CS24 Week 4 Lecture 2

Kyle Dewey

Overview

- Linked Lists
- Stacks
- Queues

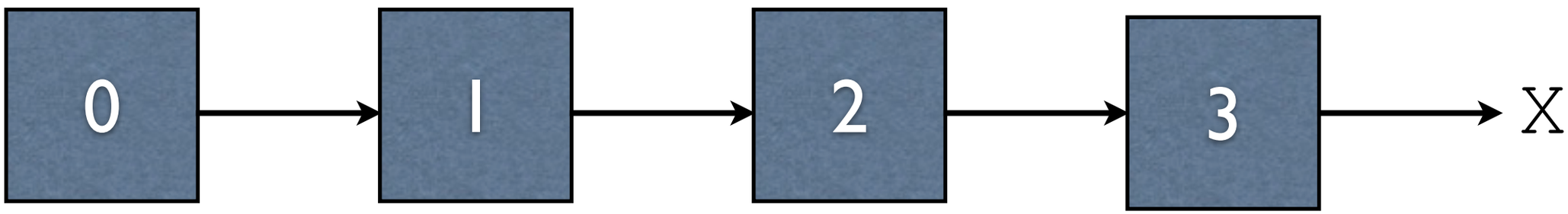
Linked Lists

Linked Lists

- Idea: have each chunk (called a *node*) keep track of both a list element *and* another chunk

- Need to keep track of only the *head* node

List: 0, 1, 2, 3



Node Representation

- What might a node look like in C?

```
struct Node {
   int item;
   struct Node* next;
};
```

Node Representation

- What might a node look like in C++?

```
class Node {
  public:
    Node(int i, Node* n);
    int getItem() const;
    void setItem(int i);
    Node* getNext() const;
    void setNext(Node* n);
  private:
    int item;
    Node* node;
};
```

C++ Implementation of Linked Lists

Stacks

Stack

- Like a linked list, these hold items
- So named because items are "stacked" on top of each other - can only access from one end
  - Last in, first out (LIFO) order

http://eli.thegreenplace.net/2011/02/04/where-the-top-of-the-stack-is-on-x86/

The Stack

- We have previously discussed **the** stack
  - Local allocation
- How does local allocation work with **the** stack?
- Why is this called **the** stack?

Stack ADT

- Stacks can only be accessed from one end
- What sort of operations are applicable?

Stack ADT

- Stacks can only be accessed from one end
- What sort of operations are applicable?
  - Create an empty stack
  - *push* an element on the stack
  - *pop* an element off the stack
  - Look at the top element without popping, often called *top*

Stack ADT Logical Level in C++

- Create an empty stack
- *push* an element on the stack
- *pop* an element off the stack
- Look at the top element without popping, often called *top* or *peek*

--Constructors? Methods? Signatures?--

Logical Level

```
class Stack {
  public:
    Stack(); // constructor

    void push(int item);
    int pop();
    int top() const;
};
```

Implementation Level

- How might we implement the stack?

```
class Stack {
  public:
    Stack(); // constructor

    void push(int item);
    int pop();
    int top() const;
};
```

Implementation Level

- Two popular choices: arrays and linked lists
  - Arrays: grow from left to right
  - Linked lists: add and remove from the head

Stacks with Linked Lists

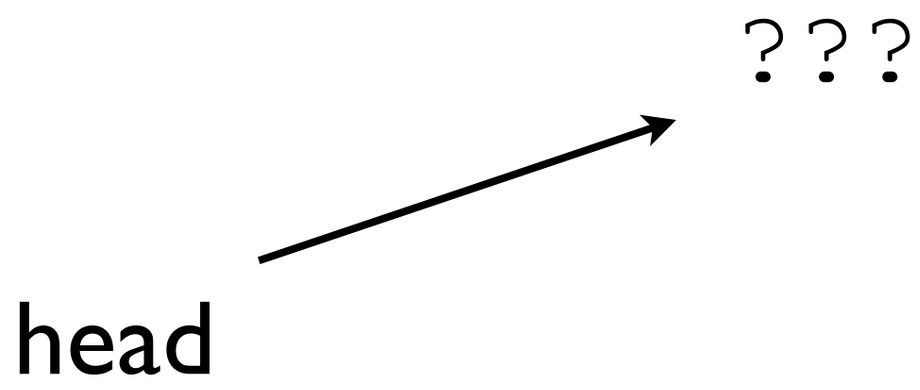
- Linked lists tend to work better for stacks. Why?
- Hint: what is problematic with an array representation?

