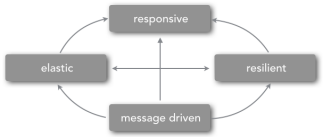
Reactive Architecture Patterns 1

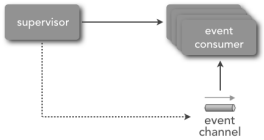
Mark Richards
Independent Consultant
Hands-on Software Architect / Published Author
Founder, DeveloperToArchitect.com
www.wmrichards.com

Author of *Software Architecture Fundamentals Video Series* (O'Reilly)
Author of Microservices Pitfalls and AntiPatterns (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)

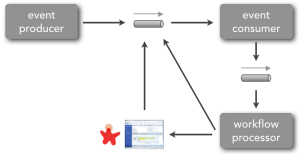
reactive architecture agenda



reactive architecture overview



consumer supervisor pattern



workflow event pattern

source code

https://github.com/wmr513/reactive

source code

```
3 import com.rabbitmq.client.Channel;
6
7 public class AMQPCommon {
8
9     public static Channel connect() throws Exception {
10        ConnectionFactory factory = new ConnectionFactory();
11        factory.setHost("127.0.0.1");
12        factory.setPort(32768);
13        Connection conn = factory.newConnection();
14        return conn.createChannel();
15    }
16
17    public static void close(Channel channel) throws Exception {
18        channel.close();
19        channel.getConnection().close();
20    }
21
22 }
```

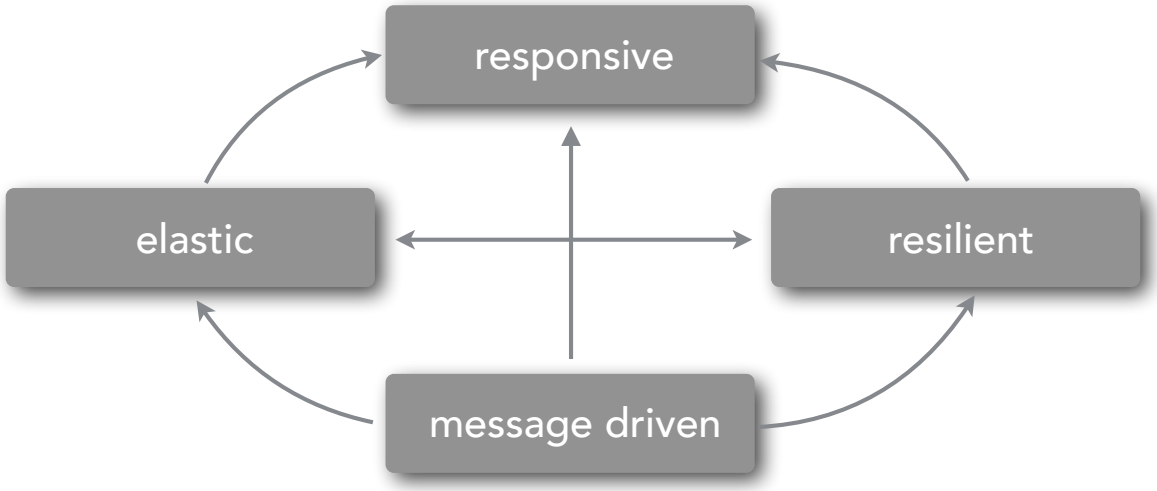
source code

```
public class AMQPInitialize {

    public static void main(String[] args) throws Exception {
        Channel channel = AMQPCommon.connect();

        //create the durable exchanges
        channel.exchangeDeclare("flow.fx", "fanout", true);
        channel.exchangeDeclare("orders.dx", "direct", true);
        System.out.println("exchanges created.");

        //create the durable queues
        channel.queueDeclare("trade.request.q", true, false, false, null);
        channel.queueDeclare("trade.response.q", true, false, false, null);
        channel.queueDeclare("config.q", true, false, false, null);
        channel.queueDeclare("flow.q", true, false, false, null);
        channel.queueDeclare("trade.eq.q", true, false, false, null);
        channel.queueDeclare("trade.1.q", true, false, false, null);
        channel.queueDeclare("trade.2.q", true, false, false, null);
        channel.queueDeclare("workflow.q", true, false, false, null);
```

