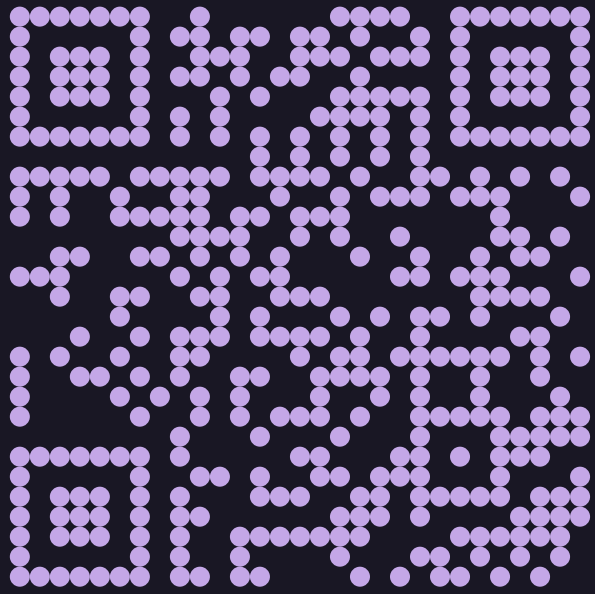
gleam (and the BEAM)

pacman

Slides available ↪ online.



https://computerbetrug.jetzt/olaf/gleam.pdf

gleam

- transpiles to erlang
- functional
- type safe
- written in rust

```
import gleam/io
import gleam/string_tree as st
pub fn main() -> Nil {
  st.from_string("Hello, ")
  |> st.append("oLaF!")
  |> st.to_string
  |> io.println
}
```

gleam (0)

- simple by design

```
pub type Person {
  Adult(name: String, age: Int, needs_glasses: Bool)
  Child(name: String)
}
```

```
fn twice(
  arg: value,
  func: fn(value) -> value
) -> value {
  func(func(arc))
}
```

gleam (1) use (0)

```
fn login(n: String, p: String) -> Result(String, Nil)
{ ... }
```

```
pub fn without_use() -> Result(String, Nil) {
  result.try(get_name(), fn(name) {
    result.try(get_passwd(), fn(passwd) {
      result.map(login(name, passwd), fn(greeting) {
        greeting <> ", " <> username
      })
    })
  })
}
```

gleam (2) use (1)

```
fn login(n: String, p: String) -> Result(String, Nil)
{ ... }
```

```
pub fn with_use() -> Result(String, Nil) {
  use name <- result.try(get_olaf_name())
  use passwd <- result.try(get_passwd())
  use greeting <- result.map(login(name, passwd))
  greeting <> ", " <> username
}
```

gleam (3) js

- can transpile to javascript
- ↳ lustre
- externs:

```
@external(javascript, "./my_package.mjs", "js_name")
pub fn example() -> GleamType
```

```
import { Ok, Error } from "../gleam_stdlib/gleam.mjs";
import { List } from "../prelude.mjs";
```

gleam (4) nix

- ↳ glistix
- no bindings for nixpkgs or nixos (yet)

the BEAM

- default target for gleam
- erlang VM
- target for many languages
- concurrent by design

erlang (0) syntax

```
1> atom. % atom
2> 5 =:= 5.
true
3> 1 =:= 0.
false
4> 1 =/= 0.
true
5> 5 =:= 5.0.
false
% ...
```

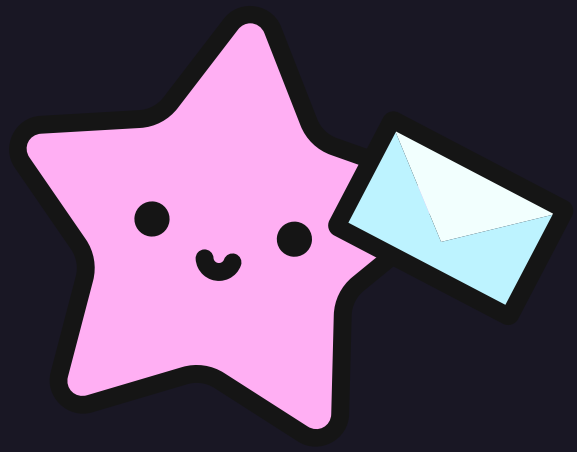
erlang (1) pattern matching

```
function prefix(Gender)
  if Gender == m then
    "Mr."
  else if Gender == f then
    "Mrs."
  else
    "Mx."
end
```

```
prefix(m) -> "Mr.";
prefix(f) -> "Mrs.";
prefix(_) -> "Mx.".
```

