Alfresco Summit 2014

Ixxus

Activiti in an Event-Driven Architecture

Robin Bramley
Chief Scientific Officer, Ixxus

We are a leading global provider of end-to-end, custom-built content solutions.

Vital Statistics

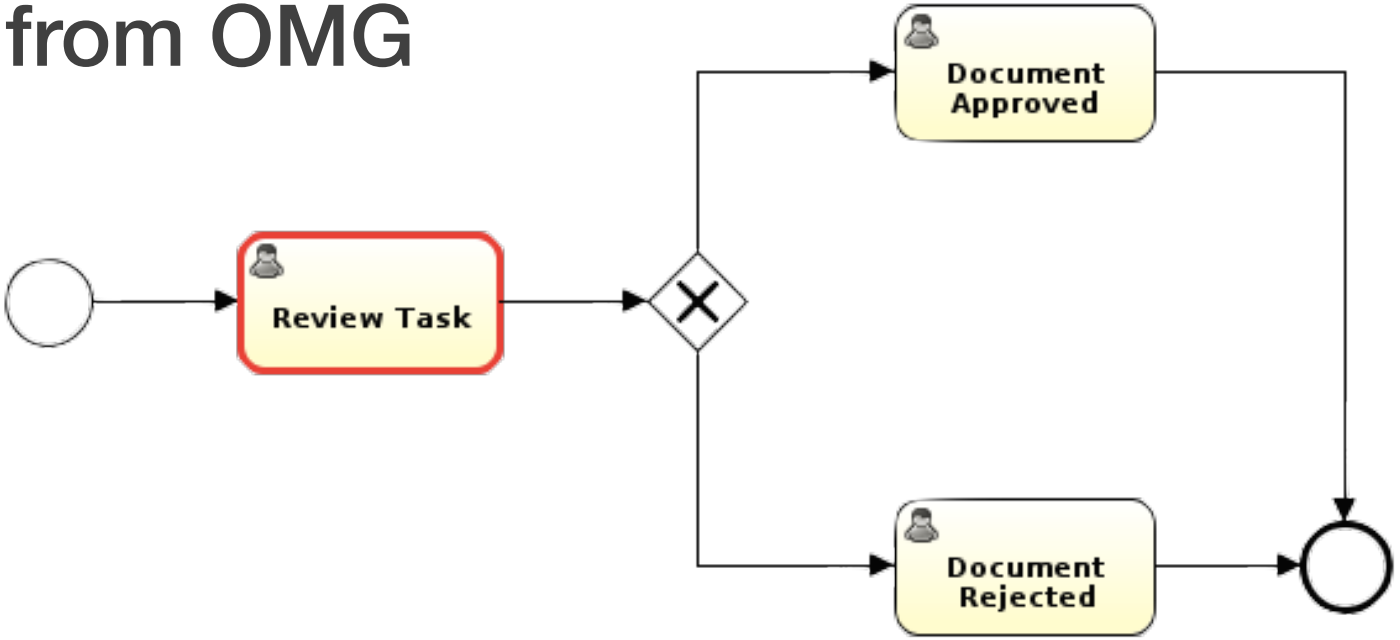
- Global Platinum Partner
- Working with Alfresco since v0.6
- Most certified consultants globally
- Business Solution of the Year 2014
- Alfresco US Deal of the Year 2014
- Alfresco Solutions Partner 2013
- Alfresco Million $ Club 2012
- Alfresco Dashlet Challenge 2012

Presented at:

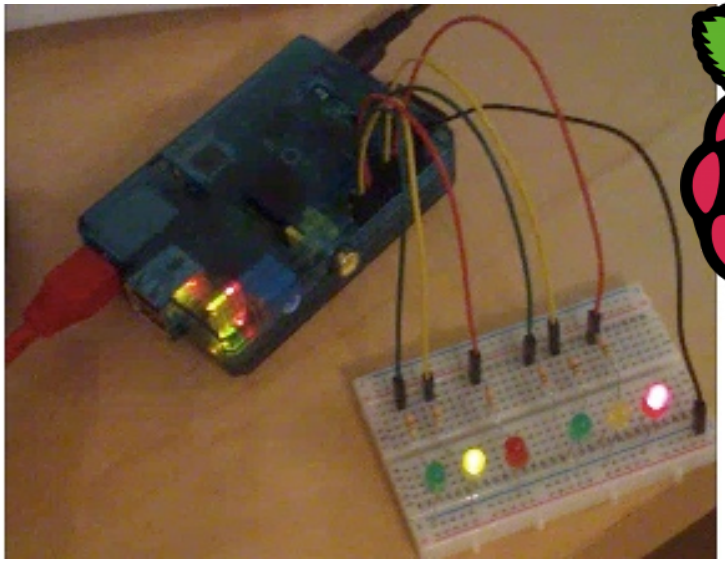
Published in:

Activiti™

- Open Source project by Alfresco
- Lightweight Workflow Engine
- Implements BPMN2.0 open standard from OMG



Activiti at scale^{-1}



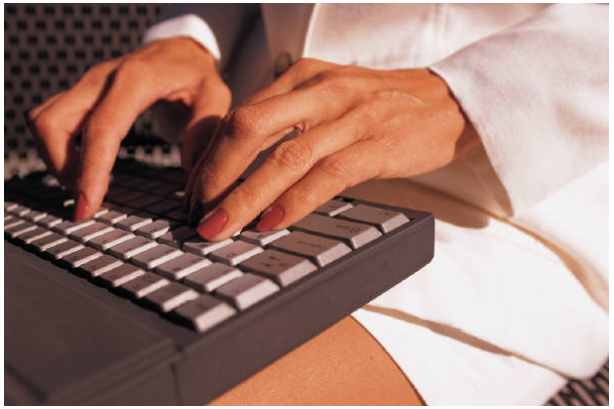
Powered by MySQL

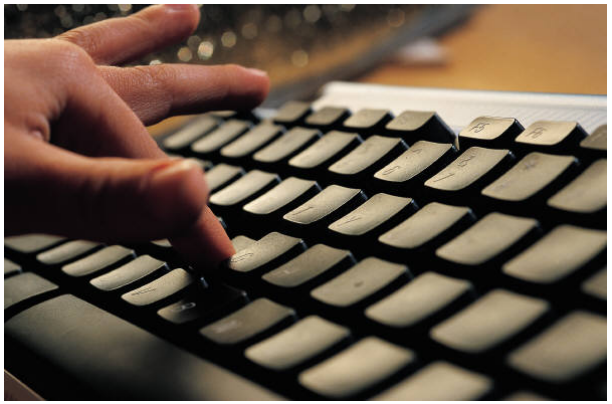
Activiti

http://www.jorambarrez.be/blog/2013/02/15/activiti-runs-on-a-raspberry-pi-model-b-with-9mb-heap/

