Comprehensive synchronization elimination for Java

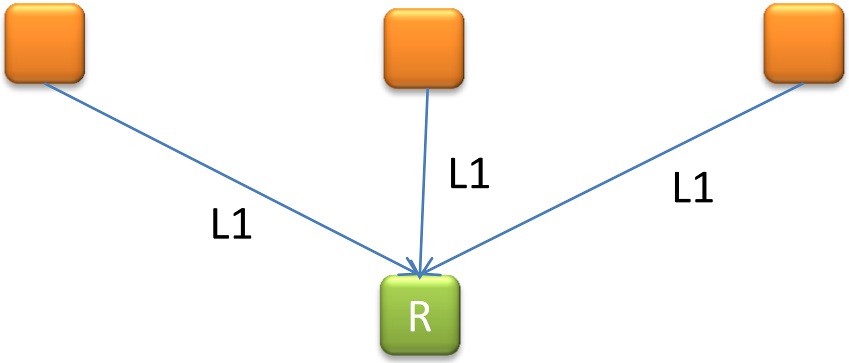
- Jonathan Aldrich, Emin Gün Sirer, Craig Chambers, Susan J. Eggers
- Presented by: Erik Jonsson

```
class Account {

  int balance;

  synchronized void withdraw(int amount) {
    balance = balance - amount;
}

  synchronized void deposit(int amount) {
    balance = balance + amount;
    }
}
```

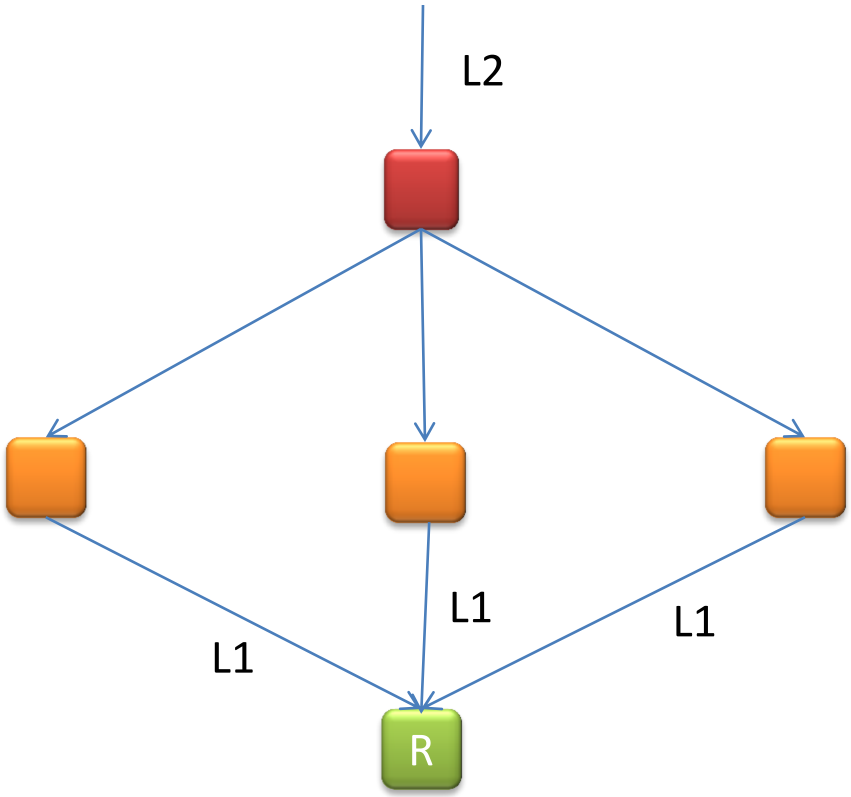
```
void transfer(Account acc1, Account acc2, int amount) {
  synchronized(acc1) {
    synchronized(acc2) {

      acc1.withdraw(amount);
      // Do something that requires a lock on both accounts!
      acc2.deposit(amount);

    }
  }
}
```

- Goal: Removing unnecessary synchronization
- Solution: Whole-program static analyses
- Thread-aware
- Three types:
  - Reentrant locks
  - Enclosed locks
  - Thread-local locks