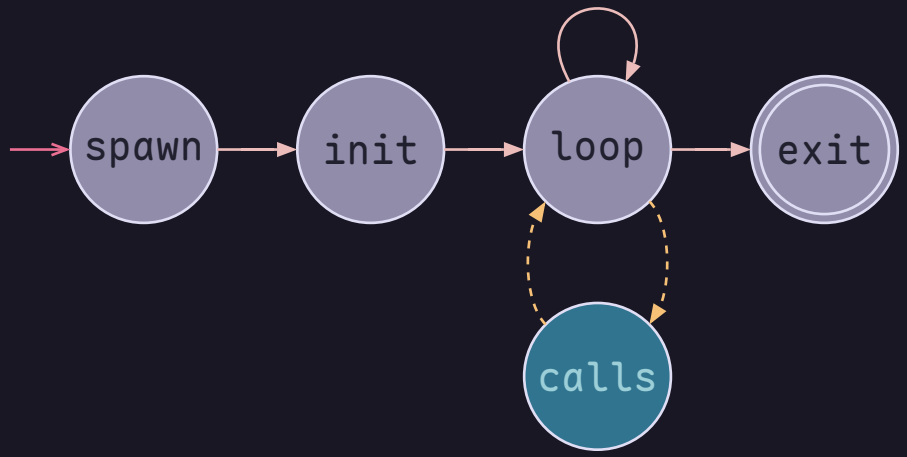
OTP (0)

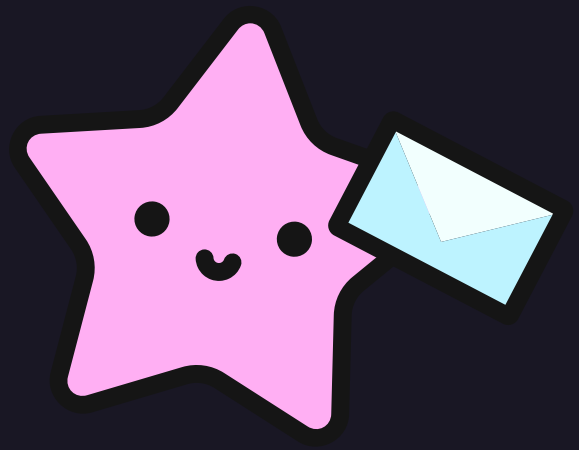
- Open Telecom Platform
- modules and patterns inside erlang
- concurrent applications
- hot code reloading



OTP (1)

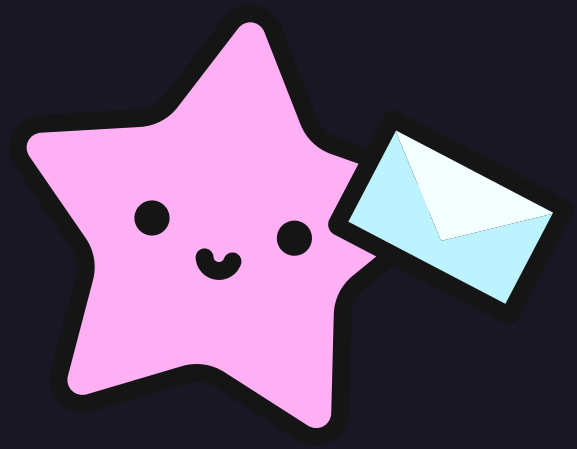
- processes hold state and receive messages





OTP (2) “let it crash”

