Architecture for Modern Engineering Practices

Neal Ford

director / software architect / meme wrangler

/thoughtworks


@neal4d

nealford.com

2023-04-17

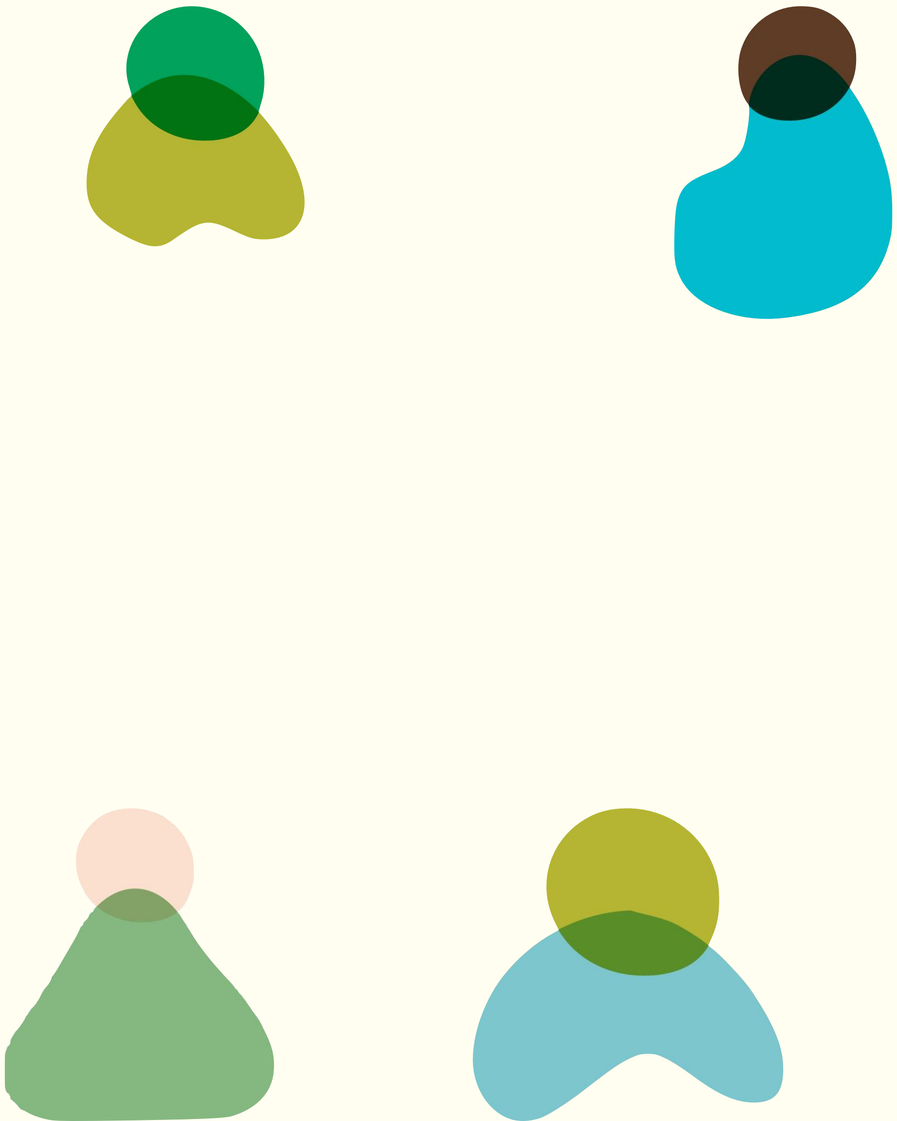
aRChiTeCtuRe
fOr
cONtiNuoUs DeLiVeRy

Continuous Delivery

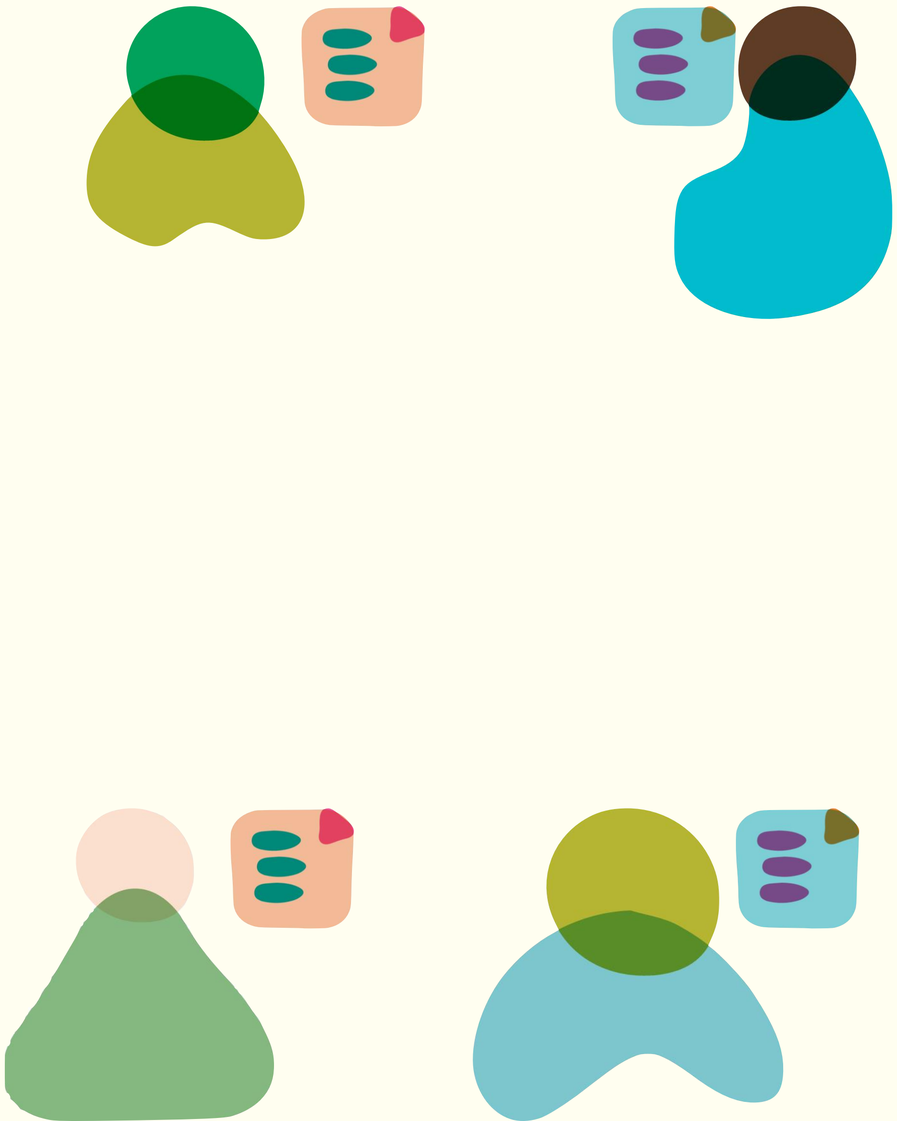
integration as an engineering practice over time

integration as an engineering practice over time

1980

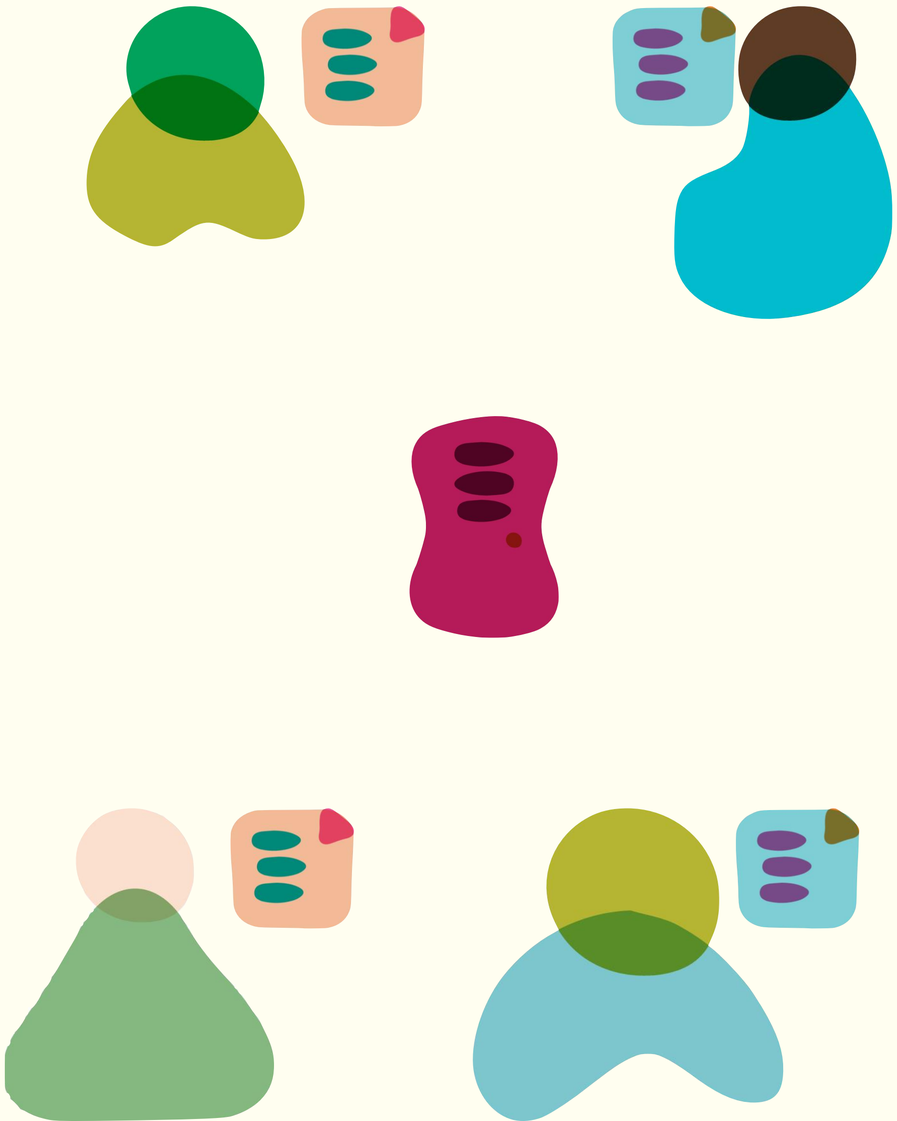


1980

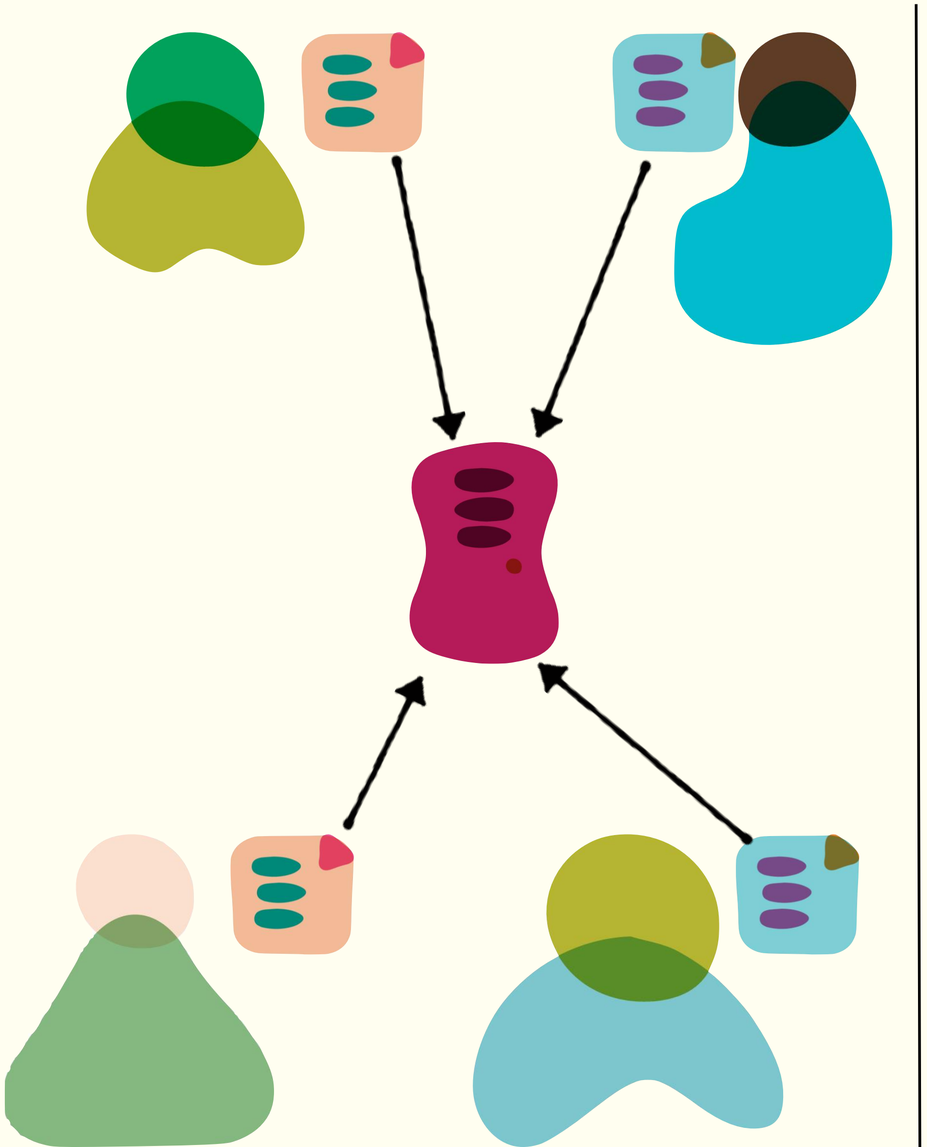


1980

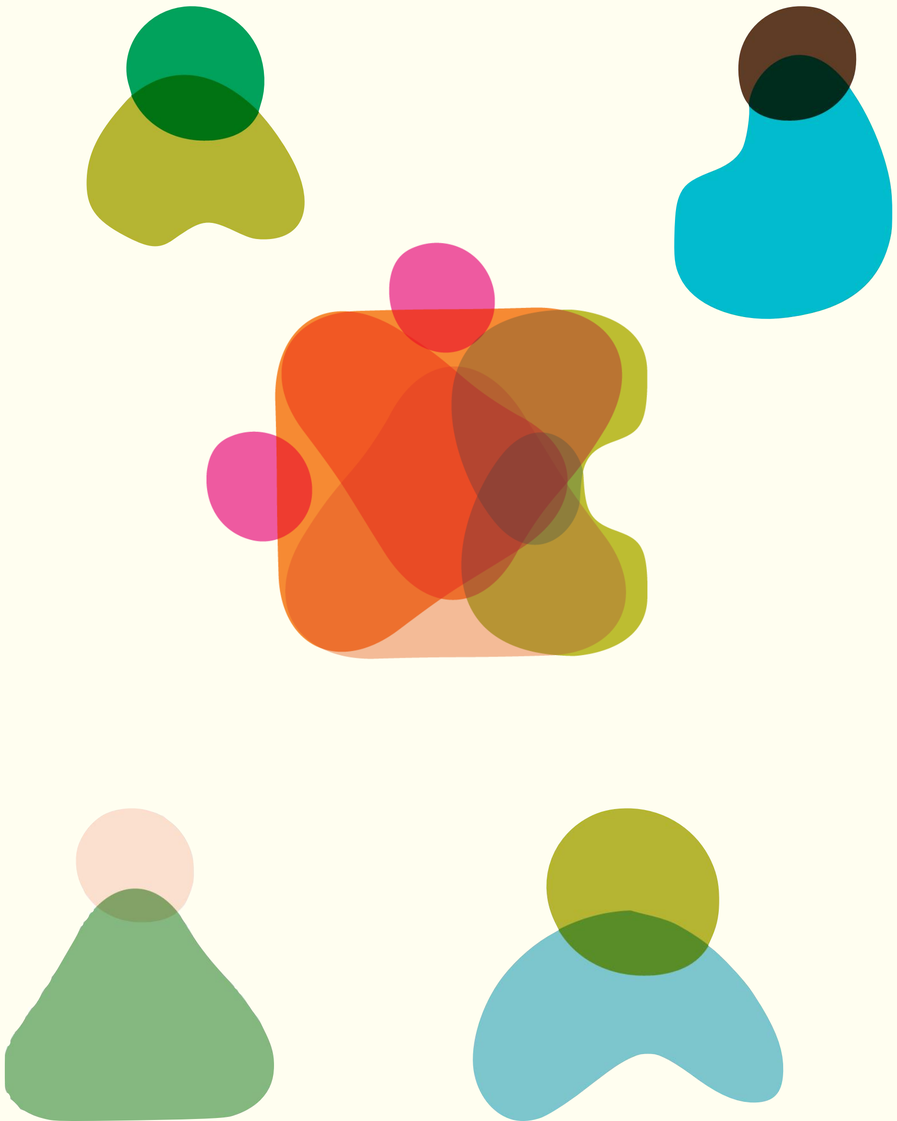
integration phase



1980



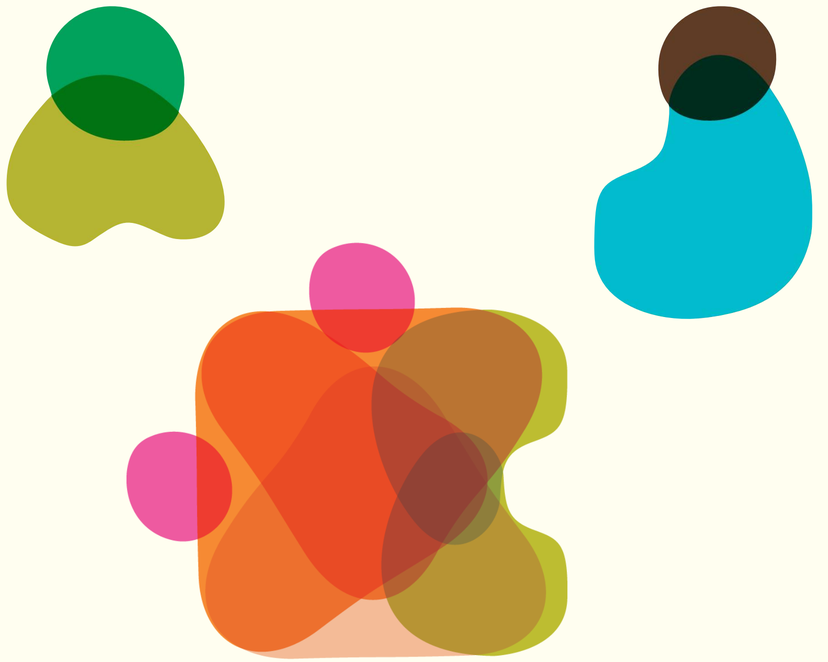
1980



1980

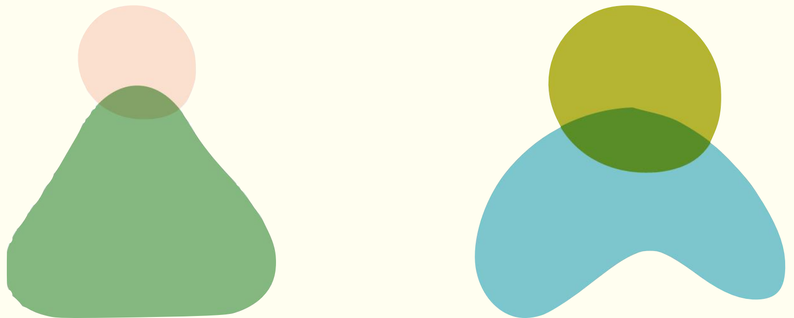
!!?!

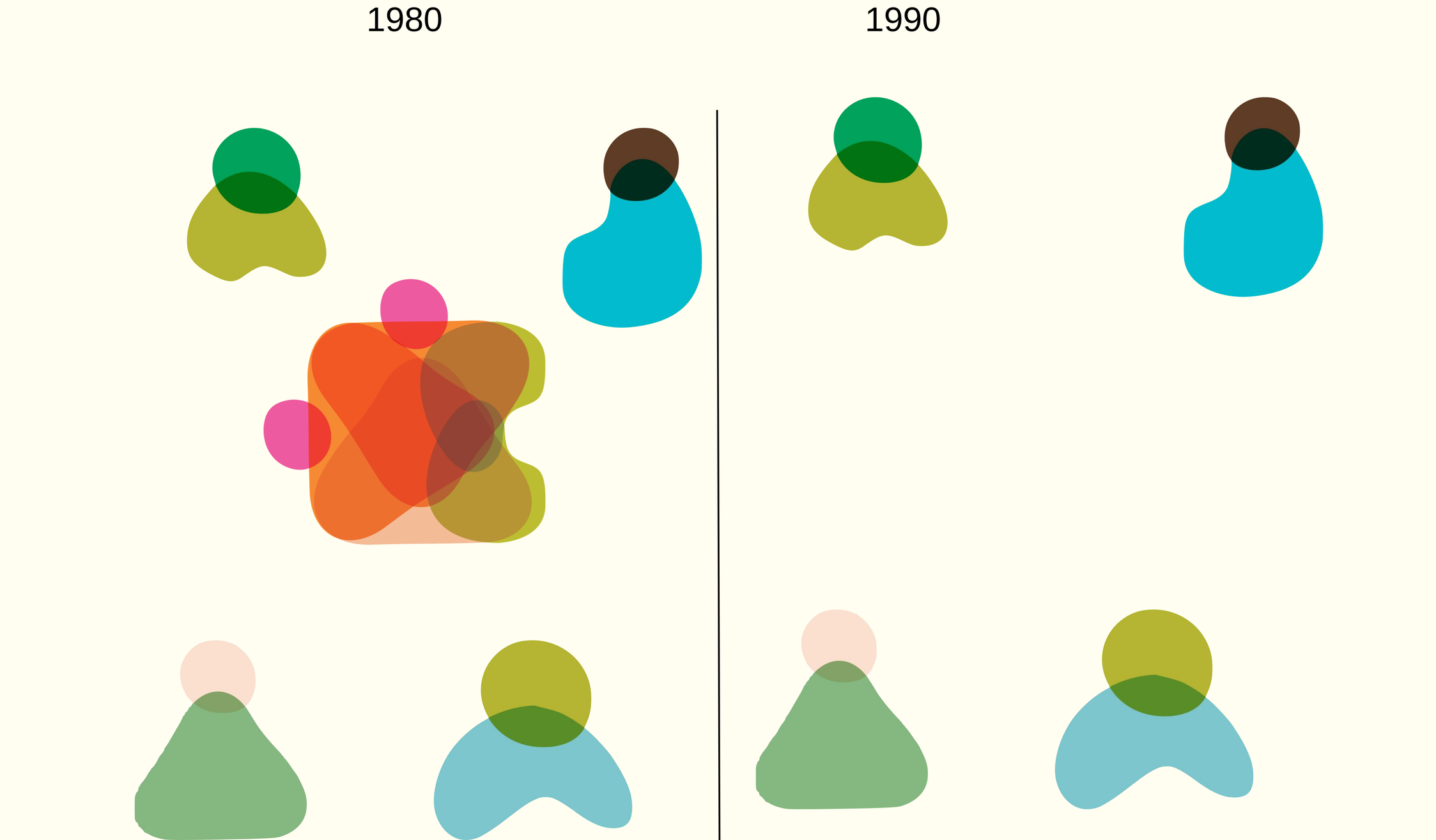
!??!?



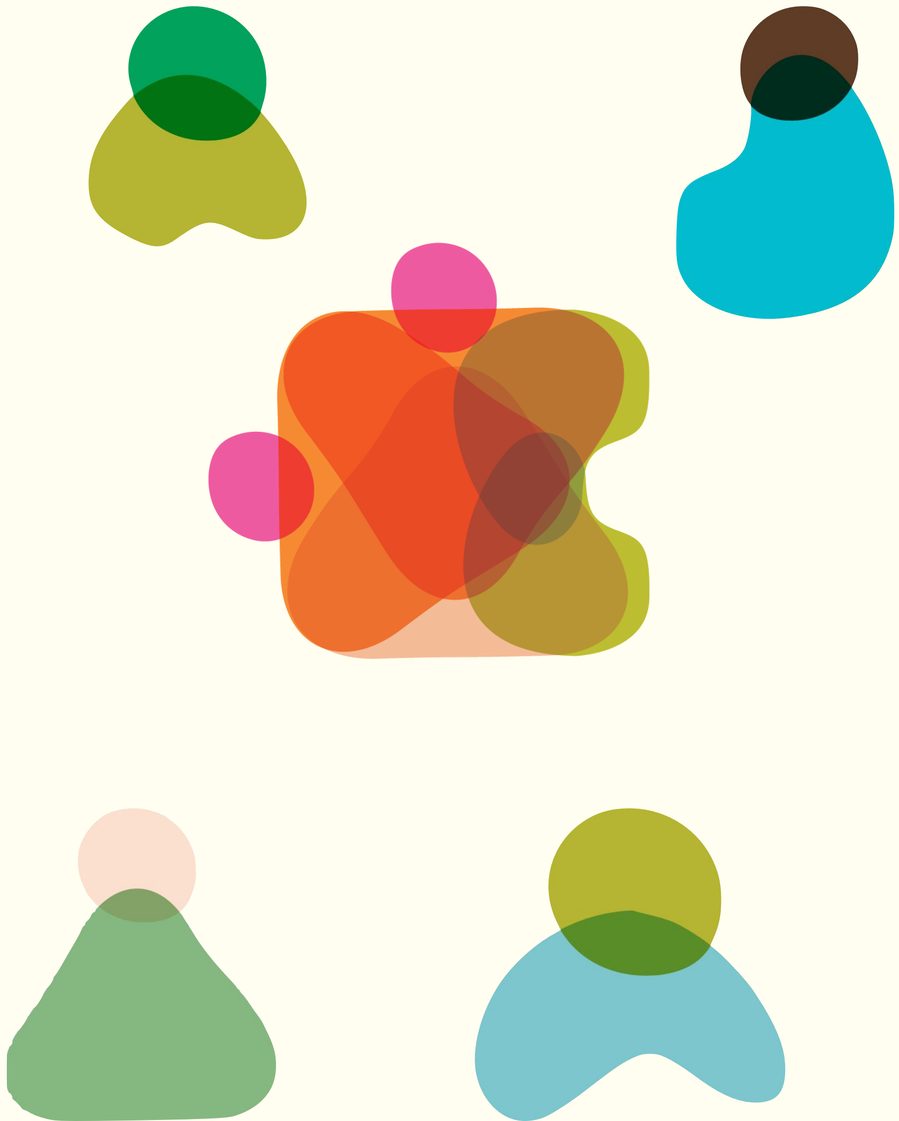
?!?

?!?

