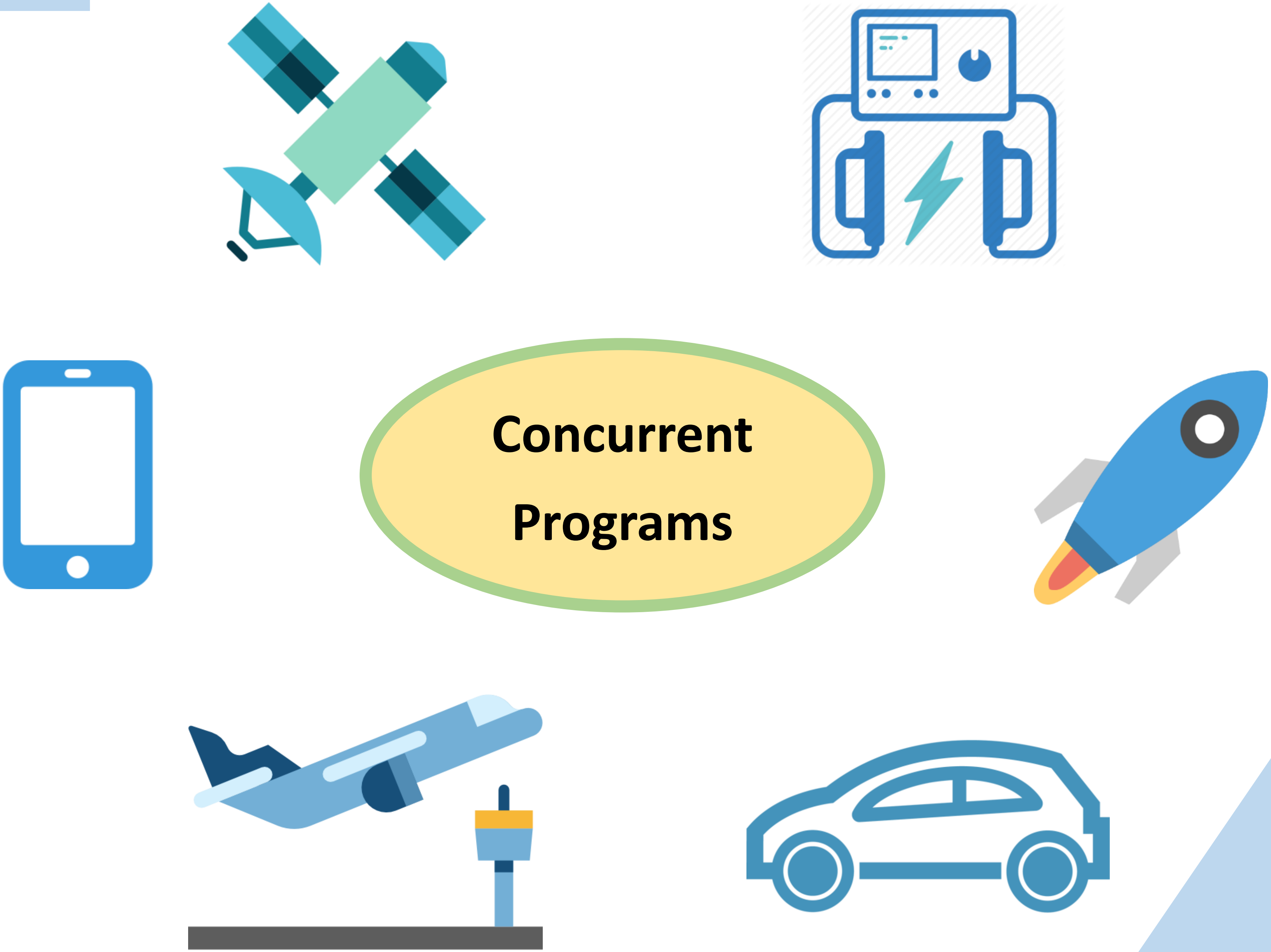
ASE 2018

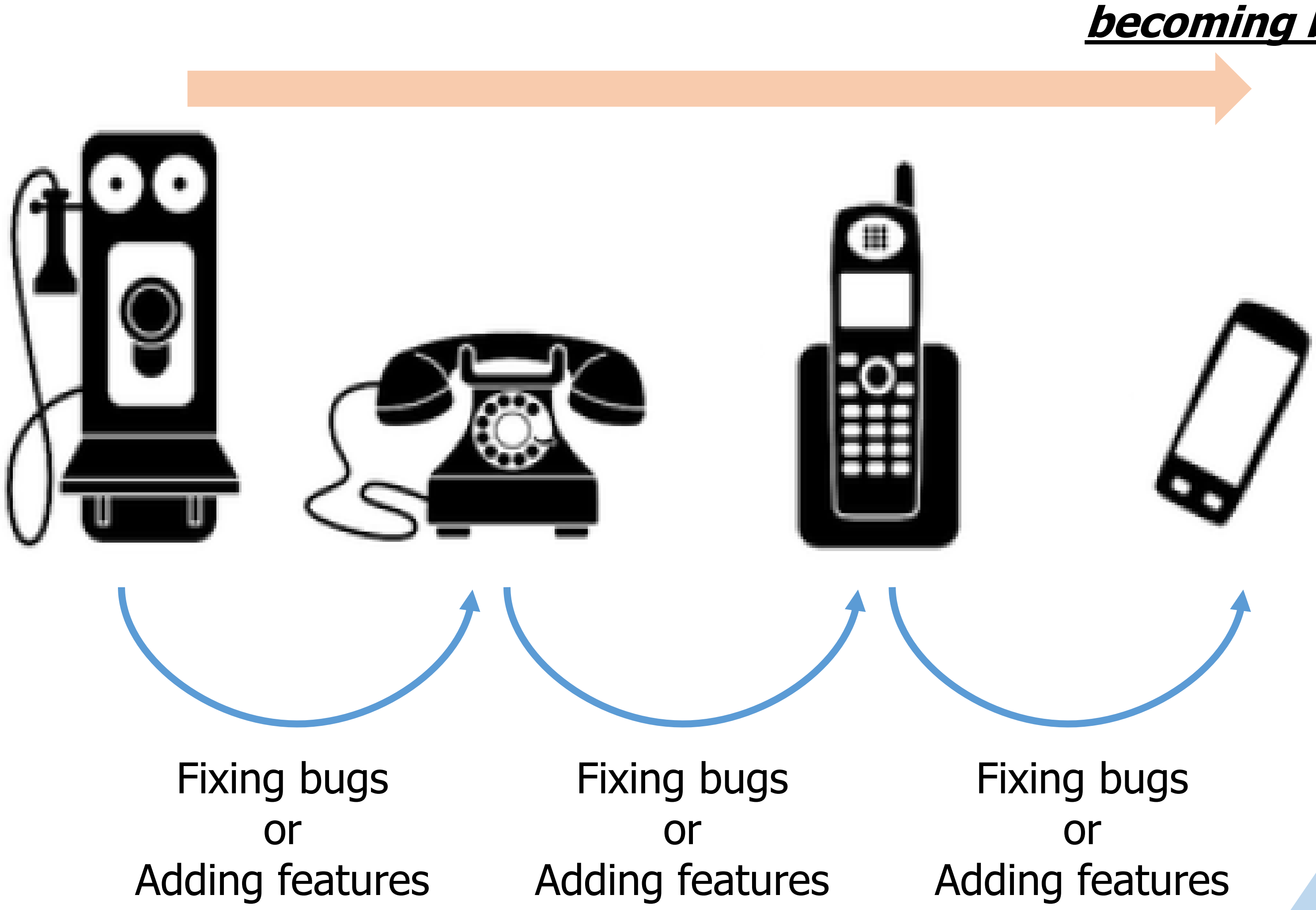
Datalog-based Scalable Semantic Diffing of Concurrent Programs

