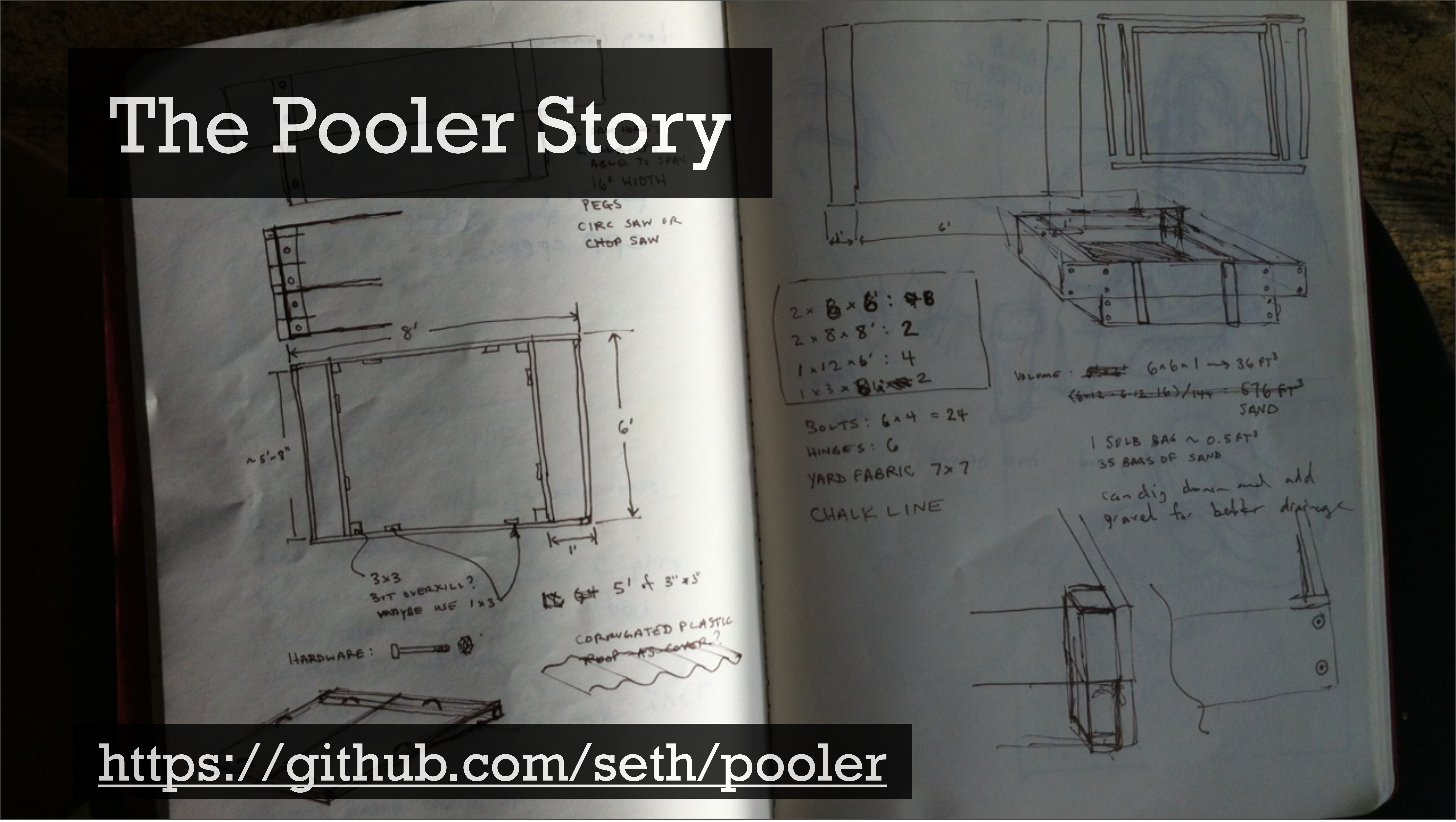
The Pooler Story



https://github.com/seth/pooler

In the summer of 2010.
Coming home from work on the bus with a couple Opscode co-workers, discussing a weekend project I was planning for my backyard.
Passenger wonders what in the world we're working on.

The Pooler Story

https://github.com/seth/pooler

answer: a sandbox.

Simple.

A box. Four sides. no bottom, no top.
Nothing is simple.

When was the last time you used a saw?

Screws or nails? Type of wood? How should the corners go together?
How much sand?

Each simple feature

a
pile of
complexity

http://www.flickr.com/photos/davidstanleytravel/5282834545/

it’s amazing to watch.
There’s a special kind of unbreakable thread that connects a simple feature to a load of complexity.

Each simple feature

1,750 lbs

http://www.flickr.com/photos/davidstanleytravel/5282834545/

and my simple sandbox required almost 2K lbs of sand

The Pooler Story

Seth Falcon
Development Lead
Opscode
@sfalcon

OPSCODE

RULE THE CLOUD

So this is the story of building a SIMPLE connection pool.
and how quickly it become not simple.

A secret uncovered

http://www.flickr.com/photos/digitalrob70/6981414442/

But it’s also the story of uncovering a secret of building robust systems with OTP.

Supervisor Driven Design

Think about the supervision tree as a principal aspect
Understand new projects by visualizing the supervision tree.

You start, if you haven't already, by reading these.
When you are learning, you can't focus on supervisors first.
You need to build an app

Supervisors Supervisors Supervisors

http://www.flickr.com/photos/whatcounts/521758821/sizes/o/in/photostream/

You aren't using enough supervisors
You aren't using them as effectively as you can

I expect to learn something

Going to share some discoveries (not my inventions) of what I think are good practices
Hoping that it isn't: **you can do all of that with gproc and 3 lines of code**

2010

We need an exclusive access connection pool

Once upon a time, it was September 2010. Experimenting with Riak.
Pool Riak pb client connections and act as cheap load balancer

Maintain a pool of members

Track in use vs free members

Simple. Right? And Erlang gives you all the primitives.

Maintain a pool of members

Track in use vs free members

Consumer crashes, recover member

Member crash, replace member

Multiple pools

Load balancing across pools

But there are a few more features we’ll need

Start members asynchronously
and in parallel

Start timeout?

Initial pool size vs max

Cull unused members after timeout

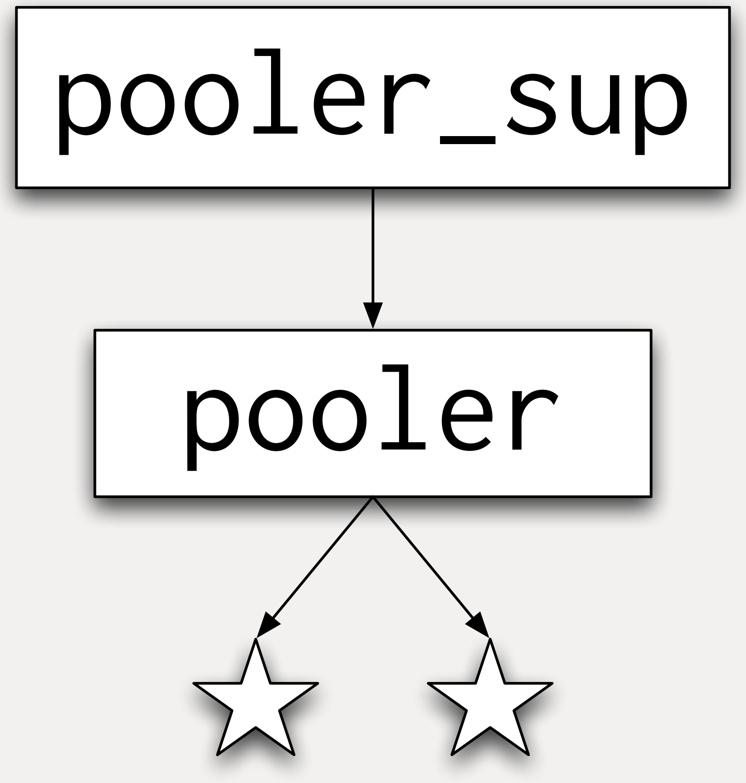
When to add members?

and yet a few more. Not as simple.

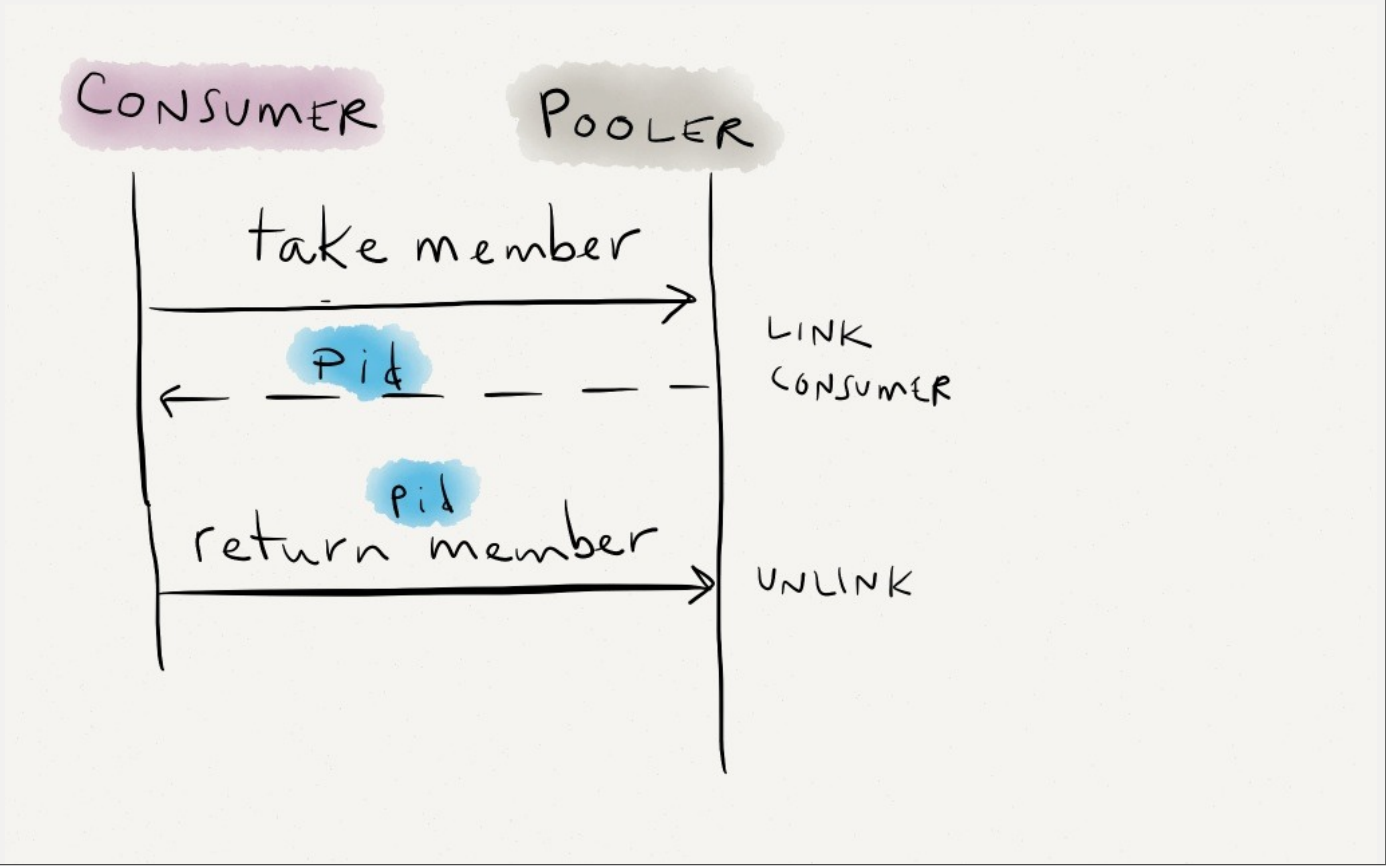
Version 0

pooler is a gen_server;

