# CS 106B
# Lecture 16: Linked Lists

Wednesday, July 26, 2017

---

Programming Abstractions
Summer 2017
Stanford University
Computer Science Department

Lecturer: Chris Gregg

reading:
Programming Abstractions in C++, Chapter 11

(Linked Wrists ↑)

# Today's Topics

- Logistics
  - Assignment 5 YEAH Hours tonight

- Linked Lists
  - Could you architect a Queue?
  - Nodes
  - Linked Lists
  - The Towers of Gondor
  - Do nodes have names?
  - Big O?
  - Stack and Queue made from a Linked List

# Assignment 5: Linked Lists and Heaps

For this assignment, you will be implementing a data structure called a "priority queue" which allows you to store keys and values based on the "priority" of the key. You will be modeling a hospital emergency room: patients with a higher priority are attended to first, even if they arrive after patients with lower priority:

For example, if the following patients arrive at the hospital in this order:

- "Dolores" with priority 5
- "Bernard" with priority 4
- "Arnold" with priority 8
- "William" with priority 5
- "Teddy" with priority 5
- "Ford" with priority 2

Then if you were to dequeue the patients to process them, they would come out in this order: Ford, Bernard, Dolores, William, Teddy, Arnold.

# More on part C for HW 5 later!

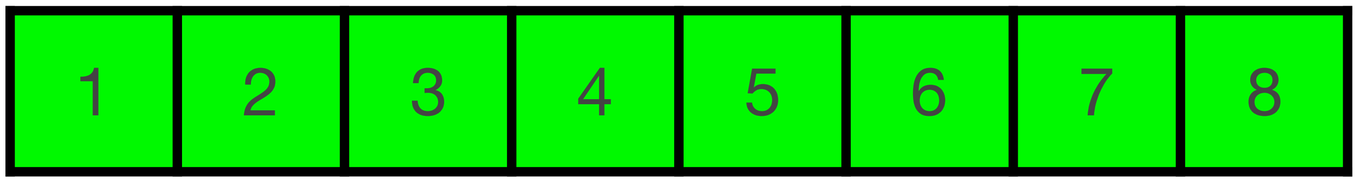
# Your job: Architect a Queue

# Easiest Solution...

```
class QueueInt {                    // in QueueInt.h
public:
    QueueInt ();                    // constructor

    void enqueue(int value); // append a value
    int dequeue();                  // return the first-in value

private:
    Vector<int> data;       // member variables
};
```

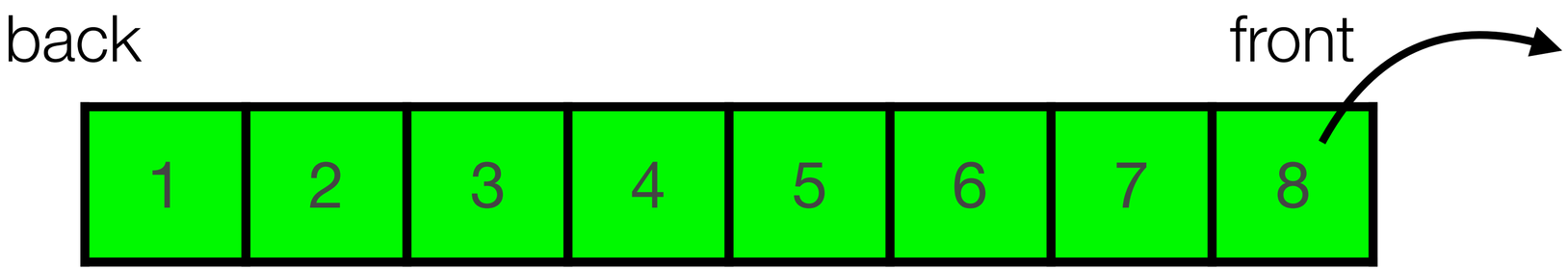
# You're next!

back front



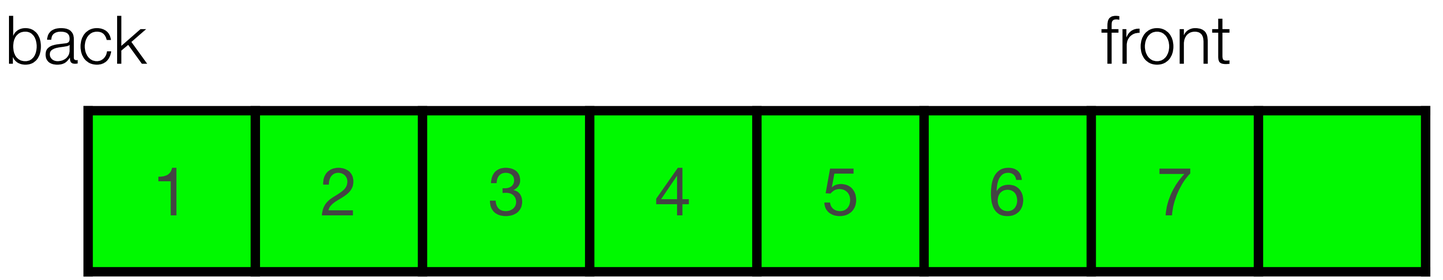

dequeue()

# You're next!



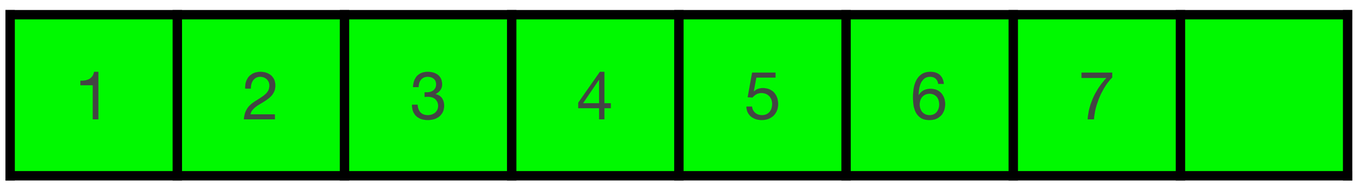

dequeue()

# Excuse Me, Coming Through



# Excuse Me, Coming Through

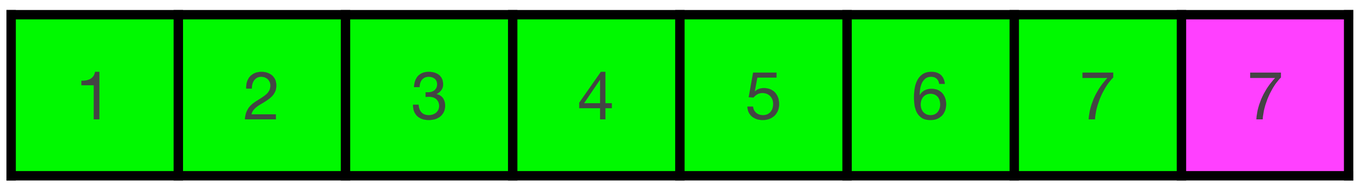
back front




enqueue(42)

# Excuse Me, Coming Through

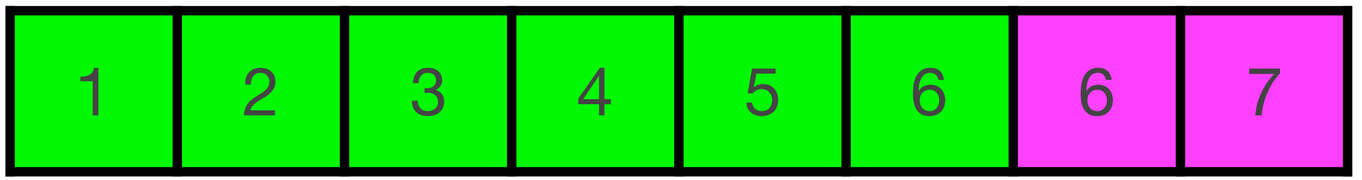
back front




enqueue(42)

# Excuse Me, Coming Through

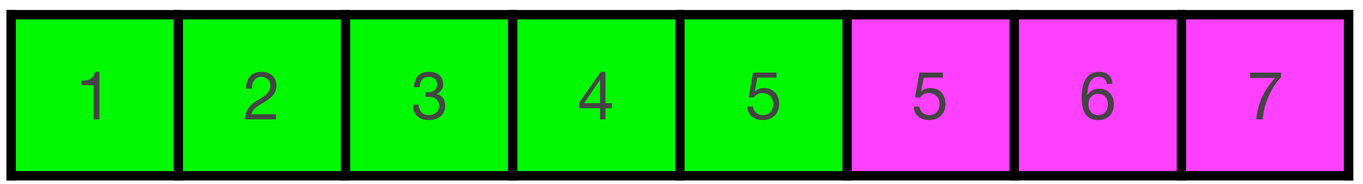
back front




enqueue(42)

# Excuse Me, Coming Through

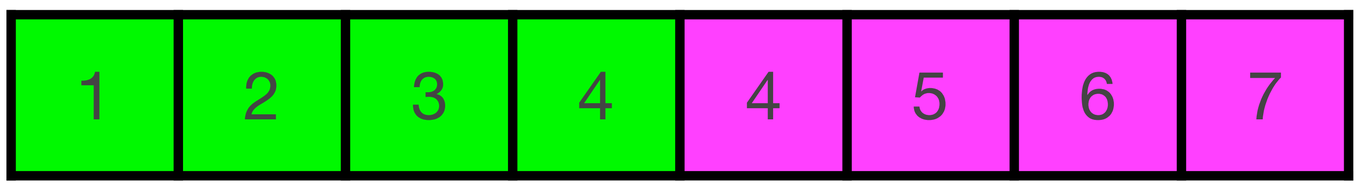
back front




enqueue(42)

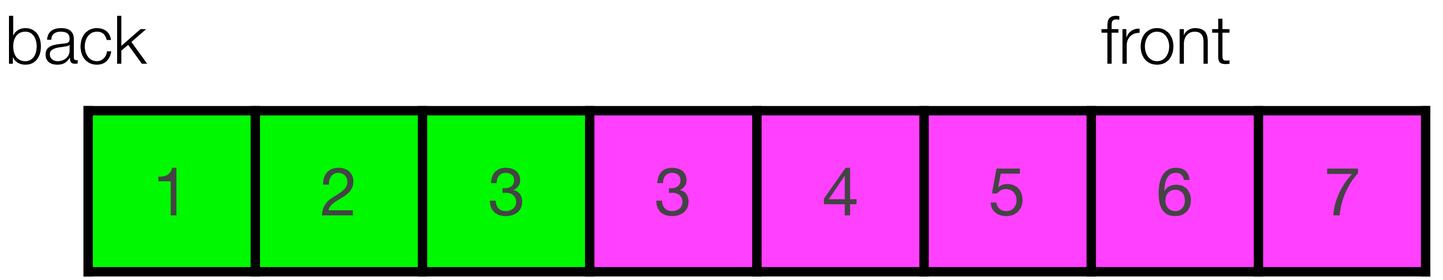
# Excuse Me, Coming Through

back front



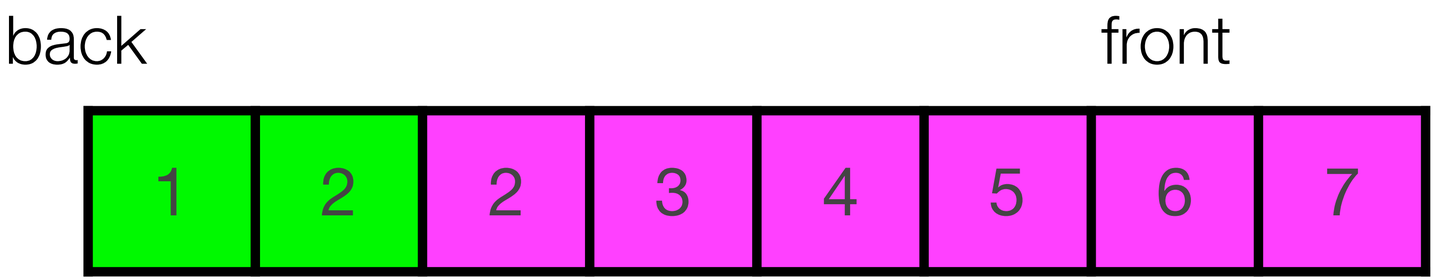

enqueue(42)

# Excuse Me, Coming Through




enqueue(42)

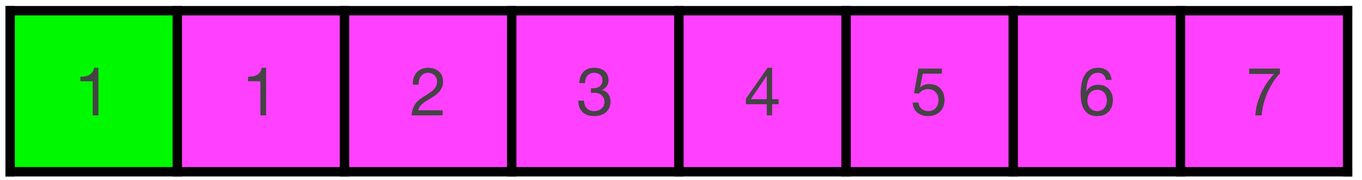
# Excuse Me, Coming Through




enqueue(42)

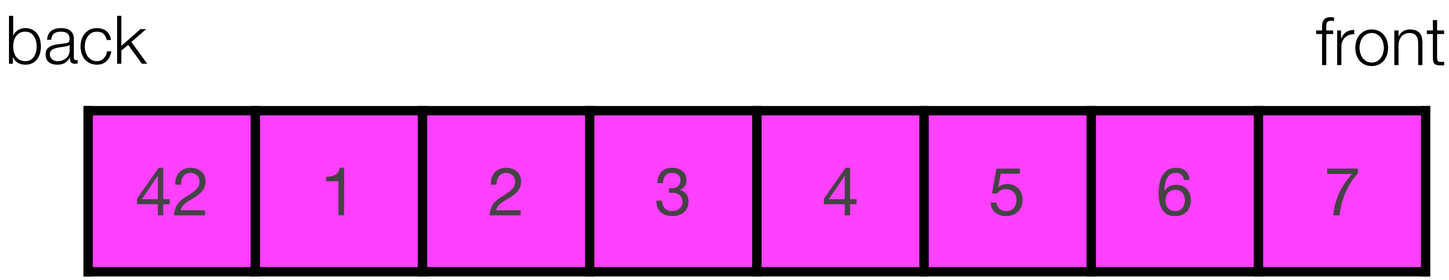
# Excuse Me, Coming Through

back front




enqueue(42)

# Excuse Me, Coming Through




enqueue(42)

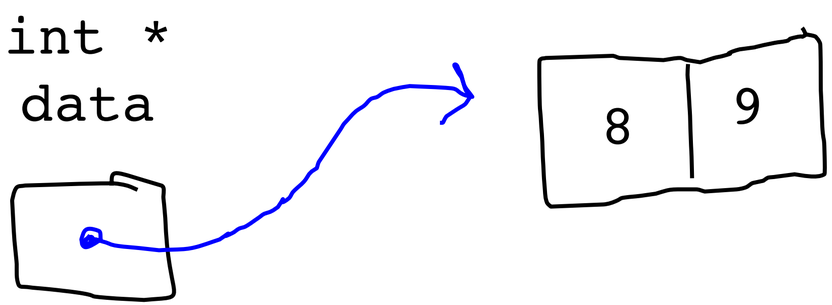
# Queue as Vector: Big O

Enqueue: O(n)

Dequeue: O(1)

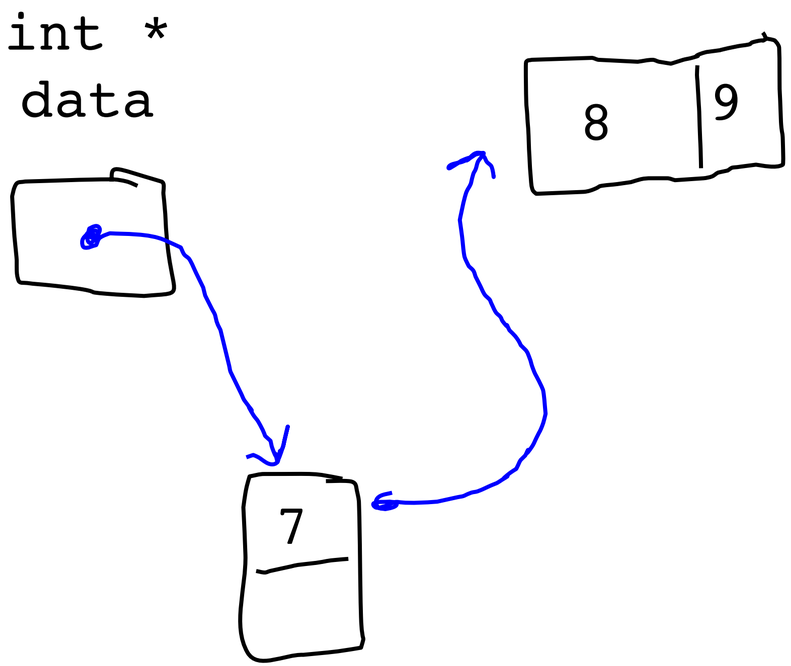

WE CAN DO
BETTER

# And Now for Something Completely Different



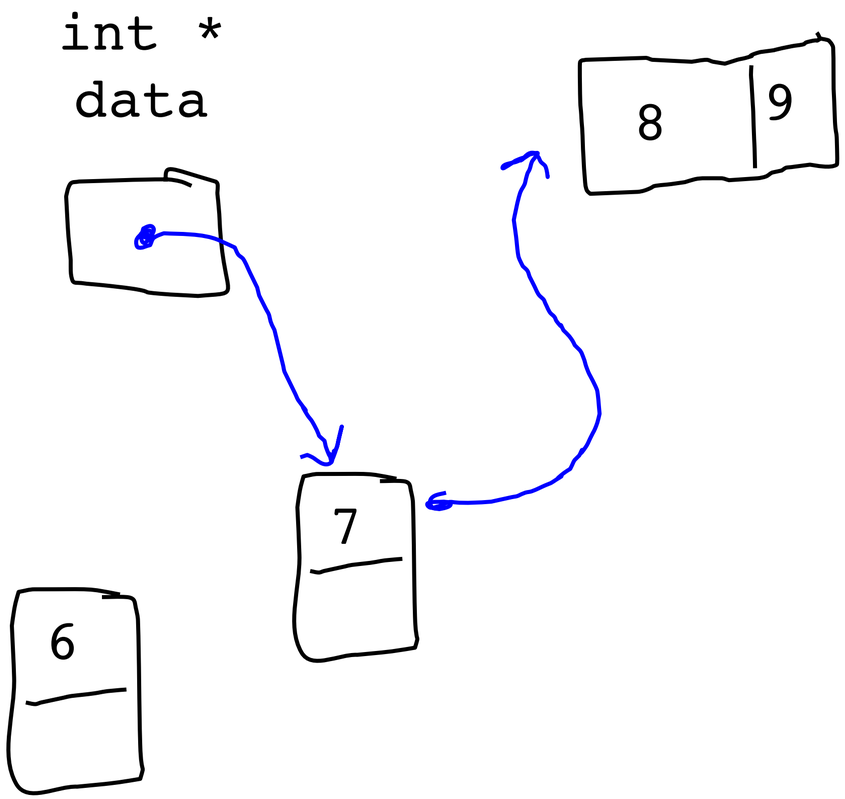

enqueue(7)

# And Now for Something Completely Different



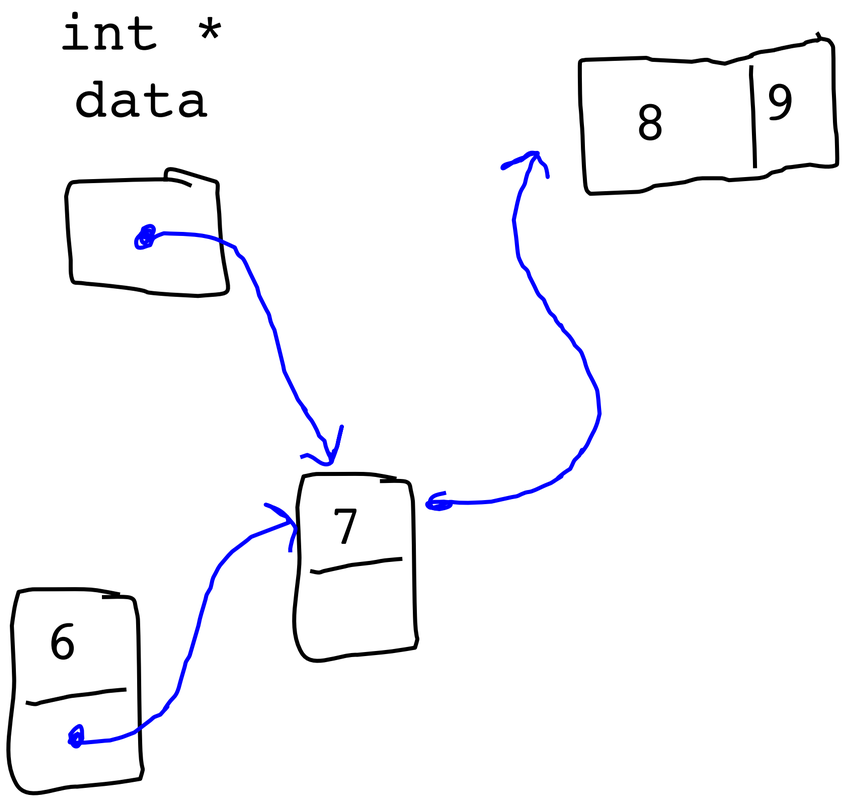

enqueue(7)