Double entry considered harmful

- *Increases risk of conflict*





Integration is essential…

… in more connected times

Health warning

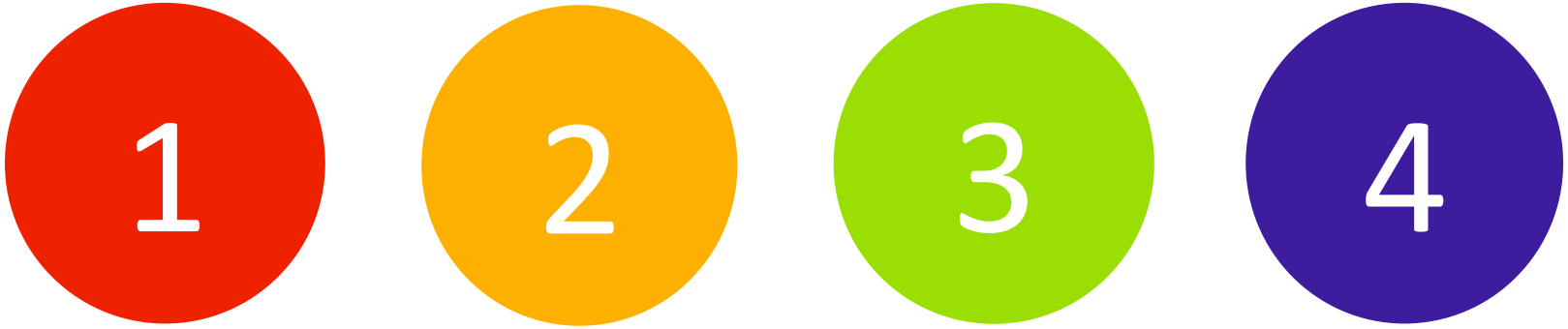
The fallacies of distributed computing

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

For more details, read the article by Arnon Rotem-Gal-Oz

Integration styles

4 styles of integration



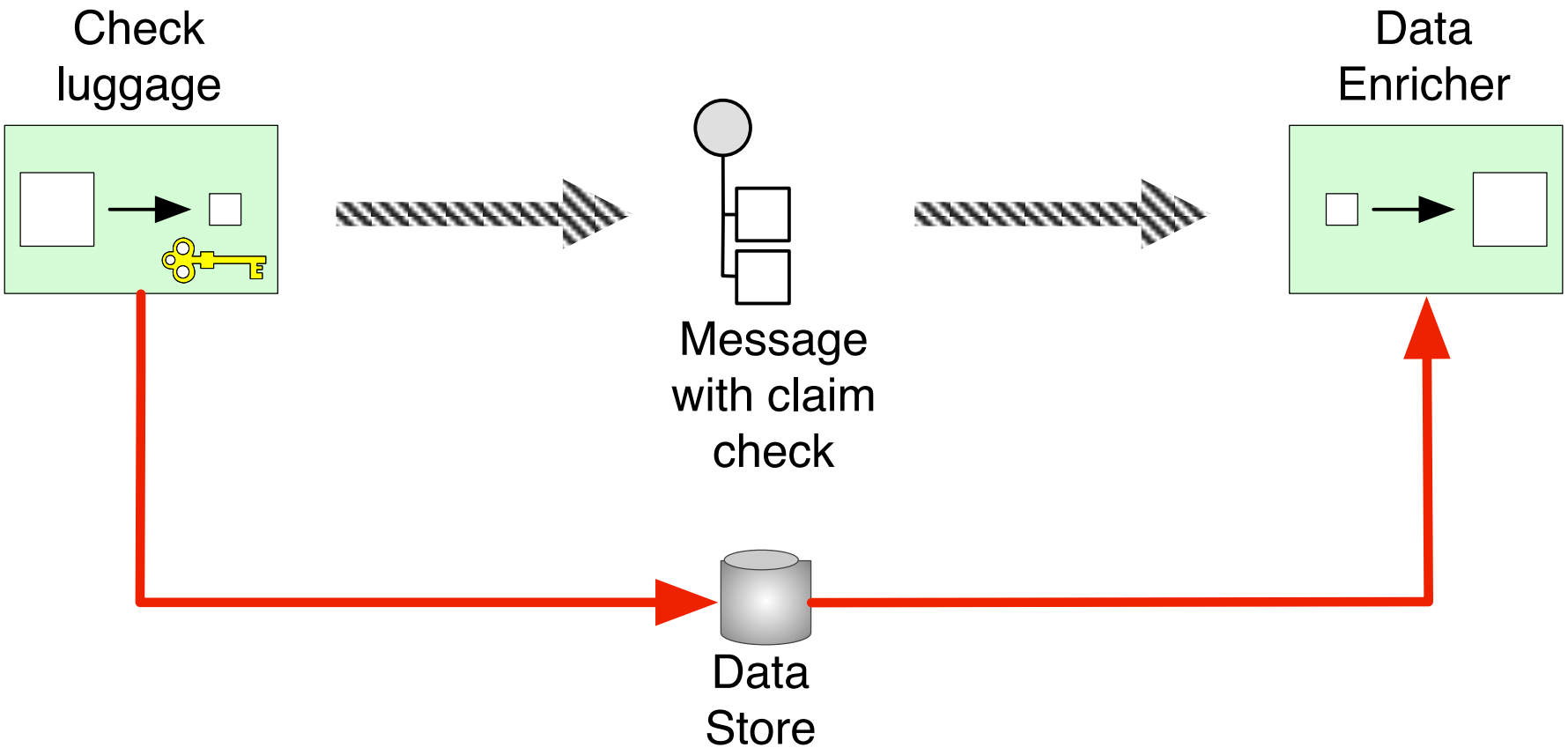
Messaging

4

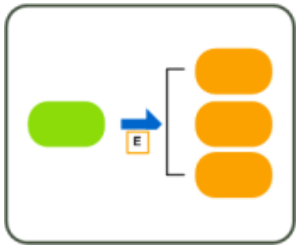
- Asynchronous messaging is inherently loosely coupled and helps to overcome latency & unreliability issues
- Can apply ‘pipes & filters’ for flexibility
- Frameworks mask a lot of the complexity from application developers

