Storm Topologies

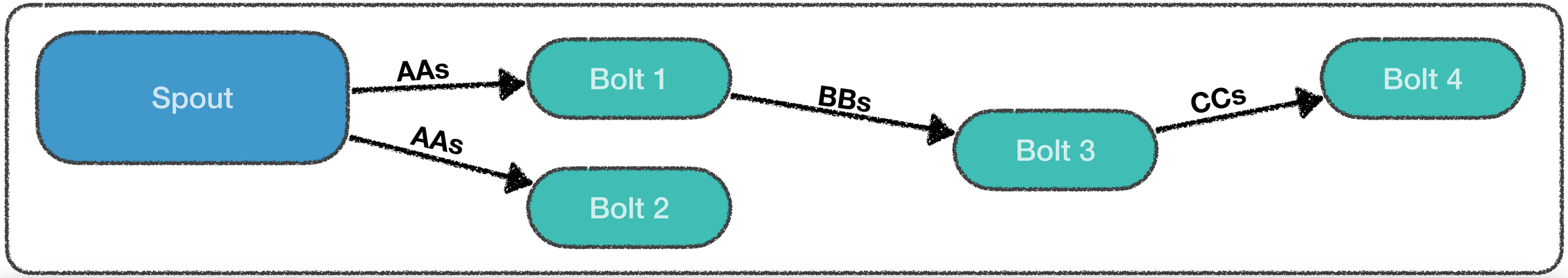
Storm Topologies

- What’s a topology;
- Spouts;
- Bolts;
- Topologies
- Reliability;
- Parallelism;
- Stream Grouping.

What’s a topology

- A topology encapsulates all the logic for a realtime application;
- Unlike Hadoop’s MapReduce jobs that eventually finish, Storm topologies can run forever;
- A topology is a graph of spouts and bolts that are connected with stream groupings;

What’s a topology



Topology Direct Acyclic Graph

Spouts

Spouts read data from an external source (Kafka, Rabbit MQ, Kestrel, etc) and emit them into the topology:

- Reliable **spout** can replay tuples if they were failed to be processed;
- Unreliable **spout** forgets about tuples as soon as they are emitted;

Spouts

```
class InstructionTranslator extends RecordTranslator[String, String] {

  override def getFieldsFor(s: String): Fields = new Fields(INSTRUCTION.name)

  override def streams(): util.List[String] = RecordTranslator.DEFAULT_STREAM

  override def apply(consumerRecord: ConsumerRecord[String, String]): util.List[AnyRef] = {

    val message       = consumerRecord.value()
    val partition     = consumerRecord.partition().toString
    val instruction   = getInstruction(message)
    val result        = instruction.map(it => new Values(it))

    result.toOption.getOrElse(null)
  }

  def getInstruction(message: String): Try[Instruction] = Try {…}

  def deserialize(event: String): Feed = {…}
```

Spouts

Tuple fields to be emitted

```
class InstructionTranslator extends RecordTranslator[Stri

  override def getFieldsFor(s: String): Fields = new Fields(INSTRUCTION.name)

  override def streams(): util.List[String] = RecordTranslator.DEFAULT_STREAM

  override def apply(consumerRecord: ConsumerRecord[String, String]): util.List[AnyRef] = {

    val message       = consumerRecord.value()
    val partition     = consumerRecord.partition().toString
    val instruction   = getInstruction(message)
    val result        = instruction.map(it => new Values(it))

    result.toOption.getOrElse(null)
  }

  def getInstruction(message: String): Try[Instruction] = Try {…}

  def deserialize(event: String): Feed = {…}
```

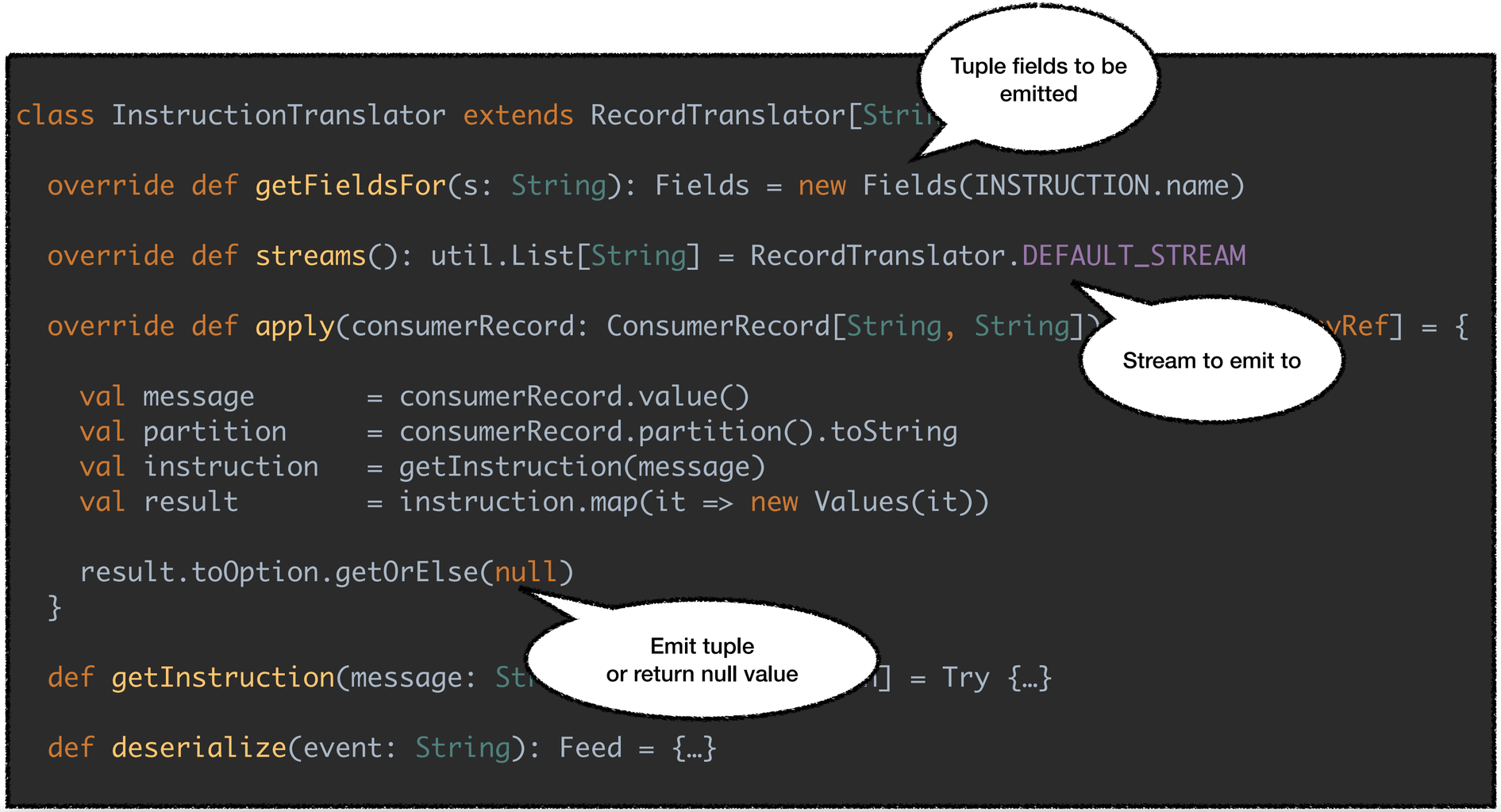
Spouts

```
class InstructionTranslator extends RecordTranslator[Stri

  override def getFieldsFor(s: String): Fields = new Fields(INSTRUCTION.name)

  override def streams(): util.List[String] = RecordTranslator.DEFAULT_STREAM

  override def apply(consumerRecord: ConsumerRecord[String, String])          vRef] = {

    val message       = consumerRecord.value()
    val partition     = consumerRecord.partition().toString
    val instruction   = getInstruction(message)
    val result        = instruction.map(it => new Values(it))

    result.toOption.getOrElse(null)
  }

  def getInstruction(message: String): Try[Instruction] = Try {…}

  def deserialize(event: String): Feed = {…}
```

