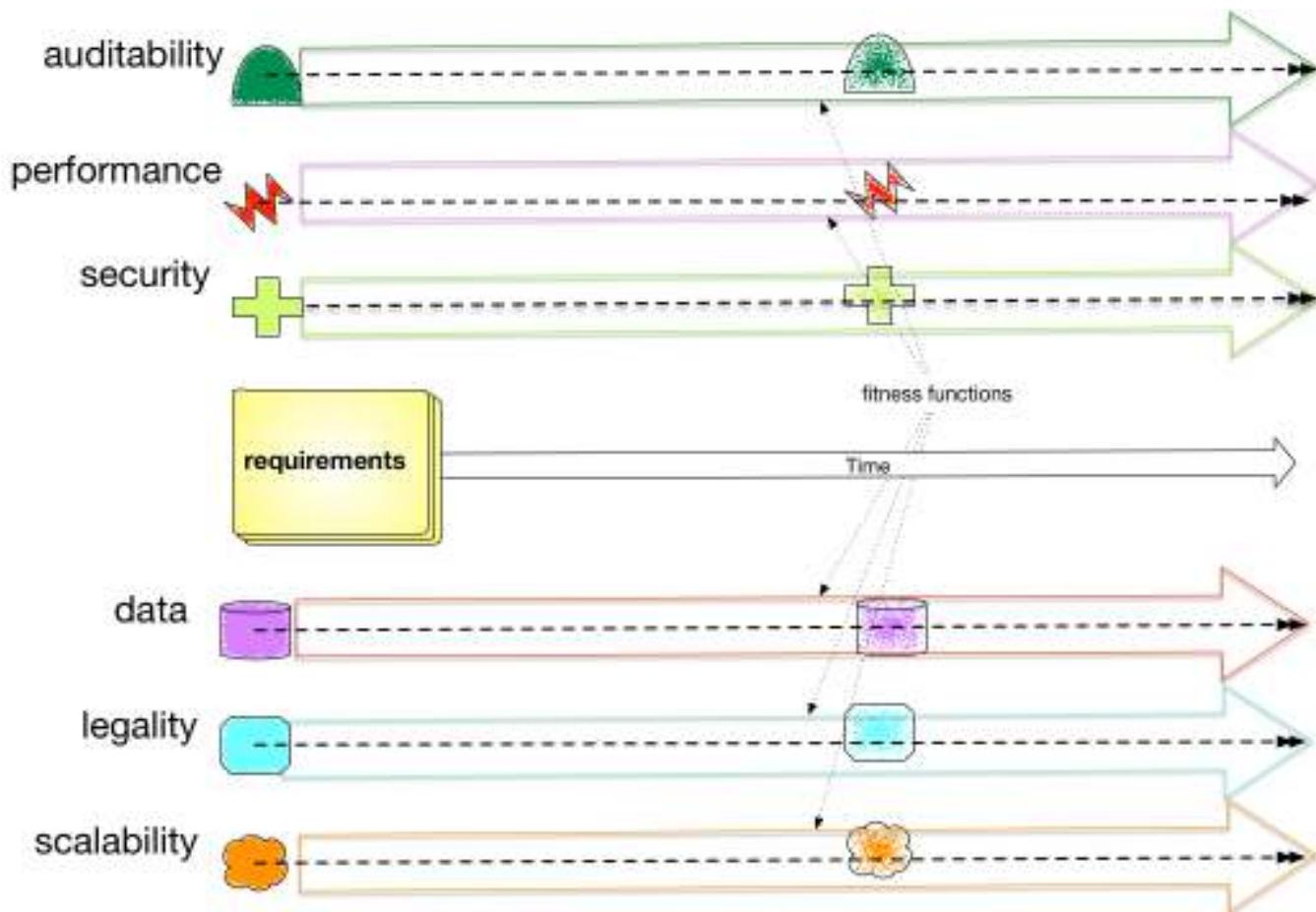


Mechanics

1. Identify dimensions affect by evolution
2. Define Fitness Function(s) for Each Dimension



2. Define Fitness Function(s) for Each Dimension



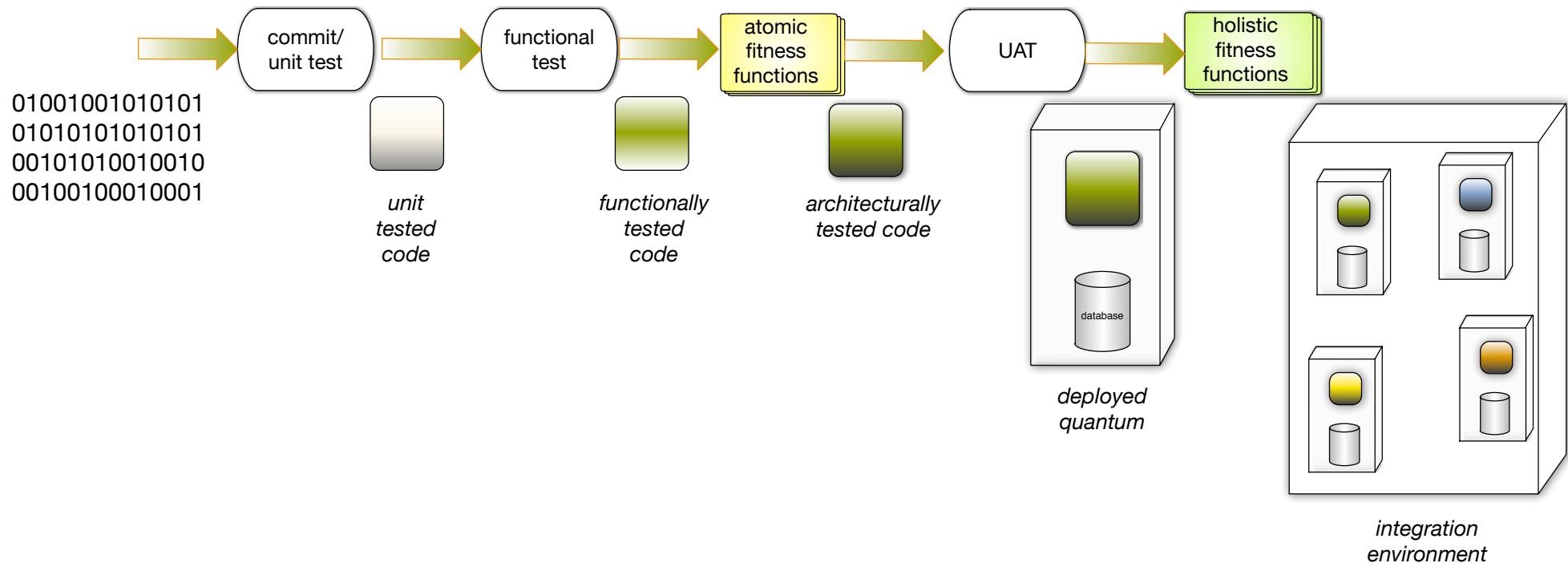


Mechanics

1. Identify dimensions affected by evolution
2. Define Fitness Function(s) for Each Dimension
3. Use Deployment Pipelines to Automate Fitness Functions



3. Use Deployment Pipelines to Automate Fitness Functions





Mechanics

1. Identify dimensions affected by evolution
2. Define Fitness Function(s) for Each Dimension
3. Use Deployment Pipelines to Automate Fitness Functions



Mechanics

1. Identify dimensions affected by evolution
2. Define Fitness Function(s) for Each Dimension
3. Use Deployment Pipelines to Automate Fitness Functions

Greenfield Projects



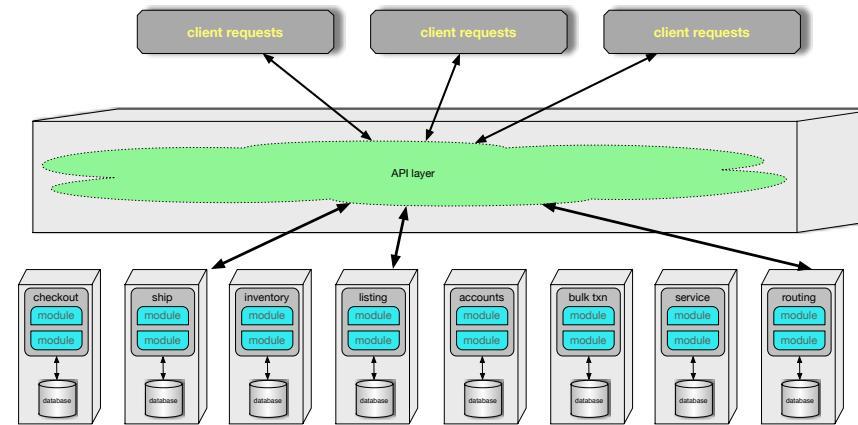
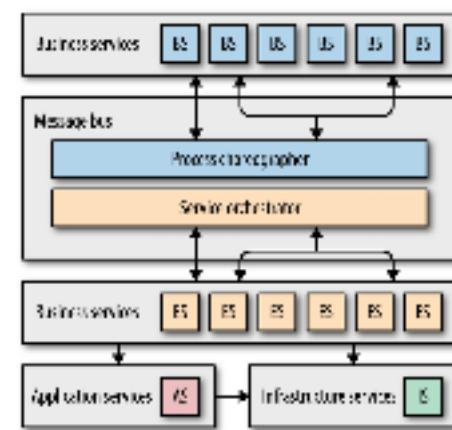
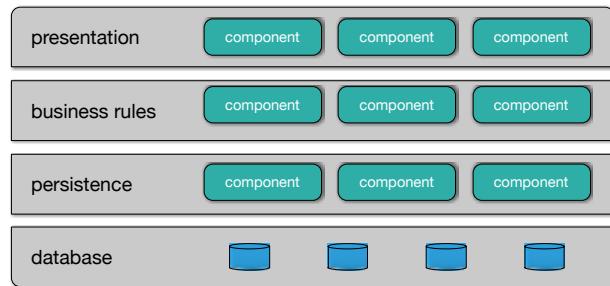
Greenfield Projects

- implement incremental change at project inception
- fitness functions definition easier before implementation
- architects don't have to untangle legacy coupling points
- protective fitness functions from project outset
- choose an architecture pattern that support evolution

Brownfield Projects



Appropriate Coupling & Cohesion

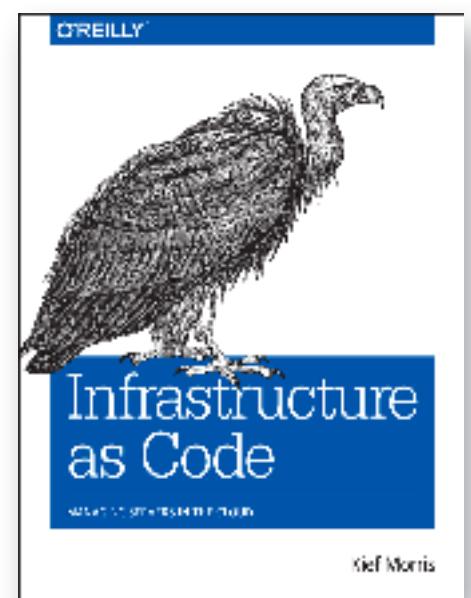
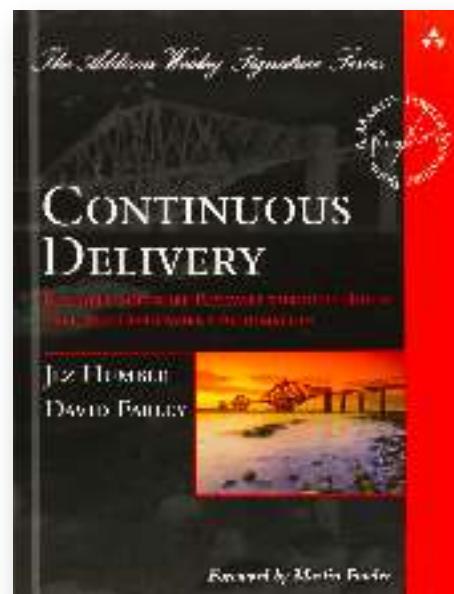




**Understand the business
problem before choosing
an architecture.**

Domain/Architecture Isomorphism

Improve Engineering Practices



What About COTS?*

- **quantum size**: the package
- **incremental change**: generally scores poorly
- **appropriate coupling**: generally scores poorly
- **fitness functions**: generally scores^{very} poorly

*Commercial Off-the-shelf Software



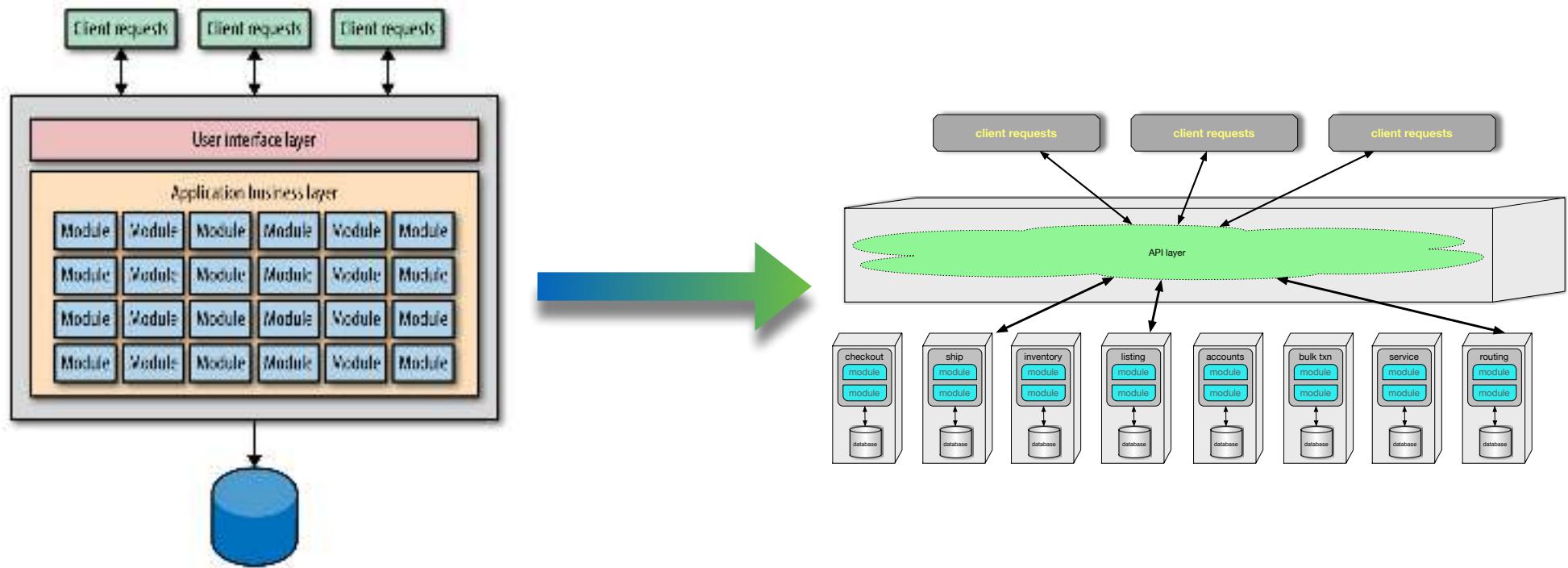
**Work diligently to
hold integration
points to your level of
maturity...**

