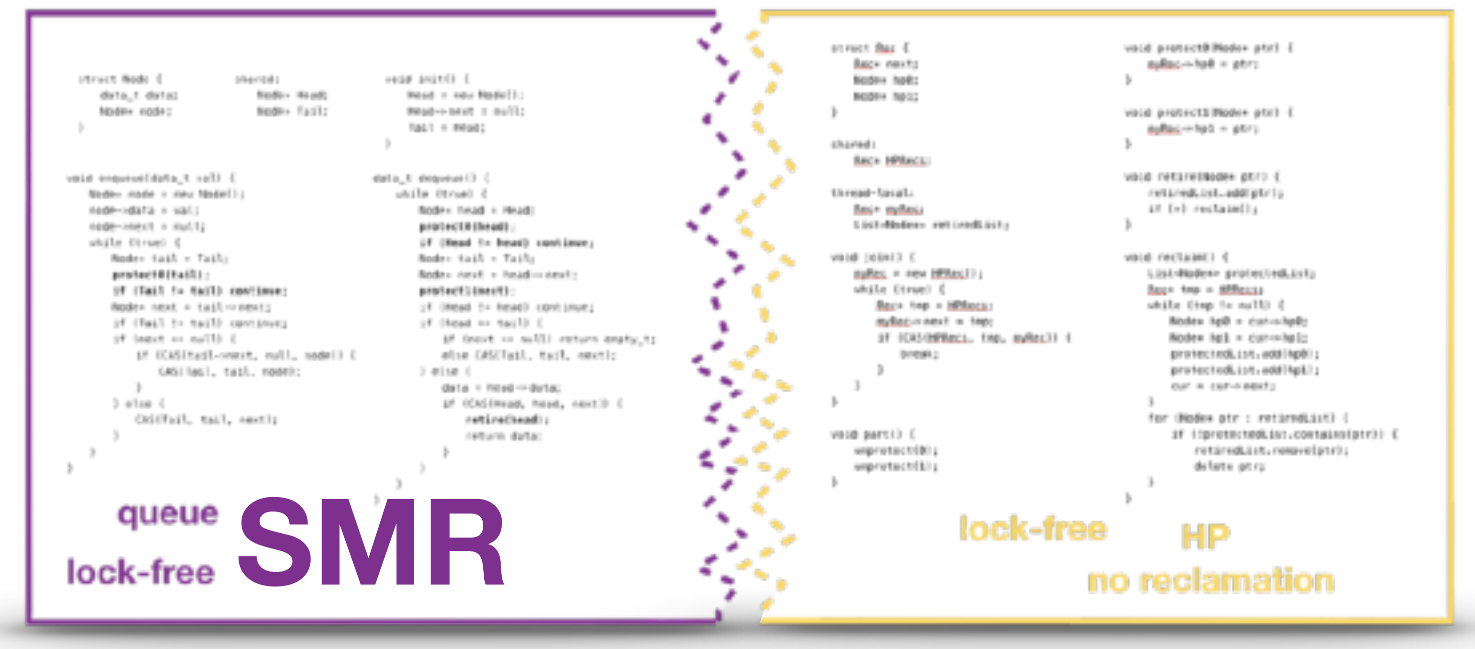
Experiments

Data Structure with HP	Types	Annot.	Lin.
Treiber's stack	0.7s ✓	12s ✓	1s ✓
Michael&Scott's queue	0.6s ✓	11s ✓	4s ✓
DGLM queue	0.6s ✓	1s ✗*	5s ✓
Vechev&Yahav's 2CAS set	1.2s ✓	13s ✓	98s ✓
Vechev&Yahav's CAS set	1.2s ✓	3.5h ✓	42m ✓
ORVYY set	1.2s ✓	3.2h ✓	47m ✓
Michael's set	1.2s ✓	90s ✗*	t/o 🕒

* imprecision in the back-end verifier

Final Approach



```
struct Node {              shared:              void init() {
    data_t data;               Node* Head;          Head = new Node();
    Node* node;                Node* Tail;          Head->next = null;
}                                                   Tail = Head;
                                                 }

void enqueue(data_t val) {                    data_t dequeue() {
    Node* node = new Node();                      while (true) {
    node->data = val;                                 Node* head = Head;
    node->next = null;                                protect0(head);
    while (true) {                                    if (Head != head) continue;
        Node* tail = Tail;                            Node* tail = Tail;
        protect0(tail);                               Node* next = head->next;
        if (Tail != tail) continue;                   protect1(next);
        Node* next = tail->next;                      if (Head != head) continue;
        if (Tail != tail) continue;                   if (head == tail) {
        if (next == null) {                               if (next == null) return empty_t;
            if (CAS(tail->next, null, node)) {            else CAS(Tail, tail, next);
                CAS(Tail, tail, node);                } else {
            }                                             data = head->data;
        } else {                                          if (CAS(Head, head, next)) {
            CAS(Tail, tail, next);                            retire(head);
        }                                                     return data;
    }                                                     }
}                                                     }
                                                  }
                                              }
```

```
struct Rec {                          void protect0(Node* ptr) {
    Rec* next;                            myRec->hp0 = ptr;
    Node* hp0;                        }
    Node* hp1;
}                                     void protect1(Node* ptr) {
                                          myRec->hp1 = ptr;
shared:                               }
    Rec* HPRecs;
                                      void retire(Node* ptr) {
thread-local:                             retiredList.add(ptr);
    Rec* myRec;                           if (*) reclaim();
    List<Node*> retiredList;          }

void join() {                         void reclaim() {
    myRec = new HPRec();                  List<Node*> protectedList;
    while (true) {                        Rec* tmp = HPRecs;
        Rec* tmp = HPRecs;                while (tmp != null) {
        myRec->next = tmp;                    Node* hp0 = cur->hp0;
        if (CAS(HPRecs, tmp, myRec)) {        Node* hp1 = cur->hp1;
            break;                            protectedList.add(hp0);
        }                                     protectedList.add(hp1);
    }                                         cur = cur->next;
}                                         }
                                          for (Node* ptr : retiredList) {
void part() {                                 if (!protectedList.contains(ptr)) {
    unprotect(0);                                 retiredList.remove(ptr);
    unprotect(1);                                 delete ptr;
}                                             }
                                          }
                                      }
```