INFINUM

Recurring events on Elixir *(the naive approach)*

VEDRAN HRNČIĆ

Static schedules vs Dynamic schedules

- weekly status email
- daily data cleanup
- dispatch notifications

- medication plans
- reminders

system service(cron)

Heroku scheduler add-on

Sidekiq-scheduler

Quantum

??

Why Elixir?

- Concurrency – Erlang can run thousands, even millions of concurrent processes
- Fault tolerance – processes are completely isolated (share no memory)
- Error isolation – supervision trees provide fine control over error propagation
- Scalability – VM takes advantage of multiple CPU cores
- Distribution – builtin primitives for running service on multiple machines

GenServer

- process is purely functional and stateless computation unit
- abstraction for building long-running stateful server processes
- compatible with OTP supervision interface

Long running process
with vanilla Elixir

```elixir
defmodule Counter do
  def start_link() do
    {:ok, spawn(fn() -> loop(0) end)}
  end

  def inc(pid, offset) do
    send(pid, {{:inc, offset}, self()})

    receive do
      value -> value
    end
  end

  def reset(pid) do
    send(pid, {:reset, self()})

    :ok
  end

  defp loop(value) do
    receive do
      {{:inc, offset}, caller} ->
        next_value = value + offset
        send(caller, next_value)
        loop(next_value)
      {:reset, _caller} ->
        loop(0)
    end
  end
end
```

Long running process
with GenServer

```elixir
defmodule Counter do
  use GenServer

  def start_link() do
    GenServer.start_link(Counter, :ok)
  end

  def inc(pid, offset) do
    GenServer.call(pid, {:inc, offset})
  end

  def reset(pid) do
    GenServer.cast(pid, :reset)
  end

  def init(_) do
    {:ok, 0}
  end

  def handle_call({:inc, offset}, _from, value) do
    next_value = value + offset
    {:reply, next_value, next_value}
  end

  def handle_cast(:reset, _value) do
    {:noreply, 0}
  end
end
```

Supervisor

- process which supervises other processes (children)
- define how processes are started and how they are shutdown (via strategies)
- provide fault-tolerance (define boundaries for error impact)

Registry

- in essence key–value process storage
- commonly used to name and locate processes (associate PID with process name)
- when process dies, associated values are automatically deleted

System overview

Every twelve hours from
21.10.2019 9:30

Every eight hours from
21.10.2019 10:00

Erlang VM

callback(ts, ctx)





Translating schedule configuration

Every Monday at 9:00
from 21.10.2019

~N[2019-10-21 9:00]

~N[2019-10-28 9:00]

~N[2019-11-04 9:00]

~N[2019-11-11 9:00]