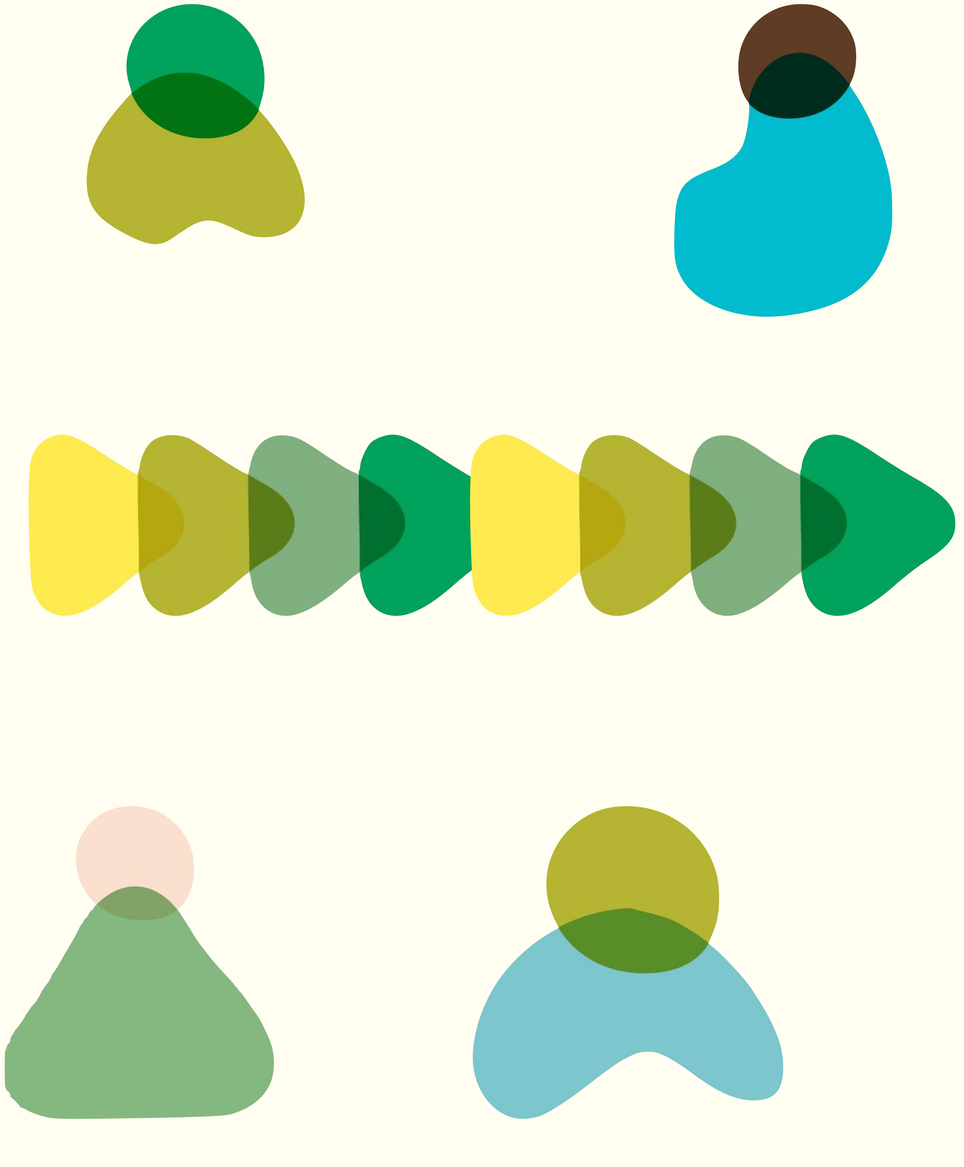



1980
1990

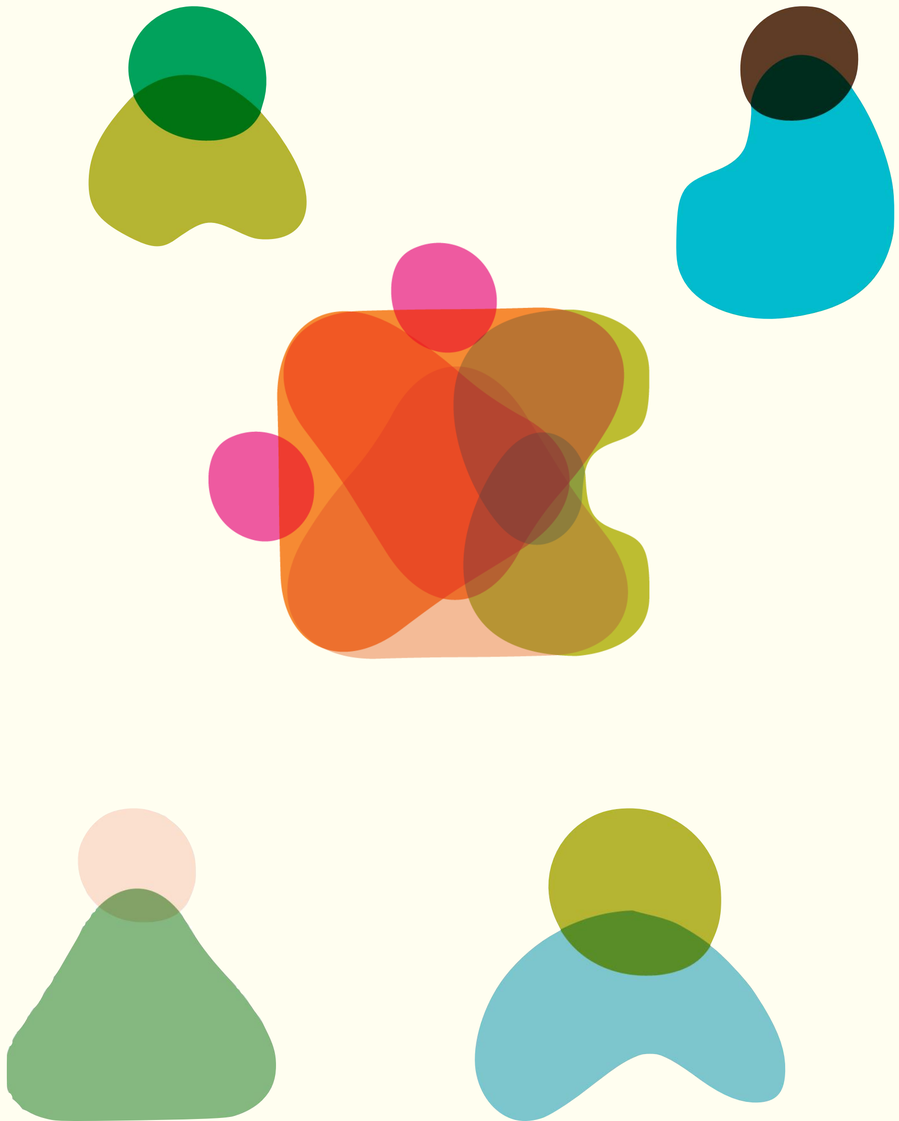
1980



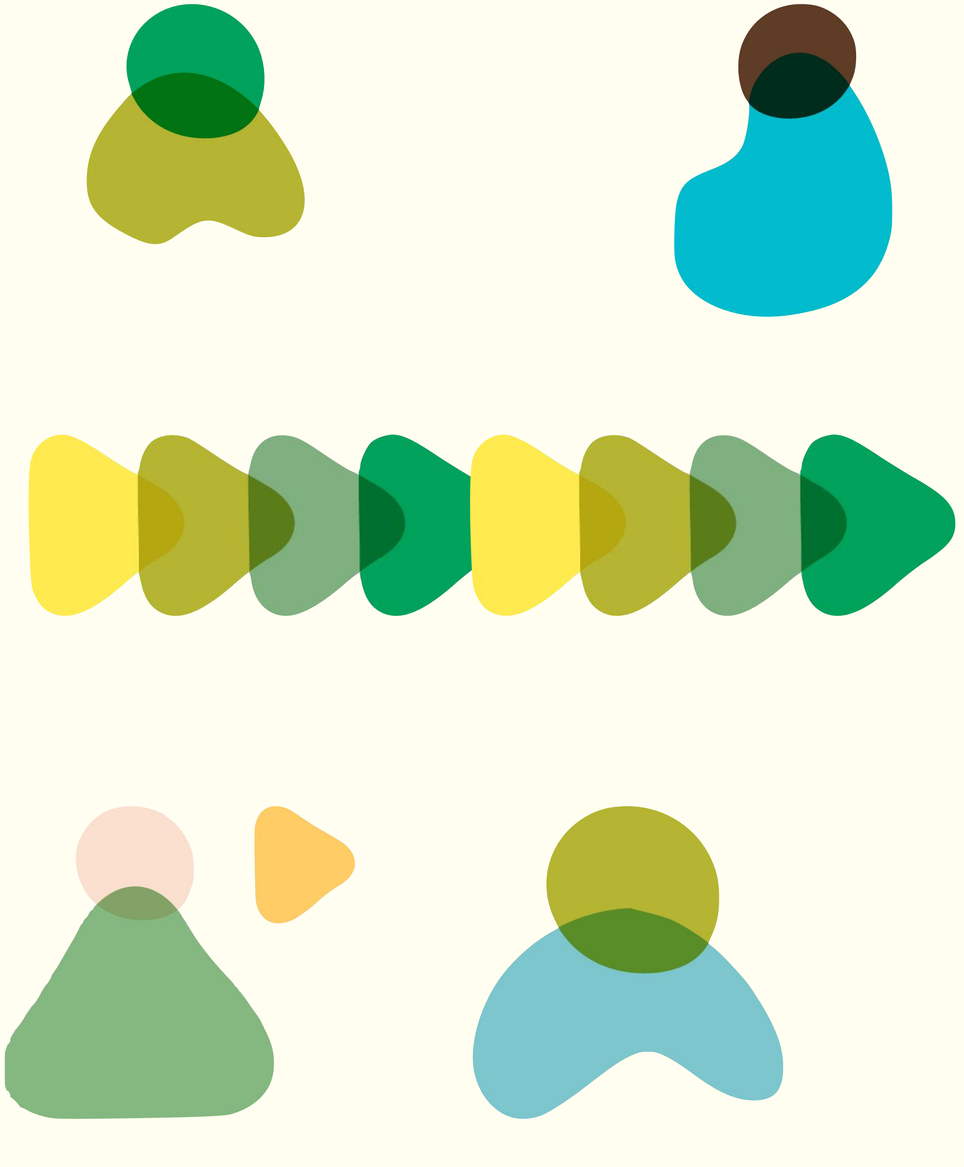
1990



1980

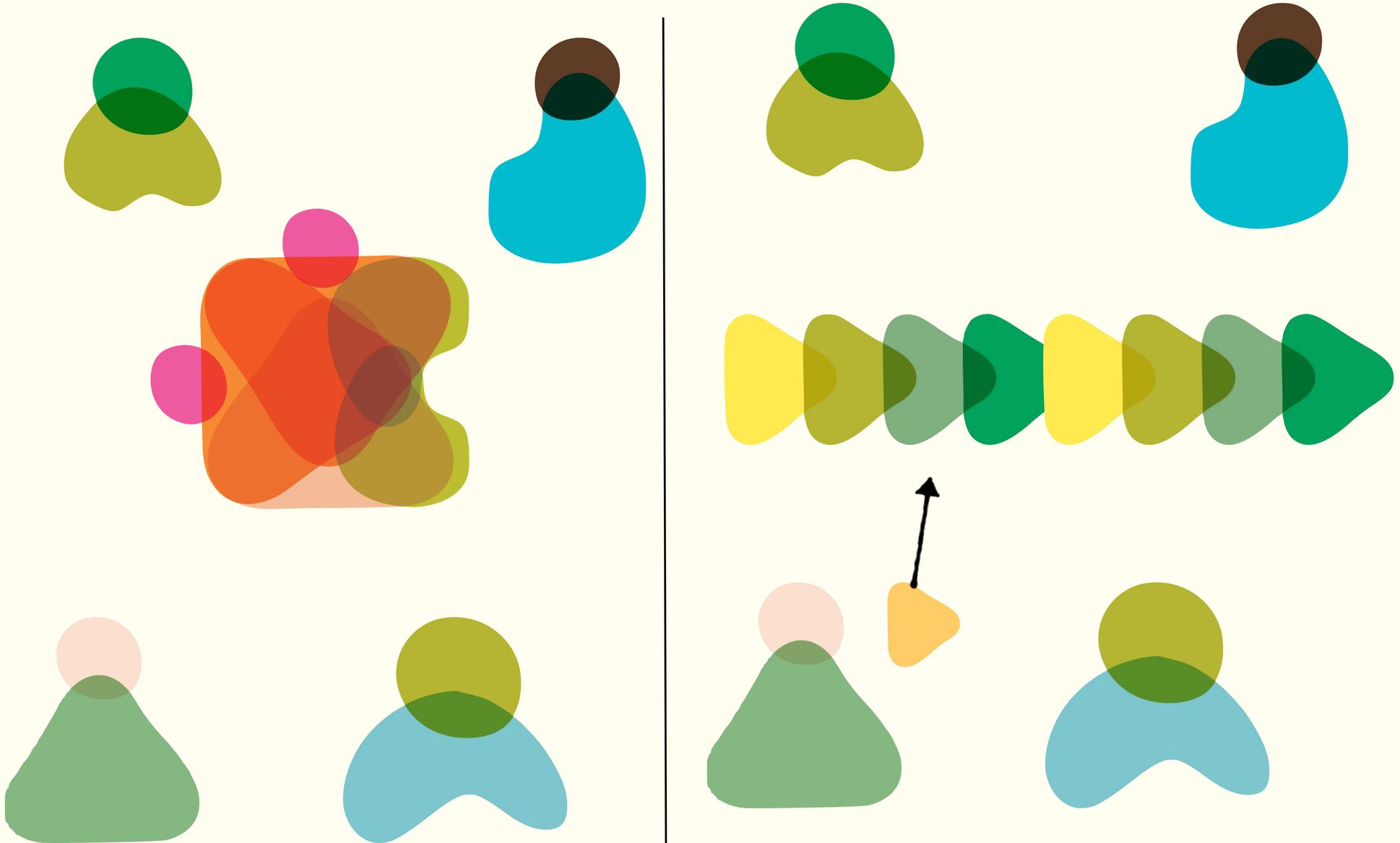


1990

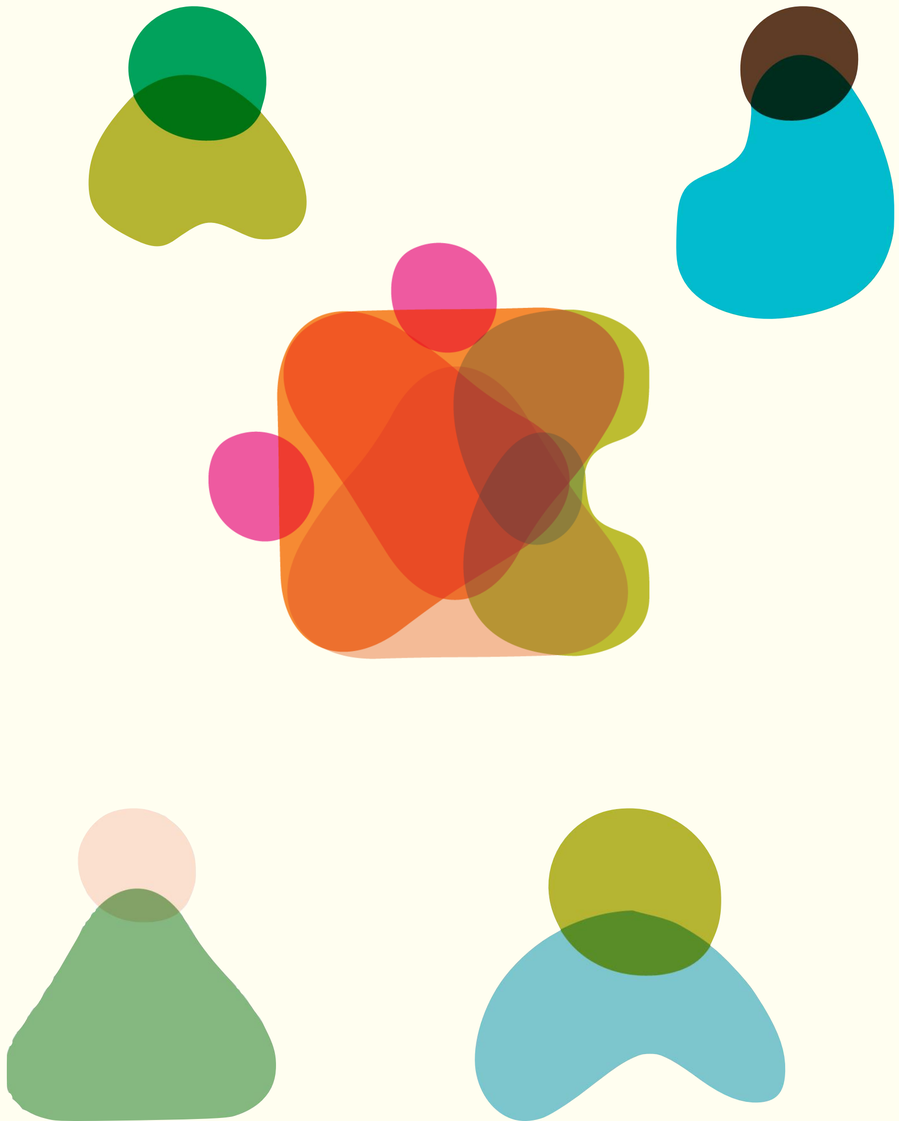


1980

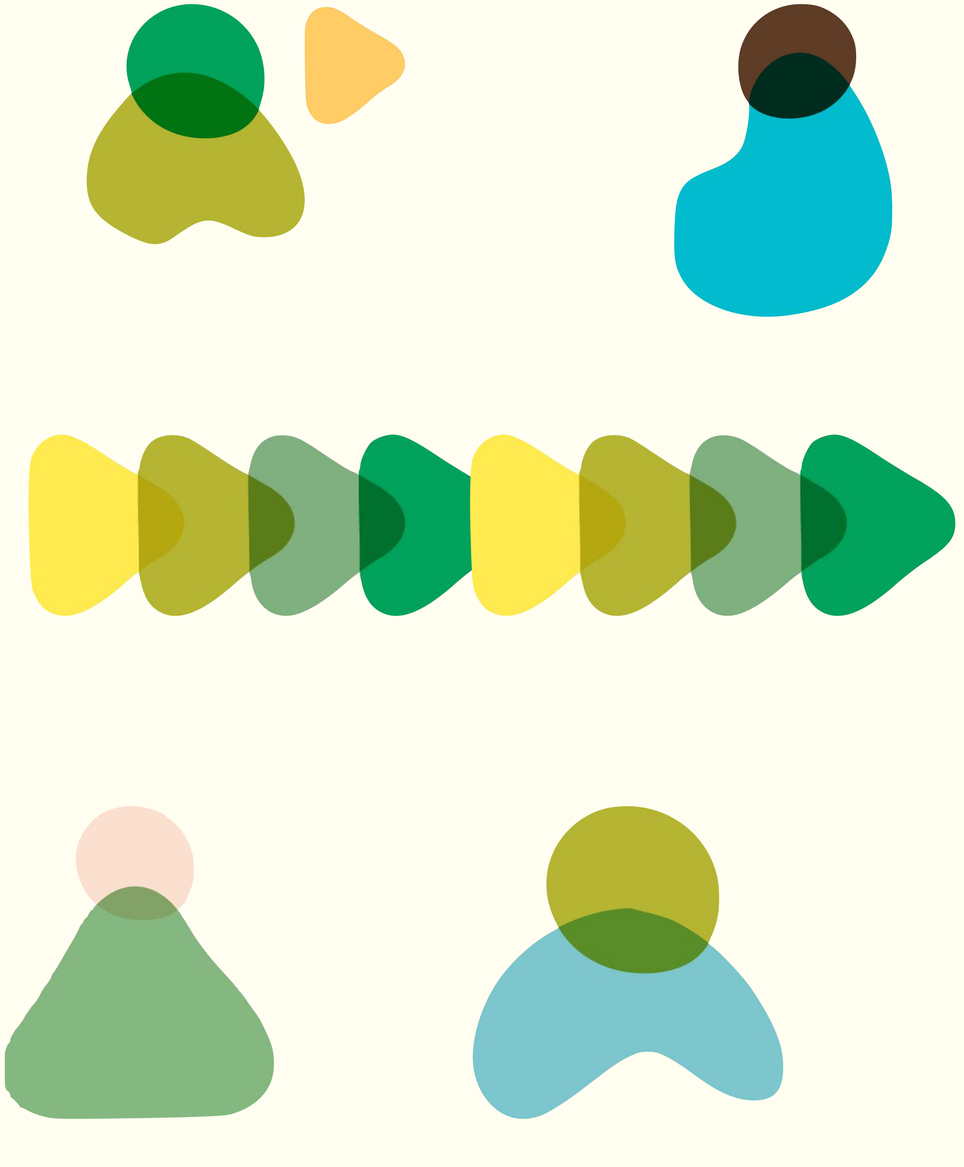
1990



1980

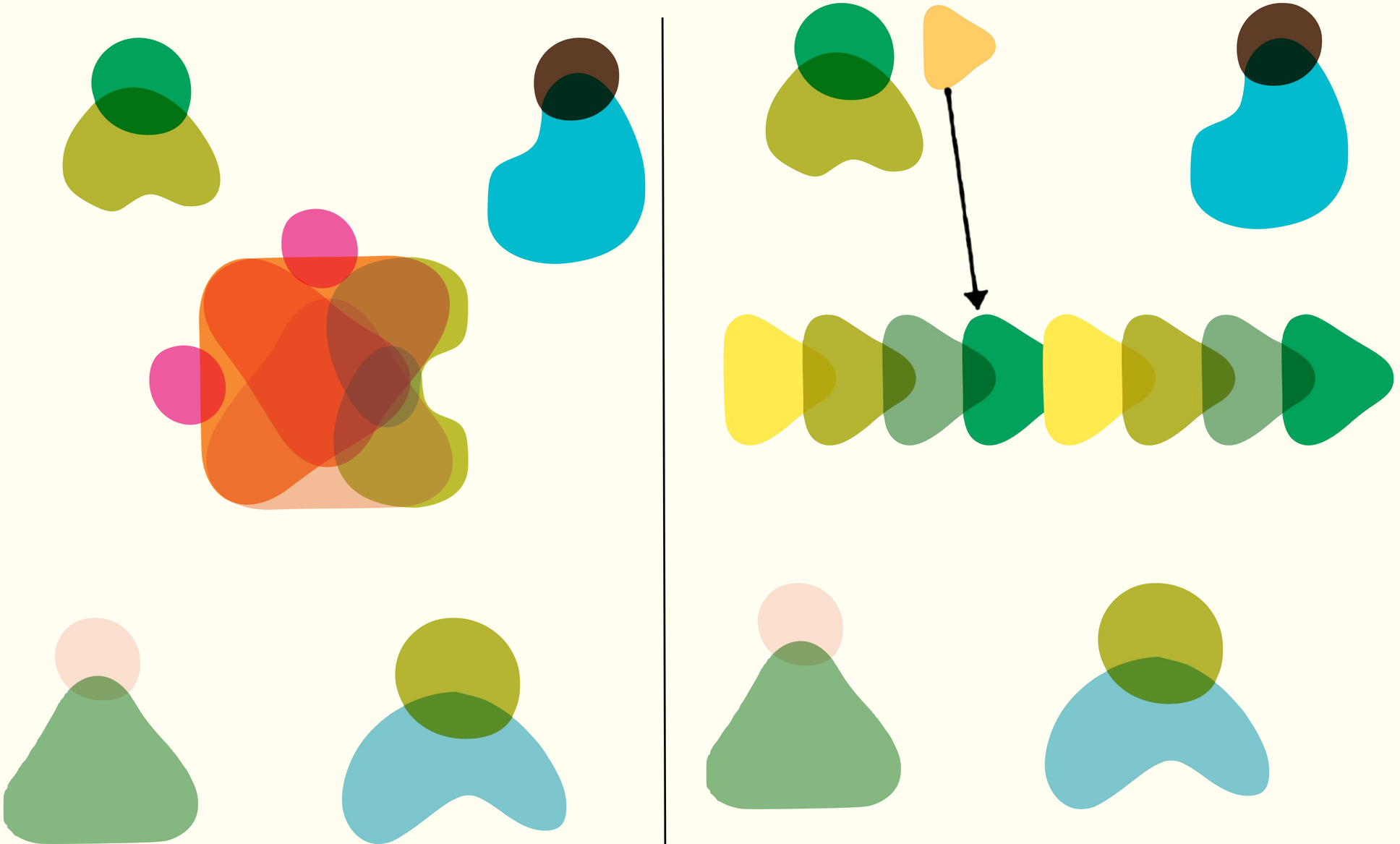


1990

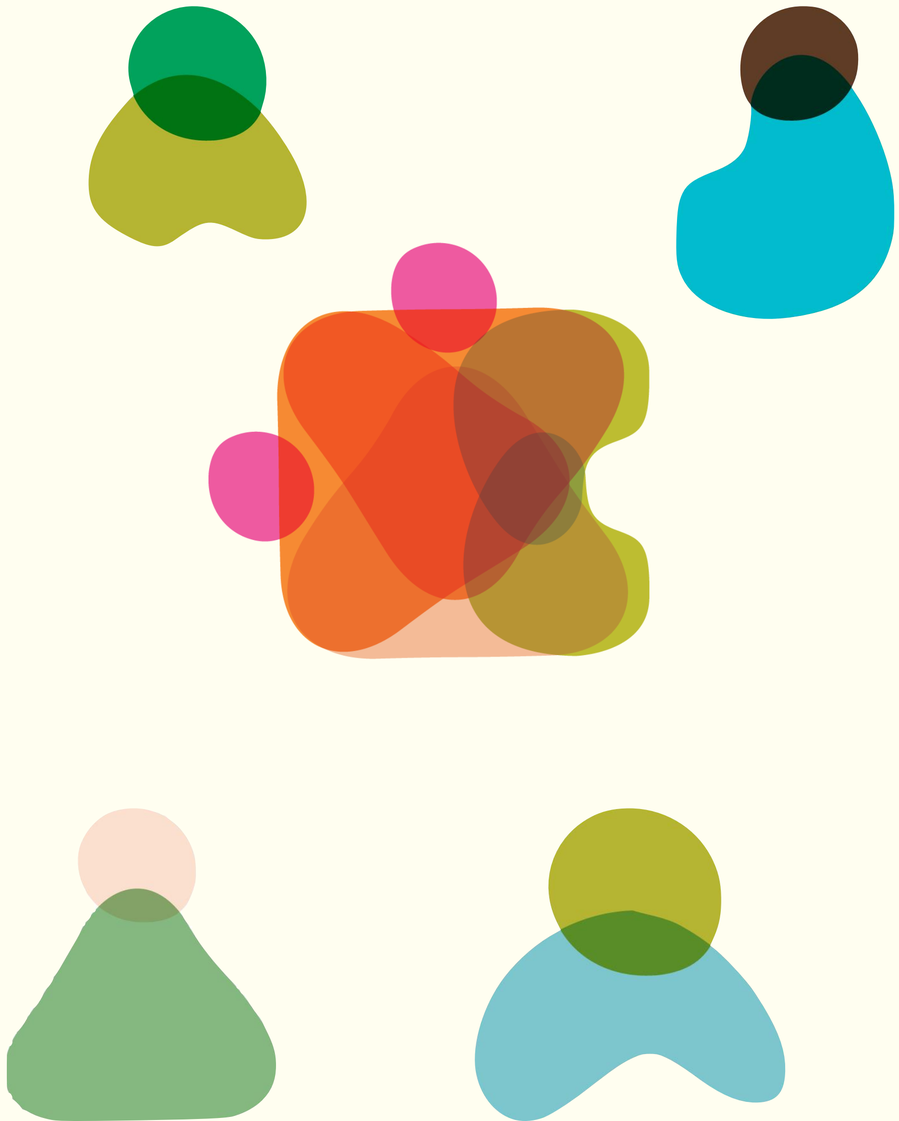


1980

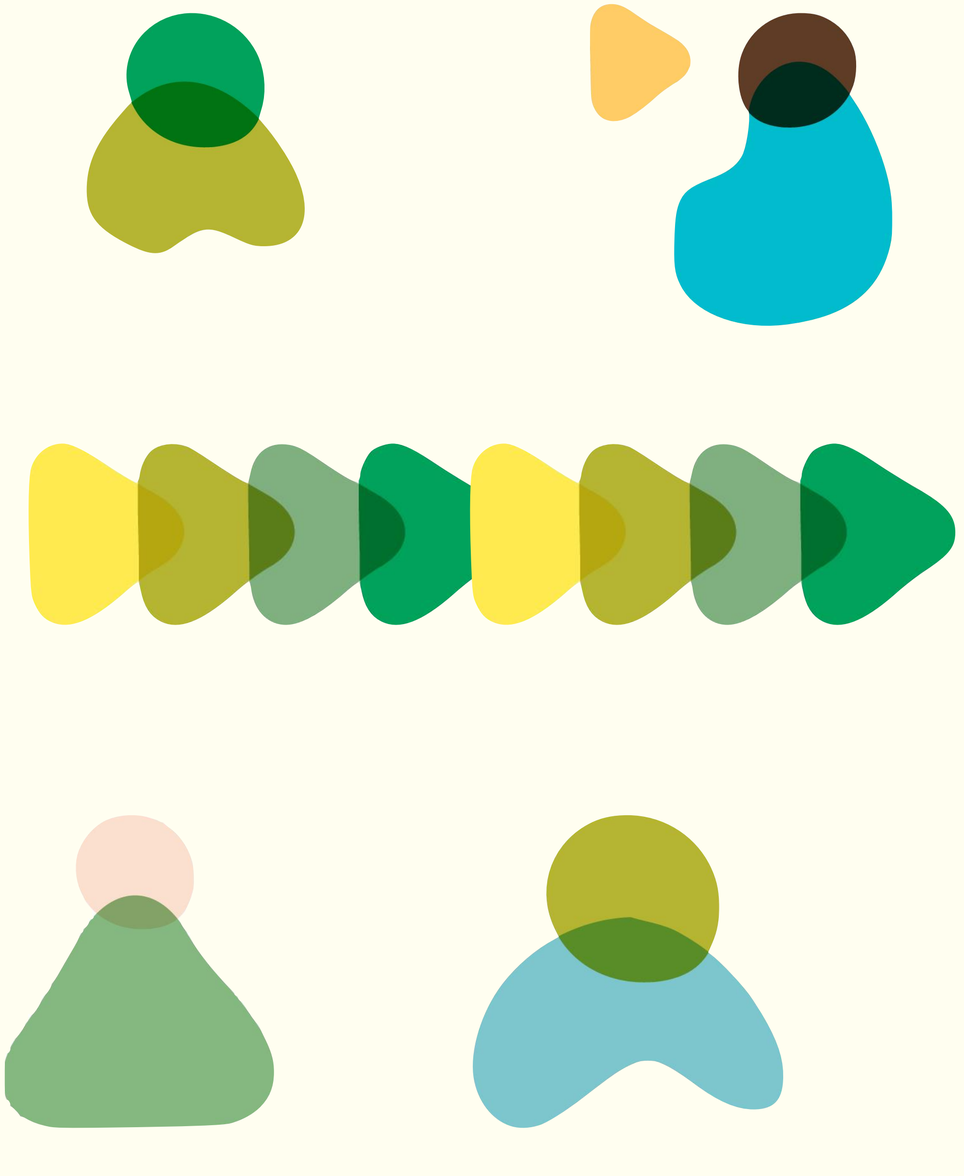
1990



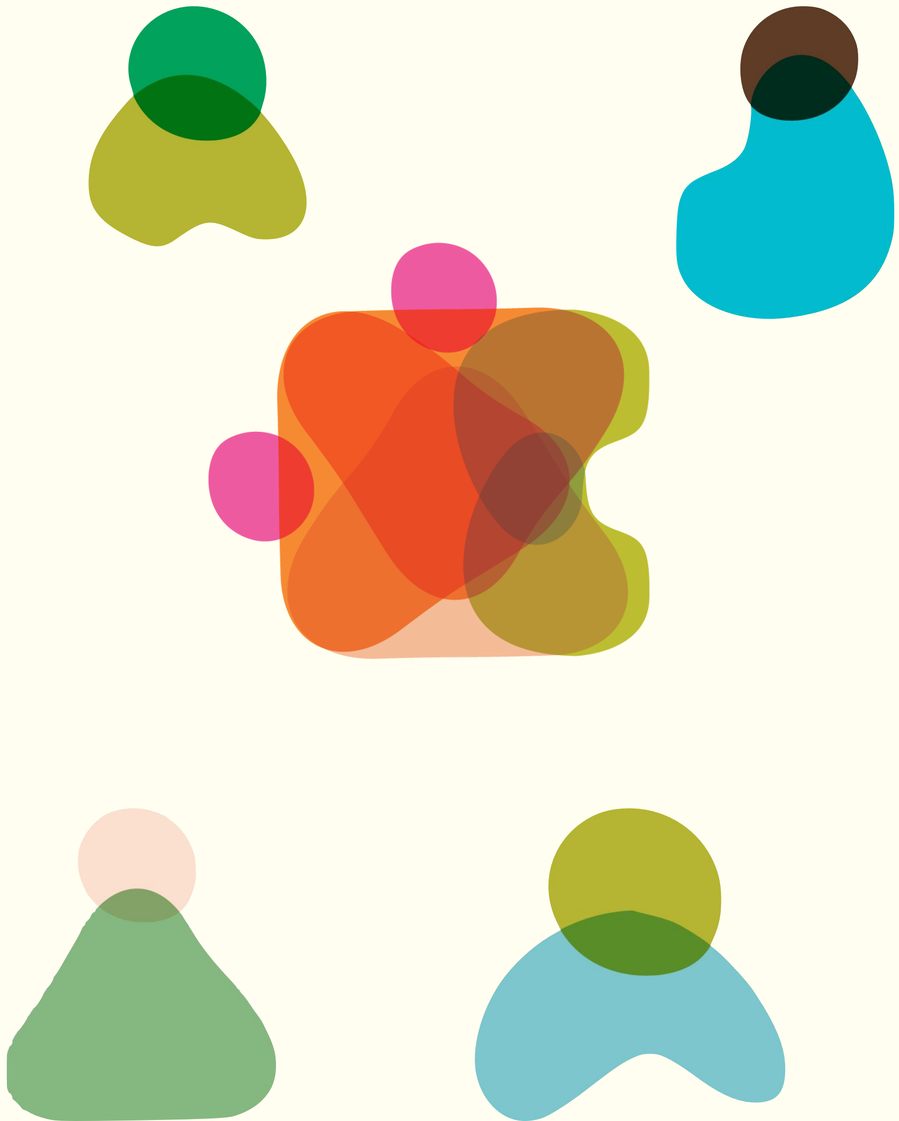
1980



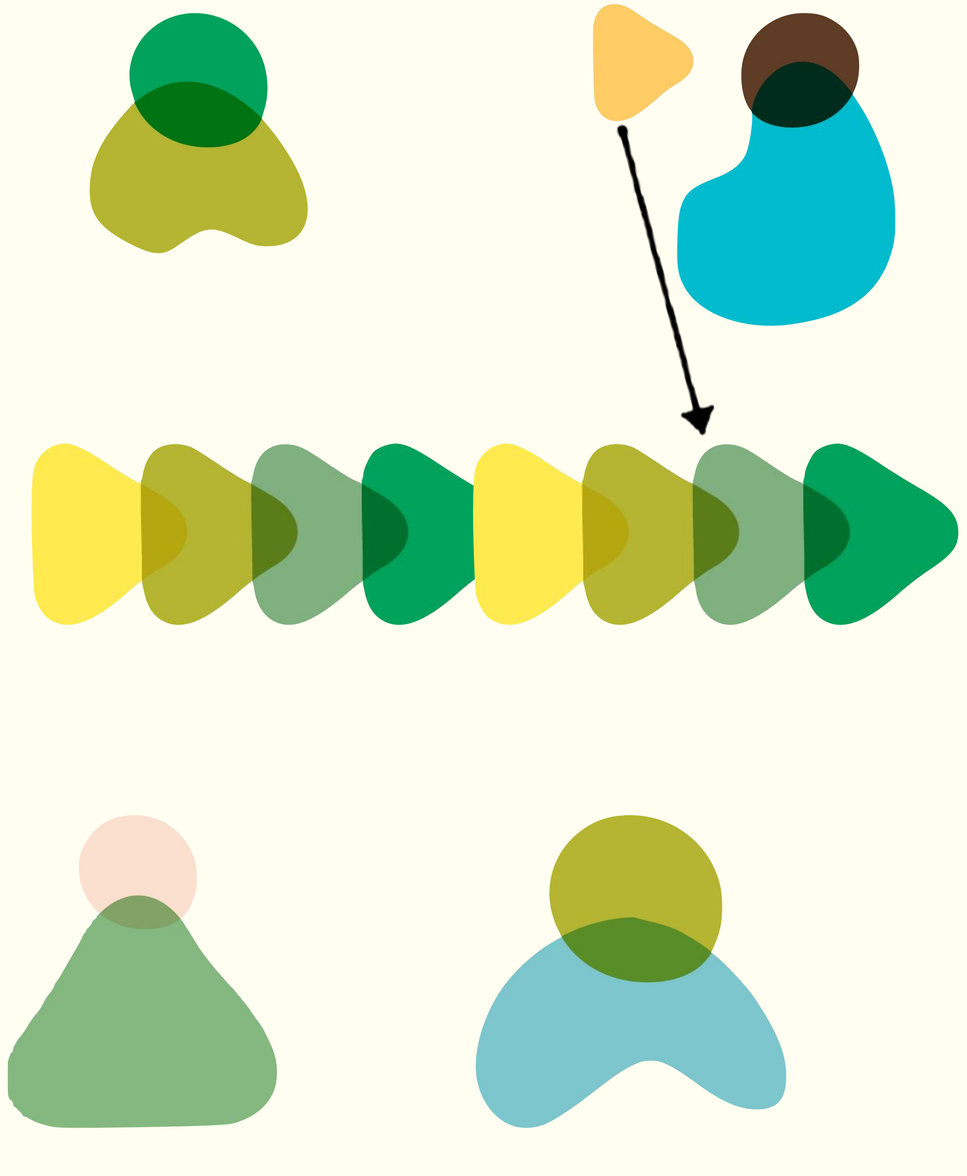
1990



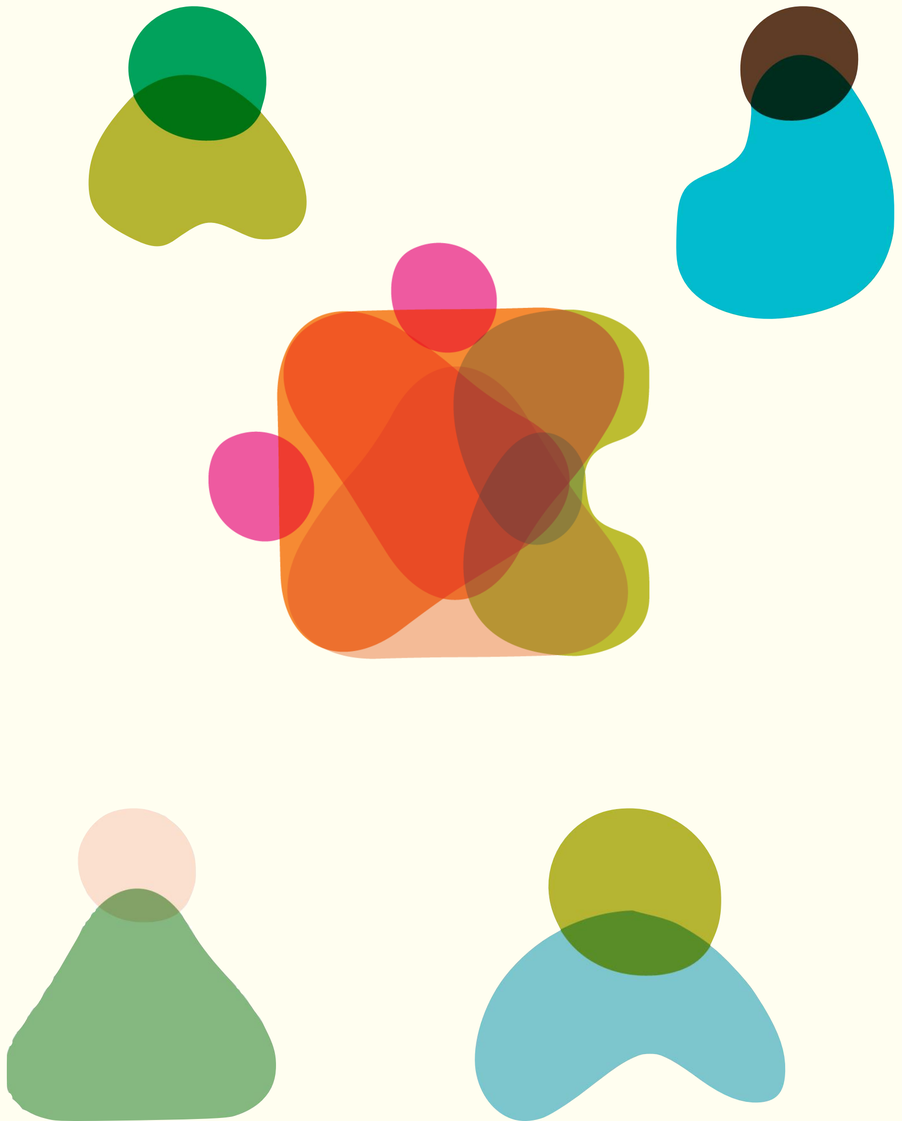
1980



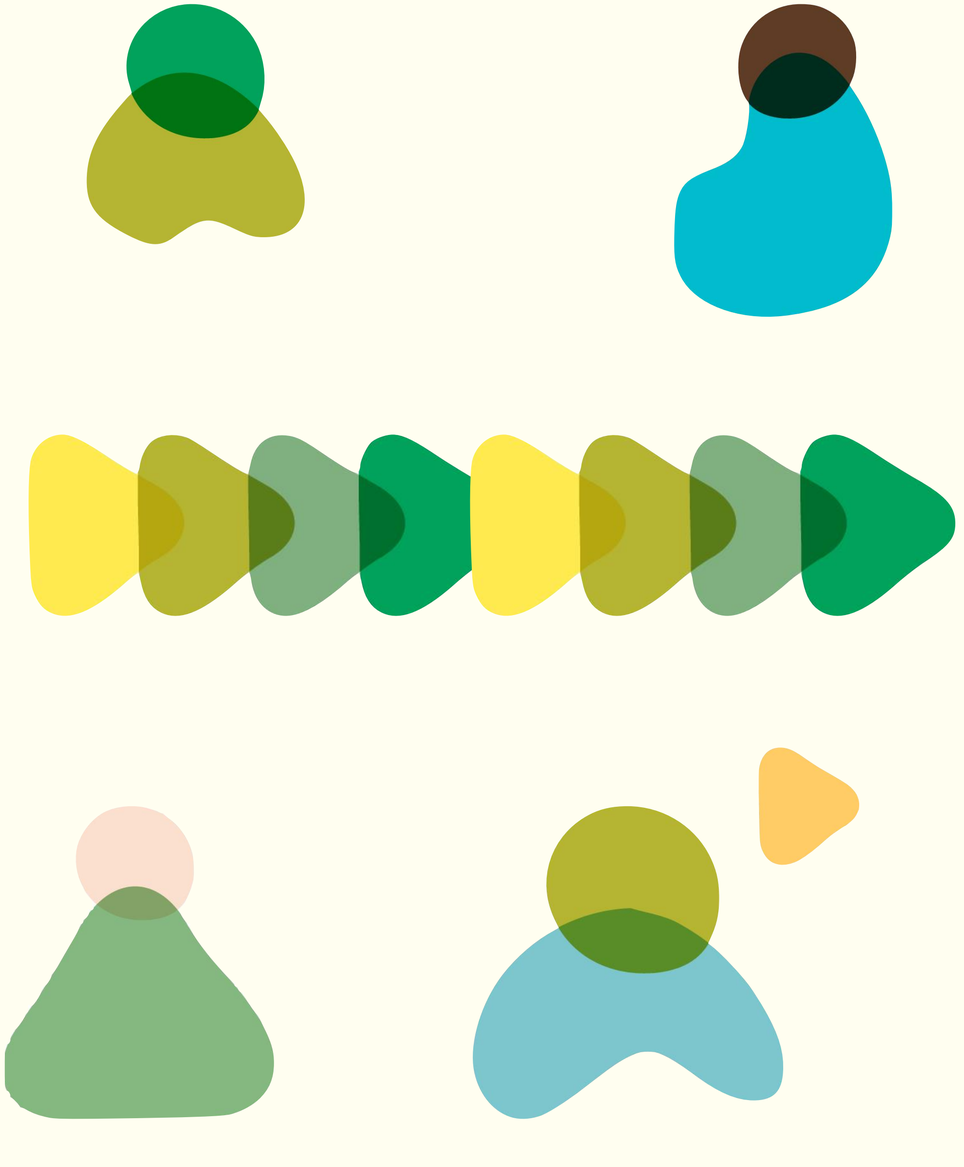
1990



1980

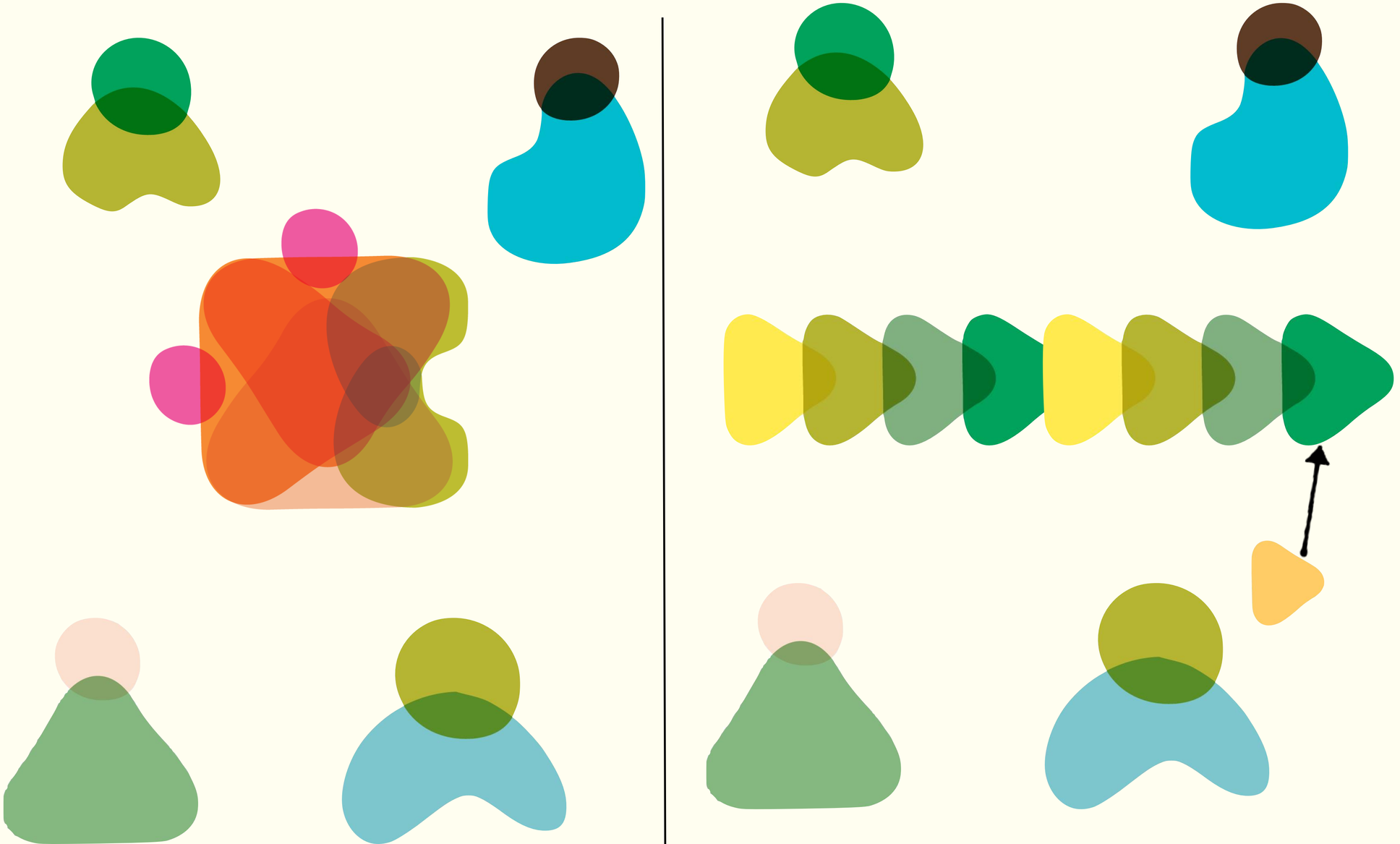


1990

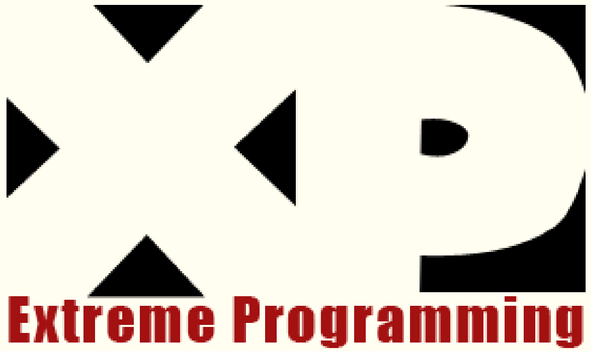


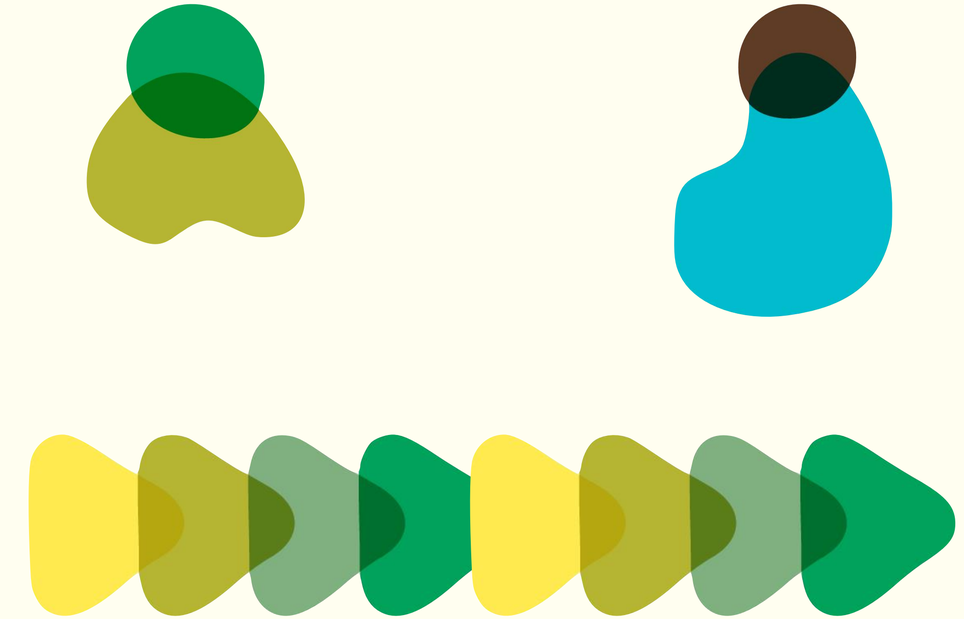
1980

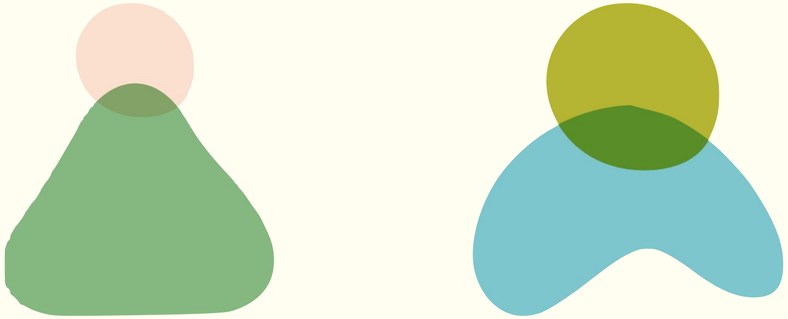
1990



1990

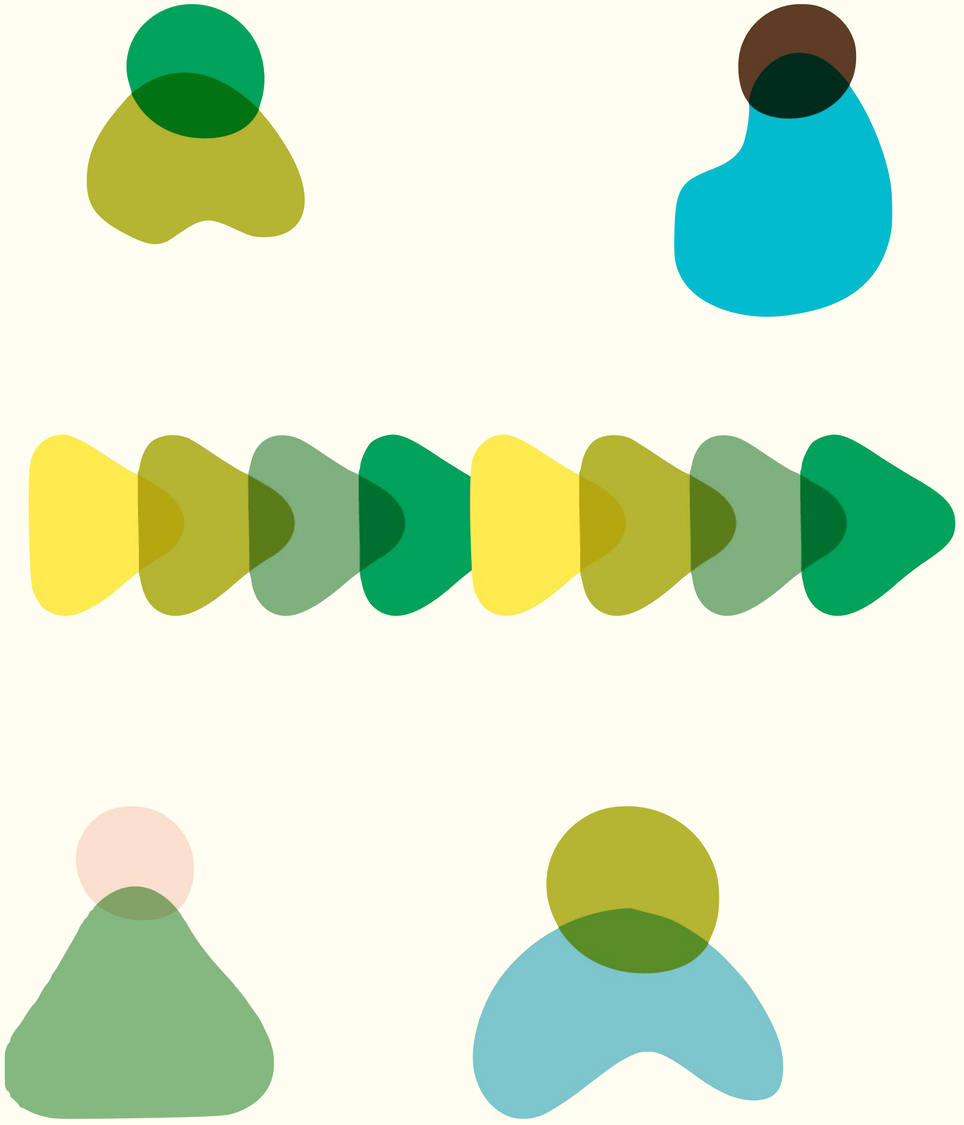






1990

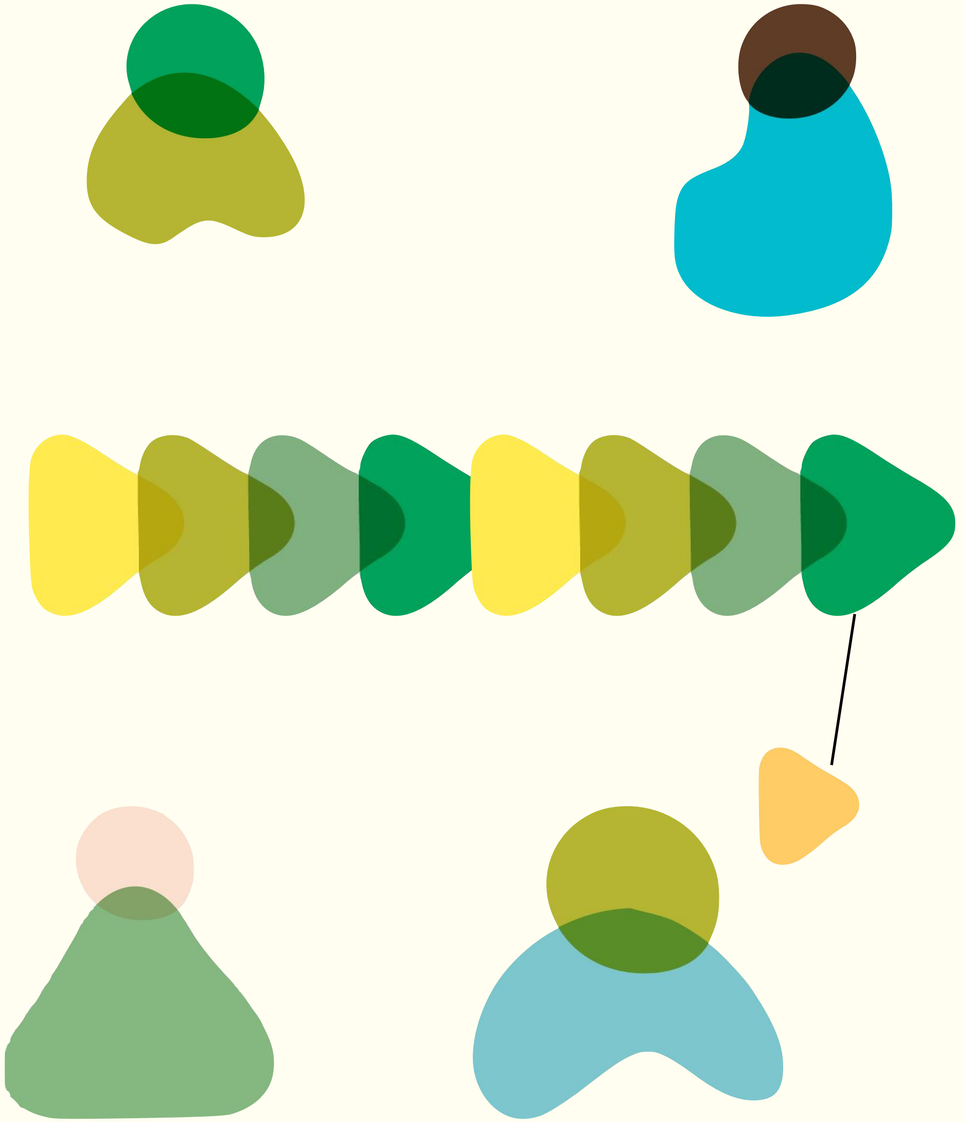
continuous integration:



1990

continuous integration:

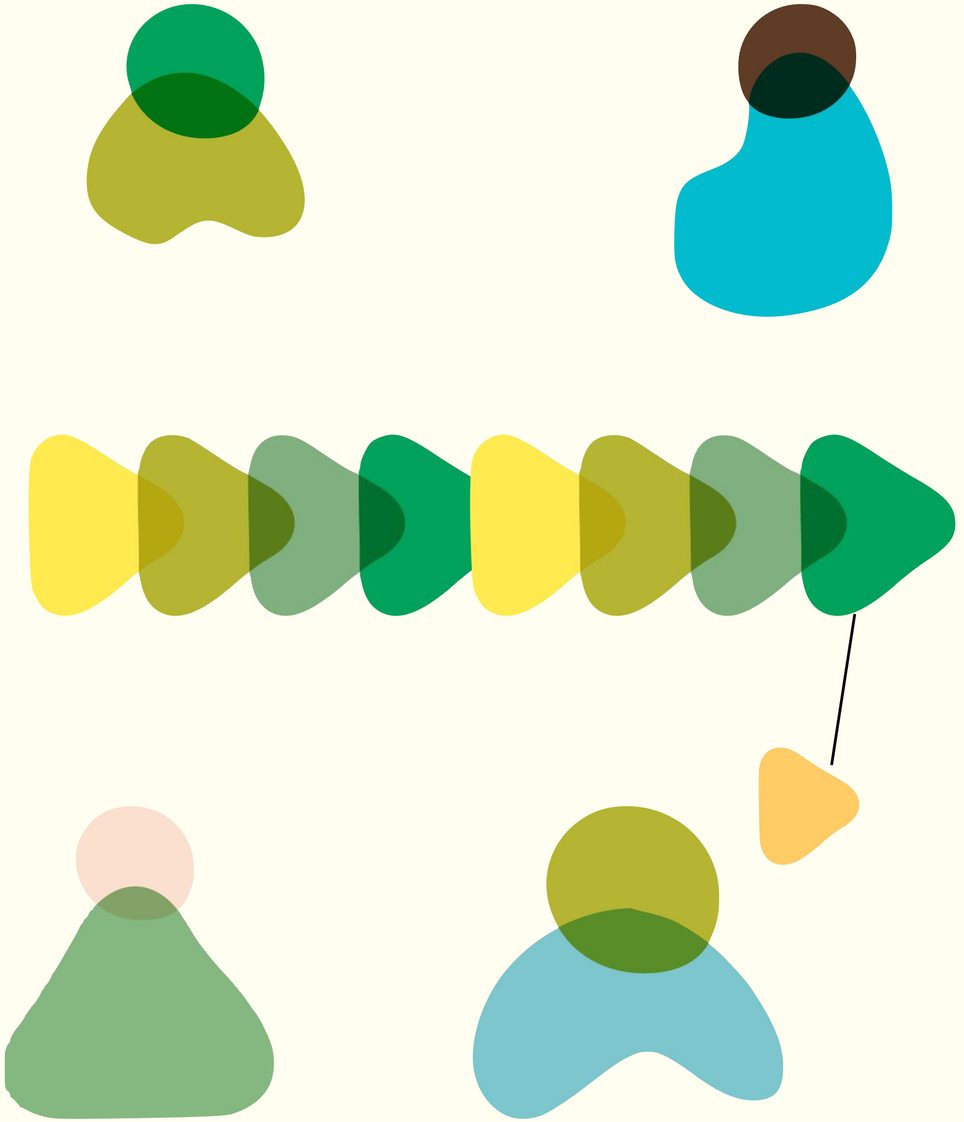
everyone commits to trunk at least once per day



1990

continuous integration:

everyone commits to trunk at least once per day *



*teams using feature branching aren’t doing continuous integration**

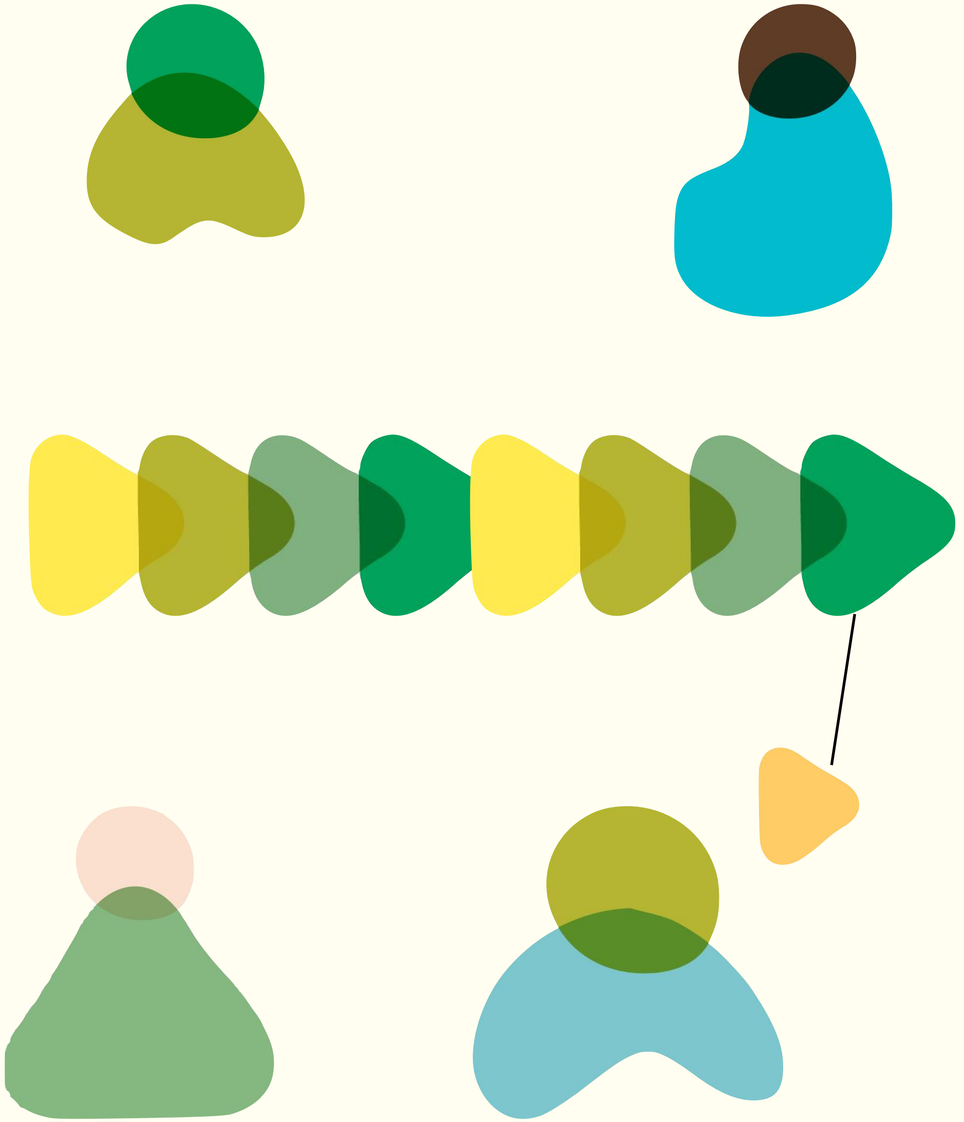
continuous integration:

everyone commits to trunk at least once per day *

*teams using feature branching aren’t doing continuous integration**

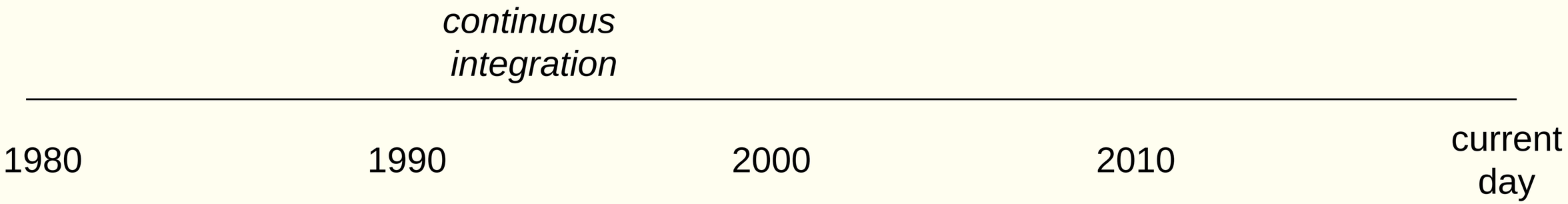
**unless all branches live < 1 day

1990



integration as an engineering practice over time

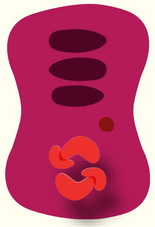
integration as an engineering practice over time



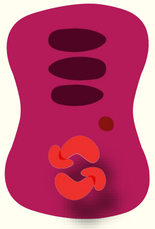
integration as an engineering practice over time

continuous integration

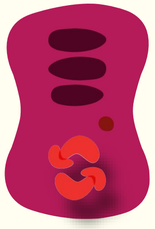
1980 1990 2000 2010 current day



continuous integration

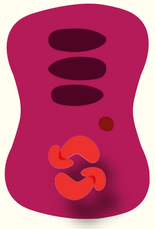


continuous integration



canonical integration point

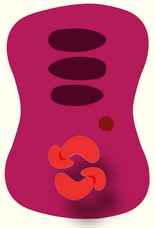
continuous integration



canonical integration point

useful platform to do other *stuff*

continuous integration

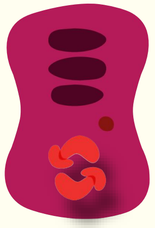


canonical integration point

useful platform to do other *stuff*

unit testing

continuous integration



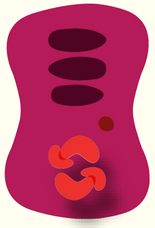
canonical integration point

useful platform to do other *stuff*

unit testing

functional testing

continuous integration



canonical integration point

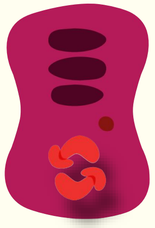
useful platform to do other *stuff*

unit testing

functional testing

integration testing

continuous integration



canonical integration point

useful platform to do other *stuff*

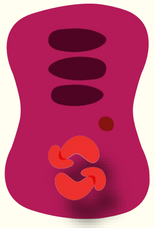
unit testing

functional testing

integration testing

automated machine provisioning

continuous integration



canonical integration point

useful platform to do other *stuff*

unit testing

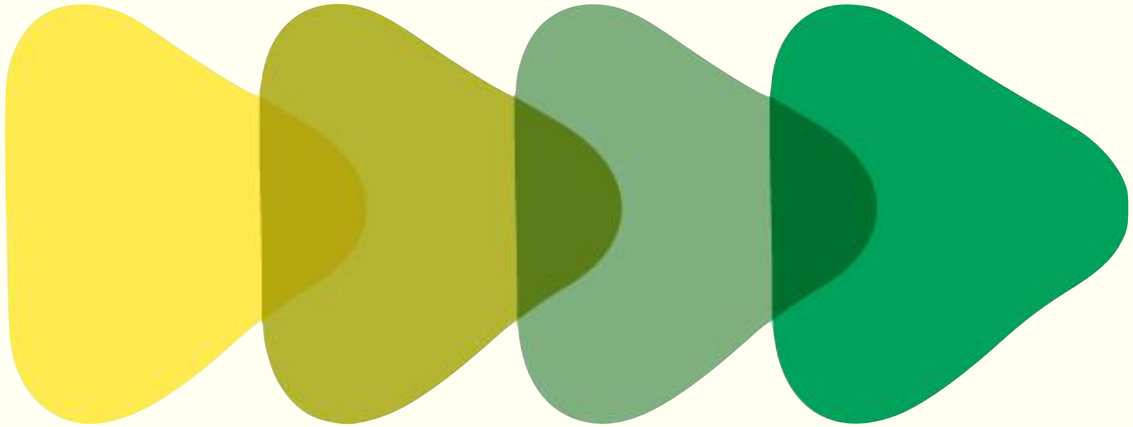
functional testing

integration testing

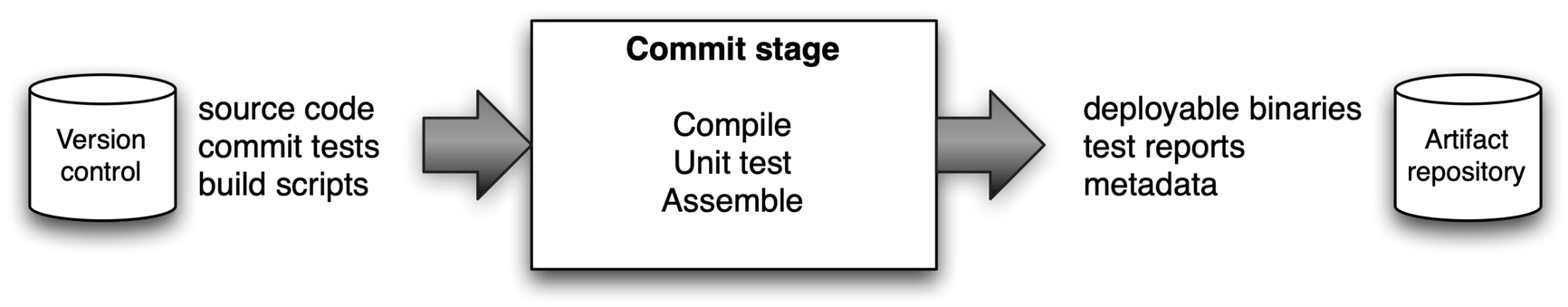
automated machine provisioning

deployment to production?

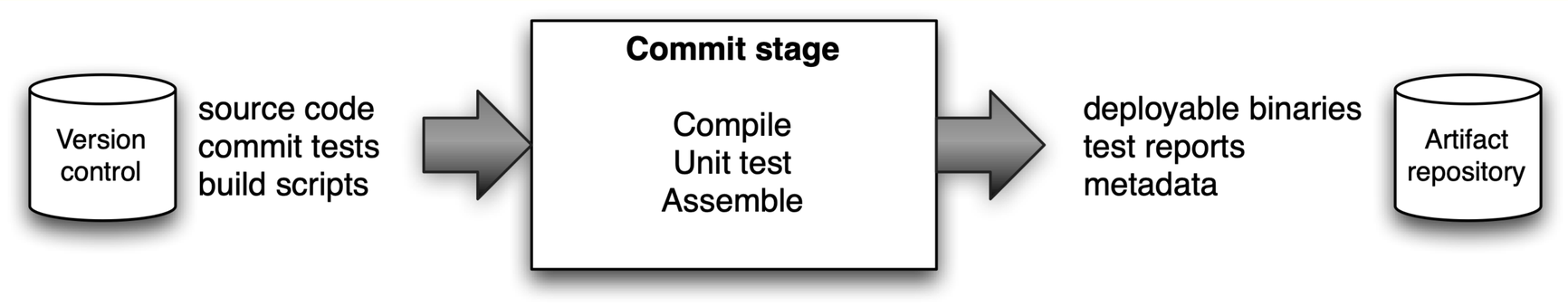
Deployment Pipelines



commit Stage

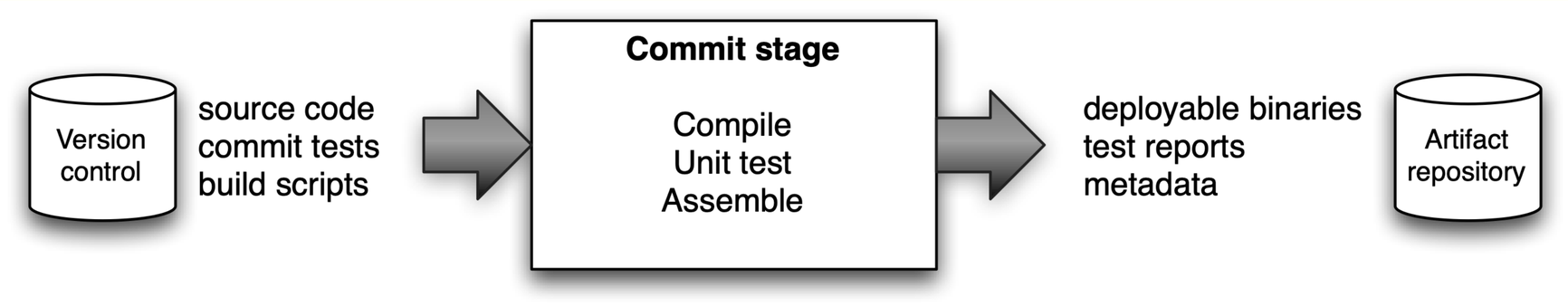


commit Stage



Run against each check-in

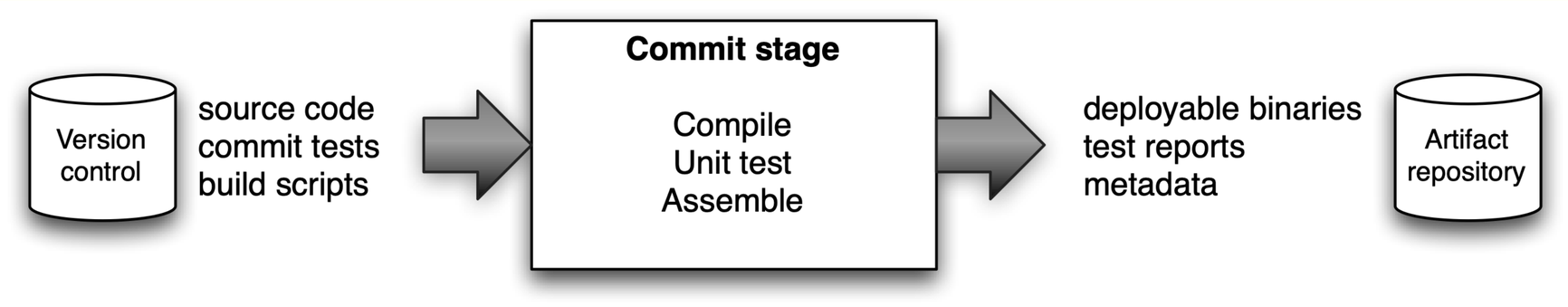
commit Stage



Run against each check-in

Starts building a release candidate

commit Stage

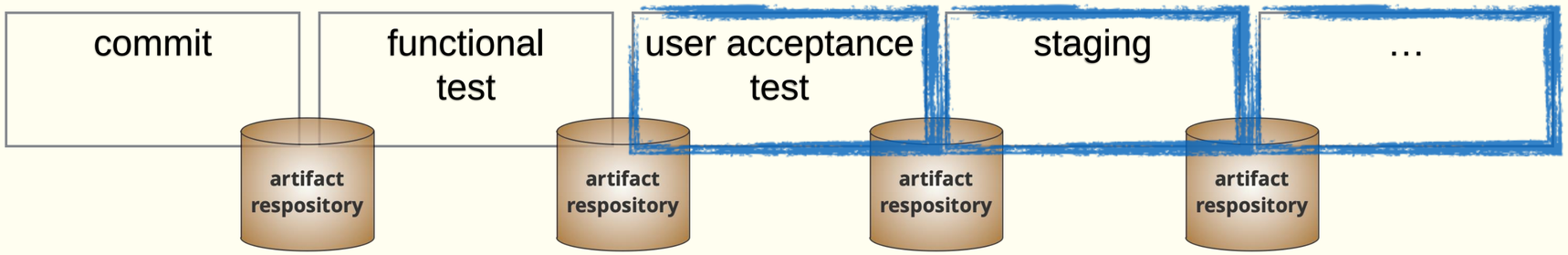


Run against each check-in

Starts building a release candidate

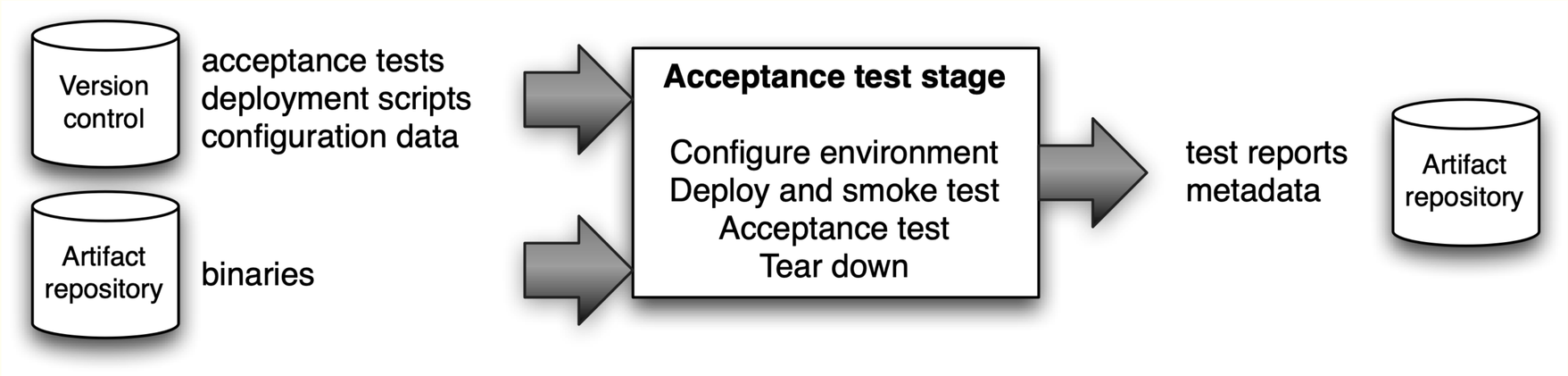
If it fails, fix it immediately

Pipeline Construction

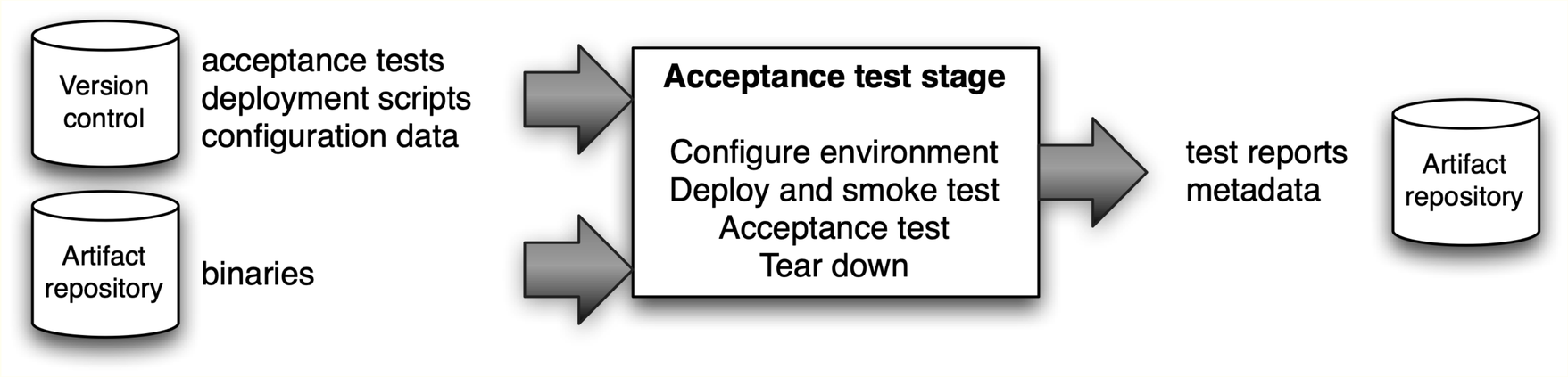


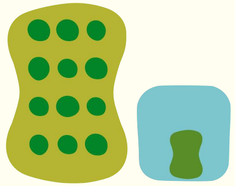
Pipeline stages = feedback opportunities

UAT Stage


Version control
acceptance tests
deployment scripts
configuration data
Artifact repository
binaries
Acceptance test stage
Configure environment
Deploy and smoke test
Acceptance test
Tear down
test reports
metadata
Artifact repository

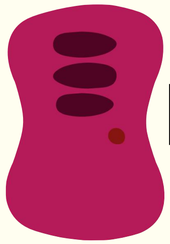
UAT Stage





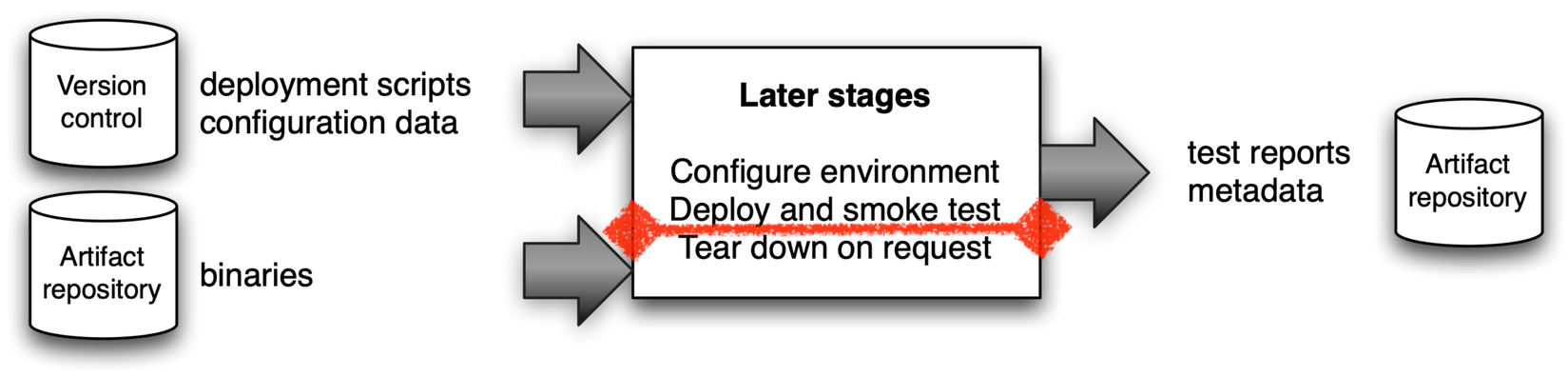
End-to-end tests in production-like environment

Triggered when upstream stage passes

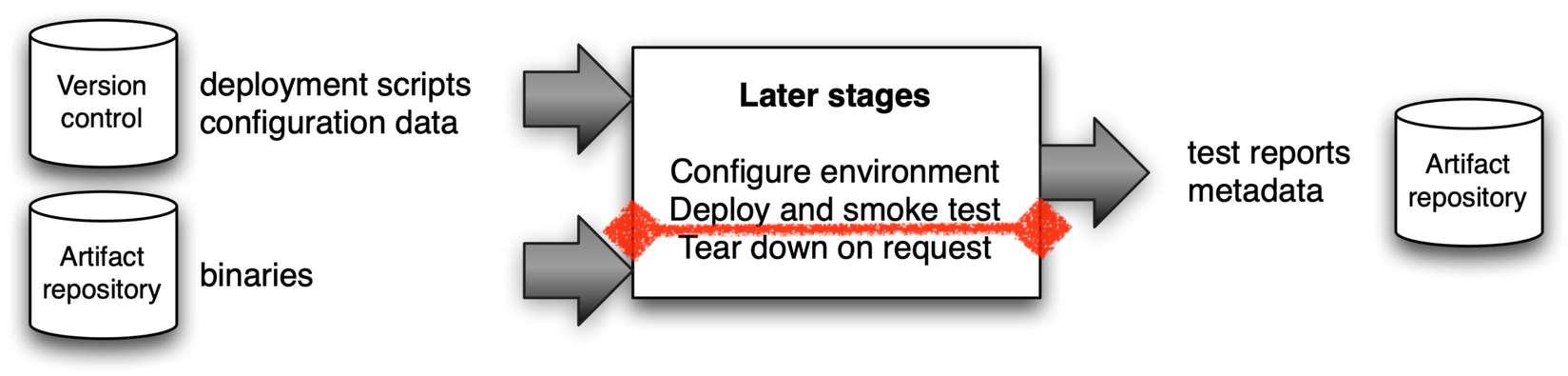


First DevOps-centric build

Manual Stage

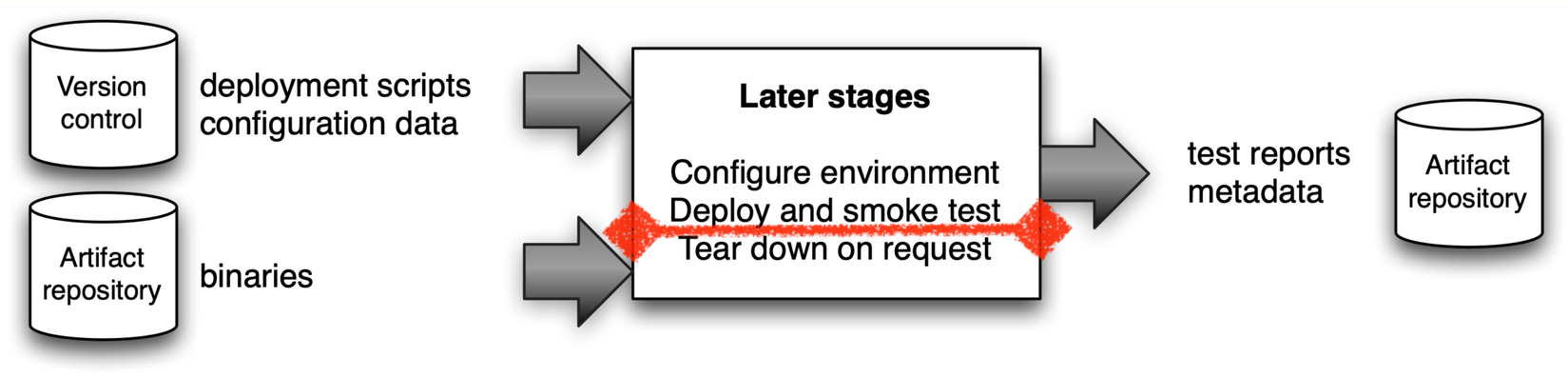


Manual Stage



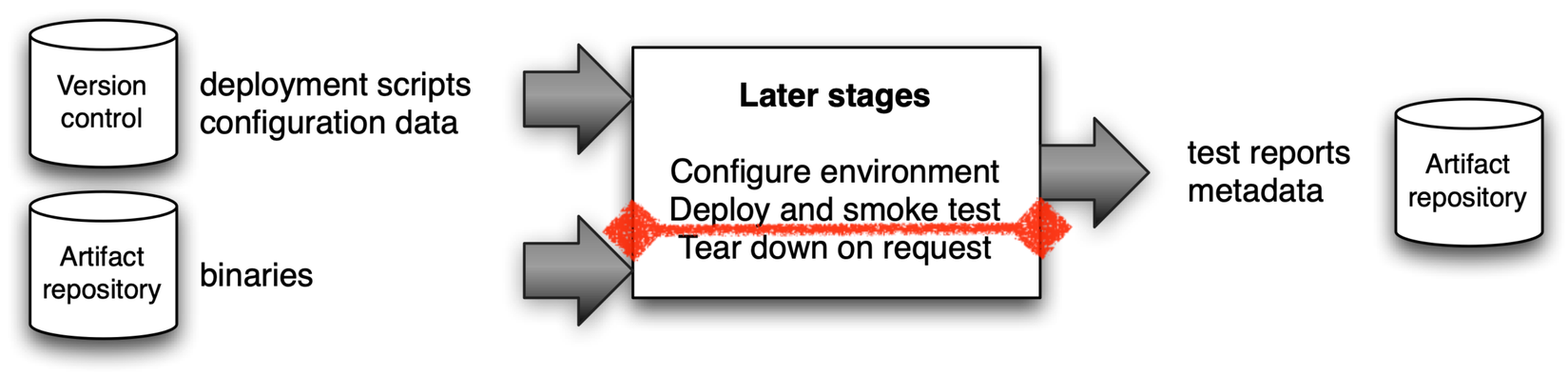
- UAT, staging, integration, production, …

Manual Stage

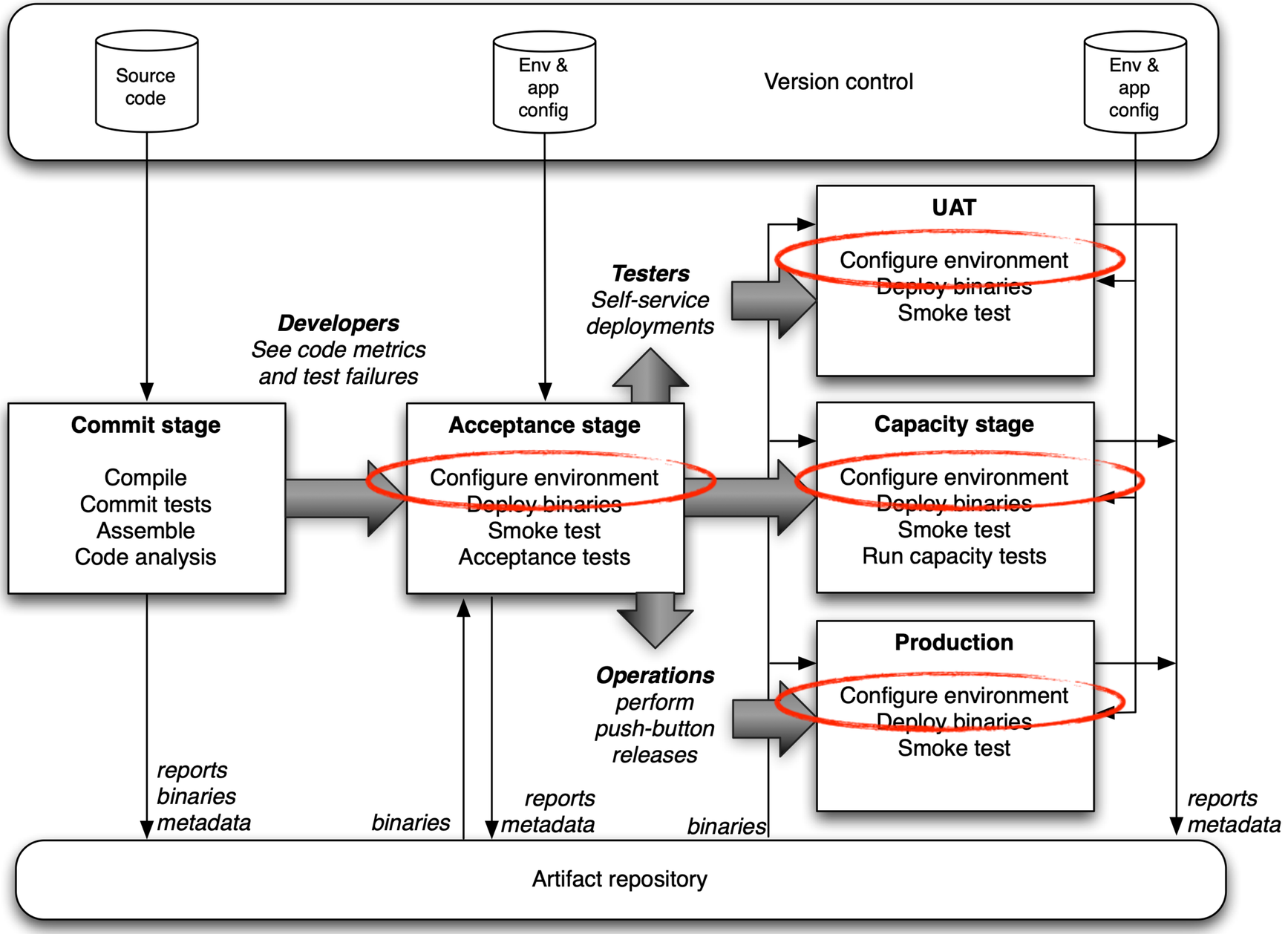


- UAT, staging, integration, production, …
- Push versus Pull model

Manual Stage

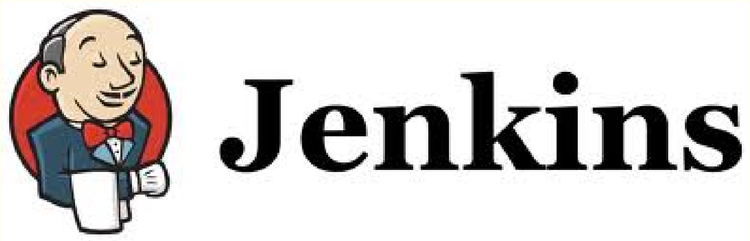


- UAT, staging, integration, production, …
- Push versus Pull model
- Deployments self-serviced through push-button process

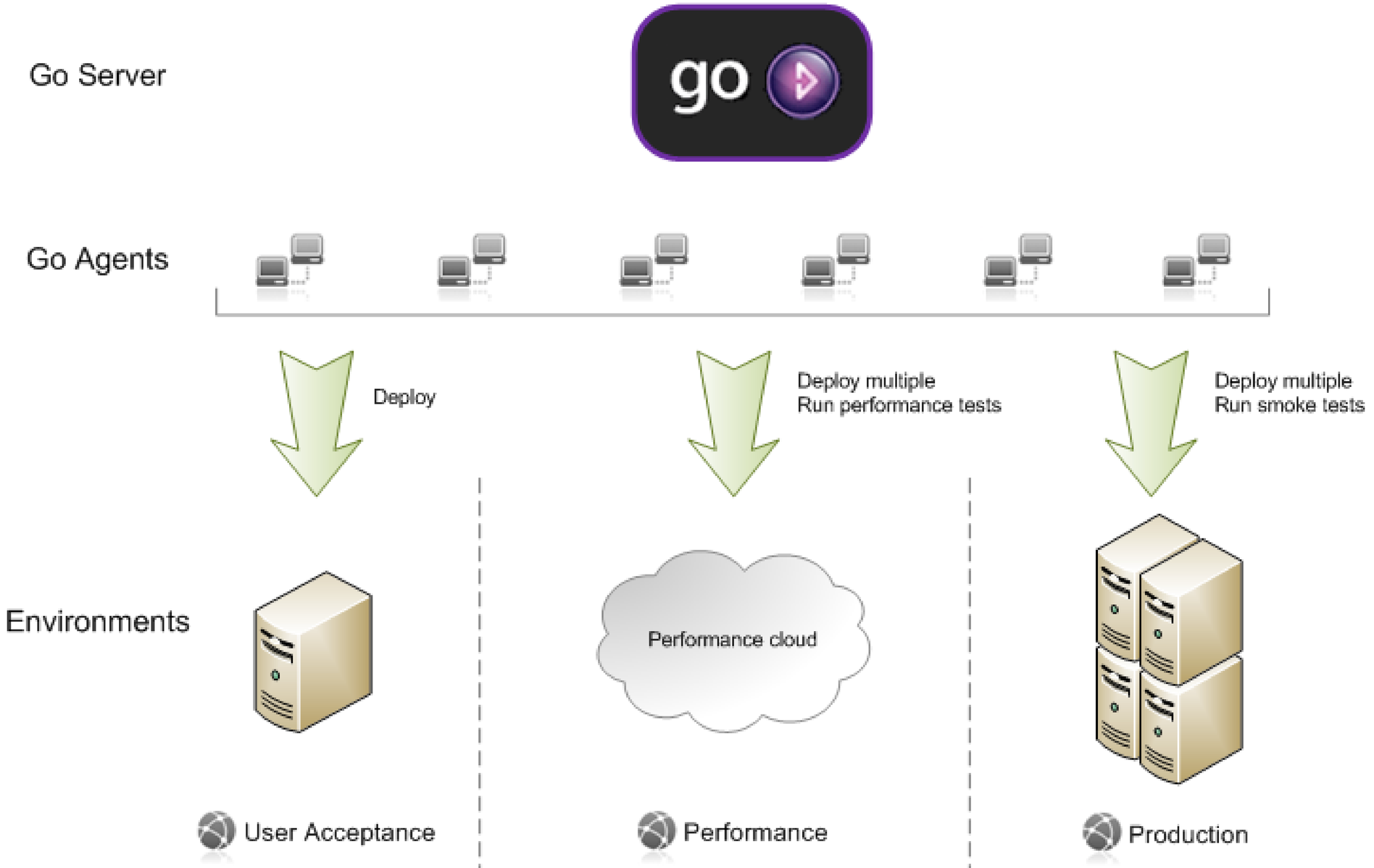

Source code
Env & app config
Version control
Env & app config
UAT
Configure environment
Deploy binaries
Smoke test
Testers
Self-service deployments
Developers
See code metrics and test failures
Commit stage
Compile
Commit tests
Assemble
Code analysis
Acceptance stage
Configure environment
Deploy binaries
Smoke test
Acceptance tests
Capacity stage
Configure environment
Deploy binaries
Smoke test
Run capacity tests
Production
Configure environment
Deploy binaries
Smoke test
Operations
perform push-button releases
reports
binaries
metadata
binaries
reports
metadata
binaries
reports
metadata
Artifact repository

