Software Design and Development Conference 2015

Software Architecture Patterns

Mark Richards

Hands-on Software Architect

Author of *Enterprise Messaging Video Series* (O’Reilly)

Author of *Java Message Service 2nd Edition* (O’Reilly)

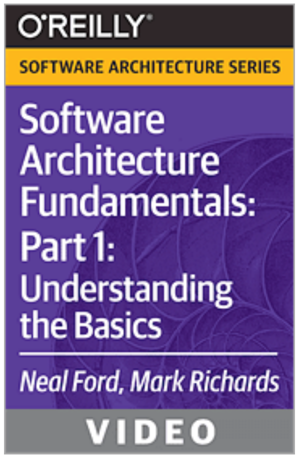
Co-author of *Software Architecture Fundamentals Video Series* (O'Reilly)

{SDD} 2015

Software Design & Development

Barbican Centre, London, 11-15 May 2015

Software Architecture Fundamentals Video Series
Enterprise Messaging Video Series

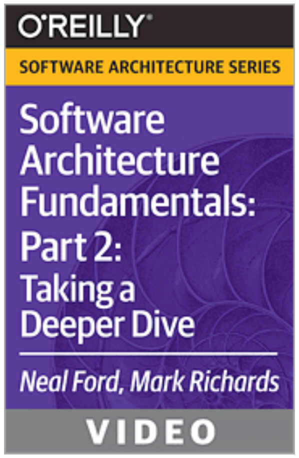

O'REILLY

SOFTWARE ARCHITECTURE SERIES

Software Architecture Fundamentals: Part 1: Understanding the Basics

Neal Ford, Mark Richards

VIDEO

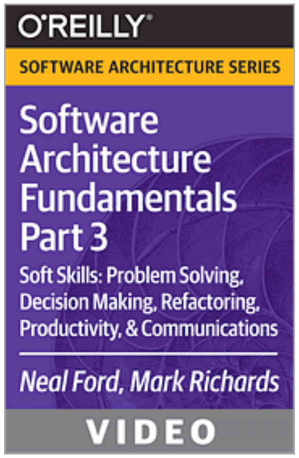

O'REILLY

SOFTWARE ARCHITECTURE SERIES

Software Architecture Fundamentals: Part 2: Taking a Deeper Dive

Neal Ford, Mark Richards

VIDEO


O'REILLY

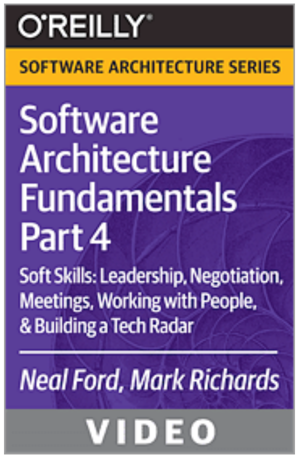
SOFTWARE ARCHITECTURE SERIES

Software Architecture Fundamentals Part 3

Soft Skills: Problem Solving, Decision Making, Refactoring, Productivity, & Communications

Neal Ford, Mark Richards

VIDEO


O'REILLY

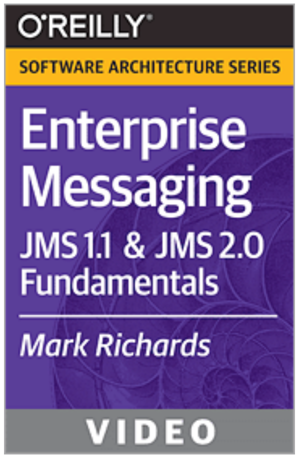
SOFTWARE ARCHITECTURE SERIES

Software Architecture Fundamentals Part 4

Soft Skills: Leadership, Negotiation, Meetings, Working with People, & Building a Tech Radar

Neal Ford, Mark Richards

VIDEO


O'REILLY

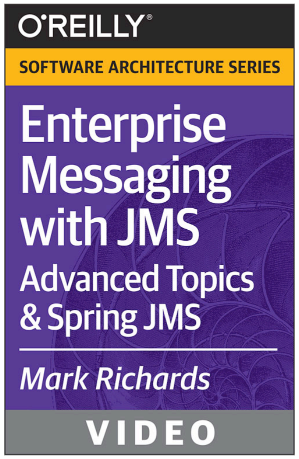
SOFTWARE ARCHITECTURE SERIES

Enterprise Messaging

JMS 1.1 & JMS 2.0 Fundamentals

Mark Richards

VIDEO


O'REILLY

SOFTWARE ARCHITECTURE SERIES

Enterprise Messaging with JMS

Advanced Topics & Spring JMS

Mark Richards

VIDEO

agenda

introduction

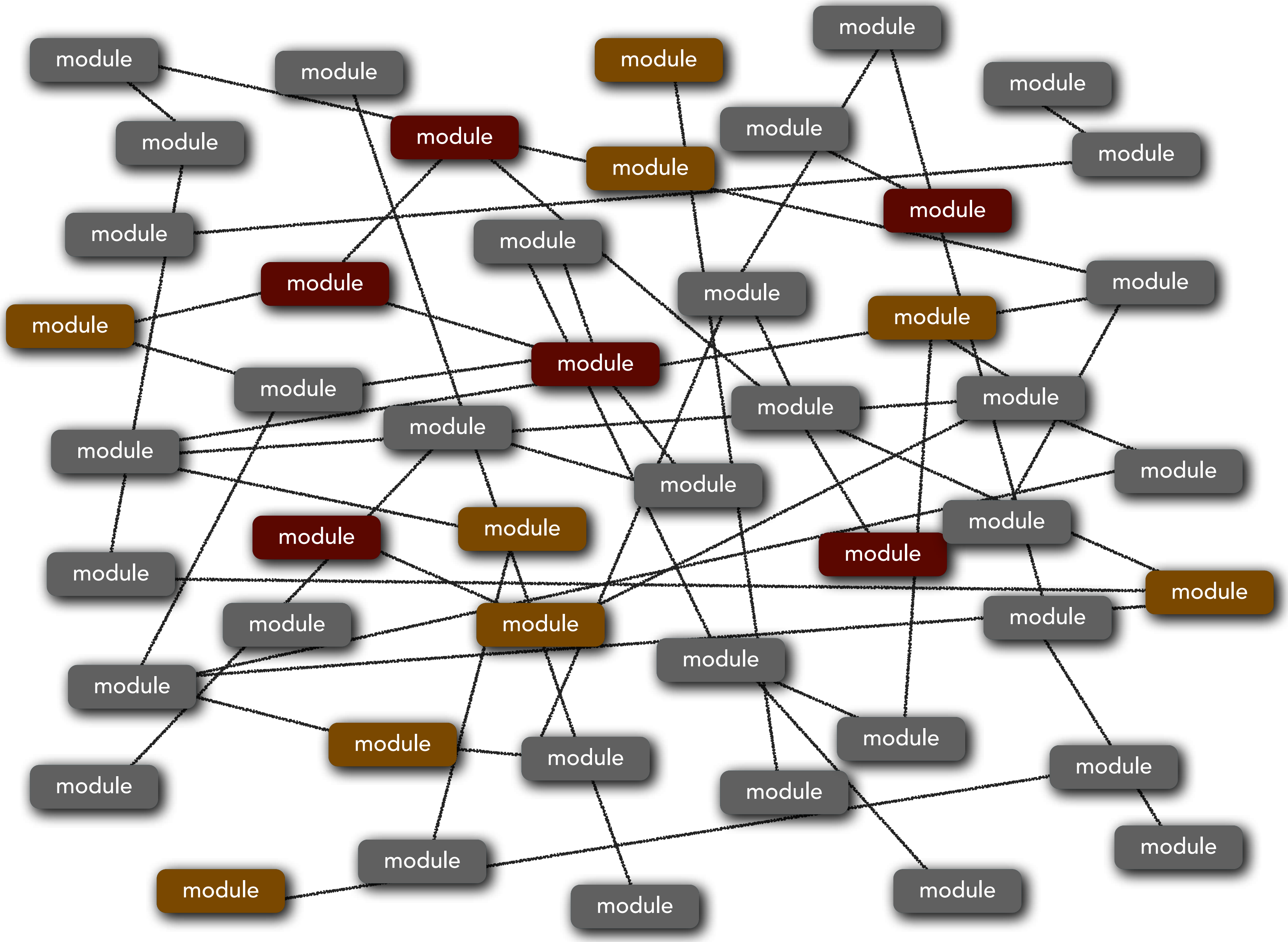
layered architecture pattern

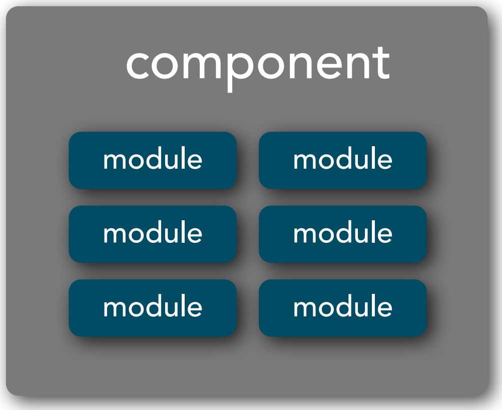
event-driven architecture pattern

microkernel architecture pattern

space-based architecture pattern

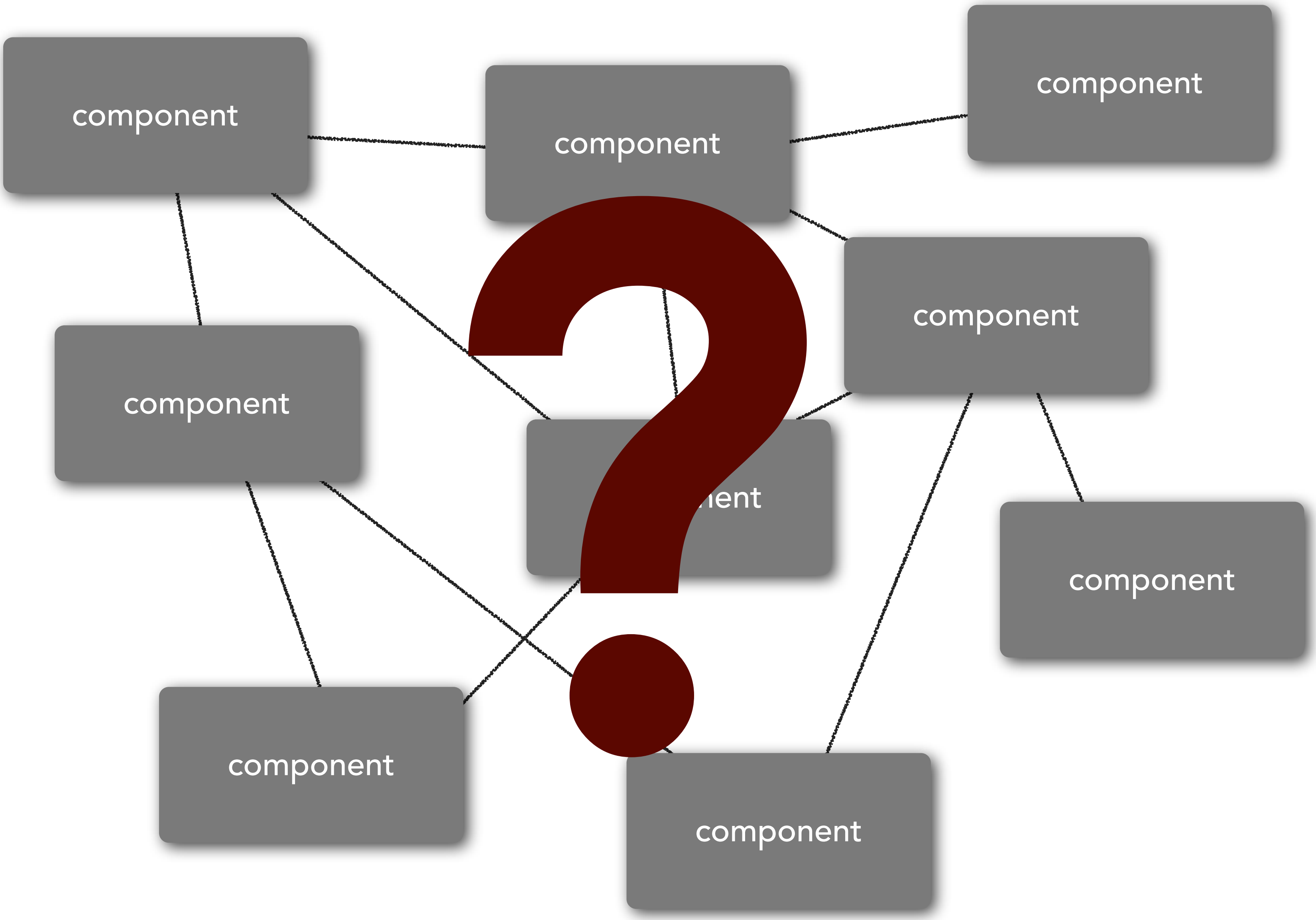
Software Architecture Pattern Analysis

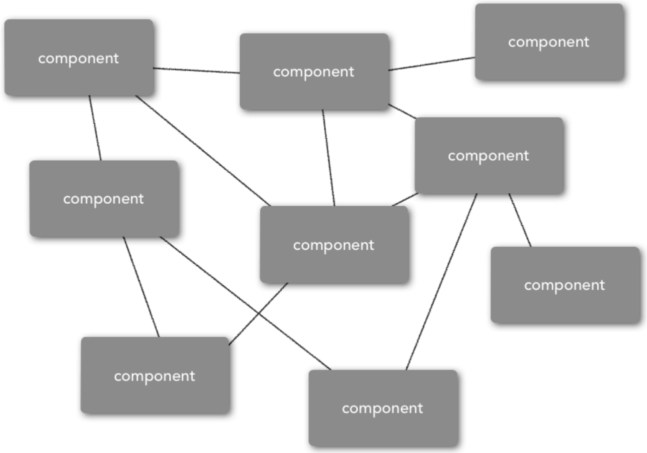

module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module



component

an encapsulated unit of software consisting of one or more modules that has a specific role and responsibility in the system


component
component
component
component
component
ent
component
component
component
component



how are components classified?

how do components interact?

does the architecture scale?

how responsive is the architecture?

is there a logical flow to the components?

