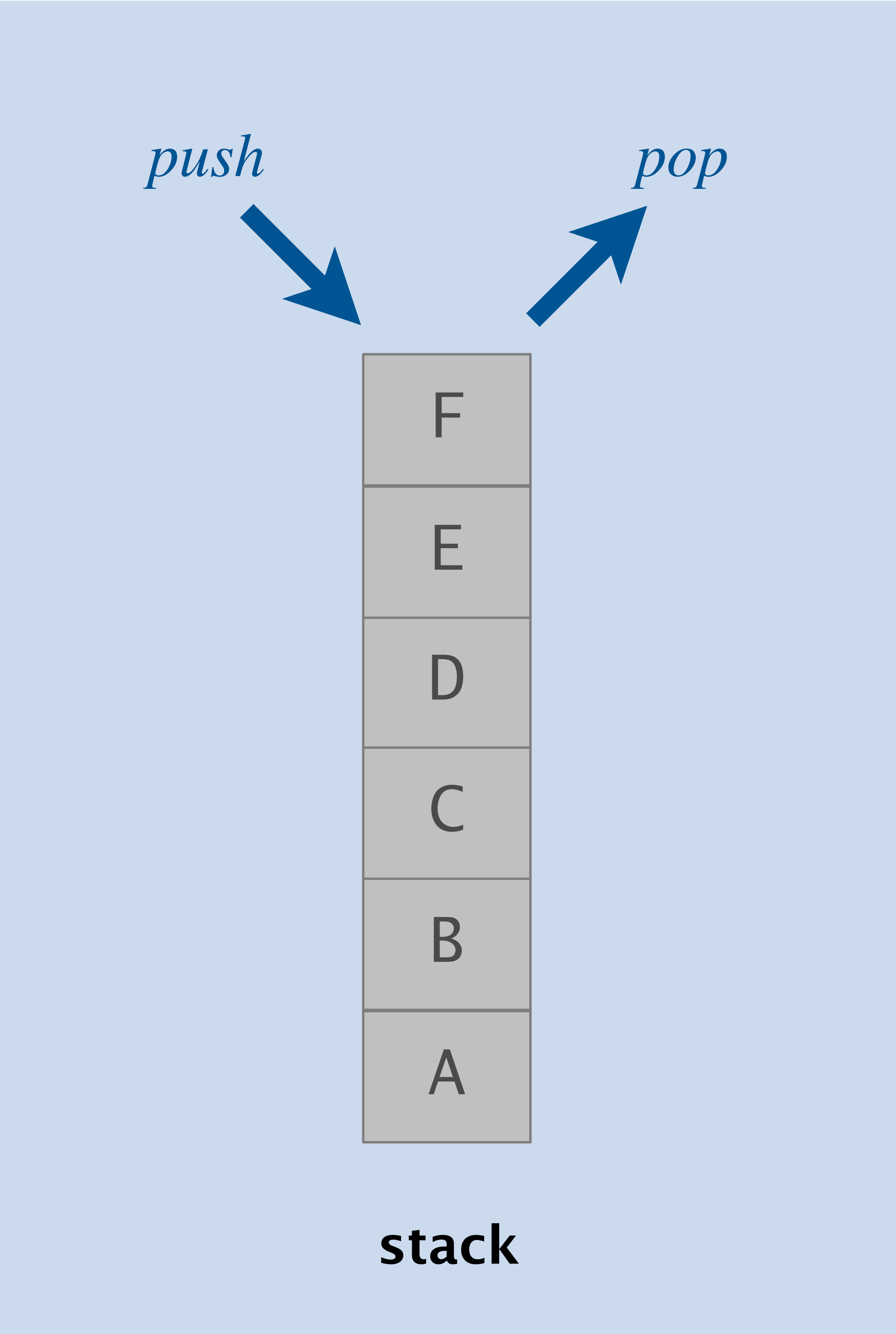
Stacks and queues

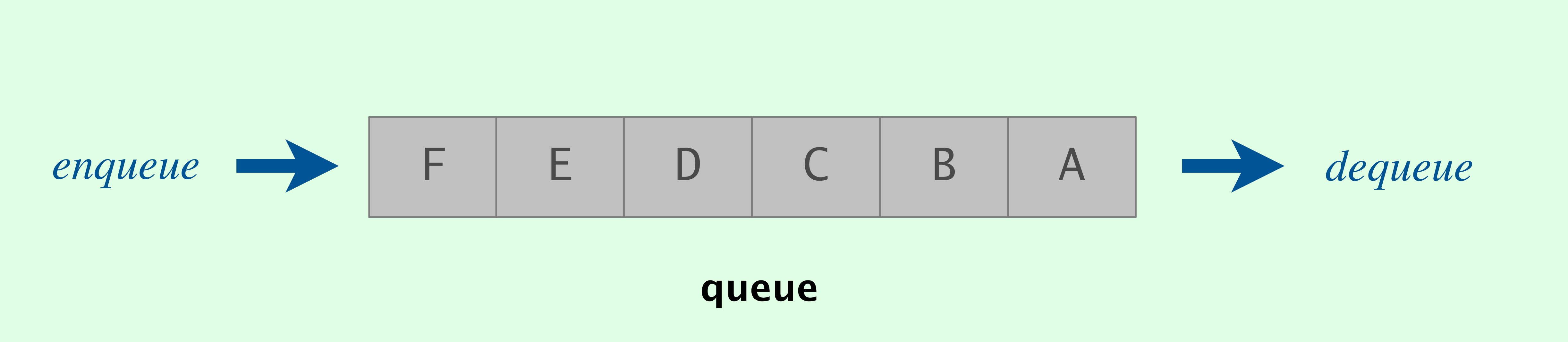
Fundamental data types.

