Software Architecture Fundamentals Workshop

Part 2: A Deeper Dive

Mark Richards

Independent Consultant

Hands-on Enterprise / Integration Architect

Published Author / Conference Speaker

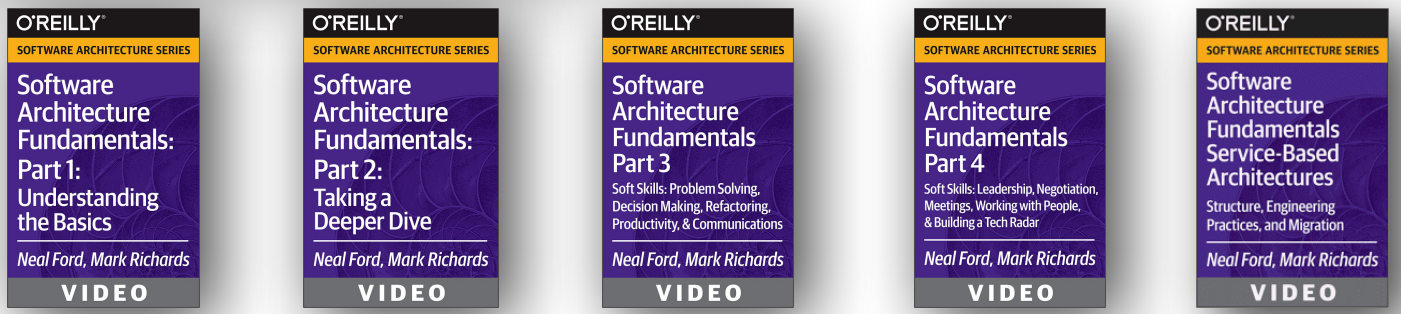
http://www.wmrichards.com

http://www.linkedin.com/pub/mark-richards/0/121/5b9

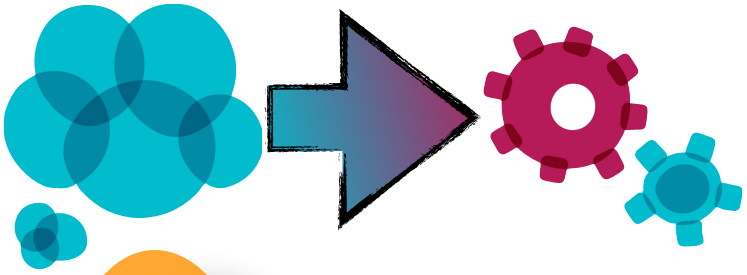
ThoughtWorks®

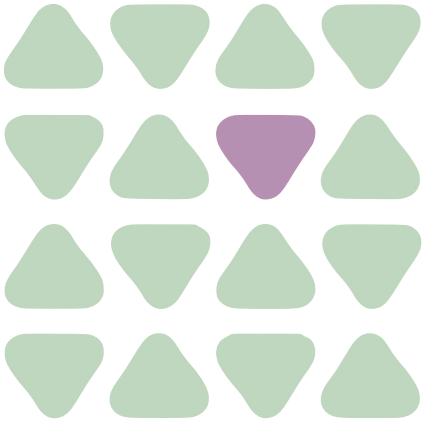
NEAL FORD

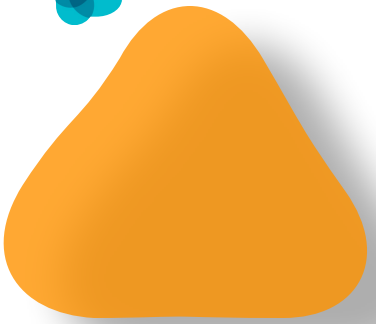
Director / Software Architect / Meme Wrangler

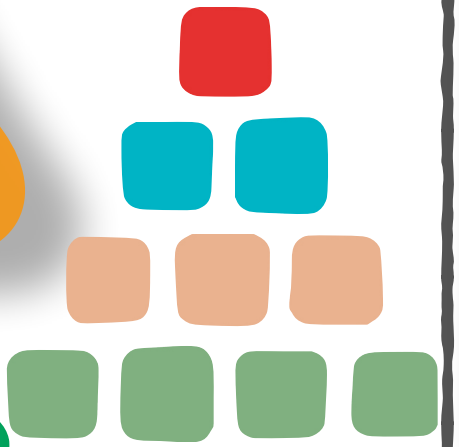


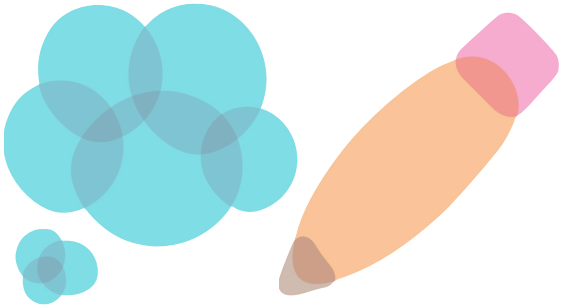
agenda

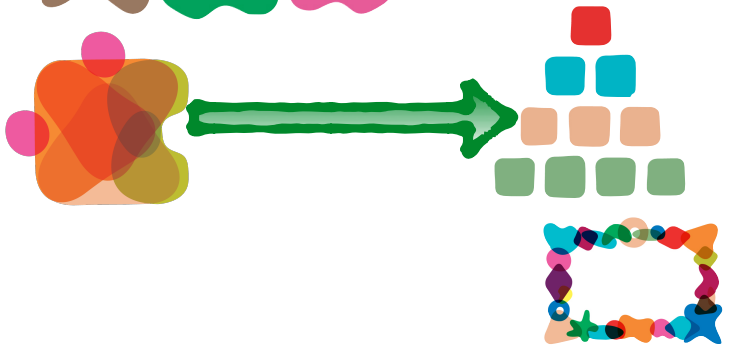


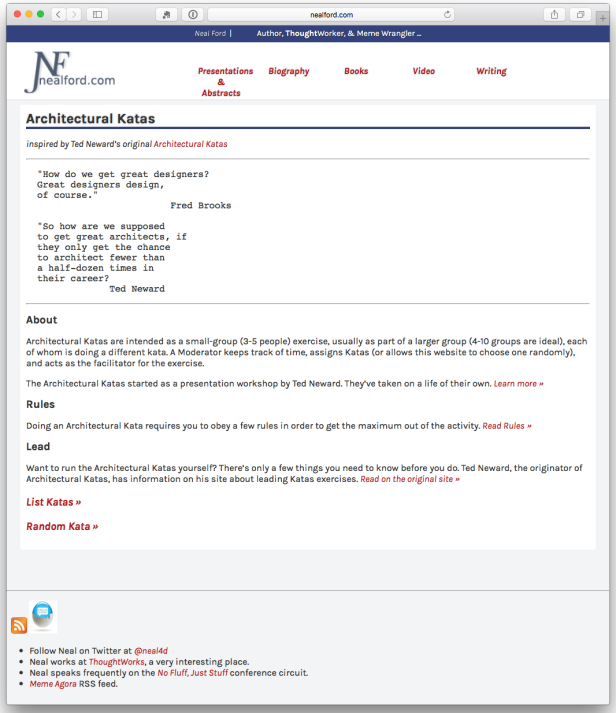






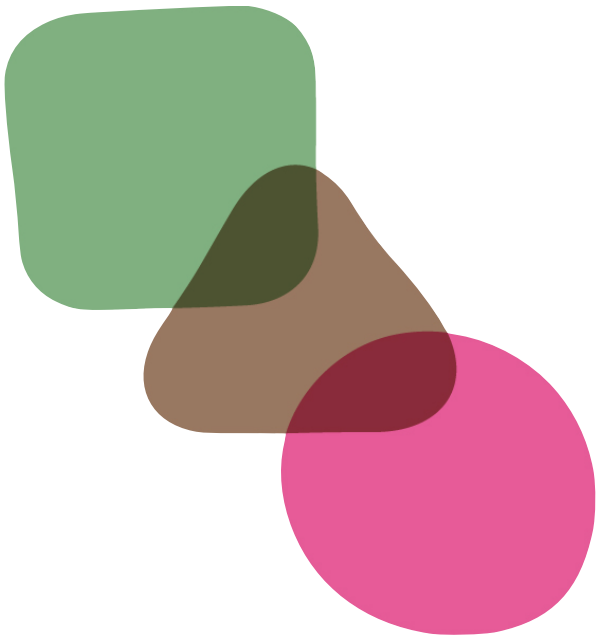




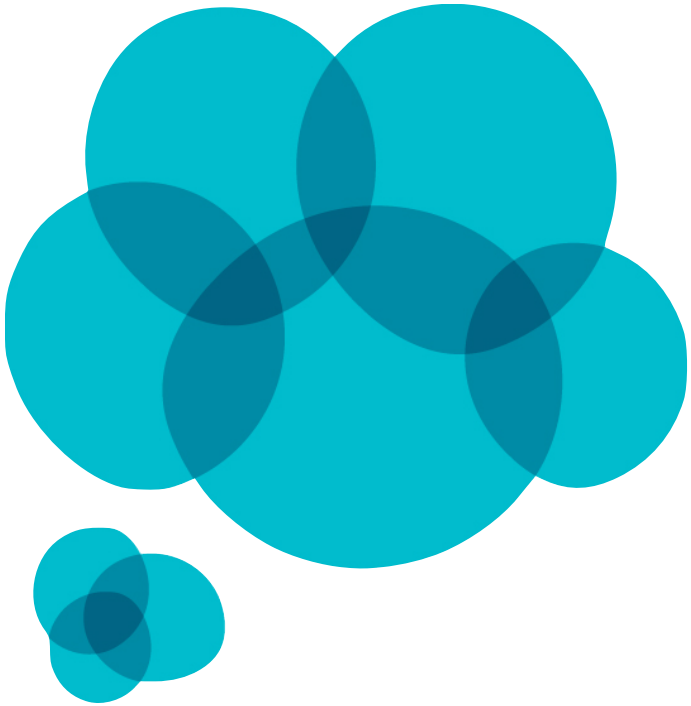


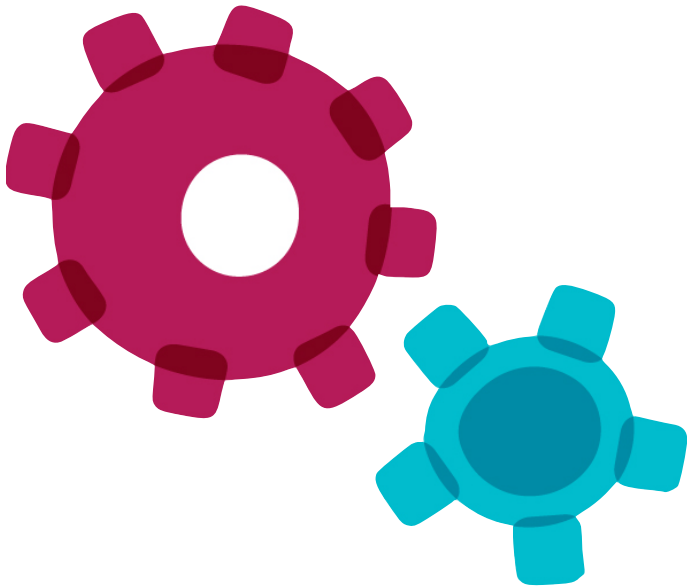
nealford.com/katas/

Software architecture reflects the mapping between capabilities and constraints.



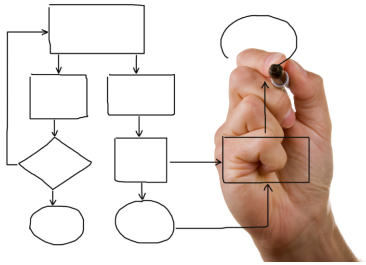
translating requirements





translating requirements

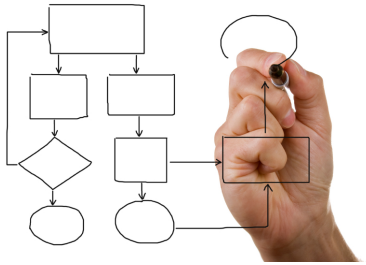
"our business is constantly changing to meet new demands of the marketplace"



???

translating requirements

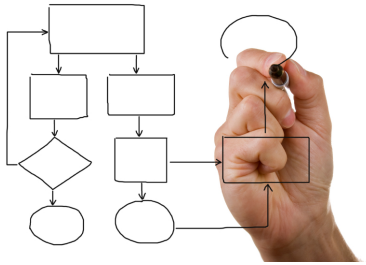
"due to new regulatory requirements, it is imperative that we complete end-of-day processing in time"



???

translating requirements

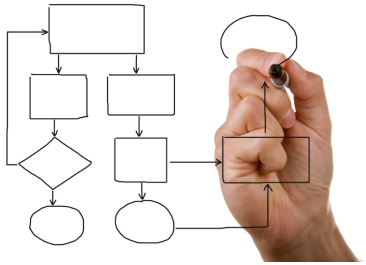
"we need faster time to market to remain competitive"



???

translating requirements

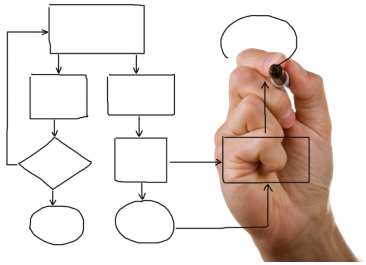
"our plan is to engage heavily in mergers and acquisitions in the next three years"



???

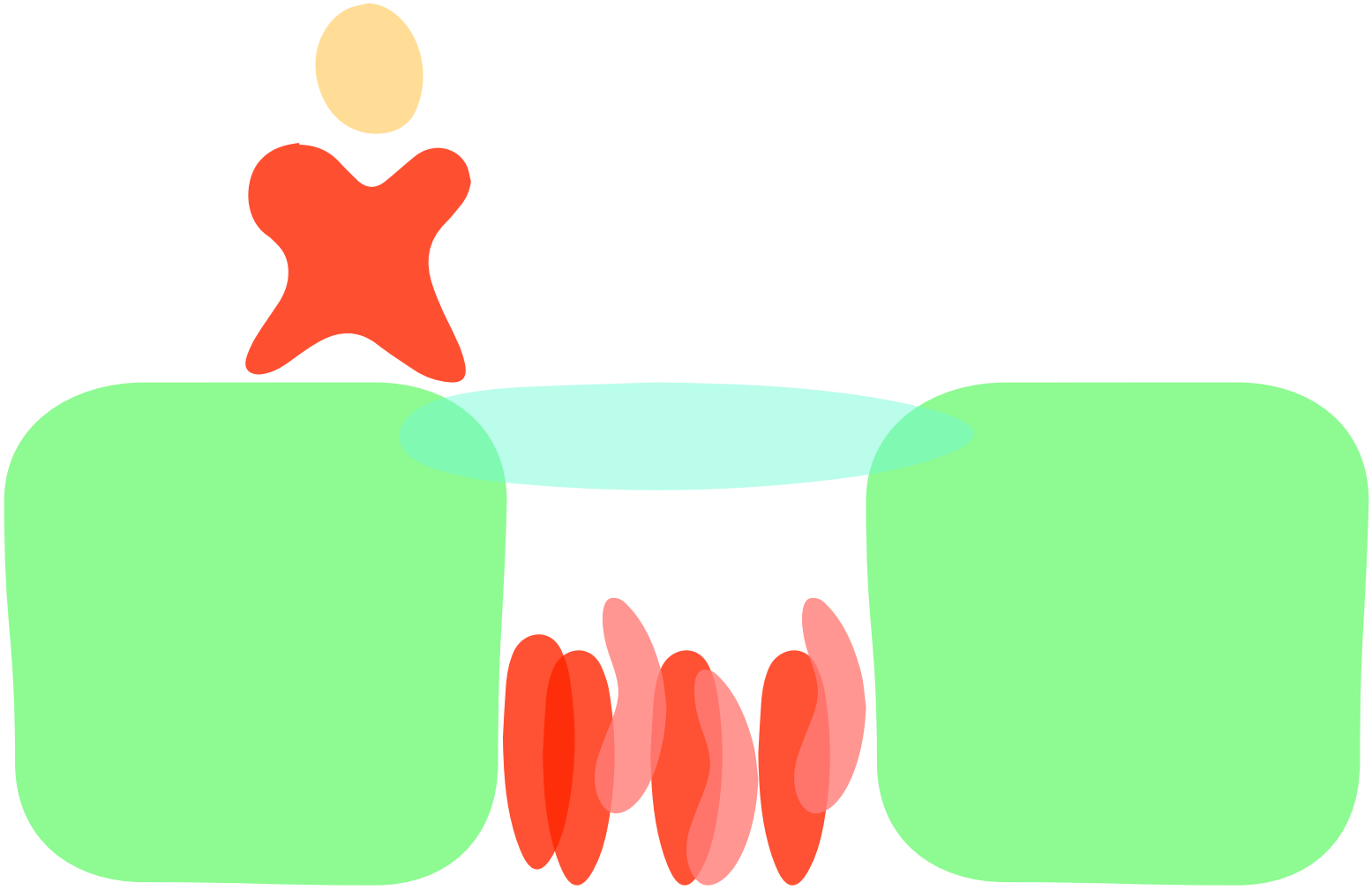
translating requirements

"we have a very tight timeframe and budget for this project"



???

architecture pitfall



armchair architecture

whiteboard sketches are handed off as final architecture standards without proving out the design

armchair architecture

occurs when you have non-coding architects

occurs when architects are not involved in the full project lifecycle

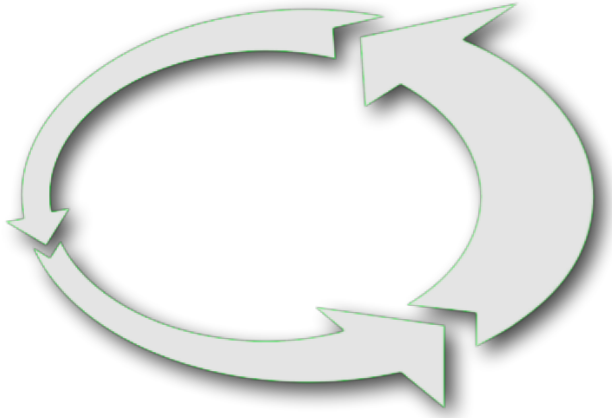
occurs when architects don't know what they are doing

architecting for change

architecting for change

business is in a constant state of change

increased competition



regulatory changes

mergers

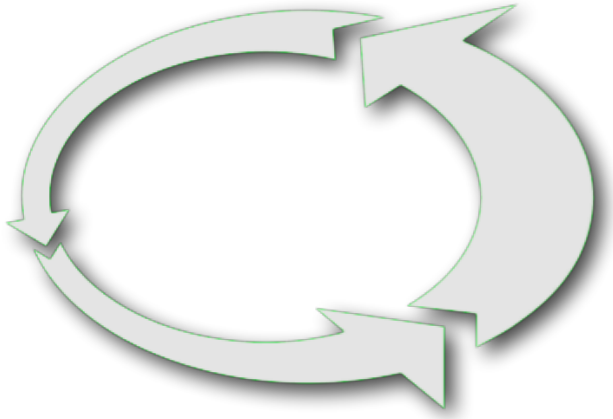
growth

acquisitions

architecting for change

technology is in a constant state of change

platforms



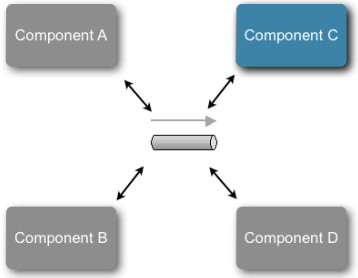
products

languages

patterns

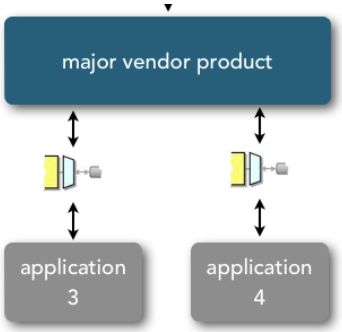
frameworks

techniques for change

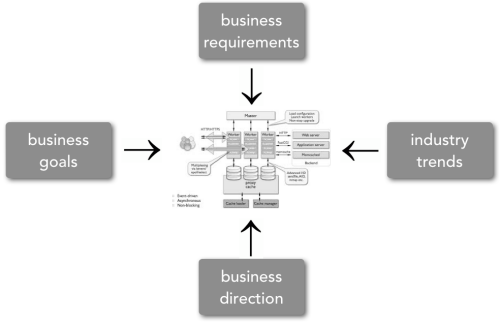


reduce dependencies

leverage standards

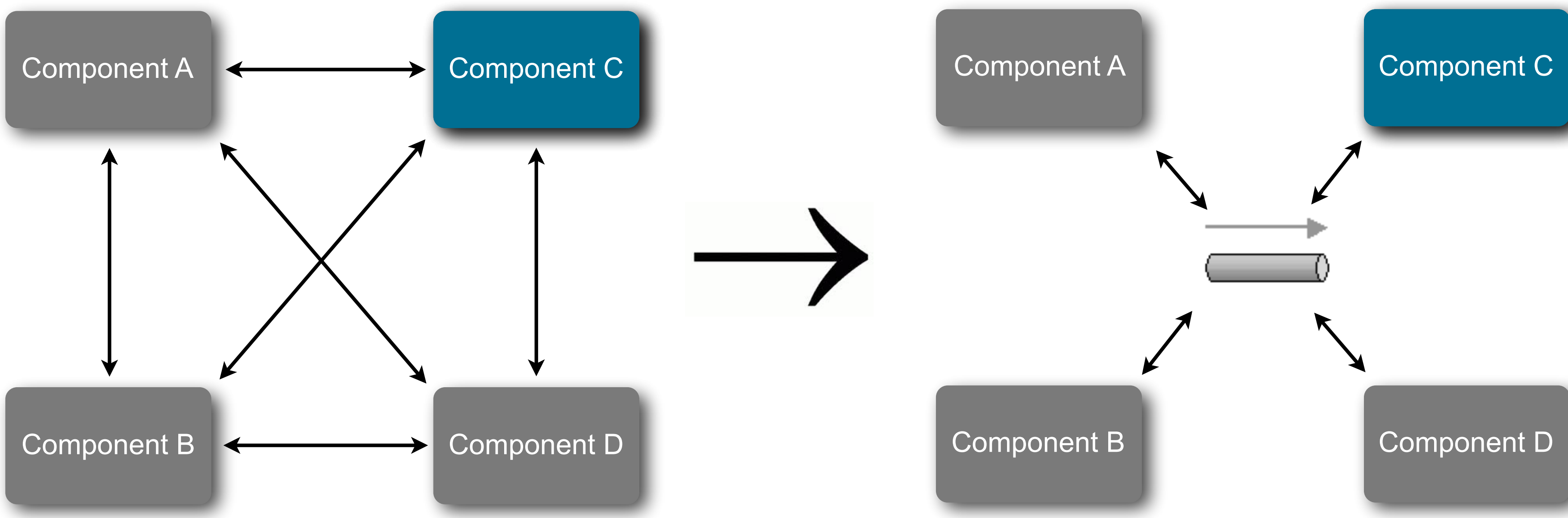


create product-agnostic architectures



create domain-specific architectures

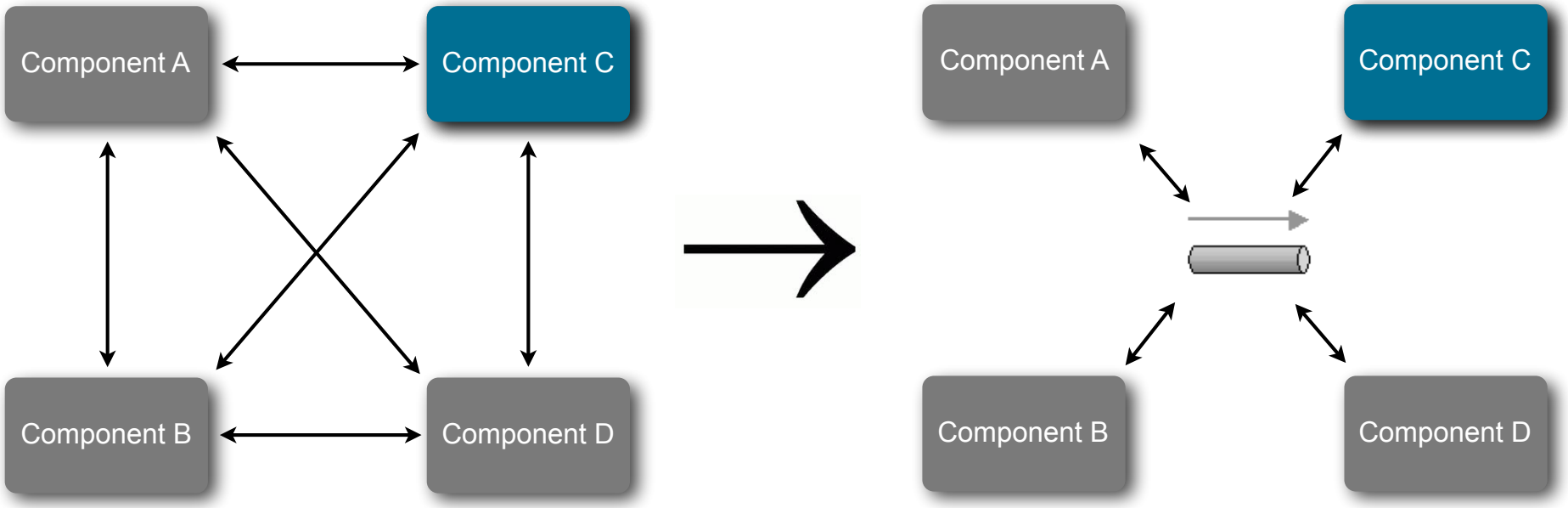
reduce dependencies



less coordination required (release schedules)

components can evolve independently

reduce dependencies



messaging service bus adapters

architecture patterns

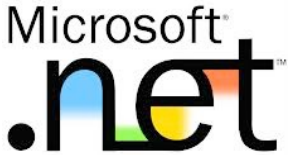
leverage standards

industry standards

<?xml?>

leverage standards

corporate standards



leverage standards

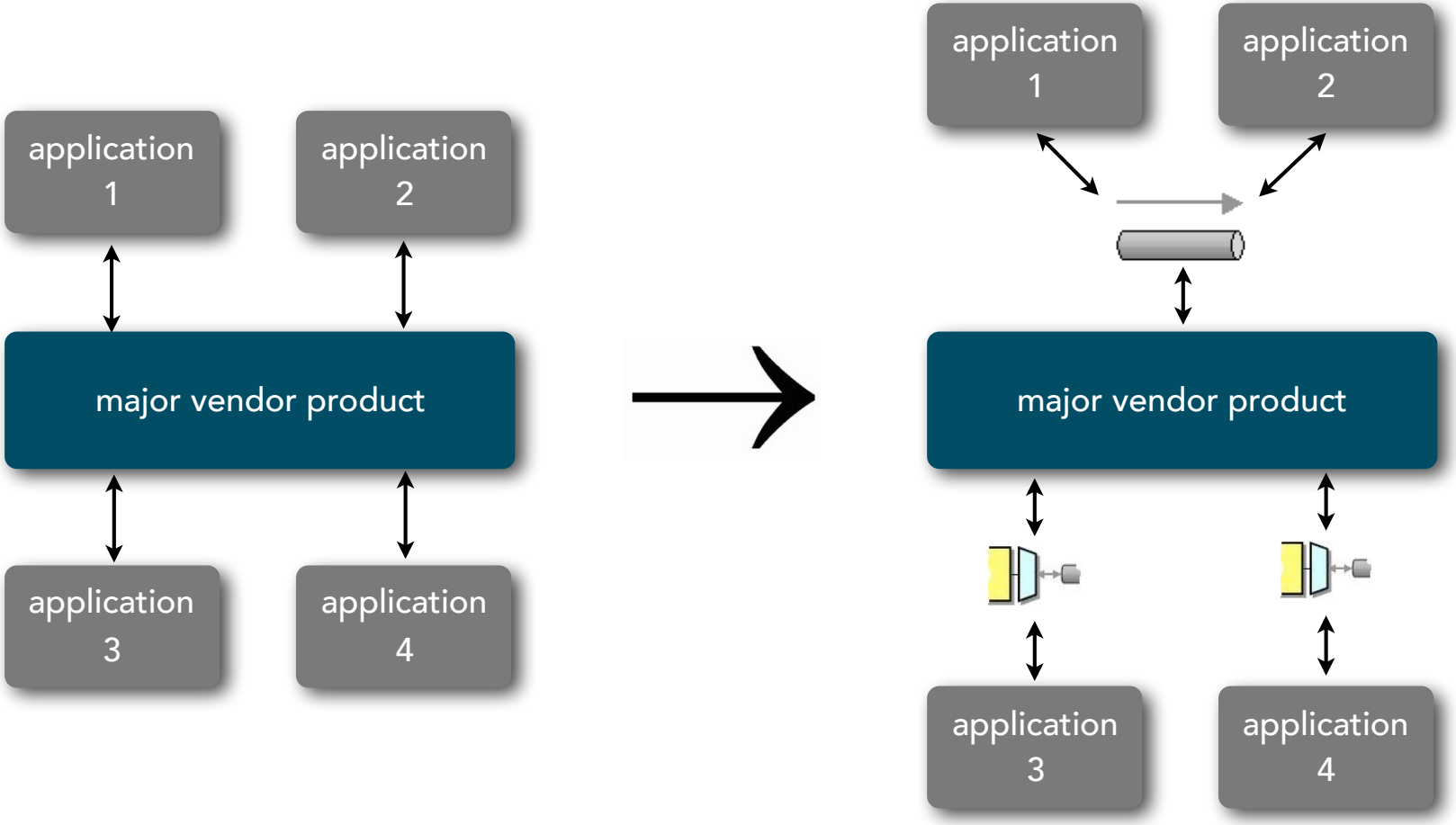
standards may not always be your first choice, but they significantly help in reducing the effort for change

