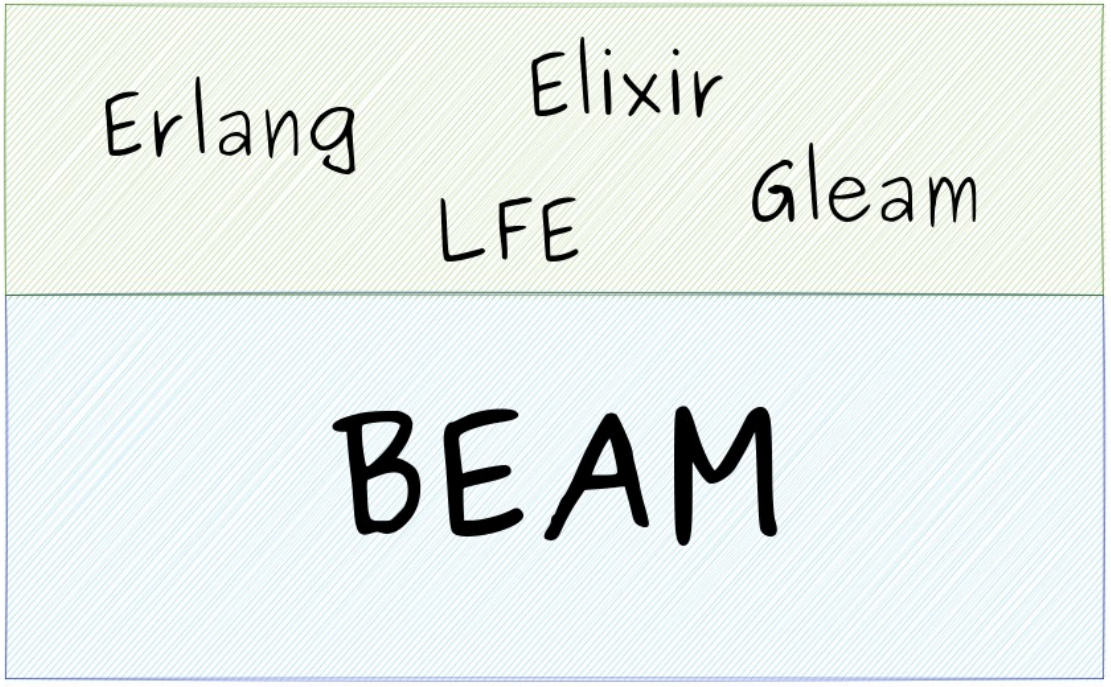
Concurrency in action

The bartender asks what they want.
Two threads walk into a bar.


Erlang
Elixir
LFE
Gleam
BEAM

process

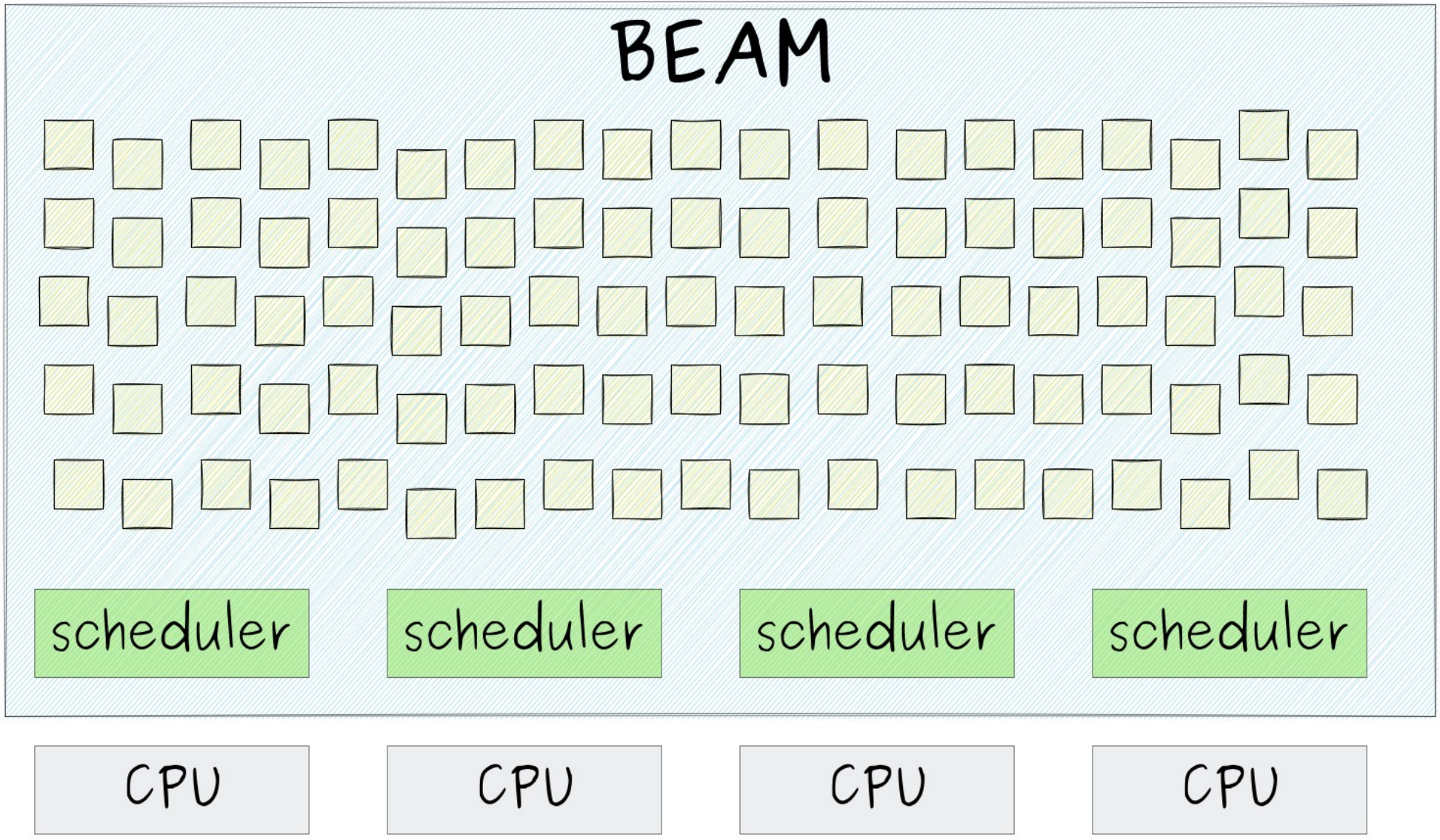
foo(...)
bar(...)
...

```
...
spawn(fn -> ... end)
...
```

