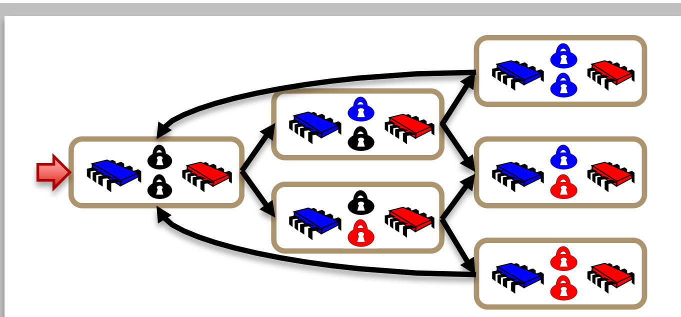
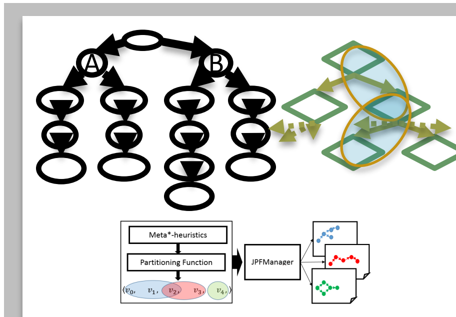
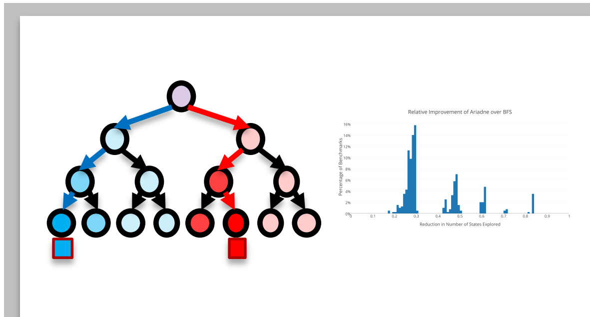
Exceptional service in the national interest

Sandia National Laboratories







Ariadne: Hybridizing Directed Model Checking and Static Analysis

Reed Milewicz and Peter Pirkelbauer

March 2017

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-mission laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

Overview

- Who am I?
- Background; Research Questions
- Related Work
- Introducing Ariadne
- Design and Implementation
- Experimental Results

Who am I?

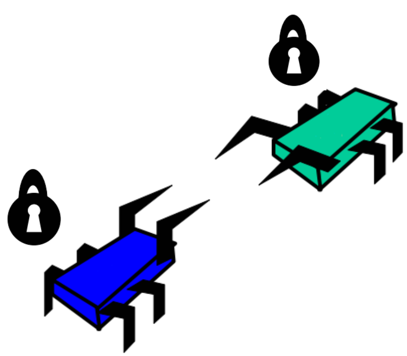
Sandia National Laboratories

- Postdoctoral researcher at Sandia National Laboratories
- My areas of interest
  - Empirical Software Engineering
  - Static and Dynamic Analysis
  - Source-to-Source Transformation Systems
  - Machine Learning

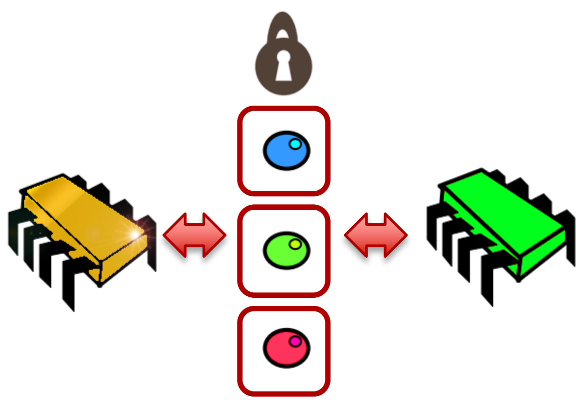
Background

Background

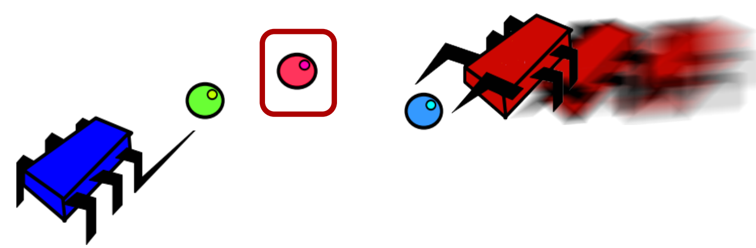
- Concurrency bugs are notoriously resistant to testing, highly latent, and can be very dangerous.



