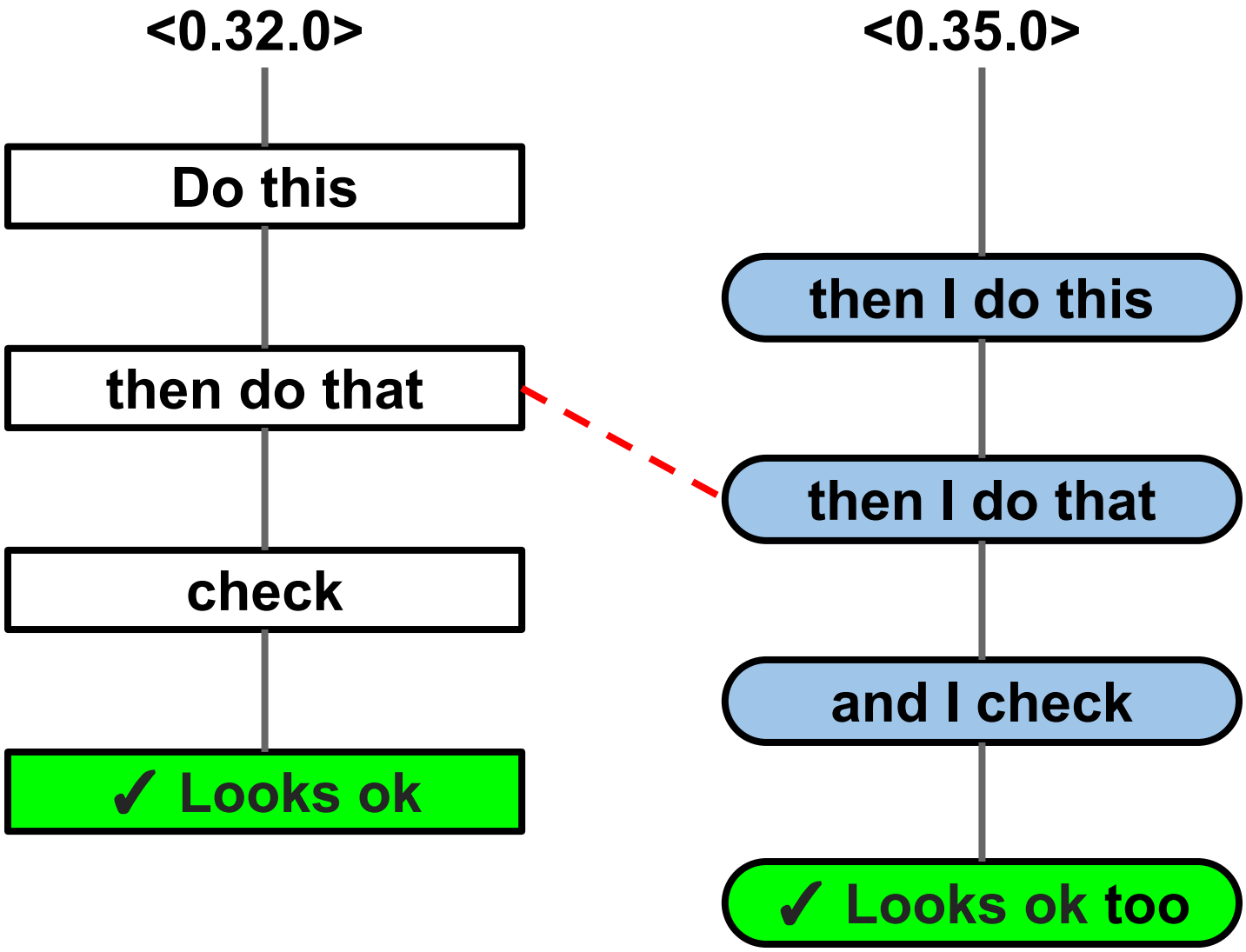
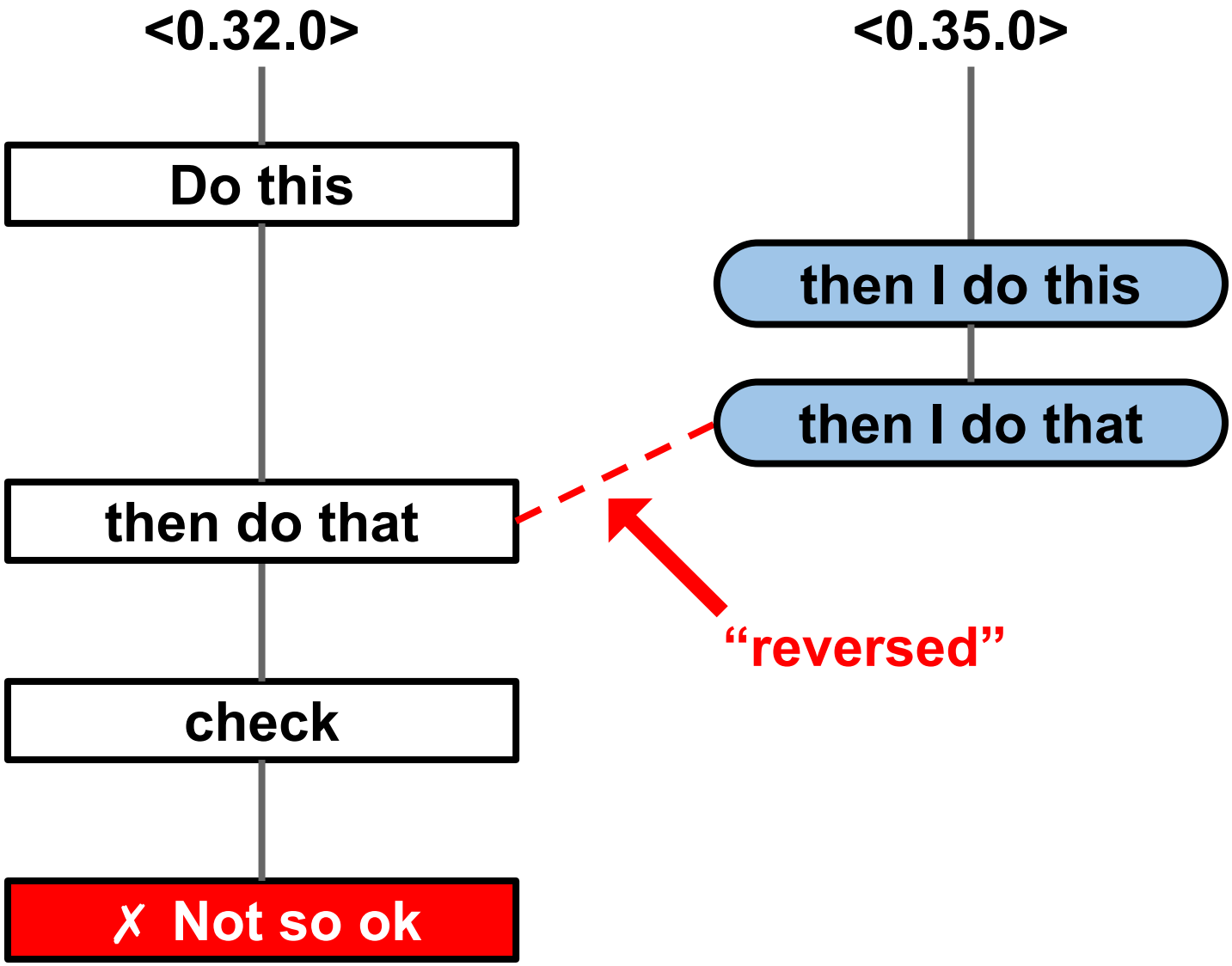
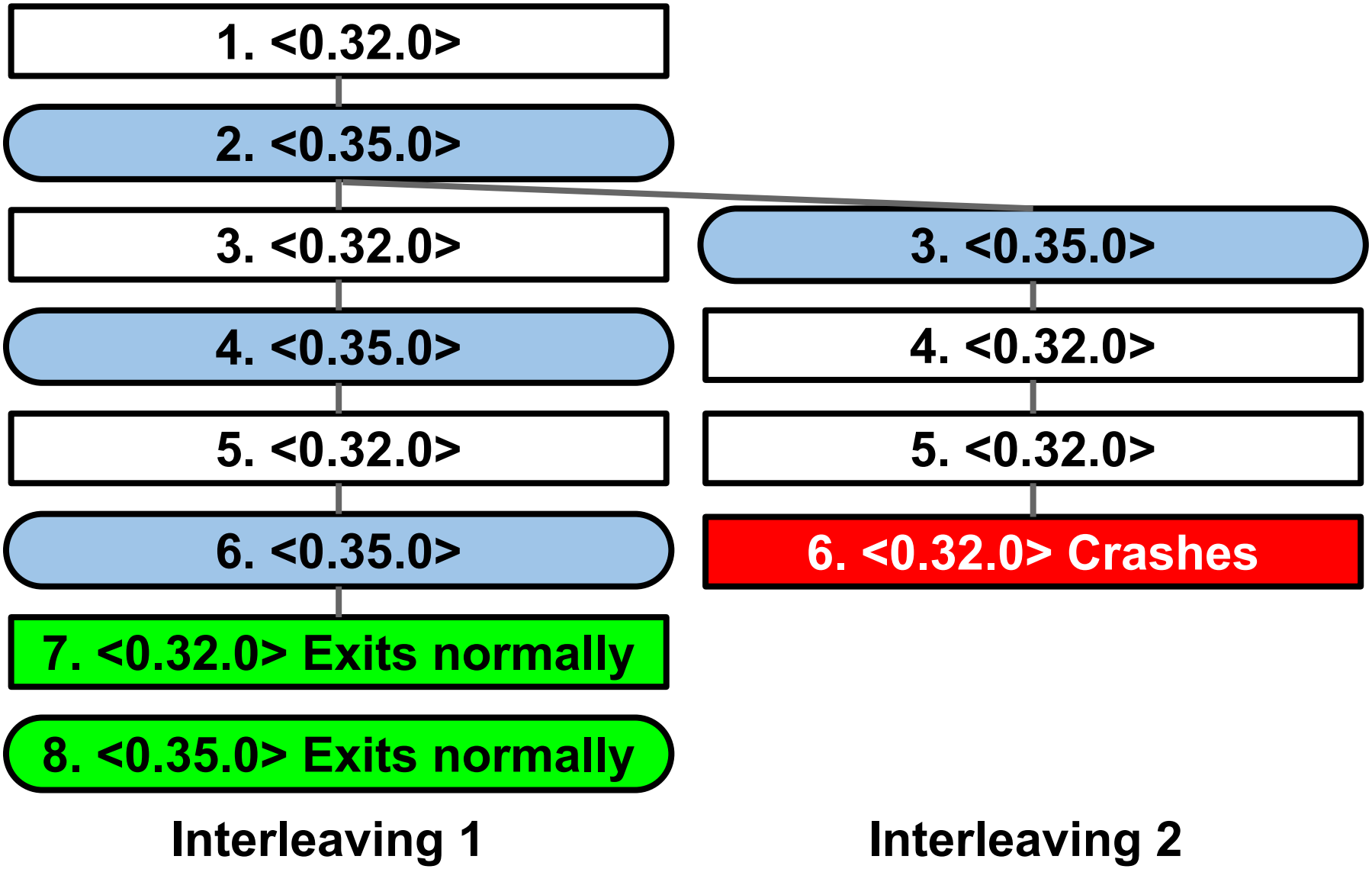
Testing...

