ParaFuzz: Fuzzing Multicore OCaml programs

“KC” Sivaramakrishnan

joint work with
Sumit Padhiyar and Adharsh Kamath

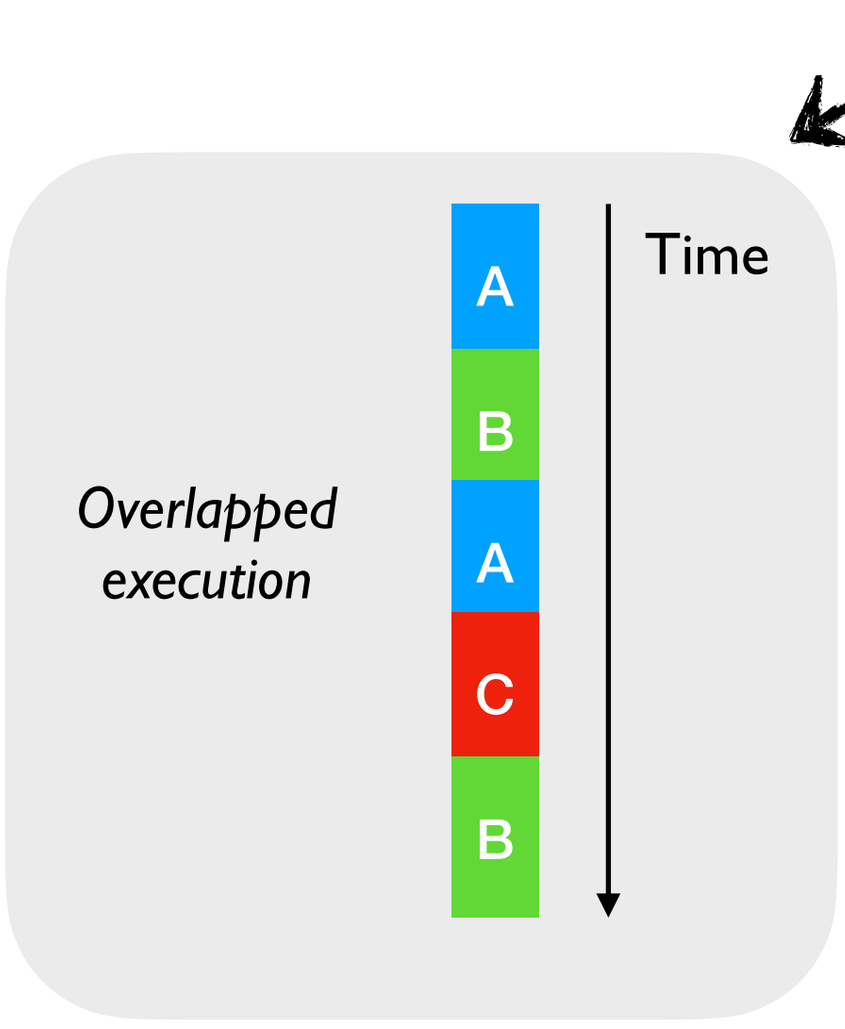
IIT MADRAS

Multicore OCaml

