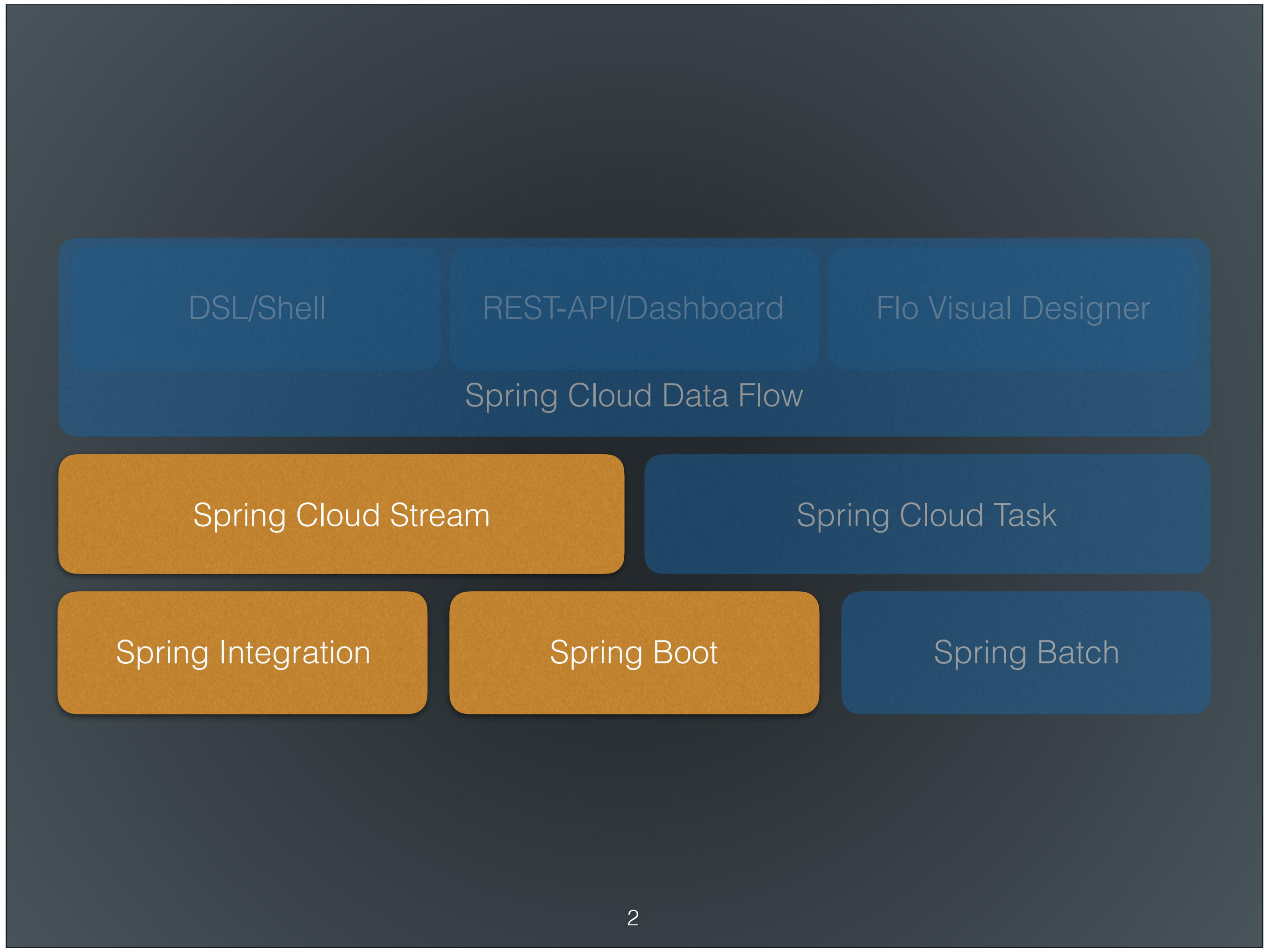
