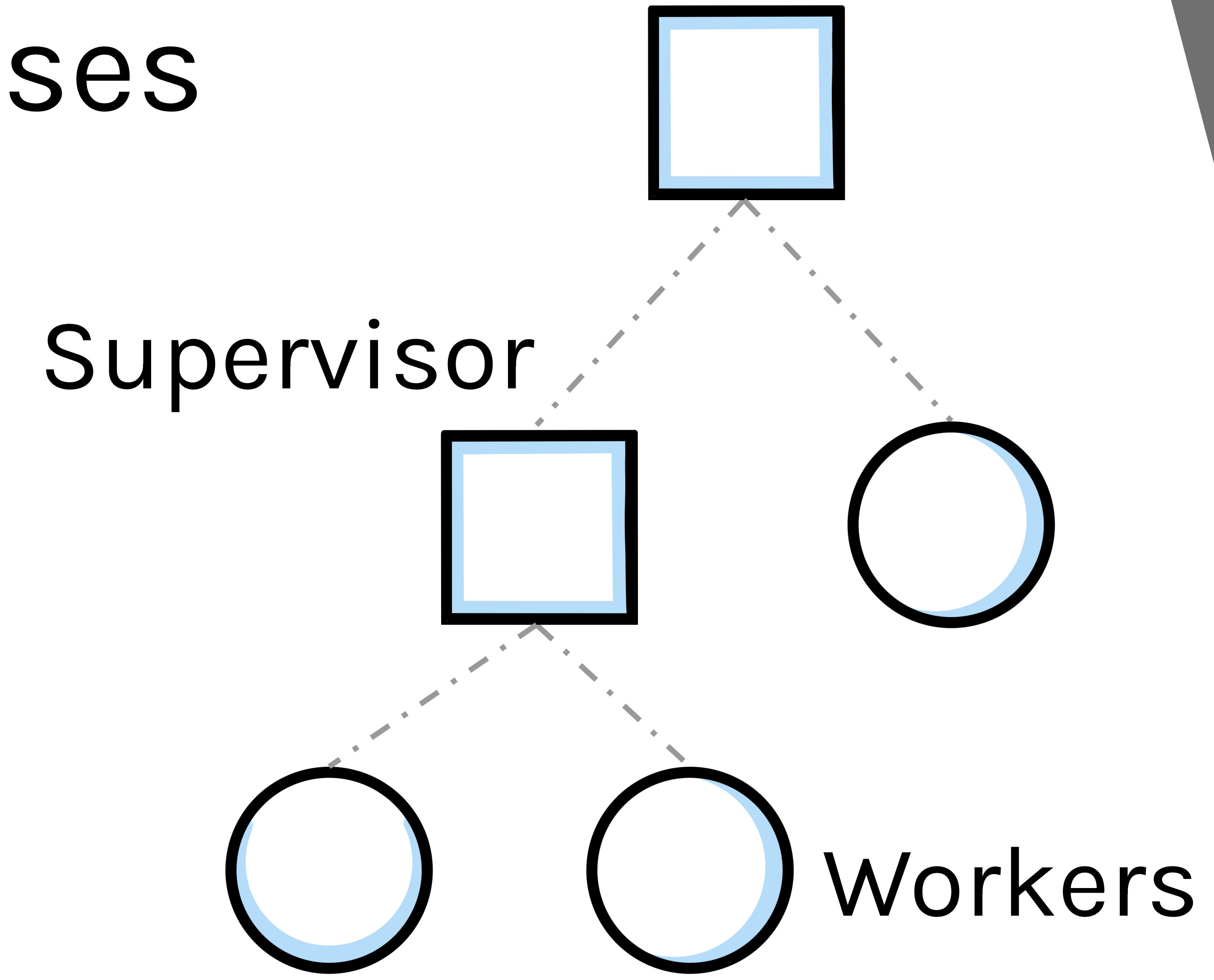
SUPERVISORS

Supervisors

- Supervisors
- Supervisor Example
- Generic Supervisors
- Dynamic Children
- Non OTP-compliant Processes

Supervisors

- Erlang systems consist of supervision trees
- Supervisors will start child processes
  - Workers
  - Supervisors
- Supervisors will monitor their children
  - Through links and trapping exits
- Supervisors can restart the children when they terminate



Supervisor Example

```erlang
-module(my_supervisor).
-export([start/2, init/1, stop/1]).

start(Name, ChildSpecList) ->
    register(Name, spawn(?MODULE, init, [ChildSpecList])).

stop(Name) -> Name ! stop.

init(ChildSpecList) ->
    process_flag(trap_exit, true),
    loop(start_children(ChildSpecList, [])).
```

Supervisor Example

Assumes the child links to the supervisor

```
start_children([], Acc) -> Acc;
start_children([{Mod,Fun,Args} | ChildSpecList], Acc) ->
    {ok, Pid} = apply(Mod, Fun, Args),
    start_children(ChildSpecList, [{Pid, {Mod,Fun,Args}} | Acc]).

loop(ChildList) ->
    receive
      {'EXIT', Pid, Reason} ->
          NewChildList = restart_child(Pid, ChildList),
          loop(NewChildList);
       stop ->
           terminate(ChildList)
   end.
```

Supervisor Example

```
restart_child(Pid, ChildList) ->
    {Pid, {Mod,Fun,Args}} = lists:keyfind(Pid, 1, ChildList),
    {ok, NewPid} = apply(Mod, Fun, Args),
    [{NewPid, {Mod,Fun,Args}}|lists:keydelete(Pid, 1, ChildList)].

terminate([]) -> ok;
terminate([{Pid, _} | ChildList]) ->
   exit(Pid, kill),
  terminate(ChildList).
```

