Code Change in Erlang

Lennart Öhman
Sjöland & Thyselius Telecom AB

lennart.ohman@st.se

Sjöland&Thyselius
Creative Software Solutions

Code Change in Erlang

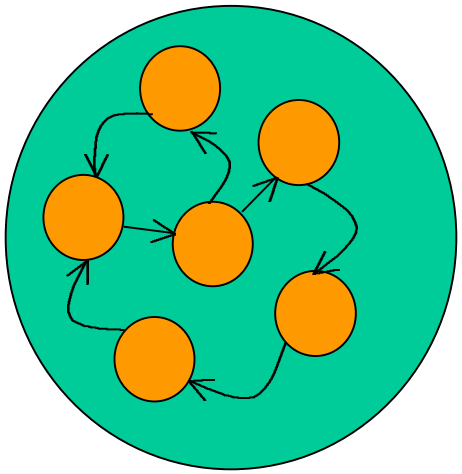
”We like to upgrade all code at the same time.”

and

”There is no inherent way to identify involved processes.”

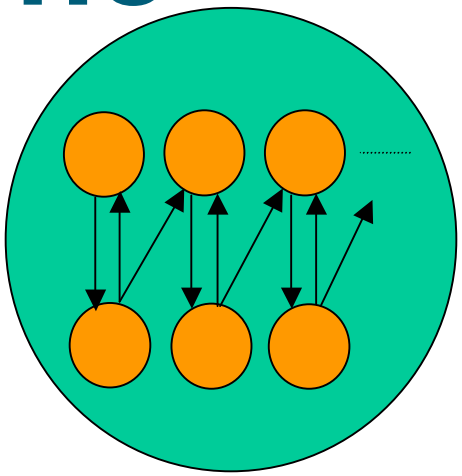
Process states

- Code modules
- Internal data structures
- Relevant external data structures