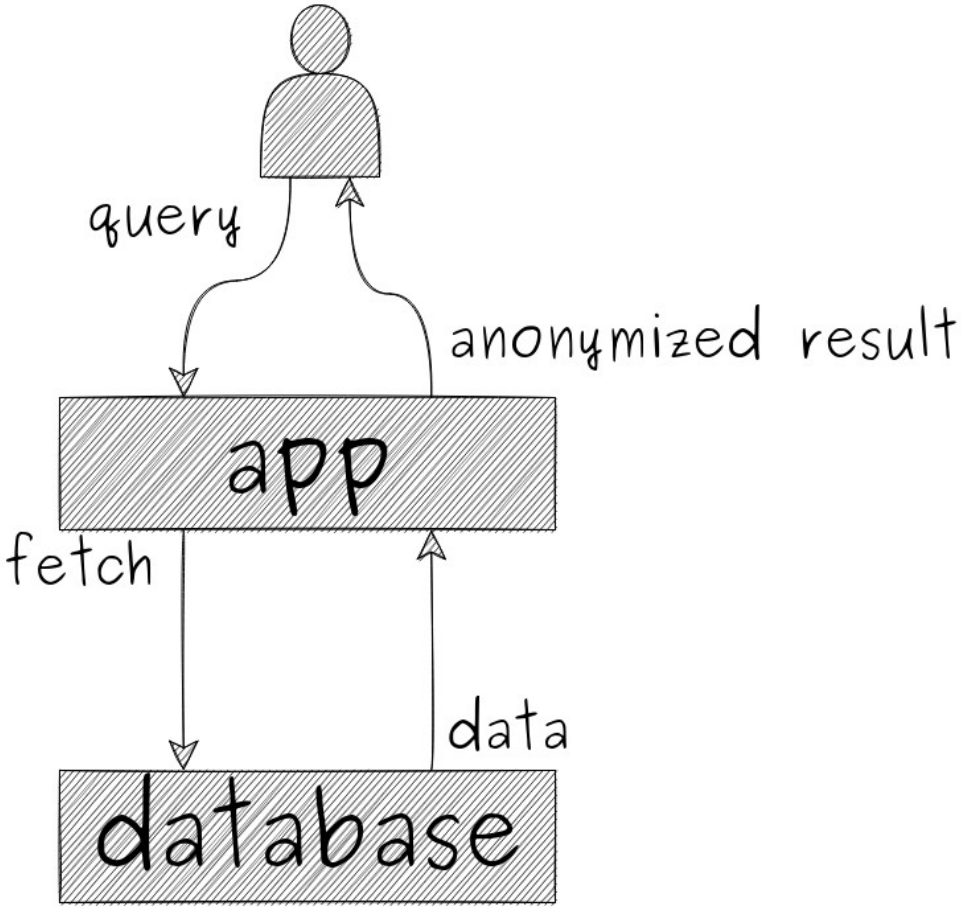
```
pid = spawn(fn -> ... end)
```

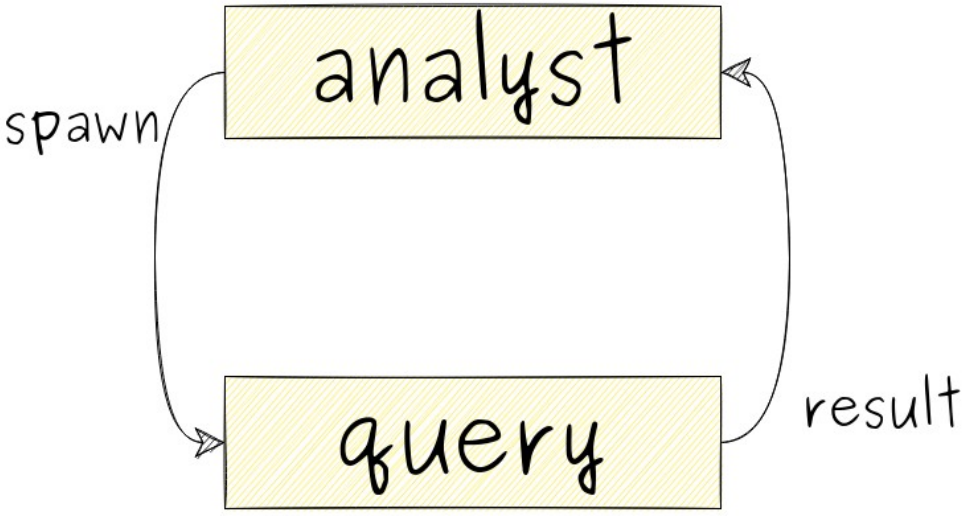

```
send(
  pid_of_b,
  some_message
)
```

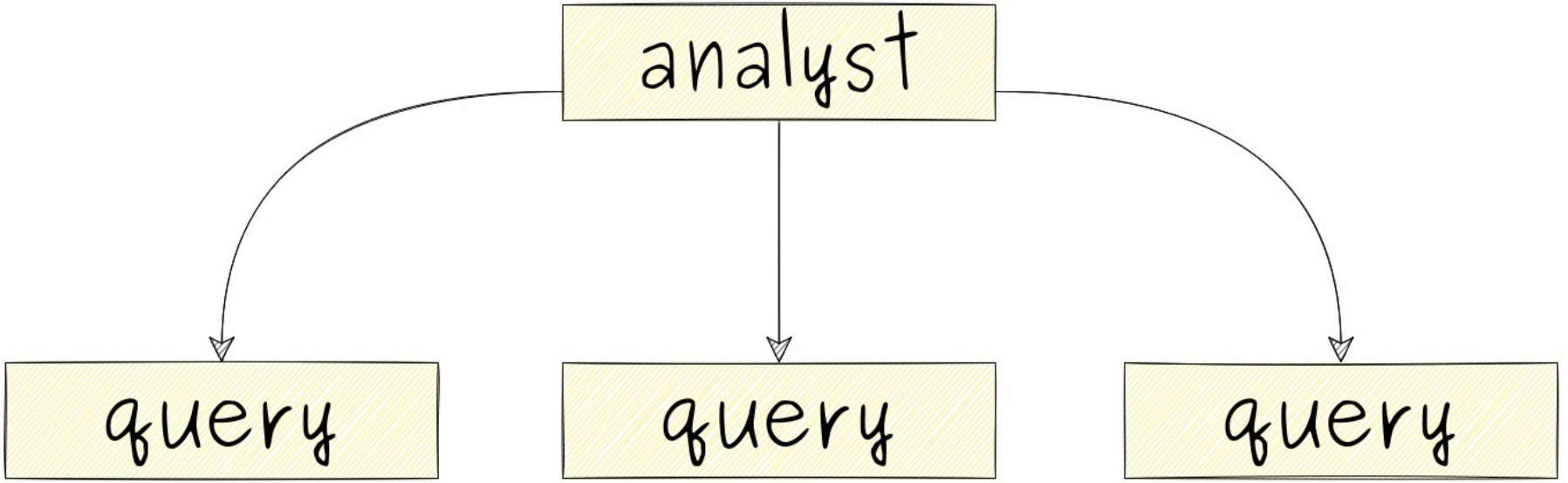
```
receive do
  message ->
    handle(message)
end
```


BEAM
scheduler
scheduler
scheduler
scheduler
CPU
CPU
CPU
CPU

long running task


query
anonymized result
app
fetch
data
database


analyst
spawn
query
result


analyst
query
query
query

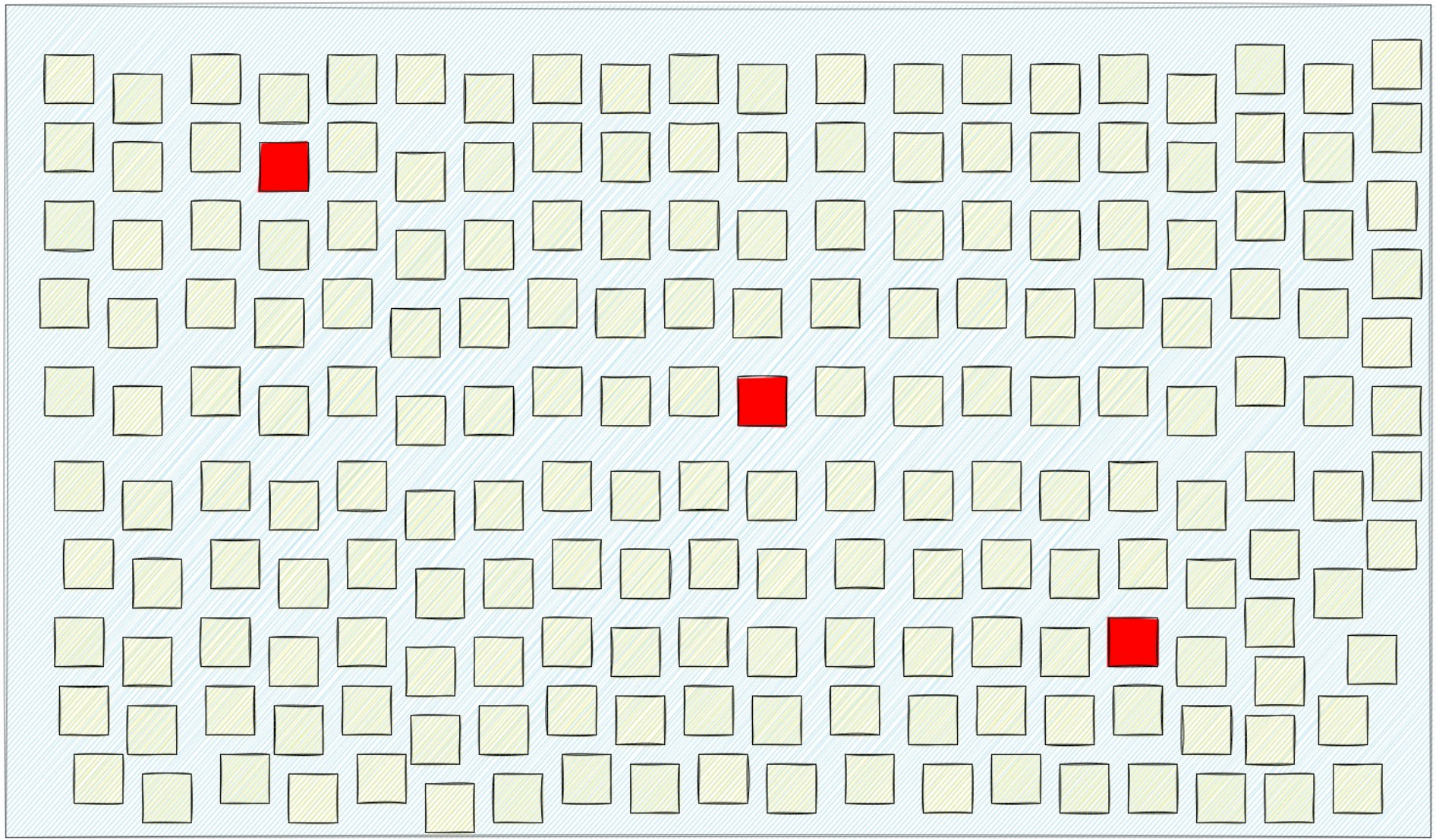
```
spawn(fn ->
  result = run_query(query)
  send(analyst_pid, result)
end)
```

```
def loop(state) do
  receive do
    message ->
      new_state = handle(state, message)
      loop(new_state)
  end
end
```

```
receive do
  {:run_query, query} ->
    query_pid = spawn(fn -> ... end)
    new_state = store(state, query_pid)
    loop(new_state)

  ...
end
```