Chungha Sung | Shuvendu K. Lahiri | Constantin Enea
Chao Wang

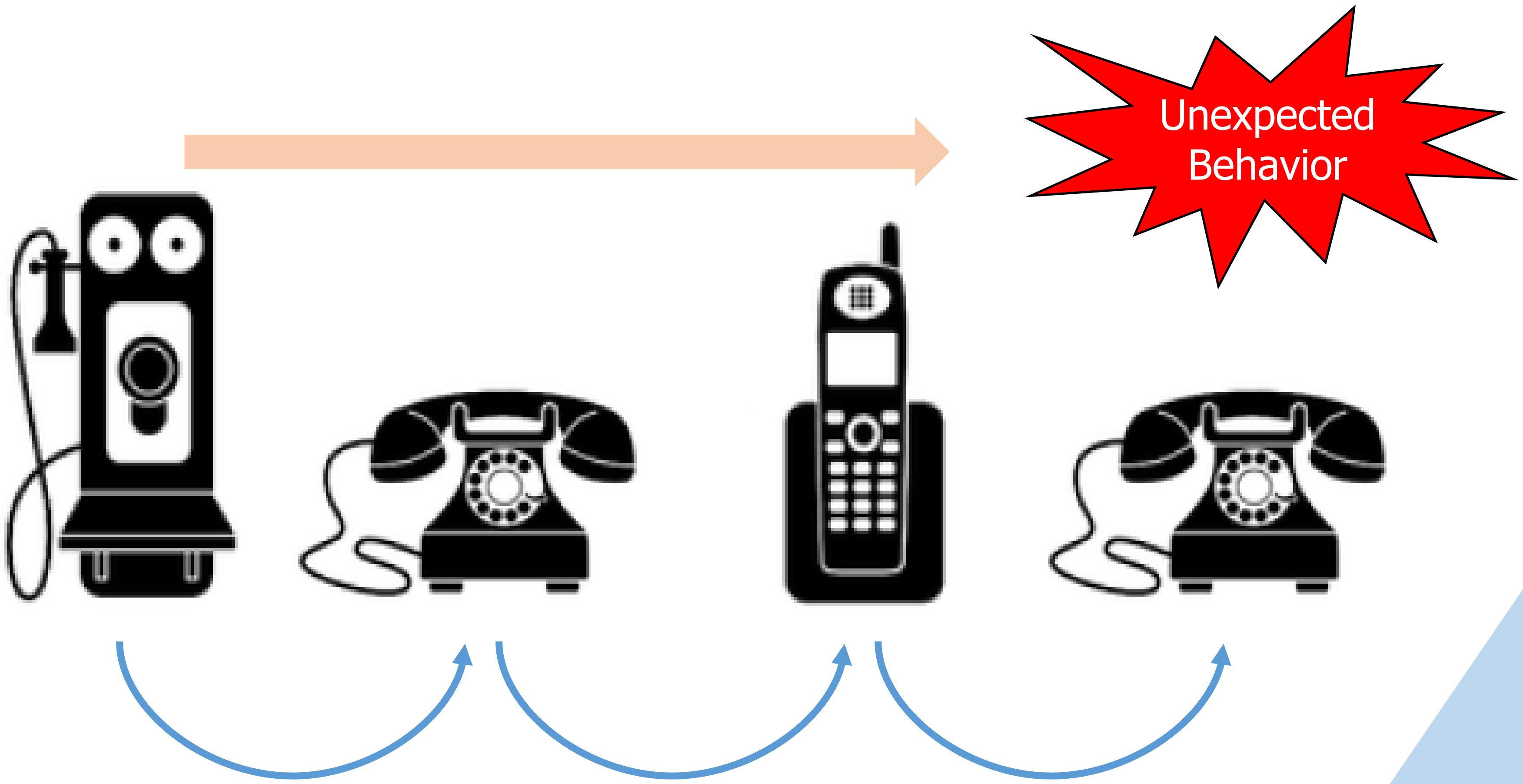
USC University of Southern California
Microsoft Research
université PARIS DIDEROT PARIS 7


Concurrent
Programs

Evolving Software



Evolving Software



Fixing bugs
or
Adding features

Fixing bugs
or
Adding features

Fixing bugs
or
Adding features

Thread 1

```
lock(a);
x = 1;
y = x;
unlock(a);
```

Thread 2

```
lock(a);
x = 0;
unlock(a);
```

Thread 1

```
~~lock(a);~~
x = 1;
y = x;
~~unlock(a);~~
```

Thread 2

```
~~lock(a);~~
x = 0;
~~unlock(a);~~
```

New Read-from edge is created!!

Comparison after a change

Program

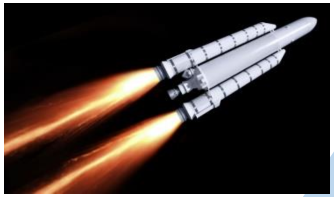
```
while bytecount<0x20000:
    client.CChaltcpu();

    randcode=client.CCpeekiramword(0x08);
    randcount=client.CCpeekiramword(0x0A);

    if randcount!=lastcount and randcode==0xbeef:
        #New bytes are ready, halted in steady state.
        lastcount=randcount;
        for a in range(bytestart,bytestart+bytescount):
            file.write(chr(client.CCpeekirambyte(a)));
            bytecount=bytecount+1;
        print "Got 0x%06x bytes." % bytescount;
        print "%04x %04x: %02x%02x..." % (
            client.CCpeekiramword(0x08),
            client.CCpeekiramword(0x0a),
            client.CCpeekirambyte(0x0C),
            client.CCpeekirambyte(0x0d));

    client.CCreleasecpu();
```

