Thousand Island

The first in a 4 part yak shave[1]

@mattrudel - github.com/mtrudel - mat@geeky.net

[1] yak shave (noun): Any apparently useless activity which, by allowing you to overcome intermediate difficulties, allows you to solve a larger problem.

Thousand Island

A pure Elixir socket server, embodying OTP ideals

- Sits underneath an e.g. HTTP server
- Listens for client connections over TCP
- Hands them off to an upper protocol layer
- Provides send / receive / &c functionality
- Handles transport concerns (TLS, connection draining, etc)
- Does this efficiently and scalably

This is a talk about an

Air Conditioner


LENNOX


SET TEMP.
20°C
HEAT
FAN
AUTO
ON/OFF
SILENCE FP
MODE
TEMP
TIMER ON
FAN
TIMER OFF
SLEEP
SWING
DIRECT
TURBO
SELF CLEAN
LED
FOLLOW ME
LENNOX

What I want to build

What I want to build

How to build a custom HomeKit accessory?

- Homebridge

How to build a custom HomeKit accessory?

- ~~Homebridge~~
- HAP-Python

How to build a custom HomeKit accessory?

- ~~Homebridge~~
- ~~HAP-Python~~
- Numerous C libs

How to build a custom HomeKit accessory?

- ~~Homebridge~~
- ~~HAP-Python~~
- ~~Numerous C libs~~
- I've been meaning to play with Nerves anyway, so...

So… I guess I'm writing a HomeKit library?

