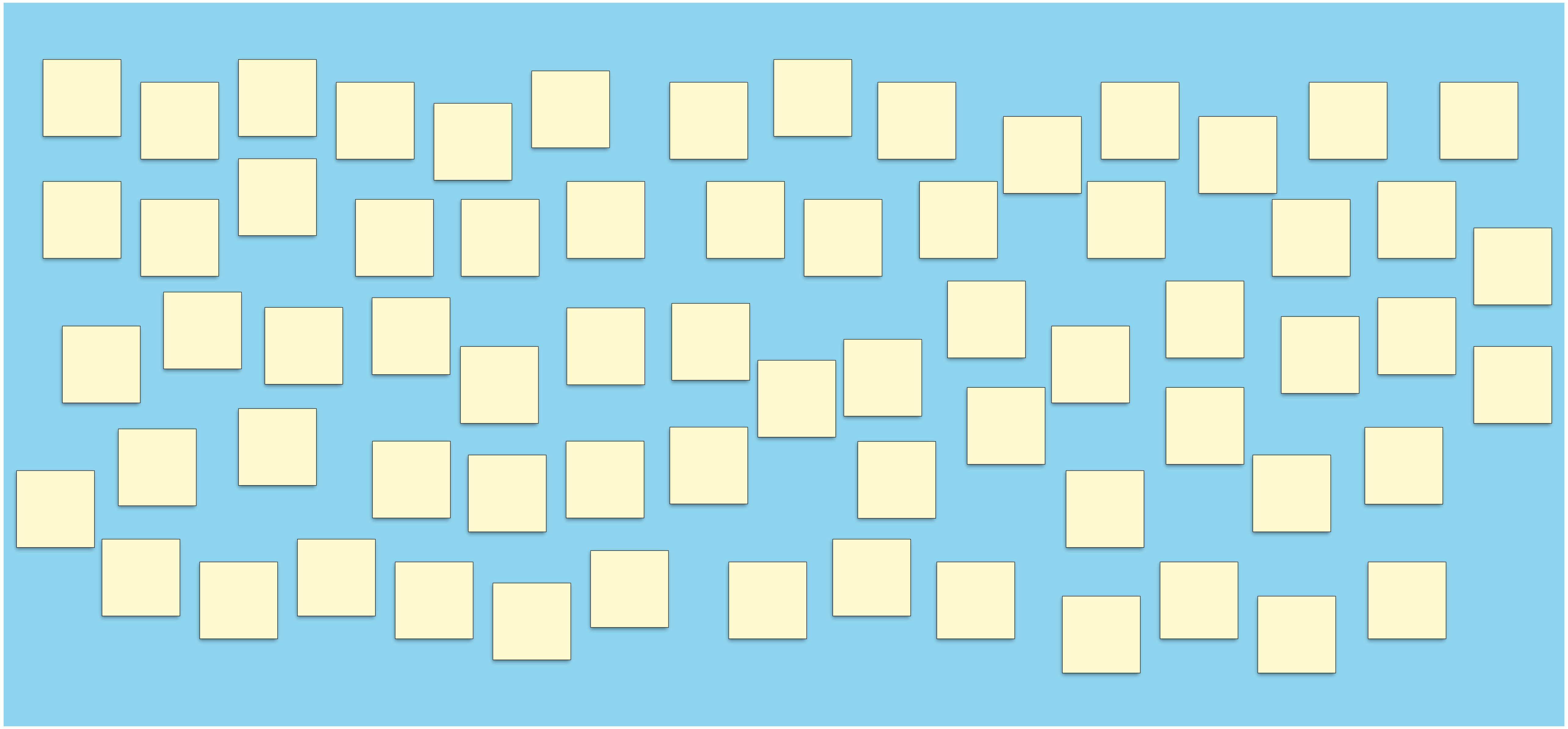
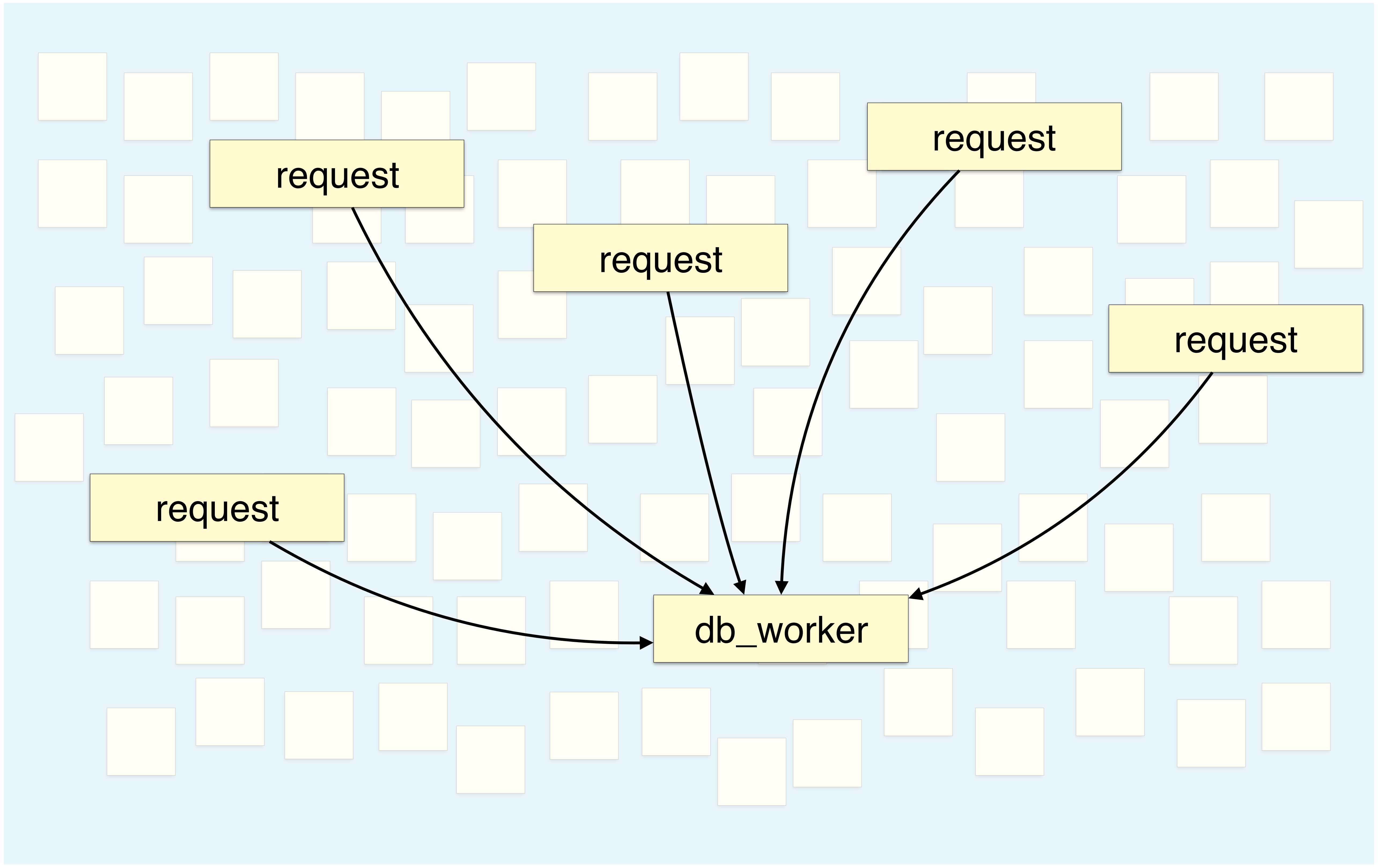
discovering processes

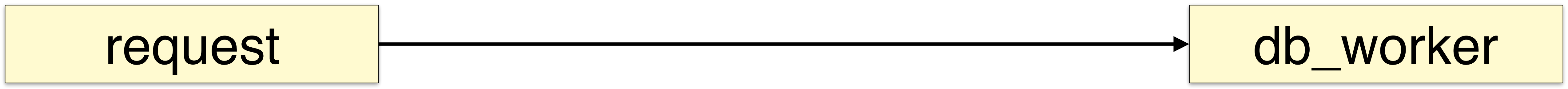
@sasajuric

aircloak.com

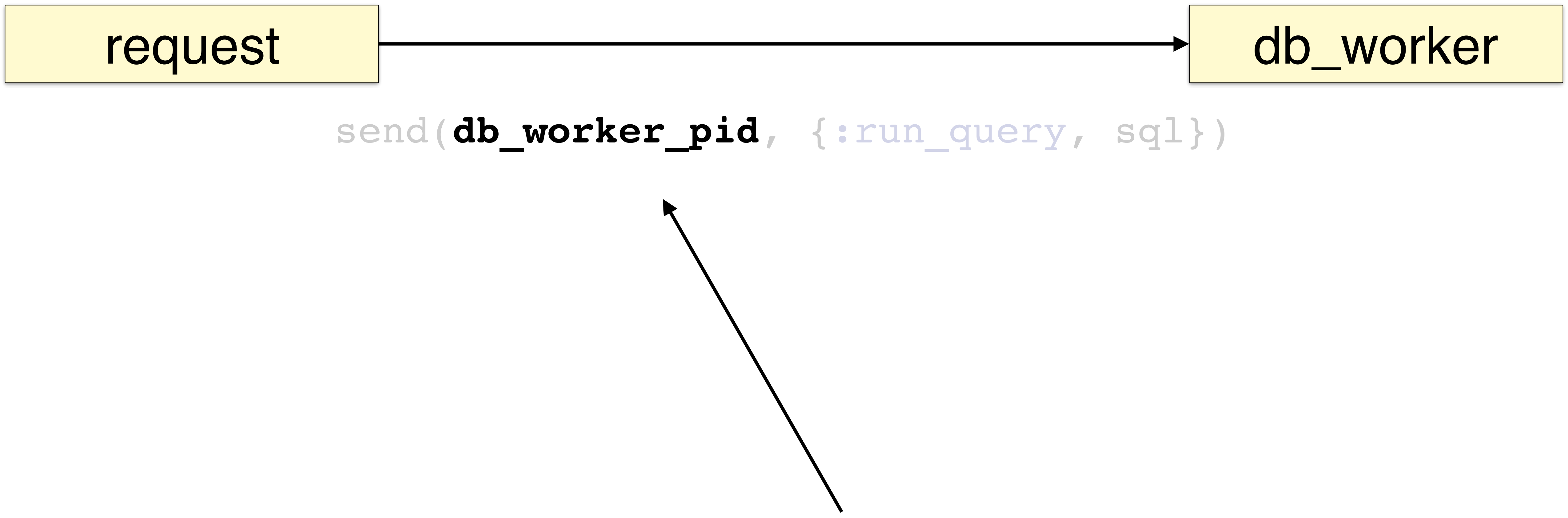


“Erlang powered system”


