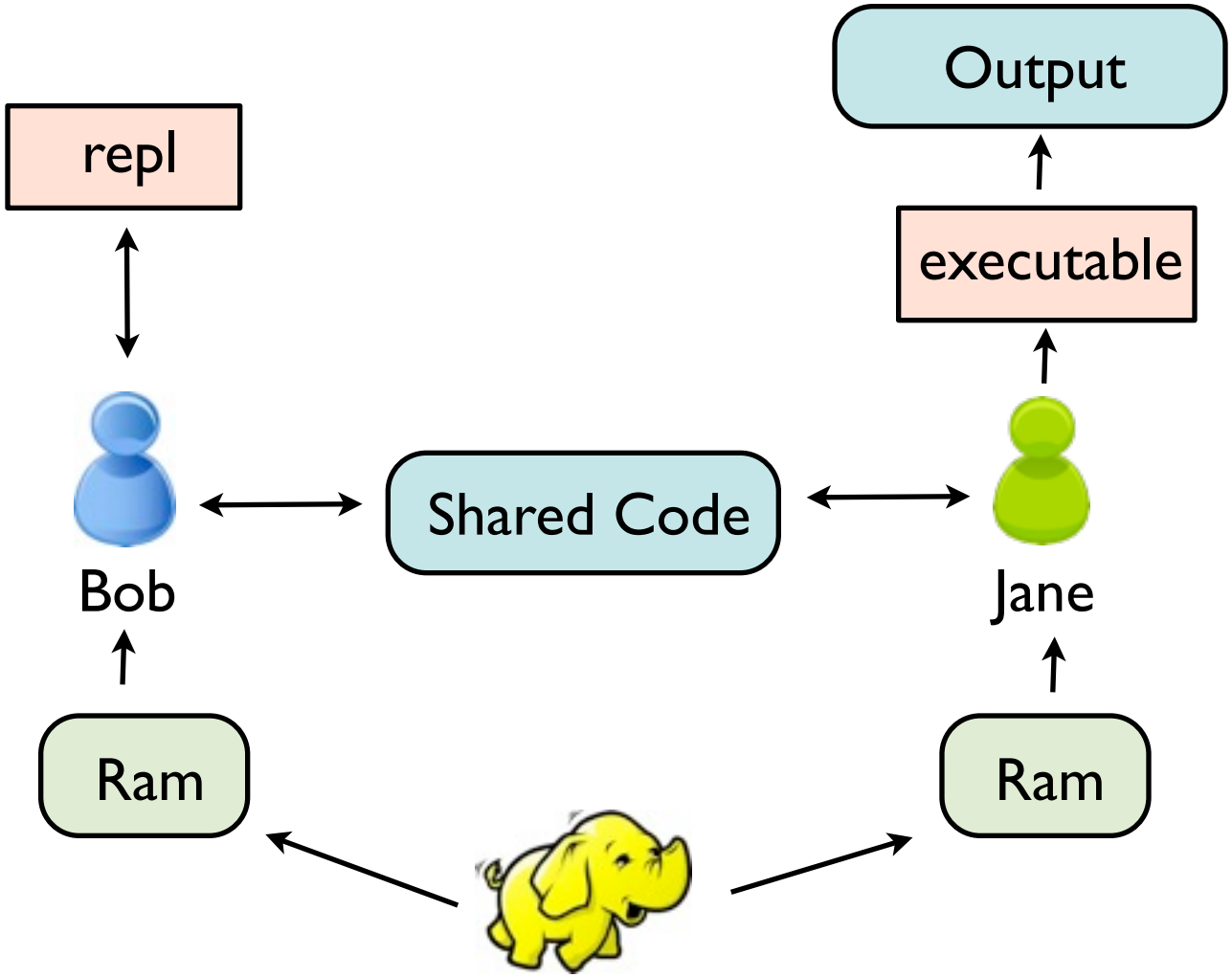