This is a talk about an

~~Air Conditioner~~

HomeKit Library

HAP

HomeKit for Elixir & Nerves

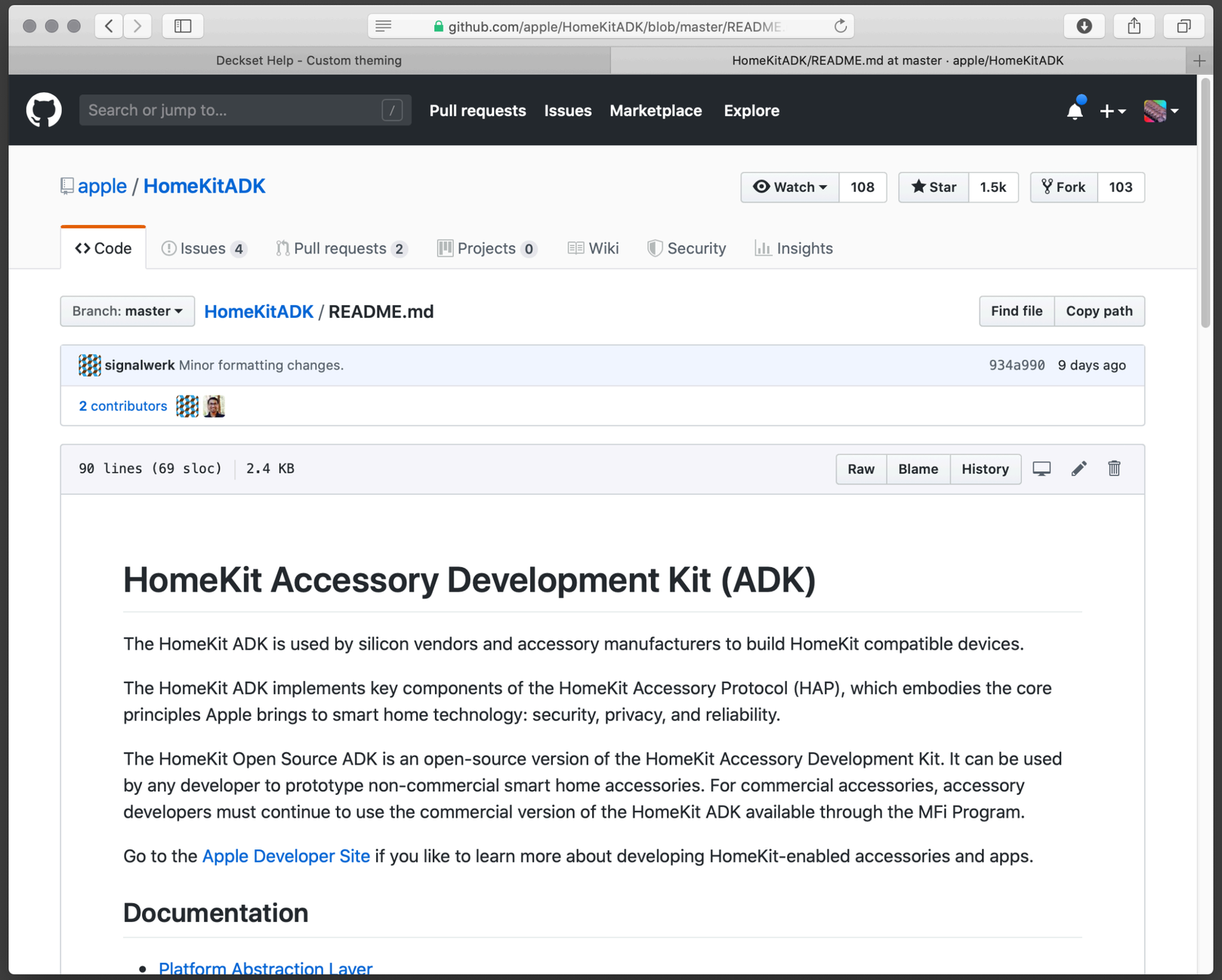
https://github.com/mtrudel/hap

HomeKit Accessory Protocol Specification

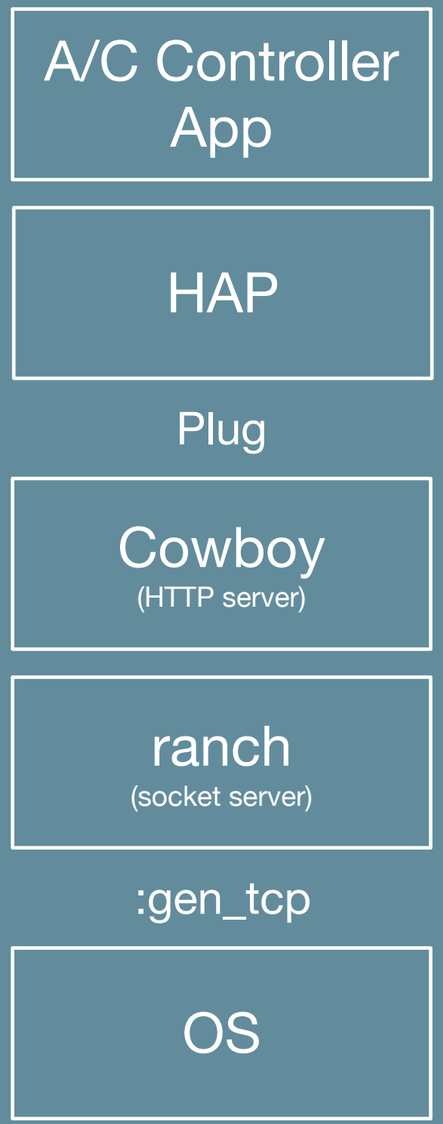
Non-Commercial Version

Release R2

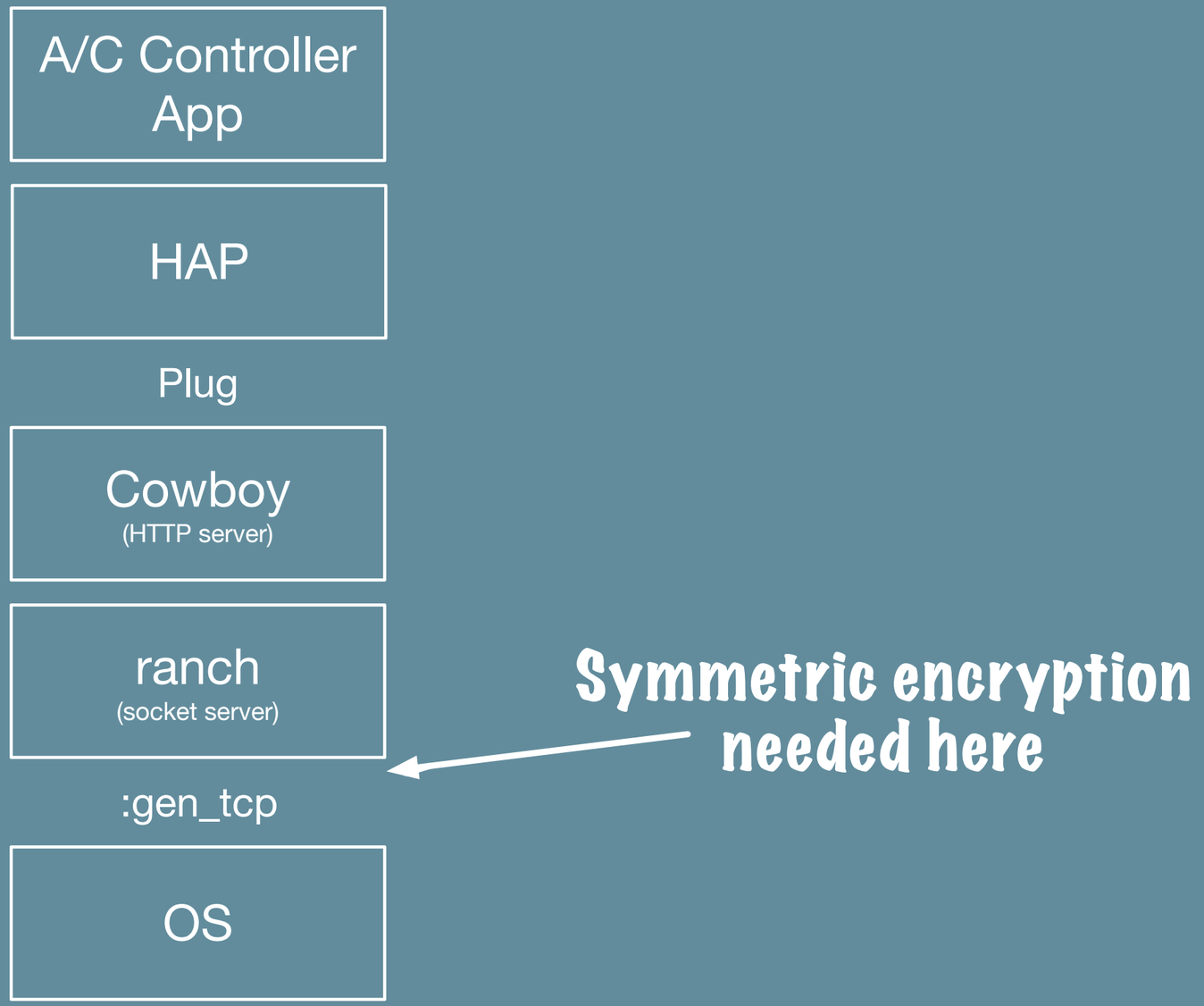
Developer


github.com/apple/HomeKitADK/blob/master/README.
Deckset Help - Custom theming
HomeKitADK/README.md at master · apple/HomeKitADK
Search or jump to...
Pull requests
Issues
Marketplace
Explore
apple / HomeKitADK
Watch 108
Star 1.5k
Fork 103
Code
Issues 4
Pull requests 2
Projects 0
Wiki
Security
Insights
Branch: master
HomeKitADK / README.md
Find file
Copy path
signalwerk Minor formatting changes.
934a990 9 days ago
2 contributors
90 lines (69 sloc) 2.4 KB
Raw
Blame
History
HomeKit Accessory Development Kit (ADK)
The HomeKit ADK is used by silicon vendors and accessory manufacturers to build HomeKit compatible devices.
The HomeKit ADK implements key components of the HomeKit Accessory Protocol (HAP), which embodies the core principles Apple brings to smart home technology: security, privacy, and reliability.
The HomeKit Open Source ADK is an open-source version of the HomeKit Accessory Development Kit. It can be used by any developer to prototype non-commercial smart home accessories. For commercial accessories, accessory developers must continue to use the commercial version of the HomeKit ADK available through the MFi Program.
Go to the Apple Developer Site if you like to learn more about developing HomeKit-enabled accessories and apps.
Documentation
Platform Abstraction Layer

HomeKit devices run an HTTP server



HomeKit devices run a non-standard HTTP server



Welp, I guess I'm writing a web server

This is a talk about an

~~Air Conditioner~~

~~HomeKit Library~~

Web Server

Bandit

A Plug-First Web Server

https://github.com/mtrudel/bandit

What does a Web Server do, exactly?

1. Listen for connections
2. Handle each connection
    1. Parse an HTTP request into a `Plug.Conn` struct
    2. Pass this struct to a `Plug` implementation such as Phoenix
    3. Provide backing support for `Plug.Conn.send_resp`, &c.

