

# OTP Workshop

[https://github.com/fishcakez/empex\\_workshop.git](https://github.com/fishcakez/empex_workshop.git)

# OTP Workshop

Links, Monitors and Tasks

# What is the state of the system?



Good State



Bad State

What is the state of the system?



What is the state of the system?



What is the state of the system?



# Let it crash

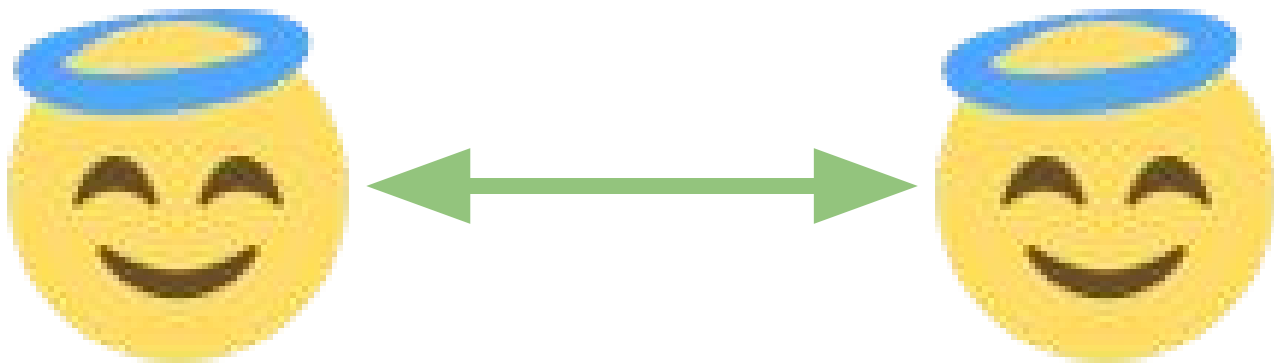
- ANY process can enter BAD state at ANY time
- ANY process can crash at ANY time
- ANY process depending on a crashed process can enter BAD state
- Crash ANY process in BAD state
- Restart ANY crashed process to GOOD state

# Links

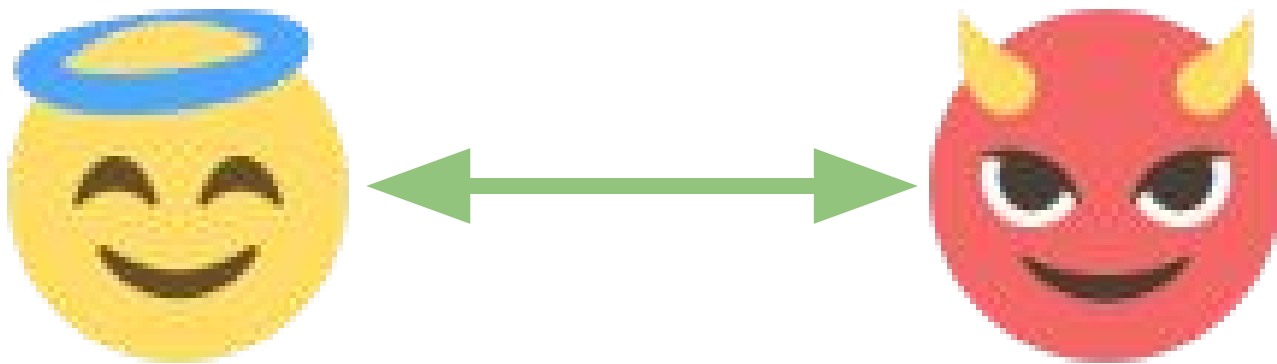
- Bidirectional relationship between two processes
- `spawn_link/1,3`
- `Process.link/1`
- `Process.unlink/1`