- Value: collection of objects.
- Operations: add, remove, iterate, size, test if empty.

Stack. Remove the item most recently added.

Queue. Remove the item least recently added.





Programming assignment 2

Deque. Remove either the most recently or the least recently added item.

Randomized queue. Remove a random item.

Your job.

- Step 1. Identify a data structure that meets the performance requirements.
- Step 2. Implement it from scratch.

← *think carefully about step 1 before proceeding to step 2*

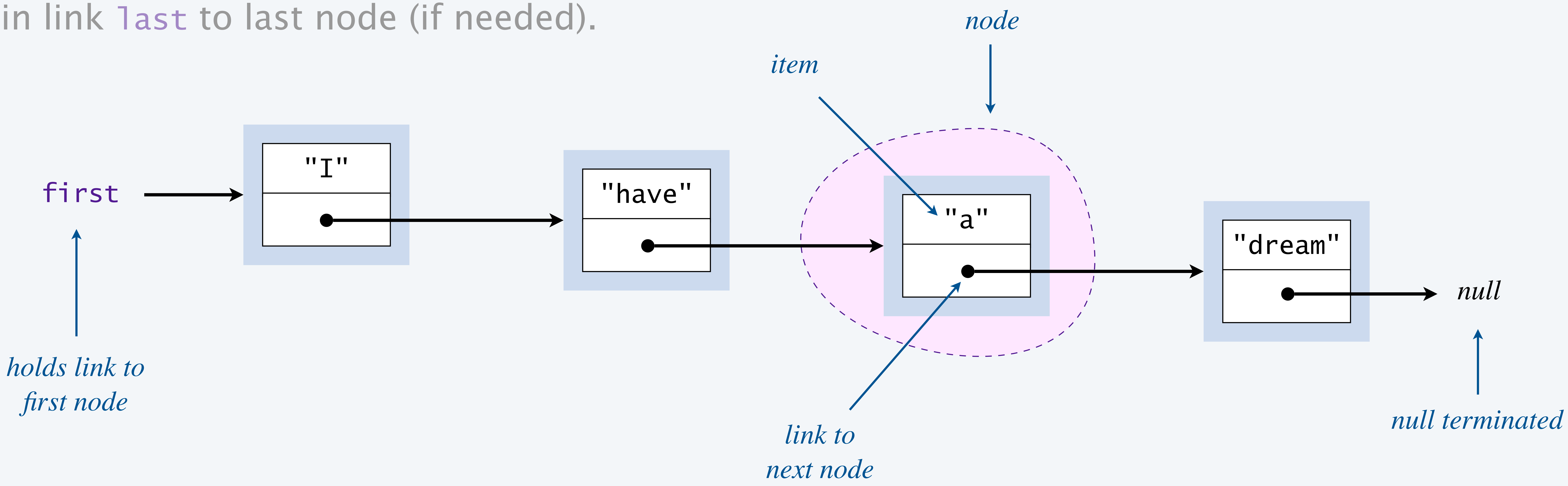
Linked lists

Last lecture. Use a resizable array to implement all operations in amortized $\Theta(1)$ time.

This lecture. Use a singly linked list to implement all operations in $\Theta(1)$ time in the worst case.

Singly linked list.

- Each node stores an item and a link/pointer to the next node in the sequence.
- Last node links to `null`.
- Maintain link `first` to first node.
- Maintain link `last` to last node (if needed).



Possible memory representation of an array

Java array. The elements in an array are stored contiguously in memory.

Consequences.

- Accessing array element i takes $\Theta(1)$ time.
- Cannot change the length of an array.
- When passing an array to a function, the function can change array elements.

reference variable a[]
(holds memory address of elements)
memory
address
286
a[]
304
array elements
4
304
"I"
"have"
"a"
"dream"
length of array

memory representation
(using poetic license)

Possible memory representation of a singly linked list

Java linked list. The nodes in a singly linked list are stored non-contiguously in memory.

Consequences.

- Accessing i^{th} node in a singly linked list takes $\Theta(i)$ time.
- Easy to change the length of a singly linked list.

memory address
of Node object
memory address
of next node
286
first
304
318
330
"have"
330
304
"I"
286
"dream"
0
"a"
318
object reference
(holds memory address of Node object)
Node object
null reference

memory representation
(using poetic license)