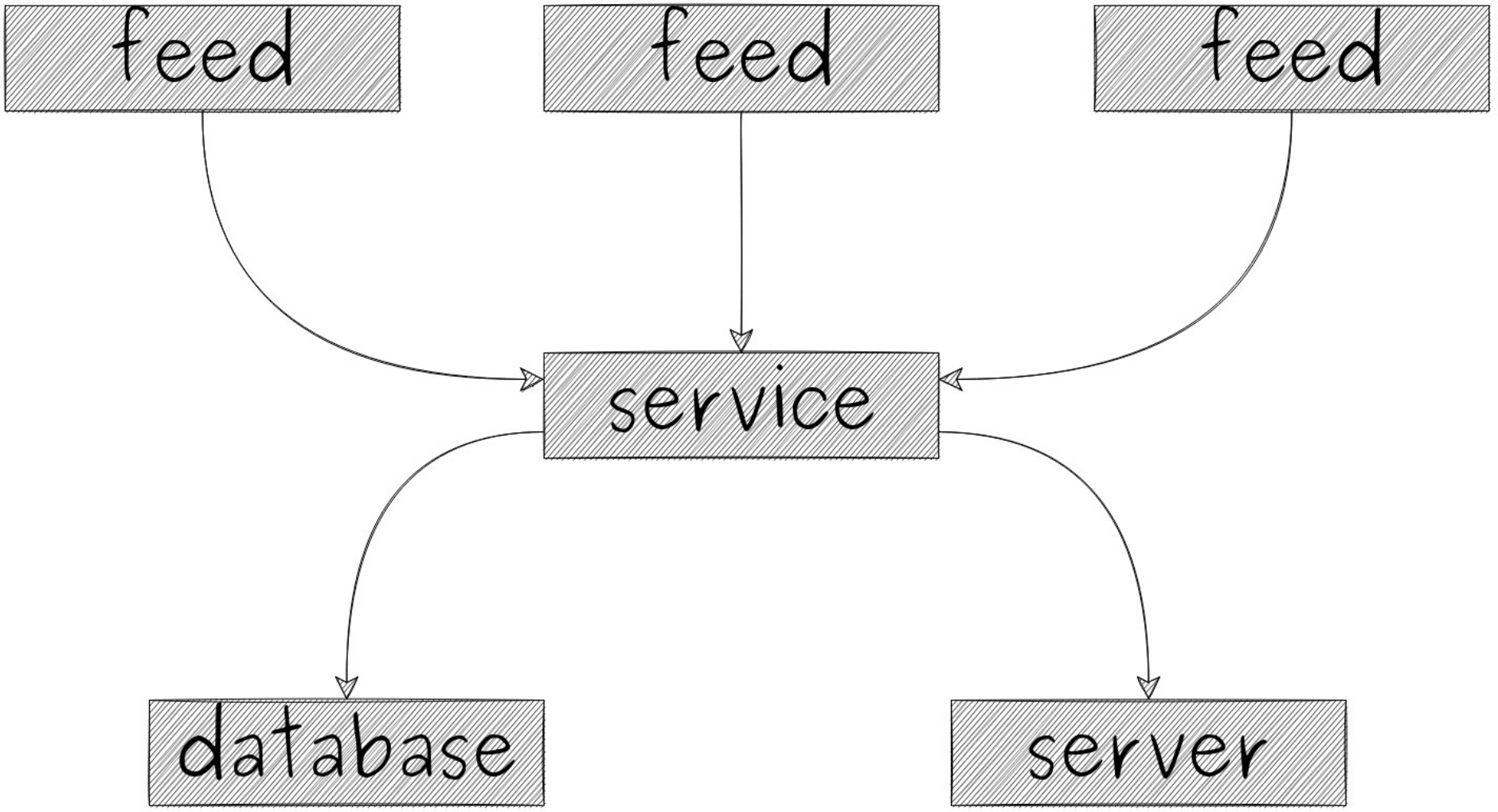
```
receive do
  {:query_result, query_pid, result} ->
    report_result(result)
    new_state = remove(state, query_pid)
    loop(new_state)

  ...
end
```

