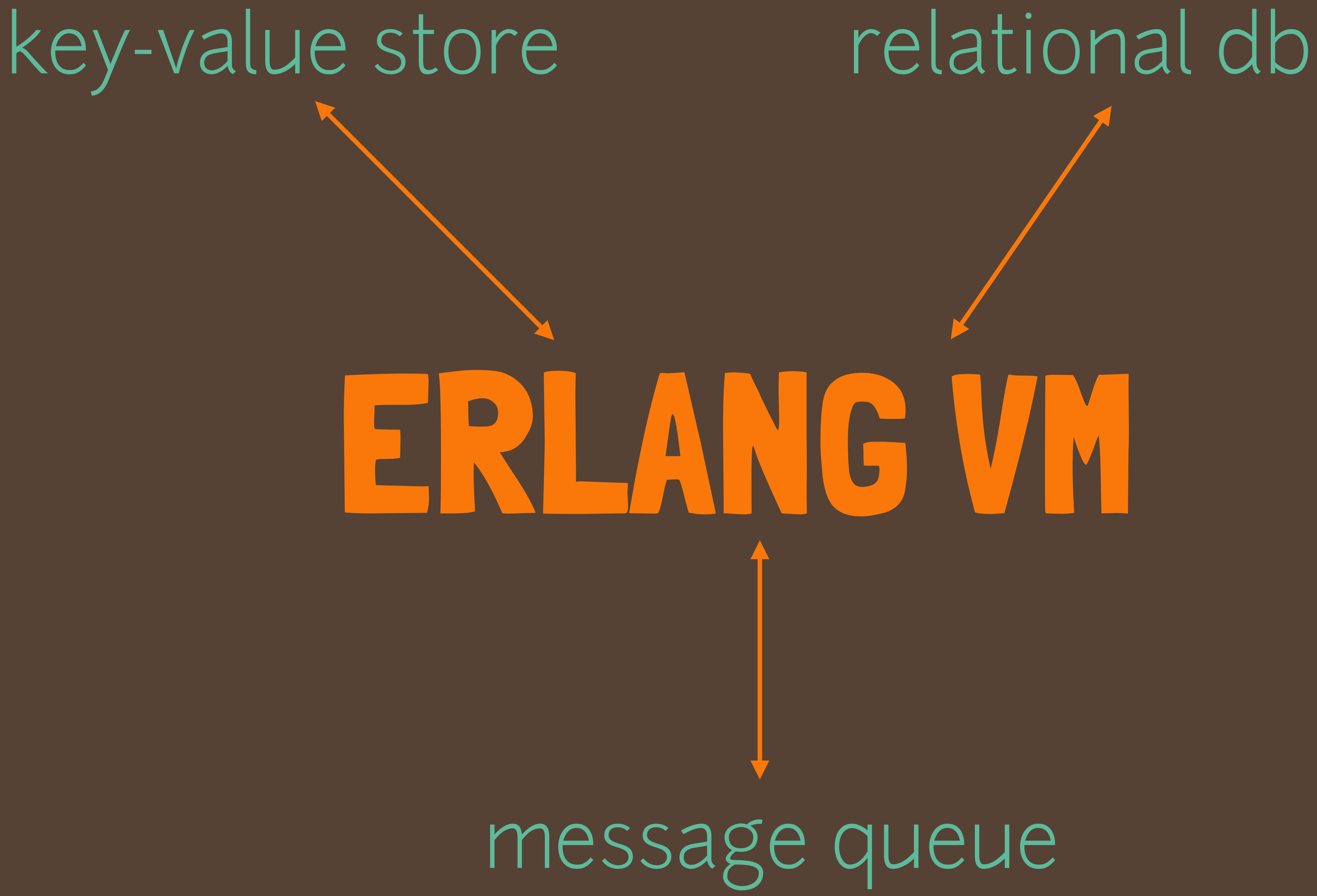
CONCURRENT *and* RESILIENT CONNECTIONS *to* *outside the BEAM*