Problem naturally splits into two parts

1. Listen for connections (socket server, e.g: Thousand Island or ranch)
2. Handle each connection (web server, e.g: Bandit or Cowboy)

This is a talk about an

~~Air Conditioner~~

~~HomeKit Library~~

~~Web Server~~

Socket Server

Thousand Island

A pure Elixir socket server, embodying OTP ideals

https://github.com/mtrudel/thousand_island

What is a socket server? (repeat)

- Sits underneath an e.g. HTTP server
- Listens for client connections over TCP
- Hands them off to an upper protocol layer
- Provides send / receive / &c functionality
- Handles transport concerns (TLS, connection draining, etc)
- Does this efficiently and scalably

Version 0: Hello, World

```
{:ok, listen_socket} = :gen_tcp.listen(4000, [active: false])    # Listen (this binds the port)
{:ok, connection_socket} = :gen_tcp.accept(listen_socket)        # Accept (this waits for a connection)
:gen_tcp.send(connection_socket, "Hello, World")                 # Interact with the client
:gen_tcp.close(connection_socket)                                # Close the connection
```

Problem: Only works once

```
{:ok, listen_socket} = :gen_tcp.listen(4000, [active: false])
{:ok, connection_socket} = :gen_tcp.accept(listen_socket)
:gen_tcp.send(connection_socket, "Hello, World")
:gen_tcp.close(connection_socket)
```

```
# Listen (this binds the port)
# Accept (this waits for a connection)
# Interact with the client
# Close the connection
```

Version 1: Listen on repeat

```
defmodule SocketServer do
  def run do
    {:ok, listen_socket} = :gen_tcp.listen(4000, [active: false])  # Listen (this binds the port)
    accept(listen_socket)
  end

  defp accept(listen_socket) do
    {:ok, connection_socket} = :gen_tcp.accept(listen_socket)      # Accept (this waits for a connection)
    :gen_tcp.send(connection_socket, "Hello, World")               # Interact with the client
    :gen_tcp.close(connection_socket)                              # Close the connection
    accept(listen_socket)                                          # Listen for the next connection
  end
end
```

Version 2: Make it generic

```
defmodule Handler do
  def handle(socket) do
    :gen_tcp.send(socket, "Hello, World") # Substitute HTTP server implementation here
  end
end

defmodule SocketServer do
  def run(handler) do
    {:ok, listen_socket} = :gen_tcp.listen(4000, [active: false])
    accept(listen_socket, handler)
  end

  def accept(listen_socket, handler) do
    {:ok, connection_socket} = :gen_tcp.accept(listen_socket)
    handler.handle(connection_socket)
    :gen_tcp.close(connection_socket)
    accept(listen_socket, handler)
  end
end
```

Problem: Only one connection at a time

```
defmodule Handler do
  def handle(socket) do
    :gen_tcp.send(socket, "Hello, World") # Substitute HTTP server implementation here
  end
end

defmodule SocketServer do
  def run(handler) do
    {:ok, listen_socket} = :gen_tcp.listen(4000, [active: false])
    accept(listen_socket, handler)
  end

  def accept(listen_socket, handler) do
    {:ok, connection_socket} = :gen_tcp.accept(listen_socket)
    handler.handle(connection_socket)
    :gen_tcp.close(connection_socket)
    accept(listen_socket, handler)
  end
end
```

Version 3: One process per connection (simplified)

```
defmodule Handler do
  def handle(socket) do
    :gen_tcp.send(socket, "Hello, World") # Substitute HTTP server implementation here
  end
end

defmodule SocketServer do
  def run(handler) do
    {:ok, listen_socket} = :gen_tcp.listen(4000, [active: false])
    accept(listen_socket, handler)
  end

  def accept(listen_socket, handler) do
    {:ok, connection_socket} = :gen_tcp.accept(listen_socket)
    Task.start_link(fn ->
      handler.handle(connection_socket)
      :gen_tcp.close(connection_socket)
    end)
    accept(listen_socket, handler)
  end
end
```

