Defensive Programming vs. Let It Crash

Lou Xun aquarhead@ela.build / Ela Workshop

A critical bug happened...

- Resources exhausted
- Users disconnect, system becomes laggy
- Cascading failure
- Loads of log, errors everywhere
- "What the hell is the *root cause*?"

You started debugging...

- Follow the stacktrace
- Looks like it's some cleanup code
- Something is wrong in the try block (without log!)
- And you failed to recover it in the catch block

What if you had...

- The exact location where things *start* to fail
- The exact "context" at there
  - local variables, global variables
  - object state
  - etc..
- Essential informations are very easy to lose

You're defensive, because...

- If you don't `catch`, a `raise` will propagate to the top...
- and crash *everything*
- You write code that you wish *never* get executed in production
- You're hiding a broken *state* in the system

You program defensively
because you cannot simply

Let It Crash

Lou Xun aquarhead@ela.build / Ela Workshop

Erlang's Process

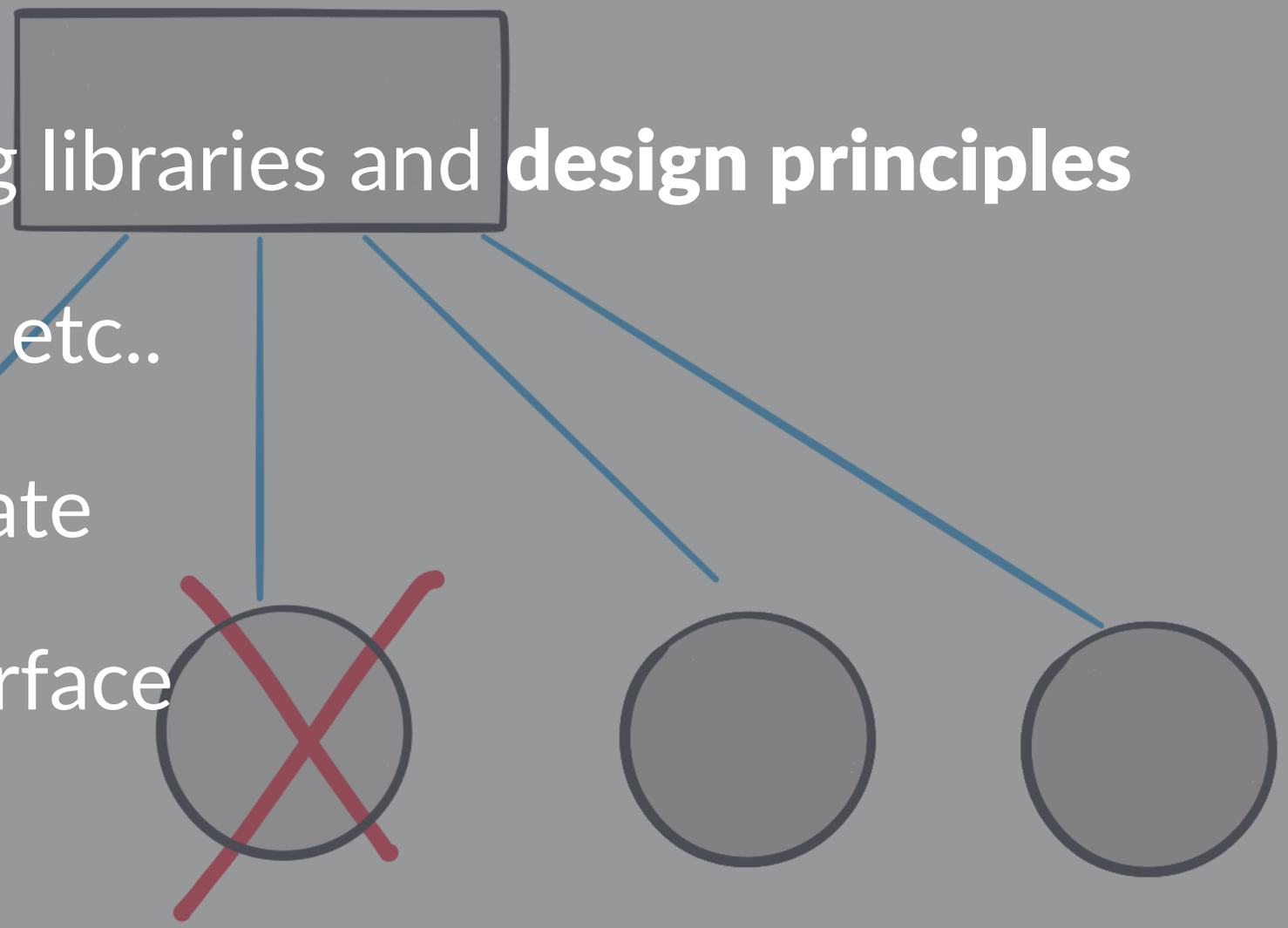
- Isolated
  - one does *not* kill all
- Share nothing
  - less context to be concerned
- Lightweight
  - use as many as you need
  - not as many as you can

Lou Xun aquarhead@ela.build / Ela Workshop

Other ingredients

- A functional language
  - much less state
- Message passing
  - async operation
- Monitor & Link
  - handle process down, fail as a unit
- Preemptive scheduling, Distribution

