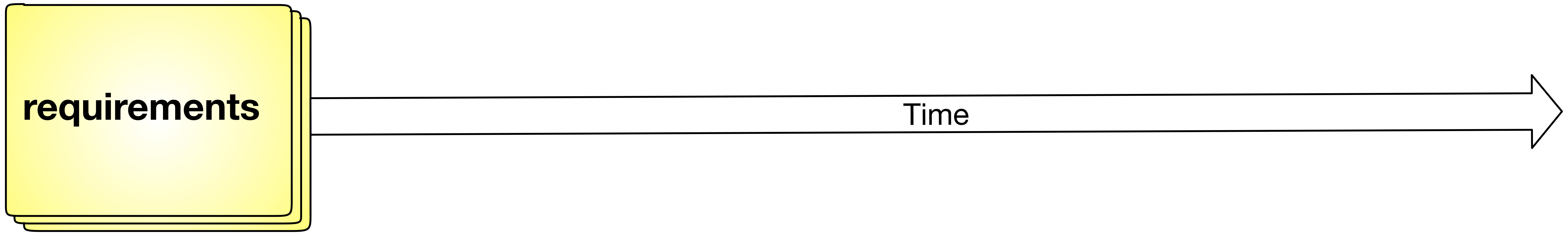
Evolutionary Architecture

Rebecca Parsons
CTO
ThoughtWorks
@rebeccaparsons

in the good old days,
architects could plan.

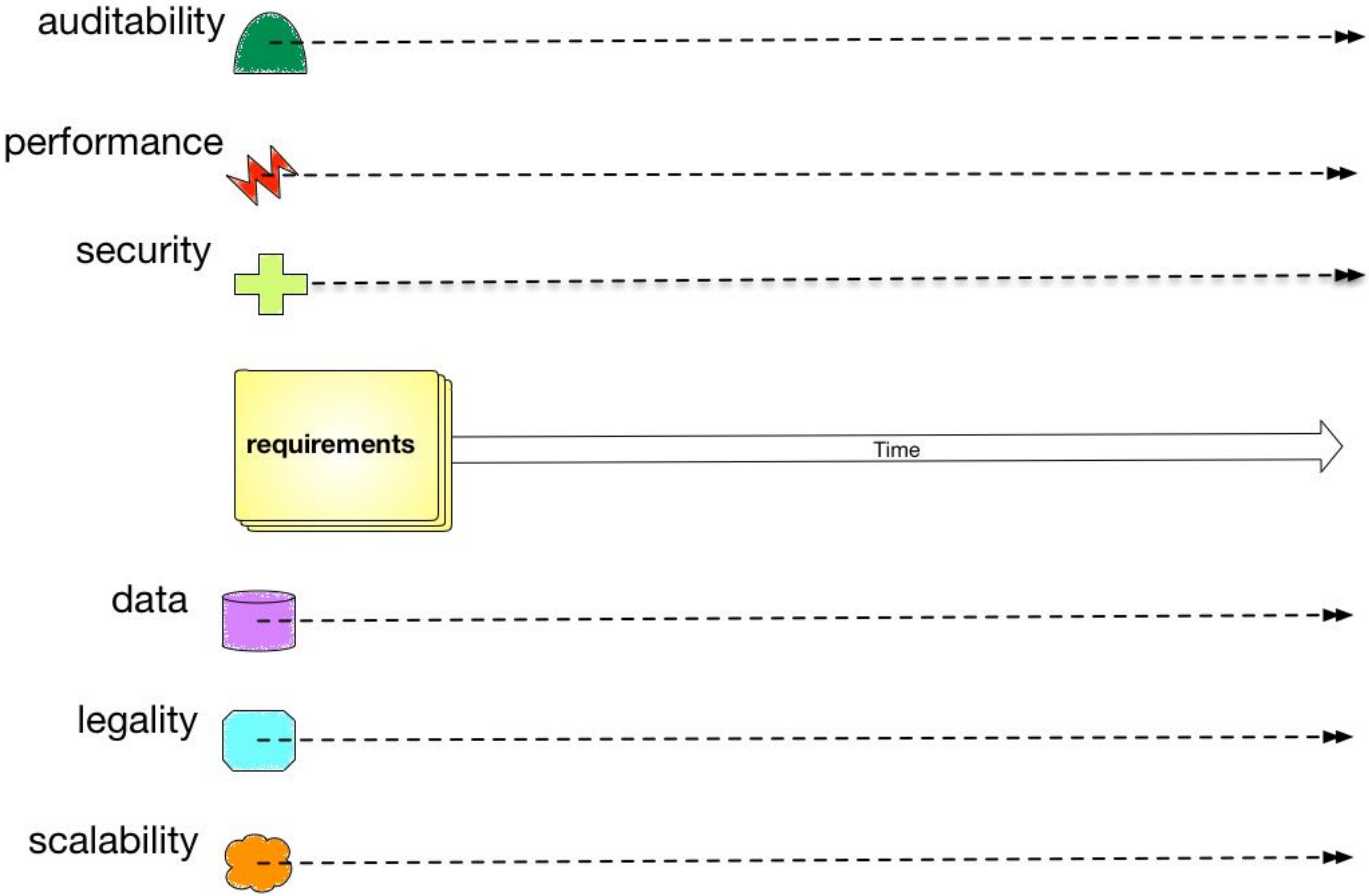
but now, everything changes all the time!

… often in unexpected ways



requirements

Time



auditability

performance

security

requirements

Time

data

legality

scalability

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?

What constitutes good in our context?

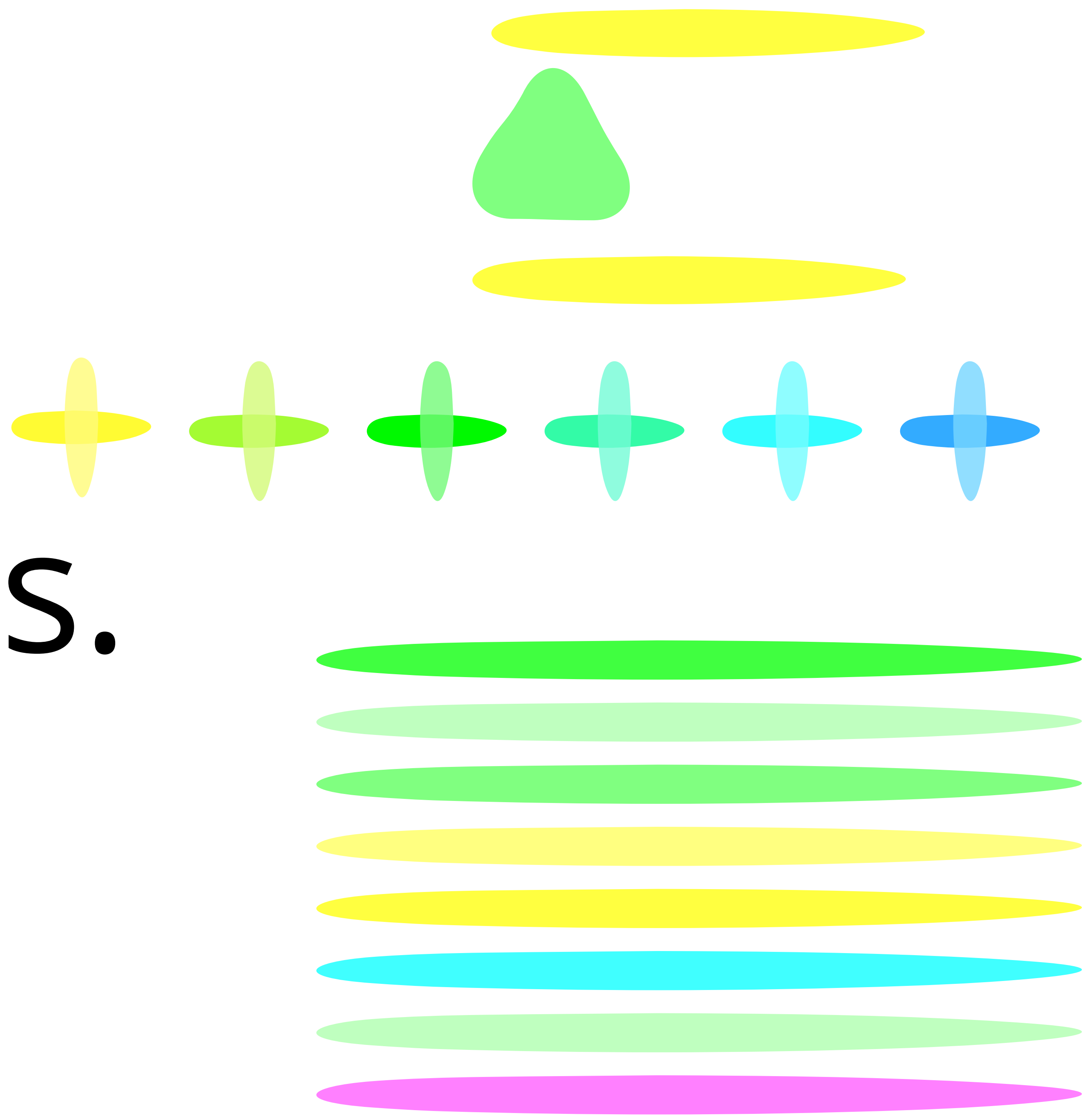
These questions pertain to governance.

Evolutionary Architecture

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

Evolutionary Architecture

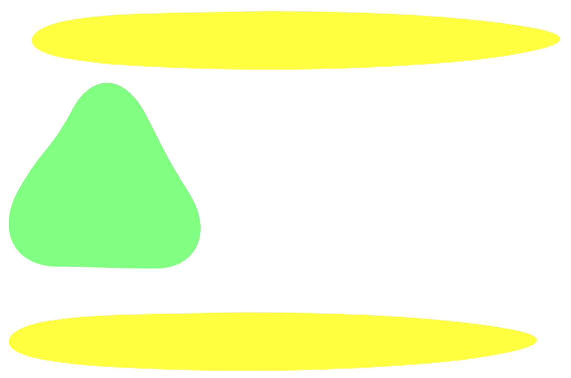
An evolutionary architecture supports guided,
incremental change
across multiple dimensions.