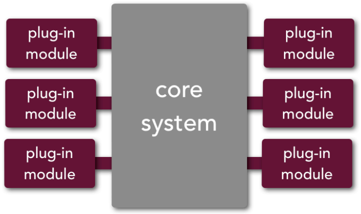
what are the deployment characteristics?

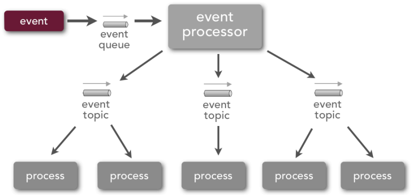
how does the architecture respond to change?

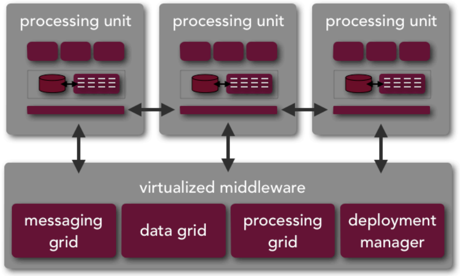
is the architecture extensible and if so how?

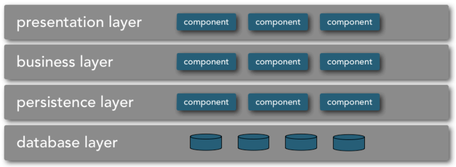
how maintainable is the architecture?

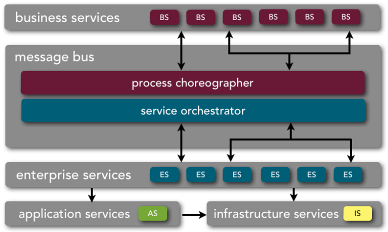
architecture patterns help define the basic characteristics and behavior of the application