Enclosing locks



Enclosing locks

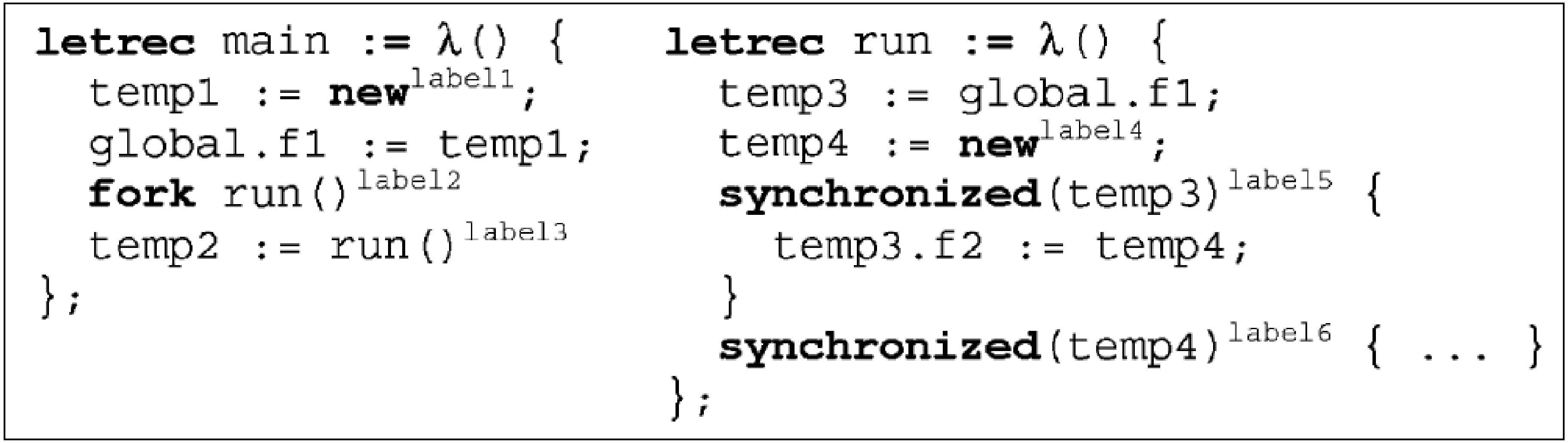


Thread-local locks

- The protected object is not accessed by more than one thread
- Multi(o): The object o may be accessed by multiple threads
- Most studied type of unnecessary synchronization

aliases(id_1, id_2)

- aliases(id_1, id_2)
- aliases(f_1, f_2)
- ref(base, f, o)
- ref(id, o)
- immutable(f)
- called(p, $label_q$)
- creator(o)
- synch_aliases(label)
- synch_keys(label)
- lookup(id_f)



```
letrec main := λ() {
  temp1 := new^label1;
  global.f1 := temp1;
  fork run()^label2
  temp2 := run()^label3
};
```

```
letrec run := λ() {
  temp3 := global.f1;
  temp4 := new^label4;
  synchronized(temp3)^label5 {
    temp3.f2 := temp4;
  }
  synchronized(temp4)^label6 { ... }
};
```

$$\frac{read(f,t_1)\quad written(f,t_2)}{multi(f)}\quad(t_1 \neq t_2)$$

$$\frac{eval(\texttt{S}_1\ \texttt{;}\ \texttt{S}_2, t)}{eval(\texttt{S}_1, t)\quad eval(\texttt{S}_2, t)}$$

$$\frac{eval(\texttt{id}_0\ \texttt{:=}\ \texttt{id}_\texttt{F}(\texttt{id}_1\texttt{..}\texttt{id}_\texttt{n})^{\texttt{label}}, t)}{eval(\texttt{S}, t)}\ (\texttt{letrec id}_\texttt{F}\ \texttt{:=}\ \lambda(\texttt{id}_1\texttt{..}\texttt{id}_\texttt{n})\ \{\ \texttt{S}\ \})$$

$$\frac{eval(\texttt{fork id}_\texttt{F}()^{\texttt{label}}, t)}{eval(\texttt{S}, \texttt{id}_\texttt{F})}\ (\texttt{letrec id}_\texttt{F}\ \texttt{:=}\ \lambda()\ \{\ \texttt{S}\ \})$$

$$\frac{eval(\texttt{id}_1\texttt{.f}\ \texttt{:=}\ \texttt{id}_2, t)}{written(f, t)}$$

$$\frac{eval(\texttt{id}_1\ \texttt{:=}\ \texttt{id}_2\texttt{.f}, t)}{read(f, t)}$$

$$\frac{called(p, label_q)\quad called(p, label_r)}{multi(p)}\ (label_q \neq label_r)$$