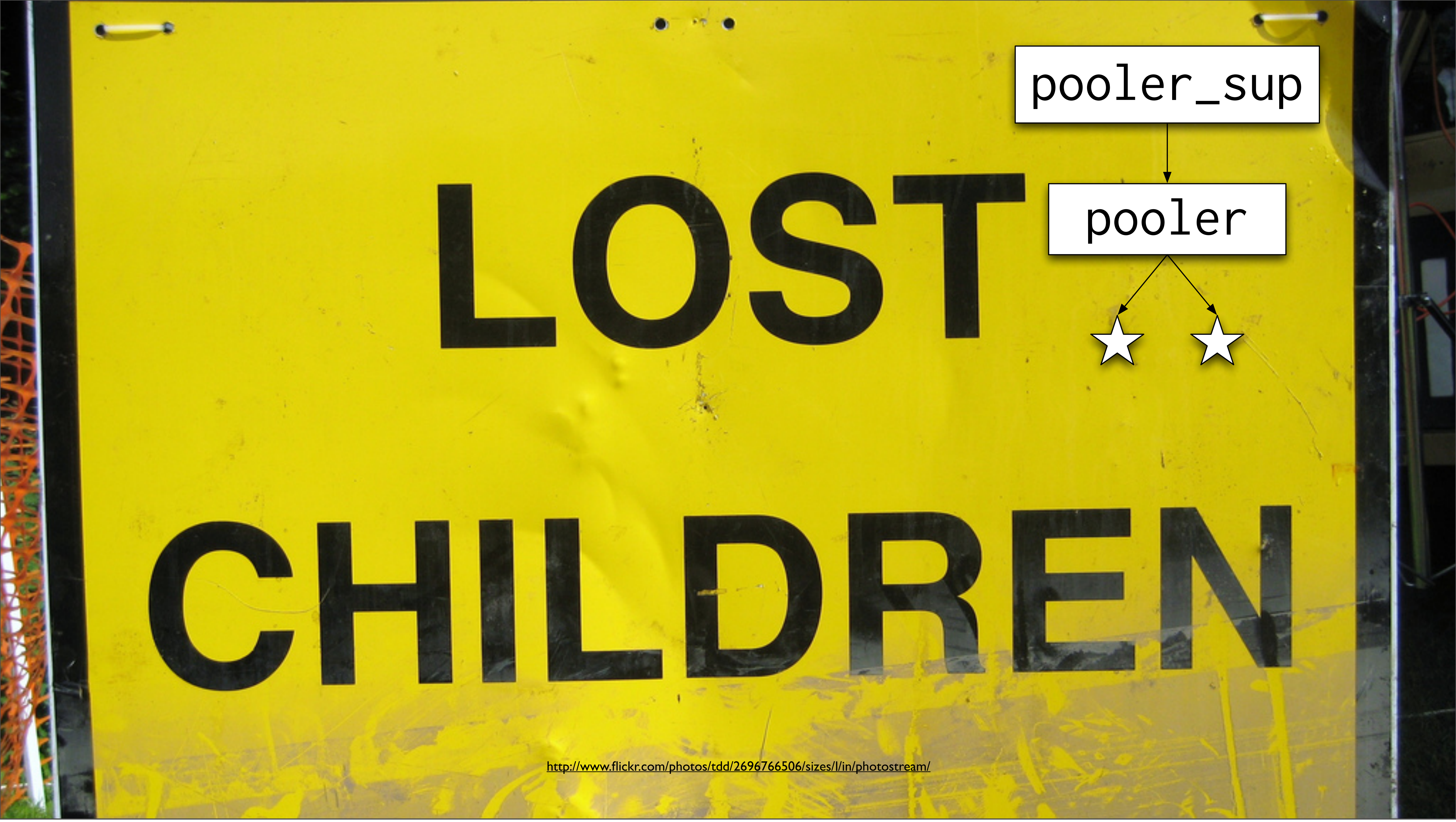
calls PoolMember:start_link



Version 0 is here for illustrating the evolution. Simplest possible thing.
members are unsupervised.



Here’s the basic message flow for using pooler



http://www.flickr.com/photos/tdd/2696766506/sizes/l/in/photostream/

Unsupervised children is sad panda.

No unsupervised processes

(Rule 1)

Know your processes:

 where they are;

 where they're from

Hot code upgrade

Keep process spawning explicit

When everybody is supervised, you can easily find a process and know where it is from.

Know your processes:

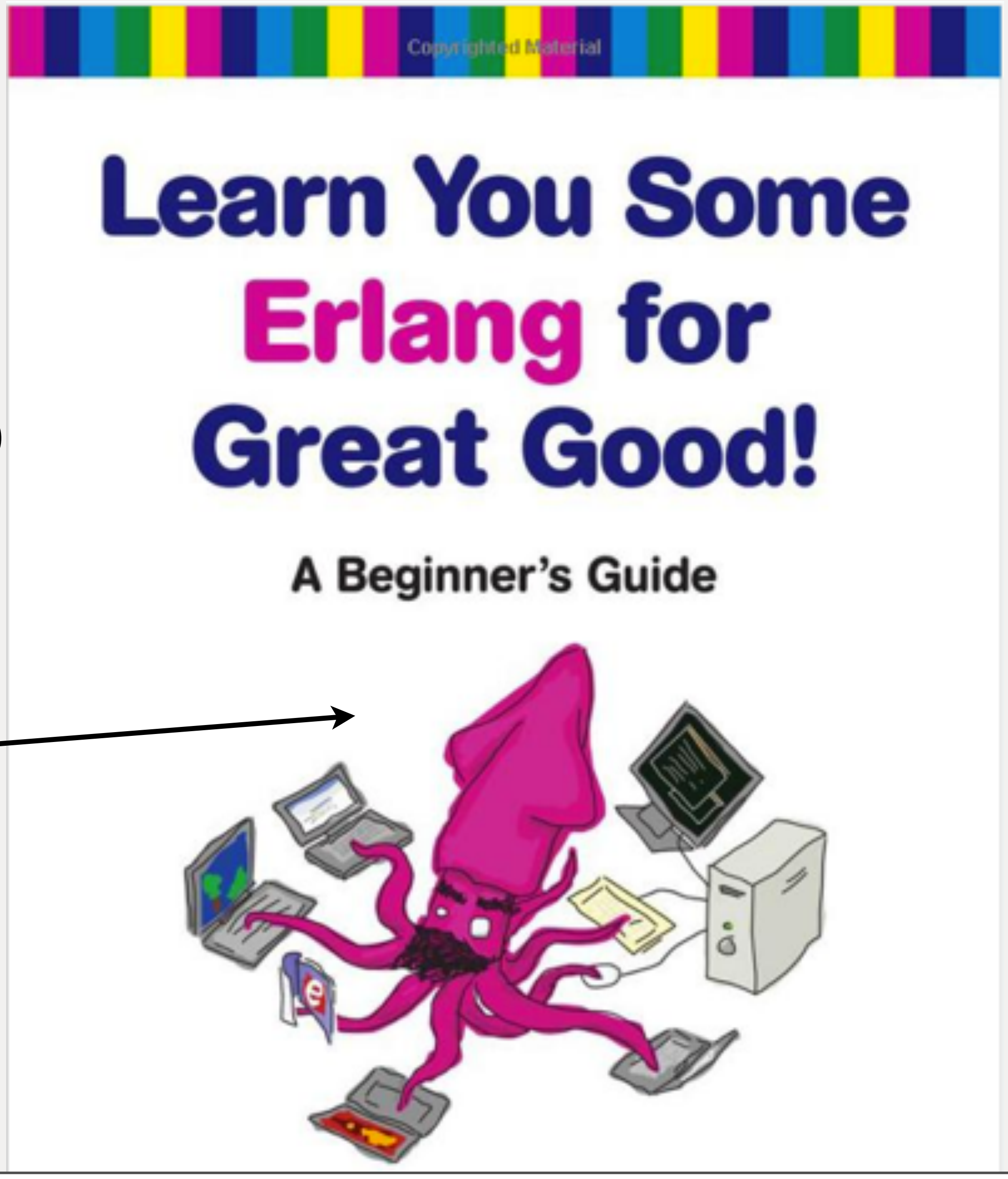
where they are;

where they're from

Hot code upgrade

Keep process spawning exp

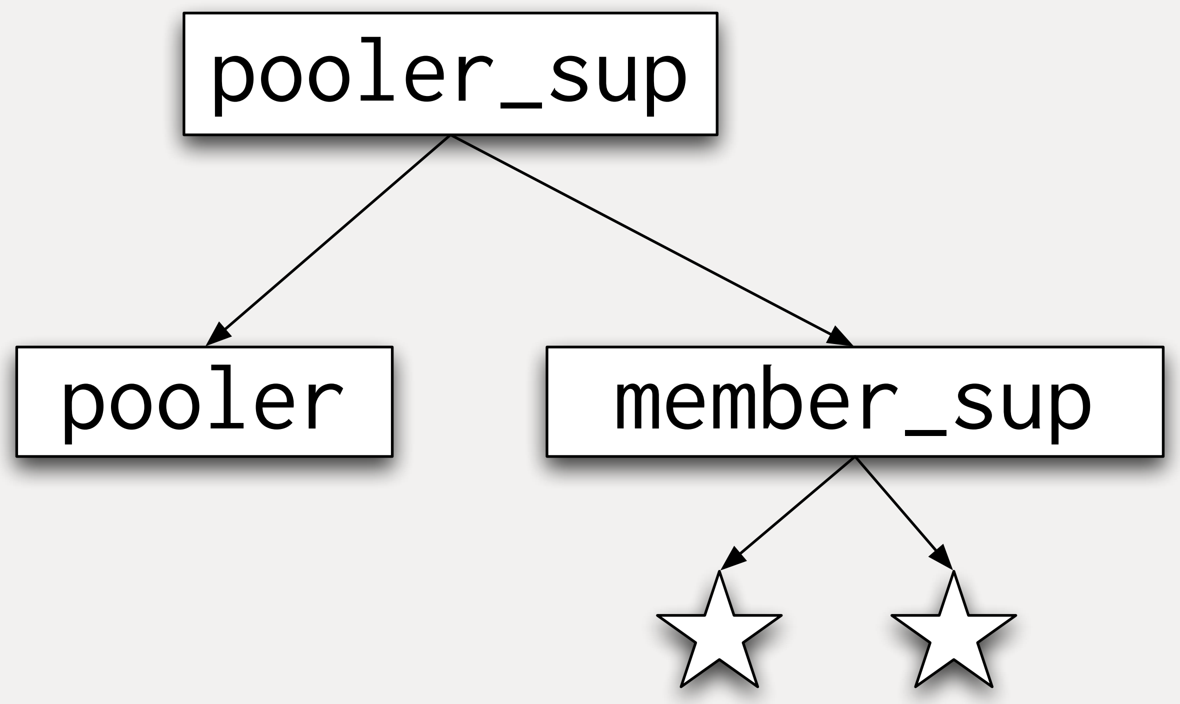
The squid will come after you ⟶



easier to track down process leaks (which could, over time starve vm of ram)

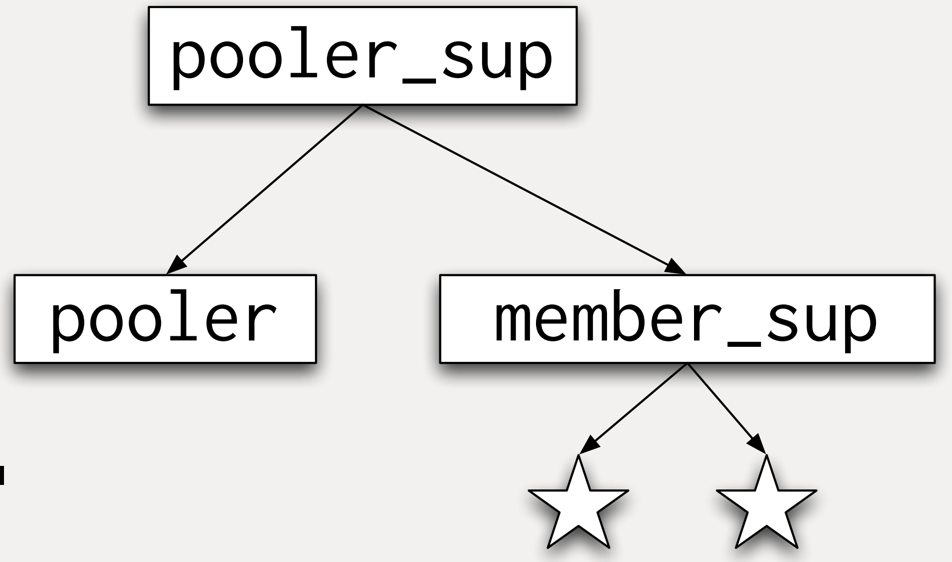
Version 1

Rule 1 satisfied.



member_sup supervises members as simple_one_for_one

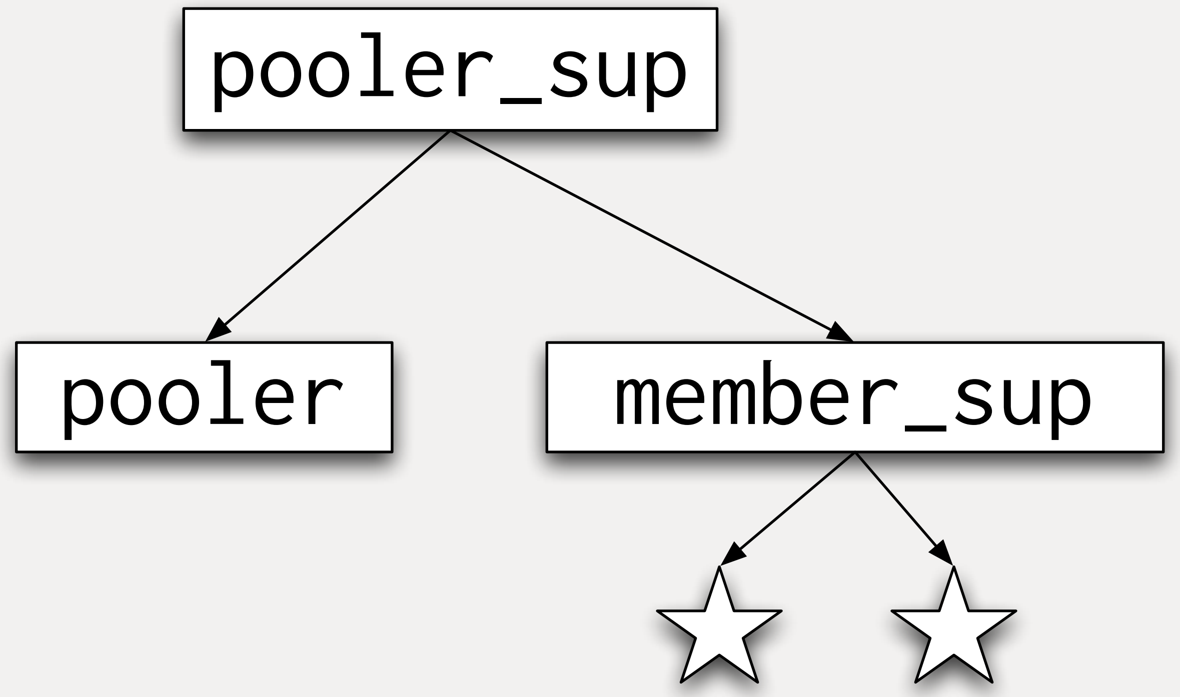
member_sup supervises pool members



```
-module(member_sup).
-behaviour(supervisor).
-export([start_link/1, init/1]).

init({Mod, Fun, Args}) ->
    Worker = {Mod, {Mod, Fun, Args},
              temporary, brutal_kill,
              worker, [Mod]},
    Specs = [Worker],
    Restart = {simple_one_for_one, 1, 1},
    {ok, {Restart, Specs}}.
```

member_sup embeds MFA for member at init, so for pooling different types of members, need another member_sup

