CS842: Automatic Memory Management and Garbage Collection

Mark and sweep

Schedule

	M	W
Sept 14	Intro/Background	Basics/ideas
Sept 21	Allocation/layout	GGGGC
Sept 28	Mark/Sweep	Mark/Sweep
Octo 5	Copying GC	Ref C
Octo 12	Thanksgiving	Mark/Compact
Octo 19	Partitioning/Gen	Generational
Octo 26	Other part	Runtime
Nove 2	Final/weak	Conservative
Nove 9	Ownership	Regions etc
Nove 16	Adv topics	Adv topics
Nove 23	Presentations	Presentations
Nove 30	Presentations	Presentations

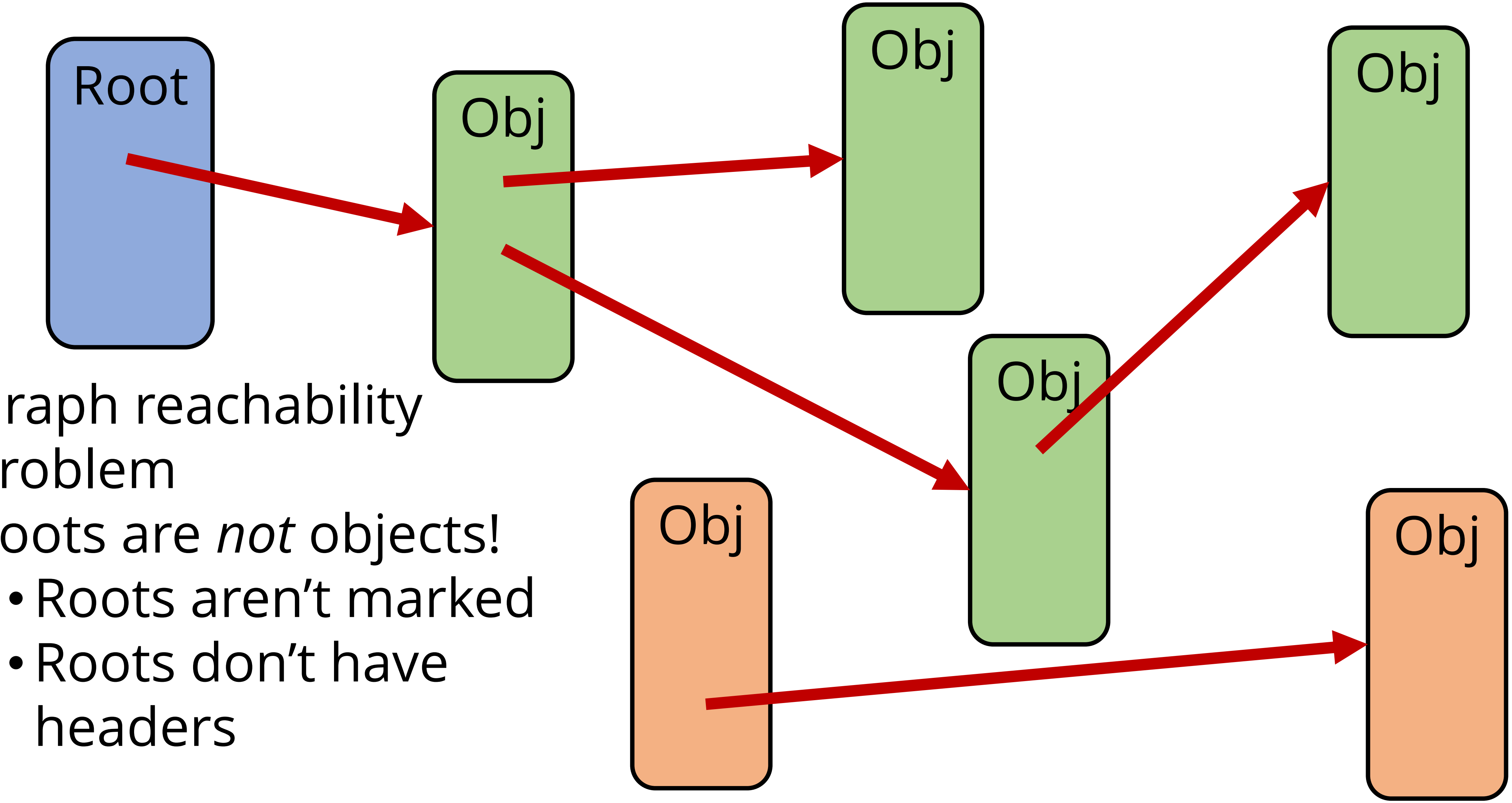
Review

- Memory manager: Allocation and revocation
- Revocation linked to allocation
- Scan heap for reachable objects, sweep to free unreachable ones

