Adventures in Corfu:
Testing and Verifying Chain Repair Protocols Using Concuerror

Kostis Sagonas & Stavros Aronis

UPPSALA UNIVERSITET

Concuerror

By Eric Gaba (Sting - fr:Sting) - Own work; Data sources:Topography: NASA Shuttle Radar Topography Mission (SRTM3 v.2) (public domain) https://commons.wikimedia.org/w/index.php?curid=1862405

Scott L. Fritchie
@slfritchie

Following

I was all ready to have a celebratory "New algorithm works!" tweet. Then the DPOR model execution w/Concuerror found an invalid case. Ouch.

RETWEET 1 LIKES 5

9:16 AM - 23 Jun 2016

1 5

The adventure of this EUC talk starts with a tweet

Scott Lystig Fritchie

"Jack-of-many bit-centric trades, many of them Erlang flavored."

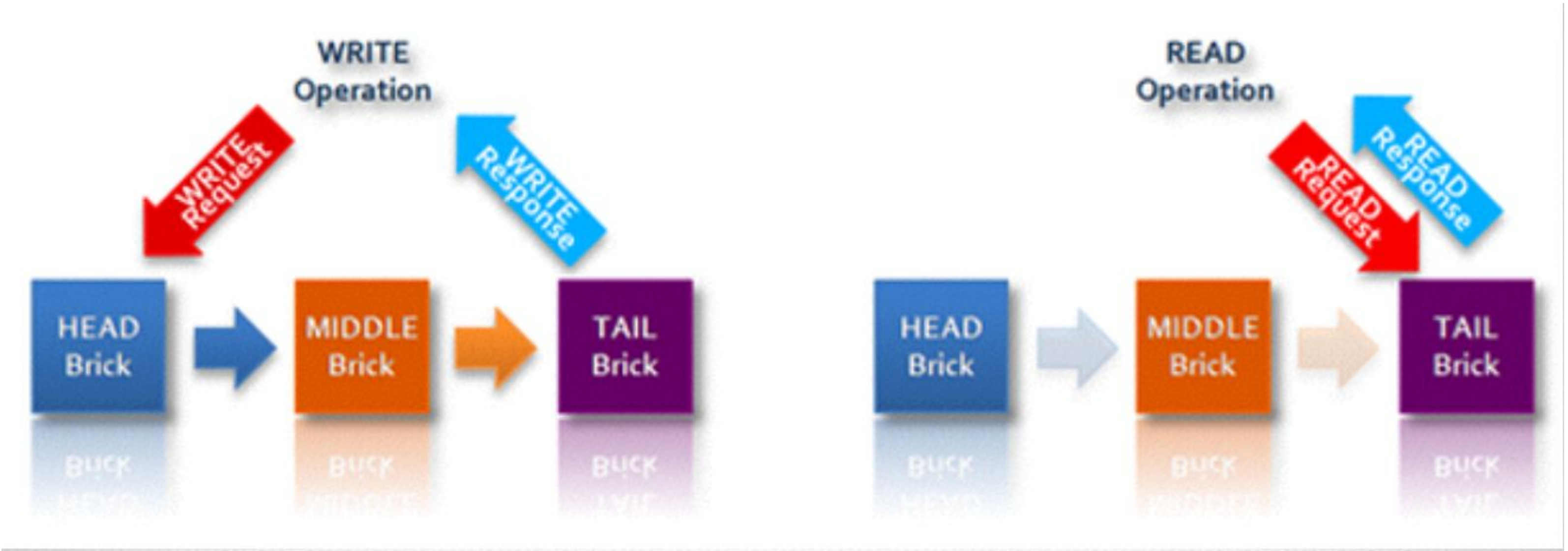
- Sendmail, Inc.
- Gemini Mobile (Hibari)
- Erlang/OTP (DTrace)
- Basho (Riak, Machi)
- VMware (CorfuDB)

Talk Overview

- Chain Replication and Chain Repair
- Systematic Concurrency Testing
- Concuerror (demo)
- Our CORFU case study experience
- Concuerror improvements and their impact

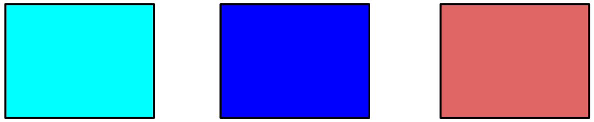
Chain Replication

- A variant of master/slave replication
- Strict chain order!

