Reactive Architecture Patterns

Mark Richards
Independent Consultant
Hands-on Software Architect
Published Author / Conference Speaker
www.wmrichards.com

Author of *Software Architecture Fundamentals Video Series* (O'Reilly)
Author of Microservices vs. Service-Oriented Architecture (O'Reilly)
Author of *Enterprise Messaging Video Series* (O'Reilly)
Author of *Java Message Service 2nd Edition* (O'Reilly)

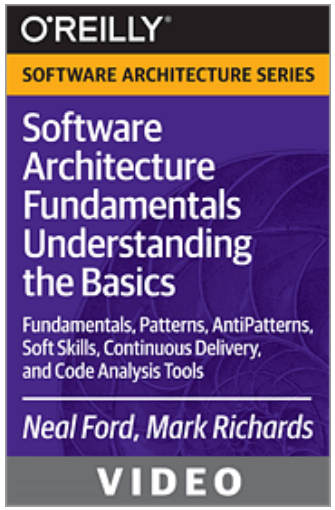
Software Architecture Fundamentals Training

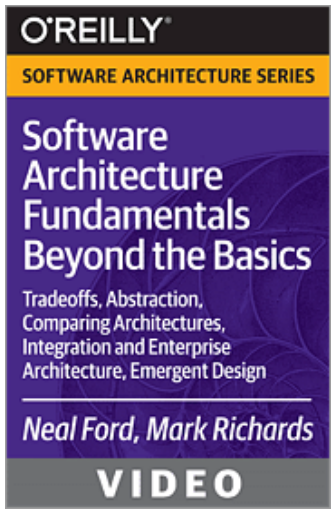
https://nofluffjuststuff.com//n/training/schedule

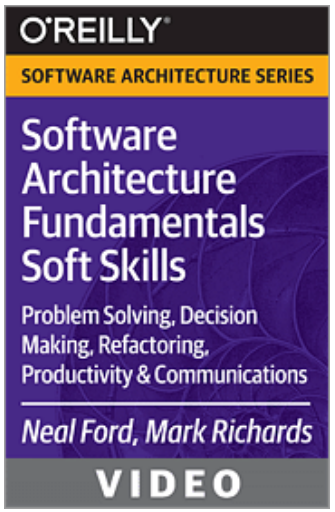
Training Event Schedule

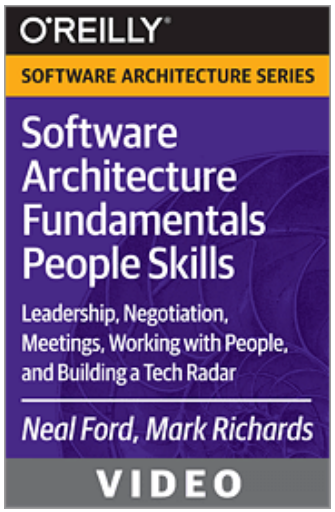
	Architecture Training with Mark Richards	Minneapolis, MN	June 13 - 15, 2016
	Architecture Training with Mark Richards	Columbus, OH	September 19 - 21, 2016
	Architecture Training with Mark Richards	Dallas, TX	October 24 - 26, 2016
	Architecture Training with Mark Richards	Boston, MA	November 14 - 16, 2016

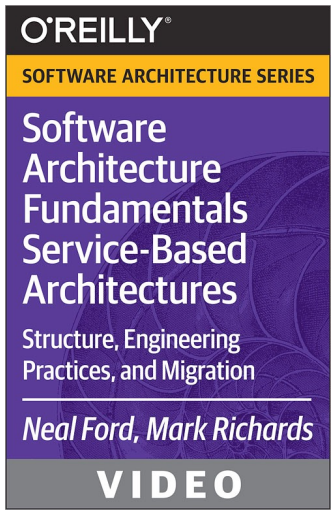
Software Architecture Fundamentals Video Series
Enterprise Messaging Video Series

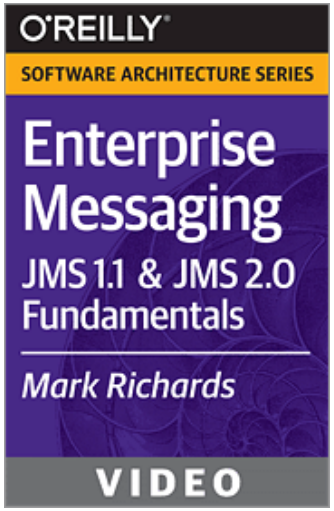


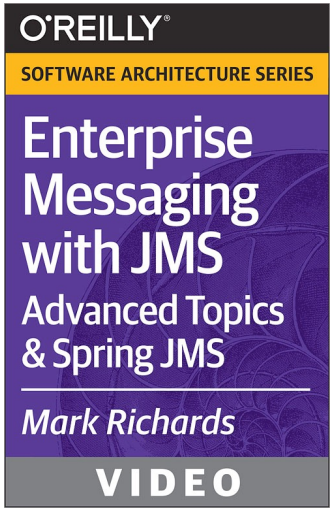












agenda

reactive architecture overview

messaging standards

channel monitor pattern

consumer supervisor pattern

producer control flow pattern

thread delegate pattern

source code

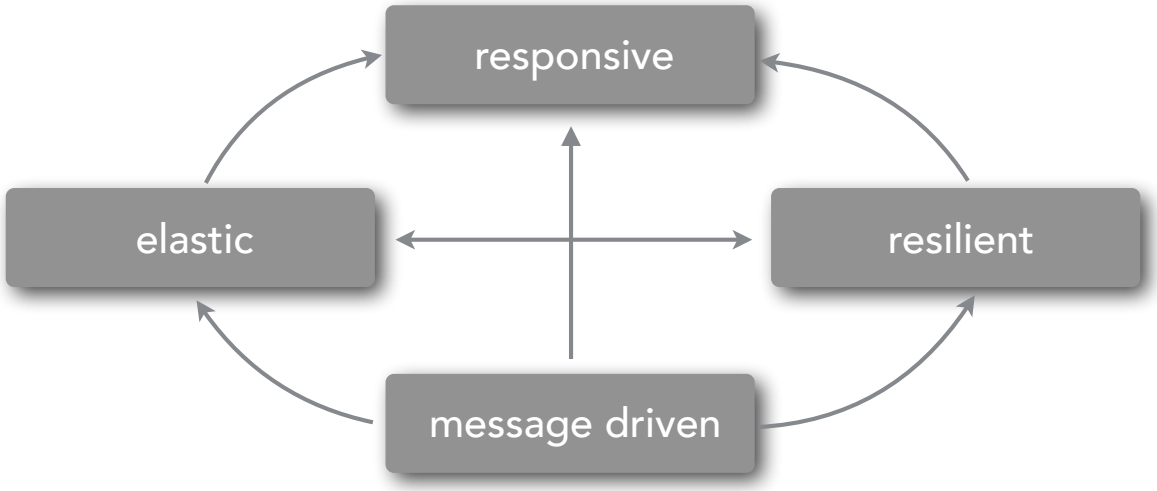
https://github.com/wmr513/reactive

Reactive Architecture Overview

reactive architecture

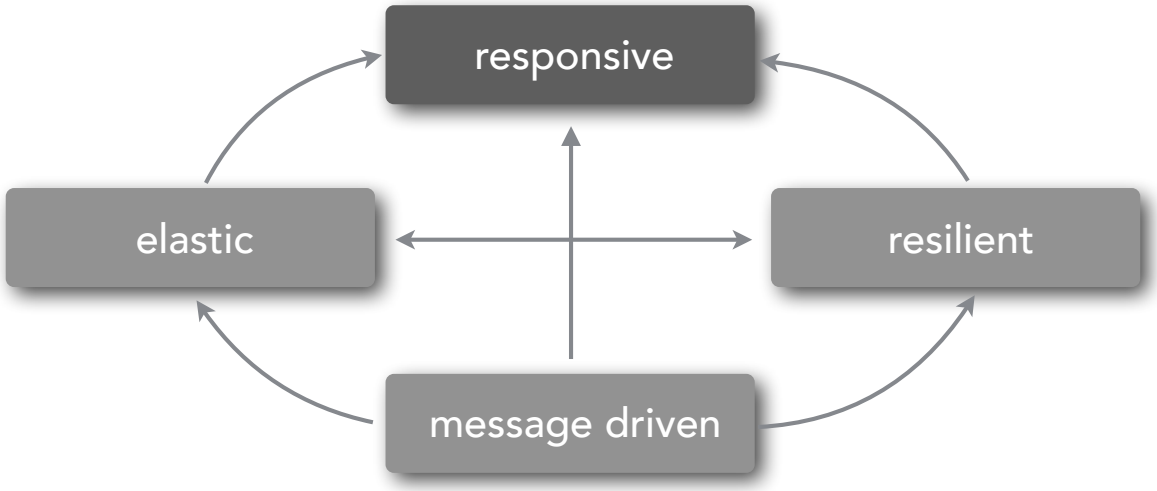
reactive architecture

reactive manifesto



reactive architecture

reactive manifesto

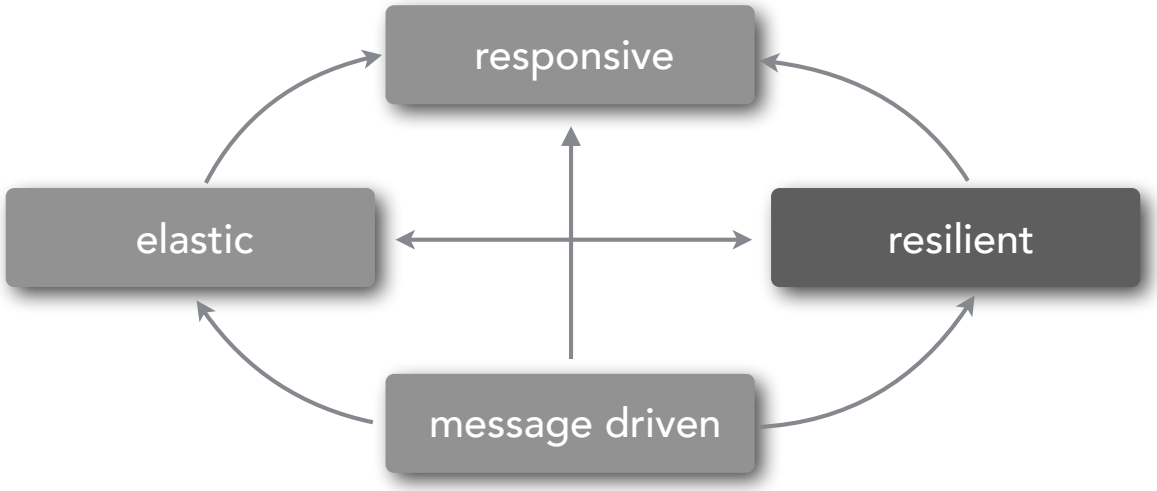


the system responds in a consistent, rapid, and timely manner whenever possible

how the system reacts to users

reactive architecture

reactive manifesto

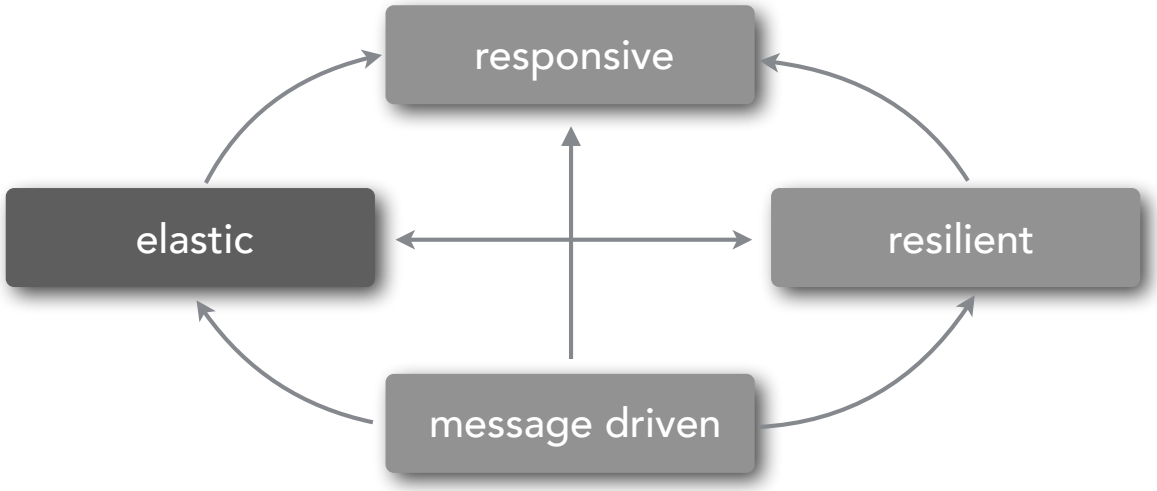


the system stays responsive after a failure through replication, containment, isolation, and delegation