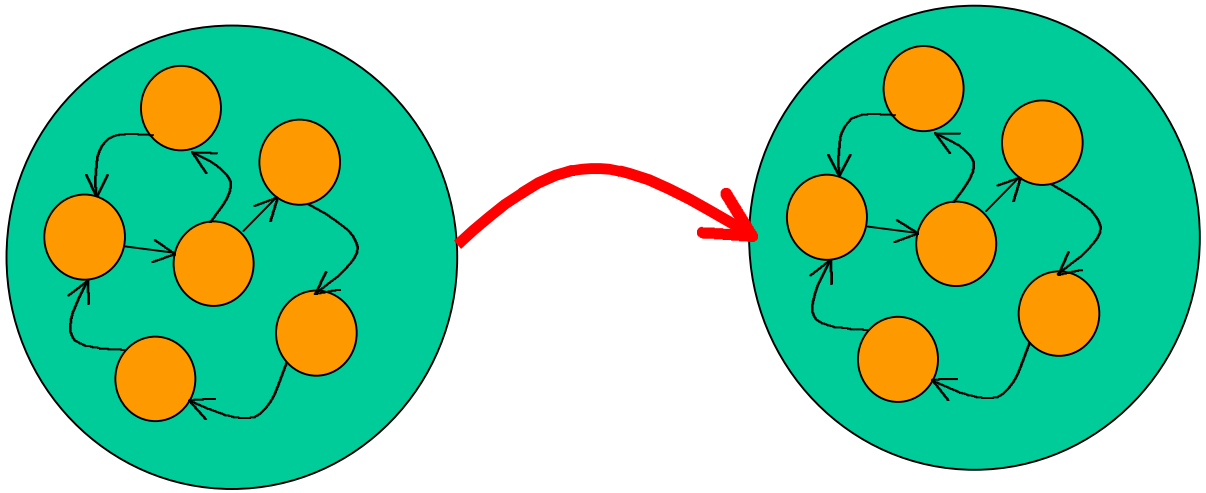


Many states can be grouped into one

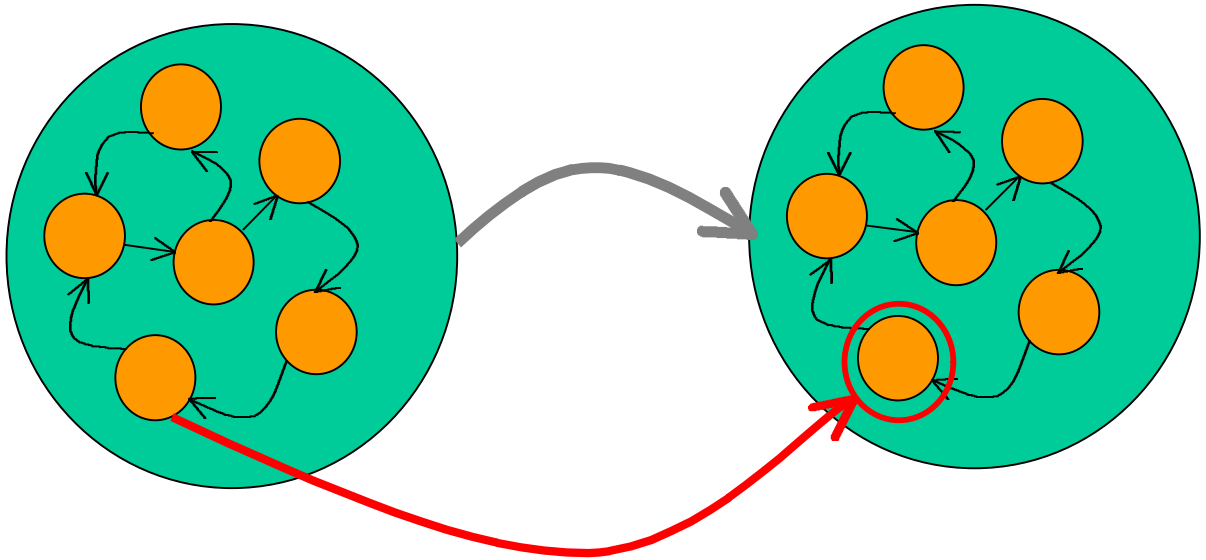
```
f(D) ->
   io:format(“Date:~p~n”,[D]),
   f(date()).
```



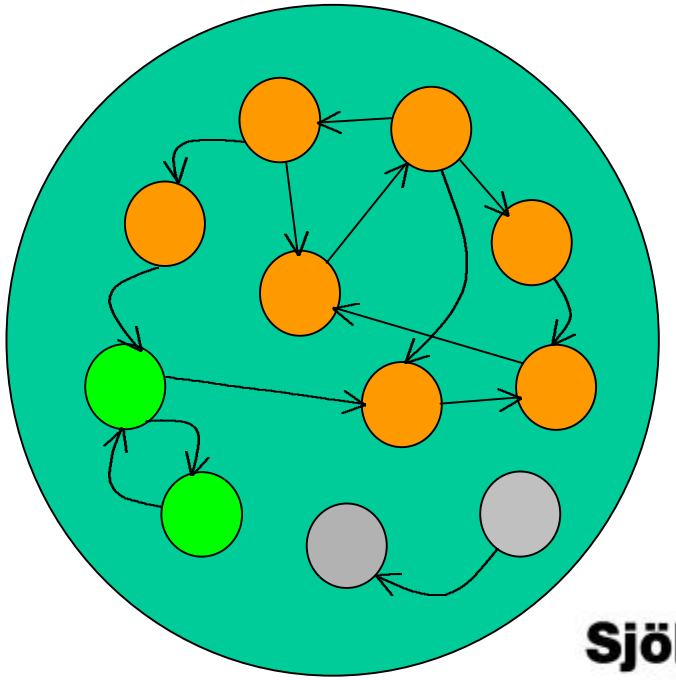
Dependencies between processes



In a certain state a process depends on another process



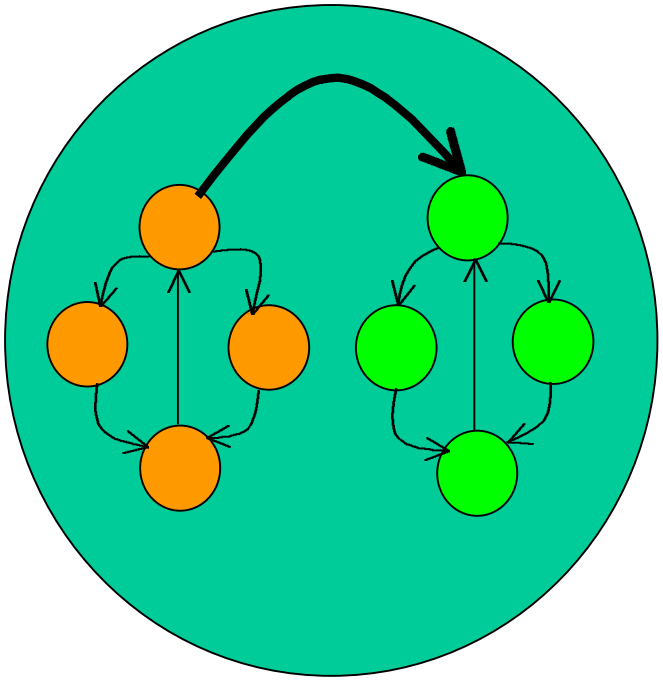
Changing code is adding and/or removing process states