$$\frac{multi(p)\quad called(q, label_p)}{multi(q)}$$

$$\frac{read(f, t_1)\quad written(f, t_2)}{multi(f)}\ (t_1 \neq t_2)$$

$$\frac{read(f, t)\quad written(f, t)\quad multi(t)}{multi(f)}$$

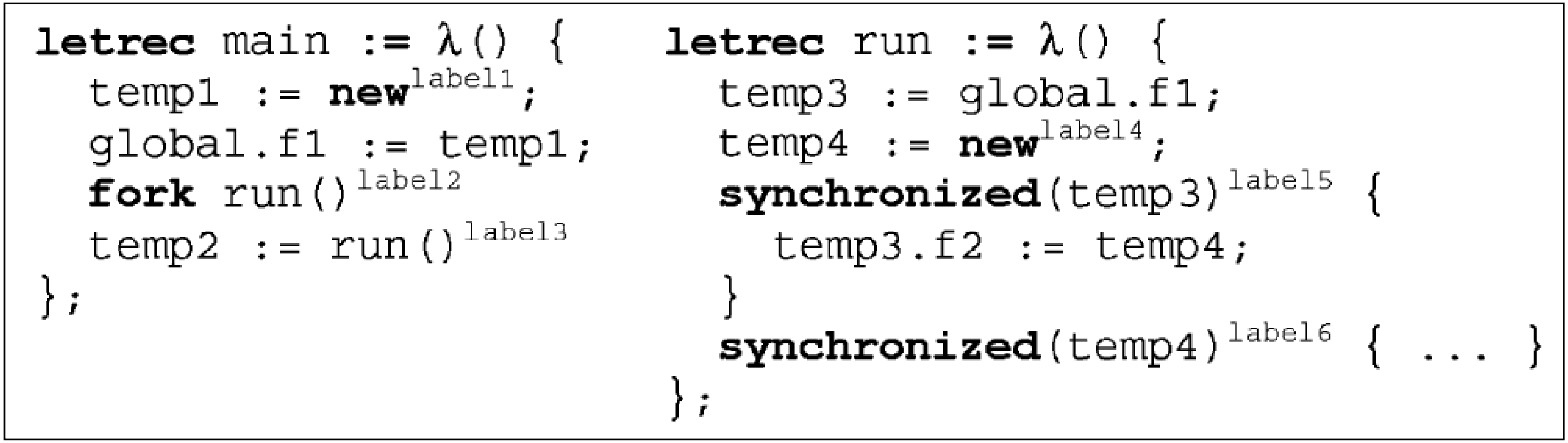
$$\frac{ref(\texttt{global}, f, o)\quad multi(f)}{multi(o)}$$

$$\frac{multi(b)\quad ref(b, f, o)\quad multi(f)}{multi(o)}$$

- eval(s, t)
- read(f, t)
- written(f, t)
- multi(p)

- multi(t)
- multi(f)
- multi(o)

- eval(s, t)
- read(f, t)
- written(f, t)
- multi(p)
- multi(t)
- multi(f)
- multi(o)



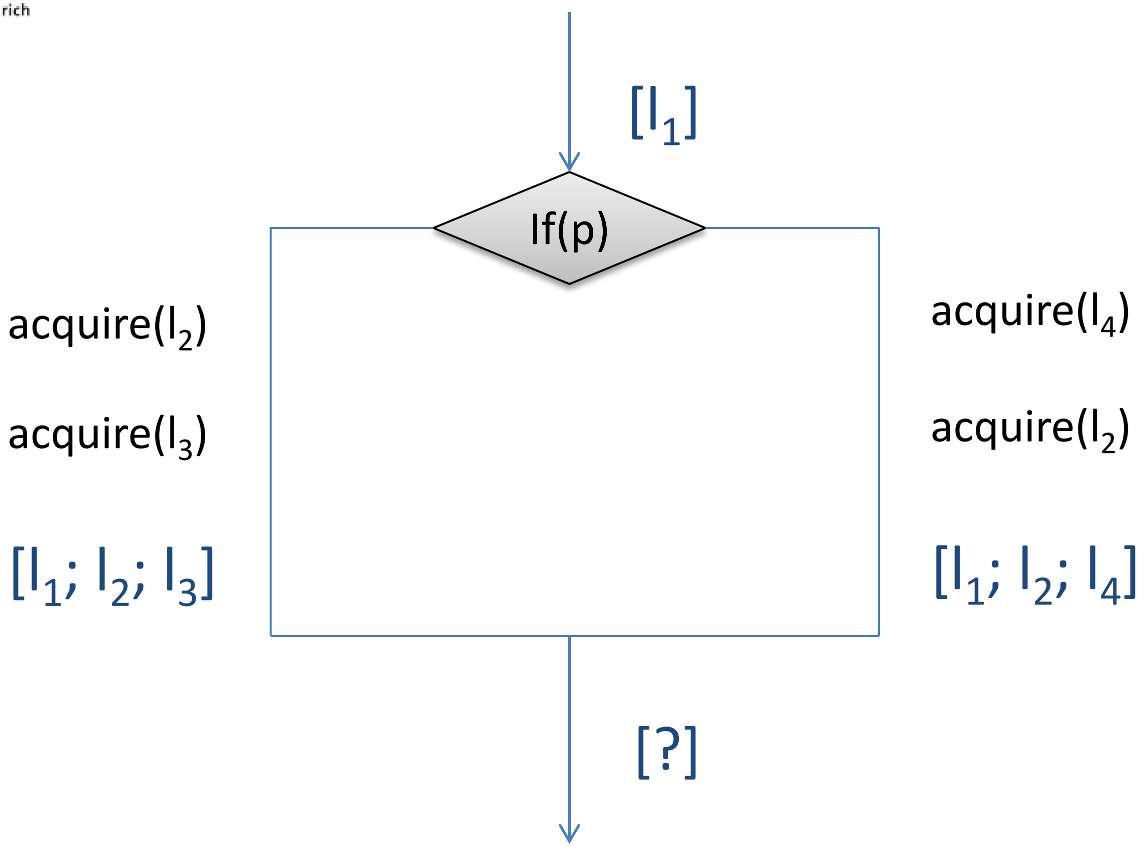
```
letrec main := λ() {            letrec run := λ() {
  temp1 := new^label1;            temp3 := global.f1;
  global.f1 := temp1;             temp4 := new^label4;
  fork run()^label2               synchronized(temp3)^label5 {
  temp2 := run()^label3             temp3.f2 := temp4;
};                                }
                                  synchronized(temp4)^label6 { ... }
                                };
```