- Adds native support for *concurrency* and *parallelism* to OCaml

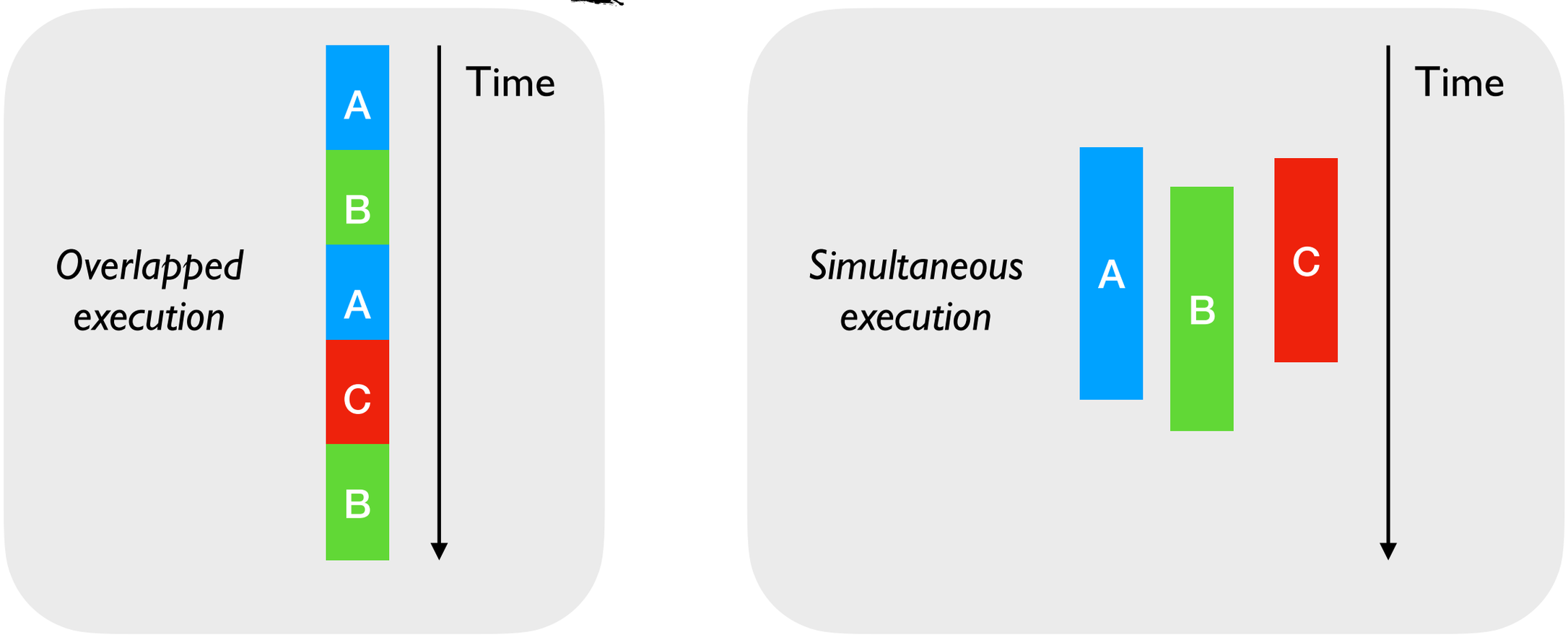
Multicore OCaml

- Adds native support for *concurrency* and *parallelism* to OCaml



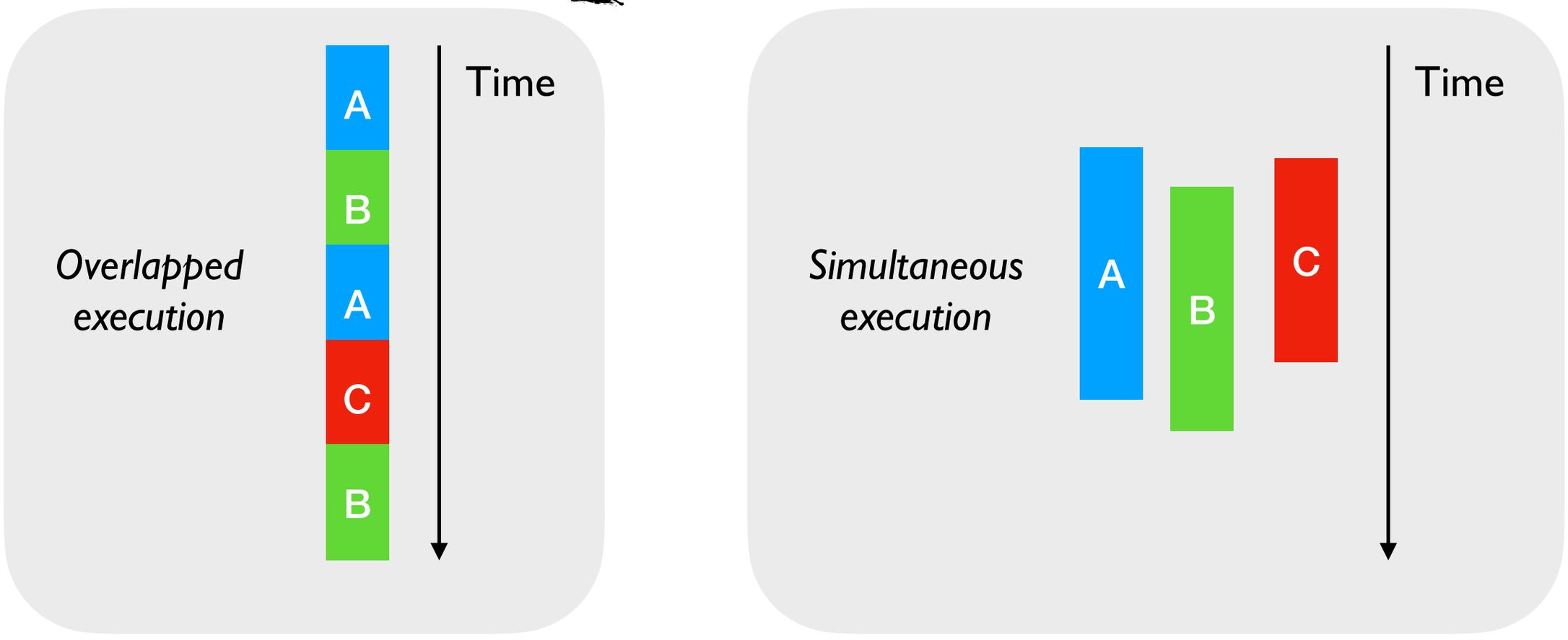