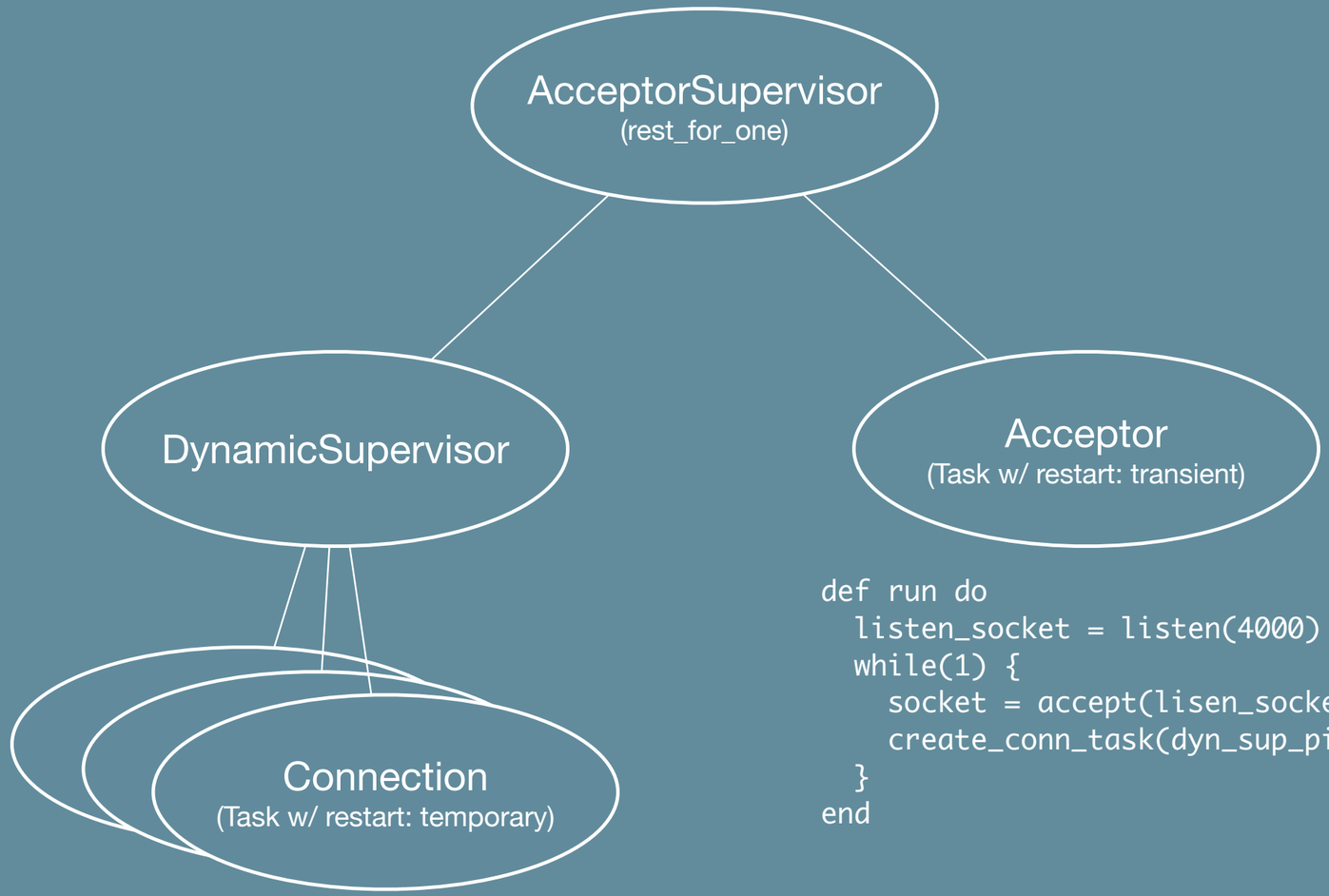
Problem: No Supervision

```
defmodule Handler do
  def handle(socket) do
    :gen_tcp.send(socket, "Hello, World") # Substitute HTTP server implementation here
  end
end

defmodule SocketServer do
  def run(handler) do
    {:ok, listen_socket} = :gen_tcp.listen(4000, [active: false])
    accept(listen_socket, handler)
  end

  def accept(listen_socket, handler) do
    {:ok, connection_socket} = :gen_tcp.accept(listen_socket)
    Task.start_link(fn ->
      handler.handle(connection_socket)
      :gen_tcp.close(connection_socket)
    end)
    accept(listen_socket, handler)
  end
end
```