Review

- Mutator yields
- Collector decides when to collect
- Collector controls allocation
- “Stop the world”: Collector in complete control of heap

Marking



- Graph reachability problem
- Roots are *not* objects!
    - Roots aren’t marked
    - Roots don’t have headers

The mark algorithm (one version)

```
markPhase():
  worklist := new Queue
  foreach loc in roots:
    ref := *loc
    if ref != NULL and !marked(ref):
      mark(ref)
      worklist.push(ref)
      markWorklist()

markWorklist():
  while (ref := worklist.pop()):
    foreach loc in ref->header.descriptor->ptrs:
      child := *(ref+loc)
      if child != NULL and !marked(child):
        mark(child)
        worklist.push(child)
```

The mark algorithm (one version)

```
markPhase():
  worklist := new Queue
  foreach loc in roots:
    ref := *loc
    if ref != NULL and !marked(ref):
      mark(ref)
      worklist.push(ref)
      markWorklist()

markWorklist():
  while (ref := worklist.pop()):
    foreach loc in ref->header.descriptor->ptrs:
      child := *(ref+loc)
      if child != NULL and !marked(child):
        mark(child)
        worklist.push(child)
```

Root task very different from object task!

Scan order

- Presented algorithm:
  - Follows root pointers to completion before moving on to another root pointer
  - Is breadth-first for heap objects

This should make you angry!

Scan order

- Objects often form cliques
- Object cliques:
  - Are allocated around the same time
  - Mostly point at each other
  - Should be allocated near each other

Scan order: Address-first?

- We could sort worklist by ref address
- Time to sort usually overwhelms saved time scanning

Mark bit

- Without mark bit, graph reachability trace may never end!
- Mark bit can be in header…
- Or, can keep a side table
- If in header: Where to put the bit?

Mark bit

```c
struct ObjectHeader {
  struct GCTypeInfo *typeInfo;
  char markBit;
};

void mark(struct ObjectHeader *hdr) {
  hdr->markBit = 1;
}

int isMarked(struct ObjectHeader *hdr) {
  return hdr->markBit;
}
```

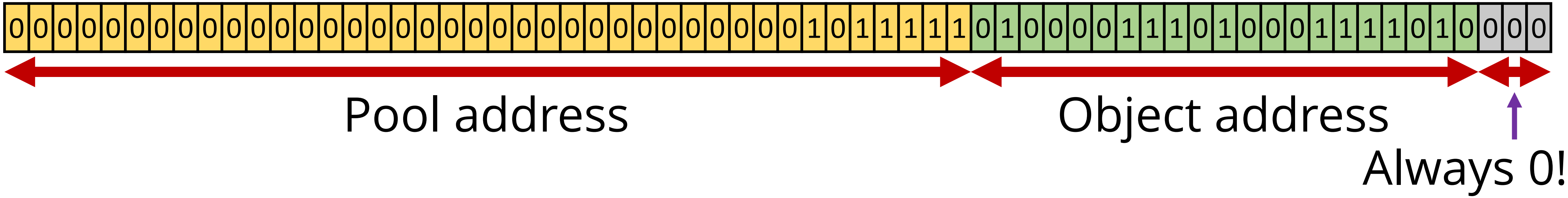
Mark bit

```
struct ObjectHeader {
   struct GCTypeInfo *typeInfo;
   char markBit;
};

void mark(struct ObjectHeader *hdr) {
   hdr->markBit = 1;
}

int isMarked(struct ObjectHeader *hdr) {
   return hdr->markBit;
}
```

How much larger are objects when this is added?

Bit-sneaky

- There are three[1] wasted bits in our header



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	0	1	0	0	0	0	1	1	1	0	1	0	0	0	1	1	1	1	0	1	0	0	0	0

- *All* pointers have this extra space!