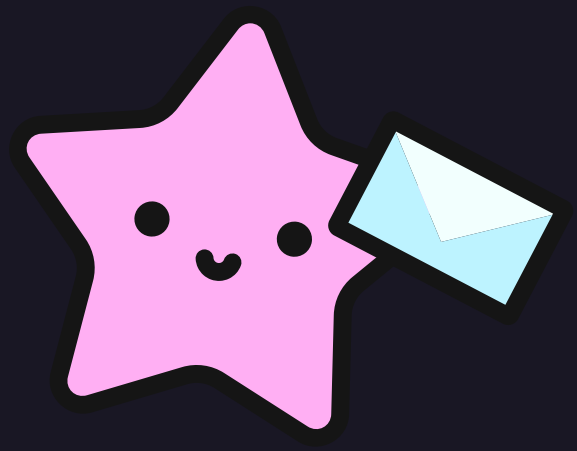
- *“let some other process do the error recovery”*
- *“do not program defensively”*



OTP (3) links

```
chain(0) ->
  receive
    _ -> ok
  after 2000 ->
    exit("chain dies here")
  end;
chain(N) ->
  Pid = spawn(
    fun() -> chain(N-1) end
  ),
  link(Pid),
  receive
    _ -> ok
  end.
```

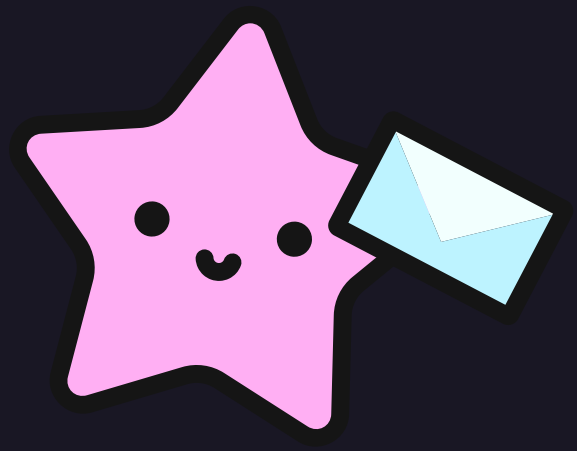
```
[s]=[3]=[2]=[1]=[0]
[s]=[3]=[2]=[1]=*dead*
[s]=[3]=[2]=*dead*
[s]=[3]=*dead*
[s]=*dead*
*dead, error message*
[s] <-- restarted
```



OTP (4) traps

```
1> process_flag(trap_exit,
true).
true
2> spawn_link(
  fun() -> linkmon:chain(3)
end
).
% <0.49.0>
3> receive X -> X end.
% {
%  'EXIT',<0.49.0>,
%  "chain dies here"
% }
```

```
[s]=[3]=[2]=[1]=[0]
[s]=[3]=[2]=[1]=*dead*
[s]=[3]=[2]=*dead*
[s]=[3]=*dead*
[s]<--{'EXIT,Pid,
  "chain dies here"}
[s]<-- still alive
```



OTP (5) monitors

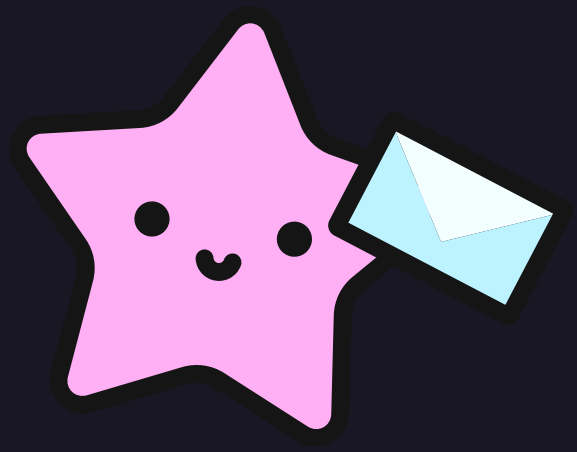
```
1> erlang:monitor(process, spawn(fun() ->
timer:sleep(500) end)).
% #Ref<0.0.0.77>
2> flush().
% Shell got
%  {'DOWN',#Ref<0.0.0.77>,process,<0.63.0>,normal}
% ok
```

