

lunatic

Writing Rust the Erlang way

Bernard Kolobara, 19/7/2022

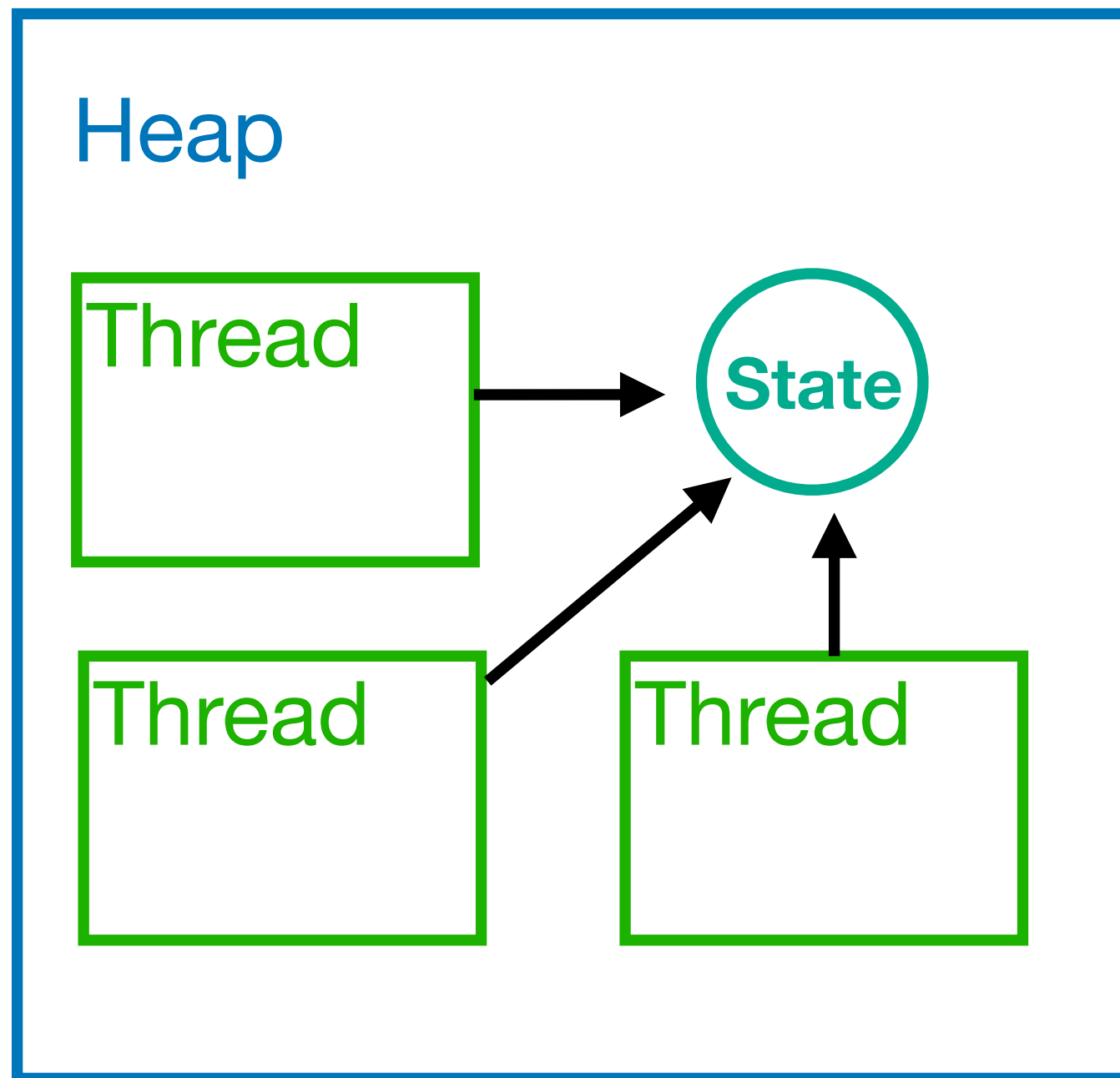