Sjöland&Thyselius
Creative Software Solutions

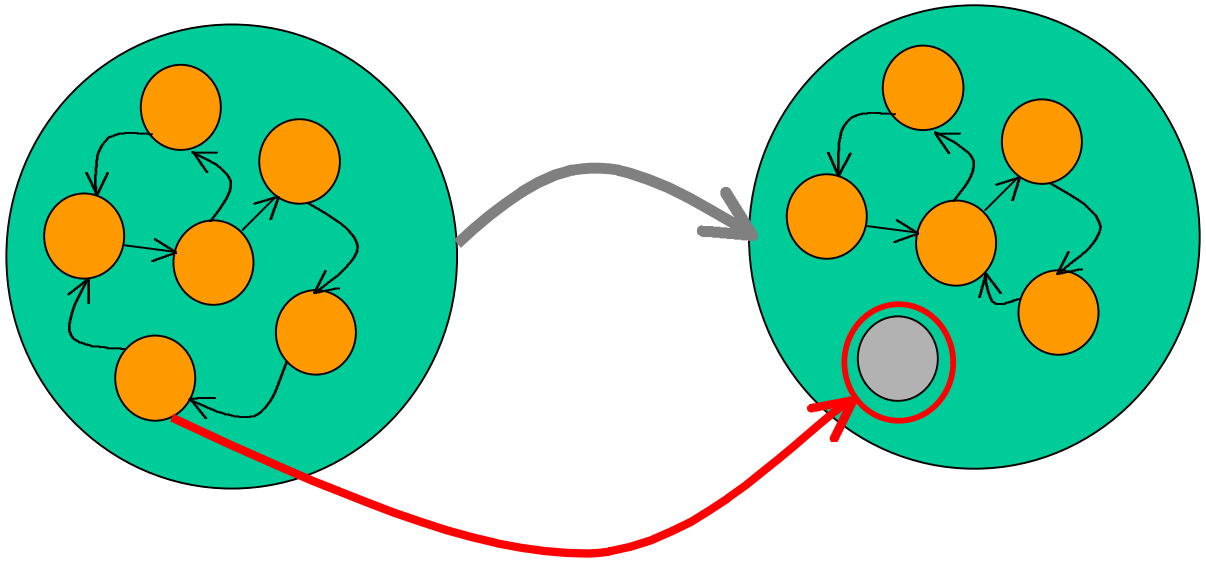
A process may have to be made to change its state