Reactive Architecture Overview

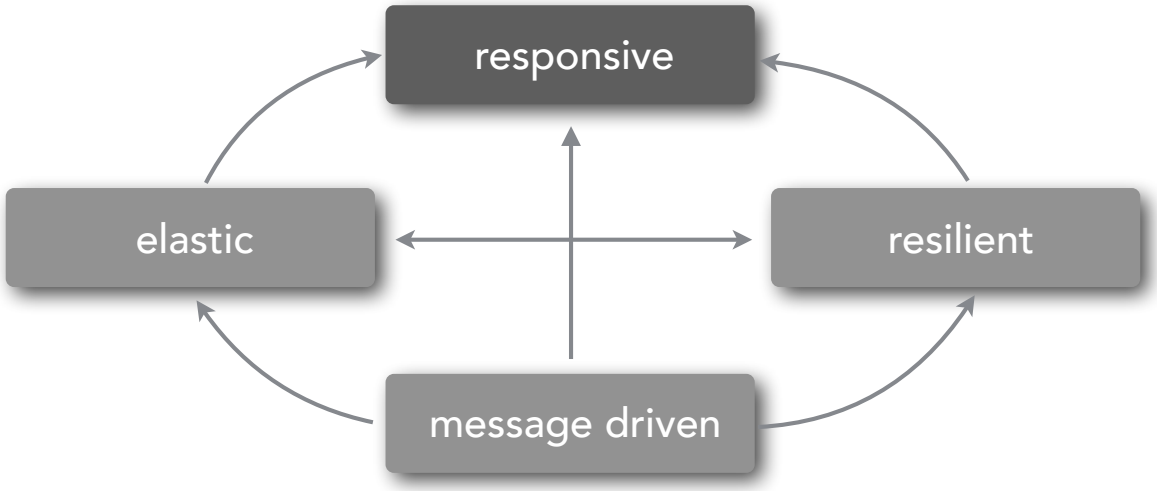
reactive architecture

reactive manifesto



reactive architecture

reactive manifesto

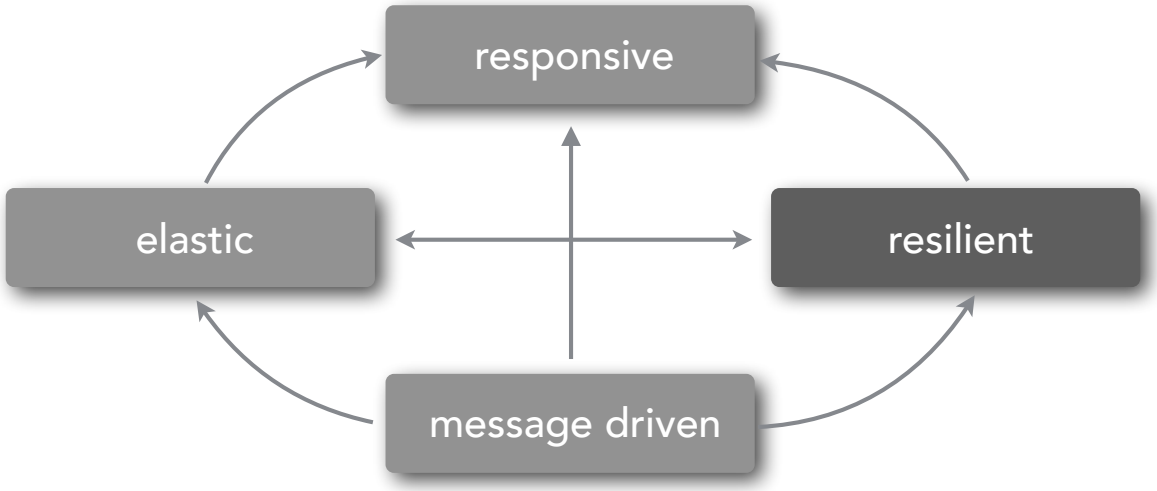


the system responds in a consistent, rapid, and timely manner whenever possible

how the system reacts to users

reactive architecture

reactive manifesto

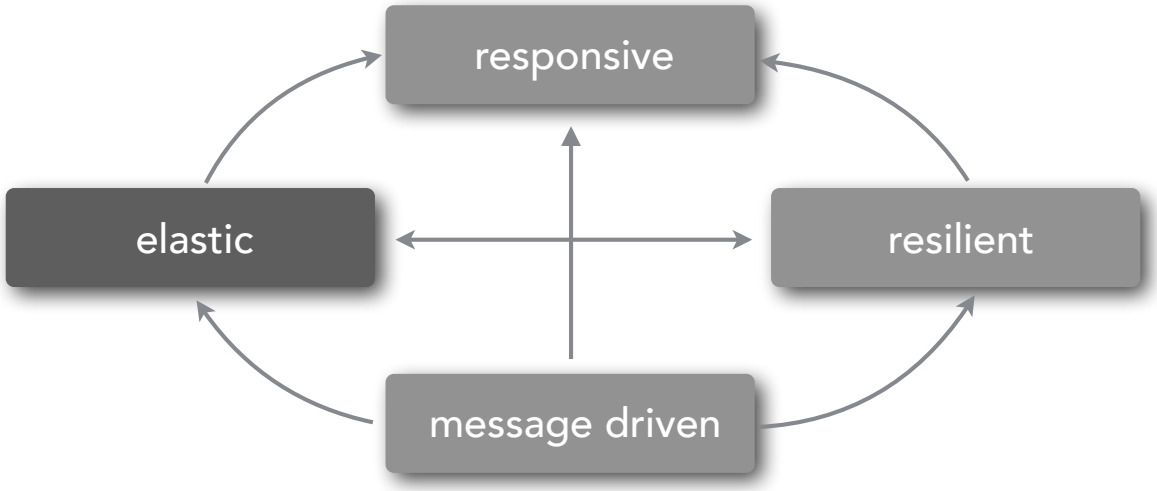


the system stays responsive after a failure through replication, containment, isolation, and delegation

how the system reacts to failures

reactive architecture

reactive manifesto

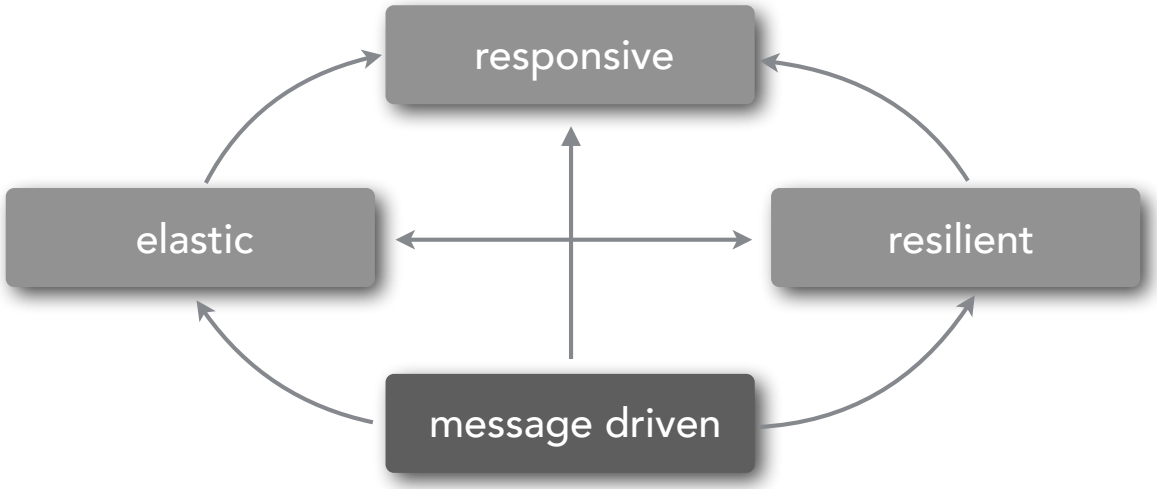


the system stays responsive under varying workload

how the system reacts to load

reactive architecture

reactive manifesto

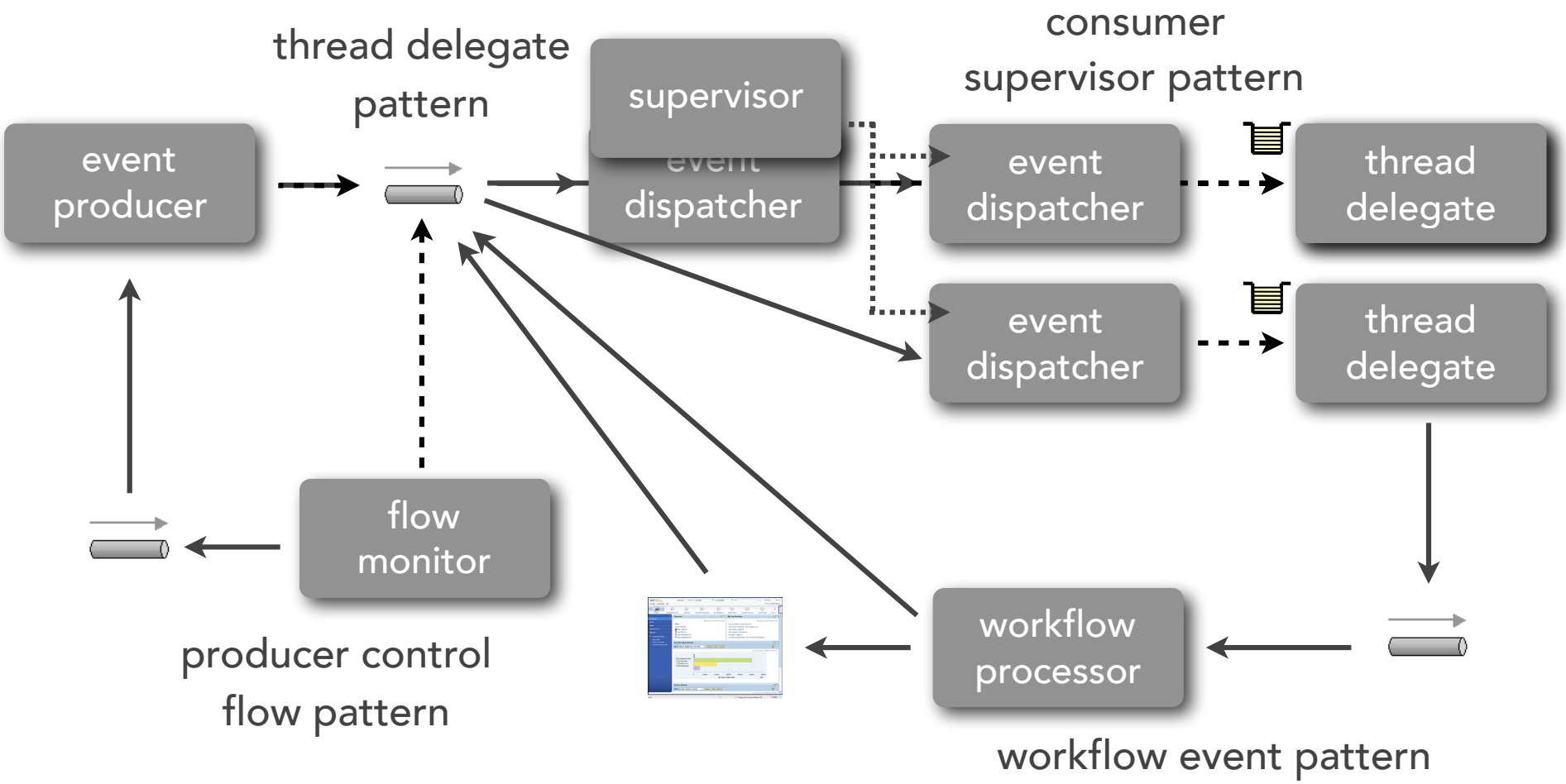


the system relies on asynchronous messaging to ensure loose coupling, isolation, location transparency, and error delegation

