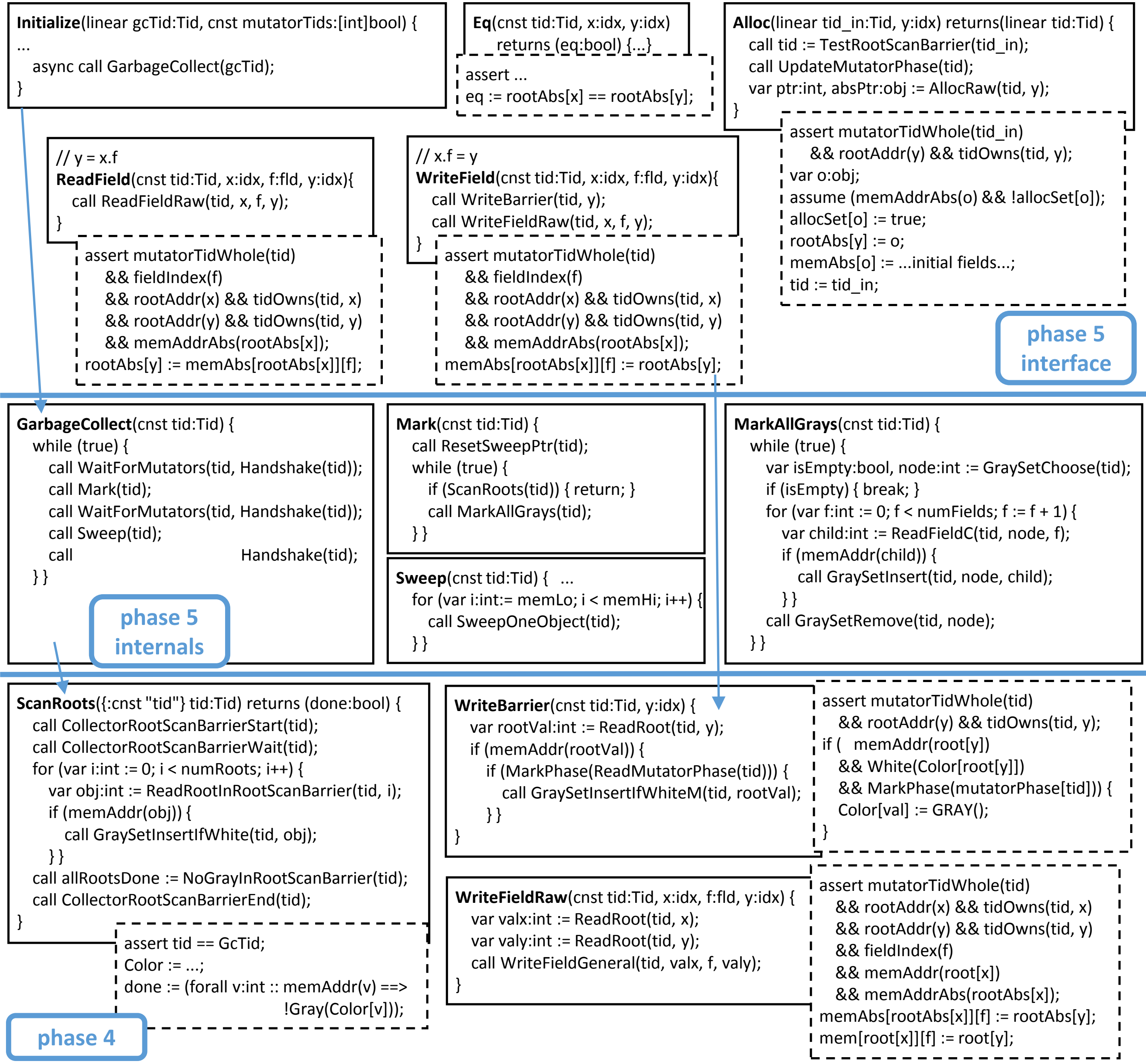

```
Initialize(linear gcTid:Tid, cnst mutatorTids:[int]bool) {
...
  async call GarbageCollect(gcTid);
}
```

```
Eq(cnst tid:Tid, x:idx, y:idx)
   returns (eq:bool) {...}
```

```
assert ...