how the system reacts to failures

reactive architecture

reactive manifesto

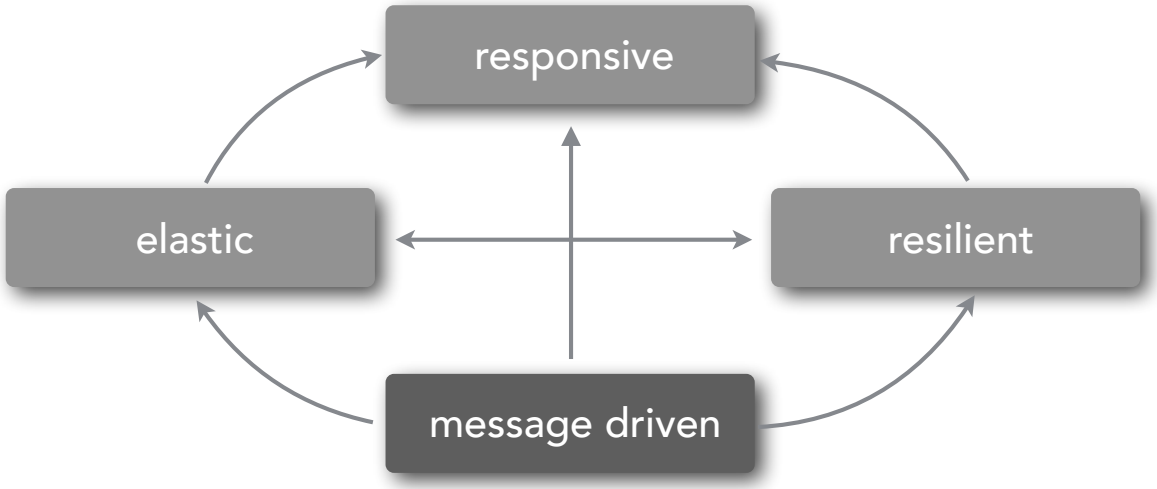


the system stays responsive under varying workload

how the system reacts to load

reactive architecture

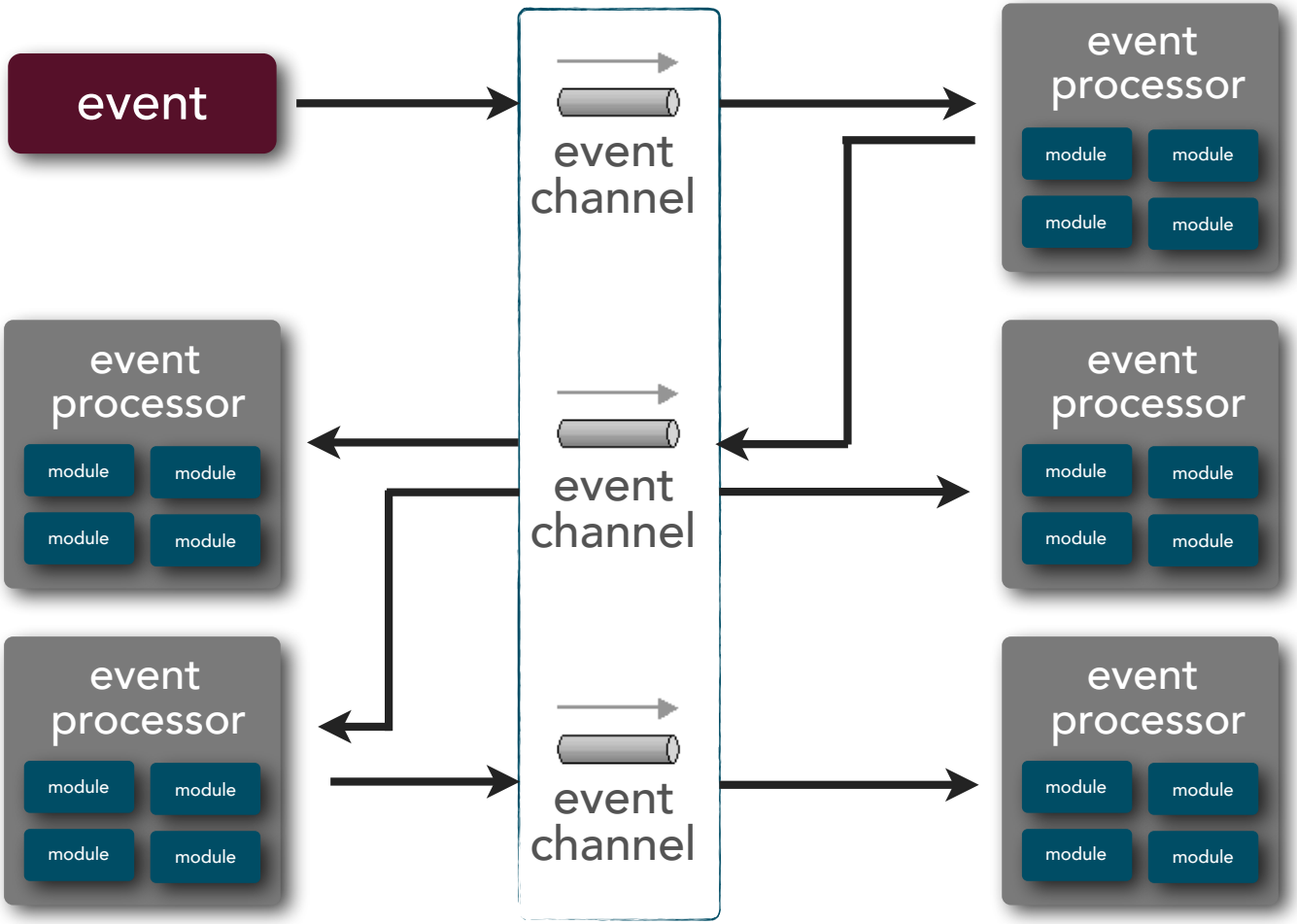
reactive manifesto



the system relies on asynchronous messaging to ensure loose coupling, isolation, location transparency, and error delegation

how the system reacts to events

event-driven architecture



reactive architecture

reactive architecture

vs.

reactive programming

Messaging Standards

messaging standards

java message service

advanced message queuing protocol

messaging standards

java message service

java specification and api that provides a standard way for java programs to send and receive messages within the java platform.

messaging standards

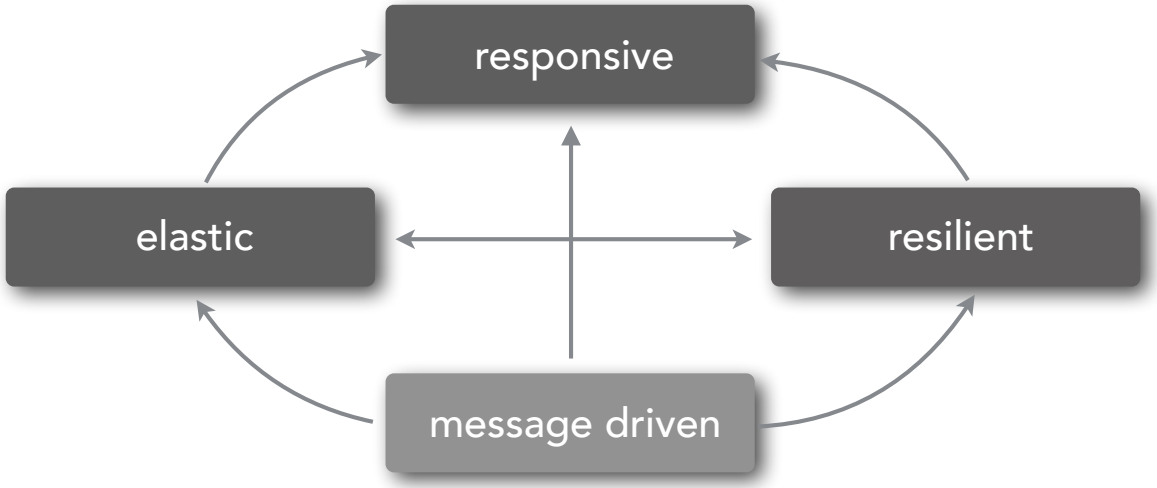
advanced message queuing protocol

open specification that defines an industry standard wire-level messaging protocol used to send and receive messages across all platforms.

Channel Monitor Pattern

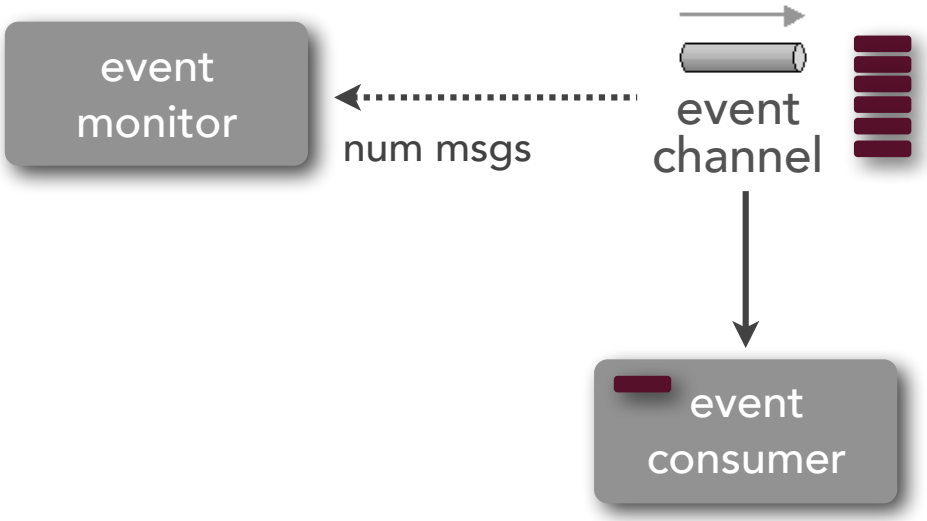
channel monitor pattern

how can you determine the current load on an event channel without consuming events?

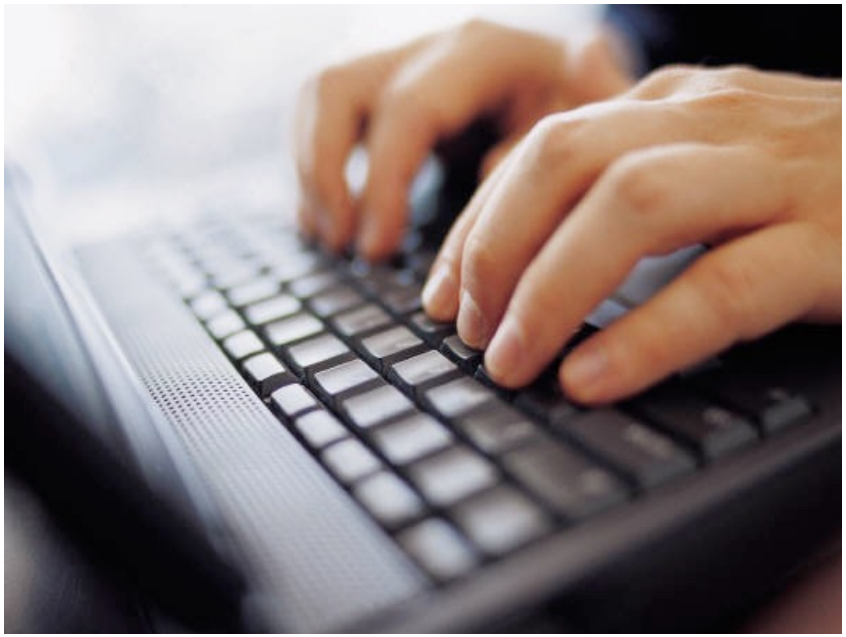


channel monitor pattern

how can you determine the current load on an event channel without consuming events?



channel monitor pattern



let’s see the basic setup and issue…

channel monitor pattern

Monitor.java

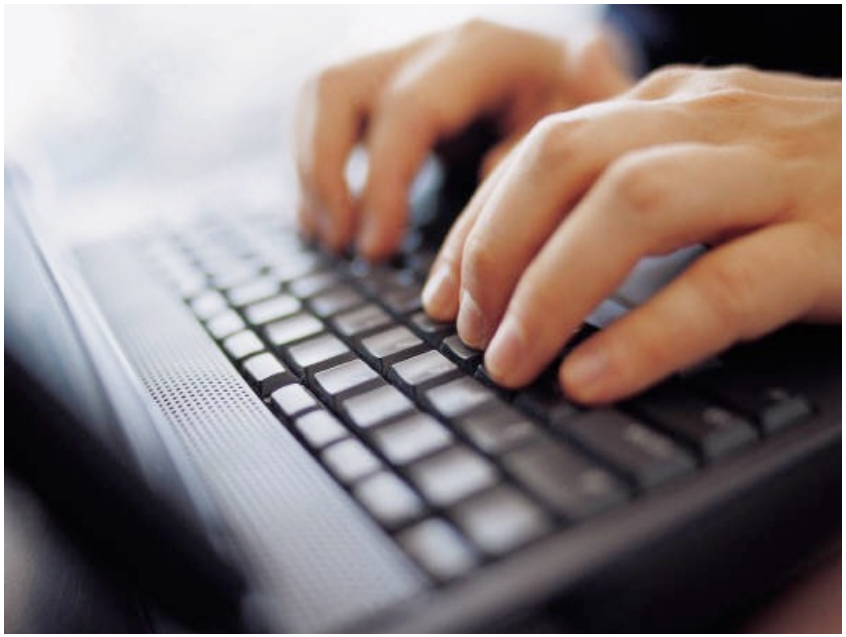
```
Channel channel = AMQPCommon.connect();
long consumers = channel.consumerCount("trade.eq.q");
long queueDepth = channel.messageCount("trade.eq.q");


DeclareOk queue = channel.queueDeclare("trade.eq.q",...);
long consumers = queue.getConsumerCount();
long queueDepth = queue.getMessageCount();
```

channel monitor pattern

Consumer.java

```
Channel channel = AMQPCommon.connect();
channel.basicQos(1);
channel.basicConsume("trade.eq.q", false, consumer);
QueueingConsumer.Delivery msg = consumer.nextDelivery();
channel.basicAck(msg.getEnvelope().getDeliveryTag(), false);
```

channel monitor pattern

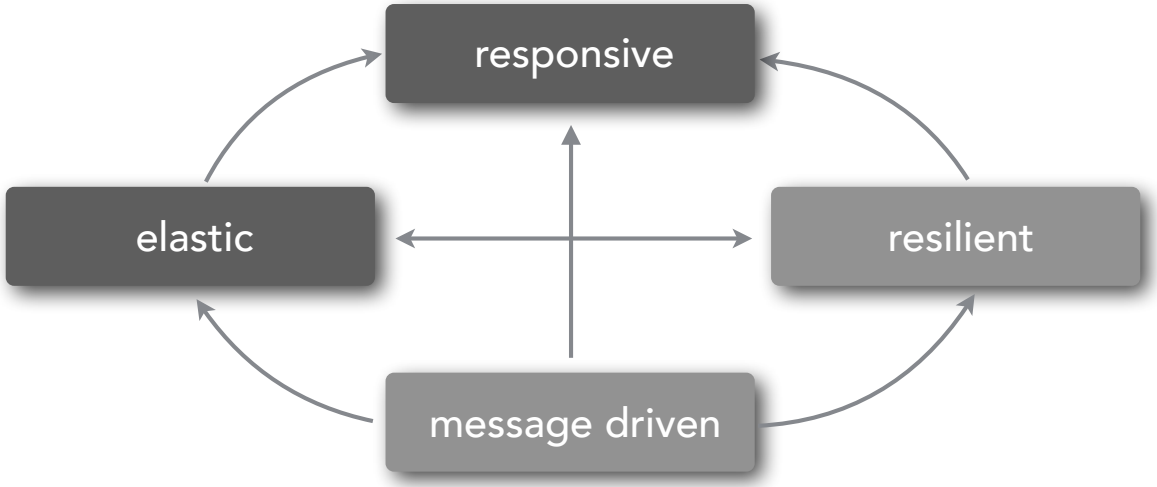


let’s see the result…

Consumer Supervisor Pattern

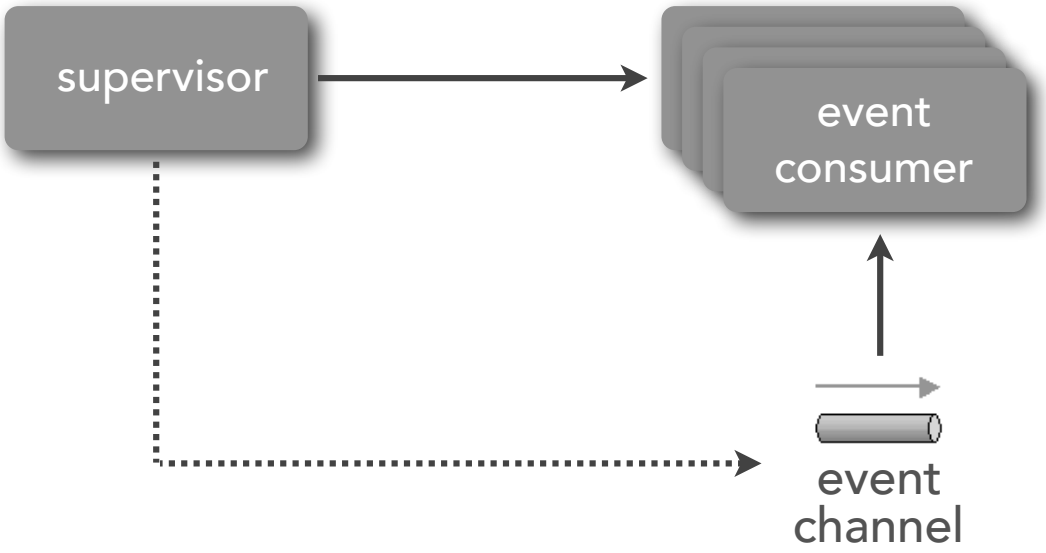
consumer supervisor pattern

how can you react to varying changes in load to event consumers to ensure consistent response time?