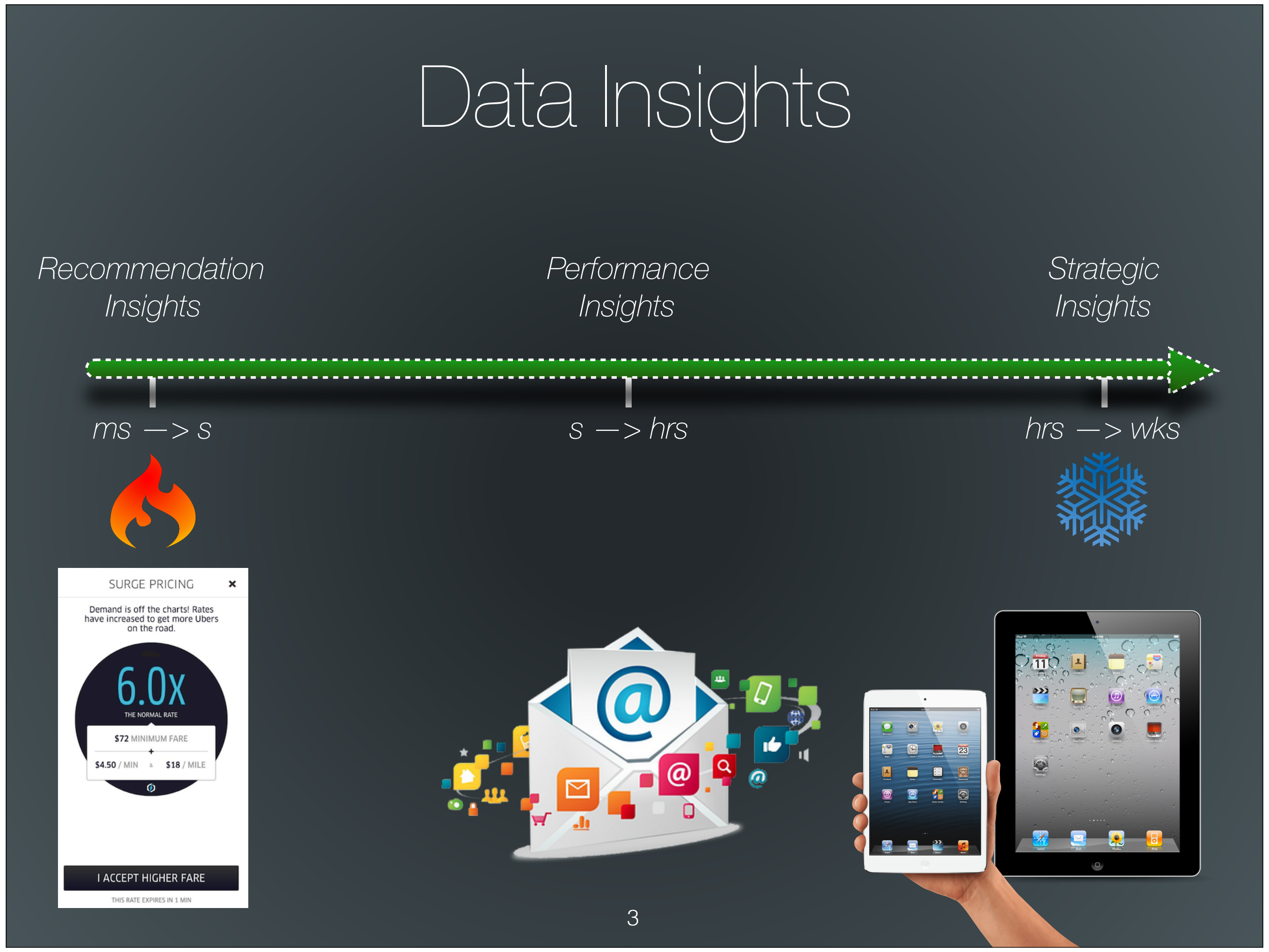
Spring Cloud Stream