Stacks with Linked Lists

- Linked lists tend to work better for stacks. Why?
  - Easily `push` by adding an element to the front
  - Easily `pop` by removing an element from the front
  - No embedded maximum size

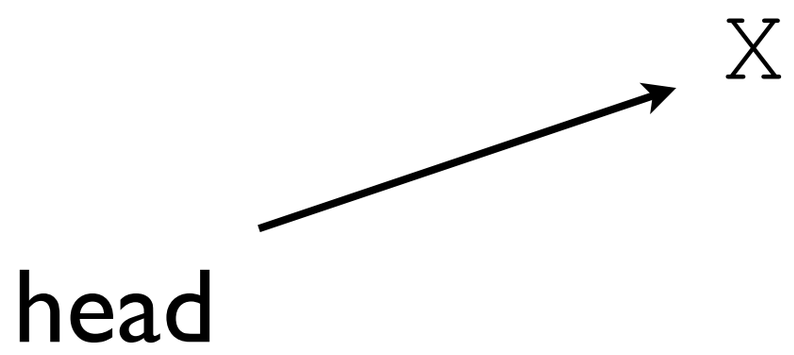
Stacks with Linked Lists

```
IntStack stack;
```



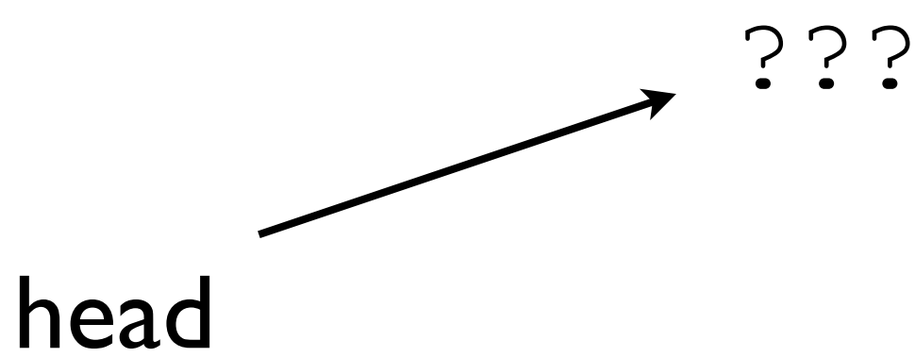
Stacks with Linked Lists

```
IntStack stack;
```



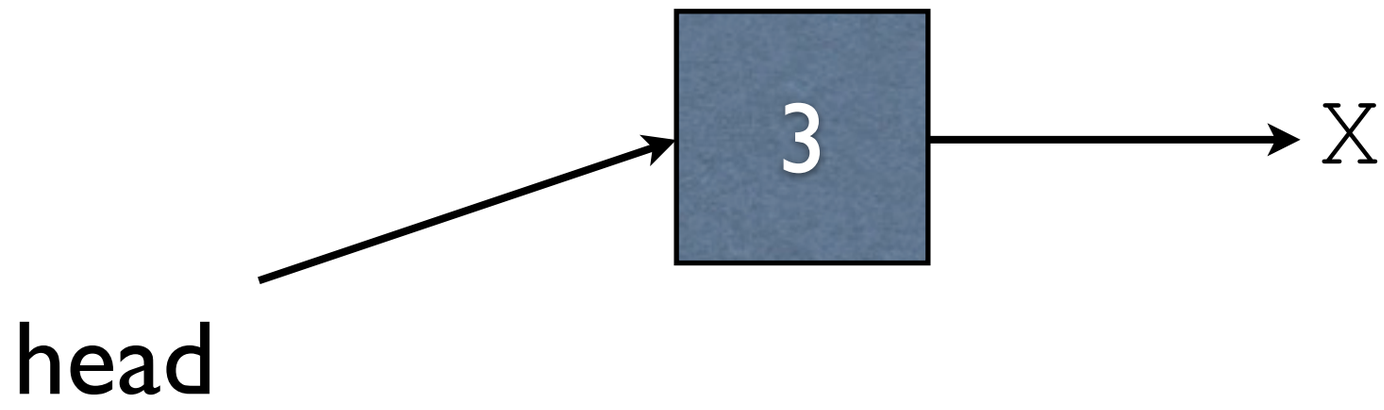
Stacks with Linked Lists

```
IntStack stack;
stack.push(3);
```