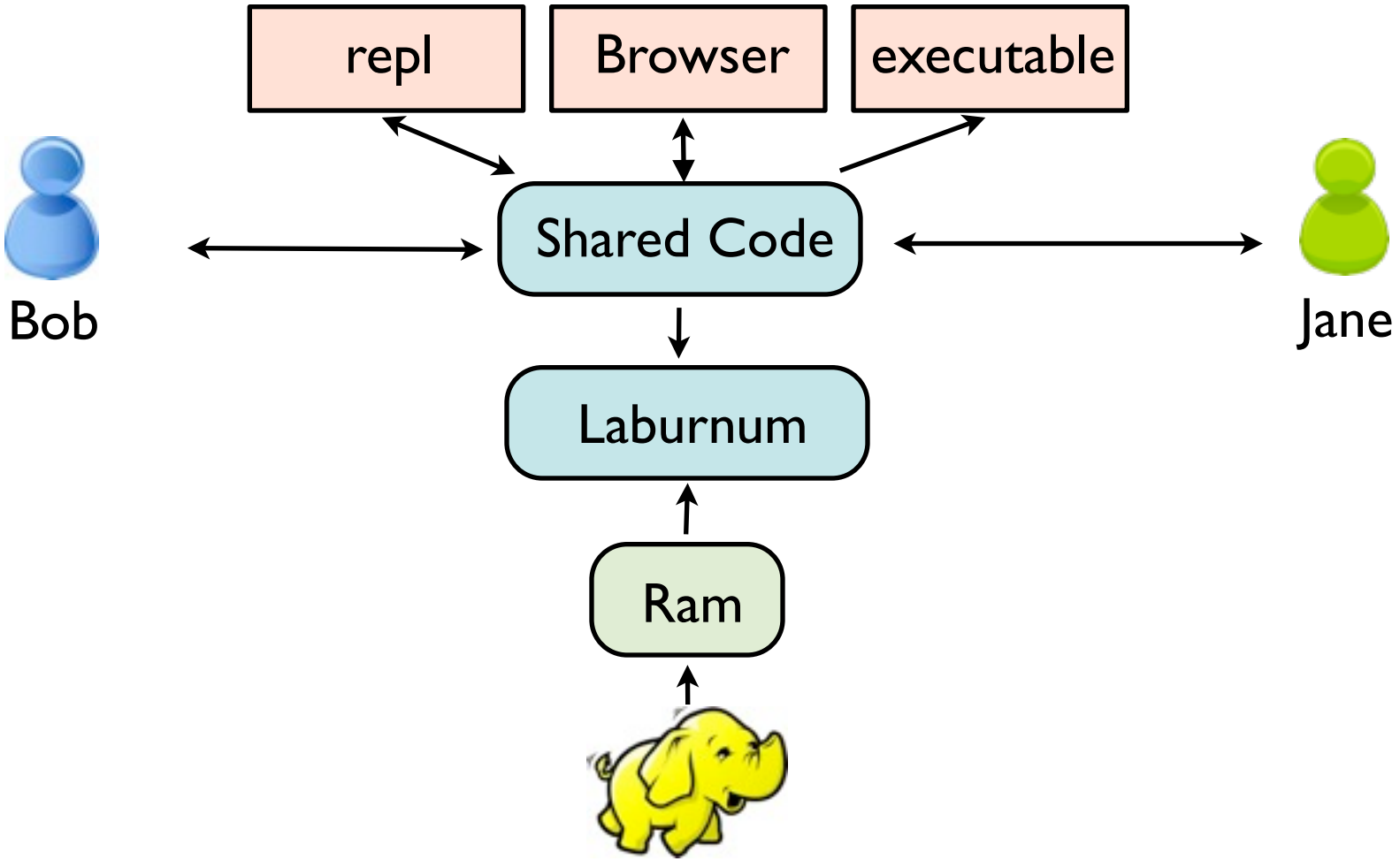
QUANTIFIND

