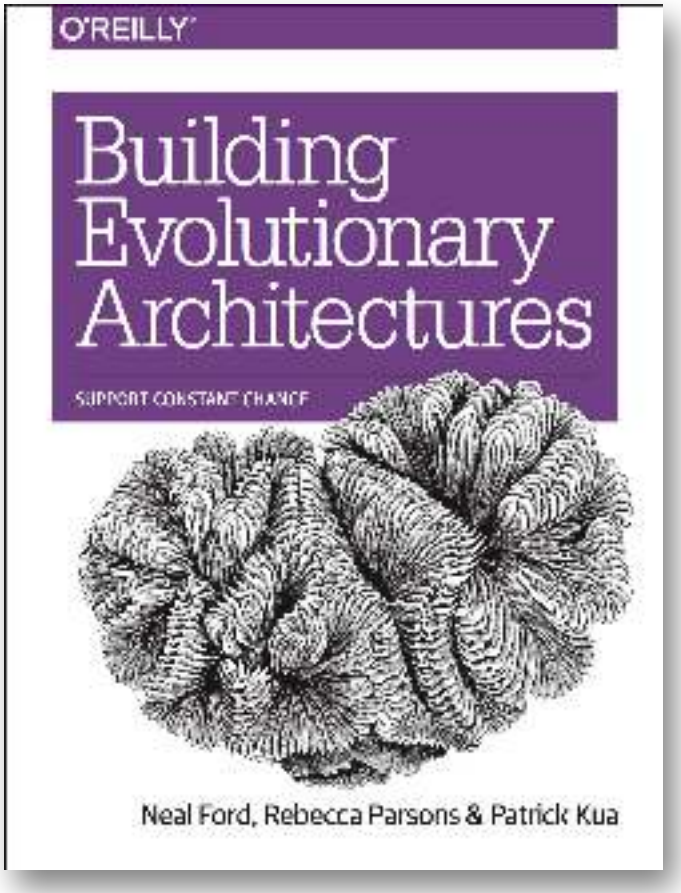
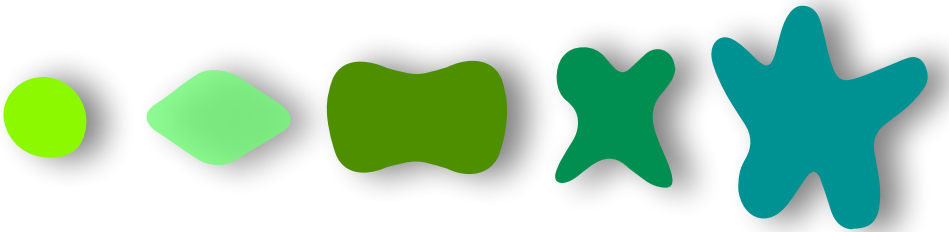
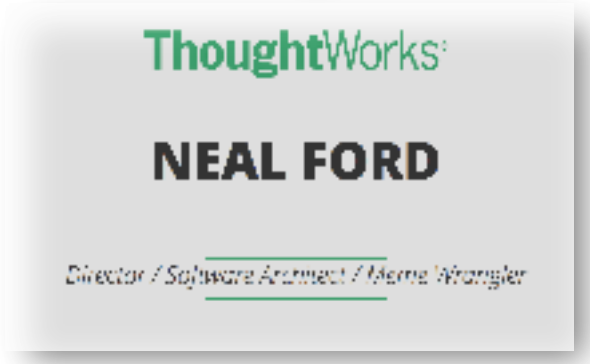
Building Evolutionary Architectures







with Rebecca Parsons & Pat Kua

@neal4d

nealford.com

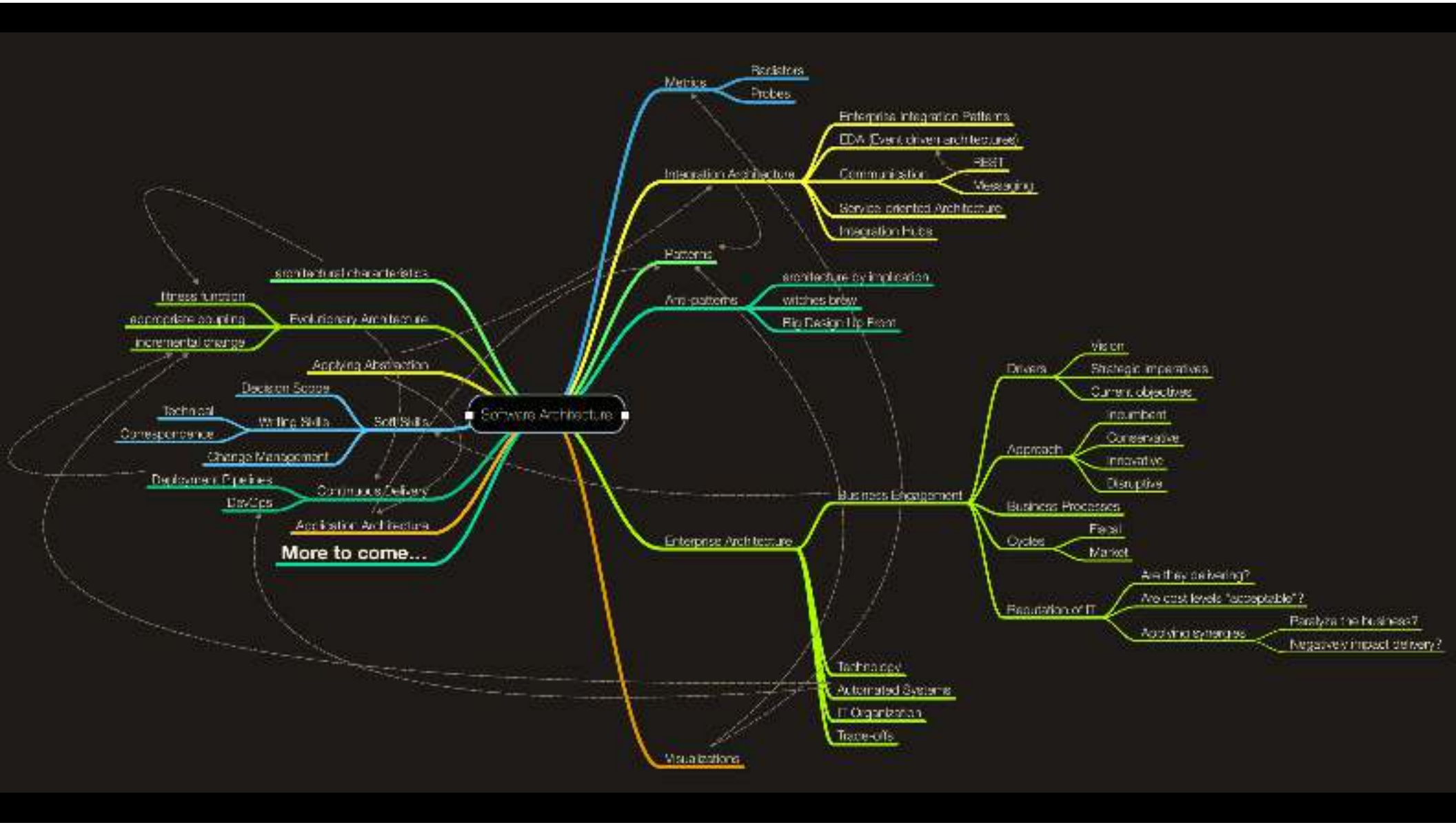
Rebecca Parsons

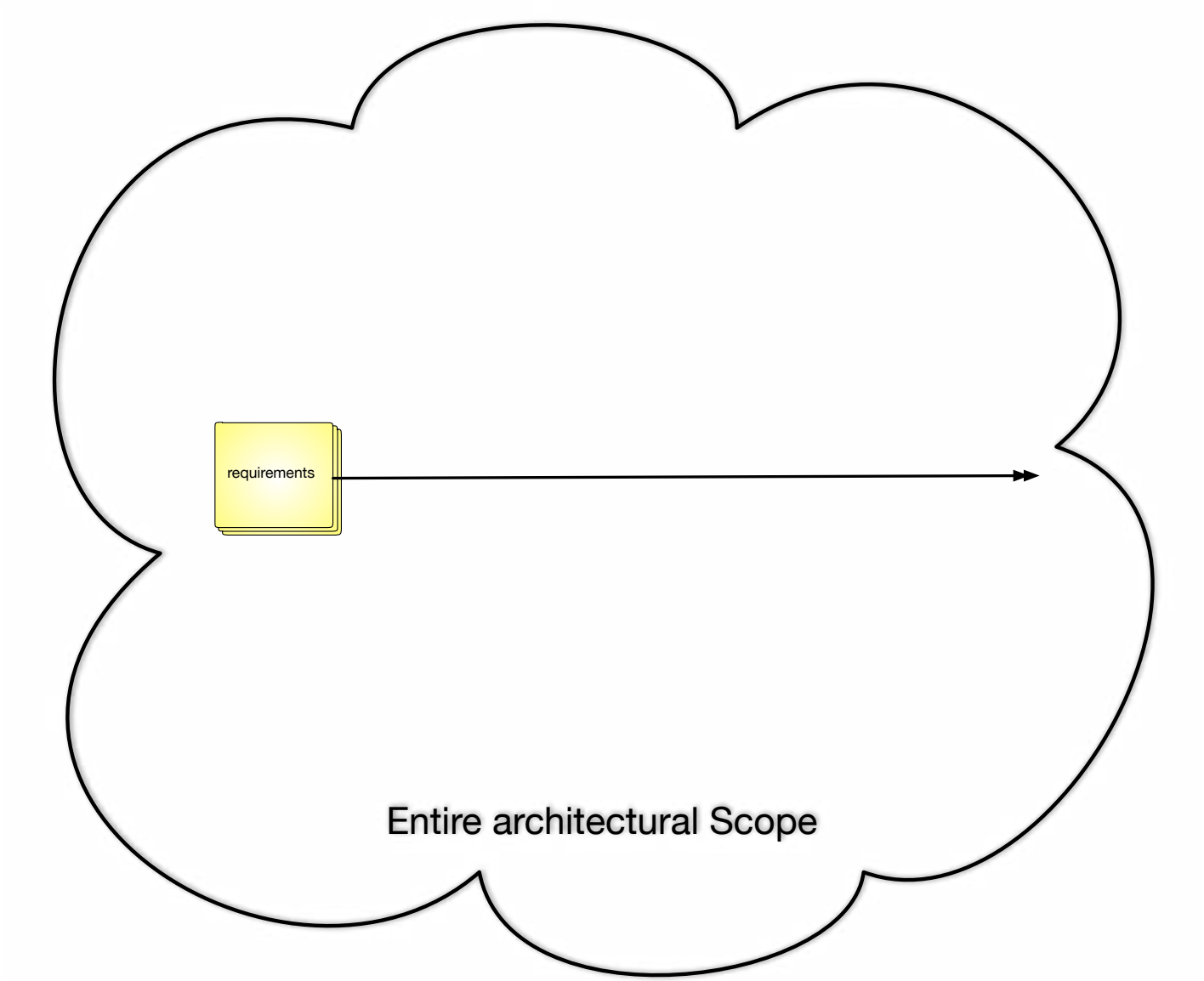
Pat Kua

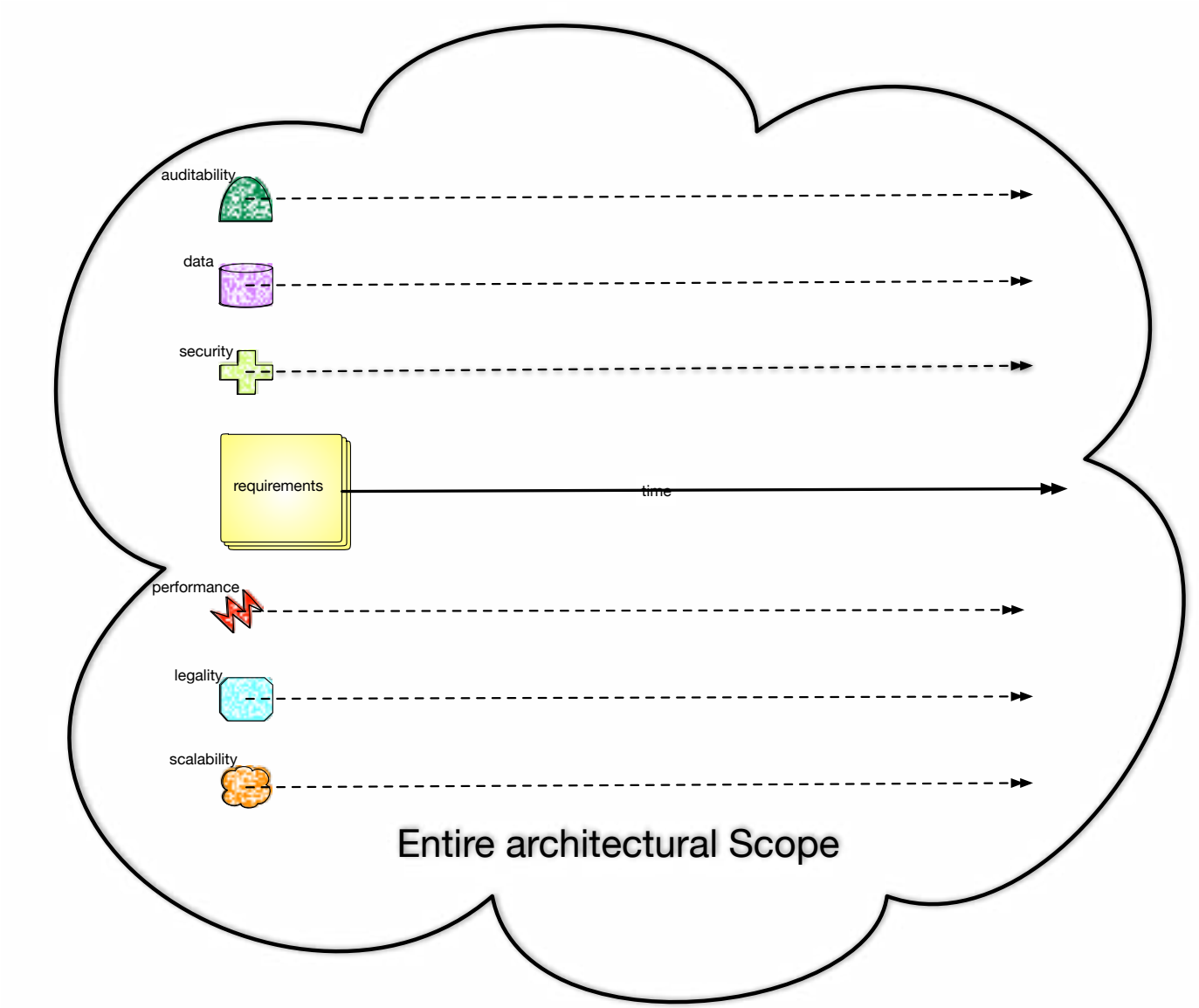
Neal Ford

Photos by Martin Fowler:
http://martinfowler.com/albums/ThoughtWorkers/

What is Software Architecture?


Software Architecture
Metrics
Residence
Probes
Integration Architecture
Enterprise Integration Patterns
EDA (Event driven architecture)
Communication
REST
Messaging
Service oriented Architecture
Integration Rules
Patterns
Anti-patterns
architecture by implication
vitches brew
Big Design Up Front
Business Engagement
Drivers
Vision
Strategic imperatives
Current objectives
Approach
incumbent
Conservative
innovative
Disruptive
Business Processes
Cycles
Fiscal
Market
Reputation of IT
Are they delivering?
Are cost levels "acceptable"?
Are they agile?
Positive impact on business?
Negative impact on agility?
Enterprise Architecture
Technology
Automated Systems
IT Organization
Trade-offs
Visualizations
Evolutionary Architecture
architectural characteristics
fitness function
appropriate coupling
incremental change
Applying Abstraction
Soft Skills
Decision Scope
Writing Skills
Technical
Correspondence
Change Management
Continuous Delivery
Deployment Pipelines
DevOps
Architecture Refactoring
More to come...


requirements
Entire architectural Scope


auditability
data
security
requirements
time
performance
legality
scalability
Entire architectural Scope

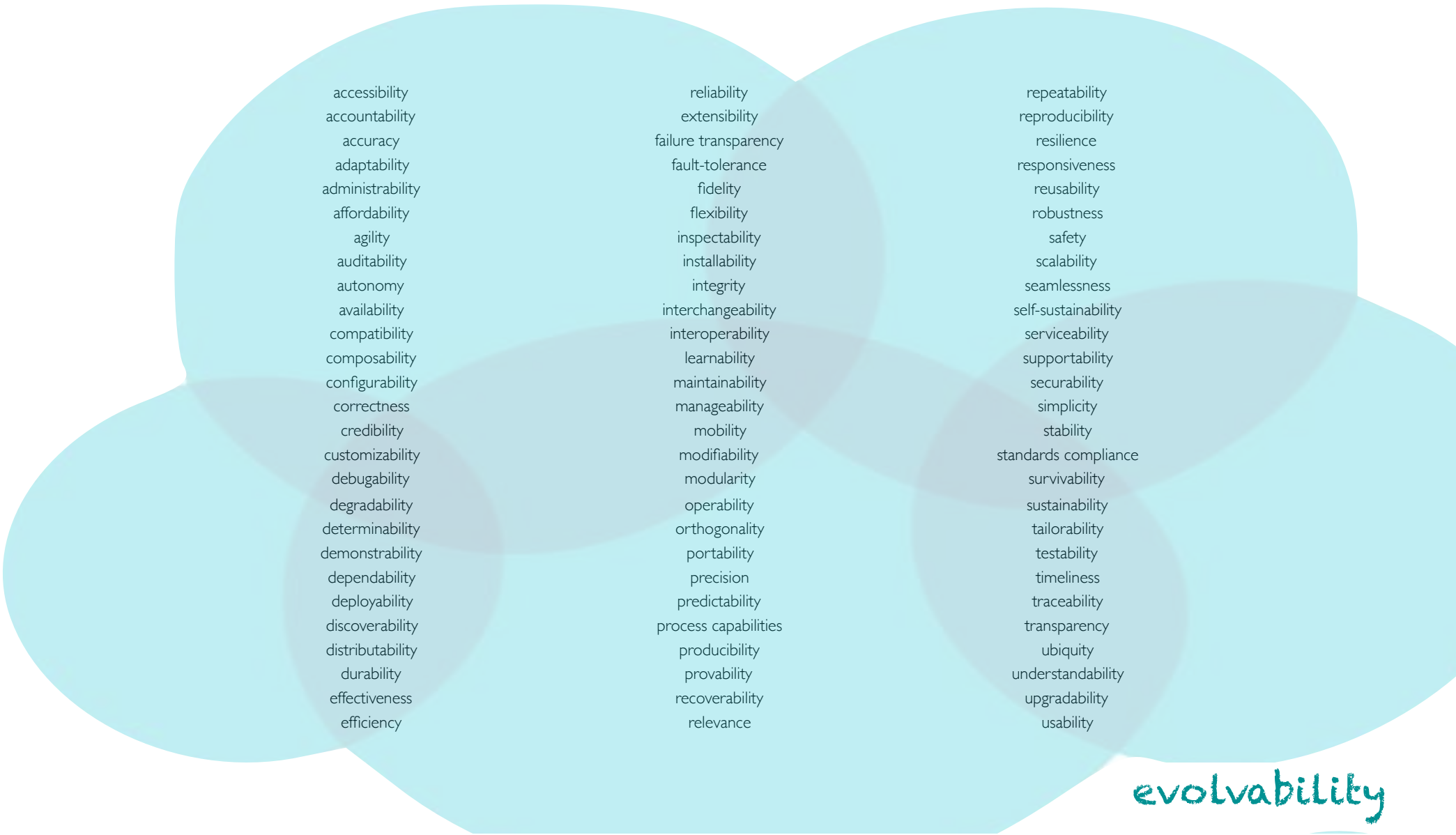
ility

accessibility
accountability
accuracy
adaptability
administrability
affordability
agility
auditability
autonomy
availability
compatibility
composability
configurability
correctness
credibility
customizability
debugability
degradability
determinability
demonstrability
dependability
deployability
discoverability
distributability
durability
effectiveness
efficiency

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

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

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


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

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

How is long term planning possible when things change unexpectedly?

Dynamic Equilibrium

> Architecture is the decisions
> that you wish you could get right early
> in a project.

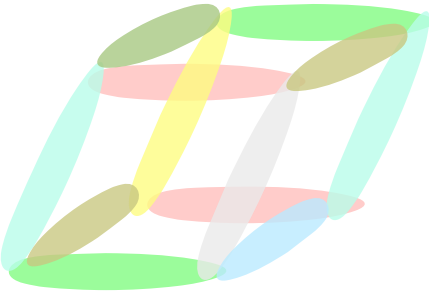
— Ralph Johnson

things that people perceive
as hard to change.

What if we build architectures that expect change?

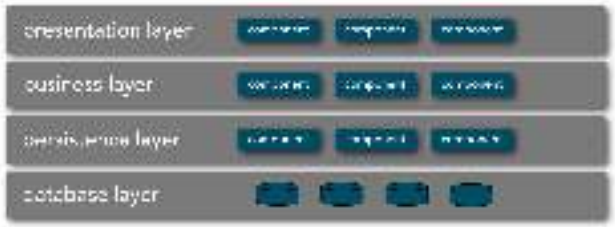
Definition:

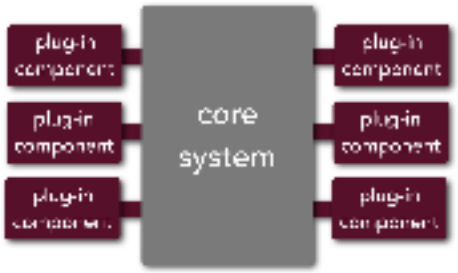
An evolutionary architecture supports incremental, guided change as a first principle across multiple dimensions.

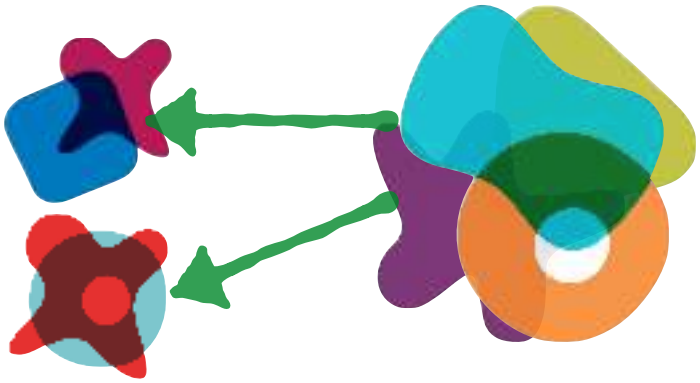


Perspectives on Architecture

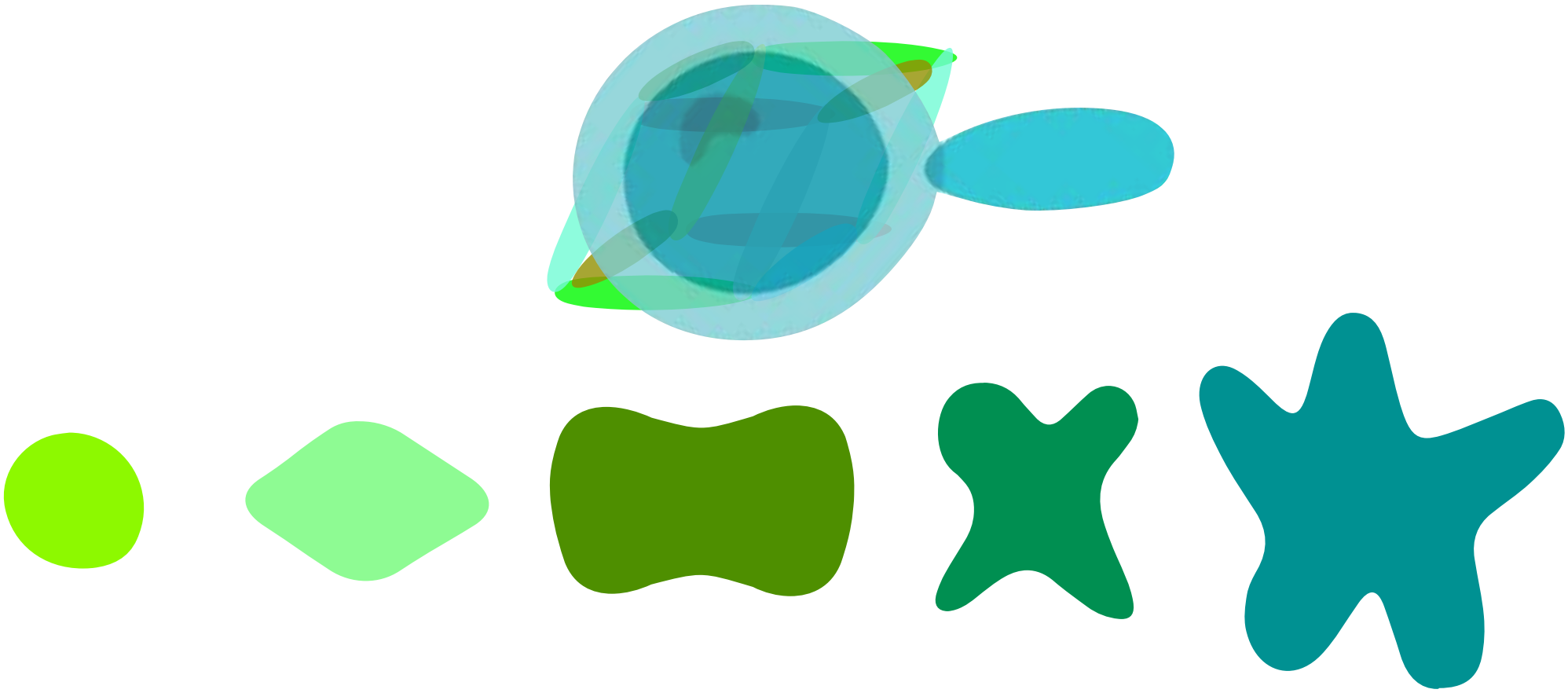
Technical Architecture



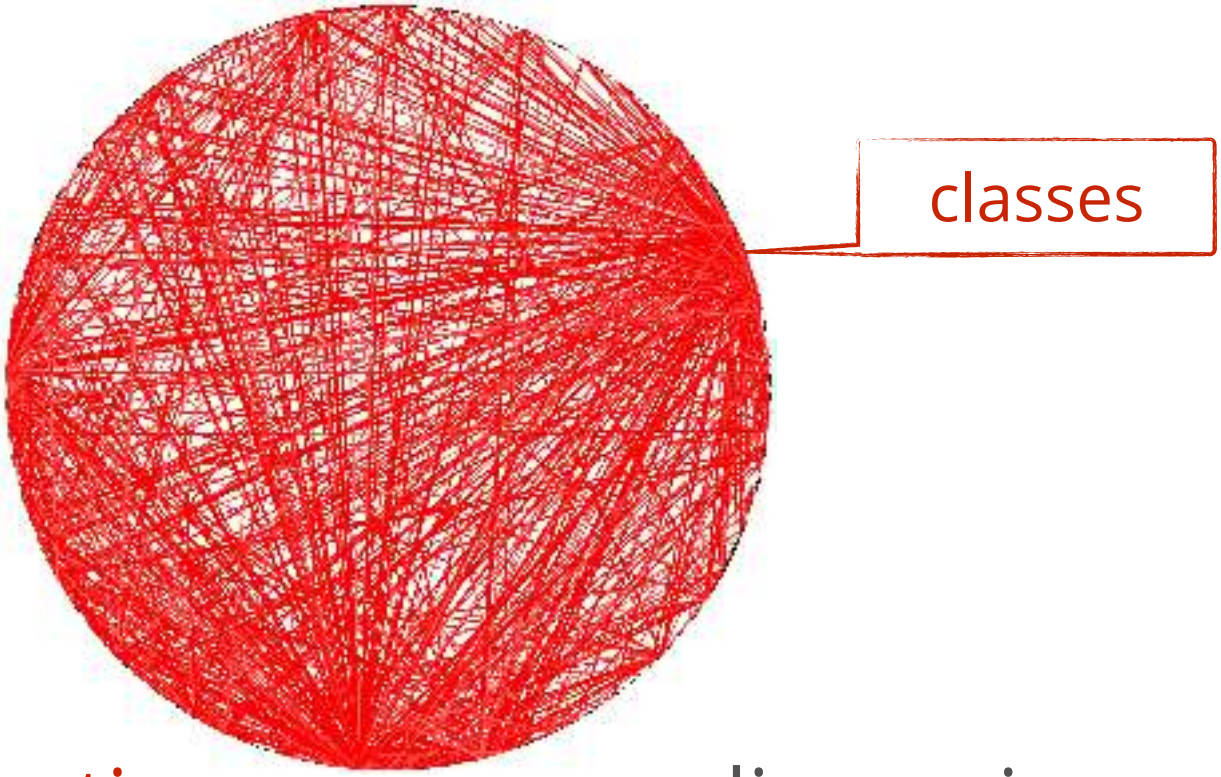




Evolvability of Architectures



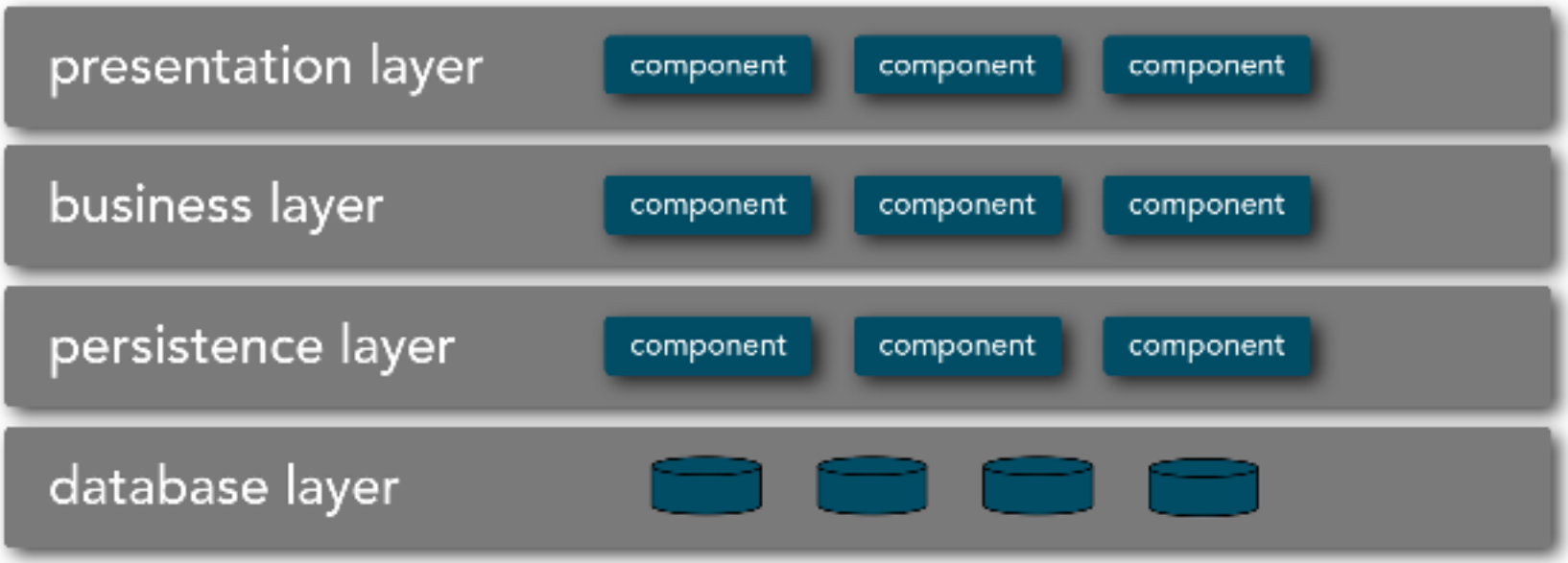
Big Ball of Mud



coupling connections

dimensions : 

