Programmazione concorrente

Laurea Magistrale in Ingegneria Informatica
Università Tor Vergata
Docente: Romolo Marotta

Safe Memory Reclamation

Safe Memory Reclamation

WHAT is SMR?

- Approach to release/reuse memory only when it is safe
- A bit different to Garbage Collectors

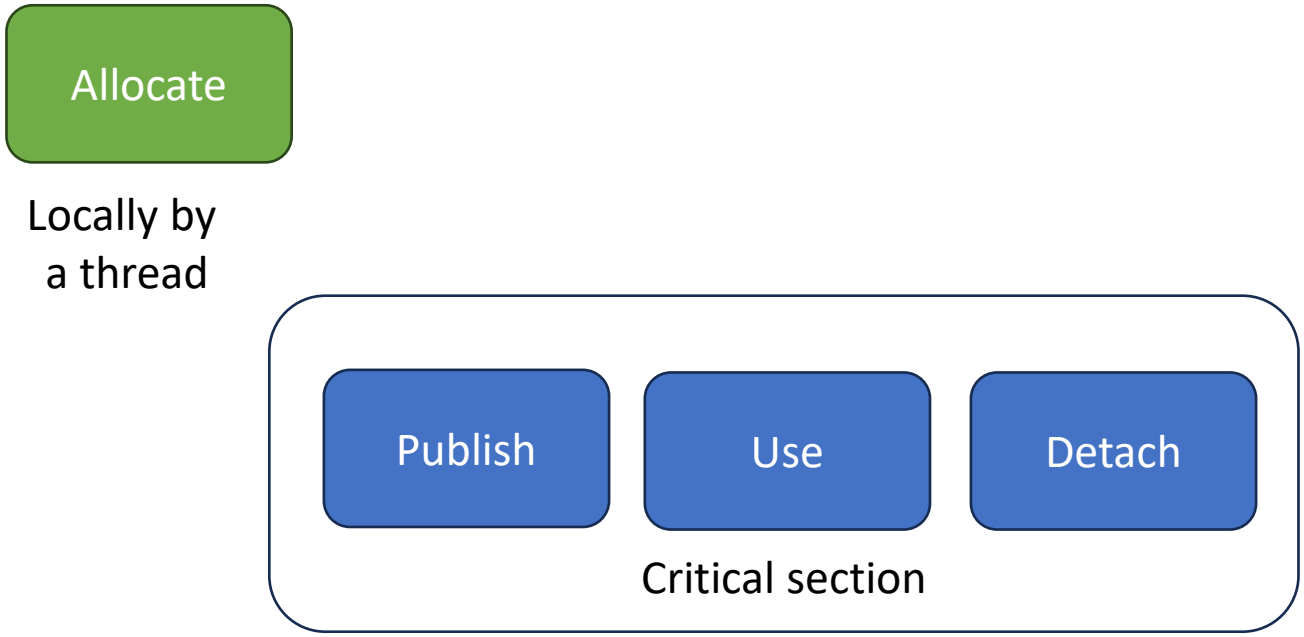
WHY SMR?

- In non-blocking algorithms, shared memory accesses are NOT protected by critical section via locks
- A thread can dereference a pointer whose target memory might be released (dangling pointers)
- Undefined behaviors, crash, memory corruptions