request
request
request
request
request
db_worker



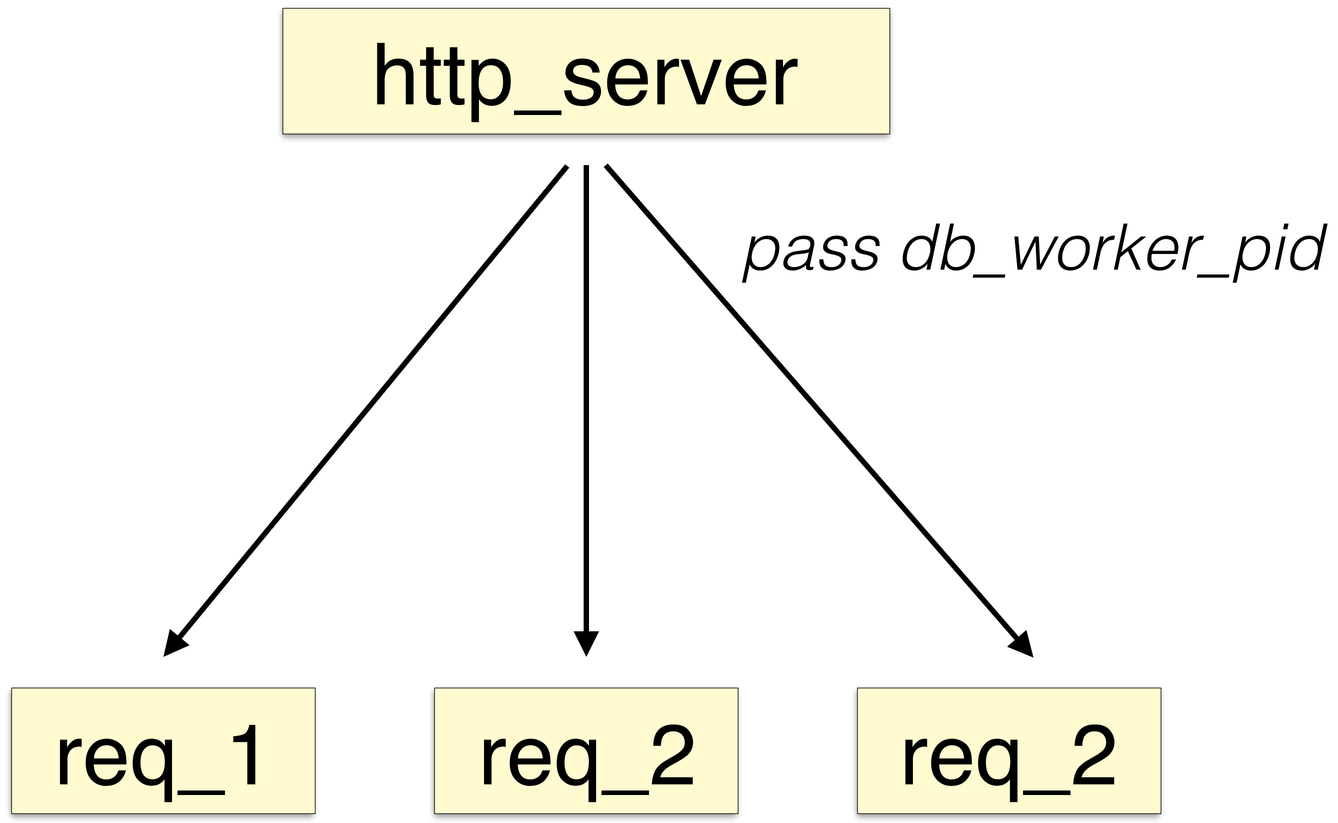
```
send(db_worker_pid, {:run_query, sql})
```