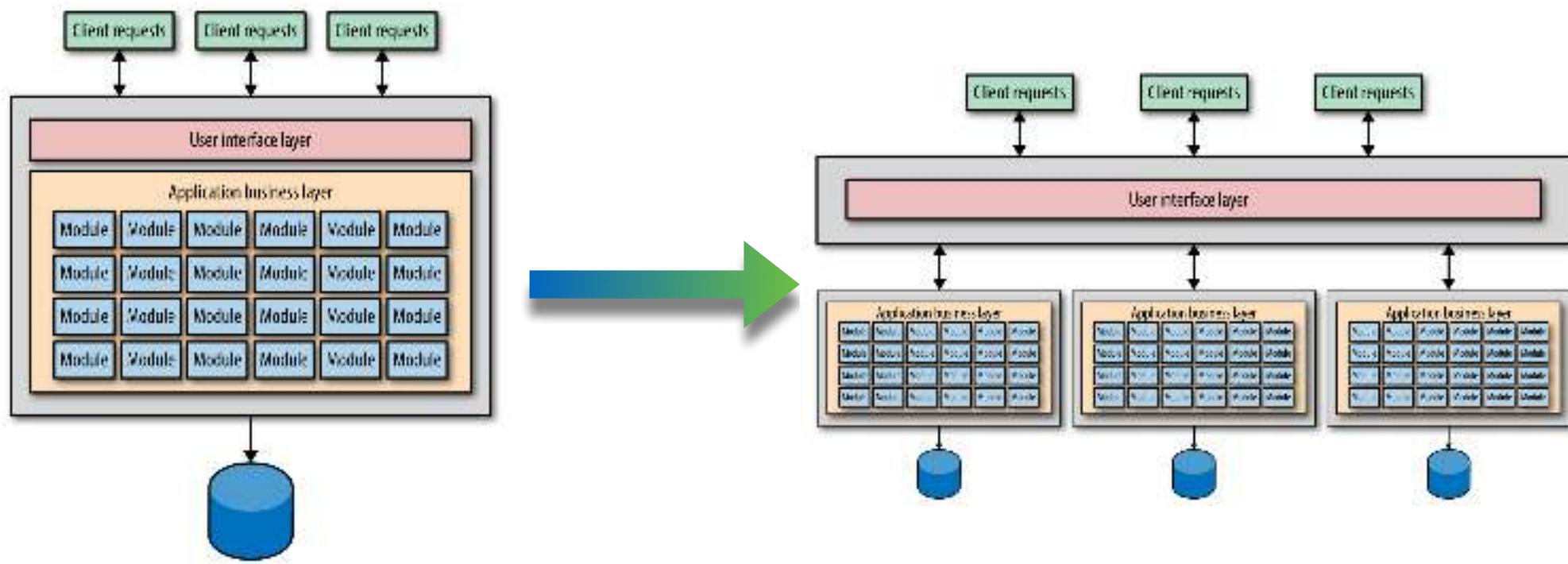


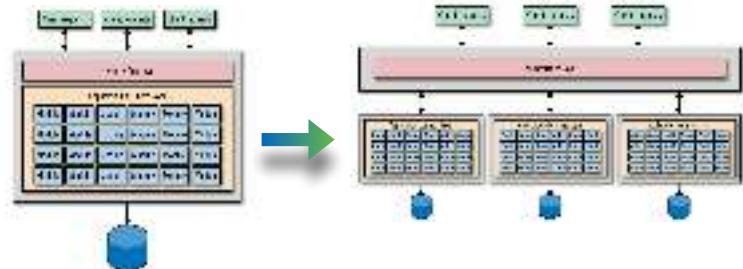
**...if that isn't possible,
realize that some parts of
the system will be
easier for developers to
evolve than others.**



Migrating Architectures







service granularity

coupling points like shared Libraries

transactional boundaries

database issues

Books & Videos



Service-Based Architectures

Structure, Engineering Practices, and Migration

By Neal Ford, Mark Richards

Publisher: O'Reilly Media

Release Date: July 2015

Duration: 6 hours 04 minutes

Explore a variety of service-based architectures—including immensely popular microservices—in this video course from Neal Ford and Mark Richards. Through a series of instructive visuals, you'll be able to compare and contrast these architectures in several ways, including their structure, engineering practices, and deployment. You'll also examine the challenges of migrating from monolithic service-oriented architectures (SOA) to smaller service-based models, and the effects this change can have on team building and company culture in general.

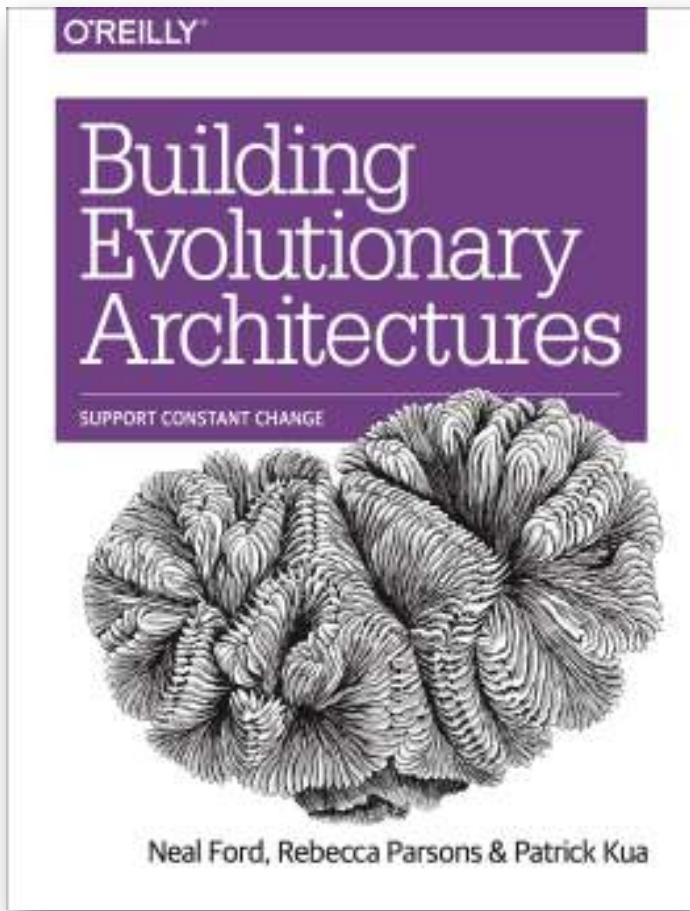
Through the course of this video, you will:

- Learn microservice principles and the organization of its components
- Review the common engineering practices used in microservice architectures
- Examine remote-access protocols of different service-based architecture styles
- Explore microservice-testing scenarios, especially in Continuous Delivery environments
- Manage shared components between services, and define your level of service granularity
- Learn about governance: how to identify, curate, and manage service architectures
- Use well-defined steps to migrate from a large SOA to a service-based architecture

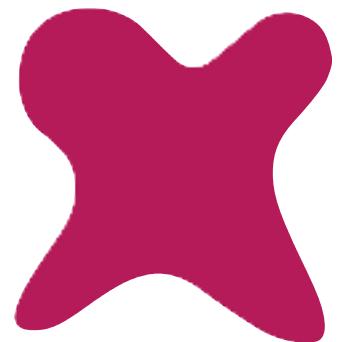
<http://shop.oreilly.com/product/0636920042655.do>

Building Evolutionary Architectures

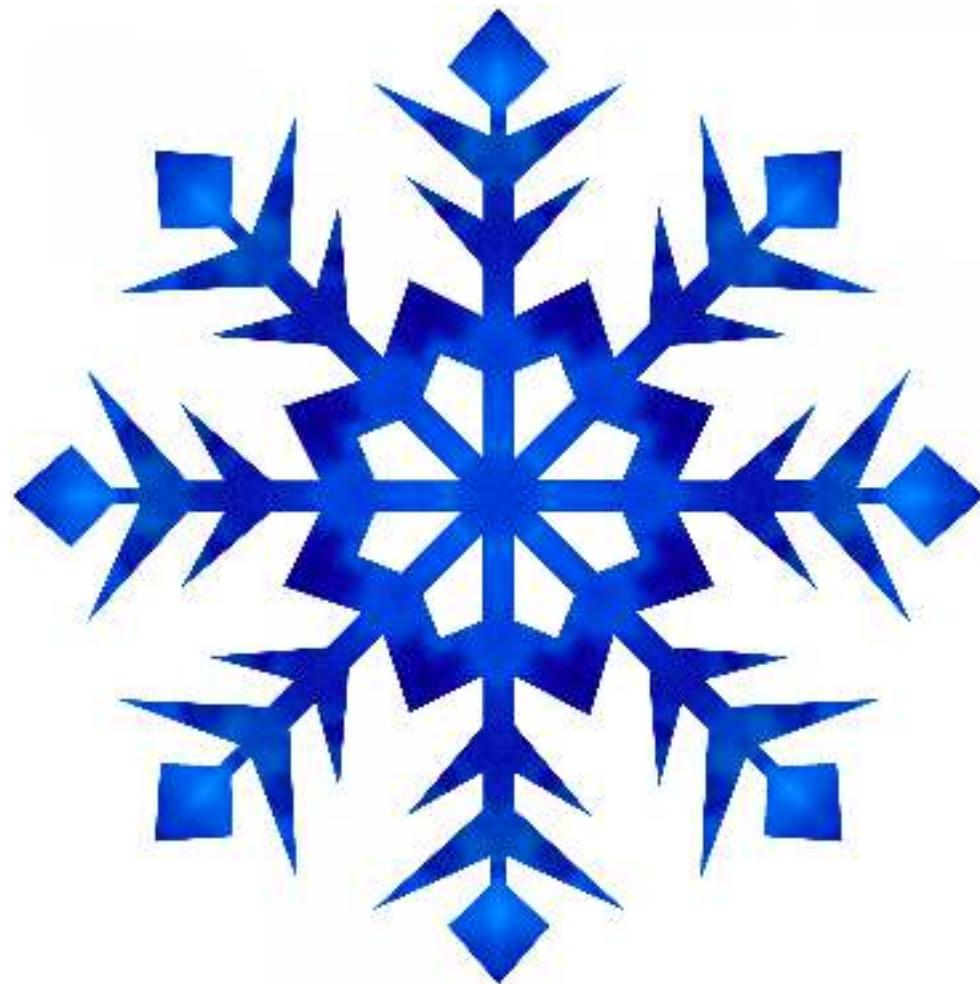
GUIDELINES FOR EVOLUTIONARY ARCHITECTURES



Remove Needless Variables



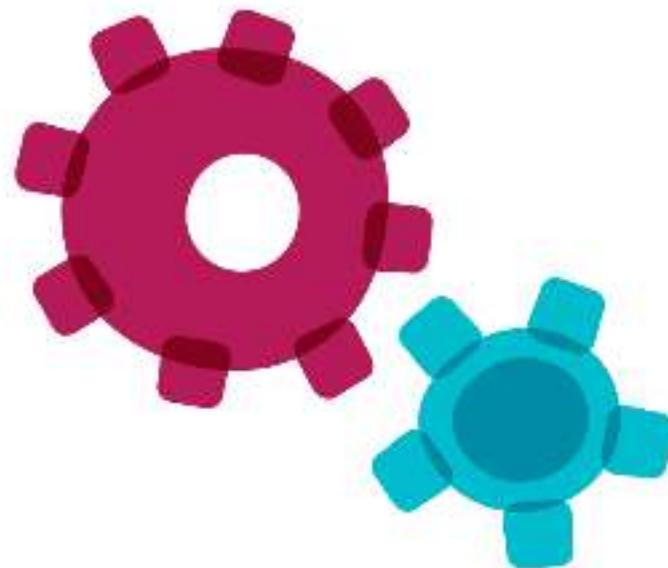
Remove Needless Variables





Remove Needless Variables

Remove Needless Variables



Knight Capital

dougseven.com/2014/04/17/knightmare-a-devops-cautionary-tale/

The screenshot shows a blog post titled "Knightmare: A DevOps Cautionary Tale" by Doug Seven. The post discusses a real-life incident where a company went bankrupt due to a failed deployment. It includes a photo of Doug Seven and a search bar.

DOUG SEVEN
Something can be learned in the course of observing things

HOME IOT WORKSHOP PRODUCT MANAGEMENT ABOUT POSTS COMMENTS

You are here: Home / DevOps / Knightmare: A DevOps Cautionary Tale

Knightmare: A DevOps Cautionary Tale

APRIL 17, 2014 BY DOUG SEVEN 31 COMMENTS

★★★★★ 70 Votes

I was speaking at a conference last year on the topics of DevOps, Configuration as Code, and Continuous Delivery and used the following story to demonstrate the importance making deployments fully automated and repeatable as part of a DevOps/Continuous Delivery initiative. Since that conference I have been asked by several people to share the story through my blog. This story is true - this really happened. This is my telling of the story based on what I have read (I was not involved in this).

This is the story of how a company with nearly \$400 million in assets went bankrupt in 45 minutes because of a failed deployment.

Background

Knight Capital Group is an American global financial services firm engaging in market making, electronic trading, and institutional sales and trading. In 2012 Knight was the largest trader in US equities with market share of around 17% on both the NYSE and NASDAQ. Knight's Electronic

SEARCH

Search this website...

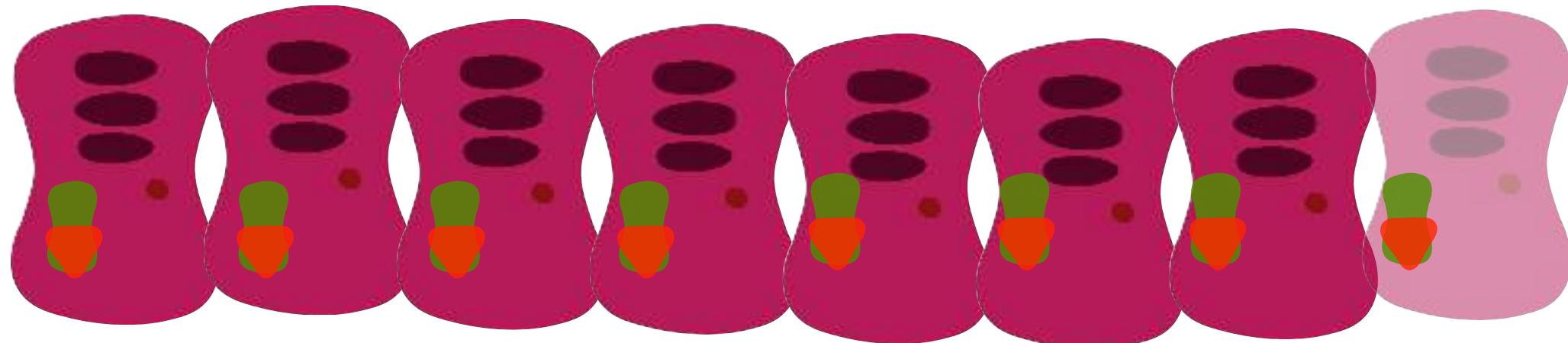
RECENT POSTS:

- IoT Workshop: Java 5 - Input Control Output
- Arduino: reading Analog Voltage
- Arduino 'Hello, World!' - sending digital output