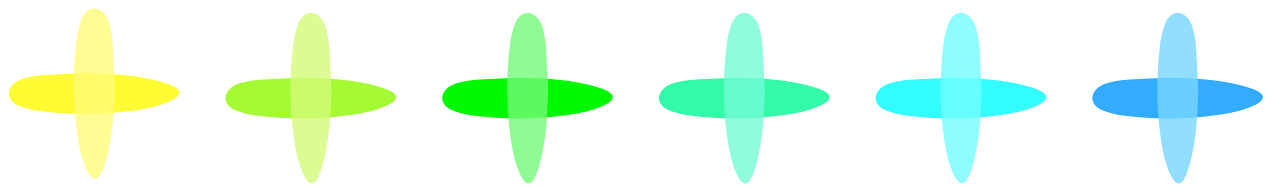
Evolutionary Architecture

An evolutionary architecture supports

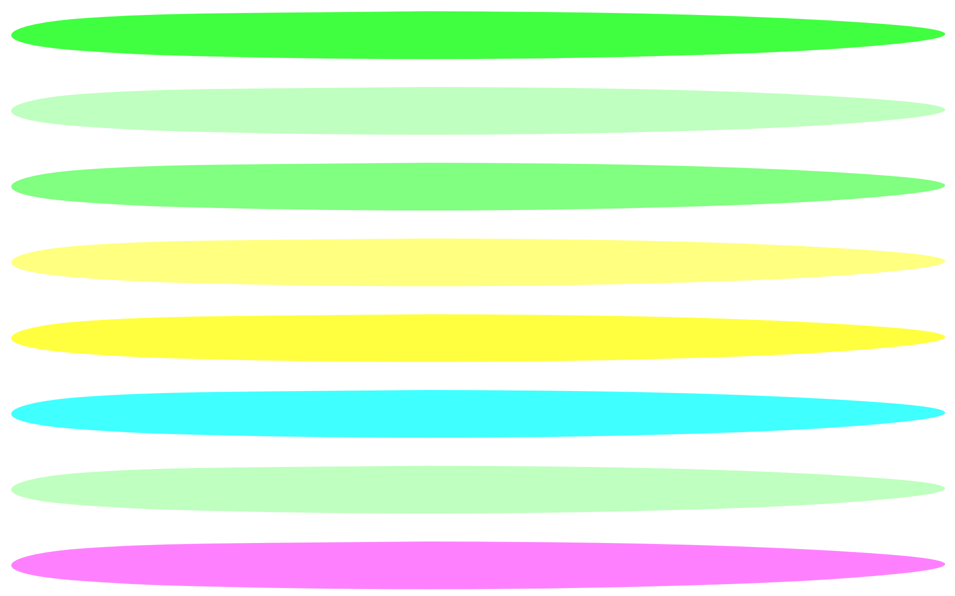
guided



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

architectural *fitness function*:

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

Evolutionary Architecture

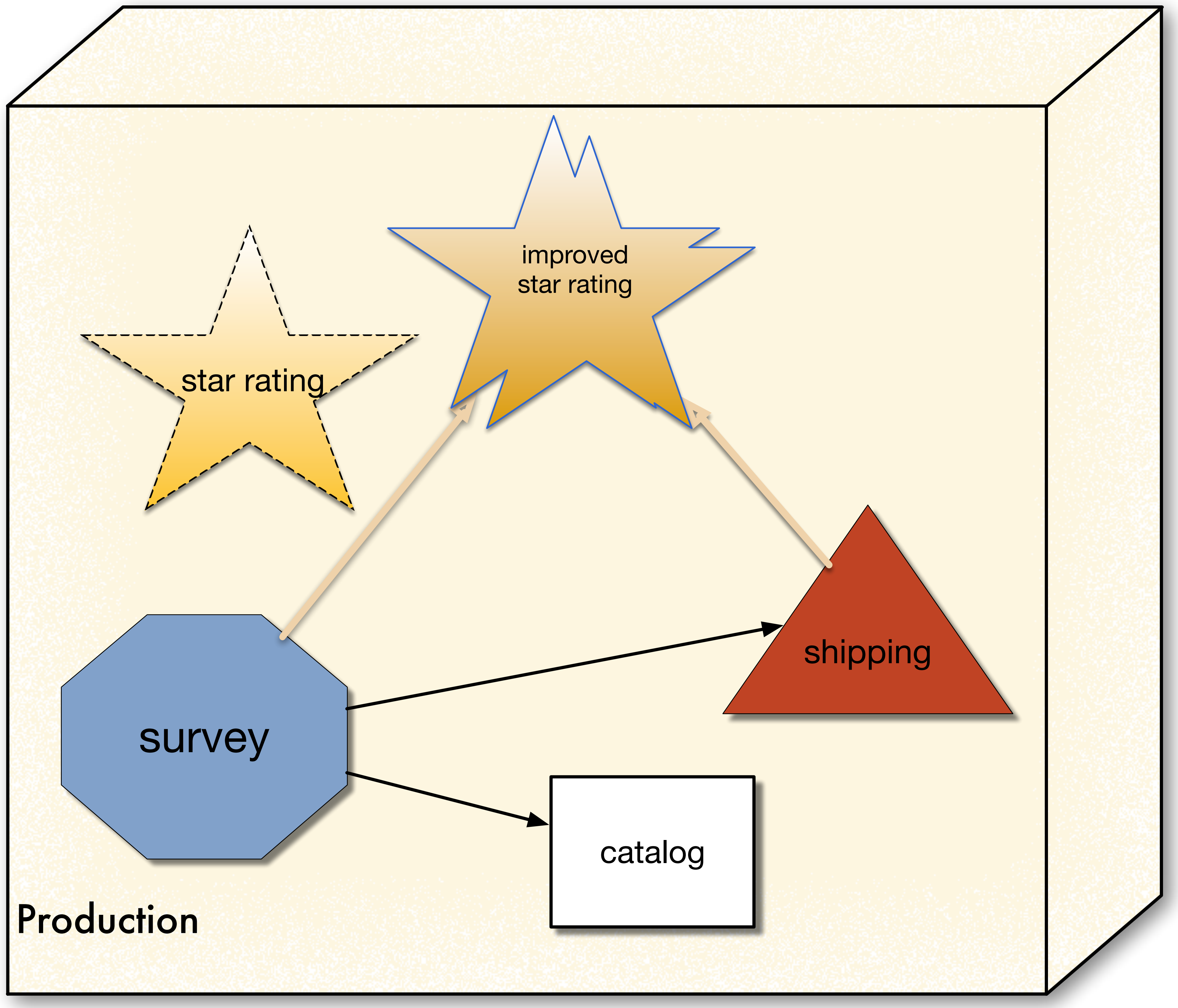
An evolutionary architecture supports

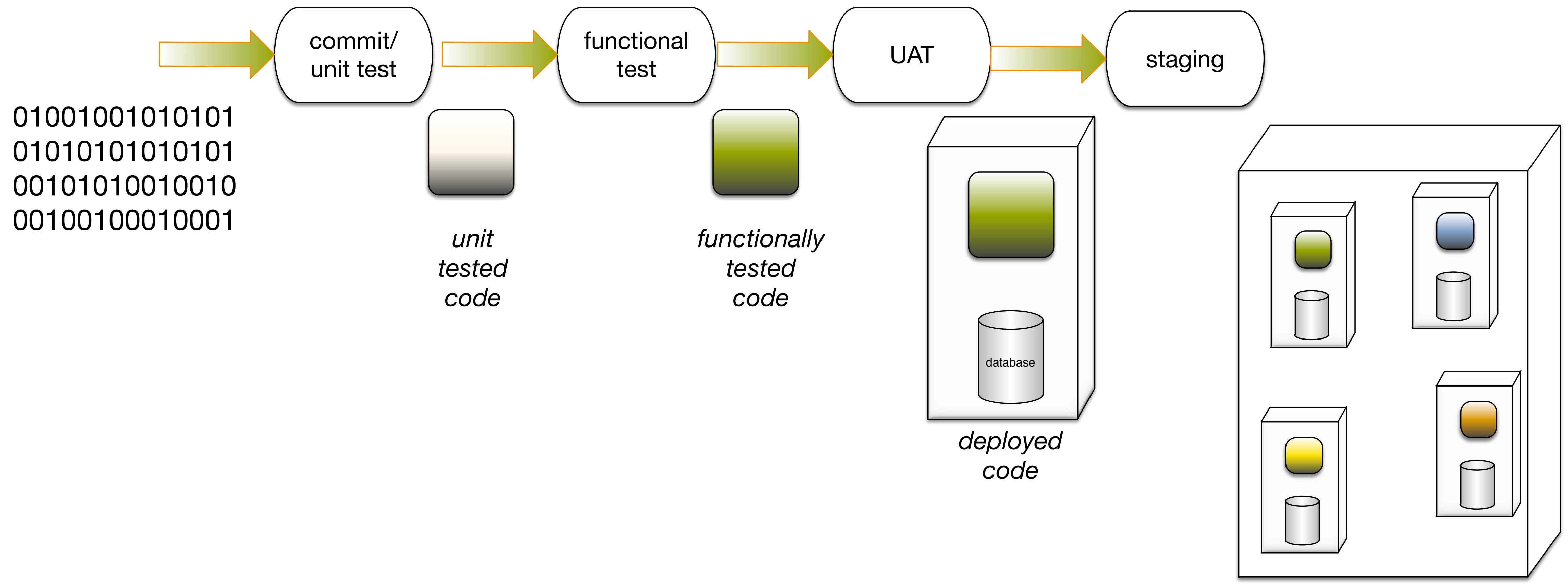
guided

incremental change

across multiple dimensions.

