

Short Introduction to McErlang: a Model Checker for Erlang Programs

Álvaro Fernández Díaz

Universidad Politécnica de Madrid

alvaro.fernandez@upm.es



Contents

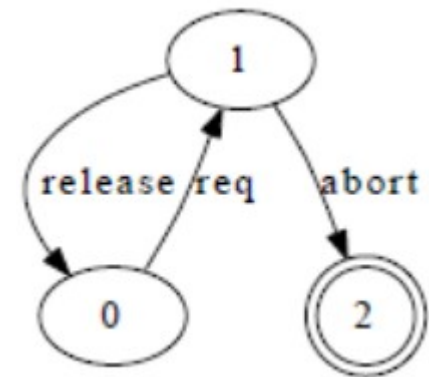
- McErlang key features
- Model Checking
- Program states in McErlang
- McErlang Preprocessing
- Some McErlang Structures
- Download and Installation
- Hands-on McErlang

Some Facts

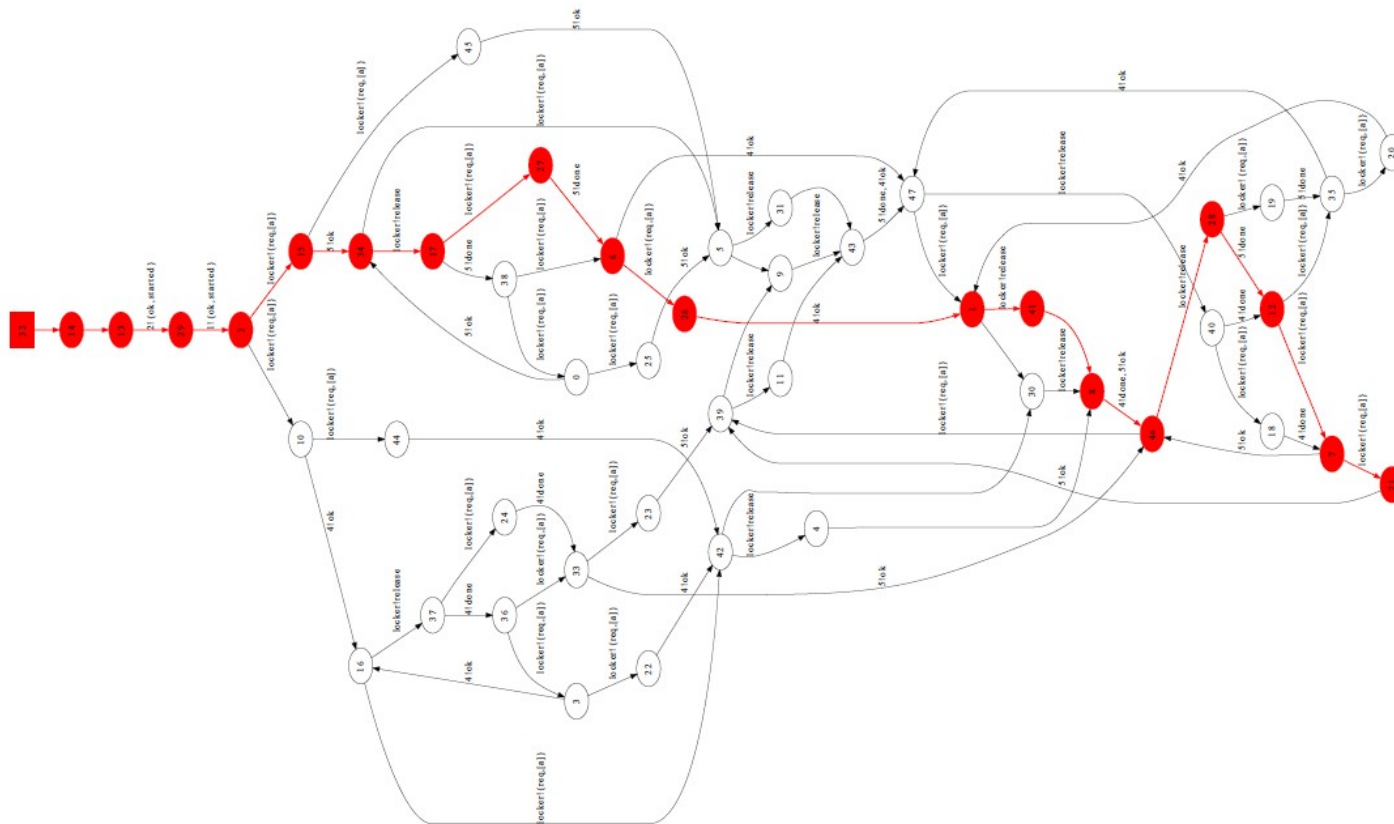
- Developed at Universidad Politécnica de Madrid and Chalmers University of Technology by Lars-Åke Fredlund, Hans Svensson, Clara Benac Earle and Alvaro Fernández Díaz.
- McErlang is useful for checking **concurrent software**, **not** for checking **sequential software**.
- **Erlang runtime** for handling communication and processes has been **replaced** with a new one written in Erlang.
- “**Everything in Erlang**” approach
 - Erlang as input language (**program as model**)
 - McErlang is implemented in Erlang
 - Properties written in Erlang

Model Checking

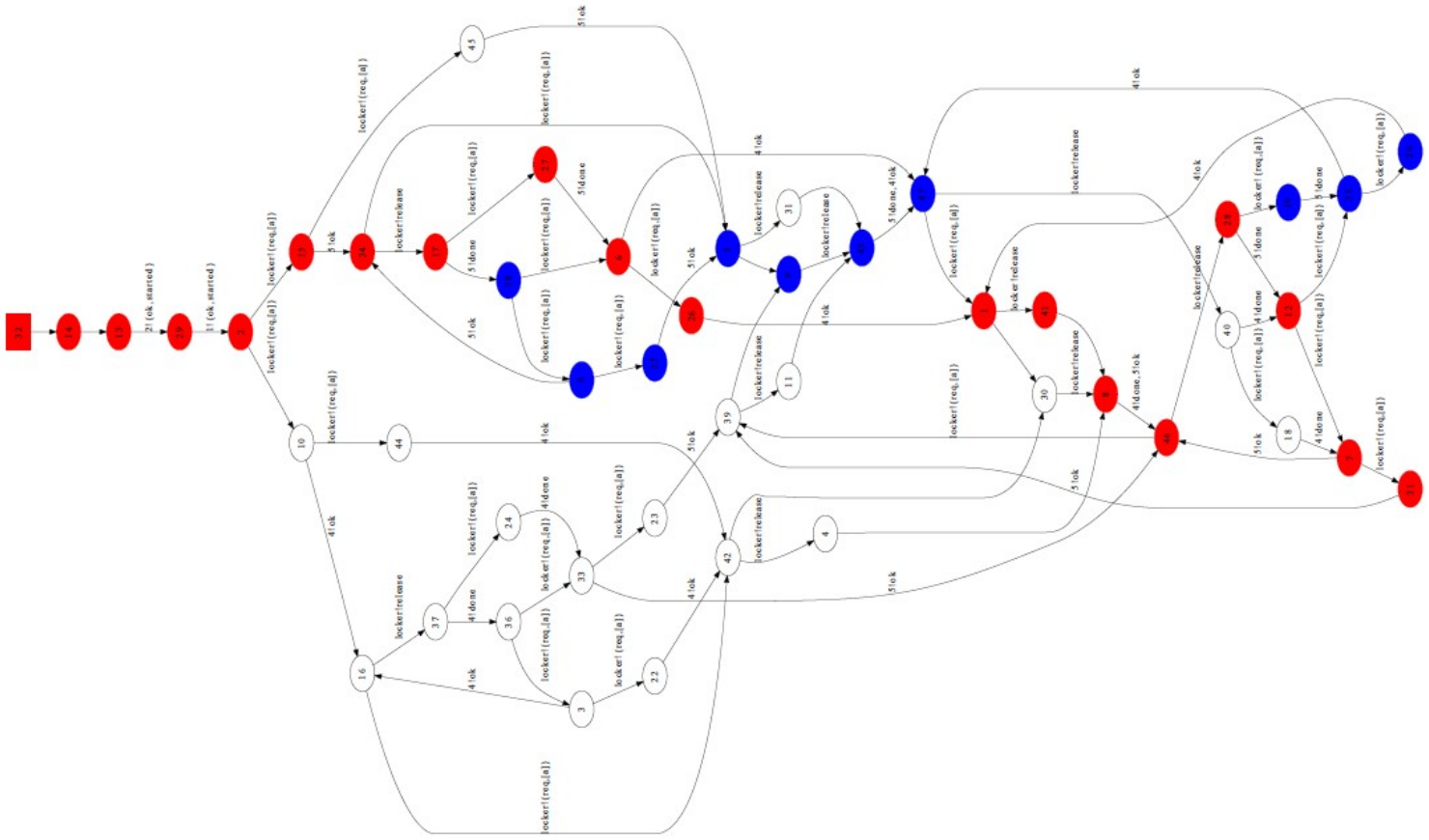
- Run a program in a controlled manner in order to visit all its states (generated through process interleaving)
- State space as a finite transition graph:
 - Nodes are program states (snapshots)
 - Edges are computations from one state to another
- Properties as automata that run in lockstep with the program.



