



Building eventing system for microservices architecture



Yaroslav Tkachenko
@sap1ens

Director of Engineering, Platform at Bench Accounting



Agenda

- Context
- Events & event sourcing
- High-level architecture
- Schema & persistence

Bench^π





Context

3 types of events:

- Application
 - Notifications
 - TODO items
 - [Messages]
- System
- Stats

Context - TODO

		INBOX STATE	DOCUMENTS	STATEMENTS	CHANGES	RAC \$ #	EAC \$ #	
ALPHA	MANUAL	FILTERING		0 1573	828	767 0	\$369k 110	\$1M 11k
A			Keep Up	0 415	100	0 0	0 0	234K 112
B								
C			Keep Up	0 21	8	0 0	16K 4	1K 10
D								
E			Keep Up	0 37	6	2 0	0 0	0 0
F								
G			Keep Up	0 0	0	0 0	0 0	0 0
H								
I			Keep Up	0 12	0	0 0	0 0	333 4
J								



Context - Notifications

19

Notifications

17

New processed statements available

×

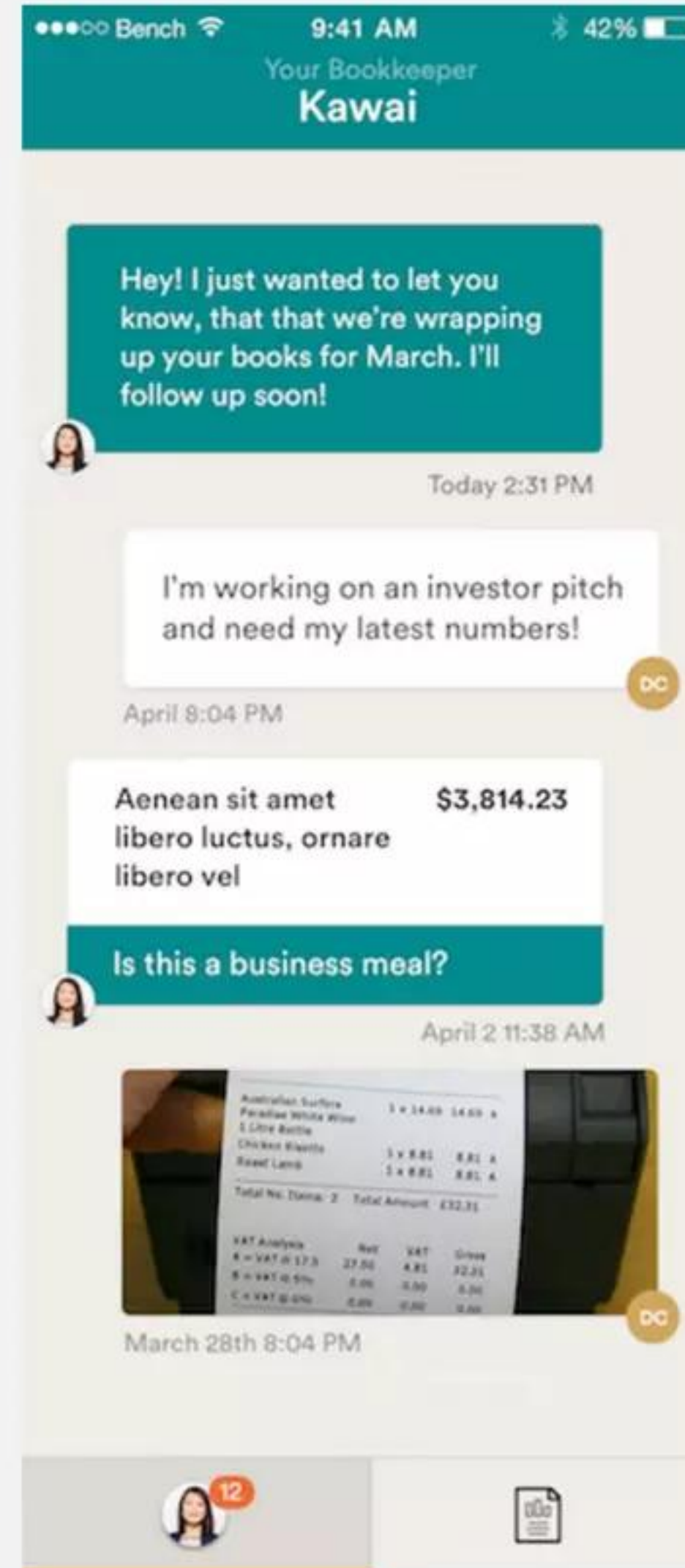
2

Client requires follow up - postpone work temporarily.

