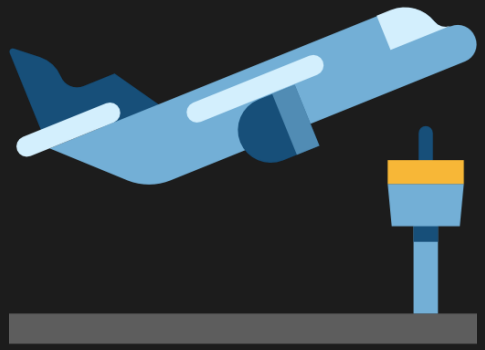
Modular Verification of Interrupt-Driven Software

Chungha Sung | *Markus Kusano* | *Chao Wang*

University of Southern California | Virginia Tech

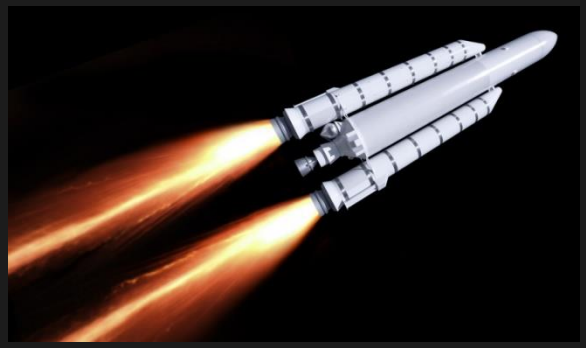
Interrupt-driven Software



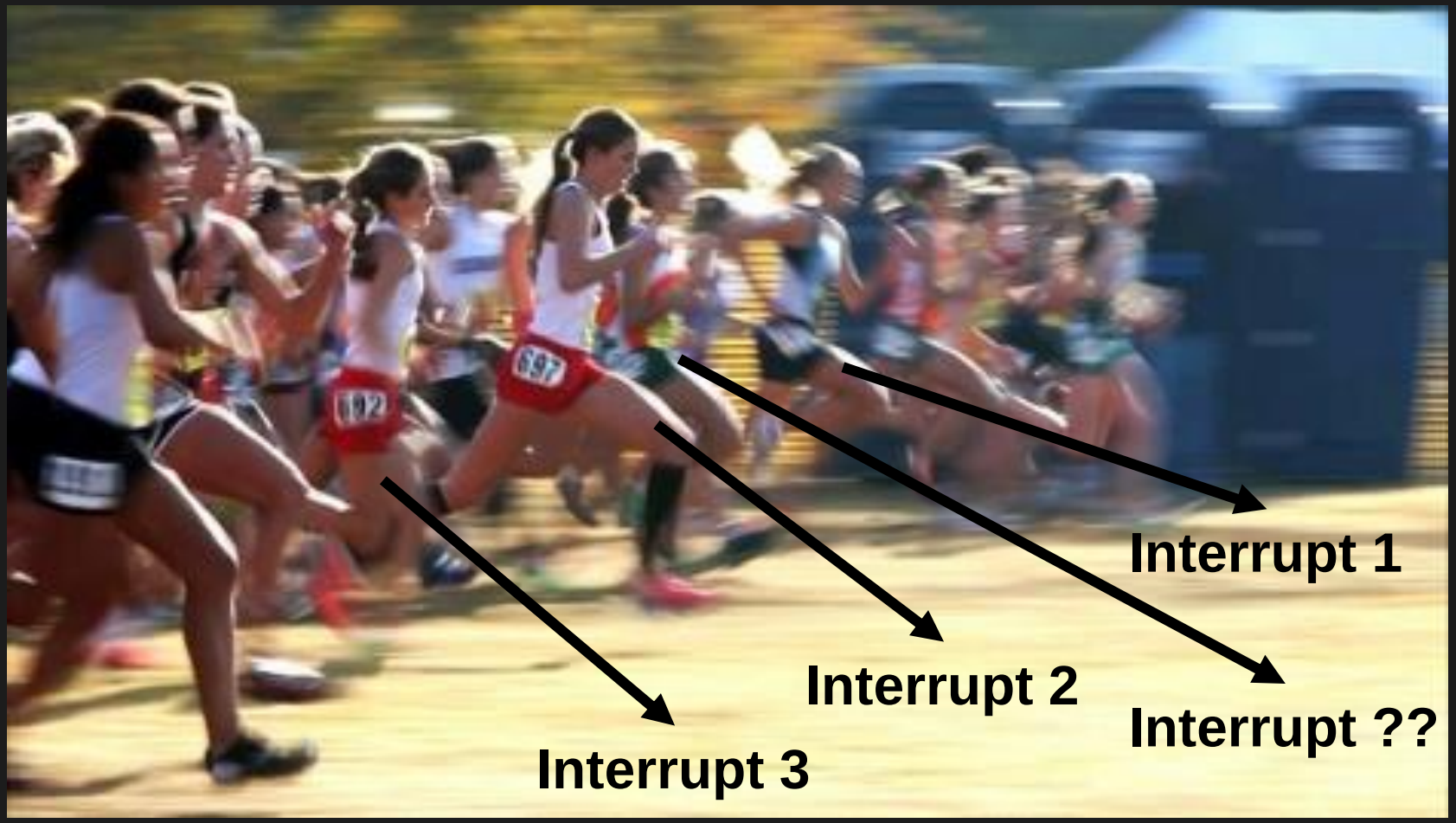
Verification of Interrupts?

Without verification

With verification



Verification of Interrupts?



Prior Works

- **Testing – Hard to explore all possible combinations**

  ex) [Regehr *ICES 2005*]
  [Wang et al. *ISSTA 2017*]

- **Bounded Model Checking - Cannot prove validity of assertions**

  ex) [Kroening et al. *DATE 2015*]

Cannot provide proof!

