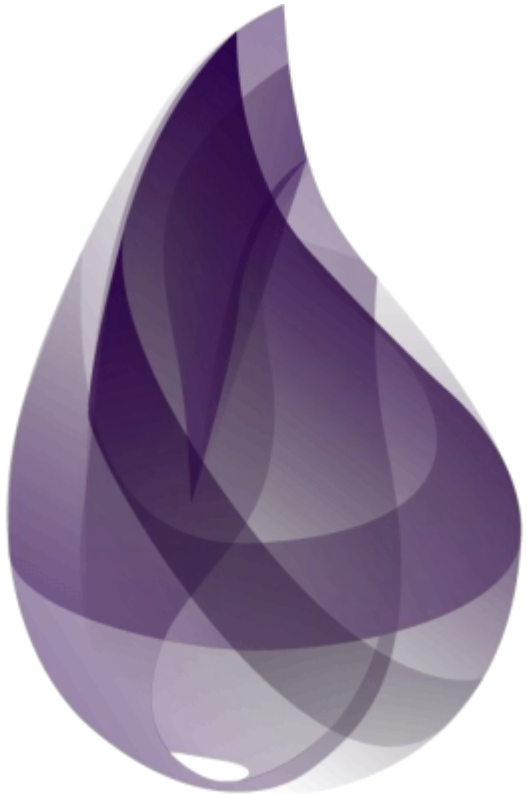
Concurrency the easy way

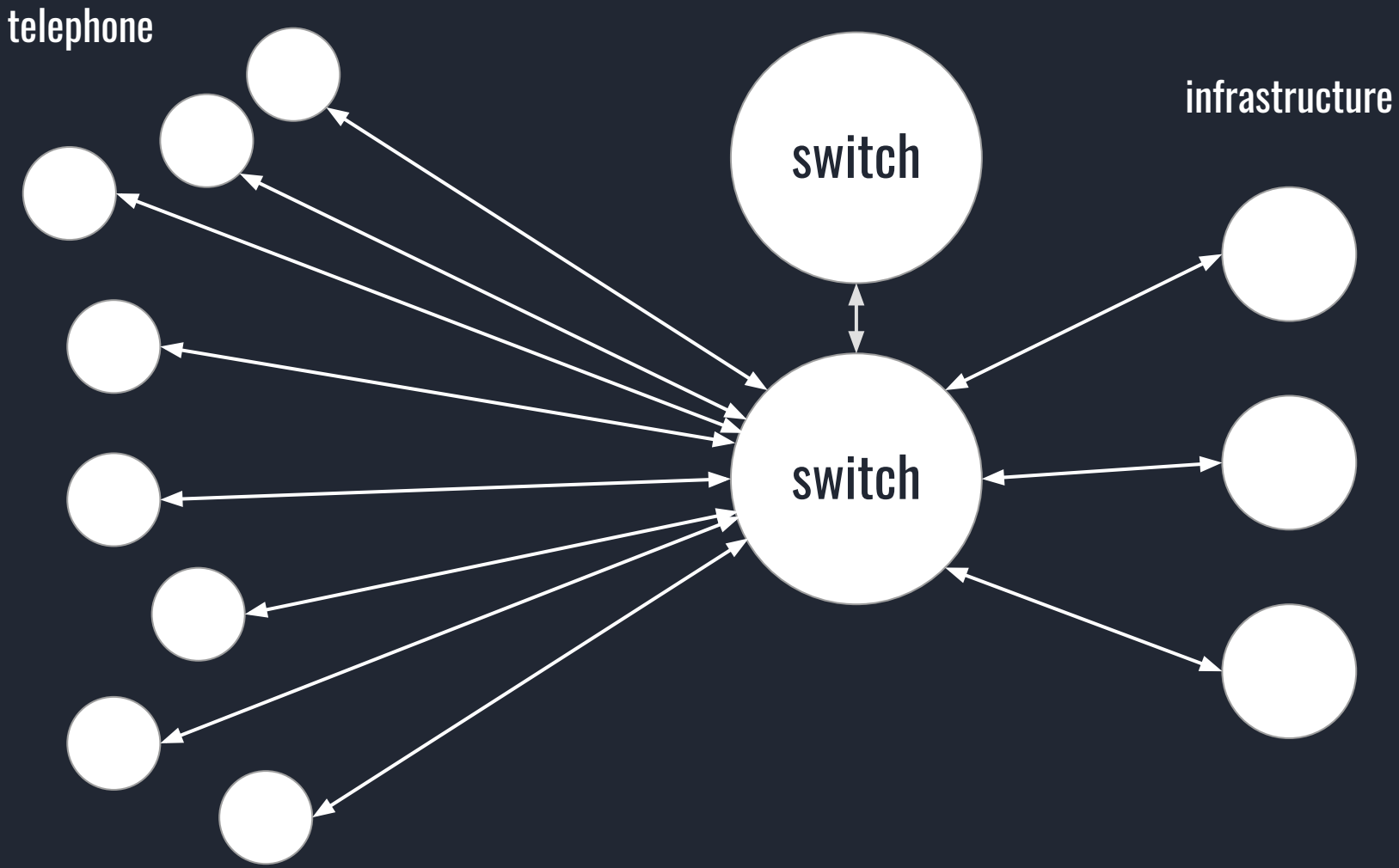
Elixir and the Erlang VM

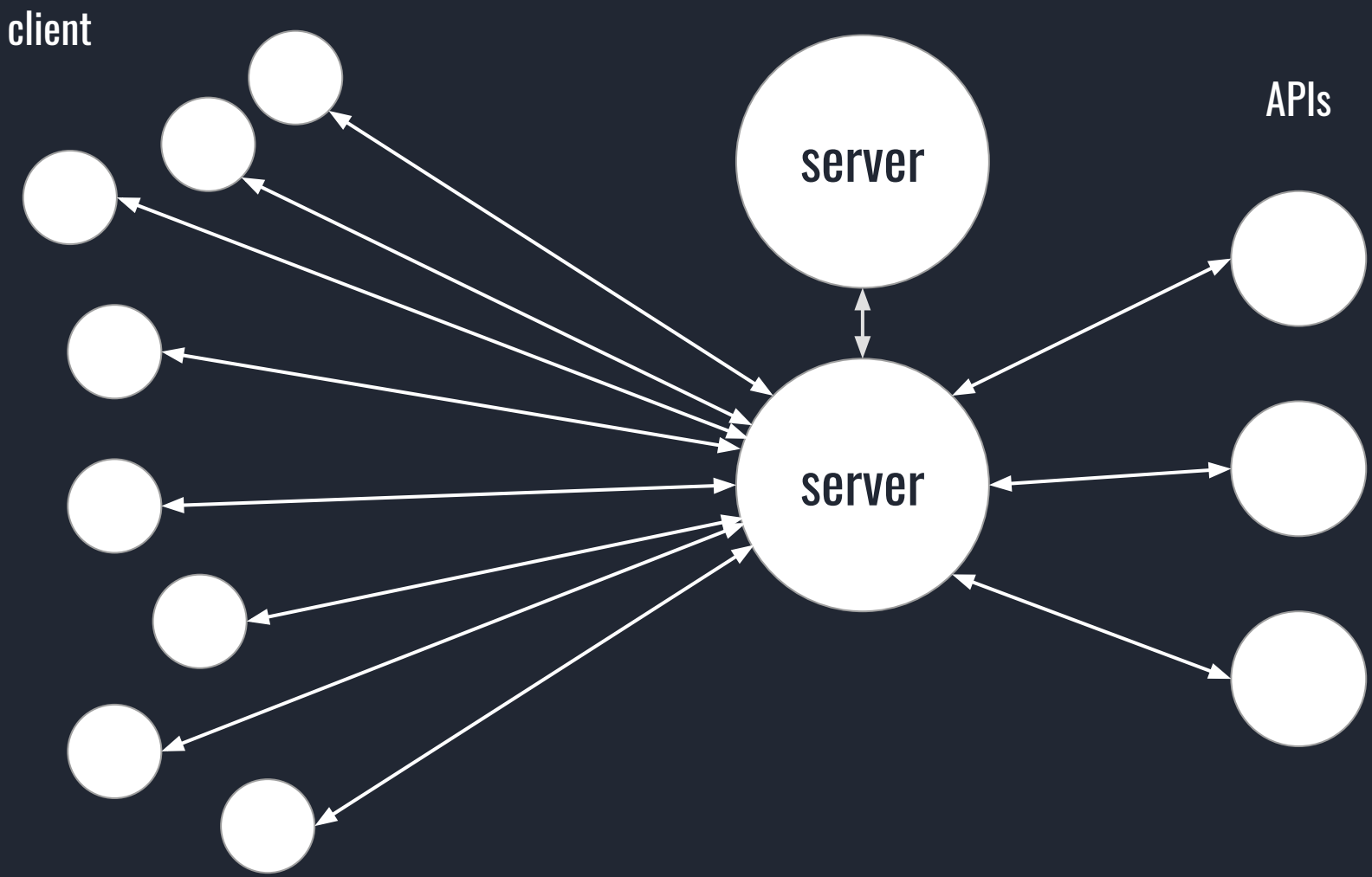


MODERN *beautiful*

Erlang VM

a.k.a. BEAM


telephone
switch
infrastructure
switch


client
server
APIs
server

WhatsApp: 2 million users on a single box

https://blog.whatsapp.com/196/1-million-is-so-2011

“we needed roughly 150 servers (...) we're now able to power those same functions on five servers (...) We could probably get away with it on two”

https://www.techworld.com/apps-wearables/how-elixir-helped-bleacher-report-handle-8x-more-traffic-3653957/

“90% of all internet traffic goes through Erlang controlled nodes”

https://twitter.com/guieevc/status/1002494428748140544

Concurrency building blocks

Concurrency fundamentals

process	isolated unit of concurrent execution
pid	unique process identifier
message	any value sent between processes
send and receive	mechanism for sending and receiving messages
monitor	unidirectional exit notification
link	terminate a group of processes
:trap_exit	handle own or others' termination gracefully
:kill	forceful process shutdown