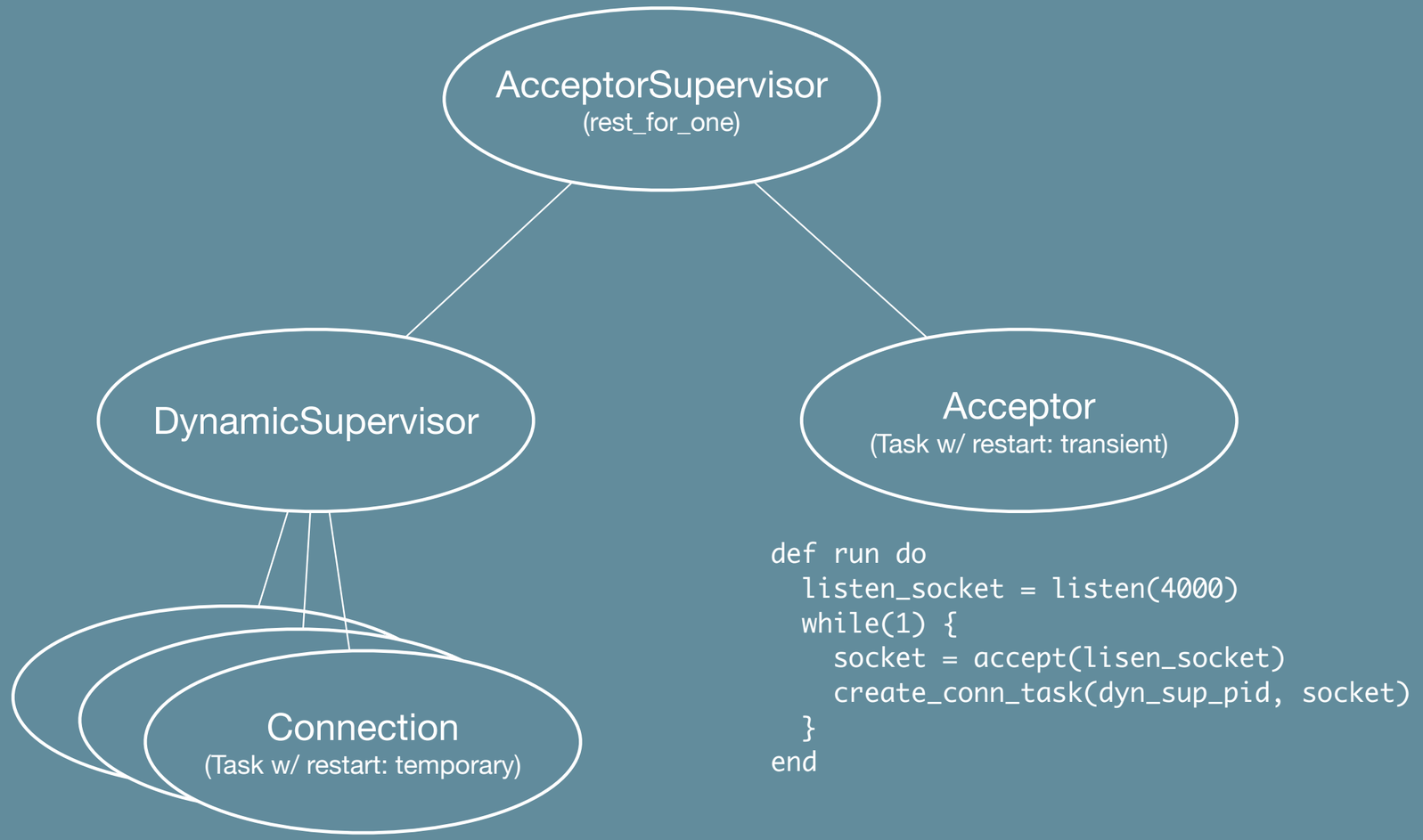
Version 4: Task Supervision




```
def run do
  listen_socket = listen(4000)
  while(1) {
    socket = accept(lisen_socket)
    create_conn_task(dyn_sup_pid, socket)
  }
end
```

```
def run(socket) do
  handler.handle(socket)
  close(socket)
end
```