Our Approach

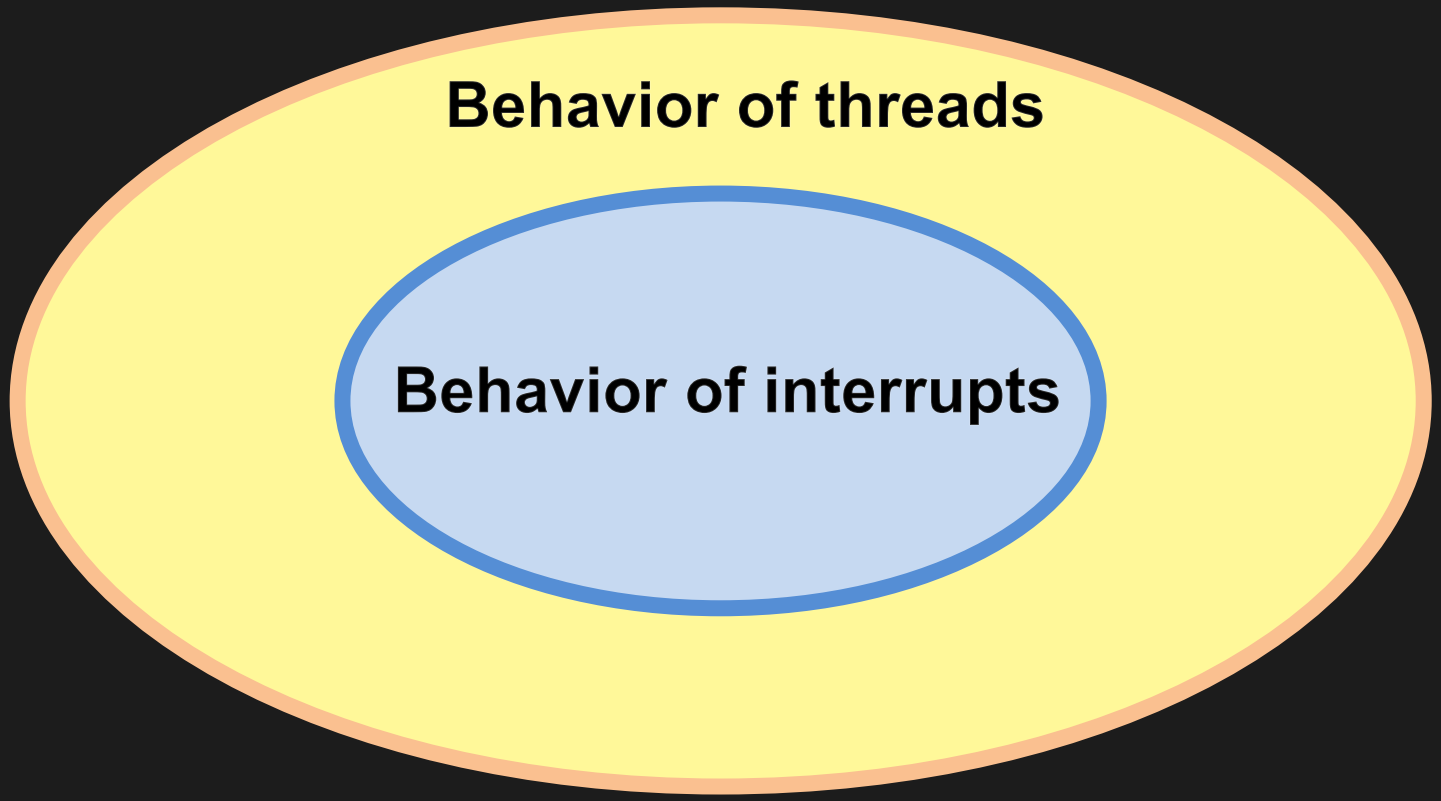
- **Abstract interpretation – good for obtaining proofs**

  [Cousot & Cousot *POPL 1977*]

- **Modular analysis – Only for thread behavior**

  ex) [Miné *VMCAI 2014*]
  [Kusano & Wang *FSE 2016/2017*]

Behavior of Interrupts



Behavior of Interrupts

Under thread behavior

```
T1() {
    a = 1;
    x = a;
};
```

```
T2() {
    a = 2;
};
```

Under interrupt behavior (T1's priority > T2' priority)

```
T1() {
    a = 1;
    x = a;
};
```

```
T2() {
    a = 2;
};
```

Outline

Motivation

Contribution
(The first modular verification method for interrupt-driven software)

