```
uintprt_t read(uintptr_t *ptr) {
  auto it = writes.find(addr);
  if (it != writes.end())
    return *it;
  uintptr_t val = std::atomic_ref<uintptr_t>(*ptr).load(std::memory_order_acquire);
  while (start_time != globals.lock.val) {
    start_time = validate();
    val = std::atomic_ref<uintptr_t>(*ptr).load(std::memory_order_acquire);
  }
  reads.push_back({addr, val});
  return val;
}
void commitTx() {
  if (writes.empty()) {
    reads.clear();
    return;
  }
  uint64_t from = start_time;
  while (!lock.compare_exchange_strong(from, from + 1))
    from = validate();
  start_time = from;
  for (auto w : writes)
    std::atomic_ref<uintptr_t>(*w.first).store(w.second, std::memory_order_release);
  lock = 2 + start_time;
  writes.clear();
  reads.clear();
}
uint64_t validate() {
  while (true) {
    uint64_t time = lock;
    if (time & 1)
      continue;
    for (auto it = reads.begin(), e = reads.end(); it != e; ++it) {
      if (std::atomic_ref<uintptr_t>(*it.first).load(std::memory_order_acquire) !=
          it.second)
        abortTx();
    }
    if (time == lock)
      return time;
  }
}
```

FIGURE 20.14 Software transactions with value-based validation (2/2).