Gautam Altekar and Ion Stoica
University of California, Berkeley

ODR: Output-Deterministic Replay for Multicore Debugging

Debugging Is Hard

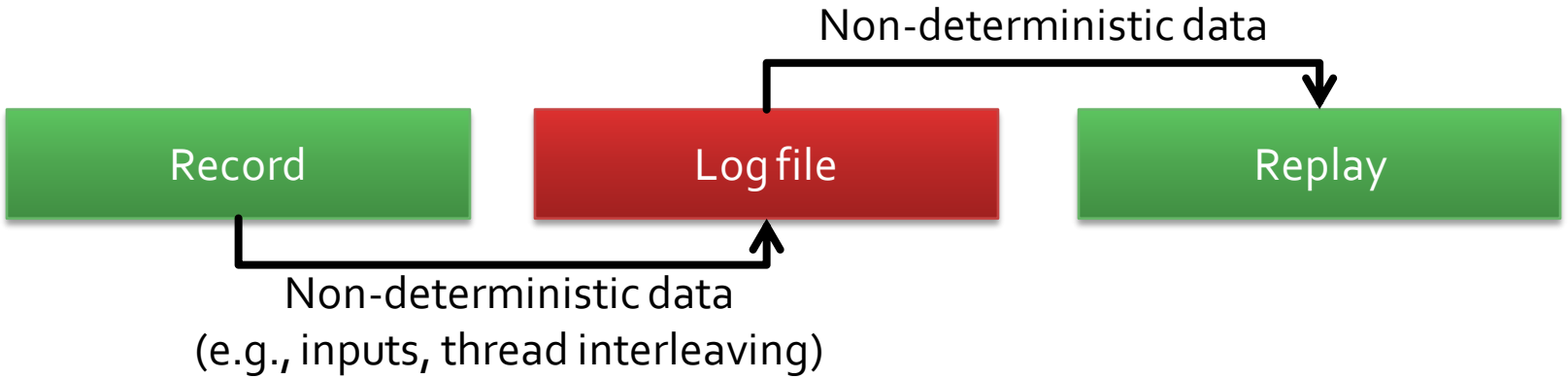
Debugging non-deterministic software failures is *really* hard

- Hard to reproduce
  - Shows up in production, but not in development
- Precludes cyclic debugging
  - Can't rerun to home-in on root cause

How can we reproduce non-deterministic failures?

Deterministic Replay Systems

Generate replica of original run, hence failures



Why deterministic replay?

- Model checking, testing, verification
  - Goal: find errors pre-production
  - Can't catch all errors
  - Can't reproduce production failures

Requirements

- Support multiprocessor operation
  - Multi-core revolution
- Support efficient recording
  - Suitable for production use
- Require no special hardware
  - Can be used today on many architectures
- Replay data races without annotation
  - Found in many apps, benign or erroneous

Related Work

	Multiple CPUs	Efficient recording	Software -only	Replays data races
Liblog,R2	**No**	Yes	Yes	Yes
SMP-ReVirt,iDNA	Yes	**No**	Yes	Yes
CapoOne,DMP	Yes	Yes	**No**	Yes
RecPlay,Kendo	Yes	Yes	Yes	**No**

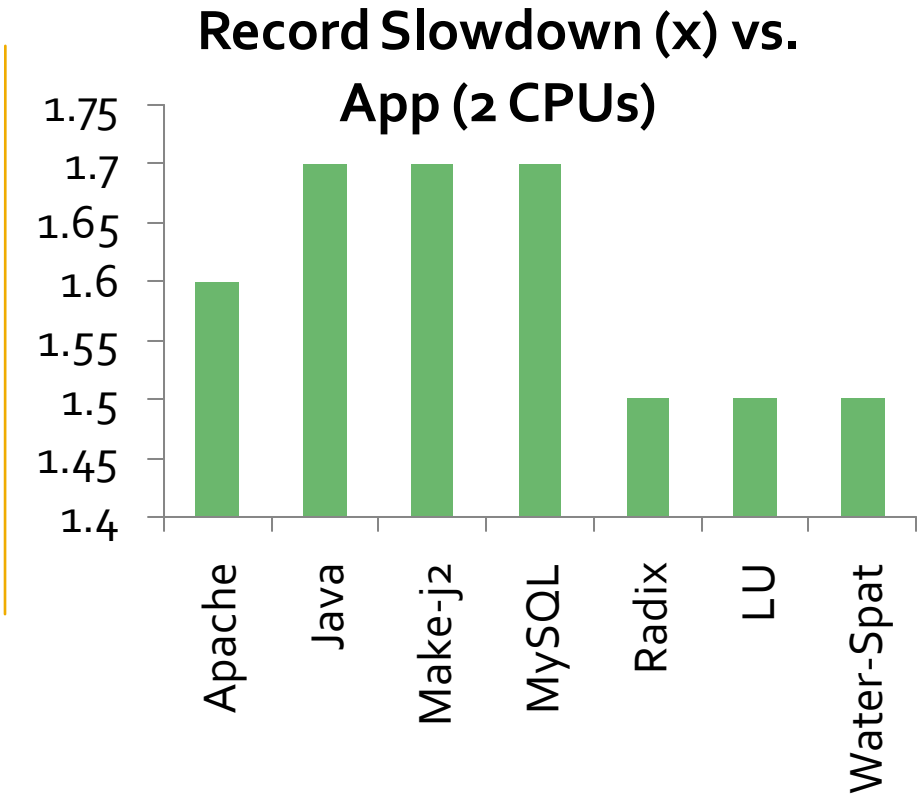
Very hard to meet all requirements

Contribution

We've built a *user-level* replay system (ODR)

Meets all requirements

- Multiple CPUs
- Efficient recording ~1.6x
- Software-only
- Replays real Linux/x86 apps
  - Apache, Java-Tomcat, MySQL, Radix, etc.