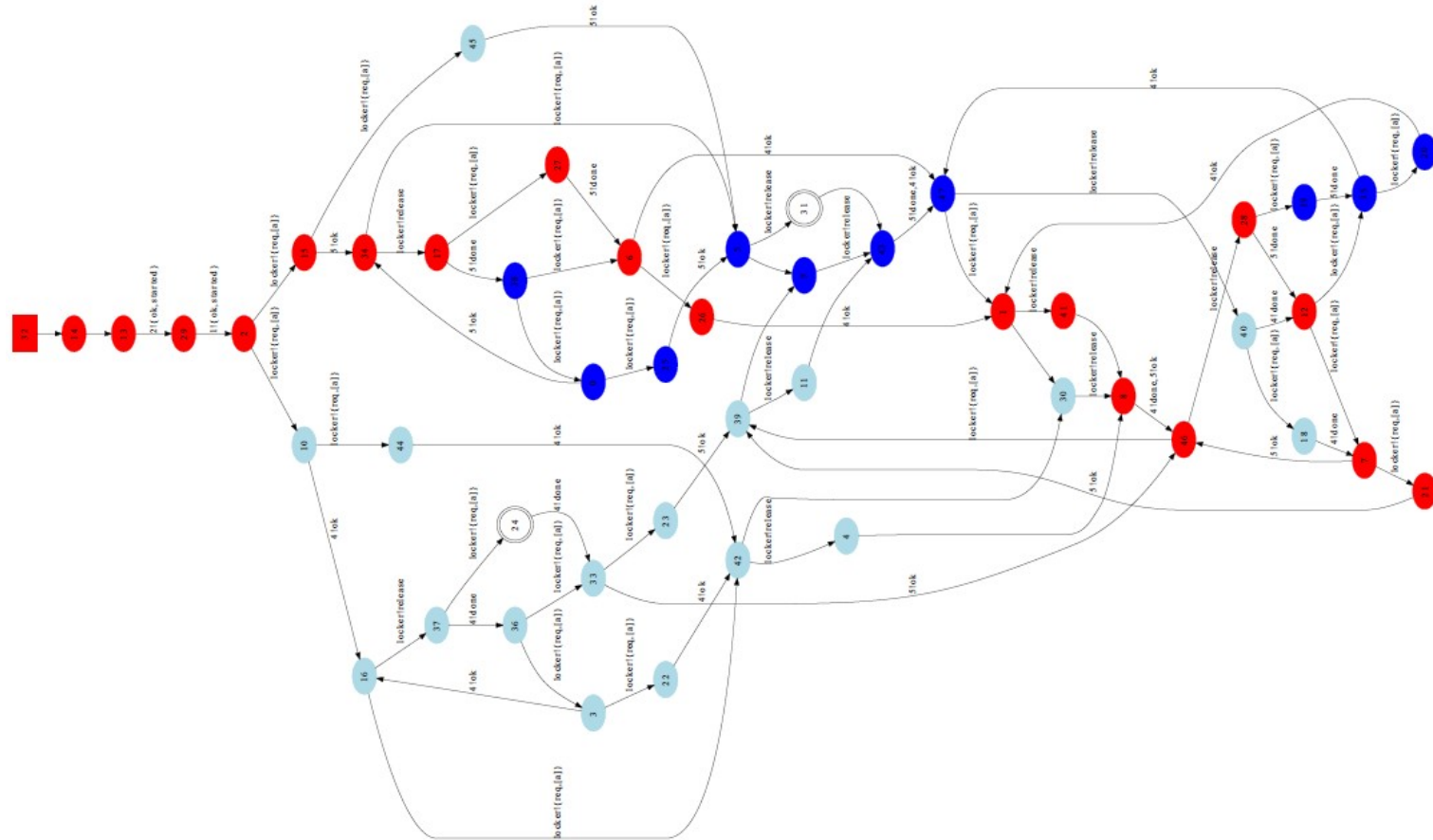
One Single Run



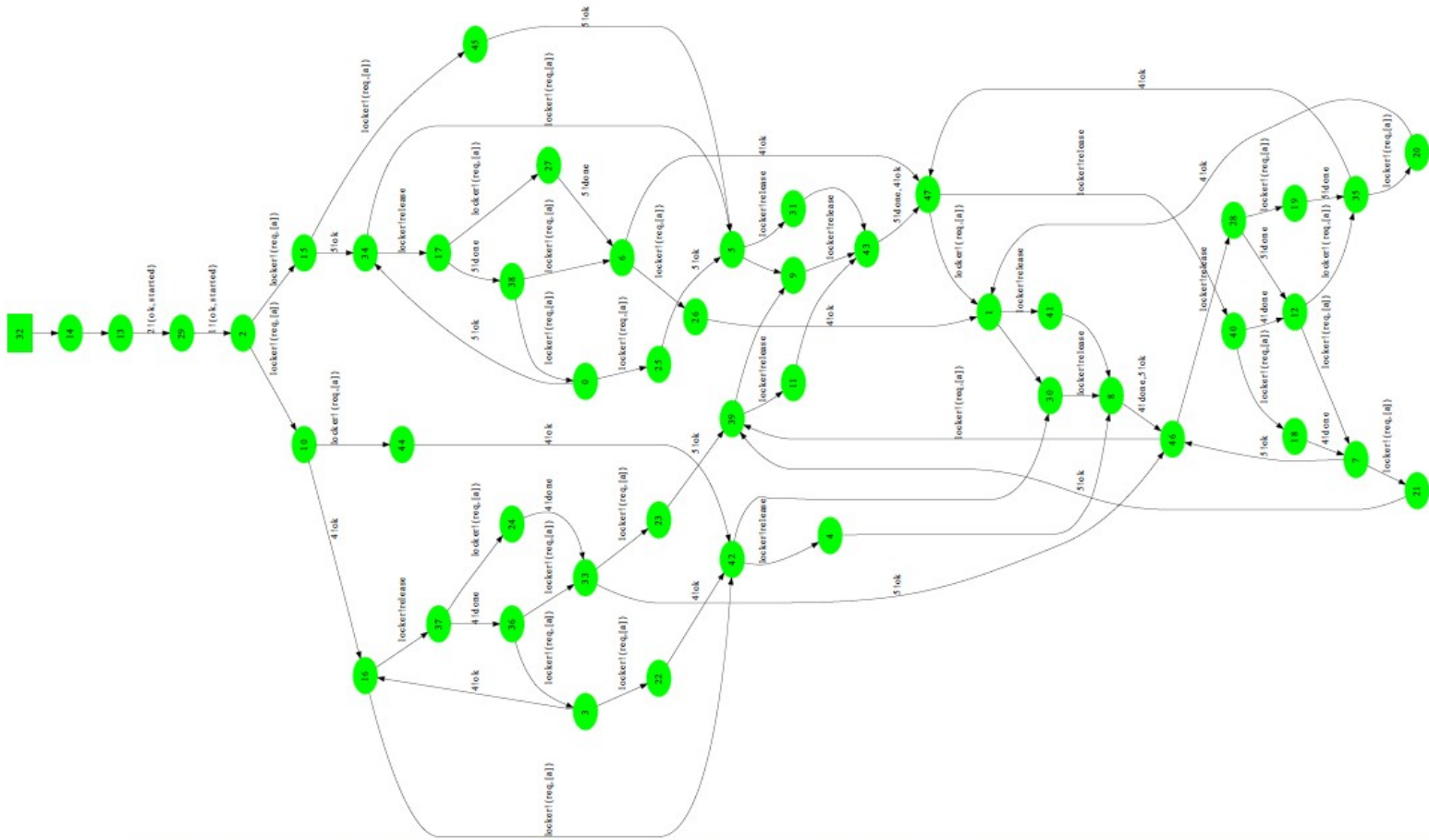
Second Run



Random Testing

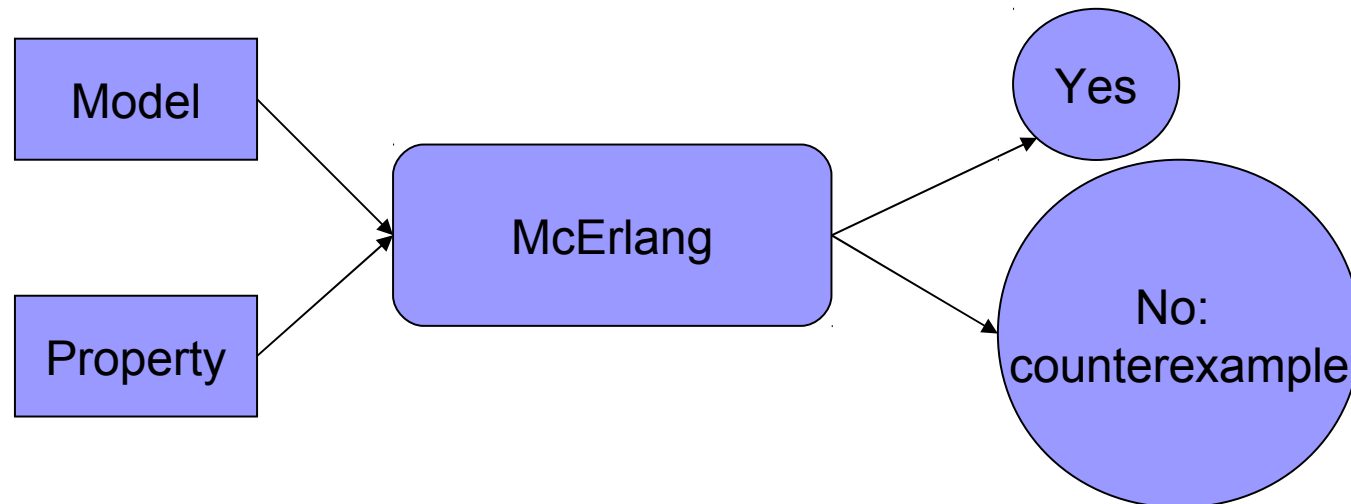


Model Checking: 100% coverage

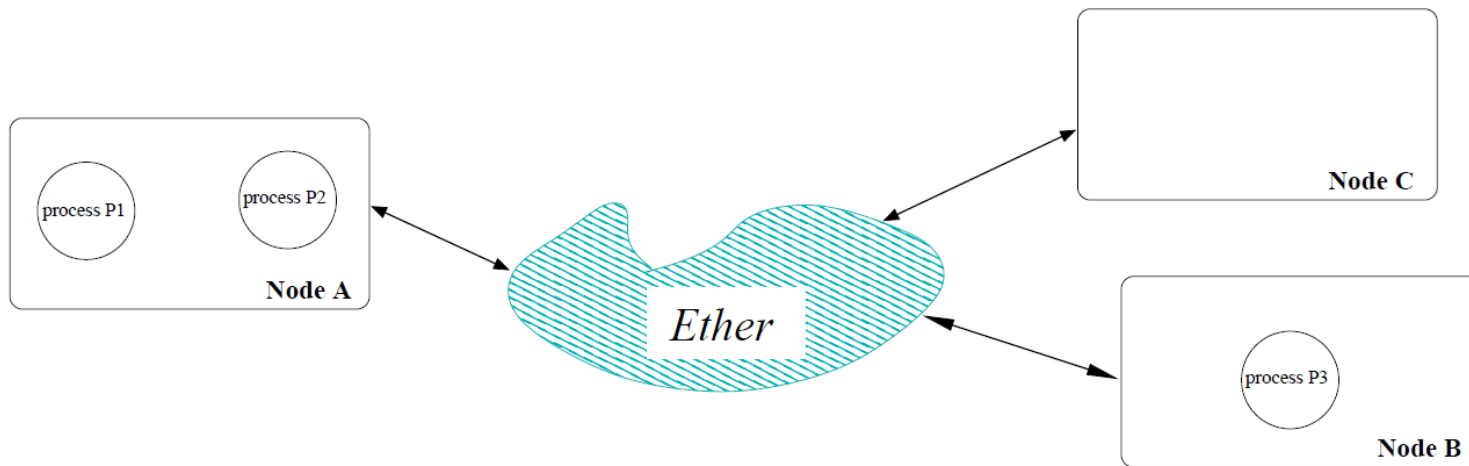


McErlang: an Erlang Model Checker

- Program as model
- Allows extension, Customizable
- Correctness properties specified in Erlang

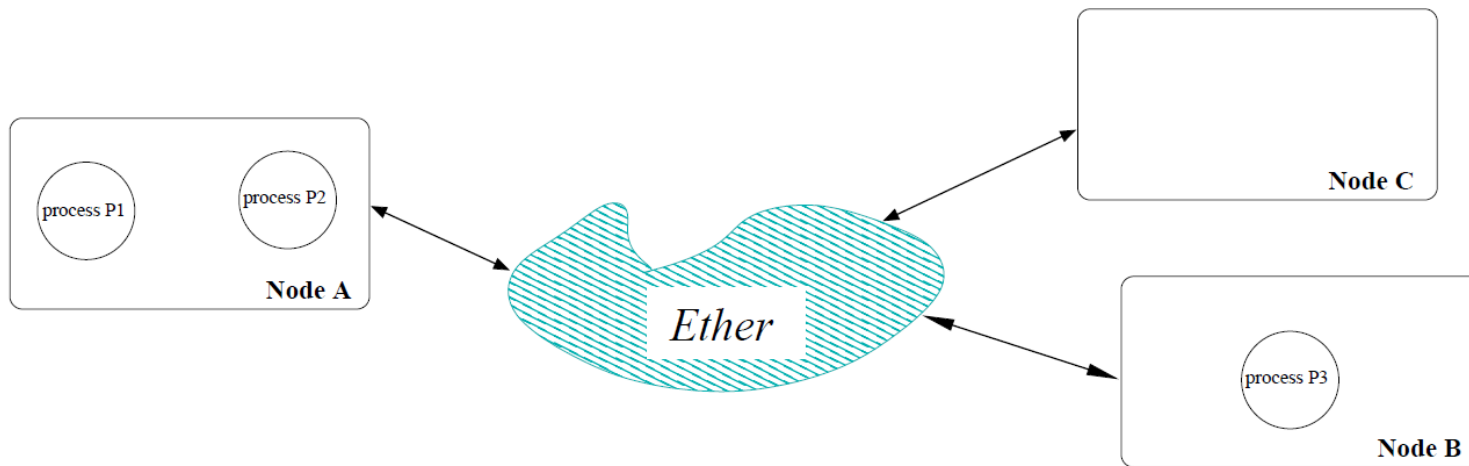


McErlang Program States



- What must be stored?
 - nodes
 - processes
 - messages in transit (the ether),
 - process mailboxes,

McErlang Program States



- States are stored in memory
 - Possibility to load previous states
- Difficulties:
 - High memory capacity required for very big models (state explosion problem).
 - Disk read/write slow Model Checking process



Gathering system information

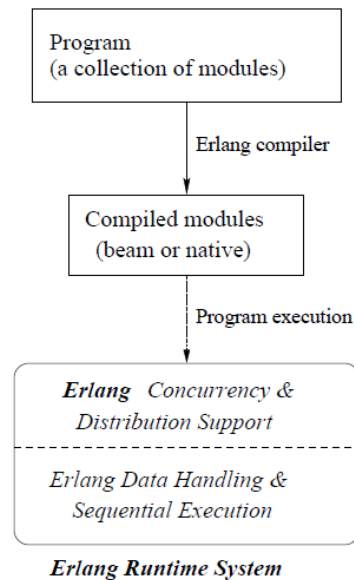
- New runtime system implementation for concurrent part and reuse of old runtime system to execute code with no side effects (process communication, creation and handling)
 - Gain control of execution flow
 - Access program states

McErlang Preprocessing

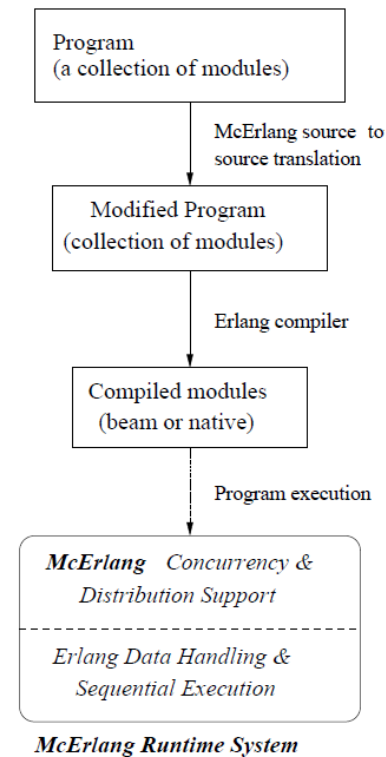
- Code has to be preprocessed by McErlang “compiler” prior to its execution.
- Some changes applied:
 - API: calls to `erlang:spawn` rewritten to calls to `mcerlang:spawn`
 - Calls to communication primitives encapsulated in anonymous functions

Workflow Comparison

Normal Erlang Workflow:



McErlang Workflow:



McErlang extensions to Erlang

- Nondeterminacy:

- `mce_ertl:choice`

([fun () -> Pid ! hello end,
 fun () -> Pid ! bye end]).

- Convenience:

- `mcerlang:spawn(new_node, fun () -> Pid ! hi end)`

- Creates node *new_node* if it does not exists

Expressing program properties

- They are represented by McErlang Monitors
- Monitors are run in lockstep with the program
- Monitors can be built
 - Manually
 - Safety monitors:
- *stateChange(State, MonitorState, Actions) ->*