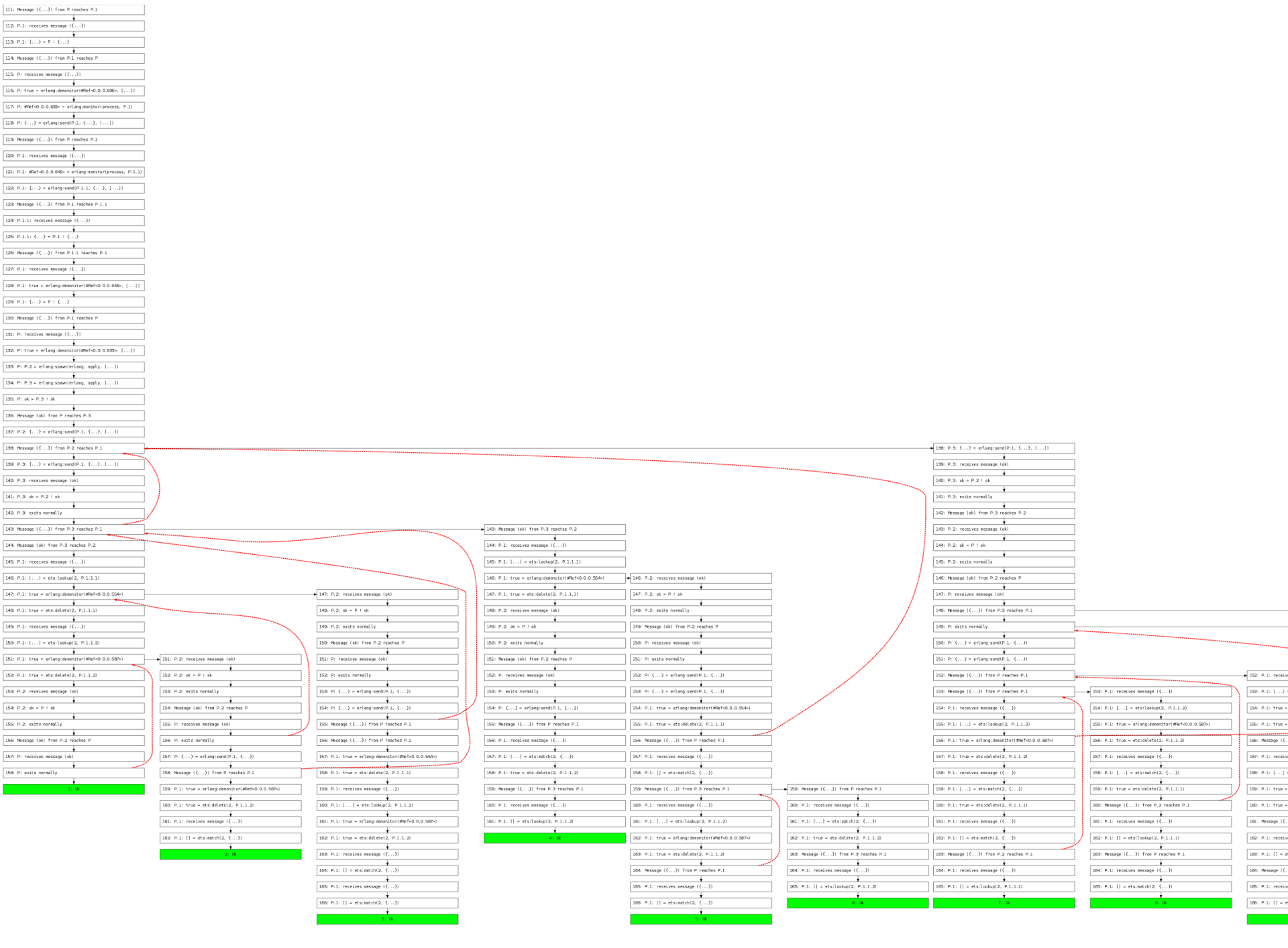

<0.32.0>
Do this
then do that
check
✓ Looks ok
<0.35.0>
then I do this
then I do that
and I check
✓ Looks ok too

Testing...