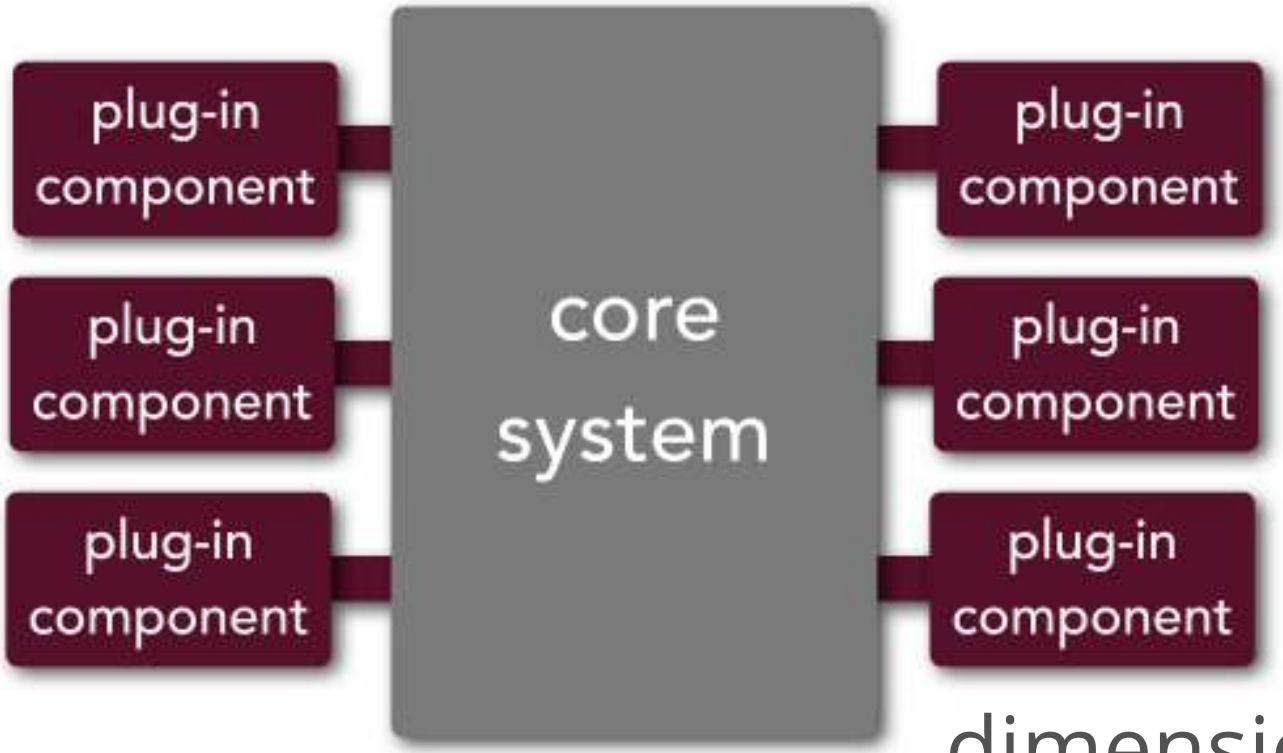
Layered Architecture



opportunities: 4

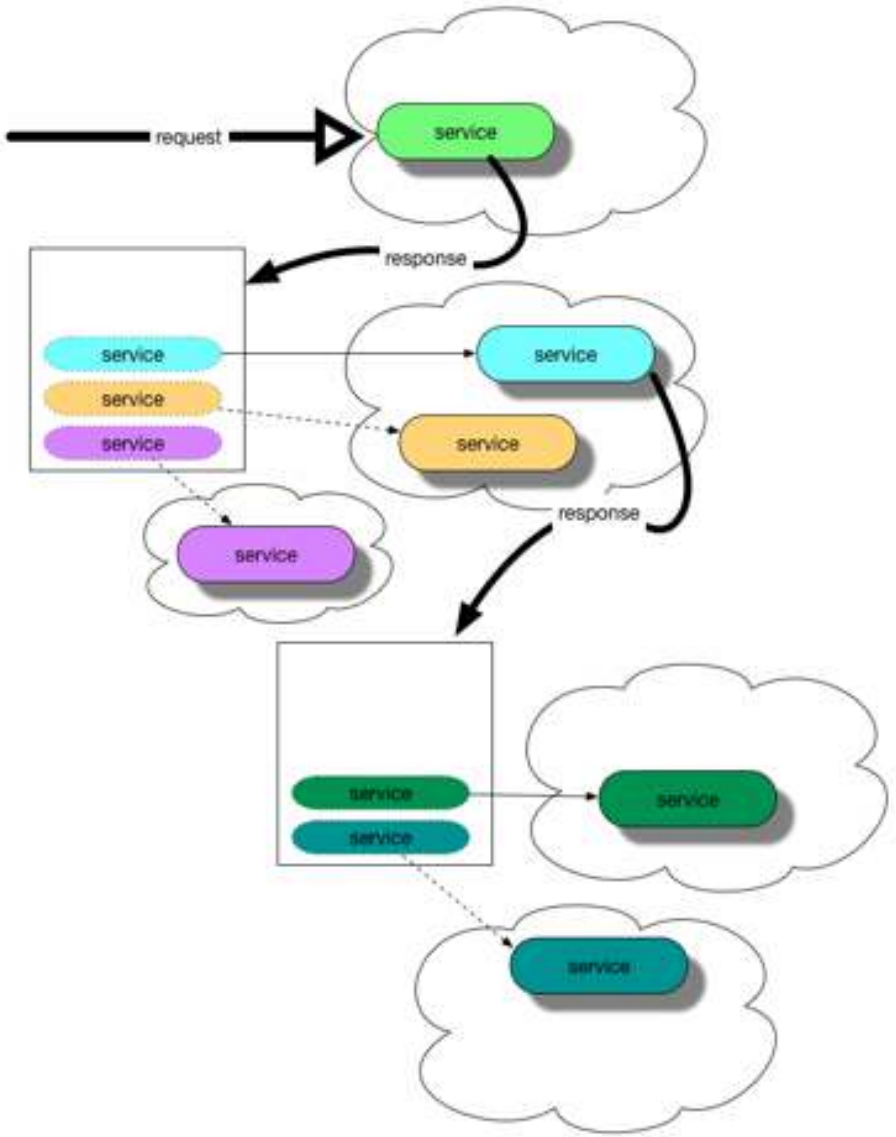
dimensions : 1

Microkernel



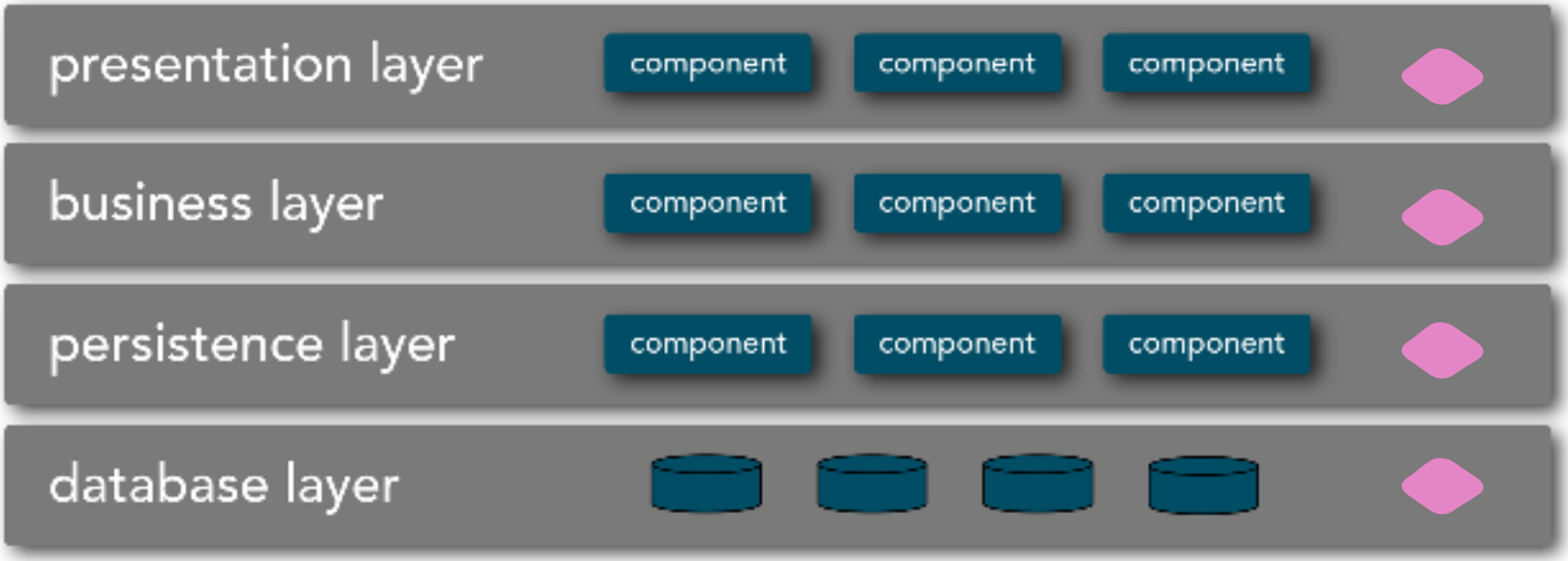
dimensions : 1

REST



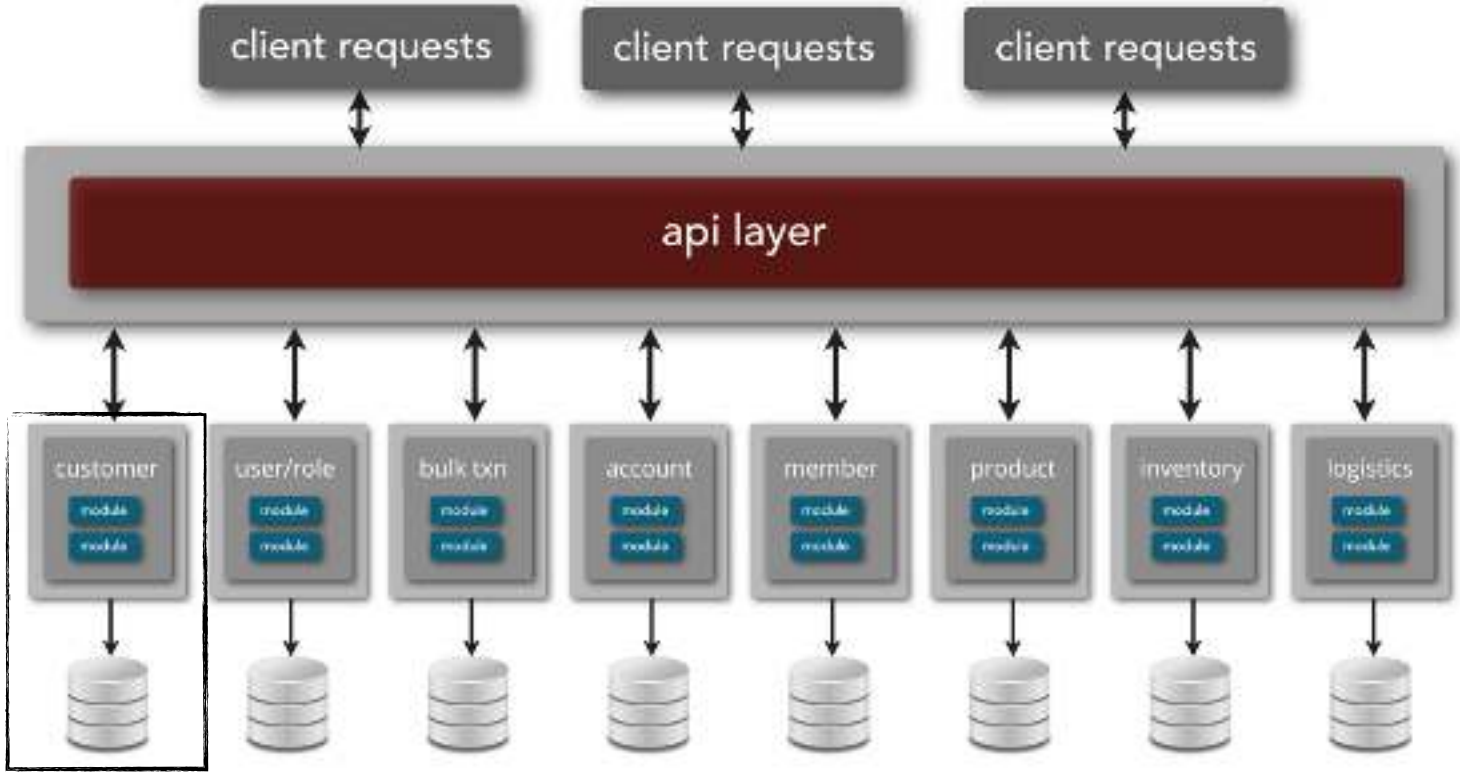
dimensions : 1

Domain Perspective



dimensions :

Microservices

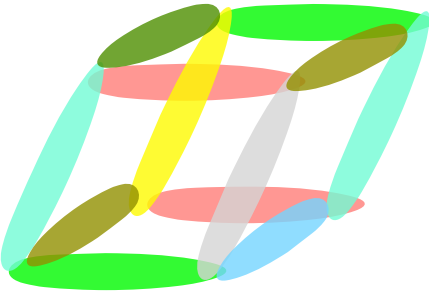


evolutionary architecture dimensions : n

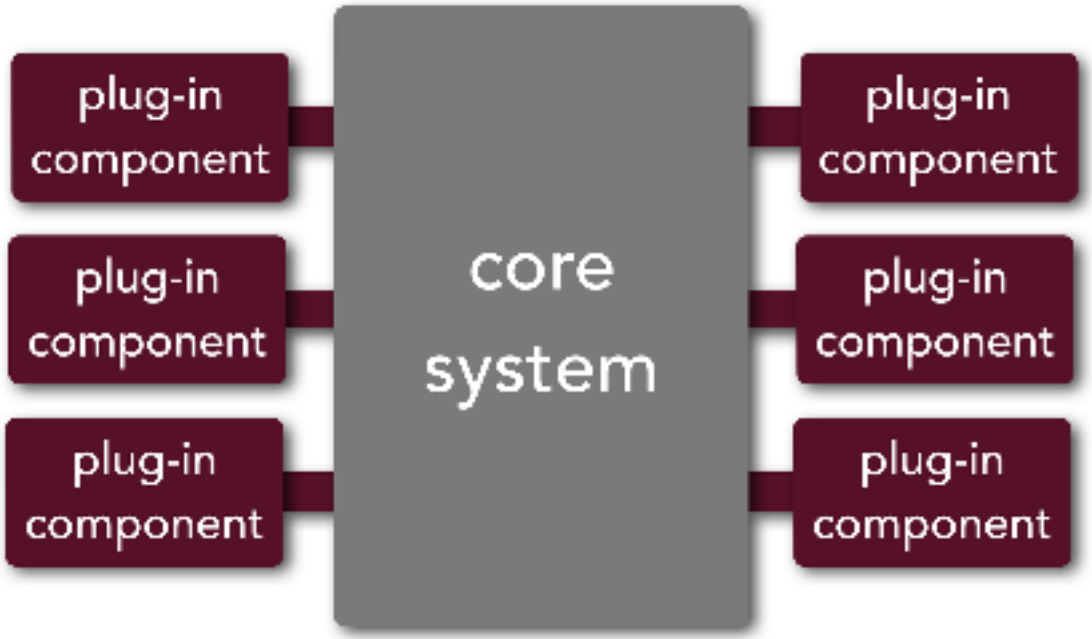
Definition:

evolutionary architecture

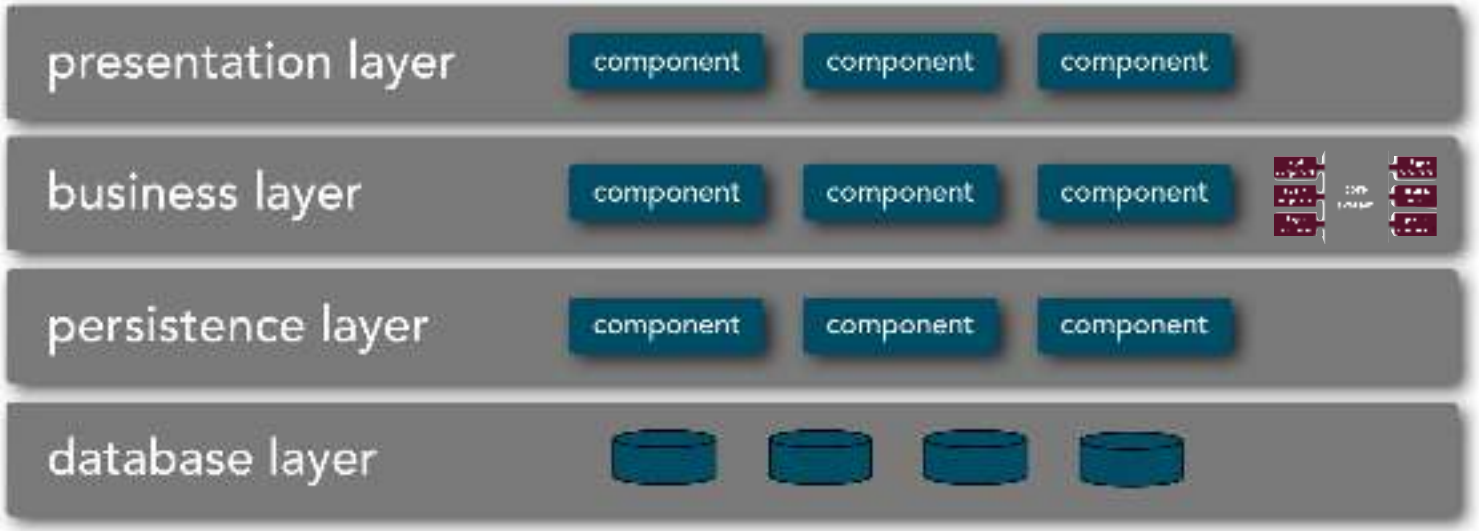
An evolutionary architecture supports incremental, guided change as a first principle across multiple dimensions.



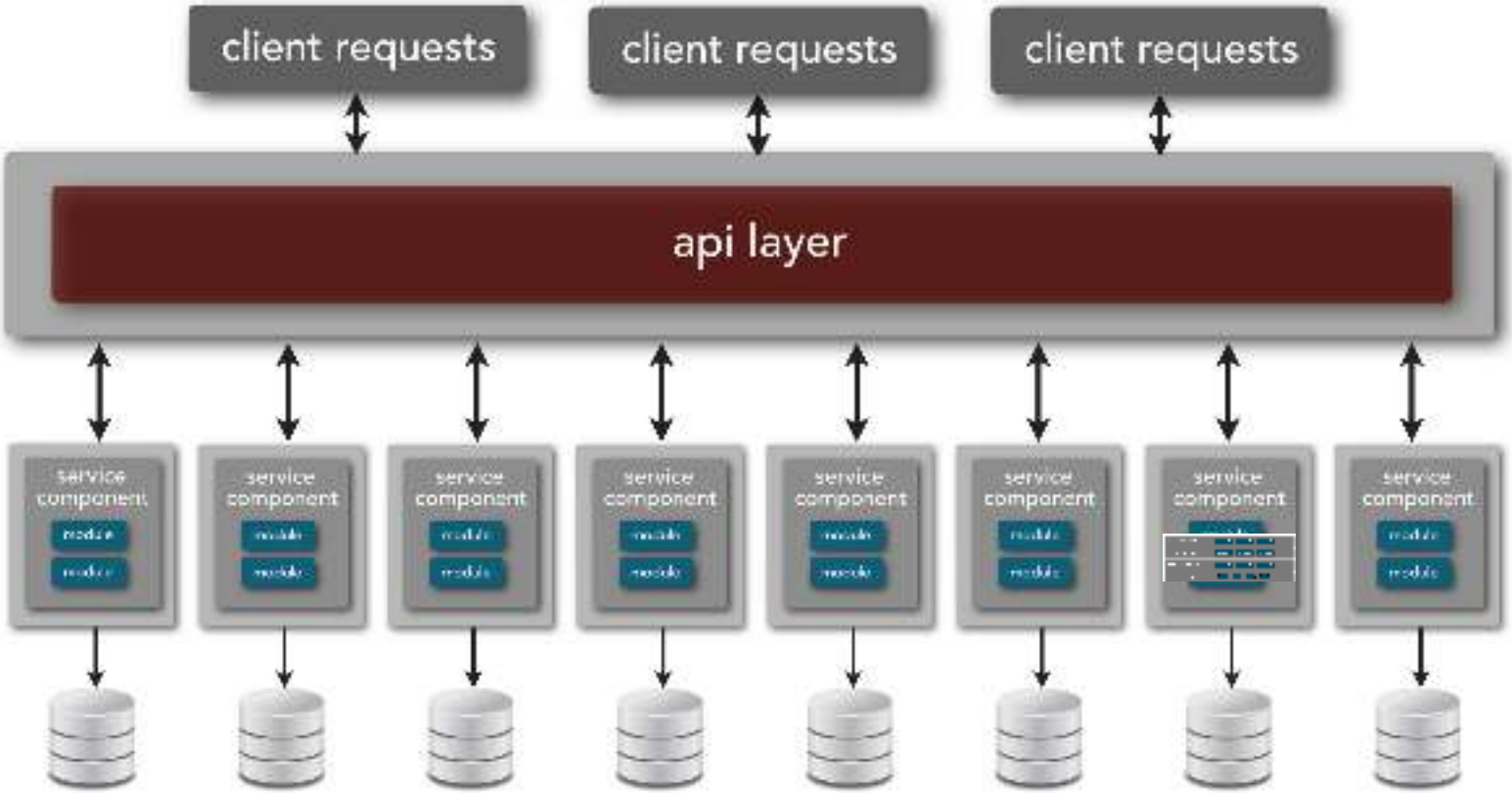
Composability



Composability



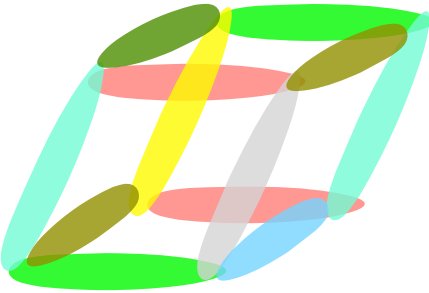
Composability



Definition:

evolutionary architecture

An evolutionary architecture supports incremental, guided change as a first principle across multiple dimensions.

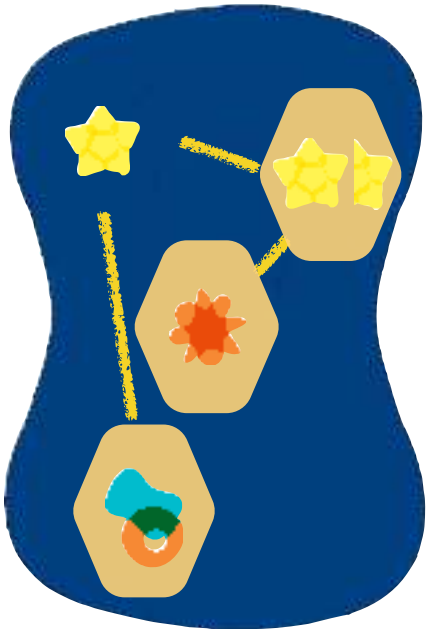


Incremental Change

Components are *deployed*.

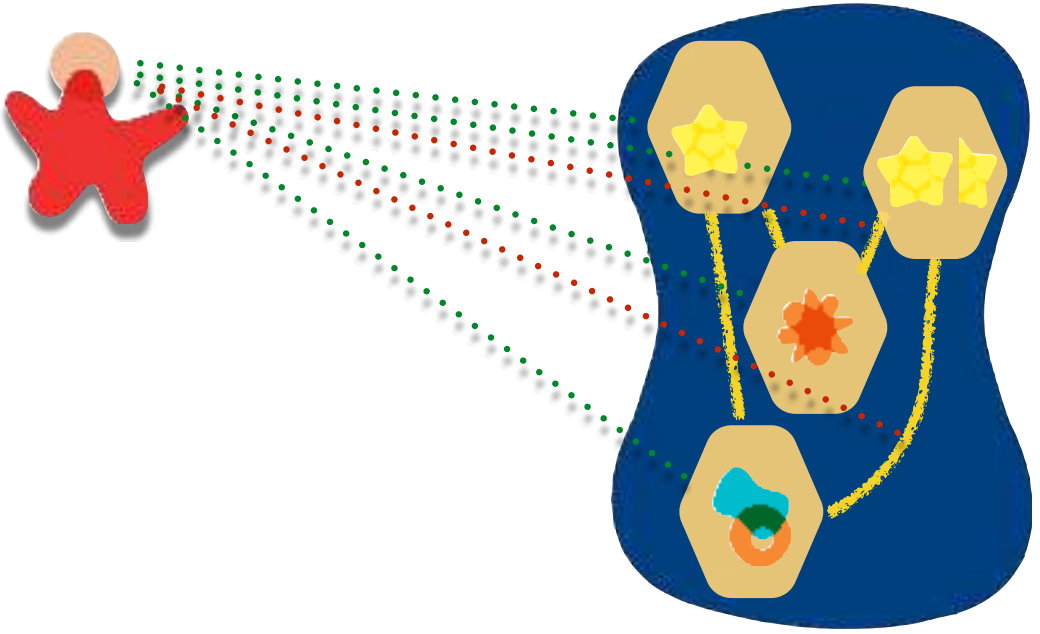
Features are released.

Applications consist of routing.



production

Incremental Change

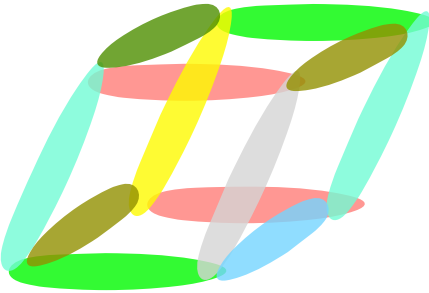


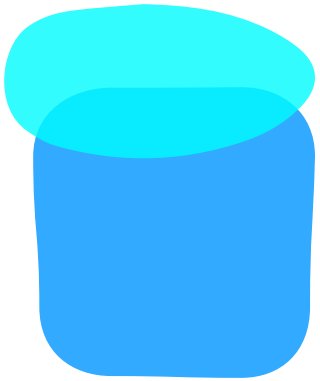
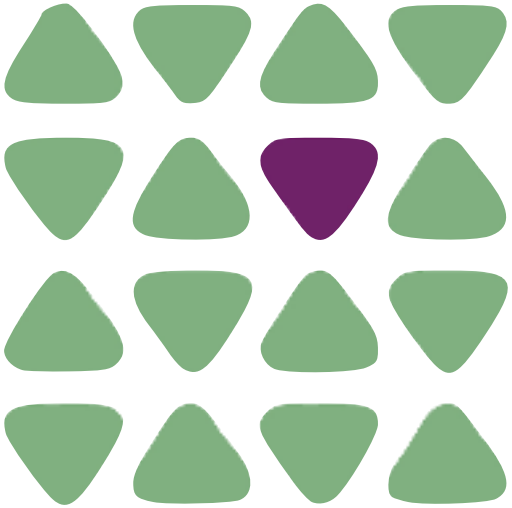
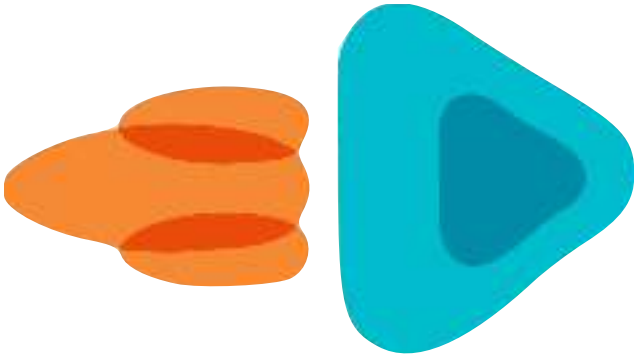
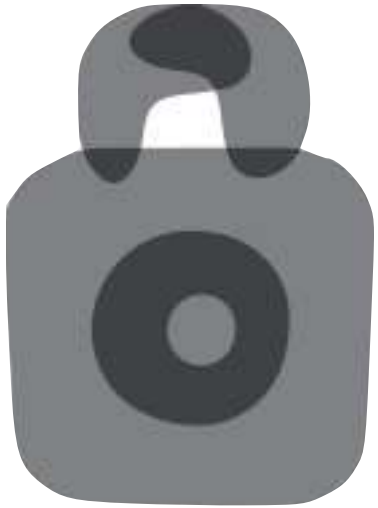
production

Definition:

evolutionary architecture

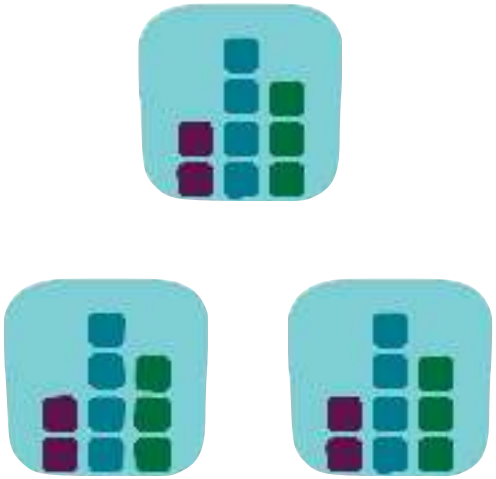
An evolutionary architecture supports incremental, guided change as a first principle across multiple dimensions.

