Towards Efficient Verification of Parallel Applications with Mc SimGrid

Joint work with Martin Quinson (Magellan) and Thierry Jéron (Devine)

Mathieu Laurent

October 14, 2024

Distributed computing

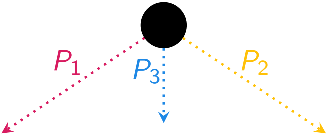
- HPC applications are distributed and concurrent
- Data shared via messages (e.g. MPI) or synchronizations (e.g. thread)
- Causes non-deterministic bugs
- Software model checking covers all cases

Content of this talk

Exploring the transition systems

A small MPI example

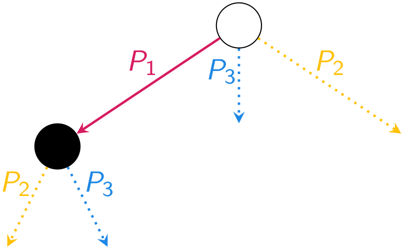
P_1/P_2	P_3
`Send`(P_3)	`Recv()`
	`Recv()`



Exploring the transition systems

A small MPI example

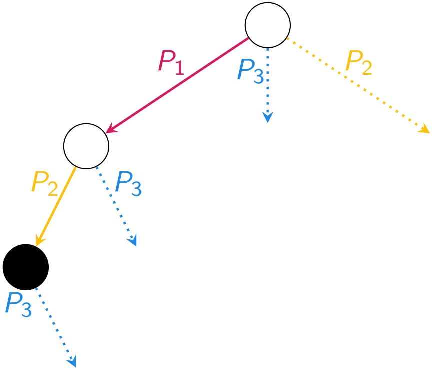
P_1/P_2	P_3
`Send`(P_3)	`Recv()`
	`Recv()`



Exploring the transition systems

A small MPI example

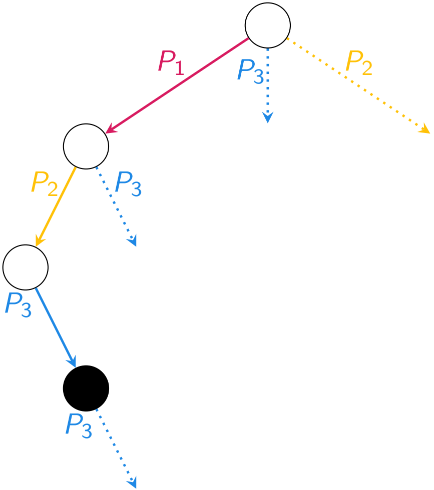
P_1/P_2	P_3
Send(P_3)	Recv()
	Recv()



Exploring the transition systems

A small MPI example

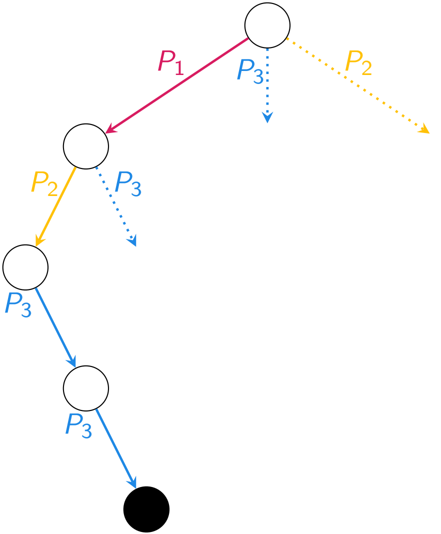
P_1/P_2	P_3
Send(P_3)	Recv()
	Recv()



Exploring the transition systems

A small MPI example

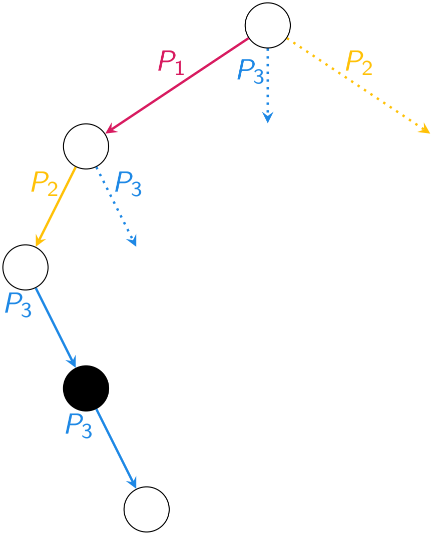
P_1/P_2	P_3
Send(P_3)	Recv()
	Recv()