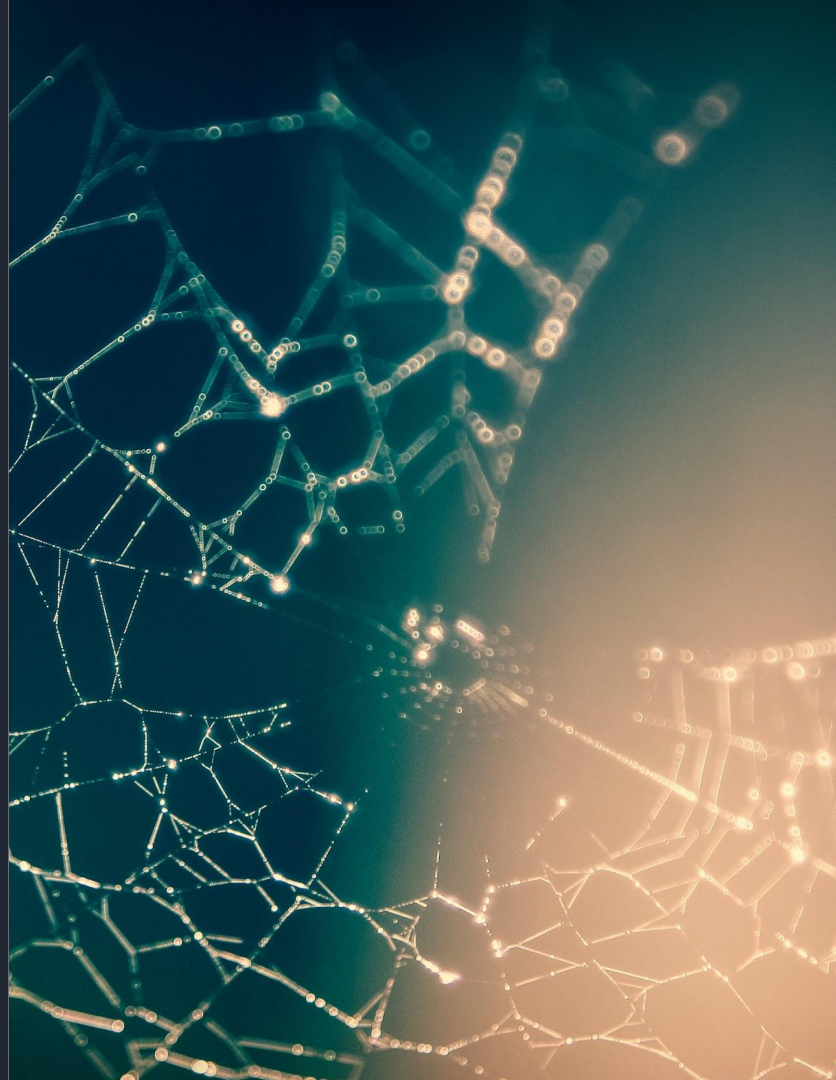
Fundamental Erlang abstractions: OTP behaviours

GenServer
generic server
handles: lifecycle, message exchange, receive loop
fill-in the blanks: state and the messages

Supervisor
supervises other processes
declare restart strategies when things go wrong
supervisors can supervise other supervisors (hierarchies!)

Application
a group of processes that does some work
“component”, “library”
you will run many of those in your application

Building a web crawler

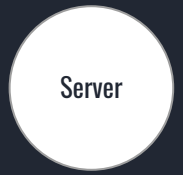


Requirements

- download a web page and extract links
- download pages concurrently
- download a page only once
- isolate errors
- retry failed pages
- rate-limit web requests

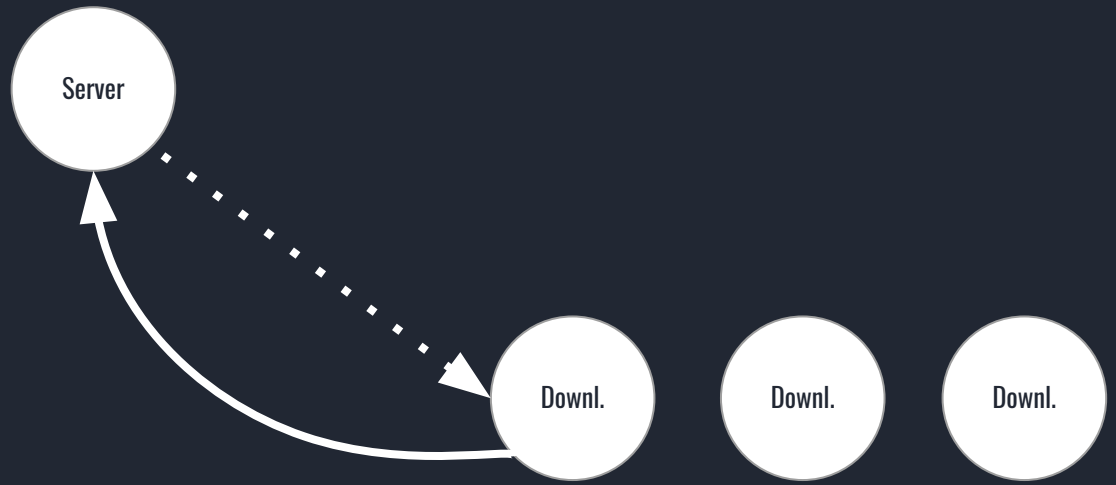
Process design

- Server
  - drives the crawling
  - caches pages
  - decides the retry on failure



