RaceSanitizer: Sampling for Data Races

Peter Goodman Angela Demke Brown Ashvin Goel
Trail of Bits, University of Toronto

How do data races happen?

- Two threads operate on the same data simultaneously
- At least one thread writes to memory
- Neither memory access is atomic

Why are data races bad?

- Introduce non-determinism and undefined behavior
- Hide behind unexercised thread interleavings
- Decrease portability (weak vs. strong memory models)

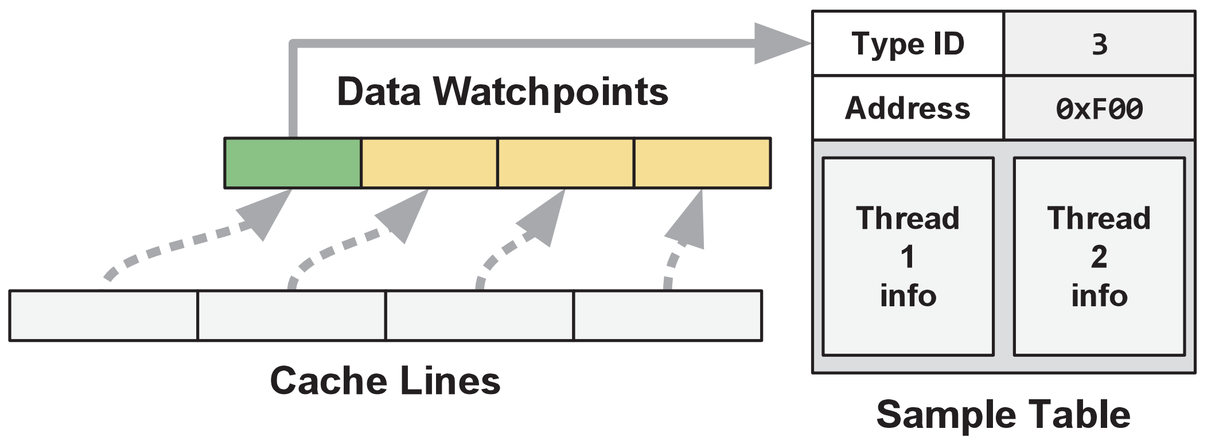
RaceSanitizer comprehensively samples all memory accesses

Goal: Instrument all memory accesses to look for data races

Key Idea: Use **infinite data watchpoints** to track which threads access what memory

Challenge: Tracking every memory access is inefficient

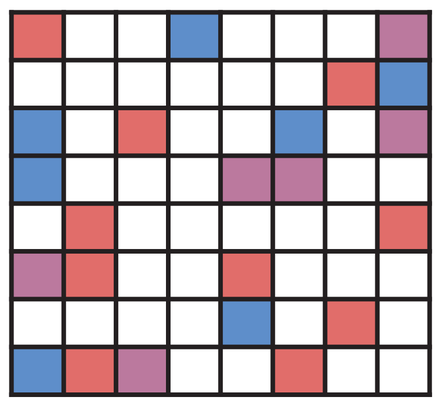
Solution: Instrument all memory accesses that touch a chosen **sample** of the entire memory space

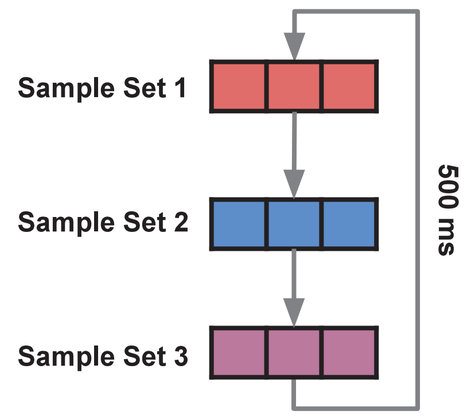


RaceSanitizer analyzes representative samples of memory

Challenges:
- Access patterns: "hot" data accessed very often
- Non-uniformity: more objects of one type than of another

Solutions:
- Sample from recently allocated objects
- Group candidate objects by type
- Sample one group of same-typed objects at a time





RaceSanitizer waits for data races to happen then catches them in the act

Thread 1: Reads and writes to the shared variable X

```
if (is_data_watched(&x)) {
  prev_count = increment_access_count(&x);
  if (!prev_count) {
    record(&x, thread_info);
    sleep(10ms);
    remove_watchpoint(&x);
  } else if (1 == prev_count &&
             is_data_watched(&x)) {
    report(&x, thread_info);
  }
}
x++;
```

Thread 2: Concurrently writes to the shared variable X while Thread 1 is sleeping

```
if (is_data_watched(&x)) {
  prev_count = increment_access_count(&x);
  if (!prev_count) {
    record(&x, thread_info);
    sleep(10ms);
    remove_watchpoint(&x);
  } else if (1 == prev_count &&
             is_data_watched(&x)) {
    report(&x, thread_info);
  }
}
x = 1;
```

```
Data race in ranges [0x7fc3aebee820, 0x7fc3aebee828) of object of size 2140504 bytes allocated at 0x7fc3aebec010
(bytes 10256 to 10264)
   Allocated by:
      /home/pag/Code/parsec-3.0/ext/splash2x/apps/volrend/obj/amd64-linux.rsan/main.C:135
         /home/pag/Code/parsec-3.0/ext/splash2x/apps/volrend/inst/amd64-linux.rsan/bin/volrend:0x405986
   Write of 8 bytes to 0x7fc3aebee820 at:
      /home/pag/Code/parsec-3.0/ext/splash2x/apps/volrend/obj/amd64-linux.rsan/adaptive.C:351
```

Other features of RaceSanitizer

- Low overhead: up to 30% (future work: optimization)
- No false-positives
- False-negatives if some code doesn't use `-fsanitize=race`

Where is the code?

You can find RaceSanitizer on the Trail of Bits GitHub page at:

https://github.com/trailofbits/RaceSanitizer