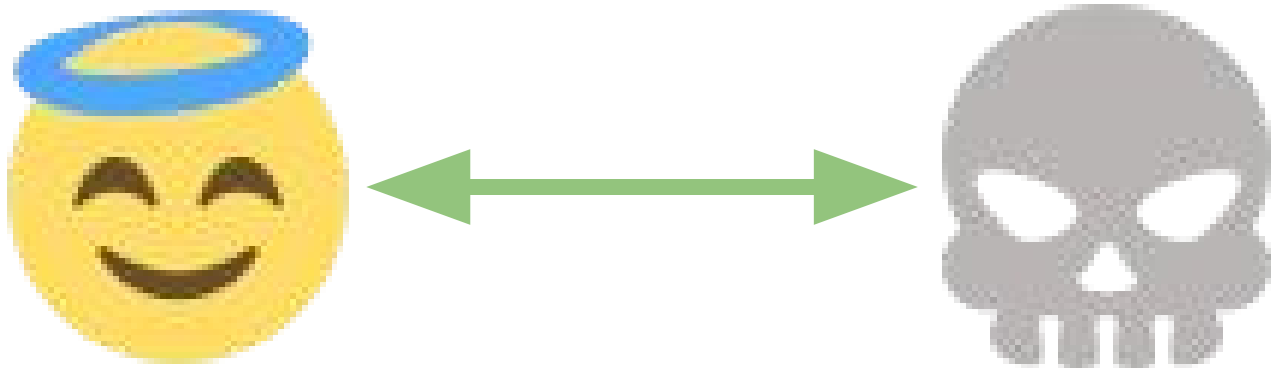
# Links



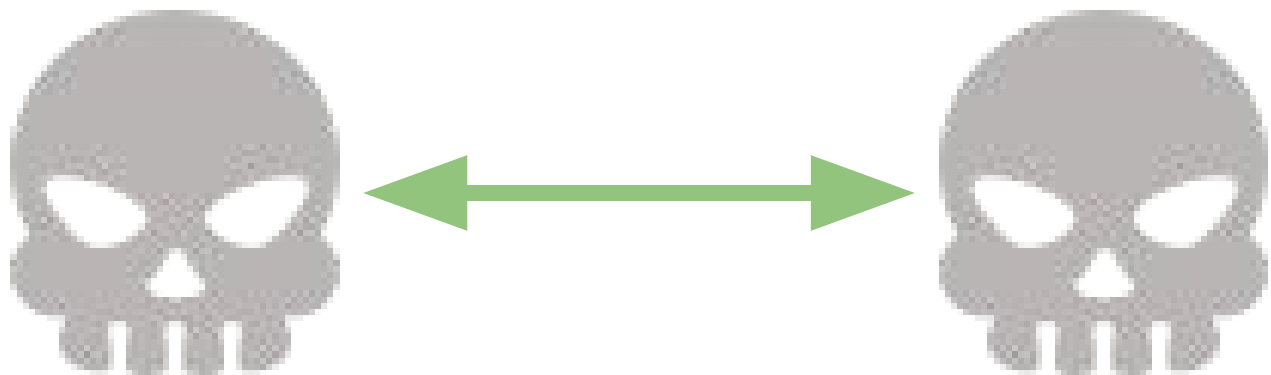
# Links



# Links



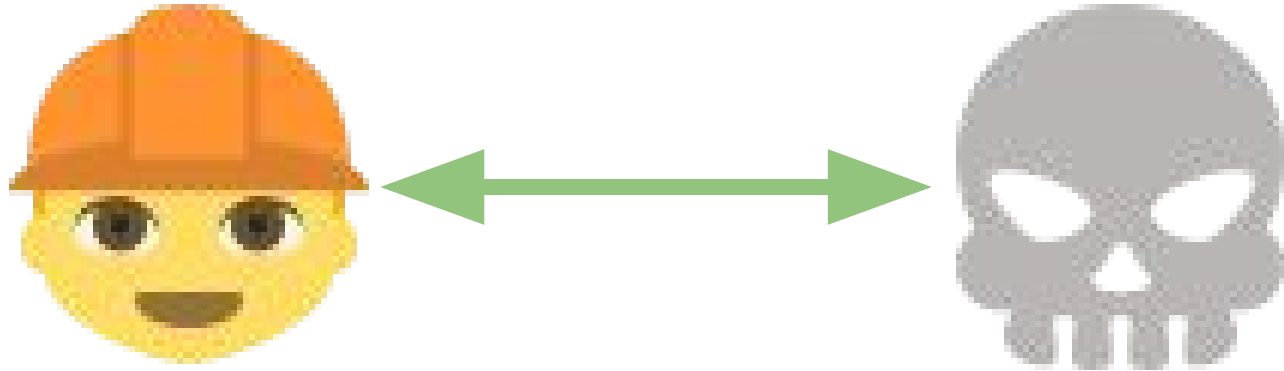
# Links



# Trapping exits

- `Process.flag(:trap_exit, true)`
- `{:EXIT, pid, reason}`
- `Process.exit/2`

# Trapping exits



# Trapping exits



# Exit reasons

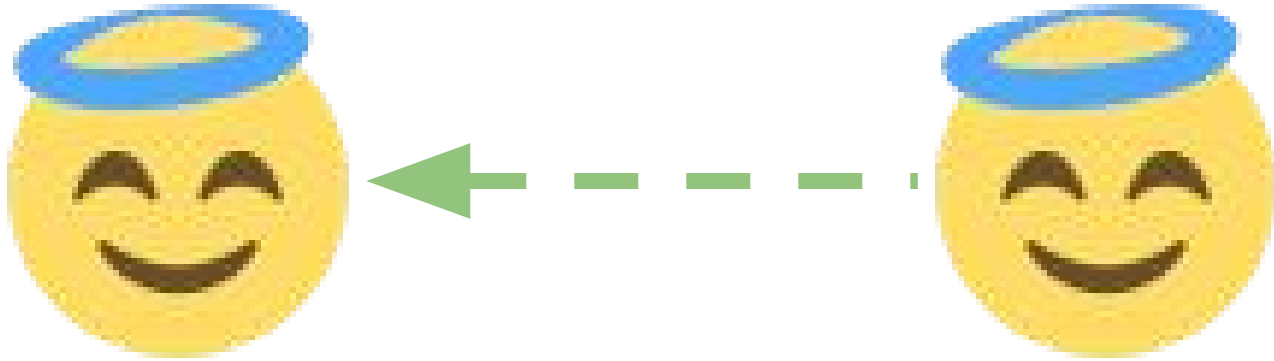
