WHEN THREADS MEET EVENTS:
EFFICIENT AND PRECISE STATIC RACE DETECTION WITH ORIGINS

Bozhen Liu*, Peiming Liu*, Yanze Li,

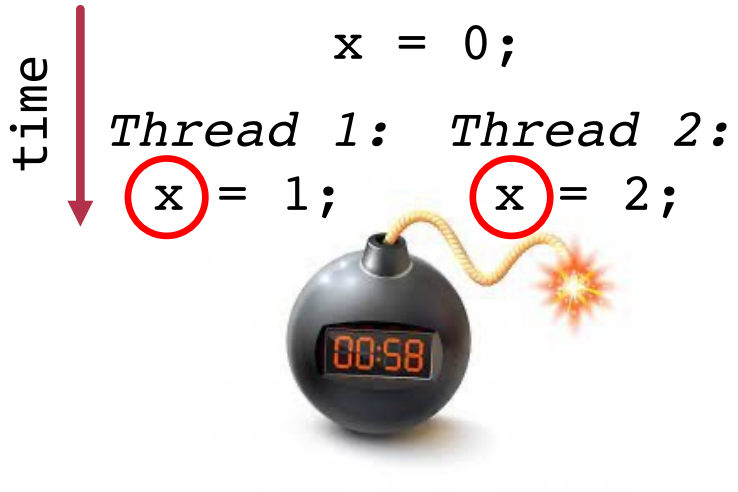
Chia-Che Tsai, Dilma Da Silva, Jeff Huang

TEXAS A&M UNIVERSITY

*Bozhen Liu and Peiming Liu contributed equally to this work.

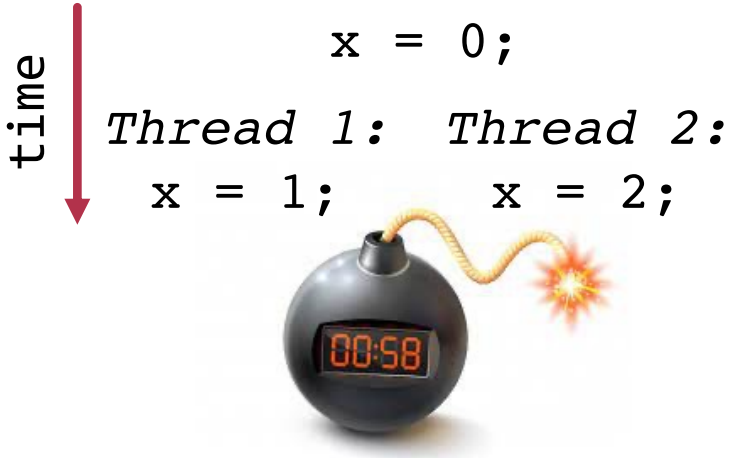
MOTIVATIONS

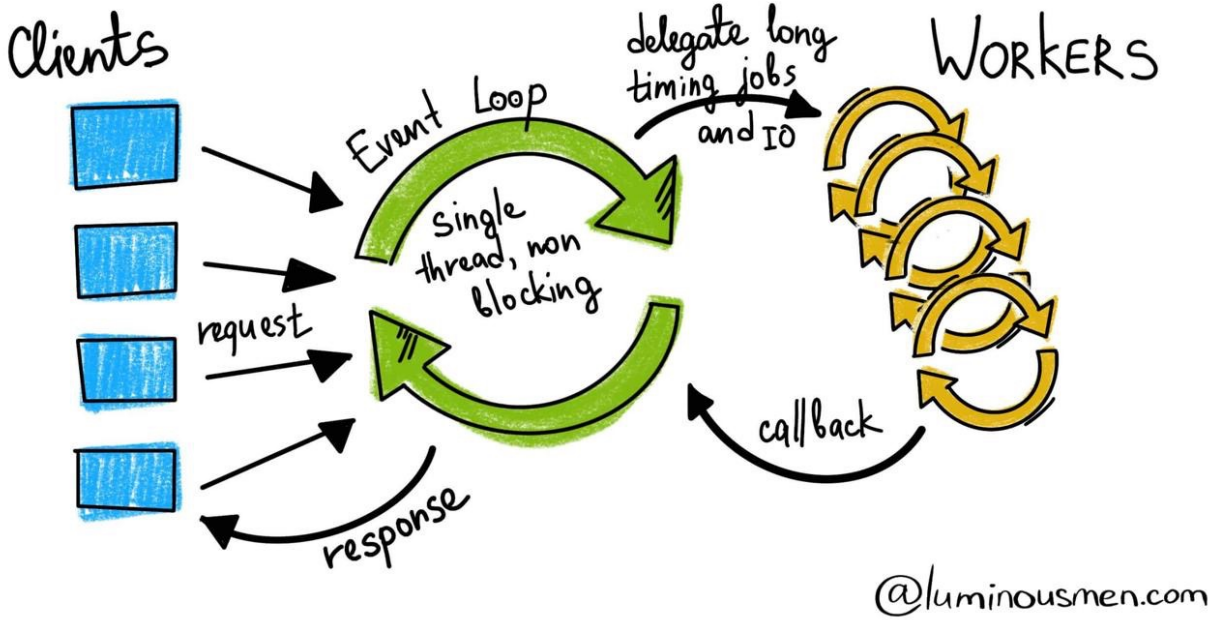
- The notorious data races
  - Easy to introduce
  - Difficult to detect and debug
  - Infer RacerD (commercial)
  - RacerX, RELAY, FastTrack, RVPredict (research)