Deadlocks



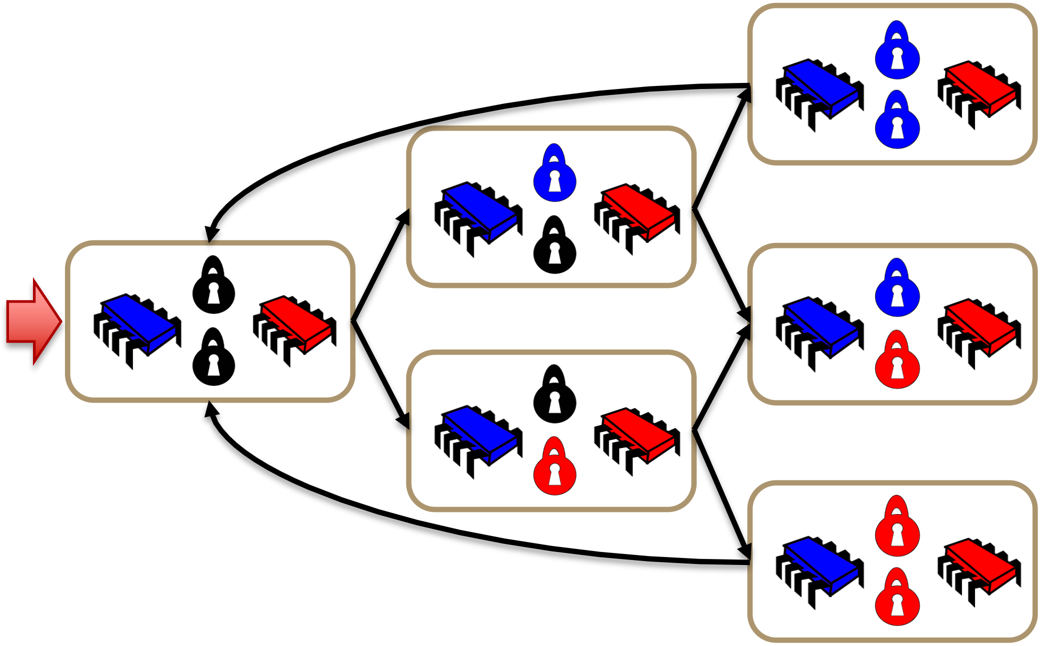
Atomicity Violations



Race Conditions

Background

- Directed model checking (DMC) is a powerful verification strategy against concurrency bugs.
- Combinatorial explosion is managed through combinations of parallel and heuristic search.



Two Dining Philosophers as Kripke Structure

Problem Statement

- **Problem:** It does not matter how good a technique is on paper <u>if no one uses it</u>!

- DMC, like most formal verification techniques, is heavy-weight and demands *significant* labor inputs. This is a barrier to adoption. (Engler and Musuvathi 2004)
  - The need for design, selection, and parameterization of heuristics is a barrier to practical use. Research on metaheuristics applicable to this domain has stalled. (Sörensen 2015)
  - Massive parallelism is now inexpensive but that doesn't remove the need for aggressive state space reduction.

Problem Statement

- Development teams often use multiple tools to test and verify their software. These sources of information can provide insights to benefit the model checking process.
- Model checking as a platform for the synthesis for different analyses.
- Challenges:
  - Need for strategies to manage imperfect and/or conflicting evidence at scale.
  - Need for loose coupling between model checker and external sources.
  - Need for rigorous, empirical methods to validate effectiveness.

