QCon

www.qconferences.com

基于scala/akka构建响应式流计算

@途牛-谢辉

促进软件开发领域知识与创新的传播

InfoQueue

ArchSummit
全球架构师峰会

[深圳] 2015年7月17日-18日

QCon
全球软件开发大会

[上海] 2015年10月15-17日

关注InfoQ官方微信
及时获取QCon演讲视频信息

议题

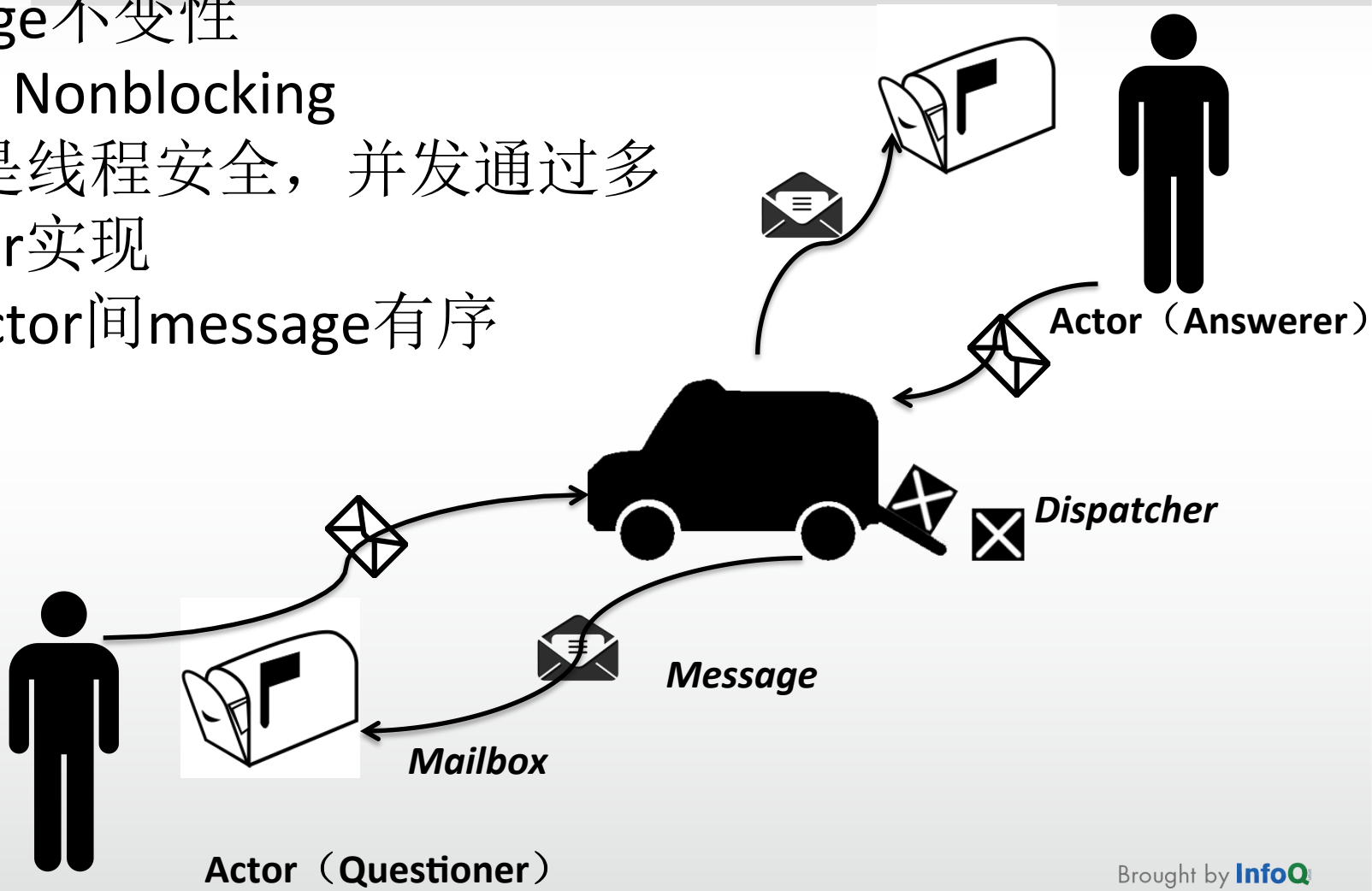
- Akka-actor
- 响应式流计算
- 与Akka-Stream对比
- 性能优化