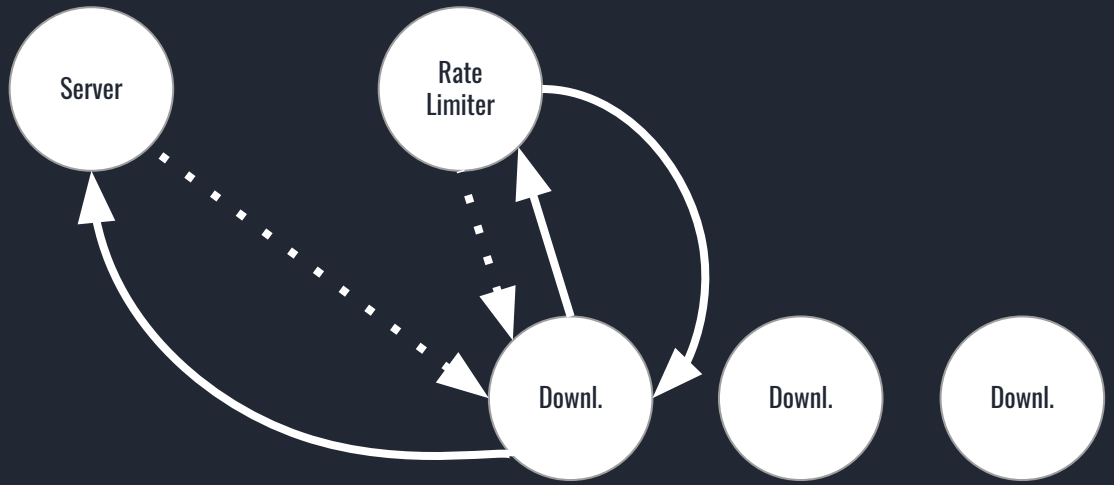
Process design (cont.)

- Server
  - drives the crawling
  - caches pages
  - decides the retry on failure
  - spawns and monitors Downloaders
- Downloader
  - downloads a single page
  - succeeds or crashes
  - reports back to the Server
  - restart: :temporary



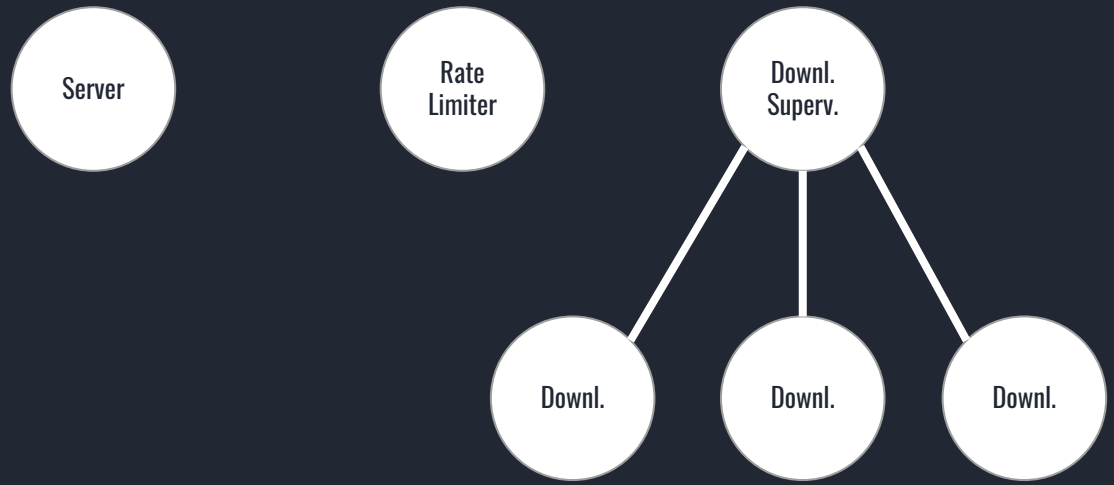
Process design (cont.)

