ELIXIR

DISTRIBUTION PRIMITIVES

OUTLINE FOR TODAY

- Get setup, connect some nodes and implement some stuff
  - Learn a little bit about node failures and monitoring
- Map/Reduce - Message delivery and idempotence
- Link Shortener - Replication
- Link Shortener part 2 the remix edition - Eventual Consistency and CRDTs

CLONE THIS REPO:

https://github.com/keathley/distsys_training

BUT, FIRST,

SOME WORDS OF CAUTION

DECENT CHANCE

YOU DON'T NEED IT

8 FALLACIES OF DISTRIBUTED SYSTEMS

TEXT

1. THE NETWORK IS RELIABLE

2. LATENCY IS ZERO

3. BANDWIDTH IS INFINITE

4. THE NETWORK IS SECURE

TEXT

5. TOPOLOGY DOESN'T CHANGE

TEXT

6. THERE IS ONE ADMINISTRATOR

7. TRANSPORT COST IS ZERO

8. THE NETWORK IS HOMOGENOUS

NODES

A DISTRIBUTED ERLANG SYSTEM CONSISTS OF A NUMBER OF ERLANG RUNTIME SYSTEMS COMMUNICATING WITH EACH OTHER. EACH SUCH RUNTIME SYSTEM IS CALLED A NODE

http://erlang.org/doc/reference_manual/distributed.html

FIRST STEPS

λ iex --sname chris

Erlang/OTP 21 [erts-10.0] [source] [64-bit] [smp:4:4] [ds:4:4:10] [async-threads:1] [hipe]

Interactive Elixir (1.7.4) - press Ctrl+C to exit (type h() ENTER for help)

iex(chris@basashi)1>

λ iex --sname ben

Erlang/OTP 21 [erts-10.0] [source] [64-bit] [smp:4:4] [ds:4:4:10] [async-threads:1] [hipe]

Interactive Elixir (1.7.4) - press Ctrl+C to exit (type h() ENTER for help)

iex(ben@basashi)1>

FIRST STEPS: NODES

iex(chris@basashi)2> Node.list

[]

---

iex(ben@basashi)2> Node.list

[]

FIRST STEPS: PING

iex(chris@basashi)3> Node.ping :ben@basashi

:pong

---

iex(ben@basashi)1> Node.ping :dave@basashi

:pang

FIRST STEPS: CONNECT

iex(ben@basashi)2> Node.connect :chris@basashi

true

---

iex(chris@basashi)4> Node.list

[:ben@basashi]

FIRST STEPS: TRANSITIVE NODES

λ iex --sname dave

Erlang/OTP 21 [erts-10.0] [source] [64-bit] [smp:4:4] [ds:4:4:10] [async-threads:1] [hipe]

Interactive Elixir (1.7.4) - press Ctrl+C to exit (type h() ENTER for help)

iex(dave@basashi)1>

---

iex(chris@basashi)2> Node.connect :chris@basashi

True

---

iex(dave@basashi)2> Node.list

[:chris@basashi, :ben@basashi]

“SECURITY”

WHILE DOCUMENTS LIKE THE OFFICIAL ERLANG DOCUMENTATION PUT COOKIES UNDER THE TOPIC OF SECURITY, THEY'RE REALLY NOT SECURITY AT ALL. IF IT IS, IT HAS TO BE SEEN AS A JOKE, BECAUSE THERE'S NO WAY ANYBODY SERIOUS CONSIDERS THE COOKIE A SAFE THING. WHY? SIMPLY BECAUSE THE COOKIE IS A LITTLE UNIQUE VALUE THAT MUST BE SHARED BETWEEN NODES TO ALLOW THEM TO CONNECT TOGETHER. THEY'RE CLOSER TO THE IDEA OF USER NAMES THAN PASSWORDS

https://learnyousomeerlang.com/distribunomicon

ERLANG SECRET COOKIE

ERLANG SECRET COOKIE

λ iex --sname ben --cookie lonestar

Erlang/OTP 21 [erts-10.0] [source] [64-bit] [smp:4:4] [ds:4:4:10] [async-threads:1] [hipe]

