Garbage Collection

CS4410: Spring 2013

Modern Languages

- Represent all records (tuples, objects, etc.) using pointers.
  - Makes it possible to support *polymorphism*.
  - e.g., ML doesn't care whether we pass an integer, two-tuple, or record to the identity function: they are all represented with 1 word.
  - Price paid: lots of loads/stores…
- By default, allocate records on the heap.
  - Programmer doesn't have to worry about lifetimes.
  - Compiler may determine that it's safe to allocate a record on the stack instead.
  - Uses a garbage collector to safely reclaim data.
  - Because pointers are *abstract*, has the freedom to re-arrange the data in the heap to support compaction.

Allocation in SML/NJ

- Reserve two registers:
  - allocation pointer (like stack pointer)
  - limit pointer
- To allocate a record of size n:
  - checks that limit-alloc > n. If not, invokes garbage collector.
  - Adds n+1 to the alloc pointer, returns old value of alloc pointer as result.
  - Extra word holds meta-data (e.g., size.)
  - Actually, amortizes the limit check across a bunch of allocations (just as we amortize stack pointer adjustment.)
  - Result: 3-5 instructions to allocate a record.

Garbage Collection:

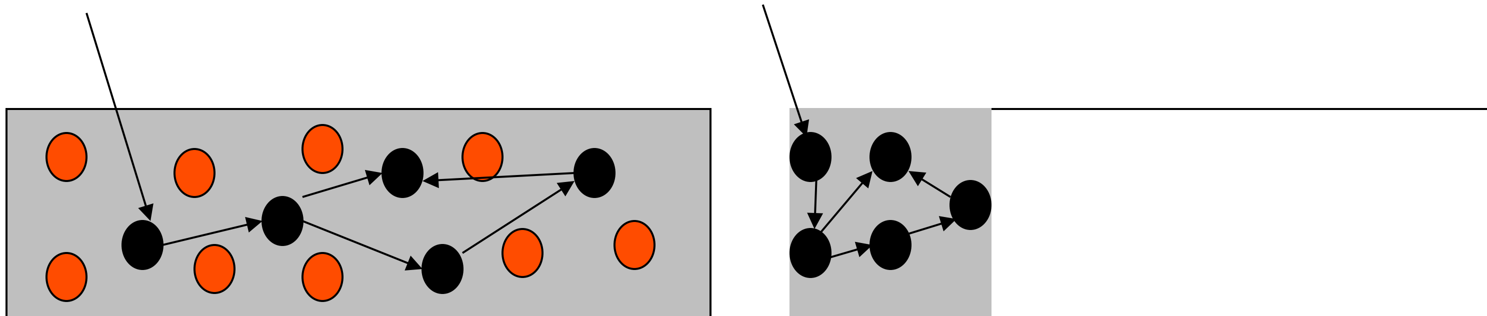
- Starting from stack, registers, & globals (roots), determine which objects in the heap are reachable following pointers.
- Reclaim any object that isn't reachable.
- Requires being able to distinguish pointer values from other values (e.g., ints).
  - SML/NJ uses the low bit:
    1 it's a scalar, 0 it's a pointer.
  - In Java, we use put the tag bits in the meta-data.
  - For BDW collector, we use heuristics:
    (e.g., the value doesn't point into an allocated object.)

Mark/Sweep Traversal:

- Reserve a mark-bit for each object.
- Starting from roots, mark all accessible objects.
- Stick accessible objects into a queue or stack.
  - queue: breadth-first traversal
  - stack: depth-first traversal
- Loop until queue/stack is empty:
  - remove marked object (say x).
  - if x points to an (unmarked) object y, then mark y and put it in the queue.
- Run through all objects:
  - If they haven't been marked, put them on the free list.
  - If they have been marked, clear the mark bit.

Copying Collection:

