```
struct ThreadContext {
  std::vector<void*> pending_reclaims;
  std::atomic<uint64_t> counter;
  ThreadContext *next;

  ThreadContext(MemManager m) {
    while (true) {
      next = m.head;
      if (m.head.compare_exchange_strong(next, this))
        break;
    }
  }
}
struct MemManager {
  static thread_local ThreadContext *self = nullptr;
  std::atomic<ThreadContext*> head;
  ...
}
MemManager::register_thread(int num) { self = new ThreadContext(this); }
MemManager::unregister_thread() { /* no-op */ }
MemManager::op_begin() { self->counter++; }
void MemManager::sched_for_reclaim(void* ptr) { self->pending_reclaims.push_back(ptr); }
bool MemManager::try_reserve(void* ptr) { return false; }
void MemManager::unreserve(void* ptr) { }

void MemManager::op_end() {
  self->counter++;
  if (pending_reclaims.count() == 0)
    return;
  wait_until_unreserved()
  for (auto p : pending_reclaims)
    free(p);
}
void MemManager::wait_until_unreserved() {
  ThreadContext* curr = head;
  while (curr) {
    uint64_t val = curr->counter;
    if (odd(val))
    do {
      wait();
    } while (curr->counter.read() == val)
    curr = curr->next;
  }
}
```

FIGURE 19.8 Epoch-based reclamation.