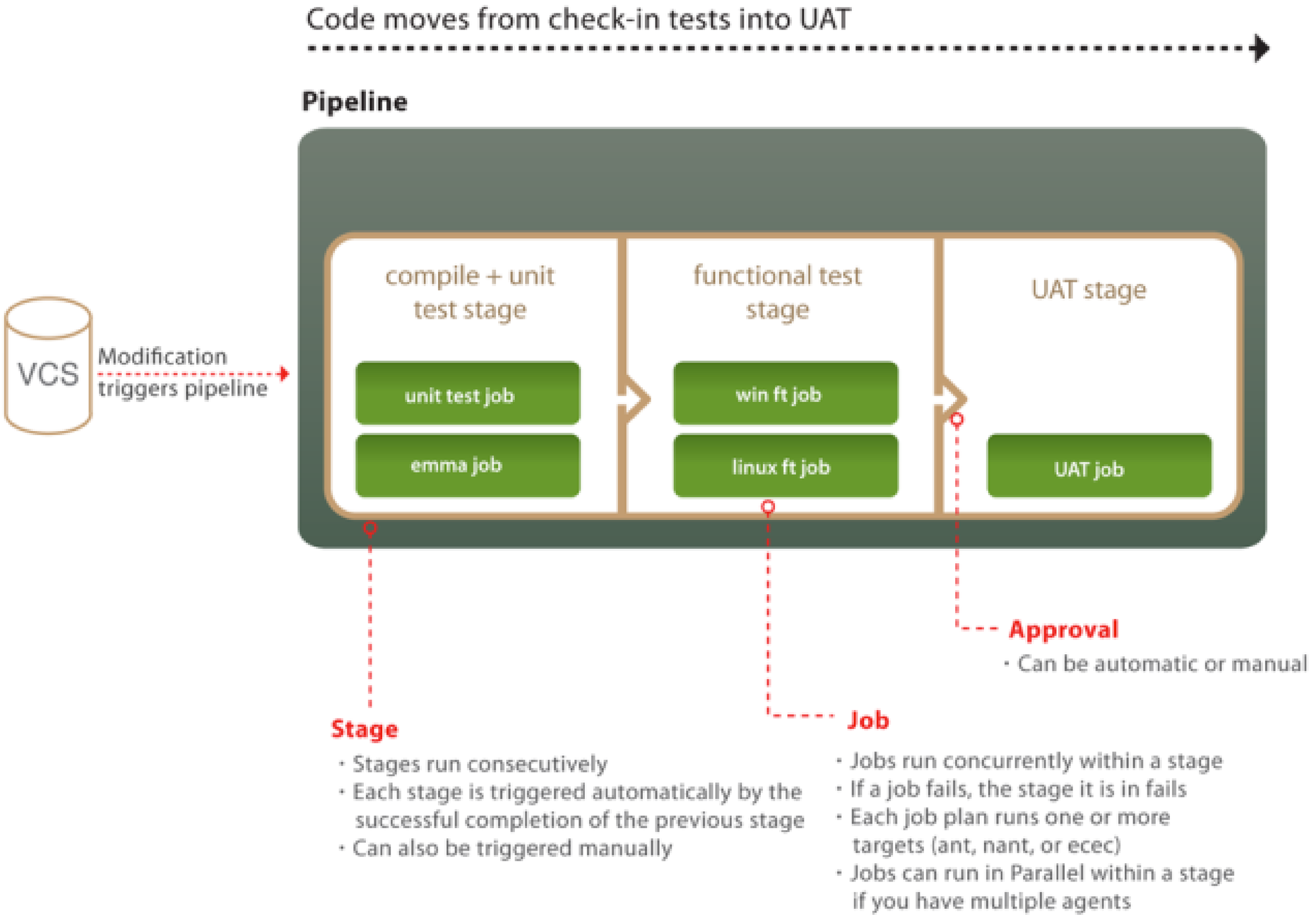
Machinery

continuous integration ++

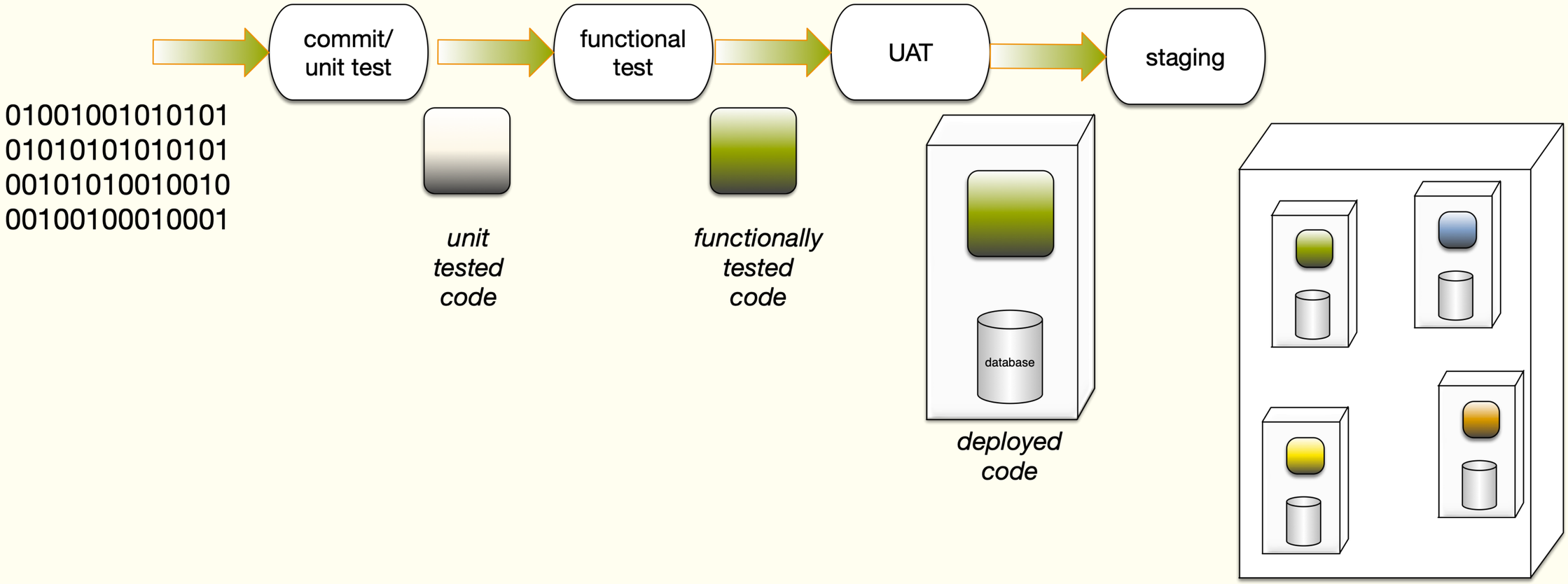


https://www.gocd.org

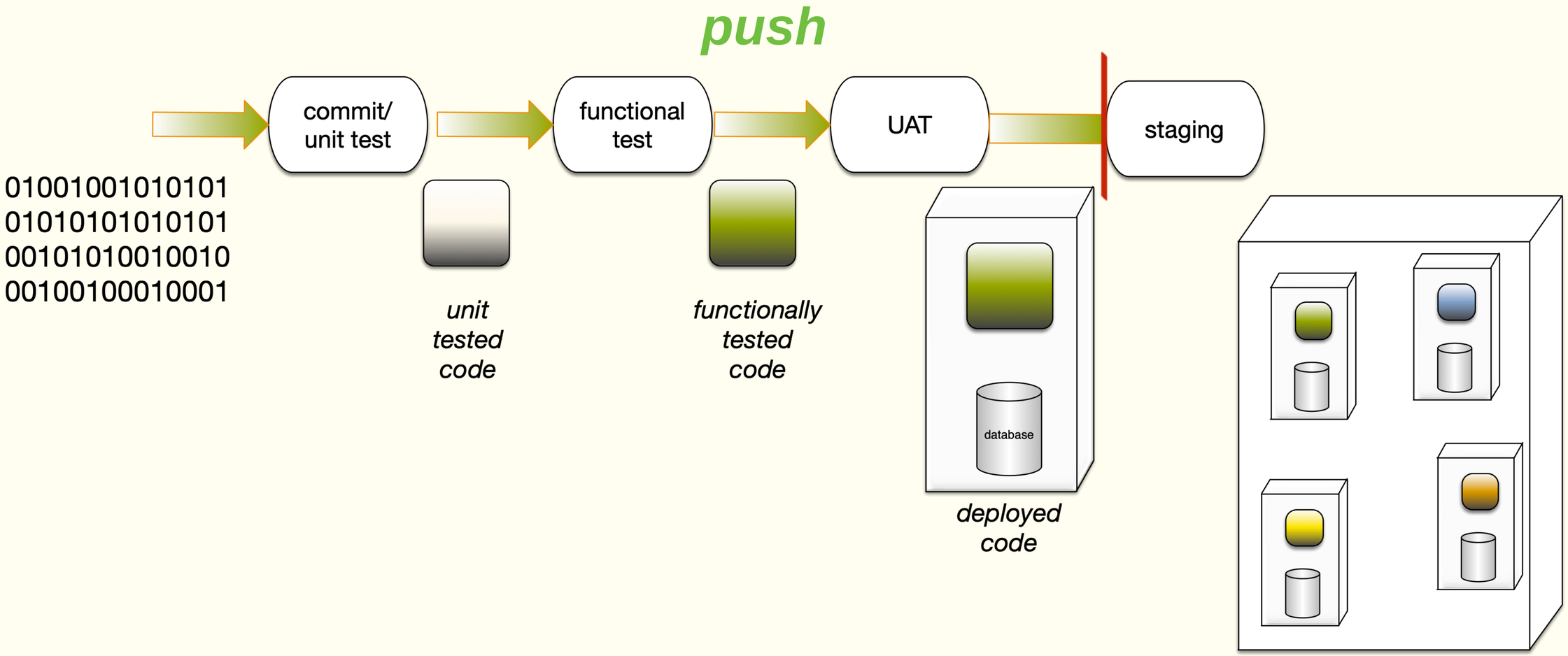

Go Server
go
Go Agents
Deploy
Deploy multiple
Run performance tests
Deploy multiple
Run smoke tests
Environments
Performance cloud
User Acceptance
Performance
Production


Code moves from check-in tests into UAT
Pipeline
VCS
Modification
triggers pipeline
compile + unit
test stage
functional test
stage
UAT stage
unit test job
emma job
win ft job
linux ft job
UAT job
Approval
• Can be automatic or manual
Stage
• Stages run consecutively
• Each stage is triggered automatically by the successful completion of the previous stage
• Can also be triggered manually
Job
• Jobs run concurrently within a stage
• If a job fails, the stage it is in fails
• Each job plan runs one or more targets (ant, nant, or ecec)
• Jobs can run in Parallel within a stage if you have multiple agents

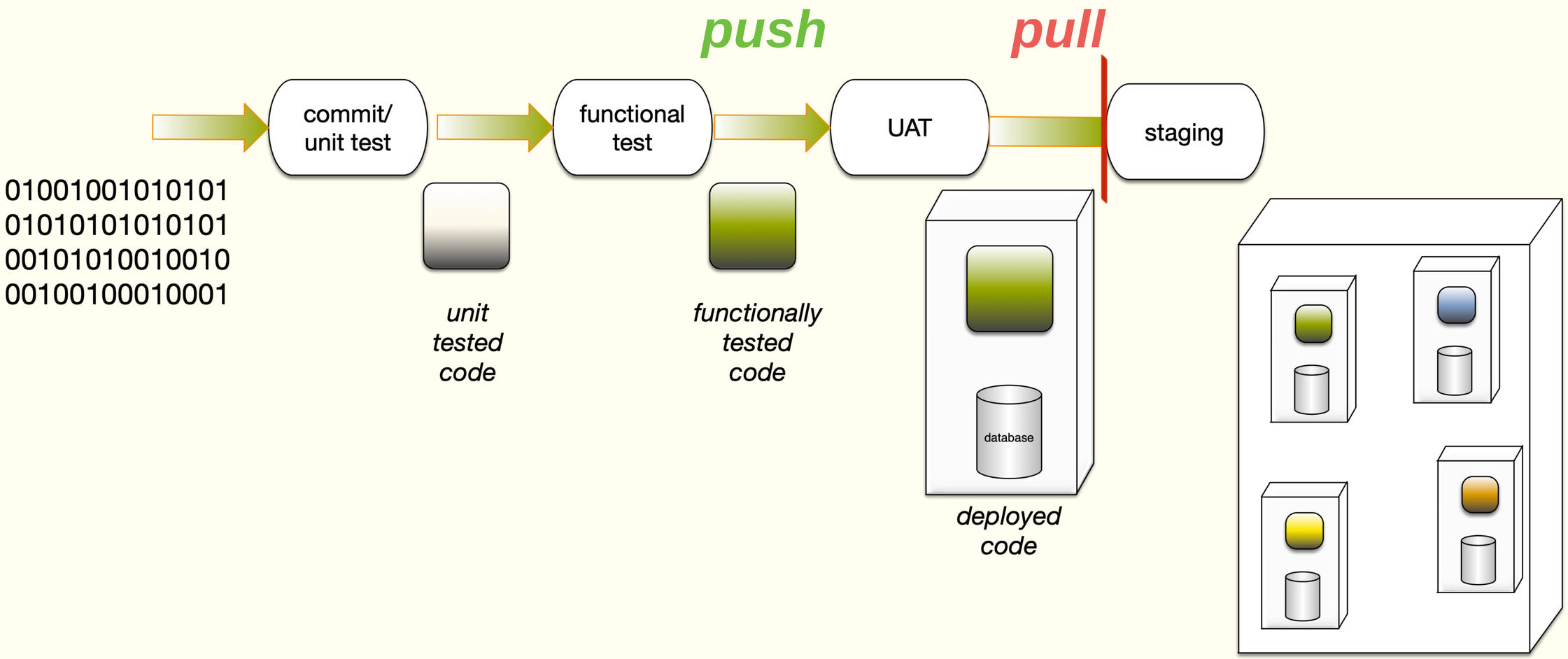
Deployment Pipeline

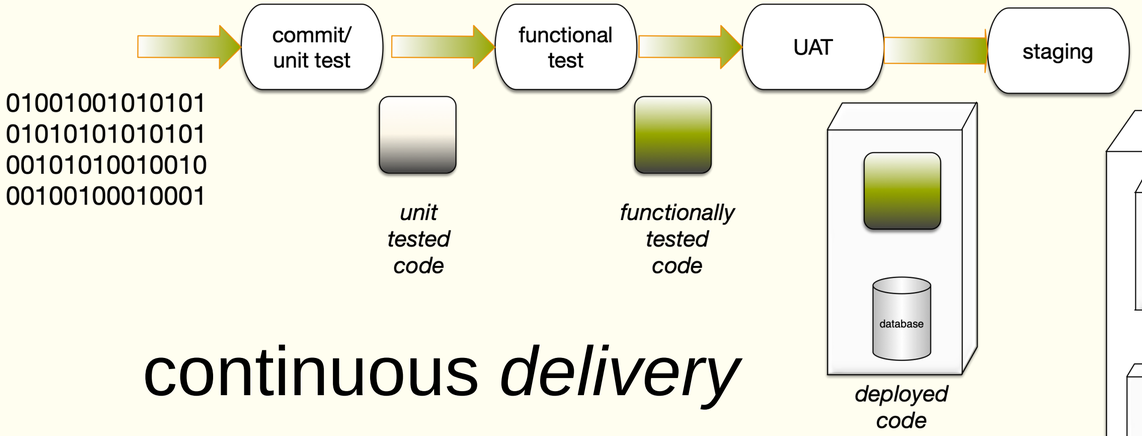


Deployment vs Delivery?



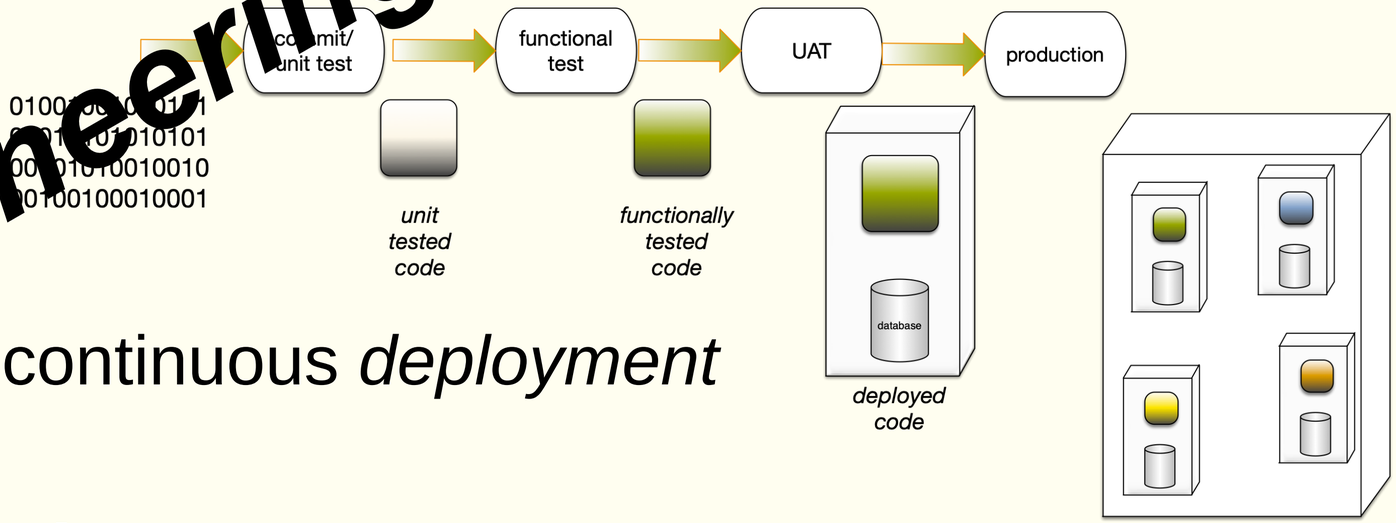
Deployment vs Delivery?





continuous *delivery*

same engineering practices

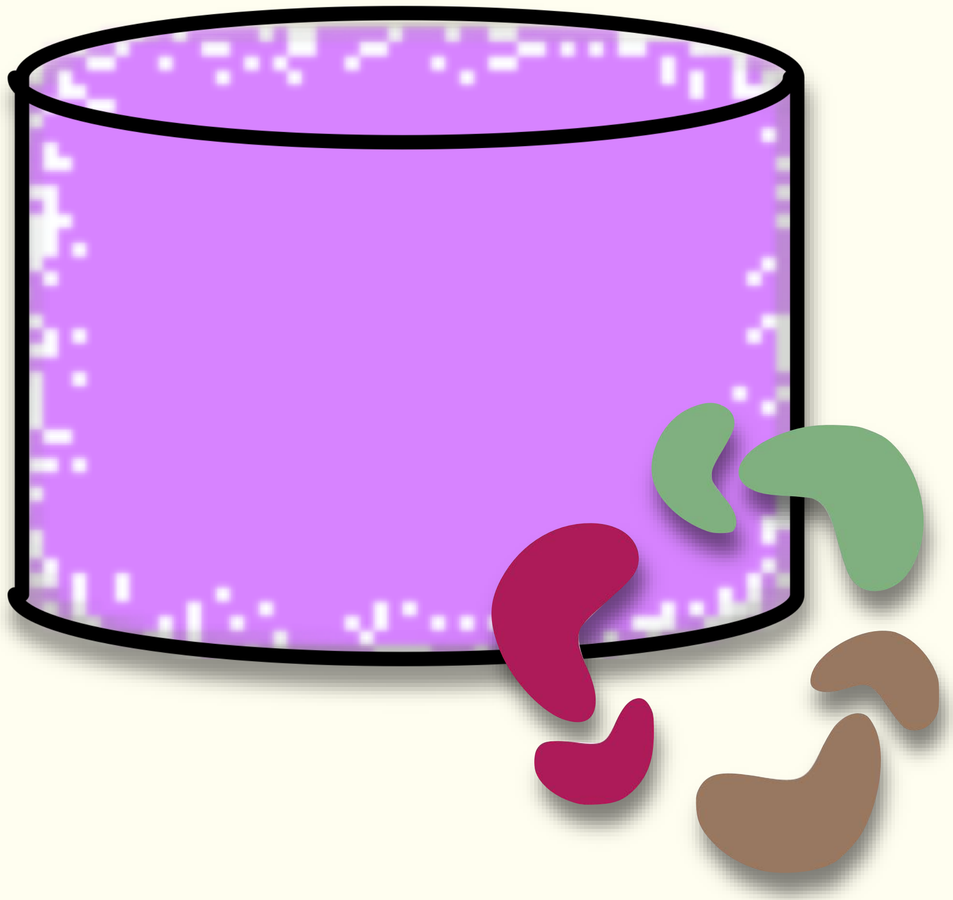


continuous *deployment*

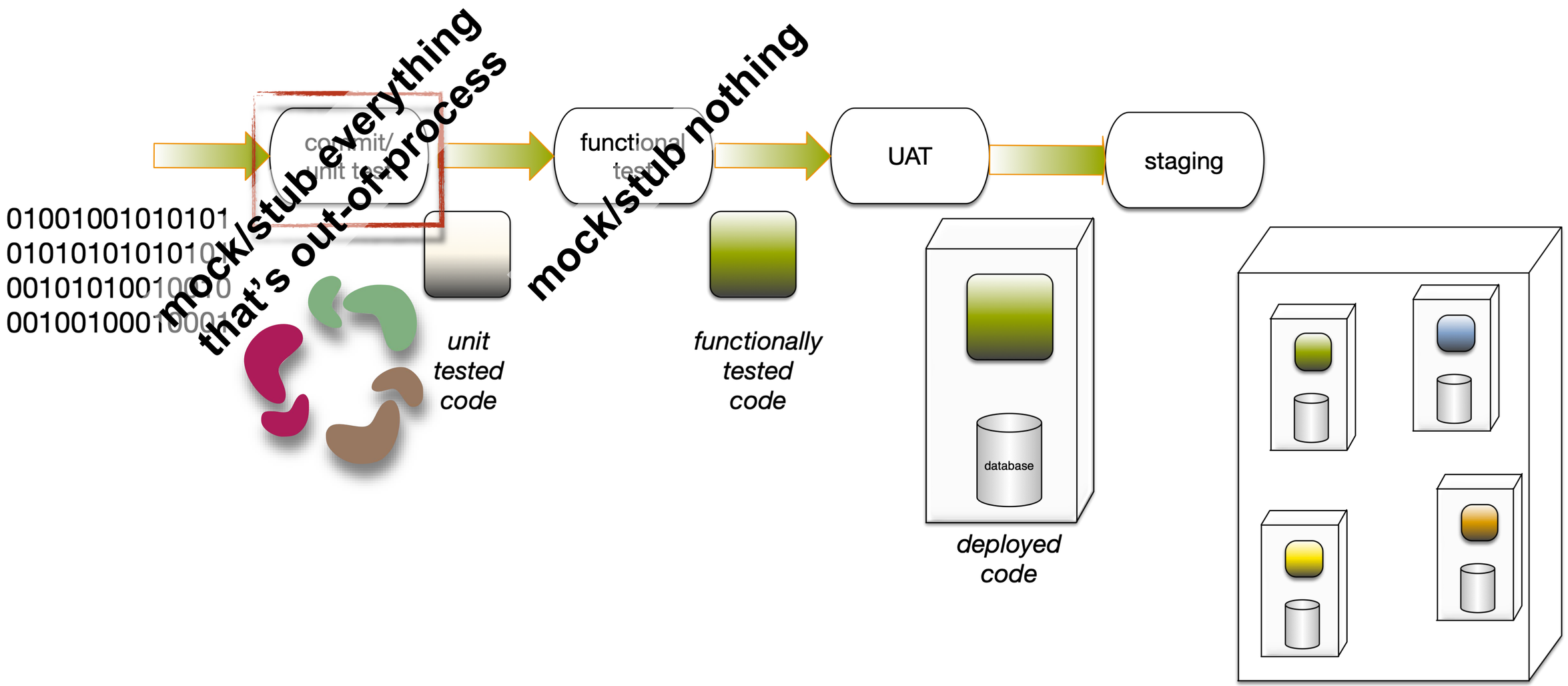
Continuous *deployment*: all ***push*** stages

Continuous *delivery*: at least one ***pull*** stage

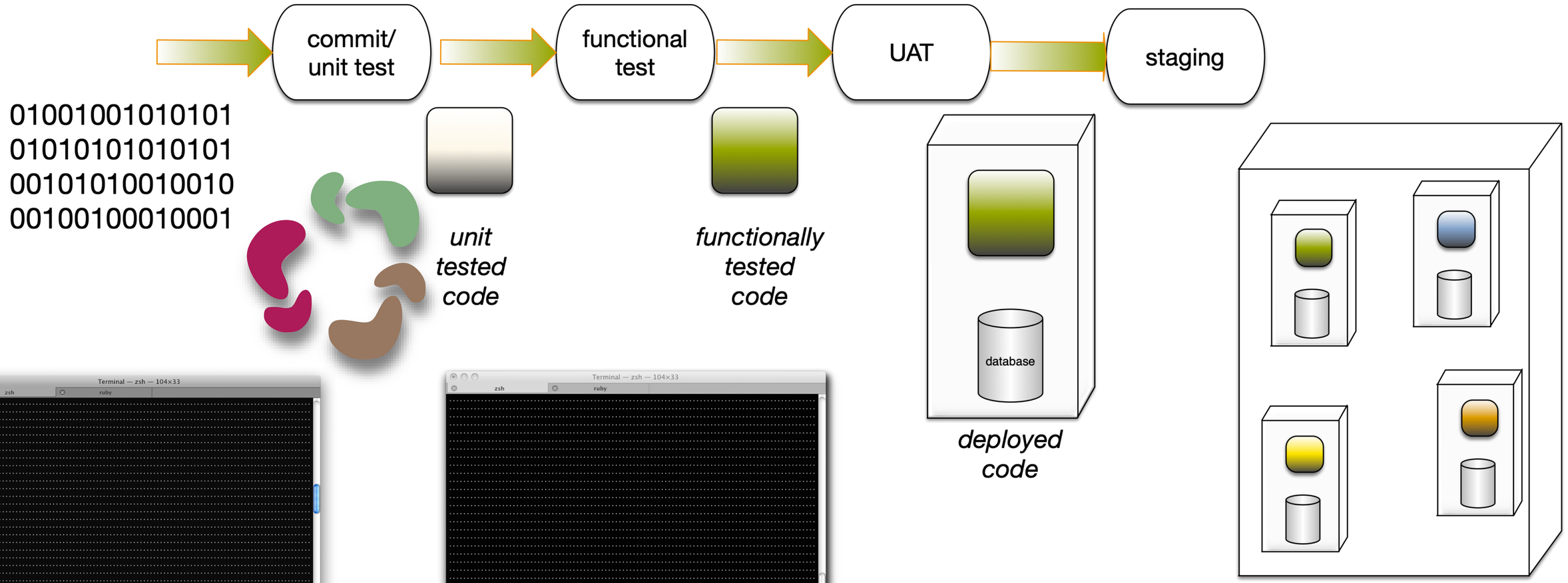
Continuous Integration for Databases

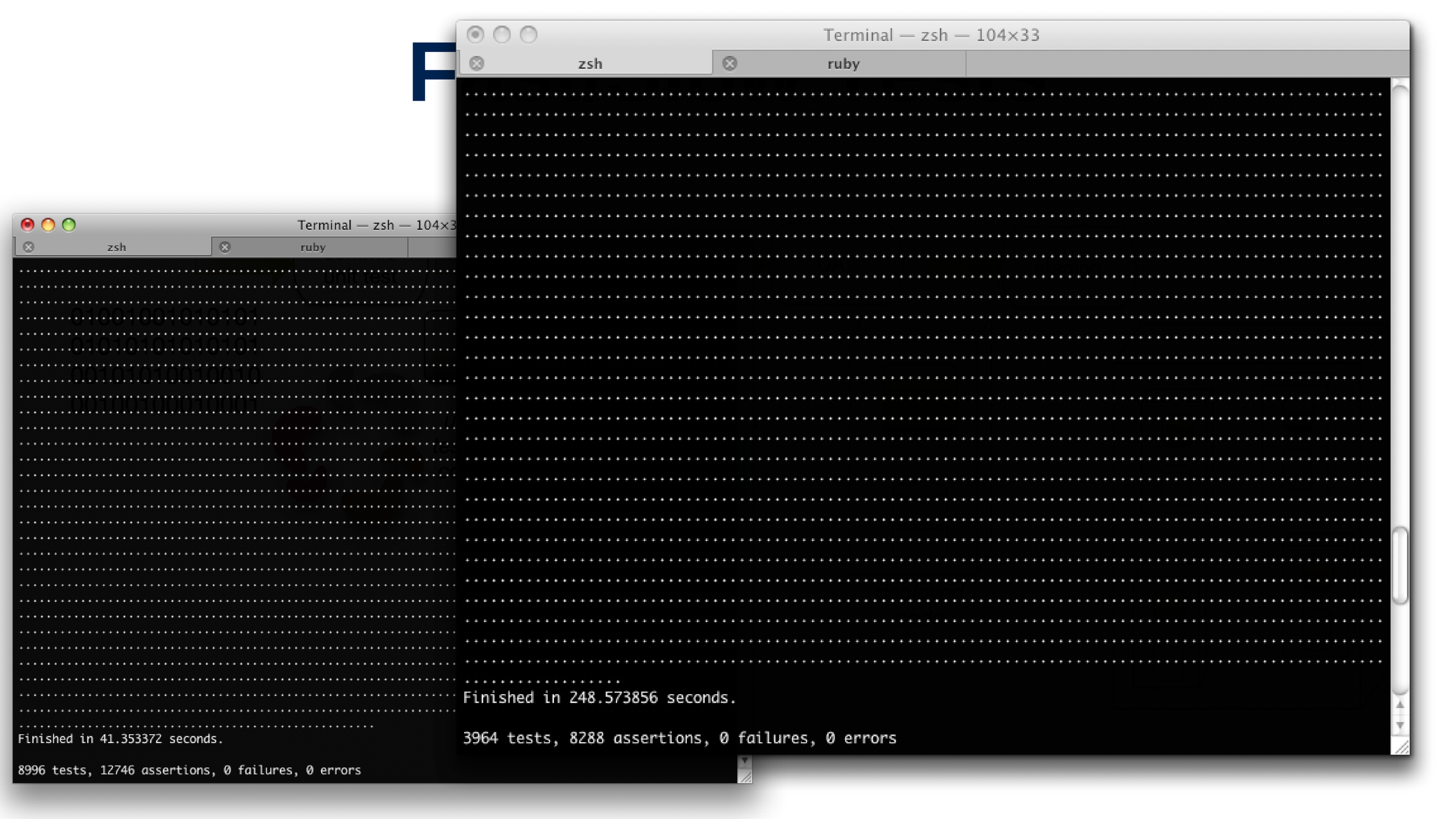


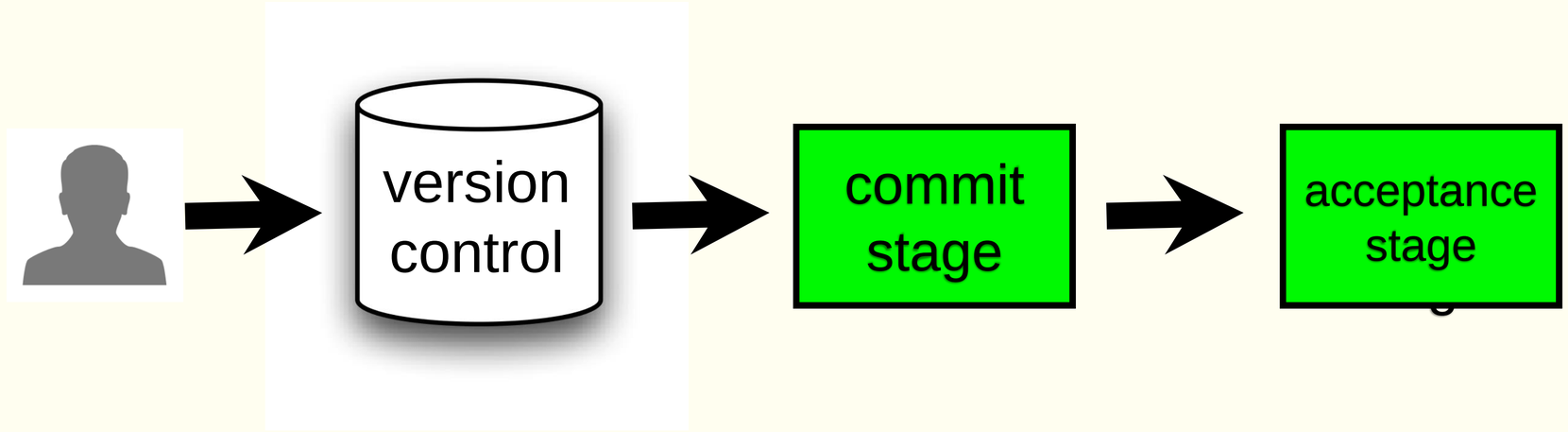
Fast/Slow Tests

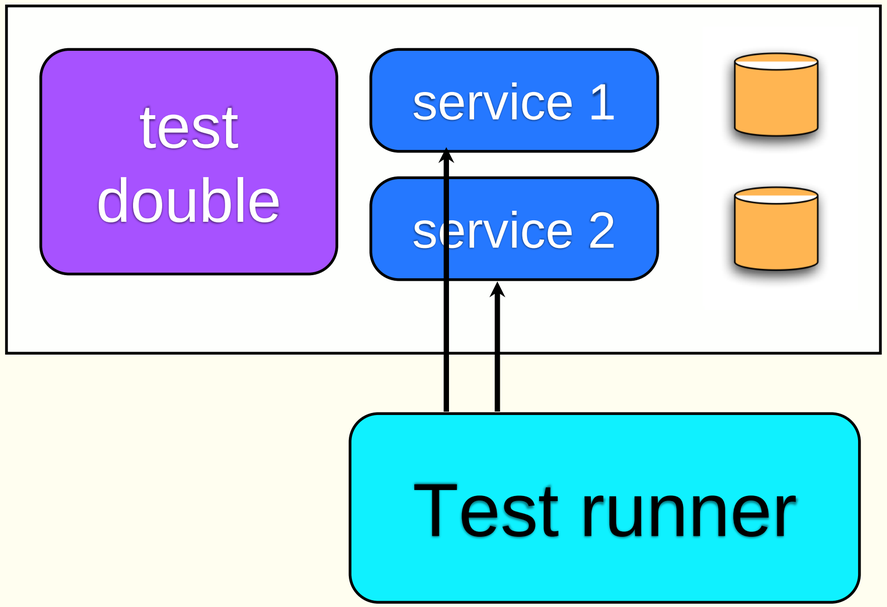


Fast/Slow Tests




Terminal — zsh — 104×33
zsh
ruby
Finished in 248.573856 seconds.
3964 tests, 8288 assertions, 0 failures, 0 errors
Terminal — zsh — 104×3
zsh
ruby
Finished in 41.353372 seconds.
8996 tests, 12746 assertions, 0 failures, 0 errors





Prepare environment

Deploy app

Create dbs, apply schema

Add app reference data

Run acceptance tests

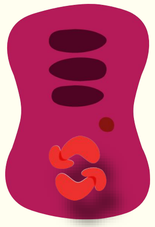
integration as an engineering practice over time

continuous integration

continuous deployment

continuous delivery

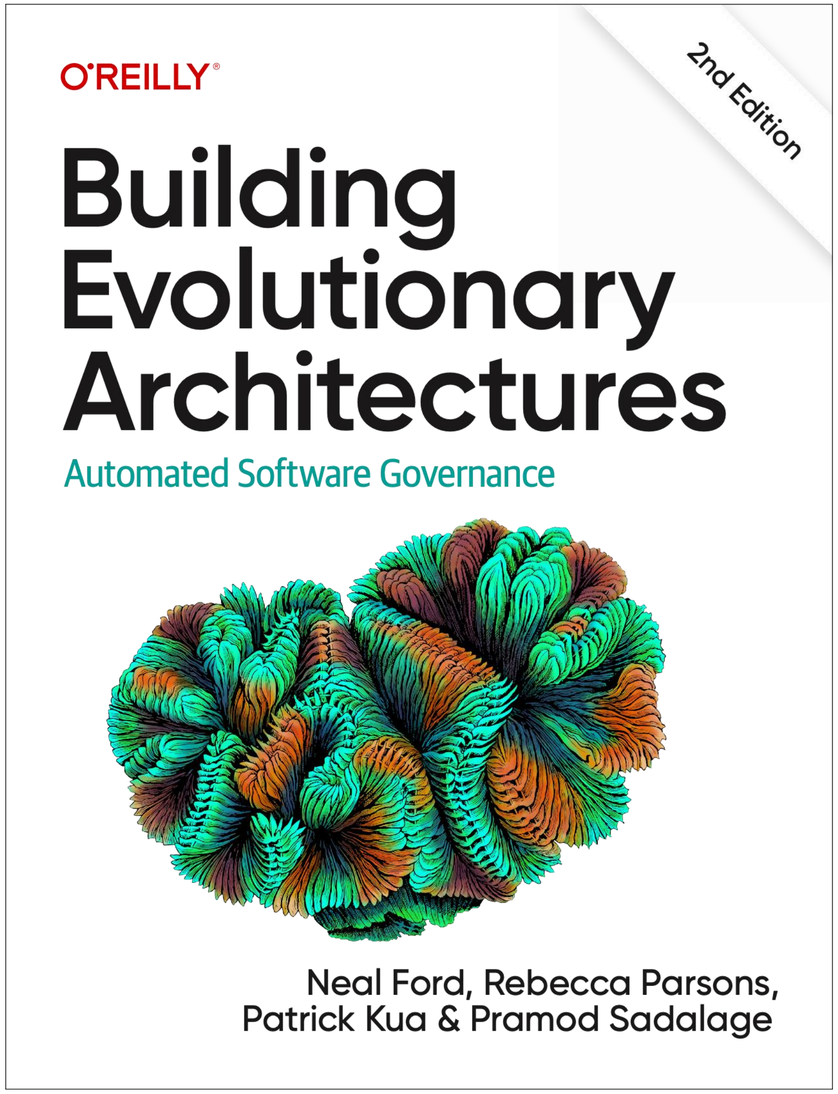
1980 1990 2000 2010 current day



aRChiTeCtuRe
fOr
cONtiNuoUs DeLiVeRy

Question:

How can we combine things like metrics and other architecture checks with continuous delivery/deployment?



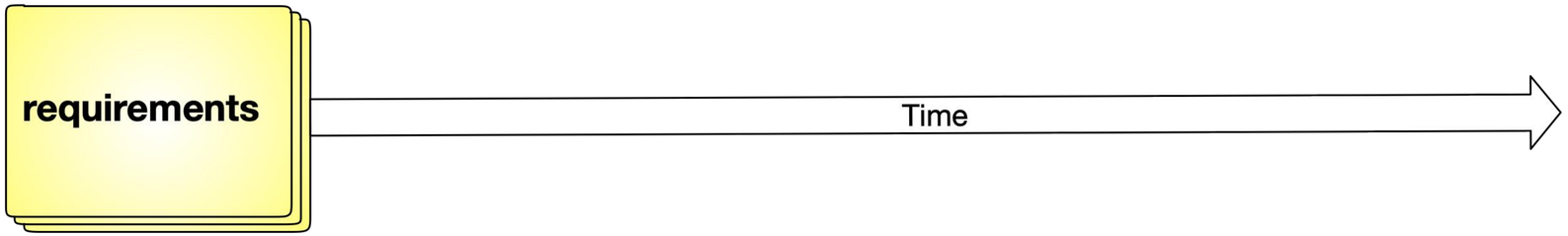
— fitness functions

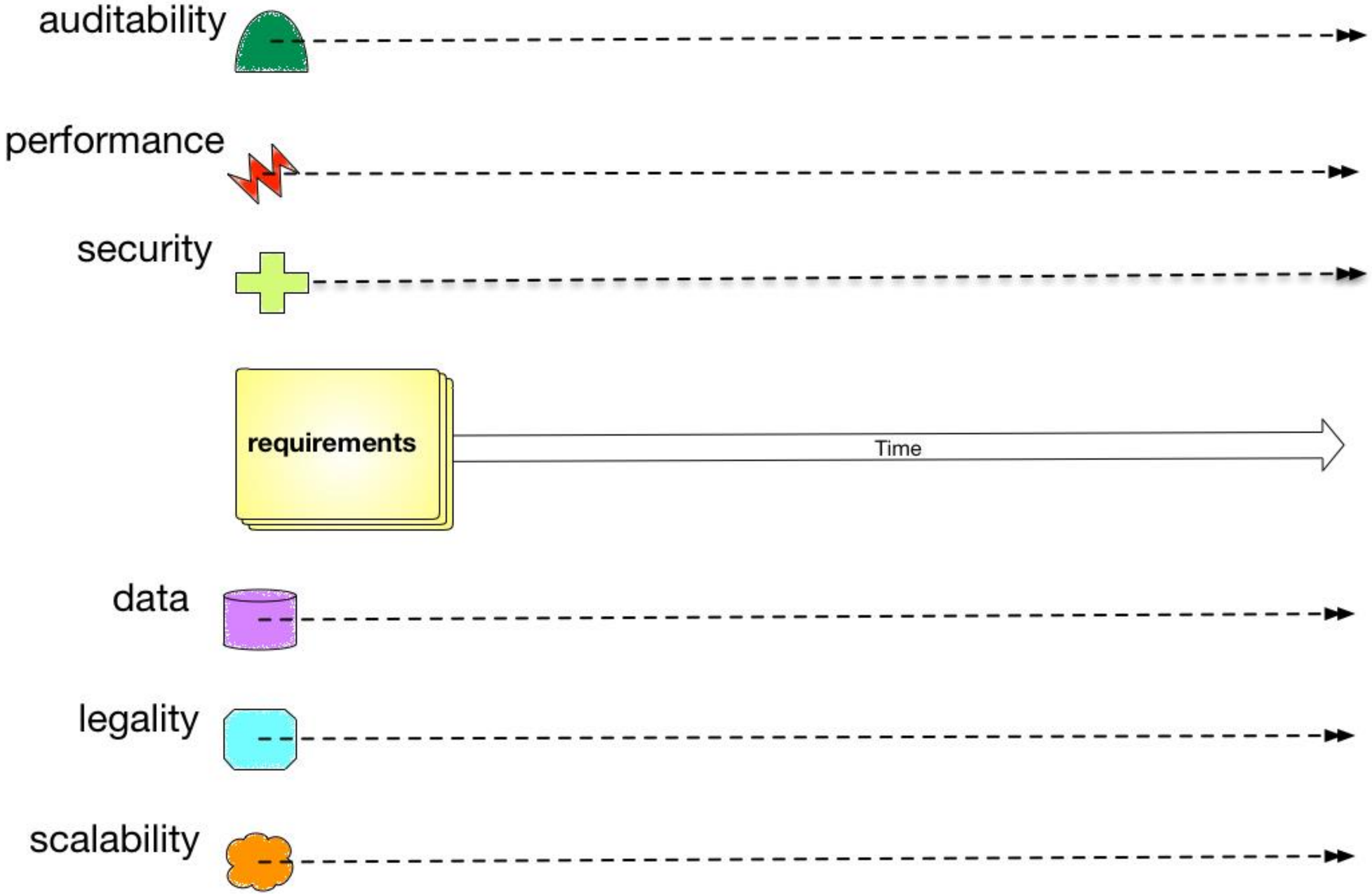
— incremental change

@neal4d
nealford.com

@rebeccaparsons

@patkua


requirements
Time


auditability
performance
security
requirements
Time
data
legality
scalability

ility

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

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

accessibility
accountability
accuracy
adaptability
administrability
affordability
agility
auditability
autonomy

reliability
extensibility
failure transparency
fault-tolerance
fidelity
flexibility
inspectability
installability
integrity

repeatability
reproducibility
resilience
responsiveness
reusability
robustness
safety
scalability
seamlessness

evolvability

debugability
degradability
determinability
demonstrability
dependability
deployability
discoverability
distributability
durability
effectiveness
efficiency

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

survivability
sustainability
tailorability
testability
timeliness
traceability
transparency
ubiquity
understandability
upgradability
usability

accessibility
accountability
accuracy
adaptability
administrability
affordability
agility
auditability
autonomy

reliability
extensibility
failure transparency
fault-tolerance
fidelity
flexibility
inspectability
installability
integrity

repeatability
reproducibility
resilience
responsiveness
reusability
robustness
safety
scalability
seamlessness

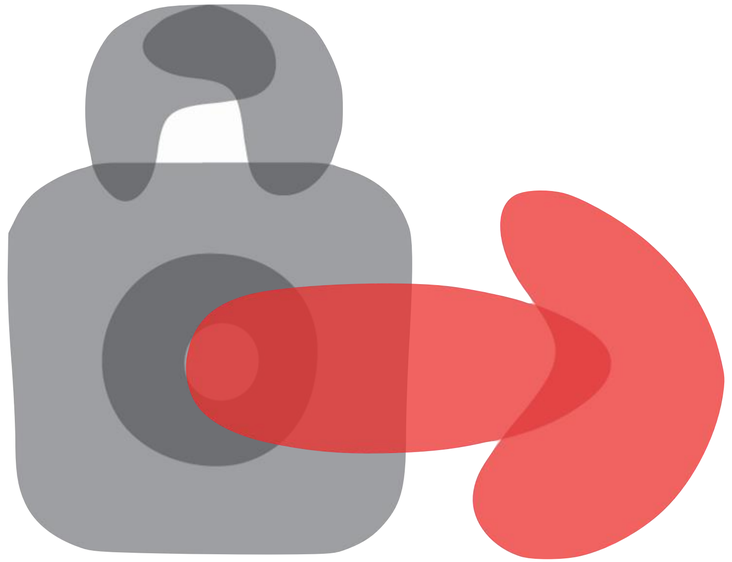
evolvability

debugability
degradability
determinability
demonstrability
dependability
deployability
discoverability
distributability
durability
effectiveness
efficiency

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

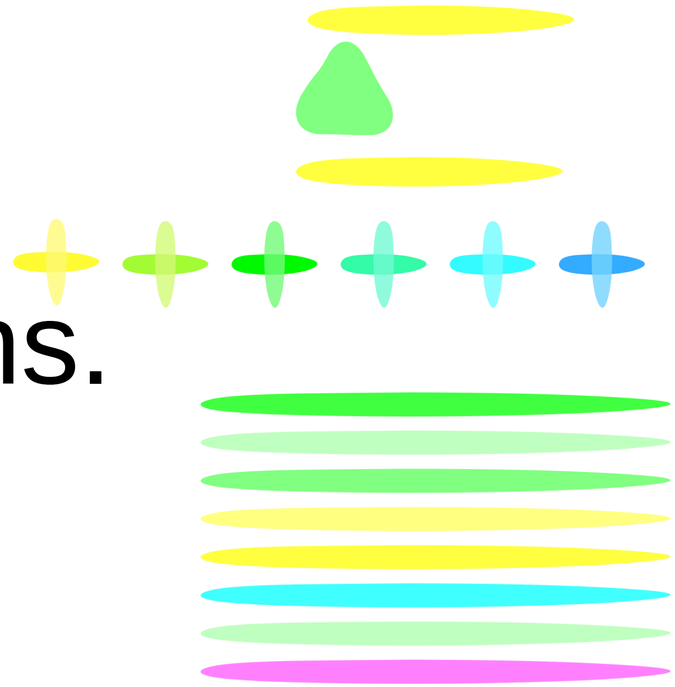
survivability
sustainability
tailorability
testability
timeliness
traceability
transparency
ubiquity
understandability
upgradability
usability

governance



Evolutionary Architecture

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



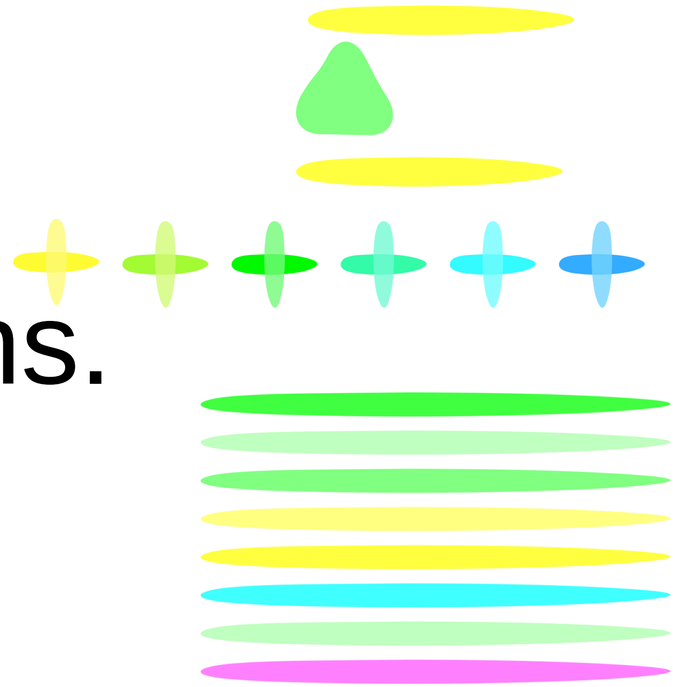
Evolutionary Architecture

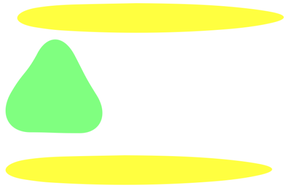
An evolutionary architecture supports

guided

incremental change

across multiple dimensions.



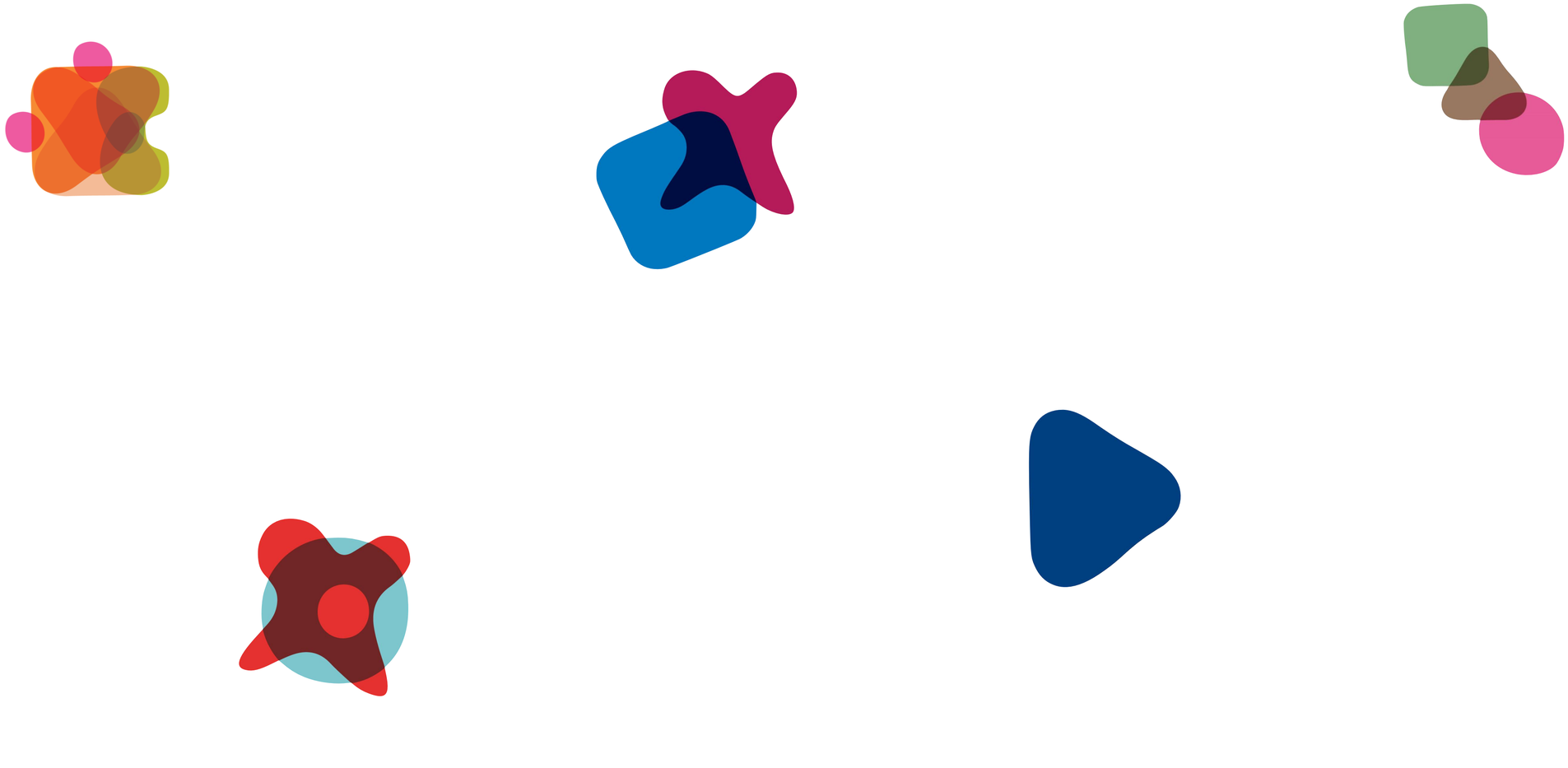


guided

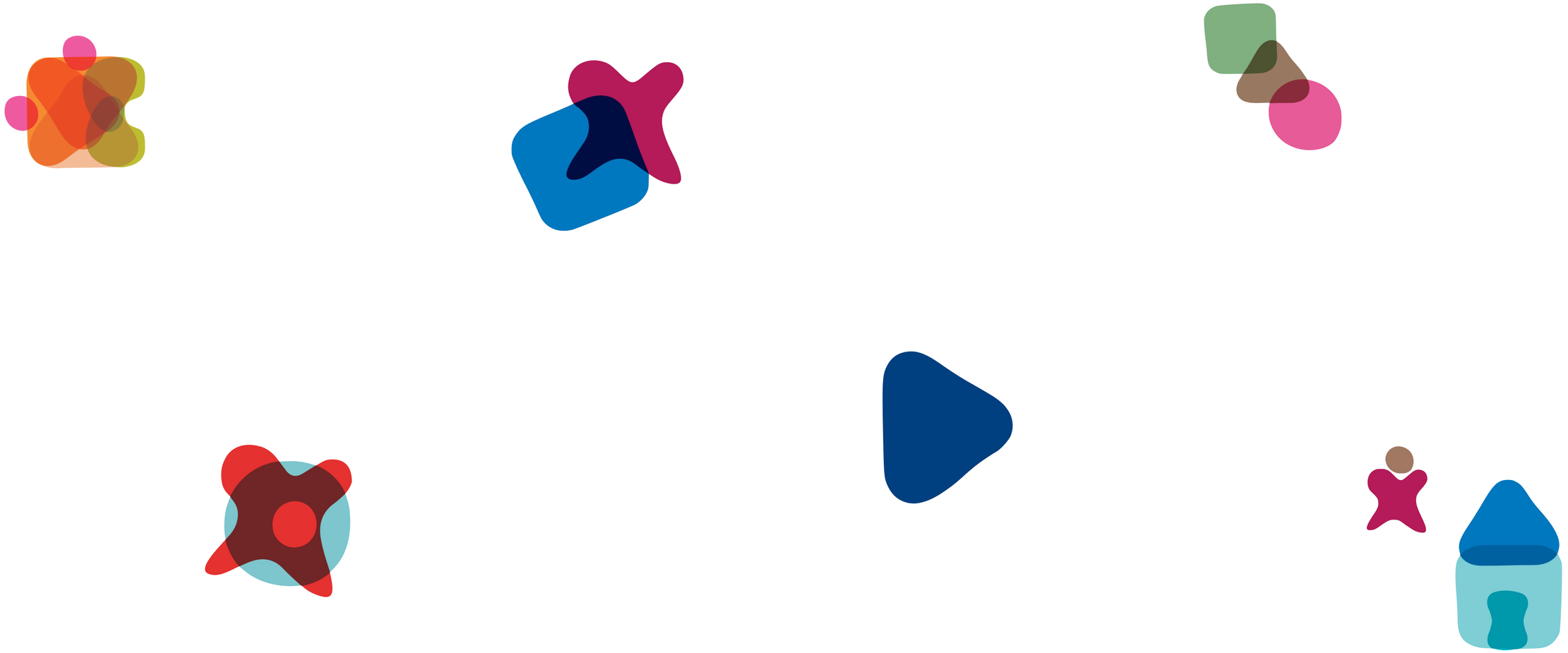
evolutionary computing fitness function:

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

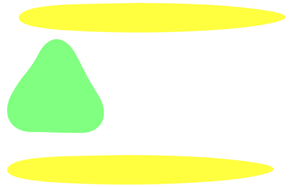
Traveling Salesperson Problem



Traveling Salesperson Problem



fitness function = length of route



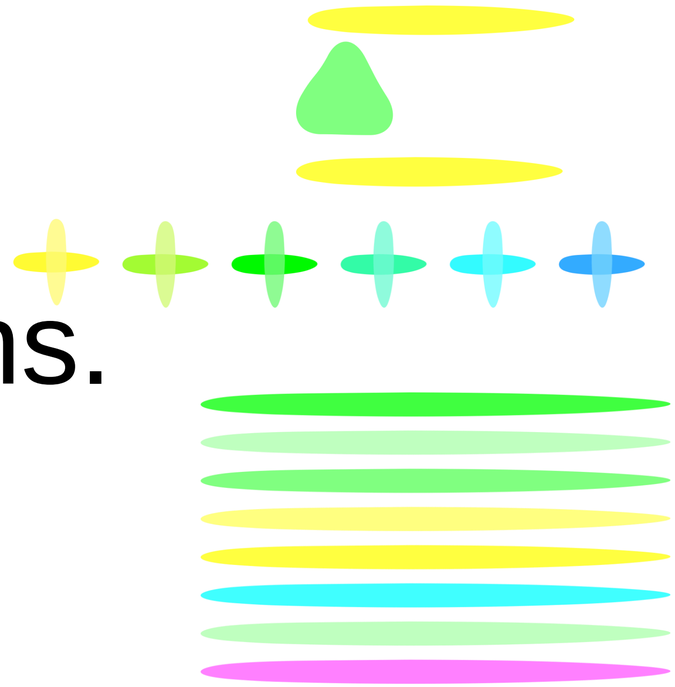
guided

architectural *fitness function*:

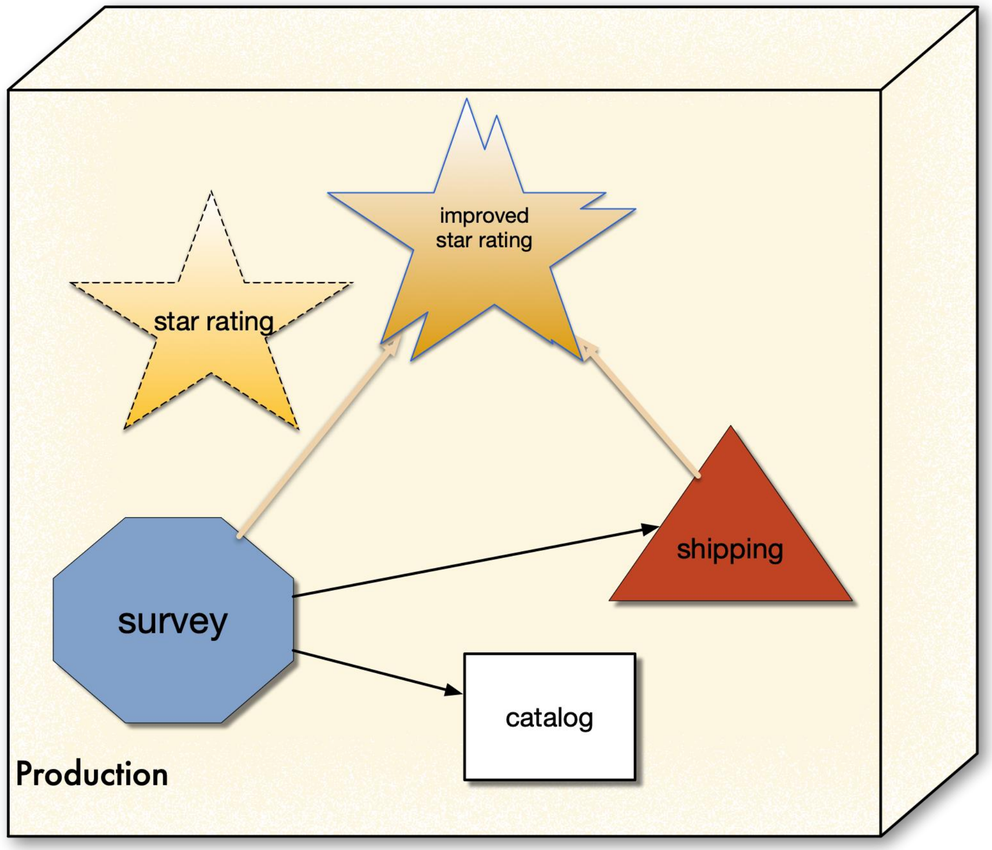
An architectural fitness function provides an objective integrity assessment of some architectural characteristic(s).