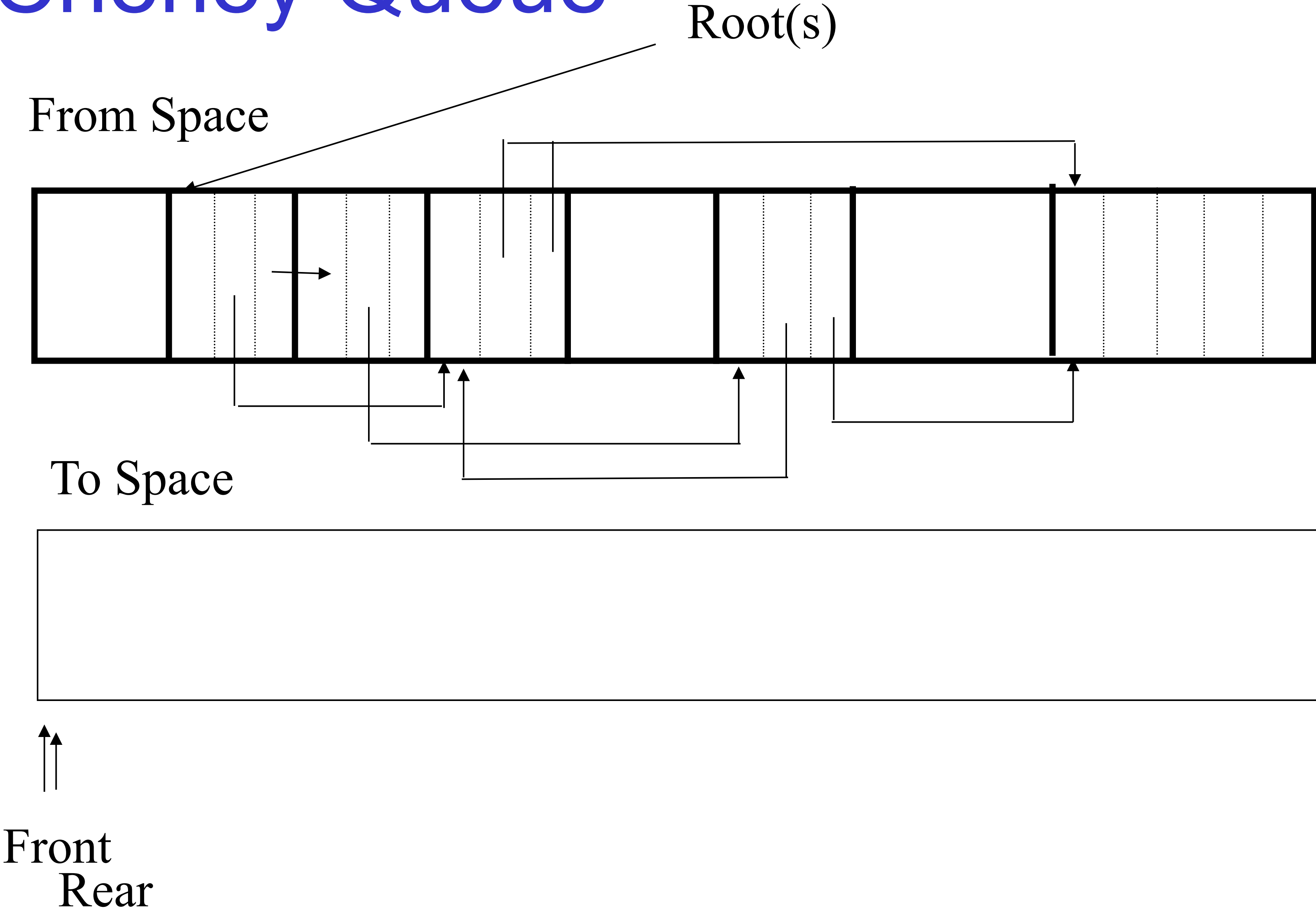
- Split data segment into two pieces.
- Allocate in 1st piece until it fills up.
- Copy the reachable data into the 2nd area, compressing out the holes corresponding to garbage objects.