[1] On 32-bit systems, two

Bit-sneaky C

```
struct ObjectHeader {
  struct GCTypeInfo *typeInfo;
};

void mark(struct ObjectHeader *hdr) {
  hdr->typeInfo = (struct GCTypeInfo *)
    ((size_t) hdr->typeInfo | 1);
}

int isMarked(struct ObjectHeader *hdr) {
  return (size_t) hdr->typeInfo & 1;
}
```

Worth it?

- If objects are small (hint: they are), every word counts
- Huge complication: Type info pointer is no longer valid!
- Must restore type info pointer later

Sweep

- Heap parsability is crucial!
- Consider heap parsability with:
  - Bump-pointer allocation
  - Free-list overallocation
  - Free object type/header

Sweep algorithm

```
sweep():
  freeList := new FreeList
  foreach ref in heap:
    if marked(ref):
      unmark(ref)
    else:
      ref *:= new FreeObject
      freeList.push(ref)
```

Sweep algorithm

```
sweep():
  freeList := new FreeList
  foreach ref in heap:
    if marked(ref):
      unmark(ref)
    else:
      ref *:= new FreeObject
      freeList.push(ref)
```

Discard old freelist

Sweep algorithm

```
sweep():
  freeList := new FreeList
  foreach ref in heap:
    if marked(ref):
      unmark(ref)
    else:
      ref *:= new FreeObject
      freeList.push(ref)
```

Discard old freelist

Must walk entire heap!

Sweep algorithm

```
sweep():
   freeList := new FreeList
   foreach ref in heap:
      if marked(ref):
         unmark(ref)
      else:
         ref *:= new FreeObject
         freeList.push(ref)
```

Discard old freelist

Must walk entire heap!

Perfect chance to unmark

Sweep algorithm

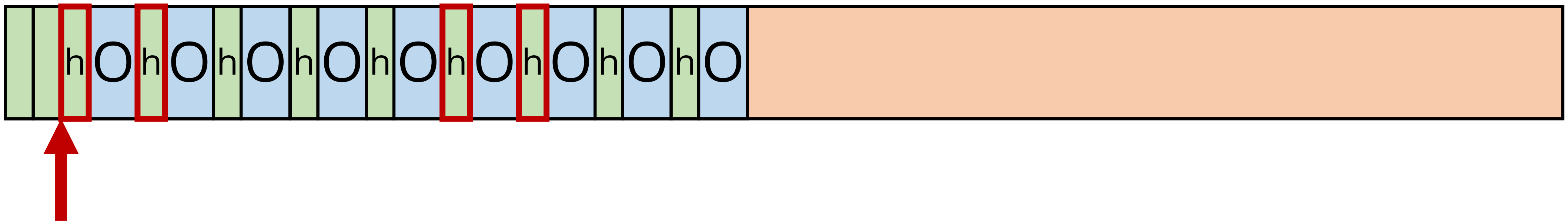
```
sweep():
  freeList := new FreeList
  foreach ref in heap:
    if marked(ref):
      unmark(ref)
    else:
      ref *:= new FreeObject
      freeList.push(ref)
```