-
- Automatically from LTL properties
- "always (P => eventually Q)"

Monitors Observational Power

- Program actions such as e.g. sending or receiving a message
- Program state such as e.g. contents of process mailboxes, name of registered processes.
- Programs can be instrumented with special “probe actions” that are easy to detect in monitors.
- Programs can be instrumented too with special “probe states”, which are persistent (actions are transient)

Some predefined monitors

- Checking that there is at least one non-deadlocked process:
 - {mce_mon_deadlock, PrivateState}
- Checking that all queues contain at most MaxQueueSize elements:
 - {mce_mon_queue, MaxQueueSize}

Some Model Checking Algorithms

- Executing a single program trace:
 - {mce_alg_simulation, PrivateState}
- Checking a safety property:
 - {mce_alg_safety, PrivateState}
- Checking any LTL property:
 - {mce_alg_buechi, PrivateState}

Some Model Checking Options

- Included in record *mce_opts*
- Compute the shortest counterexample:
 - `shortest = true | false`
- Stop immediately if an uncaught exception occurs:
 - `fail_on_exit = true | false`

Some Model Checking Options

- Avoid triggering non-zero timers (receive-after clauses) if there are actions enabled:
 - `is_infininitely_fast = true | false`
- Enable I/O actions outside McErlang:
 - `sim_external_world = true | false`

Running programs under McErlang

- **Starting McErlang:**
- *mce:start*
- *(#mce_opts{program={Module, Fun, Args},*
- *algorithm={Module, InitArgs},*
- *monitor={Module, InitArgs})*

Running programs under McErlang

- **Starting McErlang:**

- *mce:start*

- *(#mce_opts{program={Module, Fun, Args},*
- *algorithm={Module, InitArgs},*
- *monitor={Module, InitArgs})*

- Model Checking run for program `example:start(1,2)`

- *mce:start*

- *(#mce_opts{program={example, start, [1, 2]},*
- *algorithm={mce_alg_safety, void},*
- *monitor={mce_mon_test, void})*

Time for some practice!

