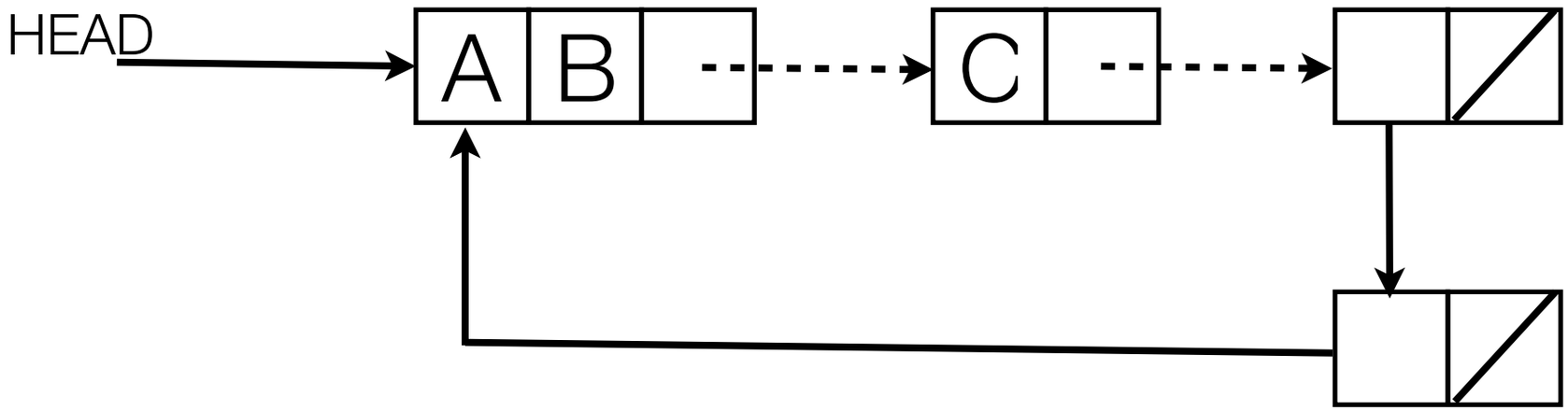
A Non-recursive List Compacting Algorithm