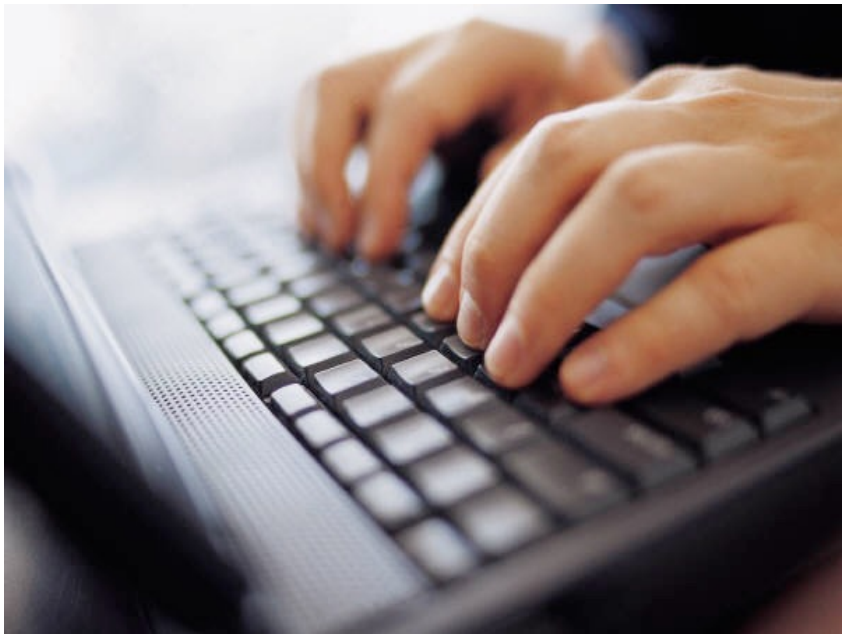


consumer supervisor pattern

how can you react to varying changes in load to event consumers to ensure consistent response time?



consumer supervisor pattern



let’s see the issue….

consumer supervisor pattern

Supervisor.java

```
private List<AMQPConsumer> consumers =
    new ArrayList<AMQPConsumer>();
Connection connection;

private void startConsumer() {
    AMQPConsumer consumer = new AMQPConsumer();
    consumers.add(consumer);
    new Thread() {
        public void run() {
          consumer.startup(connection);
    }}.start();
}
```

consumer supervisor pattern

Supervisor.java

```
private void stopConsumer() {
  if (consumers.size() > 1) {
    AMQPConsumer consumer = consumers.get(0);
    consumer.shutdown();
    consumers.remove(consumer);
   }
 }
```

consumer supervisor pattern

Supervisor.java

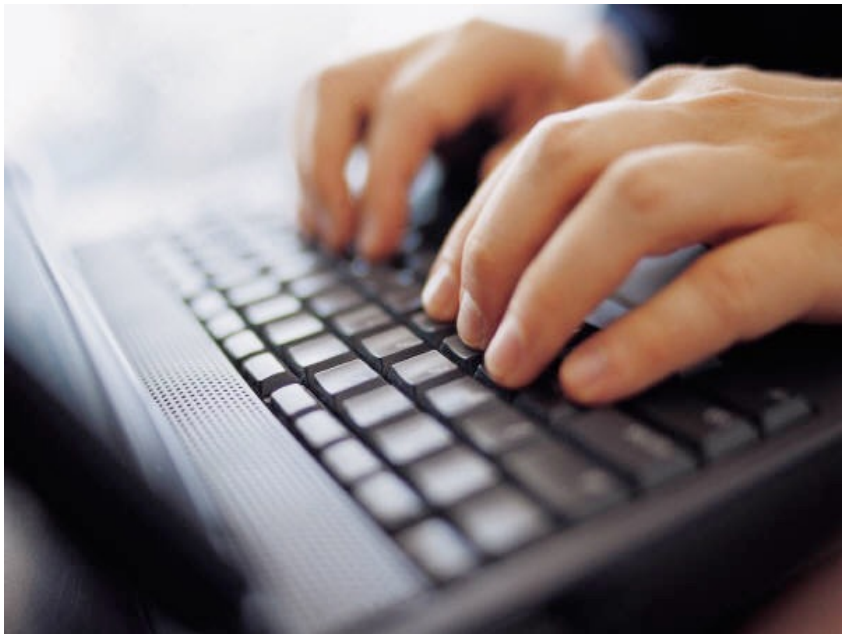
```
public void execute() throws Exception {
   Channel channel = AMQPCommon.connect();
   connection = channel.getConnection();
   startConsumer();
   while (true) {
      long queueDepth = channel.messageCount("trade.eq.q");
      long consumersNeeded = queueDepth/2;
      long diff = Math.abs(consumersNeeded - consumers.size());
      for (int i=0;i<diff;i++) {
         if (consumersNeeded > consumers.size())
            startConsumer();
         else
            stopConsumer();
      }
      Thread.sleep(1000);
   }
 }
```

consumer supervisor pattern

Consumer.java

```
private Boolean active = true;
public void startup(Connection connection) {
   Channel channel = connection.createChannel();
   QueueingConsumer consumer = new QueueingConsumer(channel);
   ...
   while (active) {
      QueueingConsumer.Delivery msg = consumer.nextDelivery();
      ...
   }
   channel.close();
}

public void shutdown() {
   synchronized(active) {
      active = false;
   }
}
```

consumer supervisor pattern

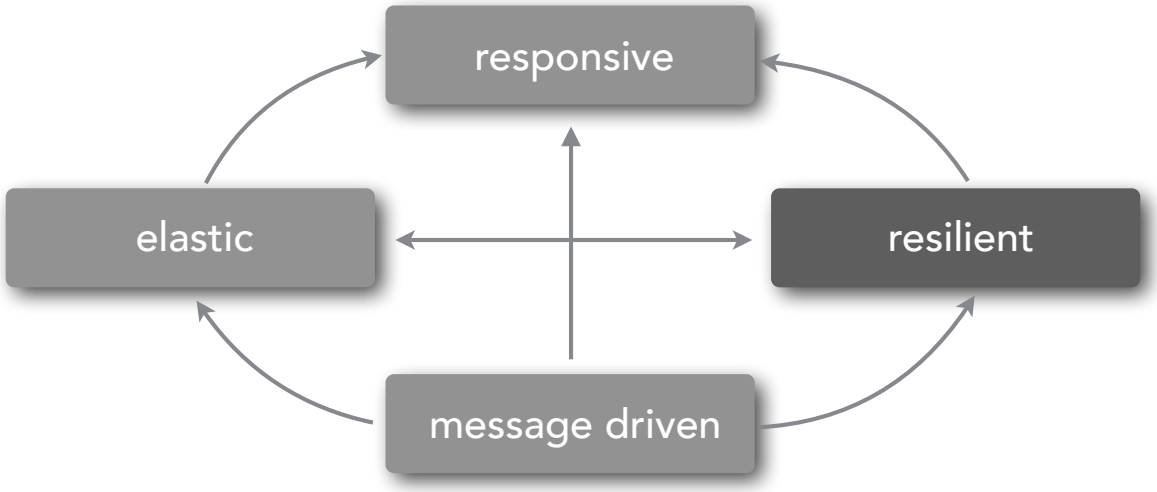


let’s see the result…

Producer Control Flow Pattern

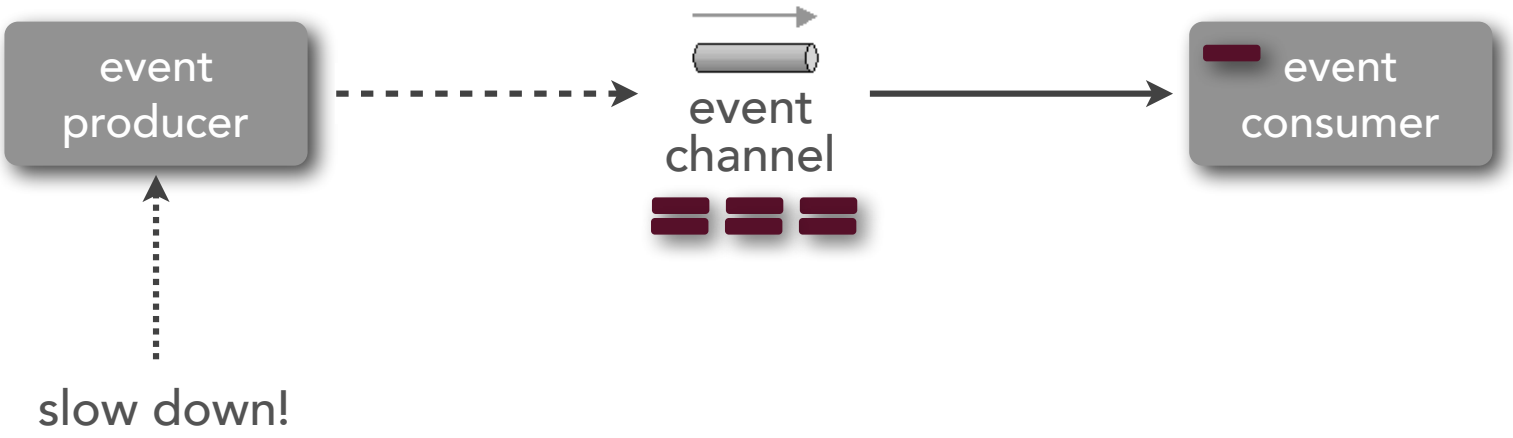
producer control flow pattern

how can you slow down message producers when the messaging system becomes overwhelmed?



producer control flow pattern

how can you slow down message producers when the messaging system becomes overwhelmed?

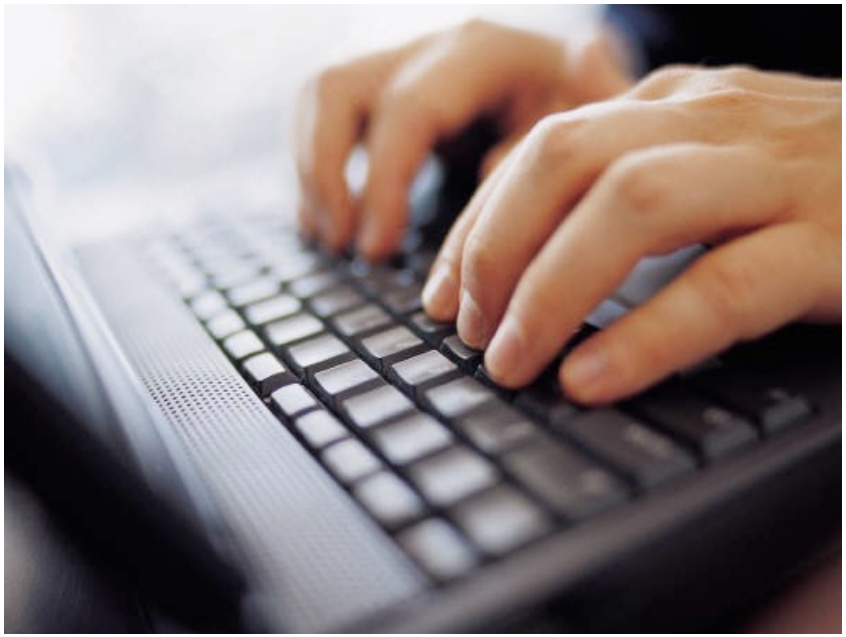


producer control flow pattern

how can you slow down message producers when the messaging system becomes overwhelmed?