how the system reacts to events

reactive architecture

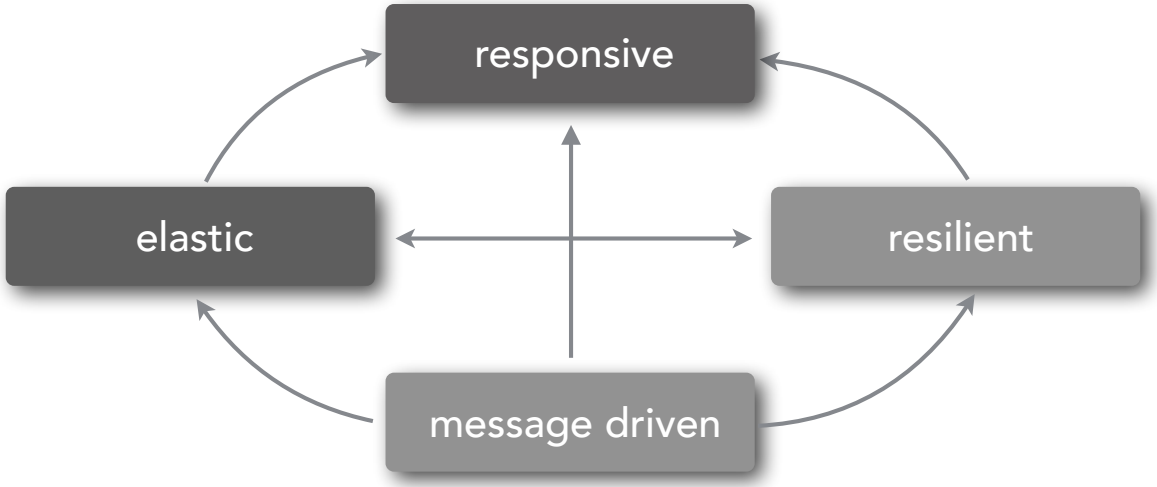
self-healing and self-monitoring systems that can automatically configure and repair themselves