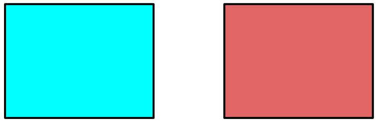
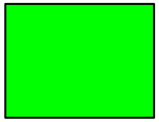


- Sequential read @ tail.
- Linearizable read @ all.
- Dirty read @ head or middle.

Chain Repair

Let's say we have chain of three servers 

Naive offline repair method:

1. Stop all surviving servers in the chain 
2. Copy tail's update history to the repairing node 
3. Restart all nodes with the configuration

HibariDB's repair is similar but places the repairing node directly on the chain and reads go to  (the old tail)

CORFU

Uses Chain Replication with three changes

1. Responsibility for replication is moved to the client
    - Clients do not communicate with each other
2. CORFU's servers implement write-once semantics
3. Identifies each chain configuration with an epoch #
    - All clients and servers are aware of the epoch #
    - The server rejects clients with a different epoch #
    - A server temporarily stops service if it receives a newer epoch # from a client

Chain Repair in CORFU

A repair during epoch #5: a client is writing a *new* value to the cluster for a data with *old* value

epoch #5	S^a_{head} value=*new*	S^b_{tail} value=*old* or value=*new*	S^c_{repair} value=*old*
epoch #6	S^a_{head} value=*new*	S^b_{middle} value=*new*	S^c_{tail} value=*old* or value=*new*

There is a race condition here, which can lead to a violation of the linearizability property