Clears the crashed child and starts a new one

Supervisor Example

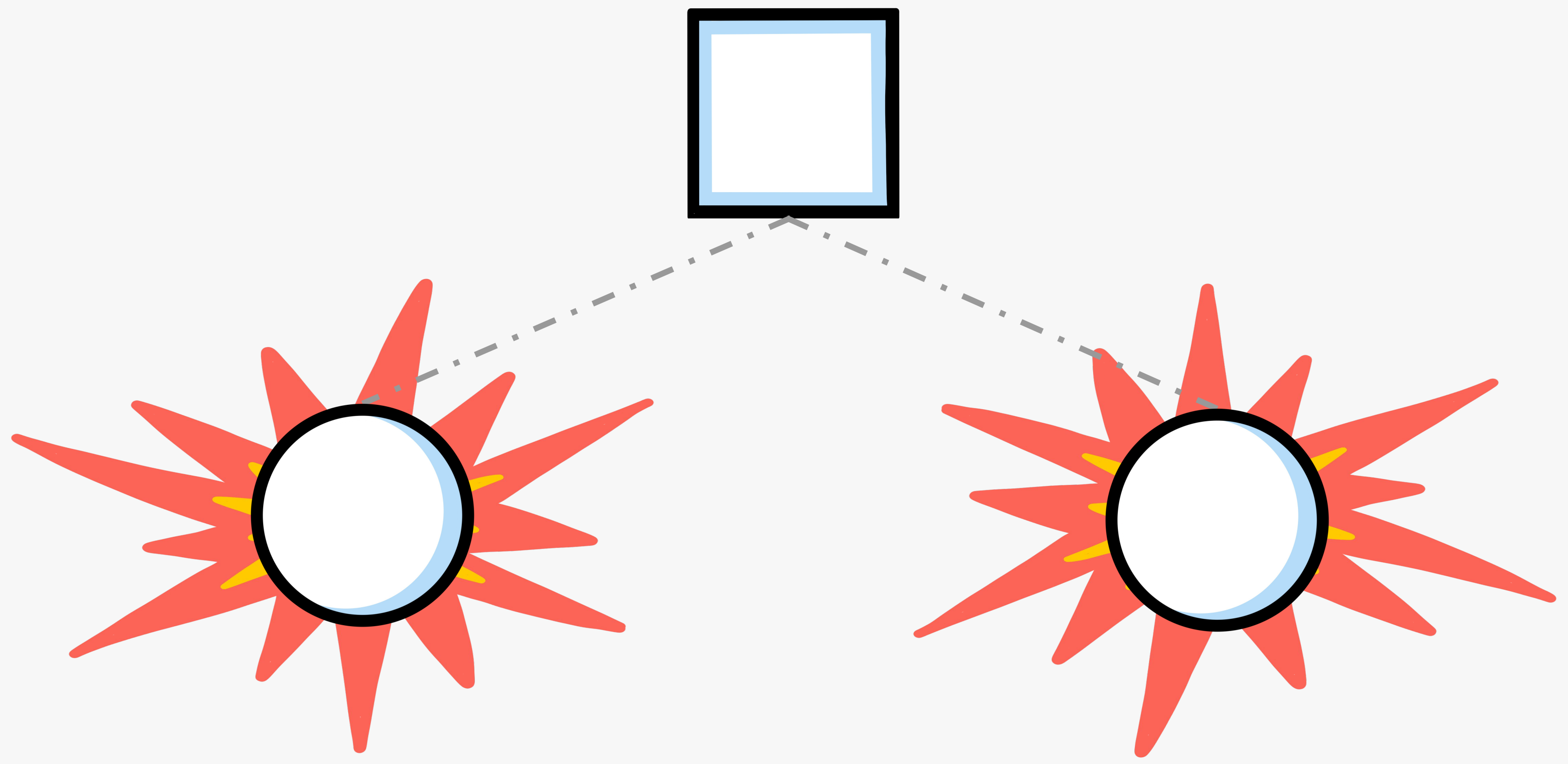
```
-module(test).
-export([start/1, init/1]).

start(Name) ->
 {ok, spawn_link(?MODULE, init, [Name])}.

init(Name) ->
    register(Name, self()),
    io:format("Started ~p~n",[Name]),
    loop().

loop() ->
    receive stop -> exit(byebye) end.
```

The test process links itself to the parent, prints out a status message and waits for a stop message

Note the non-normal exit reason

Supervisor Example

```
1> my_supervisor:start(sup,[{test, start, [a]},{test, start, [b]}]).
true

started a
started b

2> a ! stop
stop
started a

3> exit(whereis(b), kill)
true
started b
```



Generic Supervisors

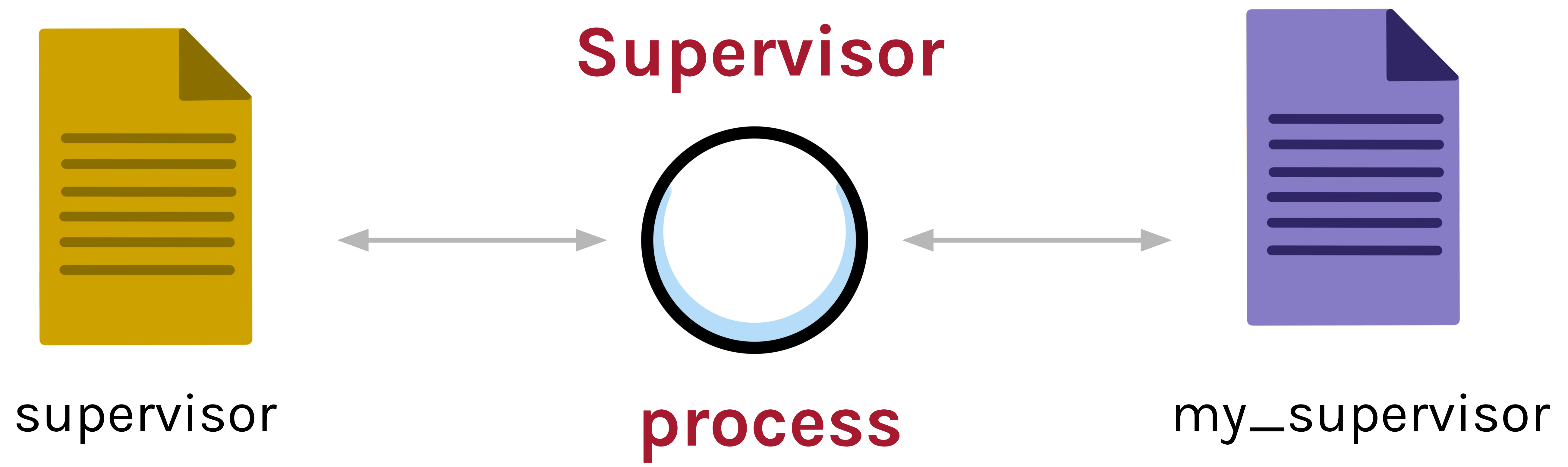
Generic

- Spawning the supervisor
- Starting the children
- Monitoring the children
- Restarting the children
- Stopping the supervisor
- Cleaning up