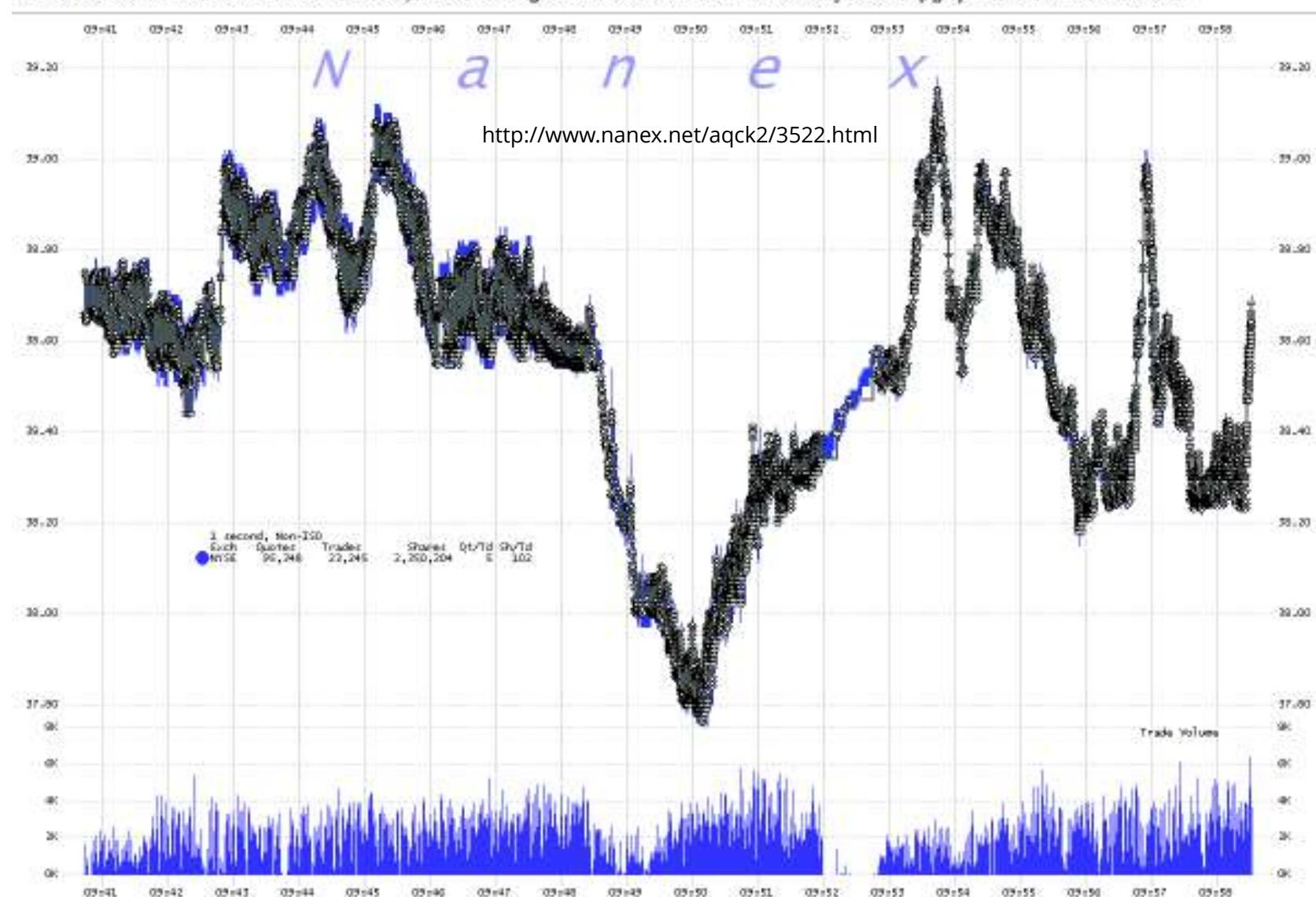
"bankrupt in 45 minutes"

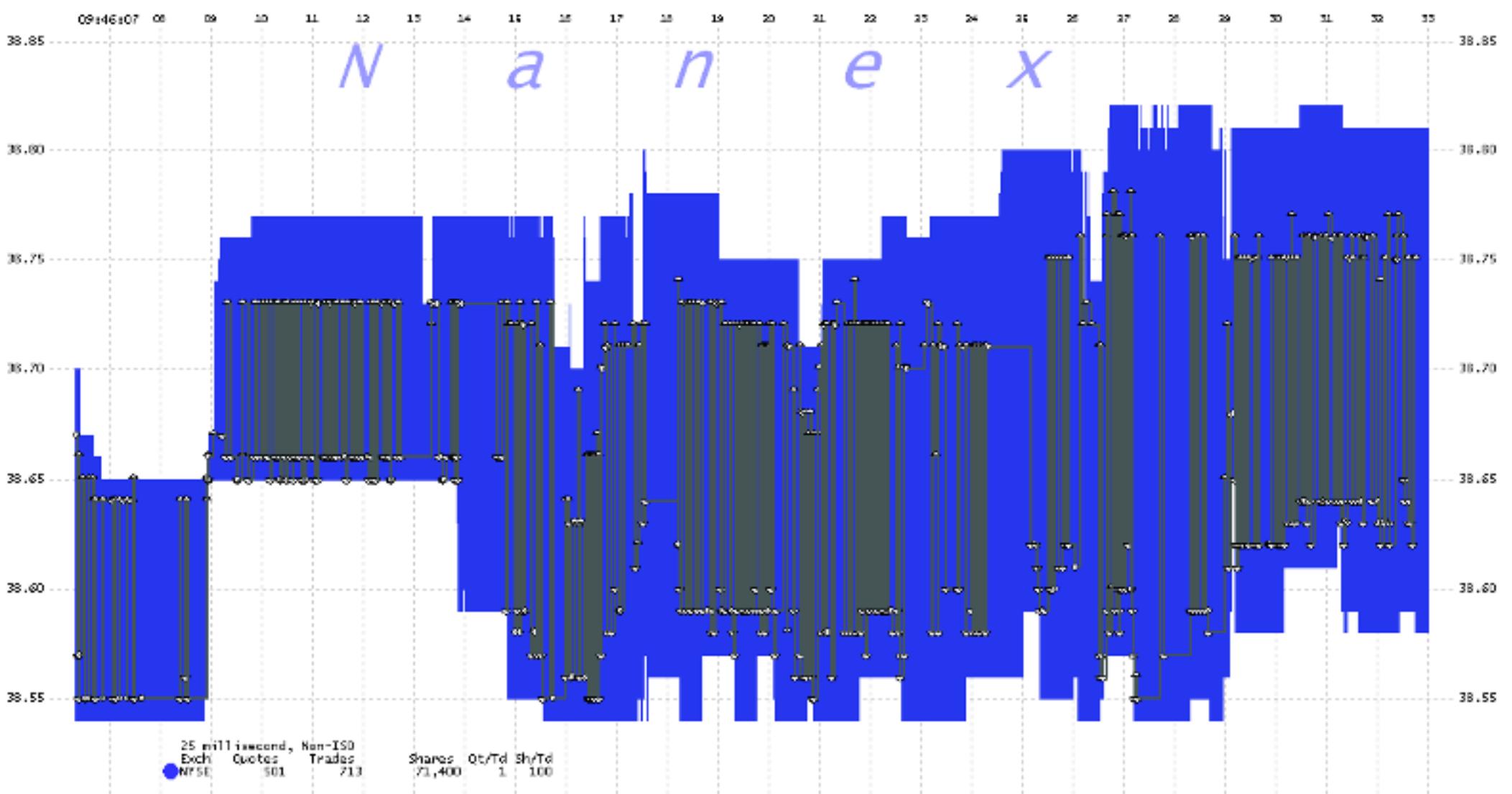
SMARS

PowerPeg



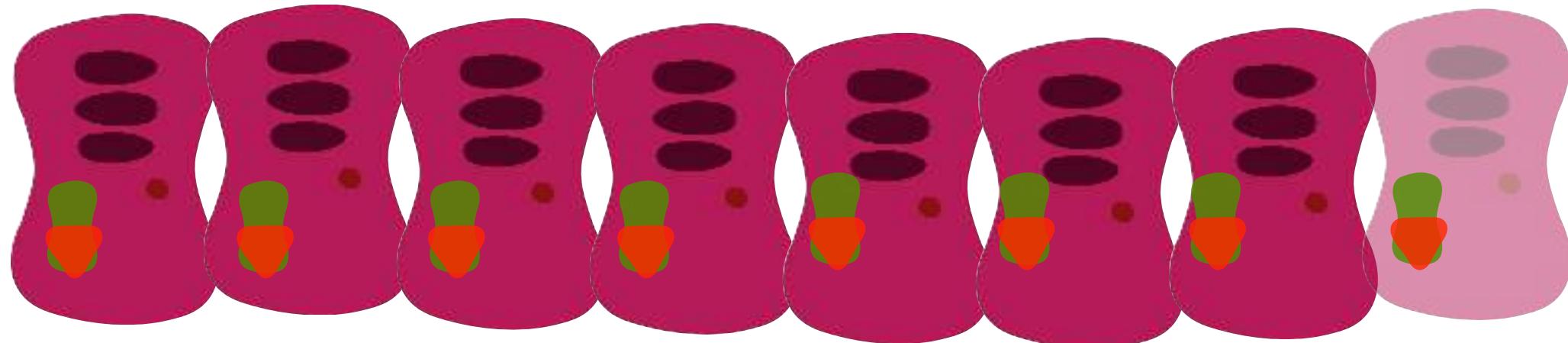
1. EXC One second interval chart. Circles are trades, the blue coloring is the NYSE bid and ask which is mostly covered by gray lines that connect the trades.





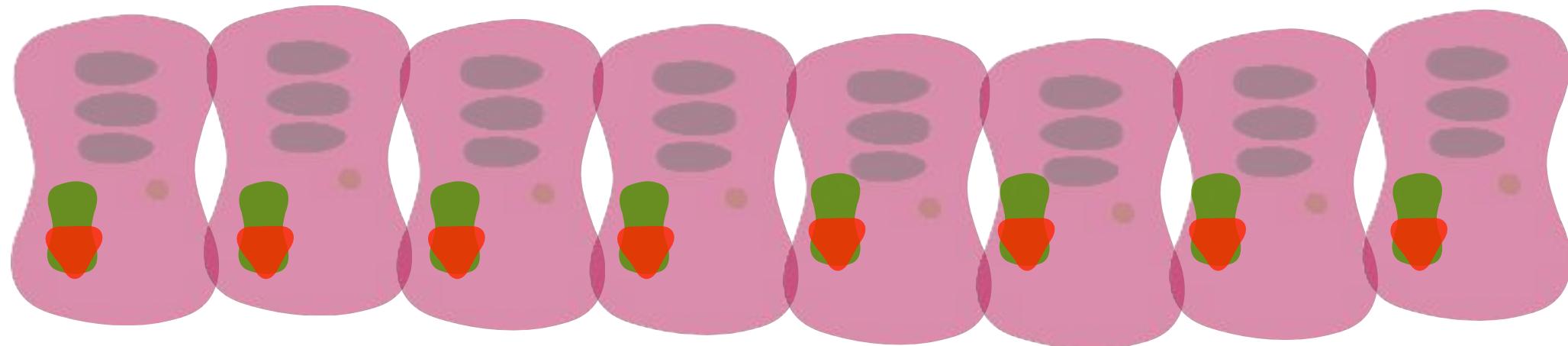
SMARS

PowerPeg



SMARS

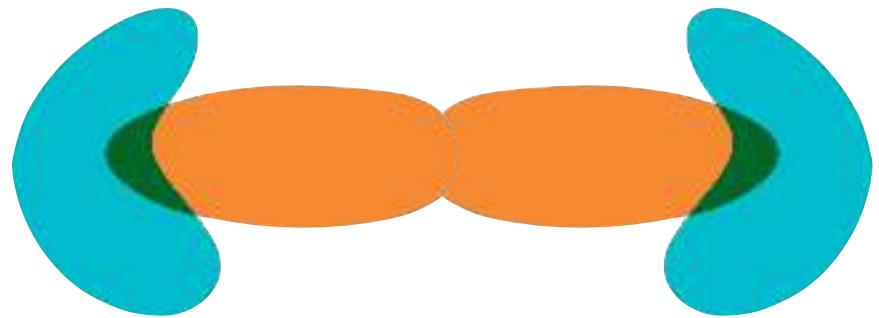
PowerPeg



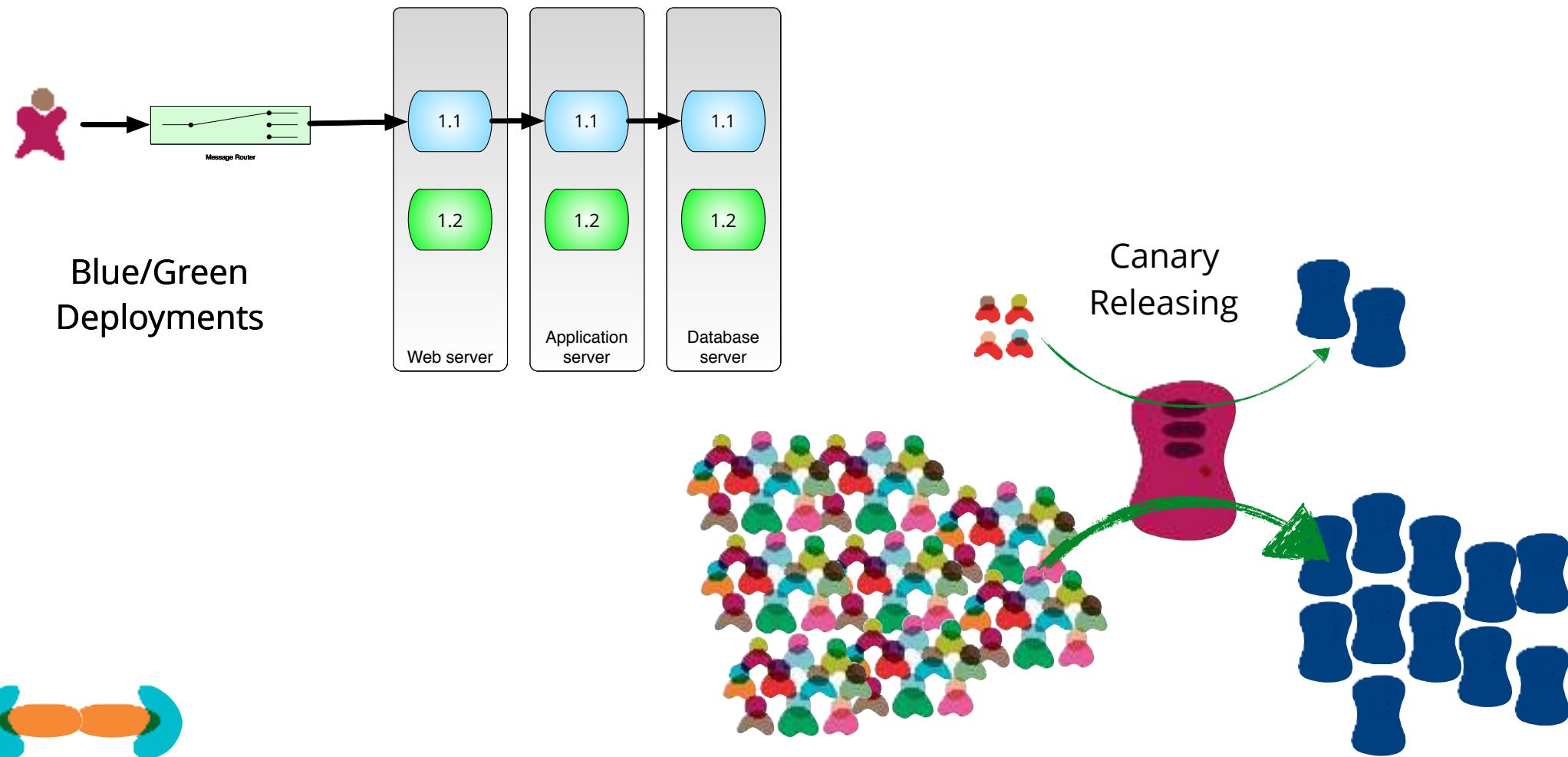


**Identify and remove
needless variability.**

Make Decisions Reversible



Make Decisions Reversible





**Make as many decisions as
possible reversible
(without overengineering).**



**Prefer Evolvable over
Predictable**



...because as we know, there are known knowns; there are things we know we know.