$$\begin{tikzcd}
\text{ets} \arrow[dr, leftrightarrow] & & \text{mnesia} \arrow[dl, leftrightarrow] \\
& \text{ERLANG VM} \arrow[d, leftrightarrow] & \\
& \text{message passing} &
\end{tikzcd}$$


key-value store
relational db
ERLANG VM
message queue

$$\begin{tikzcd}
\text{key-value store} \arrow[dr, leftrightarrow] & & \text{relational db} \arrow[dl, leftrightarrow] \\
& \text{ERLANG VM} \arrow[d, leftrightarrow] & \\
& \text{message queue} &
\end{tikzcd}$$

the outside world is SCARY

ANDREA LEOPARDI
@WHATYOUHIDE

Gothenburg,
Sweden

FOOTBALL ADDICTS

GENSERVER

TCP

the SILLY *way*

```elixir
defmodule Redis do
  def command(cmd) do






  end
end
```

```elixir
defmodule Redis do
  def command(cmd) do
    {:ok, sock} = :gen_tcp.connect(...)
    :gen_tcp.send(sock, encode(cmd))
    {:ok, data} = :gen_tcp.recv(sock, 0)
    :gen_tcp.close(sock)
    decode(data)
  end
end
```

```
defmodule Redis do
  def command(cmd) do
    {:ok, sock} = :gen_tcp.connect(...)
    :gen_tcp.send(sock, encode(cmd))
    {:ok, data} = :gen_tcp.recv(sock, 0)
    :gen_tcp.close(sock)
    decode(data)
  end
end
```

```elixir
defmodule Redis do
  def command(cmd) do
    {:ok, sock} = :gen_tcp.connect(...)
    :gen_tcp.send(sock, encode(cmd))
    {:ok, data} = :gen_tcp.recv(sock, 0)
    :gen_tcp.close(sock)
    decode(data)
  end
end
```

```elixir
defmodule Redis do
  def command(cmd) do
    {:ok, sock} = :gen_tcp.connect(...)
    :gen_tcp.send(sock, encode(cmd))
    {:ok, data} = :gen_tcp.recv(sock, 0)
    :gen_tcp.close(sock)
    decode(data)
  end
end
```

```elixir
defmodule Redis do
  def command(cmd) do
    {:ok, sock} = :gen_tcp.connect(...)
    :gen_tcp.send(sock, encode(cmd))
    {:ok, data} = :gen_tcp.recv(sock, 0)
    :gen_tcp.close(sock)
    decode(data)
  end
end
```

```elixir
defmodule Redis do
  def command(cmd) do
    {:ok, sock} = :gen_tcp.connect(...)
    :gen_tcp.send(sock, encode(cmd))
    {:ok, data} = :gen_tcp.recv(sock, 0)
    :gen_tcp.close(sock)
    decode(data)
  end
end
```