OTP (6) behaviours

- a OTP module containing the generic logic to manage types of tasks
- templates for specific logic

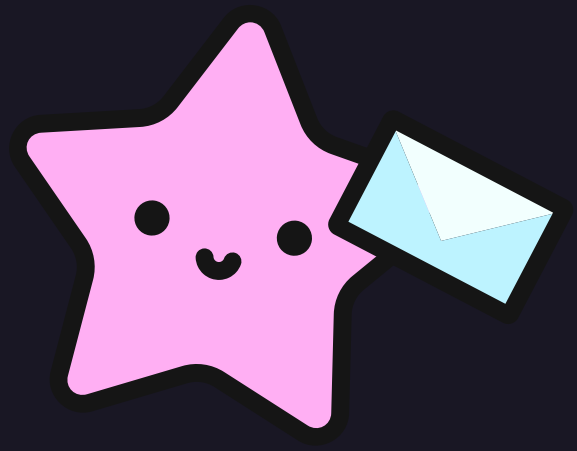
```
-behaviour(BehaviourName).
```

- combination of FP concepts
  - hold state



OTP (7) gen_server

- common OTP behaviour
- client-server model inside the VM
- concurrent
- callbacks: `init/1`, `handle_call/3`, `handle_cast/2`, `terminate/2`, …

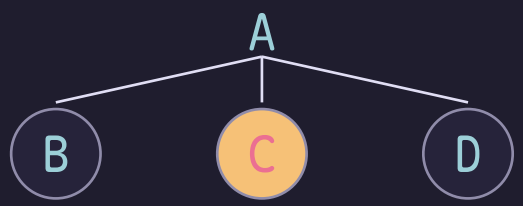
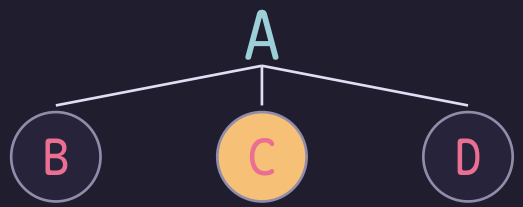
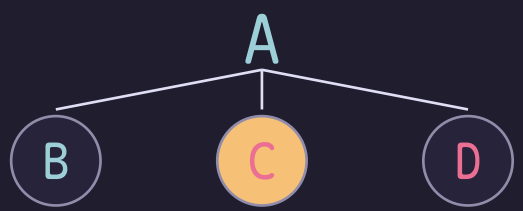


OTP (8) supervisors (0)

- common OTP behaviour
- callbacks `init/1` and `start_link/0`
- build up to a “supervisor-tree”

OTP (9) supervisors (1)

fails restarted

one_for_one	
one_for_all	
rest_for_one	

OTP (10) supervisors (2)