Stateless Model Checking

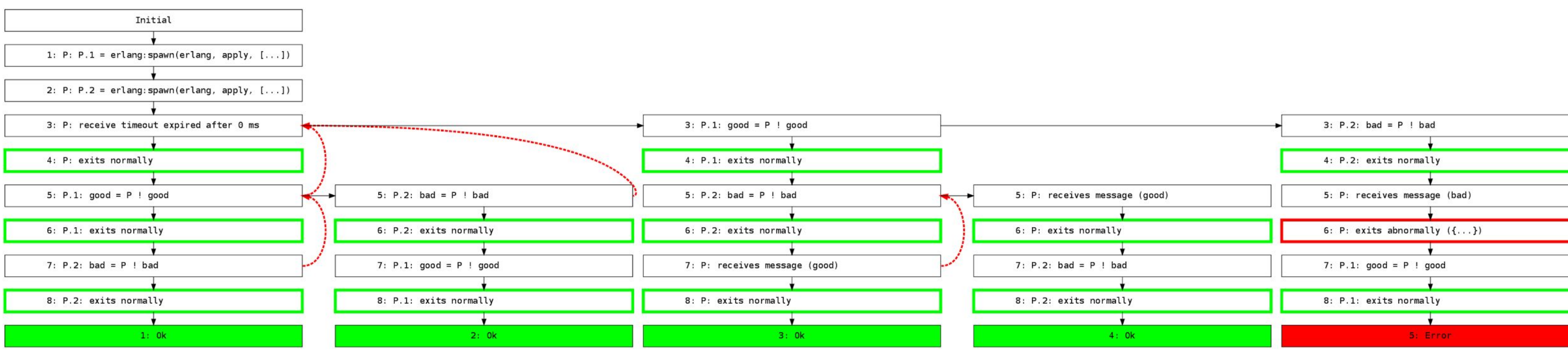
Systematic Concurrency Testing

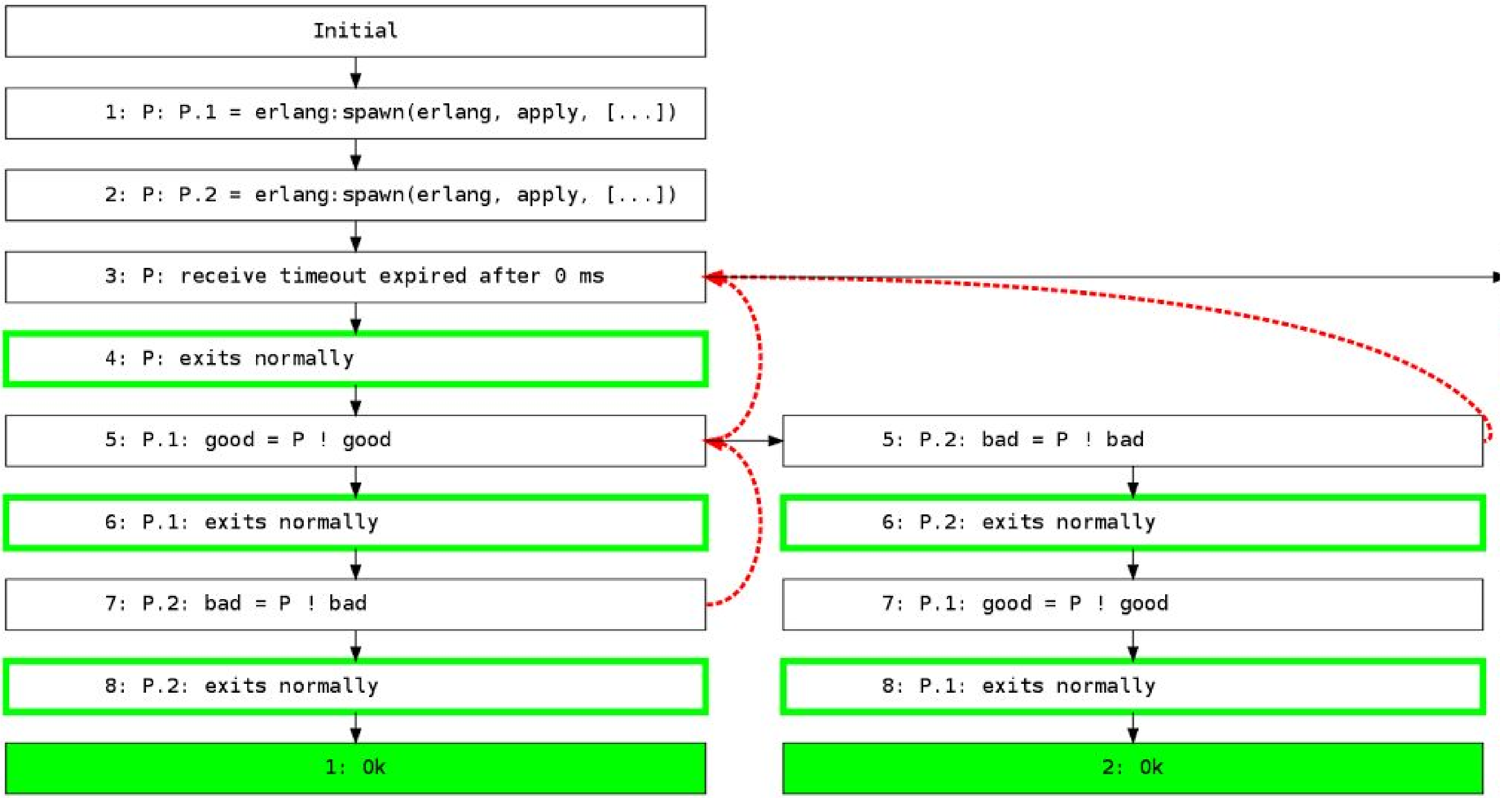
Systematic Concurrency Testing

- Assume that you only have one ‘scheduler’:
  - Run an arbitrary execution...
- Then:
  - Backtrack to a point where some other process could have been chosen to run (pick the latest)…
  - From there, continue with another execution…
- Repeat until all choices have been explored.