opening

NEW

connections is

EXPENSIVE

keeping the socket in a

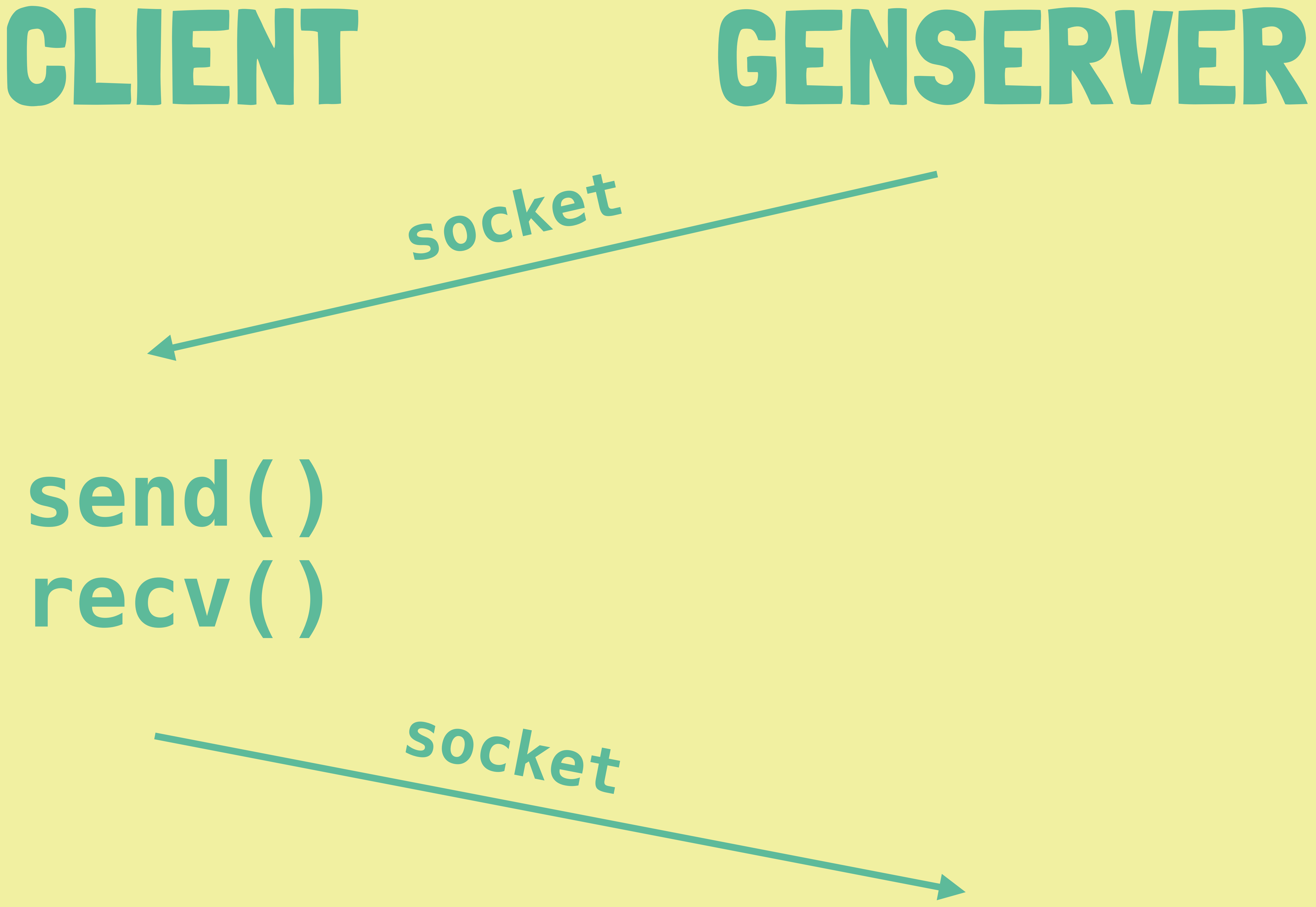
GENSERVER

TWO WAYS

blocking *non-blocking*

TWO WAYS

blocking



```
send()
recv()
```

```elixir
defmodule Redis do
  def command(conn, cmd) do
    sock = checkout(conn)
    # send and recv
    checkin(conn, sock)
  end
end
```

```elixir
defmodule Redis do
  def command(conn, cmd) do
    sock = checkout(conn)
    # send and recv
    checkin(conn, sock)
  end
end
```

```
{:error, :checked_out}
```