larger resource pool

better integration with other systems

product-agnostic architecture

isolate products to avoid vendor lock-in

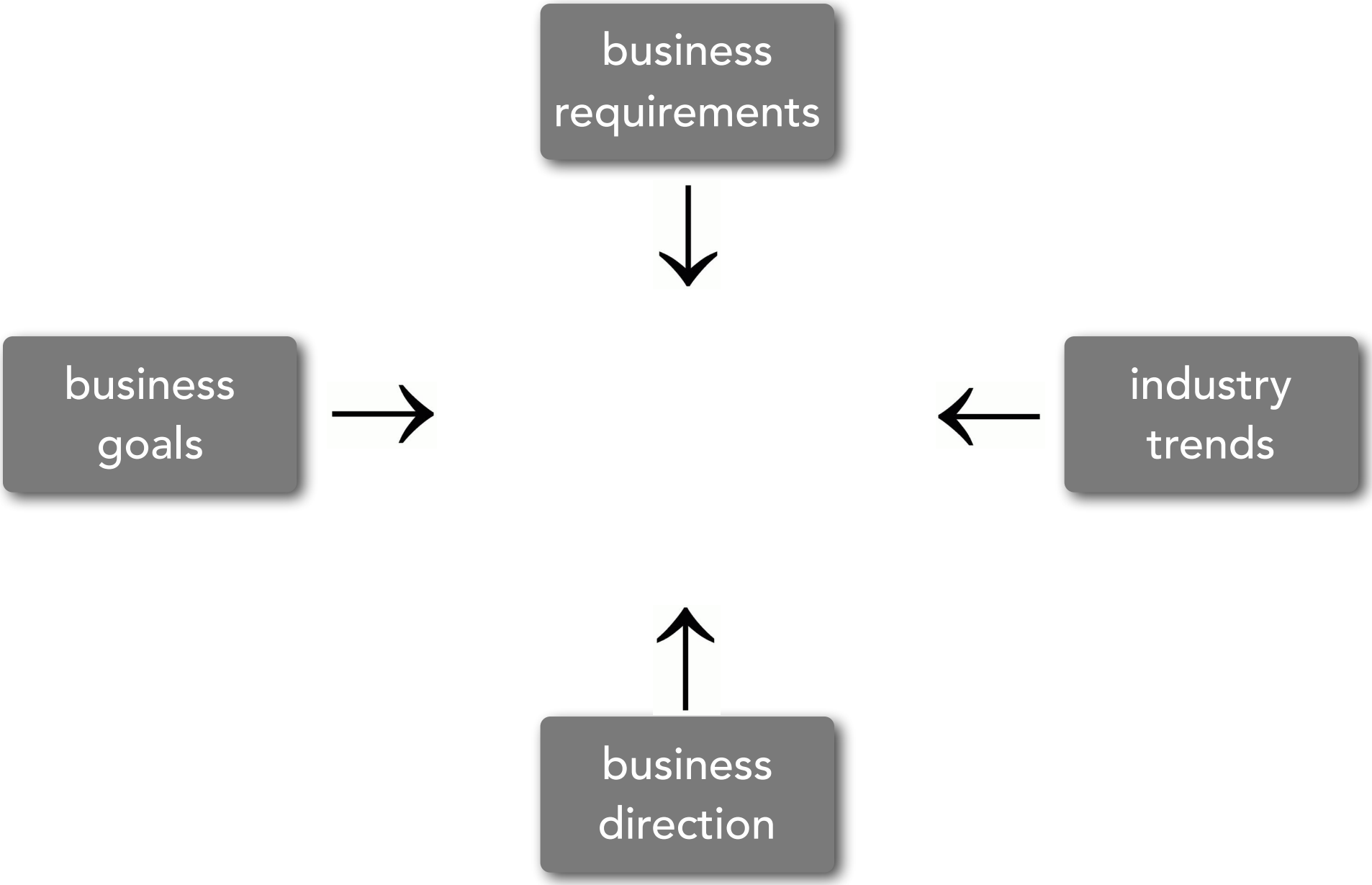


domain-specific architecture

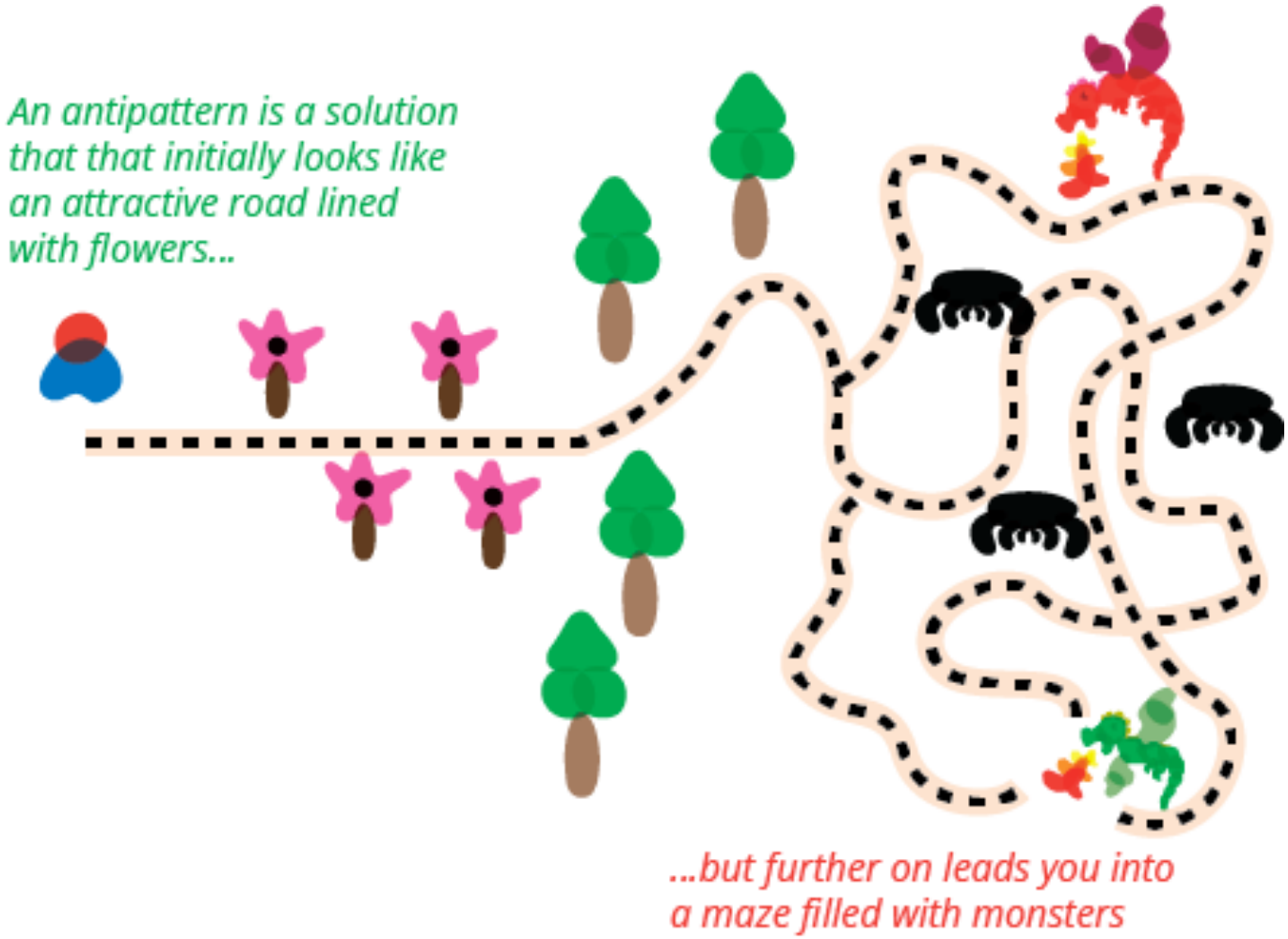
generic architectures are difficult to change because they are too broad and take into account scenarios that aren't used

limit the scope of the architecture by taking into account drivers, requirements, business direction, and industry trends

domain-specific architecture



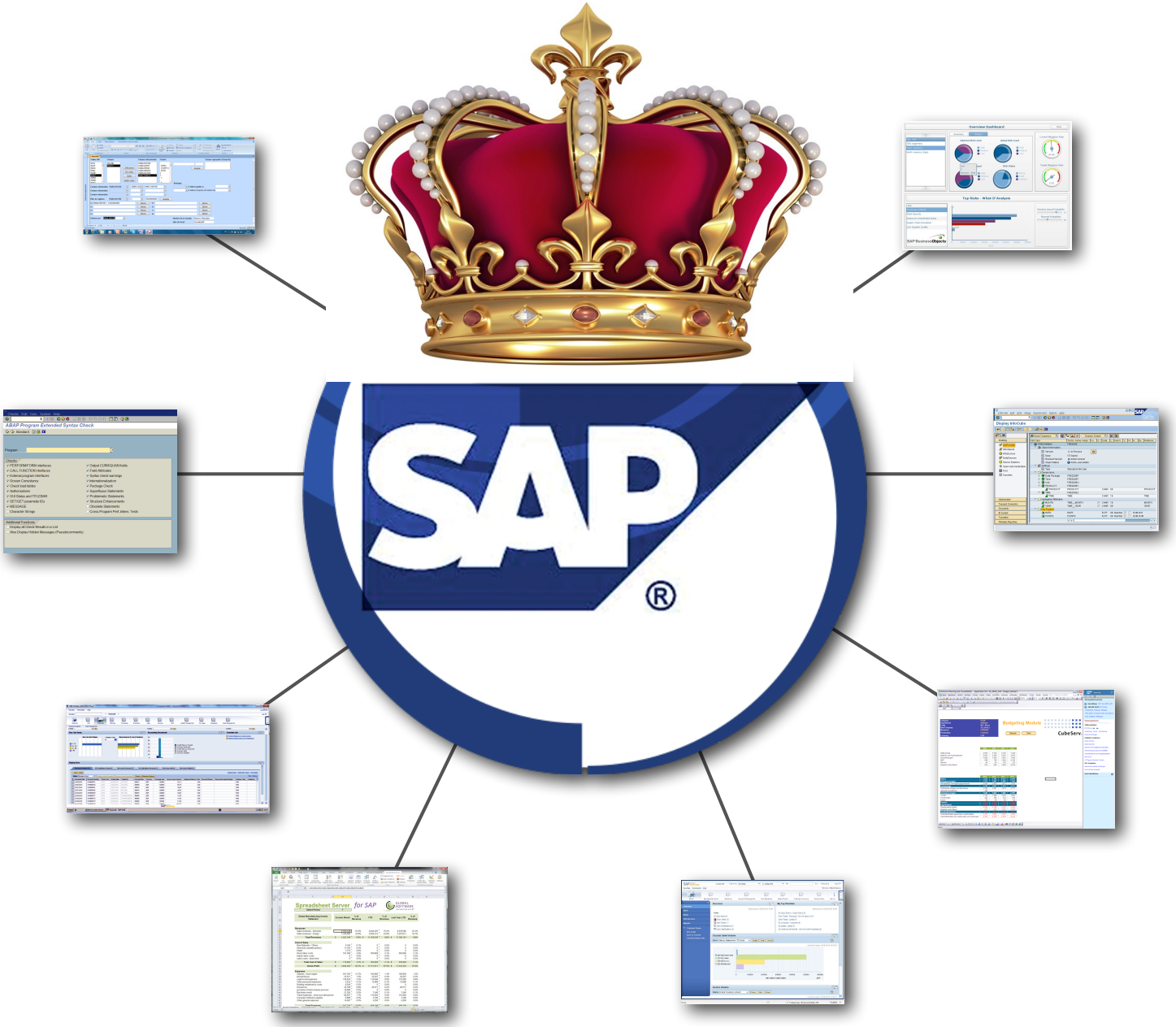
Architecture Anti-pattern



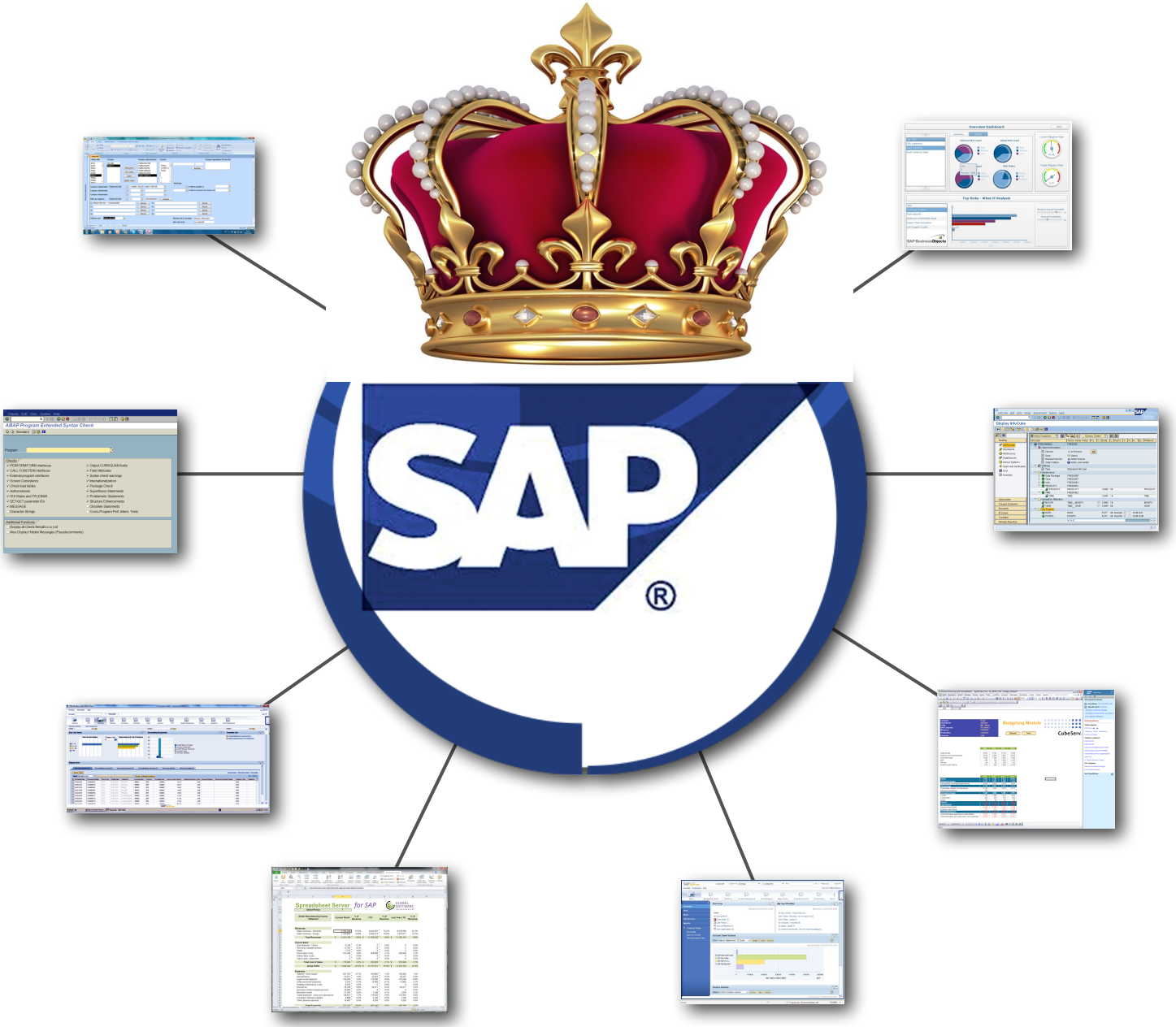
vendor king

product-dependent architectures leading to a loss of control of architecture and development costs

vendor king


SAP
ABAP Program Extended Syntax Check
Spreadsheet Server for SAP
Budgeting Module
CubeServ

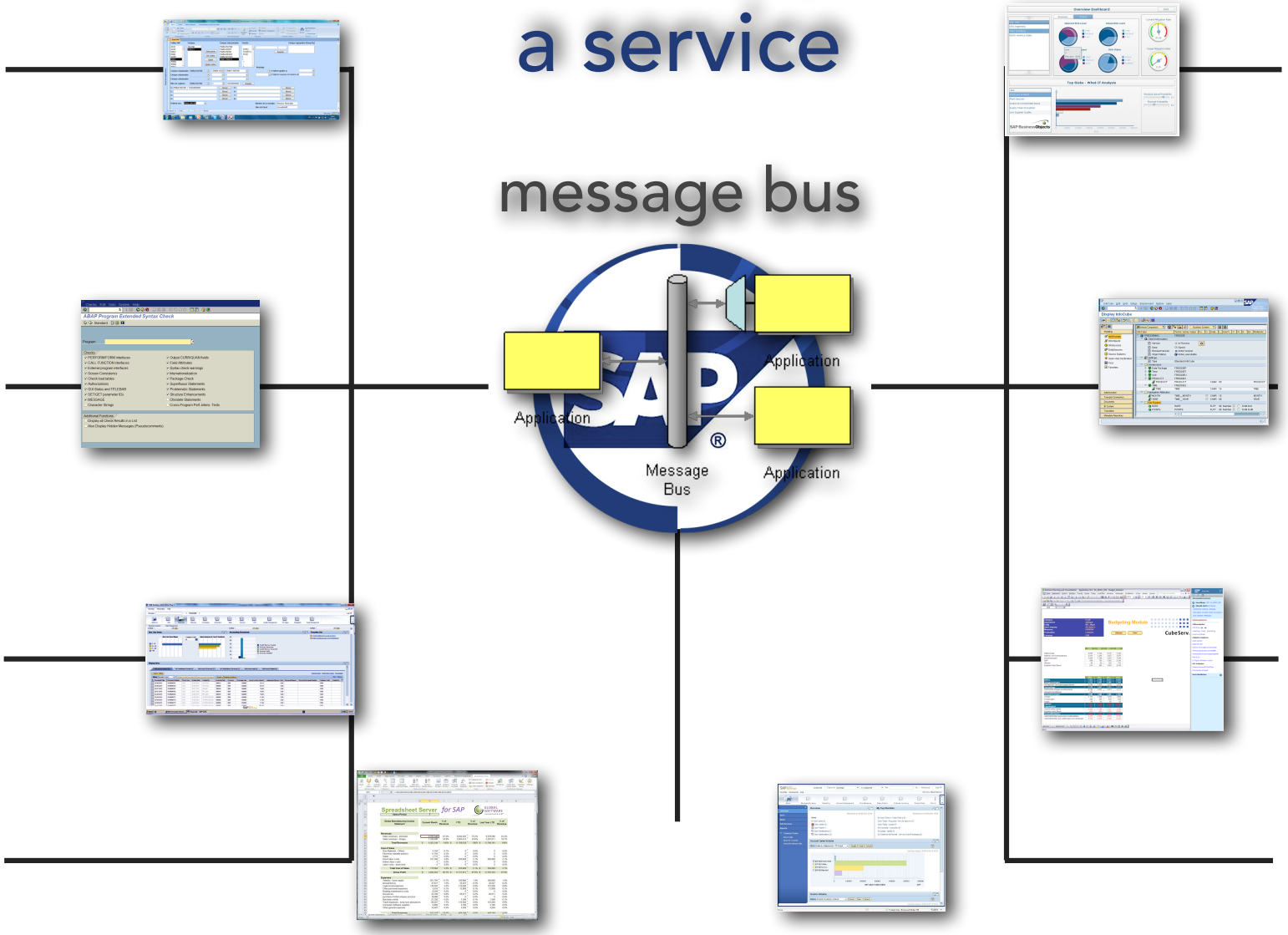
vendor king


SAP
ABAP Program Extended Syntax Check
Spreadsheet Server for SAP
Budgeting Module
CubeServ

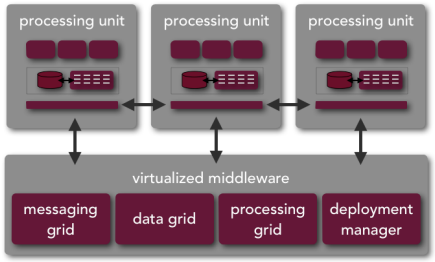
vendor king

vendor app as a service

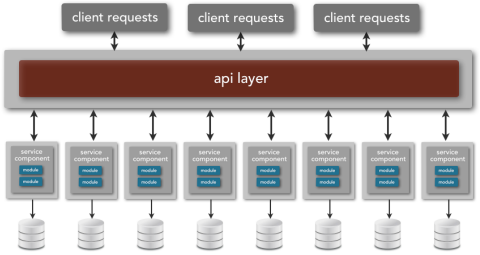
message bus



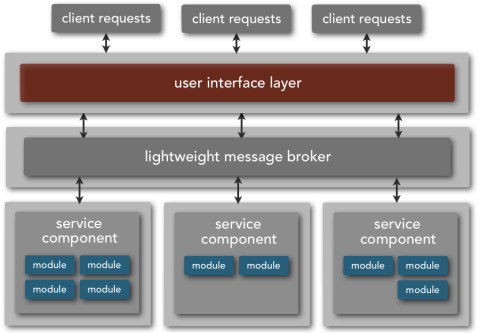
(More) Architecture Patterns



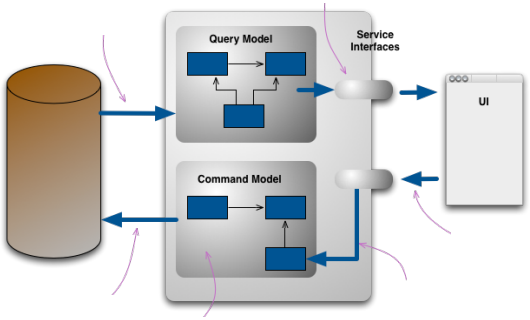
space-based architecture



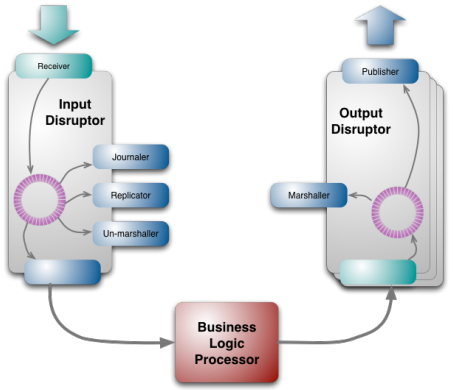
microservices architecture



service-based architectures



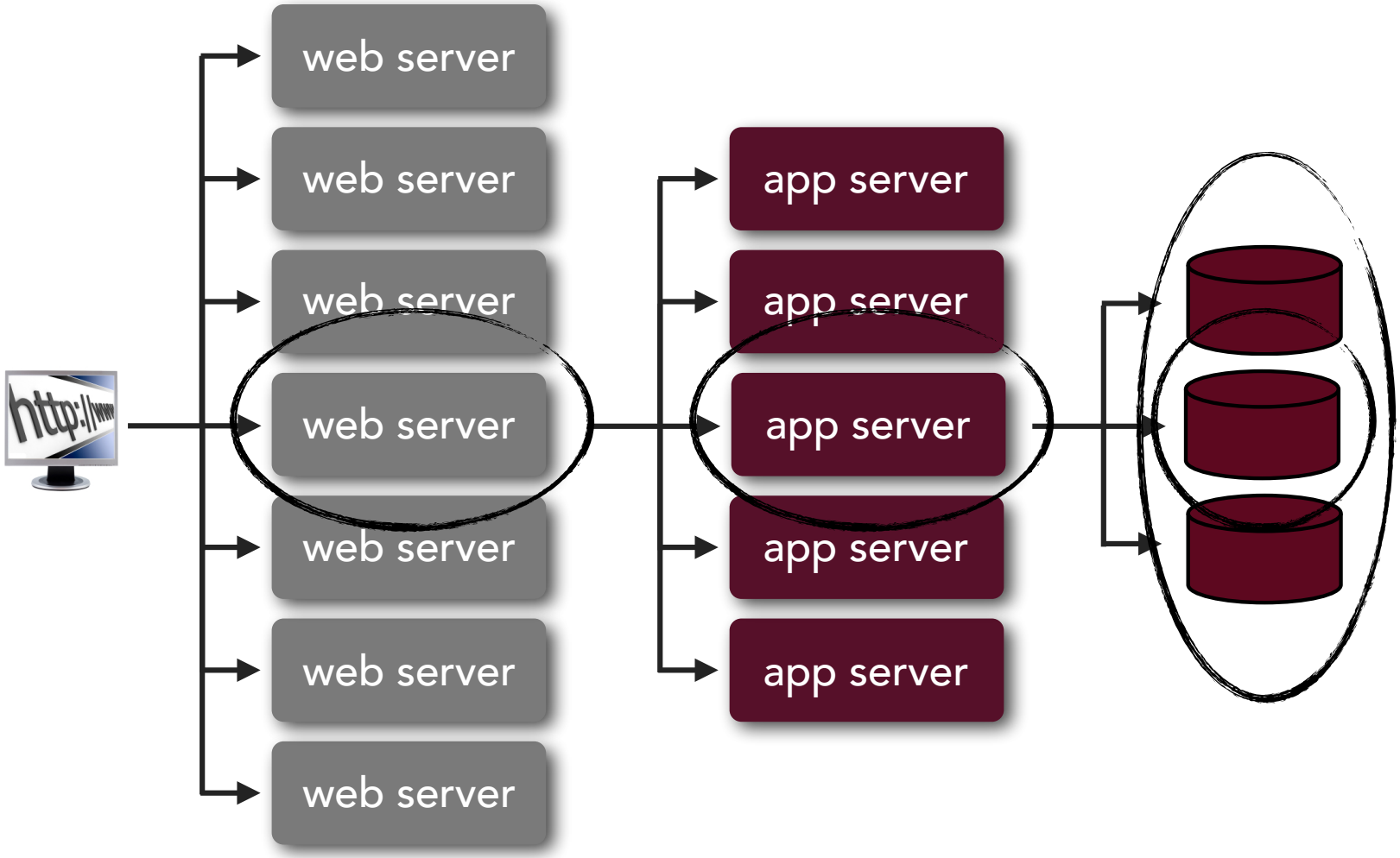
CQRS



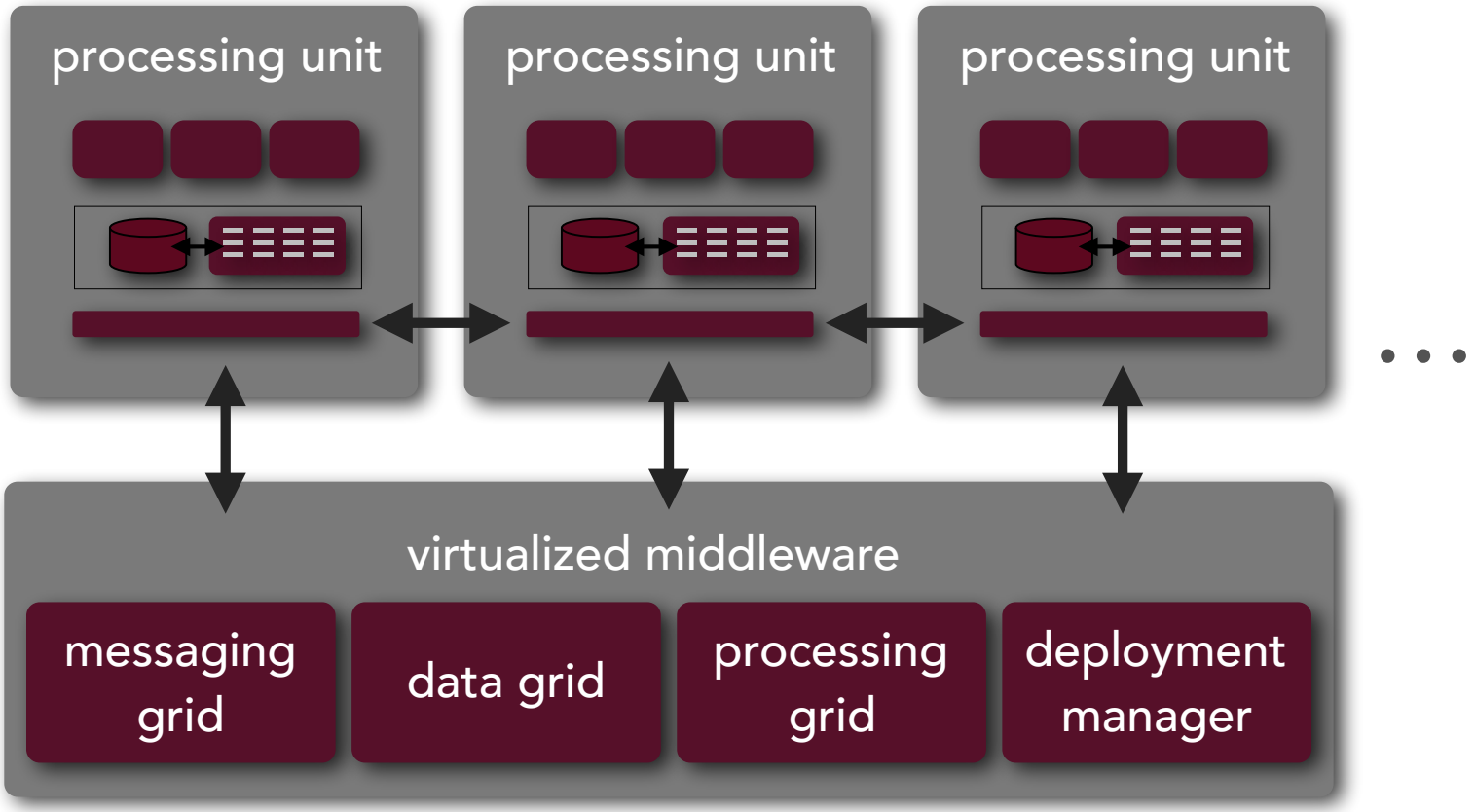
LMAX

space-based architecture

let's talk about scalability for a moment...

