Orchestrating Producers and Consumers like an Octopus

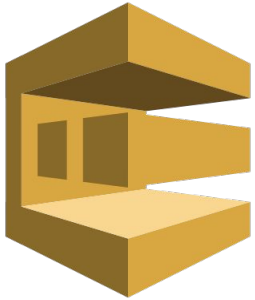
Jusabe Guedes
@dojusa

be simple

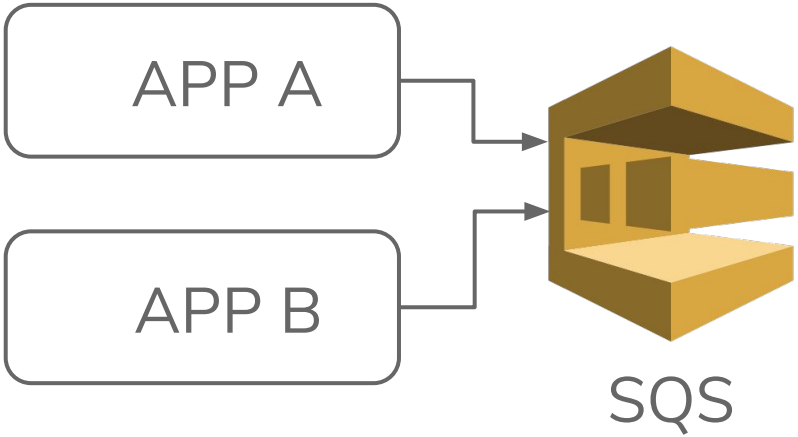
be simple

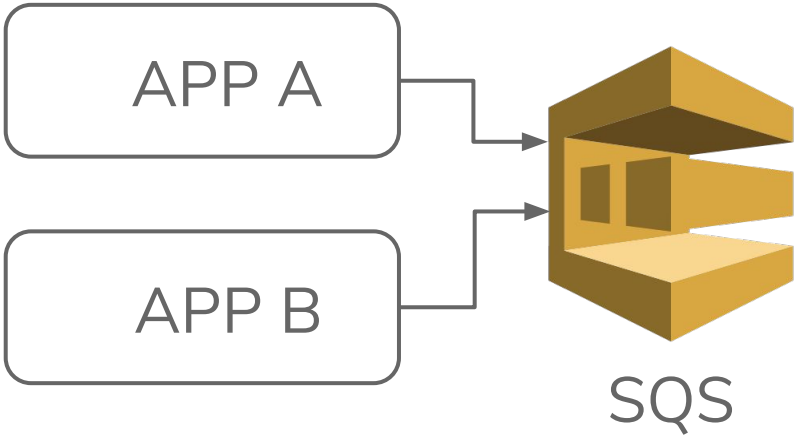
show a way, ~~not the way~~ of doing it

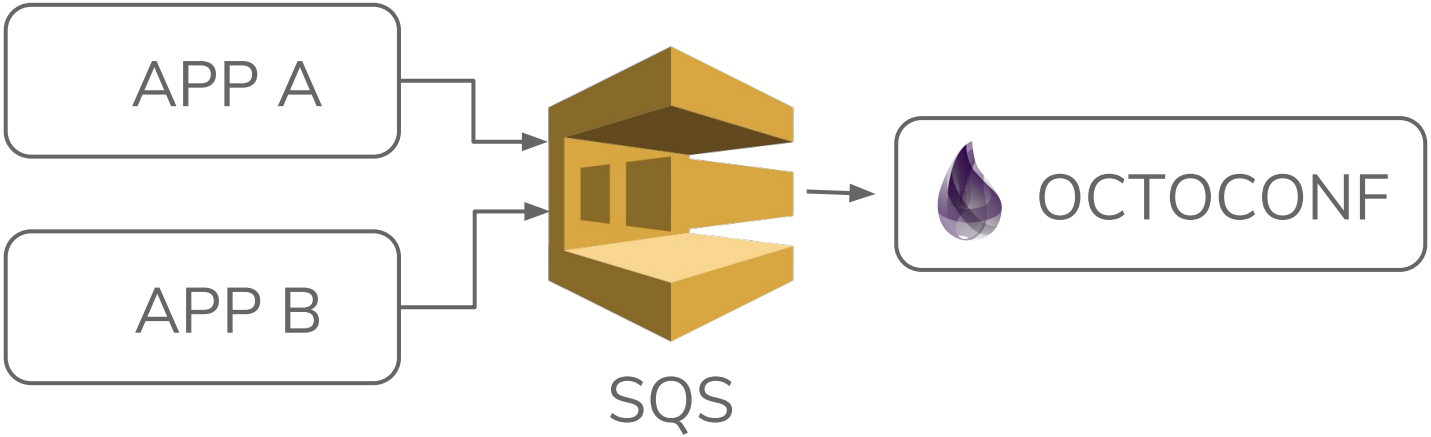
what is the problem?



SQS


APP A
APP B
SQS


APP A
APP B
SQS

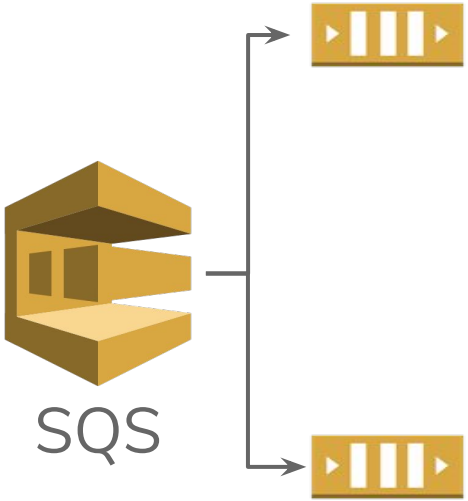

APP A
APP B
SQS
OCTOCONF

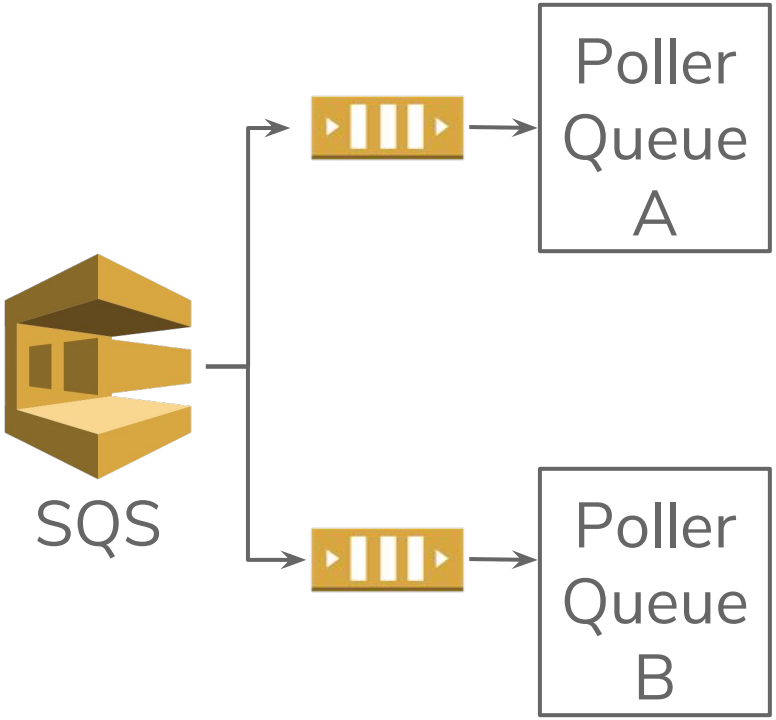

APP A
APP B
SQS
OCTOCONF
THIRD PARTY APP
THIRD PARTY APP
THIRD PARTY APP

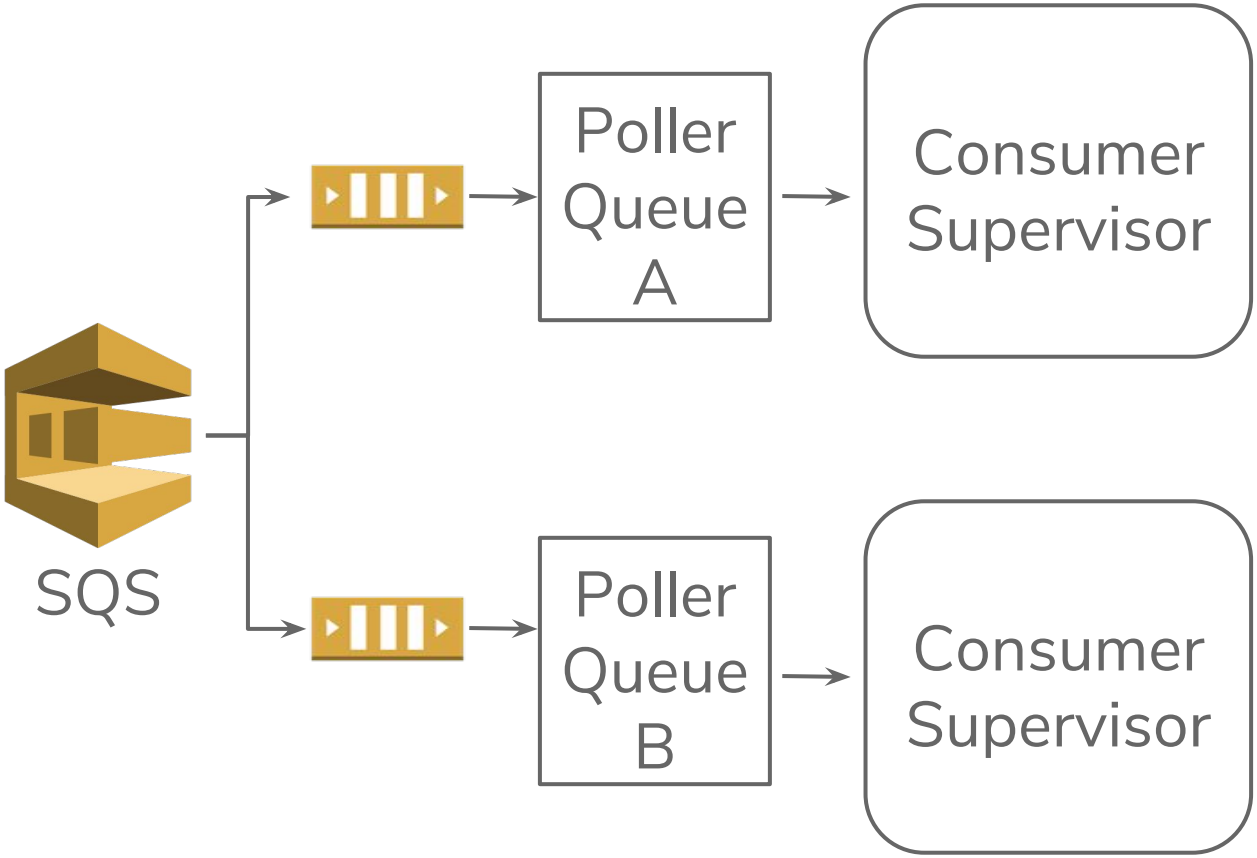
STEPS

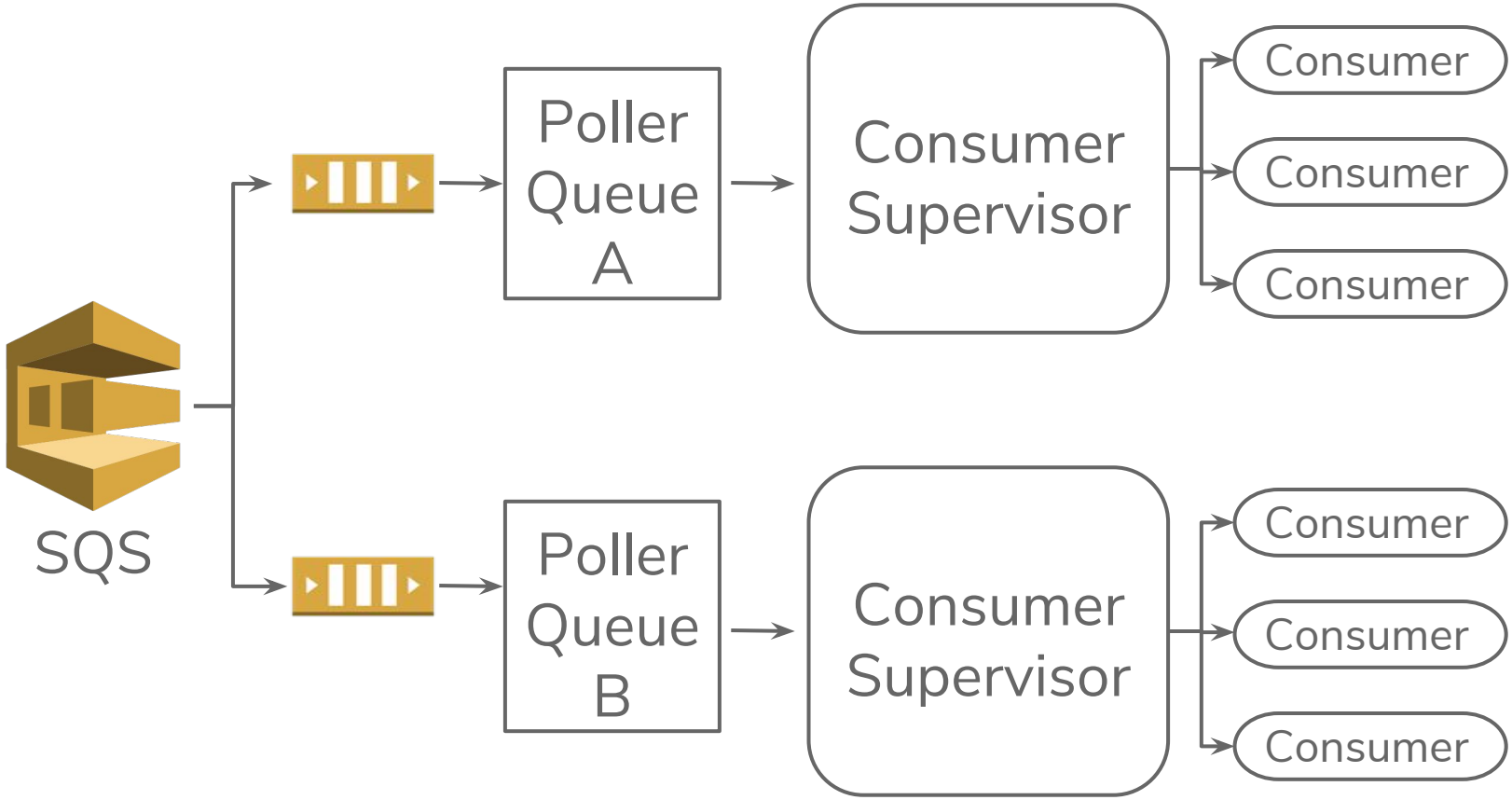
- A queue needs to be consumed
- Each element must pass through a pipeline
- Go to a different bucket
- Must work independently
- Flushed whenever is possible

