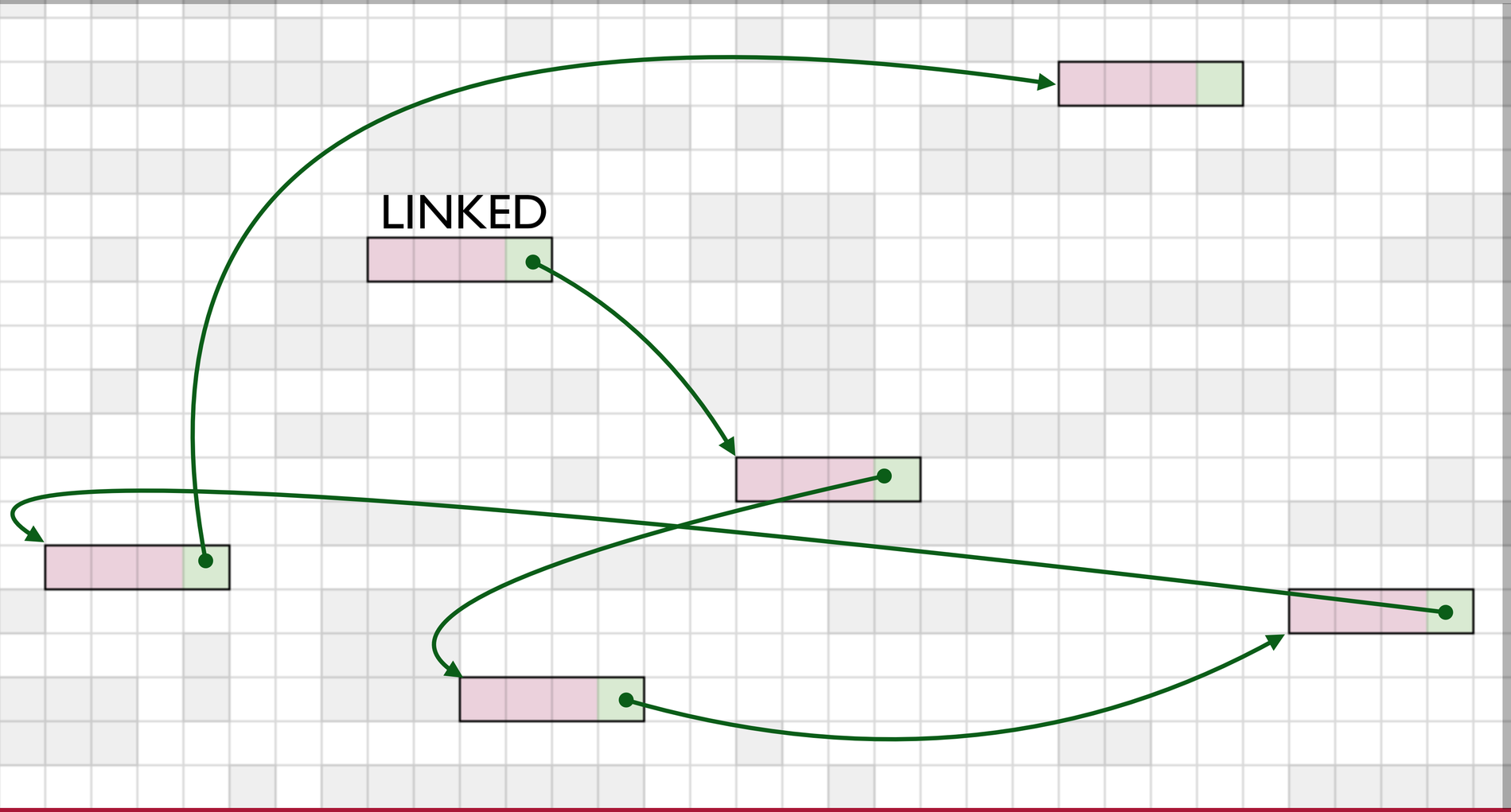
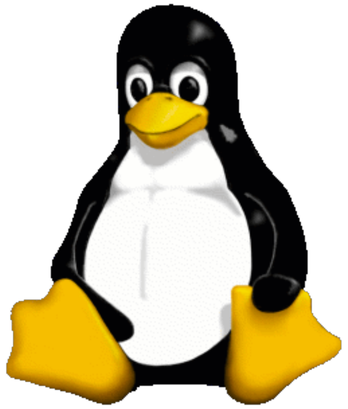
MEMORY