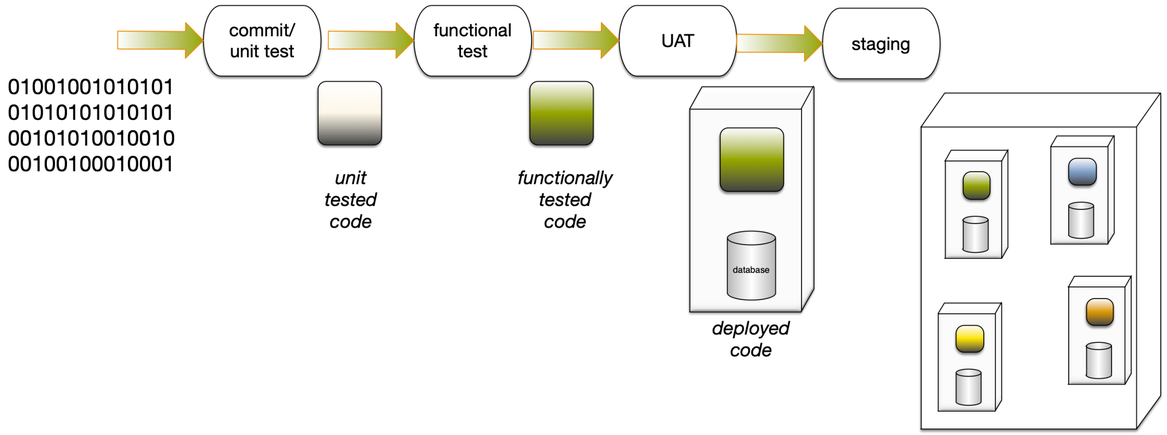
Evolutionary Architecture

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



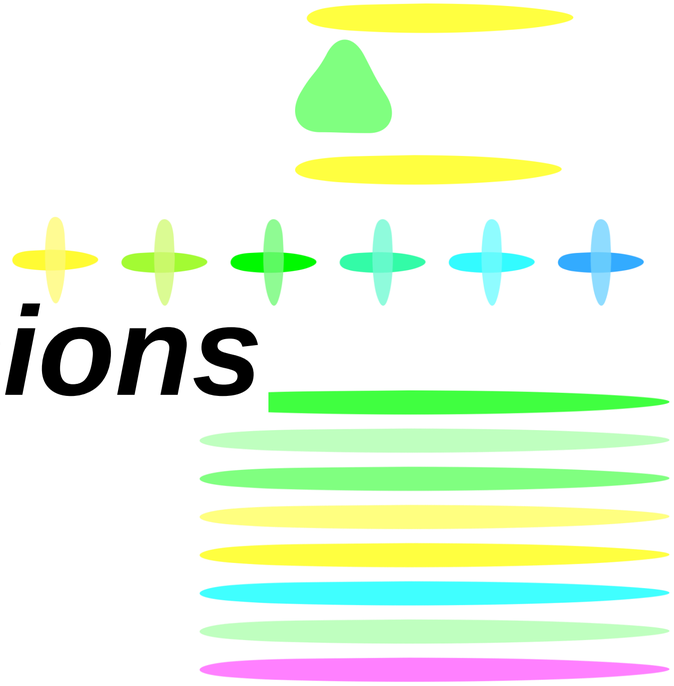
incremental



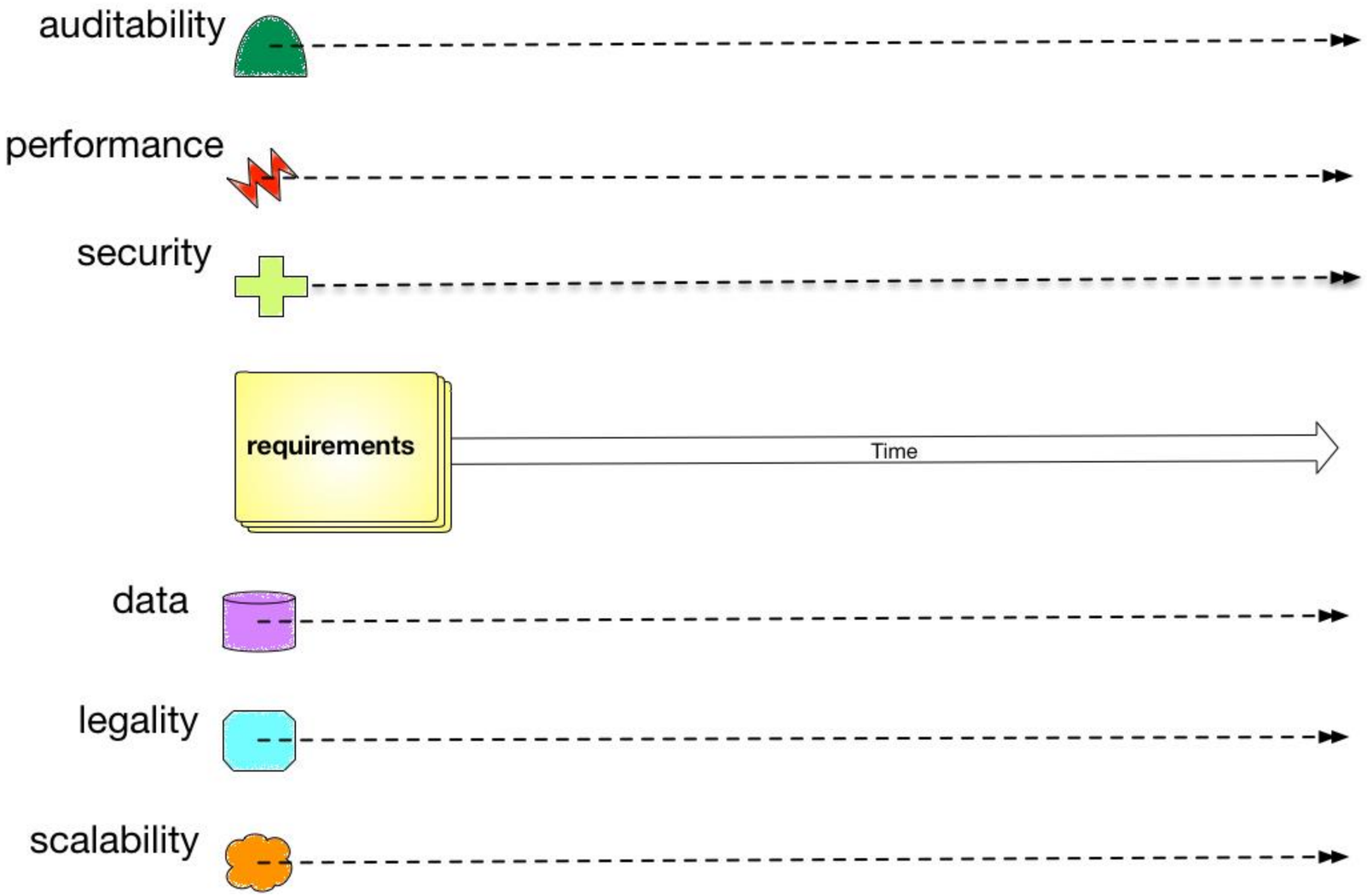


Evolutionary Architecture

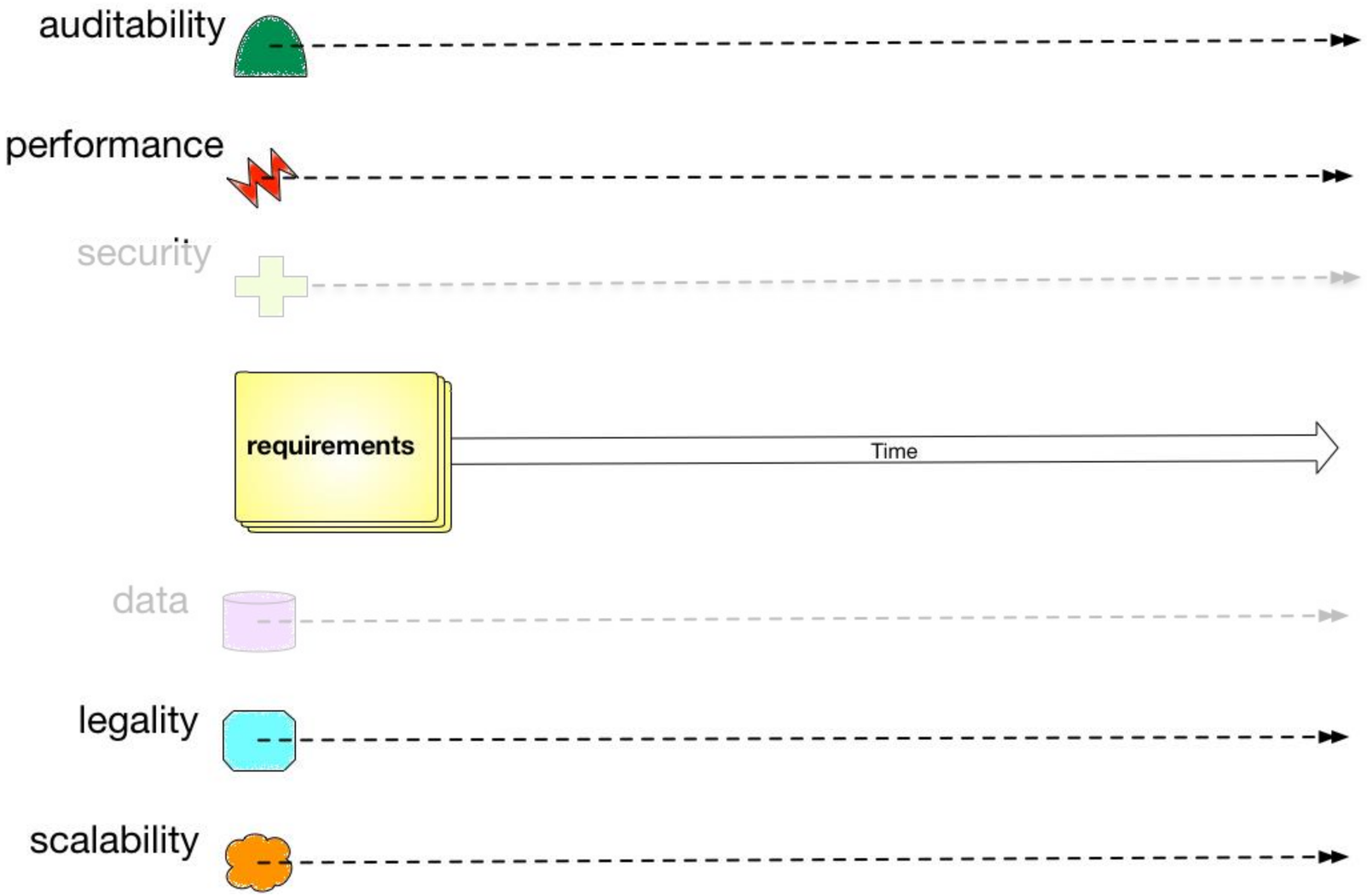
An evolutionary architecture supports
guided,
incremental change
across ***multiple dimensions***



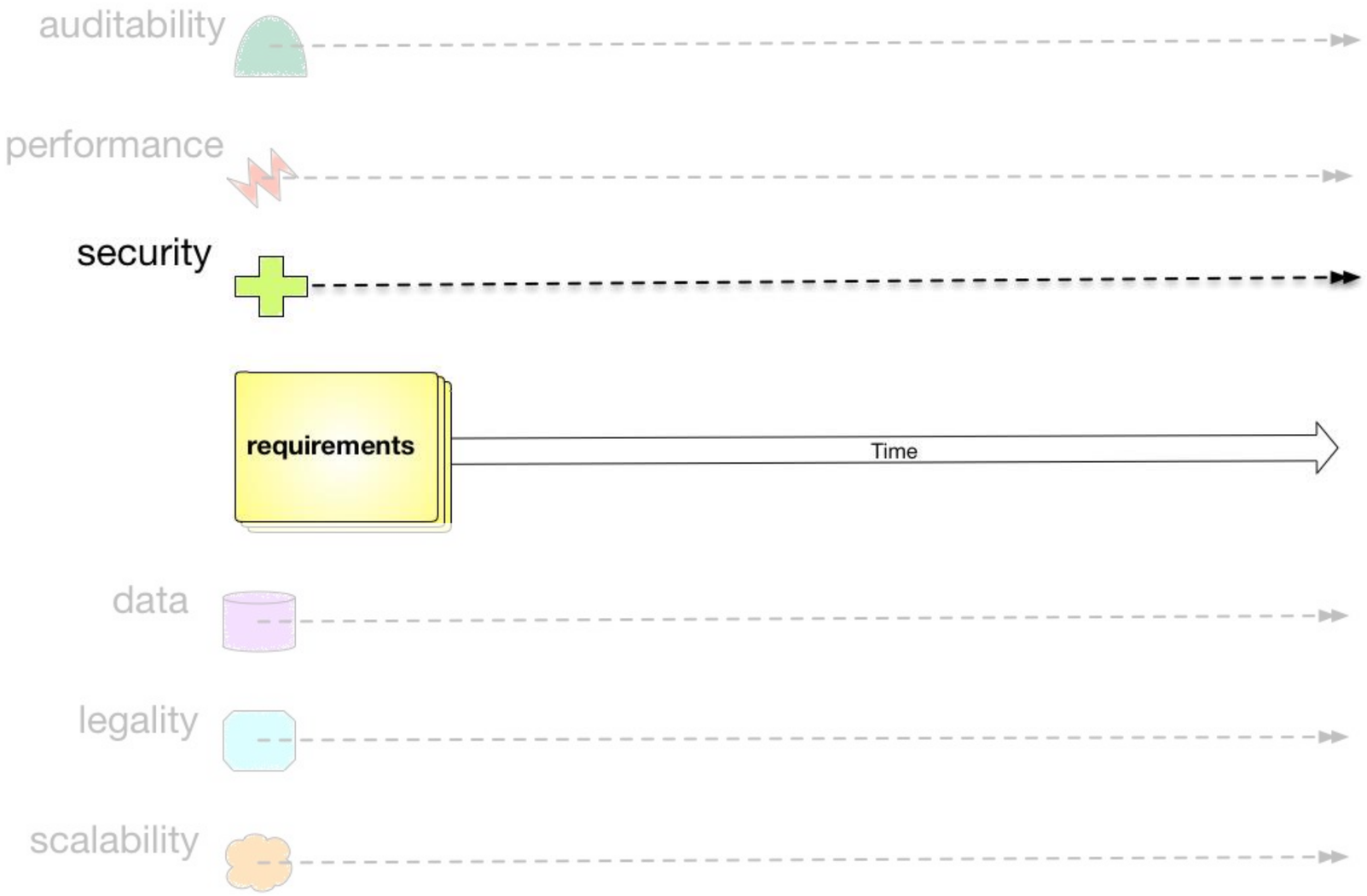
multiple dimensions



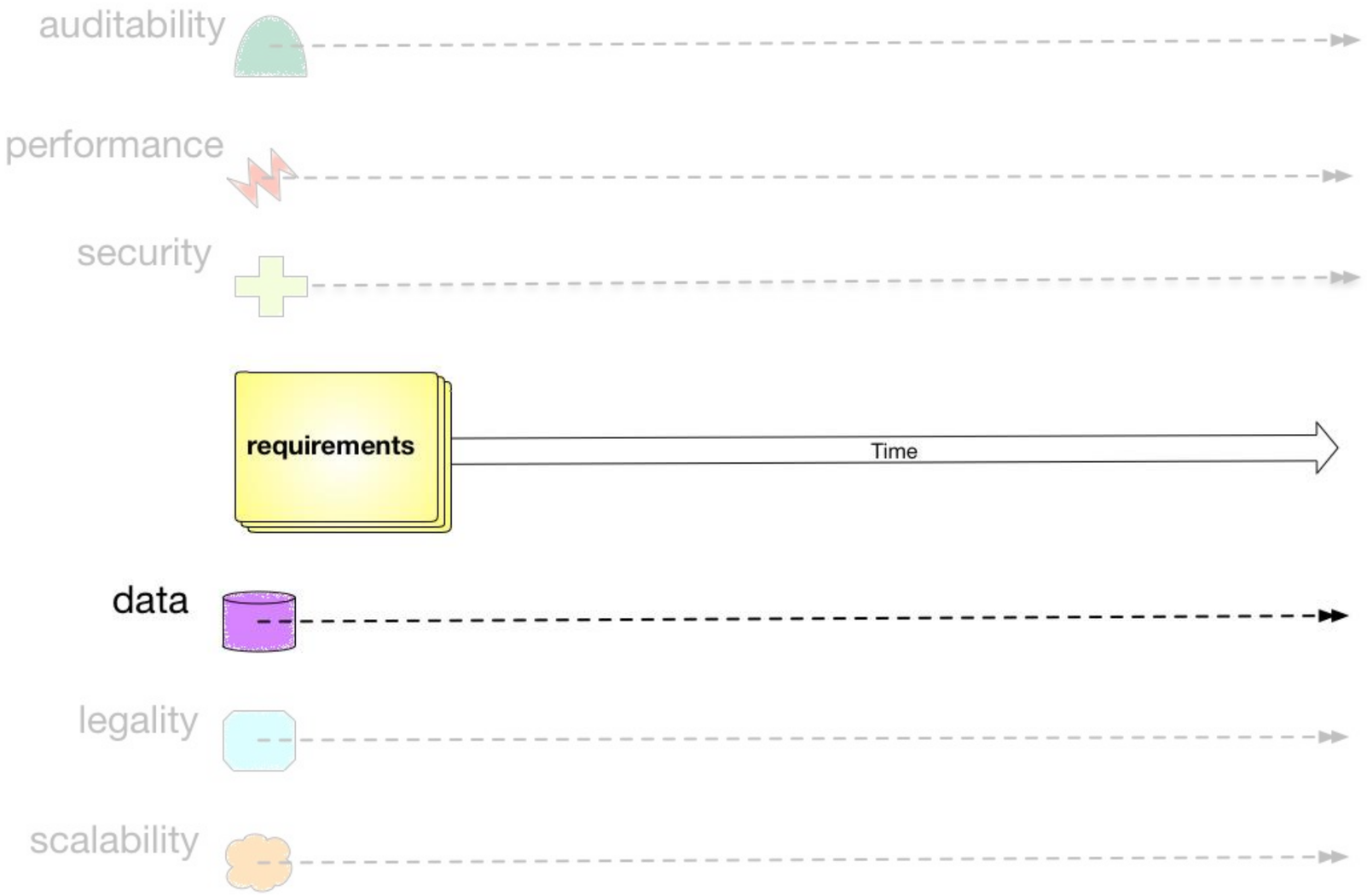
multiple dimensions



multiple dimensions

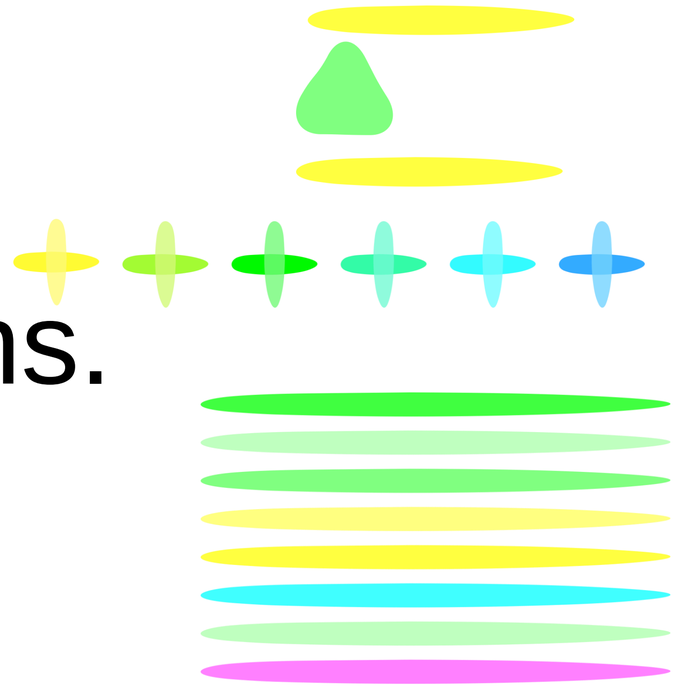


multiple dimensions



Evolutionary Architecture

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



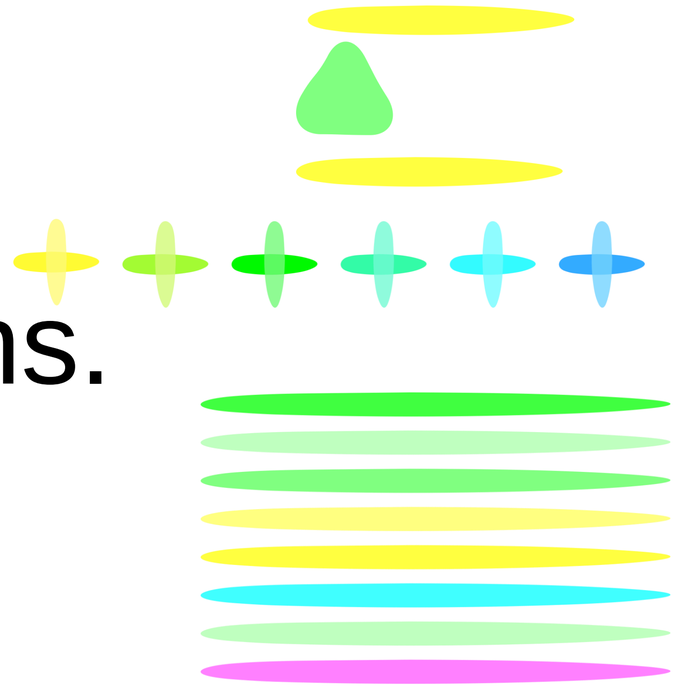
Evolutionary Architecture

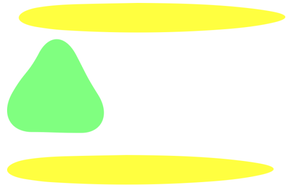
An evolutionarv architecture supports

guided

incremental change

across multiple dimensions.



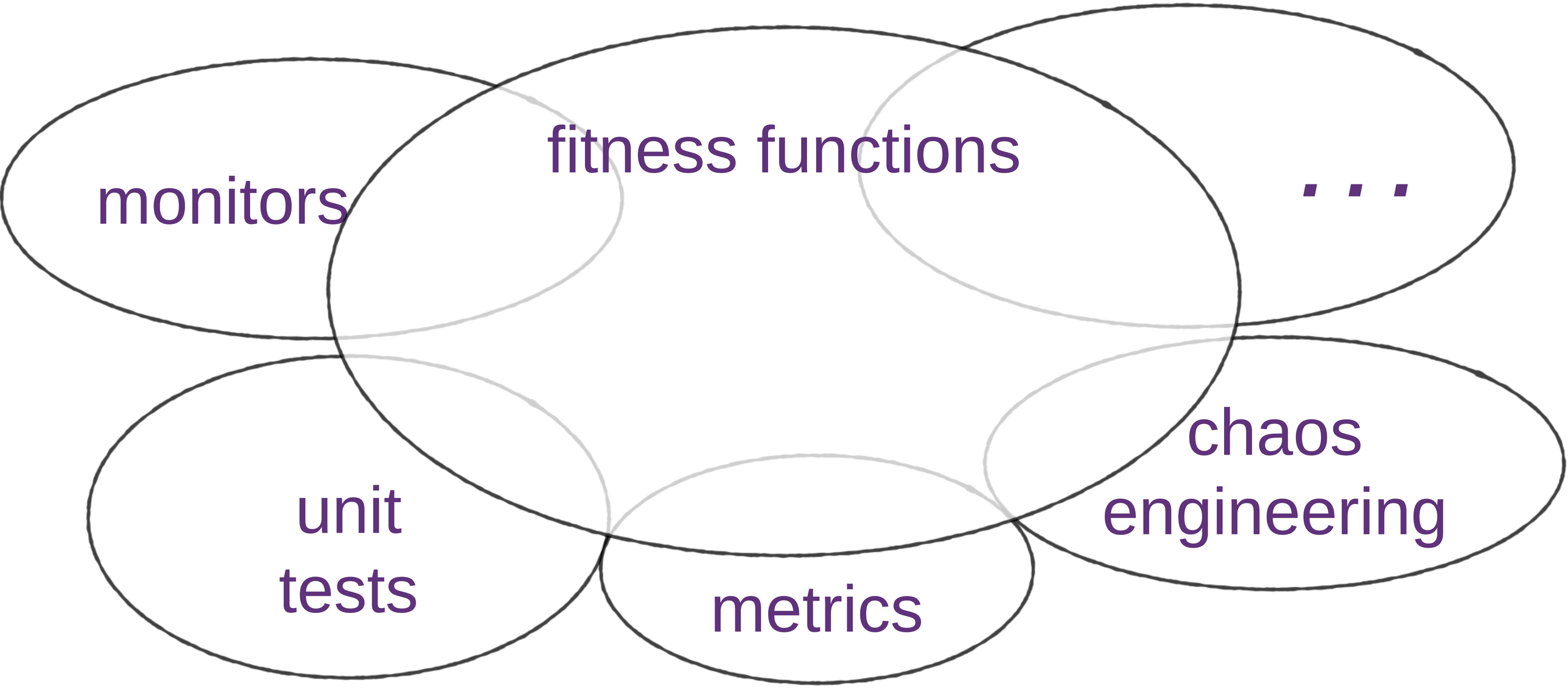


guided

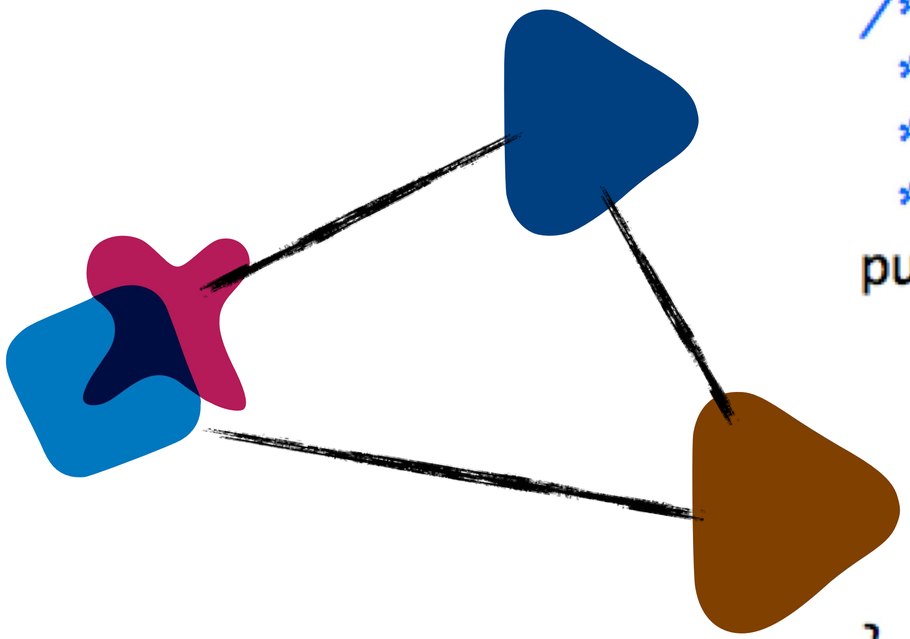
architectural *fitness function*:

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

Fitness Functions



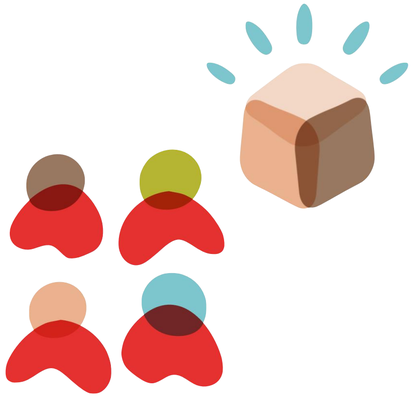
Cyclic Dependency Function

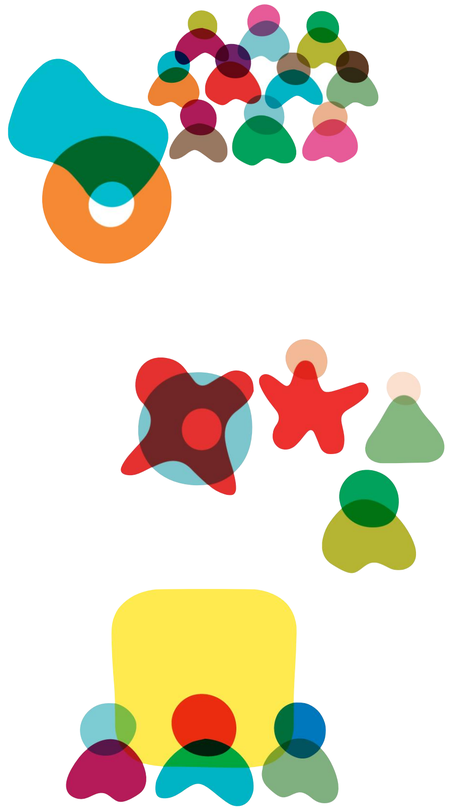


```
/**
 * Tests that a package dependency cycle does not
 * exist for any of the analyzed packages.
 */
public void testAllPackages() {

    Collection packages = jdepend.analyze();

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

clarkware.com/software/JDepend.html

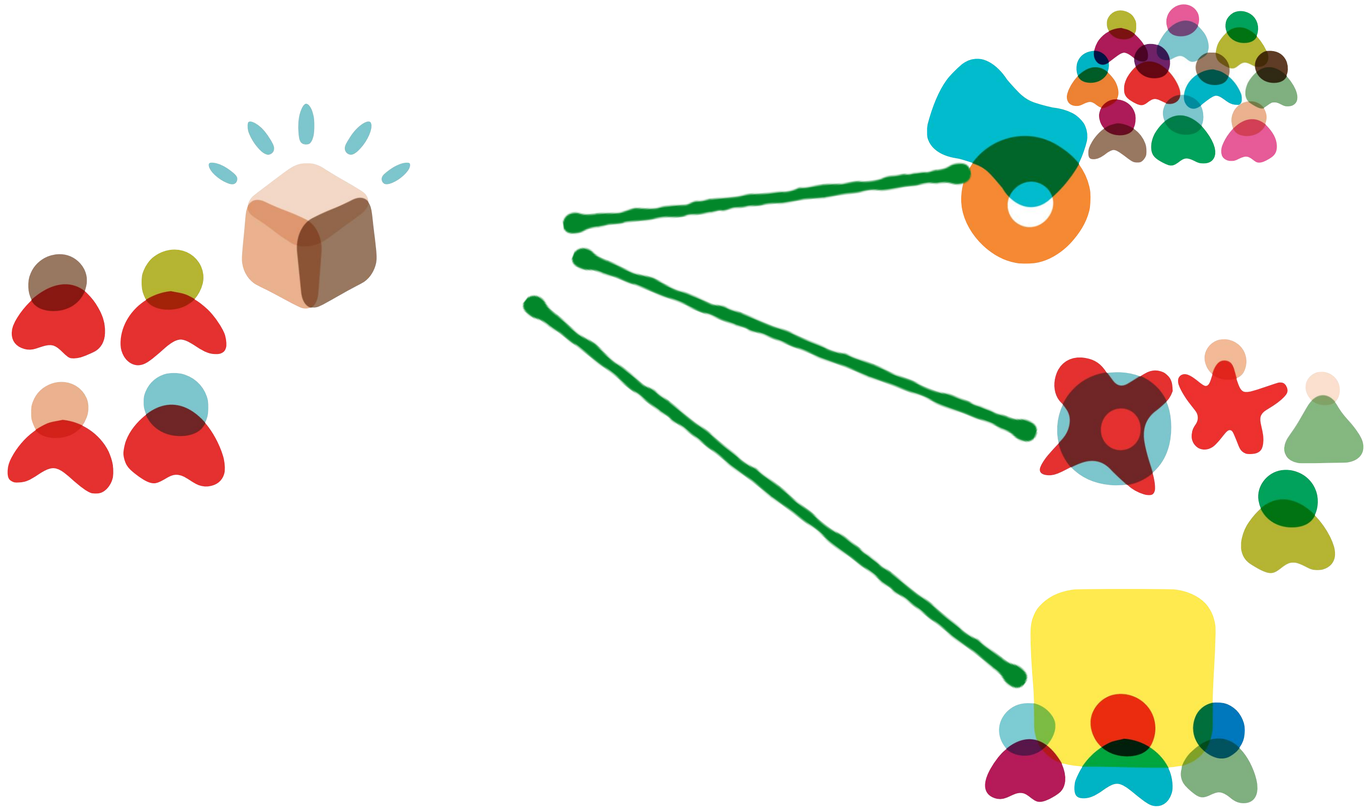
Consumer Driven Contracts





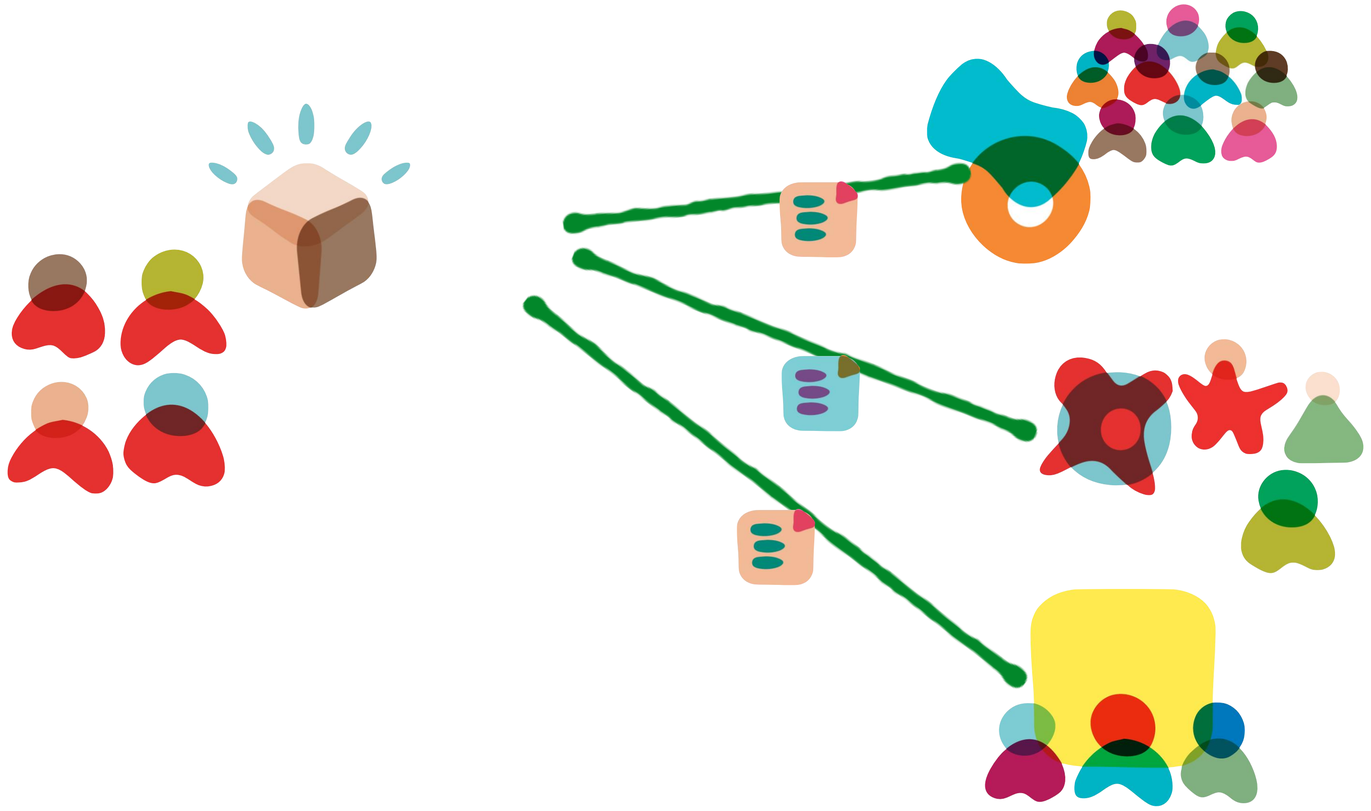
martinfowler.com/articles/consumerDrivenContracts.html

Consumer Driven Contracts



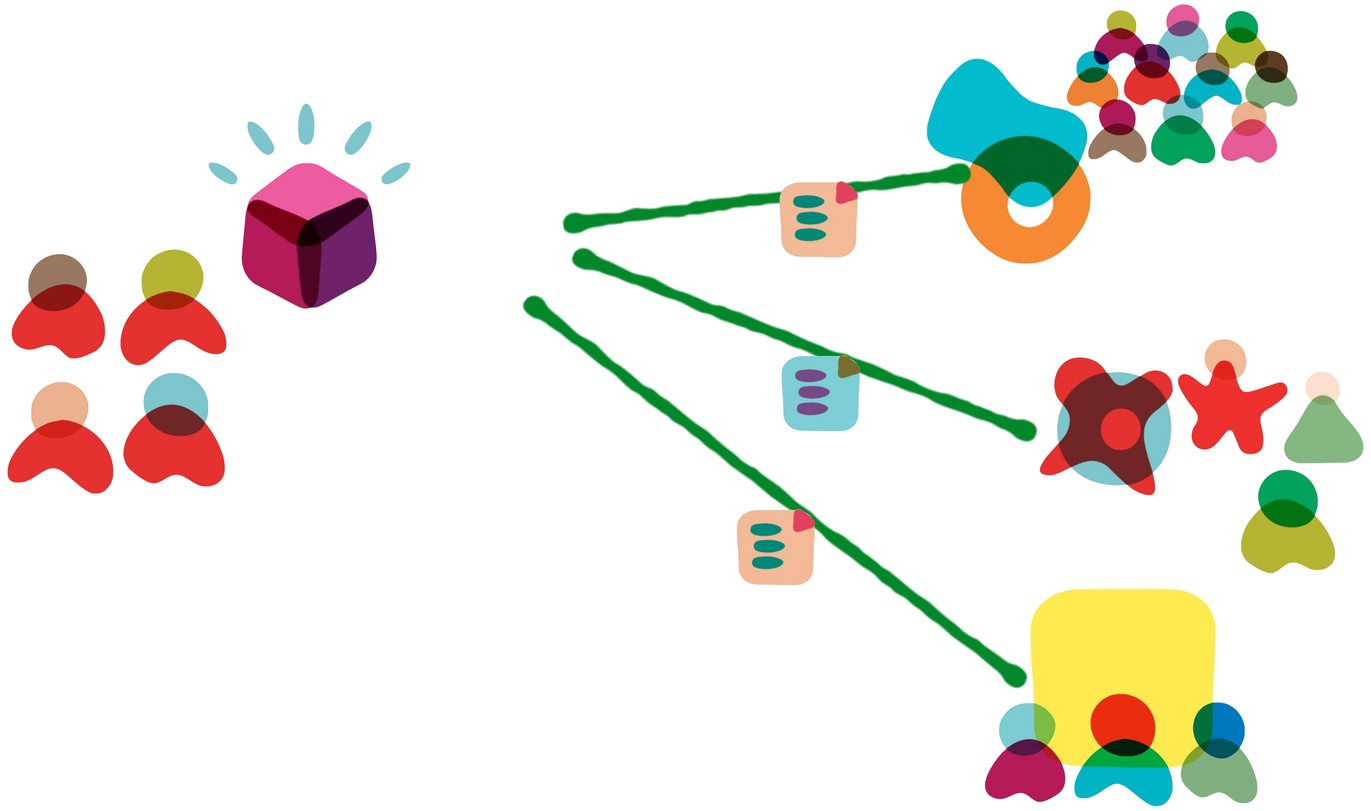
martinfowler.com/articles/consumerDrivenContracts.html

Consumer Driven Contracts



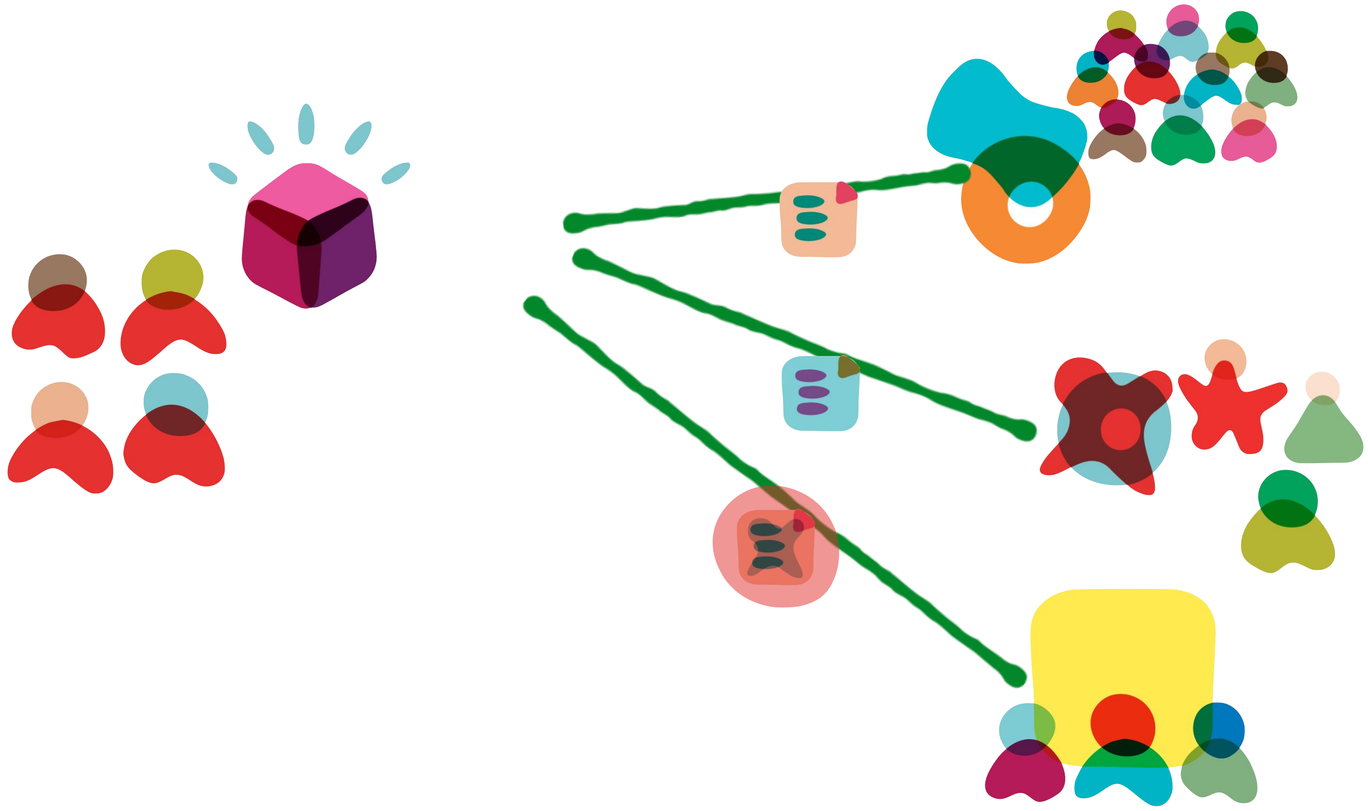
martinfowler.com/articles/consumerDrivenContracts.html

Consumer Driven Contracts



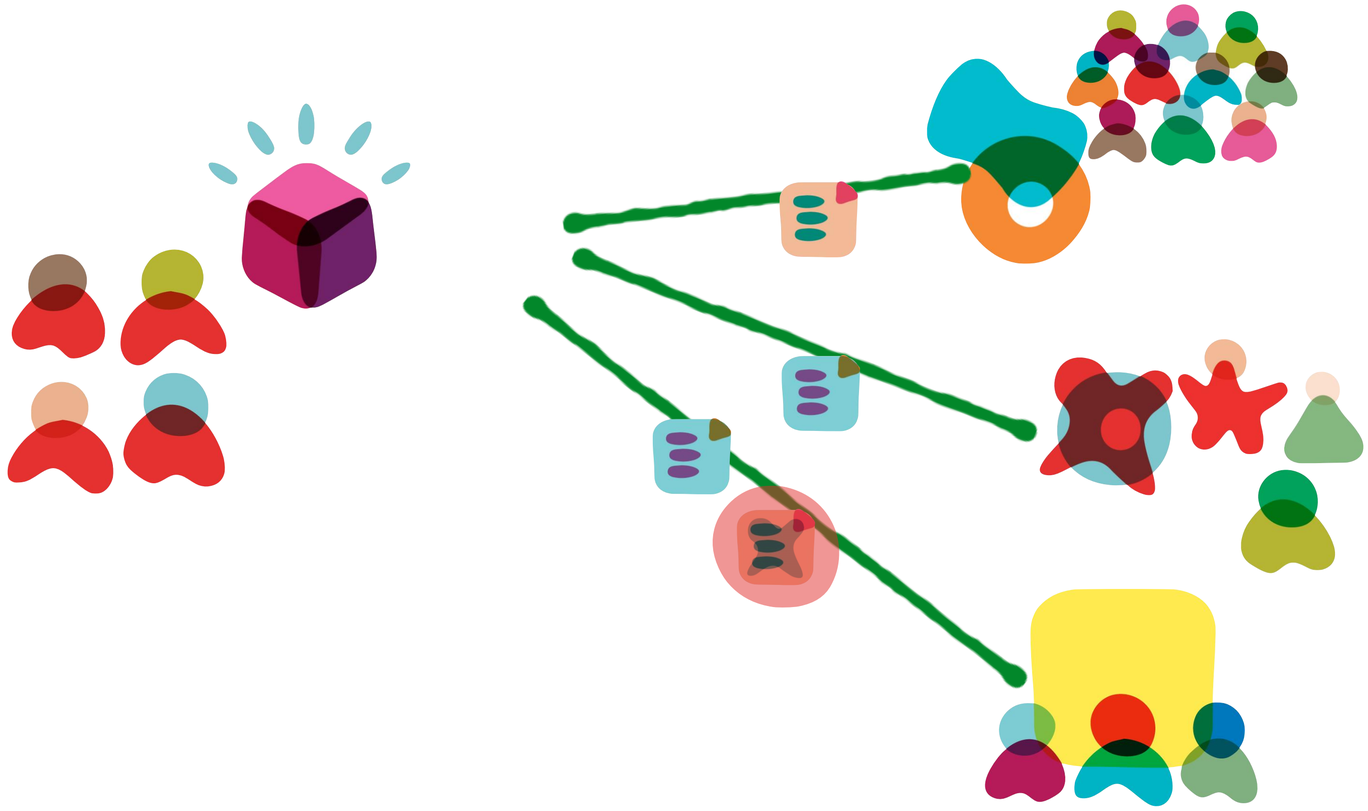
martinfowler.com/articles/consumerDrivenContracts.html

Consumer Driven Contracts



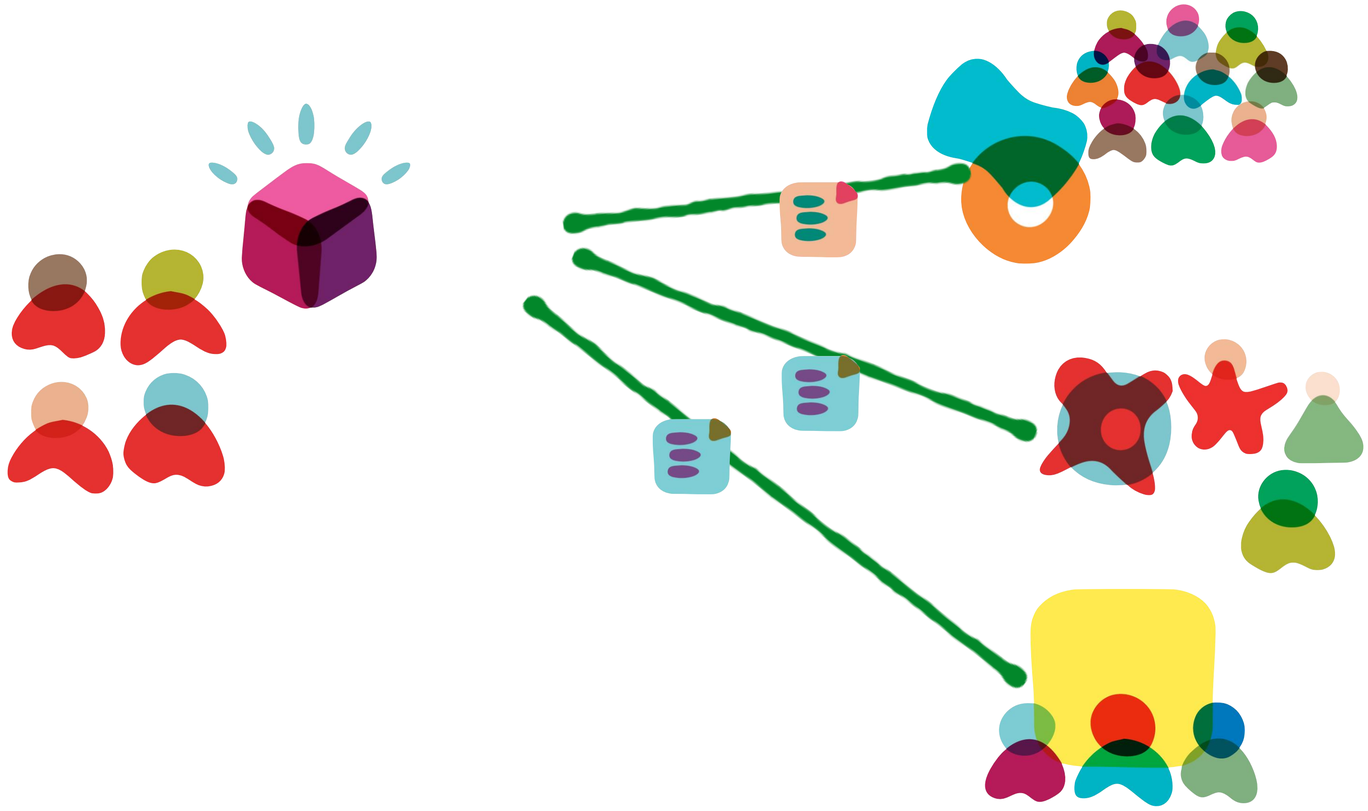
martinfowler.com/articles/consumerDrivenContracts.html

Consumer Driven Contracts



martinfowler.com/articles/consumerDrivenContracts.html

Consumer Driven Contracts



martinfowler.com/articles/consumerDrivenContracts.html


amazon
web services
CHAOS
MONKEY


amazon
web services
CHAOS
MONKEY


SIMIAN
ARMY

chaos monkey

chaos gorilla

latency
monkey

doctor
monkey

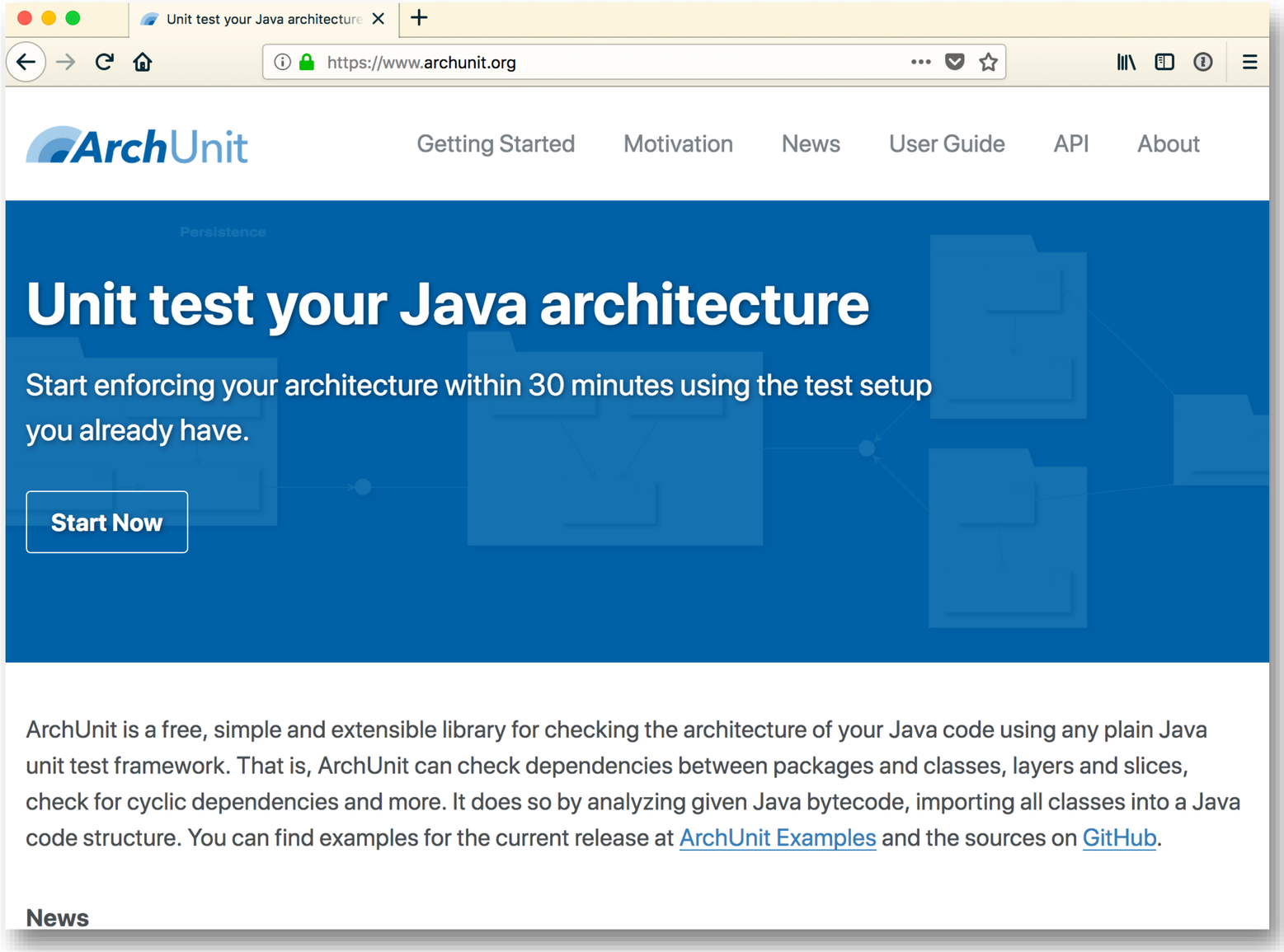
janitor monkey


conformity
monkey
SIMIAN
ARMY


security monkey
conformity monkey
SIMIAN
ARMY

ArchUnit

https://www.archunit.org/



ArchUnit

https://www.archunit.org/

```java
@AnalyzeClasses(packages = "com.tngtech.archunit.example")
public class CodingRulesTest {

    @ArchTest
    private final ArchRule no_access_to_standard_streams = NO_CLASSES_SHOULD_ACCESS_STANDARD_STREAMS;

    @ArchTest
    private void no_access_to_standard_streams_as_method(JavaClasses classes) {
        noClasses().should(ACCESS_STANDARD_STREAMS).check(classes);
    }

    @ArchTest
    private final ArchRule no_generic_exceptions = NO_CLASSES_SHOULD_THROW_GENERIC_EXCEPTIONS;

    @ArchTest
    private final ArchRule no_java_util_logging = NO_CLASSES_SHOULD_USE_JAVA_UTIL_LOGGING;

    @ArchTest
    private final ArchRule no_jodatime = NO_CLASSES_SHOULD_USE_JODATIME;

    @ArchTest
    static final ArchRule no_classes_should_access_standard_streams_or_throw_generic_exceptions =
            CompositeArchRule.of(NO_CLASSES_SHOULD_ACCESS_STANDARD_STREAMS)
                    .and(NO_CLASSES_SHOULD_THROW_GENERIC_EXCEPTIONS);

}
```

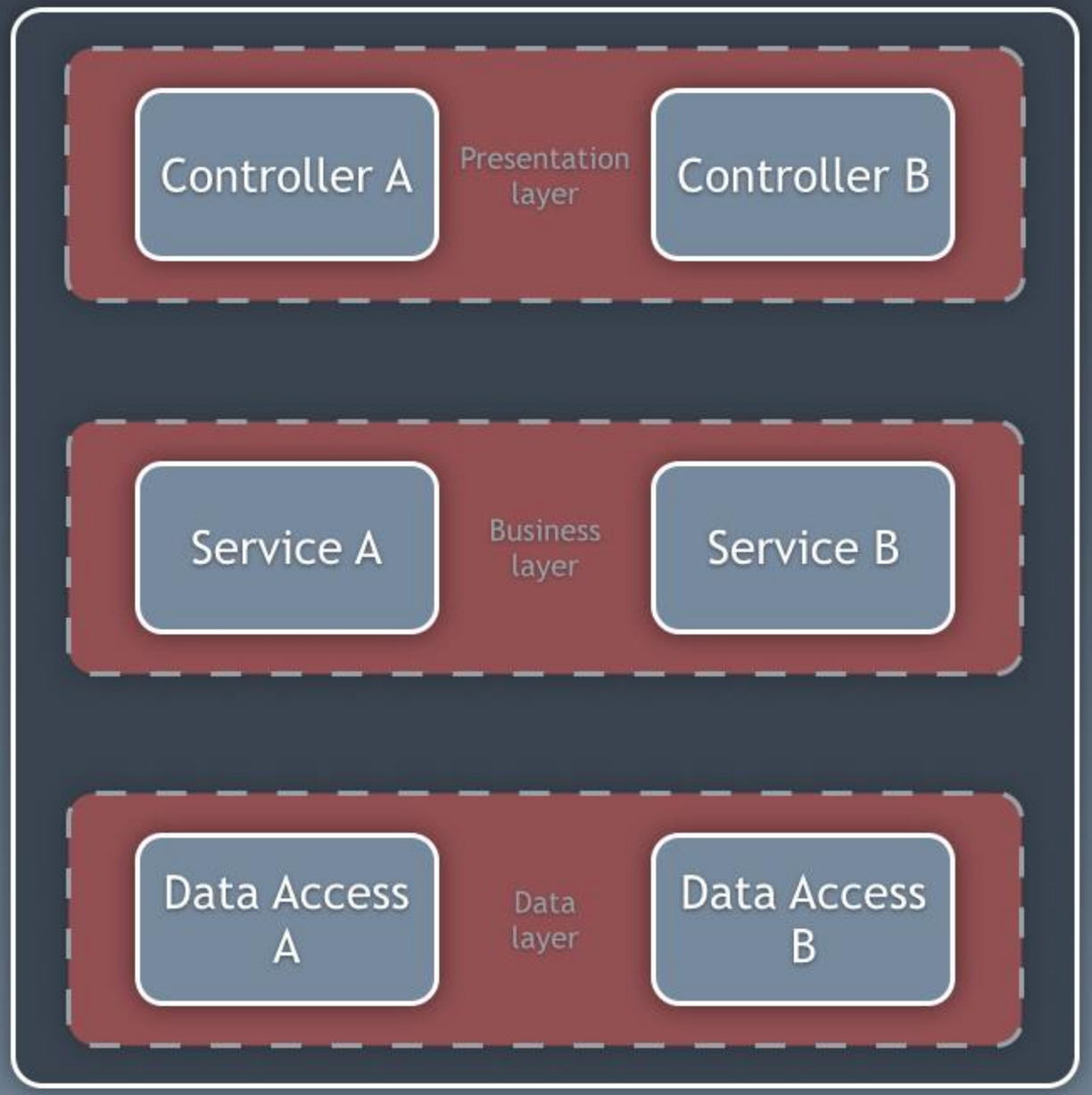
coding rules

ArchUnit

https://www.archunit.org/

coding rules

```java
@AnalyzeClasses(packages = "com.tngtech.archunit.example")
public class ControllerRulesTest {

    @ArchTest
    static final ArchRule controllers_should_only_call_secured_methods =
            classes().that().resideInAPackage("..controller..")
                    .should().onlyCallMethodsThat(areDeclaredInController().or(are(annotatedWith(Secured.class))));

    @ArchTest
    static final ArchRule controllers_should_only_call_secured_constructors =
            classes()
                    .that().resideInAPackage("..controller..")
                    .should().onlyCallConstructorsThat(areDeclaredInController().or(are(annotatedWith(Secured.class))));

    @ArchTest
    static final ArchRule controllers_should_only_call_secured_code_units =
            classes()
                    .that().resideInAPackage("..controller..")
                    .should().onlyCallCodeUnitsThat(areDeclaredInController().or(are(annotatedWith(Secured.class))));

    @ArchTest
    static final ArchRule controllers_should_only_access_secured_fields =
            classes()
                    .that().resideInAPackage("..controller..")
                    .should().onlyAccessFieldsThat(areDeclaredInController().or(are(annotatedWith(Secured.class))));

    @ArchTest
    static final ArchRule controllers_should_only_access_secured_members =
            classes()
                    .that().resideInAPackage("..controller..")
                    .should().onlyAccessMembersThat(areDeclaredInController().or(are(annotatedWith(Secured.class))));

    private static DescribedPredicate<JavaMember> areDeclaredInController() {
        DescribedPredicate<JavaClass> aPackageController = GET_PACKAGE_NAME.is(PackageMatchers.of("..controller..", "java.."))
                .as("a package '..controller..'");
        return are(declaredIn(aPackageController));
    }

}
```



Package by layer (horizontal slicing)

ArchUnit

https://www.archunit.org/

```java
@AnalyzeClasses(packages = "com.tngtech.archunit.example")
public class LayeredArchitectureTest {
    @ArchTest
    static final ArchRule layer_dependencies_are_respected = layeredArchitecture()

            .layer("Controllers").definedBy("com.tngtech.archunit.example.controller..")
            .layer("Services").definedBy("com.tngtech.archunit.example.service..")
            .layer("Persistence").definedBy("com.tngtech.archunit.example.persistence..")

            .whereLayer("Controllers").mayNotBeAccessedByAnyLayer()
            .whereLayer("Services").mayOnlyBeAccessedByLayers("Controllers")
            .whereLayer("Persistence").mayOnlyBeAccessedByLayers("Services");

    @ArchTest
    static final ArchRule layer_dependencies_are_respected_with_exception = layeredArchitecture()

            .layer("Controllers").definedBy("com.tngtech.archunit.example.controller..")
            .layer("Services").definedBy("com.tngtech.archunit.example.service..")
            .layer("Persistence").definedBy("com.tngtech.archunit.example.persistence..")

            .whereLayer("Controllers").mayNotBeAccessedByAnyLayer()
            .whereLayer("Services").mayOnlyBeAccessedByLayers("Controllers")
            .whereLayer("Persistence").mayOnlyBeAccessedByLayers("Services")

            .ignoreDependency(SomeMediator.class, ServiceViolatingLayerRules.class);
}
```

layer dependency

ArchUnit

https://www.archunit.org/

```
@ArchTest
static final ArchRule services_should_not_access_controllers =
        noClasses().that().resideInAPackage("..service..")
                .should().accessClassesThat().resideInAPackage("..controller..");

@ArchTest
static final ArchRule persistence_should_not_access_services =
        noClasses().that().resideInAPackage("..persistence..")
                .should().accessClassesThat().resideInAPackage("..service..");

@ArchTest
static final ArchRule services_should_only_be_accessed_by_controllers_or_other_services =
        classes().that().resideInAPackage("..service..")
                .should().onlyBeAccessed().byAnyPackage("..controller..", "..service..");

@ArchTest
static final ArchRule services_should_only_access_persistence_or_other_services =
        classes().that().resideInAPackage("..service..")
                .should().onlyAccessClassesThat().resideInAnyPackage("..service..", "..persistence..", "java..");
```

```
@ArchTest
static final ArchRule services_should_not_depend_on_controllers =
        noClasses().that().resideInAPackage("..service..")
                .should().dependOnClassesThat().resideInAPackage("..controller..");

@ArchTest
static final ArchRule persistence_should_not_depend_on_services =
        noClasses().that().resideInAPackage("..persistence..")
                .should().dependOnClassesThat().resideInAPackage("..service..");

@ArchTest
static final ArchRule services_should_only_be_depended_on_by_controllers_or_other_services =
        classes().that().resideInAPackage("..service..")
                .should().onlyHaveDependentClassesThat().resideInAnyPackage("..controller..", "..service..");

@ArchTest
static final ArchRule services_should_only_depend_on_persistence_or_other_services =
        classes().that().resideInAPackage("..service..")
                .should().onlyDependOnClassesThat().resideInAnyPackage("..service..", "..persistence..", "java..");
```

layer dependency

NetArchTest

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

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

NetArchTest

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

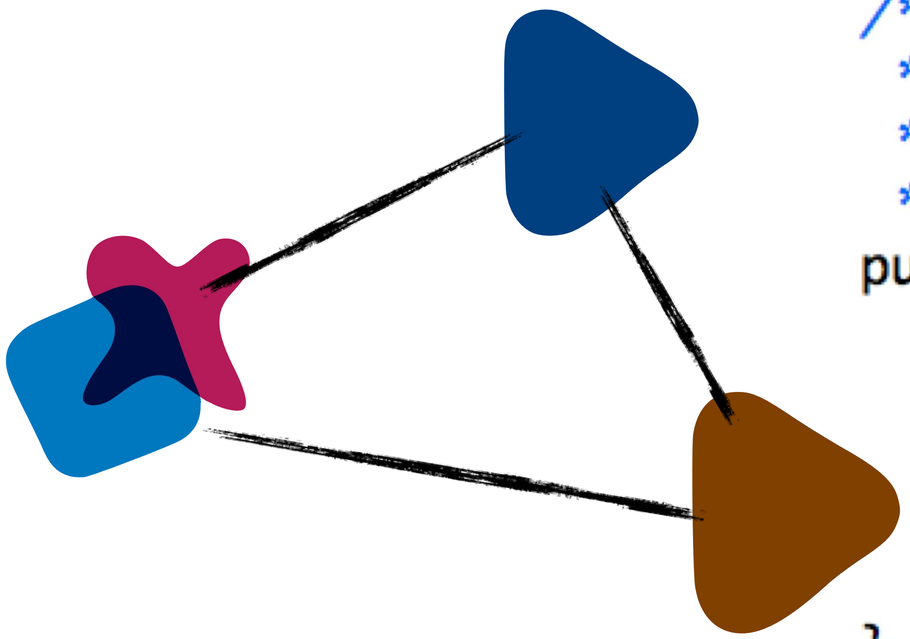
```
// Only classes in the data namespace can have a dependency on System.Data
result = Types.InCurrentDomain()
    .That().HaveDependencyOn("System.Data")
    .And().ResideInNamespace(("ArchTest"))
    .Should().ResideInNamespace(("NetArchTest.SampleLibrary.Data"))
    .GetResult().IsSuccessful;
```

NetArchTest

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

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

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

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

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

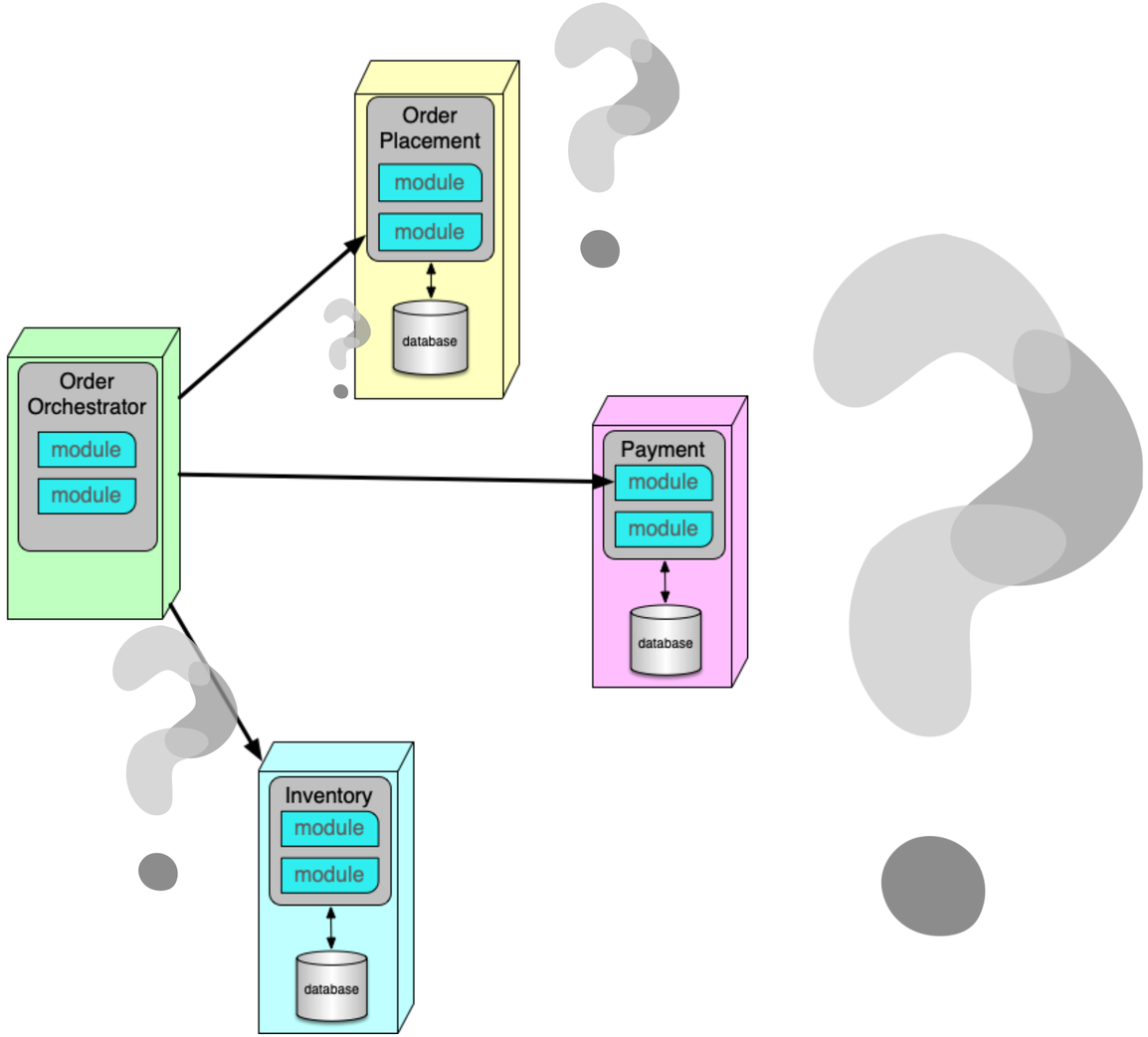
Cyclic Dependency Function



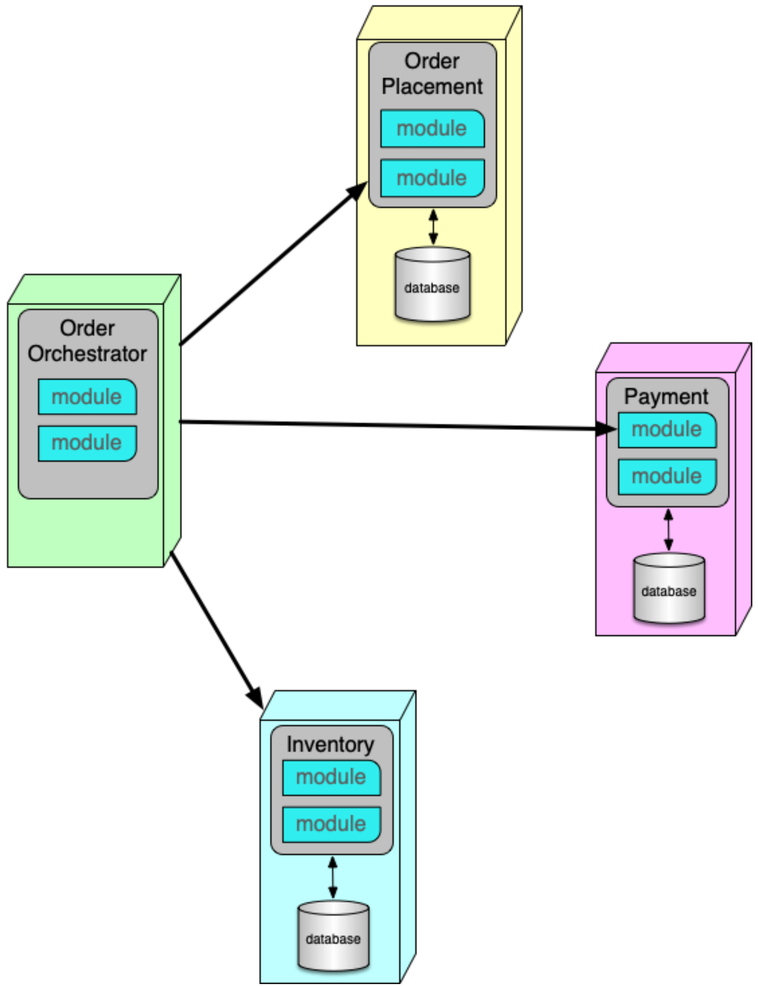
```
/**
 * Tests that a package dependency cycle does not
 * exist for any of the analyzed packages.
 */
public void testAllPackages() {

    Collection packages = jdepend.analyze();

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

clarkware.com/software/JDepend.html

Coupling in microservices?