Related Work

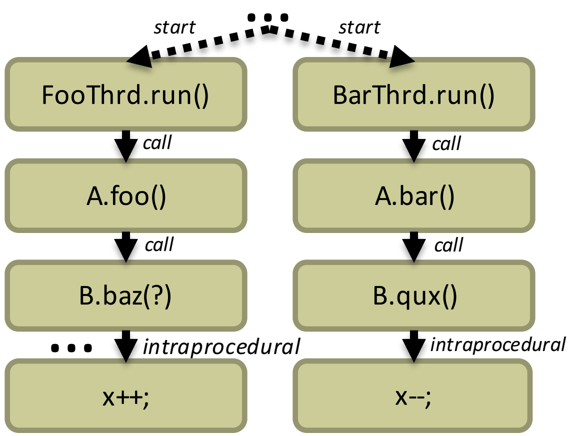
- This is not the first work on this subject, nor will it be the last:
  - Refining static analysis through model checking (Post et al. 2008; Darke et al. 2012; Muske and Khedker 2015)
  - Bi-directional collaboration between static analysis and model checking (Beyer et al. 2007; Chen and MacDonald 2008)
  - Analogous work for dynamic analysis and model checking (Groce and Joshi 2008; Milewicz and Pirkelbauer 2016)
- How this work differs:
  - Our emphasis on producing more informed heuristics than state space reduction.
  - Our aim is to elaborate strategies for using model checking as a loosely-coupled platform for the synthesis of analyses.

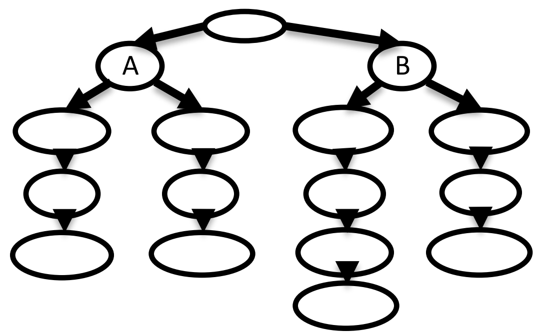
Ariadne: Design and Implementation

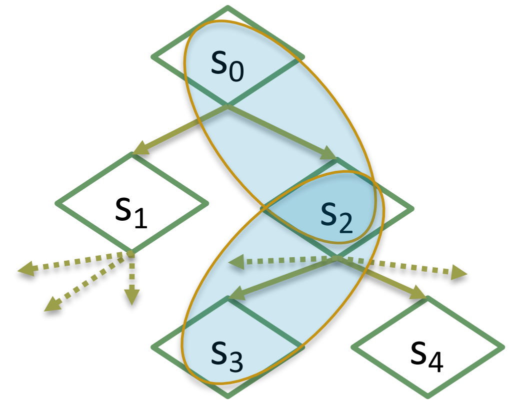
Introducing Ariadne

- An open-source toolchain and algorithm
  - Translates reports of suspected race conditions of a static analyzer (Petablox)…
  - To dynamic metadata annotations using a source-to-source compiler (ROSE)…
  - Which are exploited by directed model checker (Java Pathfinder).

Design and Implementation







Call graph,
Interthread CFG

Abstract Syntax
Tree

Kripke Structure
(State Space)

Petablox

```
public static void foo(){B.baz(false)}
public static void bar(){B.qux()}
```

Code 5.1: class A (Client)

```
private static int x = 0;
public static synchronized void baz(boolean flag){
  if(flag)
    x++;
  else
    B.norf();
}
private static void norf() { B.baz(true) };
public static void qux(){x--;}
```

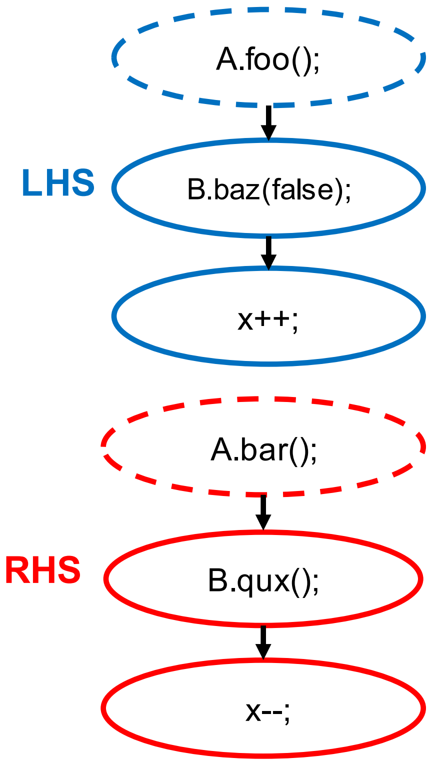
Code 5.2: class B (Library)

```
Possible race condition detected!
Path of abstract thread #1:
  A.foo() in Thread.run()
  B.baz(false) in A.foo
  x++ in B.baz
Path of abstract thread #2:
  A.bar() in Thread.run()
  B.qux() in A.bar
  x-- in B.qux
```