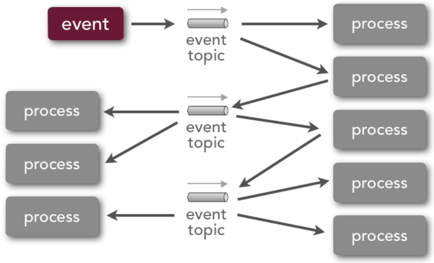


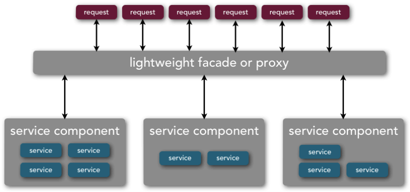


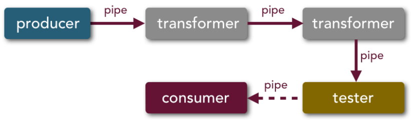




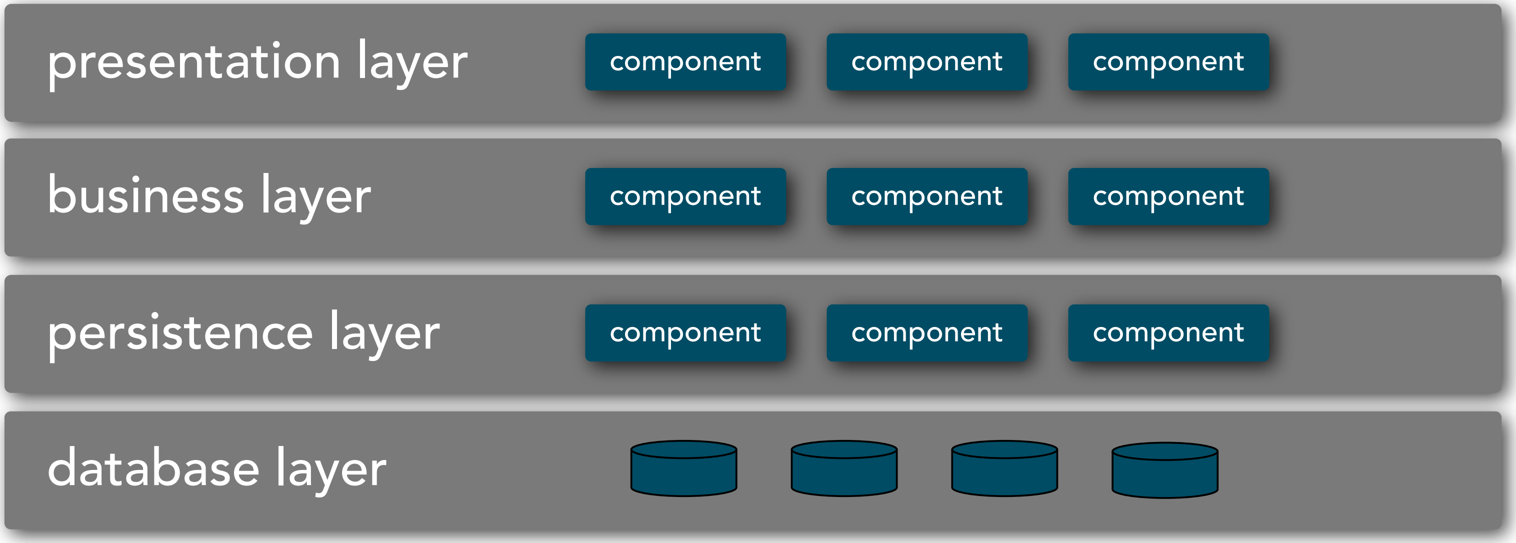








layered architecture


presentation layer

component

component

component

business layer

component

component

component

persistence layer

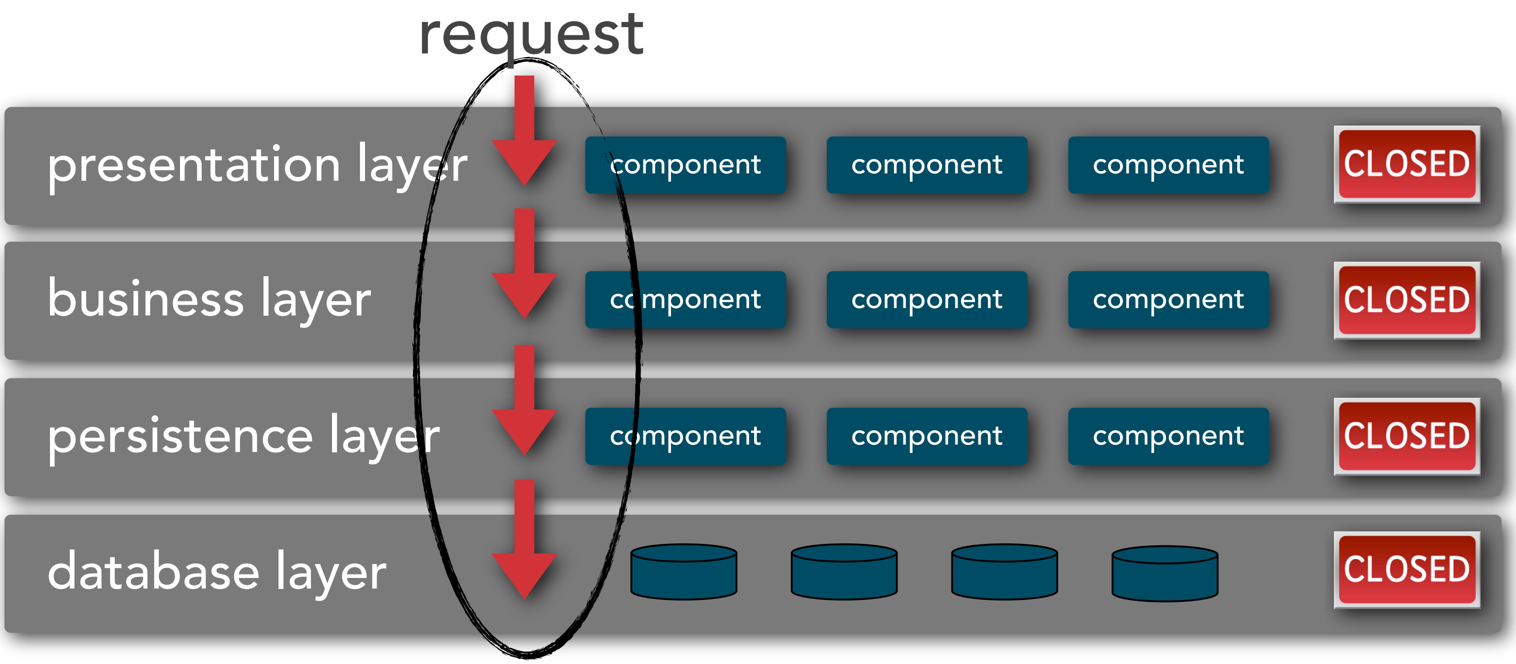
component

component

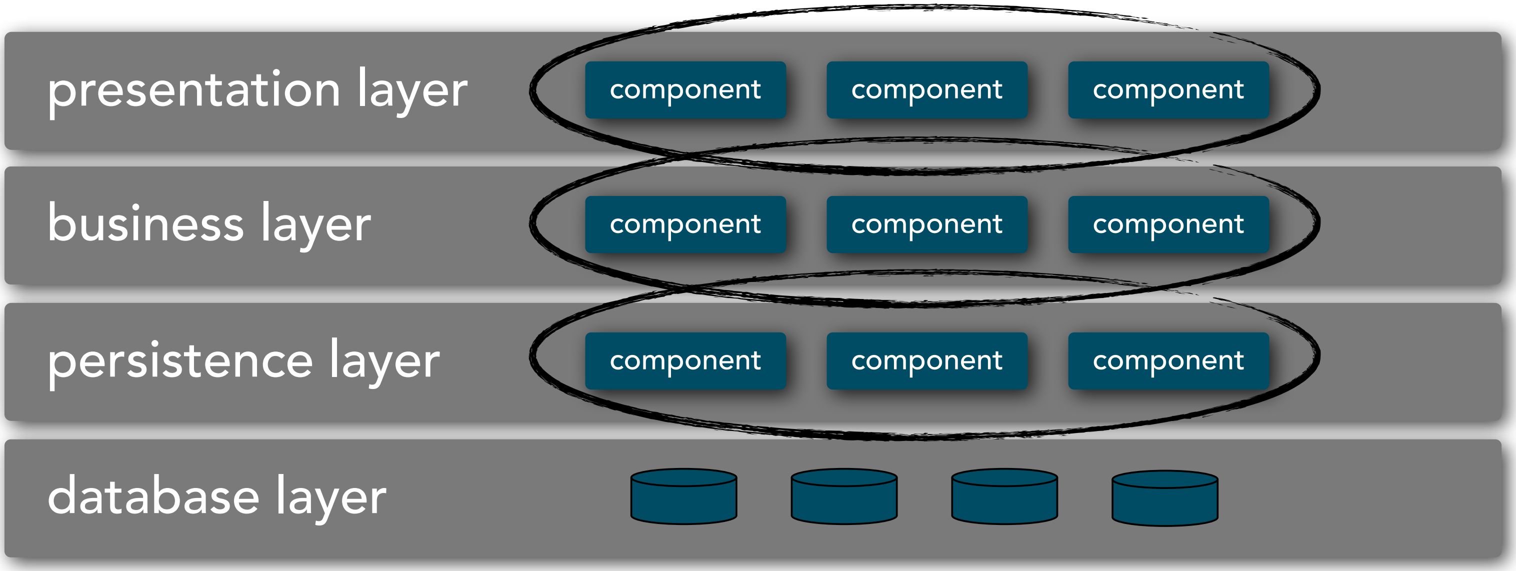
component

database layer

layered architecture

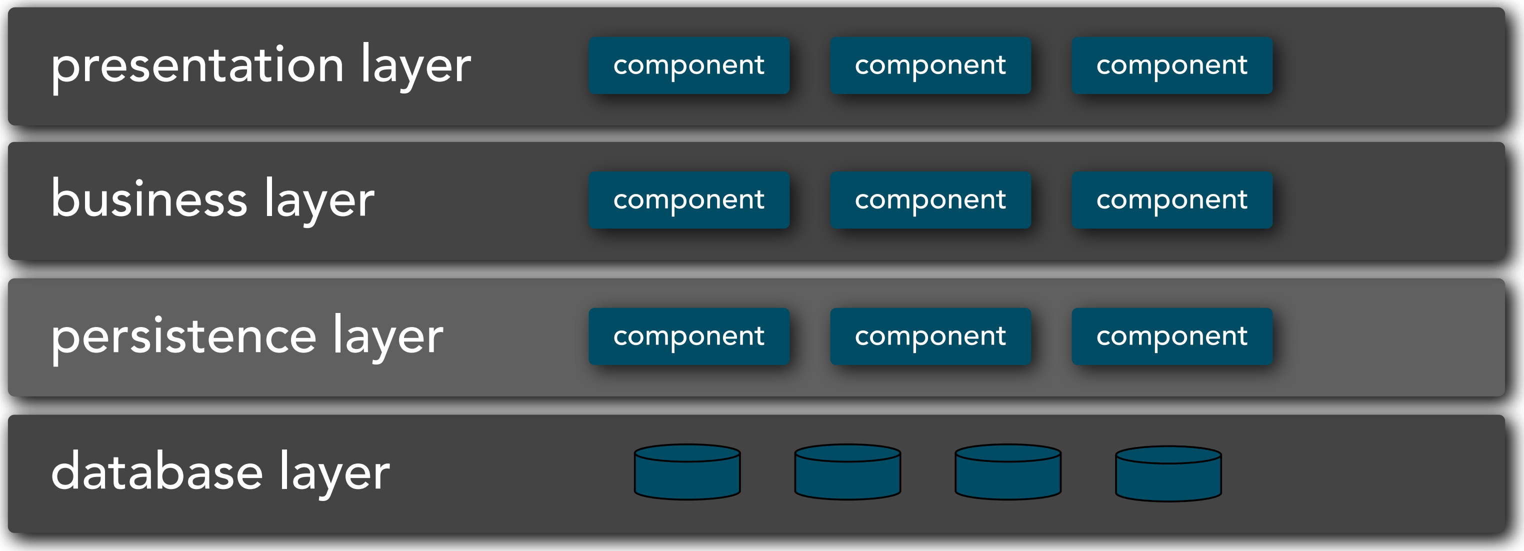


layered architecture



separation of concerns

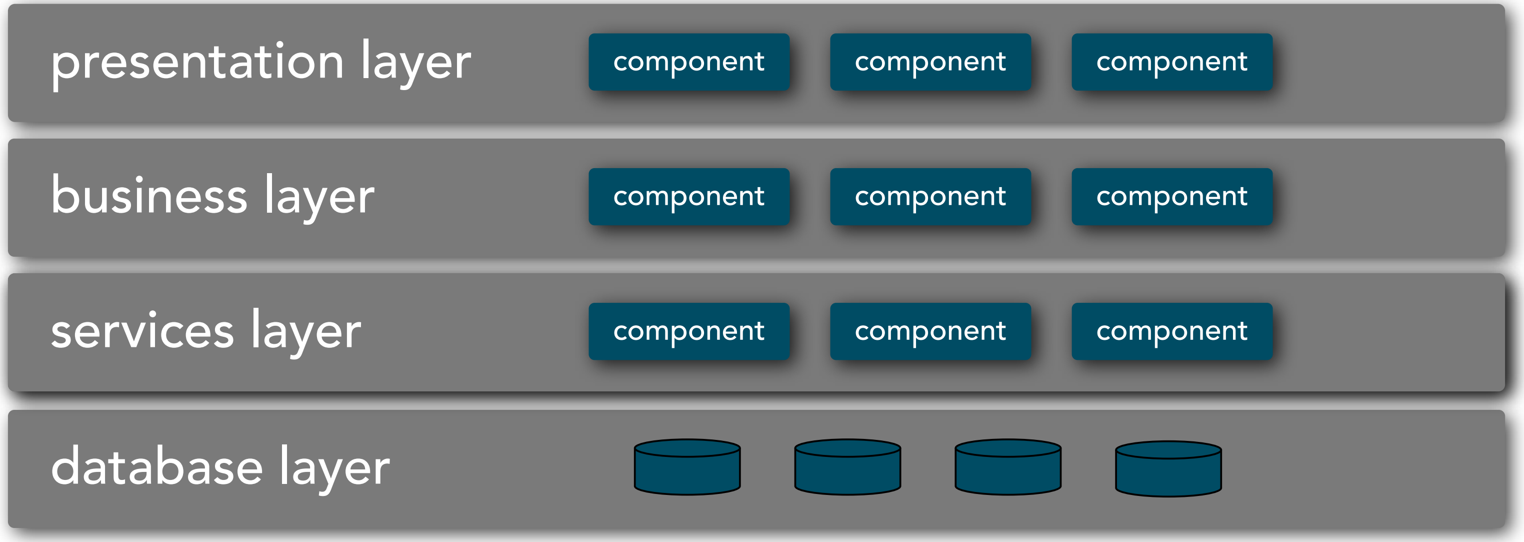
layered architecture



layers of isolation

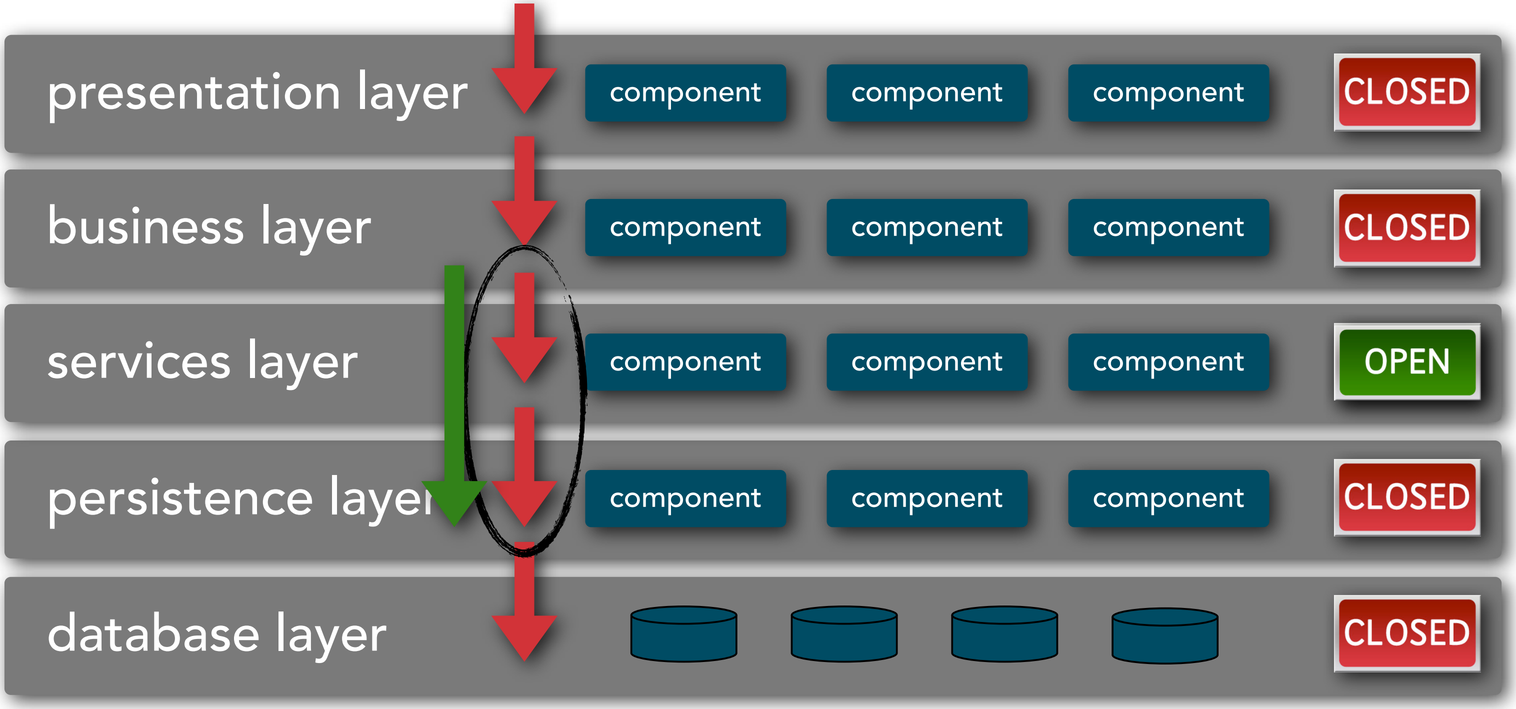
layered architecture

hybrids and variants



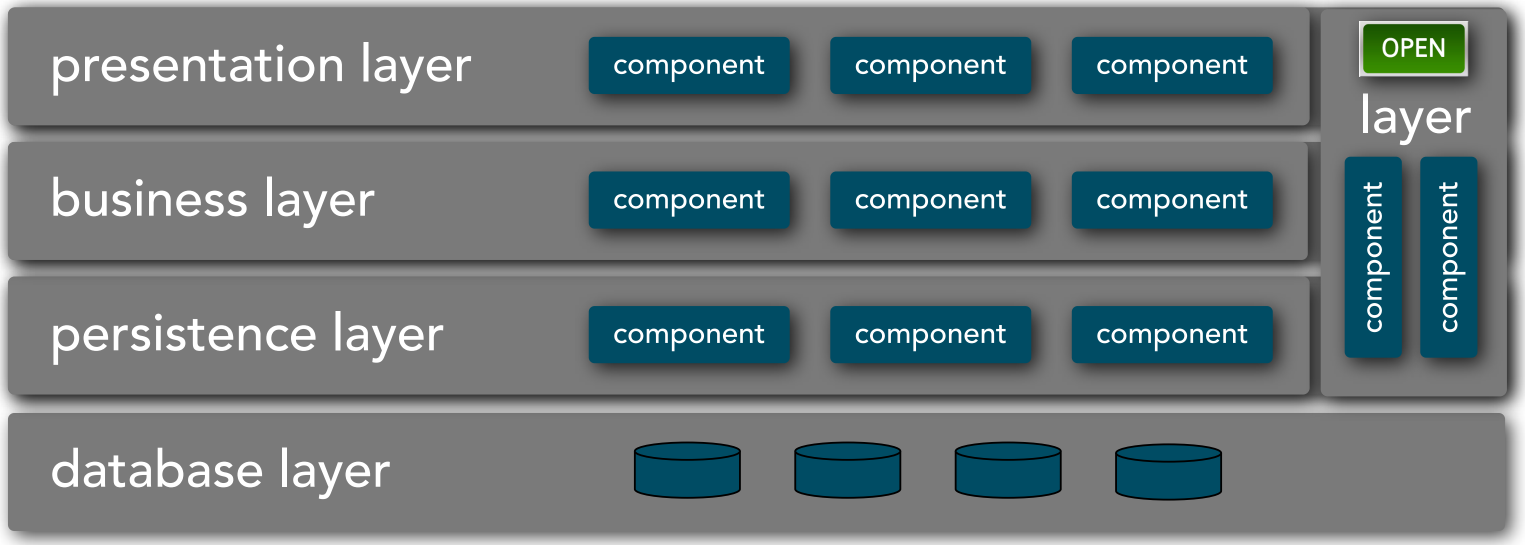