MOTIVATIONS

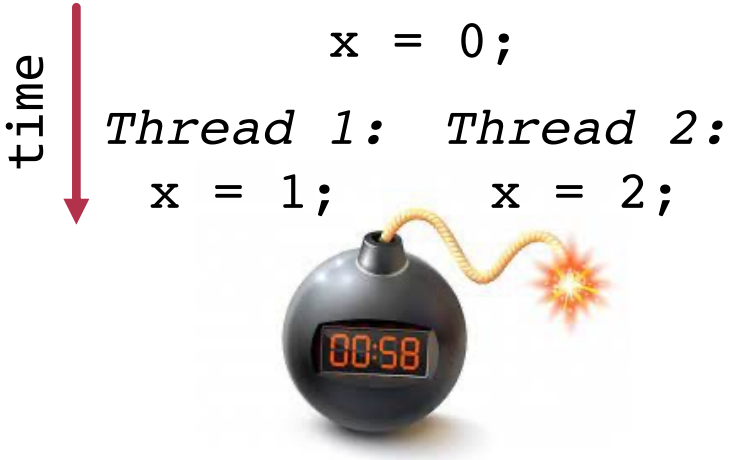
- The notorious data races
- Finding races is even Challenging now
  - Threads + Events

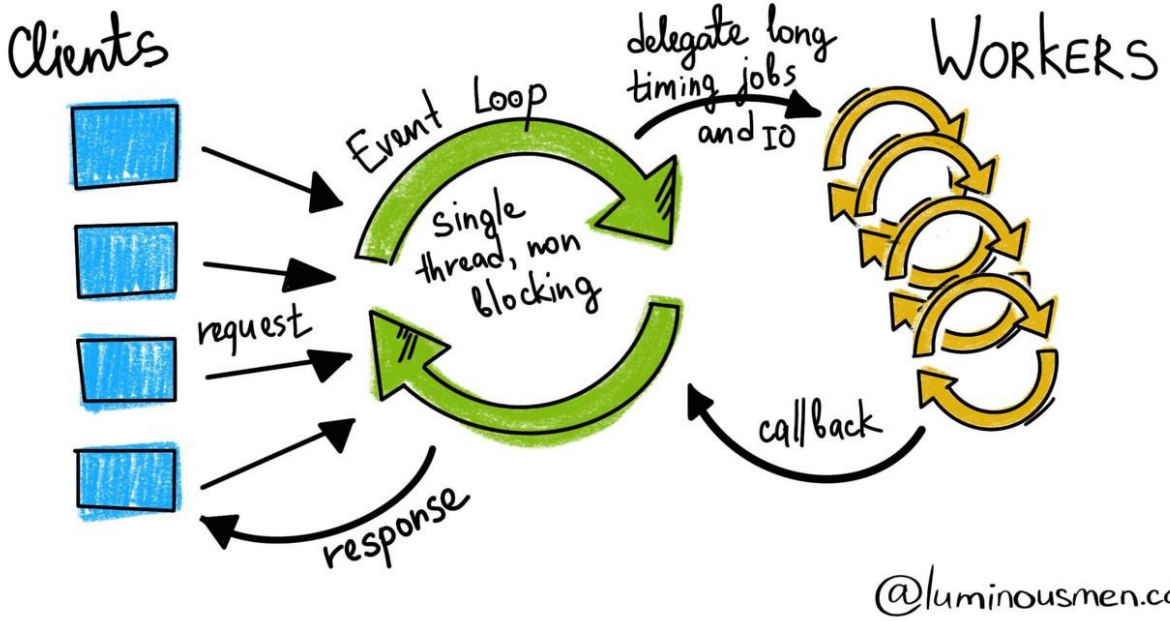




MOTIVATIONS

- The notorious data races:
- Finding races is even Challenging now
  - Threads + Events
  - Large scales of code bases





Java App	LOC	C/C++ App	LOC
Lucene	655,523	Memcached	19,038
HBase	738,495	Redis	148,183
Hadoop	1,087,799	Sqlite3	523,831