Experiments

Conclusion

Overview

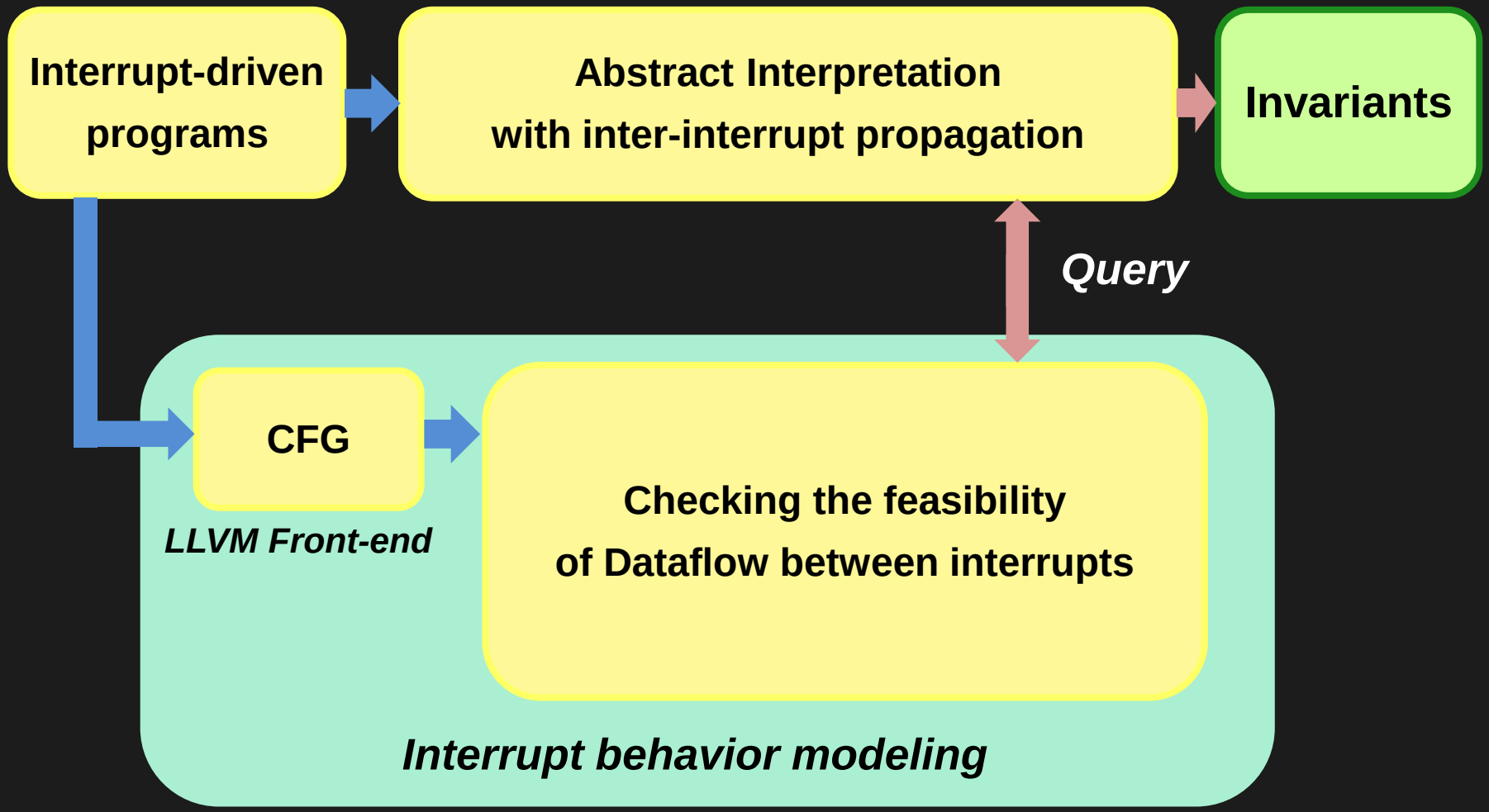
Modular analysis

Data-flow feasibility
based on interrupt behavior

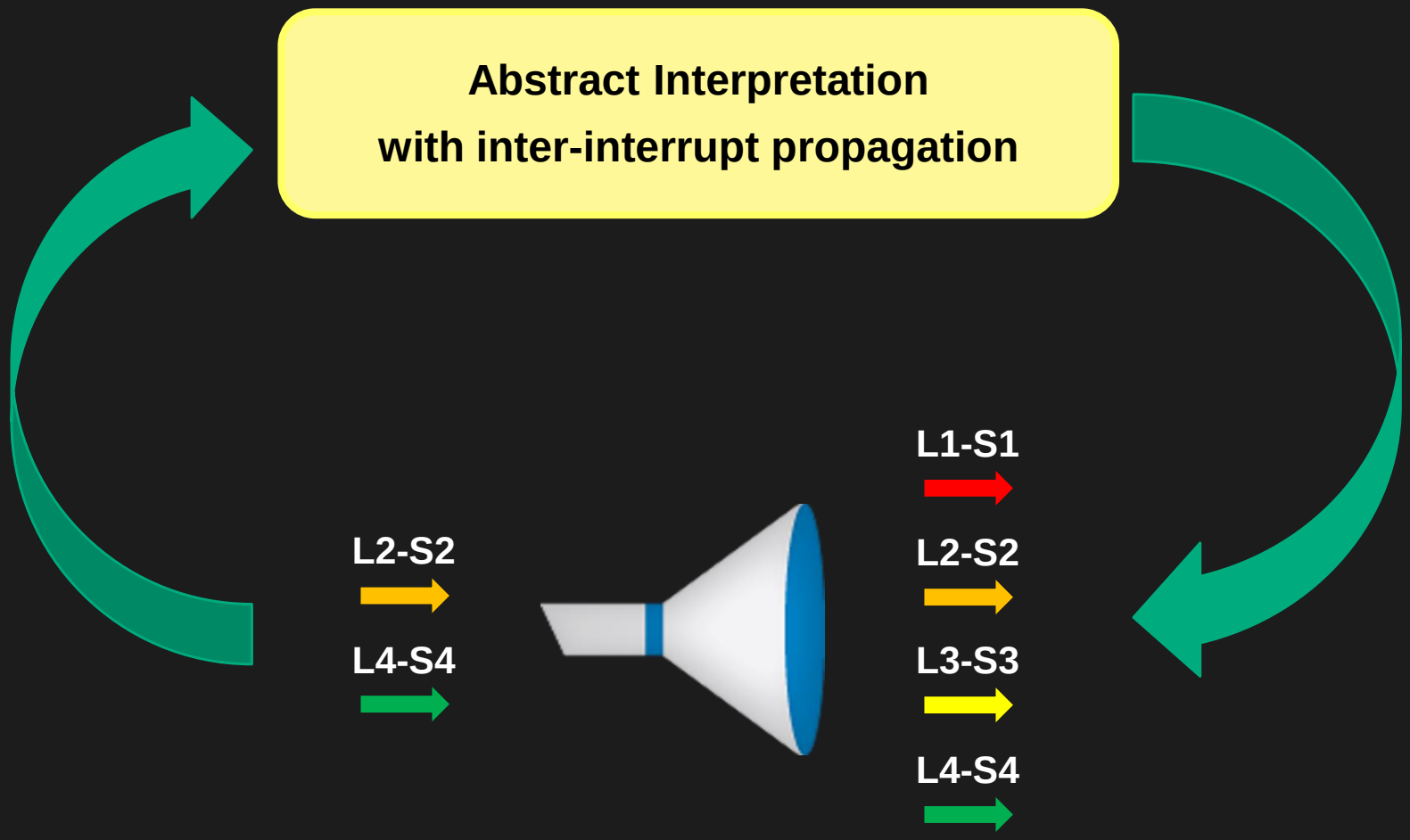
Modular analysis for interrupts

More accurate!!

Overview


Interrupt-driven programs
Abstract Interpretation with inter-interrupt propagation
Invariants
Query
CFG
LLVM Front-end
Checking the feasibility of Dataflow between interrupts
Interrupt behavior modeling

Feasibility Checking



Motivating Example 1

Priority: L < H

```
Irq_L() {
   x = 1;
};
```

```
Irq_H() {
   x = 0;
   assert(x == 0);
};
```

Motivating Example 1

Priority: L < H

```
Irq_L() {
  x = 1;
};
```

```
Irq_H() {
  x = 0;
  assert(x == 0);
};
```

<u>Thread behavior: The assertion can be violated!</u>

Motivating Example 1

Priority: L < H

```
Irq_L() {
  x = 1;
};
```

```
Irq_H() {
  x = 0;
  assert(x == 0);
};
```

<u>Interrupt behavior: The assertion holds!</u>

Motivating Example 2

Priority: L < H

```
Irq_L() {
   x = 1;
};
```

```
Irq_H() {
   assert(x == 0);
};
```

Motivating Example 2

Priority: L < H

```
Irq_L() {
  x = 1;
};
```

```
Irq_H() {
  assert(x == 0);
};
```