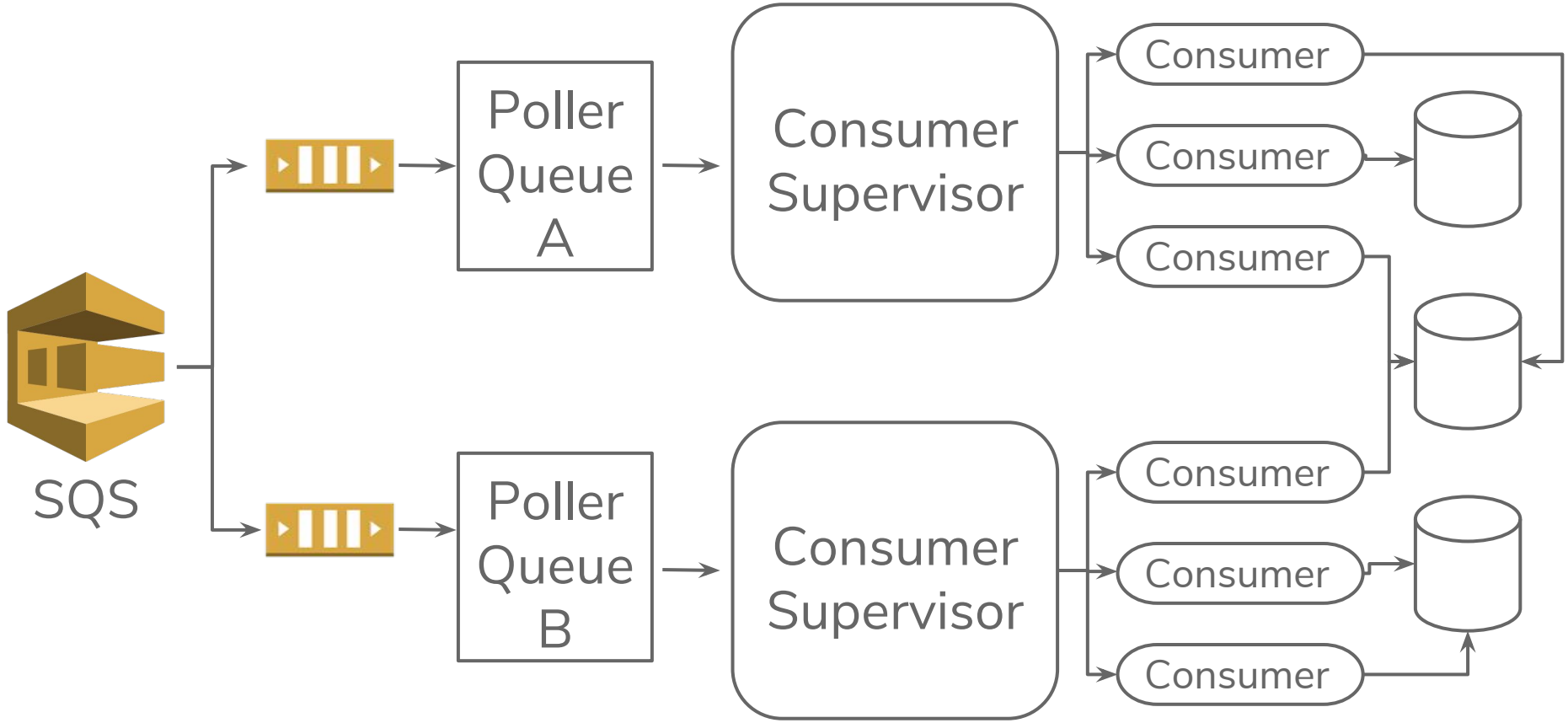

SQS


SQS

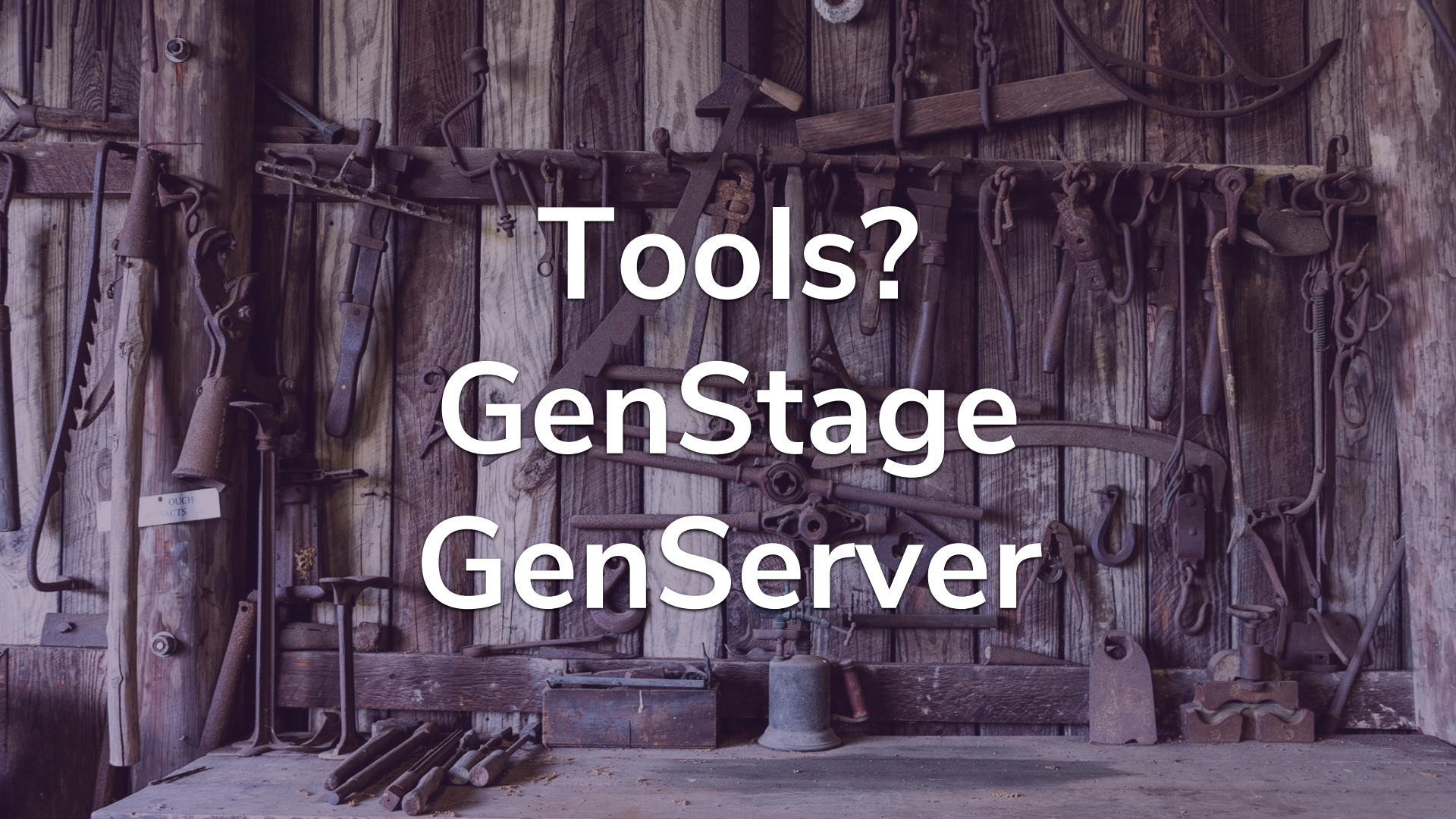

SQS
Poller Queue A
Poller Queue B


SQS
Poller Queue A
Consumer Supervisor
Poller Queue B
Consumer Supervisor


SQS
Poller Queue A
Consumer Supervisor
Consumer
Consumer
Consumer
Poller Queue B
Consumer Supervisor
Consumer
Consumer
Consumer


SQS
Poller Queue A
Poller Queue B
Consumer Supervisor
Consumer Supervisor
Consumer
Consumer
Consumer
Consumer
Consumer
Consumer


Tools?



Tools?
GenStage
GenServer


why GenStage?


why GenStage?
back-pressure

1.

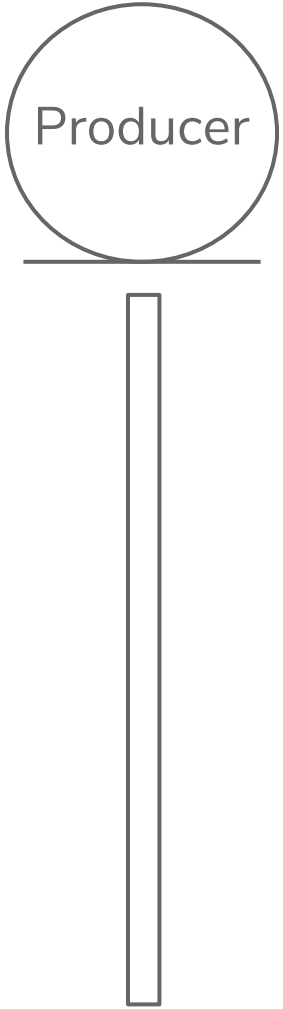
QUEUE CONSUMPTION

GenStage

1.

QUEUE CONSUMPTION

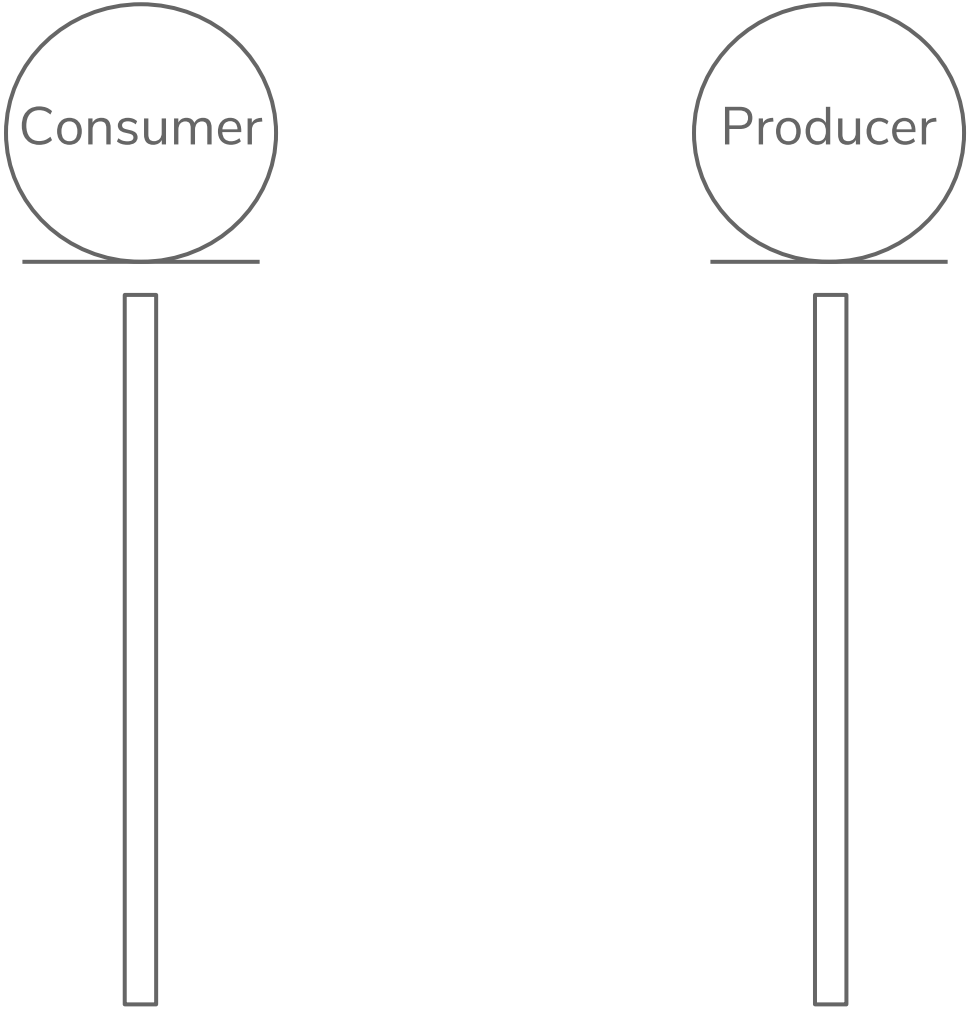
GenStage



1.

QUEUE CONSUMPTION

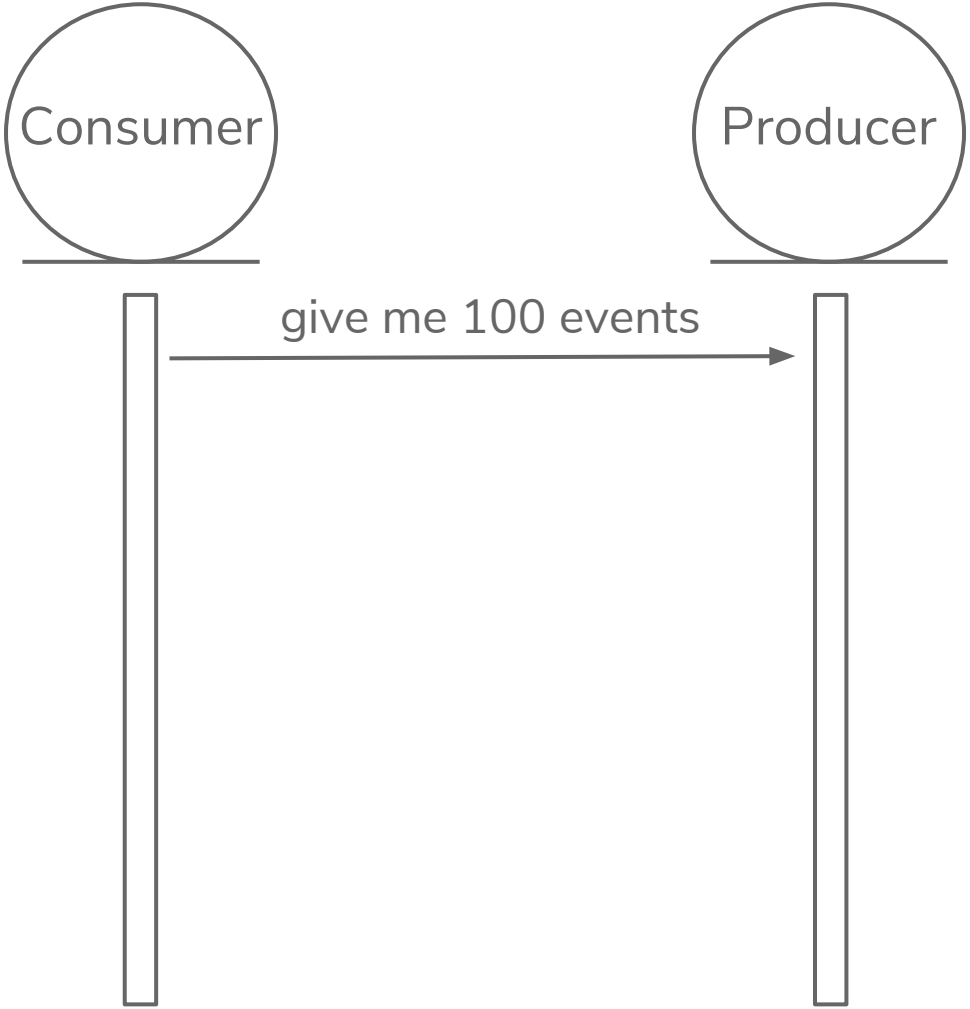
GenStage



1.