Petablox Bug Report

Translating Petablox Reports

- Race conditions can be understood as paths through the call graph, which gives way to a measure of relative distance to a violation.

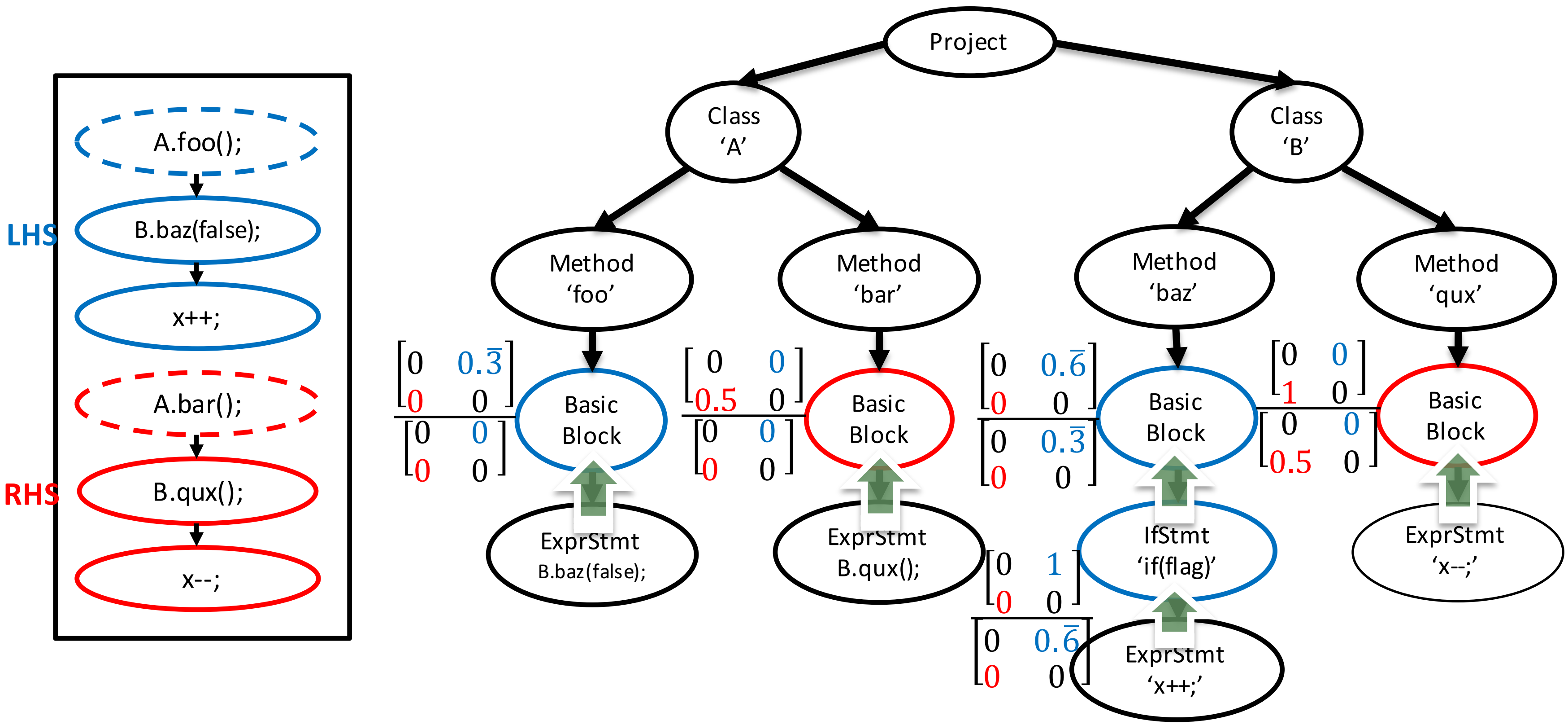


$$B_i^b = \begin{array}{c} \\ lhs \\ rhs \end{array} \begin{array}{c} \begin{array}{cc} lhs & rhs \end{array} \\ \begin{bmatrix} 0 & v(b, lhs_0) \\ v(b, rhs_0) & 0 \end{bmatrix} \end{array}$$