×



Context - Messaging





Context - Legacy system

Multiple issues:

- Designed for a couple of use-cases, schema is not extendable
- Wasn't built for microservices
- Tight coupling
- New requirements: messaging (web & mobile)



Events

Event - simply a fact that something happened



Events

Event:

- Immutable
- Contains:
 - timestamp
 - metadata
 - context
 - payload



Events

Event Sourcing ensures that all changes to application state are stored as a sequence of events. Not just can we query these events, we can also use the event log to reconstruct past states, and as a foundation to automatically adjust the state to cope with retroactive changes.

Martin Fowler



Events

Event Sourcing **!=** CQRS (Command Query Responsibility Segregation)



Events

Event Sourcing can be **simple**, *without* new frameworks or NoSQL databases



Events

Entry-level, Synchronous & Transactional Event Sourcing

<https://softwaremill.com/entry-level-event-sourcing/>

Adam Warski



Events

So...



Events

You won't see:

- Akka Clustering
- Akka Persistence
- Akka Streams
- CQRS
- NoSQL

You will see:

- Akka
- ActiveMQ/Camel
- Slick 3 with Postgres (JSONB)



High-level architecture - ActiveMQ

Queue

- Reliable
- Replicated
- Load balanced

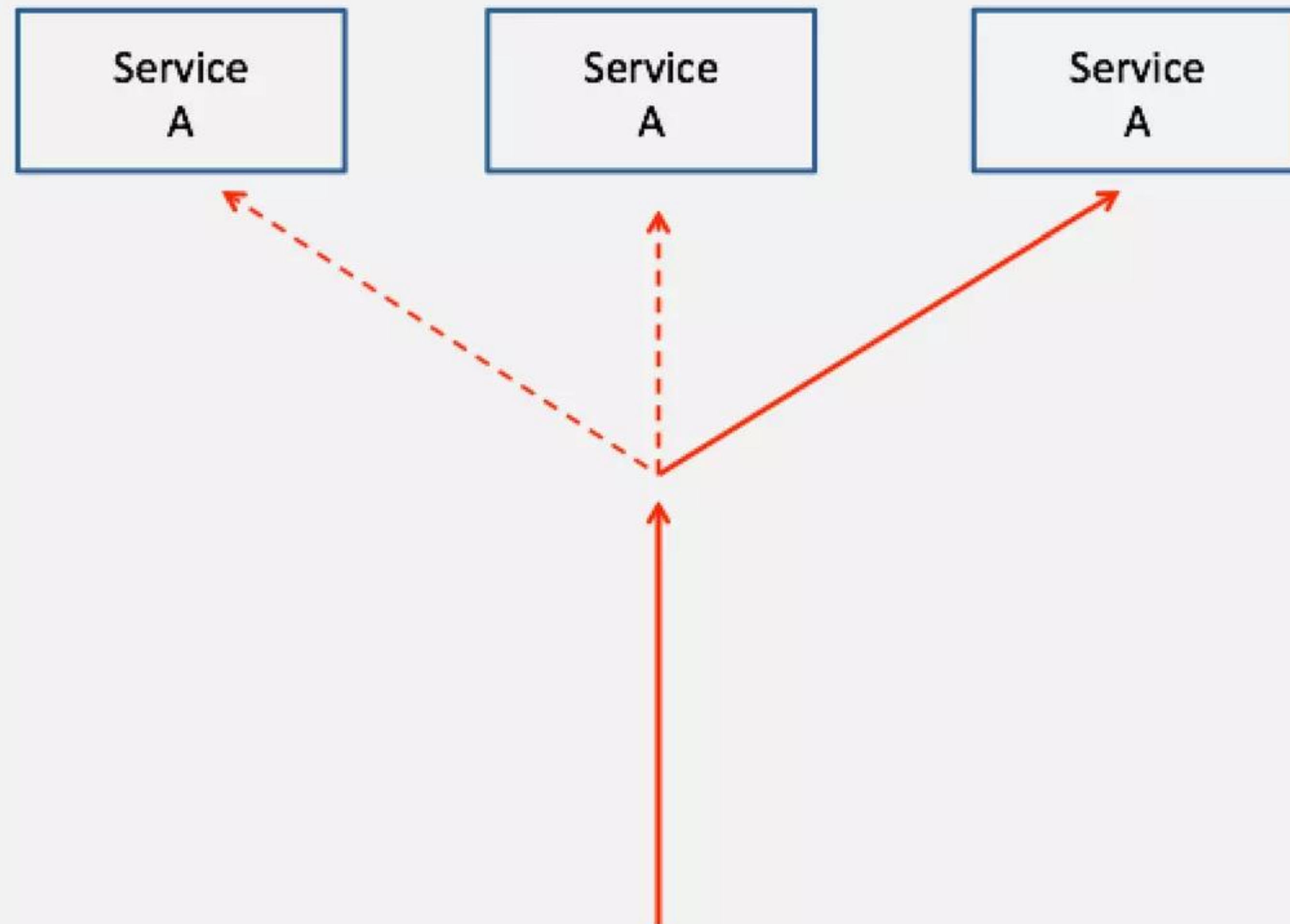
Topic

- Pub/Sub
- Broadcast



High-level architecture - ActiveMQ