- Changing to a new internal datastructure means shifting to a new set of states

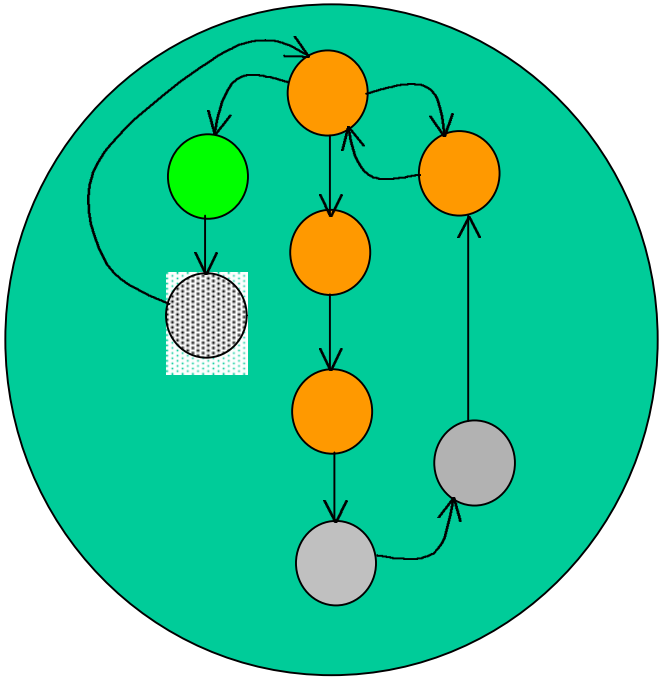


Sjöland&Thyselius
Creative Software Solutions

A process dependency may become exhausted



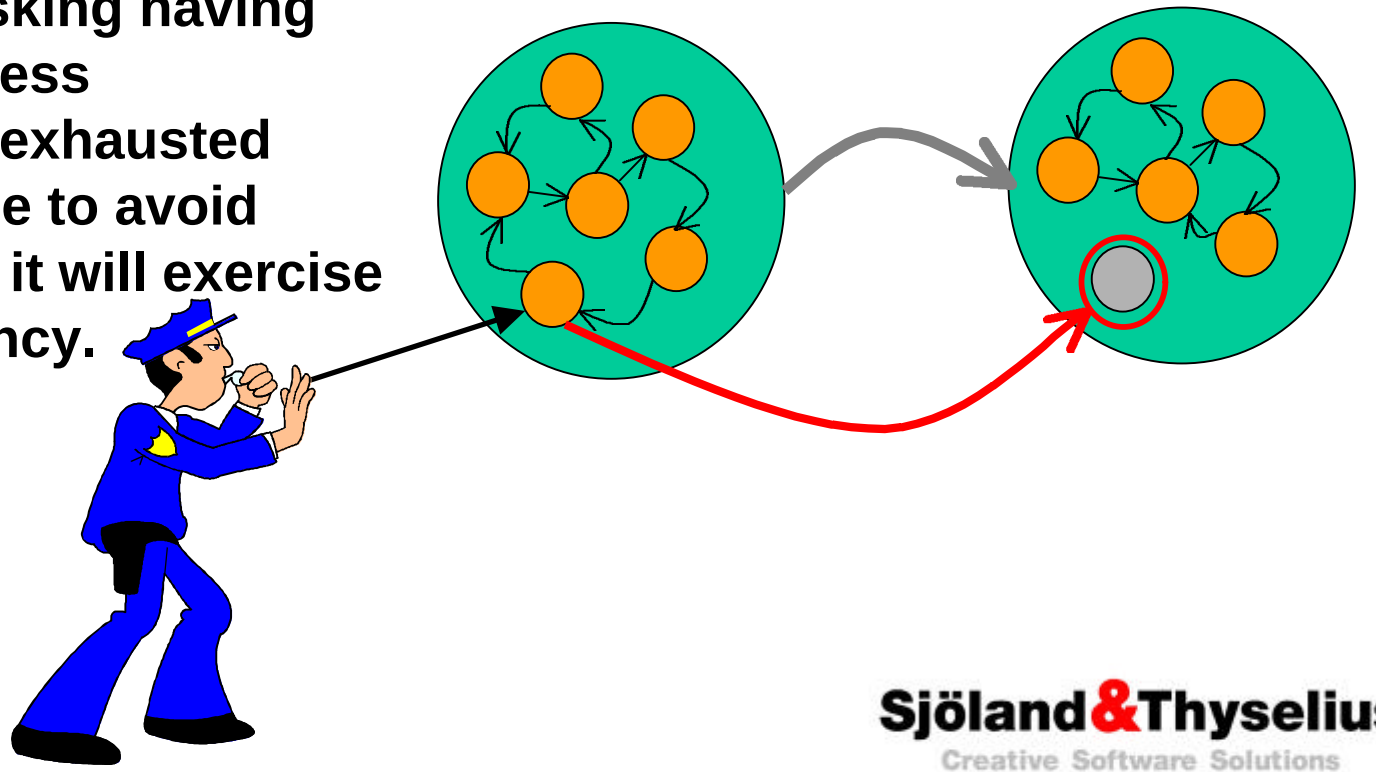
A process may enter a dead-end code path



