fastip

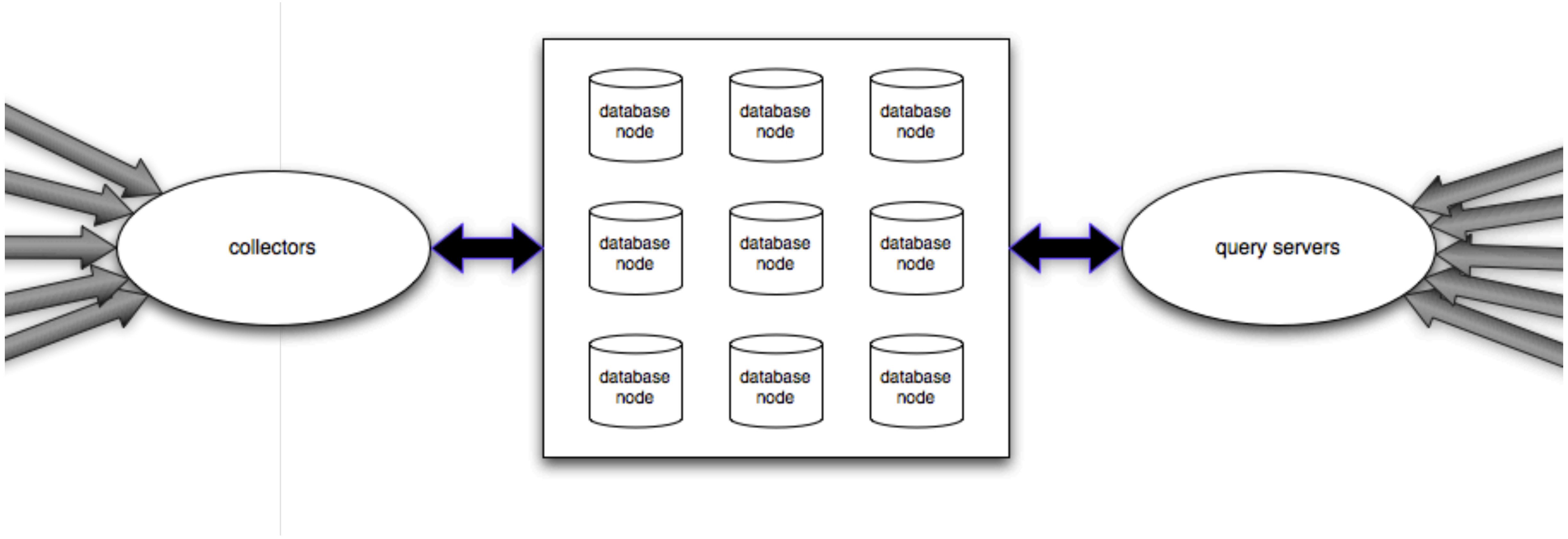
Scala & Erlang

Why Scala and Erlang?

Different strengths
Different applications

JVM
data crunching

Erlang networking/ communication


collectors
database node
database node
database node
database node
database node
database node
database node
database node
database node
query servers

Erlang distribution protocol is great!

Many distributed systems problems are handled.

But not all.

Let’s talk about two:

Let’s talk about two: Polyglot systems

Let’s talk about two:
Polyglot systems
Load balancing

Load balancing

gen_lb

- Cluster discovery
- Process shaped endpoints
- Supports call and cast

Using It

```
gen_lb:start_link(Seed, Service, Strategy).

gen_lb:call(LB, {request, Data}, Timeout).
```

LB Strategies

```
strategy(Nodes, Request, Context) ->
  ...
  {Node, Context2}.
```

Cluster Discovery

```
{admin,seed@local} ! {cluster, self(), Ref},
receive
        {cluster, Ref, Nodes} -> ...
end.
```

Service Ping

```
{Service, Node} ! {ping, self(), Ref},
receive
  {pong, Ref} -> ok
end.
```

Service Shape

```
receive
  {Pid, Ref, Request} -> ...
end.

Pid ! {self(), Ref, Results}.
```

Open source!

https://github.com/fastip/gen_lb

JVM Nodes

JInterface

- Full implementation of Distributed Erlang.
- Part of the OTP Distribution.
- Custom type mappings.
- Mailbox oriented API.

Usage

```
node = new OtpNode(name, cookie)

mbox = node.createMbox(“service”)

mbox.receive()

mbox.send(pid, msg)
```

Usage (cont)

```
elements = new OtpErlangObject[5]

elements[0] = new OtpErlangLong(10)

...

msg = new OtpErlangTuple(elements)
```

Written in 2000.

JInterface Problems

- Usability of type mappings.
- Performance.
- No actor API.
- No monitors.
- Links are broken.

Link Behavior

- Link breakages only fire explicitly.
- No handling for errors.
- No handling for Mbox GC.

Error Handling

```
try {
  ...
} catch (final Exception e) { }
```

We Can Do Better.

Scalang

Easy Polyglot Distributed Systems

Scalang

- Netty networking stack.
- Jetlang actors.
- Native Scala type mappings.
- Process monitors.
- Real supervision.

Usage

```
val node = ErlangNode(name, cookie)

val pid = node.spawnProcess(new MyProc)
```

Usage (cont)

```
class MyProc extends Process {
  def delivery(msg : Any) = msg match {
    case (pid, ref, req) =>
      pid ! (self, ref, “response”)
  }
}
```

Trap Exits

```
class MyProc extends Process {
  def handleExit(from : Pid, reason :Any) {
    println(“oh noes “ + reason)
  }
}
```

Case Classes

```
case class MyStruct(name : String, num : Int)
```

Serialize Into Records

MyStruct(“watdo”, 5)

{my_struct, “watdo”, 5}

gen_lb Integration

- Optional admin service.
- Optional service ping.
- Services in gen_lb shape.

Open source coming soon...

Follow @fastip on twitter for announcement.

Build reliable,
distributed systems.
Simply.

Erlang Protocol

Many different platforms.

Questions?

We’re Hiring!

jobs@fastip.com

https://fastip.com/jobs