QCon

AKKA

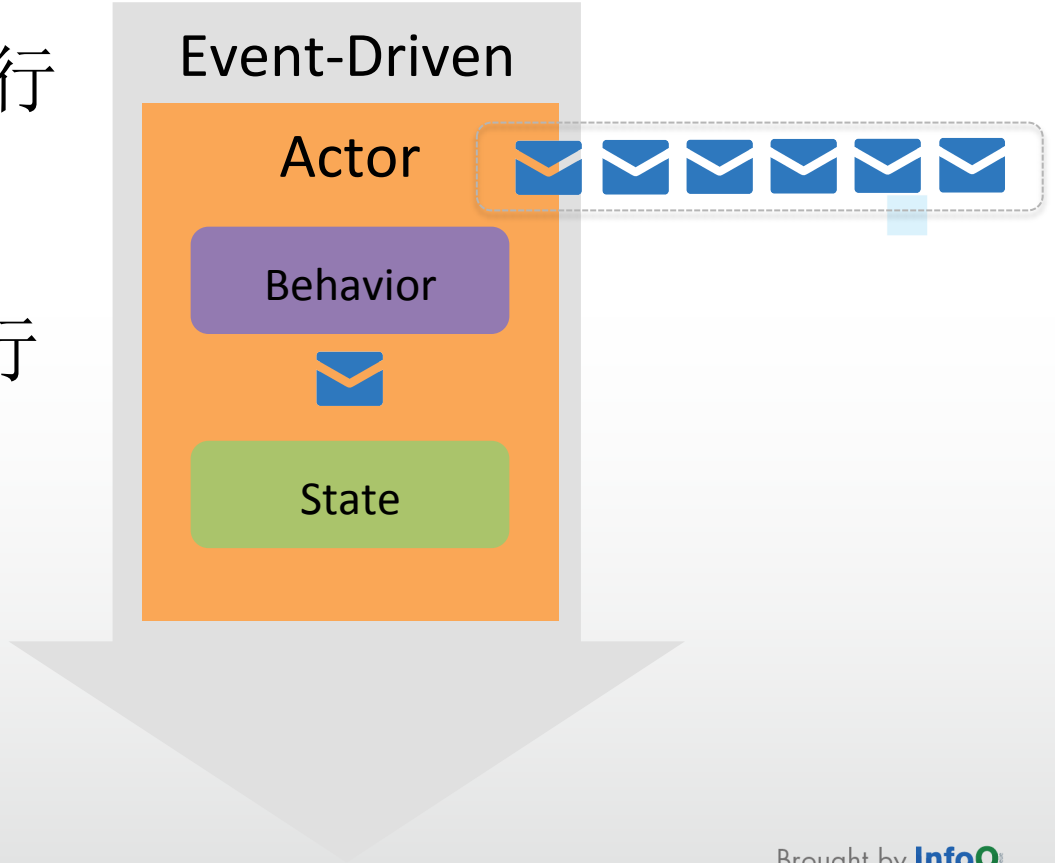
AKKA

- Message不变性
- 异步、Nonblocking
- Actor是线程安全，并发通过多个Actor实现
- 两两actor间message有序



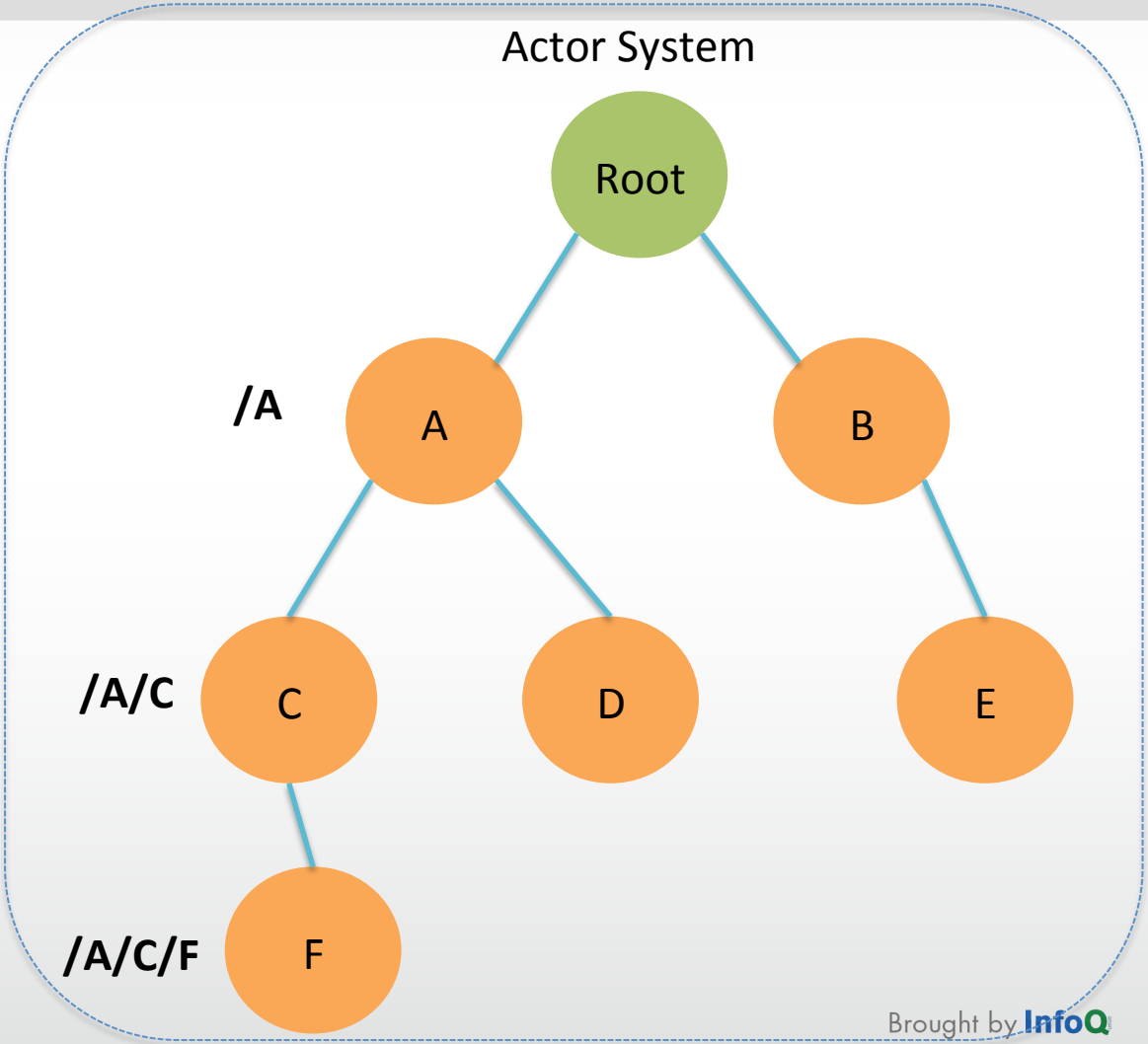
Actor Model

- Mailbox：消息队列
- Behavior：响应消息的行为
- State：FSM
- Run：附着在线程上执行



Actor System

- 树形结构：
  Parent-Children
- Supervisor Strategy
  One-For-One
  All-For-One
- Path
- Dispatcher
  ThreadPool