We also know there are known unknowns; that is to say we know there are some things we do not know.

But there are also unknown unknowns—the ones we don't know we don't know.

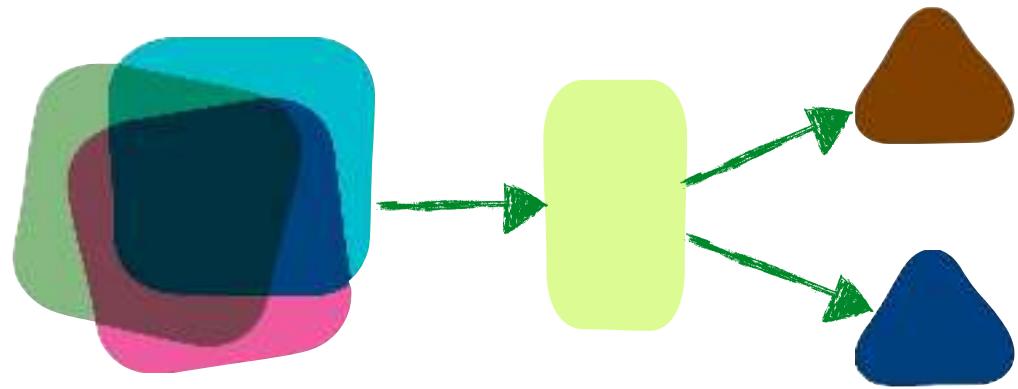
-former US Secretary of Defense Donald Rumsfeld

unknown unknowns

All architectures become iterative because of unknown unknowns; agile just recognizes this and does it sooner.

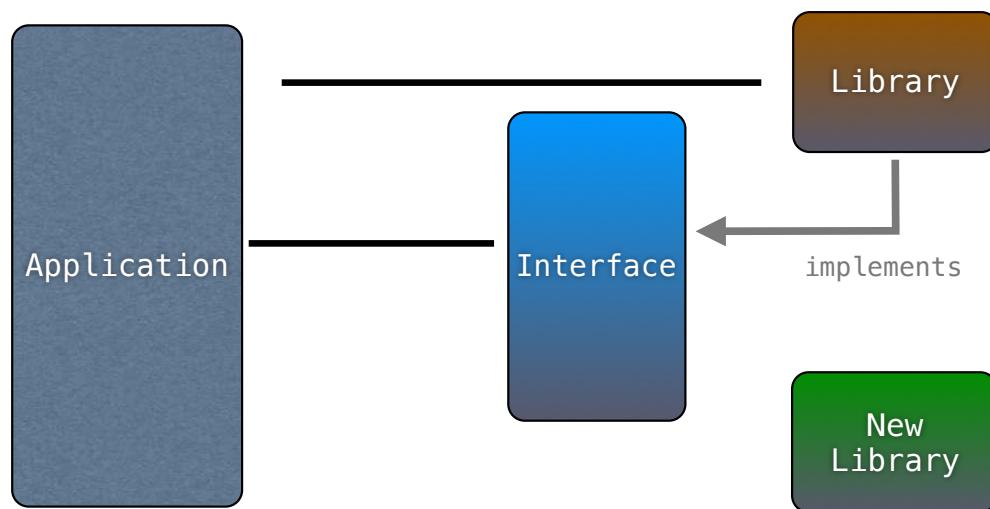
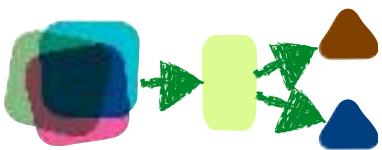
-Mark Richards

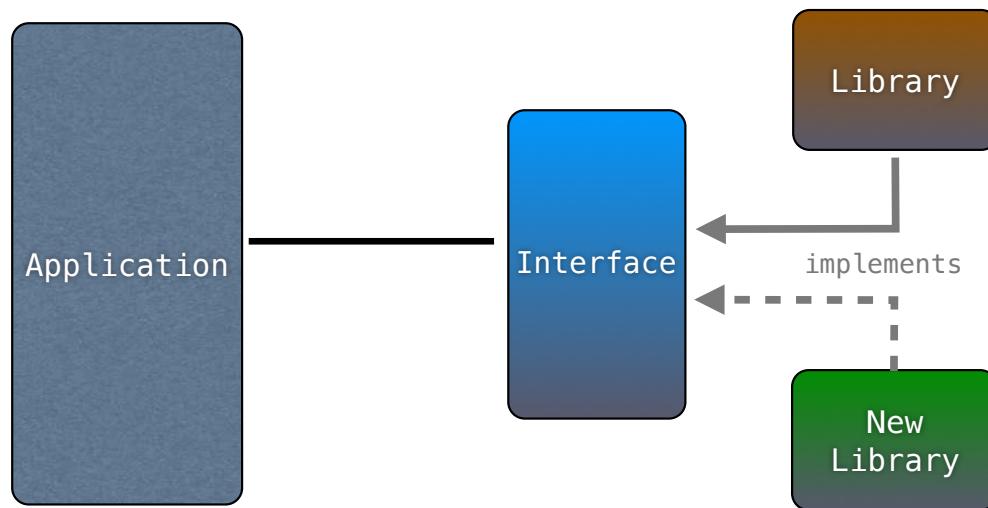
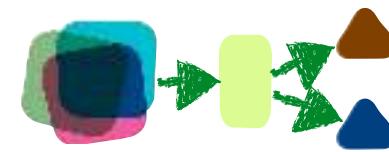


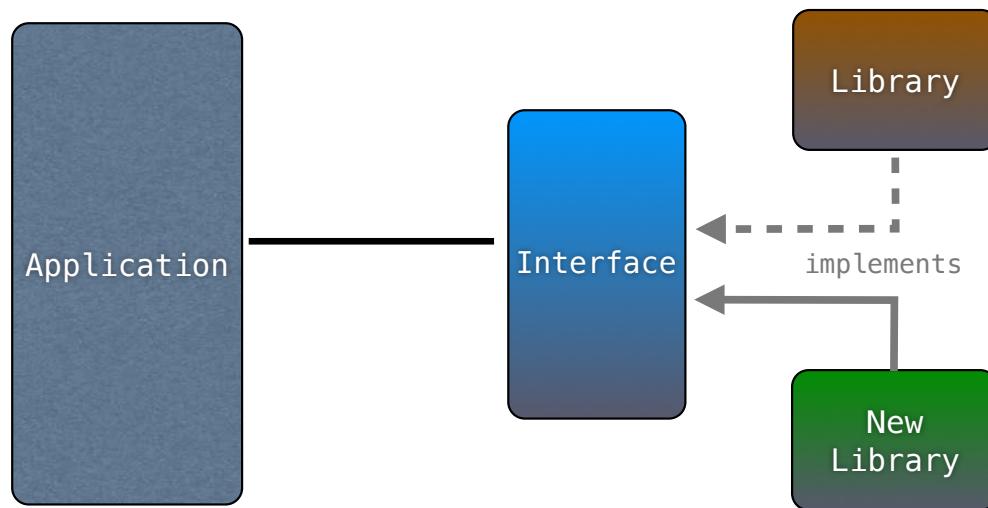
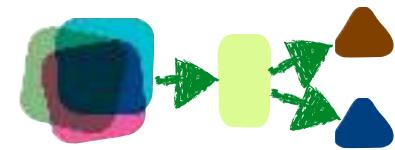


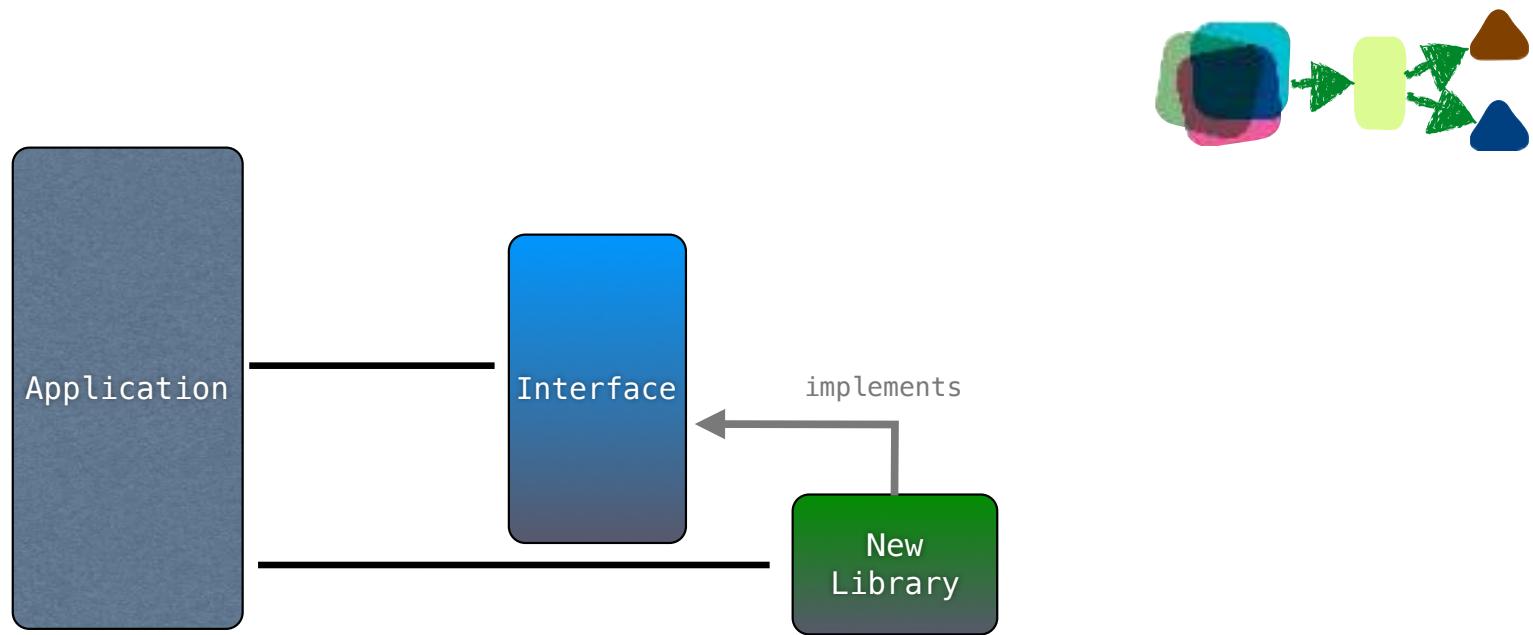
Build Anti-corruption Layers



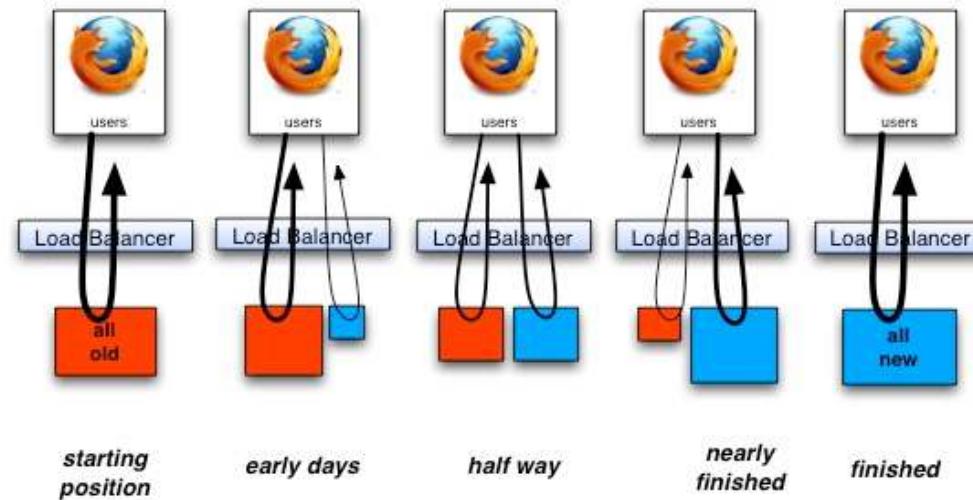








Strangler Pattern

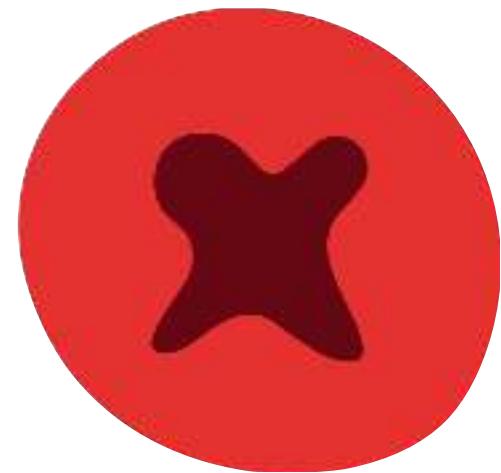


make something new that obsoletes a small percentage of something old

put them live together

rinse, repeat

Build Sacrificial Architectures



MARTINFOWLER.COM

Intro Videos Design Agile Refactoring FAQ About Me All Sections ThoughtWorks

SacrificialArchitecture

Martin Fowler
20 October 2014

You're sitting in a meeting, contemplating the code that your team has been working on for the last couple of years. You've come to the decision that the best thing you can do now is to throw away all that code, and rebuild on a totally new architecture. How does that make you feel about that doomed code, about the time you spent working on it, about the decisions you made all that time ago?