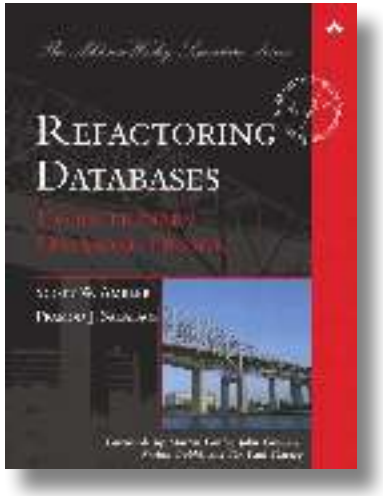


Perspectives on Architecture

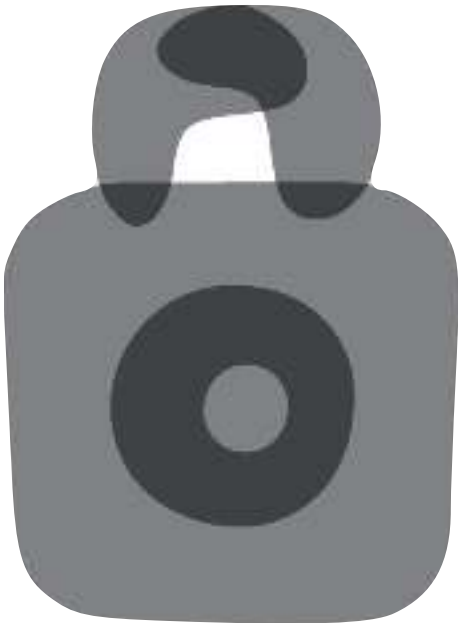
Data Architecture





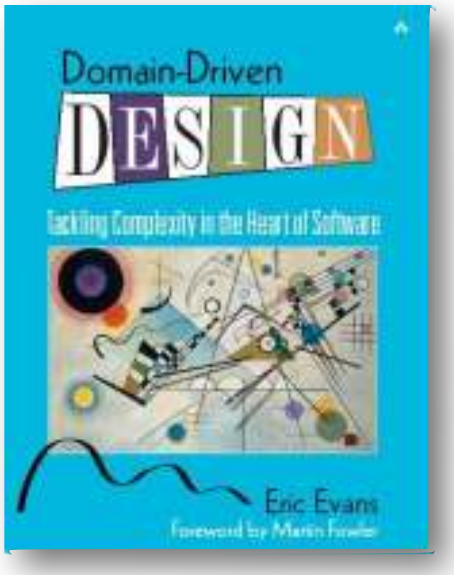
Perspectives on Architecture

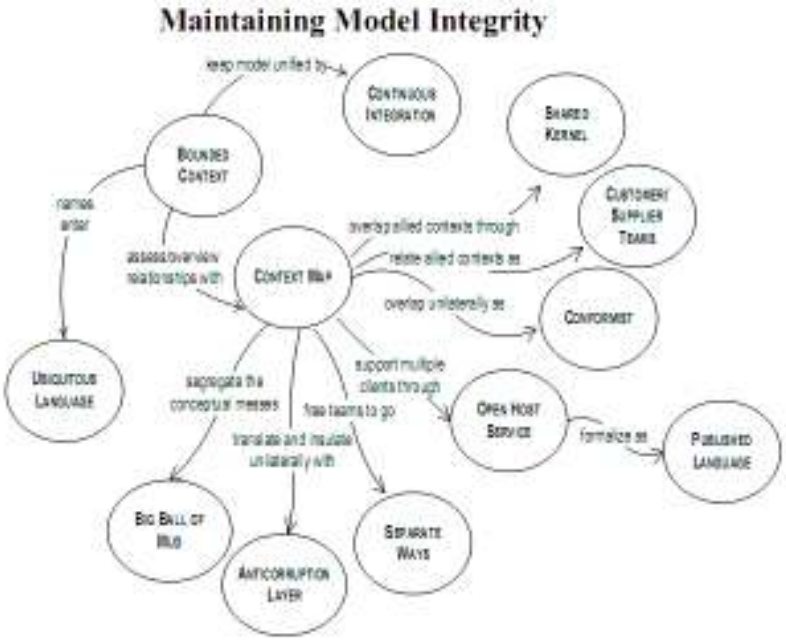
Security Architecture



Perspectives on Architecture

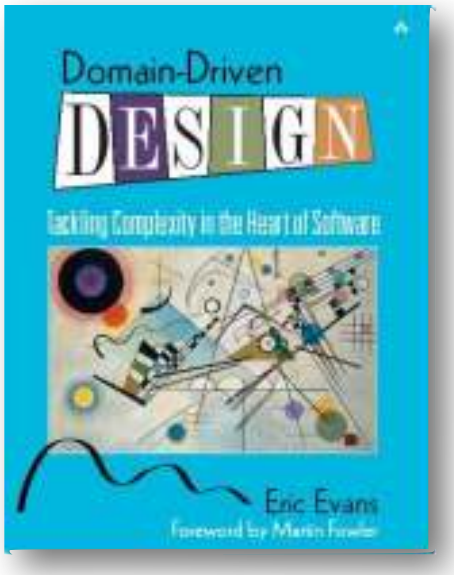
Domain Architecture

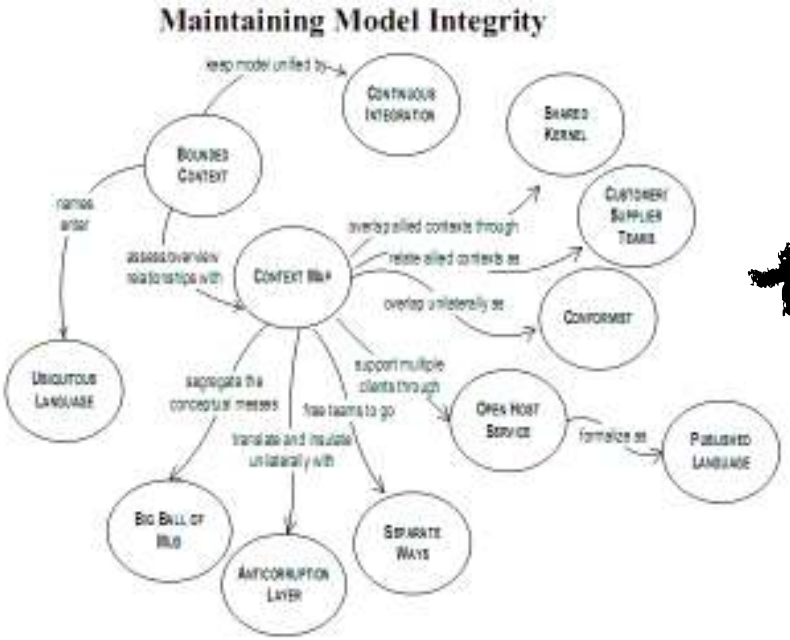




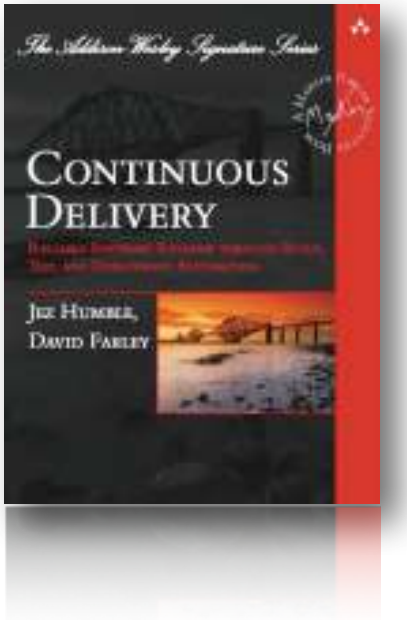
Microservices

Domain Architecture





+



Fitness Functions

ω

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

Architecture Fitness Functions

ω

metrics

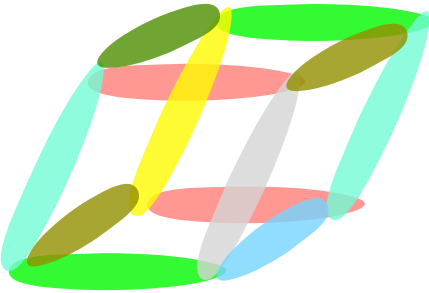
tests

Σ

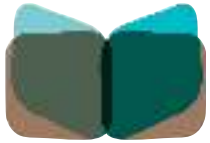
Definition:

evolutionary architecture

An evolutionary architecture supports incremental, guided change as a first principle across multiple dimensions.



Agenda

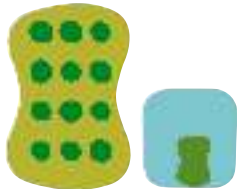


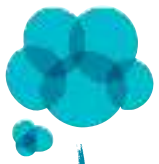
definition

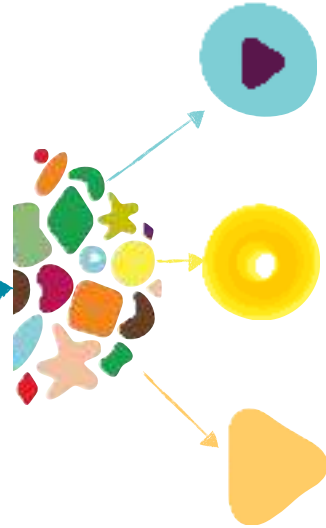
incremental change

fitness functions

appropriate coupling







Fitness Function

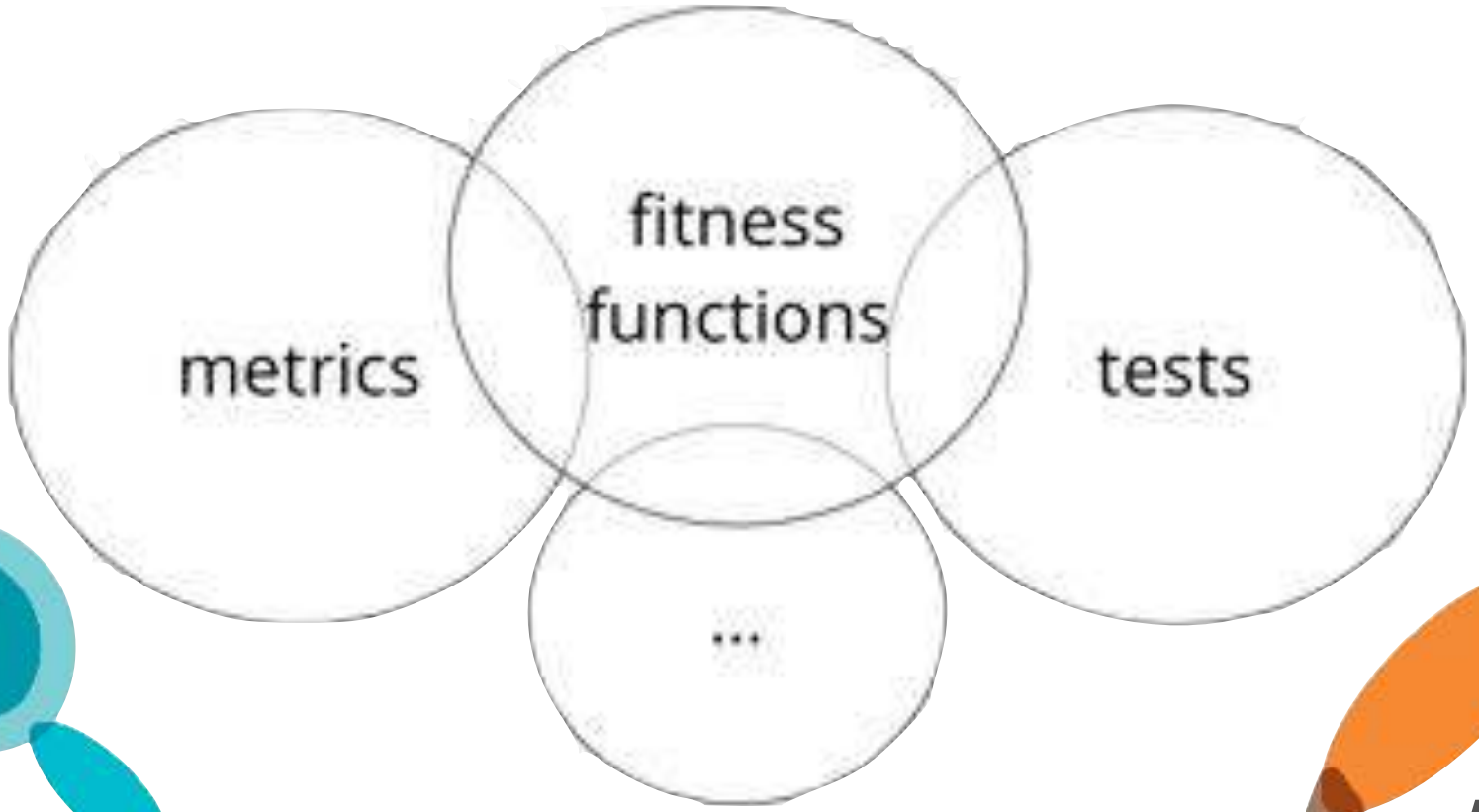
a particular type of objective function that is used to summarize...how close a given design solution is to achieving the set aims.

Architecture Fitness Functions

metrics

tests

Architecture Fitness Functions

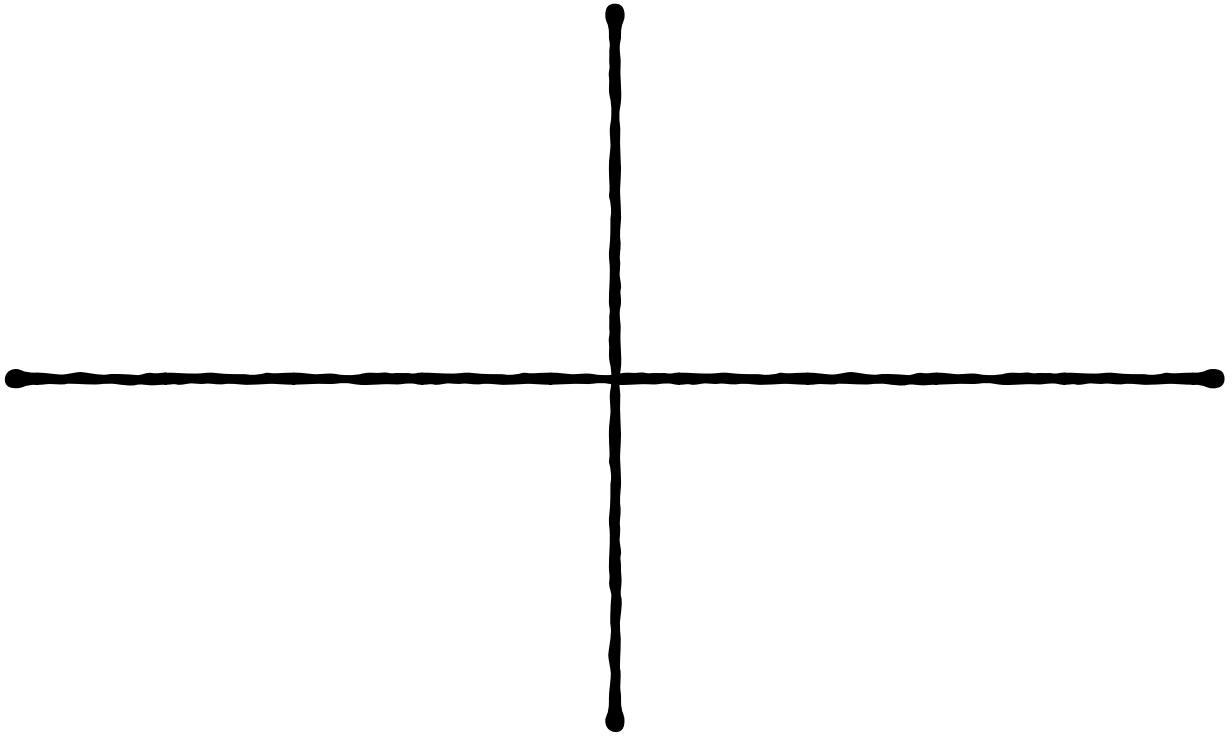


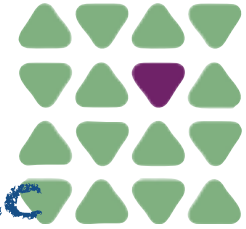
metrics

tests

Fitness Function

atomic





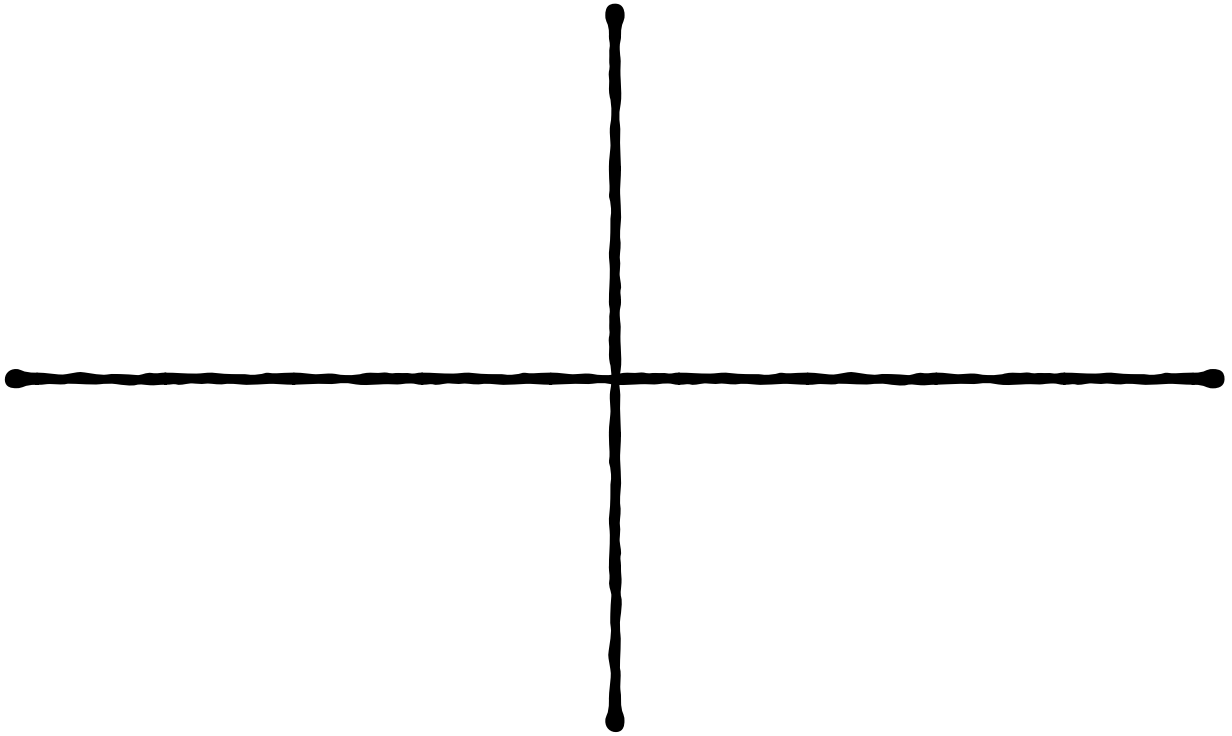
holistic

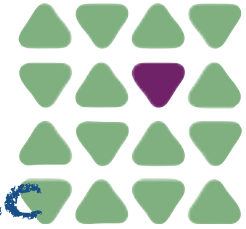
batch

continuous

Fitness Function

atomic





holistic

batch

continuous

Fitness Function

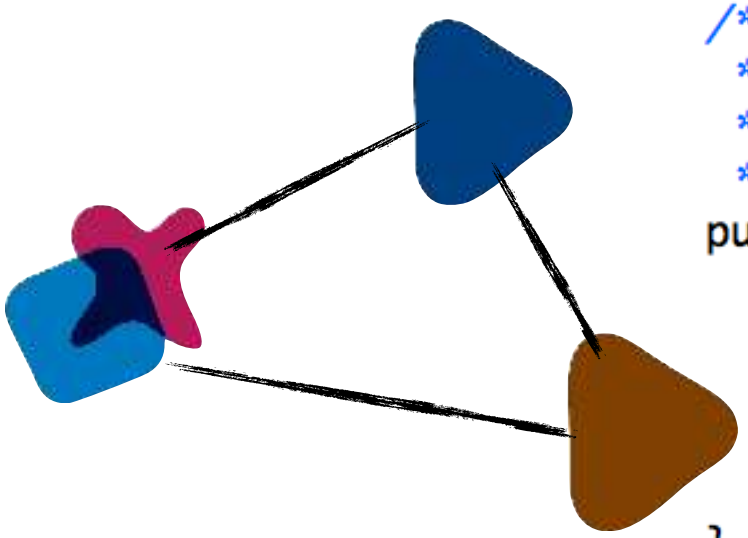
atomic

batch

holistic

continuous

Cyclic Dependency Function



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

    Collection packages = jdepend.analyze();