<u>Thread behavior: The assertion can be violated!</u>

Motivating Example 2

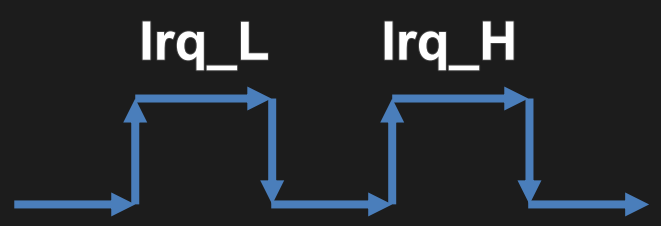
Priority: L < H

```
Irq_L() {
  x = 1;
};
```

```
Irq_H() {
  assert(x == 0);
};
```

<u>Thread behavior: The assertion can be violated!</u>

<u>Interrupt behavior: The assertion can be violated as well!</u>



Motivating Example 3

Priority: L < H

```
Irq_L() {
   assert(x == 0);
};
```

```
Irq_H() {
   if (…)
      x = 1;
   x = 0;
};
```

Motivating Example 3

Priority: L < H

```
Irq_L() {
  assert(x == 0);
};
```

```
Irq_H() {
  if (…)
    x = 1;
  x = 0;
};
```

<u>Thread behavior: The assertion can be violated!</u>

Motivating Example 3

Priority: L < H

```
Irq_L() {
  assert(x == 0);
};
```

```
Irq_H() {
  if (…)
    x = 1;
  x = 0;
};
```