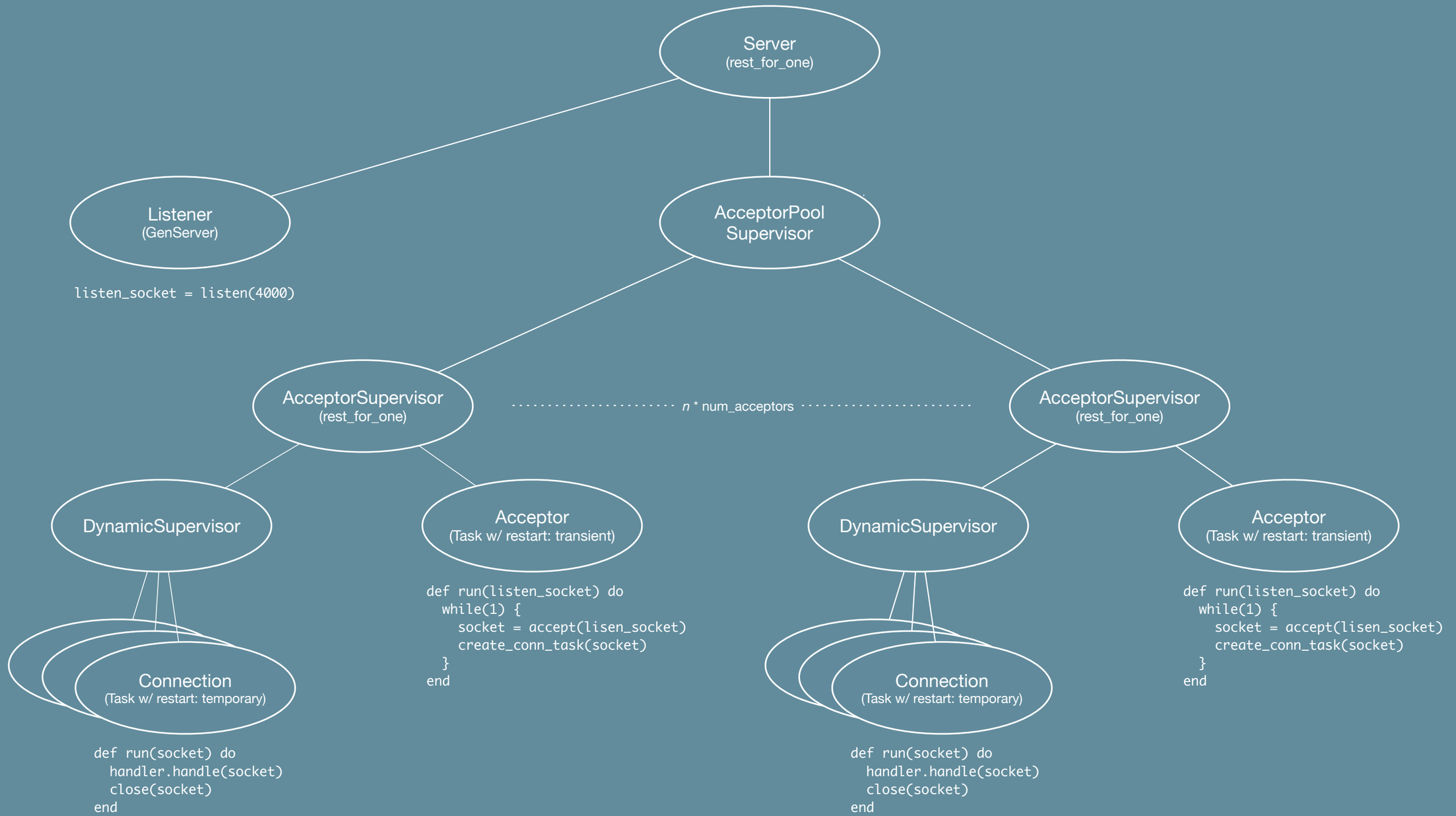
Problem: Only one process is listening for connections



