# aRChiTectUrE cHArAcTErISticS

Neal Ford

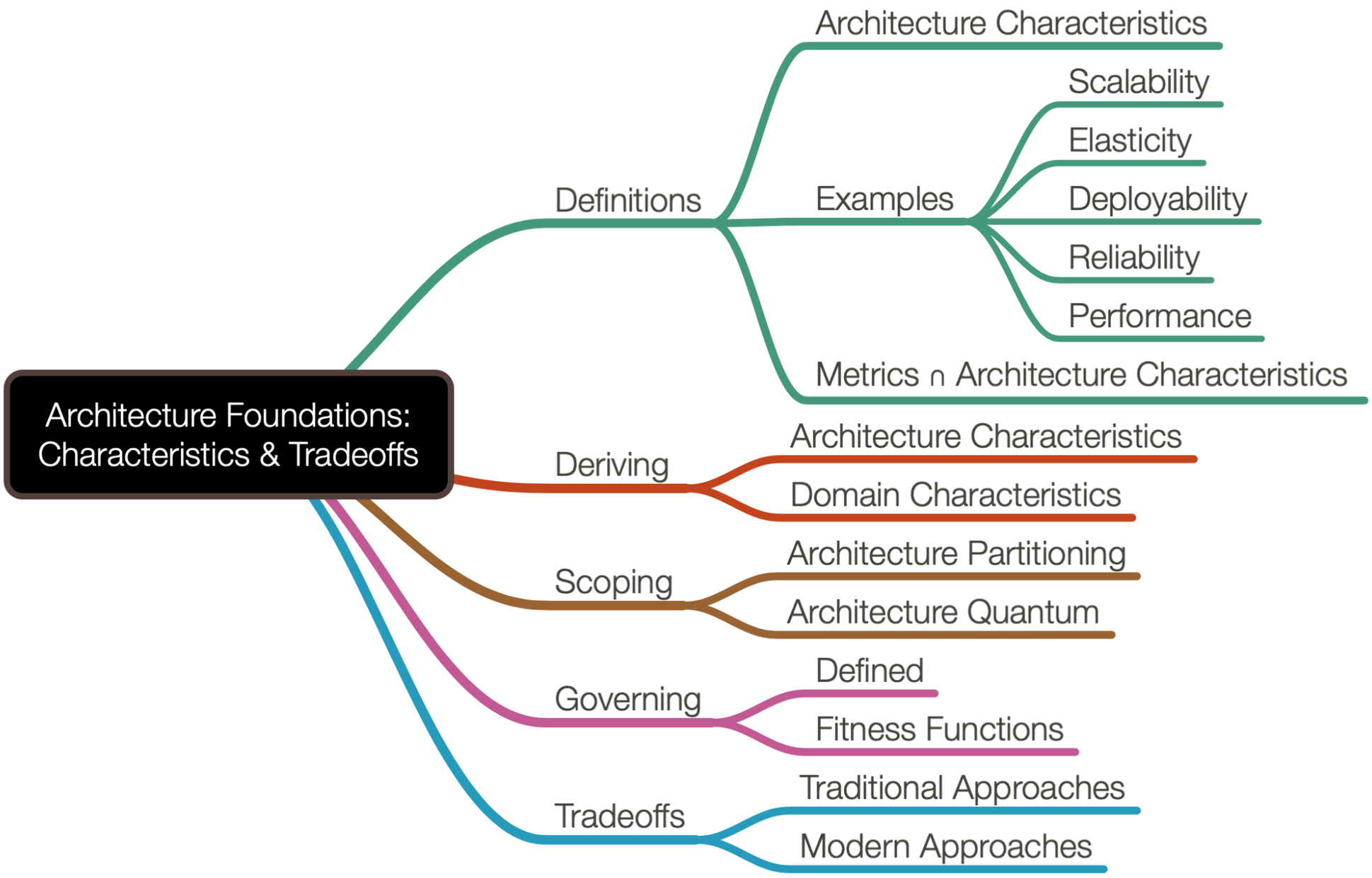
director / software architect / meme wrangler

/thoughtworks

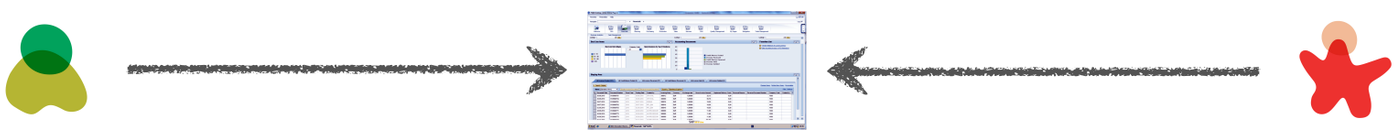
@neal4d

http://nealford.com

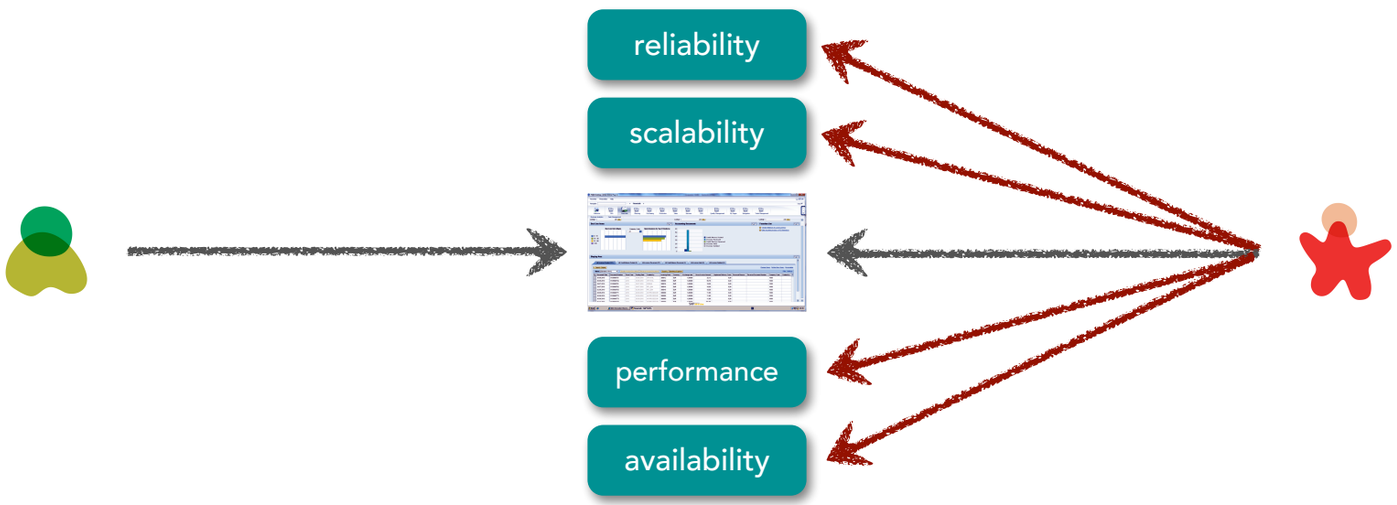
2021-08-02


Architecture Foundations: Characteristics & Tradeoffs
Definitions
Architecture Characteristics
Examples
Scalability
Elasticity
Deployability
Reliability
Performance
Metrics ∩ Architecture Characteristics
Deriving
Architecture Characteristics
Domain Characteristics
Scoping
Architecture Partitioning
Architecture Quantum
Governing
Defined
Fitness Functions
Tradeoffs
Traditional Approaches
Modern Approaches

# Architecture Characteristics



# Architecture Characteristics



# Architecture Characteristics

1. synergistic

2. ill-defined

3. voluminous

# Architecture Characteristics

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
evolvability
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
reliability
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

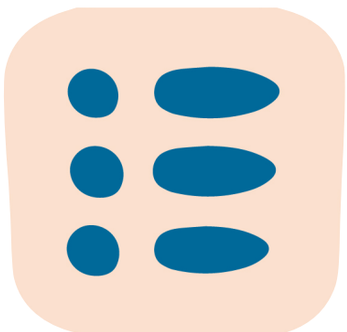
# Architecture Characteristics

1. synergistic

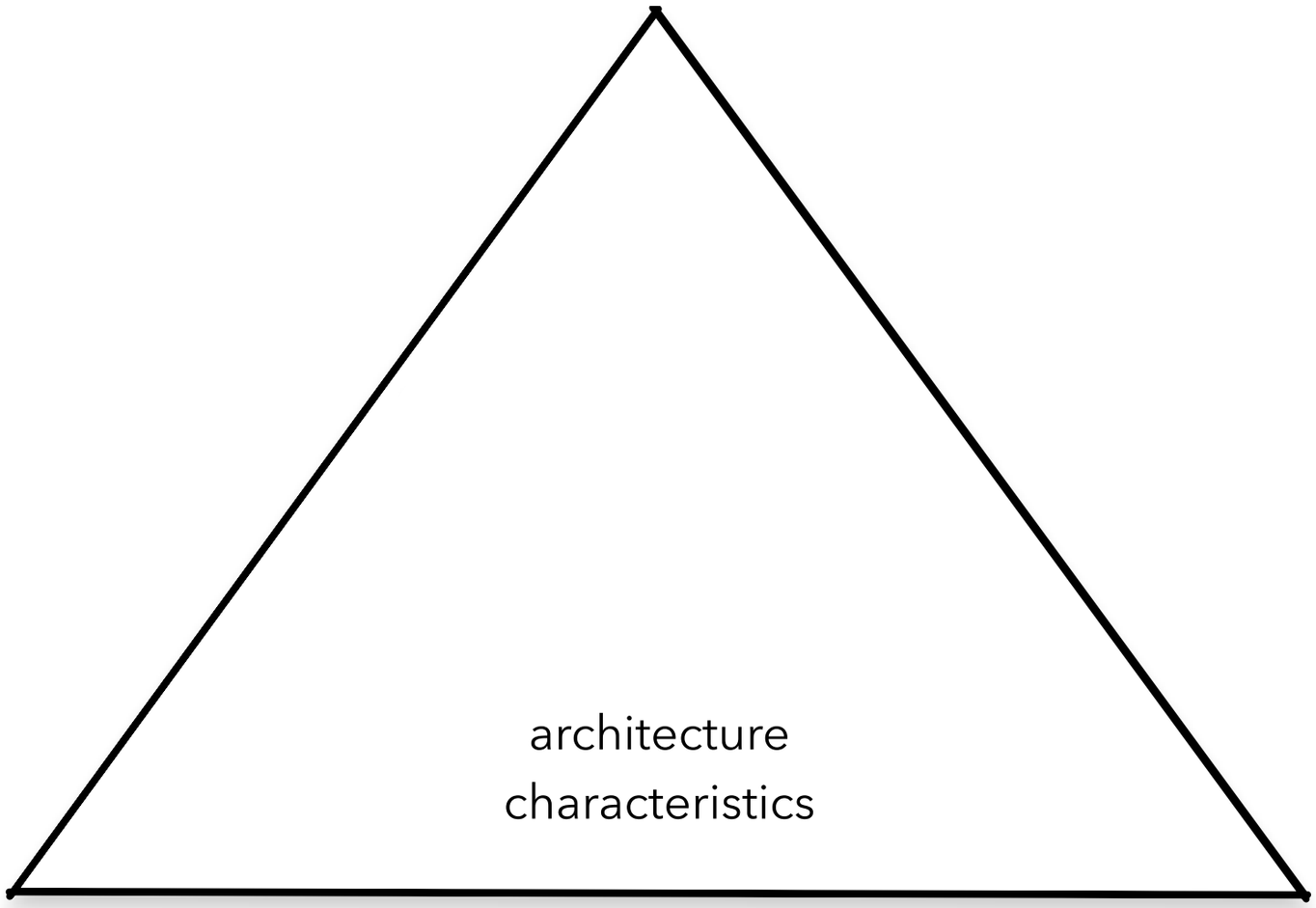
2. ill-defined

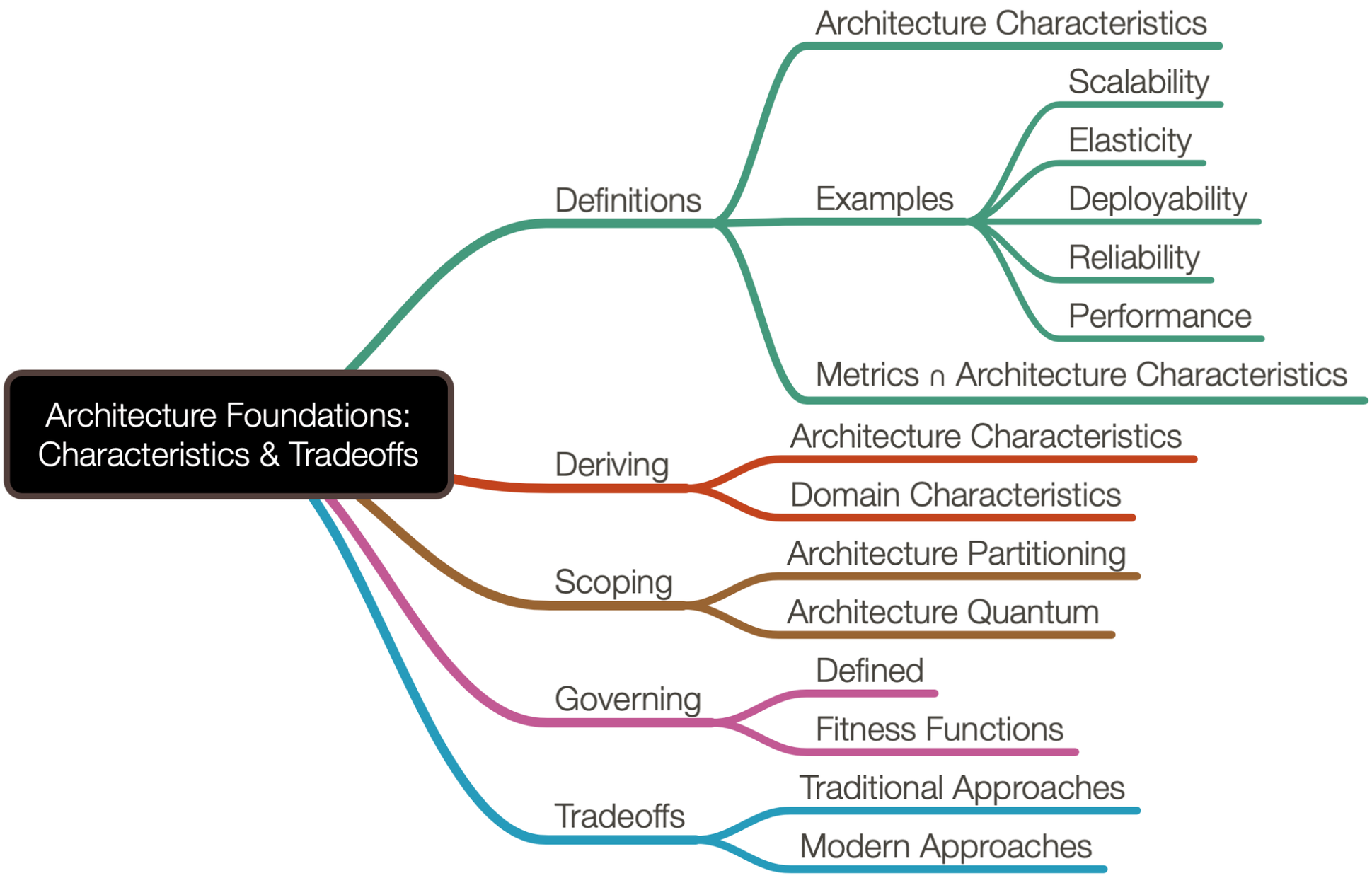
4. numerous

categories

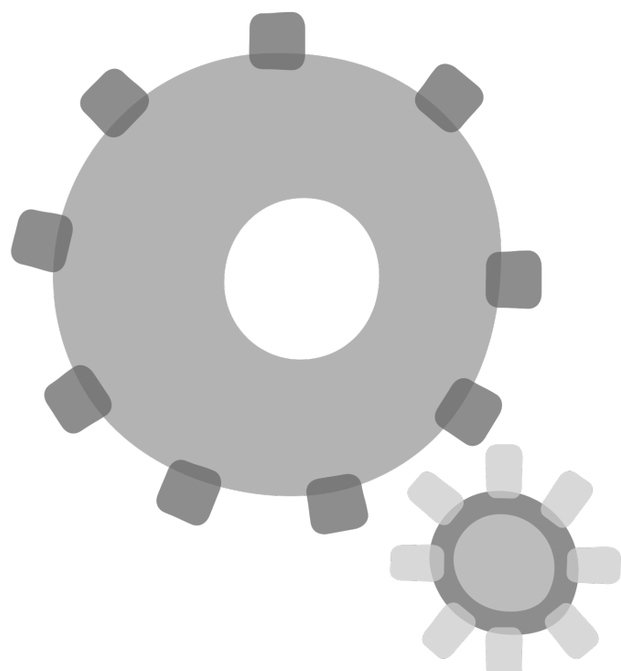


3. voluminous

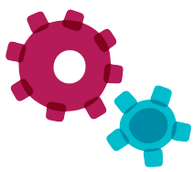

architecture
characteristics

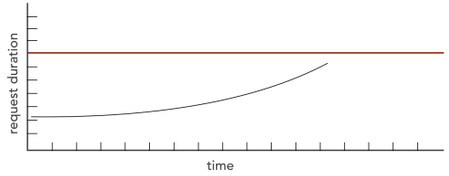

Architecture Foundations: Characteristics & Tradeoffs
Definitions
Architecture Characteristics
Examples
Scalability
Elasticity
Deployability
Reliability
Performance
Metrics ∩ Architecture Characteristics
Deriving
Architecture Characteristics
Domain Characteristics
Scoping
Architecture Partitioning
Architecture Quantum
Governing
Defined
Fitness Functions
Tradeoffs
Traditional Approaches
Modern Approaches

# Operational Architecture Characteristics

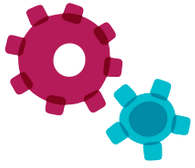


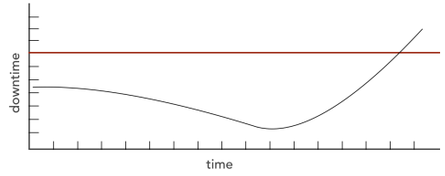
# Performance






measure and track average response times (cumulative)



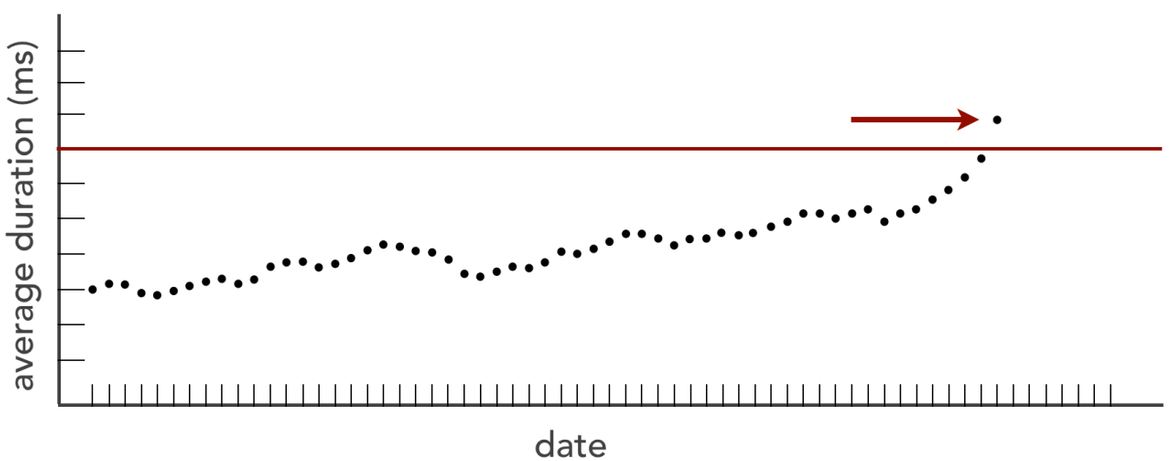



measure and track maximum response times

# Cumulative Average Response Times

(per request, domain, or application context)

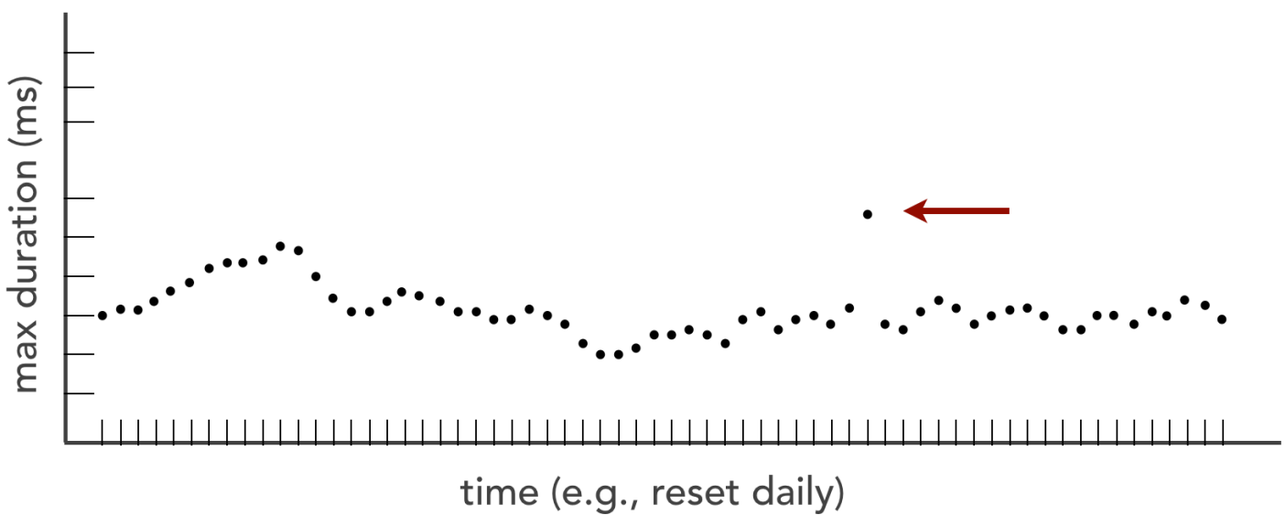
trigger alert when the average response times
within a particular context exceeds 1400ms



# Maximum Response Times

(per request, domain, or application context)

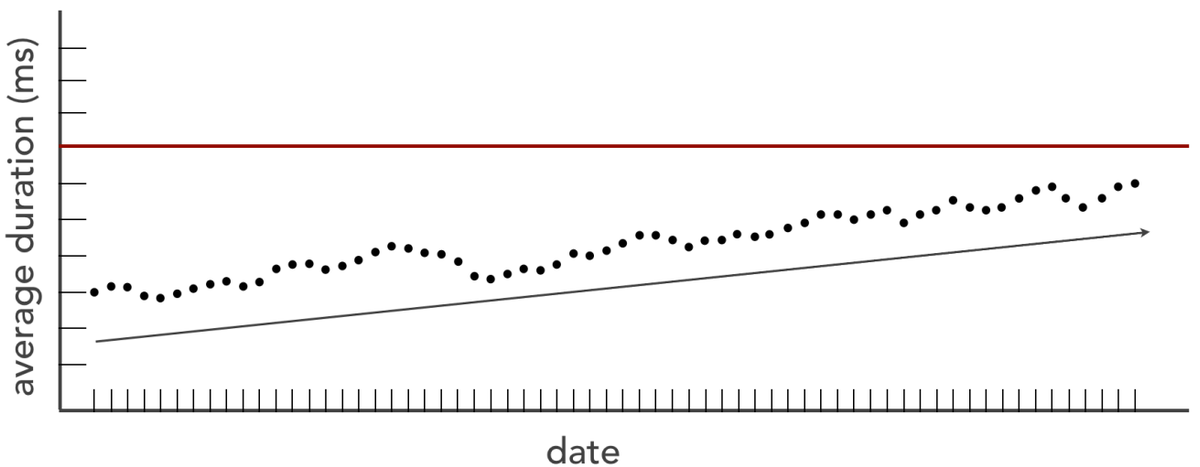
trigger alert anytime the maximum response time of selected requests exceeds 2000ms



# Cumulative Average Response Times

(per request, domain, or application context)

trigger alert when the average response times continues to increase over a 2 month period of time



# Performance

https://developers.google.com/web/tools/lighthouse/

https://www.google.com/gmail/about/
Nov 4, 2018, 11:37 AM PST
Emulated Nexus 5X, Simulated Slow 4G network

| 93 | 50 | 43 | 86 | 100 |
|---|---|---|---|---|
| Performance | Progressive Web App | Accessibility | Best Practices | SEO |

Score scale: 90-100 50-89 0-49

## Performance

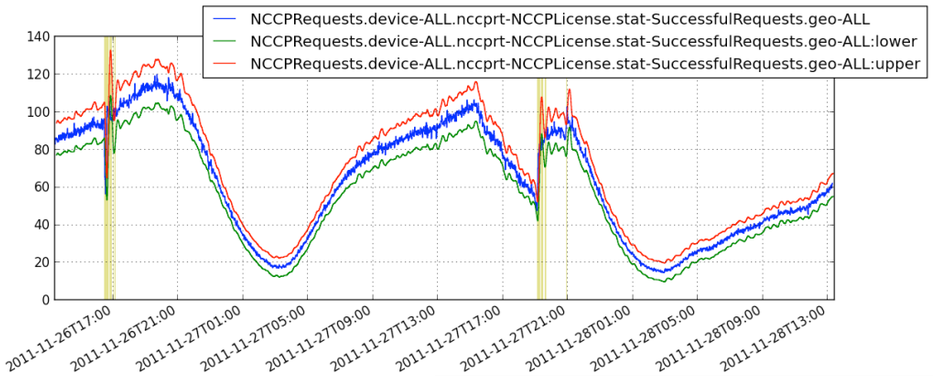
93

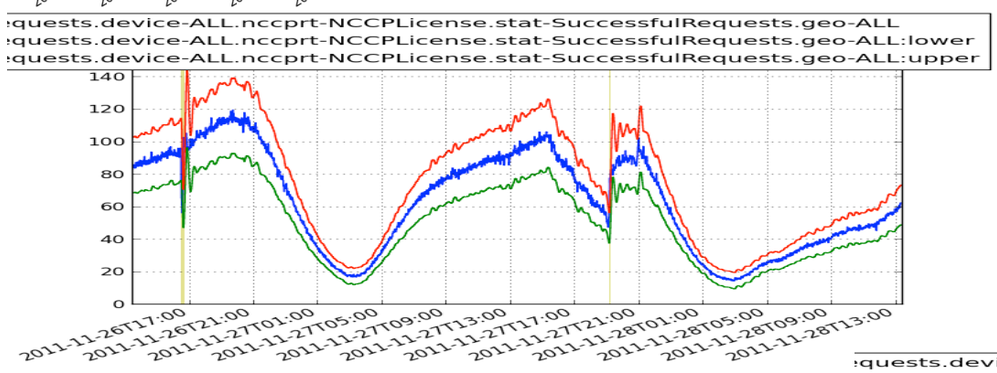
### Metrics

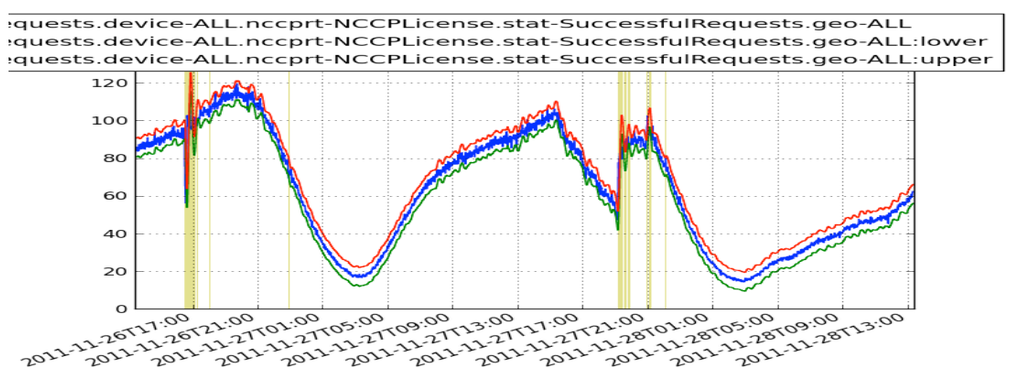
| | | | |
|---|---|---|---|
| First Contentful Paint | 1.6 s | First Meaningful Paint | 1.6 s |
| Speed Index | 3.3 s | First CPU Idle | 3.5 s |
| Time to Interactive | 3.7 s | Estimated Input Latency | 40 ms |

Values are estimated and may vary.

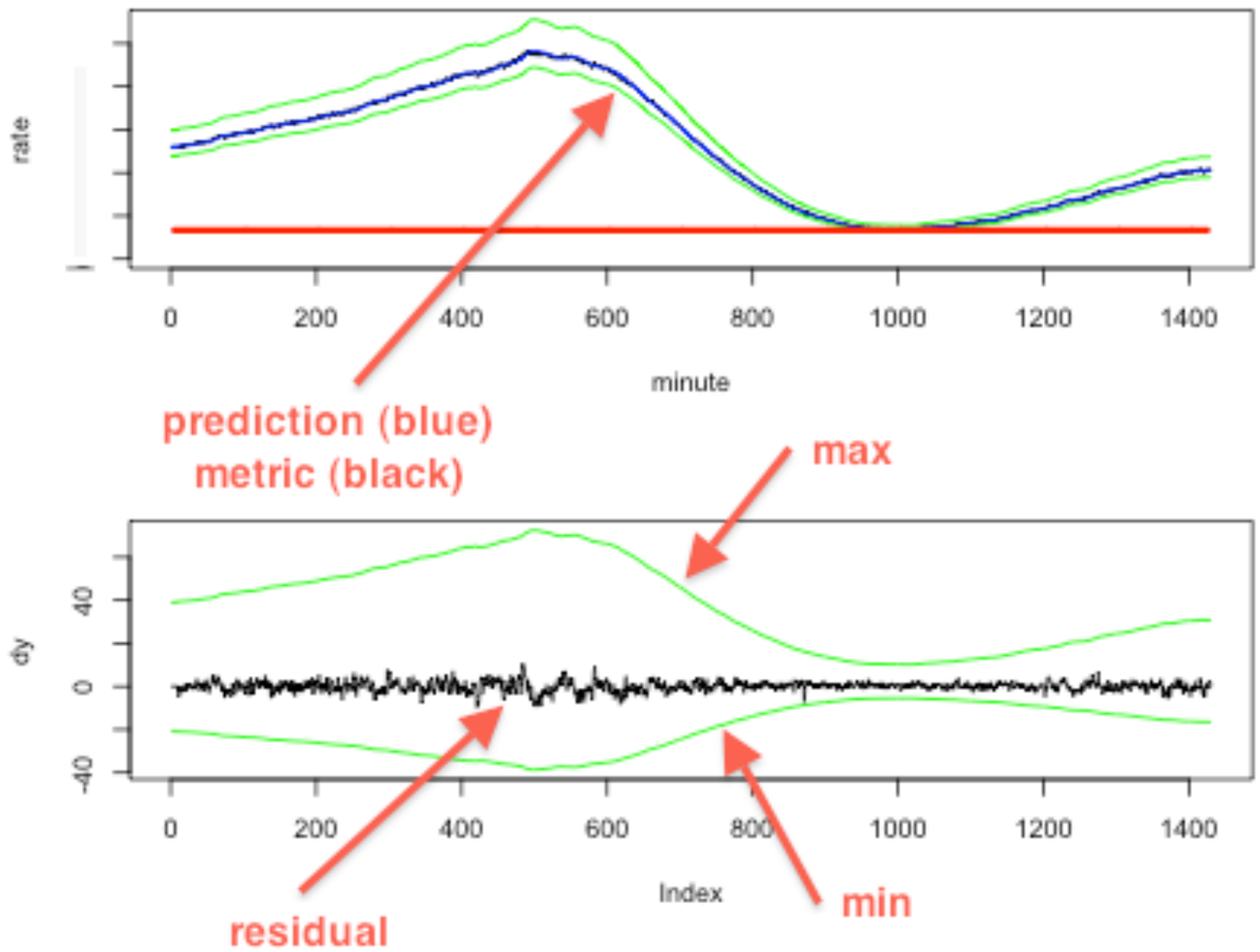
# Double Exponential Smoothing



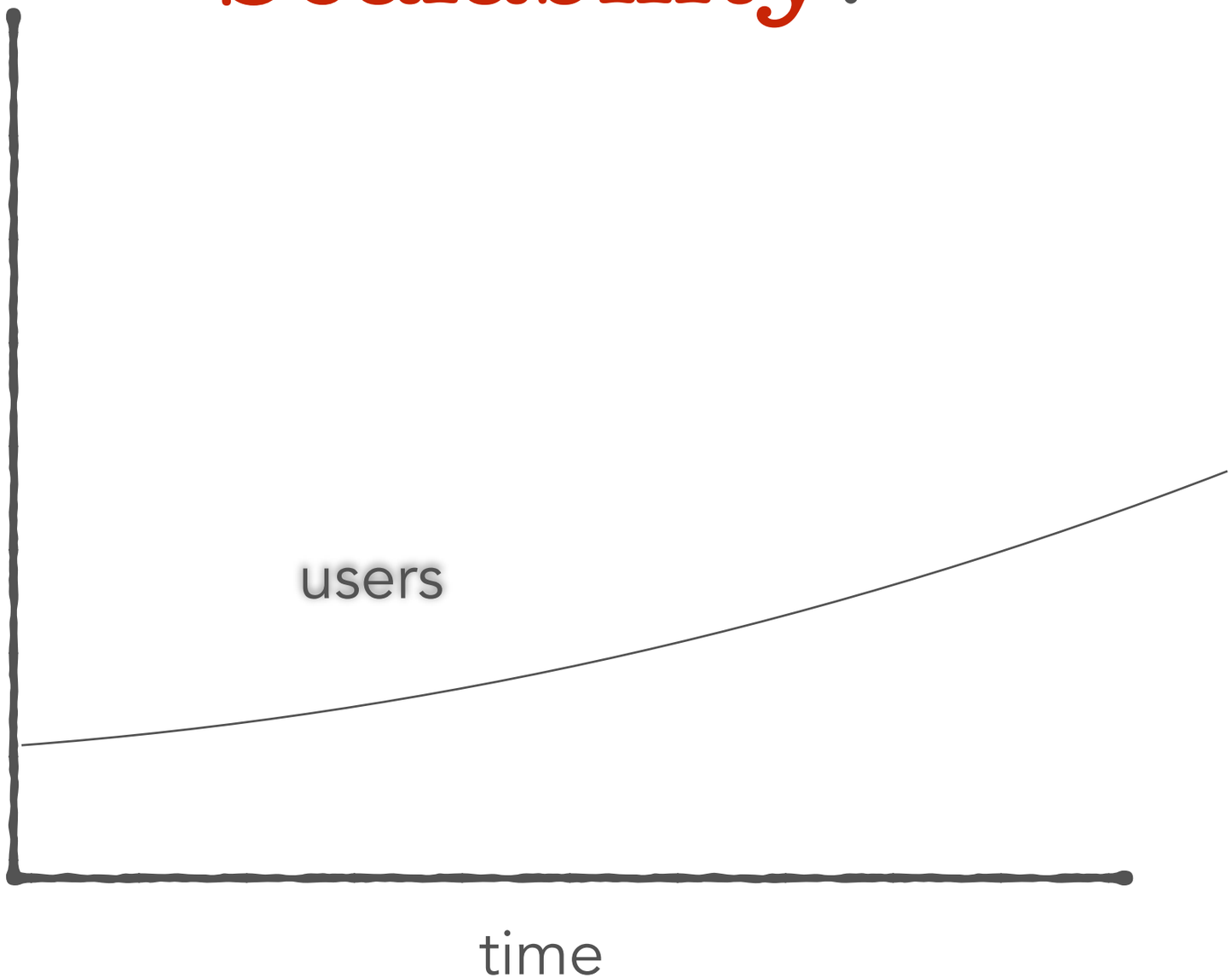



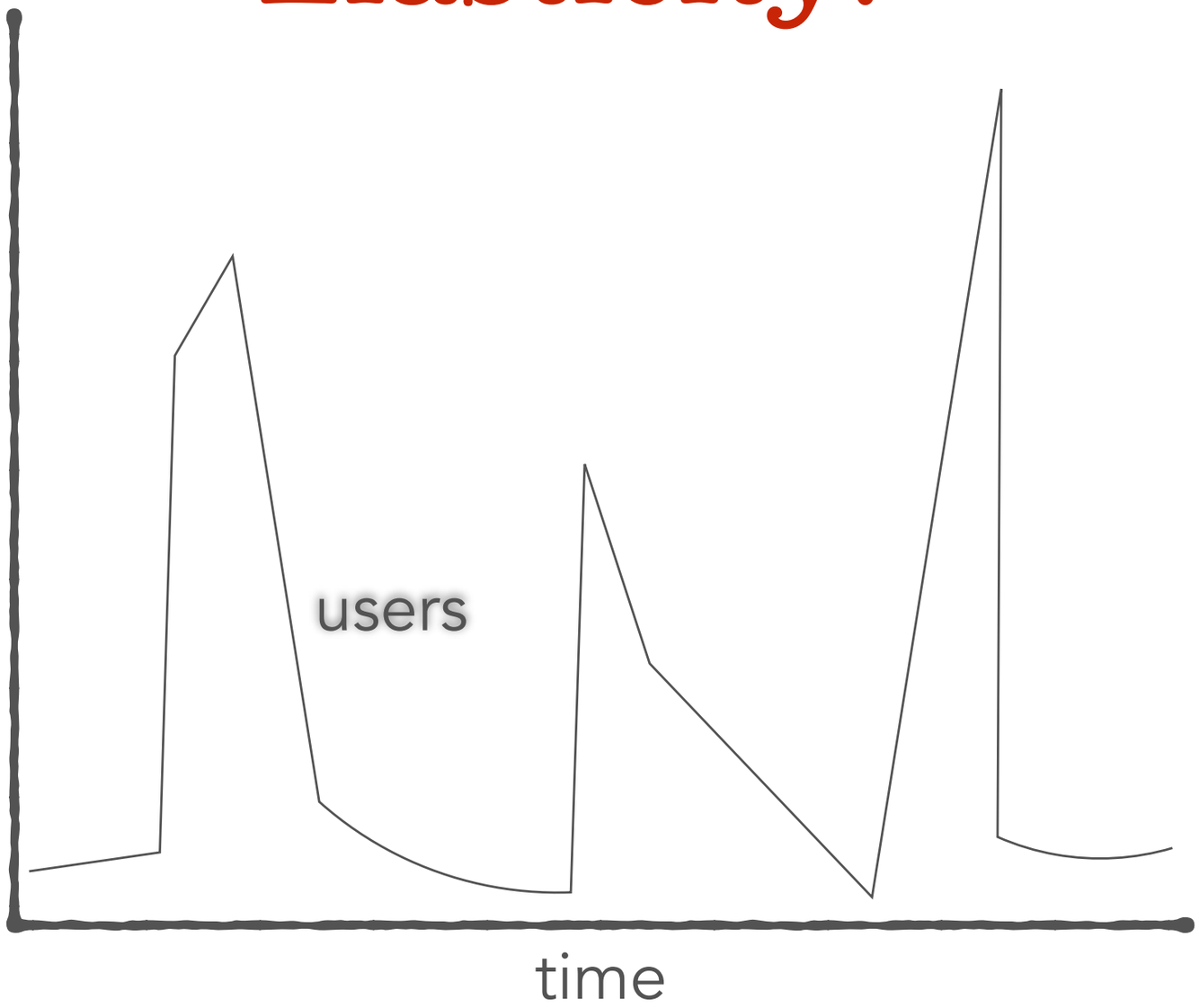
# Alert Tuning Fitness Function



# Scalability:

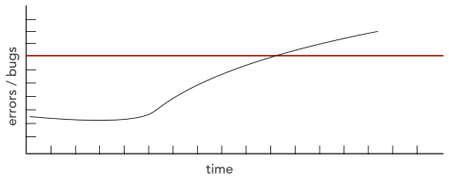


# Elasticity:

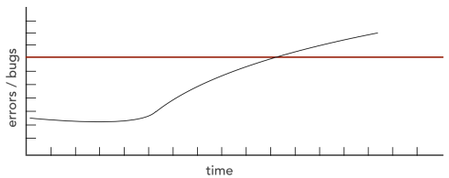


# Quality Architecture Characteristics

# Fault Tolerance




measure and track number of users impacted by a system or service crash




measure and track number of requests impacted by a system or service crash

# Reliability

the ability of a system or component to function under stated conditions for a specified period of time.