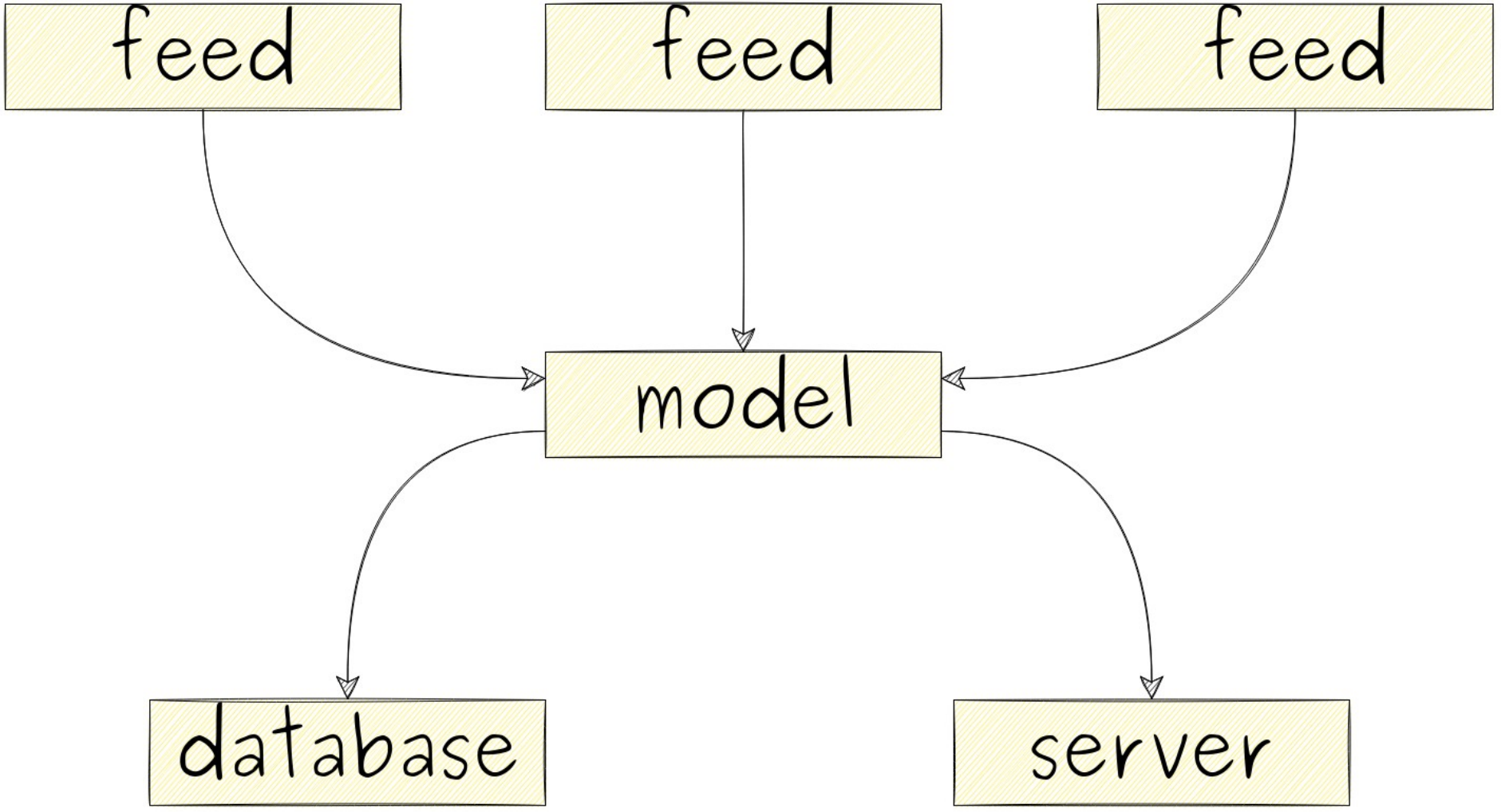
handling crashes

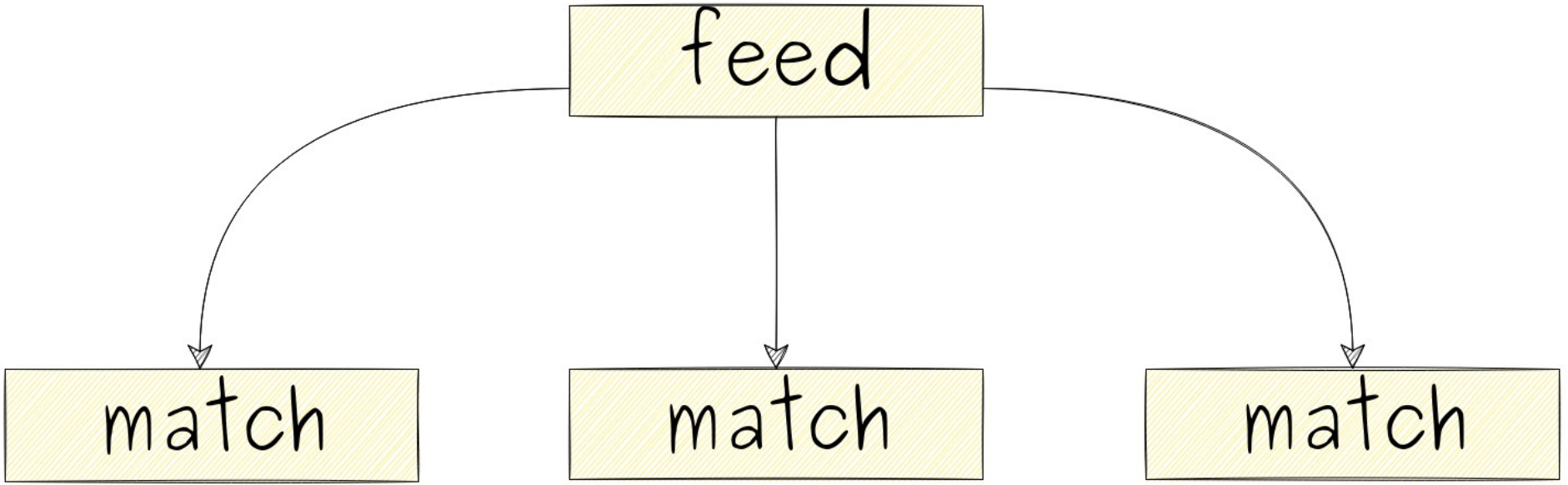


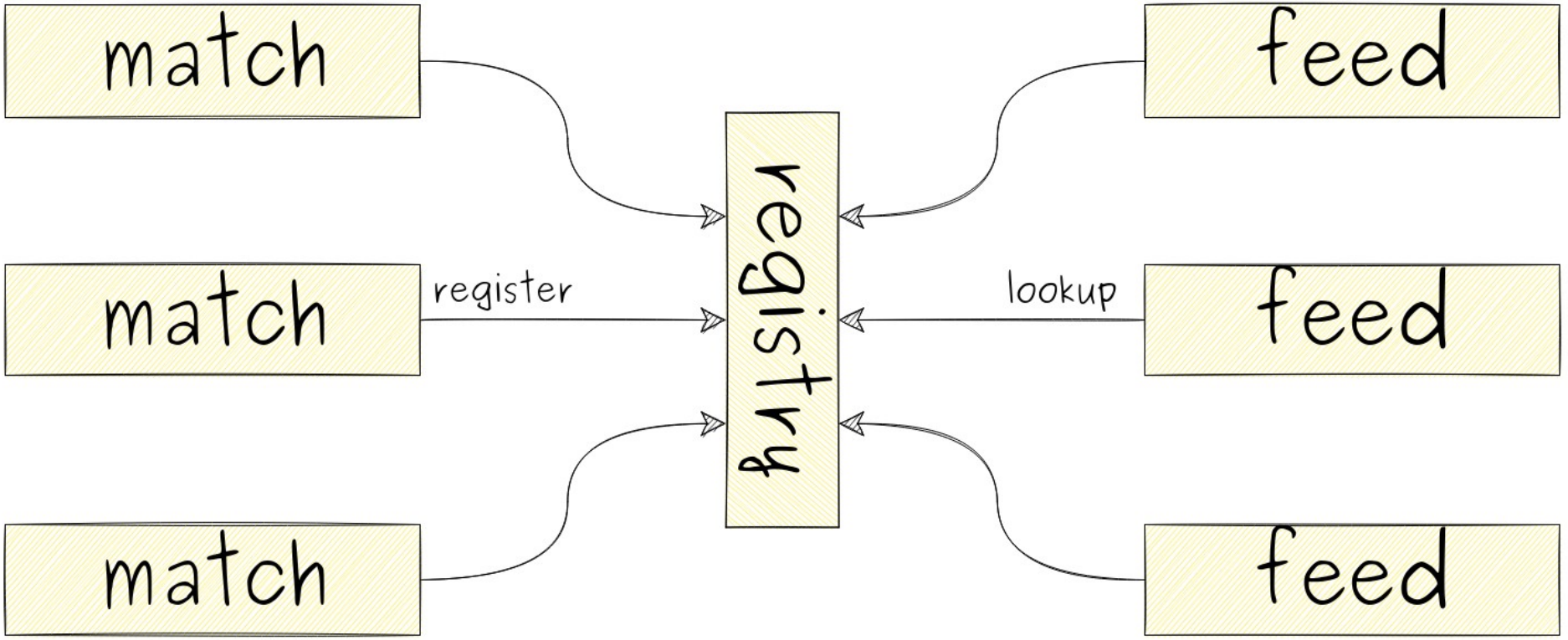
```
receive do
  {:EXIT, query_pid, reason} ->
    if reason != :normal,
      do: report_crash(reason)

    new_state = remove(state, query_pid)
    loop(new_state)

  ...
end
```