Cocktail

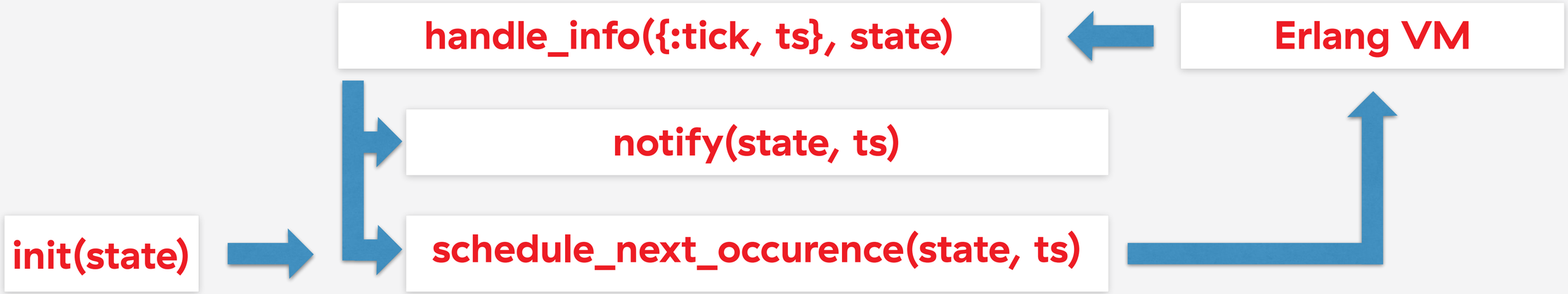
Schedule process behaviour

Stream<DateTime>

unique identifier

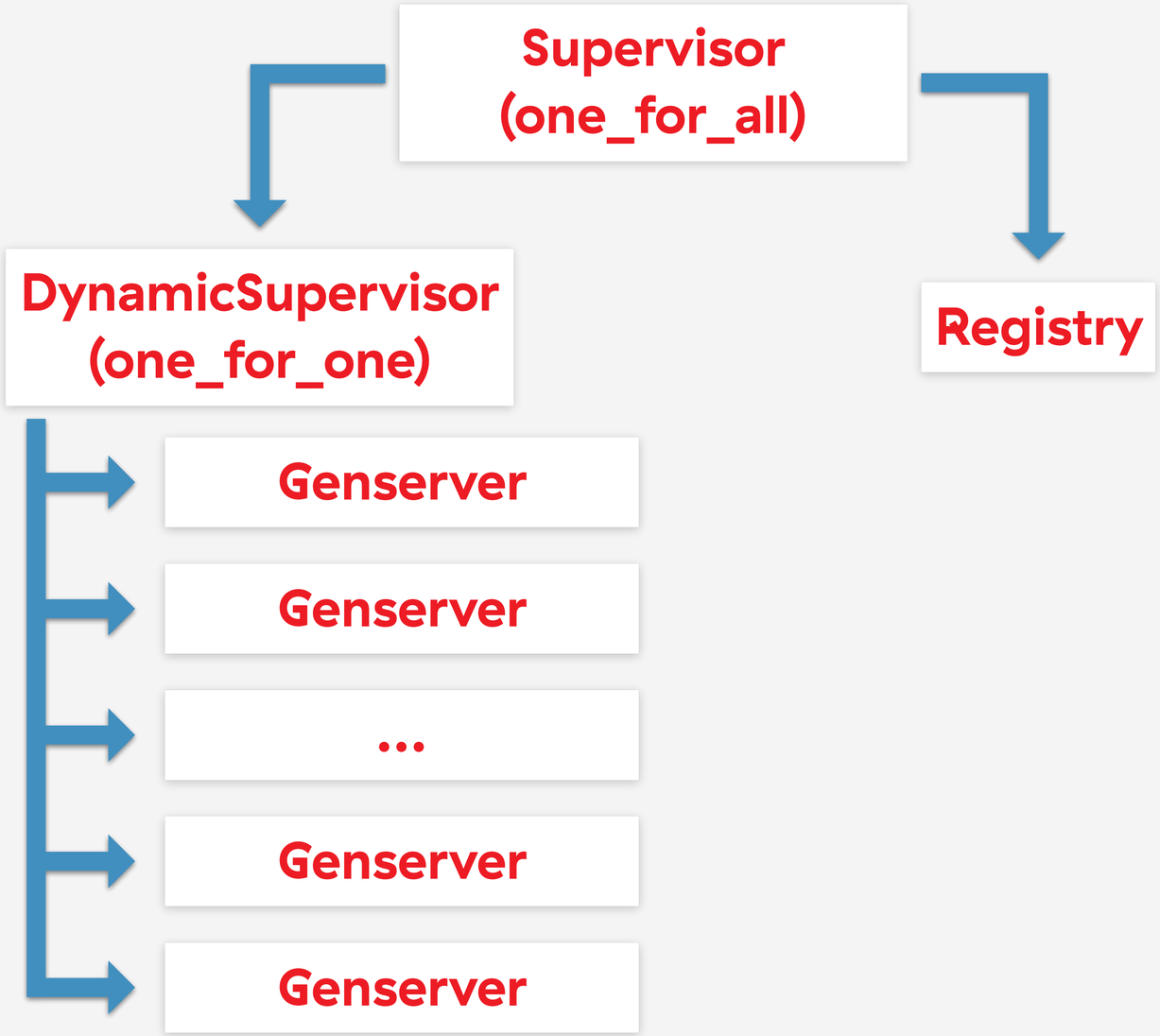
callback

Schedule GenServer

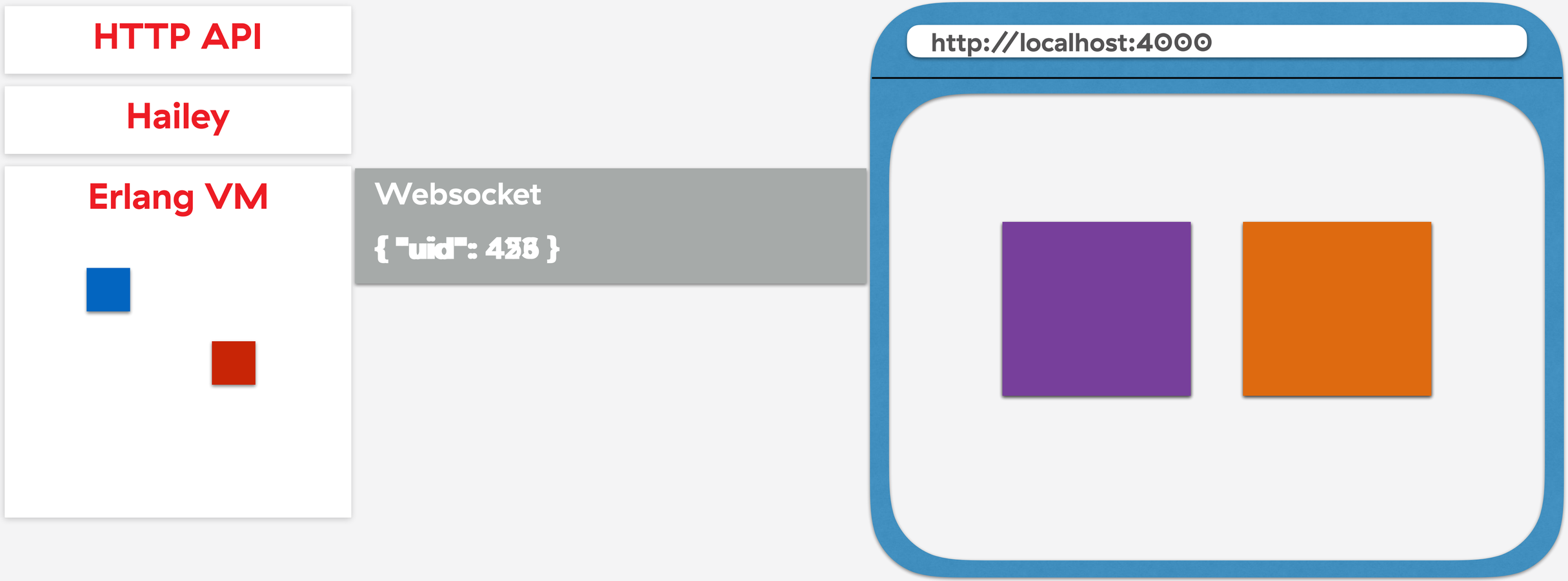



```
defp schedule_next_occurence(state, last_occurrence_at) do
  ts = next_occurence(state.occurrences, DateTime.utc_now(), last_occurrence_at)
  timeout = max(DateTime.diff(ts, DateTime.utc_now(), :millisecond), 1)

  Process.send_after(self(), {:tick, ts}, timeout)

  {:noreply, state}
end
```

OTP system architecture



Demo architecture



INFINUM

Questions?

VEDRAN.HRNCIC@INFINUM.COM

Visit infinum.co or find us on social networks:

infinum.co infinumco infinumco infinum