Sjöland&Thyselius
Creative Software Solutions

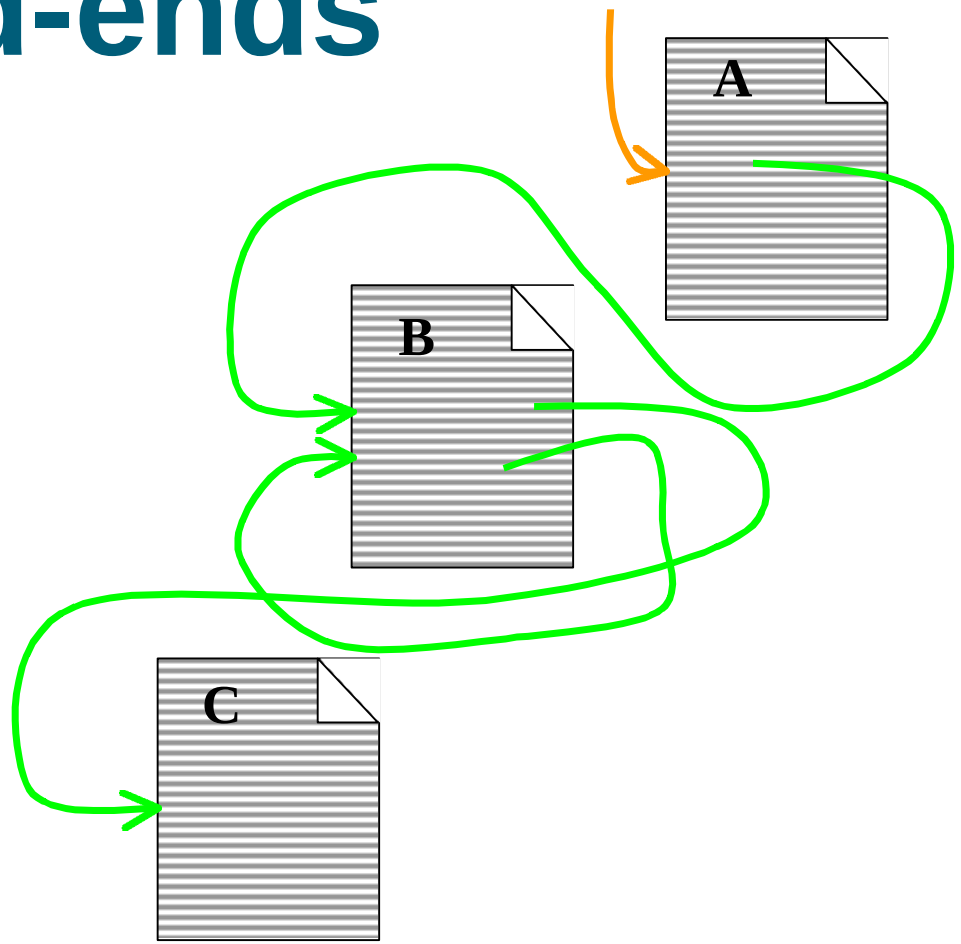
Avoid exhausting a dependency

- A process risking having an inter-process dependency exhausted must be made to avoid states where it will exercise the dependency.



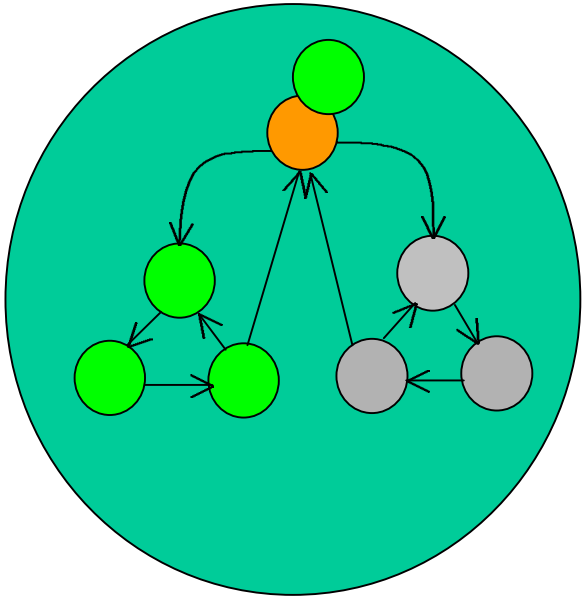
Avoiding dead-ends

- If the new code modules are backwards compatible and new code is without cyclic references (between the modules), they may be inserted in a ”clever” order.



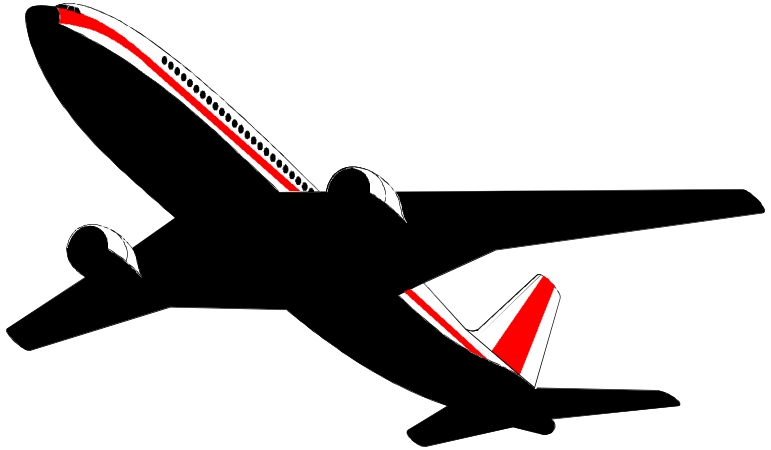
Avoiding dead-ends

- **Being backwards compatible means not depending on whether reaching the new states from an old or a new state (implemented in another module)**