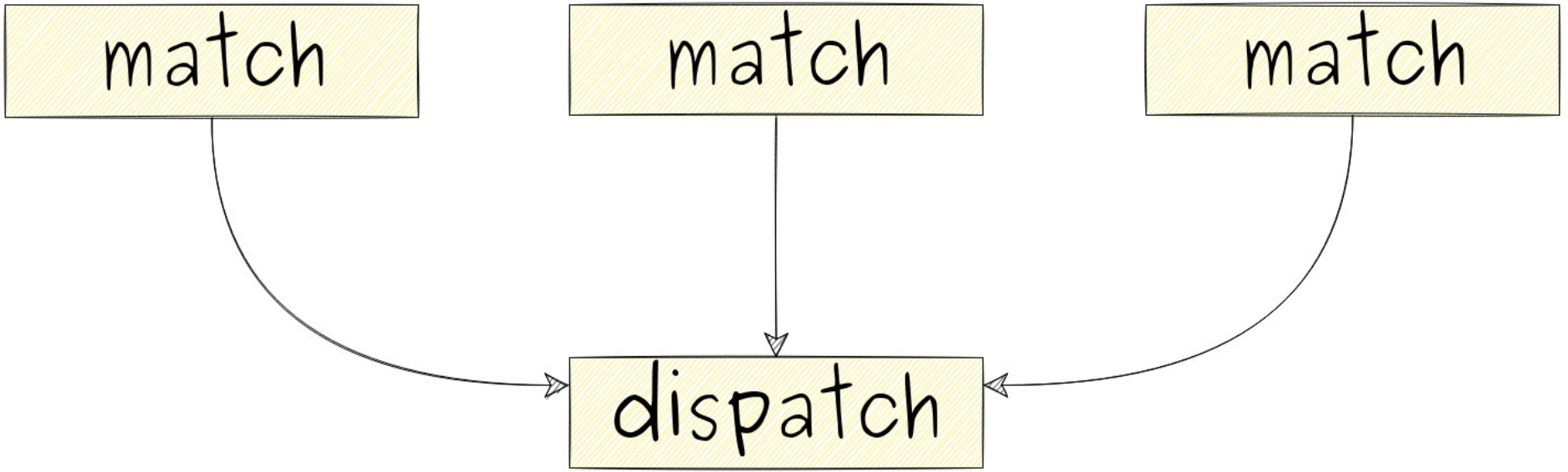
processing pipeline

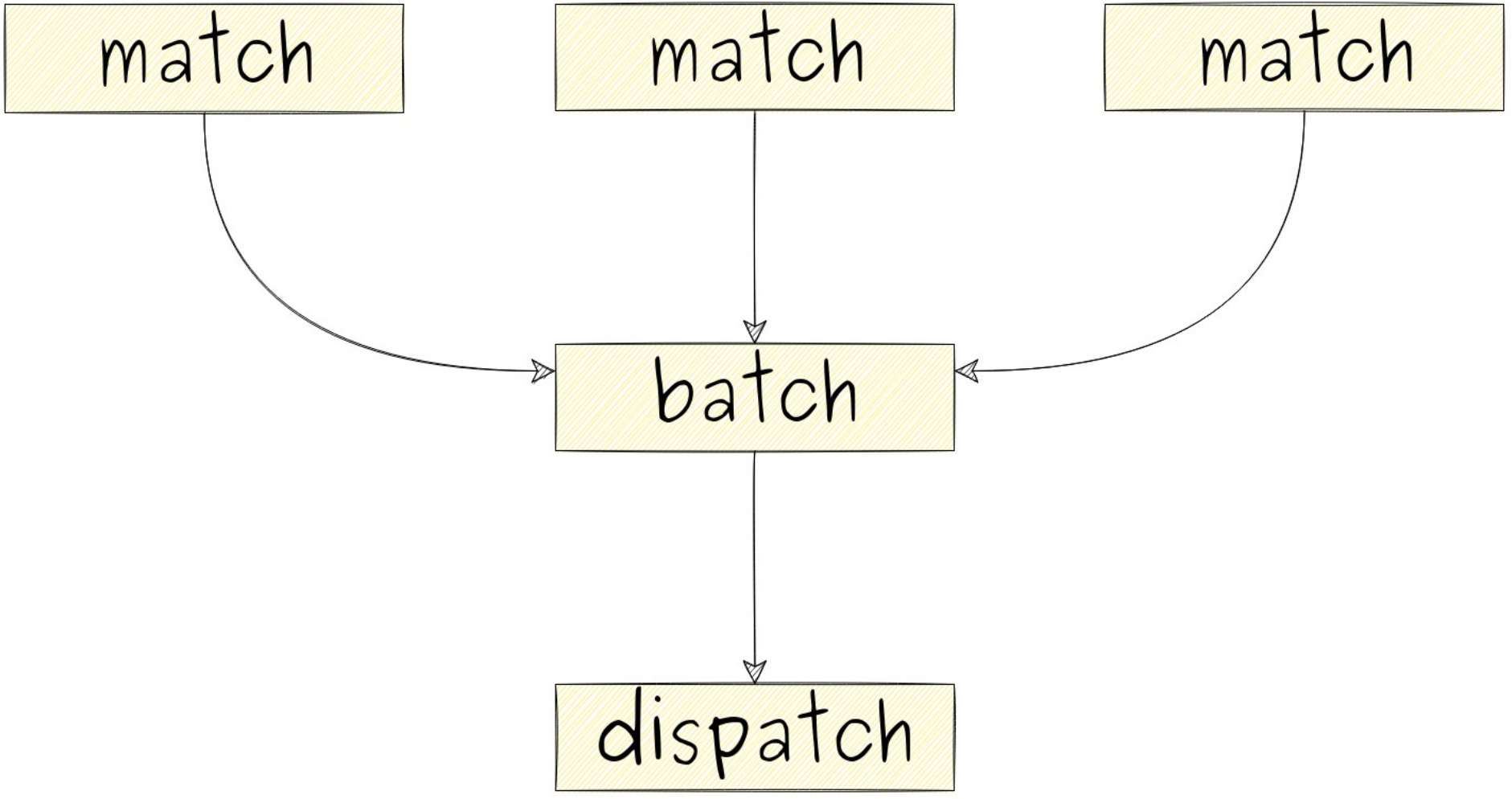

feed
feed
feed
service
database
server

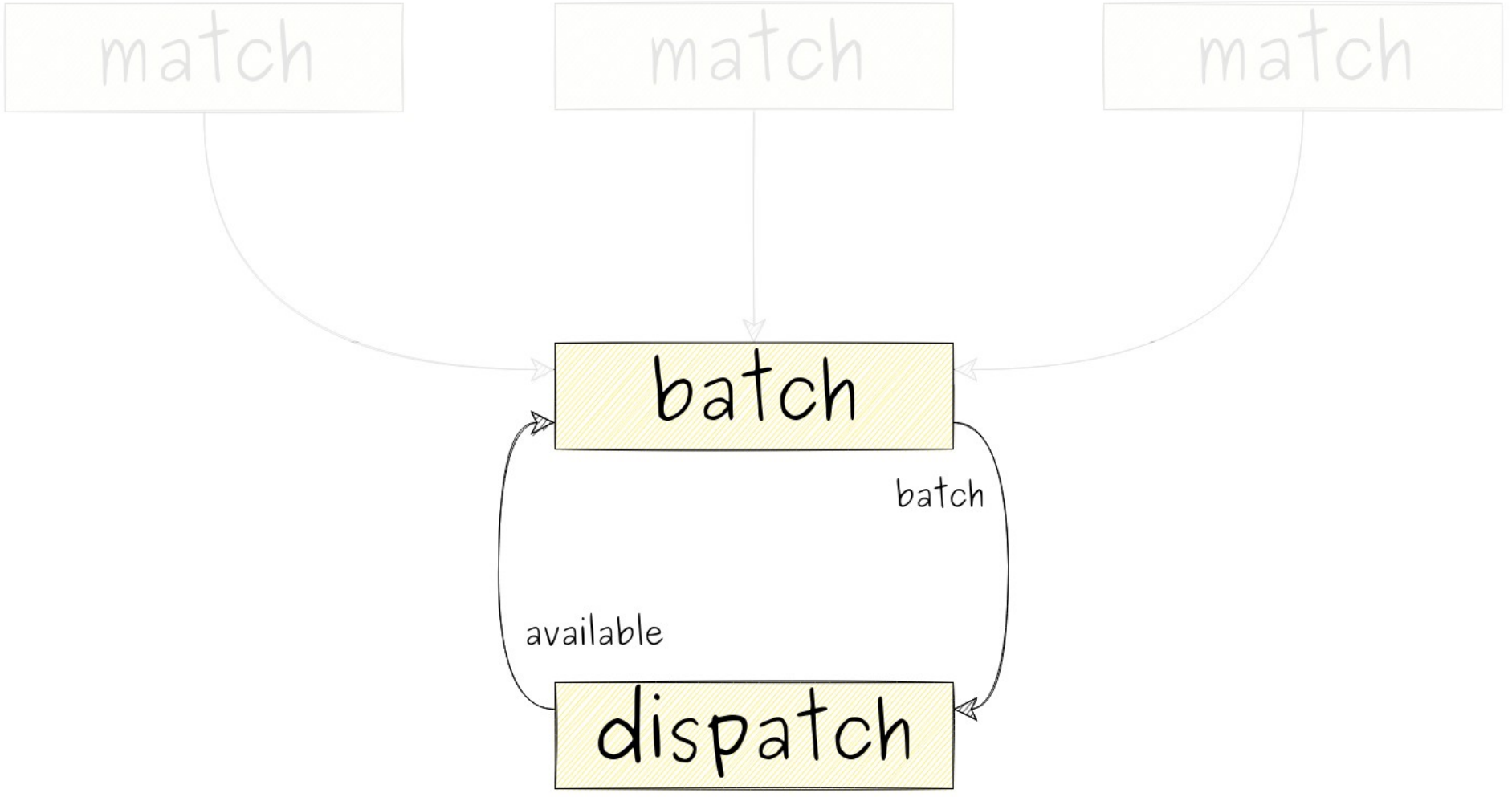

feed
feed
feed
model
database
server


feed
match
match
match


match
match
match
register
registry
lookup
feed
feed
feed


match
match
match
