Crashing for Reliability

Ihor Kuz

seL4 summit 2023

Computer Doesn’t Work. What Do You Do?

HAVE YOU TRIED
TURNING IT OFF AND ON AGAIN?

Challenges

Why does Restart Work?

- Erroneous State
  - Some state making system misbehave
  - Not sure what specifically or how to make state good again
  - Restart -> Clean Slate: known good state
  - *Tada!* It works
- Problem
  - Drastic approach
  - Large downtime
  - Lose good working state
  - Might not fix the erroneous state

Restart in Resilient Systems

PROTECTION

RESILIENCE

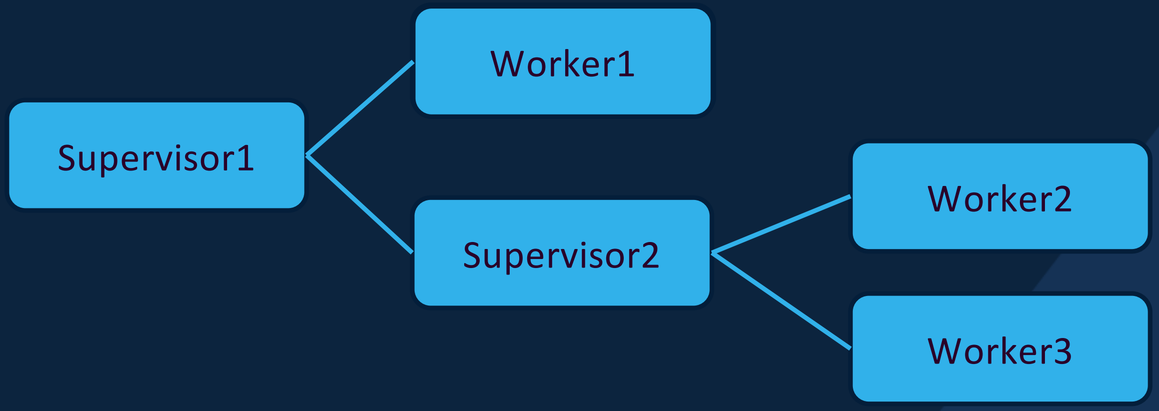
LIFECYCLE

We want Clean Slate, but:

- Reduce downtime
- Reduce loss of non-erroneous state
- Deal with persistent failures

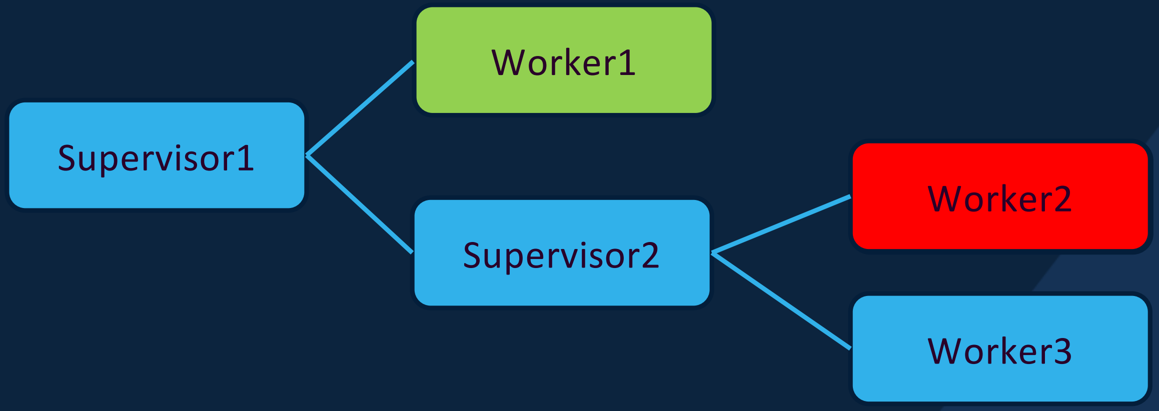
Erlang/OTP and Let it Crash

- Erlang/OTP
  - Developing highly reliable systems at Ericsson
  - Erlang: functional language, actor concurrency
  - OTP: Platform - Design principles, libraries, tools
- Supervision Trees



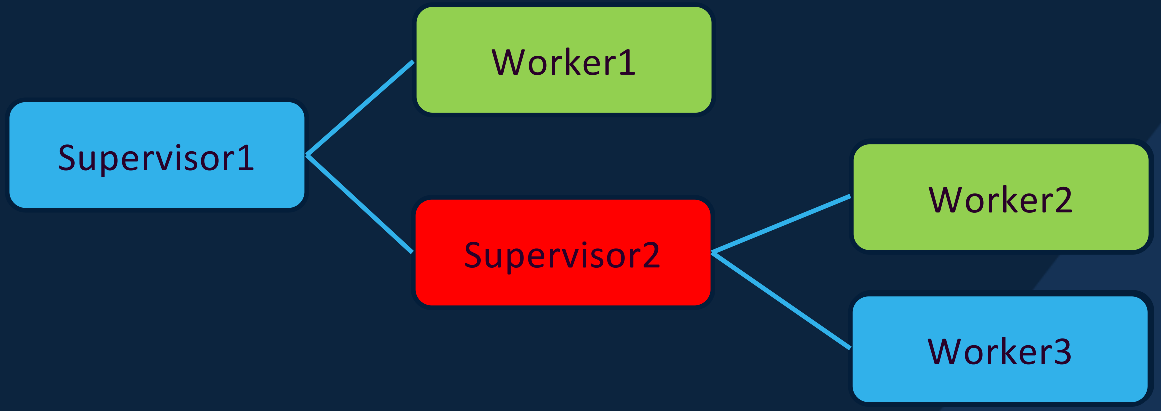
Erlang/OTP and Let it Crash

- Erlang/OTP
  - Developing highly reliable systems at Ericsson
  - Erlang: functional language, actor concurrency
  - OTP: Platform - Design principles, libraries, tools
- Supervision Trees



Erlang/OTP and Let it Crash

- Erlang/OTP
  - Developing highly reliable systems at Ericsson
  - Erlang: functional language, actor concurrency
  - OTP: Platform - Design principles, libraries, tools
- Supervision Trees



Why does Let it Crash Work?