- `:normal`
- `:killed (:kill)`
- `:shutdown`
- `{:shutdown, reason}`
- `:noprocs`
- `:noconnect`
- `other_reason`



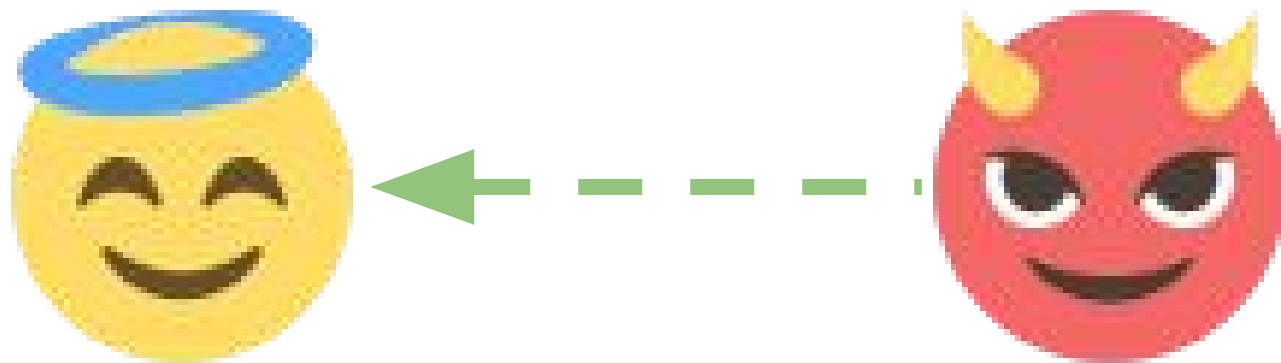
# Monitors

- Unidirectional relationship between two processes
- `spawn_monitor/1,3`
- `Process.monitor/1`
- `Process.demonitor/1,2`
- `{:DOWN, ^ref, :process, pid, reason}`