Here’s the workshop roadmap again. In this session, we are going to review Spring Cloud Stream and the projects that it builds on.



Data Insights

Recommendation Insights — *ms —> s*

Performance Insights — *s —> hrs*

Strategic Insights — *hrs —> wks*

SURGE PRICING ×

Demand is off the charts! Rates have increased to get more Ubers on the road.

6.0x

THE NORMAL RATE

$72 MINIMUM FARE

+

$4.50 / MIN + $18 / MILE

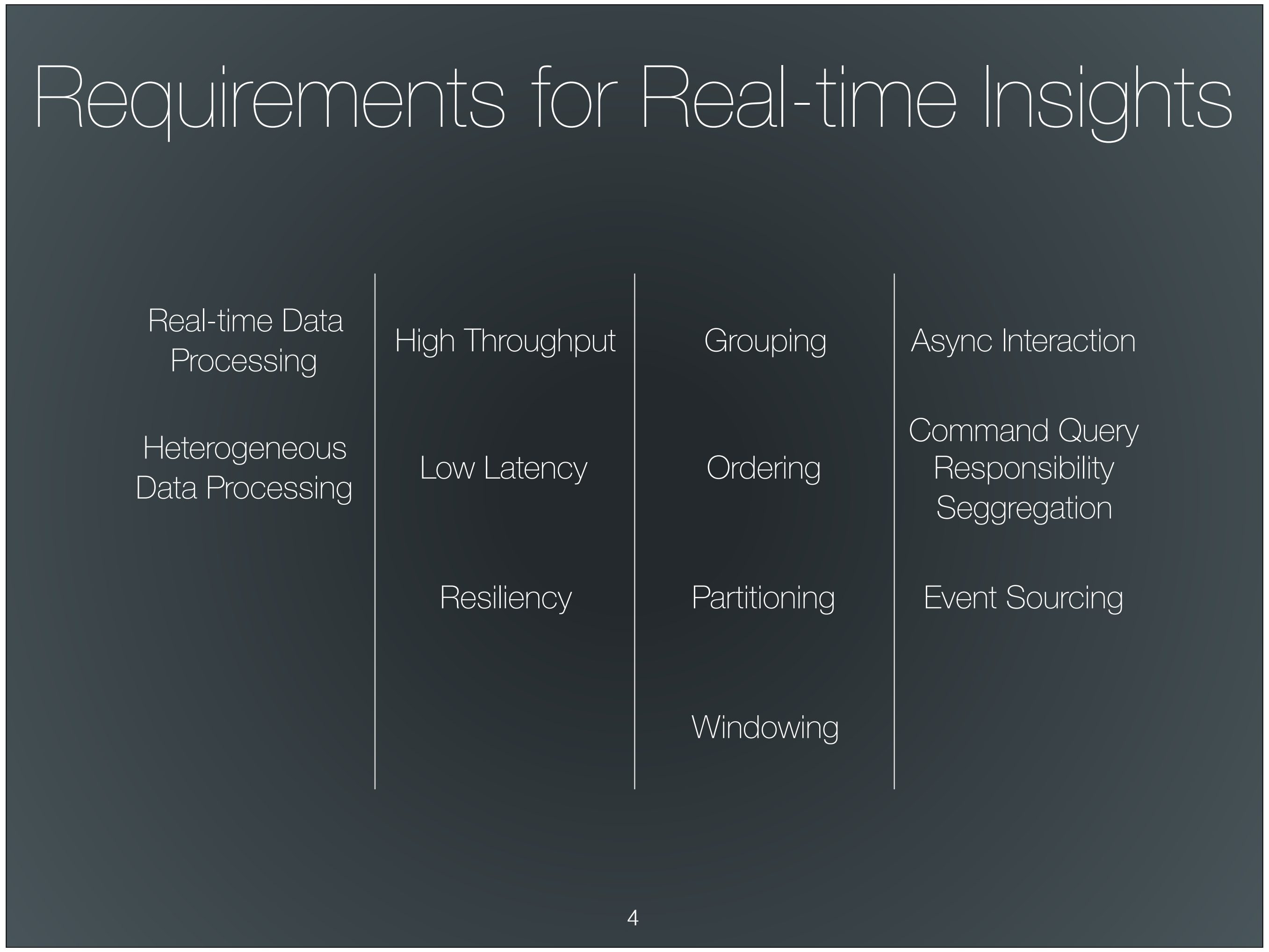
I ACCEPT HIGHER FARE

THIS RATE EXPIRES IN 1 MIN

In an enterprise setting, there is requirement to process data at various stages. Some require sub-second processing and there are cases for weeks to sometimes years worth of data that needs collected for larger strategic reasons.

To simplify this, let’s review this chart.

1) **ms -> s** processing requires insights right when the event occurred; users need immediate actionable insights to make decision. For example, think of amazon carts - you get recommendations such as “*users who have bought this also bought 1, 2, 3 …*”
2) At times we would need some data collection to make decisions such as optimizing your recommendation engine or for that matter think of a email marketing campaign - whether it works or not, you wouldn’t know until you’ve had some provable data points to back it. The optimizations on such campaigns can be incremental from then on.
3) The historical data along with real-time events tell a different story. A story that could relate to strategic decisions at the enterprise. Examples include “acquisition of a company to fill the growing demands”, “expanding line of products to be more lucrative” etc.



Requirements for Real-time Insights

Real-time Data Processing	High Throughput	Grouping	Async Interaction
Heterogeneous Data Processing	Low Latency	Ordering	Command Query Responsibility Seggregation
	Resiliency	Partitioning	Event Sourcing
		Windowing	

Now that we have a general idea about enterprise data and the various stages they add value to the business, let’s review what streaming data requirements are. The business requirements that need sub-second to seconds decisions.