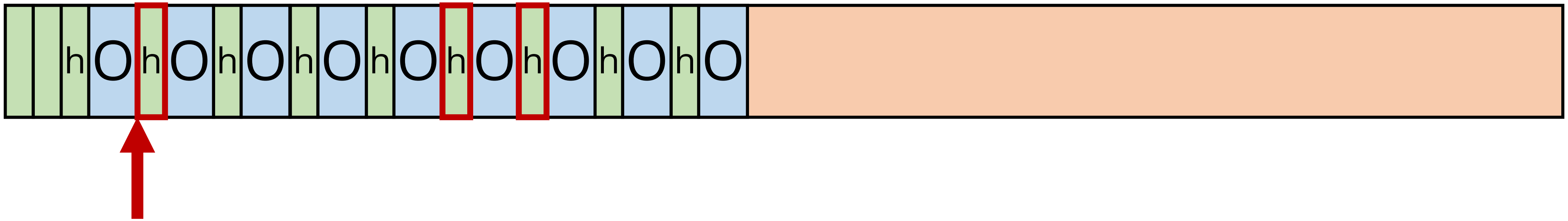
Discard old freelist

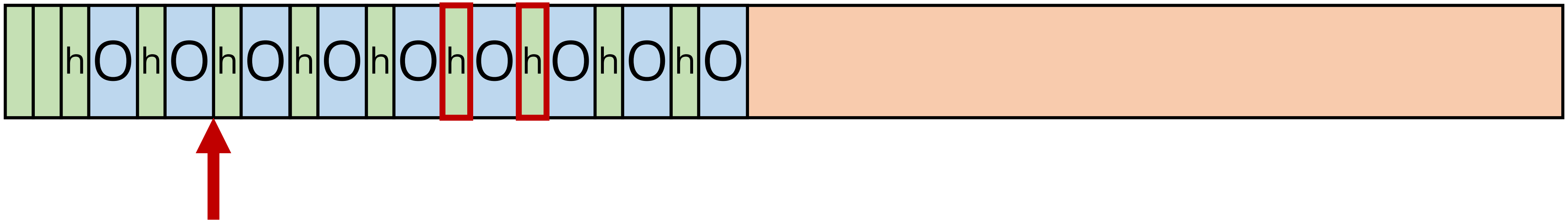
Must walk entire heap!