Enterprise Integration Patterns

EAI Patterns



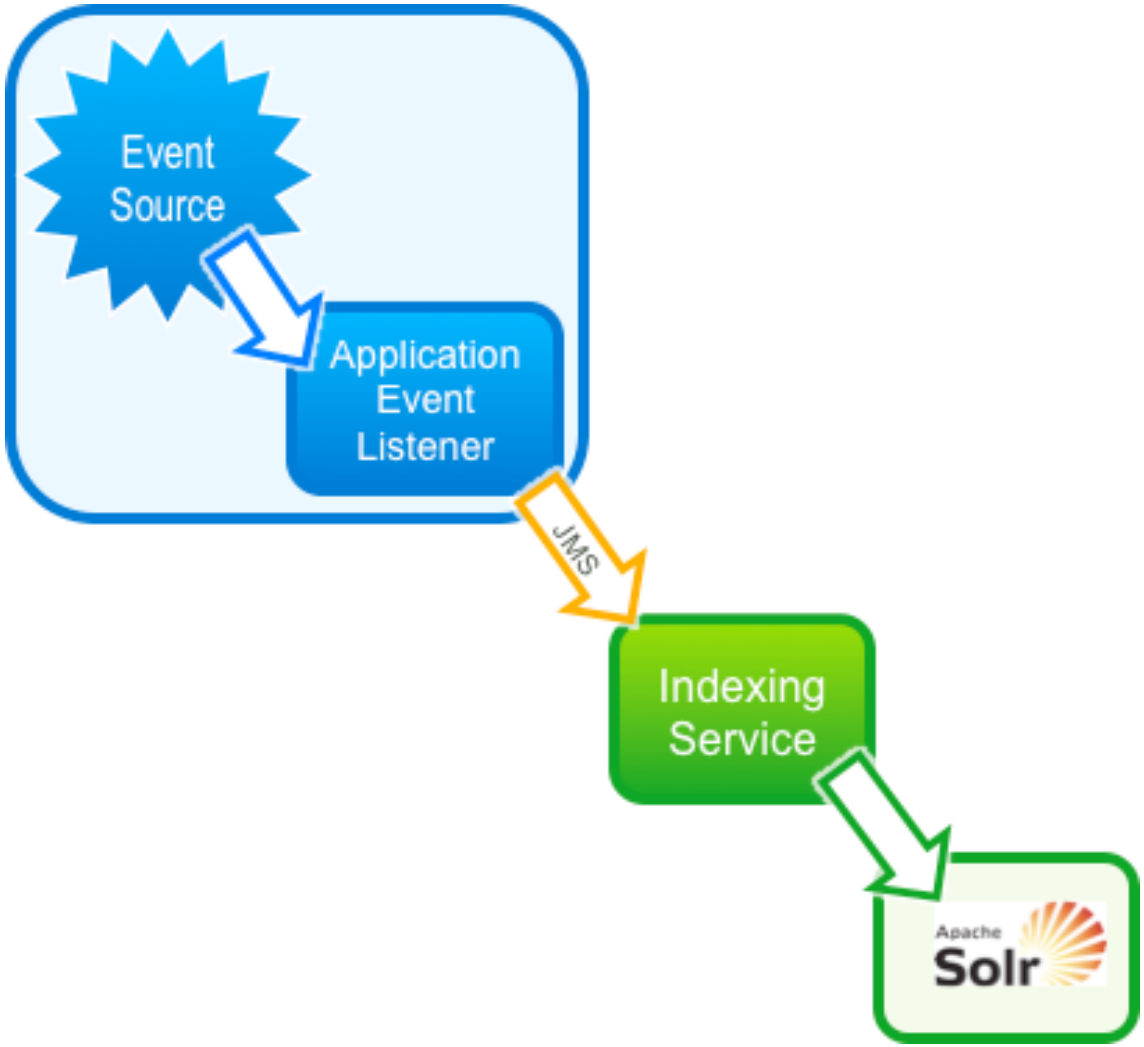
Event-Driven Architecture



- Loosely integrating systems using the observer pattern
- Systems emit events when something happens
- Systems listen for events that they are interested in and take action when they are triggered
- Emitting applications are unaware of which applications are listening to its events – good for extensibility

Emitting events

Events in Spring



Event Publishing in Spring

```
public class Xyz implements ApplicationEventPublisherAware {
```

```
private ApplicationEventPublisher applicationEventPublisher;

public setApplicationEventPublisher(ApplicationEventPublisher
aep) {
    applicationEventPublisher = aep;
}
```

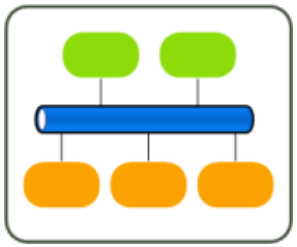
```
...
public create(Map<String,String> foo) {
   ...
   XyzCreatedEvent evt = new XyzCreatedEvent(foo);
   applicationEventPublisher.publishEvent(evt);
```

Spring Event Listener

```
public class IndexingListener implements ApplicationListener {
    public onApplicationEvent(ApplicationEvent event) {
        if (event instanceof XyzCreatedEvent) {
                    XyzCreatedEvent evt = (XyzCreatedEvent) evt;
                    ...
                    // handle created event
        } else if (event instanceof XyzDeletedEvent) {
                    ...
```

https://www.flickr.com/photos/jadawin42/12403799463

Middleware

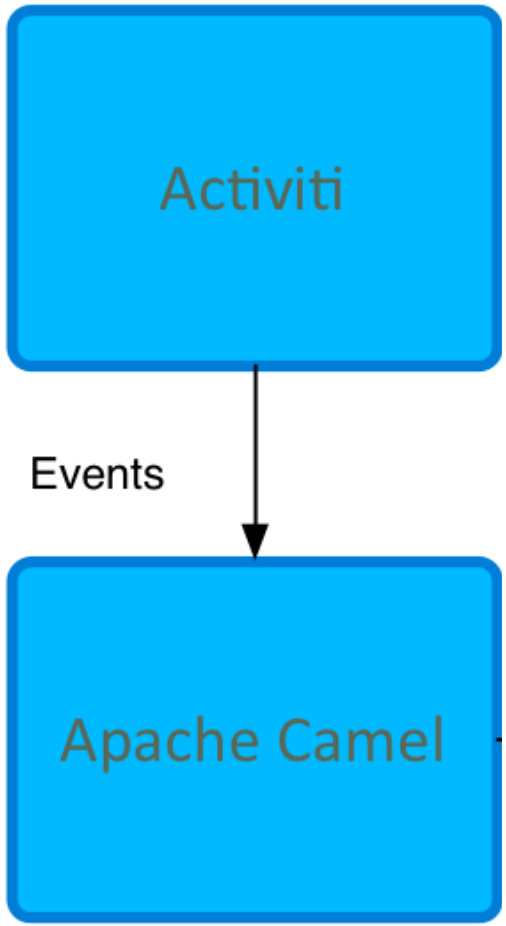


- **Middleware tools aim to simplify integration & minimise point-to-point *'spaghetti'***
- **Provide messaging services**
  - E.g. Transformation, Routing, Augmentation, Notification
- **Support multiple protocols**
  - E.g. JMS, FTP, HTTP

Lightweight middleware

- The centralised Enterprise Service Bus (ESB) model has been criticised for being too monolithic
- Lighter-weight options exist

Simple Overview (1)



Propagating events from Activiti

Service Task

- Additional process config
- Extra Activiti dependency
- No custom code

Events (5.15+)

- Have to write an `ActivitiEventListener` to produce a message (for a Camel route)
- Easier to retrofit if you have lots of process definitions!

Camel Service Task

```
<process id="activitiAdhoc" isExecutable="true">
  <startEvent id="theStart" />
  <sequenceFlow id="flow1" sourceRef="theStart"
targetRef="processStarted" />

  <serviceTask id="processStarted" activiti:type="camel" />
  <sequenceFlow id="flow2" sourceRef="processStarted"
targetRef="task" />

  <userTask id="task" name="User Task"/>
  <sequenceFlow id="flow3" sourceRef="task" targetRef="processEnded" />

  <serviceTask id="processEnded" activiti:type="camel" />
  <sequenceFlow id="flow4" sourceRef="processEnded"
targetRef="theEnd" />

  <endEvent id="theEnd" />
</process>
```

Camel Service Task


```
<route>
  <from uri="activiti:activitiAdhoc:processStarted" />
  ...
</route>
```

ActivitiEventListener

```
public class ActivitiListener implements ActivitiEventListener {

  @EndpointInject(uri='activemq:queue:activiti.start')
  ProducerTemplate startProducer

  @Override
  public void onEvent(ActivitiEvent event) {

    switch (event.getType()) {
      case ActivitiEventType.TASK_CREATED:
        startProducer.sendBody(new StartEvent())
        break
```