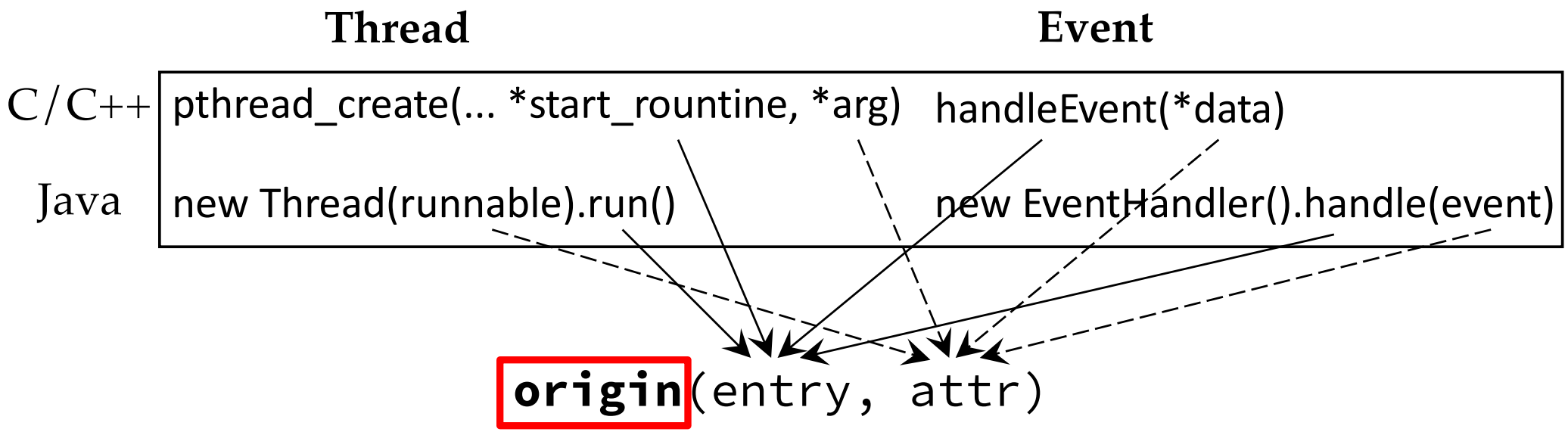
O2: THE PRACTICAL SOLUTION

- Scalable and Efficient
  - 8 min for millions lines of code: 70x faster (up to 568x)
  - Comparable performance with RacerD (V1.0.0)
- Precise
  - Reduce false positives by 77% (up to 99%)
  - 4.33x fewer races than RacerD (V1.0.0)
- New Races Detected by O2
  - more than 40 unique previously unknown races
  - confirmed or fixed by developers

App	#New Races
Linux	6
TDengine	6
Redis/RedisGraph	5
OVS	3
cpqueue	7
mrlock	5
Memcached	3
Firefox Focus	2
Zookeeper	1
HBase	1
Tomcat	1