Program after a change

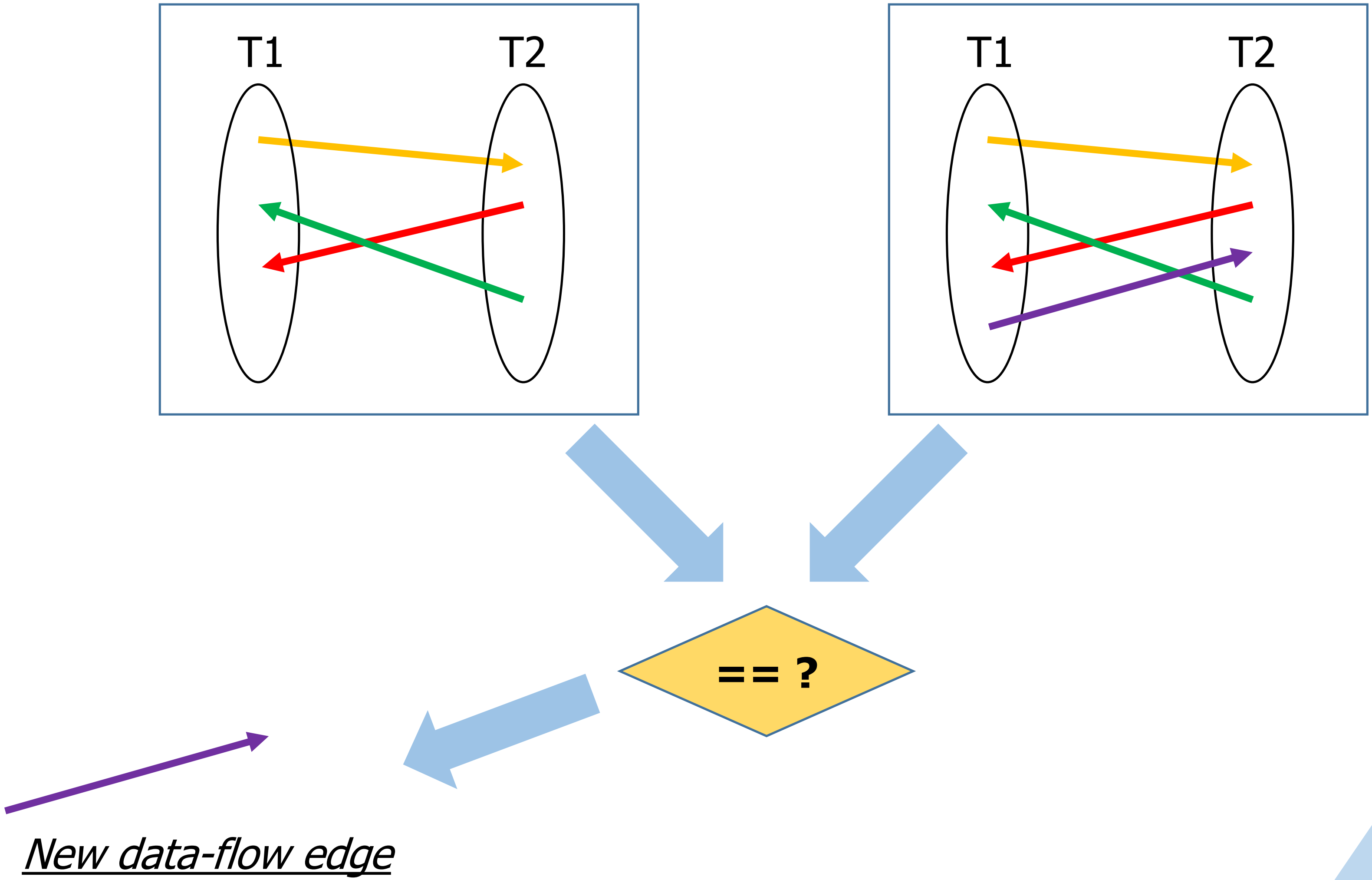

```
while bytecount<0x20000:
    client.CChaltcpu();

    randcode=client.CCpeekiramword(0x08);
    randcount=client.CCpeekiramword(0x0A);

    if randcount!=lastcount and randcode==0xbeef:
        #New bytes are ready, halted in steady state.
        lastcount=randcount;
        for a in range(bytestart,bytestart+bytescount):
            file.write(chr(client.CCpeekirambyte(a)));
            bytecount=bytecount+1;
        print "Got 0x%06x bytes." % bytescount;
        print "%04x %04x: %02x%02x..." % (
            client.CCpeekiramword(0x08),
            client.CCpeekiramword(0x0a),
            client.CCpeekirambyte(0x0C),
            client.CCpeekirambyte(0x0d));

    client.CCreleasecpu();
```


<u>Is there any unexpected new behavior?</u>

NO!



Semantic difference



Prior work

- Bounded Model Checking (BMC) based approach
  - Need to instrument code with assertions
  - Interleaving enumeration => expensive

[Bouajjani et al. *SAS 2017*]

Our approach

- Constraint-based scalable program analysis
  - No code instrumentation needed
  - No interleaving enumeration
  - **10x to 1000x faster**
  - **Practically accurate**

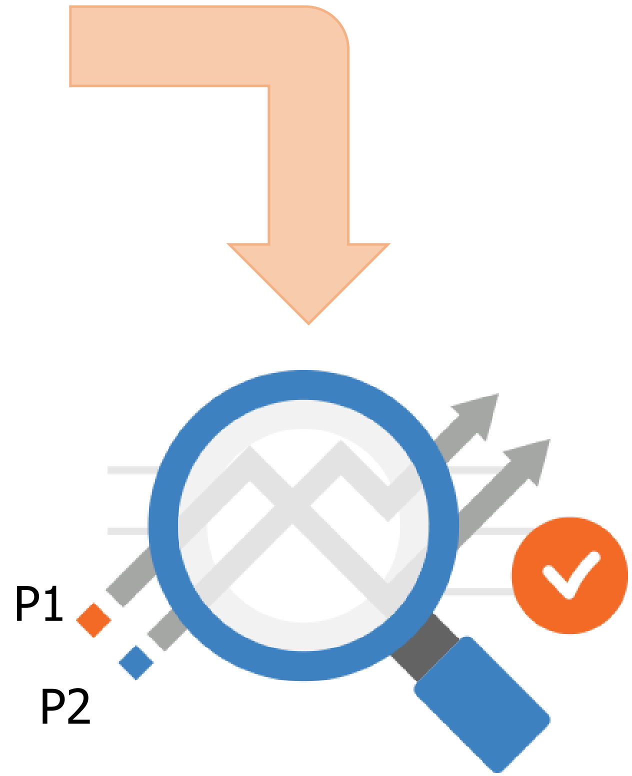
Outline

- Motivation
- ***Contribution***
  (Scalable approximate semantic diffing)
- Experiments
- Conclusion