Intuition

***For debugging*, not necessary to produce identical run**

Often suffices to produce *any* run that has the *same outputs*

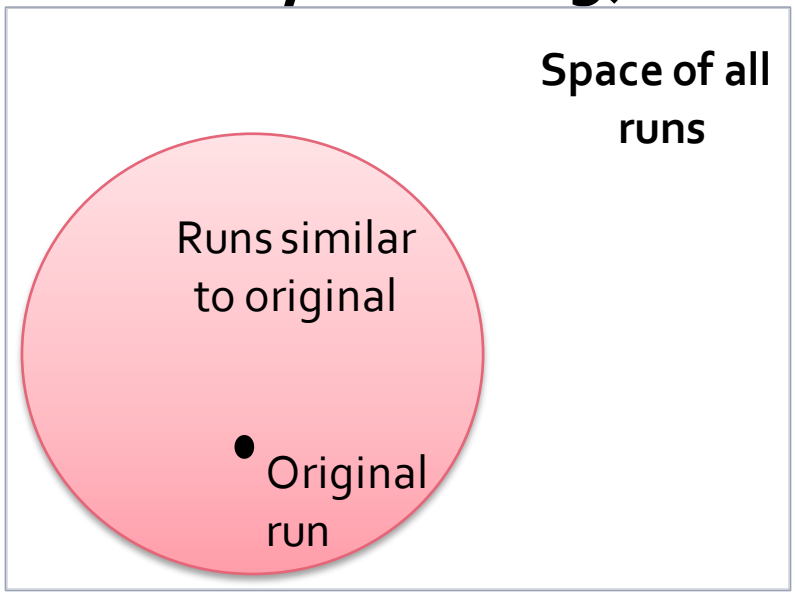
Outline

- ✓ Overview
- **Leveraging the intuition**
- Achieving output-determinism
- Results
- Future work

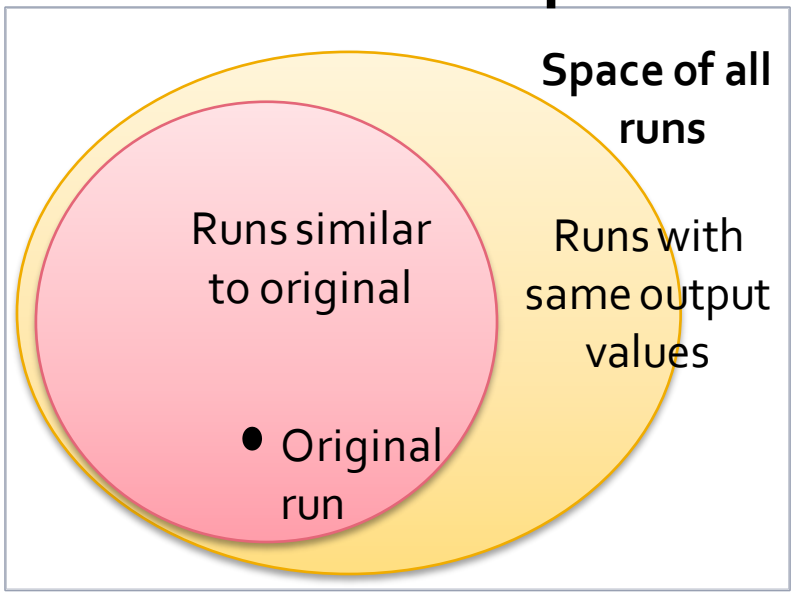
Approach and Related Work

Relax the determinism requirements

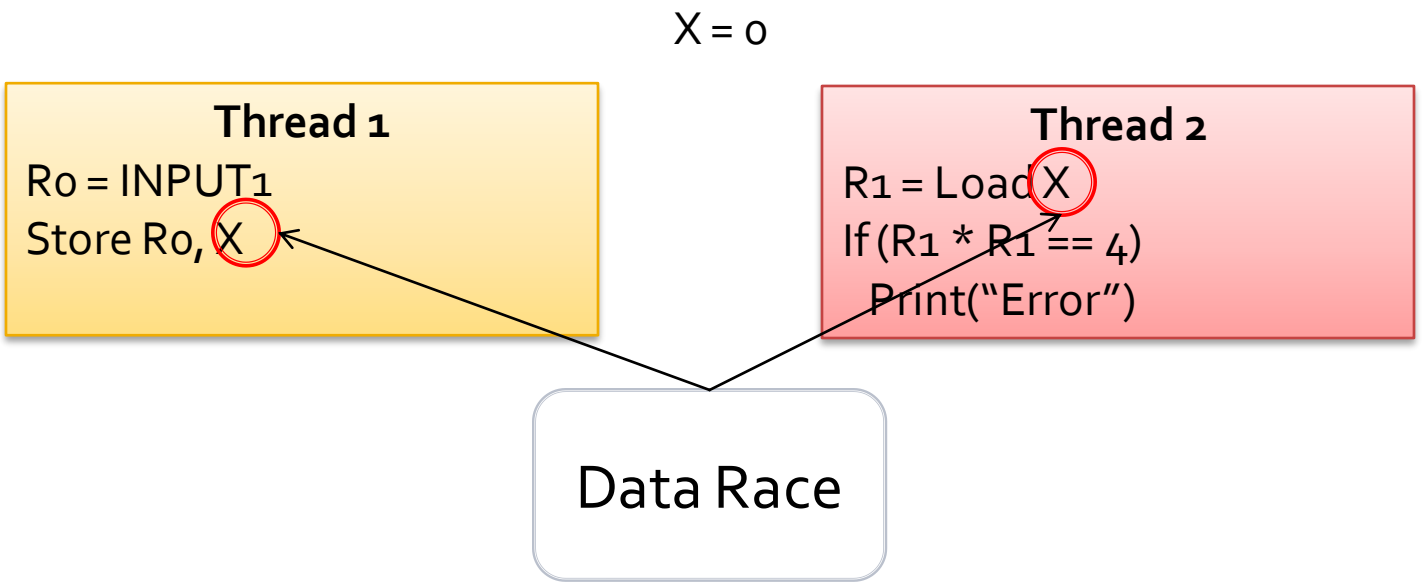
Not a new idea (e.g., Instant Replay '86, Stone '88, PRES '09)



ODR takes it to extreme; replay may differ a lot, but same outputs



An Example


X = 0
Thread 1
R0 = INPUT1
Store R0, X
Thread 2
R1 = Load X
If (R1 * R1 == 4)
Print("Error")
Data Race