Tuple fields to be emitted

Stream to emit to

Spouts


class InstructionTranslator extends RecordTranslator[Stri
override def getFieldsFor(s: String): Fields = new Fields(INSTRUCTION.name)
override def streams(): util.List[String] = RecordTranslator.DEFAULT_STREAM
override def apply(consumerRecord: ConsumerRecord[String, String])
vRef] = {
val message = consumerRecord.value()
val partition = consumerRecord.partition().toString
val instruction = getInstruction(message)
val result = instruction.map(it => new Values(it))
result.toOption.getOrElse(null)
}
def getInstruction(message: St
] = Try {…}
def deserialize(event: String): Feed = {…}
Tuple fields to be emitted
Stream to emit to
Emit tuple or return null value

Bolts

All the processing in topologies is done in **bolts**:

- **Bolts** can do filtering, functions, aggregations, joins, persisting to a data store, collect data in-memory, connect to an external service, etc.

- They can also emit to more than one stream.

Bolts

```
trait ProcessorBolt extends BaseRichBolt {

  Var output: OutputCollector = _
  Val processor: Processor[Instruction]

  override def prepare(map: util.Map[_, _], context: TopologyContext, output: OutputCollector): Unit = {
    output = output
 }

  override def execute(tuple: Tuple): Unit = {

    val instruction = tuple.getValueByField(INSTRUCTION.name).asInstanceOf[Instruction]
    val results = processor.process(instruction)

    results.map(result => output.emit(tuple, new Values(result)))
    output.ack(tuple)
  }

  override def declareOutputFields(outputFieldsDeclarer: OutputFieldsDeclarer): Unit = {
    outputFieldsDeclarer.declare(new Fields(MARKET.name))
  }
}
```

Bolts

```
trait ProcessorBolt                                    hBolt {

  Var output: Outpu
  Val processor: Proc                                  uction]

  override def prepare(map: util.Map[_, _], context: TopologyContext, output: OutputCollector): Unit = {
    output = output
 }

  override def execute(tuple: Tuple): Unit = {

    val instruction = tuple.getValueByField(INSTRUCTION.name).asInstanceOf[Instruction]
    val results = processor.process(instruction)

    results.map(result => output.emit(tuple, new Values(result)))
    output.ack(tuple)
  }

  override def declareOutputFields(outputFieldsDeclarer: OutputFieldsDeclarer): Unit = {
    outputFieldsDeclarer.declare(new Fields(MARKET.name))
  }
}
```

Dependencies are initialized here

Bolts

```
trait ProcessorBolt [illegible]hBolt {

  Var output: Output[illegible]
  Val processor: Proc[illegible]uction]

  override def prepare(map: util.Map[_, _], context: TopologyContext, output: OutputCollector): Unit = {
    output = output
  }

  override def execute(tuple: Tuple): Unit = {

    val instruction = tuple.getValueByFi[illegible]N.name).asInstanceOf[Instruction]
    val results = processor.process(inst[illegible]

    results.map(result => output.emit(tuple, new Values(result)))
    output.ack(tuple)
  }

  override def declareOutputFields(outputFieldsDeclarer: OutputFieldsDeclarer): Unit = {
    outputFieldsDeclarer.declare(new Fields(MARKET.name))
  }
}
```

Dependencies are initialized here

Anchoring to the input tuple

Bolts

```
trait ProcessorBolt [illegible]hBolt {

  Var output: Output[illegible]
  Val processor: Proc[illegible]uction]

  override def prepare(map: util.Map[_, _], context: TopologyContext, output: OutputCollector): Unit = {
    output = output
  }

  override def execute(tuple: Tuple): Unit = {

    val instruction = tuple.getValueByFi[illegible]N.name).asInstanceOf[Instruction]
    val results = processor.process(inst[illegible]

    results.map(result => output.emit(tuple, new Values(result)))
    output.ack(tuple)
  }

  override def declareOutputFields(outputFieldsDeclarer: OutputFieldsDeclarer): Unit = {
    outputFieldsDeclarer.declare(new Fields(MARKET.name))
  }
}
```

Dependencies are initialized here

Anchoring to the input tuple

Tuple fields to be emitted

Bolts

```
trait OutputBolt extends BaseRichBolt with LazyLogging {

  Var output: OutputCollector = _
  val source: OutputSource[Market]

  override def prepare(map: util.Map[_, _], topologyContext: TopologyContext, outputCollector: OutputCollector): Unit = {
    output = outputCollector
  }

  override def execute(tuple: Tuple): Unit = {

    val market  = tuple.getValueByField(MARKET.name).asInstanceOf[Market]

    source.write(key, market) match {
      case Success(_) => output.ack(tuple)
      case Failure(_) => output.fail(tuple)
   }
  }

  override def declareOutputFields(declarer: OutputFieldsDeclarer): Unit = {}
```

Bolts

```
trait OutputBolt extends BaseRichBolt with LazyLogging {

  Var output: OutputCollector = _
  val source: OutputSource[Market]

  override def prepare(map: util.Map[_, _], topologyContext: TopologyContext, outputCollector: OutputCollector): Unit = {
    output = outputCollector
  }

  override def execute( uple): Unit = {

    val market  = tu Field(MARKET.name).asInstanceOf[Market]

    source.write(key, market) tch {
      case Success(_) => output.ack(tuple)
      case Failure(_) => output.fail(tuple)
   }
  }

  override def declareOutputFields(declarer: OutputFieldsDeclarer): Unit = {}
```