$$v(b, \text{path}) = \begin{cases} \frac{\text{path.index}(b)}{\text{path.length}} & \text{if } b \in \text{path} \\ 0 & \text{otherwise} \end{cases}$$

$$M_{race}^b = \oplus_{i=0}^{\|R\|} B_i^b = \begin{array}{c|ccccc|} & lhs_0 & rhs_0 & \cdots & lhs_{\|R\|} & rhs_{\|R\|} \\ \hline lhs_0 & 0 & v(b, lhs_0) & & & 0 \\ rhs_0 & v(b, rhs_0) & 0 & & & \\ \vdots & & & \ddots & & \\ lhs_{\|R\|} & & & & 0 & v(b, lhs_{\|R\|}) \\ rhs_{\|R\|} & 0 & & & v(b, rhs_{\|R\|}) & 0 \end{array}$$

Translating Petablox Reports

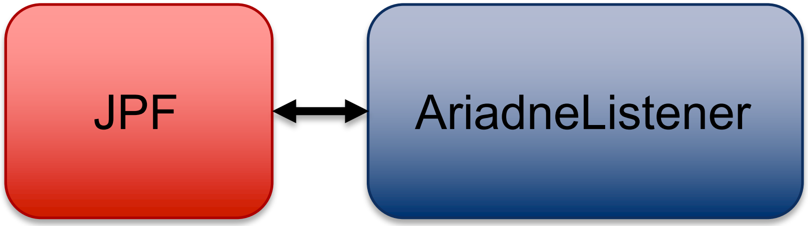


Translating Petablox Reports

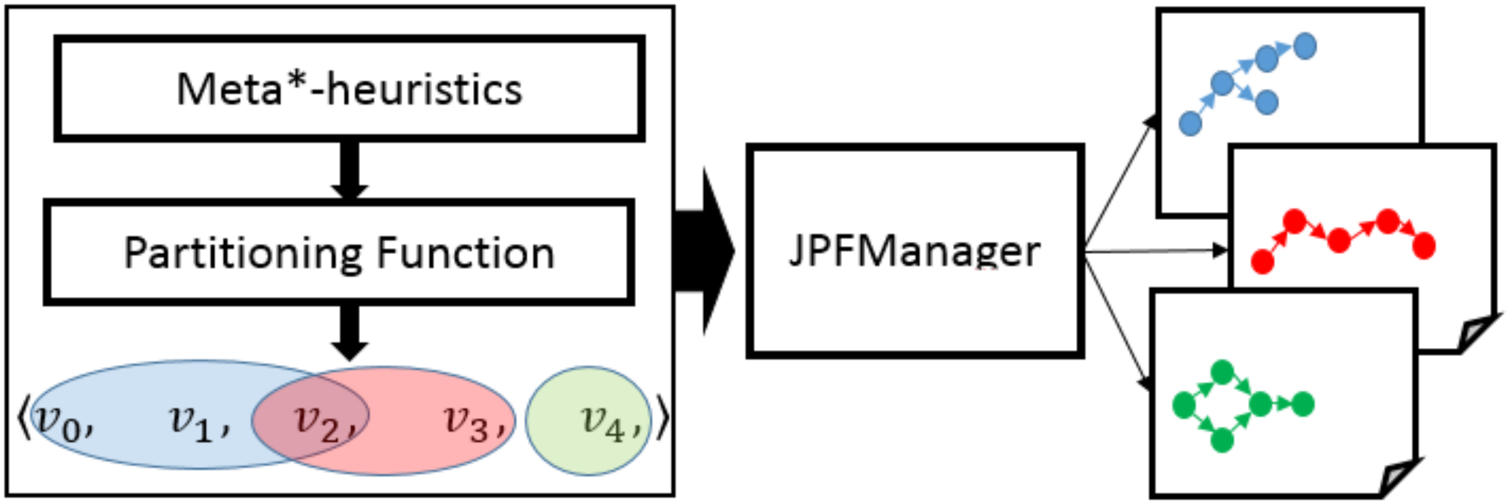
```
public static void foo(){
  float[] rmatrix_foo_enter = {{0.333},{0.0}};
  Ariadne.annotate(rmatrix_foo_enter);
  B.baz(false);
  float[] rmatrix_foo_exit = {{0.0},{0.0}};
  Ariadne.annotate(rmatrix_foo_exit);
}
public static void bar(){
  float[] rmatrix_bar_enter = {{0.0},{0.5}};
  Ariadne.annotate(rmatrix_bar_enter);
  B.qux()
  float[] rmatrix_bar_exit = {{0.0},{0.0}};
  Ariadne.annotate(rmatrix_bar_exit);
}
```