incremental

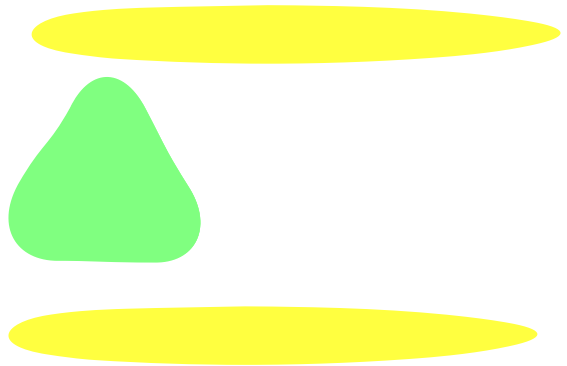




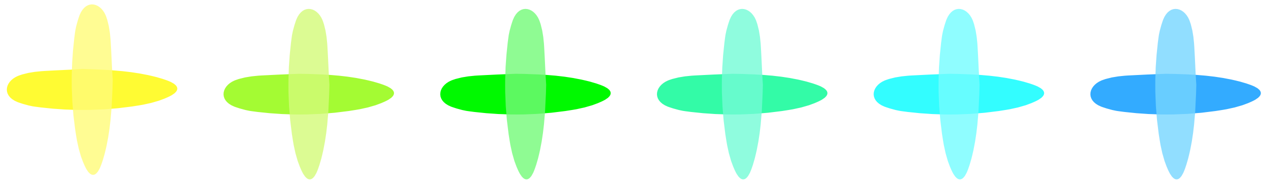
Evolutionary Architecture

An evolutionary architecture supports

guided,

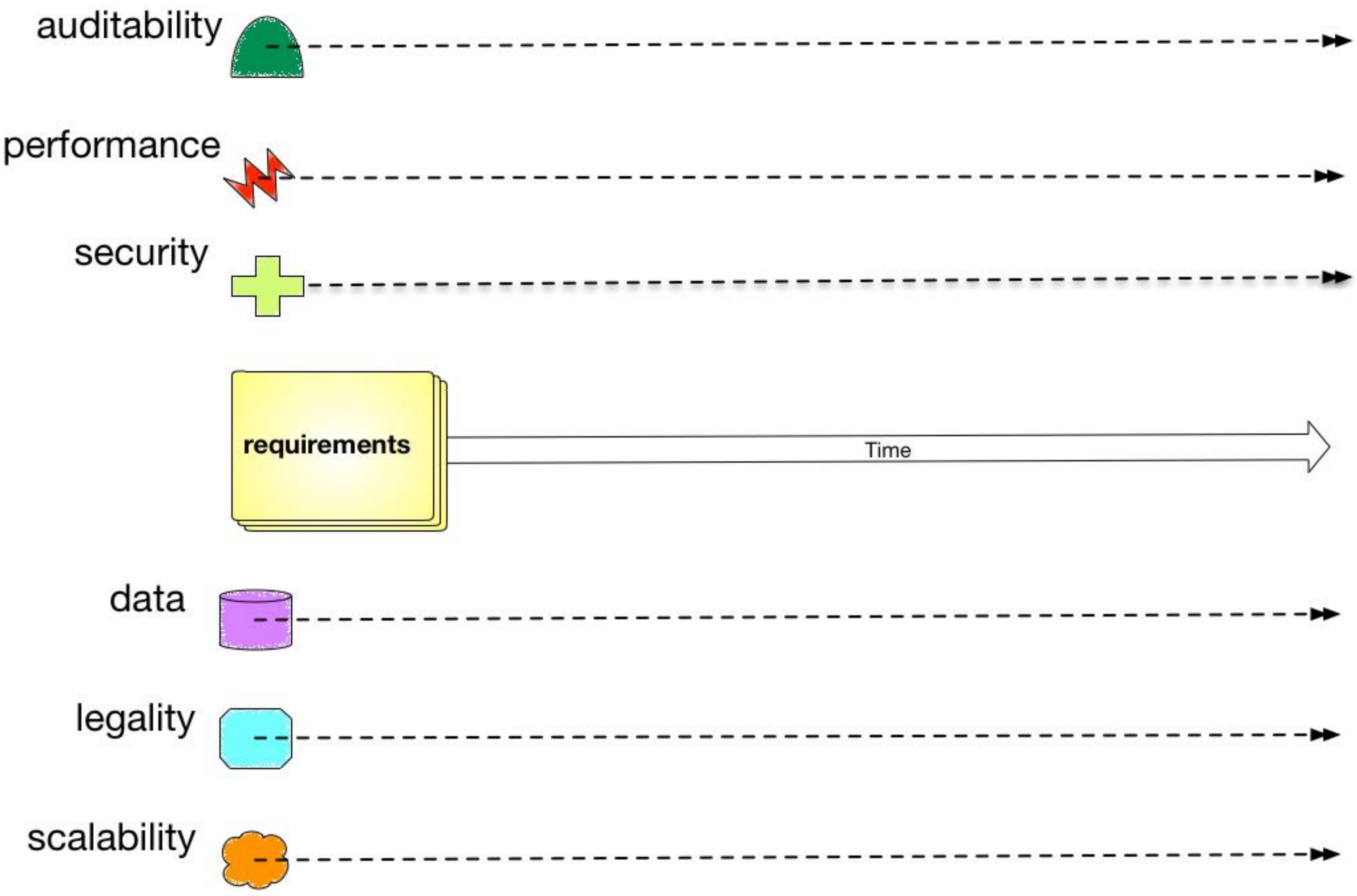


incremental change



across multiple dimensions

multiple dimensions



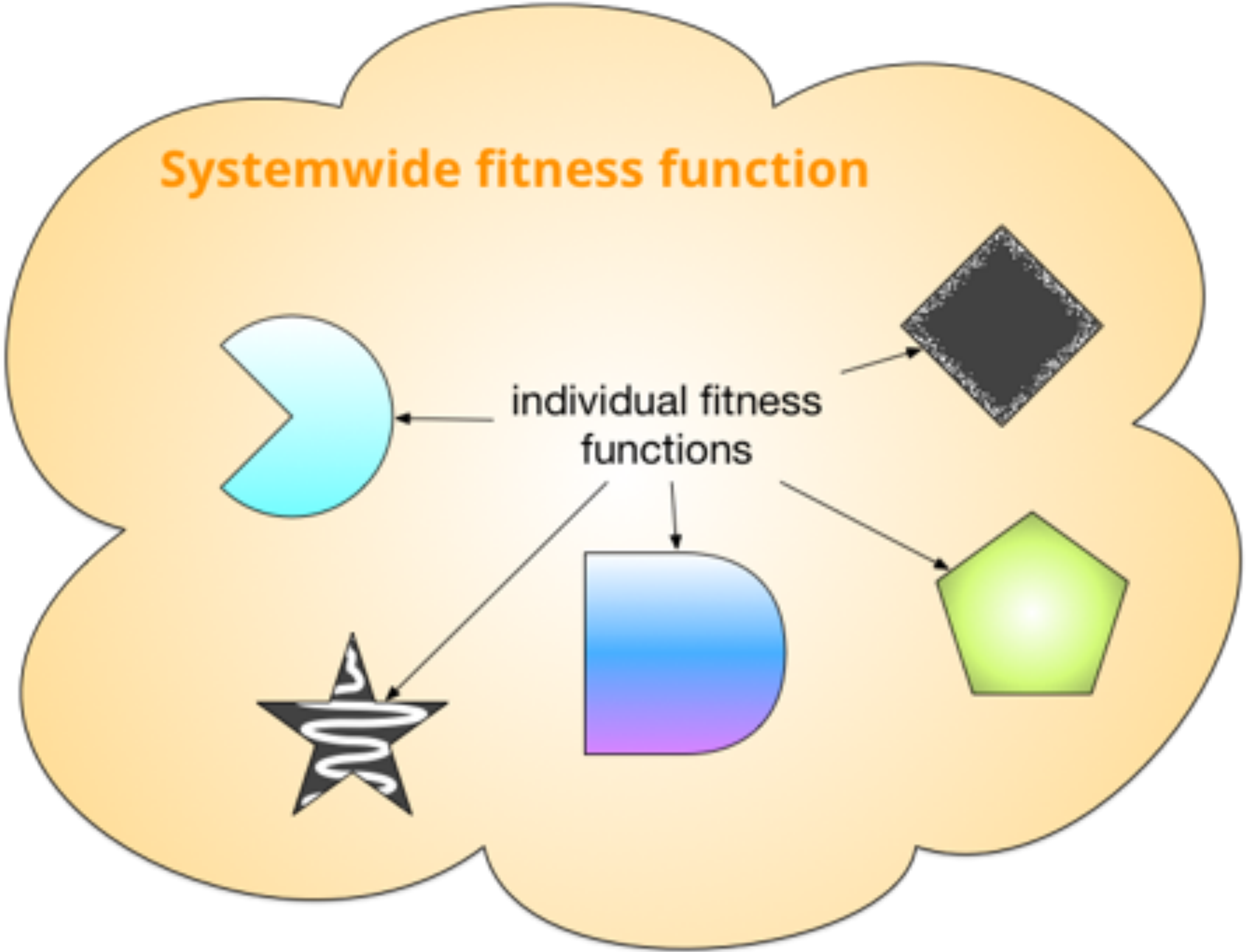
Evolutionary Architecture

An evolutionary architecture supports
guided,
incremental change
across multiple dimensions

So what about governance?

Fitness functions provide the basis for governance.

System-wide Fitness Function

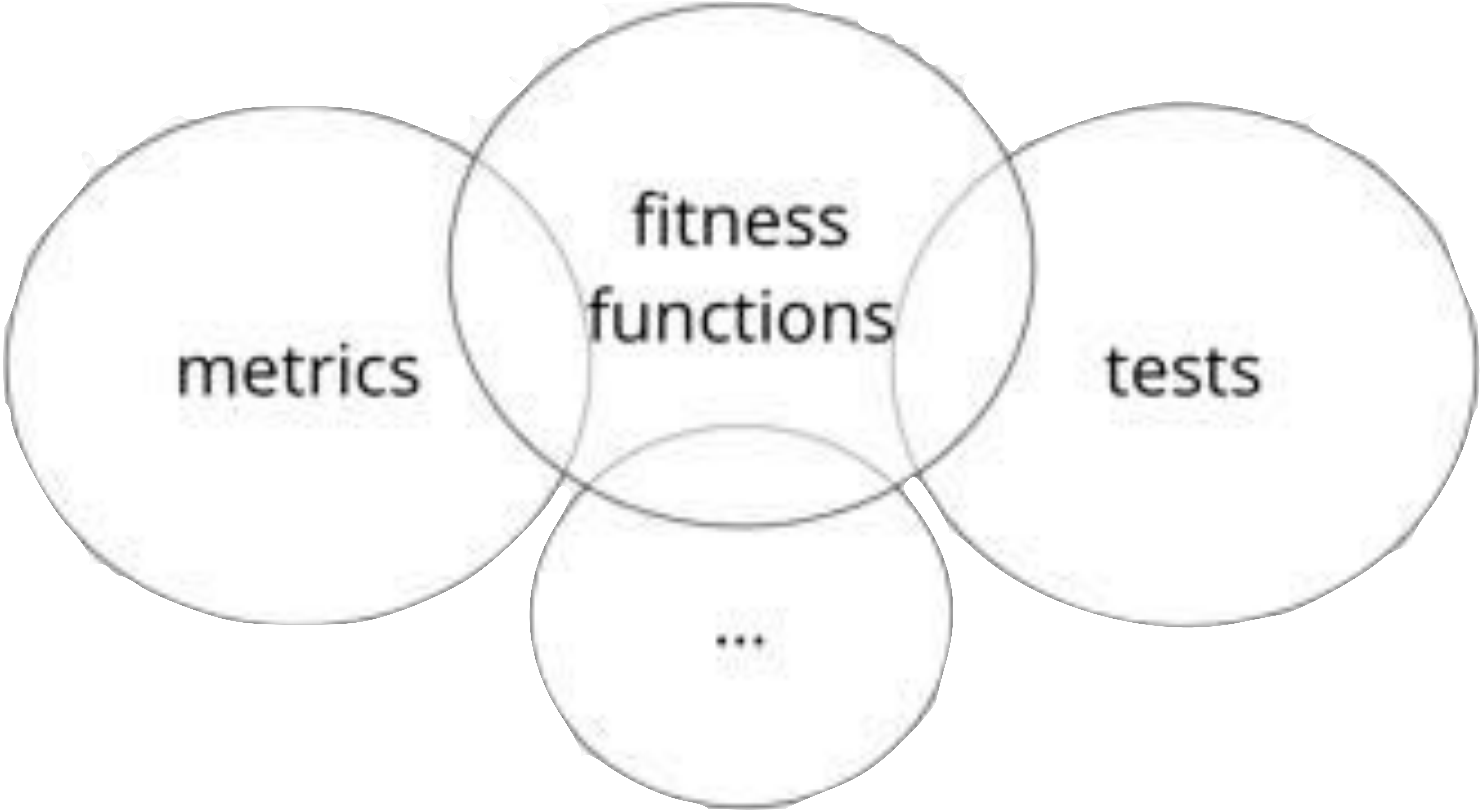


The target architecture is the system-wide fitness function.