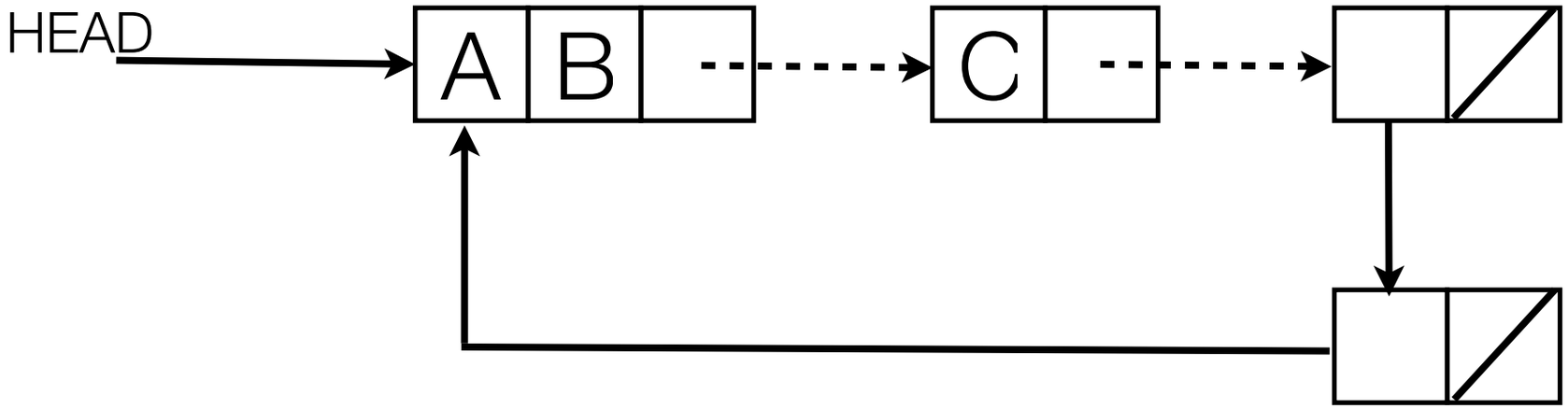
C. J. CHENEY

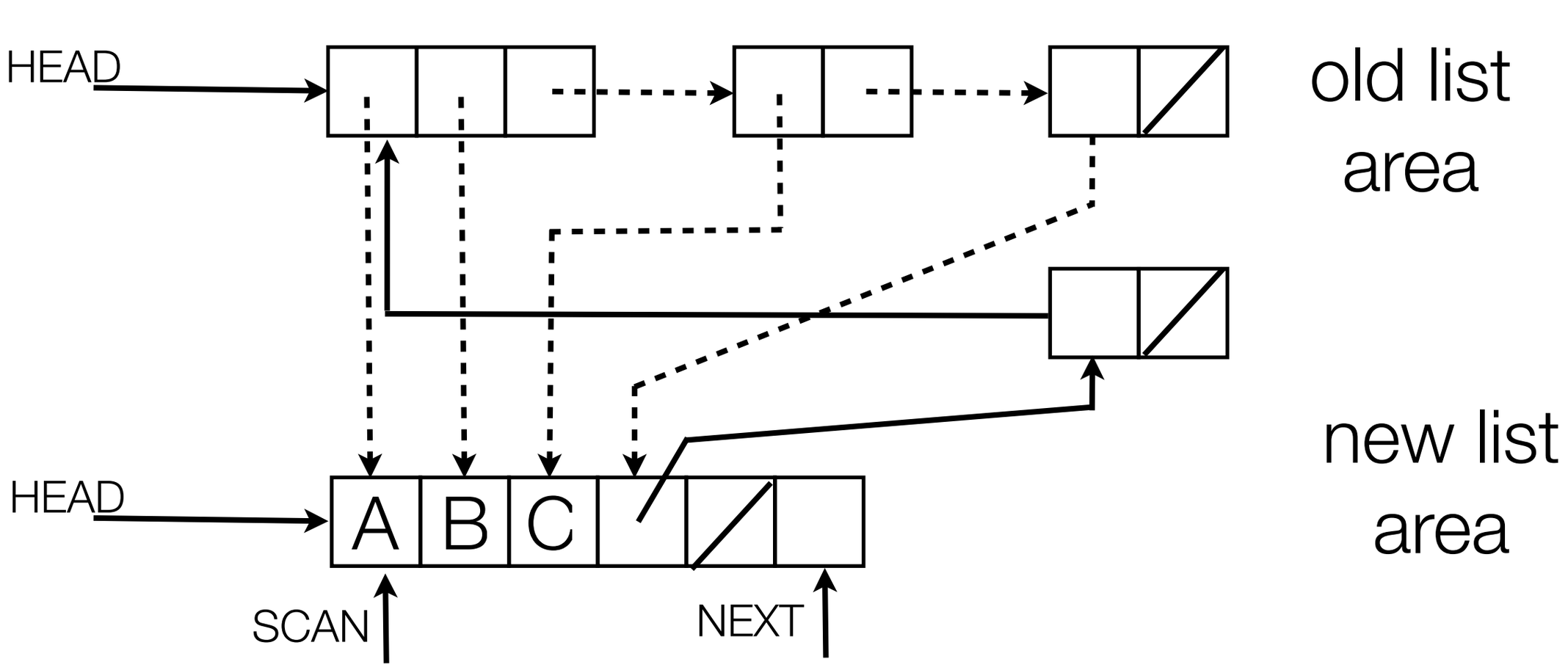
Introduction

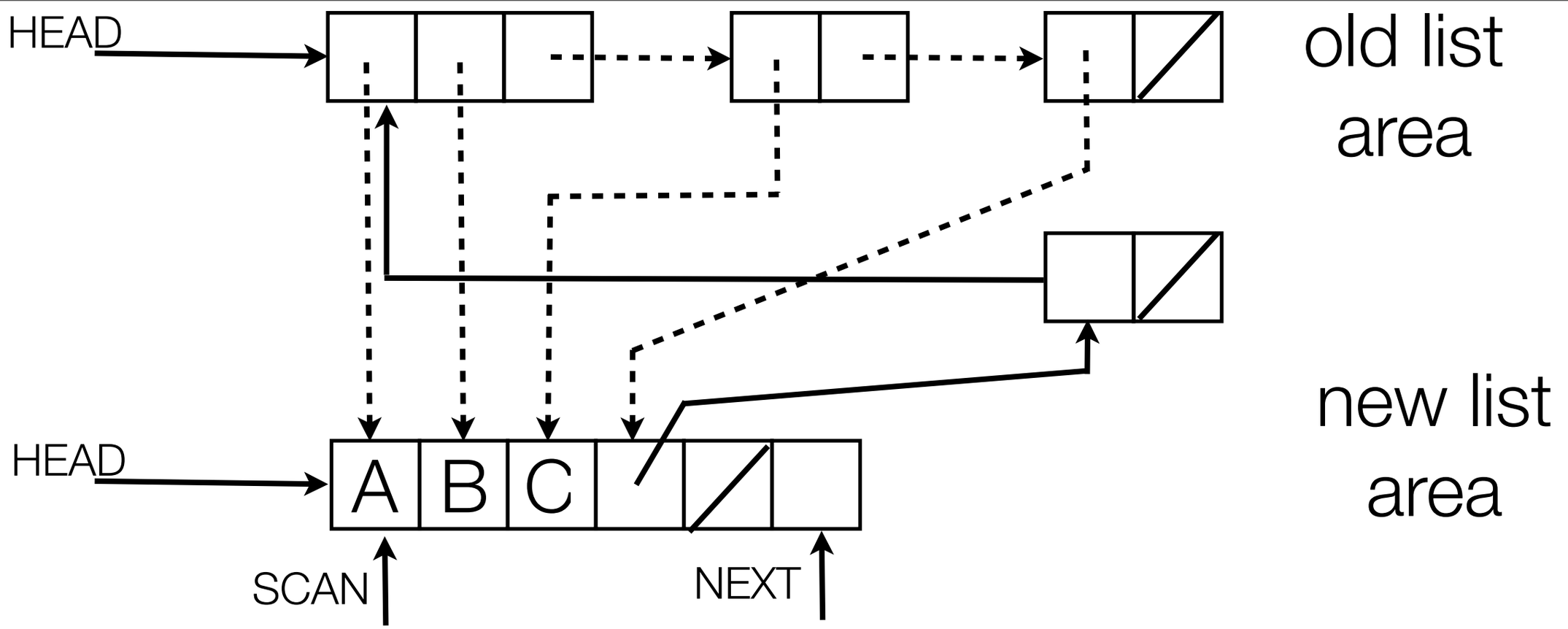
- Recursive list compacting algorithms presented by
  - Hansen
  - Fenichel and Yochelson
- Chenny presents a Non-recursive list compacting algorithm
  - The function COPYLIST copies list in the CDR direction
  - List pointers copied without transformation
  - Perform a linear scan of the new list area
  - When a list pointer is encountered invoke COPYLIST to copy the sublist