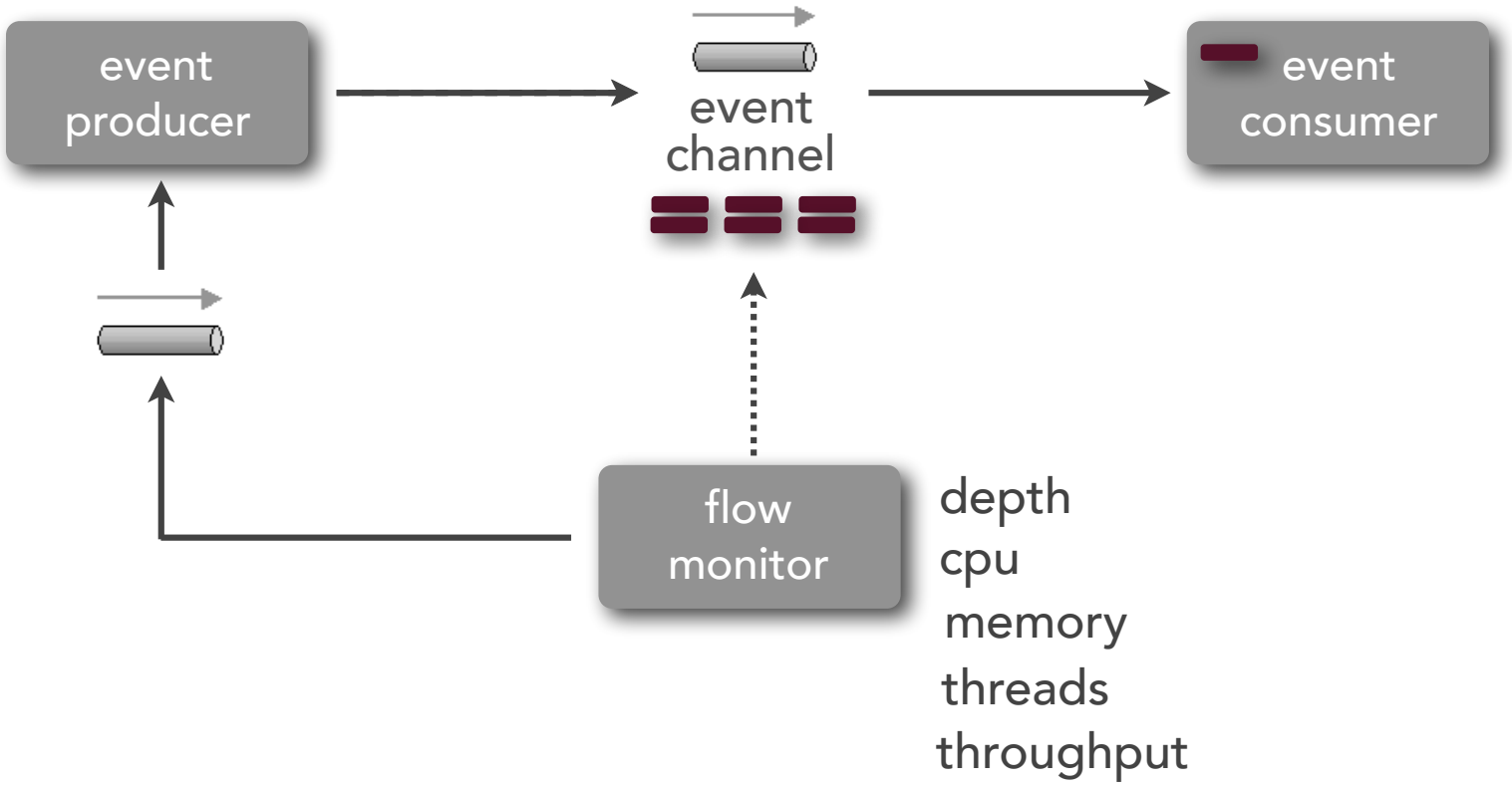
shutdown (broker) vs. slowdown (pattern)

producer control flow pattern



let’s see the issue….

producer control flow pattern



producer control flow pattern

Initialize.java

```
channel.exchangeDeclare("flow.fx", "fanout", true);
channel.queueDeclare("flow.q", true, false, false, null);
channel.queueBind("flow.q", "flow.fx", "");
```

producer control flow pattern

FlowMonitor.java

```
public void execute() throws Exception {
    Channel channel = AMQPCommon.connect();
    long threshold = 10;
    boolean controlFlow = false;
    while (true) {
        long queueDepth = channel.messageCount("trade.eq.q");
        if (queueDepth > threshold && !controlFlow) {
            controlFlow = enableControlFlow(channel);
        } else if (queueDepth <= (threshold/2) && controlFlow) {
            controlFlow = disableControlFlow(channel);
        }
        Thread.sleep(3000);
    }
}
```

producer control flow pattern

FlowMonitor.java

```
private boolean enableControlFlow(Channel channel) {
    byte[] msg = String.valueOf(true).getBytes();
    channel.basicPublish("flow.fx", "", null, msg);
    return true;
}


private boolean disableControlFlow(Channel channel) {
    byte[] msg = String.valueOf(false).getBytes();
    channel.basicPublish("flow.fx", "", null, msg);
    return false;
}
```

producer control flow pattern

Producer.java

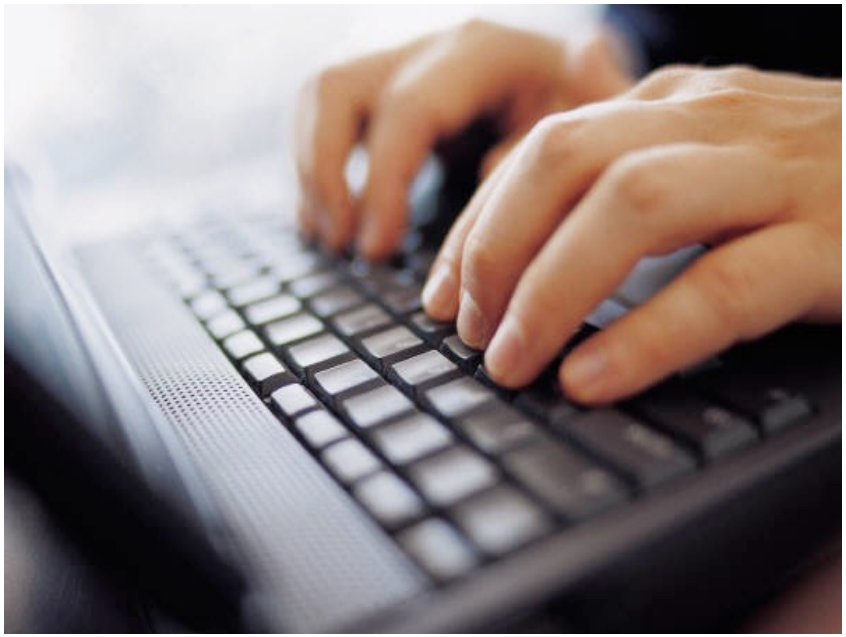
```
public void startListener() {
    new Thread() {
      public void run() {
       Channel channel = connection.createChannel();
       QueueingConsumer consumer = new QueueingConsumer(channel);
       channel.basicConsume("flow.q", true, consumer);
       while (true) {
          QueueingConsumer.Delivery msg = consumer.nextDelivery();
          boolean controlFlow =
            new Boolean(new String(msg.getBody())).booleanValue();
          synchronized(delay) {
               delay = controlFlow ? 3000l : 0l; }
       }
    }}.start();
}
```

producer control flow pattern

Producer.java

```
private void produceMessages() {
   Channel channel = connection.createChannel();
   ...
   channel.basicPublish("", "trade.q", null, message);
   Thread.sleep(delay);
}
```

producer control flow pattern

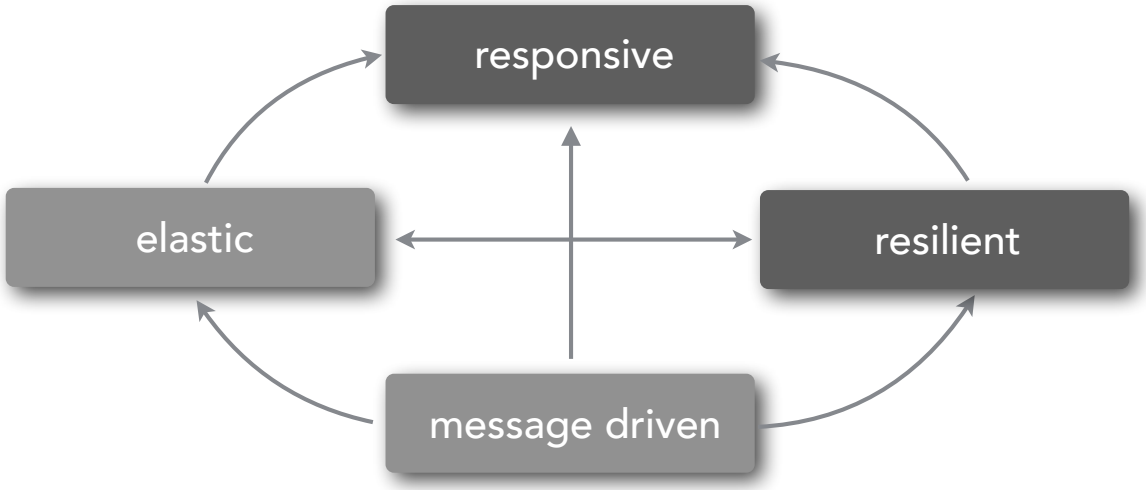


let’s see the result…

Thread Delegate Pattern

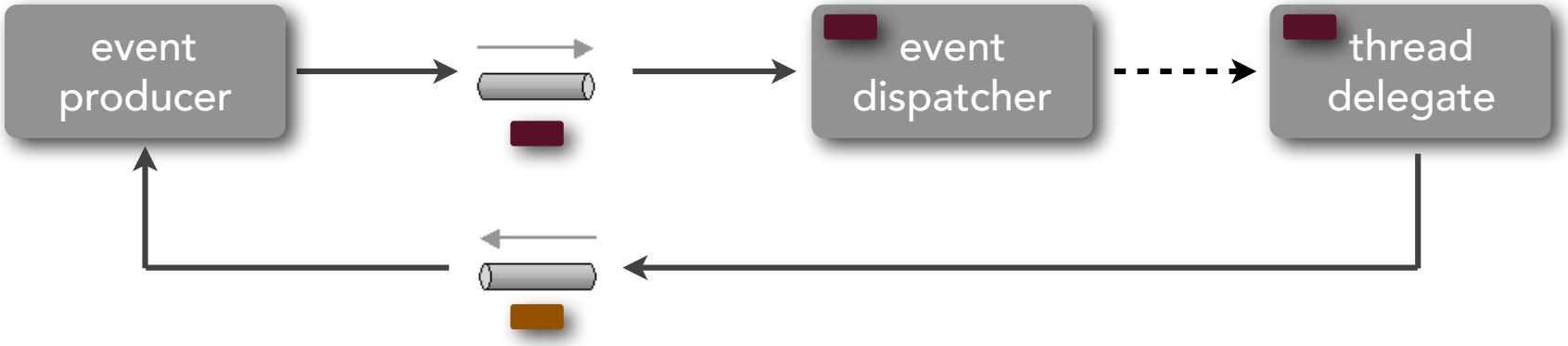
thread delegate pattern

how can you consume messages faster than they are being produced?



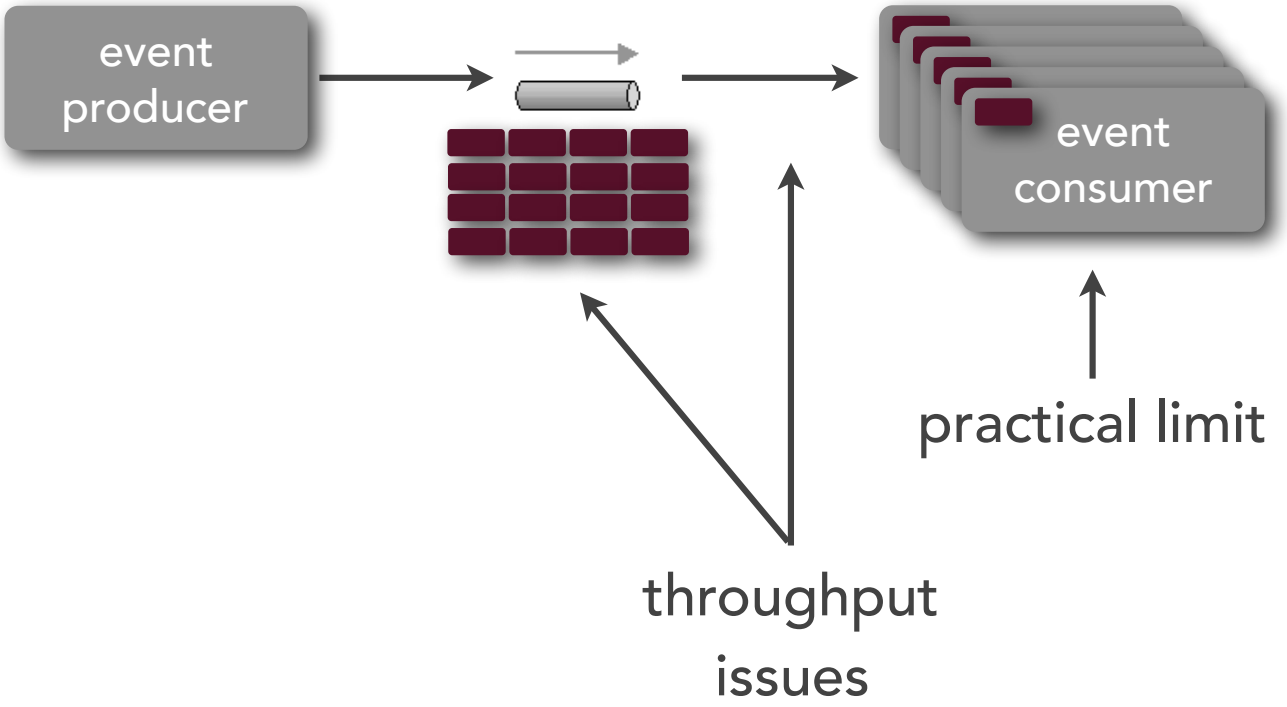
thread delegate pattern

how can you consume messages faster than they are being produced?