Erlang/**OTP**



- A set of Erlang libraries and **design principles**
- `gen_server` etc..
  - maintain state
  - unified interface
- `supervisor`
  - sole job is *supervising* children
  - start, stop, monitor, restart

State & Restart

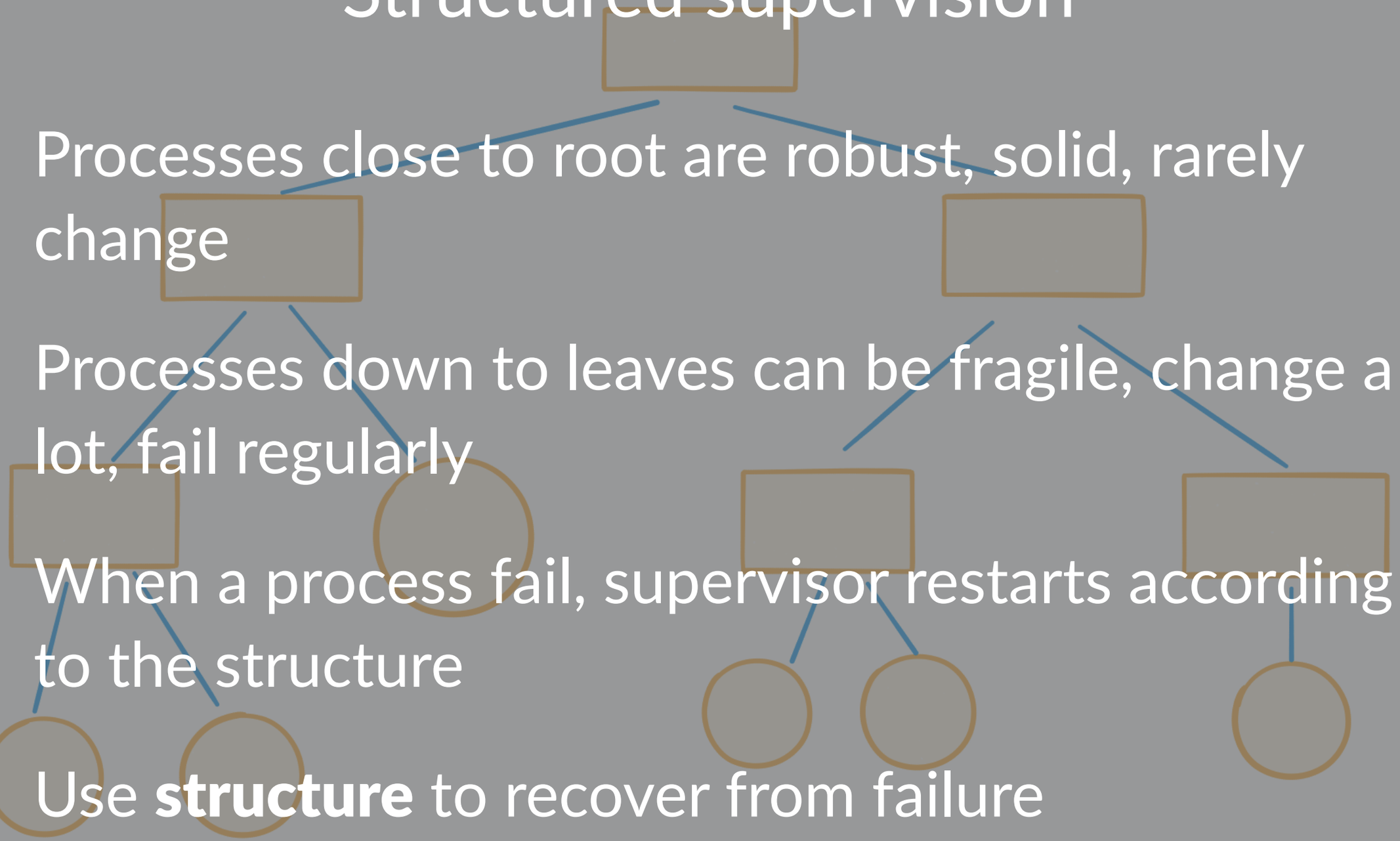
- **State**
- Why restart works?
    - What can restart fix?
        - Cannot fix everything (of course)
        - Clean state
- Isn't it the same as restarting my daemon?

Lou Xun aquarhead@ela.build / Ela Workshop

Supervision Tree

Lou Xun aquarhead@ela.build / Ela Workshop

Structured supervision



- Processes close to root are robust, solid, rarely change
- Processes down to leaves can be fragile, change a lot, fail regularly
- When a process fail, supervisor restarts according to the structure
- Use **structure** to recover from failure
- Not code

Lou Xun aquarhead@ela.build / Ela Workshop

Let It Crash

- Don't be afraid of failures, exceptions or crashes
- They are *tools* we can use
- No more defensive programming, only code as if everything goes right
- Or if you absolutely know how to deal with a certain failure
- Otherwise just **let it crash**

vs. Defensive Programming

- Supervision Tree vs. Nested `try..catch`
- Process vs. ?
- Self-healing in production
- Fail early, fail fast aids debugging
- Avoid writing unused error handling code
    - Productivity, readability

Thanks

http://ferd.ca/the-zen-of-erlang.html

Lou Xun aquarhead@ela.build / Ela Workshop