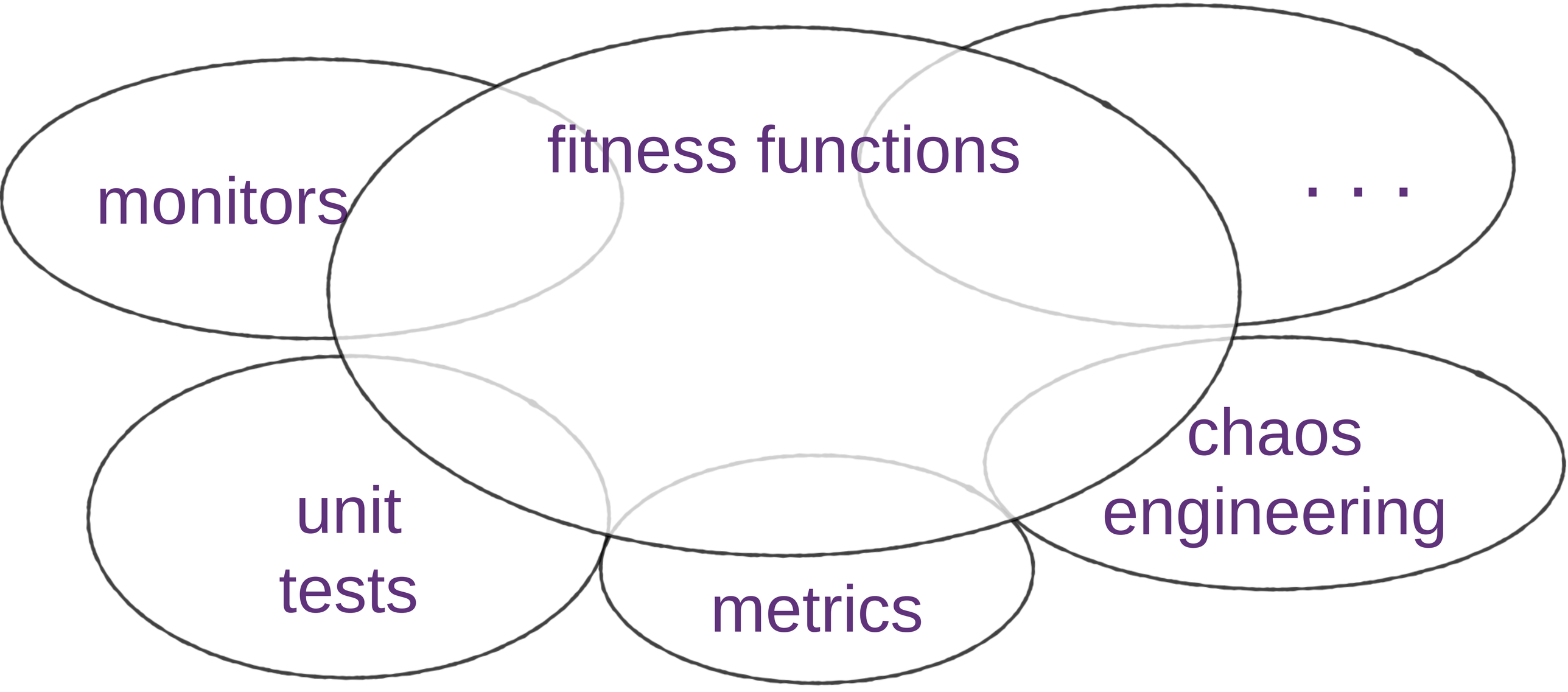
Coupling in microservices



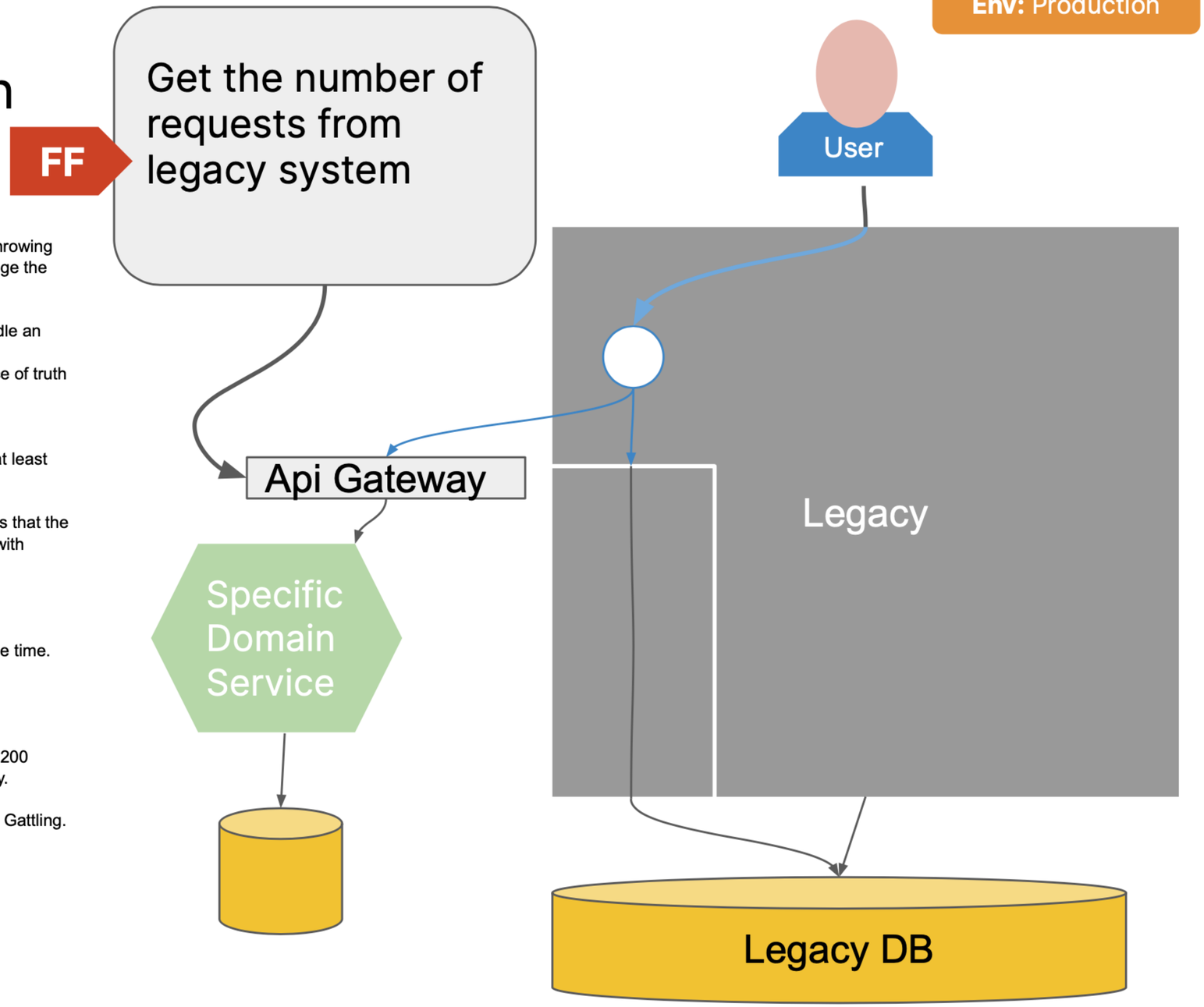
Coupling in microservices

```
def ensure_domain_services_communicate_only_with_orchestrator
  list_of_services = List.new()
                          .add("orchestrator")
                          .add("order placement")
                          .add("payment")
                          .add("inventory")
  list_of_services.each { |service|
    service.import_logsFor(24.hours)
     calls_from(service).each { |call|
         unless call.destination.equals("orchestrator")
          raise FitnessFunctionFailure.new()
     }
   }
end
```

Fitness Functions



Concurrency Fitness Function



Question : How we can ensure that the auto scaling factor is good enough if our service is throwing concurrency issues?, Do we need to refactor the service to improve the performance or change the Auto Scaling factor?

Context: We was extrangulating a legacy System, so we created a new microservice to handle an specific part of the domain.
This new service is in production applying a double writing strategy but maintaining the source of truth in the Legacy DB.
The Concurrency SLA is 120 Request/Second.

Issue: The service is crashing and the load and concurrency tests show us that can handle at least 300 request/Second.

FF: Calculate the number of incoming calls in production to verify the max amount of requests that the service needs to support and what would be the auto scaling factor to guarantee Availability with horizontal scaling.

- Create a New Relic Query to get the # of calls per second in production.
- Make New load and concurrency tests using the new # of request per second.
- Monitor the memory , cpu and define the stress point.
- Put that FF in the pipeline to guarantee the Availability and performance over the time.

Type : Temporal**,** Automatic, Holistic.
Ility: Availability, Concurrency, Reliability, Performance

Conclusion : The SLA is too low to this service, for the average calls per second is around 1200 requests/sec, so we update the auto scaling factor using also CPU # of requests and Memory.

Technology : Ruby & Rails, Java, Spring Boot, New Relic , AWS Kubernetes, Karate, Scala, Gattling.

Domain : Health Care

Fitness Function Guidelines

most fitness function are local

Fitness Function Guidelines

most fitness function are local

global only when
universally applied.

Fitness Function Guidelines

most fitness function are local

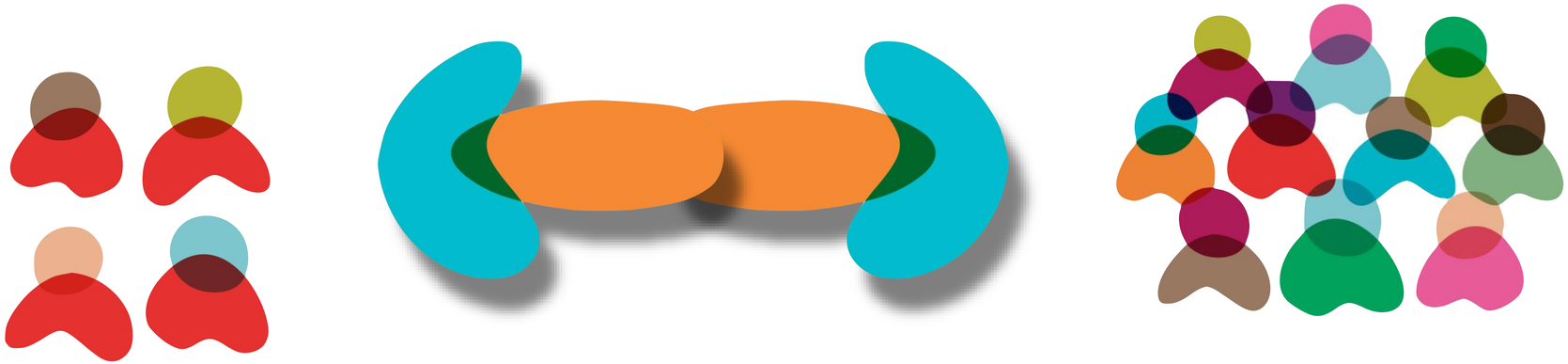
global only when
universally applied.

No architect  states!

Fitness Function Guidelines

developers must collaborate

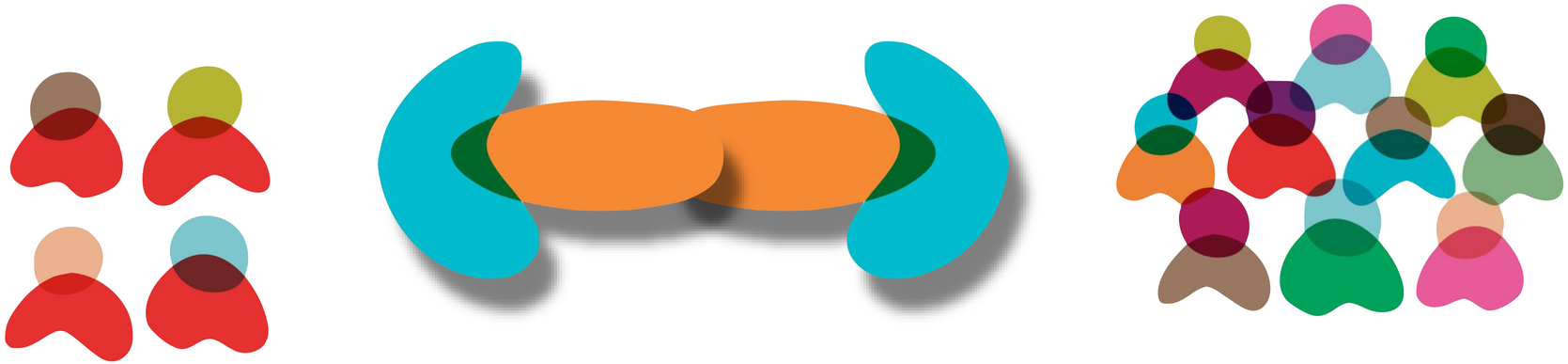
architects



developers

Fitness Function Guidelines

developers must collaborate



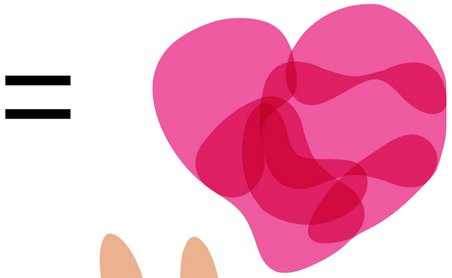
No architect  states!

Fitness Function Guidelines

prefer automation but
rely on manual when needed

Fitness Function Guidelines

prefer automation but
rely on manual when needed

automation = 

when needed

Question:

What characteristics of architecture make continuous delivery/deployment easier?

Question:

What does *agility* mean as an architecture characteristic?

agility

deployability

agility

testability

deployability

modularity

agility

testability

controlled coupling

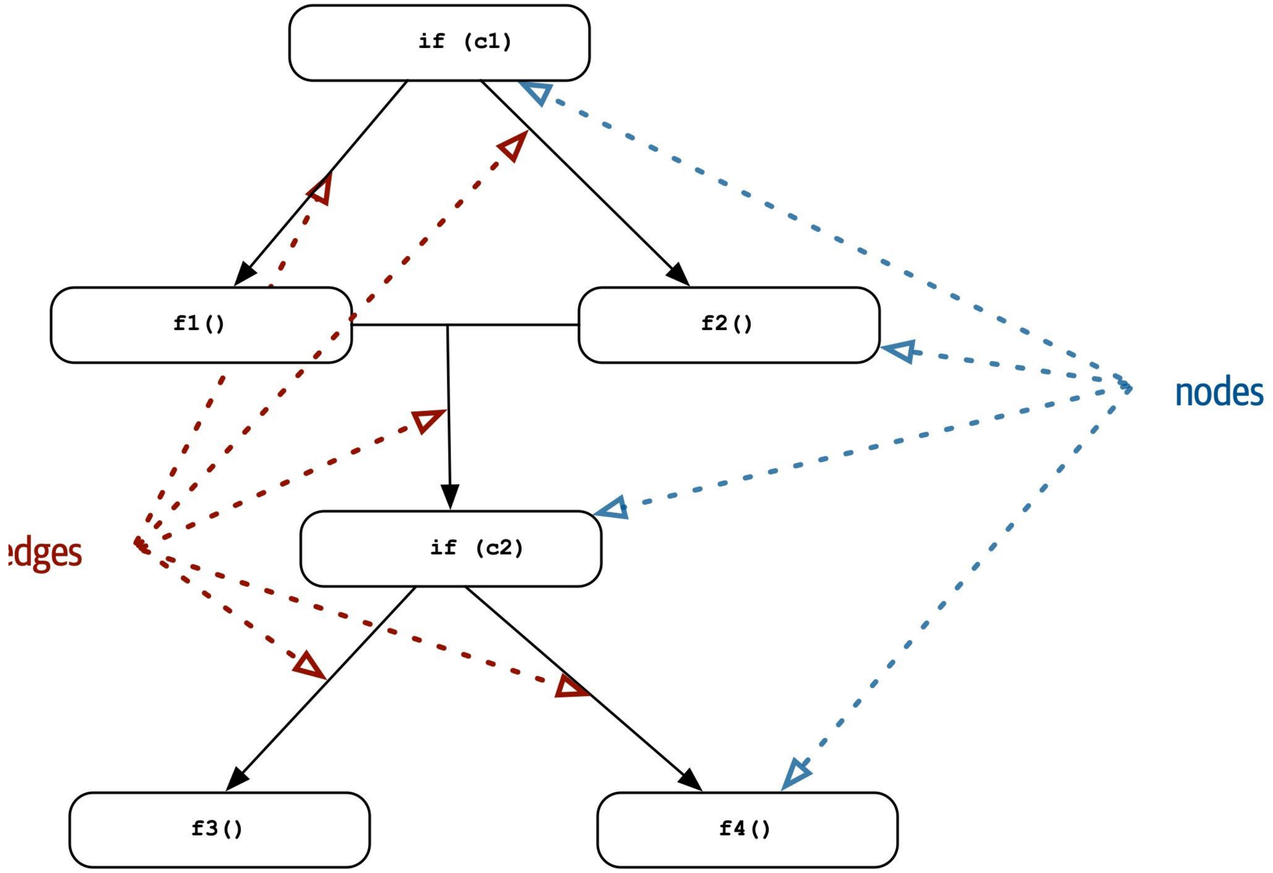

metrics

metrics

cyclomatic complexity

V(G)= e - n + 2

```
public void decision(int c1, int c2) {
    if (c1 < 100)
        return 0;
    else if (c1 + C2 > 500)
       return 1;
    else
      return -1;
}
```



metrics

Chidamber & Kemerer Metrics

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

metrics

Chidamber & Kemerer Metrics

✓LCOM (Lack of Cohesion in Methods)

$$LCOM96b = \frac{1}{a}\sum_{j-1}^{a} m - \mu(Aj)$$

metrics

Chidamber & Kemerer Metrics

✓LCOM (Lack of Cohesion in Methods)

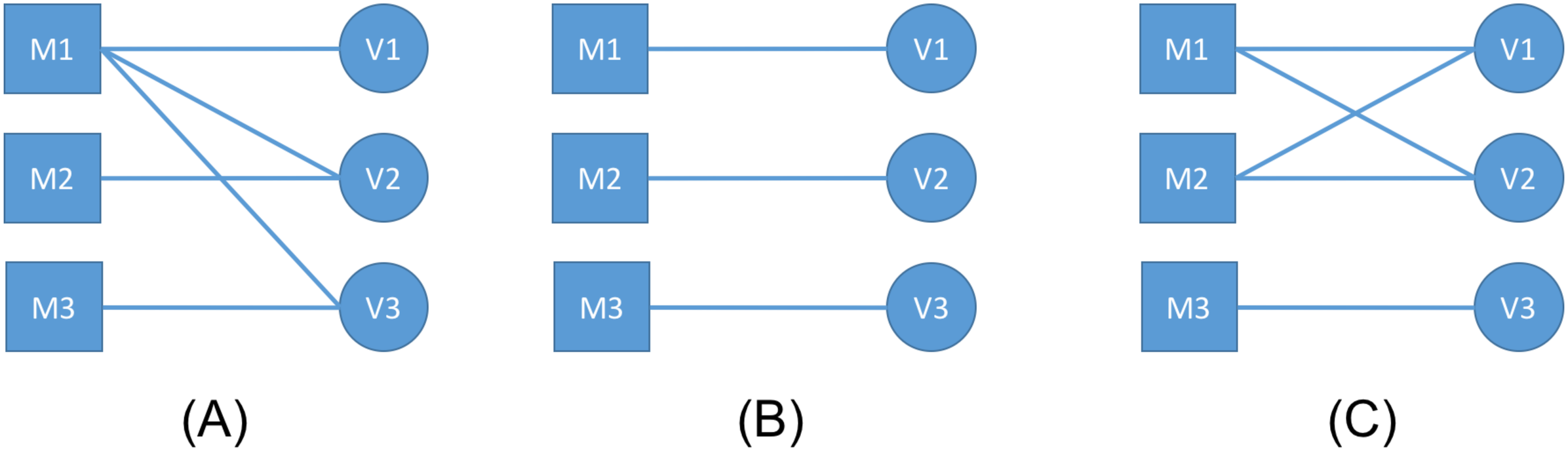
~~The LCOM is a count of the number of method pairs whose similarity is 0 (i.e. σ() is a null set) minus the count of method pairs whose similarity is not zero.~~

The sum of sets of methods not shared via sharing fields.

metrics

Chidamber & Kemerer Metrics

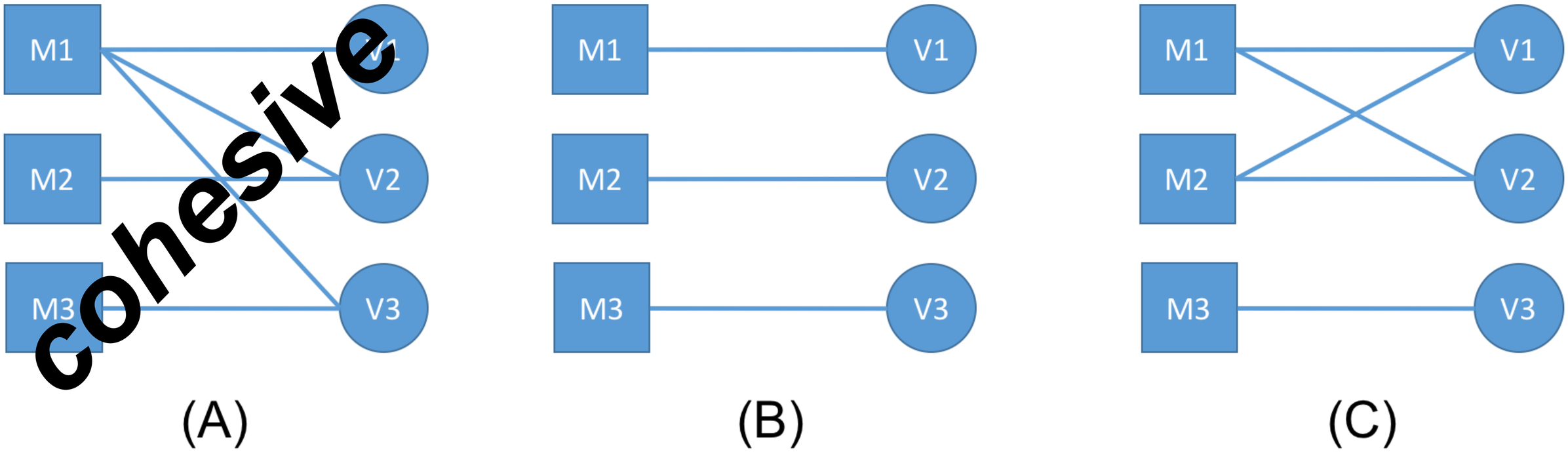
✓LCOM (Lack of Cohesion in Methods)



metrics

Chidamber & Kemerer Metrics

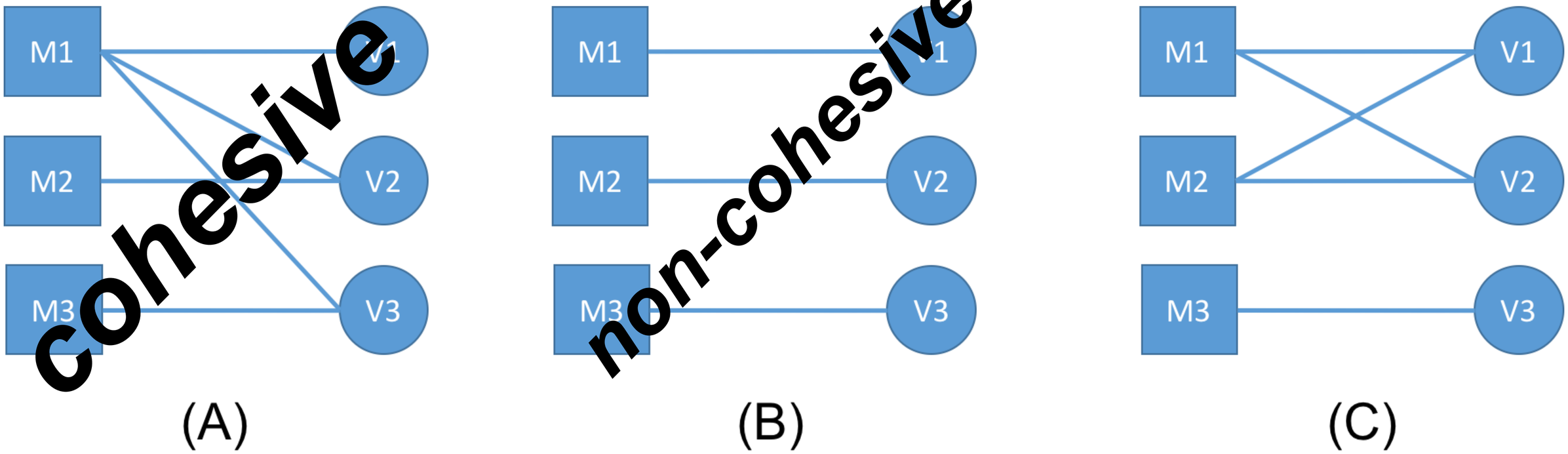
✓LCOM (Lack of Cohesion in Methods)



metrics

Chidamber & Kemerer Metrics

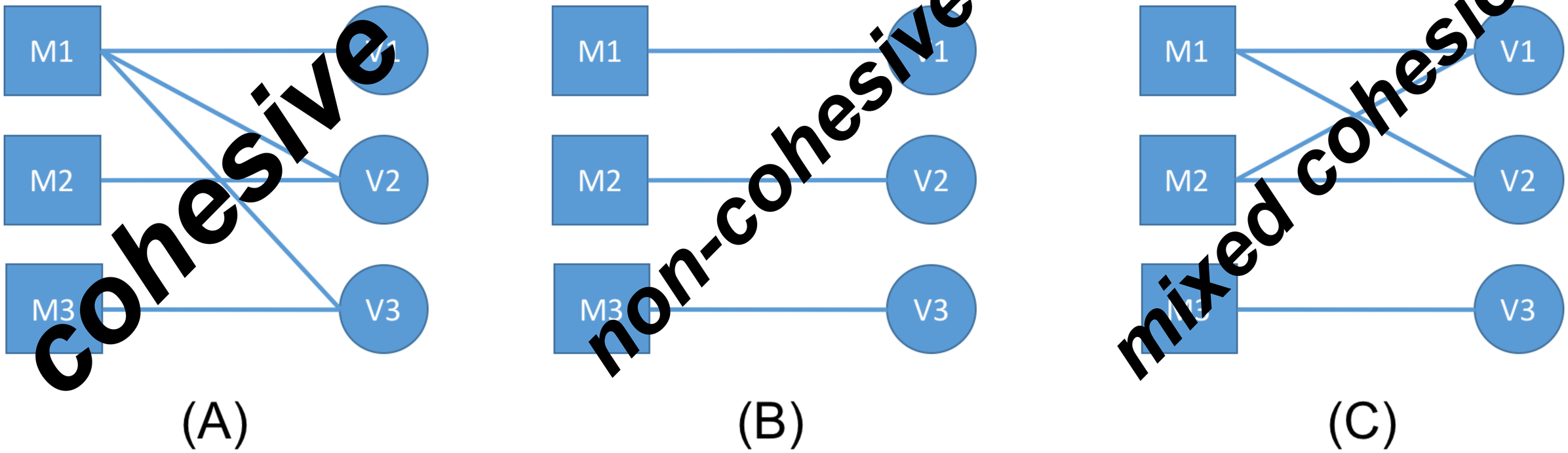
✓LCOM (Lack of Cohesion in Methods)



metrics

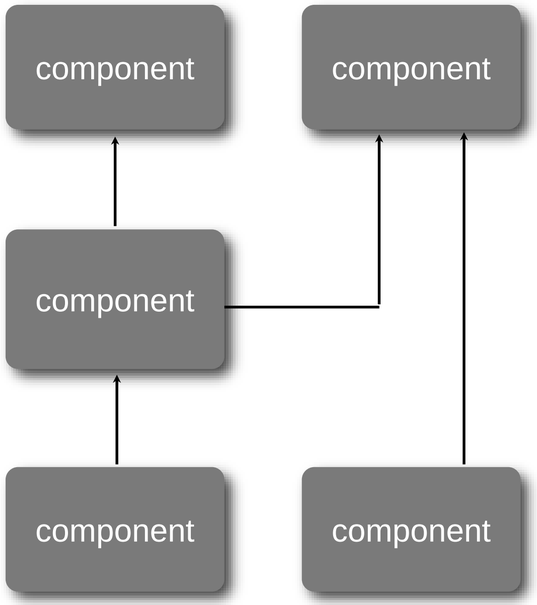
Chidamber & Kemerer Metrics

✓LCOM (Lack of Cohesion in Methods)



component identification

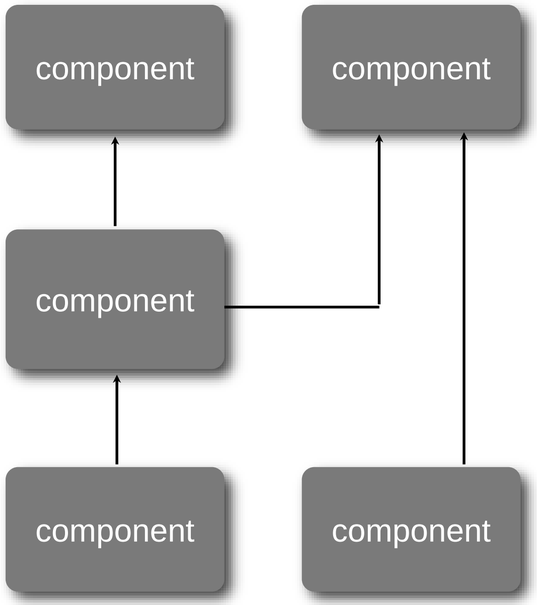
as an architect, you should think about the artifacts within the architecture in terms of *components*



component:

component identification

as an architect, you should think about the artifacts within the architecture in terms of *components*

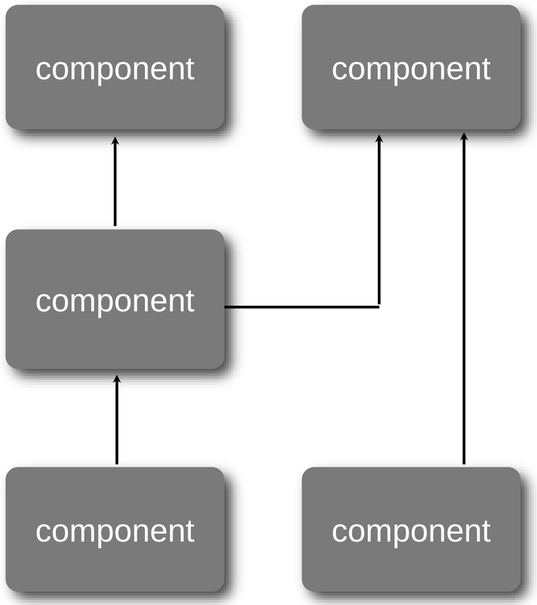


component:

building block of the application

component identification

as an architect, you should think about the artifacts within the architecture in terms of *components*



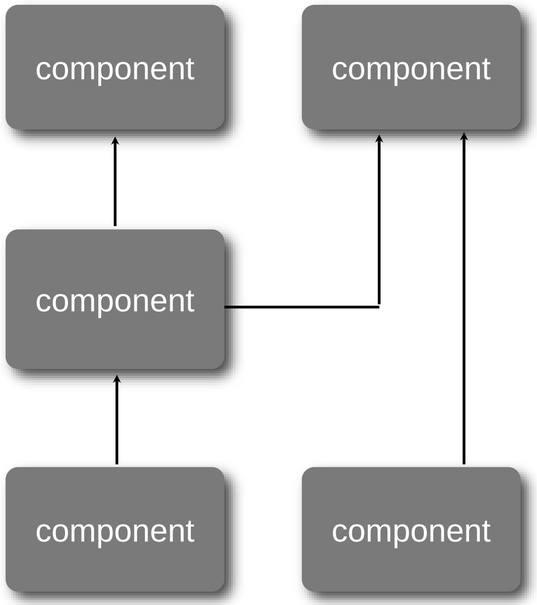
component:

building block of the application

well defined set of operations

component identification

as an architect, you should think about the artifacts within the architecture in terms of *components*



component:

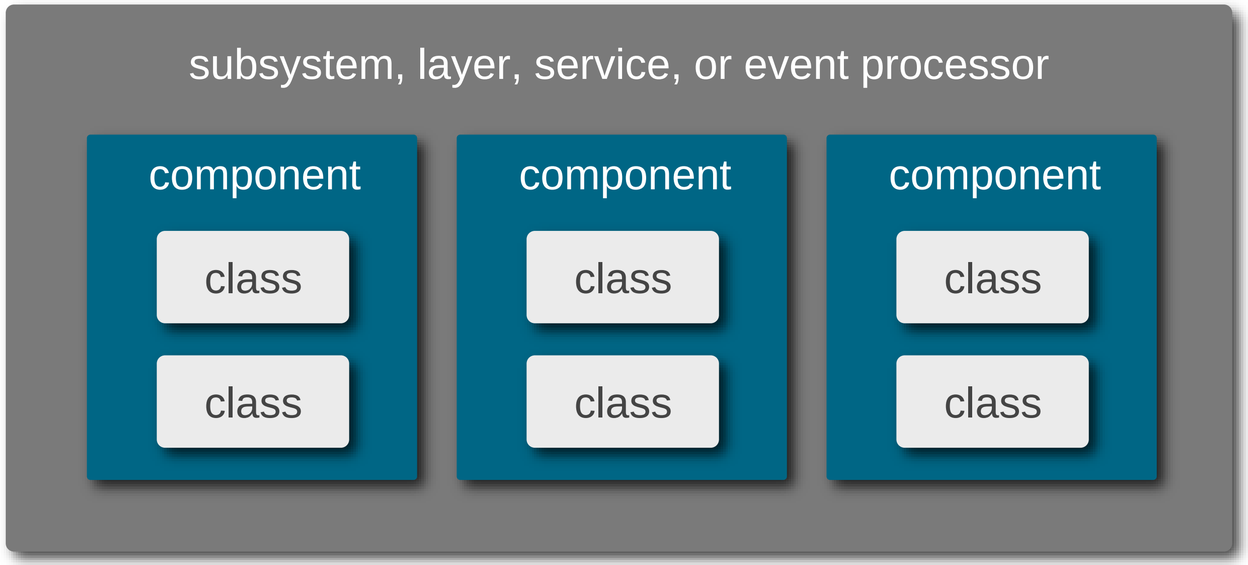
building block of the application

well defined set of operations

well defined role and responsibility

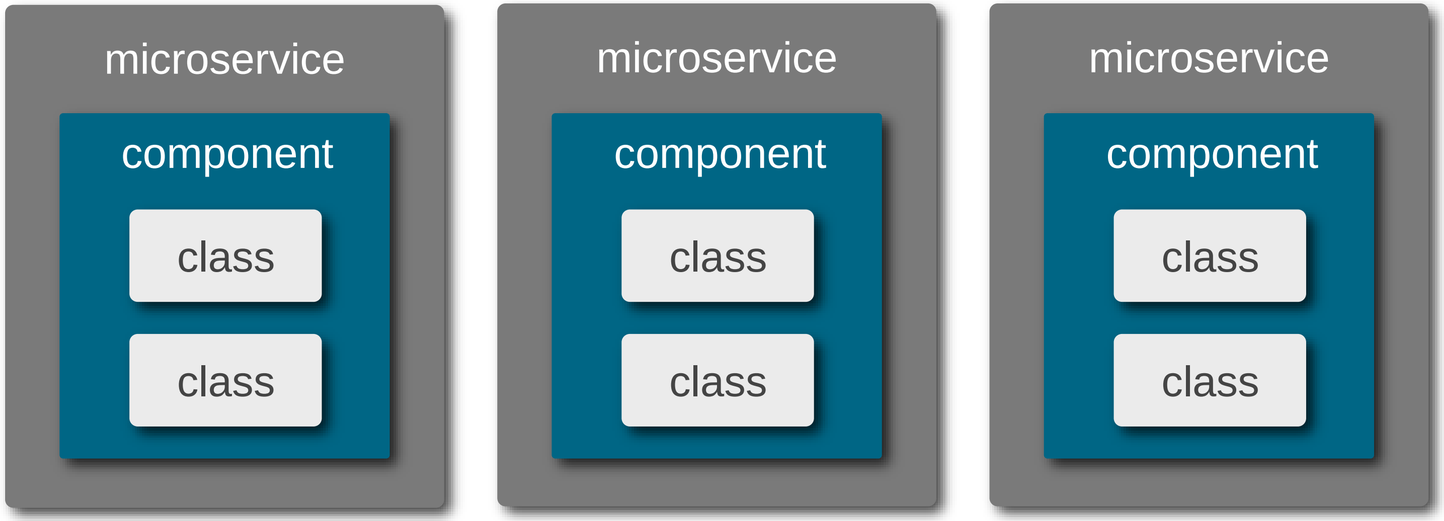
component identification

component scope



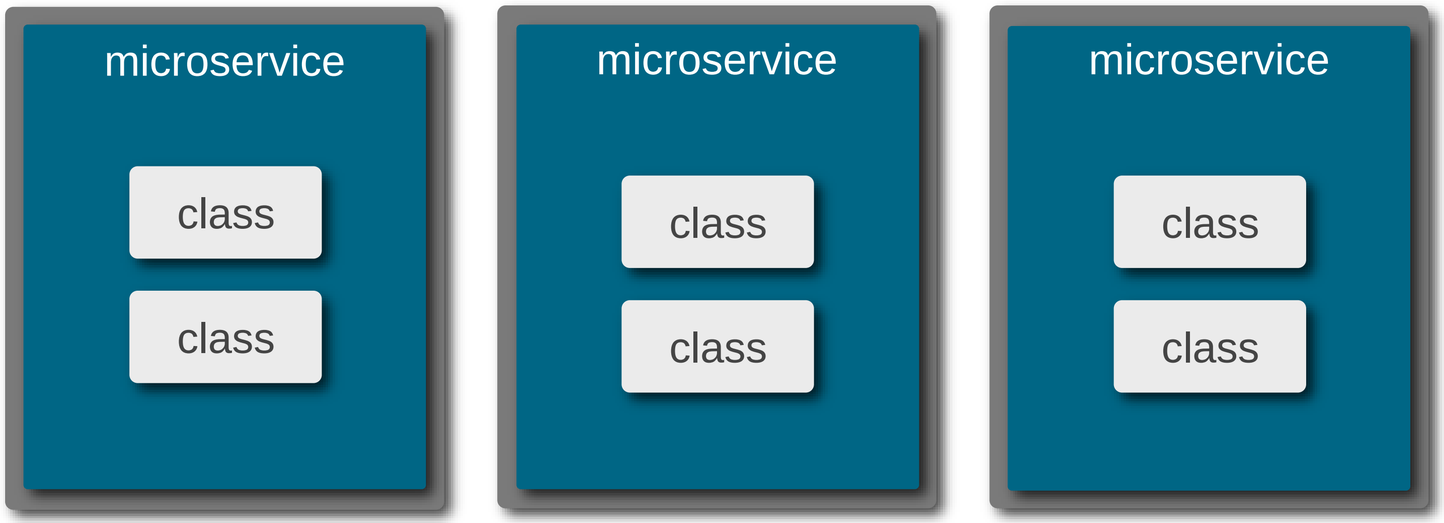
component identification

component scope



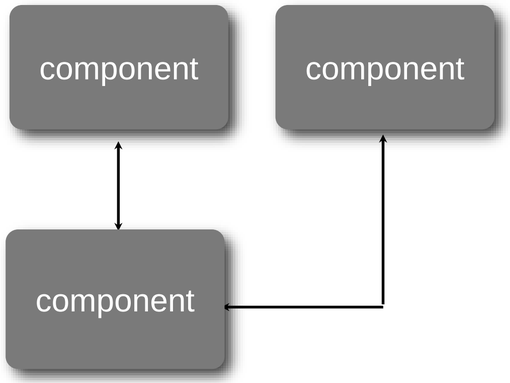
component identification

component scope



component coupling

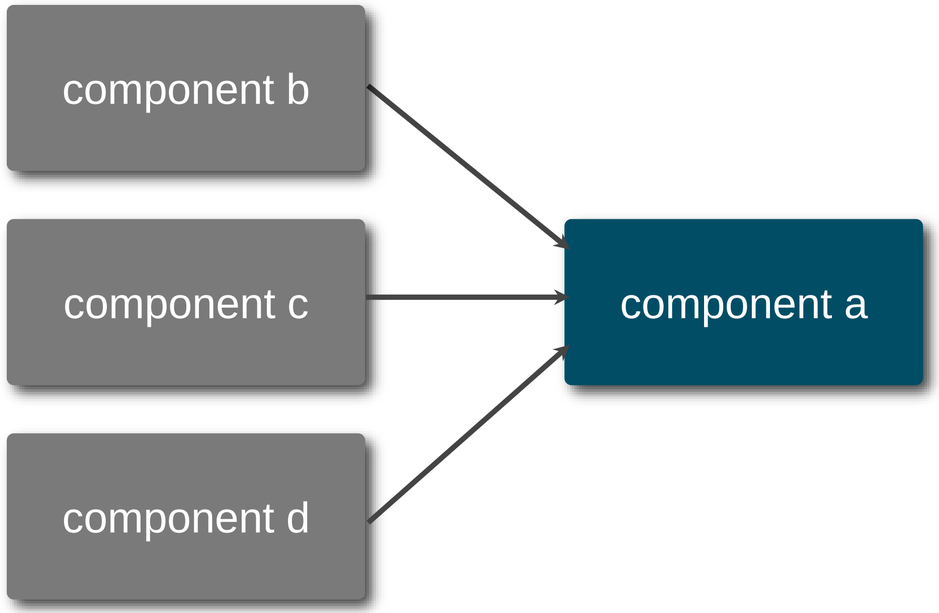
the extent to which components know about each other



component coupling

afferent coupling

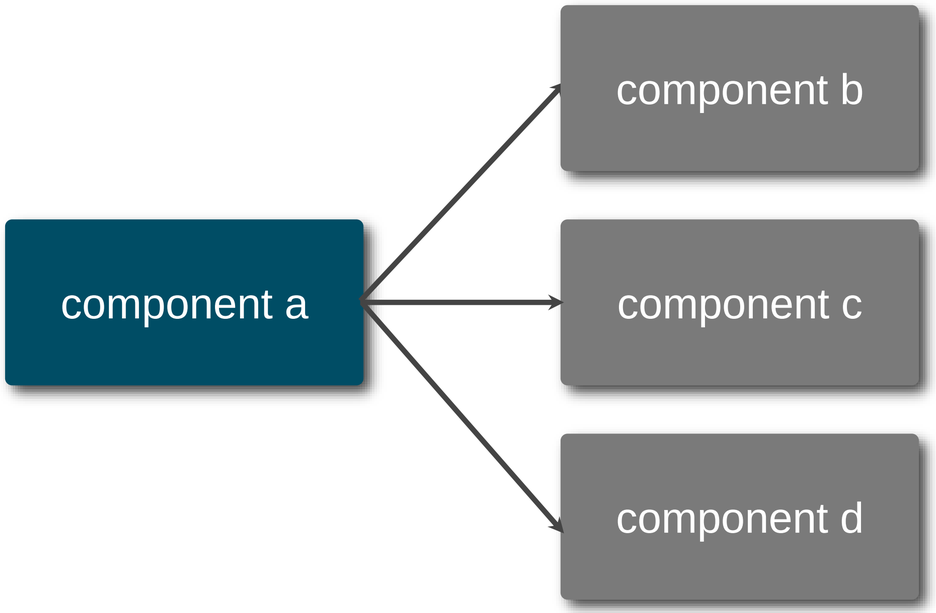
the degree to which other components are dependent on the target component



component coupling

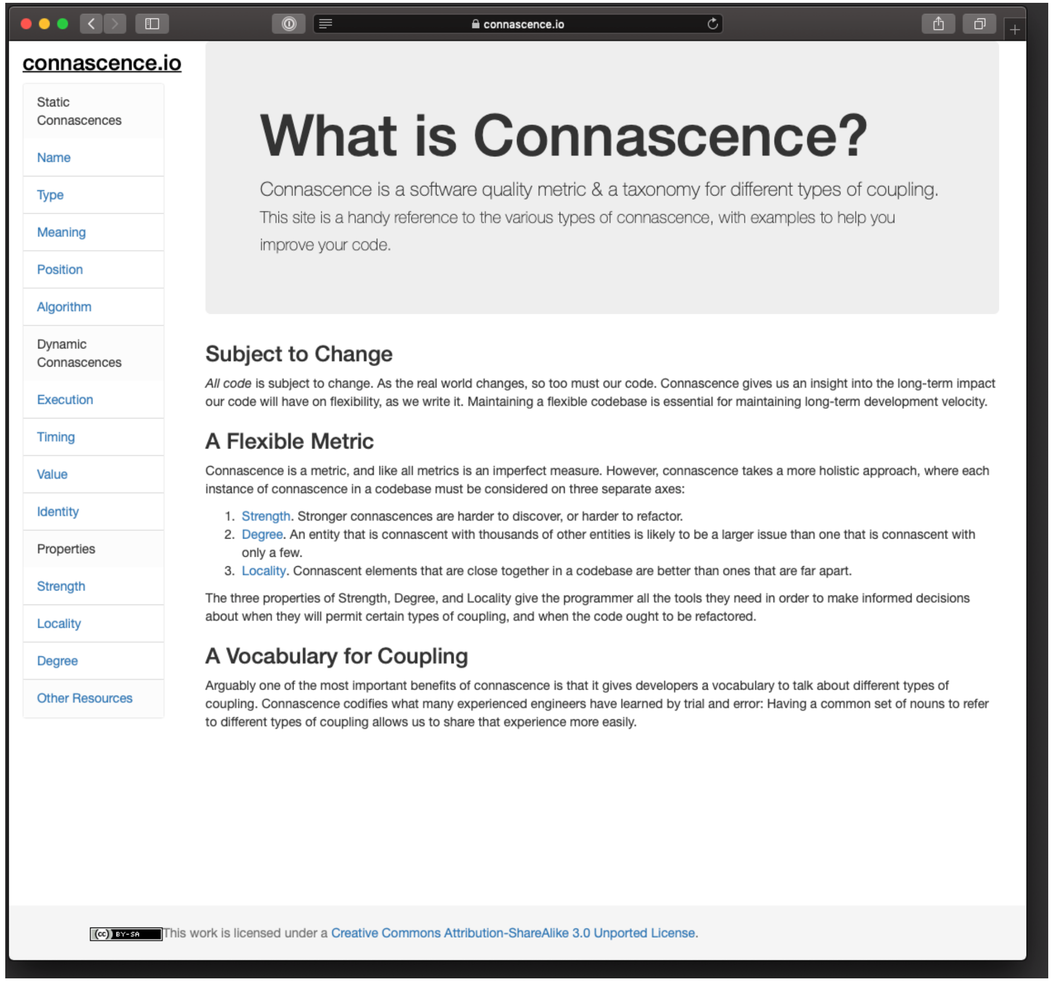
efferent coupling

the degree to which the target component is dependent on other components

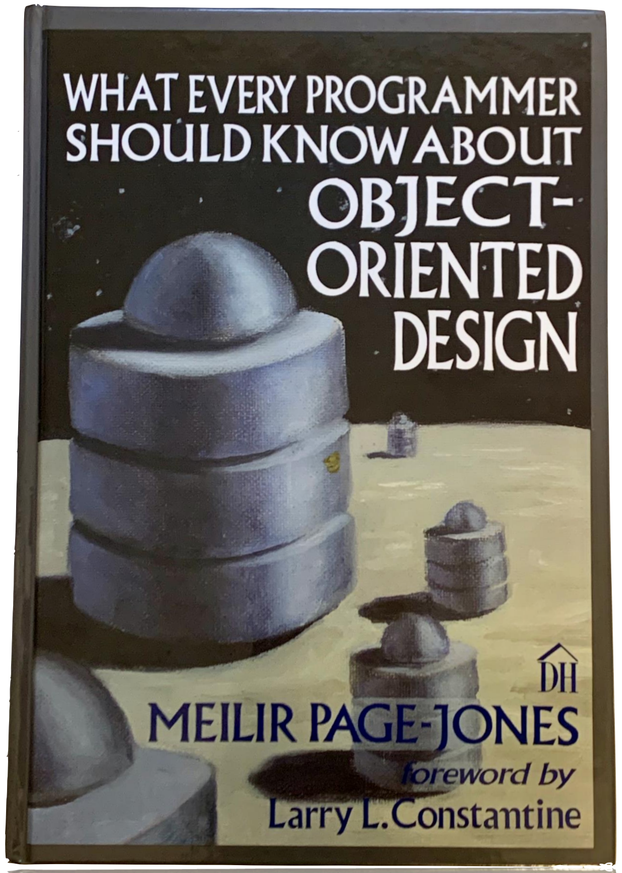


connascence

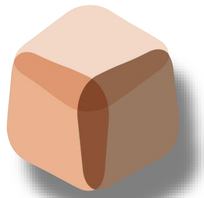
https://connascence.io/name.html



connascence



Two components are connascent if a change in one would require the other to be modified in order to maintain the overall correctness of the system.



static

dynamic

connascence

name

type

meaning

algorithm

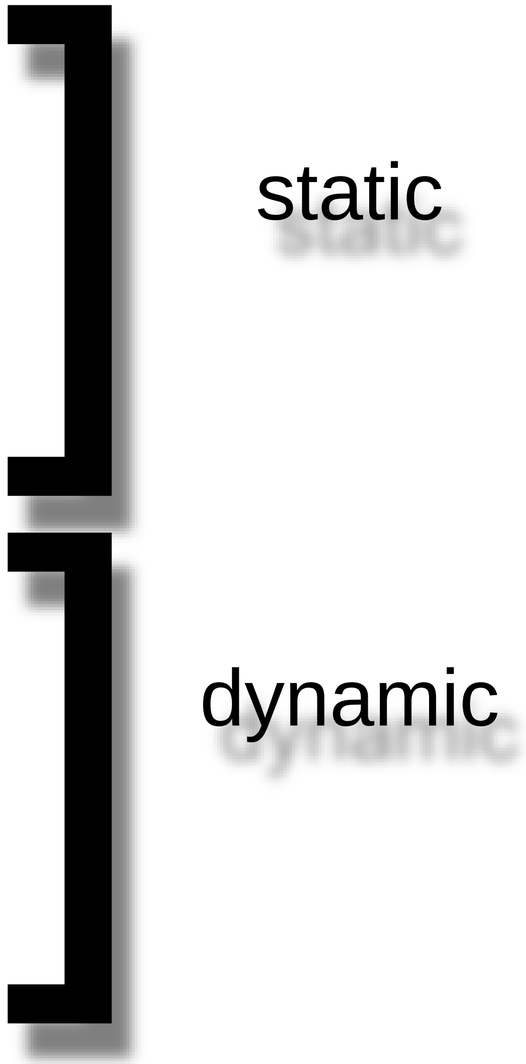
position

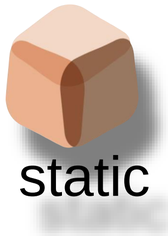
execution order

timing

value

identity

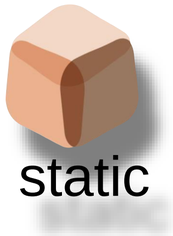




connascence of meaning

multiple components must agree on the meaning of particular values

```
def valid_credit_card_number?(cc_number)
  # check for "test" credit card numbers
  if cc_number == "9999-9999-9999-9999"
    # test verification
  else
    # some more code
  end
end
```



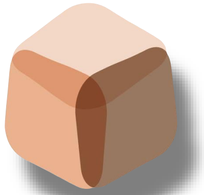
connascence of meaning

multiple components must agree on the meaning of particular values

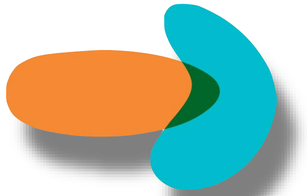
```
def valid_credit_card_number?(cc_number)
  # check for "test" credit card numbers
  if cc_number == "9999-9999-9999-9999"
    # test verification
  else
    # some more code
  end
end
```

How would you fix this ?!?

connascence properties



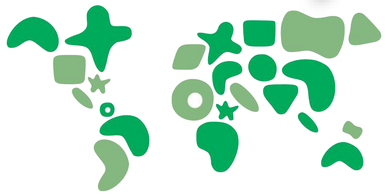
static



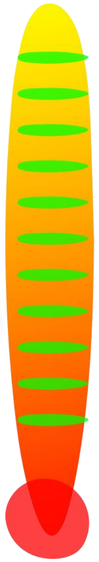
dynamic

Strength

Locality

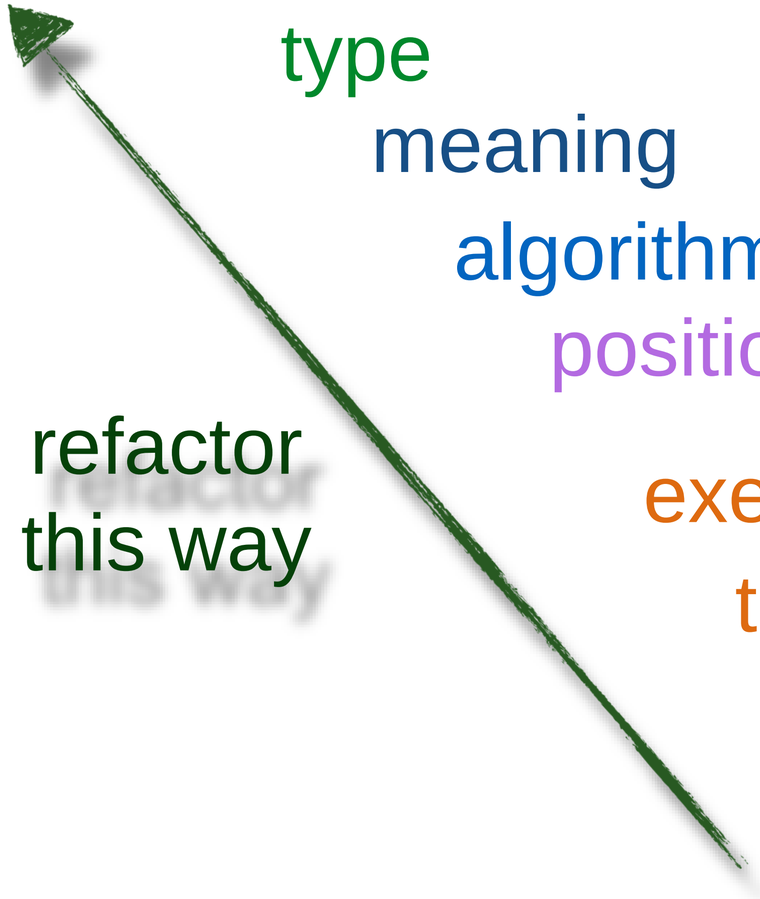


Degree



connascence properties

Strength



name

type

meaning

algorithm

position

refactor
this way

execution order

timing

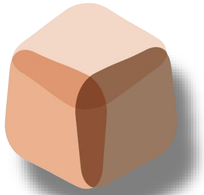
value

identity

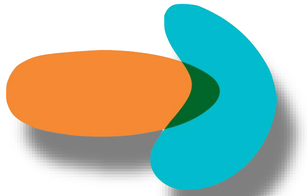
static

dynamic

connascence properties



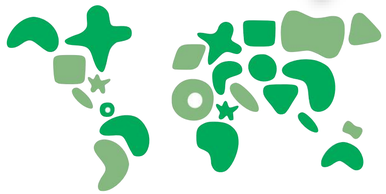
static



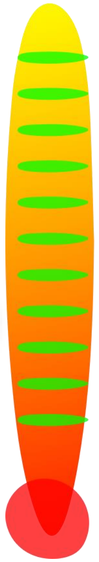
dynamic

Strength

Locality

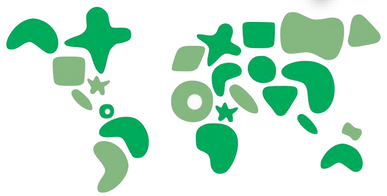


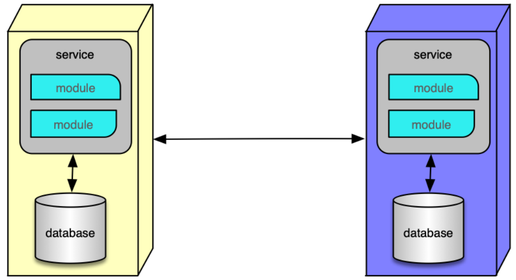
Degree



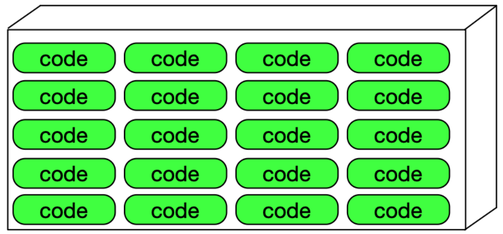
connascence properties

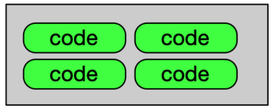
Locality





Proximal code (in the same module, class, or function) should have more & higher forms of connascence than less proximal code (in separate modules, or even codebases).



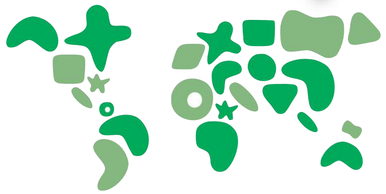


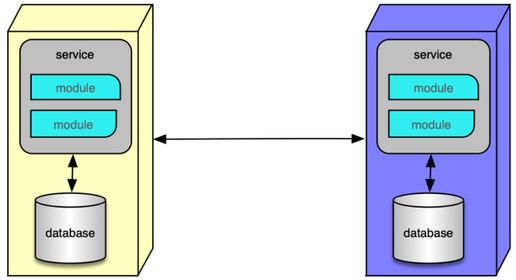
static

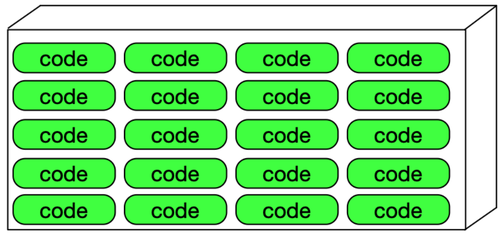
dynamic

connascence properties

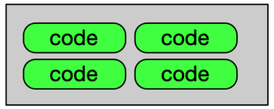
Locality

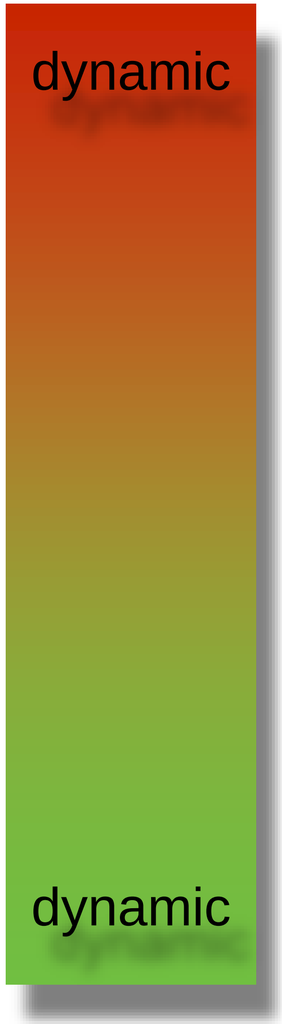




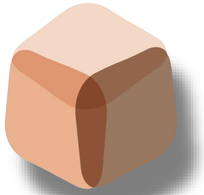


locality matters!