©2001 Shannon Burns

www.shannonburns.com



Download and Installation

- Download McErlang source code:

svn checkout https://babel.ls.fi.upm.es/svn/McErlang/trunk McErlang

- Compile Mcerlang:
(cd McErlang; make)

Download and Installation

- Include McErlang scripts to command path:

```
export PATH=~/.McErlang/scripts:$PATH
```

Setting up Emacs Interface

- Edit file: `$HOME$/.emacs`
- Add the following piece of code:

```
;; McErlang Emacs Mode -- Configuration Start
(add-to-list 'load-path PathToMcErlangFile*)
(autoload 'mcerl-erlang-mode-hook "mcerl-ext" "McErlang Mode" t)
(add-hook 'erlang-mode-hook 'mcerl-erlang-mode-hook)
(setq mcerl-max-menu-length 30)
;; McErlang Emacs Mode -- Configuration End
```

* Absolute path of file `mcerl-ext.el` (for instance
"`/home/user/McErlang/McErlang_Emacs_Mode`")

Program to verify: multi-agent system

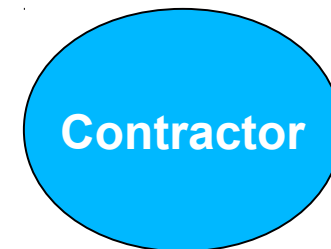
- **Manager**

- Tries to allocate a set of tasks
- Similar to an “auctioneer”

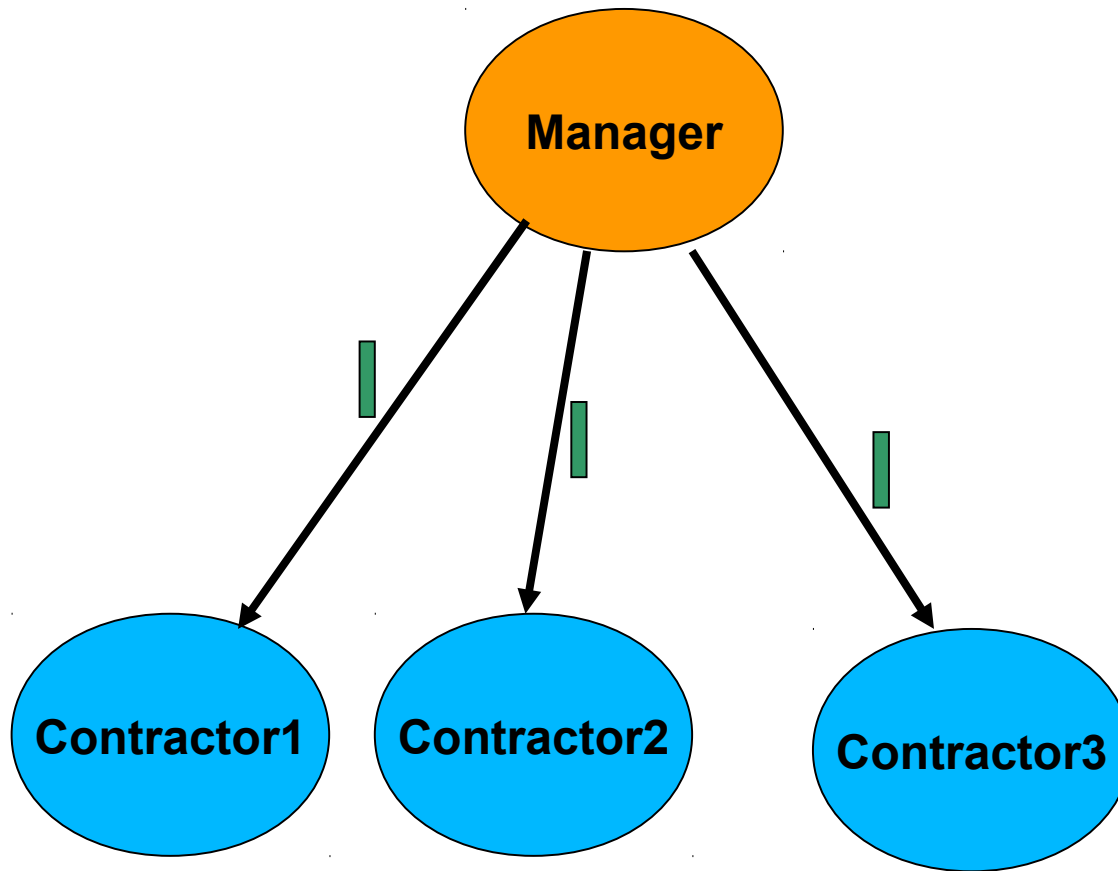


- **Contractor**

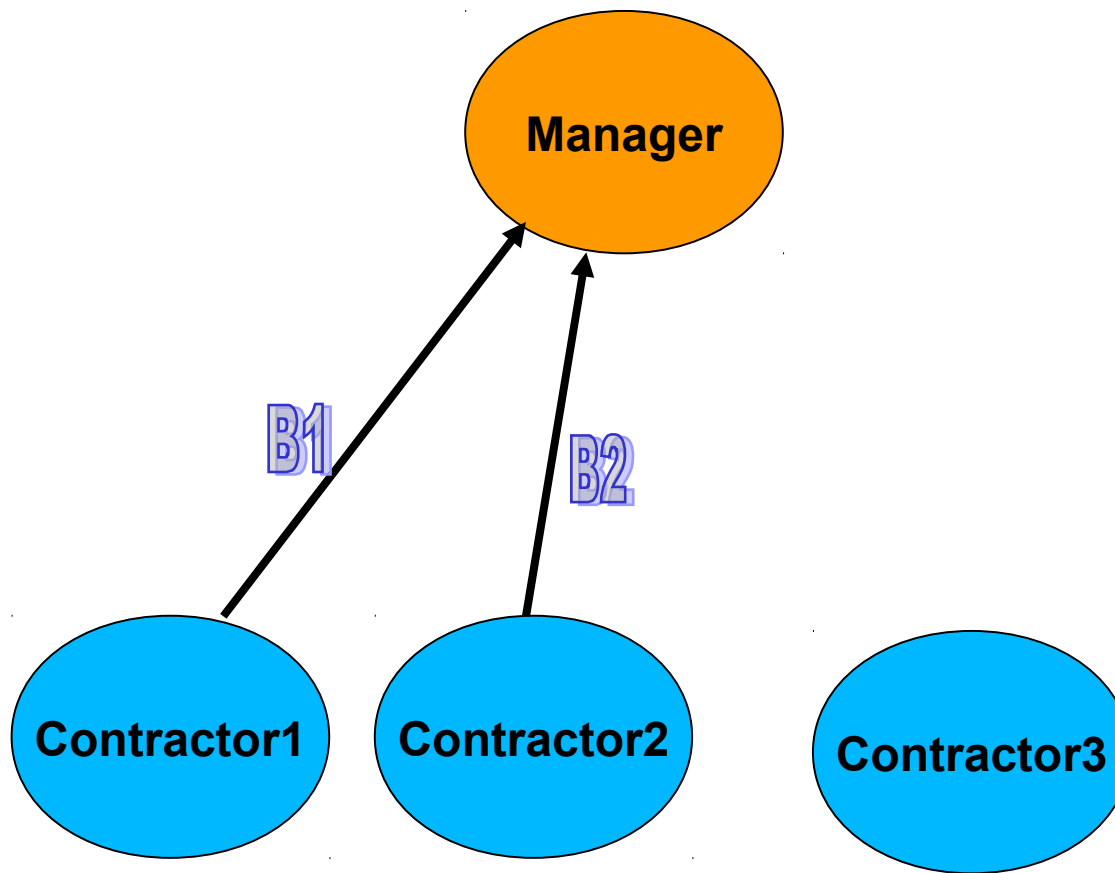
- Tries to get tasks allocated to itself
- Similar to a “bidder”