Classic Guarantee: Value-Determinism

X = 0

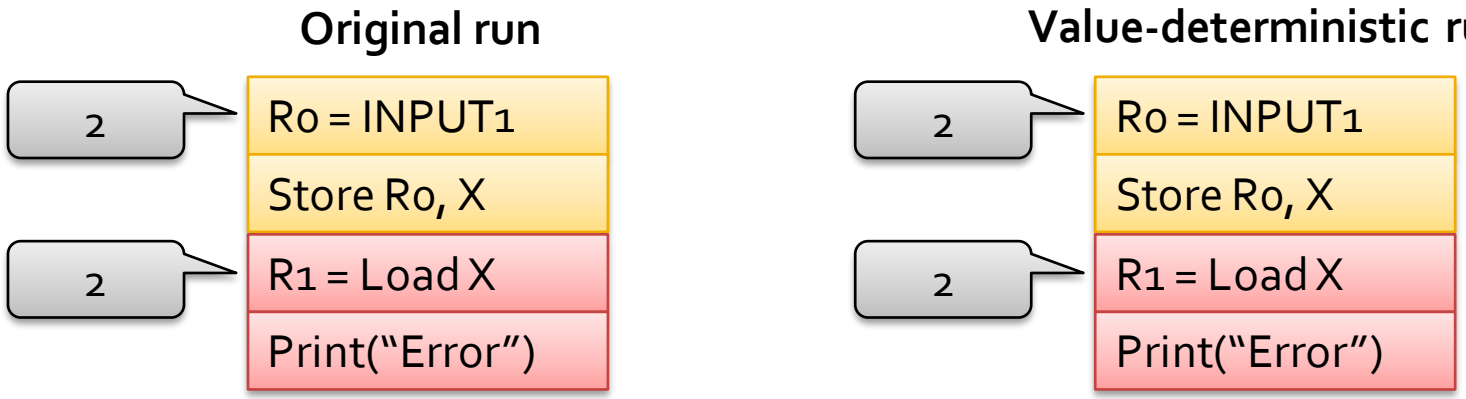
Thread 1

```
R0 = INPUT1
Store R0, X
```

Thread 2

```
R1 = Load X
If (R1 * R1 == 4)
  Print("Error")
```

Replay run reads/writes same values as original



Our Approach: Output-Determinism

X = 0

Thread 1

```
R0 = INPUT1
Store R0, X
```

Thread 2

```
R1 = Load X
If (R1 * R1 == 4)
  Print("Error")
```

Replay run produces the same output as original (Output: segfaults, core dumps, console messages, etc.)

Original output
Error

Output-deterministic output
Error

Output-Determinism Isn't Perfect

X = 0

Thread 1

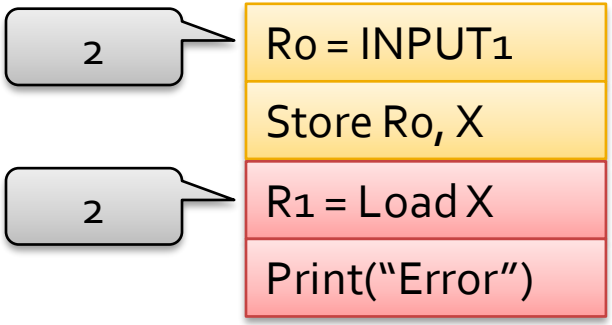
```
R0 = INPUT1
Store R0, X
```

Thread 2

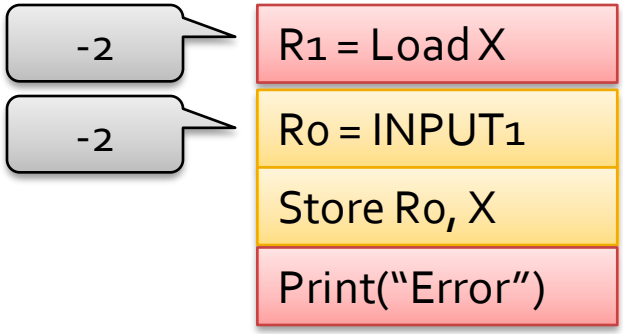
```
R1 = Load X
If (R1 * R1 == 4)
  Print("Error")
```

Ordering nor values are reproduced

Original run



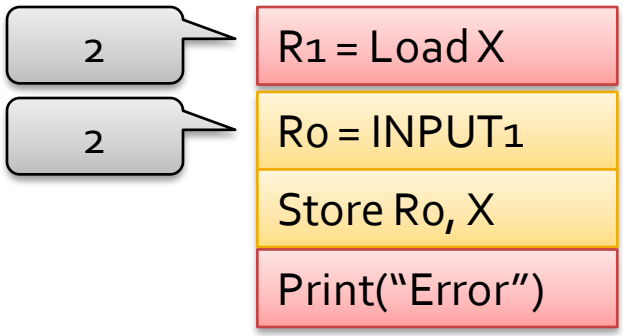
Output-deterministic run



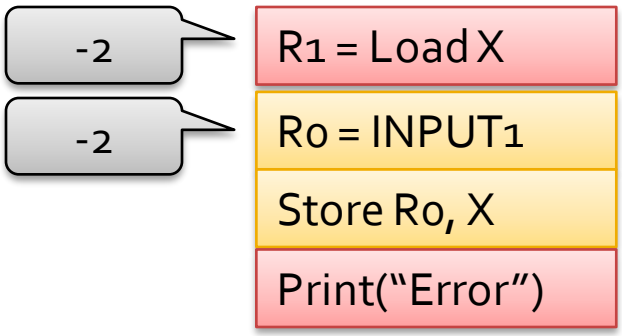
Output-Determinism Is Useful For Debugging

1. Reproduces most visible signs of failures
   - By way of reproducing all outputs
2. Provides variable values that are consistent with that failure
   - Enables reasoning about failure's root cause

Value-deterministic total-ordering:



Output-deterministic total-ordering:



The Benefit of Output-Determinism

If we shoot for output-determinism, values don't have to be the same; so no need to record them