Multicore OCaml

- Adds native support for *concurrency* and *parallelism* to OCaml



Multicore OCaml

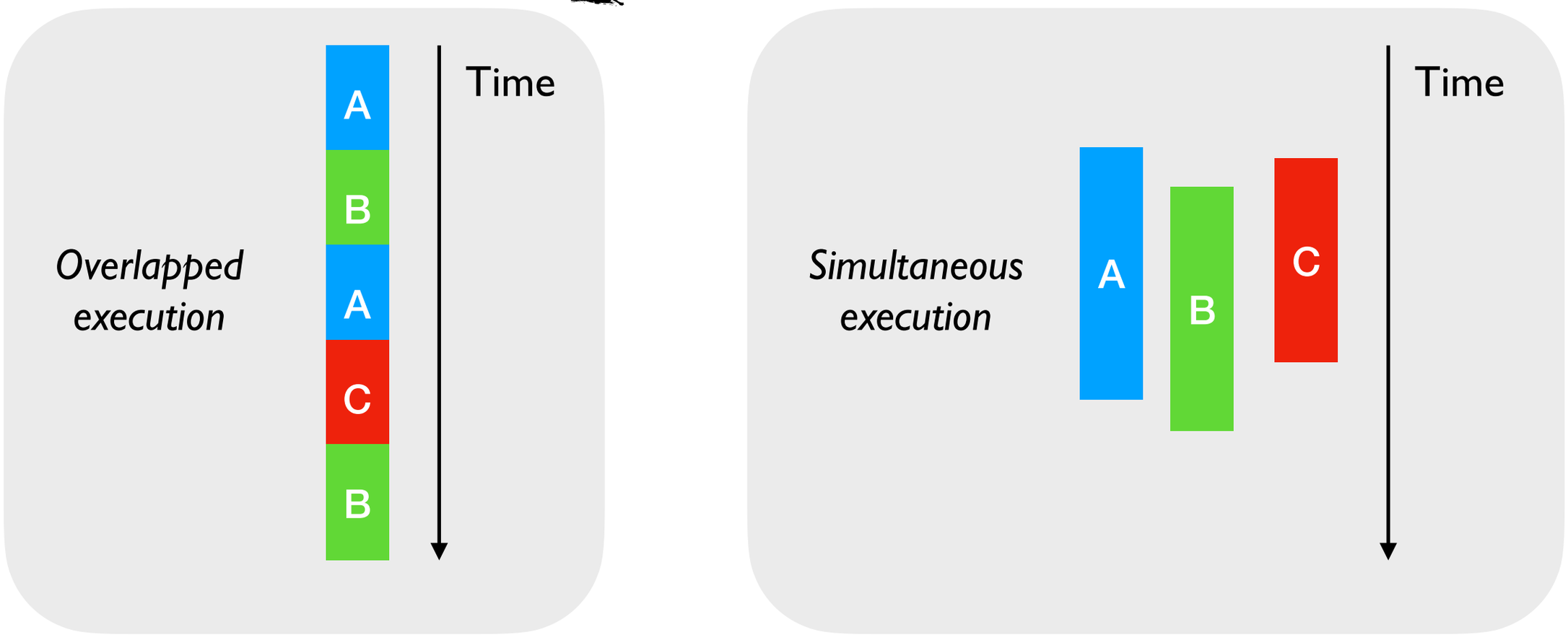
- Adds native support for *concurrency* and *parallelism* to OCaml