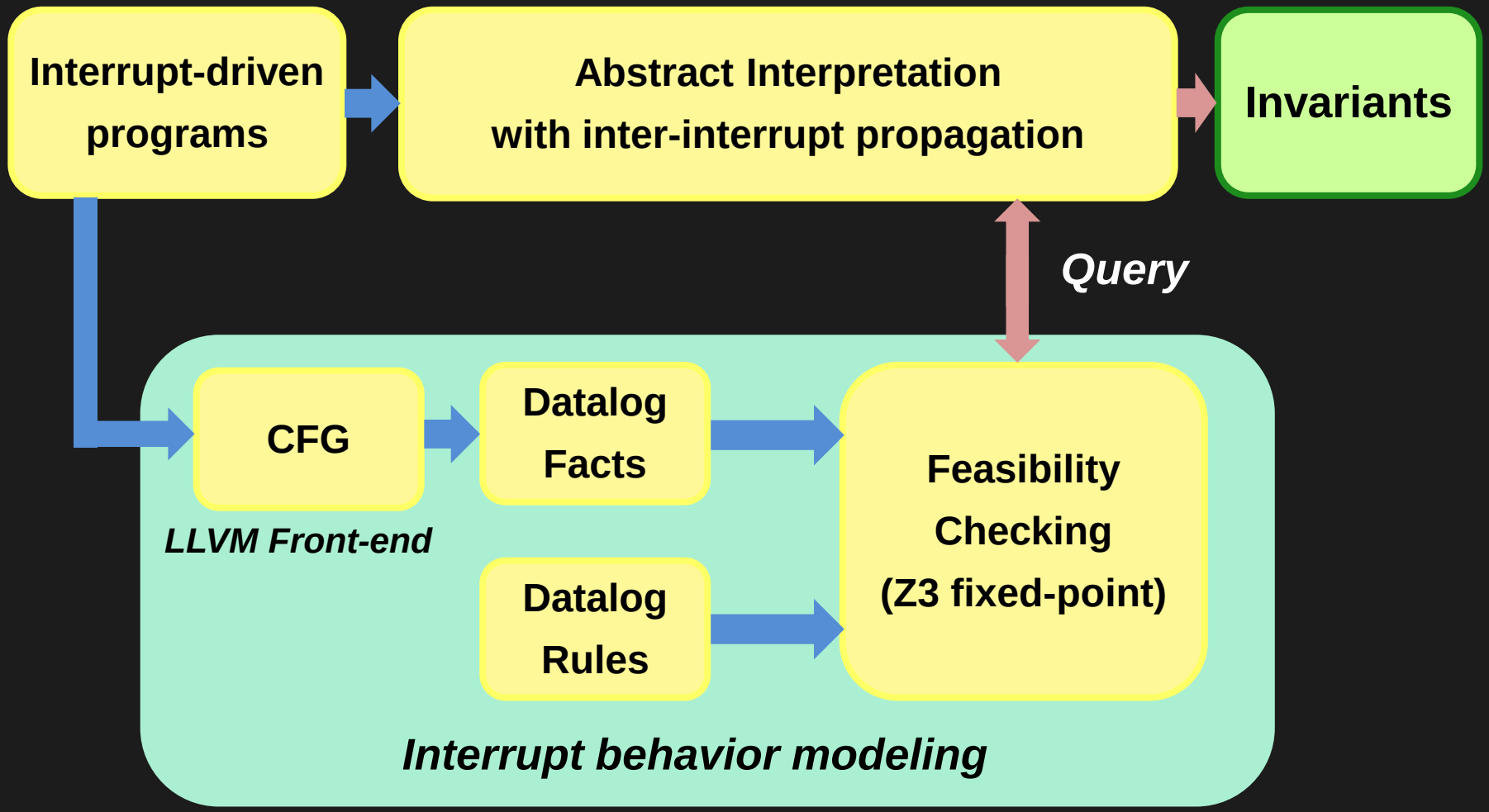
Post-dominate

<u>Interrupt behavior: The assertion holds!</u>

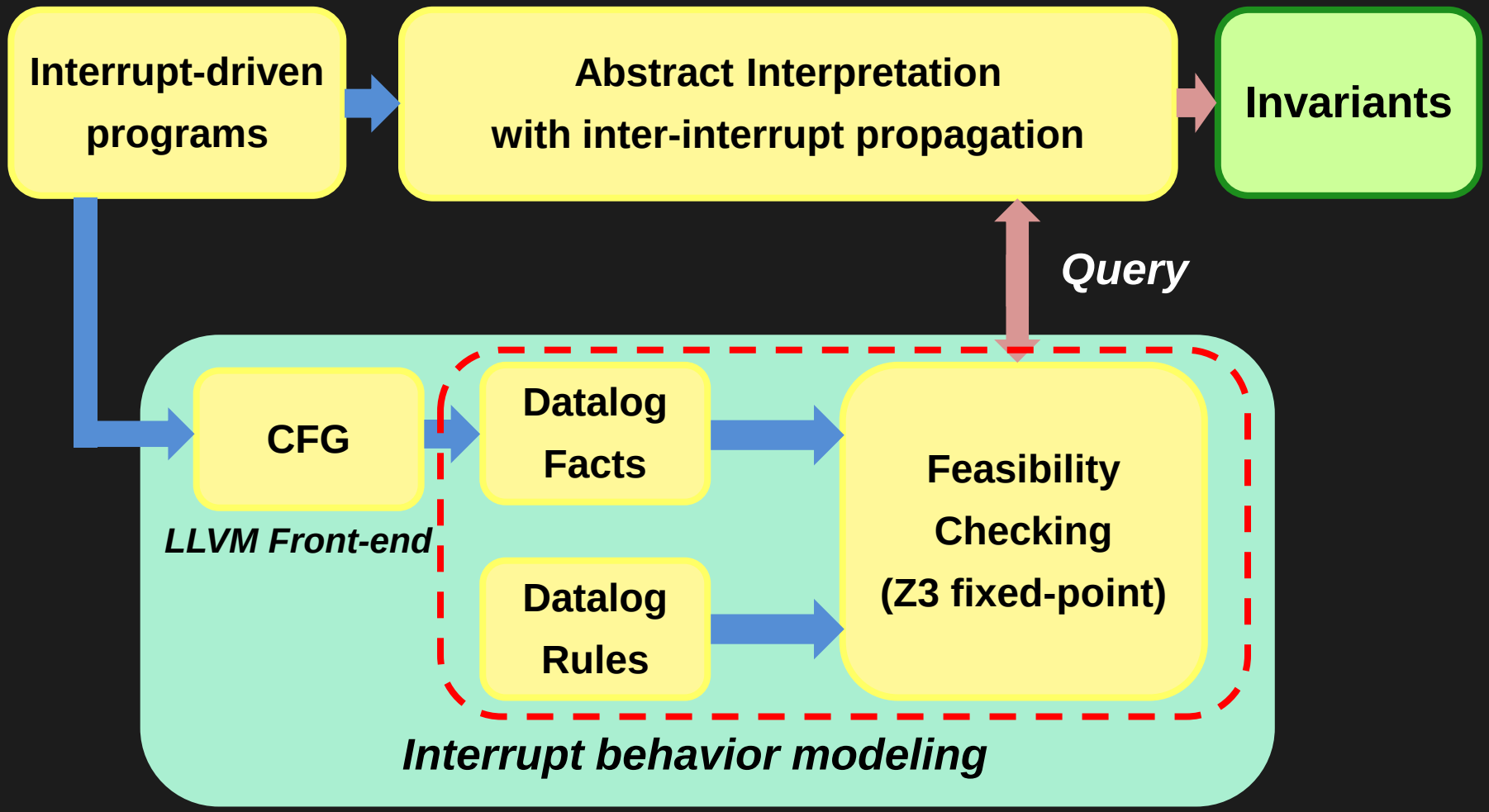
Summary

	Thread behavior (Existing)	Interrupt behavior (Our approach)
Example1	Warning	***Proof***
Example2	Warning	Warning
Example3	Warning	***Proof***

Implementation



Implementation



Program Analysis in Datalog

[Whaley & Lam, 2004]
[Livshits & Lam, 2005]

Interrupt-driven software → Datalog facts

Datalog rules

Datalog Engine

Data-flow Feasibility between interrupts

What is Datalog?

Declarative language for deductive databases [Ullman 1989]

Facts

parent (bill, mary)
parent (mary, john)

Rules

ancestor (X, Y) ← parent (X, Y)
ancestor (X, Y) ← parent (X, Z), ancestor (Z, Y)

New relationship: ancestor (bill, john)

Datalog Translation

```
Irq_L() {
  x = 1;
};
```

NoPreempt

```
Irq_H() {
  x = 0;
  assert(x == 0);
};
```

NoPreempt (s1, s2) <- Pri(s1, p1) & Pri(s2, p2) & (p2 ≥ p1)

NoPreempt (x=1, x==0) <- Pri(x=1, L) & Pri(x==0, H) & (H ≥ L)

Datalog Translation

```
Irq_L() {
  x = 1;
};
```

Dominate

```
Irq_H() {
  x = 0;
  assert(x == 0);
};
```

CoveredLoad

CoverdLoad(l) <- Load(l, v) & Store (s, v) & Dom (s, l)

CoveredLoad(x==0) <- Load(x==0) & Store(x=0) & Dom(x=0, x==0)

Datalog Translation

```
Irq_L() {
  x = 1;
};
```

NoPreempt

MustNotReadFrom

```
Irq_H() {
  x = 0;
  assert(x == 0);
};
```

CoveredLoad

MustNotReadFrom(l, s) <-
CoveredLoad(l) & NoPreempt (s, l) for the same variable

MustNotReadFrom(x==0, x=1) <-
CoveredLoad(x==0) & NoPreempt (x=1, x==0) for x

Datalog Translation

```
Irq_L() {                          Irq_H() {
  assert(x == 0);   NoPreempt        if (…)
};                                     x = 1;
                                       x = 0;
                                   };
```

NoPreempt (s1, s2) <- Pri(s1, p1) & Pri(s2, p2) & (p2 ≥ p1)

NoPreempt (x==0, x=1) <- Pri(x==0, L) & Pri(x=1, H) & (H ≥ L)

Datalog Translation

```
Irq_L() {
  assert(x == 0);
};
```

```
Irq_H() {
  if (…)
InterceptedStore x = 1;
  x = 0;
};
```

Post-dominate

InterceptedStore(s1) <- Store(s1, v) & Store(s2, v) & PostDom(s1, s2)

InterceptedStore(x=1) <- Store(x=1) & Store(x=0) & PostDom(x=0, x=1)

Datalog Translation

```
Irq_L() {
  assert(x == 0);
};
```

```
Irq_H() {
  if (…)
    x = 1;
  x = 0;
};
```