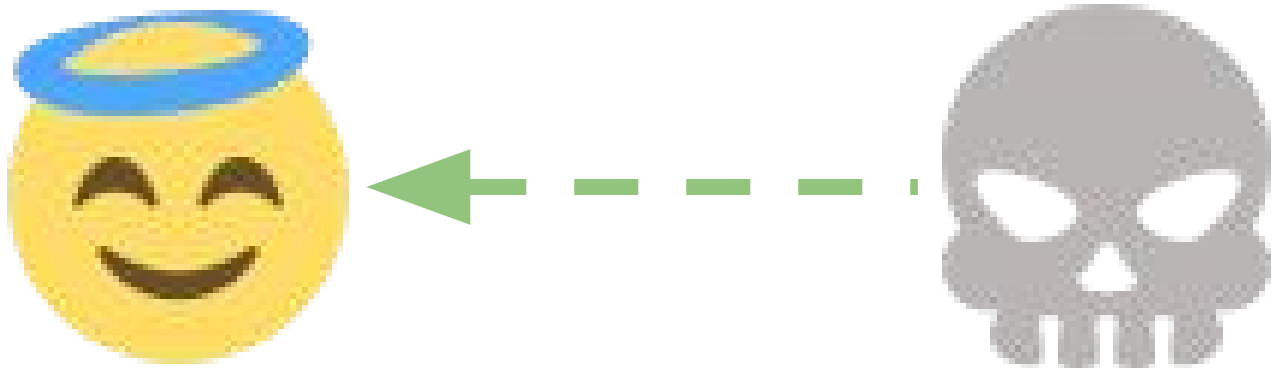
# Monitors



# Monitors



# Monitors



# Monitors



# Process.demonitor(ref, [option, ...])

- [:flush] - guarantees :DOWN message won't be delivered
- [:info] - information about whether monitor existed
- [:flush, :info] - guarantees no :DOWN and whether :flush required

# Task

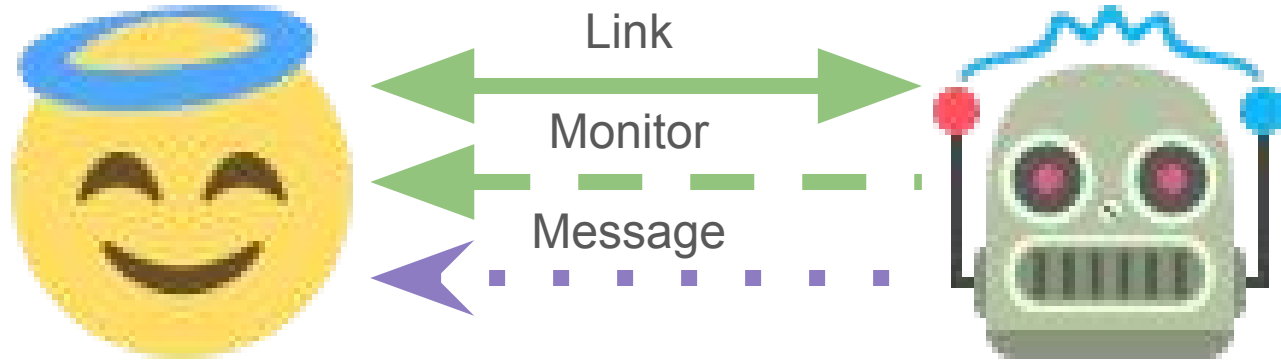
- `Task.start/1,3`
- `Task.start_link/1,3`
- `Task.async/1,3`
- `Task.await/1,2`
- `Task.yield/1,2`
- `iex --logger-sasl-reports true`