VS

```
:queue.in()
```

POOLING :|

TWO WAYS

non-blocking

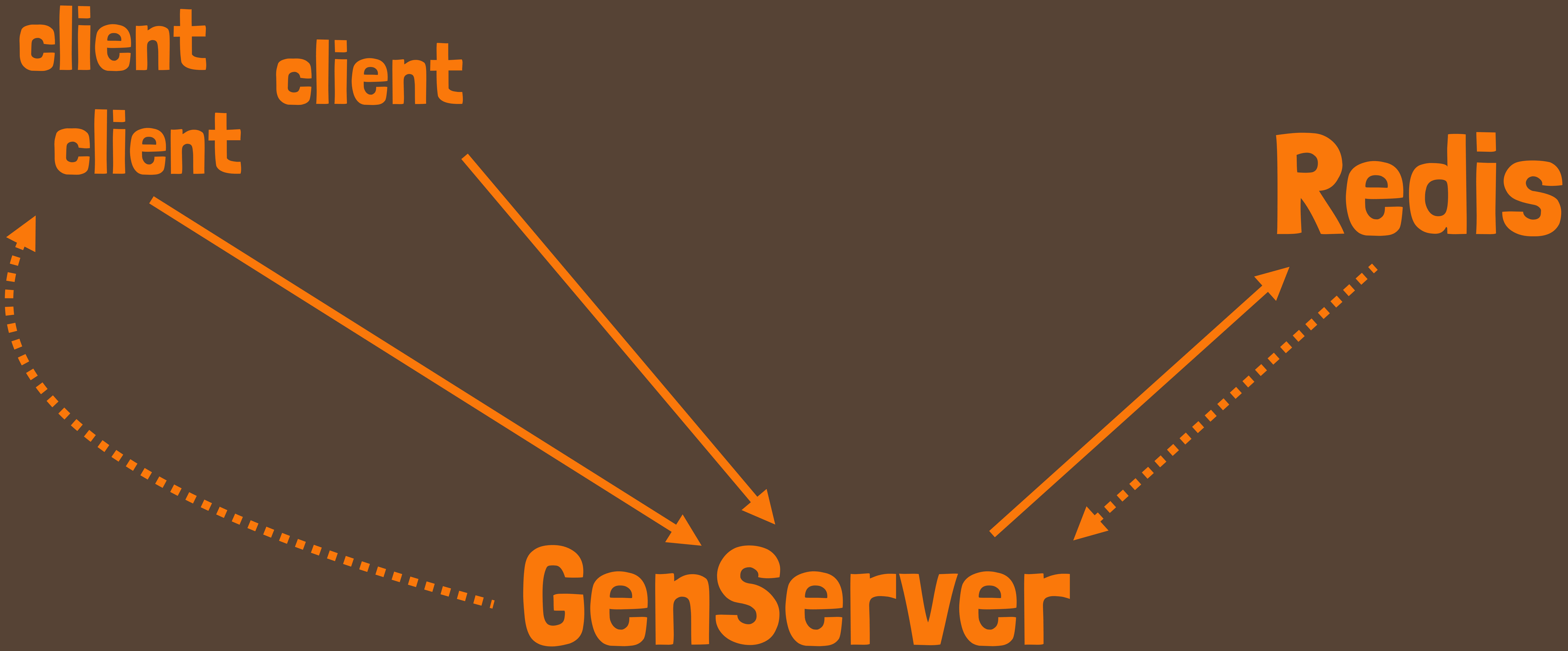
```
active: true
```

+

```
{:noreply, _}
```