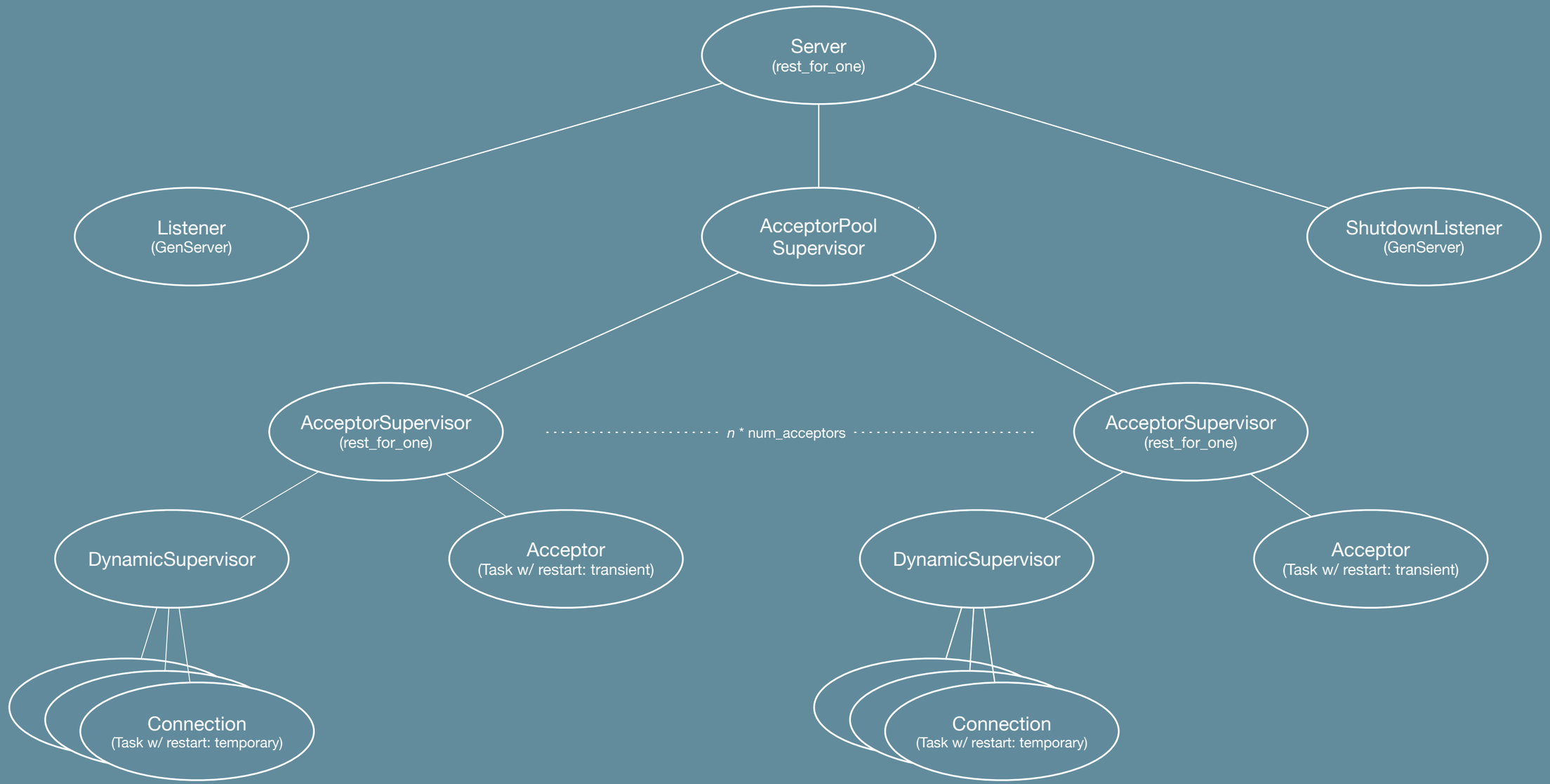

```
def run do
  listen_socket = listen(4000)
  while(1) {
    socket = accept(lisen_socket)
    create_conn_task(dyn_sup_pid, socket)
  }
end
```

```
def run(socket) do
  handler.handle(socket)
  close(socket)
end
```

Version 5: Multiple acceptors



This is where Thousand Island is today



Other Features

- Fast (performance basically equal to ranch)
- Simple (~1100 LoC, w/ exhaustive docs)
- Supports TCP, TLS & Unix Domain sockets
- Loads of config options
- Connection draining
- Fully wired for telemetry

Status

(Unwinding the Yak Shave)

Thousand Island

- Feature Complete ✅
- No known issues, but test coverage is 👎
- Currently `v0.1.x` on `hex.pm`
- `v1.0` is imminent

https://github.com/mtrudel/thousand_island

Bandit (HTTP Server)

- HTTP/1.1 almost complete
- No HTTP/2.0 or Websocket support yet
- Drop-In Cowboy replacement today (for simple apps)
- **Lots** of fun problems to solve; help welcome!
- Hopefully present in Q2 2020
- Long Term Goal: Dethrone Cowboy (?!)

https://github.com/mtrudel/bandit

HAP (HomeKit Library)

- Discovery, `Pair-Setup` & `Pair-Verify` done
- Roughly ~20% complete overall
- Hard stuff mostly done
- Hopefully present Q3 2020

https://github.com/mtrudel/hap

A/C Controller App

- Nerves-hosted HomeKit Accessory
- Non-existent at this point
- PoC by Q4 2020?

http://github.com/mtrudel/thousand_island

http://github.com/mtrudel/bandit

http://github.com/mtrudel/hap