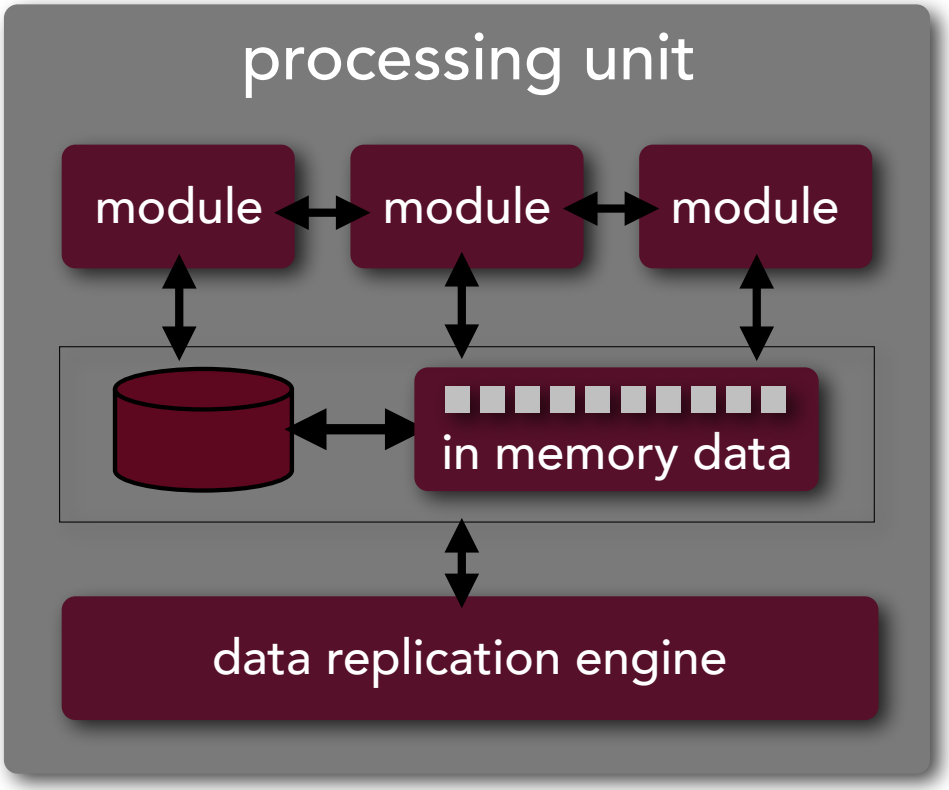


space-based architecture



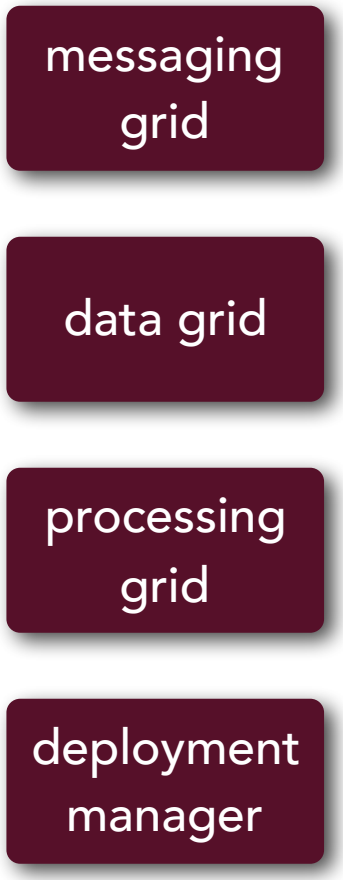
space-based architecture

processing unit



space-based architecture

middleware

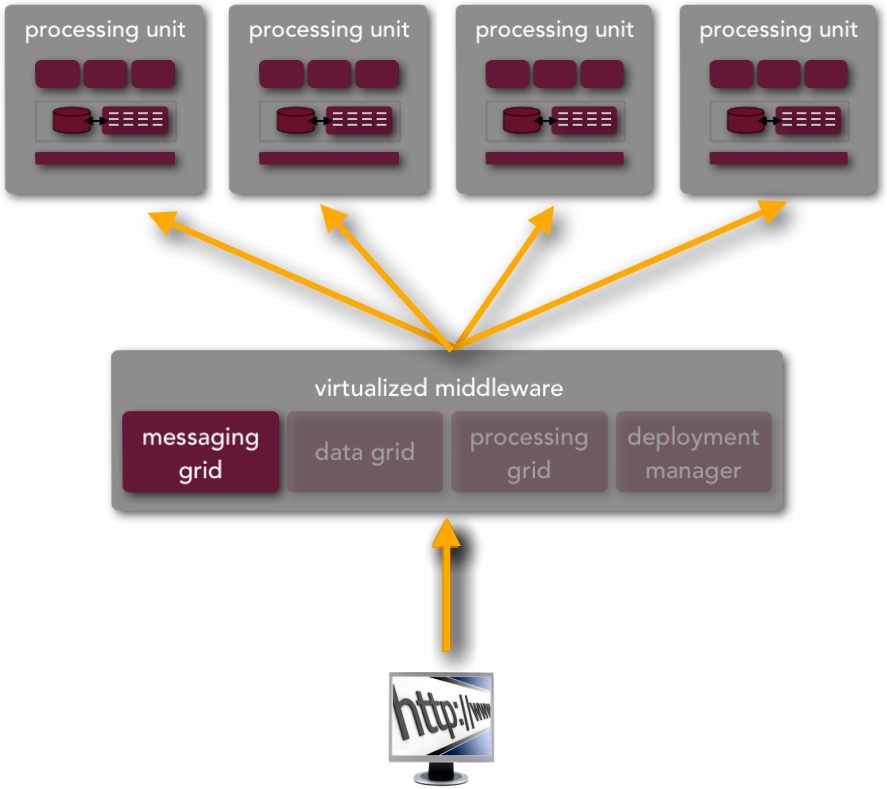


space-based architecture

middleware

- messaging grid
- data grid
- processing grid
- deployment manager

manages input request and session



space-based architecture

middleware

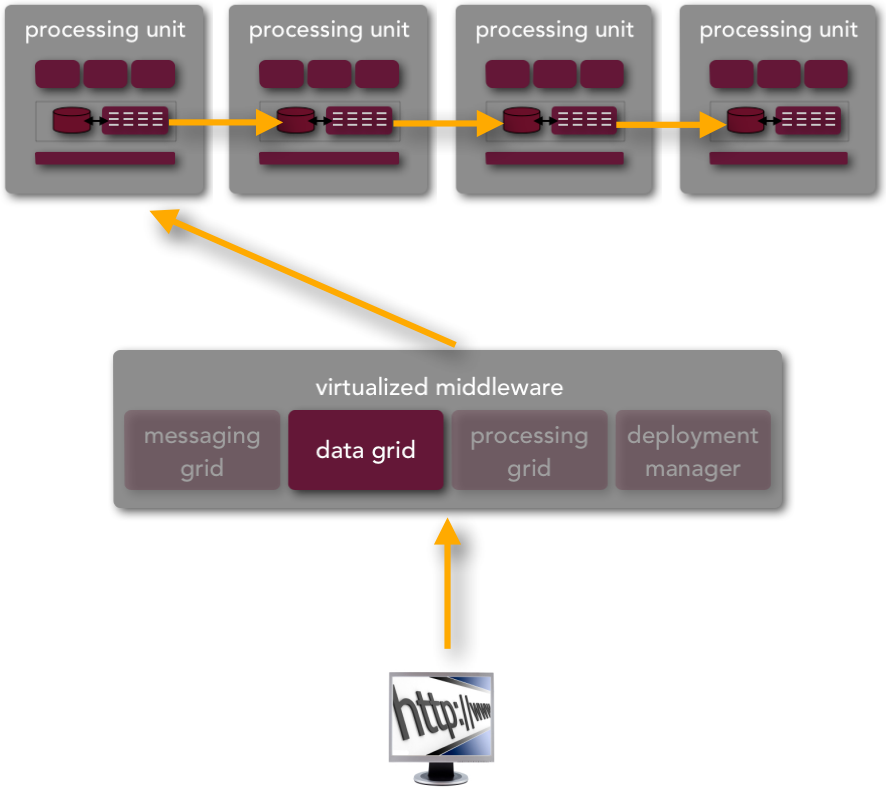
messaging grid

data grid

processing grid

deployment manager

manages data replication between processing units



space-based architecture

middleware

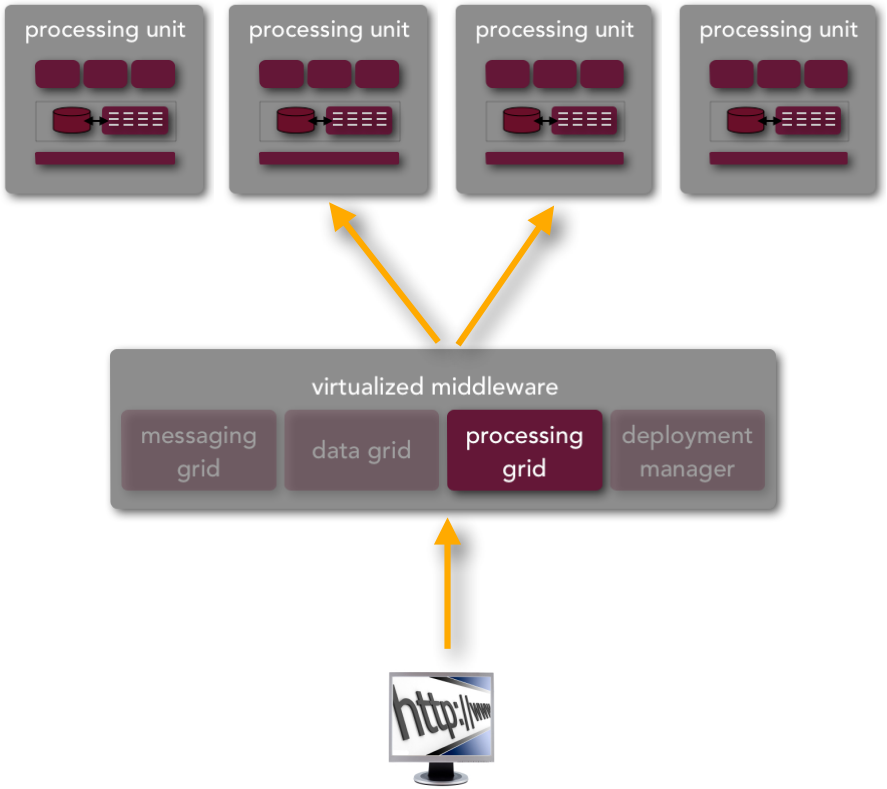
messaging grid

data grid

processing grid

deployment manager

manages distributed request processing



space-based architecture

middleware

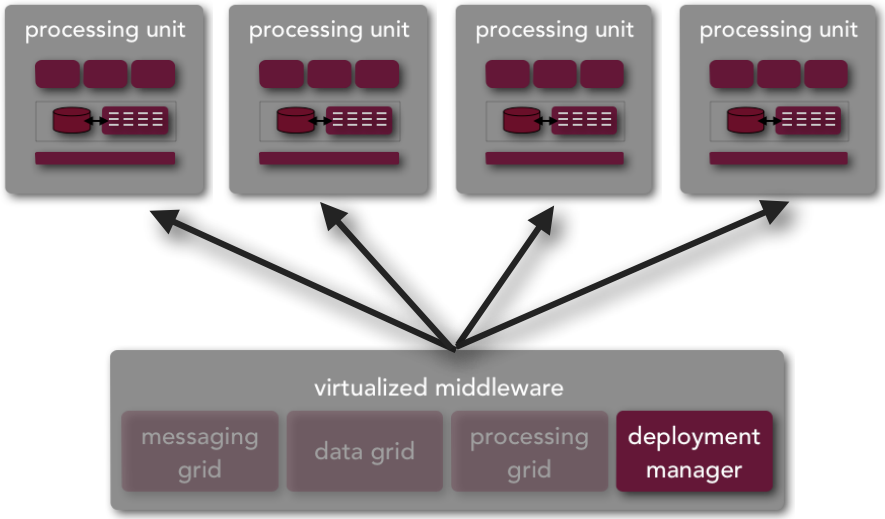
messaging grid

data grid

processing grid

deployment manager

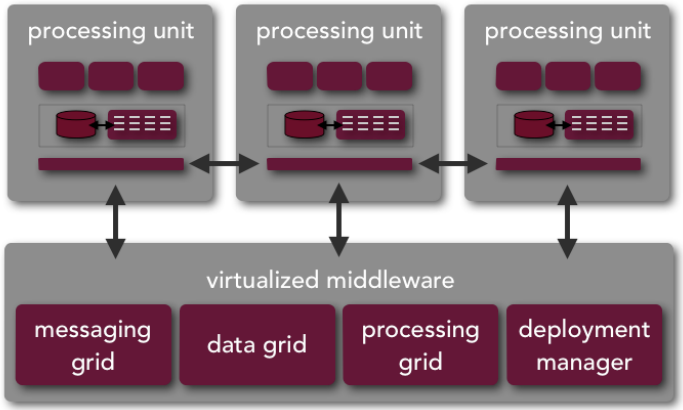
manages dynamic processing unit deployment



space-based architecture

it's all about variable scalability...

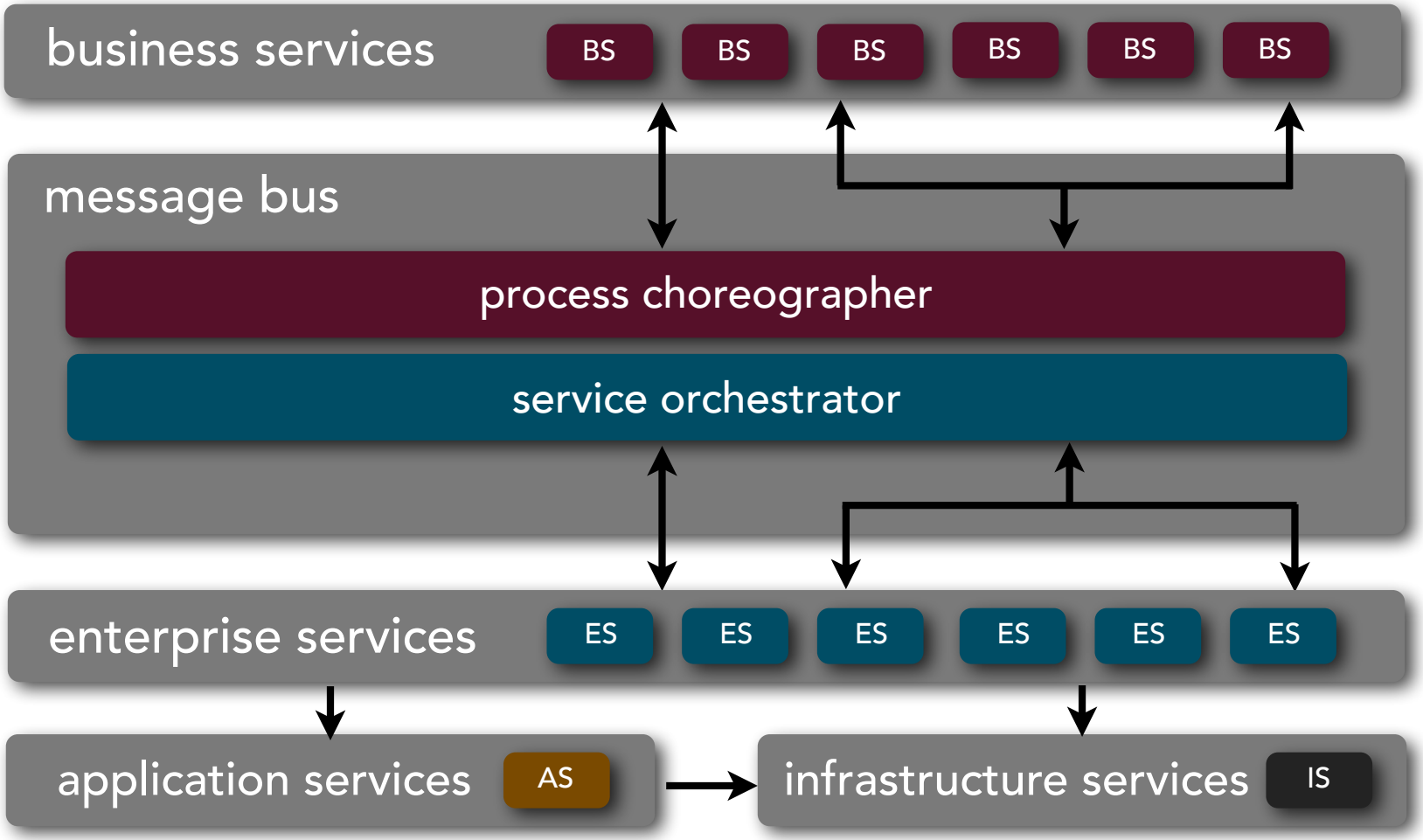
good for applications that have variable load or inconsistent peak times



not a good fit for traditional large-scale relational database systems

relatively complex and expensive pattern to implement

service-oriented architecture



service-oriented architecture

abstract enterprise-level coarse-grained services owned and defined by business users

no implementation - only name, input, and output data represented as wsdl, bpel, xml, etc.

ExecuteTrade

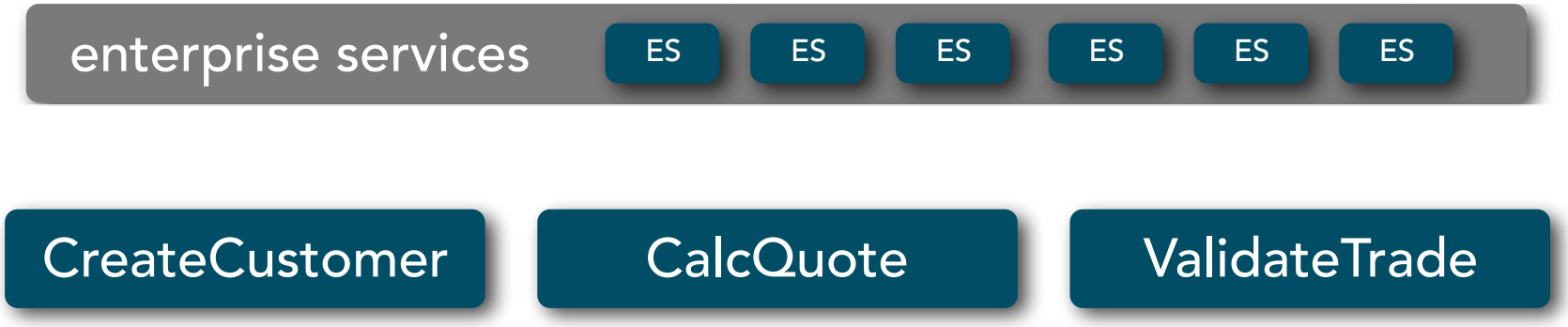
PlaceOrder

ProcessClaim

service-oriented architecture

concrete enterprise-level coarse-grained services
owned by shared services teams

custom or vendor implementations that are one-to-one
or one-to-many relationship with business services



service-oriented architecture

concrete application-level fine-grained services
owned by application teams

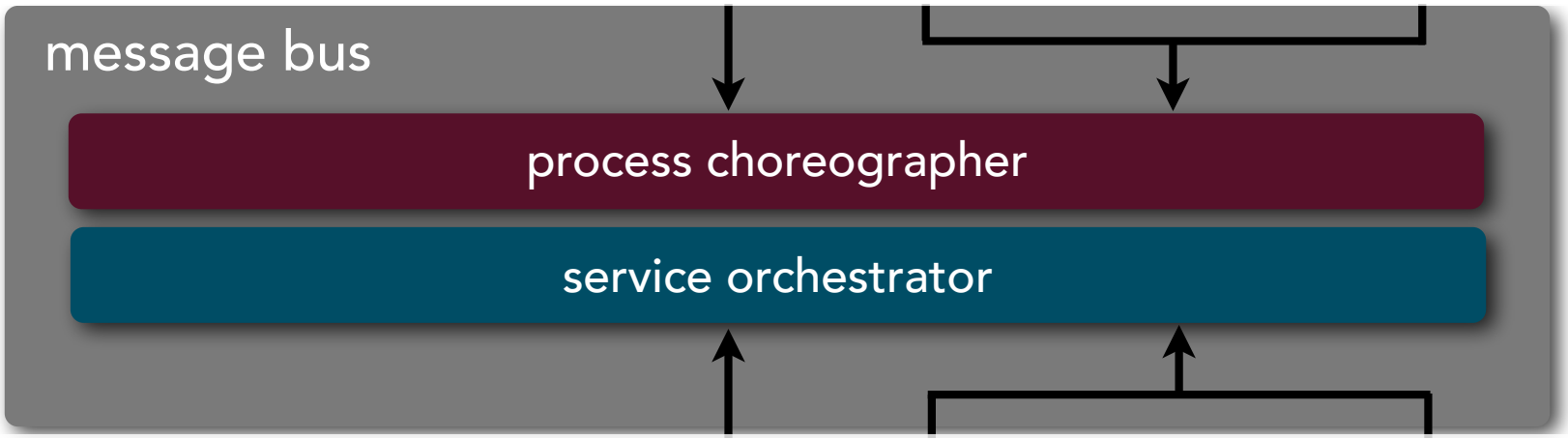
bound to a specific application context

service-oriented architecture

concrete enterprise-level fine-grained services owned by infrastructure or shared services teams

implements non-business functionality to support both enterprise and business services