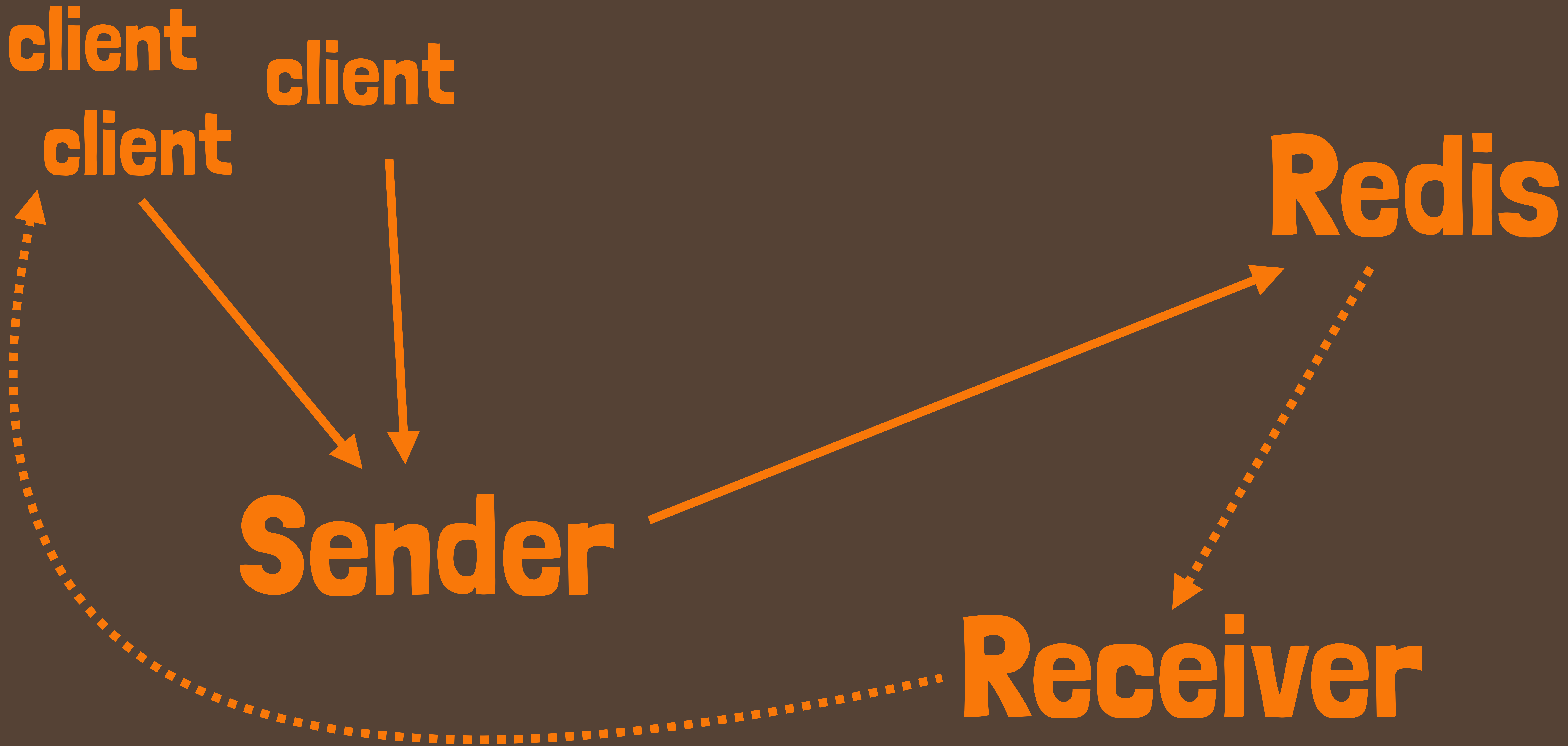
+

```
GenServer.reply/2
```


client
client
client
Redis
GenServer

tcp is

FULL DUPLEX


client
client
client
Sender
Redis
Receiver

TWO WAYS

blocking	*non-blocking*
copies less data +	– copies more data
enc/dec on client +	– enc/dec in server
needs pooling –	+ doesn't need pooling
doesn't use full duplex –	+ uses full duplex

RESIL
IENCY

$$\begin{tikzcd}\text{GenServer} \arrow[r, leftrightarrow] & \text{Redis}\end{tikzcd}$$

$$\begin{tikzcd}\text{GenServer} \arrow[rr, leftrightarrow, "/" marking] & & \text{Redis}\end{tikzcd}$$