layered architecture

hybrids and variants



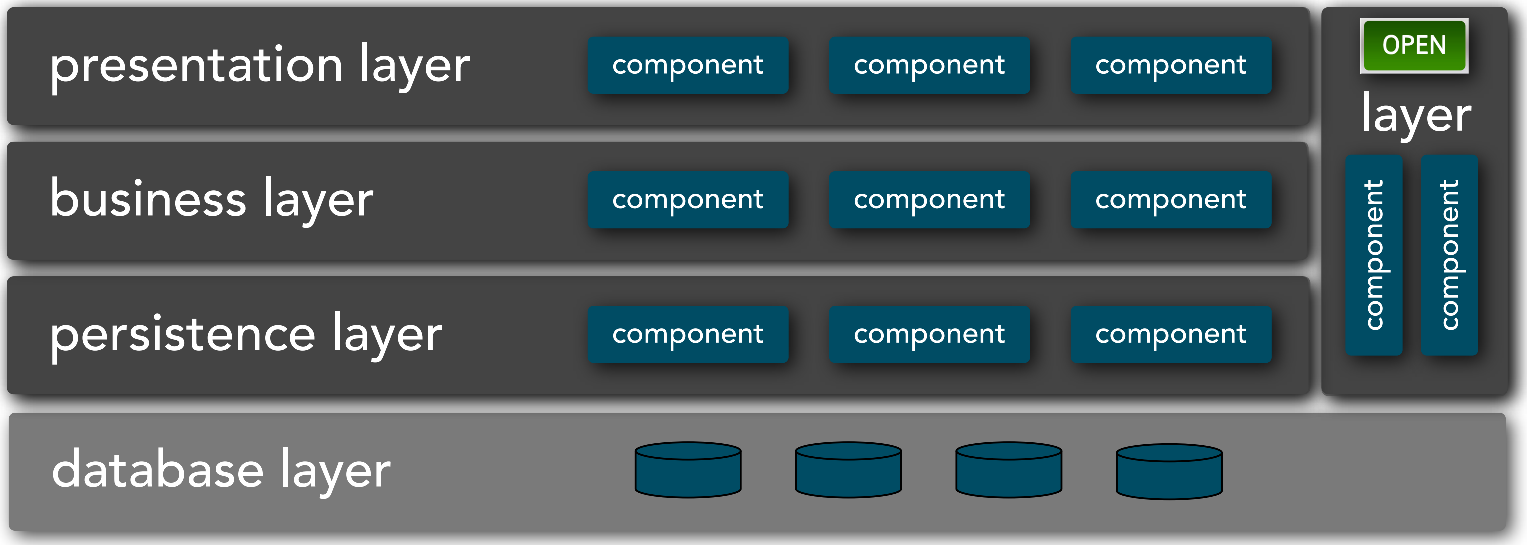
layered architecture

hybrids and variants



layered architecture

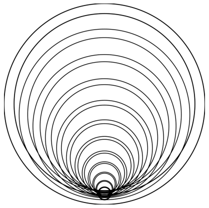
hybrids and variants



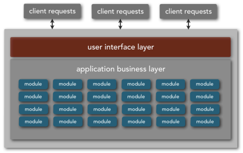
layered architecture

considerations

good general purpose architecture and a good starting point for most systems



watch out for the architecture sinkhole anti-pattern

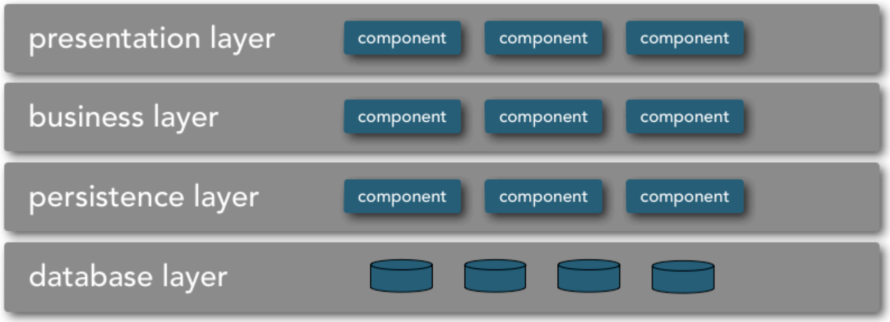


tends to lend itself towards monolithic applications

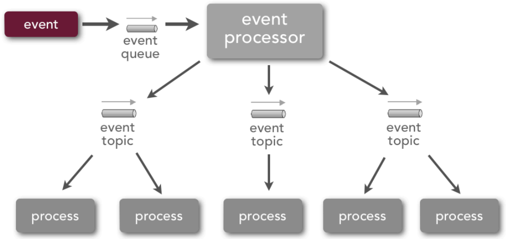
layered architecture

analysis

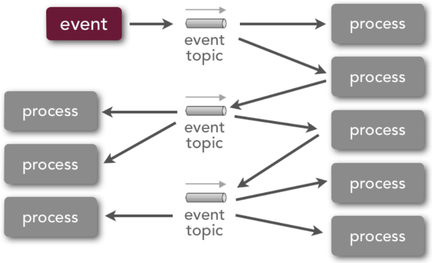
overall agility	👎
deployment	👎
testability	👍
performance	👎
scalability	👎
development	👍
complexity	👍
loose coupling	👎



