# Improving precision of data race verification via partial model checking

~~RaceHarness: Environment Modelling for Sound Static Race Detection~~

Jevgenijs Protopopovs[1], Danel Ahman[1], Patrick Lam[2], Vesal Vojdani[1]

[1]University of Tartu, Estonia

[2]University of Waterloo, Canada

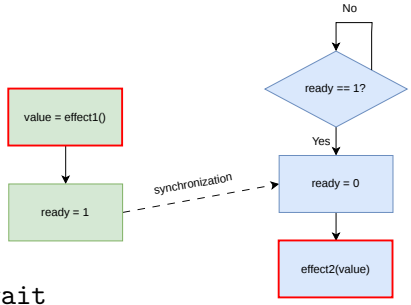
April 25, 2026

# The issue: value-based synchronization...

```
_Atomic _Bool ready = false;
int value = -1;

void thread1(void) {
  value = effect1();
  atomic_store(&ready, true);
}

void thread2(void) {
  while (!atomic_compare_exchange_strong(
    &ready, &(_Bool){true}, false)) {} // Busy wait
  effect2(value);
}
```

# The issue: value-based synchronization...

```
_Atomic _Bool ready = false;
int value = -1;

void thread1(void) {
  value = effect1();
  atomic_store(&ready, true);
}

void thread2(void) {
  while (!atomic_compare_exchange_strong(
    &ready, &(_Bool){true}, false)) {} // Busy wait
  effect2(value);
}
```



# The issue: value-based synchronization...

**Analyzer verdict:**

```
[Warning][Race] Memory location value (race with conf. 110): (example.c:6:5-6:15)
  write with [region:{value}, thread:[main, thread1@example.c:32:2-32:43]] (conf. 110)  (exp: & value) (example.c:16:3-16:20)
  read with [region:{value}, thread:[main, thread2@example.c:33:2-33:43]] (conf. 110)  (exp: & value) (example.c:25:3-25:17)
[Info][Race] Memory locations race summary:
  safe: 0
  vulnerable: 0
  unsafe: 1
  total memory locations: 1
```

Goblint: False positive!

```
[racer] Data race (must) on value at offset {[0..31]}
  :
      In thread thread2:66:

              read at examplecrv5bitj.pp8mky4r7m.over-approximation.c:55

      In thread thread1:65:

              write at examplecrv5bitj.pp8mky4r7m.over-approximation.c:34
[cc] Statistics:
      - #threads: 3
      - #edges: 0
```

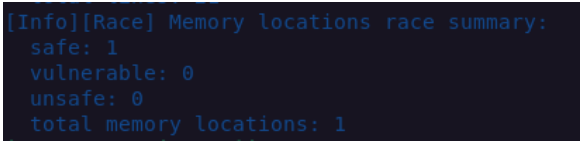
RacerF: False positive!

# The end goal: mutual exclusion!

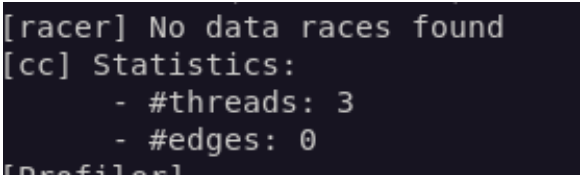
```
void thread1(void) {
  pthread_mutex_lock(&mutex1);
  pthread_barrier_wait(&init_barrier);
  value = effect1();
  pthread_mutex_unlock(&mutex1);
}

void thread2(void) {
  pthread_barrier_wait(&init_barrier);
  pthread_mutex_lock(&mutex1);
  effect2(value);
  pthread_mutex_unlock(&mutex1);
}
```

# The end goal: mutual exclusion!

```
void thread1(void) {
  pthread_mutex_lock(&mutex1);
  pthread_barrier_wait(&init_barrier);
  value = effect1();
  pthread_mutex_unlock(&mutex1);
}

void thread2(void) {
  pthread_barrier_wait(&init_barrier);
  pthread_mutex_lock(&mutex1);
  effect2(value);
  pthread_mutex_unlock(&mutex1);
}
```




Goblint is happy




RacerF too (with a slightly different barrier)

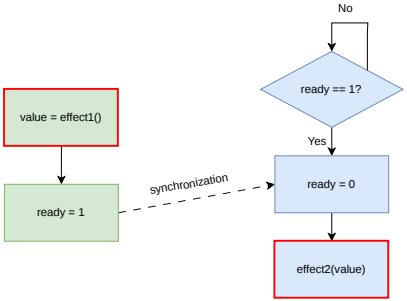
# Pipeline: refinement via model checking

**Pipeline**

Program model $\rightarrow$ Model checking $\rightarrow$ Concurrency invariants $\rightarrow$ Harness synthesis $\rightarrow$ Analysis

# Pipeline: refinement via model checking

```
_Atomic _Bool ready = false;
int value = -1;

void thread1(void) {
  value = effect1();
  atomic_store(&ready, true);
}
void thread2(void) {
  while (!atomic_compare_exchange_strong(
    &ready, &(_Bool){true}, false)) {}
  effect2(value);
}
```