Process – take 2

```
<process id="activitiAdhoc" isExecutable="true">
  <startEvent id="theStart" />
  <sequenceFlow id="flow1" sourceRef="theStart" targetRef=”task" />

  <userTask id="task" name="User Task"/>
  <sequenceFlow id="flow2" sourceRef="task" targetRef=”theEnd" />

  <endEvent id="theEnd" />
</process>
```

Activiti configuration

```
processEngineConfiguration(
 org.activiti.spring.SpringProcessEngineConfiguration) {

  deploymentResources
    = "file:./grails-app/conf/processes/*.bpmn20.xml”

  eventListeners = [activitiListener]

  ...
}
```

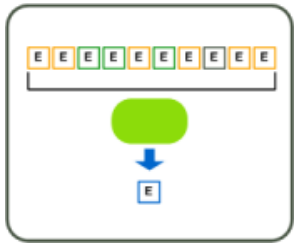
Starting the process

```
ProcessInstance processInstance =
  runtimeService.startProcessInstanceByKey("adhoc")

String processId = processInstance.getProcessInstanceId()

Task task =
  taskService.createTaskQuery()
    .executionId(processId).singleResult()
```

Demo

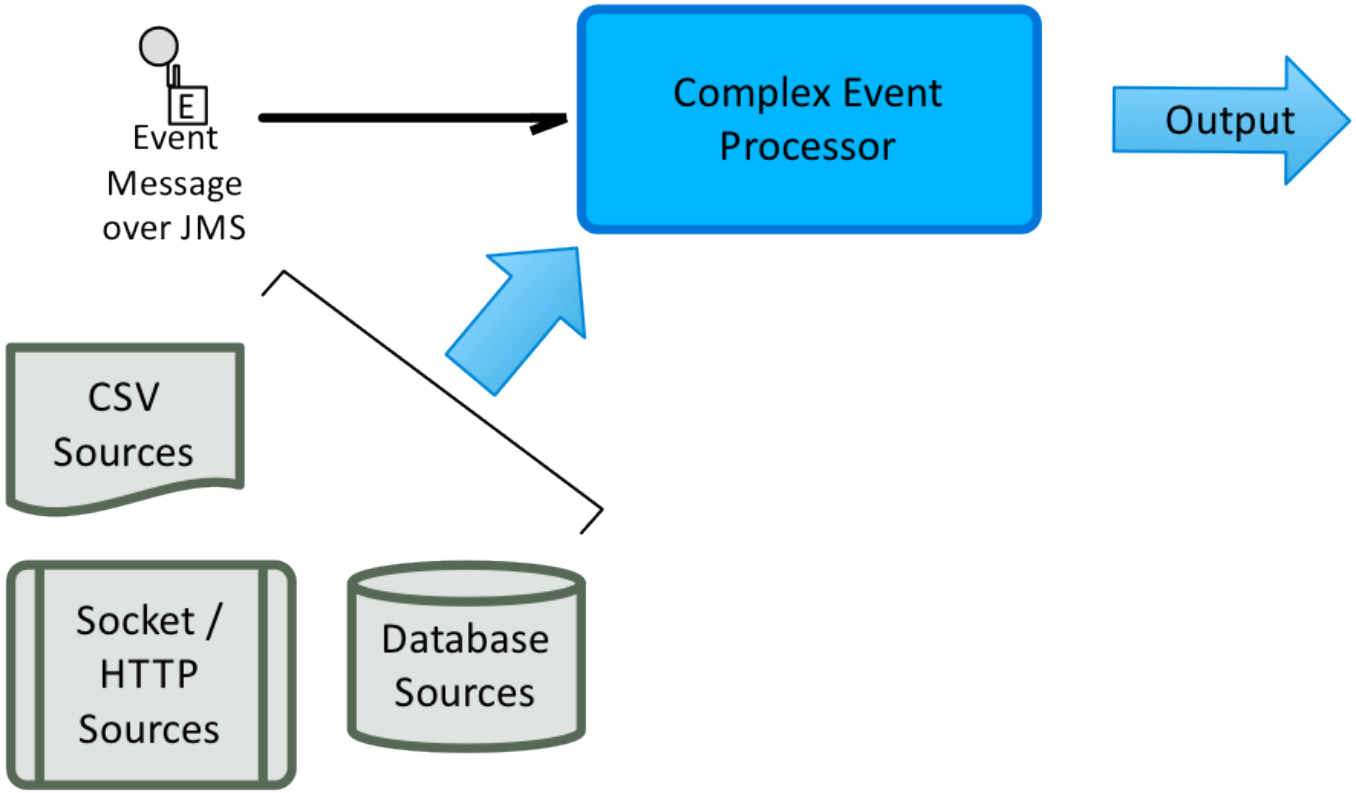
Event-driven Analytics

Complex Event Processing



- Complex Event Processing involves monitoring an event stream to identify activity patterns or correlations (within time windows)
- Used in Financial Services trading systems for fraud detection
- Other uses include billing/metering, delayed shipment with RFID tracking