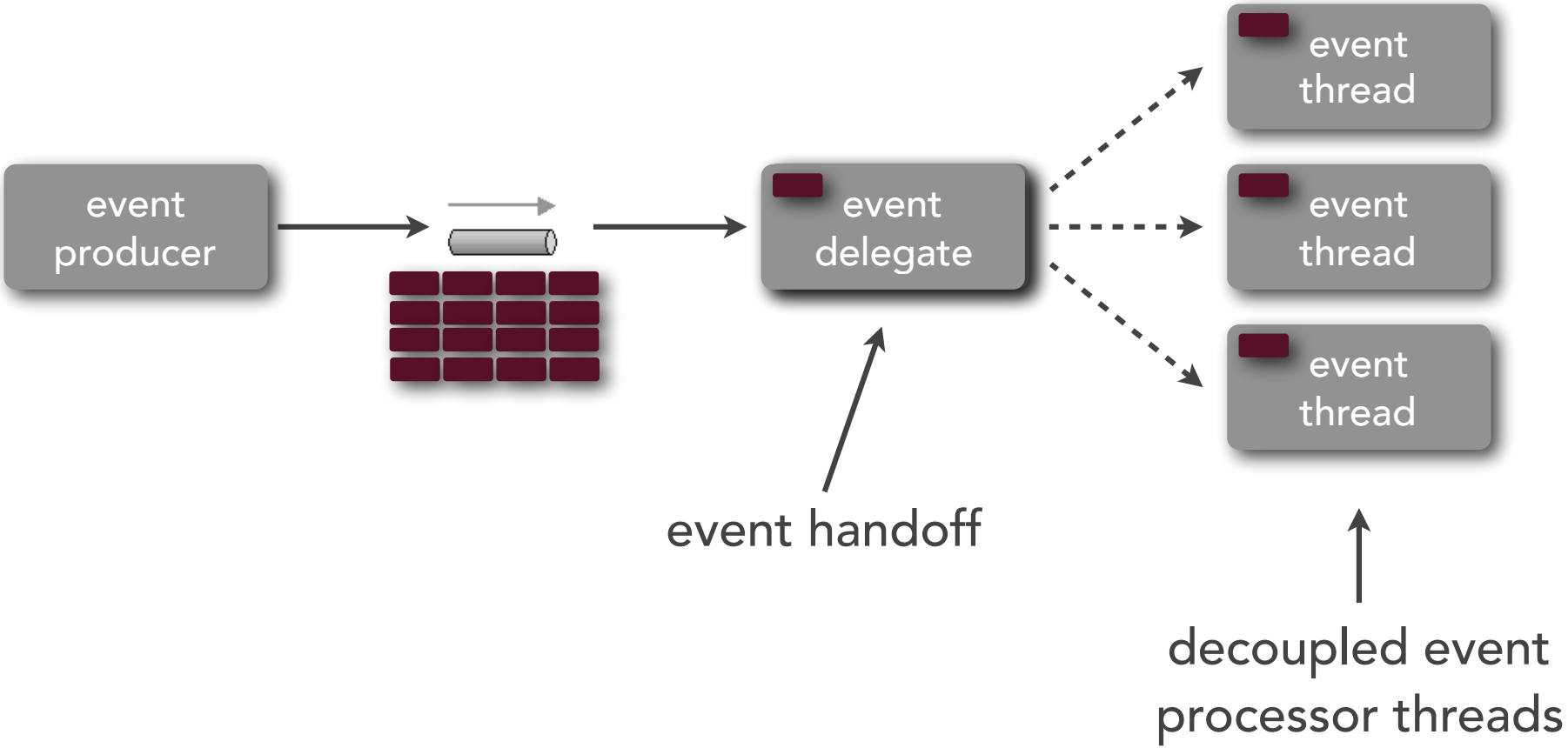
thread delegate pattern

the issue...

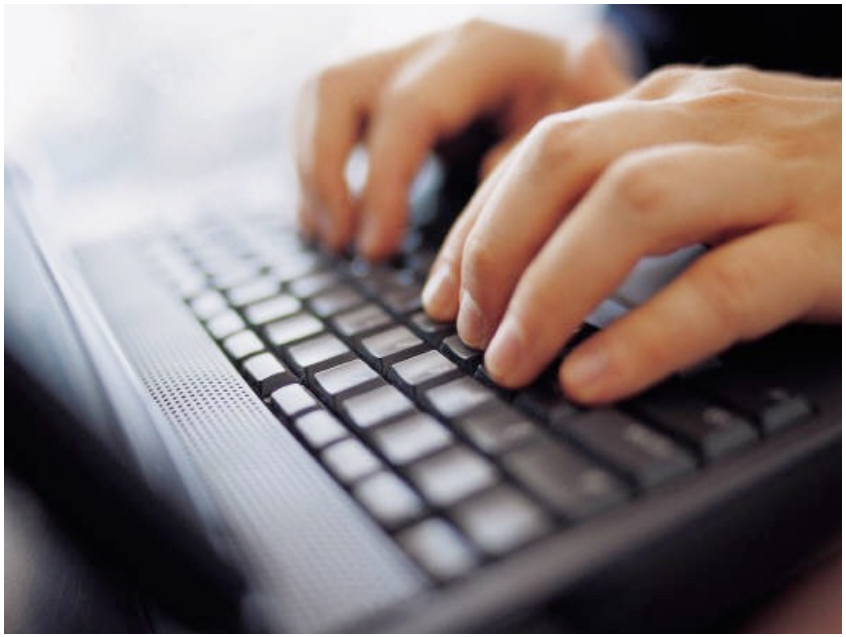


thread delegate pattern

the pattern implementation…



thread delegate pattern



let's see the issue…

thread delegate pattern

Dispatcher.java

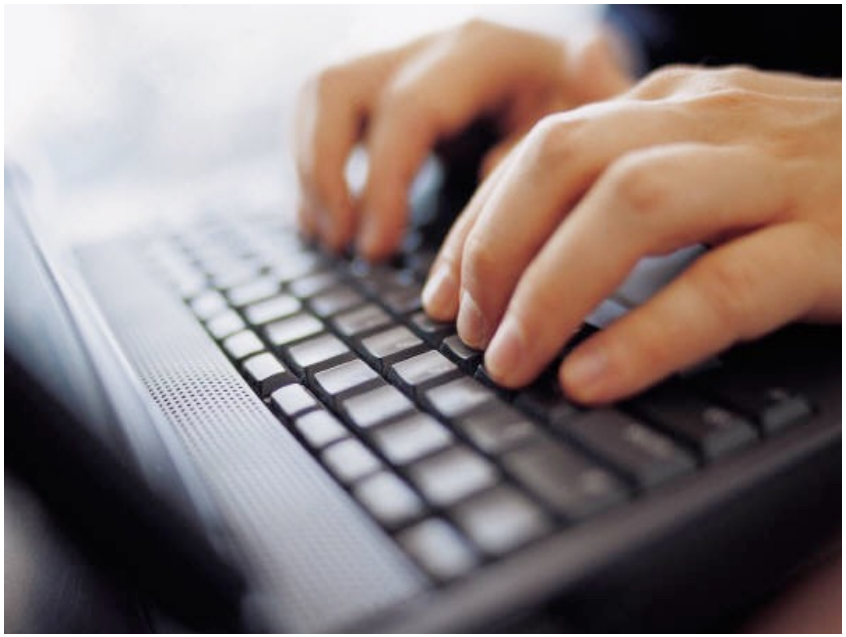
```
Channel channel = AMQPCommon.connect();
QueueingConsumer consumer = new QueueingConsumer(channel);
channel.basicConsume("trade.eq.q", true, consumer);

while (true) {
   QueueingConsumer.Delivery msg = consumer.nextDelivery();
   new Thread(new POJOThreadProcessor(
      new String(msg.getBody()))).start();
}
```

thread delegate pattern

Processor.java

```
public class POJOThreadProcessor implements Runnable {
  private String message;

   public POJOThreadProcessor(String message) {
      this.message = message;
   }

    public void run() {
      //process message
      Thread.sleep(2000);
      System.out.println("Trade placed: " + message);
      //send response back to producer
    }
 }
```

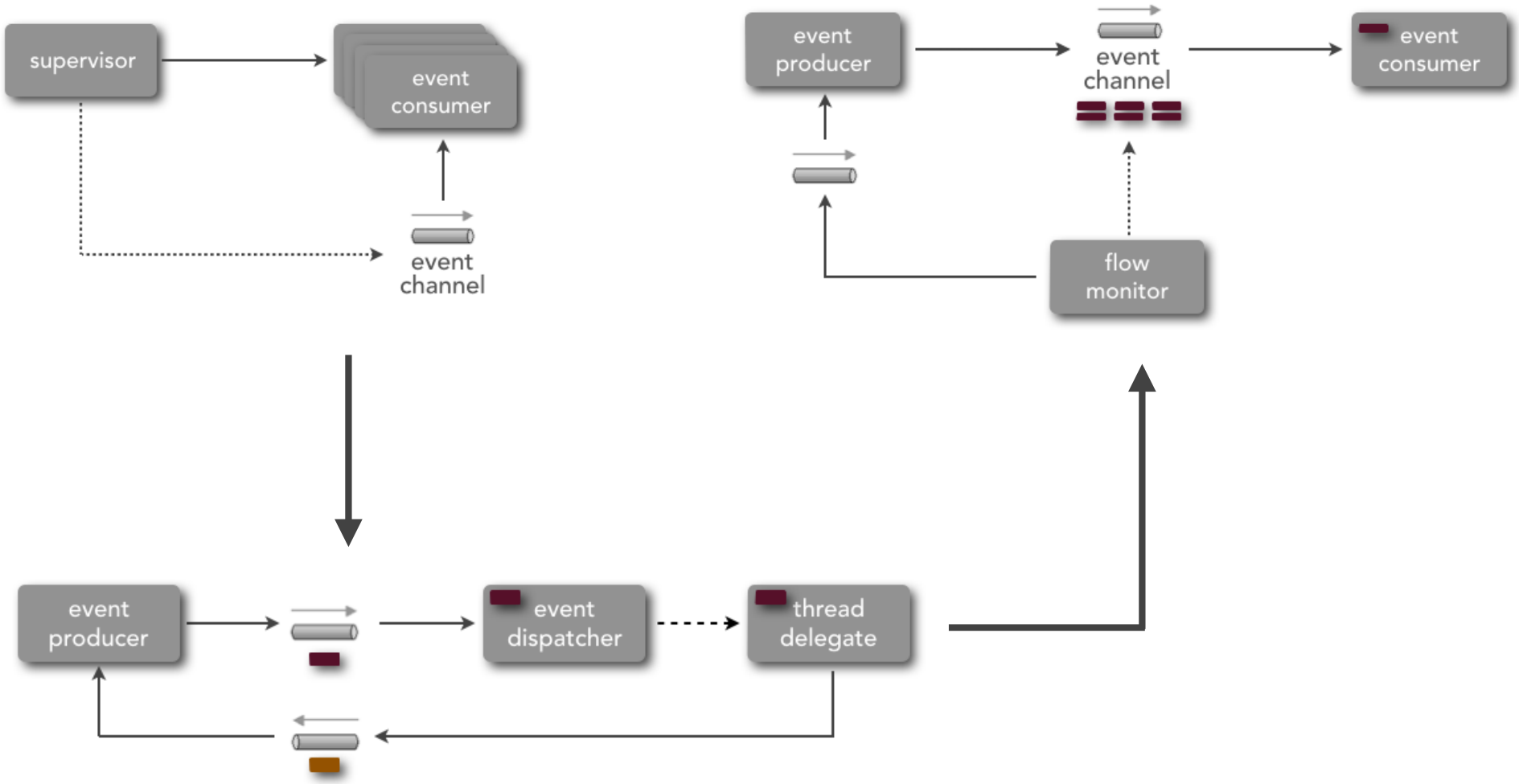
thread delegate pattern



let’s see the result…

Combining Patterns

combining patterns



Reactive Architecture Patterns

Mark Richards

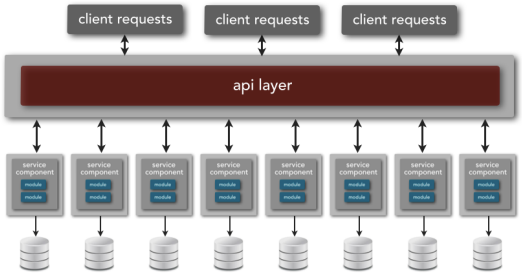
Independent Consultant
Hands-on Software Architect
Published Author / Conference Speaker

http://www.wmrichards.com
https://www.linkedin.com/in/markrichards3
@markrichardssa

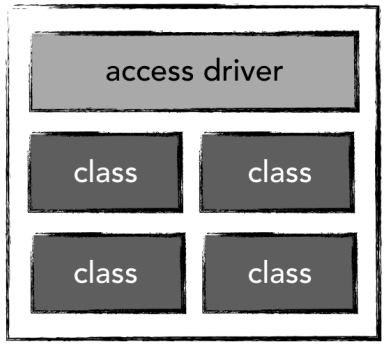
Event Router and Event Consumer Patterns