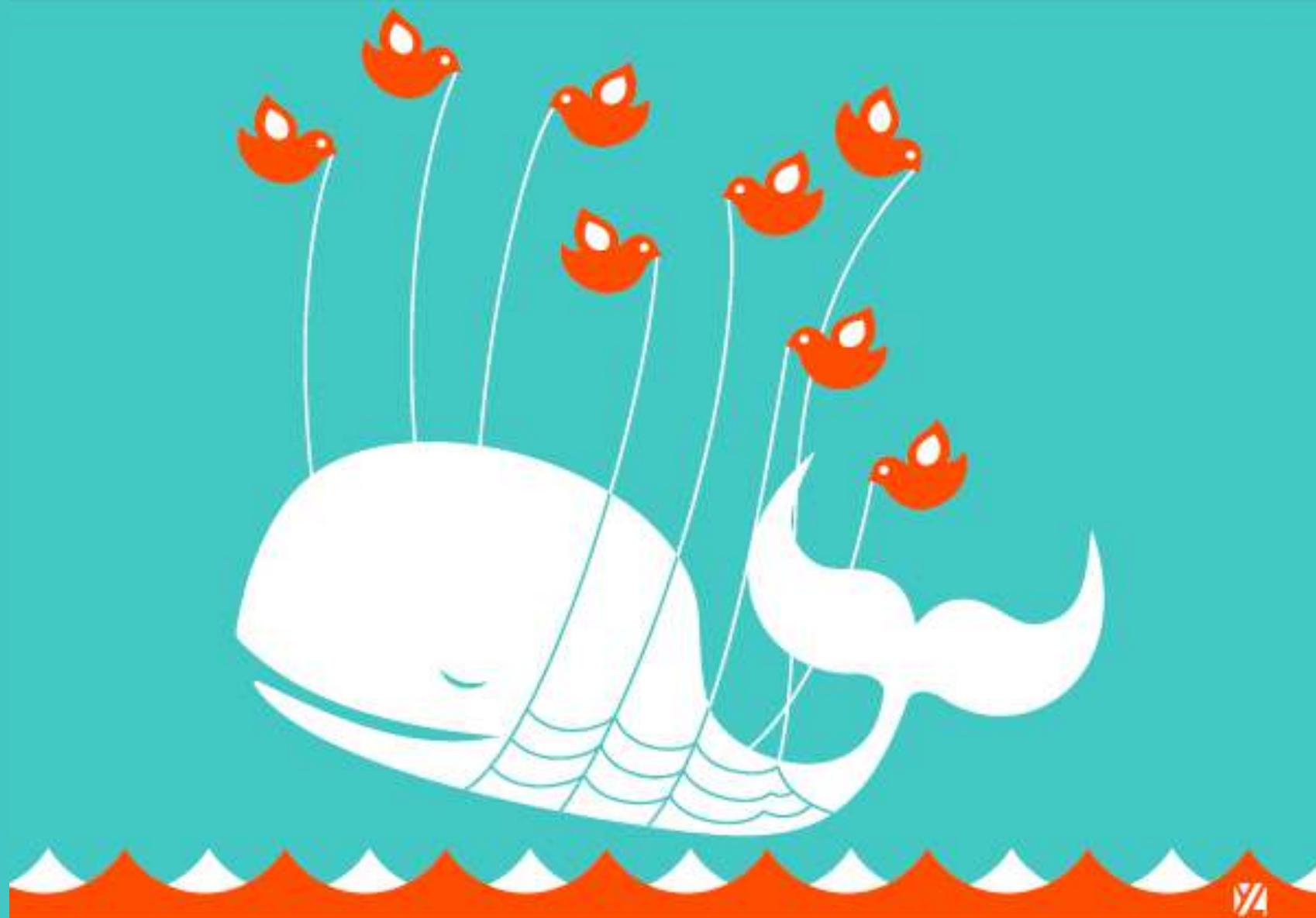
For many people throwing away a code base is a sign of failure, perhaps understandable given the inherent exploratory nature of software development, but still failure.

But often the best code you can write now is code you'll discard in a couple of years time.

Find similar articles at these tags

process theory
evolutionary design
application architecture

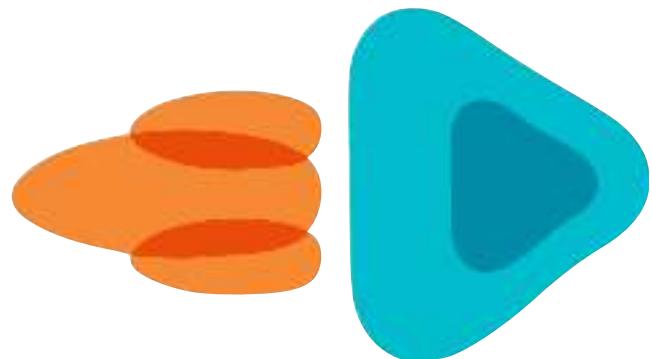
<https://martinfowler.com/bliki/SacrificialArchitecture.html>



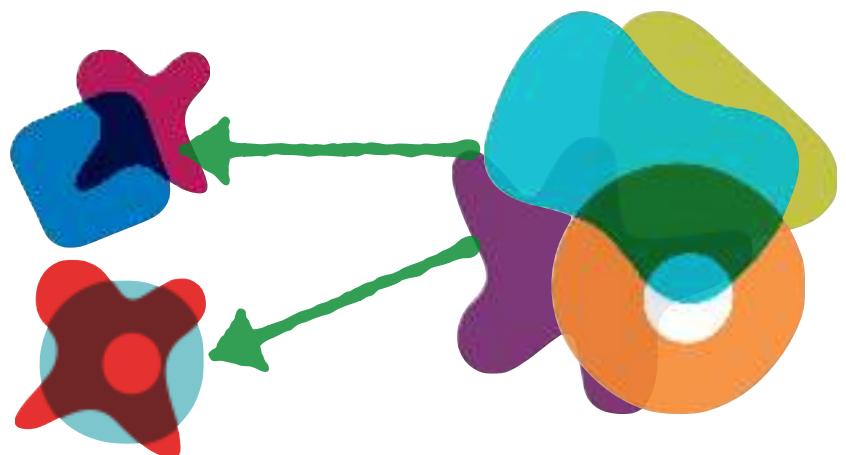


**Build Sacrificial
Architectures**

Mitigate External Change



(transitive) dependency management



```
1 module.exports = leftpad;
2 function leftpad(str, len, ch) {
3     str = String(str);
4     var i = -1;
5     if (!ch && ch !== 0) ch = ' ';
6     len = len - str.length;
7     while (++i < len) {
8         str = ch + str;
9     }
10    return str;
11 }
```

The screenshot shows a GitHub pull request interface. The title of the pull request is "I've Just Liberated My Modules". The description below the title reads: "Allow modules to provide browserify modules as their default, making the following code work: module.exports = browserify('my-module').bundle();". The body of the pull request contains several paragraphs of text explaining the motivation behind the change, the impact on browserify, and the potential benefits for the community. At the bottom of the pull request page, there is a "Review" section with a "Review Requested" button.

I've Just Liberated My Modules

New Feature for #1444

Allow modules to provide browserify modules as their default, making the following code work:
module.exports = browserify('my-module').bundle();

When I was writing lib, didn't everyone use a common module name? And I didn't want to require everyone to change the name of their module, since they may already be using it. I support capitalizing them, browserify provides it's own single, and using it, who would be stupid to change the name of their module, either? :)

This pull request makes me realize that BWF is overreaching, and when corporate is more concerned than the people, and I do mean most, that's Power to the People.

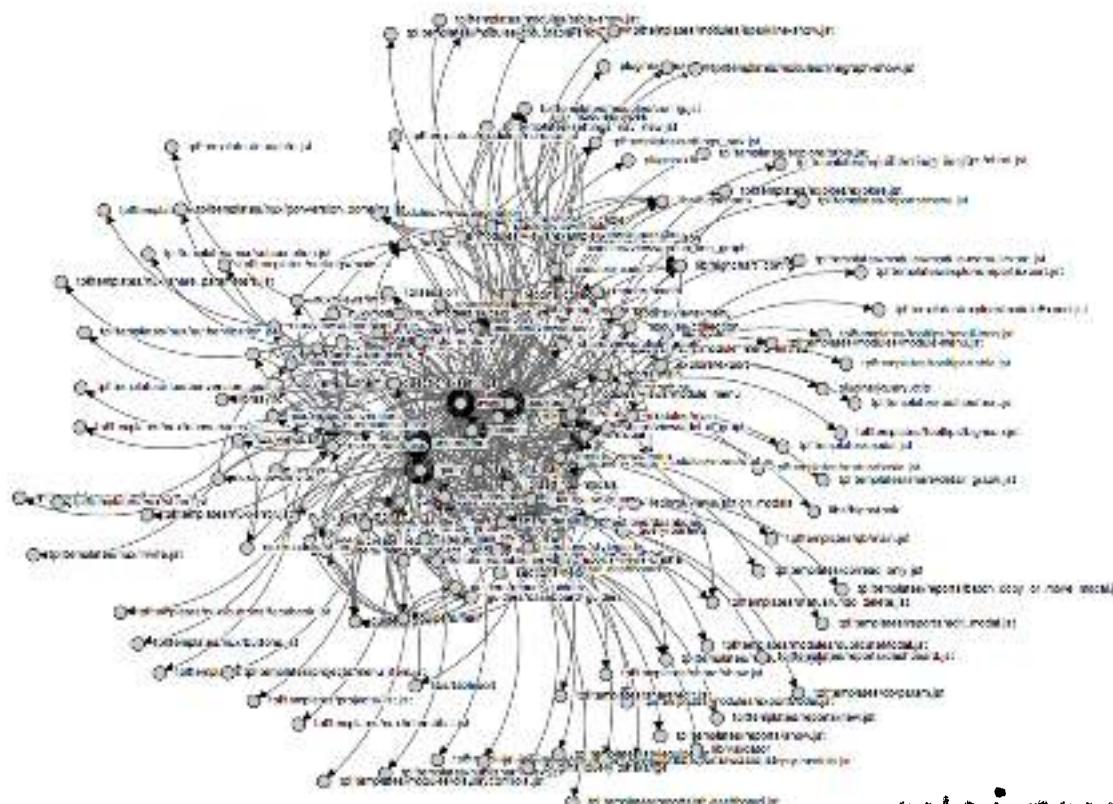
Writing BWF is no longer a job for the ID that is open source, it's a job for the people that are writing it.

This is another good addition to browserify, and believe that soon, source community will eventually create a truly free alternative for BWF.

Please update this if your stuff just gets taken down or something. We can share polyfill responsibility in repositories (new dependency) or have someone take ownership of any module in my opinion. It's really important to the community.

Review Requested

(transitive) dependency management



uniquely ubiquitous



Dependency
management will be
our "Goto Considered
Harmful" moment.

Chris Ford

<https://www.thoughtworks.com/profiles/chris-ford>

A Malicious Module on npm
posted Jan 24, 2016 by Adam Baldwin

boldic.js was a package on npm that was published at version 0.6. It's package.json file had the shell root that executed the command rm -rf ./.tmp. It was created on 01/26/2015 at 12:00 and it remained online until 10/20/2015. It was last updated four days ago by its author (F002) — giving us the date of 10/20/2015.

The goal of this example is to demonstrate what could happen if malicious code is published to npm. It's reasonable to say that it is required to mitigate a rogue package from doing harm.

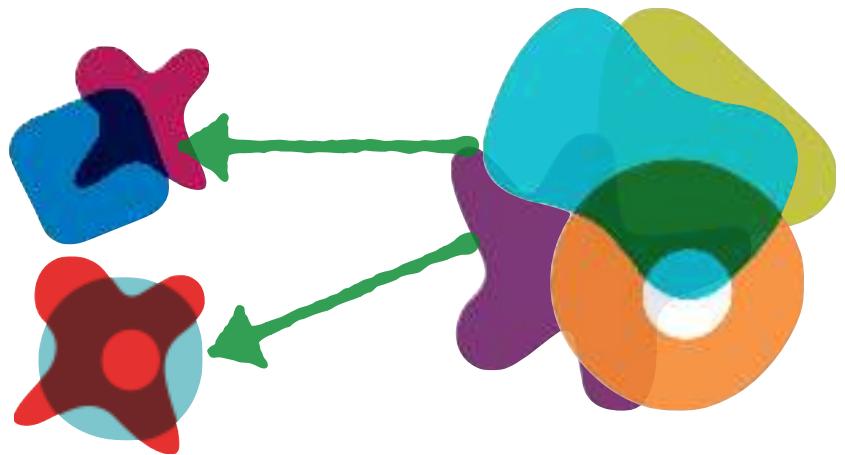
There are two ways to stop or warn of something like this:

- The Node.js core team can add some checks to detect suspicious code.
- The responsibility of validating packages is left up to the open source software.



immutable dependencies

dependency governance



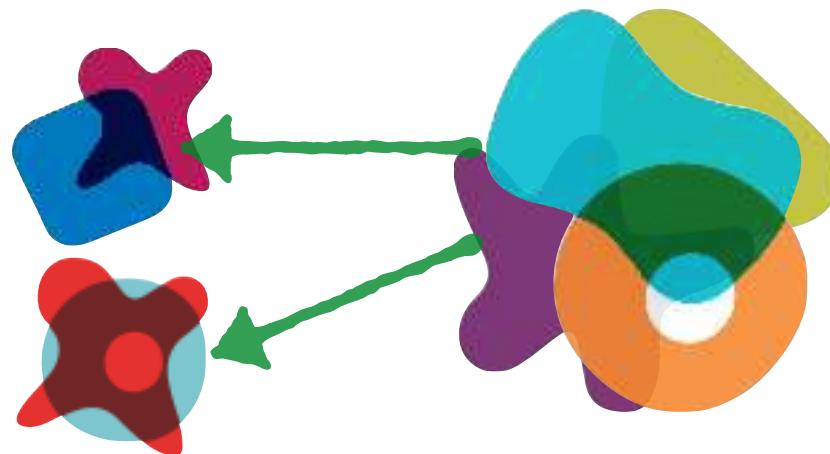
Libraries versus Frameworks

you call its code

it calls your code

pull

push



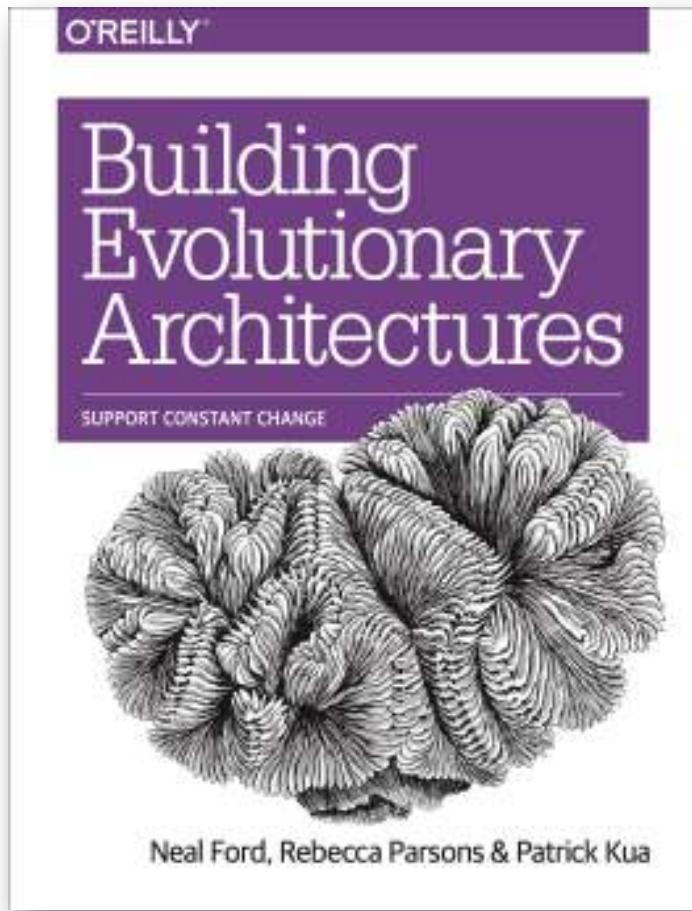


Prefer Continuous
Delivery over snapshots
for (external)
dependencies.