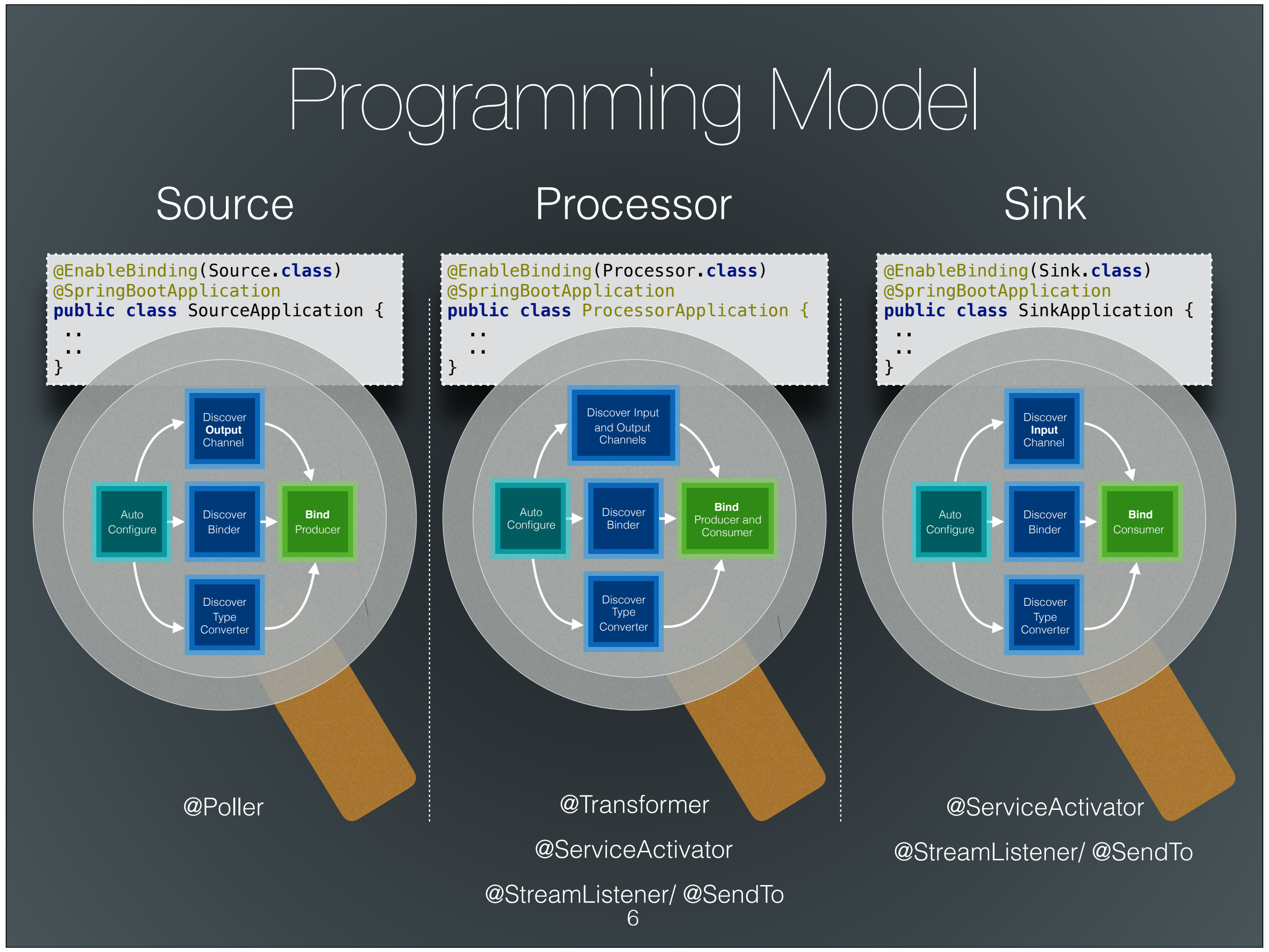
Streaming = Long Running

Spring Integration

enables lightweight messaging within Spring-based applications and supports integration with external systems

Spring Cloud Stream

enables you to create powerful event-driven streaming data applications with a minimal amount of coding



Programming Model

Source

```
@EnableBinding(Source.class)
@SpringBootApplication
public class SourceApplication {
  ..
  ..
}
```

@Poller

Processor

```
@EnableBinding(Processor.class)
@SpringBootApplication
public class ProcessorApplication {
  ..
  ..
}
```

Discover Input and Output Channels
Auto Configure
Discover Binder
Bind Producer and Consumer
Discover Type Converter

@Transformer

@ServiceActivator

@StreamListener/ @SendTo

Sink

```
@EnableBinding(Sink.class)
@SpringBootApplication
public class SinkApplication {
  ..
  ..
}
```