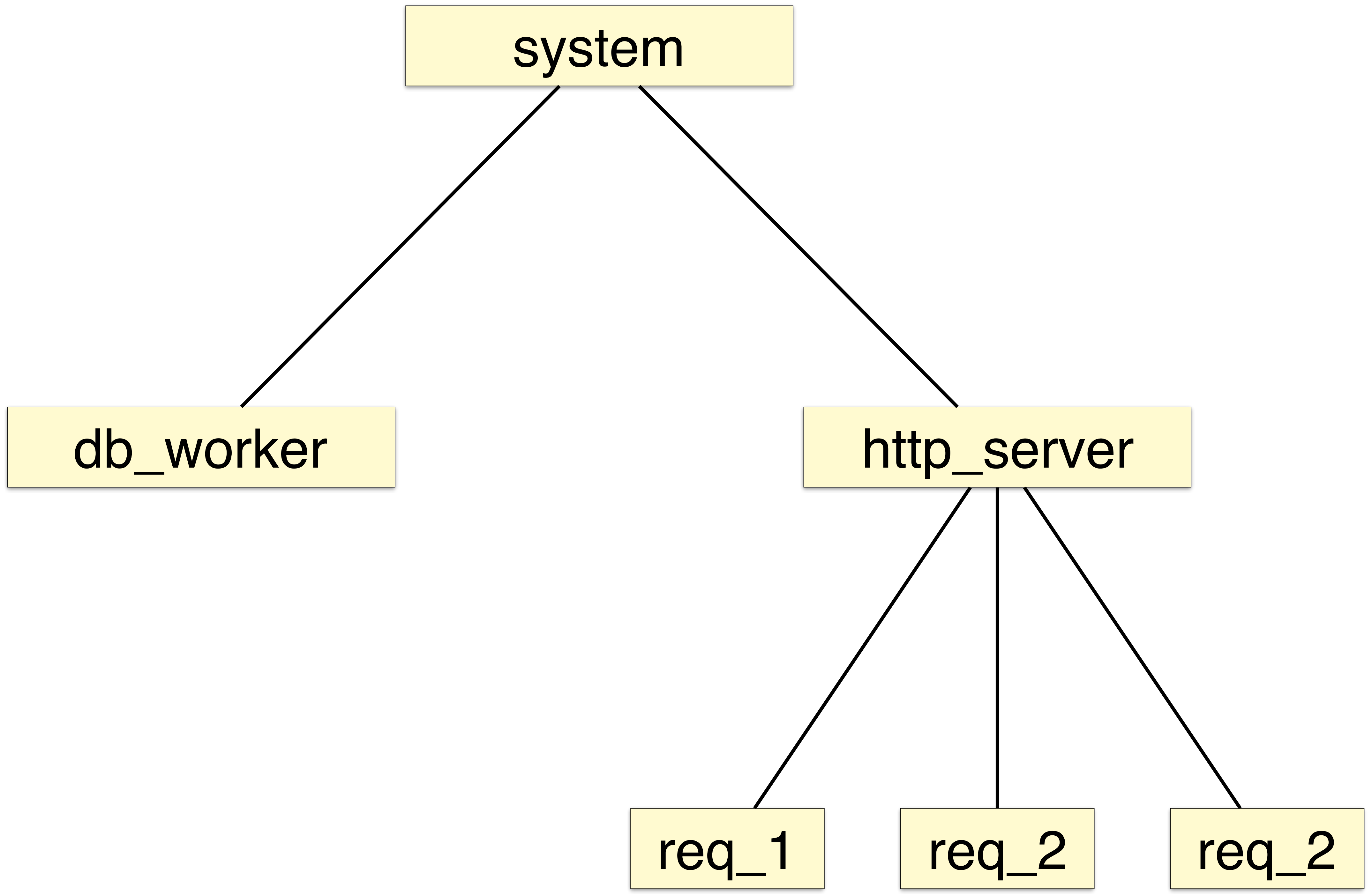


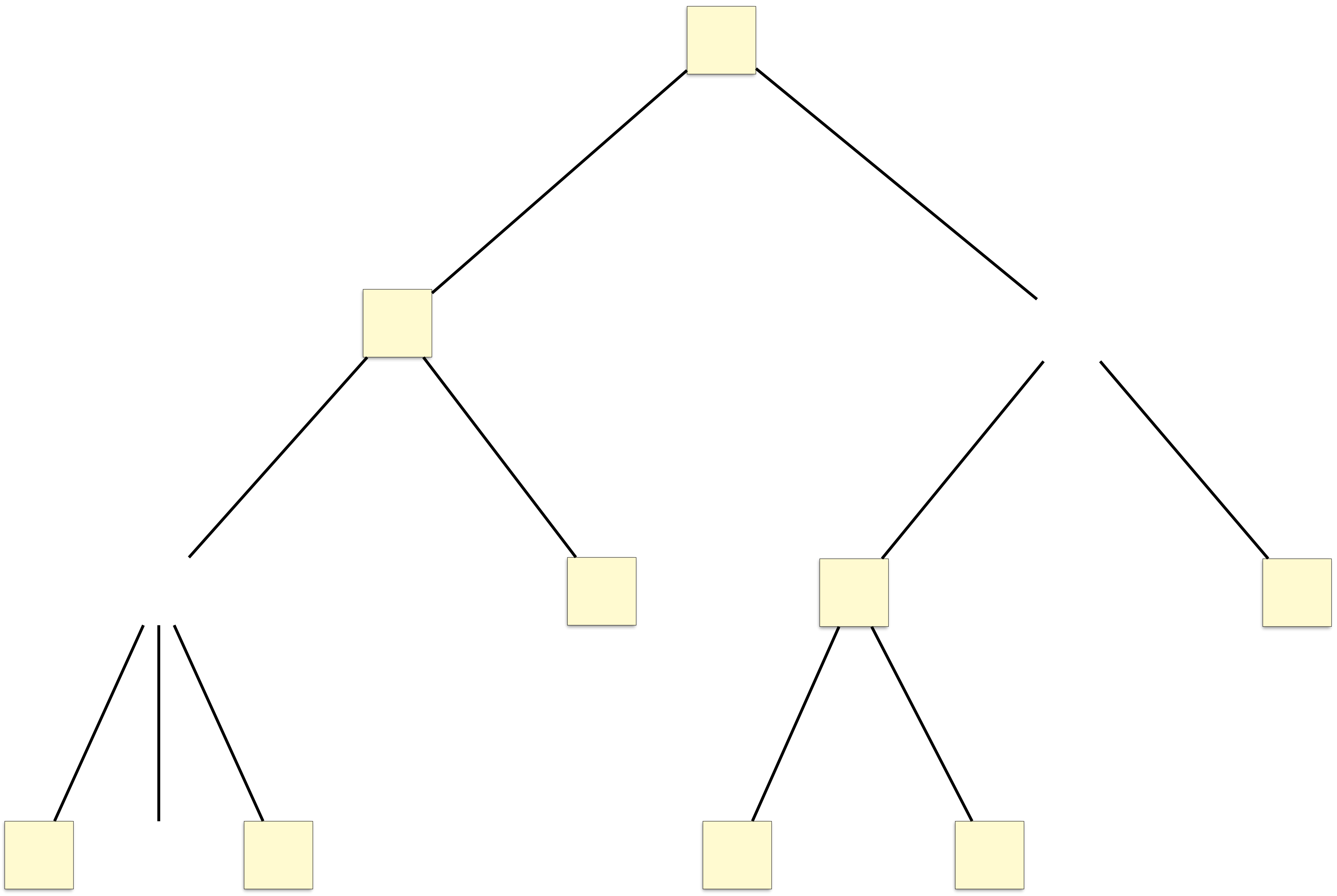
```
send(db_worker_pid, {:run_query, sql})
```

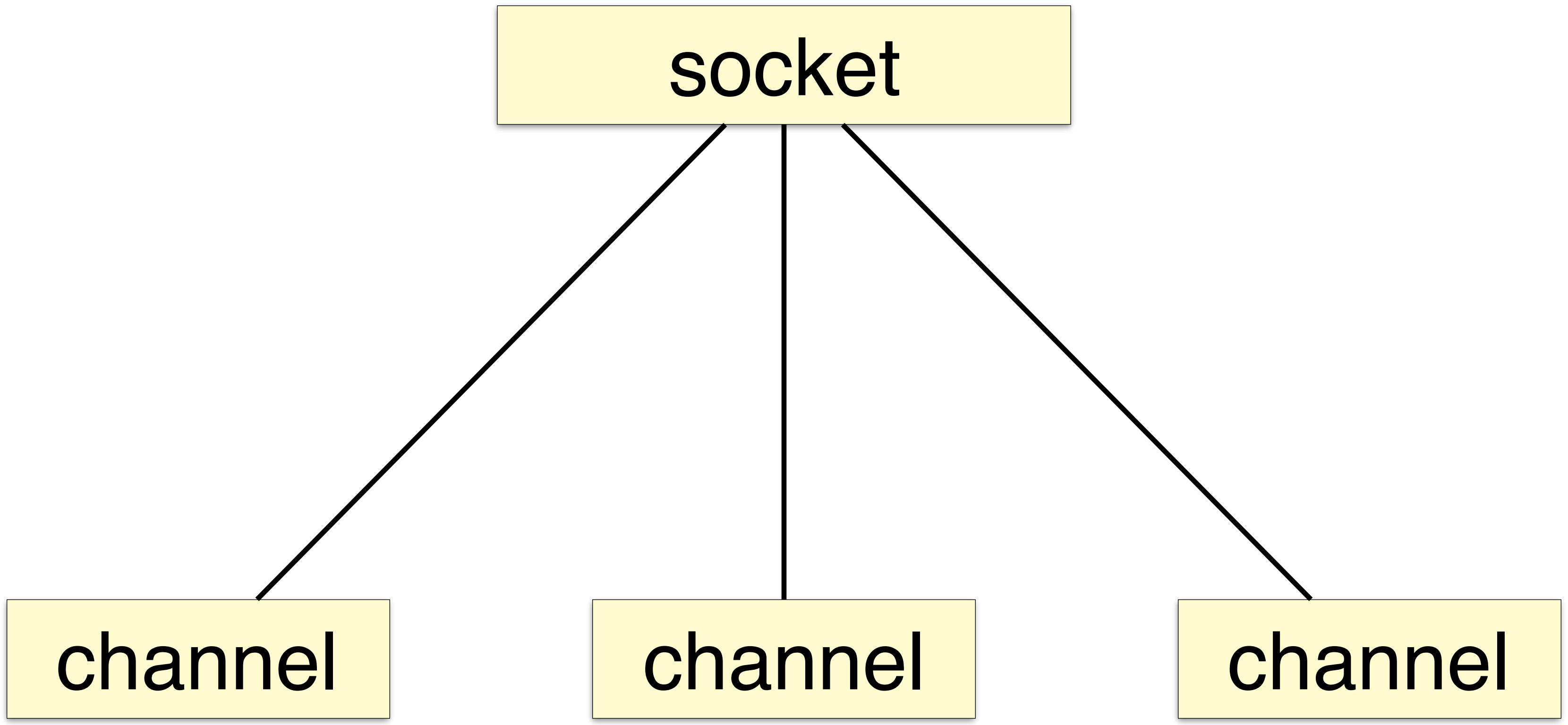
How do we get this ???

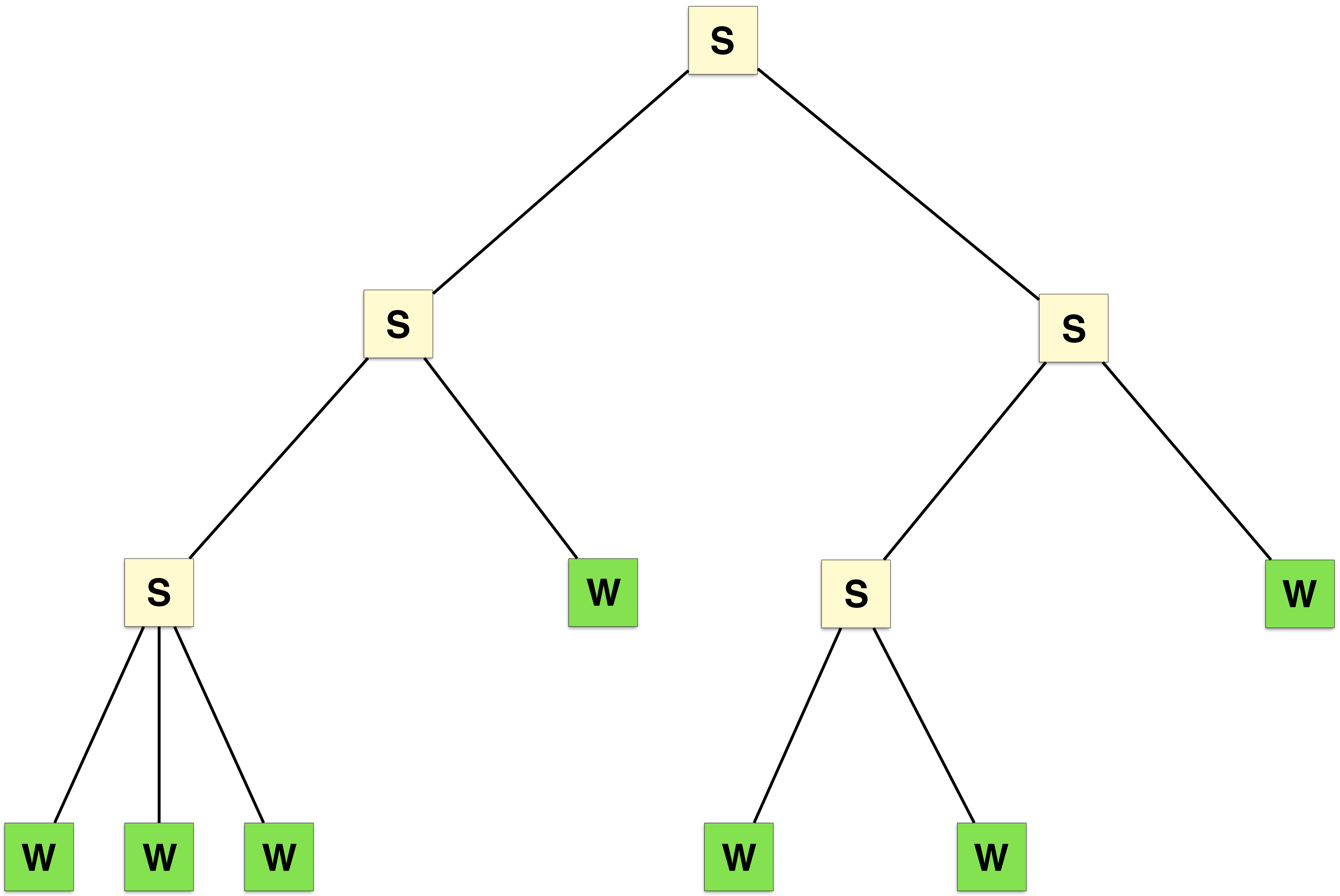
```elixir
{:ok, db_worker_pid} = DbWorker.start(...)

HttpServer.start(db_worker_pid, ...)
```

