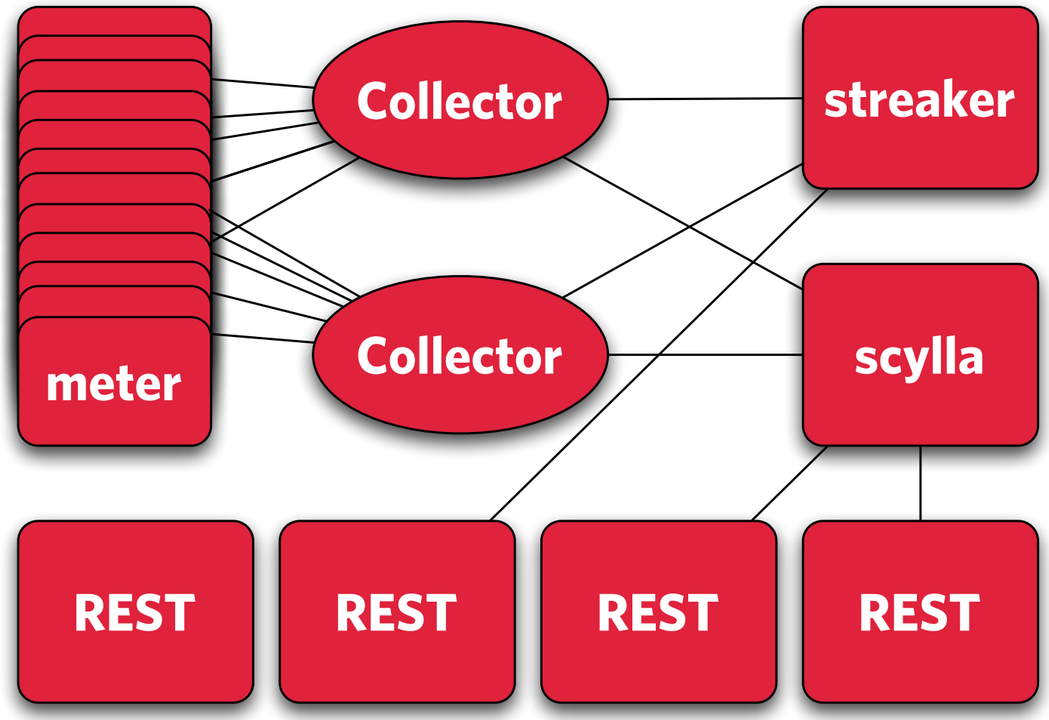
Building Polyglot Systems With Scalang

Cliff Moon
@moonpolysoft

bunday

Why Scala and Erlang?

- Complementary sets of features.
- Compatible type systems.
- Separates concerns.



Boundary Architecture

bunday

First Revision

- JInterface and wrappers.
- Cumbersome code.
- Wrong level of abstraction.
- Not performant.

JInterface Wrappers

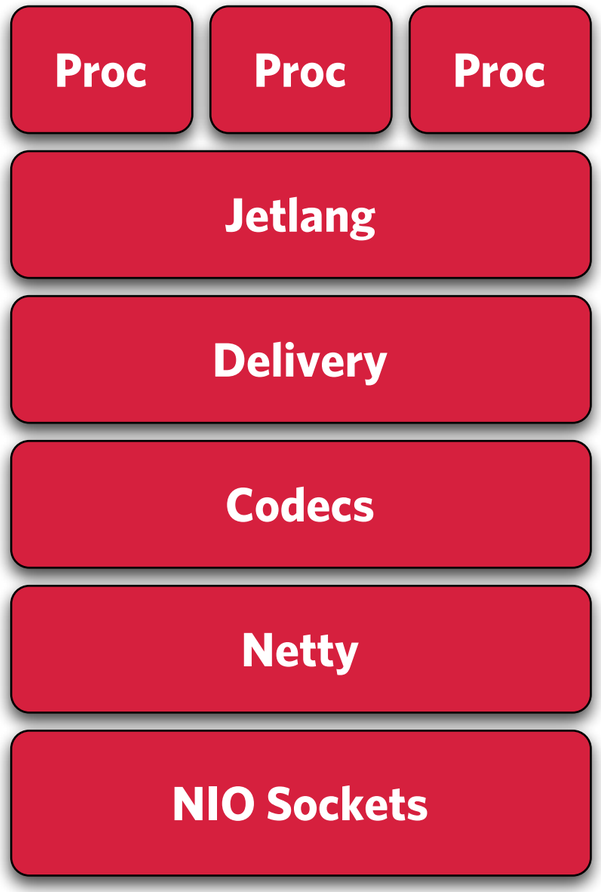
```
def task(m : OtpErlangObject, mbox : OtpMbox) = m match {
  case ETuple(EAtom("cluster"), pid : Pid, ref : Ref) =>
```

bunday

Scalang

- Correctness of behavior.
- Performance.
- Simplicity.

bunday



Scalang Architecture

bandcamp

Using Scalang

Node

```
//make sure epmd is running
val node = Node(“scala@localhost.local”, “cookie”)
```

bunday

Processes

```
class Echo(ctx : ProcessContext) extends Process(ctx) {
   override def onMessage(msg : Any) = msg match {
     case (pid : Pid, m : String) =>
        pid ! m
   }
}
val echoPid = node.spawn[Echo](“echo_server”)
```

bunday

Sending Messages

- Send messages to a Pid.
- Send messages to a local registered name.
- Send messages to a remote registered name.