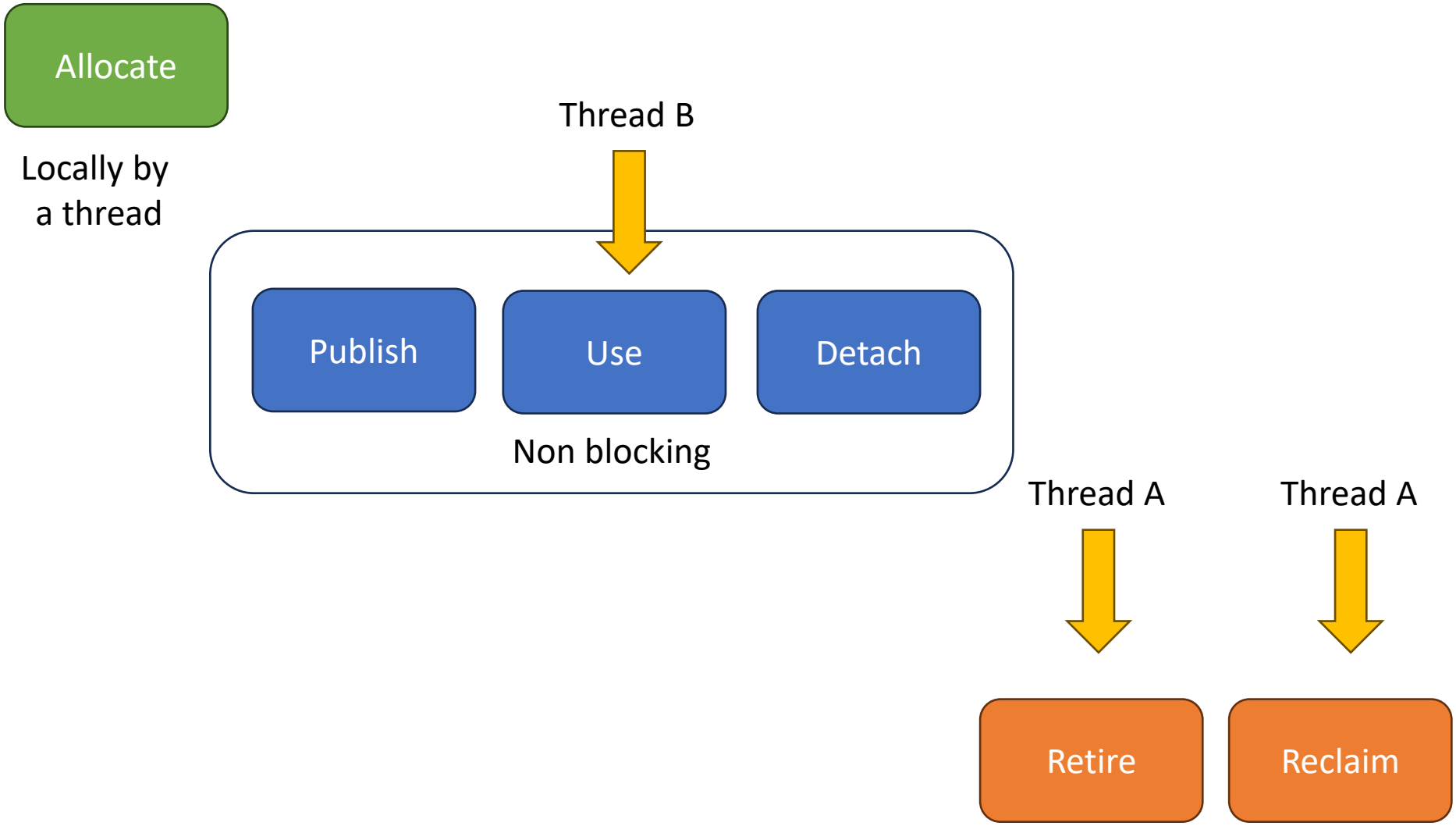
Terminologia SMR

- Allocation: creating the object
- Publish: making it visible to other threads
- Detach: removing it from shared structures
- Retiring: marking it for future reclamation
- Reclaim: actual deallocation

Life cycle of a shared object



Life cycle of a shared object



SMR design

- Avoid stop-the-world approaches
- **Robustness**: a stalled thread should prevent only a **bounded** number of blocks from being reclaimed
- Threads needs to:
  - Maintain a list of memory buffers for later reclamation (retired list)
  - Identify grace periods for buffer reusage

Hazard Pointers

- Thread:
  - publish hazard pointers that they are going to use
  - unpublish pointers when unused by replacing it with NULL
- Memory referenced by a non-null HP cannot be reclaimed
- Let be:
  - N the number of threads
  - K the maximum number of HPs per thread
- We need:
  - NxK array of HPs publicly available to all threads
- The scheme can be generalized to support dynamic number of threads/HPs

Hazard Pointers: how to reclaim?

- After detaching a memory buffer, put references into a retired list
- Periodically scan the HP array
- While scanning HPs build a fast map for later searches (e.g. hashmap or binary search tree)
- For each retired buffer check the map
  - If present re-put in a new retired list
  - If not reclaim memory
- Swap the old retired list with the new one

Hazard Pointers: why works?

- Identifies the end of per-pointer grace periods as
  - The buffer is detached (and hence retired)
  - There is no HP pointing to it

Hazard Pointers: example

```
structure NodeType { Data:DataType; Next:*NodeType; }
// Shared variables
Head,Tail:*NodeType;
// Initially both Head and Tail point to a dummy node

Enqueue(data:DataType) {
 1:   node ← NewNode();
 2:   node^.Data ← data;
 3:   node^.Next ← null;
      while true {
 4:       t ← Tail;
 5:       next ← t^.Next;
 6:       if (Tail ≠ t) continue;
 7:       if (next ≠ null) { CAS(&Tail,t,next); continue; }
 8:       if CAS(&t^.Next,null,node) break;
      }
 9:   CAS(&Tail,t,node);
}
```

```
Dequeue() : DataType {
      while true {
11:       h ← Head;
12:       t ← Tail;
13:       next ← h^.Next;
14:       if (Head ≠ h) continue;
15:       if (next = null) return EMPTY;
16:       if (h = t) { CAS(&Tail,t,next); continue;}
17:       data ← next^.Data;
18:       if CAS(&Head,h,next) break;
      }
19:   return data;
}
```

Hazard Pointers: why works?

- Identifies the end of per-pointer grace periods as
  - The buffer is detached (and hence retired)
  - There is no HP pointing to it
- Threads double check that just published HP are still usable