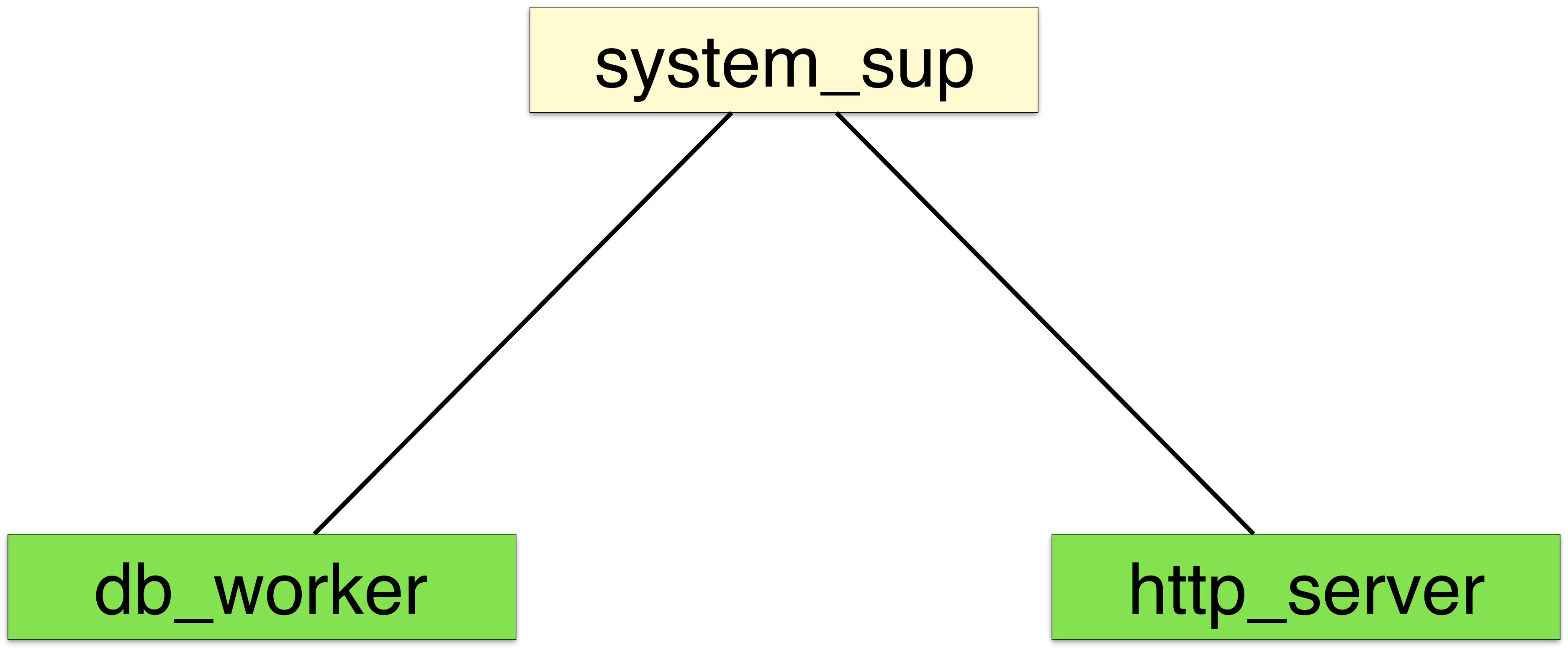

http_server
pass db_worker_pid
req_1
req_2
req_2


system
db_worker
http_server
req_1
req_2
req_2



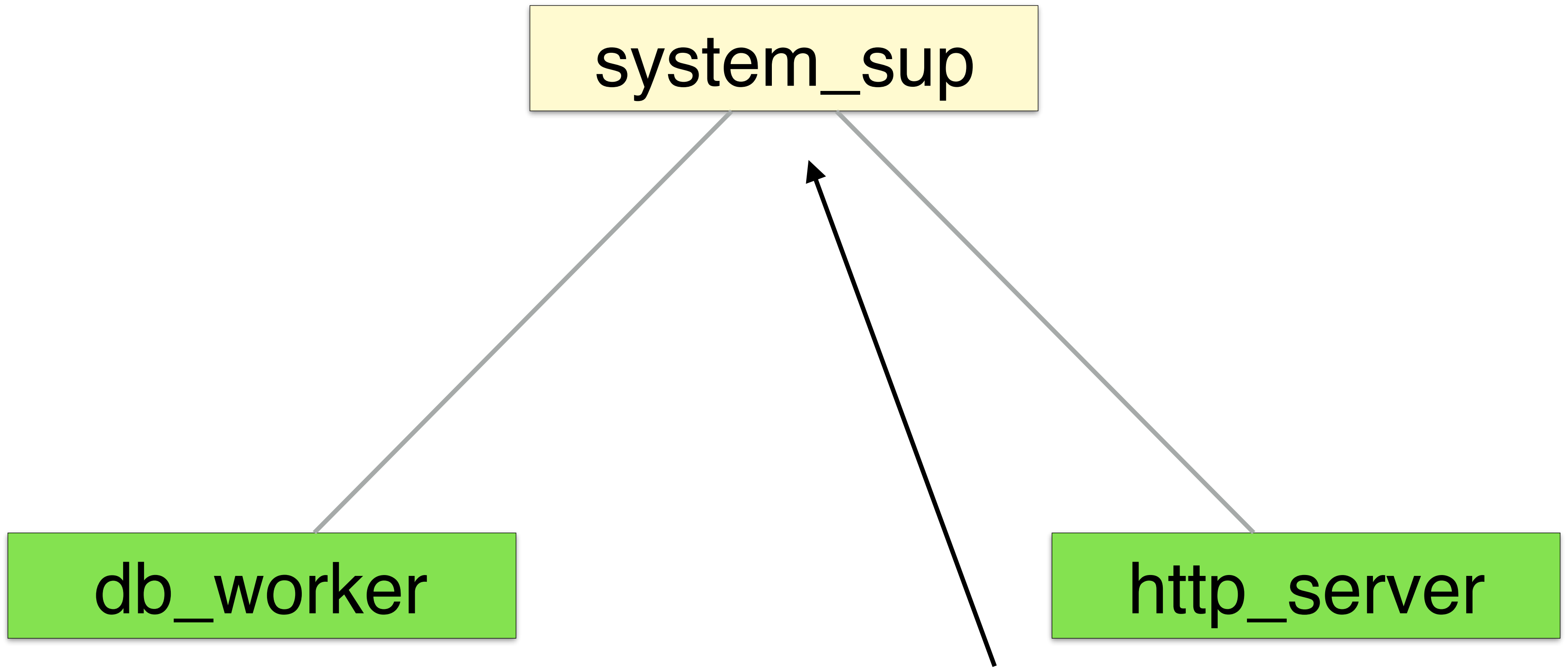

socket
channel
channel
channel


S
S
S
S
W
S
W
W
W
W
W
W


system_sup
db_worker
http_server

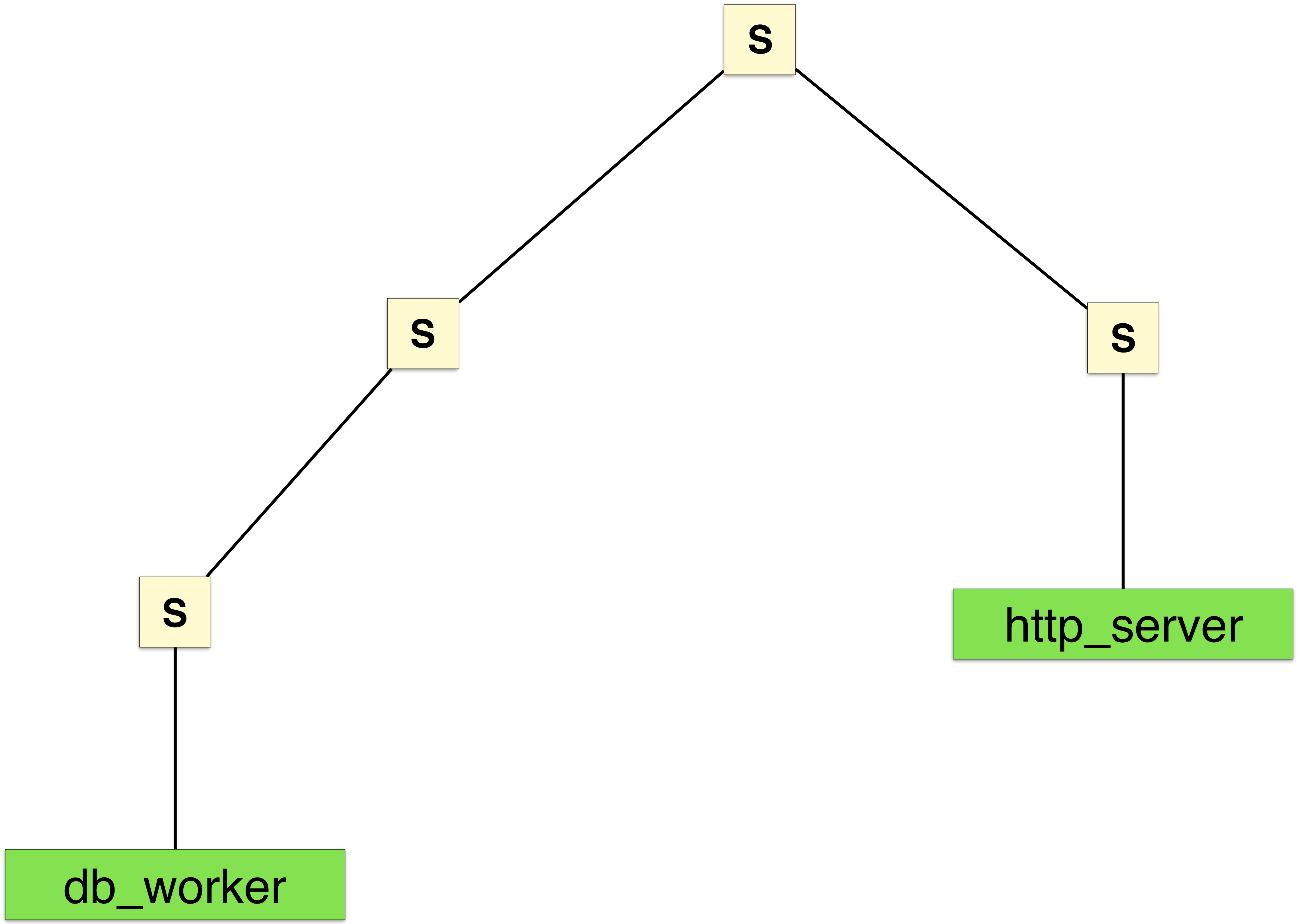
```
Supervisor.start_link(
  [
    worker(DbWorker, [...]),
    worker(HttpServer, [...])
  ],