NoPreempt

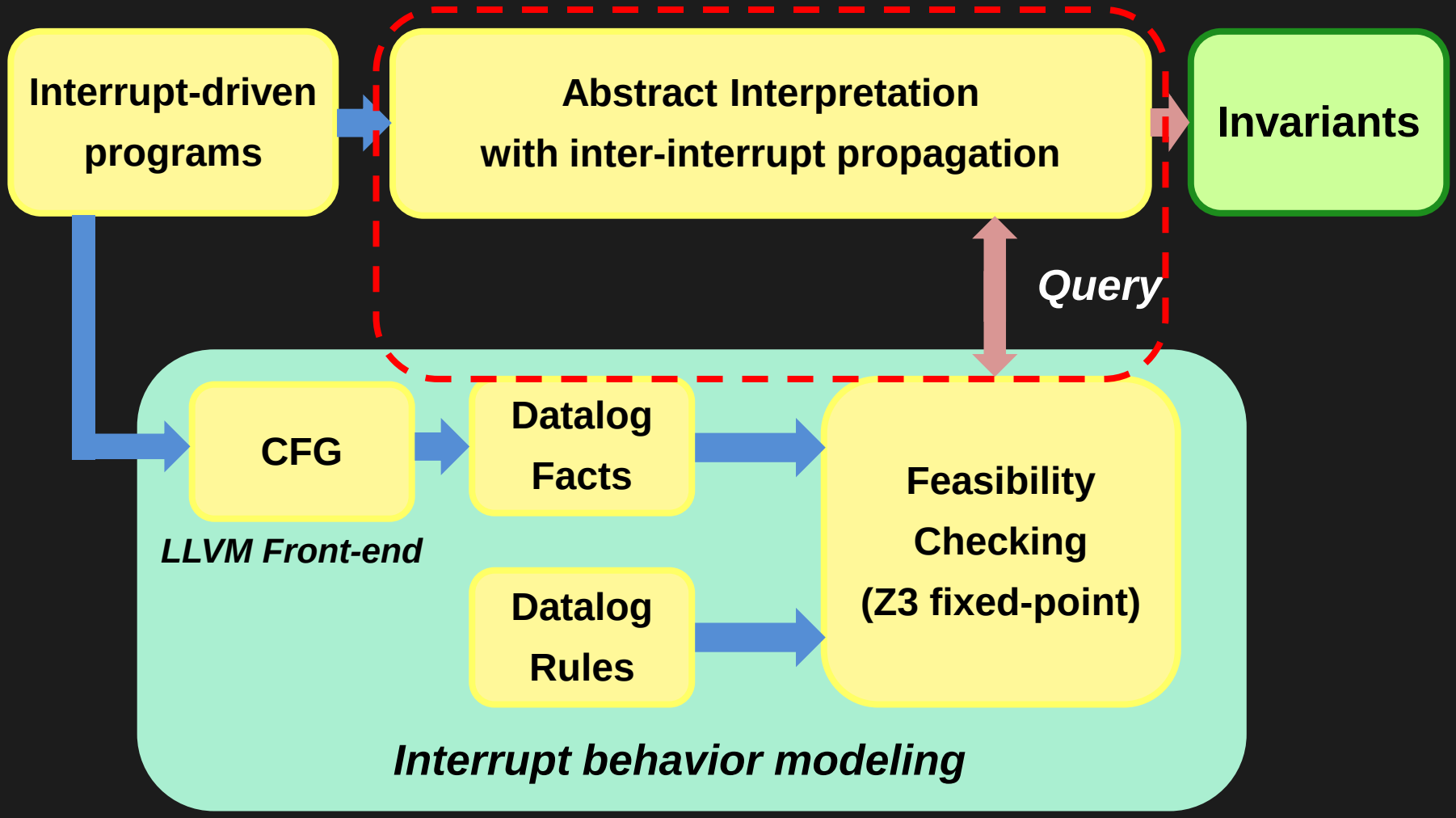
MustNotReadFrom

InterceptedStore

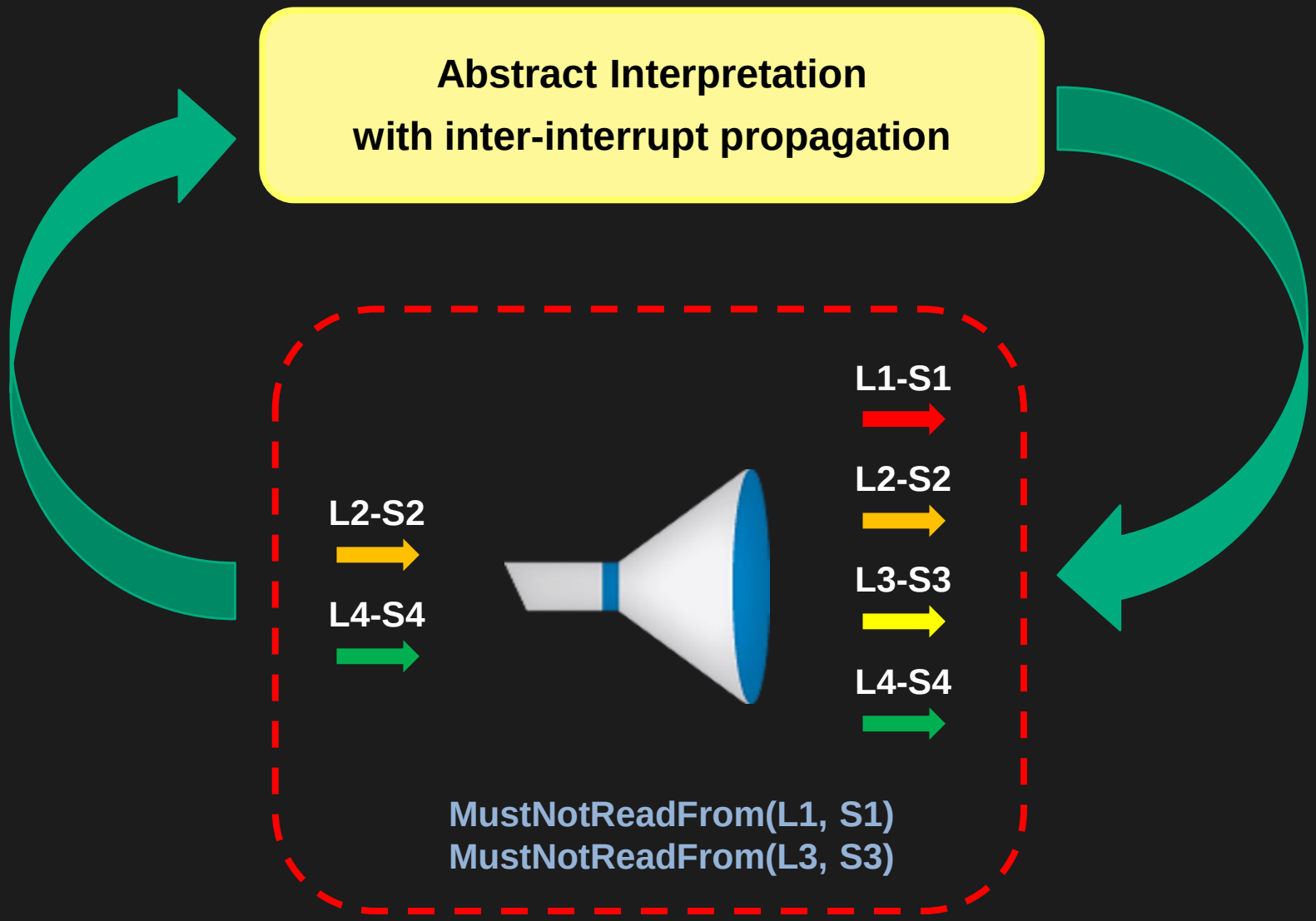
MustNotReadFrom(l, s) <-
InterceptedStore(s) & NoPreempt(l, s) for the same variable