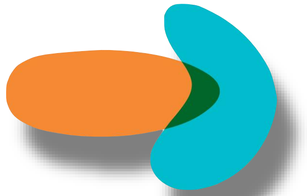




connascence properties



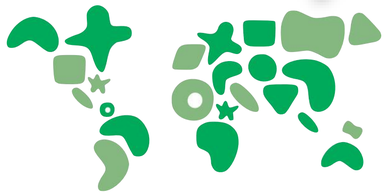
static



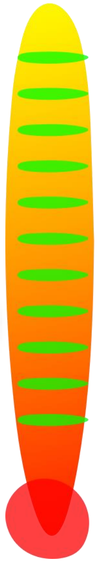
dynamic

Strength

Locality

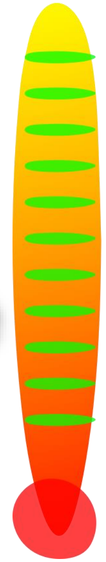


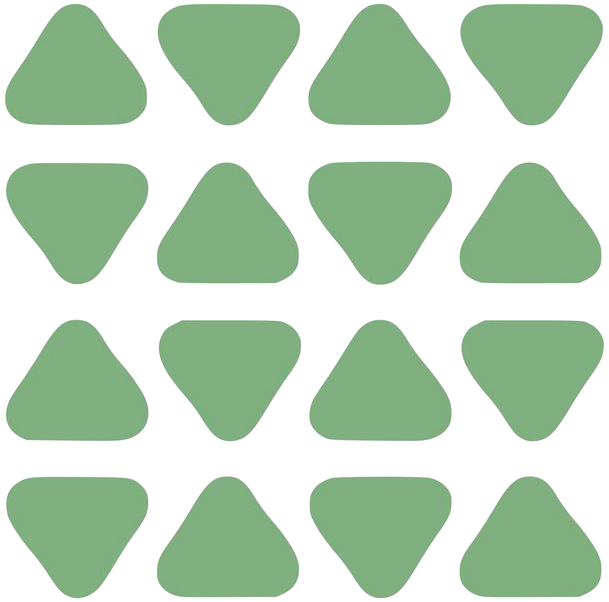
Degree

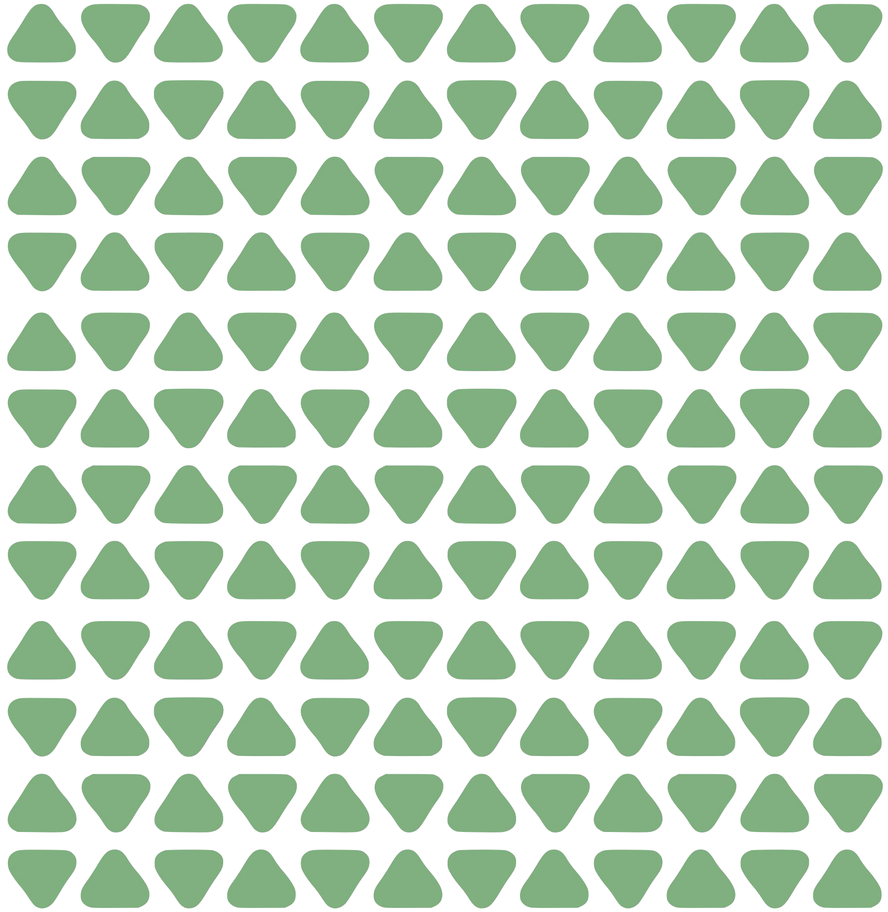


connascence properties

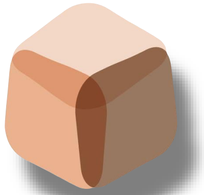
Degree



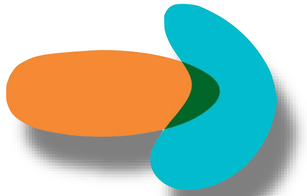




Connascence Properties



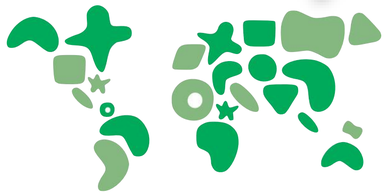
static



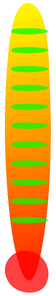
dynamic

Strength

Locality

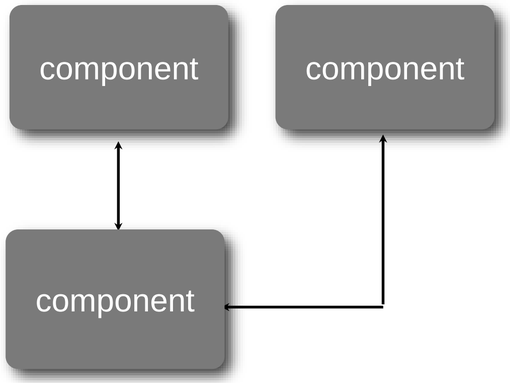


Degree



Component Coupling

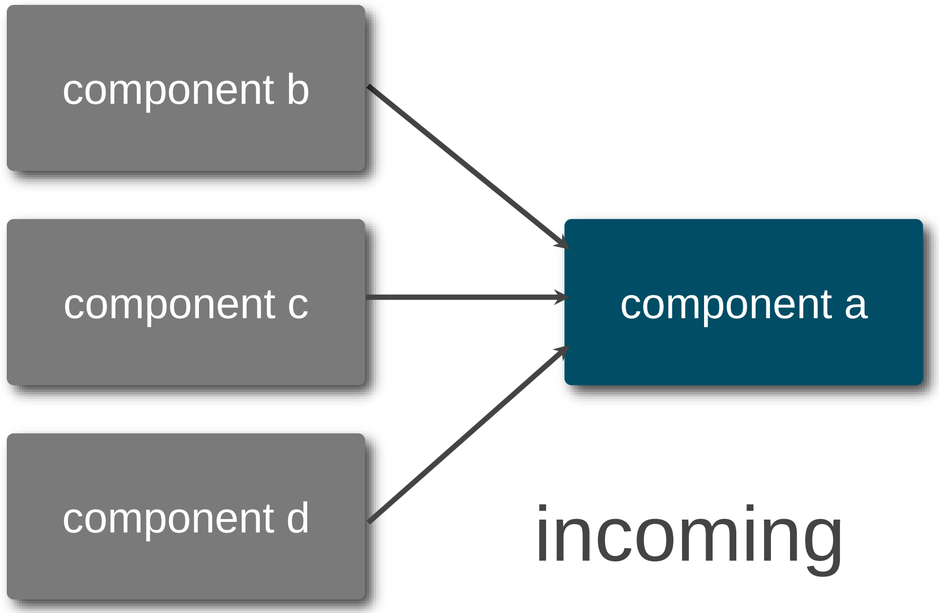
the extent to which components know about each other



Component Coupling

afferent coupling

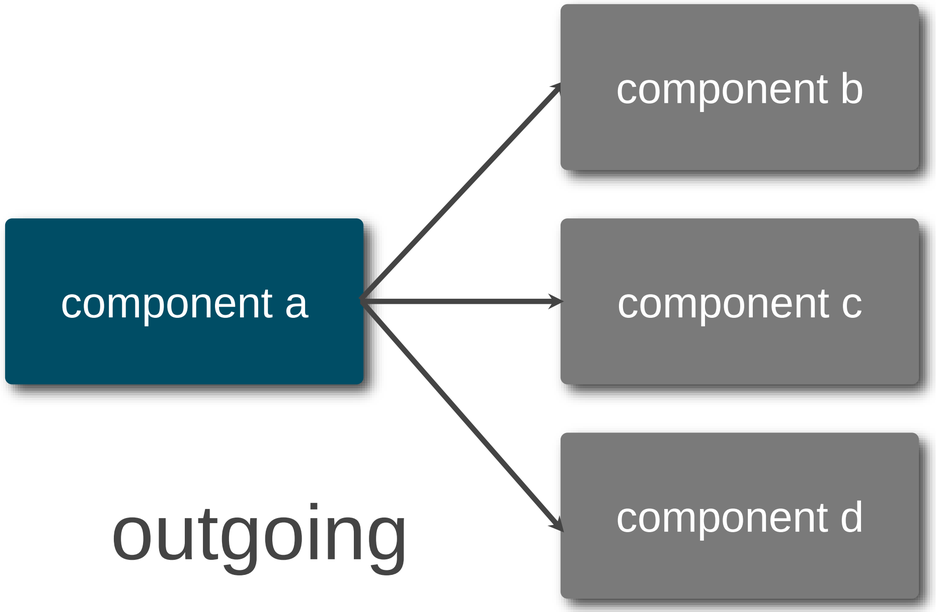
the degree to which other components are dependent on the target component



Component Coupling

efferent coupling

the degree to which the target component is dependent on other components



dependency metrics

abstractness

$$A = \frac{\sum m^a}{\sum m^c + \sum m^a}$$

where:
m^a => abstract elements
m^c => concrete elements

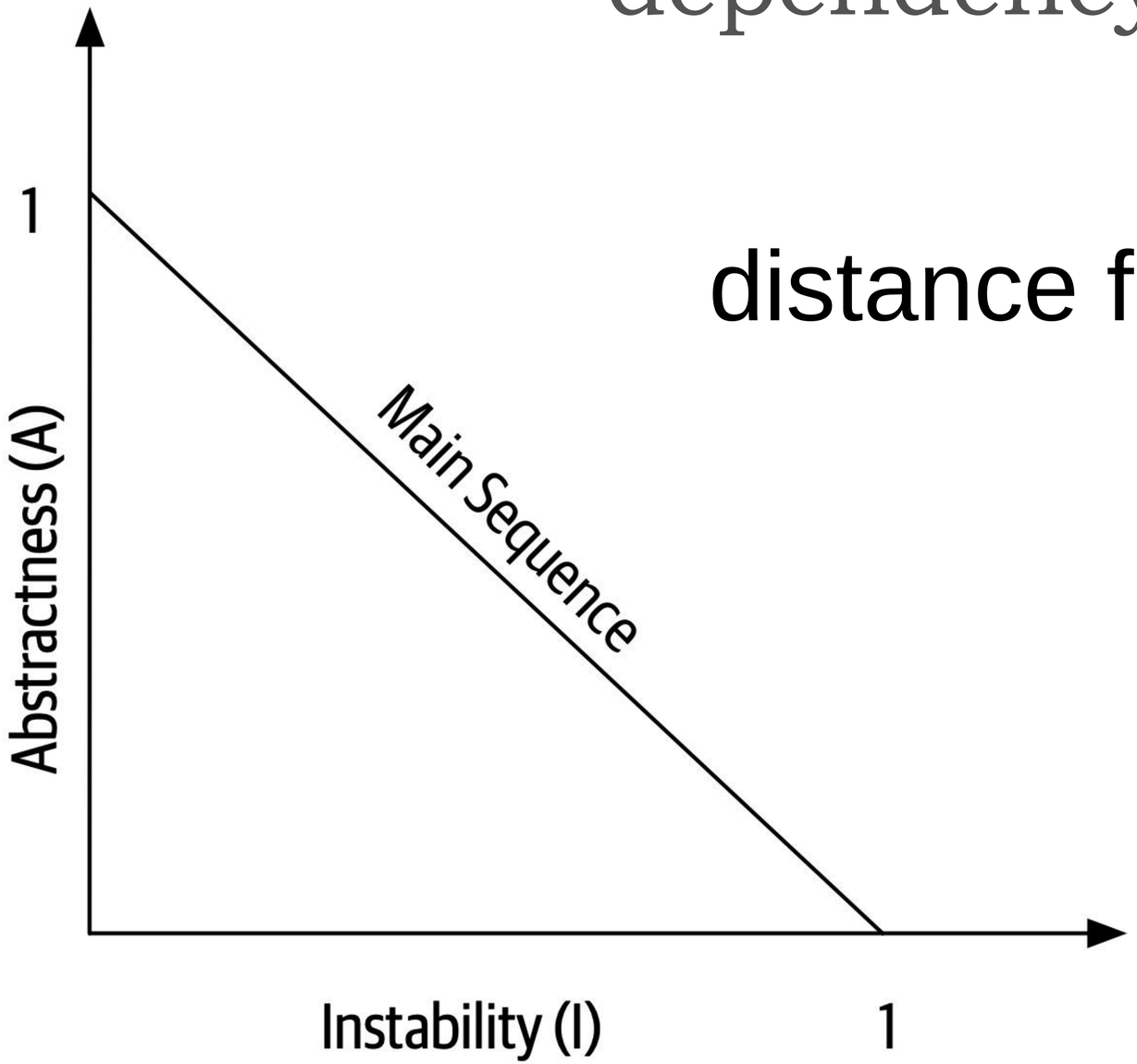
dependency metrics

instability

$$I = \frac{C^e}{C^e + C^a}$$

where:
c^e => efferent coupling
c^a => afferent coupling

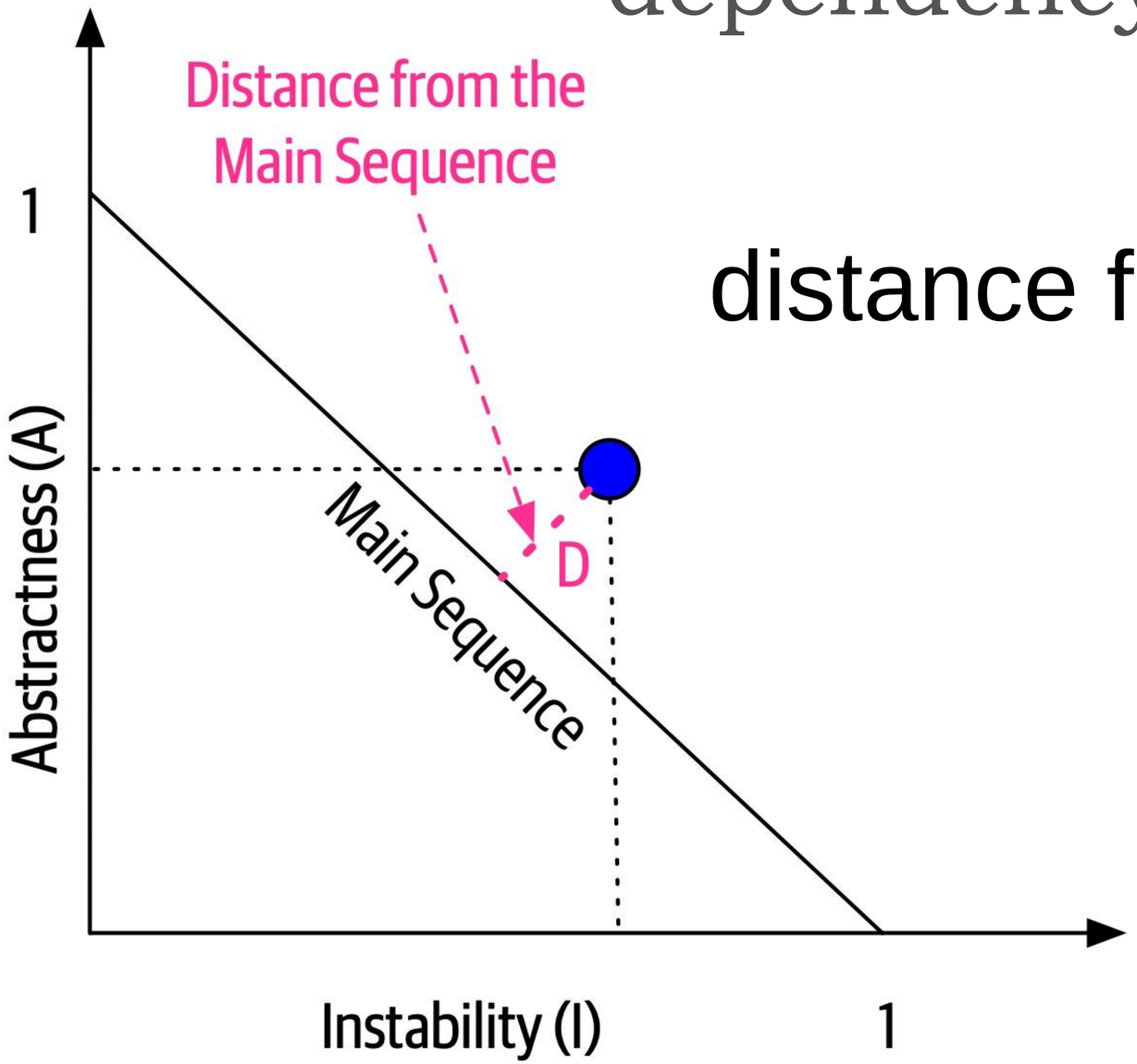
dependency metrics



distance from the main sequence

$$D = |A + I - 1|$$

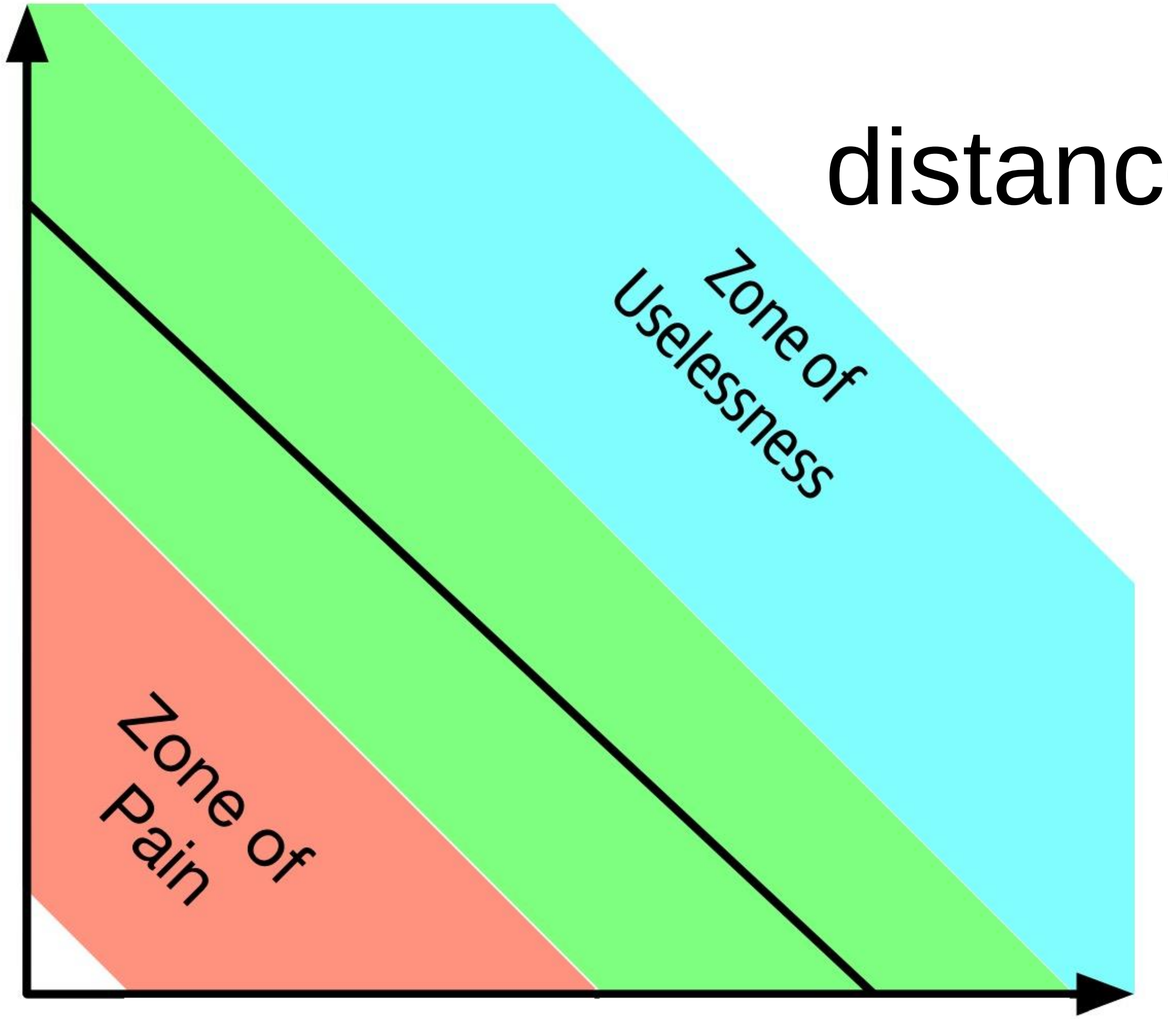
dependency metrics



distance from the main sequence

$$D = |A + I - 1|$$

dependency metrics



distance from the main sequence

$$D = |A + I - 1|$$

distance from the main sequence

component cohesion

component cohesion

the degree and manner to which the operations of a component are related to one another

metrics ∩ architecture characteristics

Chidamber & Kemerer Metrics

✓DIT (depth of inheritance tree)

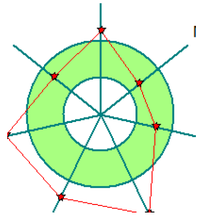
✓WMC (weighted methods/class; sum of CC)

✓CE (efferent coupling count)

✓CA (afferent coupling count)

metrics ∩ architecture characteristics

architecture characteristics mapping



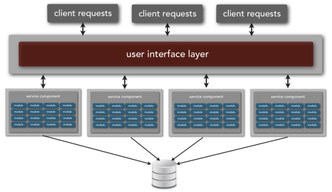
component size

complexity / WMC

coupling (CE, CA, CT)

inheritance depth (DIT)

percent comments



modularity

maintainability

testability

availability

deployment

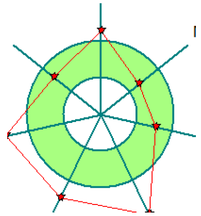
reliability

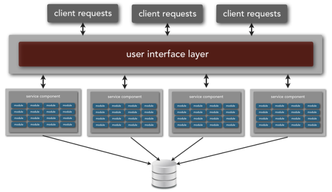
scalability

evolutionary / migration

metrics ∩ architecture characteristics

architecture characteristics mapping





component size

complexity / WMC

coupling (CE, CA, CT)

inheritance depth (DIT)

percent comments

modularity

maintainability

testability

availability

deployment

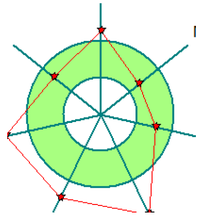
reliability

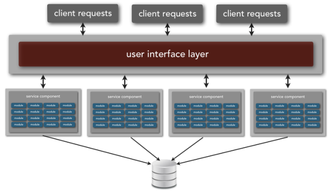
scalability

evolutionary / migration

metrics ∩ architecture characteristics

architecture characteristics mapping





component size

complexity / WMC

coupling (CE, CA, CT)

inheritance depth (DIT)

percent comments

modularity

maintainability

testability

availability

deployment

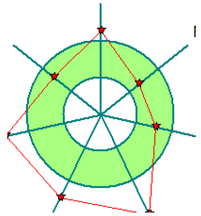
reliability

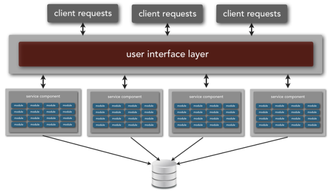
scalability

evolutionary / migration

metrics ∩ architecture characteristics

architecture characteristics mapping





component size

complexity / WMC

coupling (CE, CA, CT)

inheritance depth (DIT)

percent comments

modularity

maintainability

testability

availability

deployment

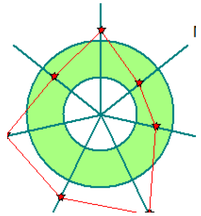
reliability

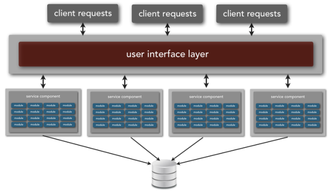
scalability

evolutionary / migration

metrics ∩ architecture characteristics

architecture characteristics mapping





component size

complexity / WMC

coupling (CE, CA, CT)

inheritance depth (DIT)

percent comments

modularity

maintainability

testability

availability

deployment

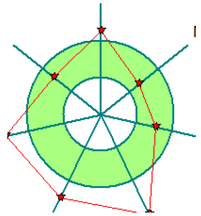
reliability

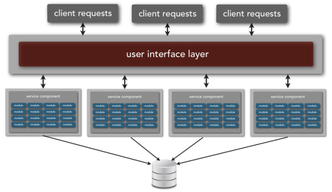
scalability

evolutionary / migration

metrics ∩ architecture characteristics

architecture characteristics mapping





component size

complexity / WMC

coupling (CE, CA, CT)

inheritance depth (DIT)

percent comments

modularity

maintainability

testability

availability

deployment

reliability

scalability

evolutionary / migration

continuous delivery ∩ architecture

good engineering practices help preserve important architectural characteristics

continuous delivery ∩ architecture

good engineering practices help preserve important architectural characteristics

awareness of engineering informs architectural decisions

continuous delivery ∩ architecture

good engineering practices help preserve important architectural characteristics

awareness of engineering informs architectural decisions

better component isolation allows for faster cycle time

Question:

How do teams doing continuous deployment handle things like release branches?

Question:

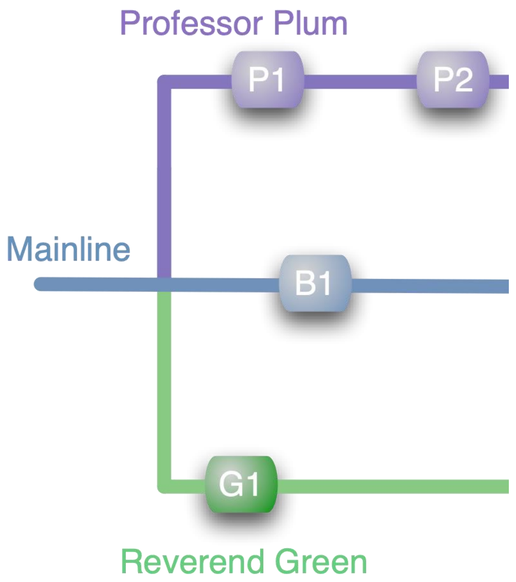
How can teams support advanced architecture tricks like A/B testing while doing continuous delivery?

Question:

How can teams make structural changes to architecture (*continuously!*) without breaking other parts by unexpected side effects?!?

Question:

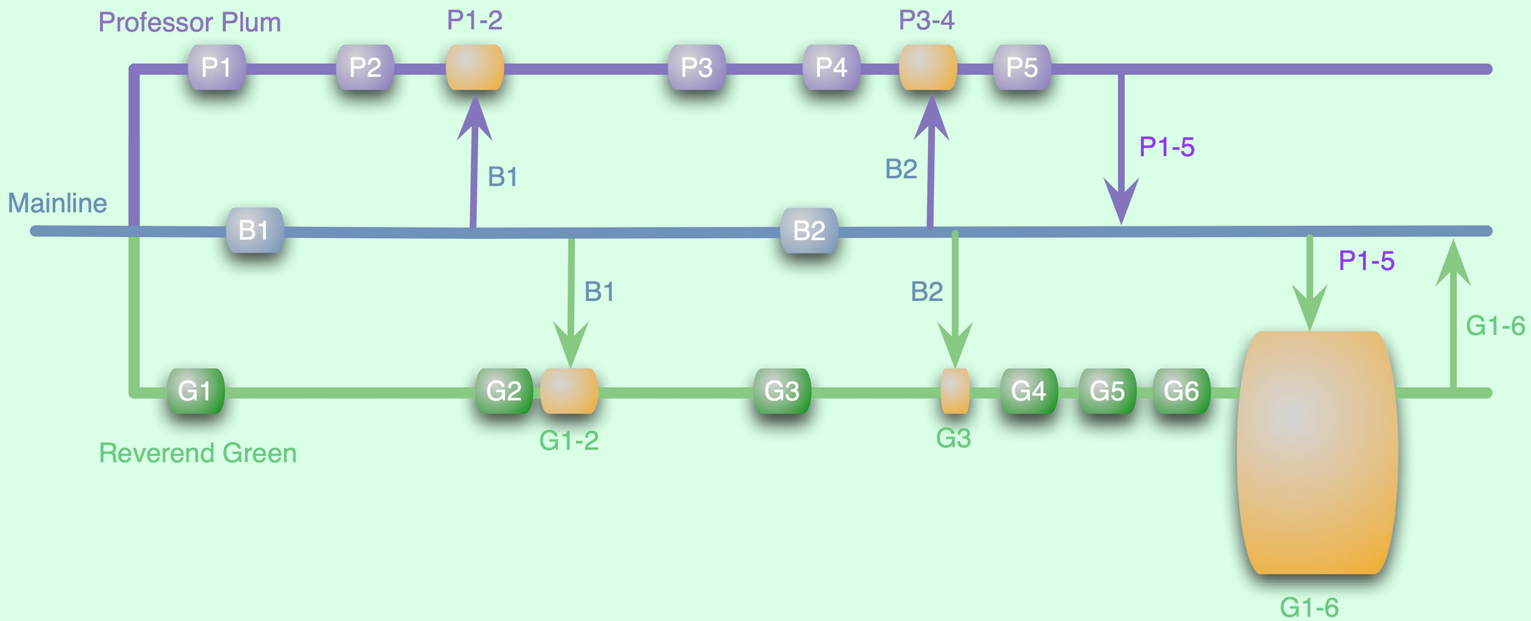
How do teams reduce risk when doing continuous delivery/deployment?



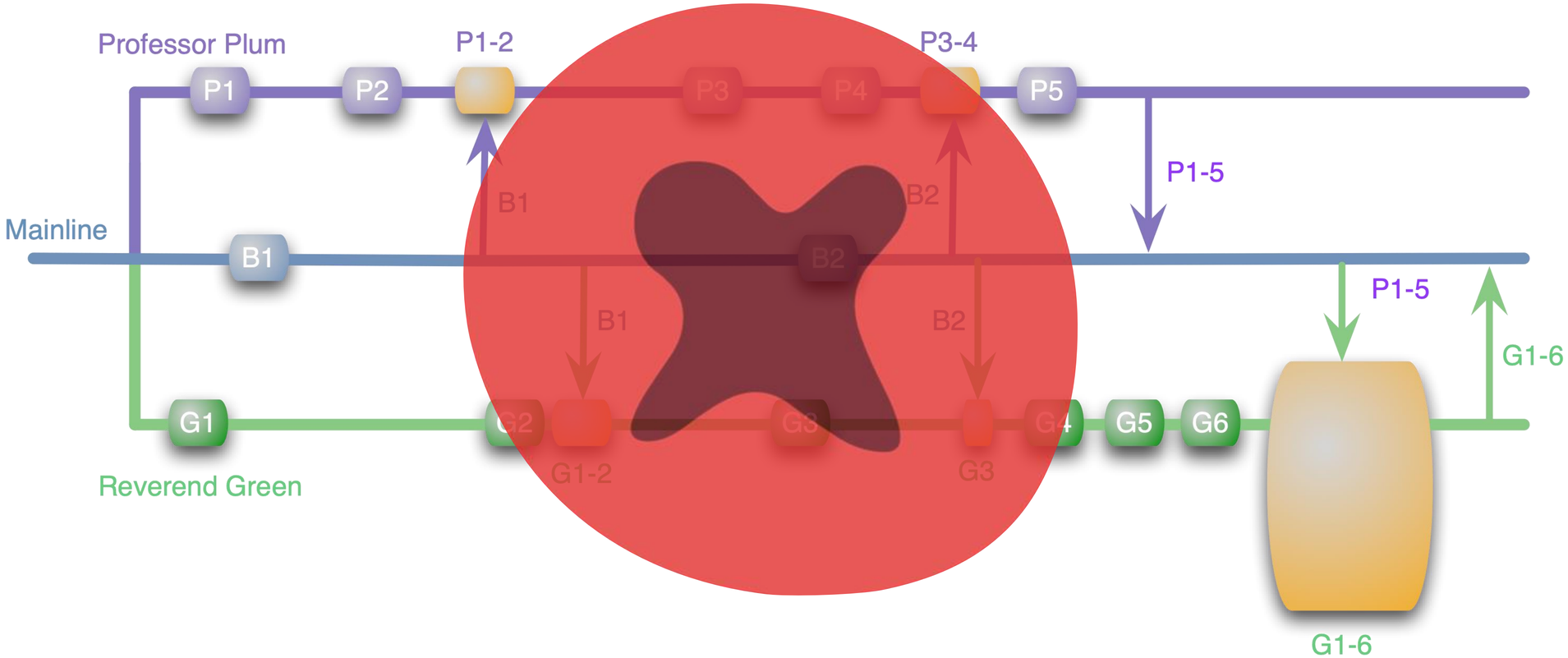
merge ambush!

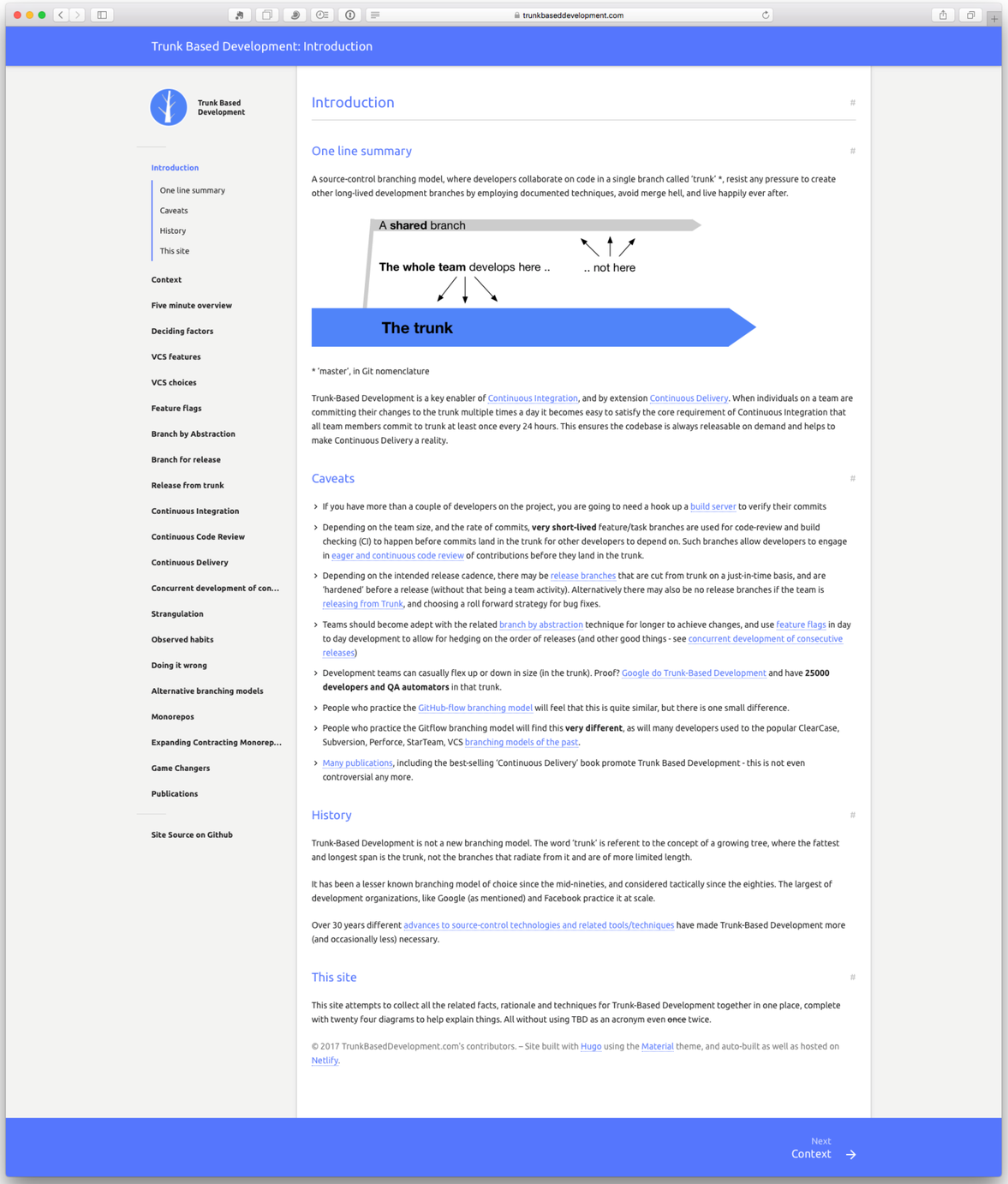
Feature Branching

trunk-based development versus feature branching



trunk-based development





Paul Hammant

trunk-based development

https://trunkbaseddevelopment.com

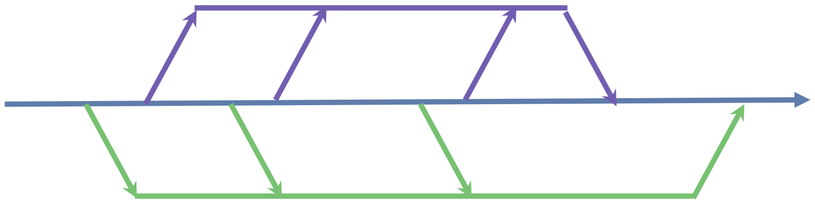
trunk-based development

does it!!

https://cacm.acm.org/magazines/2016/7/204032-why-google-stores-billions-of-lines-of-code-in-a-single-repository/fulltext

don’t panic!

Feature Branching



1

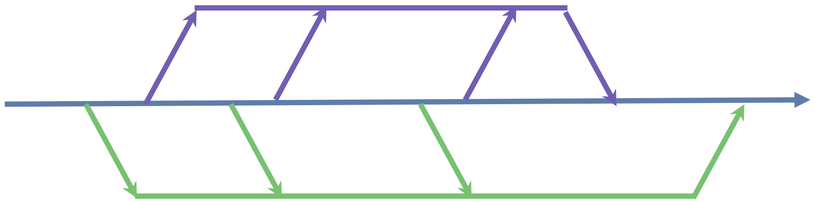
Big Scary Merge

Cherry Picking

Continuous Integration

Untrusted Contributors

Feature Branching



1 ***Big Scary Merge***

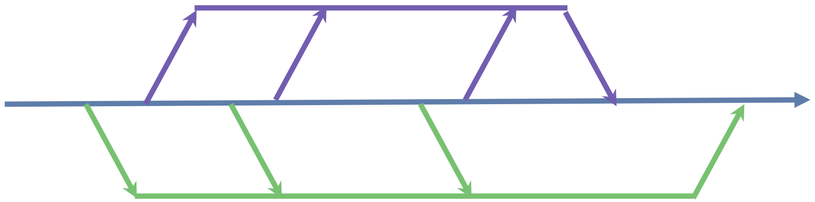
2 ***Discouraging refactoring***

Cherry Picking

Untrusted Contributors

Continuous Integration

Feature Branching



1 ***Big Scary Merge***

2 ***Discouraging refactoring***

3 ***Hard to combine features***

Cherry Picking

Untrusted Contributors

Continuous Integration

http://martinfowler.com/articles/feature-toggles.html

martinfowler.com

MARTIN FOWLER

Feature Toggles

Feature toggles are a powerful technique, allowing teams to modify system behavior without changing code. They fall into various usage categories, and it's important to take that categorization into account when implementing and managing toggles. Toggles introduce complexity. We can keep that complexity in check by using smart toggle implementation practices and appropriate tools to manage our toggle configuration, but we should also aim to constrain the number of toggles in our system.

08 February 2016

Pete Hodgson

Pete Hodgson is a consultant at ThoughtWorks, where he's spent the last few years helping teams become awesome at sustainable delivery of high-quality software. He's particularly passionate about web-based mobile, ruby, agile, functional approaches to software, and beer.

Find **similar articles** to this by looking at these tags: delivery · application architecture · continuous integration

Contents expand

"Feature Toggling" is a set of patterns which can help a team to deliver new functionality to users rapidly but safely. In this article on Feature Toggling we'll start off with a short story showing some typical scenarios where Feature Toggles are helpful. Then we'll dig into the details, covering specific patterns and practices which will help a team succeed with Feature Toggles.

A Toggling Tale

Picture the scene. You're on one of several teams working on a sophisticated town planning simulation game. Your team is responsible for the core simulation engine. You have been tasked with increasing the efficiency of the Spline Reticulation algorithm. You know this will require a fairly large overhaul of the implementation which will take several weeks. Meanwhile other members of your team will need to

before

```
function reticulateSplines(){
  // current implementation lives here
}
```

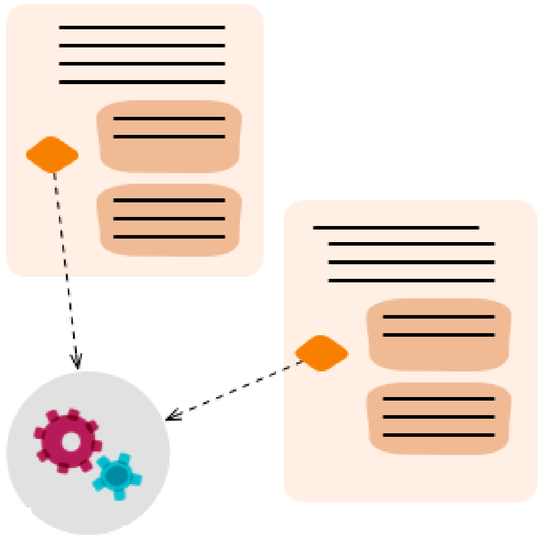
these examples all use JavaScript ES2015

after

```
function reticulateSplines(){
  var useNewAlgorithm = false;
  // useNewAlgorithm = true; // UNCOMMENT IF YOU ARE WORKING ON THE NEW SR ALGORITHM

  if( useNewAlgorithm ){
    return enhancedSplineReticulation();
  }else{
    return oldFashionedSplineReticulation();
  }
}

function oldFashionedSplineReticulation(){
  // current implementation lives here
}

function enhancedSplineReticulation(){
  // TODO: implement better SR algorithm
}
```

```
function reticulateSplines(){
  if( featureIsEnabled("use-new-SR-algorithm") ){
    return enhancedSplineReticulation();
  }else{
    return oldFashionedSplineReticulation();
  }
}
```

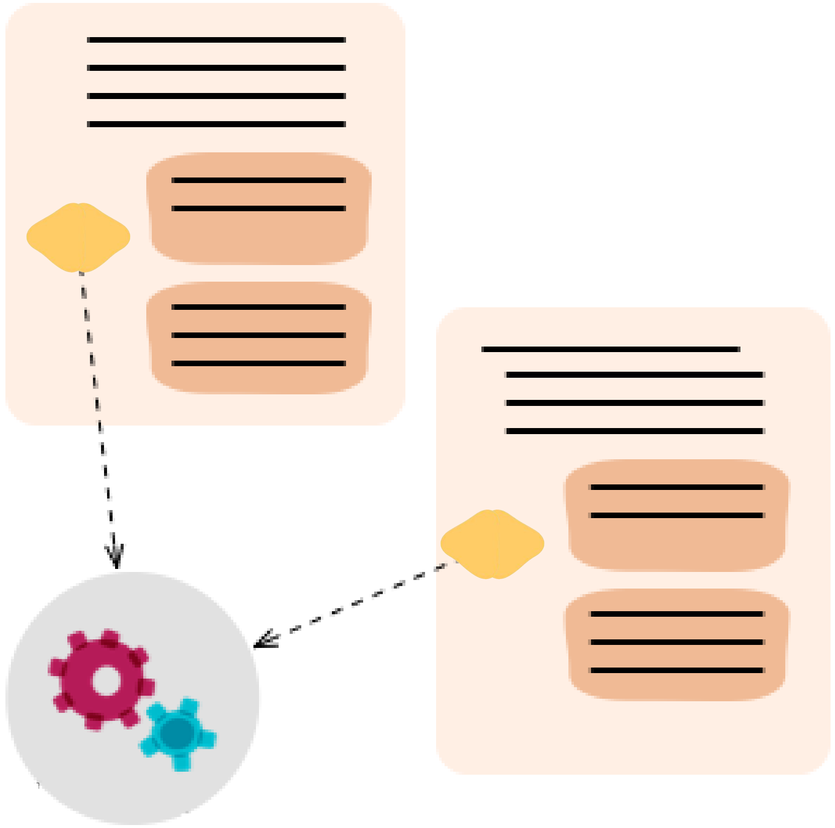
make toggle dynamic

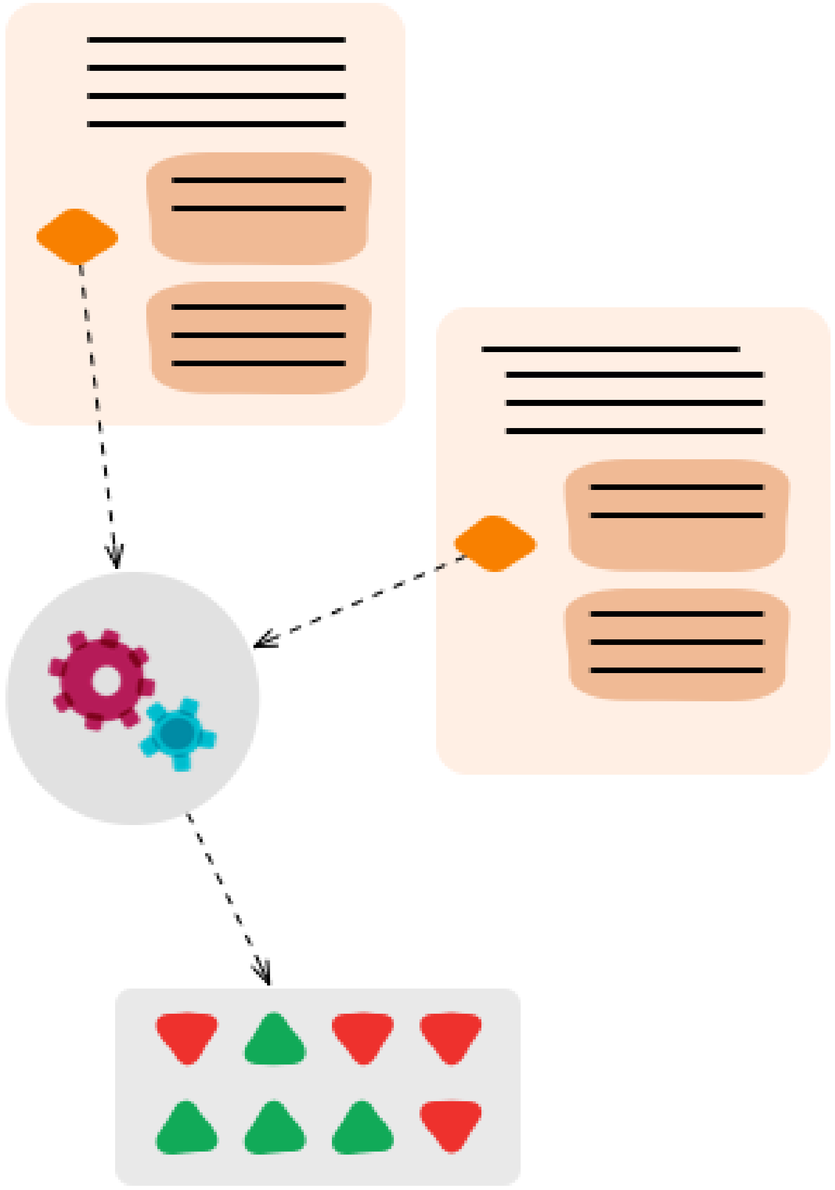


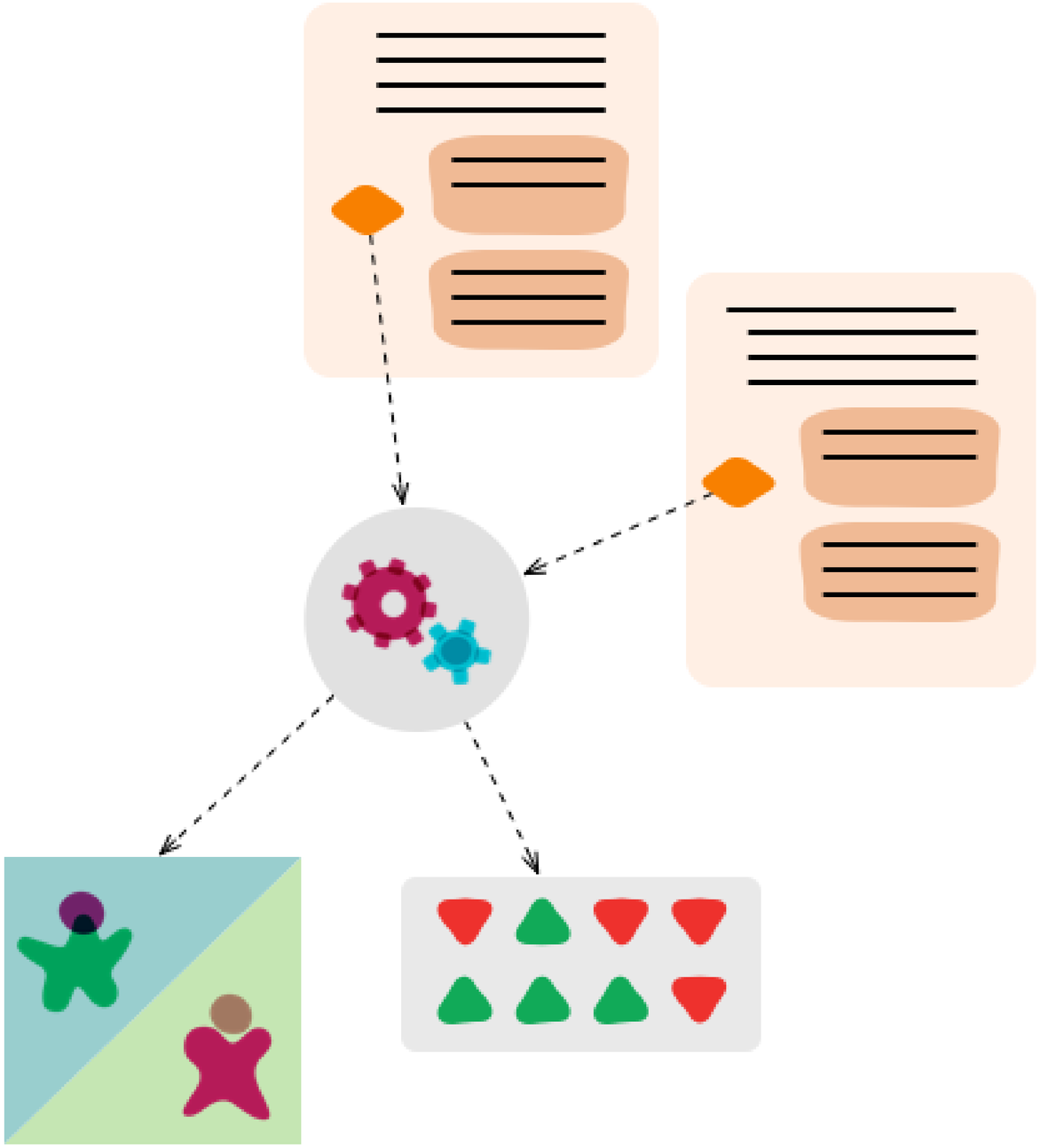
```
function createToggleRouter(featureConfig){
  return {
    setFeature(featureName,isEnabled){
      featureConfig[featureName] = isEnabled;
    },
    featureIsEnabled(featureName){
      return featureConfig[featureName];
    }
  };
}
```

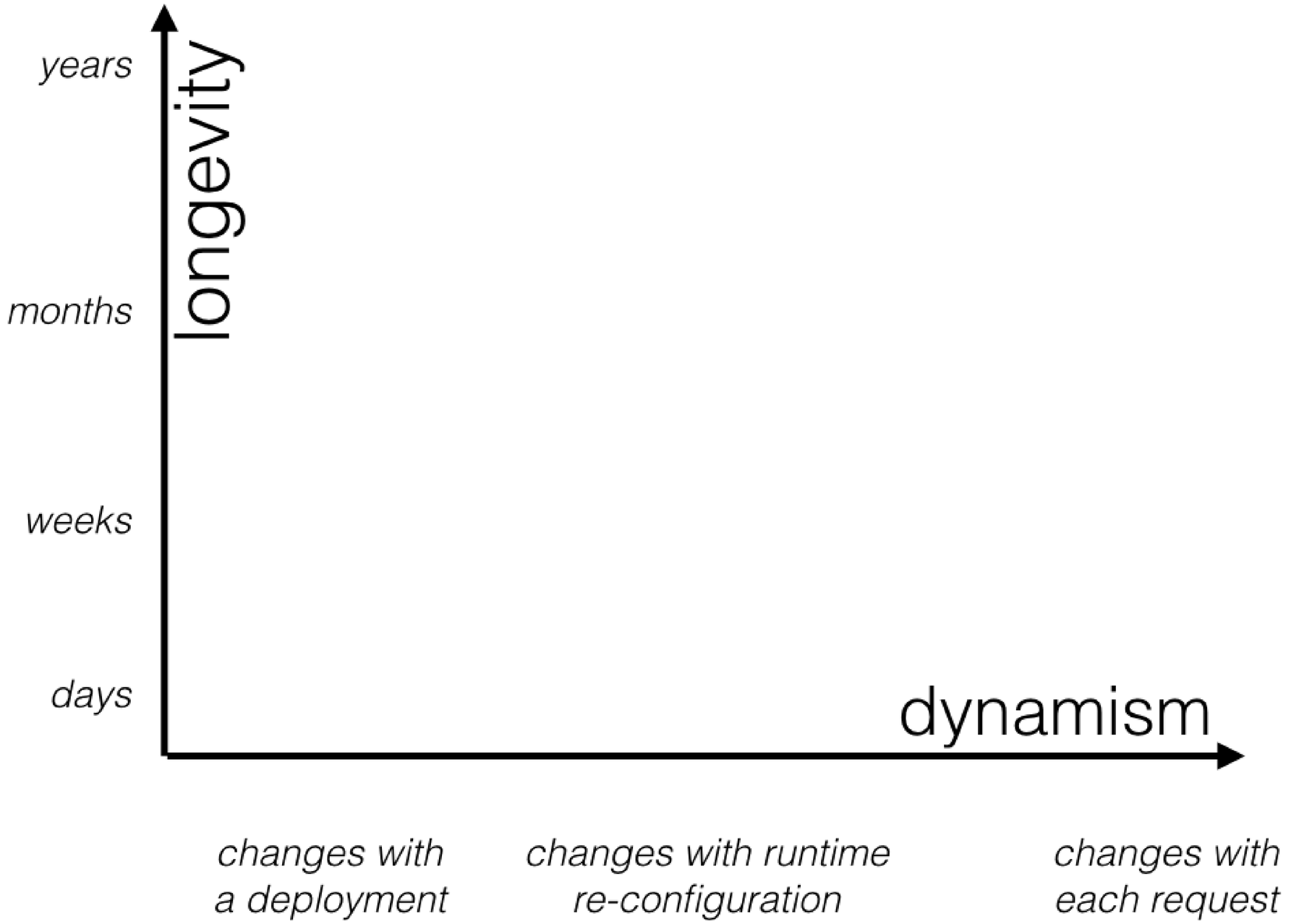
note that we're using ES2015's method shorthand

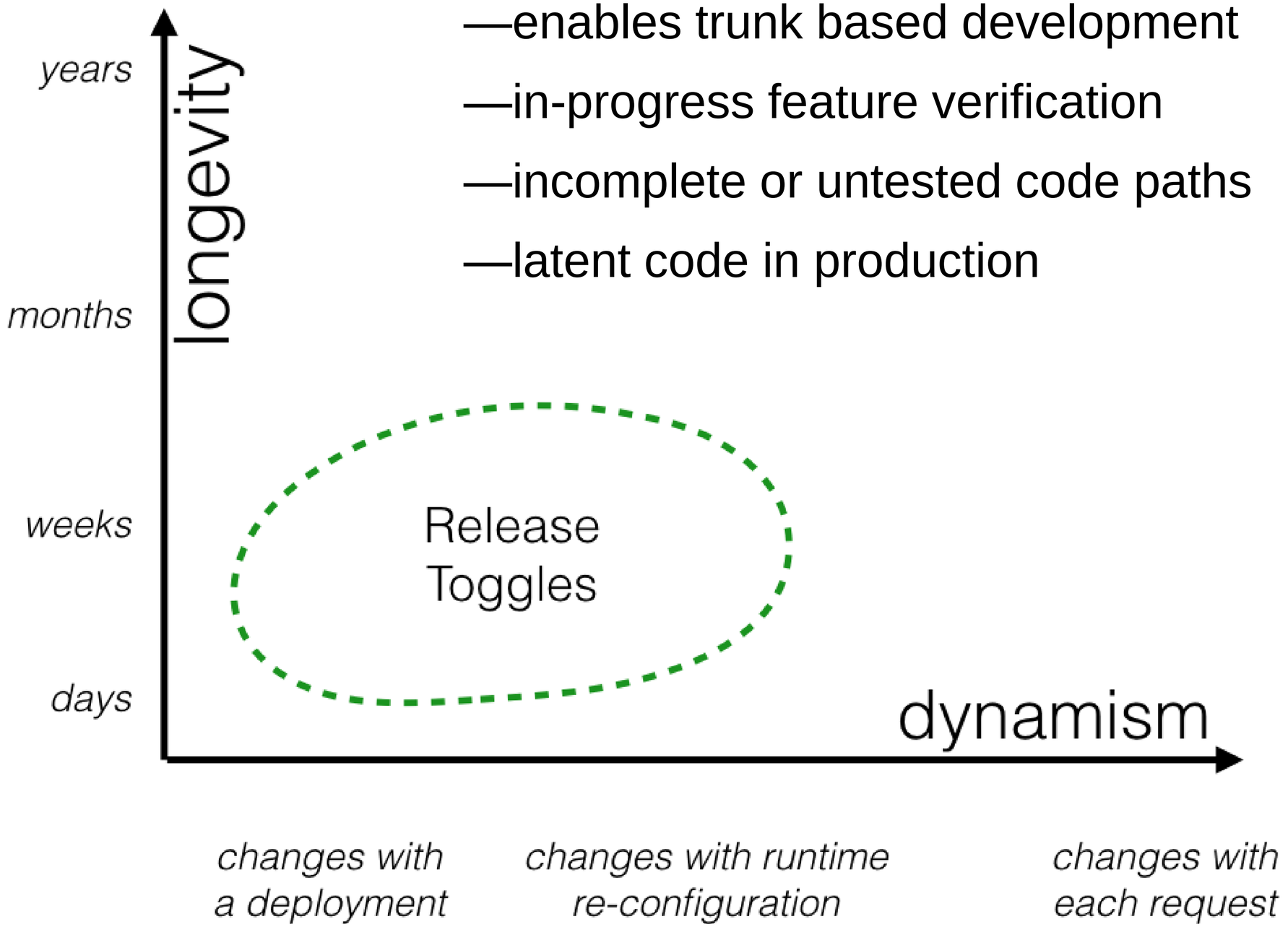
```
describe( 'spline reticulation', function(){
  let toggleRouter;
  let simulationEngine;

  beforeEach(function(){
    toggleRouter = createToggleRouter();
    simulationEngine = createSimulationEngine({toggleRouter:toggleRouter});
  });

  it('works correctly with old algorithm', function(){
    // Given
    toggleRouter.setFeature("use-new-SR-algorithm",false);

    // When
    const result = simulationEngine.doSomethingWhichInvolvesSplineReticulation();

    // Then
    verifySplineReticulation(result);
  });

  it('works correctly with new algorithm', function(){
    // Given
    toggleRouter.setFeature("use-new-SR-algorithm",true);

    // When
    const result = simulationEngine.doSomethingWhichInvolvesSplineReticulation();

    // Then
    verifySplineReticulation(result);
  });
});
```