Agenda

1. Why Erlang?
2. How?
3. Demo app

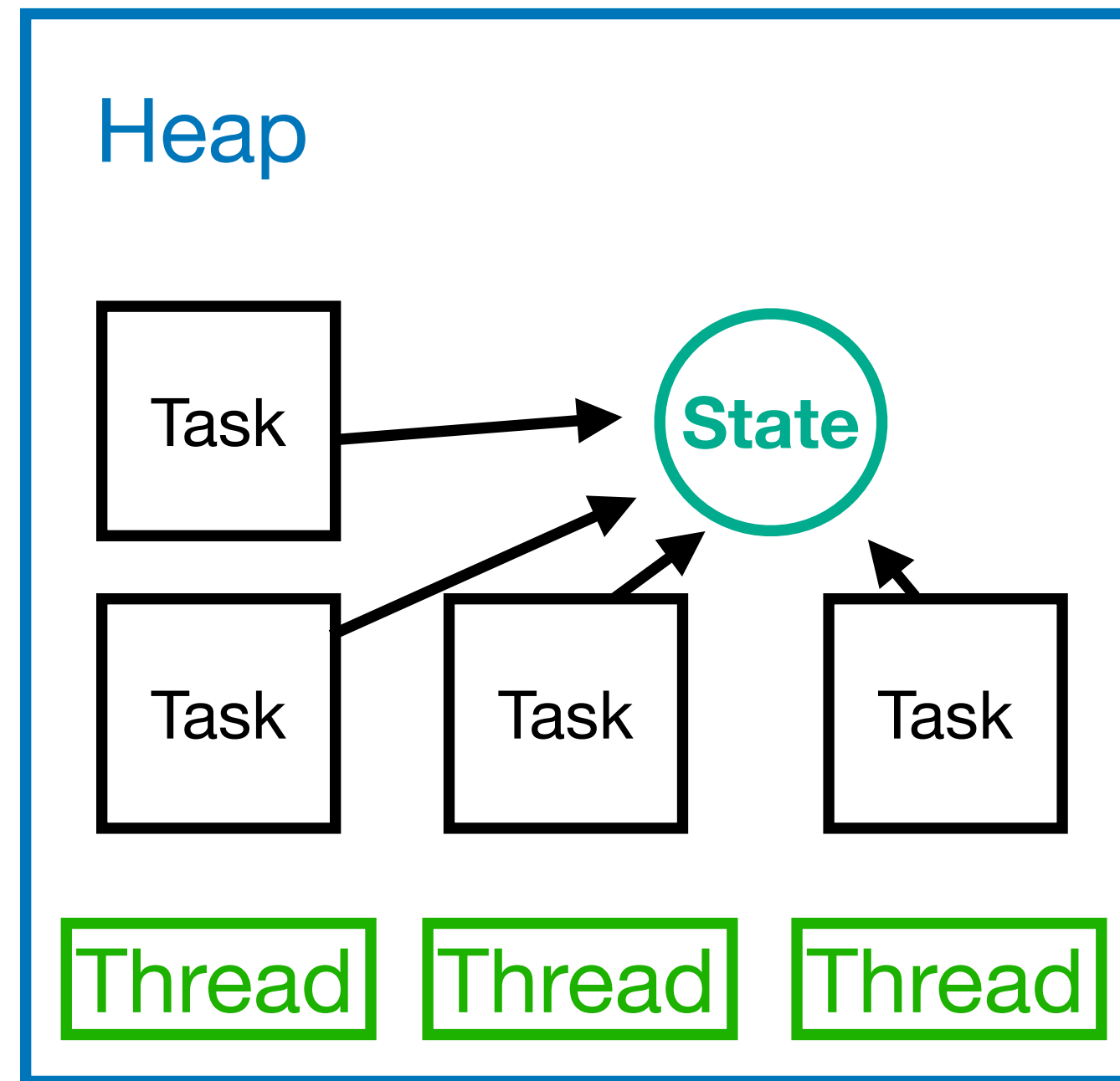
Why Erlang?