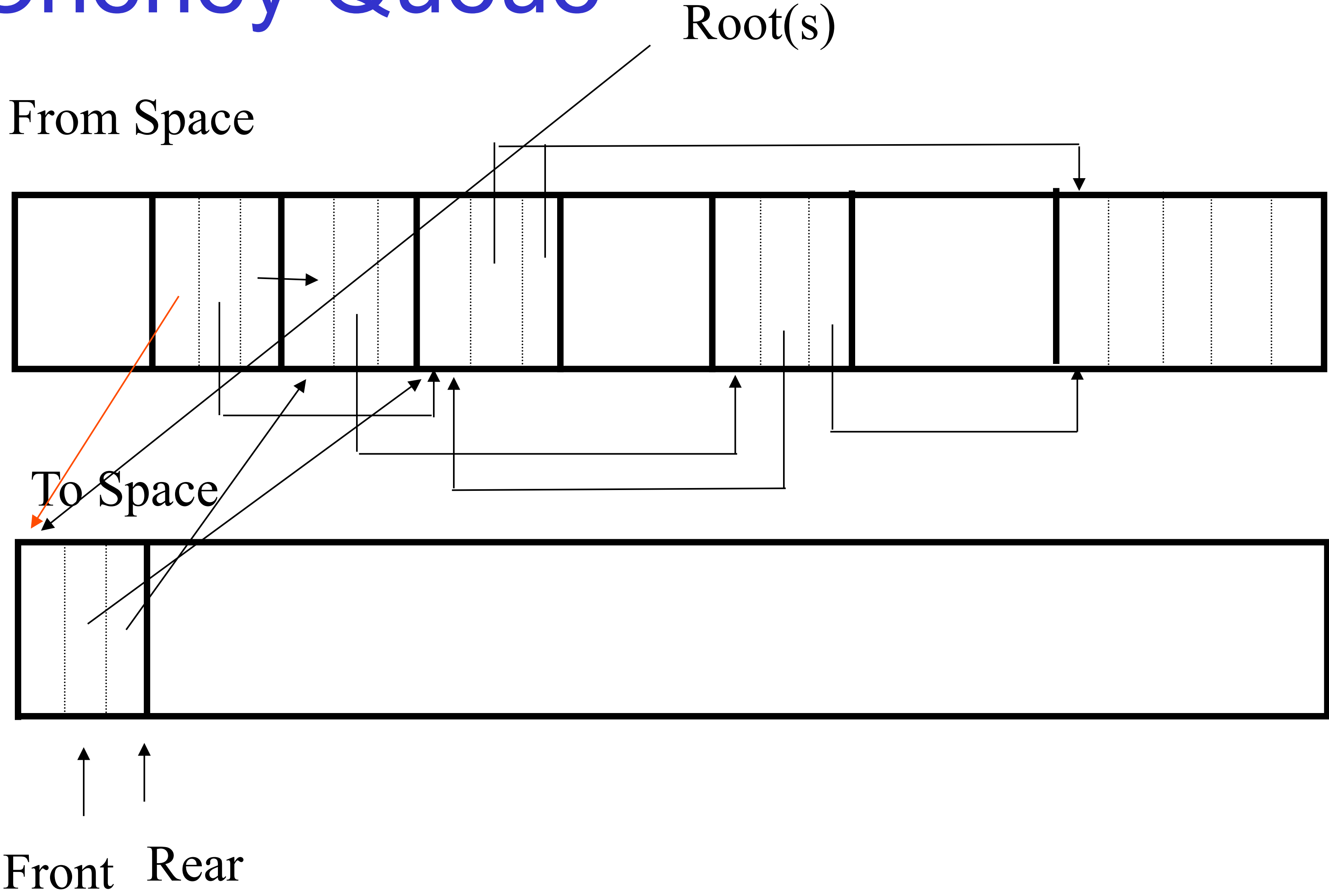
Algorithm: Queue-Based

- Initialize front/rear to beginning of to-space.
  - A trick for representing the queue using the to-space.
- Enqueue the items pointed to by roots.
  - Copy the objects into to-space (bump rear pointer).
  - Place a *forwarding pointer* in the old copy that points to the new copy.
- While queue is not empty:
  - Dequeue a word (i.e., bump front pointer).
  - If the word is a pointer to an unforwarded object, then enqueue the object and set its forwarding pointer.
  - If the word is a pointer to a forwarded object, overwrite the word with the address of the new copy.

