Reducer Hyperobjects

CSE 549

Recitation 9

Motivation

```
1   bool has_property(Node *);
2   std::list<Node *> output_list;
3   // ...
4   void walk(Node *x)
5   {
6     if (x) {
7       if (has_property(x))
8         output_list.push_back(x);
9       walk(x->left);
10      walk(x->right);
11    }
12  }
```

Motivation

```
1   bool has_property(Node *);
2   std::list<Node *> output_list;
3   // ...
4   void walk(Node *x)
5   {
6      if (x) {
7         if (has_property(x))
8            output_list.push_back(x);
9         cilk_spawn walk(x->left);
10        walk(x->right);
11        cilk_sync;
12     }
13  }
```

Solutions

- Locks

```
 8        if (has_property(x)) {
 9          L.lock();
10          output_list.push_back(x);
11          L.unlock();
12        }
```

  - Can create a bottleneck during run time, or possibly cause deadlock

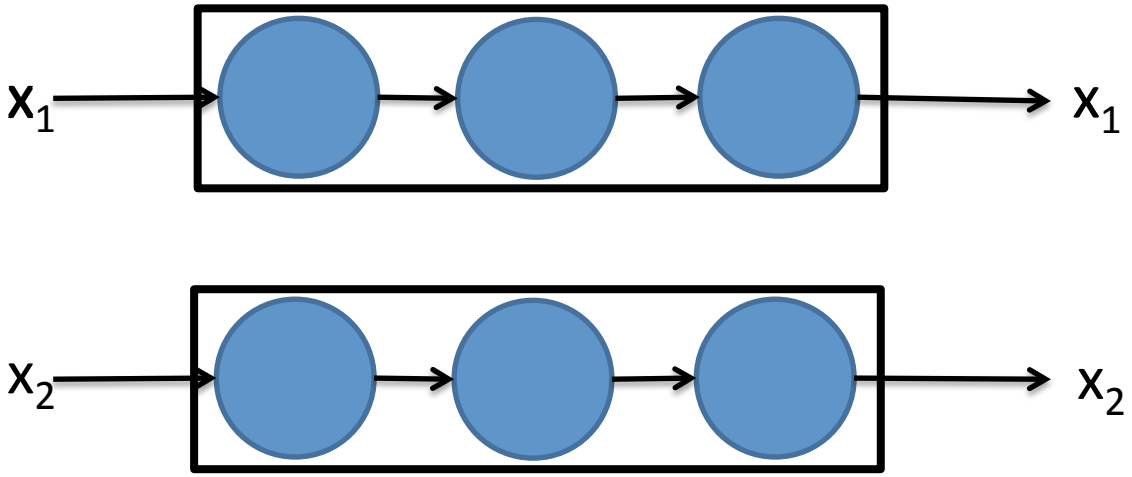
- Restructuring

Restructuring

```
1   bool has_property(Node *);
2   std::list<Node *> output_list;
3   // ...
4   void walk(Node *x)
5   {
6     if (x) {
7       if (has_property(x))
8         output_list.push_back(x);
9       cilk_spawn walk(x->left);
10      walk(x->right);
11      cilk_sync;
12    }
13  }
```

Solutions

- Hyperobjects
  - What are they?



Reducer Hyperobjects

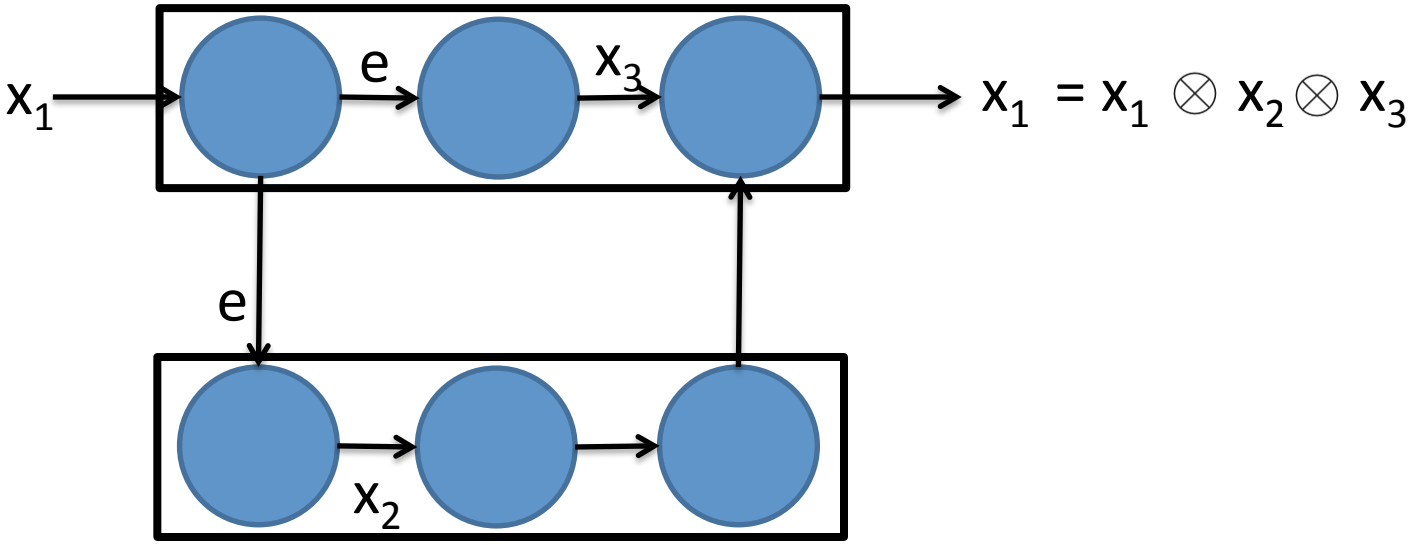
- Based on an algebraic monoid:
  - A set, T (integers, Nodes, etc.)
  - An associative binary operator, $\otimes$ (+, x, concatenation, etc.)
  - An identity, e, such that $e \otimes a = a \otimes e = a$ (0, 1, empty set, etc.)