eq := rootAbs[x] == rootAbs[y];
```

```
Alloc(linear tid_in:Tid, y:idx) returns(linear tid:Tid) {
  call tid := TestRootScanBarrier(tid_in);
  call UpdateMutatorPhase(tid);
  var ptr:int, absPtr:obj := AllocRaw(tid, y);
}
```

```
assert mutatorTidWhole(tid_in)
   && rootAddr(y) && tidOwns(tid, y);
var o:obj;
assume (memAddrAbs(o) && !allocSet[o]);
allocSet[o] := true;
rootAbs[y] := o;
memAbs[o] := ...initial fields...;
tid := tid_in;
```

```
// y = x.f
ReadField(cnst tid:Tid, x:idx, f:fld, y:idx){
  call ReadFieldRaw(tid, x, f, y);
}
```

```
assert mutatorTidWhole(tid)
   && fieldIndex(f)
   && rootAddr(x) && tidOwns(tid, x)
   && rootAddr(y) && tidOwns(tid, y)
   && memAddrAbs(rootAbs[x]);
rootAbs[y] := memAbs[rootAbs[x]][f];
```

```
// x.f = y
WriteField(cnst tid:Tid, x:idx, f:fld, y:idx){
  call WriteBarrier(tid, y);
  call WriteFieldRaw(tid, x, f, y);
}
```

```
assert mutatorTidWhole(tid)
   && fieldIndex(f)
   && rootAddr(x) && tidOwns(tid, x)
   && rootAddr(y) && tidOwns(tid, y)
   && memAddrAbs(rootAbs[x]);
memAbs[rootAbs[x]][f] := rootAbs[y];
```

```
GarbageCollect(cnst tid:Tid) {
  while (true) {
    call WaitForMutators(tid, Handshake(tid));
    call Mark(tid);
    call WaitForMutators(tid, Handshake(tid));
    call Sweep(tid);
    call                         Handshake(tid);
  } }
```

```
Mark(cnst tid:Tid) {
  call ResetSweepPtr(tid);
  while (true) {
    if (ScanRoots(tid)) { return; }
    call MarkAllGrays(tid);
  } }
```

```
Sweep(cnst tid:Tid) {  ...
  for (var i:int:= memLo; i < memHi; i++) {
    call SweepOneObject(tid);
  } }
```

```
MarkAllGrays(cnst tid:Tid) {
  while (true) {
    var isEmpty:bool, node:int := GraySetChoose(tid);
    if (isEmpty) { break; }
    for (var f:int := 0; f < numFields; f := f + 1) {
      var child:int := ReadFieldC(tid, node, f);
      if (memAddr(child)) {
        call GraySetInsert(tid, node, child);
    } }
    call GraySetRemove(tid, node);
  } }
```

```
ScanRoots({:cnst "tid"} tid:Tid) returns (done:bool) {
  call CollectorRootScanBarrierStart(tid);
  call CollectorRootScanBarrierWait(tid);
  for (var i:int := 0; i < numRoots; i++) {
    var obj:int := ReadRootInRootScanBarrier(tid, i);
    if (memAddr(obj)) {
      call GraySetInsertIfWhite(tid, obj);
  } }
  call allRootsDone := NoGrayInRootScanBarrier(tid);
  call CollectorRootScanBarrierEnd(tid);
}
```

```
assert tid == GcTid;
Color := ...;
done := (forall v:int :: memAddr(v) ==>
                         !Gray(Color[v]));
```

```
WriteBarrier(cnst tid:Tid, y:idx) {
  var rootVal:int := ReadRoot(tid, y);
  if (memAddr(rootVal)) {
    if (MarkPhase(ReadMutatorPhase(tid))) {
      call GraySetInsertIfWhiteM(tid, rootVal);
  } }
}
```

```
assert mutatorTidWhole(tid)
   && rootAddr(y) && tidOwns(tid, y);
if (   memAddr(root[y])
   && White(Color[root[y]])
   && MarkPhase(mutatorPhase[tid])) {
   Color[val] := GRAY();
}
```

```
WriteFieldRaw(cnst tid:Tid, x:idx, f:fld, y:idx) {
  var valx:int := ReadRoot(tid, x);
  var valy:int := ReadRoot(tid, y);
  call WriteFieldGeneral(tid, valx, f, valy);
}
```

```
assert mutatorTidWhole(tid)
   && rootAddr(x) && tidOwns(tid, x)
   && rootAddr(y) && tidOwns(tid, y)
   && fieldIndex(f)
   && memAddr(root[x])
   && memAddrAbs(rootAbs[x]);
memAbs[rootAbs[x]][f] := rootAbs[y];
mem[root[x]][f] := root[y];
```