# And Now for Something Completely Different



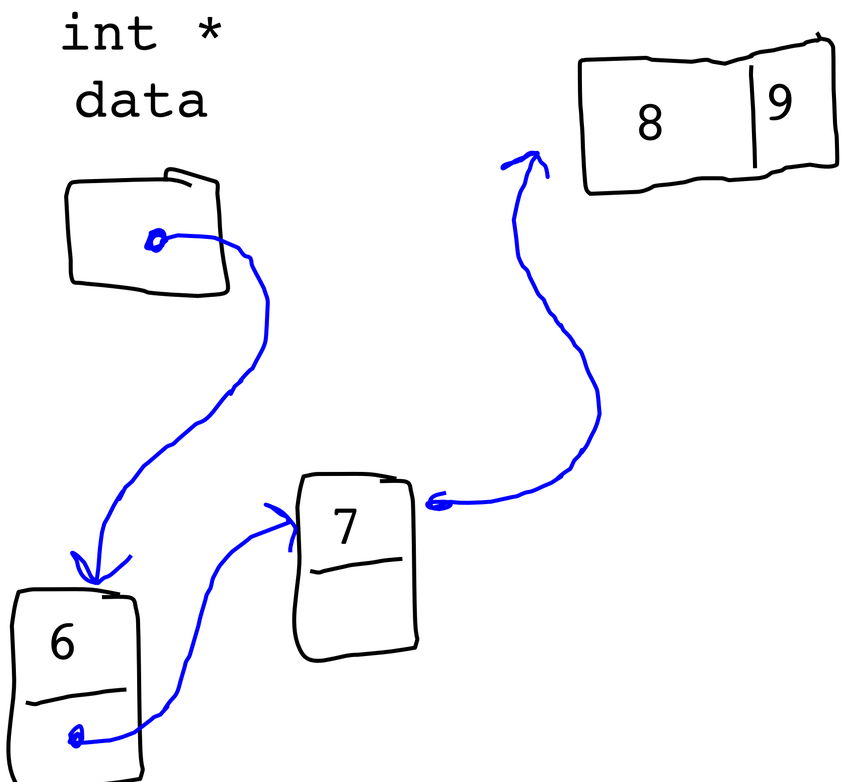

enqueue(6)

# And Now for Something Completely Different



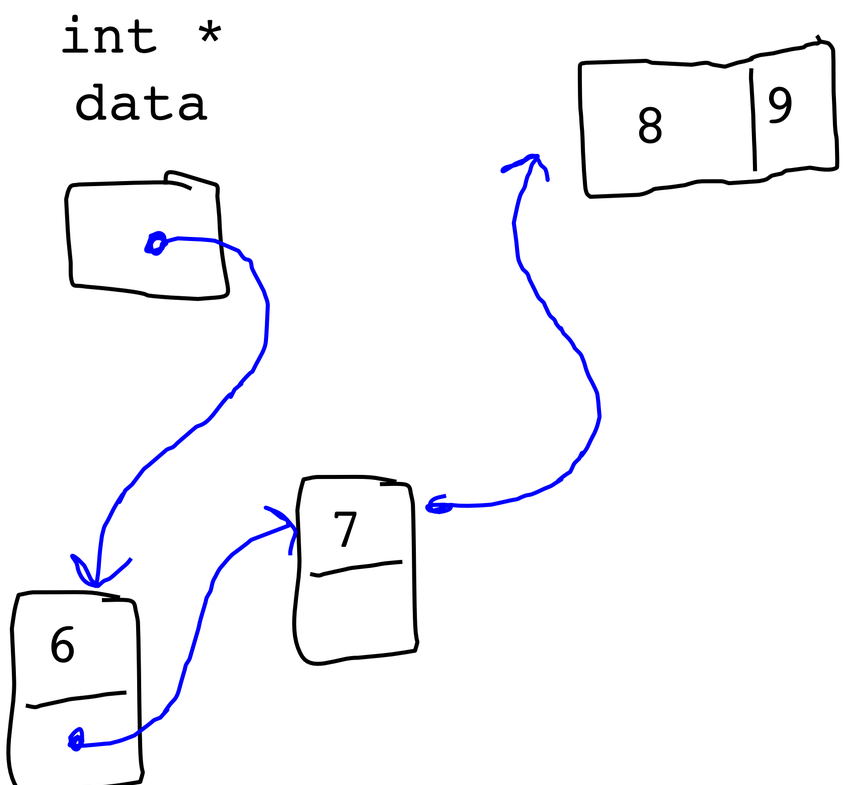

enqueue(6)

# And Now for Something Completely Different




enqueue(6)

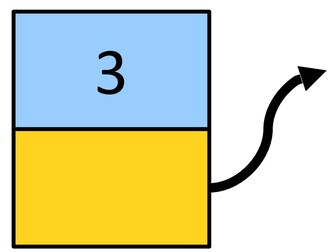
# And Now for Something Completely Different

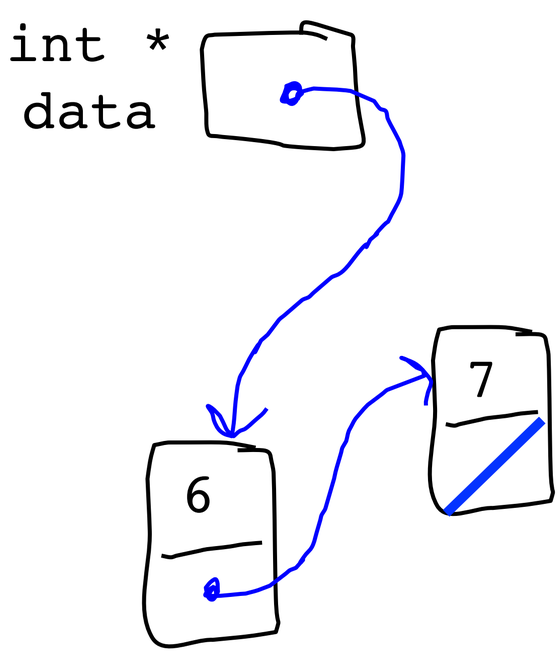



Now we have a way to add to the front in O(1) time!

# Linked List

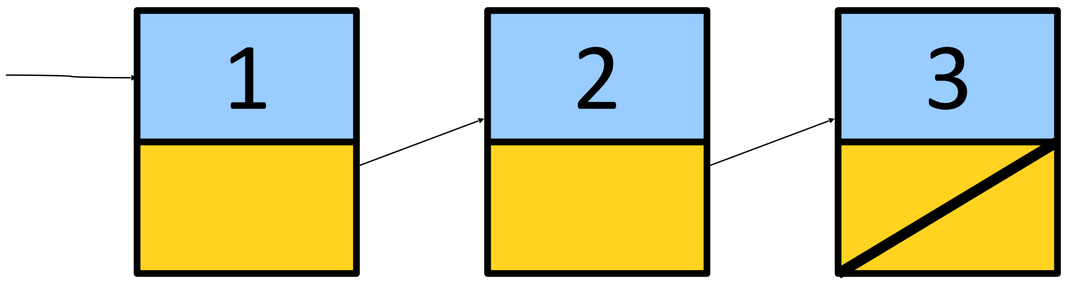
- A linked list is a chain of nodes.
- Each node contains two pieces of information:
  - Some piece of data that is stored in the sequence
  - A link to the next node in the list.
- We can traverse the list by starting at the first cell and repeatedly following its link.



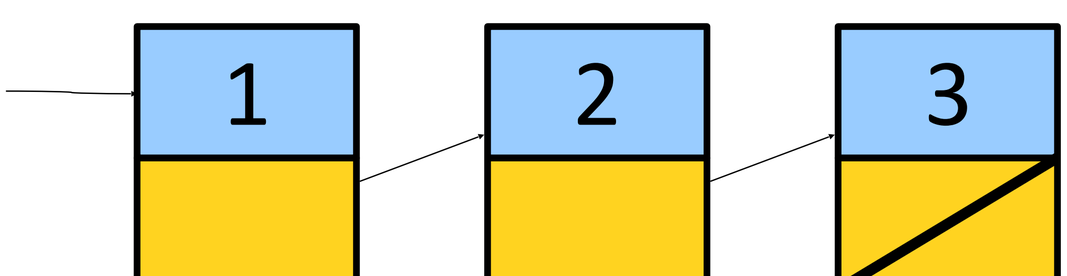
# Linked Lists

- A linked list is a data structure for storing a sequence of elements.
- Each element is stored separately from the rest.
- The elements are then chained together into a sequence.



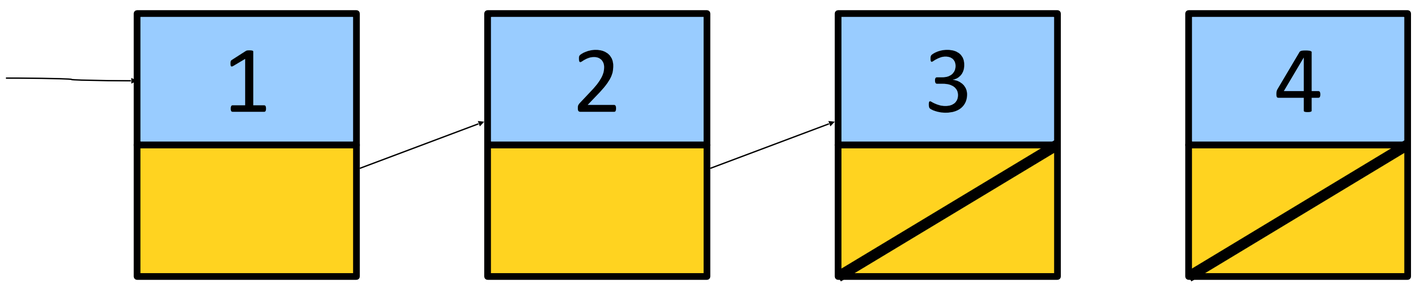

# Linked Lists

- A linked list is a data structure for storing a sequence of elements.
- Each element is stored separately from the rest.
- The elements are then chained together into a sequence.



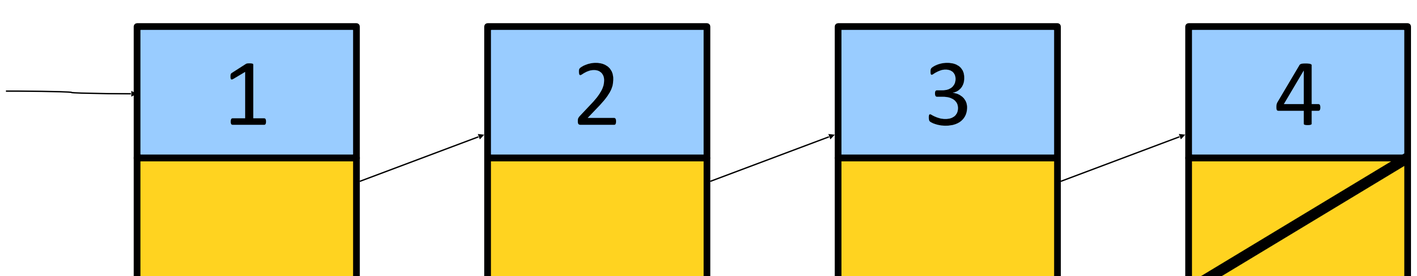
# Linked Lists

- A linked list is a data structure for storing a sequence of elements.
- Each element is stored separately from the rest.
- The elements are then chained together into a sequence.



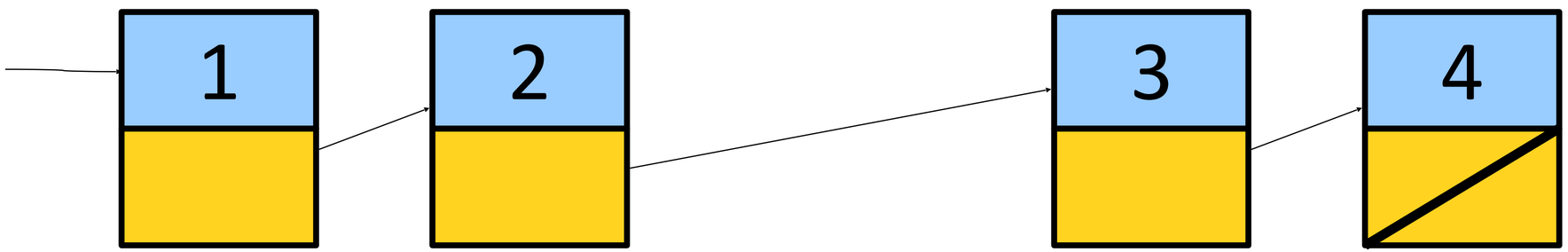
# Linked Lists

- A linked list is a data structure for storing a sequence of elements.
- Each element is stored separately from the rest.
- The elements are then chained together into a sequence.



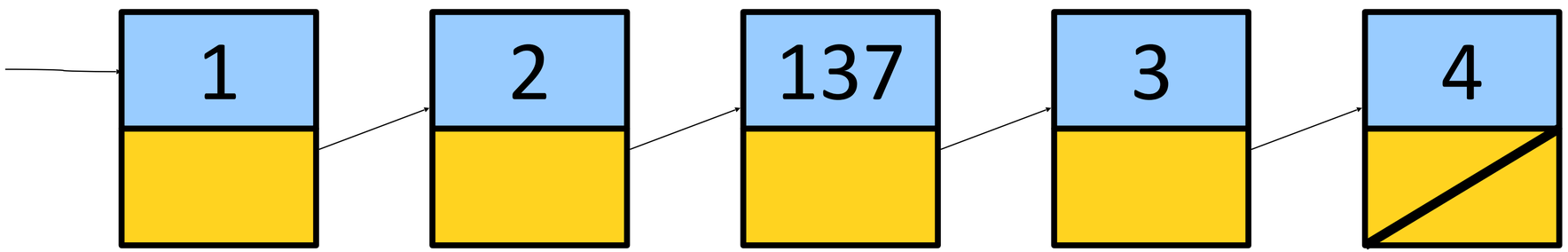
# Linked Lists

- A linked list is a data structure for storing a sequence of elements.
- Each element is stored separately from the rest.
- The elements are then chained together into a sequence.



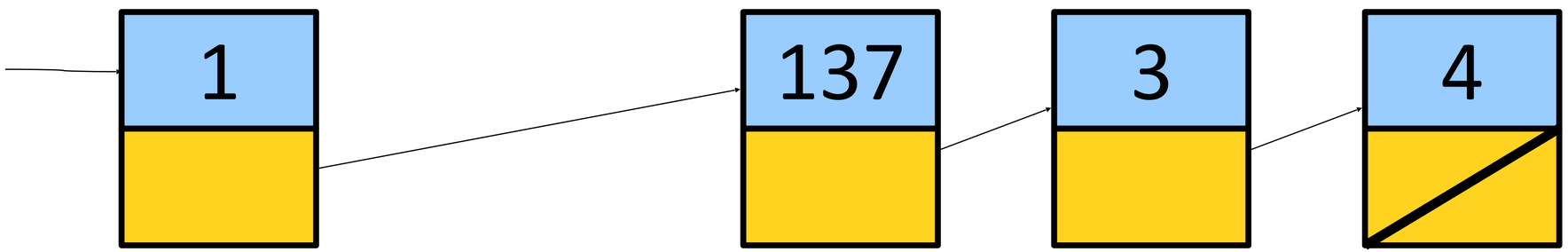
# Linked Lists

- A linked list is a data structure for storing a sequence of elements.
- Each element is stored separately from the rest.
- The elements are then chained together into a sequence.



# Linked Lists

- A linked list is a data structure for storing a sequence of elements.
- Each element is stored separately from the rest.
- The elements are then chained together into a sequence.



# Why Linked Lists?

- Can efficiently splice new elements into the list or remove existing elements anywhere in the list.
- Never have to do a massive copy step;
- Has some tradeoffs; we'll see this later.

# Linked List Structure

- For simplicity, let's assume we're building a linked list of strings.
- We can represent a node in the linked list as a structure:

```
struct Node {
    string value;
    /* ? */ next;
};
```

# Linked List of Strings

- For simplicity, let's assume we're building a linked list of strings.
- We can represent a node in the linked list as a structure:

```
struct Node {
    string value;
    Node* next;
};
```

# Linked List of Strings

- For simplicity, let's assume we're building a linked list of strings.
- We can represent a node in the linked list as a structure:

```
struct Node {
    string value;
    Node* next;
};
```

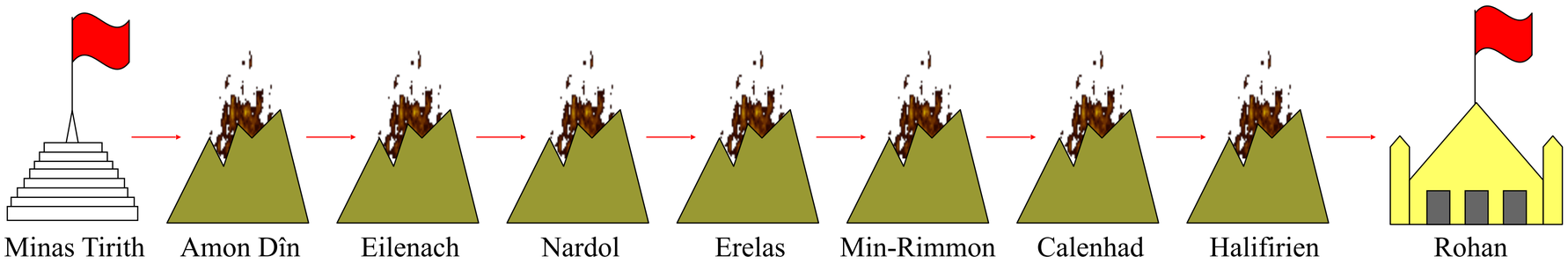
- The structure is defined recursively!

# Always!

Draw a picture

# Lord of the Linked Lists

In a scene that was brilliantly captured in Peter Jackson's film adaptation of *The Return of the King*, Rohan is alerted to the danger to Gondor by a succession of signal fires moving from mountain top to mountain top. This scene is a perfect illustration of the idea of message passing in a linked list.

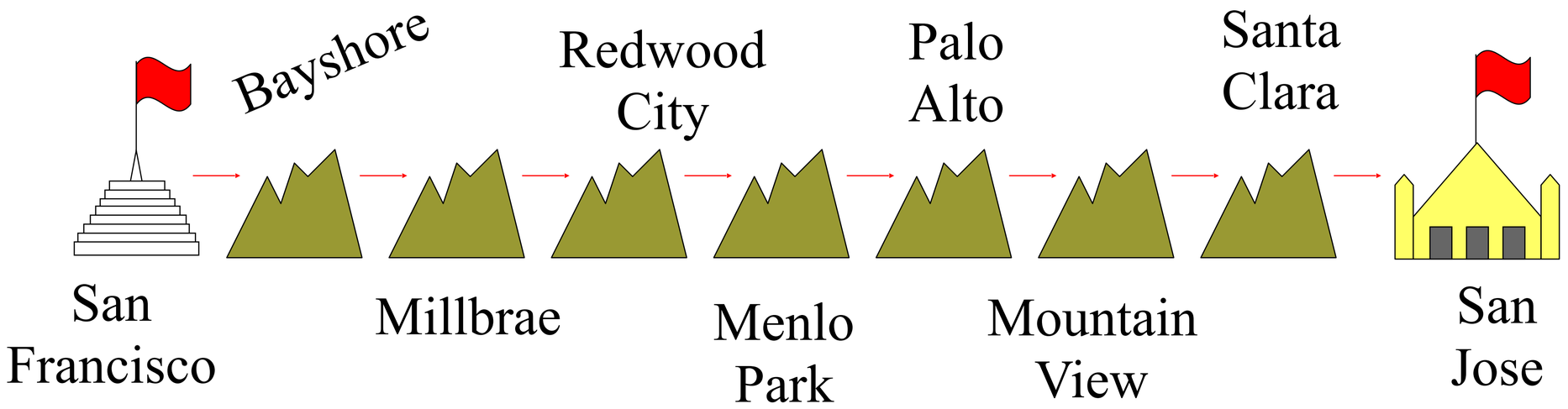



THE LORD OF THE RINGS
THE RETURN OF THE KING
TM

https://www.youtube.com/watch?v=i6LGJ7evrAg

# Lord of the Linked Lists

Step 1: Make this linked list




Step 2: Light the fires….

```
Lighting the fire of San Francisco!
```

# Lord of the Linked Lists

```
struct Tower {
   string name;   /* The name of this tower     */
   Tower *link;   /* Pointer to the next tower */
};
```

# Lord of the Linked Lists

```
// add the first tower
Tower * head = new Tower;
head->name = “San Jose”;
head->link = NULL;
```

# Lord of the Linked Lists

```
// add the first tower
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;
```

# Lord of the Linked Lists

```
// add the first tower
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;
```

# Lord of the Linked Lists

```
// add the first tower
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;
```

# Linked List Trace

```
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);
```

```
struct Tower{
    string name;
    Tower * link;
};
```

```
Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```

# Linked List Trace

```
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);
```

```
struct Tower{
    string name;
    Tower * link;
};
```

```
Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```

# Linked List Trace

```
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);
```

```
struct Tower{
    string name;
    Tower * link;
};
```

```
Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```

# Linked List Trace

```
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);
```

```
struct Tower{
    string name;
    Tower * link;
};
```

```
Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```

# Linked List Trace

```
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);
```

```
struct Tower{
    string name;
    Tower * link;
};
```

```
Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```

# Linked List Trace