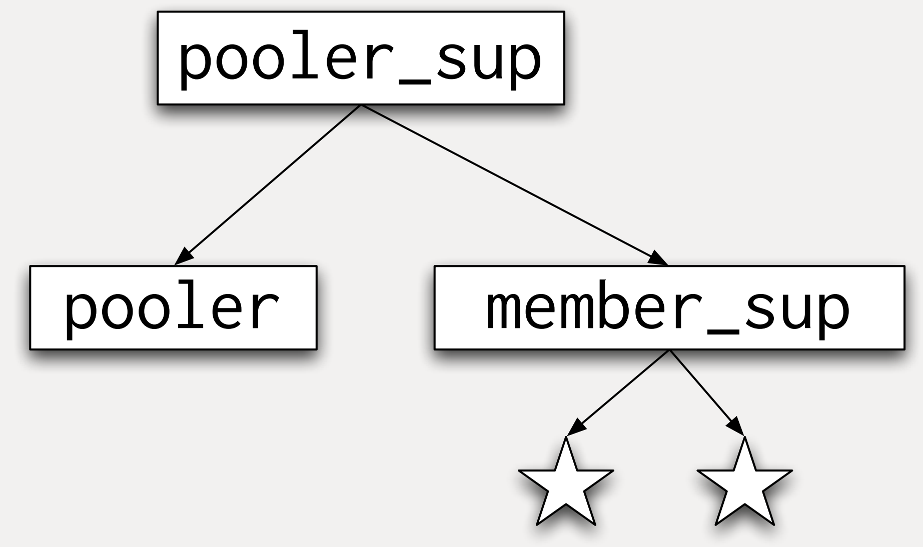
pooler starts members with start_child



```
supervisor:start_child(member_sup, [])
```

Here’s how the pooler gen_server starts pool members.

static child spec starts worker_sup



```
-module(pooler_sup).
-behaviour(supervisor).

init([]) ->
    Config = application:get_all_env(pooler),
    Pooler = {pooler, ...},
    MemberSup = {member_sup,
                 {member_sup, start_link, [Config]},
                 permanent, 5000, supervisor,
                 [member_sup]},
    Specs = [Pooler, MemberSup]
    {ok, {{one_for_one, 5, 10}, Specs}}.
```

And finally, how the member_sup is wired into the top-level supervisor in pooler

No unsupervised processes

supervisor:start_child

spawn
start_link

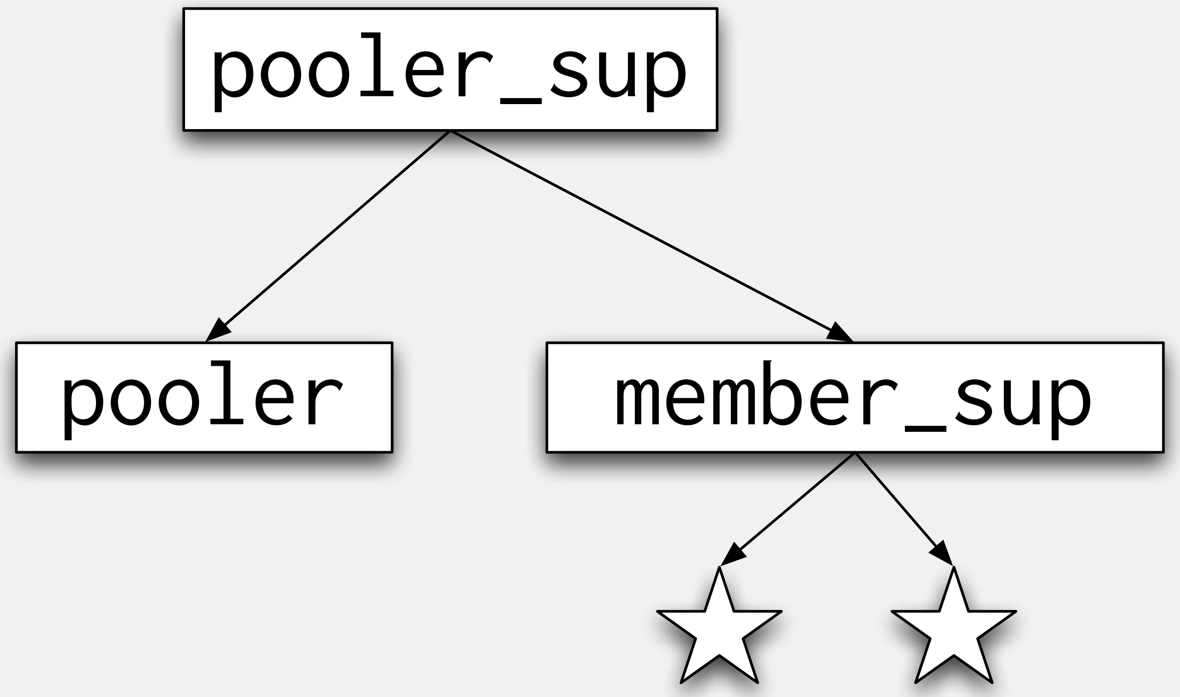
→

supervisor
+
simple_one_for_one
worker

Look for instances of spawn and start_link. Add aa simple_one_for_one supervisor and replace the spawn/start_link calls with supervisor:start_child calls.

Version 1

Rule 1 satisfied.

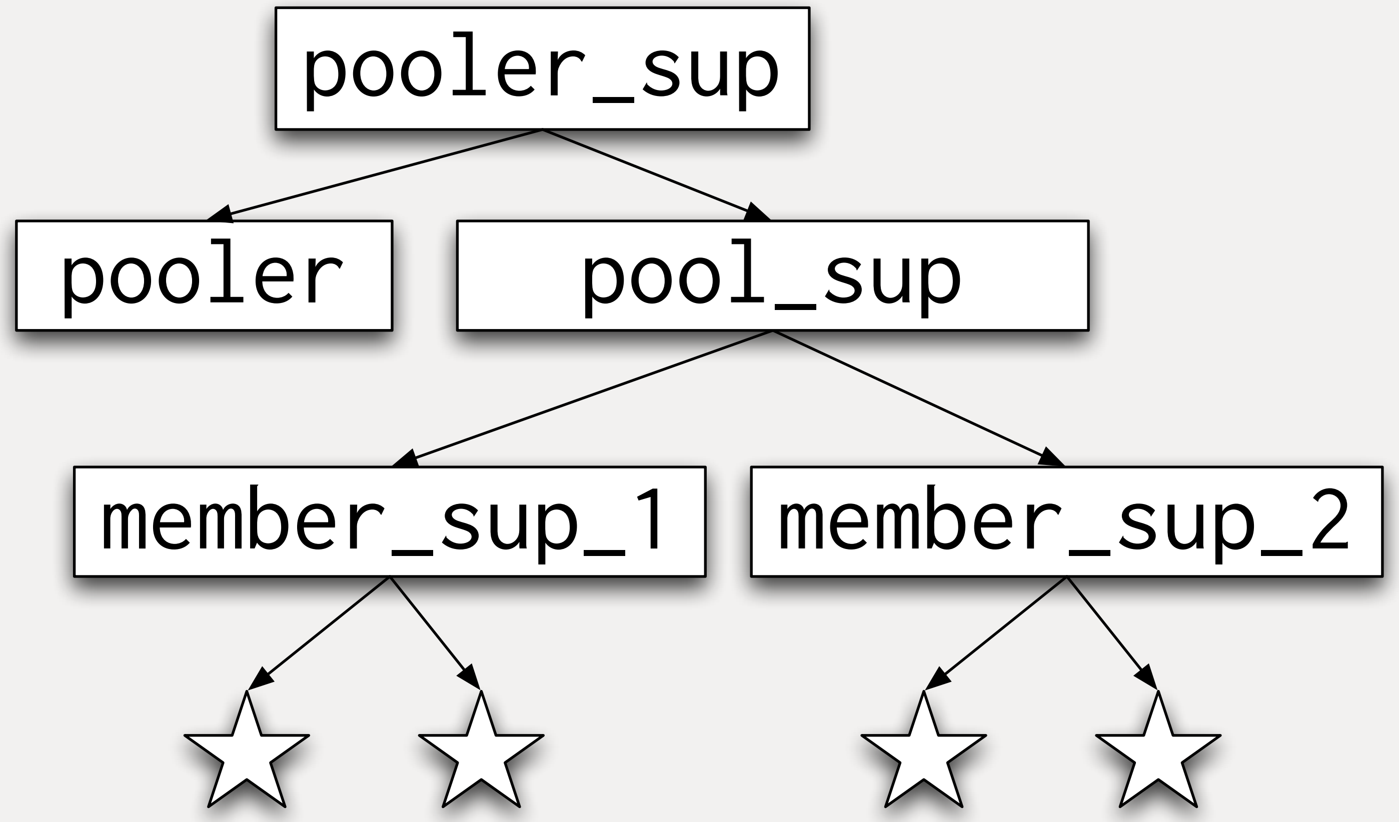


But no multiple pools.

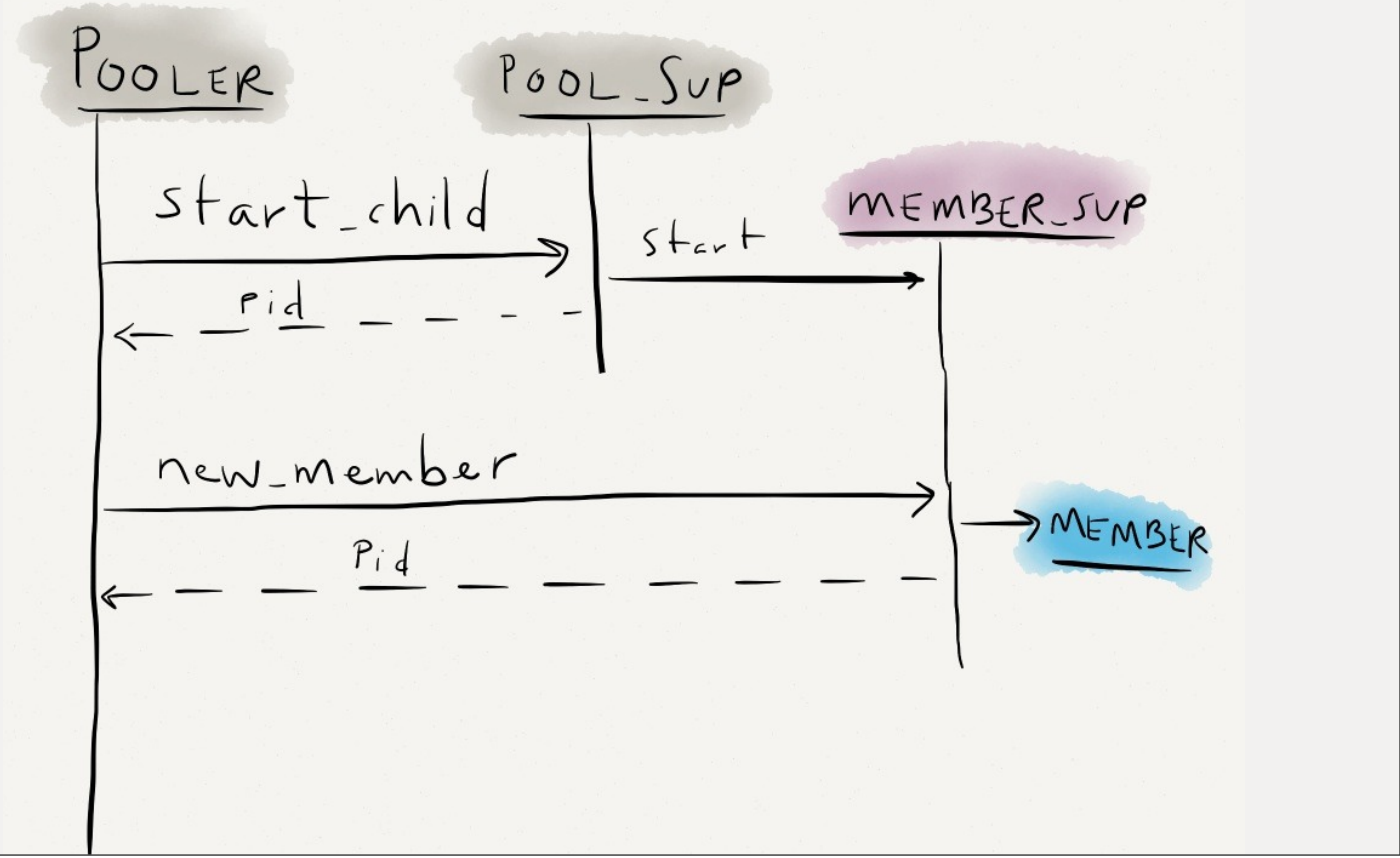
The member_sup carries the MFA to start a member of a given type
Want each pool to have a member_sup.

Create supervisors dynamically

simple_one_for_one and supervisor:start_link can be used for supervisors too.



Probably not news to you. But very useful.