Governance relies on these fitness functions.

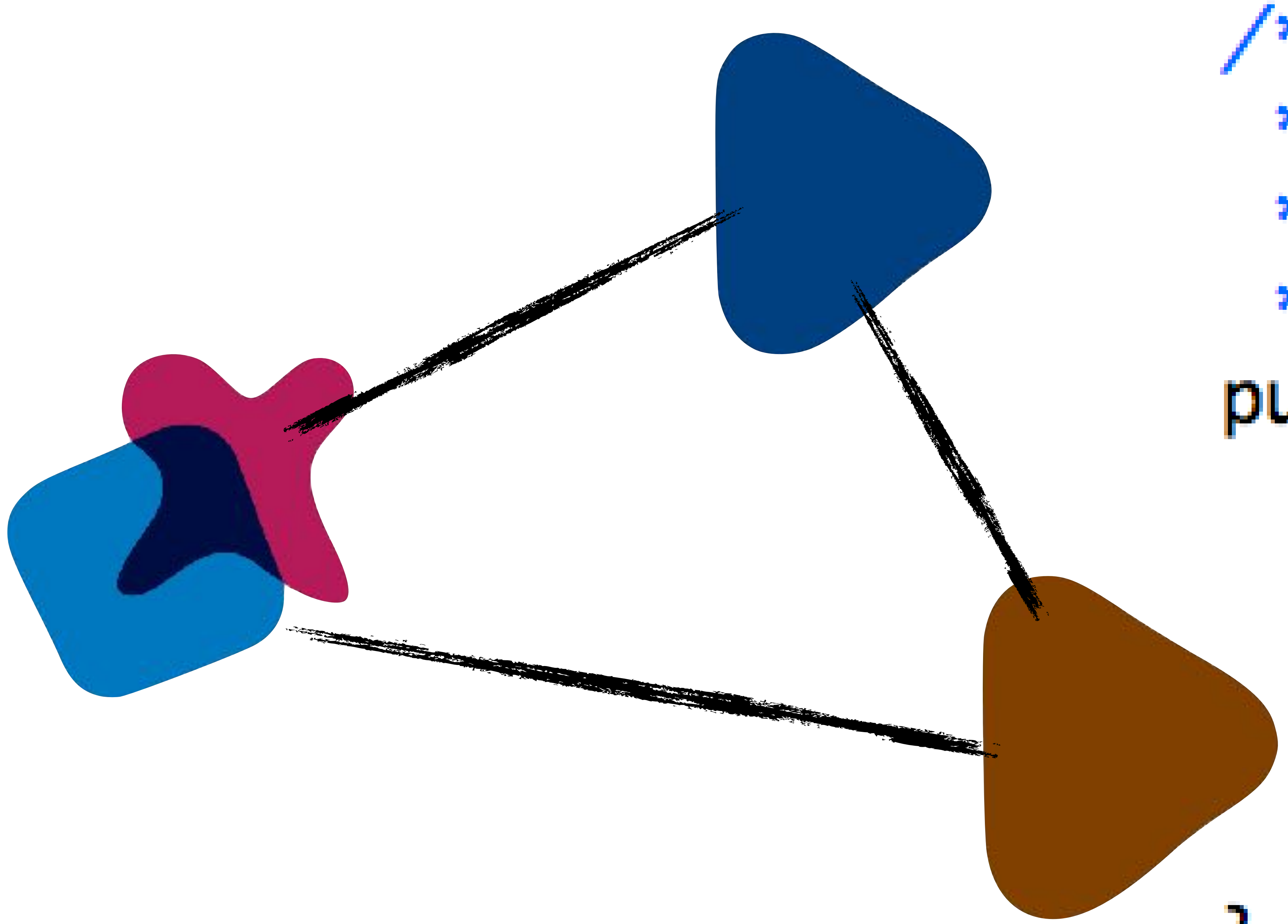
Our target architecture is now defined by outcomes rather than implementations.

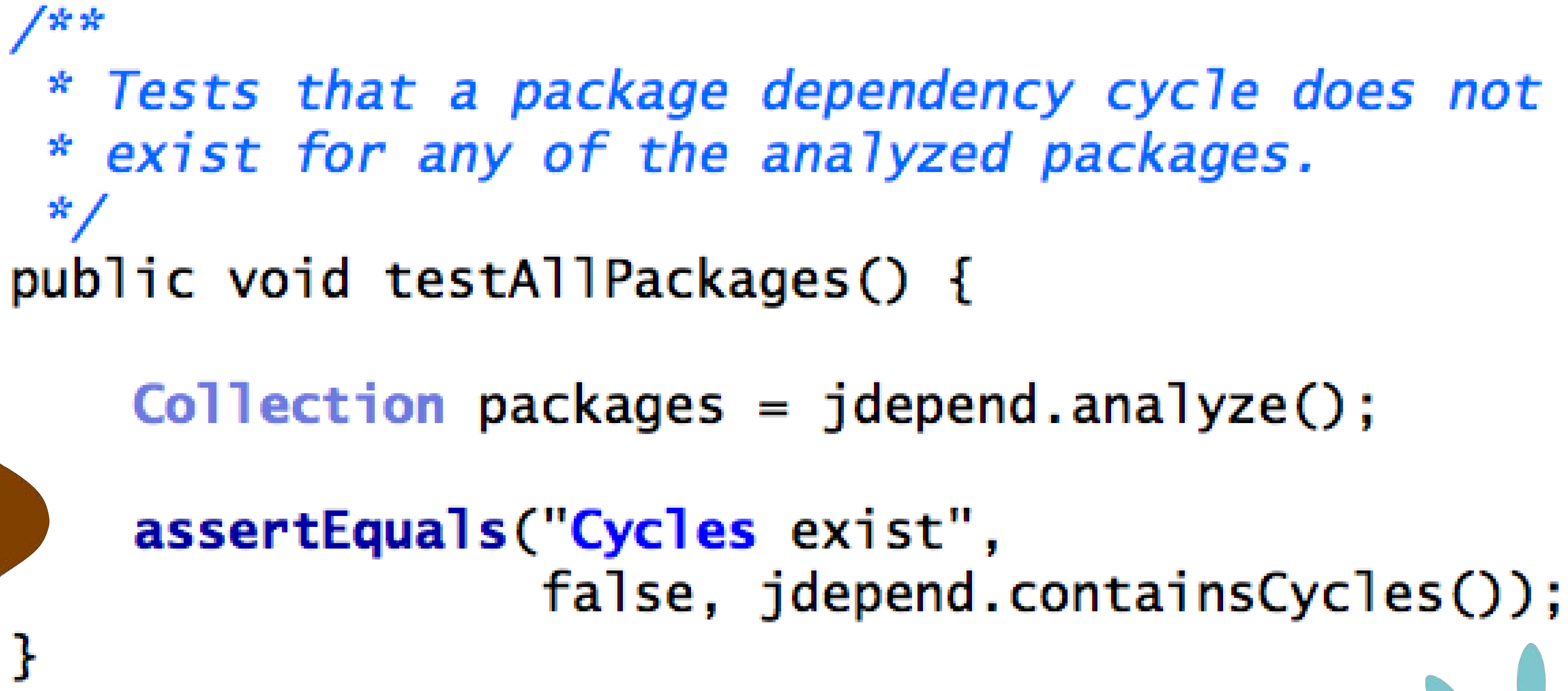
Fitness Functions



What about some examples?