event consumer vs. event router

event processor design



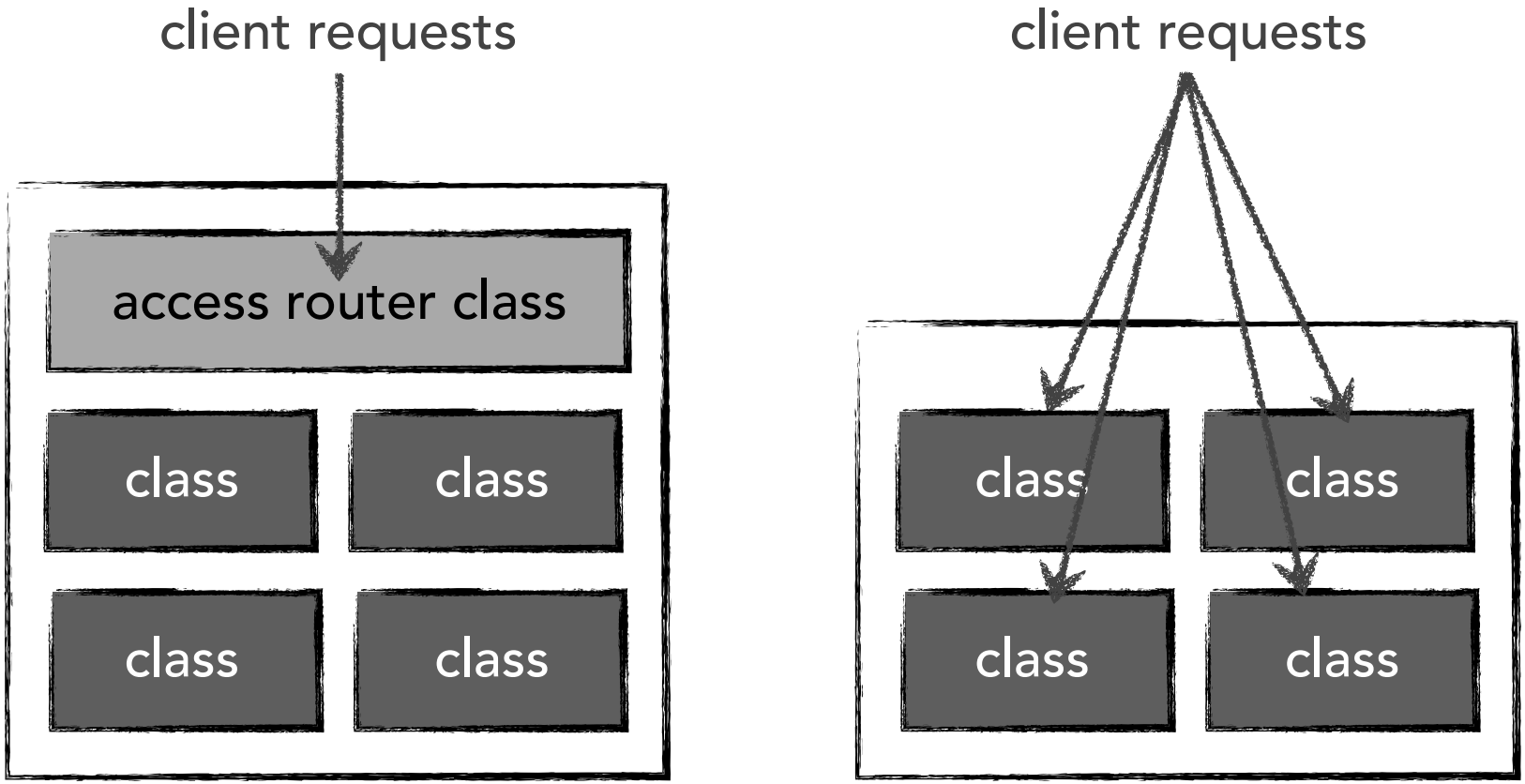
event-driven microservices



service template design

event consumer vs. event router

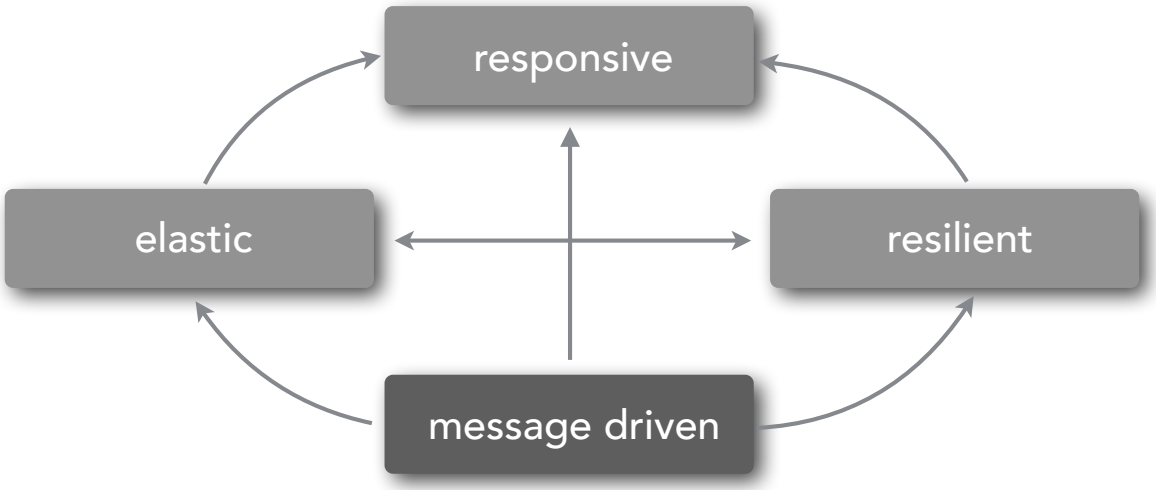
service template design



Event Router Pattern

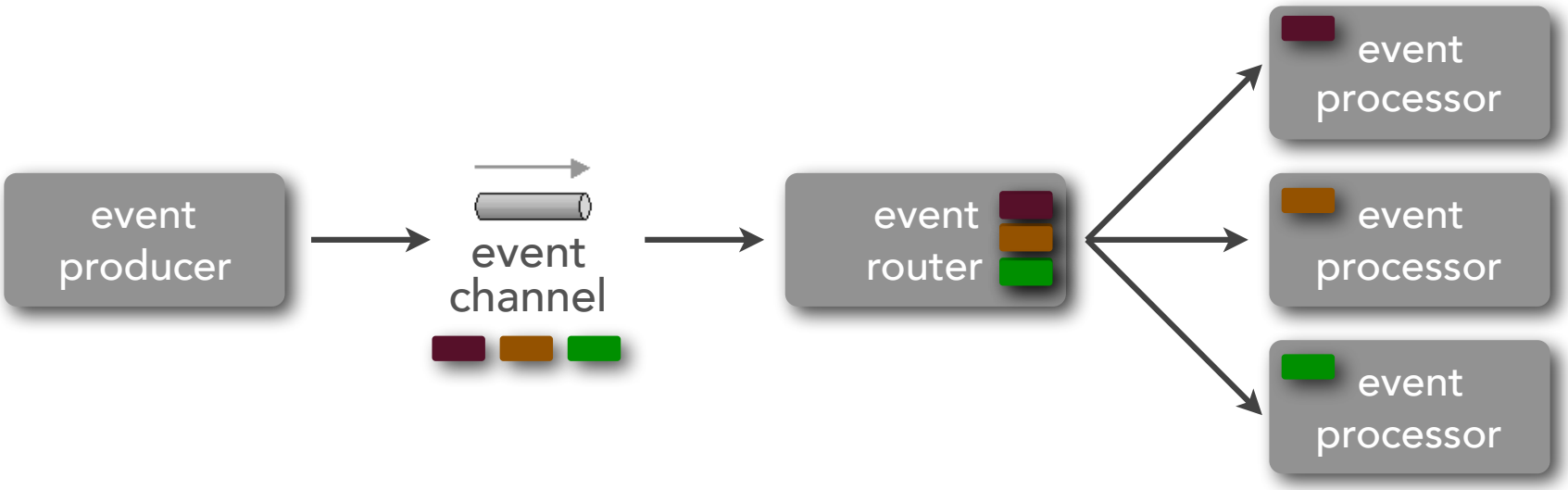
event router pattern

how do you consolidate the processing of different messages from the same producer?



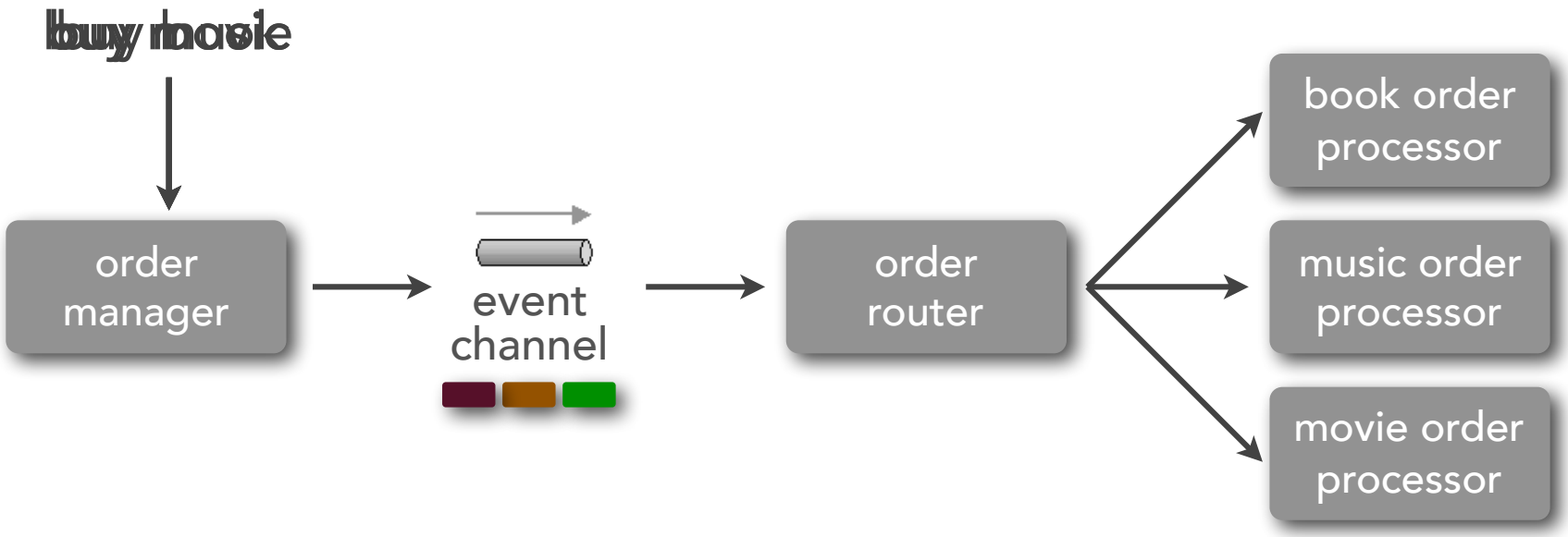
event router pattern

how do you consolidate the processing of different messages from the same producer?



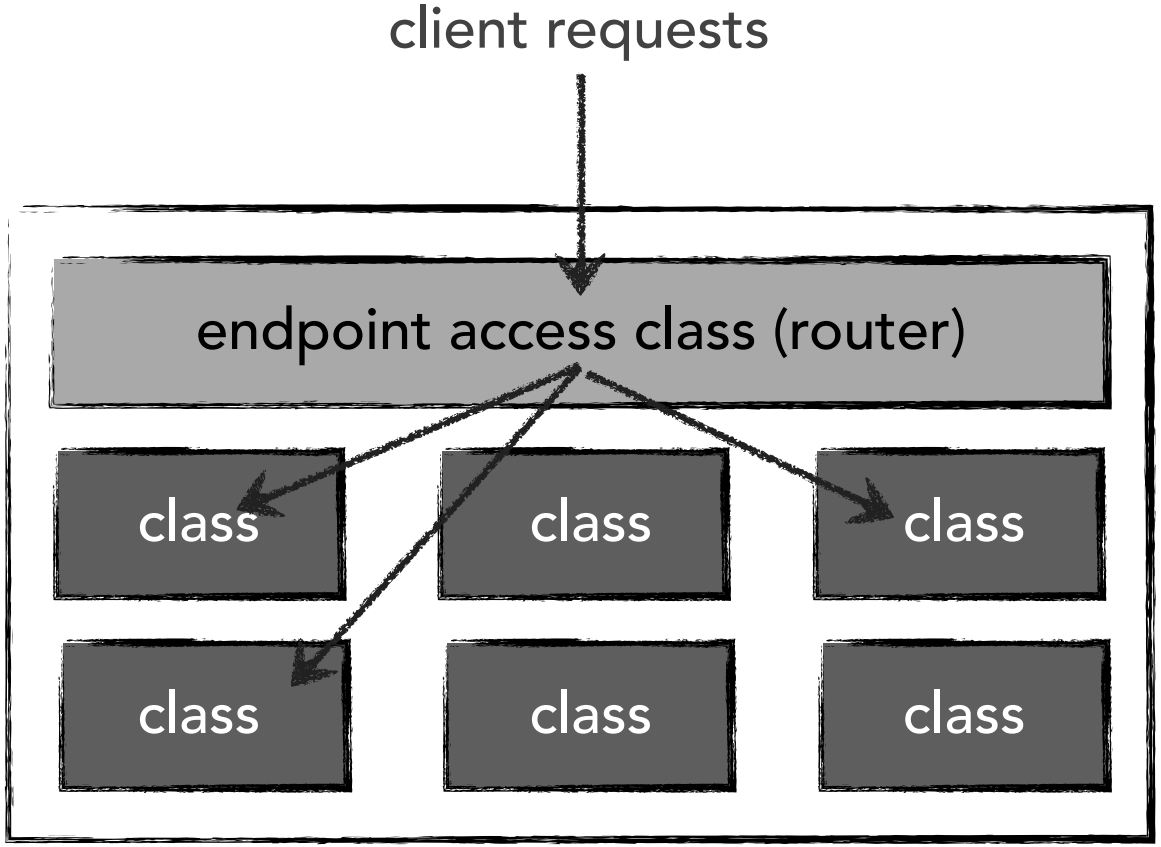
event router pattern

example



event router pattern

context: service template design



event router pattern

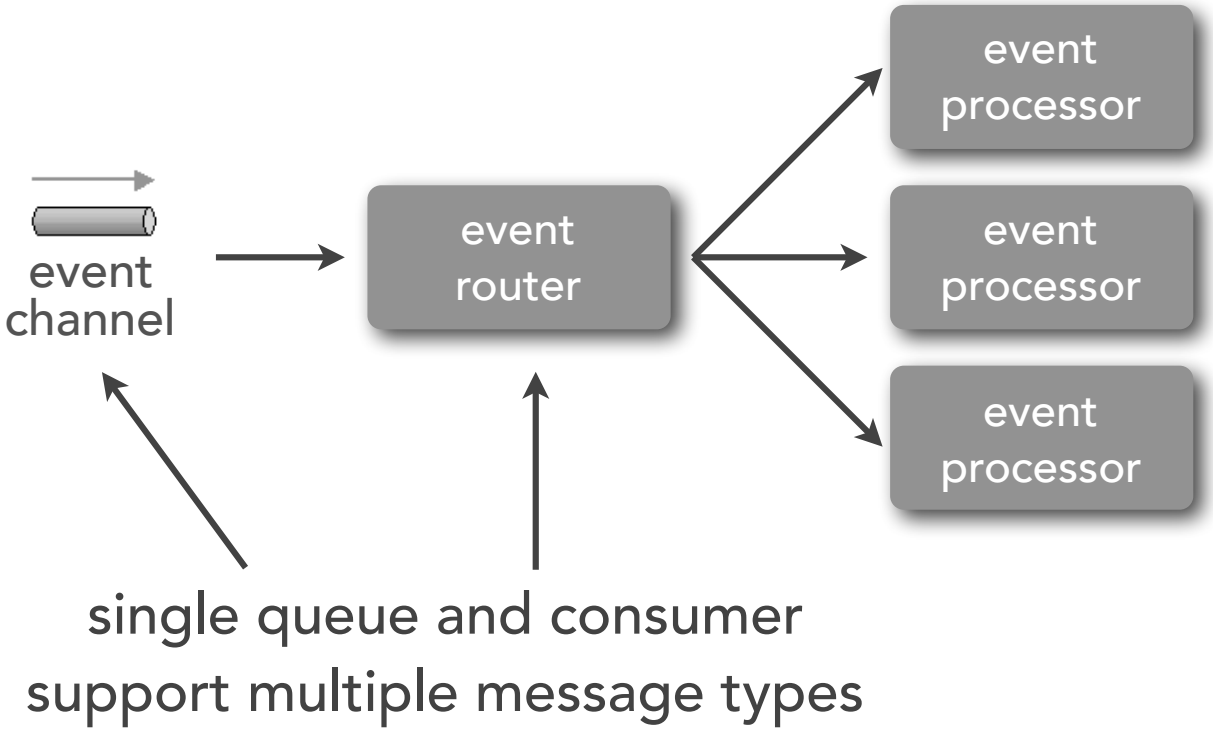
Producer.java

```
AMQP.BasicProperties.Builder builder =
     new AMQP.BasicProperties().builder();
Map<String, Object> props = new HashMap<String, Object>();
props.put("type", "book");
builder.headers(props);
AMQP.BasicProperties header = builder.build();
channel.basicPublish("", "order.q", header, msgBody);
```