# Reliability

composite

the ability of a system or component to function under stated conditions for a specified period of time.

Q: How many ways can engineers measure *reliability*?

# Availability

$$X(t) = \begin{cases} 1, & \text{sys functions at time } t \\ 0, & \text{otherwise} \end{cases}$$

The availability A(t) at time t > 0 is:

$$A(t) = \Pr[X(t) = 1] = E[X(t)].$$

$$A_c = \frac{1}{c} \int_0^c A(t)\, dt.$$

# Availability

Limiting average availability:

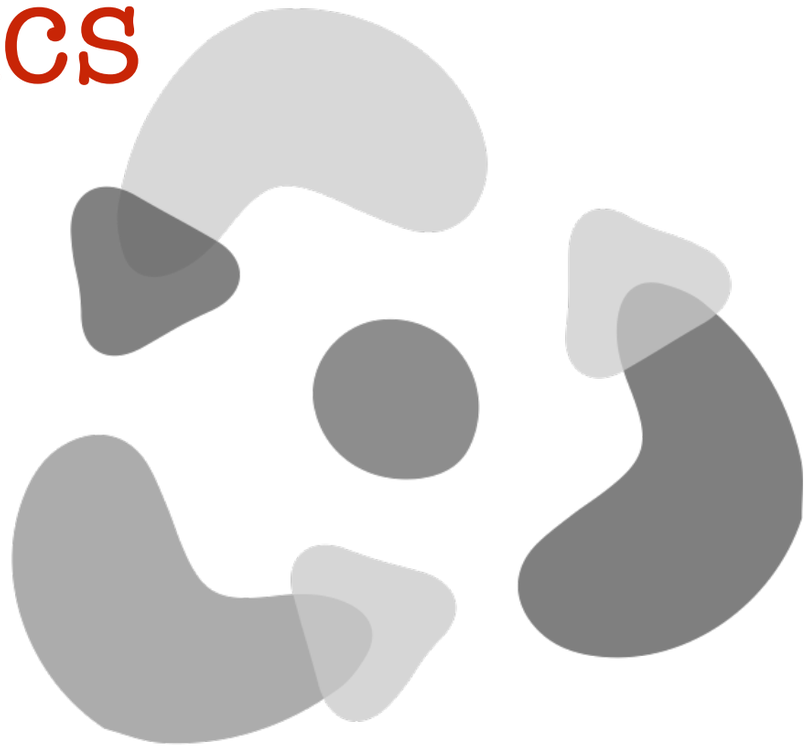
$$A_\infty = \lim_{c\to\infty} A_c = \lim_{c\to\infty} \frac{1}{c}\int_0^c A(t)\,dt, \quad c > 0$$

Q: How do engineers typically measure *availability*?

# High Availability

| Availability % | Downtime per year[note 1] | Downtime per month | Downtime per week | Downtime per day |
|---|---|---|---|---|
| 55.5555555% ("nine fives") | 162.33 days | 13.53 days | 74.92 hours | 10.67 hours |
| **90% ("one nine")** | **36.53 days** | **73.05 hours** | **16.80 hours** | **2.40 hours** |
| 95% ("one and a half nines") | 18.26 days | 36.53 hours | 8.40 hours | 1.20 hours |
| 97% | 10.96 days | 21.92 hours | 5.04 hours | 43.20 minutes |
| 98% | 7.31 days | 14.61 hours | 3.36 hours | 28.80 minutes |
| **99% ("two nines")** | **3.65 days** | **7.31 hours** | **1.68 hours** | **14.40 minutes** |
| 99.5% ("two and a half nines") | 1.83 days | 3.65 hours | 50.40 minutes | 7.20 minutes |
| 99.8% | 17.53 hours | 87.66 minutes | 20.16 minutes | 2.88 minutes |
| **99.9% ("three nines")** | **8.77 hours** | **43.83 minutes** | **10.08 minutes** | **1.44 minutes** |
| 99.95% ("three and a half nines") | 4.38 hours | 21.92 minutes | 5.04 minutes | 43.20 seconds |
| **99.99% ("four nines")** | **52.60 minutes** | **4.38 minutes** | **1.01 minutes** | **8.64 seconds** |
| 99.995% ("four and a half nines") | 26.30 minutes | 2.19 minutes | 30.24 seconds | 4.32 seconds |
| **99.999% ("five nines")** | **5.26 minutes** | **26.30 seconds** | **6.05 seconds** | **864.00 milliseconds** |
| **99.9999% ("six nines")** | **31.56 seconds** | **2.63 seconds** | **604.80 milliseconds** | **86.40 milliseconds** |
| **99.99999% ("seven nines")** | **3.16 seconds** | **262.98 milliseconds** | **60.48 milliseconds** | **8.64 milliseconds** |
| **99.999999% ("eight nines")** | **315.58 milliseconds** | **26.30 milliseconds** | **6.05 milliseconds** | **864.00 microseconds** |
| **99.9999999% ("nine nines")** | **31.56 milliseconds** | **2.63 milliseconds** | **604.80 microseconds** | **86.40 microseconds** |

# Process Architecture Characteristics



# Agility?

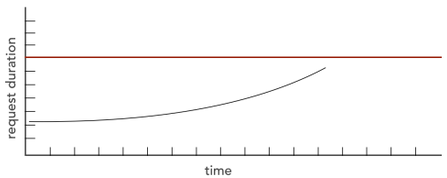
composite

deployability

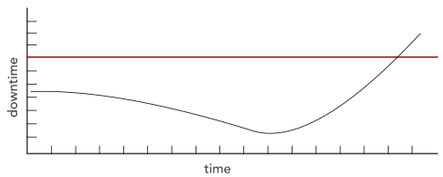
Agility?

# Deployability

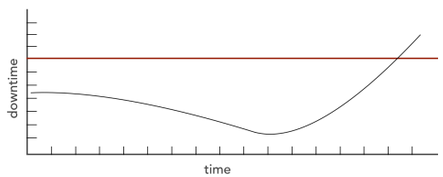
the ease of, risk, and frequency of deployment




measure and track number of actual hours spent to deploy




measure and track failed deployments or errors resulting from deployment

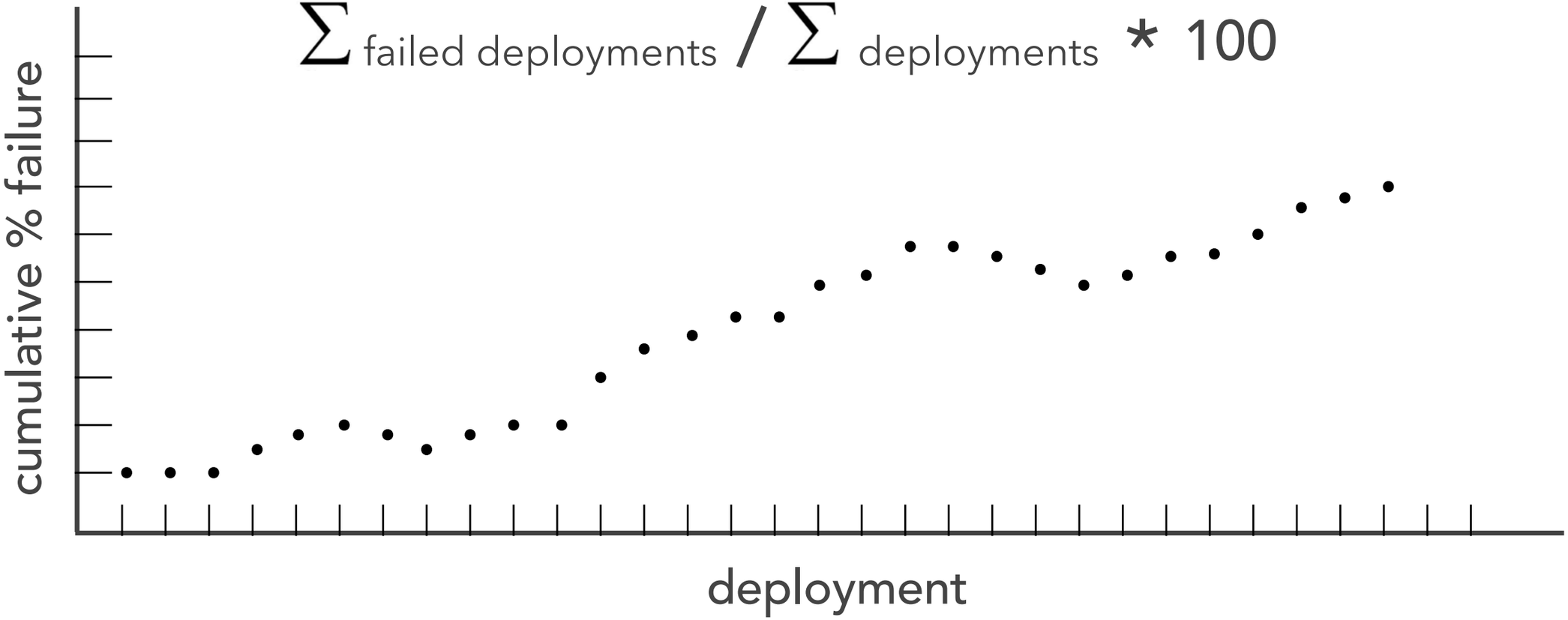



measure and track the frequency of deployment

# Deployment Errors/Failures

(per service, domain, or application context)

$$\sum \text{failed deployments} \;/\; \sum \text{deployments} \;*\; 100$$



testability

deployability

Agility?

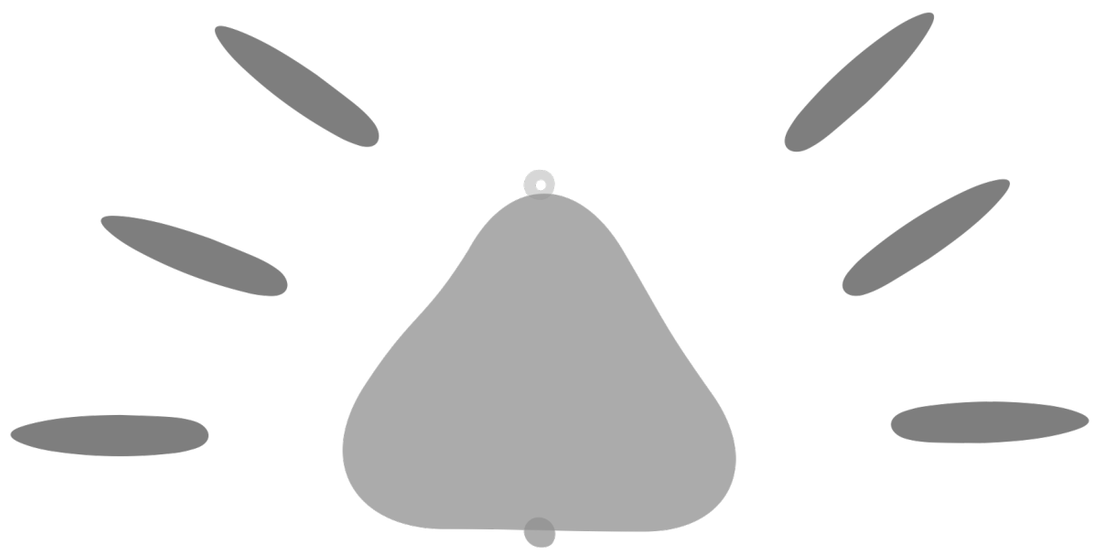
testability

deployability

Agility?

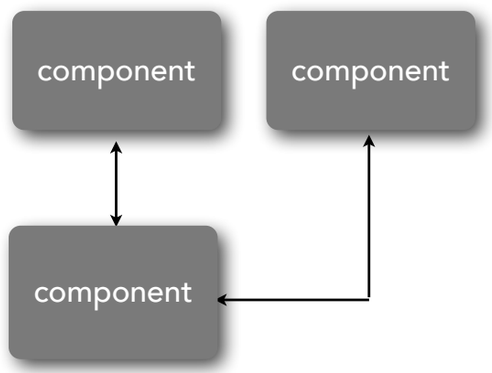
modularity

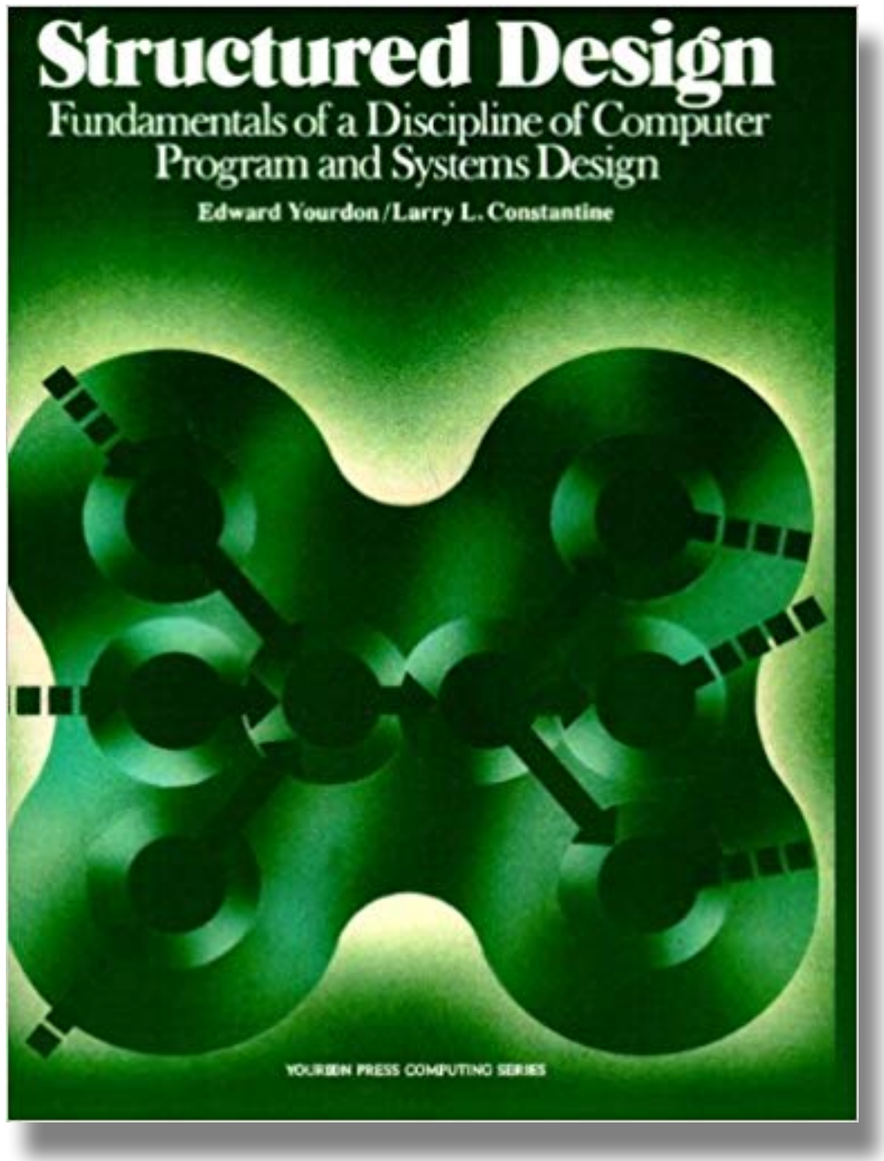
# Internal Quality Architecture Characteristics



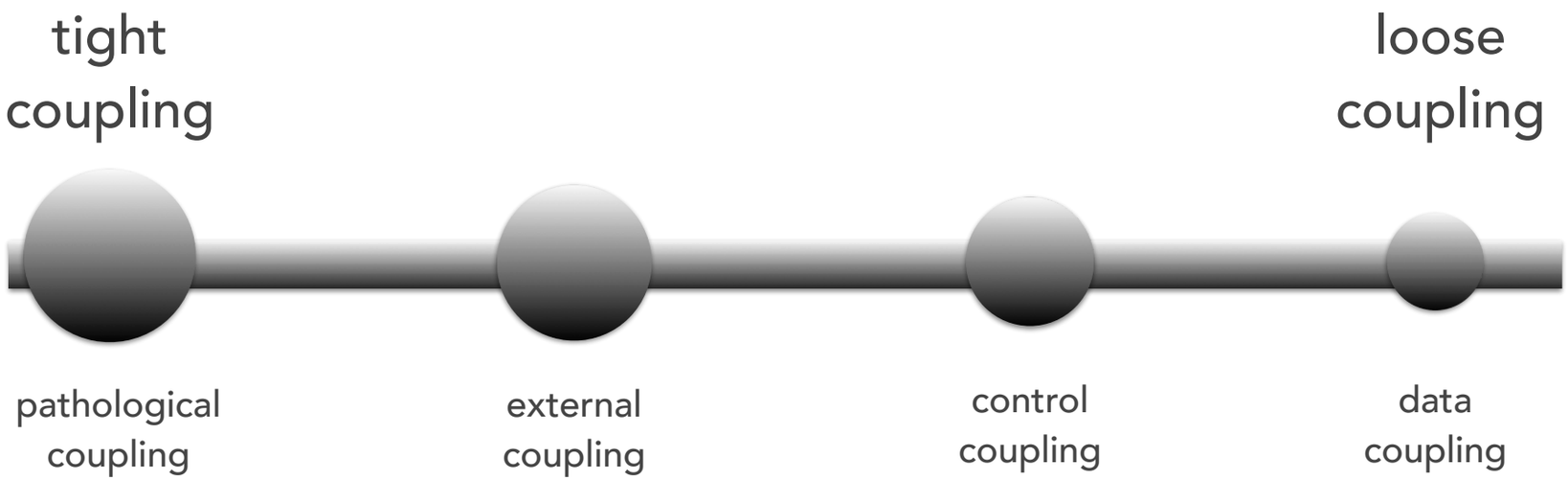
# Component Coupling

the extent to which components know about each other




Structured Design
Fundamentals of a Discipline of Computer Program and Systems Design
Edward Yourdon / Larry L. Constantine
YOURDON PRESS COMPUTING SERIES

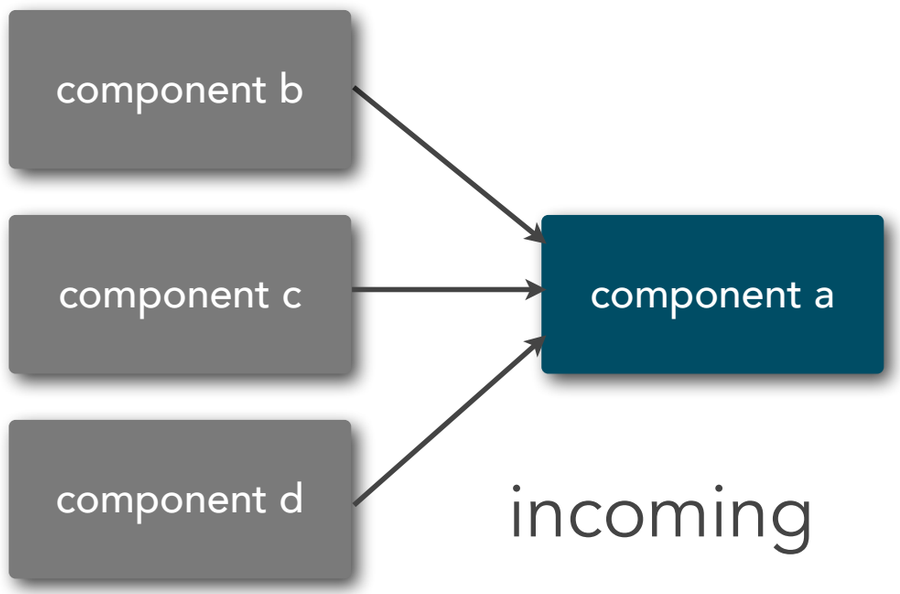
# Component Coupling



# Component Coupling

## afferent coupling

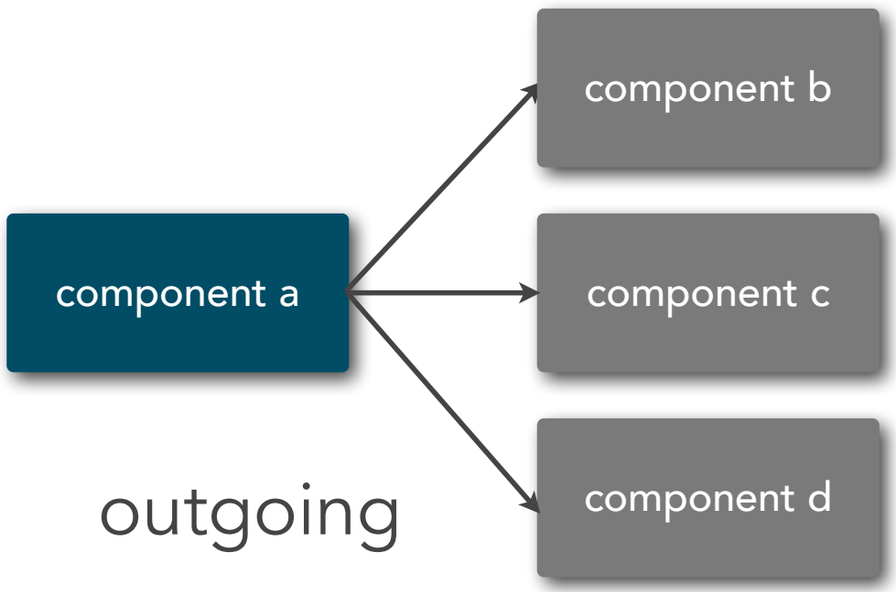
the degree to which other components are dependent on the target component



# Component Coupling

## efferent coupling

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



# Abstractness

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

where:

$m^a$ => abstract elements

$m^c$ => concrete elements

# Instability

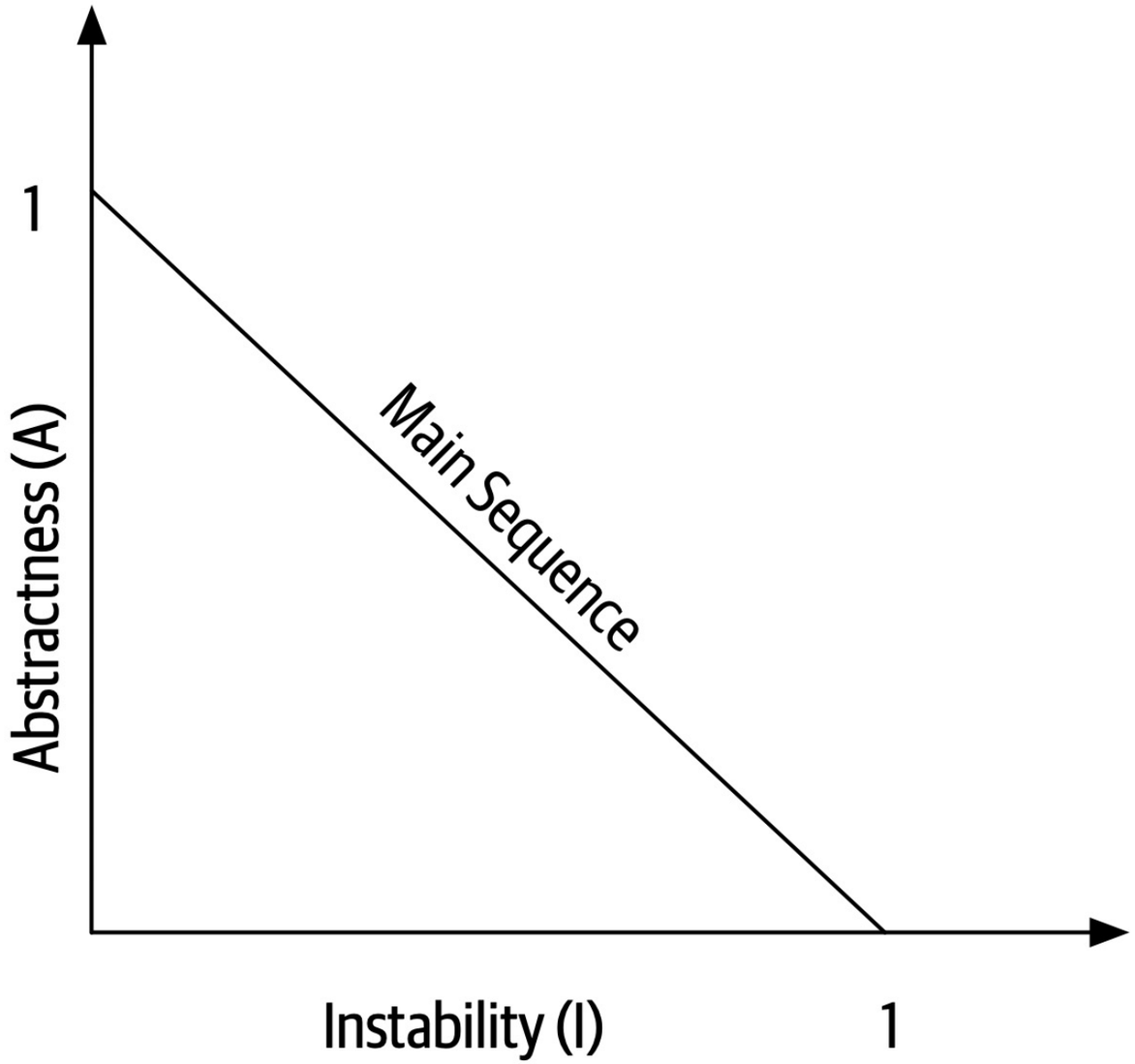
$$I = \frac{C^e}{C^e + C^a}$$

where:

$c^e$ => efferent coupling

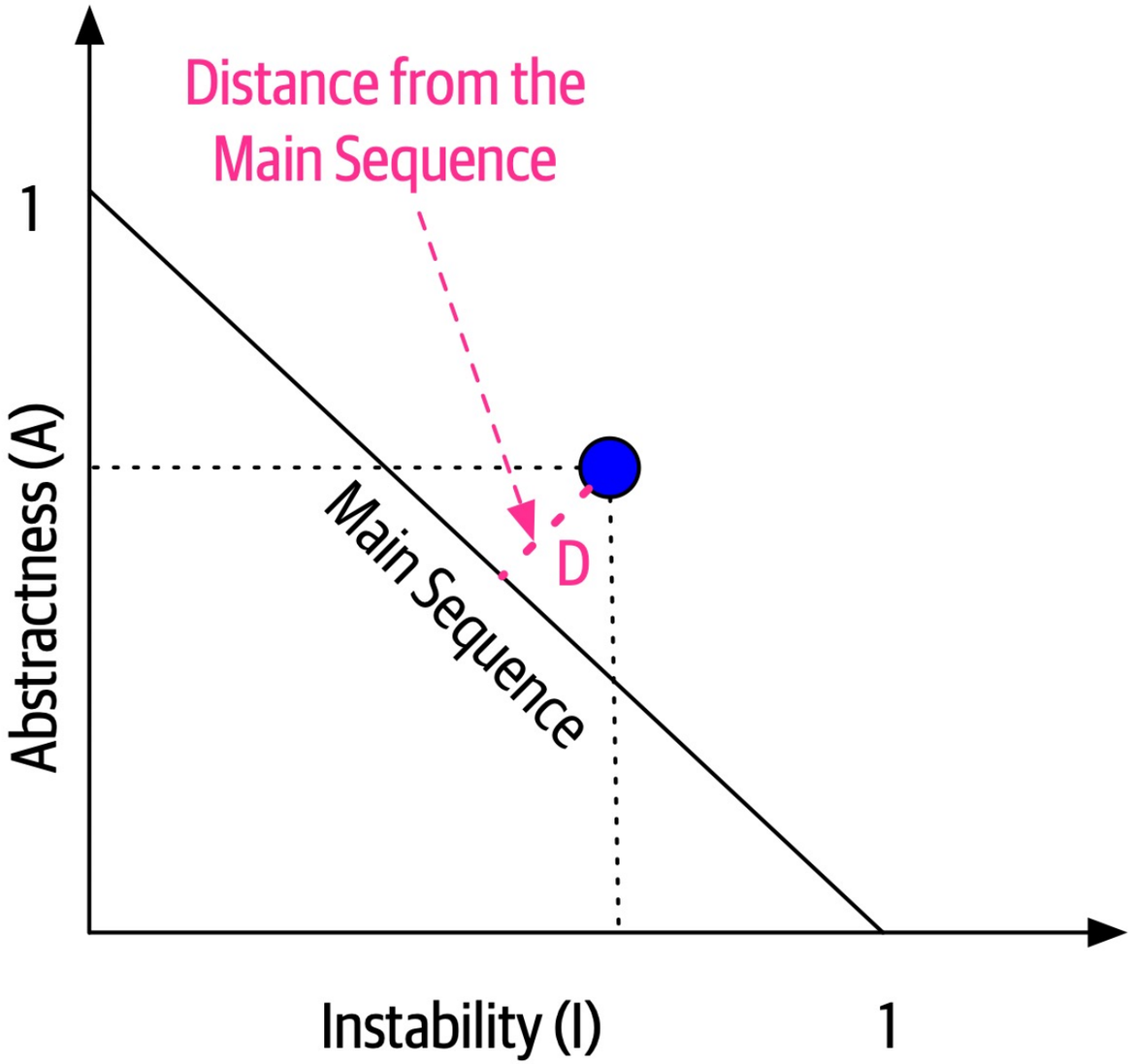
$c^a$ => afferent coupling

# Distance from the Main Sequence



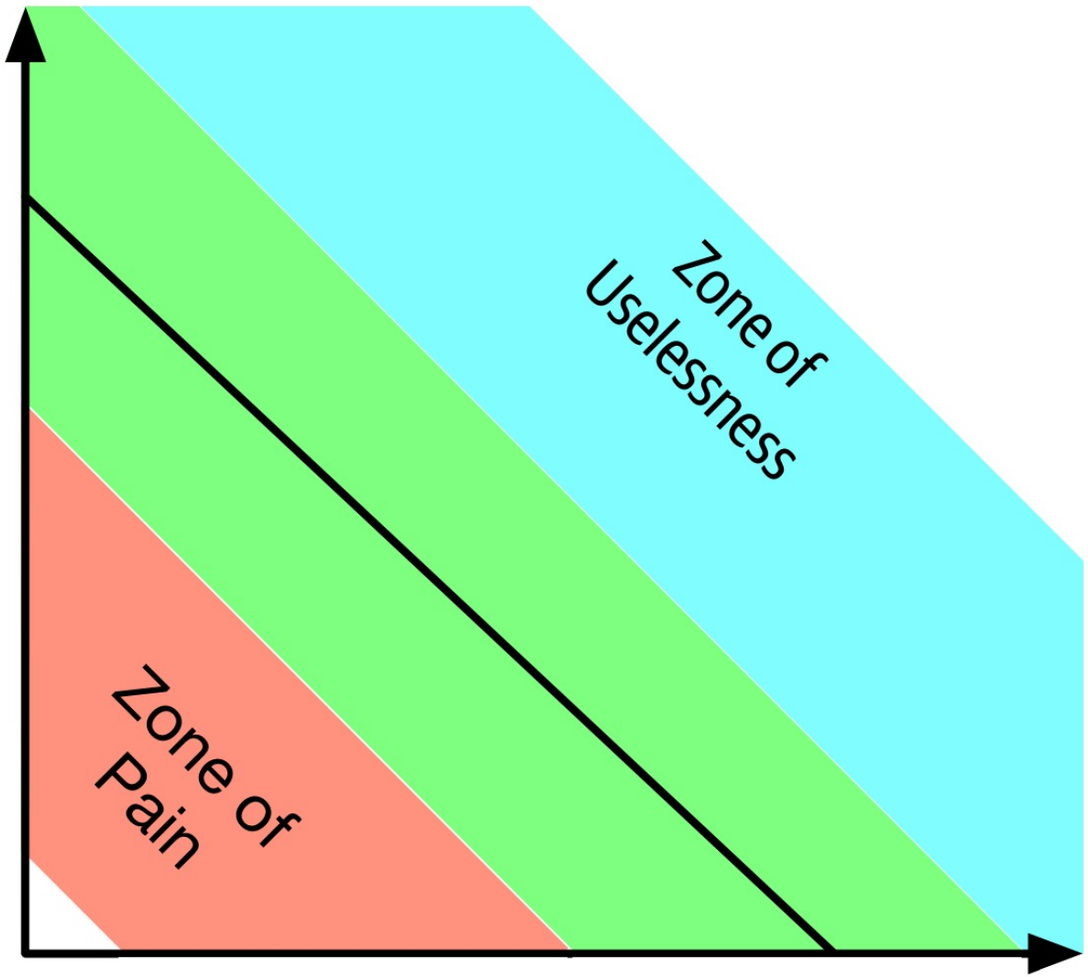

$$D = |A + I - 1|$$

# Distance from the Main Sequence



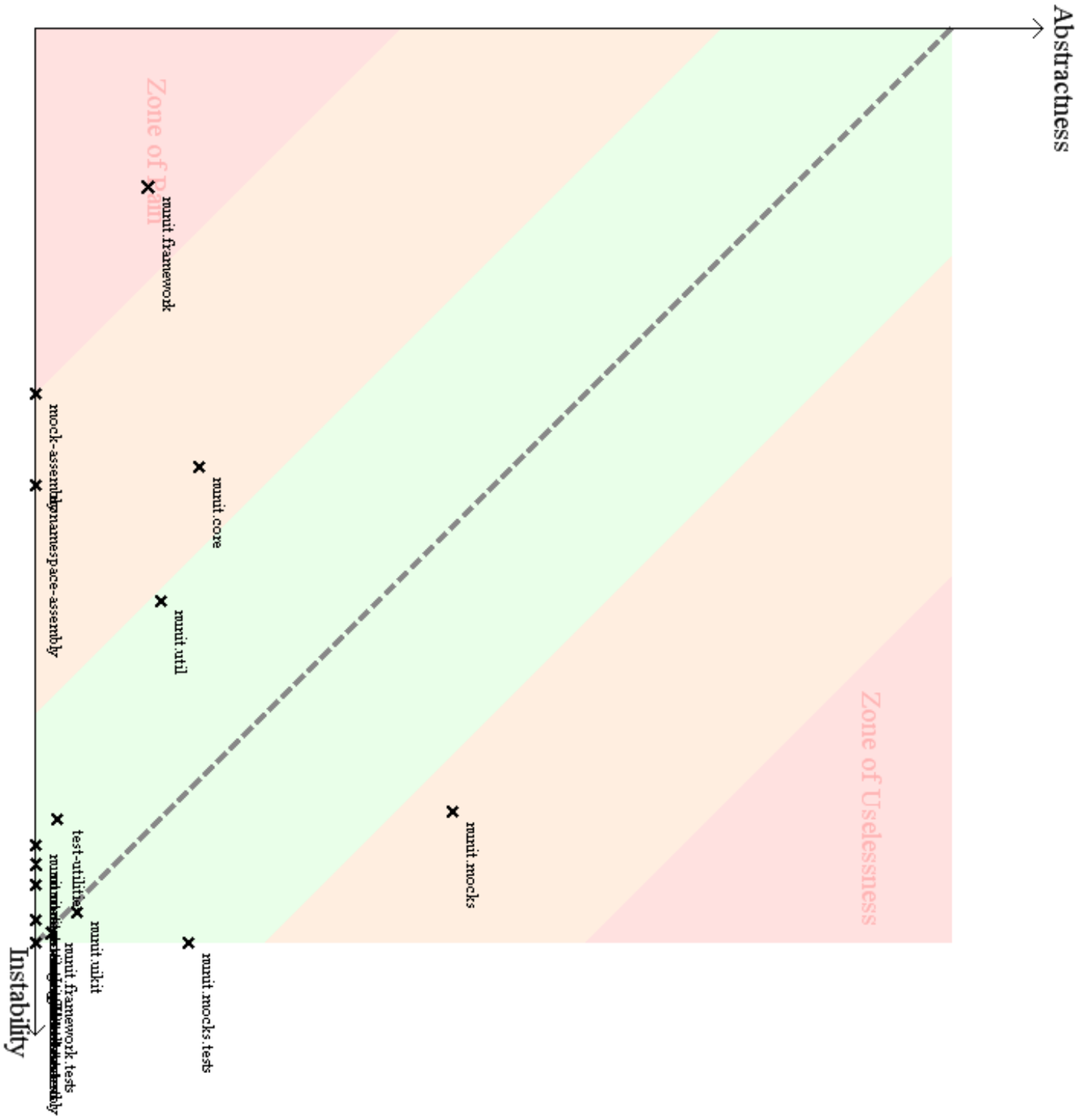

$$D = |A + I - 1|$$

# Distance from the Main Sequence

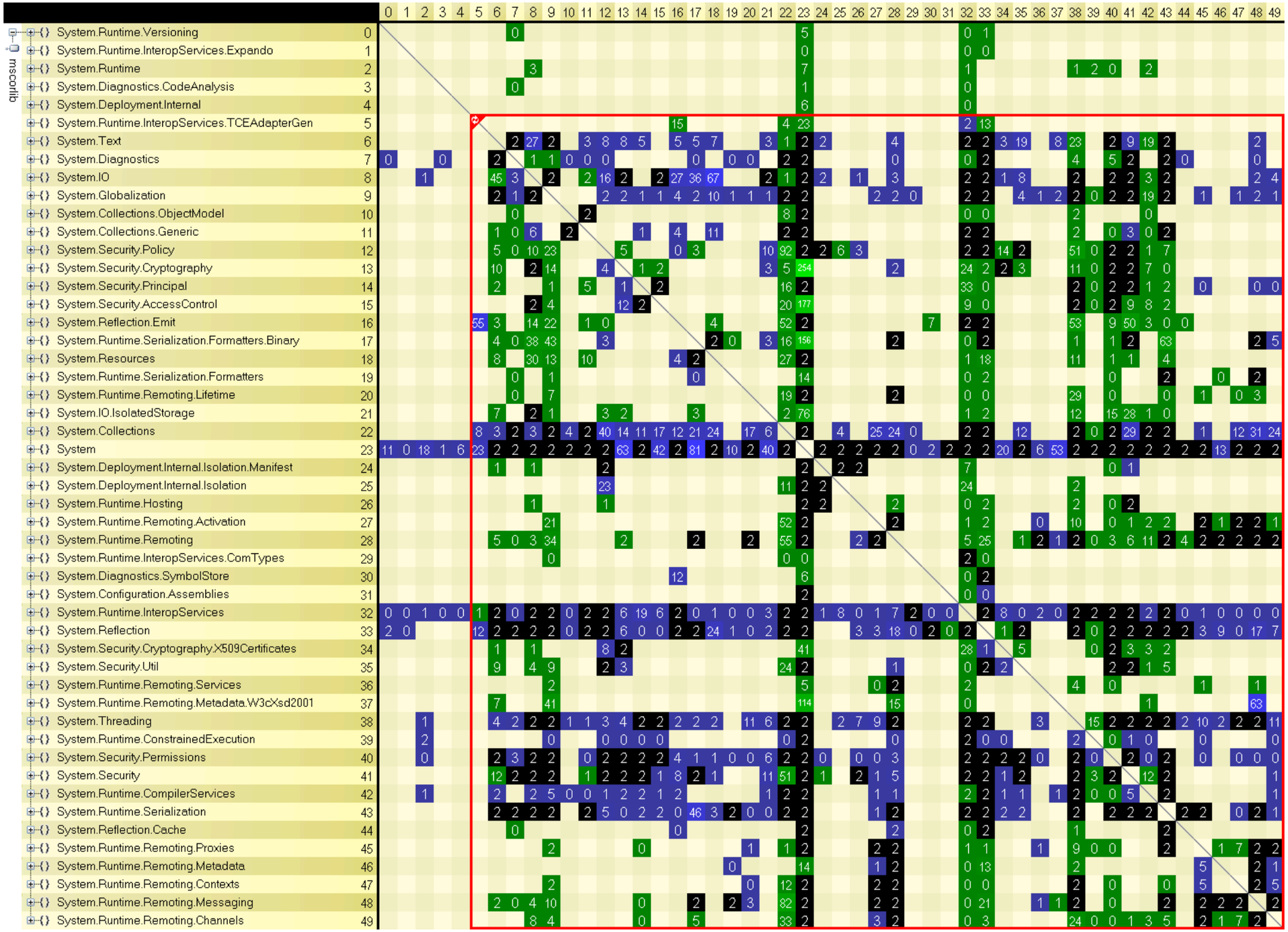



$$D = |A + I - 1|$$

# distance from the main sequence



# Dependency Metrics



# Component Cohesion

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


Metrics

# Chidamber & Kemerer Metrics

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

# Chidamber & Kemerer Metrics

✅ LCOM (Lack of Cohesion in Methods)

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

# 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.

# Chidamber & Kemerer Metrics

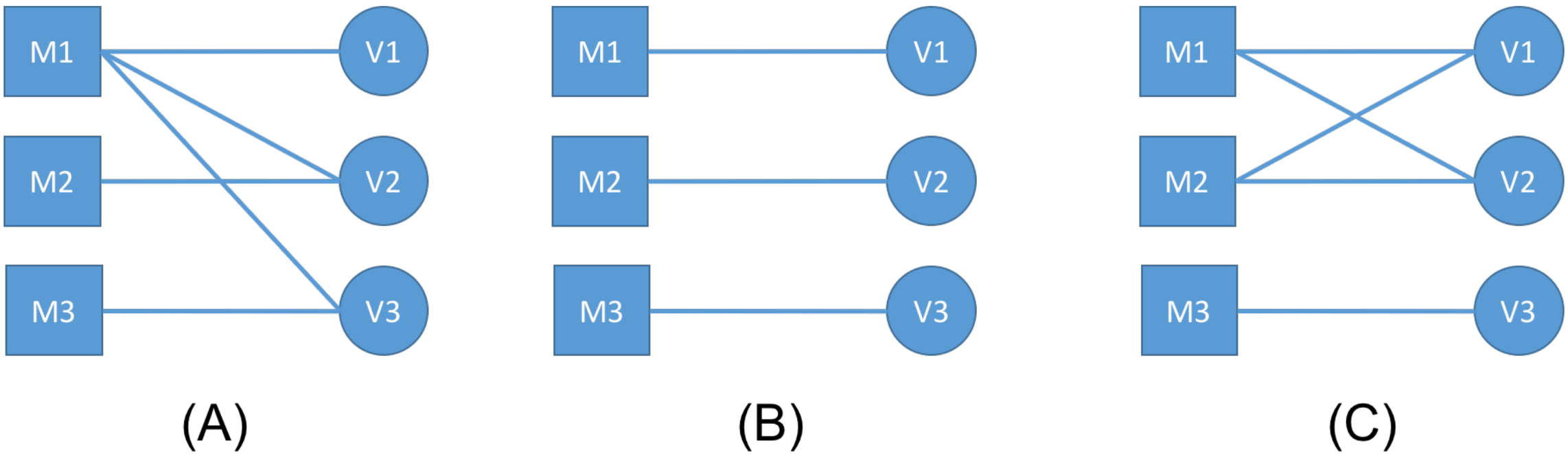
✓ LCOM (Lack of Cohesion in Methods)

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.