Cyclic Dependency Function



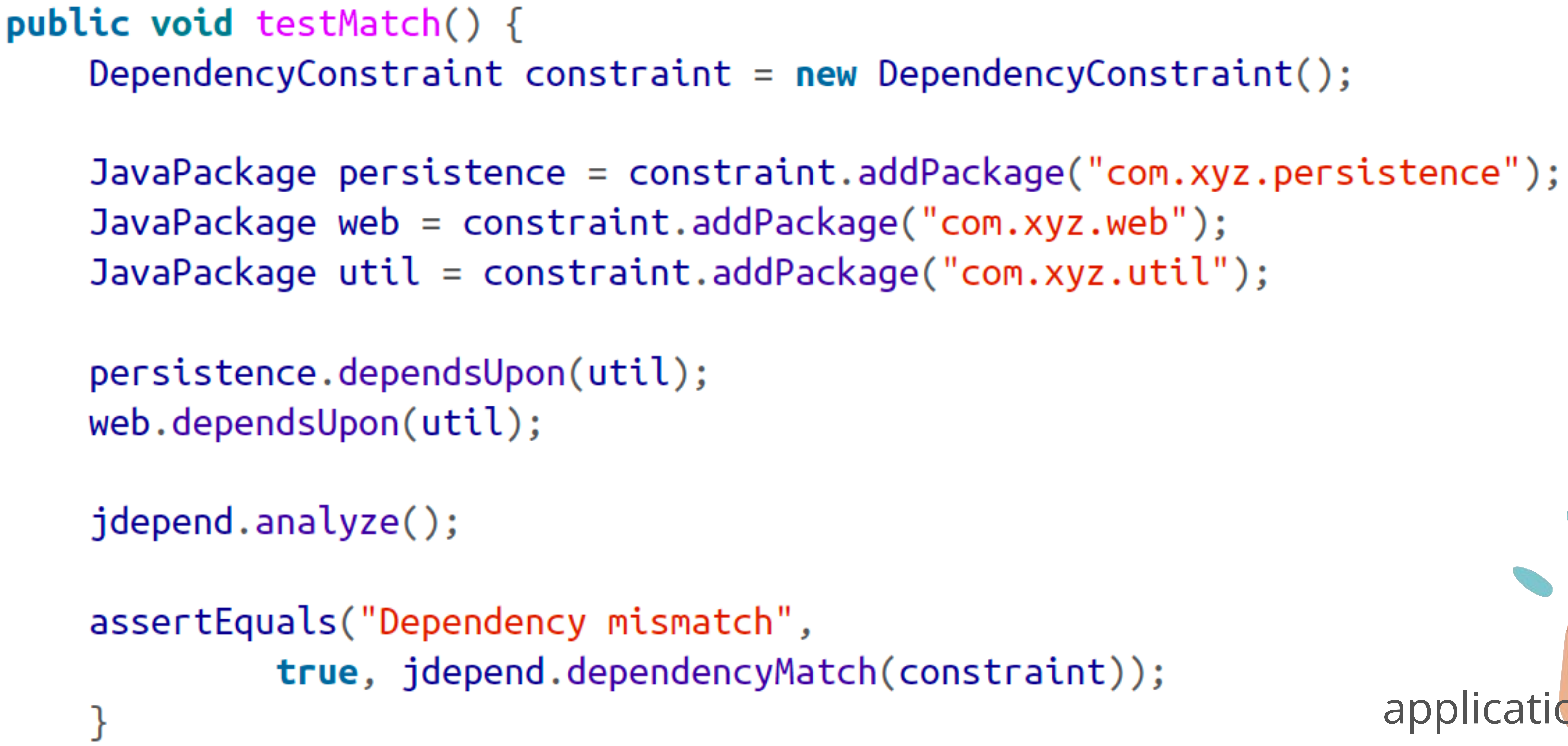


```java
/**
 * Tests that a package dependency cycle does not
 * exist for any of the analyzed packages.
 */
public void testAllPackages() {

    Collection packages = jdepend.analyze();

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

application

clarkware.com/software/JDepend.html

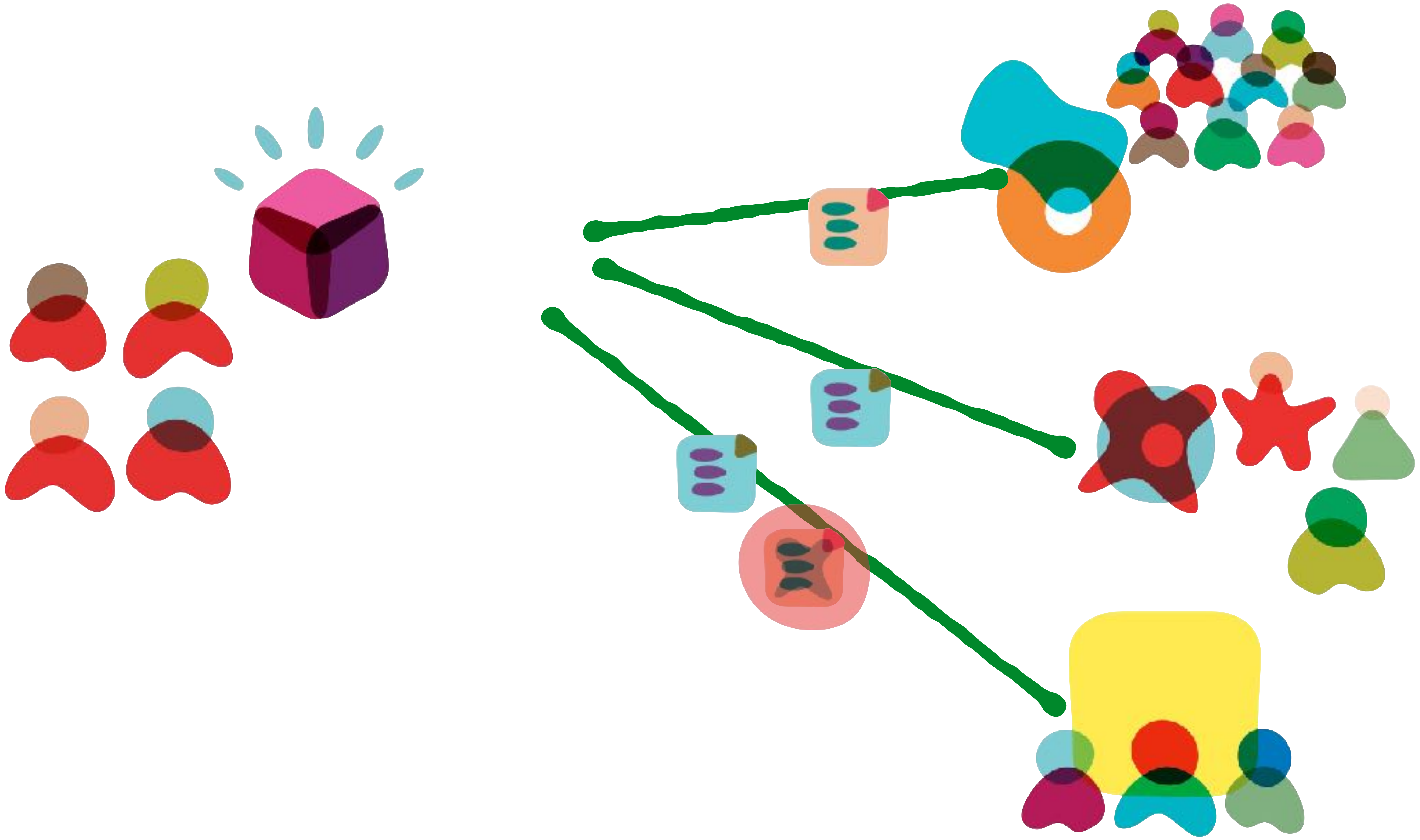
Coupling Fitness Function



```
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));
    }
```

application

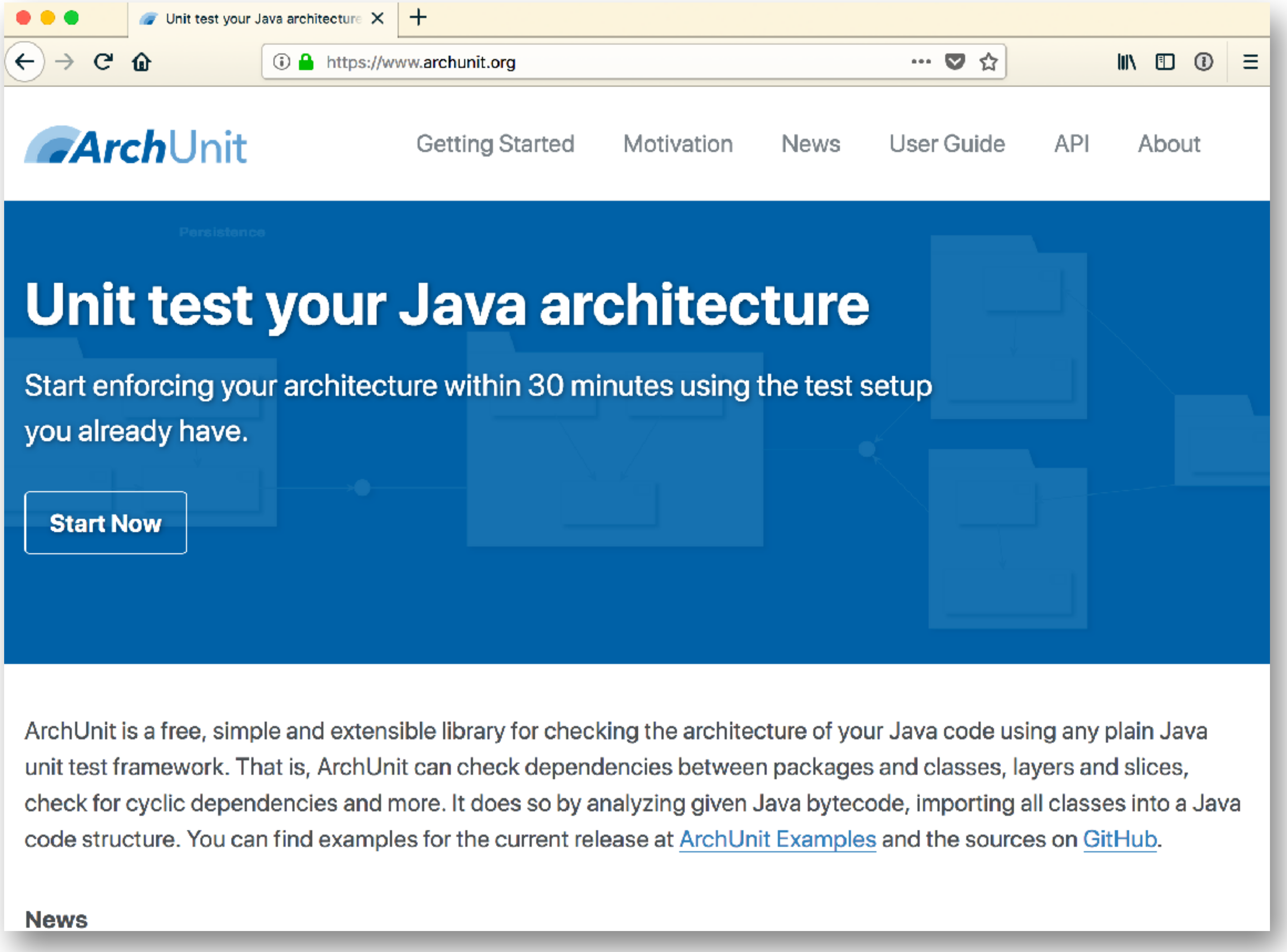
Consumer Driven Contracts



martinfowler.com/articles/consumerDrivenContracts.html

ArchUnit

https://www.archunit.org/