QUEUE CONSUMPTION

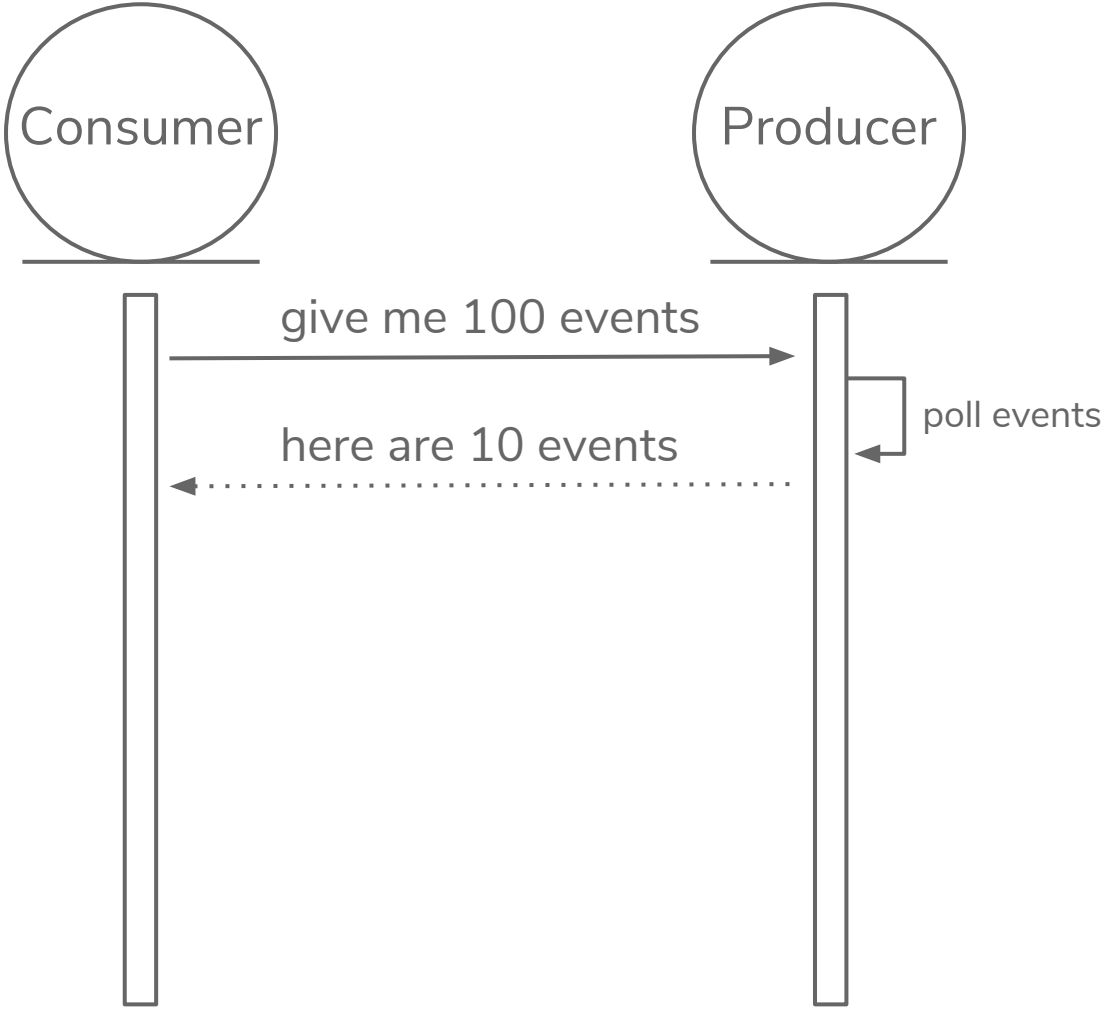
GenStage



1.

QUEUE CONSUMPTION

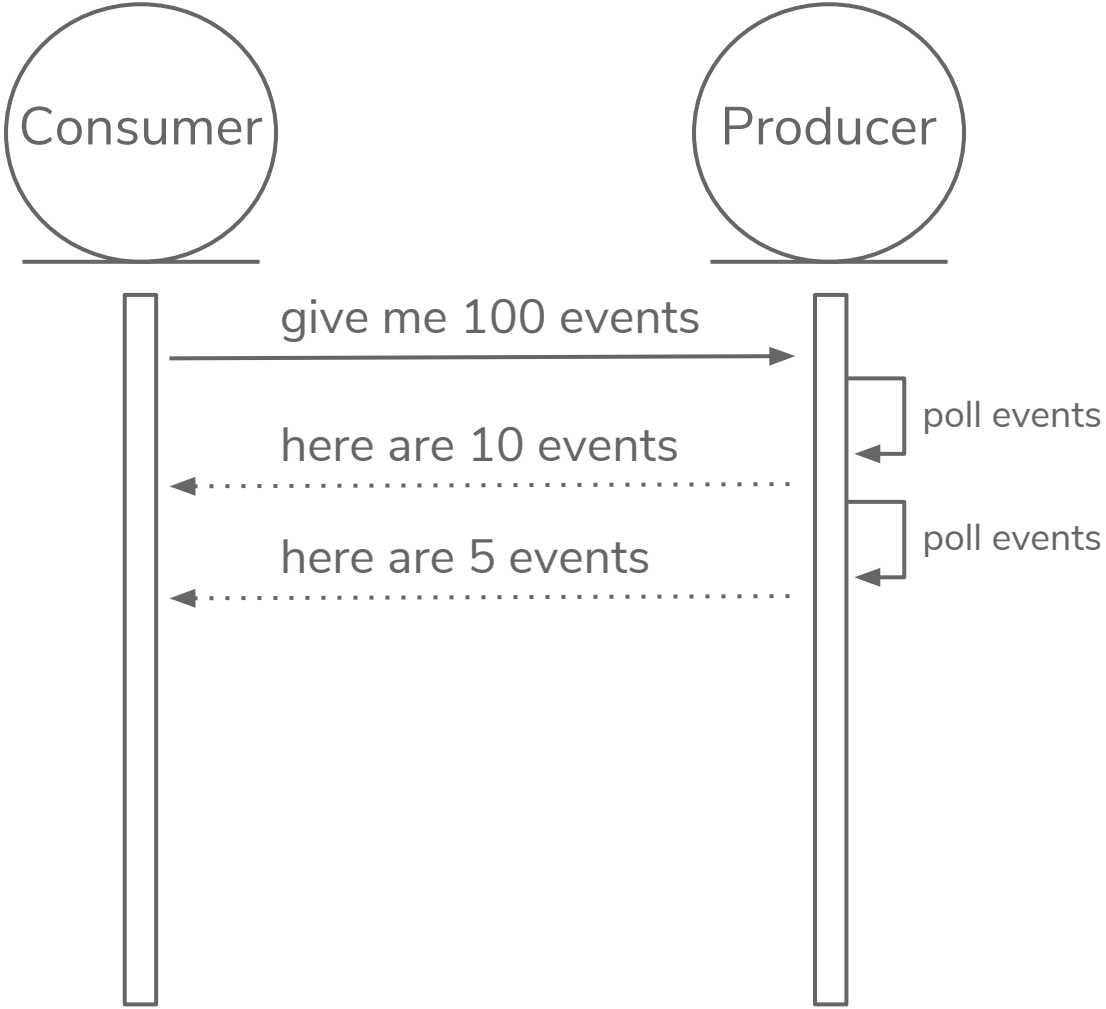
GenStage



1.

QUEUE CONSUMPTION

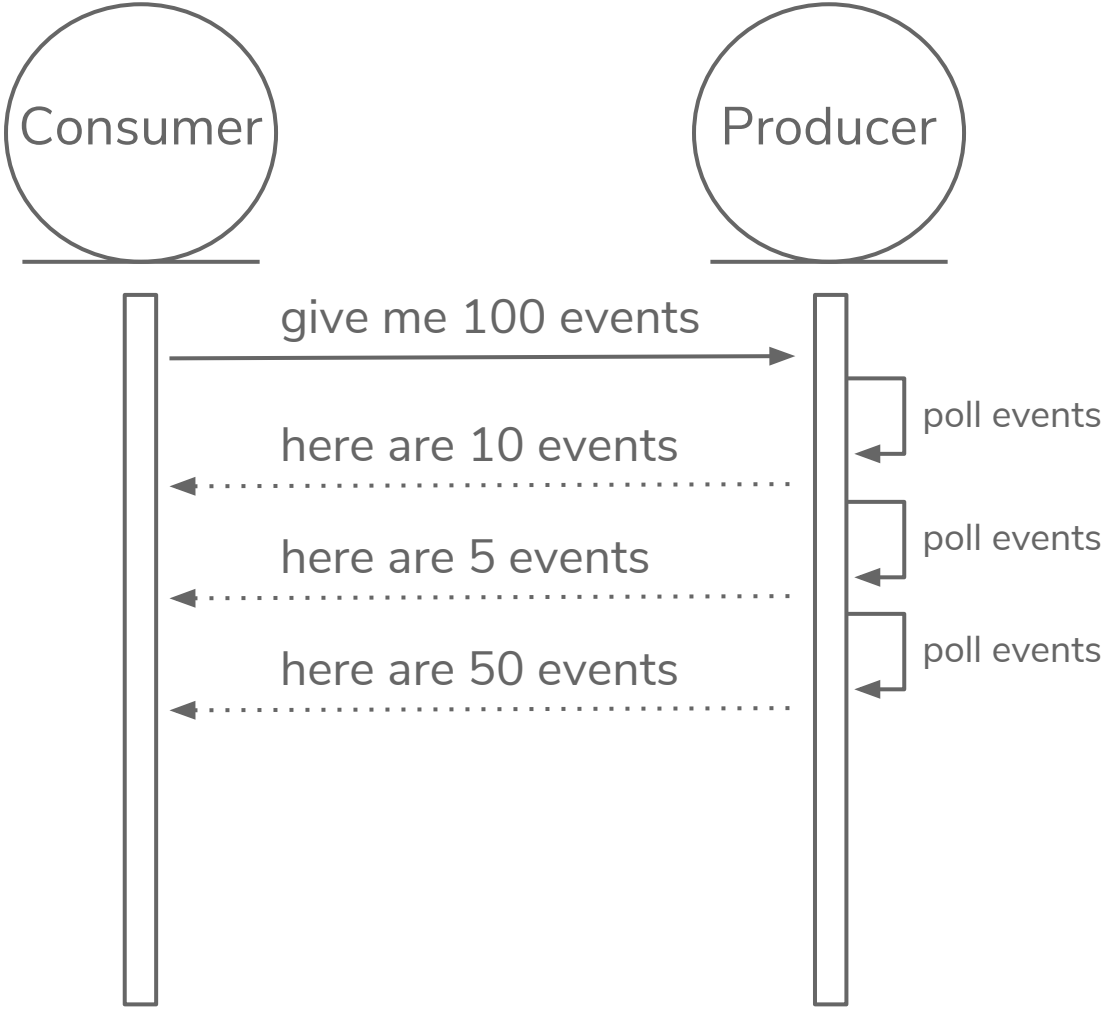
GenStage



1.

QUEUE CONSUMPTION

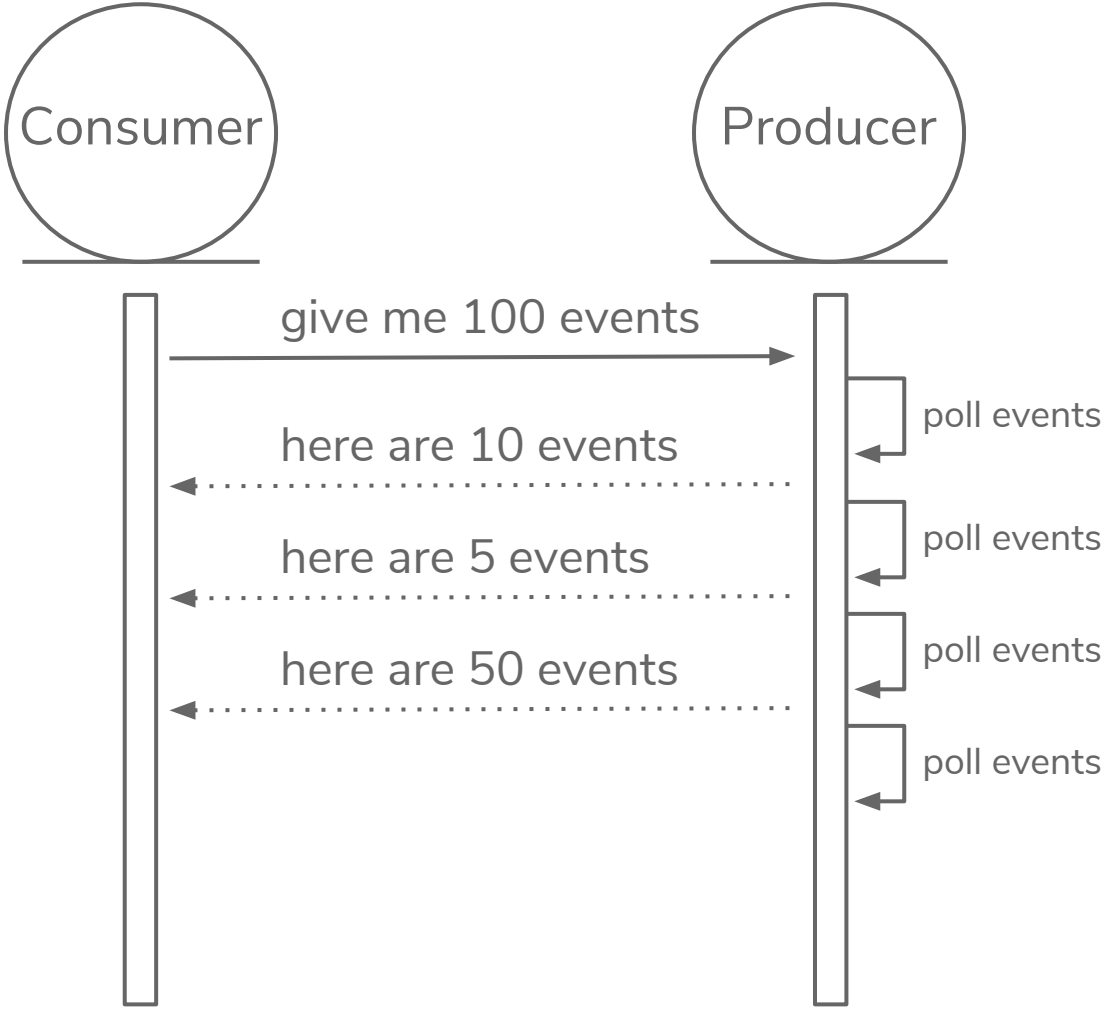
GenStage



1.