```
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);
```

```
struct Tower{
    string name;
    Tower * link;
};
```

```
Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```

# Linked List Trace

```
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);
```

```
struct Tower{
    string name;
    Tower * link;
};
```

```
Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```

# Linked List Trace

```
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);
```

```
struct Tower{
    string name;
    Tower * link;
};
```

```
Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```

# Linked List Trace

```
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);
```

```
struct Tower{
    string name;
    Tower * link;
};
```

```
Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```

# Linked List Trace

```
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);
```

```
struct Tower{
    string name;
    Tower * link;
};
```

```
Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```

# Linked List Trace

```
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);
```

```
struct Tower{
    string name;
    Tower * link;
};
```

```
Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```

# Linked List Trace

```
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);
```

```
struct Tower{
    string name;
    Tower * link;
};
```

```
Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```

# Linked List Trace

```
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);
```

```
struct Tower{
    string name;
    Tower * link;
};
```

```
Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```

# Linked List Trace

```
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);
```

```
struct Tower{
    string name;
    Tower * link;
};
```

```
Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```

# Linked List Trace

```
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);
```

```
struct Tower{
    string name;
    Tower * link;
};
```

```
Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```

# Linked List Trace

```
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);
```

```
struct Tower{
    string name;
    Tower * link;
};
```

```
Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```

# Linked List Trace

```
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);
```

```
struct Tower{
    string name;
    Tower * link;
};
```

```
Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```

# Linked List Trace

```
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);
```

```
struct Tower{
    string name;
    Tower * link;
};
```

```
Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```

# Linked List Trace

```
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);
```

```
struct Tower{
    string name;
    Tower * link;
};
```

```
Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```

# Linked List Trace

```
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);
```

```
struct Tower{
    string name;
    Tower * link;
};
```

```
Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```

# Linked List Trace

```
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);
```

```
struct Tower{
    string name;
    Tower * link;
};
```

```
Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```

# Linked List Trace

```
// main
Tower * head = new Tower;
head->name = "San Jose";
head->link = NULL;

head = createTower("Santa Clara", head);
head = createTower("Mountain View", head);
head = createTower("Palo Alto", head);
head = createTower("Menlo Park", head);
head = createTower("Redwood City", head);
head = createTower("Millbrae", head);
head = createTower("Bayshore", head);
head = createTower("San Francisco", head);
```

```
struct Tower{
    string name;
    Tower * link;
};
```

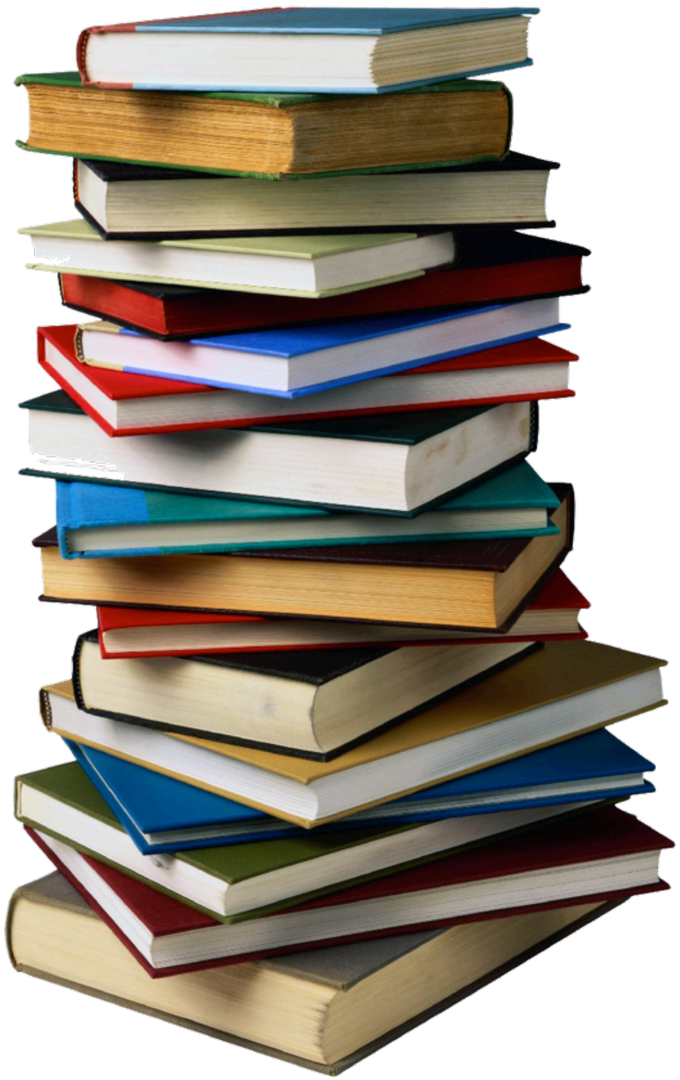
```
Tower *createTower(string name, Tower *link) {
    Tower *tp = new Tower;
    tp->name = name;
    tp->link = link;
    return tp;
}
```

```
void signal(Tower *start) {
    if (start != NULL) {
        cout << "Lighting " << start->name << endl;
        signal(start->link);
    }
}
```

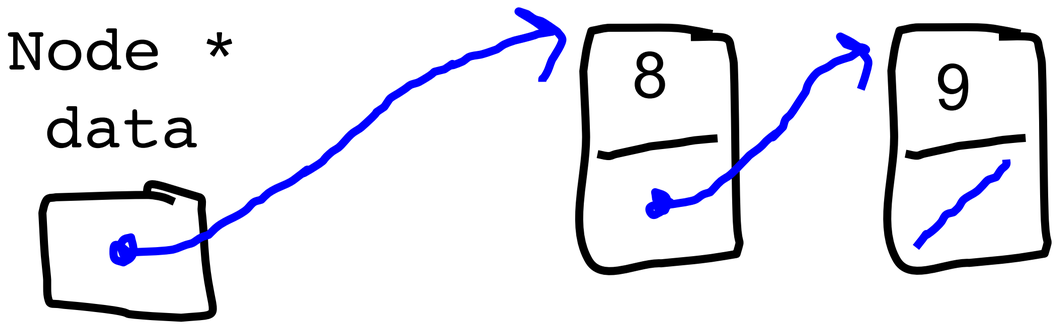
```
signal(head);
```

# Lord of the Linked Lists


# How is the Stack Implemented?



```
struct Node{
    int value;      /* The value of this elem    */
    Node *link;     /* Pointer to the next node */
};
```


Node *
data
8
9

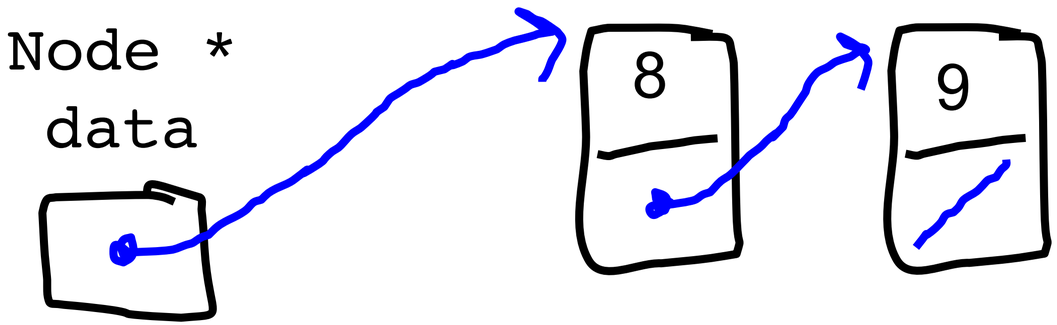

LELAND STANFORD JUNIOR UNIVERSITY
DIE LUFT DER FREIHEIT WEHT
1891

# Stack

```
class StackInt {                    // in StackInt.h
public:
    StackInt ();                    // constructor

    void push(int value); // append a value
    int pop();                      // return the first-in value

private:
    struct Node {
      int value;
      Node * link;
    };
    Node * data;        // member variables
};
```

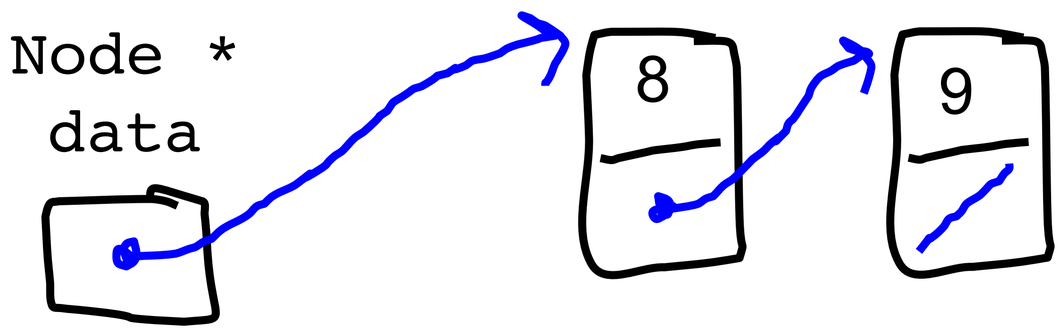
# Stack Implementation

```
void StackInt::push(int v) {
    Node * temp = new Node;
    temp->value = v;
    temp->link = data;
    data = temp;
}

int StackInt::pop() {
    int toReturn = data->value;
    Node * temp = data;
    data = temp->link;
    delete temp;
    return toReturn;
}
```


Node *
data
8
9

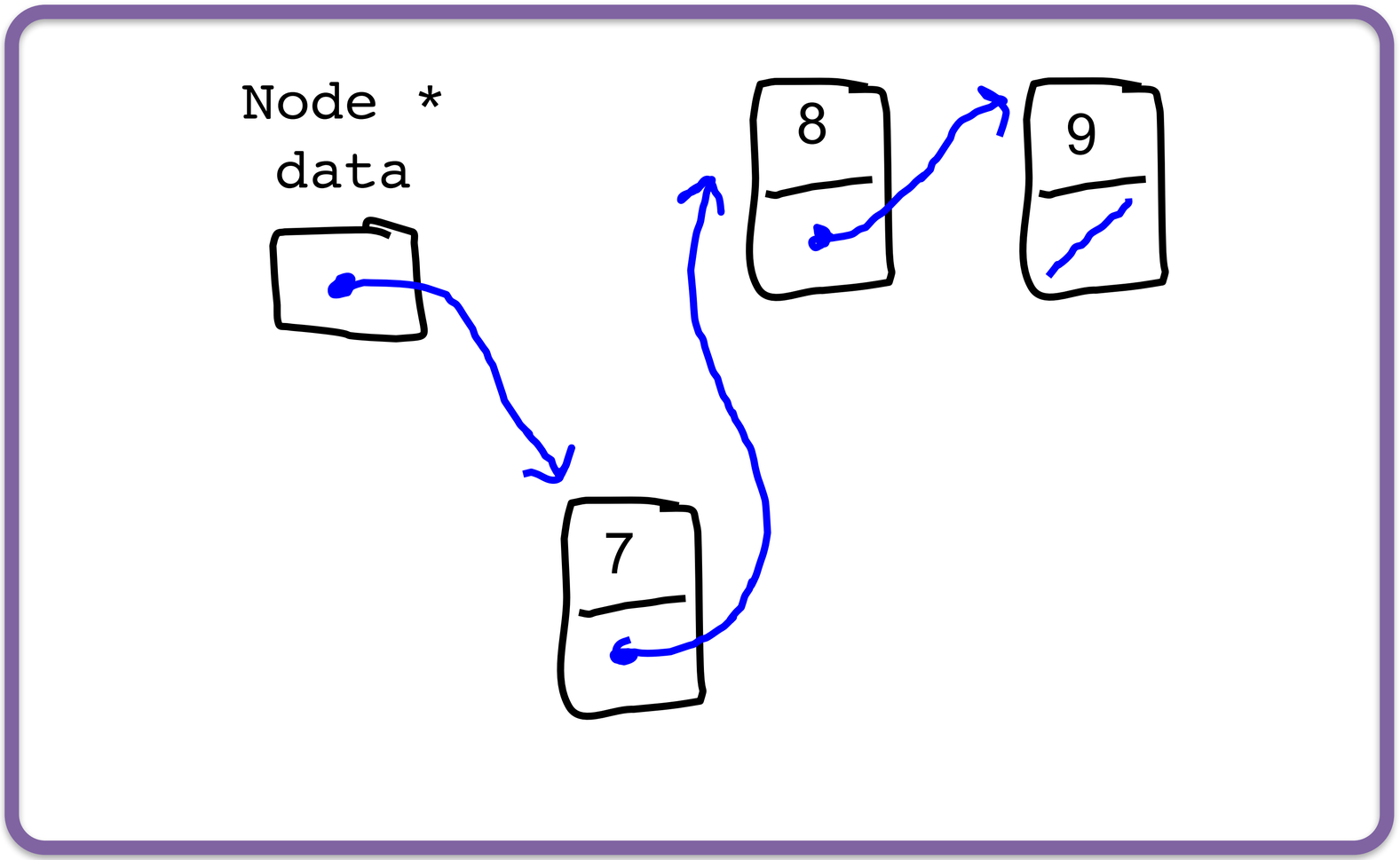

LELAND STANFORD JUNIOR UNIVERSITY
DIE LUFT DER FREIHEIT WEHT
1891

```
push(7);
```

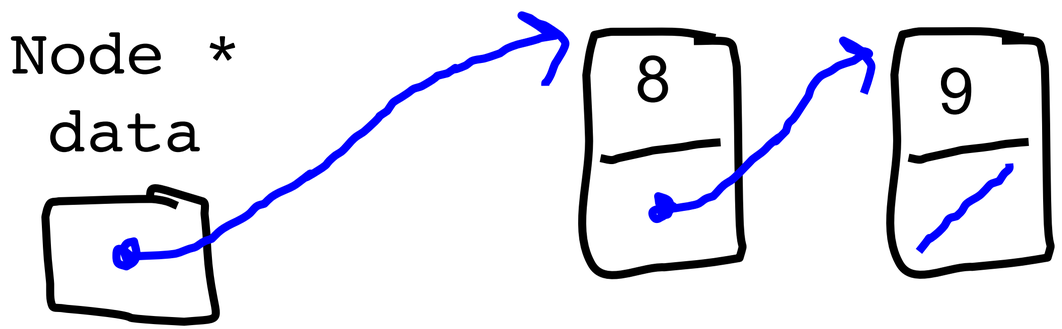

Node *
data
8
9


LELAND STANFORD JUNIOR UNIVERSITY
DIE LUFT DER FREIHEIT WEHT
1891

# Goal of Push



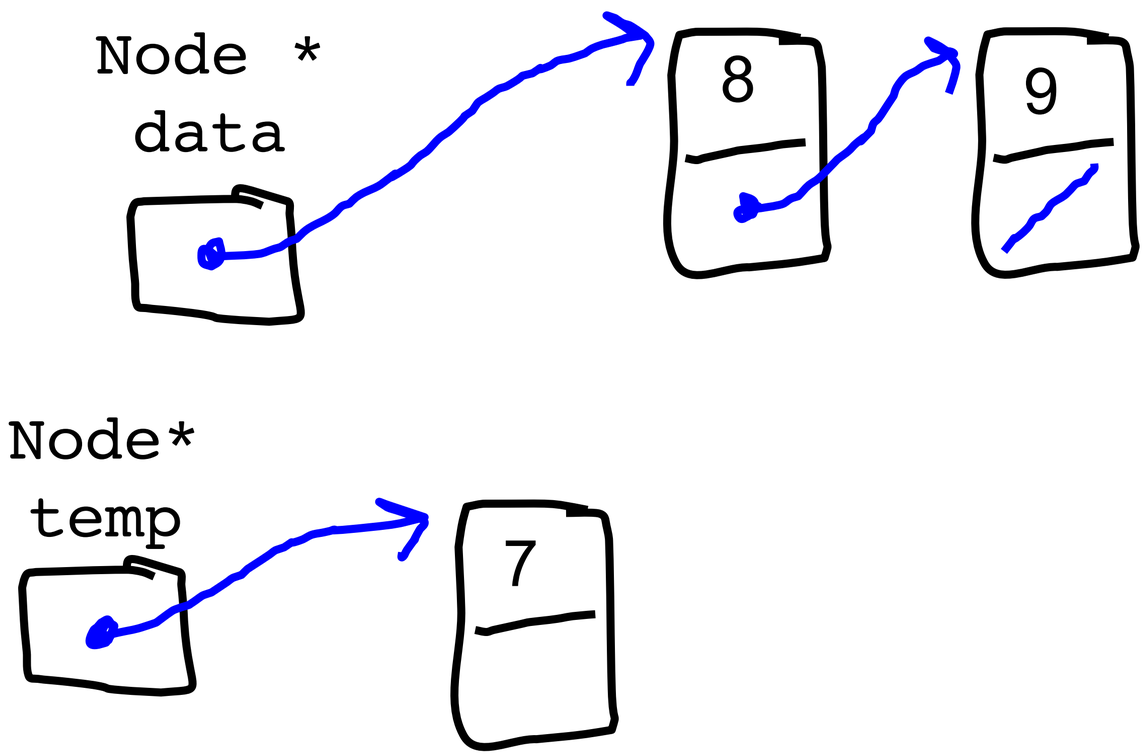
```
push(7);
```


Node *
data
8
9


LELAND STANFORD JUNIOR UNIVERSITY
DIE LUFT DER FREIHEIT WEHT
1891

# Stack is a Linked List

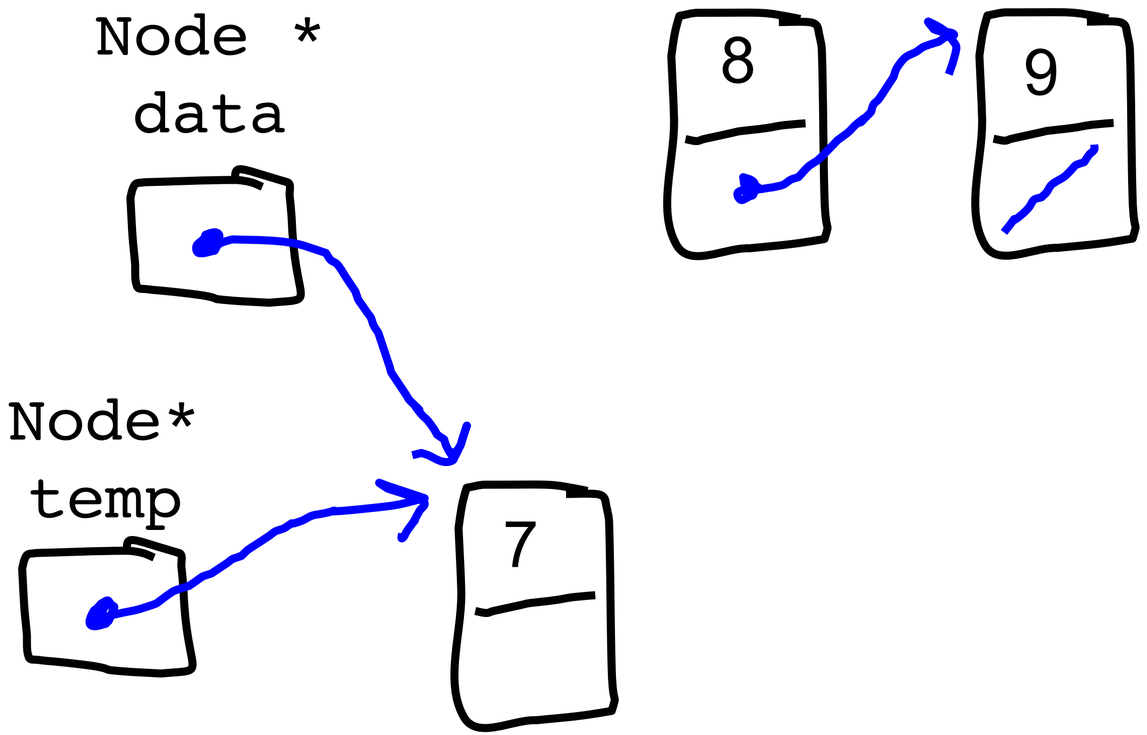
```
push(7);
```

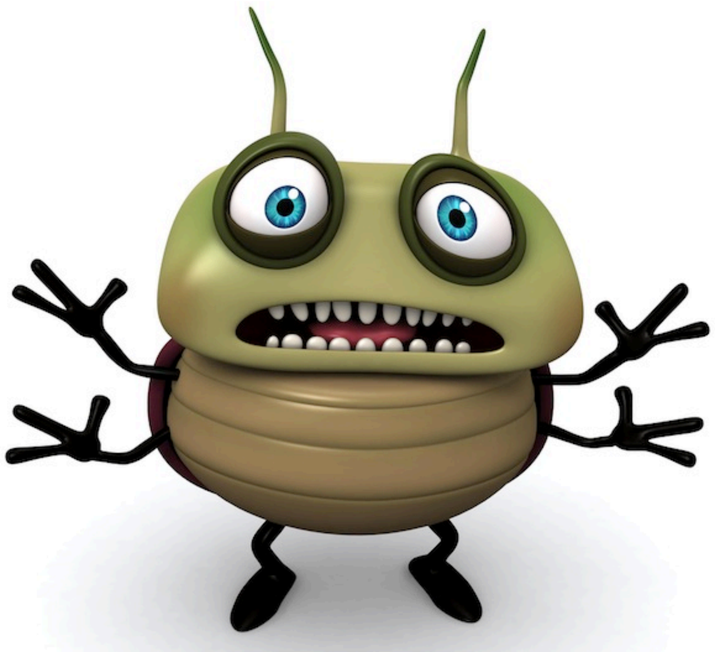