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

clarkware.com/software/JDepend.html

Coupling Fitness Function

```
protected void setUp() throws IOException {
    jdepend = new JDepend();
    jdepend.addDirectory("/path/to/project/util/classes");
    jdepend.addDirectory("/path/to/project/ejb/classes");
    jdepend.addDirectory("/path/to/project/web/classes");
}


public void testMatch() {
    DependencyConstraint constraint = new DependencyConstraint();
    JavaPackage ejb  = constraint.addPackage("com.xyz.ejb");
    JavaPackage web  = constraint.addPackage("com.xyz.web");
    JavaPackage util = constraint.addPackage("com.xyz.util");

    ejb.dependsUpon(util);
    web.dependsUpon(util);

    jdepend.analyze();

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

Fitness Function

atomic

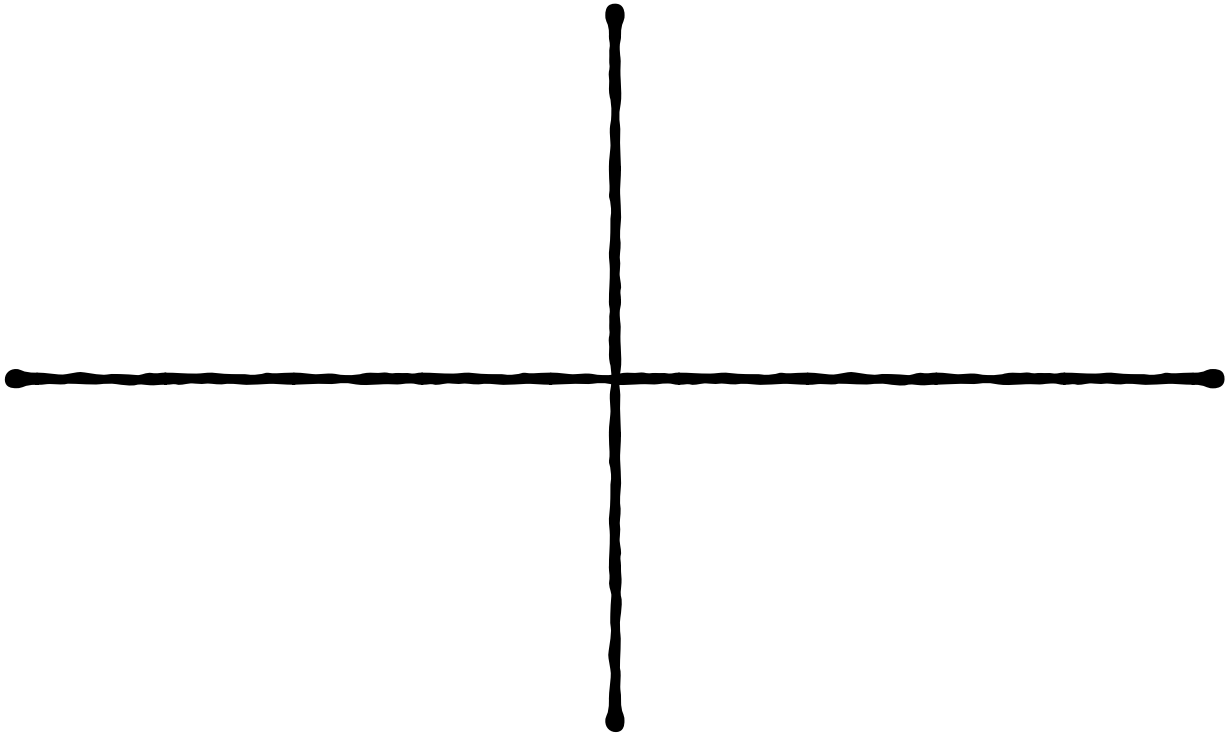
batch

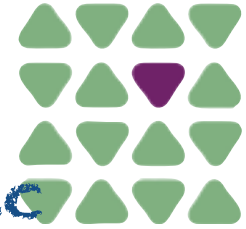
holistic

continuous

Fitness Function

atomic





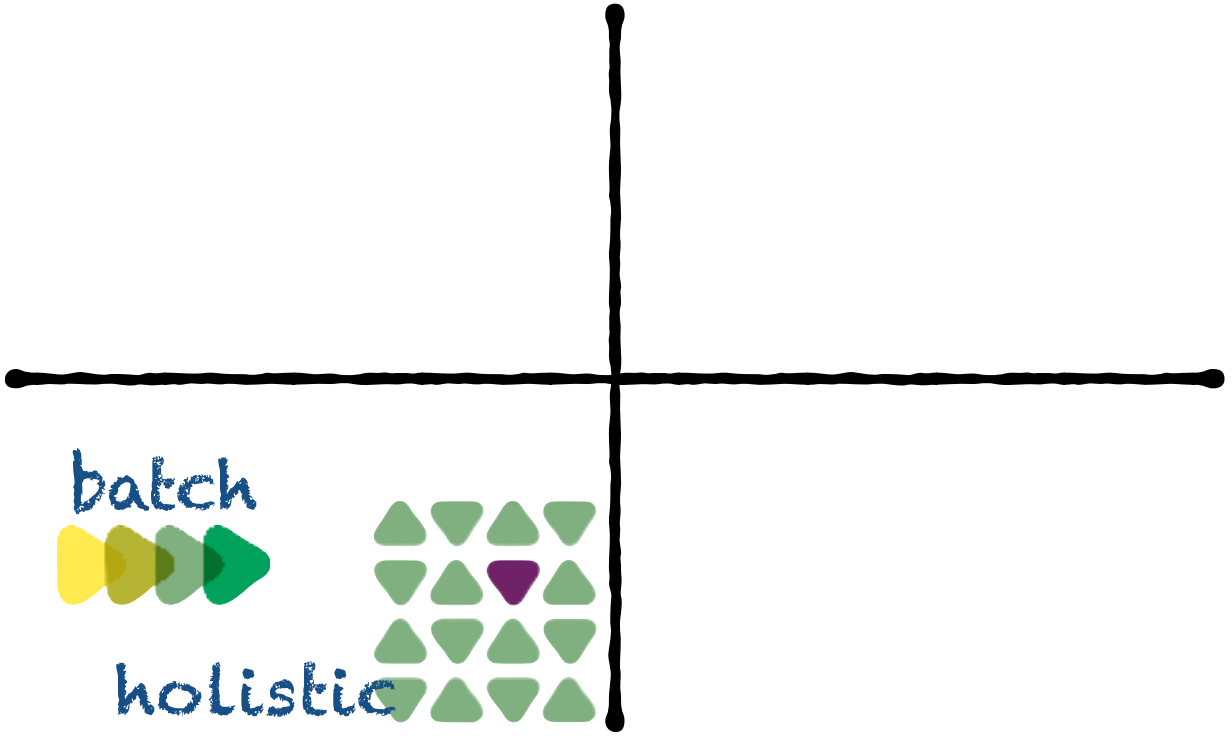
holistic

batch

continuous

Fitness Function

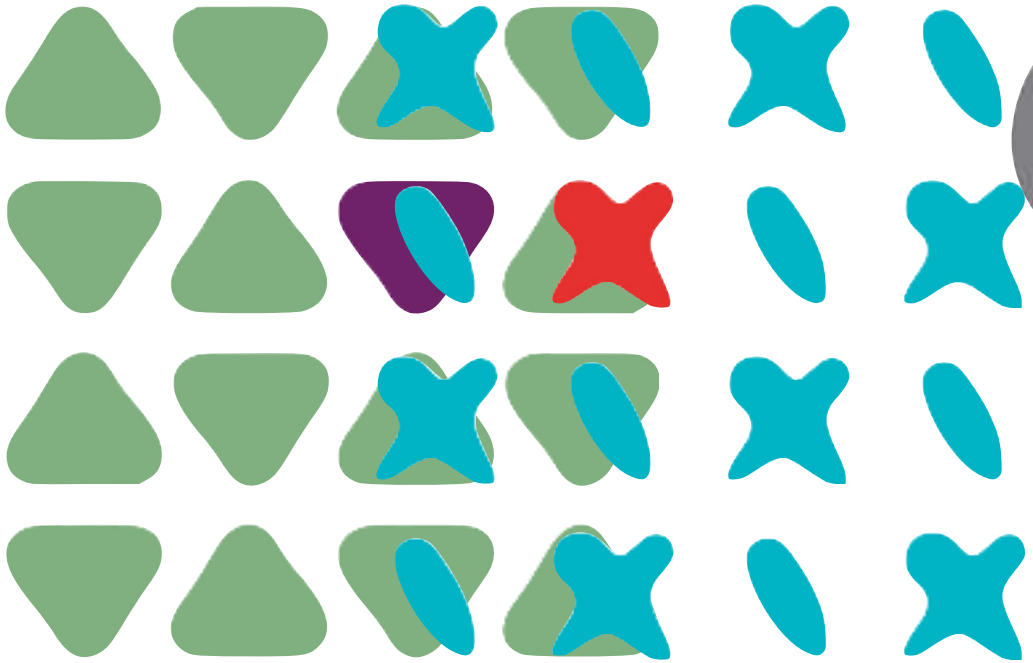
atomic

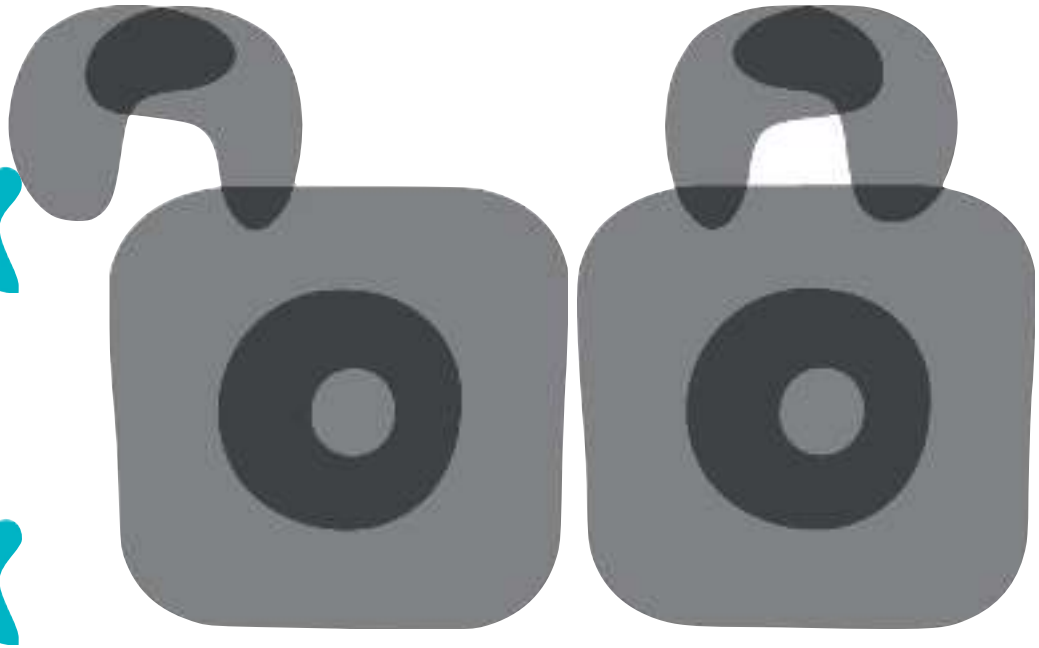


batch

holistic

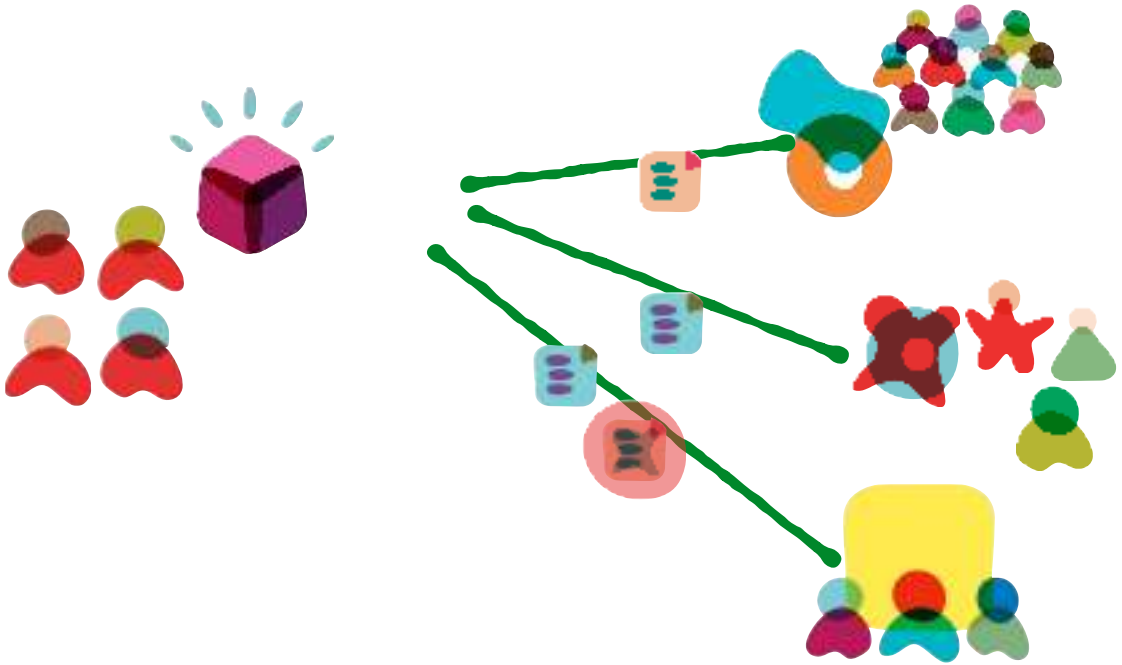
continuous





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

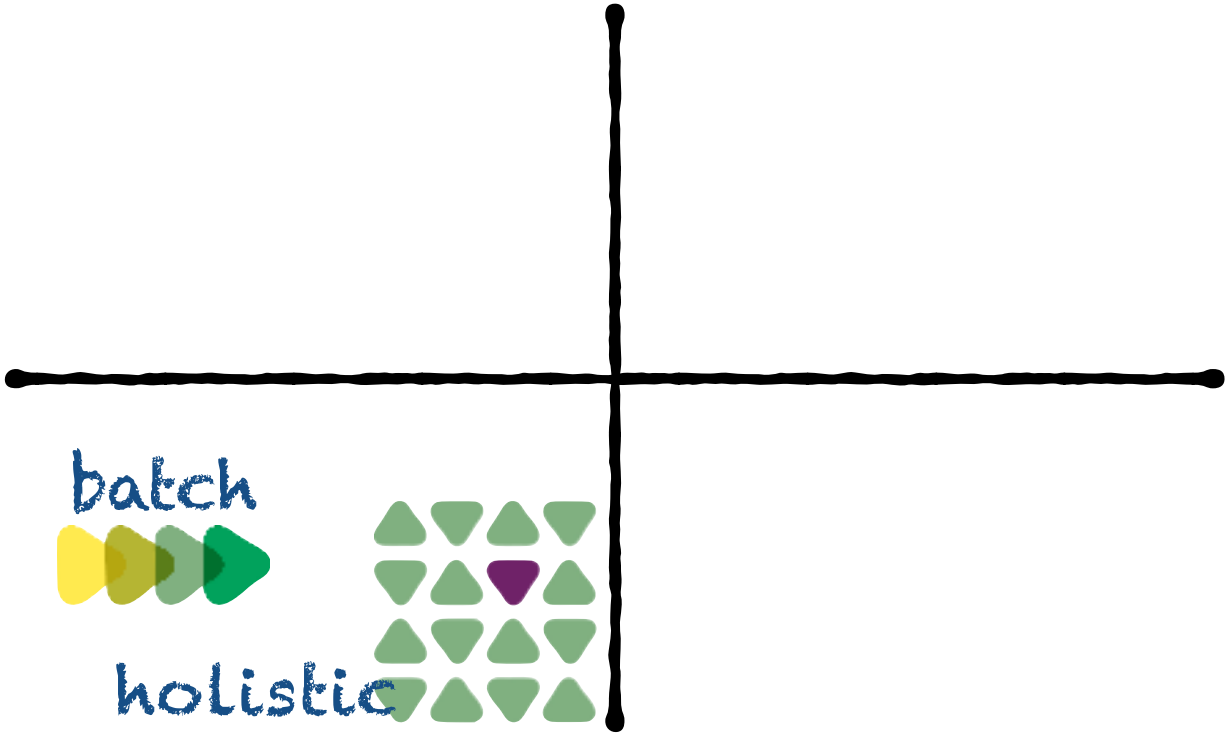
Consumer Driven Contracts



martinfowler.com/articles/consumerDrivenContracts.html

Fitness Function

atomic



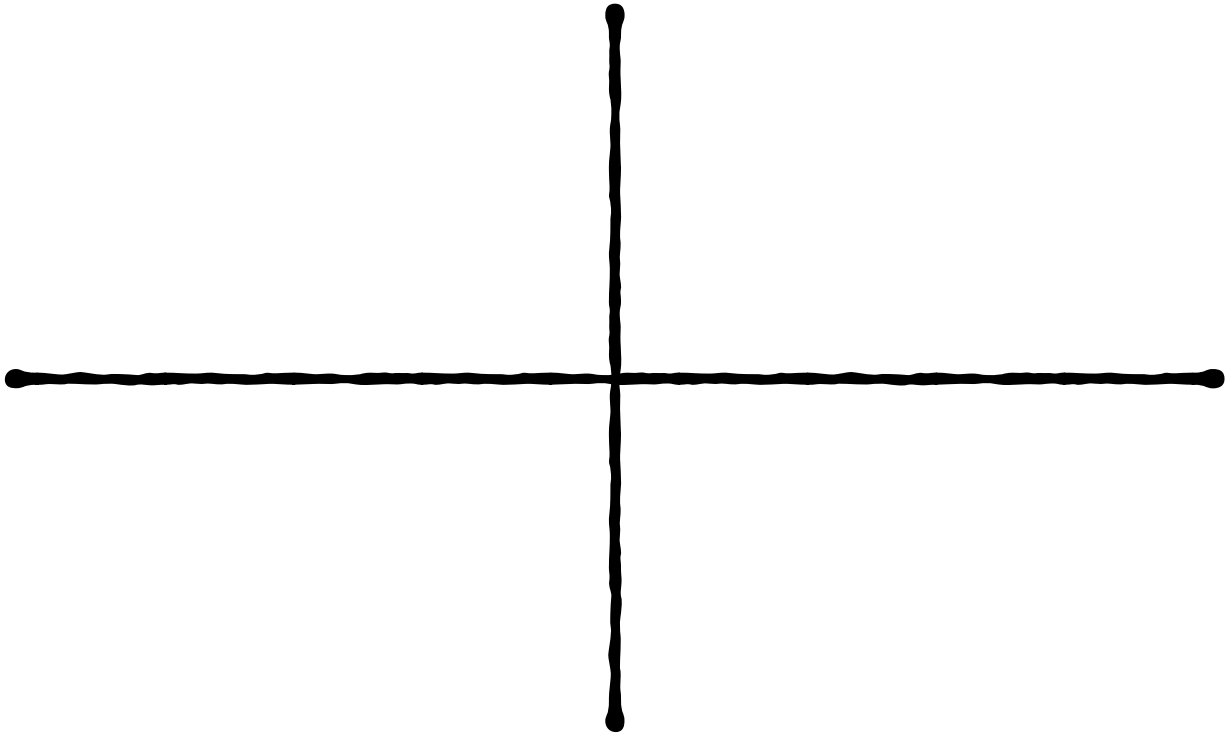
batch

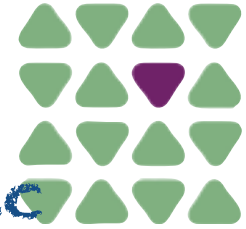
holistic

continuous

Fitness Function

atomic





holistic

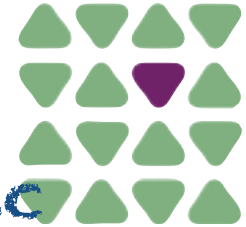
batch

continuous

Fitness Function

atomic

continuous

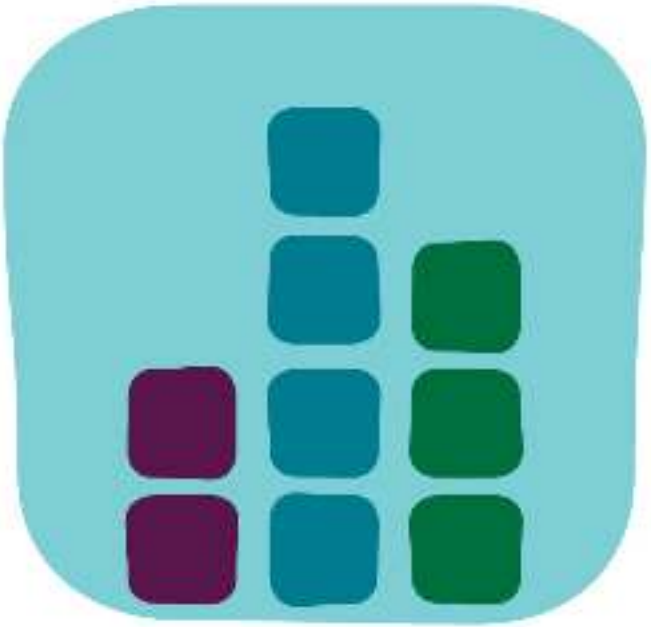


holistic

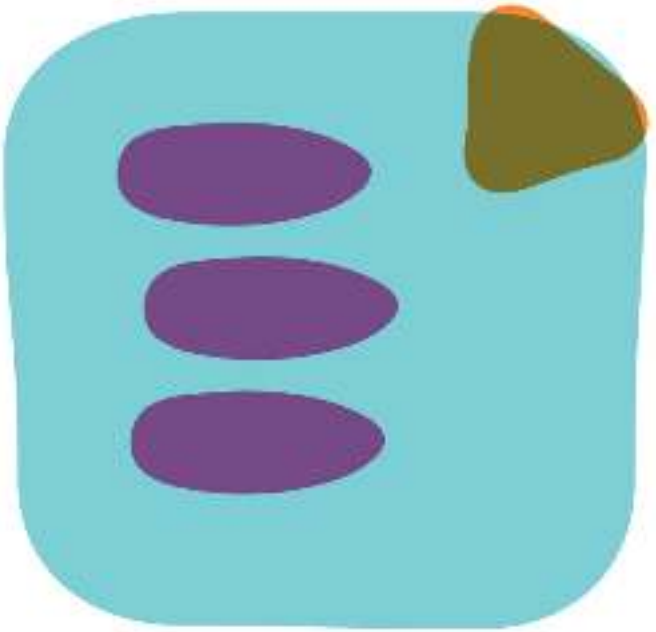
batch

atomic

continuous 



monitoring

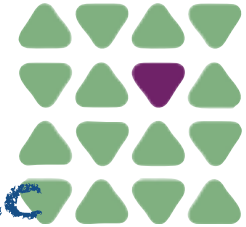


logging

Fitness Function

atomic

continuous

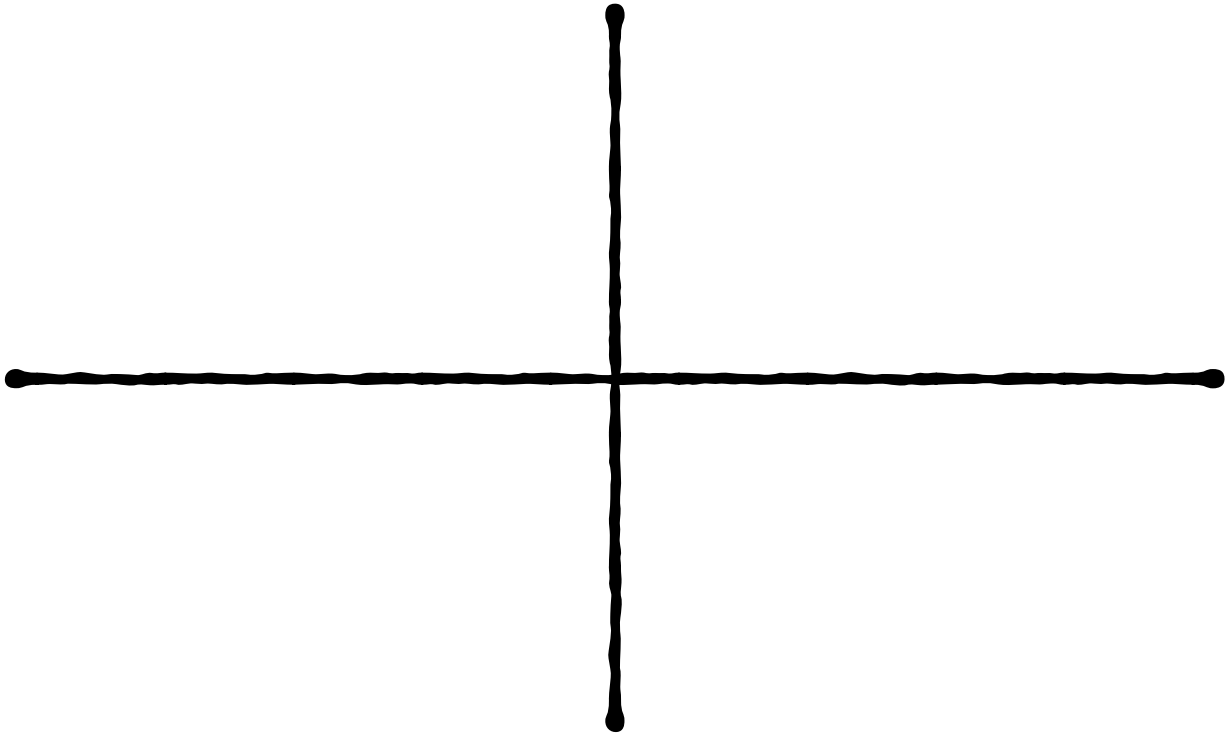


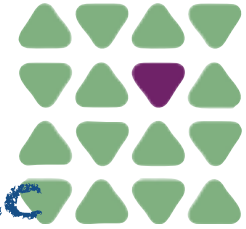
holistic

batch

Fitness Function

atomic





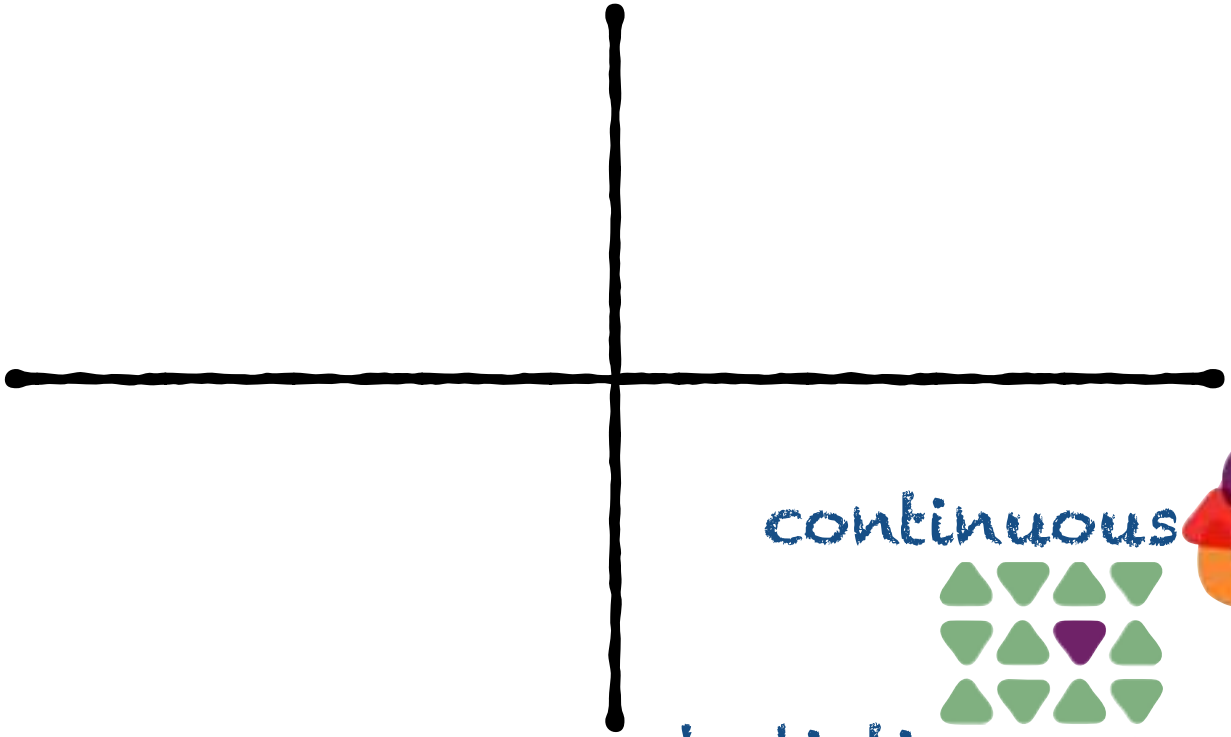
holistic

batch

continuous

Fitness Function

atomic



continuous

holistic

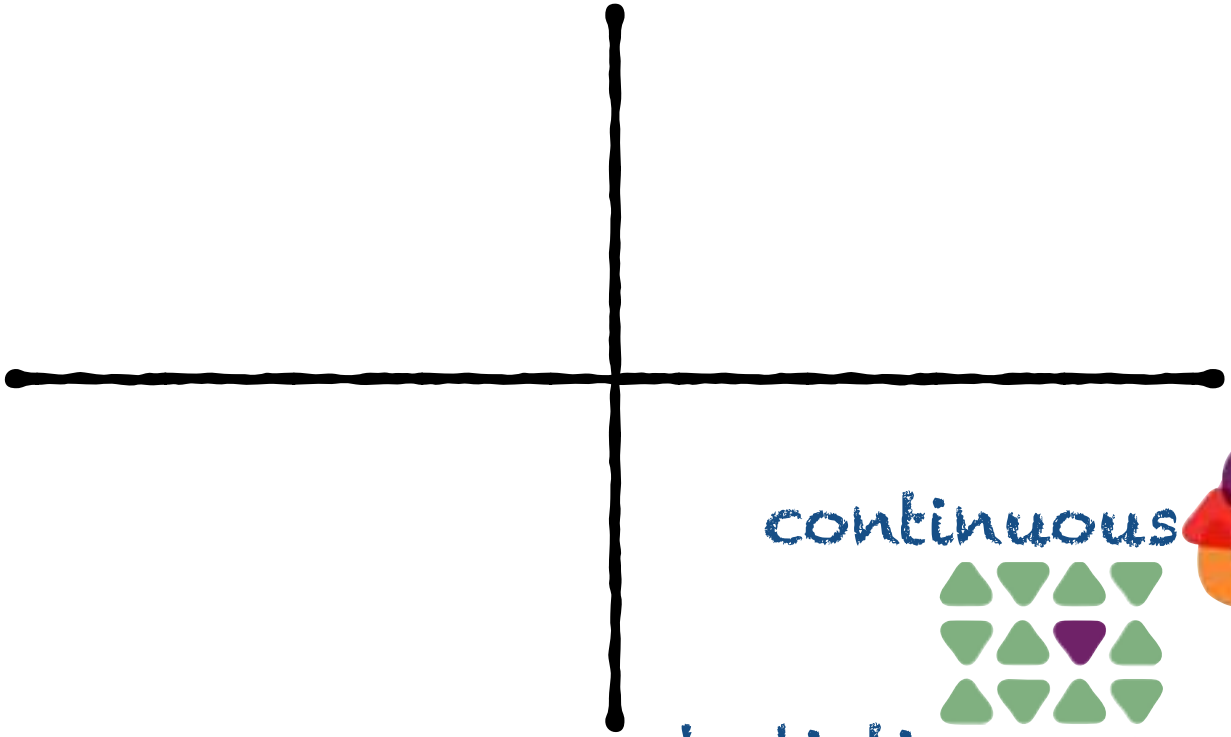
batch

holistic

resilience-ility?

Fitness Function

atomic



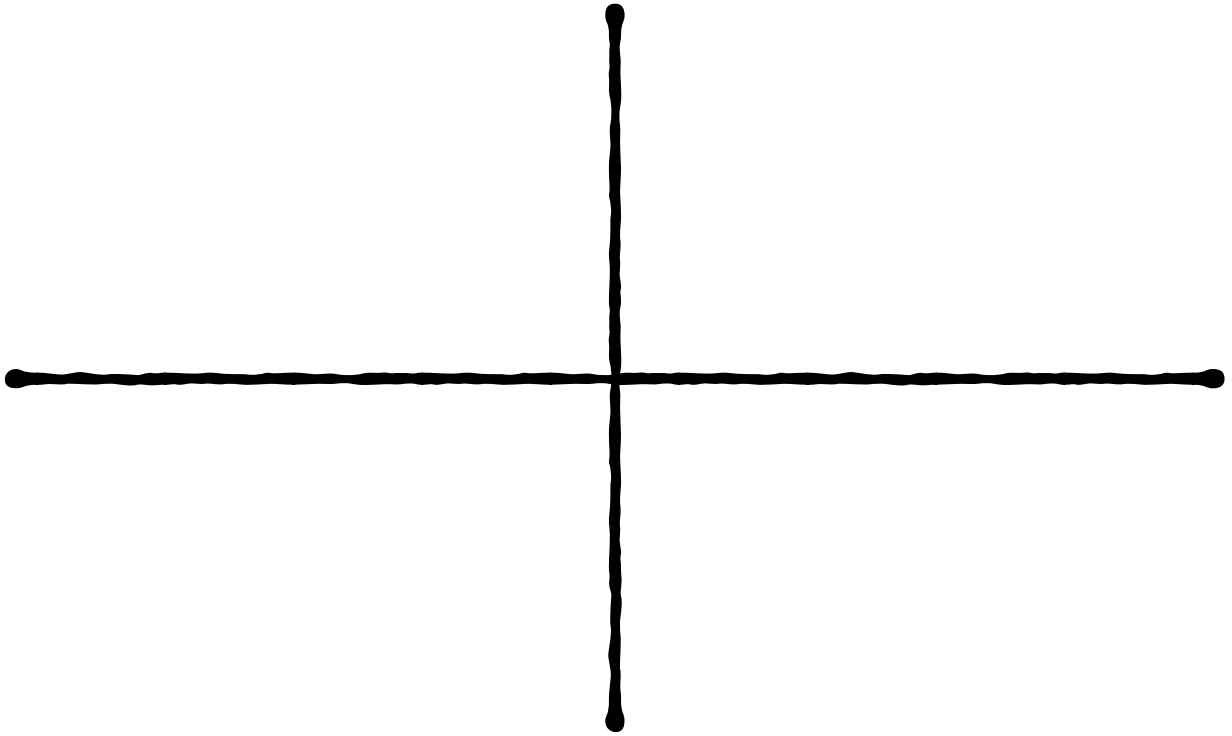
continuous

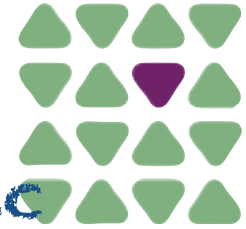
holistic

batch

Fitness Function