Dynamic Metadata

- The annotations are calls to a metadata library.
- The annotation method itself does nothing, but calls to it are intercepted by a special listener.
- The listener decorates JPF states with annotations which can then be leveraged by a heuristic search.

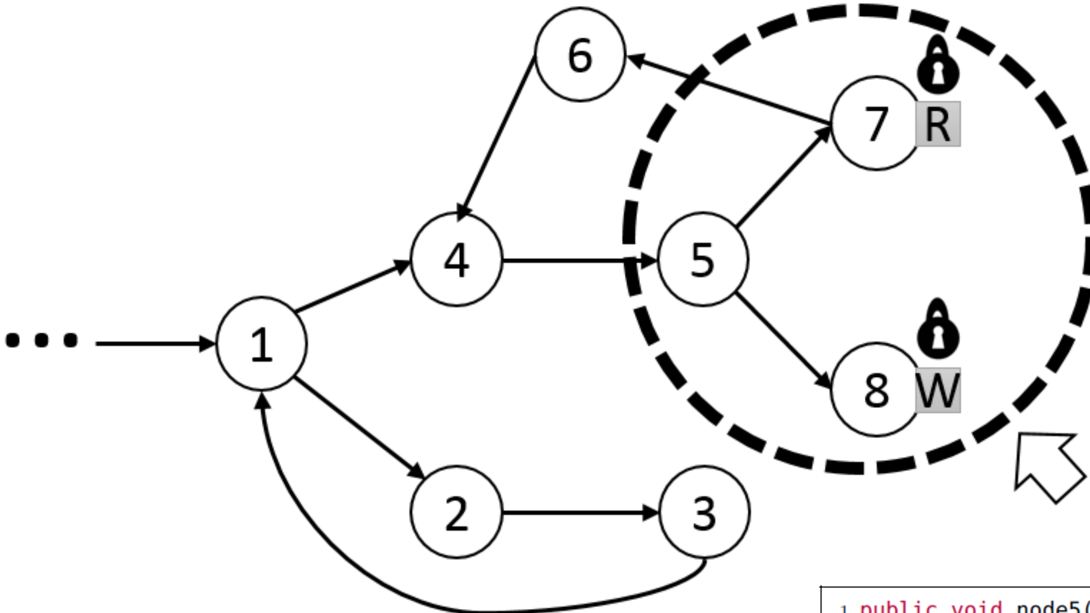


Multiobjective Search



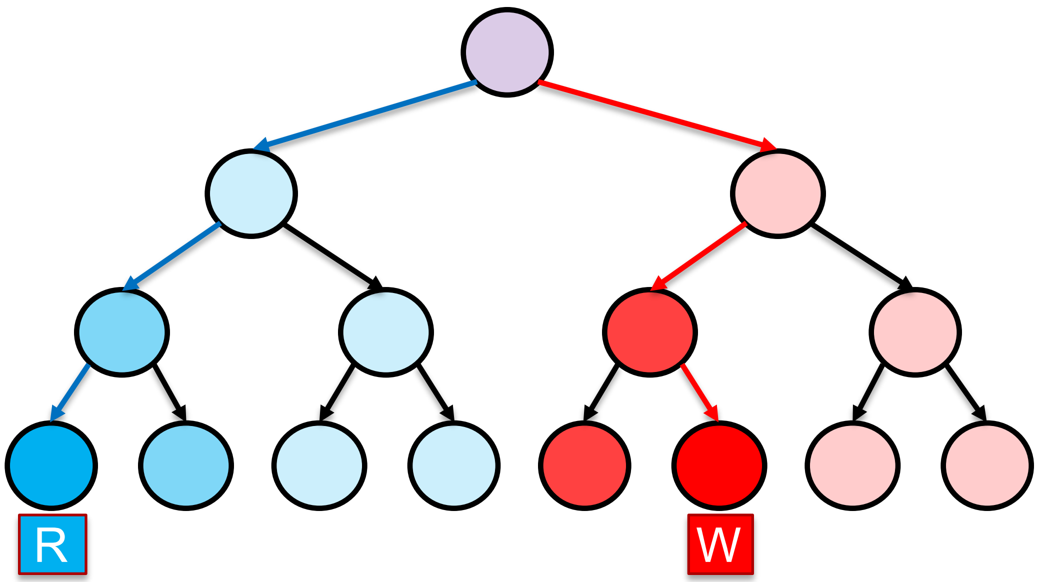
Experimental Results