<0.32.0>
<0.35.0>
Do this
then I do this
then I do that
then do that
"reversed"
check
✗ Not so ok

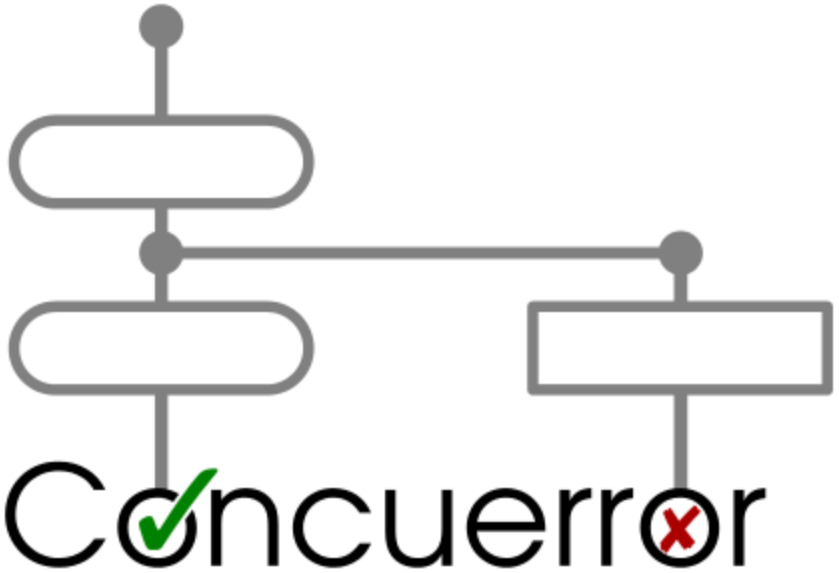
Testing...


1. <0.32.0>
2. <0.35.0>
3. <0.32.0>
4. <0.35.0>
5. <0.32.0>
6. <0.35.0>
7. <0.32.0> Exits normally
8. <0.35.0> Exits normally
3. <0.35.0>
4. <0.32.0>
5. <0.32.0>
6. <0.32.0> Crashes
Interleaving 1
Interleaving 2


111: Message ({...}) from P reaches P.1
112: P.1: receives message ({...})
113: P.1: {...} = P ! {...}
114: Message ({...}) from P.1 reaches P
115: P: receives message ({...})
116: P: true = erlang:demonitor(#Ref<0.0.0.606>, [...])
117: P: #Ref<0.0.0.633> = erlang:monitor(process, P.1)
118: P: {...} = erlang:send(P.1, {...}, [...])
119: Message ({...}) from P reaches P.1
120: P.1: receives message ({...})
121: P.1: #Ref<0.0.0.646> = erlang:monitor(process, P.1.1)
122: P.1: {...} = erlang:send(P.1.1, {...}, [...])
123: Message ({...}) from P.1 reaches P.1.1
124: P.1.1: receives message ({...})
125: P.1.1: {...} = P.1 ! {...}
126: Message ({...}) from P.1.1 reaches P.1
127: P.1: receives message ({...})
128: P.1: true = erlang:demonitor(#Ref<0.0.0.646>, [...])
129: P.1: {...} = P ! {...}
130: Message ({...}) from P.1 reaches P
131: P: receives message ({...})
132: P: true = erlang:demonitor(#Ref<0.0.0.633>, [...])
133: P: P.2 = erlang:spawn(erlang, apply, [...])
134: P: P.3 = erlang:spawn(erlang, apply, [...])
135: P: ok = P.3 ! ok
136: Message (ok) from P reaches P.3
137: P.2: {...} = erlang:send(P.1, {...}, [...])
138: Message ({...}) from P.2 reaches P.1
139: P.3: {...} = erlang:send(P.1, {...}, [...])
140: P.3: receives message (ok)
141: P.3: ok = P.2 ! ok
142: P.3: exits normally
143: Message ({...}) from P.3 reaches P.1
144: Message (ok) from P.3 reaches P.2
145: P.1: receives message ({...})
146: P.1: [...] = ets:lookup(2, P.1.1.1)
147: P.1: true = erlang:demonitor(#Ref<0.0.0.554>)
148: P.1: true = ets:delete(2, P.1.1.1)
149: P.1: receives message ({...})
150: P.1: [...] = ets:lookup(2, P.1.1.2)
151: P.1: true = erlang:demonitor(#Ref<0.0.0.587>)
152: P.1: true = ets:delete(2, P.1.1.2)
153: P.2: receives message (ok)
154: P.2: ok = P ! ok
155: P.2: exits normally
156: Message (ok) from P.2 reaches P
157: P.1: receives message (ok)
158: P: exits normally
1: Ok
151: P.2: receives message (ok)
152: P.2: ok = P ! ok
153: P.2: exits normally
154: Message (ok) from P.2 reaches P
155: P: receives message (ok)
156: P: exits normally
157: P: {...} = erlang:send(P.1, {...})
158: Message ({...}) from P reaches P.1
159: P.1: true = erlang:demonitor(#Ref<0.0.0.587>)
160: P.1: true = ets:delete(2, P.1.1.2)
161: P.1: receives message ({...})
162: P.1: [] = ets:match(2, {...})
2: Ok
147: P.2: receives message (ok)
148: P.2: ok = P ! ok
149: P.2: exits normally
150: Message (ok) from P.2 reaches P
151: P: receives message (ok)
152: P: exits normally
153: P: {...} = erlang:send(P.1, {...})
154: P: {...} = erlang:send(P.1, {...})
155: Message ({...}) from P reaches P.1
156: Message ({...}) from P reaches P.1
157: P.1: true = erlang:demonitor(#Ref<0.0.0.554>)
158: P.1: true = ets:delete(2, P.1.1.1)
159: P.1: receives message ({...})
160: P.1: [...] = ets:lookup(2, P.1.1.2)
161: P.1: true = erlang:demonitor(#Ref<0.0.0.587>)
162: P.1: true = ets:delete(2, P.1.1.2)
163: P.1: receives message ({...})
164: P.1: [] = ets:match(2, {...})
165: P.1: receives message ({...})
166: P.1: [] = ets:match(2, {...})
3: Ok
143: Message (ok) from P.3 reaches P.2
144: P.1: receives message ({...})
145: P.1: [...] = ets:lookup(2, P.1.1.1)
146: P.1: true = erlang:demonitor(#Ref<0.0.0.554>)
147: P.1: true = ets:delete(2, P.1.1.1)
148: P.2: receives message (ok)
149: P.2: ok = P ! ok
150: P.2: exits normally
151: Message (ok) from P.2 reaches P
152: P: receives message (ok)
153: P: exits normally
154: P: {...} = erlang:send(P.1, {...})
155: Message ({...}) from P reaches P.1
156: P.1: receives message ({...})
157: P.1: [...] = ets:match(2, {...})
158: P.1: true = ets:delete(2, P.1.1.2)
159: Message ({...}) from P.3 reaches P.1
160: P.1: receives message ({...})
161: P.1: [] = ets:lookup(2, P.1.1.2)
4: Ok
146: P.2: receives message (ok)
147: P.2: ok = P ! ok
148: P.2: exits normally
149: Message (ok) from P.2 reaches P
150: P: receives message (ok)
151: P: exits normally
152: P: {...} = erlang:send(P.1, {...})
153: P: {...} = erlang:send(P.1, {...})
154: P.1: true = erlang:demonitor(#Ref<0.0.0.554>)
155: P.1: true = ets:delete(2, P.1.1.1)
156: Message ({...}) from P reaches P.1
157: P.1: receives message ({...})
158: P.1: [] = ets:match(2, {...})
159: Message ({...}) from P.3 reaches P.1
160: P.1: receives message ({...})
161: P.1: [...] = ets:lookup(2, P.1.1.2)
162: P.1: true = erlang:demonitor(#Ref<0.0.0.587>)
163: P.1: true = ets:delete(2, P.1.1.2)
164: Message ({...}) from P reaches P.1
165: P.1: receives message ({...})
166: P.1: [] = ets:match(2, {...})
5: Ok
159: Message ({...}) from P reaches P.1
160: P.1: receives message ({...})
161: P.1: [...] = ets:match(2, {...})
162: P.1: true = ets:delete(2, P.1.1.2)
163: Message ({...}) from P.3 reaches P.1
164: P.1: receives message ({...})
165: P.1: [] = ets:lookup(2, P.1.1.2)
6: Ok
138: P.3: {...} = erlang:send(P.1, {...}, [...])
139: P.3: receives message (ok)
140: P.3: ok = P.2 ! ok
141: P.3: exits normally
142: Message (ok) from P.3 reaches P.2
143: P.2: receives message (ok)
144: P.2: ok = P ! ok
145: P.2: exits normally
146: Message (ok) from P.2 reaches P
147: P: receives message (ok)
148: Message ({...}) from P.3 reaches P.1
149: P: exits normally
150: P: {...} = erlang:send(P.1, {...})
151: P: {...} = erlang:send(P.1, {...})
152: Message ({...}) from P reaches P.1
153: Message ({...}) from P reaches P.1
154: P.1: receives message ({...})
155: P.1: [...] = ets:lookup(2, P.1.1.2)
156: P.1: true = erlang:demonitor(#Ref<0.0.0.587>)
157: P.1: true = ets:delete(2, P.1.1.2)
158: P.1: receives message ({...})
159: P.1: [...] = ets:match(2, {...})
160: P.1: true = ets:delete(2, P.1.1.1)
161: P.1: receives message ({...})
162: P.1: [] = ets:match(2, {...})
163: Message ({...}) from P.2 reaches P.1
164: P.1: receives message ({...})
165: P.1: [] = ets:lookup(2, P.1.1.1)
7: Ok
152: P.1: receive
153: P.1: [...] =
154: P.1: true =
155: P.1: true =
156: Message ({...
157: P.1: receive
158: P.1: [...] =
159: P.1: true =
160: Message ({...
161: P.1: receive
162: P.1: receive
163: P.1: [] = et
164: Message ({...
165: P.1: receive
166: P.1: [] = et
8: Ok