Effect Handlers

Multicore OCaml

- Adds native support for *concurrency* and *parallelism* to OCaml



Effect Handlers

Domains

Testing Parallel Programs

```
let test i =
  let x = Atomic.make i in
  let y = Atomic.make 0 in
  let dom = Domain.spawn (fun () ->
    if (Atomic.get x = 10) then Atomic.set y 2)
  in
  Atomic.set x 0;
  Atomic.set y 1;
  Domain.join dom;
  assert (Atomic.get y <> 2)
```

Testing Parallel Programs

```
let test i =
  let x = Atomic.make i in
  let y = Atomic.make 0 in
  let dom = Domain.spawn (fun () ->
    if (Atomic.get x = 10) then Atomic.set y 2)
  in
  Atomic.set x 0;
  Atomic.set y 1;
  Domain.join dom;
  assert (Atomic.get y <> 2)
```

- The assertion can fail for a particular *input* and *scheduling* combination

Testing Parallel Programs

```
let test i =
  let x = Atomic.make i in
  let y = Atomic.make 0 in
  let dom = Domain.spawn (fun () ->
    if (Atomic.get x = 10) then Atomic.set y 2)
  in
  Atomic.set x 0;
  Atomic.set y 1;
  Domain.join dom;
  assert (Atomic.get y <> 2)
```

- The assertion can fail for a particular *input* and *scheduling* combination
- Logic bugs are more than just detecting data races
  - ✦ *No data races here*

Testing Parallel Programs

```
let test i =
  let x = Atomic.make i in
  let y = Atomic.make 0 in
  let dom = Domain.spawn (fun () ->
    if (Atomic.get x = 10) then Atomic.set y 2)
  in
  Atomic.set x 0;
  Atomic.set y 1;
  Domain.join dom;
  assert (Atomic.get y <> 2)
```

- The assertion can fail for a particular *input* and *scheduling* combination
- Logic bugs are more than just detecting data races
  - ✦ *No data races here*
- *How can we help test Multicore OCaml programmers detect such bugs?*

Existing solutions

- **Testing**
  - ✦ Stress testing — run the program over and over again and hope that the assertion is triggered
  - ✦ Random testing — generate random inputs, and perturb the OS scheduler (*somehow*) to trigger bugs

Existing solutions

- **Testing**
  - ✦ Stress testing — run the program over and over again and hope that the assertion is triggered
  - ✦ Random testing — generate random inputs, and perturb the OS scheduler (*somehow*) to trigger bugs
- **Model checking** — SPIN, TLC model checkers
  - ✦ Strong guarantees, but not practical with limited time budget
  - ✦ Often works on a model of the program and not directly on the source code