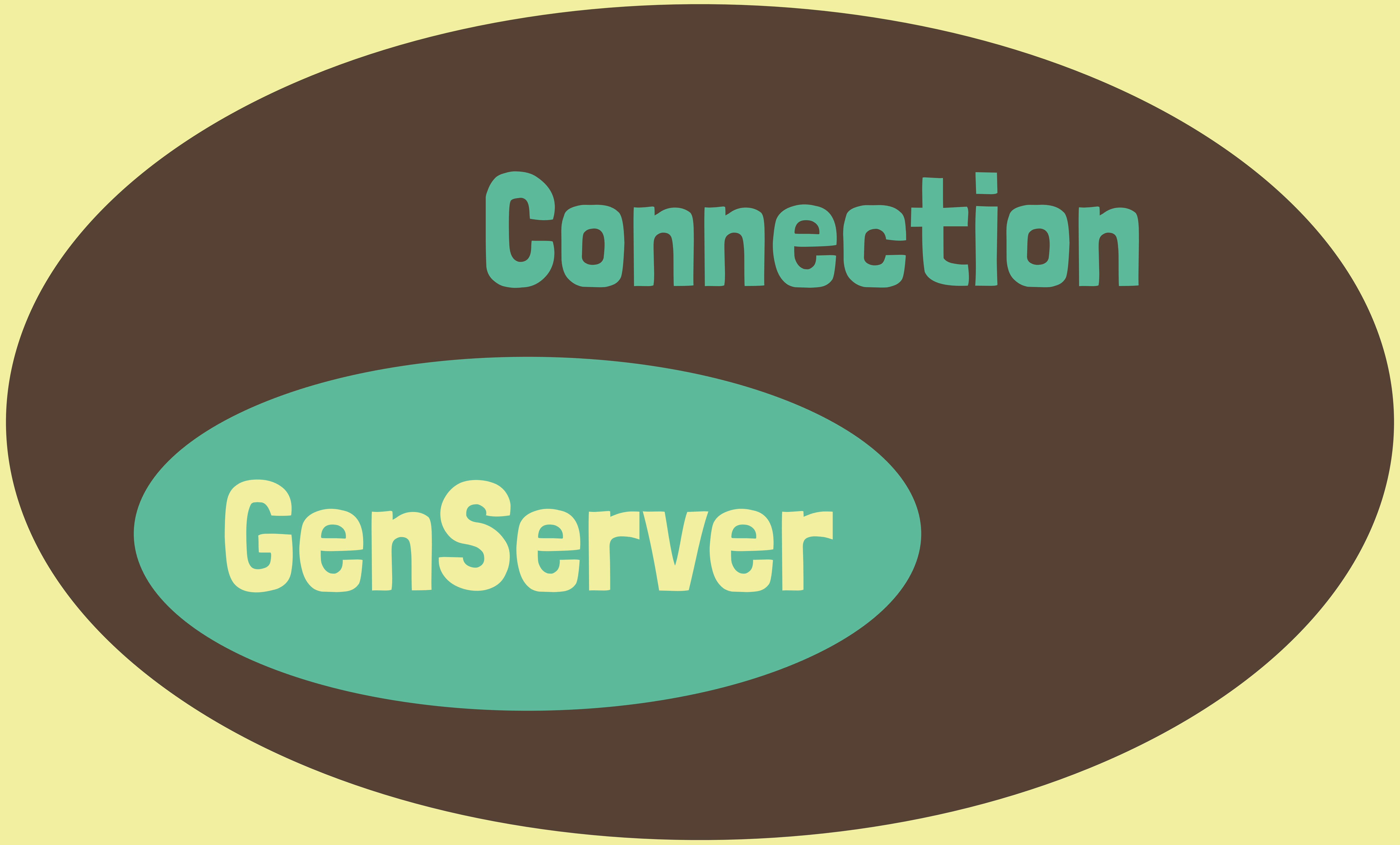
```
{:tcp_closed,
 reason}
```

↓

backoff + reconnect

hex.pm/packages/connection

gen_


Connection
GenServer

```
init/1
connect/2
disconnect/2
handle_call/3
handle_cast/2
handle_info/2
terminate/2
code_change/3
```

```
{:backoff,
 timeout,
 state}
```

```
{:disconnect,
 reason,
 state}
```