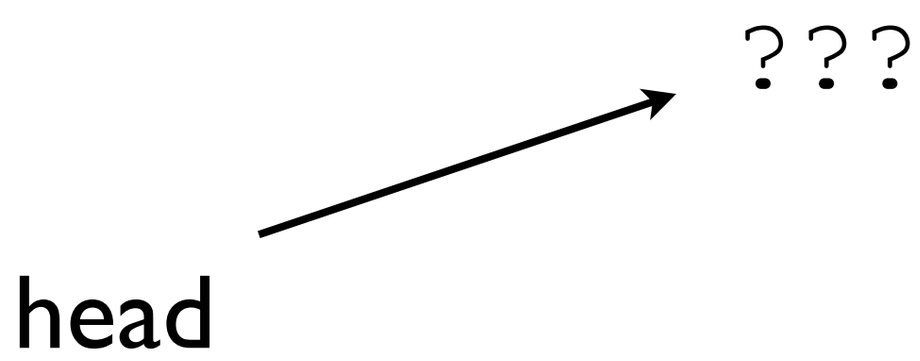
Stacks with Linked Lists

```
IntStack stack;
stack.push(3);
```



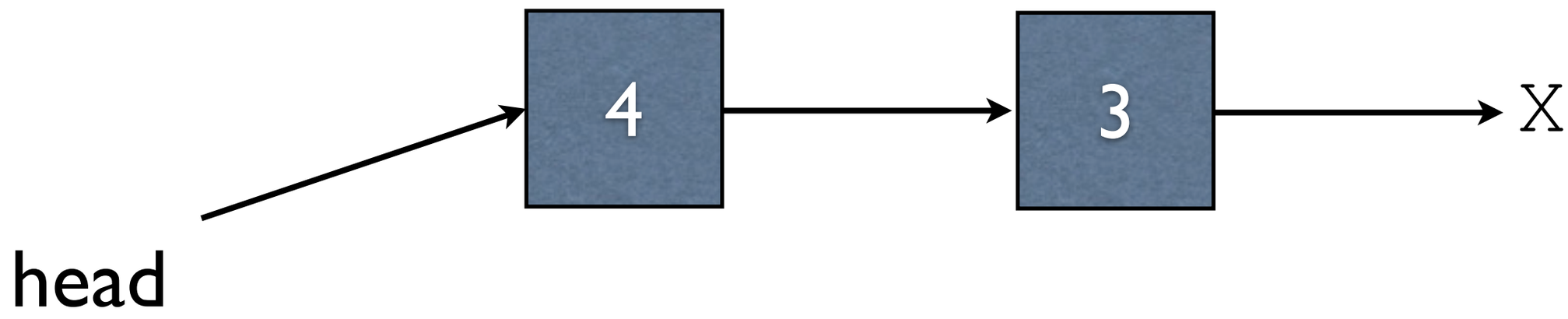
Stacks with Linked Lists

```
IntStack stack;
stack.push(3);
stack.push(4);
```