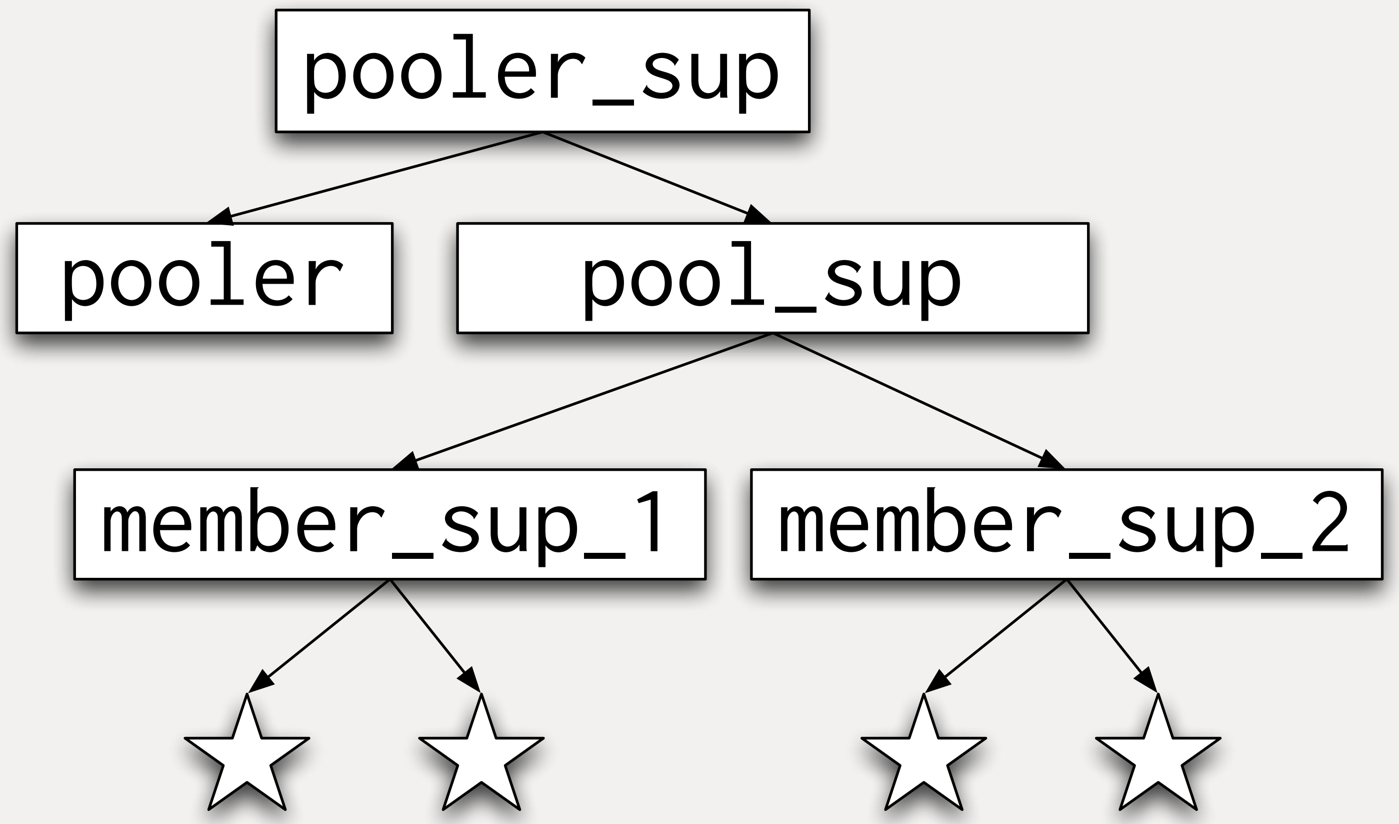


Here's the message flow for pooler adding a new pool and then adding a new member to the new pool.

Version 2

Rule 1 satisfied.

Multiple pools!

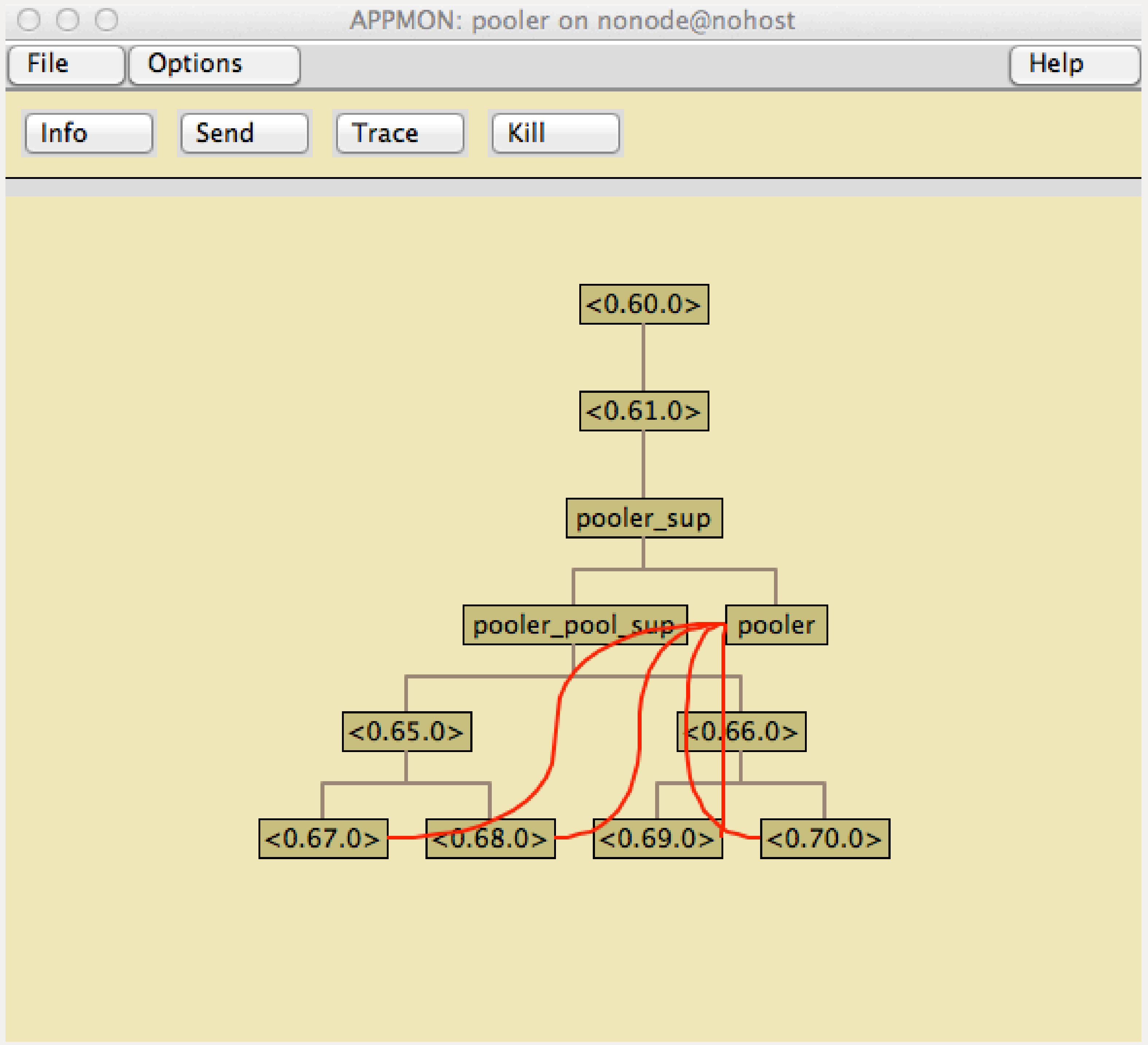


multiple pools

all supervised

init_count, max_count

cull_interval, max_age



This is the state of pooler 0.0.2.

http://www.flickr.com/photos/8927927@N02/6837374725/

time passes... dream sequence

OPSCODE

RULE THE CLOUD

2012

Good News!

OPSCODE

RULE THE CLOUD

2012

Good News!

Facebook is a customer

OPSCODE

RULE THE CLOUD

2012

Bad News...

They need the new stuff next week

We were using poolboy, but saw lockup of pool under load. This was also found at basho and then fixed via QuickCheck. Bug related to queueing when full, different feature/complexity trade-off. pooler just returns an error when full. No queue. With pooler, no hang under load. But..

Start Up Problems

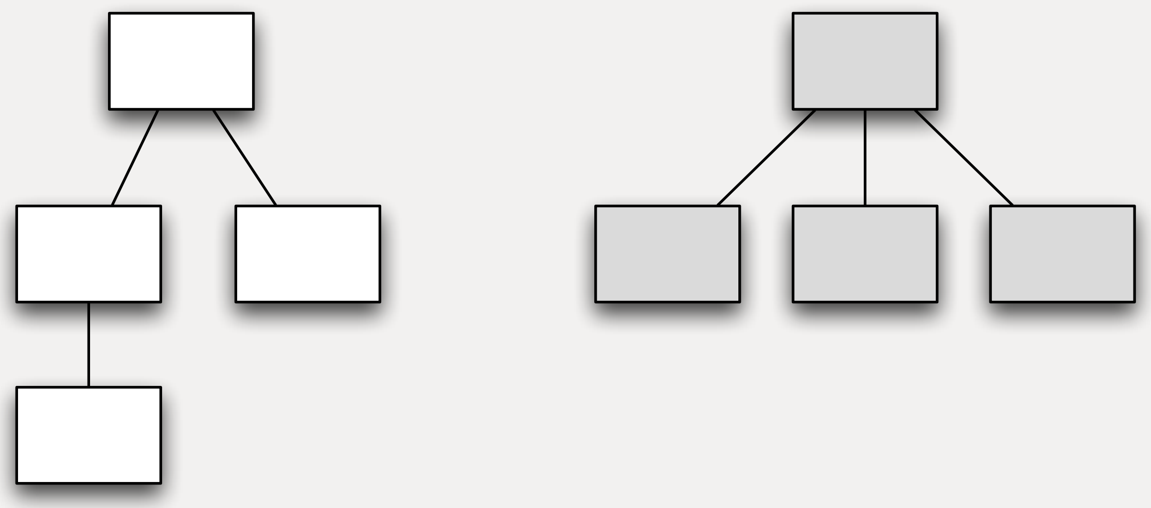
pooler doesn’t know about it’s members. But needs member’s apps to start before it.
And wanted to keep pool config as static.

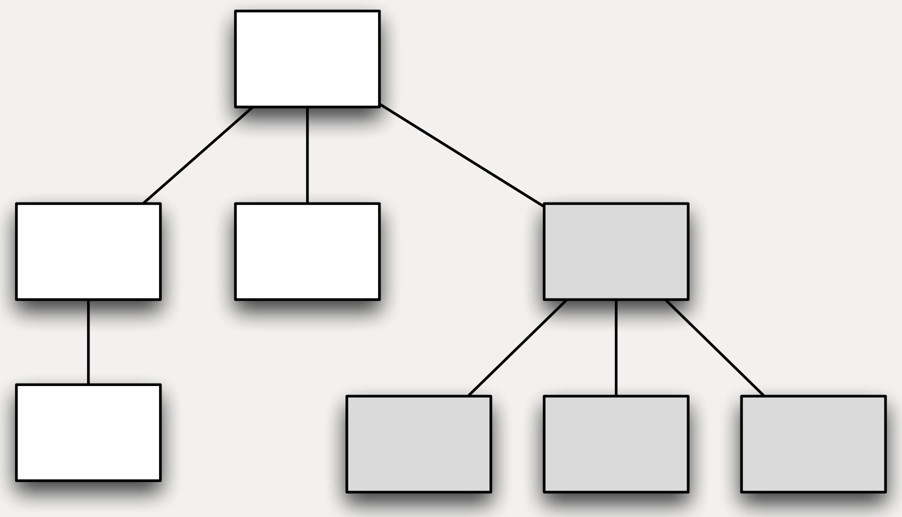
pooler has no deps.
pooler calls emysql:start_link.

Who calls
application:start(emysql)?