MustNotReadFrom(x==0, x=1) <-
InterceptedStore(x=1) & NoPreempt(x==0, x=1) for x

Implementation



Feasibility Check



Experimental Results 1

Summary	
Num. of Benchmarks	35
Total LOC	22,541 lines
Total number of pairs	5,116
Number of filtered pairs	3,560
Analysis time	***64.21 s***

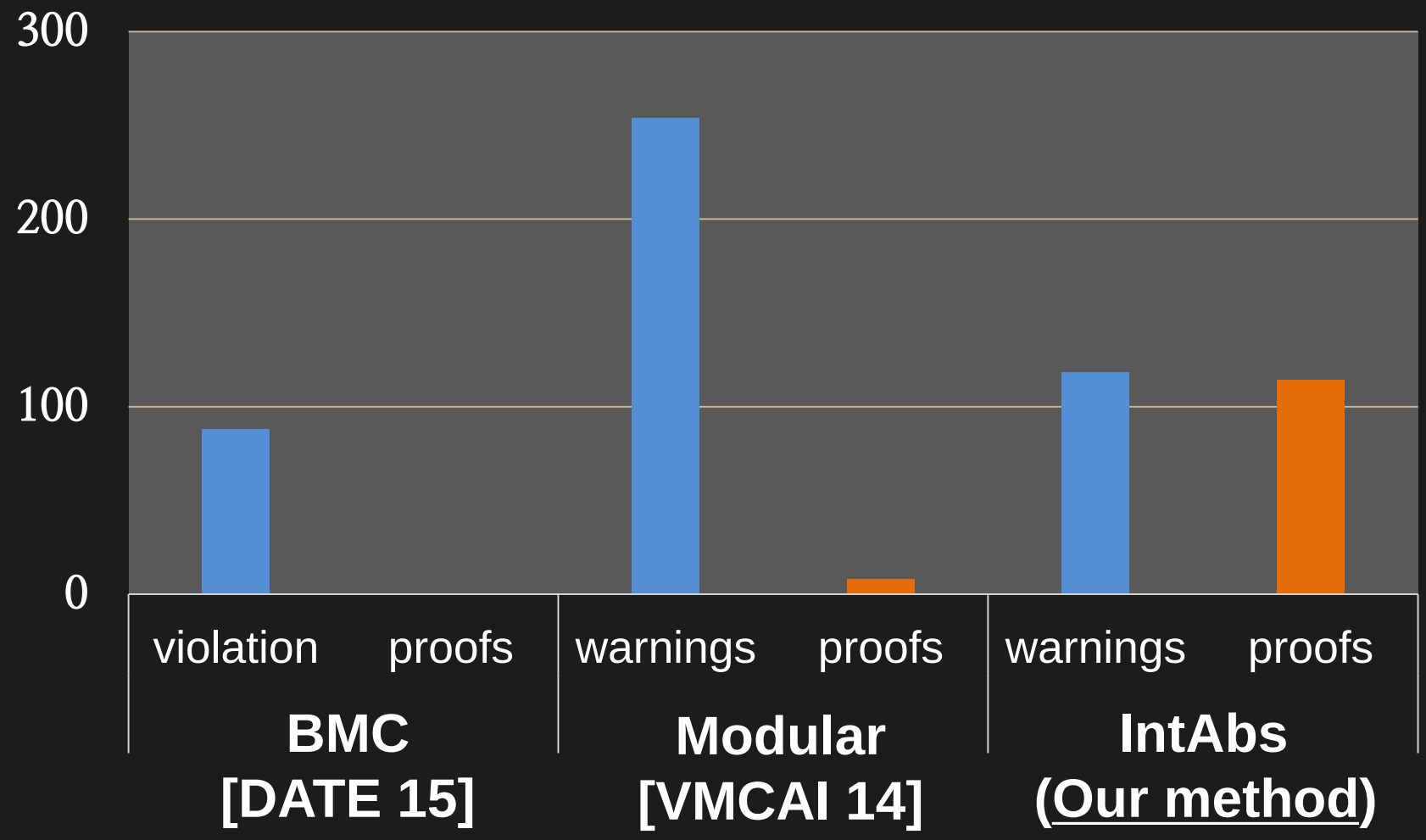
69%

Comparison

- **Bounded Model Checking for interrupts**
  [Kroening et al. *DATE 2015*]