Overview

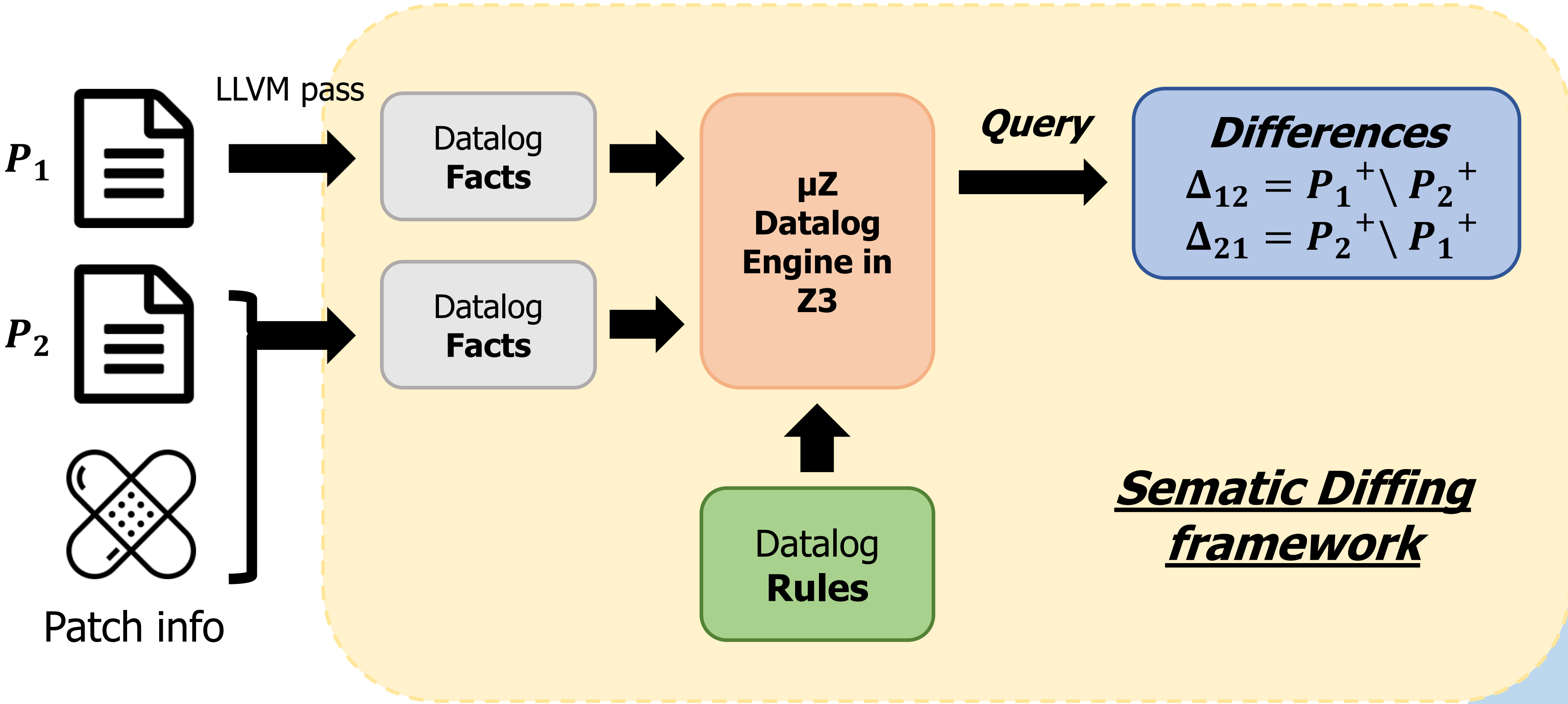
__Scalable & Pratically Accurate!__

Datalog inference rules
for semantic diffing



Compare the allowed data-flow edges over two programs

Overview



LLVM pass
P_1
P_2
Patch info
Datalog
Facts
Datalog
Facts
μZ
Datalog
Engine in
Z3
Datalog
Rules
Query
Differences
$\Delta_{12} = {P_1}^{+} \setminus {P_2}^{+}$
$\Delta_{21} = {P_2}^{+} \setminus {P_1}^{+}$
Sematic Diffing
framework

Example

```
Thread1() {
    t = 0;
    x = 1;
    create(Thread2);
    lock(a);
    …
    assert(x != t);
    unlock(a);
}
```

```
Thread2() {
    lock(a);
    t = x;
    …
    x = 2;
    unlock(a);
}
```

Example

```
Thread1() {
    t = 0;
    x = 1;
    create(Thread2);
    lock(a);
    …
    assert(x != t);
    unlock(a);
}
```

```
Thread2() {
    lock(a);
    t = x;
    …
    x = 2;
    unlock(a);
}
```

Example

t=0, x=1

```
Thread1() {
    t = 0;
    x = 1;
    create(Thread2);
    lock(a);
    …
    assert(x != t);
    unlock(a);
}
```

```
Thread2() {
    lock(a);
    t = x;
    …
    x = 2;
    unlock(a);
}
```

Example

```
Thread1() {
    t = 0;
    x = 1;
    create(Thread2);
    lock(a);
    ...
    assert(x != t);
    unlock(a);
}
```

```
Thread2() {
    lock(a);
    t = x;
    ...
    x = 2;
    unlock(a);
}
```

Example

```
Thread1() {
    t = 0;
    x = 1;
    create(Thread2);
    lock(a);
    ...
    assert(x != t);
    unlock(a);
}
```

```
Thread2() {
    lock(a);
    t = x;
    ...
    x = 2;
    unlock(a);
}
```

Example

```
Thread1() {
    t = 0;
    x = 1;
    create(Thread2);
    lock(a);
    ...
    assert(x != t);
    unlock(a);
}
```

t=0, x=1

```
Thread2() {
    lock(a);
    t = x;
    ...
    x = 2;
    unlock(a);
}
```

Assertion is not violated

Example

```
Thread1() {
    t = 0;
    x = 1;
    create(Thread2);
    lock(a);
    …
    assert(x != t);
    unlock(a);
}
```

```
Thread2() {
    lock(a);
    t = x;
    …
    x = 2;
    unlock(a);
}
```

t=1, x=2

Assertion is not violated

Example after a change

```
Thread1() {
    t = 0;
    x = 1;
    create(Thread2);
    ~~lock(a);~~
    …
    assert(x != t);
    ~~unlock(a);~~
}
```

```
Thread2() {
    lock(a);
    t = x;
    …
    x = 2;
    unlock(a);
}
```

Example after a change

Thread1() {
    t = 0;
    x = 1;
    create(Thread2);
    ~~lock(a);~~
    …
    assert(**x** != **t**);
    ~~unlock(a);~~
}