Avoiding dead-ends

- **Try to make modules backwards compatible – leave old states.**



Sjöland&Thyselius
Creative Software Solutions

Avoiding dead-ends

- **Otherwise – processes must be made avoid states leading to dead-ends.**

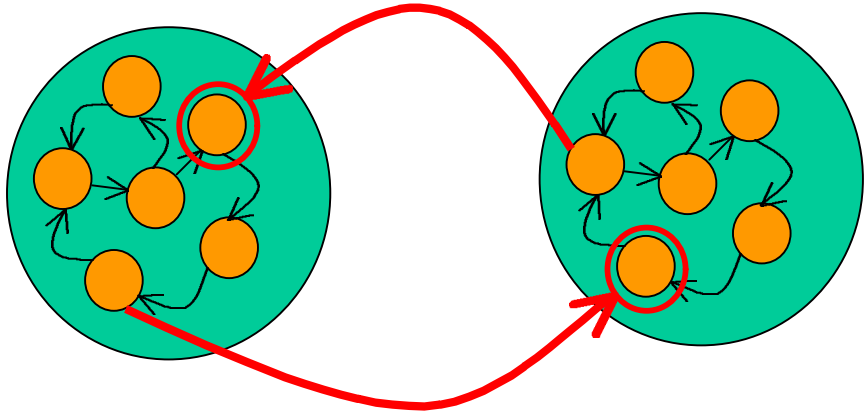
Sjöland&Thyselius
Creative Software Solutions

Suspending processes

- Entering a special state from which it can not transition to states leading to dependencies.
- A state from which it can not enter a path leading to a dead-end.

Processes must be suspended atomically

- **Processes having cyclic dependencies may end up in dead-lock if suspended one at a time.**

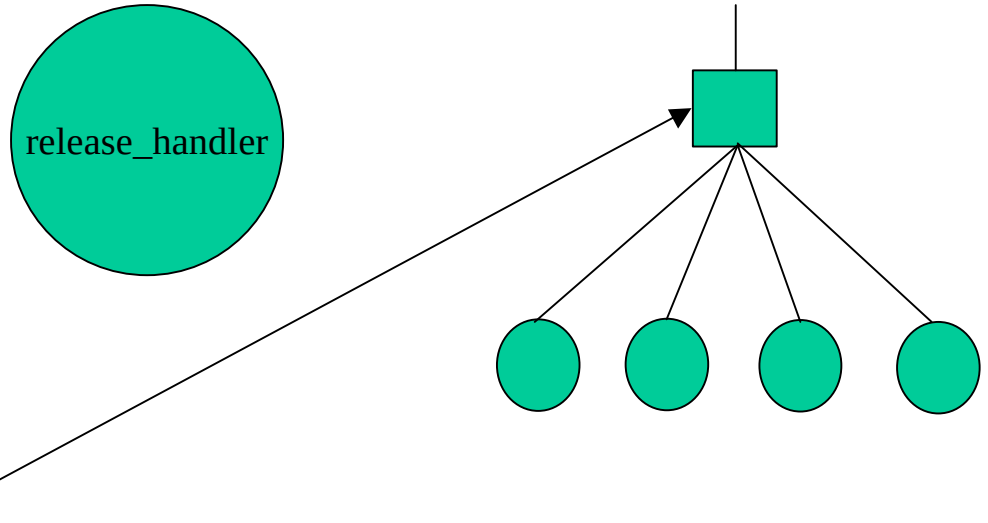


Upgrade – how to

- Find processes which must be suspended.
- Choose as few as possible.
- Suspend them all at the same time.
- Insert new code.
- Resume suspended processes.

Identifying processes

- **In OTP processes are identified by modules they use, specified in their children specification**