  strategy: :one_for_one
)
```

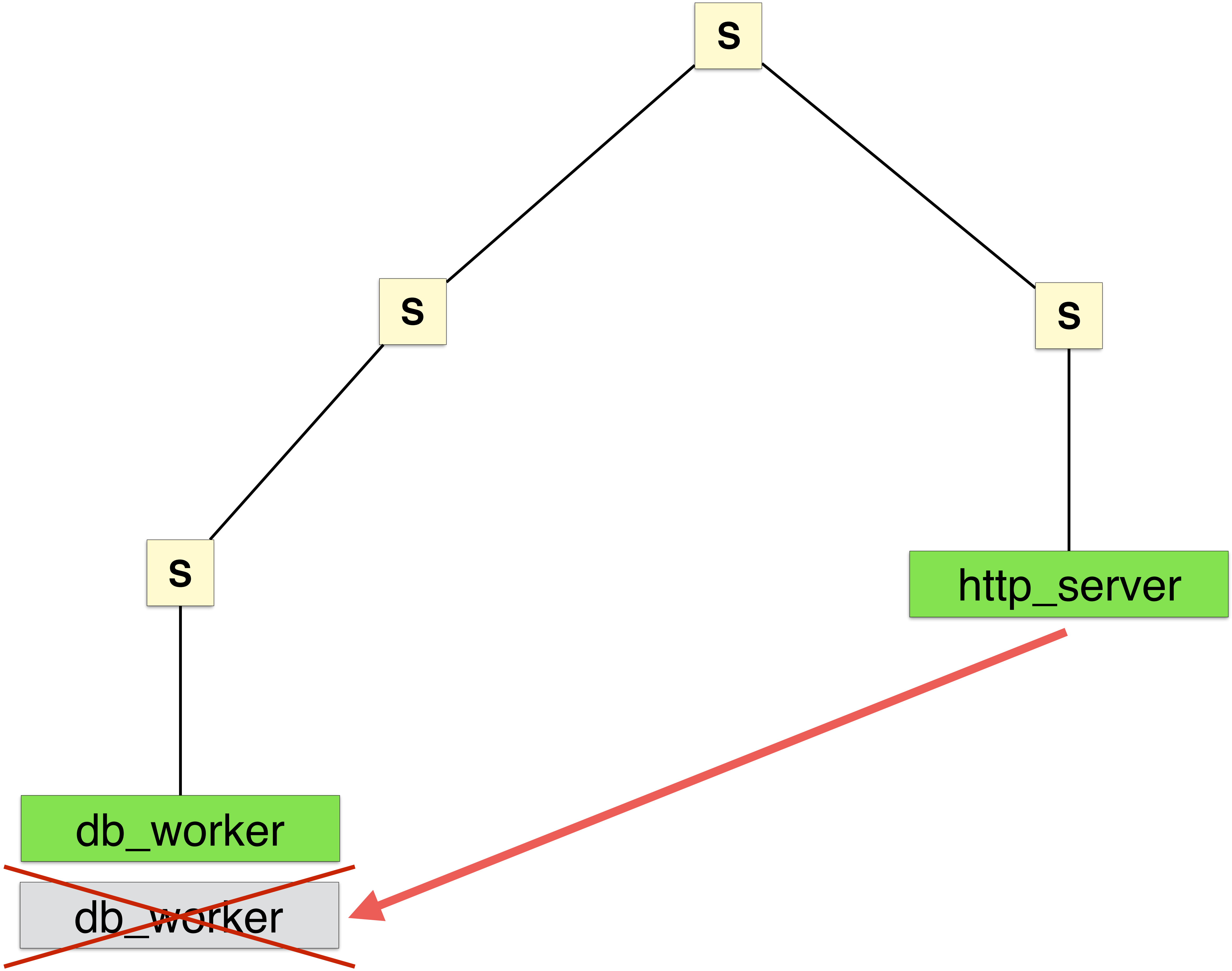
```
Supervisor.which_children(supervisor_pid)
|> Enum.find(...)
```

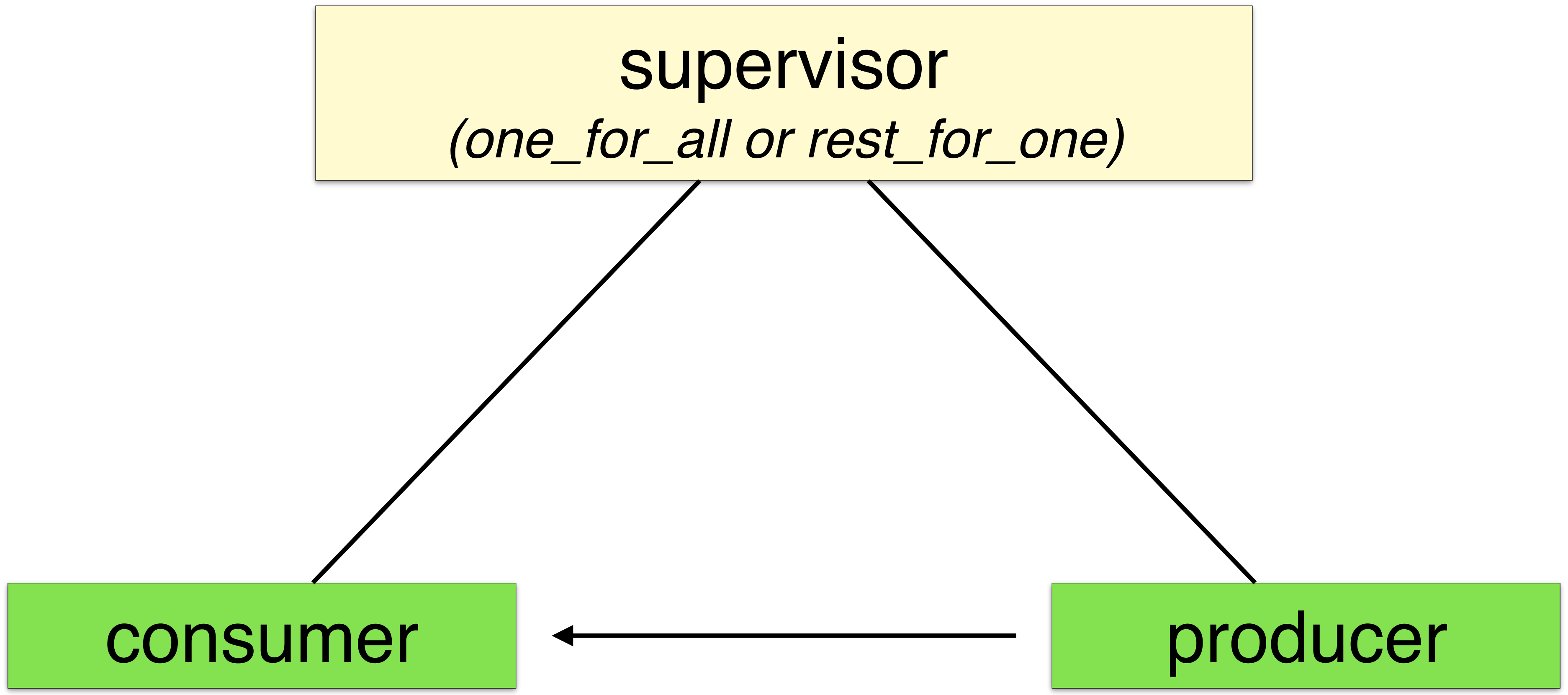


```
Supervisor.find_children(
  parent_pid
)
```

```elixir
def init(_) do
  send(self(), :get_db_worker)
  ...
end

def handle_info(:get_db_worker, state) do
  Supervisor.which_children(parent_pid)
  |> find_db_worker
  |> store_to_state
end
```