Consumer Supervisor Pattern

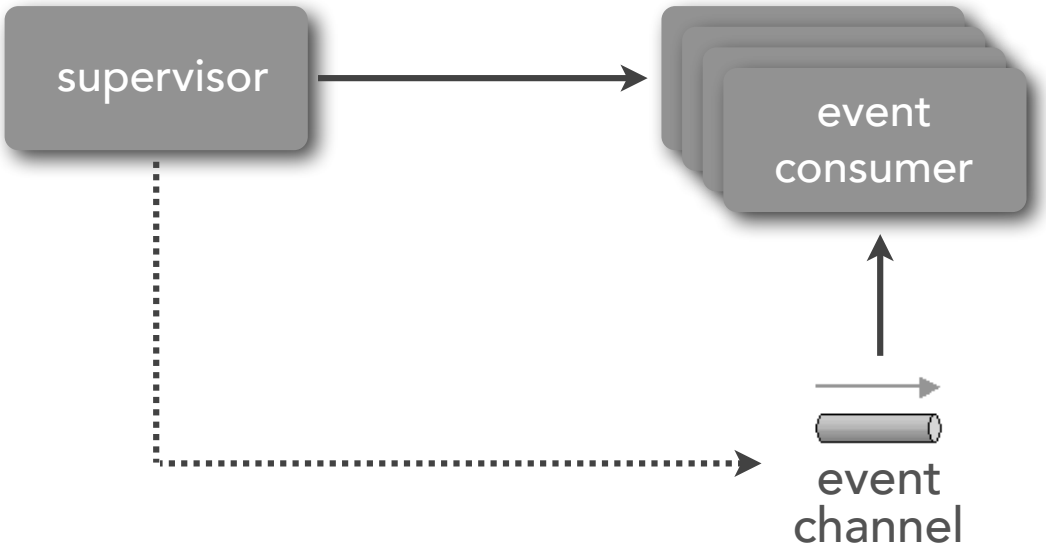
consumer supervisor pattern

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

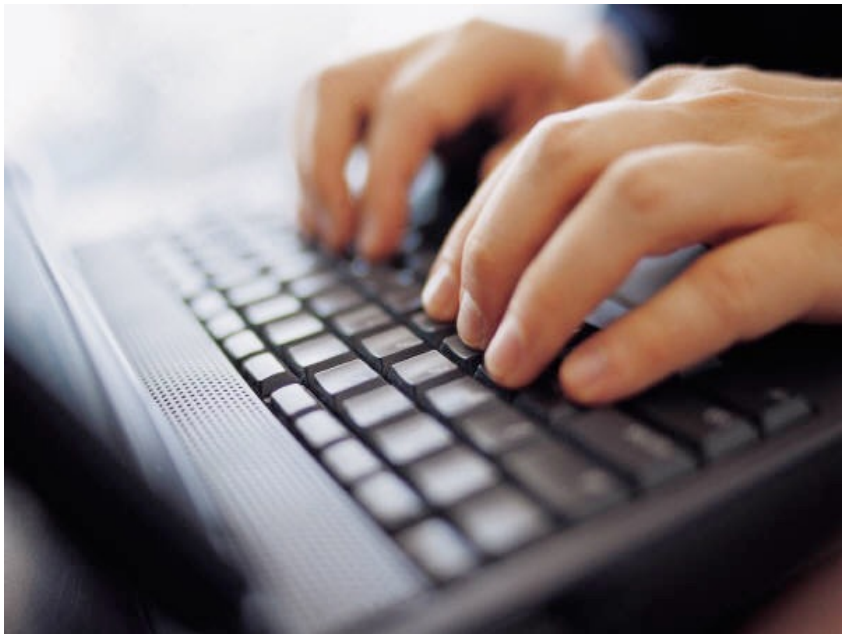


consumer supervisor pattern

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

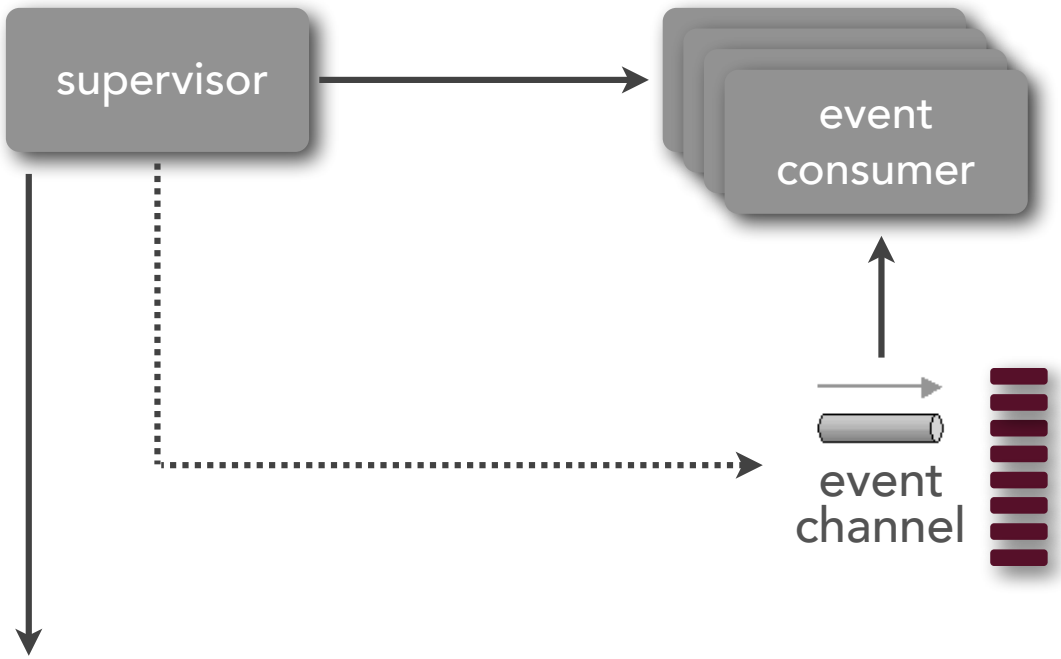


consumer supervisor pattern



let’s see the issue….

consumer supervisor pattern



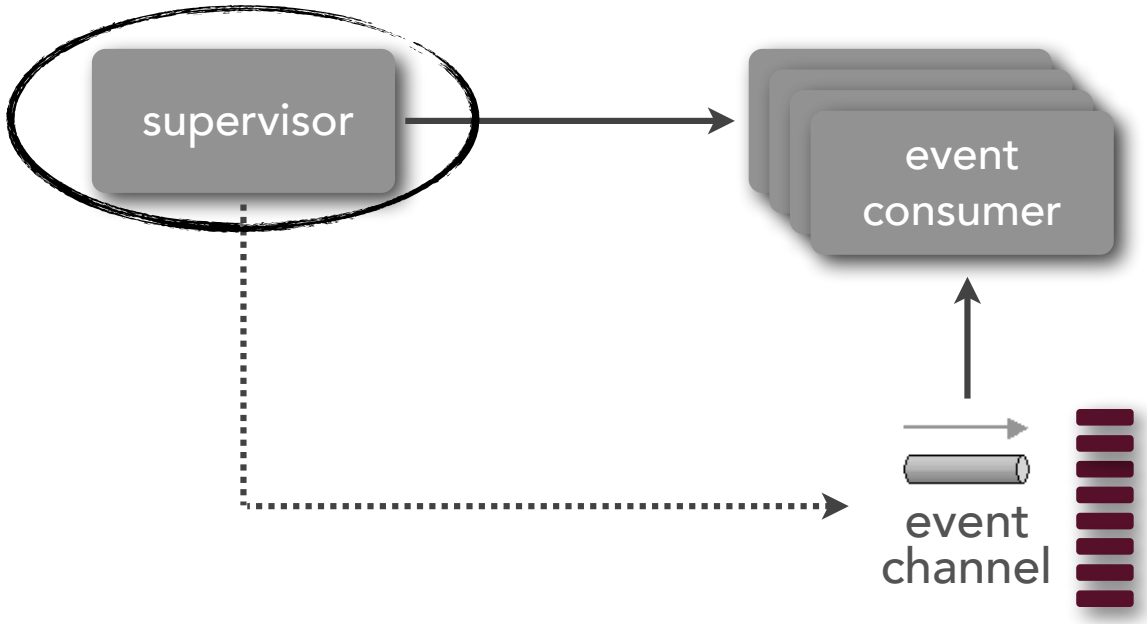
continually monitor queue depth (e.g., 1000ms)

determine consumers needed (e.g., depth/2)

apply max threshold (e.g., 1000 consumers)

add or remove consumers as needed

consumer supervisor pattern



consumer supervisor pattern

supervisor

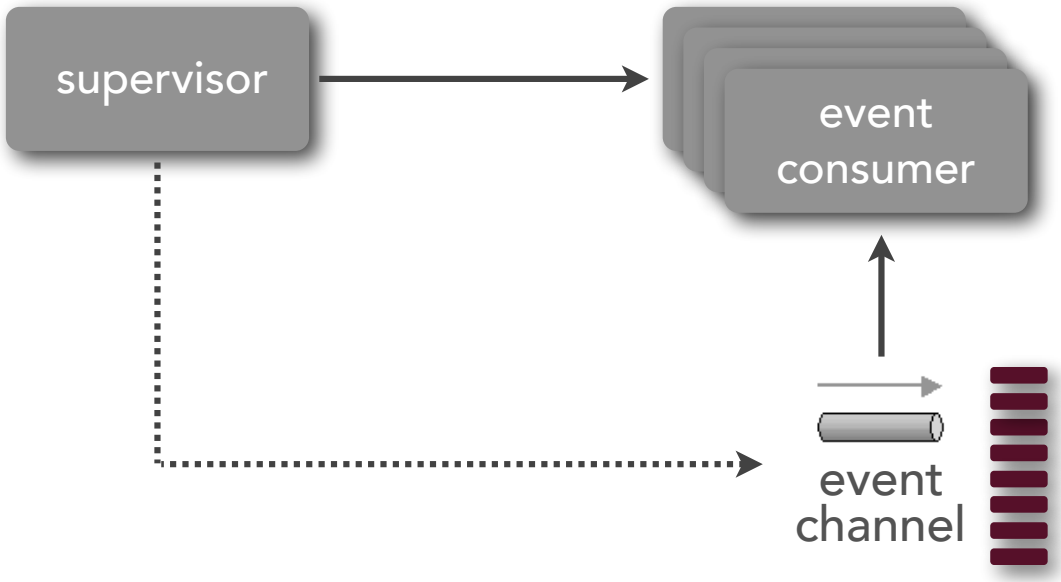
```
private List<MyConsumer> consumers =
    new ArrayList<MyConsumer>();

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

consumer supervisor pattern

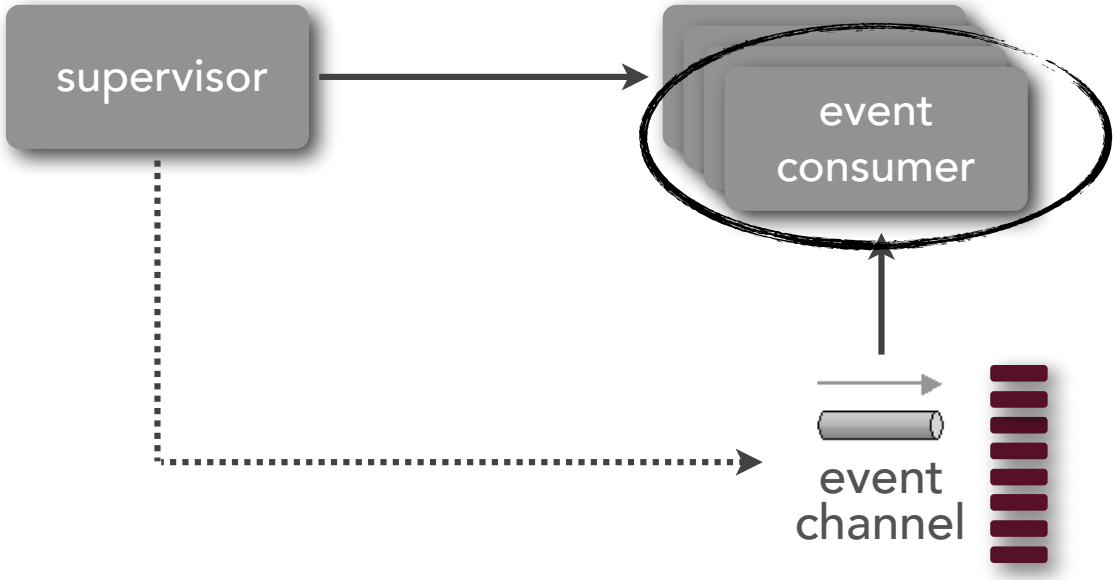
supervisor