What we did:

- Rely on previous analyses
- Apply inference rules
- See if we can infer multi(o)
- If not, locks can be safely removed

Enclosed locks

- Want to compute which locks are held at each program point
- Hard to do precisely
- Construct a graph



[l_1]

If(p)

acquire(l_2)

acquire(l_3)

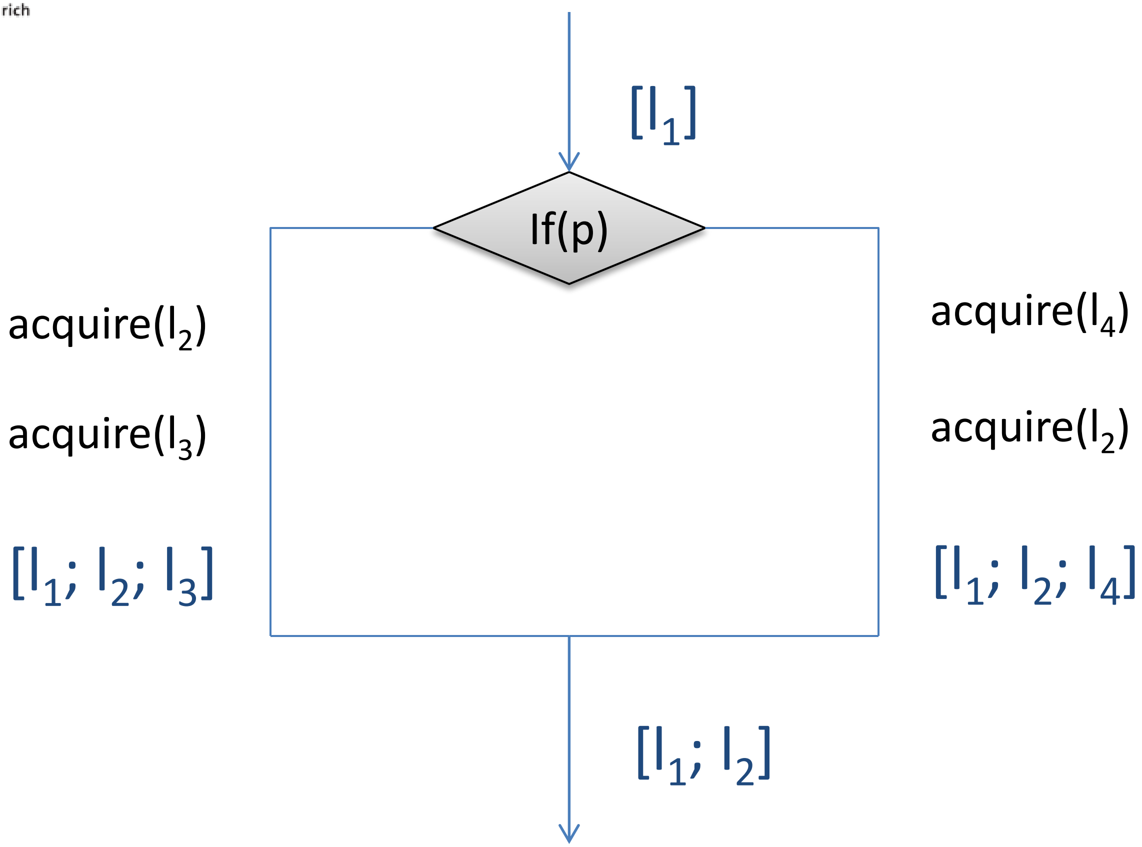
[l_1; l_2; l_3]

acquire(l_4)

acquire(l_2)

[l_1; l_2; l_4]

[?]