atomic



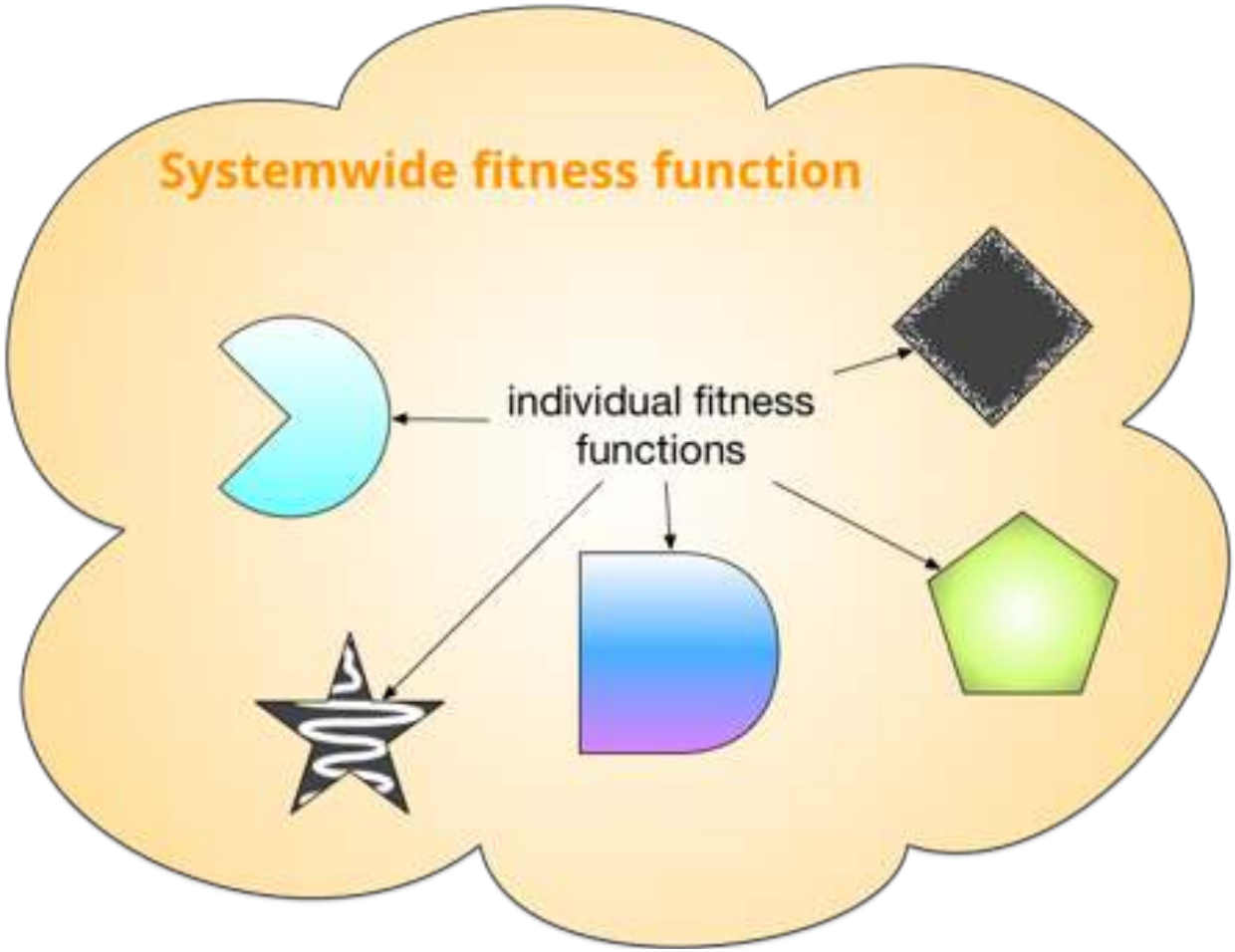


holistic

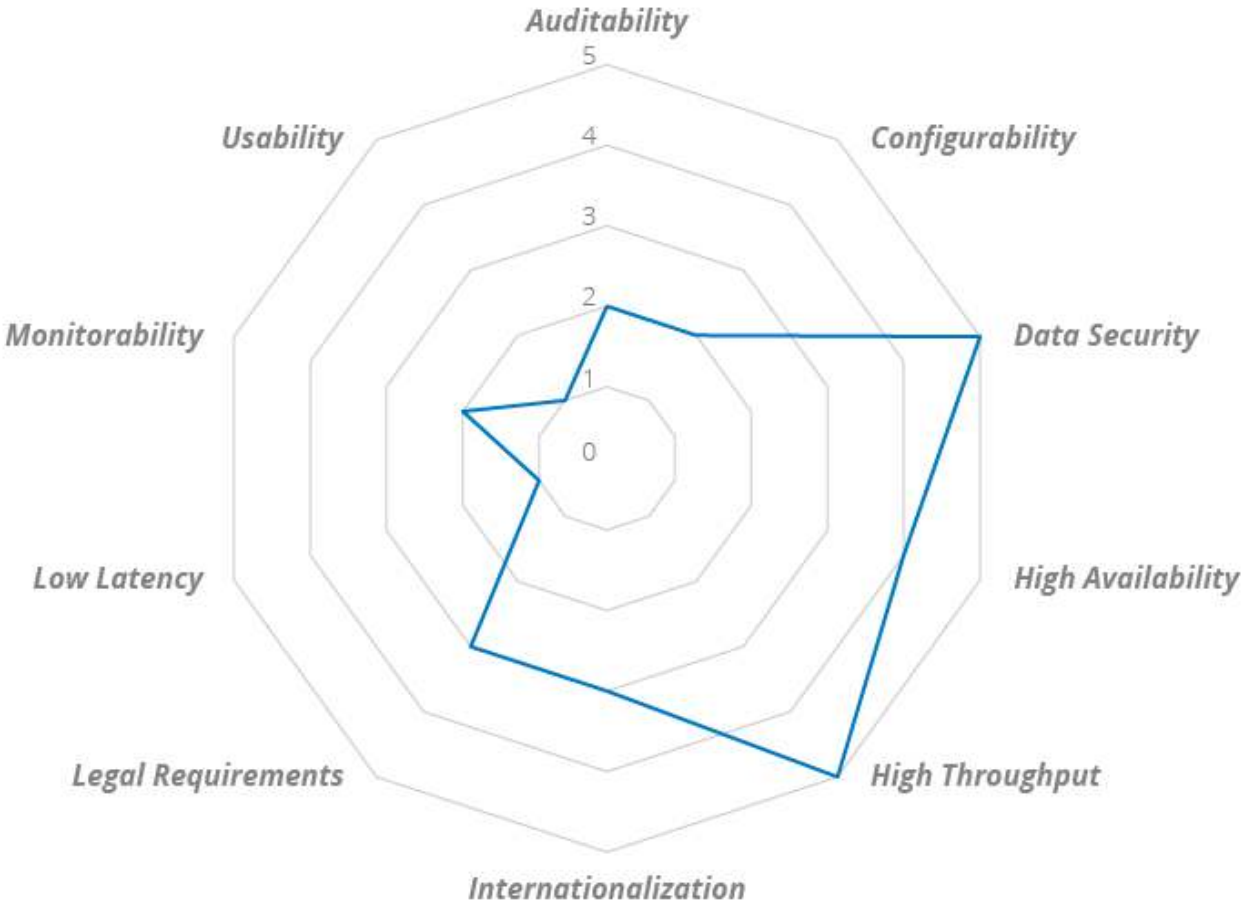
batch

continuous

System-wide Fitness Function

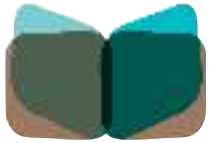


Fitness Function Fit



Guided Evolution

Agenda

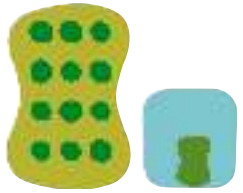


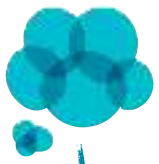
definition

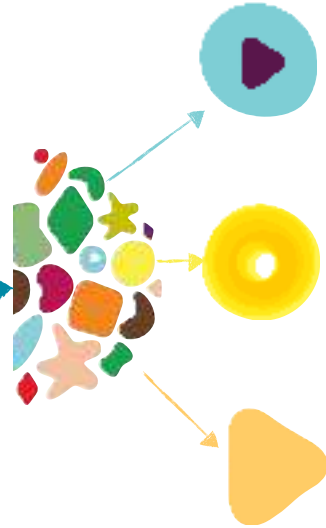
incremental change

fitness functions

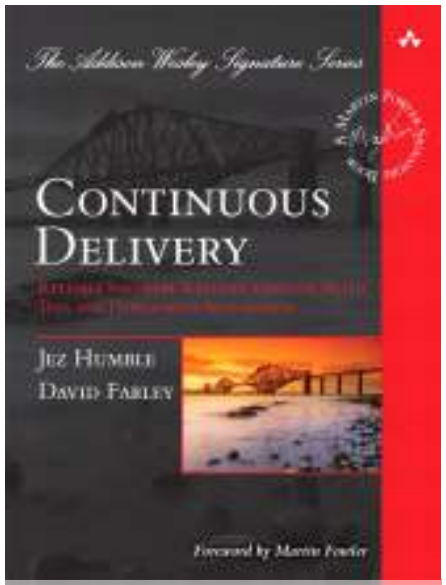
appropriate coupling

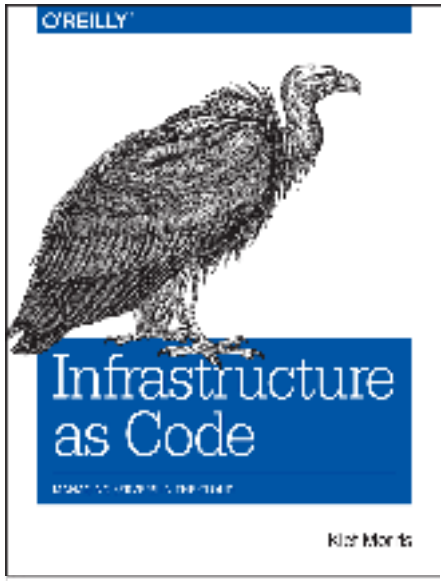


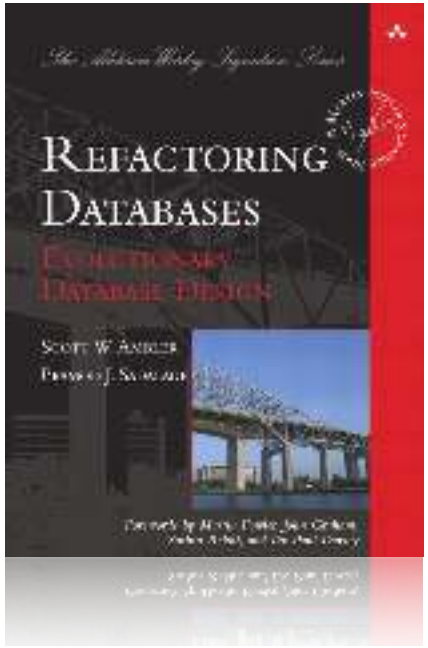




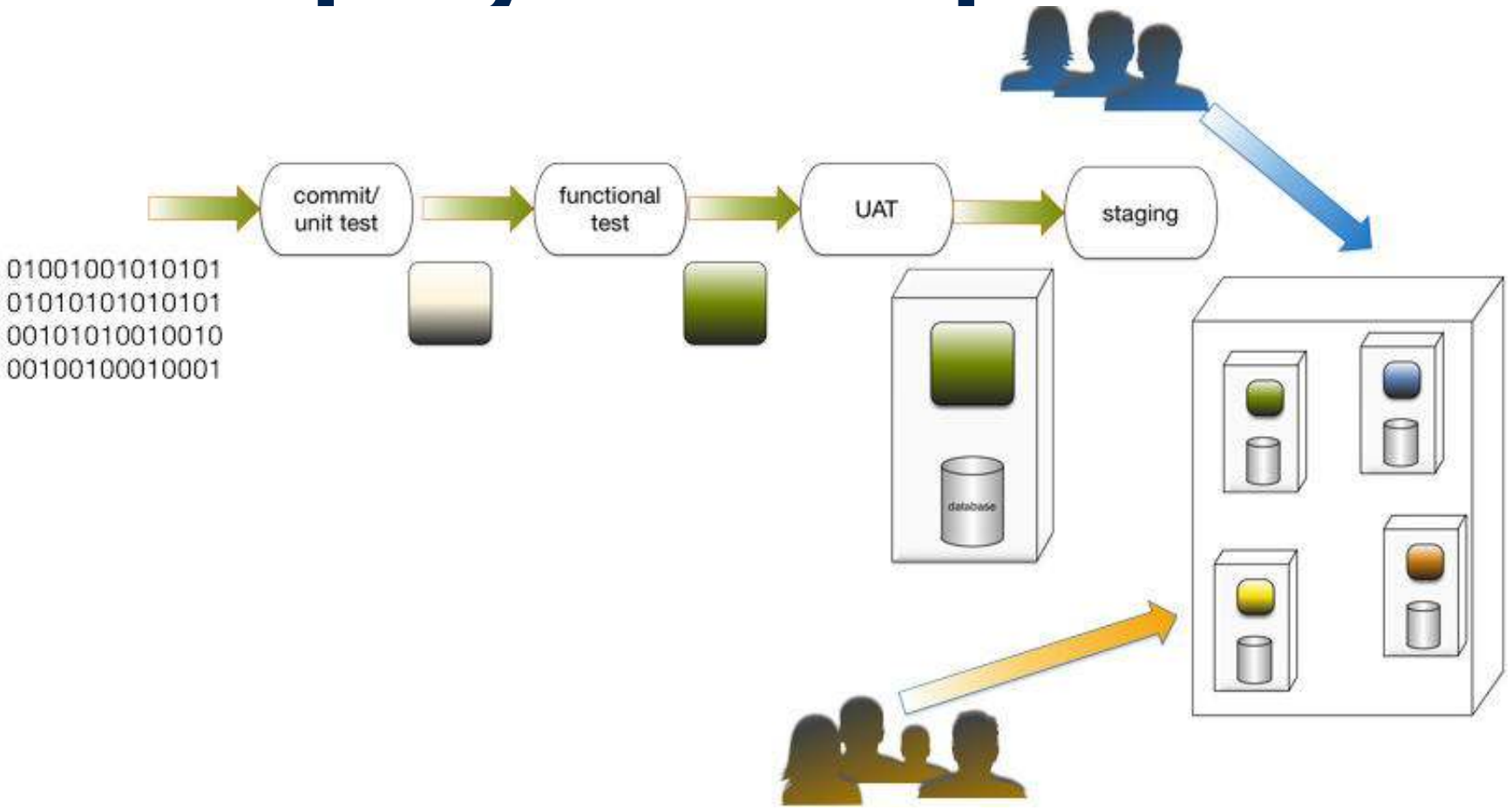
Prerequisites



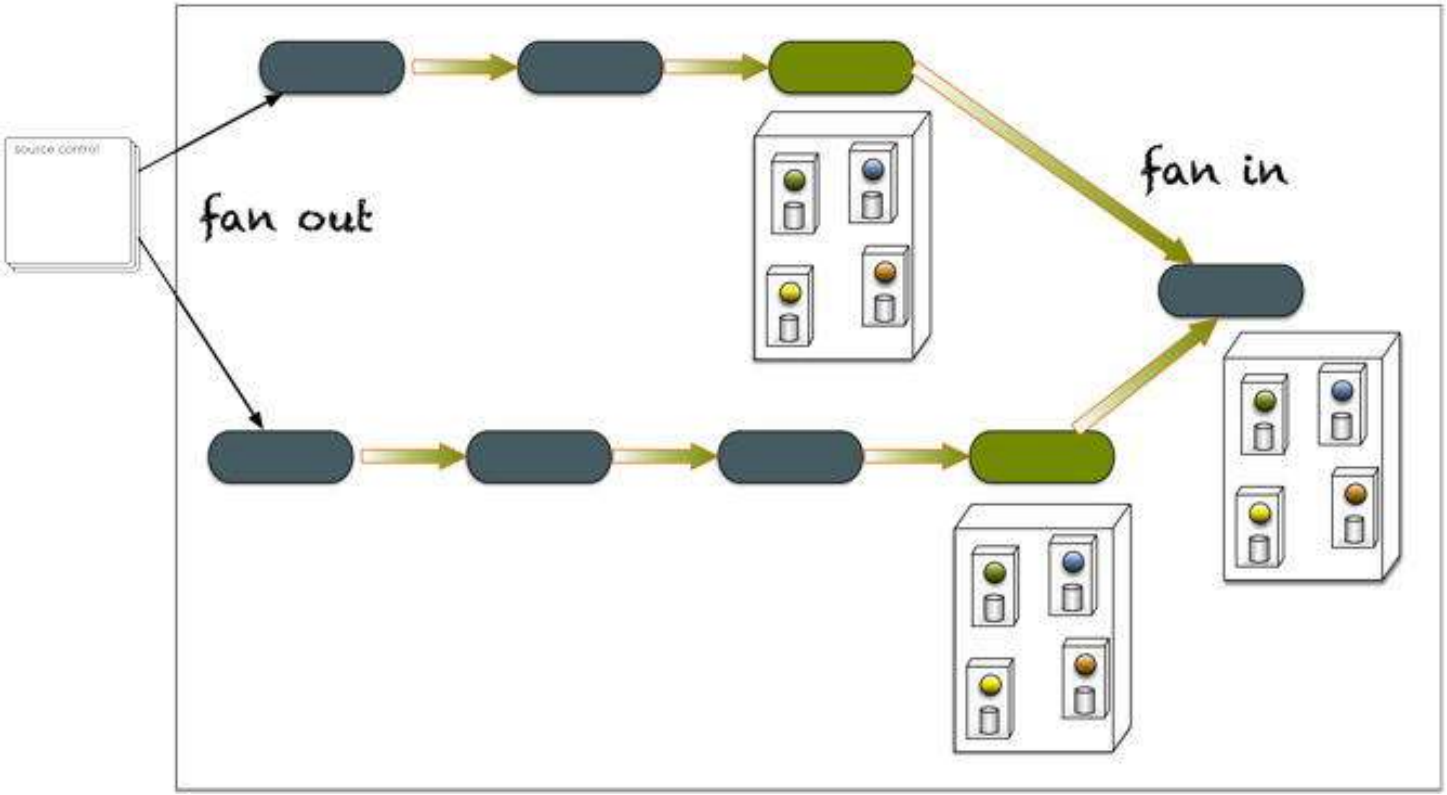




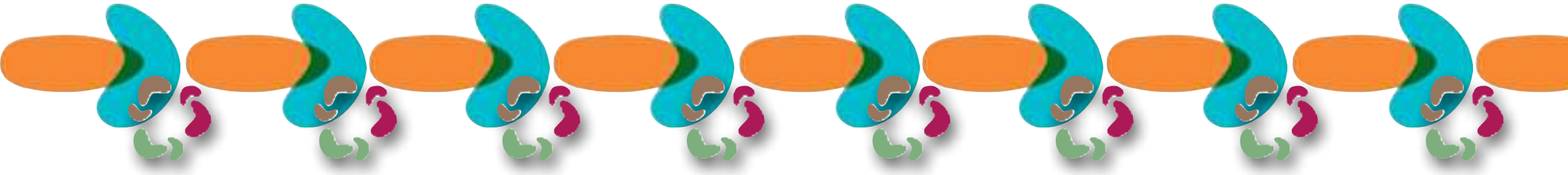
Deployment Pipeline



Deployment Pipeline



Incremental Change

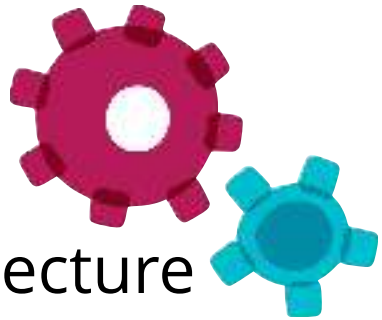


$$V \propto C$$

where

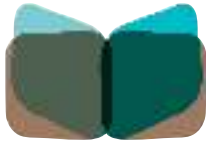
c = cycle time

v = maximum speed of new generations



Engine of evolutionary architecture

Agenda

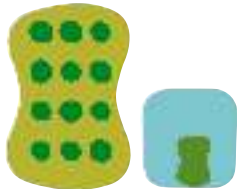


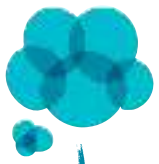
definition

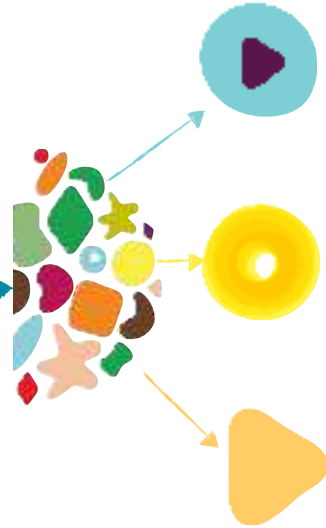
incremental change

fitness functions

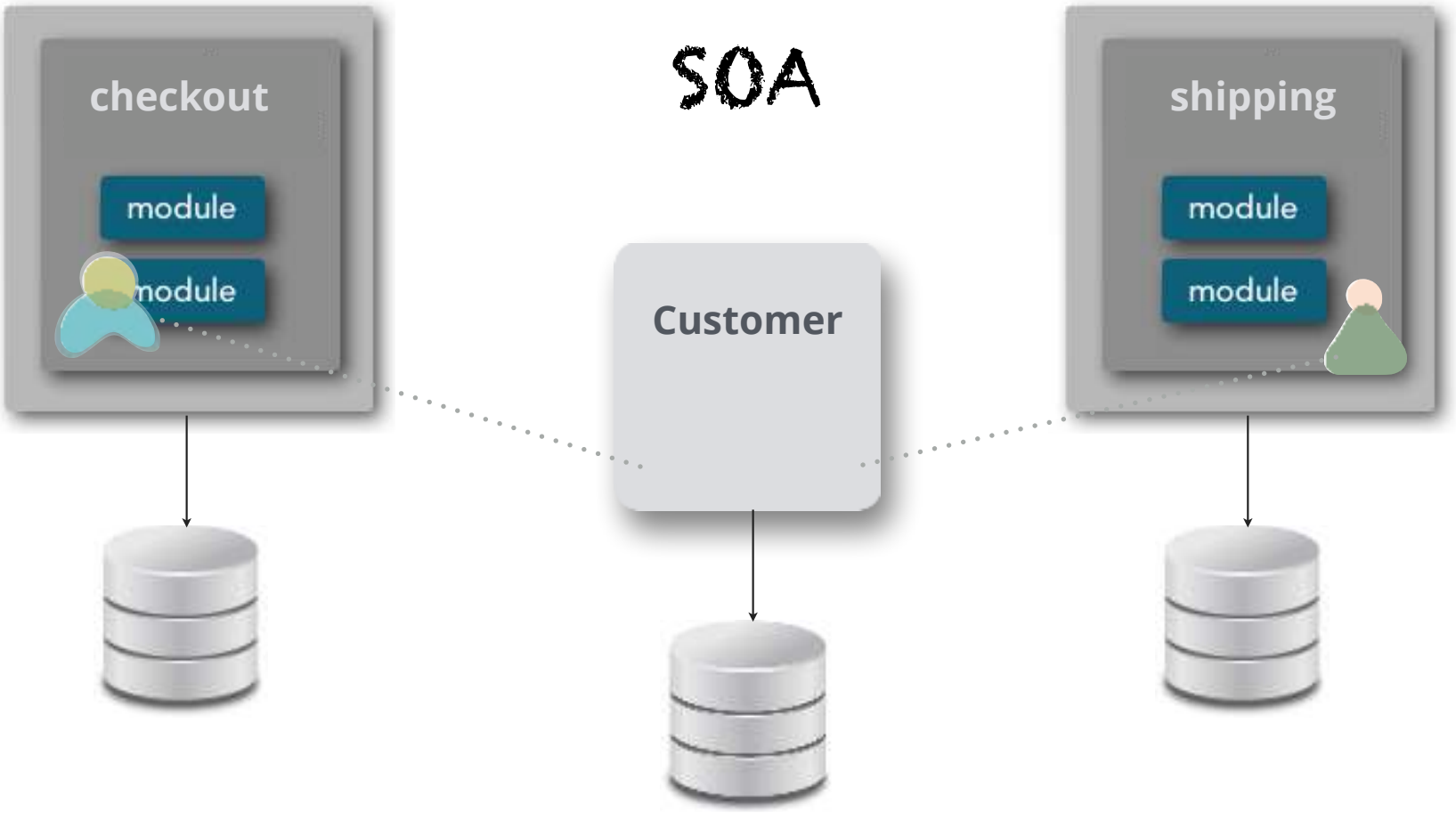
appropriate coupling



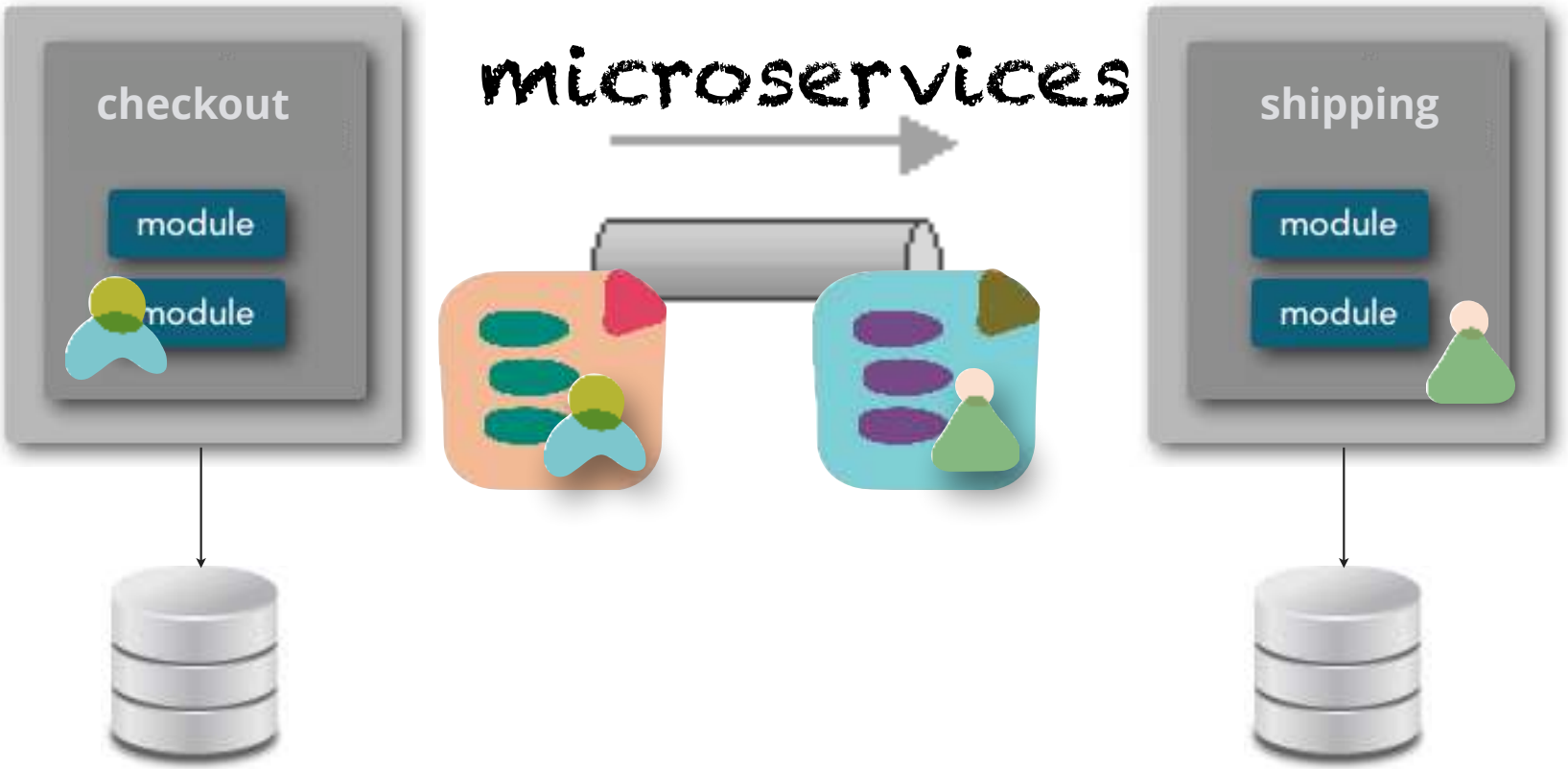


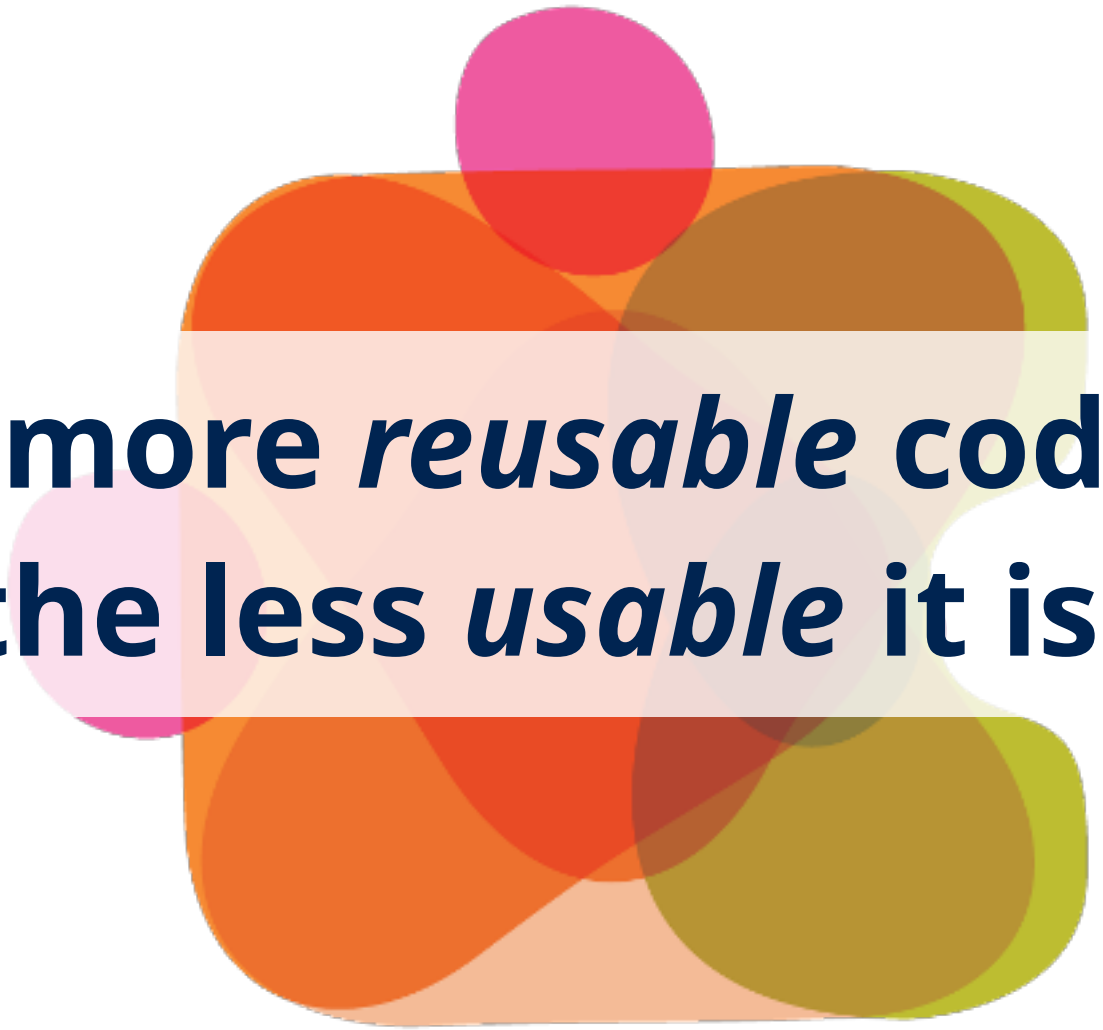


Code Reuse (Over Time)



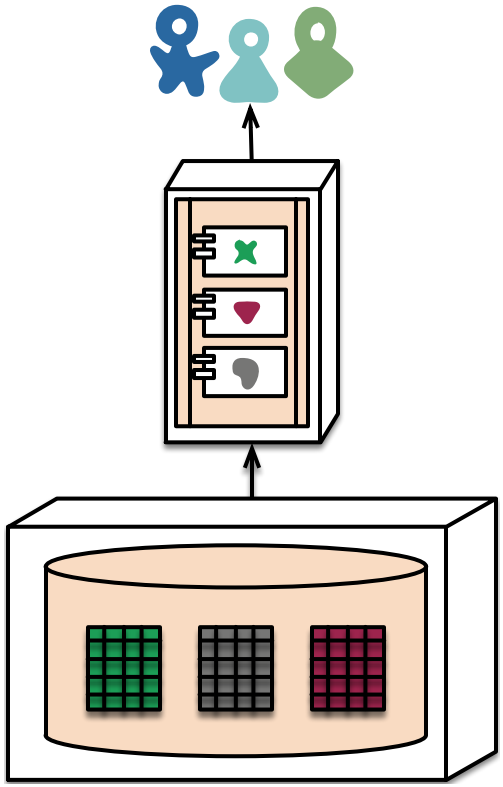
Code Reuse (Over Time)



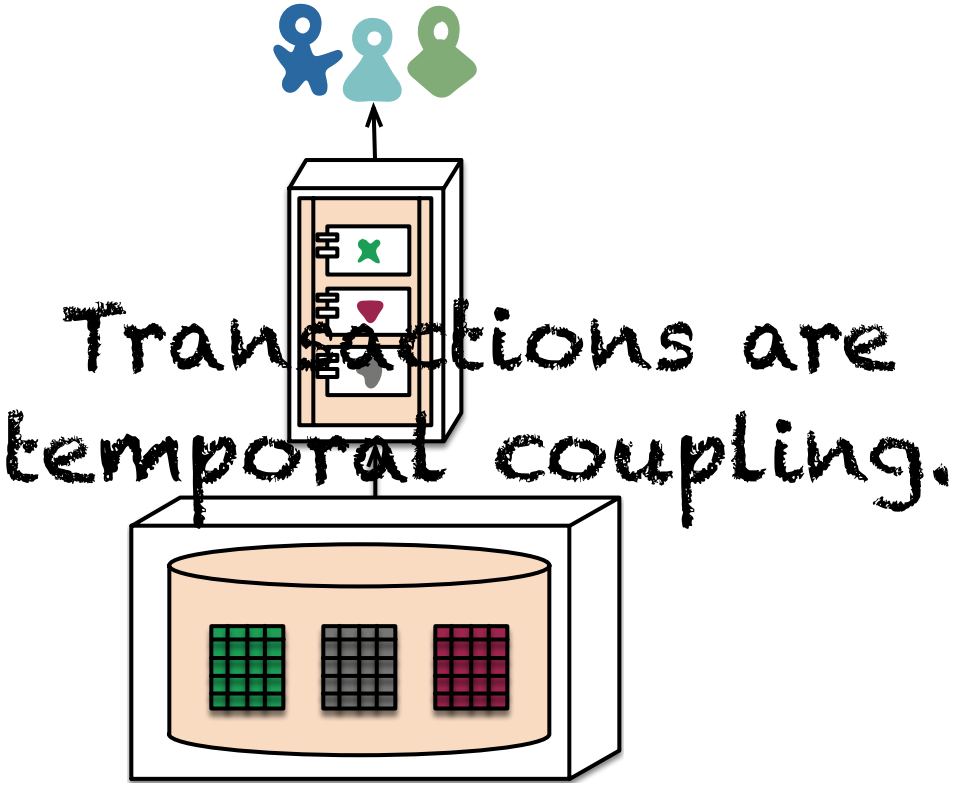


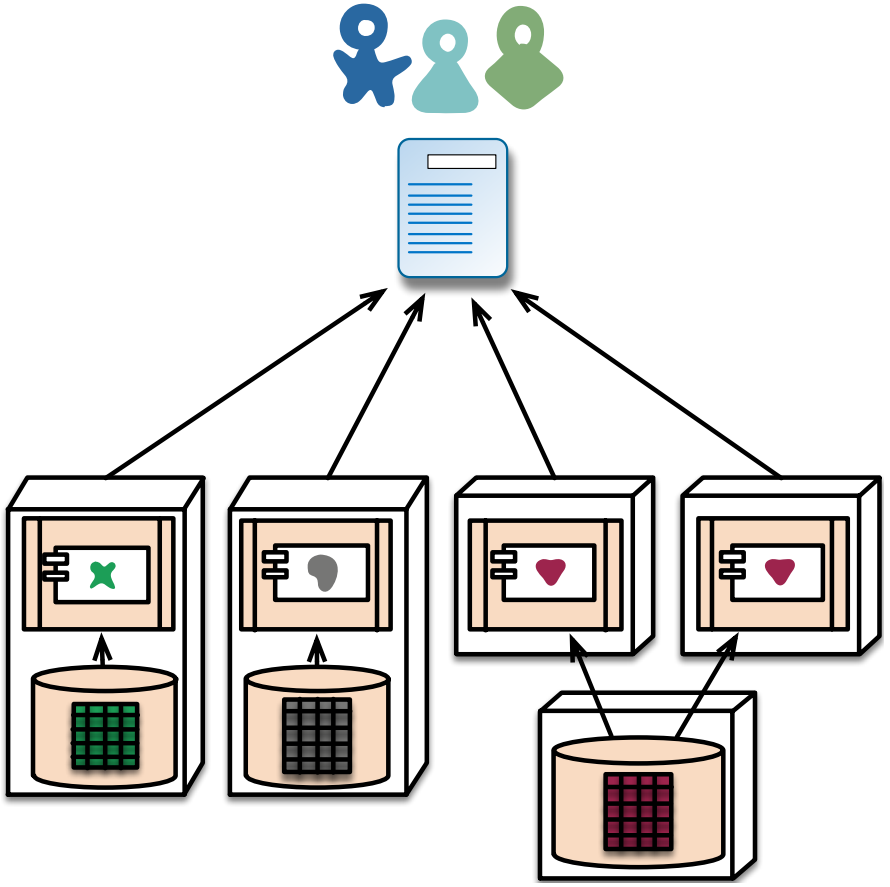
The more *reusable* code is, the less *usable* it is.

Decentralized Data Management



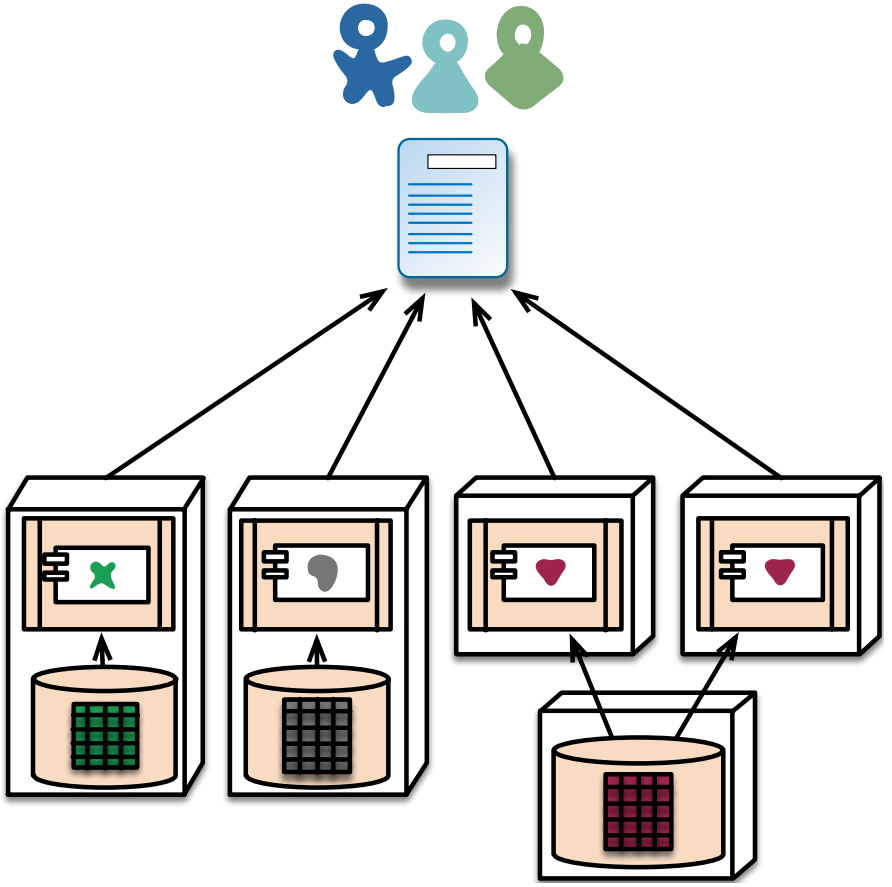
Decentralized Data Management



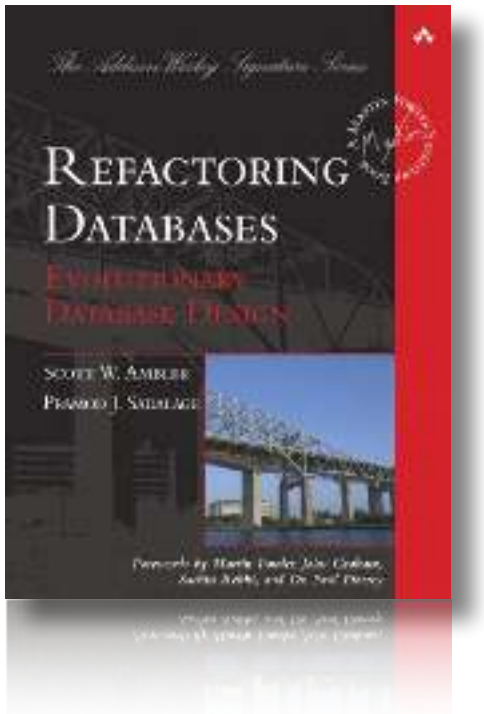


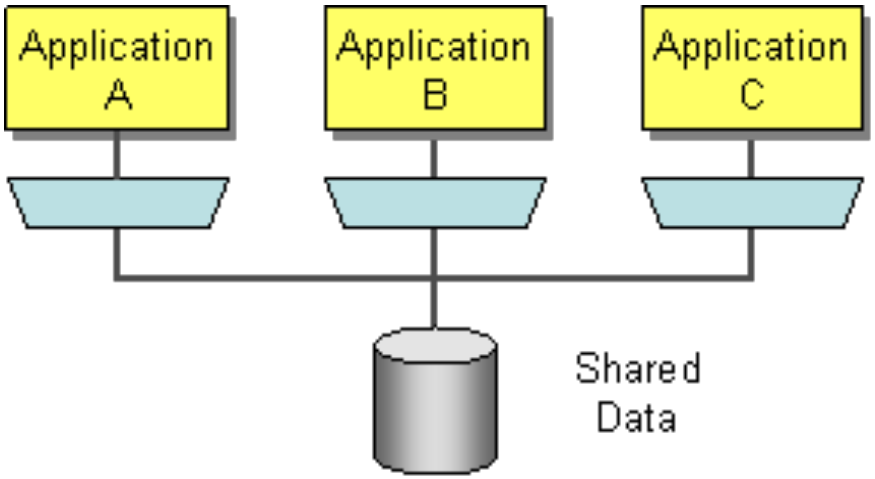
Decentralized Data Management

Limit transactional contexts.



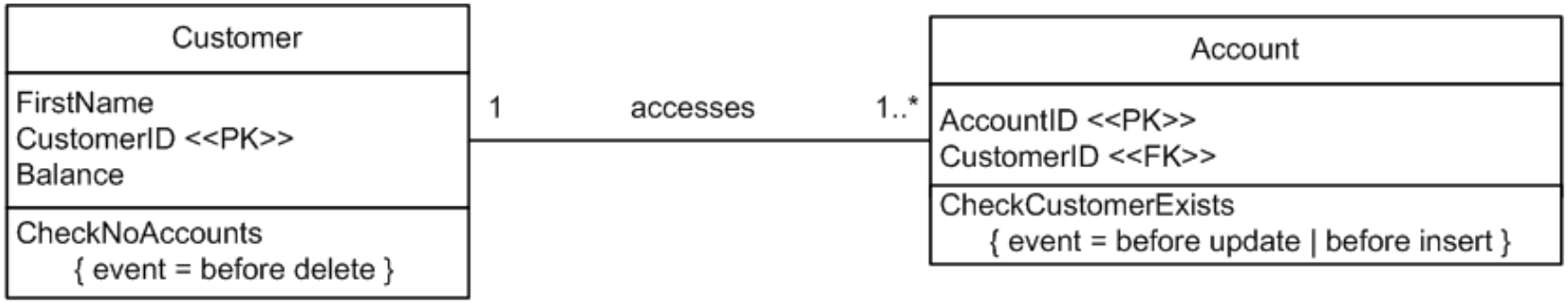
Evolutionary Database Design





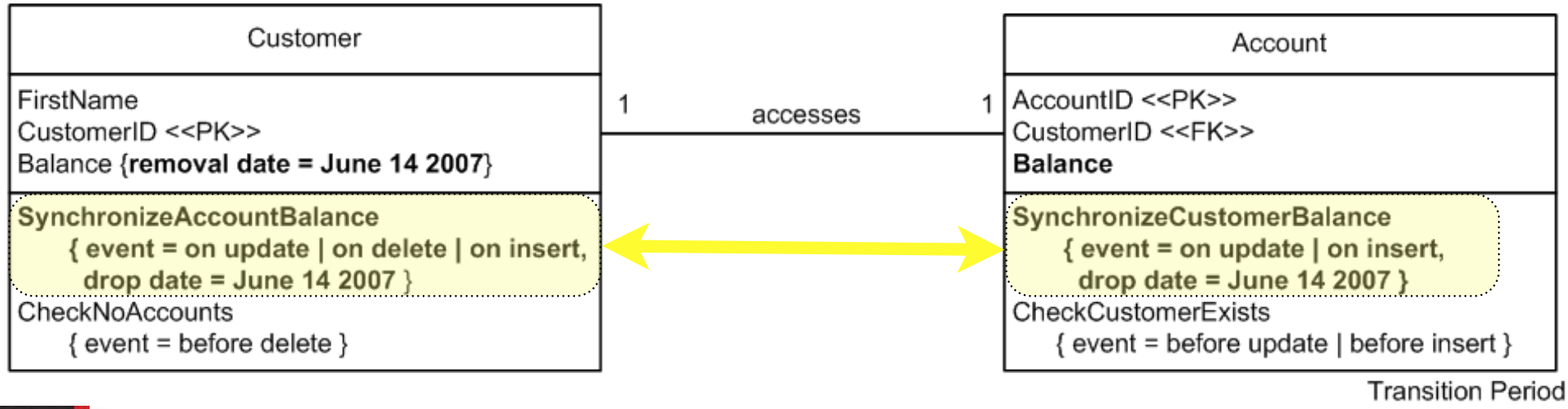
http://databaserefactoring.com/

Evolving Columns

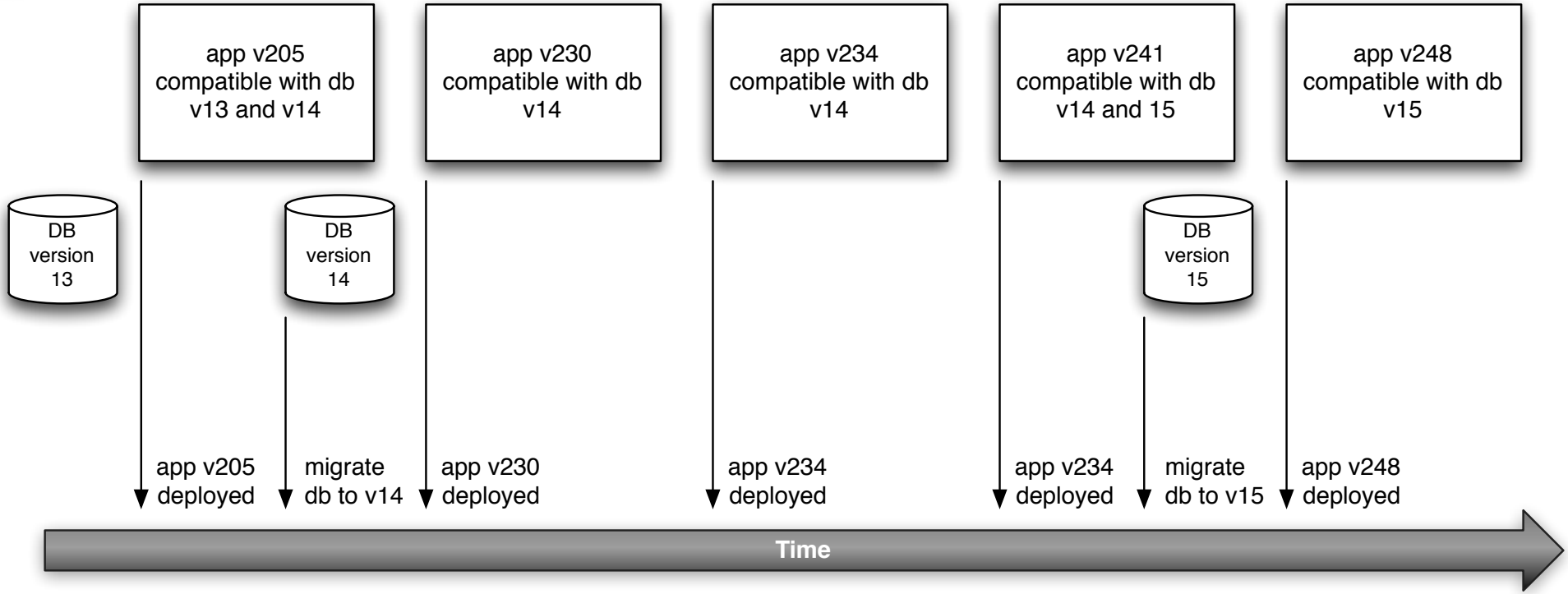


Original Schema

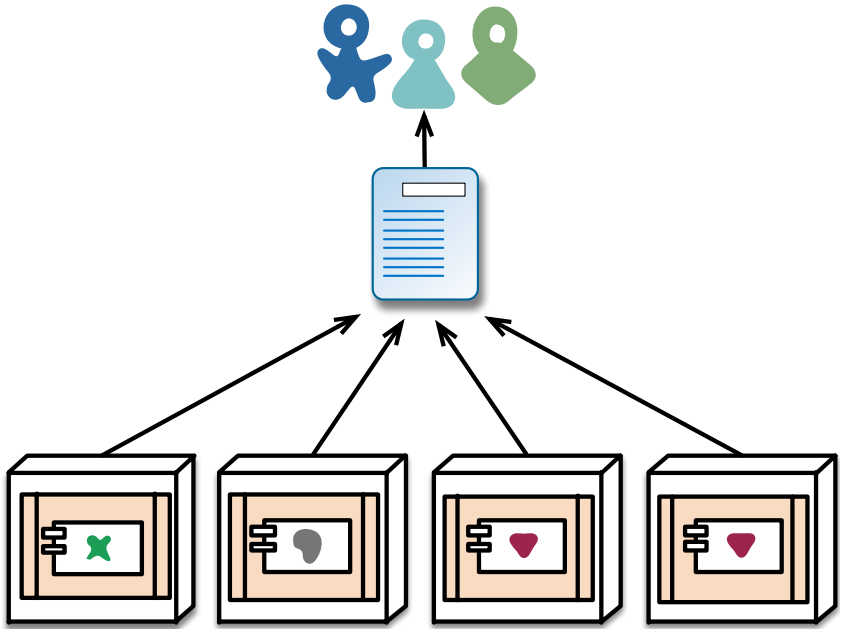
Transition


Customer
FirstName
CustomerID <<PK>>
Balance {removal date = June 14 2007}
SynchronizeAccountBalance
{ event = on update | on delete | on insert,
drop date = June 14 2007 }
CheckNoAccounts
{ event = before delete }
1
accesses
1
Account
AccountID <<PK>>
CustomerID <<FK>>
Balance