a problem caused by trying to keep things simple and only use static pool config

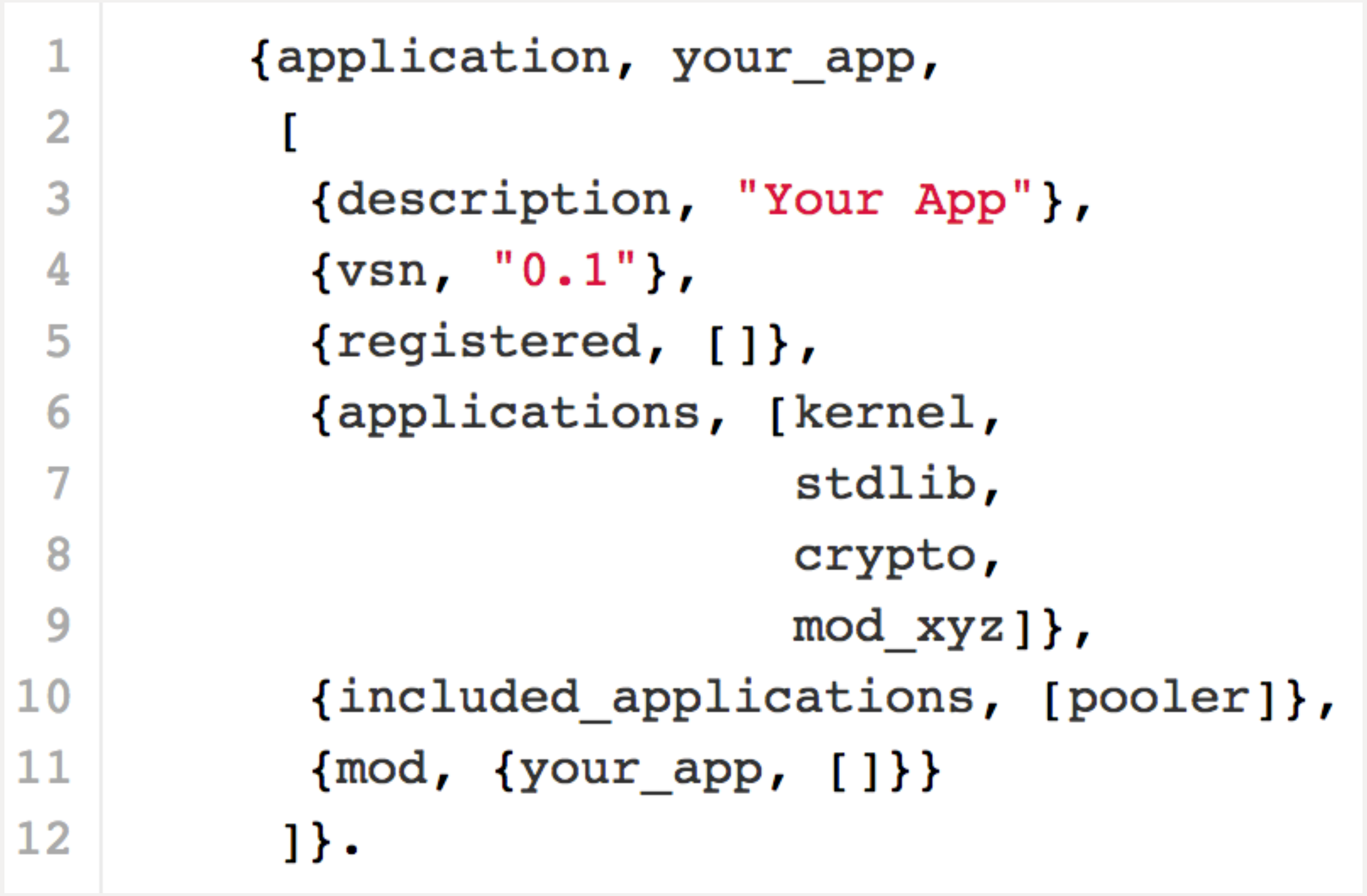
included_applications





L: two separate apps
R: one app includes another

in your app:



```erlang
{application, your_app,
 [
  {description, "Your App"},
  {vsn, "0.1"},
  {registered, []},
  {applications, [kernel,
                  stdlib,
                  crypto,
                  mod_xyz]},
  {included_applications, [pooler]},
  {mod, {your_app, []}}
 ]}.
```

To use pooler as an included app, do this

in your app:

```
-module(your_app_sup).
-behaviour(supervisor).

init([]) ->
    Pooler = {pooler_sup,...},
    Worker = {your_worker,...},
    Restart = {one_for_one, 1, 1},

    {ok, {Restart, [Pooler, Worker]}}.
```

and then start pooler’s top-level supervisor somewhere in your supervision tree.

in pooler:

take care with application:get_env

application_get_env/1 infers the application which will change if used in included_application context.
application_get_env/2 is unambiguous so you know where code will look for config.
config should be name spaced so /2 is better all around.
(20 min mark)

Under Load

http://www.flickr.com/photos/3059349393/3709115244/sizes/l/in/photostream/

Two things

http://www.flickr.com/photos/3059349393/3709115244/sizes/l/in/photostream/

Two small lessons learned when testing pooler embedded in a system put under load

Cast is crazy, so call me (maybe)

return_member was a cast. Chosen as an optimization. Can end up overwhelming pooler’s mailbox.

When in doubt, call

Back pressure avoids
overwhelming mailbox

Don’t optimize with cast without measuring.
If you know deadlock isn’t a concern, try call first
If call isn’t fast enough, consider redesign, not cast

Mind your timeouts

Don’t fear ∞

```
gen_server:call(?SERVER, take_member, infinity)
```

Members started in-line with pooler server loop

Slow member start triggers timeout

Under extreme load and certain error conditions within the system (not pooler in isolation) default timeouts for gen_server:call result in falling off a cliff of failure.

call +∞

Run slower

Degrade with load

But still run

Changing to call with infinity gives (somewhat) more graceful degradation under failure and avoids some death spiral scenarios.

http://www.flickr.com/photos/yourbartender/5379244544/sizes/l/in/photostream/

Time to ride off into the sunset?

2013

In production at Opscode

http://www.flickr.com/photos/yourbartender/5379244544/sizes/l/in/photostream/

pooler used in production to pool postgres db connections in Opscode Private, Hosted, and Open Source Chef Servers.

2013

In production at Opscode

Load tested at Facebook

http://www.flickr.com/photos/yourbartender/5379244544/sizes/l/in/photostream/

We’re not done

http://www.flickr.com/photos/yourbartender/5379244544/sizes/l/in/photostream/

Single gen_server serving all pools

Doesn’t fit our our evolved use cases. Want to pool different things pg and redis. Want isolation.

Can’t dynamically add pools

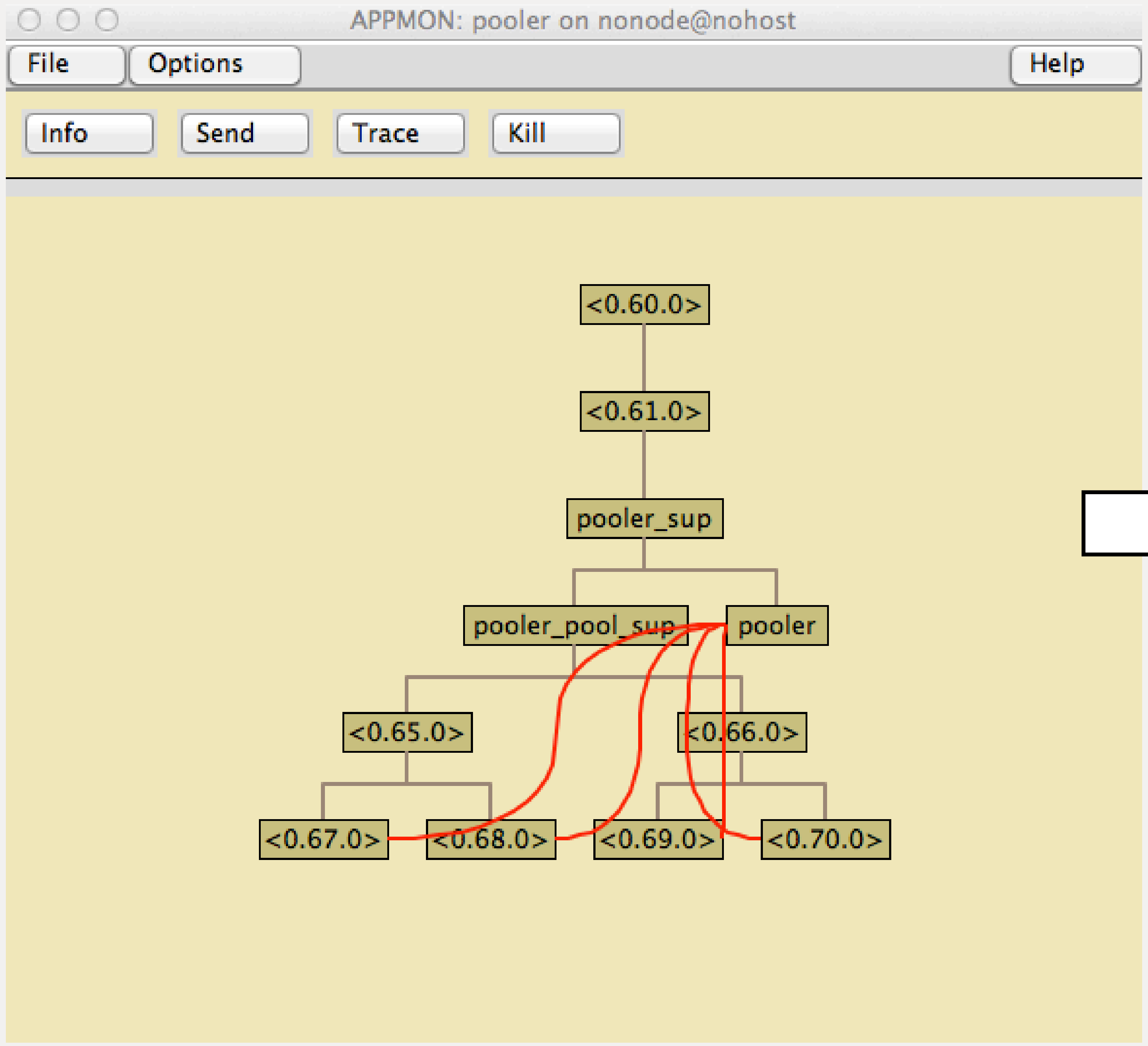
When pooling different things, adding pools at run time makes sense. Also solves the startup ordering problem.
pooler should be more of a generic service. runs in the background.

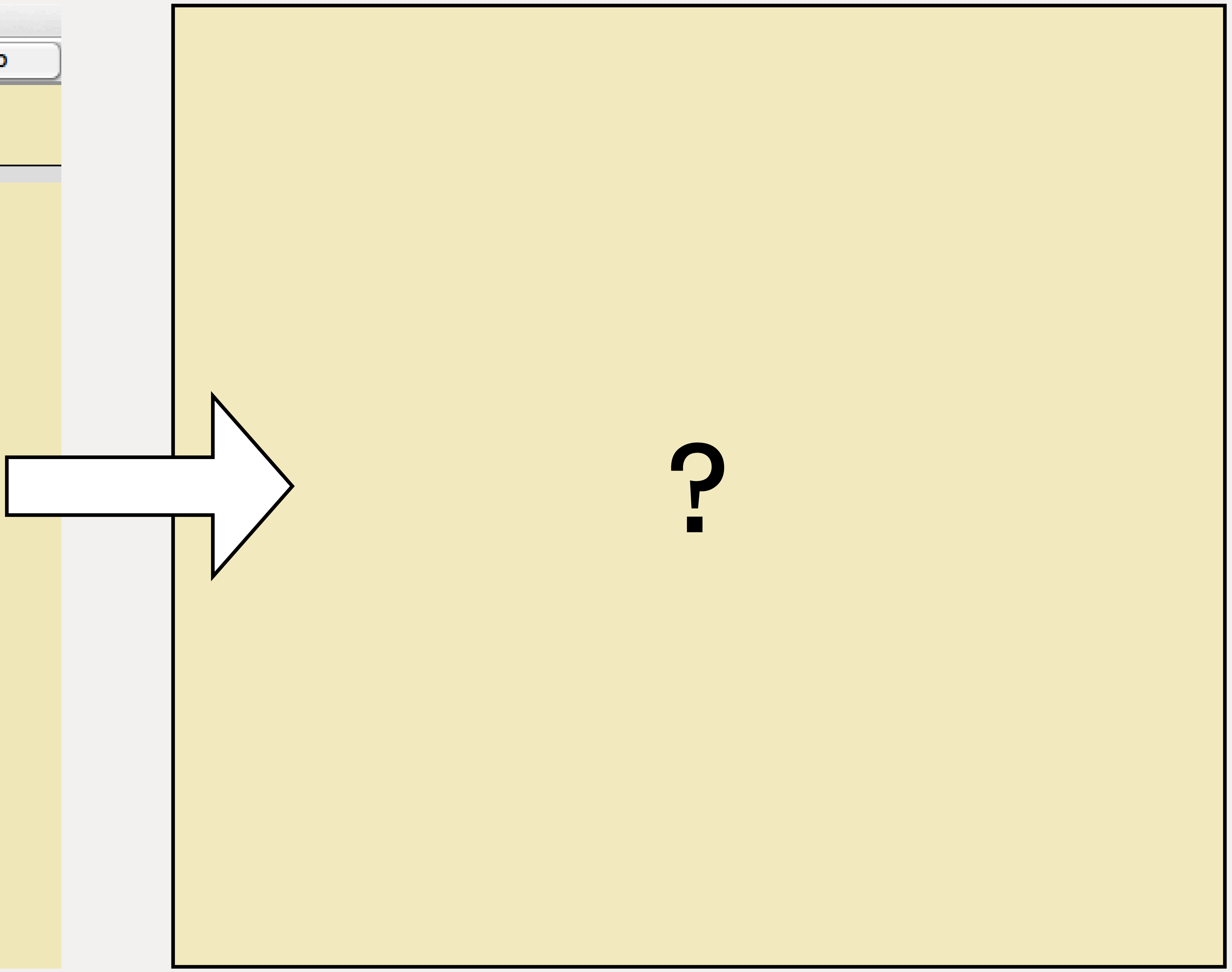
In-line synchronous member start

want improved dynamic pool growth -- add a batch, not just one
minimize impact on perf for slow starting members and member crashes