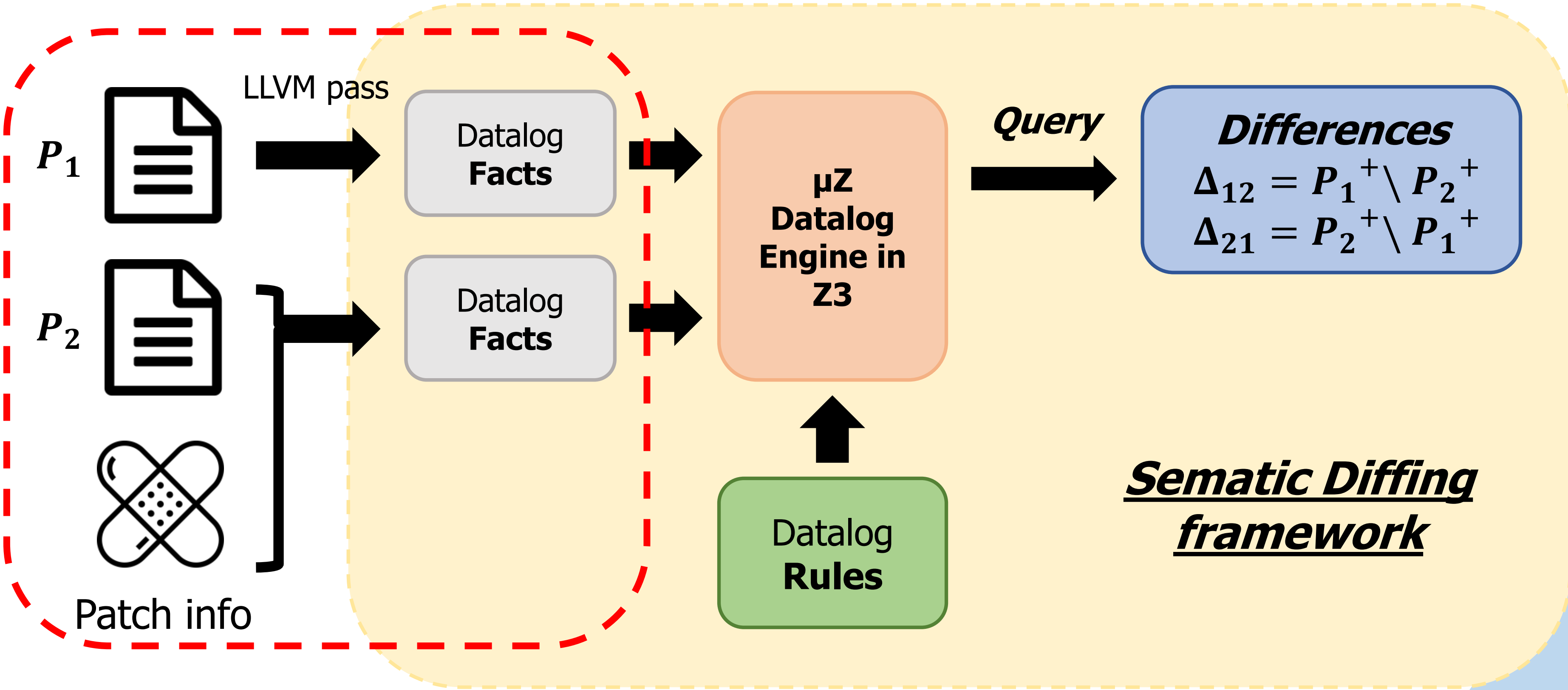
Thread2() {
    lock(a);
    t = **x**;
    …
    x = 2;
    unlock(a);
}

Read-from

Read-from

<u>Assertion is violated</u>

Overview



LLVM pass
P_1
P_2
Datalog Facts
Datalog Facts
Patch info
μZ Datalog Engine in Z3
Query
Differences
$\Delta_{12} = {P_1}^{+} \setminus {P_2}^{+}$
$\Delta_{21} = {P_2}^{+} \setminus {P_1}^{+}$
Datalog Rules
Sematic Diffing framework

Program Analysis in Datalog

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

Evolving concurrent programs → Datalog **facts**

Datalog **Rules**

Datalog Engine

Semantic difference checking between the two programs

What is Datalog?

- Declarative language for deductive database [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

```
Thread1() {
    t = 0;
1:  x = 1;
    create(Thread2);
    lock(a);
    …
2:  assert(x != t);
    unlock(a);
}
```

```
Thread2() {
    lock(a);
3:  t = x;
    …
4:  x = 2;
    unlock(a);
}
```

MustHappenBefore relations

<u>po (s1, s2) -> MustHB (s1, s2)</u>

ThreadOrder(s1, t1, s2, t2) ->

MustHB(s1, s2)

Inferred relations

MustHB: (**{1, 2}, {3, 4}**, {1, 3}, {1, 4})

Datalog Translation

```
Thread1() {
    t = 0;
1: x = 1;
    create(Thread2);
    lock(a);
    …
2: assert(x != t);
    unlock(a);
}
```

```
Thread2() {
    lock(a);
3: t = x;
    …
4: x = 2;
    unlock(a);
}
```

MustHappenBefore relations

po (s1, s2) -> MustHB (s1, s2)

ThreadOrder(s1, t1, s2, t2) -> MustHB(s1, s2)

Inferred relations

MustHB: ({1, 2}, {3, 4}, **{1, 3}, {1, 4})**

Datalog Translation

```
Thread1() {
    t = 0;
1: x = 1;
    create(Thread2);
    lock(a);
    …
2: assert(x != t);
    unlock(a);
}
```

```
Thread2() {
    lock(a);
3: t = x;
    …
4: x = 2;
    unlock(a);
}
```

MayHappenBefore relations

MustHB (s1, s2) -> MayHB (s1, s2)

Not ThreadOrder(s1, t1, s2, t2) -> MayHB(s2, s1)

Inferred relations

MustHB: ({1, 2}, {3, 4}, {1, 3}, {1, 4})

MayHB: (**{1, 2}, {3, 4}, {1, 3}, {1, 4}**, {2, 3}, {2, 4}, {3, 2}, {4, 2})

Datalog Translation

```
Thread1() {
    t = 0;
1: x = 1;
    create(Thread2);
    lock(a);
    …
2: assert(x != t);
    unlock(a);
}
```

```
Thread2() {
    lock(a);
3: t = x;
    …
4: x = 2;
    unlock(a);
}
```

MayHappenBefore relations

MustHB (s1, s2) -> MayHB (s1, s2)

Not ThreadOrder(s1, t1, s2, t2) -> MayHB(s2, s1)

Inferred relations

MustHB: ({1, 2}, {3, 4}, {1, 3}, {1, 4})

MayHB: ({1, 2}, {3, 4}, {1, 3}, {1, 4}, **{2, 3}, {2, 4}, {3, 2}, {4, 2}**)

Datalog Translation

```
Thread1() {                  Thread2() {
    t = 0;                       lock(a);
1: x = 1;                    3: t = x;
    create(Thread2);             ...
    lock(a);                 4: x = 2;
    ...                          unlock(a);
2: assert(x != t);           }
    unlock(a);
}
```