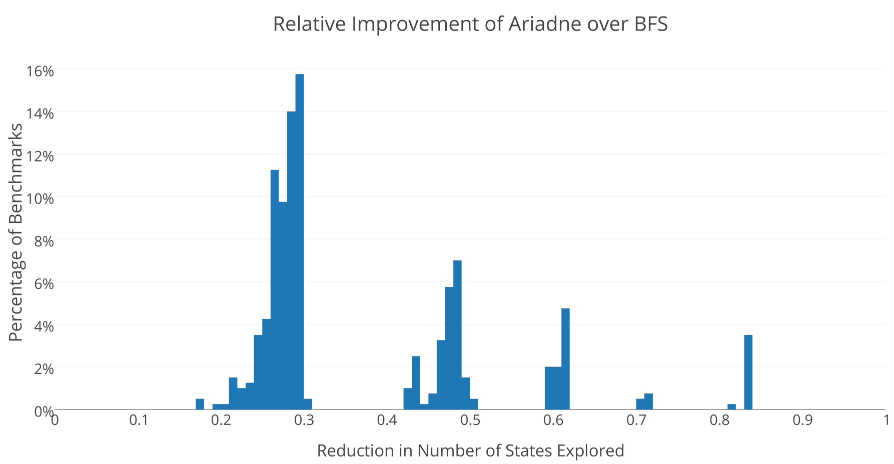
Benchmark Generation




```java
public void node5(){
  int choice = Verify.getInt(0,1);
  if(choice == 0) node7();
  if(choice == 1) node8();
}
public void node7(){ read_loc_1(); node6(); }
public void node8(){ write_loc_1(); }
public int read_loc_1(){
  synchronized(lock1){ return memloc1; }
}
public void write_loc_1(){
  synchronized(lock1){ memloc1 = 1; }
}
```