CONTIGUOUS



Your application

Operating System



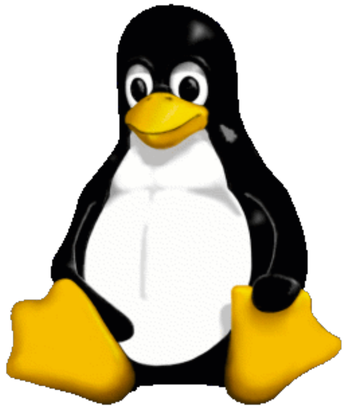
Memory

1	2	3	4
5	6	7	8
9	10	11	12

Your application

Operating System

Memory



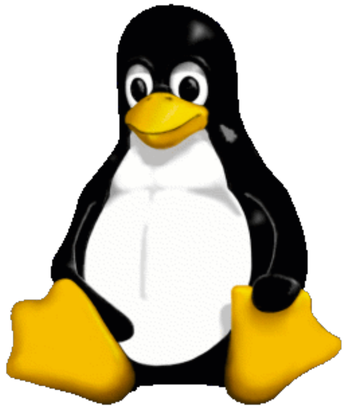
1	2	3	4
5	6	7	8
9	10	11	12

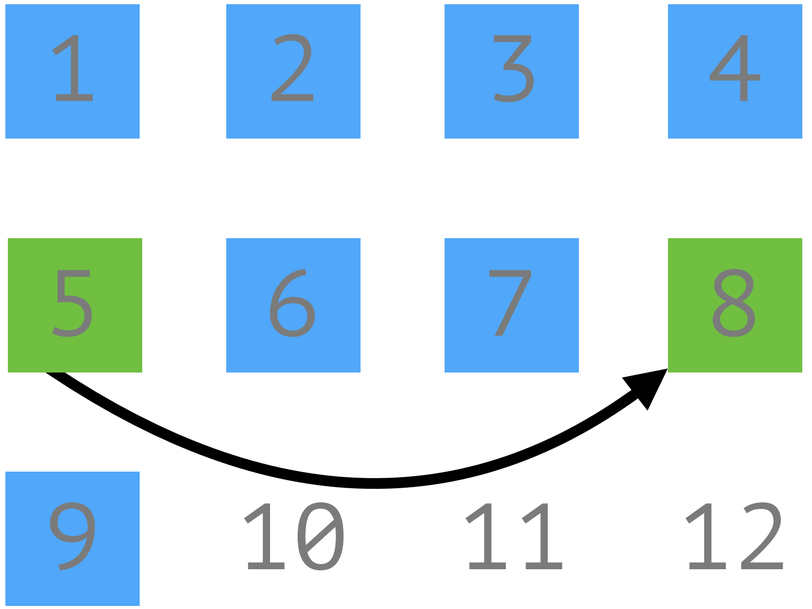
I just need one block of memory for now…doesn't matter where!

Your application

Operating System

Memory





Okay, I'm adding a new item. I'll use the previous cell you gave me to point to the next