$[l_1]$

If(p)

acquire(l_2)

acquire(l_3)

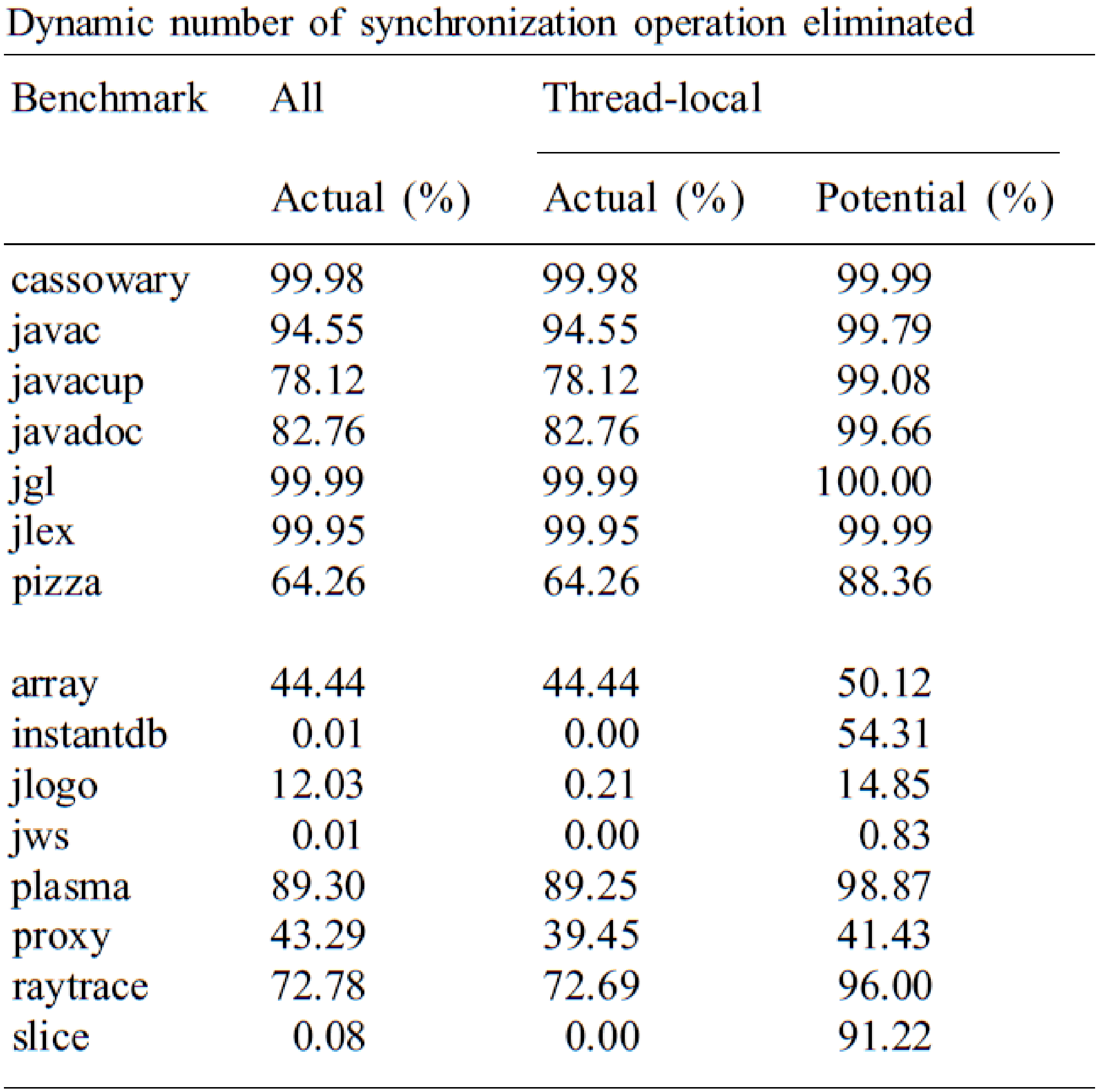
$[l_1; l_2; l_3]$

acquire(l_4)

acquire(l_2)