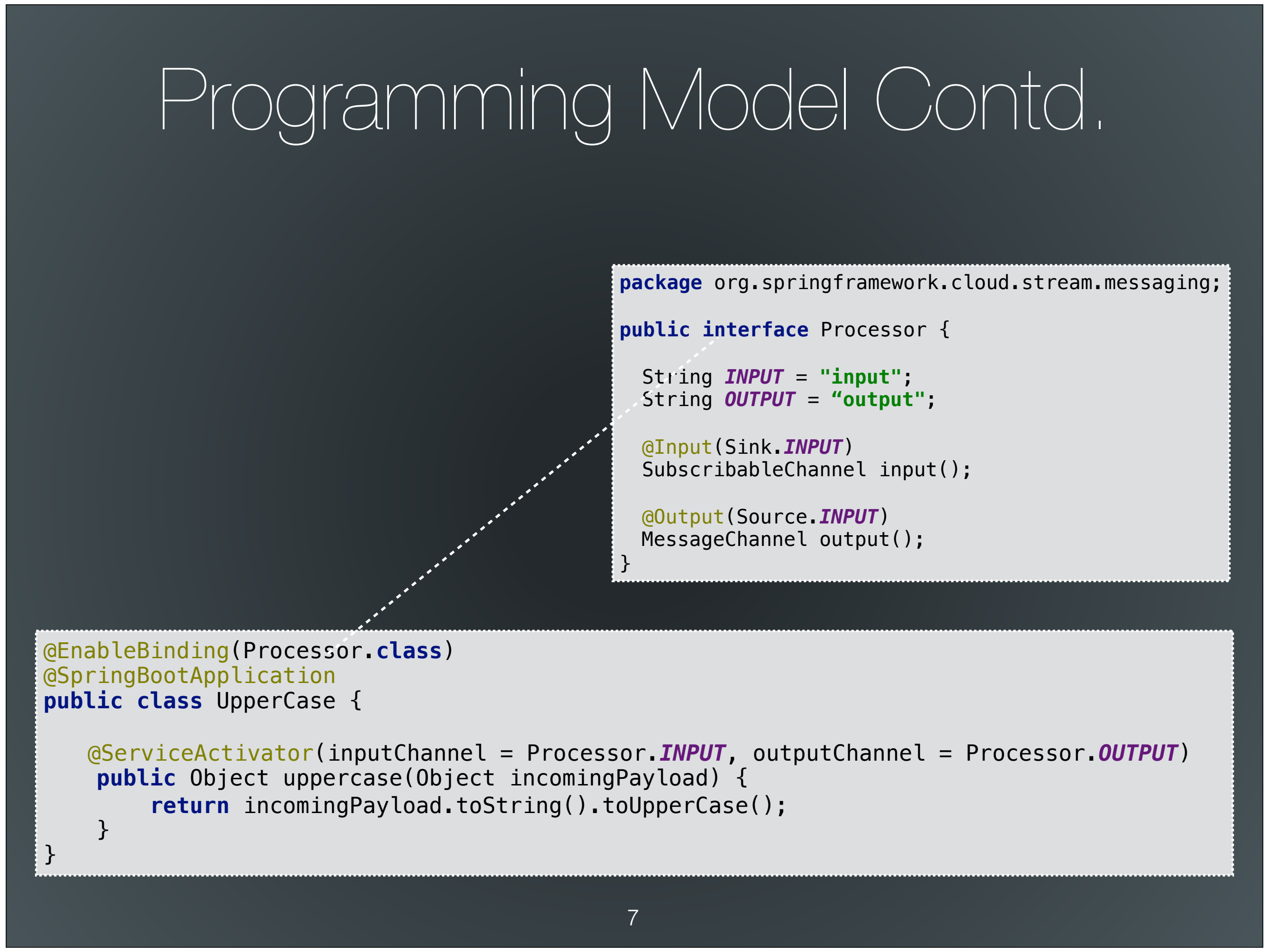
@ServiceActivator

@StreamListener/ @SendTo

A quick glance at 3 OOTB types of applications in SCSt. This is extendable; you can write your own interface with multiple channels in it, too. In other words, these 3 OOTB types aren't the only ones possible via SCSt.

Since SCSt builds upon SI at its foundation, all that applies in SI can be leveraged here as well.

High-level annotations to interact with the payload offered by SI:
http://docs.spring.io/spring-integration/api/org/springframework/integration/annotation/package-summary.html



Programming Model Contd.

```
package org.springframework.cloud.stream.messaging;

public interface Processor {

  String INPUT = "input";
  String OUTPUT = “output";

  @Input(Sink.INPUT)
  SubscribableChannel input();

  @Output(Source.INPUT)
  MessageChannel output();
}
```