Initial
1: P: true = erlang:register(bank, P)
2: P: P.1 = erlang:spawn(erlang, apply, [...])
3: P: P.2 = erlang:spawn(erlang, apply, [...])
4: P: P.3 = erlang:spawn(erlang, apply, [...])
5: P.1: money = bank ! money
6: P.1: exits normally
7: Message (money) from P.1 reaches P
8: P.3: true = erlang:unregister(bank)
9: P.3: true = erlang:register(bank, P.3)
10: P.3: receive timeout expired after 0 ms
11: P.3: exits normally
12: P: receives message (money)
13: P: bank_got_money = P.2 ! bank_got_money
14: P: exits normally
15: Message (bank_got_money) from P reaches P.2
16: P.2: receives message (bank_got_money)
17: P.2: exits normally
1: Ok
8: P: receives message (money)
9: P: bank_got_money = P.2 ! bank_got_money
10: Message (bank_got_money) from P reaches P.2
11: P.2: receives message (bank_got_money)
12: P.2: exits normally
13: P: exits normally
14: P.3: Exception badarg raised by: erlang:unregister(bank)
15: P.3: exits abnormally ({...})
2: Error
5: P.3: true = erlang:unregister(bank)
6: P.3: true = erlang:register(bank, P.3)
7: P.3: receive timeout expired after 0 ms
8: P.3: exits normally
9: P.1: Exception badarg raised by: bank ! money
10: P.1: exits abnormally ({...})
3: Error
8: P.1: money = bank ! money
9: P.1: exits normally
10: Message (money) from P.1 reaches P.3
11: P.3: exits normally
4: Error([P,P.2] blocked)
7: P.1: money = bank ! money
8: P.1: exits normally
9: Message (money) from P.1 reaches P.3
10: P.3: receives message (money)
11: P.3: robber_got_money = P.2 ! robber_got_money
12: P.3: exits normally
13: Message (robber_got_money) from P.3 reaches P.2
14: P.2: receives message (robber_got_money)
15: P.2: exits normally
5: Error([P] blocked)
6: P.1: Exception badarg raised by: bank ! money
7: P.1: exits abnormally ({...})
8: P.3: true = erlang:register(bank, P.3)
9: P.3: receive timeout expired after 0 ms
10: P.3: exits normally
6: Error

?



Into Real Code

Stavros Aronis

UPPSALA
UNIVERSITET

RELEASE

Concuerror

- … is a tool for **systematic** testing
- … runs a test under **all** possible interleavings
- … detects abnormal process exits
- … reports all the events that lead to the crash

Efficient, easy to use

Optimal DPOR, automatic instrumentation, and more...