MayReadFrom relations

MayHB (s1, s2) & St(s1) & Ld(s2) -> MayRF (s1, s2)

Inferred relations

MustHB: ({1, 2}, {3, 4}, {1, 3}, {1, 4})

MayHB: ({1, 2}, {3, 4}, {1, 3}, {1, 4}, {2, 3}, {2, 4}, {3, 2}, {4, 2})

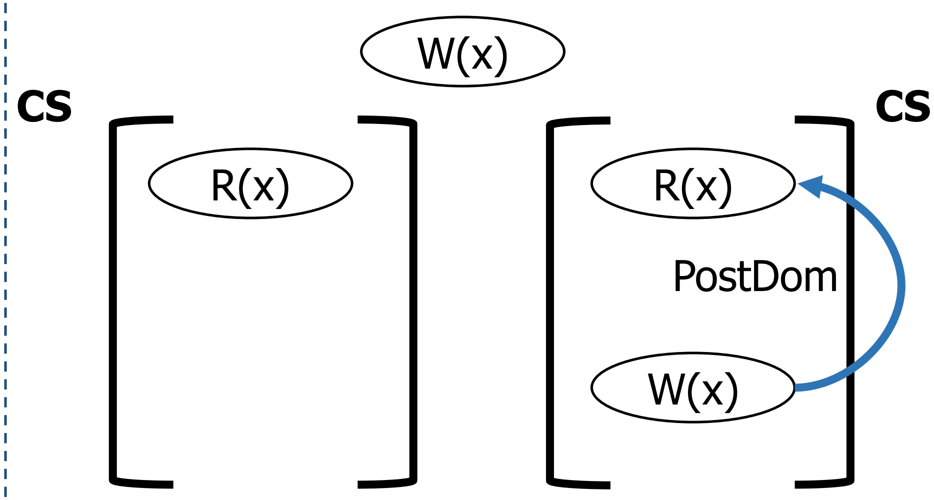
MayRF: ({1, 2}, {1, 3}, {3, 2}, {4, 2})

Datalog Translation

```
Thread1() {
    t = 0;
1: x = 1;
    create(Thread2);
    lock(a);
    …
2: assert(x != t);
    unlock(a);
}
```

```
Thread2() {
    lock(a);
3: t = x;
    …
4: x = 2;
    unlock(a);
}
```

Rank2 relations

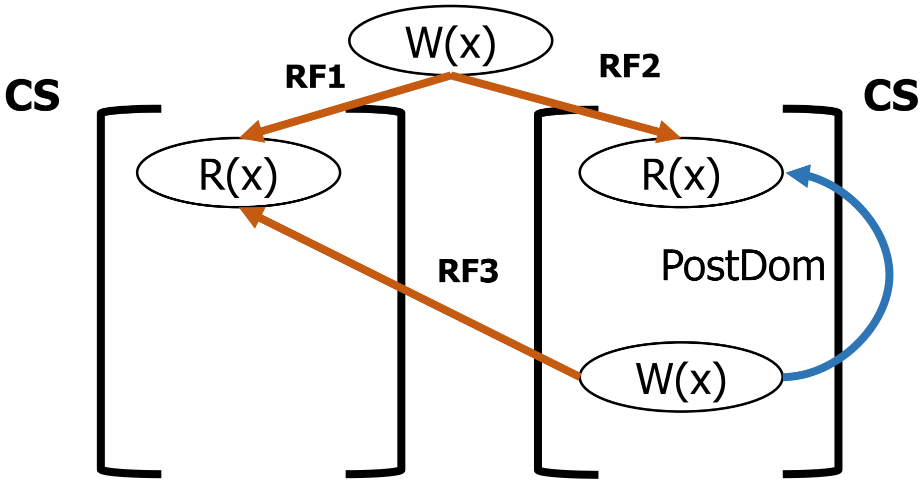


Datalog Translation

```
Thread1() {
    t = 0;
1: x = 1;
    create(Thread2);
    lock(a);
    …
2: assert(x != t);
    unlock(a);
}
```

```
Thread2() {
    lock(a);
3: t = x;
    …
4: x = 2;
    unlock(a);
}
```

Rank2 relations

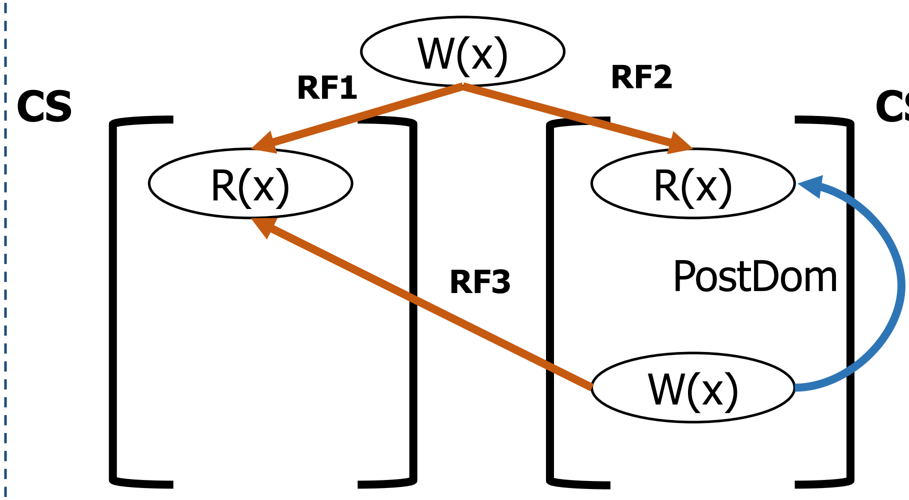


Datalog Translation

```
Thread1() {
    t = 0;
1: x = 1;
    create(Thread2);
    lock(a);
    ...
2: assert(x != t);
    unlock(a);
}
```

```
Thread2() {
    lock(a);
3: t = x;
    ...
4: x = 2;
    unlock(a);
}
```

Rank2 relations



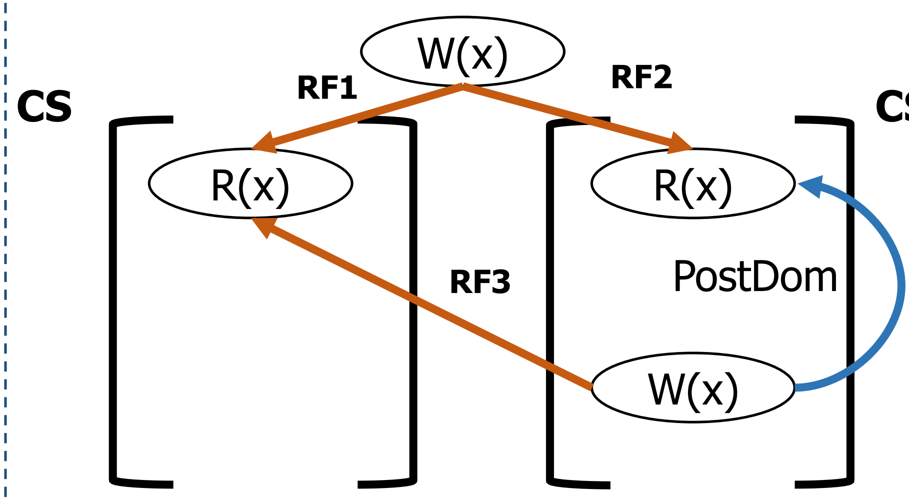
RF1 -> not RF3
RF2 -> not RF1

Datalog Translation

```
Thread1() {
    t = 0;
1: x = 1;
    create(Thread2);
    lock(a);
    …
2: assert(x != t);
    unlock(a);
}
```

```
Thread2() {
    lock(a);
3: t = x;
    …
4: x = 2;
    unlock(a);
}
```