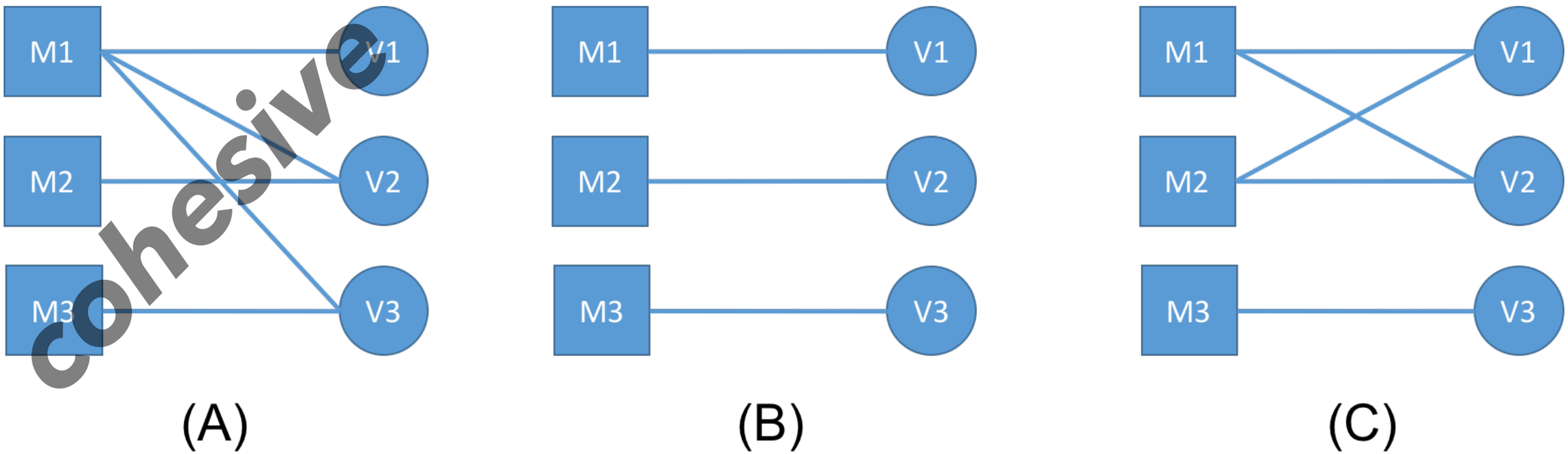
# Chidamber & Kemerer Metrics

✓ LCOM (Lack of Cohesion in Methods)



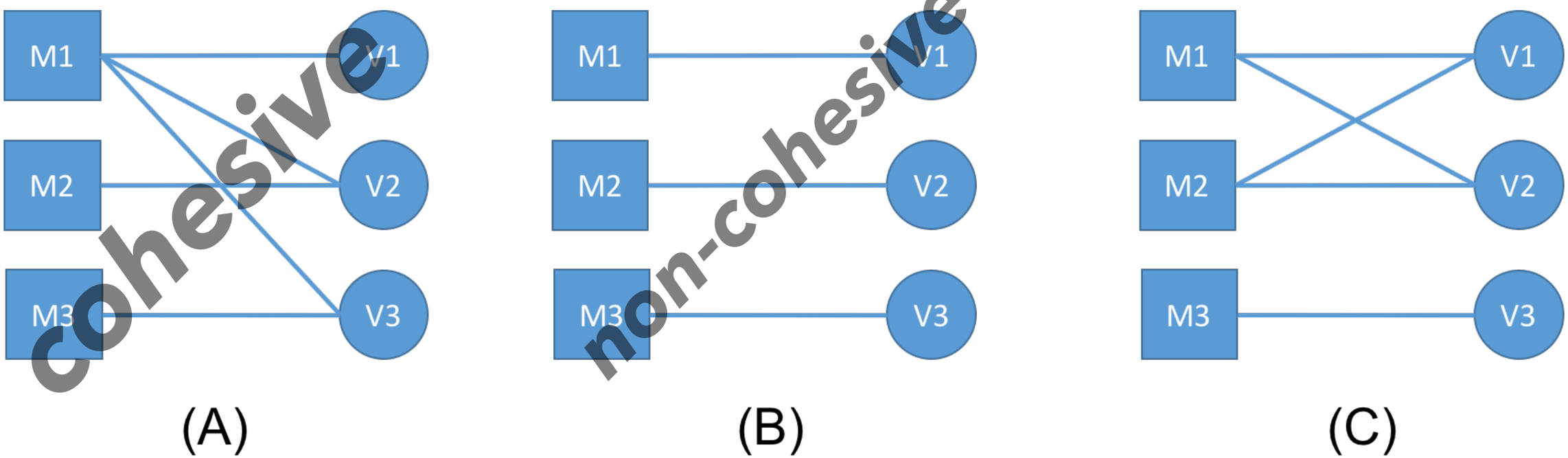
# Chidamber & Kemerer Metrics

✓ LCOM (Lack of Cohesion in Methods)



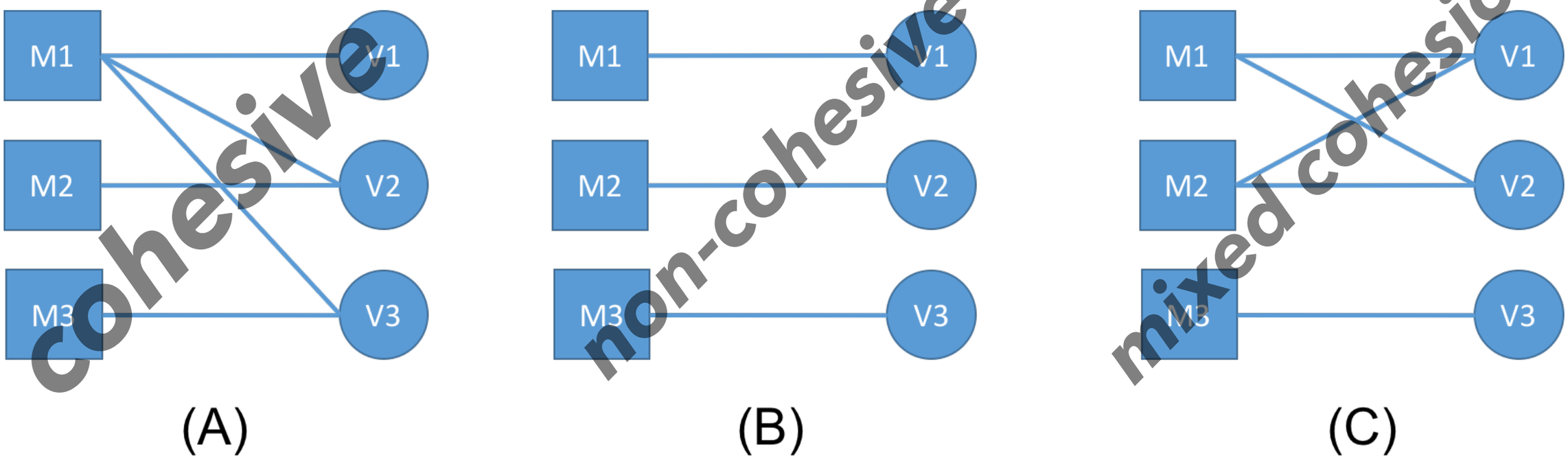
# Chidamber & Kemerer Metrics

✓ LCOM (Lack of Cohesion in Methods)

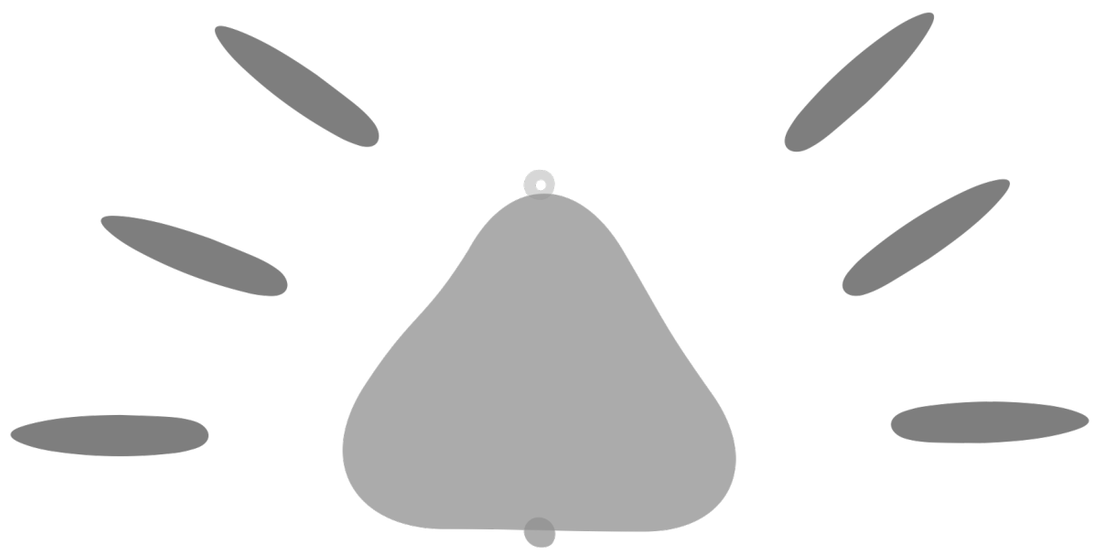


# Chidamber & Kemerer Metrics

✓ LCOM (Lack of Cohesion in Methods)

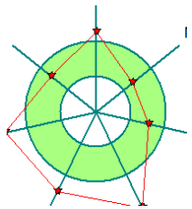


# Internal Quality Architecture Characteristics



# Metrics ∩ Architecture Characteristics

## architecture characteristics mapping



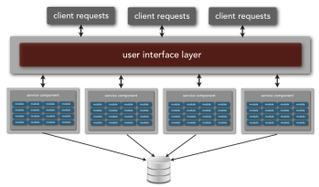
component size

complexity / WMC

coupling (CE, CA, CT)

inheritance depth (DIT)

percent comments




modularity

maintainability

testability

availability

deployment

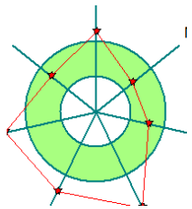
reliability

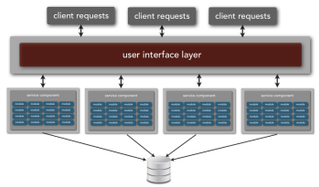
scalability

evolutionary / migration

# Metrics ∩ Architecture Characteristics

## architecture characteristics mapping






component size

complexity / WMC

coupling (CE, CA, CT)

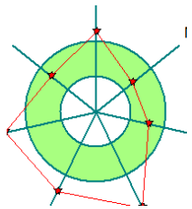
inheritance depth (DIT)

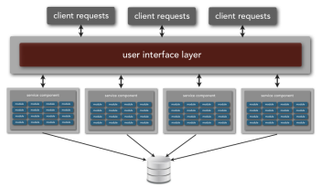
percent comments

- 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

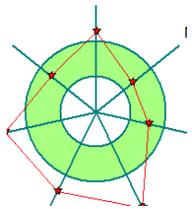
reliability

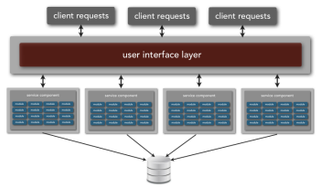
scalability

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

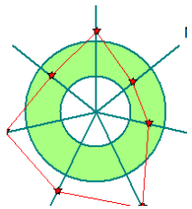
reliability

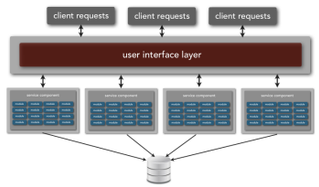
scalability

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

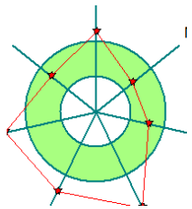
reliability

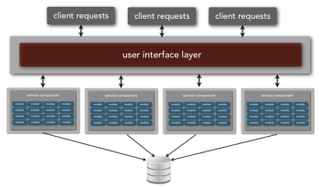
scalability

evolutionary / migration

# Metrics ∩ Architecture Characteristics

## architecture characteristics mapping






component size

complexity / WMC

coupling (CE, CA, CT)

inheritance depth (DIT)

percent comments

modularity

maintainability

testability

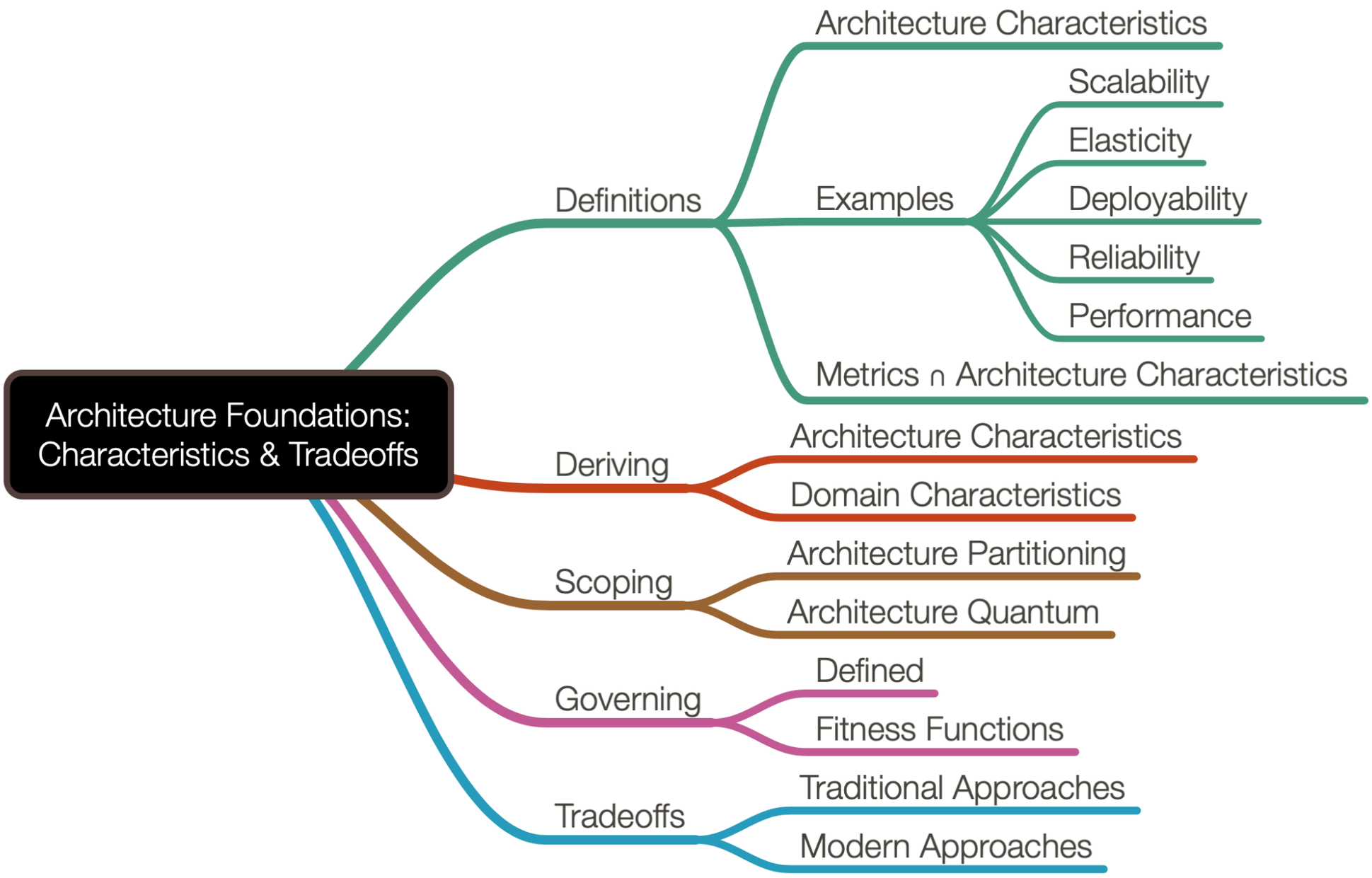
availability

deployment

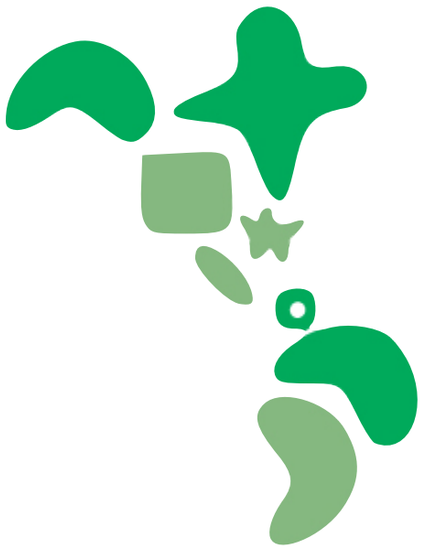
reliability

scalability

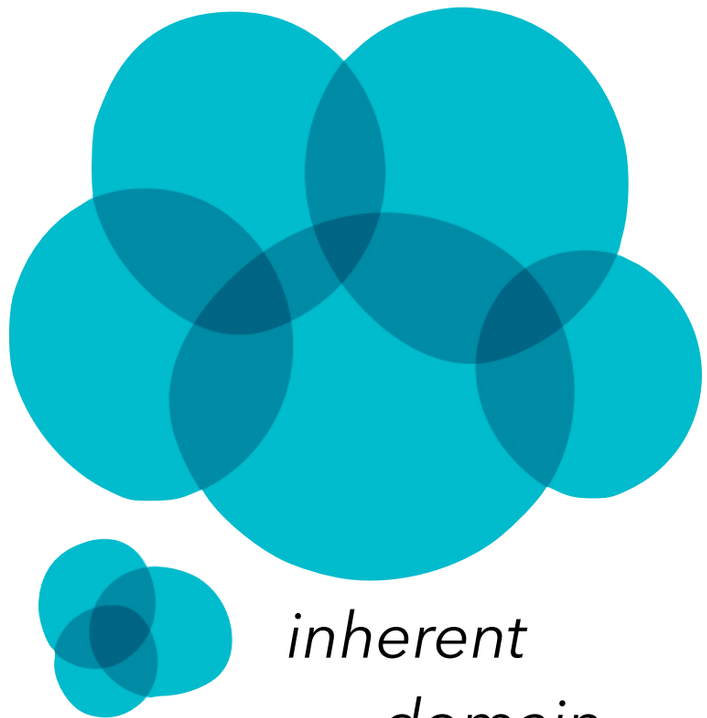
evolutionary / migration


Architecture Foundations: Characteristics & Tradeoffs
Definitions
Architecture Characteristics
Examples
Scalability
Elasticity
Deployability
Reliability
Performance
Metrics ∩ Architecture Characteristics
Deriving
Architecture Characteristics
Domain Characteristics
Scoping
Architecture Partitioning
Architecture Quantum
Governing
Defined
Fitness Functions
Tradeoffs
Traditional Approaches
Modern Approaches

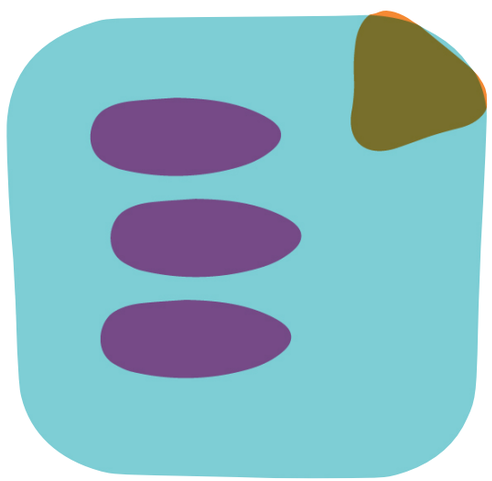
# Three Sources



*environmental knowledge*

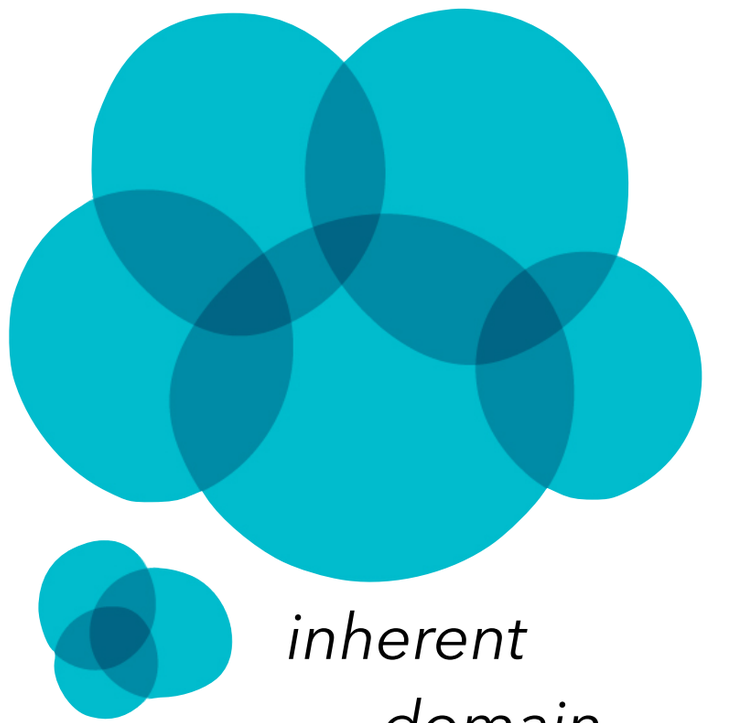


*inherent domain knowledge*

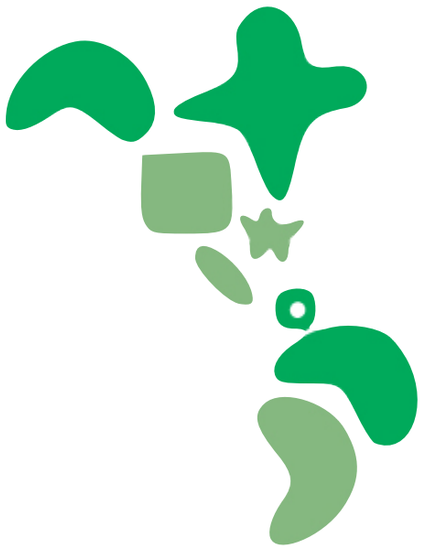


*explicit requirements*

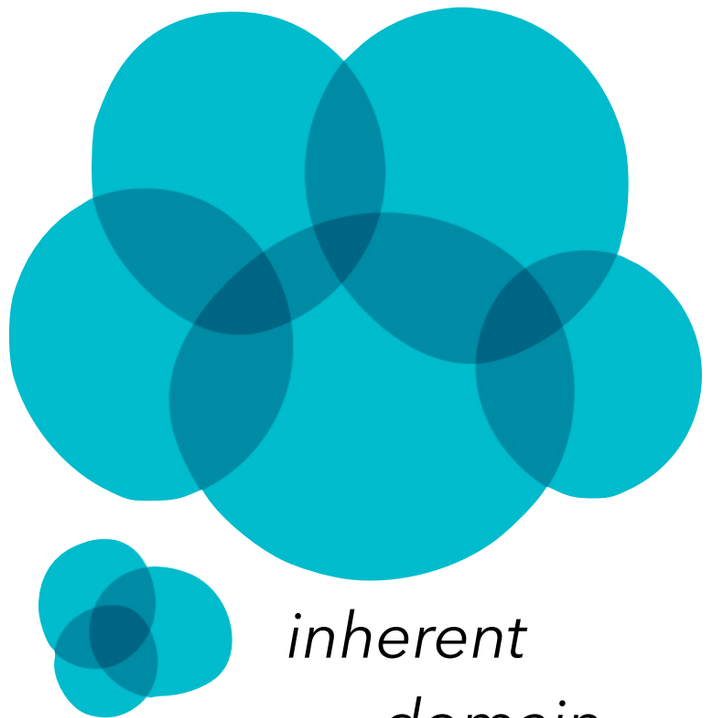
# Three Sources



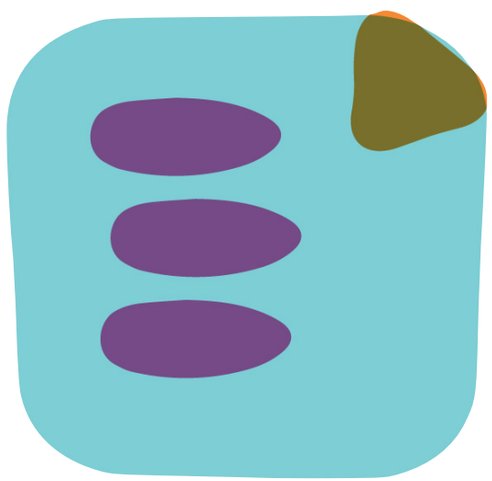
# Three Sources



*environmental knowledge*



*inherent domain knowledge*



*explicit requirements*

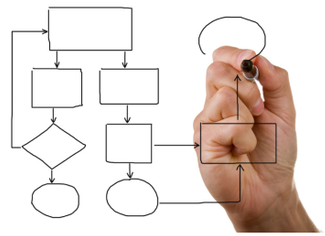
# Three Sources

*environmental*

*knowledge*

# Architecture Characteristics

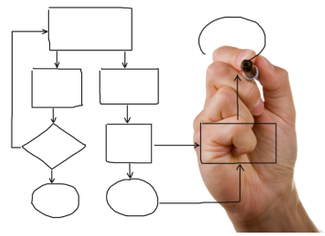
"our business is constantly changing to meet new demands of the marketplace"



— *extensibility*
— *maintainability*
— *agility*
— *modularity*

# Architecture Characteristics

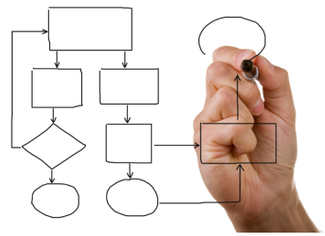
"due to new regulatory requirements, it is imperative that we complete end-of-day processing in time"



— *performance*
— *scalability*
— *availability*
— *reliability*

# Architecture Characteristics

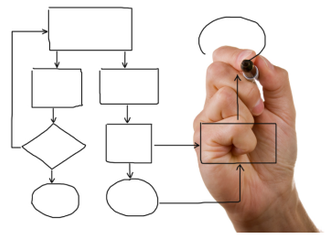
"we need faster time to market to remain competitive"



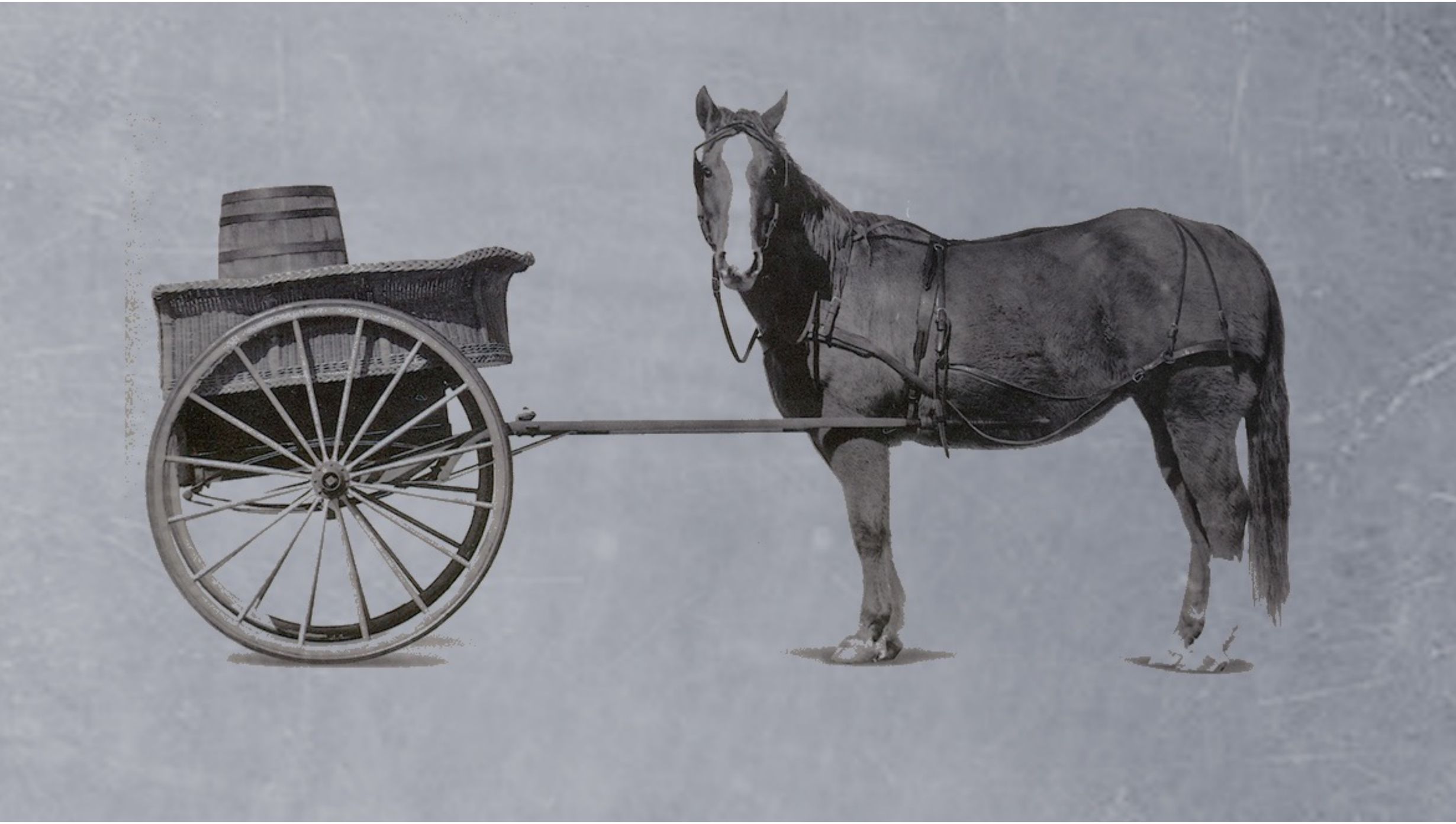
— *maintainability*
— *agility*
— *modularity*
— *deployability*
— *testability*

# Architecture Characteristics

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

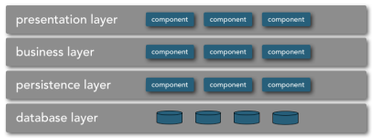


— *scalability*
— *extensibility*
— *openness*
— *standards-based*



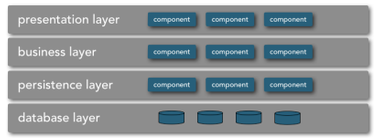
# Architecture Characteristics

feasibility

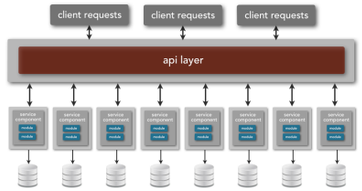


# Architecture Characteristics

feasibility

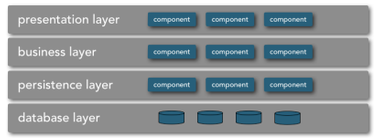



agility

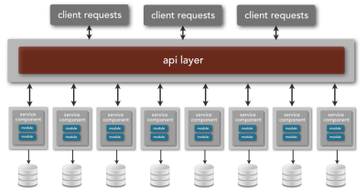


# Architecture Characteristics

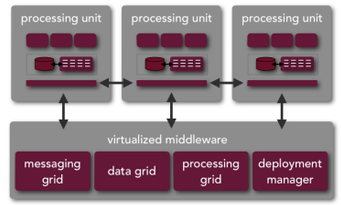
feasibility →




agility →

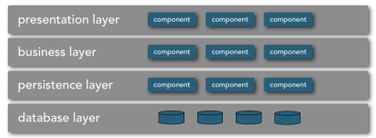



elasticity →

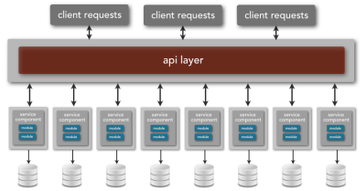


# Architecture Characteristics

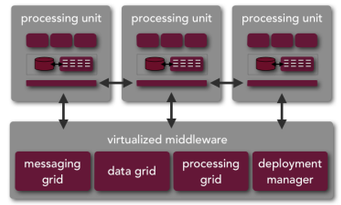
feasibility



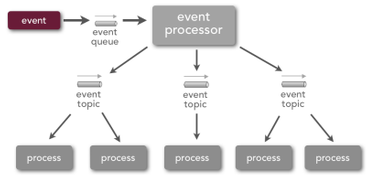

agility



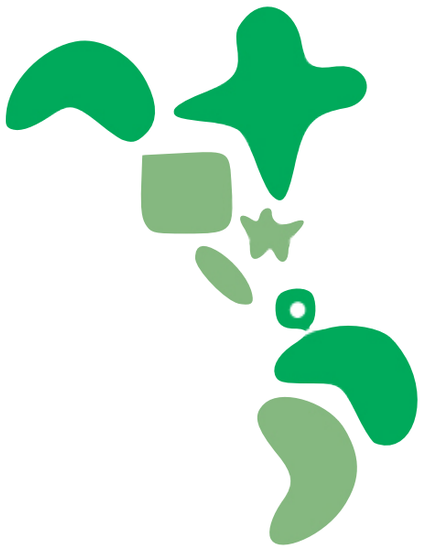

elasticity



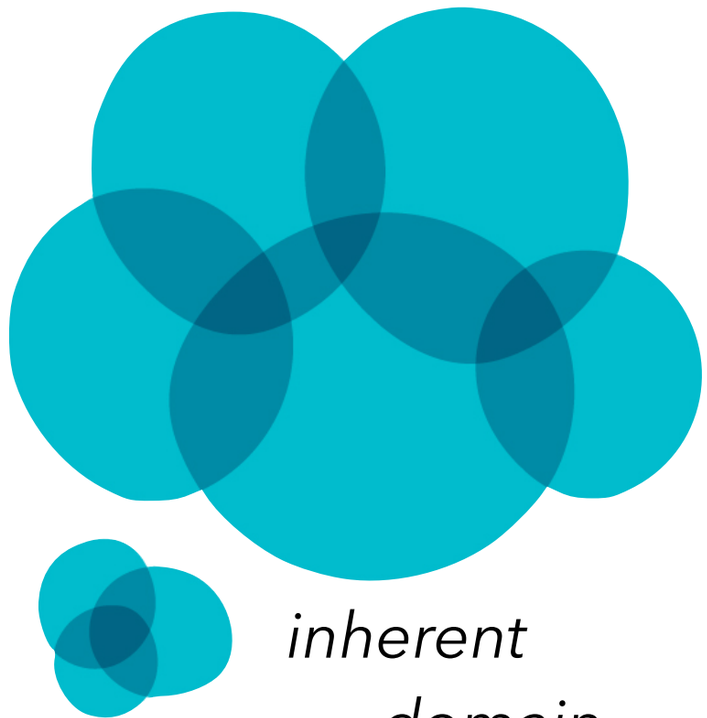

scalability



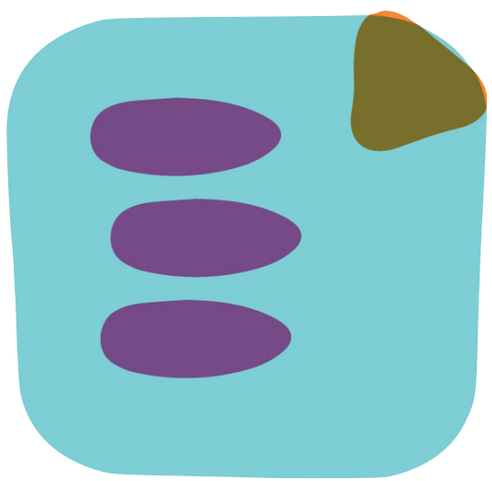
# Three Sources



*environmental knowledge*



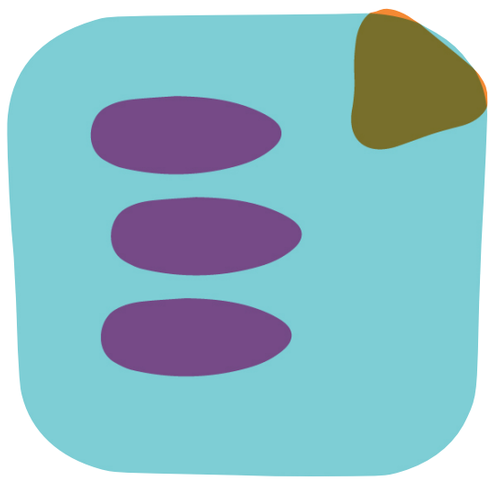
*inherent domain knowledge*



*explicit requirements*

# Three Sources

*explicit*
*requirements*



**Your Architectural Kata is...**

# Going Going Gone!

An auction company wants to take their auctions online to a nationwide scale--customers choose the auction to participate in, wait until the auction begins, then bid during the live auction as if they were there in the room, with the auctioneer.