Sending Messages - in process

```
aPid ! ‘do_something

‘regname ! ‘do_another_thing

(‘regname, ‘erl@localhost.local) ! ‘do_something_else
```

bunday

Sending Messages - outside proc

```
node.send(aPid, ‘do_something)

node.send(‘regname, ‘do_another_thing)

node.send( (‘regname, ‘erl@localhost.local), ‘do_something_else)
```

bunday

Error Handling

- Uncaught exceptions in process code.
- Implemented via bi-directional links.
- Link breakage triggers exit notification.
- Links work both intra and inter - node.
- Not pre-emptive on the Scalang side.

Error Handling

```
class ExitHandler(ctx : ProcessContext) extends Process(ctx) {
  override def trapExit(from : Pid, msg : Any) {
    log.warn("exit notification from %s", from)
  }
}
```

bounday

Type Mappings

From Erlang	To Scala	From Scala	To Erlang
Small Integer	Int	Byte	Small Integer
Integer	Int	Int	Integer
Float	Double	Long	Small Bignum
Boolean	Boolean	Double	Float
Atom	Symbol	Symbol	Atom
Reference	Reference	Reference	Reference
Port	Port	Port	Port
Pid	Pid	Pid	Pid
Small Tuple	Tuple	Fun	Fun
Large Tuple	BigTuple	String	String
String	String	List	List
List	List	BigInteger	Large Bignum
Binary	ByteBuffer	Array[Byte]	Binary
Small Bignum	Long	ByteBuffer	Binary
Large Bignum	BigInt	BitString	Bitstring
Fun	Fun	Tuple	Tuple
Bitstring	Bitstring	BigTuple	Tuple

Rich Type Mappings

- Invoked for tagged tuples.
- User-supplied decode logic.
- Avoids performance penalty of reflective instantiation.

Rich Type Mappings

```
('struct, List(
  ('key, value),
  ('key2, value2),
  ... ))
```

```
{struct, [
  {key, Value},
  {key2, Value2},
  ... ]}
```

Rich Type Mappings

```
object StructFactory extends TypeFactory {
  def createType(name : Symbol, arity : Int, reader : TermReader) : Any
  = {
    reader.mark
    (name, arity) match {
      case (‘struct,2) => Some(readMap(reader))
      case _ => reader.reset; None
    }
  }
}
```

bunday

Rich Type Mappings

```
val node = Node(name,cookie,NodeConfig(
                                    typeFactory = StructFactory))
```

bunday

Services

- Initialization parameters.
- Built in call and cast support.
- Transparent interoperability with gen_server.

Services

- handleCall - gen_server:call/2 - request & respose
- handleCast - gen_server:cast/2 - request
- handleInfo - Raw message received.

Services

```
case class EchoArgs(name : Symbol)
class Echo(ctx : ServiceContext[EchoArgs]) extends Service(ctx) {
  val EchoArgs(name) = ctx.args

  override def handleCall(from : (Pid,Reference), req : Any) : Any = {
    (name, req)
  }
}
```