Acking the input tuple

Bolts

```
trait OutputBolt extends BaseRichBolt with LazyLogging {

  Var output: OutputCollector = _
  val source: OutputSource[Market]

  override def prepare(map: util.Map[_, _], topologyContext: TopologyContext, outputCollector: OutputCollector): Unit = {
    output = outputCollector
  }

  override def execute( uple): Unit = {

    val market  = tu Field(MAR me).asInstanceOf[Market]

    source.write(key, market) tch {
      case Success(_) => output.ack(tu
      case Failure(_) => output.fail(tuple)
   }
  }

  override def declareOutputFields(declarer: OutputFieldsDeclarer): Unit = {}
```

Acking the input tuple

You can also fail the input tuple

Bolts

```
trait OutputBolt extends BaseRichBolt with LazyLogging {

  Var output: OutputCollector = _
  val source: OutputSource[Market]

  override def prepare(map: util.Map[_, _], topologyContext: TopologyContext, outputCollector: OutputCollector): Unit = {
    output = outputCollector
  }

  override def execute(tuple): Unit = {

    val market  = tuple.getValueByField(MARKET_FIELD_name).asInstanceOf[Market]

    source.write(key, market) match {
      case Success(_) => output.ack(tuple)
      case Failure(_) => output.fail(tuple)
   }
  }

  override def declareOutputFields(declarer: OutputFieldsDeclarer): Unit = {}
```

Acking the input tuple

You can also fail the input tuple

There's no fields to declare

Topologies

As said before **topologies** encapsulate all the logic for the application, it is in the **topology** builder where we define stream groupings between spout and bolts.

Topologies

```
trait SimpleTopology extends SpoutModule with ProcessorModule with OutputModule {

  val SPOUT     : String = "SimpleSpout"
  val PROCESSOR : String = "SimpleBolt"
  val OUTPUT    : String = "SecondSimpleBolt"


  def name() = "SimpleTopology"

  def create(): StormTopology = {

    val builder = new TopologyBuilder()

    builder.setSpout(SPOUT, getSpout(), 1)

    builder.setBolt(PROCESSOR, getProcessor(), 1).shuffleGrouping(SPOUT)

    builder.setBolt(OUTPUT, getOutput(), 1).shuffleGrouping(PROCESSOR)

    builder.createTopology()
  }

  def config(): Config = {
    val cg = new Config
    cg.setNumWorkers(1)
    cg
  }
}
```

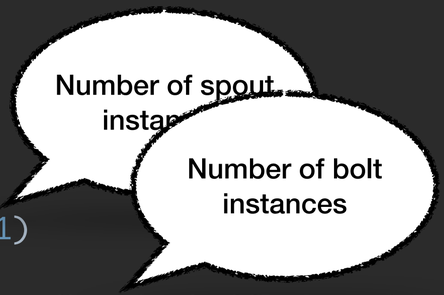
Topologies

```
trait SimpleTopology extends SpoutModule with ProcessorModule with OutputModule {

  val SPOUT     : String = "SimpleSpout"
  val PROCESSOR : String = "SimpleBolt"
  val OUTPUT    : String = "SecondSimpleBolt"


  def name() = "SimpleTopology"

  def create(): StormTopology = {

    val builder = new TopologyBuilder()

    builder.setSpout(SPOUT, getSpout(), 1)

    builder.setBolt(PROCESSOR, getProcessor(), 1).shuffleGrouping(SPOUT)

    builder.setBolt(OUTPUT, getOutput(), 1).shuffleGrouping(PROCESSOR)

    builder.createTopology()
  }

  def config(): Config = {
    val cg = new Config
    cg.setNumWorkers(1)
    cg
  }
}
```

Number of spout instances

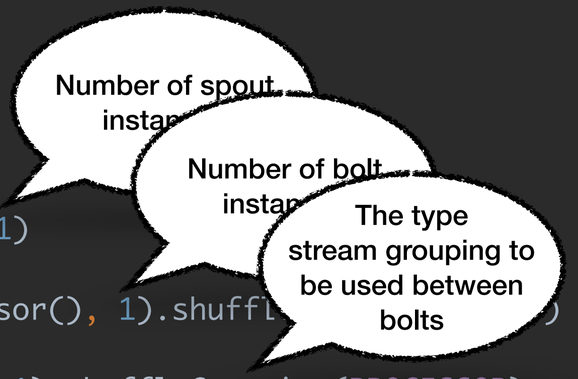
Topologies

```
trait SimpleTopology extends SpoutModule with ProcessorModule with OutputModule {

  val SPOUT     : String = "SimpleSpout"
  val PROCESSOR : String = "SimpleBolt"
  val OUTPUT    : String = "SecondSimpleBolt"


  def name() = "SimpleTopology"

  def create(): StormTopology = {

    val builder = new TopologyBuilder()

    builder.setSpout(SPOUT, getSpout(), 1)

    builder.setBolt(PROCESSOR, getProcessor(), 1).shuffleGrouping(SPOUT)

    builder.setBolt(OUTPUT, getOutput(), 1).shuffleGrouping(PROCESSOR)

    builder.createTopology()
  }

  def config(): Config = {
    val cg = new Config
    cg.setNumWorkers(1)
    cg
  }
}
```



Topologies

```
trait SimpleTopology extends SpoutModule with ProcessorModule with OutputModule {

  val SPOUT     : String = "SimpleSpout"
  val PROCESSOR : String = "SimpleBolt"
  val OUTPUT    : String = "SecondSimpleBolt"


  def name() = "SimpleTopology"

  def create(): StormTopology = {

    val builder = new TopologyBuilder()

    builder.setSpout(SPOUT, getSpout(), 1)

    builder.setBolt(PROCESSOR, getProcessor(), 1).shuffl

    builder.setBolt(OUTPUT, getOutput(), 1).shuffleGrouping(PROCESSOR)

    builder.createTopology()
  }

  def config(): Config = {
    val cg = new Config
    cg.setNumWorkers(1)
    cg
  }
}
```



Topologies

```
trait SimpleTopology extends SpoutModule with ProcessorModule with OutputModule {

  val SPOUT     : String = "SimpleSpout"
  val PROCESSOR : String = "SimpleBolt"
  val OUTPUT    : String = "SecondSimpleBolt"


  def name() = "SimpleTopology"

  def create(): StormTopology = {

    val builder = new TopologyBuilder()

    builder.setSpout(SPOUT, getSpout(), 1)

    builder.setBolt(PROCESSOR, getProcessor(), 1).shuffl

    builder.setBolt(OUTPUT, getOutput(), 1).shuffleGrouping(PROCESSOR)

    builder.createTopology
  }

  def config(): Config
    val cg = new Confic
    cg.setNumWorkers(1)
    cg
  }
}
```