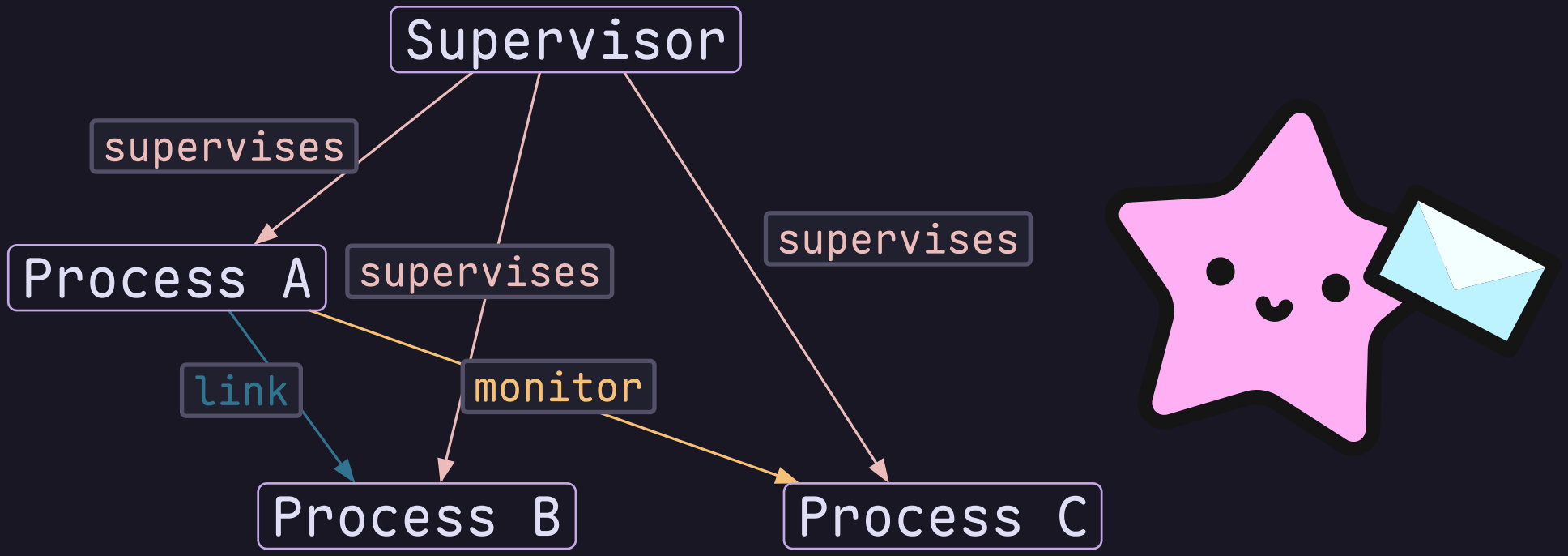
Restart Policy	normal exit	abnormal exit
permanent	restart	restart
transient	stay dead	restart
temporary	stay dead	stay dead

- 2 types of children
  - worker
  - supervisor

OTP (11) supervisors (3)

```
start_link() -> supervisor:start_link({local, ?
MODULE}, ?MODULE, []).
init([]) ->
    WorkerSpec = #{
        id => my_worker,
        start => {my_worker, start_link, []},
        restart => permanent, %% Always restart
        shutdown => 5000,      %% Give it 5 seconds
        type => worker
    },
    SupervisorSpec = #{
        strategy => one_for_one,
        intensity => {10, 60} %% Allow 10fails/minute
    },
    {ok, {SupervisorSpec, [WorkerSpec]}}.
```

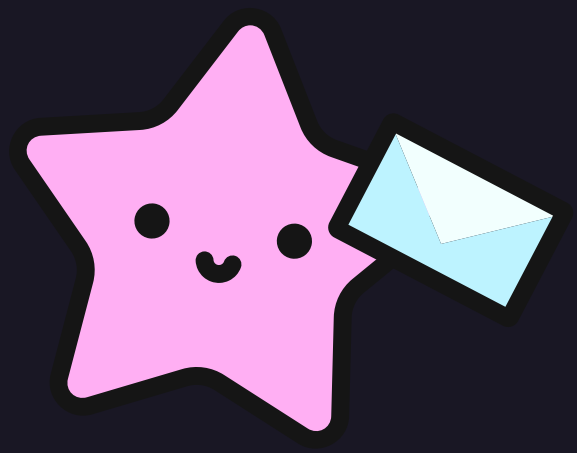
OTP (12) example (0)



OTP (13) example (1)

Supervisor

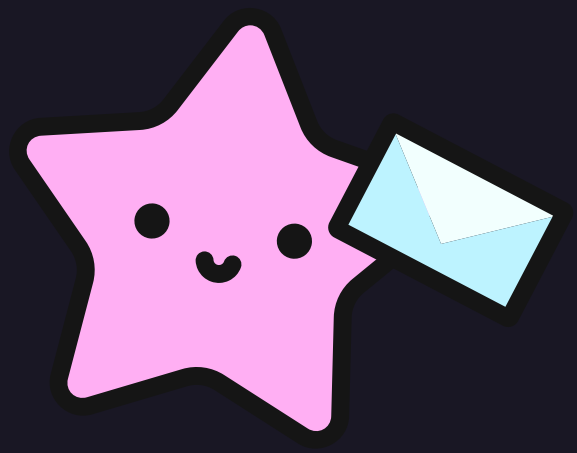
```
init([]) ->
  B_spec = #{
      id => worker_b,
      start => {worker_b, start_link, []},
      restart => permanent,
      shutdown => 5000,
      type => worker
  },
  % same for C and A
  % supervisor spec omitted
  {ok, {SupervisorSpec, [B_spec, C_spec, A_spec]}}.
```