```
 private void startConsumer() {
    MyConsumer consumer = new MyConsumer();
    consumers.add(consumer);
    new Thread(()->consumer.startup(connection)).start();
 }

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

consumer supervisor pattern



consumer supervisor pattern

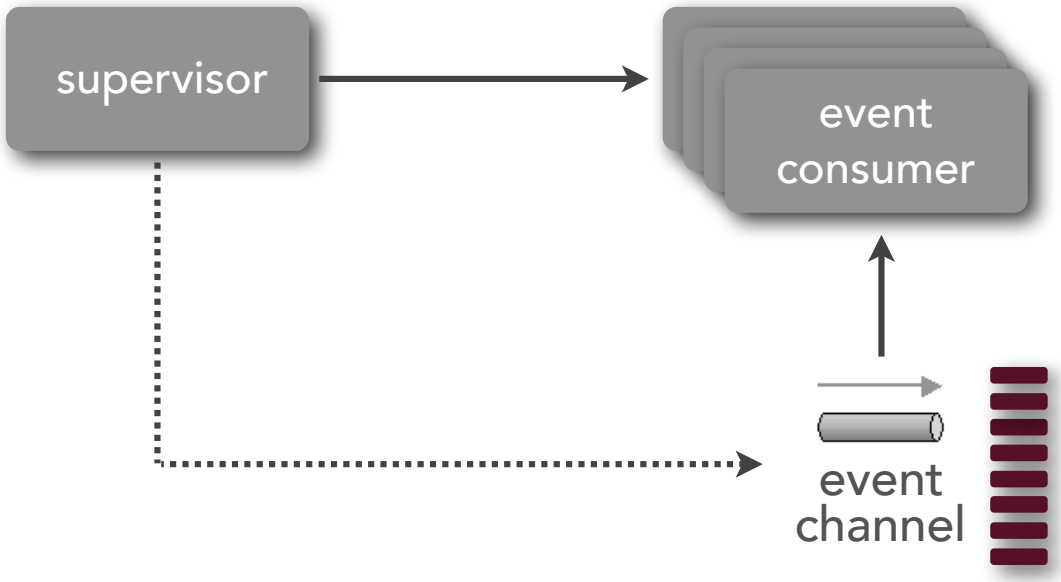


consumer supervisor pattern

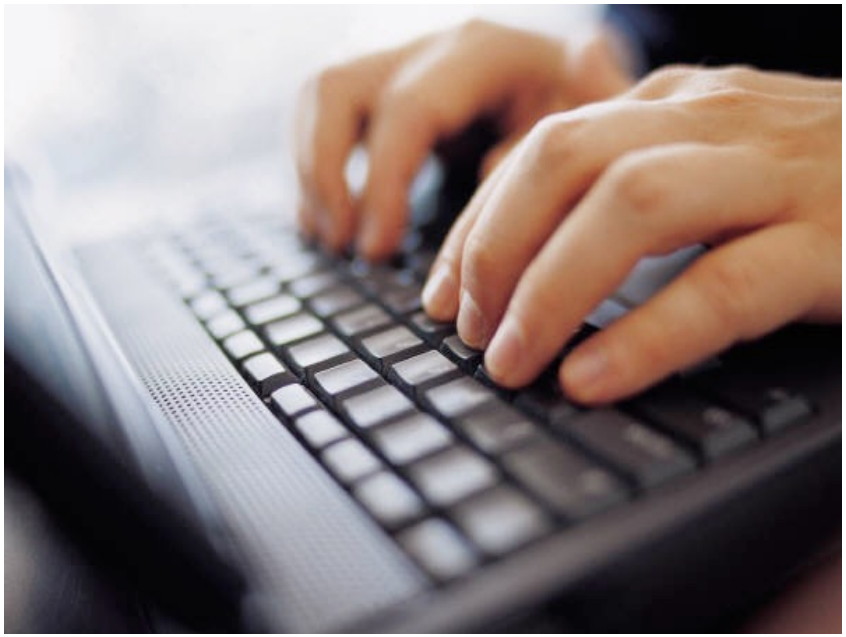
event consumer

```
private Boolean active = true;

public void startup(Connection connection) {
   //get broker session or channel and create consumer
   while (active) {
      msg = getNextMessageFromQueue(10000);