bunday

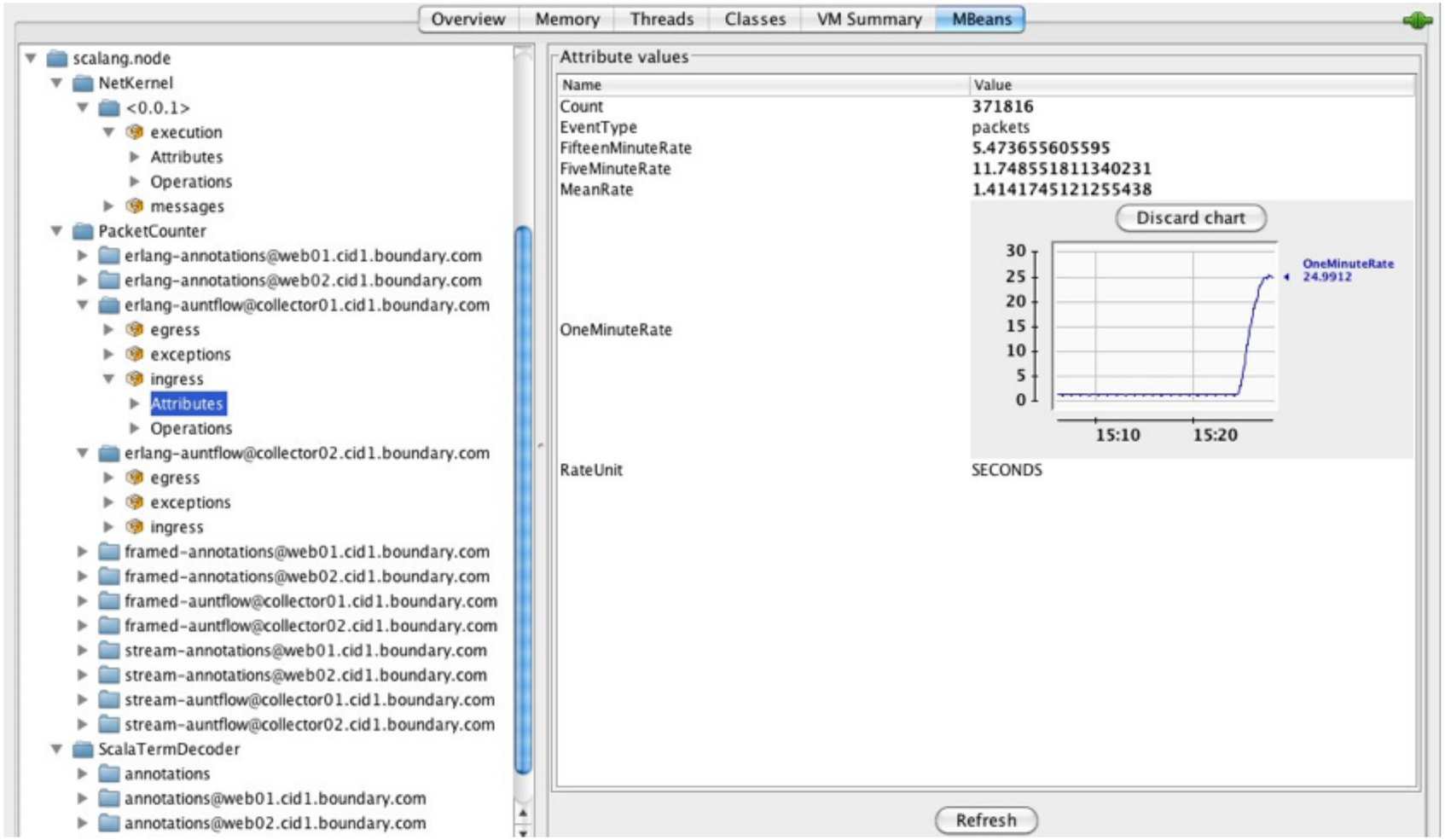
Anonymous Processes

```
node.spawn { mbox =>
  val msg = mbox.receive
  // do some long running work
  node.send('master, (results, mbox.self) )
}
```

bounday

Runtime Metrics

- Message meters per connection.
- Execution histograms per process.
- Serialization time histograms.
- Message queue size gauges.
- Internal thread pool metrics.



Runtime Metrics

Performance Tuning

- ThreadPoolFactory - pluggable thread pool implementations.
- Elastic thread pools from overlock.
- Threads tuned to max(8, cpu_count * 2).
- Beware of starvation.

Thread Pools

- Boss pool - Initial connection and accept handling.
- Worker pool - Non blocking reads and writes.
- Actor pool - Process callbacks.
- Batch Executor - Per-process execution logic.

Thread Pools

```
val node = Node(name,cookie,NodeConfig(
                                    poolFactory = MyThreadPools))
```

Process Patterns

Deferred Reply

```
def handleCall(from : (Pid, Reference), msg : Any) : Any = {
  ctx.node.spawn { mbox =>
    // long running work
    reply(from, response)
  }
  'noreply
}
```

bunday

State Machine

```
class Stateful(ctx : ProcessContext) extends Process(ctx) {
  @volatile var state = ‘a
  override def onMessage(msg : Any) = (msg, state) match {
    case (‘event, ‘a) => ...
  }
}
```

bunday

Worker Pools

```
class StatelessWorker(ctx : ProcessContext) extends Process(ctx) {
  def onMessage(msg : Any) = msg match { ... }
}

//experimental!
node.spawn[StatelessWorker](reentrant = true)
```

bunday

Supervision Trees

```
class TaskMaster(ctx : ProcessContext) extends Process(ctx) {
  def onMessage(msg : Any) = {
    //distribute work to workers
  }

  override def handleExit(worker : Pid, reason : Any) {
    //remove worker from pool, or restart
  }
}
```

bunday

Admin Endpoint

```
class Admin(ctx : ServiceContext[Args]) extends Service(ctx) {
  def handleCall(from: (Pid,Reference), req : Any) = req match {
    case (‘reassign, node : Symbol) =>
      //reassign message stream to new node
  }
}
```

bunday

Admin Endpoint

```
ok = gen_server:call({admin, backend@data01},
                                        {reassign, cruncher@data02}).
```

Demo!

bounday

Conclusion

- Wrap services in Scalang actors.
- Throw out your RPC codegen tools.
- Embrace a mesh topology.
- Let Erlang and Scala do the things they are good at.

Questions?

https://github.com/boundary/scalang
https://github.com/boundary/overlock
Thank you.