# Task.async/1,3

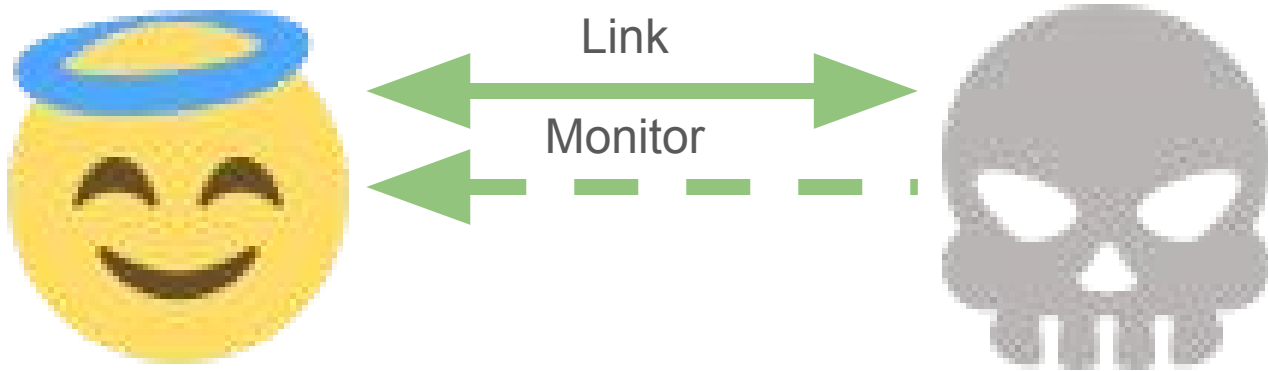
- `%Task{pid: pid, ref: ref} = Task.async(fn() -> ... end)`
- Spawn with link and monitor
- `{^ref, result}`
- `{:DOWN, ^ref, :process, pid, reason}`



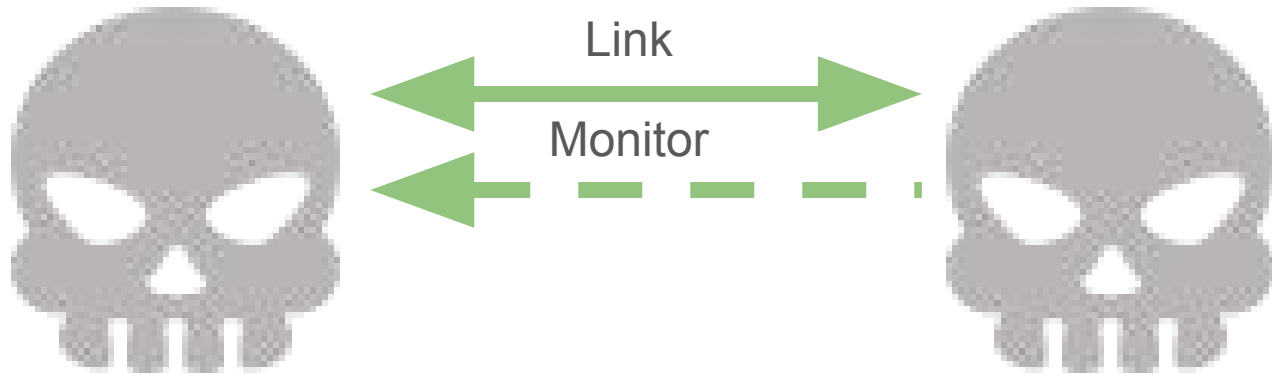
# Task.async/1,3



Task.async/1,3



Task.async/1,3



# Task.await/1,2

- Task.await(%Task{ref: ref}, timeout)
- Process.demonitor(ref, [:flush])
- On timeout: { :timeout, { Task, :await, [%Task{ref: ref}, timeout] } }
- On crash: { reason, { Task, :await, [%Task{ref: ref}, timeout] } }

# Task.yield/1,2

- Task.yield(%Task{ref: ref}, timeout)
- Does not call Process.demonitor/1,2 on timeout
- On result: {:ok, result}
- On timeout: nil
- On crash: {:exit, reason}

# Task problems