      if (msg != null)
        //process message
   }
   //close broker session or channel
}

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

consumer supervisor pattern

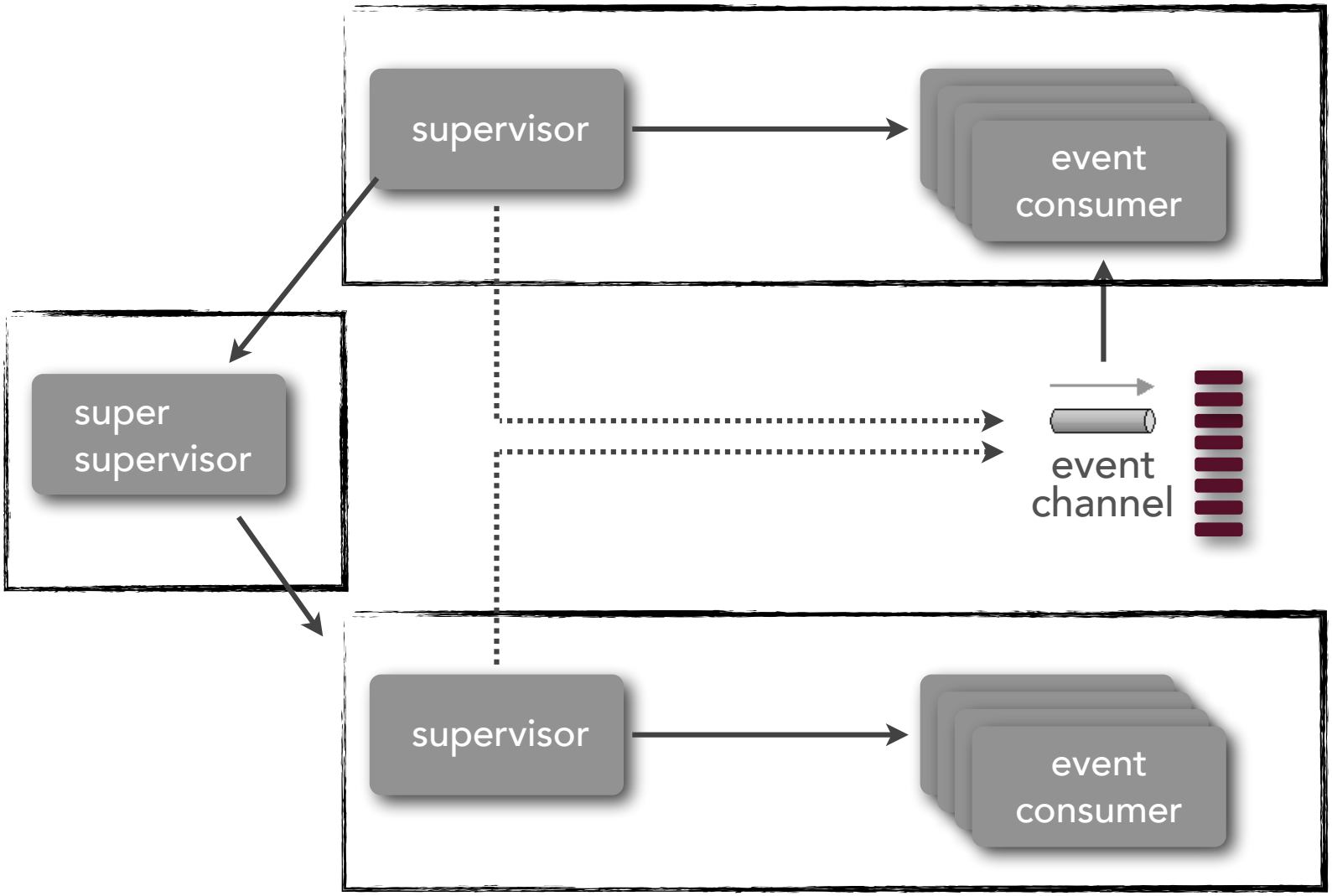


consumer supervisor pattern



let’s see the result…

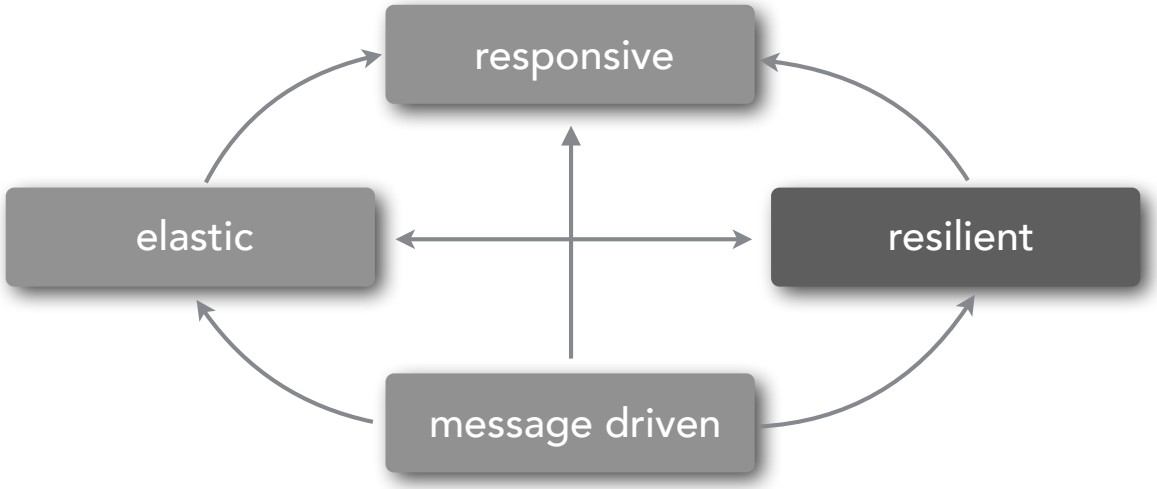
consumer supervisor pattern



Workflow Event Pattern

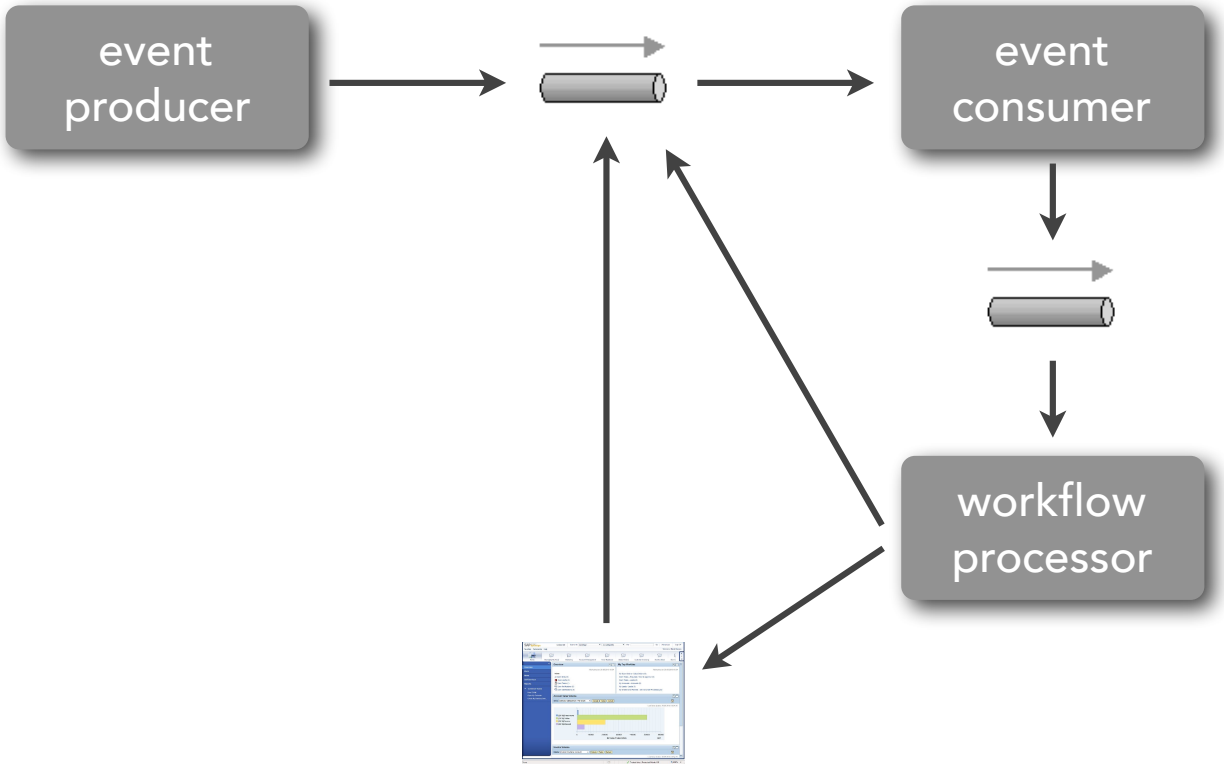
workflow event pattern

how can you handle error conditions without failing the transaction?



workflow event pattern

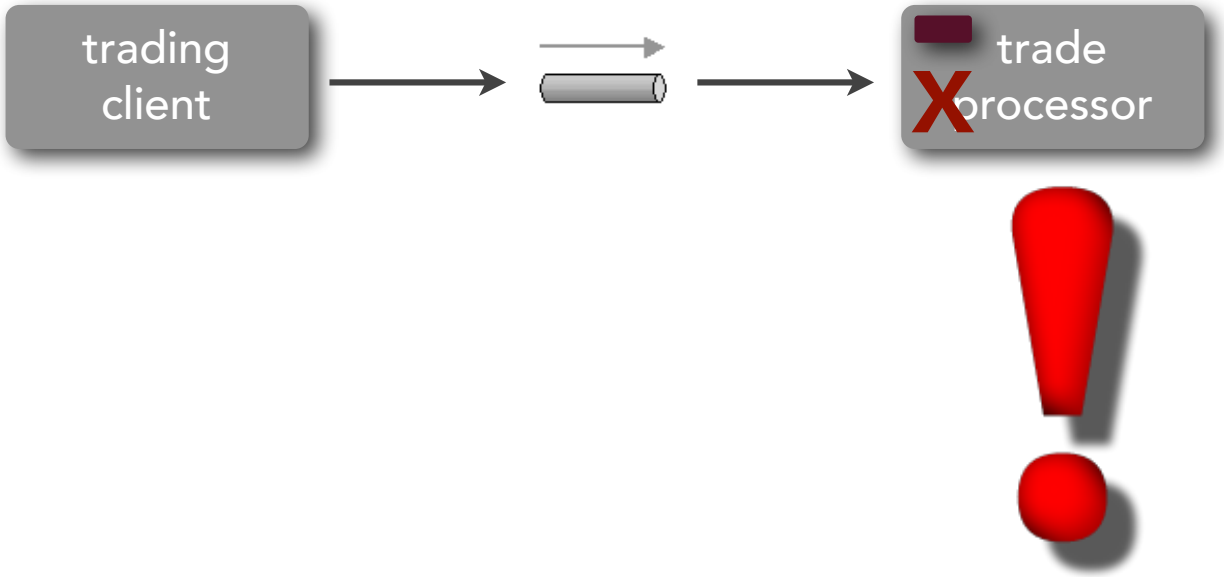
how can you handle error conditions without failing the transaction?