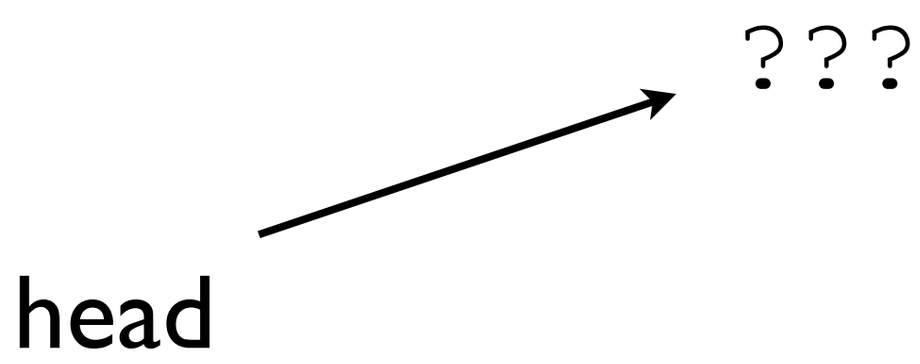
Stacks with Linked Lists

```
IntStack stack;
stack.push(3);
stack.push(4);
```



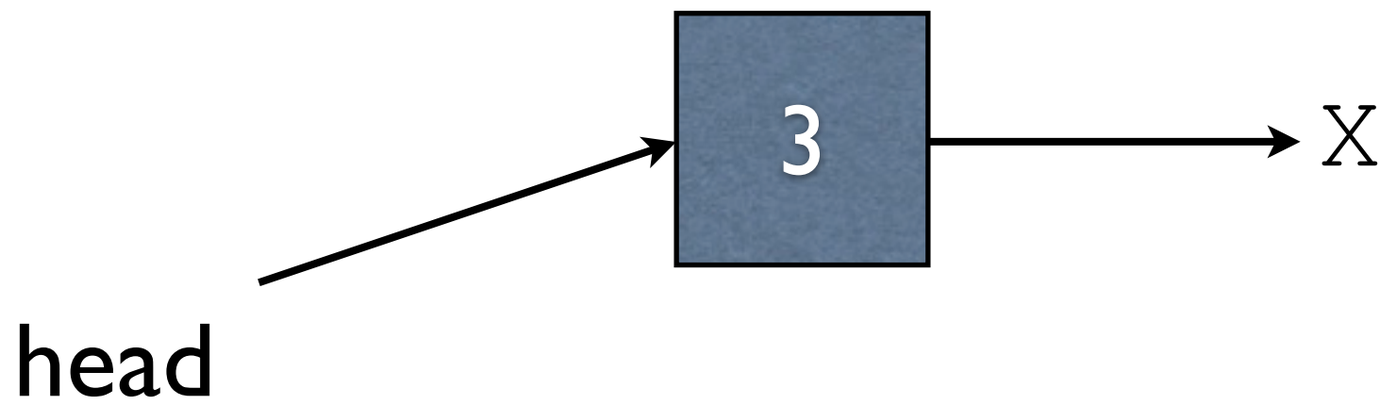
Stacks with Linked Lists

```
IntStack stack;
stack.push(3);
stack.push(4);
stack.pop();
```



Stacks with Linked Lists

```
IntStack stack;
stack.push(3);
stack.push(4);
stack.pop();
```



Queues

Motivation

- A grocery store has one cashier
- Ten people want to checkout
- People form a line based on when they arrived at the cashier
  - First in, first out (FIFO) order

Queue ADT

- Queues have a concept of two ends - the front and back
- What sort of operations might be on a queue ADT?

Queue ADT

- Queues have a concept of two ends - the front and back
- What sort of operations might be on a queue ADT?
  - Adding to the queue, often called `enqueue`
  - Removing from the queue, often called `dequeue`

Queue ADT Logical Level in C++

- Create an empty queue
- `enqueue` **an item**
- `dequeue` **an item**

--Constructors? Methods? Signatures?--

Logical Level

```
class Queue {
  public:
    Queue(); // constructor

    void enqueue(int item);
    int dequeue();
};
```

Implementing Queues

- How might we implement a queue?