Specific

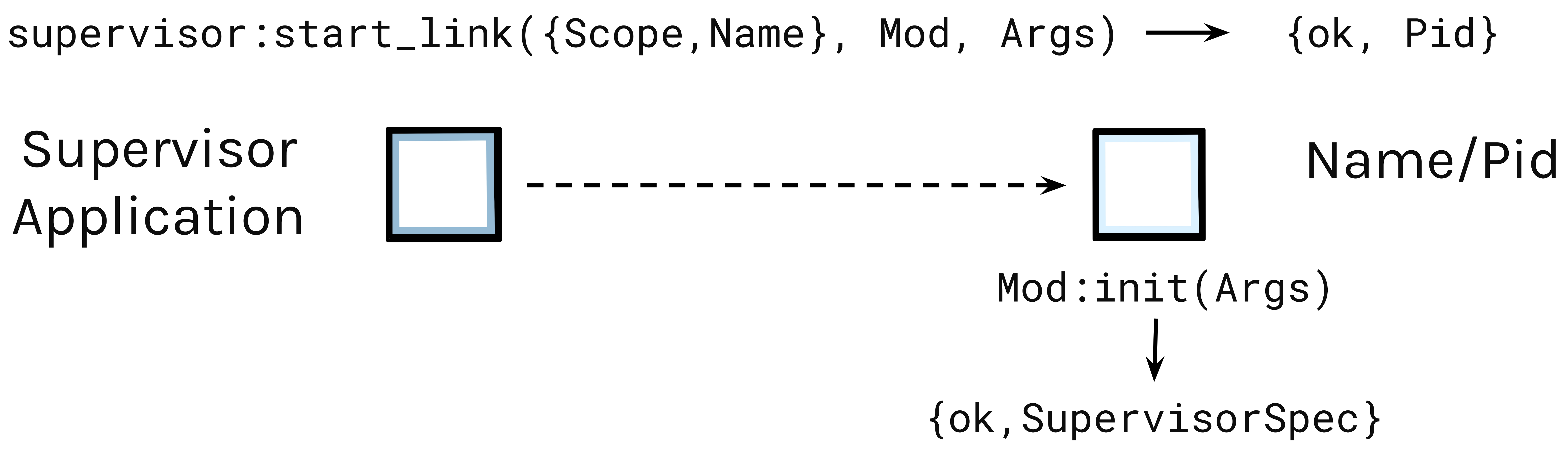
- What children to start
- Specific child handling
  - Start, Restart
- Child dependencies
- Supervisor name
- Supervisor behaviours

Generic Supervisors



- Supervisors are implemented in the **supervisor** module
- The behaviour directive must be included in the callback module

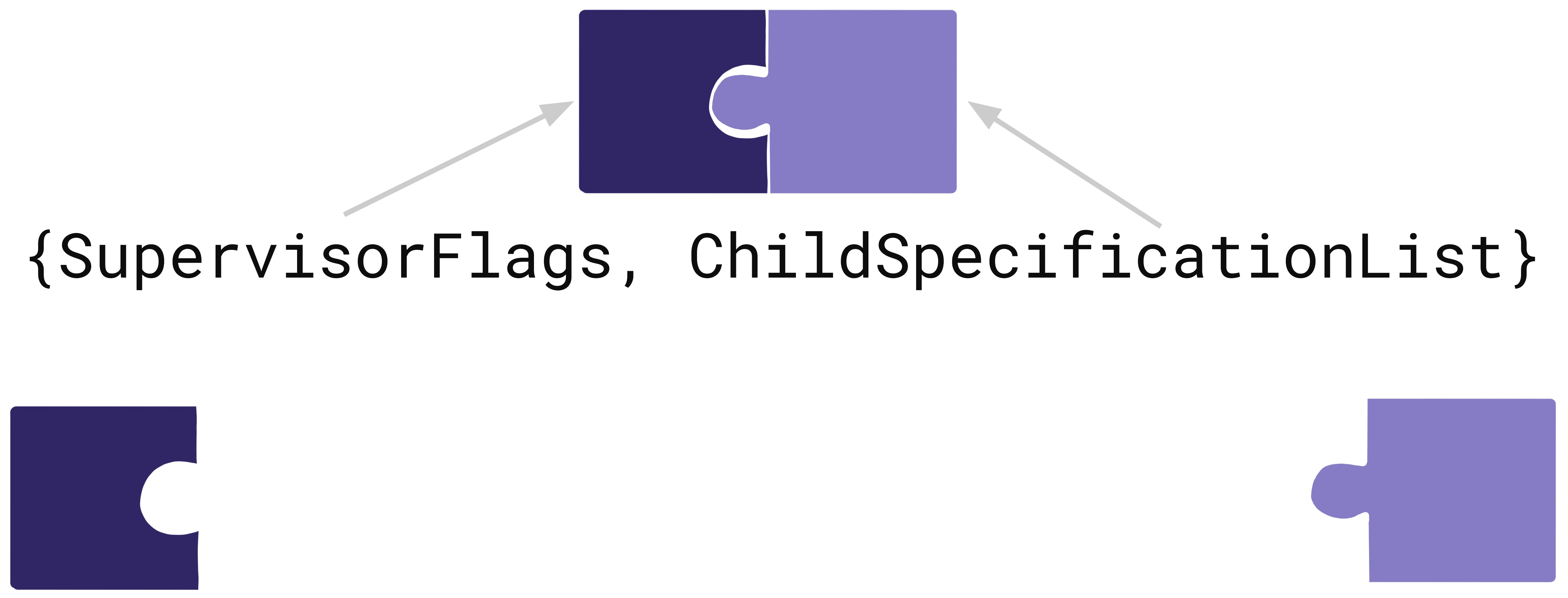
Generic Supervisors



- **supervisor:start_link/3** creates a new supervisor
  - **Name** is the process name. **Scope** can be **local** or **global**
  - **Mod** is the name of the callback module
  - **Args** are the arguments passed to the init function
- **Mod:init/1** is called by the supervisor in the callback module
  - It returns a supervisor specification