event router pattern

consequences

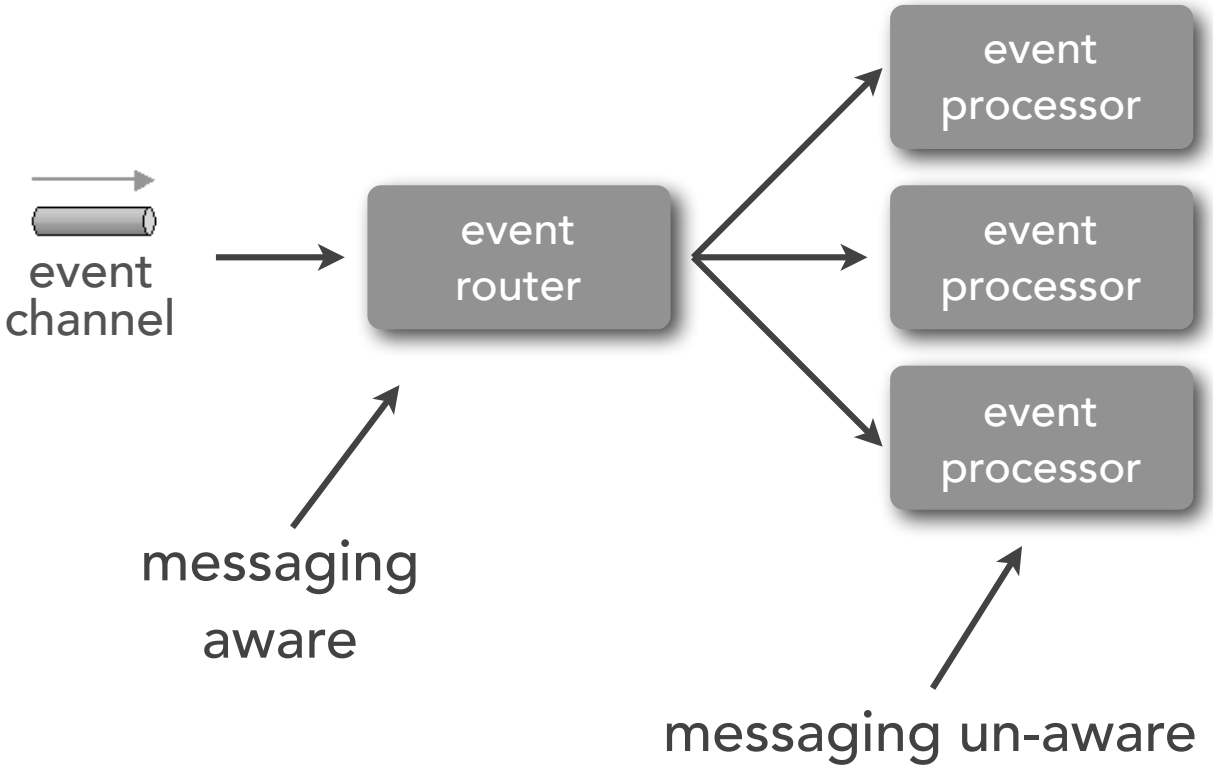
simplifies messaging infrastructure



event router pattern

consequences

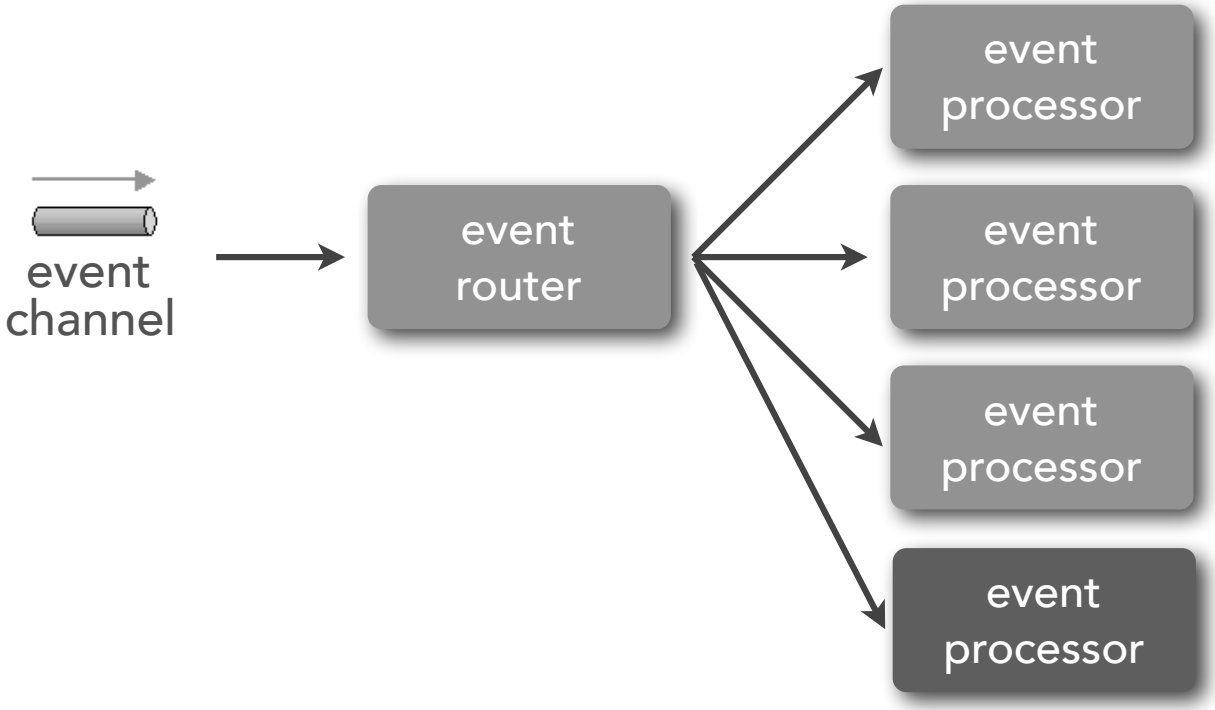
decoupled event processors



event router pattern

consequences

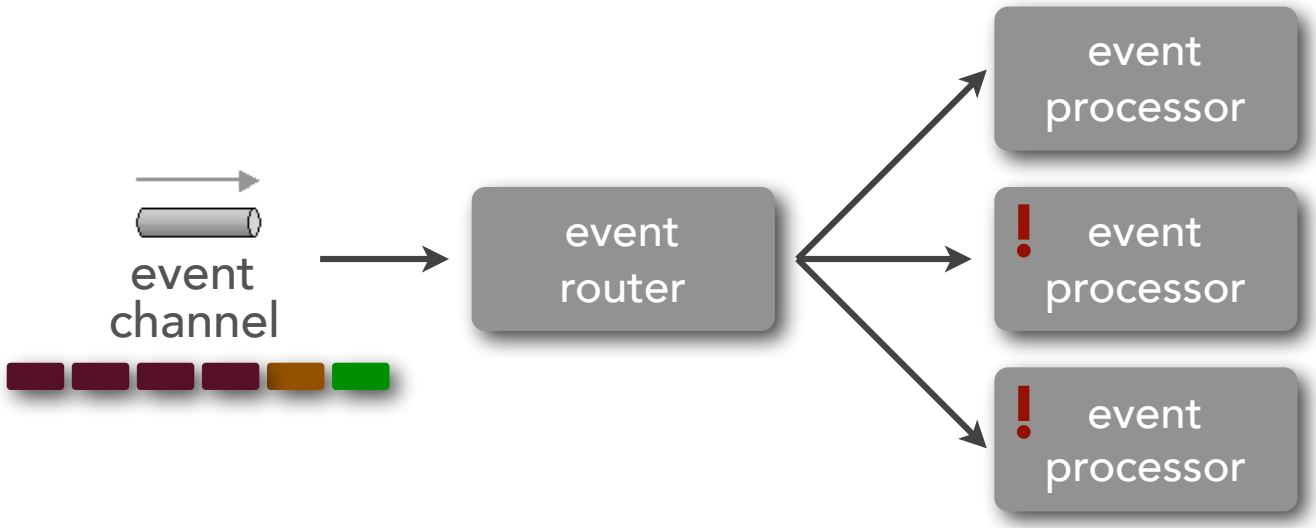
support for evolutionary design



event router pattern

consequences

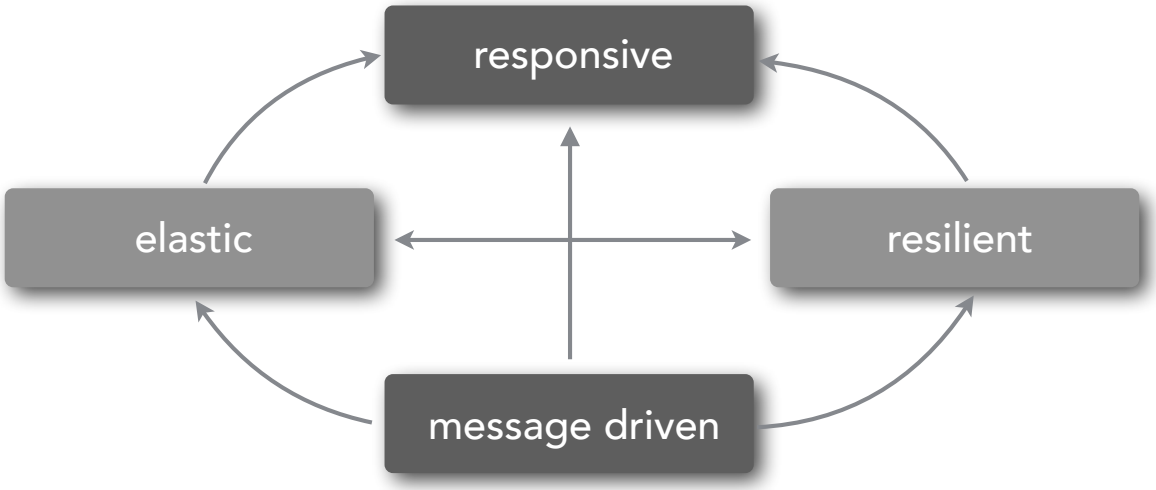
poor message balancing and message distribution



Event Consumer Pattern

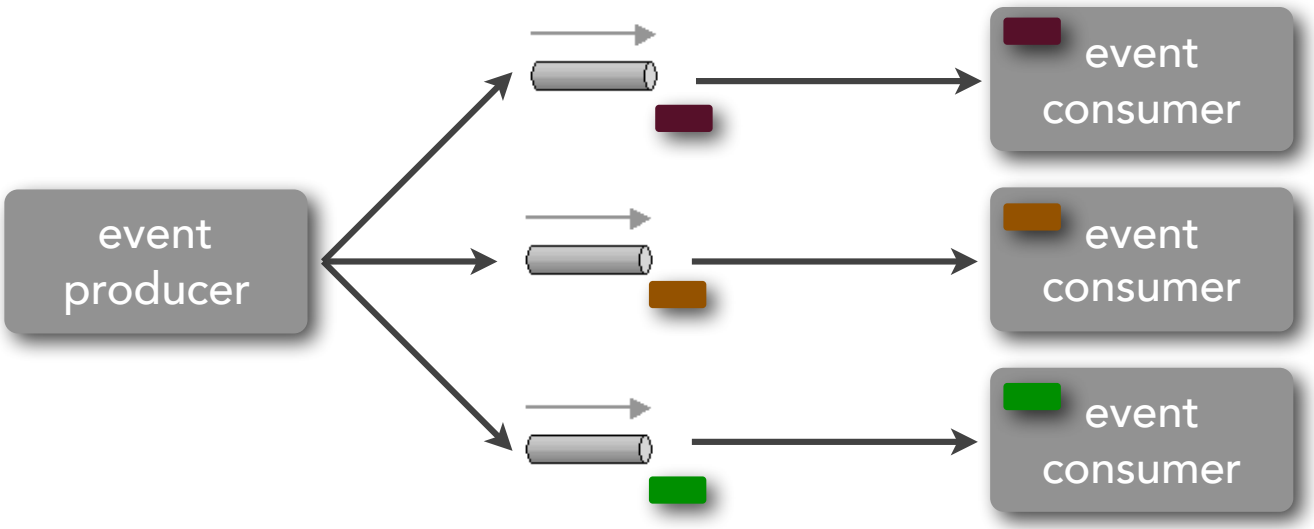
event consumer pattern

how do you optimize the event flow when consuming different messages from the same producer?



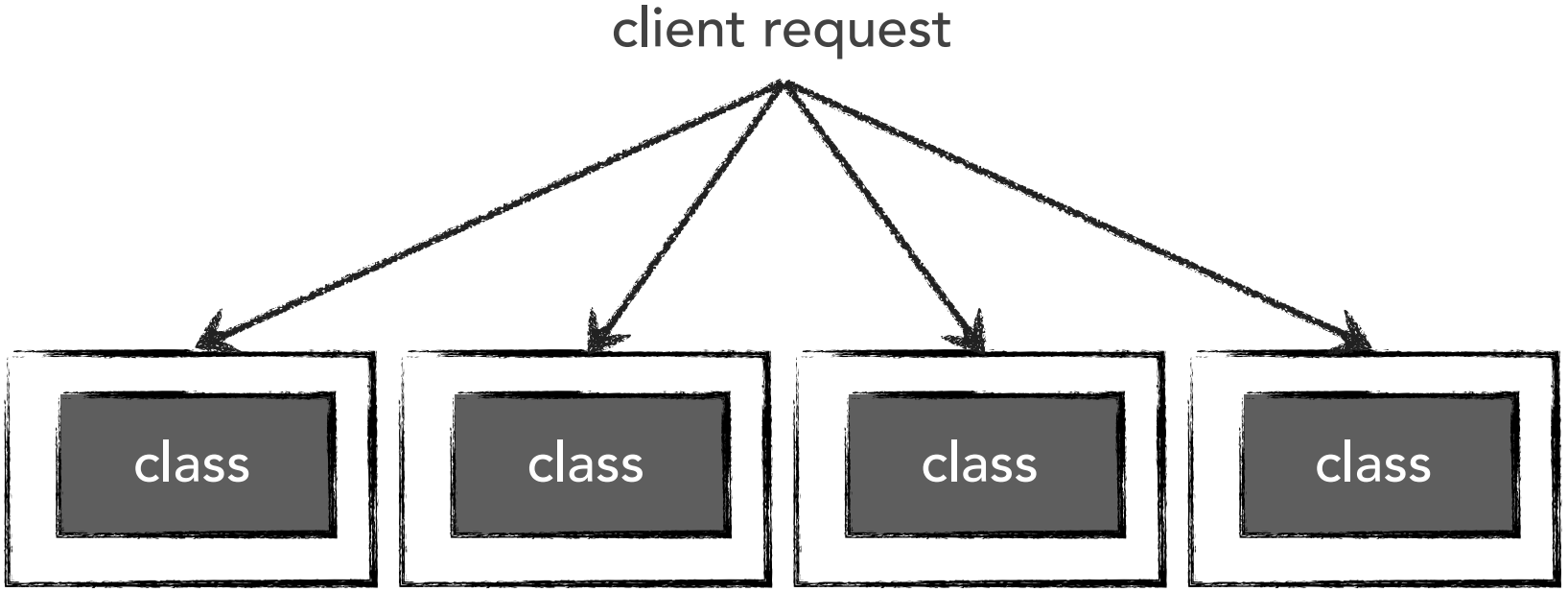
event consumer pattern

how do you optimize the event flow when consuming different messages from the same producer?