Unit test your Java architecture

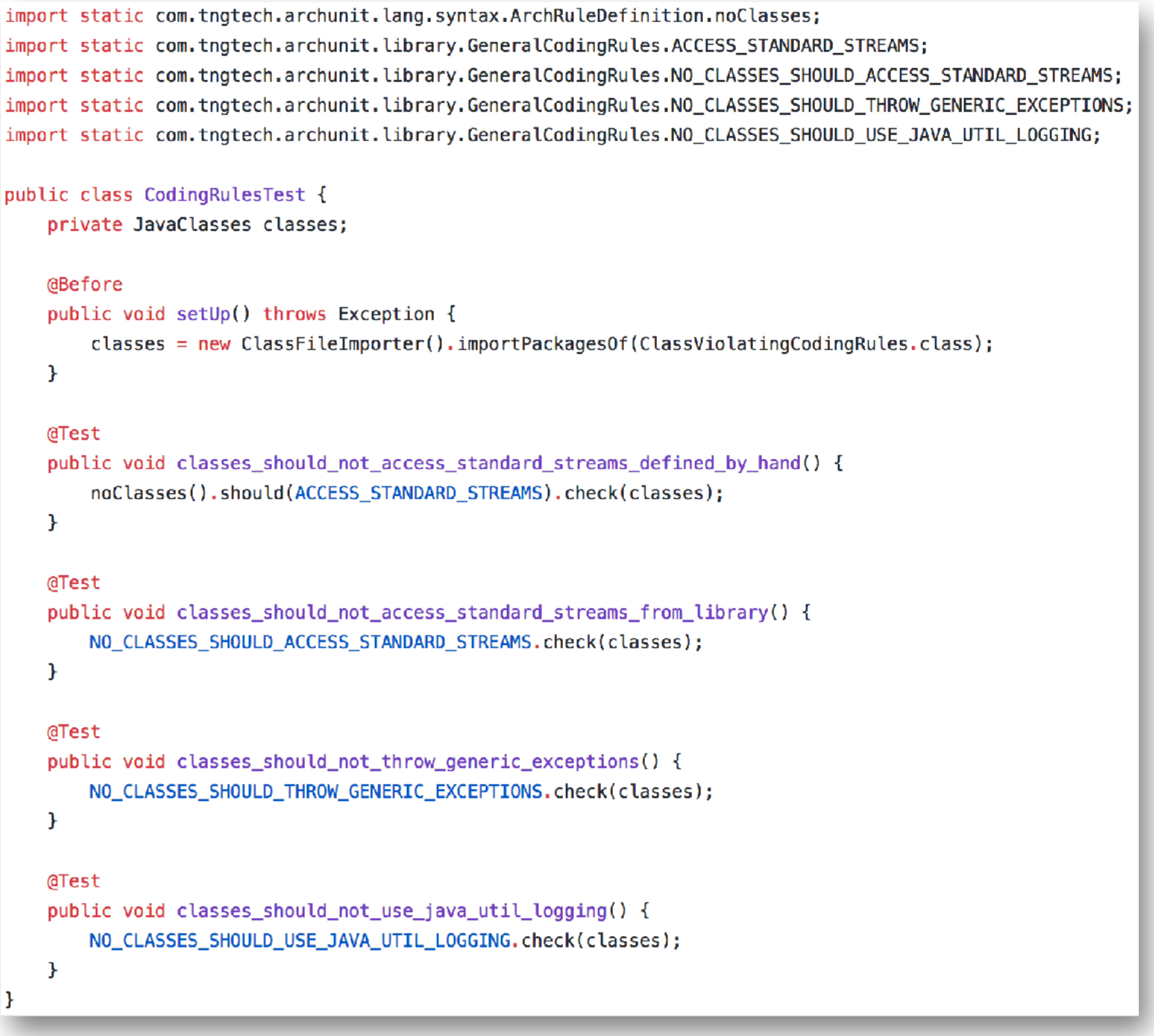
Start enforcing your architecture within 30 minutes using the test setup you already have.

ArchUnit is a free, simple and extensible library for checking the architecture of your Java code using any plain Java unit test framework. That is, ArchUnit can check dependencies between packages and classes, layers and slices, check for cyclic dependencies and more. It does so by analyzing given Java bytecode, importing all classes into a Java code structure. You can find examples for the current release at ArchUnit Examples and the sources on GitHub.

News

ArchUnit

https://www.archunit.org/

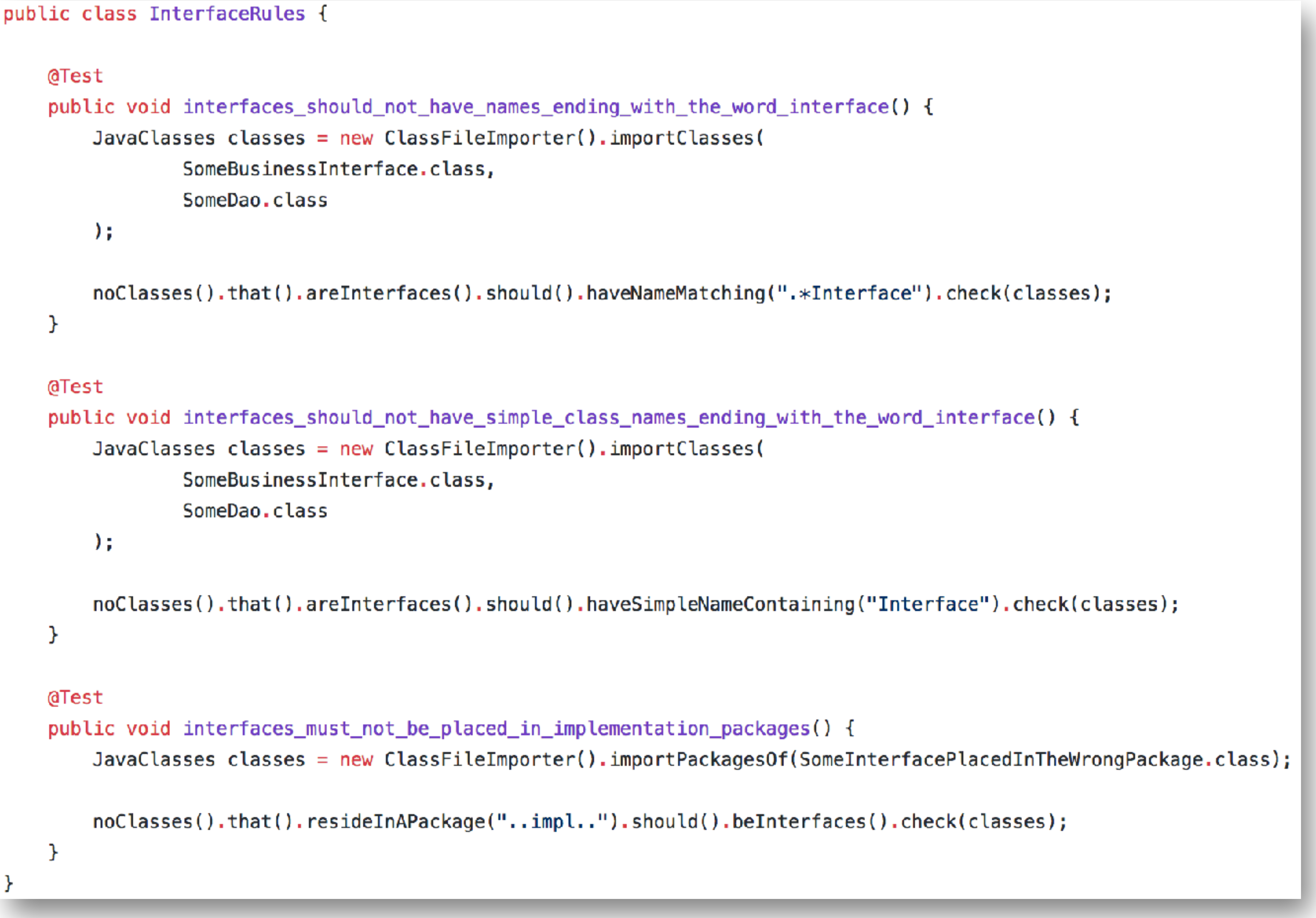