Exploring the transition systems

A small MPI example

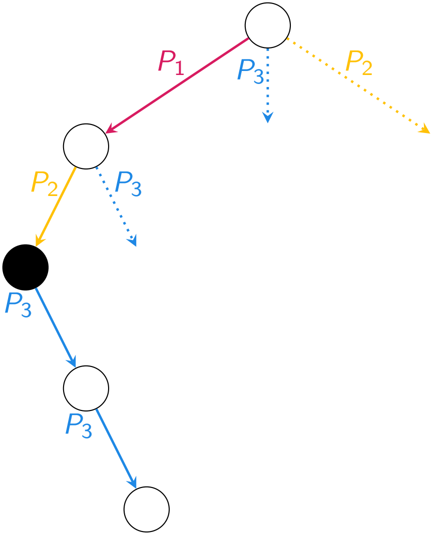
P_1/P_2	P_3
Send(P_3)	Recv()
	Recv()



Exploring the transition systems

A small MPI example

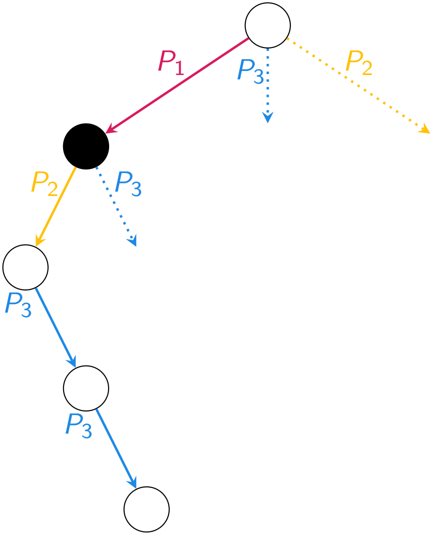
P_1/P_2	P_3
Send(P_3)	Recv()
	Recv()



Exploring the transition systems

A small MPI example

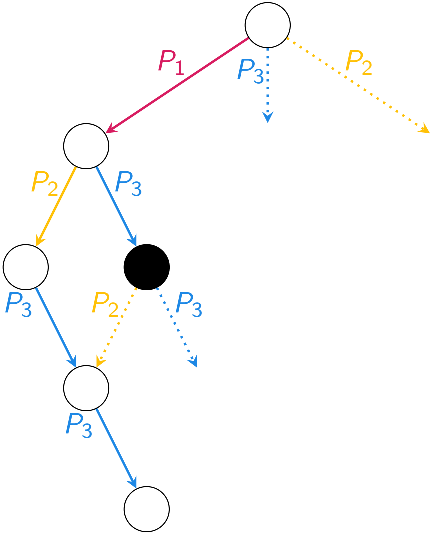
P_1/P_2	P_3
`Send(`P_3`)`	`Recv()`
	`Recv()`



Exploring the transition systems

A small MPI example

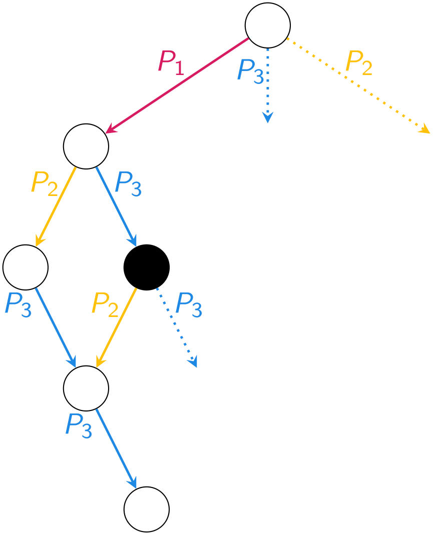
P_1 / P_2	P_3
Send(P_3)	Recv()
	Recv()