Rank2 relations



RF1 -> not RF3
RF2 -> not RF1

Inferred relations

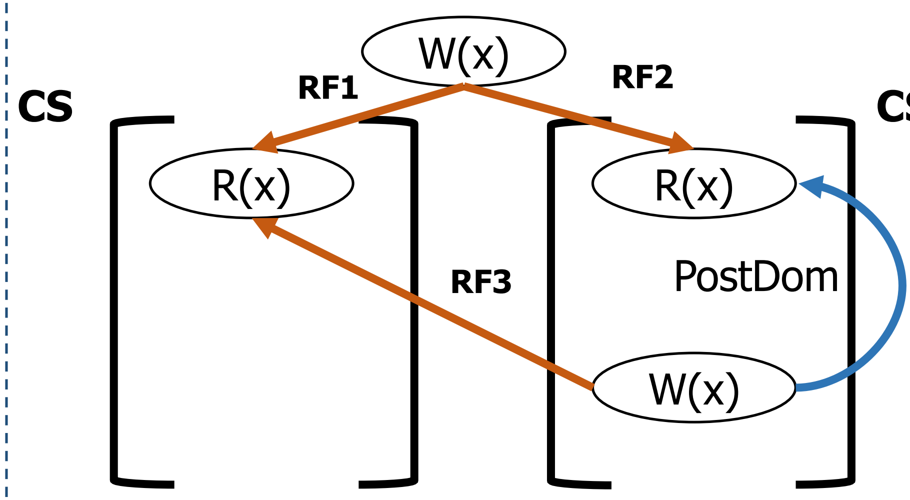
MustHB: ({1, 2}, {3, 4}, {1, 3}, {1, 4})
MayHB: ({1, 2}, {3, 4}, {1, 3}, {1, 4}, {2, 3}, {2, 4}, {3, 2}, {4, 2})
MayRF: ({1, 2}, {1, 3}, {3, 2}, {4, 2})
Rank2: ([{1, 2} -> {1, 3}], [{1, 3} -> {4, 2}])

Datalog Translation

Thread1() {
  t = 0;
  1: x = 1;
  create(Thread2);
  ~~lock(a);~~
  …
  2: assert(x != t);
  ~~unlock(a);~~
}