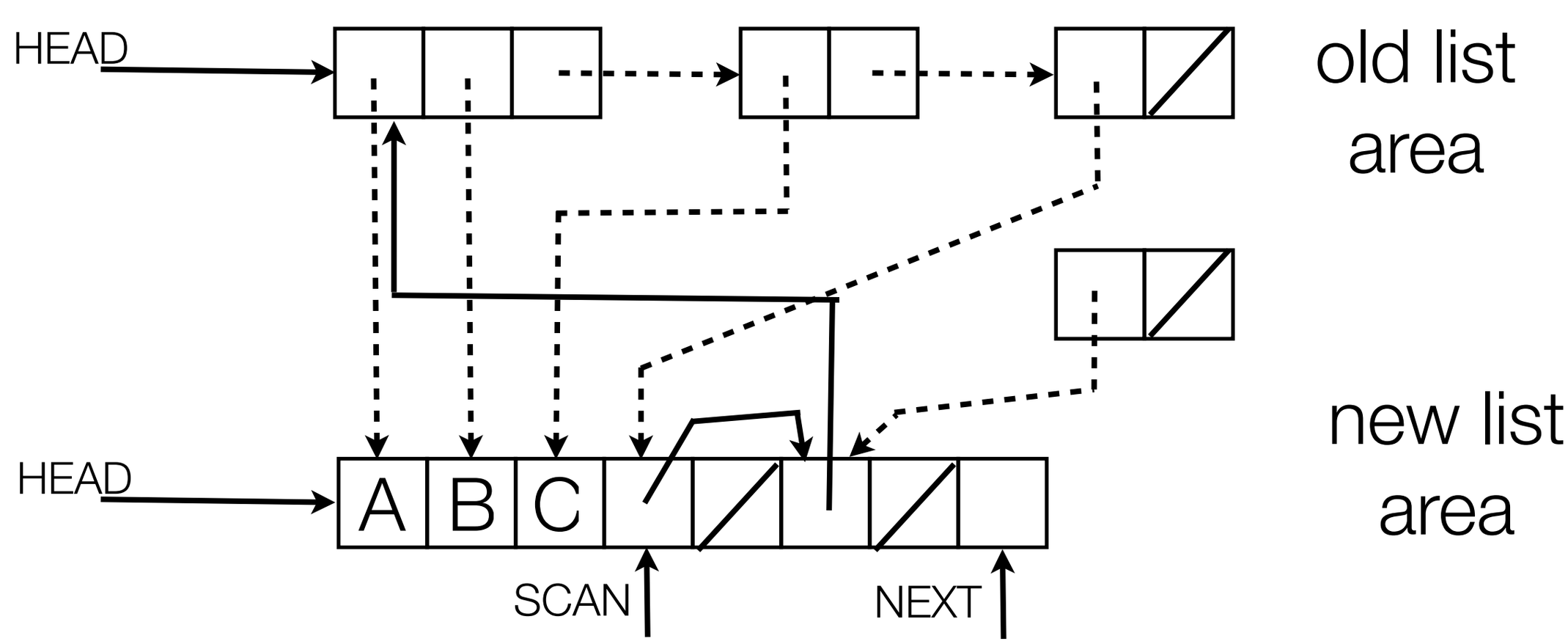
Cheney's Algorithm

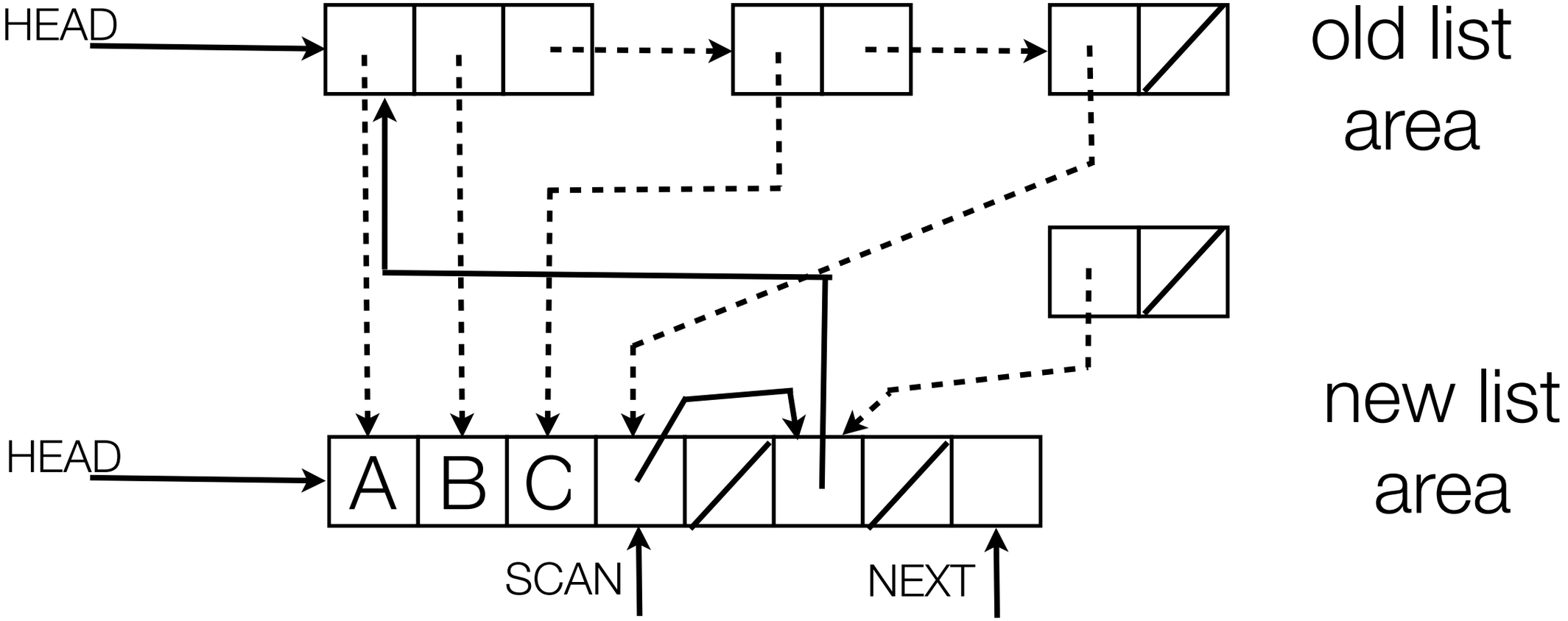


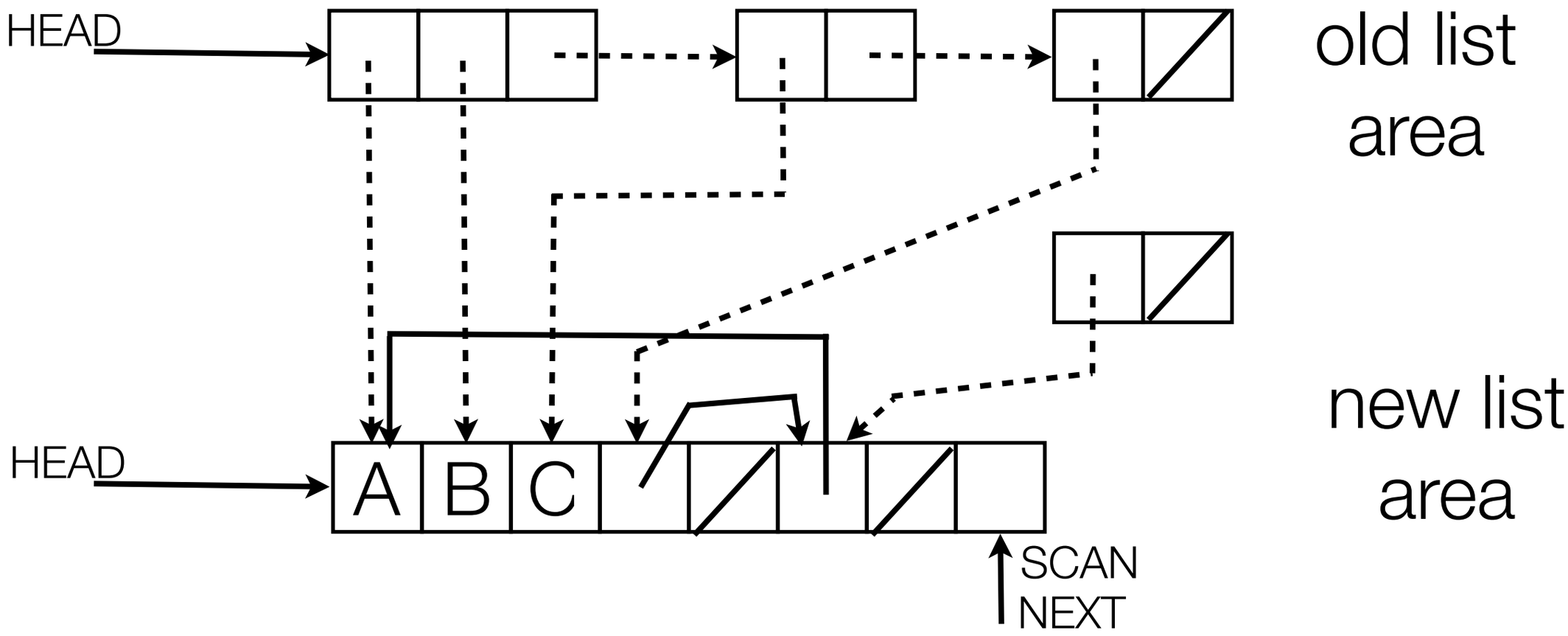

HEAD
A
B
C


HEAD
old list
area
new list
area
HEAD
A
B
C
SCAN
NEXT


HEAD
old list
area
HEAD
A
B
C
new list
area
SCAN
NEXT


HEAD
old list
area
HEAD
A
B
C
new list
area
SCAN
NEXT


HEAD
old list
area