service-oriented architecture



mediation and routing

process choreography

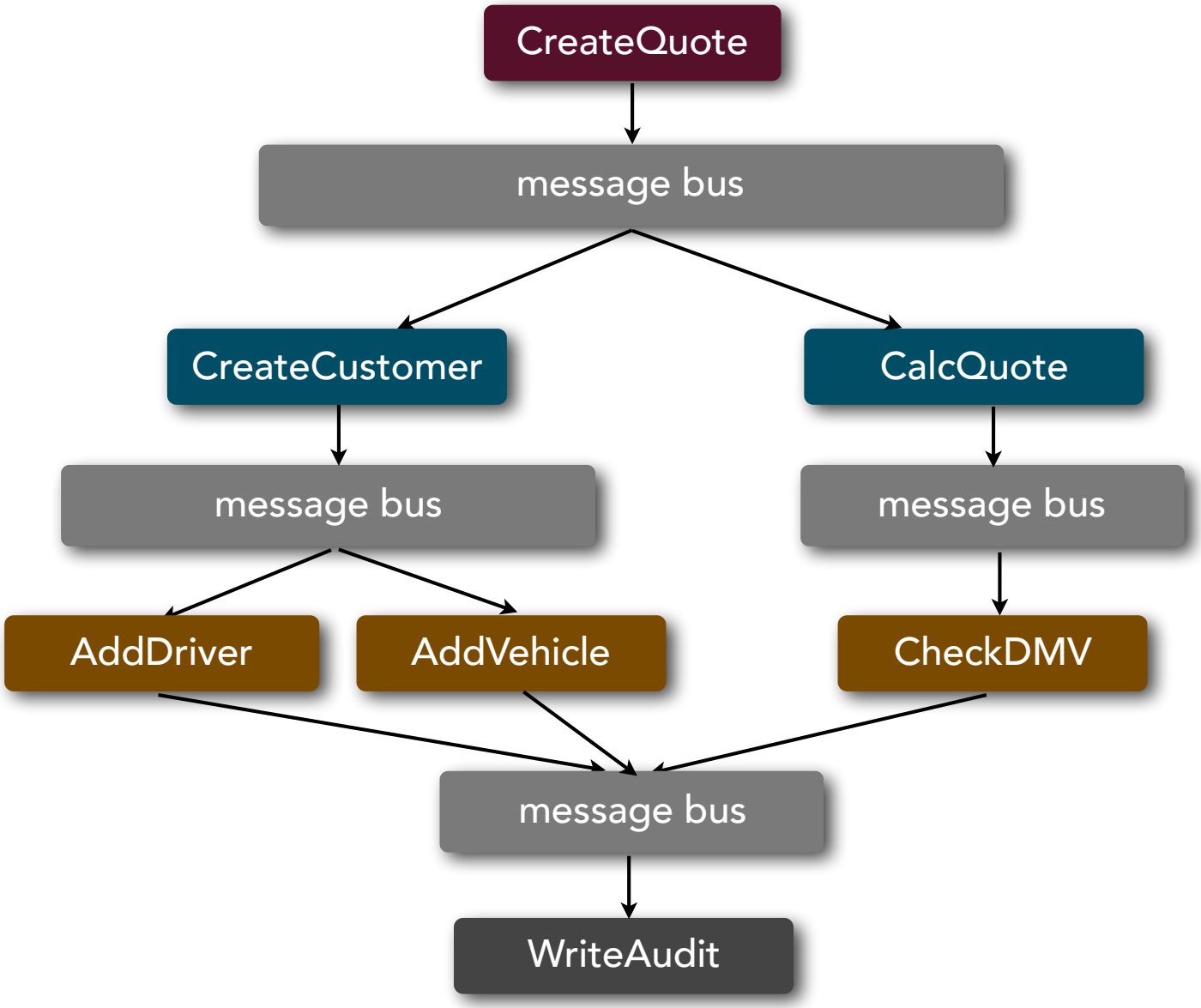
service orchestration

message enhancement

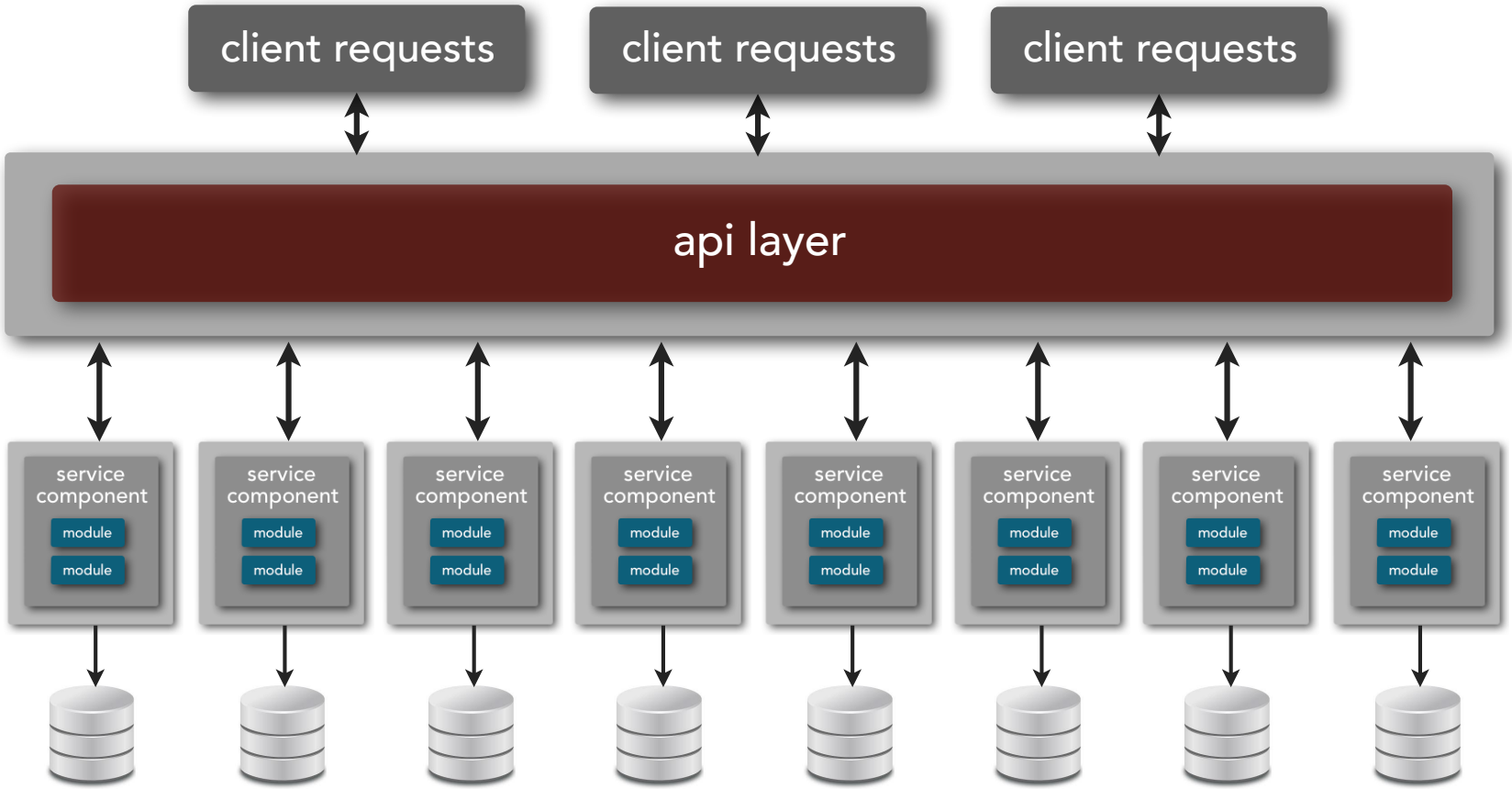
message transformation

protocol transformation

service-oriented architecture

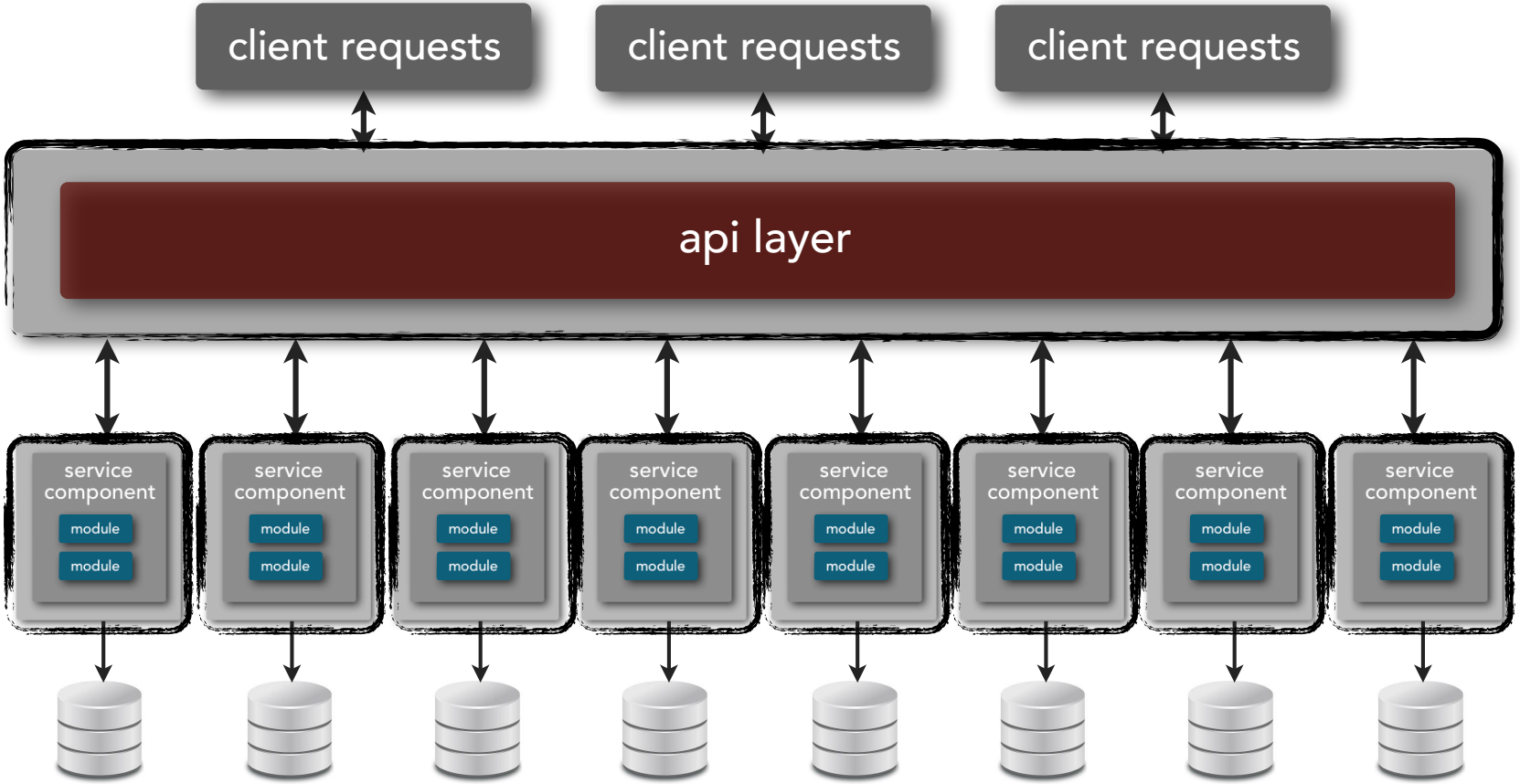


microservices architecture



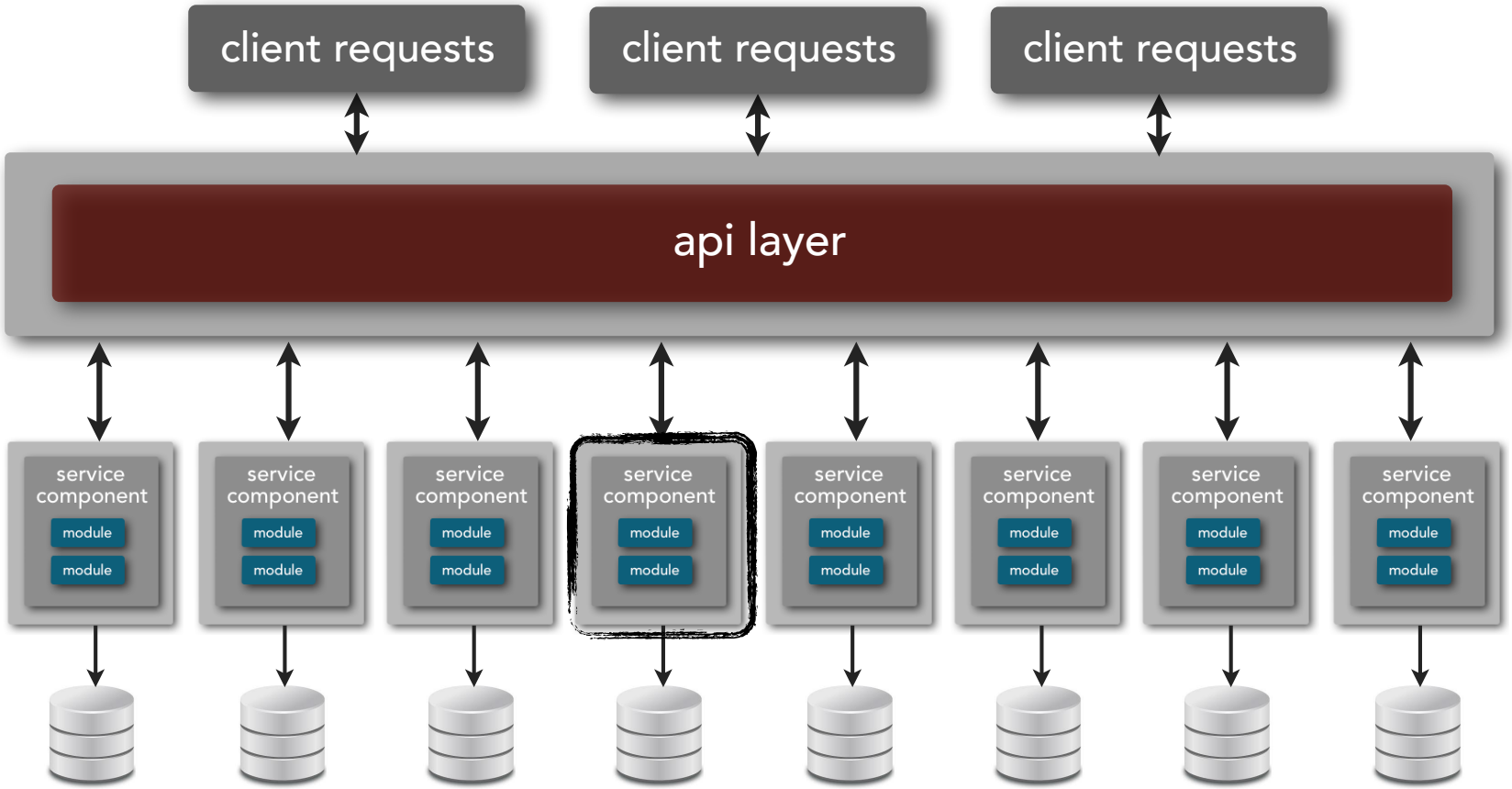
microservices architecture

separately deployed components



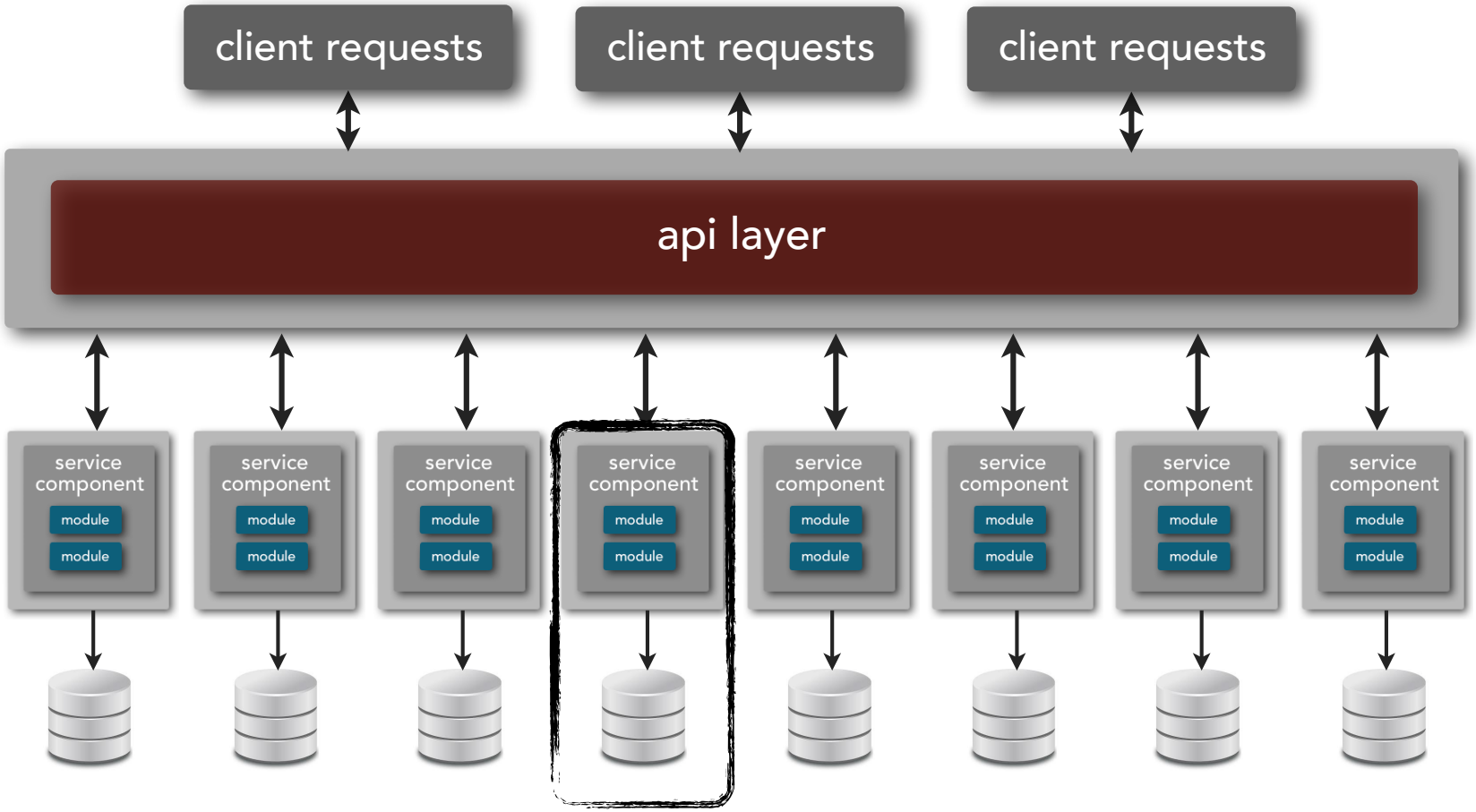
microservices architecture

service component



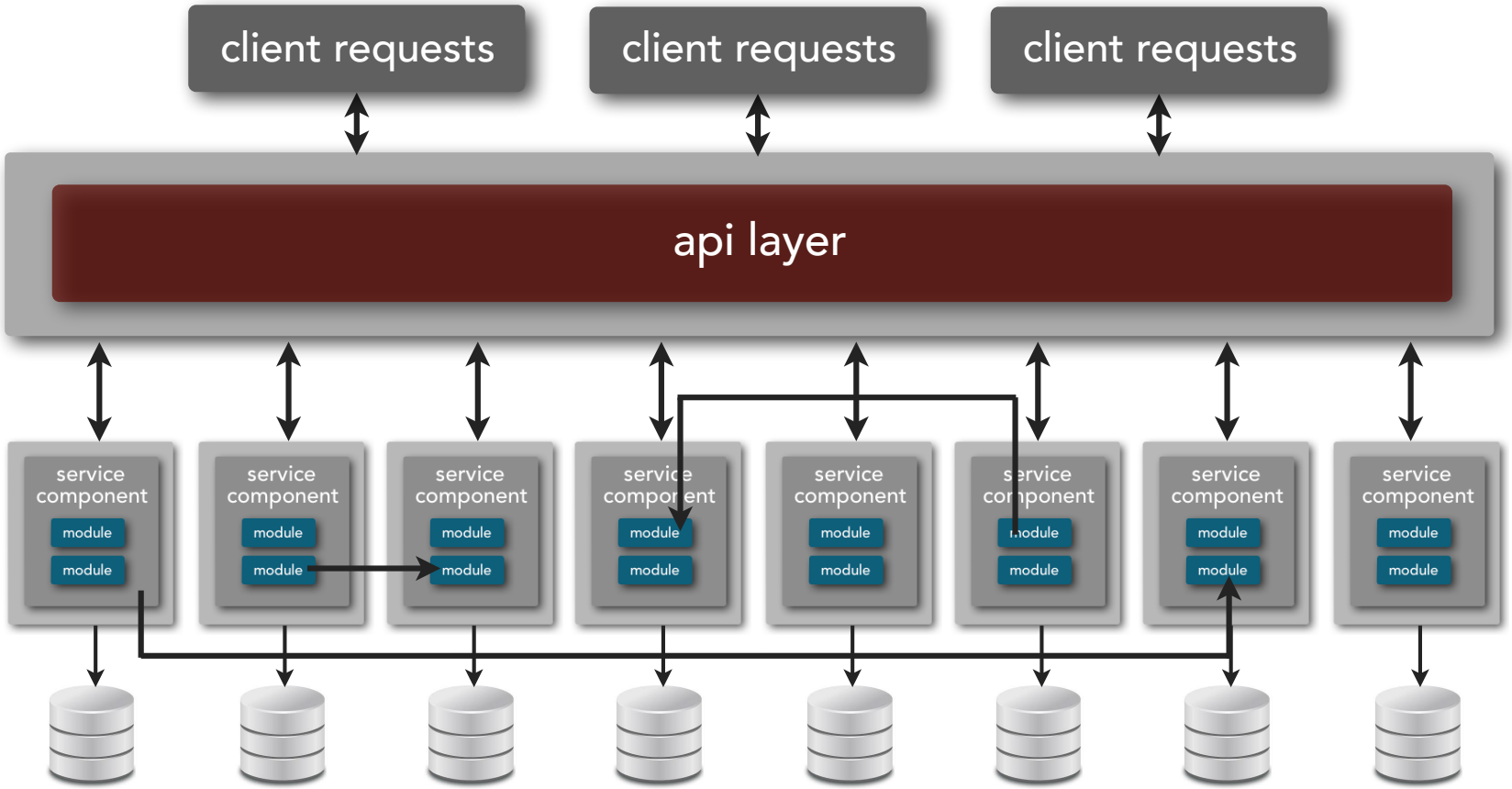
microservices architecture

bounded context



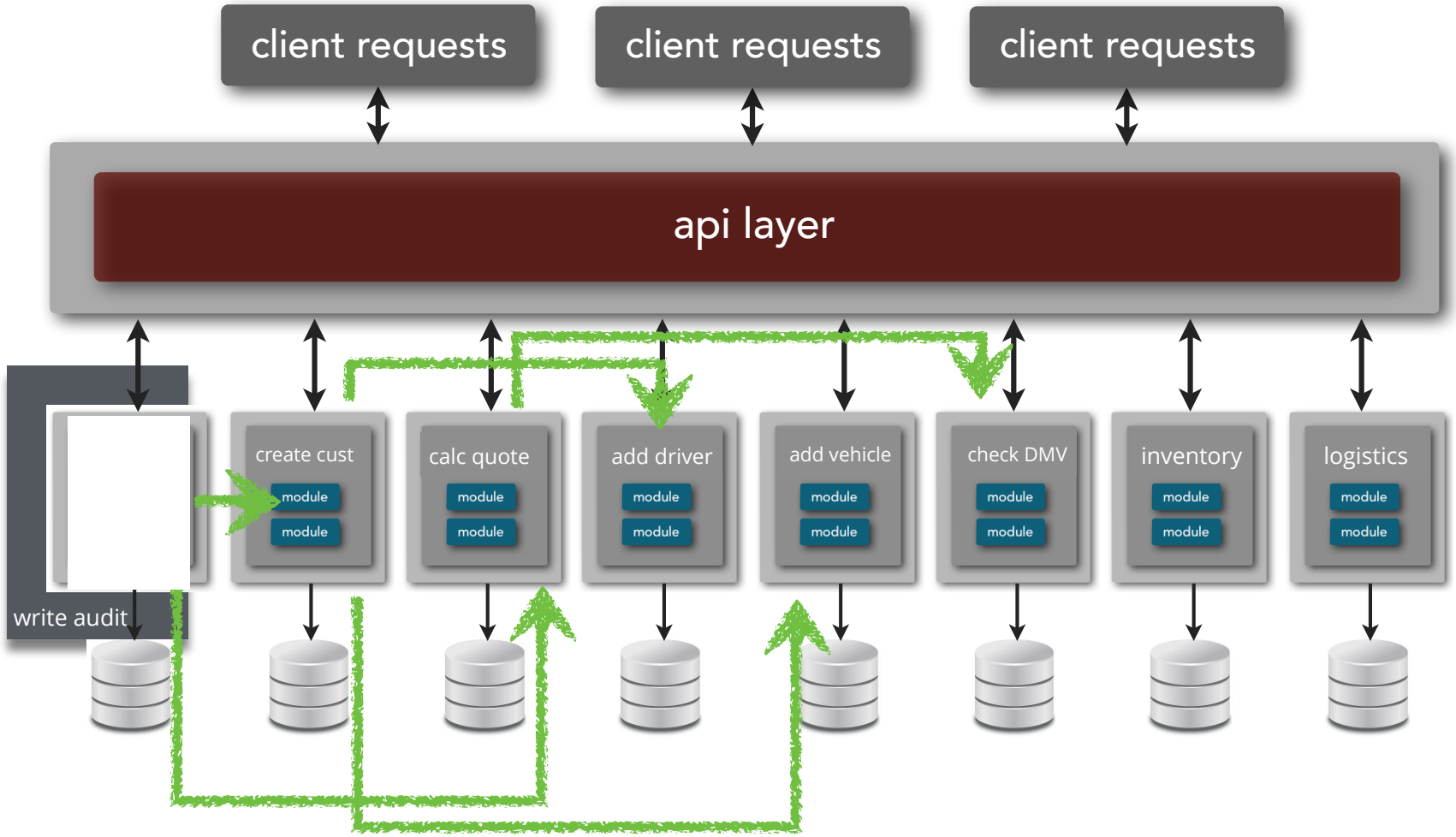
microservices architecture

service orchestration

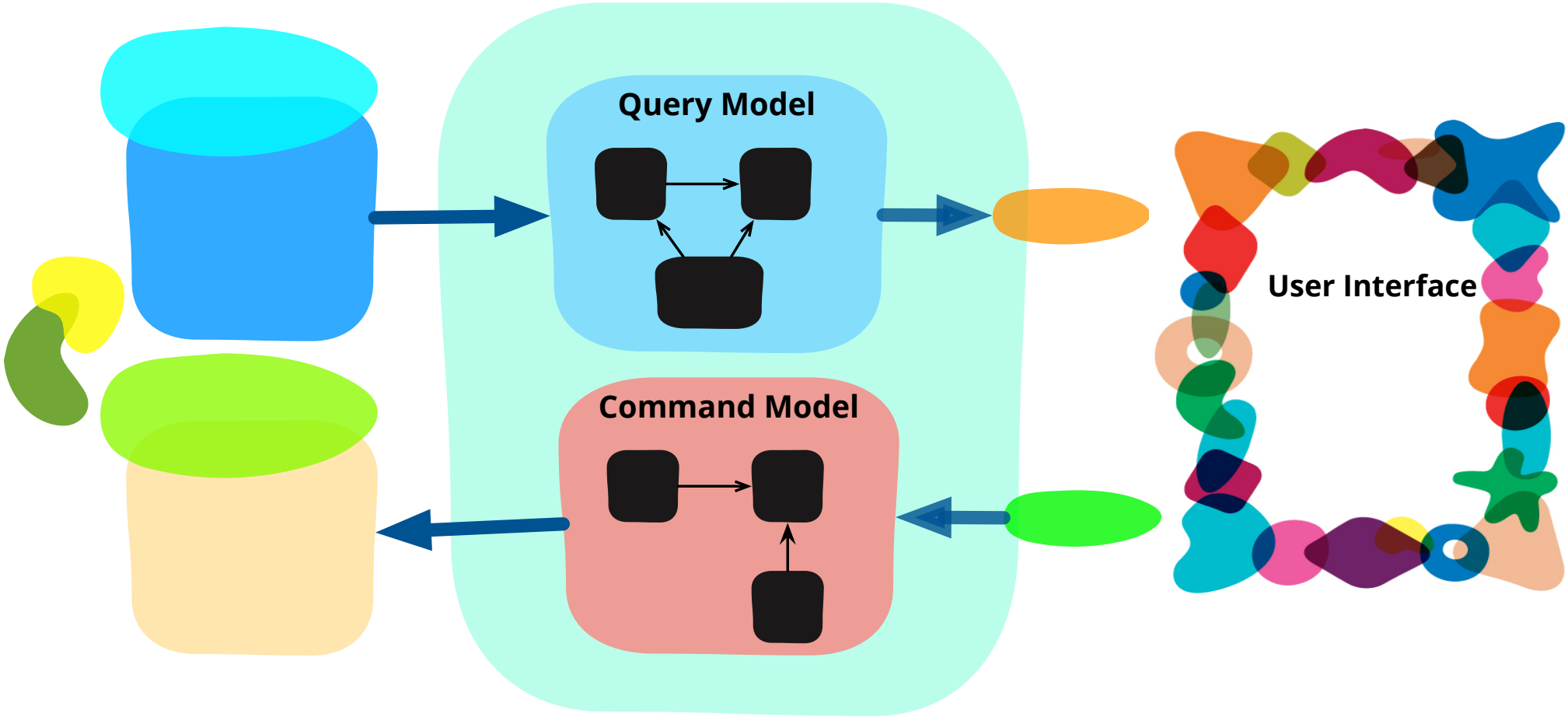


microservices architecture

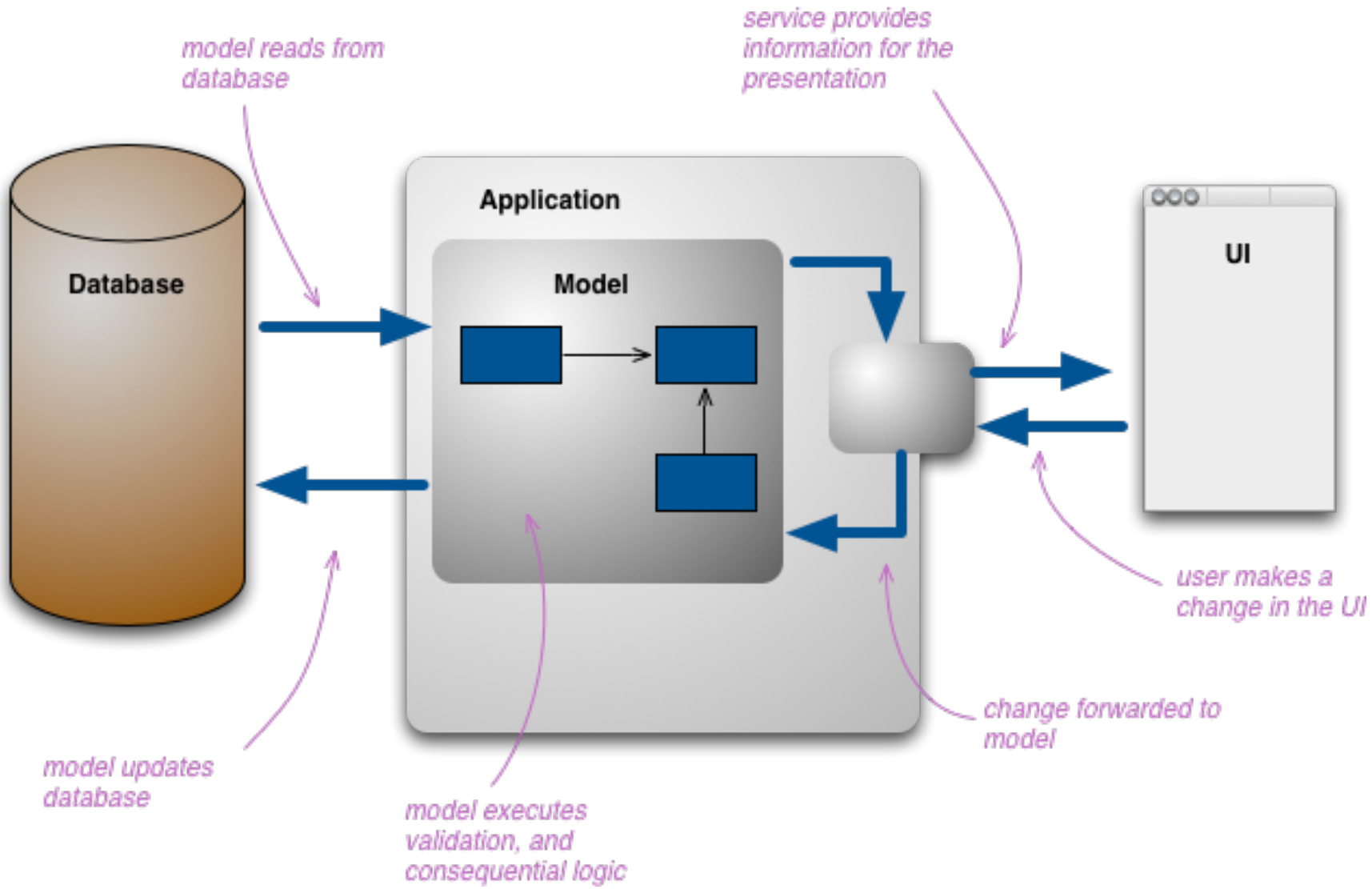
service orchestration



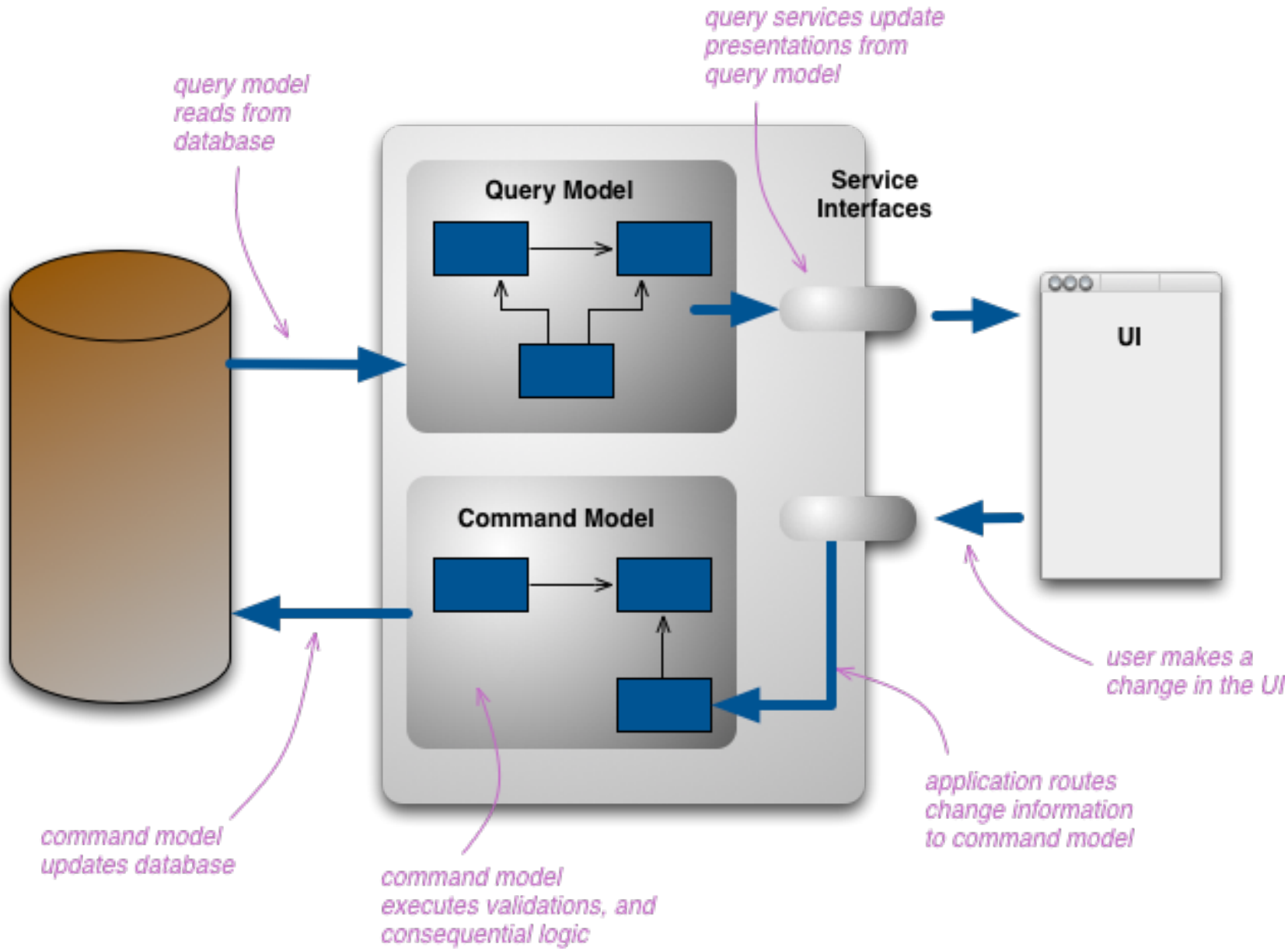
CQRS



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



traditional



CQRS Command Query Responsibility Separation

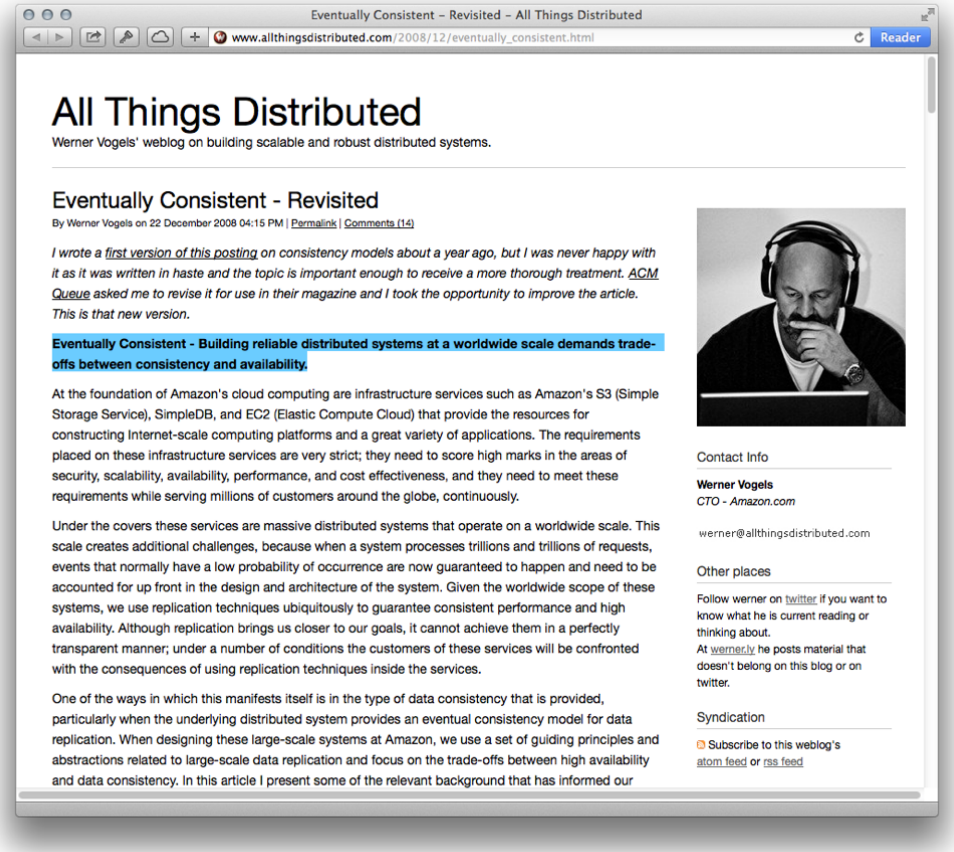
CQRS natural fits

task-based user interface

meshes well with event sourcing

eventual consistency

eventual consistency



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

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

CQRS natural fits

task-based user interface

meshes well with event sourcing

eventual consistency

consistency or availability
(but never both)