Generic Supervisors: specifications

Supervisor Specification



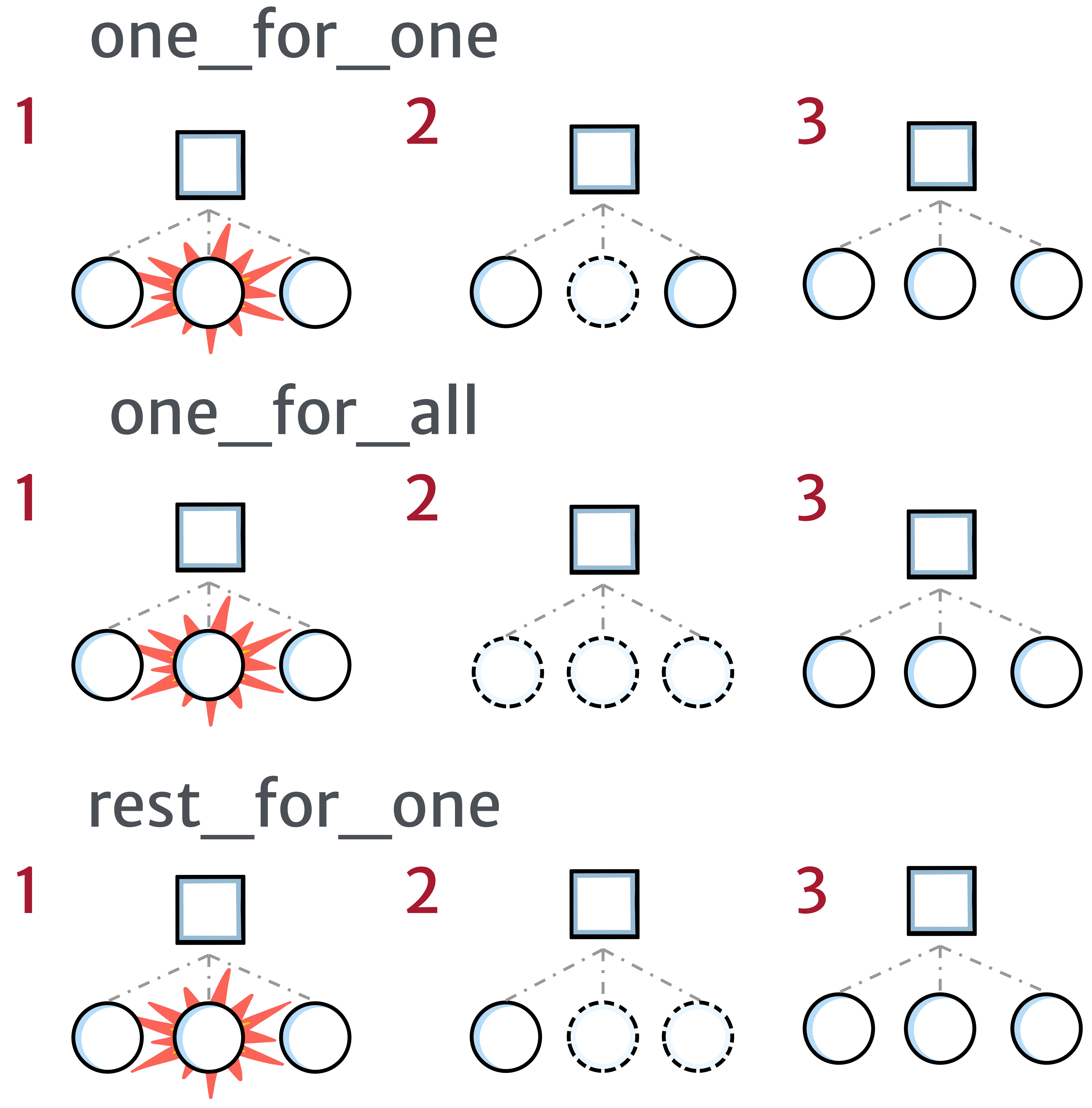
```
{SupervisorFlags, ChildSpecificationList}
```

```
#{strategy, intensity, period}
```

```
[#{id, start, restart, shutdown,
   type, modules}]
```

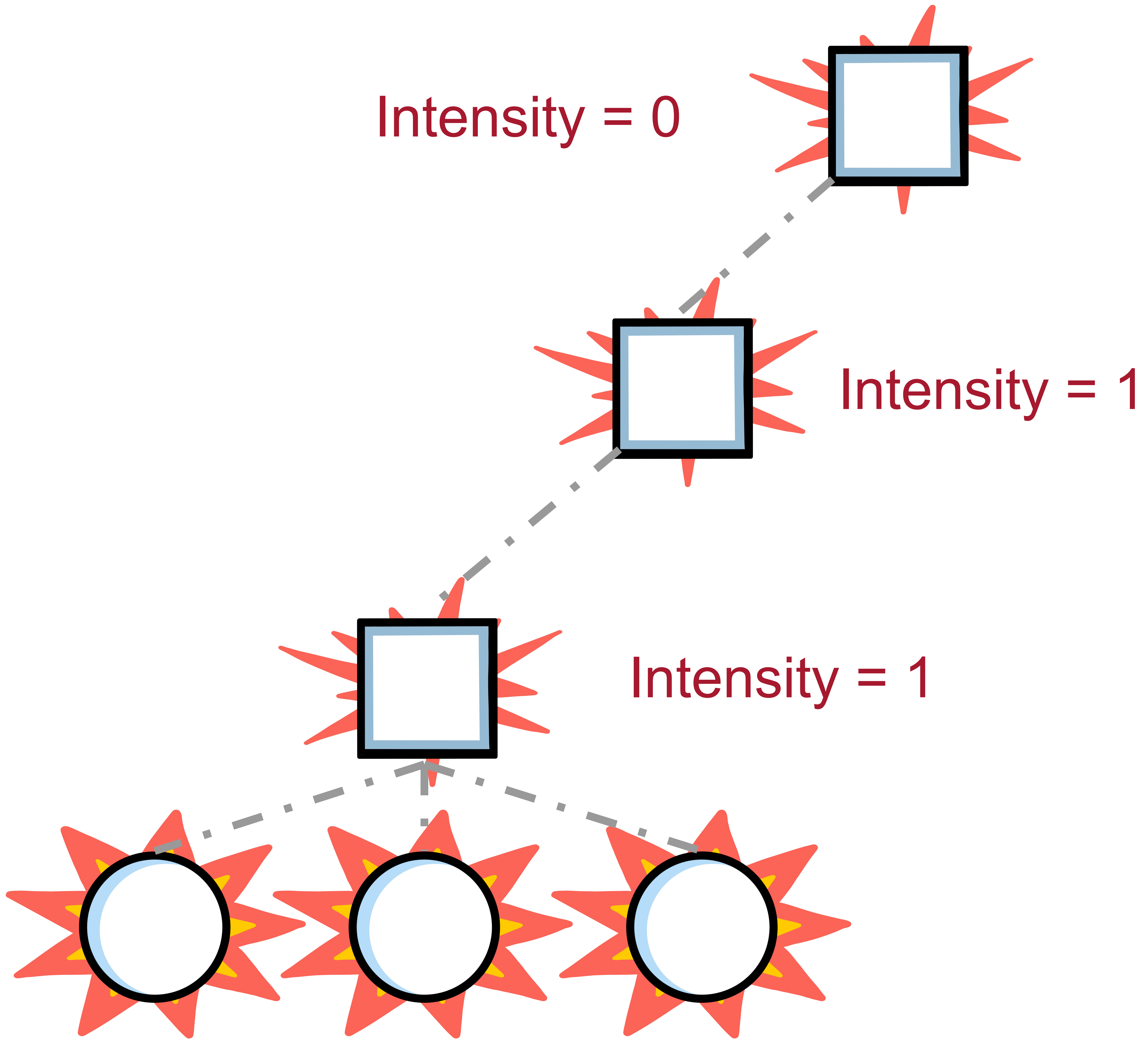
- The supervisor specification is a tuple containing:
  - Supervisor non-generic information on the restart strategy
  - Child specifications for all static children