Number of spout instan

Number of bolt instan

The type stream grouping to be used between bolts

The number of Workers

Topologies

Name	Id	Owner	Status	Uptime	Num workers	Num executors	Num tasks	Replication count	Assigned Mem (MB)	Scheduler Info
RAMP_INSTRUCTIONS_PROCESSOR-football-119913_102	RAMP_INSTRUCTIONS_PROCESSOR-football-119913_102-48-1556614523	storm	ACTIVE	27d 6h 59m 59s	22	1774	1774	3	18304	

Topology stats

Window	Emitted	Transferred	Complete latency (ms)	Acked	Failed
10m 0s	72982	82778	6.600	10000	
3h 0m 0s	1380836	1601501	6.173	229306	
1d 0h 0m 0s	15330332	17881216	11.148	2616412	
All time	557816000	649495000	9.753	94589000	

Spouts (All time)

Search:

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked	Failed	Error Host	Error Port	Last error	Error Time
Instructions	50	50	94806000	94806000	9.753	94589000	0				
Replication Instructions	50	50	0	0	0.000	0	0				

Showing 1 to 2 of 2 entries

Bolts (All time)

Search:

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)	Acked	Failed	Error Host	Error Port	Last error	Error Time
CircuitBreaker Outputsource	1	1	14000	14000	0.000	17.401	157000	17.197	157000	0				
Dual Mapping	154	154	94582000	94582000	0.002	0.036	94602000	0.028	94597000	0				
ES Logger Outputsource	44	44	0	0	0.000	0.235	91689000	0.227	91691000	0				
FMM Create Outputsource	44	44	0	0	0.165	195.152	1759000	216.619	1763000	0				
FMM Refresh Outputsource	88	88	0	0	0.000	96.442	52000	109.000	52000	0				
FMM Result Outputsource	88	88	0	0	0.147	131.147	1131000	123.818	1128000	0				
FMM Update Outputsource	836	836	0	0	0.187	67.665	88795000	67.729	88784000	0				
Instructions Processor	154	154	368396000	460075000	0.008	3.461	94617000	3.463	94615000	0				
PPB Outputsource	44	44	0	0	0.002	0.201	91683000	0.185	91678000	0				
SPORTEX Outputsource	44	44	0	0	0.000	0.130	91694000	0.115	91693000	0				
WatchDog Tick Outputsource	1	1	4000	4000	0.000	87.615	39000	89.375	40000	0				
WatchDog Update Outputsource	154	154	14000	14000	0.002	0.101	93306000	0.088	93311000	0				

Topologies

Name	Id	Owner	Status	Uptime	Num workers	Num executors	Num tasks	Replication count	Assigned Mem (MB)	Scheduler Info
RAMP_INSTRUCTIONS_PROCESSOR-football-119913_102	RAMP_INSTRUCTIONS_PROCESSOR-football-119913_102-48-1556614523	storm	ACTIVE	27d 6h 59m 59s	22	1774	1774	3	18304	

Topology stats

Window	Emitted	Transferred	Complete latency (ms)	Acked	Failed
10m 0s	72982	82778	6.600	10000	
3h 0m 0s	1380836	1601501	6.173	229306	
1d 0h 0m 0s	15330332	17881216	11.148	2616412	
All time	557816000	649495000	9.753	94589000	

Spouts (All time)

Search:

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked	Failed	Error Host	Error Port	Last error	Error Time
Instructions	50	50	94806000	94806000	9.753	94589000	0				
Replication Instructions	50	50	0	0	0.000	0	0				

Showing 1 to 2 of 2 entries

Bolts (All time)

Search:

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)	Acked	Failed	Error Host	Error Port	Last error	Error Time
CircuitBreaker Outputsource	1	1	14000	14000	0.000	17.401	157000	17.197	157000	0				
Dual Mapping	154	154	94582000	94582000	0.002	0.036	94602000	0.028	94597000	0				
ES Logger Outputsource	44	44	0	0	0.000	0.235	91689000	0.227	91691000	0				
FMM Create Outputsource	44	44	0	0	0.165	195.152	1759000	216.619	1763000	0				
FMM Refresh Outputsource	88	88	0	0	0.000	96.442	52000	109.000	52000	0				
FMM Result Outputsource	88	88	0	0	0.147	131.147	1131000	123.818	1128000	0				
FMM Update Outputsource	836	836	0	0	0.187	67.665	88795000	67.729	88784000	0				
Instructions Processor	154	154	368396000	460075000	0.008	3.461	94617000	3.463	94615000	0				
PPB Outputsource	44	44	0	0	0.002	0.201	91683000	0.185	91678000	0				
SPORTEX Outputsource	44	44	0	0	0.000	0.130	91694000	0.115	91693000	0				
WatchDog Tick Outputsource	1	1	4000	4000	0.000	87.615	39000	89.375	40000	0				
WatchDog Update Outputsource	154	154	14000	14000	0.002	0.101	93306000	0.088	93311000	0				

Topologies

Name	Id	Owner	Status	Uptime	Num workers	Num executors	Num tasks	Replication count	Assigned Mem (MB)	Scheduler Info
RAMP_INSTRUCTIONS_PROCESSOR-football-119913_102	RAMP_INSTRUCTIONS_PROCESSOR-football-119913_102-48-1556614523	storm	ACTIVE	27d 6h 59m 59s	22	1774	1774	3	18304	

Topology stats

Window	Emitted	Transferred	Complete latency (ms)	Acked	Failed
10m 0s	72982	82778	6.600	10000	
3h 0m 0s	1380836	1601501	6.173	229306	
1d 0h 0m 0s	15330332	17881216	11.148	2616412	
All time	557816000	649495000	9.753	94589000	

Spouts (All time)

Search:

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked	Failed	Error Host	Error Port	Last error	Error Time
Instructions	50	50	94806000	94806000	9.753	94589000	0				
Replication Instructions	50	50	0	0	0.000	0	0				

Showing 1 to 2 of 2 entries

Bolts (All time)

Search:

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)	Acked	Failed	Error Host	Error Port	Last error	Error Time
CircuitBreaker Outputsource	1	1	14000	14000	0.000	17.401	157000	17.197	157000	0				
Dual Mapping	154	154	94582000	94582000	0.002	0.036	94602000	0.028	94597000	0				
ES Logger Outputsource	44	44	0	0	0.000	0.235	91689000	0.227	91691000	0				
FMM Create Outputsource	44	44	0	0	0.165	195.152	1759000	216.619	1763000	0				
FMM Refresh Outputsource	88	88	0	0	0.000	96.442	52000	109.000	52000	0				
FMM Result Outputsource	88	88	0	0	0.147	131.147	1131000	123.818	1128000	0				
FMM Update Outputsource	836	836	0	0	0.187	67.665	88795000	67.729	88784000	0				
Instructions Processor	154	154	368396000	460075000	0.008	3.461	94617000	3.463	94615000	0				
PPB Outputsource	44	44	0	0	0.002	0.201	91683000	0.185	91678000	0				
SPORTEX Outputsource	44	44	0	0	0.000	0.130	91694000	0.115	91693000	0				
WatchDog Tick Outputsource	1	1	4000	4000	0.000	87.615	39000	89.375	40000	0				
WatchDog Update Outputsource	154	154	14000	14000	0.002	0.101	93306000	0.088	93311000	0				

Topologies

Name	Id	Owner	Status	Uptime	Num workers	Num executors	Num tasks	Replication count	Assigned Mem (MB)	Scheduler Info
RAMP_INSTRUCTIONS_PROCESSOR-football-119913_102	RAMP_INSTRUCTIONS_PROCESSOR-football-119913_102-48-1556614523	storm	ACTIVE	27d 6h 59m 59s	22	1774	1774	3	18304	

Topology stats

Window	Emitted	Transferred	Complete latency (ms)	Acked	Failed
10m 0s	72982	82778	6.600	10000	
3h 0m 0s	1380836	1601501	6.173	229306	
1d 0h 0m 0s	15330332	17881216	11.148	2616412	
All time	557816000	649495000	9.753	94589000	

Spouts (All time)

Search:

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked	Failed	Error Host	Error Port	Last error	Error Time
Instructions	50	50	94806000	94806000	9.753	94589000	0				
Replication Instructions	50	50	0	0	0.000	0	0				

Showing 1 to 2 of 2 entries

Bolts (All time)

Search:

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)	Acked	Failed	Error Host	Error Port	Last error	Error Time
CircuitBreaker Outputsource	1	1	14000	14000	0.000	17.401	157000	17.197	157000	0				
Dual Mapping	154	154	94582000	94582000	0.002	0.036	94602000	0.028	94597000	0				
ES Logger Outputsource	44	44	0	0	0.000	0.235	91689000	0.227	91691000	0				
FMM Create Outputsource	44	44	0	0	0.165	195.152	1759000	216.619	1763000	0				
FMM Refresh Outputsource	88	88	0	0	0.000	96.442	52000	109.000	52000	0				
FMM Result Outputsource	88	88	0	0	0.147	131.147	1131000	123.818	1128000	0				
FMM Update Outputsource	836	836	0	0	0.187	67.665	88795000	67.729	88784000	0				
Instructions Processor	154	154	368396000	460075000	0.008	3.461	94617000	3.463	94615000	0				
PPB Outputsource	44	44	0	0	0.002	0.201	91683000	0.185	91678000	0				
SPORTEX Outputsource	44	44	0	0	0.000	0.130	91694000	0.115	91693000	0				
WatchDog Tick Outputsource	1	1	4000	4000	0.000	87.615	39000	89.375	40000	0				
WatchDog Update Outputsource	154	154	14000	14000	0.002	0.101	93306000	0.088	93311000	0				

Reliability

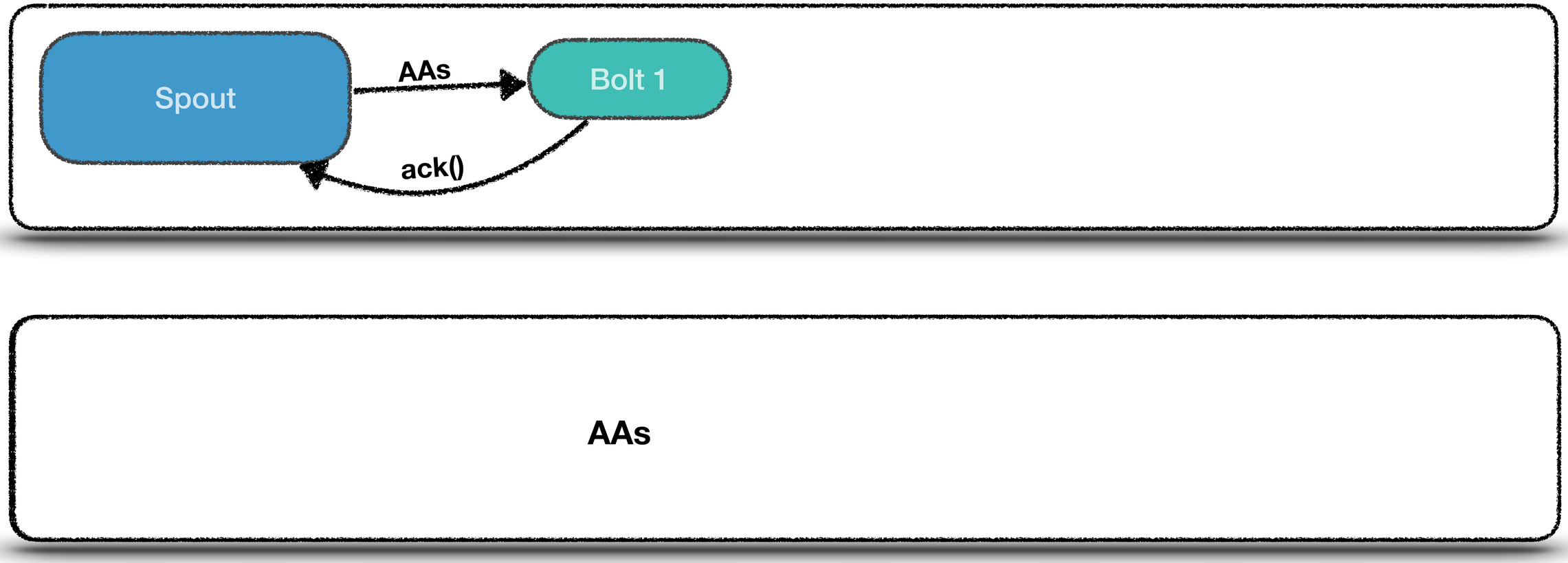
In order to have some level of **reliability**, one needs:

- A fault-tolerant Storm cluster infrastructure;
- Guarantee message processing;
  - A reliable spout
  - An anchored tuple stream
  - To ack each processed tuple or notifies of failure;