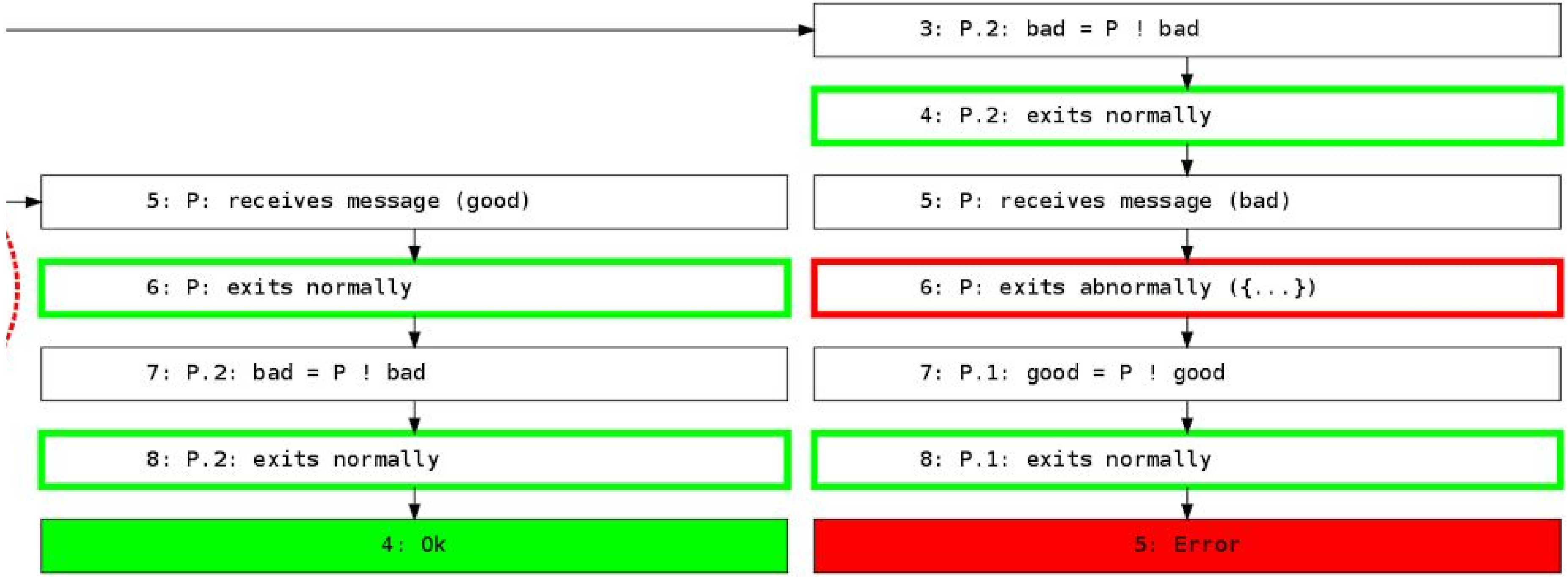
```erlang
-module(foo).
-export([main/0]).

main() ->
  P = self(),
  _P1 = spawn(fun () -> M = bar:good(P) end),
  _P2 = spawn(fun () -> M = bar:bad(P) end),
  receive
     good -> …, ok;
     bad  -> …, throw(error);
     _Msg -> …, ok
  after 0 -> …, ok
  end.
```

```erlang
-module(bar).
-export([good/1,...,ugly/1]).

good(P) -> ..., P ! good.

bad(P) -> …, P ! bad.

ugly(P) -> ... , P ! ugly.
```





Initial
1: P: P.1 = erlang:spawn(erlang, apply, [...])
2: P: P.2 = erlang:spawn(erlang, apply, [...])
3: P: receive timeout expired after 0 ms
4: P: exits normally
5: P.1: good = P ! good
6: P.1: exits normally
7: P.2: bad = P ! bad
8: P.2: exits normally
1: Ok
5: P.2: bad = P ! bad
6: P.2: exits normally
7: P.1: good = P ! good
8: P.1: exits normally
2: Ok



3: P.2: bad = P ! bad
4: P.2: exits normally
5: P: receives message (good)
5: P: receives message (bad)
6: P: exits normally
6: P: exits abnormally ({...})
7: P.2: bad = P ! bad
7: P.1: good = P ! good
8: P.2: exits normally
8: P.1: exits normally
4: Ok
5: Error

Concuerror

- A **stateless model checking** tool that
- … runs a test under **all** possible interleavings
- … detects abnormal process exits
- … reports all the events that lead to a crash

Systematic =/= Stupid

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

Fighting Combinatorial Explosion

Optimal Dynamic Partial Order Reduction

- … monitors **dependencies** between events
- … explores additional interleavings **as needed**
- … completely avoids **equivalent** interleavings

- **Dynamic**: at runtime, using concrete data
- **Optimal**: explores only different interleavings

Bounding

Do not explore all interleavings, but only a selected few based on some bounding criterion

- E.g., number of times processes can be preempted, delayed, etc.

Back to the CORFU adventure

Correctness Properties

Immutability:

- Once a value has been written in a key, no other value can be written to it

Linearizability:

- If a read sees a value for a key, subsequent reads for that key must also see the same value

Modeling CORFU

Initial model:

- Some (one or two) servers undergo a chain repair to add one more server to their chain
- Concurrently, two other clients try to write two different values to the same key
- While a third client tries to read the key twice

Modeling CORFU (cont.)