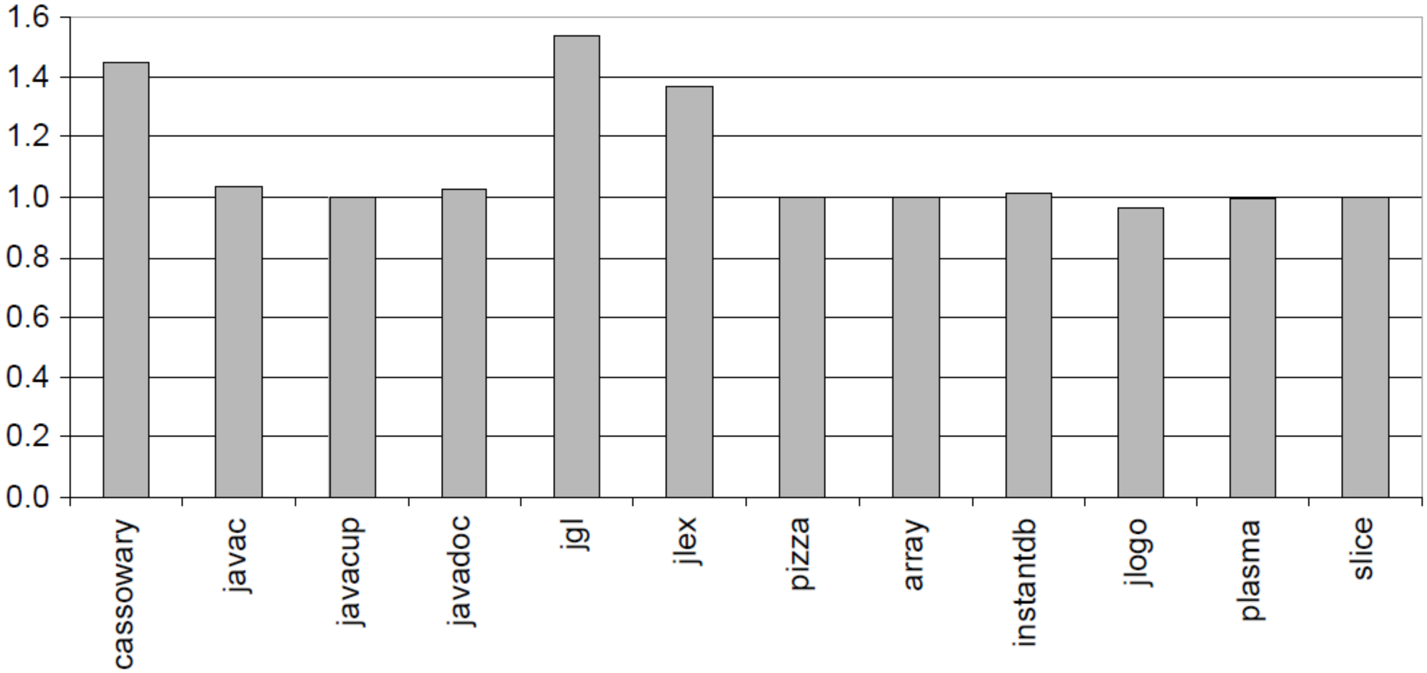
$[l_1; l_2; l_4]$

$[l_1; l_2]$



Dynamic number of synchronization operation eliminated

Benchmark	All	Thread-local	
	Actual (%)	Actual (%)	Potential (%)
cassowary	99.98	99.98	99.99
javac	94.55	94.55	99.79
javacup	78.12	78.12	99.08
javadoc	82.76	82.76	99.66
jgl	99.99	99.99	100.00
jlex	99.95	99.95	99.99
pizza	64.26	64.26	88.36
array	44.44	44.44	50.12
instantdb	0.01	0.00	54.31
jlogo	12.03	0.21	14.85
jws	0.01	0.00	0.83
plasma	89.30	89.25	98.87
proxy	43.29	39.45	41.43
raytrace	72.78	72.69	96.00
slice	0.08	0.00	91.22


1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
cassowary
javac
javacup
javadoc
jgl
jlex
pizza
array
instantdb
jlogo
plasma
slice

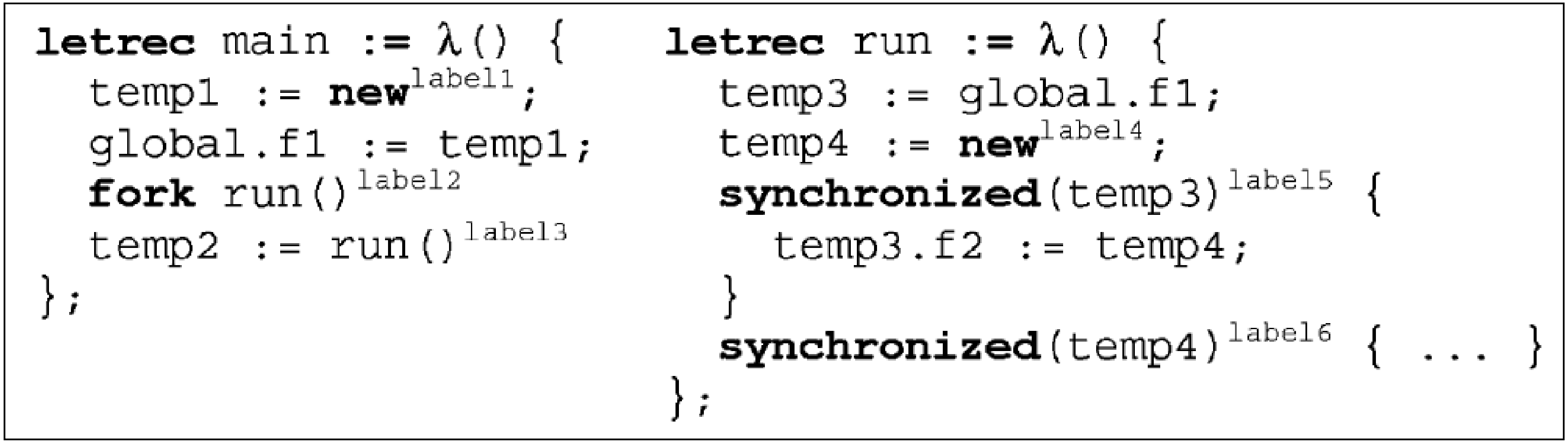
A different programming model

- Lock EVERYTHING!
- Use static analyses to remove unnecessary synchronization
- ???
- Profit