S
S
S
S
http_server
db_worker


S
S
S
S
http_server
db_worker
~~db_worker~~


supervisor
(one_for_all or rest_for_one)
consumer
producer

```
i_am(:db_worker)
```

```
who_is(:db_worker)
```

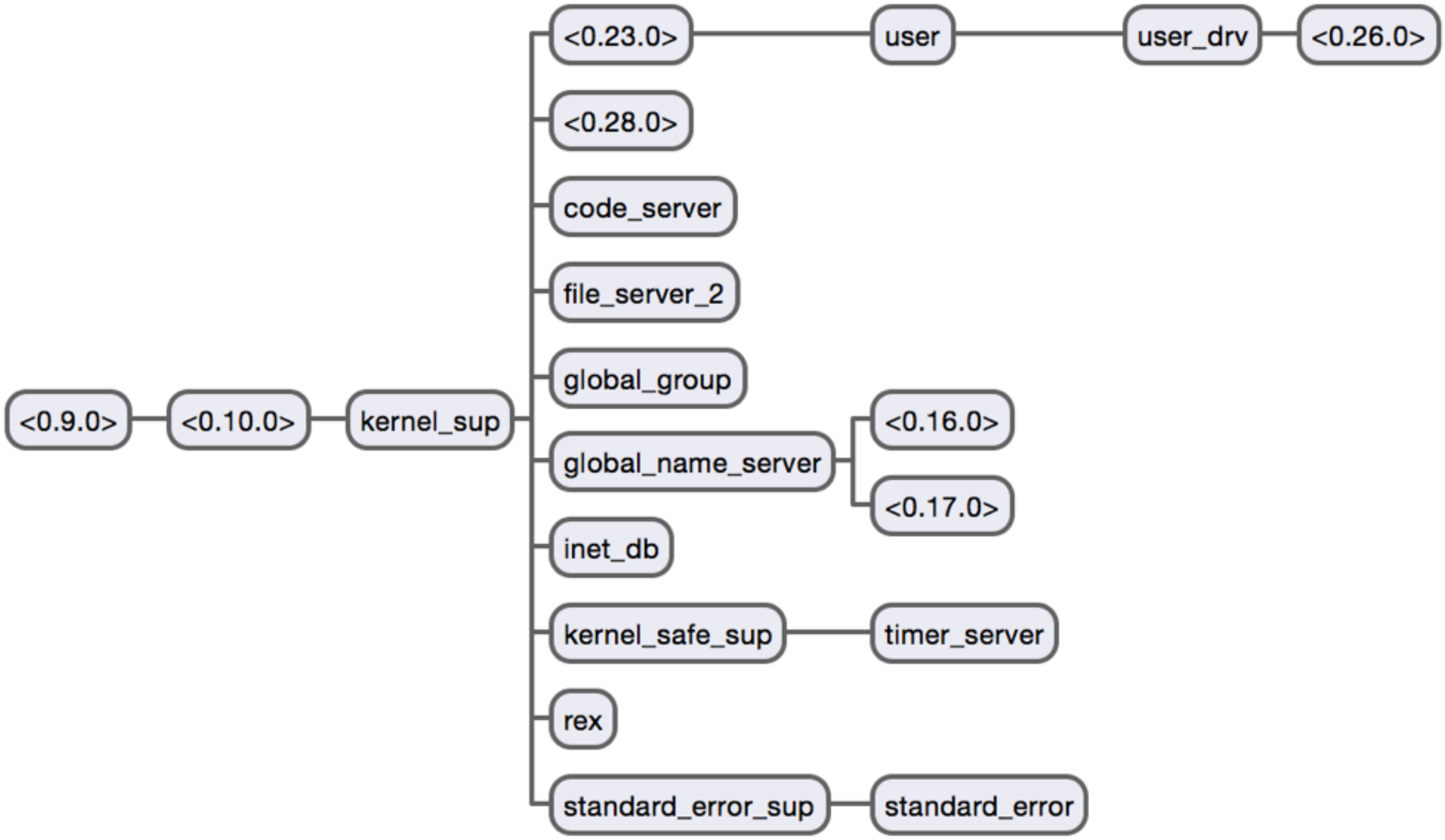
```
send(who_is(:db_worker), ...)
```

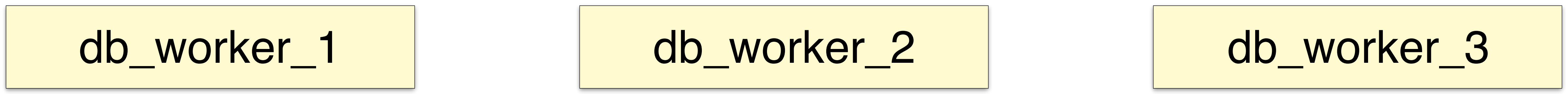
```
Process.register(self(), :db_worker)
```

```
...
send(:db_worker, {:run_query, sql})
...
```

```elixir
defmodule DbWorker do
  def start_link do
    GenServer.start_link(..., name: :db_worker)
  end

  def run_query(sql) do
    GenServer.call(:db_worker, {:run_query, sql})
  end
end
```

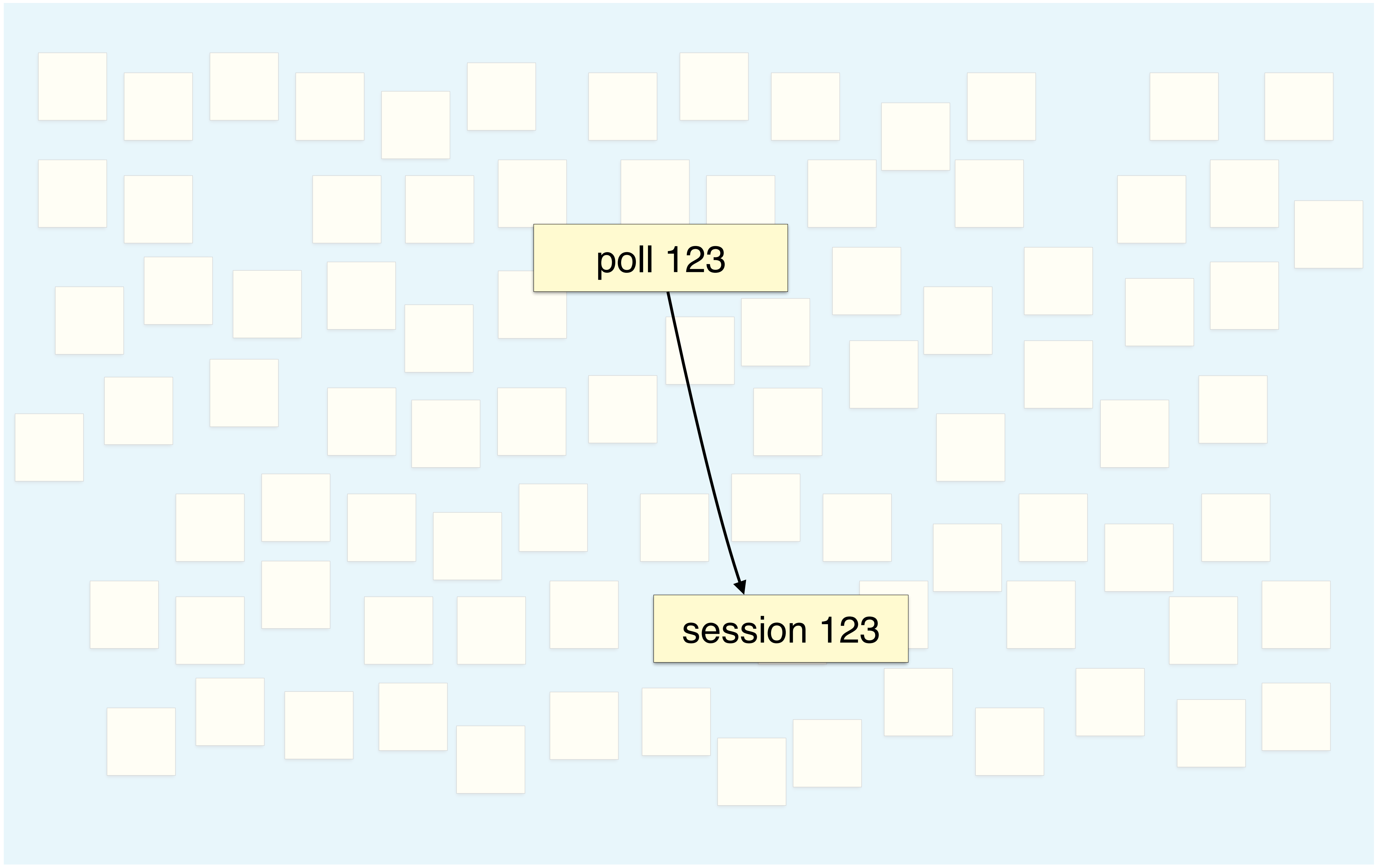
```
# in the request process
DbWorker.run_query("...")
```