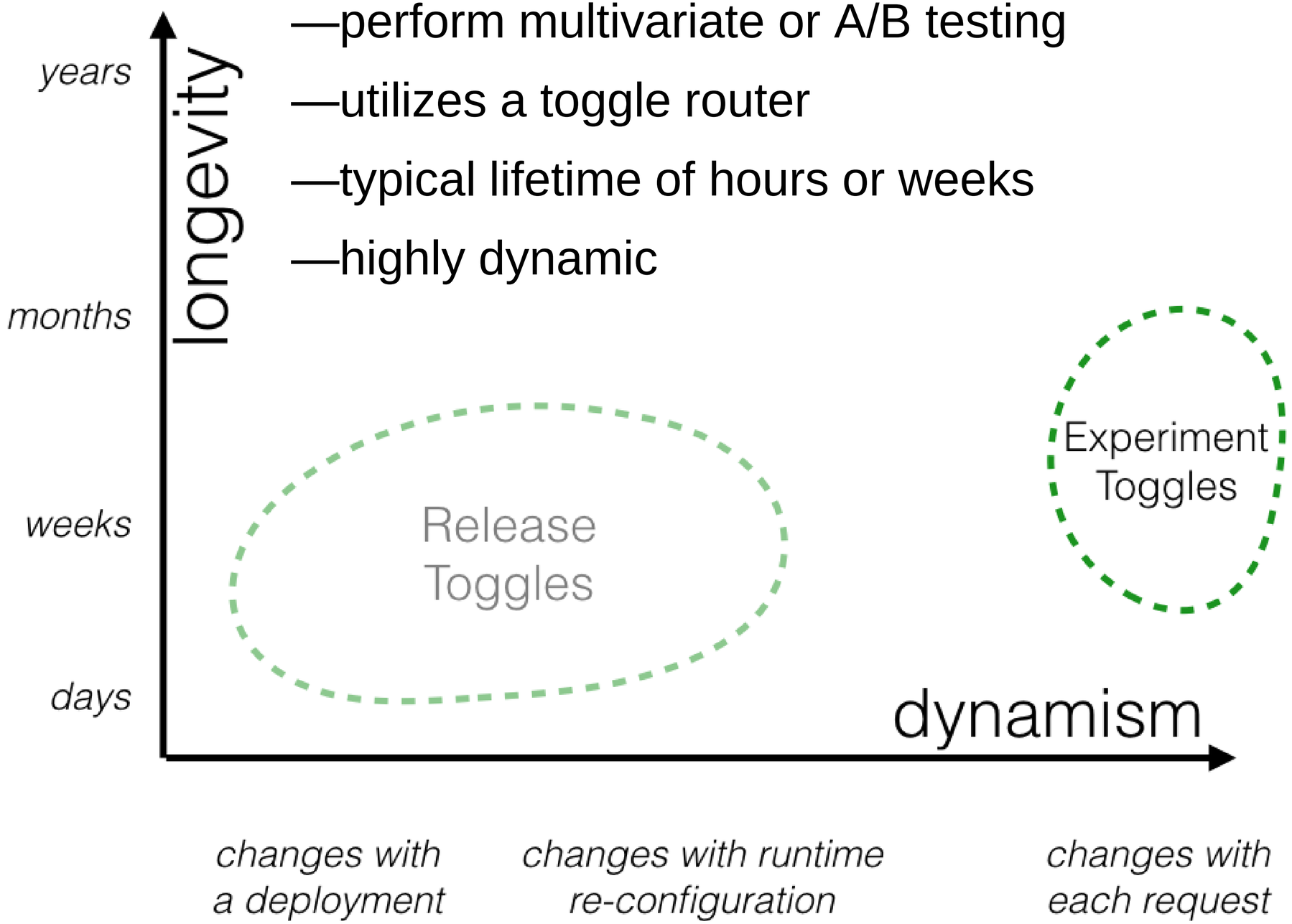


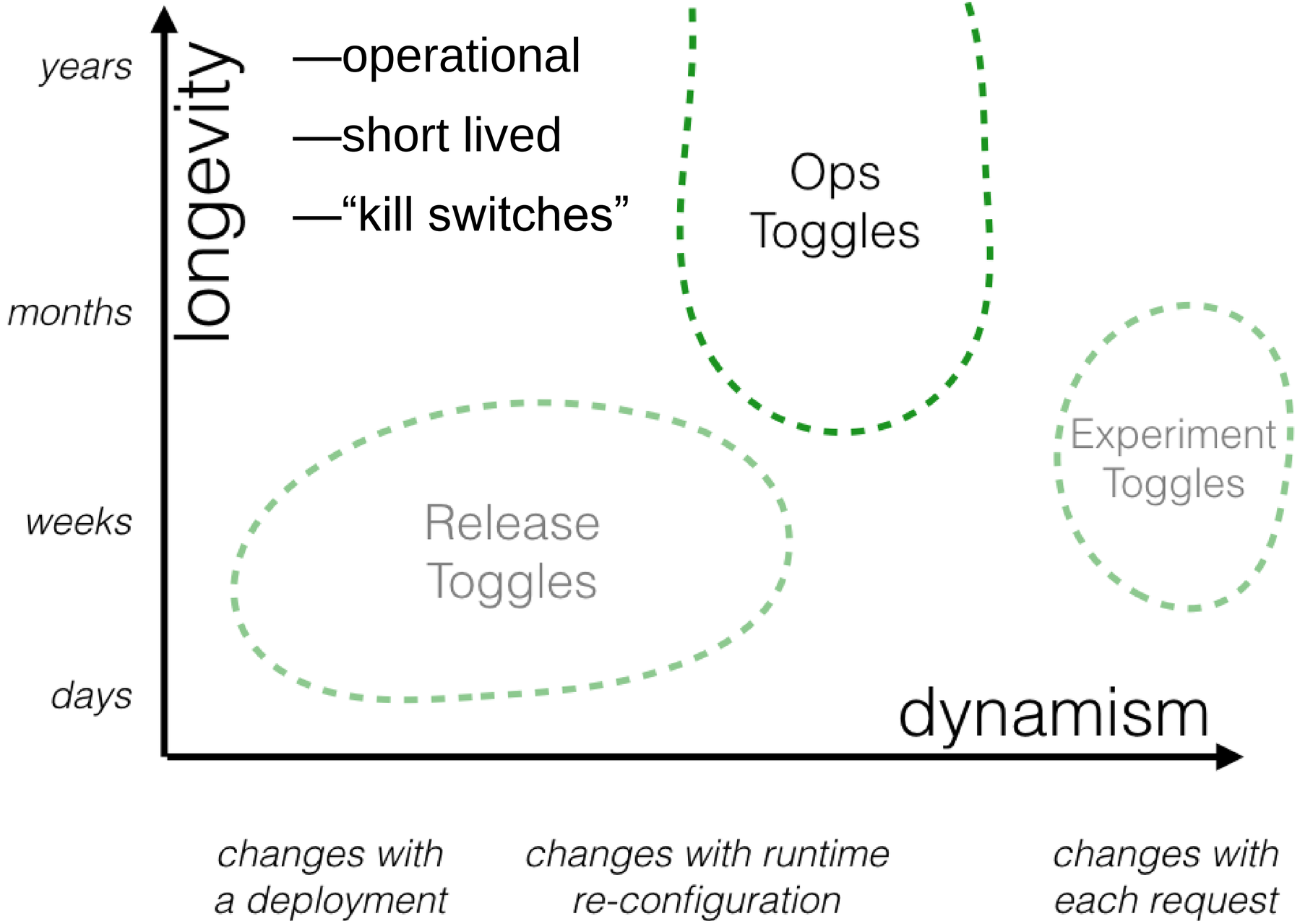


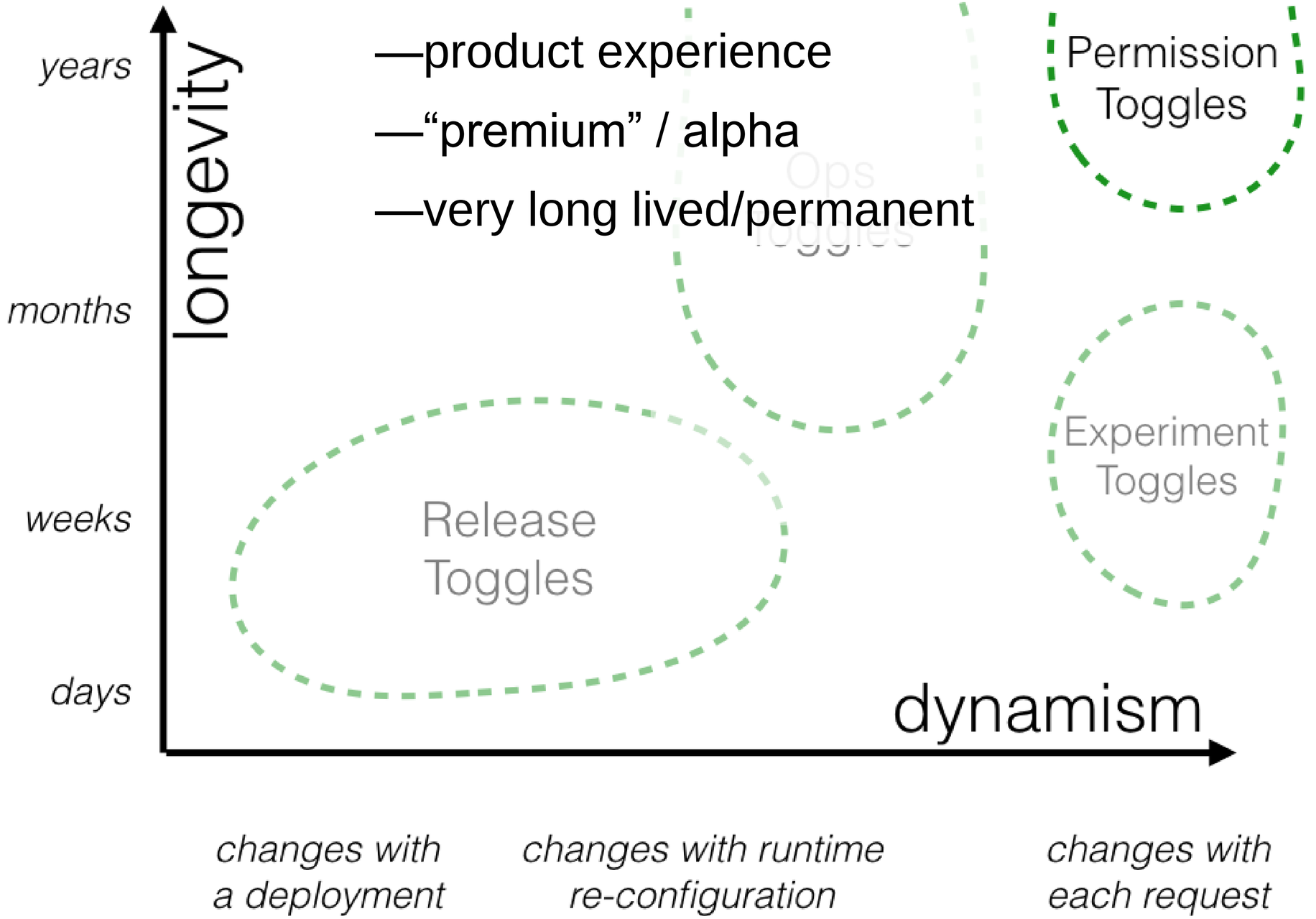



years
months
weeks
days
longevity
dynamism
changes with
a deployment
changes with runtime
re-configuration
changes with
each request

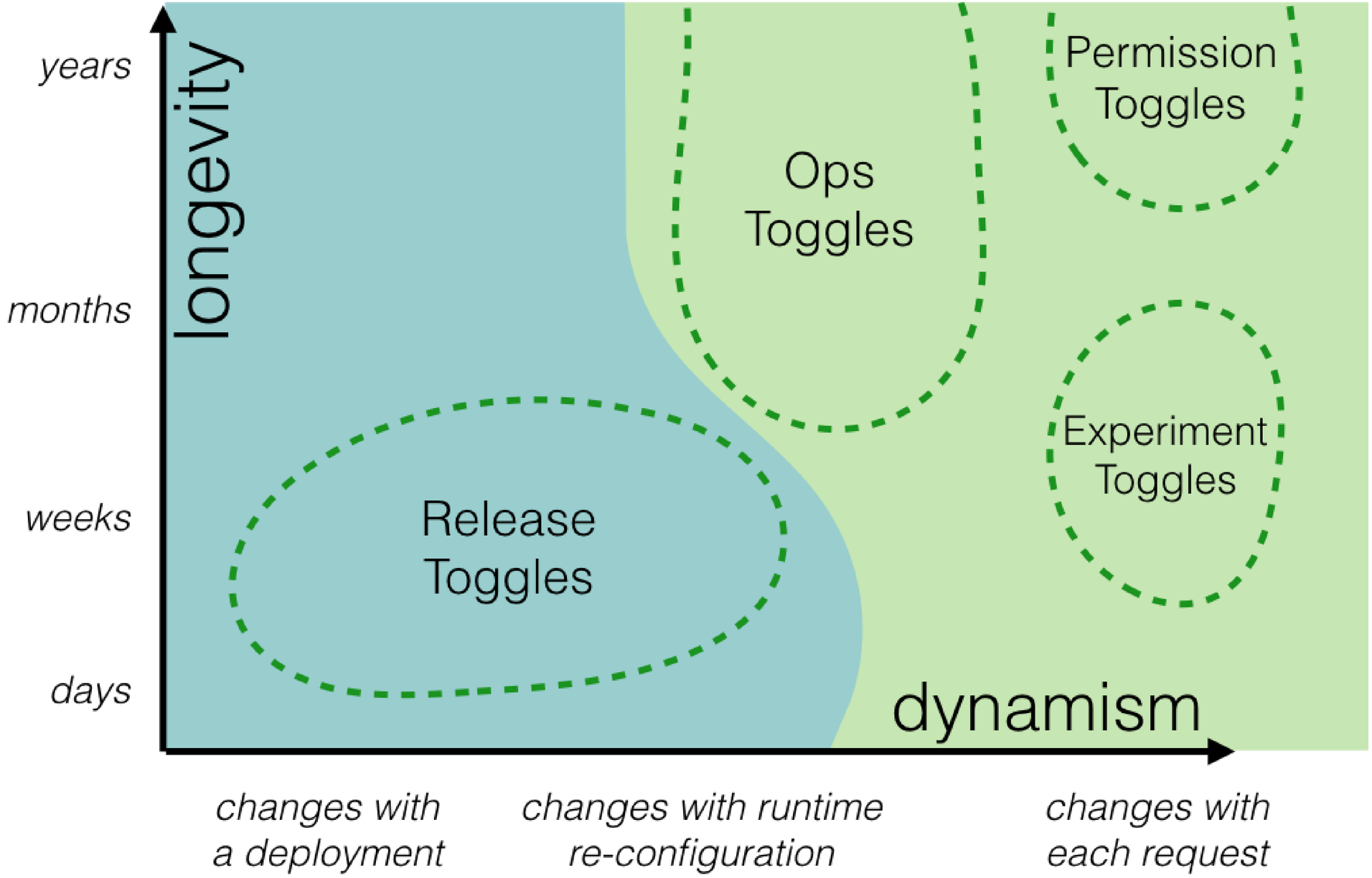

—enables trunk based development
—in-progress feature verification
—incomplete or untested code paths
—latent code in production
years
months
weeks
days
longevity
Release
Toggles
dynamism
changes with
a deployment
changes with runtime
re-configuration
changes with
each request


—perform multivariate or A/B testing
—utilizes a toggle router
—typical lifetime of hours or weeks
—highly dynamic
years
months
weeks
days
longevity
Release Toggles
Experiment Toggles
dynamism
changes with a deployment
changes with runtime re-configuration
changes with each request

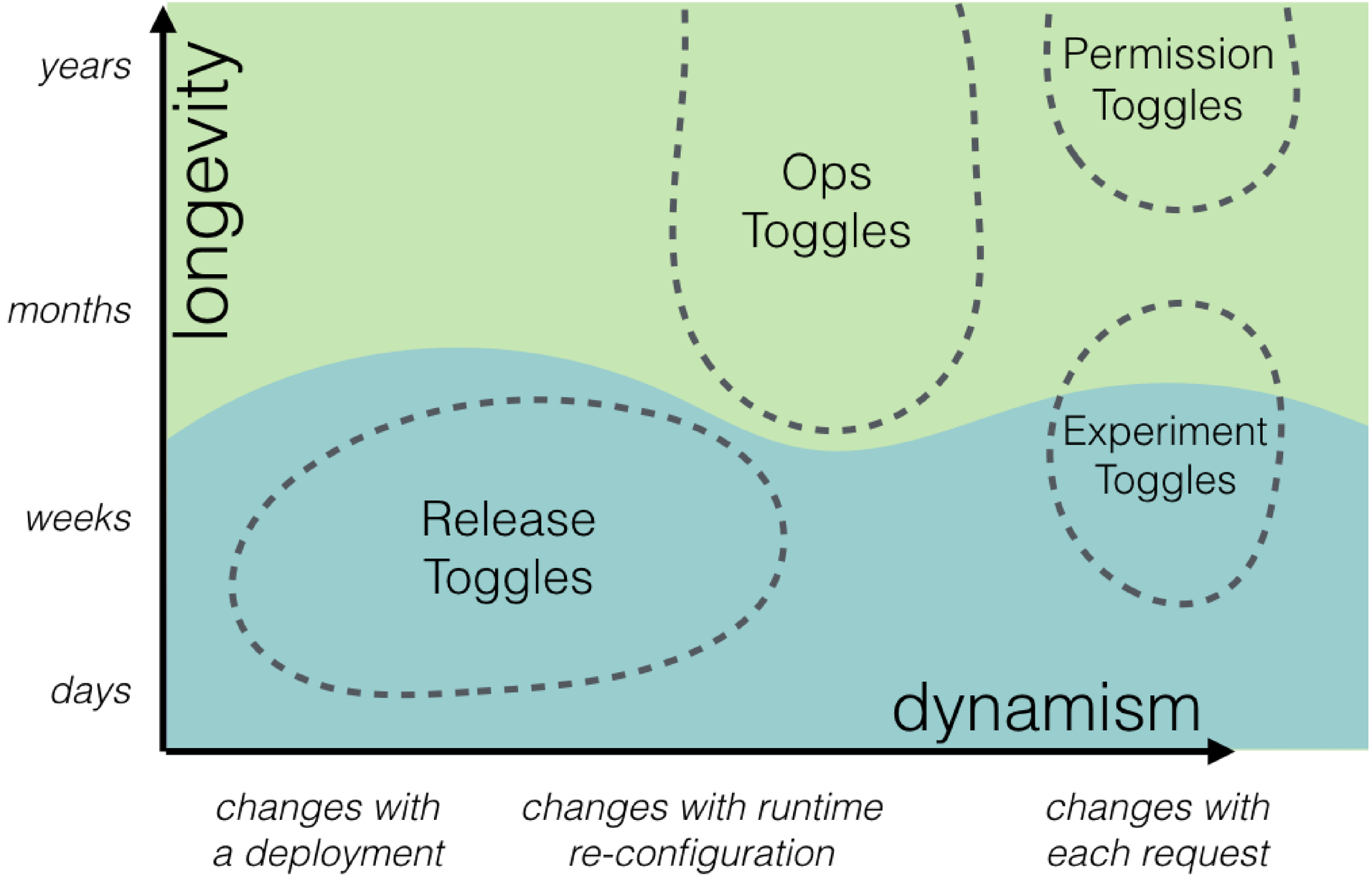

years
months
weeks
days
longevity
—operational
—short lived
—“kill switches”
Ops
Toggles
Release
Toggles
Experiment
Toggles
dynamism
changes with
a deployment
changes with runtime
re-configuration
changes with
each request


—product experience
—“premium” / alpha
—very long lived/permanent
years
months
weeks
days
longevity
Permission Toggles
Ops Toggles
Release Toggles
Experiment Toggles
dynamism
changes with a deployment
changes with runtime re-configuration
changes with each request

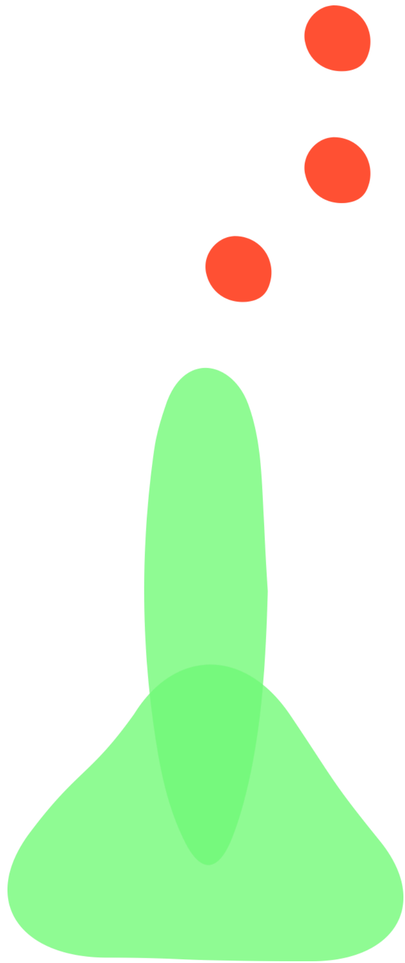
static versus dynamic toggles

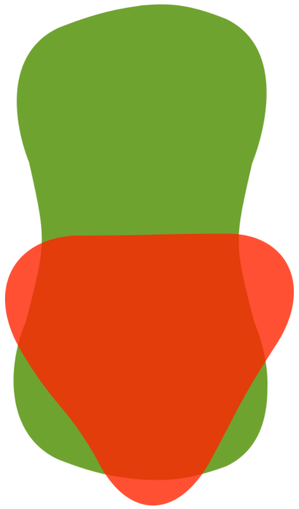


long-lived versus transient toggles



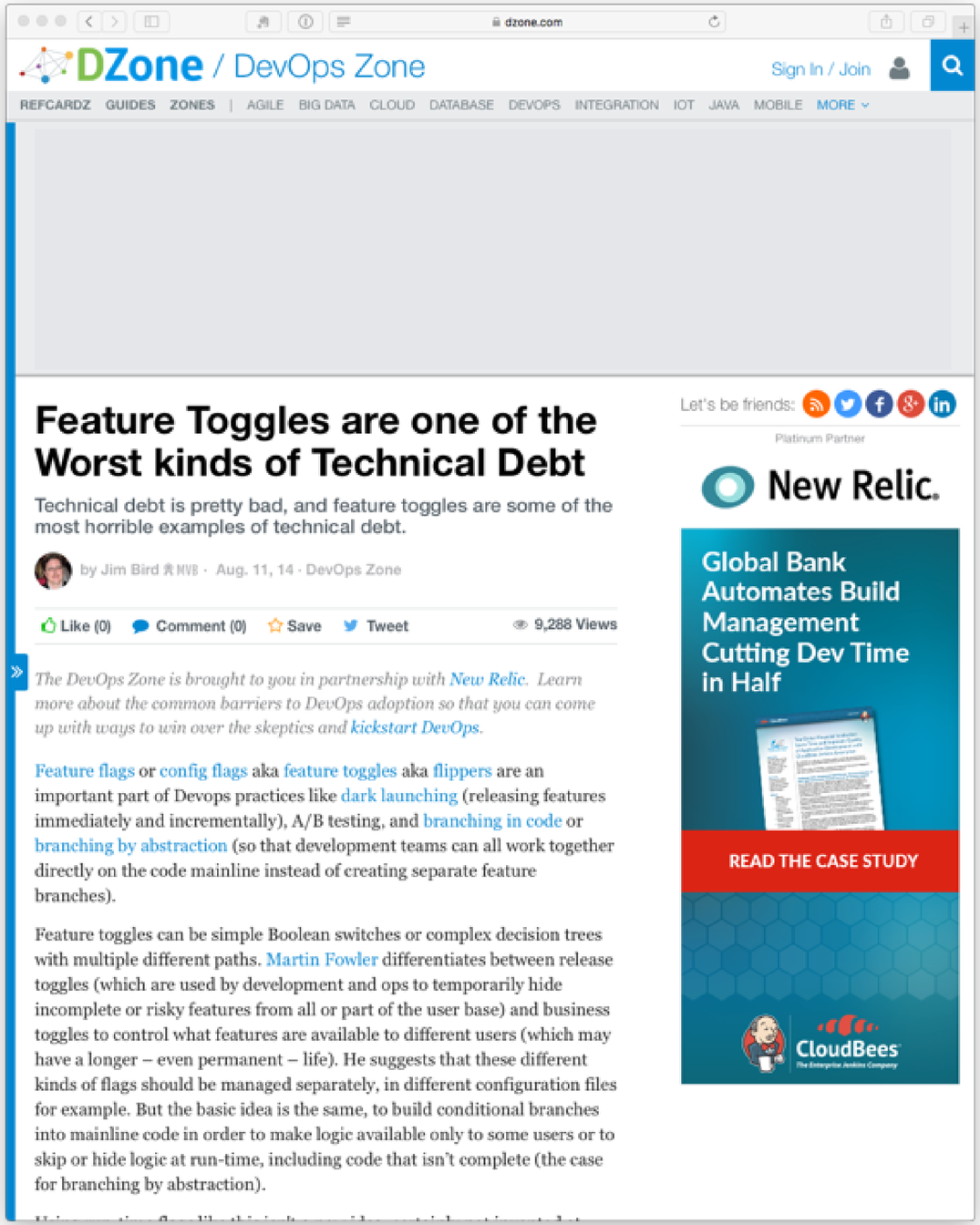
experimental toggles



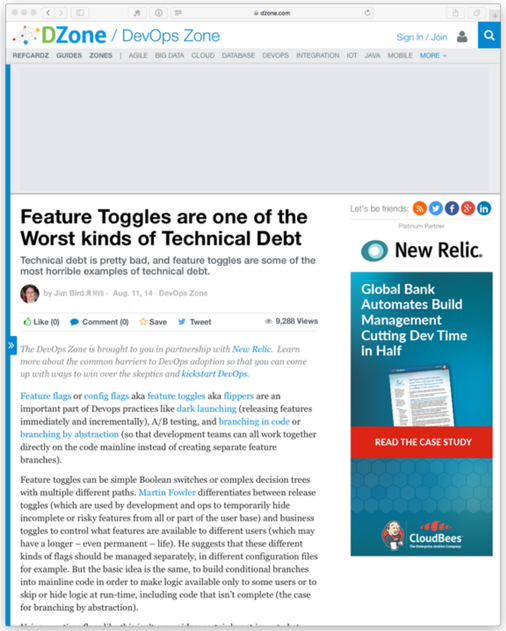


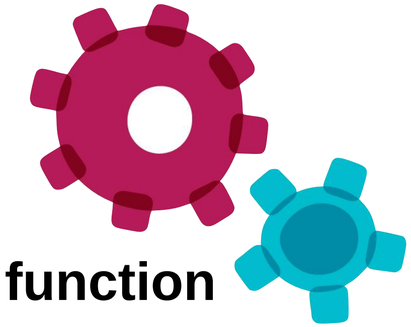
removed as soon as feature decision is resolved

Feature toggles are purposeful technical debt added to support engineering practices like Continuous Delivery.

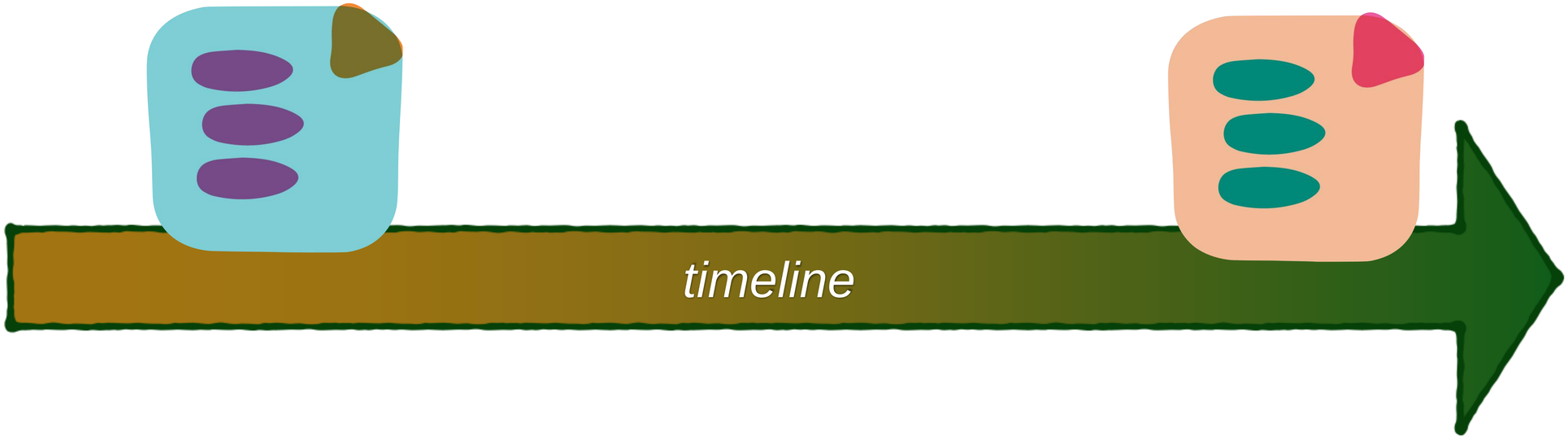


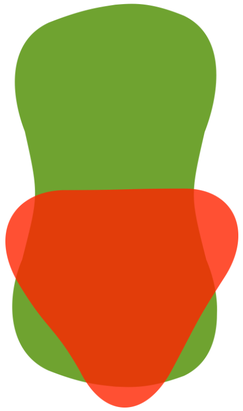
https://dzone.com/articles/feature-toggles-are-one-worst



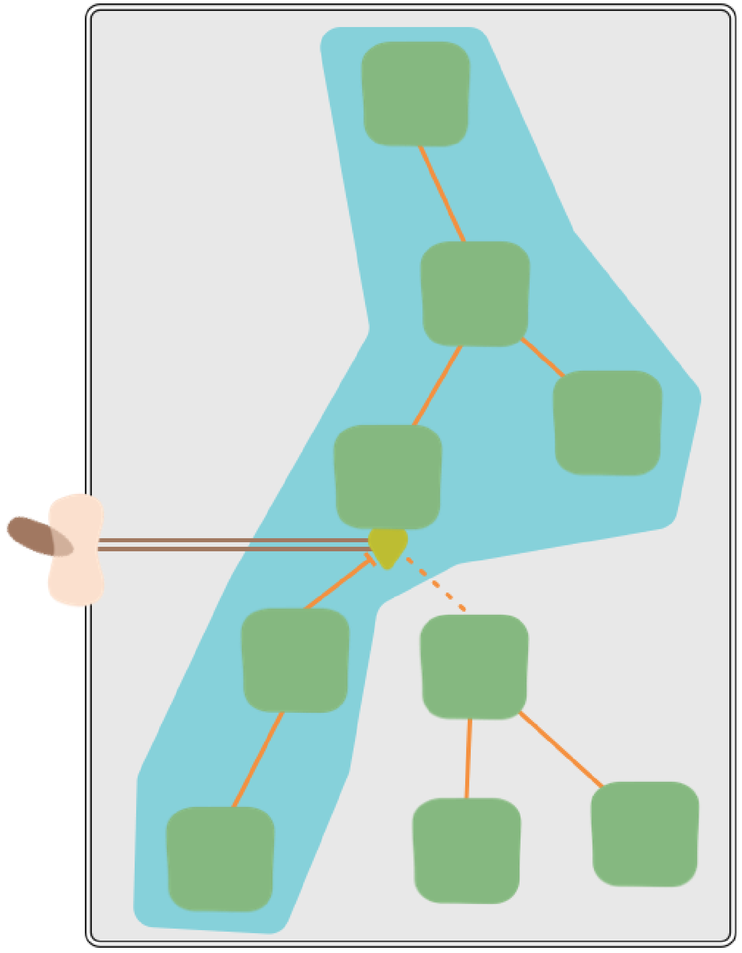


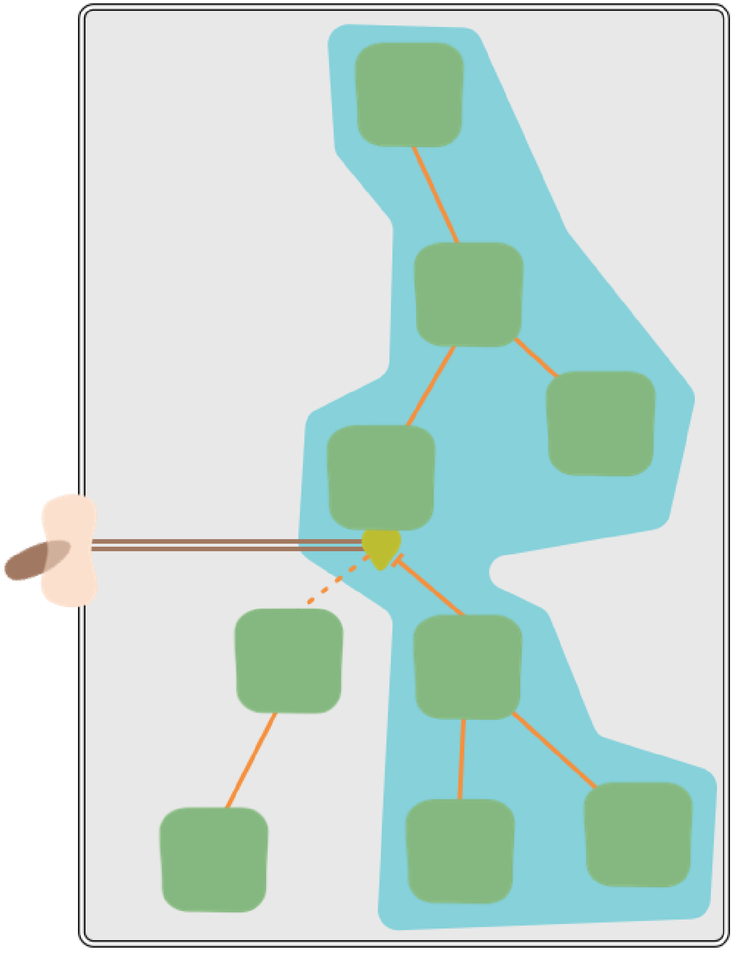
fitness function

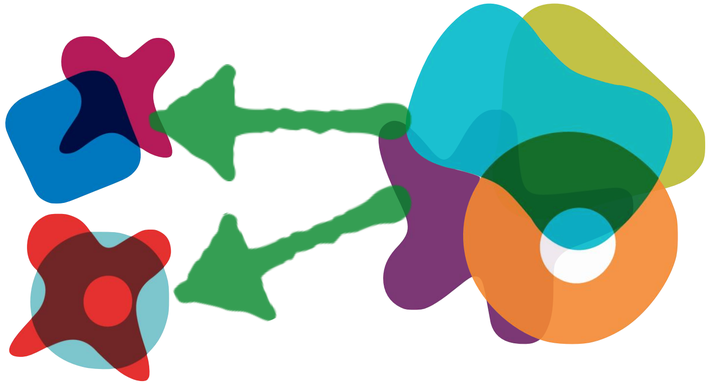




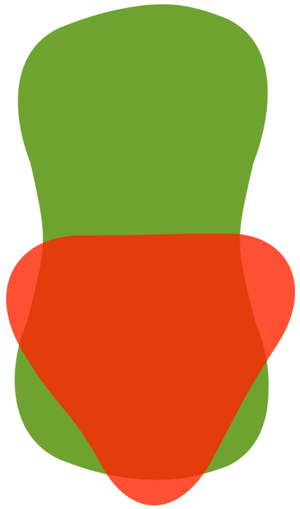

Tests

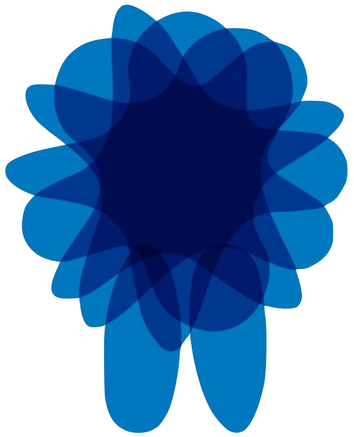


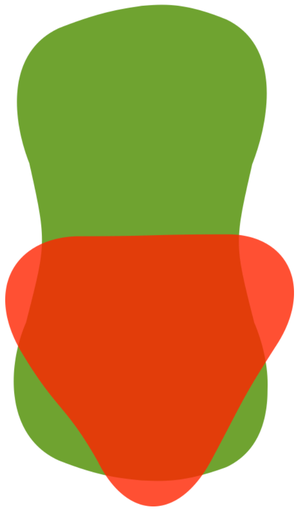




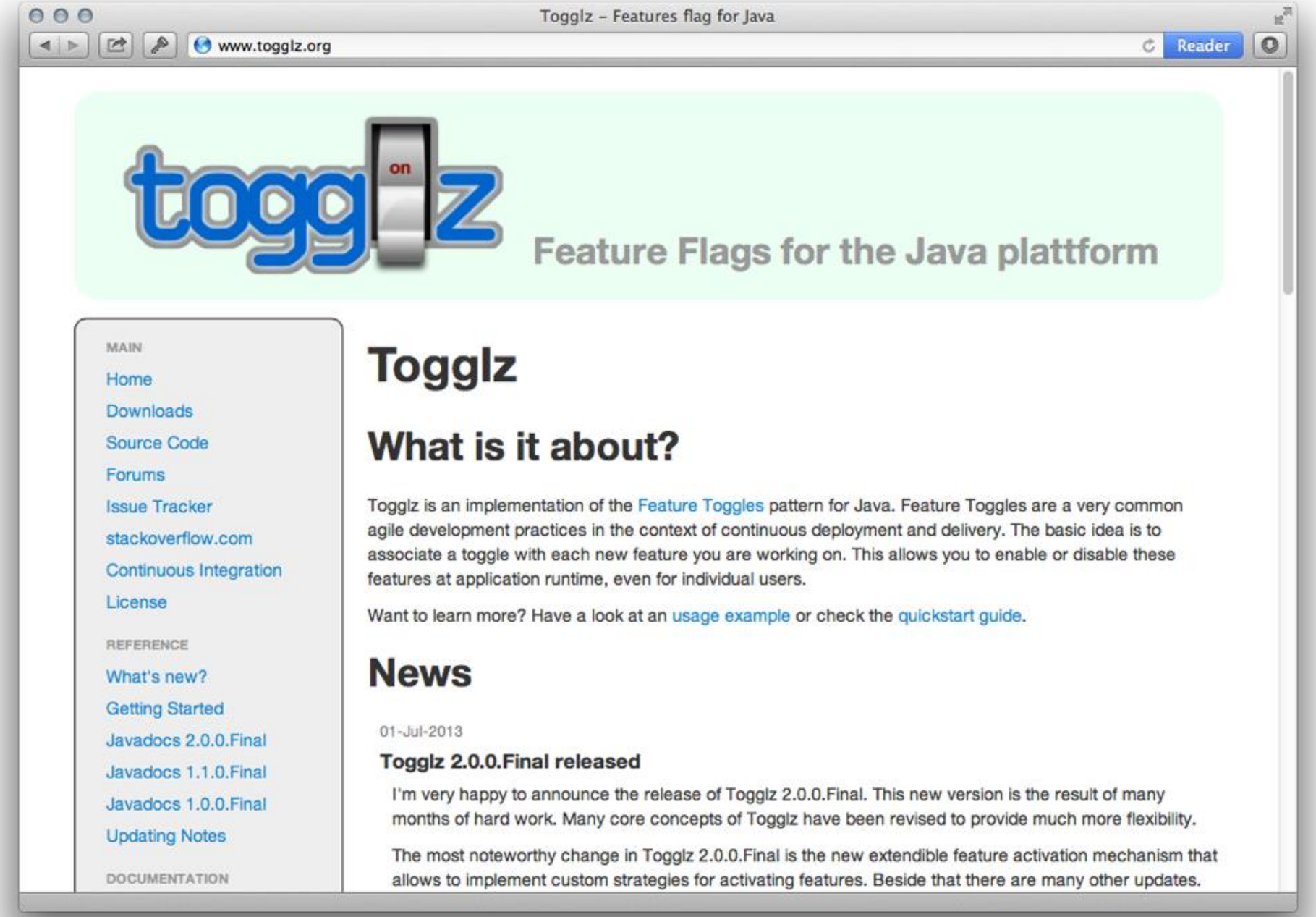
don’t create toggles that depend on other toggles.



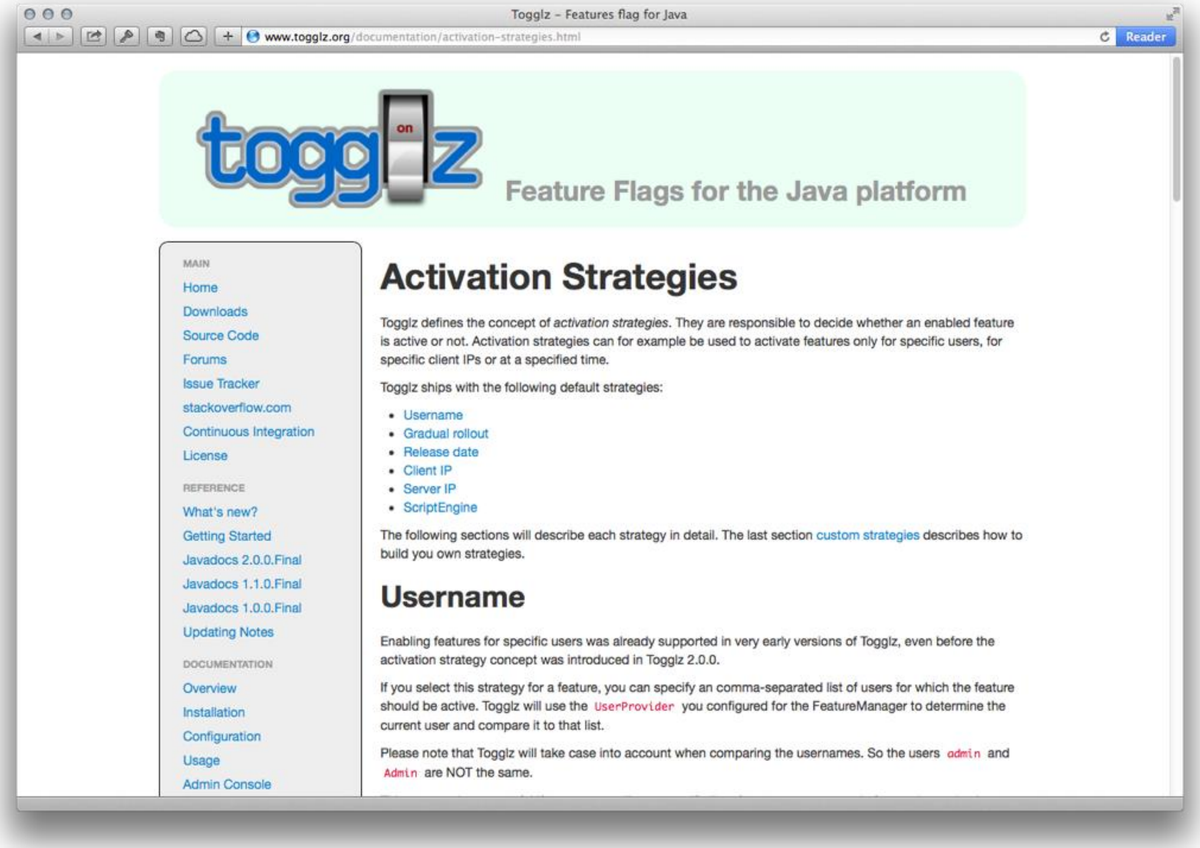




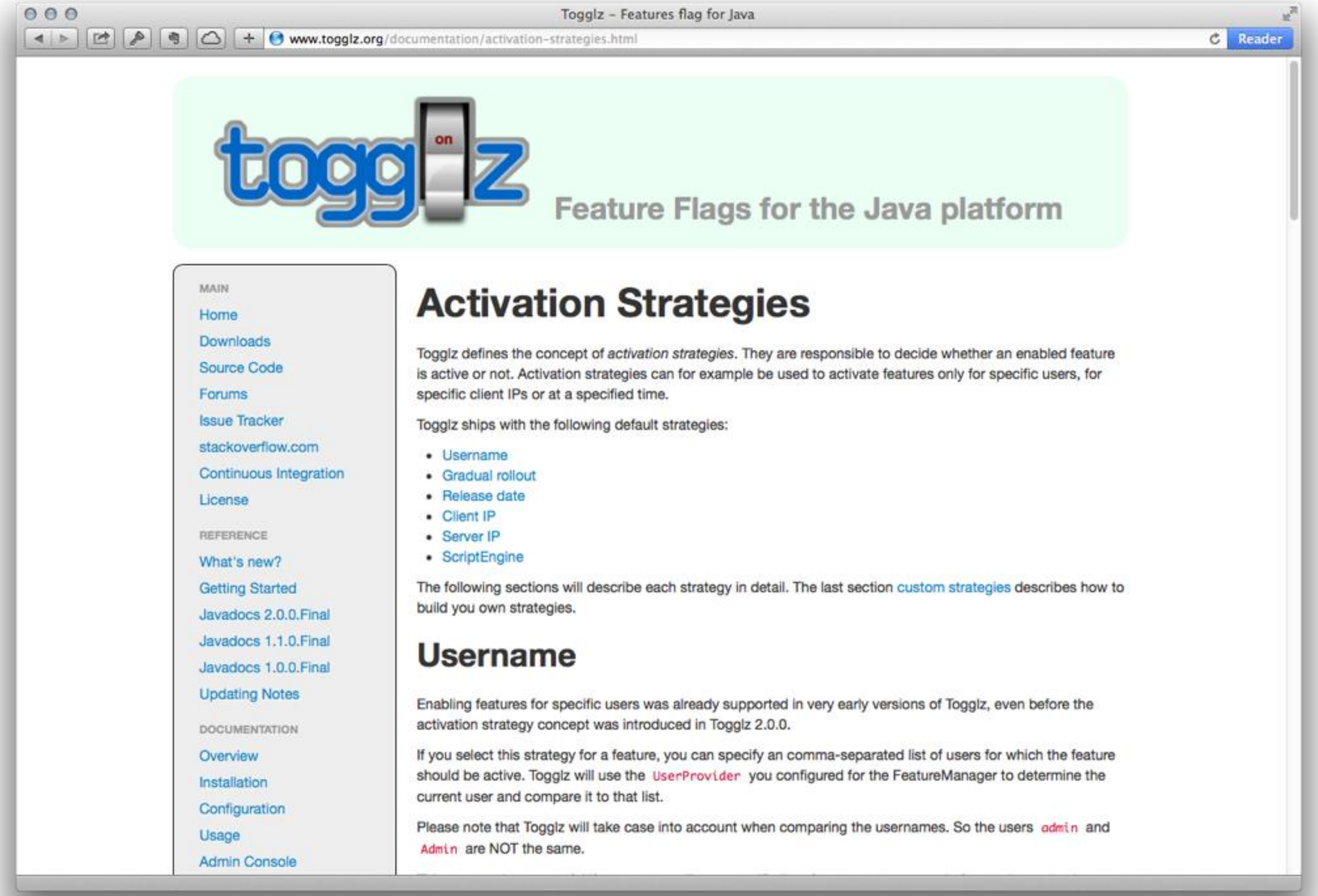
works on all platforms &
technology stacks



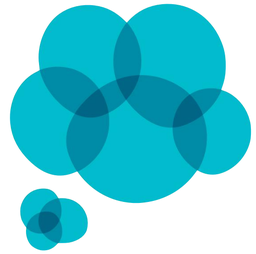
www.togglz.org



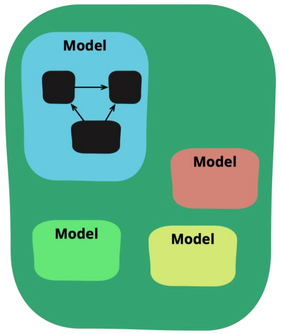

```java
@FeatureGroup
@Label("Performance Improvements")
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Performance {
  // no content
}
```

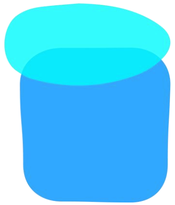


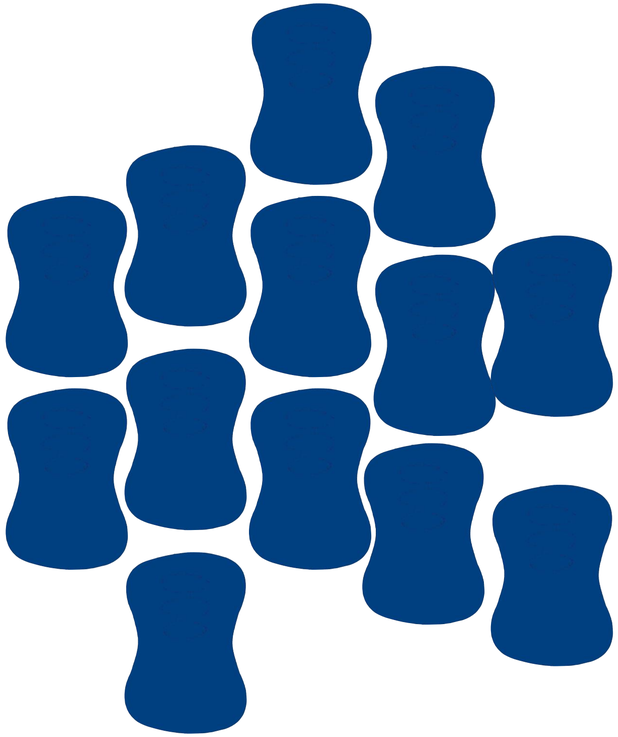
www.togglz.org

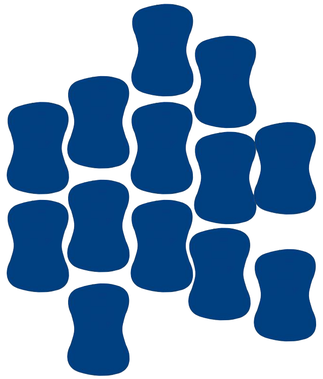


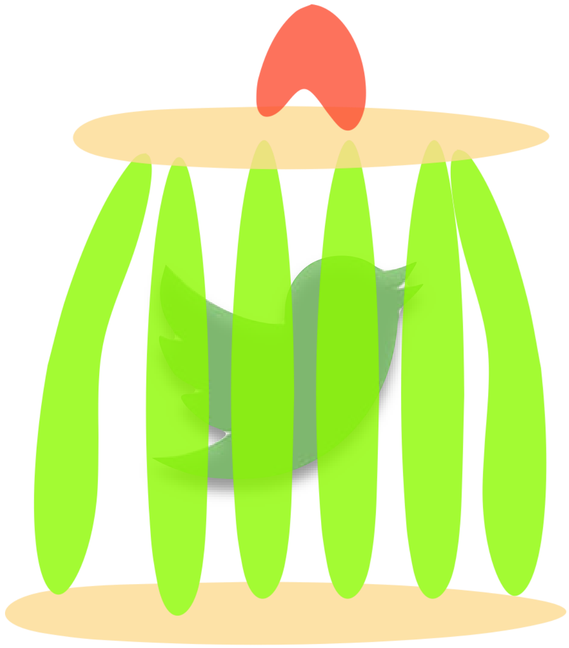

User interface


Model
Model
Model
Model



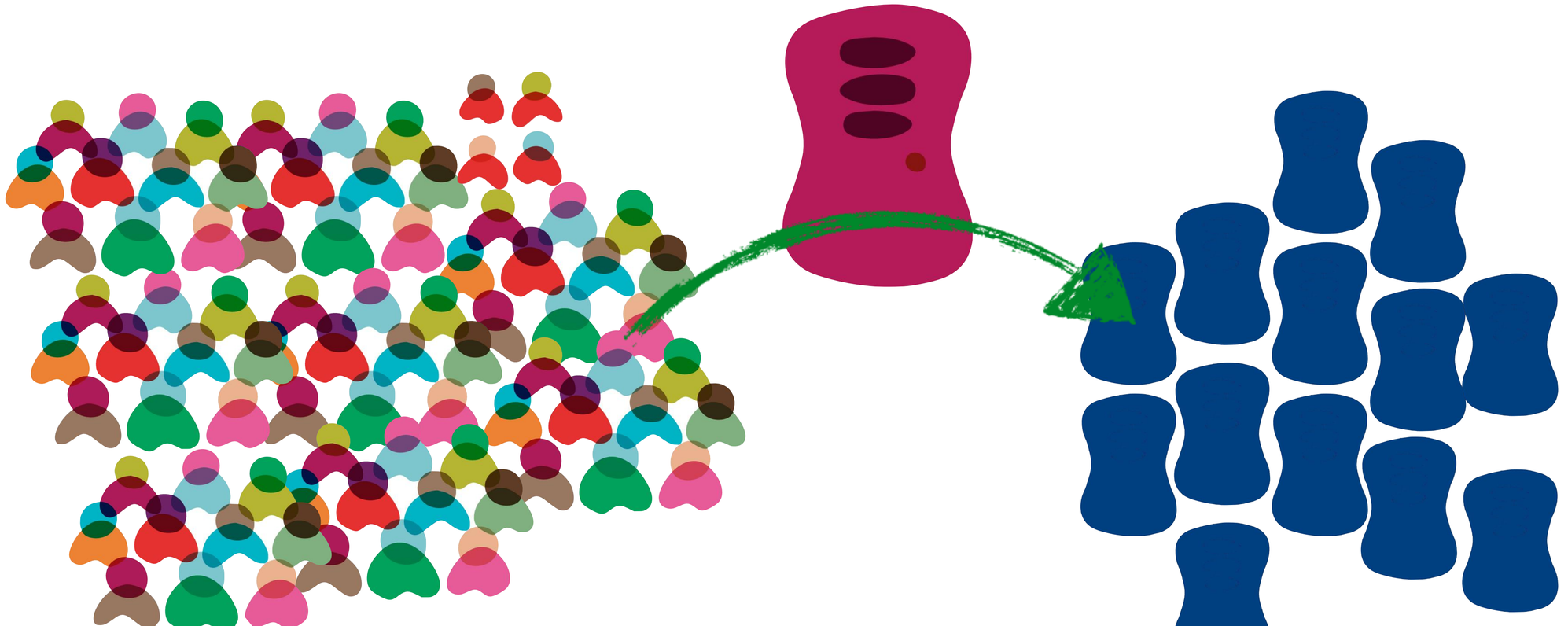




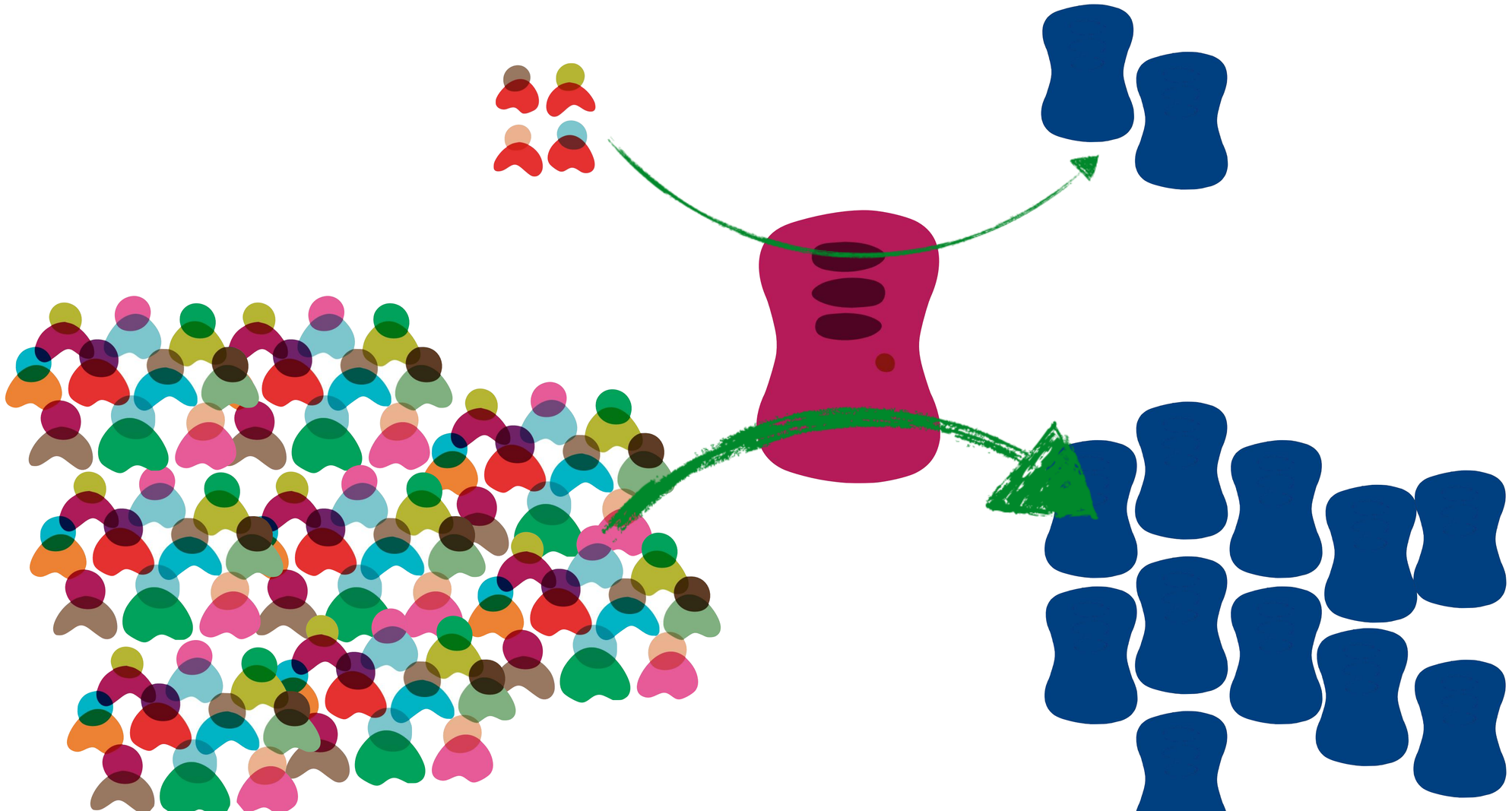


canary releasing

canary releasing



canary releasing

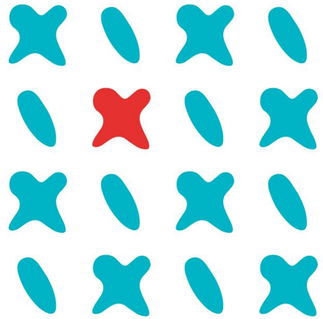


canary releasing

reduce risk of release

canary releasing

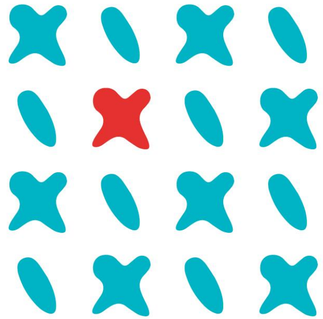
reduce risk of release



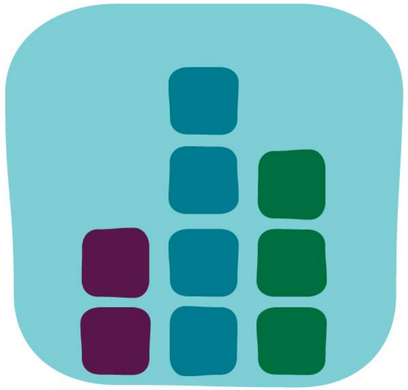
multi-variant testing

canary releasing

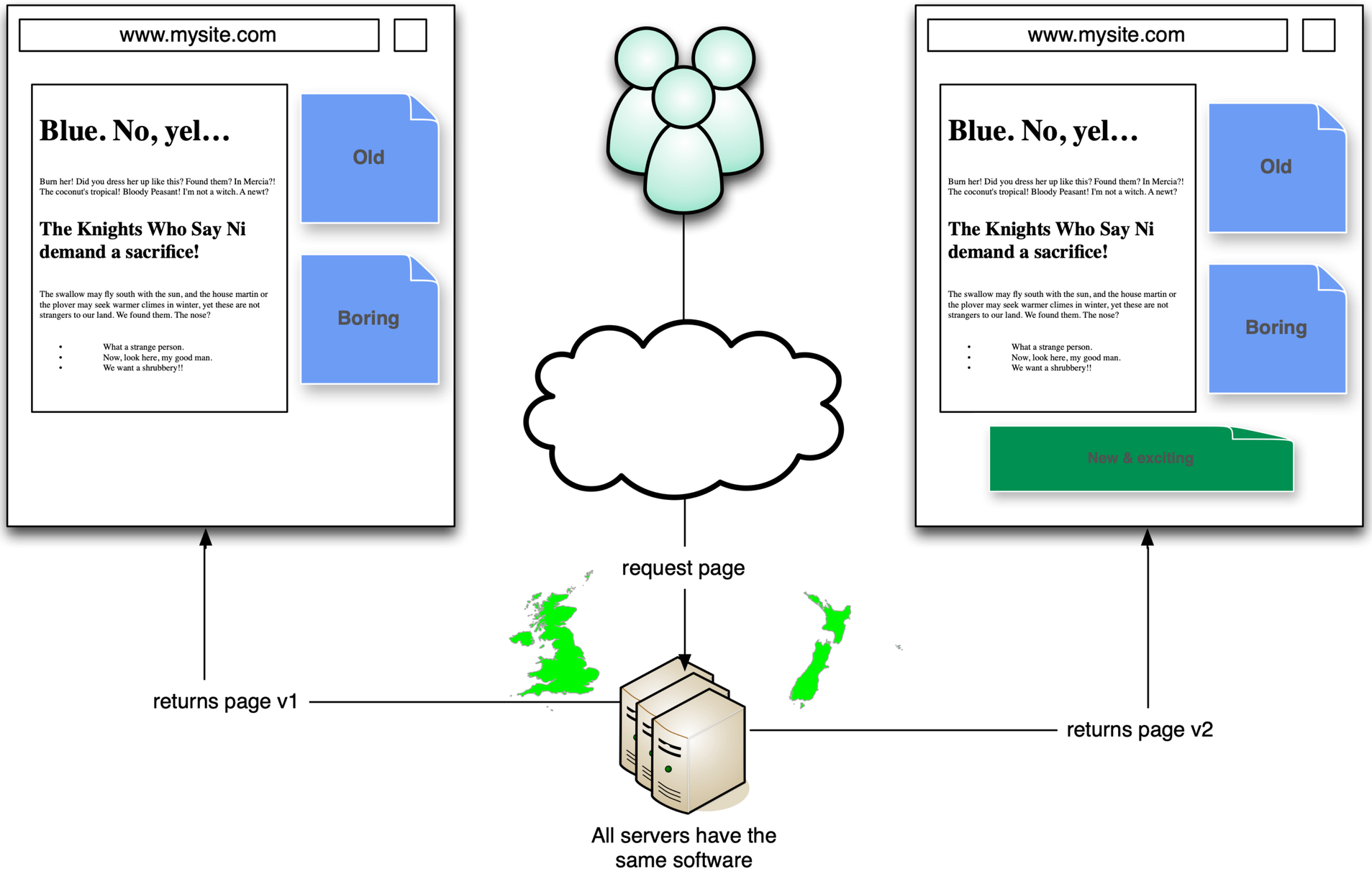
reduce risk of release



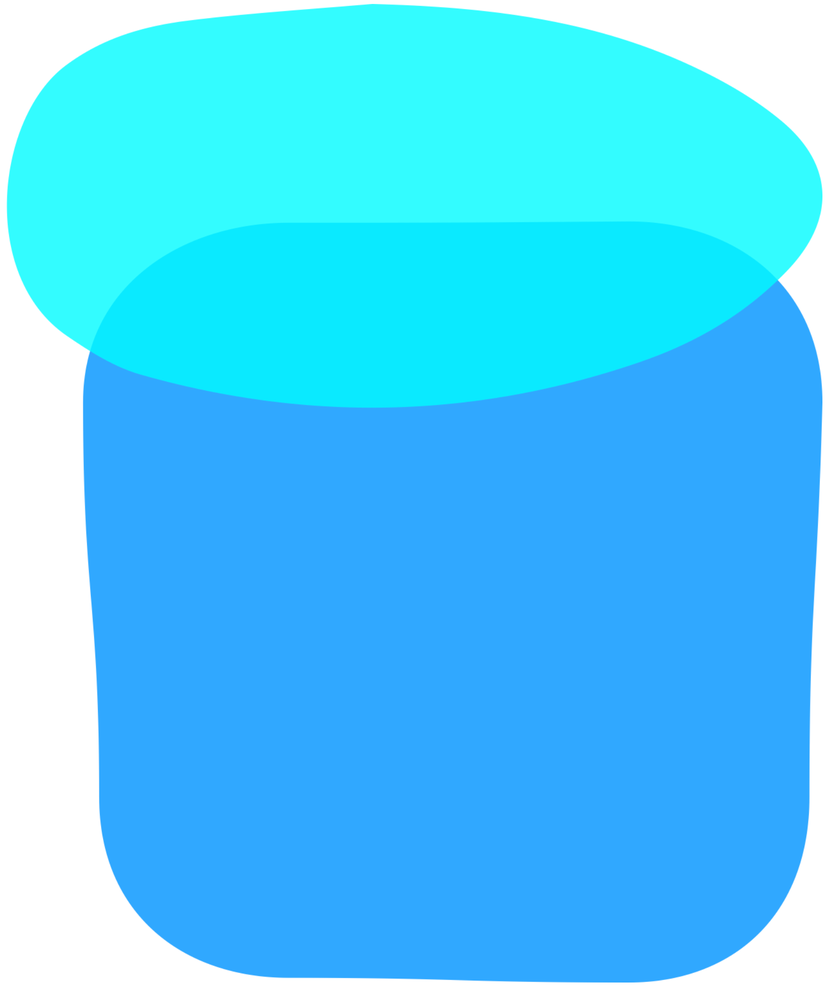
multi-variant testing



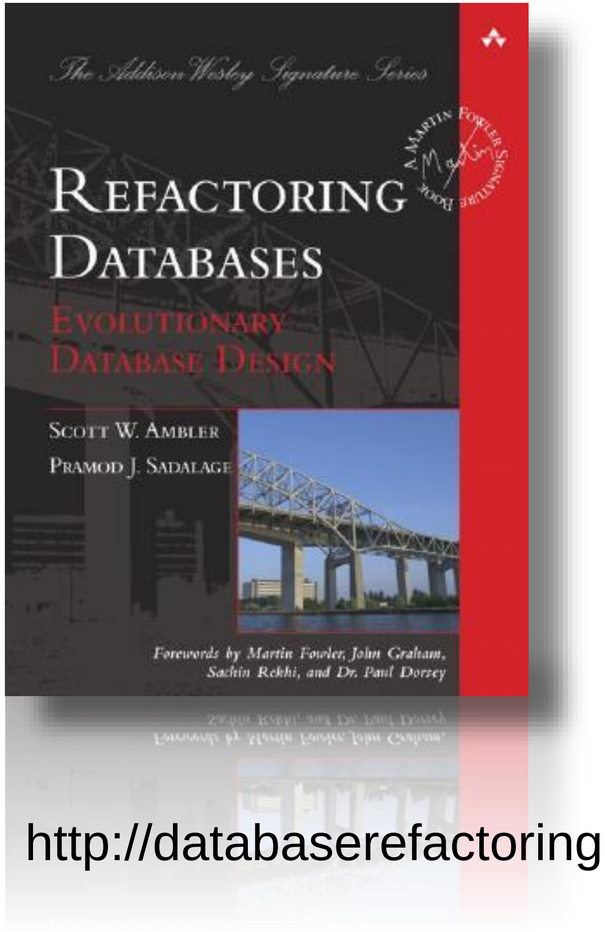
performance testing

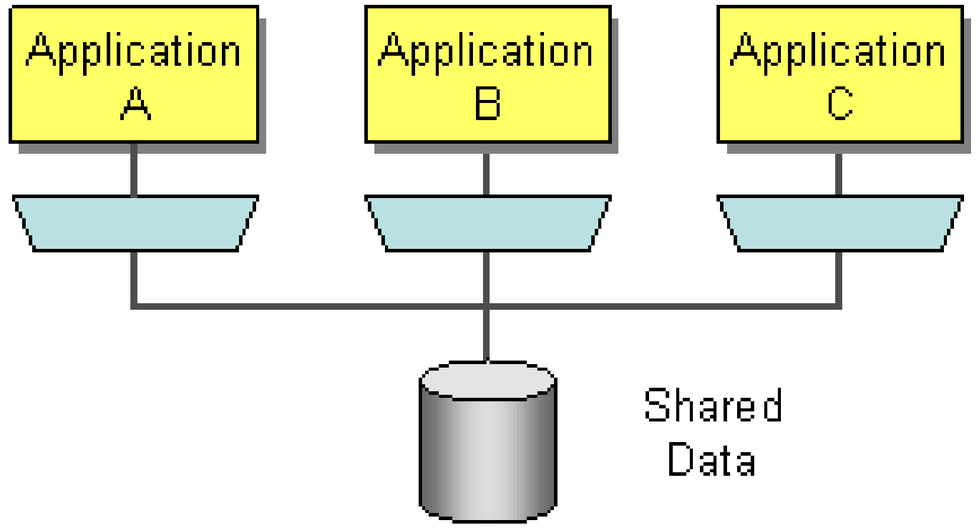

www.mysite.com
Blue. No, yel…
Burn her! Did you dress her up like this? Found them? In Mercia?! The coconut's tropical! Bloody Peasant! I'm not a witch. A newt?
The Knights Who Say Ni demand a sacrifice!
The swallow may fly south with the sun, and the house martin or the plover may seek warmer climes in winter, yet these are not strangers to our land. We found them. The nose?
What a strange person.
Now, look here, my good man.
We want a shrubbery!!
Old
Boring
www.mysite.com
Blue. No, yel…
Burn her! Did you dress her up like this? Found them? In Mercia?! The coconut's tropical! Bloody Peasant! I'm not a witch. A newt?
The Knights Who Say Ni demand a sacrifice!
The swallow may fly south with the sun, and the house martin or the plover may seek warmer climes in winter, yet these are not strangers to our land. We found them. The nose?
What a strange person.
Now, look here, my good man.
We want a shrubbery!!
Old
Boring
New & exciting
request page
returns page v1
returns page v2
All servers have the same software