```
Node * temp =  new Node;
temp -> value = 7;
```

# Stack is a Linked List

```
push(7);
```

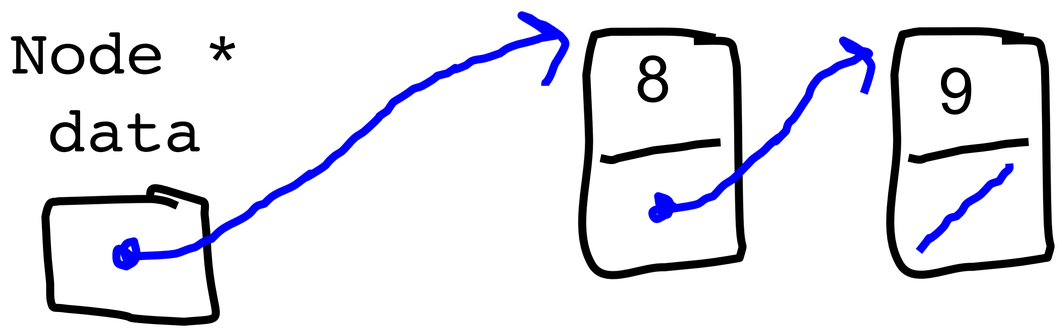


```
data = temp;
```

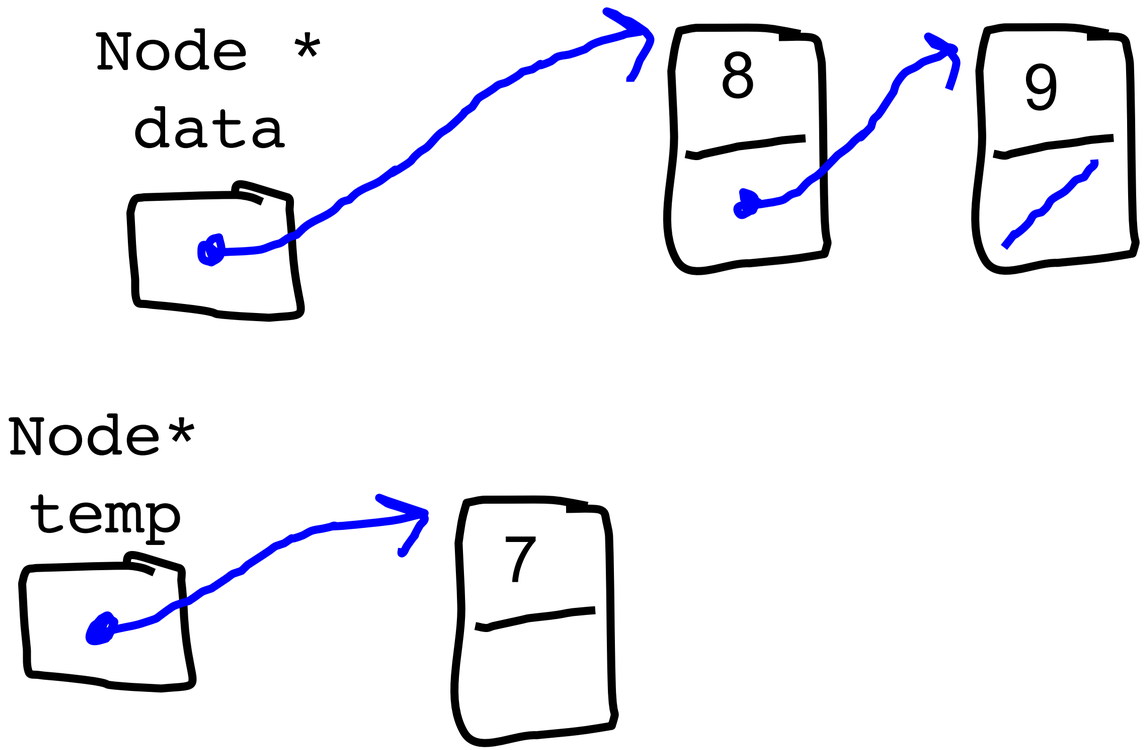
BRAAWWRRRR!

```
push(7);
```


Node *
data
8
9

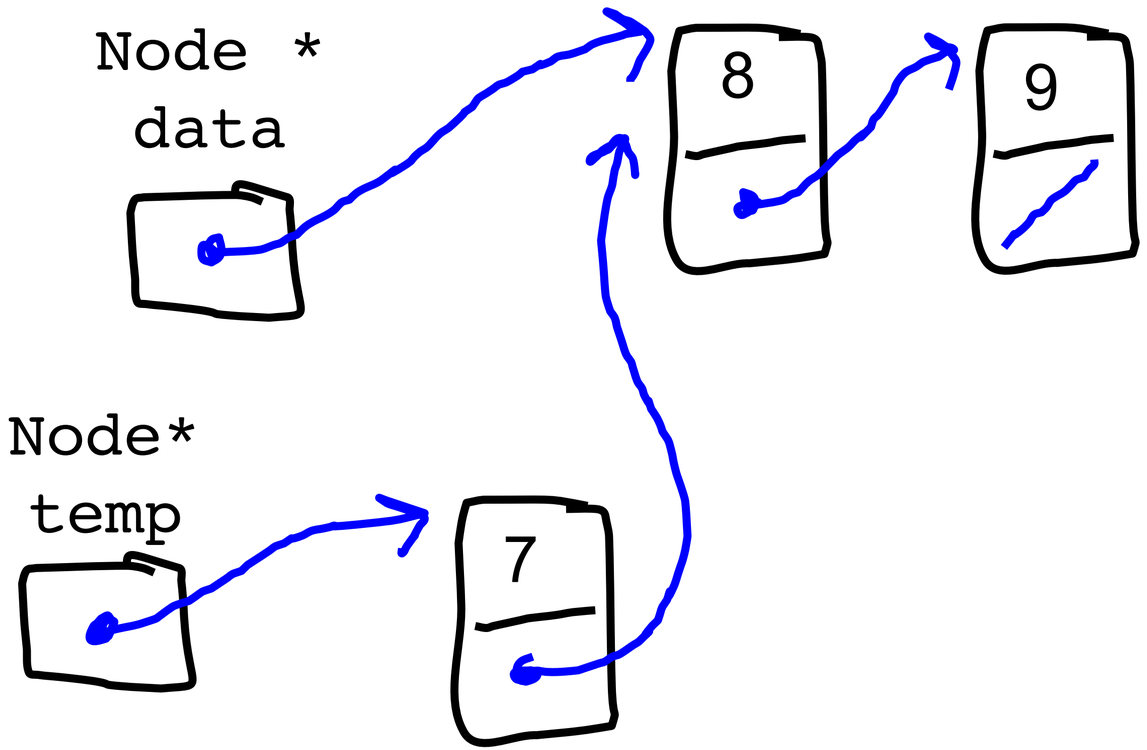

LELAND STANFORD JUNIOR UNIVERSITY
DIE LUFT DER FREIHEIT WEHT
1891

```
push(7);
```



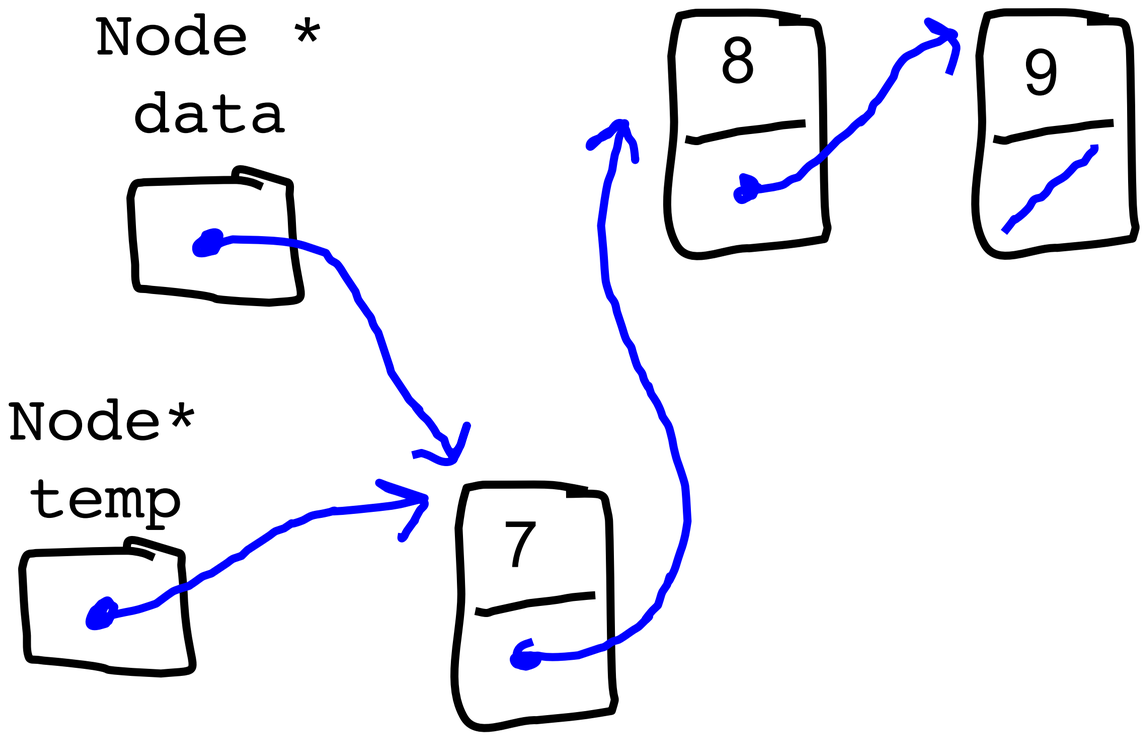

```
Node * temp =  new Node;
temp -> value = 7;
```

```
push(7);
```



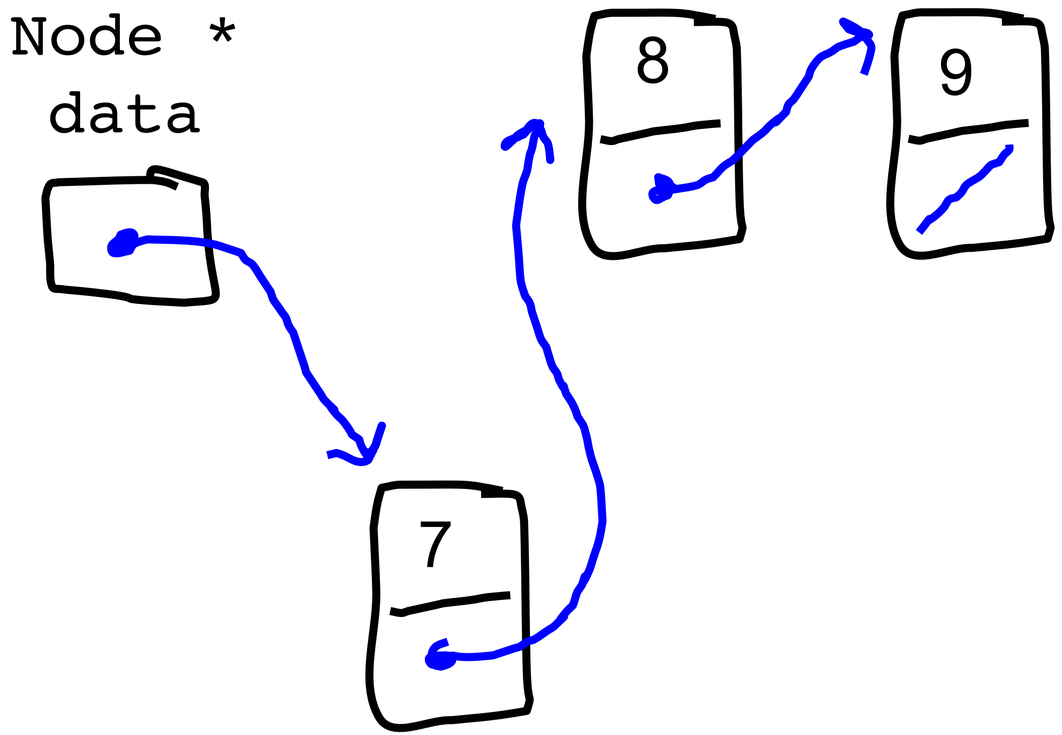

```
temp -> link = data;
```

```
push(7);
```




```
data = temp;
```

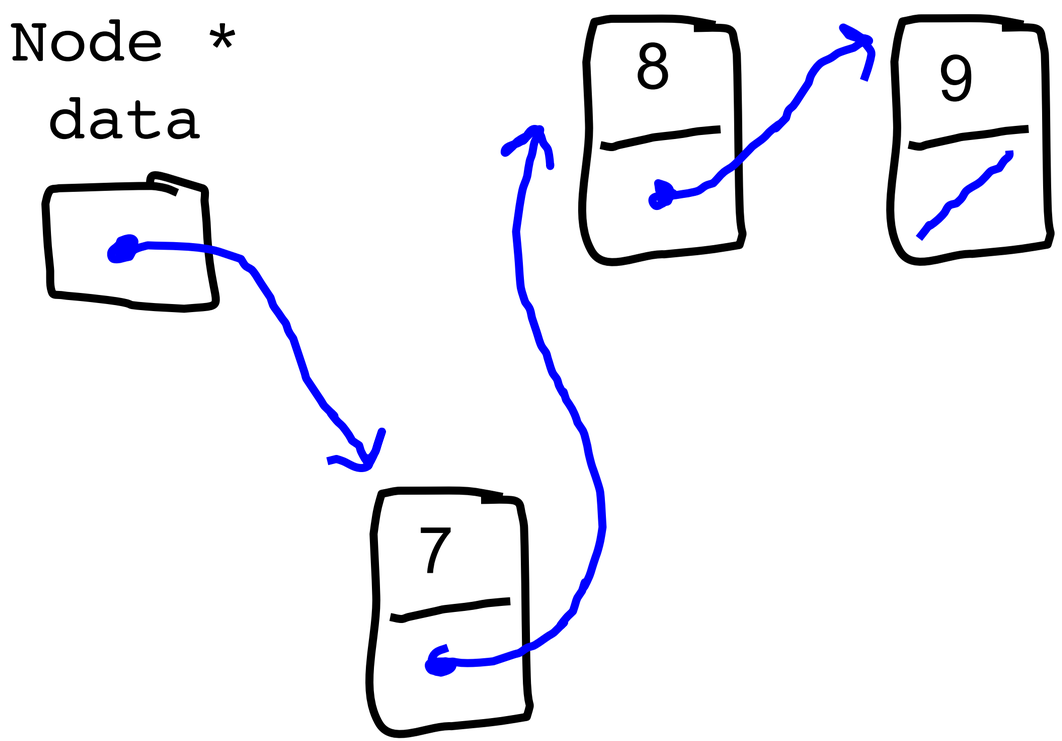
```
push(7);
```




*exit function*

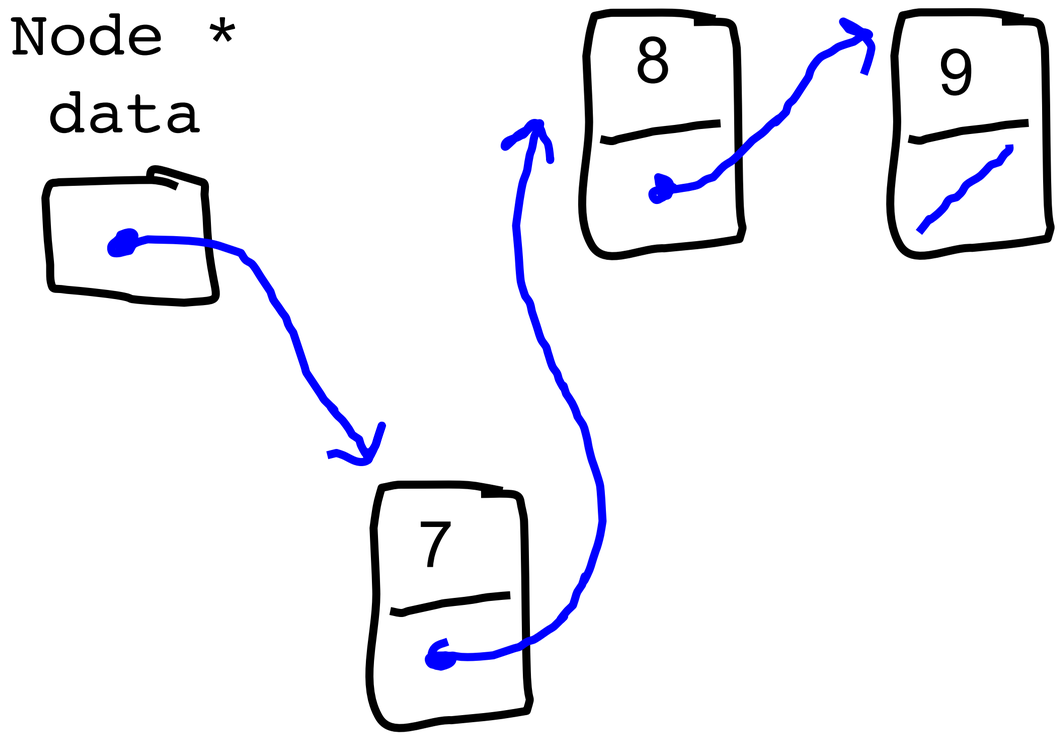
# Stack is a Linked List

```
pop();
```



# Stack is a Linked List

```
pop();
```



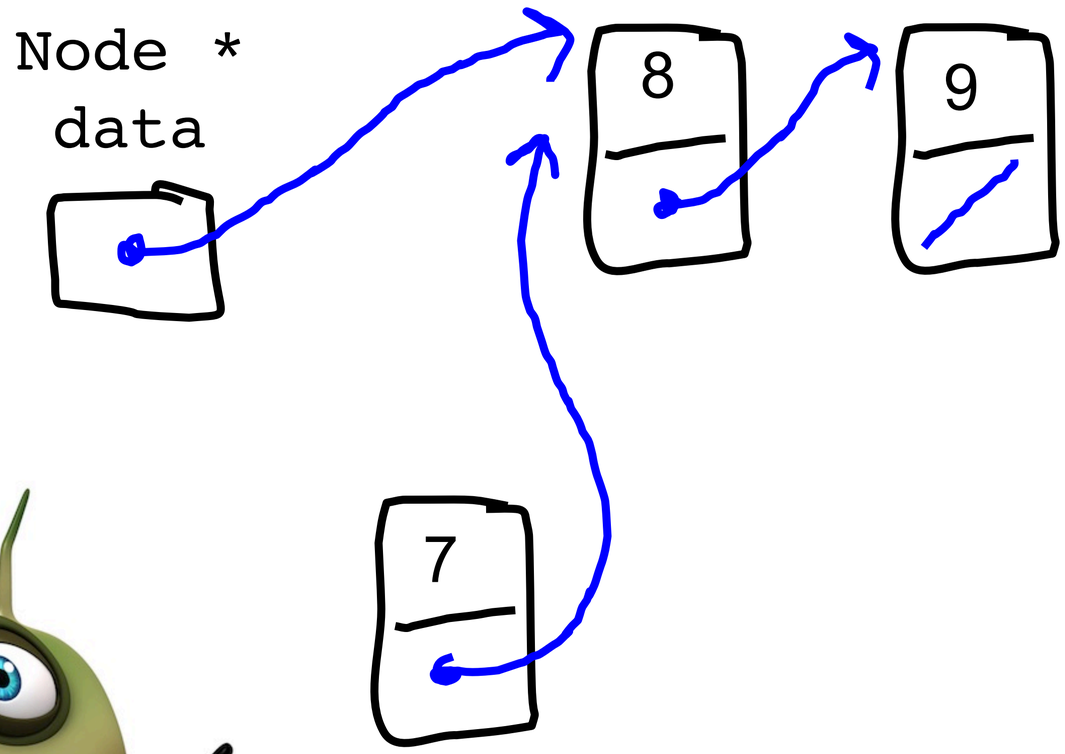

```
int toReturn
```

```
7
```

```
int toReturn = data->value;
```

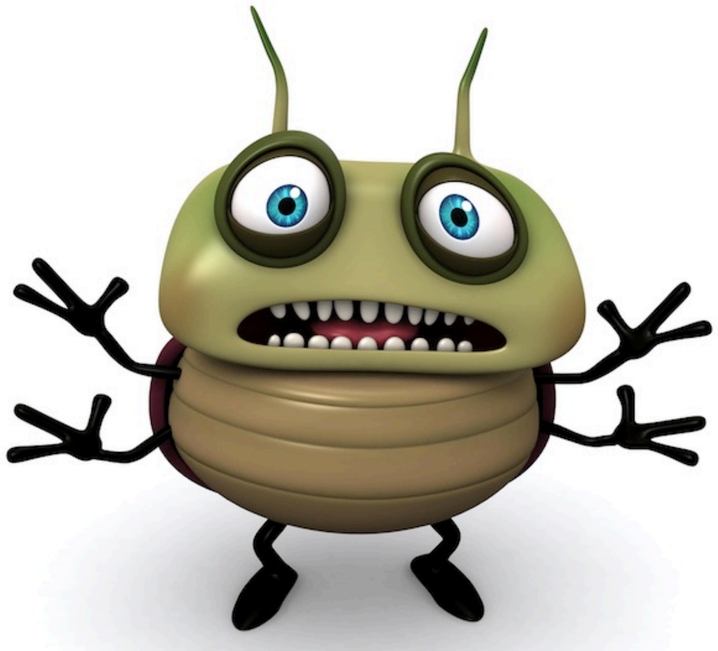
# Stack is a Linked List