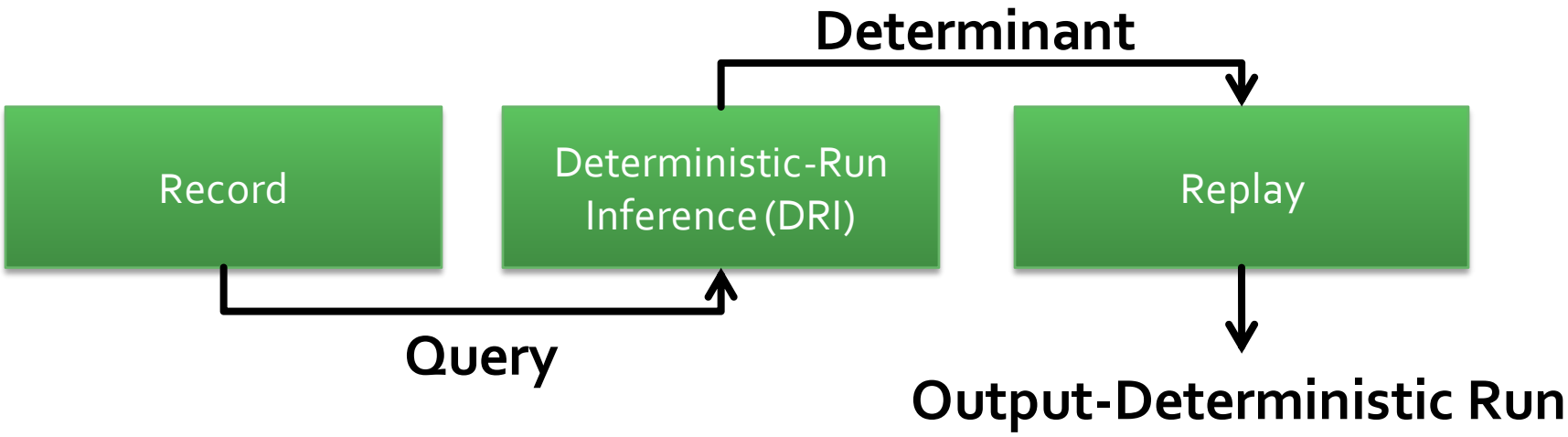
	Multiple CPUs	Efficient recording	Software-only	Replays data races	Determinism
Liblog, R2	**No**	Yes	Yes	Yes	Value
SMP-ReVirt, iDNA	Yes	**No**	Yes	Yes	Value
CapoOne	Yes	Yes	**No**	Yes	Value
Kendo, RecPlay	Yes	Yes	Yes	**No**	Value
ODR	**Yes**	**Yes**	**Yes**	**Yes**	**Output**

Outline

- ✓ Overview
- ✓ Leveraging the intuition
- ➢ **Achieving output-determinism**
- ➢ Results
- ➢ Future work

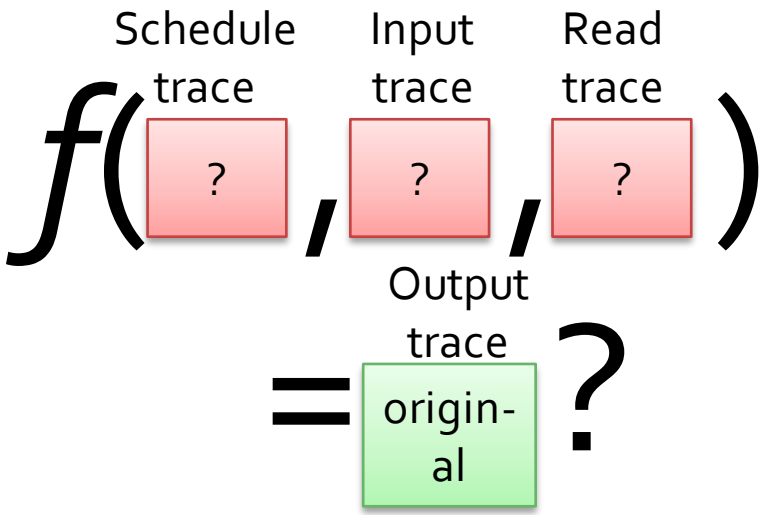
Deterministic-Run Inference (DRI)

- Query: information about original run
  - Minimally, original output
- Result: determinant
  - Includes all non-deterministic data: e.g., thread interleaving , input values, read values

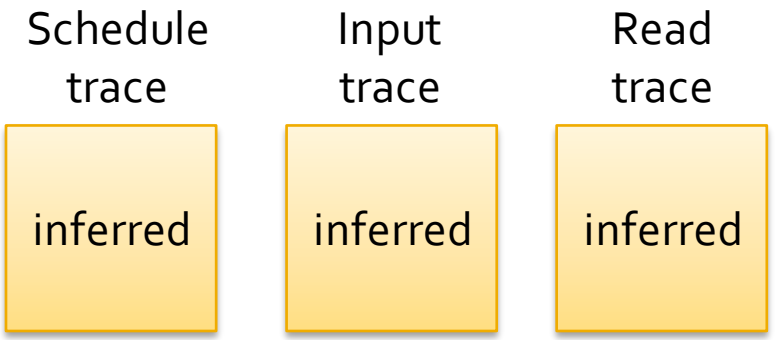


Inference: The Basic Idea

1. *Translate program into a logical formula* (Verification Condition, Floyd '67)



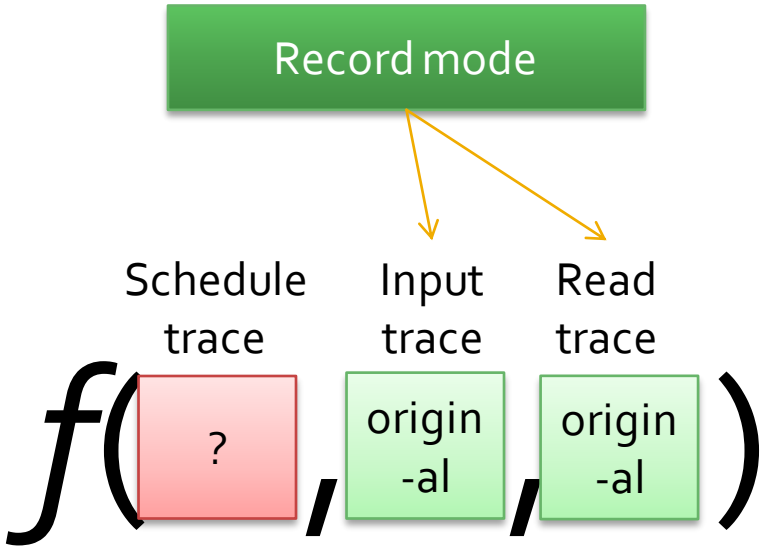
2. *Solve for the unknowns* using a formula solver (we use STP, Ganesh '07)



How Do We Scale?