dispatch


match
match
match
batch
dispatch


match
match
match
batch
batch
available
dispatch

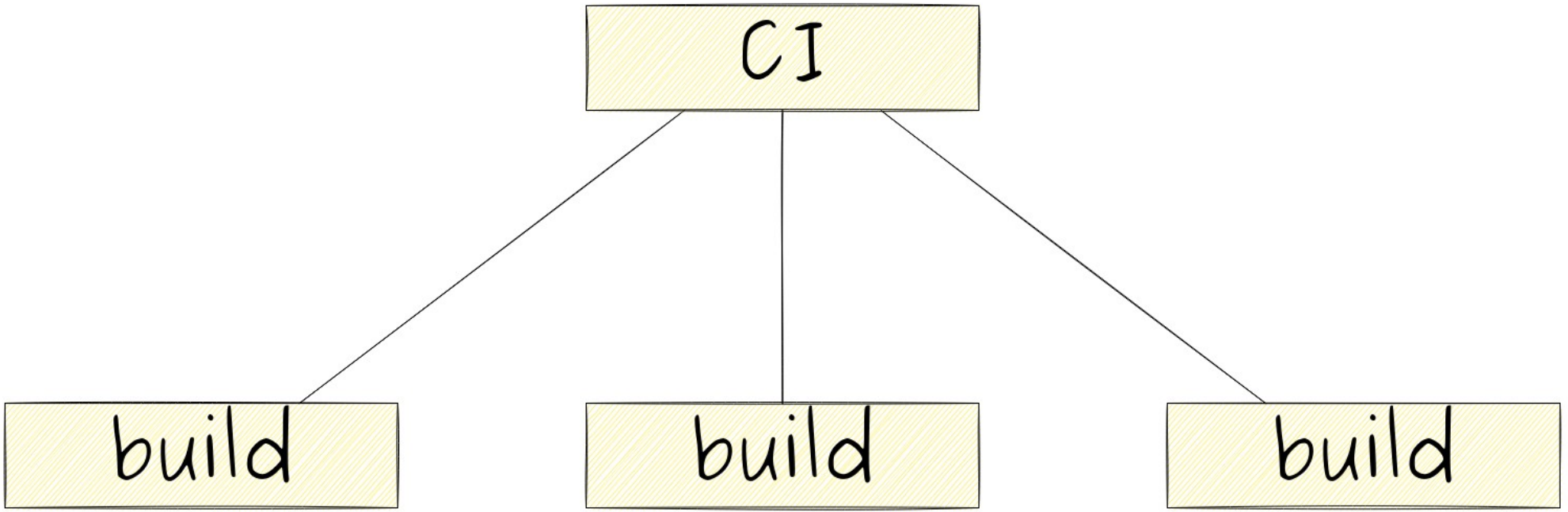
cancellation

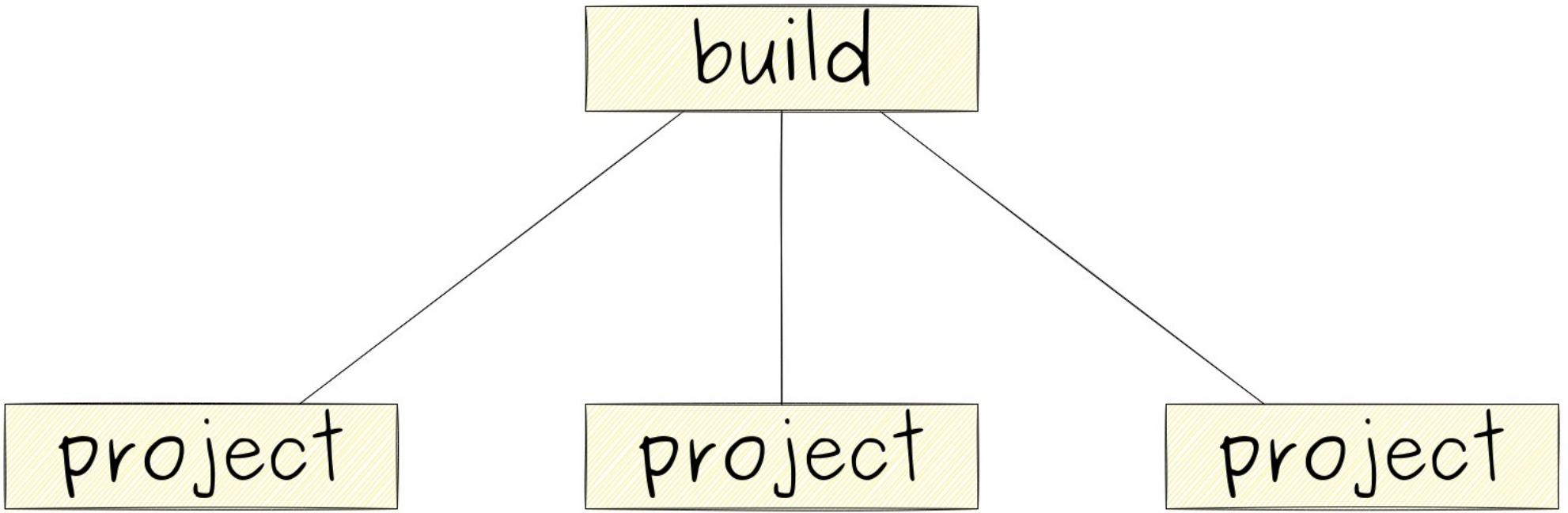
```
receive do
  {:cancel, query_id} ->
    query_pid = query_pid(state, query_id)
    Process.exit(query_pid, :shutdown)
    ...
end
```