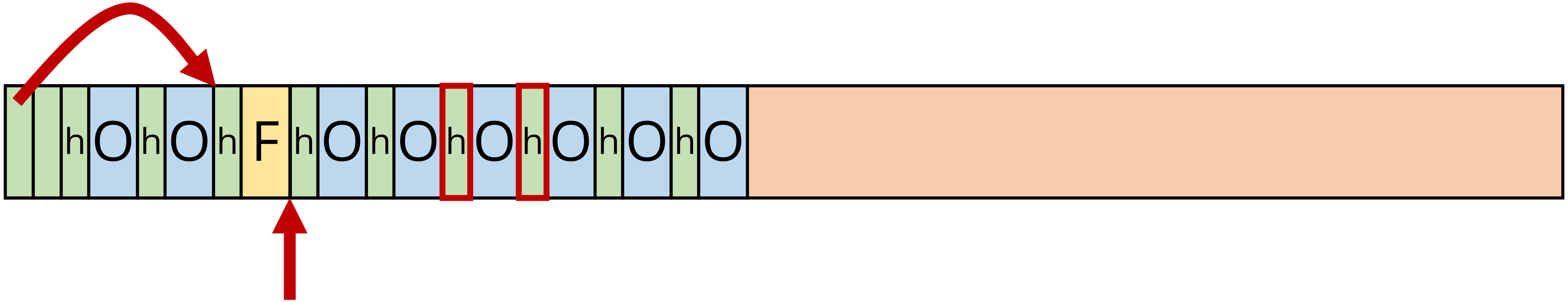
Perfect chance to unmark

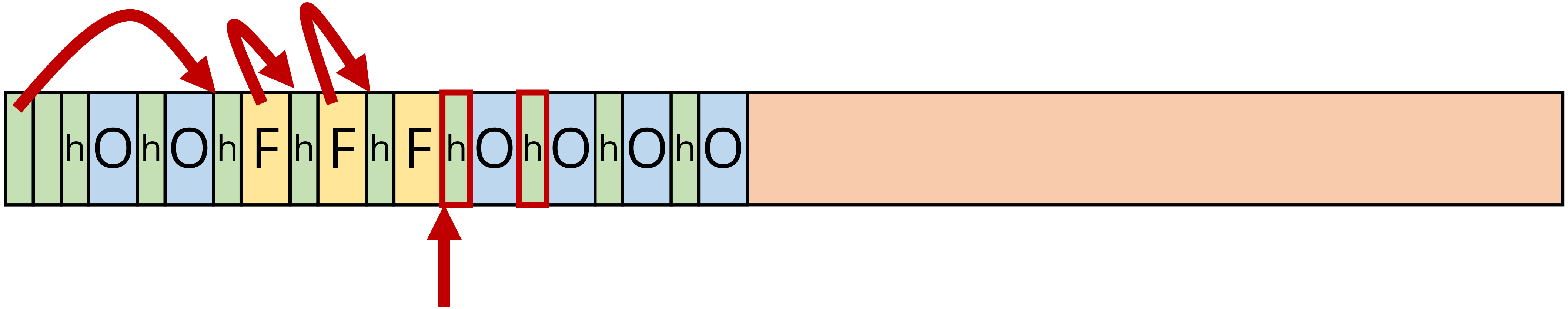
Type of objects
change in sweep

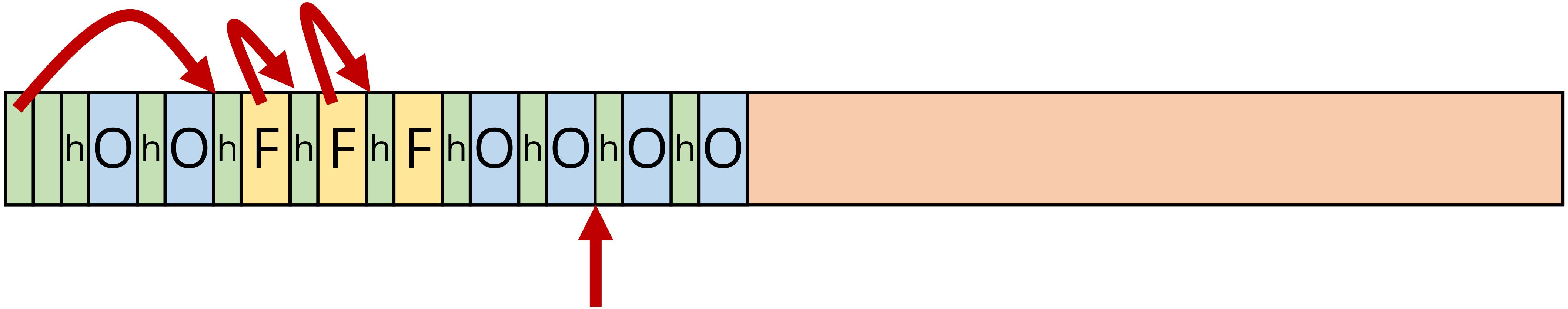

h O h O h O h O h O h O h O h O h O

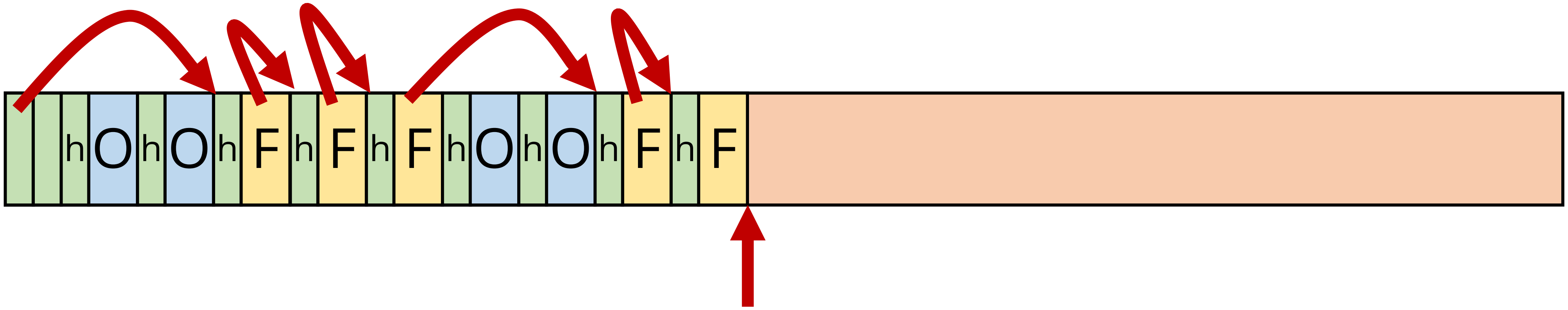

hOhOhOhOhOhOhOhOhO


h O h O h O h O h O h O h O h O h O


h
O
h
O
h
F
h
O
h
O
h
O
h
O
h
O
h
O


h
O
h
O
h
F
h
F
h
F
h
O
h
O
h
O
h
O


h
O
h
O
h
F
h
F
h
F
h
O
h
O
h
O
h
O


h
O
h
O
h
F
h
F
h
F
h
O
h
O
h
F
h
F

Performance

- Mark: O(L)
- Sweep: O(H)
- Mark-and-sweep: O(H)

Performance

- Mark: O(L)
- Sweep:
- and-sweep: O(H)

Locality, locality, locality!

Bit-swapping

- Can avoid cost of clearing bits by swapping meaning:
  - In first collection, 0 = unreachable, 1 = reachable,
  - in second collection, 1 = unreachable, 0 = reachable, etc.
- Must remember to allocate with correct mark!