event-driven architecture



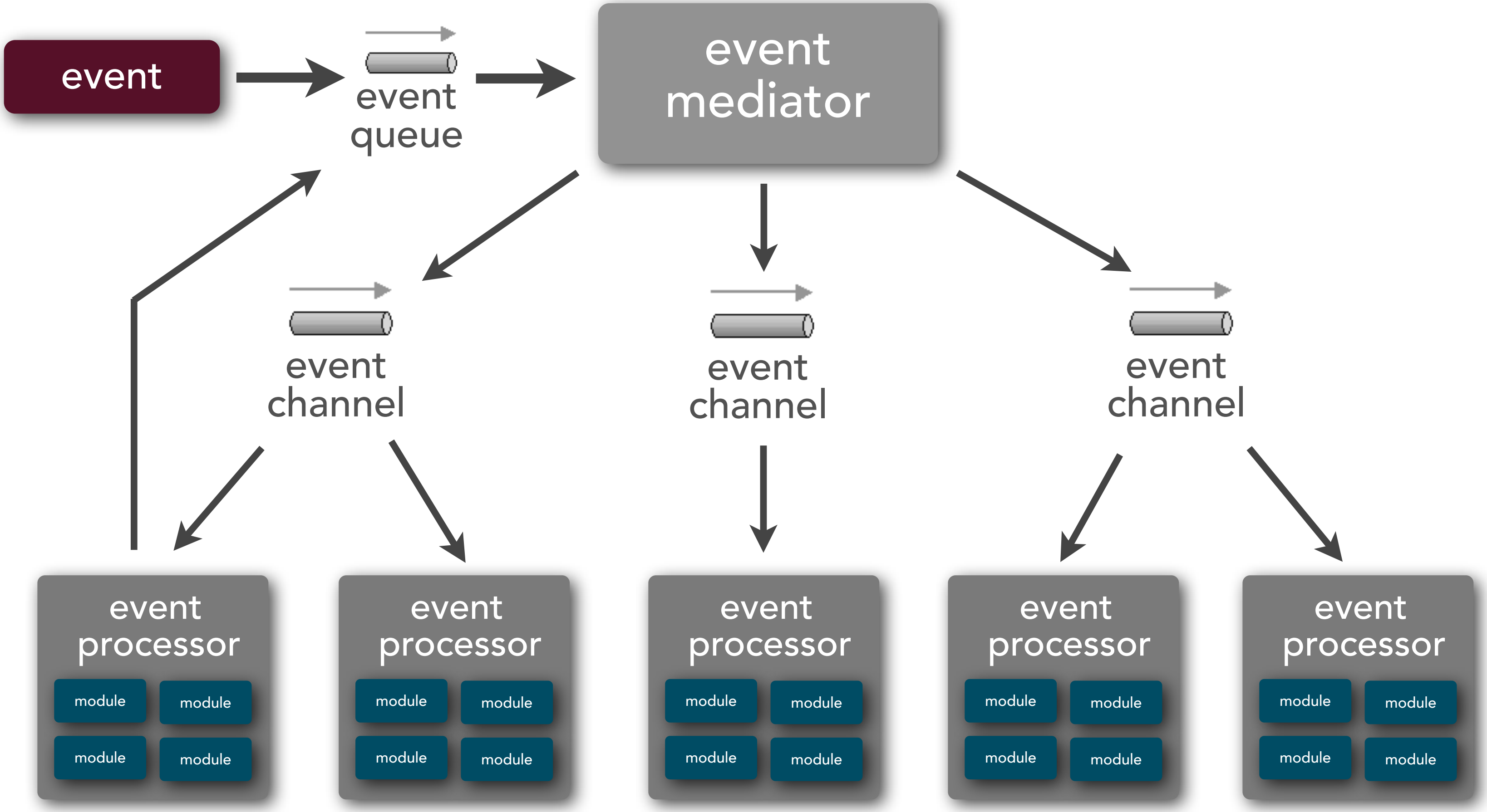
mediator topology



broker topology

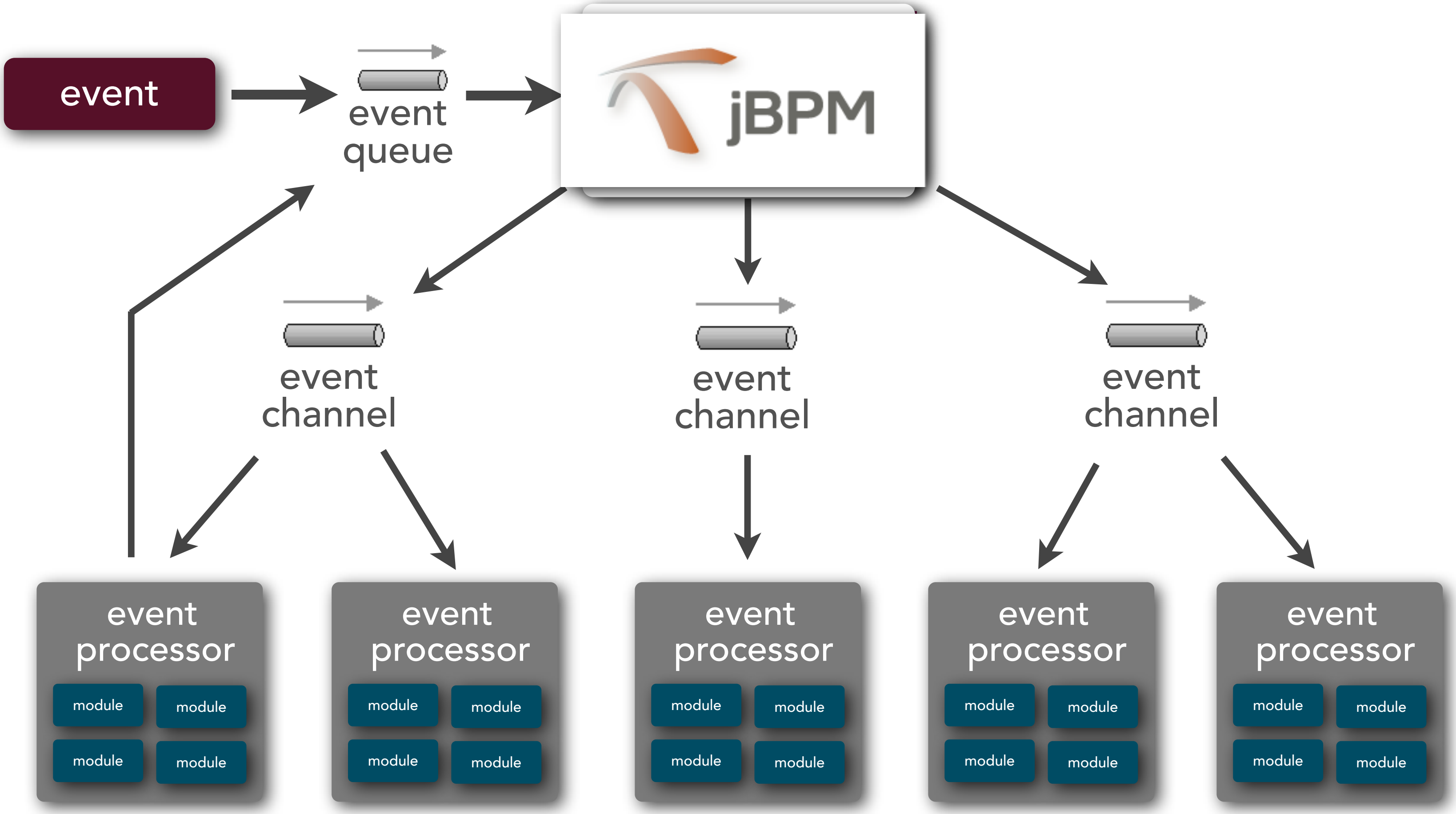
event-driven architecture

mediator topology

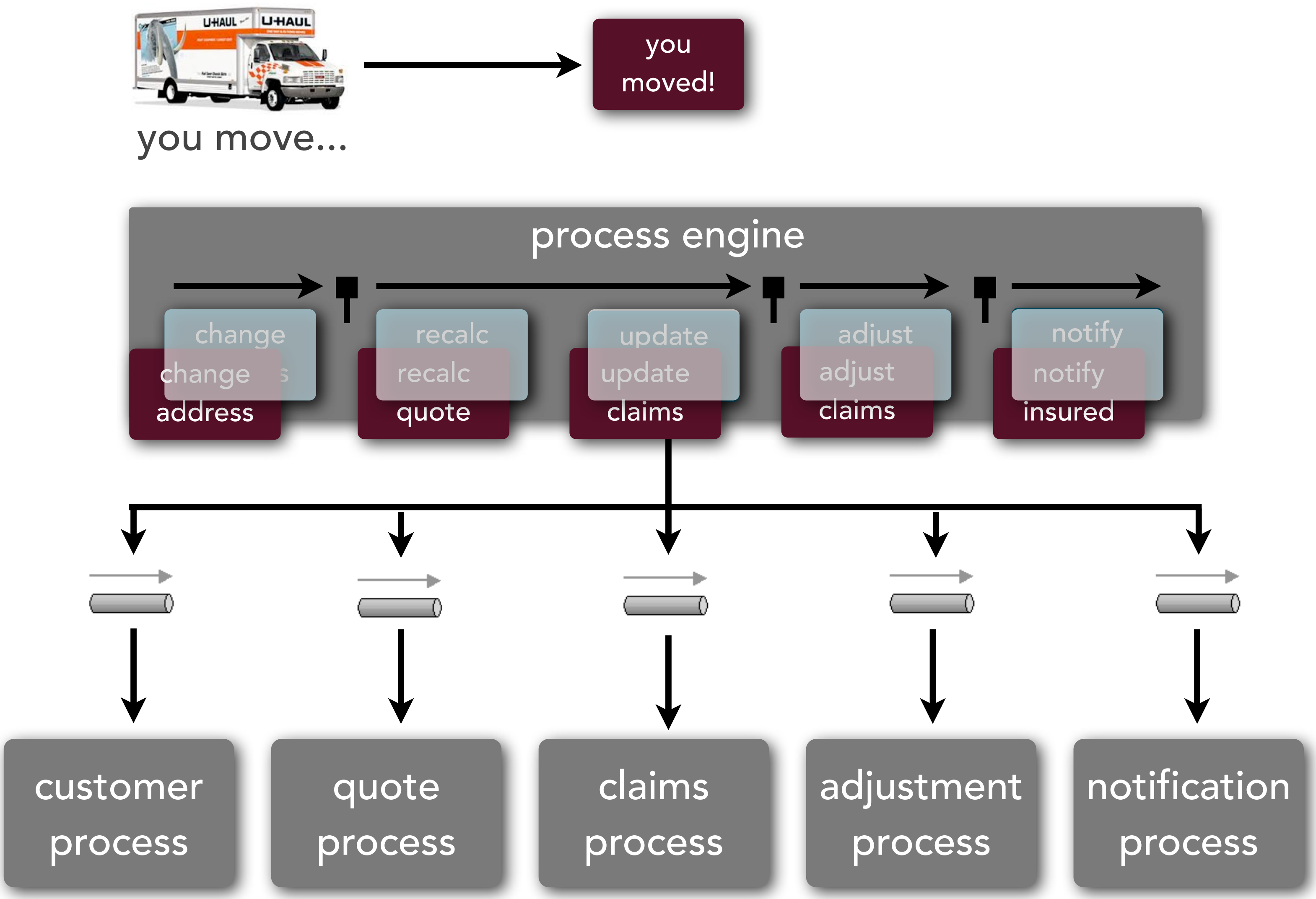


event-driven architecture

mediator topology

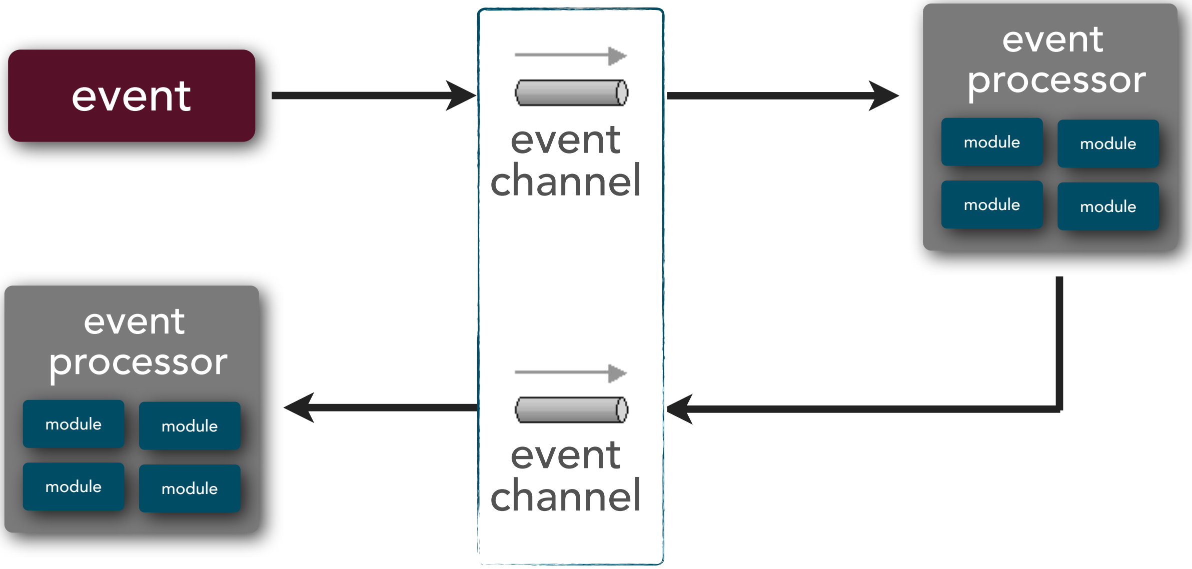


event-driven architecture



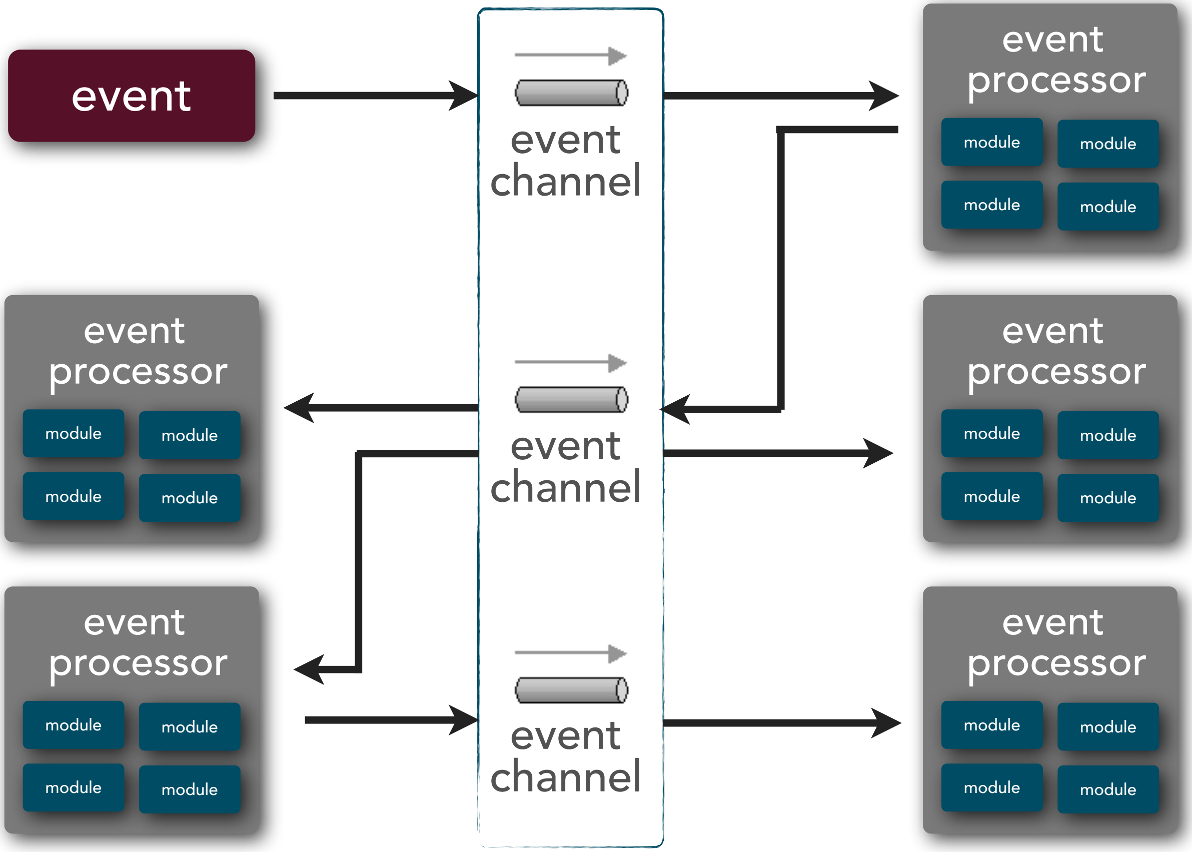
event-driven architecture

broker topology

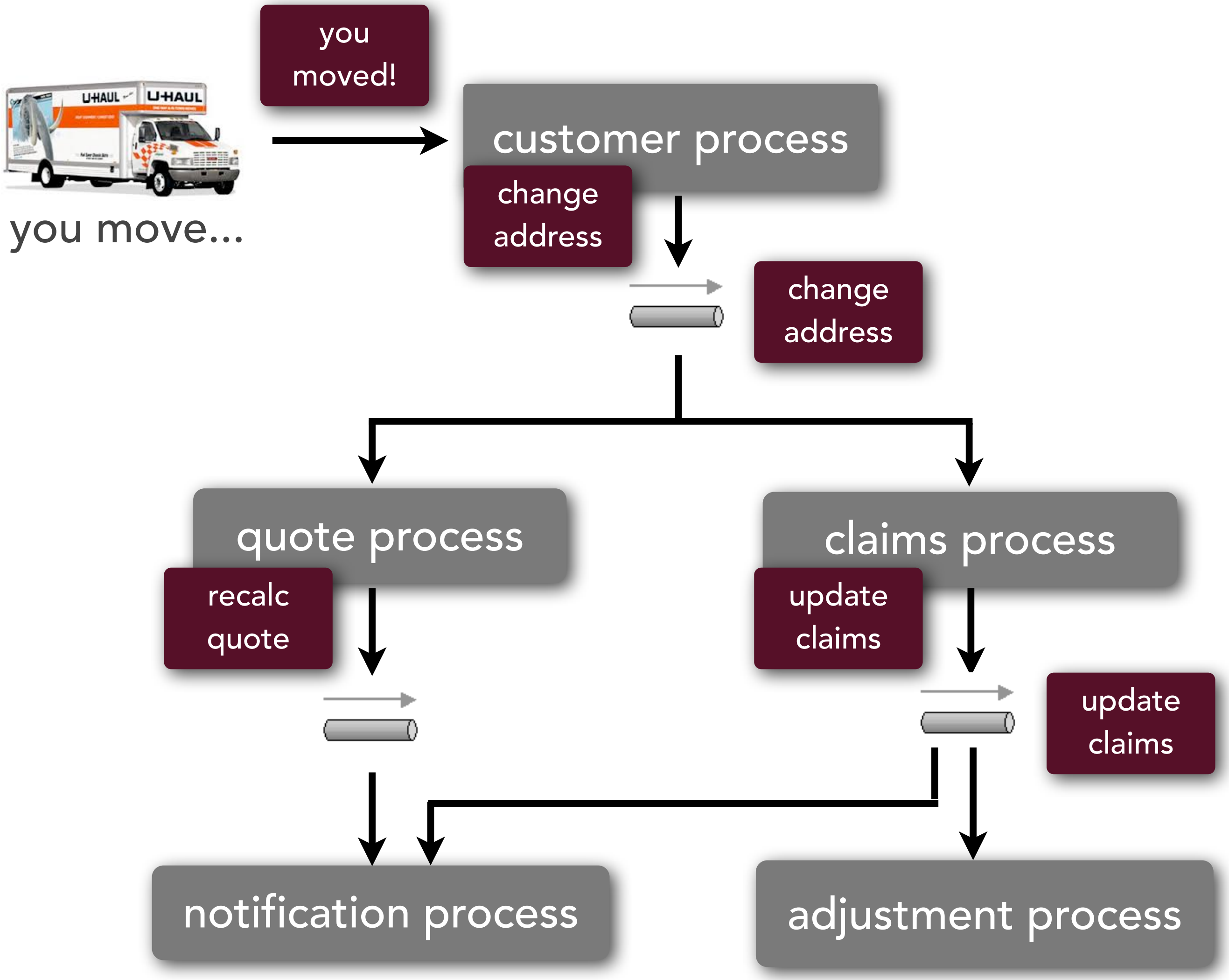


event-driven architecture

broker topology



event-driven architecture



event-driven architecture

considerations

contract creation, maintenance, and versioning can be difficult

must address remote process availability or unresponsiveness

Reconnecting

The connection has been lost. Attempting to reconnect to your session...

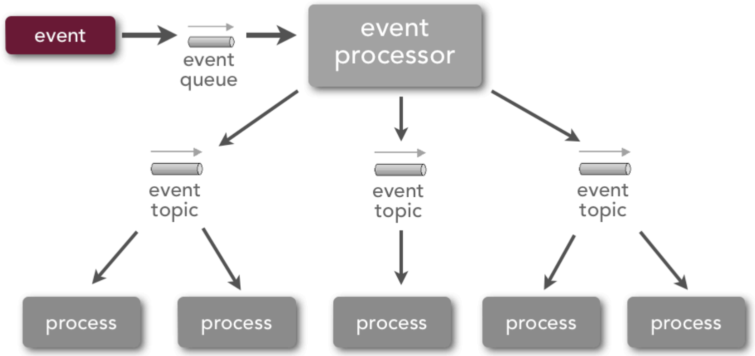
Connection attempt: 1 of 20

reconnection logic on server restart or failure must be addressed

event-driven architecture

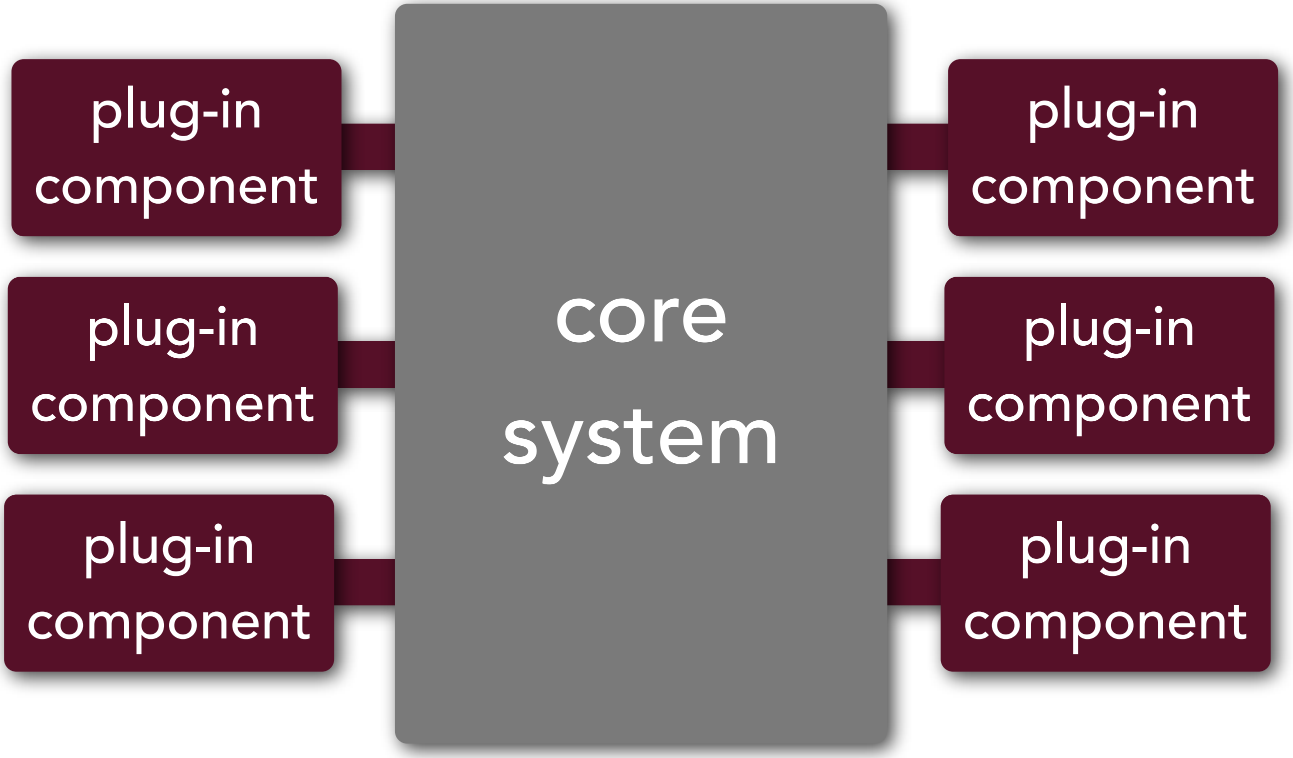
analysis

overall agility	👍
deployment	👍
testability	👎
performance	👍
scalability	👍
development	👎
complexity	👎
loose coupling	👍