Generic Supervisors: restart strategy



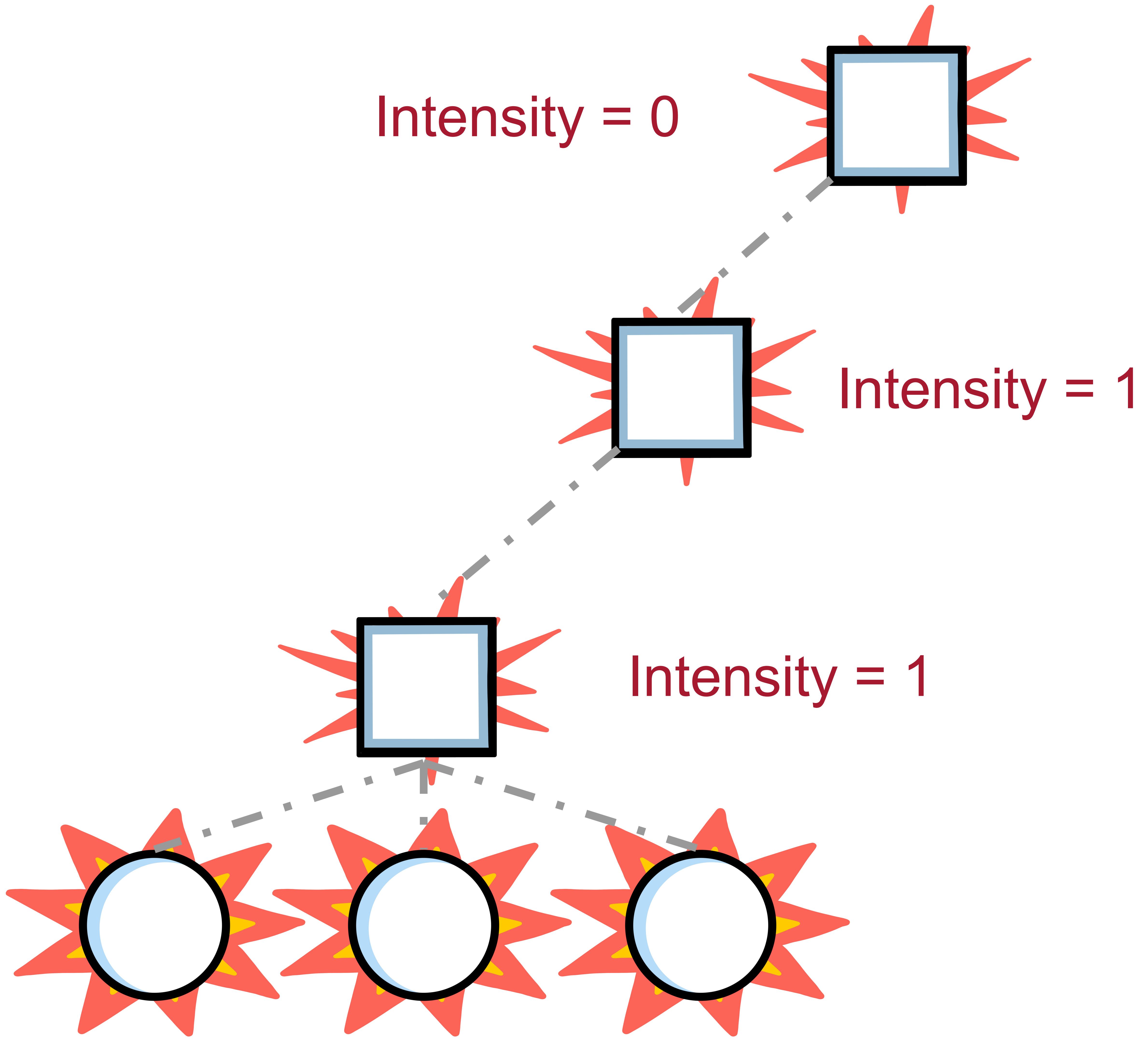
- one_for_one
  - Only the crashed process is restarted
- one_for_all
  - All processes are terminated and restarted
- rest_for_one
  - All processes started after the crashed one are terminated and restarted
- simple_one_for_one
  - Dynamic children of the same type

Generic Supervisors: **restart strategy**



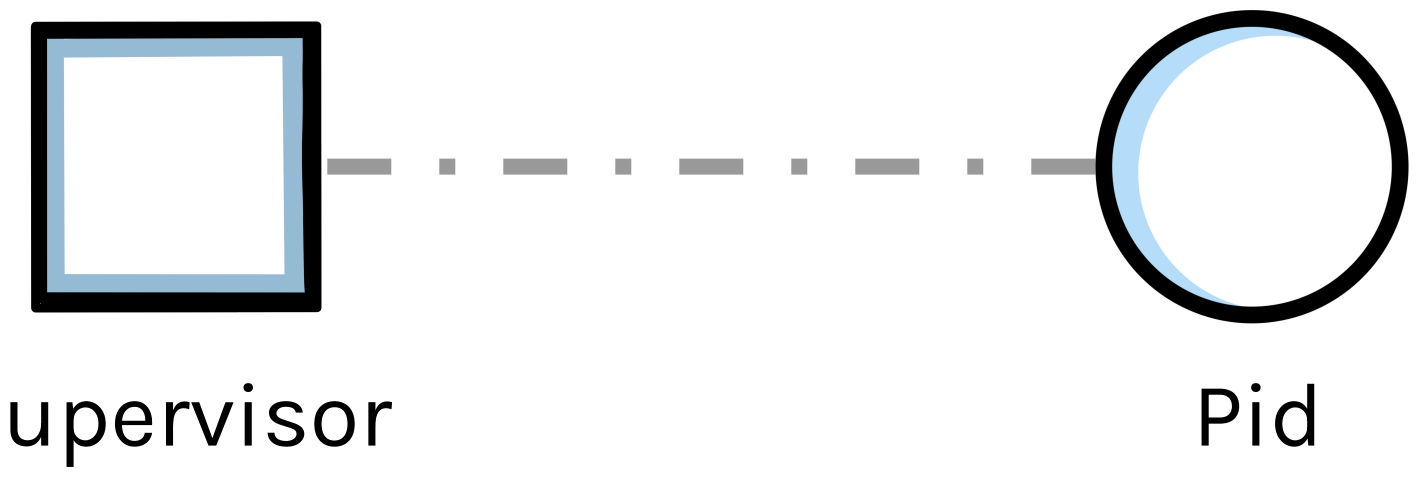
- Intensity
  - Maximum number of restarts in Max Time
- Period
  - If Intensity is reached in Period seconds, the supervisor terminates
- Crashes propagate among supervisors

Generic Supervisors: restart strategy



- Intensity
  - Maximum number of restarts in Max Time
- Period
  - If Intensity is reached in Period seconds, the supervisor terminates
- Crashes propagate among supervisors

Generic Supervisors: child specs



```
ChildSpec = #{id,
              start,
              restart,
              shutdown,
              type,
              modules}
```