Summary

- Goal: remove unnecessary synchronization
- Solution: whole-program static analysis
- Verdict: a nice complement
- Different programming model
- Some quite severe obstacles

Questions?

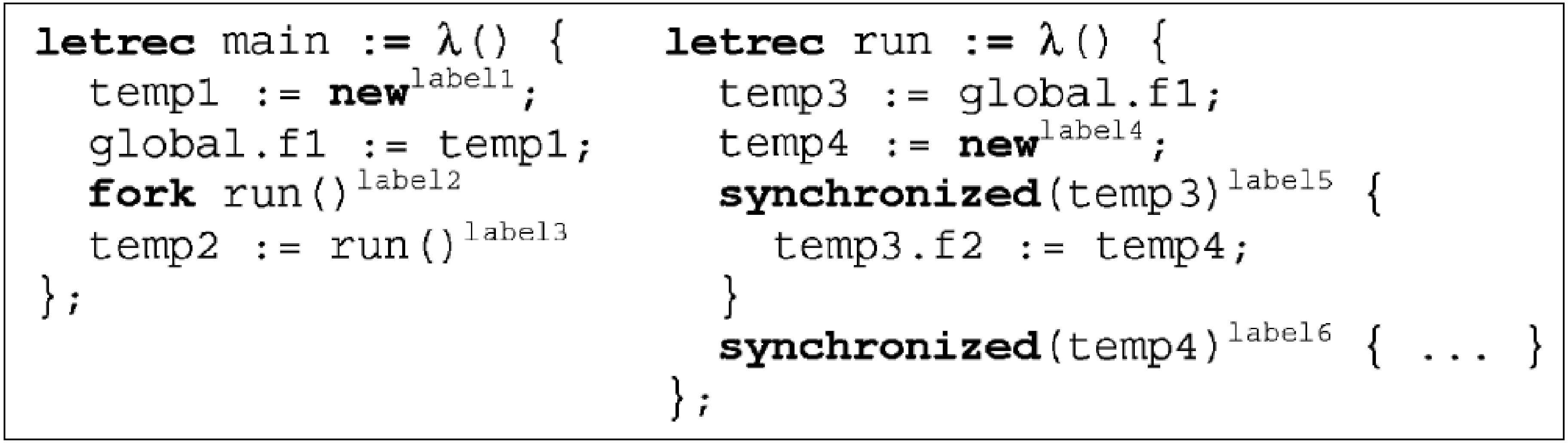


```
letrec main := λ() {
  temp1 := new^label1;
  global.f1 := temp1;
  fork run()^label2
  temp2 := run()^label3
};
```

```
letrec run := λ() {
  temp3 := global.f1;
  temp4 := new^label4;
  synchronized(temp3)^label5 {
    temp3.f2 := temp4;
  }
  synchronized(temp4)^label6 { ... }
};
```

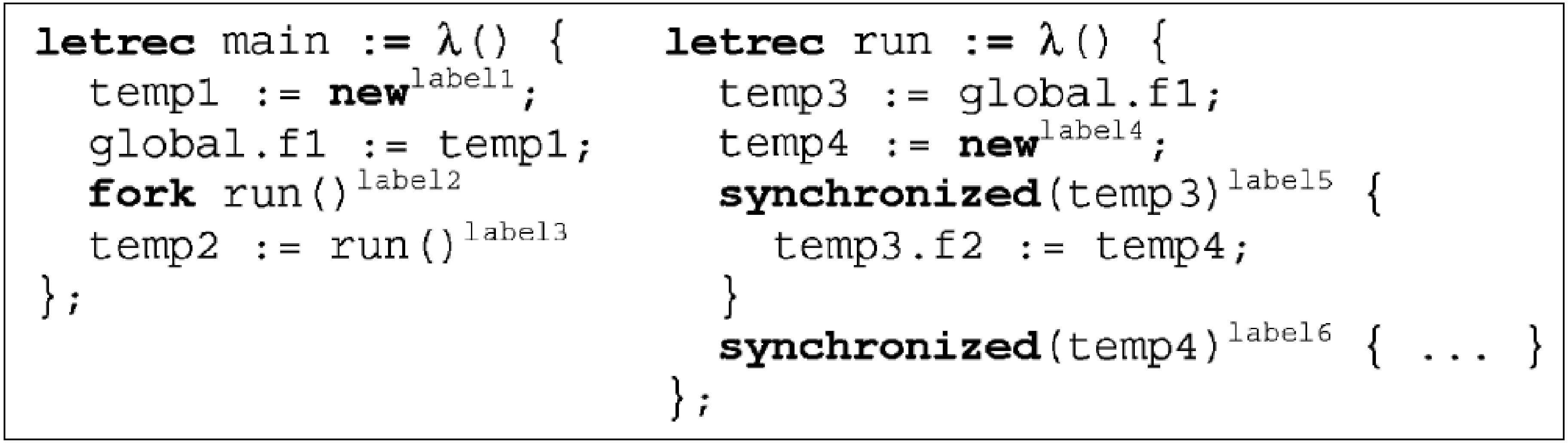
$$\frac{eval(\texttt{id}_1\texttt{.f := id}_2, t)}{written(f, t)}$$