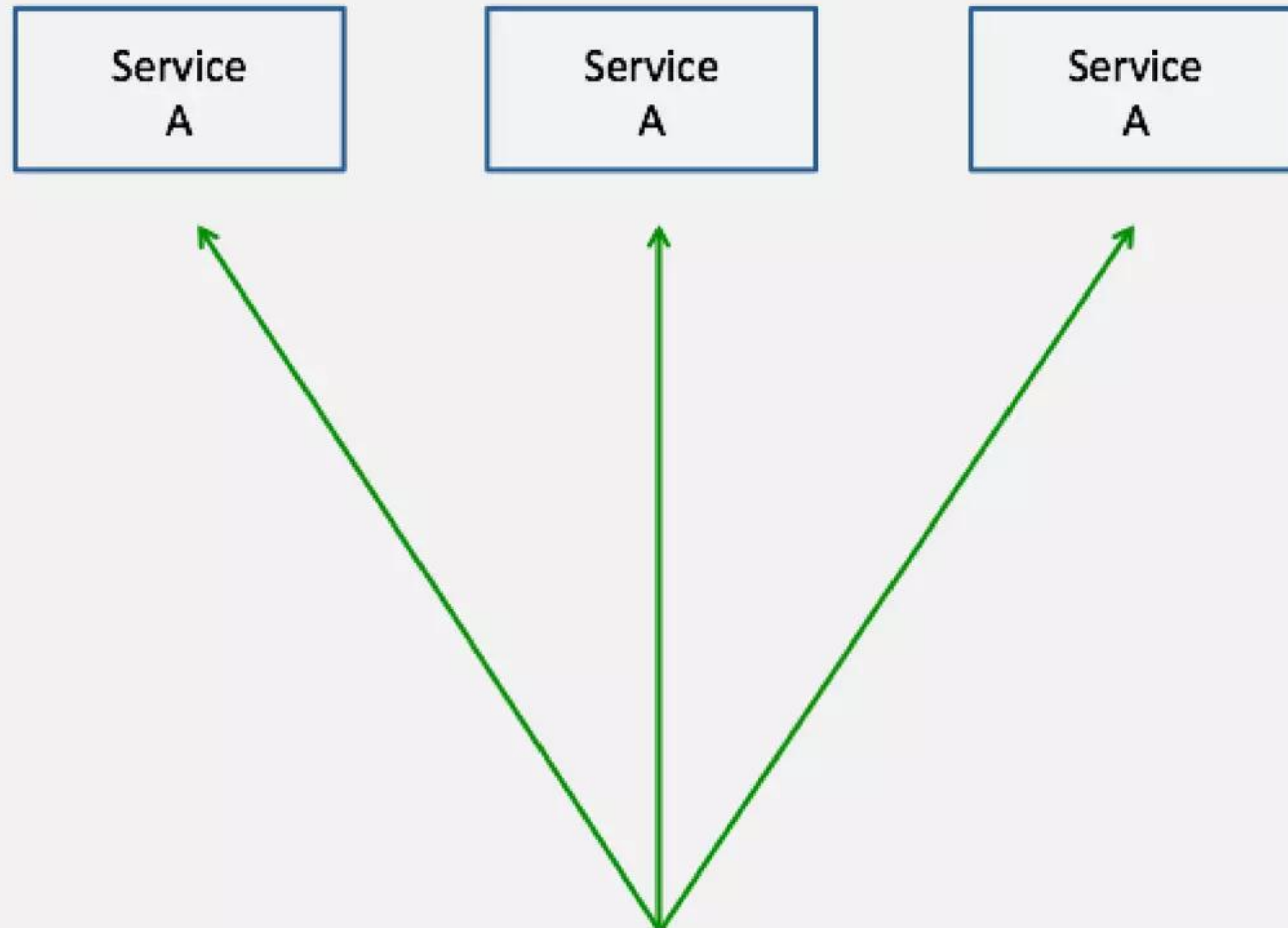
Component - Queue





High-level architecture - ActiveMQ

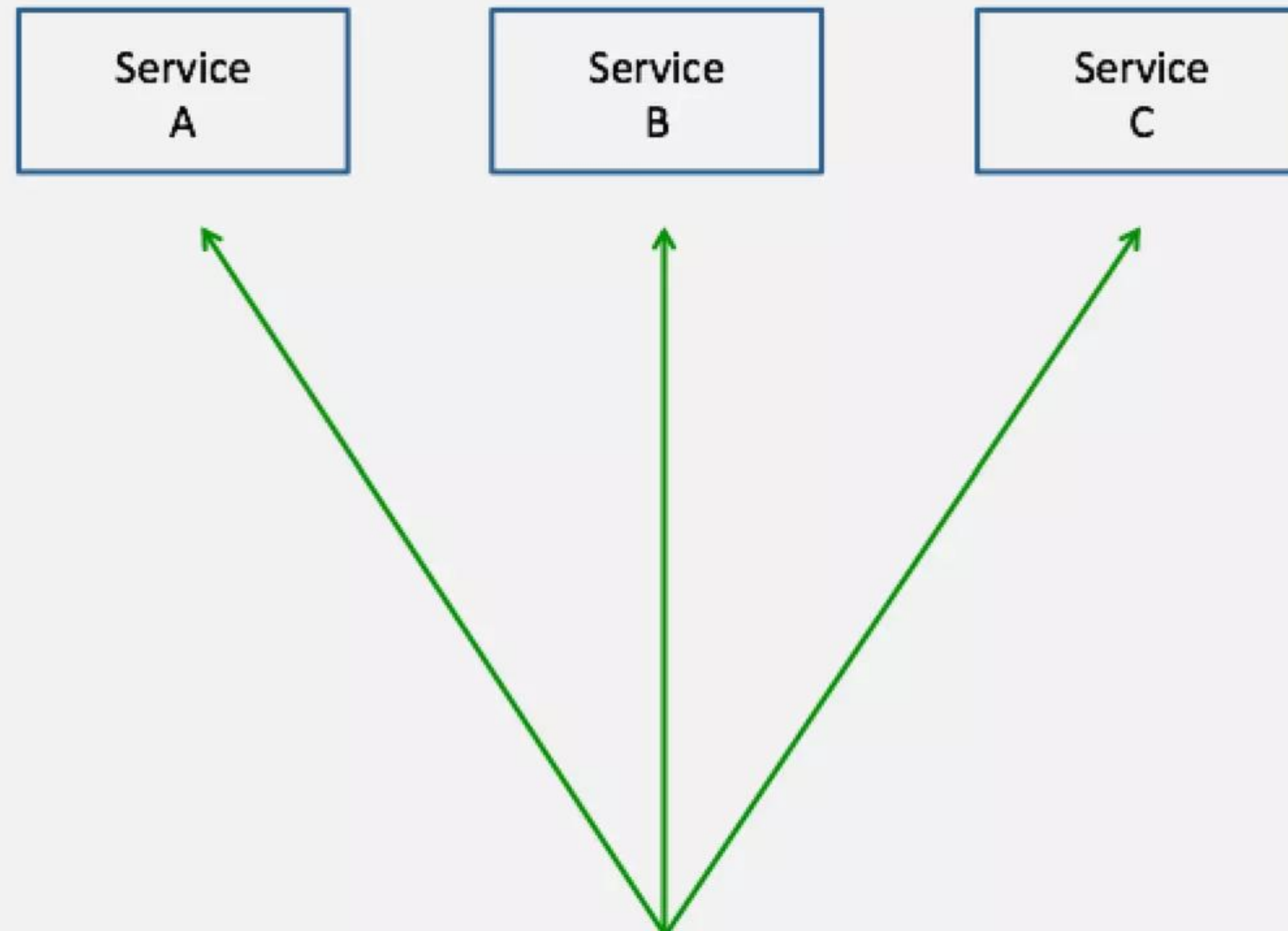
Component - Topic





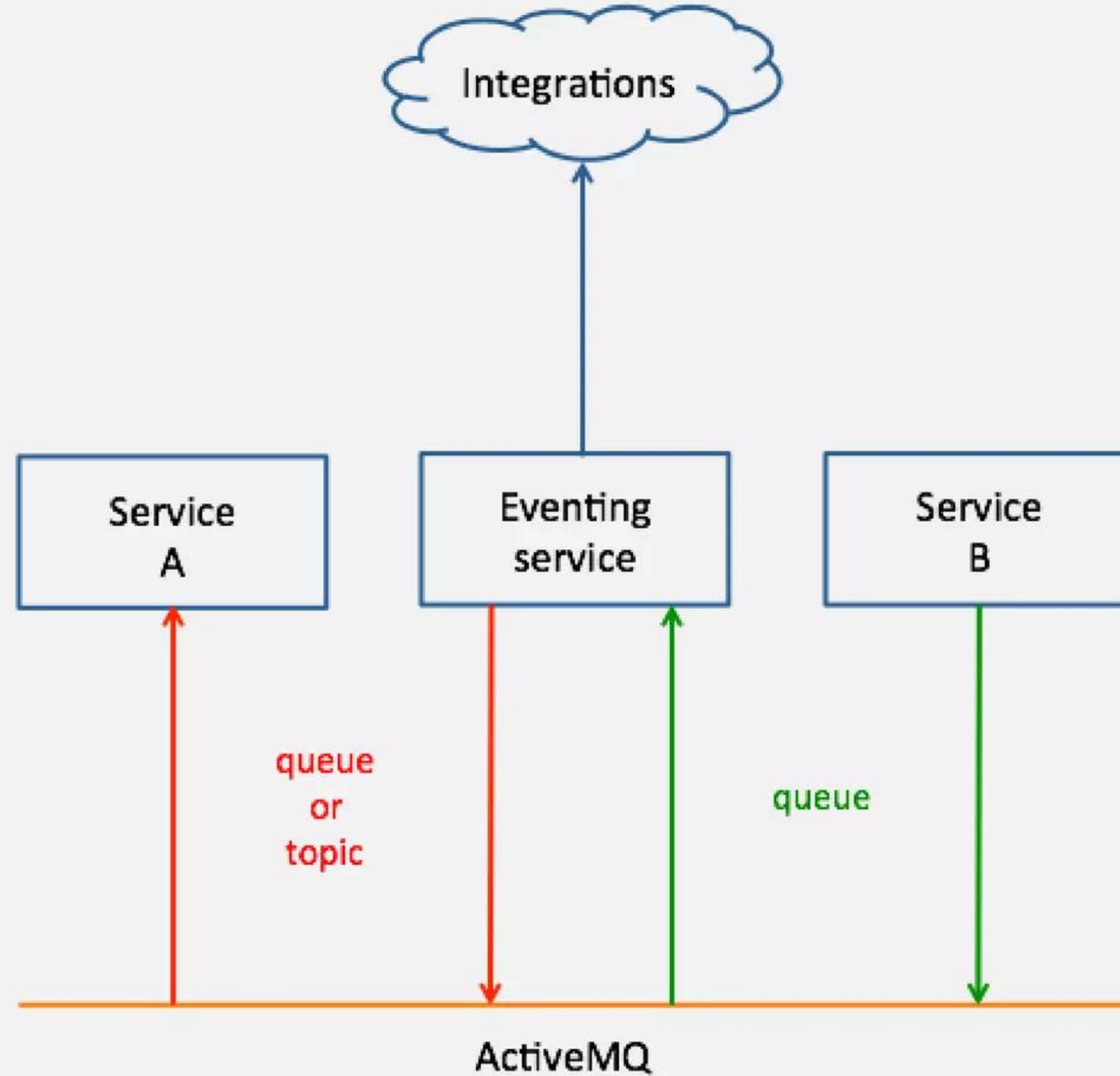
High-level architecture - ActiveMQ

Broadcast - Topic





High-level architecture





High-level architecture - Camel

```
from("direct:report")  
  .to("file:target/reports/?fileName=report.txt")
```

```
from("twitter://search?...")  
  .to("websocket:camel-tweet?sendToAll=true")
```

```
from("netty-http:http://0.0.0.0:8080")  
  .to("direct:name")
```

```
from("jms:invoices")  
  .setBody()  
  .groovy("new Invoice(request.body,currentTimeMillis())")  