**When versioning services,
prefer internal versioning to
numbering;
support only two versions at a
time.**

Building Evolutionary Architectures



HYPOTHESIS & DATA-DRIVEN DEVELOPMENT

 @neal4d
nealford.com



The Trust Engineers



RADIOLAB

PODCAST ILLUMINATE

The Trust Engineers

Sunday, February 26, 2012 - 08:30 PM

Watch Now

PODCAST

ILLUMINATE

RECENT PODCASTS

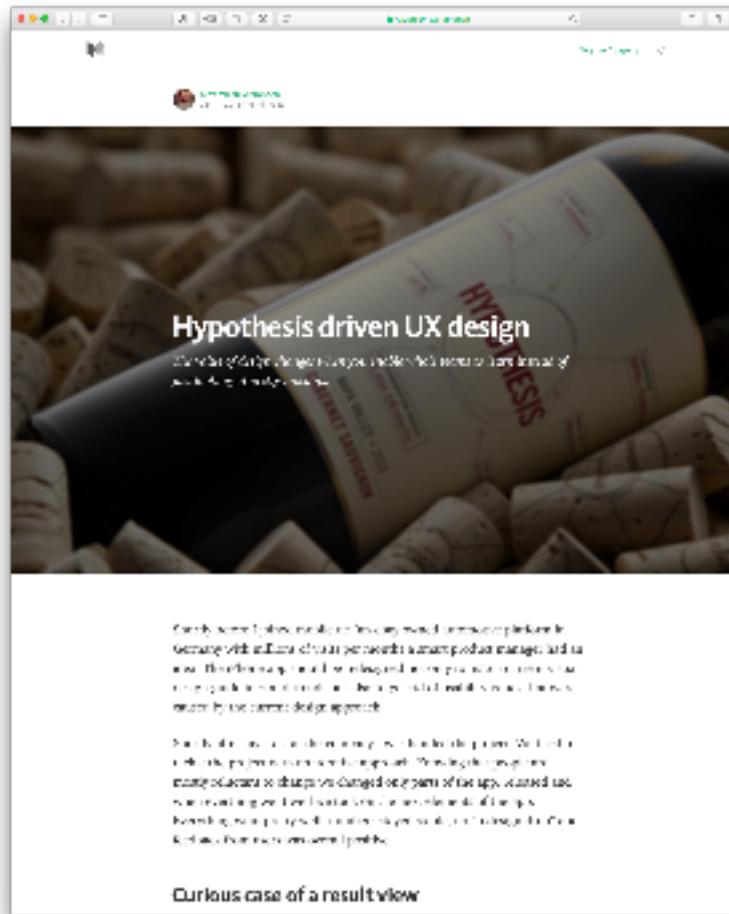
RECENT STORIES

RECENT COMMENTS

Sign Up

<http://www.radiolab.org/story/trust-engineers/>

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

audi e2-de 11:11

6.703 Treffer

Audi A6 Avant 2.0 TDIe 6-Gang
17.200 € 100 kW (136 PS)
EZ 10/2010 128.107 km
Diesel 5,3 l/100 km (komb.)
139 g CO₂/km (komb.)

Audi A6 Avant 2.7 TDI, schöne Le..
17.500 € Neu 132 kW (179 PS)
EZ 09/2006 120 km
Hybrid 7,2 l/100 km (komb.)
(Benzin/Elektro) -102 g CO₂/km (komb.)

Audi A6 Avant 2.6/SV/ZVSHZ/.. P
599 € 258.027 km
EZ 02/1988 100 kW (136 PS)

Audi A6 Avant 2.0 TDIe 6-Gang
17.200 € 100 kW (136 PS)
EZ 10/2010 128.107 km
Diesel 5,3 l/100 km (komb.)
139 g CO₂/km (komb.)

Audi A6 Avant 2.7 TDI, schöne Le..
17.500 € Neu
EZ 09/2006 120 km
Hybrid (Benzin/Elektro) 7,2 l/100 km^{*}
-102 g CO₂/km (komb.)

Sicher Parkplatz Meine Suchen Meine Favoriten Info

More Listings

audi e2-de 11:11

6.703 Treffer

Audi A6 Avant 2.0 TDIe 6-Gang
17.200 € (inkl. 19% MwSt.)
Unfallfrei, Qualitätsseiegel
30179 Hohen Neuendorf, OT Bergfelde
EZ 10/2010 128.107 km
100 kW (136 PS) Diesel
5,3 l/100 km^{*} 139 g/km^{*}

Audi A6 Avant 2.7 TDI,
schöne Lederausstattung..
17.500 € Neu
Unfallfrei, Qualitätsseiegel
65193 Wiesbaden
EZ 06/2006 120 km, 132 kW (179 PS),
Hybrid (Benzin/Elektro) 7,2 l/100 km (komb.), -102 g CO₂/km (komb.)

Audi A6 Avant
2.6/SV/ZV/SHZ/Klimatron.. P
599 €
EZ 02/1988, 258.027 km, 100 kW (136 PS), Benzin

Sicher Parkplatz Meine Suchen Meine Favoriten Info

Better Structure

audi e2-de 11:11

6.703 Treffer

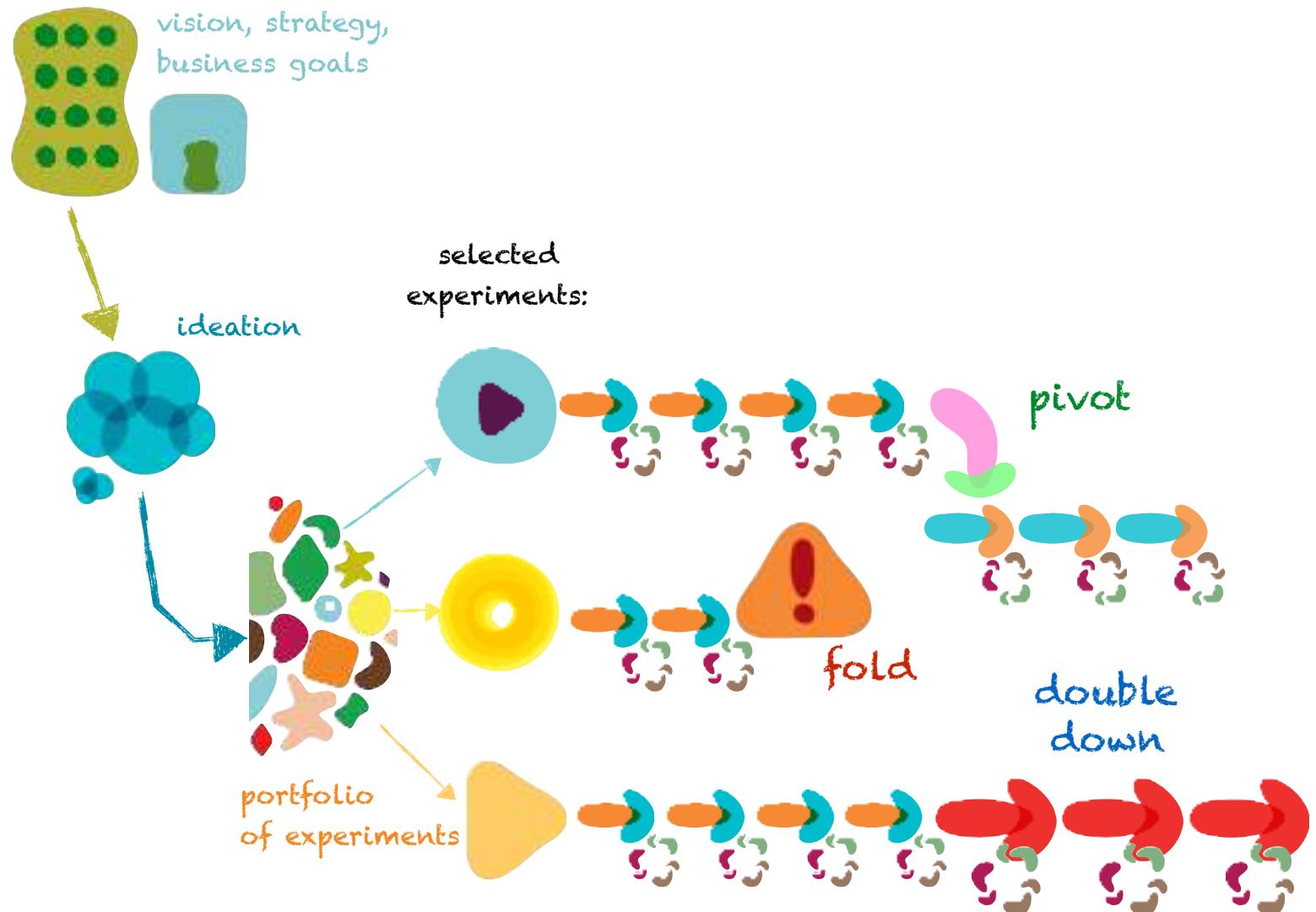
Audi A6 Avant 2.0 TDIe 6-Gang
17.200 €
Unfallfrei, Qualitätsseiegel
30179 Hohen Neuendorf, OT Bergfelde
10/2010, 128.107 km, 100 kW (136 PS), Diesel
-5,3 l/100 km (komb.), -139 g CO₂/km (komb.)

Audi A6 Avant 2.7 TDI,
schöne Lederausstattung..
17.500 € Neu
Unfallfrei, Qualitätsseiegel
65193 Wiesbaden
06/2006, 120 km, 132 kW (179 PS),
Hybrid (Benzin/Elektro) 7,2 l/100 km (komb.), -102 g CO₂/km (komb.)

Audi A6 Avant
2.6/SV/ZV/SHZ/Klimatron.. P
599 €
EZ 02/1988, 258.027 km, 100 kW (136 PS), Benzin

Sicher Parkplatz Meine Suchen Meine Favoriten Info

Better Prioritization



The screenshot shows a web browser displaying a blog post from GitHub's engineering blog. The title of the post is "Move Fast and Fix Things". The date of the post is December 15, 2015. The post discusses the challenges of managing technical debt in a large codebase as it grows and evolves. It highlights the importance of merge conflicts and how GitHub has improved its merge strategy to handle them more effectively. The post also mentions the introduction of a new merge model called "Fast-Follow" and its impact on developer productivity.

Move Fast and Fix Things

Dec 15 · 11 December 15, 2015

Anyone who has worked on a large enough codebase knows that technical debt is inevitable. As an application grows in size and complexity, the more technical debt it accrues. With GitHub's growth over the last 7 years, we have found plenty of room and cruft 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're not big about the "feature" we've taken over the years to see in our web application more than 10 million users. In fact, we do quite the opposite: we make it easier for a developer 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 those reviews efficiently and safely.

As an example, two weeks ago we replaced one of the most critical code paths in our Infrastructure code that performs merges when you push the `Merge Pull Request`. Although we couldn't prevent these kinds of conflicts through 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 merge model that GitHub uses for repositories in our platform and our [Personal](#) offerings. There are many implementation details that we've discussed, related to performance and efficiency, but the most relevant one here is the fact that repositories are always *detached*.

This means that the actual file in the repository (the one that you would see in your working directory when you clone the repository) is not actually available on disk in your repository; they are compressed and stored in the [packfile](#).

Because of this, performing a merge in a production environment is a multi-step process. GitHub uses the [fast-forward](#) merge strategy, but the way this merge strategy is performed by default when using `git merge` to merge two branches in a local repository obscures the existence of a working tree for the repository, which is then checked out on it.

The command we developed in the early days of GitHub for this function is `git merge`, 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 recursive`). With some tweaking, this strategy can be adapted to not require an explicit checkout of the repository.

To accomplish this, we wrote a shell script to run up a temporary working directory, in

Move Fast & Fix Things

```

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.id

  ancestor_tree = merge_base && Rugged::Commit.tee(rugged, merge_base).tree
  merge_options = {
    :full_indexing => true,
    :repo_path => true,
    :obj_encoding => true,
  }

  index = base.tree.merge(head.tree, ancestor_tree, merge_options)
  return nil, "merge_conflict" if 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

```

```

def create_merge_commit(author, base, head, options = {})
  commit_message = options[: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.repo
    e.use { create_merge_commit_gitiauthor, now, base, head, commit_message }
    e.try { create_merge_commit_ruggedauthor, now, base, head, commit_message }
  end
end

```



<https://github.com/github/scientist>



```
require "scientist"

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

    experiment.run
  end
end
```

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



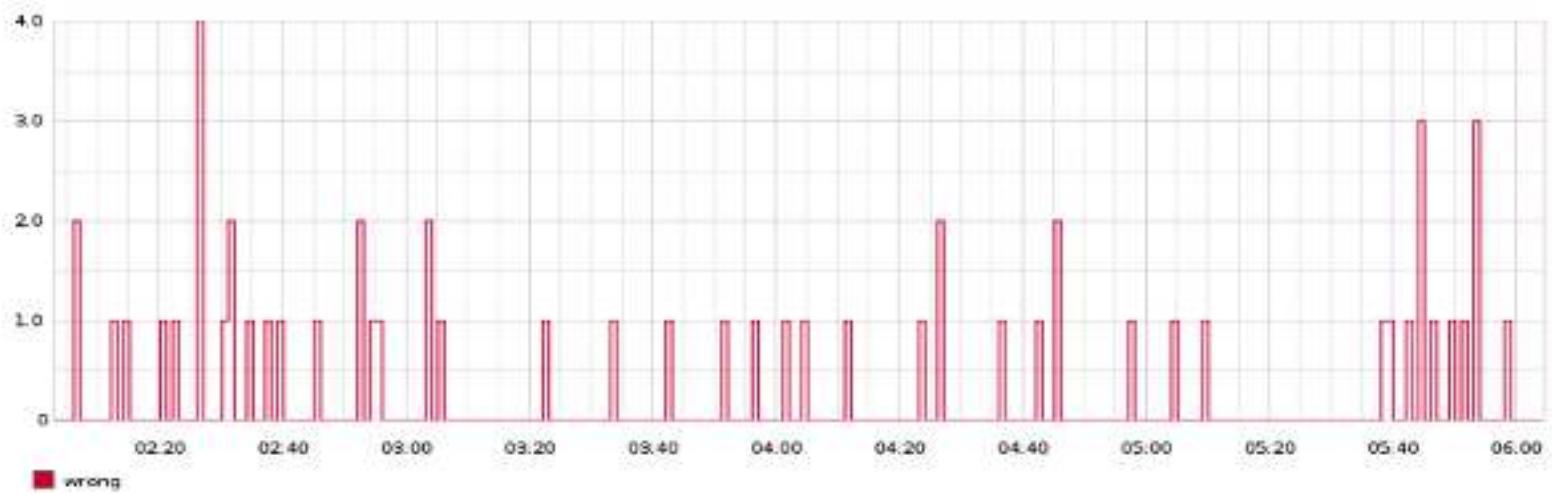
Accuracy

The number of times that the candidate and the control agree or disagree. [View mismatches](#)