- **Users**: scale up to hundreds of participants (per auction), potentially up to thousands of participants, and as many simultaneous auctions as possible
- **Requirements**:
  - bidders can see a live video stream of the auction and see all bids as they occur
  - auctions must be as real-time as possible
  - both online and live bids must be received in the order in which they are placed
  - bidders register with credit card; system automatically charges card if bidder wins
  - participants must be tracked via a reputation index
- **Additional Context:**
  - auction company is expanding aggressively by merging with smaller competitors
  - if nationwide auction is a success, replicate the model overseas
  - budget is not constrained--this is a strategic direction
  - company just exited a lawsuit where they settled a suit alleging fraud

**Your Architectural Kata is...**

# Going Going Gone!

An auction company wants to take their ***auctions online*** to a nationwide scale--customers choose the auction to participate in, wait until the auction begins, then bid during the live auction as if they were there in the room, with the auctioneer.

- **Users**: scale up to hundreds of participants (per auction), potentially up to thousands of participants, and as many simultaneous auctions as possible
- **Requirements**:
  - bidders can see a live video stream of the auction and see all bids as they occur
  - ***auctions must be as real-time as possible***
  - both online and live bids must be received in the order in which they are placed
  - bidders register with credit card; system automatically charges card if bidder wins
  - participants must be tracked via a reputation index
- **Additional Context:**
  - auction company is expanding aggressively by merging with smaller competitors
  - if nationwide auction is a success, replicate the model overseas
  - budget is not constrained--this is a strategic direction
  - company just exited a lawsuit where they settled a suit alleging fraud

availability reliability performance

**Your Architectural Kata is...**

# Going Going Gone!

An auction company wants to take their auctions online to a ***nationwide scale***--customers choose the auction to participate in, wait until the auction begins, then bid during the live auction as if they were there in the room, with the auctioneer.

- **Users**: ***scale up to hundreds of participants (per auction), potentially up to thousands of participants, and as many simultaneous auctions as possible***
- **Requirements**:
  - bidders can see a live video stream of the auction and see all bids as they occur
  - auctions must be as real-time as possible
  - both online and live bids must be received in the order in which they are placed
  - bidders register with credit card; system automatically charges card if bidder wins
  - participants must be tracked via a reputation index
- **Additional Context:**
  - auction company is expanding aggressively by merging with smaller competitors
  - ***if nationwide auction is a success, replicate the model overseas***
  - budget is not constrained--this is a strategic direction
  - company just exited a lawsuit where they settled a suit alleging fraud

availability reliability performance scalability

**Your Architectural Kata is...**

# Going Going Gone!

An auction company wants to take their auctions online to a nationwide scale--customers choose the auction to participate in, wait until the auction begins, then bid during the live auction as if they were there in the room, with the auctioneer.

- **Users**: ***scale up to hundreds of participants (per auction), potentially up to thousands of participants, and as many simultaneous auctions as possible***
- **Requirements**:
  - bidders can see a live video stream of the auction and see all bids as they occur
  - auctions must be as real-time as possible
  - both online and live bids must be received in the order in which they are placed
  - bidders register with credit card; system automatically charges card if bidder wins
  - participants must be tracked via a reputation index
- **Additional Context:**
  - auction company is expanding aggressively by merging with smaller competitors
  - if nationwide auction is a success, replicate the model overseas
  - budget is not constrained--this is a strategic direction
  - company just exited a lawsuit where they settled a suit alleging fraud

availability reliability performance scalability elasticity

**Your Architectural Kata is...**

# Going Going Gone!

An auction company wants to take their auctions online to a nationwide scale--customers choose the auction to participate in, wait until the auction begins, then bid during the live auction as if they were there in the room, with the auctioneer.

- **Users**: scale up to hundreds of participants (per auction), potentially up to thousands of participants, and as many simultaneous auctions as possible
- **Requirements**:
  - bidders can see a live video stream of the auction and see all bids as they occur
  - auctions must be as real-time as possible
  - both online and live bids must be received in the order in which they are placed
  - bidders register with credit card; system automatically charges card if bidder wins
  - ***participants must be tracked via a reputation index***
- **Additional Context:**
  - auction company is expanding aggressively by merging with smaller competitors
  - if nationwide auction is a success, replicate the model overseas
  - budget is not constrained--this is a strategic direction
  - company just exited a lawsuit where they settled a suit alleging fraud

availability reliability performance scalability elasticity

**Your Architectural Kata is...**

# Going Going Gone!

An auction company wants to take their auctions online to a nationwide scale--customers choose the auction to participate in, wait until the auction begins, then bid during the live auction as if they were there in the room, with the auctioneer.

- **Users**: scale up to hundreds of participants (per auction), potentially up to thousands of participants, and as many simultaneous auctions as possible
- **Requirements**:
  - bidders can see a live video stream of the auction and see all bids as they occur
  - auctions must be as real-time as possible
  - both online and live bids must be received in the order in which they are placed
  - ***bidders register with credit card; system automatically charges card if bidder wins***
  - participants must be tracked via a reputation index
- **Additional Context:**
  - auction company is expanding aggressively by merging with smaller competitors
  - if nationwide auction is a success, replicate the model overseas
  - budget is not constrained--this is a strategic direction
  - company just exited a lawsuit where they settled a suit alleging fraud

availability reliability performance scalability elasticity (security)

**Your Architectural Kata is...**

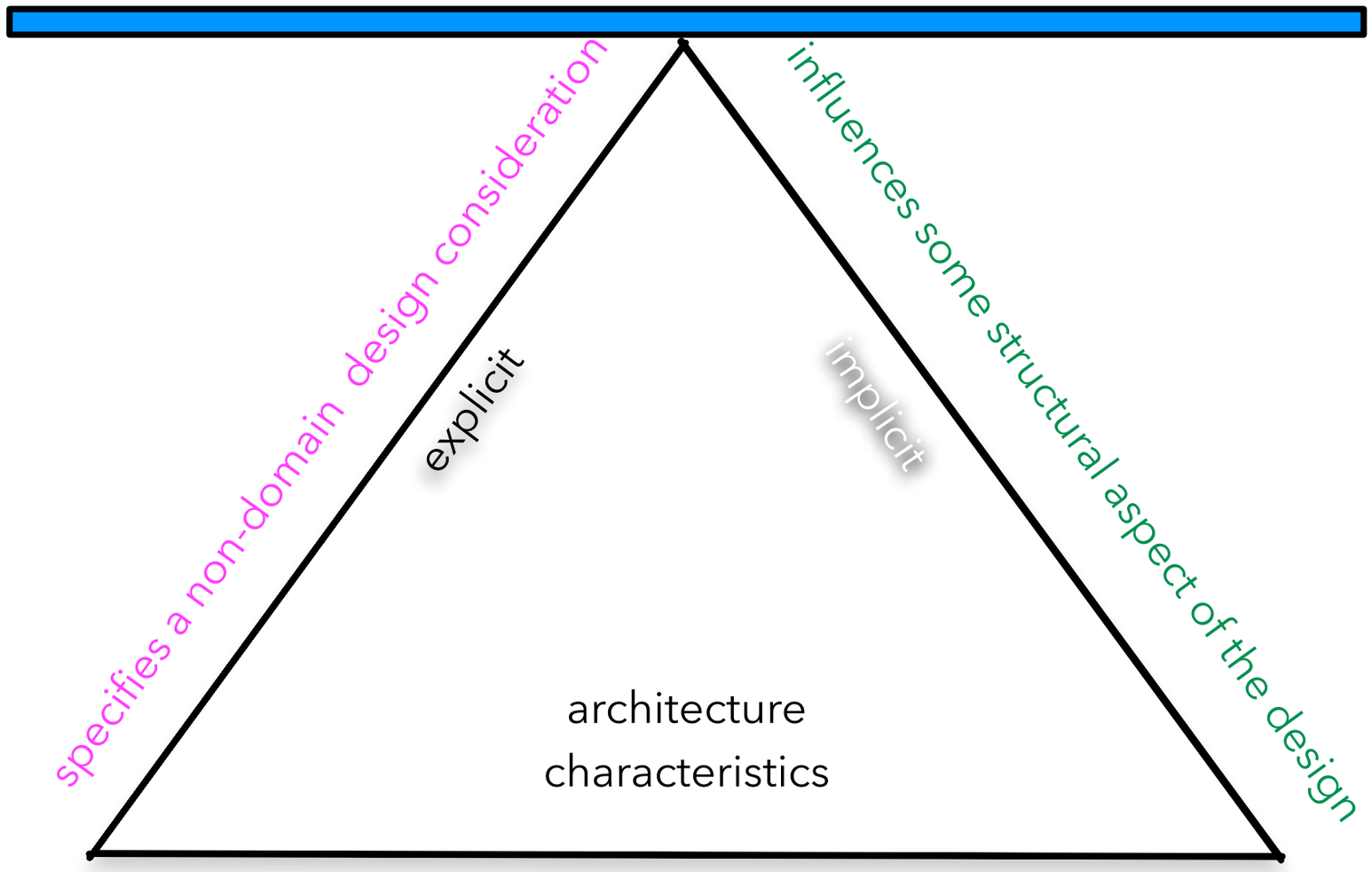
# Going Going Gone!

An auction company wants to take their auctions online to a nationwide scale--customers choose the auction to participate in, wait until the auction begins, then bid during the live auction as if they were there in the room, with the auctioneer.

- **Users**: scale up to hundreds of participants (per auction), potentially up to thousands of participants, and as many simultaneous auctions as possible
- **Requirements**:
  - bidders can see a live video stream of the auction and see all bids as they occur
  - auctions must be as real-time as possible
  - both online and live bids must be received in the order in which they are placed
  - bidders register with credit card; system automatically charges card if bidder wins
  - participants must be tracked via a reputation index
- **Additional Context:**
  - auction company is expanding aggressively by merging with smaller competitors
  - if nationwide auction is a success, replicate the model overseas
  - budget is not constrained--this is a strategic direction
  - company just exited a lawsuit where they settled a suit alleging fraud

availability reliability performance scalability elasticity (security)

design



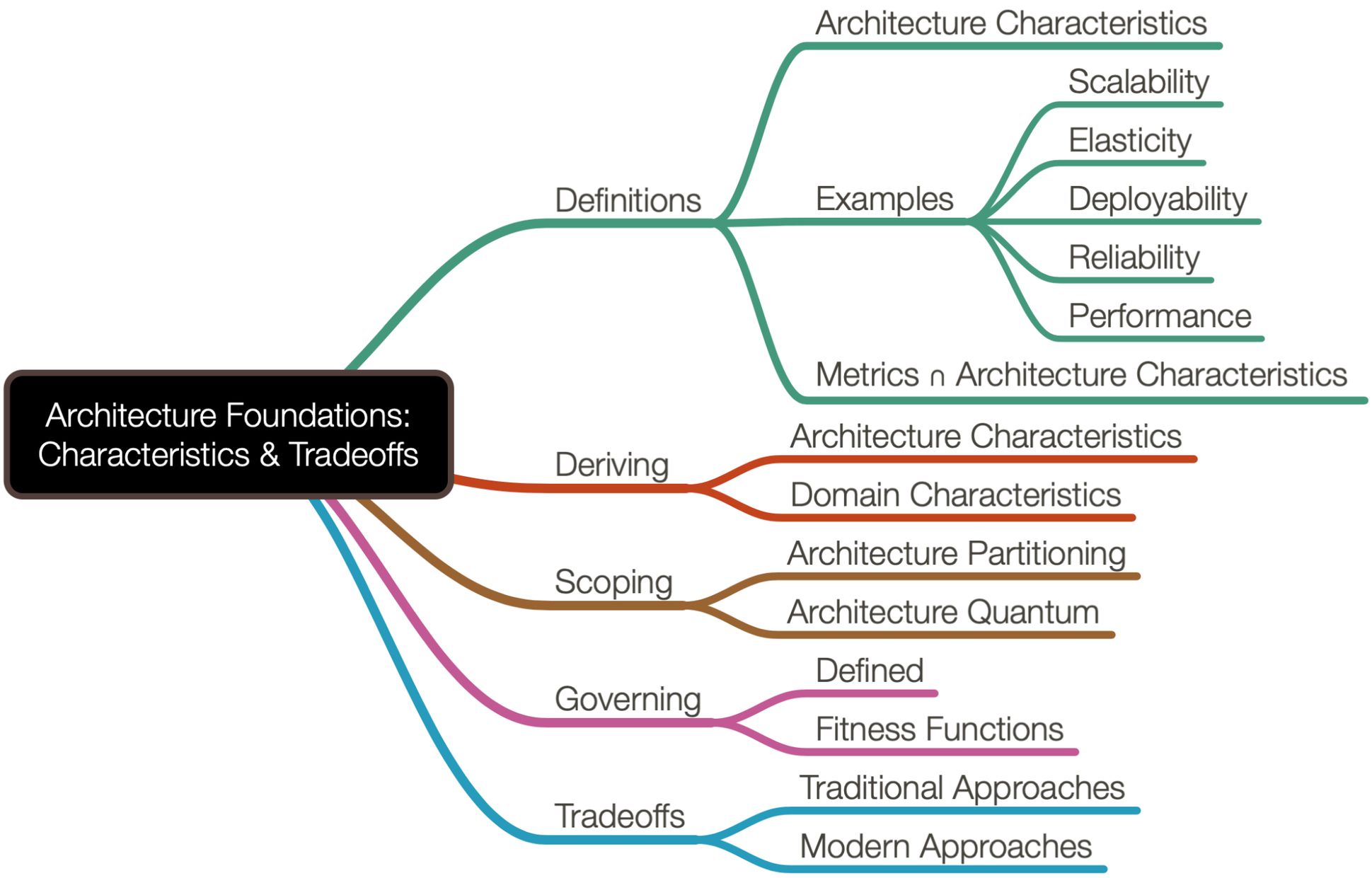

critical or important to application success

# Software Architecture by Example

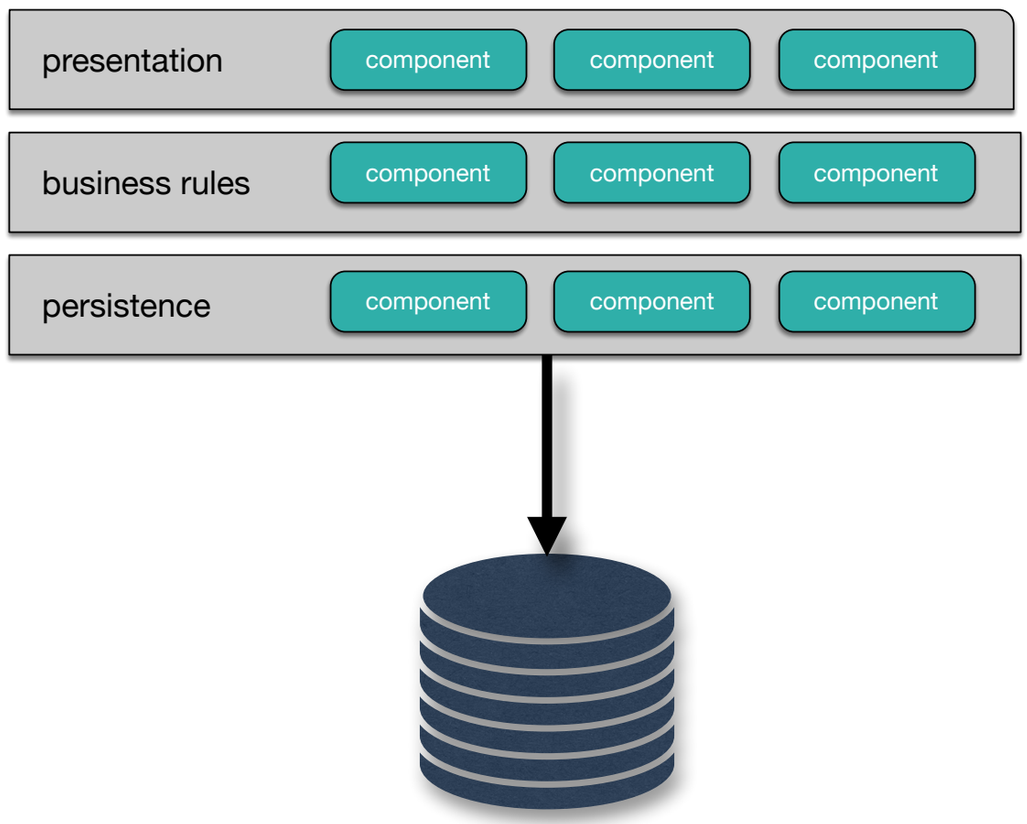
**Mark Richards**
**Independent Consultant**
Hands-on Software Architect, Published Author
Founder, DeveloperToArchitect.com
http://www.wmrichards.com
@markrichardssa

**Neal Ford**
**ThoughtWorks**
Director / Software Architect / Meme Wrangler
http://www.nealford.com
@neal4d

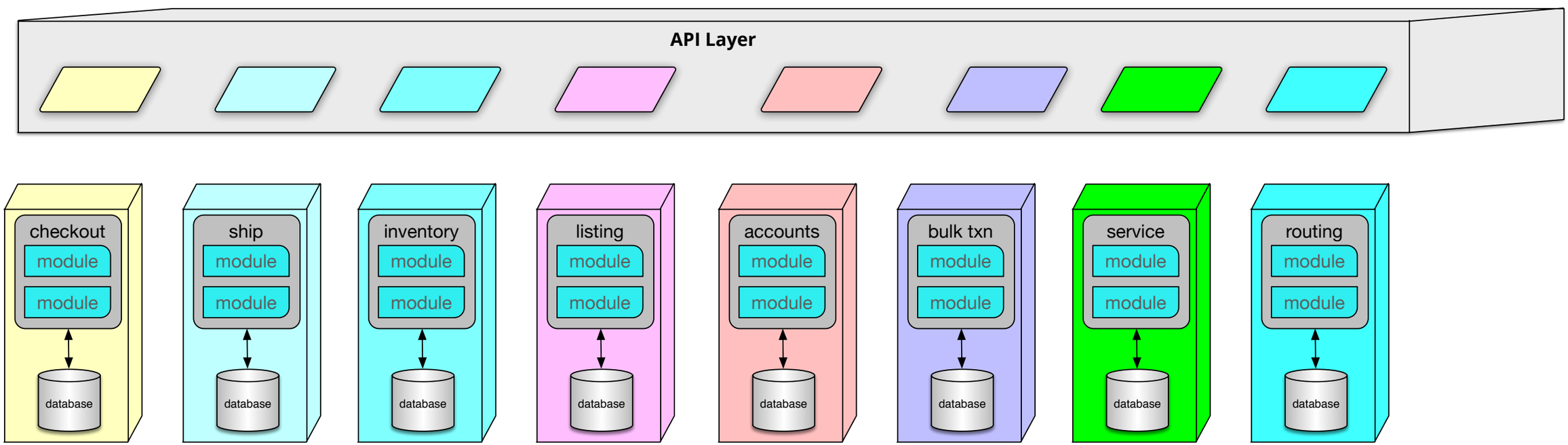
O'REILLY®


Architecture Foundations: Characteristics & Tradeoffs
Definitions
Architecture Characteristics
Examples
Scalability
Elasticity
Deployability
Reliability
Performance
Metrics ∩ Architecture Characteristics
Deriving
Architecture Characteristics
Domain Characteristics
Scoping
Architecture Partitioning
Architecture Quantum
Governing
Defined
Fitness Functions
Tradeoffs
Traditional Approaches
Modern Approaches

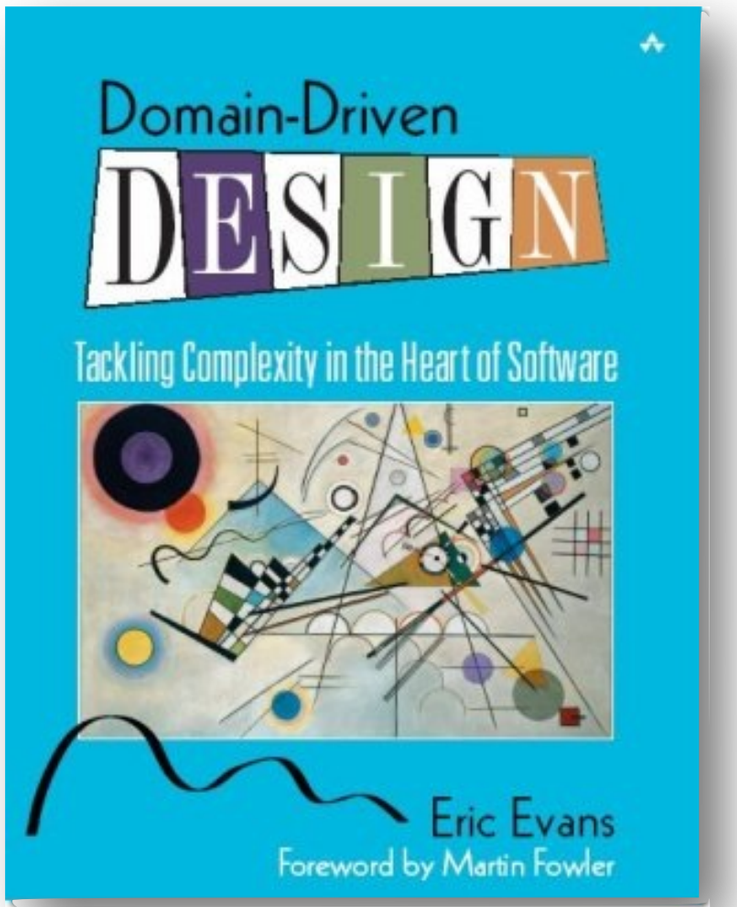
# Old Axiom

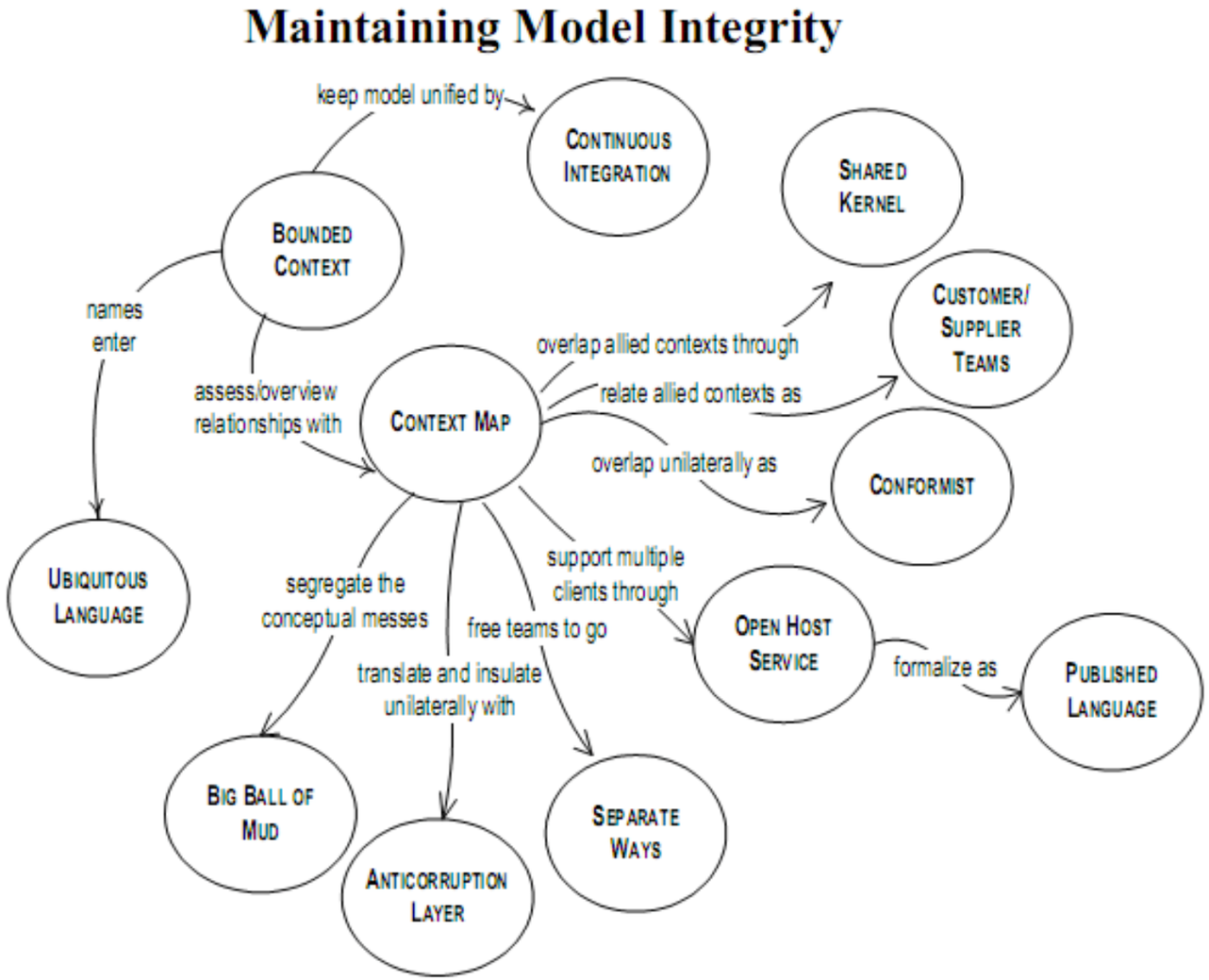

presentation
component
component
component
business rules
component
component
component
persistence
component
component
component

# Current Reality



# DDD => Architecture Quantum

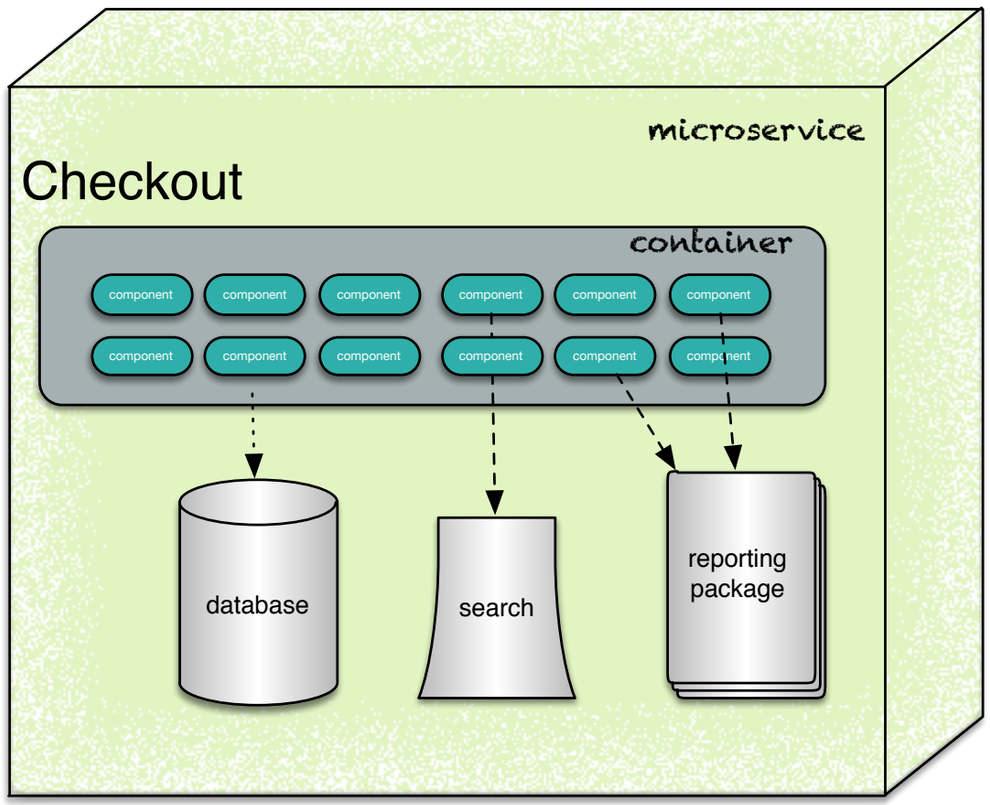


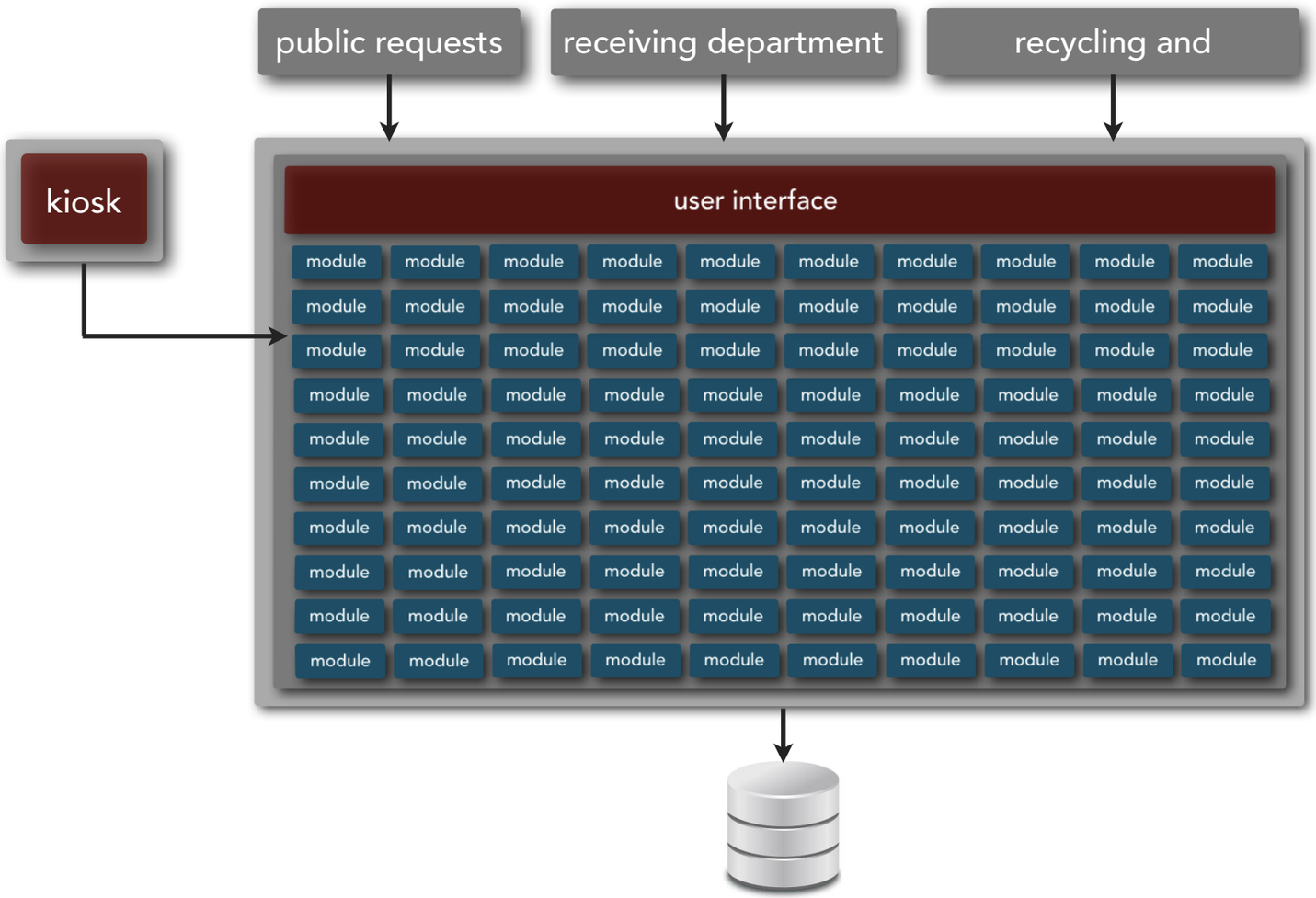
# Architecture Quantum

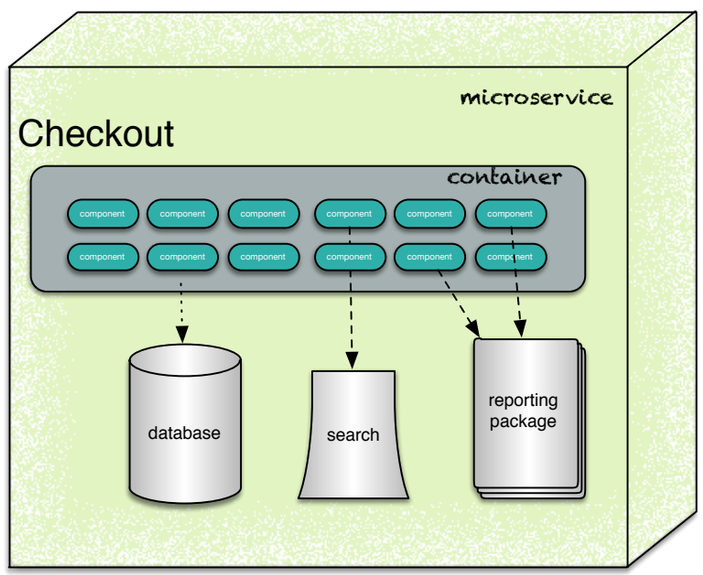
independently deployable artifact with high functional cohesion and synchronous coupling