Sharing is Caring:
Enabling Data Science Teams with Laburnum

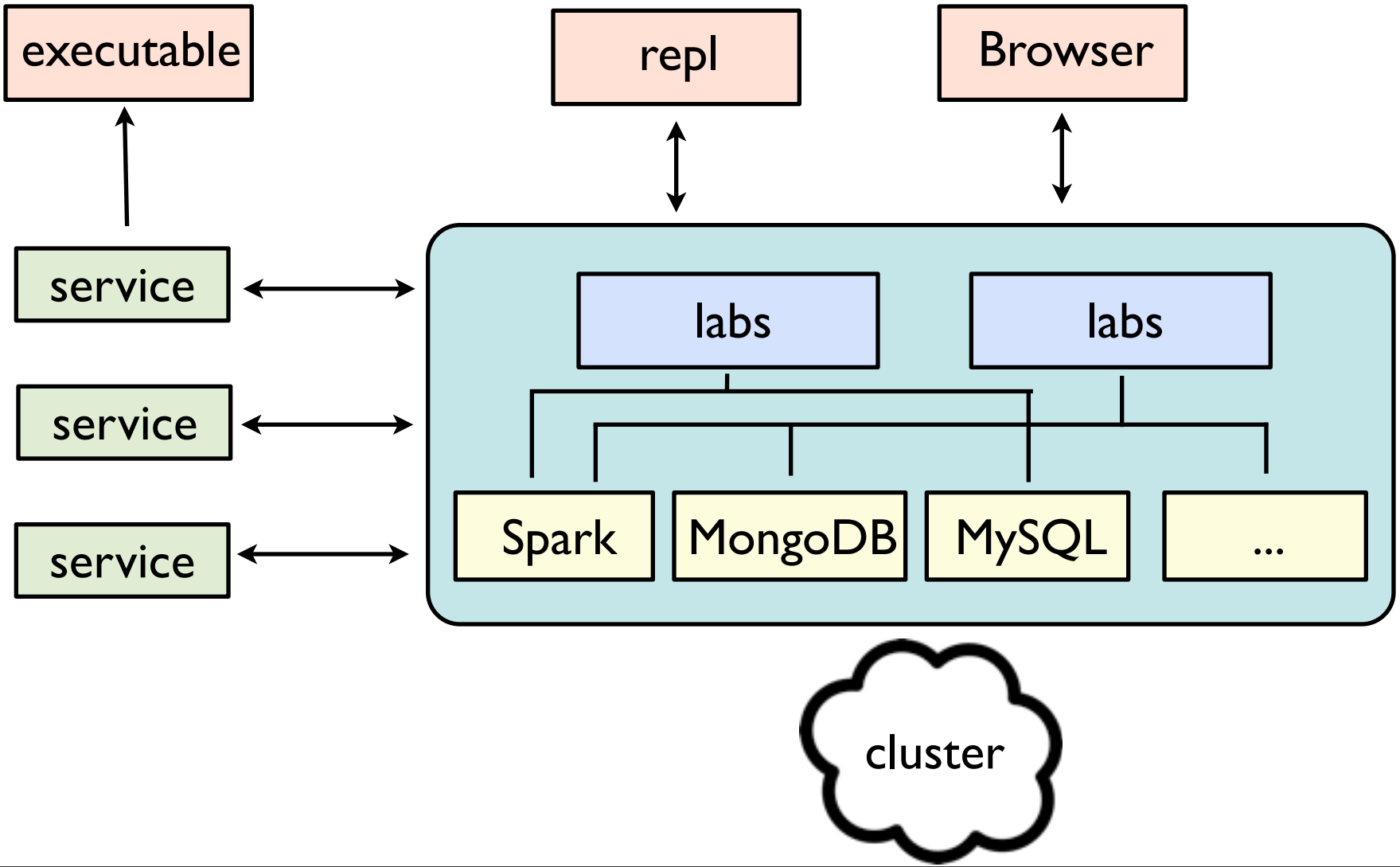
Multi-user development environment



Laburnum development environment



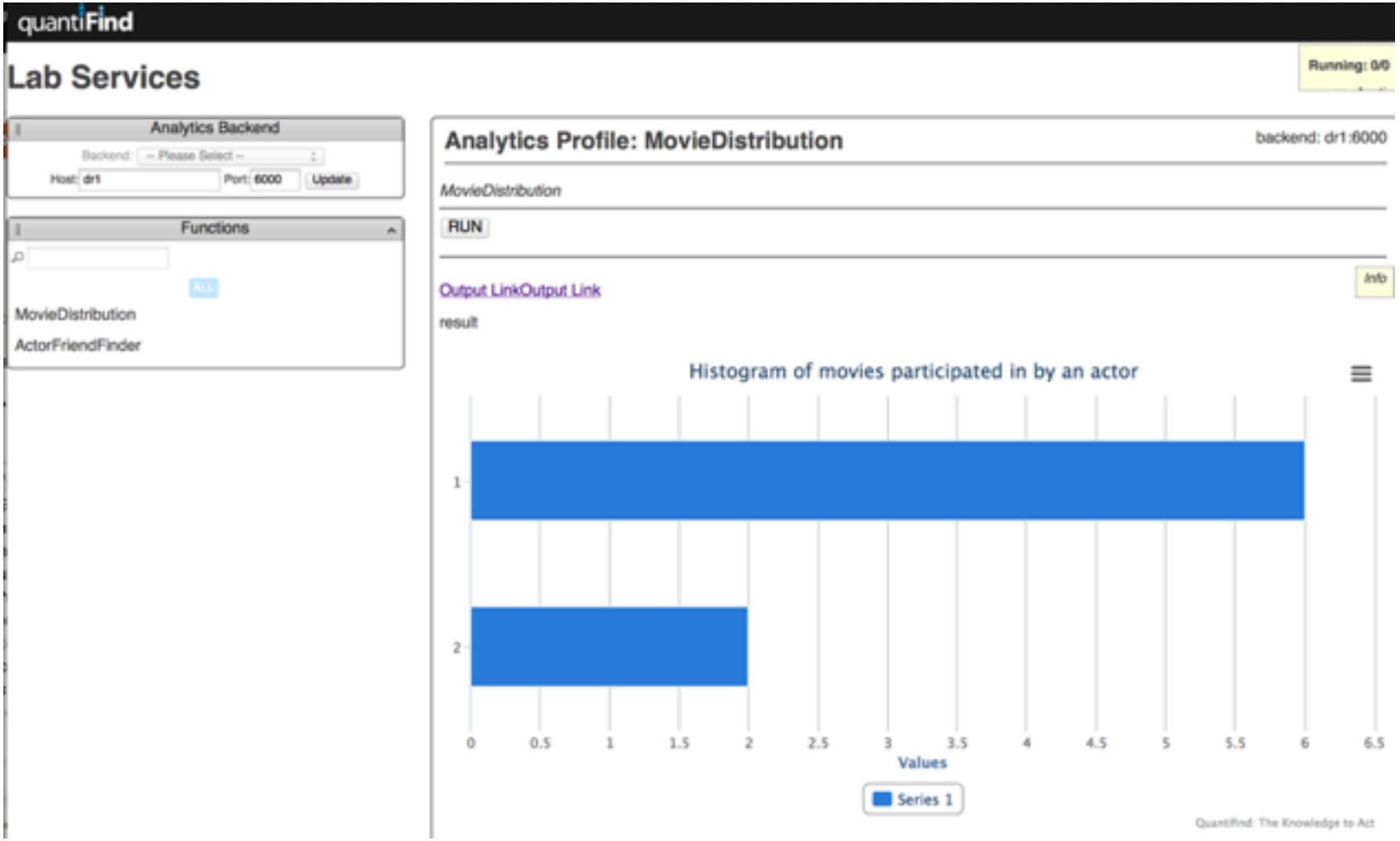
Laburnum framework



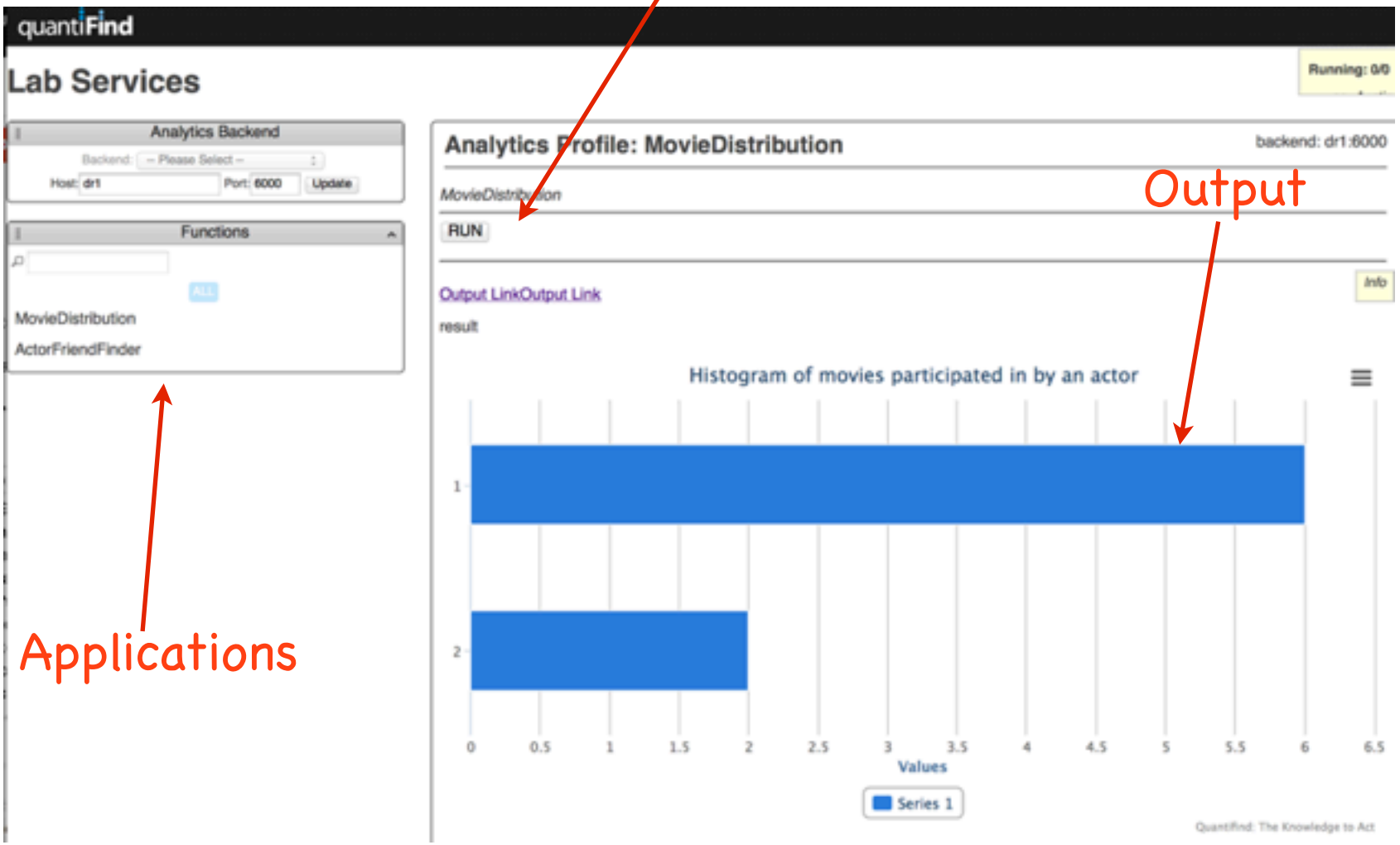
Overview

- Simple to use applications for end-users