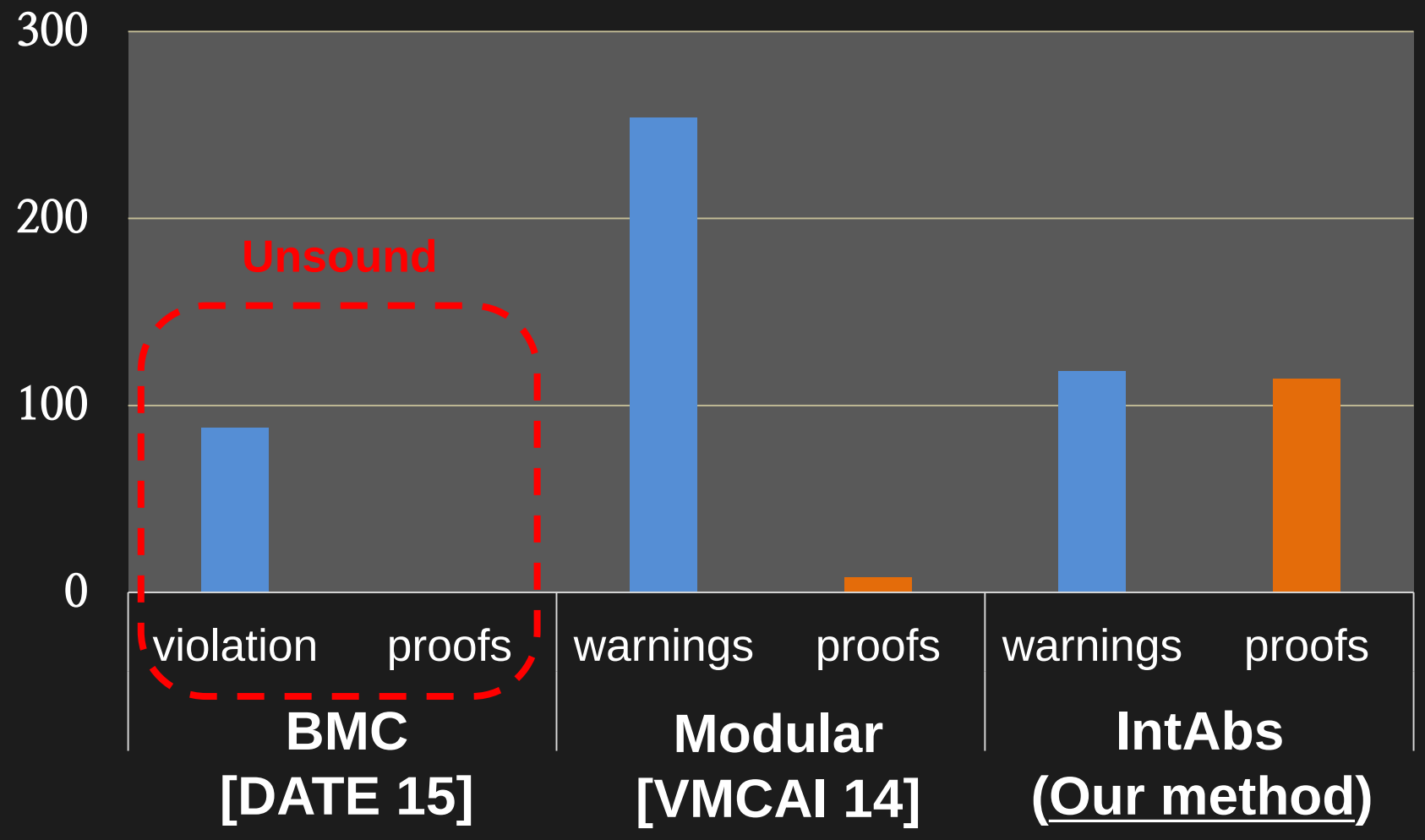
- **Modular analysis for threads**
  [Miné *VMCAI 2014*]

Experimental Results 2



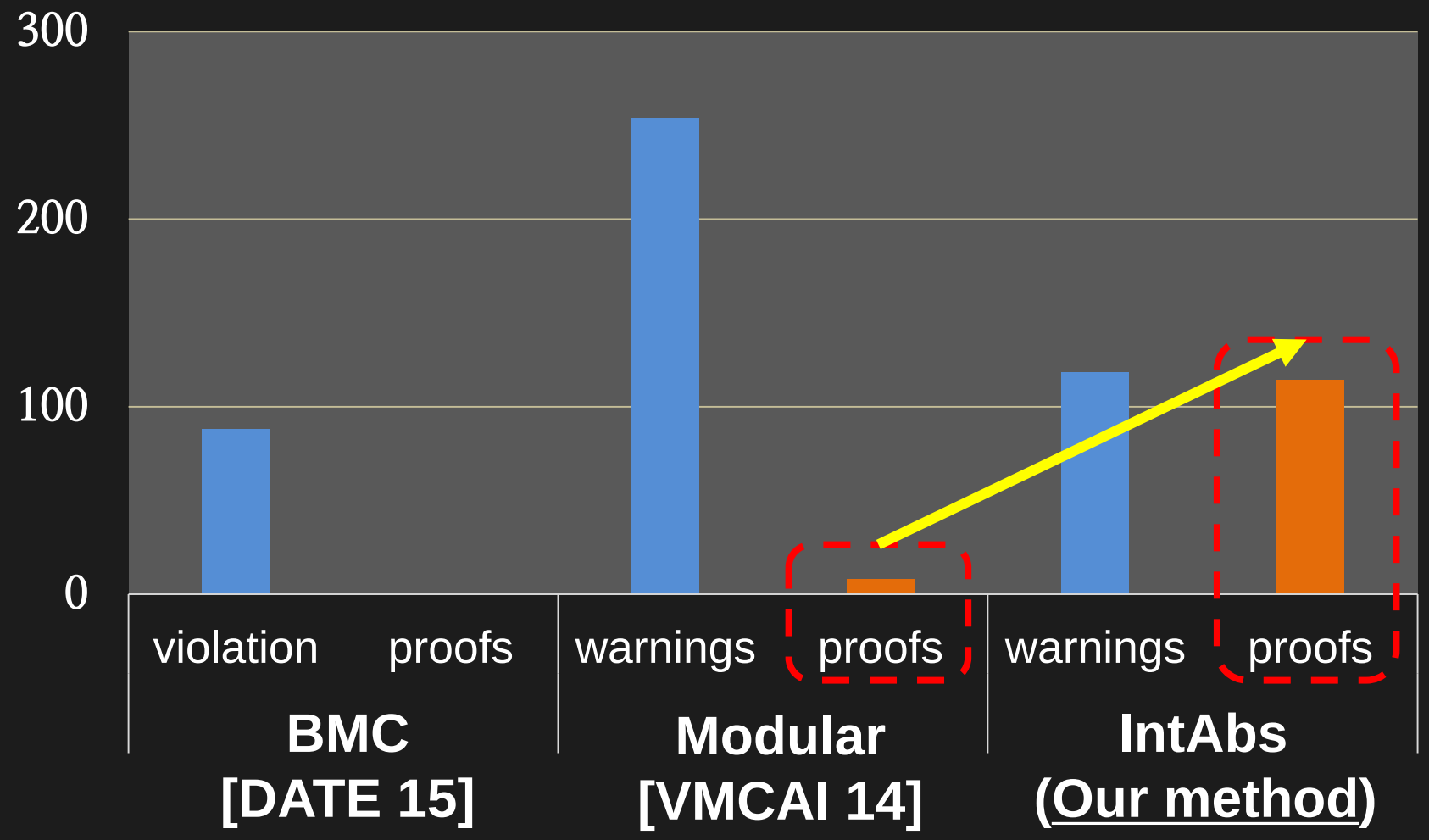
Number of warnings & proofs w.r.t each method

Experimental Results 2



Number of warnings & proofs w.r.t each method

Experimental Results 2



Number of warnings & proofs w.r.t each method

Conclusions

- **Proposed the first modular static analysis method for sound verification of interrupt-driven software**
- **Precisely identified infeasible data flows between interrupts with a declarative interrupt model**
- **Showed significant precision and performance improvements**

Thank you!

https://github.com/chunghasung/intabs