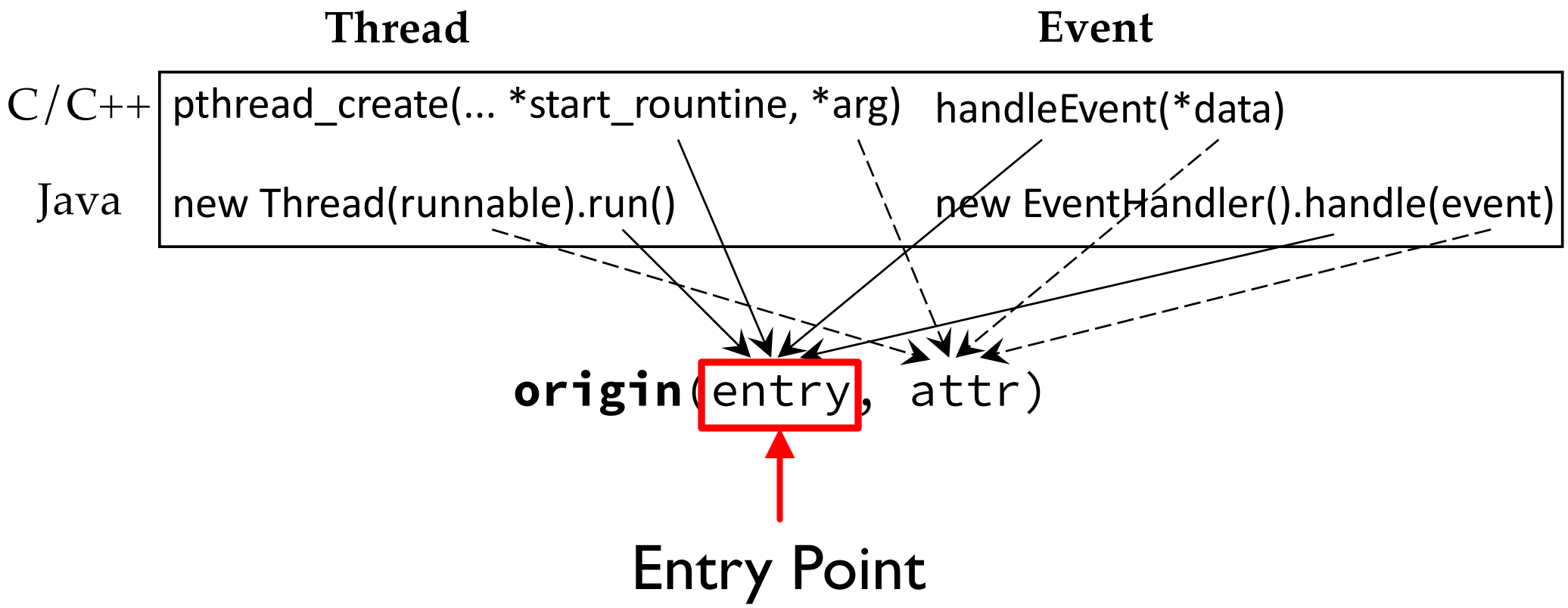
O2: POWERED BY ORIGINS

- Origins



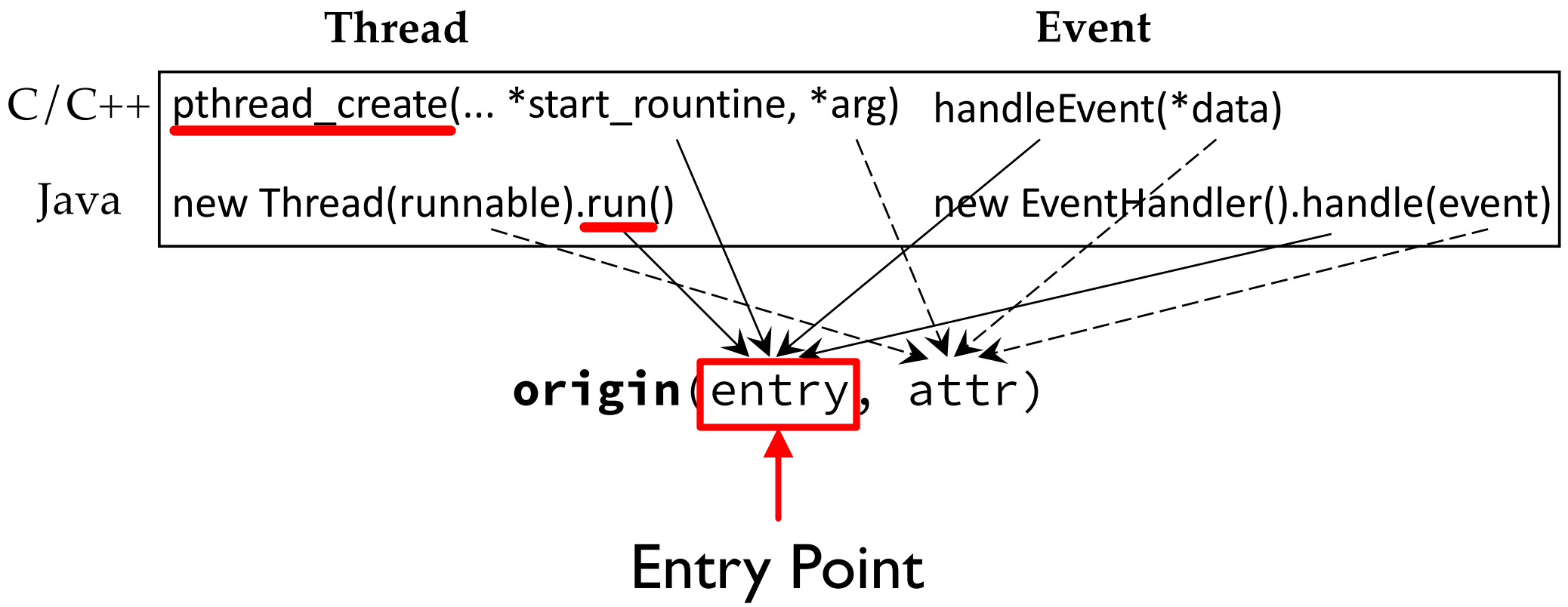
O2: POWERED BY ORIGINS

- Origins



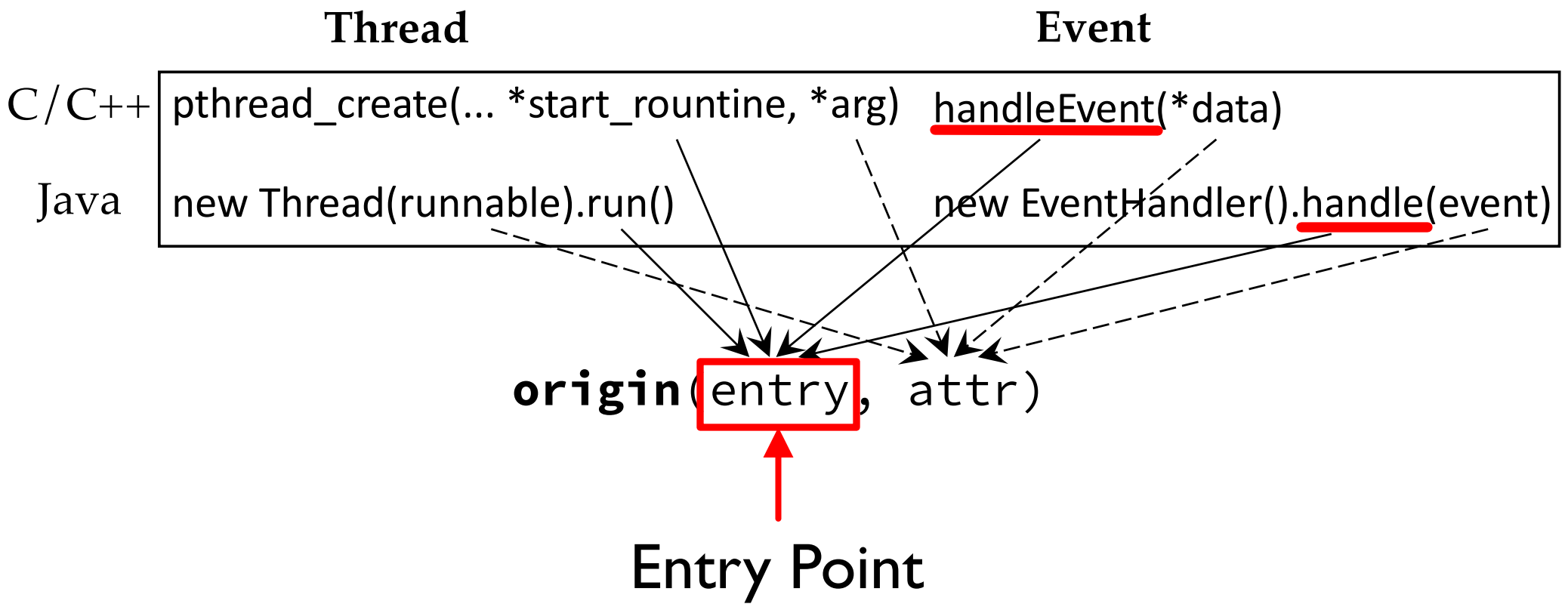
O2: POWERED BY ORIGINS

- Origins



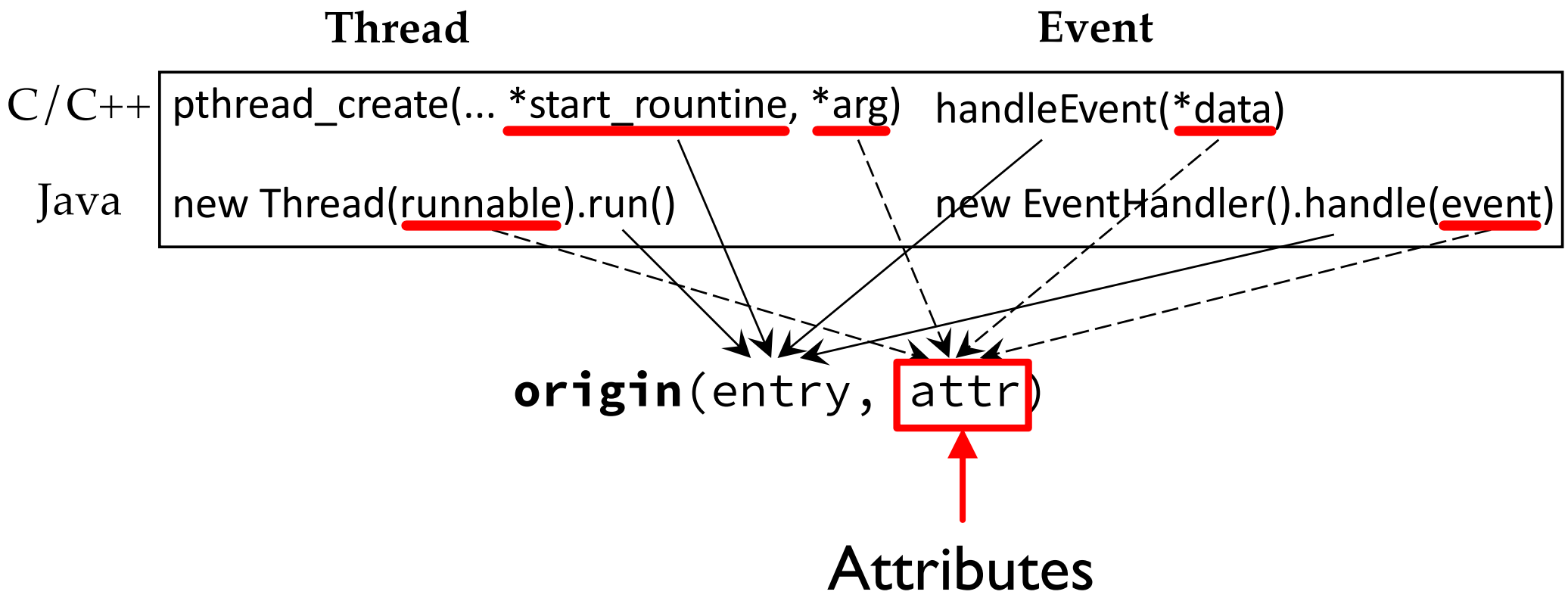
O2: POWERED BY ORIGINS

- Origins



O2: POWERED BY ORIGINS

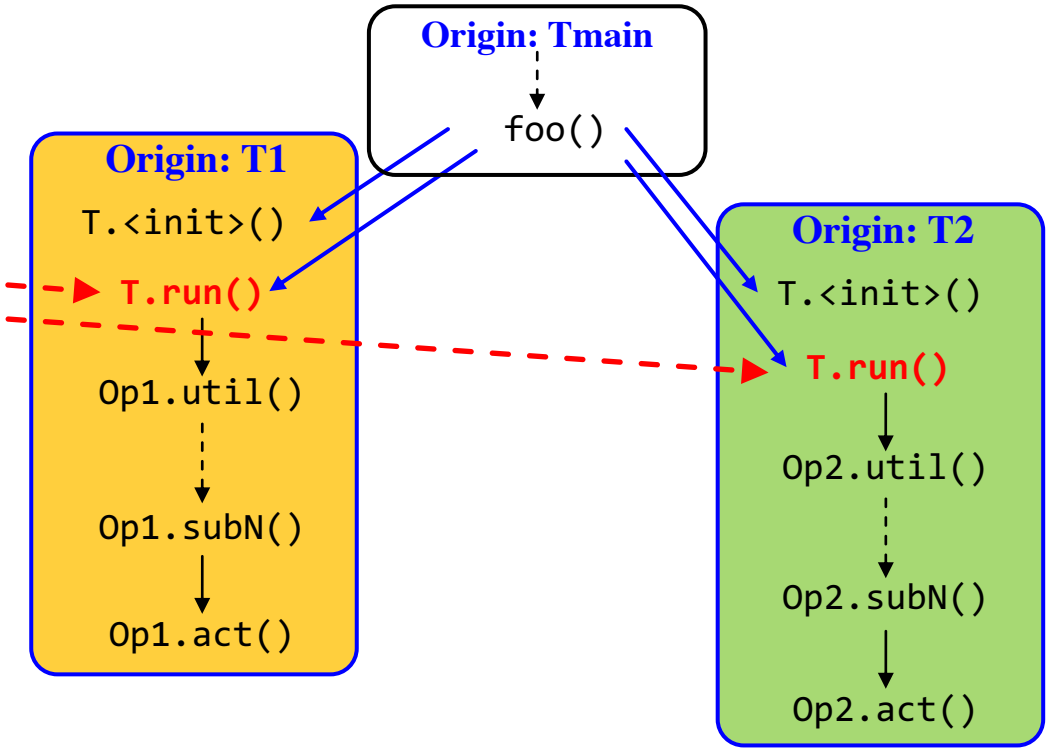