Example of Cilk++ reducer

```
1   int compute(const X& v);
2   int cilk_main()
3   {
4     const std::size_t n = 1000000;
5     extern X myArray[n];
6     // ...
7     sum_reducer<int> result(0);
8     cilk_for (std::size_t i = 0; i < n; ++i)
9       result += compute(myArray[i]);
0
1     std::cout << "The result is: "
2               << result.get_value()
3               << std::endl;
4     return 0;
5   }
```

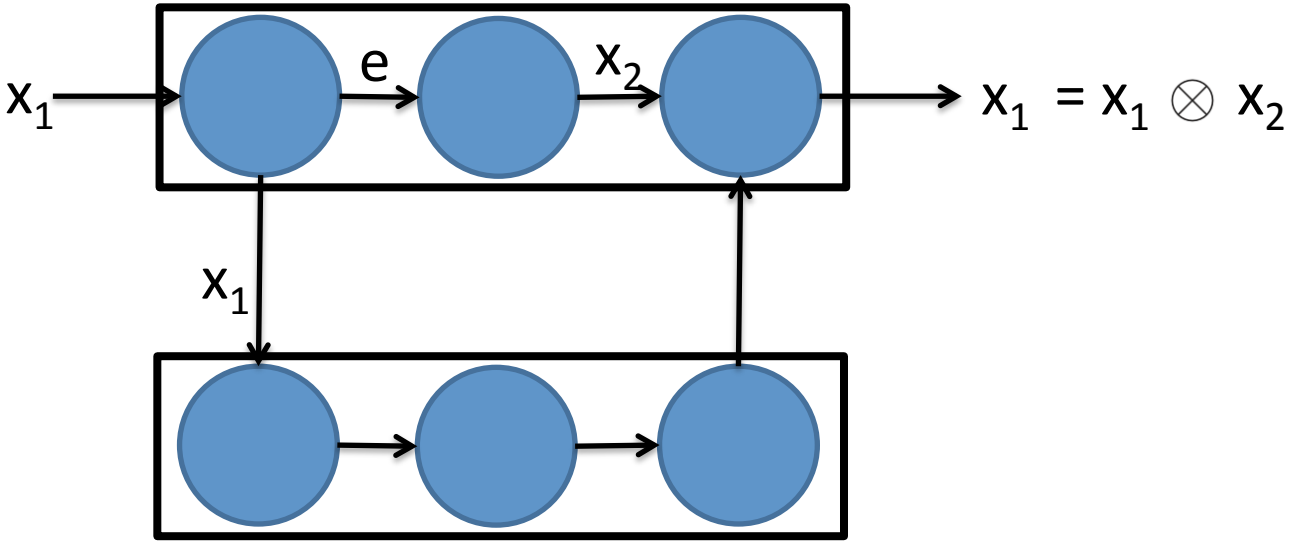
Naïve Implementation

- Create 2 views at a spawn set to e

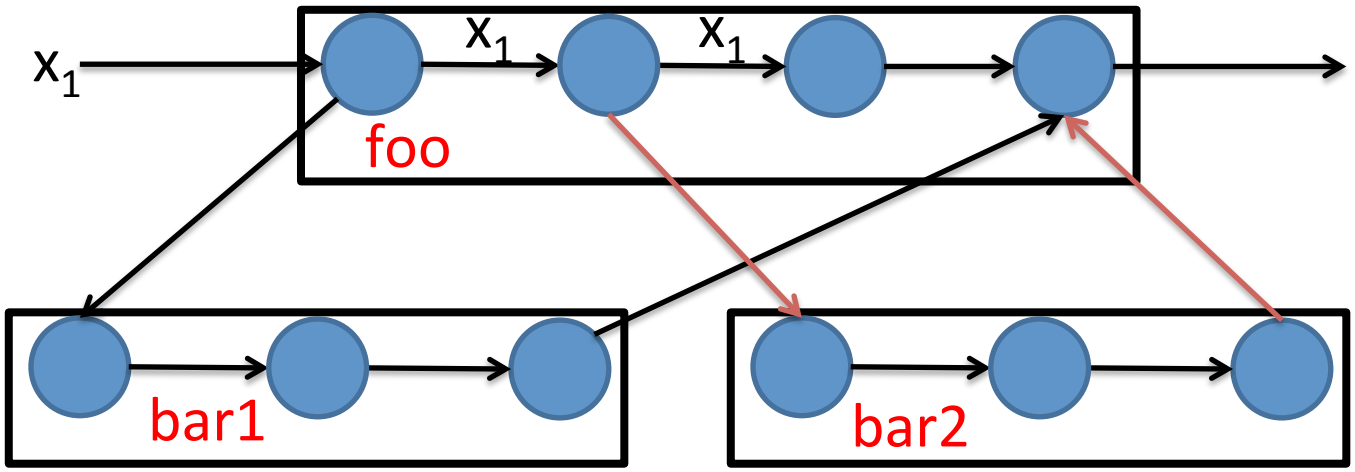