1. Concurrency model
2. Fault tolerance
3. Soft realtime

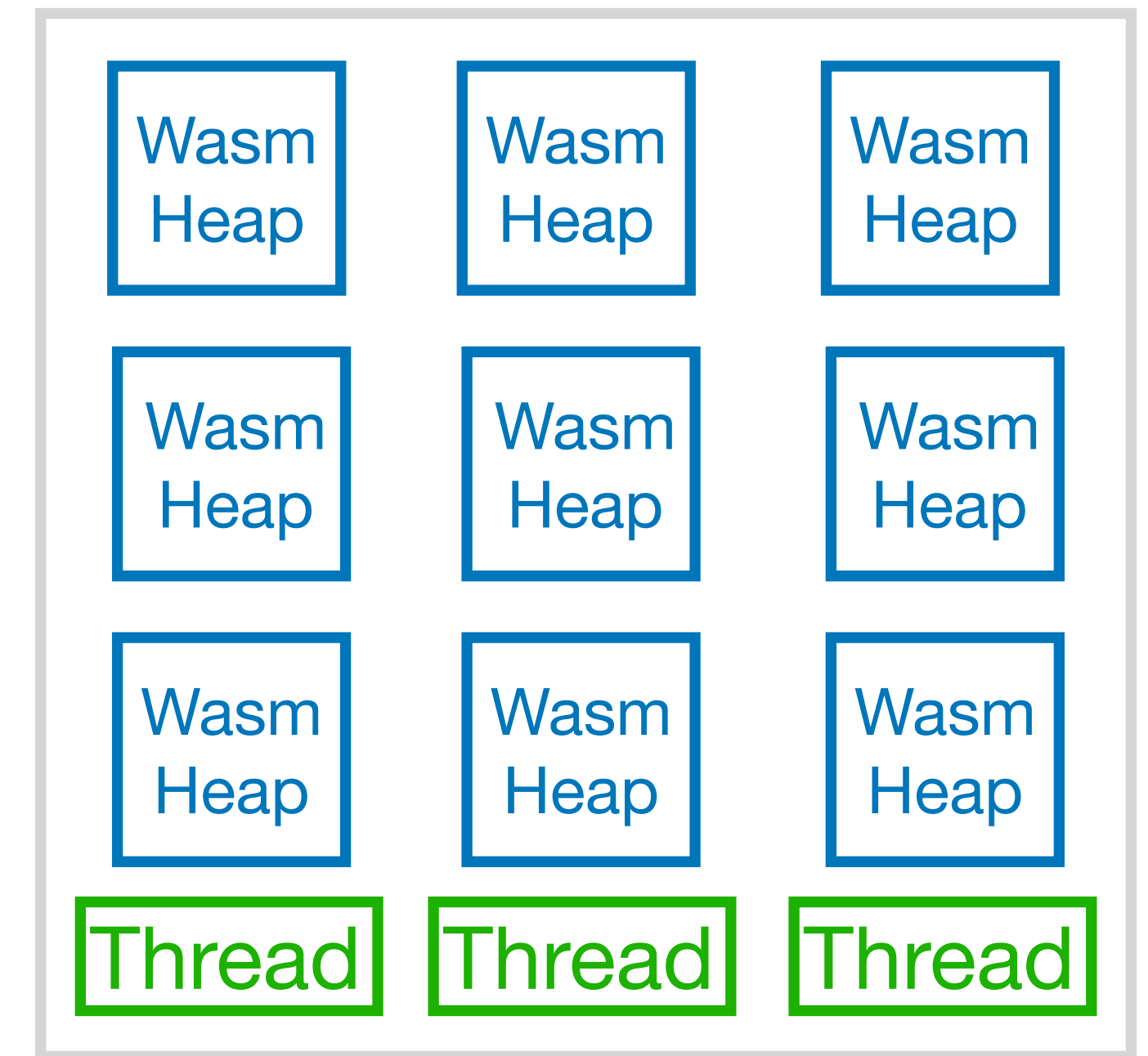
Rust



async Rust



lunatic Rust



Why Erlang?