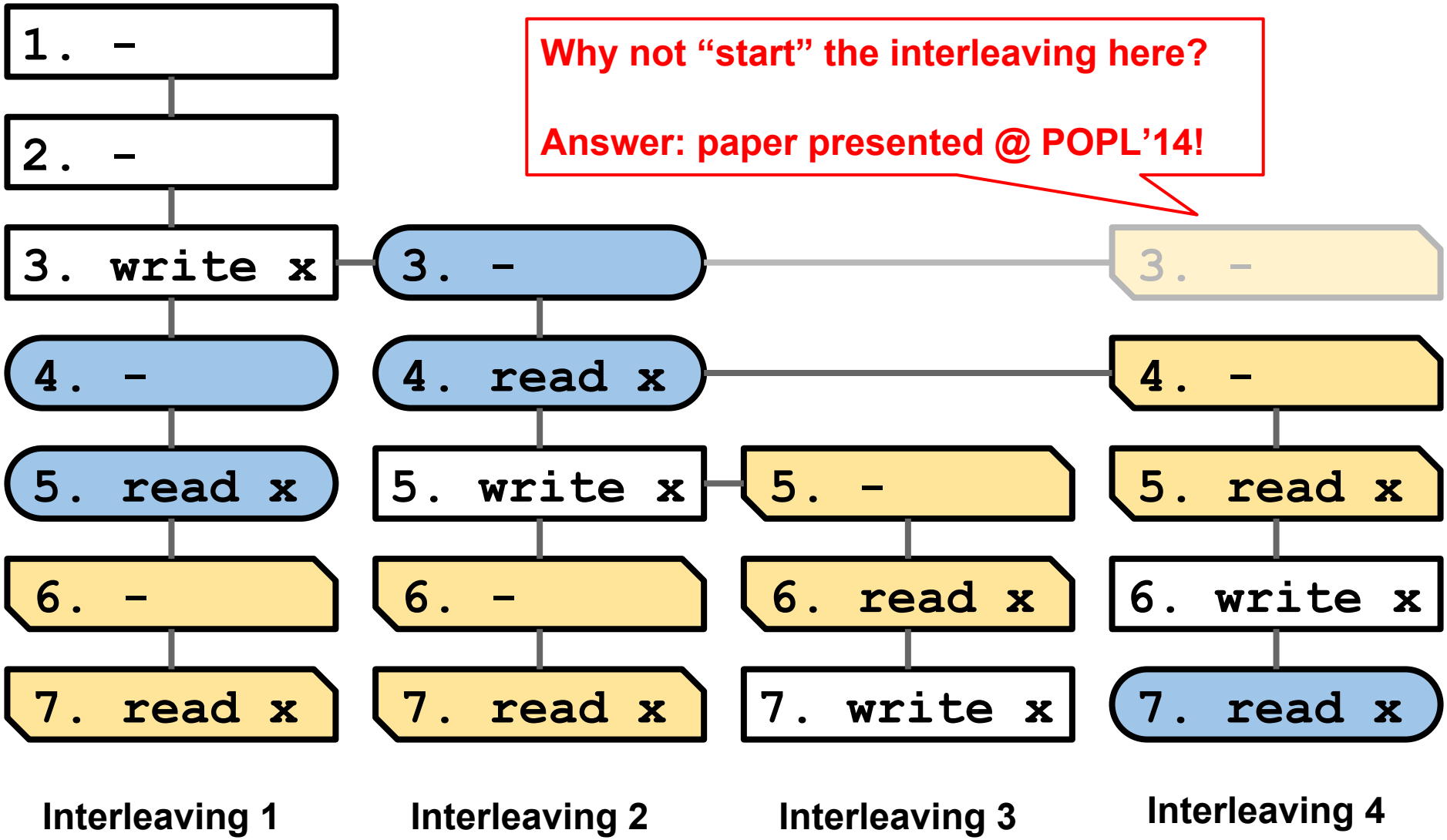
Optimal Dynamic Partial Order Reduction

Systematic =/= Stupid

- Literally “all interleavings”? Too many!
- Not all pairs of **events** are in a race
- Each interleaving should be **different**

Partial Order Reduction techniques

- … monitor **dependencies** between events
- … explore additional interleavings **as needed**
- … avoiding **equivalent** interleavings
- **Dynamic**: at runtime, using concrete data


1. –
2. –
3. write x
4. –
5. read x
6. –
7. read x
3. –
4. read x
5. write x
6. –
7. read x
5. –
6. read x
7. write x
3. –
4. –
5. read x
6. write x
7. read x
Why not “start” the interleaving here?
Answer: paper presented @ POPL’14!
Interleaving 1
Interleaving 2
Interleaving 3
Interleaving 4

Optimal DPOR vs “Classic” DPOR

- Unnecessary interleavings are not even started
- Classic DPOR: orders of magnitude better than exhaustive
- Optimal DPOR: orders of magnitude better than Classic DPOR :-)

POPL'14: Evaluation

Benchmark	Interleavings explored		Time	
	Classic	Optimal	Classic	Optimal
readers (2)	5	4	0.02s	0.02s
readers (8)	3281	256	13.98s	1.29s
readers (13)	797162	8192	**86m7s**	**1m26s**
lastzero (5)	241	64	1.08s	0.32s
lastzero (10)	53198	3328	4m47s	27.61s
lastzero (15)	9378091	**147456**	1539m	**30m13s**

Difference between Classic and Optimal

POPL'14: Evaluation

Benchmark	Interleavings explored		Time	
	Classic	Optimal	Classic	Optimal
dialyzer	12436	3600	14m46s	5m46s
gproc	14080	8104	3m3s	1m57s
poolboy	6018	2680	3m2s	1m20s

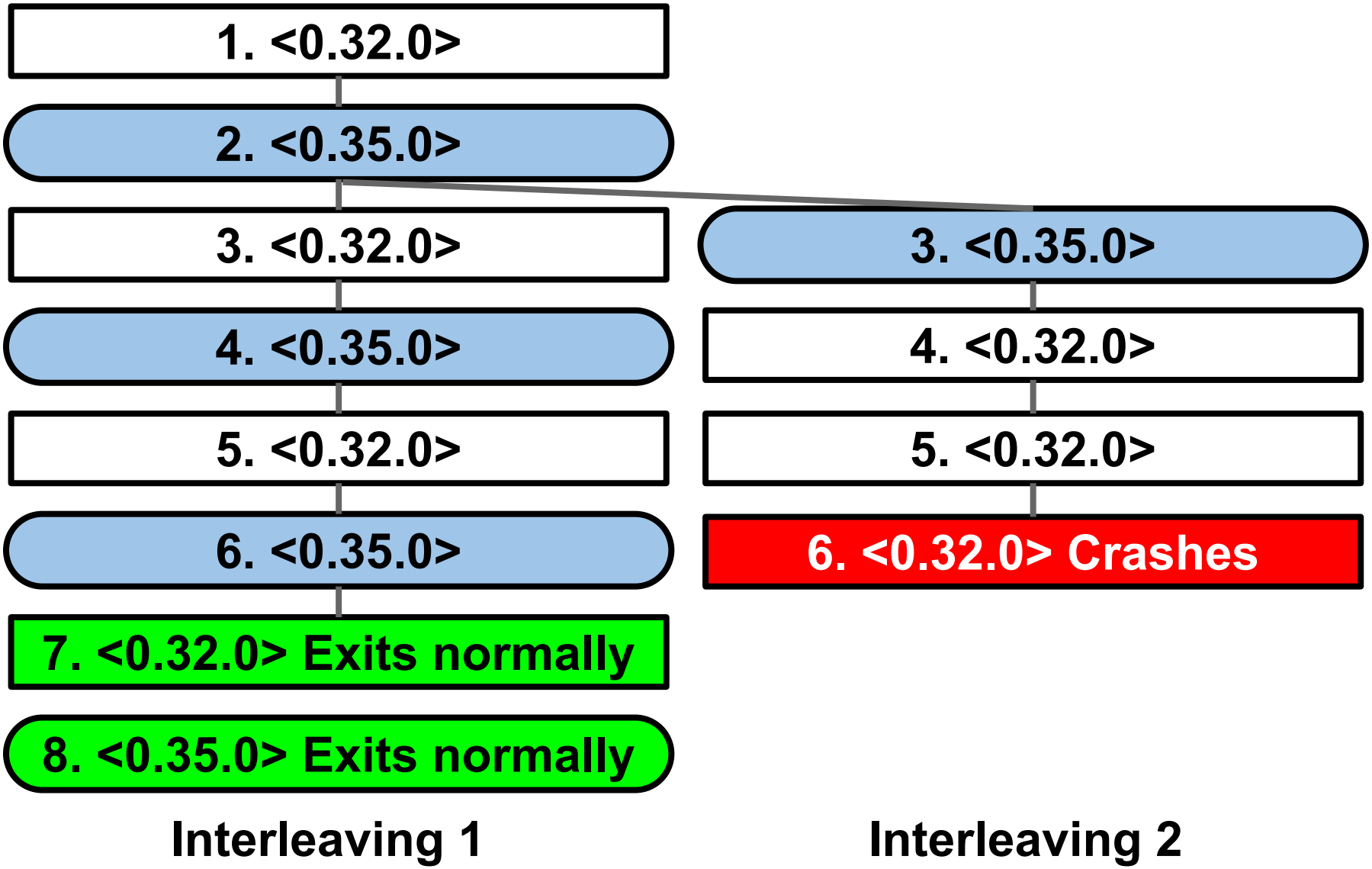
LOC: 44596 (dialyzer), 9446 (gproc), 79732 (poolboy)

Optimal DPOR: Summarry

- Not all pairs of events are racing!
- Concuerror will never even begin to explore equivalent interleavings
- Trace analysis, intelligent algorithms, tailored dependency tracking for Erlang built-ins

Automatic instrumentation

Testing...