HEAD
A
B
C
SCAN
NEXT
new list
area


HEAD
old list
area
HEAD
A
B
C
SCAN
NEXT
new list
area

Questions?

- What's the importance in having a non-recursive algorithm?

List Processing in Real Time on a Serial Computer

Henry G. Baker Jr.

Problem Statement

- Three main problems with list processing systems
  - Usually interpreted hence slow
  - Used inefficient storage structure
  - Long pauses for GC (Could be days for large Database programs)
- First two issues can be fixed by compiling
- Paper targets third problem

Solution

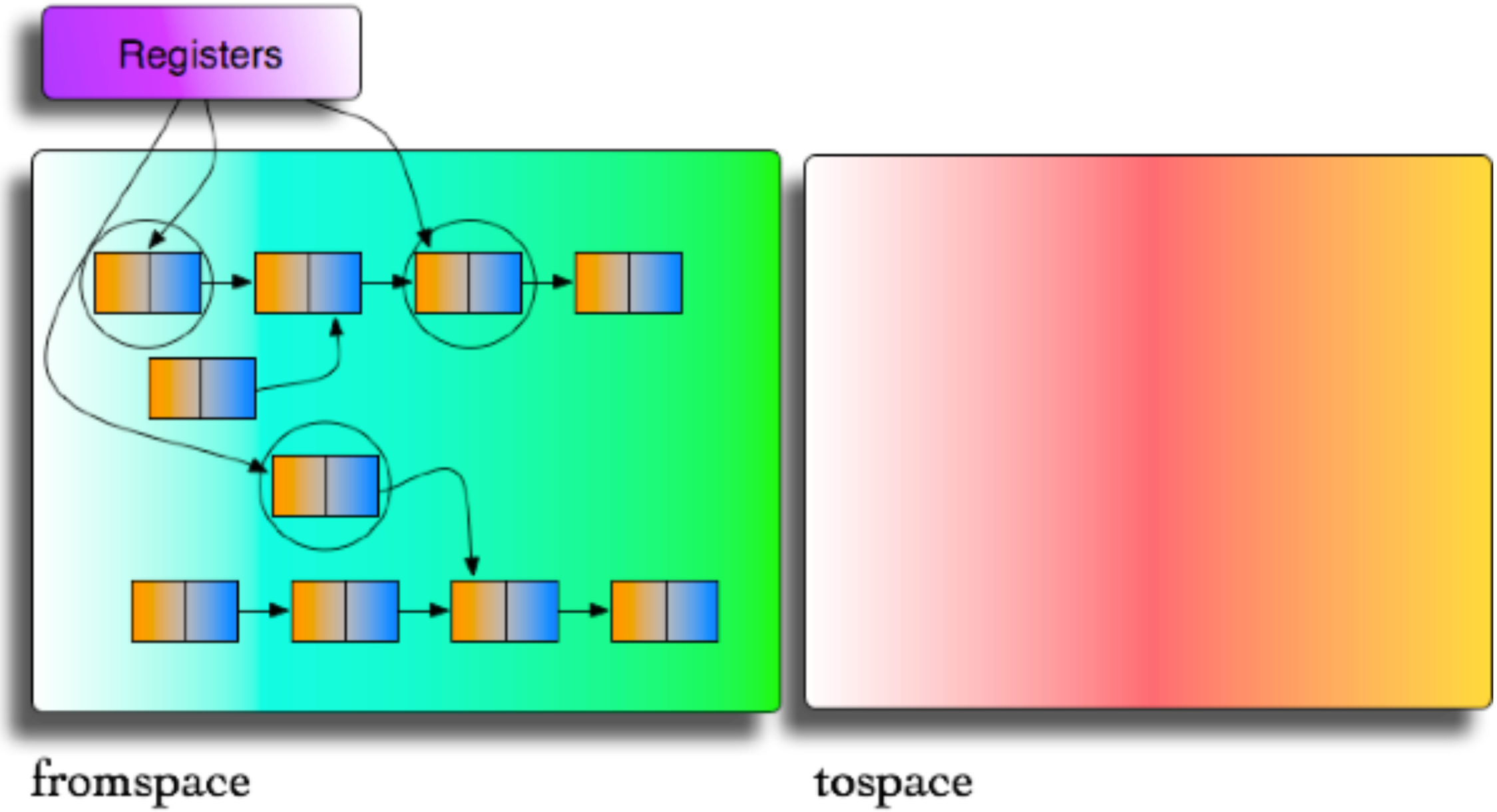
- Baker's algorithm (SRT - Serial Real-Time)
- Based on MFYCA's (Minsky-Feinchel-Yochelson-Cheney-Arnborg) algorithm