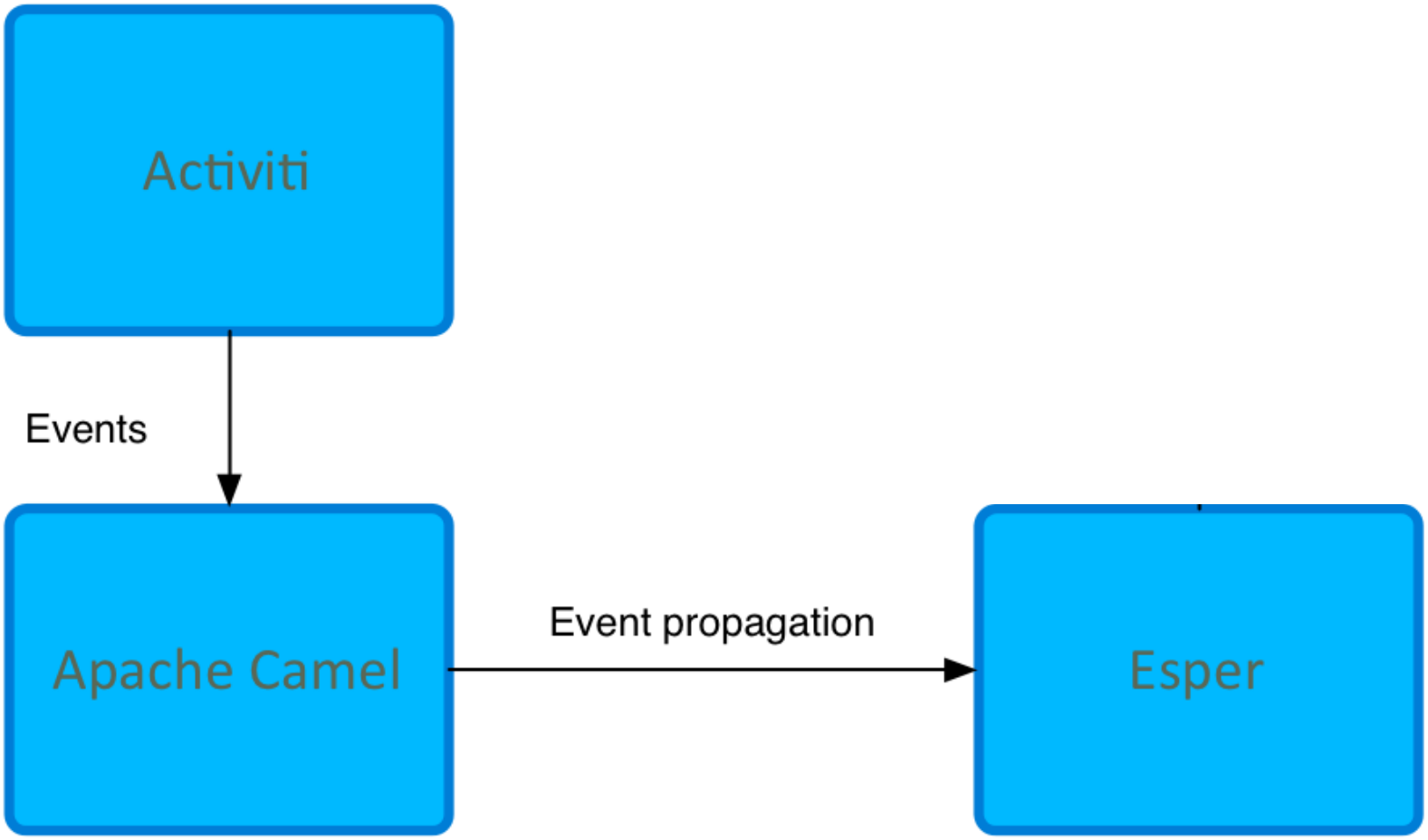
Event Processing



Esper

- An Open Source Java component for complex event processing (CEP) and event series analysis
- Event Processing Language (EPL) – declarative language for dealing with high frequency time-based event data
- Supports POJO event representations

Simple Overview (2)



Esper routes in Camel

```
from('activemq:queue:activiti.start')
.to('esper://feed')
```

```
from('esper://feed?eql=create context CtxEachMinute initiated
pattern [every timer:interval(3 sec)] terminated after 30 sec')
.to('esper://feed')
```

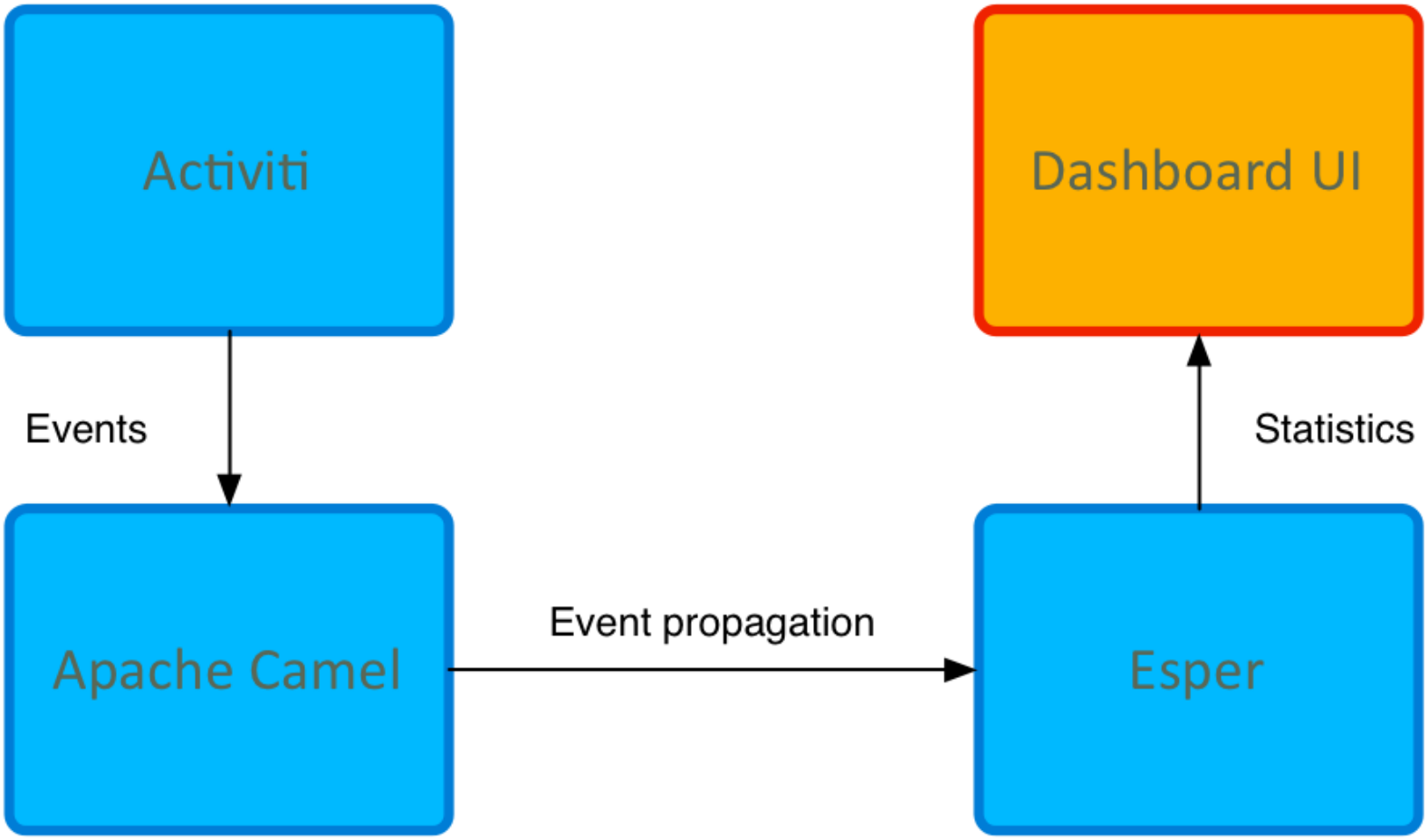
```
from('esper://feed?eql=context CtxEachMinute select count(*) as
countValue from demo.event.StartEvent output snapshot when
terminated')
.process(new Processor() {...} })
.to('activemq:queue:stats.process.started')
```