Implementing Queues

- How might we implement a queue?
  - Arrays: keep track of the front and back of the queue (hard!)
  - Linked lists: add to the front or back of the list

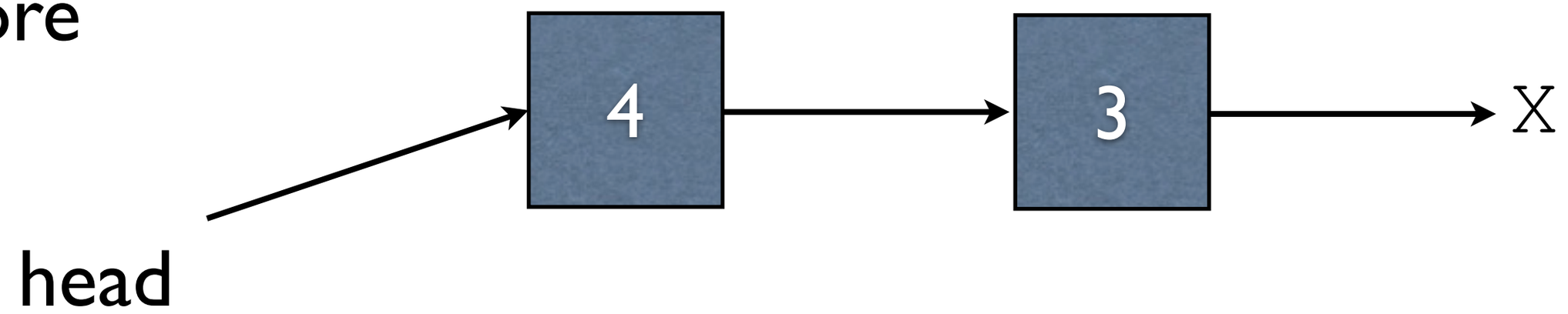
Linked List Implementation

- How is the queue represented?
- What happens on `enqueue`?
- What happens on `dequeue`?