- Id
  - Any valid Erlang Term
  - Unique for that supervisor
- Start function
  - {Module, Function, Args}
  - must call an OTP-compliant start_link function
- Restart Type
  - **permanent** is always restarted
  - **transient** is only restarted after a non-normal exit
  - **temporary** is never restarted

Generic Supervisors: child specs

Supervisor
Pid

```
ChildSpec = #{id,
             start,
             restart,
             shutdown,
             type,
             modules}
```

- Shutdown Time
  - Time the process is allowed to spend in terminate
  - Integer > 0 in ms, or **infinity** or **brutal_kill**
- Process Type
  - Used for software upgrades
  - **supervisor** for supervisors
  - **worker** for other behaviours
- Modules
  - List of modules implementing the child
  - **dynamic** if modules are not known (ex.: event handlers)

Supervisor Example

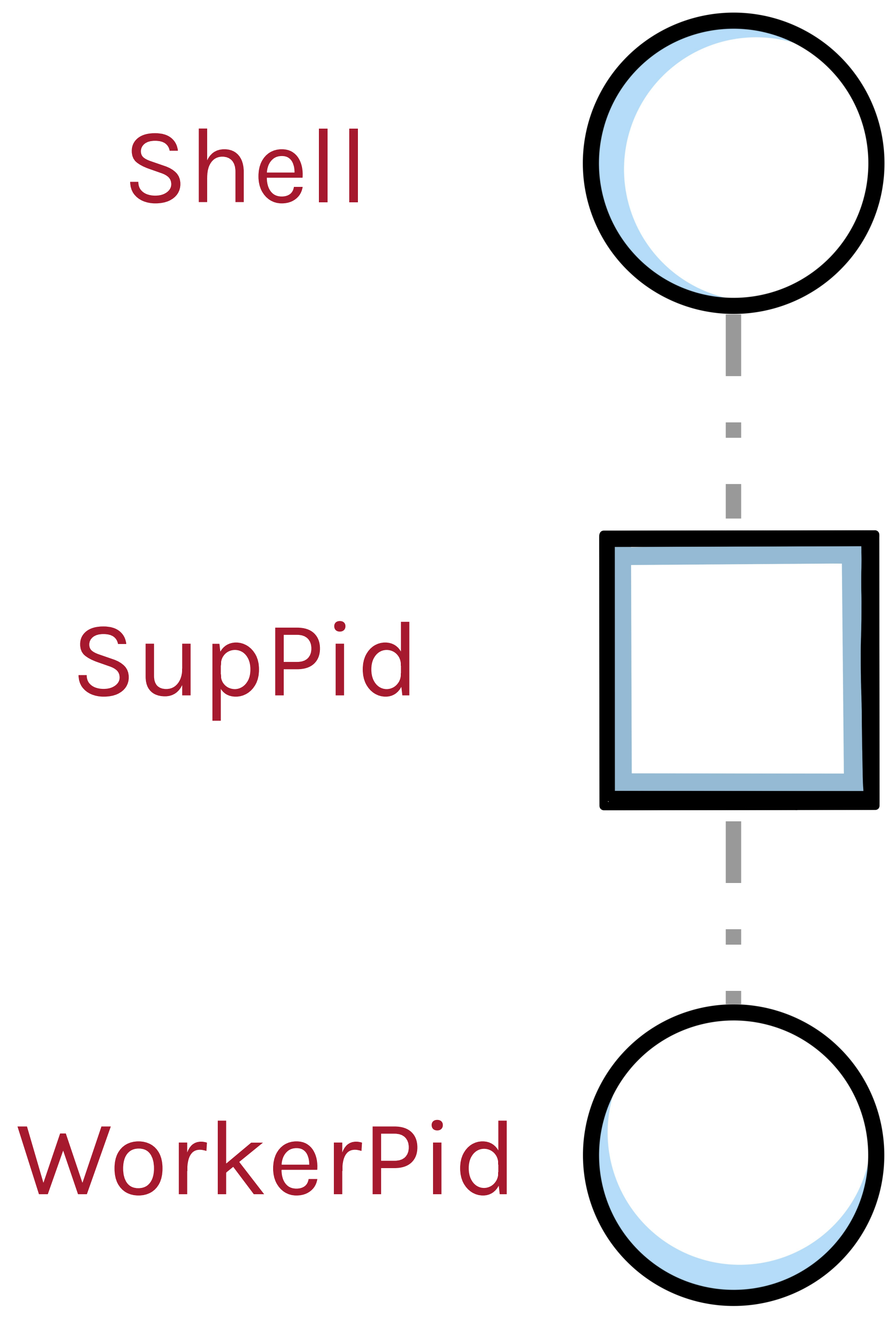
```erlang
-module(sup).
-behaviour(supervisor).
-export([start_link/0, init/1, stop/0]).

start_link() ->
  supervisor:start_link({local,?MODULE}, ?MODULE, []).

stop() -> exit(whereis(?MODULE), shutdown).

init(_) ->{ok,{#{strategy => one_for_one,
                 intensity => 2, period => 3600},
               [child(frequency)]}}.

child(Module) ->
    #{id => Module, start => {Module, start_link, []},
      restart => permanent, shutdown => 2000,
      type => worker, modules => [Module]}.
```