- Simple to write applications for data scientists
  - Minimal boilerplate
  - Intuitive information sharing with Sumac argument parsing
  - Handling cluster and memory management outside of application
- Powerful flexibility
  - Visualization tools
  - **Shared resources**
  - Dynamically issuing spark queries on shared, cached data
  - Issuing commands in a shell
  - Mixing in other technologies

Web Interface



Web Interface



Simple

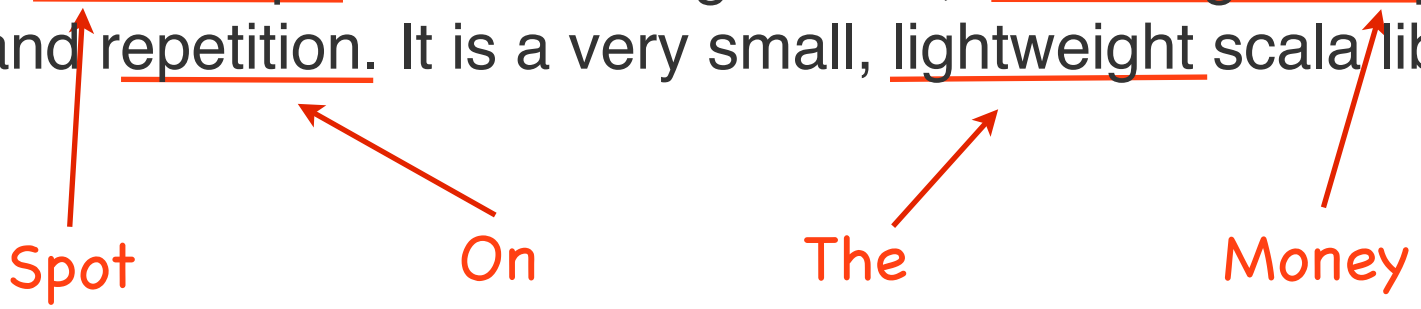
Sumac

"**Sumac** is a command line option parser and library. It tries to differentiate itself from other libraries by making it dead simple to define arguments, removing boilerplate and repetition. It is a very small, lightweight scala library."