<0.9.0>
<0.10.0>
kernel_sup
<0.23.0>
user
user_drv
<0.26.0>
<0.28.0>
code_server
file_server_2
global_group
global_name_server
<0.16.0>
<0.17.0>
inet_db
kernel_safe_sup
timer_server
rex
standard_error_sup
standard_error


db_worker_1
db_worker_2
db_worker_3

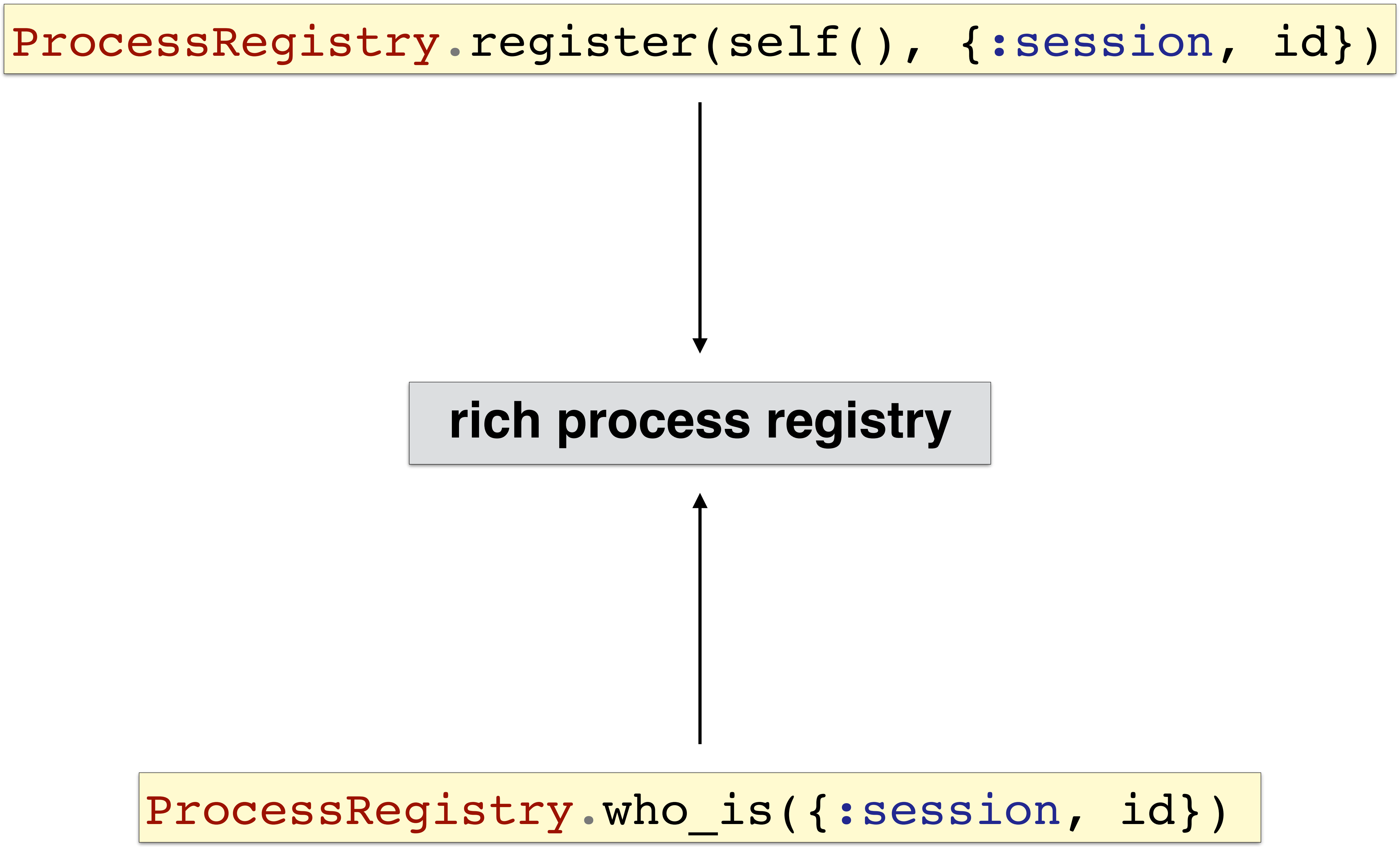
```
defmodule DbWorker do
  def start_link(worker_num) do
    GenServer.start_link(
      ...,
      name: :"db_worker_#{worker_number}"
    )
  end
end
```

```elixir
def run_query(query) do
  worker_number = pick_worker(...)
  GenServer.call(
    :"db_worker_#{worker_number}",
    ...
  )
end
```

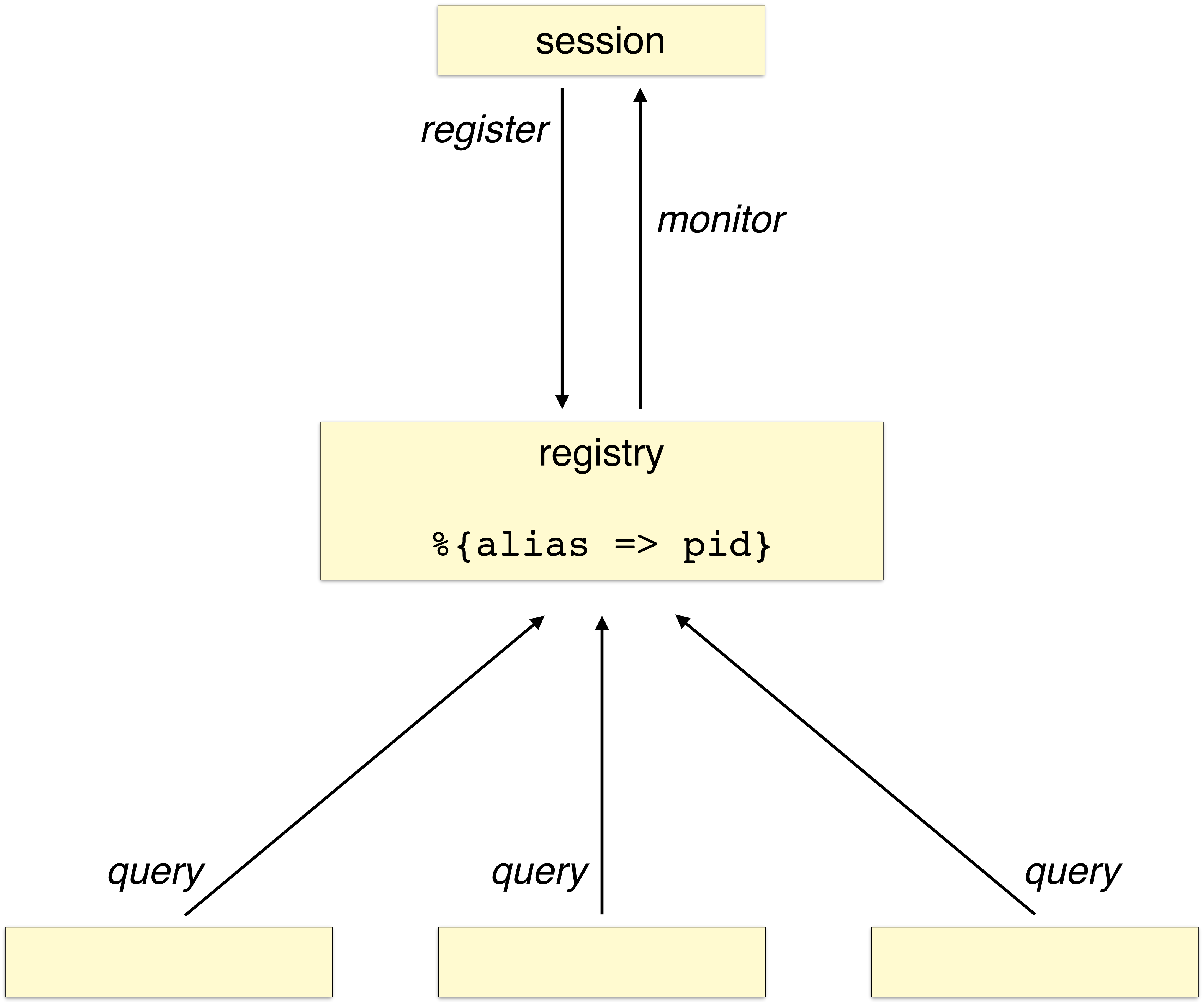

poll 123
session 123

```elixir
Process.register(self(), :"session_#{id}")
```

```elixir
send(:"session_#{id}", ...)
```


```
ProcessRegistry.register(self(), {:session, id})
```

```
ProcessRegistry.who_is({:session, id})
```