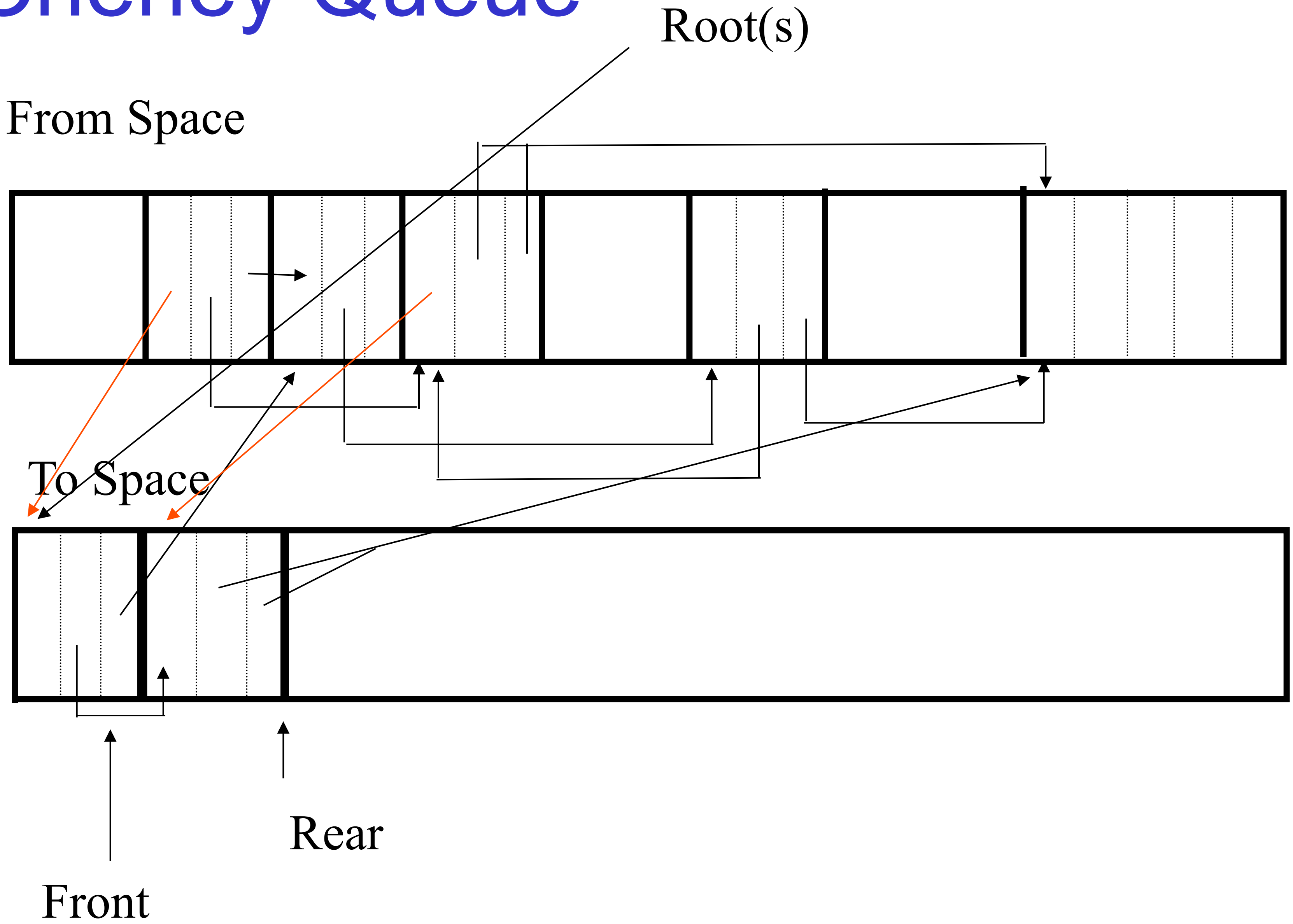
Cheney Queue



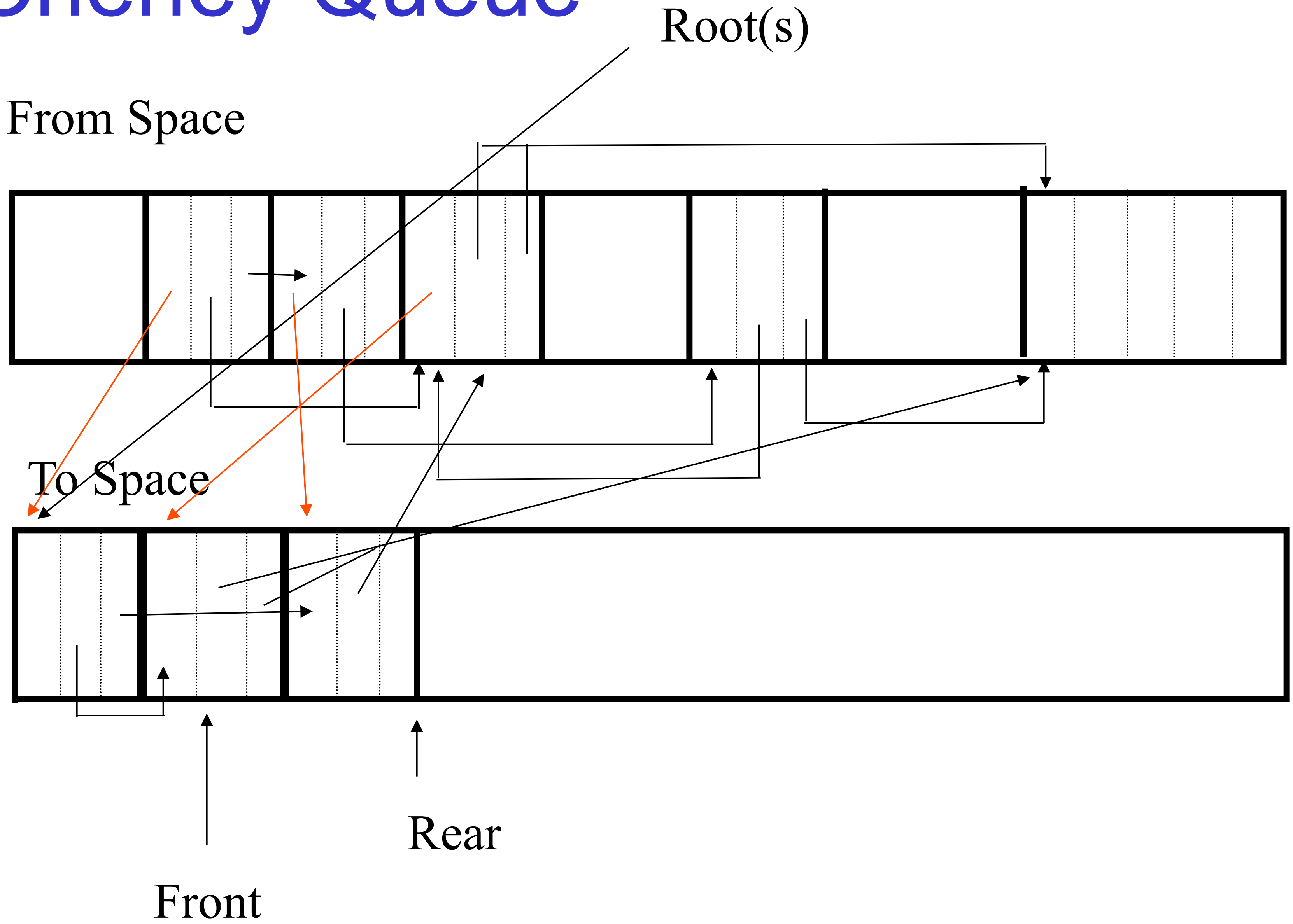
Cheney Queue



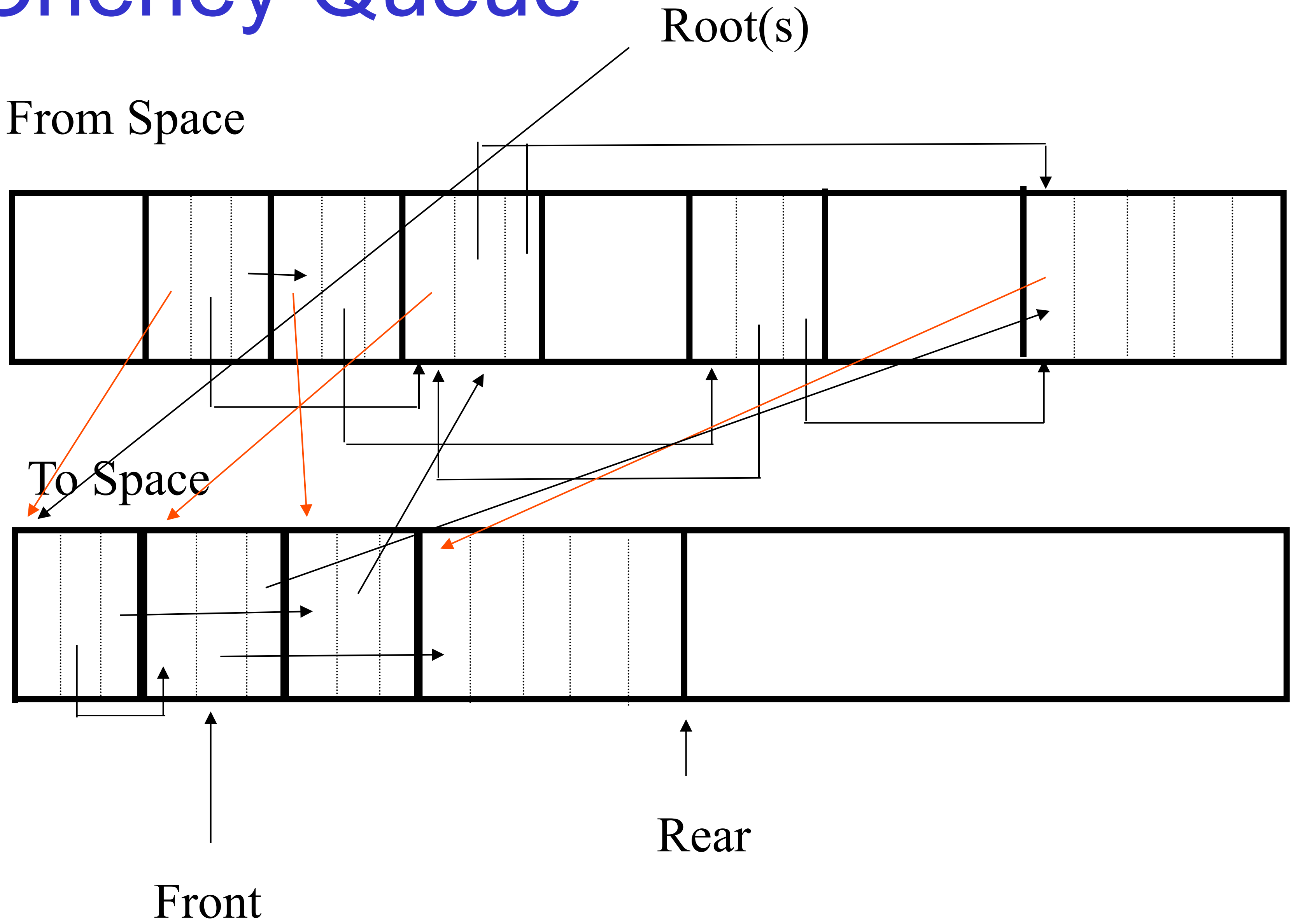
Cheney Queue



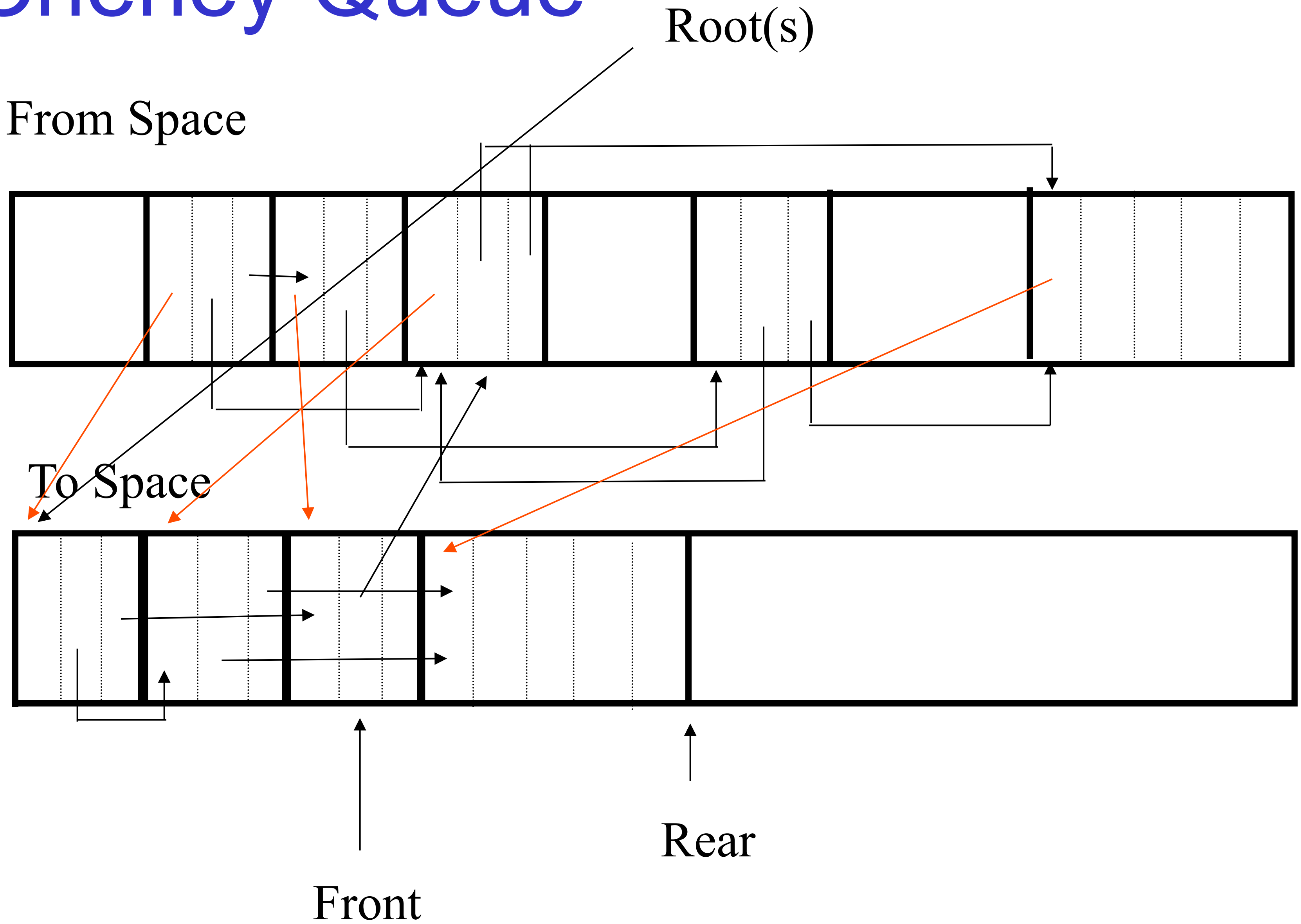
Cheney Queue



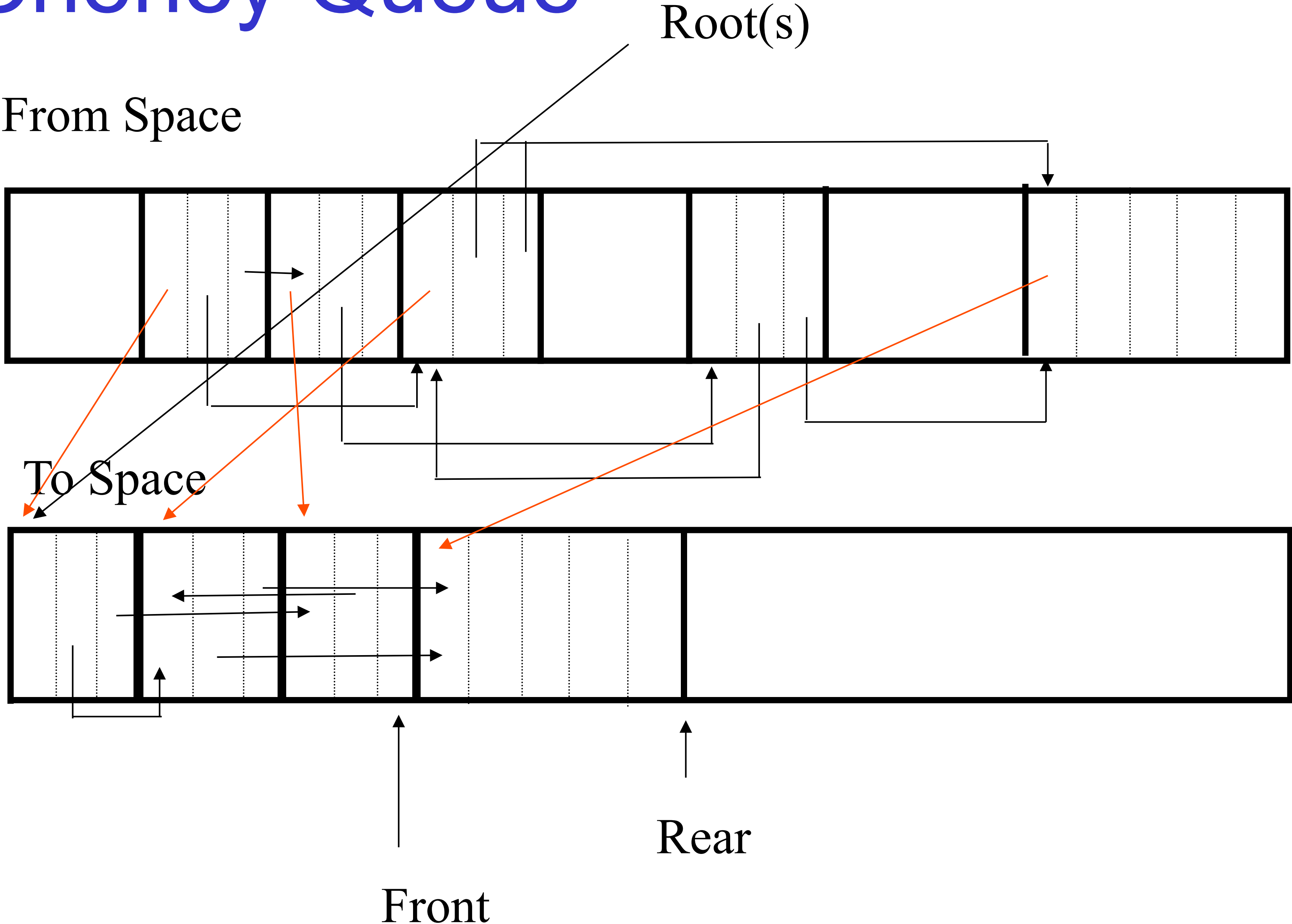
Cheney Queue



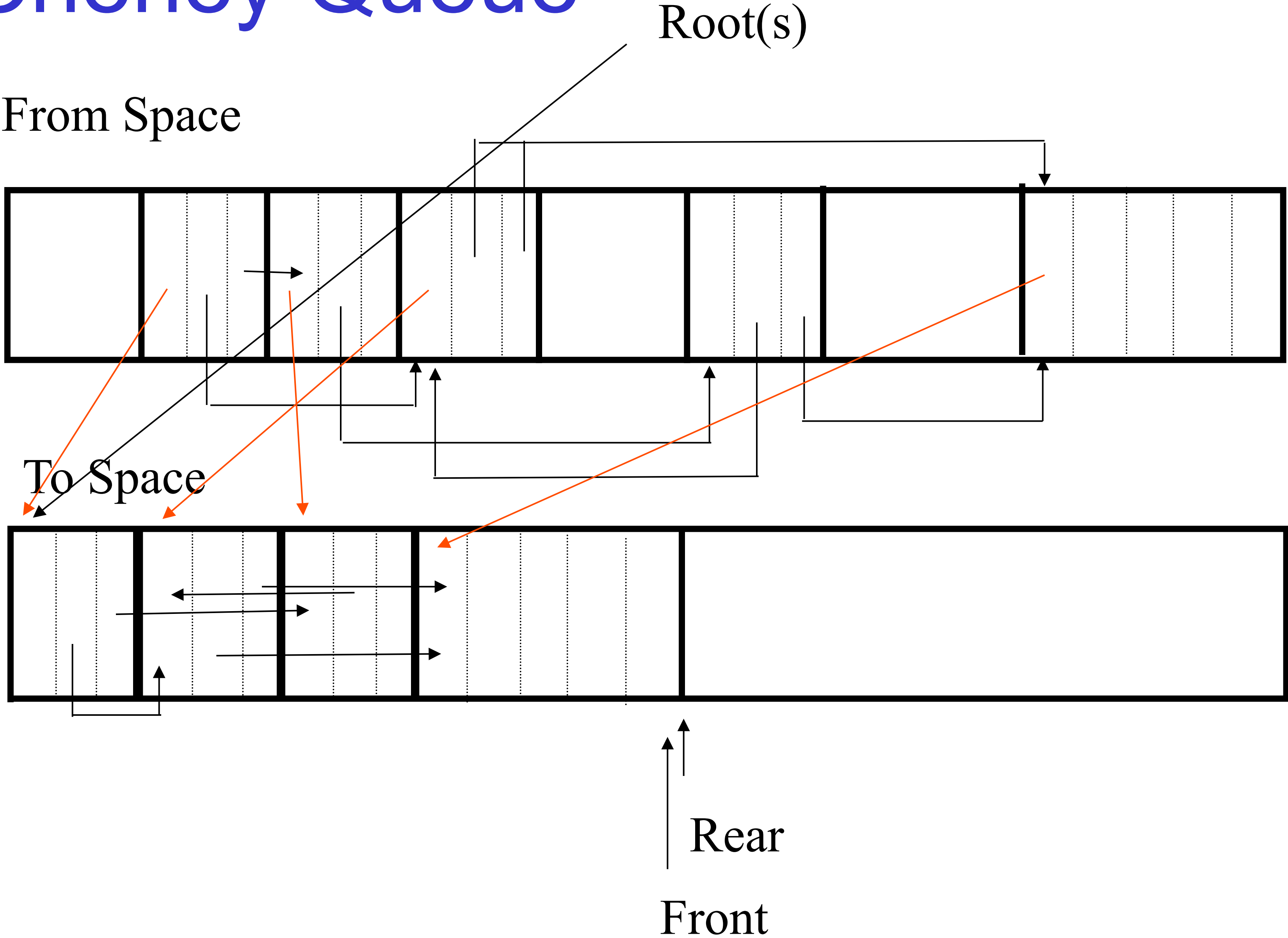
Cheney Queue



Cheney Queue



Cheney Queue

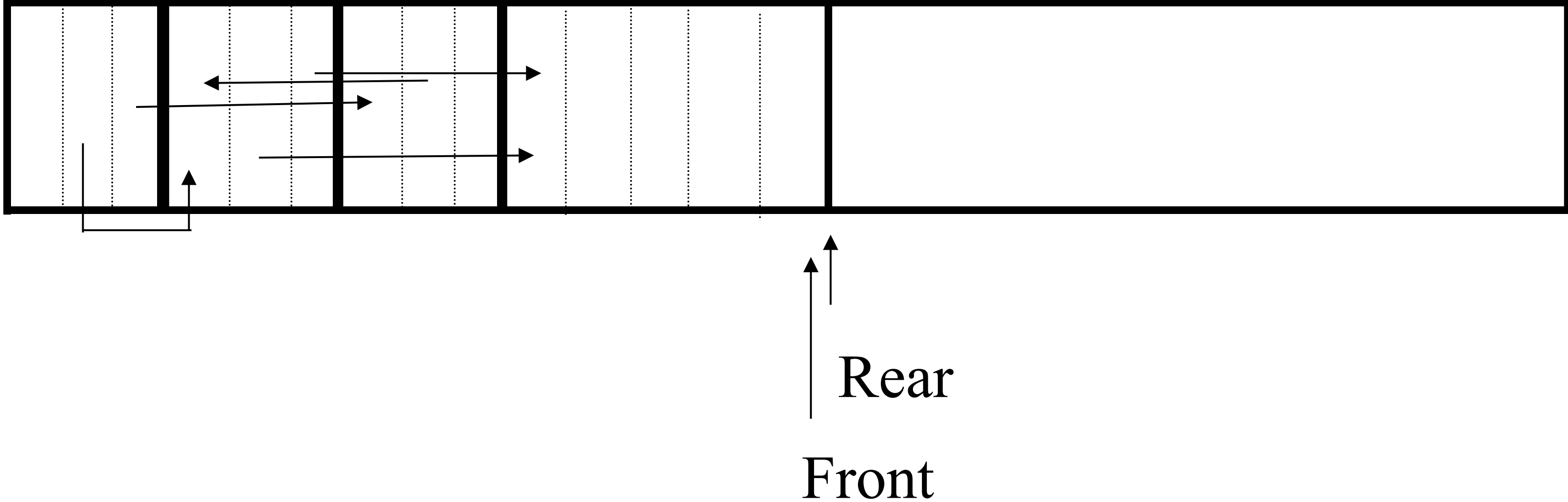