Toggling Databases



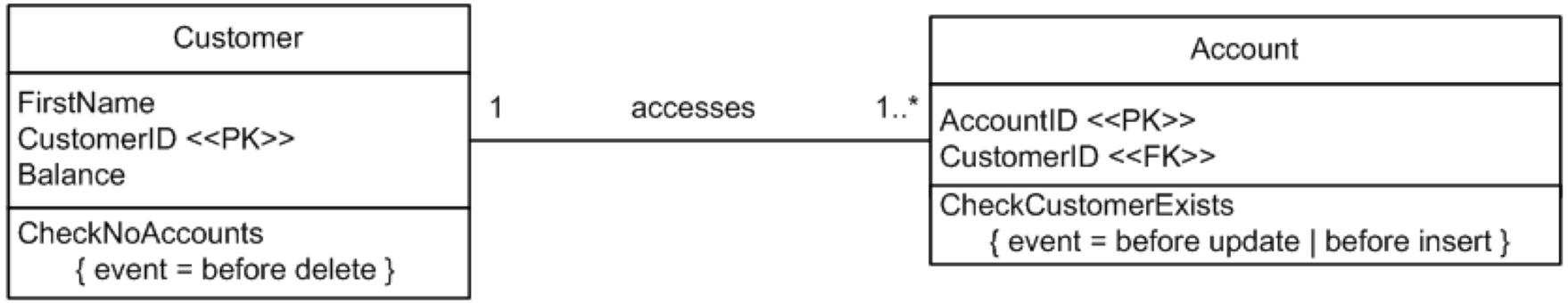
Evolutionary Database Design



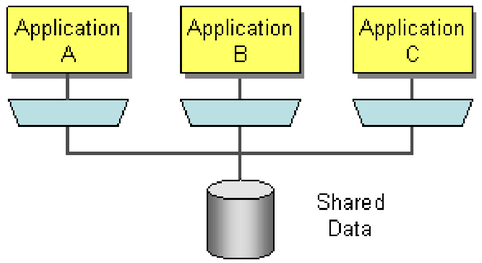


http://databaserefactoring.com/

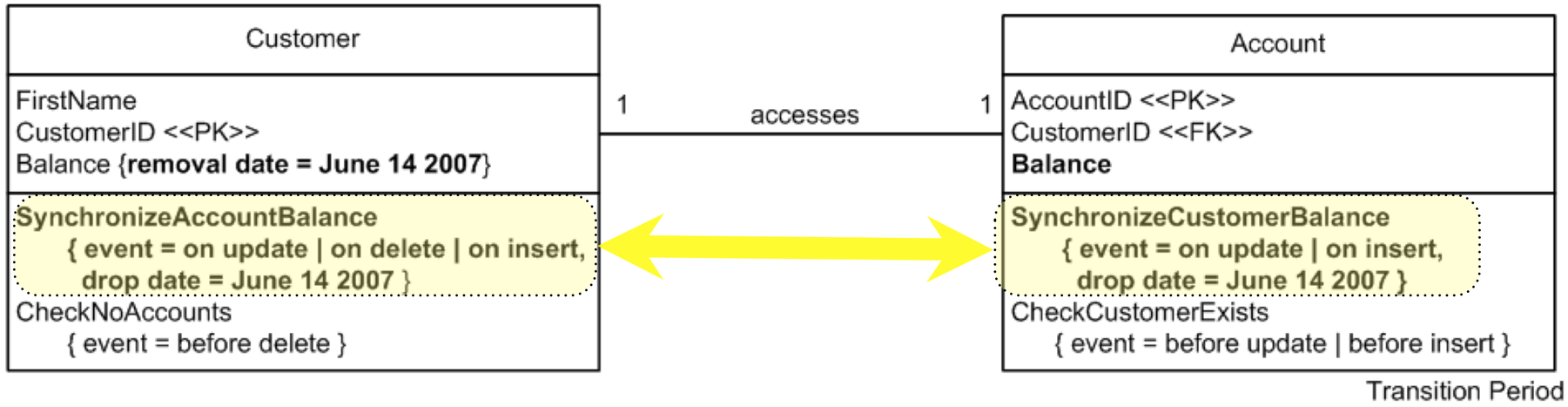
Initial Phase

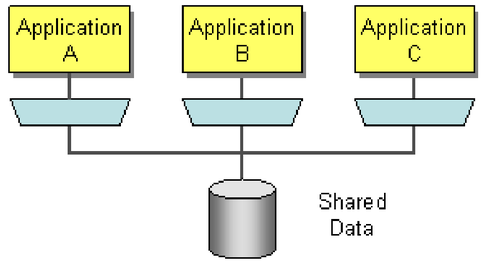


Original Schema

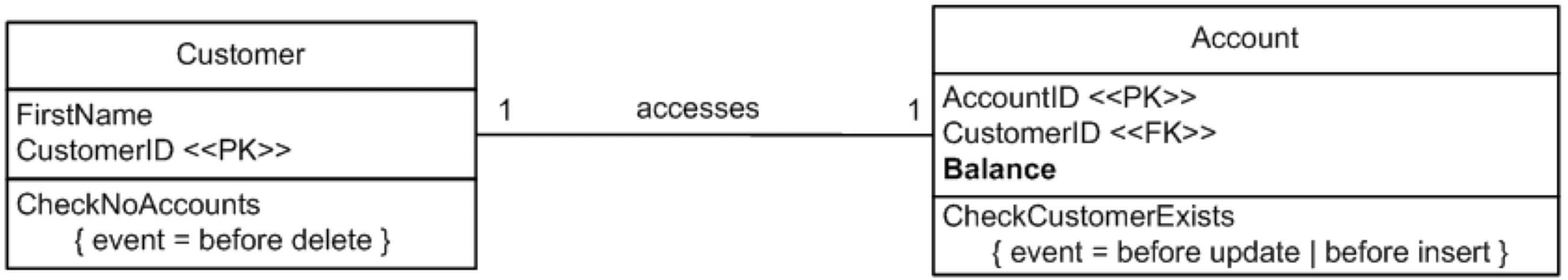


Transition Phase

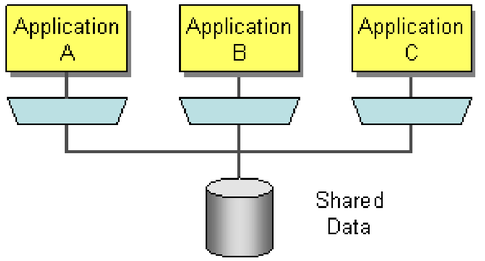




Ending State



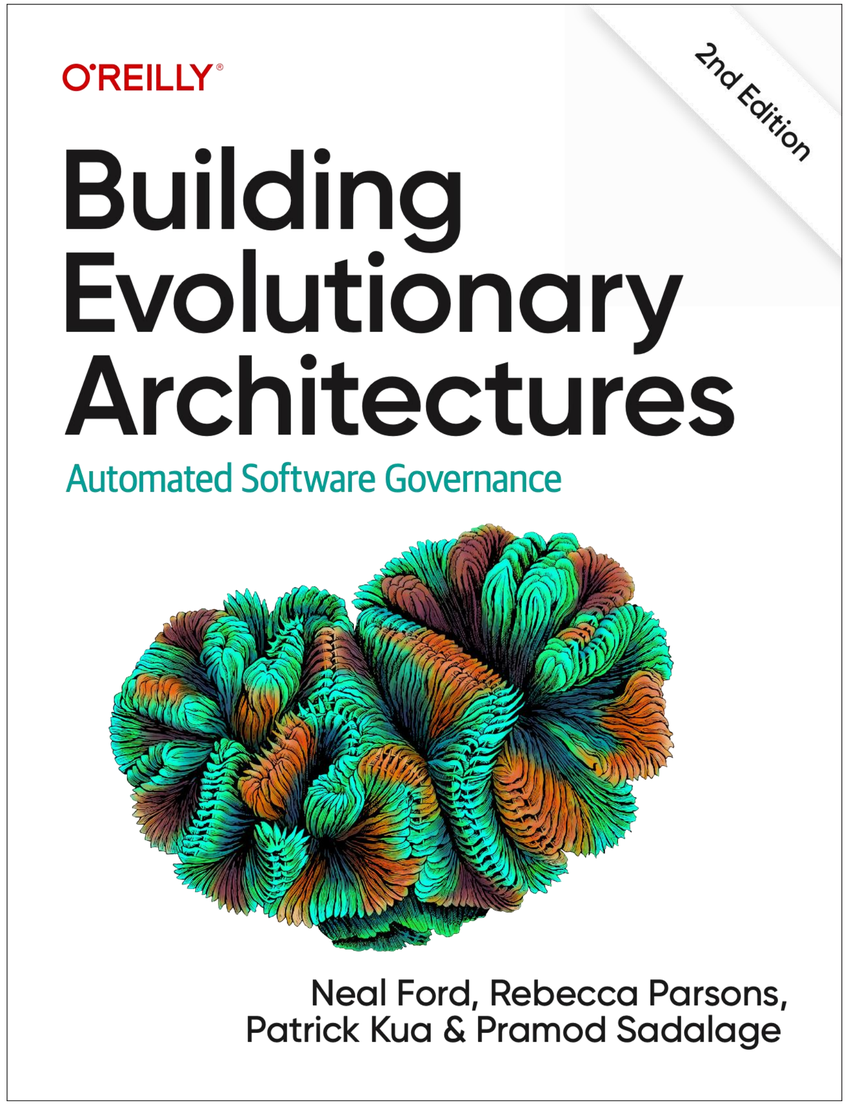
Resulting Schema



Question:

How can architects build decoupled systems that allow continuous delivery/deployment and evolution?

aRChiTeCtuRe
fOr
cONtiNuoUs DeLiVeRy

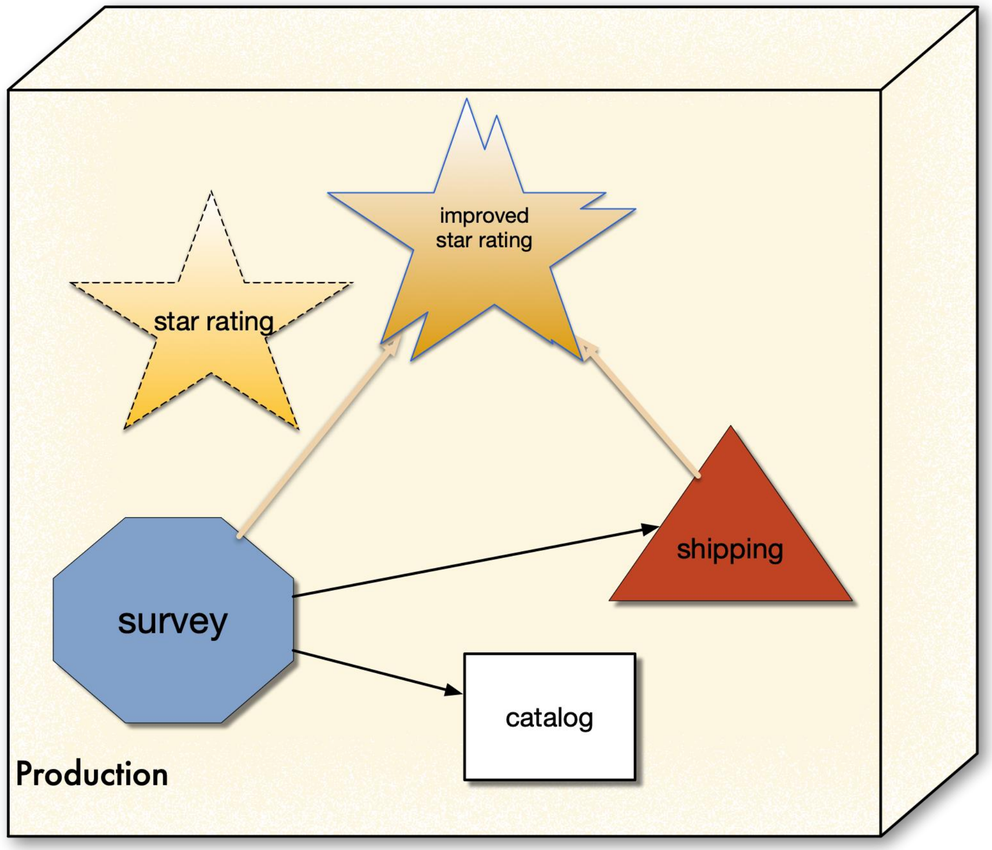


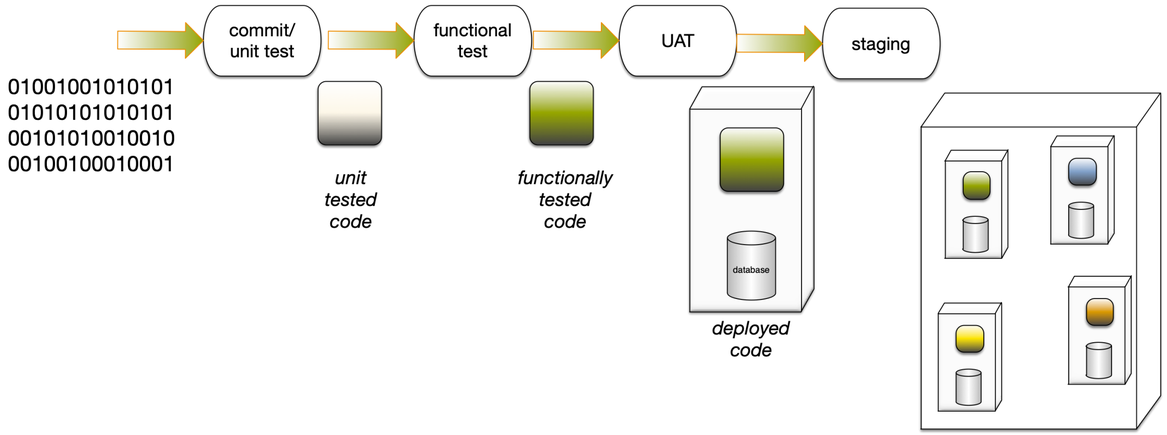
@neal4d
nealford.com

@rebeccaparsons

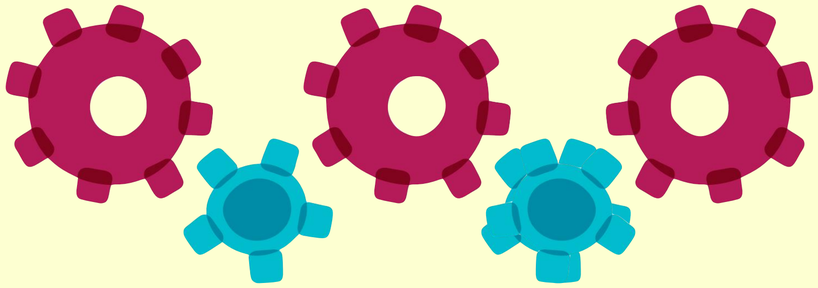
@patkua

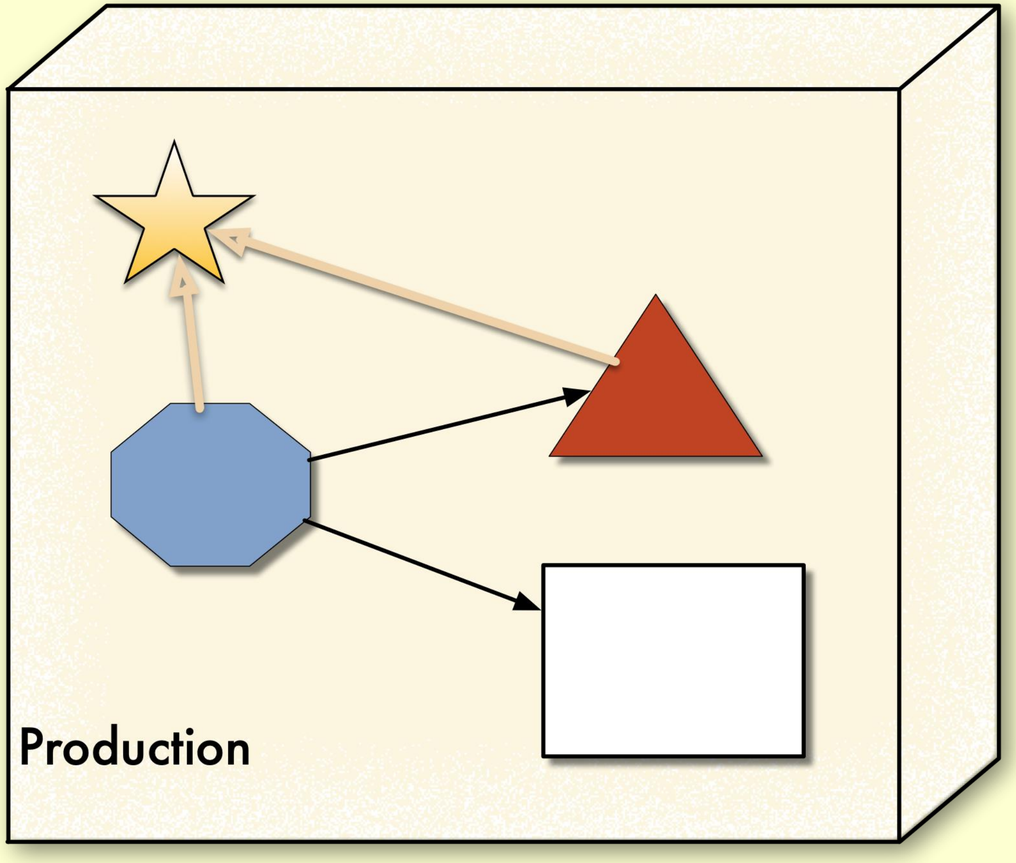
incremental



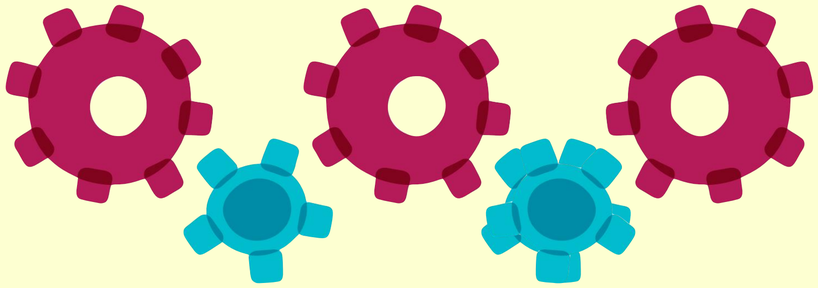


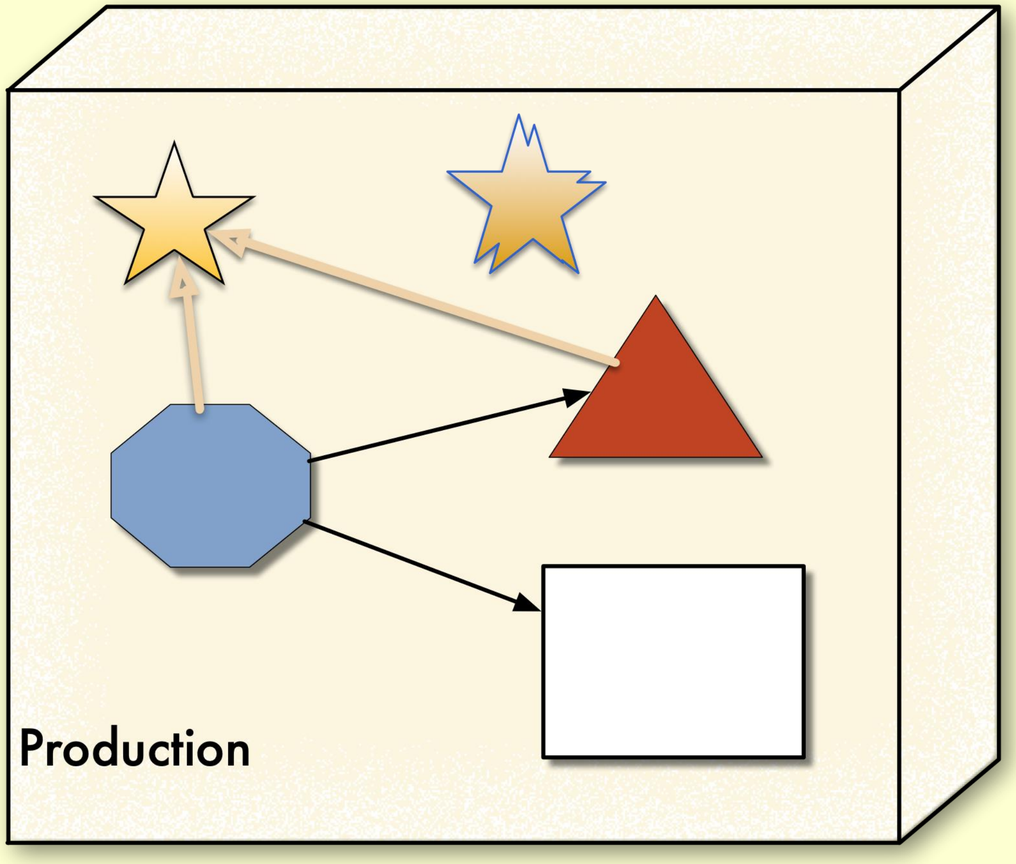
Penultimate



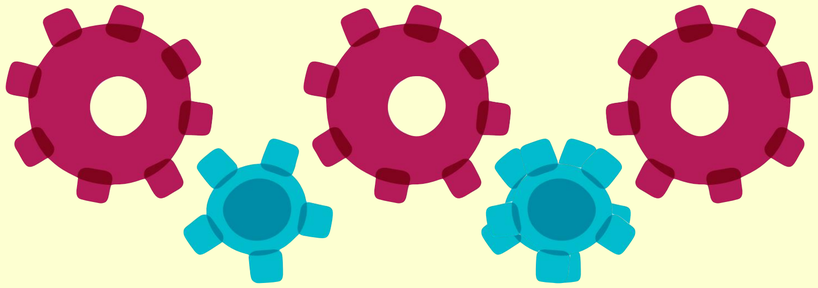


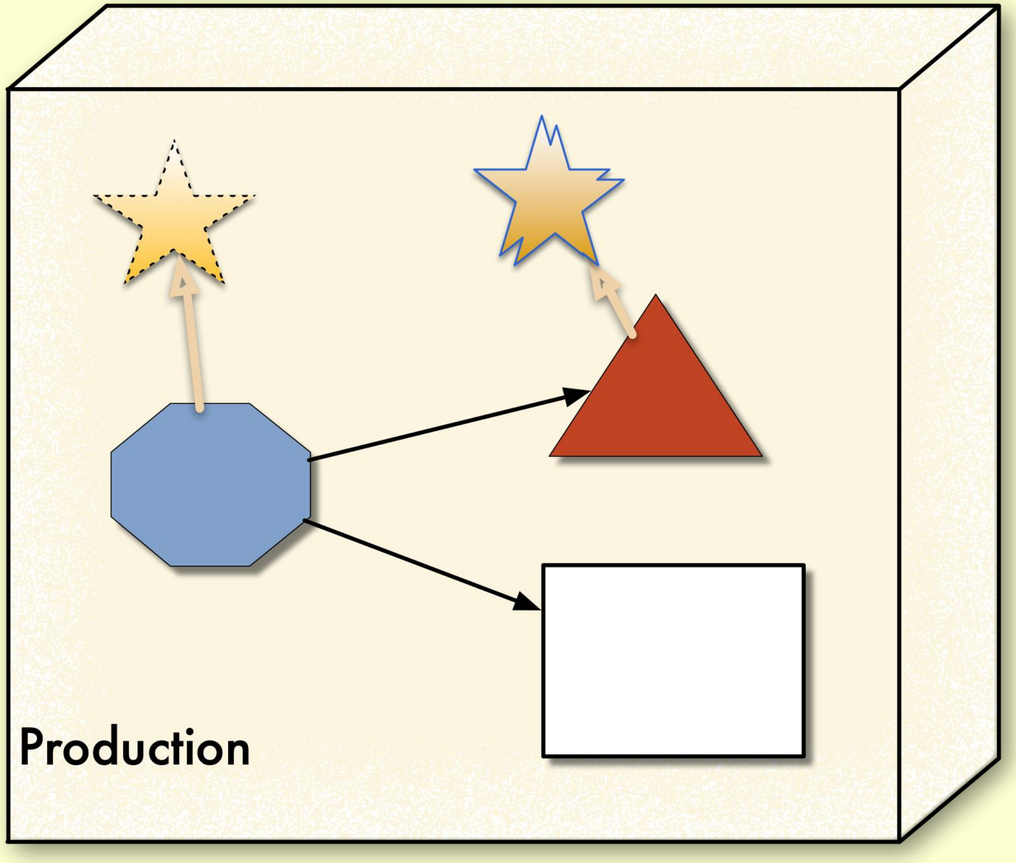
Penultimate



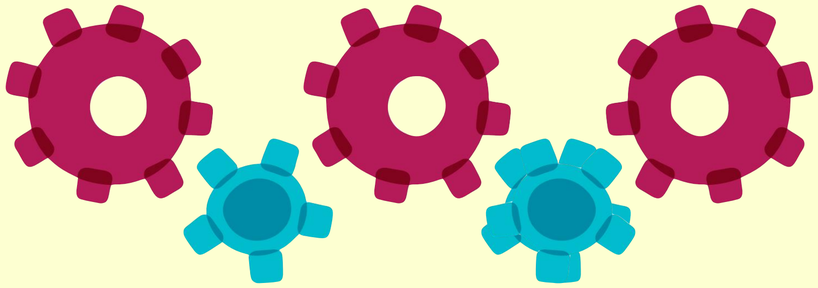


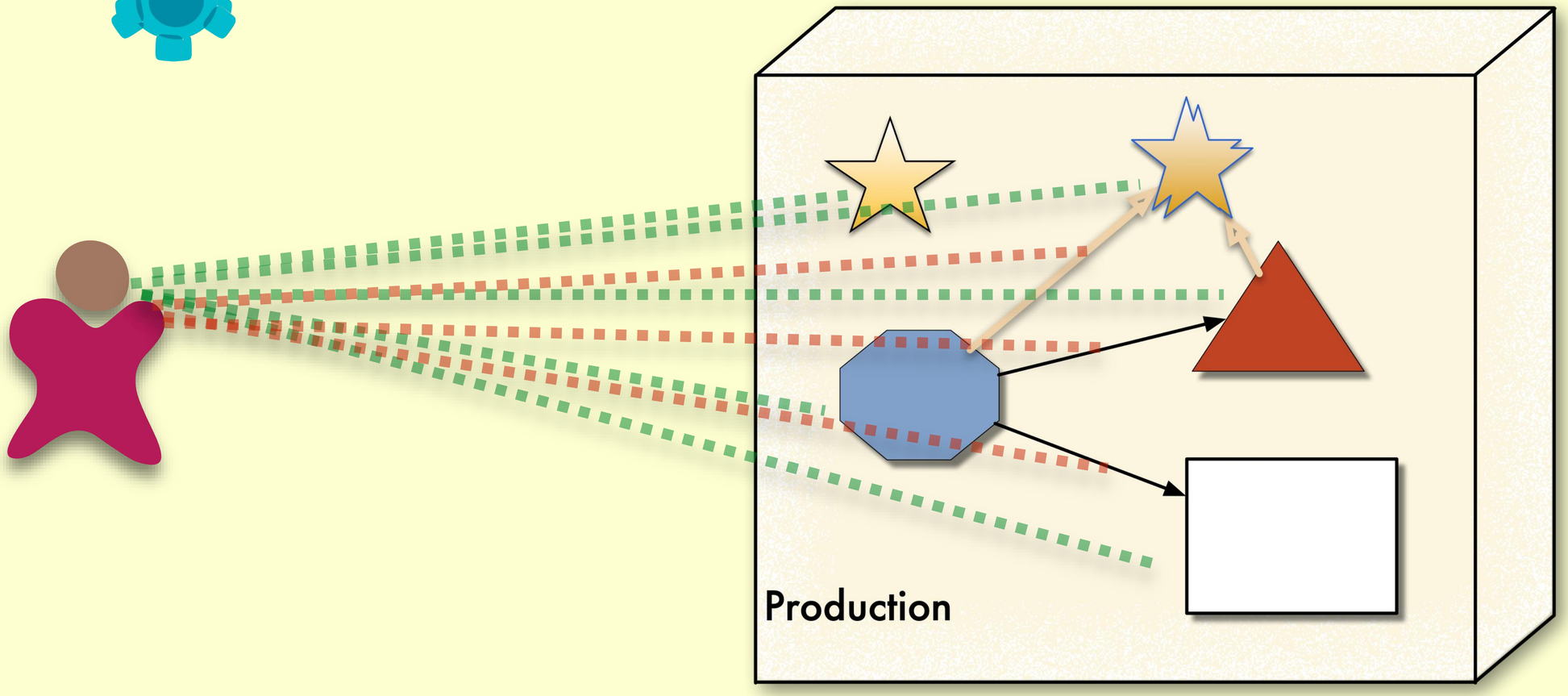
Penultimate



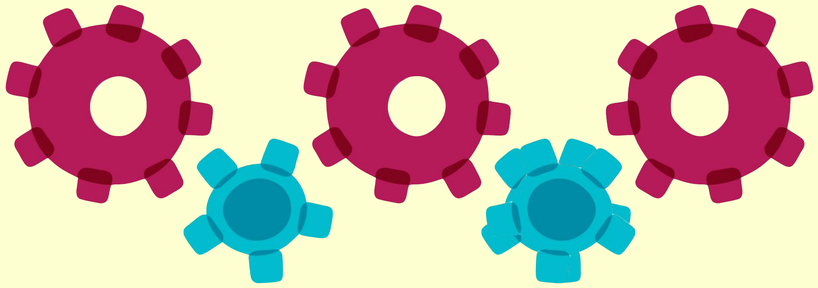


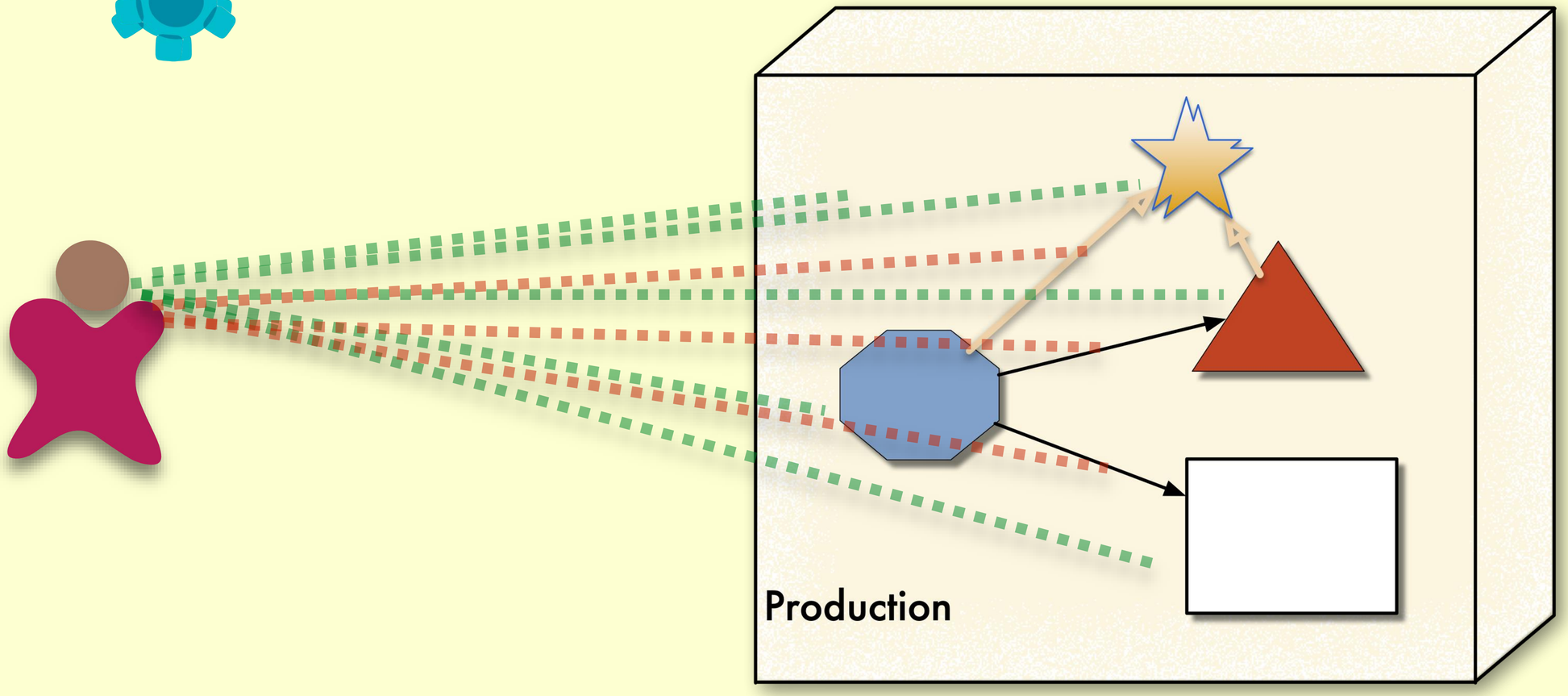
Penultimate



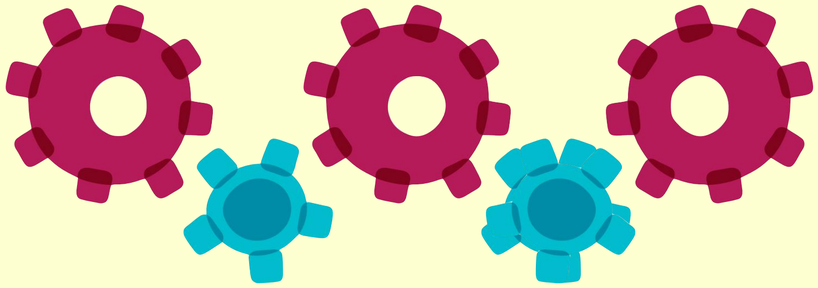


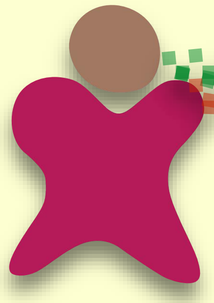
Penultimate

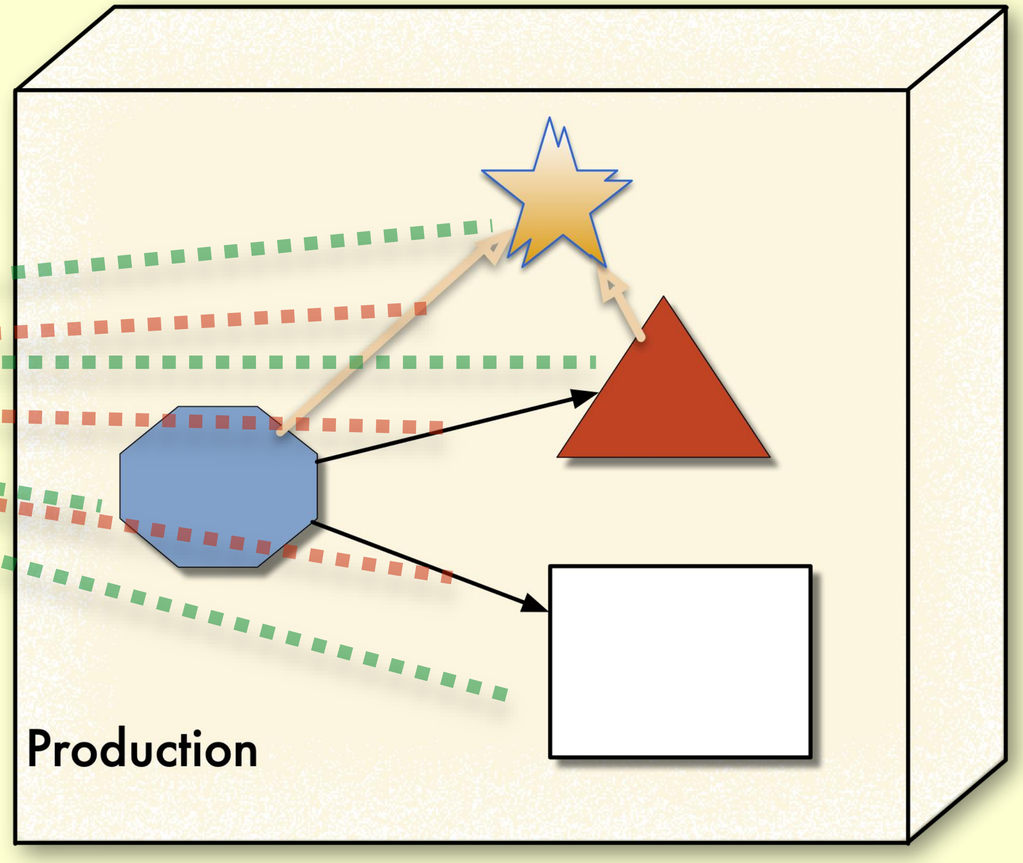


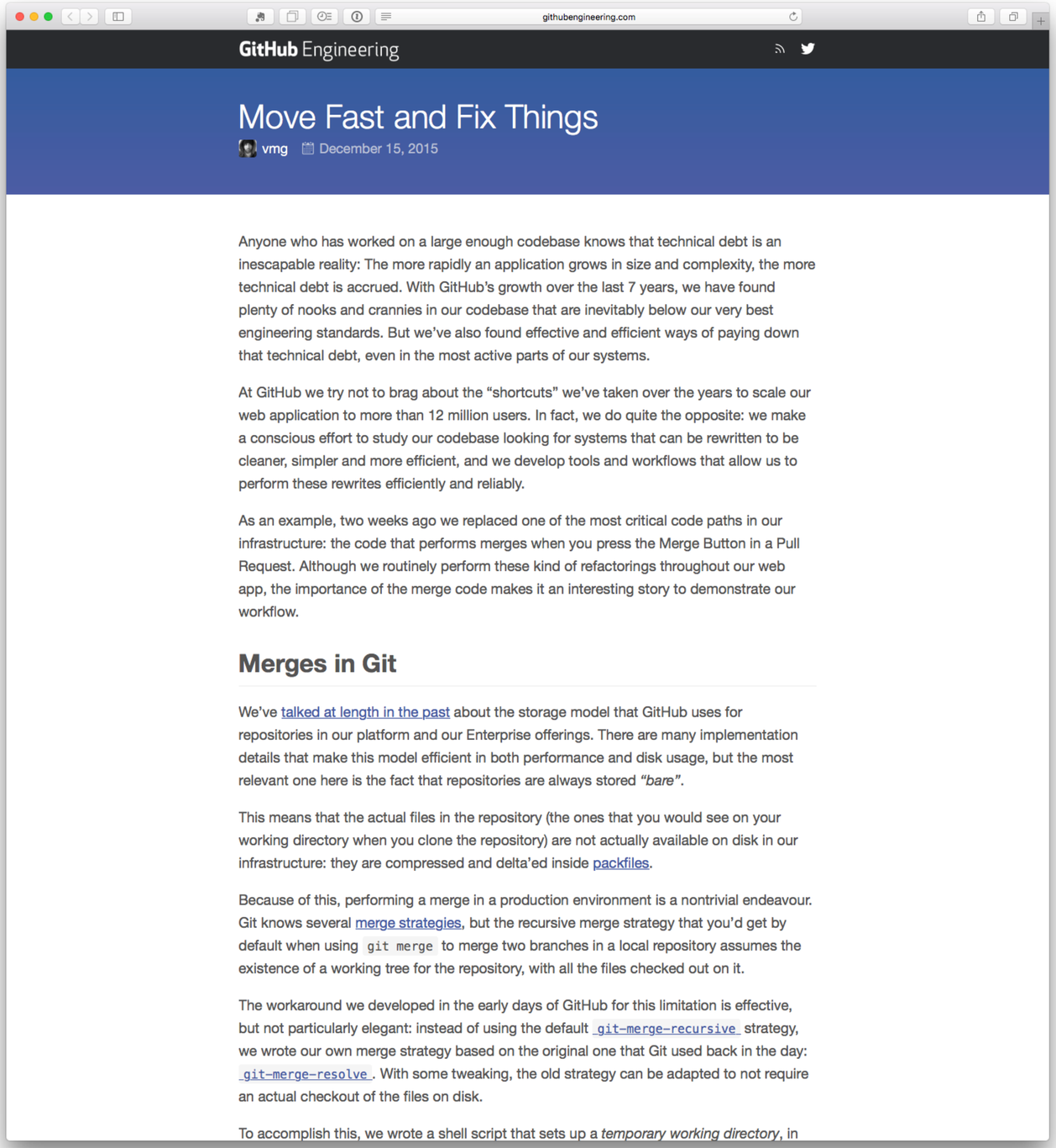


Penultimate









githubengineering.com

GitHub Engineering

Move Fast and Fix Things

vmg December 15, 2015

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

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

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

Merges in Git

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

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

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

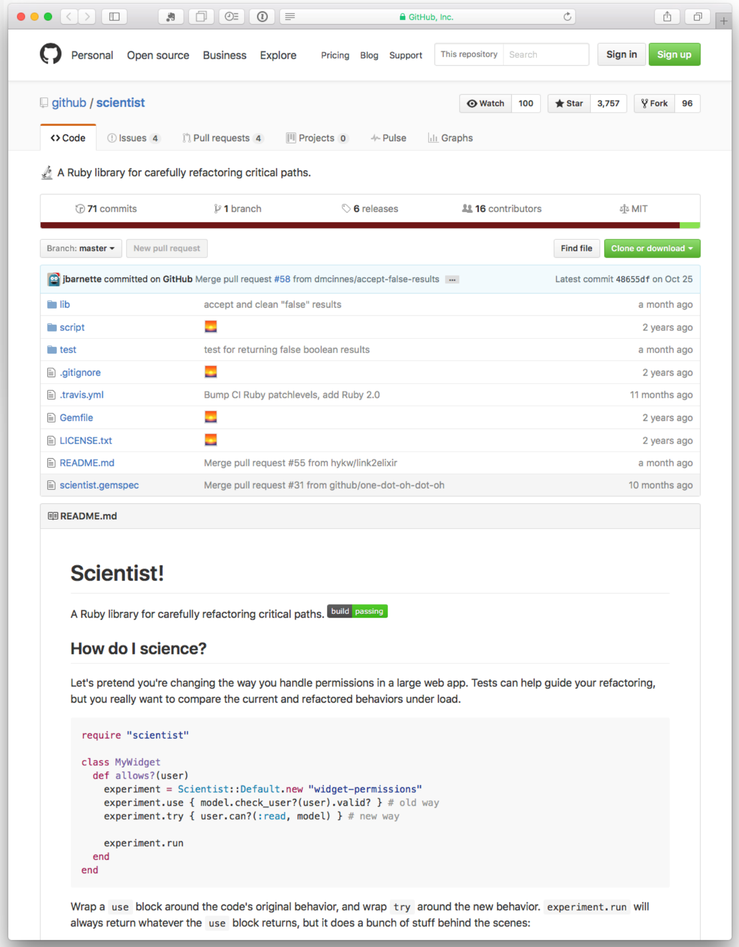
The workaround we developed in the early days of GitHub for this limitation is effective, but not particularly elegant: instead of using the default `git-merge-recursive` strategy, we wrote our own merge strategy based on the original one that Git used back in the day: `git-merge-resolve`. With some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

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

"move
fast
&
fix
things"


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

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

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

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

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

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

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



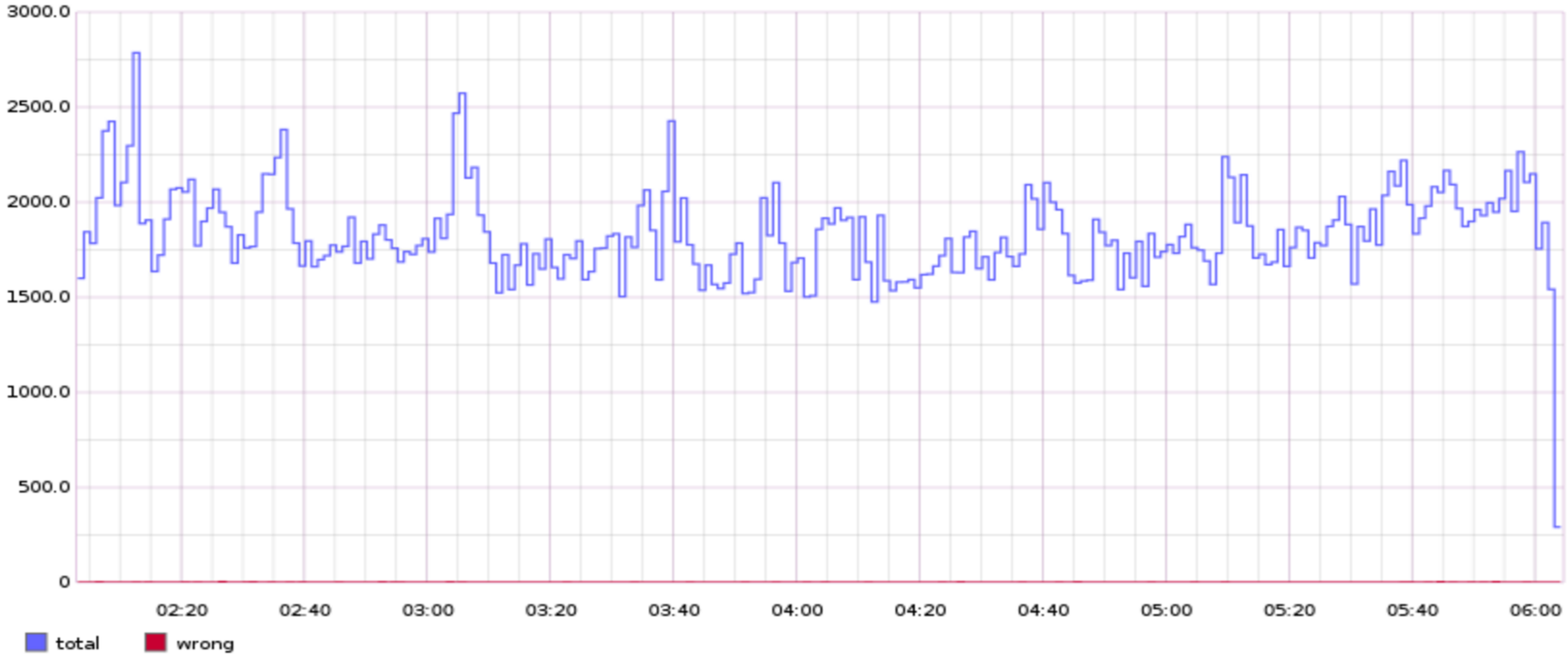
https://github.com/github/scientist


```ruby
require "scientist"

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

    experiment.run
  end
end
```

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

Accuracy

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

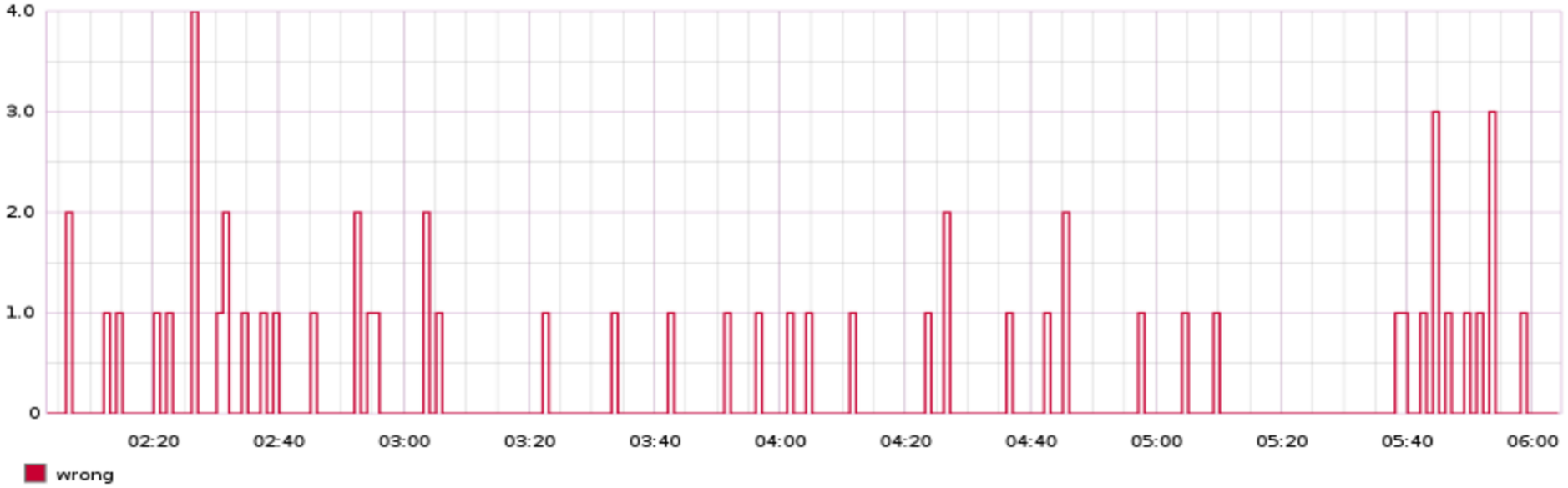


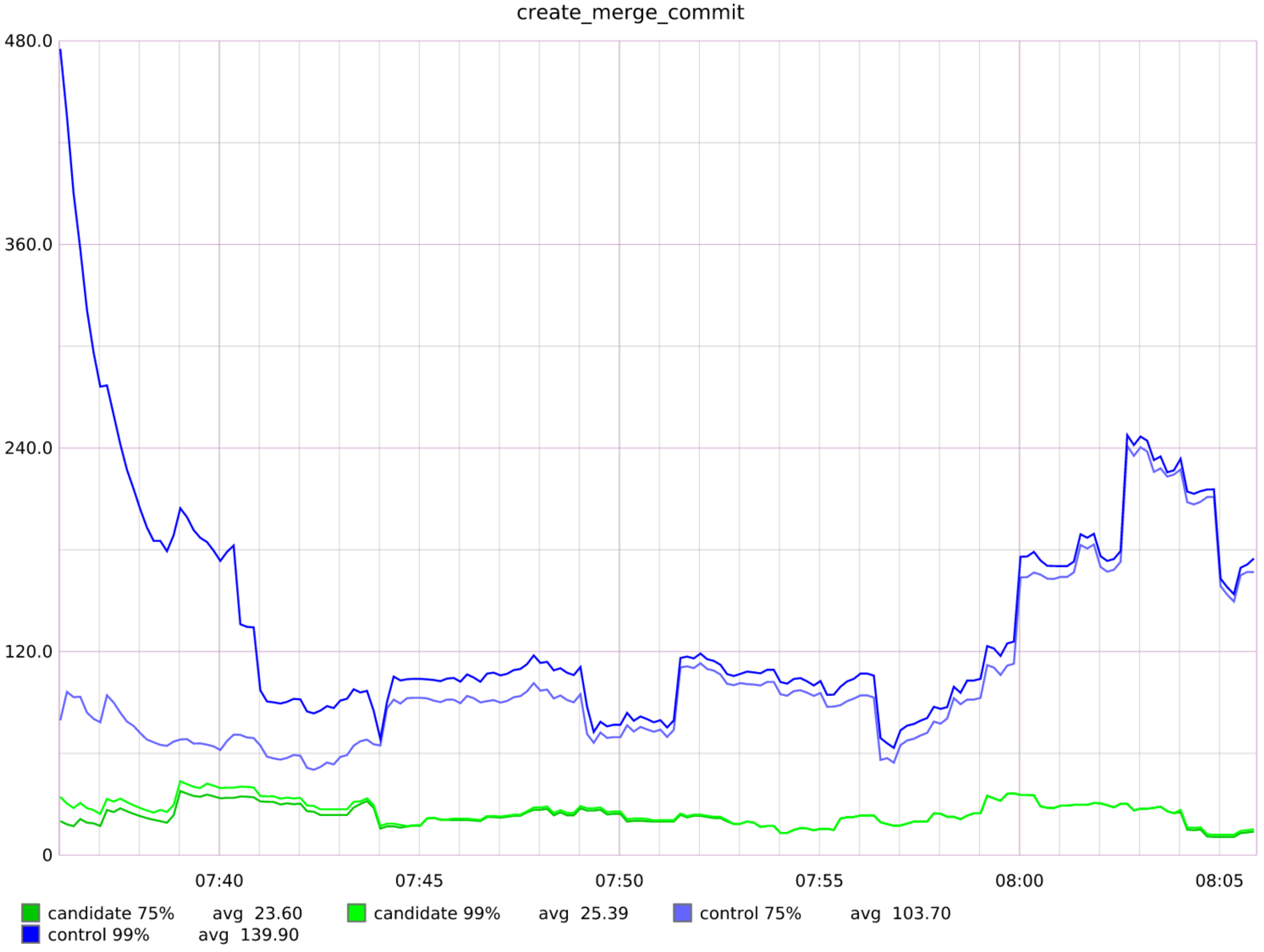

GitHub Engineering

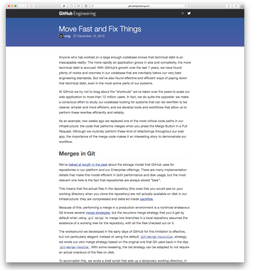
Move Fast and Fix Things

Merges in Git

The number of incorrect/ignored only.



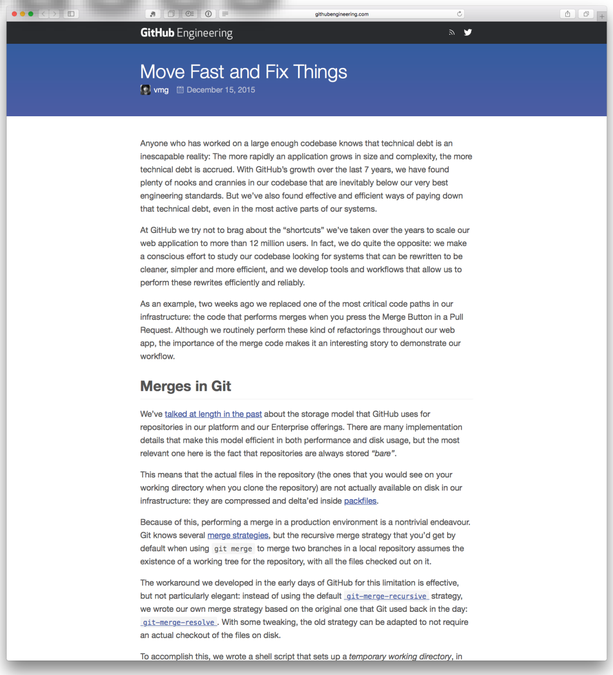

create_merge_commit
480.0
360.0
240.0
120.0
0
07:40
07:45
07:50
07:55
08:00
08:05
candidate 75% avg 23.60
candidate 99% avg 25.39
control 75% avg 103.70
control 99% avg 139.90


GitHub Engineering
Move Fast and Fix Things
Merges in Git

4 days

24 hours/
no mismatches or slow cases

> 10,000,000
comparisons



githubengineering.com

GitHub Engineering

Move Fast and Fix Things

vmg December 15, 2015

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

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

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

Merges in Git

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

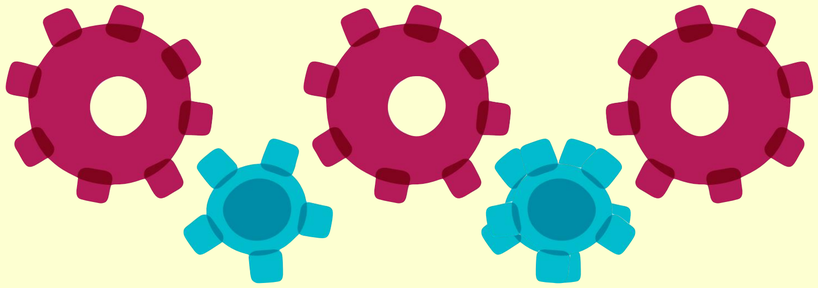
This means that the actual files in the repository (the ones that you would see on your working directory when you clone the repository) are not actually available on disk in our infrastructure: they are compressed and delta'ed inside packfiles.

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

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

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

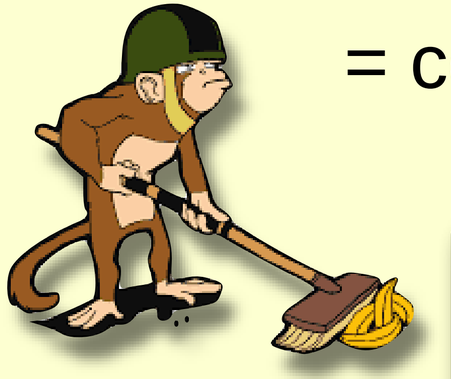
Penultimate



scientist feature toggles

+ fitness function (janitor monkey)

= continuous deployment (w/ less risk)



Data Consistency Fitness Function

Question : How we can ensure that all the data that we are saving is consistent with the legacy application before set the new database as a source of truth ?

Context: We was extrangulating a legacy System, so we created a new microservice to handle an specific part of the domain.

This new service is in production applying a double writing strategy but maintaining the source of truth in the Legacy DB.

We need to ensure the data consistency to evolve :

- Move the source of truth to the new domain DB,
- Remove the old tables in old application DB.
- Remove the old code in the legacy Application.

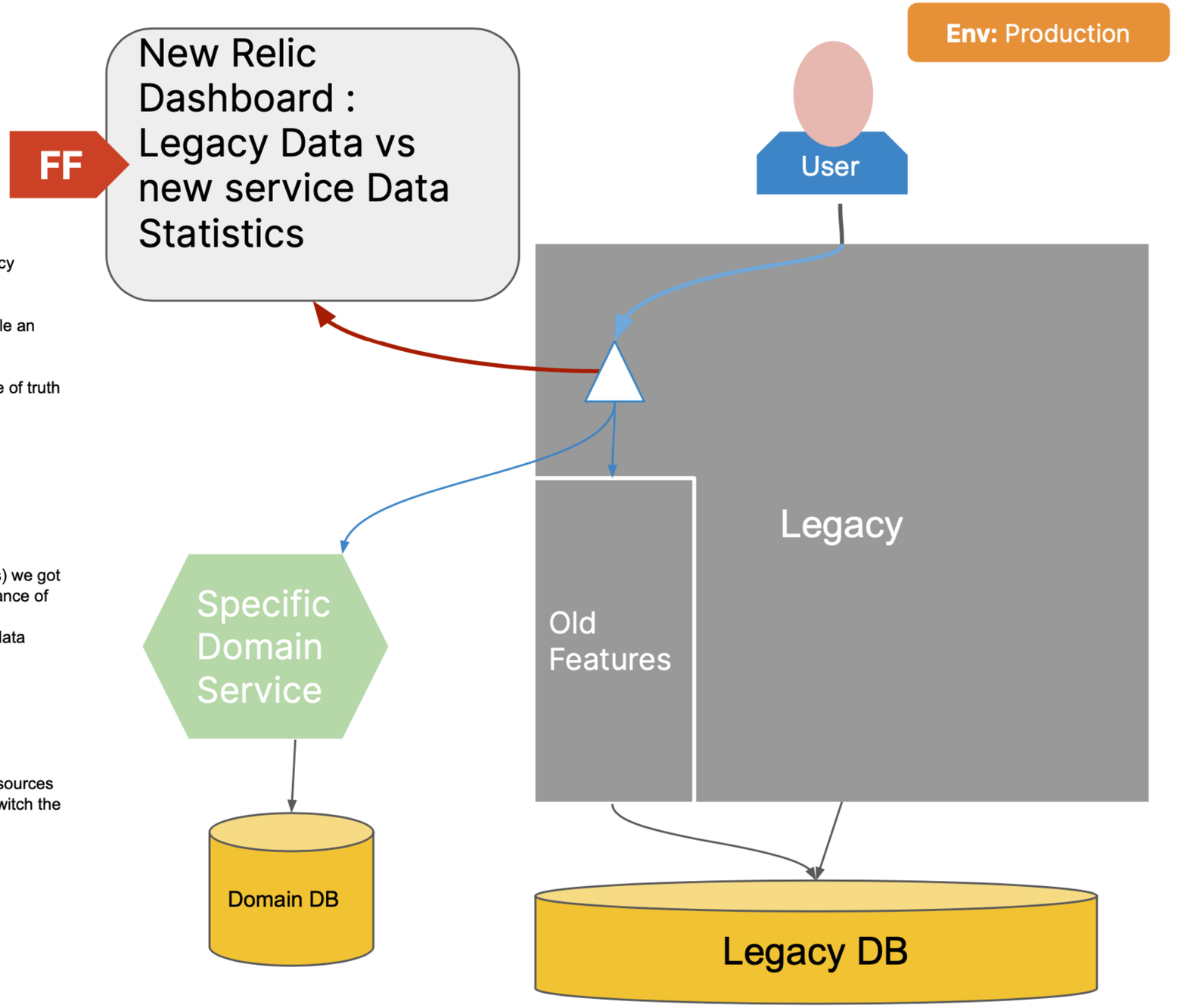
FF:

- In each call from a client(client could be a UI , Mobile App, or other web services) we got the data from Legacy and New service and compare, without affect the performance of the call.
- We Compared the data and only store the statistics, because is some sensible data
- Get a Sample of all the comparison in a couple of days.
- Review the consistency .

Type : Temporal, Automatic, Holistic.
Ility: Consistency, Reliability

Conclusion : It was pretty useful because also we identify some data that comes from other sources that was not mapped for anyone of the company, when we finished the data review, we can switch the source of truth and remove the old code and tables.

Technology : Ruby & Rails, Java, Spring Boot, New Relic
Domain : Health Care



aRChiTeCtuRe fOr cONtiNuoUs DeLiVeRy

Good engineering practices

aRChiTeCtuRe fOr cONtiNuoUs DeLiVeRy

Appropriate structure.

aRChiTeCtuRe fOr cONtiNuoUs DeLiVeRy

Apply good engineering practices to architecture.

aRChiTeCtuRe fOr cONtiNuoUs DeLiVeRy

Ability to move fast without breaking things.

Architecture for Modern Engineering Practices

Neal Ford

director / software architect / meme wrangler

/thoughtworks


@neal4d

nealford.com