- Basic idea: Do a little copying during each cons, rather than a lot of copying infrequently
- Real-time: all operations in O(1) time
- Pretty good space efficiency

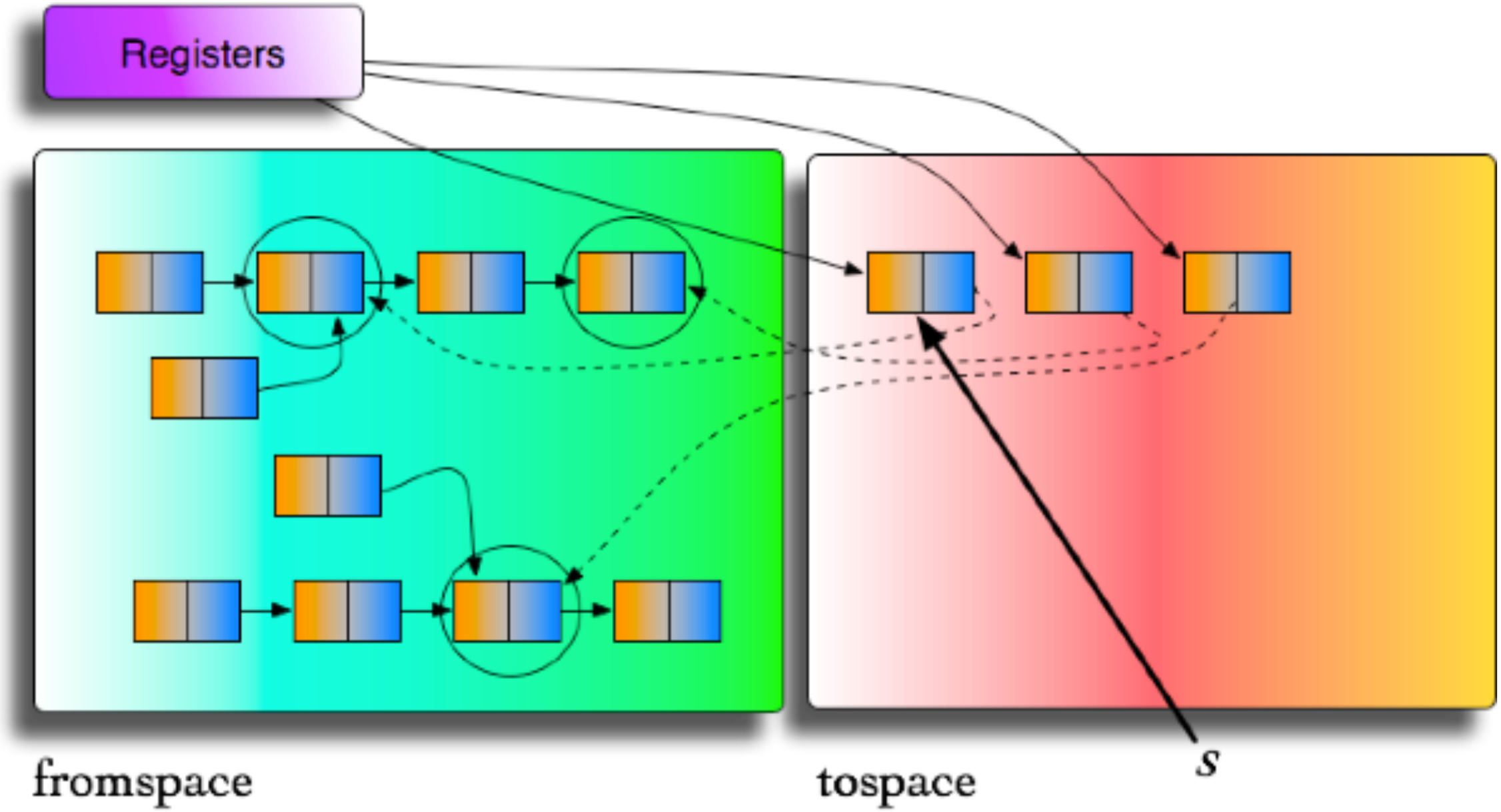
MFYCA algorithm

- A semispace copying algorithm
- Requires only one pass
- Does not require a collector stack
  - Avoided through the use of S (Scan) and B (Bottom) pointers
- Program sees addresses in to space