SynchronizeCustomerBalance
{ event = on update | on insert,
drop date = June 14 2007 }
CheckCustomerExists
{ event = before update | before insert }
Transition Period

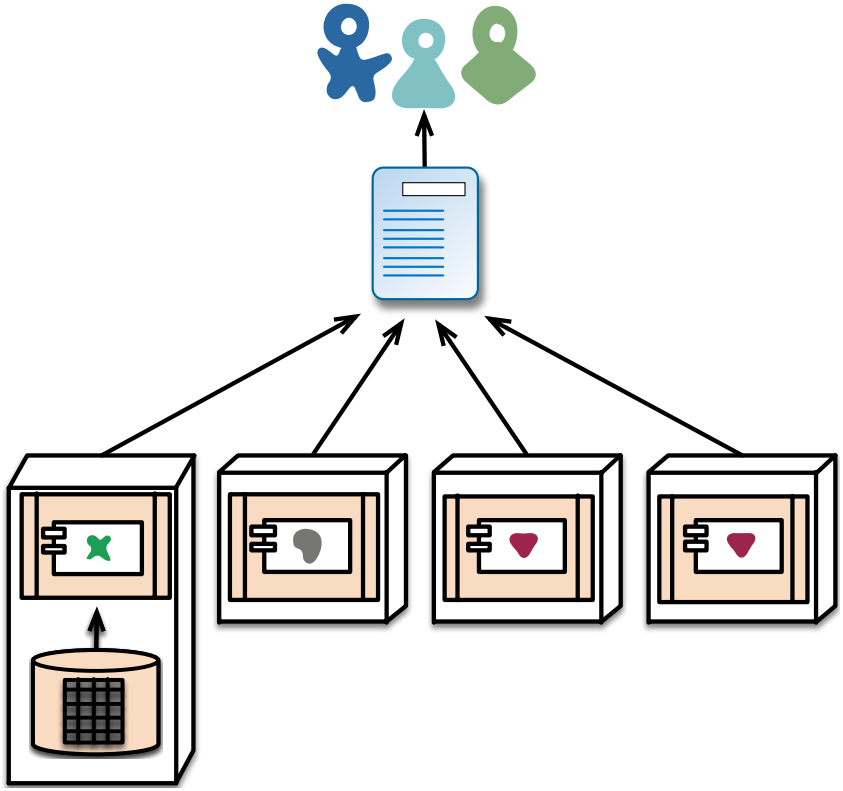
Expand/Contract Pattern



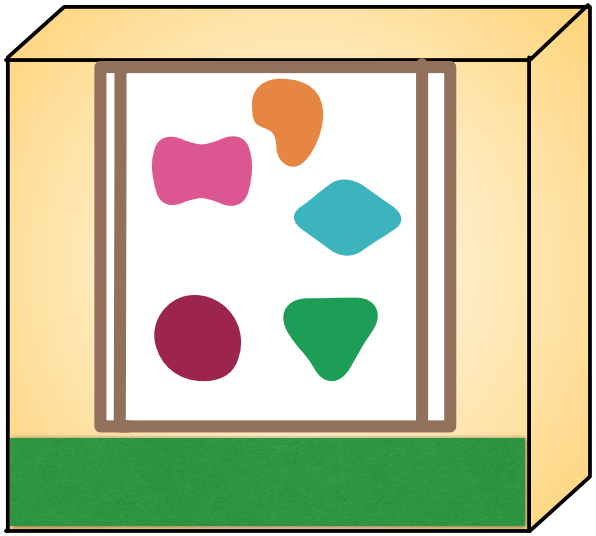
Decentralized Governance

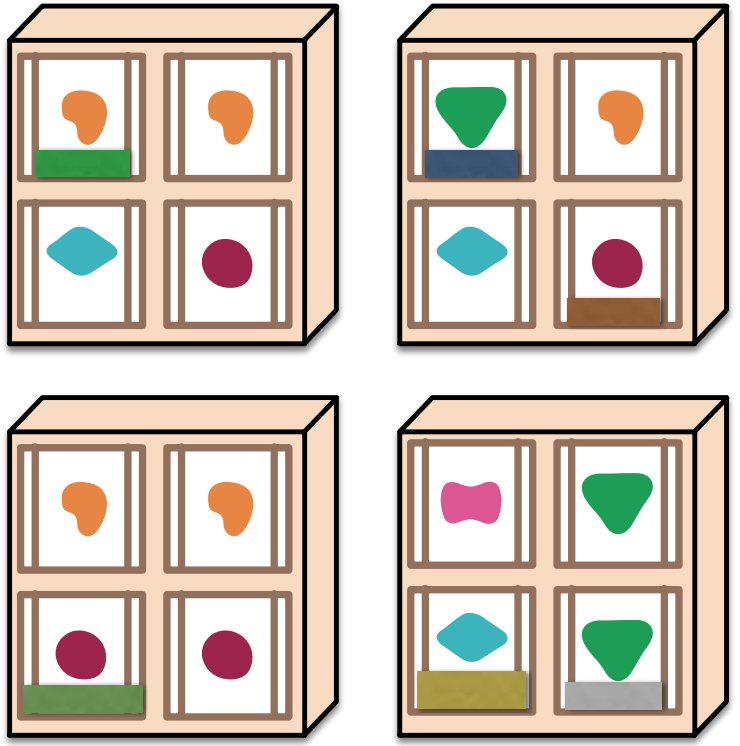


Decentralized Governance



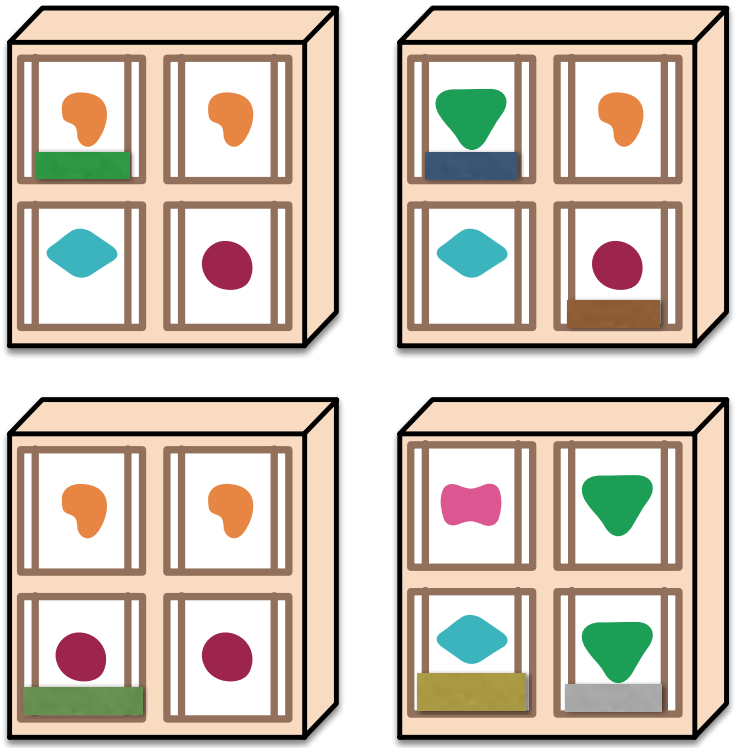
Decentralized Governance



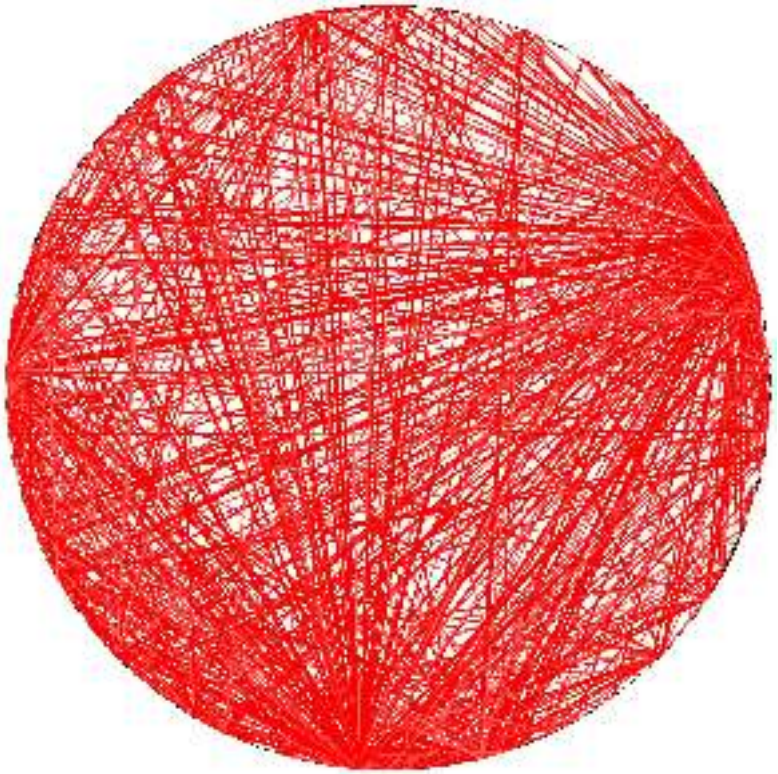


“Goldilocks” Governance

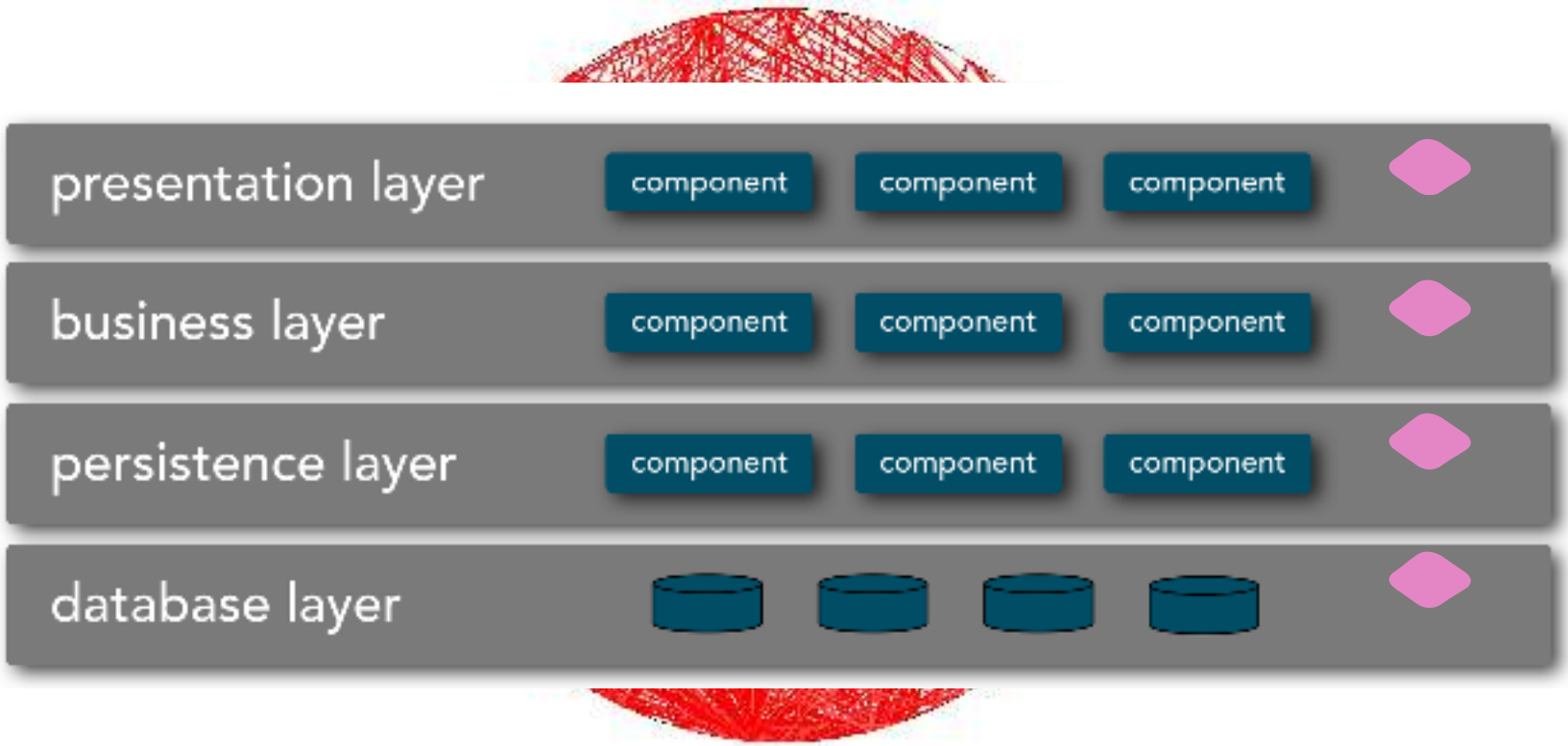
Choose technology stacks appropriate to problem scale.



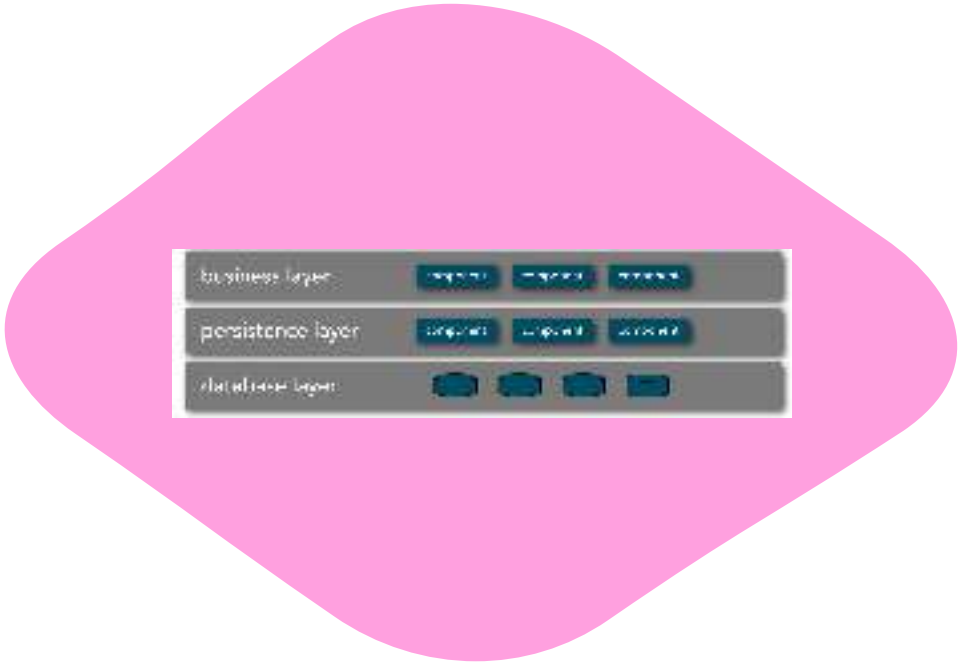
Shift to Domain-centric Architectures



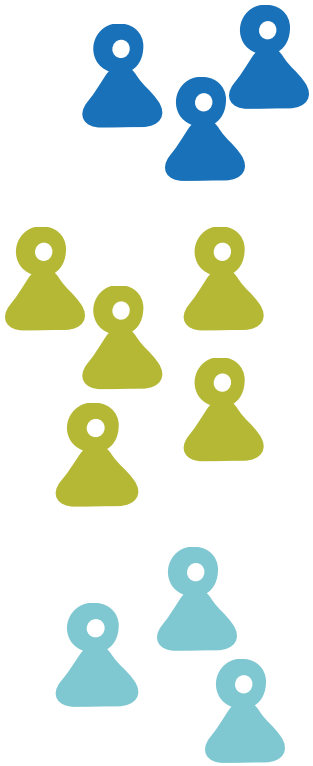
Shift to Domain-centric Architectures



Shift to Domain-centric Architectures



Incidentally Coupled Teams



user interface

server-side

DBA

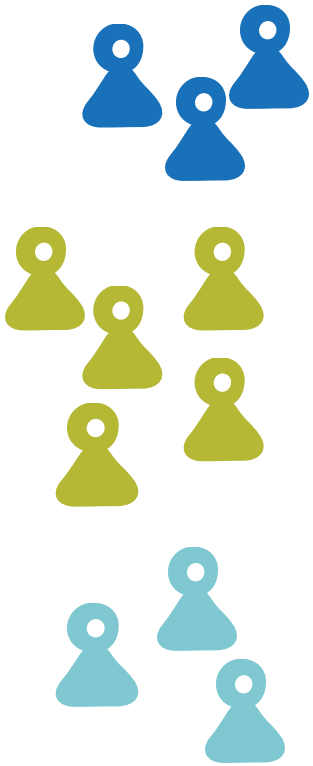
Conway's Law

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

Melvin Conway, 1968

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

Incidentally Coupled Teams



user interface

server-side

DBA

Autonomous Teams

Orders

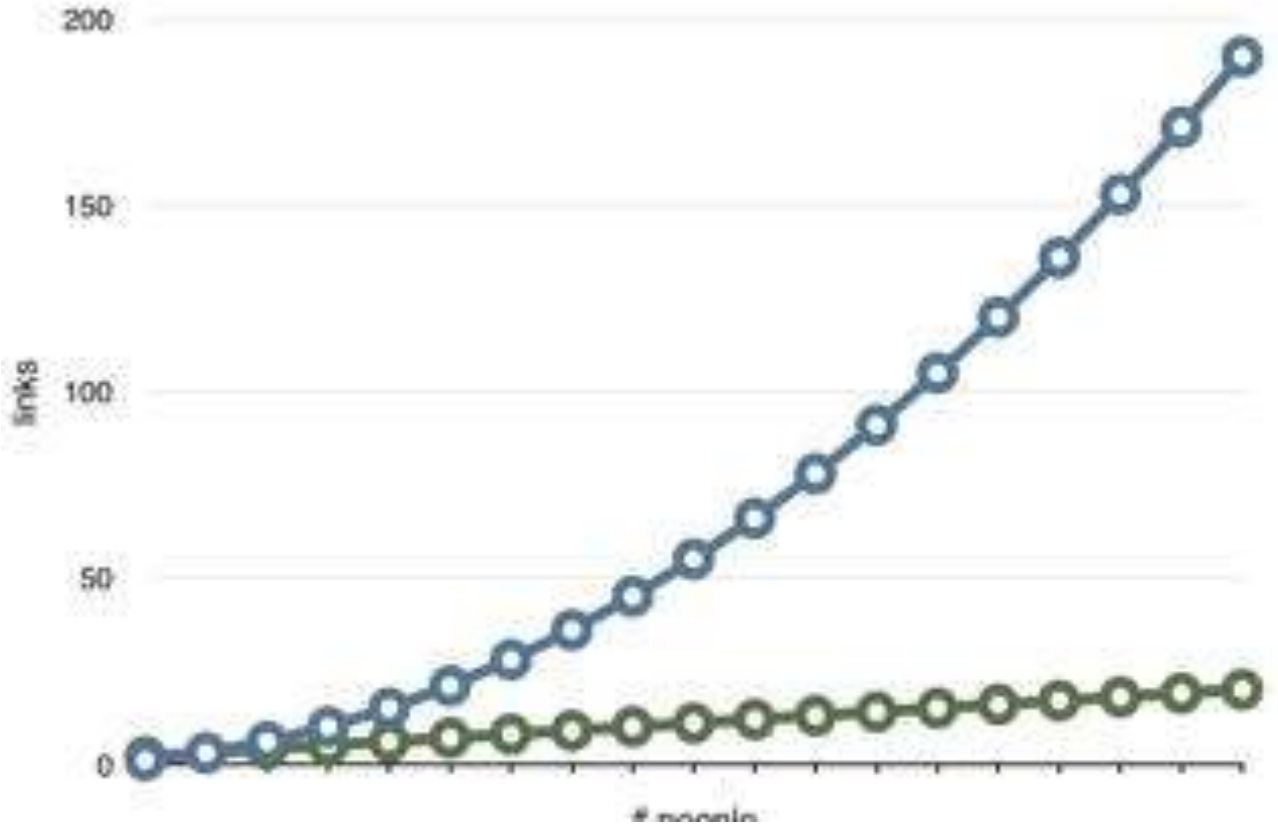
Shipping

Inverse Conway Maneuver

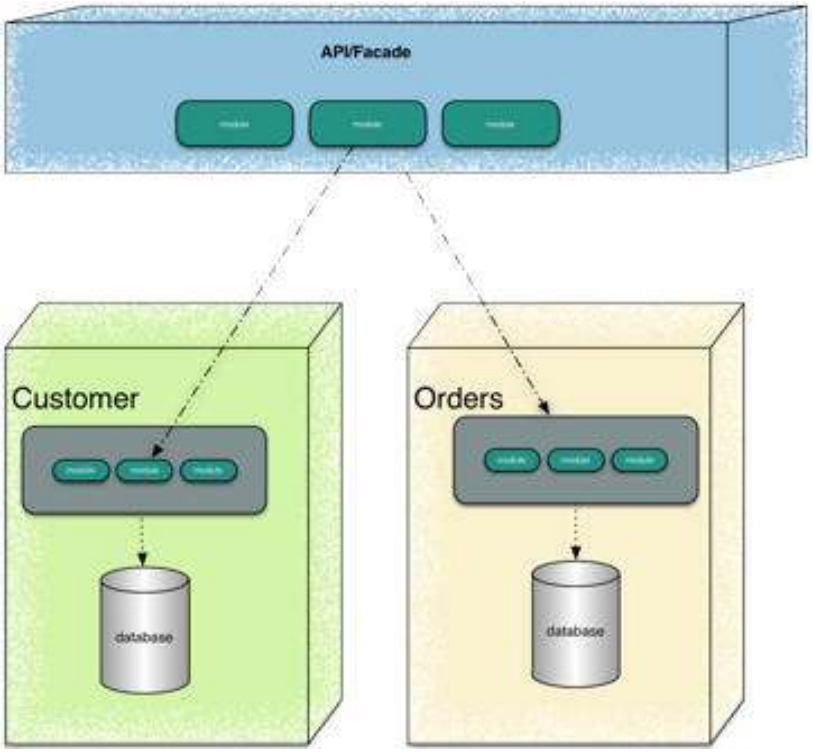
Catalog

Low Efferent Coupling between Teams

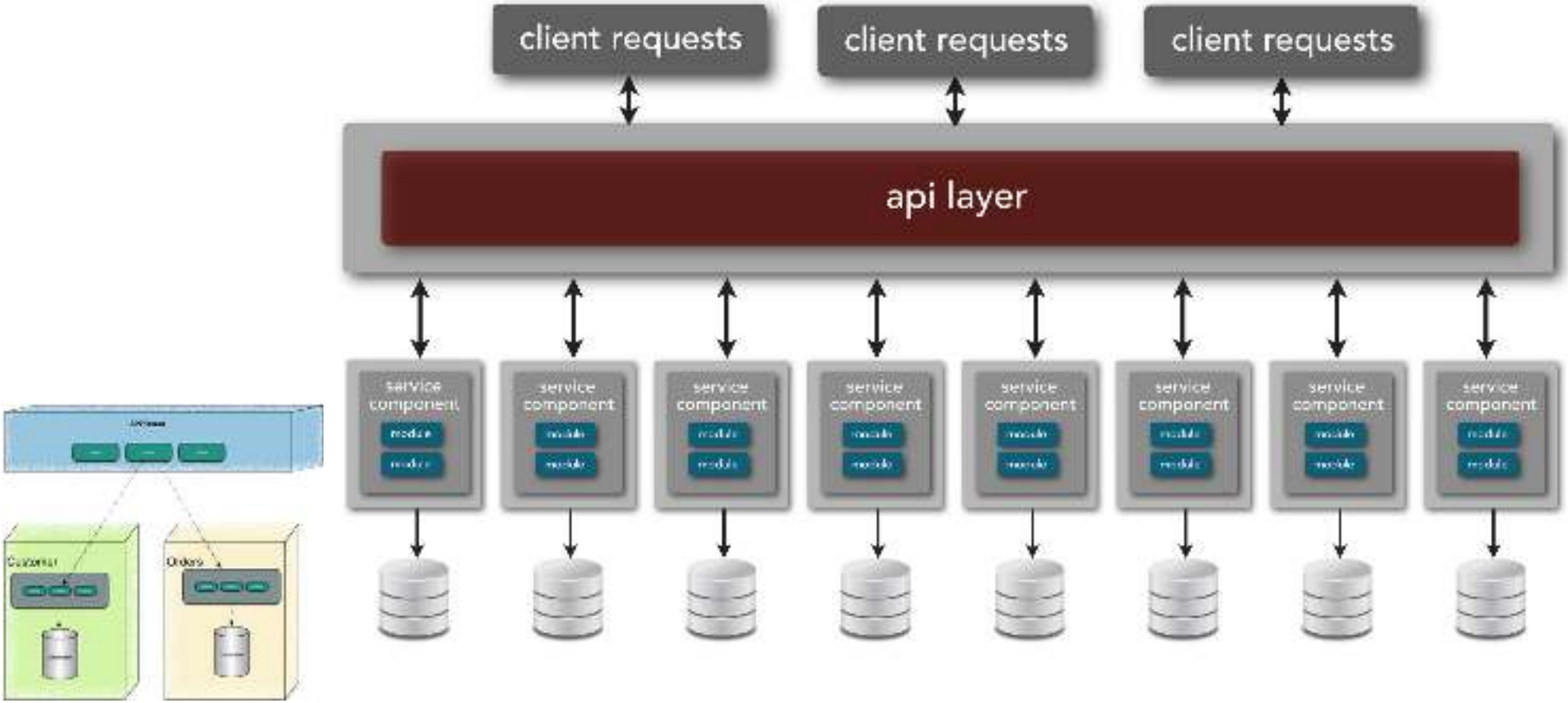
$$\frac{n(n-1)}{2}$$



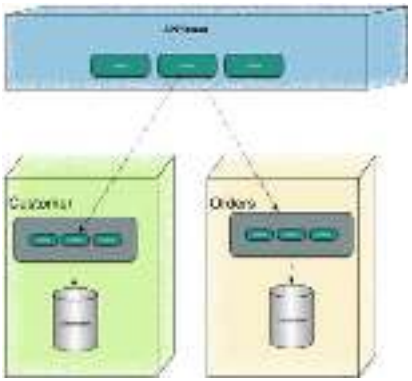
Architectural Quantum

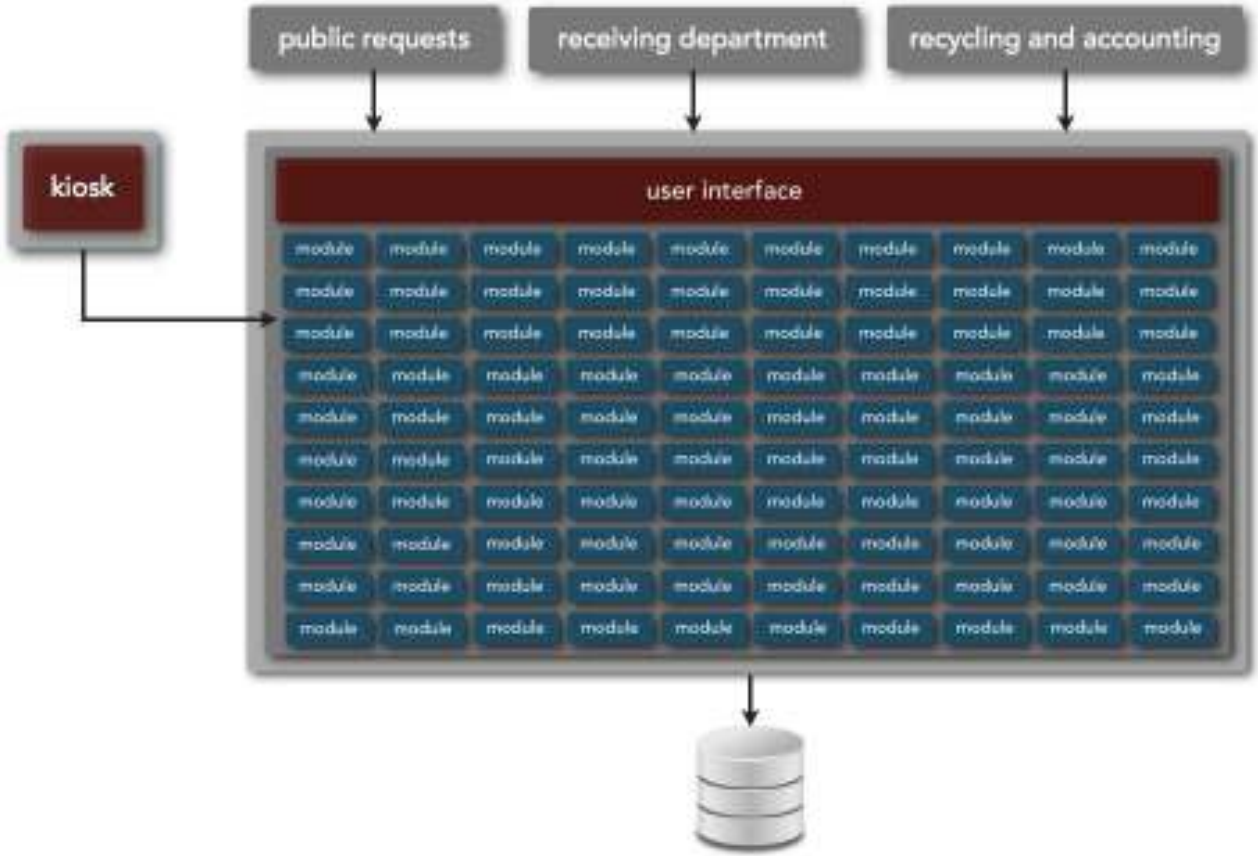


Architectural Quantum

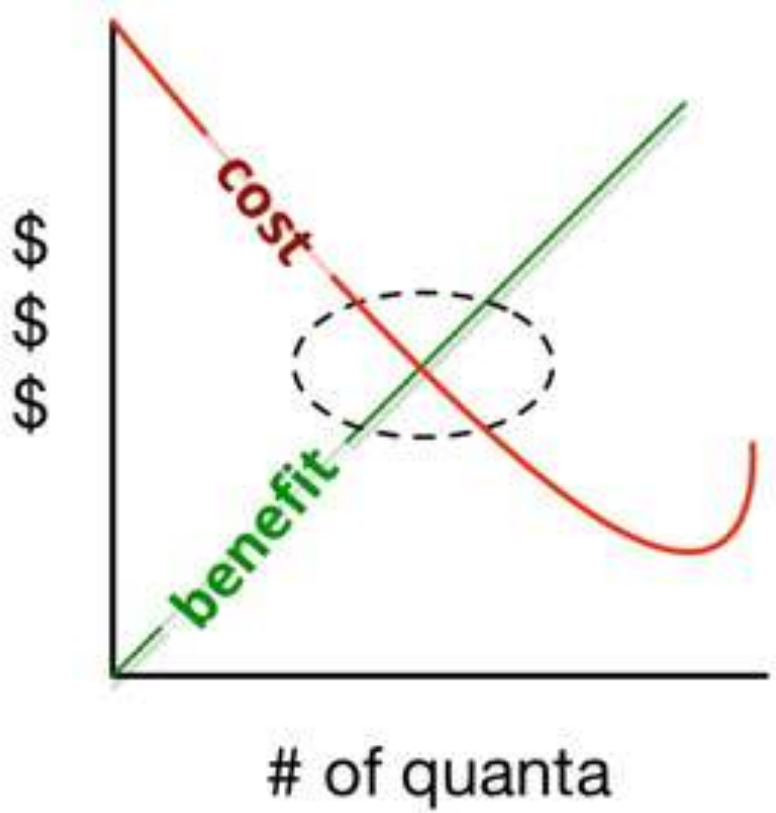


Architectural Quantum

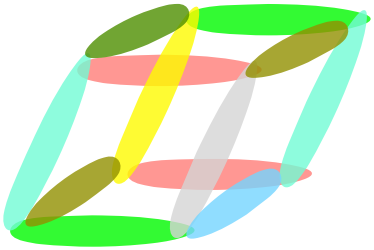




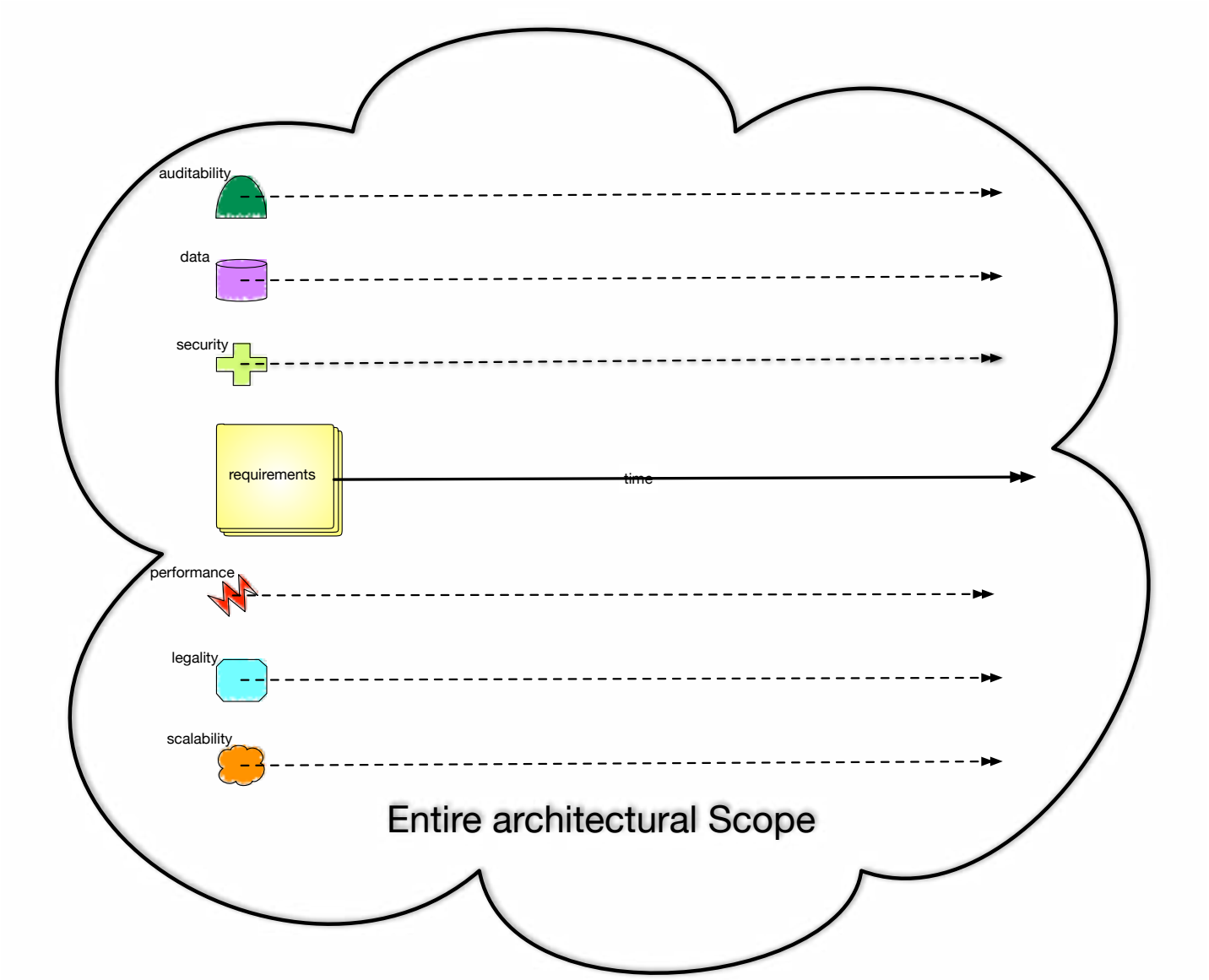
Architectural Quantum

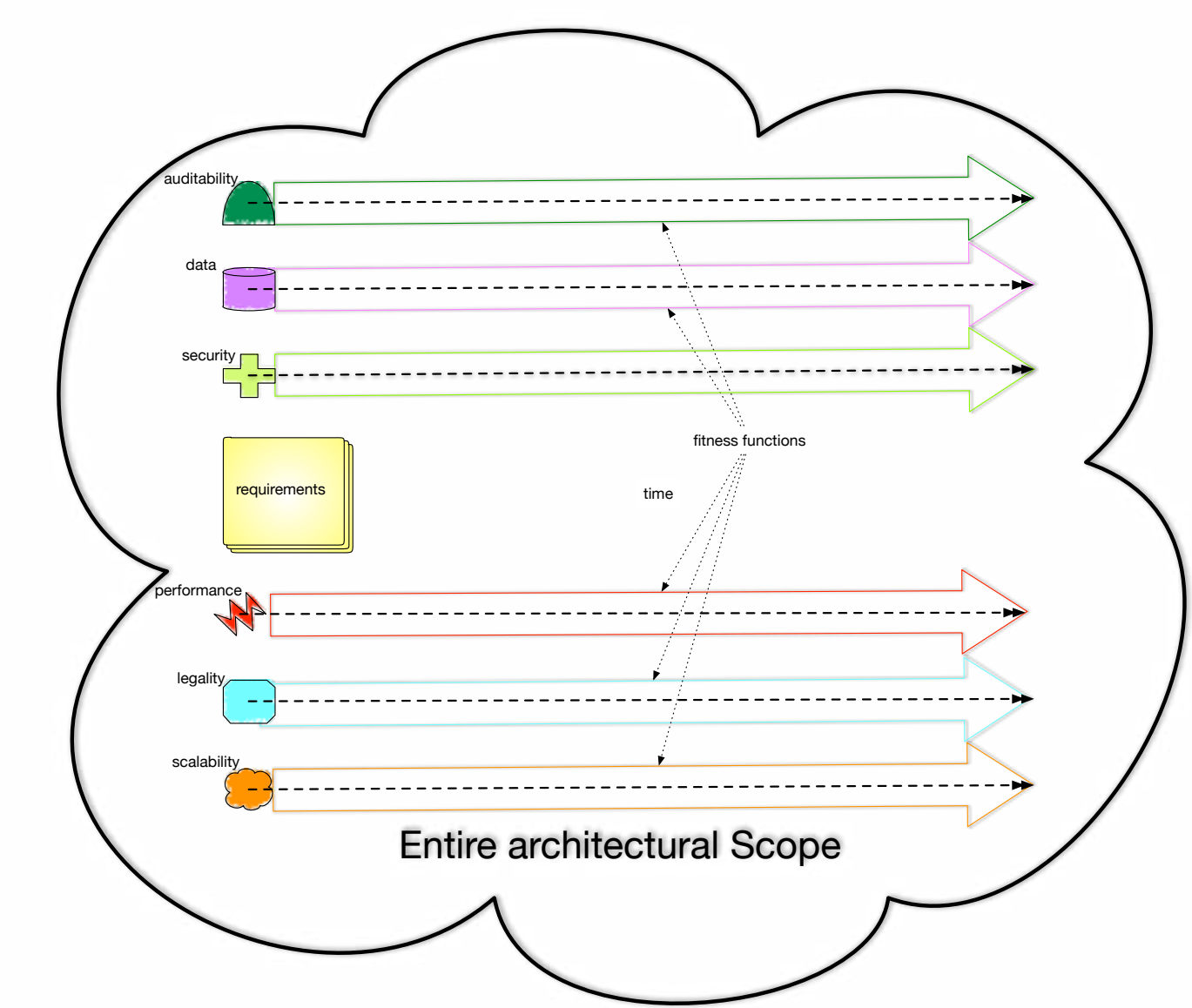


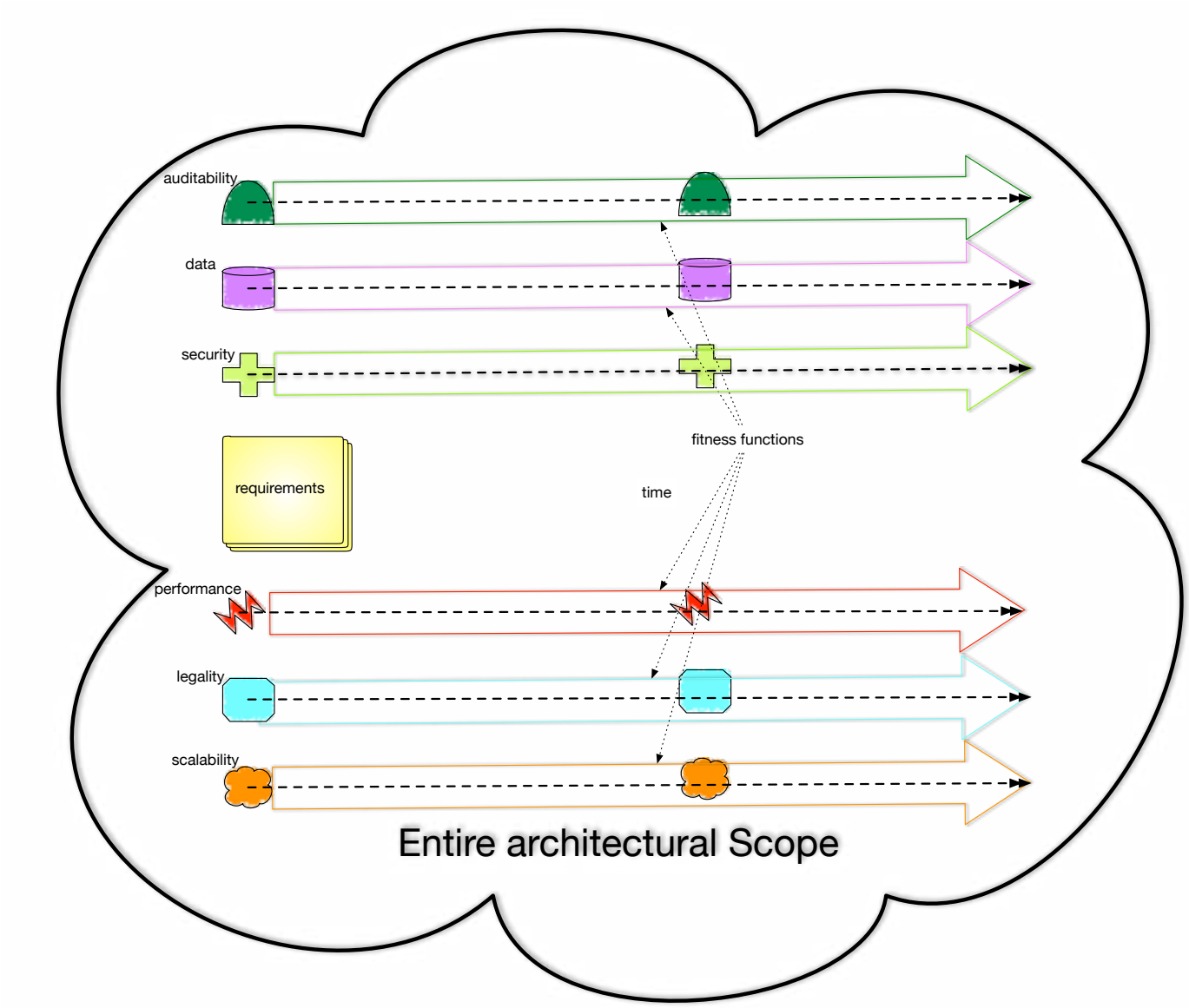
Utilizing Evolutionary Architecture



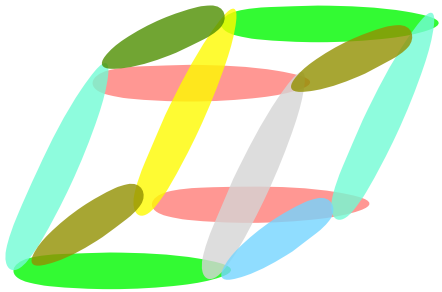
1. Choose Dimensions

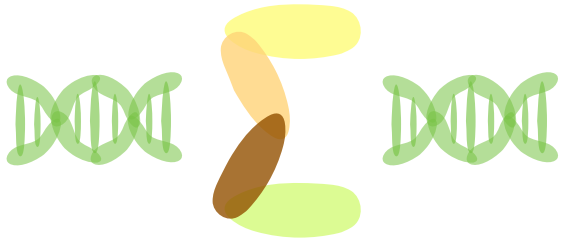

auditability
data
security
requirements
time
performance
legality
scalability
Entire architectural Scope


auditability
data
security
fitness functions
requirements
time
performance
legality
scalability
Entire architectural Scope


auditability
data
security
fitness functions
requirements
time
performance
legality
scalability
Entire architectural Scope

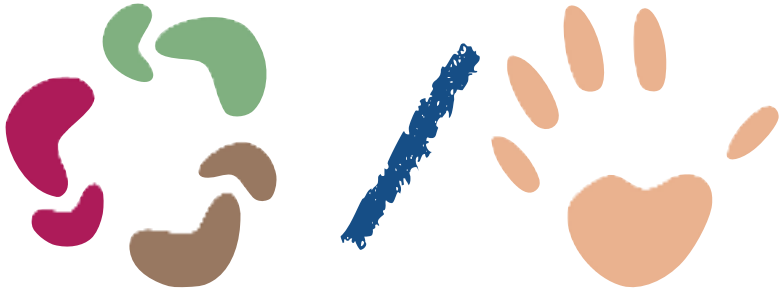
Utilizing Evolutionary Architecture



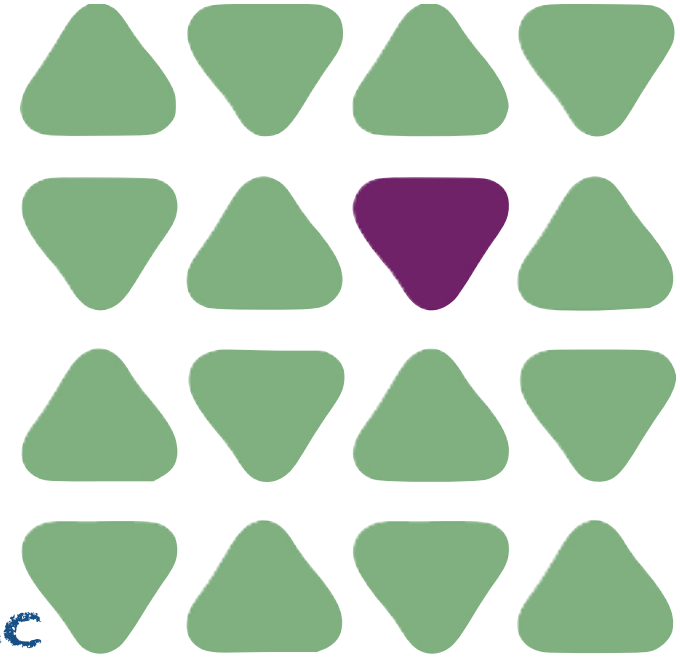