Exploring the transition systems

A small MPI example

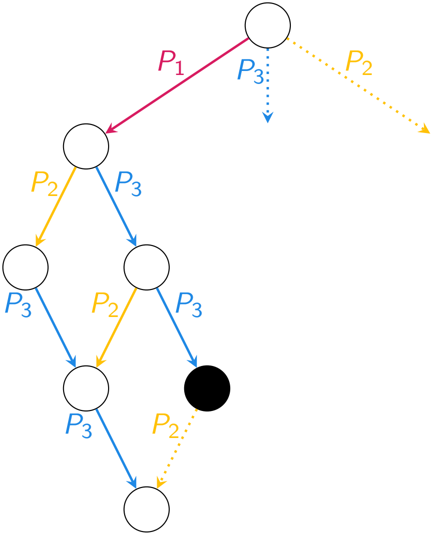
P_1/P_2	P_3
Send(P_3)	Recv()
	Recv()



Exploring the transition systems

A small MPI example

P_1 / P_2	P_3
Send(P_3)	Recv()
	Recv()



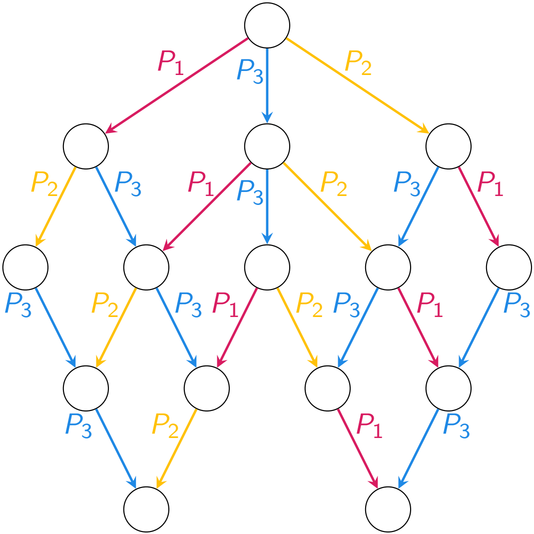
Exploring the transition systems

A small MPI example

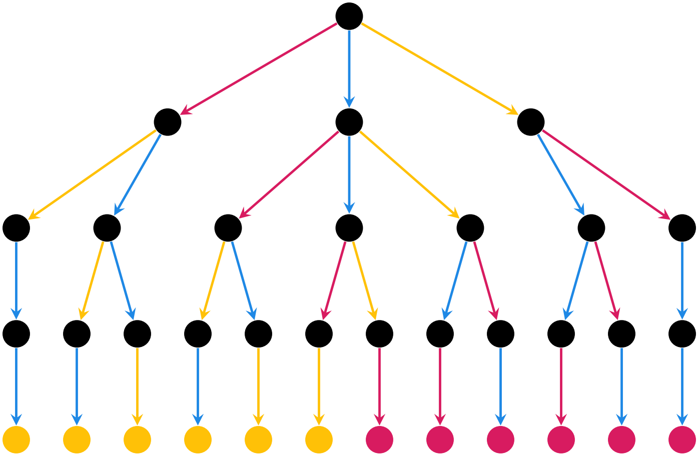
P_1/P_2	P_3
Send(P_3)	Recv()
	Recv()

Stateful exploration

15 states for 2 behaviors.



Stateless model checking

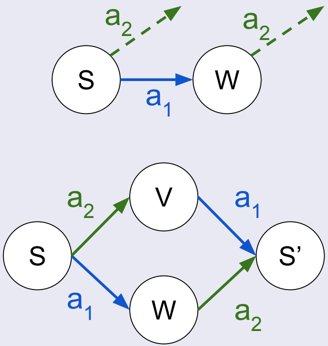


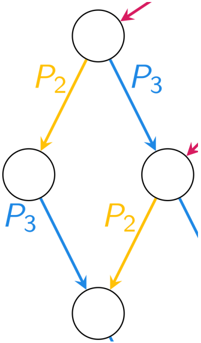
Stateless exploration

35 states for the same 2 behaviors.