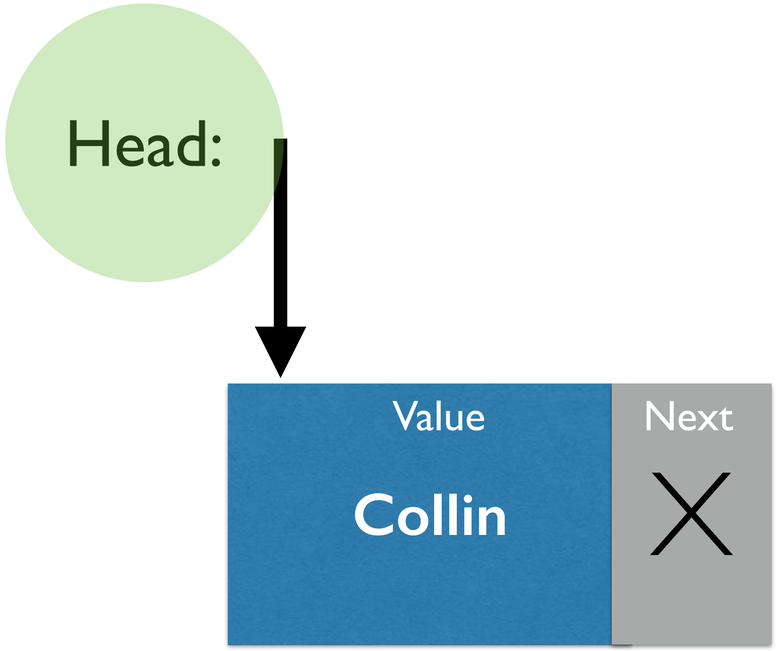
LINKED LISTS

The Linked List DS

- Data structure used for *list*, *stack*, *queue*, *deque* ADTs etc.
- Uses ***nodes*** which encapsulate a **value** and pointer(s)
- Main entity holds reference(s) to just a head and/or tail node
  - the "handle(s)"
- Each node then *points* to the **next** and/or **previous** node
  - "singly-linked" (unidirectional) vs. "doubly-linked" (bidirectional)