We use Sumac to easily and consistently pass information to applications

Sumac

"**Sumac** is a command line option parser and library. It tries to differentiate itself from other libraries by making it dead simple to define arguments, removing boilerplate and repetition. It is a very small, lightweight scala library."



http://blog.quantifind.com/posts/Sumac/

A complete & working example of labs

```scala
/**
 * A simple Labs example which loads a list of actors, caches and exposes them as an RDD
 */
class ActorPlan(actorPlanArgs: ActorPlanArgs) extends Laburnum(actorPlanArgs) with SparkPlan {
  def sparkPlanArgs = actorPlanArgs

  val args = actorPlanArgs
  val basePath = "rserver"

  // hide file argument from applications
  override def argFilterSet = super.argFilterSet ++ Set("actorsFile")

  // expose data to applications
  override def addExtraArgs[T <: FieldArgs](args: T) {
    super.addExtraArgs(args)
    ifType[ActorPlanArgs](args) {actorArgs =>
      actorArgs.actors = actorPlanArgs.actors
    }
  }

  // load data
  args.actors = LoadActors(args.actorsFile)

  // cache & interact with data
  println(args.actors.cache.count)
}

// Create a running server
object ActorPlan extends App {
  val serverArgs = new ActorPlanArgs
  serverArgs.parse(args)
  new LaburnumServer(serverArgs, new ActorPlan(serverArgs))
}

class ActorPlanArgs extends LaburnumServerArgs with SparkPlanArgs {
  var actorsFile: String = _ // this is a Sumac argument

  // store, get, and set data
  private var _actors: Option[RDD[Actor]] = None
  def actors: RDD[Actor] = _actors.getOrElse(throw new Exception("Actors RDD failed to load"))
  def actors_= (__actors: RDD[Actor]): Unit = _actors = Some(__actors)
}
```

A complete & working example of labs

```scala
/**
 * A simple Labs example which loads a list of actors, caches and exposes them as an RDD
 */
class ActorPlan(actorPlanArgs: ActorPlanArgs) extends Laburnum(actorPlanArgs) with SparkPlan {
  def sparkPlanArgs = actorPlanArgs

  val args = actorPlanArgs
  val basePath = "rserver"

  // hide file argument from applications
  override def argFilterSet = super.argFilterSet ++ Set("actorsFile")

  // expose data to applications
  override def addExtraArgs[T <: FieldArgs](args: T) {
    super.addExtraArgs(args)
    ifType[ActorPlanArgs](args) {actorArgs =>
      actorArgs.actors = actorPlanArgs.actors
    }
  }

  // load data
  args.actors = LoadActors(args.actorsFile)

  // cache & interact with data
  println(args.actors.cache.count)
}

// Create a running server
object ActorPlan extends App {
  val serverArgs = new ActorPlanArgs
  serverArgs.parse(args)
  new LaburnumServer(serverArgs, new ActorPlan(serverArgs))
}

class ActorPlanArgs extends LaburnumServerArgs with SparkPlanArgs {
  var actorsFile: String = _ // this is a Sumac argument

  // store, get, and set data
  private var _actors: Option[RDD[Actor]] = None
  def actors: RDD[Actor] = _actors.getOrElse(throw new Exception("Actors RDD failed to load"))
  def actors_= (__actors: RDD[Actor]): Unit = _actors = Some(__actors)
}
```

Spark

Expose RDD

Load & Cache

Sumac args

Starting a labs instance

```
export SPARK_MEM=3G

CLASS=com.qf.explore.imdb.ActorPlan

ROOT=/home/austin

LABS_JAR=$ROOT/Laburnum/explore/target/scala-2.10/explore-assembly-0.1-SNAPSHOT.jar

JOB_JARS=$LABS_JAR

java -Djava.library.path=/opt/mapr/lib:/usr/local/lib/libmesos.so \
     -Dspark.akka.frameSize=50 -Dspark.akka.timeout=60 \
     -Dsun.io.serialization.extendedDebugInfo=true \
     -Dspark.storage.blockManagerHeartBeatMs=300000 \
     -Dspark.serializer=org.apache.spark.serializer.KryoSerializer \
     -Dspark.kryo.registrator=com.qf.util.kryo.TransformKryoRegistrator \
  -cp $SPARK_CLASSPATH:$JOB_JARS \
  $CLASS \
  --mesosMaster local[2] \
  --root $ROOT/Laburnum/ \
  --frameworkDescription Actor-Labs \
  --servicePort 6000 \
  --serviceJarDir $ROOT/actor-services/ \
  --dynamicJarLoading true \
  --serviceJarPackage com.qf.explore.imdb \
  --actorsFile file:///home/austin/data/tiny_actors.psv \
```