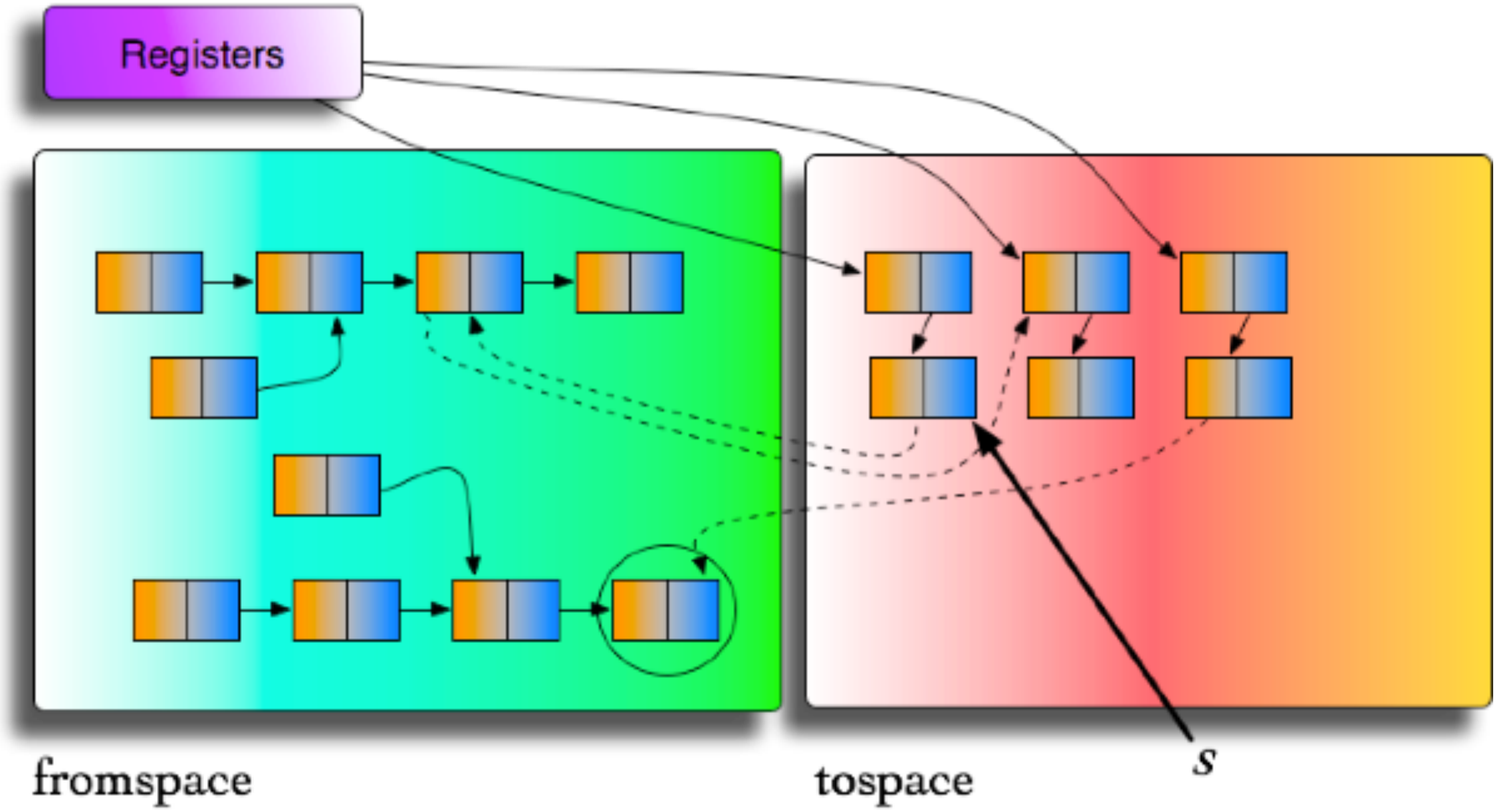
MFYCA: Initial (Post Flip)