Supervisor Example

Supervisor

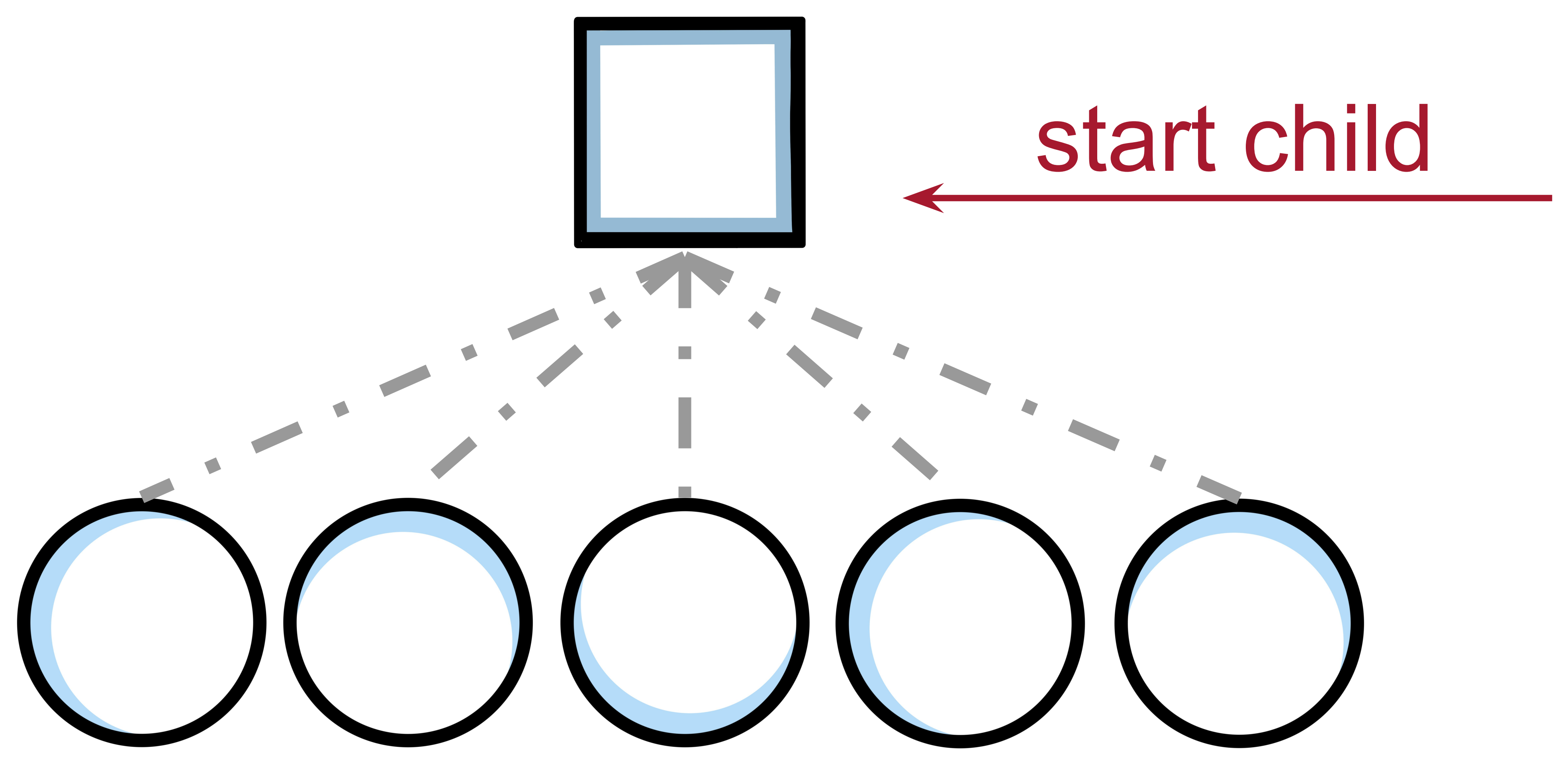
```
1> sup:start_link().
{ok, <0.42.0>}
```

```
sup:start_link() => {ok, SupPid}
sup:init([]) => {ok, {#{strategy => ...}
frequency:start_link() => {ok, WorkerPid}
```



- Synchronous start
- A crash during process start will cause the supervisor to fail
- Minimise the amount of work done during initialisation

Dynamic Children



- What if we do not know the children at start up time?
- What if they are many and all of the same type?
  - Use Dynamic Children
  - Set the supervisor restart type to simple_one_for_one
- We can add or remove children during runtime

Dynamic Children

```
1> {ok, SupPid} = my_supervisor:start_link().
{ok,<0.66.0>}
2> Spec = {lr,{lr,start_link,[]},transient,1,worker,[lr]}.
{lr, {lr, start_link, []}, transient, 1, worker, [lr]}
3> supervisor:check_childspecs([Spec]).
ok
4> supervisor:start_child(SupPid, Spec).
{ok,<0.70.0>}
5> supervisor:terminate_child(SupPid, lr).
ok
6> supervisor:restart_child(SupPid, lr).
{ok,<0.73.0>}
7> supervisor:which_children(SupPid).
[{lr,<0.73.0>,worker,[lr]}]
8> supervisor:terminate_child(SupPid, lr),
   supervisor:delete_child(SupPid, lr).
ok
```

simple_one_for_one Example

```
-module(sup).
-behaviour(supervisor).
-export([start_link/0, init/1]).

start_link() ->
    supervisor:start_link({local, ?MODULE}, ?MODULE, []).

init(_) ->
    {ok, {#{strategy => simple_one_for_one,
            intensity => 2, period => 3600},
           [#{id => test, start => {test, start_link, []},
              restart => permanent, shutdown => 2000,
              type => worker, modules => [test]}]}}.
```

- There can only be one child specification shared by all children
- The only difference will be arguments passed when starting the children.

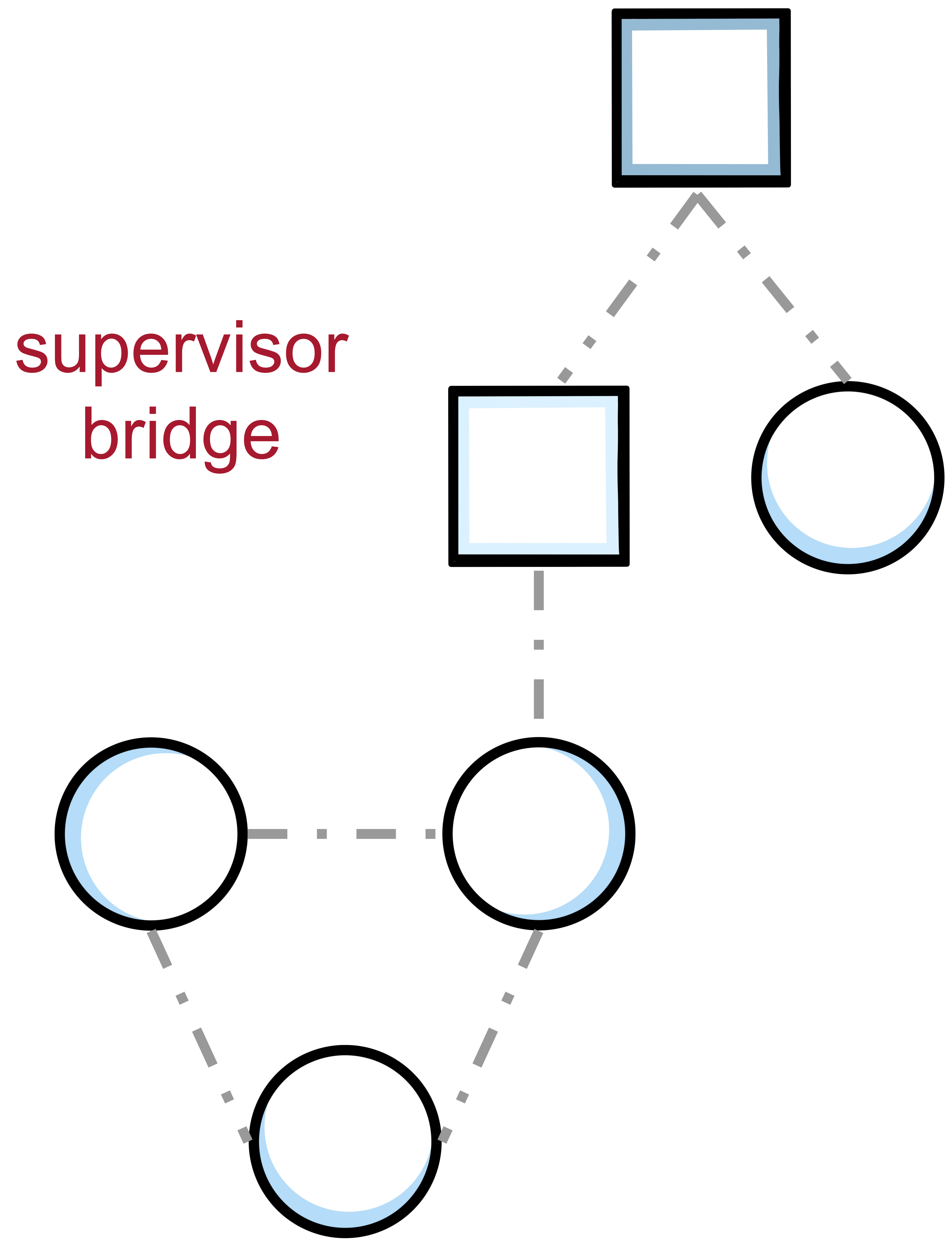
simple_one_for_one Example

```
-module(test).
-export([start_link/1, init/1]).

start_link(Name) ->
    Pid = spawn_link(test, init, [Name]),
    register(Name, Pid),
    {ok, Pid}.

init(Name) ->
    io:format("~p started~n", [Name]),
    loop().

loop() -> receive X -> X end.
```