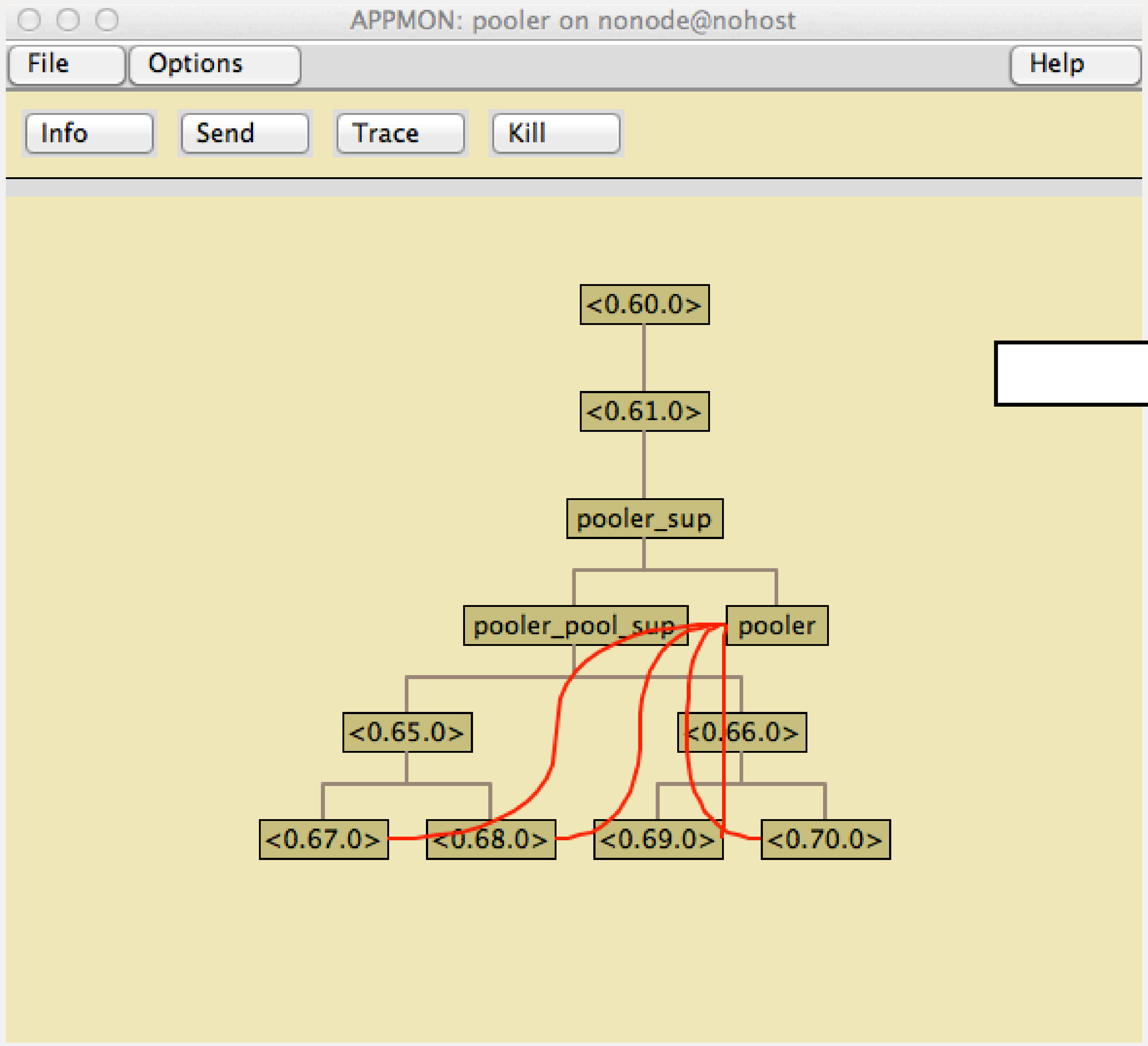
TODO

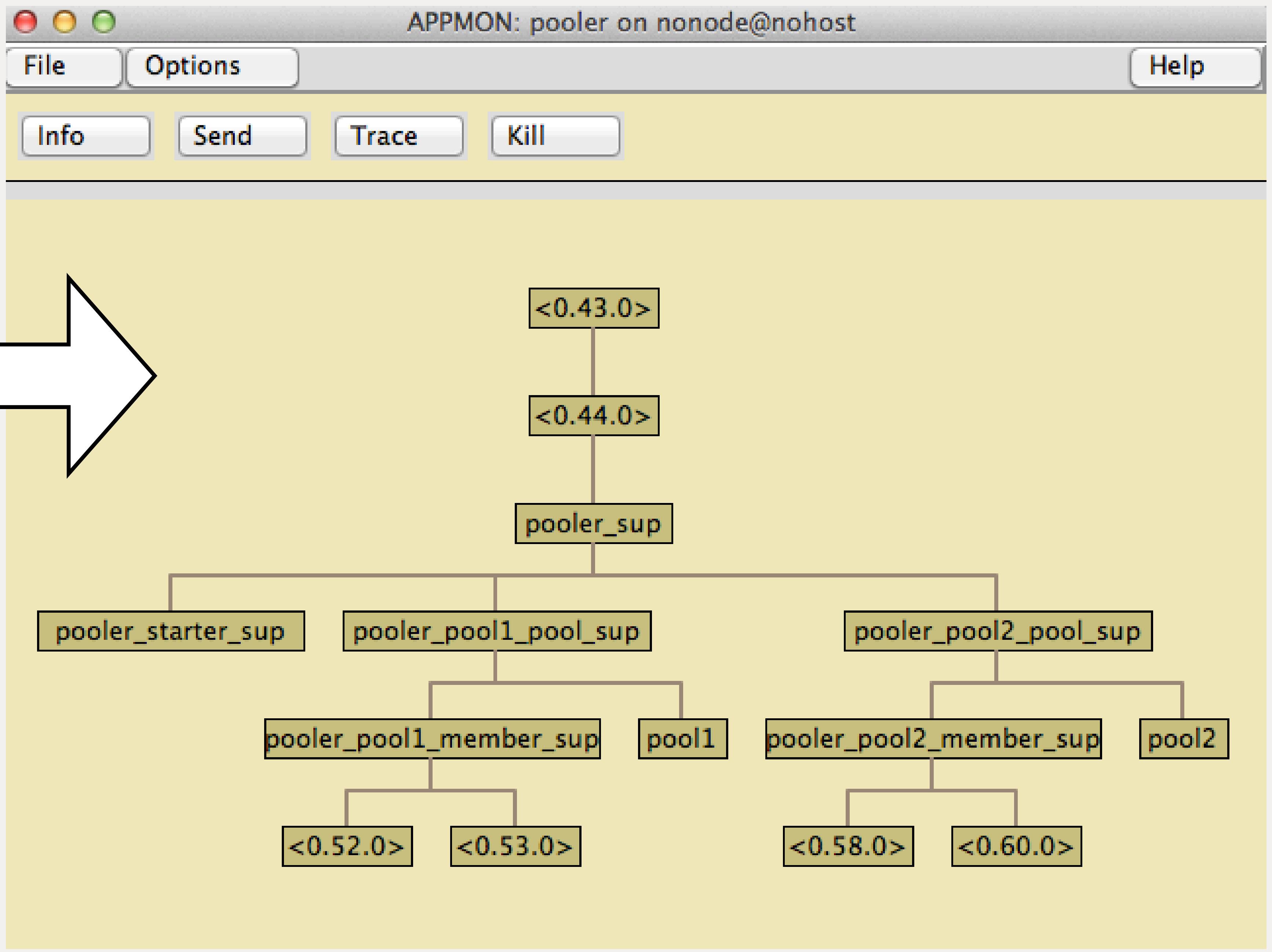
1. True multi pool
2. Async + parallel member start





What should the supervision tree look like?


APPMON: pooler on nonode@nohost
File
Options
Help
Info
Send
Trace
Kill
<0.60.0>
<0.61.0>
pooler_sup
pooler_pool_sup
pooler
<0.65.0>
<0.66.0>
<0.67.0>
<0.68.0>
<0.69.0>
<0.70.0>


APPMON: pooler on nonode@nohost
File
Options
Help
Info
Send
Trace
Kill
<0.43.0>
<0.44.0>
pooler_sup
pooler_starter_sup
pooler_pool1_pool_sup
pooler_pool2_pool_sup
pooler_pool1_member_sup
pool1
pooler_pool2_member_sup
pool2
<0.52.0>
<0.53.0>
<0.58.0>
<0.60.0>

Create supervisors dynamically

(take 2)

We did this already where we used a simple_one_for_one pattern to start new supervisors.

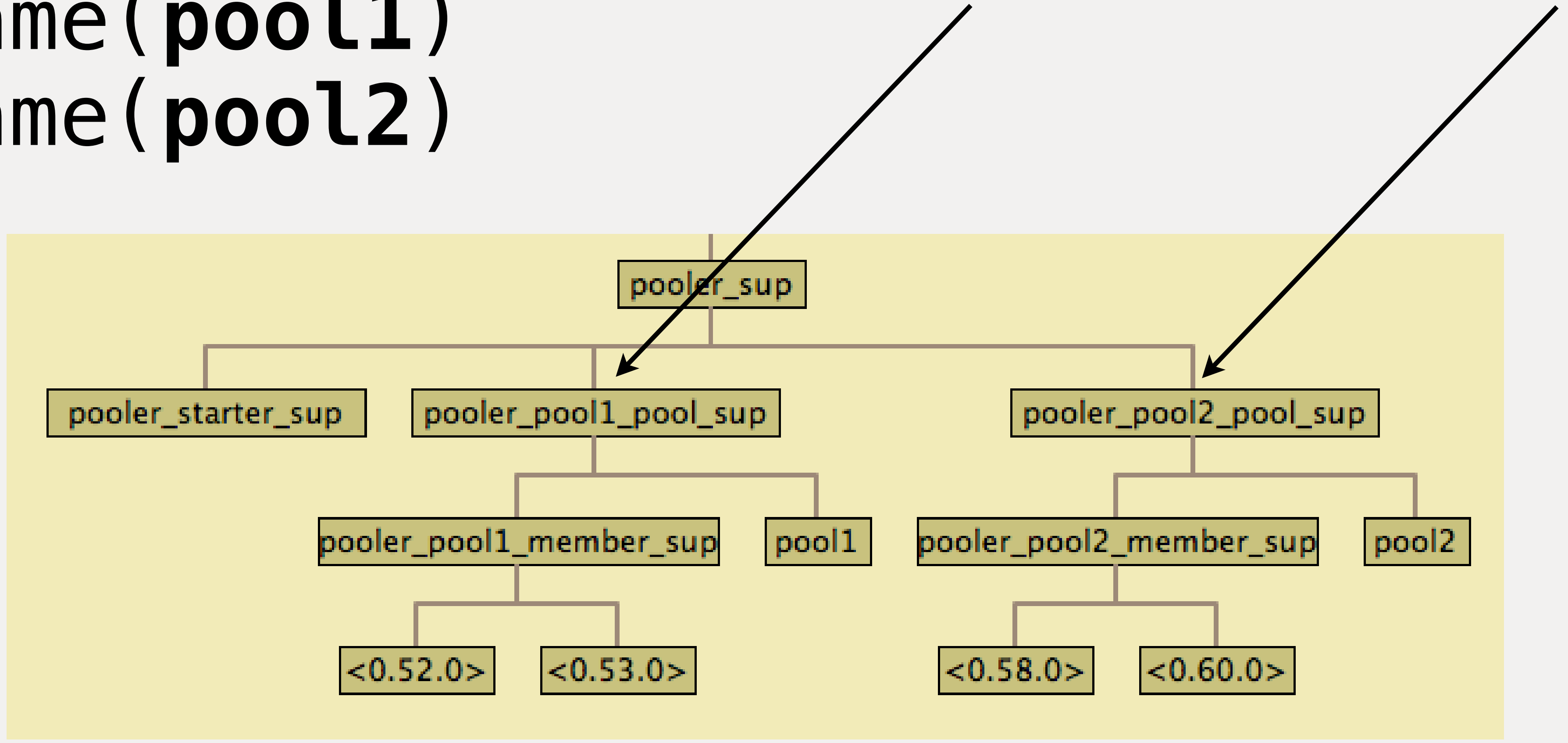
Create child spec dynamically

Call supervisor:start_link

(not simple_one_for_one)

```
pool_sup_name(Name) ->
    list_to_atom("pooler_" ++
                 atom_to_list(Name) ++
                 "_pool_sup").
```

pool_sup_name(**pool1**)
pool_sup_name(**pool2**)



```
new_pool(Config) ->
    NewPool = pooler_config:list_to_pool(Config),
    Spec = pool_sup_spec(NewPool),
    supervisor:start_child(?MODULE, Spec).

pool_sup_spec(#pool{name = Name} = Pool) ->
    SupName = pool_sup_name(Name),
    {SupName, MFA, ...}.
```