```
pop();
```




```
int toReturn
```

```
7
```

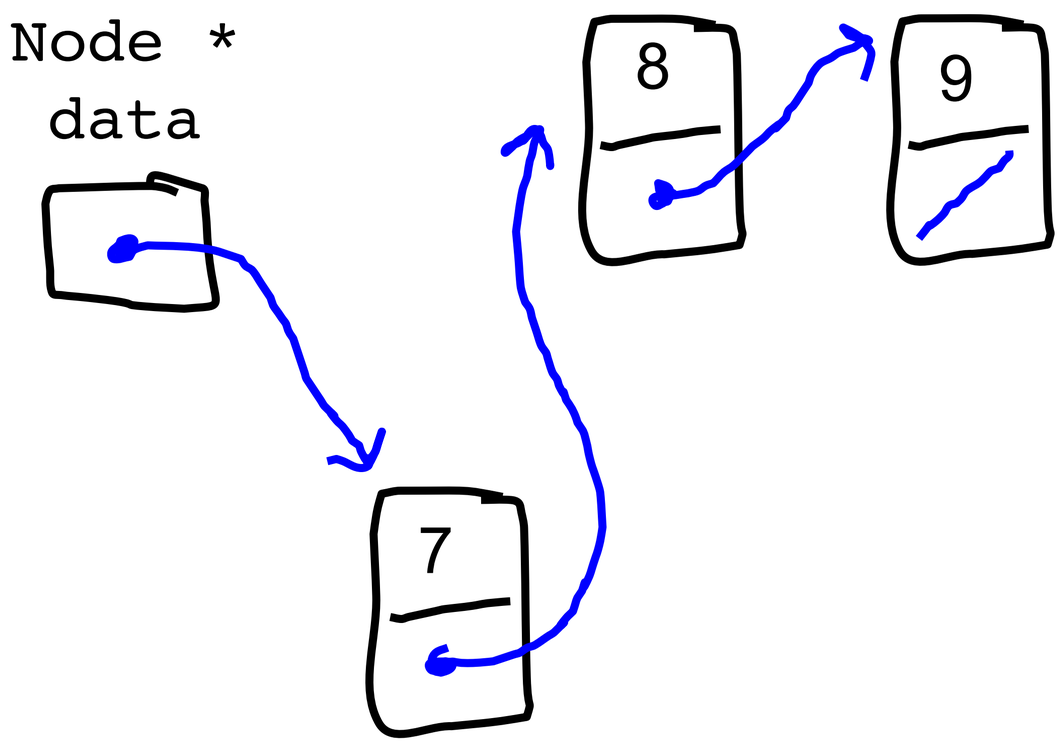


```
data = data->link;
```

That didn’t work. Let’s try again…

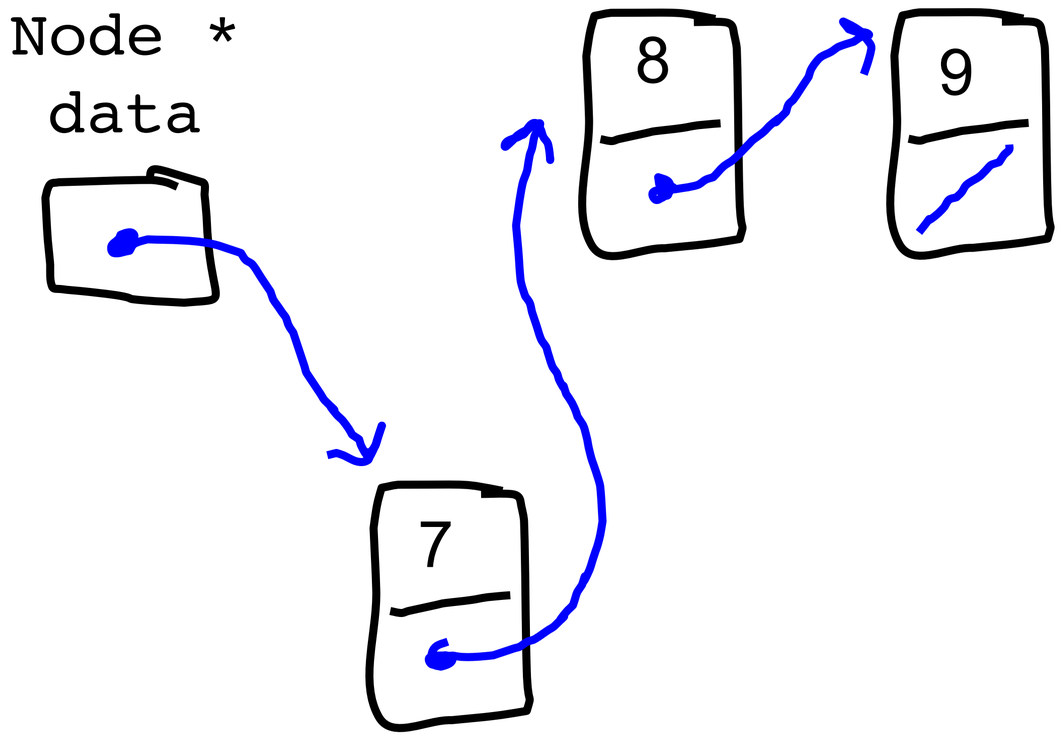
# Stack is a Linked List

```
pop();
```



# Stack is a Linked List

pop();



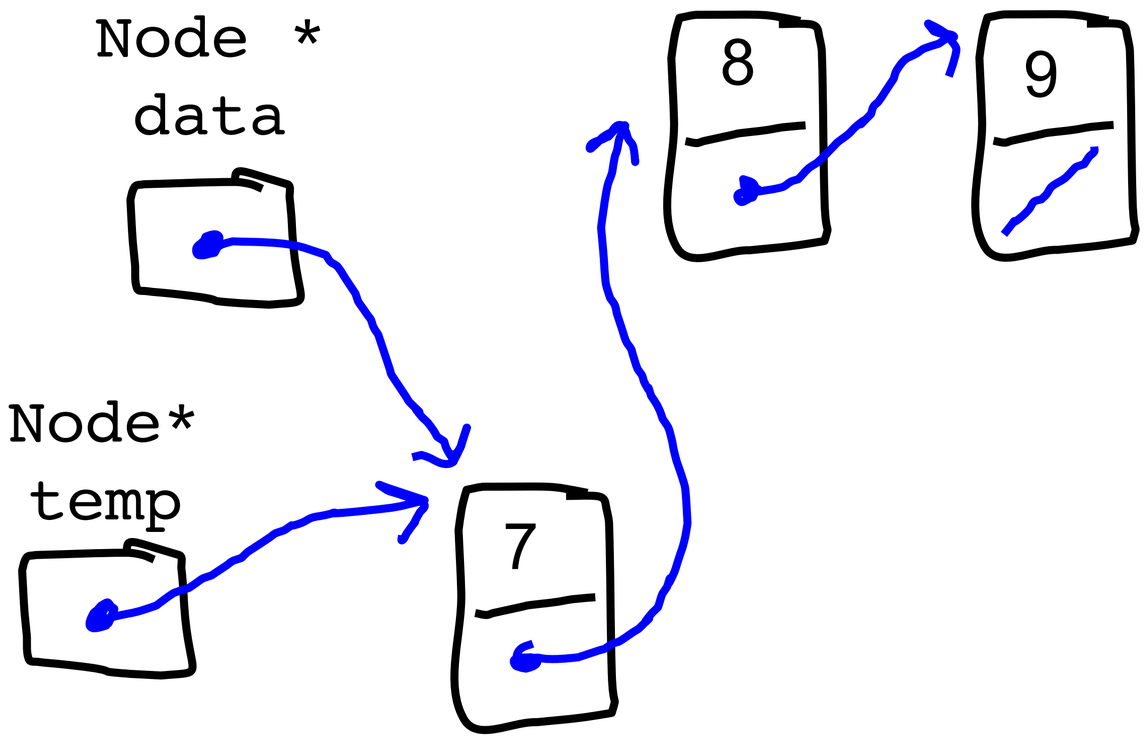

int toReturn

7

```
int toReturn = data->value;
```

# Stack is a Linked List

```
pop();
```

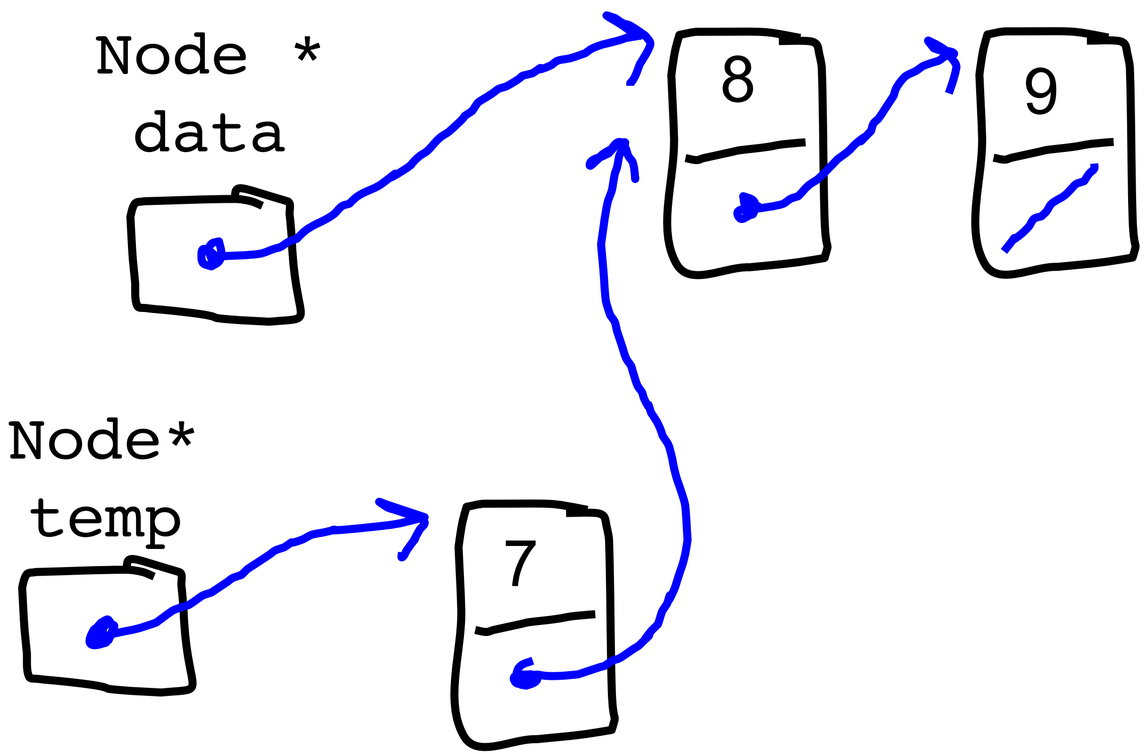



```
int toReturn
```

```
7
```

```
Node * temp = data;
```

# Stack is a Linked List




```
pop();
```

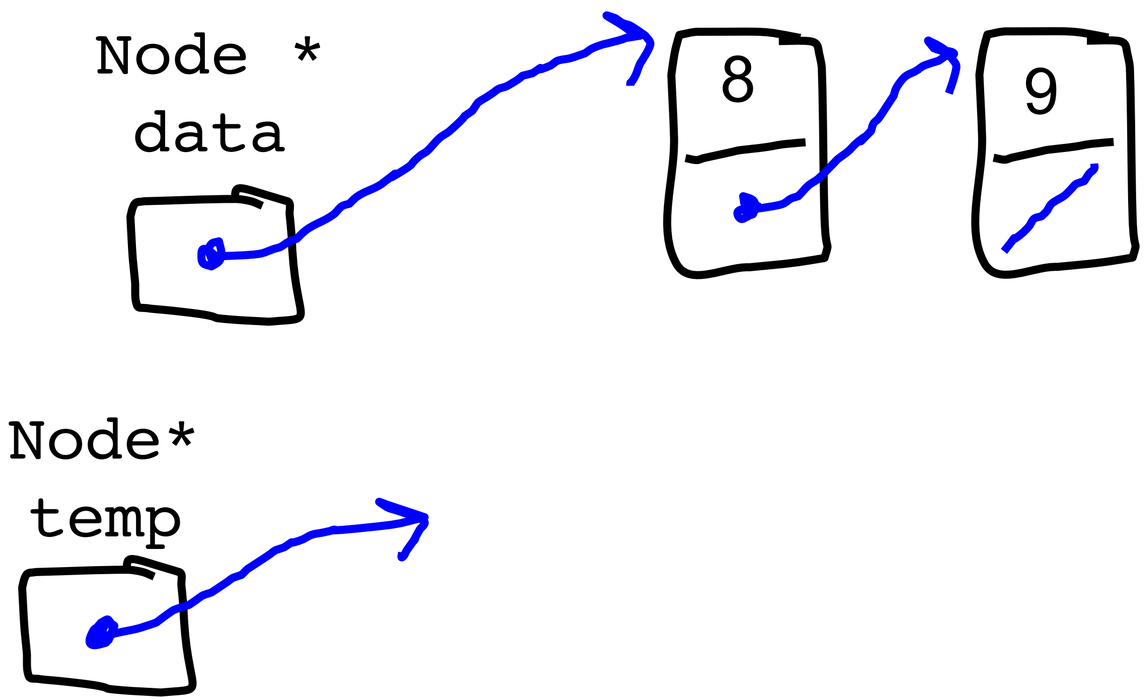
```
int toReturn
```

```
7
```

```
data = temp->link;
```

# Stack is a Linked List

```
pop();
```



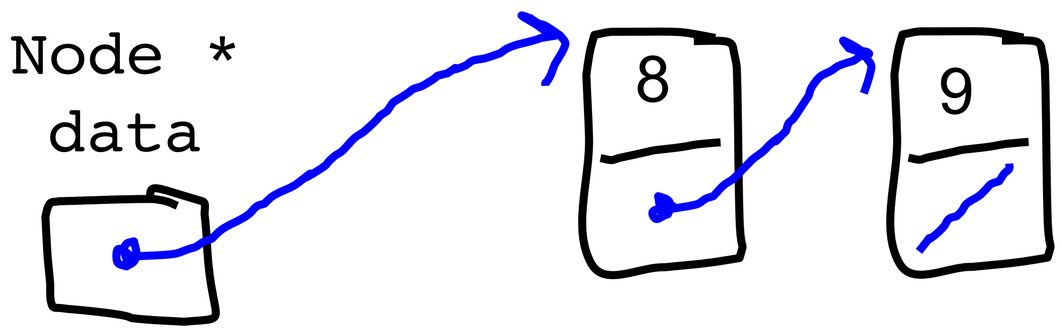

```
int toReturn
```

```
7
```

```
delete temp;
```

# Stack is a Linked List

```
pop();
```




```
int toReturn
```

```
7
```

```
return toReturn;
```

# Stack

```
class StackInt {                    // in StackInt.h
public:
    StackInt ();                    // constructor

    void push(value);                // append a value
    int pop();                      // return the first-in value

private:
    struct Node {
      int value;
      Node * link;
    };
    Node * data;                // member variables
};
```

# Stack Implementation

```
void StackInt::push(int v) {
    Node * temp = new Node;
    temp->value = v;
    temp->link = data;
    data = temp;
}