Starting a labs instance

```
export SPARK_MEM=3G

CLASS=com.qf.explore.imdb.ActorPlan

ROOT=/home/austin

LABS_JAR=$ROOT/Laburnum/explore/target/scala-2.10/explore-assembly-0.1-SNAPSHOT.jar

JOB_JARS=$LABS_JAR

java -Djava.library.path=/opt/mapr/lib:/usr/local/lib/libmesos.so \
     -Dspark.akka.frameSize=50 -Dspark.akka.timeout=60 \
     -Dsun.io.serialization.extendedDebugInfo=true \
     -Dspark.storage.blockManagerHeartBeatMs=300000 \
     -Dspark.serializer=org.apache.spark.serializer.KryoSerializer \
     -Dspark.kryo.registrator=com.qf.util.kryo.TransformKryoRegistrator \
  -cp $SPARK_CLASSPATH:$JOB_JARS \
  $CLASS \
  --mesosMaster local[2] \
  --root $ROOT/Laburnum/ \
  --frameworkDescription Actor-Labs \
  --servicePort 6000 \
  --serviceJarDir $ROOT/actor-services/ \
  --dynamicJarLoading true \
  --serviceJarPackage com.qf.explore.imdb \
  --actorsFile file:///home/austin/data/tiny_actors.psv \
```

Only labs author deals with cluster, memory, and filesystem

A complete & working example of a service

```
/**
 * Given an actor, finds actors that appear together and sort by frequency
 */
object ActorFriendFinder extends Service[ActorFriendFinderArgs] {
  def apply(args: ActorFriendFinderArgs) = {
    val name = args.name
    val movieSet = args.actors.filter(_.name == name).collect().head.movies.toSet
    val friends = args.actors.flatMap(actor => {
      val appearances = actor.movies.toSet.intersect(movieSet).size
      appearances match {
        case 0 => None
        case _ => Some(ActorFriend(actor.name, appearances))
      }
    })
    Map("Number of appearances an actor shares with %s".format(name) -> friends.collect().sortBy(-1 * _.count)
  }
}

case class ActorFriendFinderArgs() extends ActorPlanArgs {
  var name: String = _ // a Sumac argument
}

// Wrap output into a case class
case class ActorFriend(name: String, count: Int)
```

A complete & working example of a service

```scala
/**
 * Given an actor, finds actors that appear together and sort by frequency
 */
object ActorFriendFinder extends Service[ActorFriendFinderArgs] {
  def apply(args: ActorFriendFinderArgs) = {
    val name = args.name
    val movieSet = args.actors.filter(_.name == name).collect().head.movies.toSet
    val friends = args.actors.flatMap(actor => {
      val appearances = actor.movies.toSet.intersect(movieSet).size
      appearances match {
        case 0 => None
        case _ => Some(ActorFriend(actor.name, appearances))
      }
    })
    Map("Number of appearances an actor shares with %s".format(name) -> friends.collect().sortBy(-1 * _.count)
  }
}

case class ActorFriendFinderArgs() extends ActorPlanArgs {
  var name: String = _ // a Sumac argument
}

// Wrap output into a case class
case class ActorFriend(name: String, count: Int)
```

Access data

Sumac args

Output as a table

Power and Flexibility

Flexible

```
trait Service[T <: FieldArgs] extends ArgFunction[T, Any] with Logging with ArgMain[T] {
  // Call apply(args) when run stand-alone
  def main(args: T) = apply(args)
```