Demo

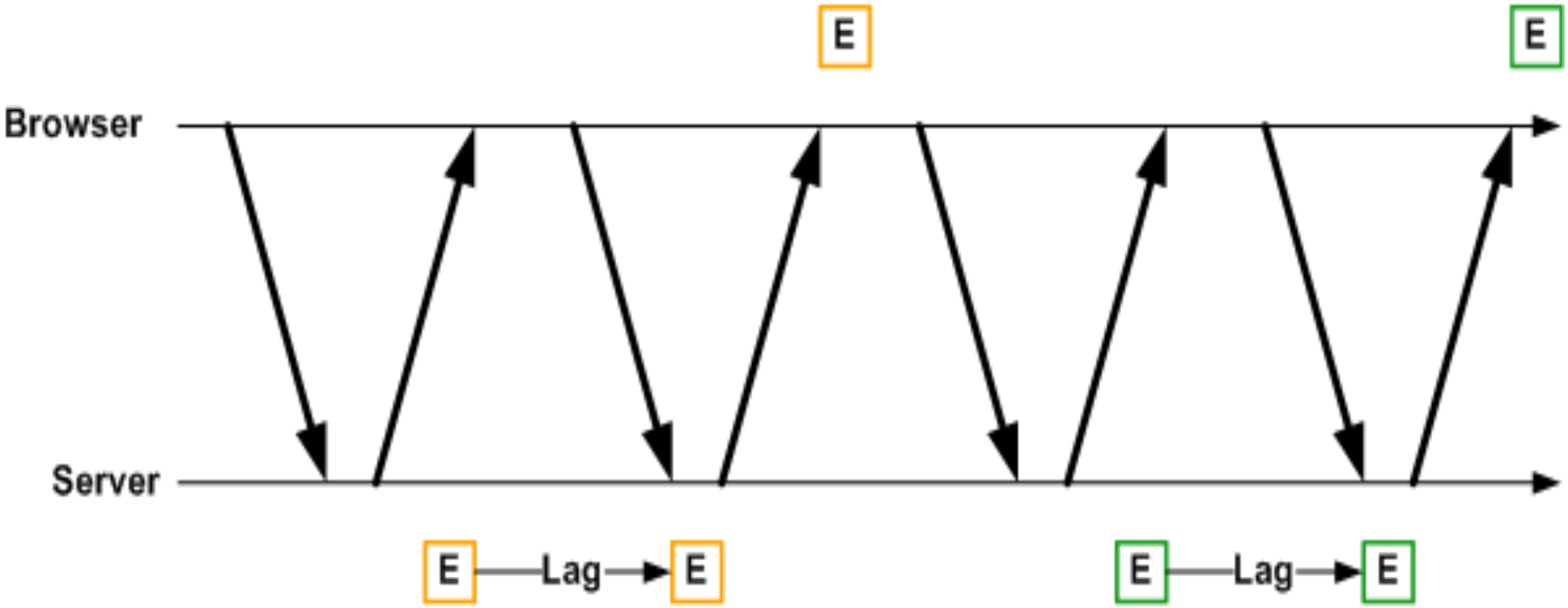
Asynchronous browser updates

https://www.flickr.com/photos/gtweb/11328349694

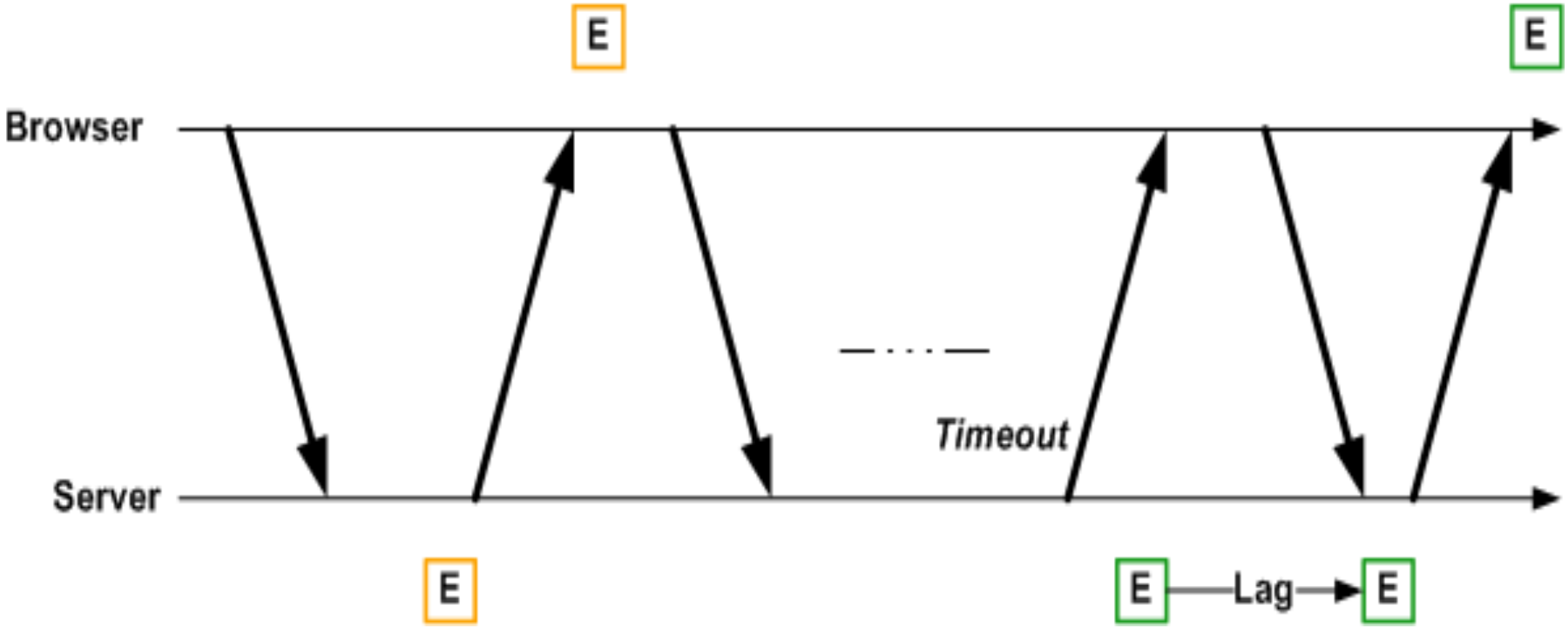
Simple Overview (3)