Reliability

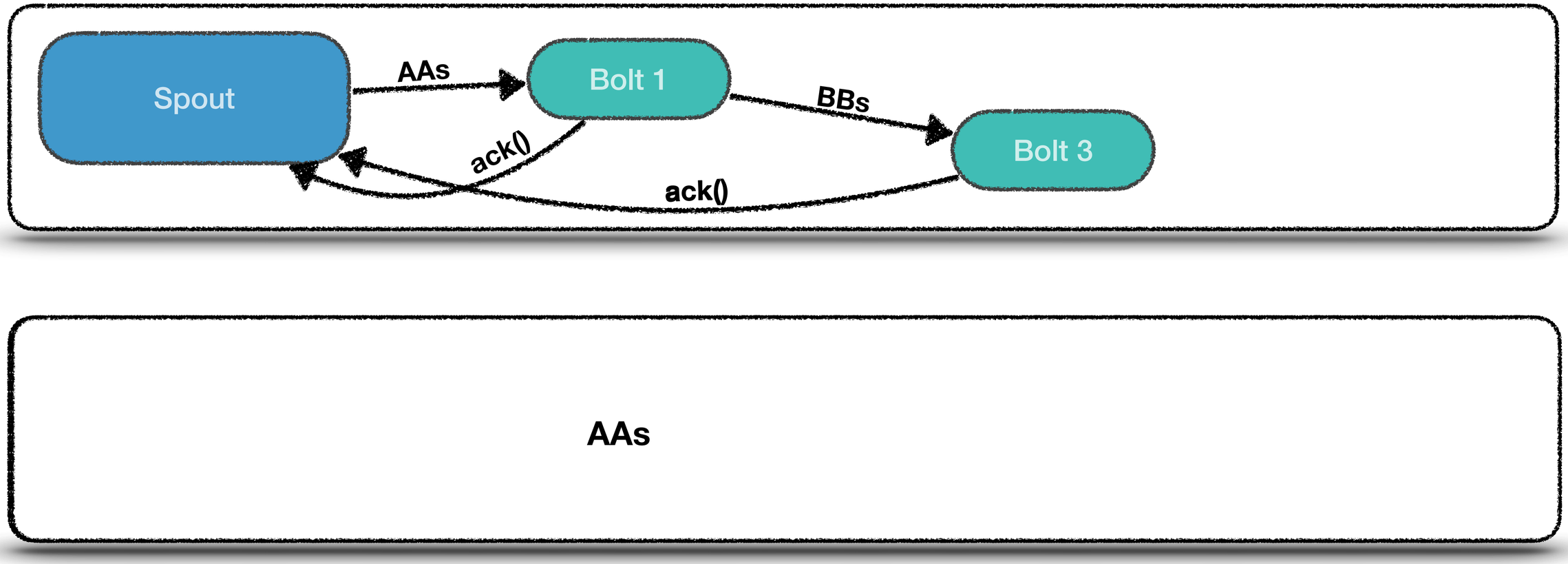
```
trait ISpout extends Serializable {
  def open(var1: Map[String, Any], var2: TopologyContext, var3: SpoutOutputCollector): Unit

  def close(): Unit

  def activate(): Unit

  def deactivate(): Unit

  def nextTuple(): Unit

  def ack(var1: Any): Unit

  def fail(var1: Any): Unit
}
```

Reliability



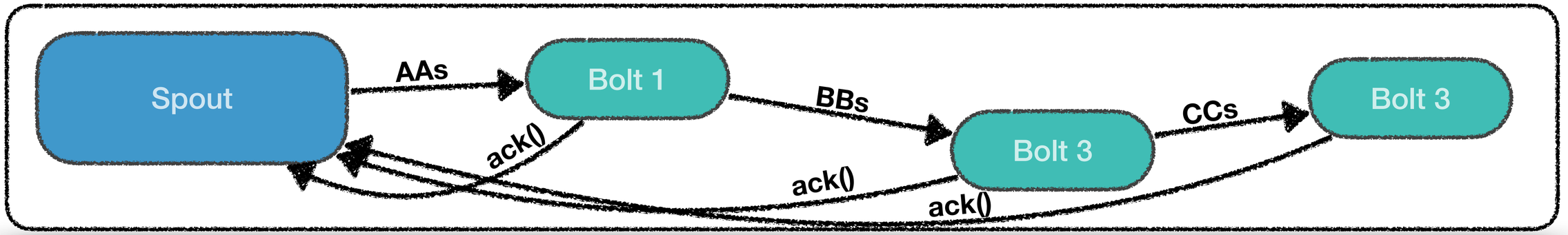
Acking & Anchoring

Reliability



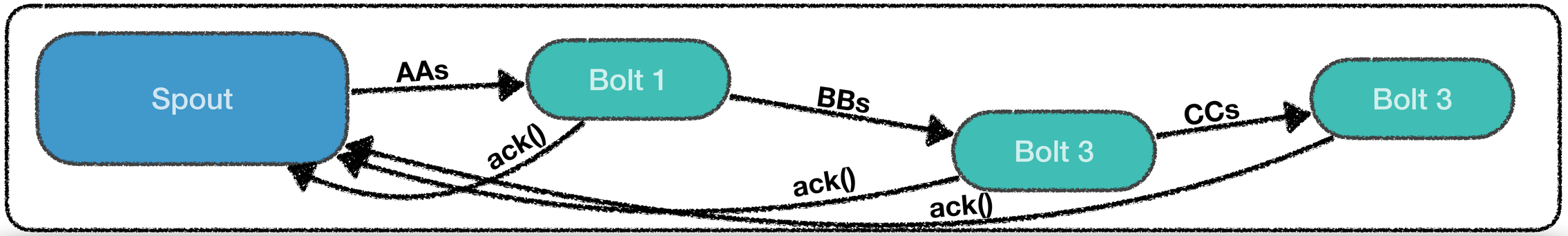
Acking & Anchoring

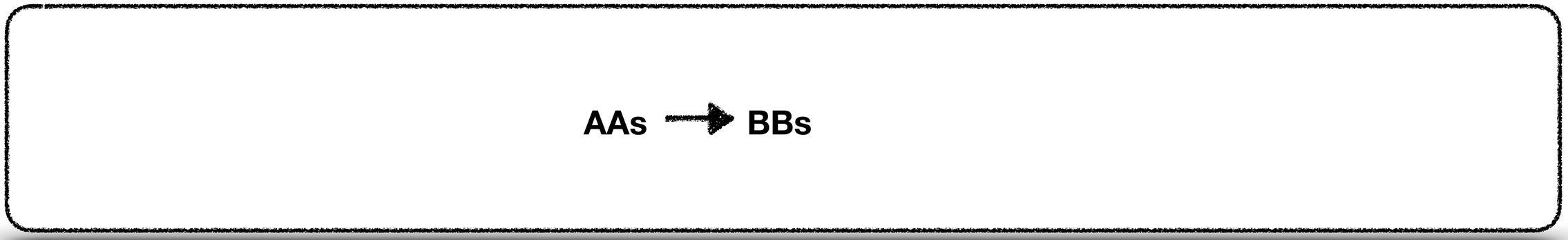
Reliability



Acking & Anchoring

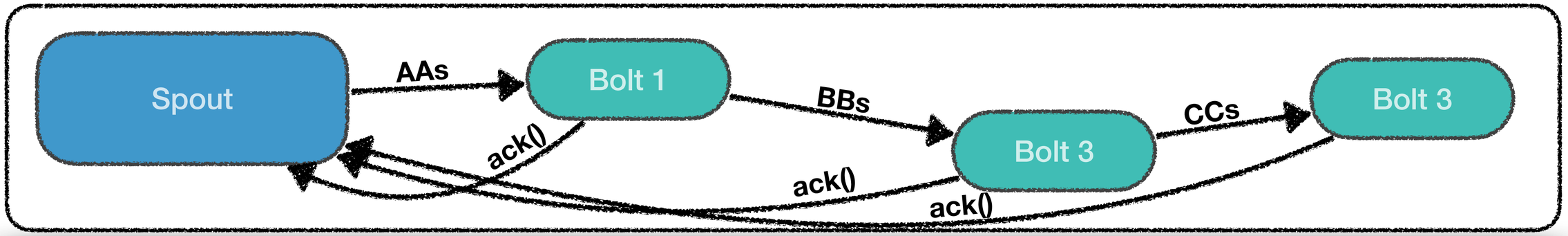
Reliability





Acking & Anchoring

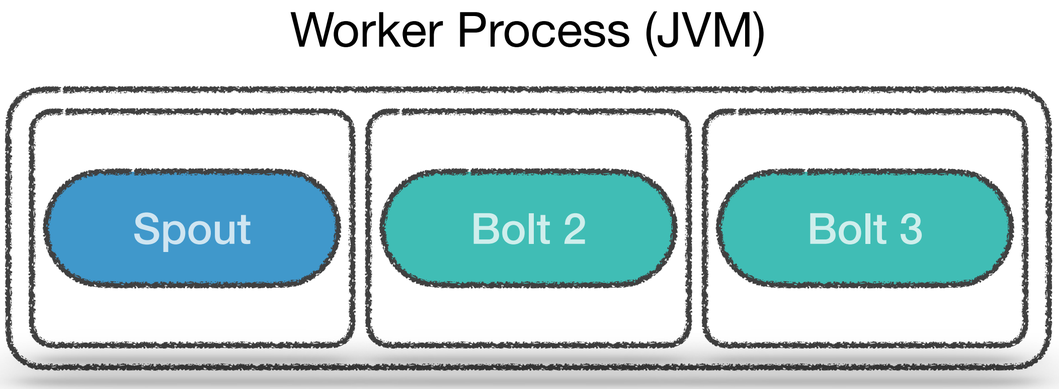
Reliability



AAs → BBs → CCs

Acking & Anchoring

Parallelism

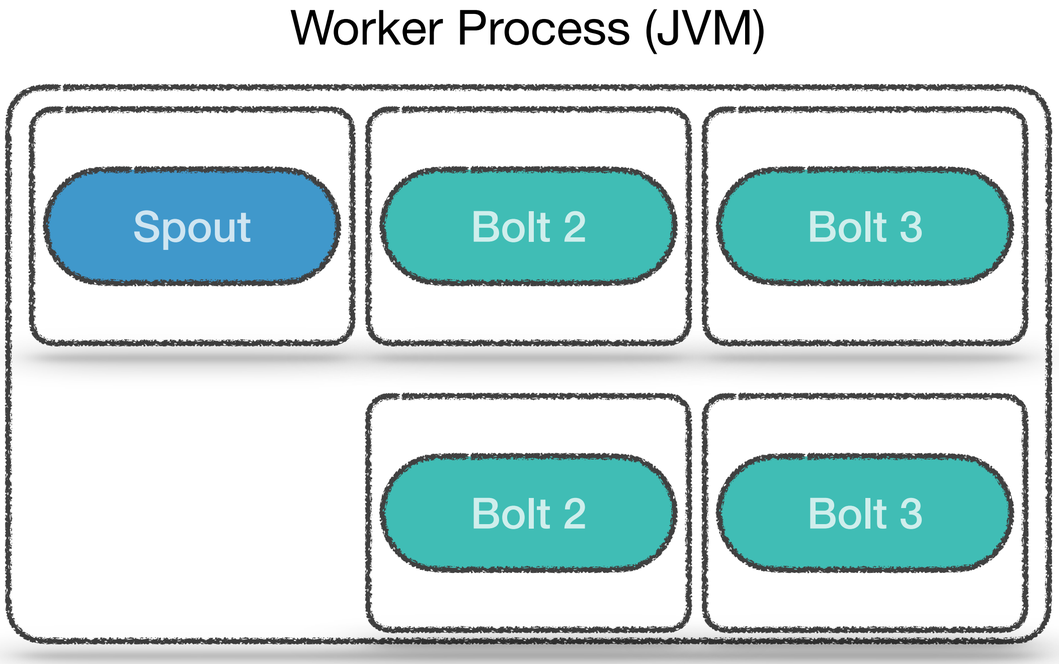



```
.setSpout(“Spout Name”, spout, 1)
.setBolt(“Bolt 2 Name”, bolt2,1)
.setBolt(“Bolt 3 Name”, bolt3,1)
…
cg.setNumWorkers(1)
```

parallelism hint
==
number of executors

number of worker processes

Parallelism

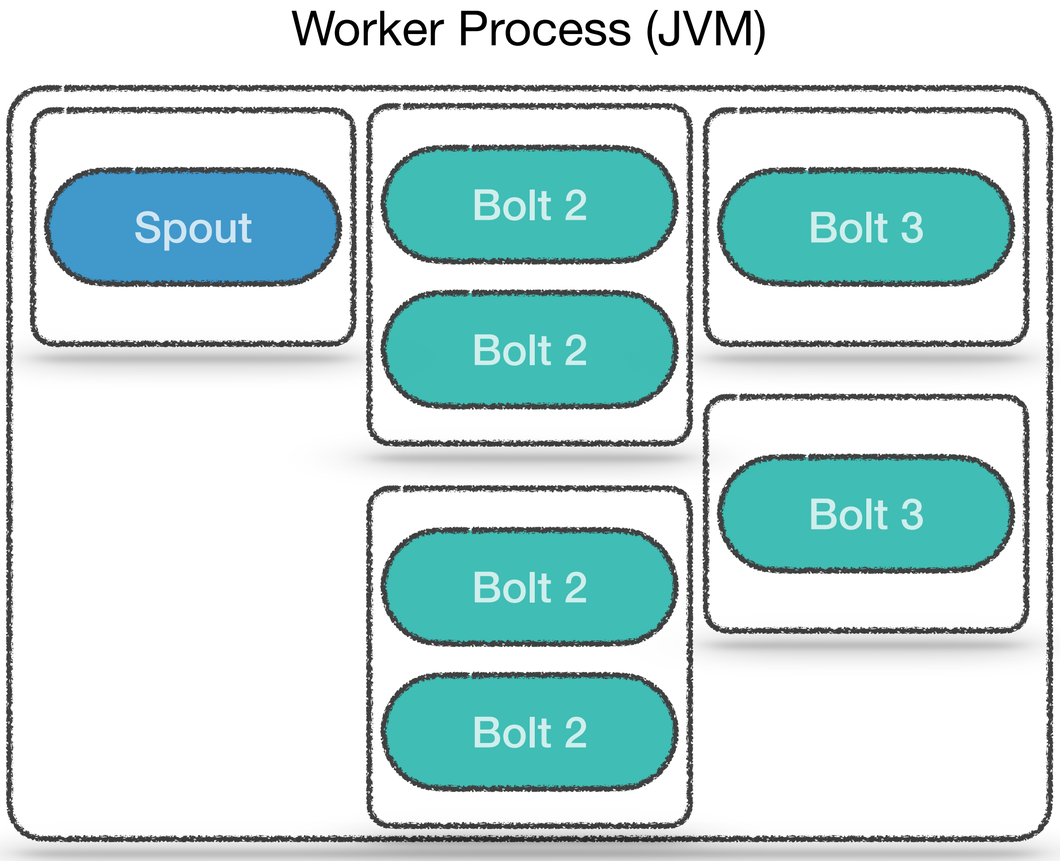



```
.setSpout(“Spout Name”, spout, 1)
.setBolt(“Bolt 2 Name”, bolt2,2)
.setBolt(“Bolt 3 Name”, bolt3,2)
…
cg.setNumWorkers(1)
```

parallelism hint
==
number of executors

number of worker processes

Parallelism

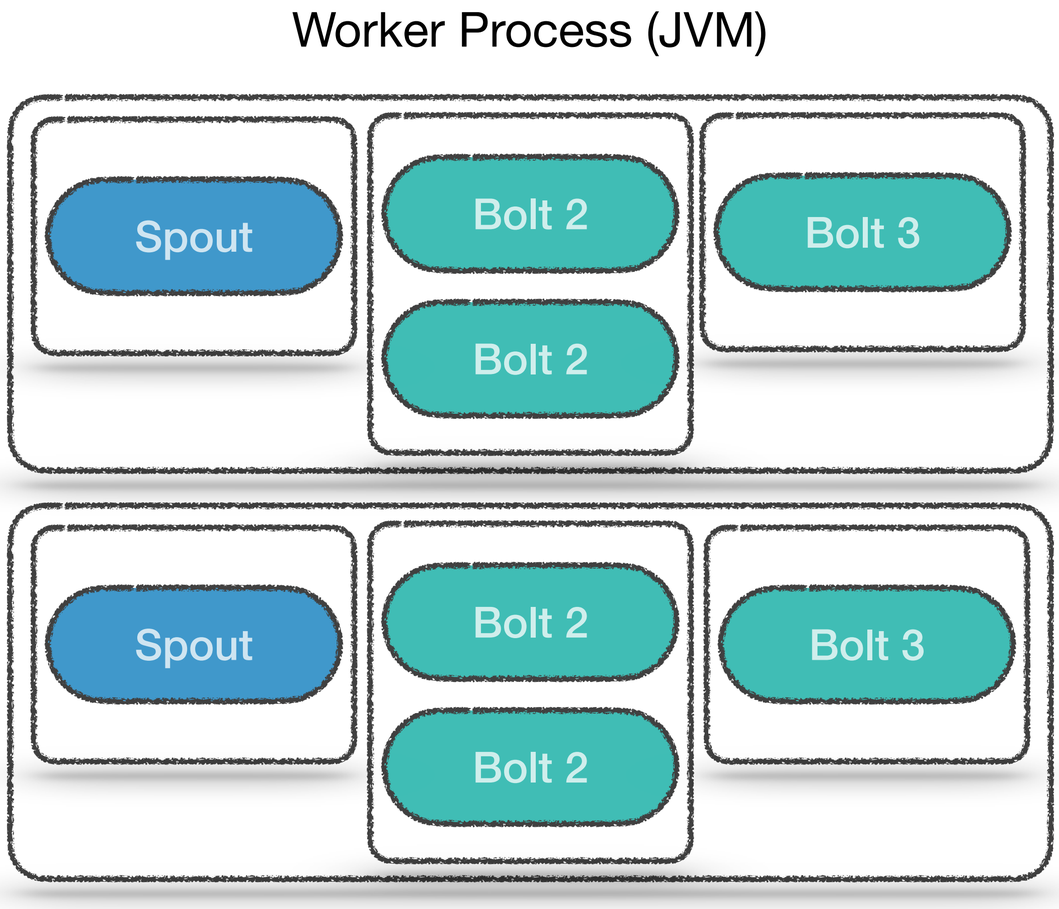


parallelism hint
==
number of executors

```
.setSpout(“Spout Name”, spout, 1)
.setBolt(“Bolt 2 Name”, bolt2, 2).setNumTasks(4)
.setBolt(“Bolt 3 Name”, bolt3, 2)
…
cg.setNumWorkers(1)
```

number of worker processes

Parallelism



parallelism hint
==
number of executors

```
.setSpout(“Spout Name”, spout, 1)
.setBolt(“Bolt 2 Name”, bolt2, 2).setNumTasks(4)
.setBolt(“Bolt 3 Name”, bolt3, 2)
…
cg.setNumWorkers(2)
```

number of worker processes

Stream grouping

Defines how the tuples are sent between instances of spouts and bolts:

- **Shuffle grouping:** Tuples are randomly distributed across bolt instances;
- **Fields grouping:** The stream is partitioned by the fields specified in the grouping;
- **Partial Key grouping:** It’s like Fields grouping, but are load balanced between two downstream bolts;
- **All grouping:** The stream is replicated across all bolt instances;
- **Global grouping:** The entire stream goes to a single one of the bolt's tasks;
- **Direct grouping: T**he producer bolt decides which task of the consumer bolt will receive the tuple;