- Server
  - drives the crawling
  - caches pages
  - decides the retry on failure
  - spawns and monitors Downloaders
- Downloader
  - downloads a single page
  - succeeds or crashes
  - reports back to the Server
  - restart: :temporary
- RateLimiter
  - blocking pool
  - monitors Downloaders



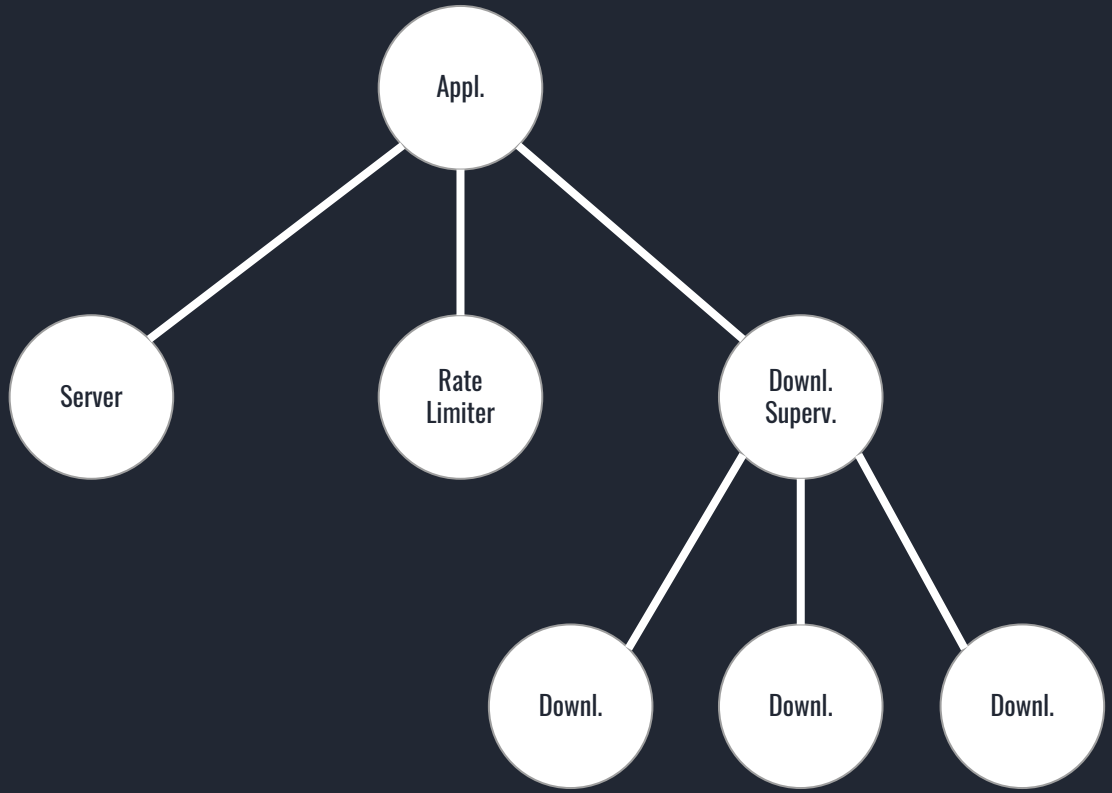
Process design (cont.)

- Downloader.Supervisor
  - proper termination of Downloaders



Process design (cont.)

- Downloader.Supervisor
  - proper termination of Downloaders
- Application
  - proper termination of children
  - restart strategy: :rest_for_one



```
$ brew install elixir
```

Q&A