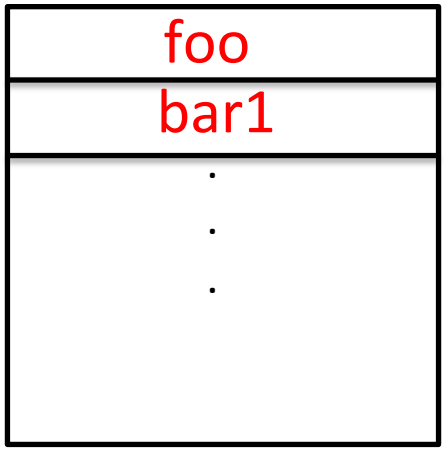
Better Way

- Create 1 view at a spawn set to e
- Give the parent view to the child



Even Better Way

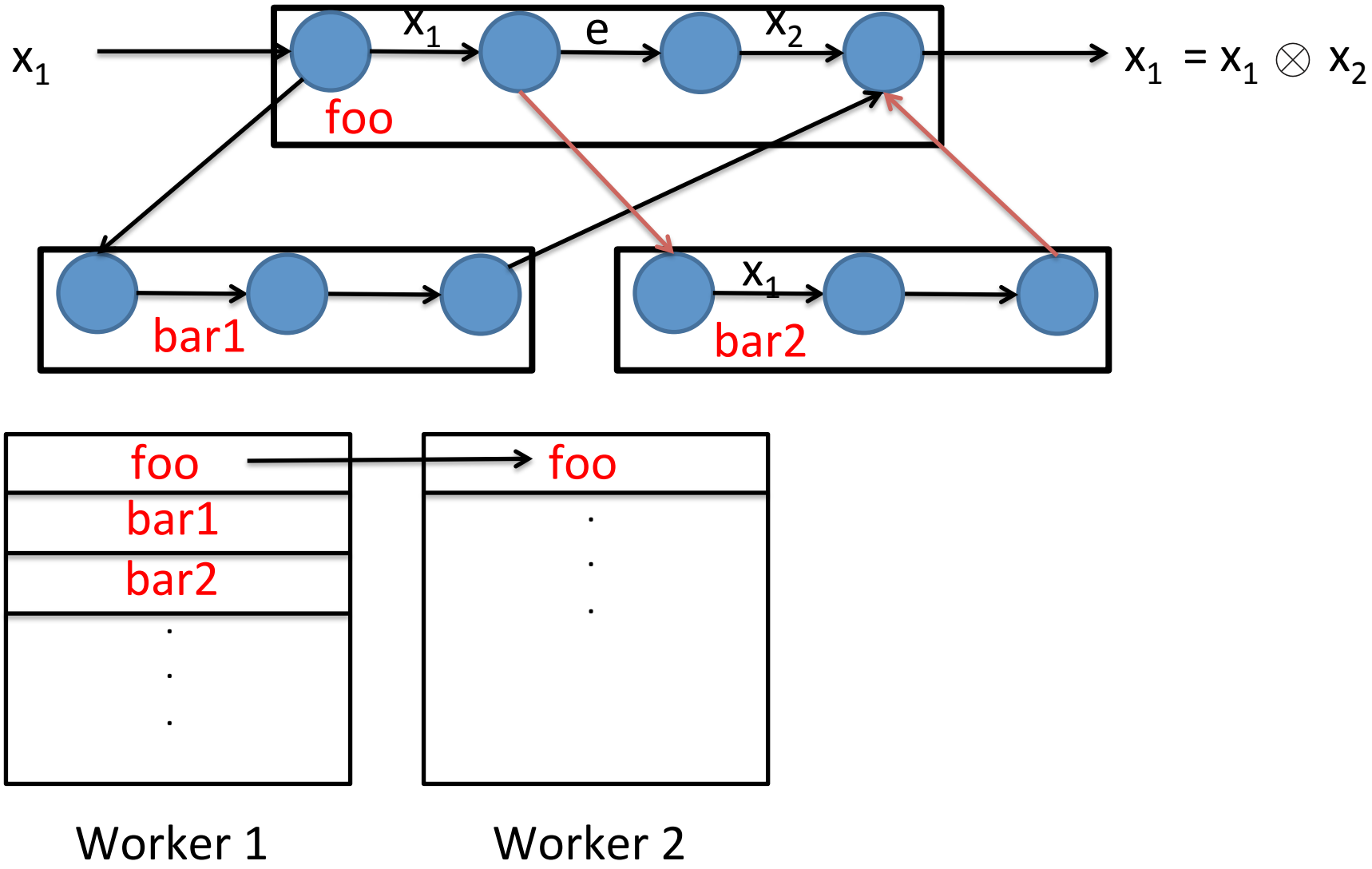




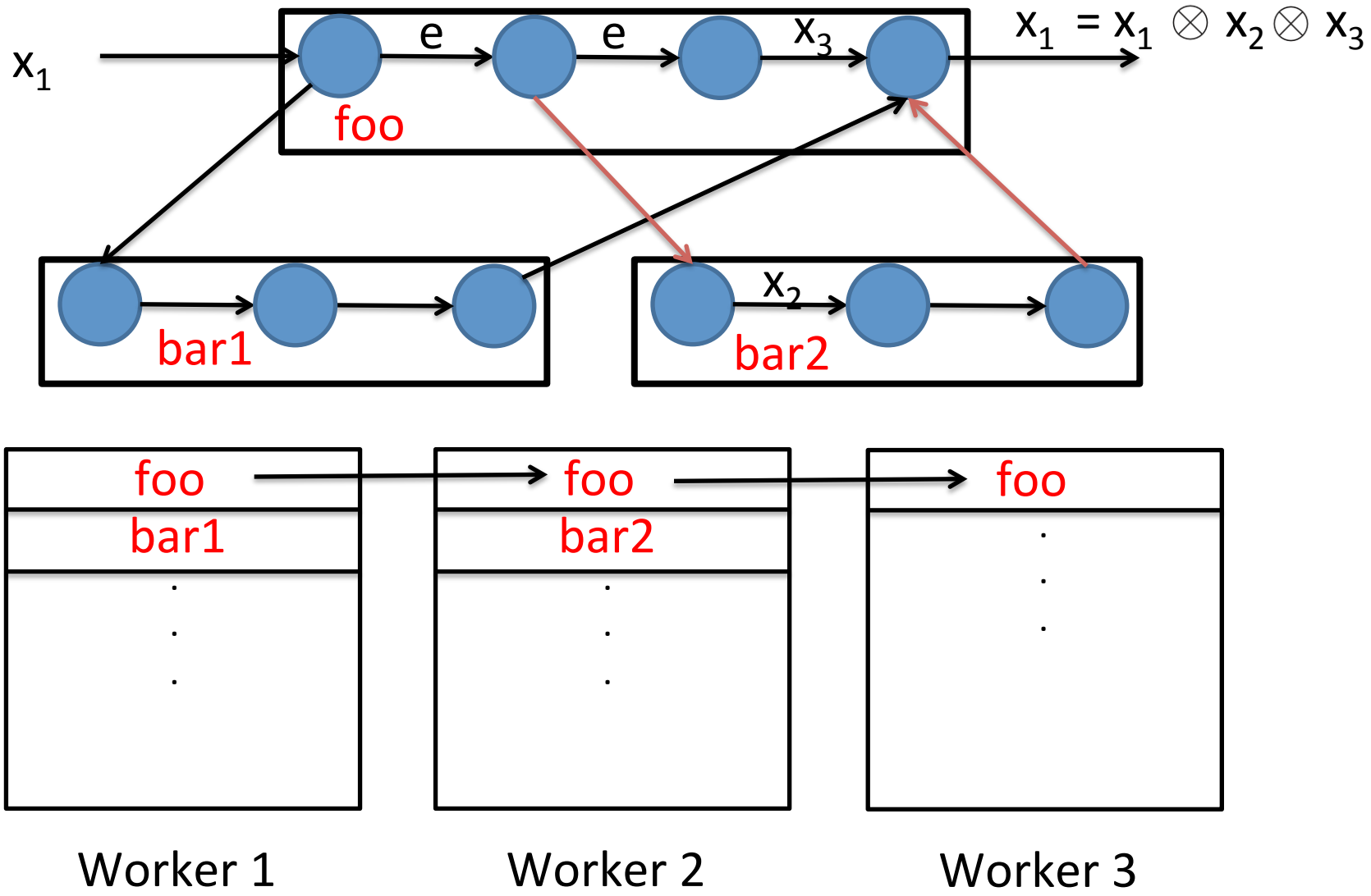
Worker 1

- Only create a view if a steal occurs

Even Better Way