# Independently deployable

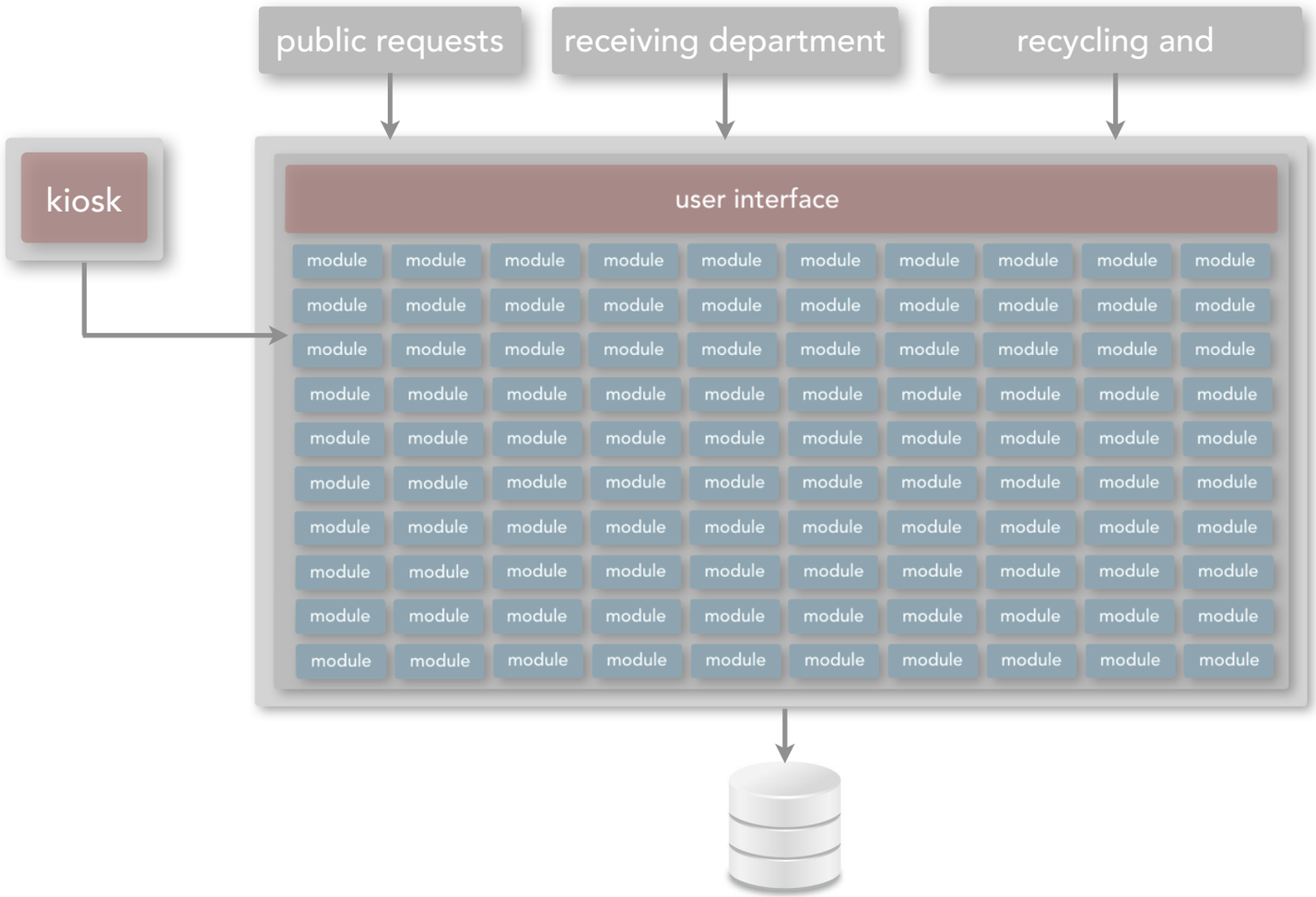


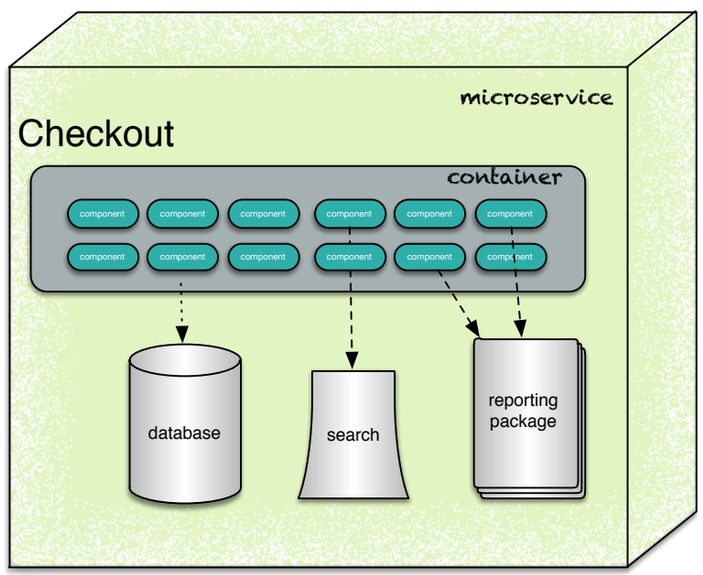
# High functional cohesion



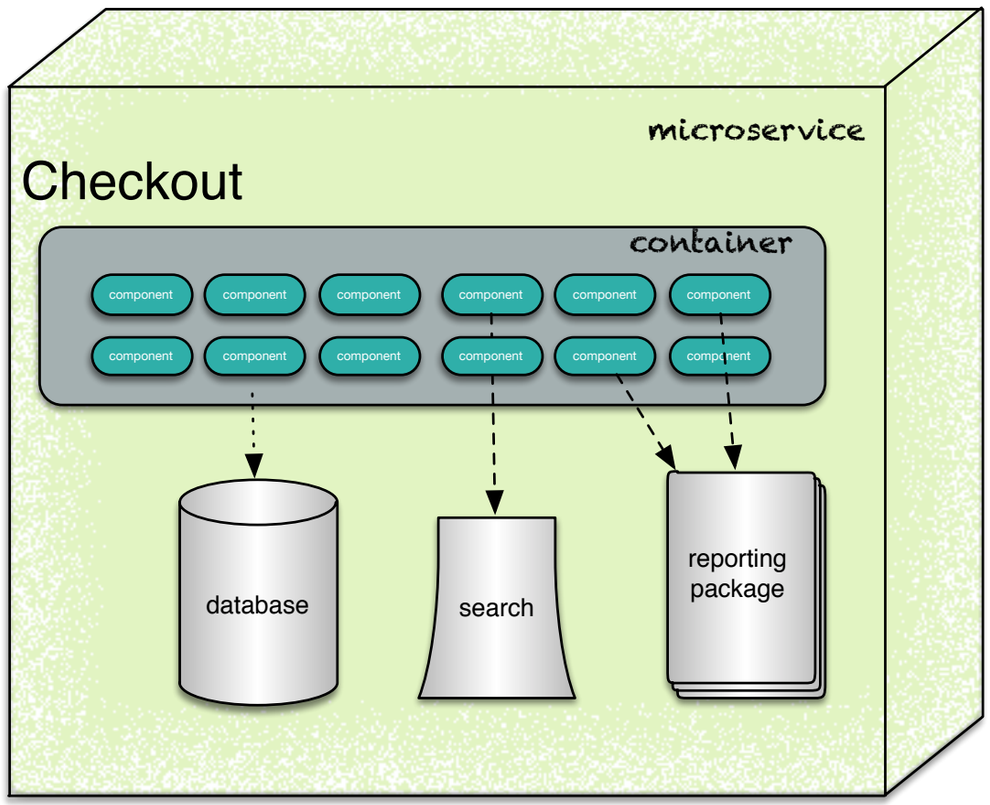



# High functional cohesion

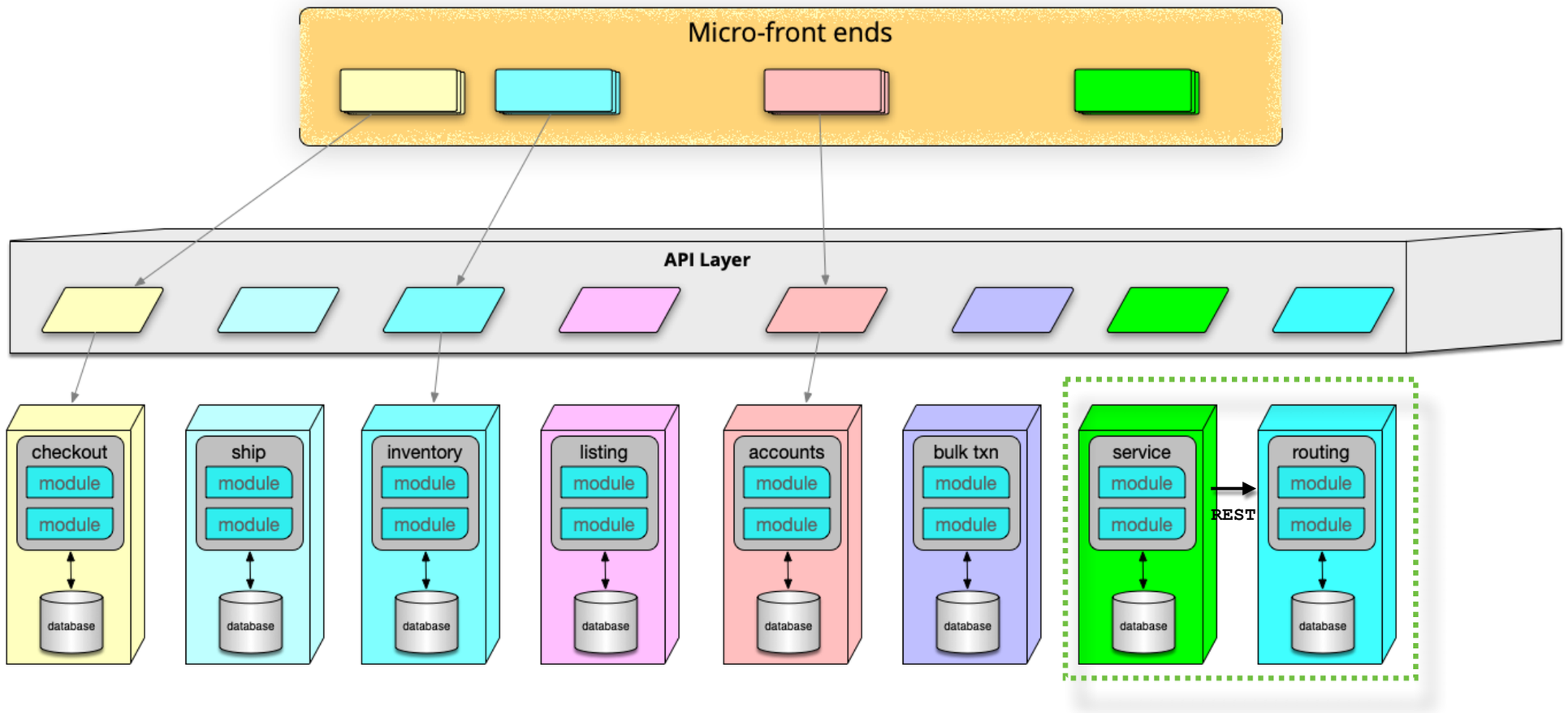


# Synchronous Coupling

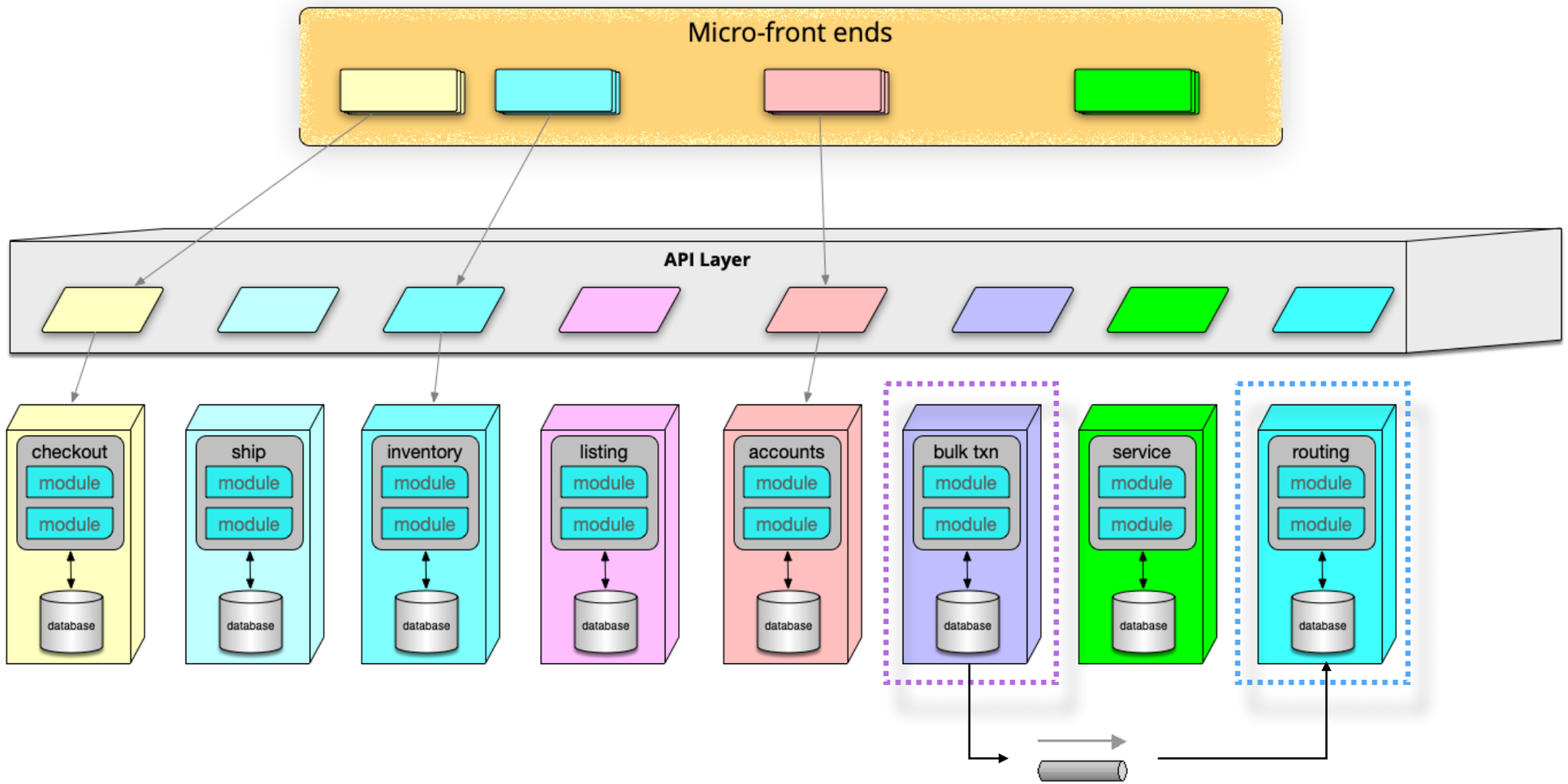


# The Value of Asynchronicity

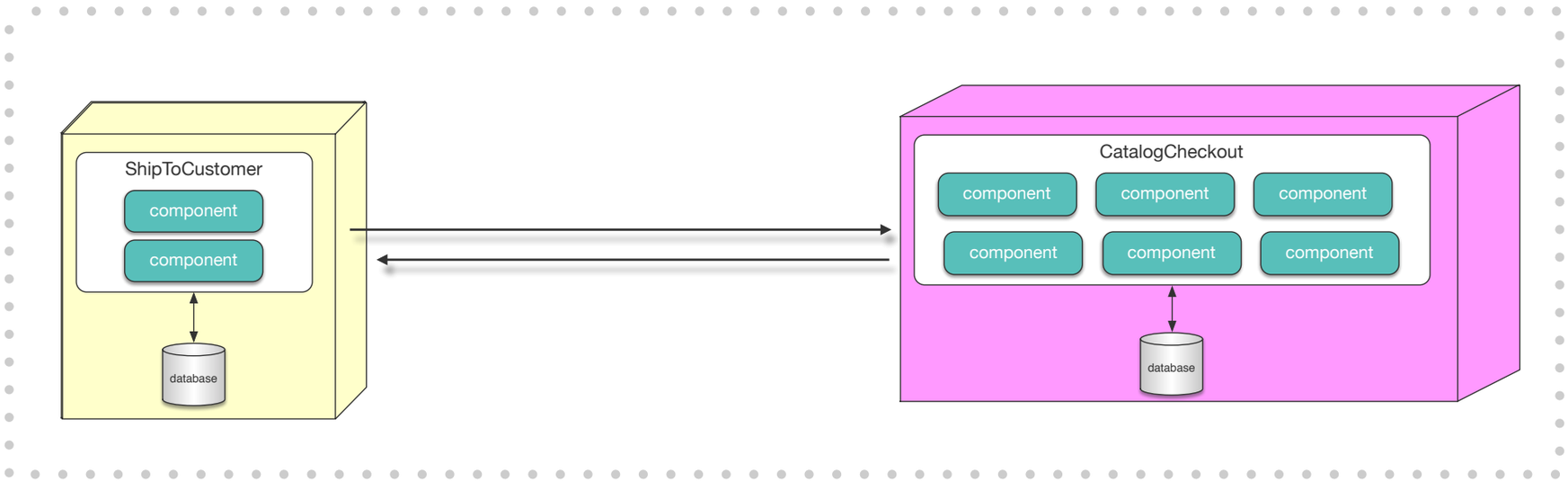



synchronous

# The Value of Asynchronicity



# Synchronous Coupling

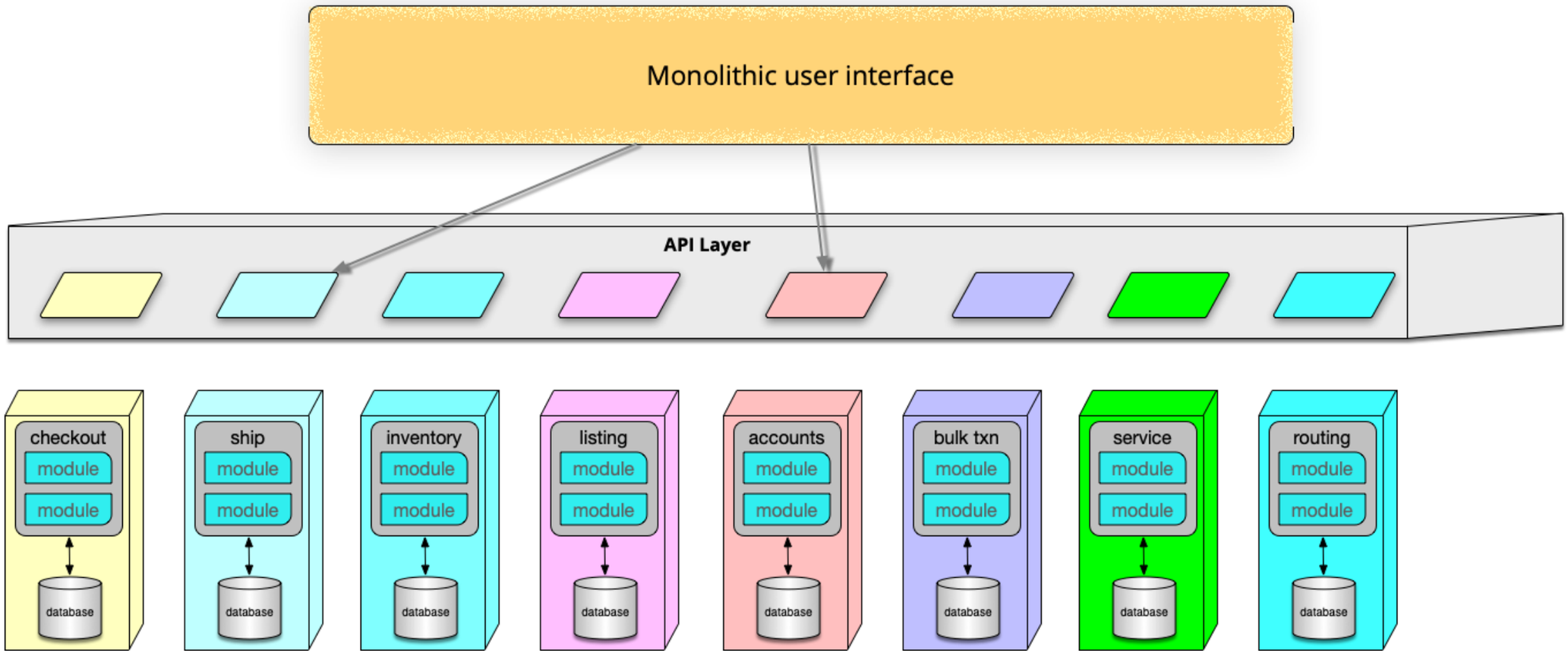



*Synchronous calls expand the scope of quanta.*

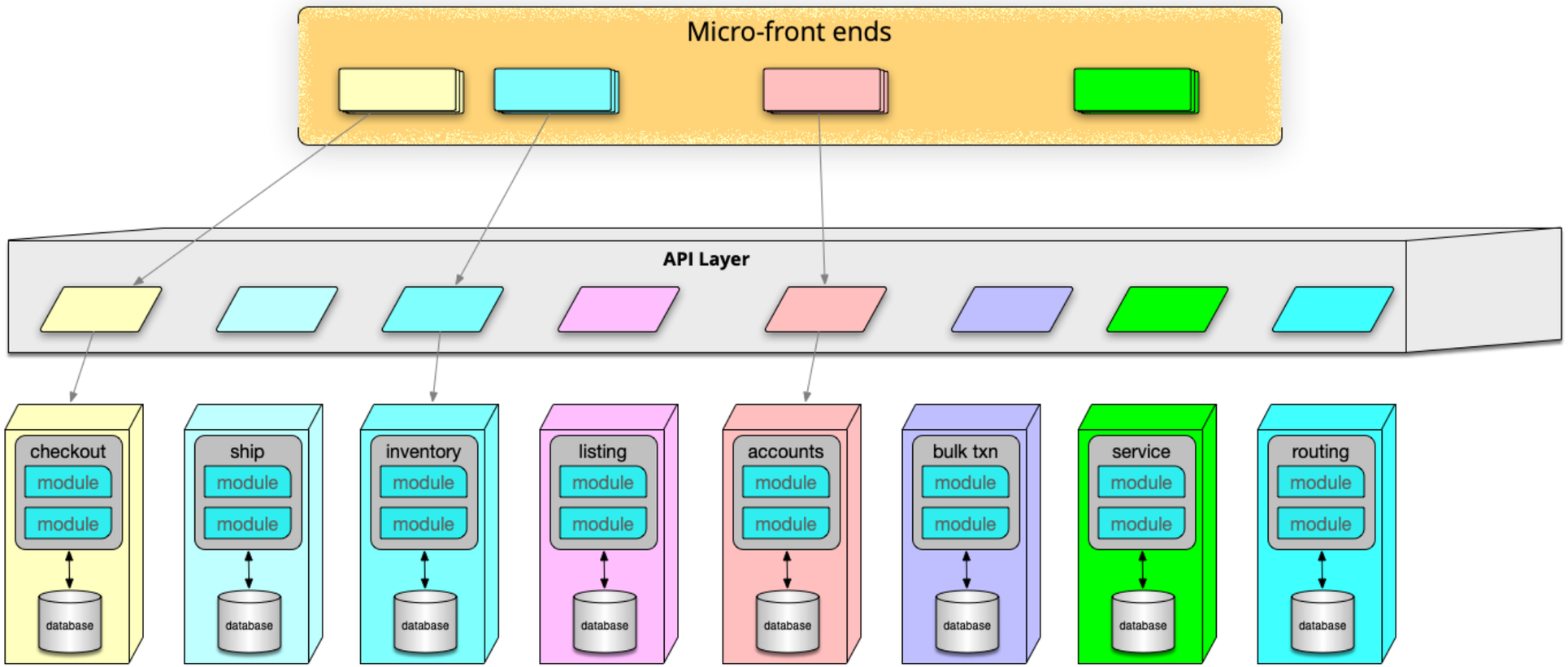
# Why Quantum?

- Operational view of architecture
- Holistic view of architecture
  - Databases
  - User interface
- Useful for structural analysis
- ***Where architecture characteristics live***

# Microservices: Monolithic UI



# Microservices: Micro Front End



**Your Architectural Kata is...**

# Going Going Gone!

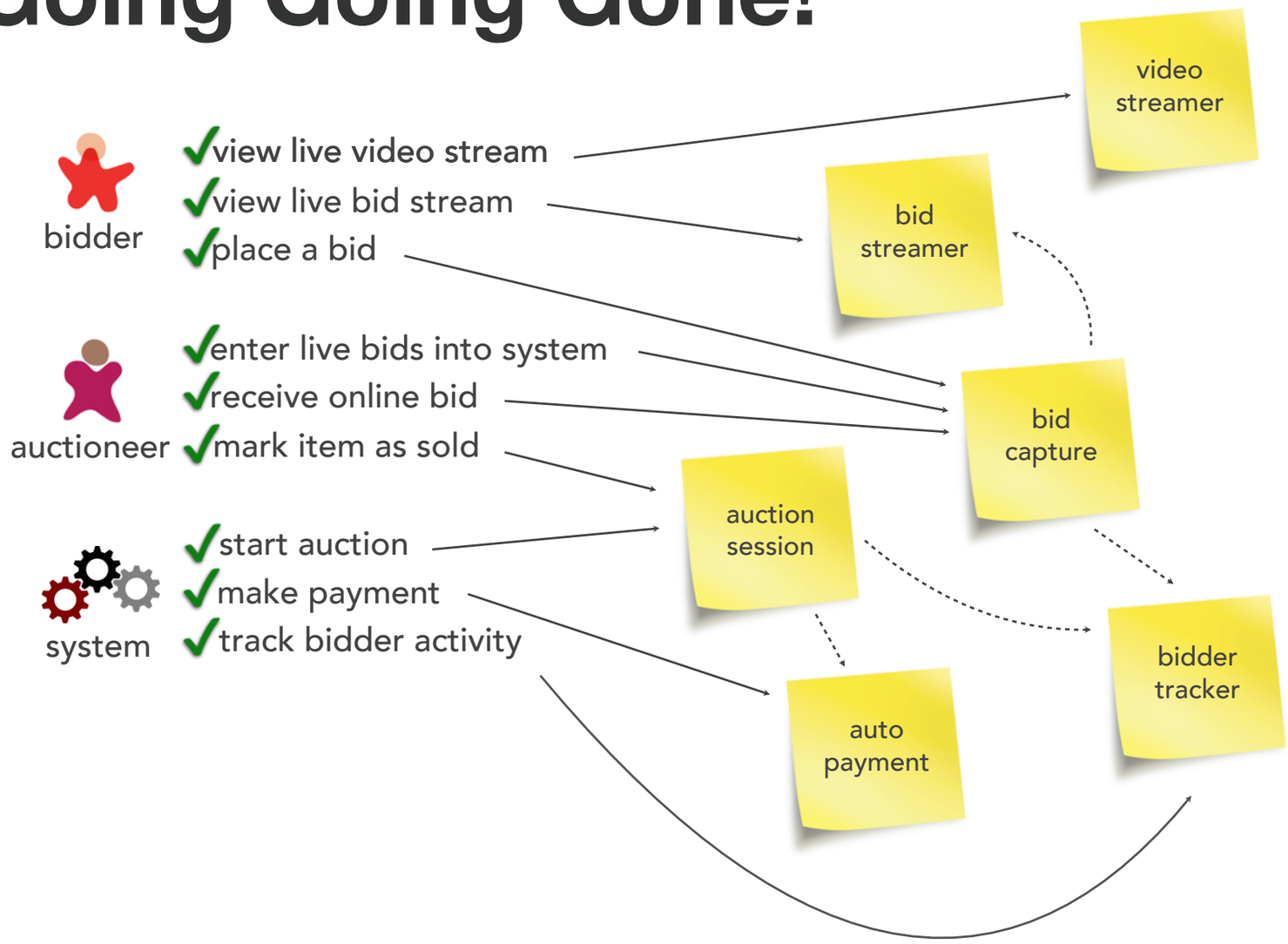
An auction company wants to take their auctions online to a nationwide scale--customers choose the auction to participate in, wait until the auction begins, then bid during the live auction as if they were there in the room, with the auctioneer.

- **Users**: scale up to hundreds of participants (per auction), potentially up to thousands of participants, and as many simultaneous auctions as possible
- **Requirements**:
  - bidders can see a live video stream of the auction and see all bids as they occur
  - auctions must be as real-time as possible
  - both online and live bids must be received in the order in which they are placed
  - bidders register with credit card; system automatically charges card if bidder wins
  - participants must be tracked via a reputation index
- **Additional Context:**
  - auction company is expanding aggressively by merging with smaller competitors
  - if nationwide auction is a success, replicate the model overseas
  - budget is not constrained--this is a strategic direction
  - company just exited a lawsuit where they settled a suit alleging fraud

availability reliability performance scalability elasticity (security)

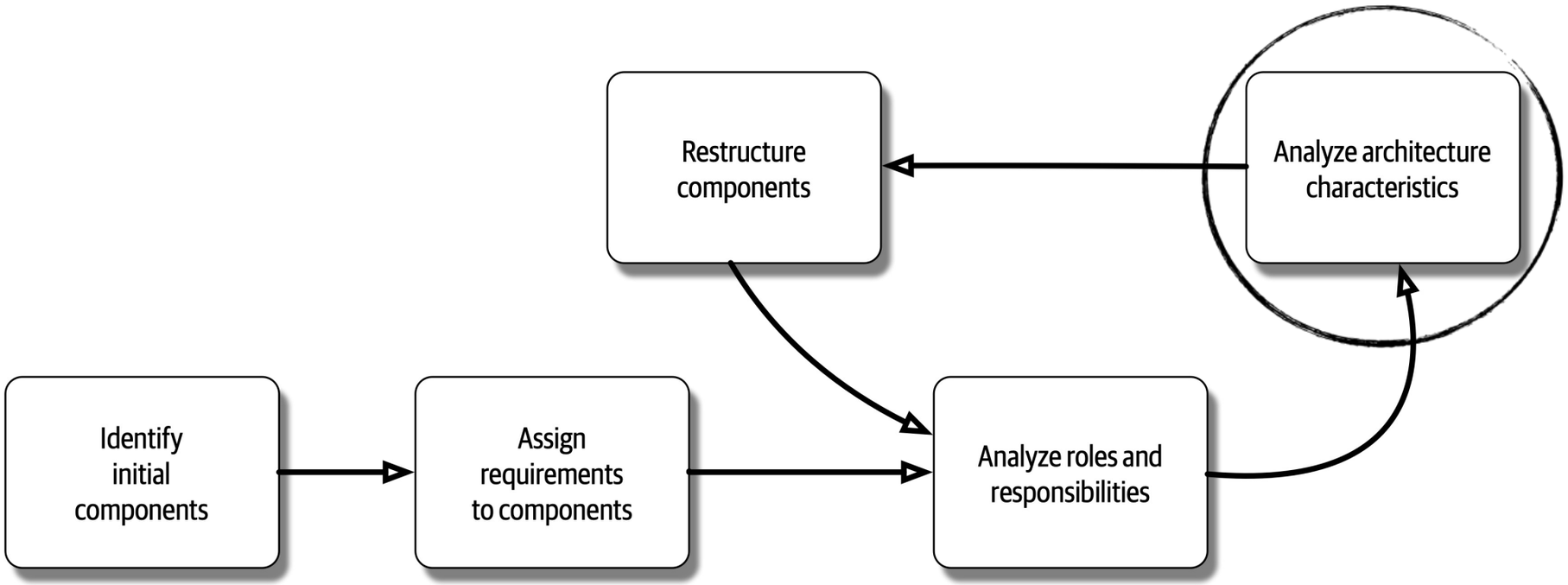
Your Architectural Kata is...

# Going Going Gone!



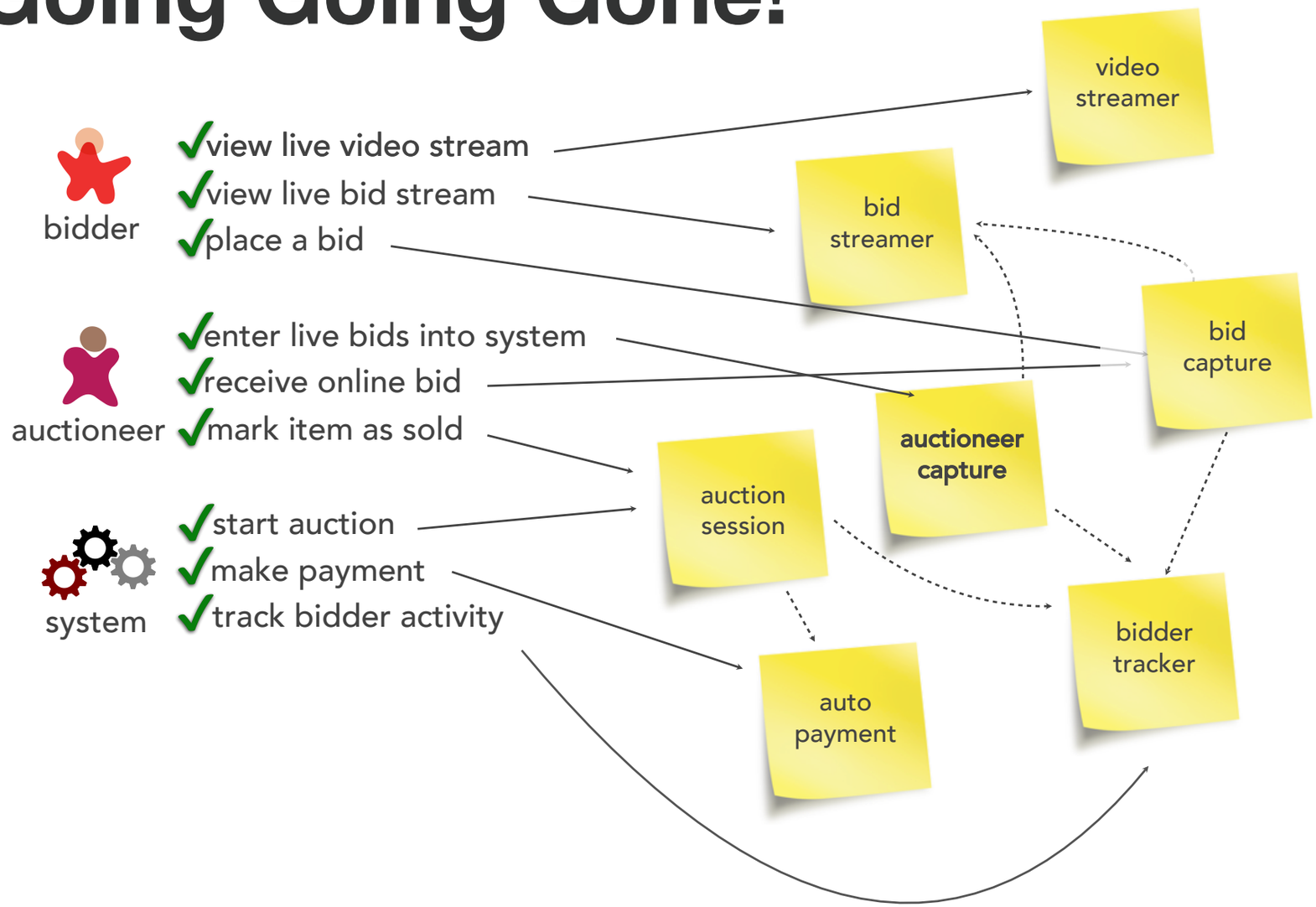
Your Architectural Kata is...

# Going Going Gone!



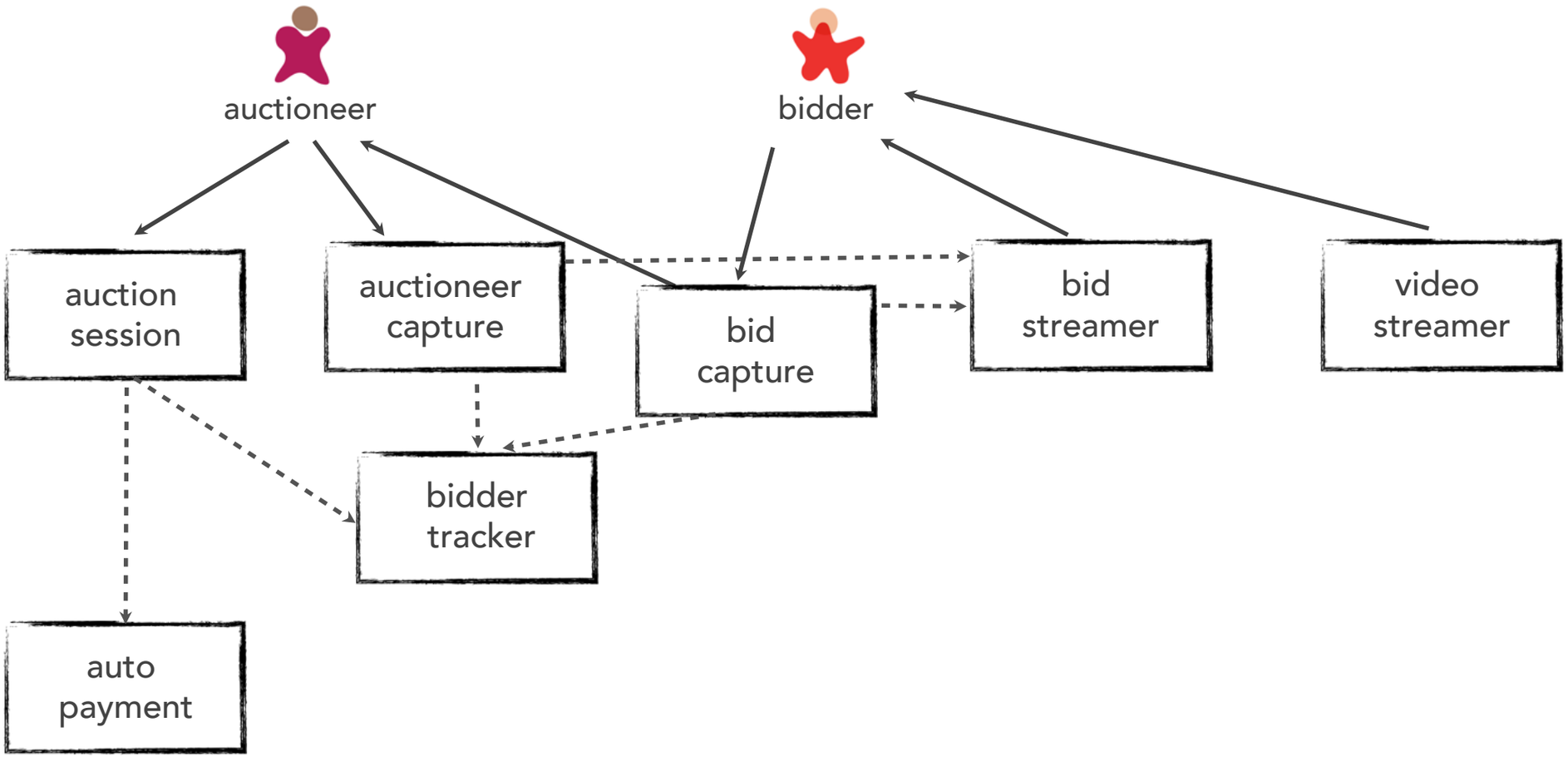
Your Architectural Kata is...

# Going Going Gone!



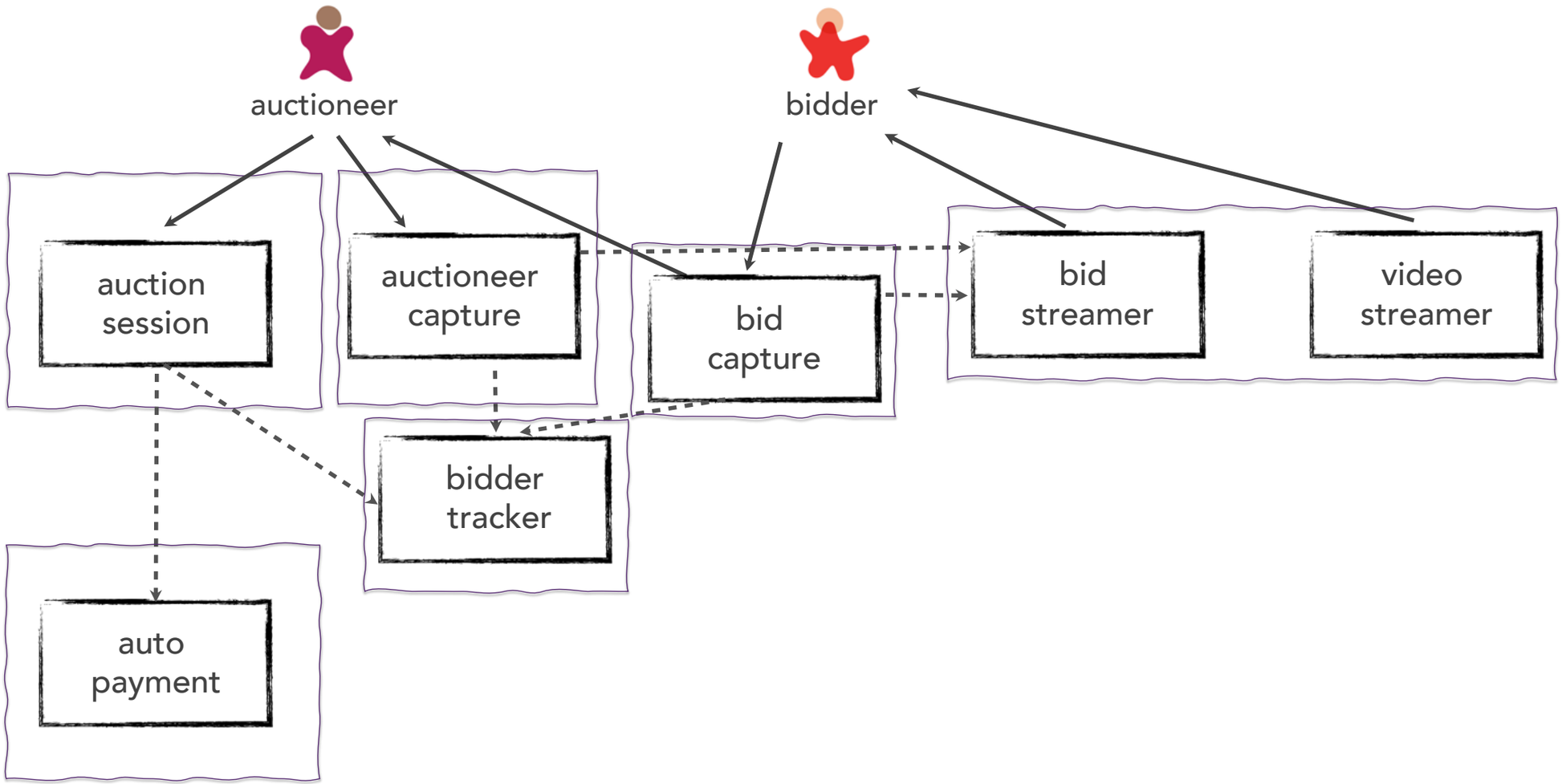
Your Architectural Kata is...