CNP Extension: call for bids

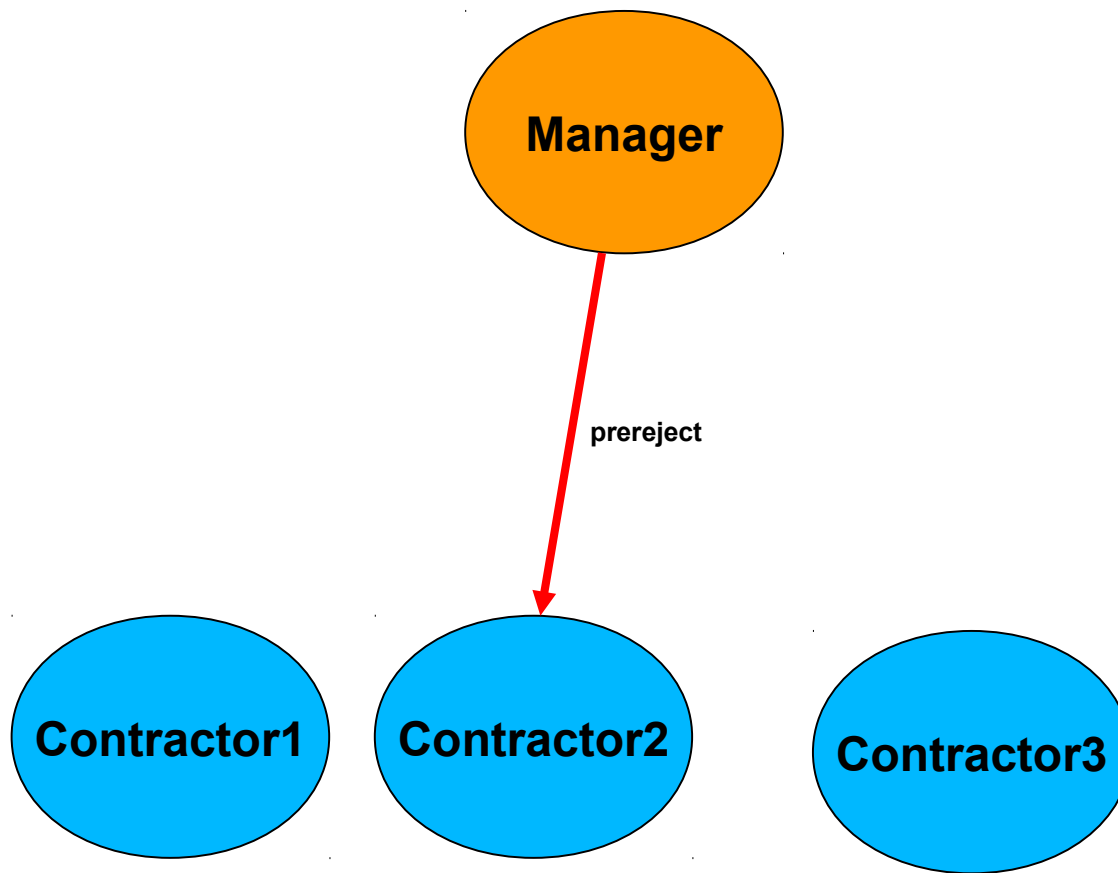


CNP Extension: prebidding phase

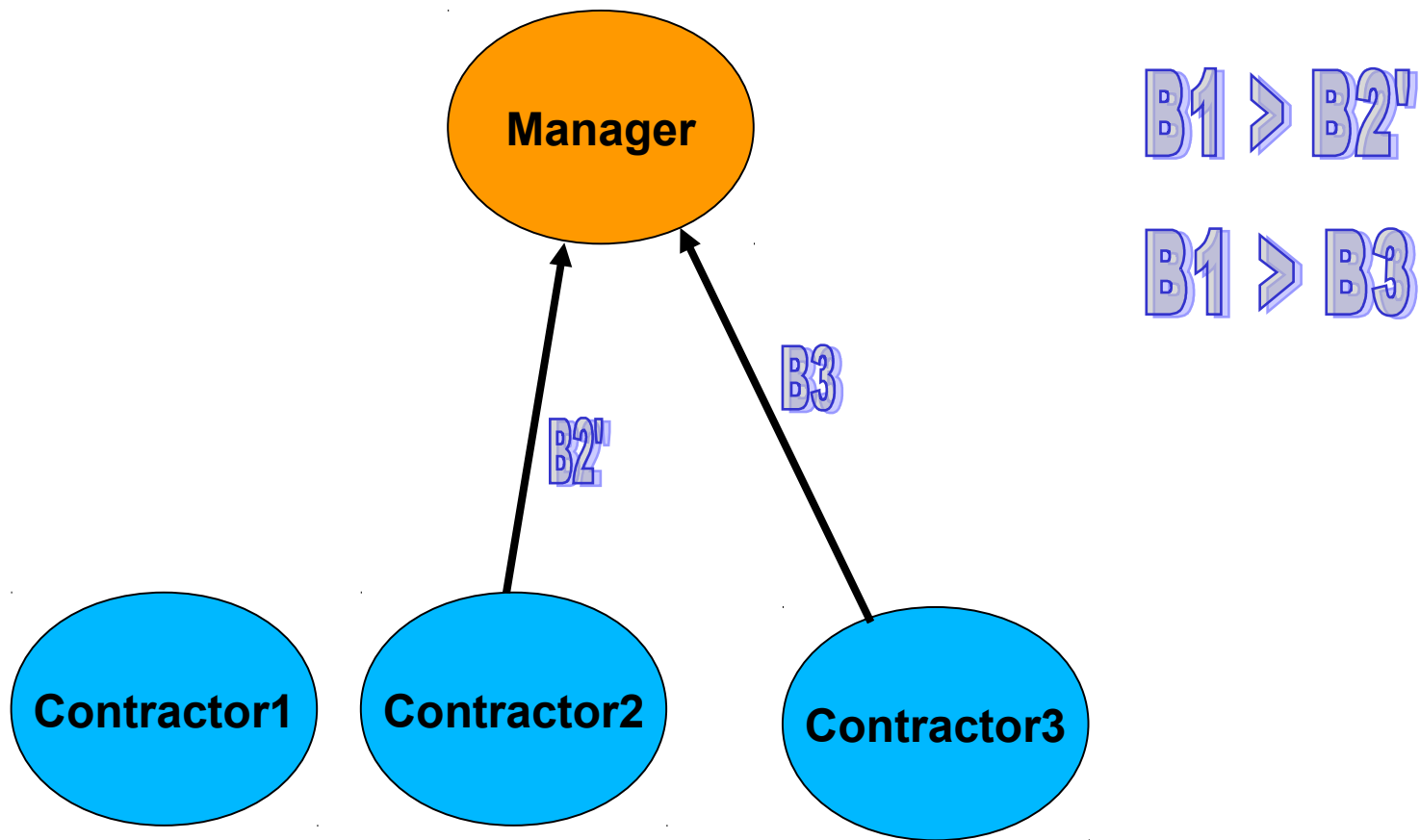


$B1 > B2$

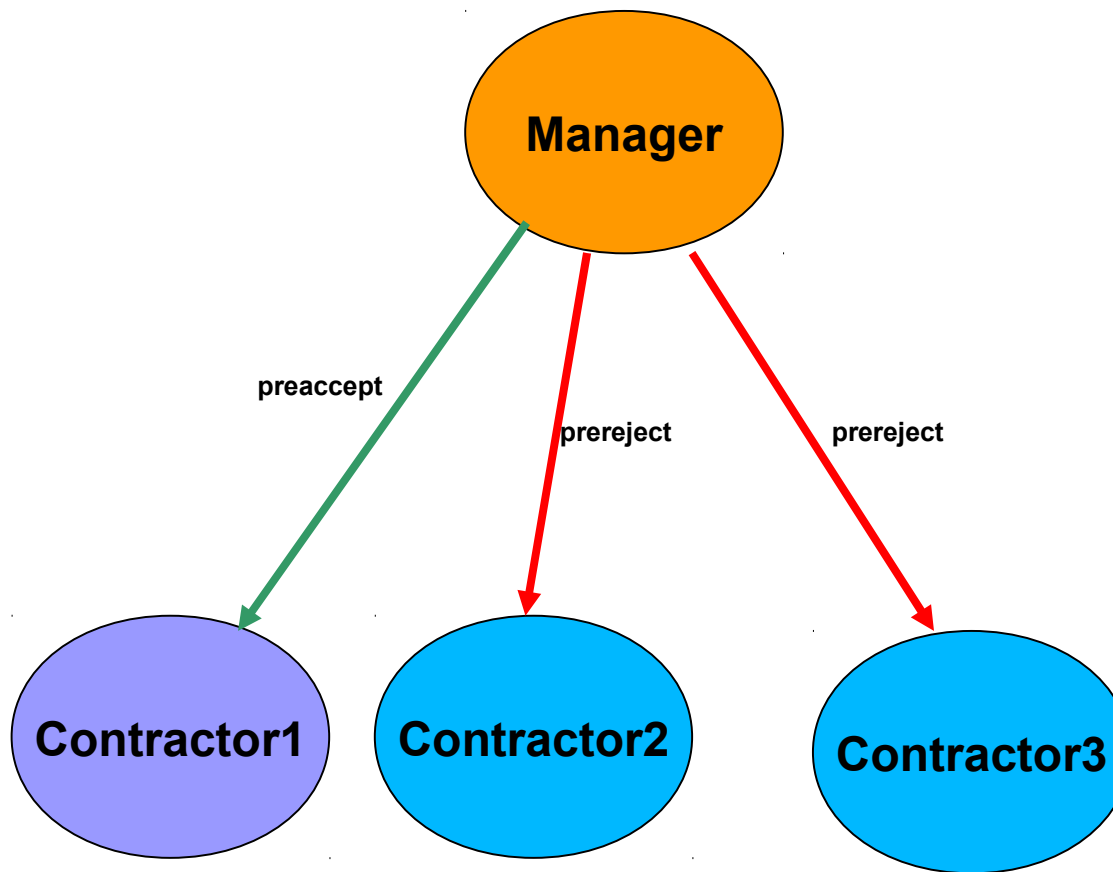
CNP Extension: prebidding phase



CNP Extension: prebidding phase



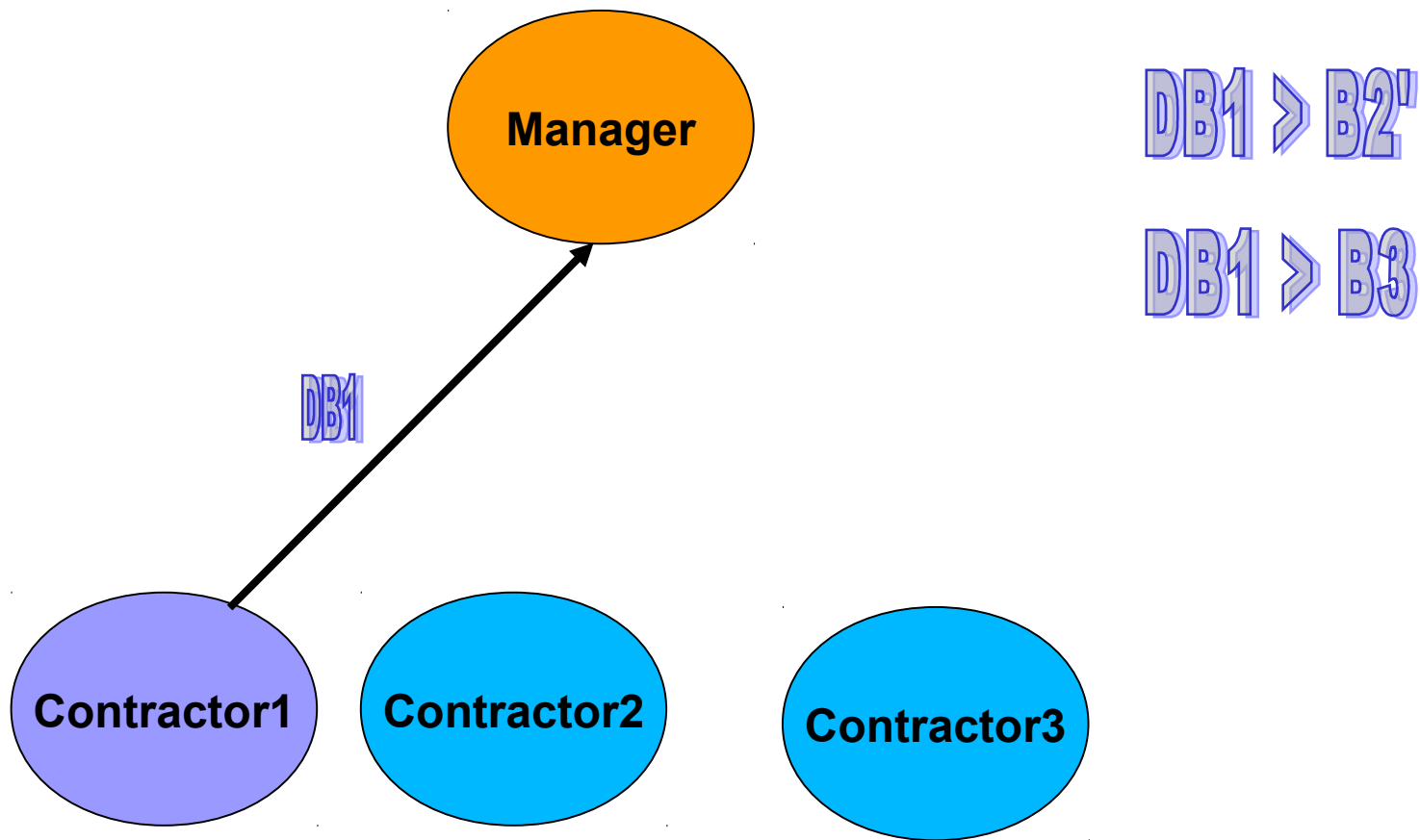
CNP Extension: preassignment phase



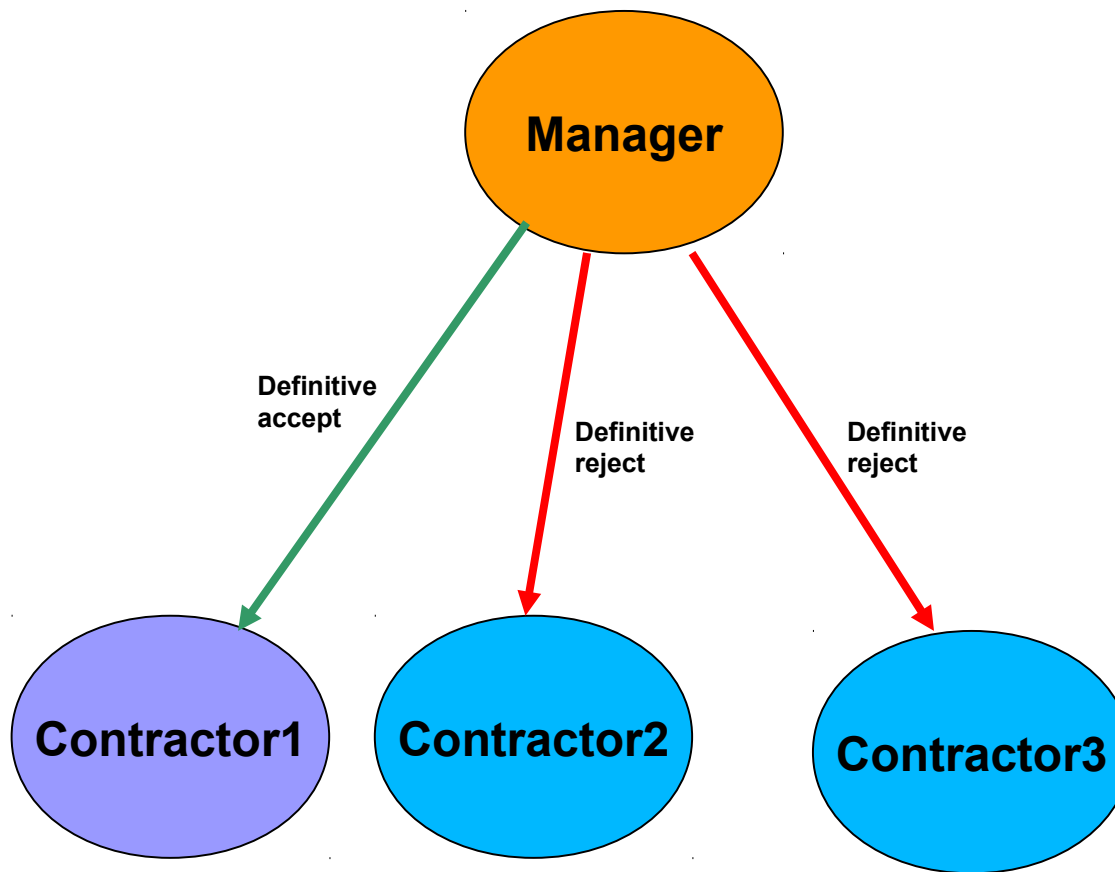
$B1 > B2'$

$B1 > B3$

CNP Extension: definitive bidding phase



CNP Extension: definitive assignment phase



DB1 > B2'

DB1 > B3

Erlang Code

- <http://babel.ls.fi.upm.es/trac/McErlang/wiki/midTermWorkshop>
- Folder EUC2010/code:
 - src/ecnp.erl
 - src/contractor.erl
 - src/manager.erl
 - src/test.erl
 - src/include/records.hrl
- Compile and try under normal Erlang

Compiling under McErlang

- **IMPORTANT** : folder ebin has to exist before compilation.

- Compiling all erlang files

```
mce_erl_compiler -sources *.erl
```

- Starting McErlang:

```
cd ebin
```

```
mcerl
```

Running code under McErlang

- Simulation run:

```
mce:start(#mce_opts{  
  program={test,test0, [N] },  
  fail_on_exit=false,  
  is_infinately_fast=true,  
  algorithm={mce_alg_simulation,void},  
  sim_external_world = true  
} ).
```



Building a deadlock detection monitor

- When is the program deadlocked?



Building a deadlock detection monitor

- When is the program deadlocked?
 - There is at least one process



Building a deadlock detection monitor

- When is the program deadlocked?
 - There is at least one process
 - All processes are blocked

Building a deadlock detection monitor

- When is the program deadlocked?
 - There is at least one process
 - All processes are blocked
 - There are no messages in transit (in “ether”)

Building a deadlock detection monitor

- When is the program deadlocked?
 - There is at least one process (P)
 - All processes are blocked (Q)
 - There are no messages in transit (in “ether”) (Q)
- LTL Property
 - “always ((not Q) or (not P))”

McErlang Debugger

- Eases the error identification process.
- Provides means to access program execution traces (normally the ones leading to a property counterexample)
- Starting McErlang Debugger for last counterexample identified:

```
mce_erl_debugger:start(mce:result())
```