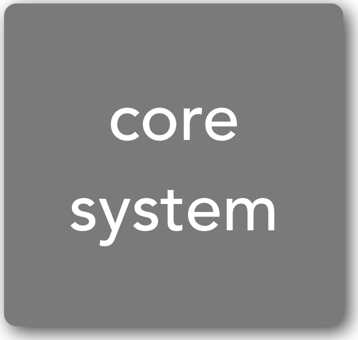
microkernel architecture

(a.k.a. plug-in architecture pattern)



microkernel architecture

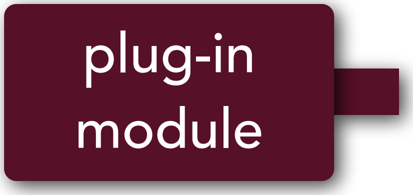
architectural components



minimal functionality to run system

general business rules and logic

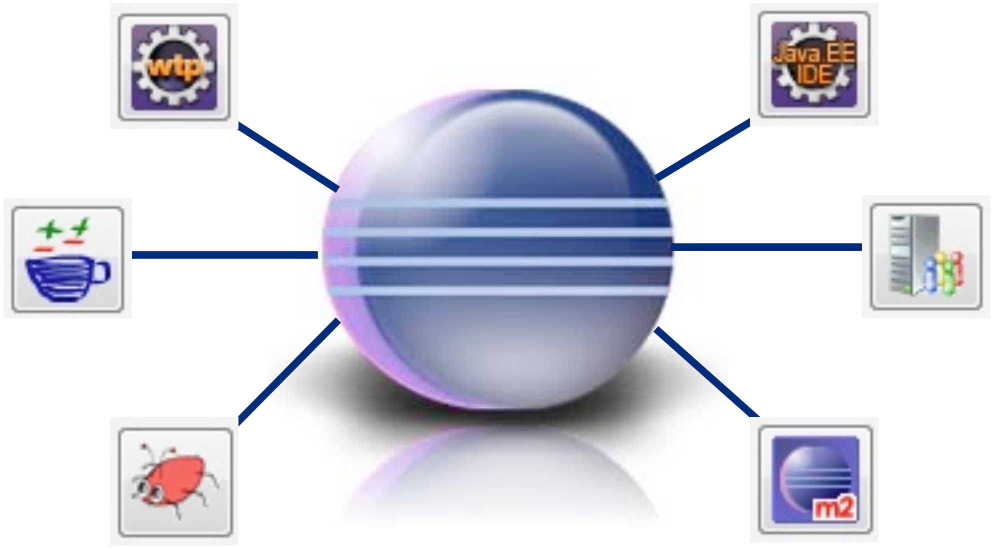
no custom processing



standalone independent module

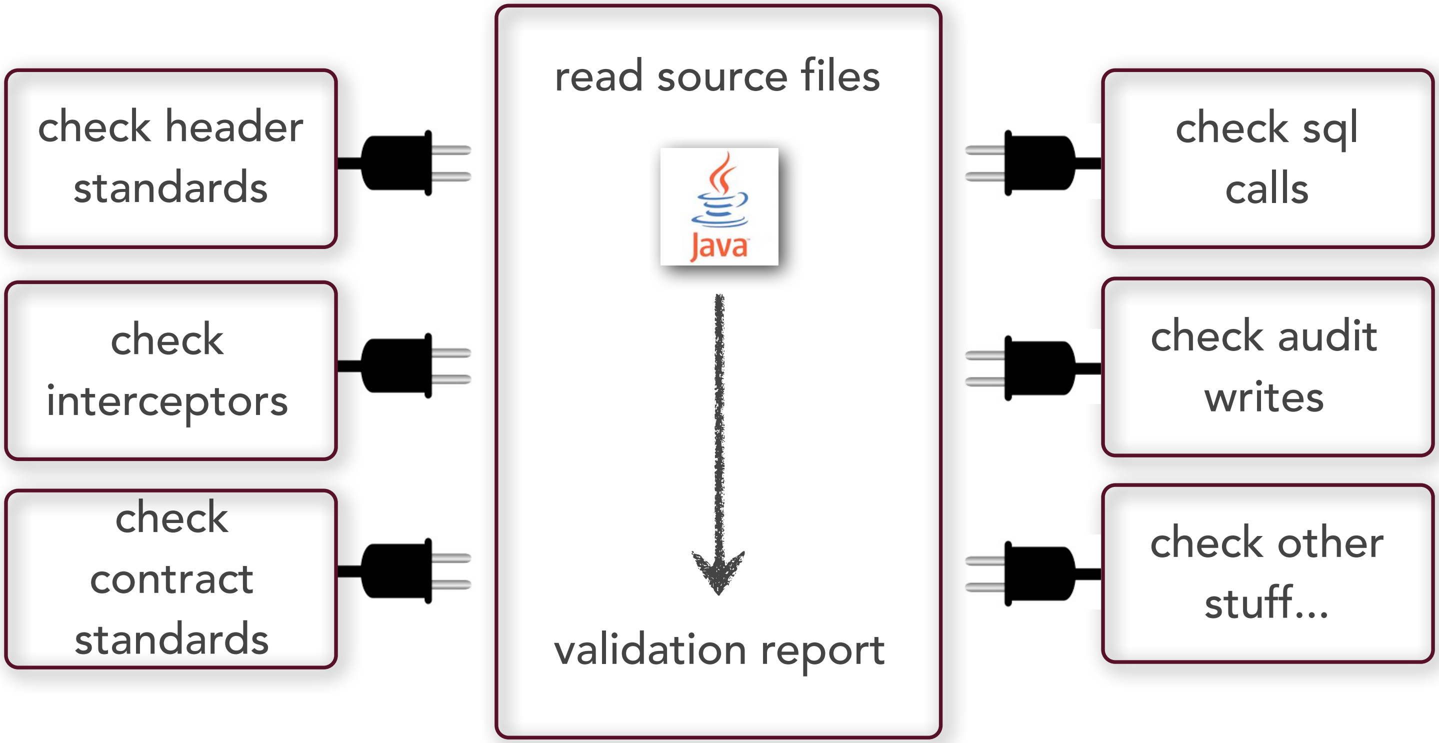
specific additional rules or logic

microkernel architecture



microkernel architecture

source validation tool



check header standards

check interceptors

check contract standards

read source files

validation report

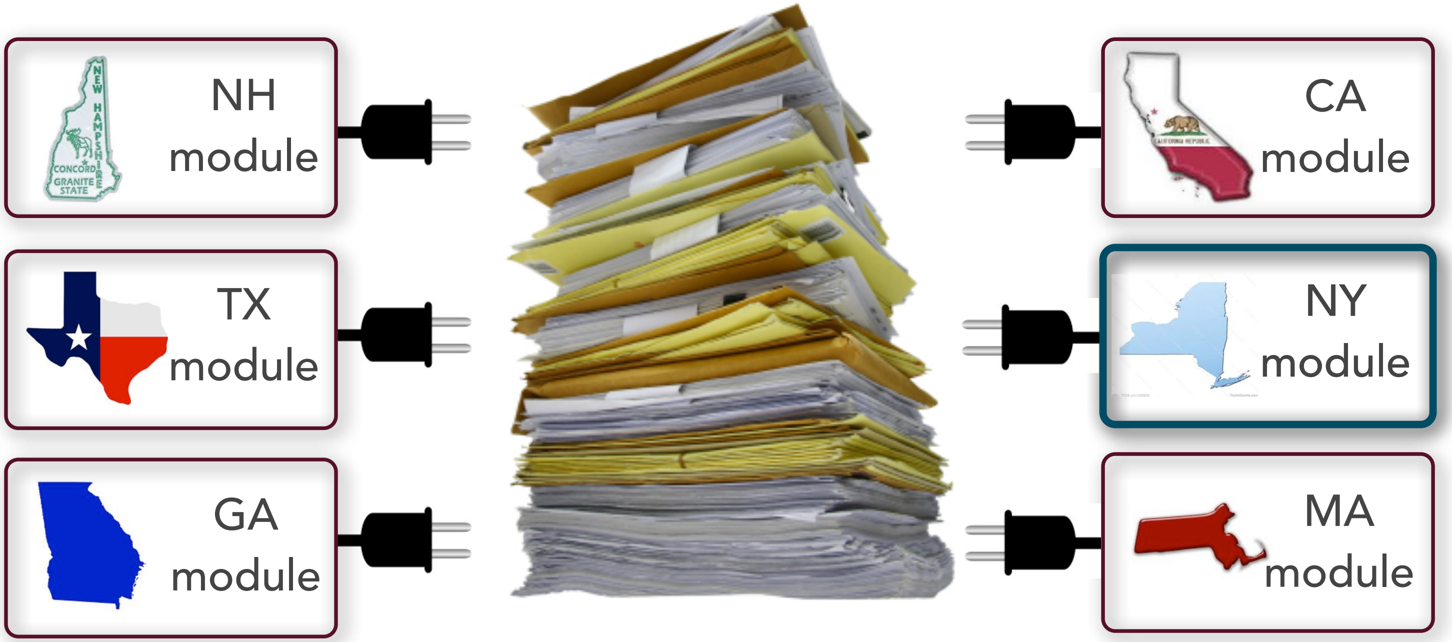
check sql calls

check audit writes

check other stuff...

microkernel architecture

claims processing



NH module

TX module

GA module

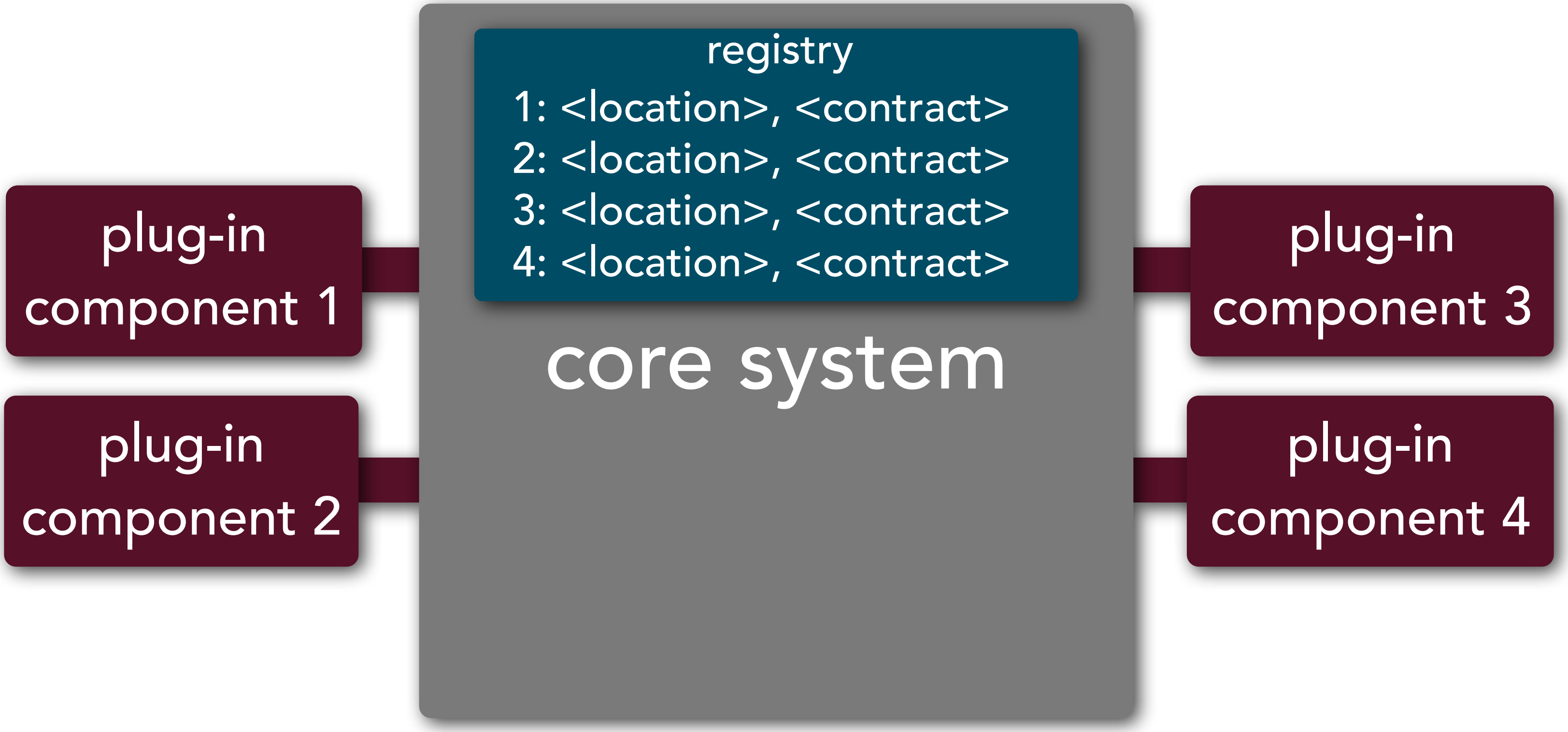
CA module

NY module

MA module

microkernel architecture

registry



registry
1: <location>, <contract>
2: <location>, <contract>
3: <location>, <contract>
4: <location>, <contract>
core system
plug-in component 1
plug-in component 2
plug-in component 3
plug-in component 4

microkernel architecture

registry

```
static {
   pluginRegistry.put(NAMING, "ValidatorNamingPlugin");
   pluginRegistry.put(SYSOUT, "ValidatorSysoutPlugin");
   pluginRegistry.put(AUDIT, "ValidatorAuditPlugin");
   pluginRegistry.put(TODO, "ValidatorTodoPlugin");
   pluginRegistry.put(COMMENTS, "ValidatorCommentsPlugin");
   pluginRegistry.put(SVC_CALLS, null);
}
```