```
import static com.tngtech.archunit.lang.syntax.ArchRuleDefinition.noClasses;
import static com.tngtech.archunit.library.GeneralCodingRules.ACCESS_STANDARD_STREAMS;
import static com.tngtech.archunit.library.GeneralCodingRules.NO_CLASSES_SHOULD_ACCESS_STANDARD_STREAMS;
import static com.tngtech.archunit.library.GeneralCodingRules.NO_CLASSES_SHOULD_THROW_GENER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

public class CodingRulesTest {
    private JavaClasses classes;

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

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

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

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

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

coding rules

ArchUnit

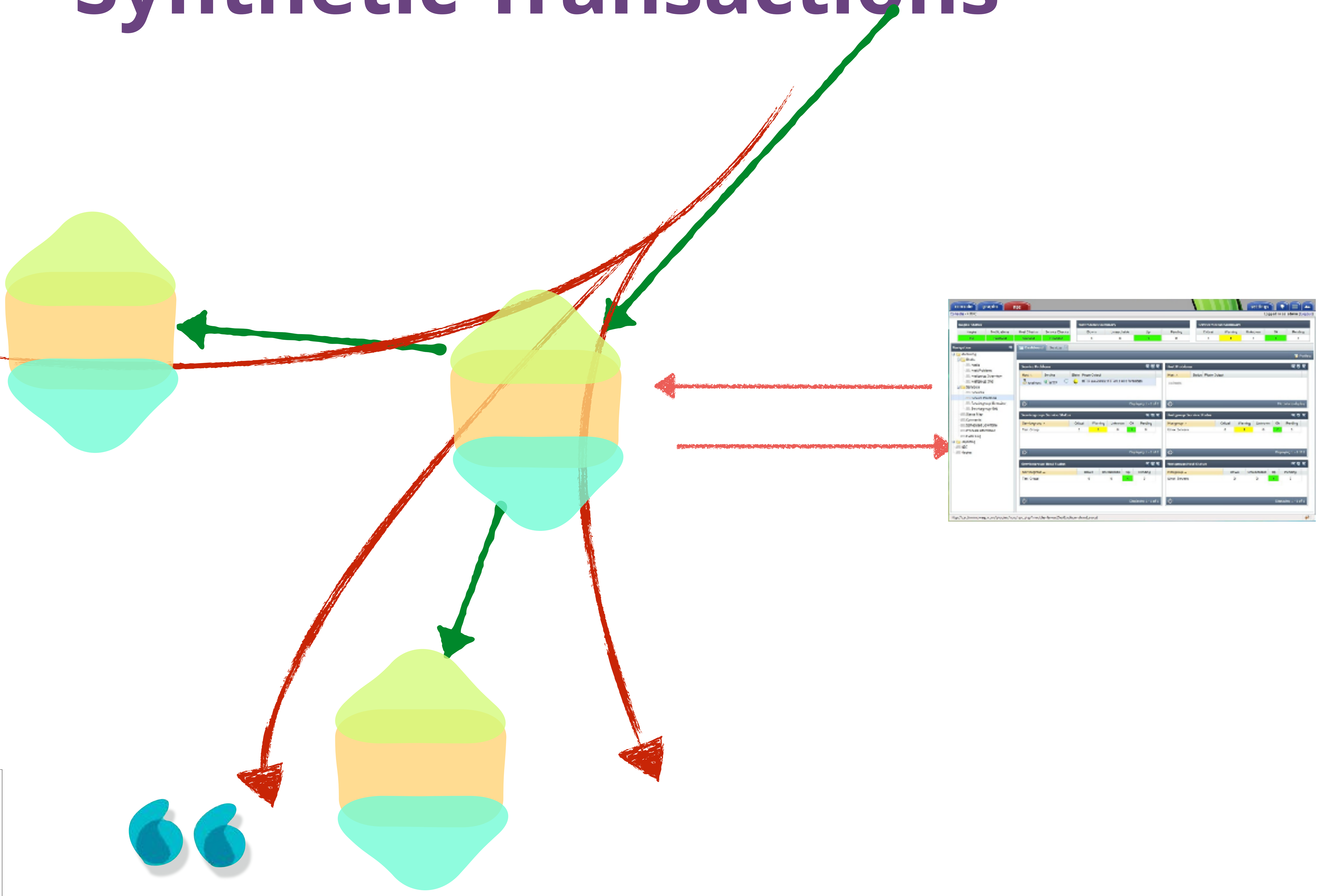
https://www.archunit.org/



```
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

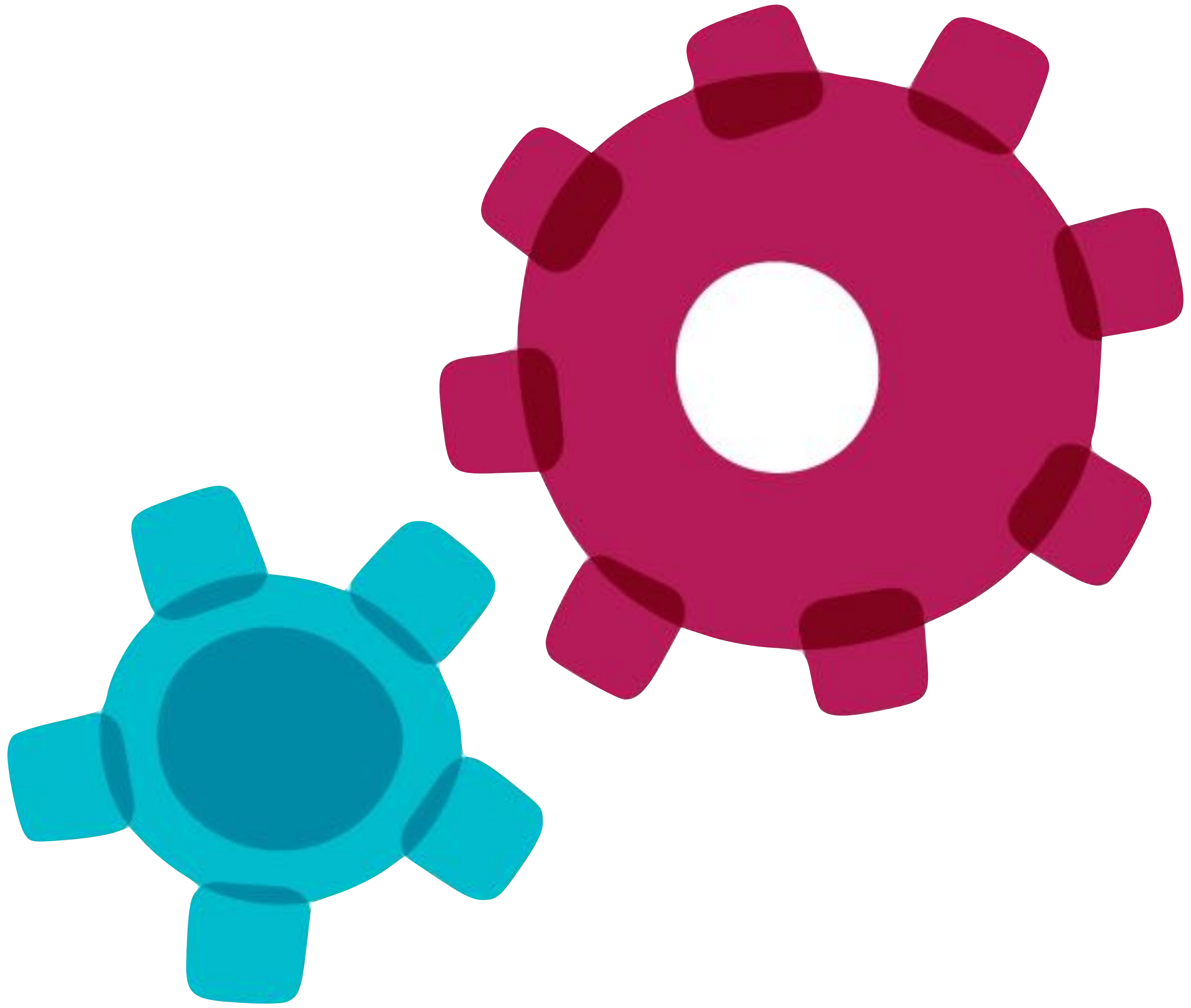
Synthetic Transactions



> *Use synthetic transactions to test production systems.*

amazon
web services
CHAOS
MONKEY

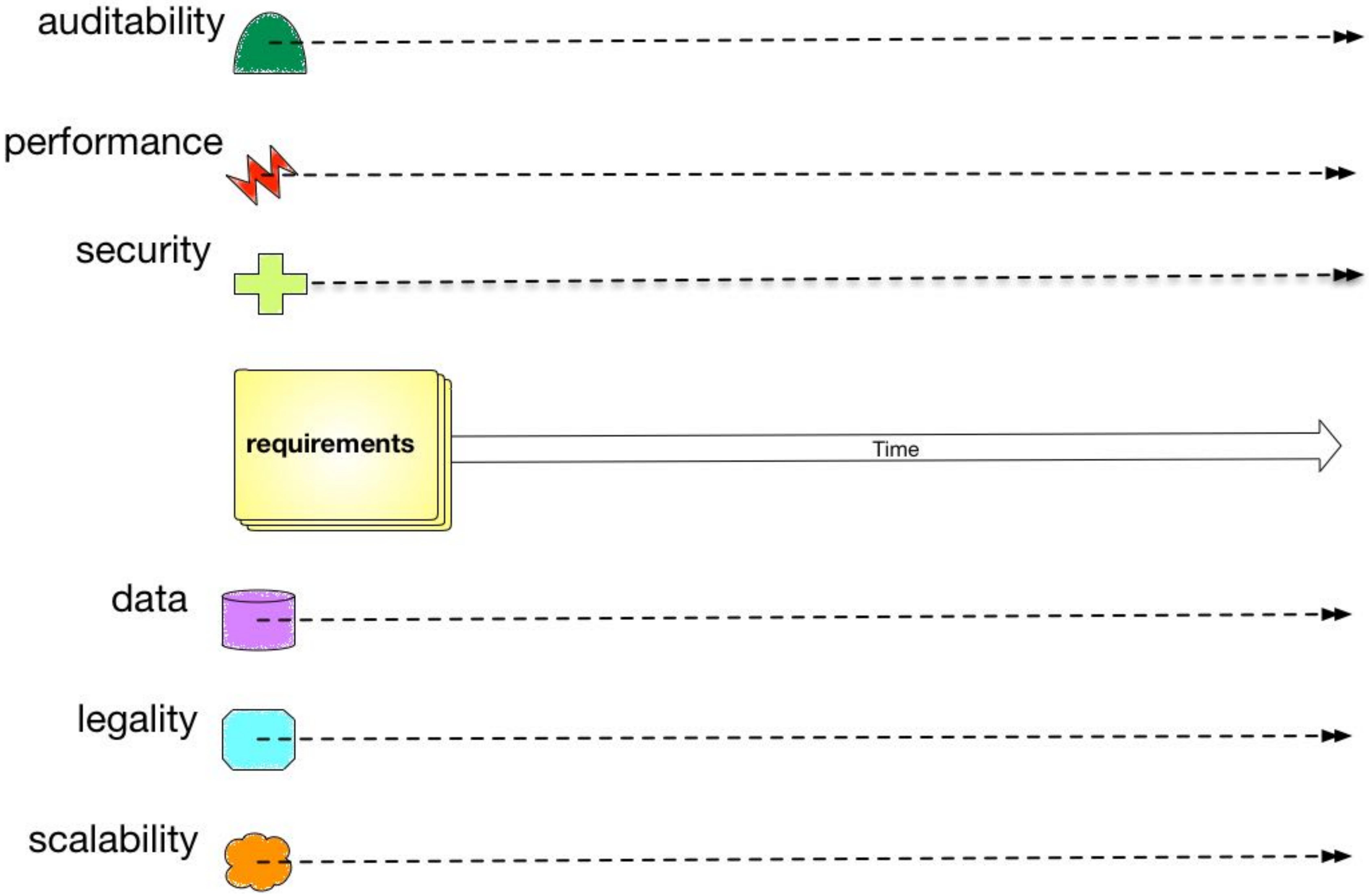
Putting evolutionary architecture into Practice



Mechanics

1. Identify dimensions affect by evolution

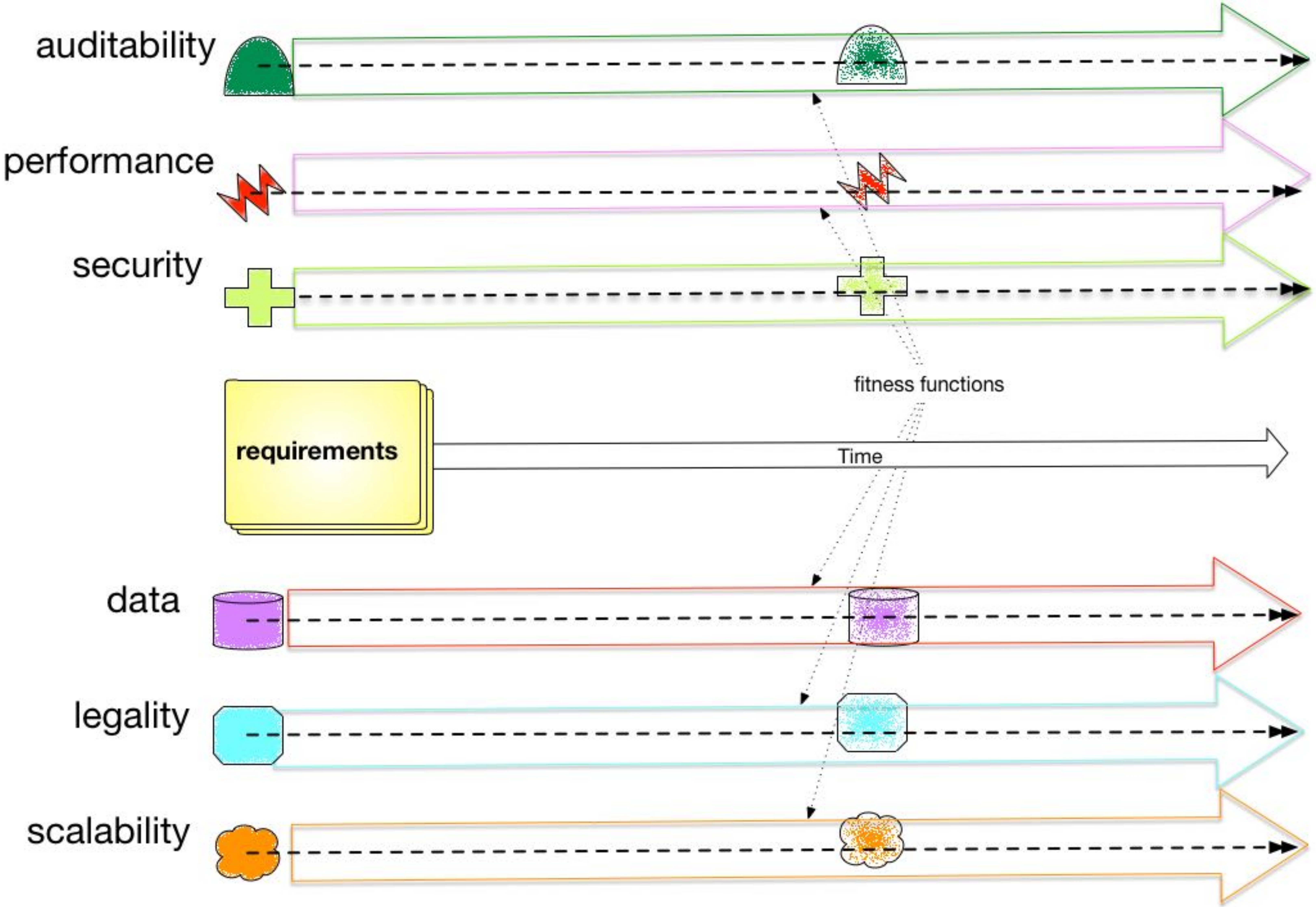
1. Identify dimensions affect by evolution



Mechanics

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

2. Define Fitness Function(s) for Each Dimension



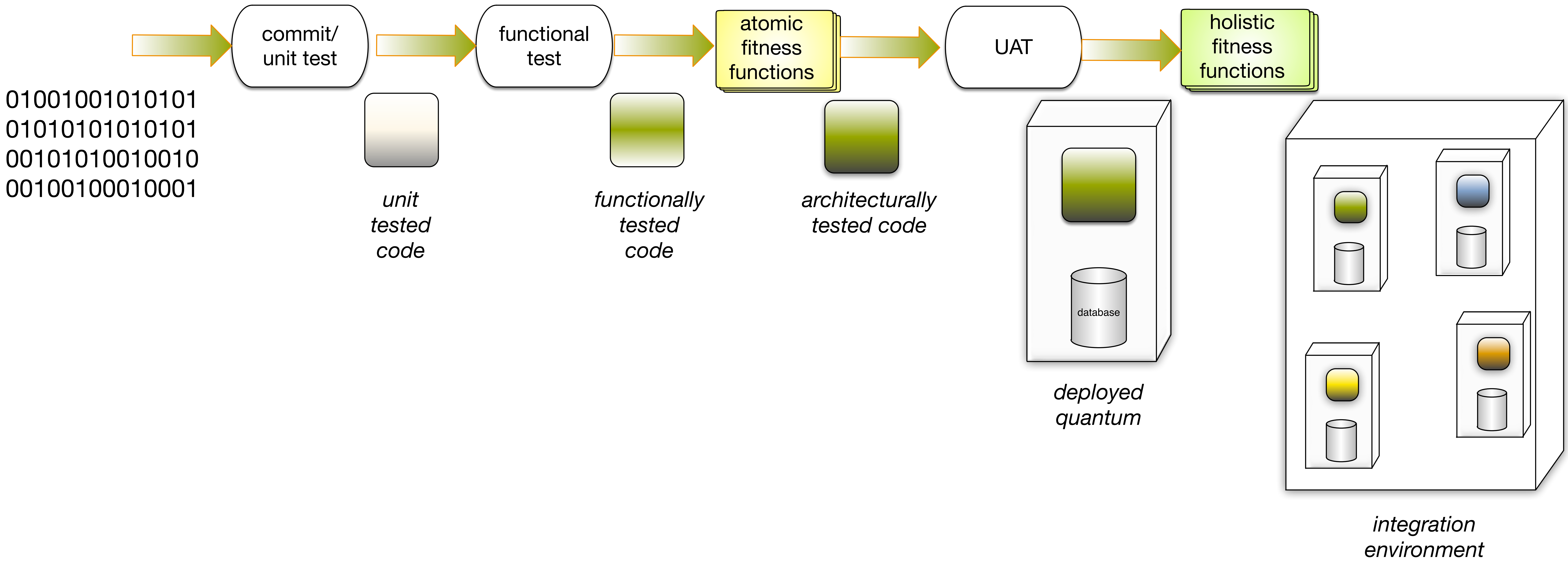
Mechanics

1. Identify dimensions affect 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



01001001010101
01010101010101
00101010010010