# Pipeline: refinement via model checking

```
_Atomic _Bool ready = false;
int value = -1;

void thread1(void) {
  value = effect1();
  atomic_store(&ready, true);
}
void thread2(void) {
  while (!atomic_compare_exchange_strong(
    &ready, &(_Bool){true}, false)) {}
  effect2(value);
}
```

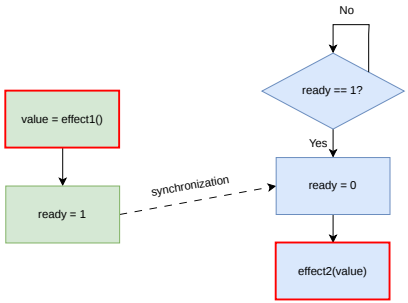
**Control locations:**

$T_1 = \{\text{value} = \text{effect1}(), \text{ready} = 1\}$

$T_2 = \{\text{ready} == 1?, \text{ready} = 0, \text{effect2}(\text{value})\}$

**Variables:**

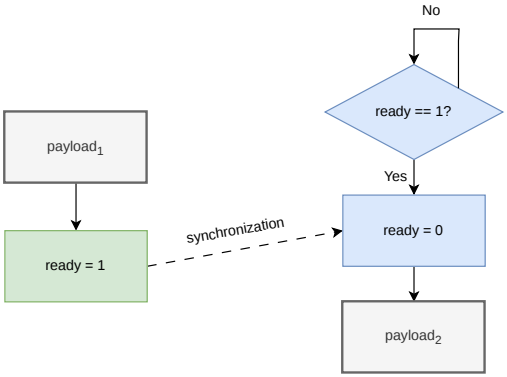
$\text{value} \in \mathbb{Z}, \text{ready} \in \mathbb{Z}$



# Pipeline: refinement via model checking

## Pipeline

Program model → Model checking → Concurrency invariants → Harness synthesis → Analysis




**Abstracted out:**

$\text{payload}_1 \mapsto$ `value = effect1()`

$\text{payload}_2 \mapsto$ `effect2(value)`

**Control locations:**

$T_1 = \{\text{payload}_1, \text{ready} = 1\}$

$T_2 = \{\text{ready} == 1?, \text{ready} = 0, \text{payload}_2\}$

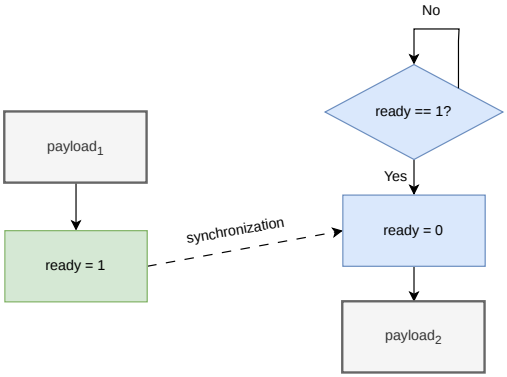
**Variables:**

$\text{value} \in \{\top\}, \text{ready} \in \mathbb{Z}$

# Pipeline: refinement via model checking

## Pipeline

Program model → Model checking → Concurrency invariants → Harness synthesis → Analysis



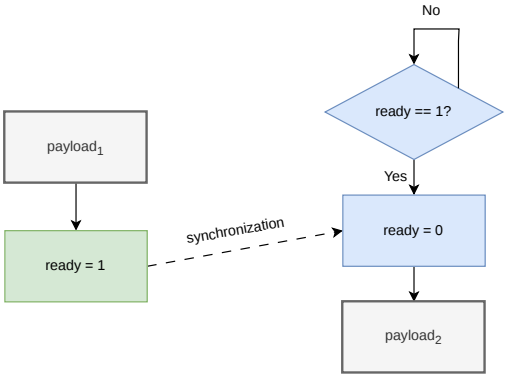

**May-Happen-in-Parallel:**

$(\text{payload}_1, \text{ready} == 1?)$
$(\text{ready} = 1, \text{ready} == 1?)$
$(\bot, \text{ready} == 1?)$
$(\bot, \text{ready} = 0)$
$(\bot, \text{payload}_2)$
$(\bot, \bot)$

# Pipeline: refinement via model checking

## Pipeline

Program model → Model checking → Concurrency invariants → Harness synthesis → Analysis




**Mutually exclusive:**

$(\text{payload}_1, \text{ready} = 0)$

$(\text{payload}_1, \text{payload}_2)$ ⟵ **!!!**

$(\text{payload}_1, \bot)$

$(\text{ready} = 1, \text{ready} = 0)$

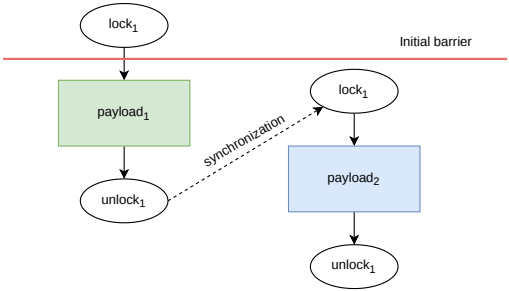
$(\text{ready} = 1, \text{payload}_2)$

$(\text{ready} = 1, \bot)$

# Pipeline: refinement via model checking

## Pipeline

Program model → Model checking → Concurrency invariants → Harness synthesis → Analysis