Interactive Elixir (1.7.4) - press Ctrl+C to exit (type h() ENTER for help)

iex(ben@basashi)1>

---

iex(chris@basashi)6> Node.list

[]

iex(chris@basashi)7> Node.connect :ben@basashi

false

ERLANG SECRET COOKIE

λ iex --sname ben --cookie lonestar

Erlang/OTP 21 [erts-10.0] [source] [64-bit] [smp:4:4] [ds:4:4:10] [async-threads:1] [hipe]

Interactive Elixir (1.7.4) - press Ctrl+C to exit (type h() ENTER for help)

iex(ben@basashi)1>

---

iex(chris@basashi)6> Node.list

[]

iex(chris@basashi)7> Node.connect :ben@basashi

false

ERLANG SECRET COOKIE

iex(chris@basashi)9> Node.set_cookie(:lonestar)

true

iex(chris@basashi)10> Node.get_cookie()

:lonestar

iex(chris@basashi)11> Node.connect :ben@basashi

true

---

iex(dave@basashi)2> Node.list

[:chris@basashi, :ben@basashi]

ERLANG SECRET COOKIE

TEXT

```
iex(chris@basashi)9> Node.set_cookie(:lonestar)

true

iex(chris@basashi)10> Node.get_cookie()

:lonestar

iex(chris@basashi)11> Node.connect :ben@basashi

true

iex(chris@basashi)12> Node.list

[:ben@basashi]
```

MESSAGE PASSING

FIRST STEPS: TRANSITIVE NODES

```
iex(chris@basashi)10> spawn (fn ->
...(1)> Process.register(self, :producer)
...(1)>   receive do
...(1)>     :foo -> IO.puts "I got called"
...(1)>   end
...(1)> end)
```

```
iex(ben@basashi)2> send({:producer, :chris@basashi}, :foo)
:foo
```

```
I got called
iex(chris@basashi)2>
```

MONITORING

FIRST STEPS: TRANSITIVE NODES

iex(chris@basashi)10> Process.monitor(pid)

{:DOWN, ref, :process, object, reason}

---

iex(ben@basashi)2> send({:producer, :chris@basashi}, :foo)

:foo

I got called

iex(chris@basashi)2>

MONITORING

```
iex(chris@basashi)10> Process.monitor(pid)

{:DOWN, ref, :process, object, reason}
```

LOCAL CLUSTER

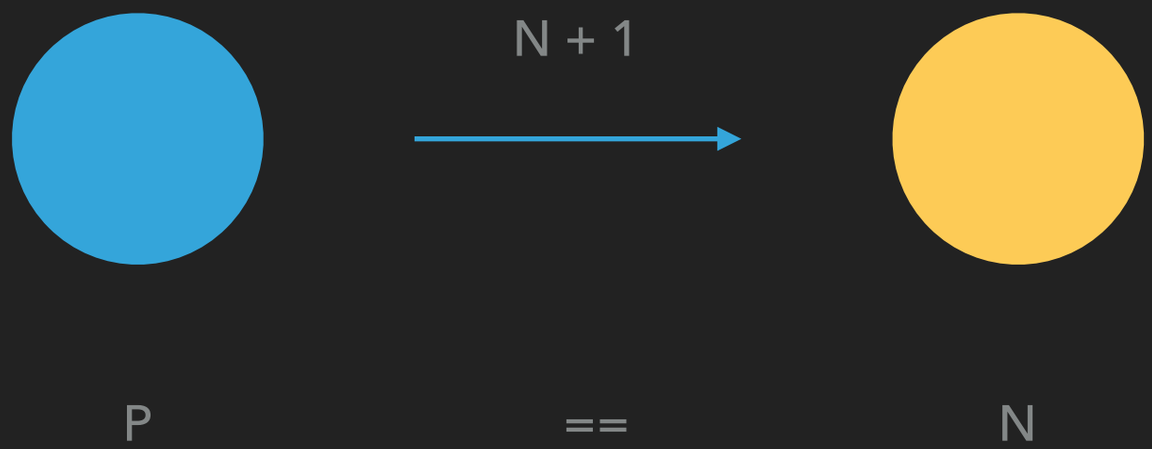
&

SCHISM

LOCAL CLUSTER & SCHISM

```
[n1, n2, n3] = LocalCluster.start_nodes(“cluster”, 3)

Schism.partition([n1, n3])

Schism.heal([n3])

Schism.partition([n1, n2])

Schism.heal([n1, n2, n3])
```

PRODUCER / CONSUMER



P == N

PRODUCER / CONSUMER PARTITIONS