Improving mark

- Depth-first vs. breadth-first vs. address-ordered
- Bitmapped mark
- Other tricks beyond scope of course

Bitmapped mark

- Connected to bitmap free-list:
  - Bitmap at beginning of pool
  - Clear bitmap before marking
  - One bit per word
  - If object is alive, mark its words in bitmap
  - Use as bitmap free-list during allocation
- With bit-swapping, *no sweep*

Improving sweep

- It’s not so bad (locality!)
- Improve by:
  - Even better cache behavior,
  - concurrent/lazy sweeping, or
  - O(1) sweep

Sweep cache behavior

- Stride of sweep always object size
- CPUs prefetch
- Object size varies
- Segregated blocks: Object size constant, perfect prefetch

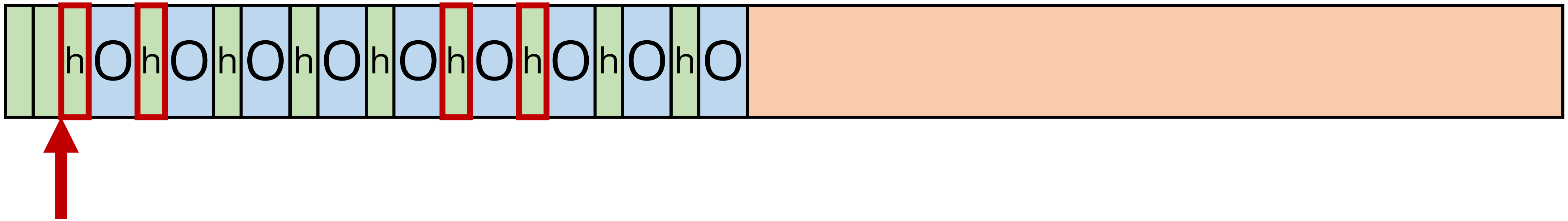
Concurrent sweep

- Mutator will never touch unmarked objects
- Sweep in a separate thread
- Must be careful about allocation/sweep races!

Lazy sweep

- Sweep during allocation
- If free-list is empty, sweep until sufficient free object is found
- Insufficient objects added to free-list