1. Concurrency model
2. Fault tolerance
3. Soft realtime

The AXD301 has achieved a NINE nines reliability

Let's put this in context: 5 nines is reckoned to be good (5.2 minutes of downtime/year). 7 nines almost unachievable ... but we did 9.

Why is this? No shared state, plus a sophisticated error recovery model.

~ Joe Armstrong (co-designer of Erlang)

A man with curly hair, wearing a white t-shirt with a red graphic, is sitting at a desk in a cluttered office. He is holding a white corded telephone receiver to his ear with his left hand. The desk is covered with various items, including a blue cup, papers, and a small figurine. In the background, there are cardboard boxes, a lamp, and a bookshelf. The overall scene is a typical office environment from the early 2000s.

IT DEPARTMENT

**HAVE YOU TRIED TURNING IT OFF
AND ON AGAIN?**

Why Erlang?

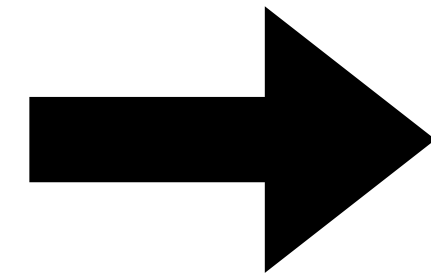
1. Concurrency model
2. Fault tolerance
3. Soft realtime

*“Tasks must not perform
computation heavy logic or
they will prevent other tasks
from executing.”*

How?

1. Compile the Rust app to WebAssembly
2. Spawn WebAssembly instances for each new process
3. Insert preemption points in-between WebAssembly instruction