Transition dependency

Two actions a_1, a_2 are independent if:



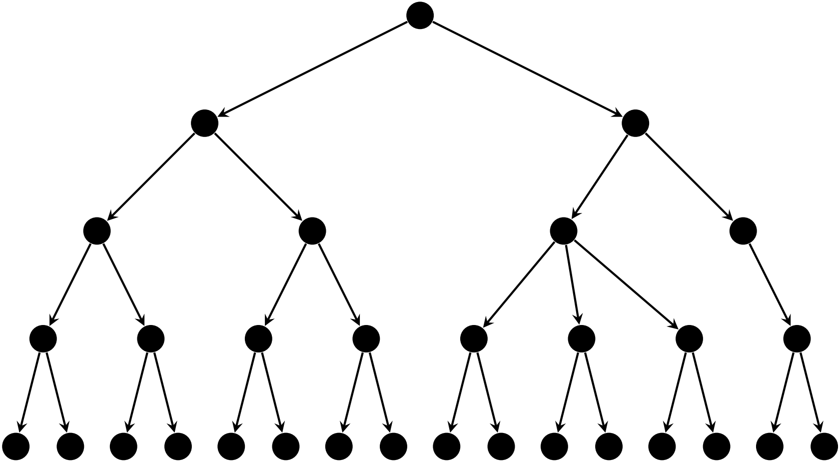


Example of two adjacent independent actions

Mazurkiewicz's traces [Maz'77]

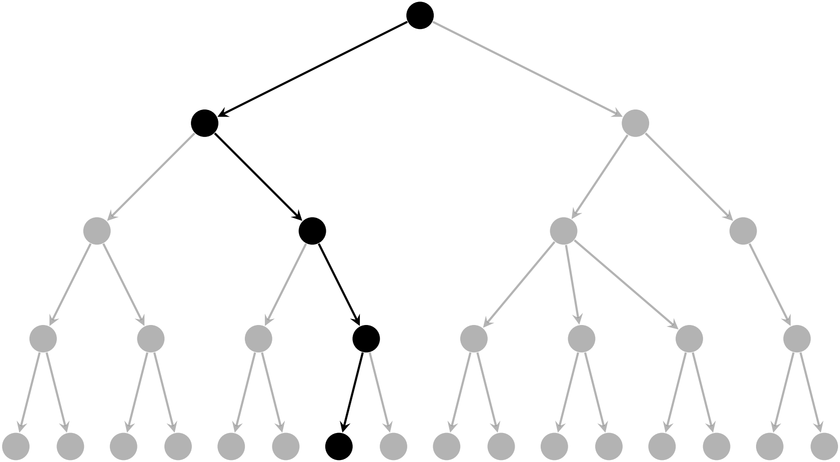
Equivalence class of executions with adjacent independent actions swapped

DPOR approach [Fla'05]



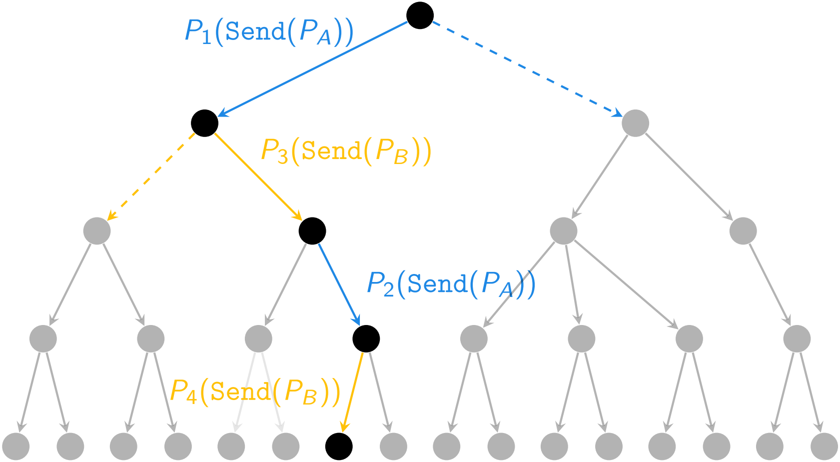
Classical depth first search algorithm

DPOR approach [Fla'05]