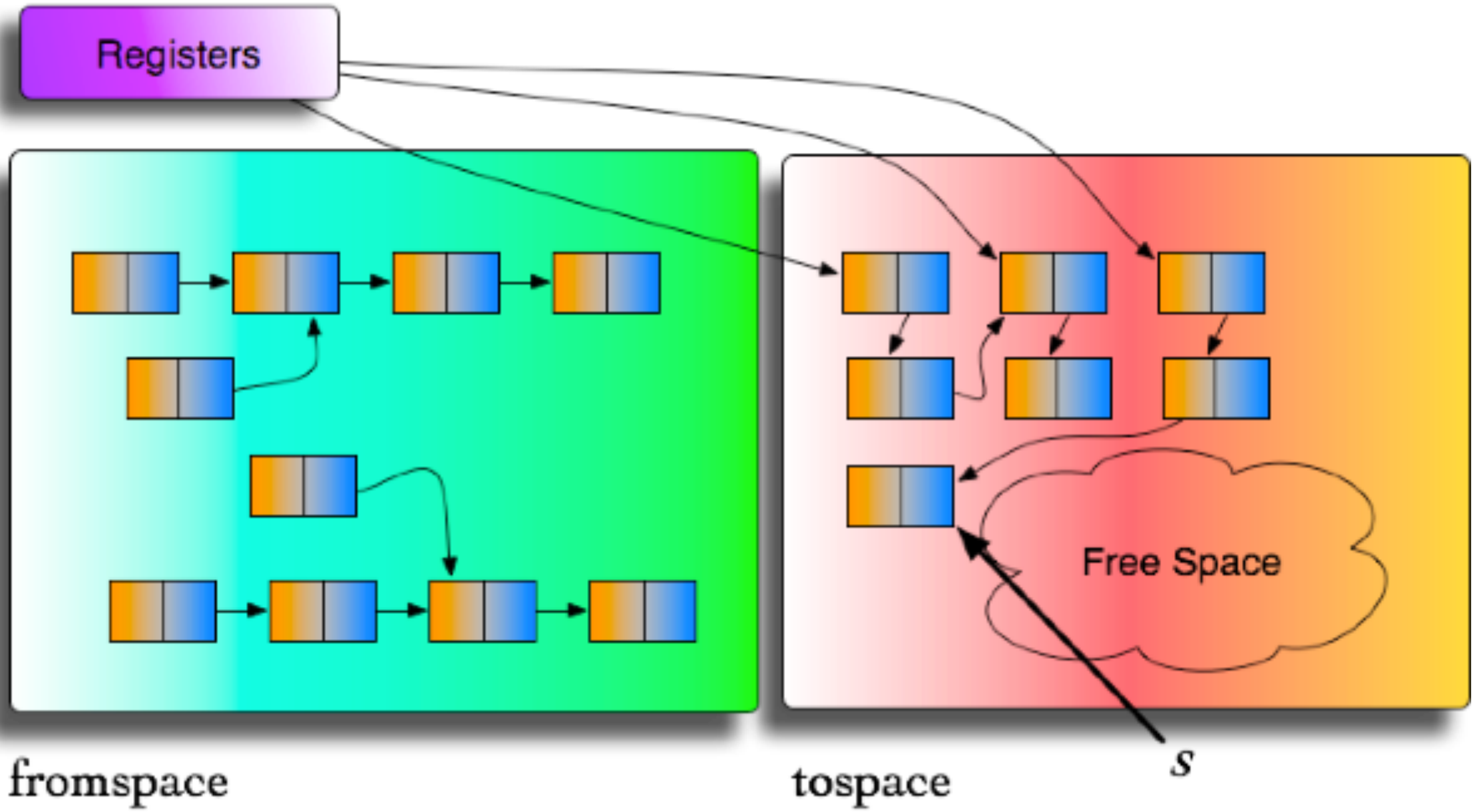
MFYCA: Copy Registers



MFYCA: Copying



MFYCA: Done



Baker's Algorithm

- When tospace fills up, do a flip and copy only roots
- At each CONS, perform k iterations of the GC loop from MFYCA
- Both semispaces now contain accessible cells
- Pretend GC completed at time of last flip
- Modify CAR, CDR: Follow forwarding addresses, move cells found in from space and update pointers
- New cells placed at top of To Space

Baker's Algorithm

- CAR, CDR can move cells before the collector has traced it
  - Does that matter?
- How about REPLACA and RPLACD?
  - REPLACA(p,q) - Suppose p is already traced and q is not
  - REPLACA(p,q) - Suppose q is already traced and q is not
  - REPLACA(p,q) - Suppose both traced or both not traced

Space requirements

- N - Number of accessible nodes
- k - Cells traced per CONS
- Maximum storage required <= N (2 + 2/k)

- Space can be reduced using CDR-coding

Space requirements

- Tradeoff between space and CONS speed by varying k
- For k > 4 space saving become insignificant
  - Doubling k = 8 gives 10% savings bit doubles cons time
- Can even make k < 1
  - With k = 1/3 need 4N storage but cons is much faster
- How about changing k dynamically?

Can we bound all operation?

- How do we handle user stacks?
  - Can grow to a unbounded size (in theory)
- Can we have a bound on ARRAY-CONS and array accessing function?
  - Doubling k = 8 gives 10% savings bit doubles cons time
- Hash Tables?

Limitations

- Virtual memory machines not supported
  - Cannot guarantee constant time
- Arbitrary size arrays not supported
- Multiple processes?

Discussion

- Paper focus a great deal on space requirement
- Size of working set?
- The "graph" of objects is traversed in breadth-first order
  - True for both MFYCA and Baker
  - What does this mean for locality?
- Read barrier overhead?

Conclusions

- Modification to MFYCA
- Real-time: all operations constant time
- Space efficiency and flexibility: can choose k for space-time tradeoff
- Proof: Correct and doesn't run out of space when it shouldn't

Conclusions

- Modification to MFYCA
- Real-time: all operations constant time
- Space efficiency and flexibility: can choose k for space-time tradeoff
- Proof: Correct and doesn't run out of space when it shouldn't