Do this
then do that
check
✓ Looks ok

<0.32.0>

then I do this
then I do that
and I check
✓ Looks ok too

<0.35.0>

Testing...
Test for

<0.32.0>

Do this

then do that

check

✗ Not so ok

<0.35.0>

then I do this

then I do that

“reversed”
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

Interleaving 1

Interleaving 2

3. <0.35.0>
4. <0.32.0>
5. <0.32.0>
6. <0.32.0> Crashes
Into Real Code

Stavros Aronis
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 $\neq$ 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
Why not “start” the interleaving here?
Answer: paper presented @ POPL’14!
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

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Interleavings explored</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Classic</td>
<td>Optimal</td>
</tr>
<tr>
<td>readers (2)</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>readers (8)</td>
<td>3281</td>
<td>256</td>
</tr>
<tr>
<td>readers (13)</td>
<td>797162</td>
<td>8192</td>
</tr>
<tr>
<td>lastzero (5)</td>
<td>241</td>
<td>64</td>
</tr>
<tr>
<td>lastzero (10)</td>
<td>53198</td>
<td>3328</td>
</tr>
<tr>
<td>lastzero (15)</td>
<td>9378091</td>
<td>147456</td>
</tr>
</tbody>
</table>

Difference between Classic and Optimal
## POPL’14: Evaluation

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Interleavings explored</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Classic</td>
<td>Optimal</td>
</tr>
<tr>
<td>dialyzer</td>
<td>12436</td>
<td>3600</td>
</tr>
<tr>
<td>gproc</td>
<td>14080</td>
<td>8104</td>
</tr>
<tr>
<td>poolboy</td>
<td>6018</td>
<td>2680</td>
</tr>
</tbody>
</table>

LOC: 44596 (dialyzer), 9446 (gproc), 79732 (poolboy)
Optimal DPOR: Summarry

- Not all pairs of events are racing!

- Concurerror will never even begin to explore equivalent interleavings

- Trace analysis, intelligent algorithms, tailored dependency tracking for Erlang built-ins
Automatic instrumentation
Interleaving 1

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

Interleaving 2

3. <0.35.0>
4. <0.32.0>
5. <0.32.0>
6. <0.32.0> Crashes
Automatic instrumentation

```
#!/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 .
   
   -l $OTP_PATH/lib/eunit/include
   -f $OTP_PATH/lib/eunit/src/*.erl
   
   -l $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_natio
       $OTP_PATH/lib/kernel/src/os.erl
       $OTP_PATH/lib/kernel/src/file.erl
       
   -l $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-source)
```
Automatic instrumentation

stavros@pc-staar721:~/poolboy (1.2.1 *)$ concuerror --pa .eunit/ -f my_test.erl -m my_test -i --ignore_error deadlock --after_timeout 1000
Concuerror started at 04 Jun 2014 17:34:19
Writing results in concuerror_report.txt

Info: Instrumented my_test
Info: Instrumented io_lib
Info: Instrumented poolboy
Info: Instrumented proplists
Info: Instrumented gen_server
Info: Instrumented gen
Info: Instrumented proc_lib
Info: Instrumented erlang
Info: Instrumented init
Info: Instrumented sys
Info: Instrumented queue
Info: Instrumented poolboy_sup
Info: Instrumented supervisor
Info: Instrumented lists
Info: Instrumented poolboy_test_worker
Info: Instrumented sets
Warning: Some errors were ignored ('--ignore_error').
Done! (Exit status: completed)
  Summary: 0 errors, 18/18 interleavings explored
stavros@pc-staar721:~/poolboy (1.2.1 *)$
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 \texttt{gen} calls
  - exit signals

- Sometimes abnormal exits are acceptable
  - e.g. due to a supervisor’s \texttt{shutdown} signal

\textit{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
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