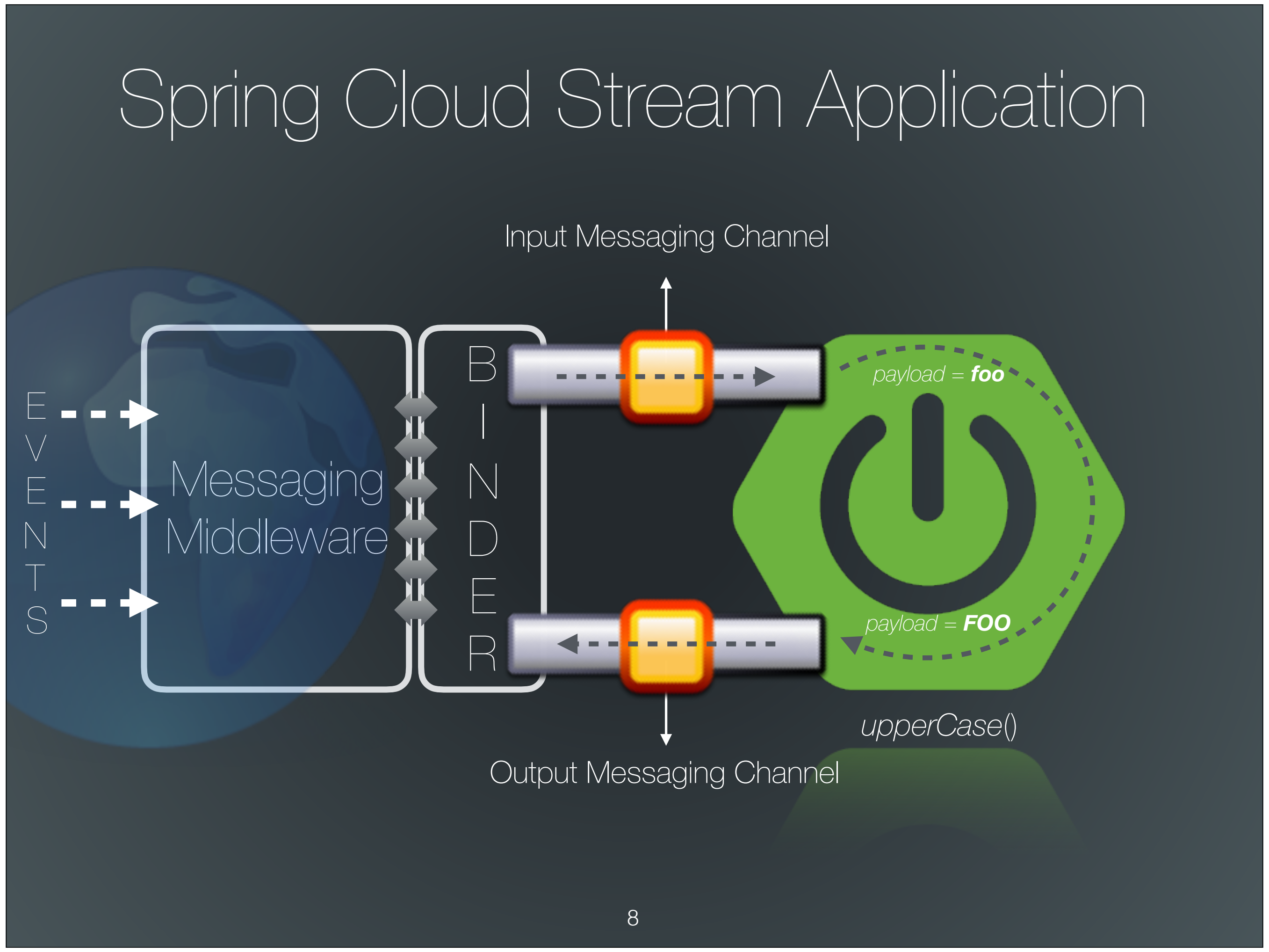
```
@EnableBinding(Processor.class)
@SpringBootApplication
public class UpperCase {

   @ServiceActivator(inputChannel = Processor.INPUT, outputChannel = Processor.OUTPUT)
    public Object uppercase(Object incomingPayload) {
        return incomingPayload.toString().toUpperCase();
    }
}
```

A deep-dive into the programming model. You’ll be doing this in the lab momentarily.

The Processor interface.

SCSt is backed by generic abstraction called “channels”. It is the channel through which the bindings to a topic/queue that occurs. The data flows through these channels backed by the messaging middleware.



Spring Cloud Stream Application

Read this visual from left -> right.

A lot of real-time events are happening; think of sensor data or the website clicks etc.

All of those events are reaching the messaging middleware and a SCSt application is subscribed to the topic where the messages land.

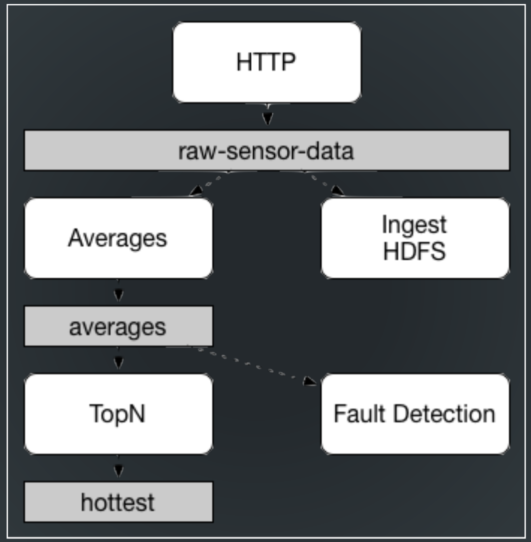
Being an event-driven application, the messages are continuously consumed by the applications through the automation of configuration and connection provided by SCSt framework.

Messages received by the applications are consumed by the “INPUT” channel and the business logic can be applied at this layer. The transformed payload is now pushed back to the messaging middleware via the “OUTPUT” channel for downstream processing.

In this example, we are seeing a simplistic use-case of payload transformation. A simple “upperCase()” example.

Demo: upperCase()

Concepts: Persistent Pub/Sub Semantics

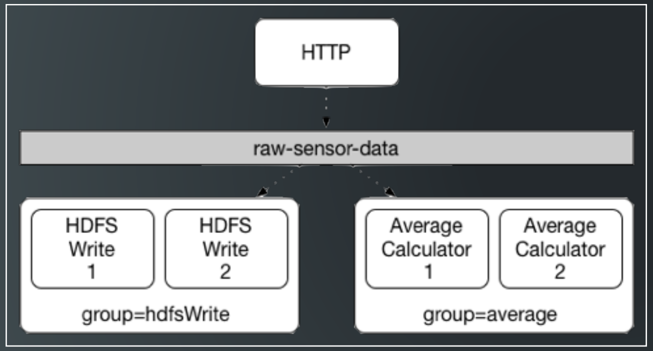


Primitives for streaming and event-driven data processing

With SCSt, you can build both streaming and event-driven applications.