1. <0.32.0>
2. <0.35.0>
3. <0.32.0>
4. <0.35.0>
5. <0.32.0>
6. <0.35.0>
7. <0.32.0> Exits normally
8. <0.35.0> Exits normally
3. <0.35.0>
4. <0.32.0>
5. <0.32.0>
6. <0.32.0> Crashes
Interleaving 1
Interleaving 2

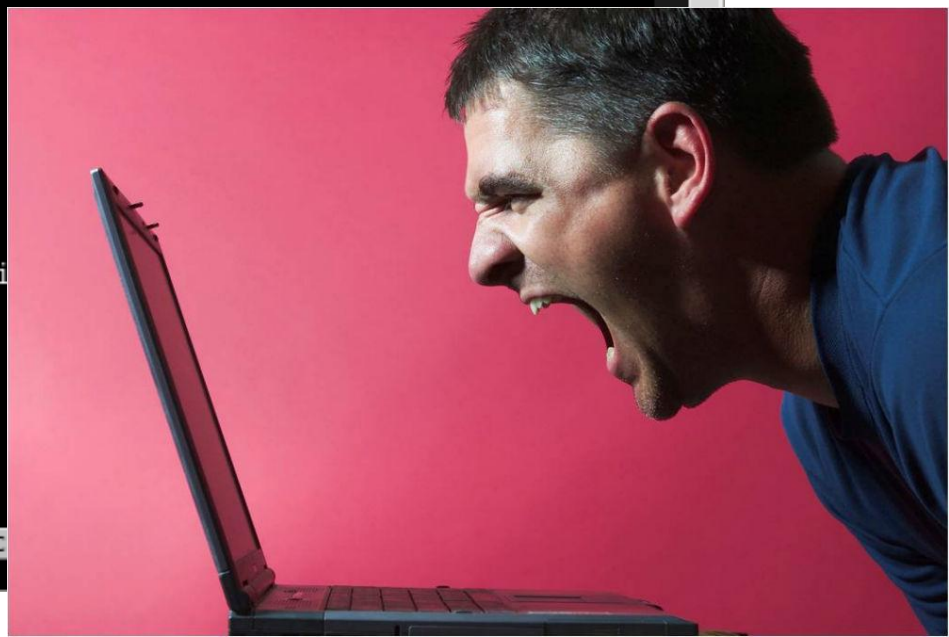
Automatic instrumentation

emacs23@pc-staar721.it.uu.se

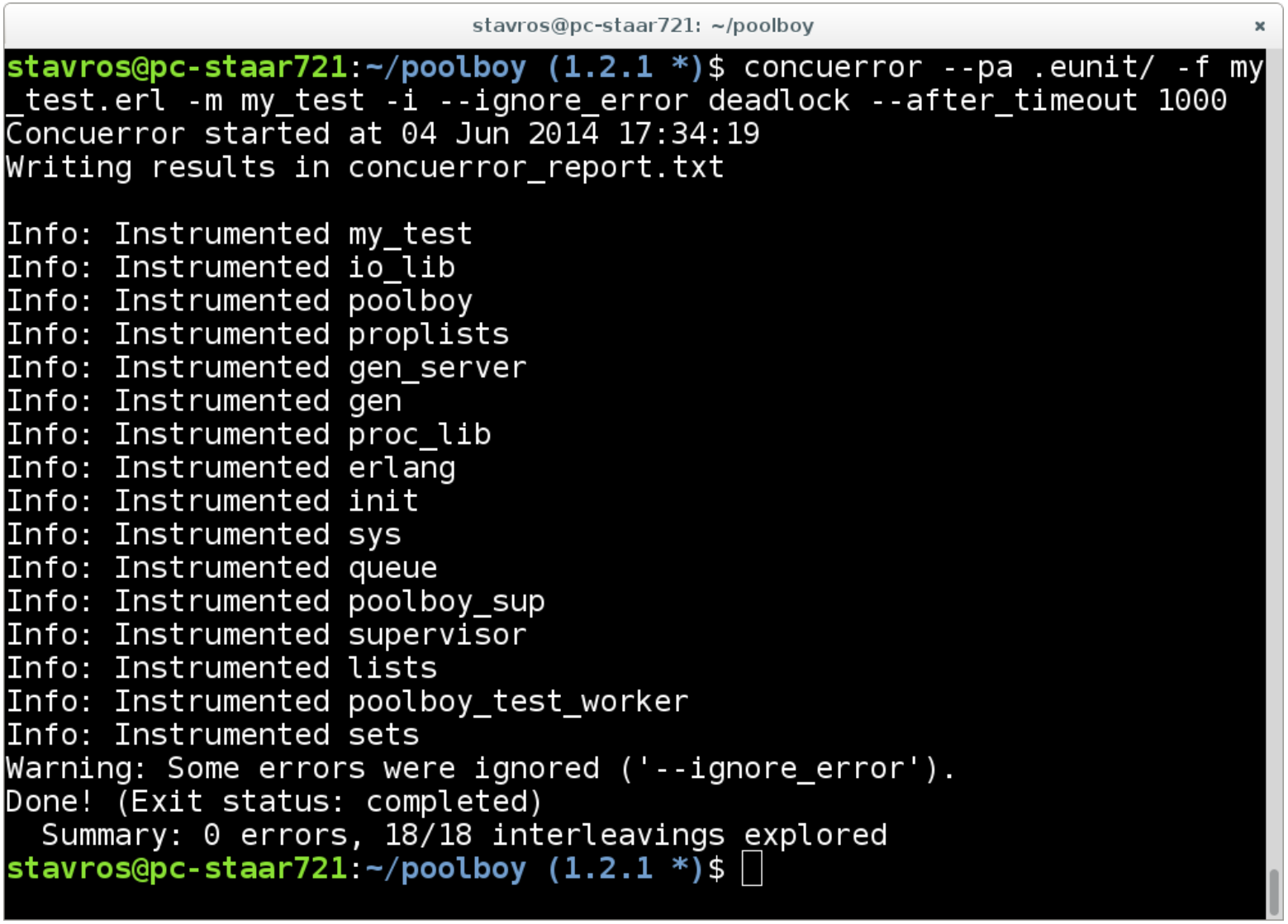
```
#!/bin/bash

OTP_PATH=../otp
CONC_PATH=../Concuerror

$CONC_PATH/concuerror -t poolboy_tests -p 0 --dpor \
    -f src/*.erl test/*.erl --wait-messages -T 2000 \
    --fail-uninstrumented \
    --ignore crypto crypto_app erl_prim_loader epp erl_parse code \
             public_key erl_syntax compile prim_file global \
    -pa . \
    \
    -I $OTP_PATH/lib/eunit/include \
    -f $OTP_PATH/lib/eunit/src/*.erl \
    \
    -I $OTP_PATH/lib/kernel/include \
    -f $OTP_PATH/lib/kernel/src/inet_parse.erl \
       $OTP_PATH/lib/kernel/src/error_logger.erl
       $OTP_PATH/lib/kernel/src/application*.erl
       $OTP_PATH/lib/kernel/src/gen_tcp.erl \
       $OTP_PATH/lib/kernel/src/inet_tcp.erl \
       $OTP_PATH/lib/kernel/src/inet.erl \
       $OTP_PATH/lib/kernel/src/inet_db.erl \
       $OTP_PATH/lib/kernel/src/inet_gethost_nati
       $OTP_PATH/lib/kernel/src/os.erl \
       $OTP_PATH/lib/kernel/src/file.erl \
    \
    -I $OTP_PATH/lib/stdlib/include \
    -f $OTP_PATH/lib/stdlib/src/dict.erl \
       $OTP_PATH/lib/stdlib/src/queue.erl \
       $OTP_PATH/lib/stdlib/src/sets.erl \
       $OTP_PATH/lib/stdlib/src/proplists.erl \
--:---  run_conc (1).sh   Top (16,37)    (Shell-sc
```



Automatic instrumentation



Automatic instrumentation

- If you need fully instrumented code, do it automatically!
- Not even **`+debug_info`** is required
- **`Instrumented erlang.erl`**?? Oh yes!