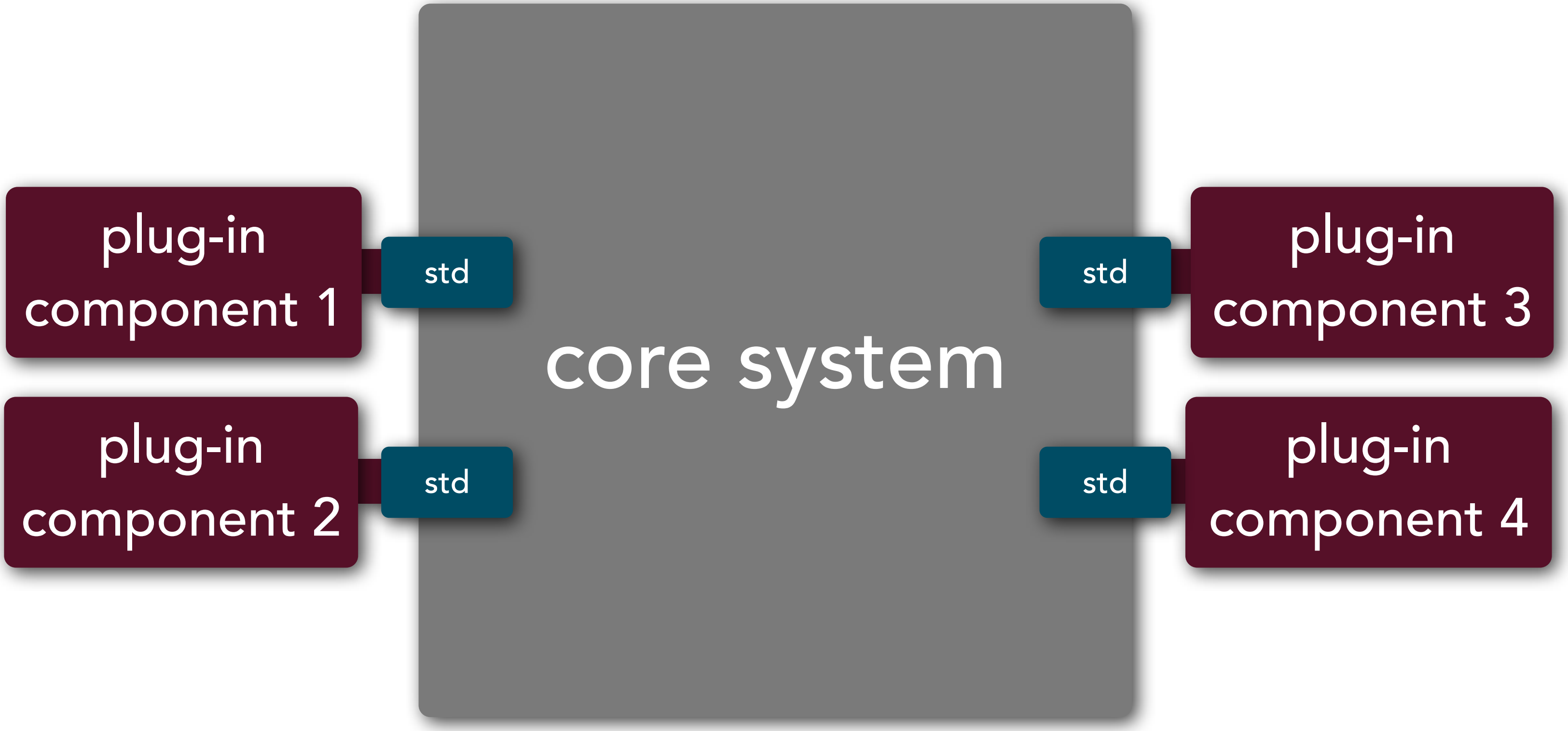
microkernel architecture

registry

```
private String executeChecks(String moduleName) throws Exception {
   for (Map.Entry<String, String> entry : pluginRegistry.entrySet()) {
      if (entry.getValue() != null) {
         Class<?> c = Class.forName(PLUGIN_PKG + entry.getValue());
         Constructor<?> con = c.getConstructor();
         ValidatorPlugin plugin = (ValidatorPlugin)con.newInstance();
         data = plugin.execute(data);
      }
   }
}
```

microkernel architecture

plug-in contracts



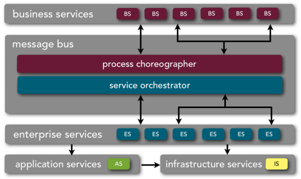
microkernel architecture

plug-in contracts

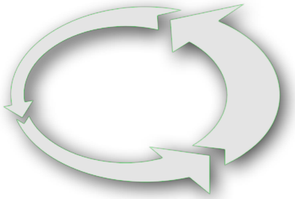
```
public class ValidatorData {
   public String moduleName;           //input
   public List<String> moduleContents; //input
   public String validationResults;    //output
}

public interface ValidatorPlugin {
   public ValidatorData execute(ValidatorData data);
}
```

microkernel architecture

considerations



can be embedded or used as part of another pattern



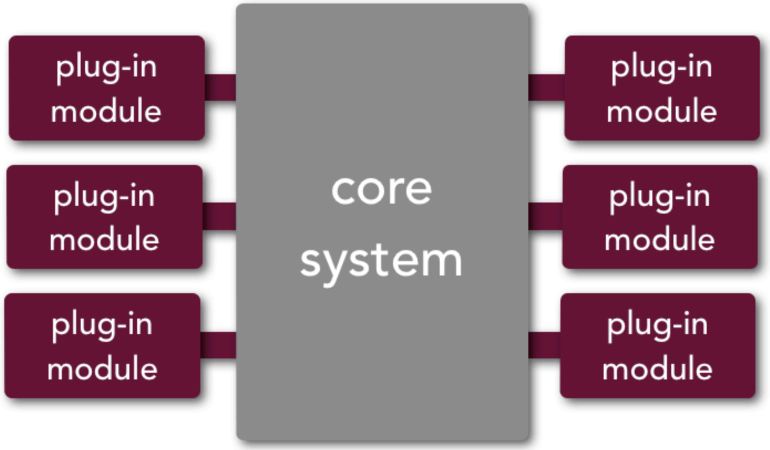
great support for evolutionary design and incremental development

great pattern for product-based applications

microkernel architecture

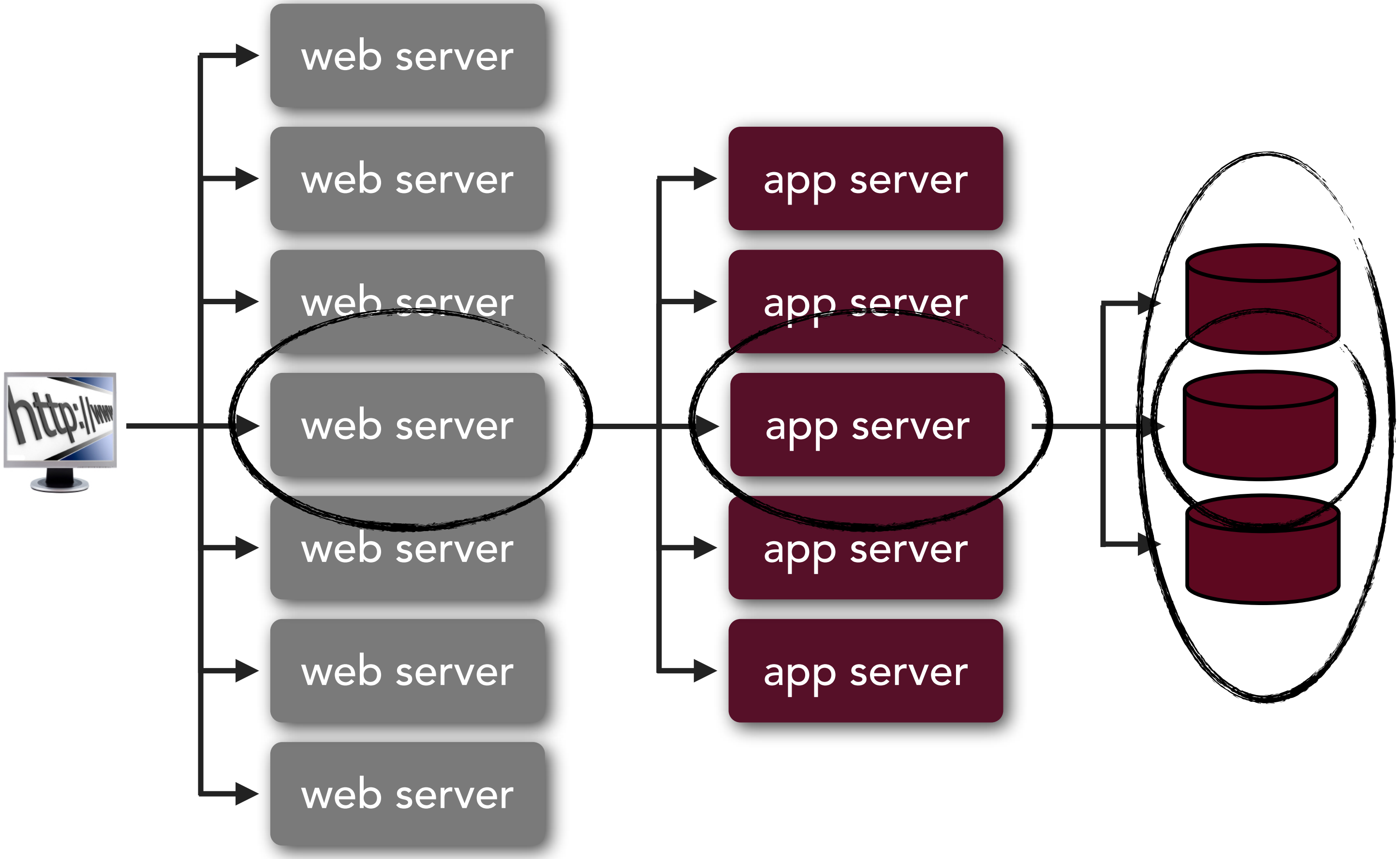
analysis

overall agility	👍
deployment	👍
testability	👍
performance	👍
scalability	👎
development	👍
complexity	👎
loose coupling	👍