Existing solutions

- **Testing**
  - ✦ Stress testing — run the program over and over again and hope that the assertion is triggered
  - ✦ Random testing — generate random inputs, and perturb the OS scheduler (*somehow*) to trigger bugs
- **Model checking** — SPIN, TLC model checkers
  - ✦ Strong guarantees, but not practical with limited time budget
  - ✦ Often works on a model of the program and not directly on the source code
- **Formal verification**
  - ✦ Requires expert knowledge and lots of time and effort

Our Approach

- Ignore concurrency for the moment, and let's focus on input non-determinism

Our Approach

- Ignore concurrency for the moment, and let's focus on input non-determinism
- Property-based testing
  - ✦ Use a generator to generate random inputs to test a function
  - ✦ *Quick-check*

Our Approach

- Ignore concurrency for the moment, and let's focus on input non-determinism
- Property-based testing
  - ✦ Use a generator to generate random inputs to test a function
  - ✦ *Quick-check*
- Fuzzing
  - ✦ Generate random inputs to crash a program
  - ✦ *AFL* — Extremely effective grey-box (coverage-guided) fuzzer

Our Approach

- Ignore concurrency for the moment, and let's focus on input non-determinism
- Property-based testing
  - ✦ Use a generator to generate random inputs to test a function
  - ✦ *Quick-check*
- Fuzzing
  - ✦ Generate random inputs to crash a program
  - ✦ *AFL* — Extremely effective grey-box (coverage-guided) fuzzer
- *Crowbar* = Fuzzing + QuickCheck
  - ✦ Coverage-guided property-fuzzing
  - ✦ https://github.com/stedolan/crowbar

ParaFuzz

ParaFuzz

- ParaFuzz = Crowbar (Grey-box Fuzzing + Property-based testing) + Parallelism

ParaFuzz

- ParaFuzz = Crowbar (Grey-box Fuzzing + Property-based testing) + Parallelism
- *How to control parallel thread scheduling?*

ParaFuzz

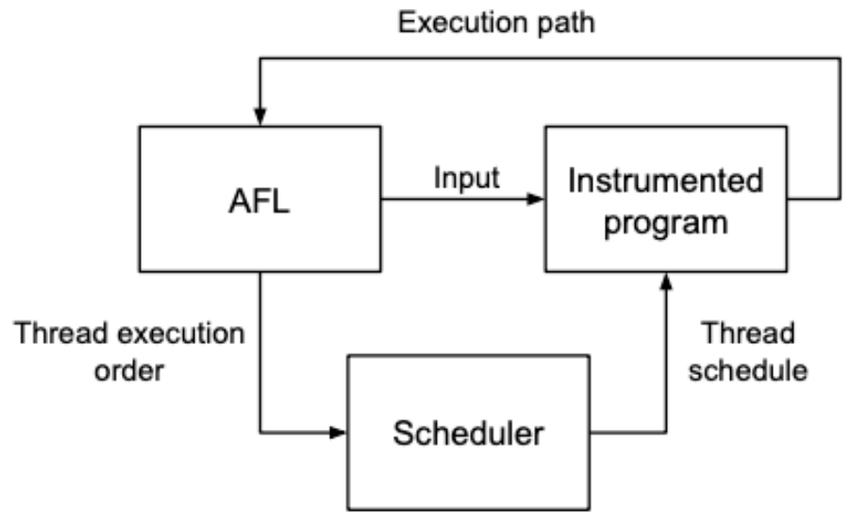
- ParaFuzz = Crowbar (Grey-box Fuzzing + Property-based testing) + Parallelism
- *How to control parallel thread scheduling?*
- Idea
  - ✦ *Mock parallelism API using an effect handler based scheduler*
  - ✦ *Yield at every synchronisation point*
    - ✤ *Use AFL to pick next thread to run from the queue of ready threads*