More...

More...

- Testing does not stop on the first crash
- All race-prone built-ins inspected
- Capturing stdout, stderr
- Detailed handling of exits and messaging

A process is exiting...

1. Status set to exiting
2. Name is unregistered
3. Timers are cancelled
4. ETS tables given away or destroyed
5. Link signals are sent
6. Monitor messages are sent

Concuerror follows the list step by step!

Under development...

Bounding, user interaction,
and exploration visualization

Bounding `(--delay_bound, -b)`

- Not all interleavings are equally probable
- Focus on those with “simpler” scheduling
- Classic DPOR supports *Preemption Bounding*
- Currently trying *Delay Bounding*

User interaction `(Tips)`

- Lots, lots, lots of racing events, e.g.
  - default timeouts for **`gen`** calls
  - exit signals

- Sometimes abnormal exits are acceptable
  - e.g. due to a supervisor's **`shutdown`** signal

User guidance can greatly
increase efficiency when debugging

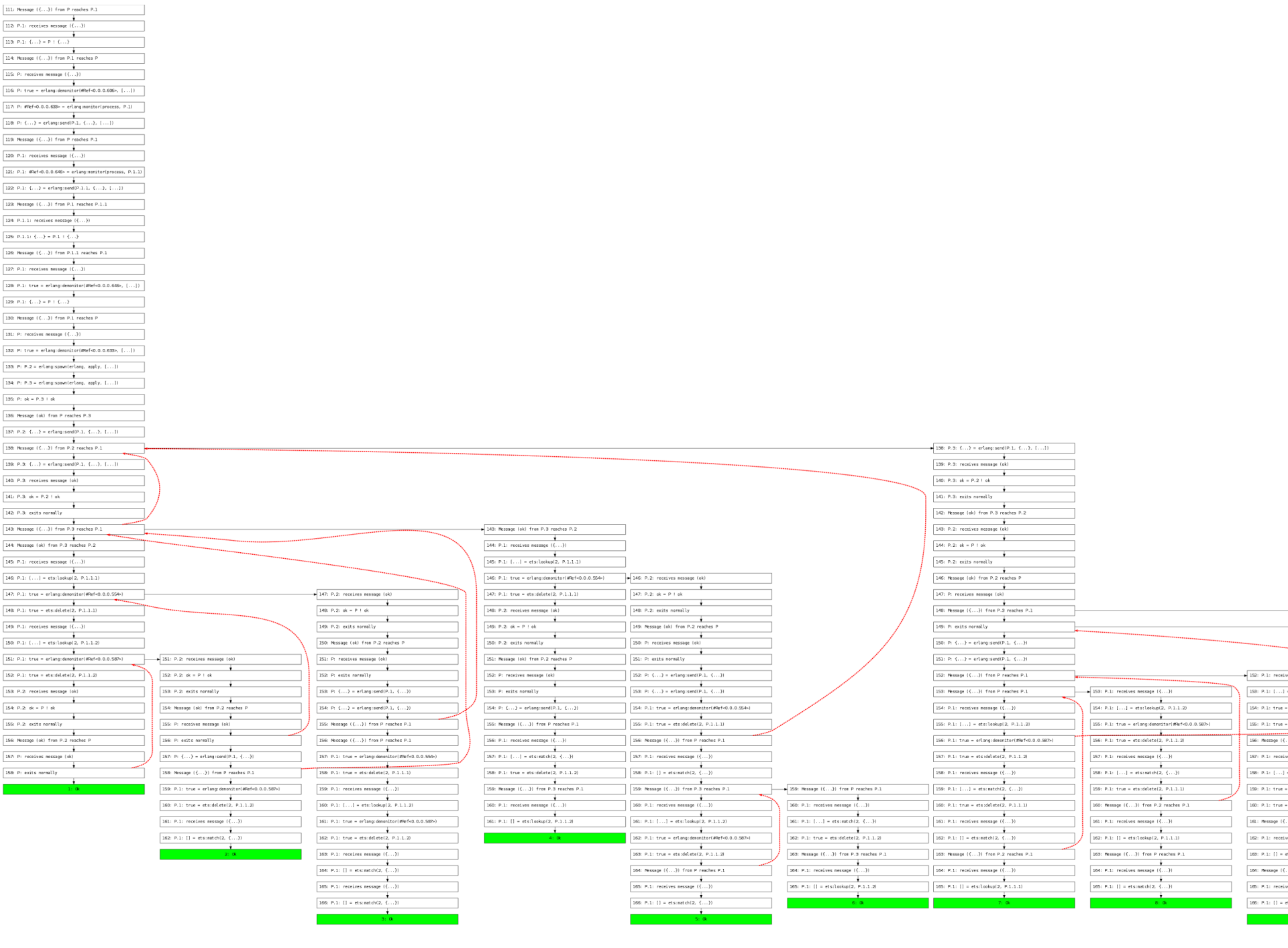
Lots, lots, lots of racing events

Example: **`erlang:register/2`**

Depends with:

- **`erlang:send/2`**
- **`erlang:unregister/1`**
- **`erlang:register/2`**
- **`erlang:whereis/1`**
- **`erlang:process_info/2`**
- **`Exit`**