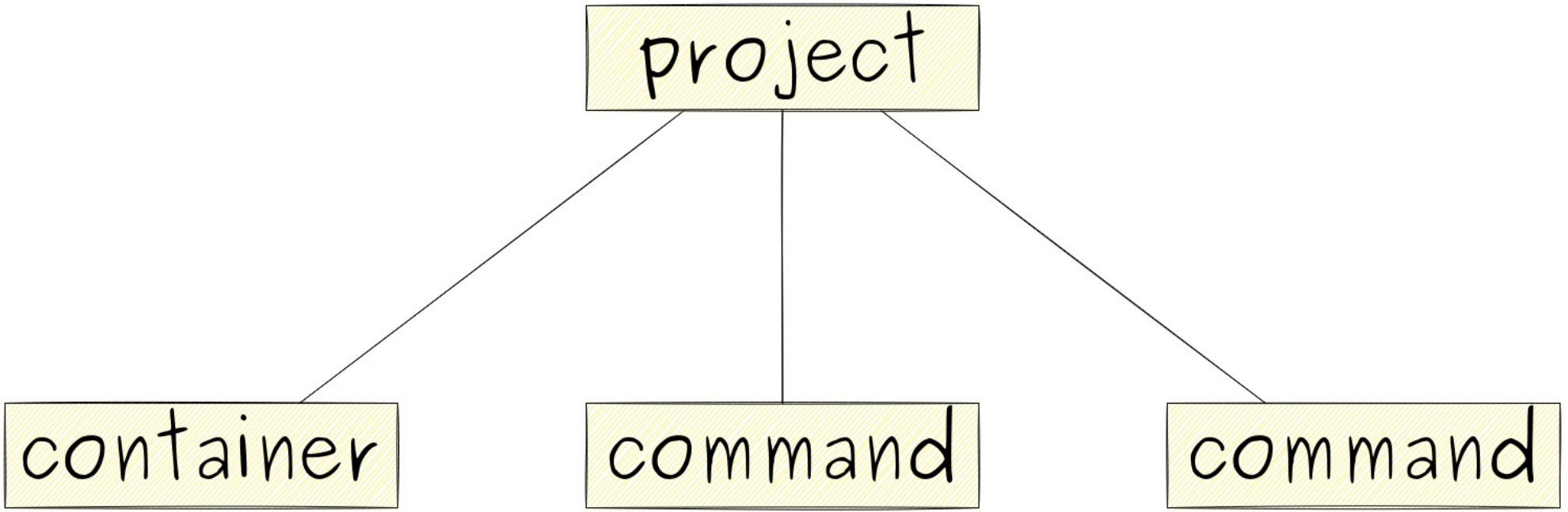
```
receive do
  {:EXIT, query_pid, reason} ->
    case reason do
      :shutdown -> report_cancelled(...)
      ...
    end

    new_state = remove(state, query_pid)
    loop(new_state)

  ...
end
```

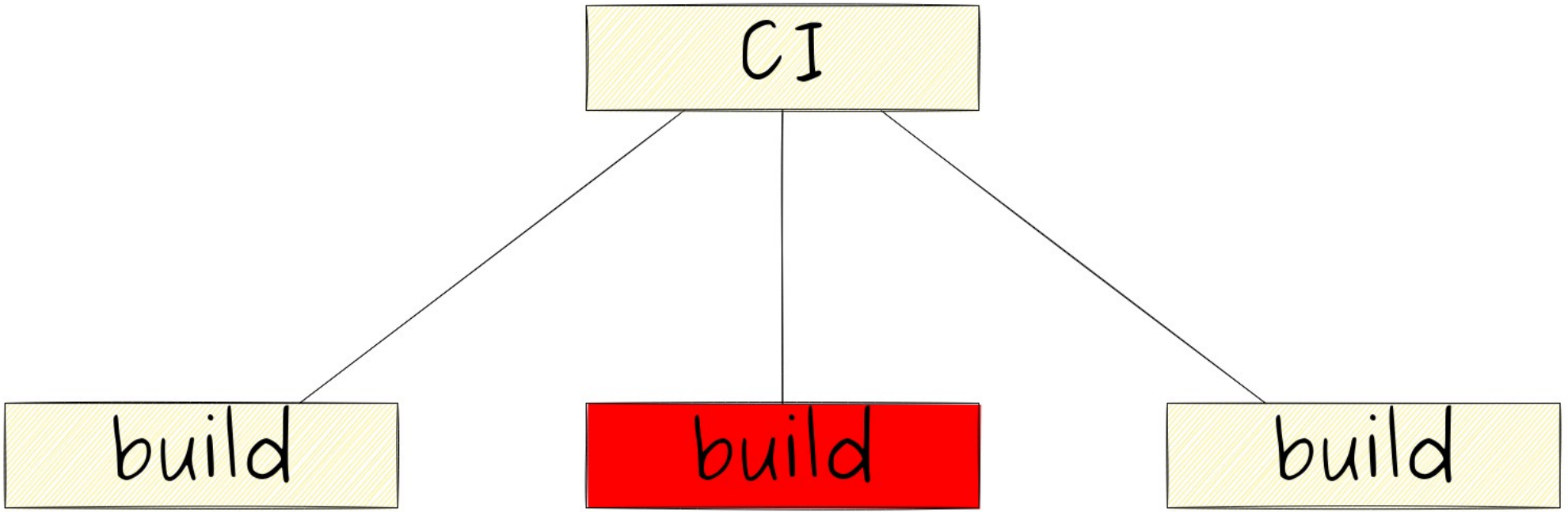
```
Process.flag(:trap_exit, true)
```

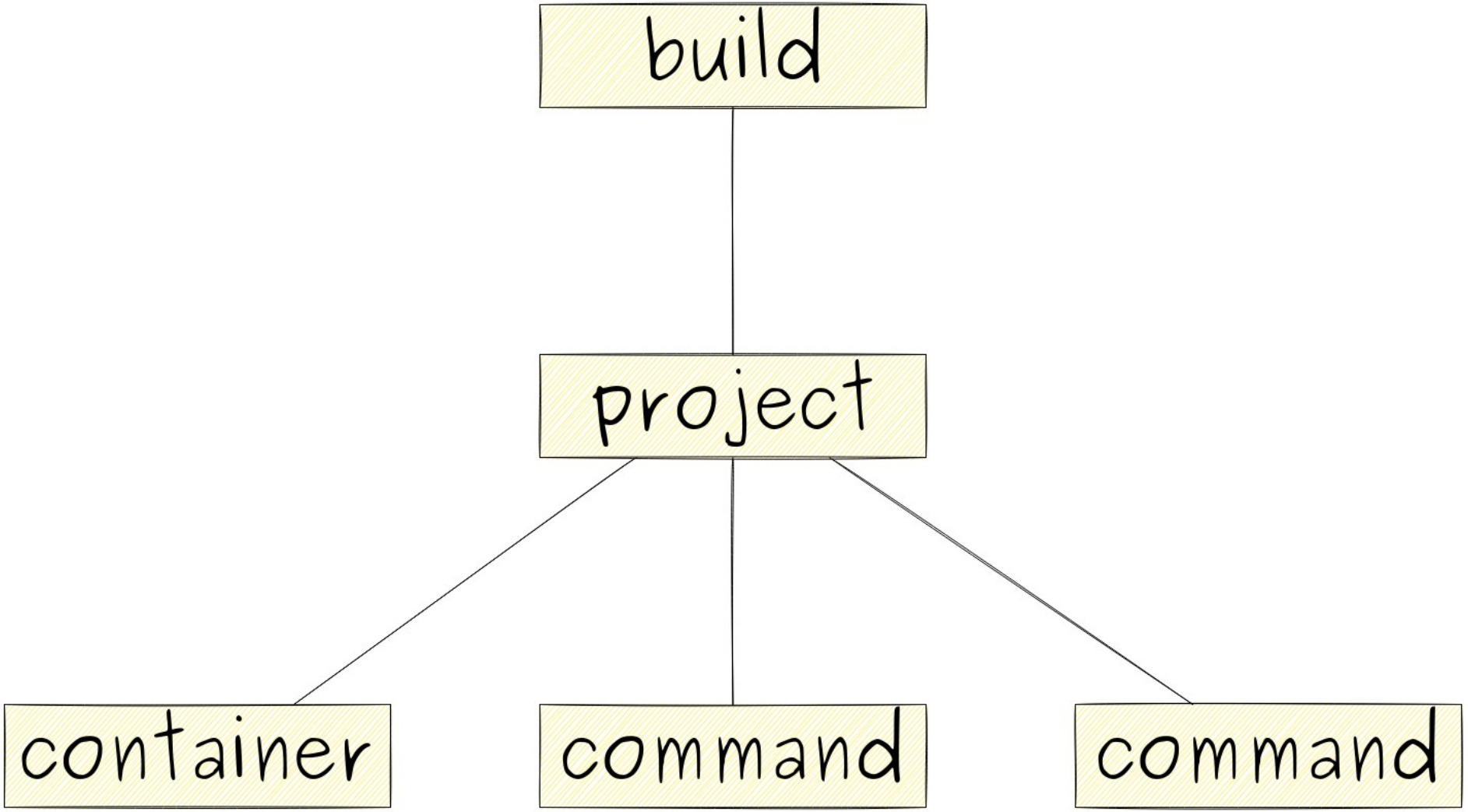
```
receive do
  {:EXIT, from_pid, reason} ->
    cleanup(...)
    exit(reason)

  ...
end
```