2. Identify Fitness Functions

atomic

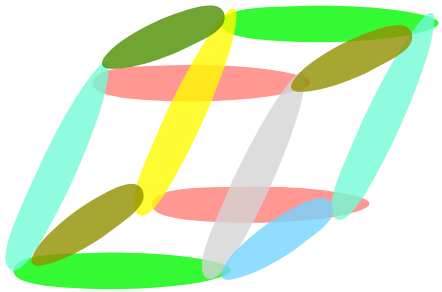


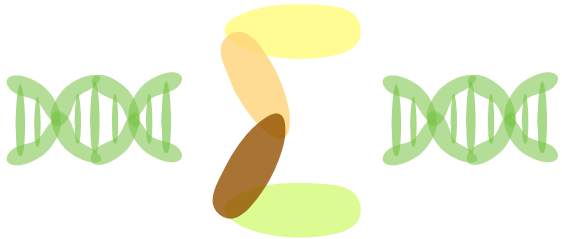
automated / manual



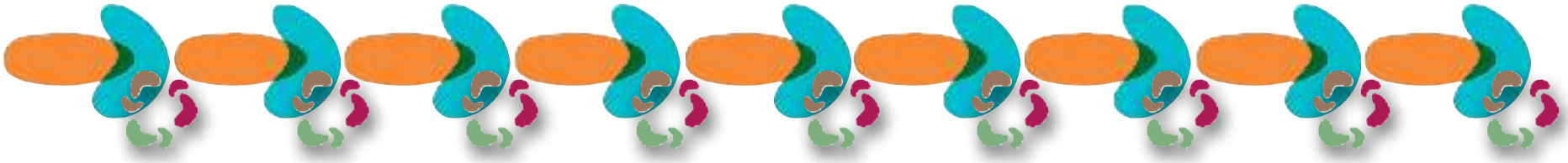
holistic

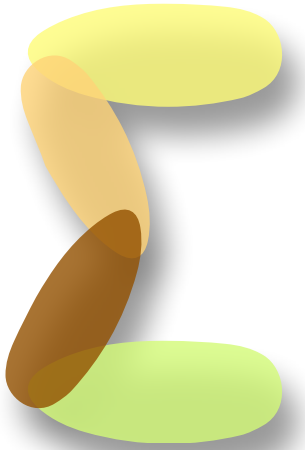
Utilizing Evolutionary Architecture

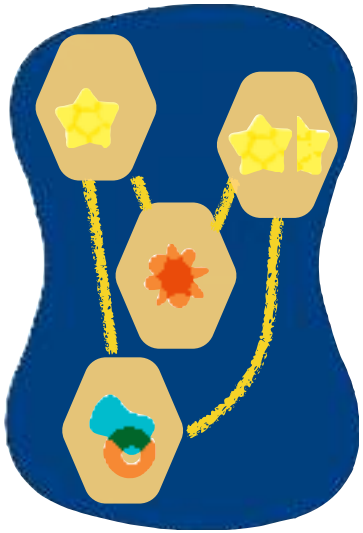




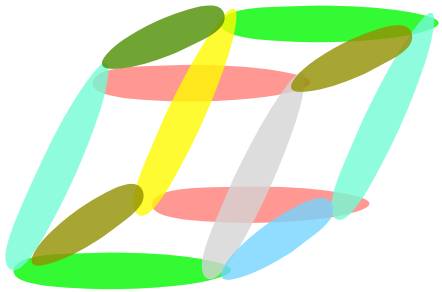
3. Apply Incremental Change

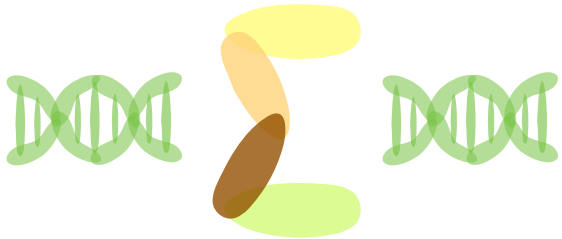


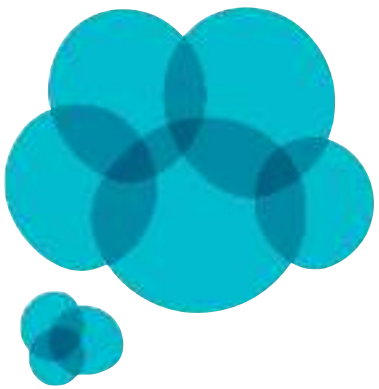


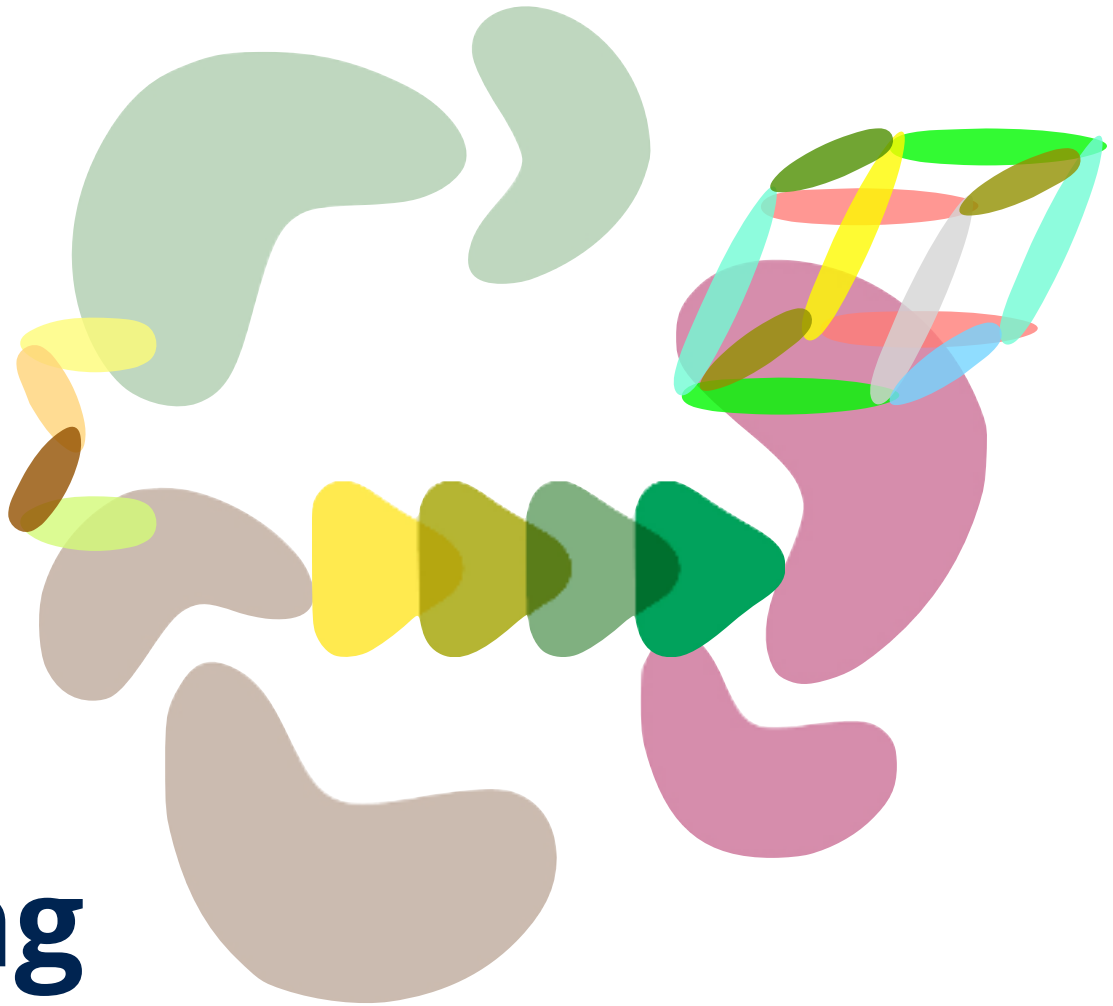


Utilizing Evolutionary Architecture



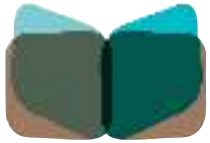






Utilizing Evolutionary Architecture

Agenda

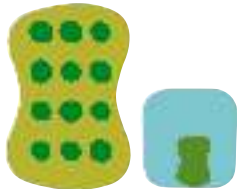


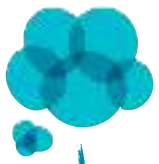
definition

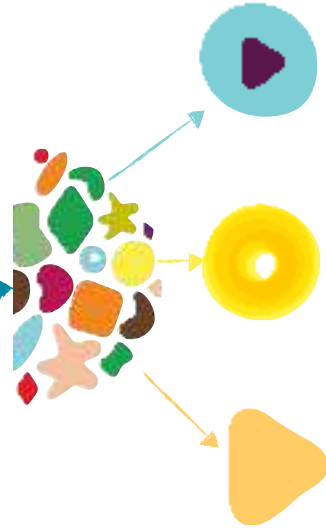
incremental change

fitness functions

appropriate coupling







Why should a company decide to build an evolutionary architecture?

Predictable versus Evolvable

Scale

Why should a company decide to build an evolutionary architecture?

Cycle Time as a Business Metric

Isolating "-ilities" at the Quantum Level

Longer Lasting Useful Systems

Advanced Business Capabilities

Why should a company decide to build an evolutionary architecture?

Why would a company choose *not* to build an evolutionary architecture?

Can't Evolve a Ball of Mud

Why would a company choose *not* to build an evolutionary architecture?

Other Architectural Characteristics Dominate

Sacrificial Architecture

Planning on Closing the Business Soon

Predictable versus Evolvable

Scale

Why should a company decide to build an evolutionary architecture?

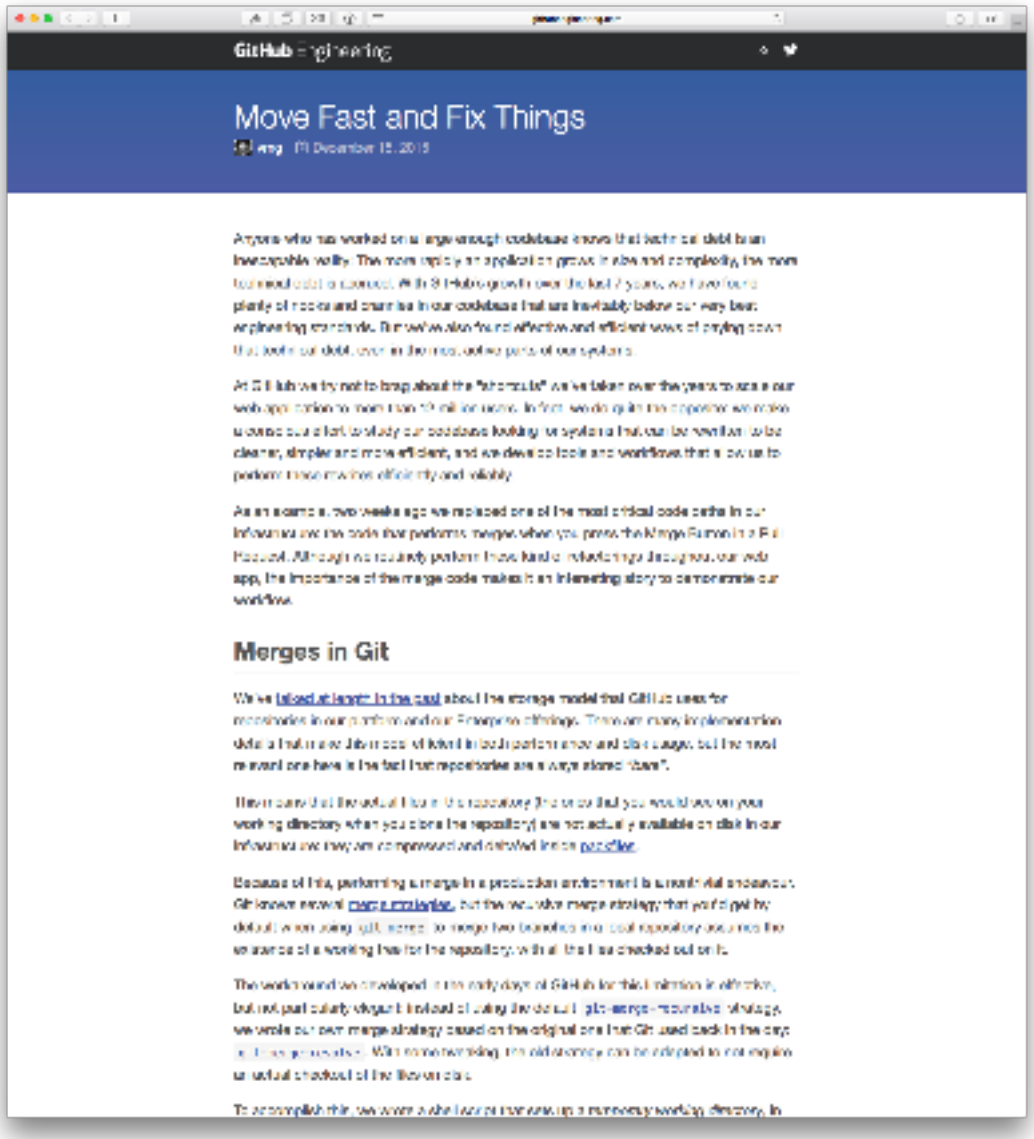
Cycle Time as a Business Metric

Isolating "-ilities" at the Quantum Level

Longer Lasting Useful Systems

Advanced Business Capabilities

Hypothesis and Data Driven Development



Move Fast & Fix Things

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

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

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

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

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

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

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

https://github.com/github/scientist

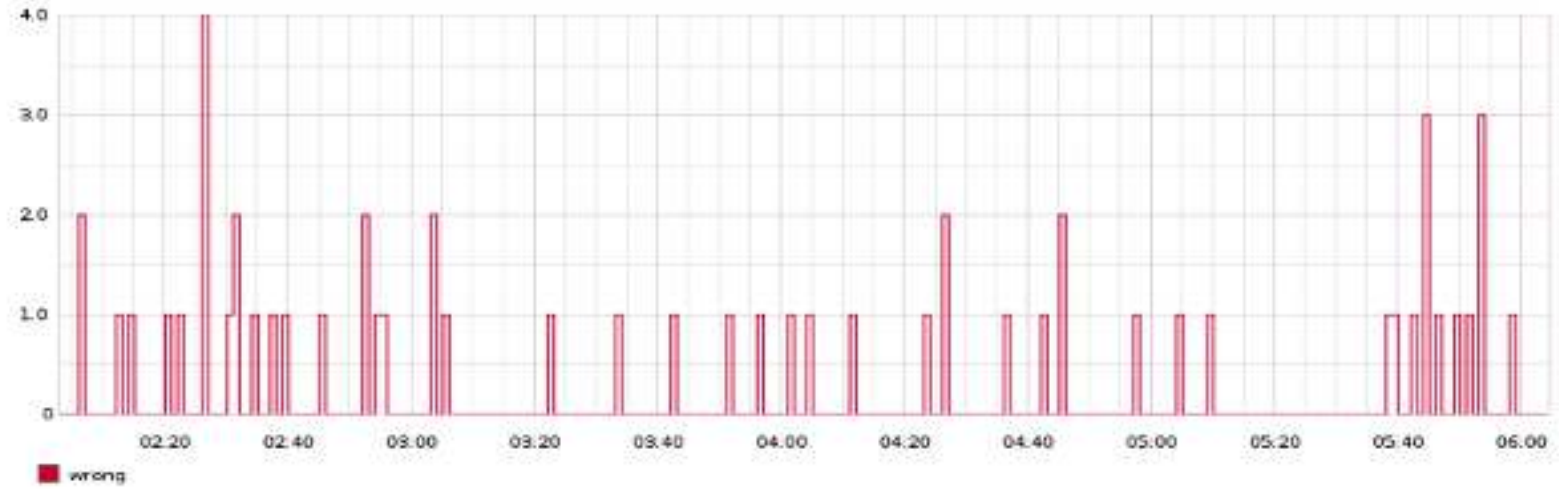
```ruby
require "scientist"