QUEUE CONSUMPTION

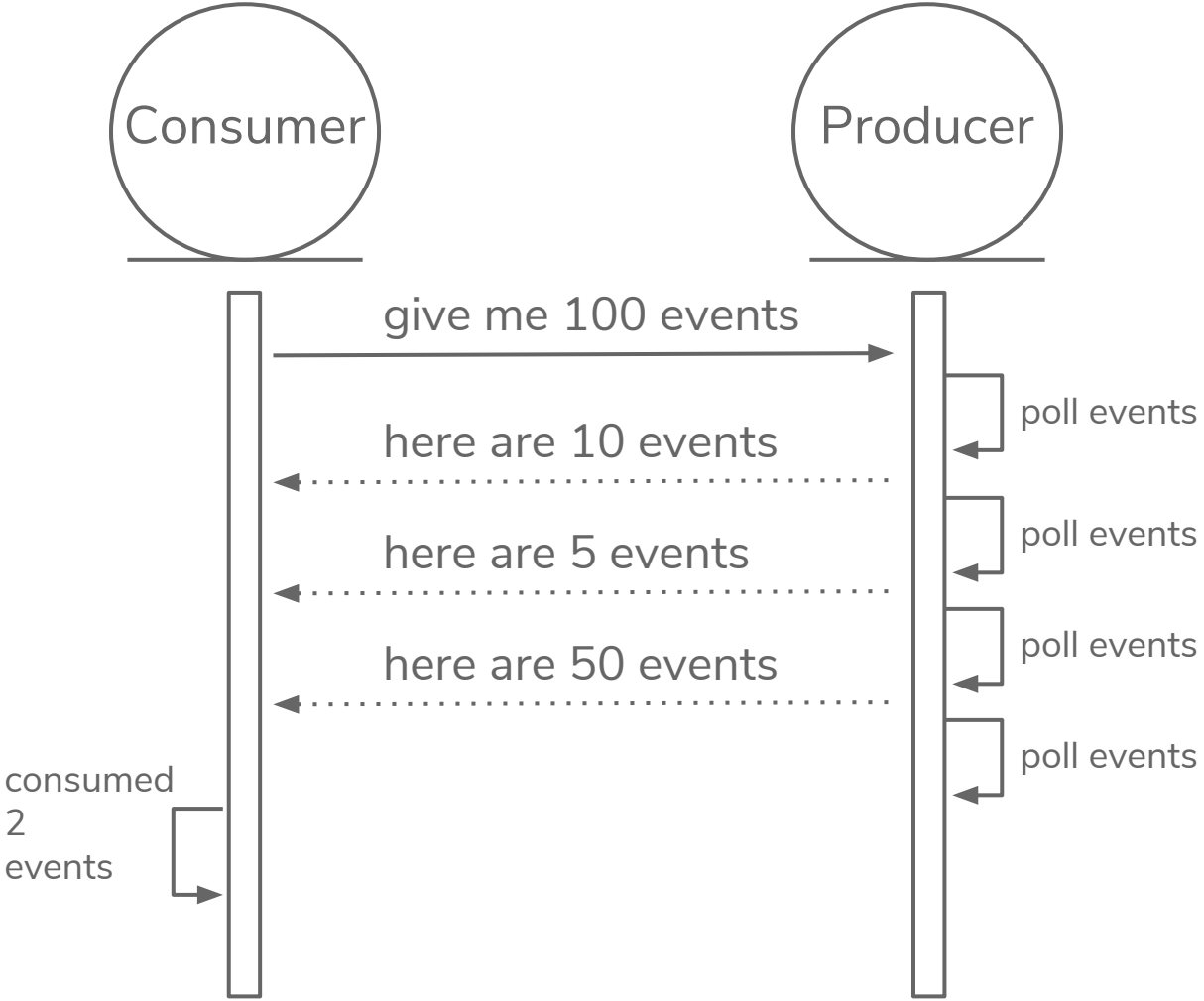
GenStage



1.

QUEUE CONSUMPTION

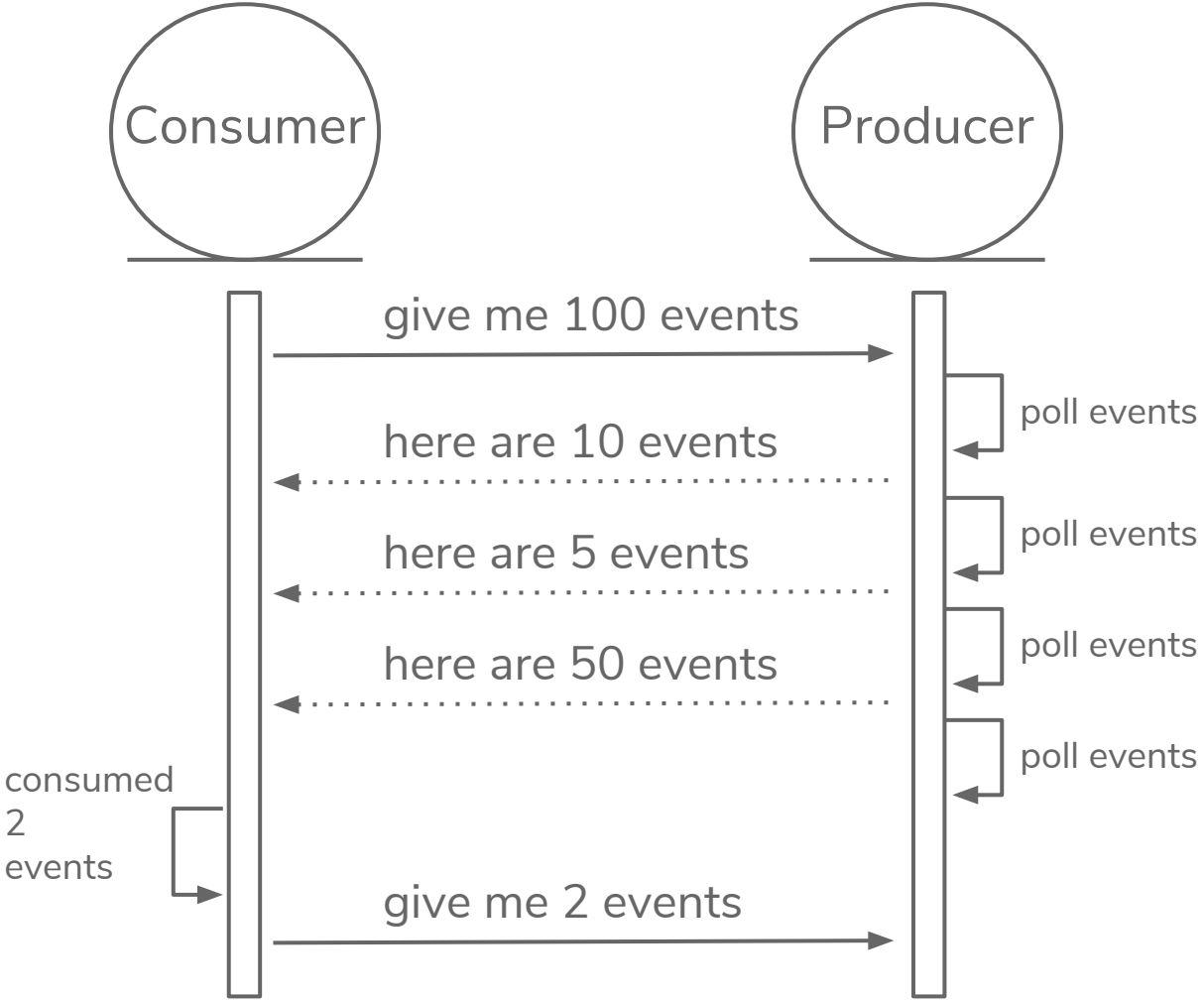
GenStage



1.

QUEUE CONSUMPTION

GenStage



“

Talk is cheap. Show me the code.

Linus Torvalds

0.
CONFIG

```
config :octoconf,
  queues: [
    %{
      name: "stock",
      handler: Octoconf.Handlers.Product,
      concurrency: 10
    },
    %{
      name: "invoice",
      handler: Octoconf.Handlers.Order,
      concurrency: 20
    }
  ]
```

1.

QUEUE CONSUMPTION

GenStage

Producer

```
defmodule Octoconf.Queues.Poller do
  use GenStage
```

1.

QUEUE CONSUMPTION

GenStage

Producer

```
defmodule Octoconf.Queues.Poller do
  use GenStage
  ...

  def init(queue) do
    {
      :producer,
      %{queue: queue[:name],
        events: :queue.new,
        pending_demand:0}
    }
  end

  ...
```

1.

QUEUE CONSUMPTION

GenStage

Producer

```
defmodule Octoconf.Queues.Poller do
  use GenStage
  ...

  def init(queue) do
    {
      :producer,
      %{queue: queue[:name],
        events: :queue.new,
        pending_demand:0}
    }
  end

  def handle_demand(demand, state) do
    state = %{
      state | pending_demand: state.pending_demand + demand
    }
    dispatch_events(state, [])
  end

  ...
```

1.

QUEUE CONSUMPTION

GenStage

Producer

```
defmodule Octoconf.Queues.Poller do
  use GenStage
  ...

  def init(queue) do
    {
      :producer,
      %{queue: queue[:name],
        events: :queue.new,
        pending_demand:0}
    }
  end

  def handle_demand(demand, state) do
    state = %{
      state | pending_demand: state.pending_demand + demand
    }
    dispatch_events(state, [])
  end

  ...
```

1.

QUEUE CONSUMPTION

GenStage

Producer

```
def dispatch_events(%{pending_demand: 0} = state, to_dispatch) do
  do_dispatch_events(state, to_dispatch)
end
```

1.

QUEUE CONSUMPTION

GenStage

Producer

```
def dispatch_events(%{pending_demand: 0} = state, to_dispatch) do
  do_dispatch_events(state, to_dispatch)
end

def dispatch_events(state, to_dispatch) do
  case :queue.out(state.events) do
    {{:value, event}, events} ->
      state = %{
        state | events: events, pending_demand: state.pending_demand - 1
      }
      dispatch_events(state, [event | to_dispatch])

    {:empty, events} ->
      state = %{state | events: events}
      do_dispatch_events(state, to_dispatch)
  end
end
```

1.

QUEUE CONSUMPTION

GenStage

Producer

```
def dispatch_events(%{pending_demand: 0} = state, to_dispatch) do
  do_dispatch_events(state, to_dispatch)
end

def dispatch_events(state, to_dispatch) do
  case :queue.out(state.events) do
    {{:value, event}, events} ->
      state = %{
        state | events: events, pending_demand: state.pending_demand - 1
      }
      dispatch_events(state, [event | to_dispatch])

    {:empty, events} ->
      state = %{state | events: events}
      do_dispatch_events(state, to_dispatch)
  end
end
```

1.

QUEUE CONSUMPTION

GenStage

Producer

```
def dispatch_events(%{pending_demand: 0} = state, to_dispatch) do
  do_dispatch_events(state, to_dispatch)
end

def dispatch_events(state, to_dispatch) do
  case :queue.out(state.events) do
    {{:value, event}, events} ->
      state = %{
        state | events: events, pending_demand: state.pending_demand - 1
      }
      dispatch_events(state, [event | to_dispatch])

    {:empty, events} ->
      state = %{state | events: events}
      do_dispatch_events(state, to_dispatch)
  end
end

defp do_dispatch_events(state, to_dispatch) do
  if state.pending_demand > 0, do: poll()
  to_dispatch = Enum.reverse(to_dispatch)
  {:noreply, to_dispatch, state}
end
```

1.

QUEUE CONSUMPTION

GenStage

Producer

```
def dispatch_events(%{pending_demand: 0} = state, to_dispatch) do
  do_dispatch_events(state, to_dispatch)
end

def dispatch_events(state, to_dispatch) do
  case :queue.out(state.events) do
    {{:value, event}, events} ->
      state = %{
        state | events: events, pending_demand: state.pending_demand - 1
      }
      dispatch_events(state, [event | to_dispatch])

    {:empty, events} ->
      state = %{state | events: events}
      do_dispatch_events(state, to_dispatch)
  end
end

defp do_dispatch_events(state, to_dispatch) do
  if state.pending_demand > 0, do: poll()
  to_dispatch = Enum.reverse(to_dispatch)
  {:noreply, to_dispatch, state}
end
```

1.

QUEUE CONSUMPTION

GenStage

Producer

```
def poll do
  GenStage.cast(self(), :poll)
end

def handle_cast(:poll, state) do
  events =
    @adapter.receive_message(state.queue)
    |> Enum.reduce(state.events, fn msg, acc ->
      Map.put(msg, :queue, state.queue)
      |> :queue.in(acc)
    end)
  dispatch_events(%{state | events: events}, [])
end
```

STEPS

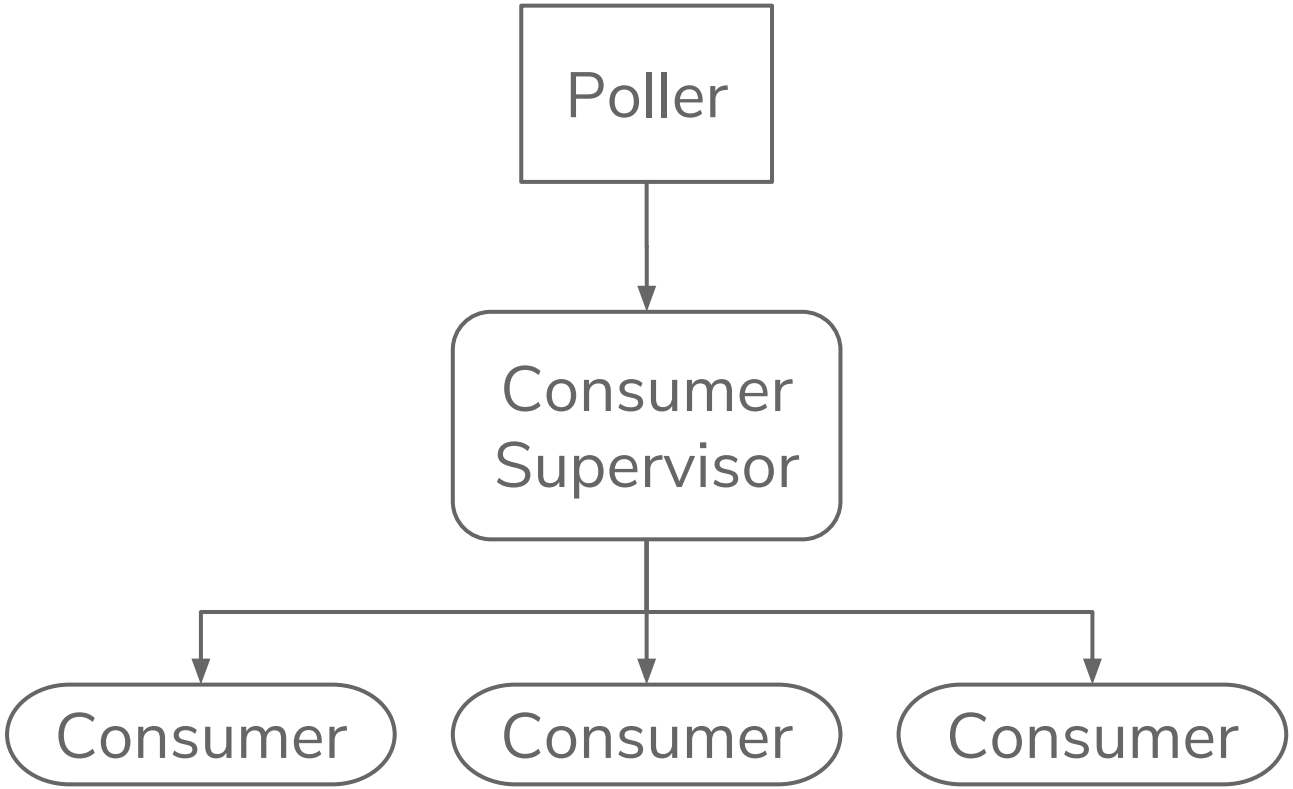
- ~~A queue needs to be consumed~~
- Each element must pass through a pipeline
- Go to a different bucket
- Must work independently
- Flushed whenever is possible

2.