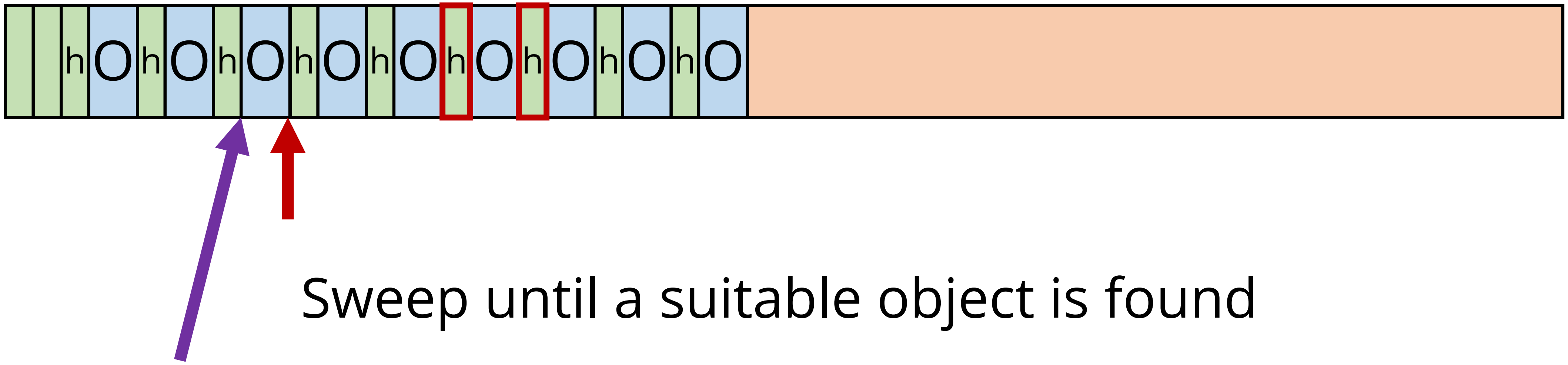
Lazy sweep



Sweep pointer maintained per pool

When allocating, if free-list is empty or has no suitable objects…

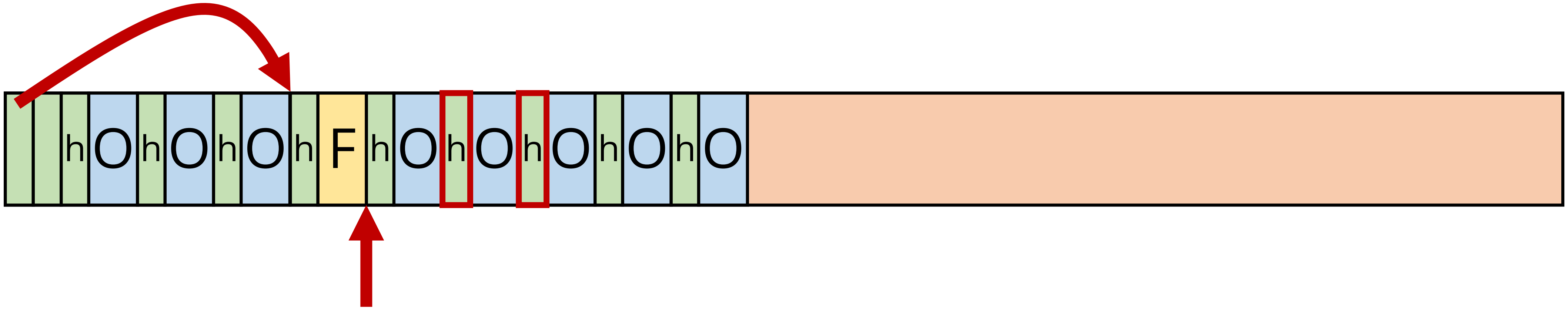
Lazy sweep



Sweep until a suitable object is found

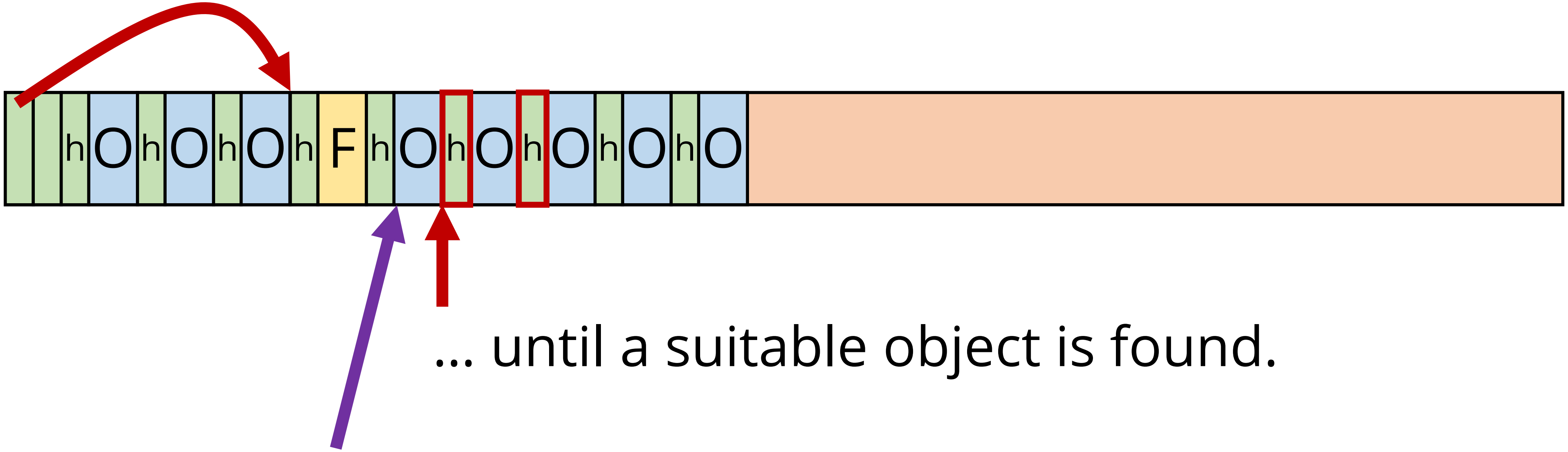
This object is returned to mutator

Lazy sweep



Unsuitable objects added to free-list during allocation

Lazy sweep



… until a suitable object is found.

This object is returned to mutator

Lazy sweep performance

- Throughput
- Responsiveness
- Latency

- Resource utilization
- Fairness