1. *Direct the inference* using more original run info

An Extreme Example



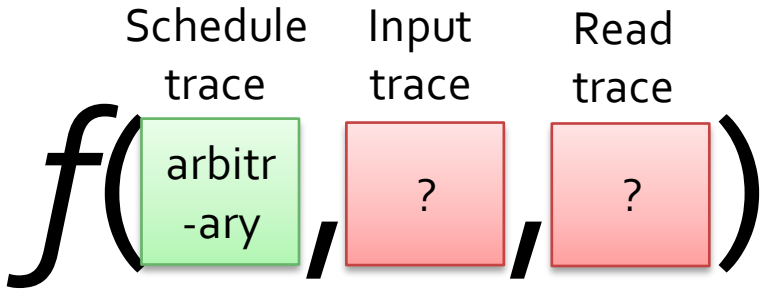
Reduces the number of unknowns

2. *Relax memory consistency* of inferred run

An Extreme Example: Null Consistency

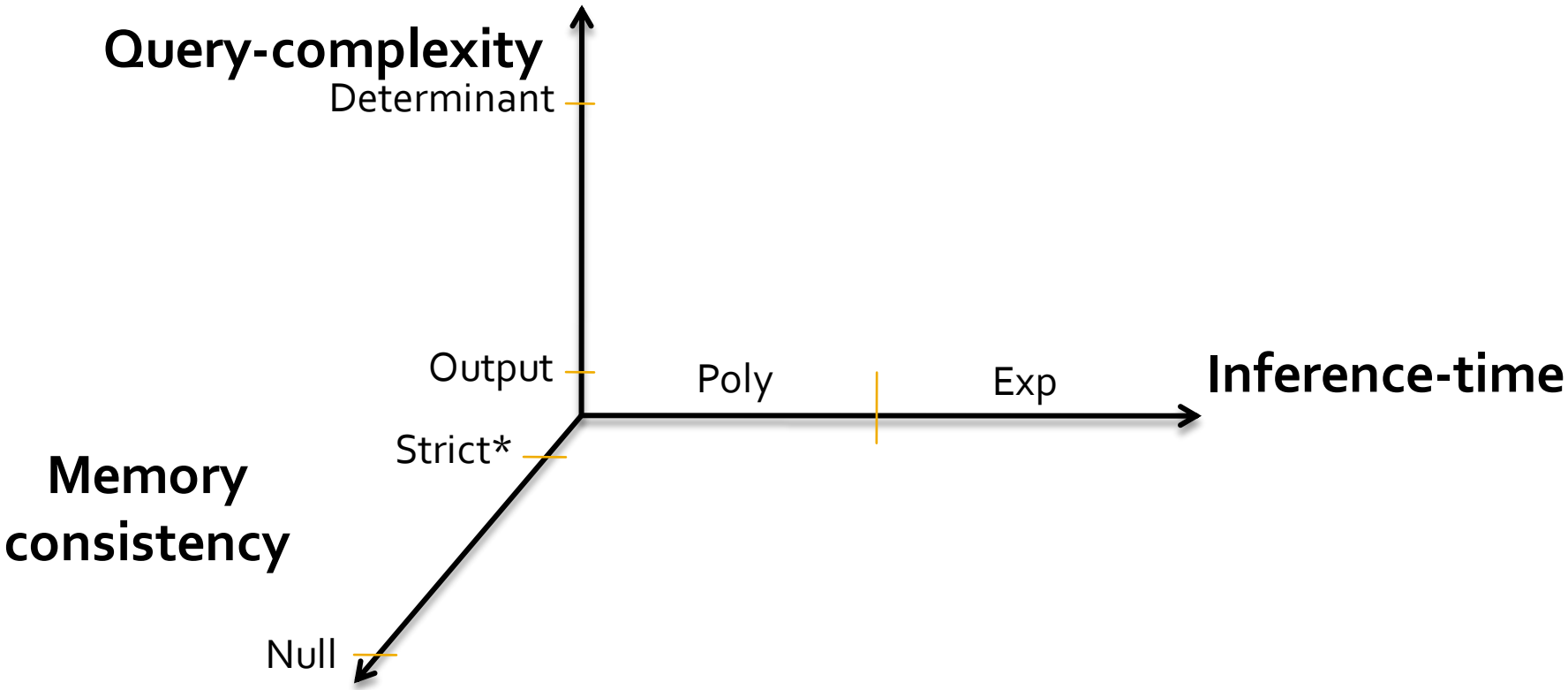
x = 0
T1: x = 1
T2: r0 = x

R0 = 5 – completely arbitrary!