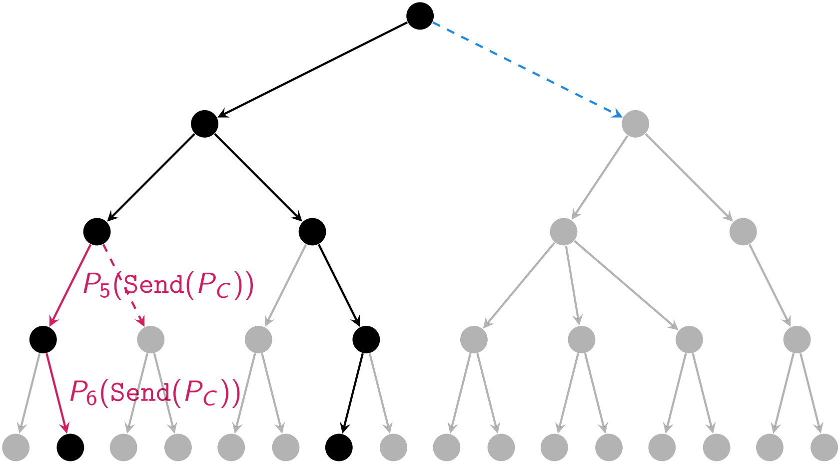
Start with an arbitrary execution

DPOR approach [Fla'05]



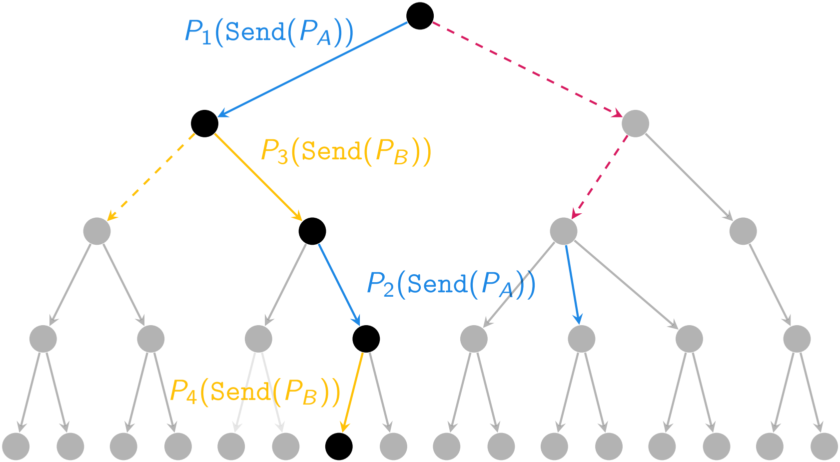
Discover dependencies

DPOR approach [Fla'05]



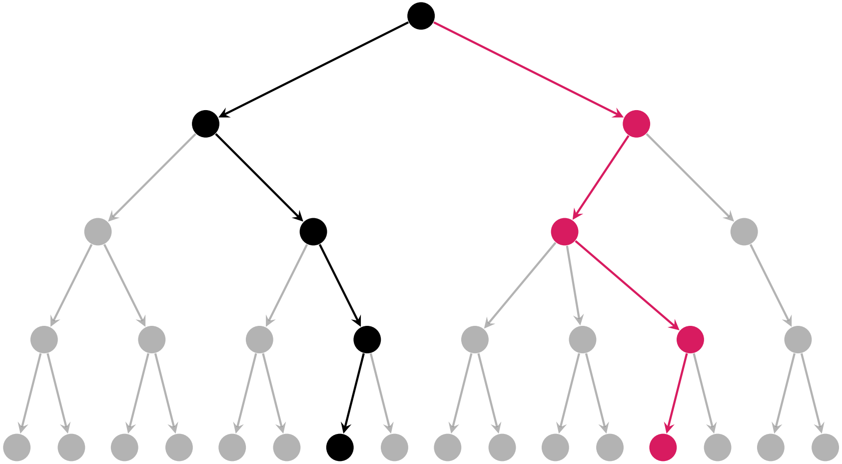
Recursive DFS exploration of what has been added

ODPOR approach [Abd'14]



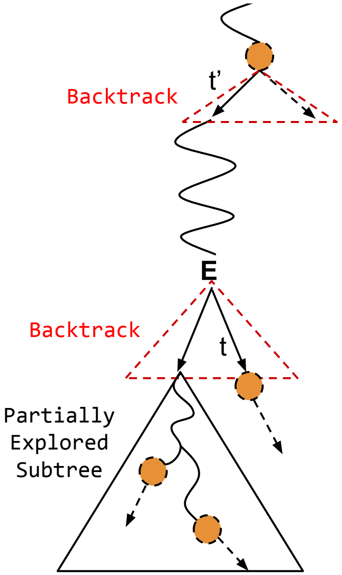
Insert sequences instead of a single step

ODPOR approach [Abd'14]



What if the only bug is far from the first guess?