# Going Going Gone!



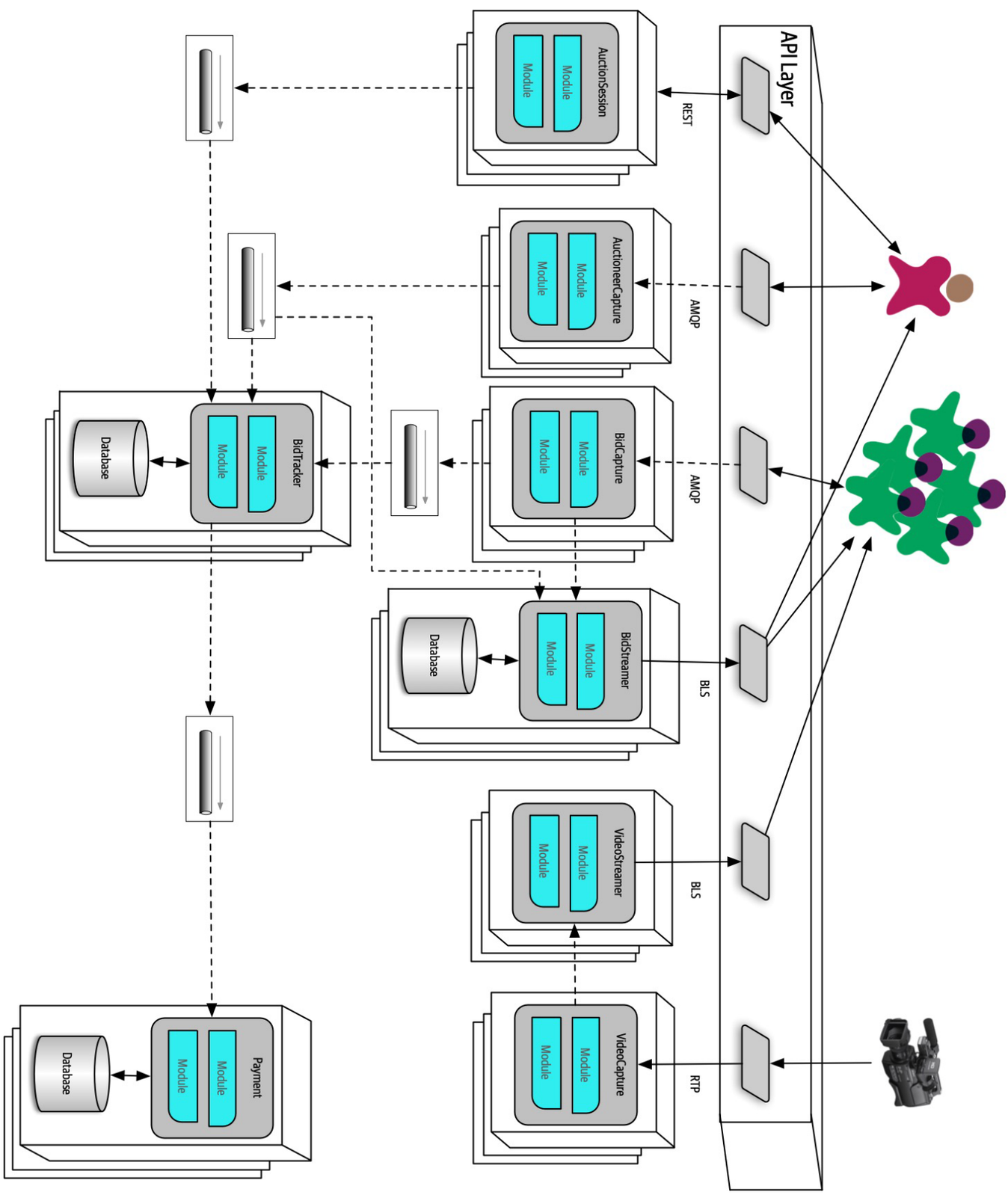
Your Architectural Kata is...

# Going Going Gone!



Your Architectural Kata is...

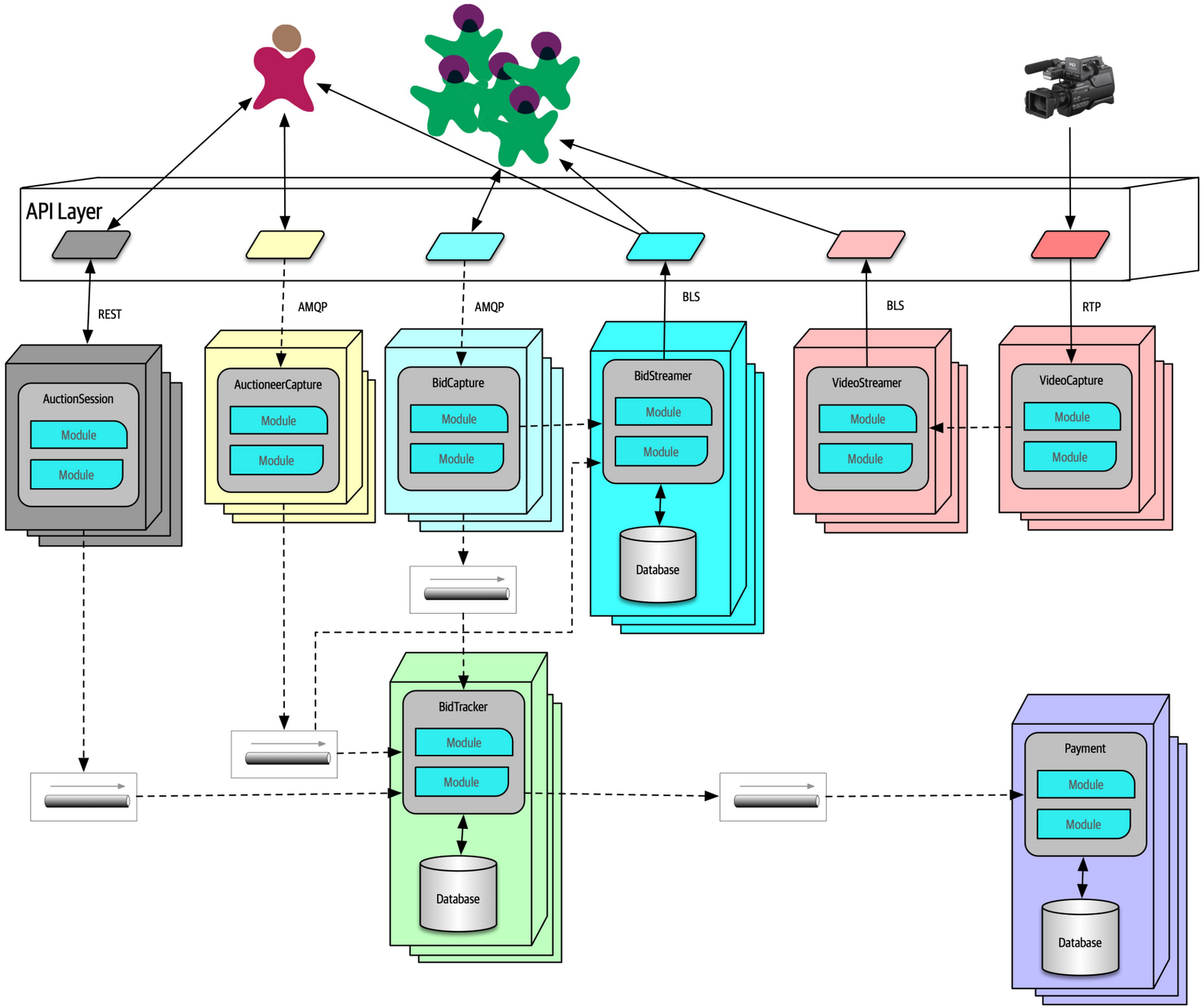
# Going Going Gone!



Your Architectural Kata is...

# Going Going Gone!

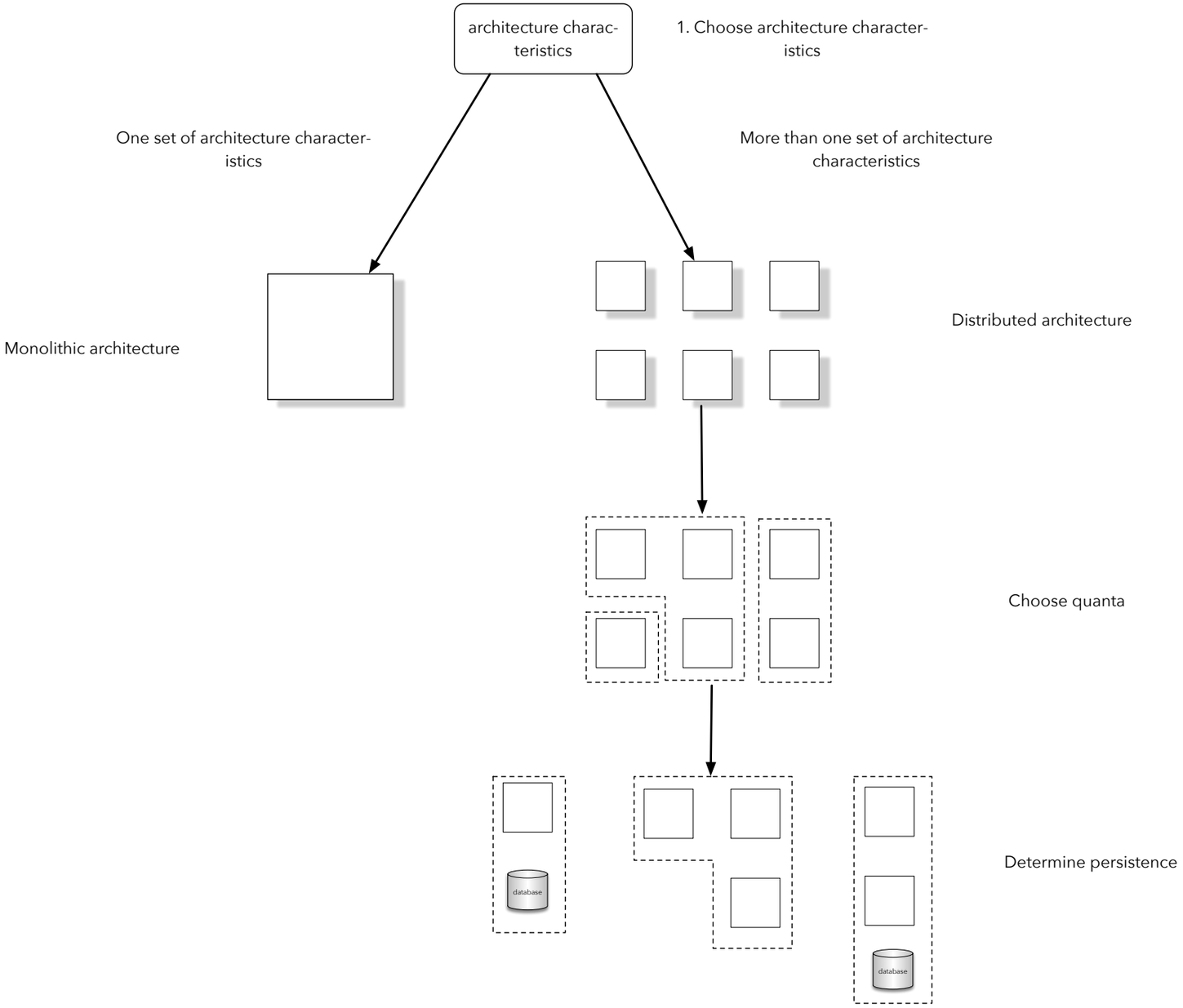
quanta



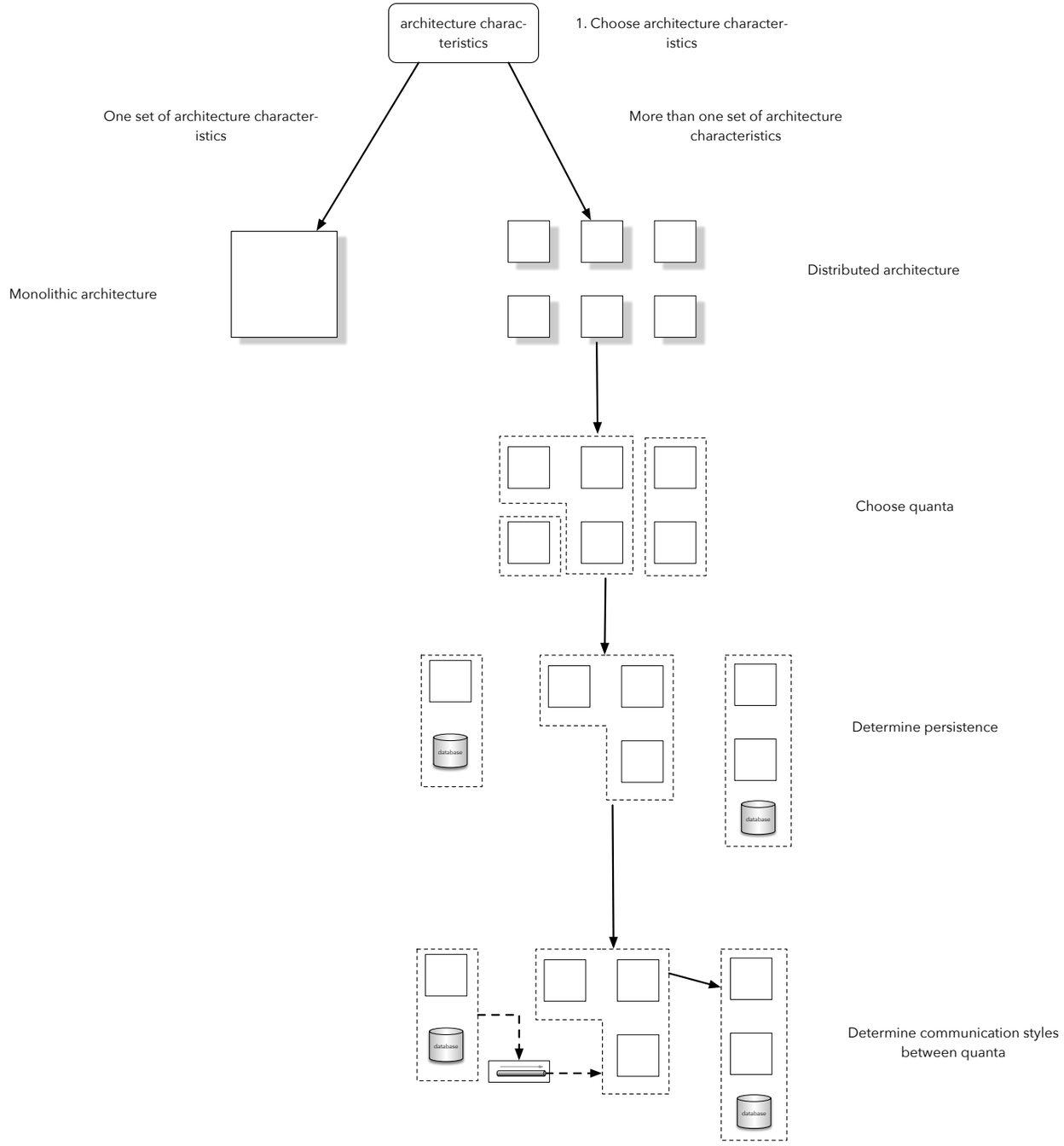
# choosing an architecture

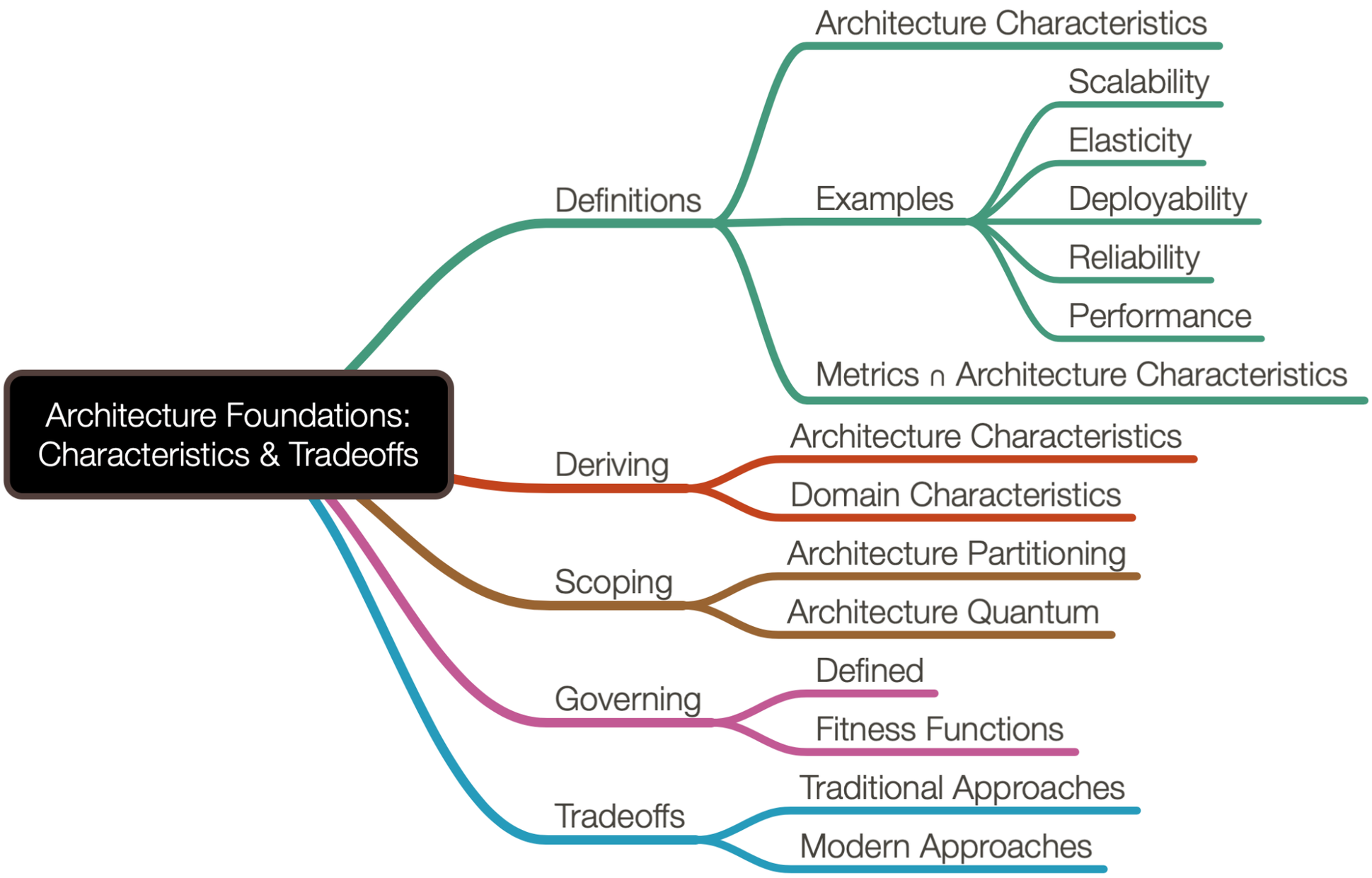
# choosing an architecture

# choosing an architecture



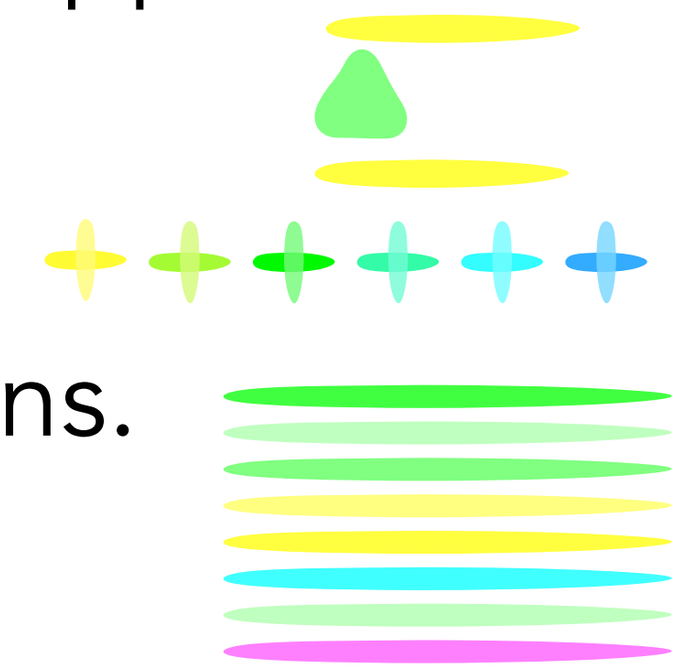
# choosing an architecture




Architecture Foundations: Characteristics & Tradeoffs
Definitions
Architecture Characteristics
Examples
Scalability
Elasticity
Deployability
Reliability
Performance
Metrics ∩ Architecture Characteristics
Deriving
Architecture Characteristics
Domain Characteristics
Scoping
Architecture Partitioning
Architecture Quantum
Governing
Defined
Fitness Functions
Tradeoffs
Traditional Approaches
Modern Approaches

# Evolutionary Architecture

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



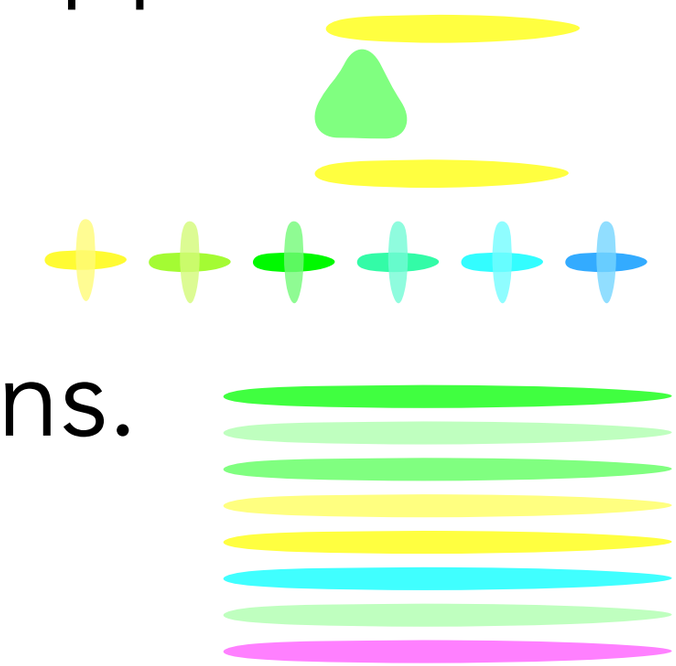
# Evolutionary Architecture

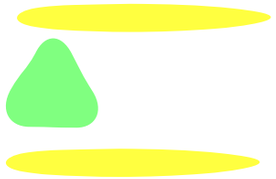
An evolutionary architecture supports

**guided**

incremental change

across multiple dimensions.



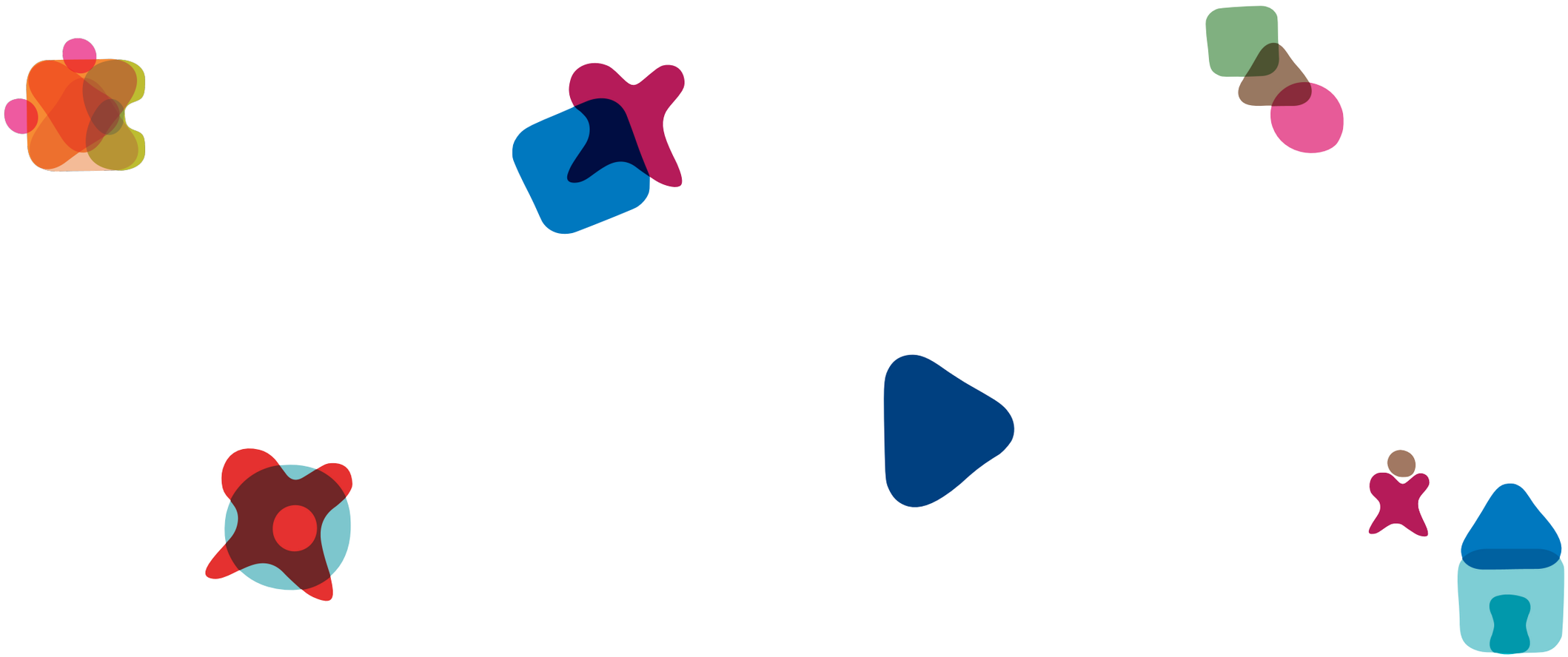


# guided

*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



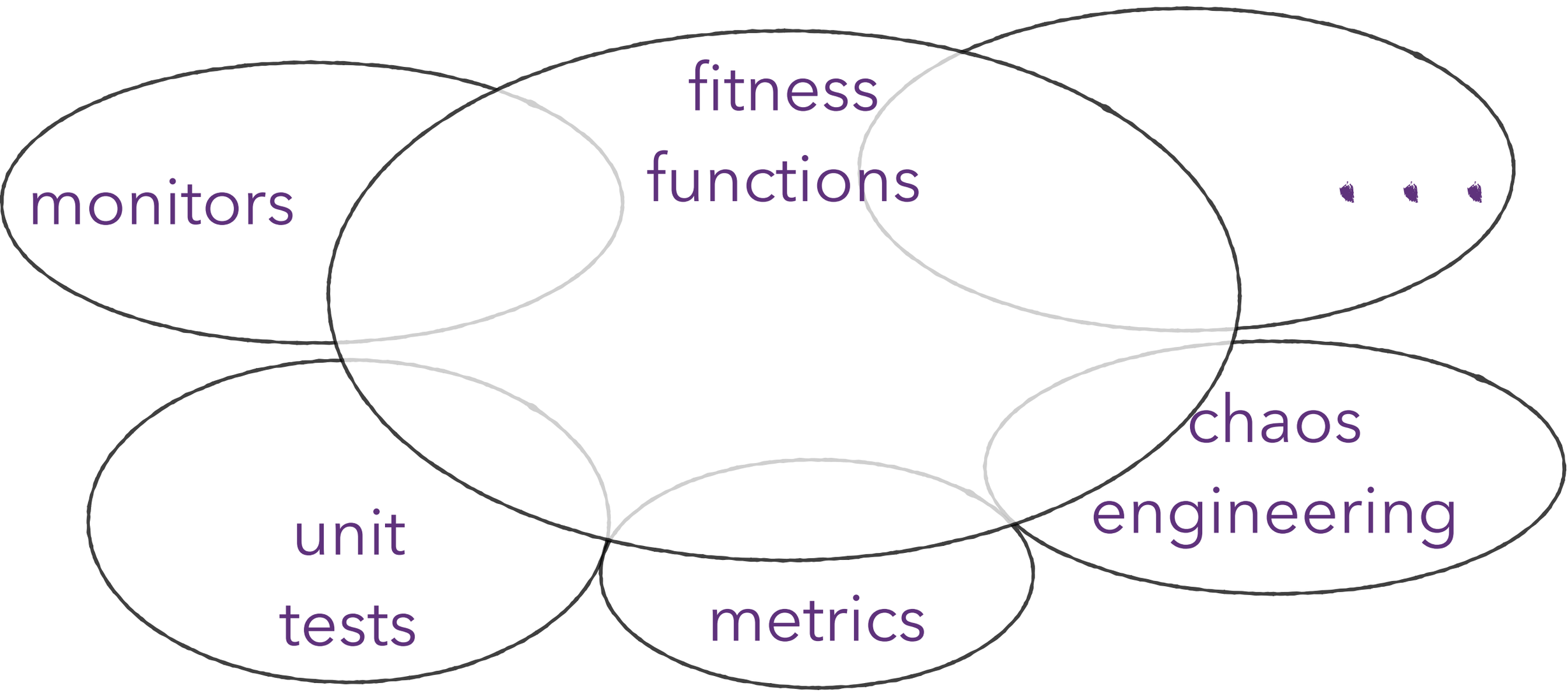
fitness function = length of route

**guided**

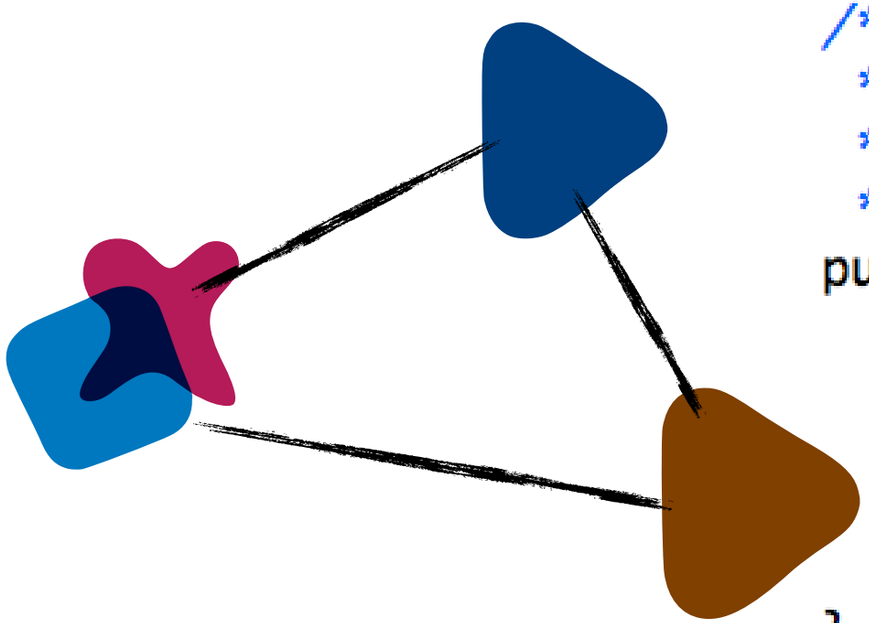
***architectural*** *fitness function*:

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

# Fitness Functions



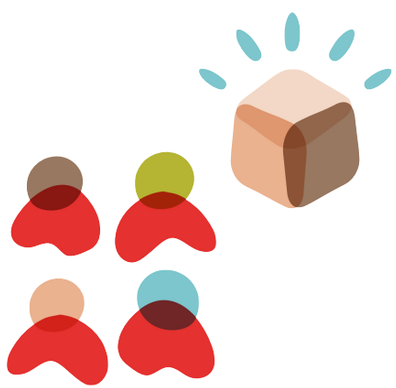
# Cyclic Dependency Function

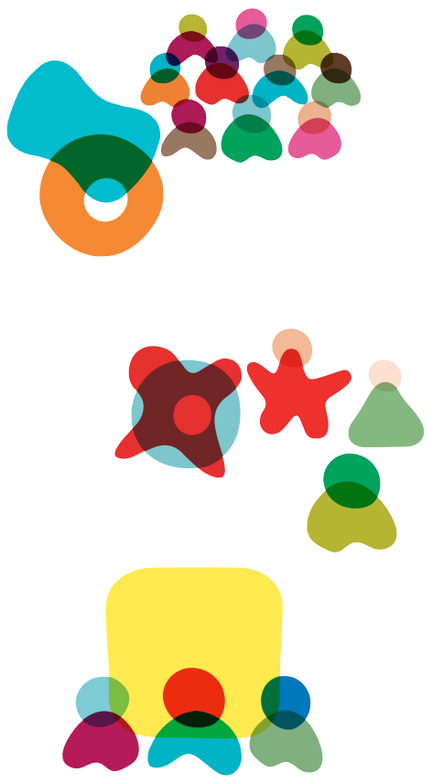


```
/**
 * Tests that a package dependency cycle does not
 * exist for any of the analyzed packages.
 */
public void testAllPackages() {

    Collection packages = jdepend.analyze();

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

clarkware.com/software/JDepend.html

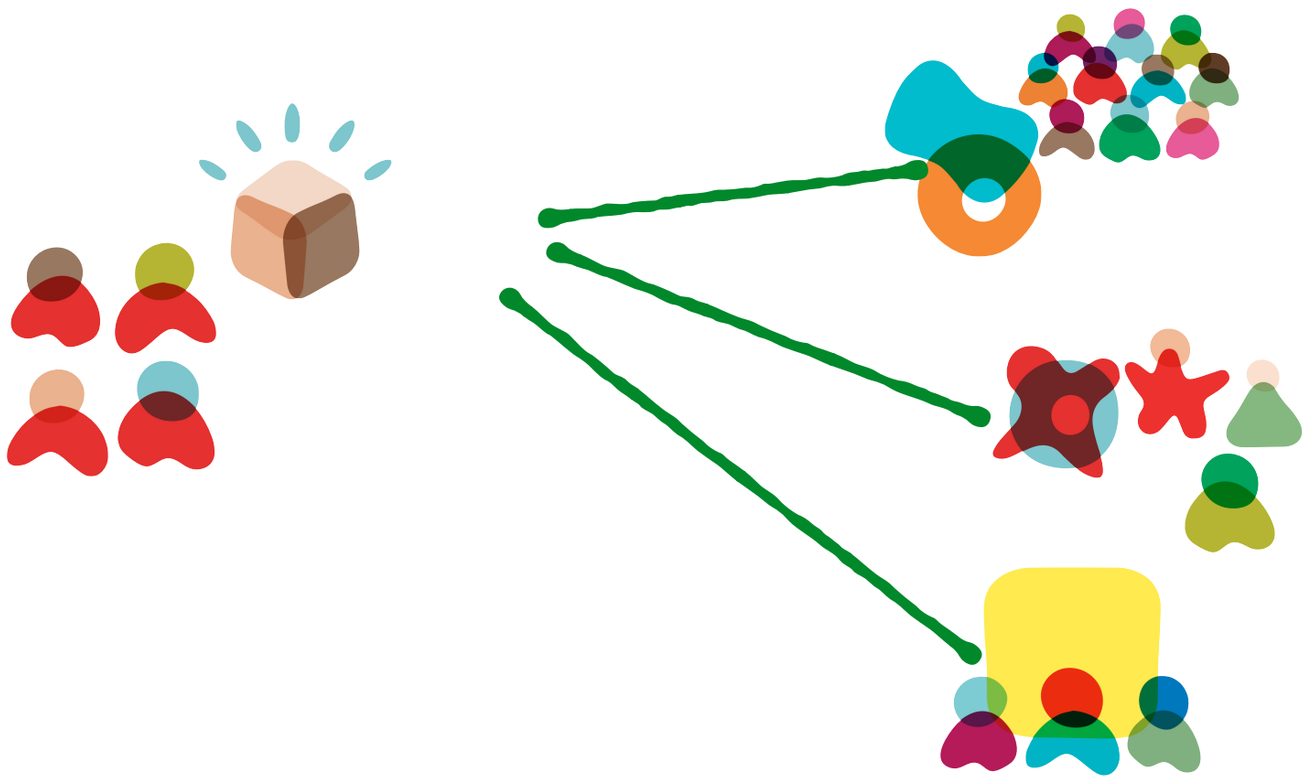
# Consumer Driven Contracts





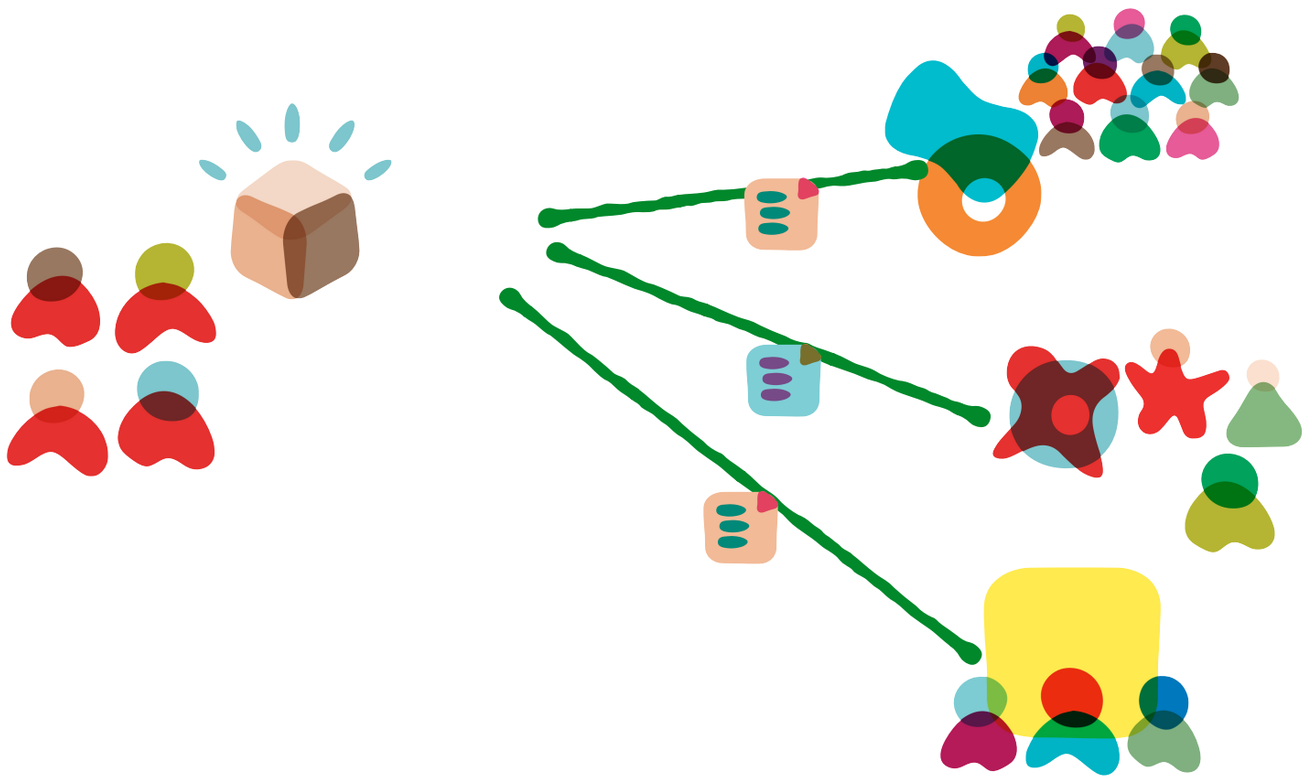
martinfowler.com/articles/consumerDrivenContracts.html

# Consumer Driven Contracts



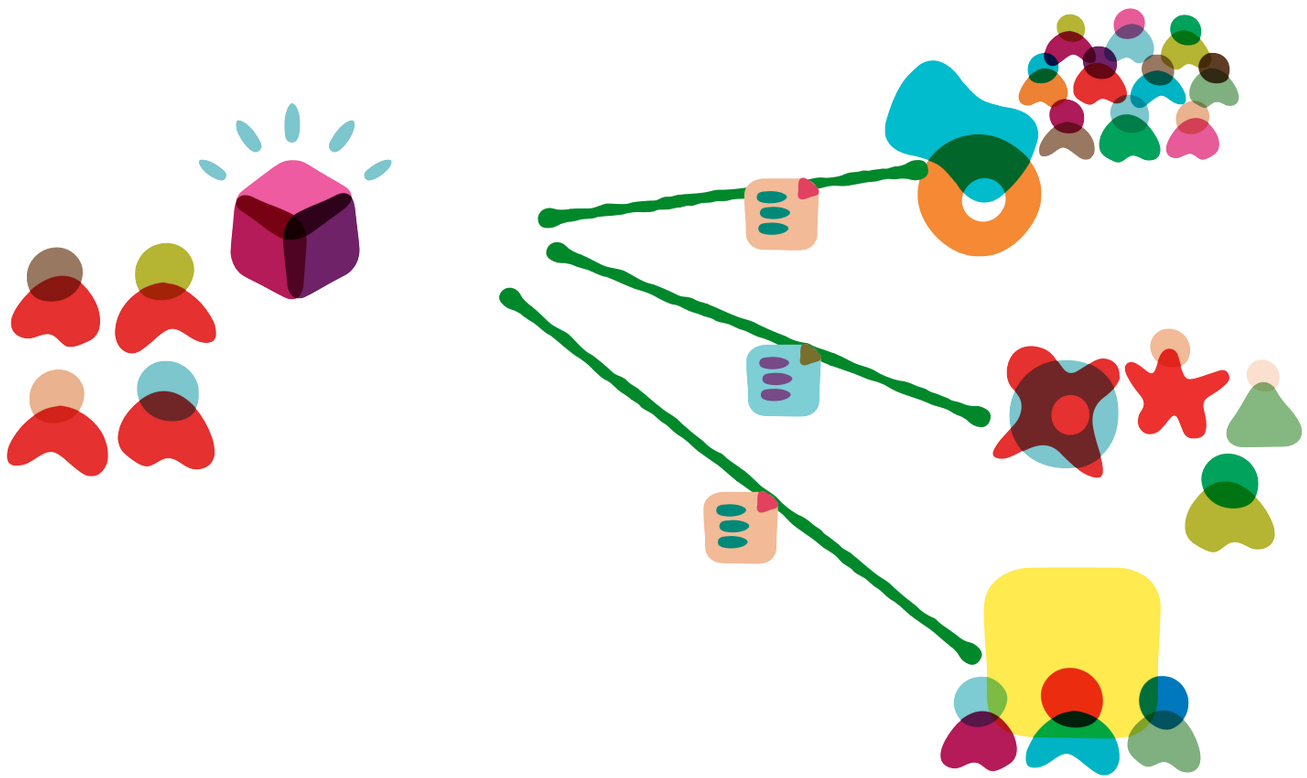
martinfowler.com/articles/consumerDrivenContracts.html

# Consumer Driven Contracts



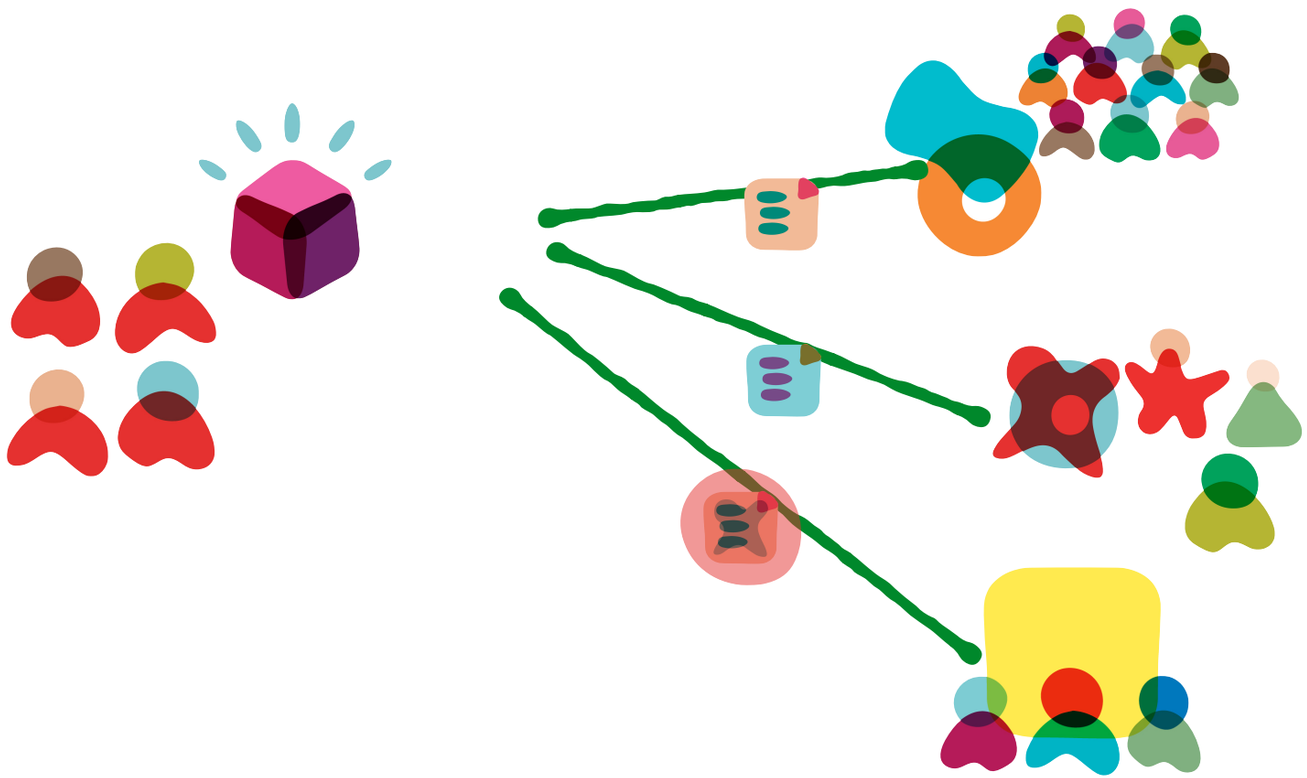
martinfowler.com/articles/consumerDrivenContracts.html

# Consumer Driven Contracts



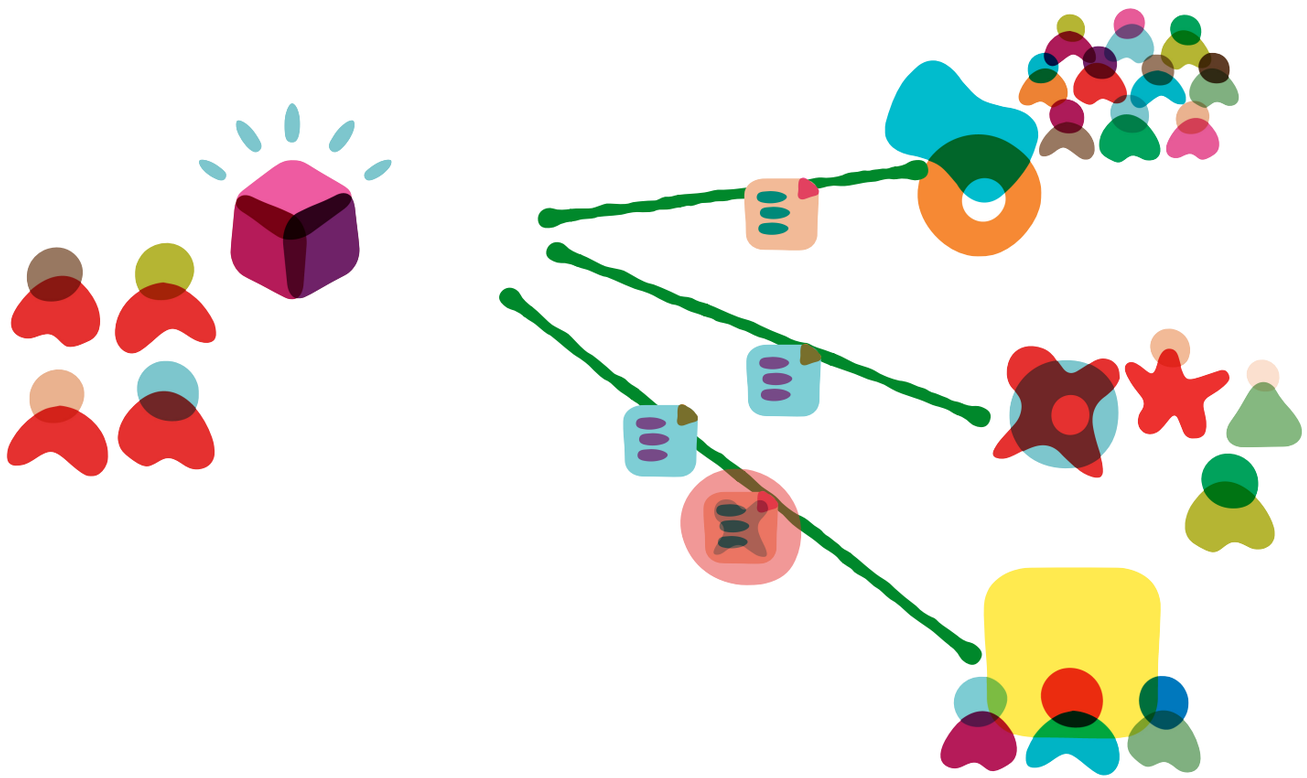
martinfowler.com/articles/consumerDrivenContracts.html

# Consumer Driven Contracts



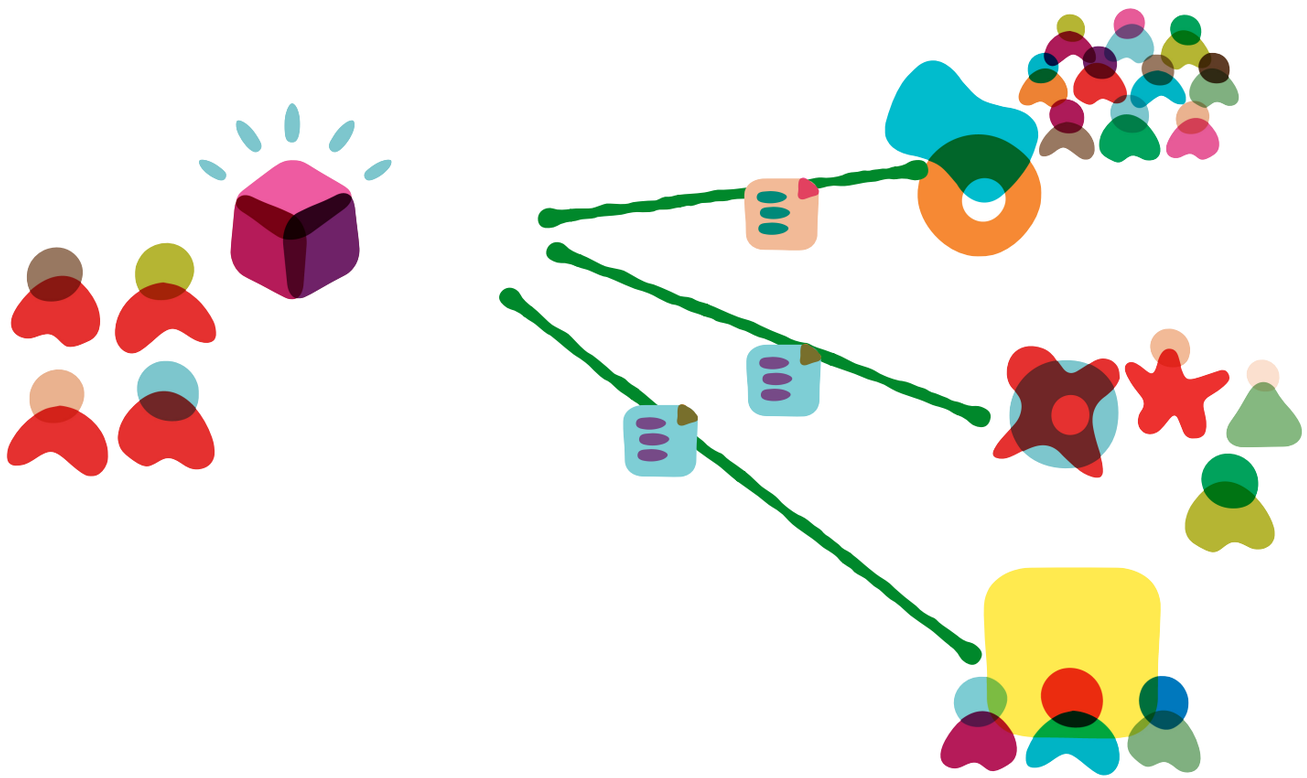
martinfowler.com/articles/consumerDrivenContracts.html

# 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


conformity monkey
SIMIAN
ARMY


security monkey
SIMIAN
ARMY

maintainable?

maintainable?

Cyclomatic complexity < 50 for all projects

maintainable?

Cyclomatic complexity < 50 for all projects

Naming conventions

maintainable?

Cyclomatic complexity < 50 for all projects

Naming conventions

maintainable?

(incoming/outgoing)

Controlled afferent/efferent coupling

Cyclomatic complexity < 50 for all projects

Naming conventions

immutability

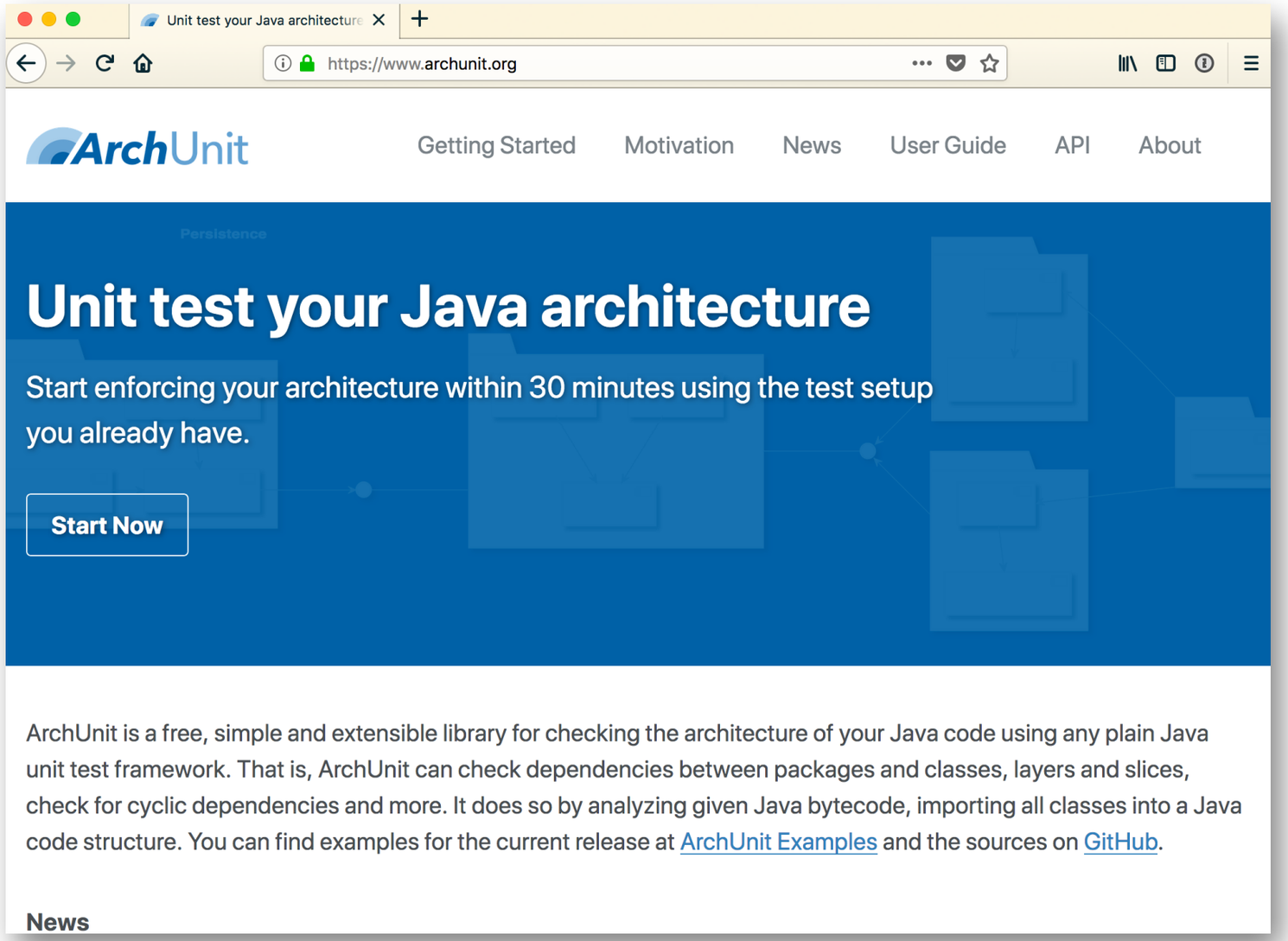
maintainable?

(incoming/outgoing)

Controlled afferent/efferent coupling

ArchUnit

https://www.archunit.org/



ArchUnit

https://www.archunit.org/

# coding rules

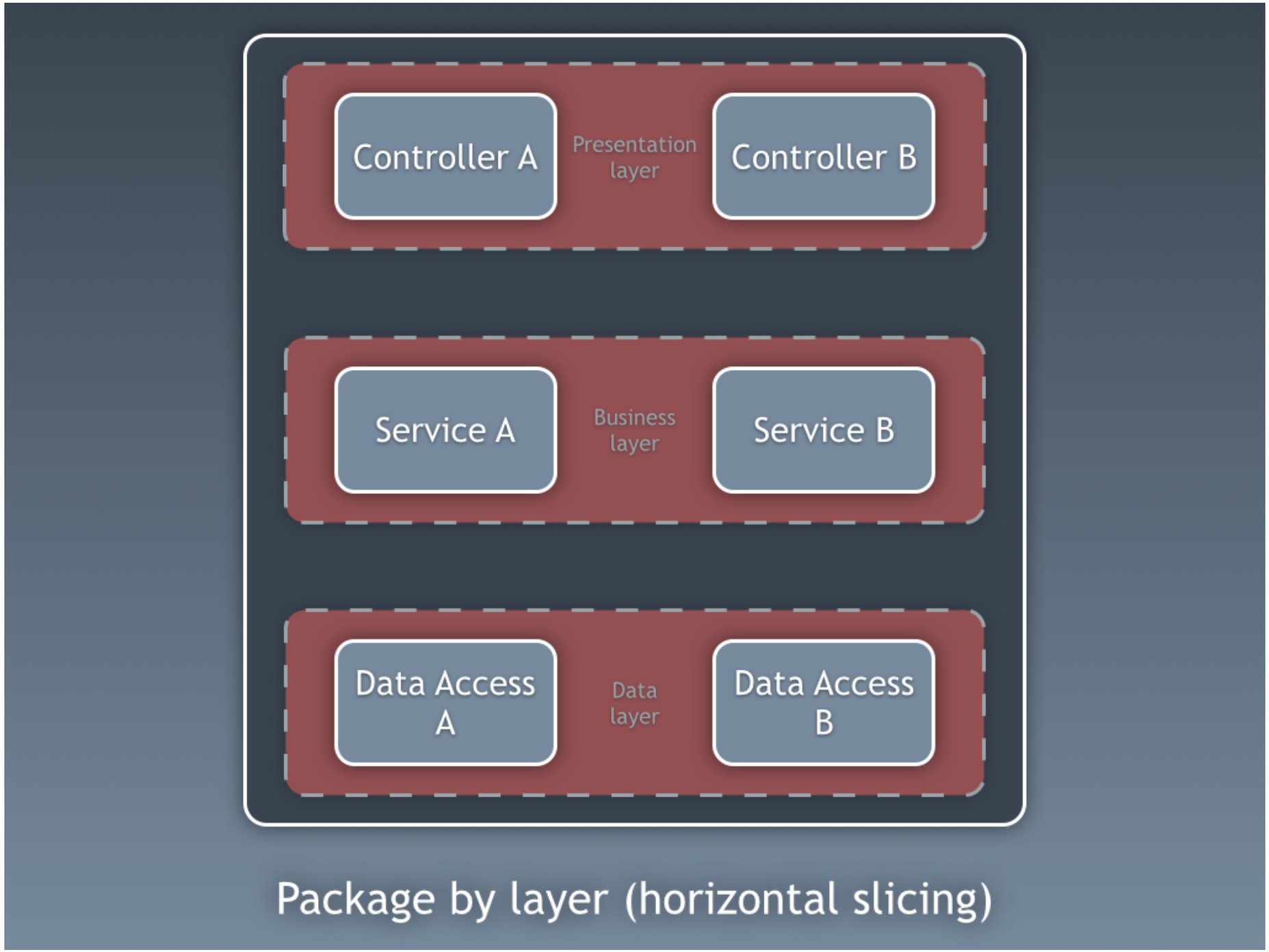
```java
import static com.tngtech.archunit.lang.syntax.ArchRuleDefinition.noClasses;
import static com.tngtech.archunit.library.GeneralCodingRules.ACCESS_STANDARD_STREAMS;
import static com.tngtech.archunit.library.GeneralCodingRules.NO_CLASSES_SHOULD_ACCESS_STANDARD_STREAMS;
import static com.tngtech.archunit.library.GeneralCodingRules.NO_CLASSES_SHOULD_THROW_G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

public class CodingRulesTest {
    private JavaClasses classes;

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

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

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

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

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

ArchUnit

https://www.archunit.org/

# interface rules

```java
public class InterfaceRules {

    @Test
    public void interfaces_should_not_have_names_ending_with_the_word_interface() {
        JavaClasses classes = new ClassFileImporter().importClasses(
                SomeBusinessInterface.class,
                SomeDao.class
        );

        noClasses().that().areInterfaces().should().haveNameMatching(".*Interface").check(classes);
    }

    @Test
    public void interfaces_should_not_have_simple_class_names_ending_with_the_word_interface() {
        JavaClasses classes = new ClassFileImporter().importClasses(
                SomeBusinessInterface.class,
                SomeDao.class
        );

        noClasses().that().areInterfaces().should().haveSimpleNameContaining("Interface").check(classes);
    }

    @Test
    public void interfaces_must_not_be_placed_in_implementation_packages() {
        JavaClasses classes = new ClassFileImporter().importPackagesOf(SomeInterfacePlacedInTheWrongPackage.class);

        noClasses().that().resideInAPackage("..impl..").should().beInterfaces().check(classes);
    }
}
```


Controller A
Presentation layer
Controller B
Service A
Business layer
Service B
Data Access A
Data layer
Data Access B
Package by layer (horizontal slicing)

ArchUnit

https://www.archunit.org/

```java
public class LayerDependencyRulesTest {
    private JavaClasses classes;

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

    @Test
    public void services_should_not_access_controllers() {
        noClasses().that().resideInAPackage("..service..")
                .should().accessClassesThat().resideInAPackage("..controller..").check(classes);
    }

    @Test
    public void persistence_should_not_access_services() {
        noClasses().that().resideInAPackage("..persistence..")
                .should().accessClassesThat().resideInAPackage("..service..").check(classes);
    }

    @Test
    public void services_should_only_be_accessed_by_controllers_or_other_services() {
        classes().that().resideInAPackage("..service..")
                .should().onlyBeAccessed().byAnyPackage("..controller..", "..service..").check(classes);
    }
}
```

# layer dependency

# 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/

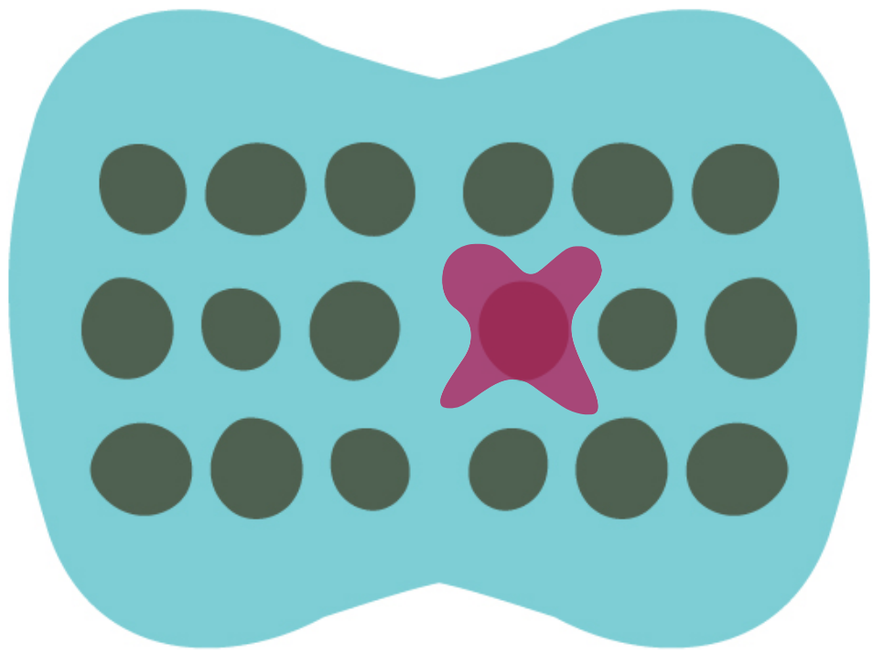
```
// Only classes in the data namespace can have a dependency on System.Data
result = Types.InCurrentDomain()
    .That().HaveDependencyOn("System.Data")
    .And().ResideInNamespace(("ArchTest"))
    .Should().ResideInNamespace(("NetArchTest.SampleLibrary.Data"))
    .GetResult().IsSuccessful;
```

# NetArchTest

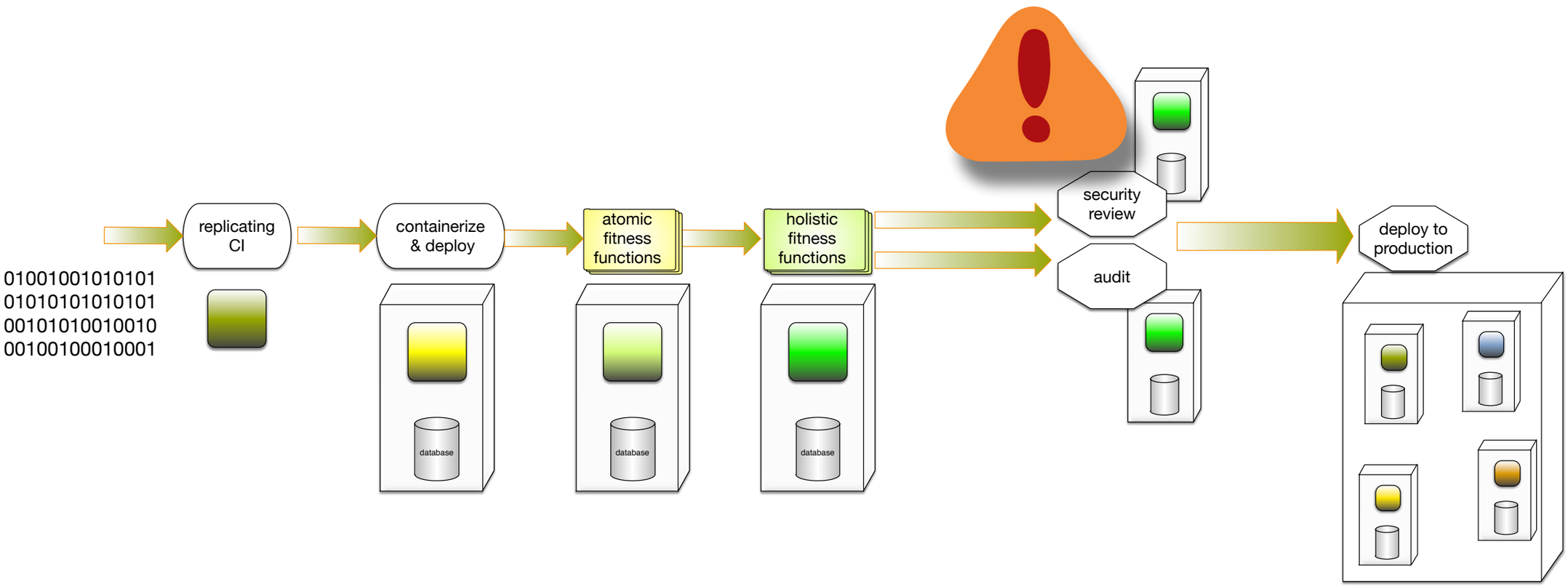
https://github.com/BenMorris/NetArchTest/

```
// 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;
```

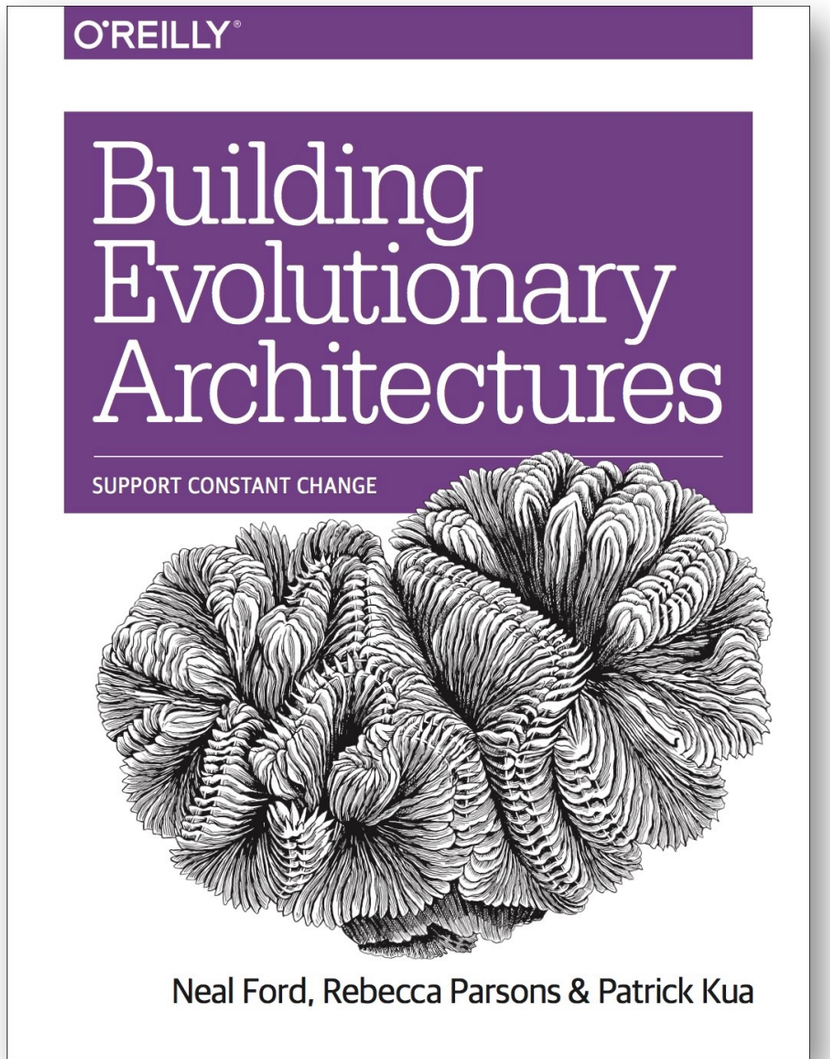
# Zero-day Security Check



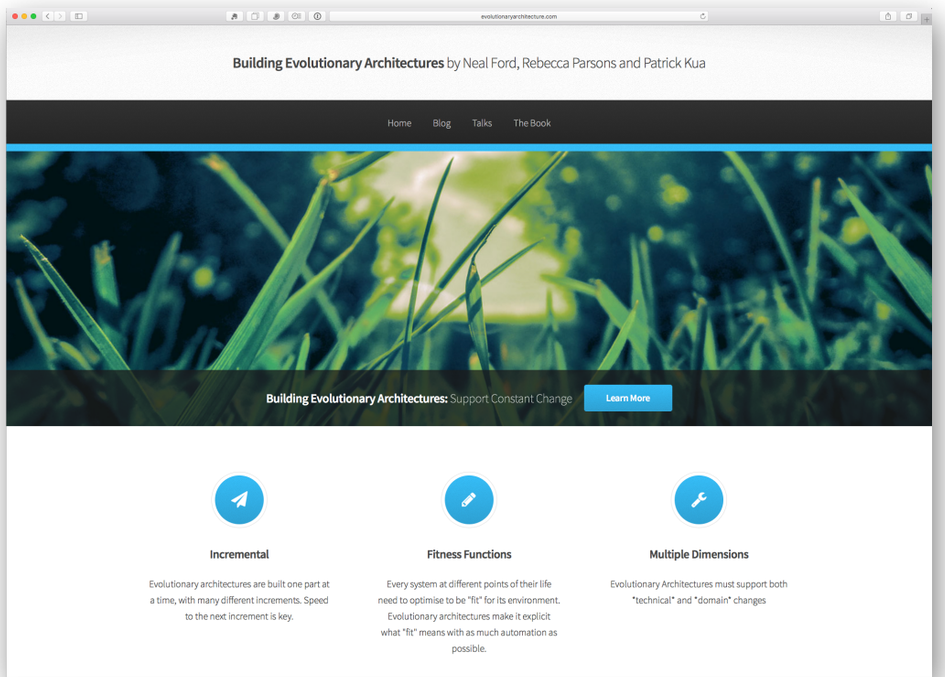
# Zero-day Security Check



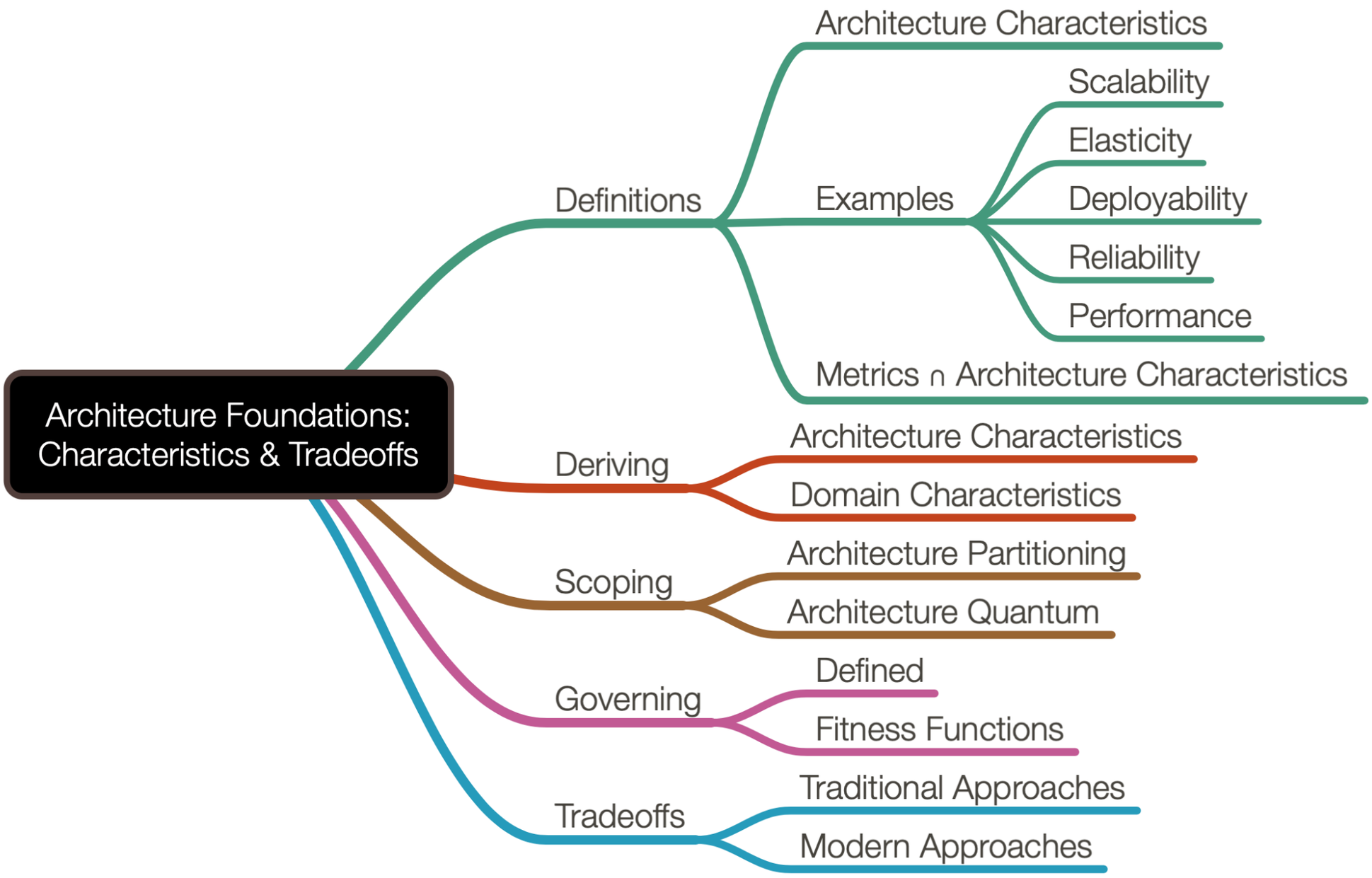
# Building Evolutionary Architectures




For more information:




http://evolutionaryarchitecture.com


Architecture Foundations: Characteristics & Tradeoffs
Definitions
Architecture Characteristics
Examples
Scalability
Elasticity
Deployability
Reliability
Performance
Metrics ∩ Architecture Characteristics
Deriving
Architecture Characteristics
Domain Characteristics
Scoping
Architecture Partitioning
Architecture Quantum
Governing
Defined
Fitness Functions
Tradeoffs
Traditional Approaches
Modern Approaches

# Analyzing Tradeoffs

Documenting

Analyzing Tradeoffs

ATAM

CBAM

# ATAM
# Architecture tradeoff analysis method

– A risk mitigation process developed by the Software Engineering Institute at the Carnegie Mellon University.