ParaFuzz

- ParaFuzz = Crowbar (Grey-box Fuzzing + Property-based testing) + Parallelism
- *How to control parallel thread scheduling?*
- Idea
  - ✦ *Mock parallelism API using an effect handler based scheduler*
    - ✦ *Yield at every synchronisation point*
      - ✤ *Use AFL to pick next thread to run from the queue of ready threads*
- Synchronisation points
  - ✦ Domain (spawn, join)
  - ✦ Atomic (get, put, compare_and_swap)
  - ✦ Mutex (lock, unlock)
  - ✦ Condition variable (wait, notify, broadcast) — also fuzz wake up order

ParaFuzz

- Idea
  - ✦ *Mock parallelism API using an effect handler based scheduler*
  - ✦ *Use AFL to pick the thread scheduling order at synchronisation points*



- Advantages
  - ✦ No false positives
  - ✦ Record and replay
  - ✦ Drop-in replacement for parallel Multicore OCaml programs

Evaluation

```
let test i =
  let x = Atomic.make i in
  let y = Atomic.make 0 in
  let dom = Domain.spawn (fun () ->
    if (Atomic.get x = 10) then Atomic.set y 2)
  in
  Atomic.set x 0;
  Atomic.set y 1;
  Domain.join dom;
  assert (Atomic.get y <> 2)
```

Testing Technique	Executions (millions)	Time (minutes)	Bug Found
ParaFuzz	0.55	10.5	**Yes**
Random	108.6	60	**No**
Stress	25.2	60	**No**

Evaluation

Name(abbreviation)	Bug type
mysql-bug(SQL)	race-condition
circular-list(CL)	race-condition
deadlock3(D3)	deadlock
buffer-if(BI)	deadlock
buffer-notify(BN)	deadlock
RAX-jpf(RAX)	deadlock
domainslib(DL)	deadlock
motivating-example(MX)	race-condition
effective-random-testing-example (ERT)	race-condition

	Stress	Random	ParaFuzz
SQL	0.00	0.00	1.00
CL	0.00	0.00	0.96
D3	0.00	0.00	1.00
BI	0.00	0.03	1.00
BN	0.00	0.00	1.00
RAX	0.00	0.00	1.00
DL	0.00	1.00	1.00
MX	0.00	0.00	1.00
ERT	0.00	0.00	1.00
Avg	0.00	0.003	0.99

Effectiveness

fraction of runs that found the bug

	Stress	Random	ParaFuzz
SQL	-	-	734.26
CL	-	-	971.16
D3	-	-	469.63
BI	-	100.76	20.36
BN	-	-	875.4
RAX	-	-	111
DL	-	0	0
MX	-	-	625.36
ERT	-	-	88.83

Efficiency

Mean-time to failure

Data races

Data races

- ParaFuzz currently assumes that the programs are data-race-free (DRF)
  - ✦ DRF programs in OCaml have SC semantics

Data races

- ParaFuzz currently assumes that the programs are data-race-free (DRF)
  - ✦ DRF programs in OCaml have SC semantics
- OCaml memory model (PLDI'18) also has a simple operational model for racy programs
  - ✦ Racy reads may return one of a subset of writes performed to a non-atomic location

Data races

- ParaFuzz currently assumes that the programs are data-race-free (DRF)
  - ✦ DRF programs in OCaml have SC semantics
- OCaml memory model (PLDI'18) also has a simple operational model for racy programs
  - ✦ Racy reads may return one of a subset of writes performed to a non-atomic location
- Extend ParaFuzz to racy programs
  - ✦ Use AFL to pick the value that a read should return
  - ✦ Force a yield at non-atomic reads and writes

Data races

- ParaFuzz currently assumes that the programs are data-race-free (DRF)
  - ✦ DRF programs in OCaml have SC semantics
- OCaml memory model (PLDI'18) also has a simple operational model for racy programs
  - ✦ Racy reads may return one of a subset of writes performed to a non-atomic location
- Extend ParaFuzz to racy programs
  - ✦ Use AFL to pick the value that a read should return
  - ✦ Force a yield at non-atomic reads and writes
- *Can we make it fast?*