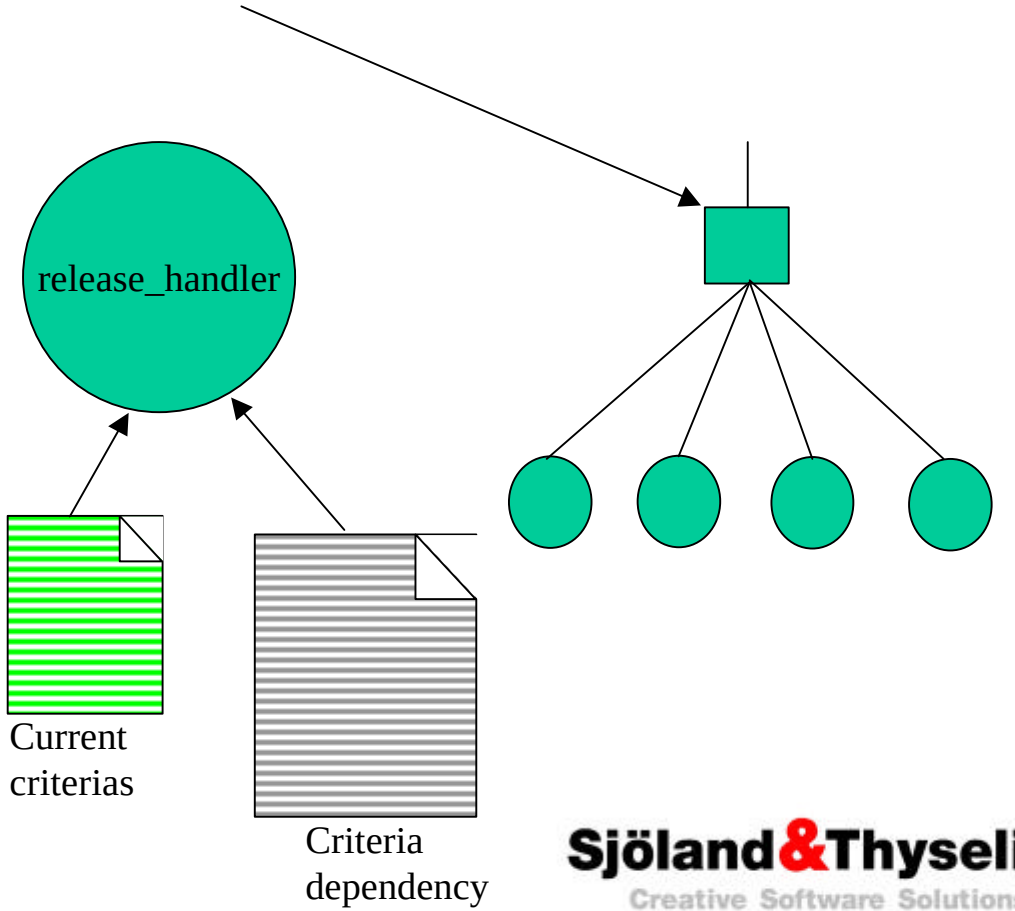


[...,{my_process,{m1,f1,[]},permanent,10000,worker,[mod1,mod2,...]},...]

Identifying processes

[...,{my_process,{m1,f1,[]},permanent,10000,worker,[{**type,a**}]},...]

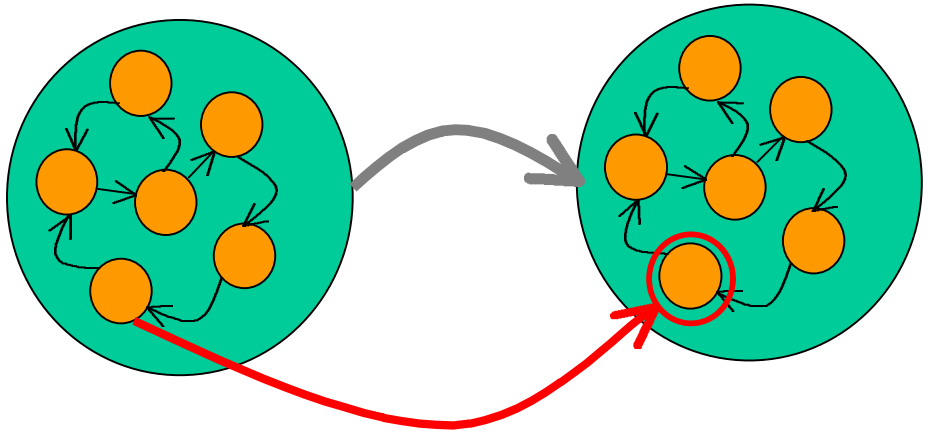
- **Introduce process type-names in the children specifications.**
- **Introduce criteria dependency des-cription with each release.**
- **Provide the release_handler with <u>current</u> criterias.**



Sjöland&Thyselius
Creative Software Solutions

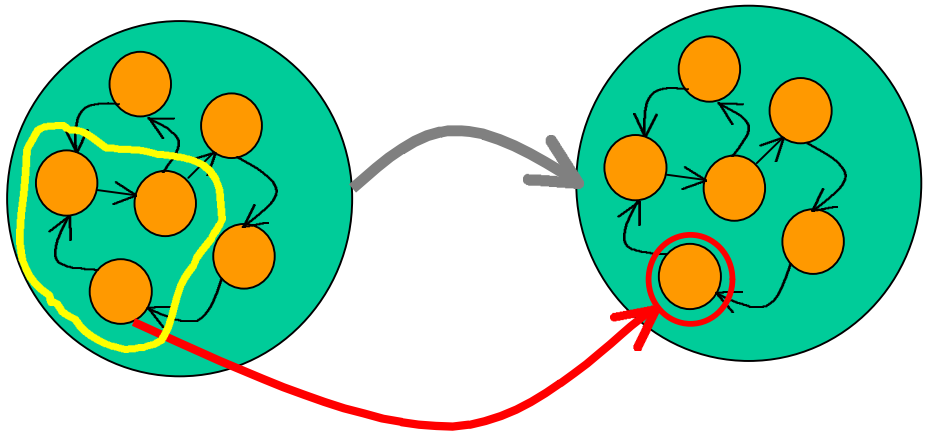
Dependency criterias

- Define situations when dependencies exists.
- Defined for named processes or processes of certain types.