- Origins



O2: EXAMPLE

```
1  public void foo(){//main thread
2    Obj s = new Obj();//o1
3    Op op1 = new Op1();//o2
4    Op op2 = new Op2();//o3
5    new T(s, op1).run();//o4
6    new T(s, op2).run();//o5
7  }
8  public class T extends Thread {
9    Obj f; Op op;//super class of Op1 and Op2
10   public T(Obj a, Op b){
11     f = a; op = b; }
12   public void run(){
13     op.util(f, new Obj());//o6
14   }
15 }
16 void util(Obj x, Obj y){
17   sub1(x, y); ...
18 }
19 void sub1(Obj x, Obj y){
20   sub2(x, y); ...
21 }          ...
22 void subN(Obj x, Obj y){
23    y.doTask();
24    act(x, y);
25    x.f = new Obj();  ...}
```

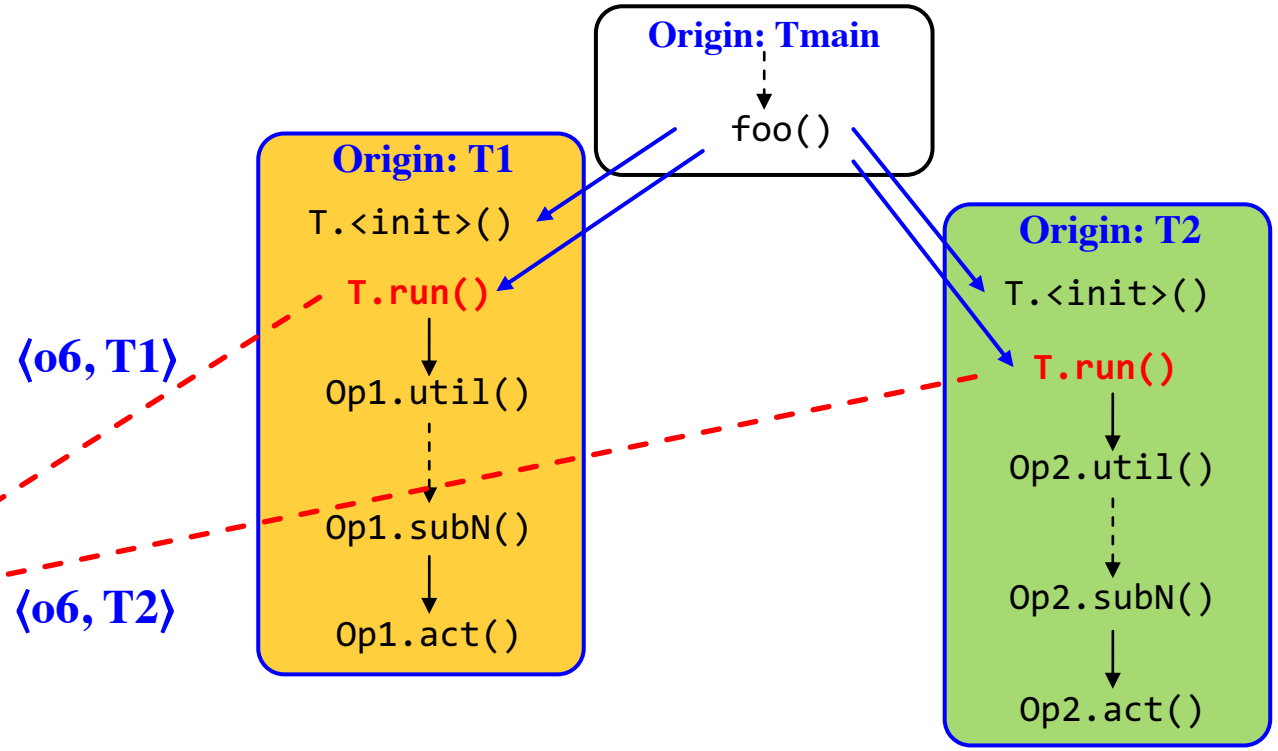
Origin-Sensitive Pointer Analysis: Call Graph