00100100010001

commit/
unit test

functional
test

atomic
fitness
functions

UAT

holistic
fitness
functions

*unit
tested
code*

*functionally
tested
code*

*architecturally
tested code*

database

*deployed
quantum*

*integration
environment*

Mechanics

1. Identify dimensions affect by evolution

2. Define Fitness Function(s) for Each Dimension

3. Use Deployment Pipelines to Automate Fitness Functions

Mechanics

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

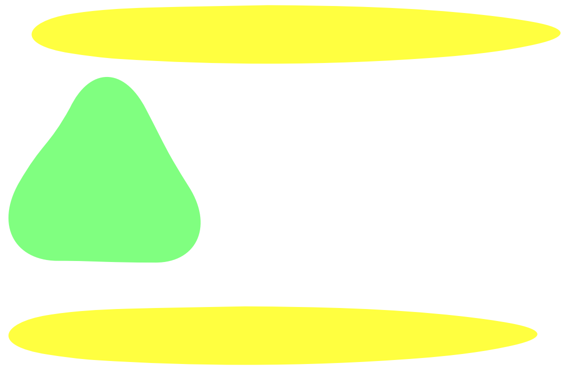
Techniques

1. Database refactoring
2. Choreography
3. Continuous delivery

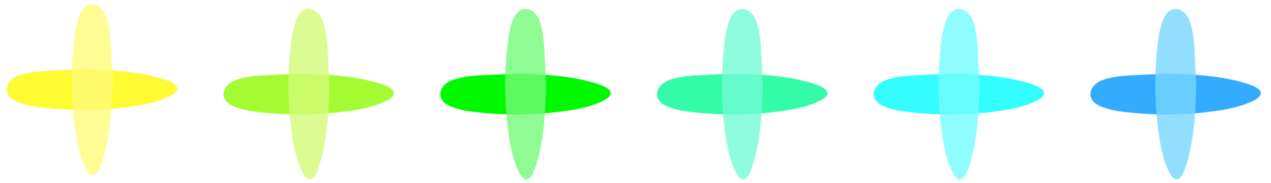
Evolutionary Architecture

An evolutionary architecture supports

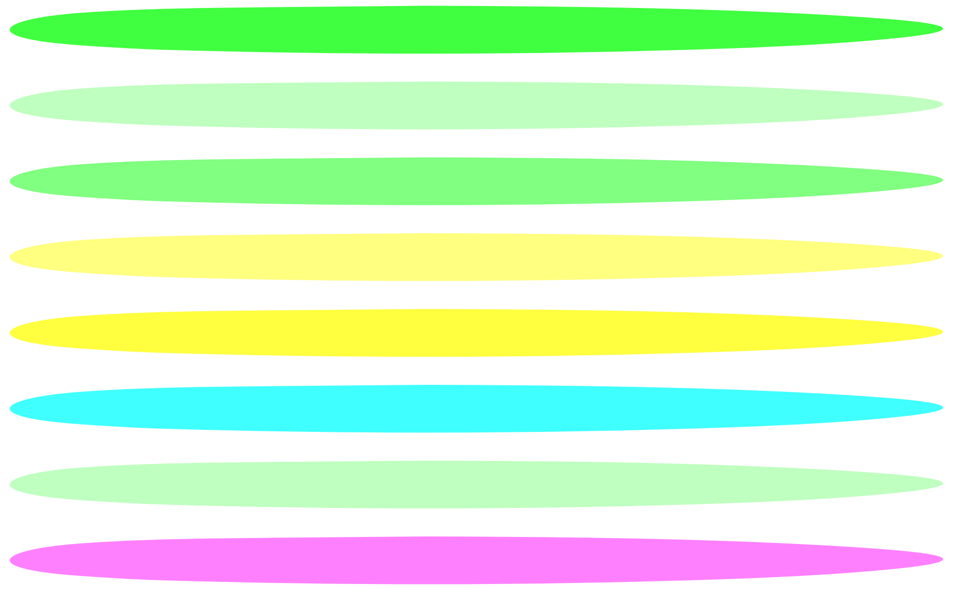
guided,



incremental change

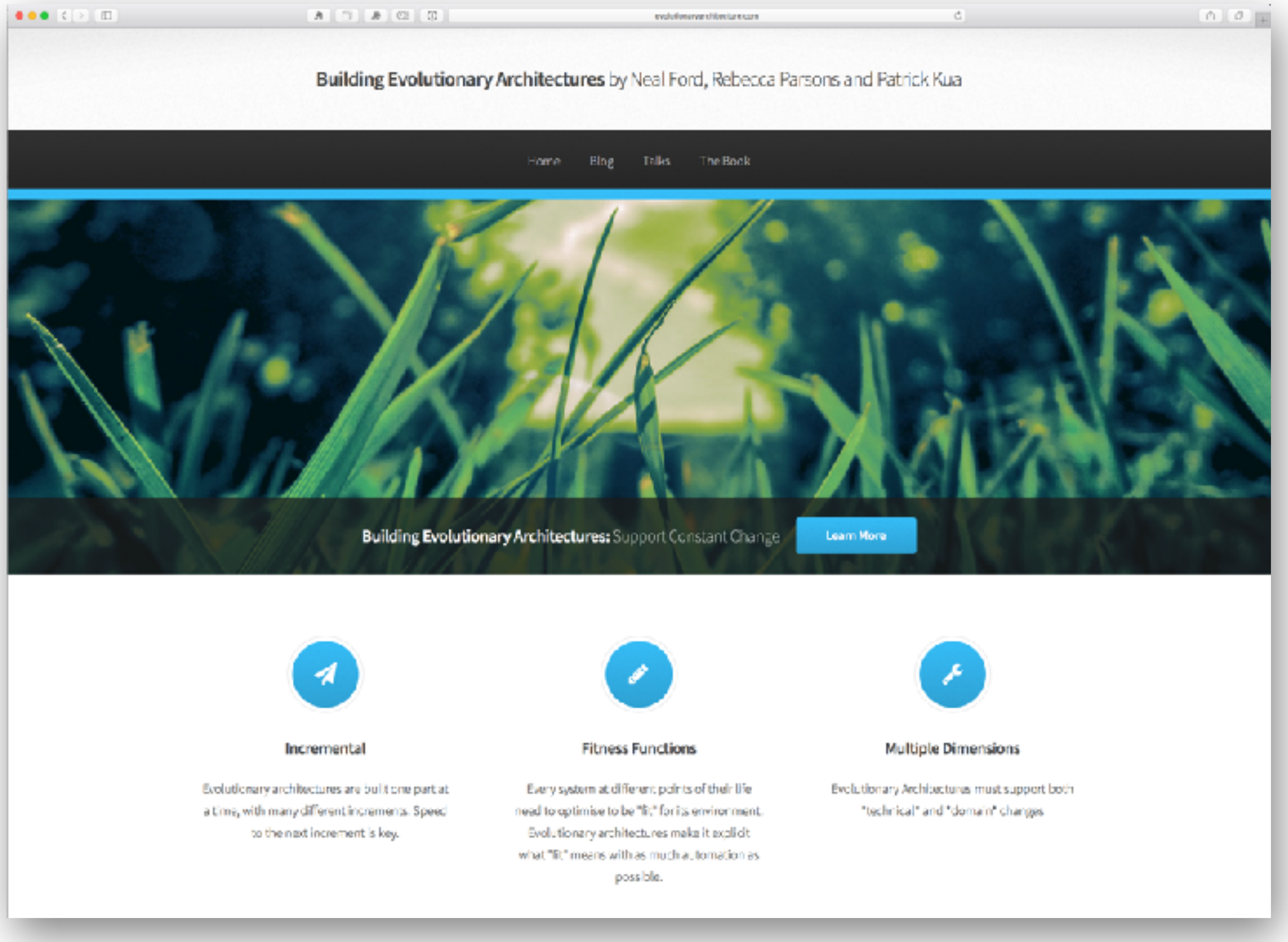


across multiple dimensions.



Building Evolutionary Architectures

For more information:



Building Evolutionary Architectures by Neal Ford, Rebecca Parsons and Patrick Kua

Building Evolutionary Architectures: Support Constant Change

Incremental

Evolutionary architectures are built one part at a time, with many different increments. Speed to the next increment is key.

Fitness Functions

Every system at different points of their life need to optimise to be "fit" for its environment. Evolutionary architectures make it explicit what "fit" means with as much automation as possible.

Multiple Dimensions

Evolutionary Architectures must support both "technical" and "domain" changes

http://evolutionaryarchitecture.com

Thank you!

http://evolutionaryarchitecture.com

@rebeccaparsons