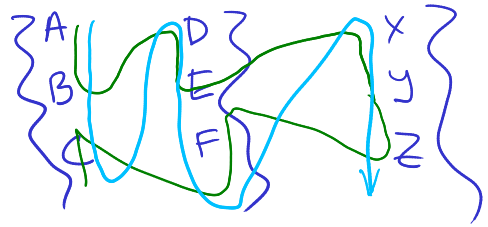
Finding Concurrency bugs

Subtle bugs.

Testing is difficult → coverage
                     → reproducing

TSVD: effective
      ideas/techniques

Key problem: interleavings



A B D E X Y Z F C

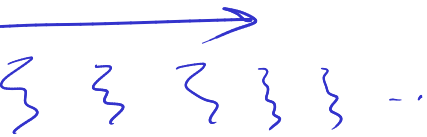
Static analysis

Non-local
Inter-procedural
Spec for function
Hard for concurrency to Infer.

Runtime testing

1.) Define a bug.
    Wrong result.
    Crash
    Invariant
2.) Explore schedules.
    Core non-determinism.

Strawman 1: run many times



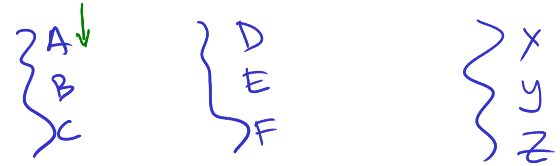
A B D E F C

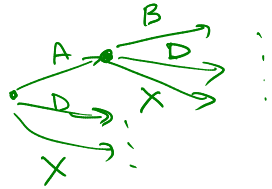
A B C D E F

Linux kernel
"lock torture"
"RCU torture".

Hard to explore diff. schedules.
Hard to reproduce.

Strawman 2: exhaustive





Good: coverage?
Bad: not practical, too many, too similar.
Good: repeatable.

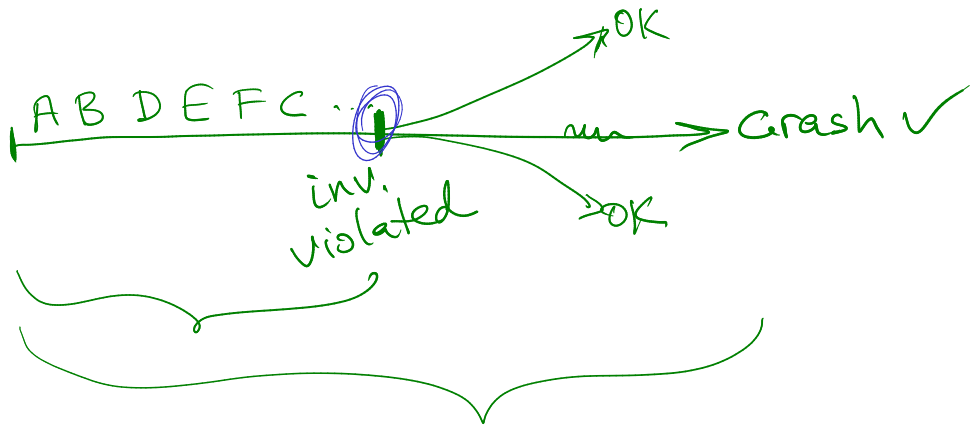
Research on exhaustive checking.

"Preemption points" → lock acquire
→ shared mem op.

Few preemptions to needed.

Optimizations to avoid equiv. schedules.

Approach: concurrency invariants



+ Fewer runs.
+ No need for exact race
- False positive

HB hard to get right.

Invariant: lock sets ←

Eraser
Inv: any shared mem. loc. accessed while holding some lock.

Invariant: data races

Inv: no concurrent r/w to same mem. loc.

Invariant: happens-before checking

Track dependencies between threads.
Inv: no un-ordered r/w to same addr.

% go test -race.
Clang TSan.

TSVD

Q: What is a bug (TSV)?
False pos? Missed bugs?

TSV invariant

C# data structures
- Dictionary ⇐
- List
- Queue
- ArrayList...

Read set
Write set.
Invariant: nothing concur. w/ write op.

Data race inv.

One mem. loc. ≤64
ATOMIC IN HW
≤128.

Read ∈ R set
Write ∈ W set.

Inv: nothing concur. w/ write op.

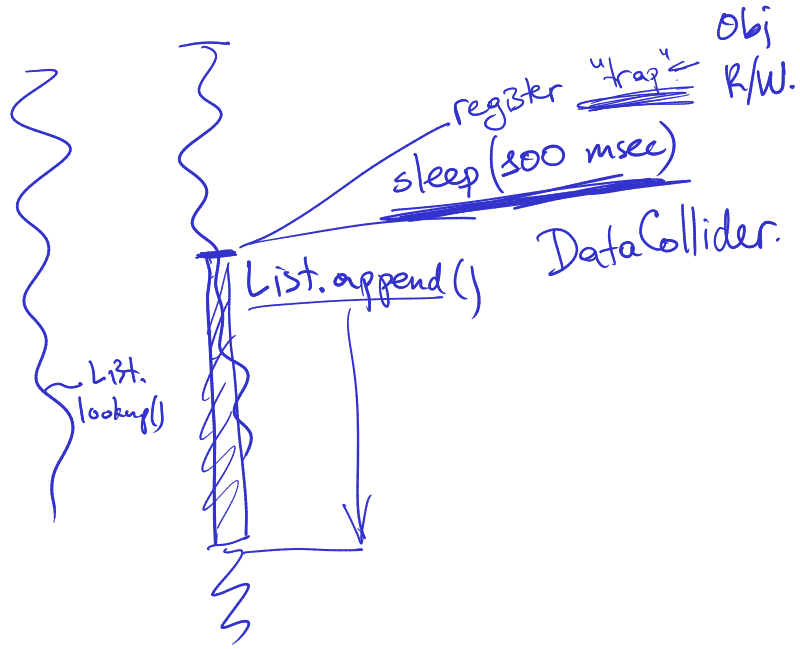
Why low FP?

Many benign data races. ⇐
~ No benign TSV races.

FP?

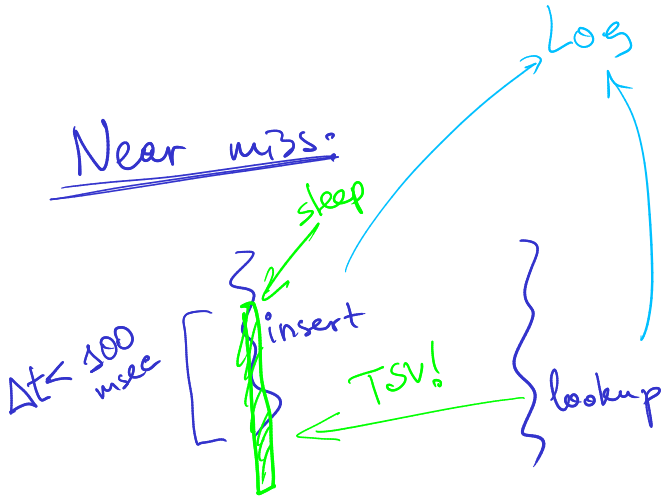
Pair (R,W) that have no crash/issue.

Scheduling



When to inject sleep?

Near misses:



Infer happens-before