  .to("mongodb:mongo?...operation=insert")
```



High-level architecture - Setup

```
trait CamelSupport extends SimpleConfigHolder {  
  
  val context = new DefaultCamelContext()  
  
  val producer = context.createProducerTemplate()  
  
  val activemqHost = config.getString("eventing.activemq.host")  
  val activemqPort = config.getString("eventing.activemq.port")  
  
  context.addComponent("activemq",  
    ActiveMQComponent.activeMQComponent(s"tcp://$activemqHost:$activemqPort"))  
}
```



High-level architecture - Setup

```
"activemq:queue:queue.eventing?  
acknowledgementModeName=CLIENT_ACKNOWLEDGE&  
transacted=true"
```



High-level architecture - Setup

```
producer.sendBodyAndHeaders(queueURI, Event.toJSON(event), headers)
```



High-level architecture - Send

```
EventingClient.buildEvent()  
  .buildSystemEvent(Event.BankError, account.benchId.toString, Component.File This)  
  .send(true)
```

```
EventingClient.buildEvent()  
  .startConfiguration(Event.SessionInvalidate, userId.toString, Component.Security)  
  .addPayloadAssets(excludedSessions)  
  .endConfiguration()  
  .sendDirect(Component.MainApp, true)
```



High-level architecture - Receive

```
import akka.camel.Consumer
```

```
trait EventingConsumer extends Actor with ActorLogging with Consumer {  
  def endpointUri = "activemq:topic:events"  
}
```