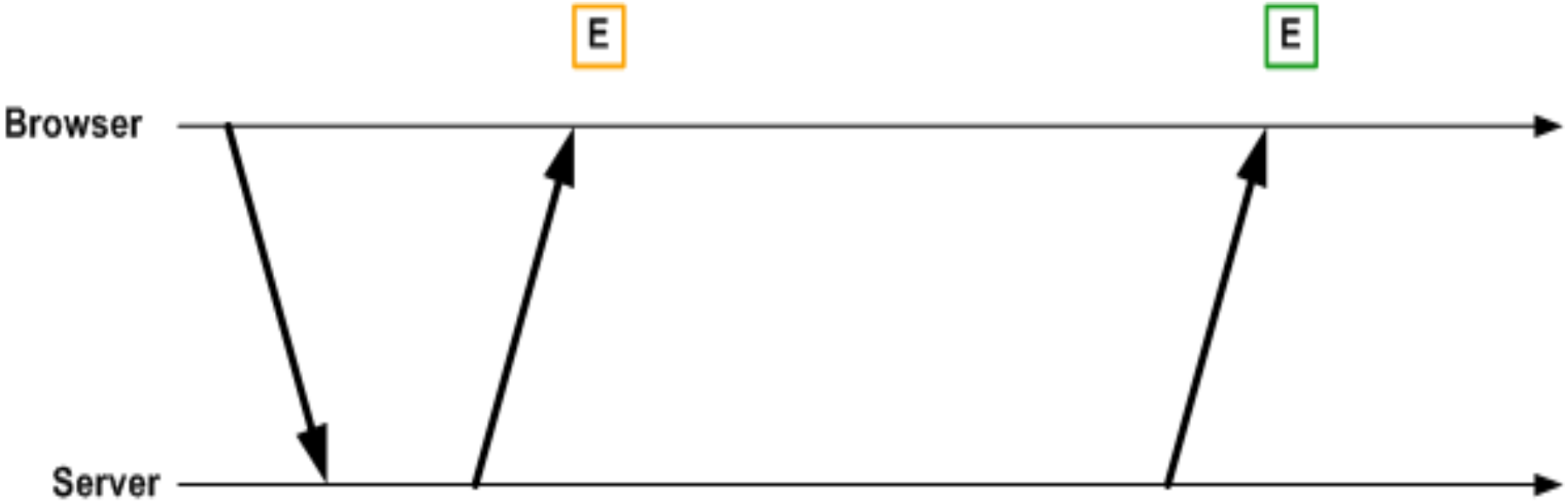
Polling



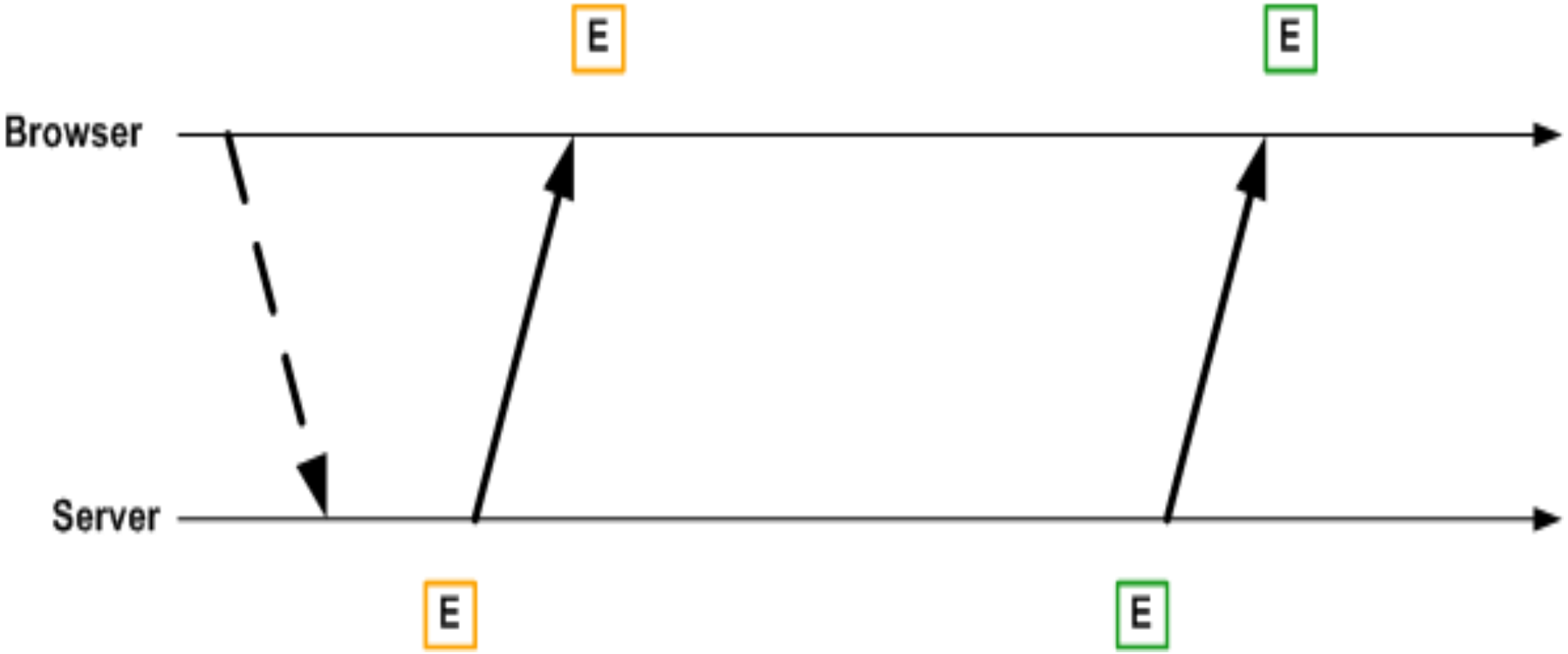
Long polling



Streaming



HTML5 Web Sockets



Atmosphere Meteor

- The Atmosphere Framework is designed to make it easier to build Asynchronous Web applications that include a mix of WebSocket, Comet and REST
- Can be deployed on *any Servlet 2.3+ container*
- Meteor is a high level wrapper used by the Grails Atmosphere Meteor Plugin

Grails Service

```
class DashboardService {
  def atmosphereMeteor

  String mapping = "/atmosphere/notification"

  public void handleStartEvent(String payload) {
    def publicResponse = publicResponse('Started', payload)
    broadcastData(mapping, publicResponse, true)
  }

  def publicResponse(String type, String payload) {
    return [type: "notification", resource: mapping, stats: type,
message: "${type}: ${payload}"] as JSON
  }
```

Camel Route

```
from('activemq:queue:stats.process.started')
.beanRef('dashboardService','handleStartEvent')
```

JavaScript Subscription

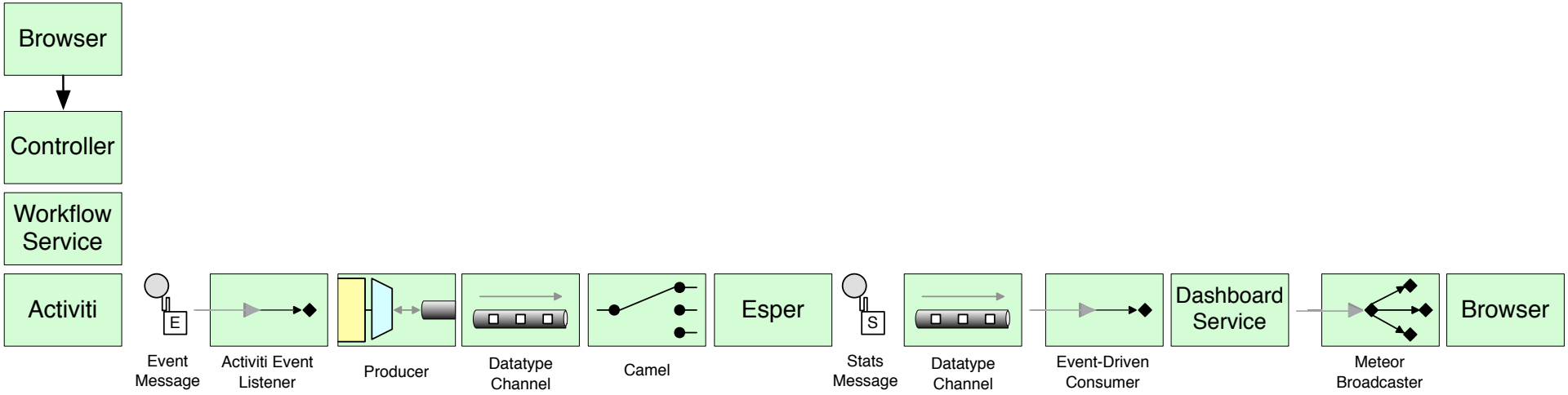
```
subscribe: function (options) {
  var defaults = {
    contentType: "application/json",
    transport: 'websocket',
    fallbackTransport: 'long-polling',
    ...
  },
  atmosphereRequest = $.extend({}, defaults, options);
  switch (options.type) {
    case 'notification':
      Jabber.notificationSubscription =
        Jabber.socket.subscribe(atmosphereRequest);
      break;
```