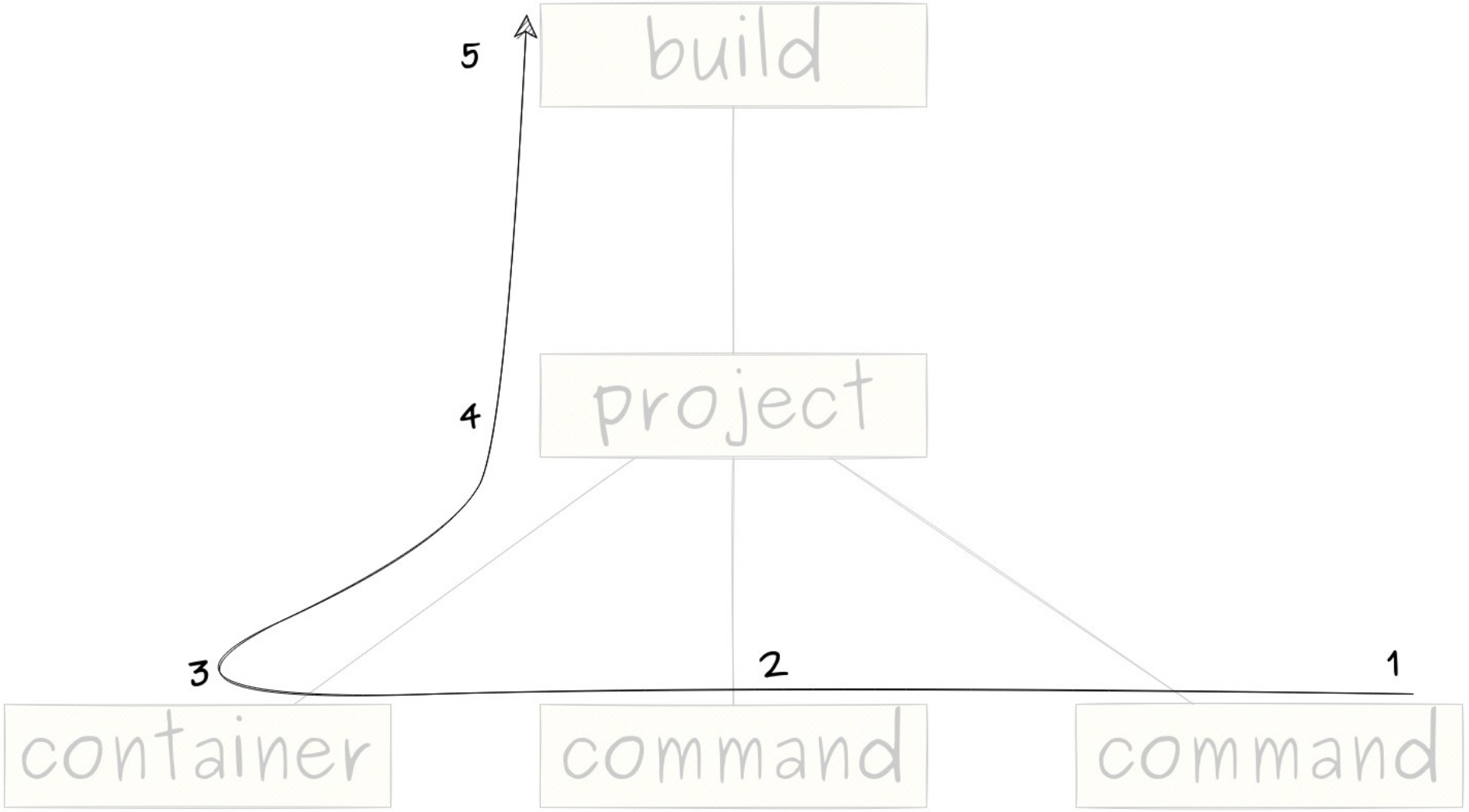

CI
build
build
build


build
project
project
project


project
container
command
command


CI
build
build
build


build
project
container
command
command


build
project
container
command
command
1
2
3
4
5

```
Process.exit(build_pid, :shutdown)

receive do
  {:EXIT, ^build_pid, _reason} ->
    ...
end
```

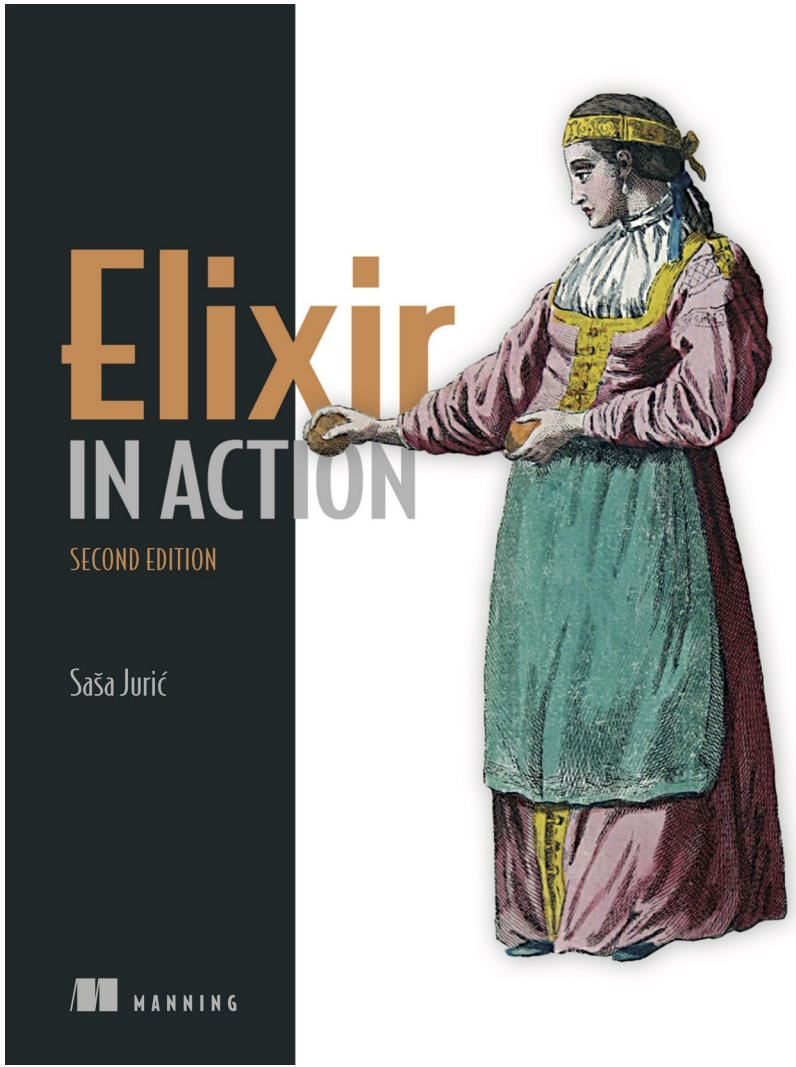
async task execution
processing pipeline
cancellation

YOW! Lambda Jam 2020

YOW! Lambda Jam 2020 - Saša Juric´- Simplifying Systems with Elixir

CHICAGO

The Soul of Erlang and Elixir • Sasa Juric • GOTO 2019



35% off using code
ctwyowsyd22
at manning.com

@sasajuric@hachyderm.io