int StackInt::pop() {
    int toReturn = data->value;
    Node * temp = data;
    data = temp->link;
    delete temp;
    return toReturn;
}
```

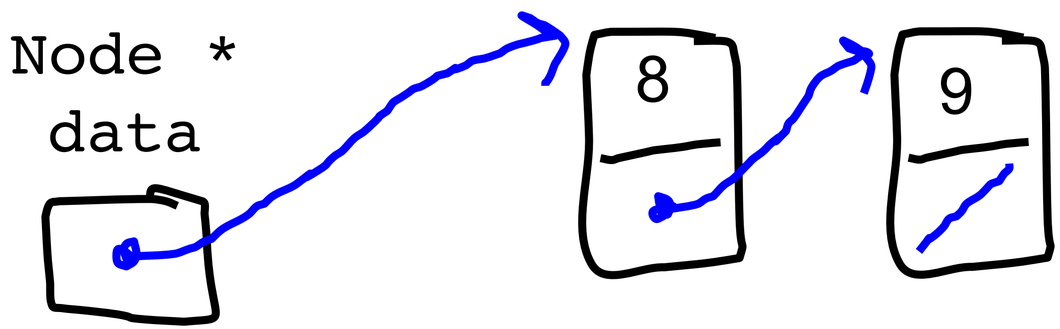
# Stack Implementation: Big O?

Big O of `push()`? O(1)

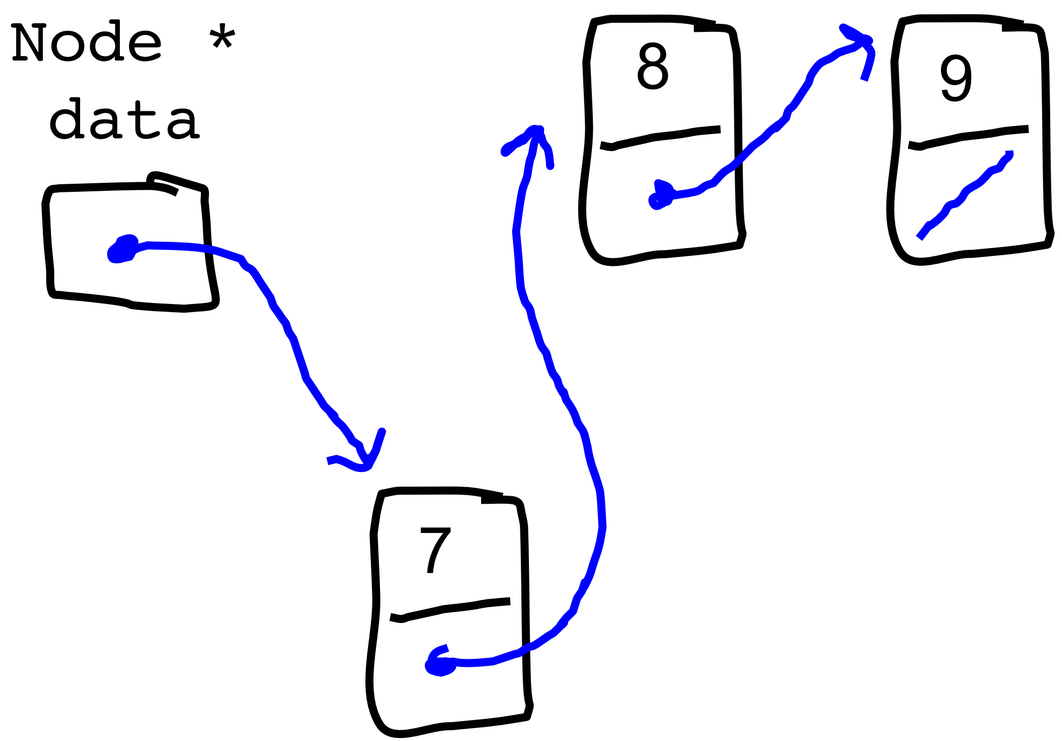
Big O of `pop()`? O(1)

Yay!

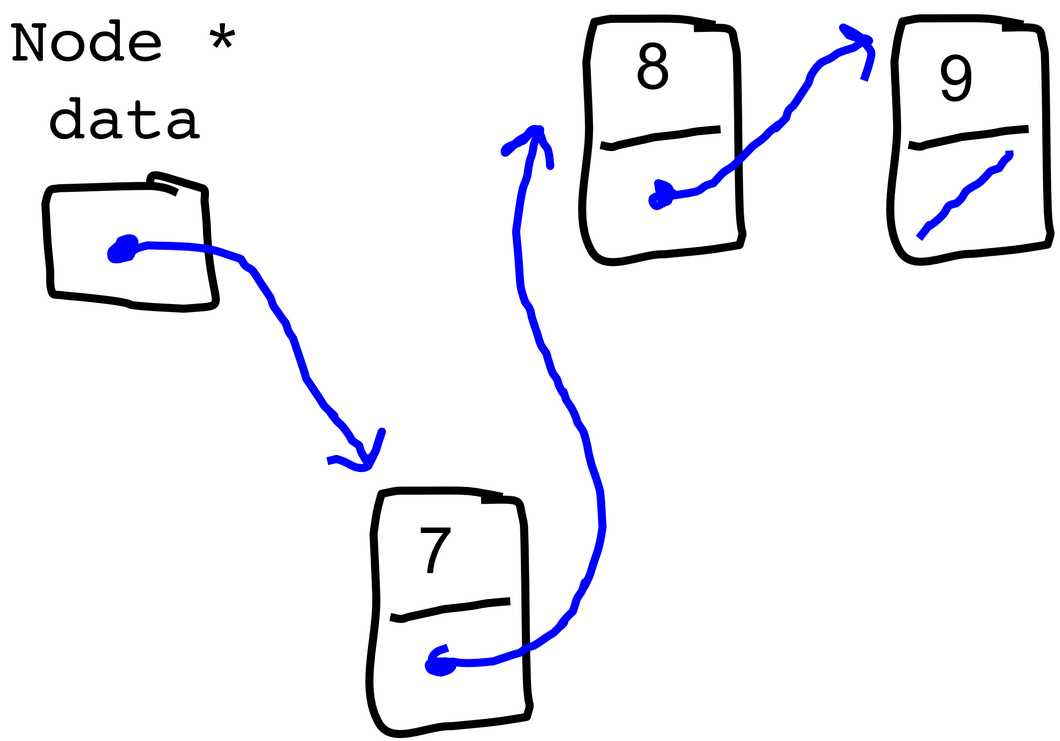
# Queue?



# Queue Enqueue?

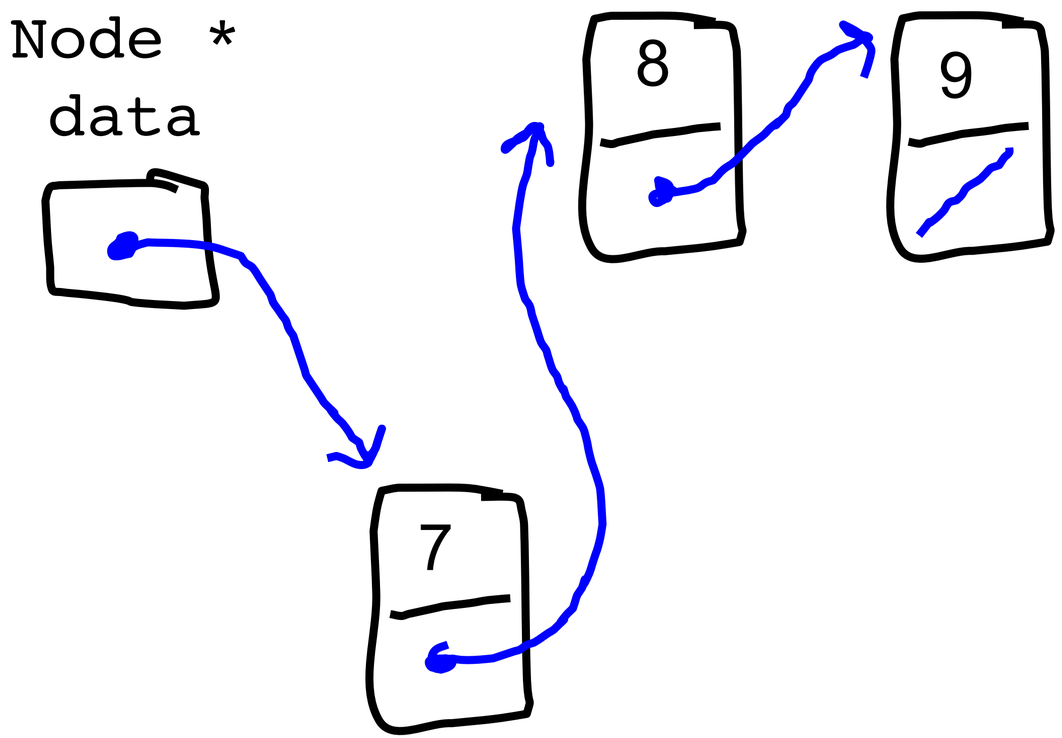


# Queue Enqueue?

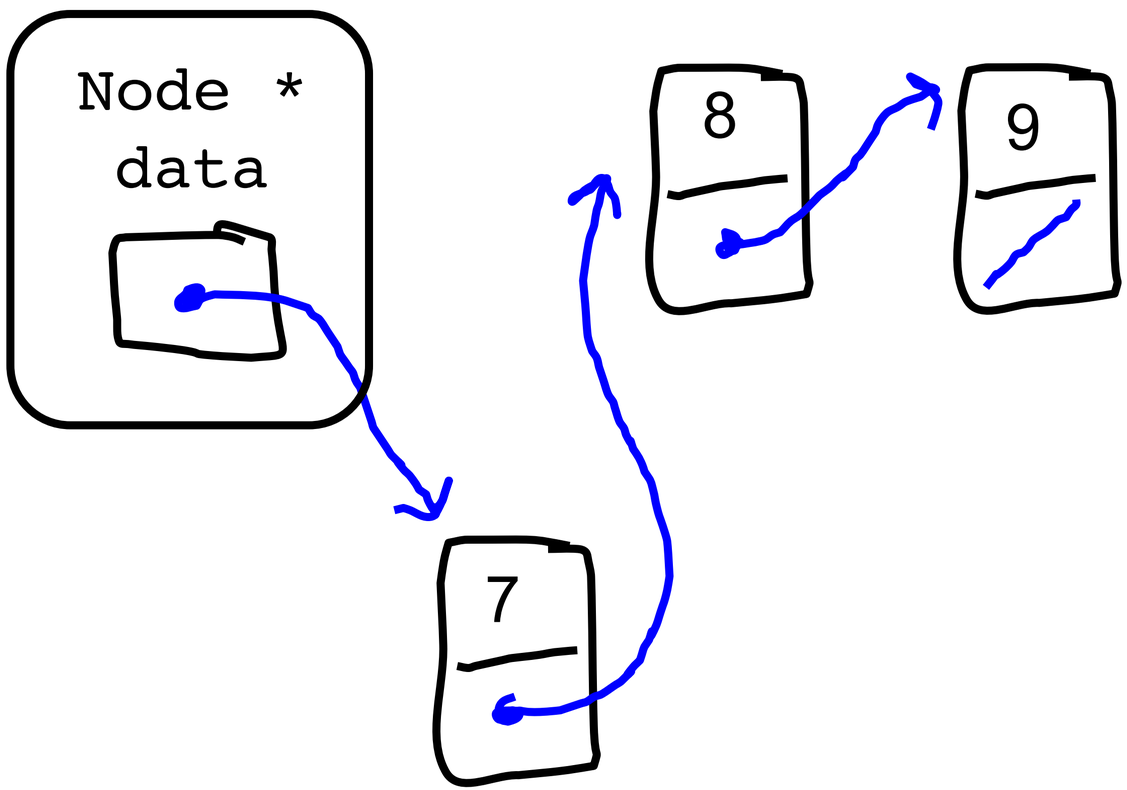



O(1)

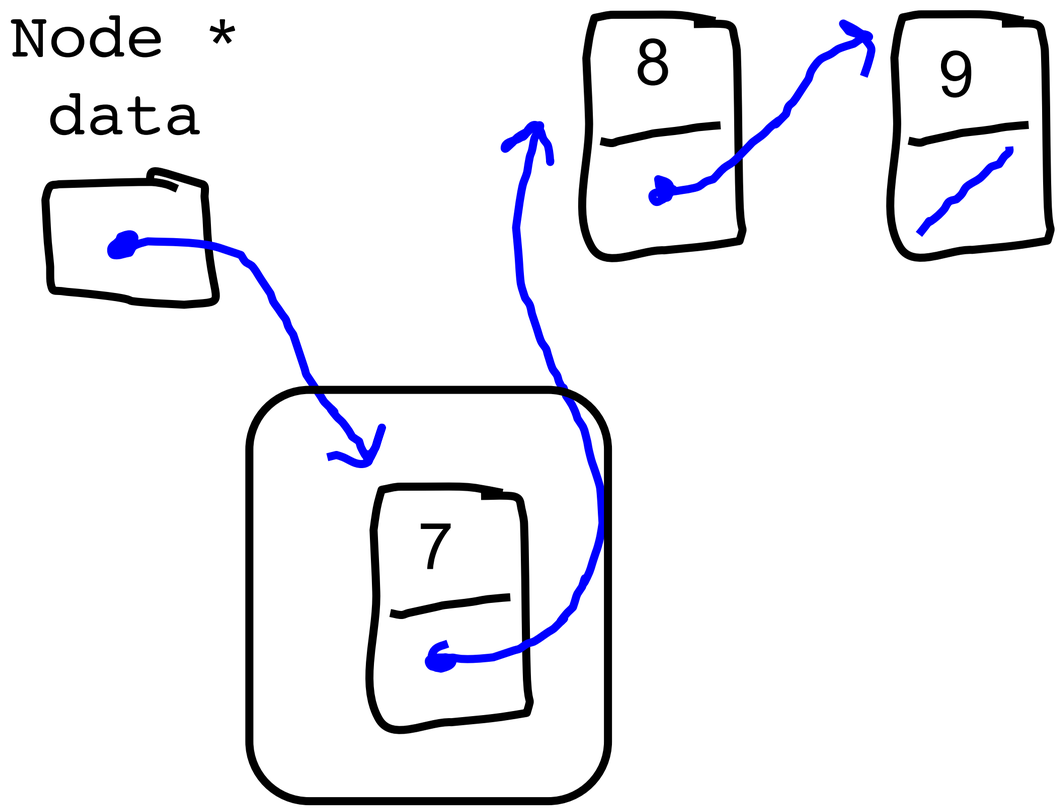
# Queue Dequeue?



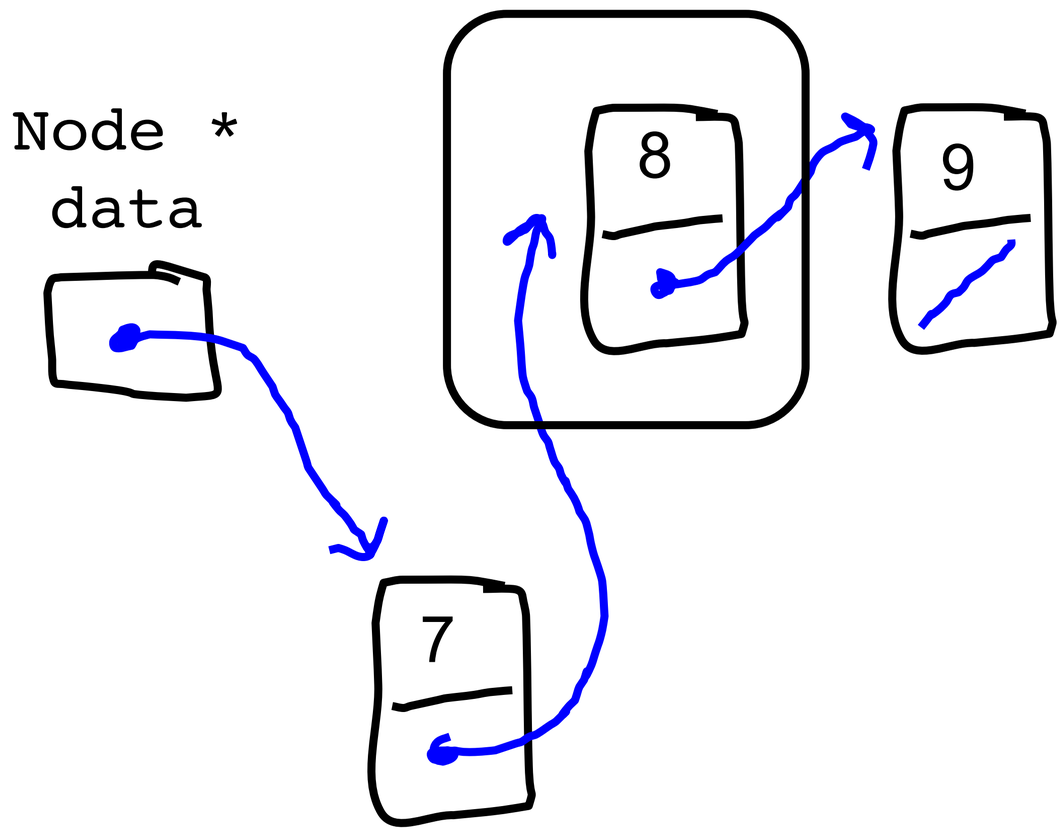
# Queue Dequeue?



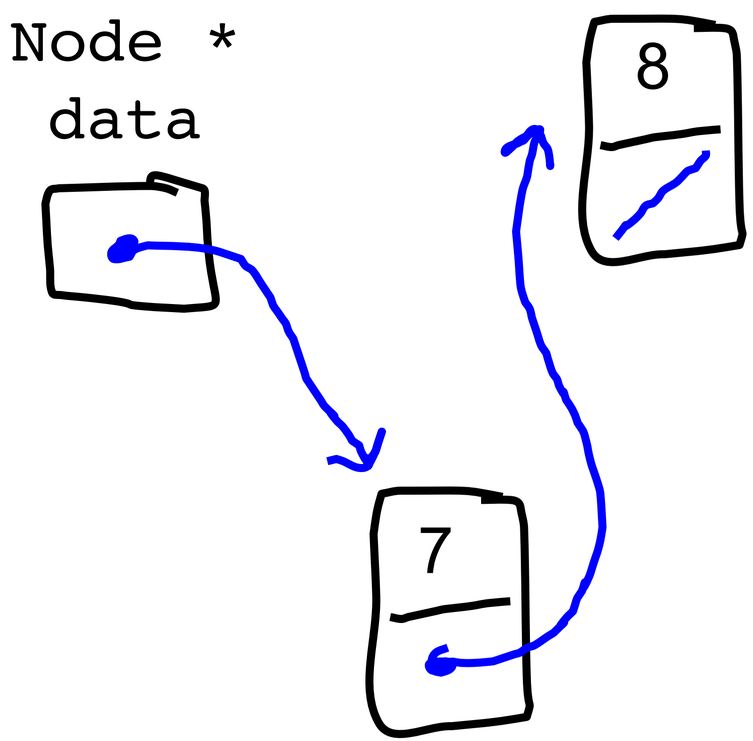
# Queue Dequeue?



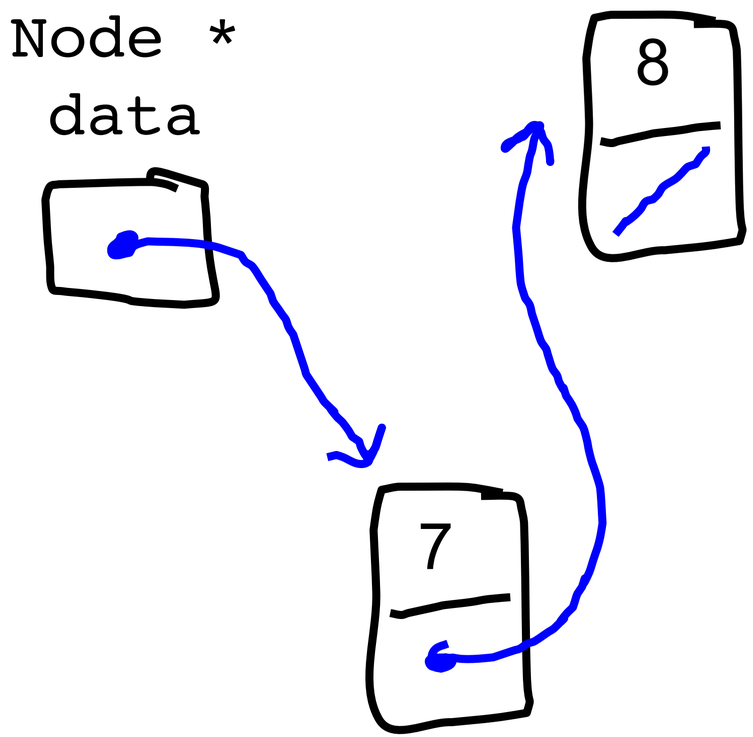
# Queue Dequeue?



# Queue Dequeue?



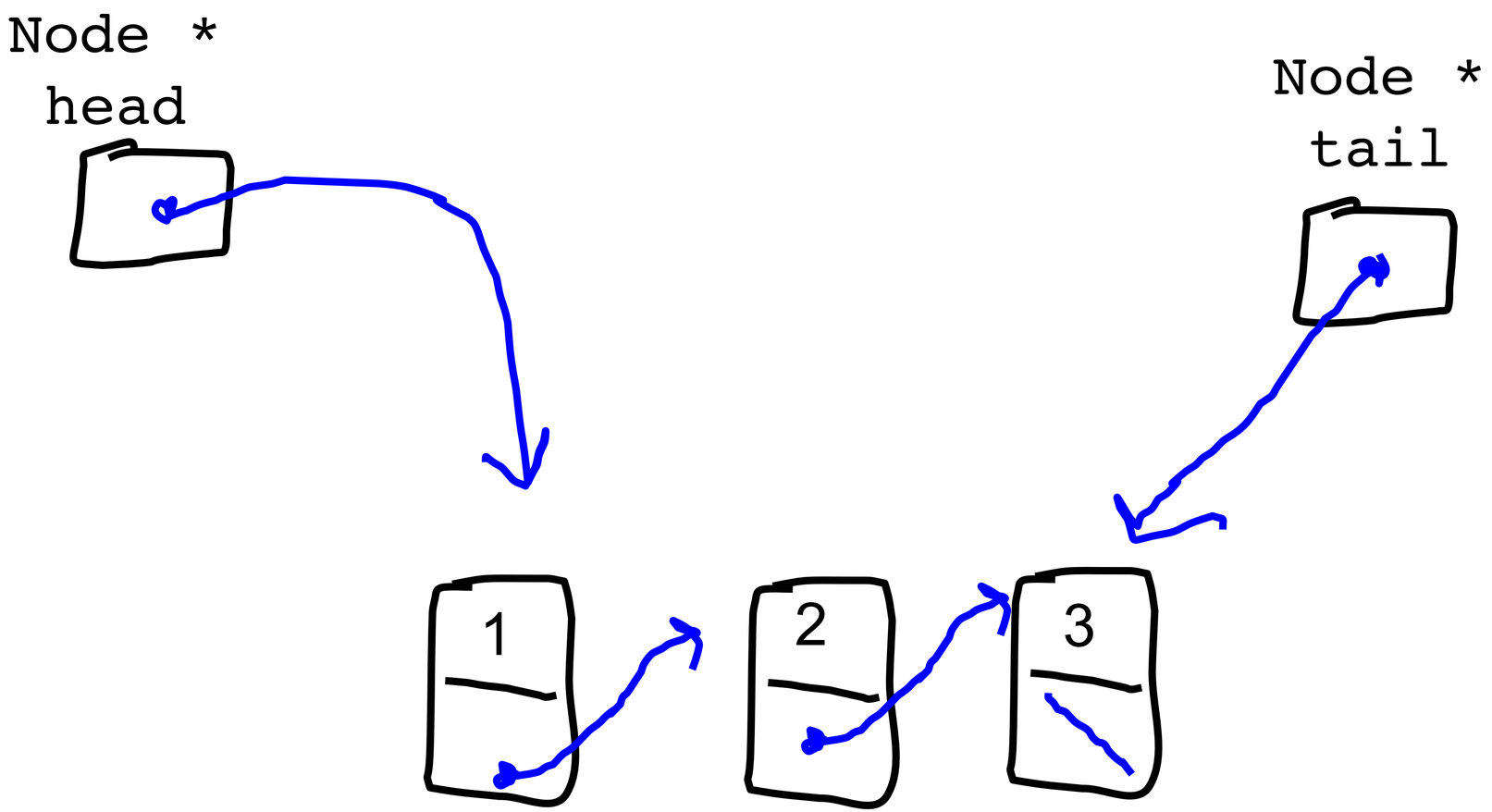
# Queue Dequeue?



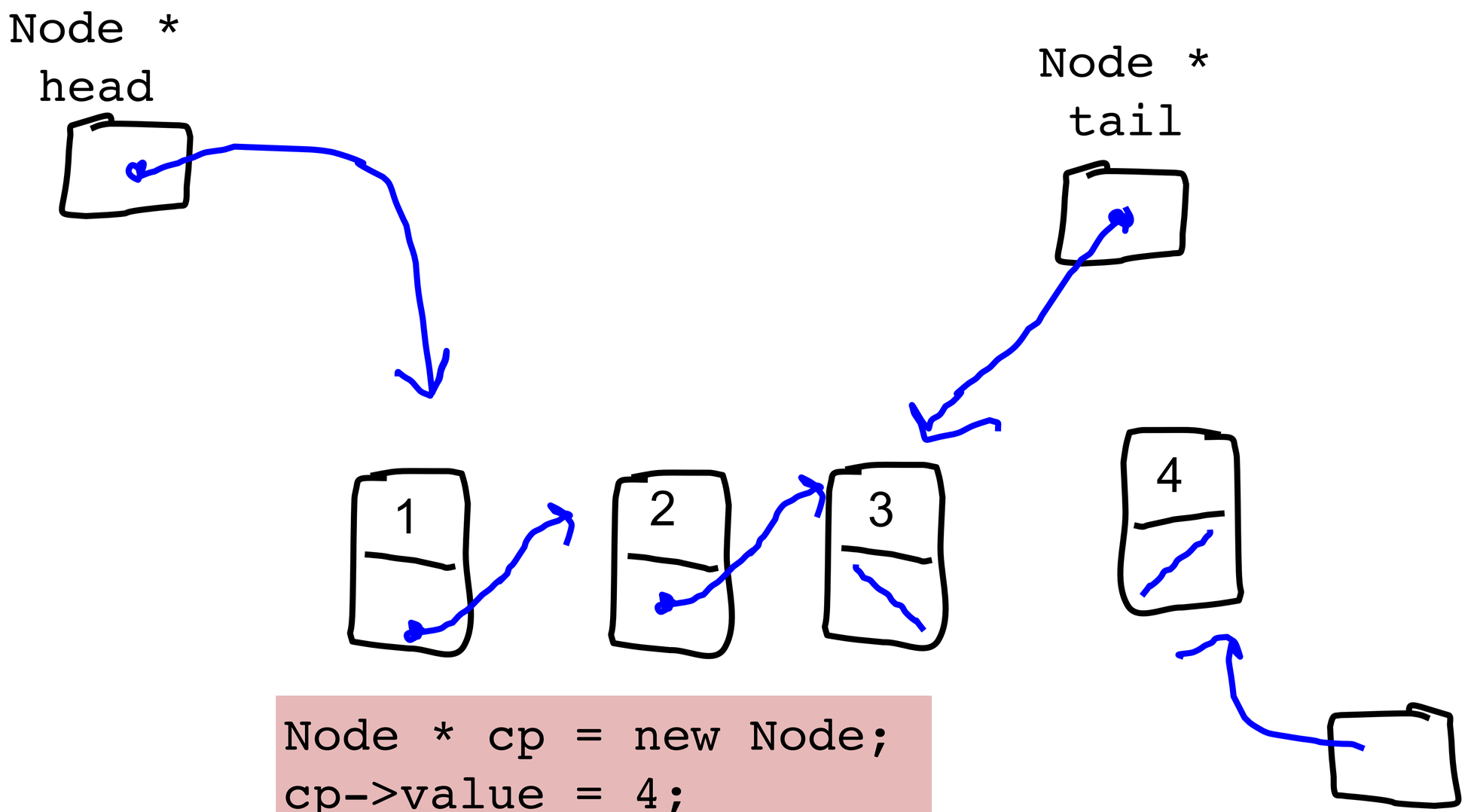

$$\mathcal{O}(n)$$

Always a Better Way

# Actual Queue: Enqueue

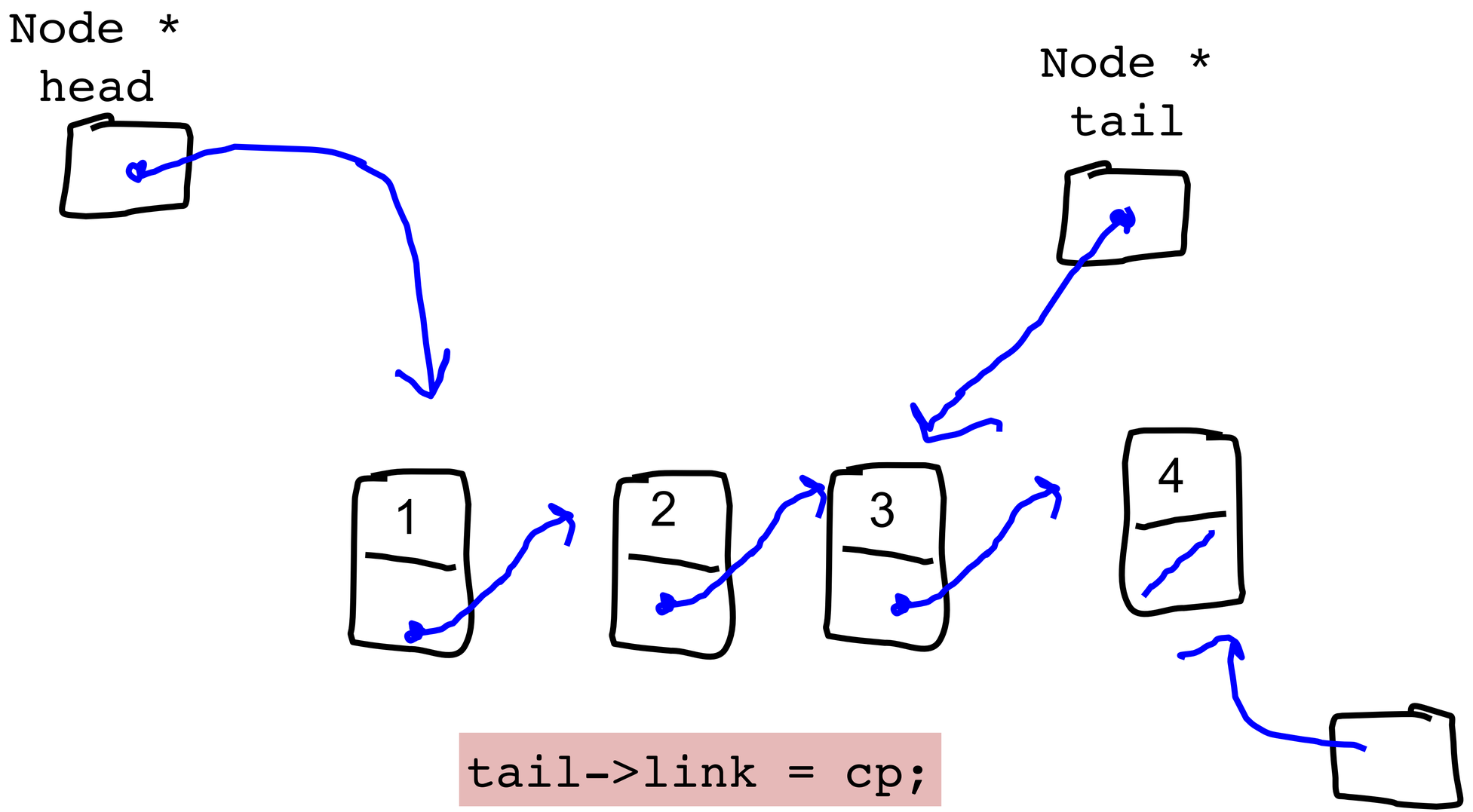


# Actual Queue: Enqueue



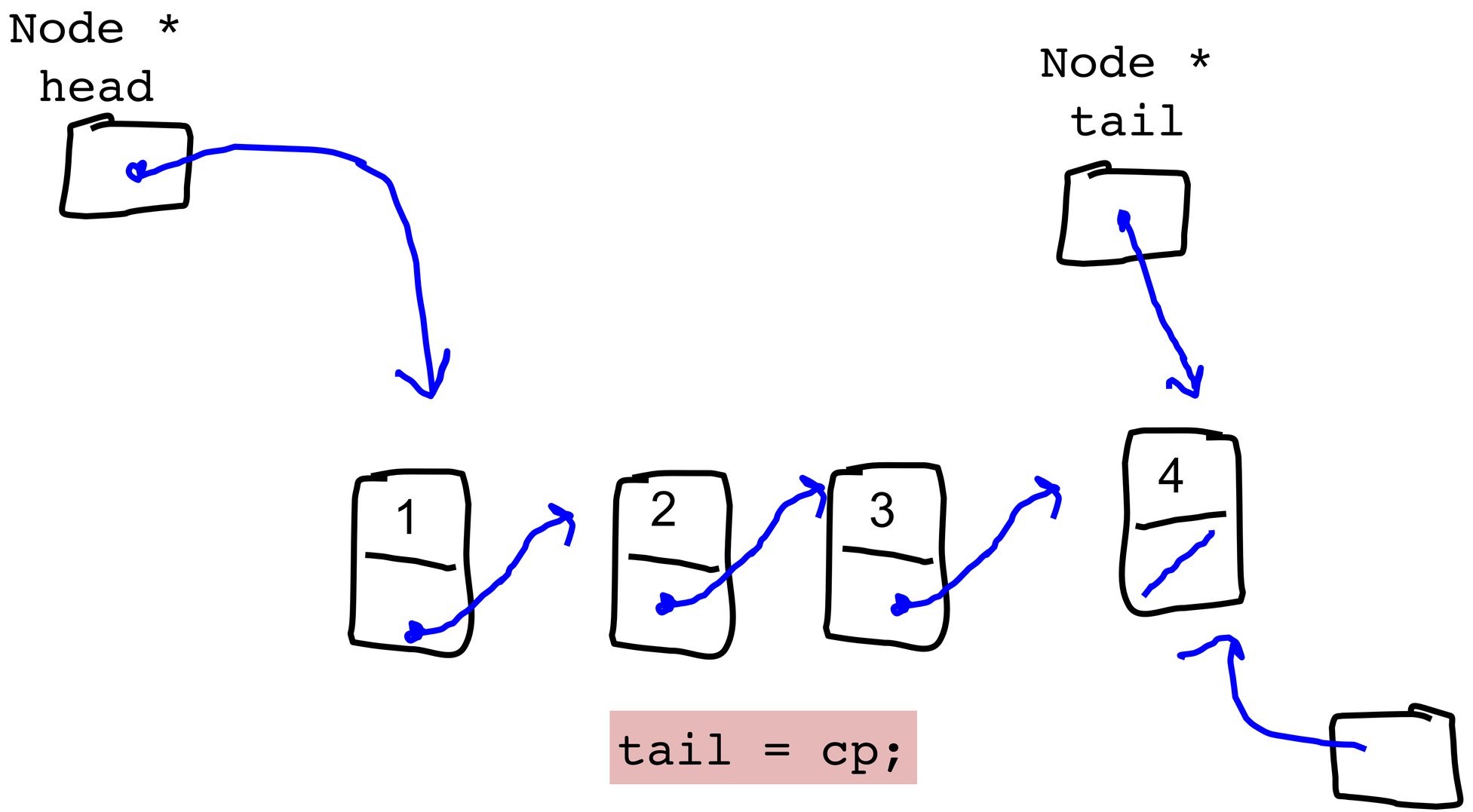