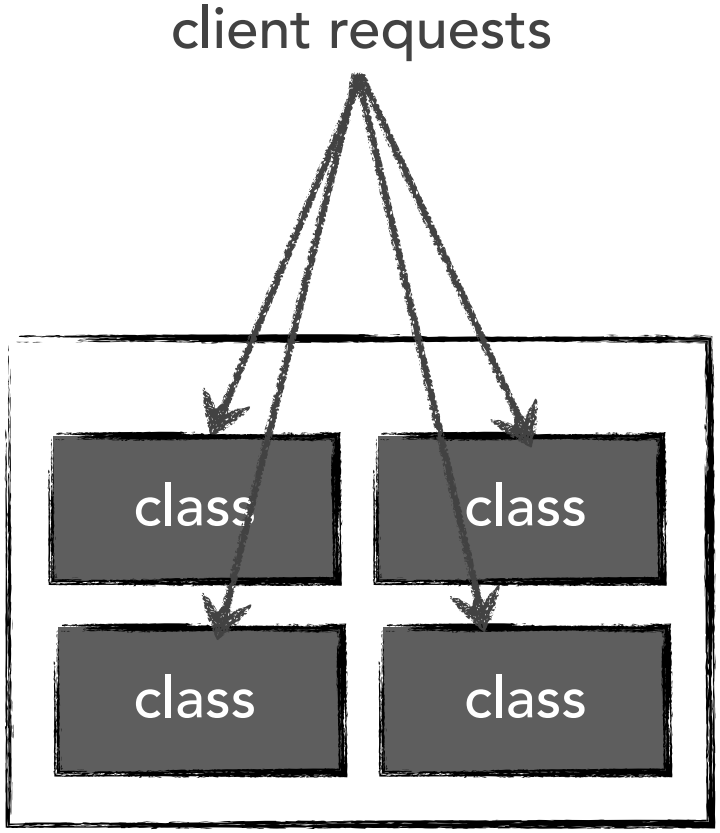
event consumer pattern

service template design



event consumer pattern

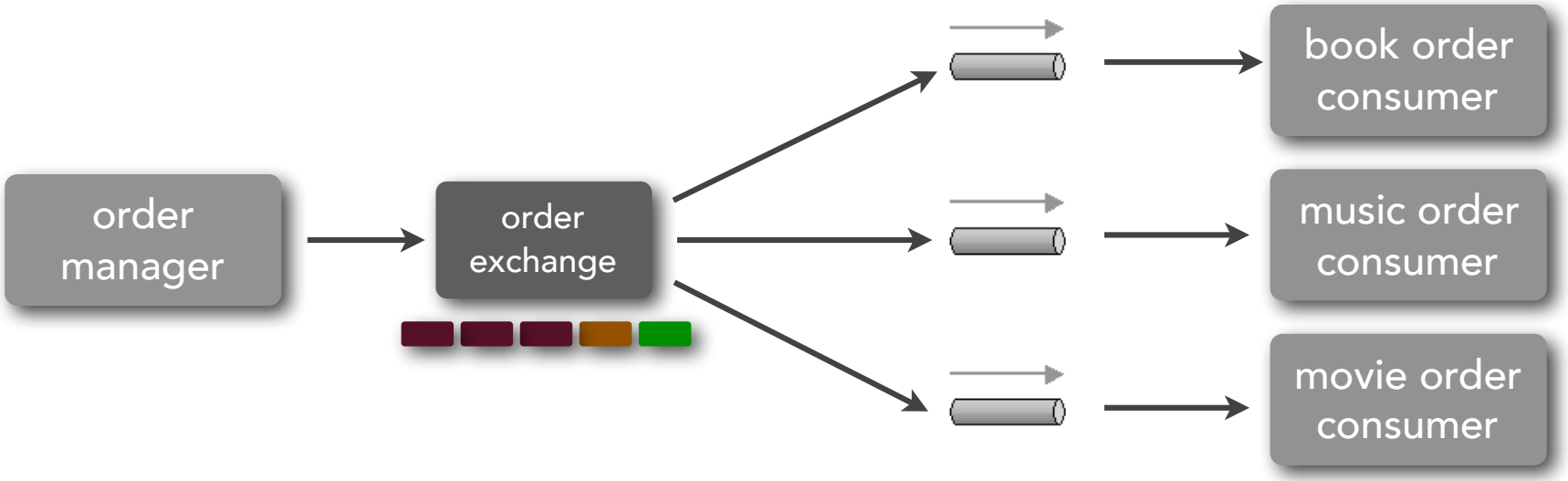
service template design



event consumer pattern

consequences

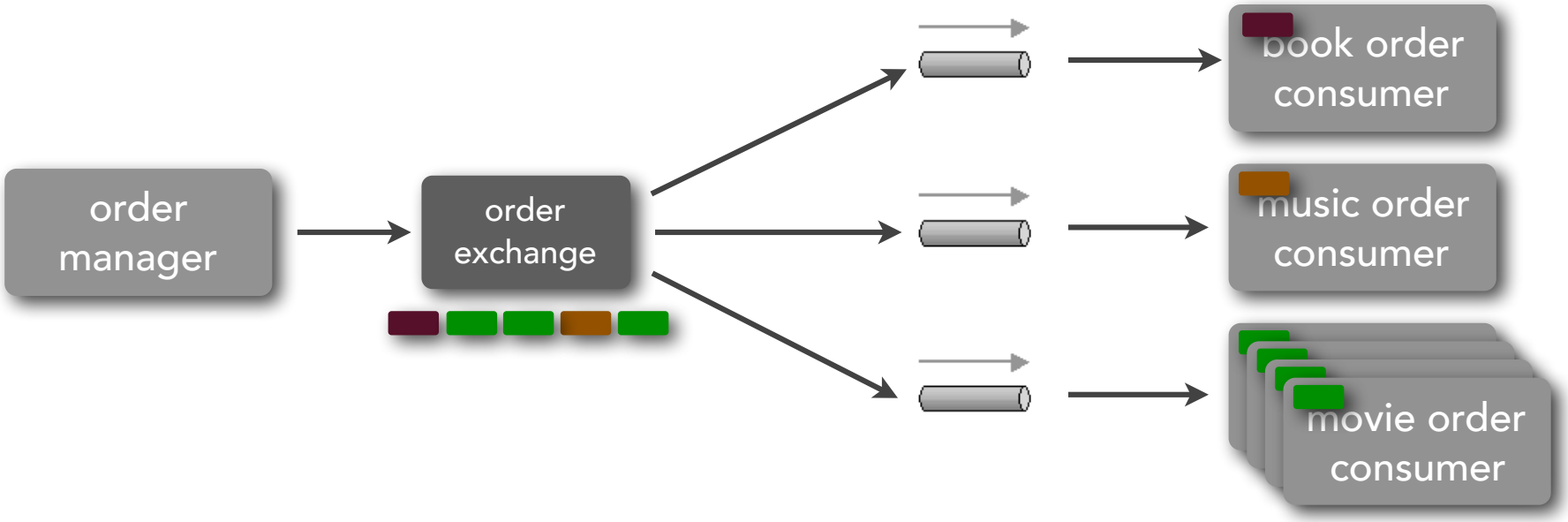
message balancing and message distribution



event consumer pattern

consequences

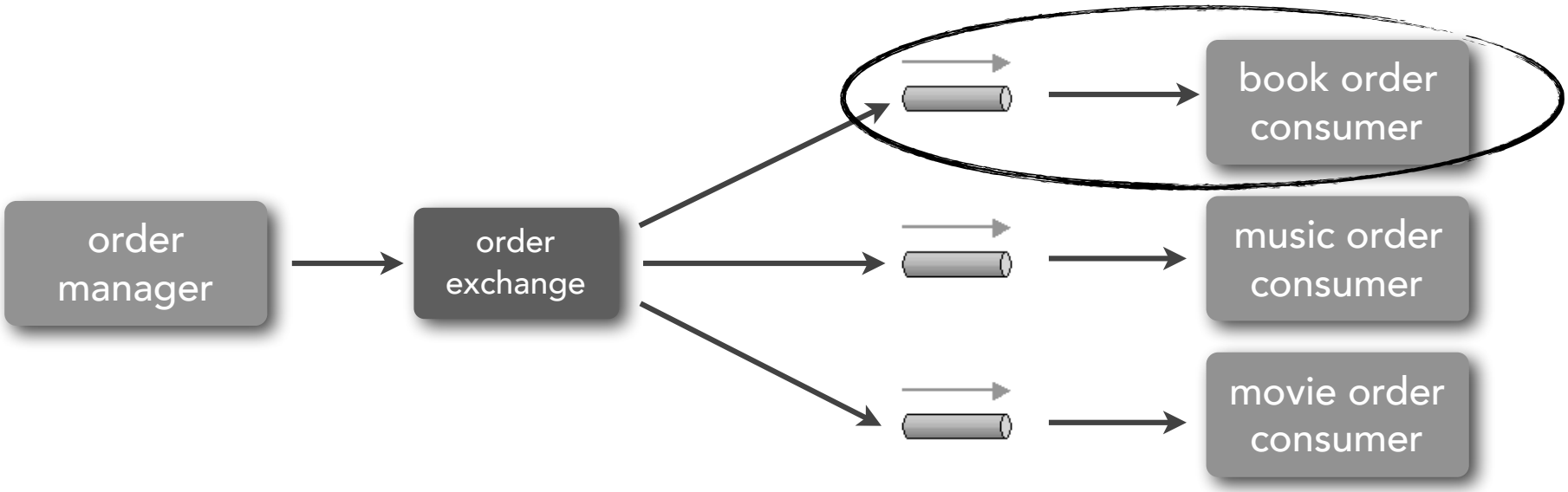
separate consumer load balancing



event consumer pattern

consequences

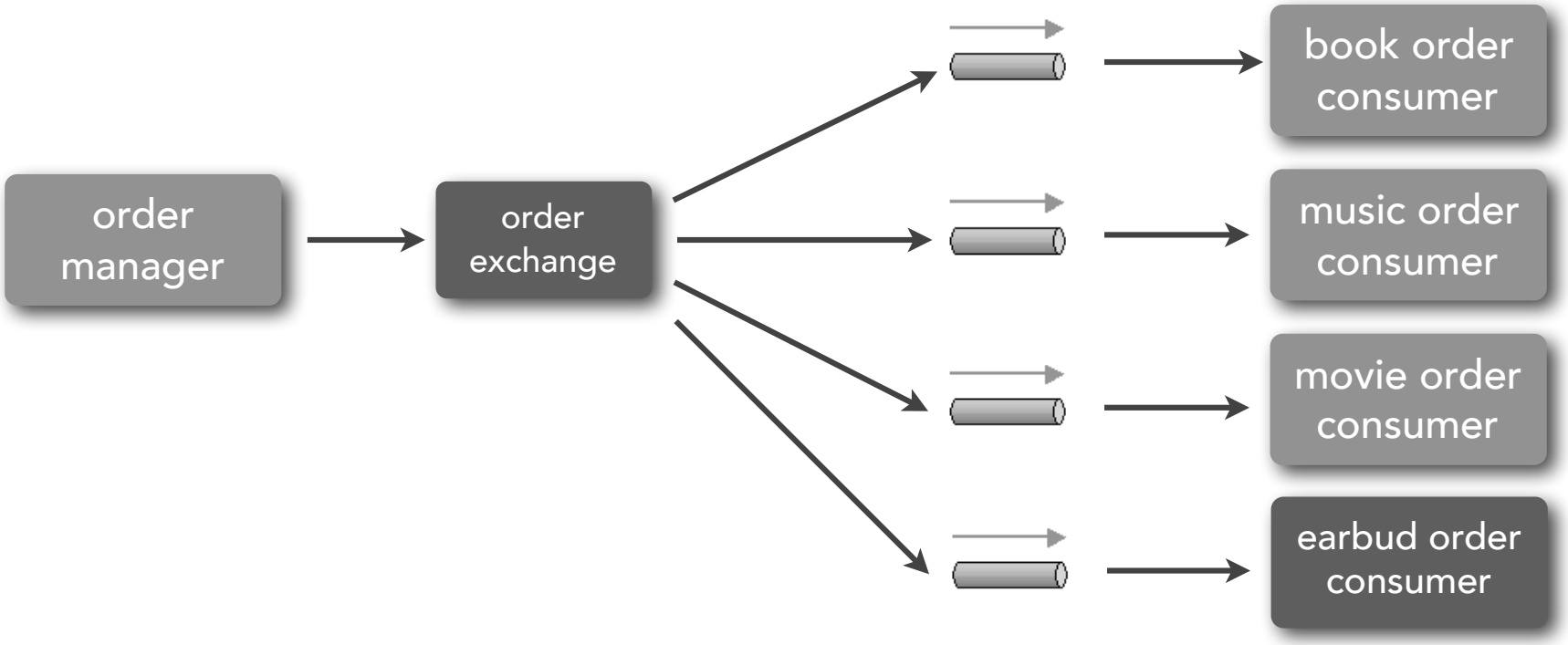
processors are messaging aware



event consumer pattern

consequences

messaging infrastructure must evolve with consumers



Reactive Architecture Patterns

Mark Richards

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

http://www.wmrichards.com
https://www.linkedin.com/in/markrichards3
@markrichardssa