O2: EXAMPLE

```
1  public void foo(){//main thread
2    Obj s = new Obj();//o1
3    Op op1 = new Op1();//o2
4    Op op2 = new Op2();//o3
5    new T(s, op1).run;//o4
6    new T(s, op2).run;//o5
7  }
8  public class T extends Thread {
9    Obj f; Op op;//super class
10   public T(Obj a, Op b){
11     f = a; op = b; }
12   public void run(){
13     op.util(f, new Obj());//o6
14   }
15 }
16 void util(Obj x, Obj y){
17   sub1(x, y); ...
18 }
19 void sub1(Obj x, Obj y){
20   sub2(x, y); ...
21 }          ...
22 void subN(Obj x, Obj y){
23    y.doTask();
24    act(x, y);
25    x.f = new Obj();  ...}
```

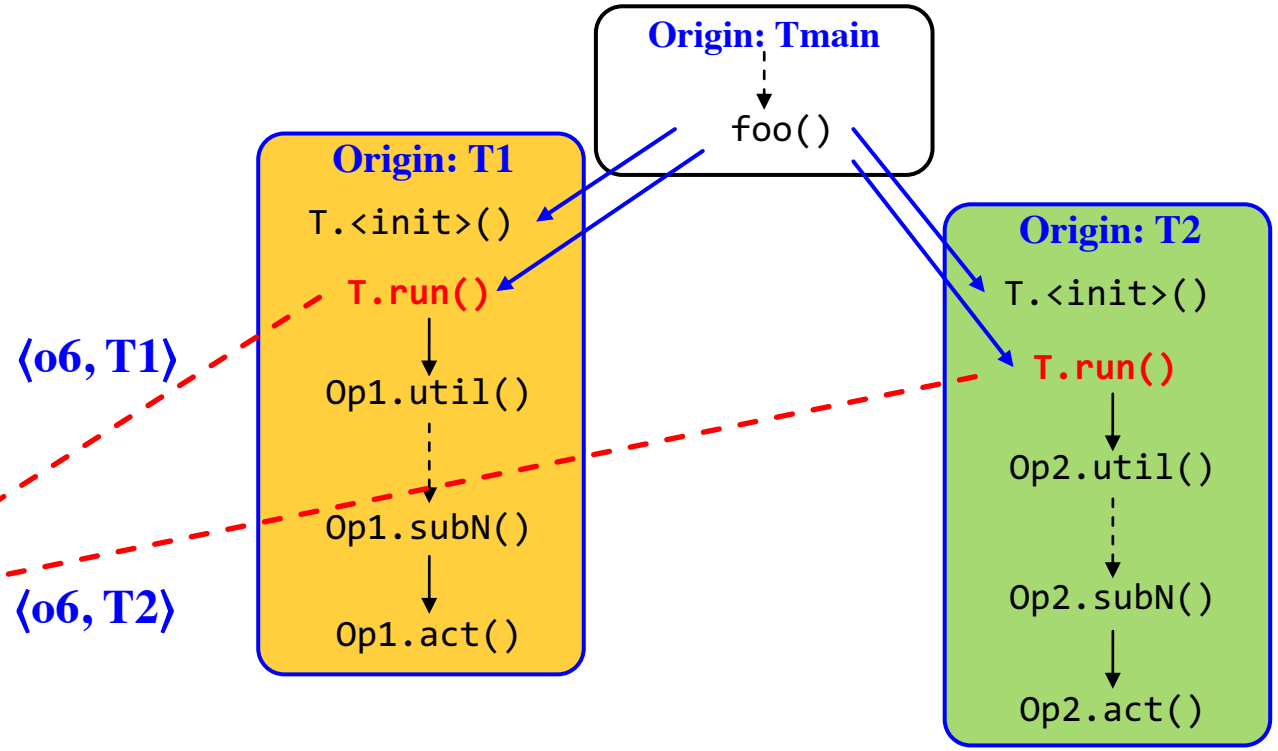
Origin-Sensitive Pointer Analysis: Call Graph



O2: EXAMPLE

```
1  public void foo(){//main thread
2    Obj s = new Obj();//o1
3    Op op1 = new Op1();//o2
4    Op op2 = new Op2();//o3
5    new T(s, op1).run;//o4
6    new T(s, op2).run;//o5
7  }
8  public class T extends Thread {
9    Obj f; Op op;//super class
10   public T(Obj a, Op b){
11     f = a; op = b; }
12   public void run(){
13     op.util(f, new Obj());//o6
14   }
15 }
16 void util(Obj x, Obj y){
17   sub1(x, y); ...
18 }
19 void sub1(Obj x, Obj y){
20   sub2(x, y); ...
21 }          ...
22 void subN(Obj x, Obj y){
23    y.doTask();
24    act(x, y);
25    x.f = new Obj();  ...}
```