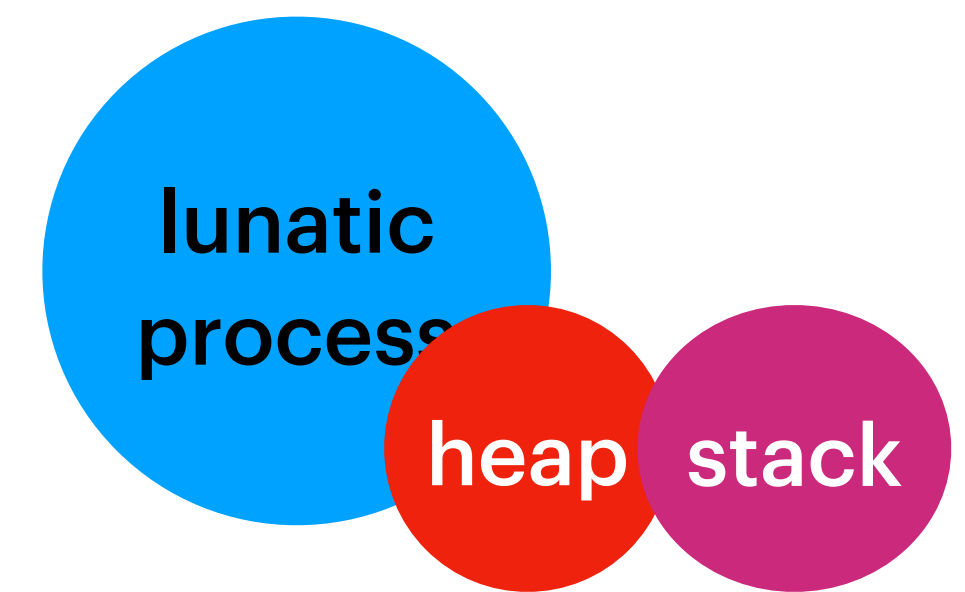
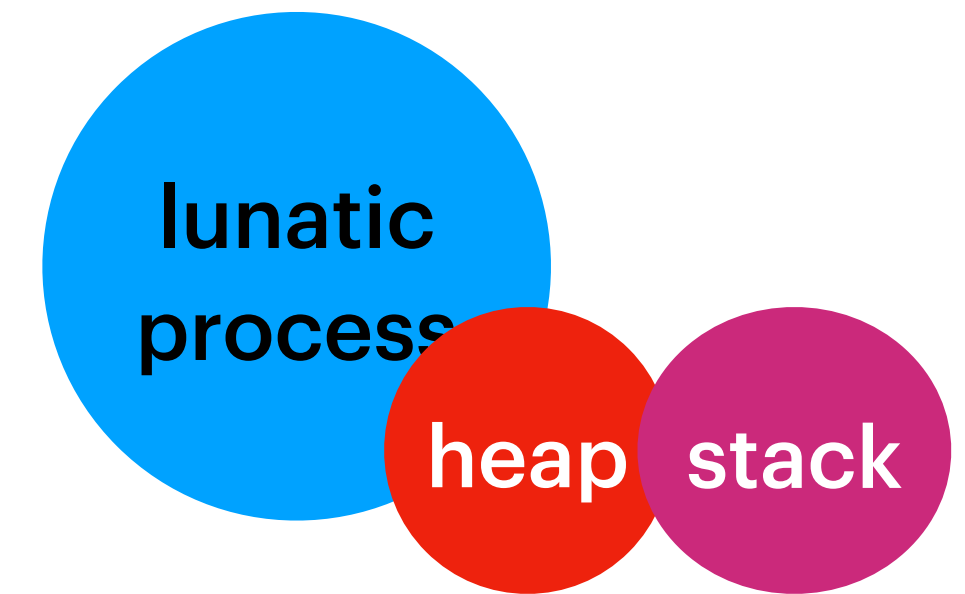
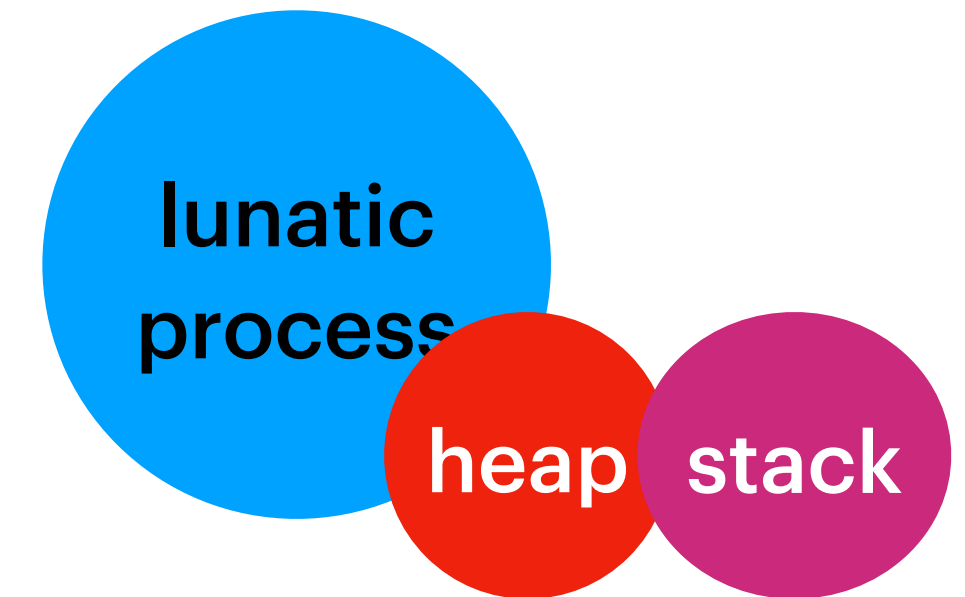
How?



compile



spawn



Demo