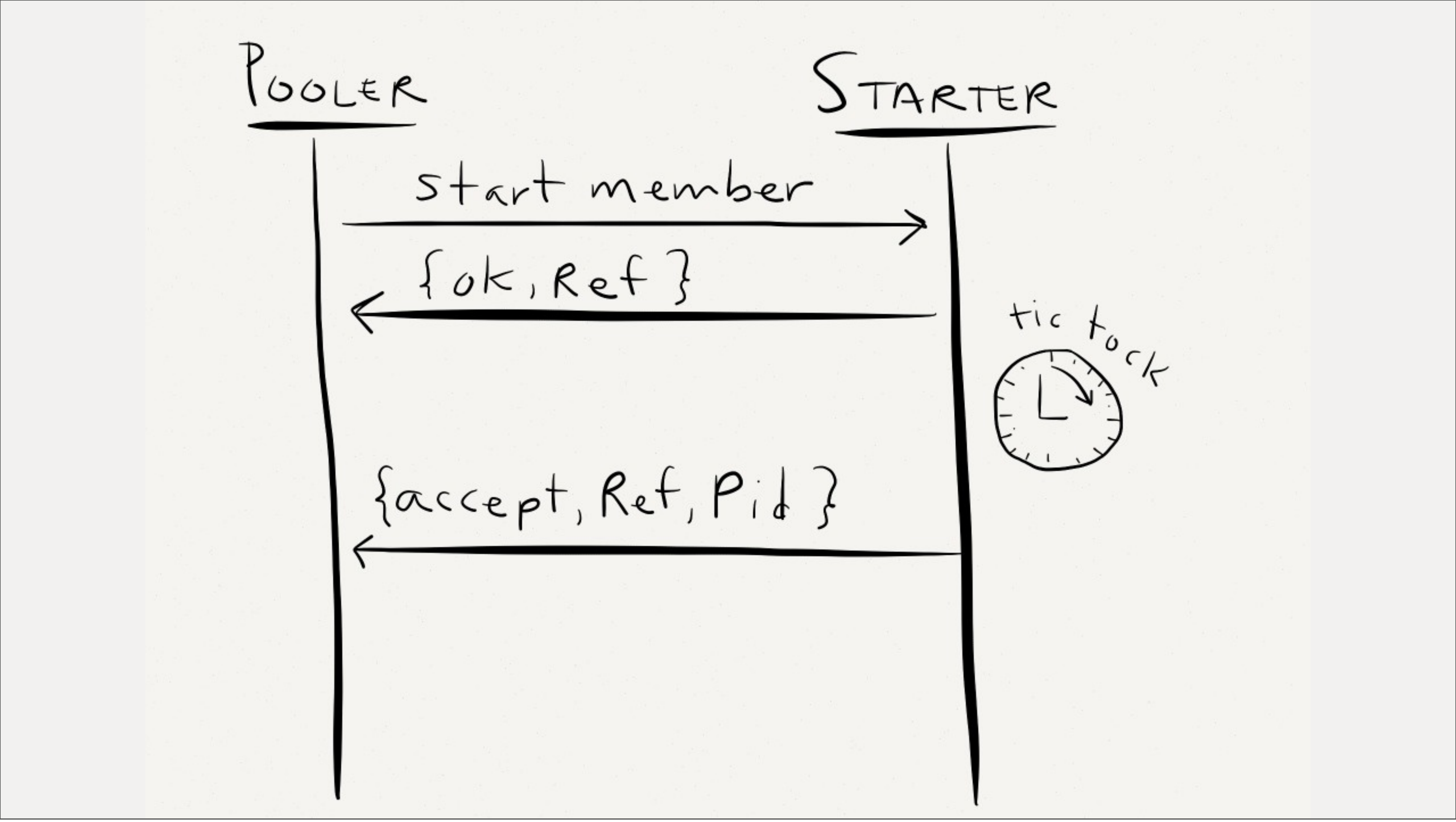
TODO

✓ 1. True multi pool
2. Async + parallel member start

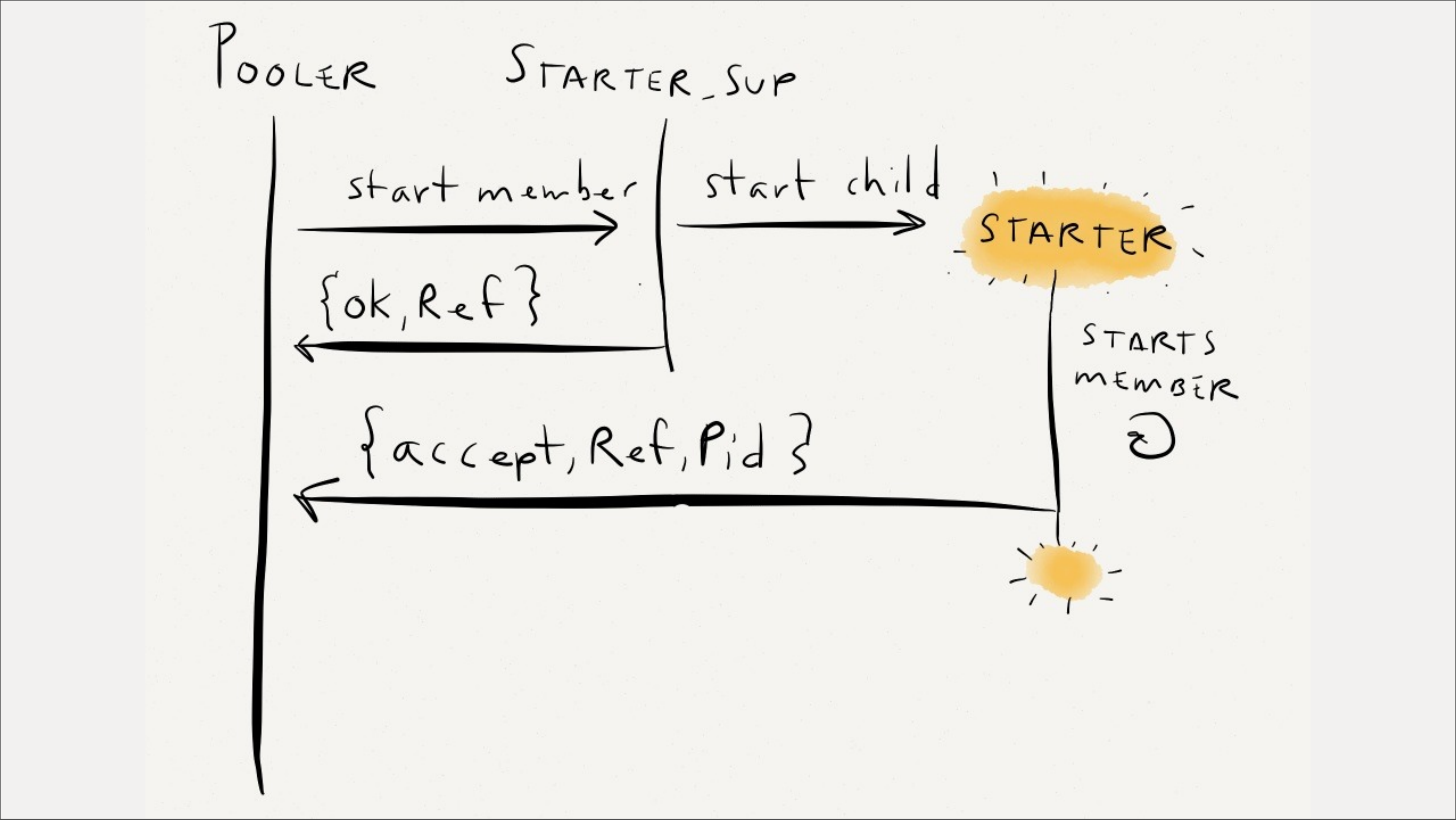
async start

supervisor:start_child(PoolSup, [])
(blocks until child ready)

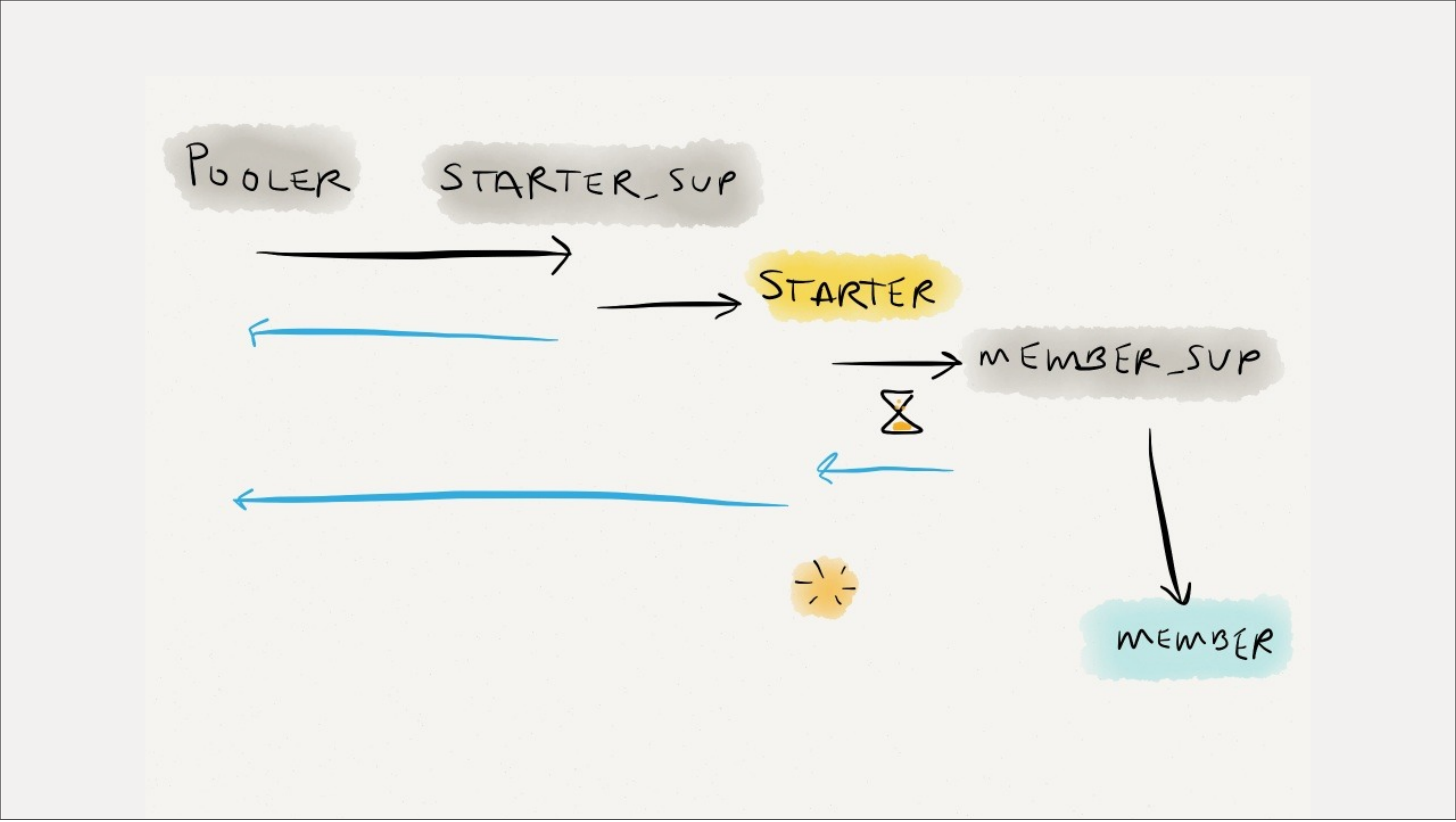
Need Another Process
(it better be supervised)



Basic flow for async member start using a starter gen_server



Actual async member start uses starter_sup and a single use starter gen_server which triggers member start by setting timeout value to 0 in return from init/1. After creating member and sending msg to appropriate pool, starter exits normally.



Another view of the async member start flow

async + parallel start

(once running)

but at init time, we want N

good for adding capacity dynamically.
does not help at pool initialization time

```
do_start_members_sync(Pool, Count) ->
    Parent = self(),
    Pids = [ launch_starter(Parent, Pool)
             || _I <- lists:seq(1, Count) ],
    gather_pids(StarterPids, []).


launch_starter(Parent, Pool) ->
    Fun = ...,
    proc_lib:spawn_link(Fun).
```

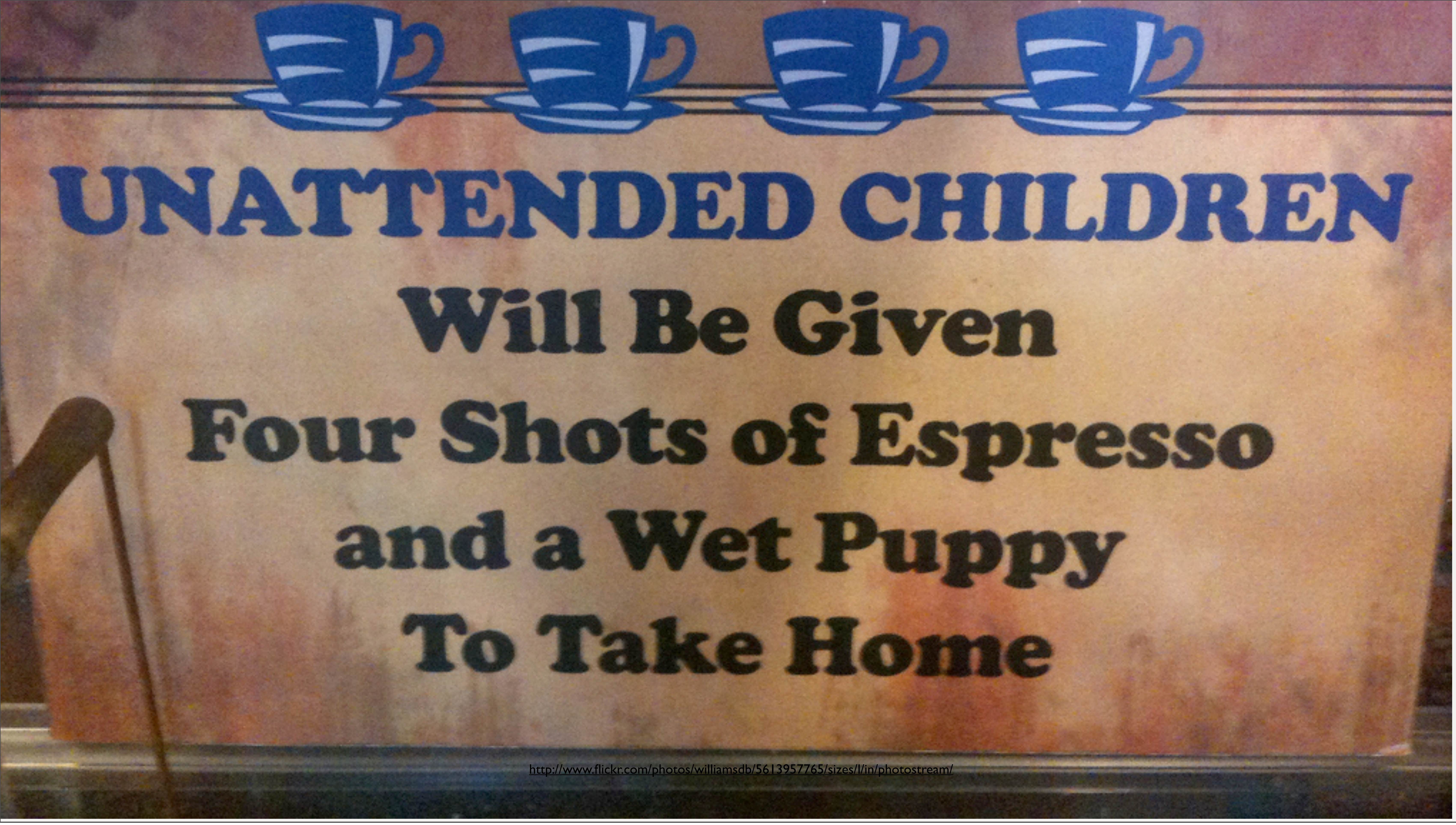
```
do_start_members_sync(Pool, Count) ->
    Parent = self(),
    Pids = [ launch_starter(Parent, Pool)
             || _I <- lists:seq(1, Count) ],
    gather_pids(StarterPids, []).


launch_starter(Parent, Pool) ->
    Fun = ...,
    proc_lib:spawn_link(Fun).
```

Think of the children!