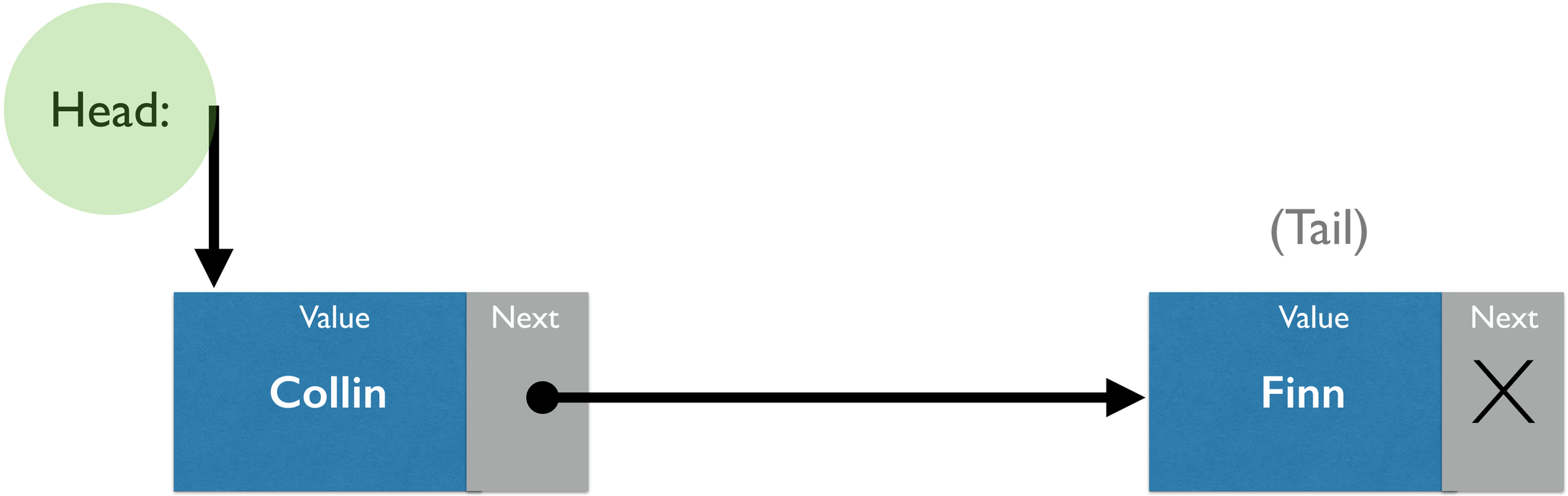
Linked List

Linked List

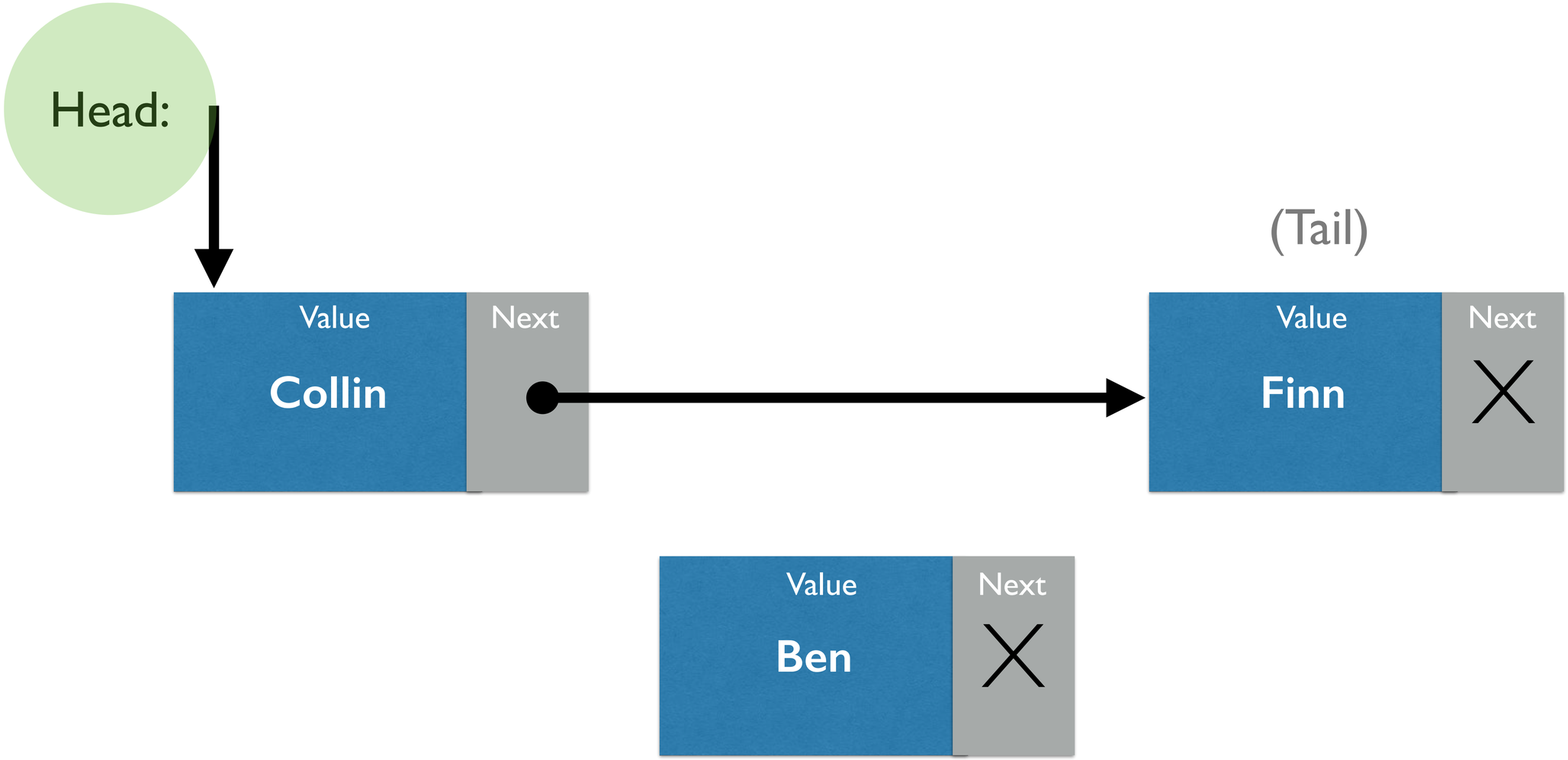


Linked List

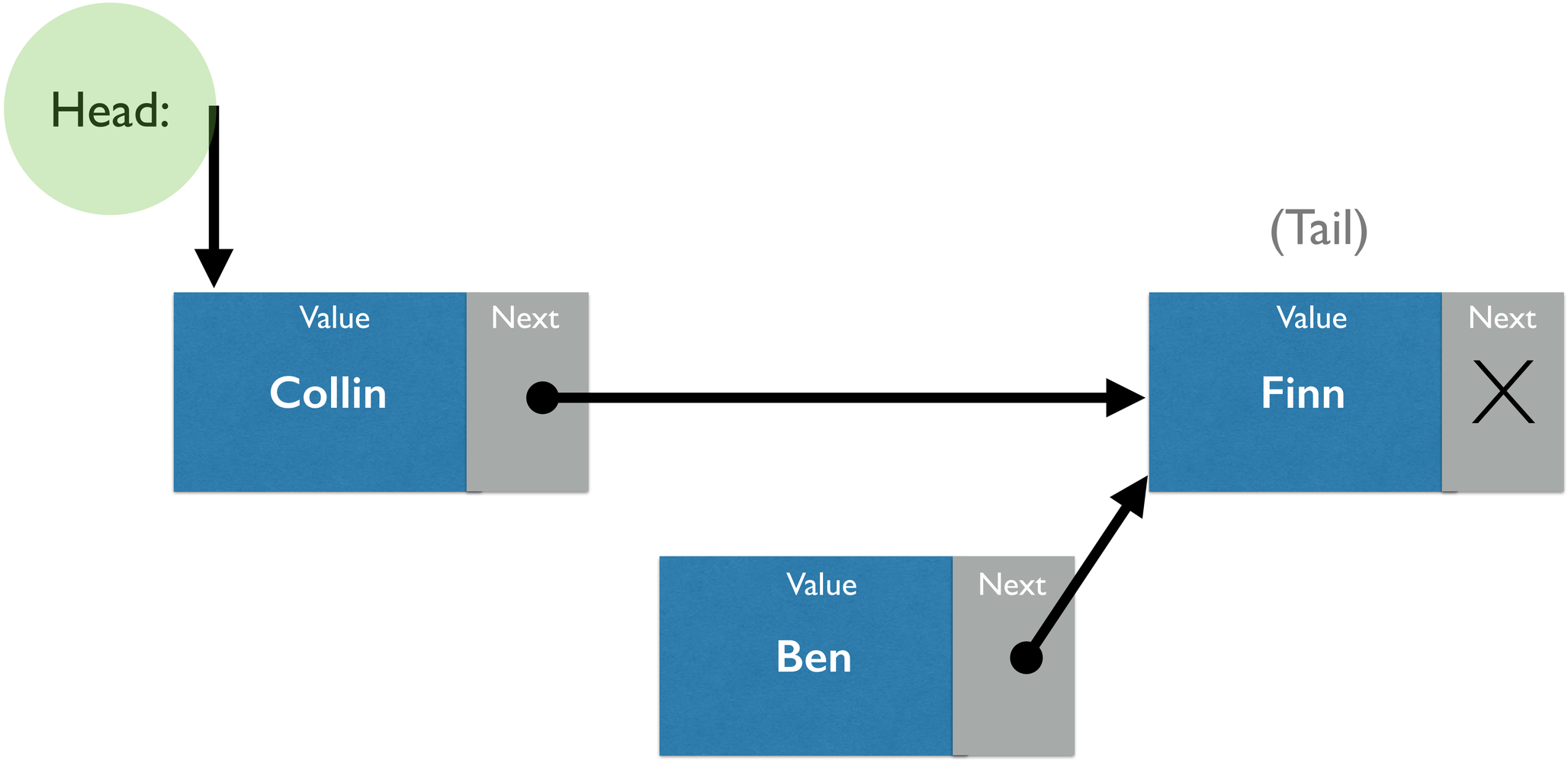
Linked List



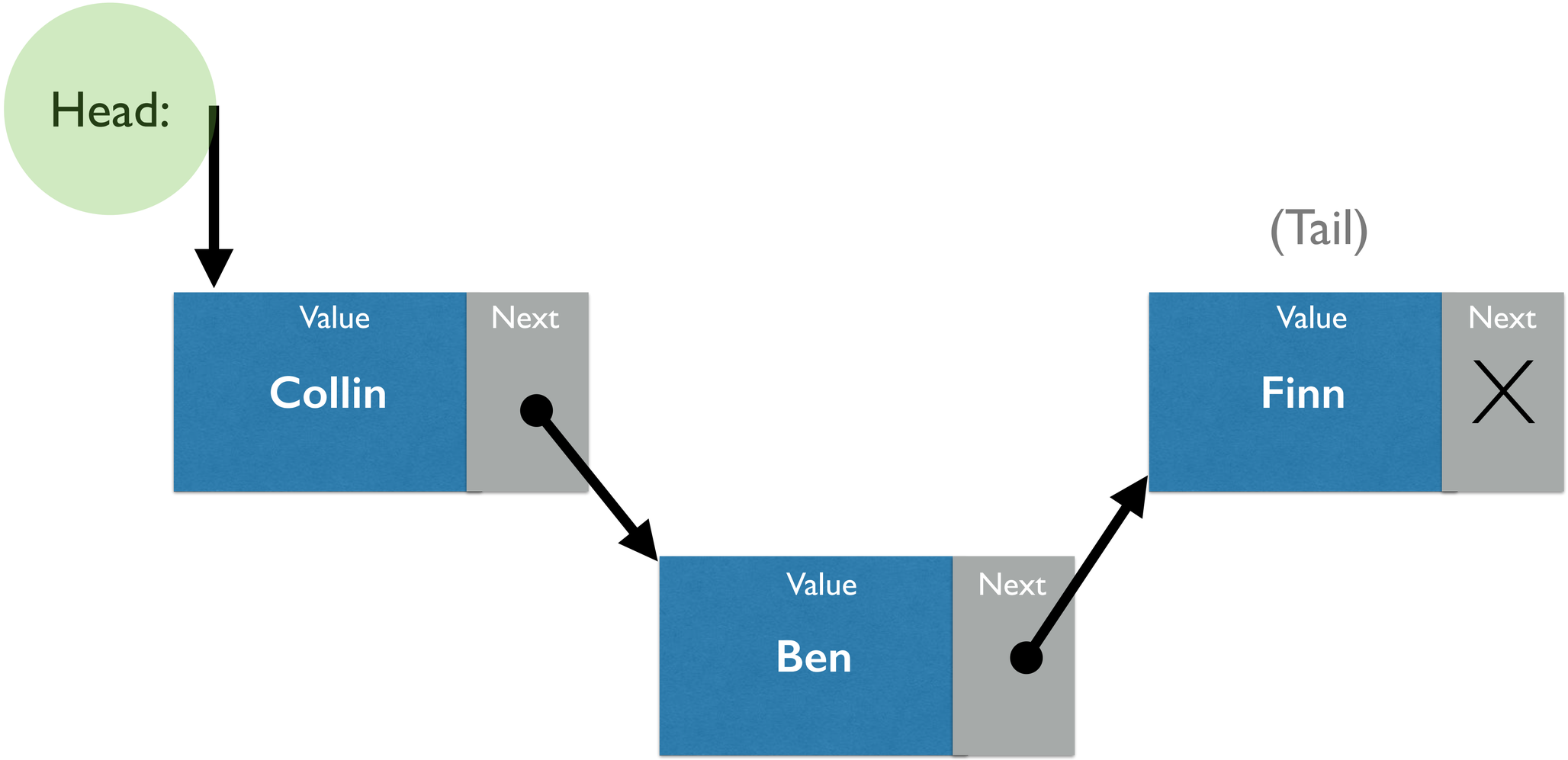
Linked List