McErlang Debugger

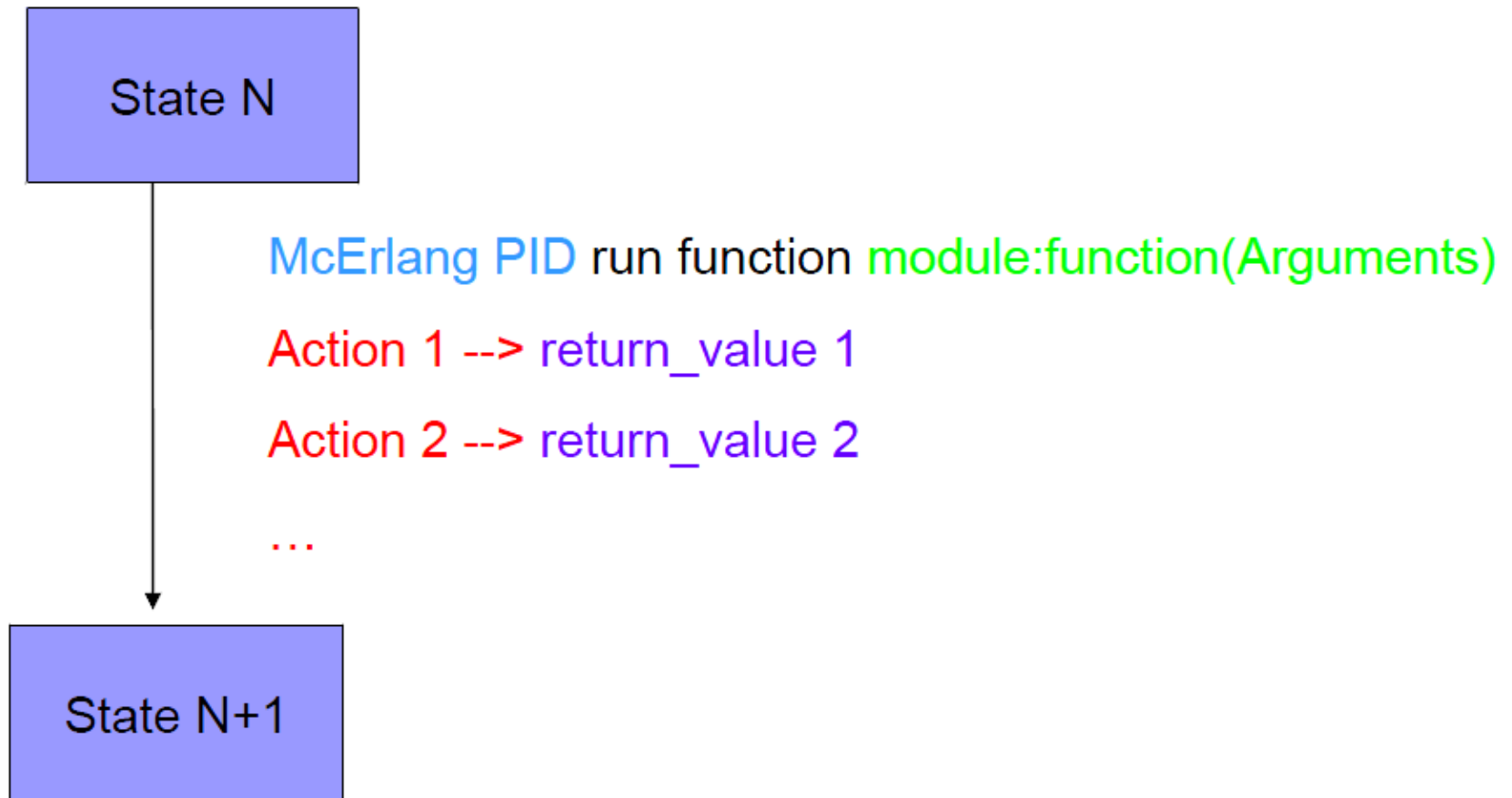
- Trace to counter example:

```
0: process <'node0@avalor-laptop',1>:  
  run function test:test0([2])  
  process_flag(trap_exit,true) --> false  
  spawn( {contractor,init,[2]},[]) --> {pid,'node0@avalor-laptop',2}  
  spawn( {contractor,init,[1]},[]) --> {pid,'node0@avalor-laptop',3}  
  spawn( {manager,startManager,  
    [[ {task,learn_McErlang,0,90,0,void,0,0,[],announced,[]} ],  
    [ {pid,'node0@avalor-laptop',2}, {pid,'node0@avalor-laptop',3} ] ] },[]) -->  
    {pid,'node0@avalor-laptop',4}  
  link( {pid,'node0@avalor-laptop',4}) --> true
```

What you probably see



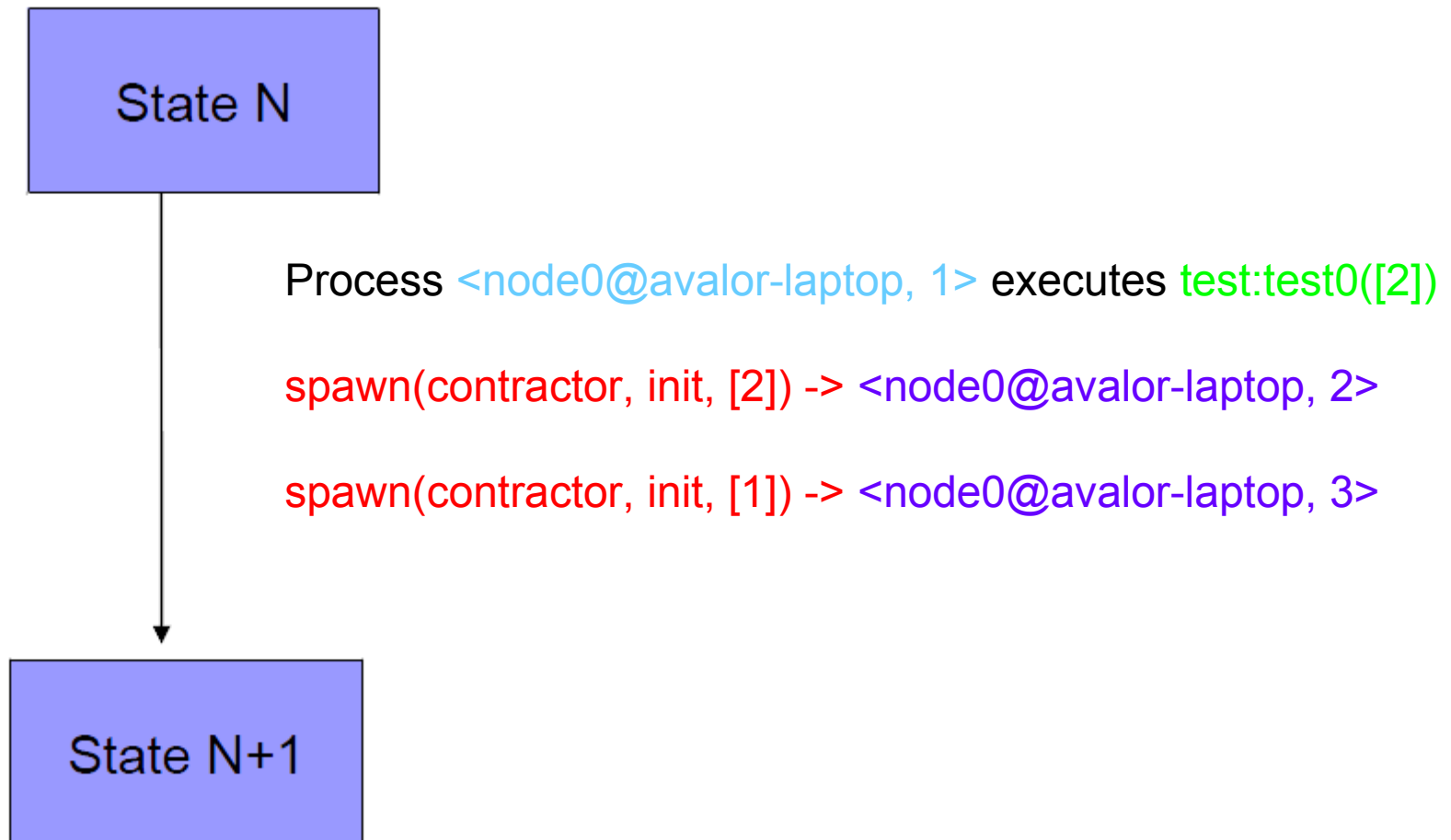
What is actually there



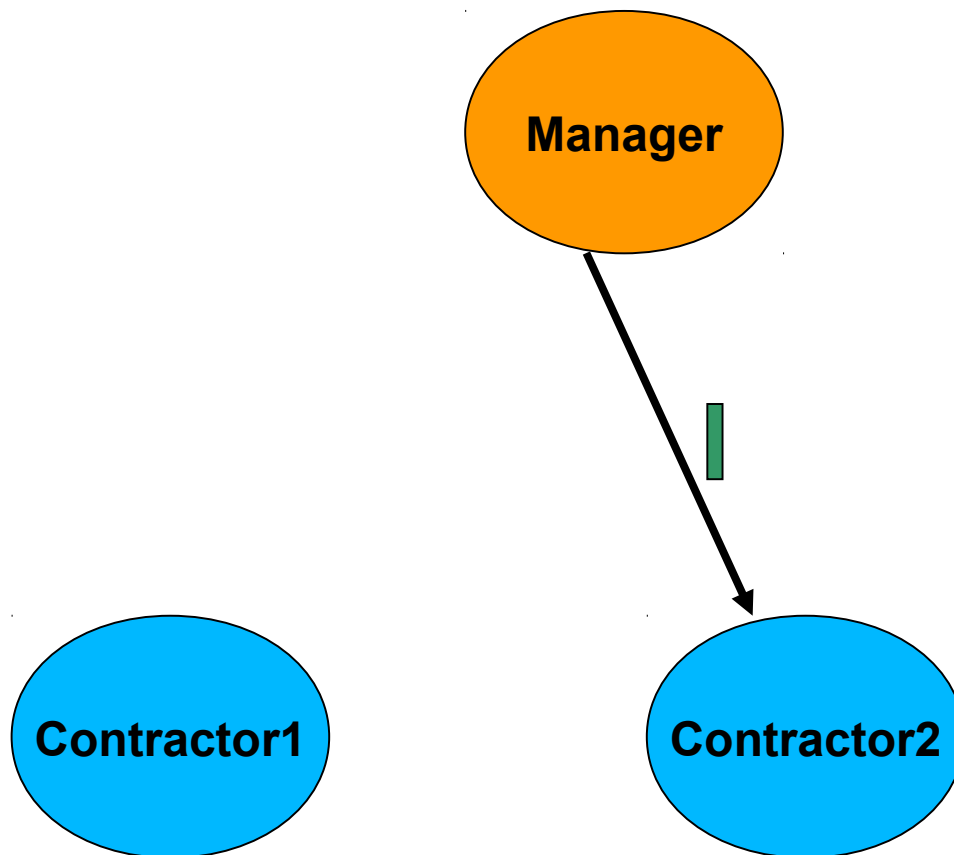
What is actually there

- 0: process <'node0@avalor-laptop',1>:
run function test:test0([2])
process_flag(trap_exit,true) --> false
spawn({contractor,init,[2]},[]) --> {pid,'node0@avalor-laptop',2}
spawn({contractor,init,[1]},[]) --> {pid,'node0@avalor-laptop',3}
spawn({manager,startManager,
[[{task,learn_McErlang,0,90,0,void,0,0,[],announced,[],}],
[{pid,'node0@avalor-laptop',2}, {pid,'node0@avalor-laptop',3}]]},
[]) --> {pid,'node0@avalor-laptop',4}
link({pid,'node0@avalor-laptop',4}) --> true