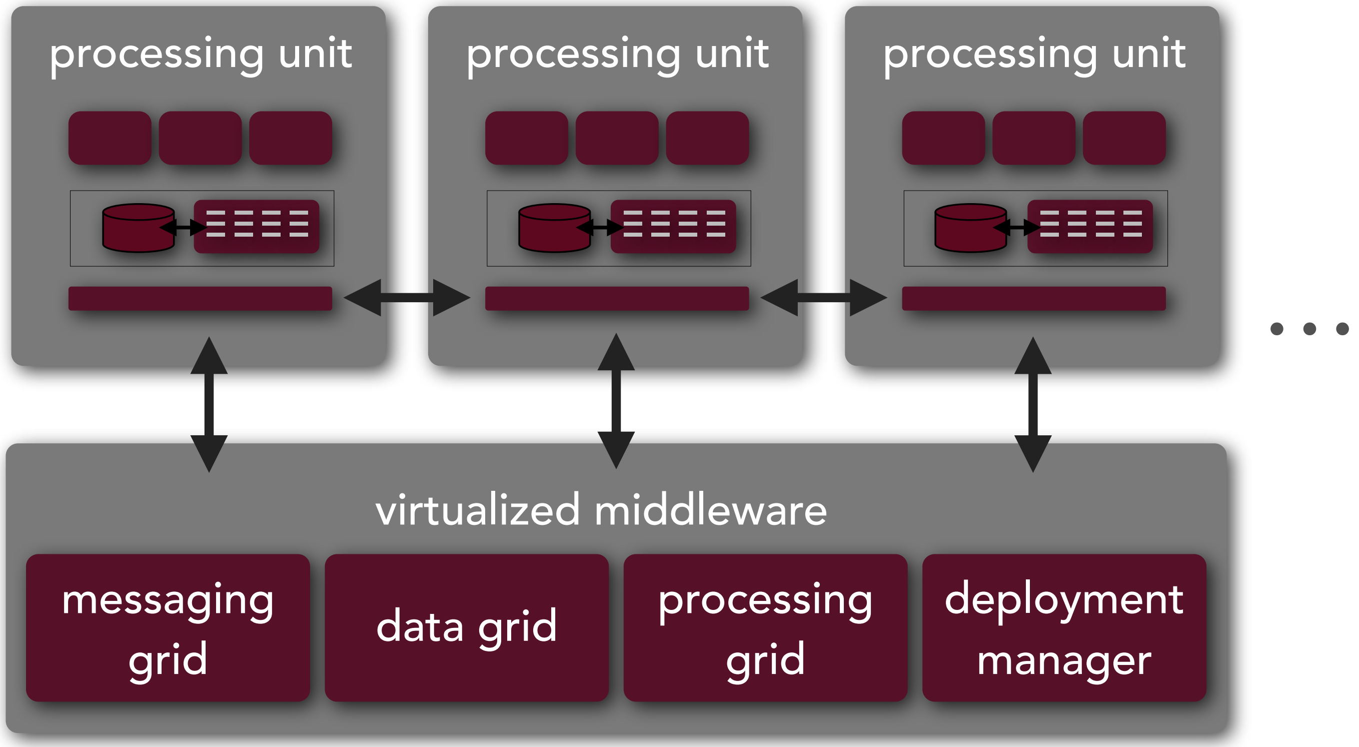


space-based architecture

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

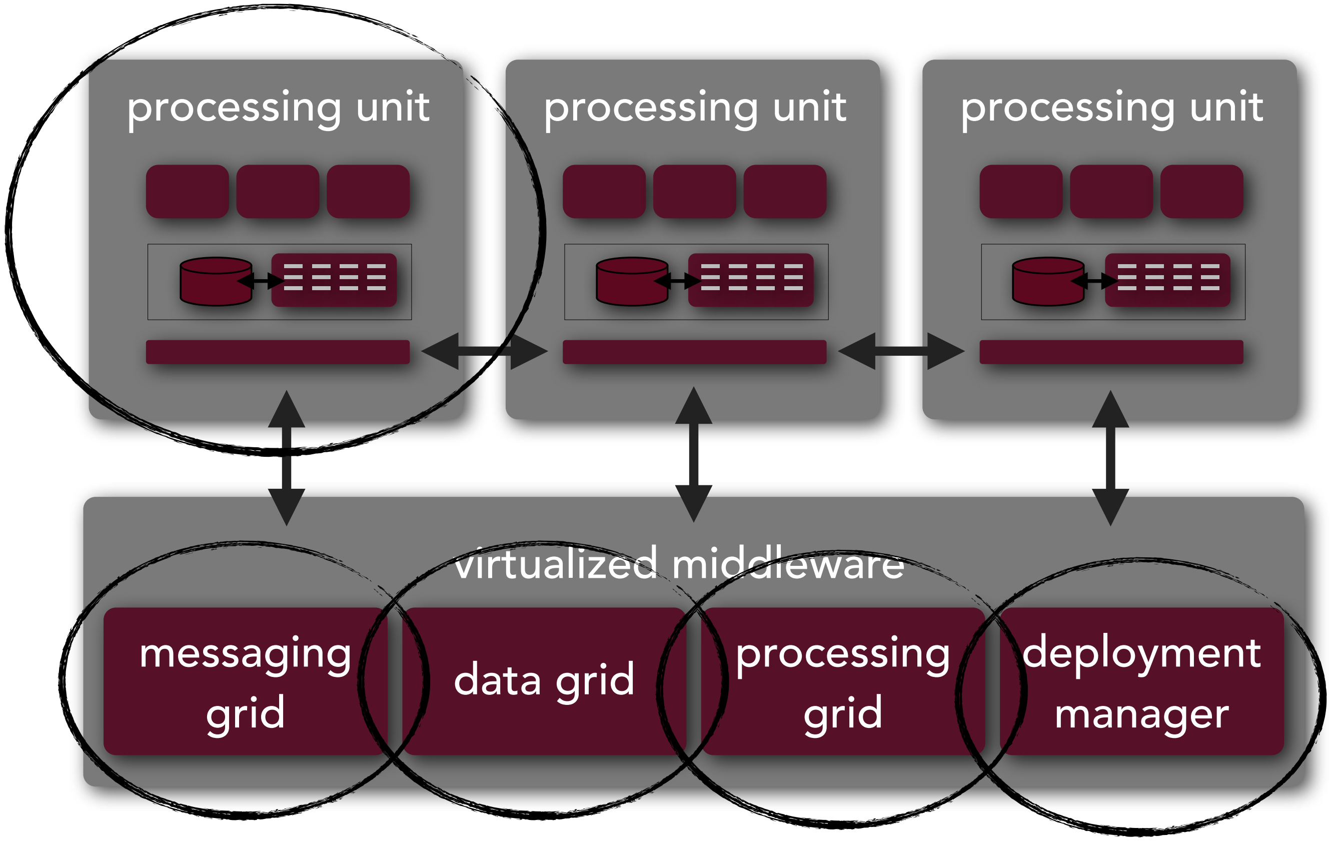


space-based architecture



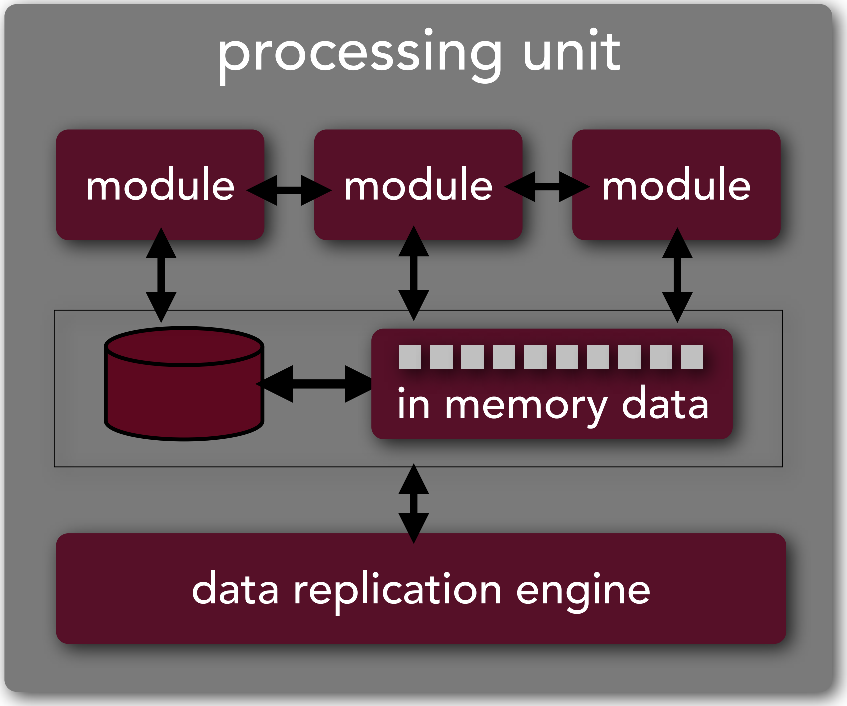
space-based architecture

architectural components



space-based architecture

processing unit



space-based architecture

middleware

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

space-based architecture

middleware

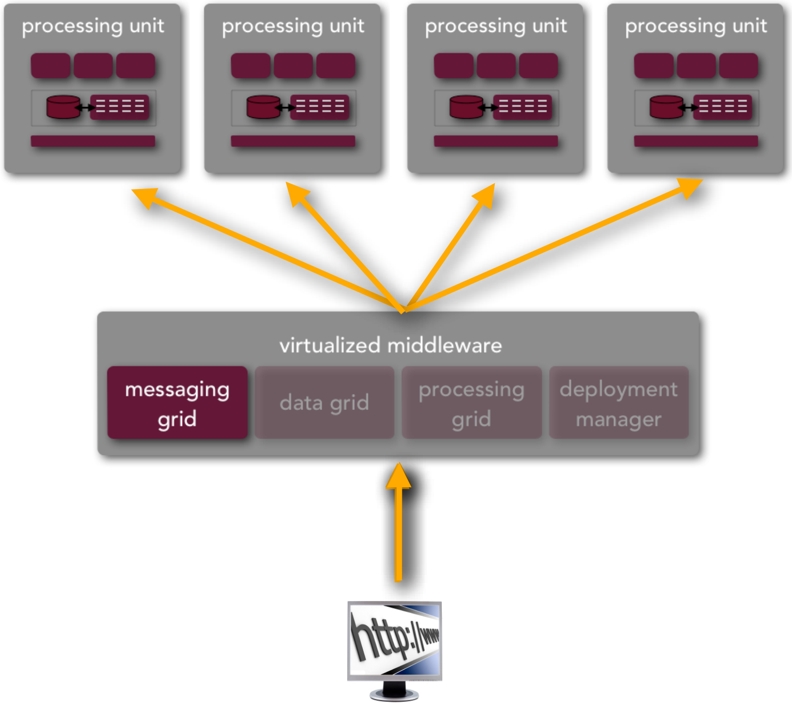
messaging grid

data grid

processing grid

deployment manager

manages input request and session



processing unit

processing unit

processing unit

processing unit

virtualized middleware

messaging grid

data grid

processing grid

deployment manager

space-based architecture

middleware

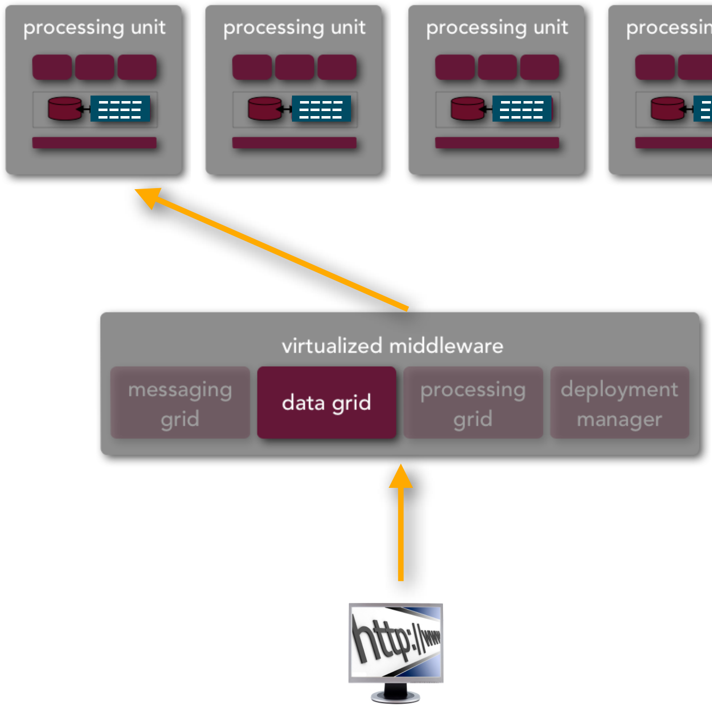
manages data replication between processing units

messaging grid

data grid

processing grid

deployment manager



space-based architecture

middleware

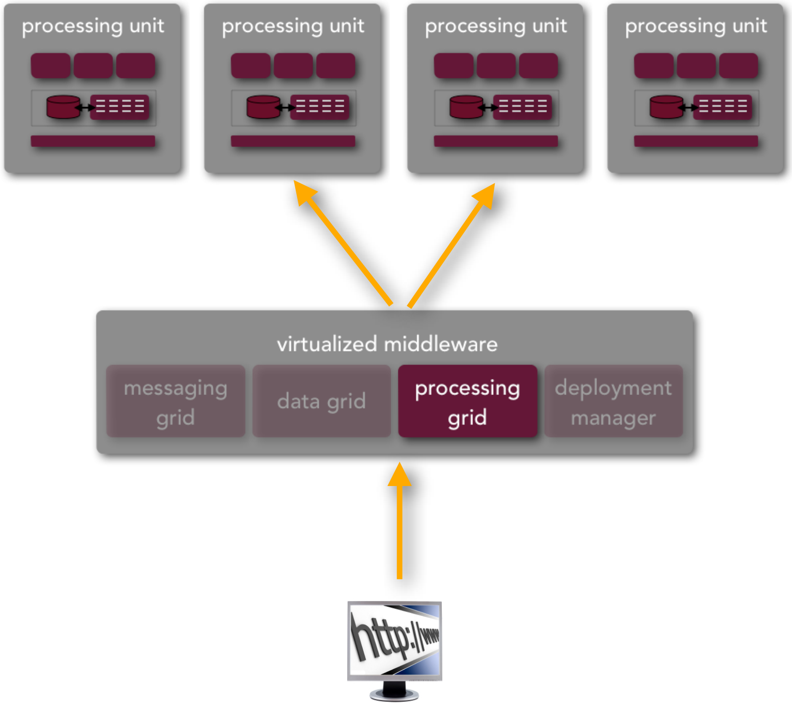
messaging grid

data grid

processing grid

deployment manager

manages distributed request processing



space-based architecture

middleware

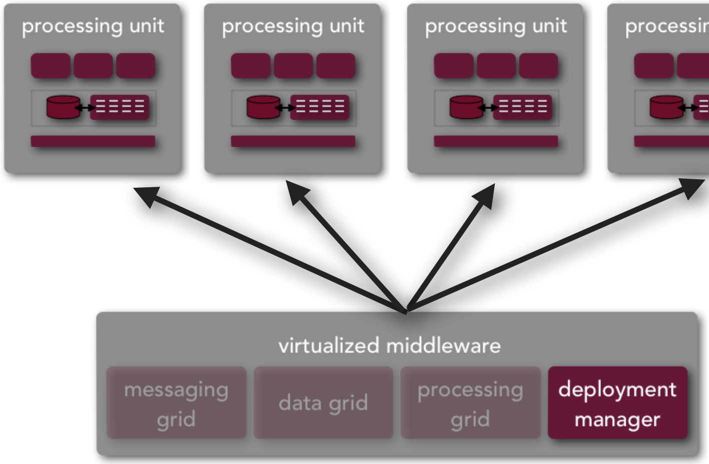
messaging grid

data grid

processing grid

deployment manager

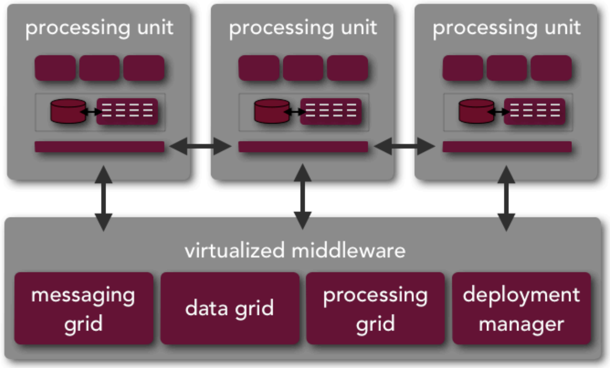
manages dynamic processing unit deployment



space-based architecture

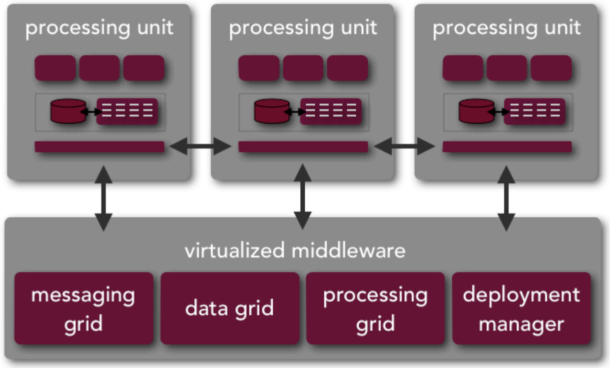
product implementations

- javaspaces
- gigaspaces
- ibm object grid
- gemfire
- ncache
- oracle coherence



space-based architecture

it's all about variable scalability...



good for applications that have variable load or inconsistent peak times

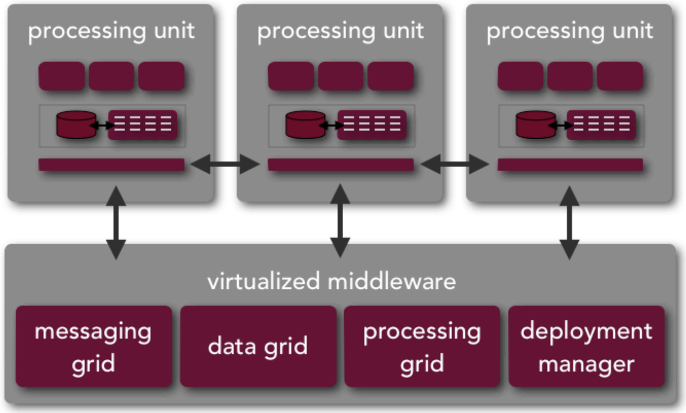
not a good fit for traditional large-scale relational database systems

relatively complex and expensive pattern to implement

space-based architecture

analysis

overall agility	👍
deployment	👍
testability	👎
performance	👎
scalability	👍
development	👍
complexity	👎



Software Architecture Patterns

Mark Richards

Independent Consultant
Hands-on Software Architect
Published Author / Conference Speaker

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