simple_one_for_one Example

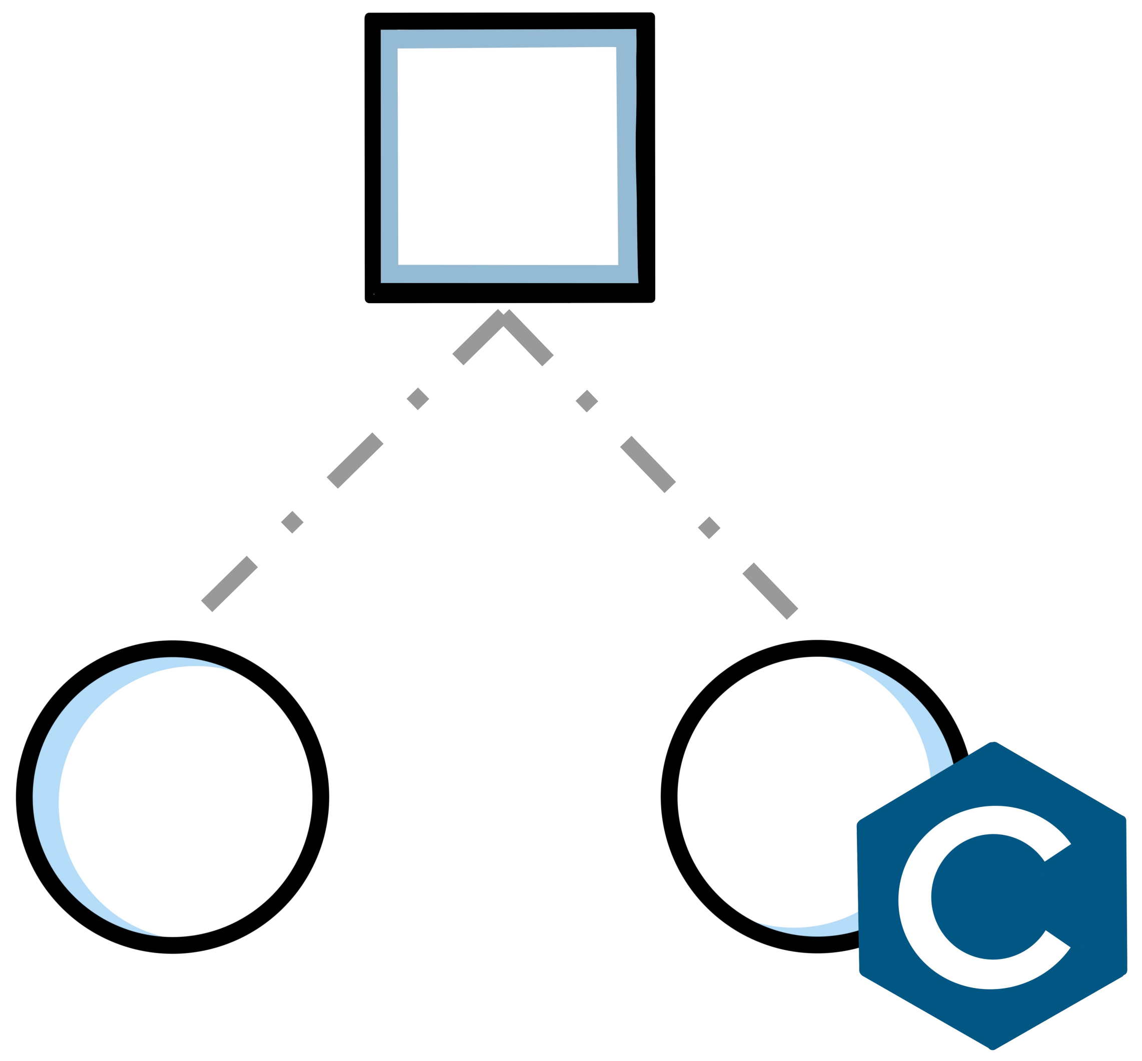
```
1> sup:start_link().
{ok,<0.42.0>}
2> supervisor:start_child(sup, [one]).
one started
{ok,<0.43.0>}
3> supervisor:start_child(sup, [two]).
two started
{ok,<0.44.0>}
4> supervisor:which_children(sup).
[{undefined,<0.44.0>,worker,[test]},
 {undefined,<0.43.0>,worker,[test]}]
```

Non OTP-compliant Processes



- Non OTP process structures can be added to supervisors
- They are done through the **supervisor_bridge** behaviour
- Acts as a supervisor to the process it is connected to
  - No code upgrade
  - No debug functionality
  - Limited supervision

Non OTP-compliant Processes



- C code can be part of the system (C Nodes, Ports, etc.)
- It can be attached to supervision trees

Non OTP-compliant Processes

- Erlang processes can be made to act as behaviours
  - Implemented for performance reasons
  - Implemented in systems without OTP
- These processes can be connected to the supervision tree
- Use the **proc_lib** module to spawn your processes
- Handle system messages in the **sys** module
- The sys debug/stats options can be added

Supervisors

- Supervisors
- Supervisor Example
- Generic Supervisor
- Dynamic Children
- Non OTP-compliant Processes