- Servers and clients are modeled as Erlang processes
- All requests are modeled as messages

Processes used by the model

  - Central coordinator
  - CORFU log servers (2 or 3)
  - Layout server process
  - CORFU reading client
  - CORFU writing clients (2)
  - Layout change and data repair process

Three Repair Methods

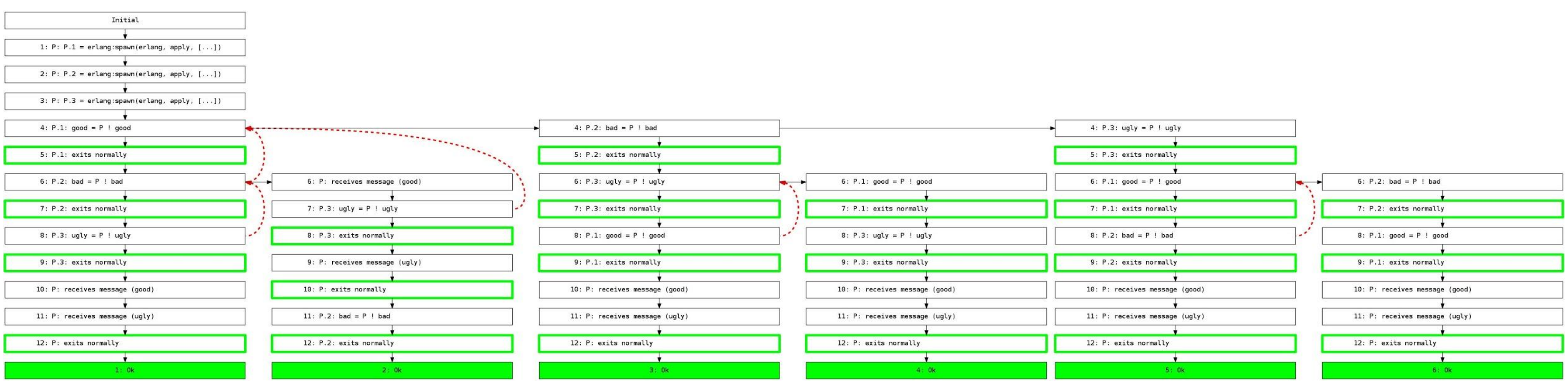
1. Add repair node at the end of chain
2. Add repair node at the start of chain
3. Add repair node in the middle
    a. Configuration with two healthy servers
    b. Configuration with one healthy server which is “logically split” into two