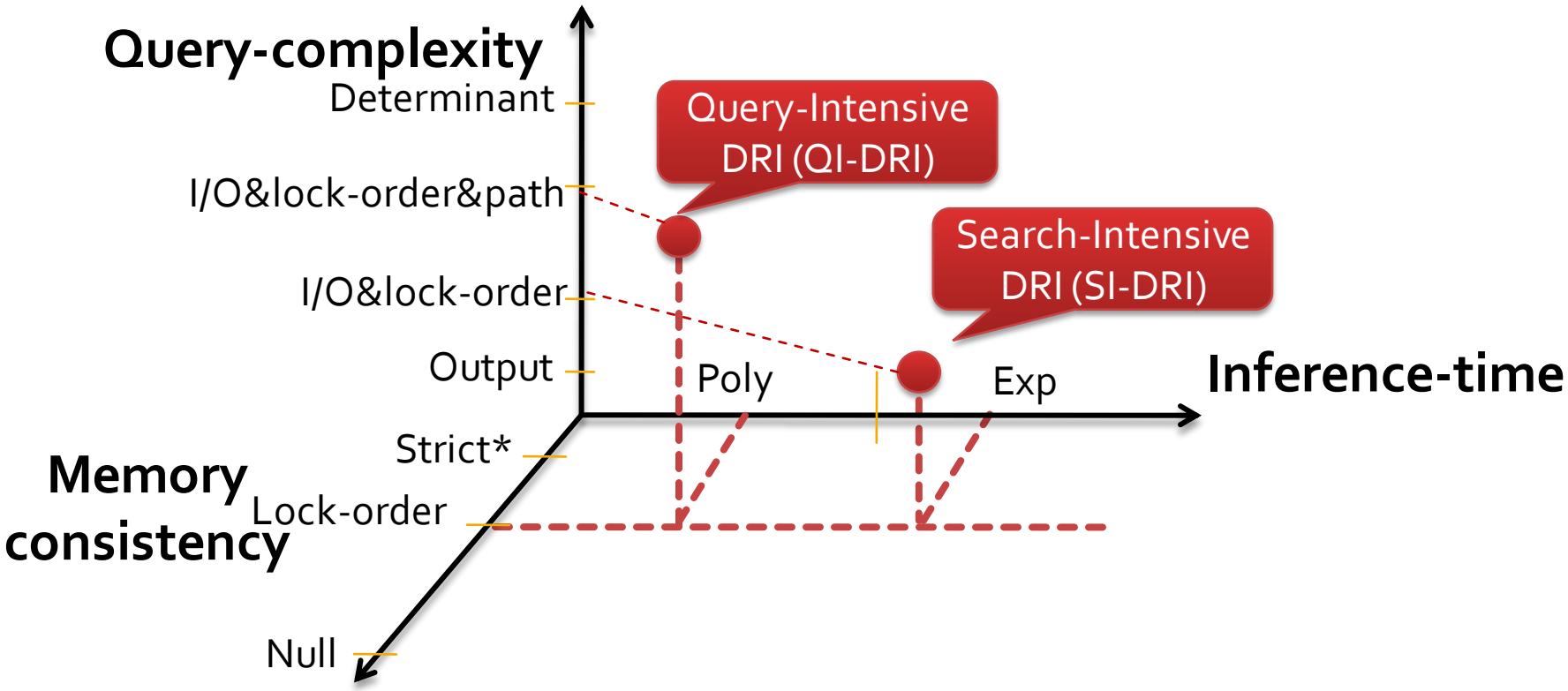
Choice of schedule-trace makes no difference

Inference Design Space



*Assumes original run is strict-consistent (*w.l.g*)

Inference Design Targets

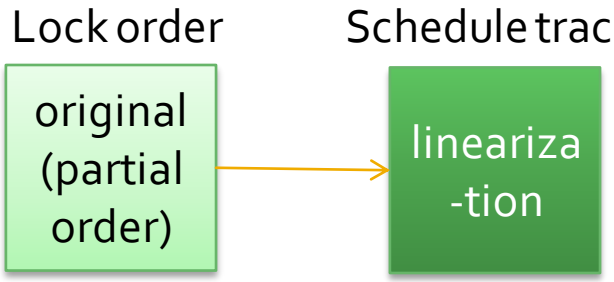


*Assumes original run is strict-consistent (*w.l.g*)