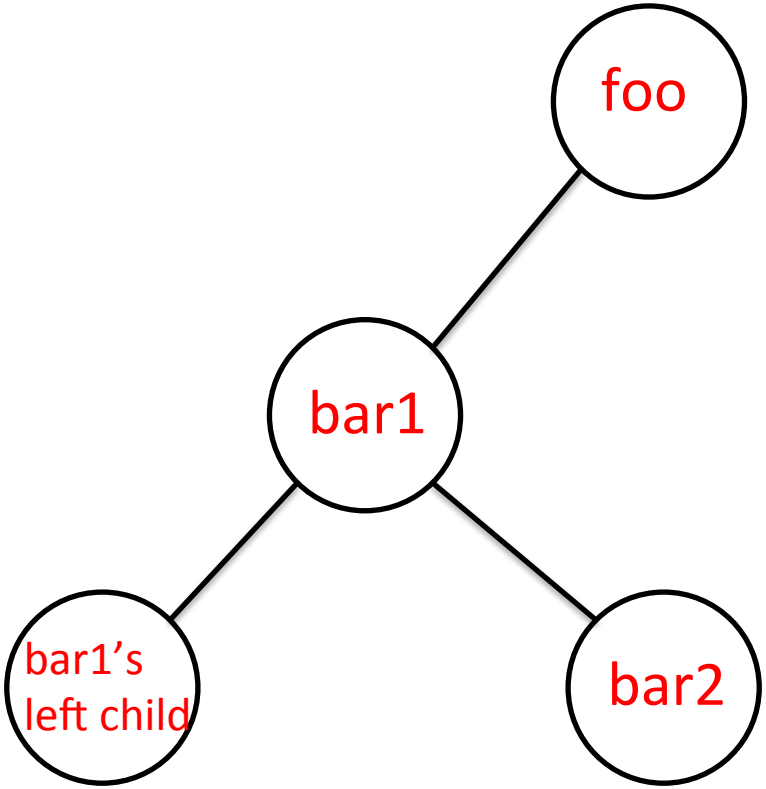


Even Better Way



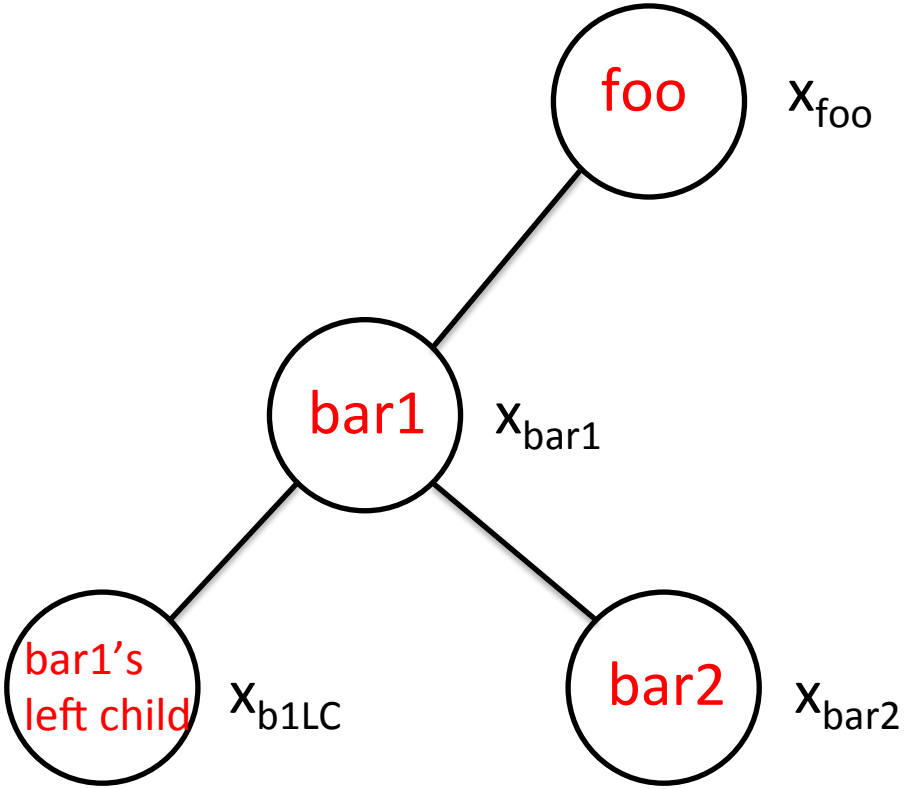
Steal Tree

- Left-child, right-sibling binary tree



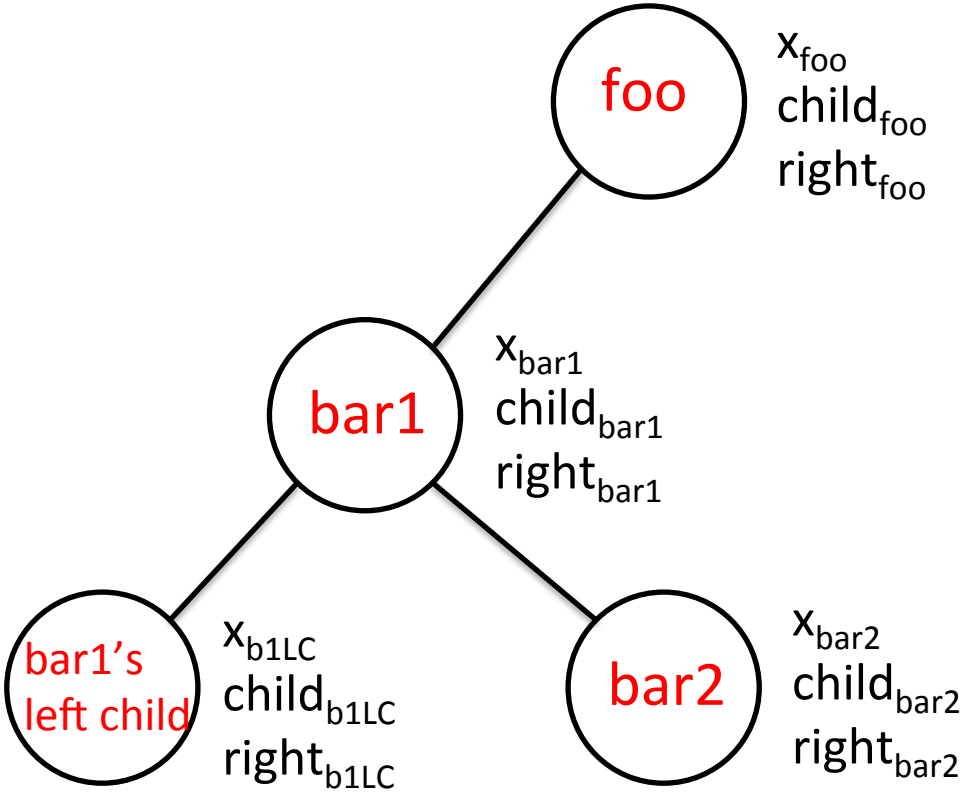
Steal Tree

- Left-child, right-sibling binary tree



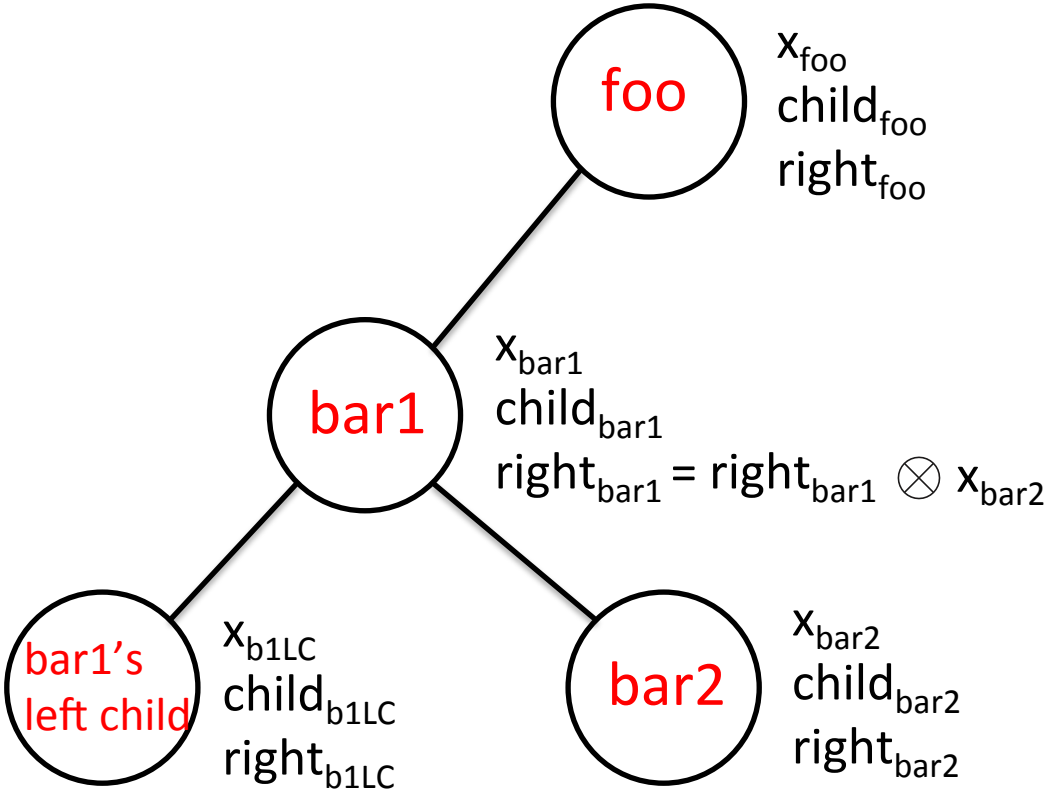