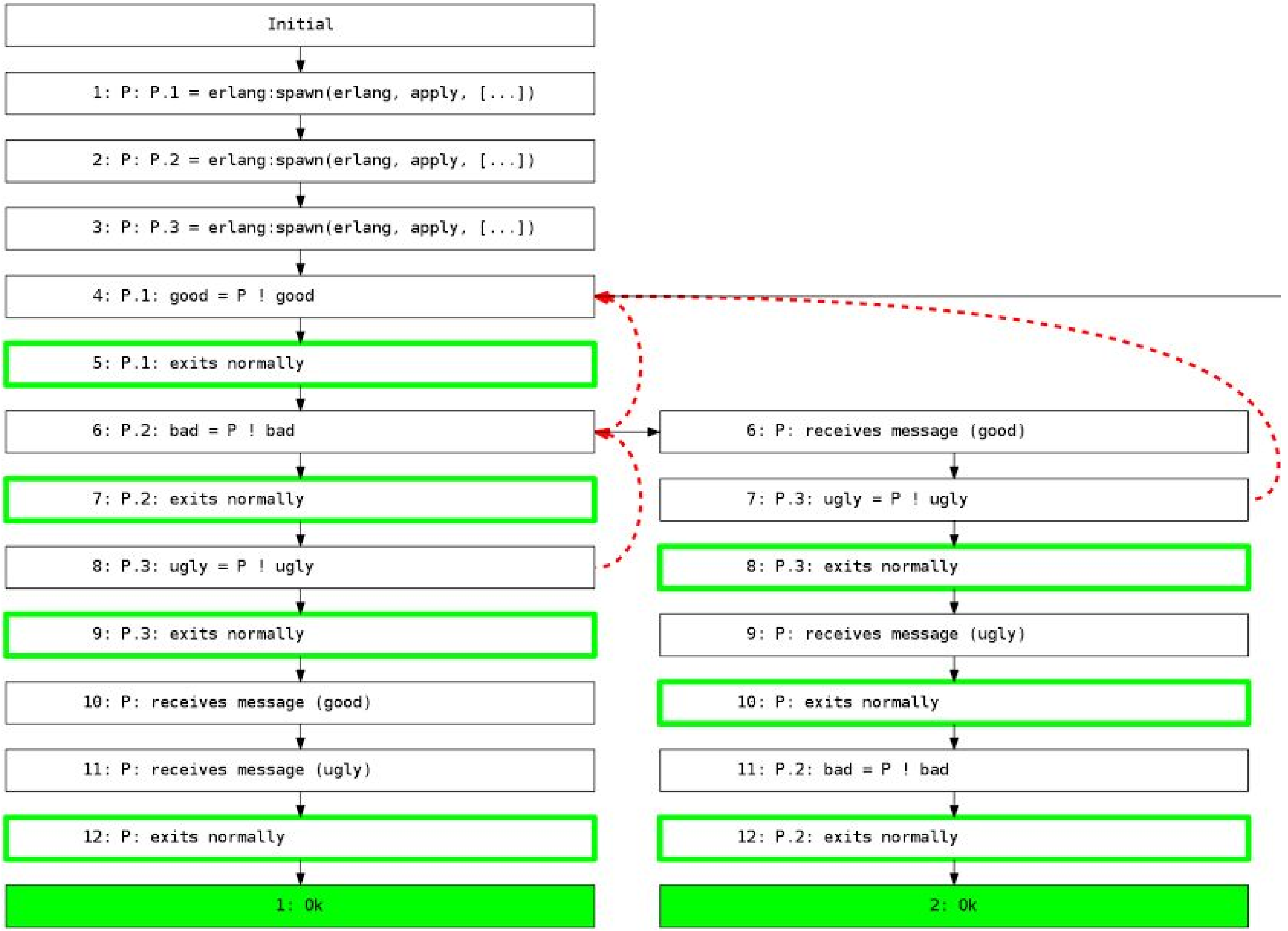
Results in vanilla Concuerror

Method	Bounded Exploration			Unbounded Exploration		
	Bug?	Traces	Time	Bug?	Traces	Time
1 (Tail)	Yes	638	57s	Yes	3 542 431	144h
2 (Head)	Yes	65	7s	Yes	389	26s
3 (Middle)	No	1257	68s	No	>30 000 000	>750h

```erlang
-module(foo2).
-export([main/0]).

main() ->
  P = self(),
  _P1 = spawn(fun () -> M = bar:good(P) end),
  _P2 = spawn(fun () -> M = bar:bad(P) end),
  _P3 = spawn(fun () -> M = bar:ugly(P) end),
  receive
    good -> …, ok
  end,
  receive
    ugly  -> …, ok
  end.
```

```erlang
-module(bar).
-export([good/1,...,ugly/1]).

good(P) -> ..., P ! good.

bad(P) -> …, P ! bad.

ugly(P) -> ... , P ! ugly.
```





Initial
1: P: P.1 = erlang:spawn(erlang, apply, [...])
2: P: P.2 = erlang:spawn(erlang, apply, [...])
3: P: P.3 = erlang:spawn(erlang, apply, [...])
4: P.1: good = P ! good
5: P.1: exits normally
6: P.2: bad = P ! bad
7: P.2: exits normally
8: P.3: ugly = P ! ugly
9: P.3: exits normally
10: P: receives message (good)
11: P: receives message (ugly)
12: P: exits normally
1: Ok
6: P: receives message (good)
7: P.3: ugly = P ! ugly
8: P.3: exits normally
9: P: receives message (ugly)
10: P: exits normally
11: P.2: bad = P ! bad
12: P.2: exits normally
2: Ok

Optimization (in Concuerror)

- Treating blocking receives, whose message patterns are all known, specially
- Avoids exploring an exponential number of "unnecessary" interleavings from sends

In CORFU's initial model, this happened in the coordinator in code like the following

```
…
receive
  {done, client_1} -> …       % block until client_1 is done
end,
...
```

Model Refinements

1. Conditional read

   Avoid issuing read operations that are sure to not result in violations

2. Convert layout server process to an ETS table

Effect of Model Refinements

Method #1 (repair node in the head)

Even without bounding, the error is found in 19 secs only (212 traces)

Method #3 (repair node in the middle)

Concuerror verifies the method

- in 48 hours
- exploring 3 931 412 traces

Conclusion

http://concuerror.com

Go give Concuerror a try!

- Efficient tool to test and verify concurrent Erlang programs (and algorithms!)
- Usability and practicality <u>are</u> design goals
- Open source, feedback is appreciated
- `concuerror --help`

Code

`github.com/aronisstav`

`cr-concuerror-experiments`

Concuerror

http://concuerror.com