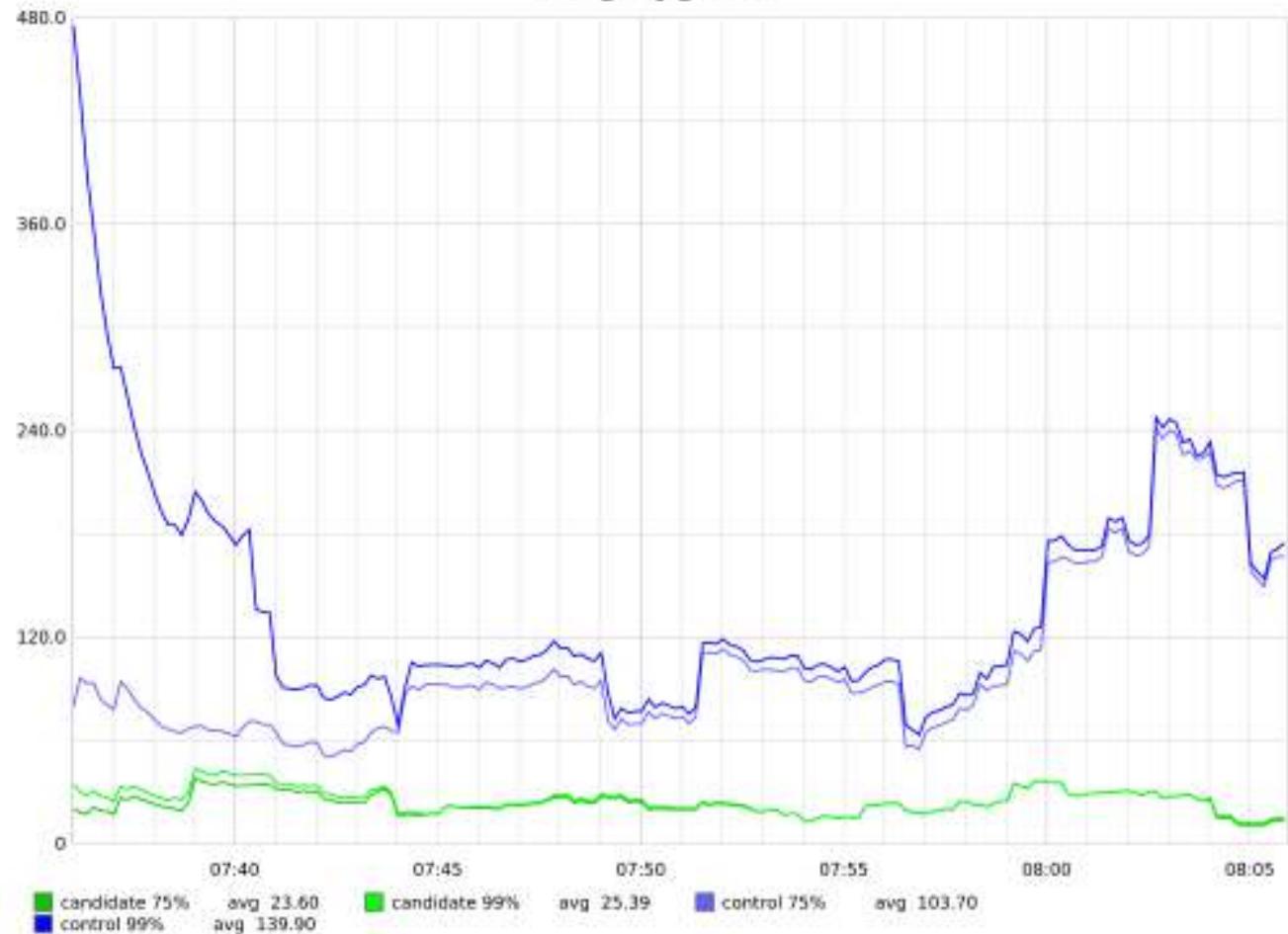


The number of incorrect/ignored items.





create_merge_commit



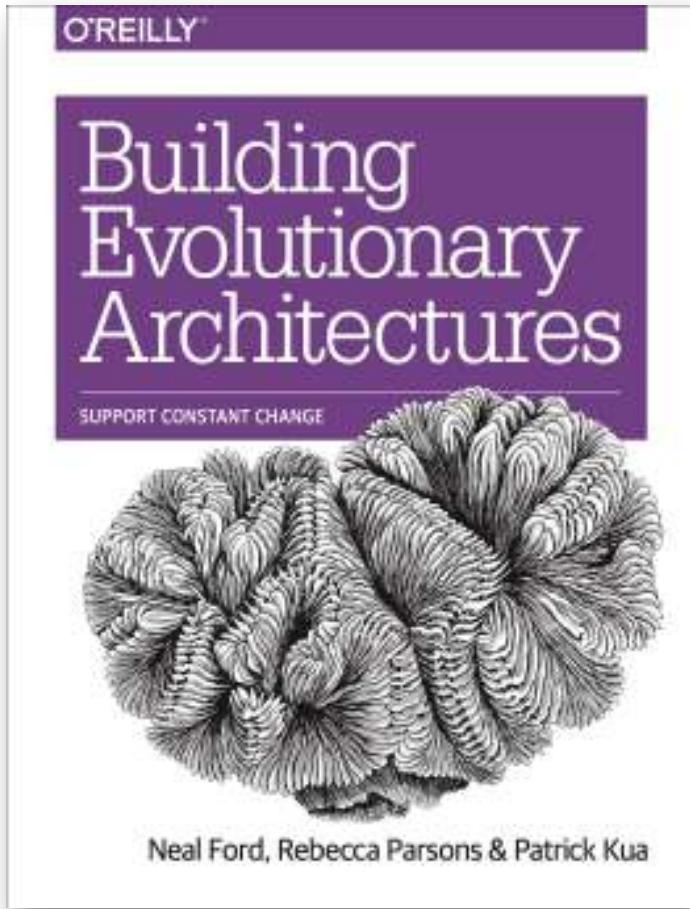
Bugs Found; Resolution

- ❑ faster conflict return because shell script exited immediately; replicated in library
- ❑ index write was causing O(n) problem; inlined into memory
- ❑ the ancestor had a file with a given filemode, while one side of the merge had removed the file and the other side had changed the filemode; bug in git!
- ❑ Github incorrectly successfully merged files w/ 768 conflicts; fixed git shell script
- ❑ new library was skipping an entire step; bug found & fixed

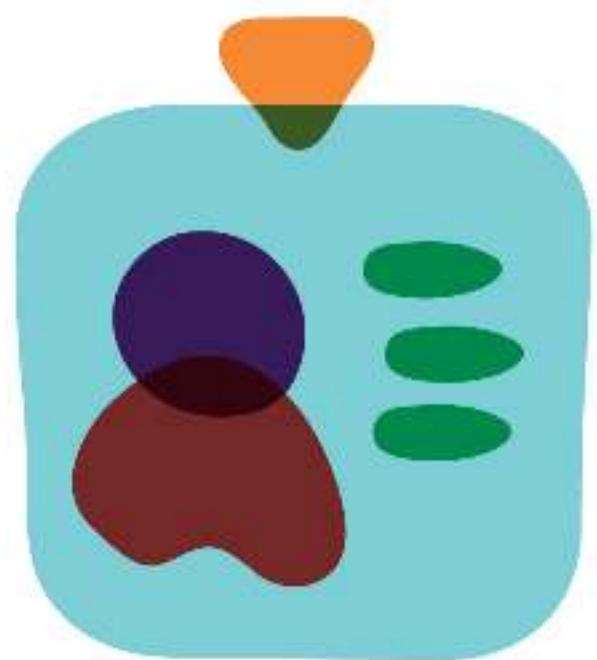


Building Evolutionary Architectures

PUTTING EVOLUTIONARY
ARCHITECTURE INTO PRACTICE



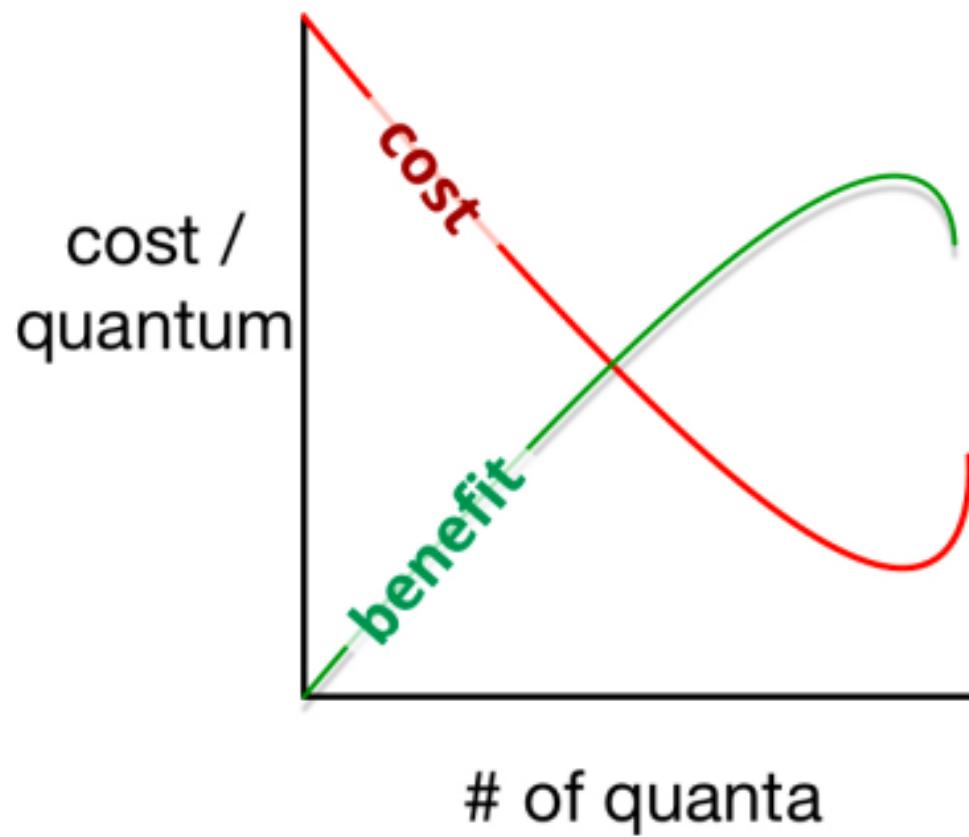
Organizational Factors



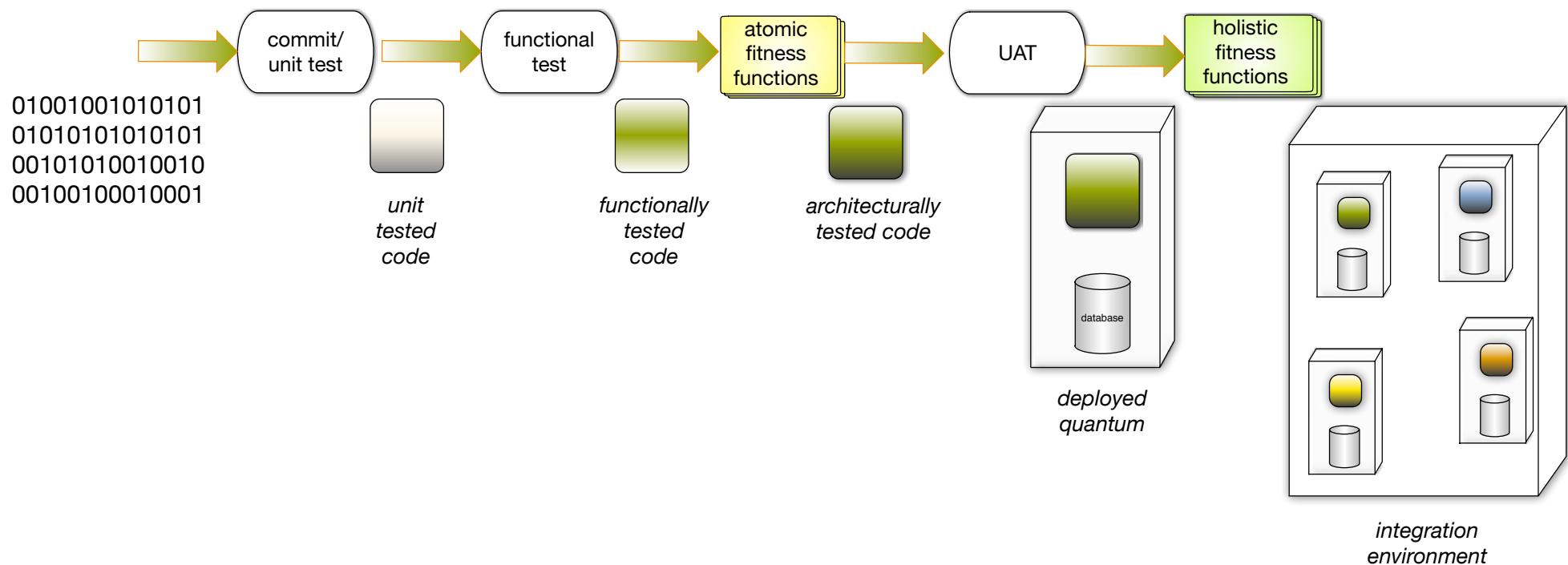
Culture of Experimentation

- Bring ideas from outside
- Encouraging explicit improvement
- Spike and stabilize
- Creating innovation time
- Following set-based development
- Connecting engineers with end-users