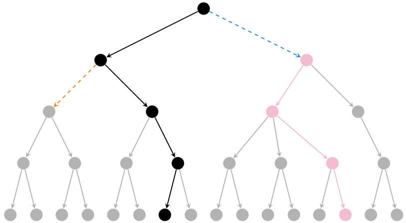
Best First (O)DPOR



- Multiple opened states
- Corresponding to partially explored executions:
- Notion of responsability between subtrees

What for?

- Alleviates the impact of early choices
- Allows the use of heuristic



Counterexample in the other half

```
While(True){
  While(!CAS(x, 0, 1)){
    y = 2;
  }
}
```

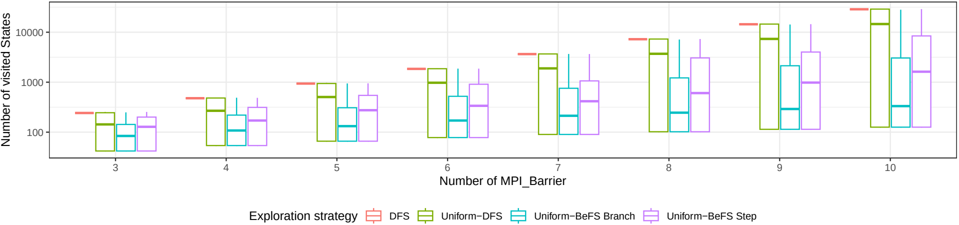
Example of a busy waiting

- Works around practical problems (as busy waiting)
- Encodes classical model checking behavior (like fairness)

Experimental results

MPI example slightly modified

P_1	P_2	P_3
Send(P_3)	Send(P_3)	MPI_Barrier()
MPI_Barrier()	MPI_Barrier()	Recv()
		Recv()

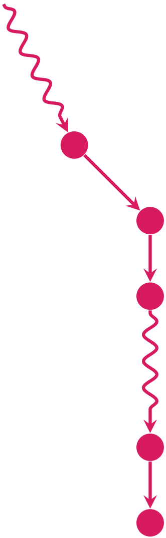


Why?

```
**************************
*** DEADLOCK DETECTED ***
**************************
] 1 actor is still active, awaiting something. Here is its status:
]  - pid 3 (2@node-10.simgrid.org) simcall CommWait(comm_id:20 src:-1 dst:3 mbox:SMPI-3(id:3))
Counter-example execution trace:
  Actor 2 in :0:() ==> simcall: iSend(mbox=3)
  Actor 1 in :0:() ==> simcall: iSend(mbox=3)
  Actor 1 in :0:() ==> simcall: iRecv(mbox=4)
  Actor 1 in :0:() ==> simcall: iRecv(mbox=4)
  Actor 2 in :0:() ==> simcall: iSend(mbox=4)
  Actor 1 in :0:() ==> simcall: WaitComm(from 2 to 1, mbox=4, no timeout)
  Actor 2 in :0:() ==> simcall: iRecv(mbox=5)
  Actor 3 in :0:() ==> simcall: iSend(mbox=4)
  Actor 1 in :0:() ==> simcall: WaitComm(from 3 to 1, mbox=4, no timeout)
  Actor 1 in :0:() ==> simcall: iSend(mbox=5)
  Actor 1 in :0:() ==> simcall: iSend(mbox=3)
  Actor 1 in :0:() ==> simcall: iRecv(mbox=4)
  Actor 1 in :0:() ==> simcall: iRecv(mbox=4)
  Actor 2 in :0:() ==> simcall: WaitComm(from 1 to 2, mbox=5, no timeout)
  Actor 2 in :0:() ==> simcall: iSend(mbox=4)
  Actor 1 in :0:() ==> simcall: WaitComm(from 2 to 1, mbox=4, no timeout)
  Actor 2 in :0:() ==> simcall: iRecv(mbox=5)
  Actor 3 in :0:() ==> simcall: iRecv(mbox=3)
  Actor 3 in :0:() ==> simcall: WaitComm(from 1 to 3, mbox=3, no timeout)
  Actor 3 in :0:() ==> simcall: iSend(mbox=4)
  Actor 1 in :0:() ==> simcall: WaitComm(from 3 to 1, mbox=4, no timeout)
  Actor 1 in :0:() ==> simcall: iSend(mbox=5)
  Actor 1 in :0:() ==> simcall: iSend(mbox=3)
  Actor 2 in :0:() ==> simcall: WaitComm(from 1 to 2, mbox=5, no timeout)
  Actor 3 in :0:() ==> simcall: iRecv(mbox=3)
  Actor 3 in :0:() ==> simcall: WaitComm(from 1 to 3, mbox=3, no timeout)
  Actor 3 in :0:() ==> simcall: iRecv(mbox=3)
  Actor 3 in :0:() ==> simcall: WaitComm(from 2 to 3, mbox=3, no timeout)
  Actor 3 in :0:() ==> simcall: iRecv(mbox=3)
```