Steal Tree

- Include 2 “placeholder” variables



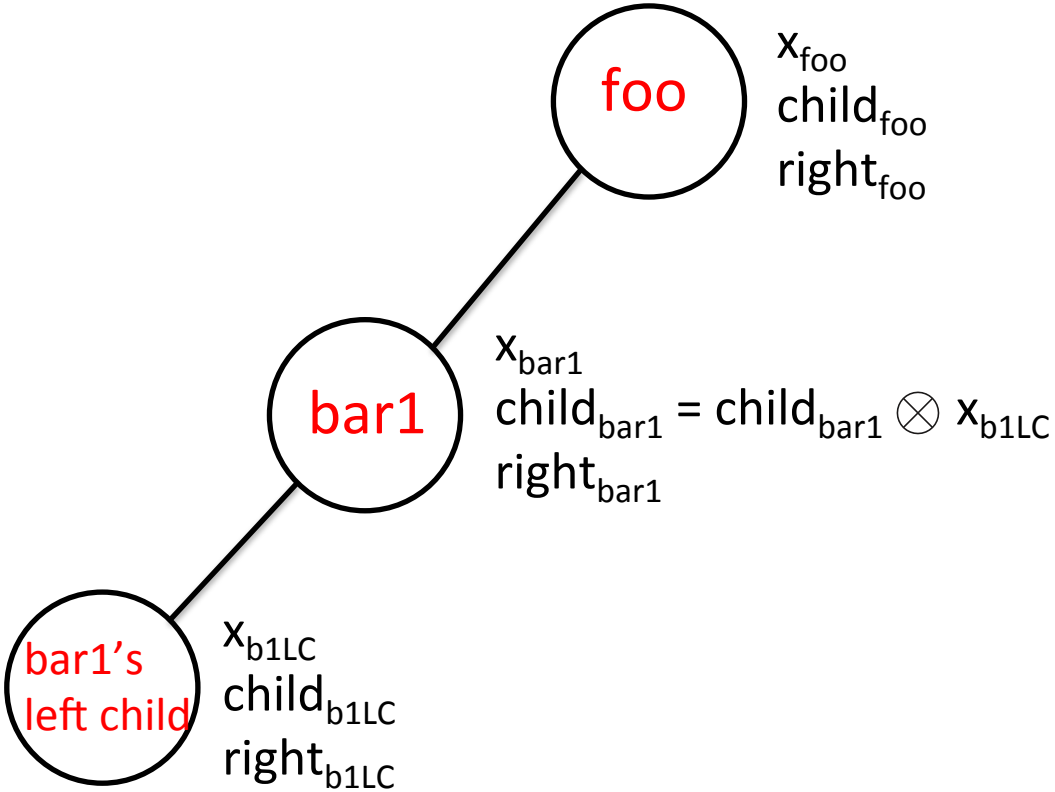
Steal Tree

- bar2 finishes first



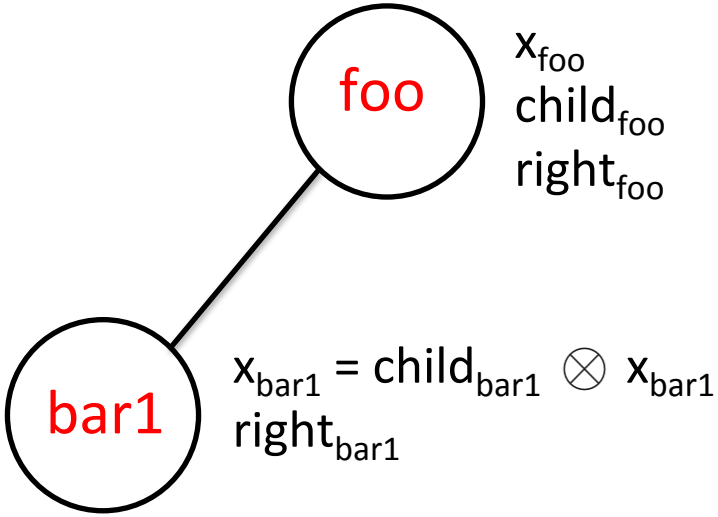
Steal Tree

- Left child finishes



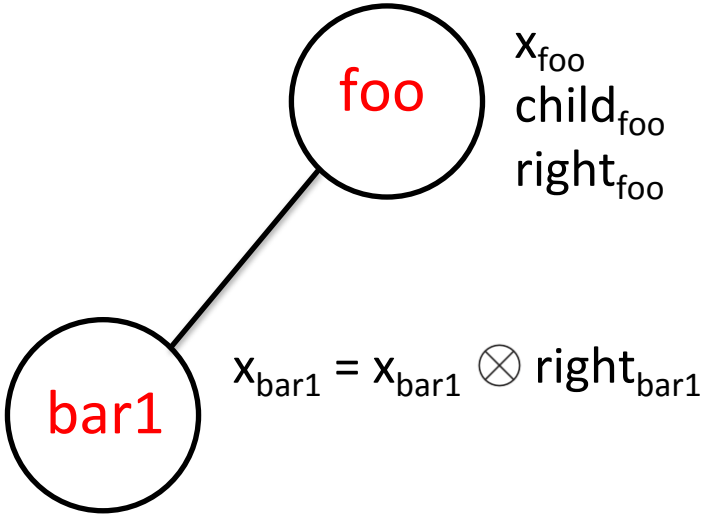
Steal Tree

- bar1 finishes