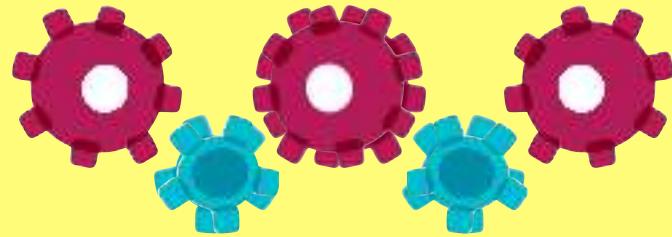
CFO & Budgeting



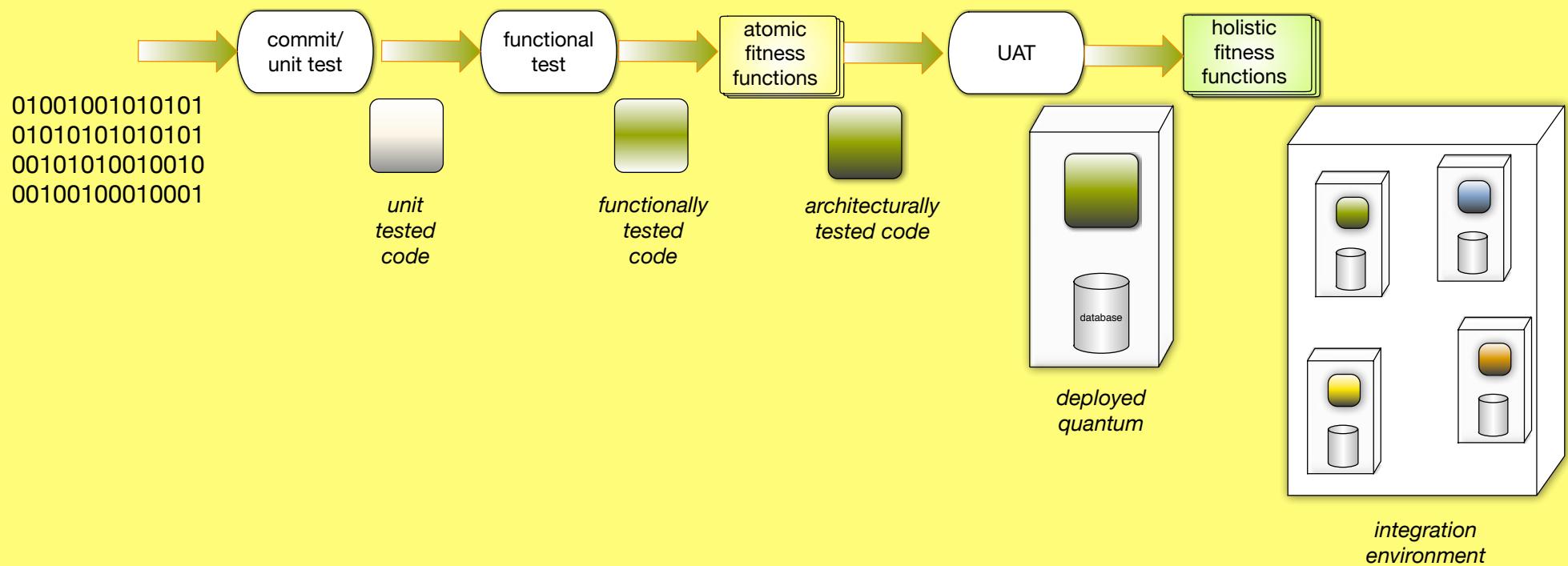
Building Enterprise Fitness Functions



Penultima↑e



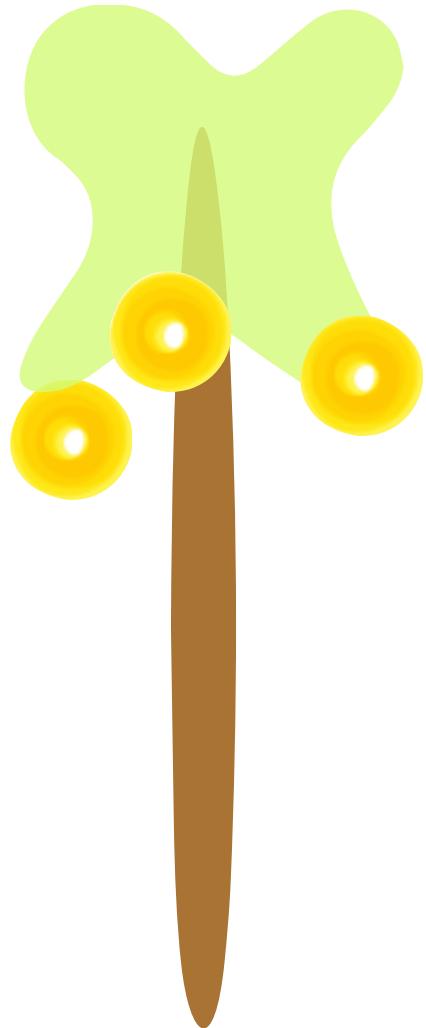
Legality of Open Source Libraries



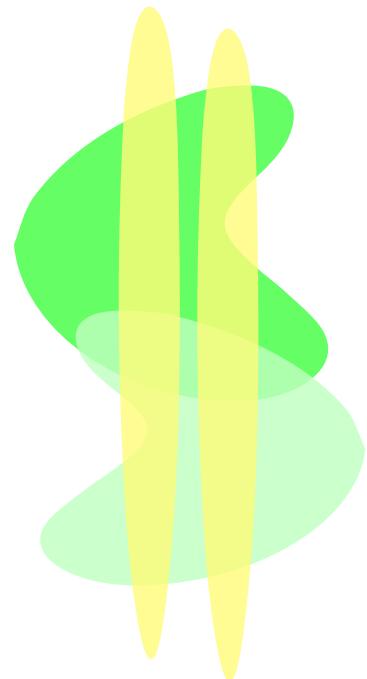
Where to Start?



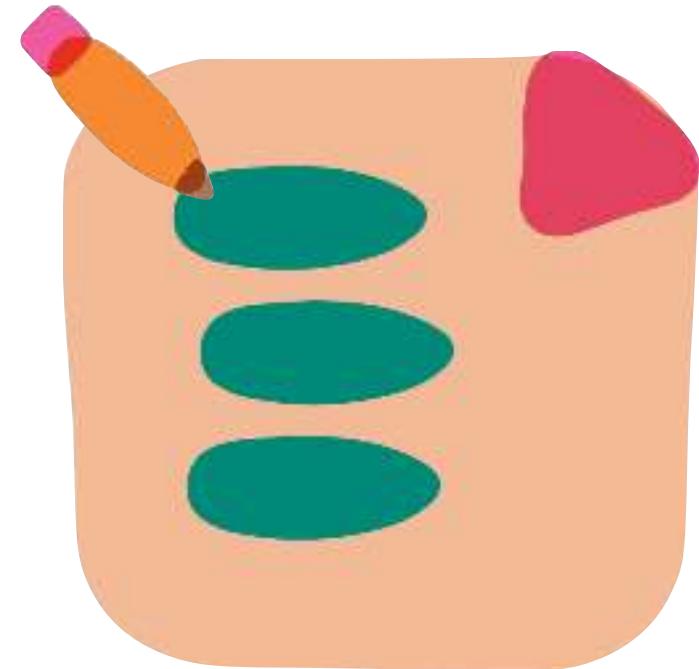
Low-hanging Fruit



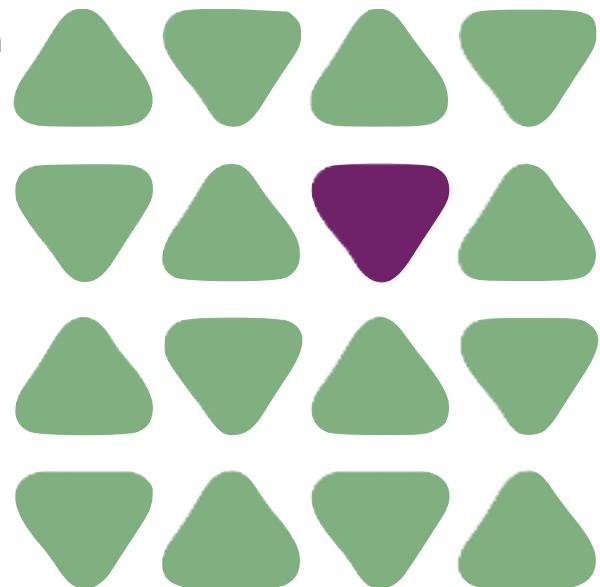
Highest Value



Testing



Infrastructure



Why



Why

- Predictable versus evolvable
- Scale
- Advanced business capabilities
- Cycle time as a business metric
- Isolating architectural characteristics at the quantum level

Why NOT

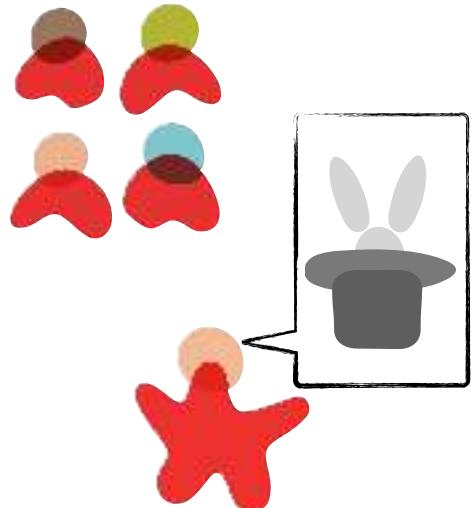


Why NOT

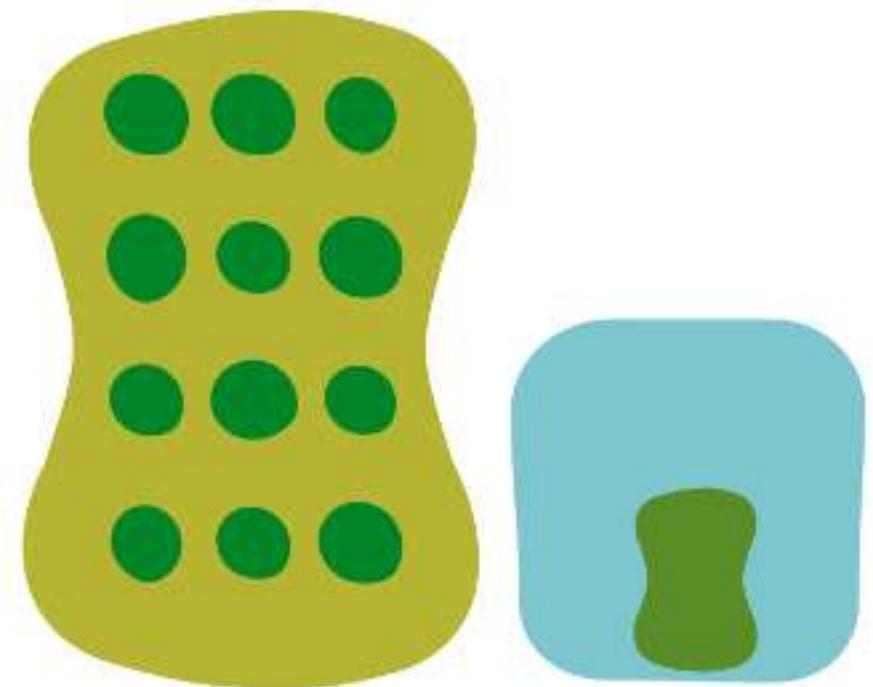
- Can't evolve a ball of mud
- Other architectural characteristics dominate
- Sacrificial architecture
- Planning on closing the business soon

Case Study: Consulting Judo

Demonstration defeats discussion!



The Business Case



The future is already here—it's just not very evenly distributed.

-William Gibson

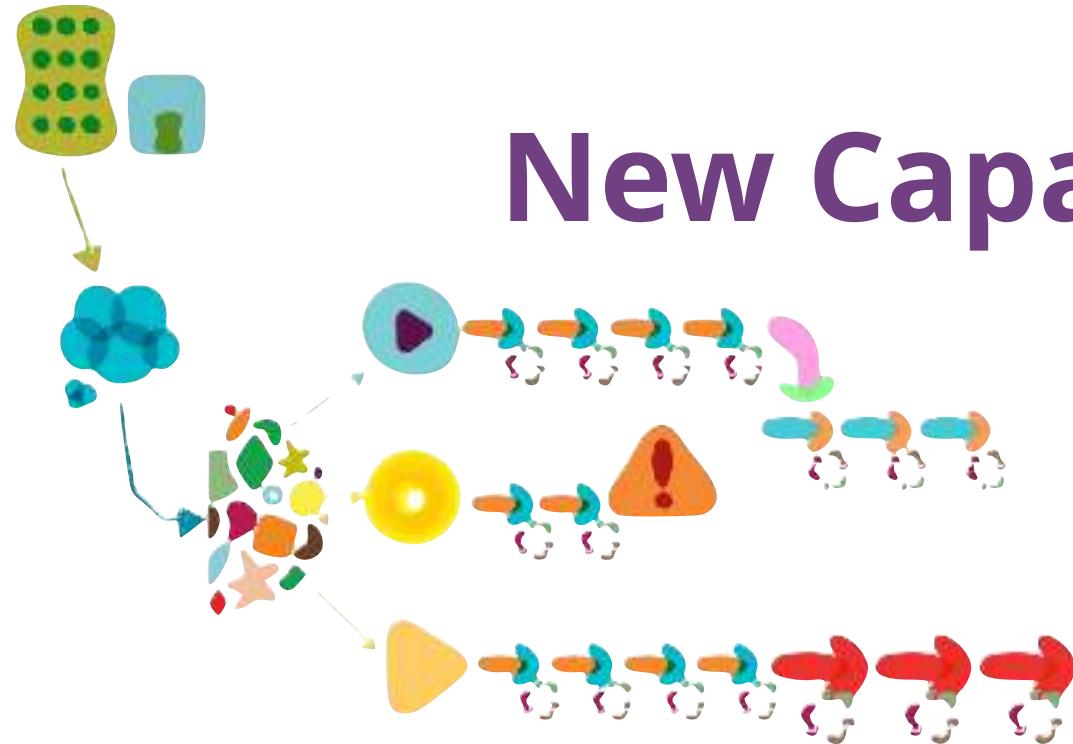
Move Fast without Breaking Things



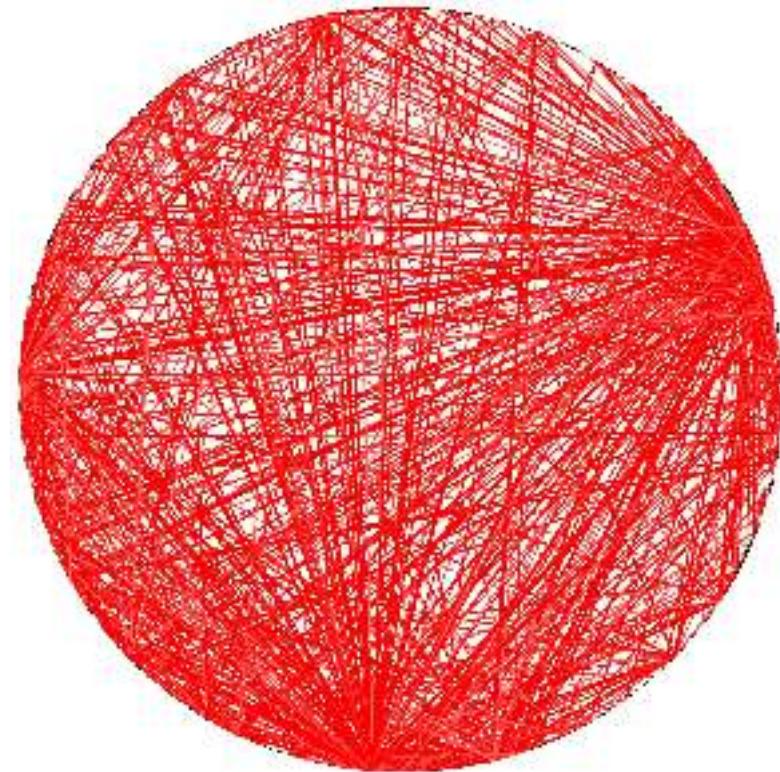
Less Risk



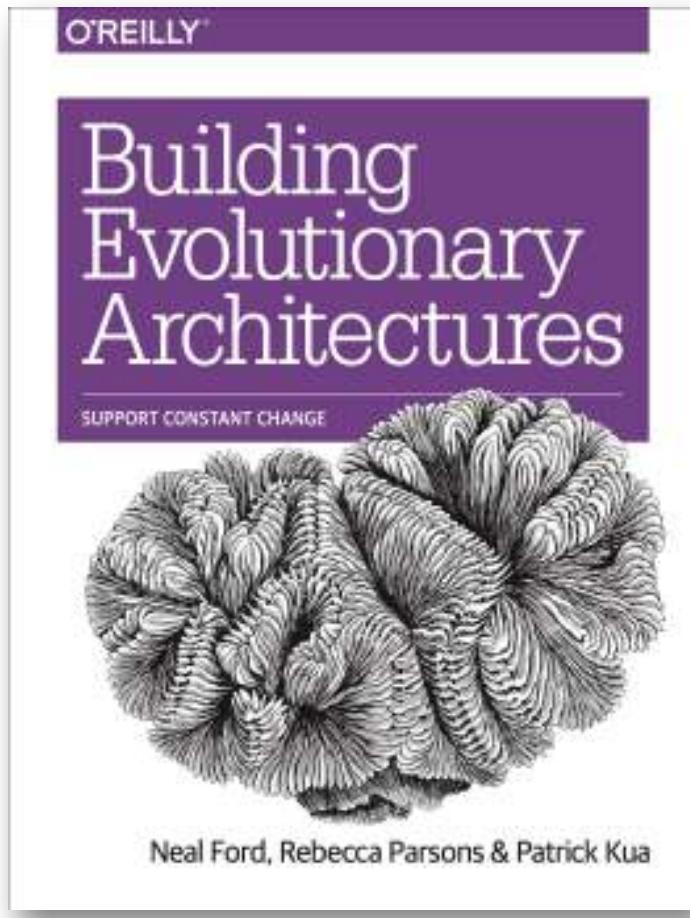
New Capabilities



Untangling the Ball of Mud



Building Evolutionary Architectures



For more information:



<http://evolutionaryarchitecture.com>