CONSUME MESSAGE

GenStage

Consumer

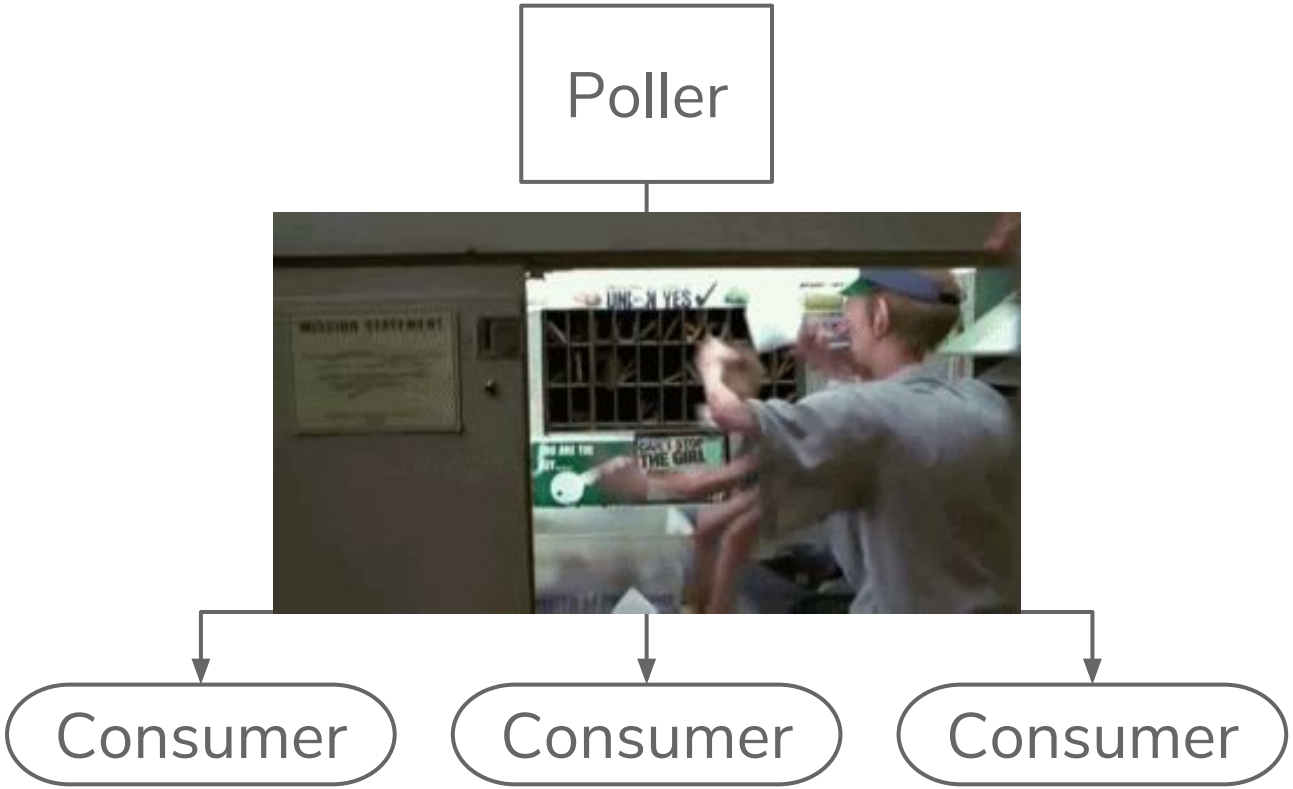


2.

CONSUME MESSAGE

GenStage

Consumer



2.

CONSUME MESSAGE

GenStage

Consumer

```
defmodule Octoconf.Queues.Consumer do
  def start_link(queue) do
    import Supervisor.Spec
    children = [
      worker(queue[:handler], [], restart: :temporary)
    ]
    opts = [
      strategy: :one_for_one,
      subscribe_to: [
        {
          Octoconf.Registry.via_tuple(
            {Octoconf.Queues.Poller, queue[:name]}
          ),
          min_demand: 0,
          max_demand: queue[:concurrency]
        }
      ]
    ]
    ConsumerSupervisor.start_link(children, opts)
  end
end
```

2.

CONSUME MESSAGE

GenStage Consumer

```
defmodule Octoconf.Queues.Consumer do
  def start_link(queue) do
    import Supervisor.Spec
    children = [
      worker(queue[:handler], [], restart: :temporary)
    ]
    opts = [
      strategy: :one_for_one,
      subscribe_to: [
        {
          Octoconf.Registry.via_tuple(
            {Octoconf.Queues.Poller, queue[:name]}
          ),
          min_demand: 0,
          max_demand: queue[:concurrency]
        }
      ]
    ]
    ConsumerSupervisor.start_link(children, opts)
  end
end
```

2.

CONSUME MESSAGE

GenStage

Consumer

```
defmodule Octoconf.Handlers.Product do
  def start_link(message) do
    Task.start_link(__MODULE__, :handle, [message])
  end

  def handle(message) do
    message
    |> do_some
    |> real
    |> stuff
    |> here
    |> Octoconf.Dispatchers.Partner.add_message
  end
end
```

STEPS

- ~~A queue needs to be consumed~~
- ~~Each element must pass through a pipeline~~
- Go to a different bucket
- Must work independently
- Flushed whenever is possible

2.

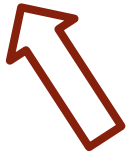
CONSUME MESSAGE

GenStage

Consumer

```
defmodule Octoconf.Handlers.Product do
  def start_link(message) do
    Task.start_link(__MODULE__, :handle, [message])
  end

  def handle(message) do
    message
    |> do_some
    |> real
    |> stuff
    |> here
    |> Octoconf.Dispatchers.Partner.add_message
  end
end
```



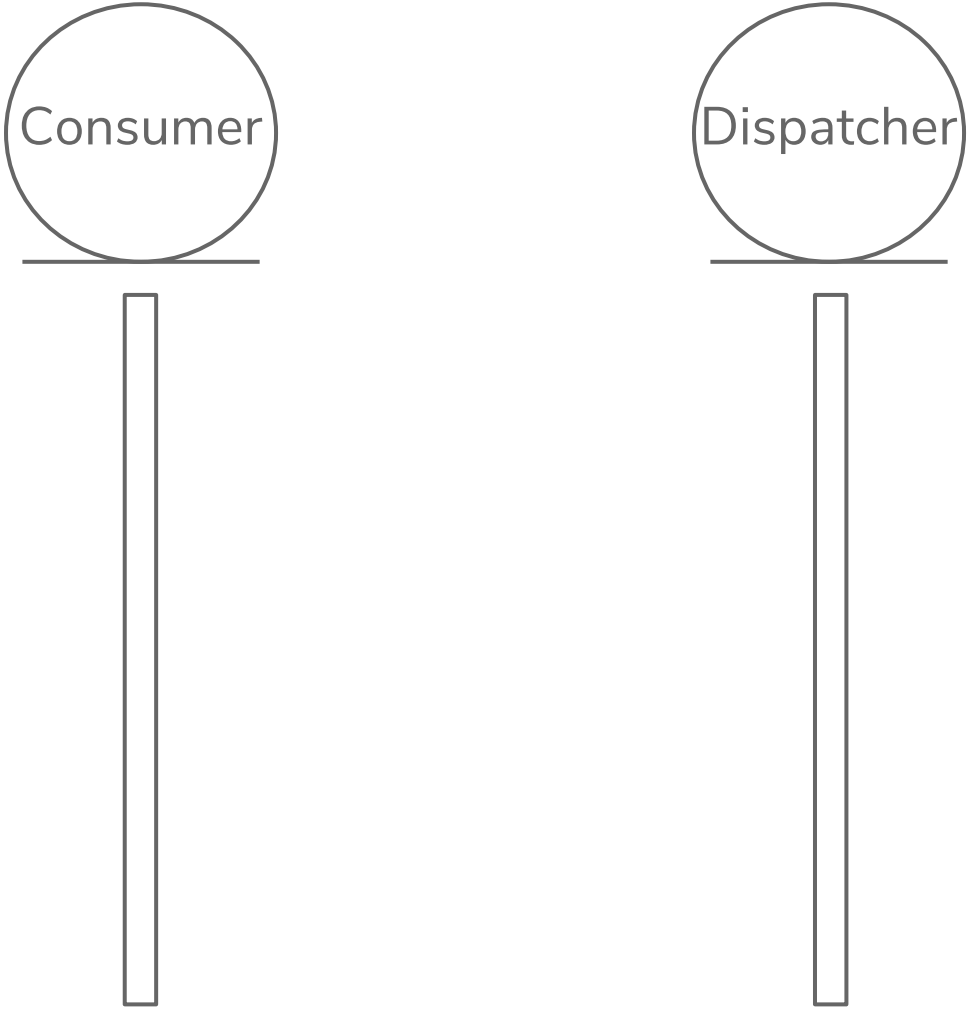
STEPS

- ~~A queue needs to be consumed~~
- ~~Each element must pass through a pipeline~~
- ~~Go to a different bucket~~
- Must work independently
- Flushed whenever is possible

3.

DISPATCH MESSAGES

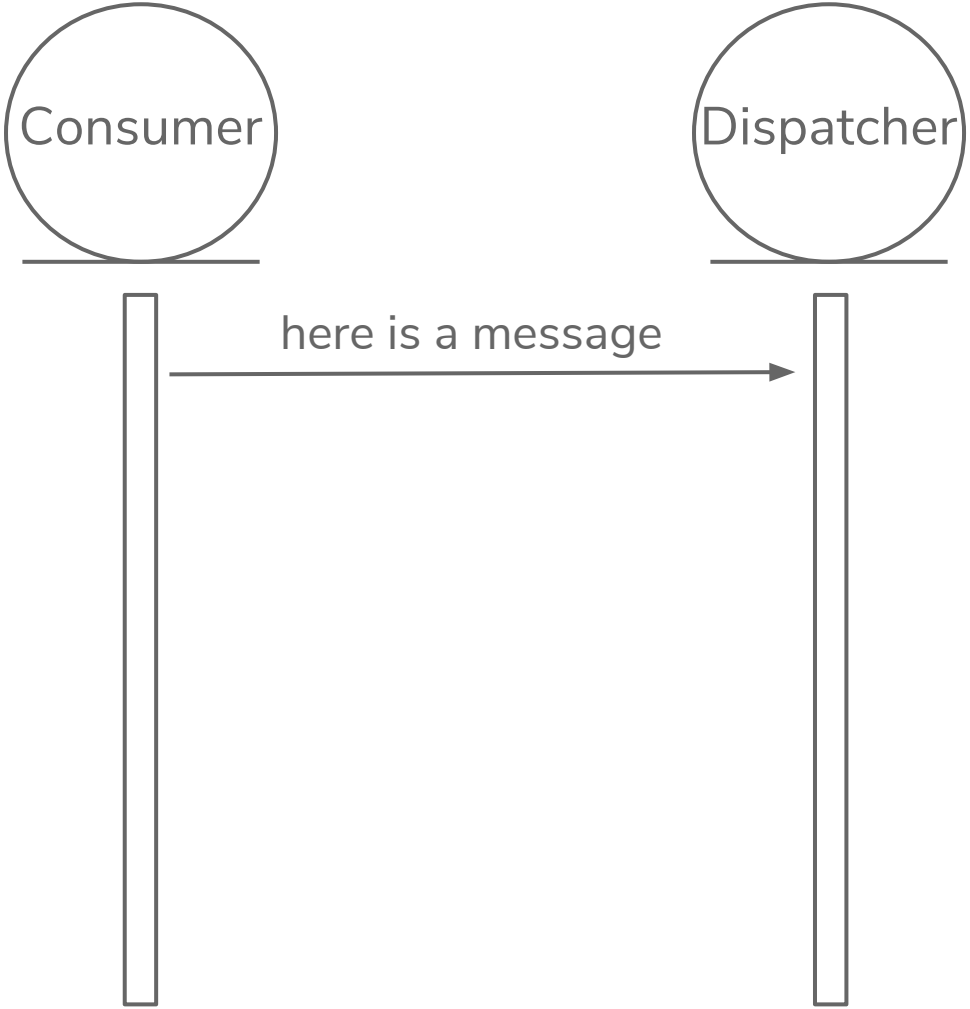
GenServer



3.