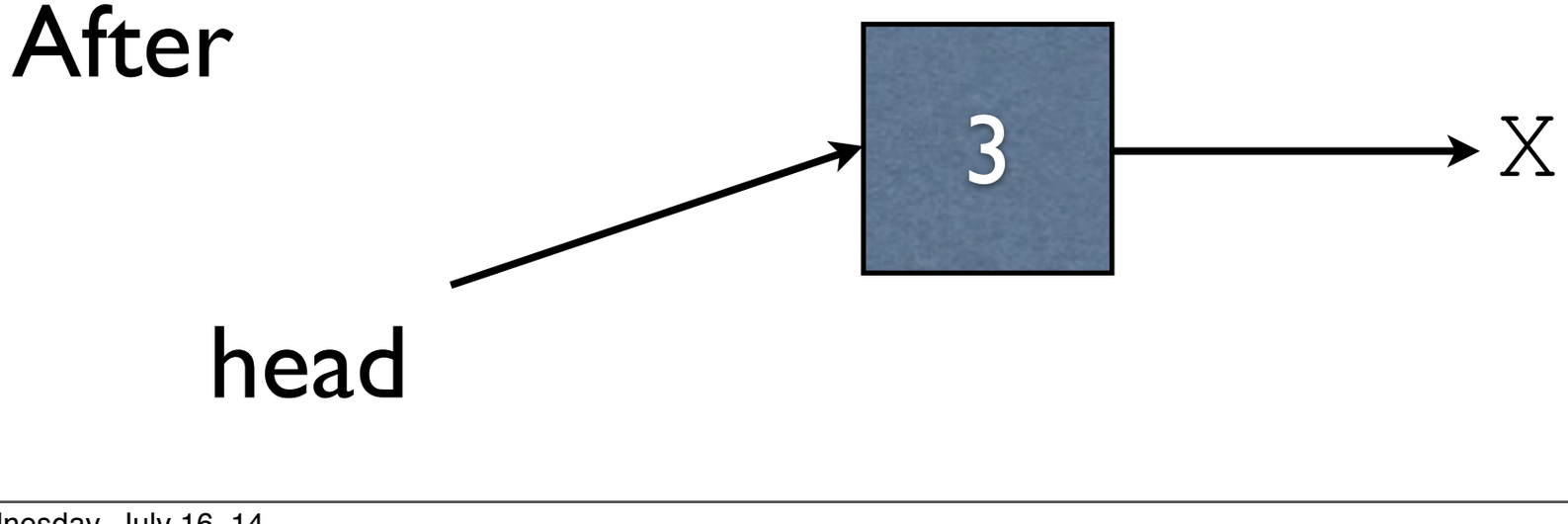
dequeue

- `dequeue` removes from the front of the line, AKA the front of the list

Before



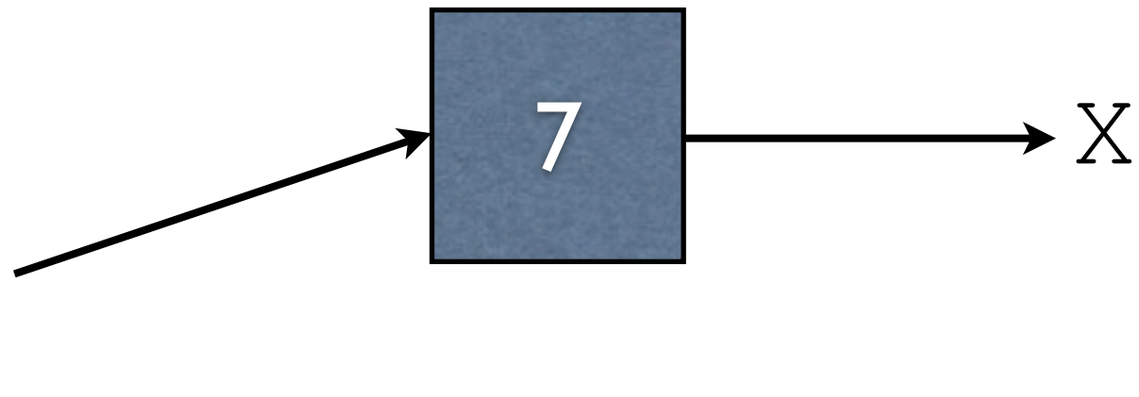
After



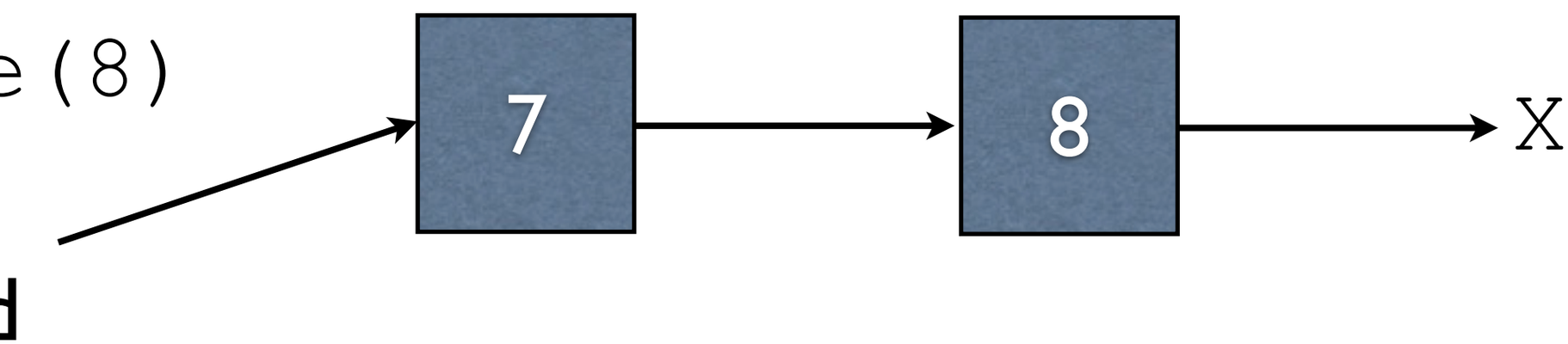
`enqueue`

- `enqueue` adds to the back of the line, AKA the back of the list

Before



After `enqueue(8)`



`enqueue` **vs.** `dequeue`

- `enqueue` **is less efficient than** `dequeue`. **Why?**

`enqueue` **vs.** `dequeue`

- `enqueue` **is less efficient than** `dequeue`. **Why?**
  - **Via** `head`, `dequeue` **has direct access to the front of the line**
  - **In contrast,** `enqueue` **must walk the entire line to get to the end**