Branching Factor Tests

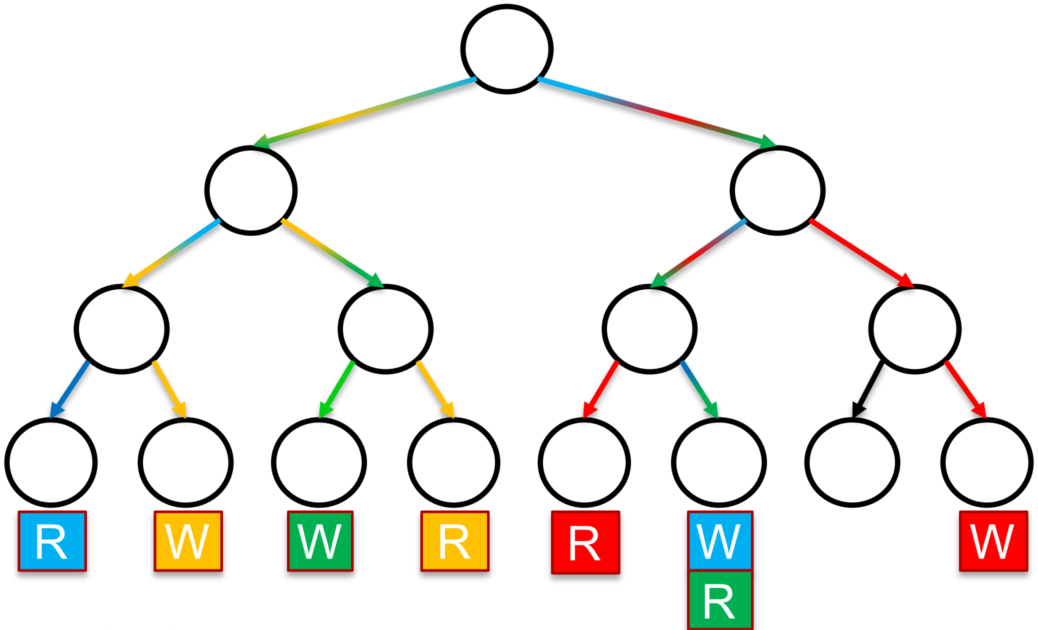


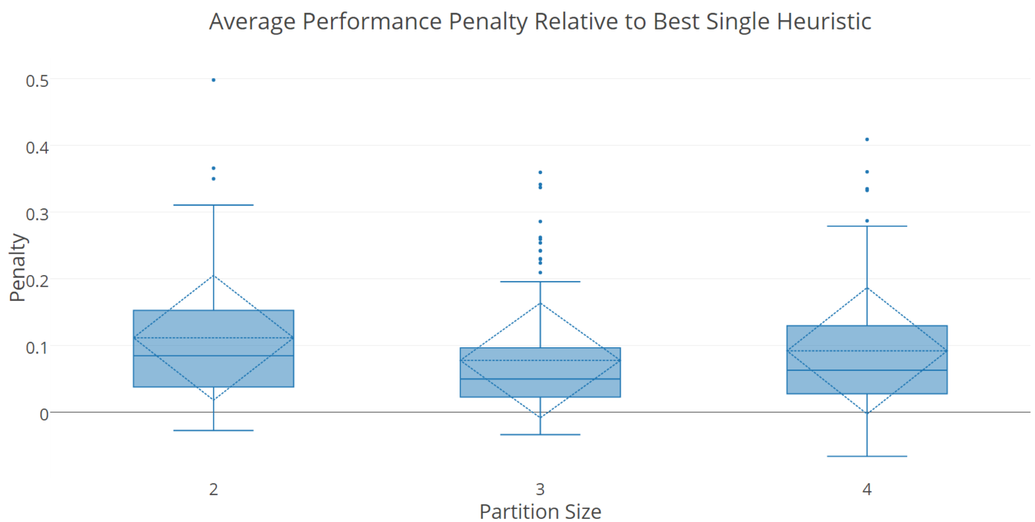


Median Reduction vs. BFS:

$$\frac{1}{e}$$

Partitioning Tests





Performance Penalty

2-partitions: **11%**

3-partitions: **7%**

4-partitions: **9%**

Elevator Benchmark Tests

Heuristic	States Explored, Path Length	Time (sec), Memory Consumed (gb)
BFS	427613 (44)	135 s (12.39 gb)
Interleaving	974806 (101)	350 s (31.77 gb)
Ariadne (single race, best)	**105120** (44)	**42** s (**5.90** gb)
Ariadne (single race, avg)	419242 (44)	136 s (7.97 gb)
Ariadne (partitioned, best)	**280818** (45)	**113.278** s (**7.29** gb)
Ariadne (partitioned, avg)	439473 (44)	138 s (7.90 gb)

4x fewer states, 263 gb required

1.52x fewer states, **79 gb** required

Partitioned, multiobjective search allows for flexible parallelization.

Ariadne: Conclusion

- Even given the imperfect nature of the static analysis (e.g. high false positive rate), we are able to find real bugs faster by integrating it into the search process.
- Dynamic metadata is a cost-effective vehicle for communicating information to the model checker.
- Multiobjective search allows us to efficiently allocate resources at scale with the volume of evidence we receive from the prior analysis.

Questions?