Visualization `(--graph)`


111: Message ({...}) from P reaches P.1
112: P.1: receives message ({...})
113: P.1: {...} = P ! {...}
114: Message ({...}) from P.1 reaches P
115: P: receives message ({...})
116: P: true = erlang:demonitor(#Ref<0.0.0.606>, [...])
117: P: #Ref<0.0.0.633> = erlang:monitor(process, P.1)
118: P: {...} = erlang:send(P.1, {...}, [...])
119: Message ({...}) from P reaches P.1
120: P.1: receives message ({...})
121: P.1: #Ref<0.0.0.646> = erlang:monitor(process, P.1.1)
122: P.1: {...} = erlang:send(P.1.1, {...}, [...])
123: Message ({...}) from P.1 reaches P.1.1
124: P.1.1: receives message ({...})
125: P.1.1: {...} = P.1 ! {...}
126: Message ({...}) from P.1.1 reaches P.1
127: P.1: receives message ({...})
128: P.1: true = erlang:demonitor(#Ref<0.0.0.646>, [...])
129: P.1: {...} = P ! {...}
130: Message ({...}) from P.1 reaches P
131: P: receives message ({...})
132: P: true = erlang:demonitor(#Ref<0.0.0.633>, [...])
133: P: P.2 = erlang:spawn(erlang, apply, [...])
134: P: P.3 = erlang:spawn(erlang, apply, [...])
135: P: ok = P.3 ! ok
136: Message (ok) from P reaches P.3
137: P.2: {...} = erlang:send(P.1, {...}, [...])
138: Message ({...}) from P.2 reaches P.1
139: P.3: {...} = erlang:send(P.1, {...}, [...])
140: P.3: receives message (ok)
141: P.3: ok = P.2 ! ok
142: P.3: exits normally
143: Message ({...}) from P.3 reaches P.1
144: Message (ok) from P.3 reaches P.2
145: P.1: receives message ({...})
146: P.1: [...] = ets:lookup(2, P.1.1.1)
147: P.1: true = erlang:demonitor(#Ref<0.0.0.554>)
148: P.1: true = ets:delete(2, P.1.1.1)
149: P.1: receives message ({...})
150: P.1: [...] = ets:lookup(2, P.1.1.2)
151: P.1: true = erlang:demonitor(#Ref<0.0.0.587>)
152: P.1: true = ets:delete(2, P.1.1.2)
153: P.2: receives message (ok)
154: P.2: ok = P ! ok
155: P.2: exits normally
156: Message (ok) from P.2 reaches P
157: P.1: receives message (ok)
158: P: exits normally
1: Ok
151: P.2: receives message (ok)
152: P.2: ok = P ! ok
153: P.2: exits normally
154: Message (ok) from P.2 reaches P
155: P: receives message (ok)
156: P: exits normally
157: P: {...} = erlang:send(P.1, {...})
158: Message ({...}) from P reaches P.1
159: P.1: true = erlang:demonitor(#Ref<0.0.0.587>)
160: P.1: true = ets:delete(2, P.1.1.2)
161: P.1: receives message ({...})
162: P.1: [] = ets:match(2, {...})
2: Ok
147: P.2: receives message (ok)
148: P.2: ok = P ! ok
149: P.2: exits normally
150: Message (ok) from P.2 reaches P
151: P: receives message (ok)
152: P: exits normally
153: P: {...} = erlang:send(P.1, {...})
154: P: {...} = erlang:send(P.1, {...})
155: Message ({...}) from P reaches P.1
156: Message ({...}) from P reaches P.1
157: P.1: true = erlang:demonitor(#Ref<0.0.0.554>)
158: P.1: true = ets:delete(2, P.1.1.1)
159: P.1: receives message ({...})
160: P.1: [...] = ets:lookup(2, P.1.1.2)
161: P.1: true = erlang:demonitor(#Ref<0.0.0.587>)
162: P.1: true = ets:delete(2, P.1.1.2)
163: P.1: receives message ({...})
164: P.1: [] = ets:match(2, {...})
165: P.1: receives message ({...})
166: P.1: [] = ets:match(2, {...})
3: Ok
143: Message (ok) from P.3 reaches P.2
144: P.1: receives message ({...})
145: P.1: [...] = ets:lookup(2, P.1.1.1)
146: P.1: true = erlang:demonitor(#Ref<0.0.0.554>)
147: P.1: true = ets:delete(2, P.1.1.1)
148: P.2: receives message (ok)
149: P.2: ok = P ! ok
150: P.2: exits normally
151: Message (ok) from P.2 reaches P
152: P: receives message (ok)
153: P: exits normally
154: P: {...} = erlang:send(P.1, {...})
155: Message ({...}) from P reaches P.1
156: P.1: receives message ({...})
157: P.1: [...] = ets:match(2, {...})
158: P.1: true = ets:delete(2, P.1.1.2)
159: Message ({...}) from P.3 reaches P.1
160: P.1: receives message ({...})
161: P.1: [] = ets:lookup(2, P.1.1.2)
4: Ok
146: P.2: receives message (ok)
147: P.2: ok = P ! ok
148: P.2: exits normally
149: Message (ok) from P.2 reaches P
150: P: receives message (ok)
151: P: exits normally
152: P: {...} = erlang:send(P.1, {...})
153: P: {...} = erlang:send(P.1, {...})
154: P.1: true = erlang:demonitor(#Ref<0.0.0.554>)
155: P.1: true = ets:delete(2, P.1.1.1)
156: Message ({...}) from P reaches P.1
157: P.1: receives message ({...})
158: P.1: [] = ets:match(2, {...})
159: Message ({...}) from P.3 reaches P.1
160: P.1: receives message ({...})
161: P.1: [...] = ets:lookup(2, P.1.1.2)
162: P.1: true = erlang:demonitor(#Ref<0.0.0.587>)
163: P.1: true = ets:delete(2, P.1.1.2)
164: Message ({...}) from P reaches P.1
165: P.1: receives message ({...})
166: P.1: [] = ets:match(2, {...})
5: Ok
159: Message ({...}) from P reaches P.1
160: P.1: receives message ({...})
161: P.1: [...] = ets:match(2, {...})
162: P.1: true = ets:delete(2, P.1.1.2)
163: Message ({...}) from P.3 reaches P.1
164: P.1: receives message ({...})
165: P.1: [] = ets:lookup(2, P.1.1.2)
6: Ok
138: P.3: {...} = erlang:send(P.1, {...}, [...])
139: P.3: receives message (ok)
140: P.3: ok = P.2 ! ok
141: P.3: exits normally
142: Message (ok) from P.3 reaches P.2
143: P.2: receives message (ok)
144: P.2: ok = P ! ok
145: P.2: exits normally
146: Message (ok) from P.2 reaches P
147: P: receives message (ok)
148: Message ({...}) from P.3 reaches P.1
149: P: exits normally
150: P: {...} = erlang:send(P.1, {...})
151: P: {...} = erlang:send(P.1, {...})
152: Message ({...}) from P reaches P.1
153: Message ({...}) from P reaches P.1
154: P.1: receives message ({...})
155: P.1: [...] = ets:lookup(2, P.1.1.2)
156: P.1: true = erlang:demonitor(#Ref<0.0.0.587>)
157: P.1: true = ets:delete(2, P.1.1.2)
158: P.1: receives message ({...})
159: P.1: [...] = ets:match(2, {...})
160: P.1: true = ets:delete(2, P.1.1.1)
161: P.1: receives message ({...})
162: P.1: [] = ets:match(2, {...})
163: Message ({...}) from P.2 reaches P.1
164: P.1: receives message ({...})
165: P.1: [] = ets:lookup(2, P.1.1.1)
7: Ok
152: P.1: receive
153: P.1: [...] =
154: P.1: true =
155: P.1: true =
156: Message ({...
157: P.1: receive
158: P.1: [...] =
159: P.1: true =
160: Message ({...
161: P.1: receive
162: P.1: receive
163: P.1: [] = et
164: Message ({...
165: P.1: receive
166: P.1: [] = et
8: Ok

Next Challenges

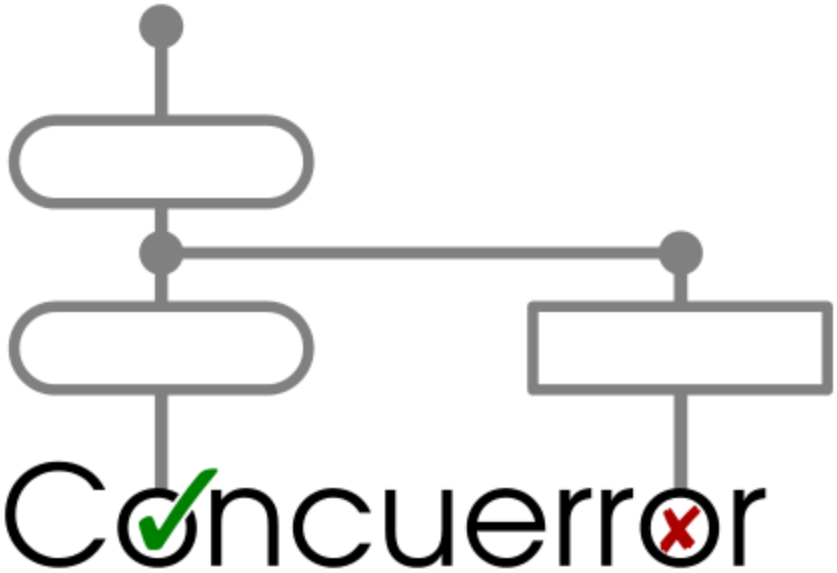
- System processes (e.g. `application`)
- Ports (and therefore file manipulation)
- Concuerror on Concuerror (on Concuerror...)

Conclusion

http://concuerror.com

Go give Concuerror a try!

- Efficient, systematic concurrency testing
- Usability and practicality are design goals
- Open source, feedback is appreciated
- **`concuerror --help`**



Thank you!

http://concuerror.com

RELEASE

UPPSALA UNIVERSITET