1) A streaming pipeline to ingest, process and write data fragments as quickly as possible (eg., etl, streaming analytics)
2) An event-driven pipeline that kicks-off downstream processing (eg., event sourcing; async interaction)

Concepts: Consumer Groups



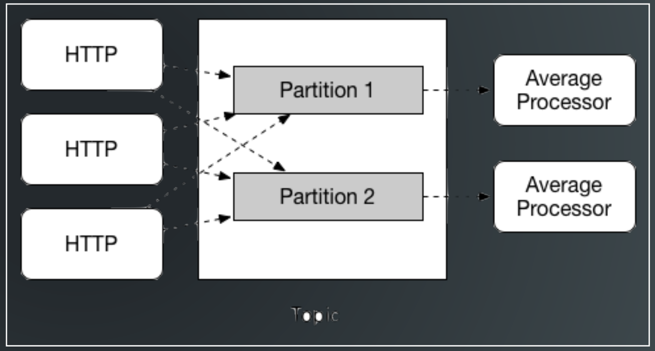
Groups of competing consumers within the pub-sub destination

Inspired from Apache Kafka’s “consumer group” functionality, SCSt provides a similar capability at the binding mechanism.

When doing this, different instances of an application are placed in a competing consumer relationship, where only one of the instances is expected to handle a given message.

Concepts: Partitioning

Primitives for stateful stream processing

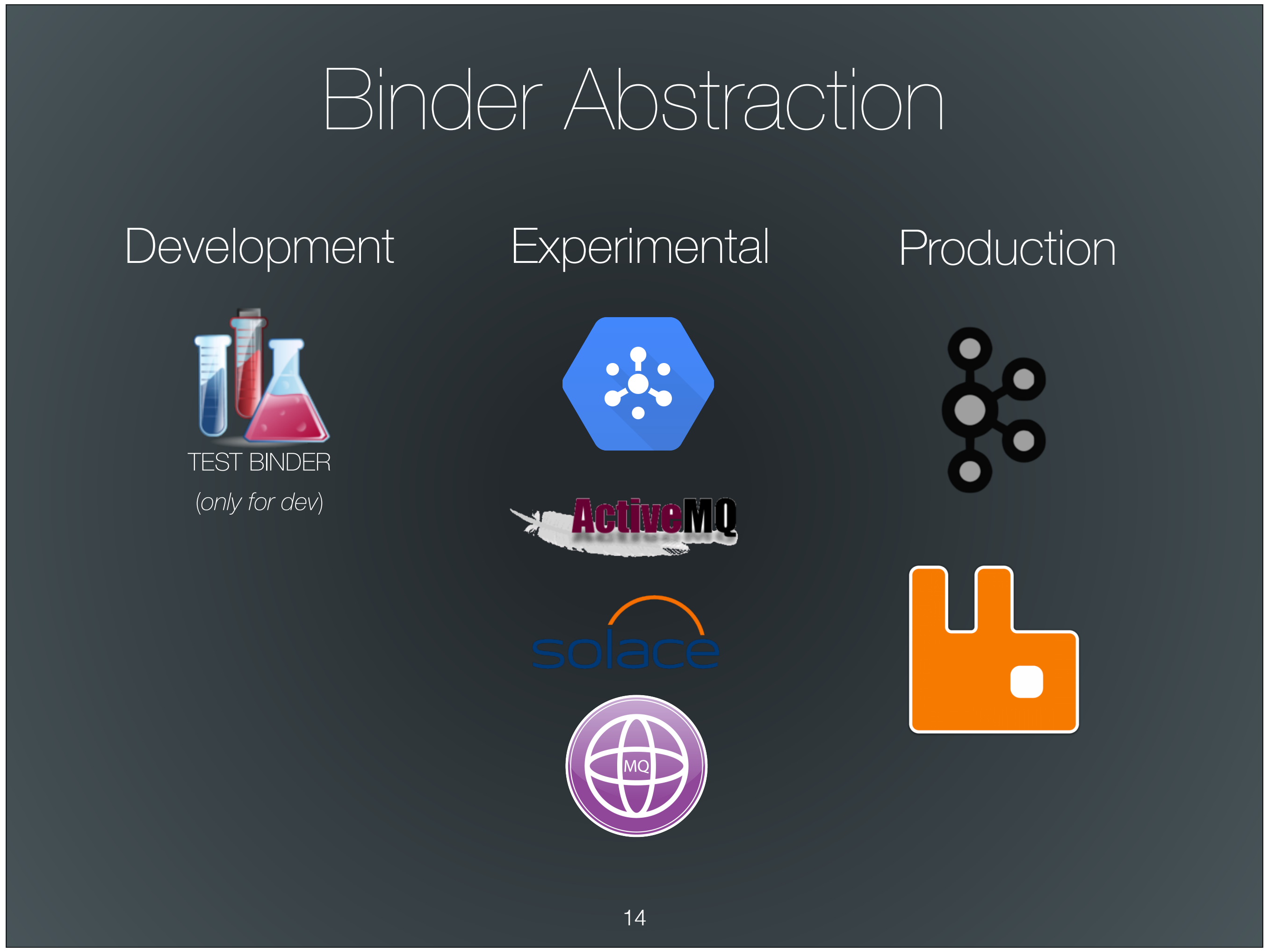


When publishing a keyed message, the SCSt’s binding mechanism deterministically maps the message to a partition based on the hash of the key. This provides a guarantee that messages with the same key are always routed to the same partition.