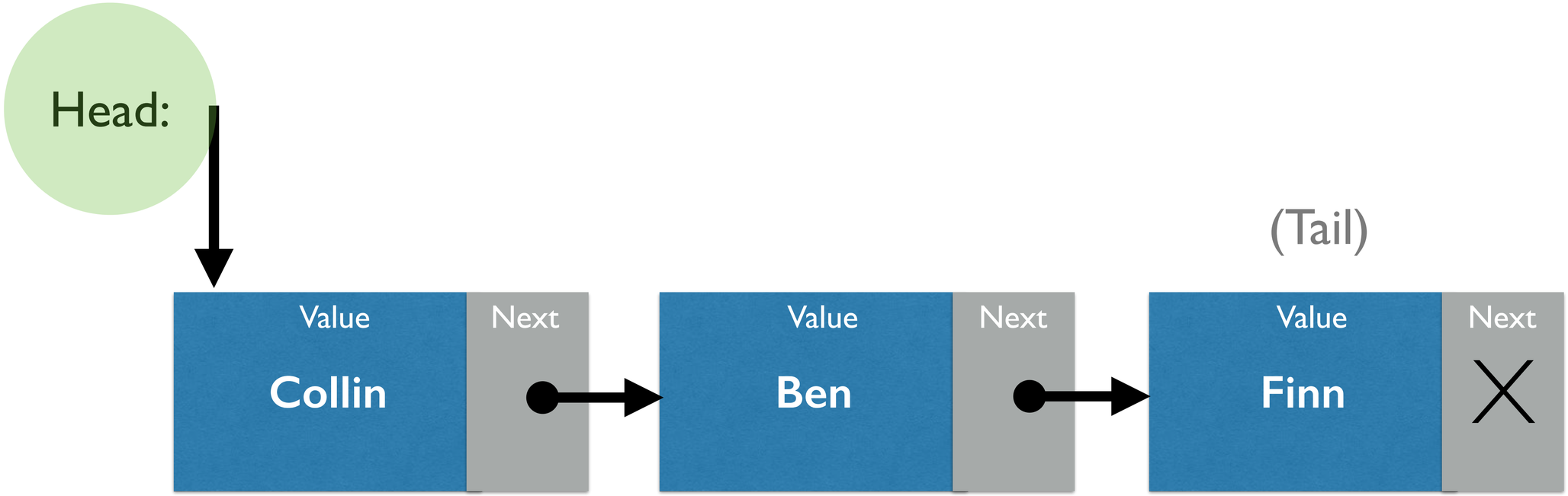
Linked List



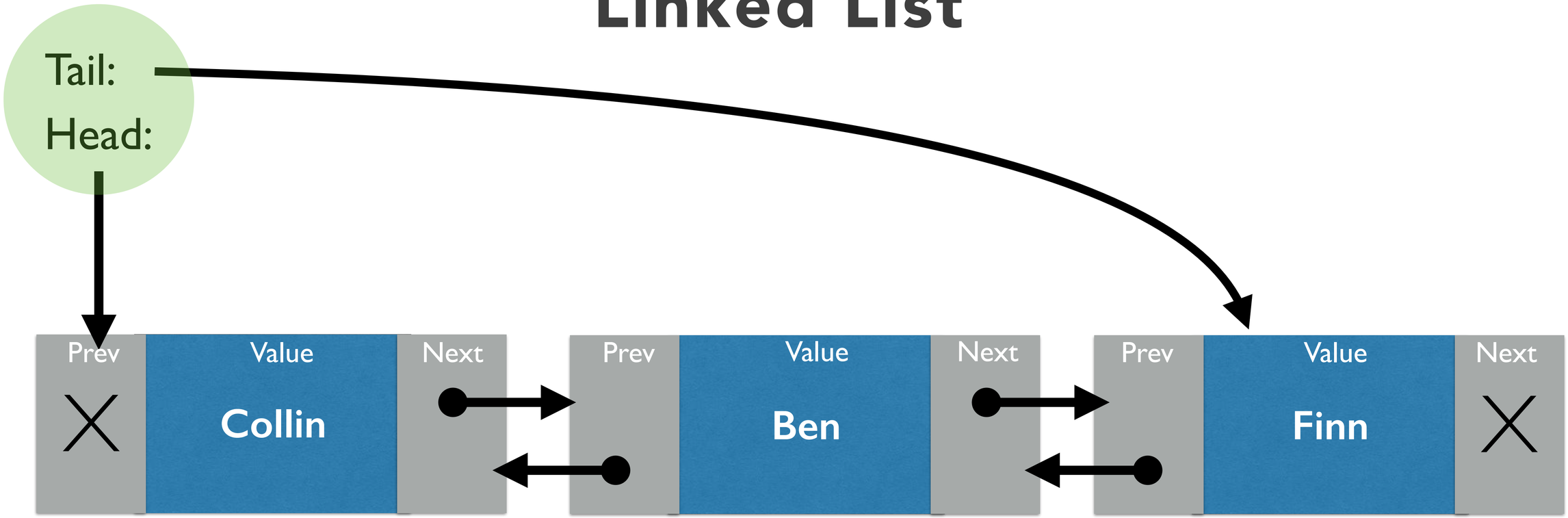
Linked List



Linked List



Linked List


Tail:
Head:
Prev
Value
Collin
Next
Prev
Value
Ben
Next
Prev
Value
Finn
Next

(some) pros/cons: Linked Lists vs Arrays

Operation	Linked List	Typed Array
Reach element in middle	Must crawl though nodes	Constant time
Insert in middle or start	Constant time (if we have ref).	Must move all following elements
Add element to end	With handle, constant time	Constant time
Space per element	Container + element + pointer(s)	Just element!
Total space	Grows as needed	Pre-reserved & limited*
Physical locality	Not likely	Best possible

WORKSHOP