complex or granular domains


LMAX

LMAX

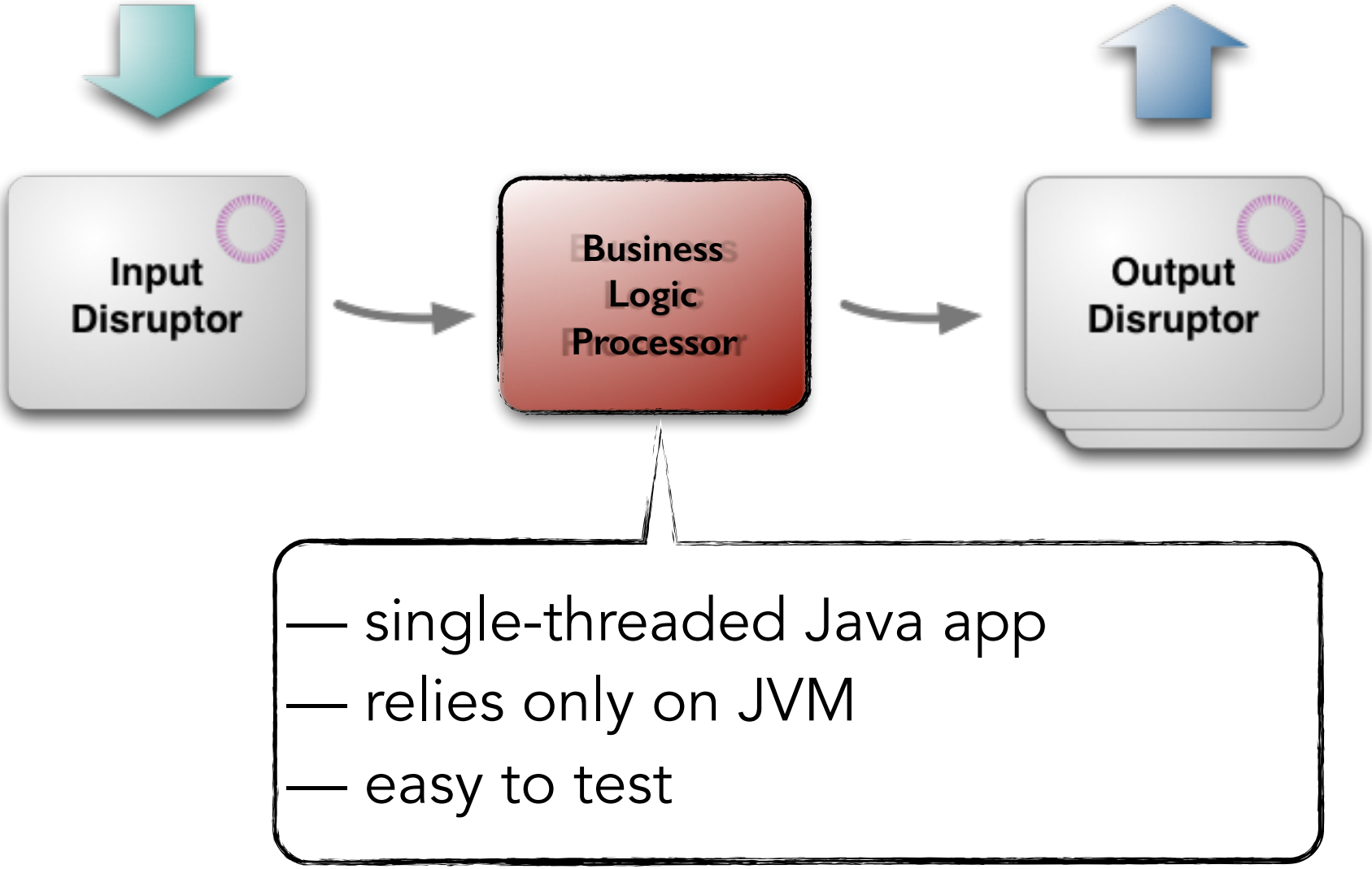
http://martinfowler.com/articles/lmax.html

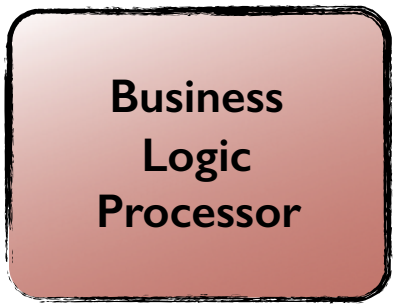
JVM-based retail financial trading platform

centers on Business Logic Processor handling 6,000,000 orders/sec on 1 thread

surrounded by Disruptors, network of lock-less queues

overall structure





business logic processor

in-memory

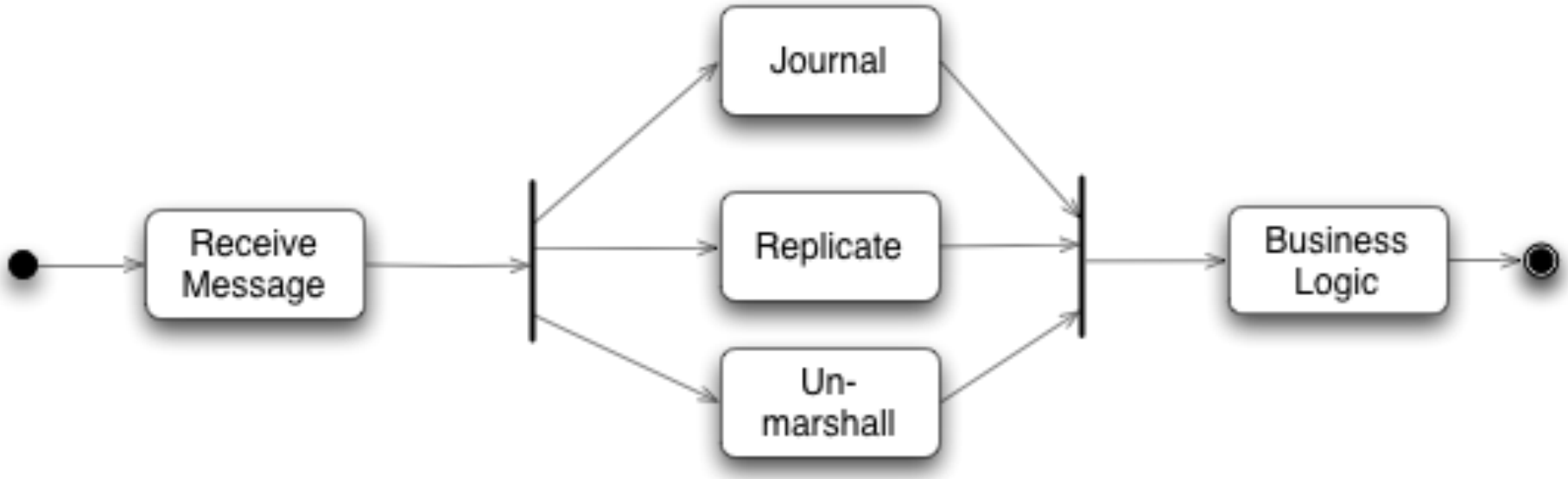
event sourcing via input disruptor

snapshots (full restart—JVM + snapshots — less than 1 min)

multiple instances running

each event processed by multiple processors but only one result used

input/output disruptors



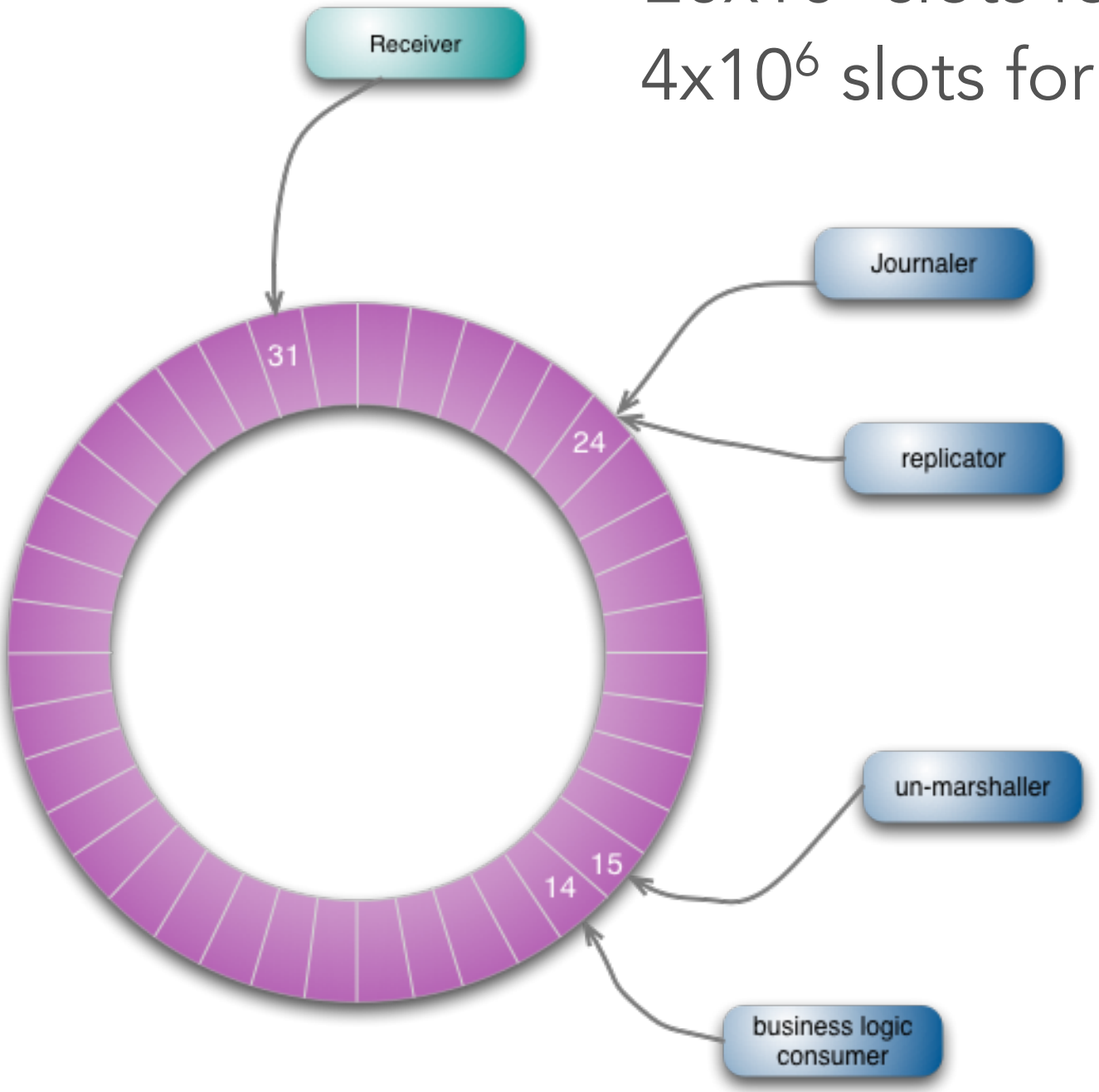
disruptors

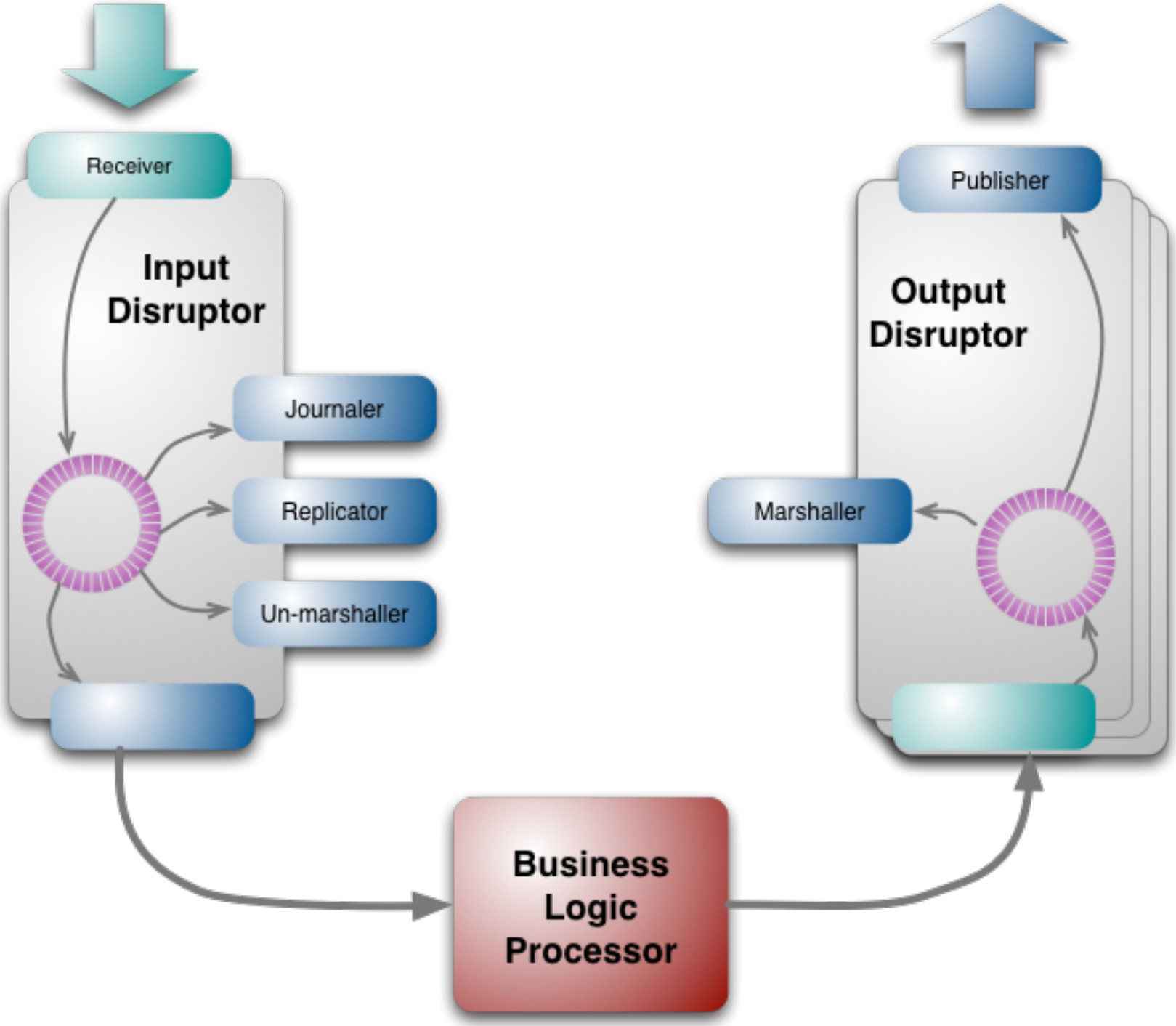
custom concurrency component

multi-cast graph of queues where producers enqueue objects and consumers dequeue in parallel

ring buffer with sequence counters

20x10^6 slots for input buffer
4x10^6 slots for output buffer




Receiver
Input
Disruptor
Journaler
Replicator
Un-marshaller
Publisher
Output
Disruptor
Marshaller
Business
Logic
Processor

“mechanical sympathy”

started with transactions

switched to Actor-based concurrency

hypothesized & measured results

CPU caching is key ➠ single writer principle

“One ring to rule them all…”

no architecture fits every circumstance

evaluate good, bad, and ugly

watch for primrose paths that turn ugly

embrace pragmatism

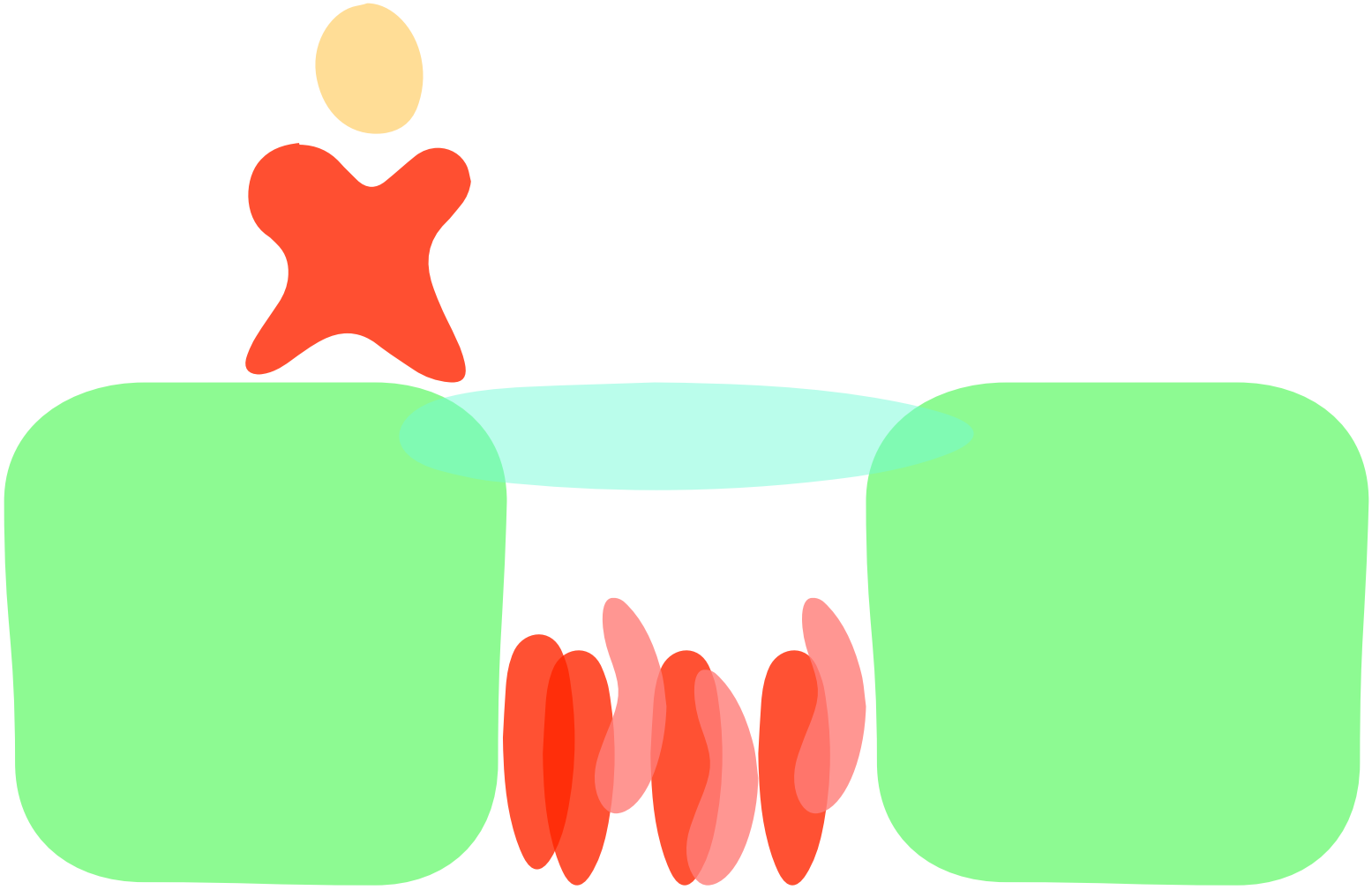
what's your day job?

building wicked cool architectures is an stimulating, rewarding intellectual challenge...

...building CRUD applications isn't

coolness sometimes equals accidental complexity

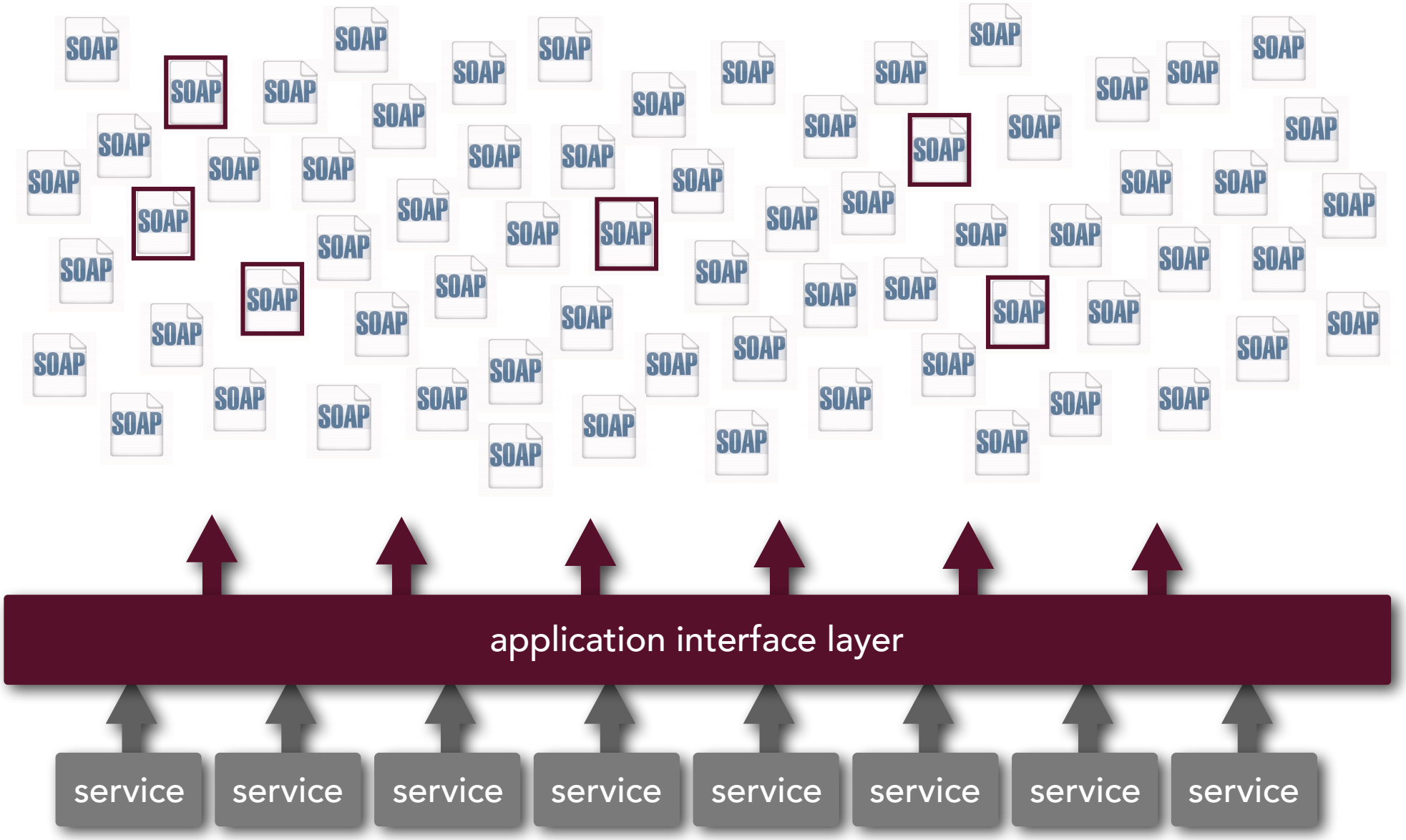
architecture pitfall



spider web architecture

creating large numbers of web services that are never used just because you can

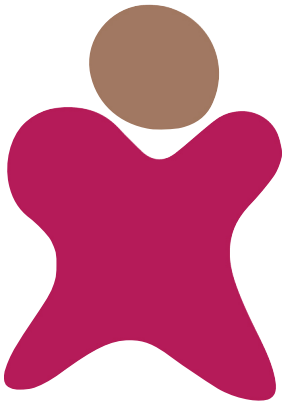
spider web architecture



spider web architecture

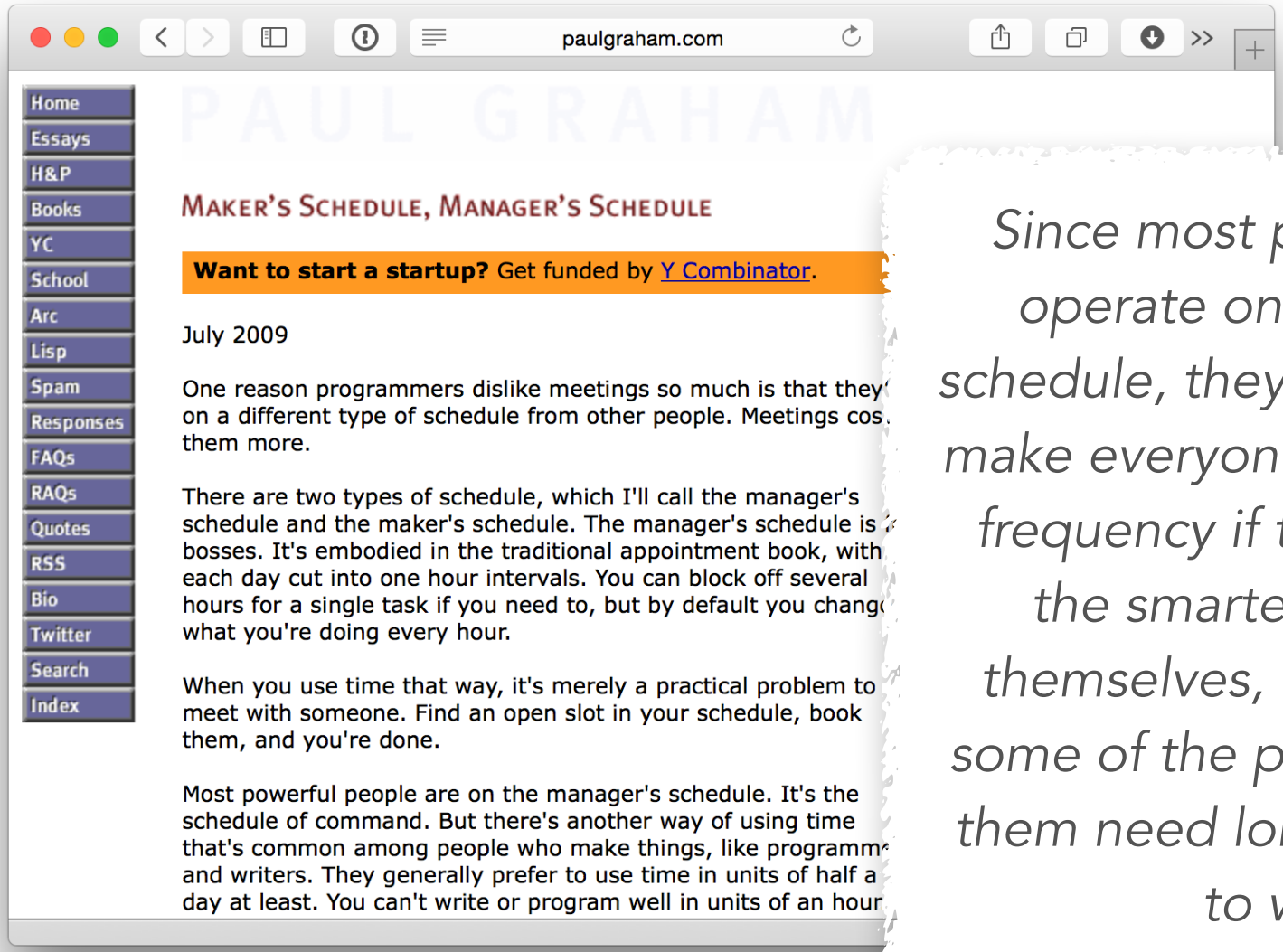
just because you can create a web service at the click of a button doesn't mean you should!

let the requirements and business needs drive what services should be exposed



Meeting Hacks

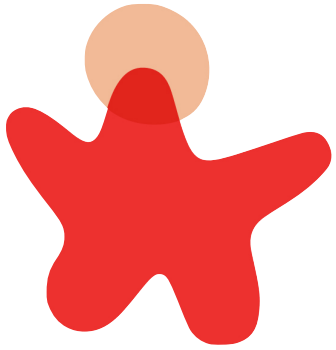
makers vs. managers



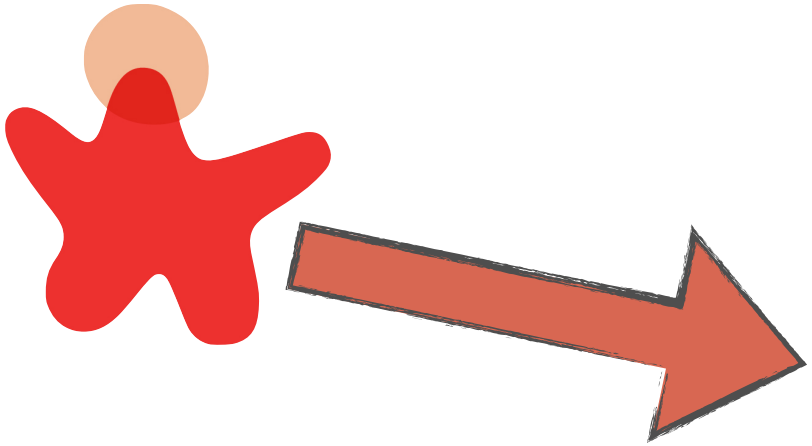
Since most powerful people operate on the manager's schedule, they're in a position to make everyone resonate at their frequency if they want to. But the smarter ones restrain themselves, if they know that some of the people working for them need long chunks of time to work in.

www.paulgraham.com/makersschedule.html

imposed upon

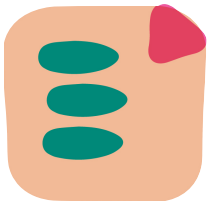


meetings...