Dependency criterias

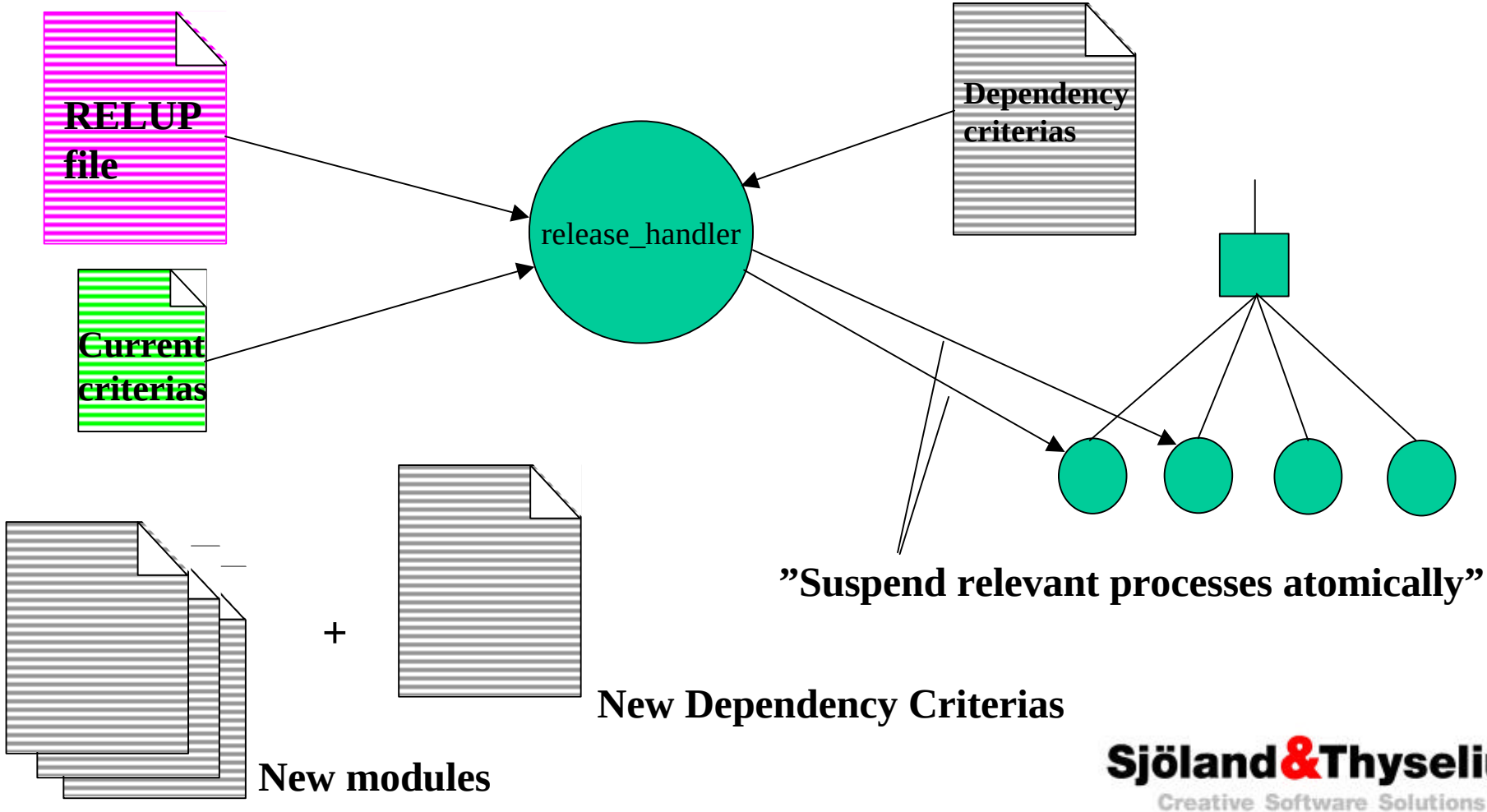
- **Include more states than necessary if it results in a less difficult criteria to formulate.**



Current criterias

- **The current situation which tells whether a process will risk visiting the process-states guarded by the dependecy criteria.**
- **We like to avoid suspending processes if not necessary.**

Release upgrading



Sjöland&Thyselius
Creative Software Solutions