Copying Garbage Collection

- Garbage collection is implicit.
  - Moves all of the live objects into one area.
  - The rest of the heap is then known to contain only garbage, and is therefore available for allocation.
  - Also called scavenging.
- Copying collectors integrate the two phases, so that most objects need to be traversed once only.
  - Objects are moved to the contiguous destination area as they are reached by this *copying traversal*.
  - The work is proportional to the amount of live data, all of which needs to be copied.

“Stop-and-Copy” Using Semispaces

- Split the heap into two contiguous semispaces, the *from-space* and the *to-space*.
  - Set the from-space to be the *current semispace*.

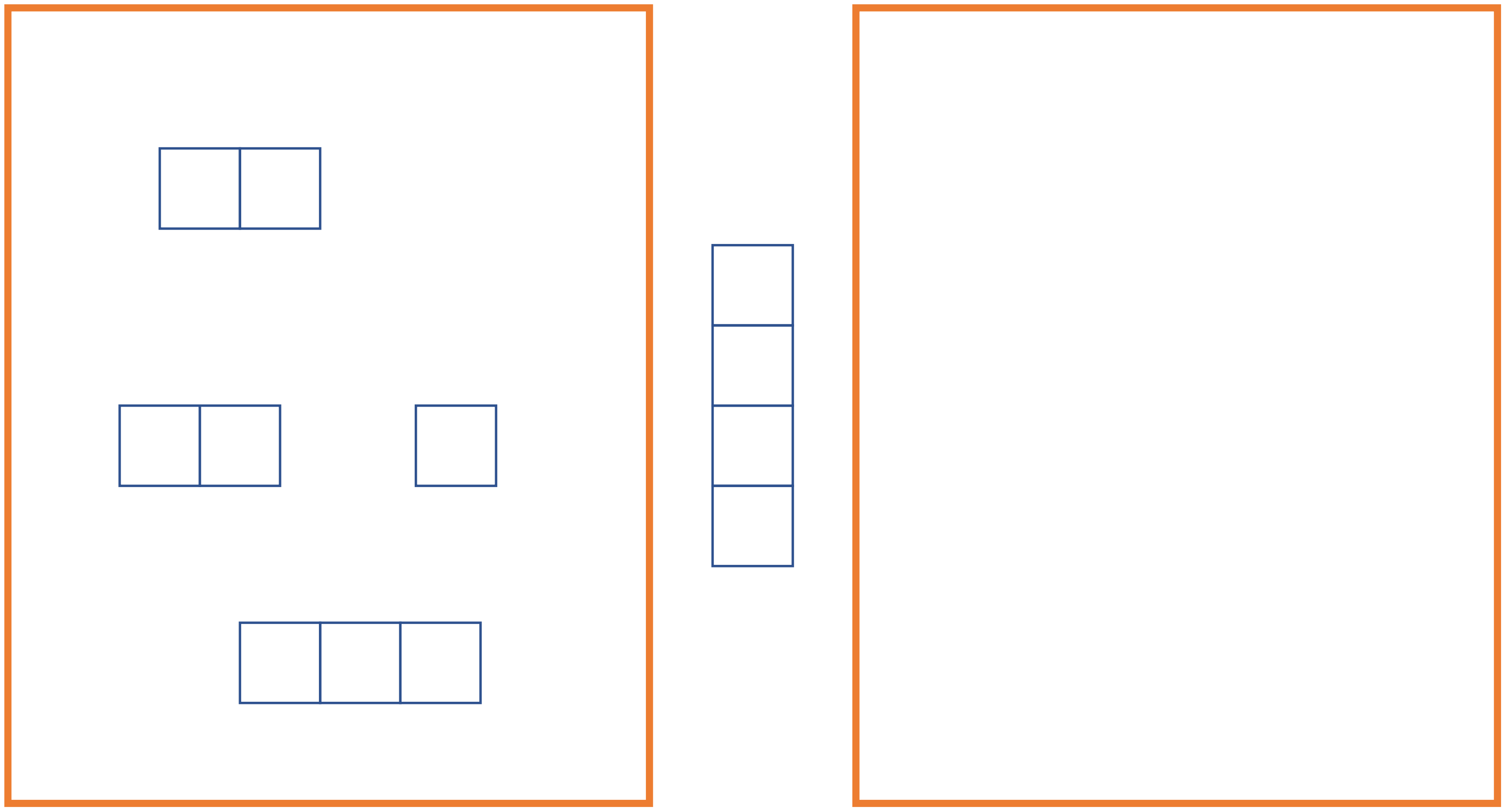
“Stop-and-Copy” Using Semispaces

- Split the heap into two contiguous semispaces, the *from-space* and the *to-space*.
  - Set the from-space to be the *current semispace*.
- Use only one of the semispaces at a time.
  - Allocate memory in the current semispace as requested by the mutator.
  - Can even be as simple as a bump allocator.