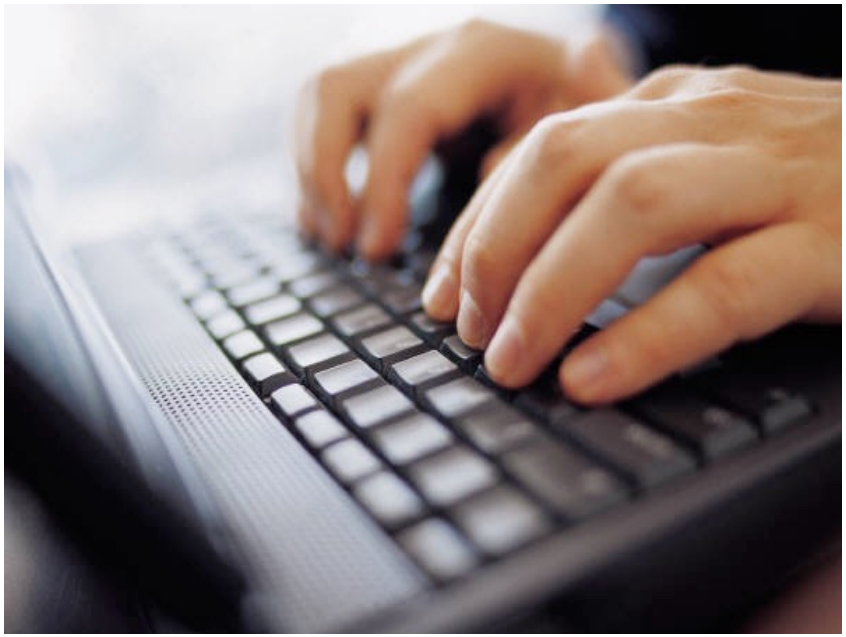
workflow event pattern

example

while asynchronously processing trades an error occurs with one of the trade orders



workflow event pattern

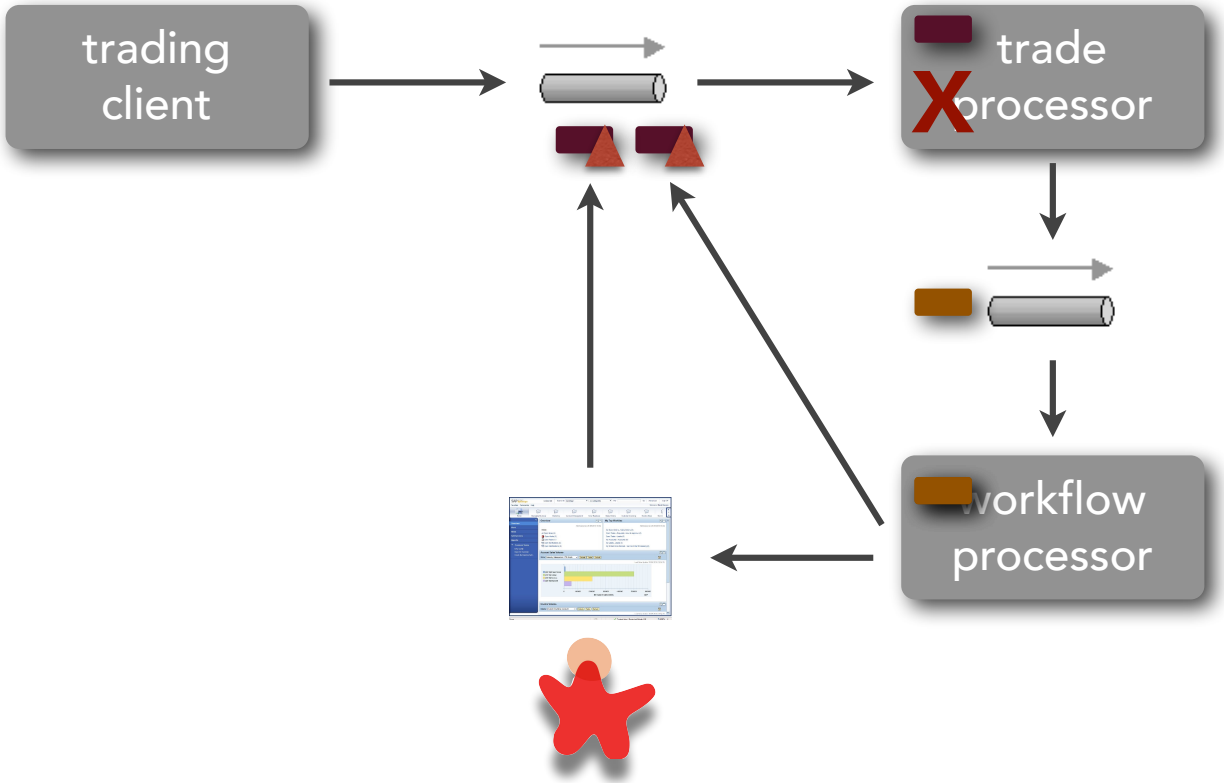


let's see the issue…

workflow event pattern

example

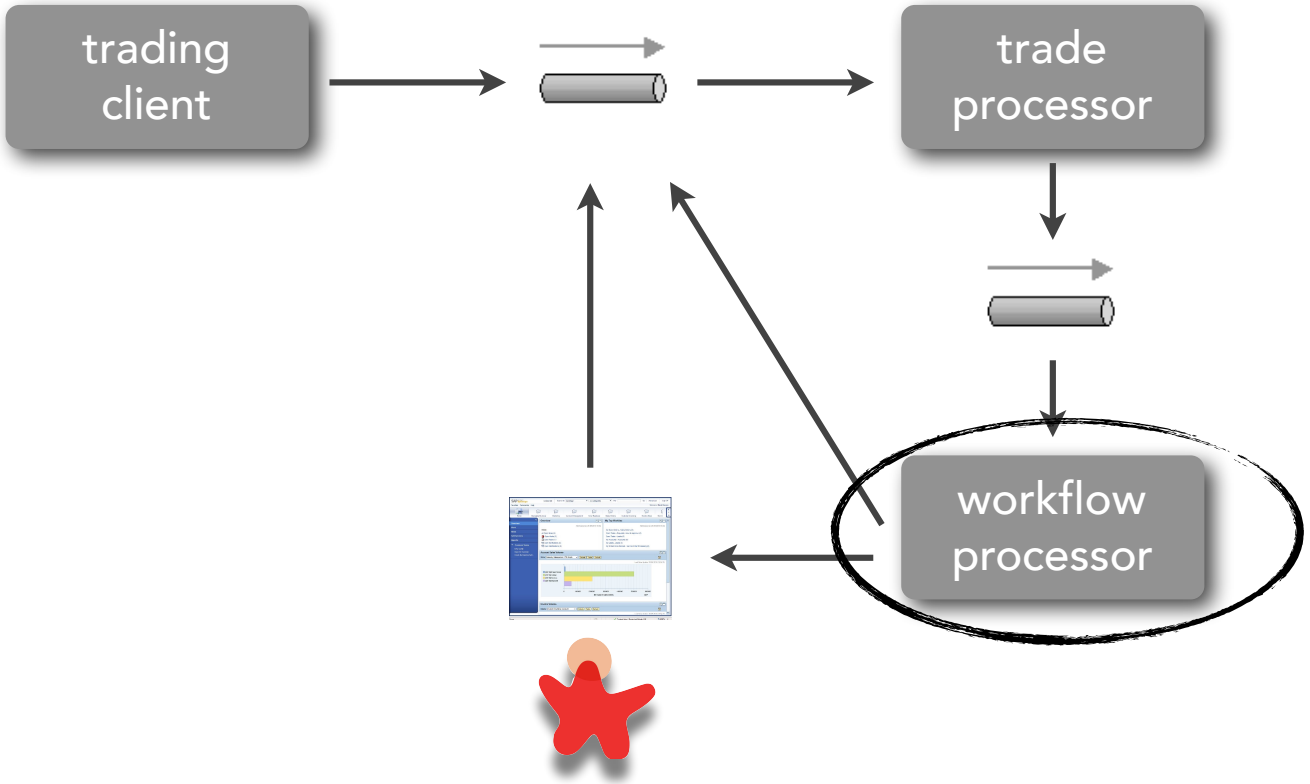
while asynchronously processing trades an error occurs with one of the trade orders



workflow event pattern

example

while asynchronously processing trades an error occurs with one of the trade orders



workflow event pattern

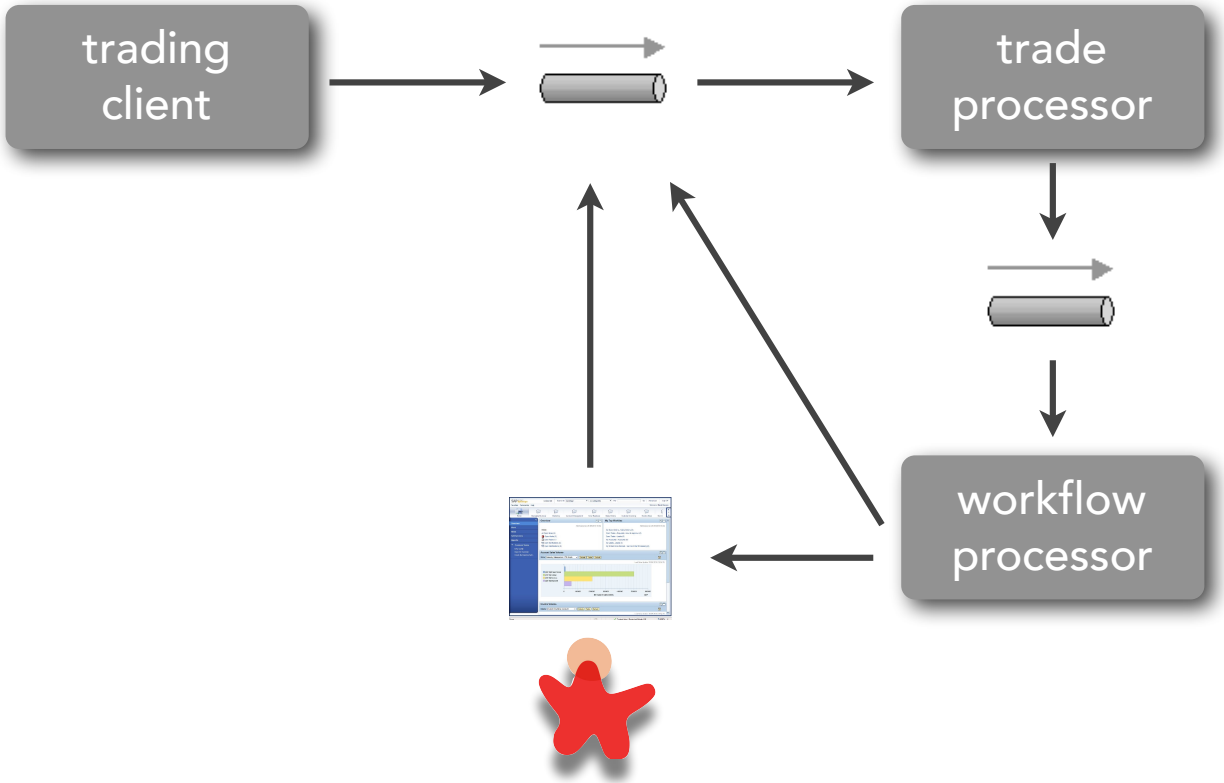
workflow processor