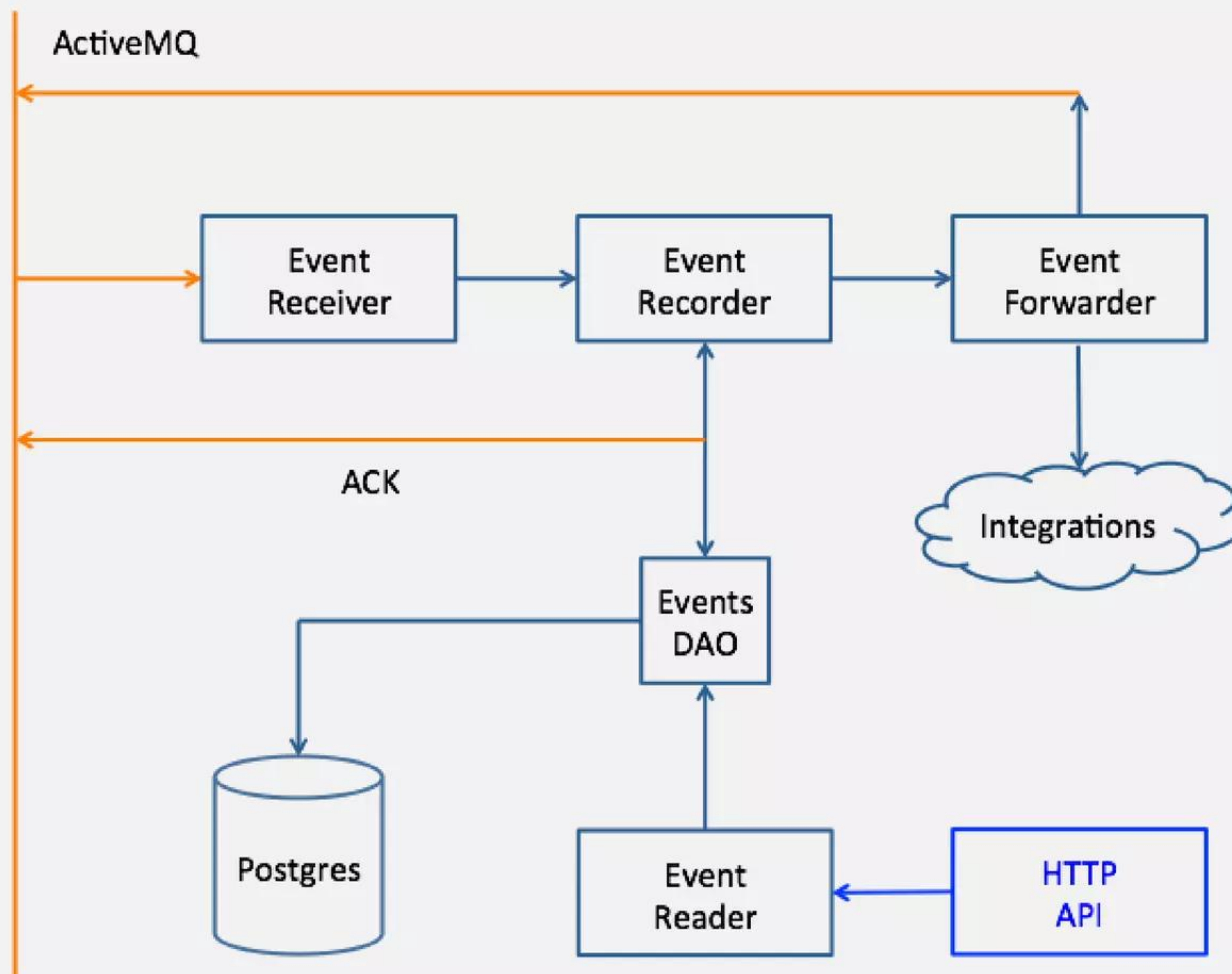


High-level architecture - Receive

```
class CustomerService extends EventingConsumer {  
  
  def receive = {  
    case e: CamelMessage if e.isEvent && e.name == "some.event.name" => {  
      e.context.personId.foreach { clientId =>  
        self ! DeleteAccount(clientId.toLong, sender())  
      }  
    }  
  }  
}
```



High-level architecture - Eventing service





High-level architecture - Event Receiver

```
override def autoAck = false
```

```
import akka.camel.Ack
```

```
sender() ! Ack
```




Schema - Legacy

```
case class InboxEvent(  
  id: ObjectId  
  name: String,  
  eventType: EventType = Inbox,  
  date: Long,  
  clientId: String,  
  itemId: String,  
  read: Boolean,  
  active: Boolean  
)
```



Schema - Legacy

```
case class InboxEvent(  
  id: ObjectId  
  name: String,  
  eventType: EventType = Inbox,  
  date: Long,  
  clientId: String,  
  itemId: String,  
  read: Boolean,  
  active: Boolean,  
  attributes: Map[String, Any]  
)
```



Schema

```
{
  "id": "2a12e2a0-b530-49ff-9e8a-6ab3923ff890",
  "createdAt": 1440610041000,
  "version": "1.0.0",
  "name": "feed.receipt.created",
  "actions": [
    {
      "id": "5cf87e73-abd5-4ed6-a1f0-661d174b38d9",
      "eventId": "2a12e2a0-b530-49ff-9e8a-6ab3923ff890",
      "createdAt": 1440610041000,
      "actionName": "viewed",
      "personId": "12345"
    }
  ],
  "context": {
    "personId": "11111",
    "eventSource": {
      "sourceType": "Person",
      "authorId": "12345",
      "authorRoles": [
        "USER"
      ]
    }
  },
  "assets": [
    {
      "assetType": "resource",
      "resourceId": "53cb38a9e4b000cda19dfa0e",
      "sourceType": "document"
    }
  ]
}
```