DISPATCH MESSAGES

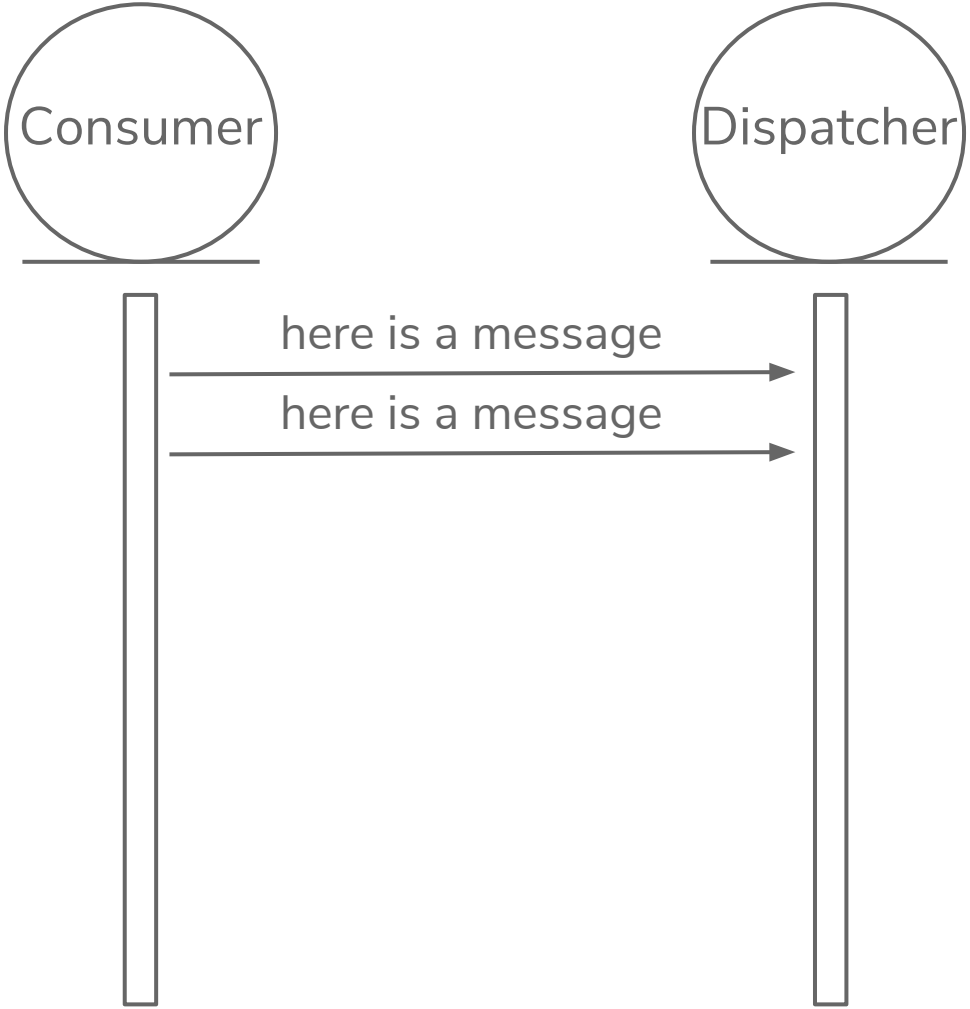
GenServer



3.

DISPATCH MESSAGES

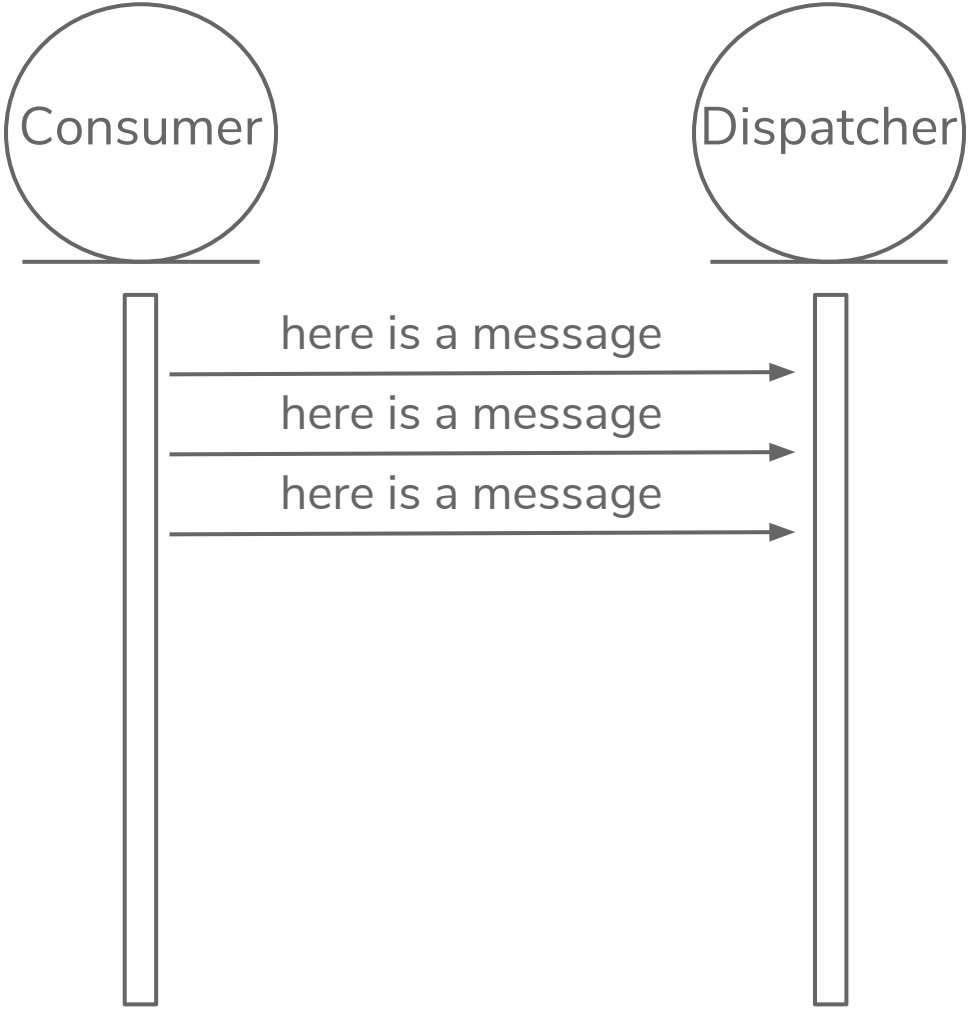
GenServer



3.

DISPATCH MESSAGES

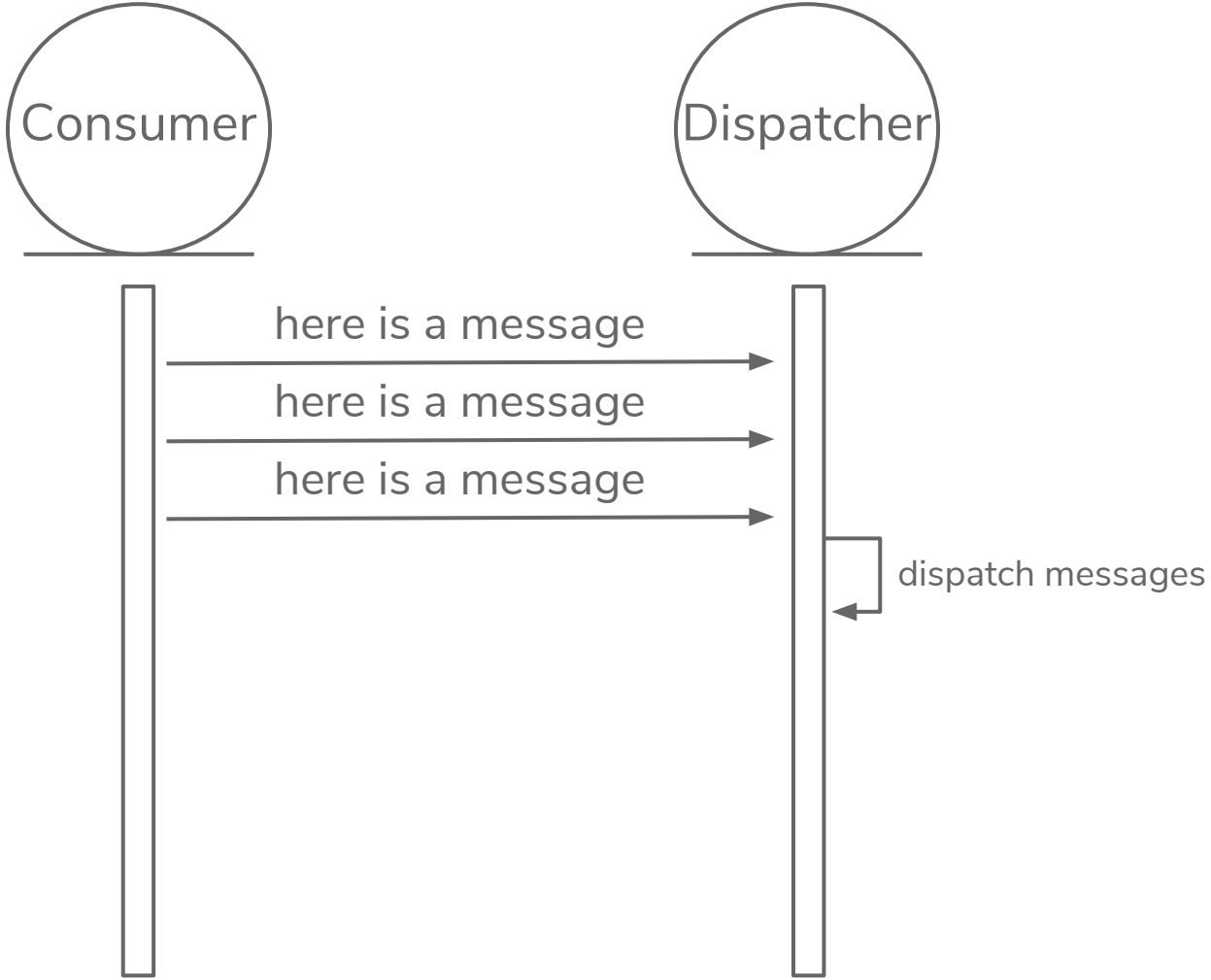
GenServer



3.

DISPATCH MESSAGES

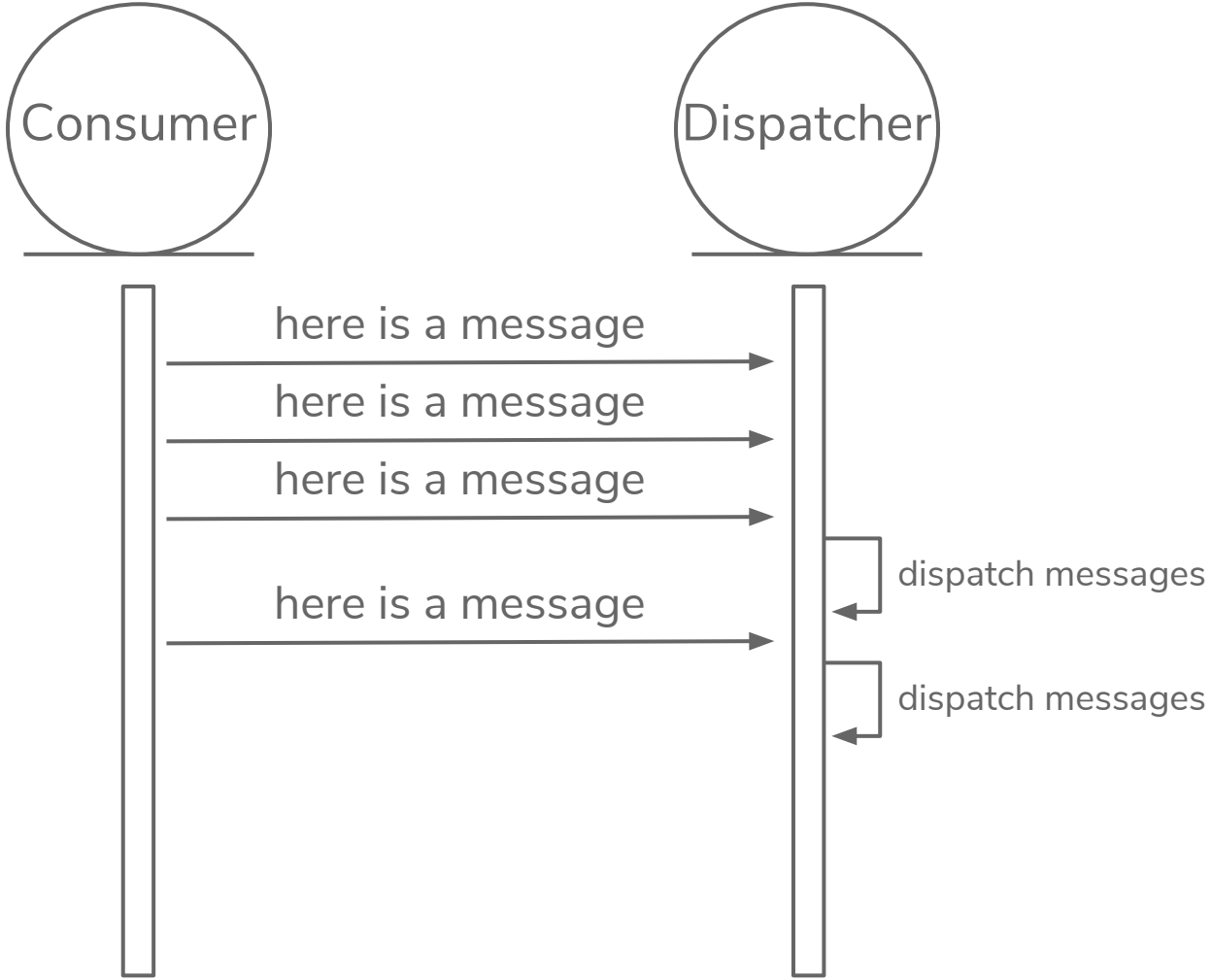
GenServer



3.

DISPATCH MESSAGES

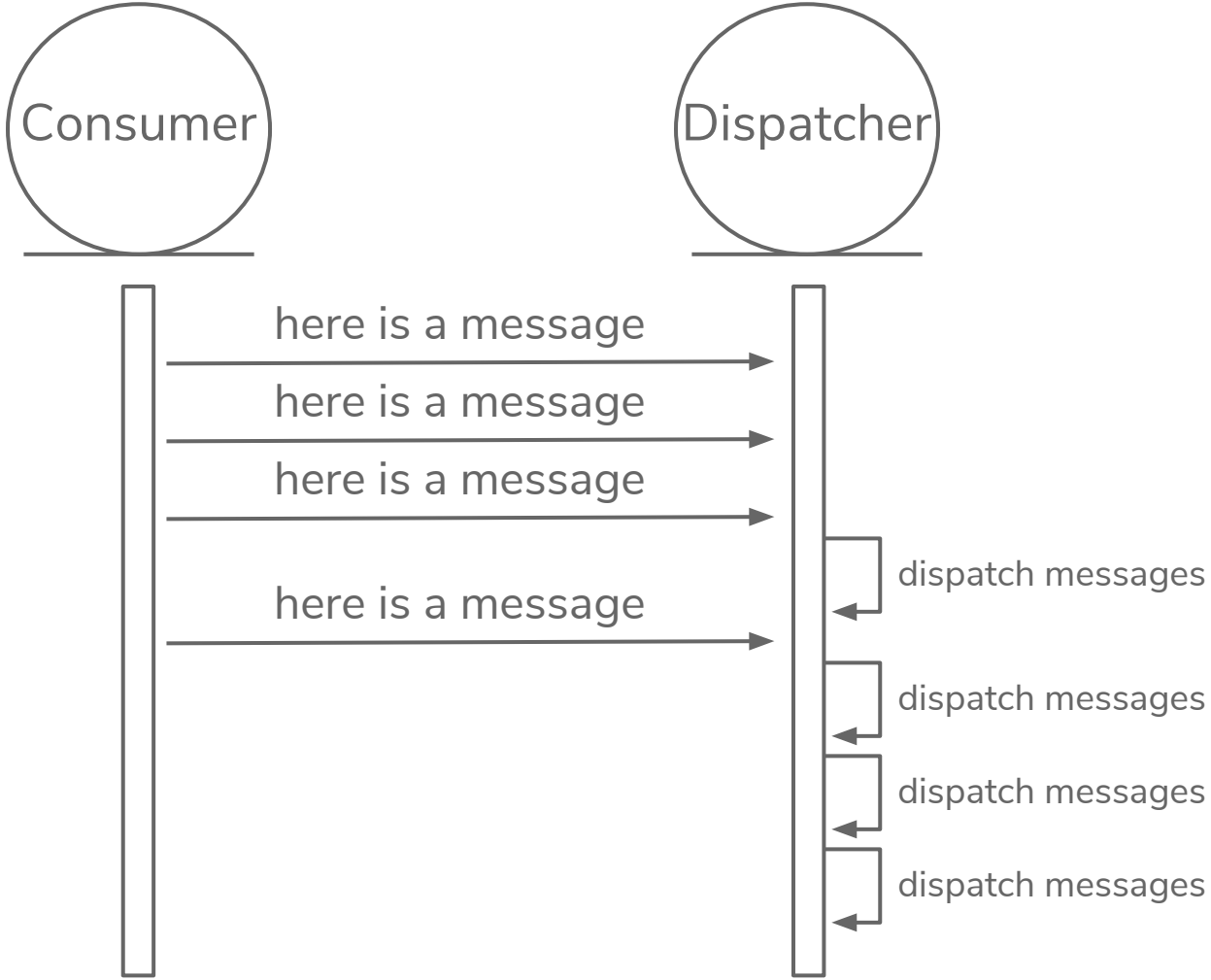
GenServer



3.