Adding async + parallel member start should be easy. This is Erlang after all.

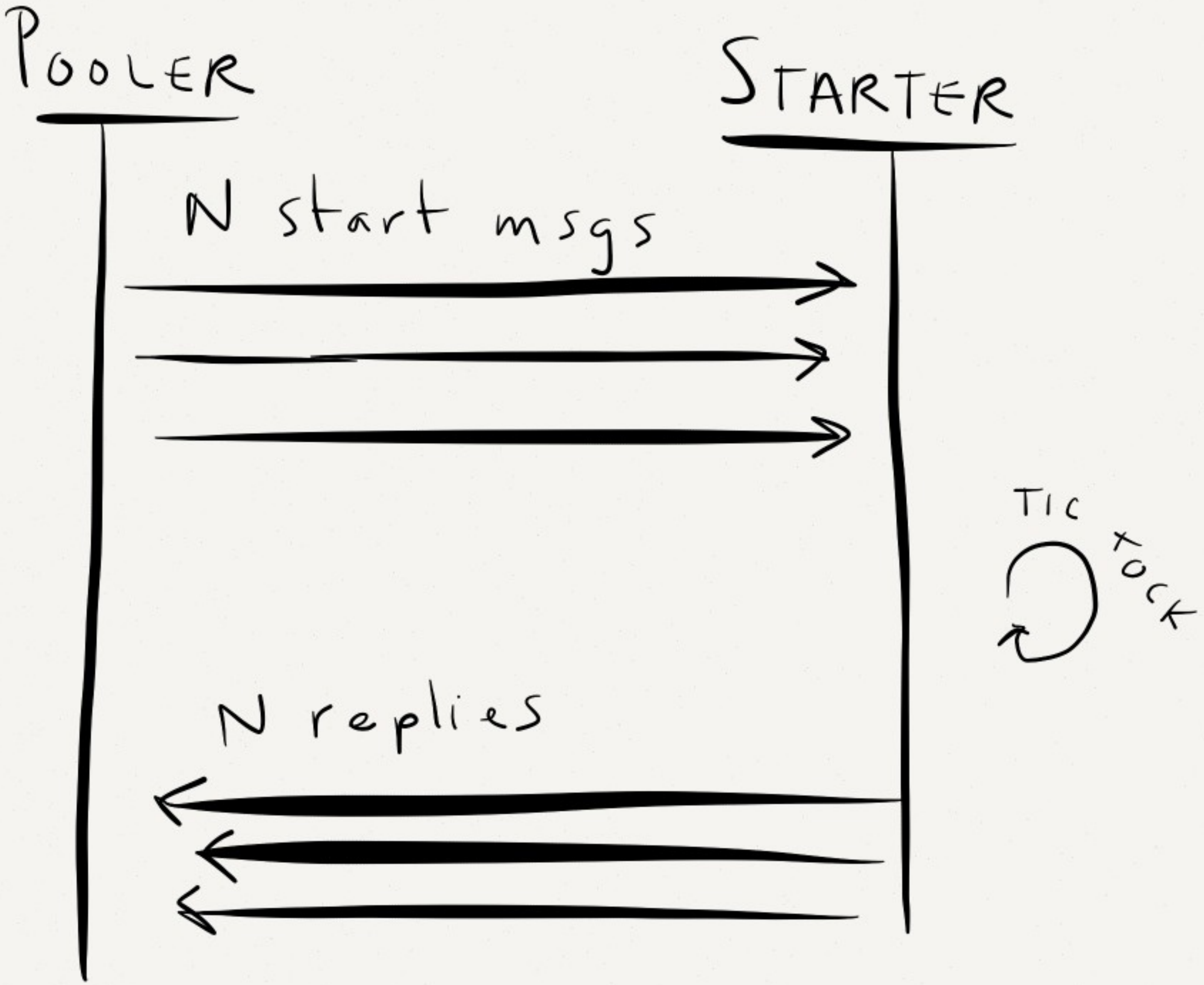
http://www.flickr.com/photos/digitalcolony/5179482430/sizes/l/in/photostream/

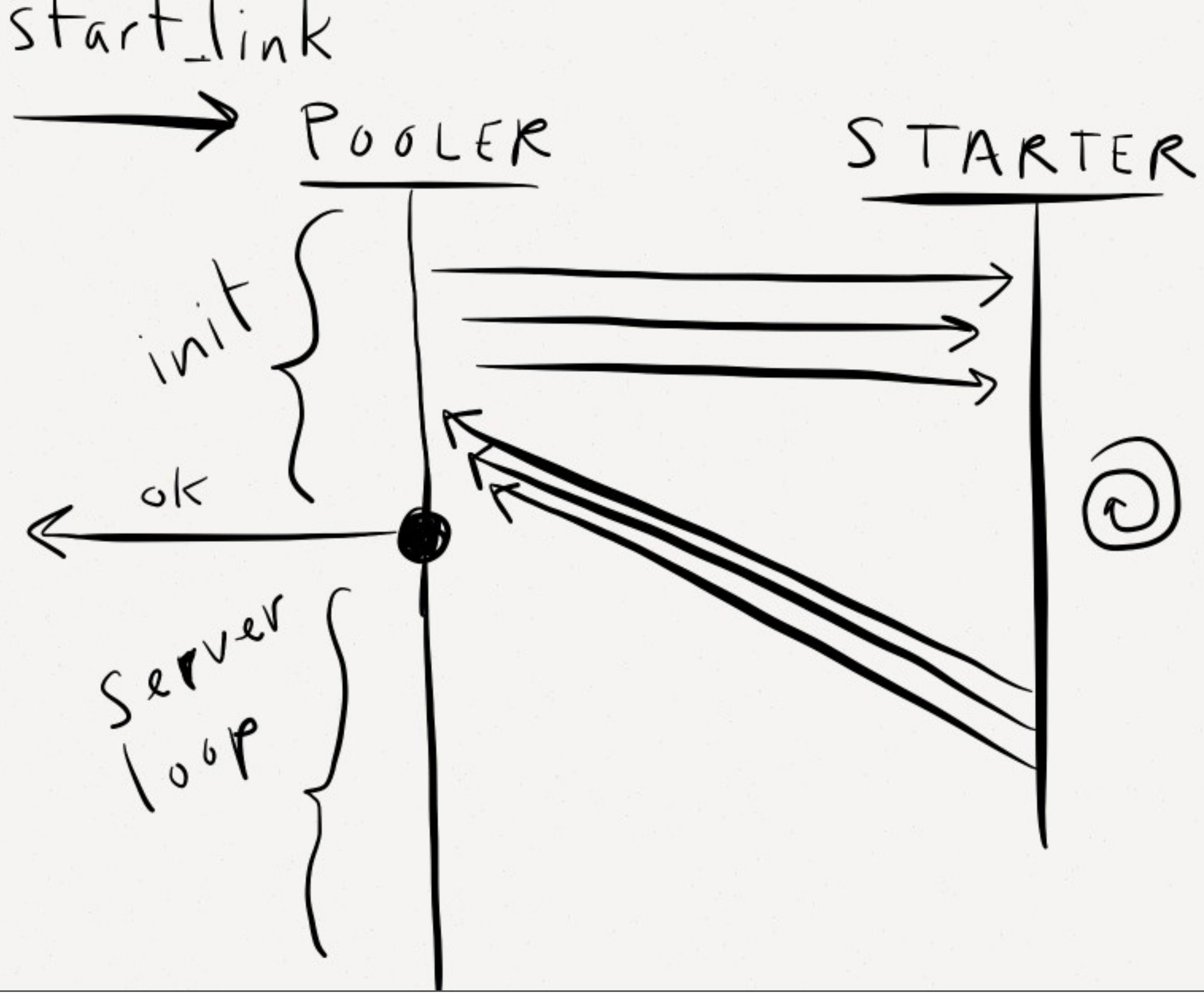
Come on,
just this one time during
init.



UNATTENDED CHILDREN
Will Be Given
Four Shots of Espresso
and a Wet Puppy
To Take Home

http://www.flickr.com/photos/williamsdb/5613957765/sizes/l/in/photostream/


POOLER
STARTER
N start msgs
TIC
TOCK
N replies


start_link
POOLER
STARTER
init
ok
server loop

in init
nobody knows your name

send raw messages in init!

TODO

✓ 1. True multi pool

✓ 2. Async + parallel member start

true multi pool

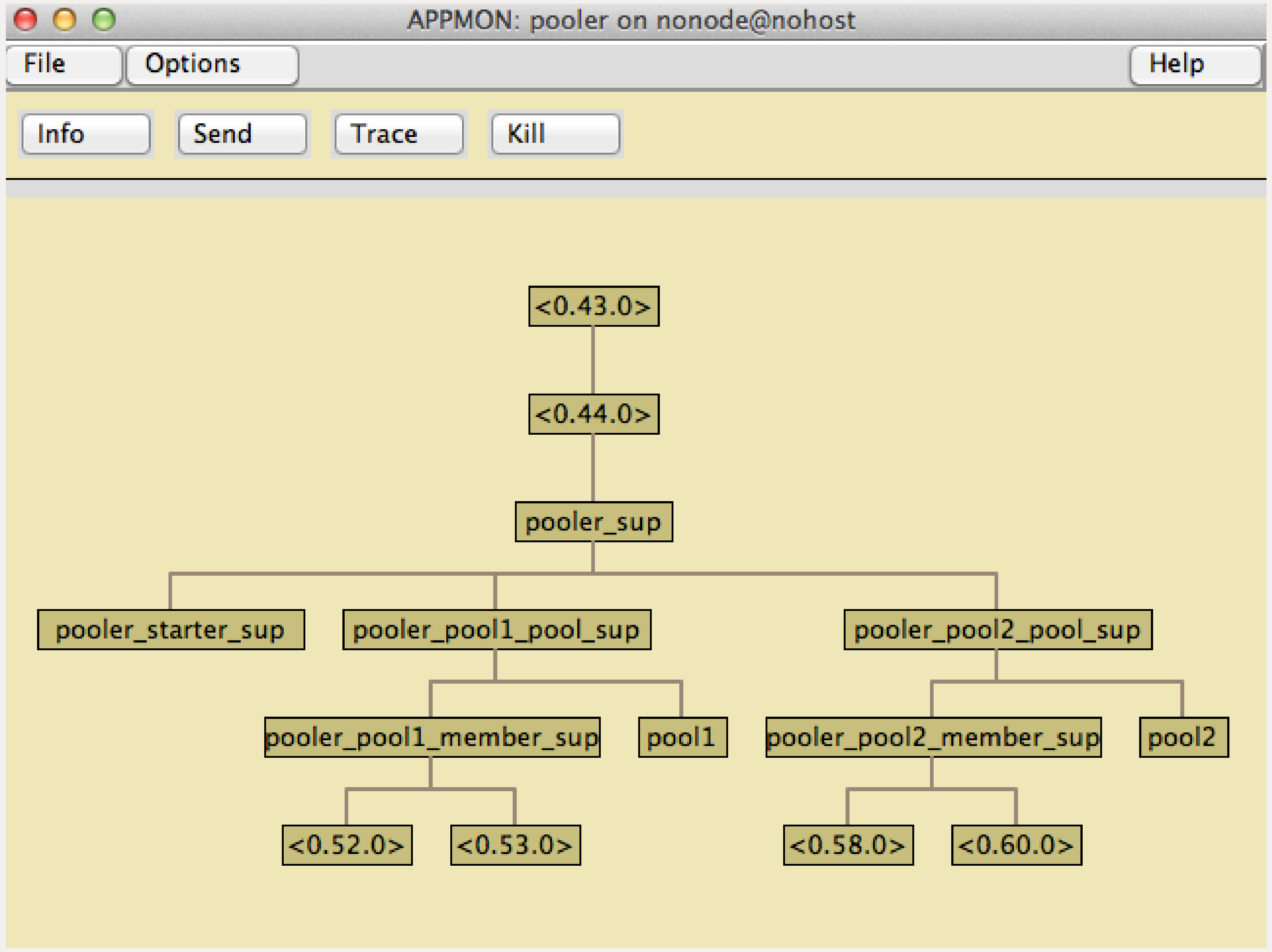
all supervised

dynamic pool size

    add batches

    start timeout

dynamic pool creation



New version now on master. Still a few finishing touches to make some of the dynamic and async features tunable (start timeout, e.g.)
Not tagged yet for release, but expected in next couple of weeks.

OPSCODE

Take Away

- Supervisor Driven Design
- No unsupervised processes
- Create supervisors on the fly
- zero timeout in init trick
- raw send/receive in init

POOL RULES
HOTEL GUESTS ONLY
NO ANIMALS IN POOL AREA
NO FOOD OR DRINKS IN THE POOL OR ON DECK
PLEASE SHOWER BEFORE ENTERING POOL
NO GLASS IN POOL OR ON DECK
NO RUNNING OR RUFF PLAY
USE THE RESTROOM... NOT THE POOL !
OPEN SUN-UP TO SUNDOWN
UNATTENDED CHILDREN WILL BE SOLD AS SLAVES

http://www.flickr.com/photos/joeshlabotnik/321872649/sizes/z/in/photostream/

OPSCODE

Thank You.

https://github.com/seth/pooler

@sfalcon