Mc SimGrid output on a simple example with only two `MPI_Barrier()`.

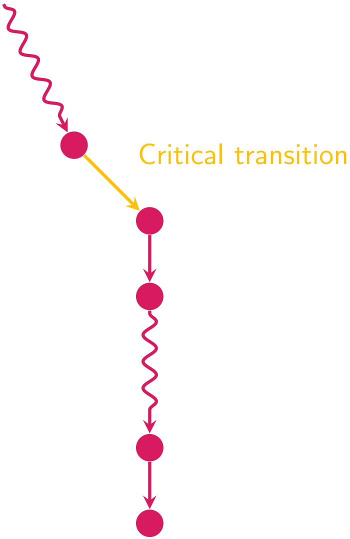
The critical transition



Critical transition

Let E be an incorrect execution,

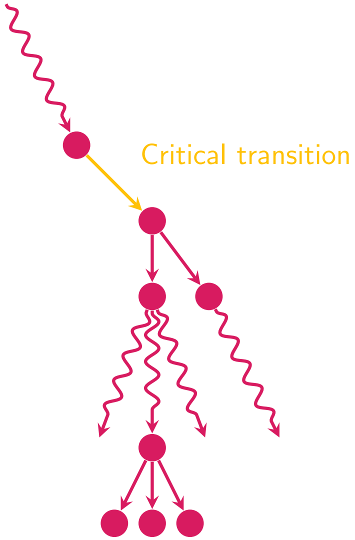
The critical transition



Critical transition

Let E be an incorrect execution,
the **critical transition** is the unique
$t = (s, a, s')\ \in E$ s.t.

The critical transition

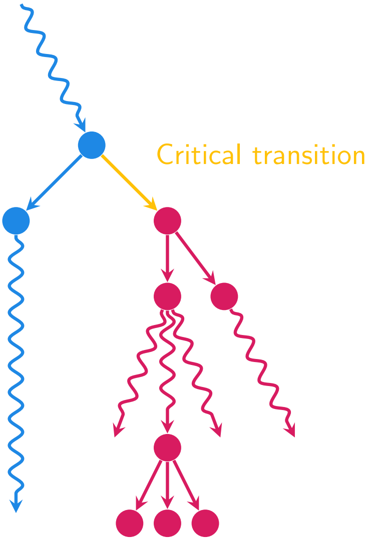


Critical transition

Let E be an incorrect execution,
the **critical transition** is the unique
$t = (s, a, s')$ $\in E$ s.t.

- every execution from s' is incorrect

The critical transition



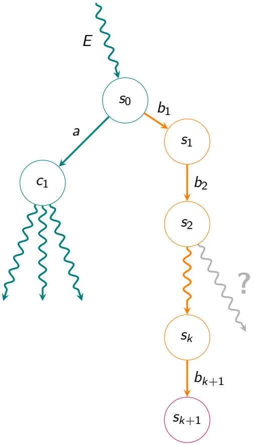
Critical transition

Let E be an incorrect execution,
the **critical transition** is the unique
$t = (s, a, s')\ \in E$ s.t.

- every execution from s' is incorrect
- there exists a correct execution from s

Critical transition: how?

- s_{k+1} violates the property
- c_1 is the root of a correct subtree
- Hence, the critical transition is in $\{b_1, \ldots, b_{k+1}\}$
- Use reduction and take a decision for the non-explored transitions



Conclusion

What we have done

- New reduction algorithms allowing arbitrary search
- Defining and computing critical transition
- Implementing our reasearch in McSimGrid

Future work

- Parallelize the implementation by BeFS ODPOR
- Develop a good benchmark to explore heuristics
- Simplify counter examples using critical section