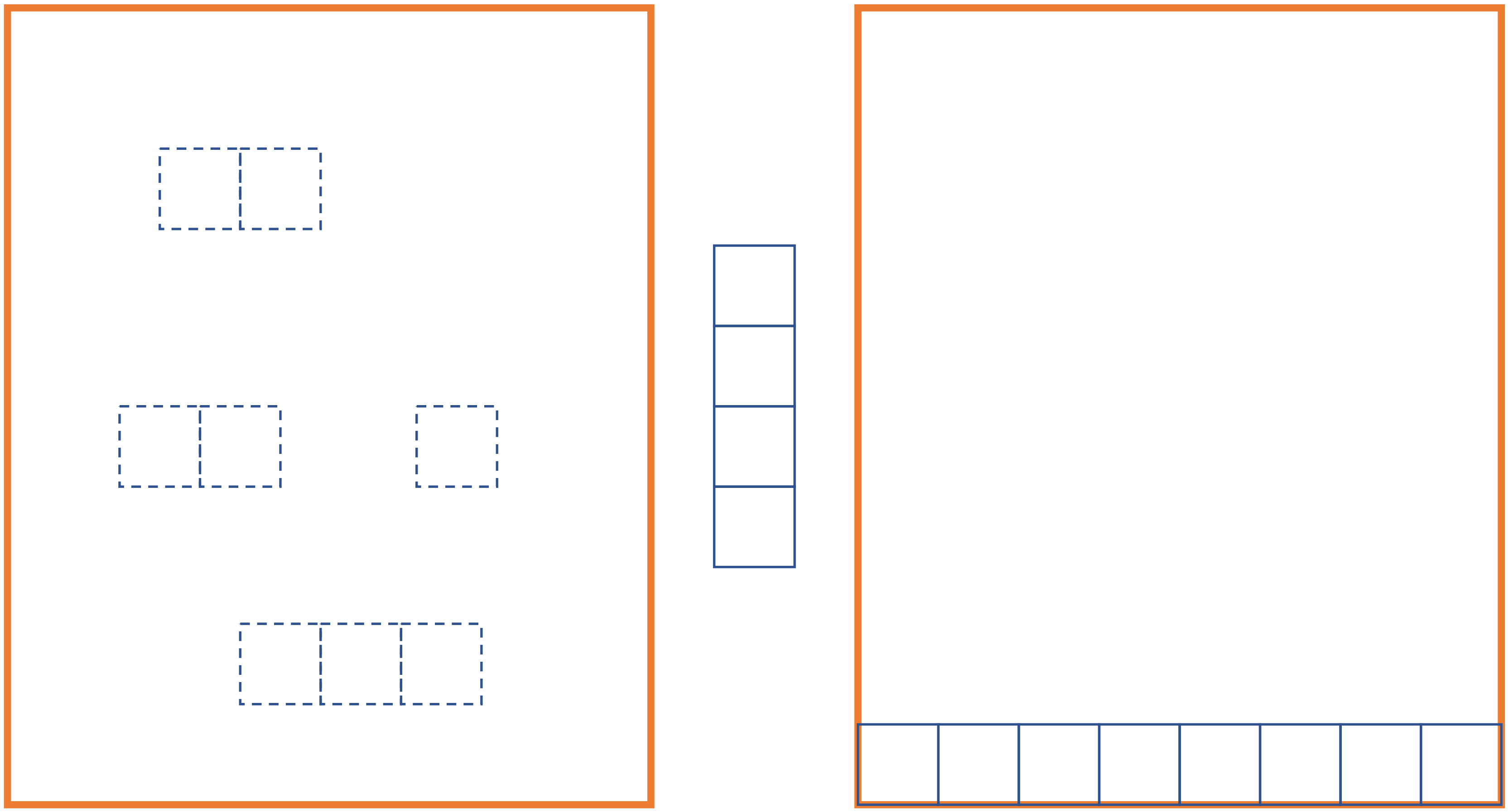
“Stop-and-Copy” Using Semispaces

- Split the heap into two contiguous semispaces, the *from-space* and the *to-space*.
  - Set the from-space to be the *current semispace*.
- Use only one of the semispaces at a time.
  - Allocate memory in the current semispace as requested by the mutator.
  - Can even be as simple as a bump allocator.
- Invoke the garbage collector when the mutator makes an allocation request that overflows the from-space.
  - Copy the live data from the from-space to the to-space.
  - After copying is complete, make the to-space the current semispace, and retry the allocation.