– ATAM is most beneficial when done early in the software development life-cycle, when the cost of changing architectures is minimal.

# Steps in the ATM Process

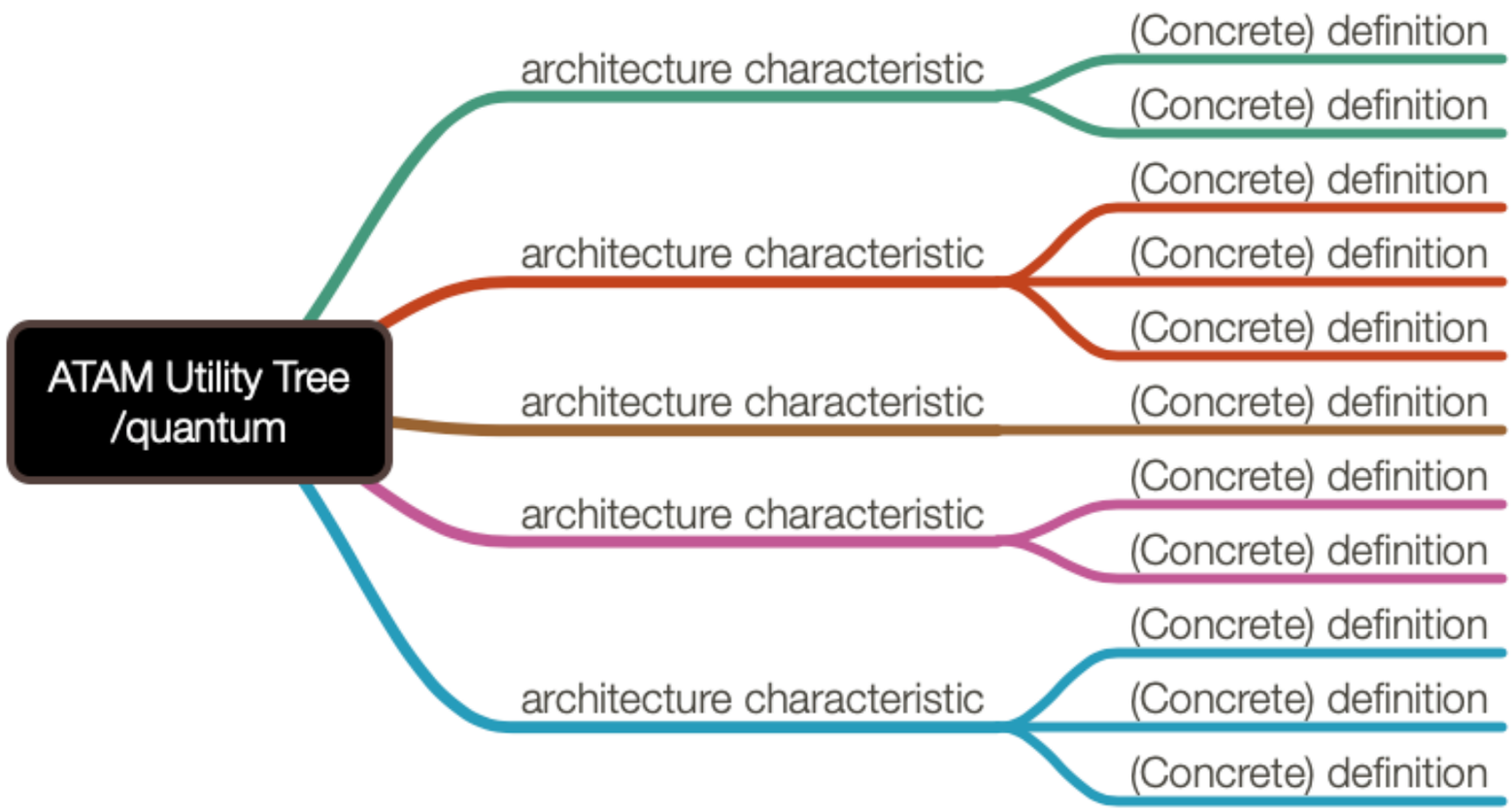
- Present ATAM – Present the concept of ATAM to the stakeholders, and answer any questions about the process.
- Present business drivers – everyone in the process presents and evaluates the business drivers for the system in question.
- Present the architecture – the architect presents the high level architecture to the team, with an 'appropriate level of detail'
- Identify architectural approaches – different architectural approaches to the system are presented by the team, and discussed.
- Generate quality attribute utility tree – define the core business and technical requirements of the system, and map them to an appropriate architectural property. Present a scenario for this given requirement.
- Analyze architectural approaches – Analyze each scenario, rating them by priority. The architecture is then evaluated against each scenario.
- Brainstorm and prioritize scenarios – among the larger stakeholder group, present the current scenarios, and expand.
- Analyze architectural approaches – Perform step 6 again with the added knowledge of the larger stakeholder community.
- Present results – provide all documentation to the stakeholders.

# Steps in the ATM Process

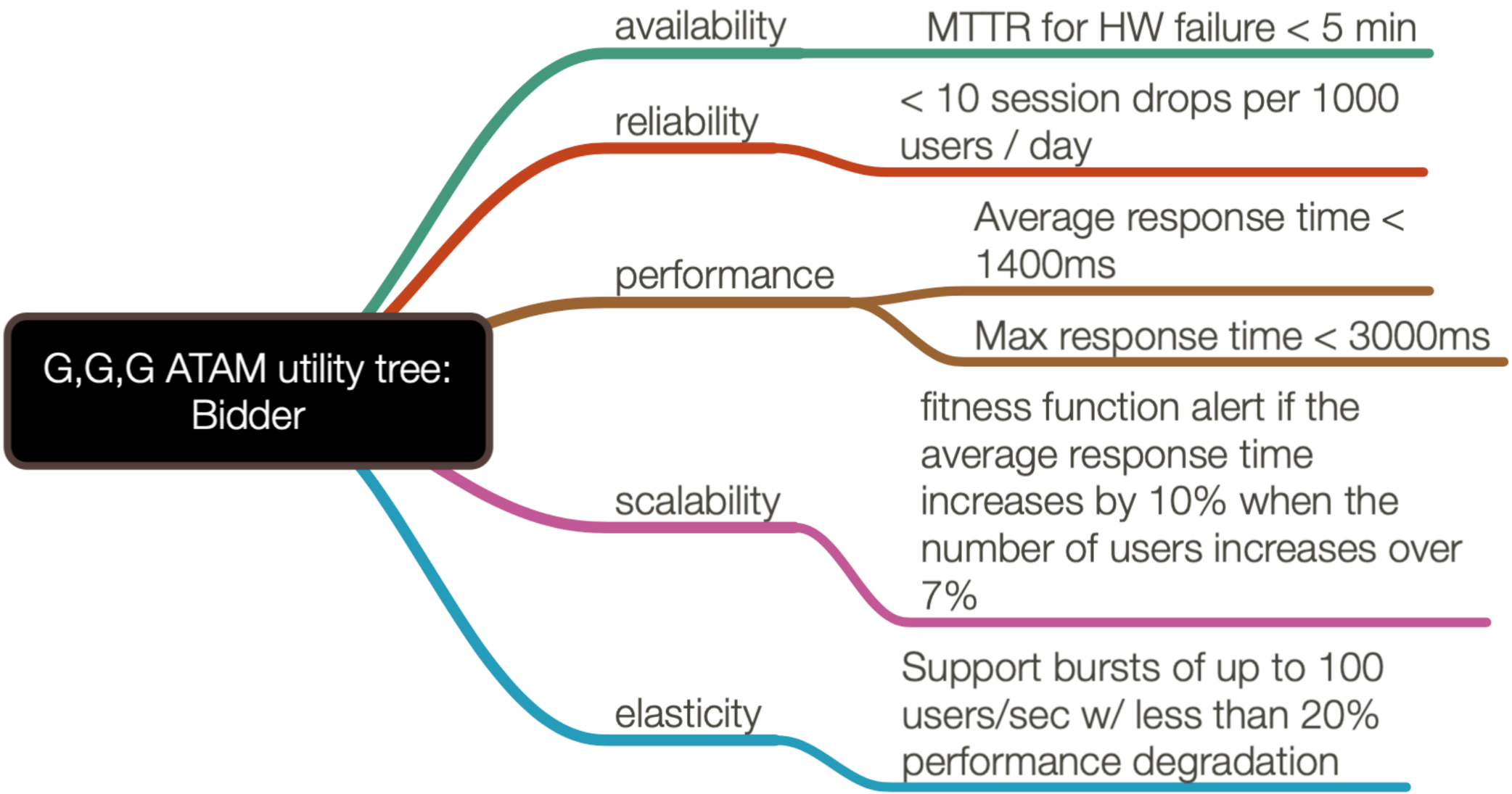
**Useful**

- ~~Present ATAM – Present the concept of ATAM to the stakeholders, and answer any questions about the process.~~
- ~~Present business drivers – everyone in the process presents and evaluates the business drivers for the system in question.~~
- ~~Present the architecture – the architect presents the high level architecture to the team, with an 'appropriate level of detail'~~
- ~~Identify architectural approaches – different architectural approaches to the system are presented by the team, and discussed.~~
- Generate quality attribute utility tree – define the core business and technical requirements of the system, and map them to an appropriate architectural property. Present a scenario for this given requirement.
- ~~Analyze architectural approaches – Analyze each scenario, rating them by priority. The architecture is then evaluated against each scenario.~~
- ~~Brainstorm and prioritize scenarios – among the larger stakeholder group, present the current scenarios, and expand.~~
- ~~Analyze architectural approaches – Perform step 6 again with the added knowledge of the larger stakeholder community.~~
- ~~Present results – provide all documentation to the stakeholders.~~

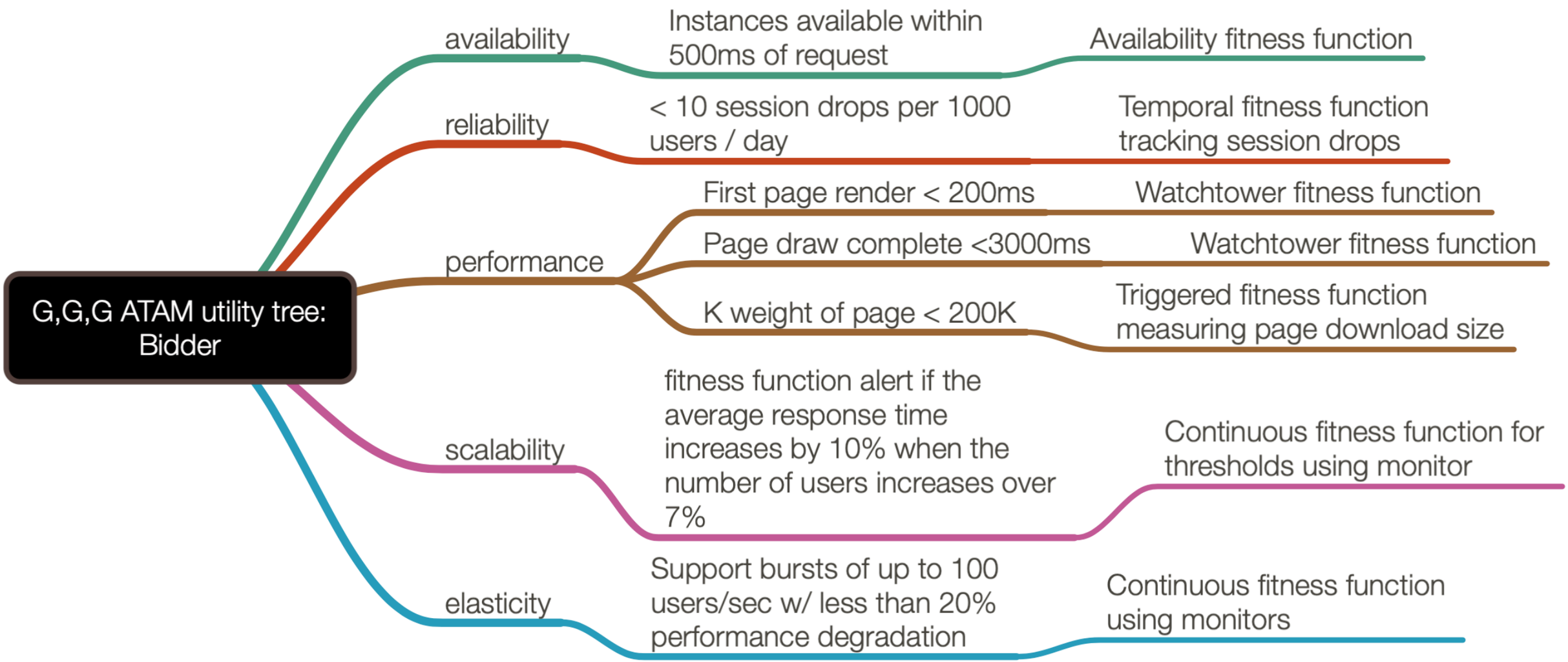
# ATAM Utility Trees



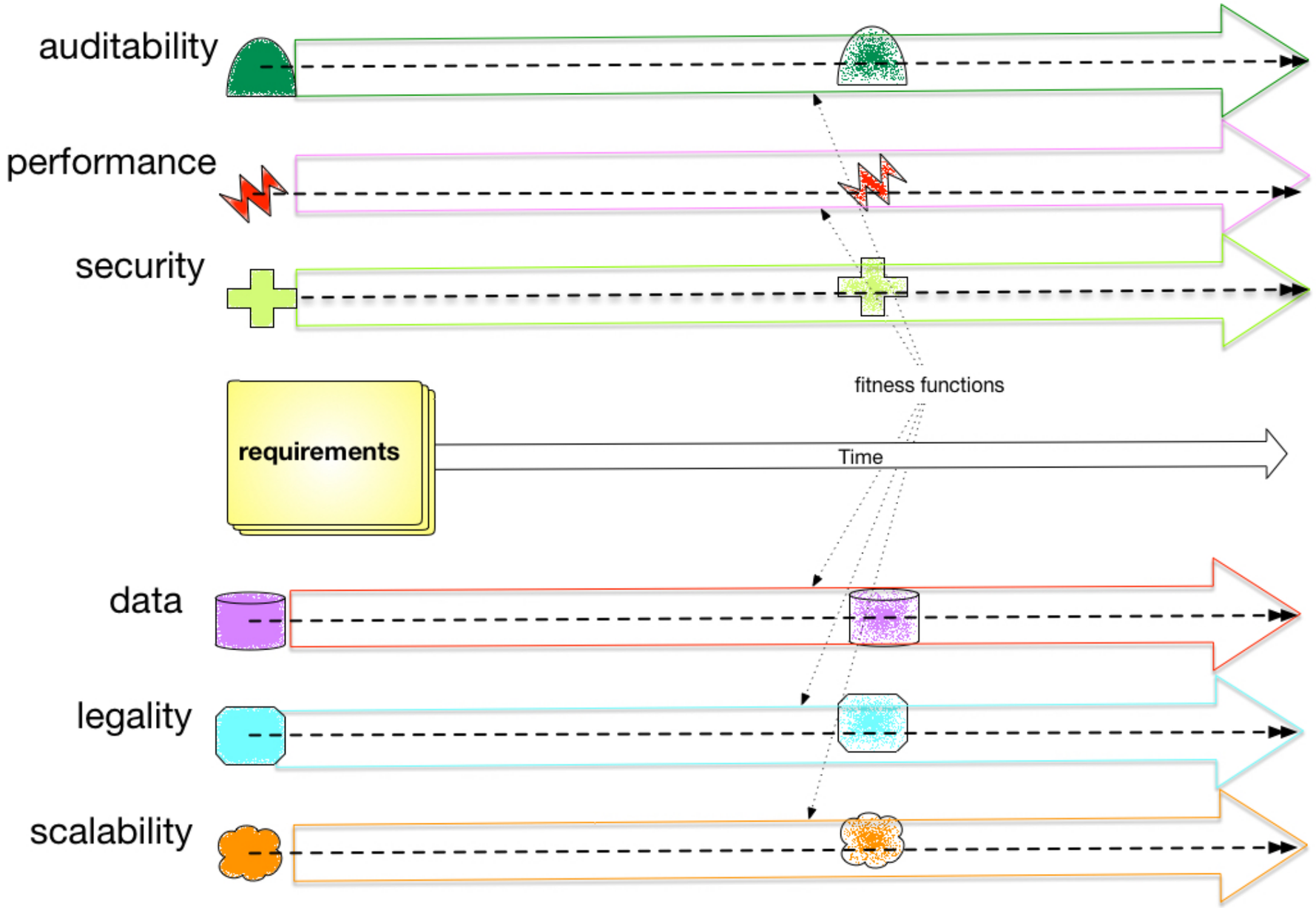
# Bidder Quantum ATAM Utility Tree

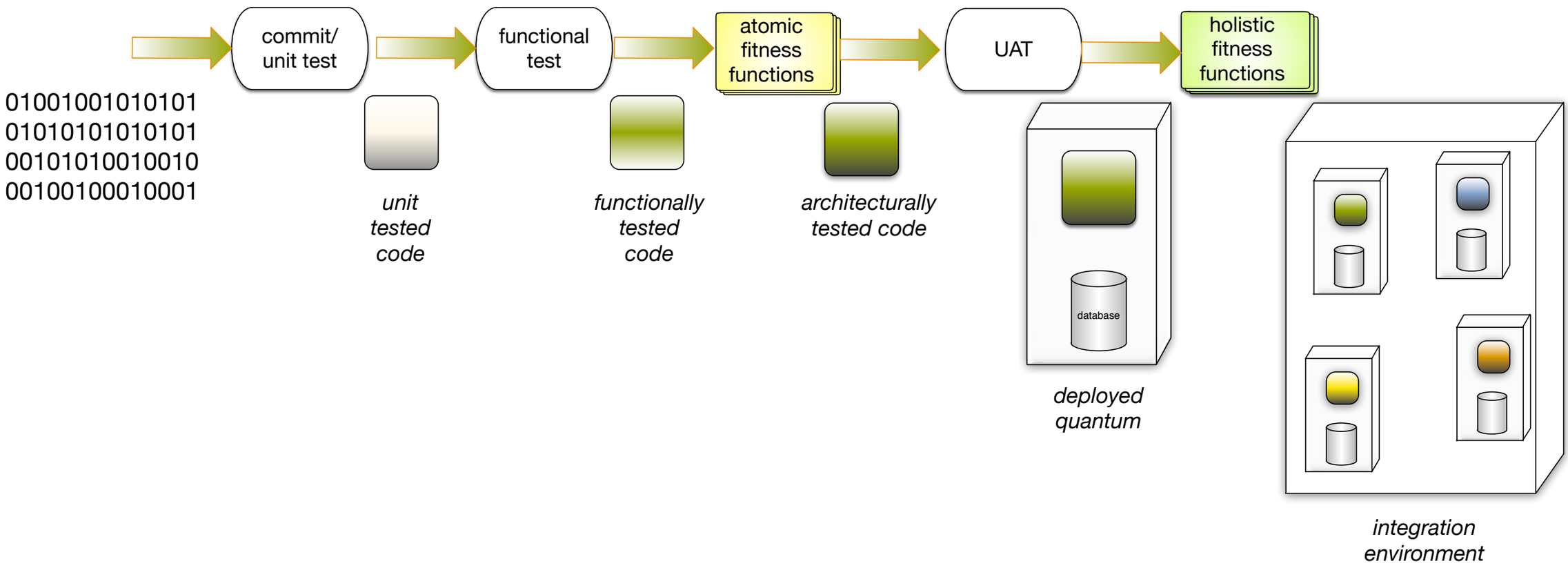


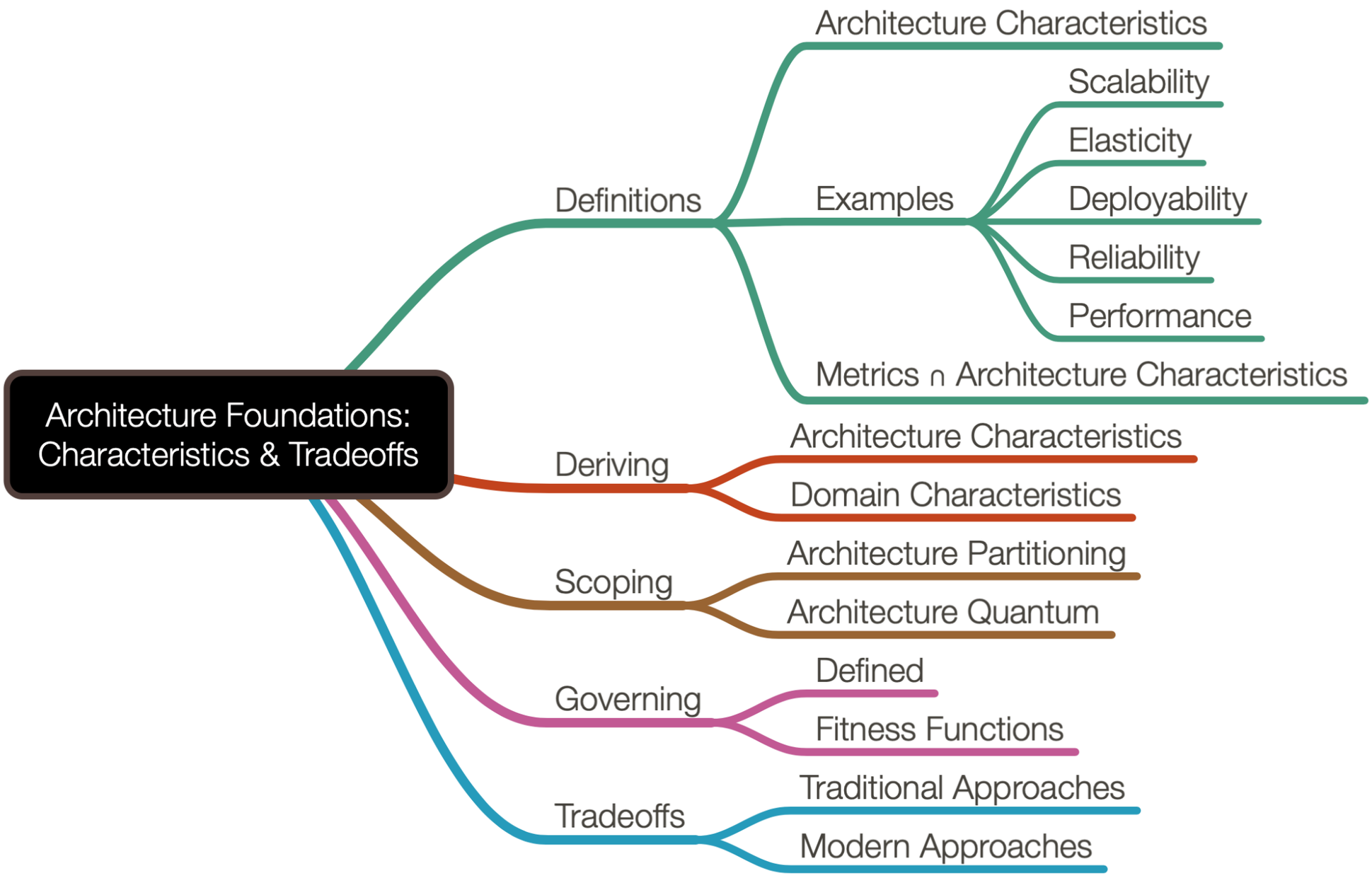
# Bidder Quantum ATAM Utility Tree Tradeoff Analysis



# fitness functions as concrete measure


auditability
performance
security
fitness functions
requirements
Time
data
legality
scalability


01001001010101
01010101010101
00101010010010
00100100010001
commit/
unit test
functional
test
atomic
fitness
functions
UAT
holistic
fitness
functions
unit
tested
code
functionally
tested
code
architecturally
tested code
database
deployed
quantum
integration
environment


Architecture Foundations: Characteristics & Tradeoffs
Definitions
Architecture Characteristics
Examples
Scalability
Elasticity
Deployability
Reliability
Performance
Metrics ∩ Architecture Characteristics
Deriving
Architecture Characteristics
Domain Characteristics
Scoping
Architecture Partitioning
Architecture Quantum
Governing
Defined
Fitness Functions
Tradeoffs
Traditional Approaches
Modern Approaches

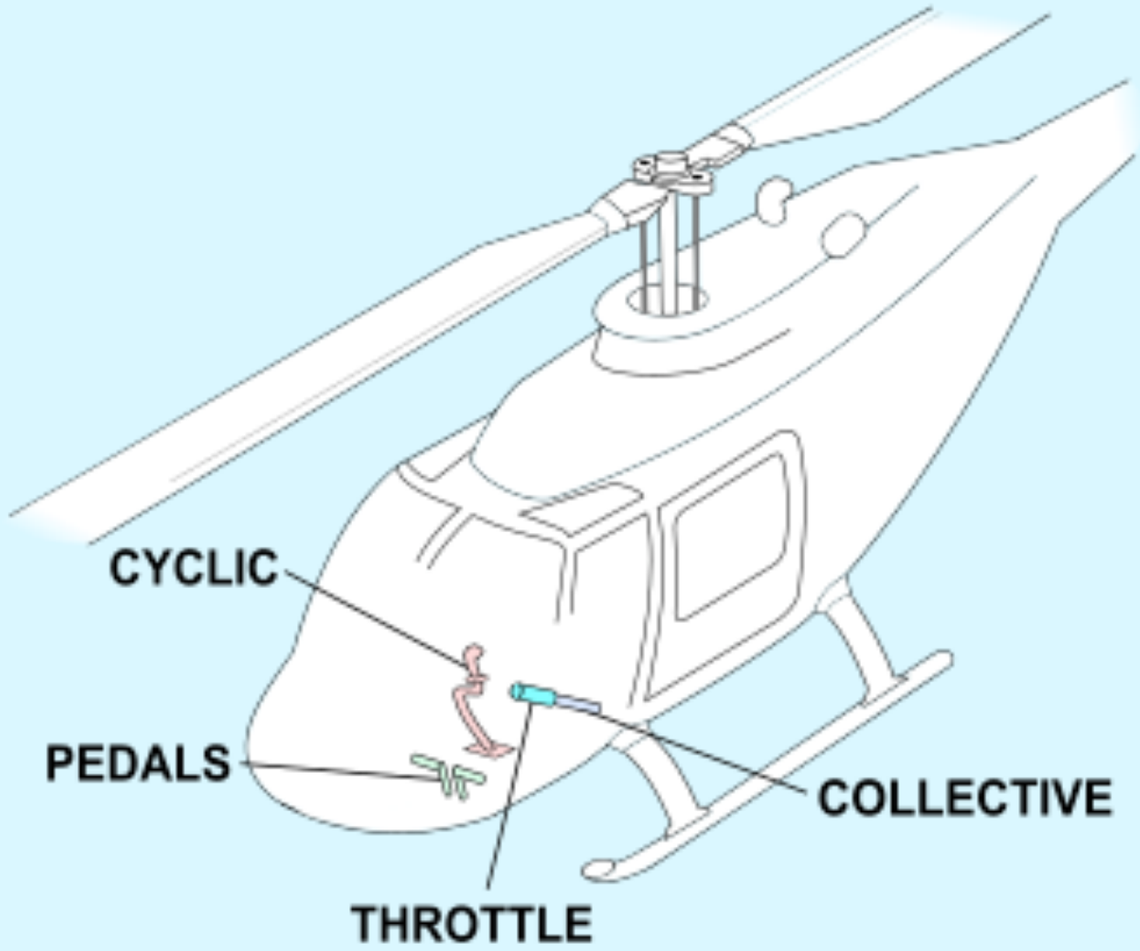
# Never best, but least worst.

Everything in software architecture is a tradeoff.

# Architecture Characteristics

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
evolvability
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
reliability
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


CYCLIC
PEDALS
COLLECTIVE
THROTTLE

# architecture characteristics

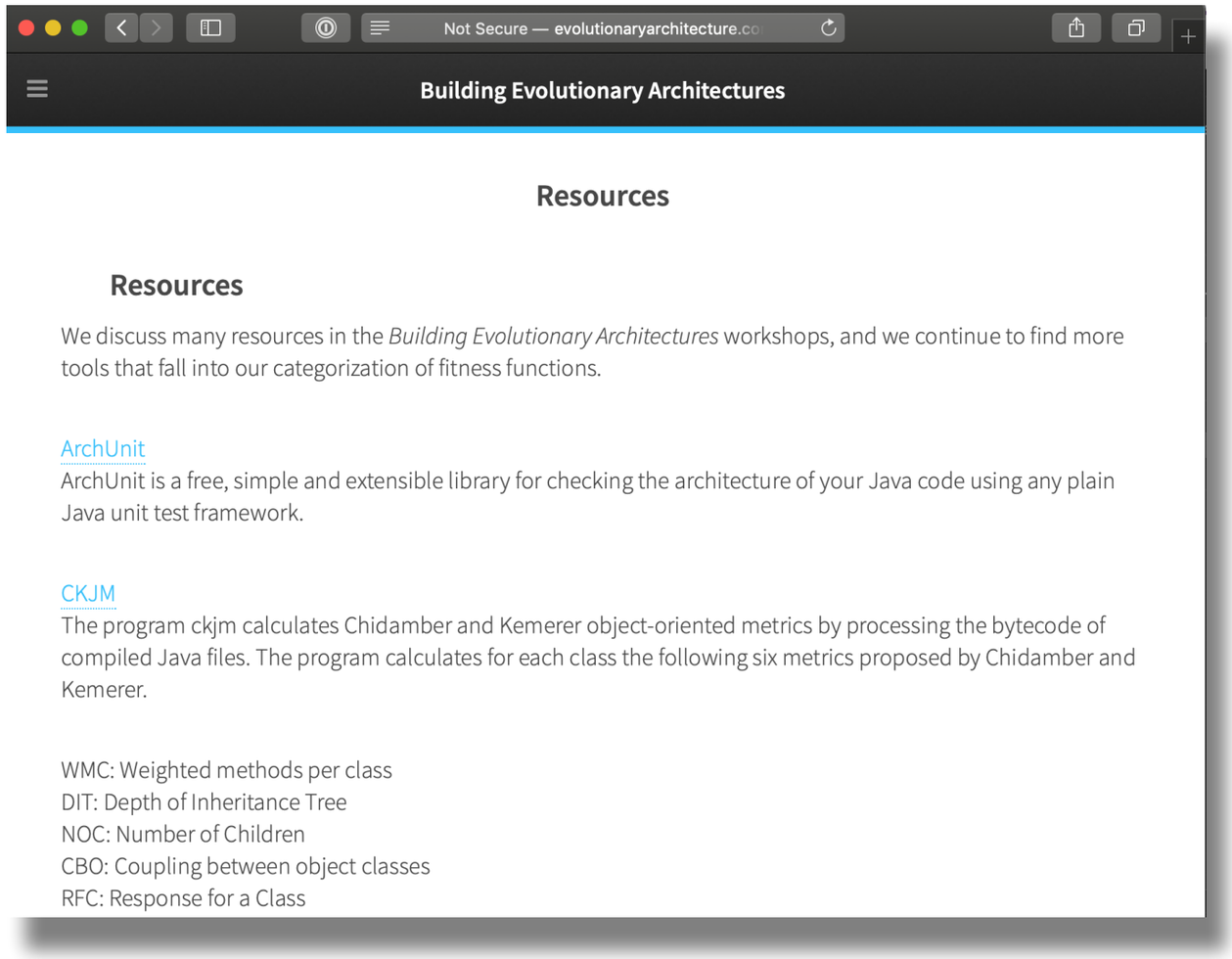
accessibility
accountability
accuracy
adaptability
administrability
affordability
agility
auditability
autonomy
availability
compatibility
composability
configurability
correctness
credibility
customizability
debugability
degradability
determinability
demonstrability
dependability
deployability
discoverability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

evolvability
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
reliability

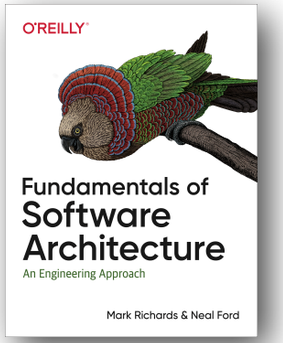
repeatability
reproducibility
resilience
responsiveness
reusability
robustness
safety
scalability
seamlessness
self-sustainability
serviceability
supportability
securability
simplicity
stability
standards compliance
survivability
sustainability
tailorability
testability
timeliness
traceability
transparency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

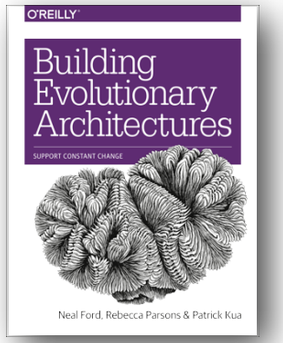
Don't overspecify

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

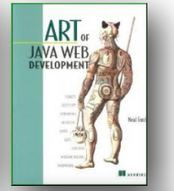


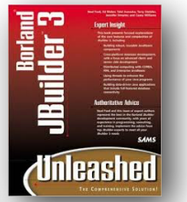

http://evolutionaryarchitecture.com/resources/



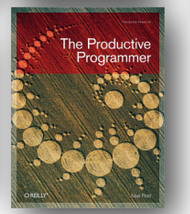



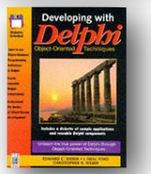


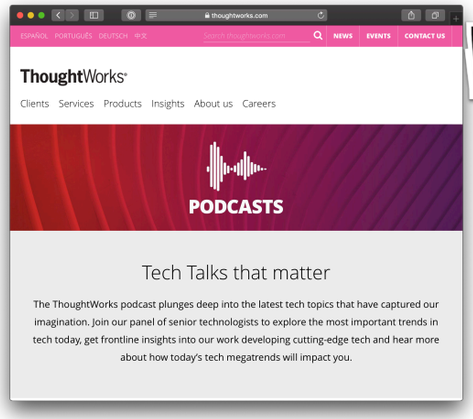


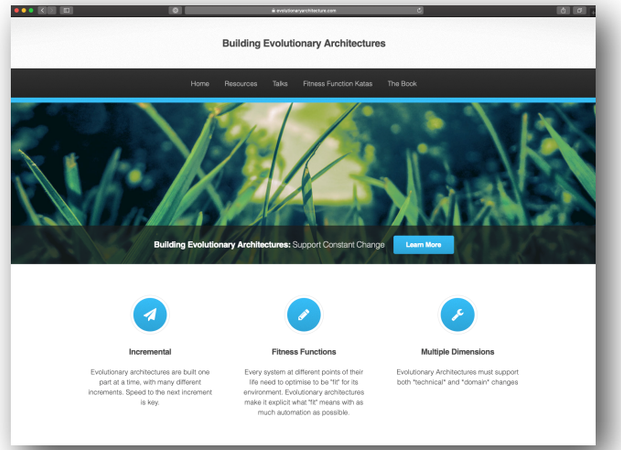




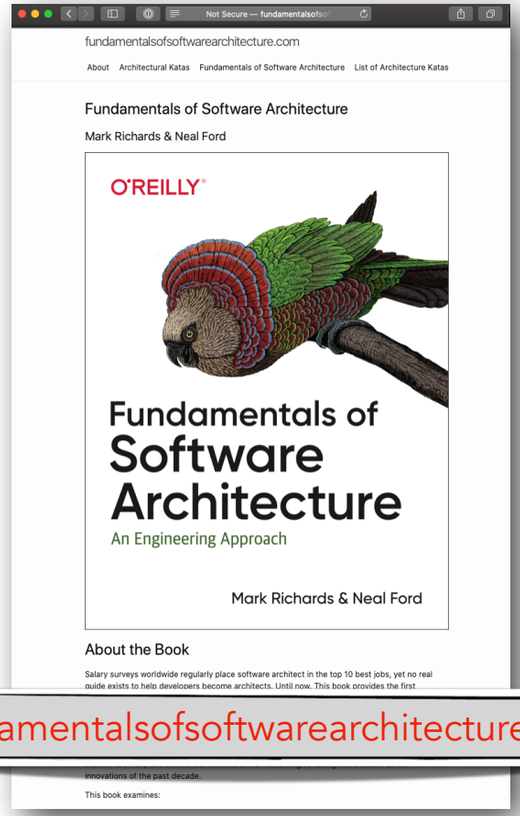



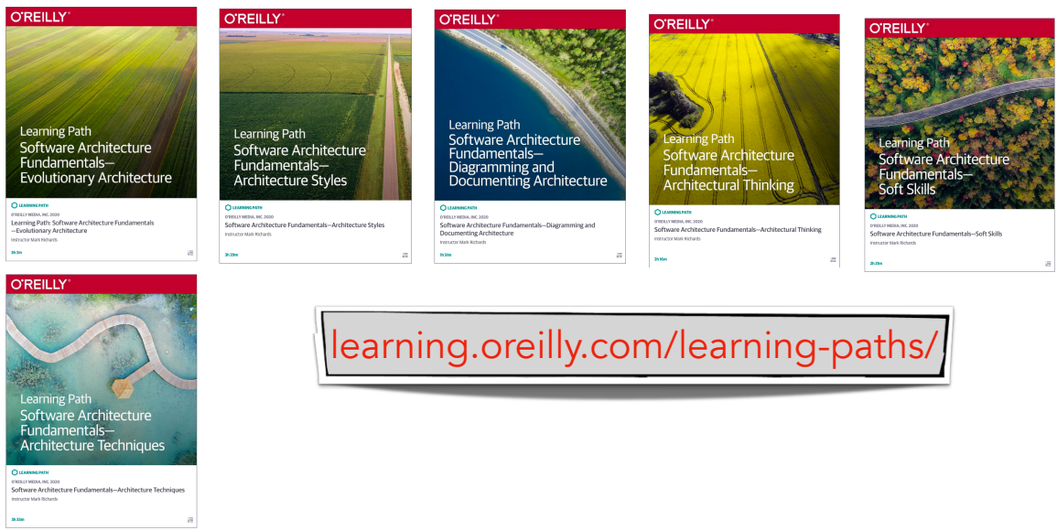




www.thoughtworks.com/podcasts




evolutionaryarchitecture.com


learning.oreilly.com/learning-paths/

fundamentalsofsoftwarearchitecture.com

ThoughtWorks®

**NEAL FORD**

*Director / Software Architect / Meme Wrangler*

@neal4d

http://nealford.com