Schema

```
{  
  "id": "2a12e2a0-b530-49ff-9e8a-6ab3923ff890",  
  "createdAt": 1440610041000,  
  "version": "1.0.0",  
  "name": "feed.receipt.created",  
  ...  
}
```



Schema

```
{  
  ...  
  "context": {  
    "personId": "11111",  
    "eventSource": {  
      "sourceType": "Person",  
      "authorId": "12345",  
      "authorRoles": [  
        "USER"  
      ]  
    }  
  },  
  ...  
}
```



Schema

```
{  
  ...  
  "assets": [  
    {  
      "assetType": "resource",  
      "resourceId": "53cb38a9e4b000cda19dfa0e",  
      "sourceType": "document"  
    }  
  ]  
}
```



Schema

```
{
  "actions": [
    {
      "id": "5cf87e73-abd5-4ed6-a1f0-661d174b38d9",
      "eventId": "2a12e2a0-b530-49ff-9e8a-6ab3923ff890",
      "createdAt": 1440610041000,
      "actionName": "viewed",
      "personId": "12345"
    }
  ],
  ...
}
```



Schema

1 Event $\leftarrow$ X Actions



Schema

ReceiptCreated
ReceiptViewed
ReceiptArchived

Receipt
↑
Viewed
Archived



Schema

The screenshot shows a web application interface with a dark sidebar on the left containing navigation links: Dashboard, Messages, Connections, Accounts, Book Feed, Documents, Upload, Folders, and Reports. The main content area displays a notification from 'Marti' with the text: 'You have unread comments on transactions' and 'You have unread comment for Bench X Testing (Testing Client) on document #550b22cfe4b00729ef2c50b0.' Below this is a list of assets, each with a count in a circle and a blurred image. Red arrows point from labels to specific fields in the notification and asset list:

- createdAt**: Points to the timestamp 'OCT 21'.
- name**: Points to the sender's name 'Marti'.
- context.personId**: Points to the user ID field in the notification header.
- assets[0].sourceType**: Points to the 'Bench X Testing (Testing Client)' text in the notification body.
- assets[0].resourceId**: Points to the document ID '#550b22cfe4b00729ef2c50b0' in the notification body.

At the bottom of the notification, there is a text input field with the placeholder text 'Type here to reply to this message...'.



Schema

Why JSON?:

- Simple
- Easy to change
- Easy to write migrations
- Log-friendly
- Can be persisted efficiently / indexed
 - MongoDB
 - Postgres JSONB
 - ...



Persistence

Event

id	created_at	version	name	context	assets
2a12e2a0-b530-49ff-9e8a-6ab3923ff890	1440610041000	1.0.0	feed.receipt.created	{"personId": "11111", "eventSource": {"authorI...	[{"assetType": "resource", "resourceId": "12398u19...

Action

id	created_at	event_id	name	person_id
25341558-aa26-46c0-a556-6dca904fc3fe	1441411936	3802003c-7adb-43b3-bffb-0efbc0ec2e21 	viewed	111



Persistence

```
class Events(tag: Tag) extends Table[EventTuple](tag, "event") {  
  def id = column[UUID]("id", O.PrimaryKey)  
  
  def createdAt = column[Long]("created_at")  
  
  def version = column[String]("version")  
  
  def name = column[String]("name")  
  
  def context = column[JValue]("context")  
  
  def assets = column[JValue]("assets")  
  
  def * = (id, createdAt, version, name, context, assets)  
}
```



Persistence

```
def findByPersonId(personId: String, params: FilteringParams = defaults): Future[Seq[Event]] =  
  run(this.filter(_.context +>> "personId" === personId), params)
```

```
def findByResourceId(resourceId: String, params: FilteringParams = defaults): Future[Seq[Event]] =  
  run(this.filter(_.assets @> filterArrayBy("resourceId", resourceId)), params)
```

```
private def filterArrayBy(field: String, value: String): LiteralColumn[JValue] =  
  Extraction.decompose(List(Map(field -> value)))
```



Summary

- Event sourcing is (can be) simple
- Don't use NoSQL until you have to
- Invest in schema
- Think about failures before they happen



We're hiring!

<https://bench.co/careers/>

- Software Engineer - Infrastructure
- Software Engineer - Platform
- Software Engineer - Frontend



Questions?



@sap1ens