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

    experiment.run
  end
end
```

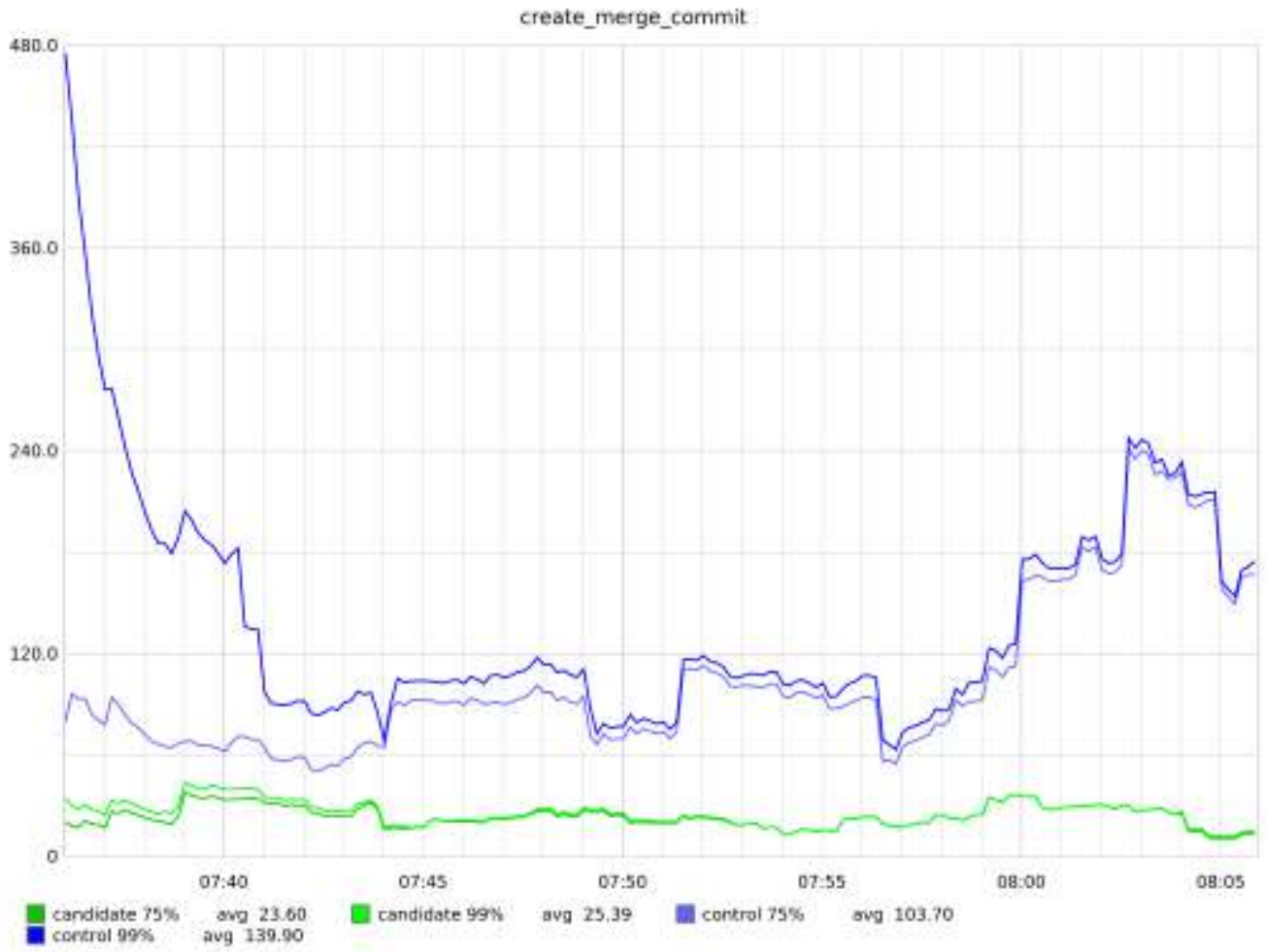
- It decides whether or not to run the try block,
- Randomizes the order in which use and try blocks are run,
- Measures the durations of all behaviors,
- Compares the result of try to the result of use,
- Swallows (but records) any exceptions raised in the try block
- Publishes all this information.

Accuracy

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

The number of incorrect/ignored only.




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

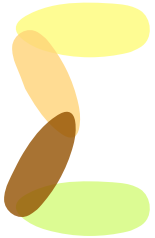
Bugs Found; Resolution

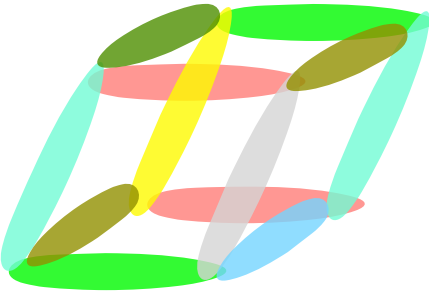
- faster conflict return because shell script exited immediately; replicated in library
- index write was causing O(n) problem; inlined into memory
- the ancestor had a file with a given filemode, whilst one side of the merge had removed the file and the other side had changed the filemode; bug in git!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

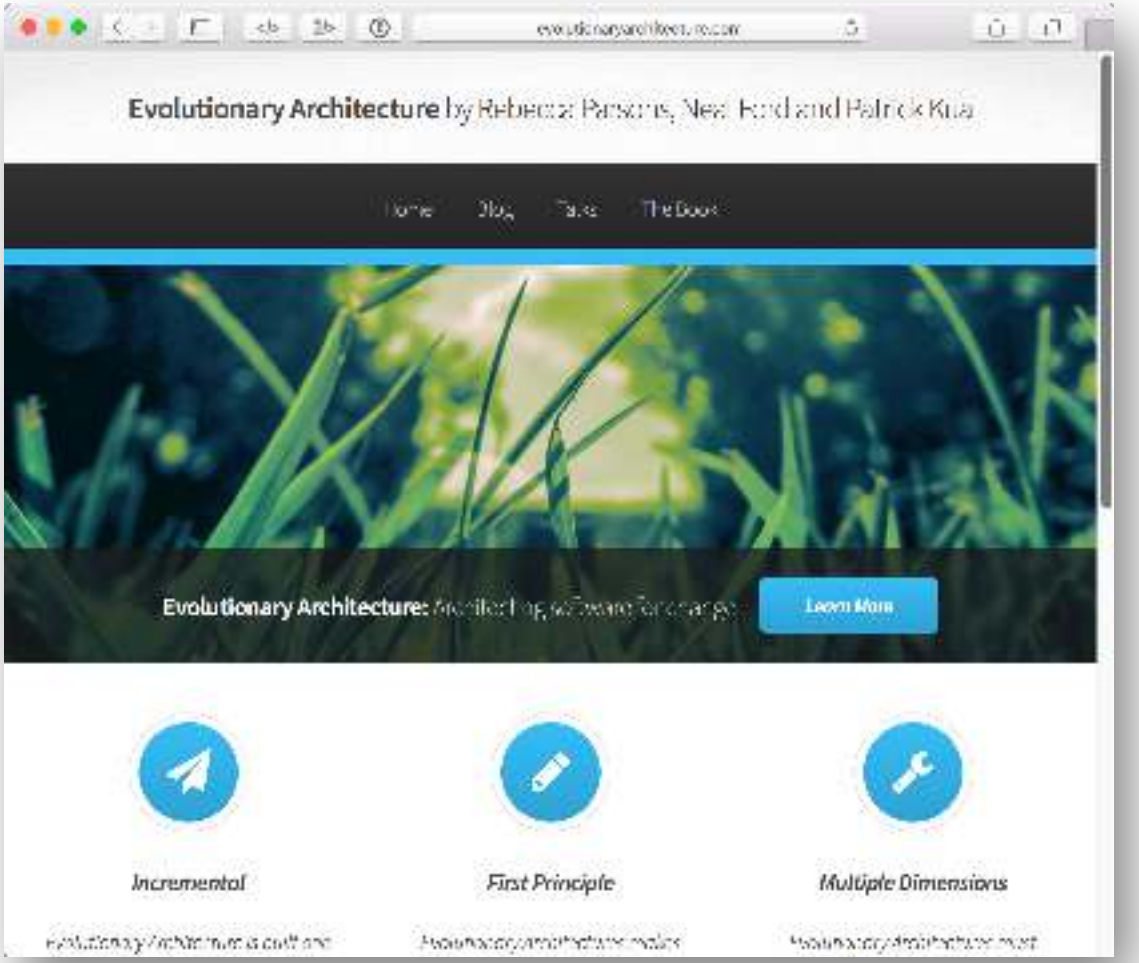
Definition:

evolutionary architecture

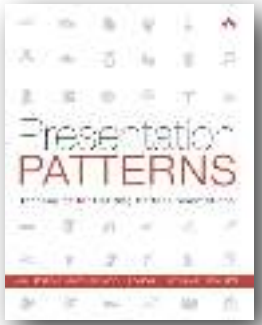
An evolutionary architecture supports incremental, guided change as a first principle across multiple dimensions.

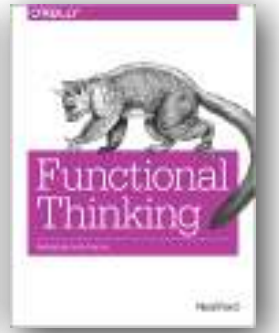






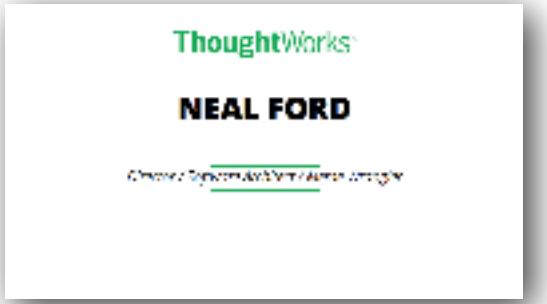
http://evolutionaryarchitecture.com



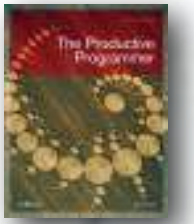


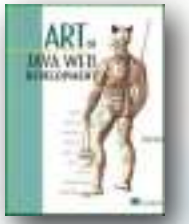
nealford.com

@neal4d



nealford.com/videos

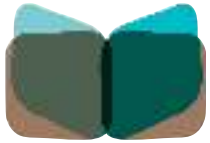




www.oreilly.com/software-architecture-video-training-series.html

nealford.com/books

Agenda



definition

incremental change

fitness functions

appropriate coupling