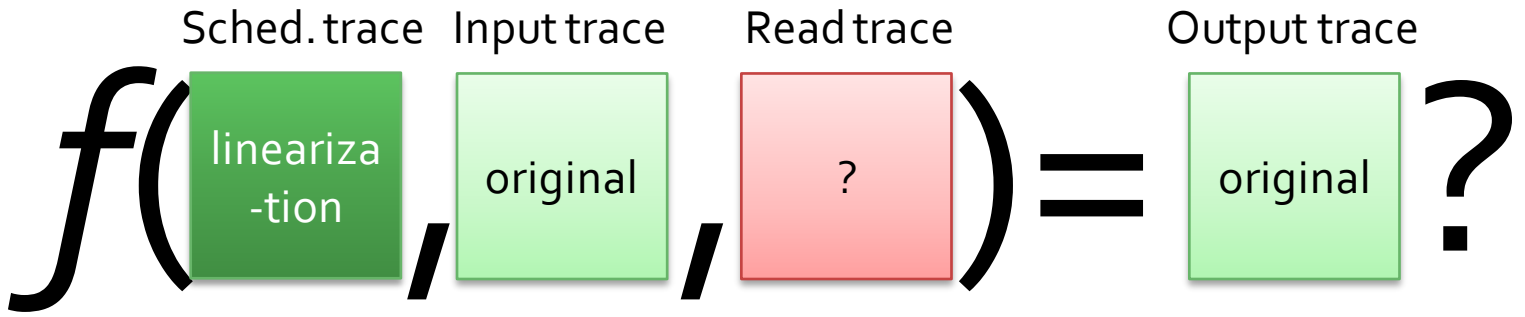
SI-DRI: The Basic Idea

Query: the original output, *input* trace and *lock-order*

1. Substitute original input trace into formula



2. Select total order of original lock-order, then substitute



QI-DRI: The Basic Idea

Query: like SI-DRI, but also *requires original path*

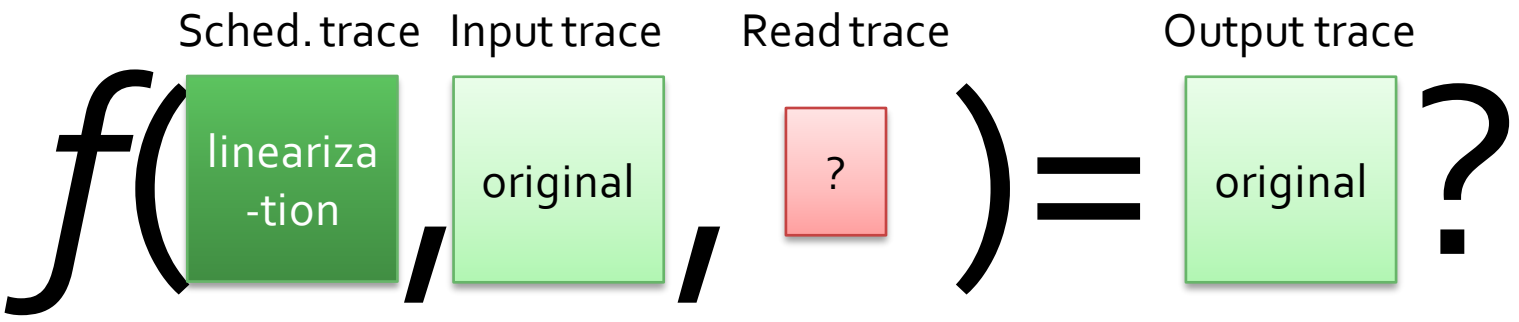
Input trace: original

Sched. trace: linearization

1. Substitute into formula (like SI-DRI)

Path trace: original

2. Used to narrow space of read traces

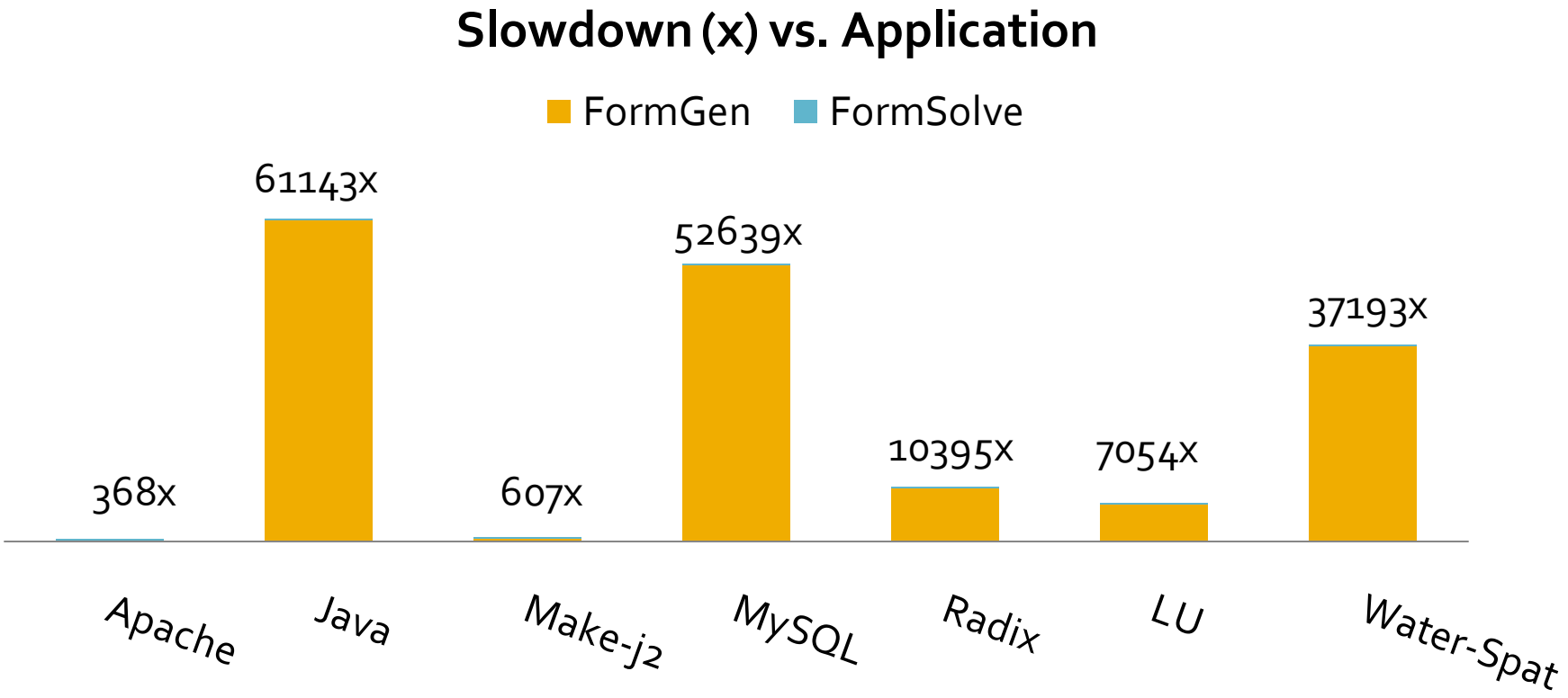


Outline

- ✓ Overview
- ✓ Leveraging the intuition
- ✓ Achieving output-determinism
- ➢ **Results**
- ➢ Future work

SI-DRI: Inference Time

It's Really Slow ~ 400x to 60,000x for 1 sec run



Key points: (1) Formula generation, not solving, is the bottleneck and (2) High variance

Formula Generation Time Explained

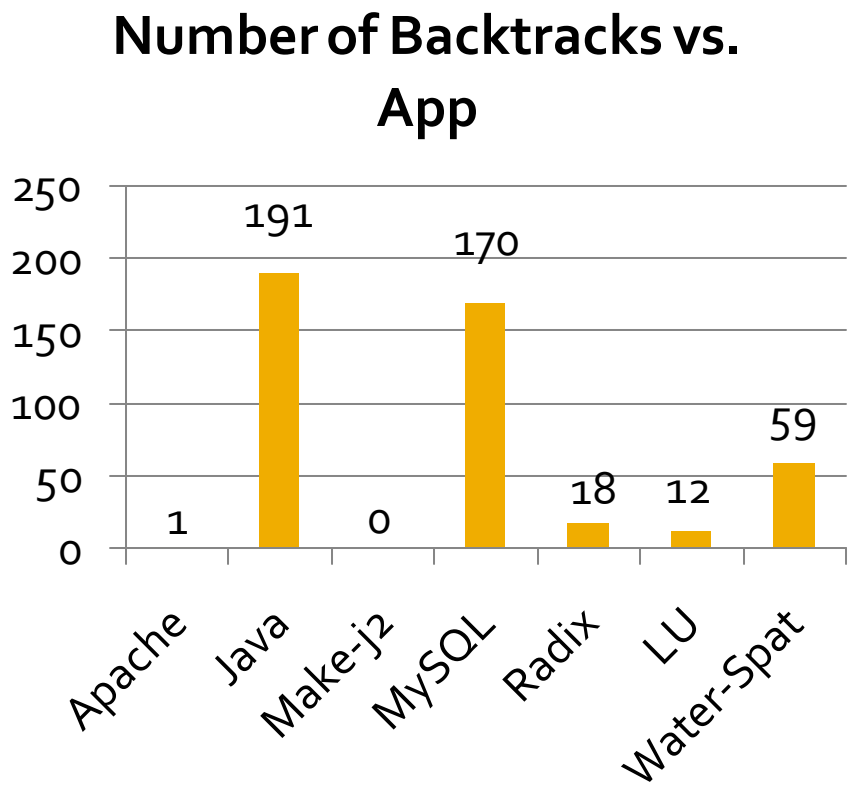
Key detail: multi-path *symbolic execution* (King '76) w/ backtracking

Why is it high?