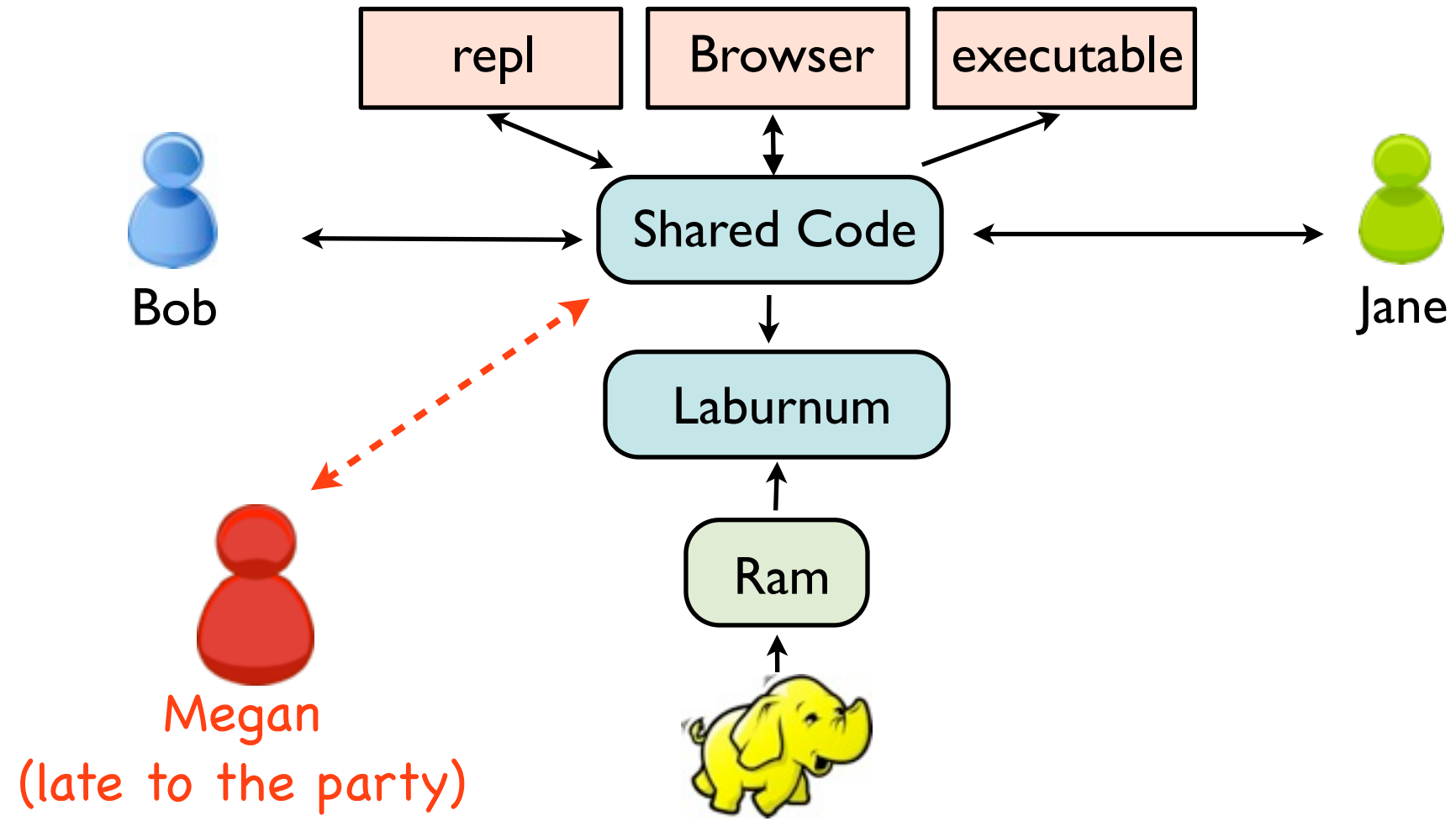
Runnable object independent of labs

Flexible

```
class SocialSourcePlan(args: SocialSourcePlanArgs)
  extends QCollectionPlan(args)
  with SparkPlan
  with MongoPlan {
```

Mix and match technologies

Laburnum development environment



Powerful

```
scala> args.actors.count

scala> 1852815
```

(1) Code in shell

(2) Boilerplate added & compiled

(4) Answer returns

```
[info] Packaging /Users/austin/IdeaProjects/Laburnum/explore/ta
rget/scala-2.10/explore_2.10-0.1-SNAPSHOT.jar ...
[info] Done packaging.
[success] Total time: 6 s, completed Dec 1, 2013 5:21:21 AM
```

```
13/12/01 05:21:25 INFO spark.SparkContext: Job finished: count
at ServiceTemplateYwZVodDBuOJWoYhe.scala:16, took 3.906018 s
```

(3) Job runs successfully

Conclusion

- Simple
- Flexible and Powerful
- **Share resources amongst users**
- Open Source
  - early 2014 (get in touch for early access)
  - Modularize interpreter and web API
  - Support common visualization libraries

QUANTIFIND

austin@quantifind.com