JavaScript Message Handler

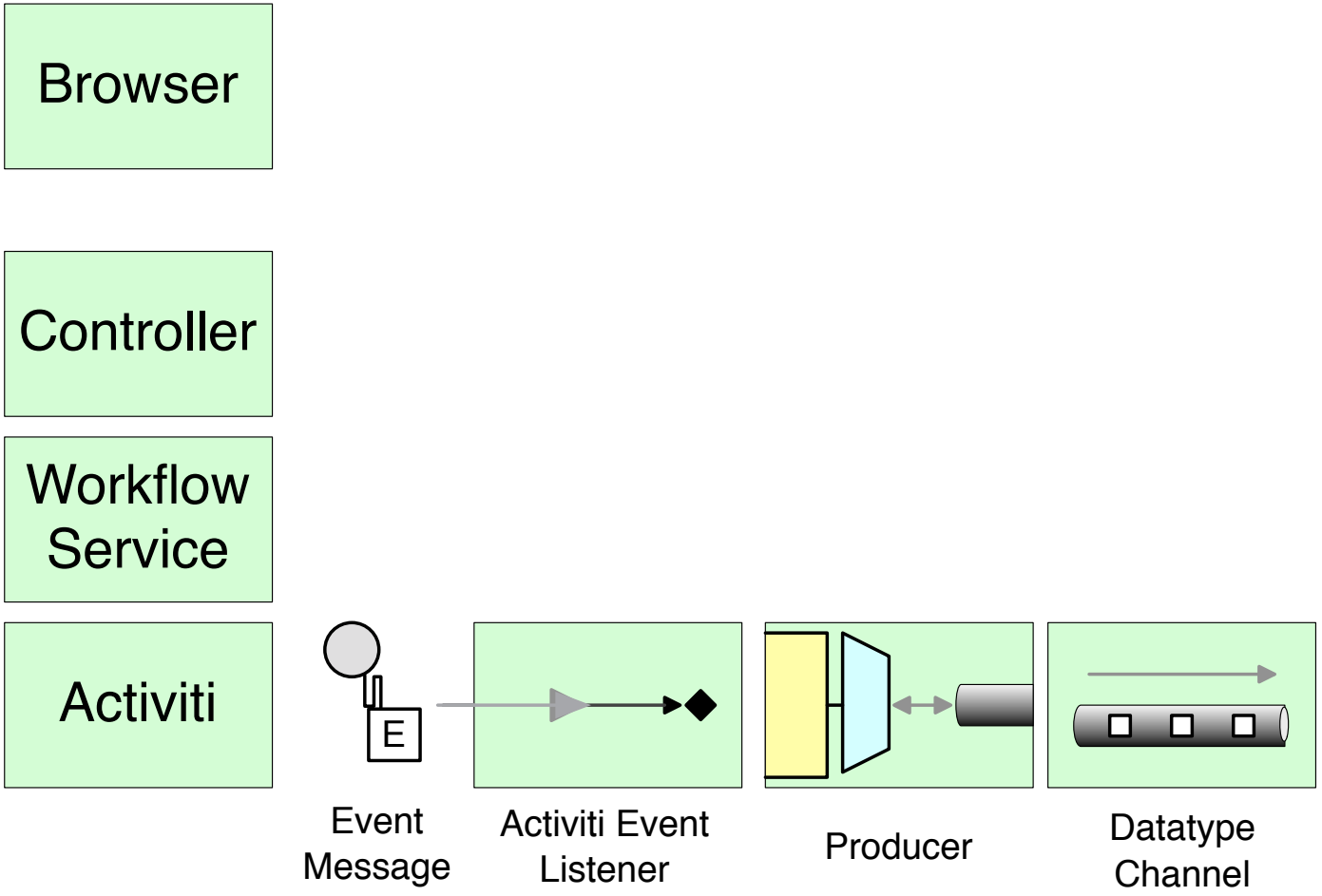
```
onMessage: function (response) {
  var data = response.responseBody;
  if ((data == '')) {
    return;
  }
  console.log(data);
  var message = JSON.parse(data);
  var type = message.type;
  if (type == 'notification') {
    $('#notification').html(message.message);
  }
};
```

Demo

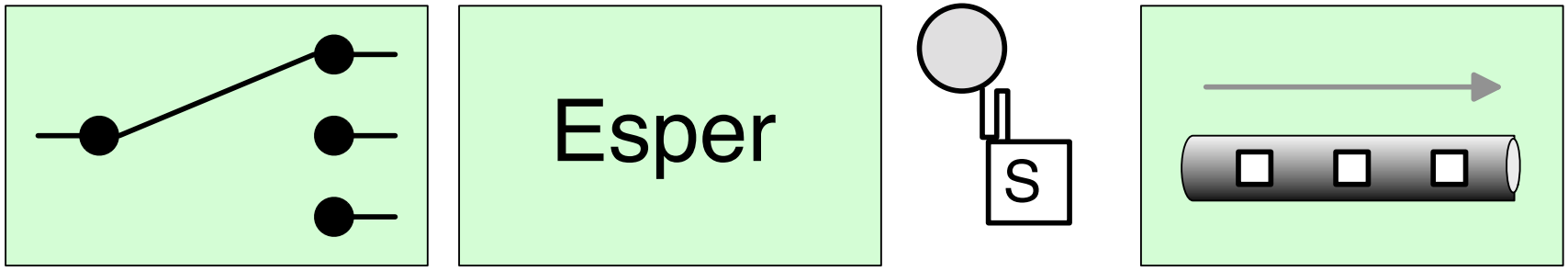
Recap in EIP notation



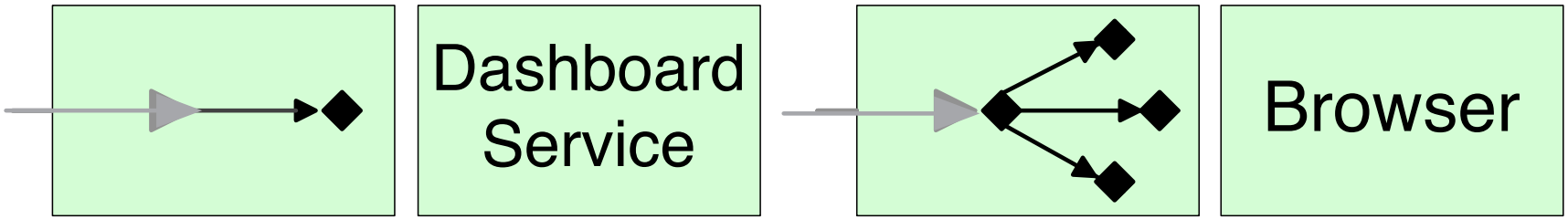
Part 1


Browser
Controller
Workflow Service
Activiti
E
Event Message
Activiti Event Listener
Producer
Datatype Channel

Part 2



Part 3


Dashboard Service
Browser
Event-Driven Consumer
Meteor Broadcaster

Questions

Resources

- http://www.ixxus.com
- Hohpe & Woolf Enterprise Integration Patterns
- http://camel.apache.org
- http://www.activiti.org/userguide/#bpmnCamelTask
- http://www.activiti.org/userguide/#eventDispatcher
- http://esper.codehaus.org
- http://camel.apache.org/esper.html
- https://github.com/Atmosphere/atmosphere/wiki/Understanding-Meteor
- http://grails.org/plugin/atmosphere-meteor