Thread2() {
  lock(a);
  3: t = x;
  …
  4: x = 2;
  unlock(a);
}

Rank2 relations

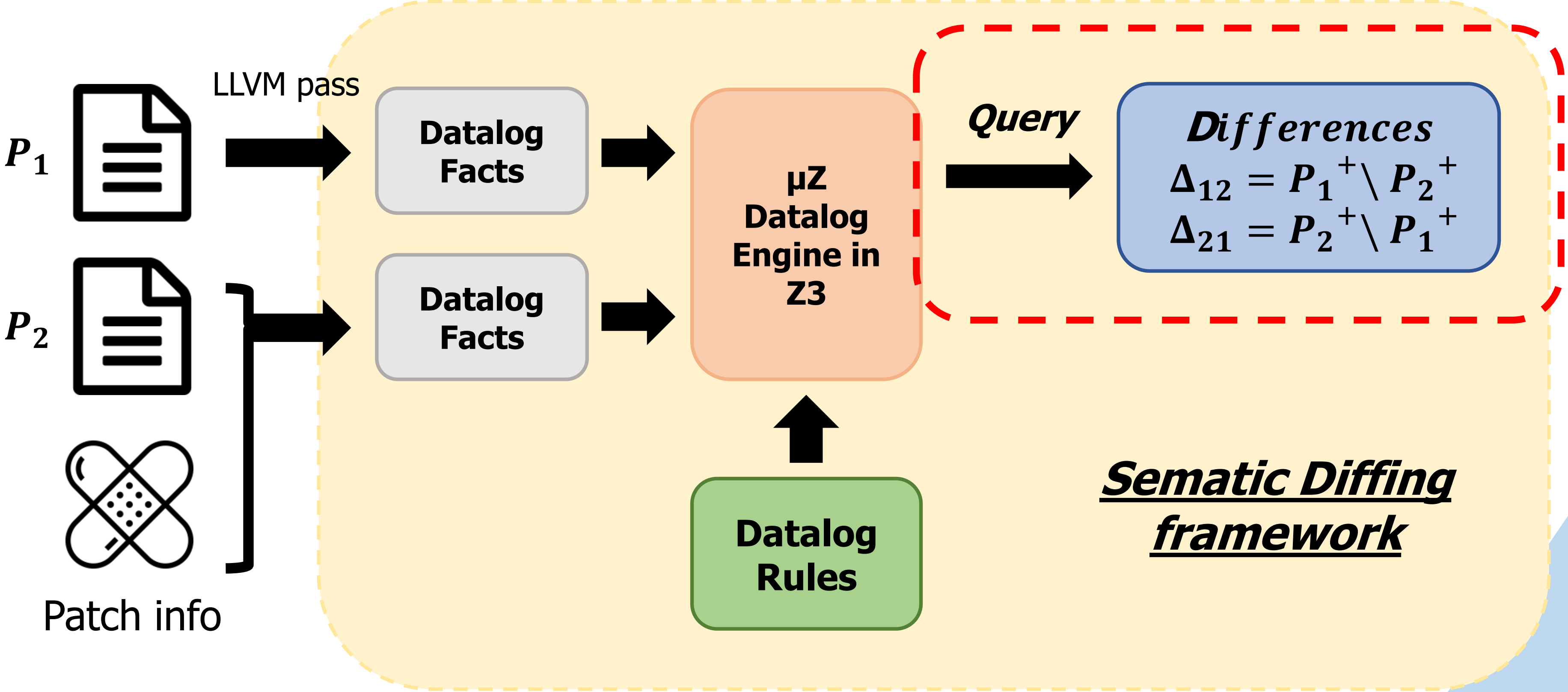


RF1 -> not RF3
RF2 -> not RF1

Inferred relations

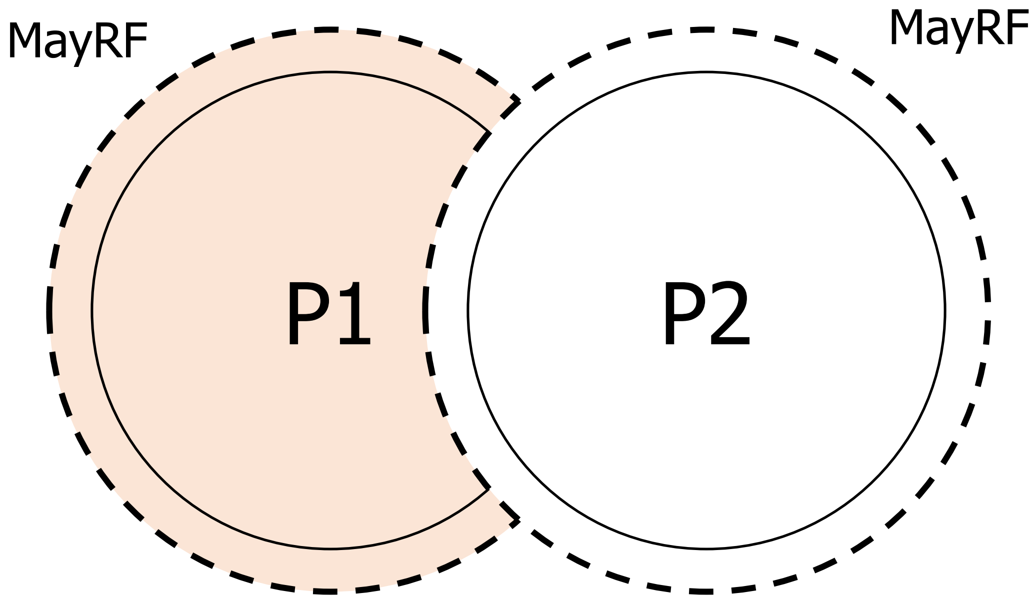
MustHB: ({1, 2}, {3, 4}, {1, 3}, {1, 4})
MayHB: ({1, 2}, {3, 4}, {1, 3}, {1, 4}, {2, 3}, {2, 4}, {3, 2}, {4, 2})
MayRF: ({1, 2}, {1, 3}, {3, 2}, {4, 2})
Rank2: ([{1, 2} -> {1, 3}], [{1, 3} -> {4, 2}], **[{1, 3} -> {1, 2}]**)

Overview



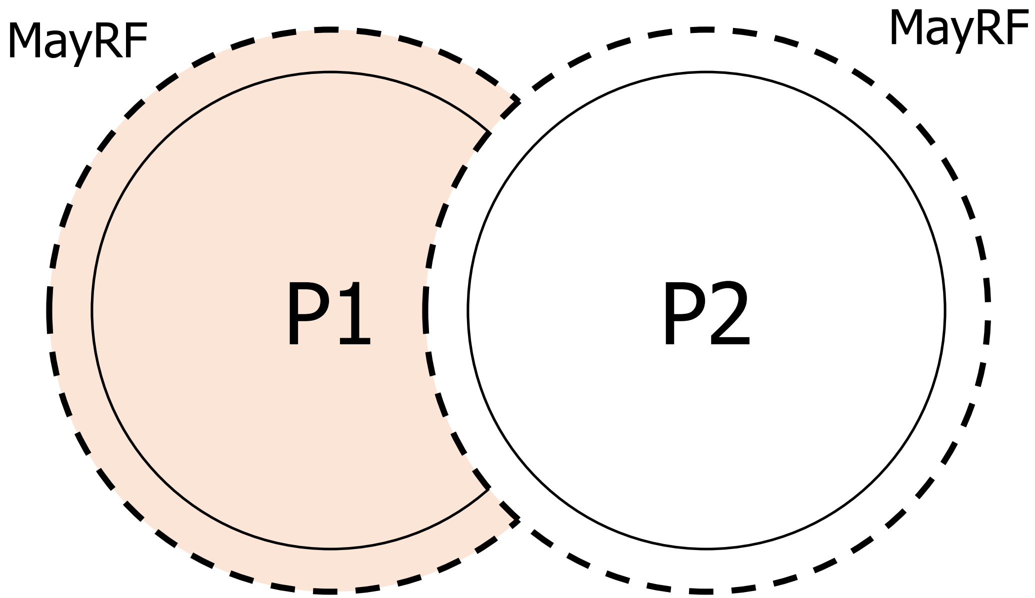
P_1
P_2
Patch info
LLVM pass
Datalog Facts
Datalog Facts
μZ Datalog Engine in Z3
Datalog Rules
Query
$Differences$
$\Delta_{12} = {P_1}^{+} \setminus {P_2}^{+}$
$\Delta_{21} = {P_2}^{+} \setminus {P_1}^{+}$
Sematic Diffing framework

Computing differences



MayRF (s1, s2, p1) & Not MayRF(s1, s2 p2) -> DiffP1-P2 (s1, s2)
MayRF (s1, s2, p2) & Not MayRF(s1, s2 p1) -> DiffP2-P1 (s2, s1)

Computing differences



May be allowed in P1

([{1, 2} -> {1, 3}], [{1, 3} -> {4, 2}])

May be allowed in P2

([{1, 2} -> {1, 3}], [{1, 3} -> {4, 2}], [{1, 3} -> {1, 2}])

Experimental Results 1

The first set	
# of apps	**41**
LOC	**5,546**
Types	**Sync, Th.Order, St.Order, Cond**
Sources	[Bouajjani et al. *SAS 2017*] [Yu & Narayanasamy *ISCA 2009*] [Beyer *TACAS 2015*] [Bloem et al. *FM 2014*] [Lu et al. *ASPLOS 2008*] [Herlihy & Shavit *The Art of Multiprocessor Programming 2008*] [*Open source bug reports*]

Comparison

- Bounded Model Checking based approach

[Bouajjani et al. *SAS 2017*]

Experimental Results 1

The first set	
Execution time of BMC-based approach	***<u>> 3 hours</u>***
Execution time of our approach (NEW)	***<u>15.57 seconds</u>***
# of differences our approach found	**402 dataflow edges (*<u>All valid</u>*)**

Experimental Results 2

The second set	
# of apps	**6**
LOC	**7,986**
Types	**Th.Order, Cond**
Sources	[Yang et al. *U. of Utah 2008*] [Yu & Narayanasamy *ISCA 2009*]
BMC-based approach	**Not available**
Execution time of our approach	***140.28 seconds***
# of differences our approach found	**72 (*All valid*)**

Conclusions

- Proposed a *Datalog based* static analysis for semantic diffing concurrent programs
- *Practically accurate* for identifying differences in thread synchronization
- Significant improvement in *scalability* especially for large programs

Thank you!

https://github.com/chunghasung/EC-Diff