Addressing Efficiency Problem

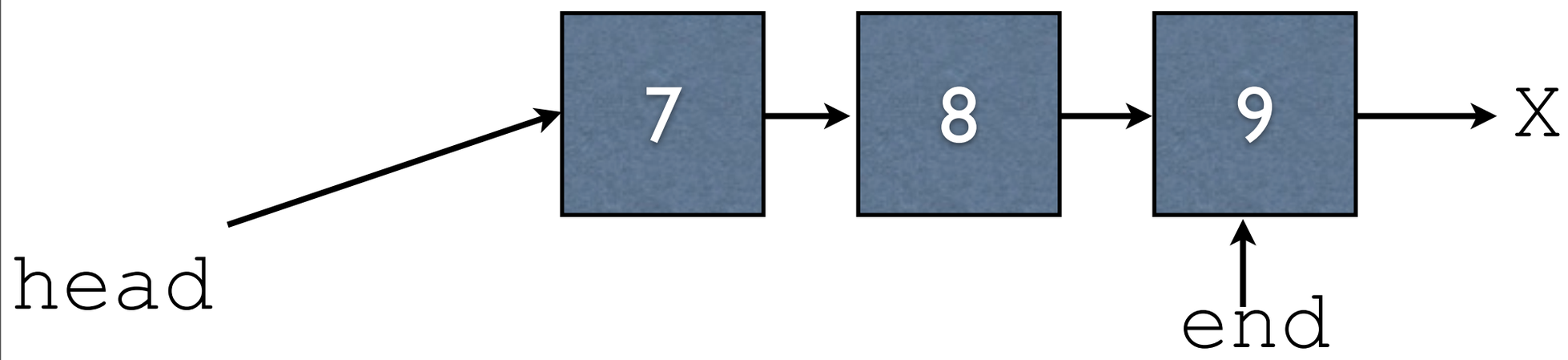
- Without diverging too much from a standard linked list, how might we address this efficiency problem?
  - Hint: `head` essentially teleports us to the front of the list

Addressing Efficiency Problem

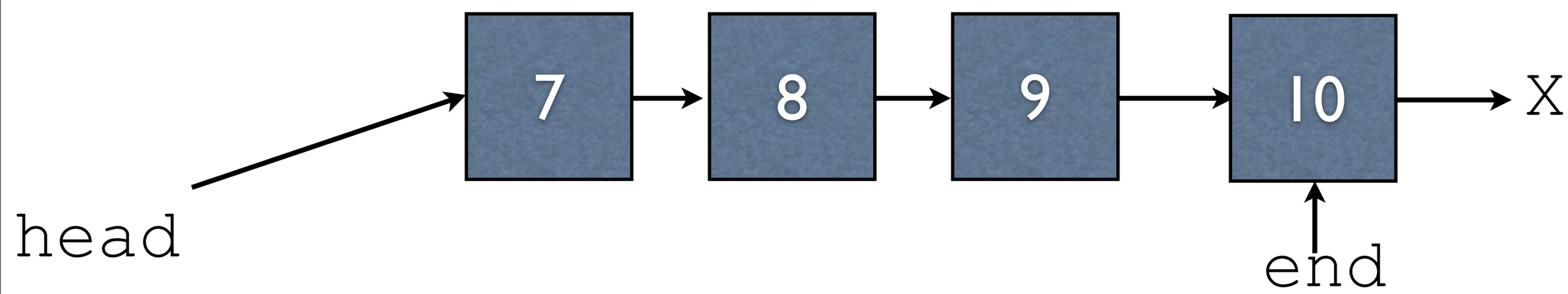
- Without diverging too much from a standard linked list, how might we address this efficiency problem?
  - Hint: `head` essentially teleports us to the front of the list
  - We could add a special `end` pointer to the end of the list

After Change

Before `enqueue(10)`



After `enqueue(10)` **- add directly to** `end`



Questions

- For a queue of empty length, what is `head` and `end`?
- For a queue of length one, what is `head` and `end`?

Questions

- For a queue of empty length, what is `head` and `end`?
  - `NULL`
- For a queue of length one, what is `head` and `end`?
  - The same element

Extra Bookkeeping

- Need to ensure that `end` always points to the last element