- Large number of backtracks, each is costly (>= 200x)

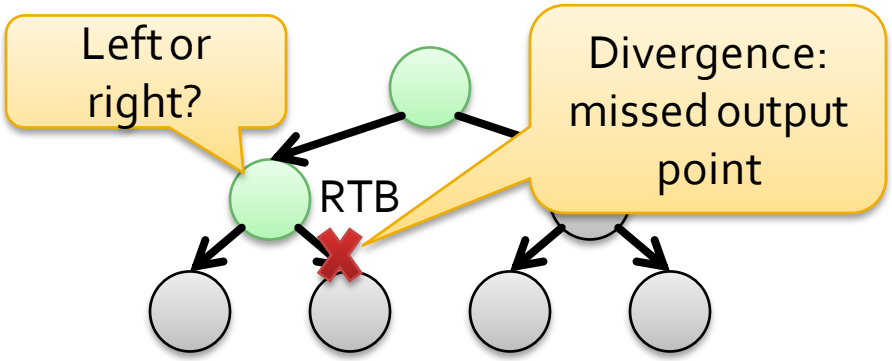
Why the high variance?

- Some apps backtrack more than others



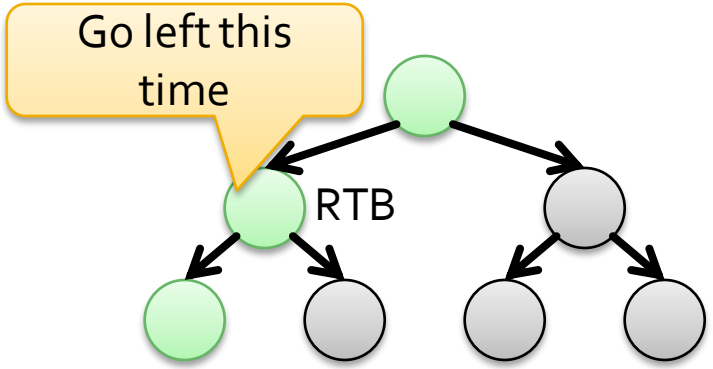
What Causes a Backtrack?

Reason: divergence induced by race-tainted branches (RTBs)



The wrong choice may result in divergent or UNSAT path

Workaround: Backtrack to most-recent RTB upon divergence

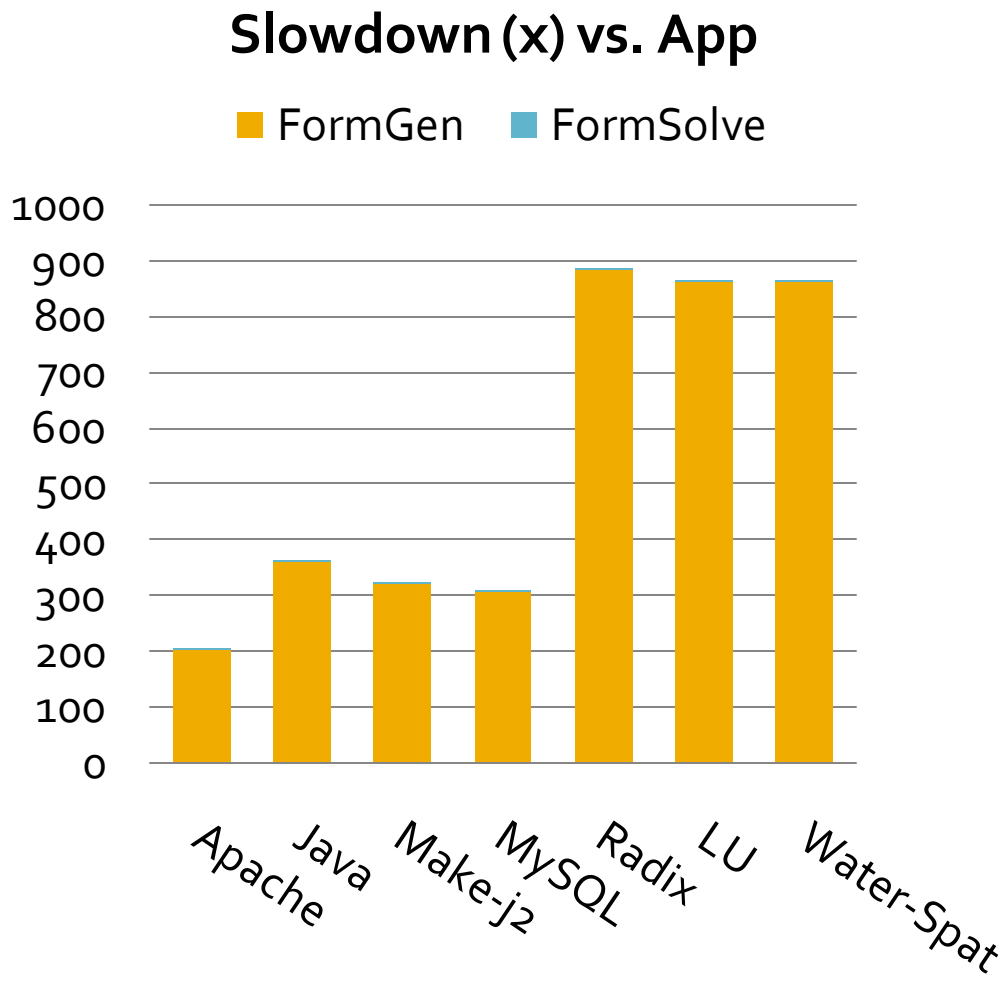


Termination: will obtain divergence-free, SAT path at some point

QI-DRI: Inference Time

Result: Two orders of magnitude improvement

- Why? *We know path*, so no need to backtrack
- Tradeoff: ~6x record-mode slowdown



Unexplored Avenues

- Reduce the path-search space
  - Observation: divergences caused by race-tainted branches (RTBs)
  - Predict and record the RTB outcomes
- Reduce the cost of each backtrack
  - Observation: RTB analysis expensive
  - Reuse results from previous iteration(s)
- Parallelize formula generation
  - On divergence, fork at RTB and explore both paths

Conclusion

What have we shown?

Aim for output-determinism, get record-efficient multiprocessor replay

What needs more work?

Inference time is impractically high, but many avenues remain unexplored

Questions?