Origin-Sensitive Pointer Analysis: Call Graph



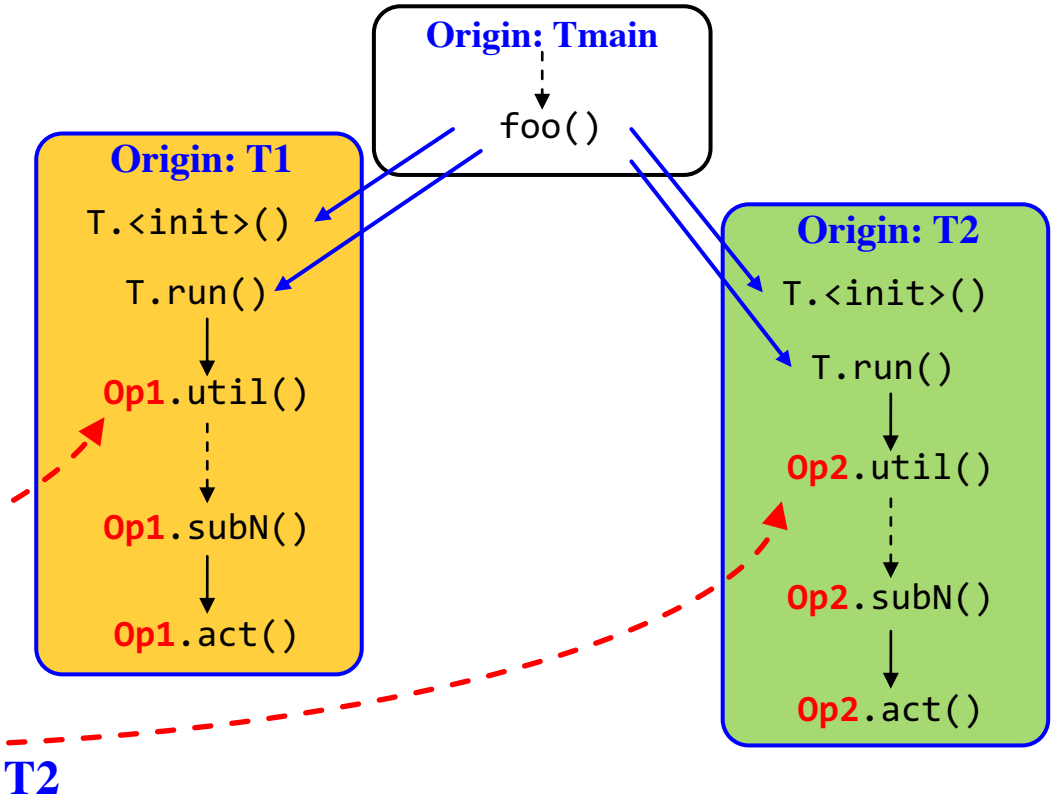
O2: EXAMPLE

```
1  public void foo(){//main thread
2    Obj s = new Obj();//o1
3    Op op1 = new Op1();//o2
4    Op op2 = new Op2();//o3
5    new T(s, op1).run();//o4
6    new T(s, op2).run();//o5
7  }
8  public class T extends Thread {
9    Obj f; Op op;//super class of Op1 and Op2
10   public T(Obj a, Op b){
11     f = a; op = b; }
12   public void run(){
13     op.util(f, new Obj());//o6
14   }
15 }
16 void util(Obj x, Obj y){
17   sub1(x, y); ...
18 }
19 void sub1(Obj x, Obj y){
20   sub2(x, y); ...
21 }          ...
22 void subN(Obj x, Obj y){
23    y.doTask();
24    act(x, y);
25    x.f = new Obj();  ...}
```

T1

T2

Origin-Sensitive Pointer Analysis: Call Graph



O2: EXAMPLE

```
1  public void foo(){//main thread
2    Obj s = new Obj();//o1
3    Op op1 = new Op1();//o2
4    Op op2 = new Op2();//o3
5    new T(s, op1).run();//o4
6    new T(s, op2).run();//o5
7  }
8  public class T extends Thread {
9    Obj f; Op op;//super class of Op1 and Op2
10   public T(Obj a, Op b){
11     f = a; op = b; }
12   public void run(){
13     op.util(f, new Obj());//o6
14   }
15 }
16 void util(Obj x, Obj y){
17   sub1(x, y); ...
18 }
19 void sub1(Obj x, Obj y){
20   sub2(x, y); ...
21 }          ...
22 void subN(Obj x, Obj y){
23    y.doTask();
24    act(x, y);
25    x.f = new Obj();  ...}
```

Origin-Sensitive Pointer Analysis

Origin-Sharing Analysis

Objects	Accessed by	Meaning
⟨o6,T1⟩	T1	Local to Origin T1
⟨o6,T2⟩	T2	Local to Origin T2
⟨o1,Tmain⟩	T1&T2	Allocated by Origin Tmain, Shared by Origins T1and T2

+

Static Data Race Detection

Race 1:

Thread T1:
/example.java

```
  23   y.doTask();
  24   act(x, y);
> 25   x.f = new Obj();
```

Thread T2:
/example.java

```
  23   y.doTask();
  24   act(x, y);
> 25   x.f = new Obj();
```

THANK YOU!

MORE DETAILS

- Full Presentation
- O2 is available
  - https://coderrect.com/download/