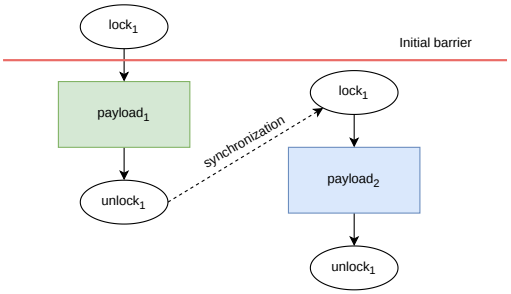



**Harness** = Model control flow +
Program payload bodies +
Model checker invariants

# Pipeline: refinement via model checking

## Pipeline

Program model → Model checking → Concurrency invariants → Harness synthesis → Analysis




```
void thread1(void) {
  pthread_mutex_lock(&mutex1);
  pthread_barrier_wait(&init_barrier);
  value = effect1();
  pthread_mutex_unlock(&mutex1);
}
void thread2(void) {
  pthread_barrier_wait(&init_barrier);
  pthread_mutex_lock(&mutex1);
  effect2(value);
  pthread_mutex_unlock(&mutex1);
}
```

# Pipeline: refinement via model checking

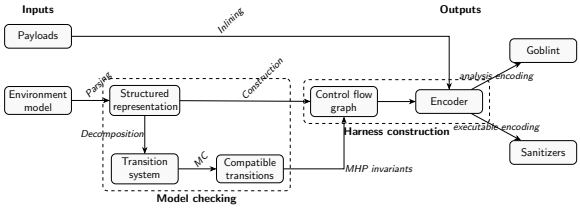
**Pipeline**

Program model → Model checking → Concurrency invariants → Harness synthesis → Analysis


Abstract interpretation: Goblint

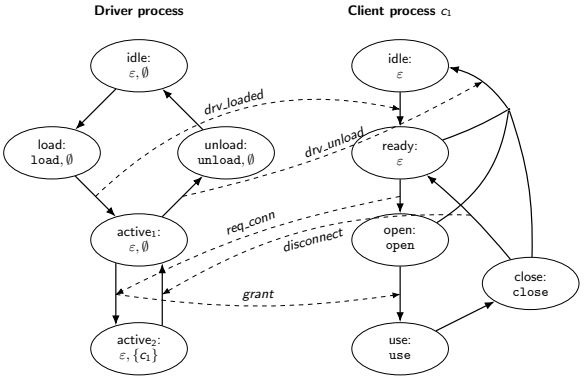

Fuzzing: gcc/clang sanitizers

# Case study: Linux kernel




- Modular verification
- Hand-crafted models and stubs
- Kernel-module protocol as model boundary
- Message passing as synchronization primitive

# Case study: Linux kernel




- Modular verification
- Hand-crafted models and stubs
- Kernel-module protocol as model boundary
- Message passing as synchronization primitive

# Case study: Results

| Module | Accesses | False positives | Baseline |
|---|---|---|---|
| **ttyprintk** | 24 | 0 | 2 |
| amd-rng | 14 | 0 | 1 |
| intel-rng | 18 | 0 | 1 |
| lp | 42 | 11 | 16 |
| **nvram** | 27 | 0 | *n/t* |
| ppdev | 69 | 26 | 28 |
| sun6i-msgbox | 31 | 2 | 3 |
| ttynull | 19 | 0 | 2 |
| adxl34x-i2c | 12 | 0 | 1 |
| efivarfs | 5 | 0 | *n/a* |
| **pcspkr** | 17 | 0 | *n/a* |
| xen-balloon | 12 | 0 | *n/a* |
| random | 28 | 6 | *n/a* |
| **vt-selection** | 16 | 0 | *n/a* |

- **Bold**—injected races preserved
- Accesses—analyzed memory accesses
- False positives—reported races with precise environment model.
- Baseline—reported races with basic environment model without synchronization

# Future work

- Automatic model learning from execution traces.
- Integrating model checker at C language level directly.
- Model validation via execution.

# Appendix: core language example

```
global model $eady := 0
global value := -1

l_1 = effect effect_1()
l_2 = store value, l_1
l_3 = const 1
l_4 = store ready, l_3
;;
```

```
l_5 = const 0
l_6 = const 1
l_7 = cmpxchg ready, l_6, l_5
l_8 = l_7 != l_6
l_9 = if l_8 then goto l_7
l_10 = load value
l_11 = effect effect_2(l_10)
;;
```

# Appendix: DSL example

```
channel cltmsg { req_conn, disconnect }
channel drvmsg { grant, drv_loaded, drv_unload }

proc pclient in drvmsg out cltmsg {
    alive: loop /* idle */
        if recv drv_loaded then { /* ready */
        send req_conn driver
        if recv drv_unload then continue alive
        else if recv grant then {
            do acquire_conn /* open */
            use: loop if nondet then do use_conn else break use /* use */
            do release_conn /* close */
            send disconnect driver } } }
run clients[2] pclient
```