Cheney Queue

Root(s)

From Space

To Space



Pros and Cons:

- Pros:
  - Fast, bump-pointer allocation.
  - Cost of GC is proportional to live data (not all of memory).
  - Compaction happens for free.
- Cons:
  - Long pauses.
  - Memory cut in half.
  - Lots of memory traffic.

Reality:

- Techniques such as *generational* or *incremental* collection can greatly reduce latency.
  - A few millisecond pause times.
- Large objects (e.g., arrays) can be copied in a "virtual" fashion without doing a physical copy.
- Some systems use a mix of copying collection (young data) and mark/sweep (old data) with support for compaction.
- A real challenge is scaling this to server-scale systems with terabytes of memory…
- Interactions with OS matter a lot: cheaper to do GC than it is to start paging…

Conservative Collectors:

- Work without help from the compiler.
  - e.g., legacy C/C++ code.
  - e.g., your compiler :-)
- Cannot accurately determine which values are pointers.
  - But can rule out some values (e.g., if they don't point into the data segment.)
  - So they must conservatively treat anything that looks like a pointer as such.
  - Two bad things result: leaks, can't move.
  - Further problems if pointers are "hidden".

The BDW Collector

- Based on mark/sweep.
  - performs sweep lazily
- Organizes free lists as we saw earlier.
  - different lists for different sized objects.
  - relatively fast (single-threaded) allocation.
- Most of the cleverness is in finding roots:
  - global variables, stack, registers, etc.
- And determining values aren't pointers:
  - blacklisting, etc.