Semispace Collector: Before Collection



Semispace Collector: After Collection



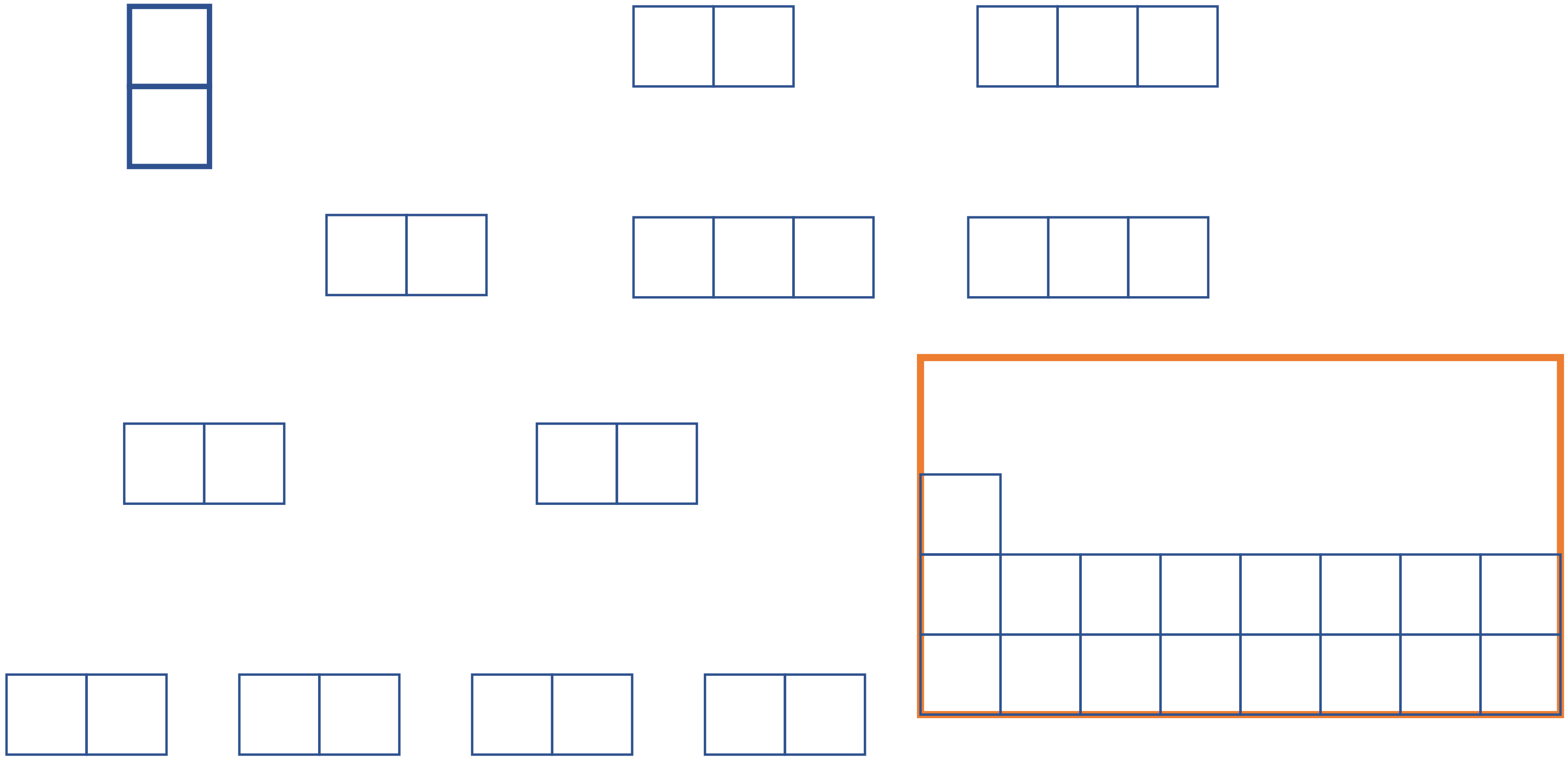
Copying Traversal: The Cheney Algorithm

- Copy the immediately-reachable objects into to-space and form the initial queue of objects for a breadth-first traversal.
  - Maintain a *scan* pointer and a *free* pointer in the to-space.
- Advance the *scan* pointer location by location through the first object.
  - For every pointer into from-space that is encountered, copy the referent into to-space at the tail of the queue, update the free pointer, and adjust the pointer to point to this new object location.
- Continue this copying traversal until *scan* and *free* point to the same location.
  - At this point, all of the objects that have been reached and copied have also been scanned for descendants.
  - This implies that the scavenging process is complete.

Copying Traversal: The Cheney Algorithm

- Copy the immediately-reachable objects into to-space and form the initial queue of objects for a breadth-first traversal.
    - Maintain a *scan* pointer and a *free* pointer in the to-space.
- Advance the *scan* pointer location by location through the first object.
    - For every pointer into from-space that is encountered, copy the referent into to-space at the tail of the queue, update the free pointer, and adjust the pointer to point to this new object location.
- Continue this copying traversal until *scan* and *free* point to the same location.
    - At this point, all of the objects that have been reached and copied have also been scanned for descendants.
    - This implies that the scavenging process is complete.
- In reality, need a slightly more complex process to ensure that an object reached by multiple paths is not copied to to-space multiple times.

The Cheney Algorithm: Example



Efficiency of Copying Collection

- Can be made arbitrarily efficient if sufficient memory is available.
    - The work done at each collection is proportional to the amount of live data at the time.
    - If approximately the same amount of data is live at any given time during the mutator’s execution, then decreasing the frequency of garbage collections will decrease the total effort.
    - A simple way to accomplish this is to make the semispaces bigger.