DISPATCH MESSAGES

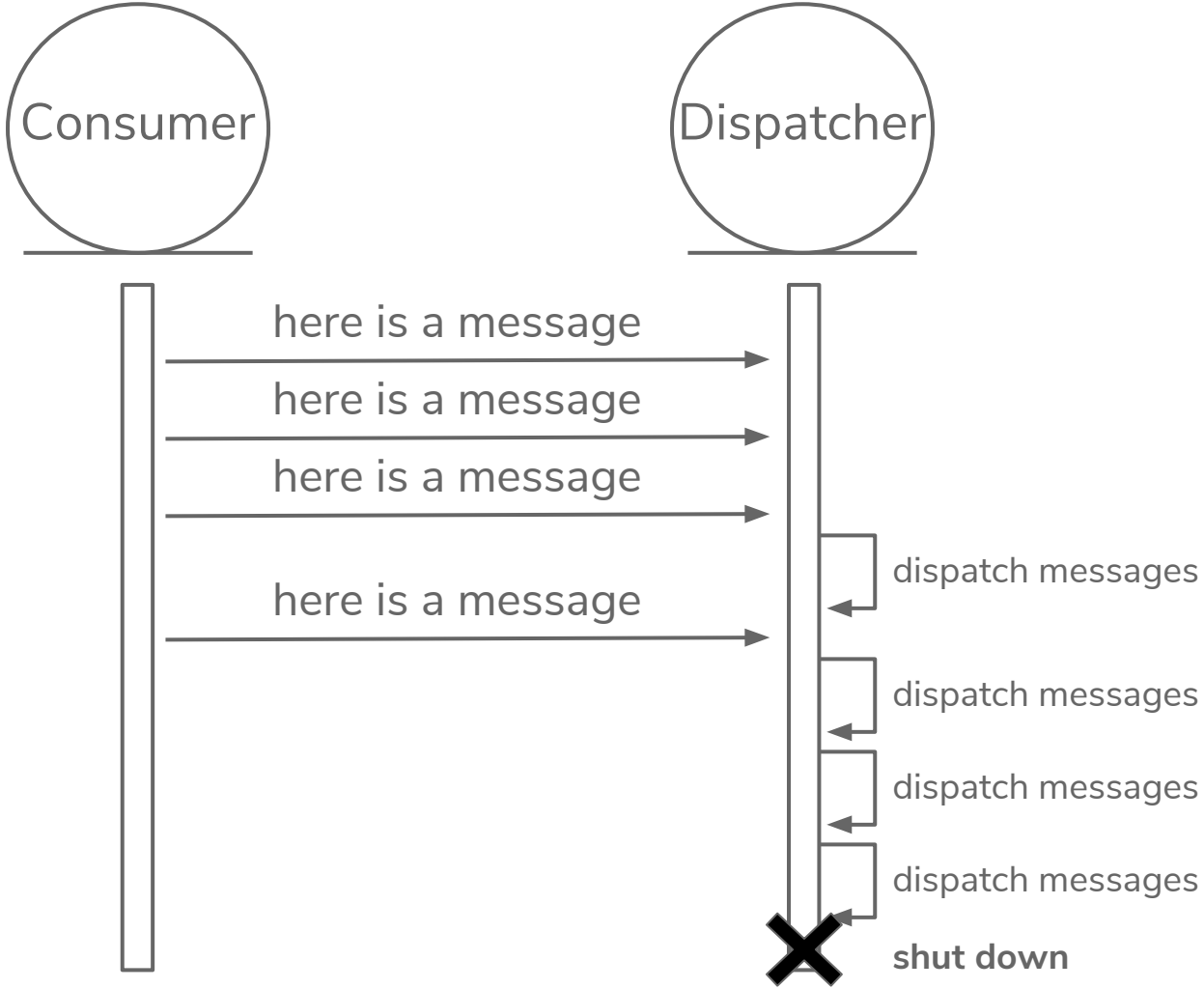
GenServer



3.

DISPATCH MESSAGES

GenServer



3.

DISPATCH MESSAGES

GenServer

```elixir
defmodule Octoconf.Dispatchers.Partner do

  ...

  def add_message(message) do
    name = process_name(message)
    unless Octoconf.Registry.exists_globally?(name) do
      Dispatchers.Supervisor.add_dispatcher(__MODULE__, name, message)
    end
    Octoconf.Registry.via_global_tuple(name)
    |> GenServer.cast({:add_message, message})
  end

  def handle_cast({:add_message, message}, state) do
    state = %{state | events: :queue.in(message, state.events)}
    {:noreply, state}
  end

  ...

end
```

3.

DISPATCH MESSAGES

GenServer

```elixir
defmodule Octoconf.Dispatchers.Partner do

  ...

  def add_message(message) do
    name = process_name(message)
    unless Octoconf.Registry.exists_globally?(name) do
      Dispatchers.Supervisor.add_dispatcher(__MODULE__, name, message)
    end
    Octoconf.Registry.via_global_tuple(name)
    |> GenServer.cast({:add_message, message})
  end

  def handle_cast({:add_message, message}, state) do
    state = %{state | events: :queue.in(message, state.events)}
    {:noreply, state}
  end

  ...

end
```

3.

DISPATCH MESSAGES

GenServer

```
defmodule Octoconf.Dispatchers.Partner do
  ...

  def init(message) do
    send(self(), :dispatch_events)
    {
      :ok,
      %{
        account: message.body[:account],
        queue: message[:queue],
        events: :queue.new,
        empty_dispatch: 0
      }
    }
  end

  ...
end
```

3.

DISPATCH MESSAGES

GenServer

```
defmodule Octoconf.Dispatchers.Partner do
  ...

  def init(message) do
    send(self(), :dispatch_events)
    {
      :ok,
      %{
        account: message.body[:account],
        queue: message[:queue],
        events: :queue.new,
        empty_dispatch: 0
      }
    }
  end

  ...
end
```

3.