消息驱动

- NonBlocking（异步）
- 无锁（消息不变性）
- Actor线程安全

- 状态拓扑
- Message通信开销

QCon

响应式流计算

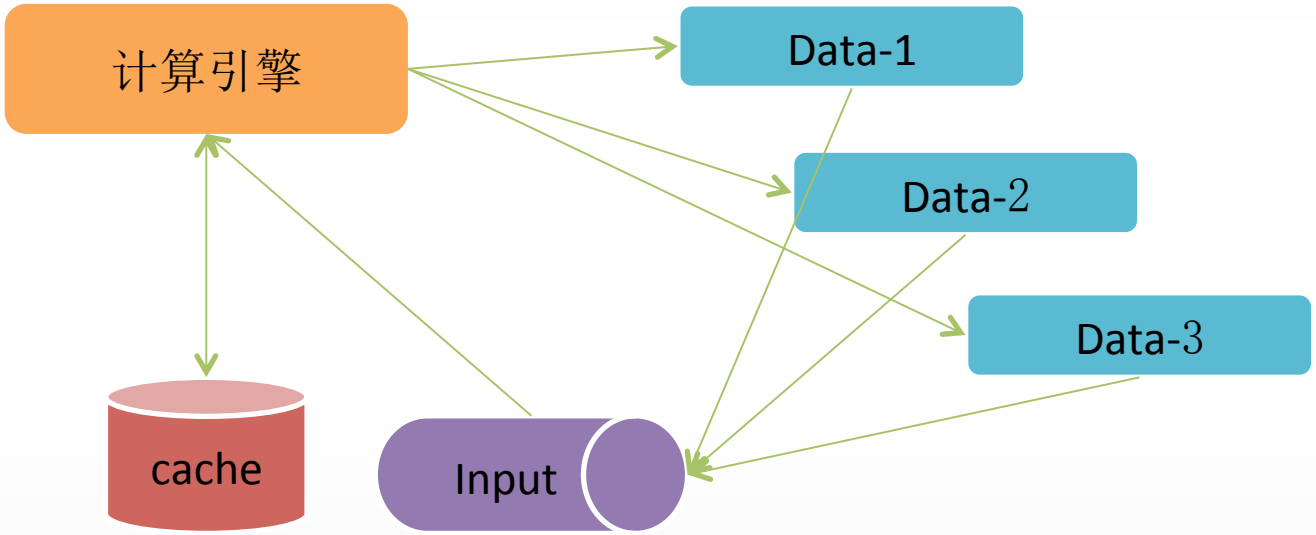
背景

- 场景：列表页
- 资源价格、余位信息变更频繁

  内存索引？
- 不同供应商能力差异

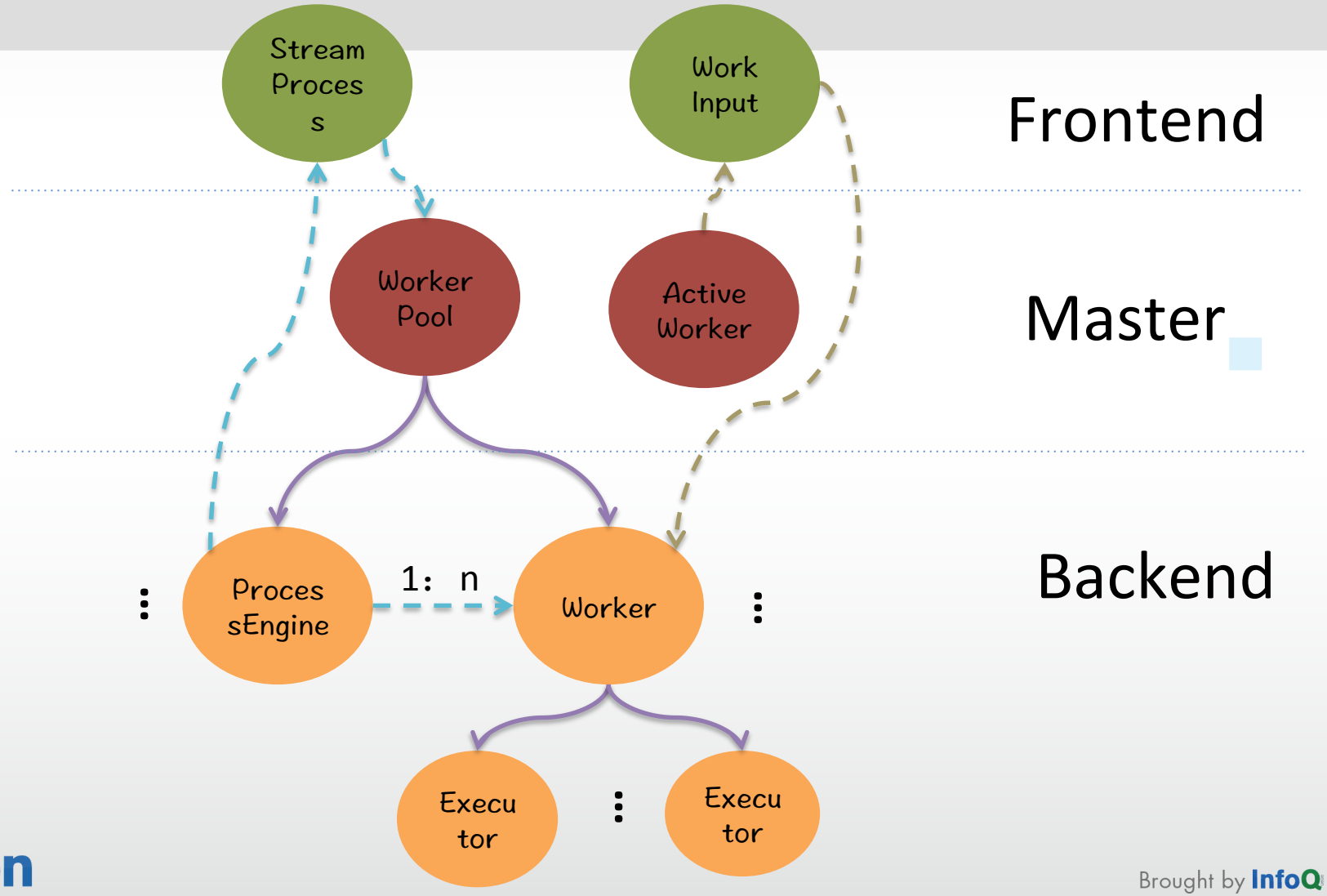
  未命中、实时抓取

流计算

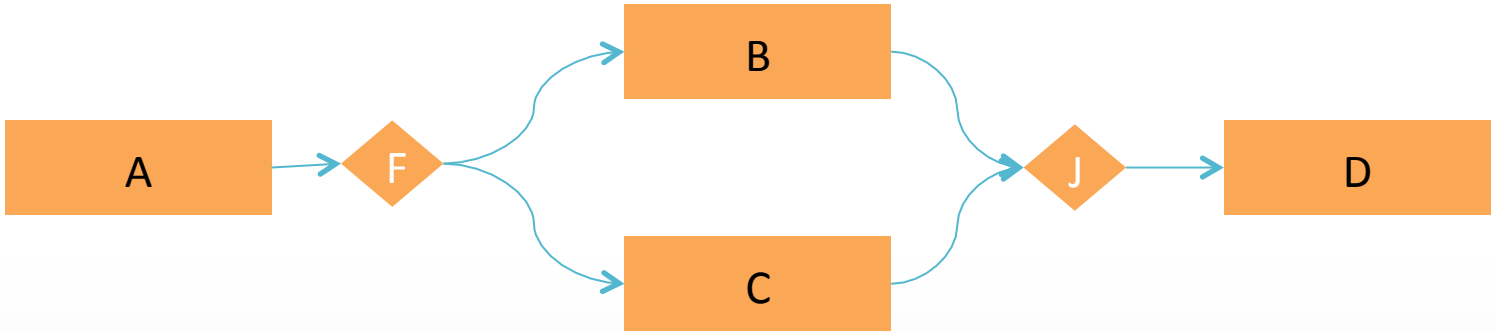


- 数据源：Lucene、cache、RPC…
- Input：流化管道
- 计算：一次Stream的聚合计算

DataFlow


Stream Process
Work Input
Frontend
Worker Pool
Active Worker
Master
Process Engine
1: n
Worker
Backend
Executor
Executor

Process

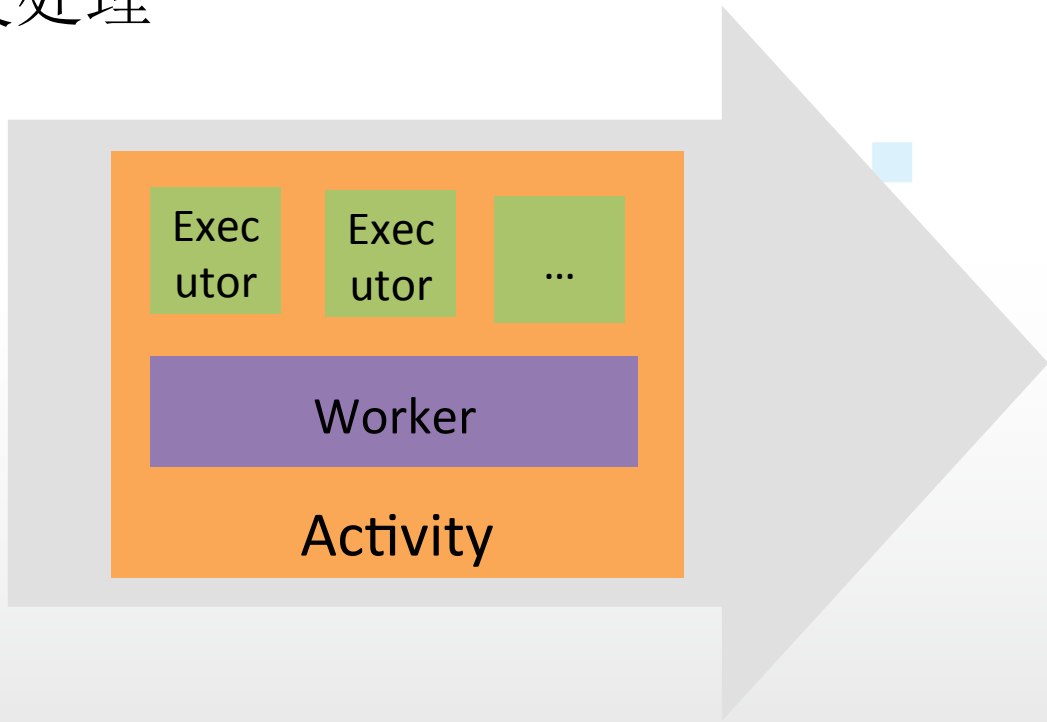


ProcessEngine

WorkActivity

- 行为
  包含一个Worker担当行为人
  多个Executor实现并发处理
- In-Out

IN：
1）前一活动的Result
2）来自WorkInput的Job流



Job Stream

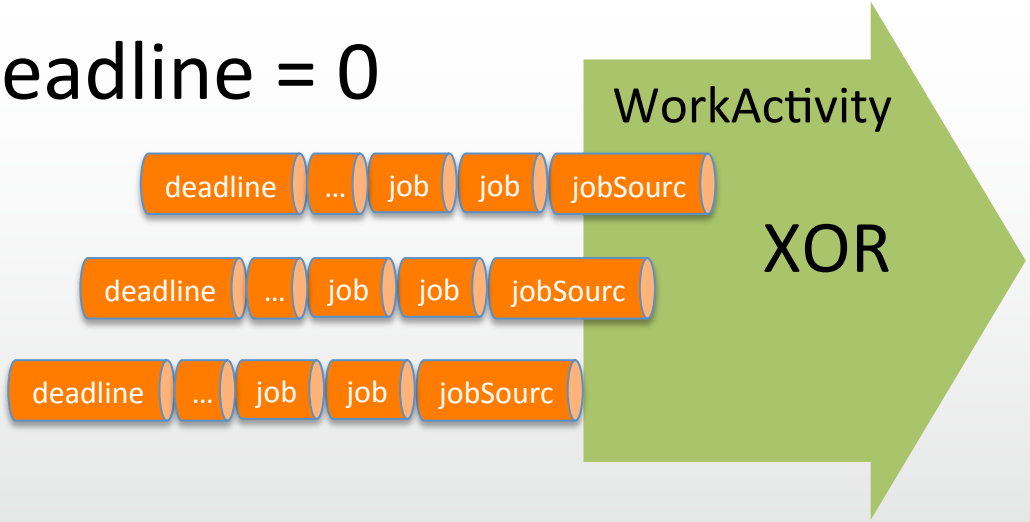
- Session： job流

  JobSource(SessionId, Timeout, …)

  Job …

  Deadline(SessionId)

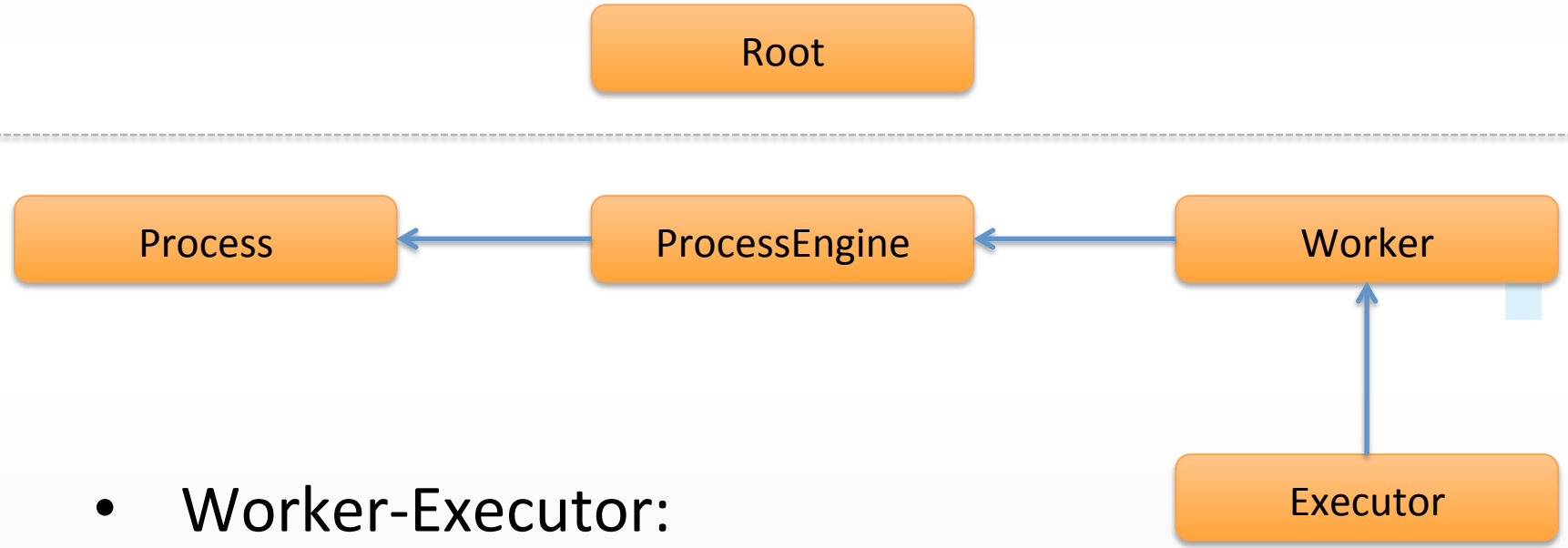
- JobSource XOR Deadline = 0



Worker FSM

```
private[dataflow] class Worker(masterClient: MasterClient) extends Actor
with FSM[WorkerState, WorkerData] {

  import context._

  startWith(Prepares, null)

  when(Prepares) {
    ...
    goto(Ready) using ActivityContext(...)
  }

  when(Ready) {
    ...
  }

  when(Doing) {
    ...
  }

  when(Done) {
    ...
  }

  when(Release) {
    ...
  }

  initialize()

}
```

Exception Propagation



- Worker-Executor:
  One-For-One Strategy
- Process-ProcessEngine-Worker:
  Exception作为Message向外传递

Time Out

- Process Time Out

  ProcessDefinition.timeout(5 seconds)

  Supervisor：StreamProcess

- Session Time Out

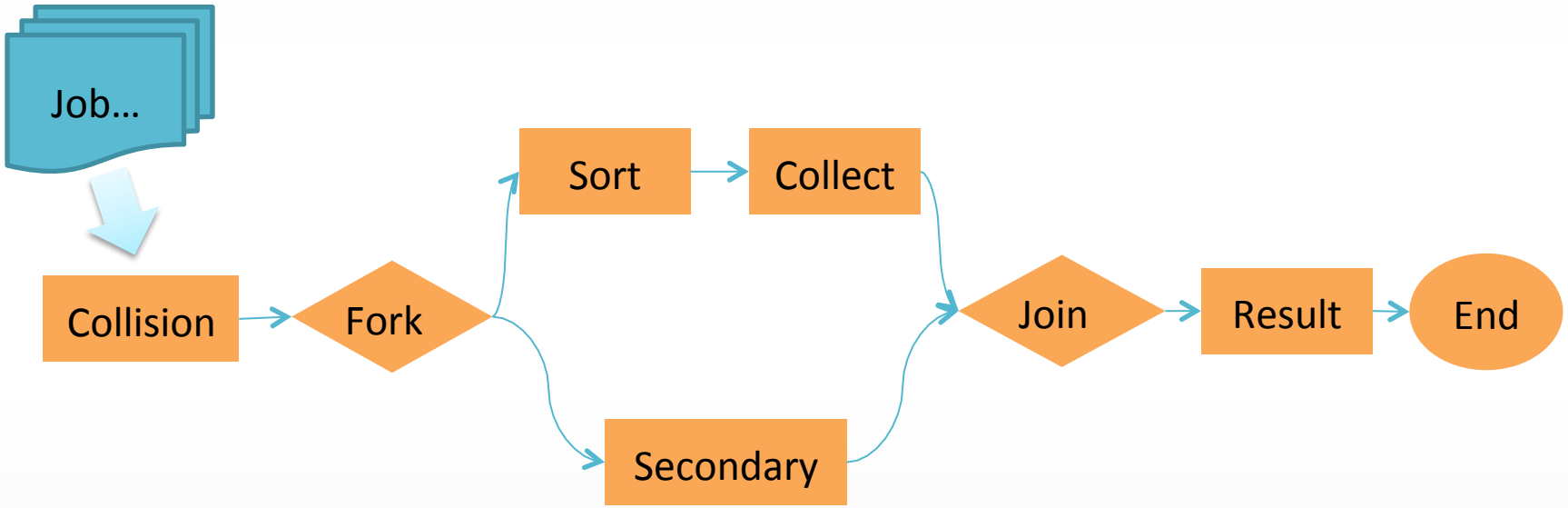
  JobSource（sessionId, 2 seconds, timeoutMsg）

  Supervisor： Worker

  Executor处理timeoutMsg

示例

一个计算流程



两组随机数分别产生1000个1~50的整数，对这两组数进行碰撞，得到碰撞数（该数在两组中都存在，则产生碰撞，碰撞后需从各自集合中移除）的集合，后对集合进行排序，并且计算出碰撞数相同次数分别有是多少。

流程DSL

```
import Flow.DSL._
val pd = Flow("sorted-secondary") {implicit builder =>
  ~~
    >>("collision", Creator[CollisionExecutor], 10)(routees => {
      case msg: Num1 =>
        routees(msg.n / 5)
      case msg: Num2 =>
        routees(msg.n / 5)
    })
  <+("f1")
  ~~
    >>("sorted", Creator[SortedExecutor], 10)(routees => {
      case n: Int =>
        routees(n / 5)
    })
    >>("collect", Creator[CollectExecutor])
  ~~
    >>("secondary", Creator[SecondaryExecutor])
  +>("j1")
    >>("result", Creator[ResultExecutor])
  $
}
```

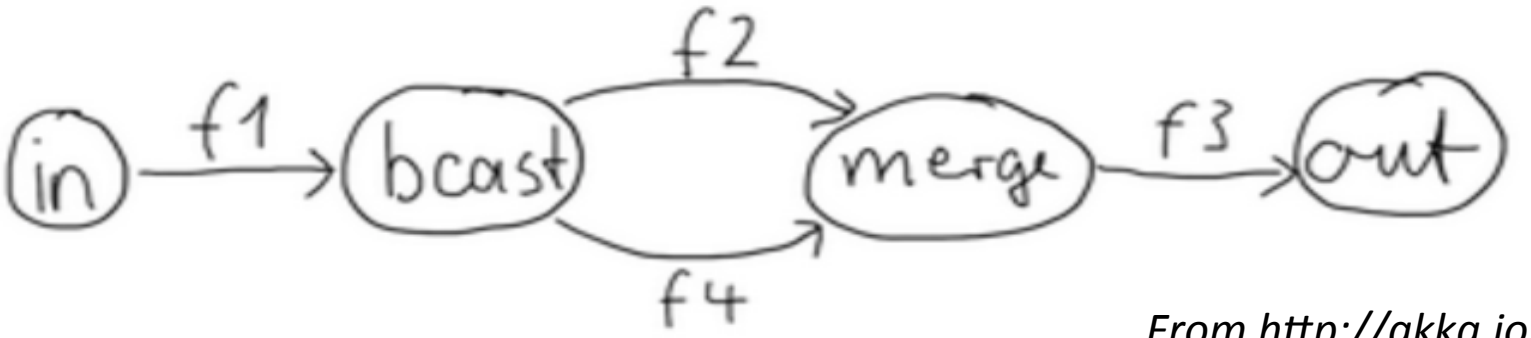
Collision Executor

```
class CollisionExecutor extends Executor {

  private val ns1 = new ListBuffer[Int]()
  private val ns2 = new ListBuffer[Int]()

  override def receive = {
    case msg: Num1 =>
      colliding(msg.n, ns2, ns1)
    case msg: Num2 =>
      colliding(msg.n, ns1, ns2)
  }

  private def colliding(n: Int, compared: ListBuffer[Int], belonged: ListBuffer[Int]) {
    val i = compared.indexOf(n)
    if (i >= 0) {
      compared.remove(i)
      emit(n)
    } else
      belonged += n
  }

}
```

Akka Stream

Akka-Stream

```
val g = FlowGraph.closed() { implicit builder:
FlowGraph.Builder =>
   import FlowGraph.Implicits._
   val in = Source(1 to 10)
   val out = Sink.ignore
   val bcast = builder.add(Broadcast[Int](2))
   val merge = builder.add(Merge[Int](2))
   val f1, f2, f3, f4 = Flow[Int].map(_ + 10)
   in ~> f1 ~> bcast ~> f2 ~> merge ~> f3 ~> out
            bcast ~> f4 ~> merge
}
```



From http://akka.io

似

	dataflow	Akka-stream
计算节点	Activity • OnReady • WorkActivity • EndActivity • ForkActivity/ JoinActivity	Graph:Module/Shape • SourceModule/SourceShape • FlowModule/FlowShape • SinkModule/SinkShape • JunctionModule/ FanInShape、FanOutShape
操作	Executor	Stage： map、filter…
流程	ProcessDefinition	RunnableFlow
运行	Cluster 、 standalone	FlowMaterializer： ActorFlowMaterializer orElse

非

	dataflow	Akka-stream
运算模型	有界流	无界流
运算单元	有状态	无状态
数据源	WorkInput 动态数据源	Source
消息传递	Push message	Back-pressure Diymatic Push/Pull
启动用时	< 20ms	> 100ms

性能调优

DataFlow

- 通信

  Local > Remote

  点到点(WorkInput)

- Actor

  Don't Ask(NonBlocking)

  循环利用

  行为单调

应用优化

- Executor
  Behavior：简单
  NonBlocking：异步库（akka-http、scalaredis…）
  并发：足量保证消费能力，routing规则均衡
- Streaming
  job-Stream：流化均匀
- Thread
  parallelism(fork-join-executor)： min、max、factor
- 数据分类
  计算型数据、渲染型数据

www.infoq.com/cn

InfoQueue

Q & A

软件
正在改变世界！