```
Node * cp = new Node;
cp->value = 4;
```

# Actual Queue: Enqueue



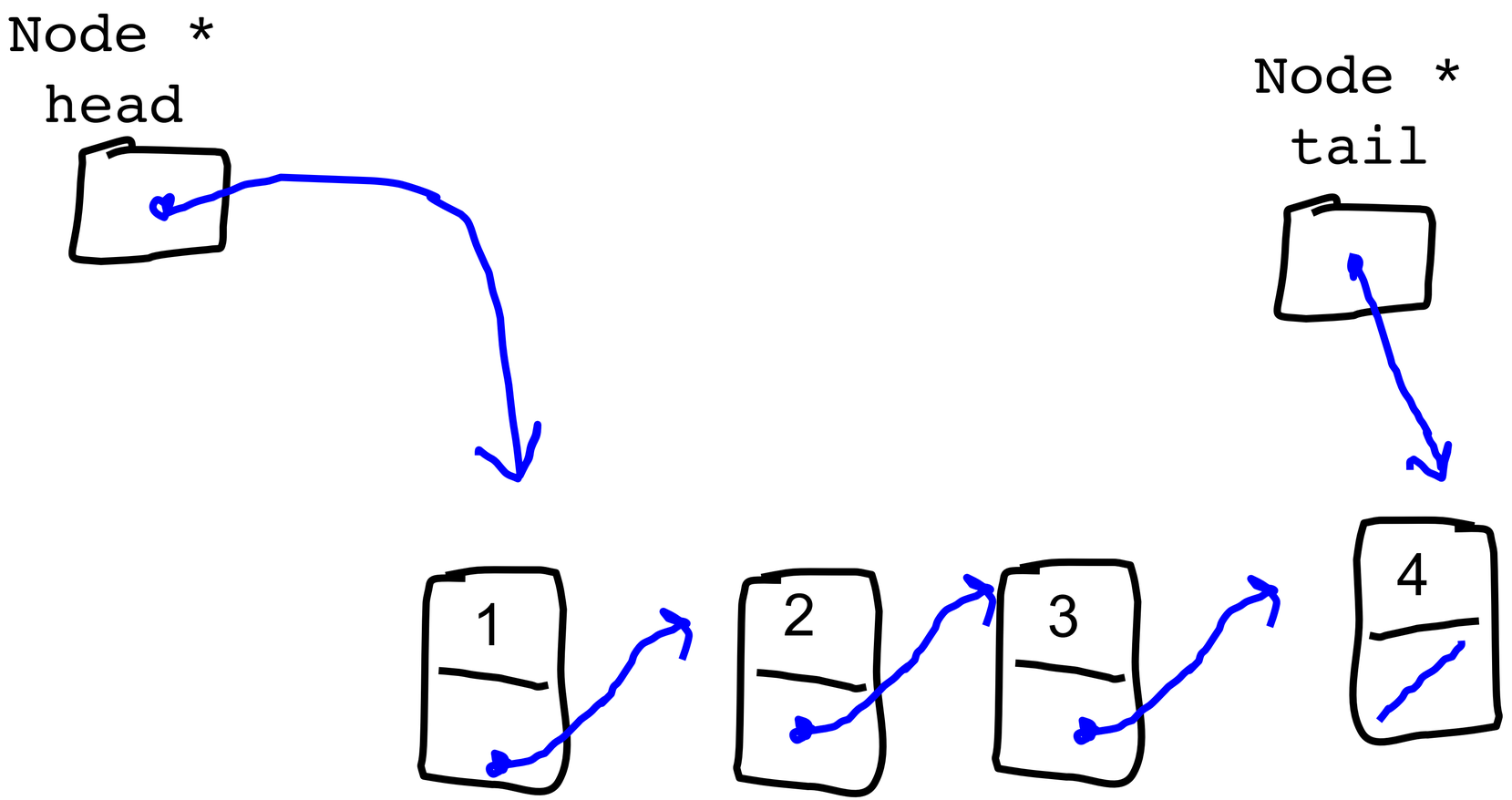

```
tail->link = cp;
```

# Actual Queue: Enqueue



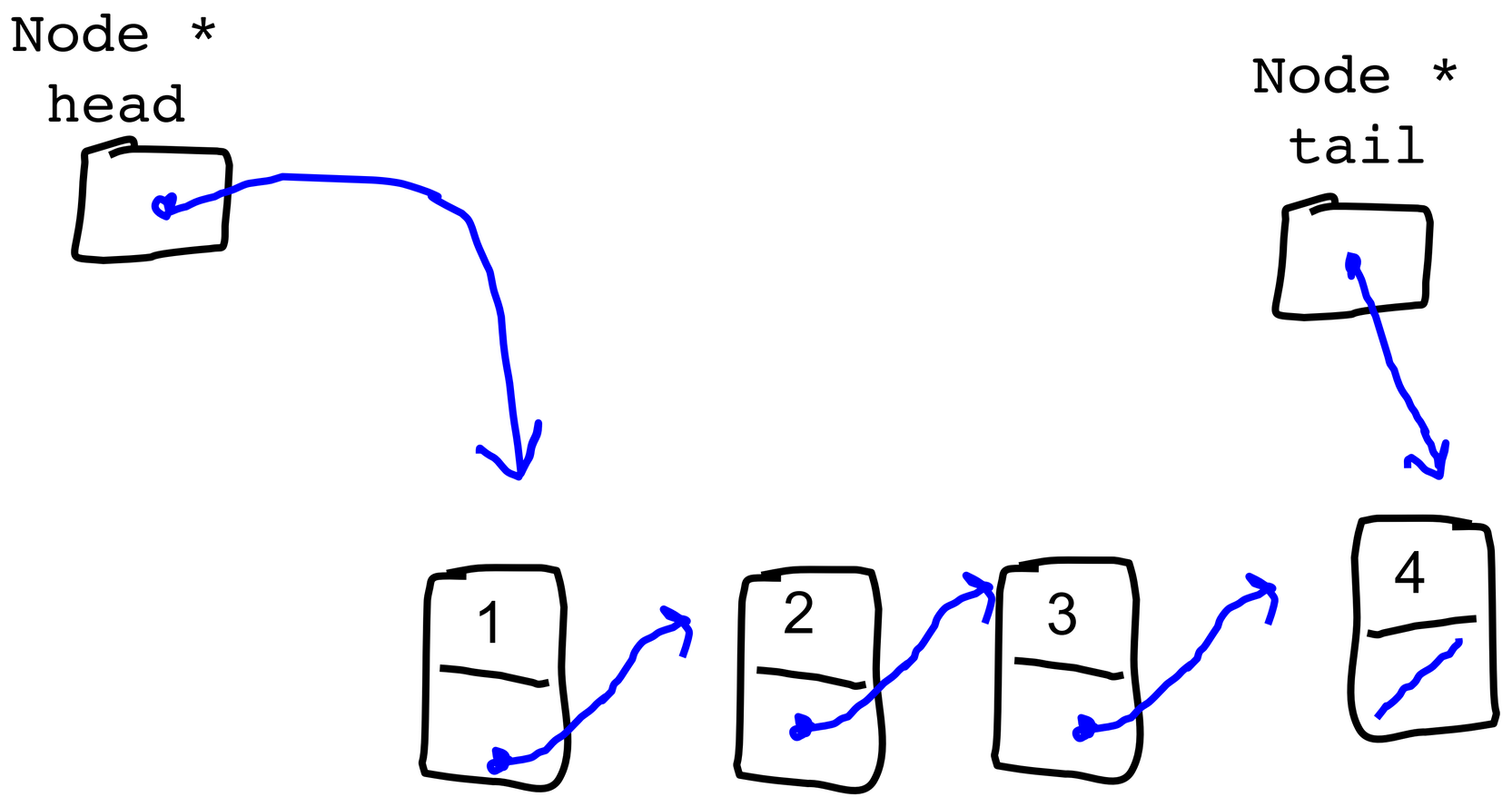