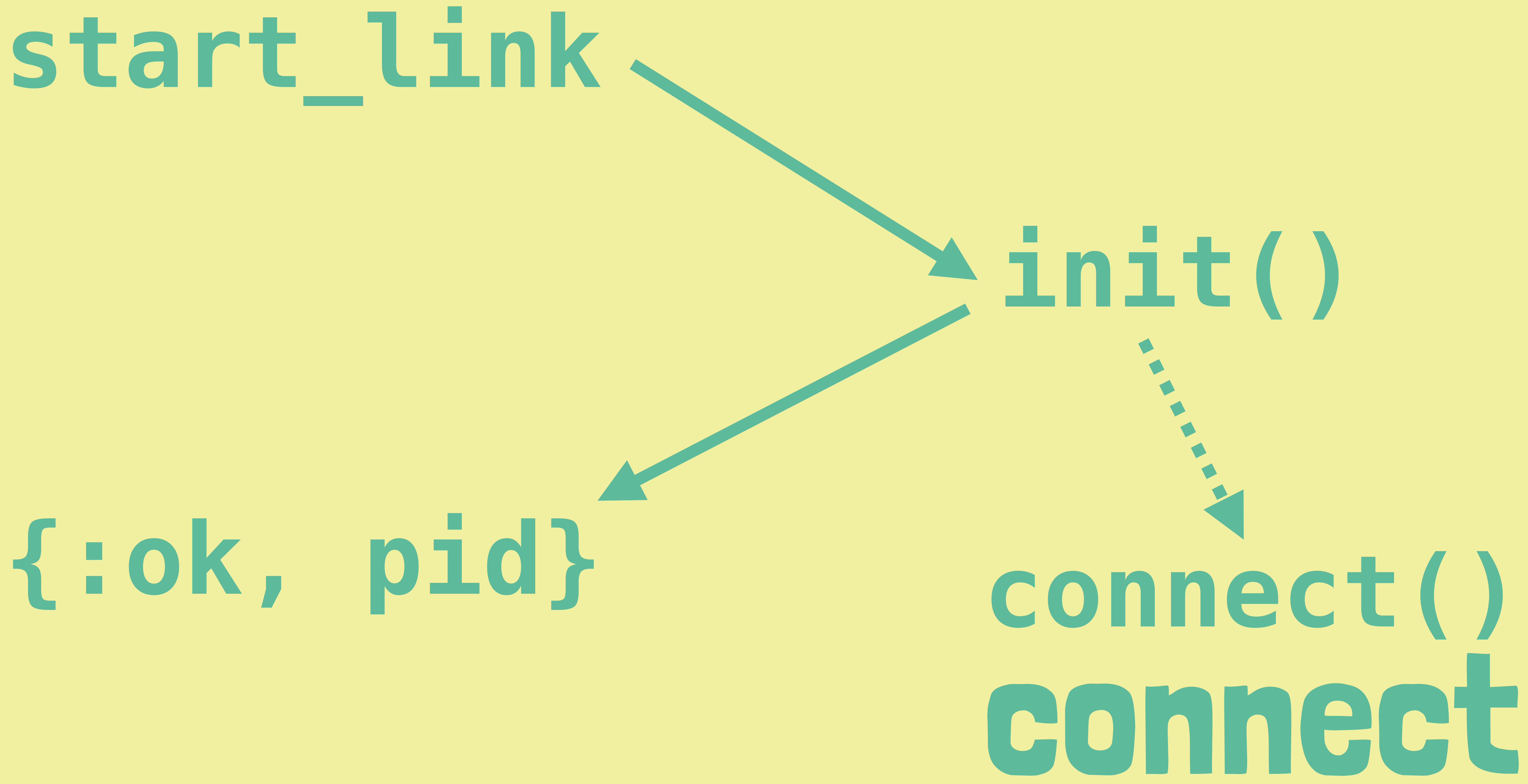
sync/async

CONNECT

SYNC

$$\begin{tikzcd}
\texttt{start\_link} \arrow[dr] & \\
& \begin{array}{l}\texttt{init()}\\ \textbf{connect}\end{array} \arrow[dl] \\
\texttt{\{:ok, pid\}} &
\end{tikzcd}$$

ASYNC



Fred Hebert

You now allow initializations with fewer guarantees: they went from "the connection is available" to "the connection manager is available".

CONNECTION

TALKING

CONCURRENCY

RESILIENCY

@whatyouhide