- Distributed architecture
  - many communicating processes
- Processes isolation
  - processes don’t share resources
- Explicit communication
  - transparently reuse new connections
- Stateless processes
  - functional, immutable data structures
  - OTP design patterns: split into stateful and non-stateful processes

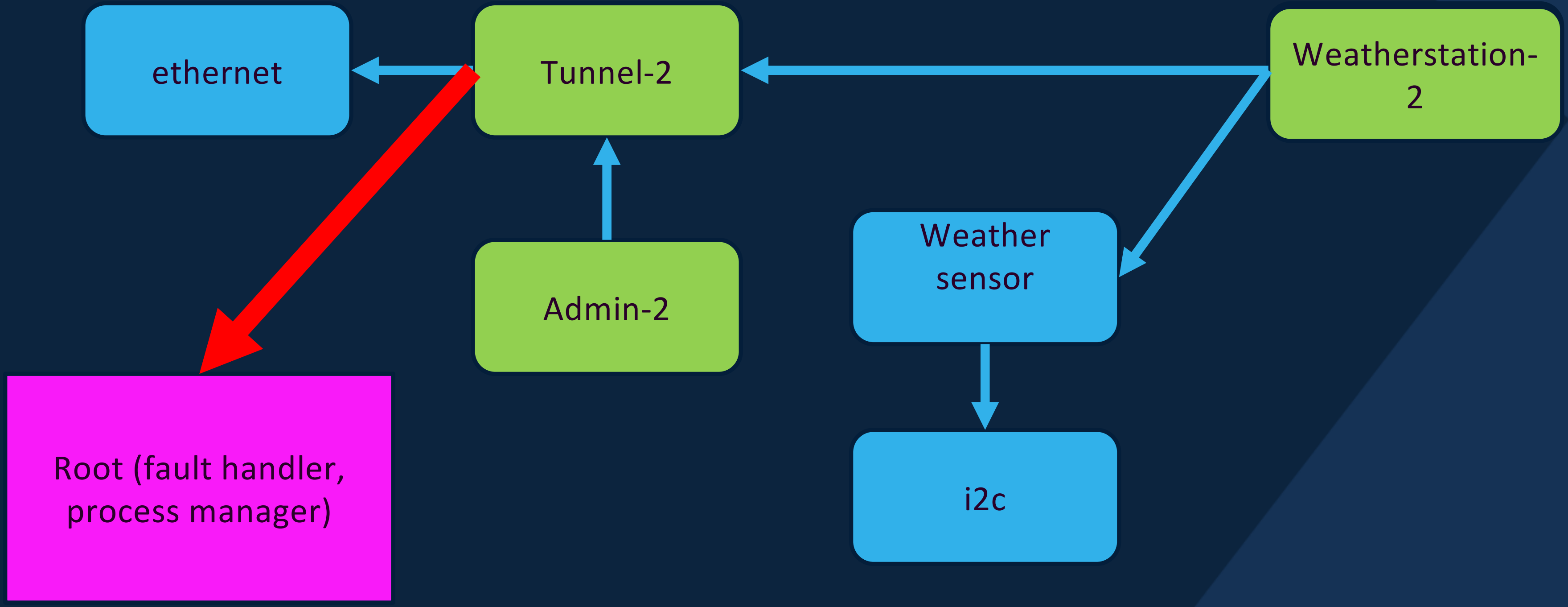
Let it Crash and seL4

- A good Match?
  - Distributed architecture ✓
  - Process isolation ✓
  - Explicit communication ✓

- Basic requirements
  - Detect fault ✓
  - Tear down process ✓
  - Start new process ✓

- Challenges
  - seL4 systems often static
    - How to restart processes? How to reset communication channels?
  - Reset non-software state
    - E.g. hardware for drivers
  - Stateless processes

The Kry10 OS Approach



Challenge: To Crash or Not To Crash?

When is it ***not*** useful to crash a component?

- Reasons
  - Component can recover from error (no need to crash)
  - Failure not caused by component state (e.g. hardware)
  - Failure not caused by erroneous state (correct state causes fault)
  - Component contains critical state (can’t afford to lose that state)
  - Erroneous state is persistent (still there after restart)
- Error Kernel
  - Set of components with critical state
  - Goal: Reduce size of error kernel
  - Design: split into components that store state vs computational components

Challenge: Dealing with Persistent Failure

What to do if a component keeps failing?

Heuristics

- Restart count
- Increase restart domain
- Sanitise persistent state
- Reinstall, revert, update component code
- Give up…

Conclusion

- Restart -> Clean Slate
- Erlang/OTP Let it Crash works as fine grained restart
- Apply to seL4 and Kry10 OS!
- Challenges
    - Crashing isn’t always possible -> reduce error kernel
    - Persistent failures -> heuristics, eventually give up