```
tail = cp;
```

# Actual Queue: Enqueue




```
return;
```

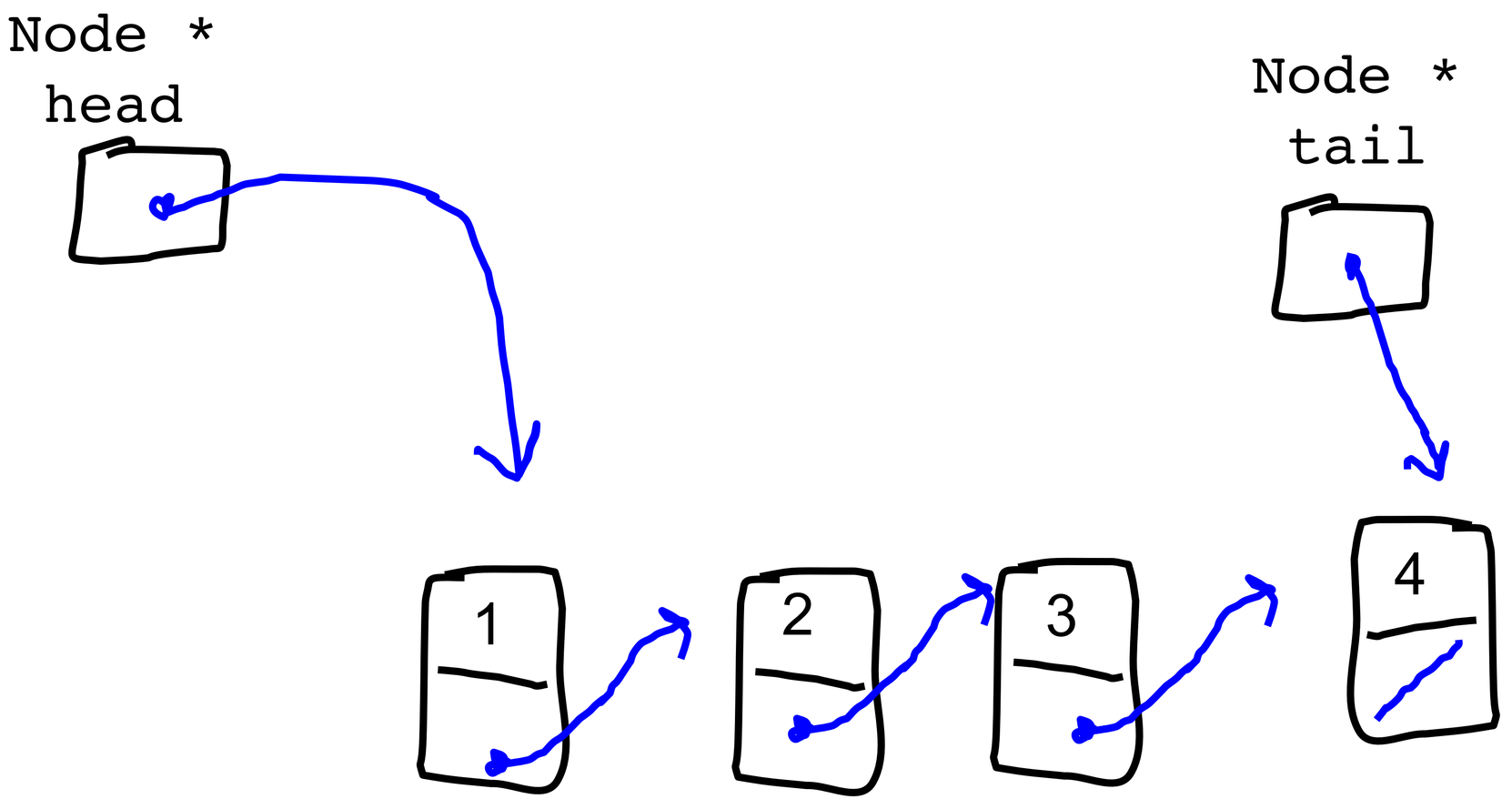
# Actual Queue: Enqueue



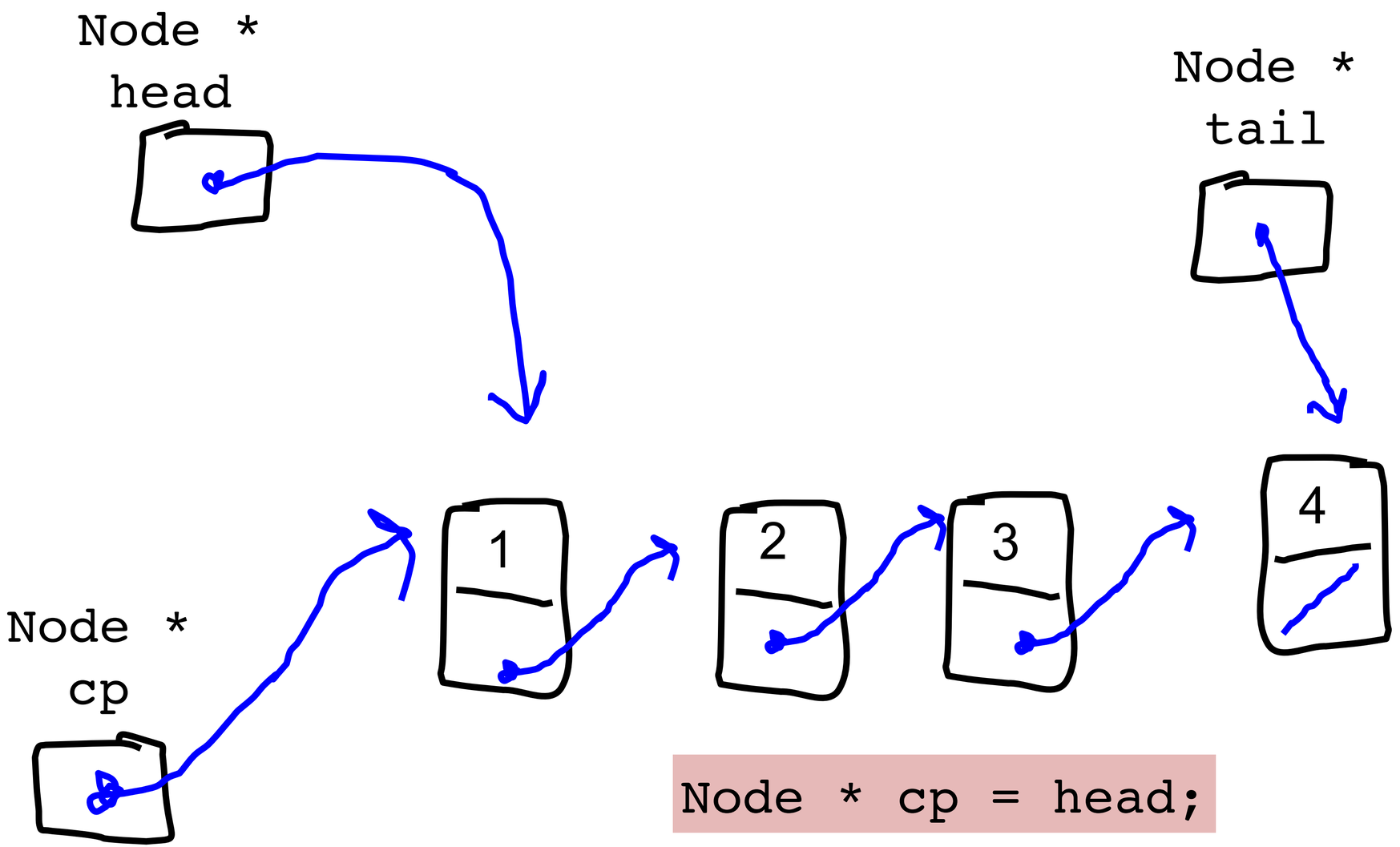

O(1)

# Dequeue

# Actual Queue: Dequeue

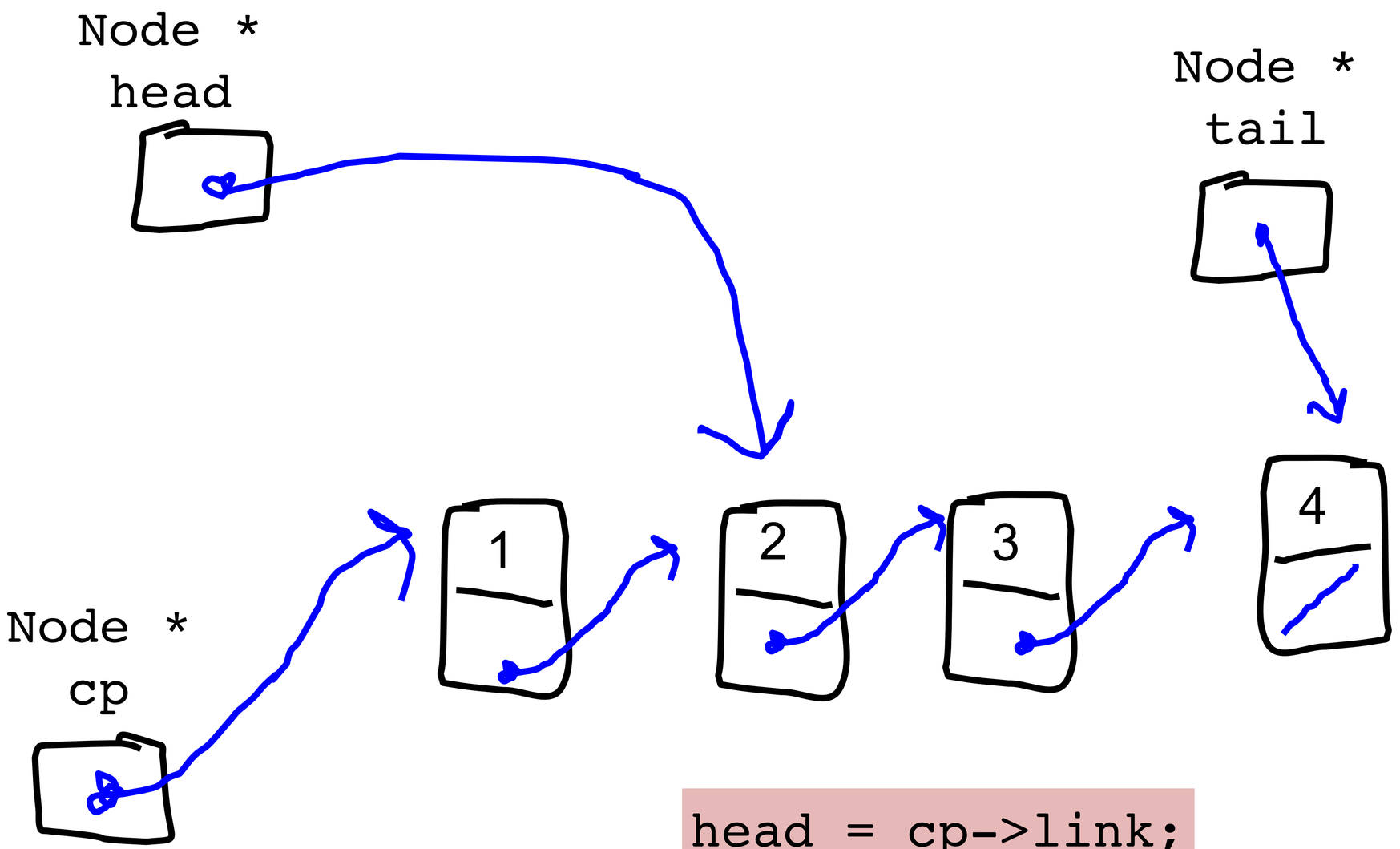


# Actual Queue: Dequeue



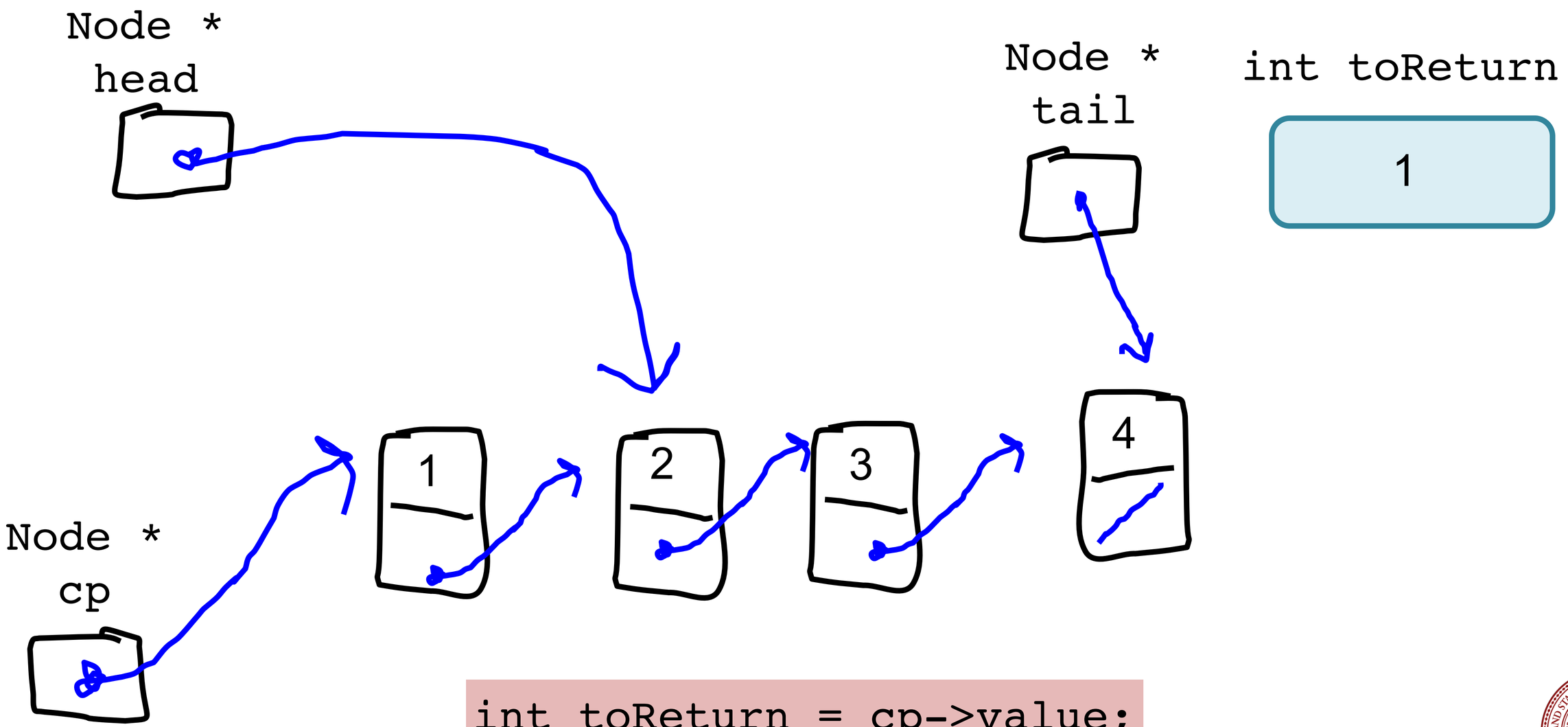

```
Node * cp = head;
```

# Actual Queue: Dequeue



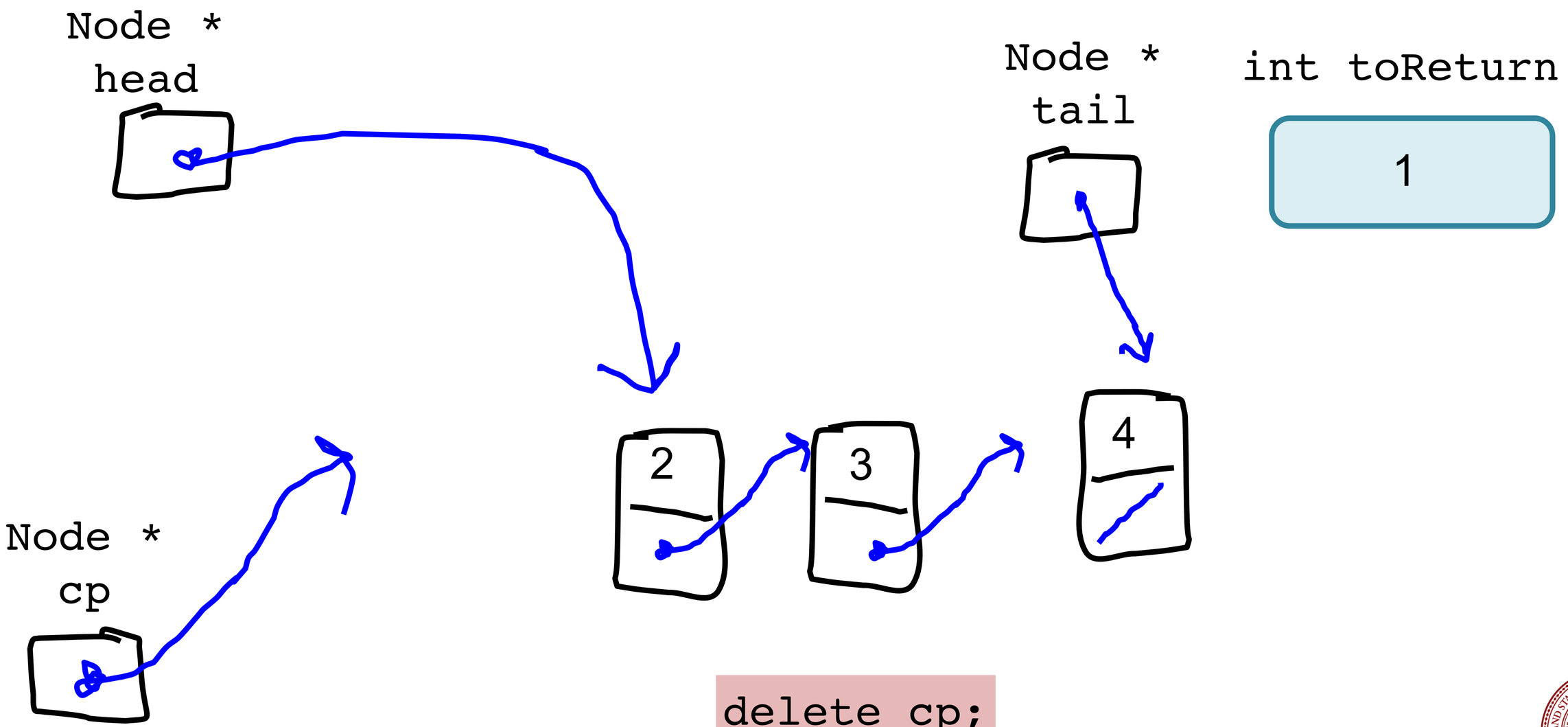

```
head = cp->link;
```

# Actual Queue: Dequeue

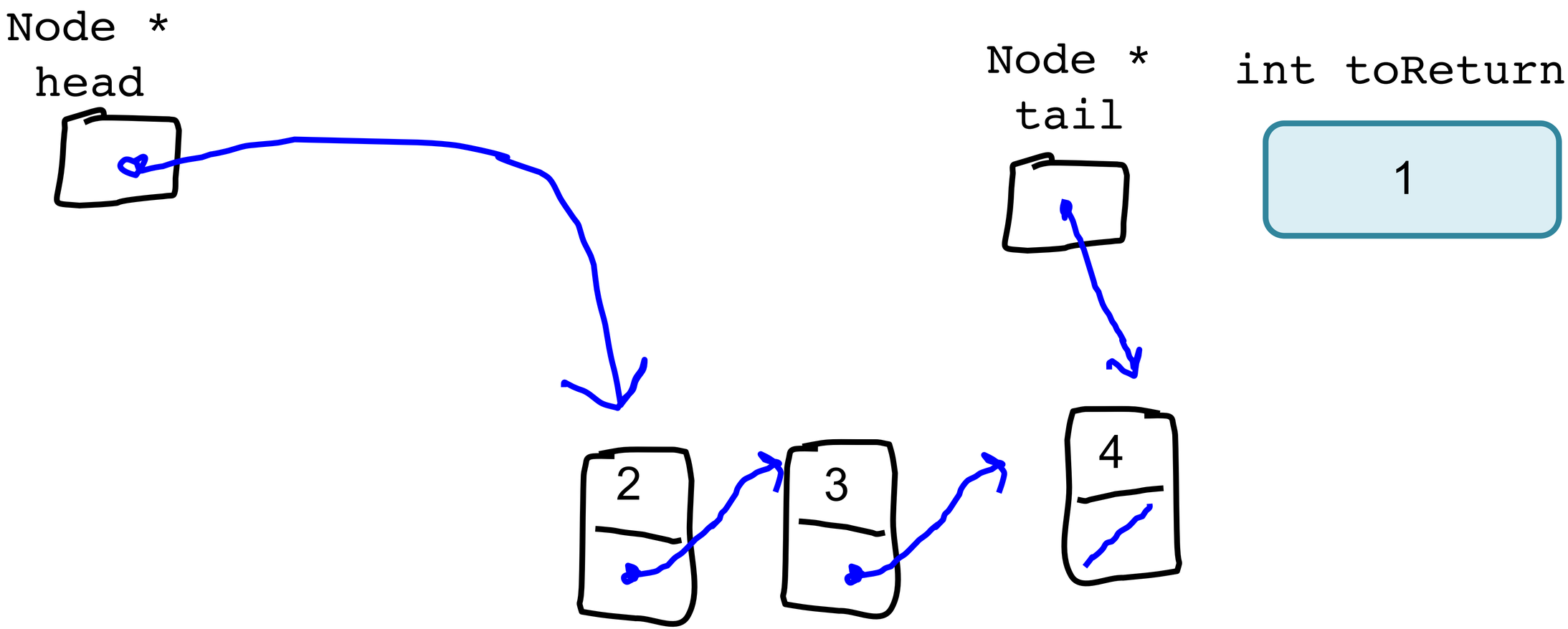



```
int toReturn = cp->value;
```

# Actual Queue: Dequeue

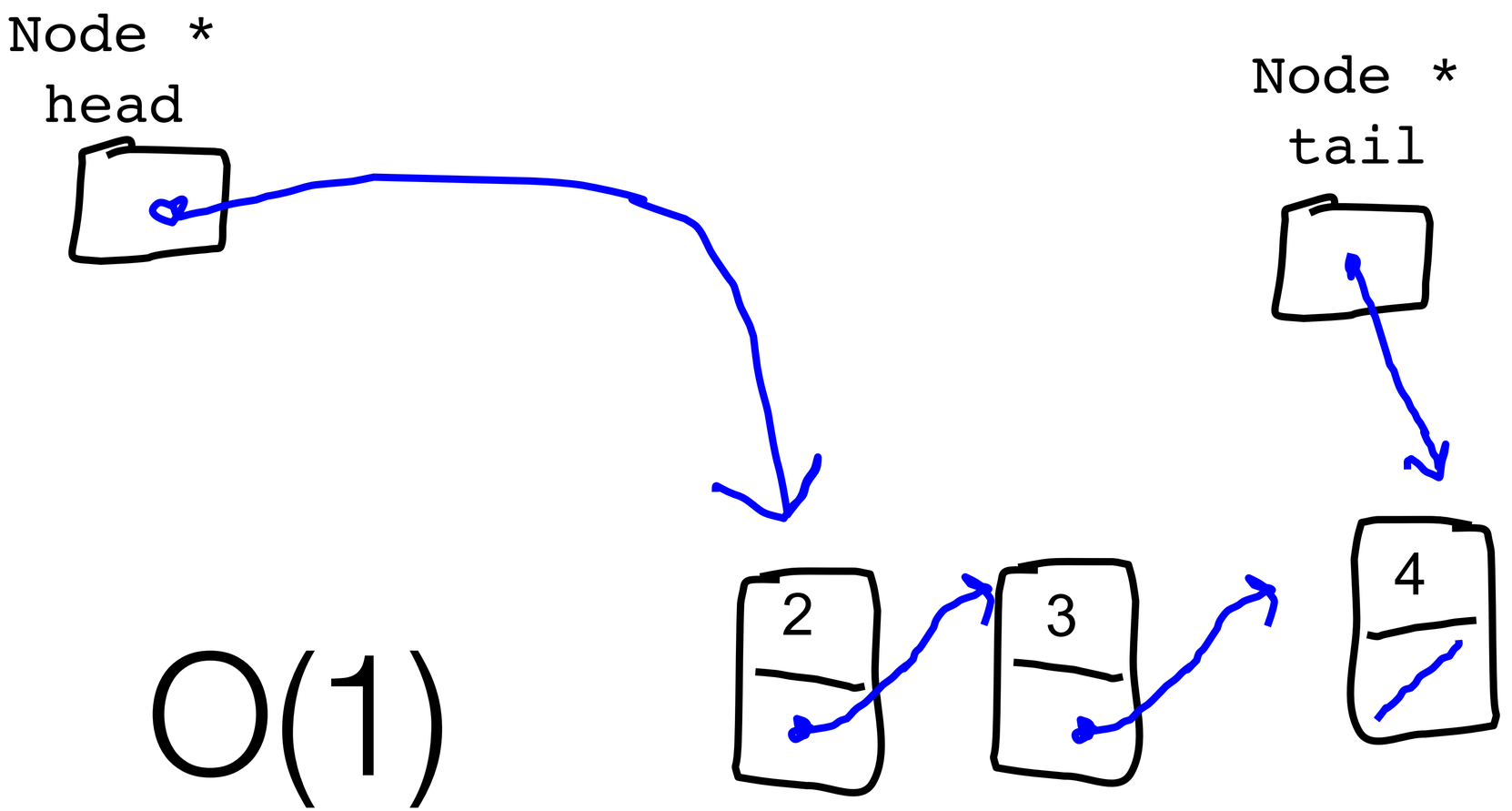


# Actual Queue: Dequeue




```
return toReturn;
```

# Actual Queue: Dequeue



# Queue

```
class QueueInt {                    // in QueueInt.h
public:
    QueueInt ();                    // constructor

    void enqueue(int value); // append a value
    int dequeue();                   // return the first-in value

private:
    struct Node {
      int value;
      Node * link;
    };
    Node * head;        // has a pointer to the first node
    Node * tail;        // and a pointer to the last node
};
```

# Queue Implementation

```
void QueueInt::enqueue(int v) {
    Node * temp = new Node;
    temp->value = v;
    tail->link = temp;
    tail = temp;
}

int QueueInt::dequeue() {
    int toReturn = head->value;
    Node * temp = head;
    head = temp->link;
    delete temp;
    return toReturn;
}
```

# Linked Lists are Excellent

| | Worst |
|---|---|
| Stack Push | $\mathcal{O}(1)$ |
| Stack Pop | $\mathcal{O}(1)$ |
| Queue Enqueue | $\mathcal{O}(1)$ |
| Queue Dequeue | $\mathcal{O}(1)$ |