$$\frac{eval(\texttt{id}_1\texttt{ := id}_2\texttt{.f}, t)}{read(f, t)}$$



```
letrec main := λ() {             letrec run := λ() {
  temp1 := new^label1;             temp3 := global.f1;
  global.f1 := temp1;              temp4 := new^label4;
  fork run()^label2                synchronized(temp3)^label5 {
  temp2 := run()^label3              temp3.f2 := temp4;
};                                 }
                                   synchronized(temp4)^label6 { ... }
                                 };
```

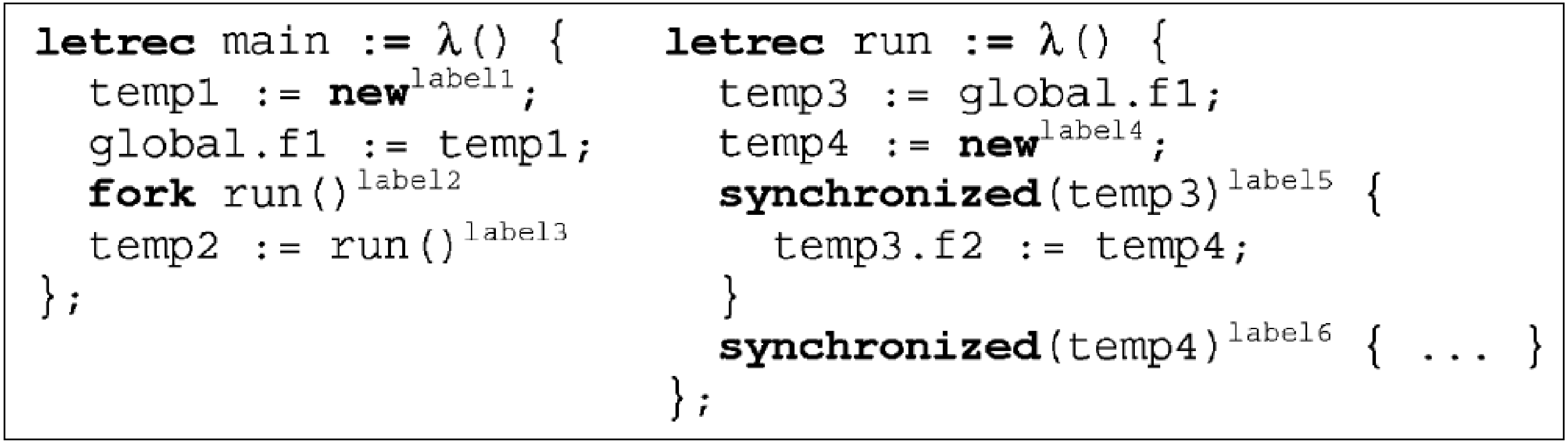
$$\frac{read(f,t_1) \quad written(f,t_2)}{multi(f)} \quad (t_1 \neq t_2)$$



```
letrec main := λ() {
  temp1 := new^label1;
  global.f1 := temp1;
  fork run()^label2
  temp2 := run()^label3
};
```

```
letrec run := λ() {
  temp3 := global.f1;
  temp4 := new^label4;
  synchronized(temp3)^label5 {
    temp3.f2 := temp4;
  }
  synchronized(temp4)^label6 { ... }
};
```

$$\frac{ref(\texttt{global}, f, o) \quad multi(f)}{multi(o)}$$



```
letrec main := λ() {
  temp1 := new^label1;
  global.f1 := temp1;
  fork run()^label2
  temp2 := run()^label3
};
```

```
letrec run := λ() {
  temp3 := global.f1;
  temp4 := new^label4;
  synchronized(temp3)^label5 {
    temp3.f2 := temp4;
  }
  synchronized(temp4)^label6 { ... }
};
```