session
register
monitor
registry
%{alias => pid}
query
query
query

gproc

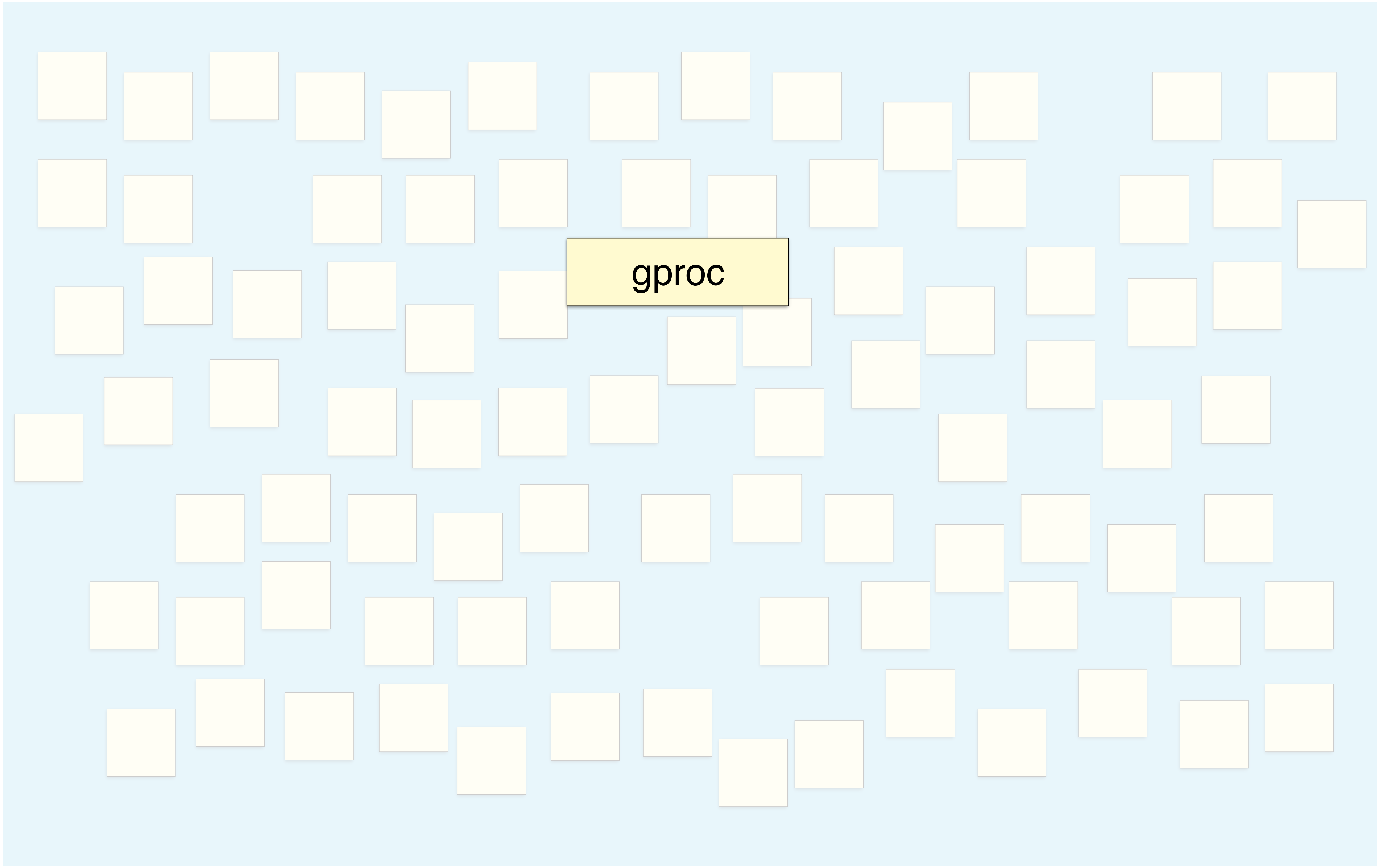
https://hex.pm/packages/gproc

https://github.com/uwiger/gproc

gproc

- rich unique aliases (names)
- rich non-unique aliases (properties)

```
defp deps do
  [{:gproc, "~> 0.5"}, ...]
end

def application do
  [applications: [:gproc, ...]]
end
```


gproc

```
:gproc.reg({
  :n,                 # unique name
  :l,                 # local registration
  {:session, 123}     # alias
})
```

```elixir
:gproc.where({:n, :l, {:session, 123}})
# :: pid | :undefined
```

```elixir
GenServer.start_link(module, arg,
  name: via_tuple
)

GenServer.call(via_tuple, request)
```

```
via_tuple = {
  :via,
  RegistryMod,
  arg
}
```

```
via_tuple = {
  :via,
  :gproc,
  {:n, :l, {:session, 123}}
}
```

```
defmodule Session do
  defp name(session_id) do
    {
      :via,
      :gproc,
      {:n, :l, {:session, session_id}}
    }
  end

  ...
end
```

```elixir
def start_link(session_id) do
  GenServer.start_link(
    module, arg,
    name: name(session_id)
  )
end
```

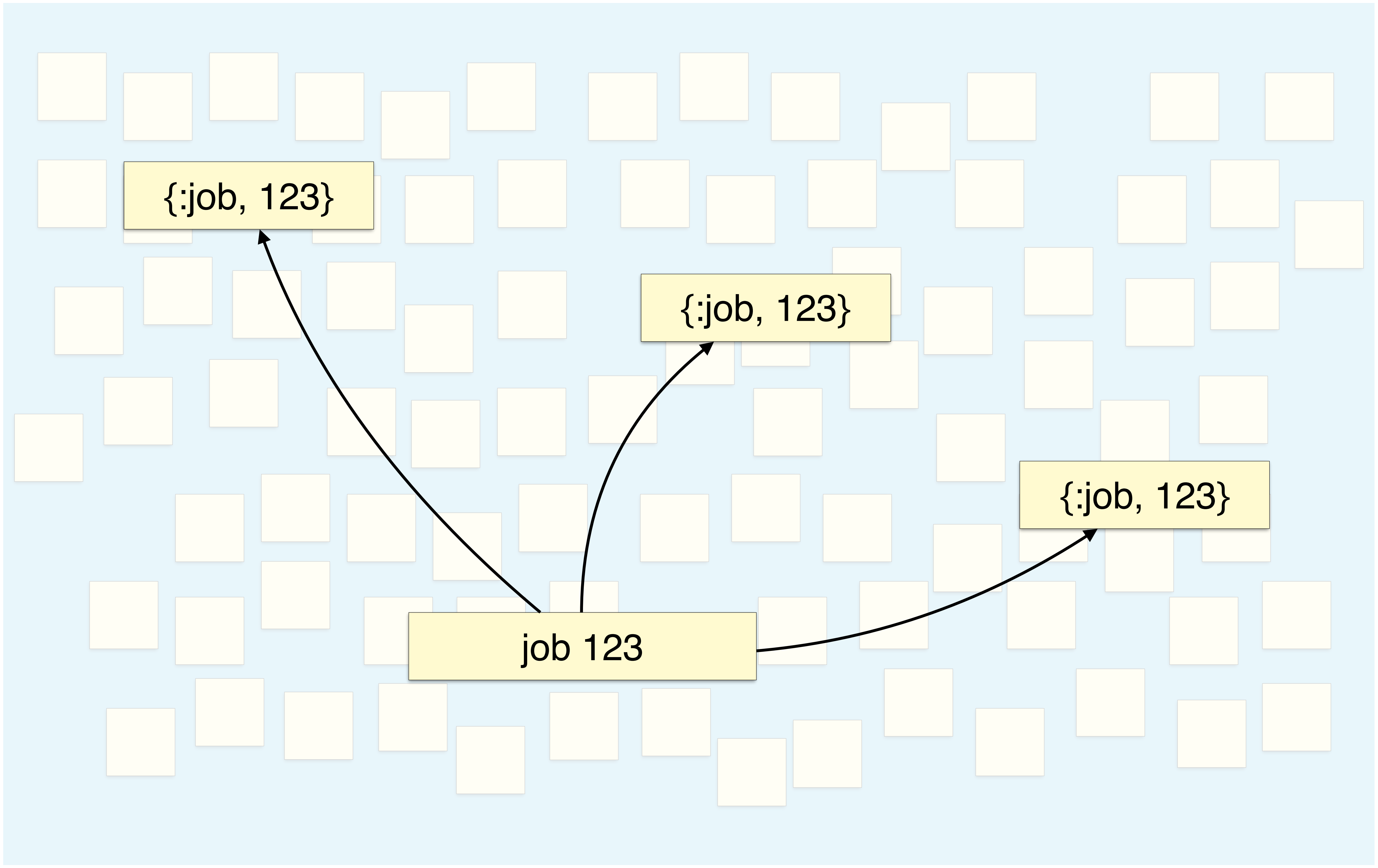
```elixir
def get_messages(session_id) do
  GenServer.call(
    name(session_id),
    :get_messages
  )
end
```

```
Session.get_messages(session_id)
```

name 1
name 2
…

name 1
name 2
…

property 1
property 2
…


{:job, 123}
{:job, 123}
{:job, 123}
job 123

```
:gproc.reg({
  :p,
  :l,
  {:job, job_id}
})
```

```elixir
:gproc.lookup_pids({:p, :l, {:job, 123}})
# :: [pid]
```

```
:gproc.send(
  {:p, :l, {:job, job_id}},
  message
)
```

rich local registration	- gproc
rich global registration	- global - pg2 - Phoenix PubSub

what?	when?
startup discovery	- small scope - high coupling - all-or-nothing
simple registration	- statical services
dynamic atoms	- a few instances - finite set of possible aliases
rich registration	- many instances - unknown number of aliases

Elixir IN ACTION

Saša Jurić

MANNING