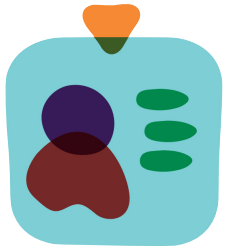
imposed by

imposed meetings



stick to the agenda

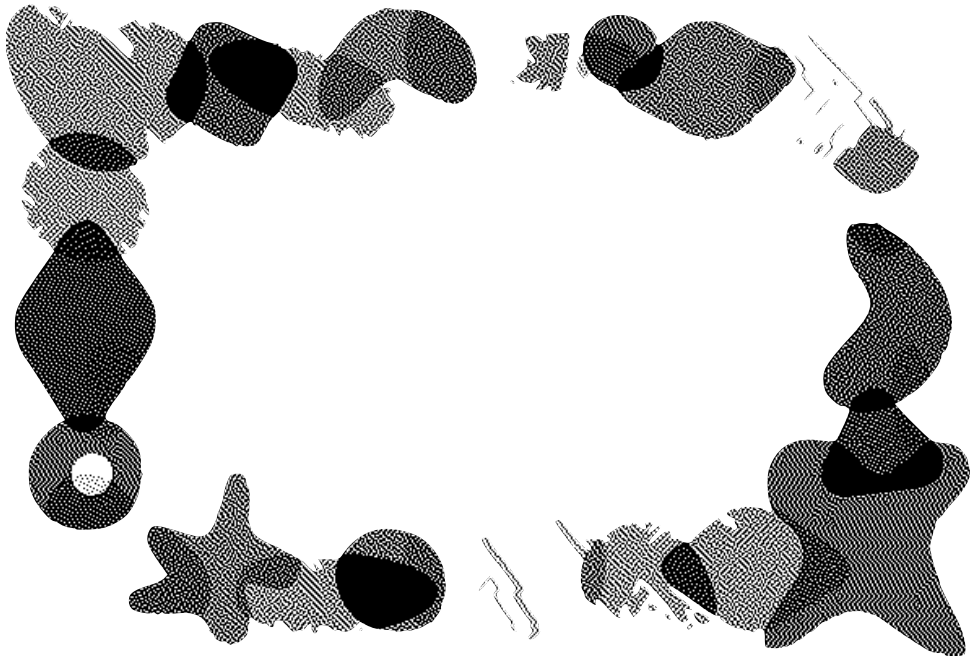
take one (or many) for the team

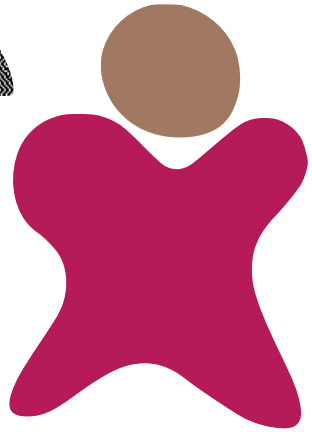


know the dramatis personæ

don’t ask important questions
you don’t already know the answer to

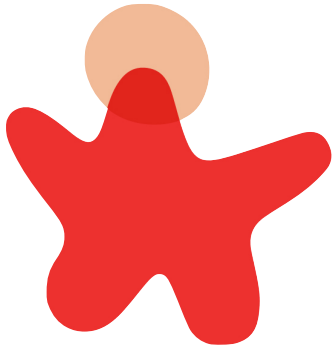
meeting imagery



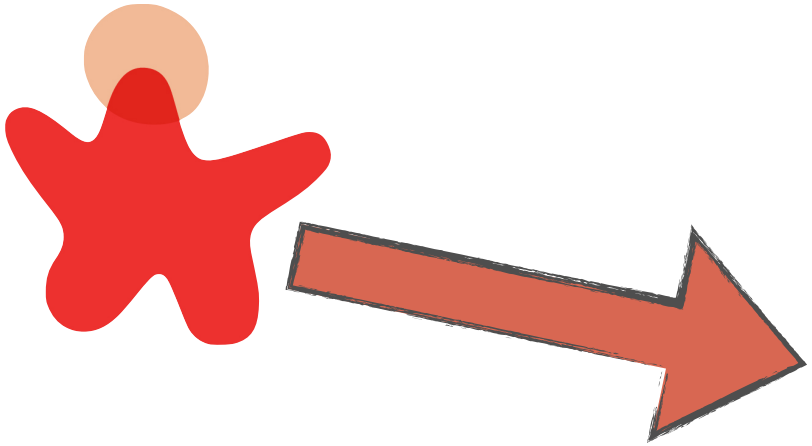


don’t improvise white board drawings in important meetings

imposed upon

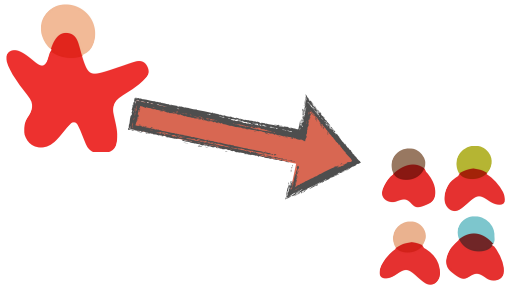


meetings...



imposed by

imposed by



dont !

is this more important than the work you are pulling people away from?

set an agenda

have someone take lightweight notes

specialized meetings

stand-up

<role>-huddle

BA-huddle

dev-huddle

—keep it short

—keep it relevant

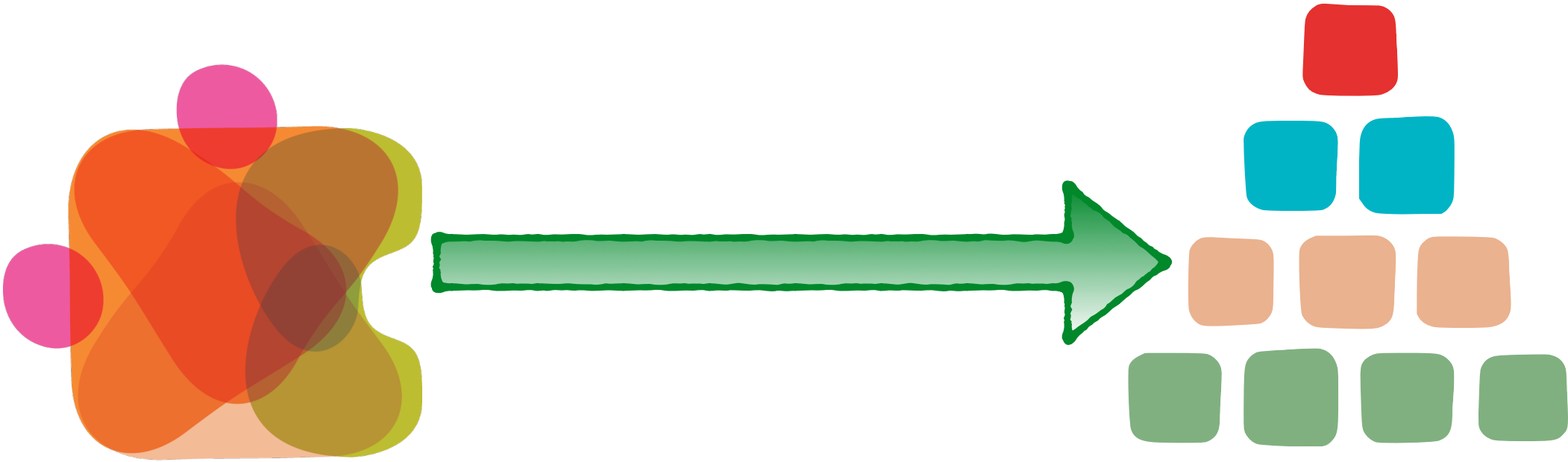
—everyone

  —too many?

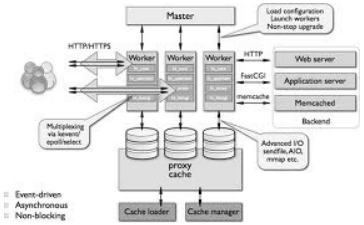
  —geographically dispersed?

—police inappropriate levels of interaction

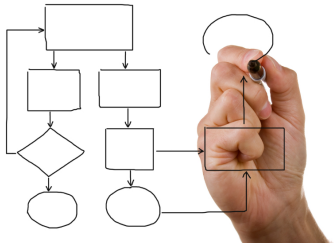
Architecture Refactoring Techniques



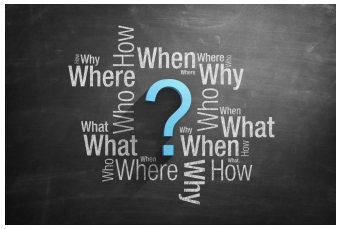
architecture refactoring



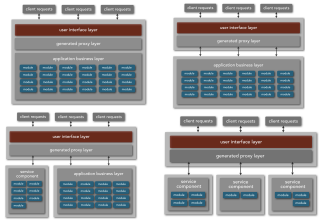
determine the architectural root cause of each issue



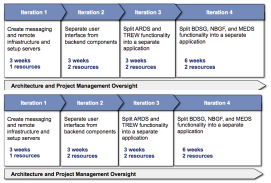
determine what architecture changes are needed



create a business case justifying the changes



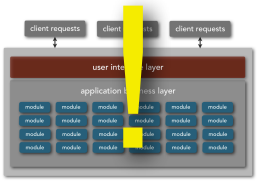
develop a high-level architecture refactoring plan



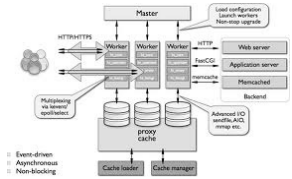
create a timeline containing estimates and resources

present your case and plan to the business for approval and funding

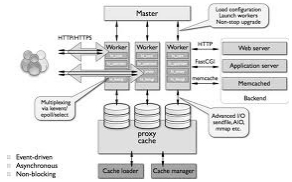
determine root cause



application builds fail almost every time something is committed to the repository

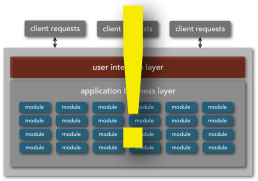


application components are too tightly coupled and dependent on one another

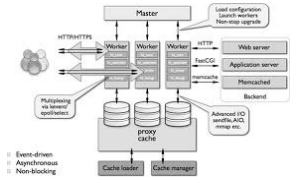


application components do not have the right level of granularity and isolation from a roles and responsibility standpoint

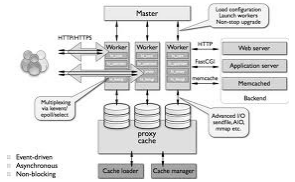
determine root cause



every time the application is deployed with new functionality, something else usually breaks

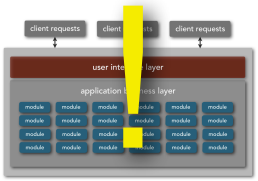


application components are too tightly coupled and dependent on one another

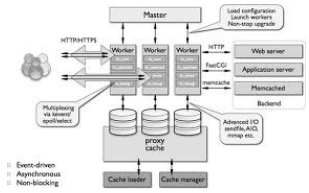


application components do not have the right level of granularity and isolation from a roles and responsibility standpoint

determine root cause

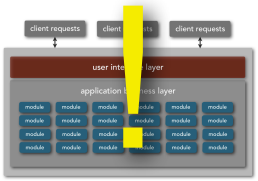


application deployments take a long time, sometimes lasting up to 25 minutes

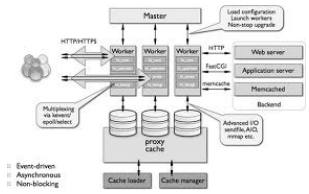


the application is a monolithic and growing quickly based on new features and functionality; it is getting too large for a single application.

determine root cause



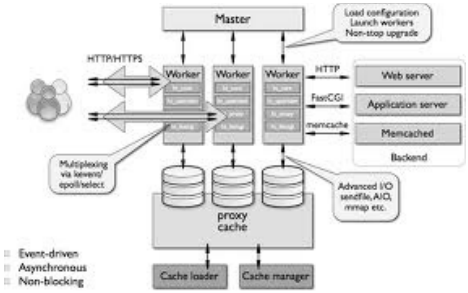
users are increasingly reporting performance issues with the application



the application is becoming too large; it is consuming about 95% of the available jvm resources during normal load (memory, cpu, threads)

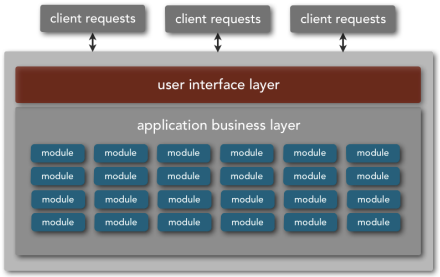
determine root cause

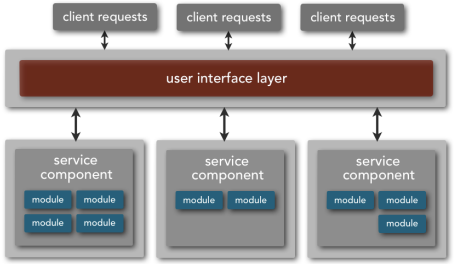
root cause summary



the application is too tightly coupled and is growing beyond what the current architecture can support

determine architecture changes



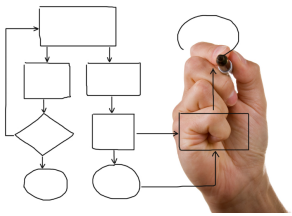


the monolithic application is too tightly coupled and is growing beyond what the current architecture can support

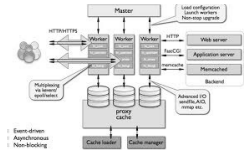
split the application into multiple deployable units, thereby decoupling components and allowing for more growth potential

justifying your case

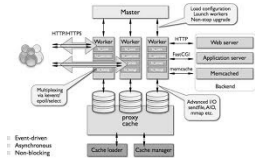
technical justification



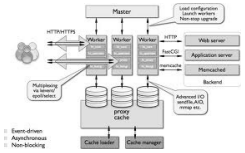
split the monolithic application into multiple deployable units, thereby decoupling components and allowing for more growth potential



components will be more decoupled, thereby eliminating frequent build issues



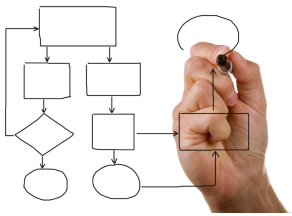
each part will use fewer jvm resources, thereby increasing performance and allow for more growth



deployment is limited to a separate application unit, thereby reducing deployment time and increasing robustness

justifying your case

business justification



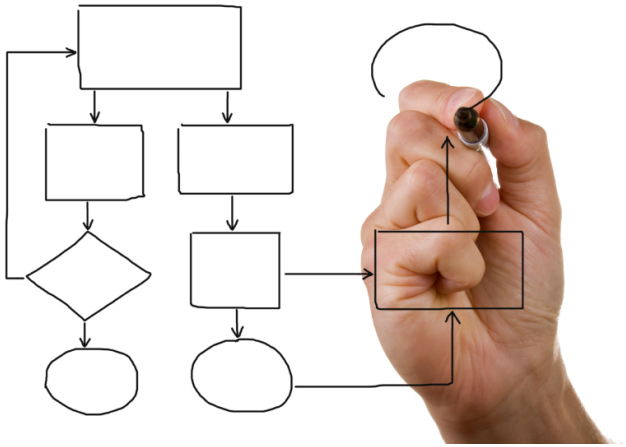
split the monolithic application into multiple deployable units, thereby decoupling components and allowing for more growth potential

new functionality can be delivered faster, thereby improving overall time to market

overall application quality will be improved, thereby reducing bugs and the associated costs of fixing them

development and deployment costs associated with developing new functionality will be significantly reduced

refactoring techniques



work in small iterations

identify the technical and business value expected at each iteration

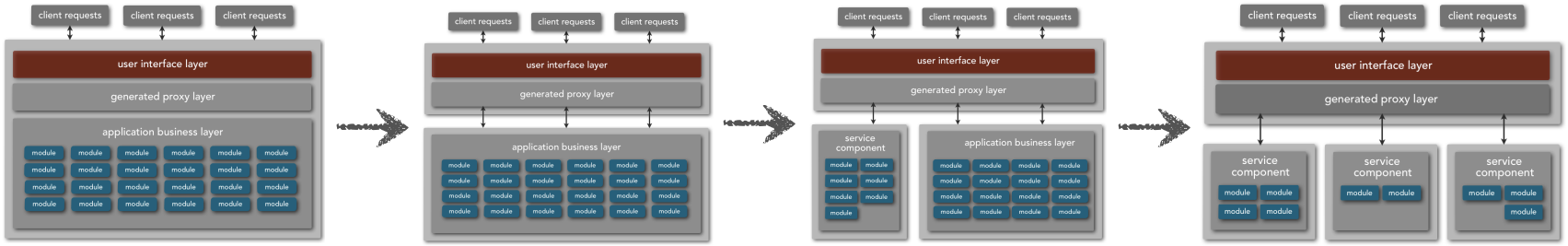
use a playbook approach to outline the architecture transformations

decide on migration vs. adaptation (or a combination of both)

refactoring techniques

playbook approach

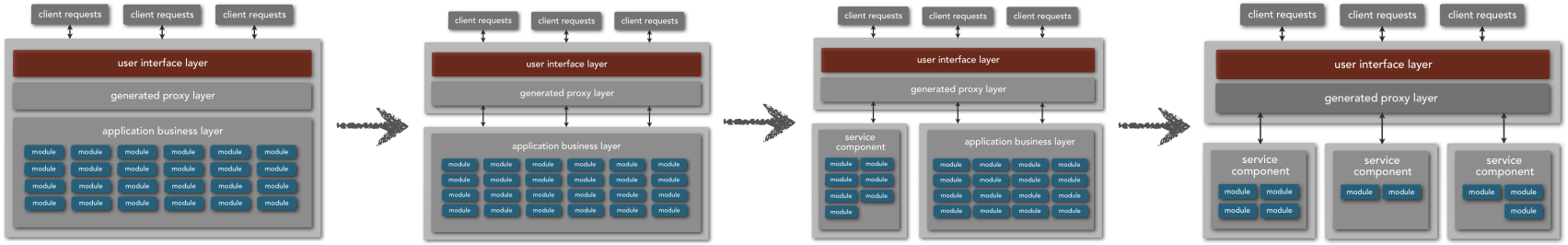
each iteration should clearly illustrate the changes to the architecture each step along the way



current state iteration 1 iteration 2 iteration 3

refactoring techniques

playbook approach



current state iteration 1 iteration 2 iteration 3

identify the purpose behind each iteration

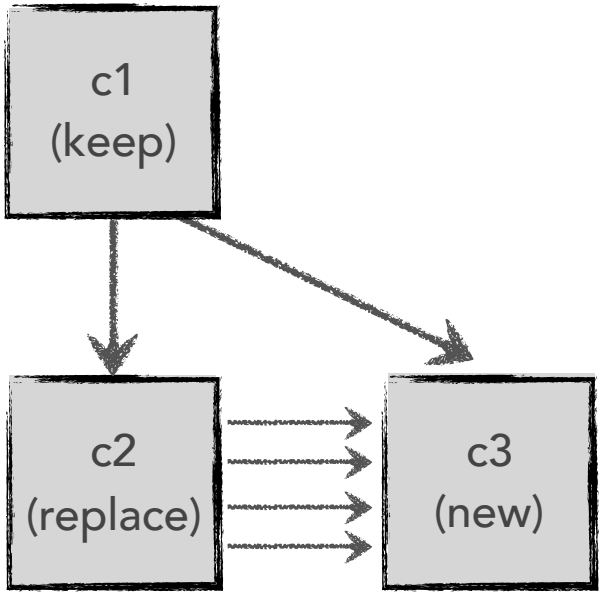
identify the technical and business value for each iteration

try to minimize "staging iterations"

keep iterations as small as possible while still providing enough technical and business value

refactoring techniques

migration vs. adaptation

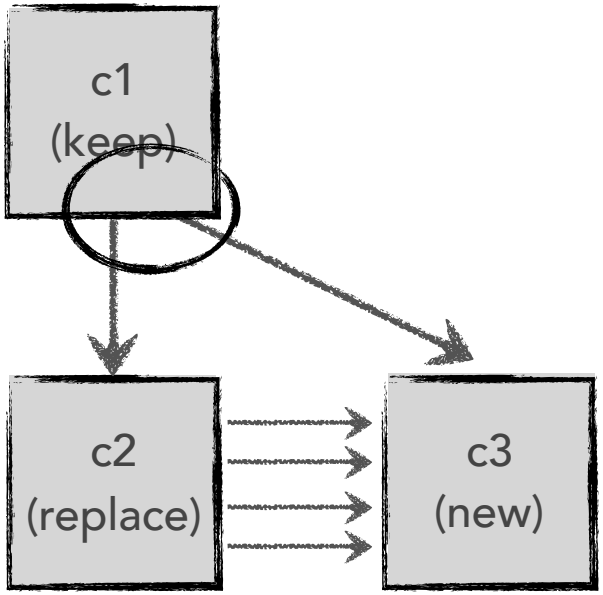


migration

the replacement of old components with new ones through migration over time

refactoring techniques

migration vs. adaptation



migration

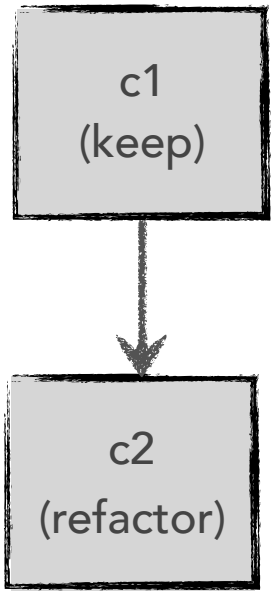
easier to rollback changes

less overall risk

requires switching logic in calling components

refactoring techniques

migration vs. adaptation

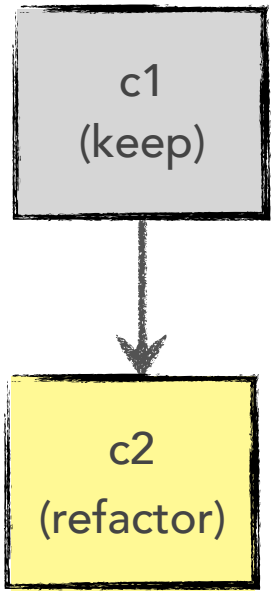


adaptation

refactoring of existing components into new functionality

refactoring techniques

migration vs. adaptation



adaptation

refactor vs. replacement

harder to rollback changes

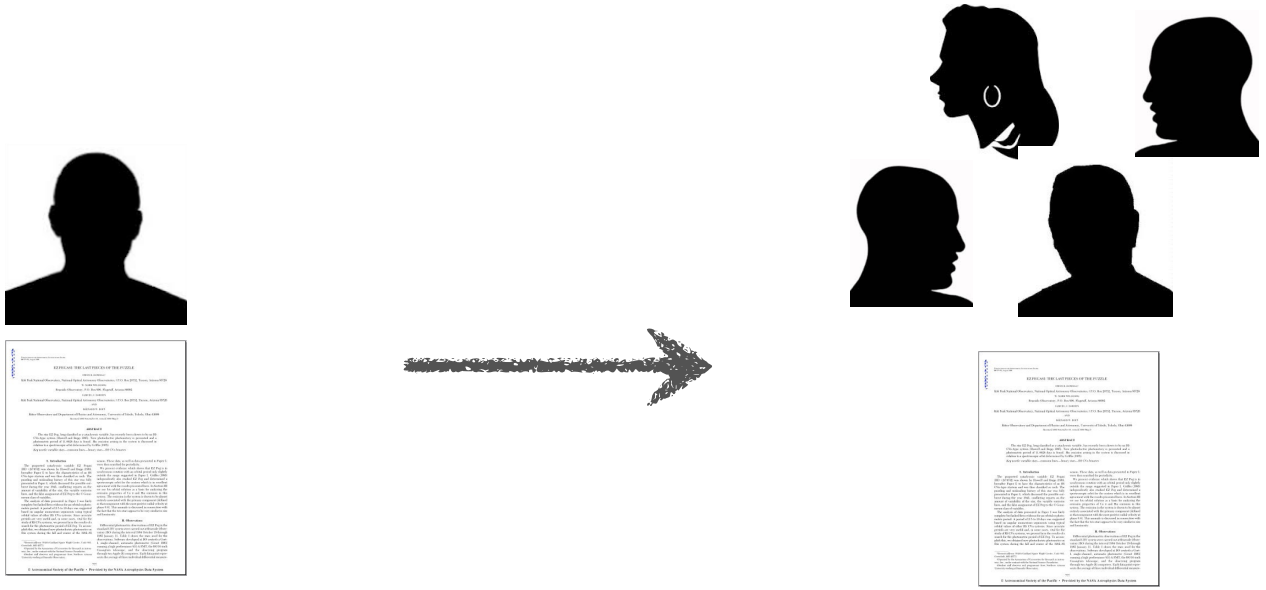
no changes to calling components

presenting your case

presenting your case

"how did things get this bad in the first place?"