Grouping/ordering is important when doing stateful stream processing.

Imagine an IoT use-case where there are millions of sensor devices are in the grid. The streaming pipeline should be in a situation to process the data that belong to each device/sensor correctly. You don’t want to process wrong sensor data and end up with wrong calculations, which of course would trigger wrong predictions - wrong insights - wrong business decisions.

Demo: Partitions & Consumer Groups



Binder Abstraction

Development	Experimental	Production
TEST BINDER *(only for dev)*		

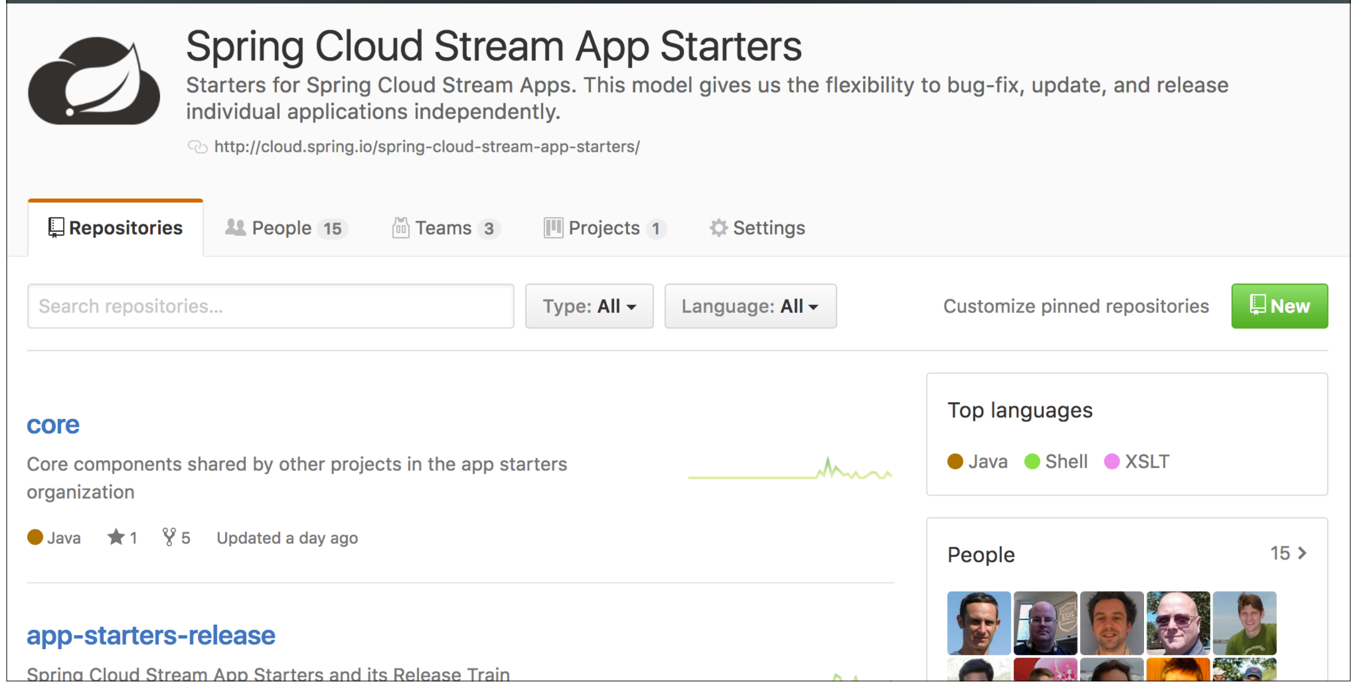
Lastly SCSt is built with binding mechanism at its core and there's an abstraction put in place at this level.

Today, we recommend Apache Kafka or RabbitMQ as the messaging middleware for production setting. However, there's few other options in various stages of release cadence.

1) Test binder is something you'd use in DEV. This is super easy for unit/integration testing
2) Experimental binders are in the process of graduating to a production quality binder
3) Of course, kafka and rabbit are ready to use in production

52 Application Starters & Counting!

https://github.com/spring-cloud-stream-app-starters



Spring Cloud Stream App Starters

Starters for Spring Cloud Stream Apps. This model gives us the flexibility to bug-fix, update, and release individual applications independently.

http://cloud.spring.io/spring-cloud-stream-app-starters/

core

Core components shared by other projects in the app starters organization

● Java ★ 1 5 Updated a day ago

app-starters-release

Spring Cloud Stream App Starters and its Release Train

Top languages

● Java ● Shell ● XSLT

People 15