OTP (14) example (2)

Process A (gen_server)

```
% other functions omitted
init([]) ->
  B_pid = whereis(worker_b), % id was given by superv.
  C_pid = whereis(worker_c),
  link(B_pid),
  C_monitor_ref = erlang:monitor(
    process, C_pid
  ),
  {ok,#{
    b_pid => B_pid,
    c_pid => C_pid,
    c_ref => C_monitor_ref
  }}.
```



concurrency in gleam (0)

- a thin, typed overrlay
- ↳ gleam_erlang
- ↳ gleam_otp

concurrency in gleam (1)

```gleam
import gleam/erlang/process.{call, send}

let assert Ok(actor) = actor.new([])
  |> actor.on_message(handle_message) |> actor.start
let subject = actor.data

send(subject, Push("Mike"))
let assert Ok("Robert") = call(subject, 10, Pop)
let assert Error(Nil) = call(subject, 10, Pop)

send(subject, Shutdown)
```

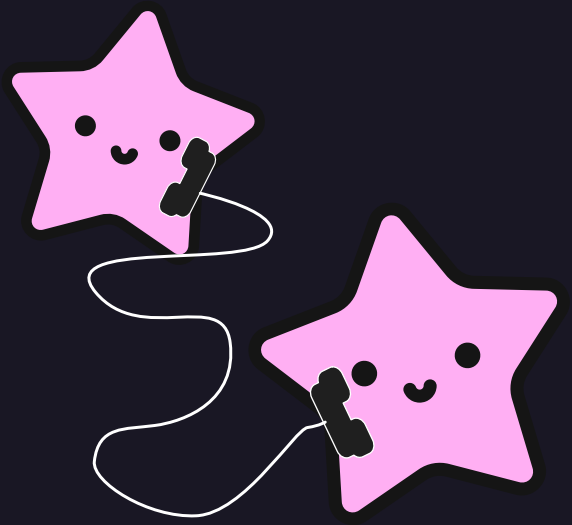
concurrency in gleam (2)

```
pub fn start_supervisor() ->
actor.StartResult(Supervisor) {
  supervisor.new(supervisor.OneForOne)
  |> supervisor.add(one_node.supervised())
  |> supervisor.add(other_node.supervised())
  |> supervisor.start
}
```

distributed systems

```
node.self() |> pprint.debug()
let name = atom.create("one@127.0.0.1")
let assert Ok(n) = node.connect(name)
  |> pprint.debug()
node.visible() |> pprint.debug()
```

```
(gleam@127.0.0.1)1> first:main().
% atom.create("gleam@127.0.0.1")
% Ok(atom.create("one@127.0.0.1"))
% [atom.create("one@127.0.0.1")]
(gleam@127.0.0.1)2>
```



- then define types in an erlang FFI

real world examples

- whatsapp
- discord
- Klarna
- Cisco
- ↳ my guestbook

⇒ high availability

highest paying programming language in 2024

read more

- ↳ gleam docs
- ↳ lernyousomeerlang.com
- ↳ erlang docs
- ↳ distributed erlang docs

how to: these slides (for 077 & quagga)

- typst (polylux, fletcher, arborly)
- gleam artwork from the ↳ website (this is the most important step)
- remix the artwork
- ↳ rose-pine
- ↳ Maple Mono NF

thanks for listening :)

https://computerbetrug.jetzt/olaf/gleam.pdf

written by human
not by AI

licensed under
↳ EUPLv1.2