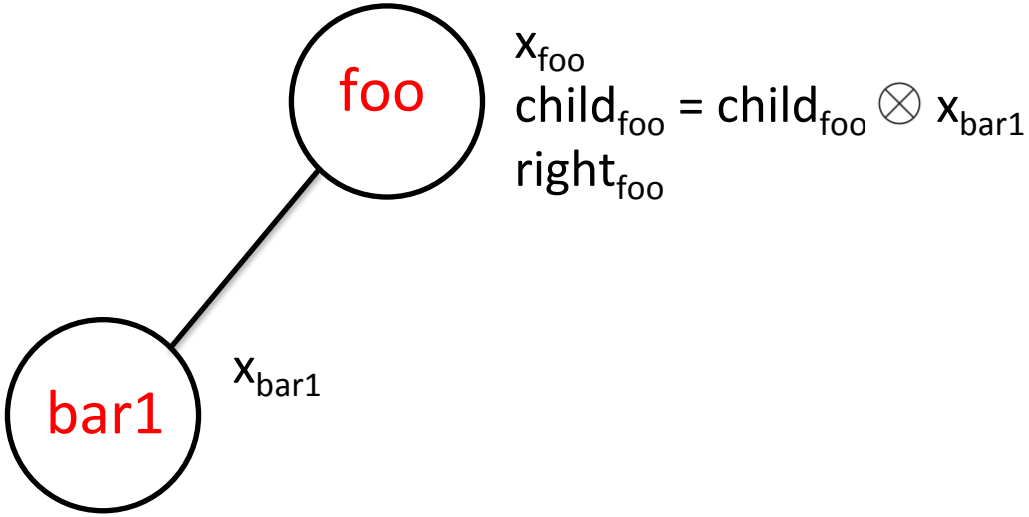
Steal Tree

- bar1 finishes



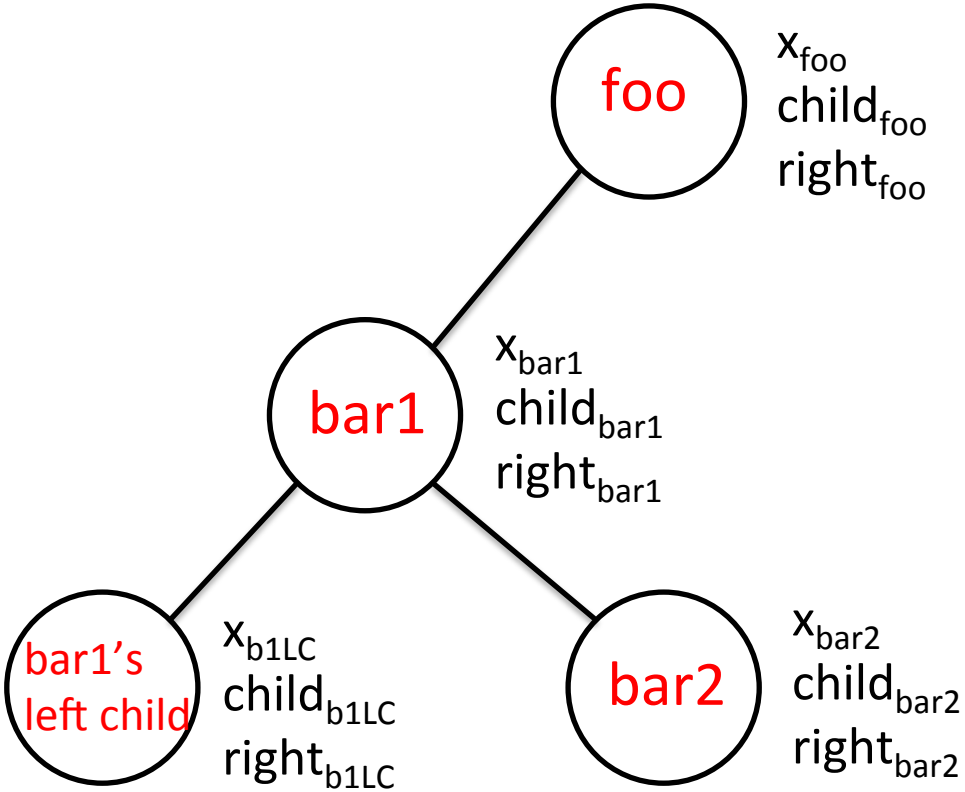
Steal Tree

- bar1 finishes

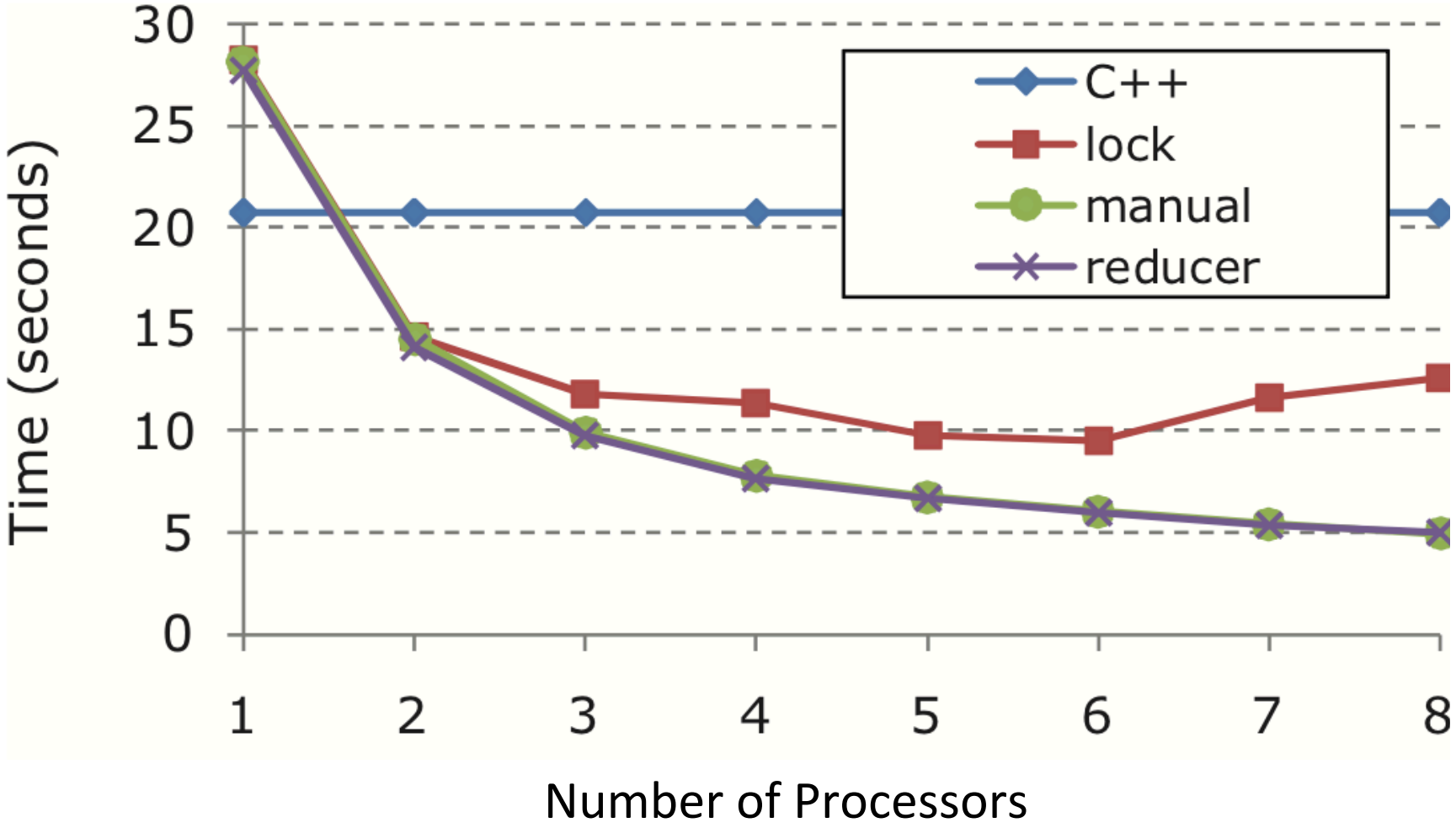


Steal Tree

- Need to use locks



Performance



Holders

```
1   holder<T> global_variable;
2   // originally: T global_variable
3
4   void proc1() {
5     cilk_for (i = 0; i < N; ++i) { //was: for
6       global_variable = f(i);
7       proc2();
8     }
9   }
10
11  void proc2() { proc3(); }
12  void proc3() { proc4(); }
13
14  void proc4() {
15    use(global_variable);
16  }
```