presenting your case



always present your plan to your immediate manager before going to the business

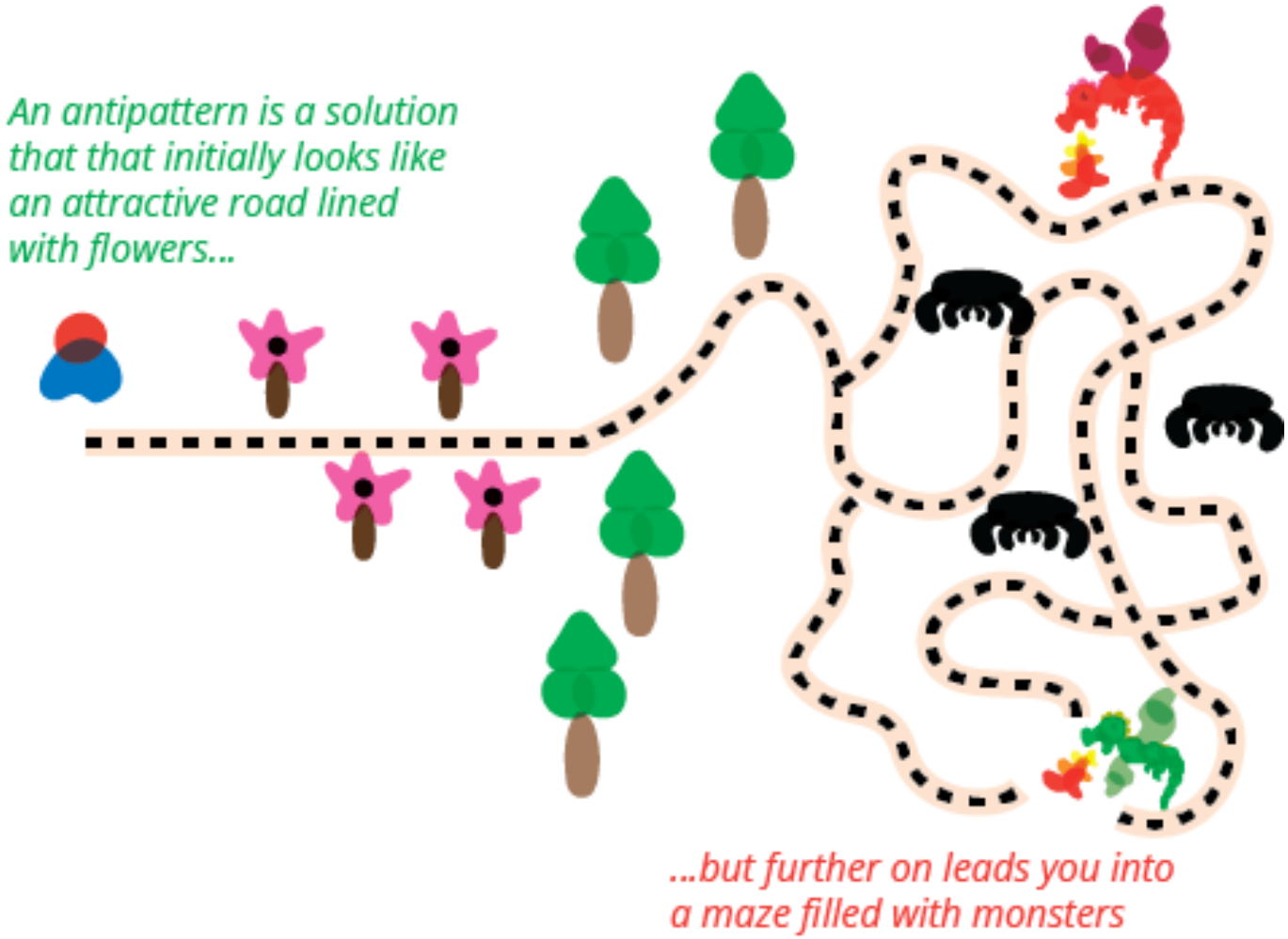
presenting your case

don't scare people -
"present with urgency, not with panic"

presenting your case

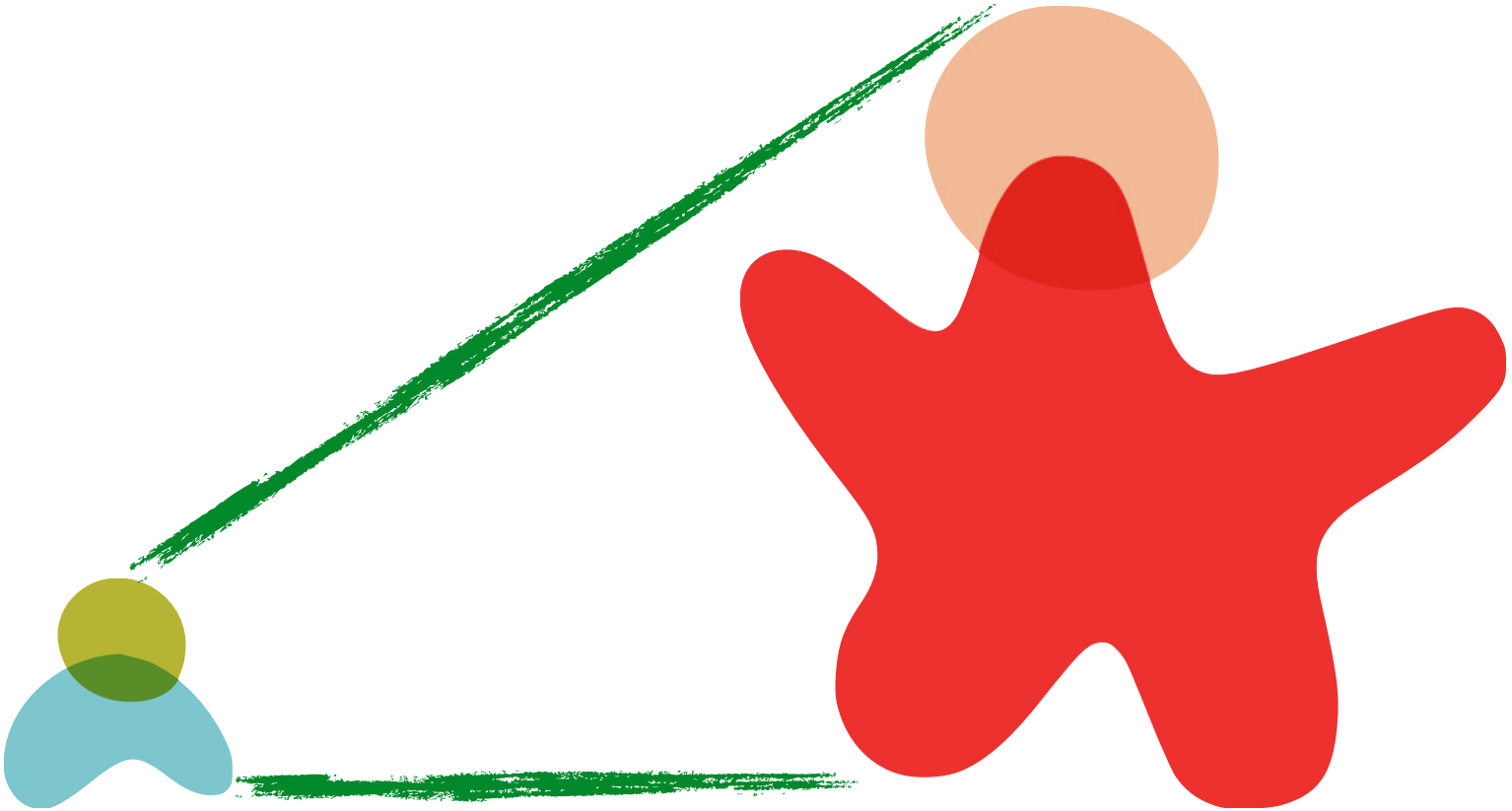
although you may know all the answers, don't be argumentative - take this as an opportunity to educate the business instead

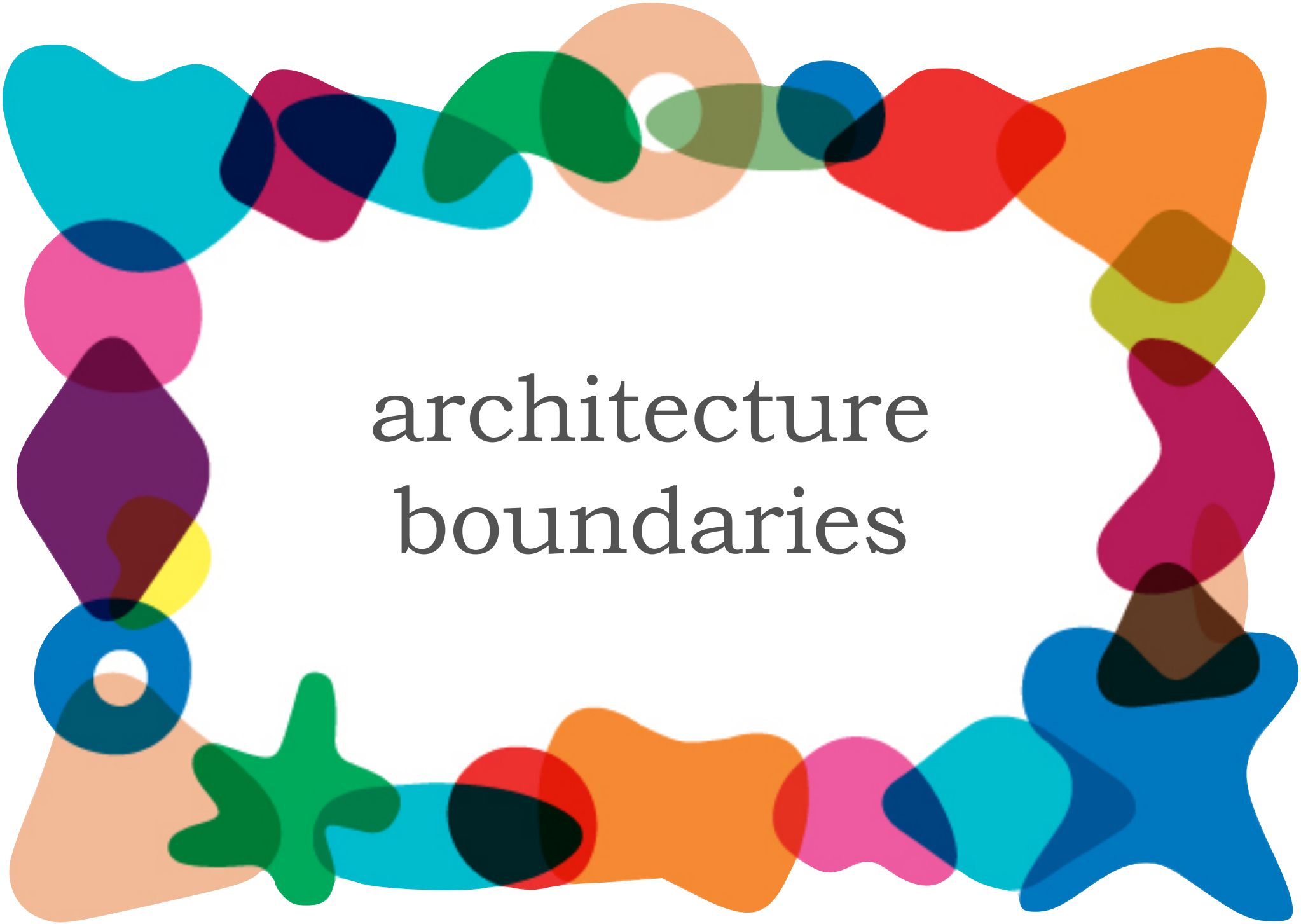
Architecture Anti-pattern



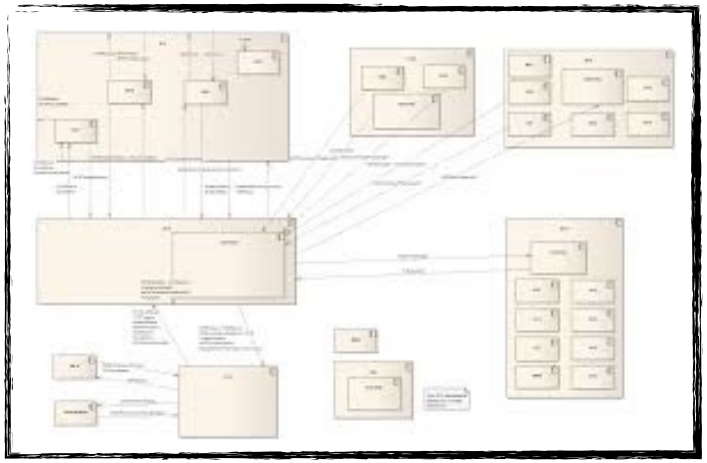
Imposter Syndrome

a term coined in the 1970s by psychologists and researchers to informally describe people who are unable to internalize their accomplishments




architecture
boundaries

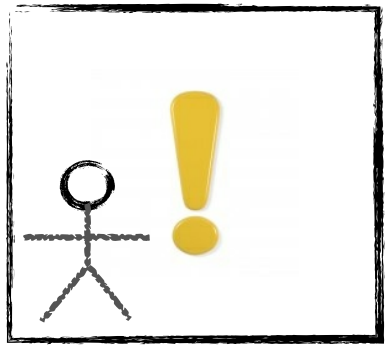
architectural boundaries



there is an art to defining the box that development teams can work in to implement the architecture

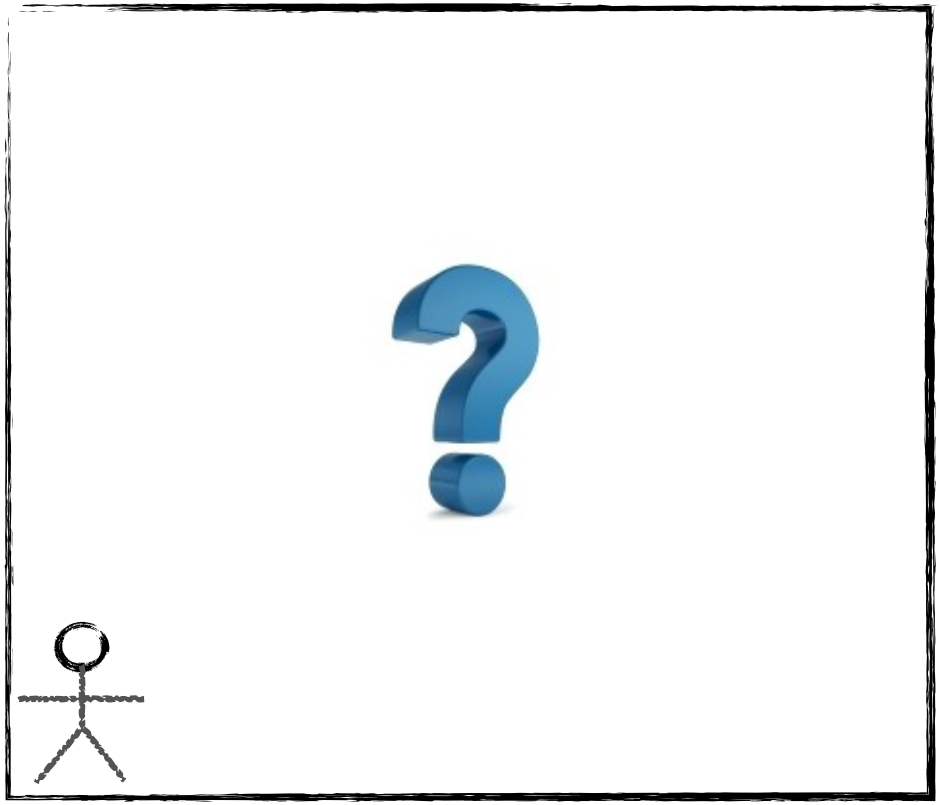
architectural boundaries

tight boundaries



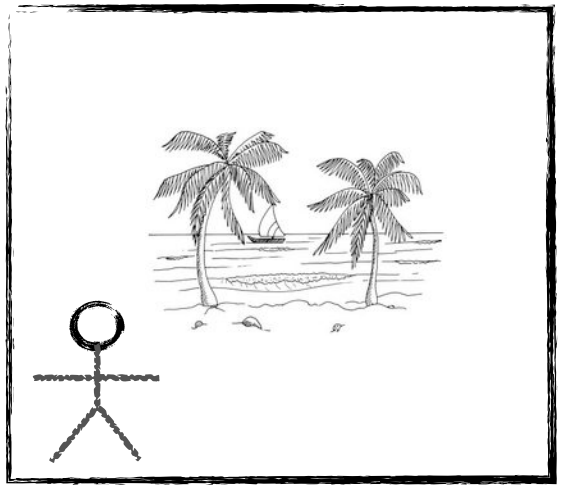
architectural boundaries

loose boundaries



architectural boundaries

appropriate boundaries



architect personalities

control freak architect

architect personalities

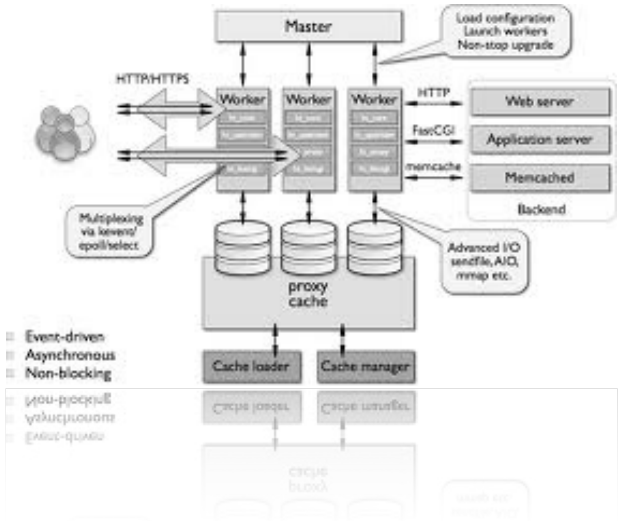
armchair architect

architect personalities

effective architect

controlling the boundaries

the architect defines the architecture and design principles used to guide technology decisions



controlling the boundaries

controlling the layered stack

development team: we decided to incorporate the guava library for the camel case conversion requirement.

"take it out. I only want you to use the core java api for this application. period."

controlling the boundaries

controlling the layered stack

development team: we decided to incorporate the guava library for the camel case conversion requirement.

"guava - that's a cool library name. carry on..."

controlling the boundaries

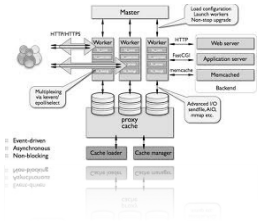
controlling the layered stack

development team: we decided to incorporate the guava library for the camel case conversion requirement.

if that's the only feature you are leveraging, you should just use the java api. if there are other features you can justify, then we can talk about it.

controlling the boundaries

controlling the layered stack



what design principle would you create to manage this type of boundary?

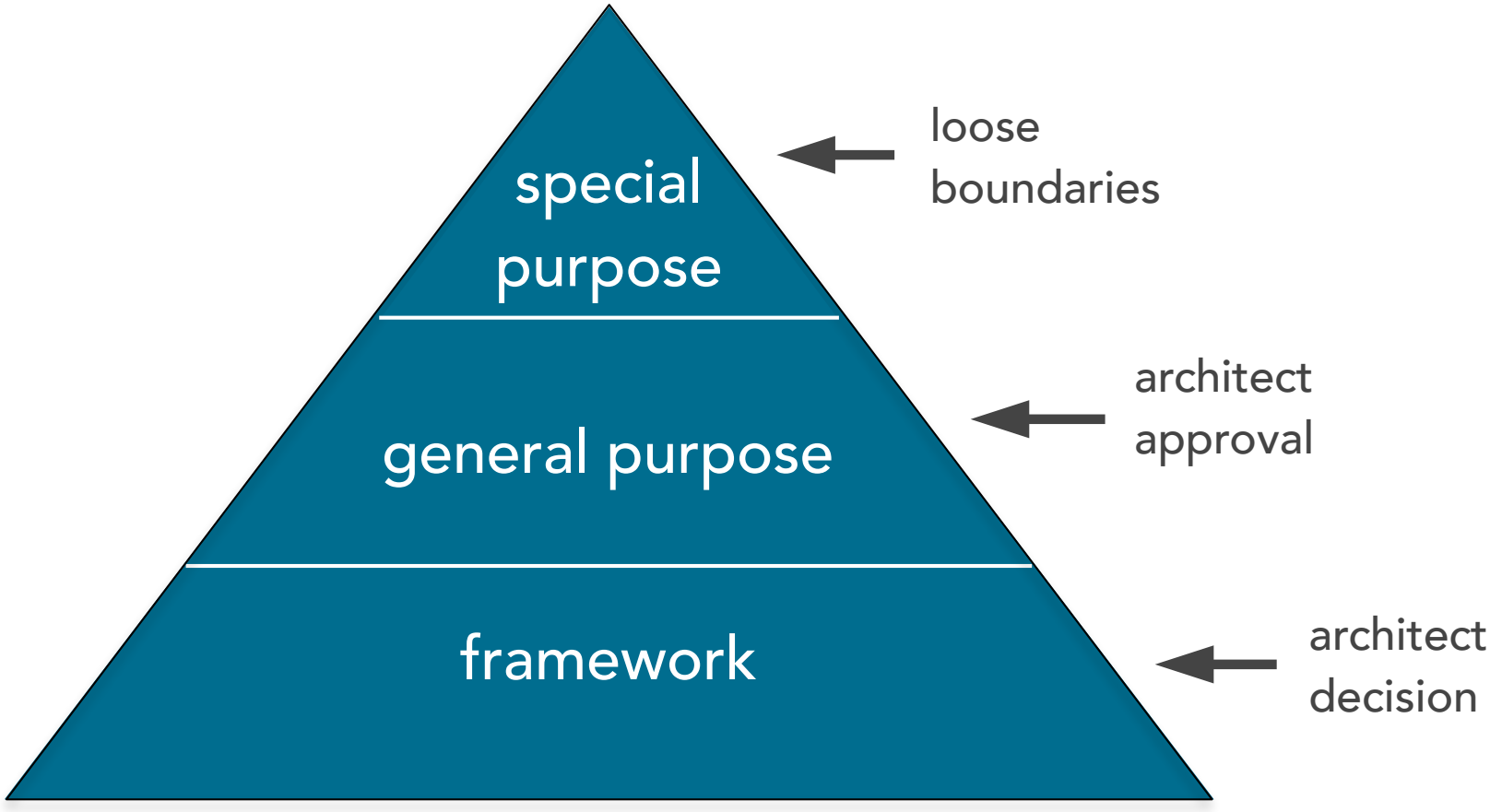
look for overlaps in existing functionality

always seek justification for adding a new library

provide guidance by making it clear what type of libraries need discussion and approval and which ones don't

controlling the boundaries

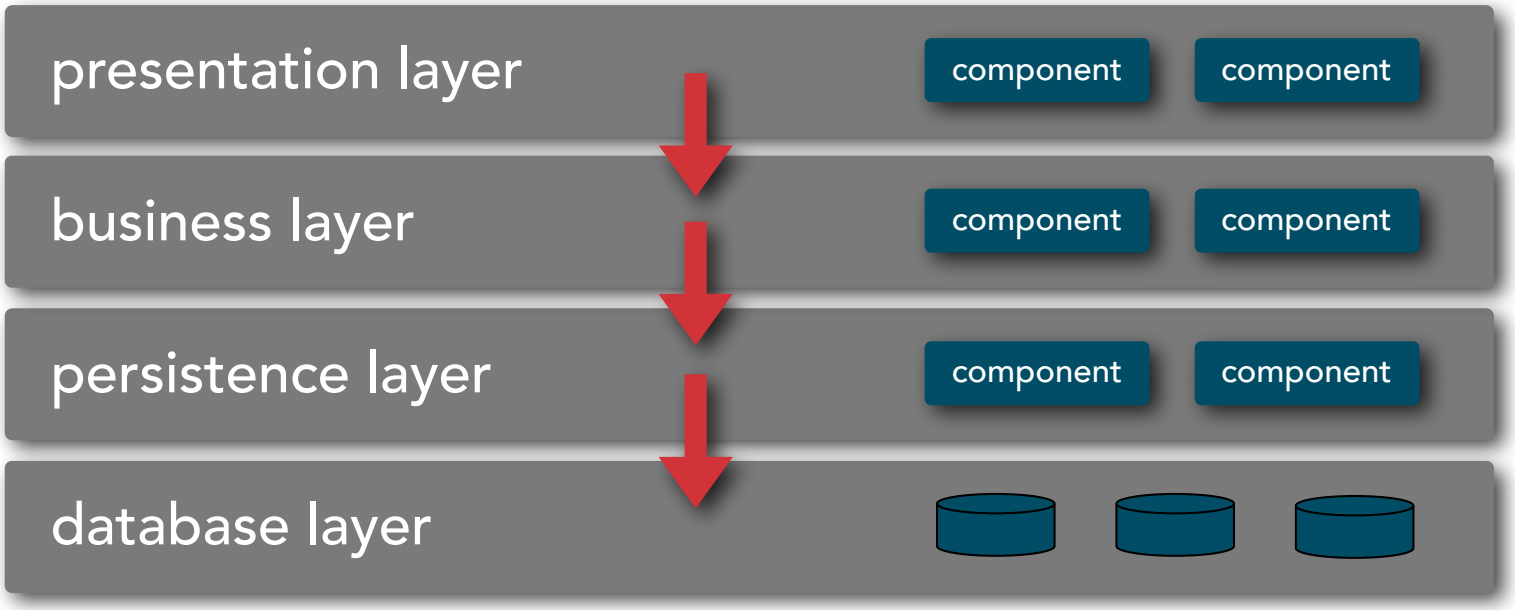
controlling the layered stack



controlling the boundaries

pattern governance

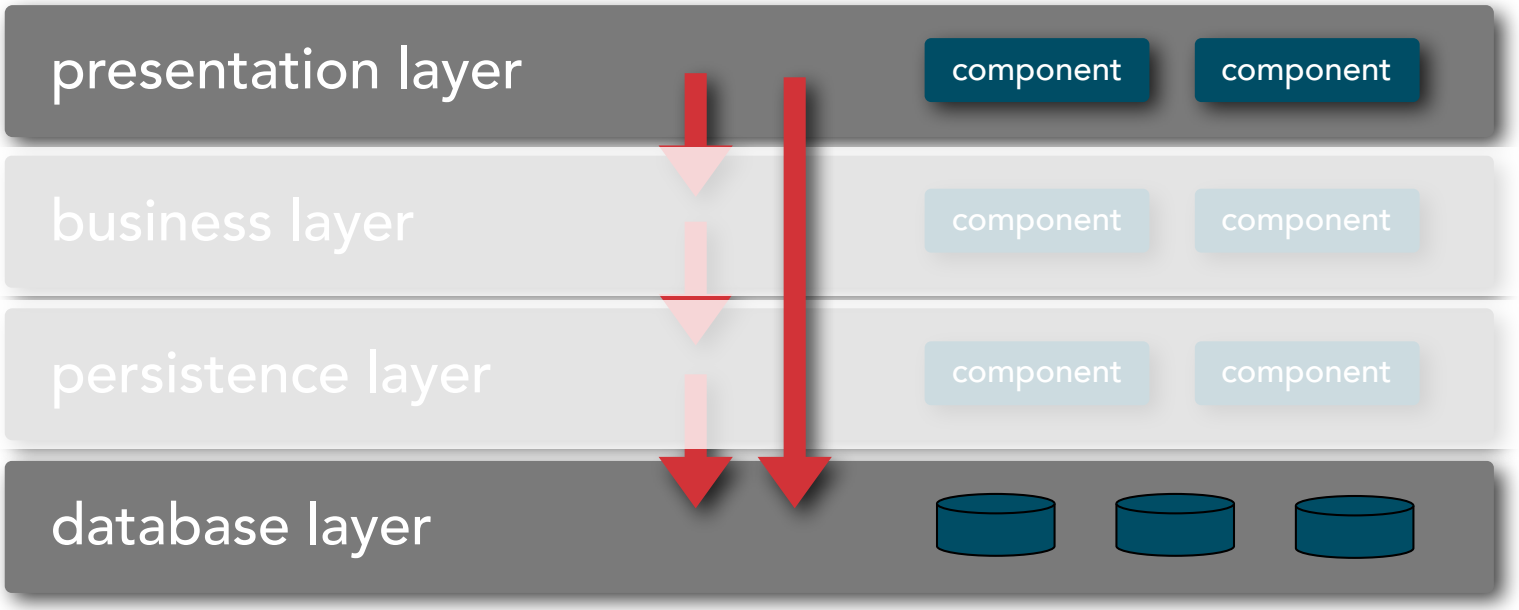
development team: we need better performance - can we access the database directly from the presentation layer?



controlling the boundaries

pattern governance

development team: we need better performance - can we access the database directly from the presentation layer?



controlling the boundaries

pattern governance

development team: we need better performance - can we access the database directly from the presentation layer?

"no."

controlling the boundaries

pattern governance

development team: we need better performance - can we access the database directly from the presentation layer?

"it doesn't matter to me. if you think it would help performance, then go for it."

controlling the boundaries

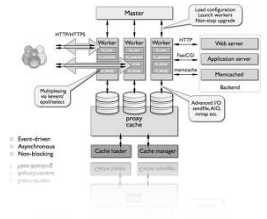
pattern governance

development team: we need better performance - can we access the database directly from the presentation layer?

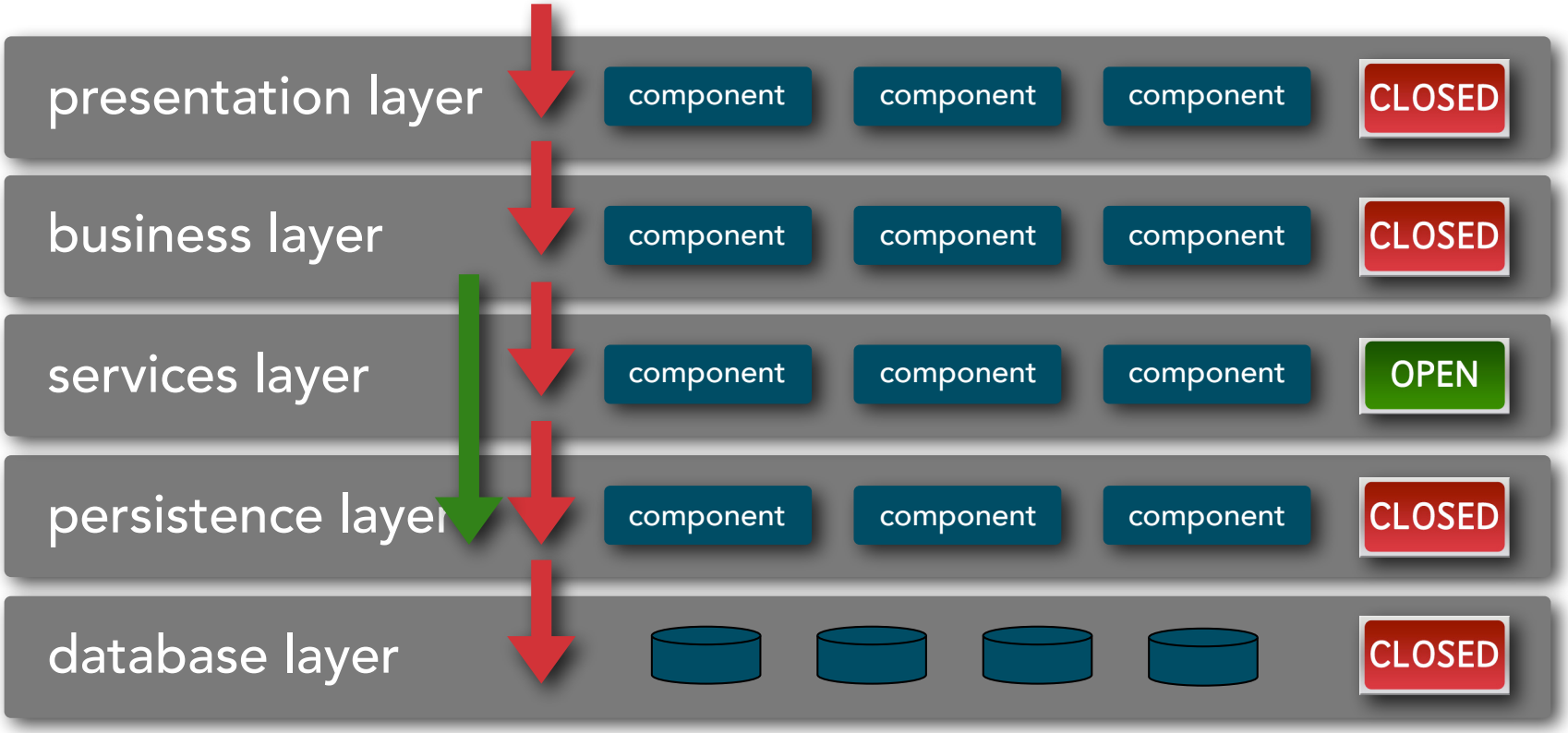
"those layers are closed so that we can better control change through layer isolation, so no. have you been able to identify what might be causing the performance issues?"

controlling the boundaries

pattern governance

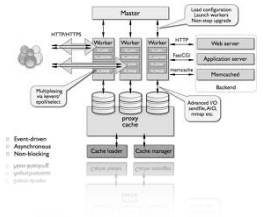


what design principle would you create to manage this type of boundary?



controlling the boundaries

pattern governance



what design principle would you create to manage this type of boundary?

clearly document and diagram the architecture

justify your reasons for the architecture decisions

make sure you effectively communicate your decisions

controlling the boundaries

architecture scope

development team: we added some really cool capabilities that might be needed sometime in the future...