```
msg = getNextMessageFromQueue();
//detect the error type and if it can be repaired
if (SHARES_ERROR) {
   String msg = new String(message.getBody());
   String newMsg = msg.substring(0, msg.indexOf(" shares"));
   System.out.println("Trade fixed: " + newMsg);
   //send new message back to original request queue
} else {
  //send message to dashboard for manual fixing
}
```

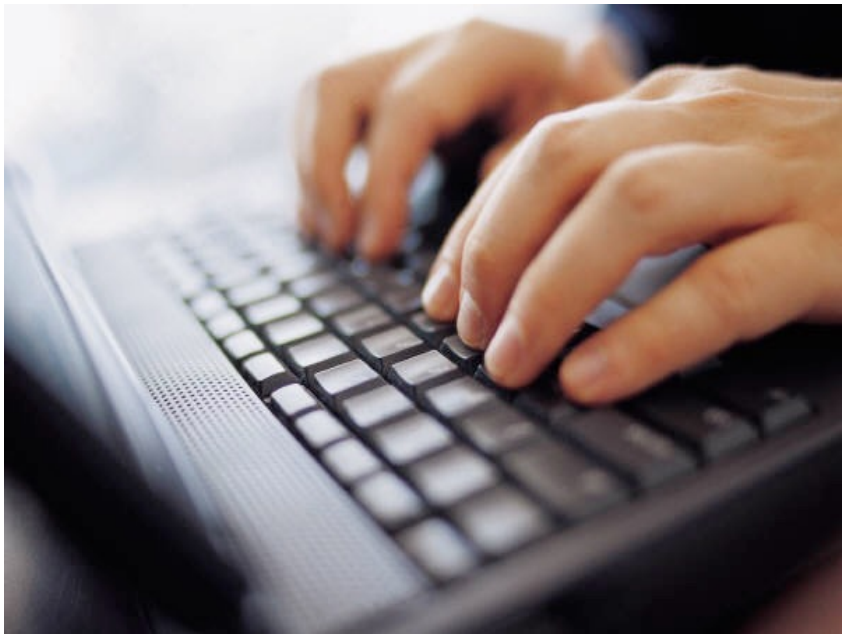
workflow event pattern

example

while asynchronously processing trades an error occurs with one of the trade orders



workflow event pattern



let’s see the result…

Reactive Architecture Patterns 1

Mark Richards
Independent Consultant
Hands-on Software Architect / Published Author
Founder, DeveloperToArchitect.com
www.wmrichards.com

Author of *Software Architecture Fundamentals Video Series* (O'Reilly)
Author of Microservices Pitfalls and AntiPatterns (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)


GREAT INDIAN
DEVELOPER™
SUMMIT 2019
Conference : April 23-26, Bangalore
12th EDITION
#gids19
Register early and get the best discounts!
www.developersummit.com
@greatindiandev
bit.ly/gidslinkedin
facebook.com/gids19
bit.ly/saltmarchyoutube
flickr.com/photos/saltmarch/