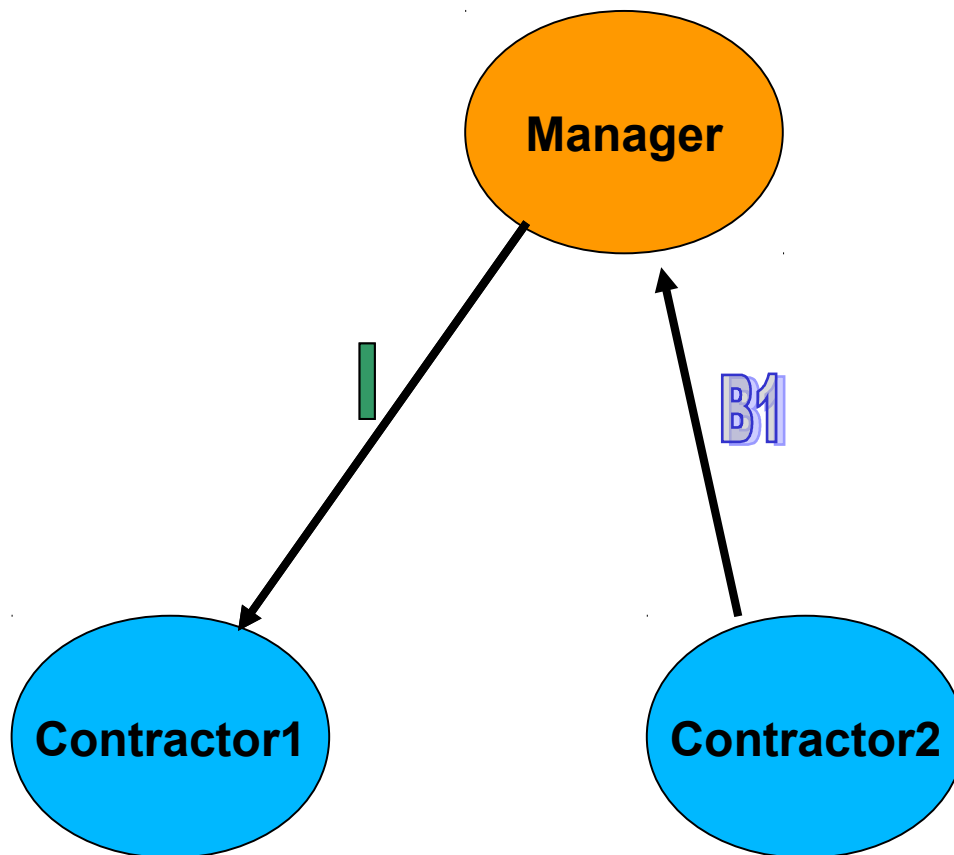
What is actually there



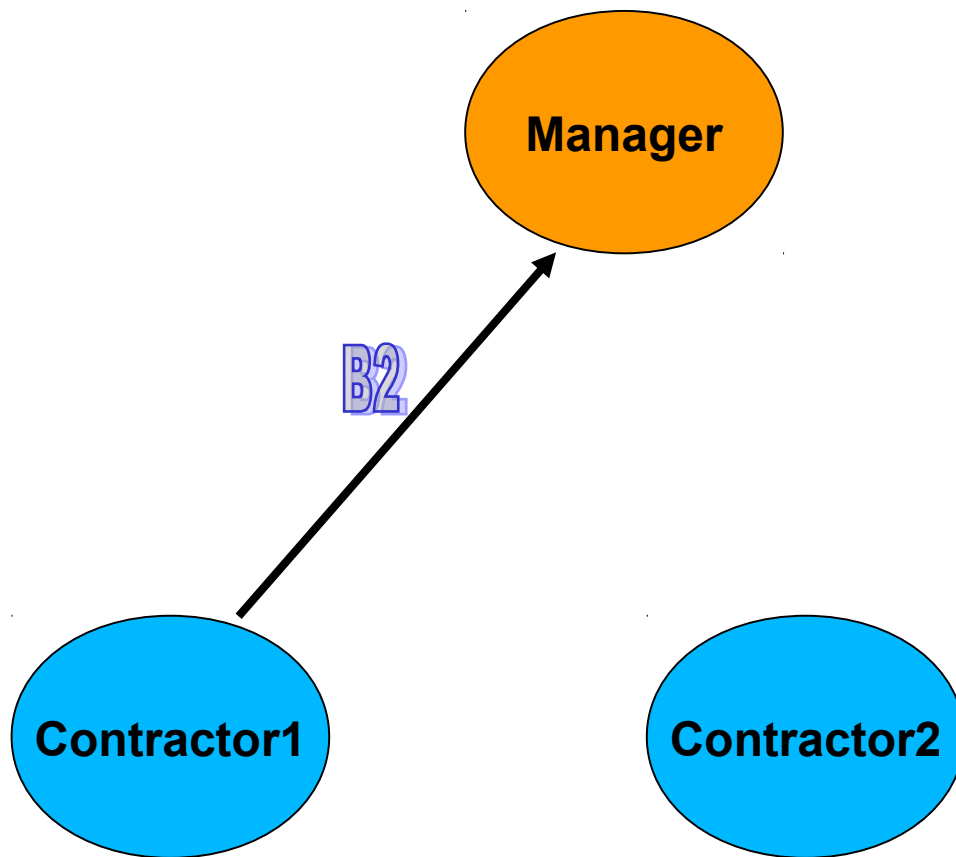
McErlang Trace



McErlang Trace

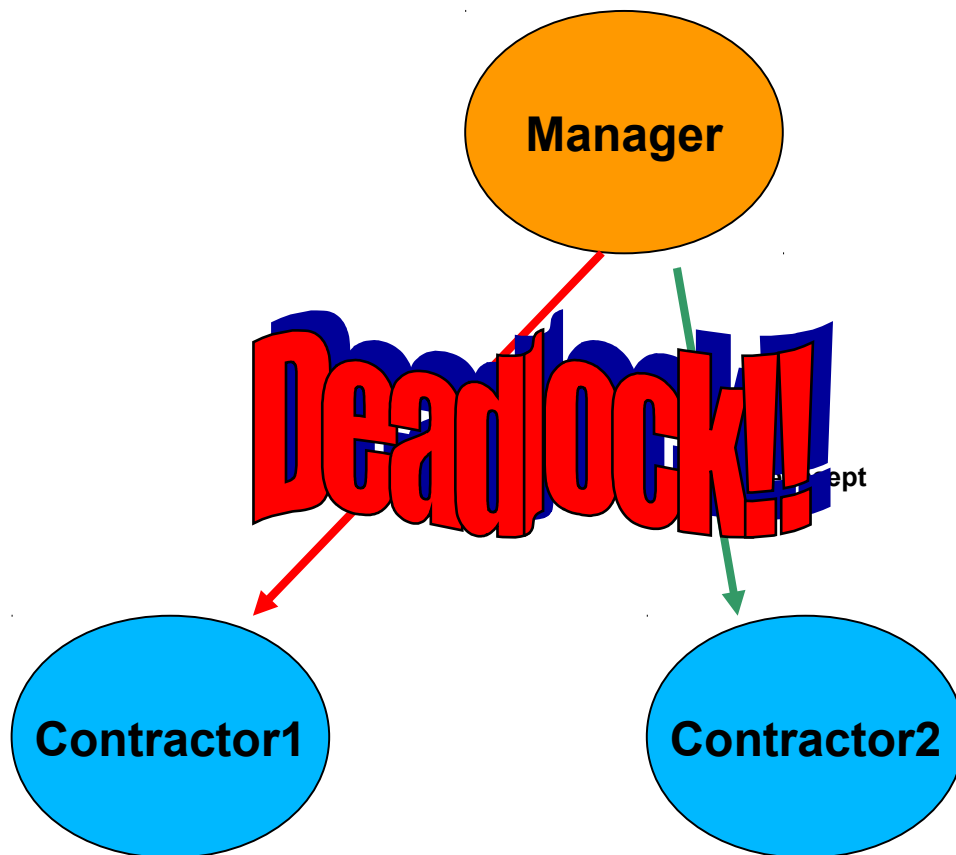


McErlang Trace



B1 > B2

McErlang Trace



B1 > B2

Thanks for your attention!



What is supported?

- Processes, nodes, links, full datatypes supported in Erlang
- Higher-order functions
- Many libraries at least partly supported: supervisor, gen_server, gen_fsm, gen_event, ets, . . .
- No real-time or discrete-time **model checking** implementation yet
 - receive
 - after 20 -> ...
 - end
 - behaves the same as
 - receive
 - after 2000-> ...
 - end.

Some bugs detected by McErlang

- Ad-hoc reimplementations of Erlang/OTP Supervisor Behavior by LambdaStream
- Robocup player implementation
- Specification of CNP extension (a Multi-agent System Negotiation Protocol)