"did i tell you to add those capabilities? didn't think so. take them out."

controlling the boundaries

architecture scope

development team: we added some really cool capabilities that might be needed sometime in the future...

"great forward thinking guys! someday the users might need that capability, and now we have it ready..."

controlling the boundaries

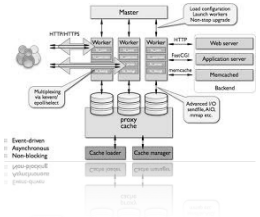
architecture scope

development team: we added some really cool capabilities that might be needed sometime in the future...

"let's verify those features with the analysts to see if we need them. if not then we'll take them out. we just need to make sure we don't do anything to prevent that capability from being added in the future."

controlling the boundaries

architecture scope



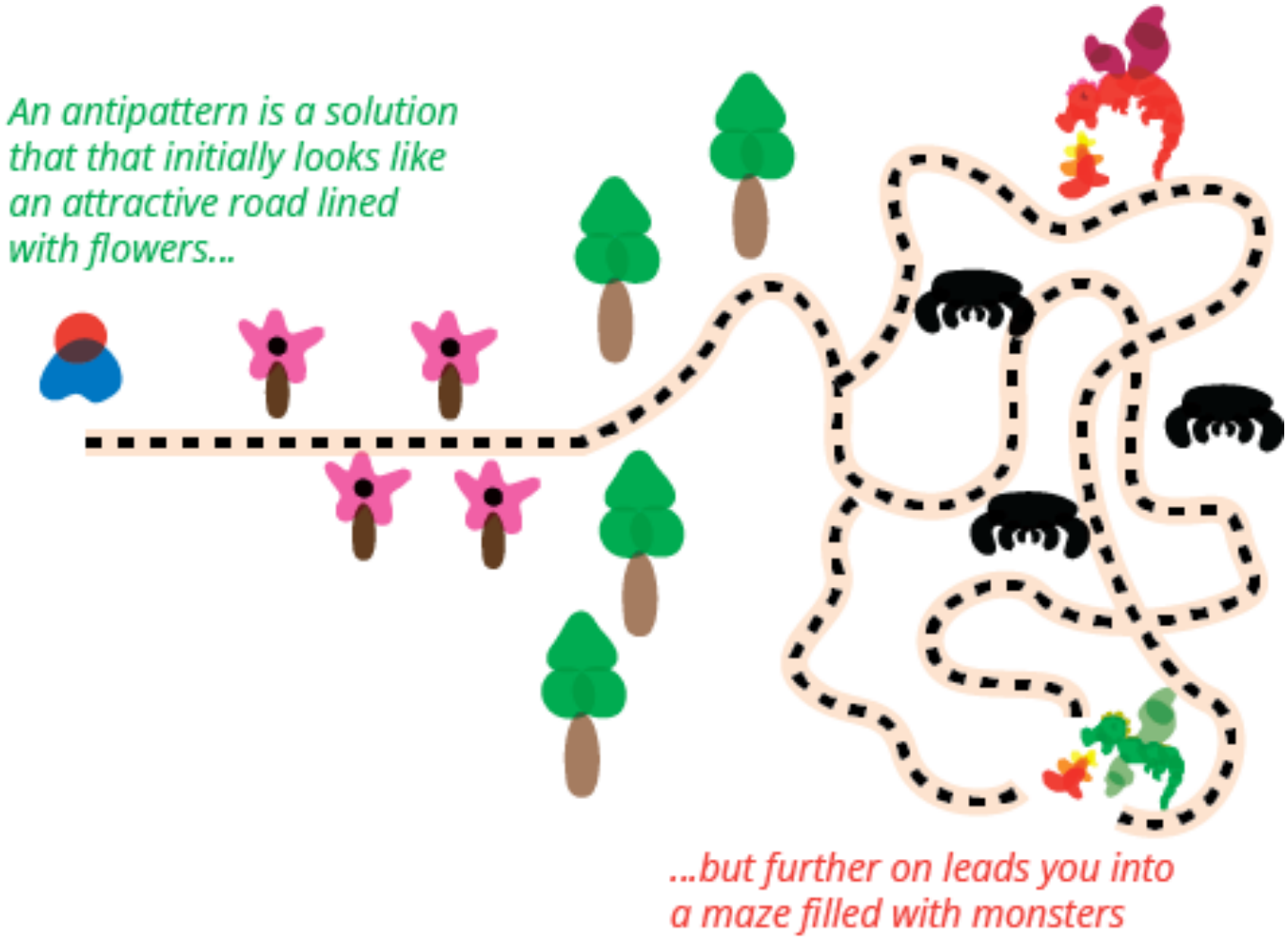
what design principle would you create to manage this type of boundary?

adding additional features over and above the requirements adds additional development, testing, and maintenance time and costs

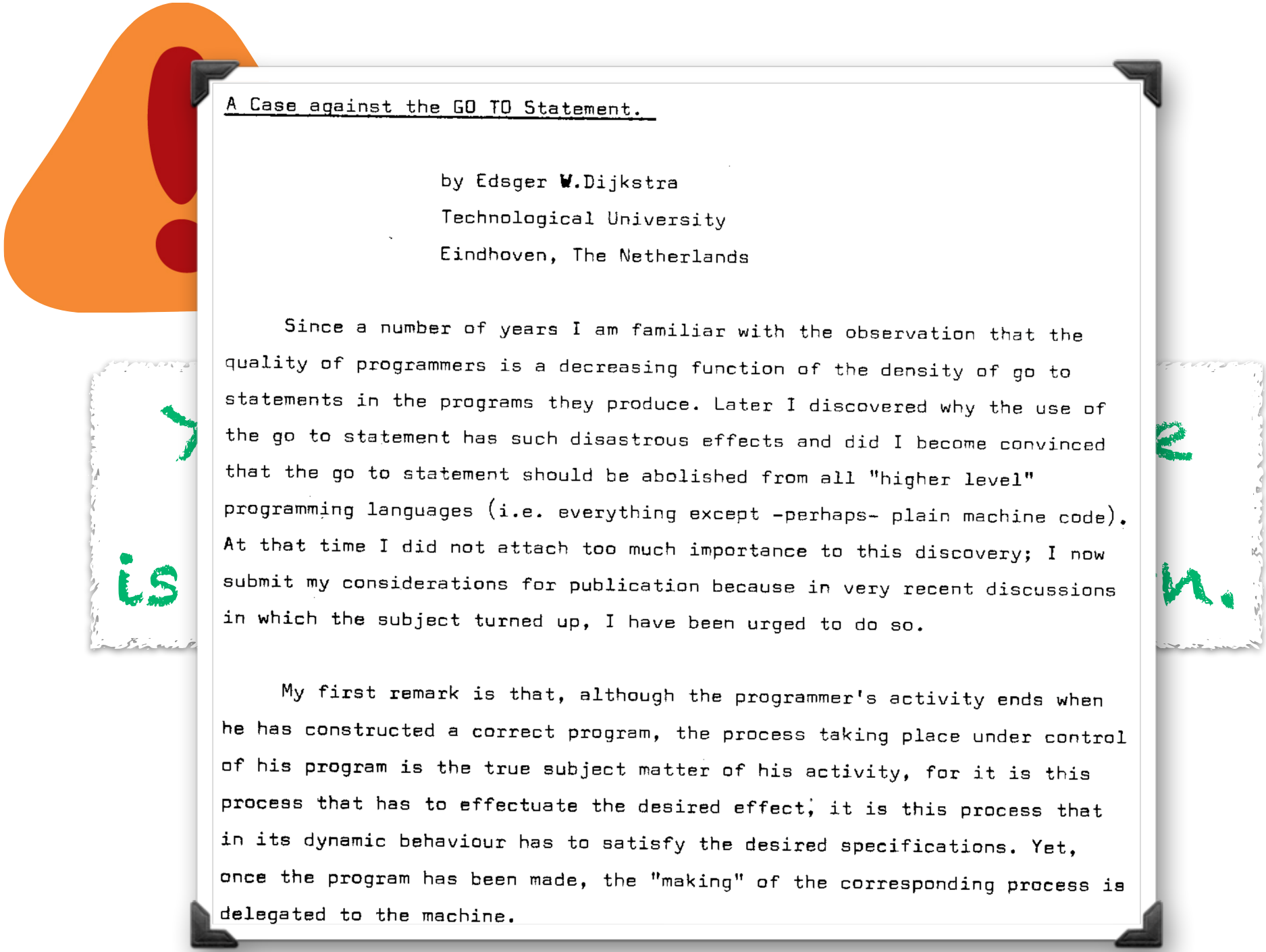
this practice can lead to the *infinity architecture anti-pattern*

document non-required features, verify it with the user community, and ***make sure you don't do anything to restrict that functionality in the future***

Architecture Anti-pattern



Yesterday's best practice is tomorrow's anti-pattern.



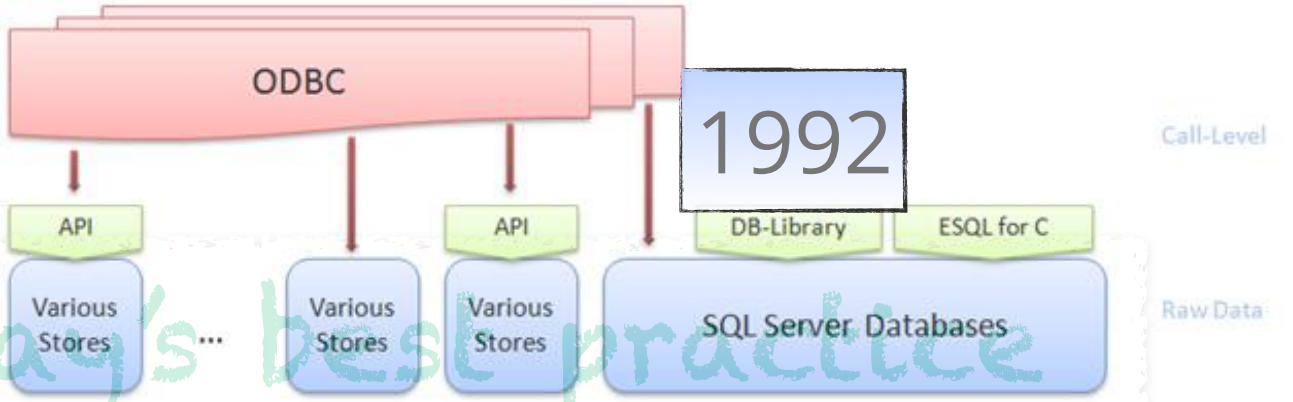
A Case against the GO TO Statement.

by Edsger W.Dijkstra
Technological University
Eindhoven, The Netherlands

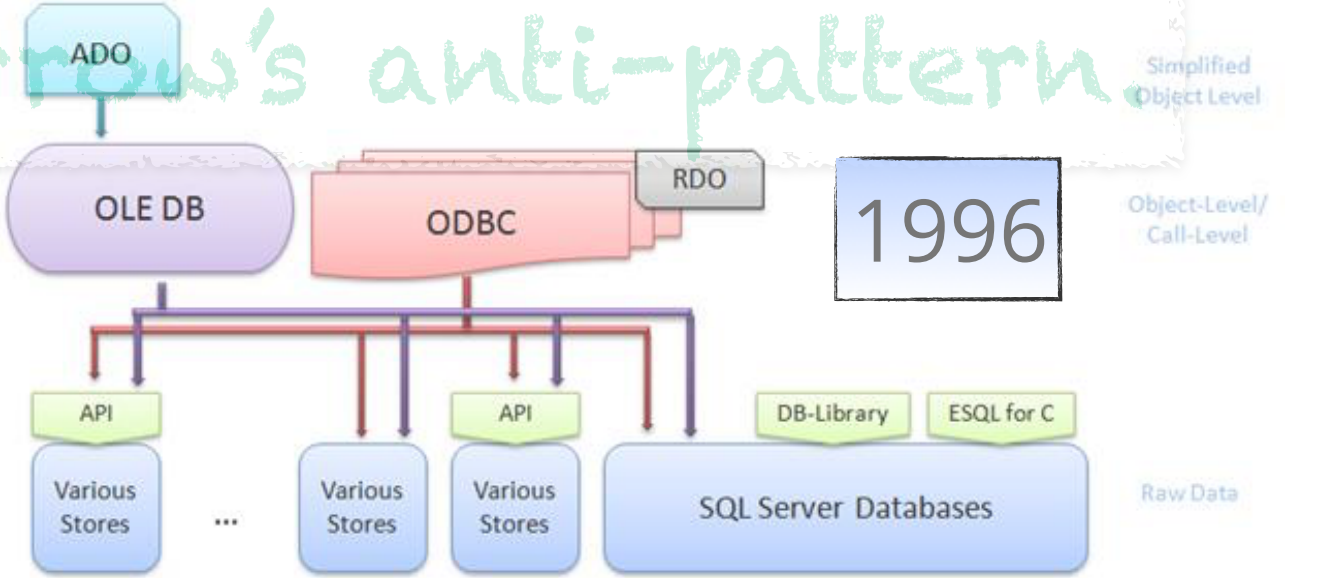
Since a number of years I am familiar with the observation that the quality of programmers is a decreasing function of the density of go to statements in the programs they produce. Later I discovered why the use of the go to statement has such disastrous effects and did I become convinced that the go to statement should be abolished from all "higher level" programming languages (i.e. everything except -perhaps- plain machine code). At that time I did not attach too much importance to this discovery; I now submit my considerations for publication because in very recent discussions in which the subject turned up, I have been urged to do so.

My first remark is that, although the programmer's activity ends when he has constructed a correct program, the process taking place under control of his program is the true subject matter of his activity, for it is this process that has to effectuate the desired effect; it is this process that in its dynamic behaviour has to satisfy the desired specifications. Yet, once the program has been made, the "making" of the corresponding process is delegated to the machine.

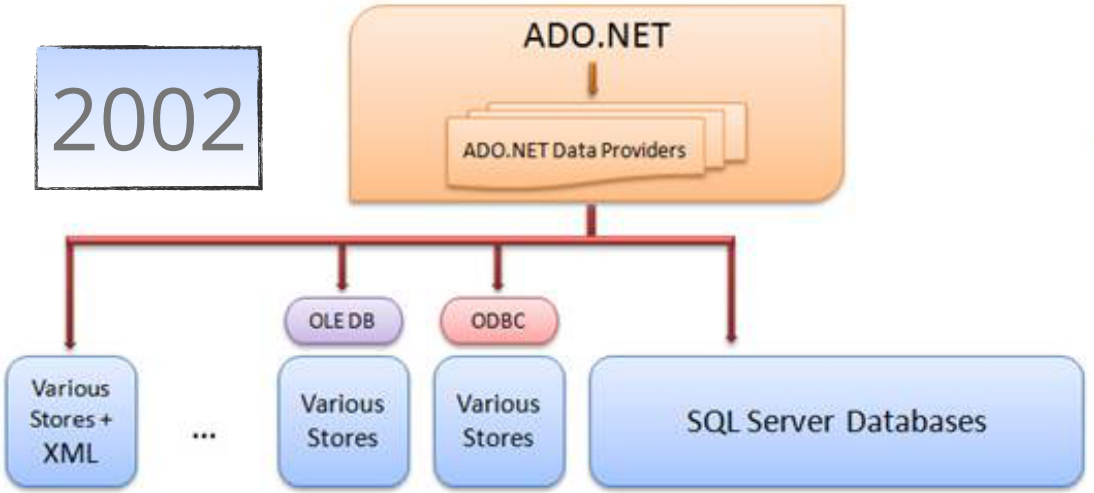
www.cs.utexas.edu/users/EWD/ewd02xx/EWD215.PDF



Yesterday's best practice is tomorrow's anti-pattern.

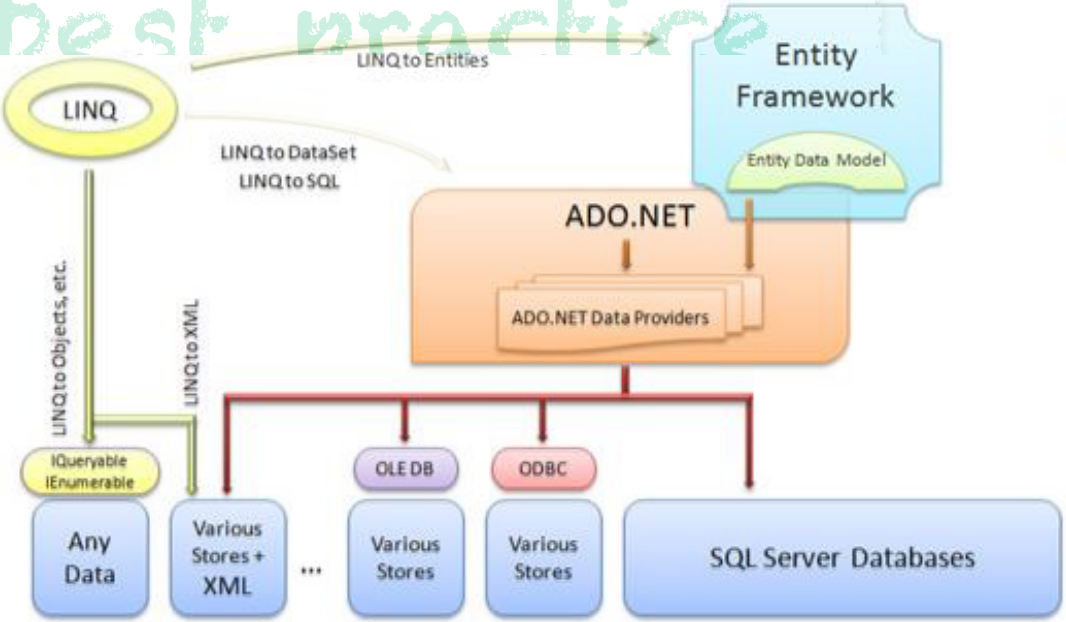


2002

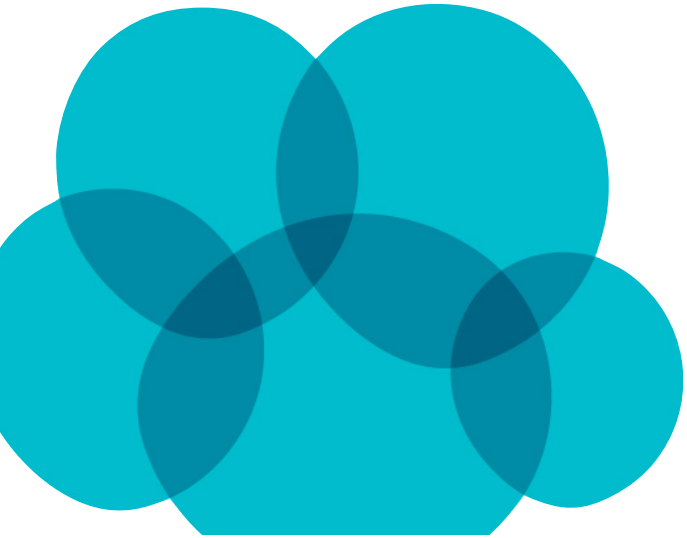


Yesterday's best practice is tomorrow's

2007



2008



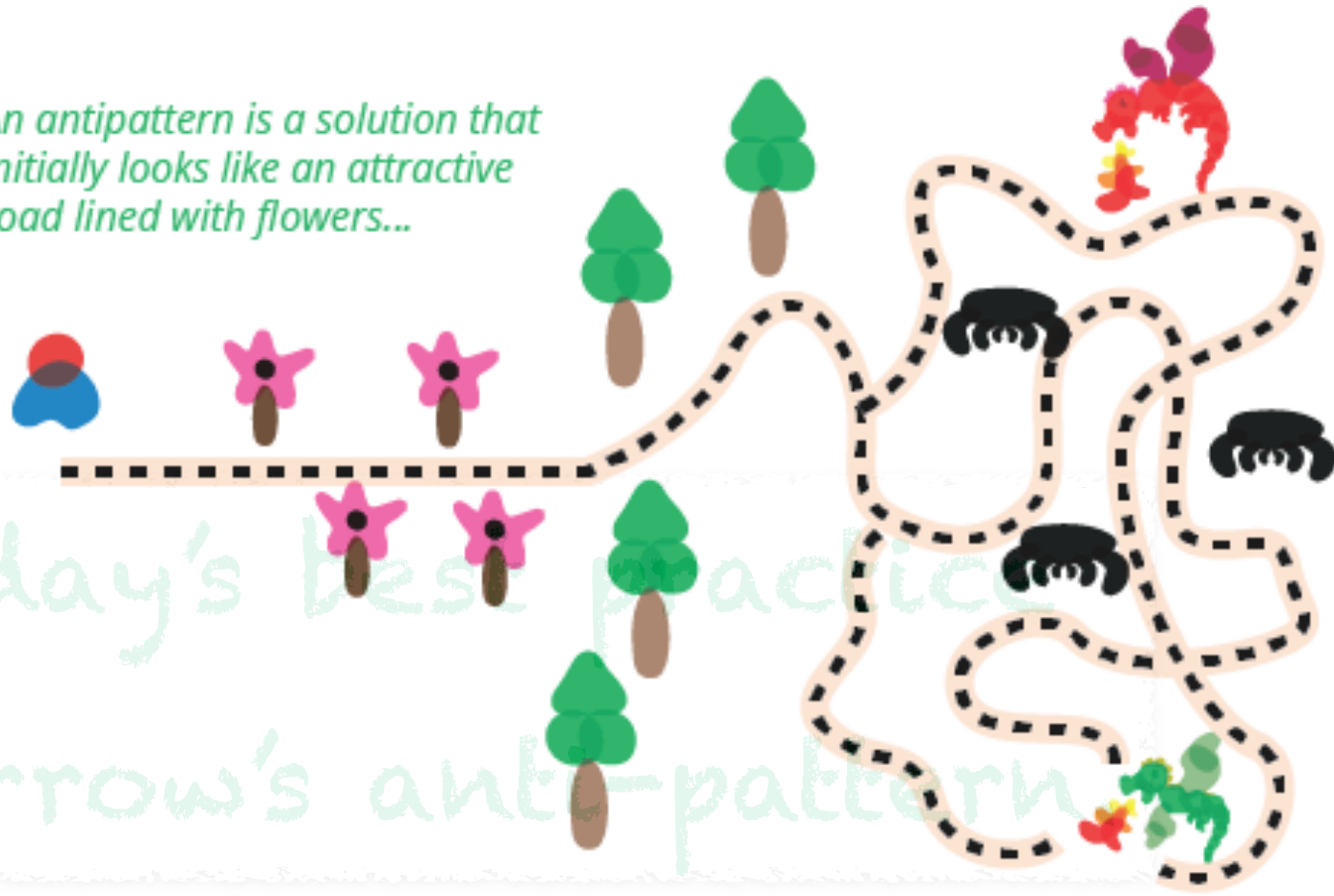
Yesterday's best practice

is tomorrow's anti-pattern.

ENTERPRISE

Java Beans

An antipattern is a solution that initially looks like an attractive road lined with flowers...



Yesterday's best practice

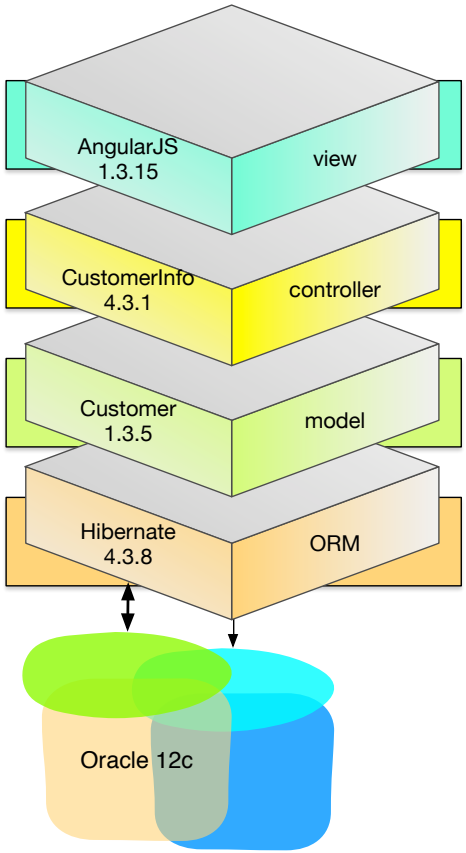
is tomorrow's anti-pattern

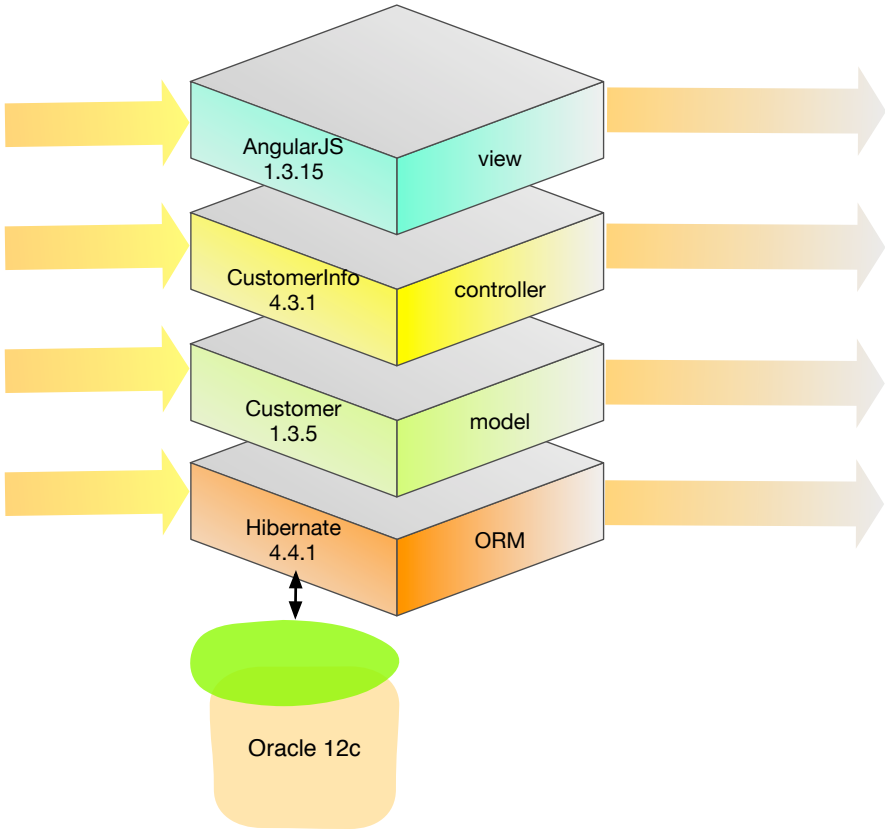
...but further on leads you into a maze filled with monsters

martinfowler.com/bliki/AntiPattern.html


Architecture is abstract
until operationalized.
nealford.com/memeagora/2015/03/30/architecture_is_abstract_until_operationalized.html

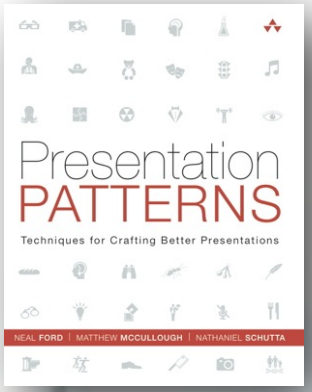
Architecture is abstract until operationalized.

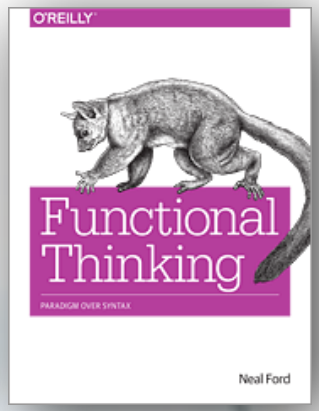




2D 3D 4D

nealford.com/memeagora/2015/03/30/architecture_is_abstract_until_operationalized.html





nealford.com

@neal4d

ThoughtWorks®

NEAL FORD

Director / Software Architect / Meme Wrangler