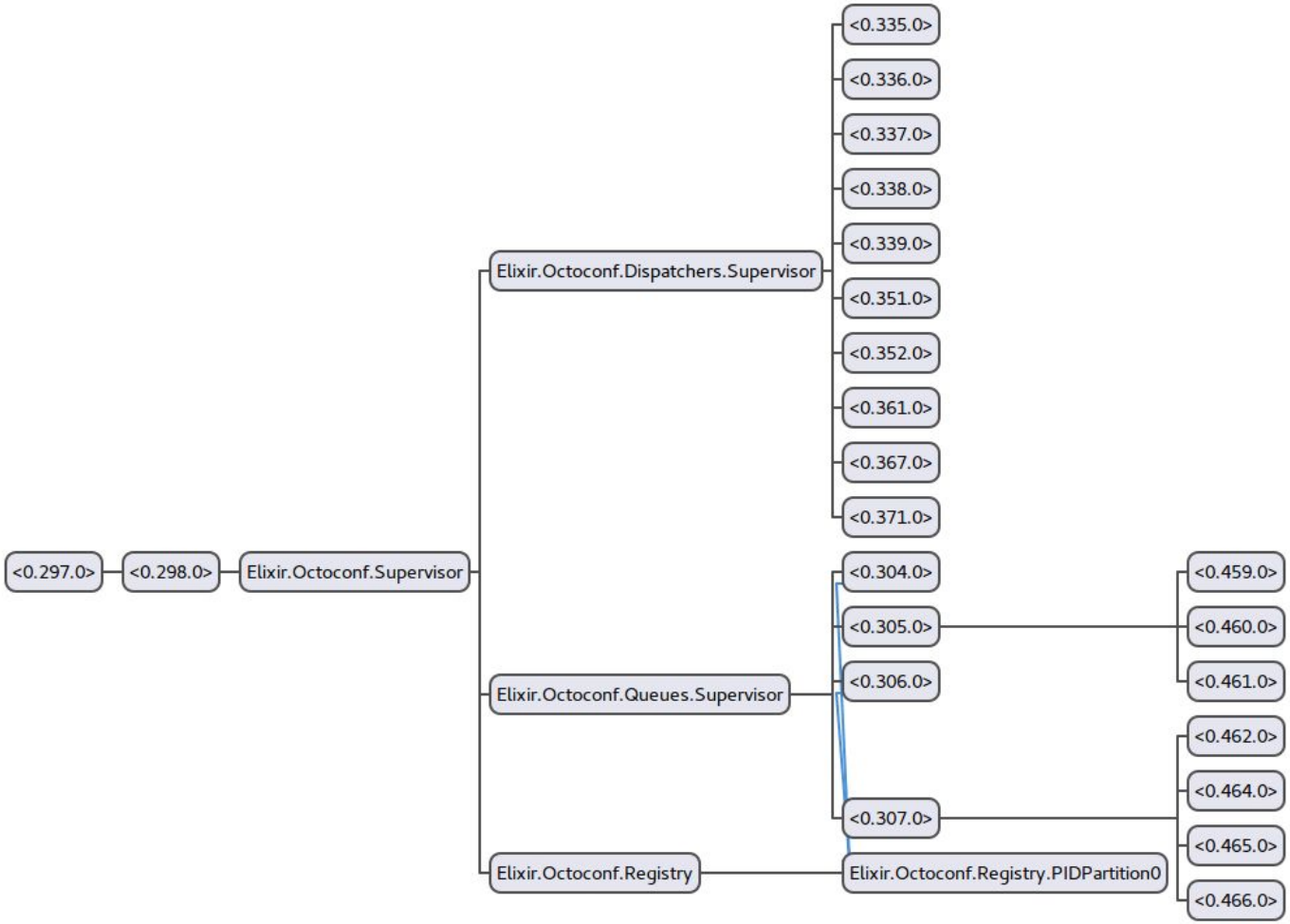
DISPATCH MESSAGES

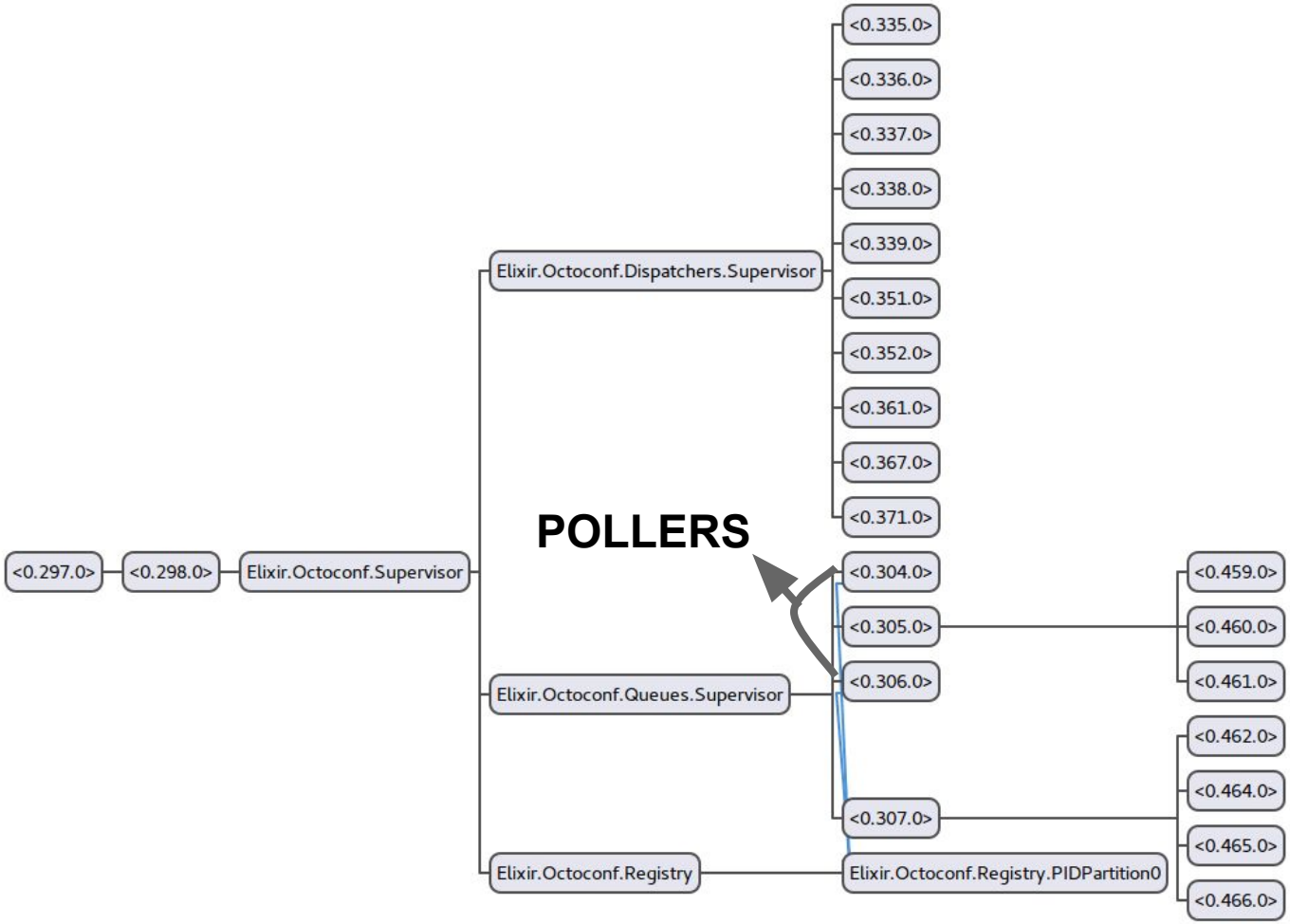
GenServer

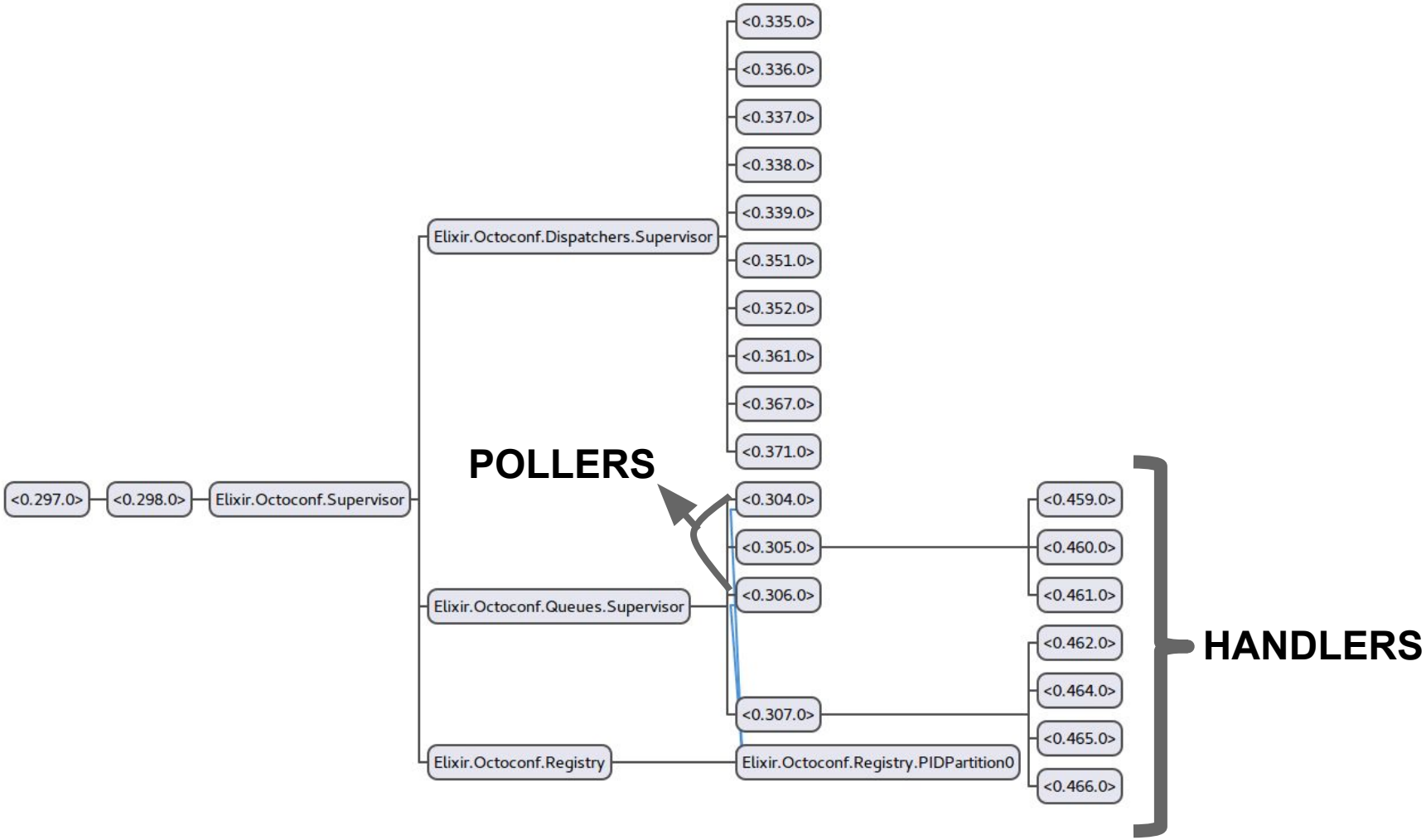
```
def handle_info(:dispatch_events, state) do
  state = dispatch_events(state)
  if state.empty_dispatch >= @empty_dispatch_limit do
    {:stop, :normal, state}
  else
    Process.send_after(
      self(),
      :dispatch_events,
      @dispatch_timeout
    )
    {:noreply, state}
  end
end
```

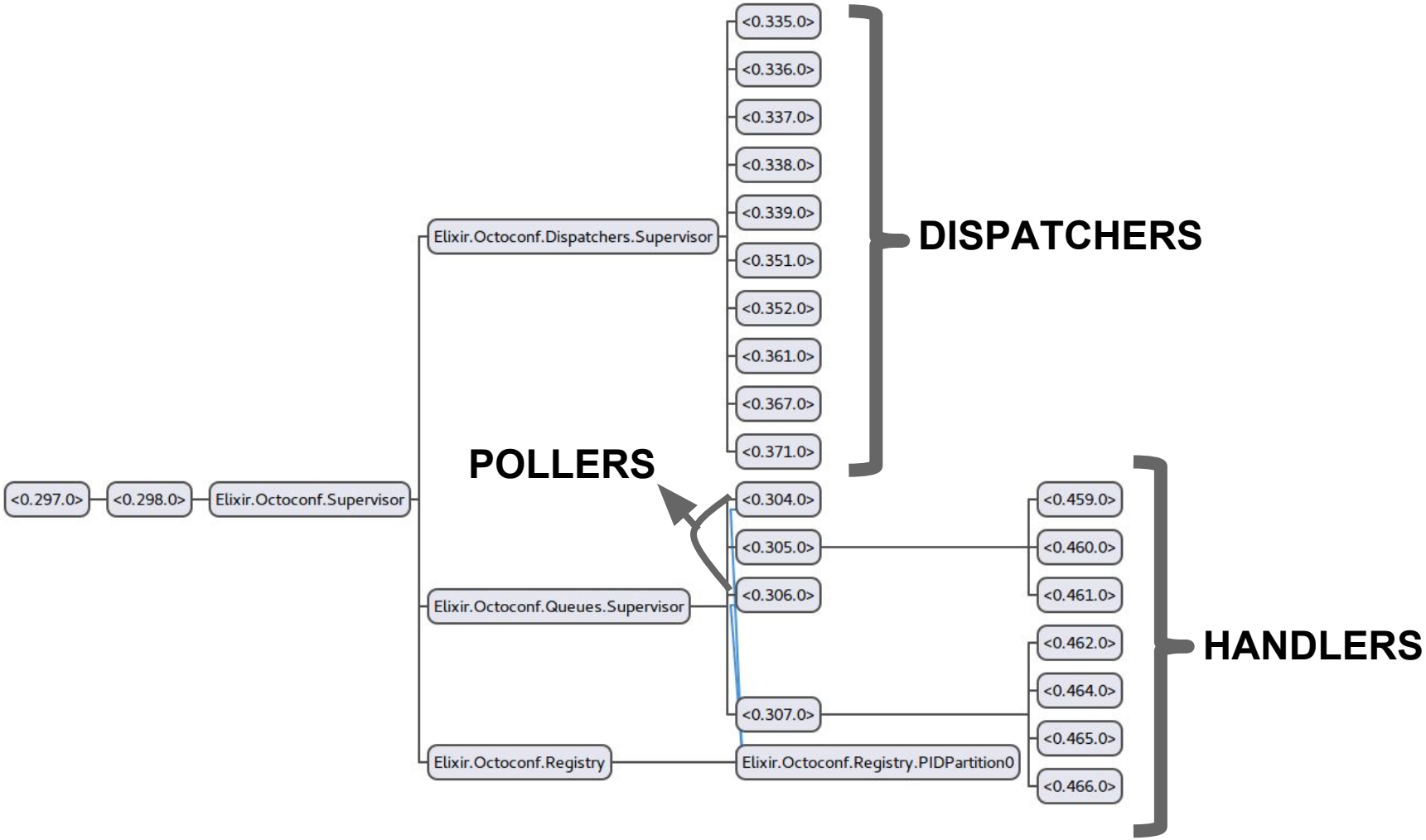
STEPS

- ~~A queue needs to be consumed~~
- ~~Each element must pass through a pipeline~~
- ~~Go to a different bucket~~
- ~~Must work independently~~
- ~~Flushed whenever is possible~~


<0.297.0>
<0.298.0>
Elixir.Octoconf.Supervisor
Elixir.Octoconf.Dispatchers.Supervisor
<0.335.0>
<0.336.0>
<0.337.0>
<0.338.0>
<0.339.0>
<0.351.0>
<0.352.0>
<0.361.0>
<0.367.0>
<0.371.0>
Elixir.Octoconf.Queues.Supervisor
<0.304.0>
<0.305.0>
<0.306.0>
<0.307.0>
<0.459.0>
<0.460.0>
<0.461.0>
<0.462.0>
<0.464.0>
<0.465.0>
<0.466.0>
Elixir.Octoconf.Registry
Elixir.Octoconf.Registry.PIDPartition0


<0.297.0>
<0.298.0>
Elixir.Octoconf.Supervisor
Elixir.Octoconf.Dispatchers.Supervisor
<0.335.0>
<0.336.0>
<0.337.0>
<0.338.0>
<0.339.0>
<0.351.0>
<0.352.0>
<0.361.0>
<0.367.0>
<0.371.0>
POLLERS
Elixir.Octoconf.Queues.Supervisor
<0.304.0>
<0.305.0>
<0.306.0>
<0.307.0>
<0.459.0>
<0.460.0>
<0.461.0>
<0.462.0>
<0.464.0>
<0.465.0>
<0.466.0>
Elixir.Octoconf.Registry
Elixir.Octoconf.Registry.PIDPartition0


<0.297.0>
<0.298.0>
Elixir.Octoconf.Supervisor
Elixir.Octoconf.Dispatchers.Supervisor
Elixir.Octoconf.Queues.Supervisor
Elixir.Octoconf.Registry
<0.335.0>
<0.336.0>
<0.337.0>
<0.338.0>
<0.339.0>
<0.351.0>
<0.352.0>
<0.361.0>
<0.367.0>
<0.371.0>
POLLERS
<0.304.0>
<0.305.0>
<0.306.0>
<0.307.0>
Elixir.Octoconf.Registry.PIDPartition0
<0.459.0>
<0.460.0>
<0.461.0>
<0.462.0>
<0.464.0>
<0.465.0>
<0.466.0>
HANDLERS


<0.297.0>
<0.298.0>
Elixir.Octoconf.Supervisor
Elixir.Octoconf.Dispatchers.Supervisor
Elixir.Octoconf.Queues.Supervisor
Elixir.Octoconf.Registry
<0.335.0>
<0.336.0>
<0.337.0>
<0.338.0>
<0.339.0>
<0.351.0>
<0.352.0>
<0.361.0>
<0.367.0>
<0.371.0>
DISPATCHERS
POLLERS
<0.304.0>
<0.305.0>
<0.306.0>
<0.307.0>
Elixir.Octoconf.Registry.PIDPartition0
<0.459.0>
<0.460.0>
<0.461.0>
<0.462.0>
<0.464.0>
<0.465.0>
<0.466.0>
HANDLERS

STATS

~ 17M msg/day

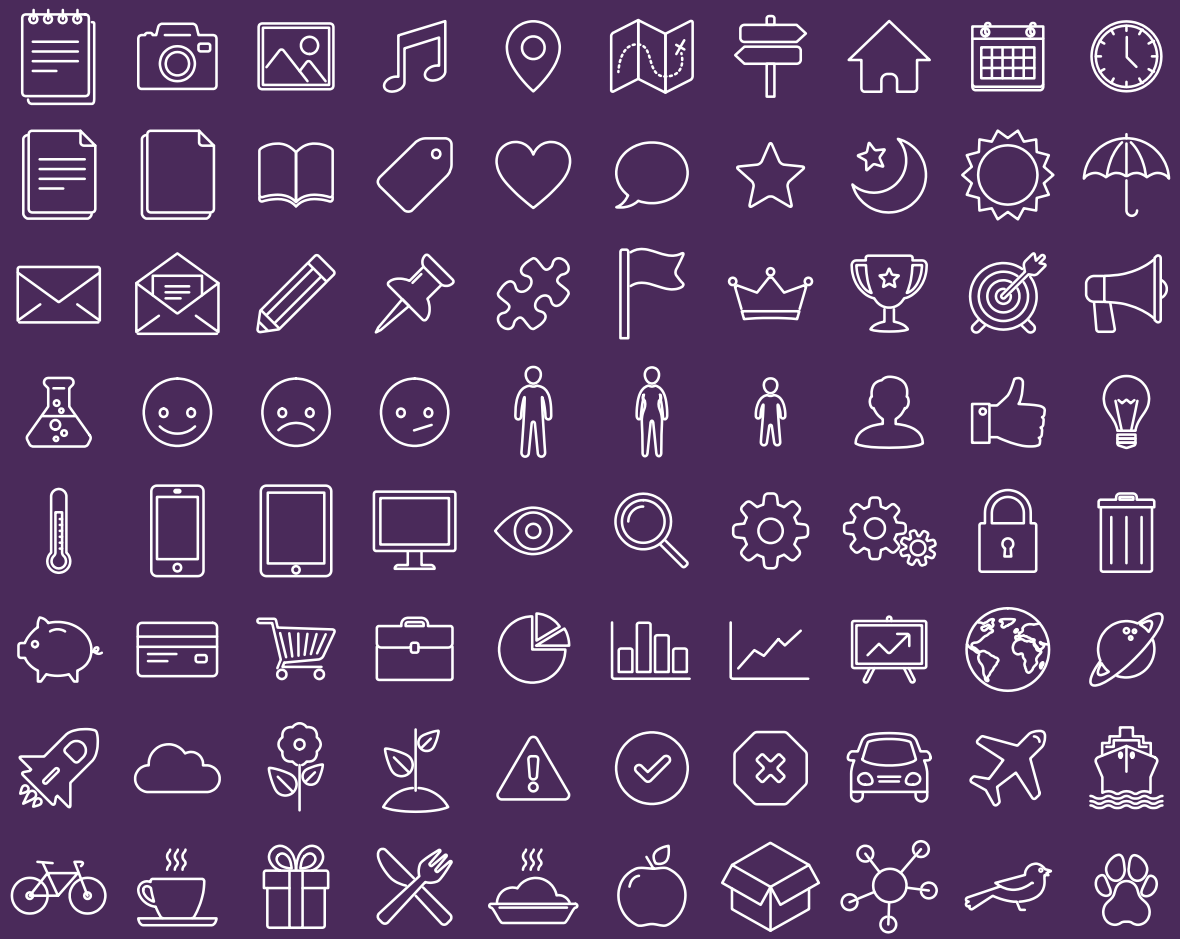
~ 19.3M req/day

~ 2.9K buckets

Obrigad☺

https://github.com/dojusa/octoconf

@dojusa



SlidesCarnival icons are editable shapes.

This means that you can:

- Resize them without losing quality.
- Change line color, width and style.

Isn't that nice? :)

Examples: