

CS 106B

Lecture 16:

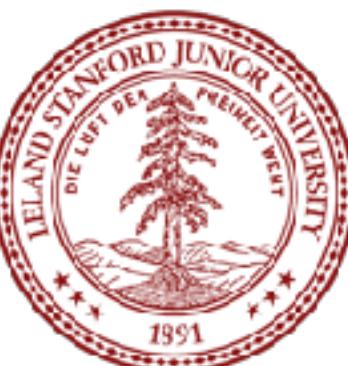
Linked Lists

Monday, April 8, 2017

Programming Abstractions
Spring 2017
Stanford University
Computer Science Department

Lecturer: Chris Gregg

reading:
Programming Abstractions in C++, Chapter 11

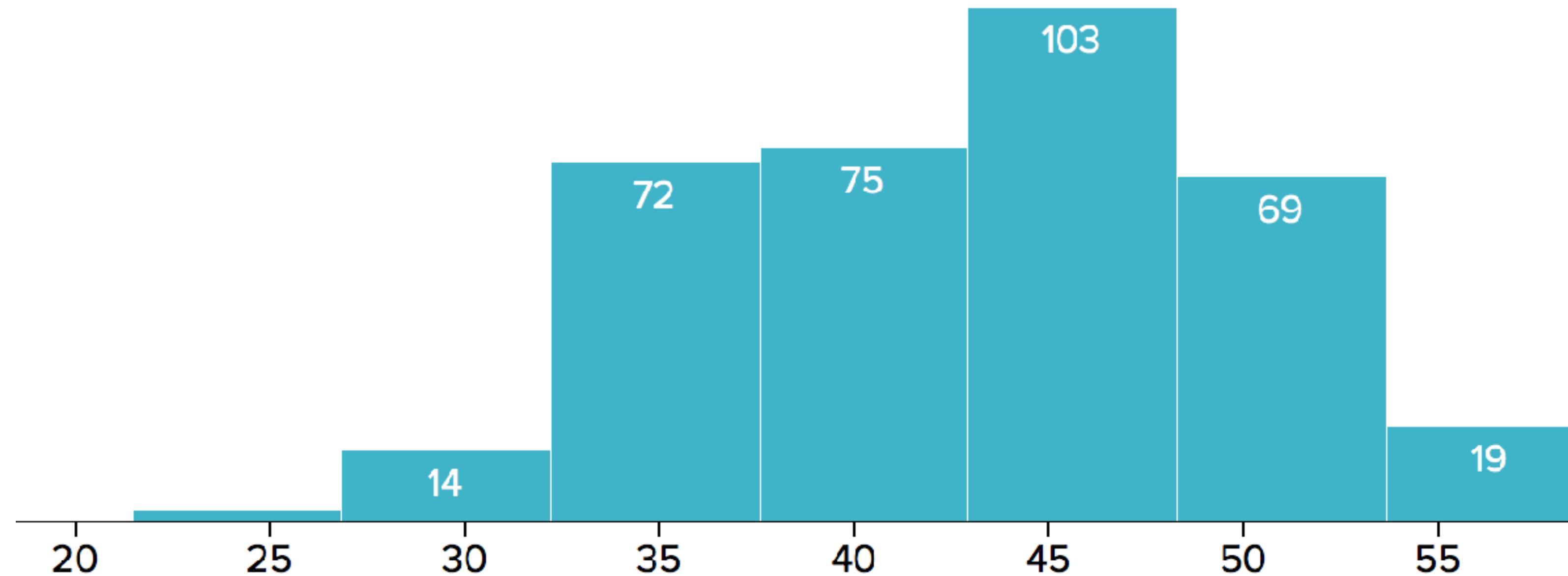


Today's Topics

- Logistics
 - Tiny feedback:
 - favorite type of dog? Tough choice...probably Corgis. Or Labs.
 - favorite programming language? Tough choice...depends on the situation!
 - C++: fast, robust, powerful
 - C: less to worry about than C++
 - Python: so fast to code in, but can be slow
 - Javascript: need something on the web?
 - Haskell: so different from regular programming
 - Piet: ???
- 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



Midterm



MEDIAN

43.0

MAXIMUM

59.0

MEAN

42.8

STD DEV

7.28

MEDIAN

72.88%

MAXIMUM

100.0%

MEAN

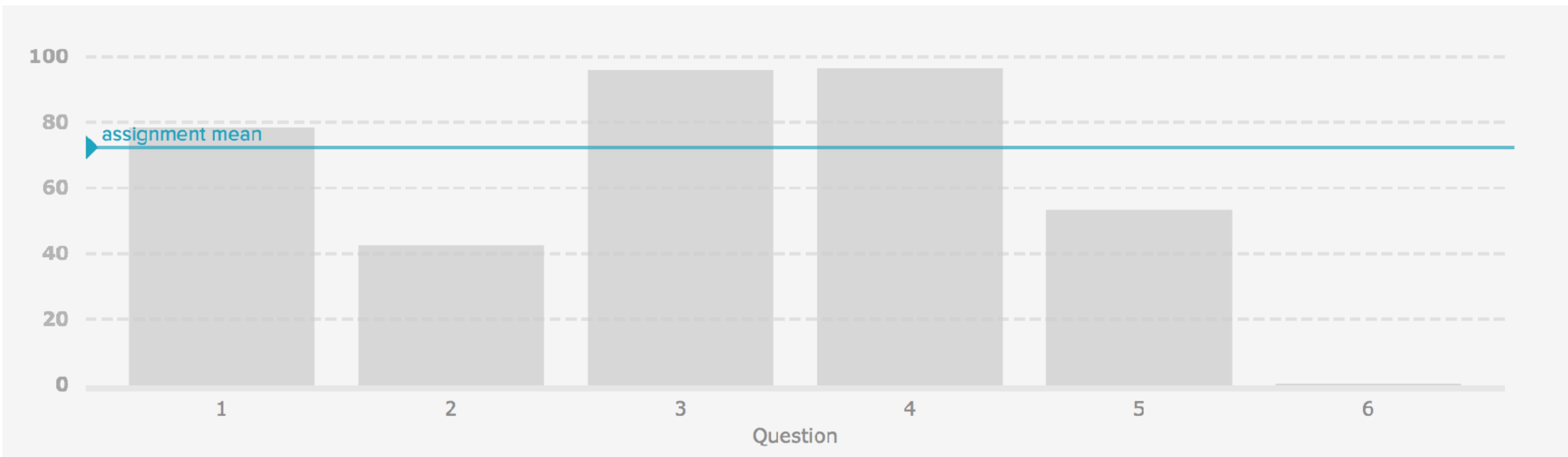
72.55%

STD DEV

12.33%



Midterm



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

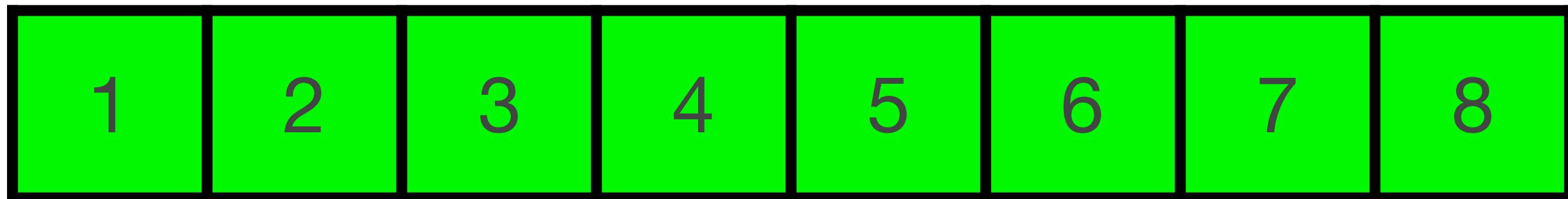
private:
    Vector<int> data;     // member variables
};
```



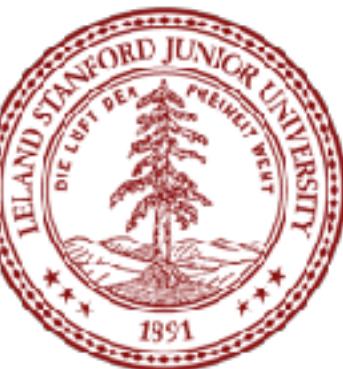
You're next!

back

front

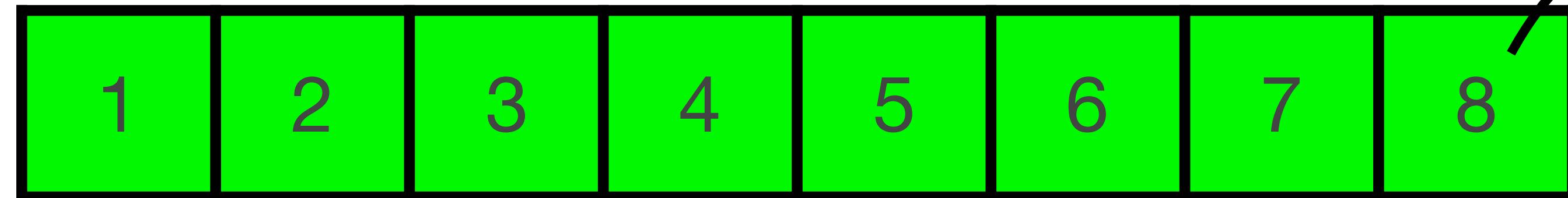


dequeue()

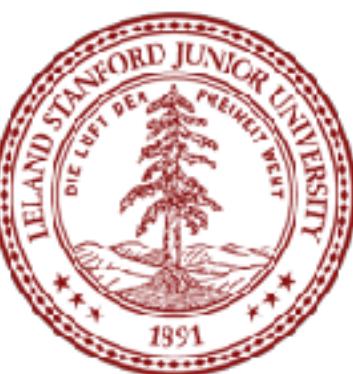


You're next!

back

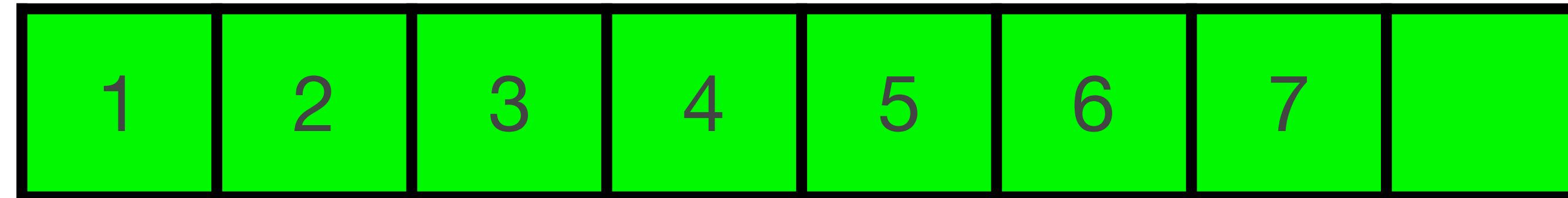


dequeue()

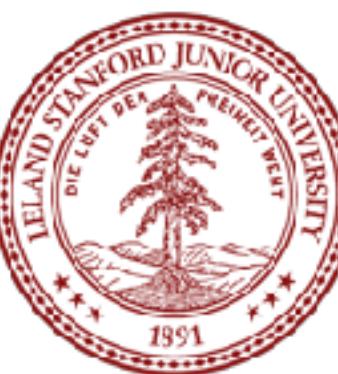


Excuse Me, Coming Through

back



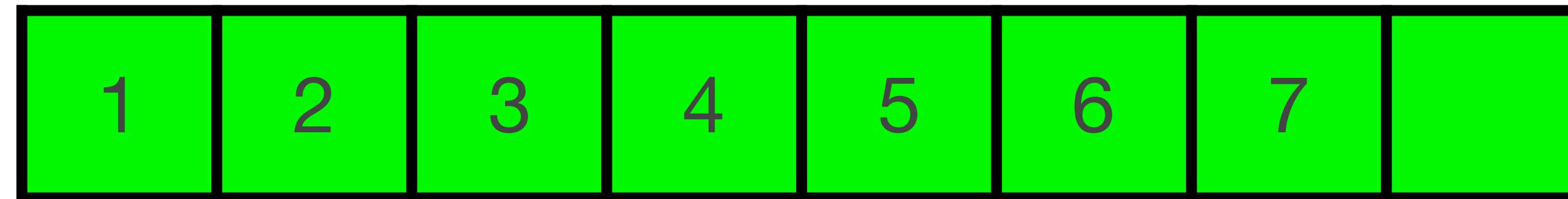
front



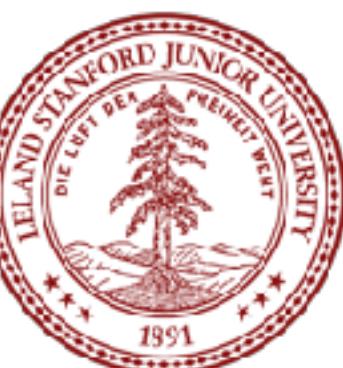
Excuse Me, Coming Through

back

front



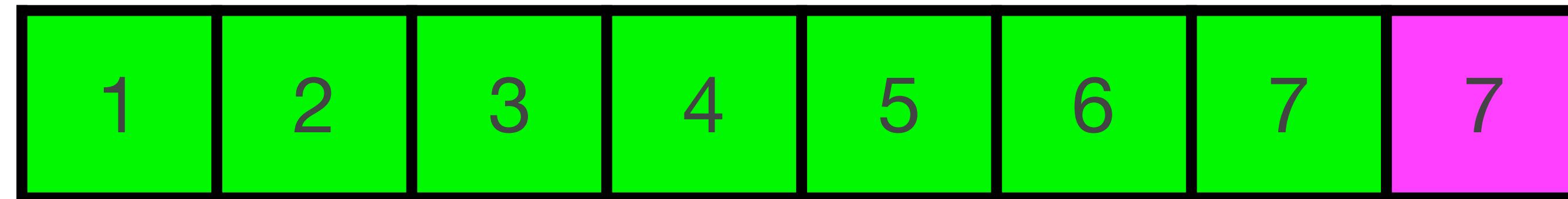
enqueue(42)



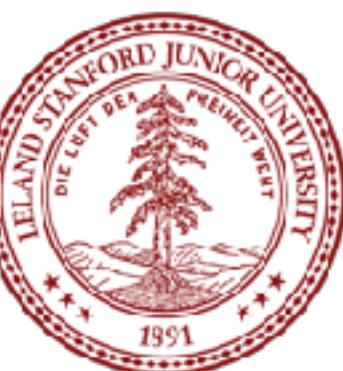
Excuse Me, Coming Through

back

front



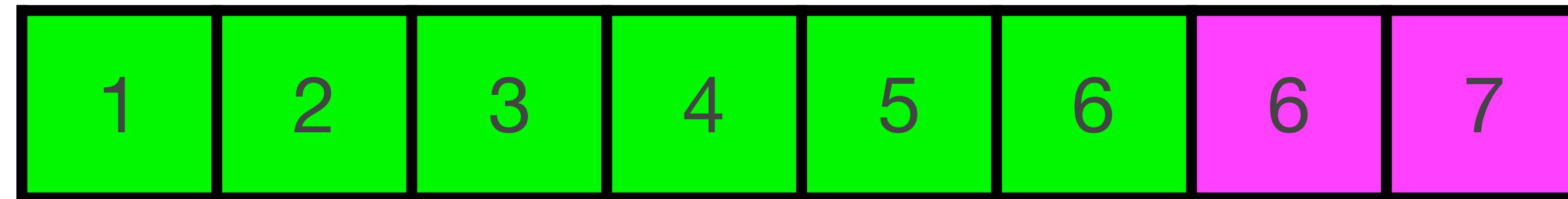
enqueue(42)



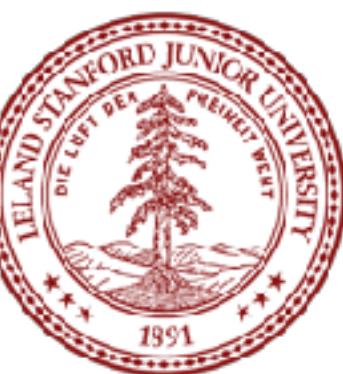
Excuse Me, Coming Through

back

front



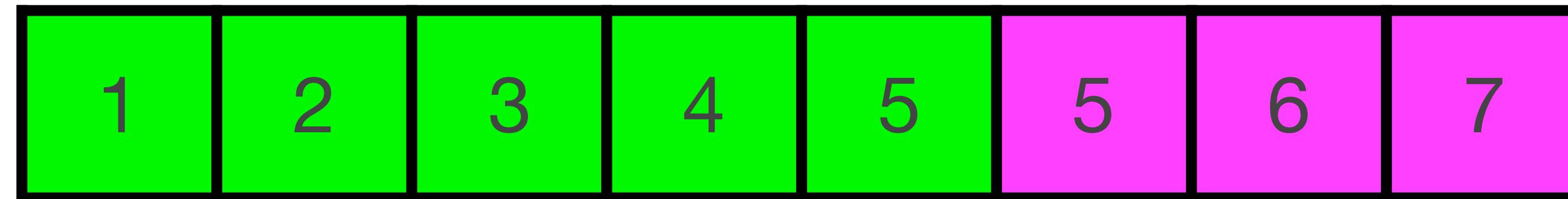
enqueue(42)



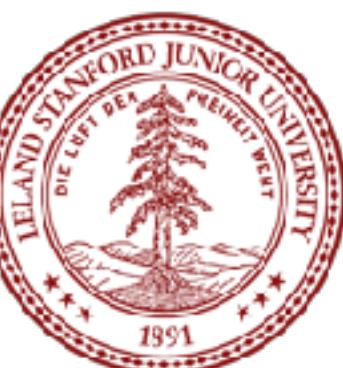
Excuse Me, Coming Through

back

front



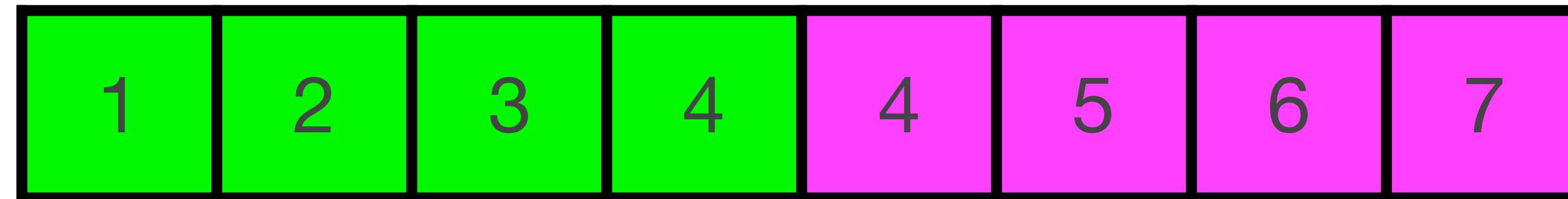
enqueue(42)



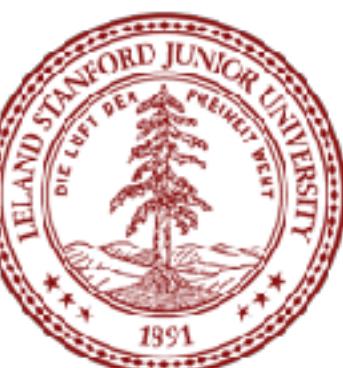
Excuse Me, Coming Through

back

front



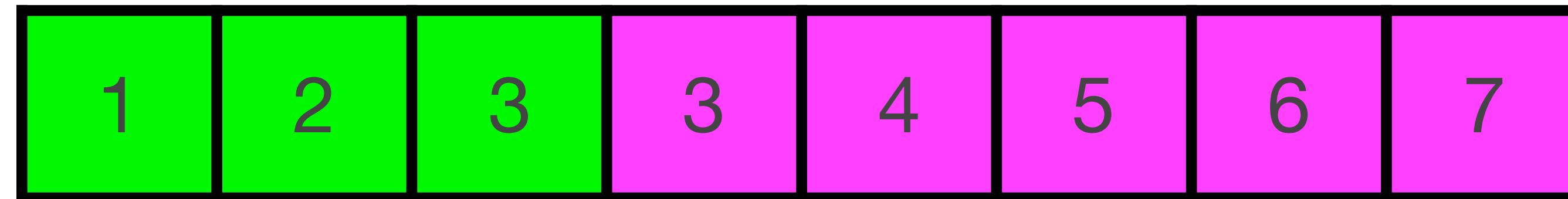
enqueue(42)



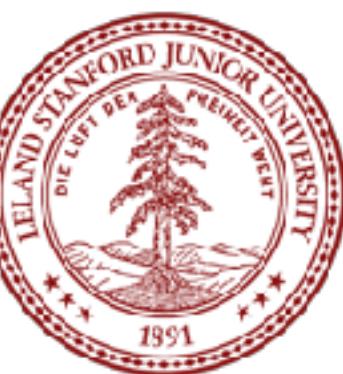
Excuse Me, Coming Through

back

front



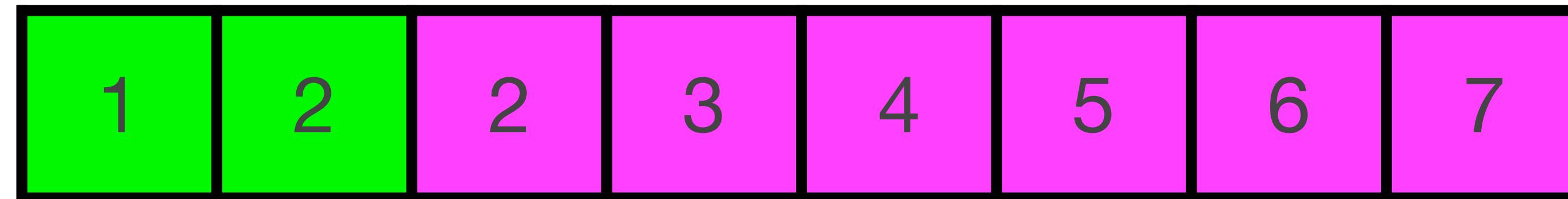
enqueue(42)



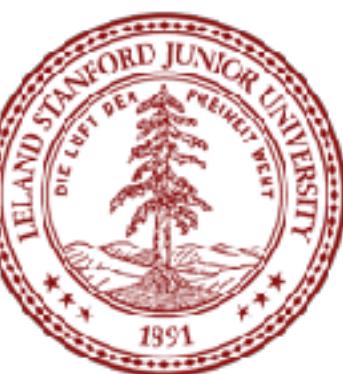
Excuse Me, Coming Through

back

front



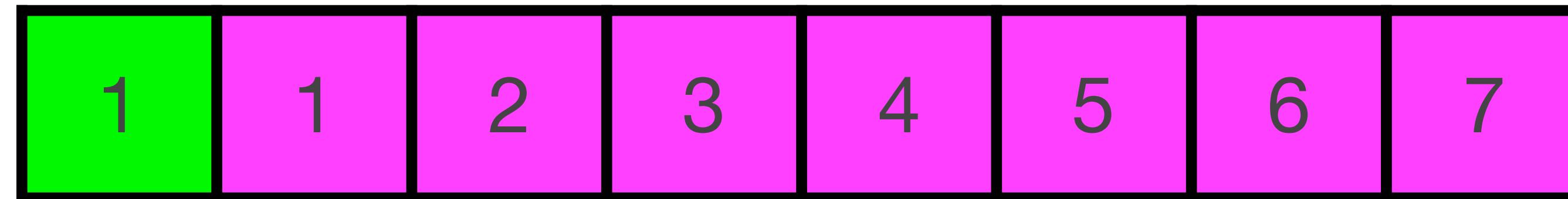
enqueue(42)



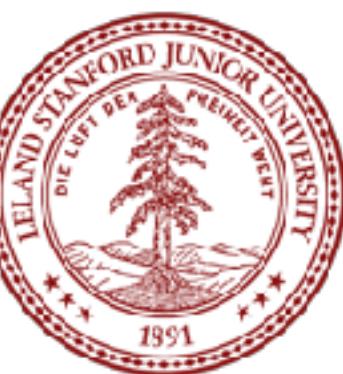
Excuse Me, Coming Through

back

front



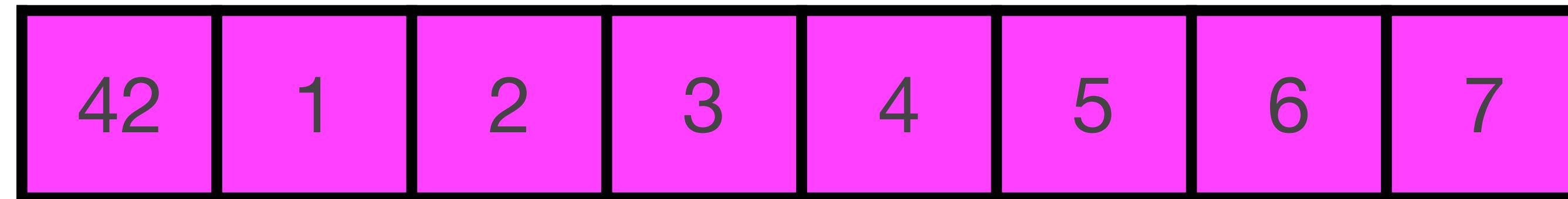
enqueue(42)



Excuse Me, Coming Through

back

front



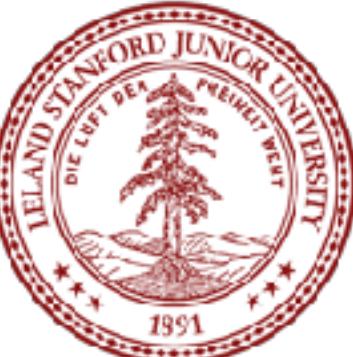
enqueue(42)



Queue as Vector: Big O

Enqueue: $O(n)$

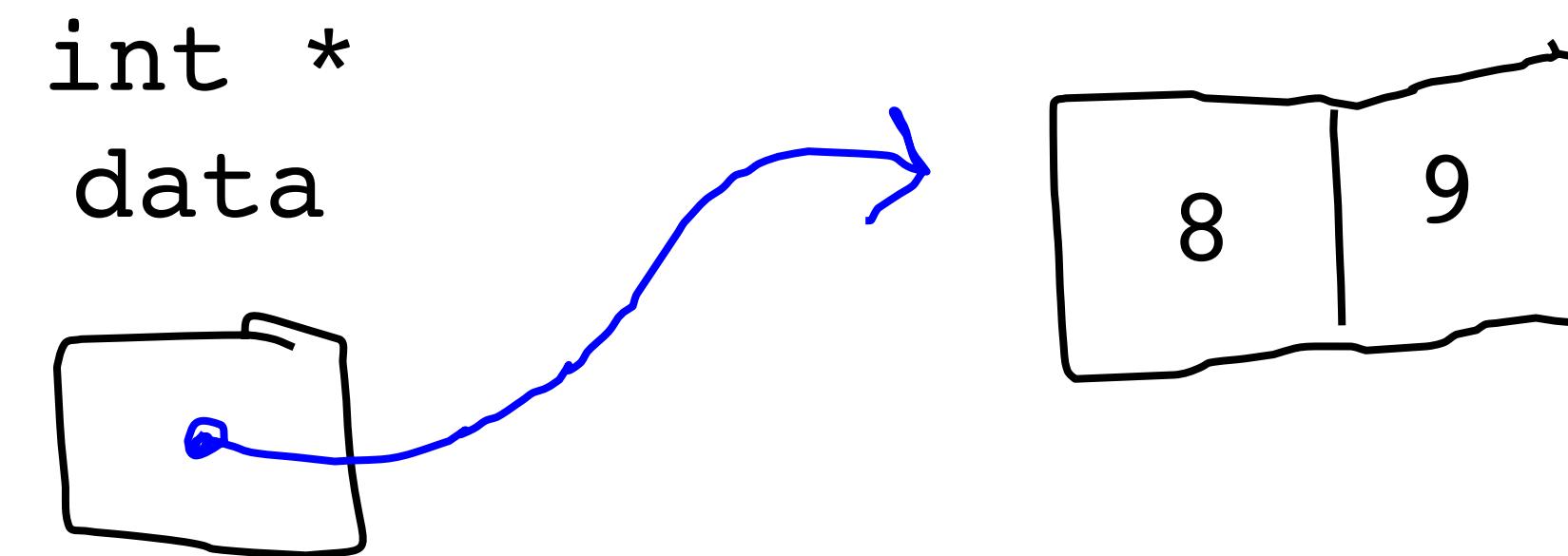
Dequeue: $O(1)$



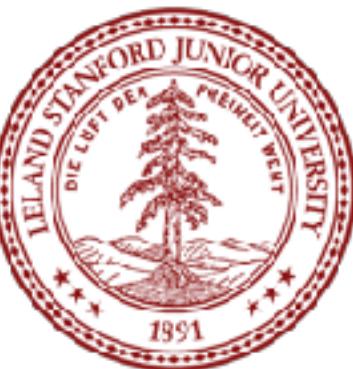
**WE CAN DO
BETTER**



And Now for Something Completely Different

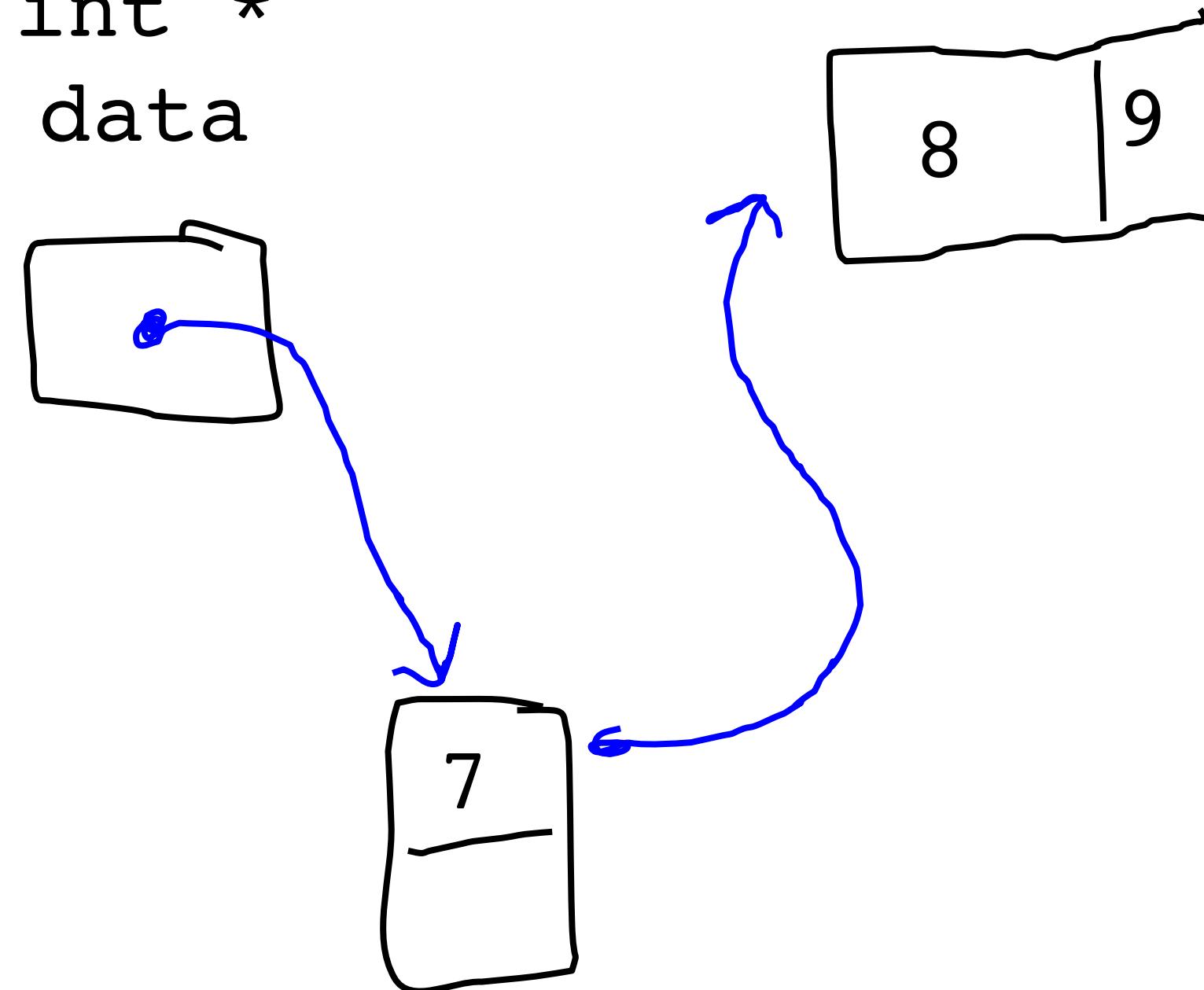


enqueue(7)

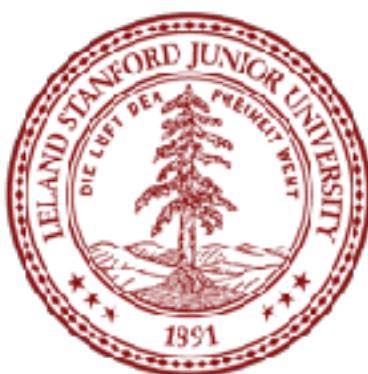


And Now for Something Completely Different

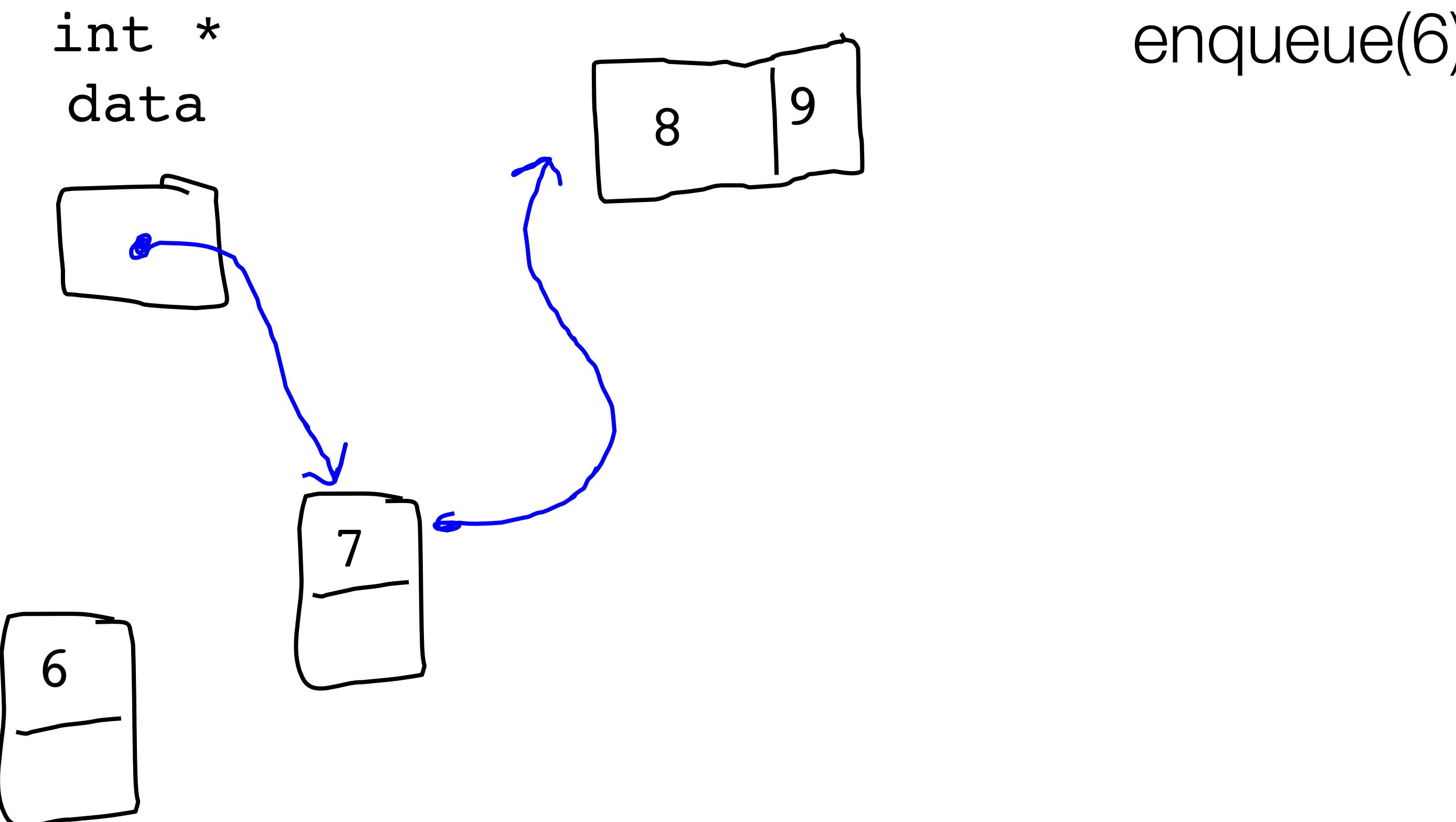
int *
data



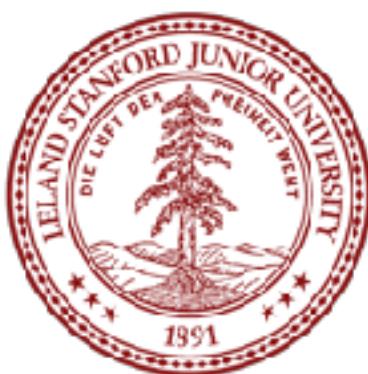
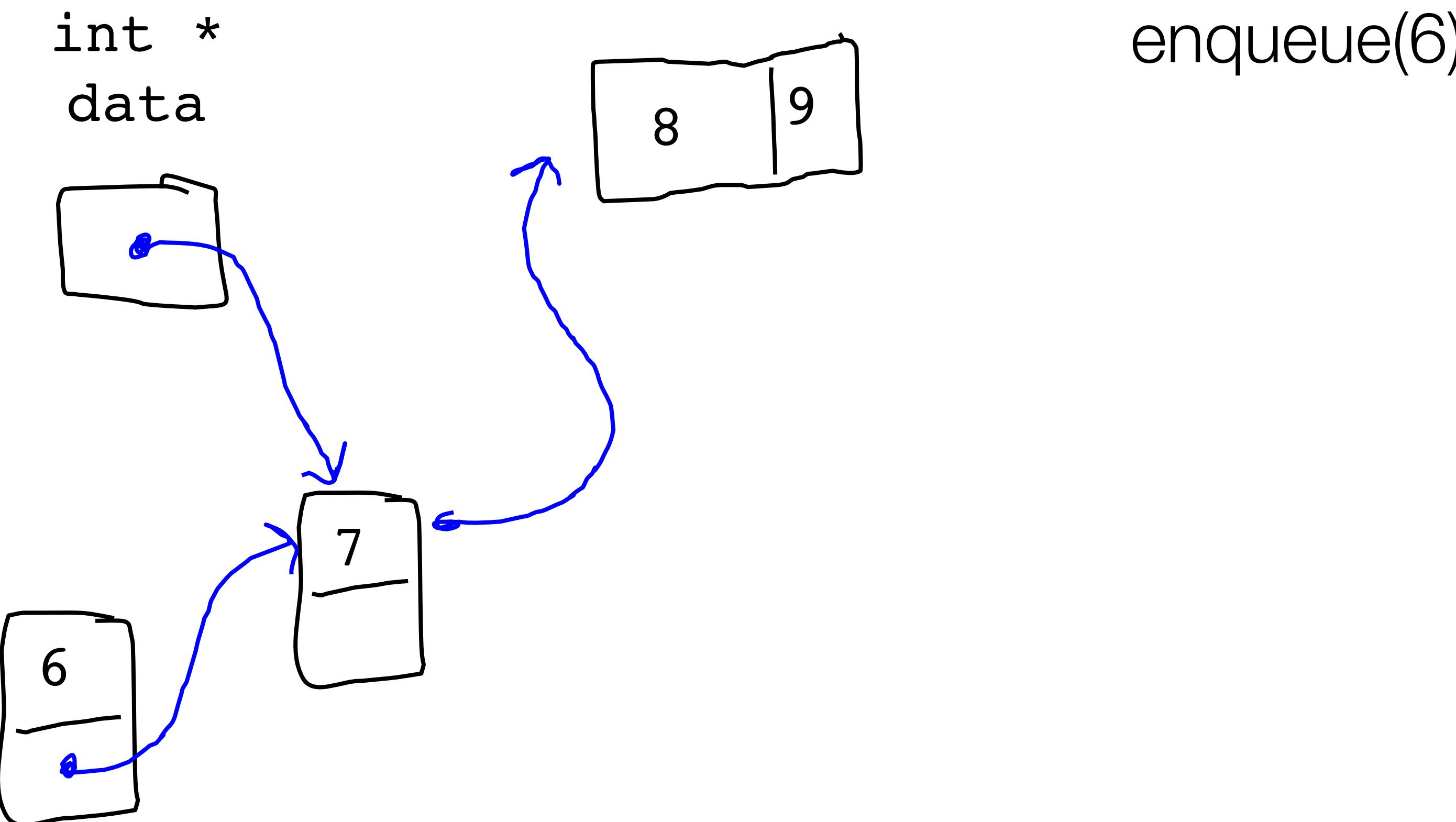
enqueue(7)



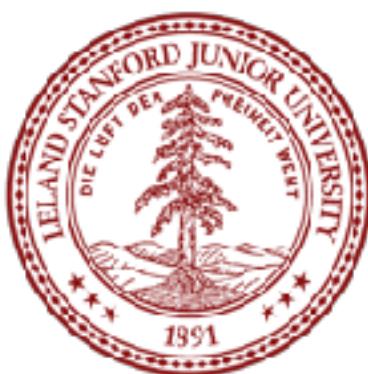
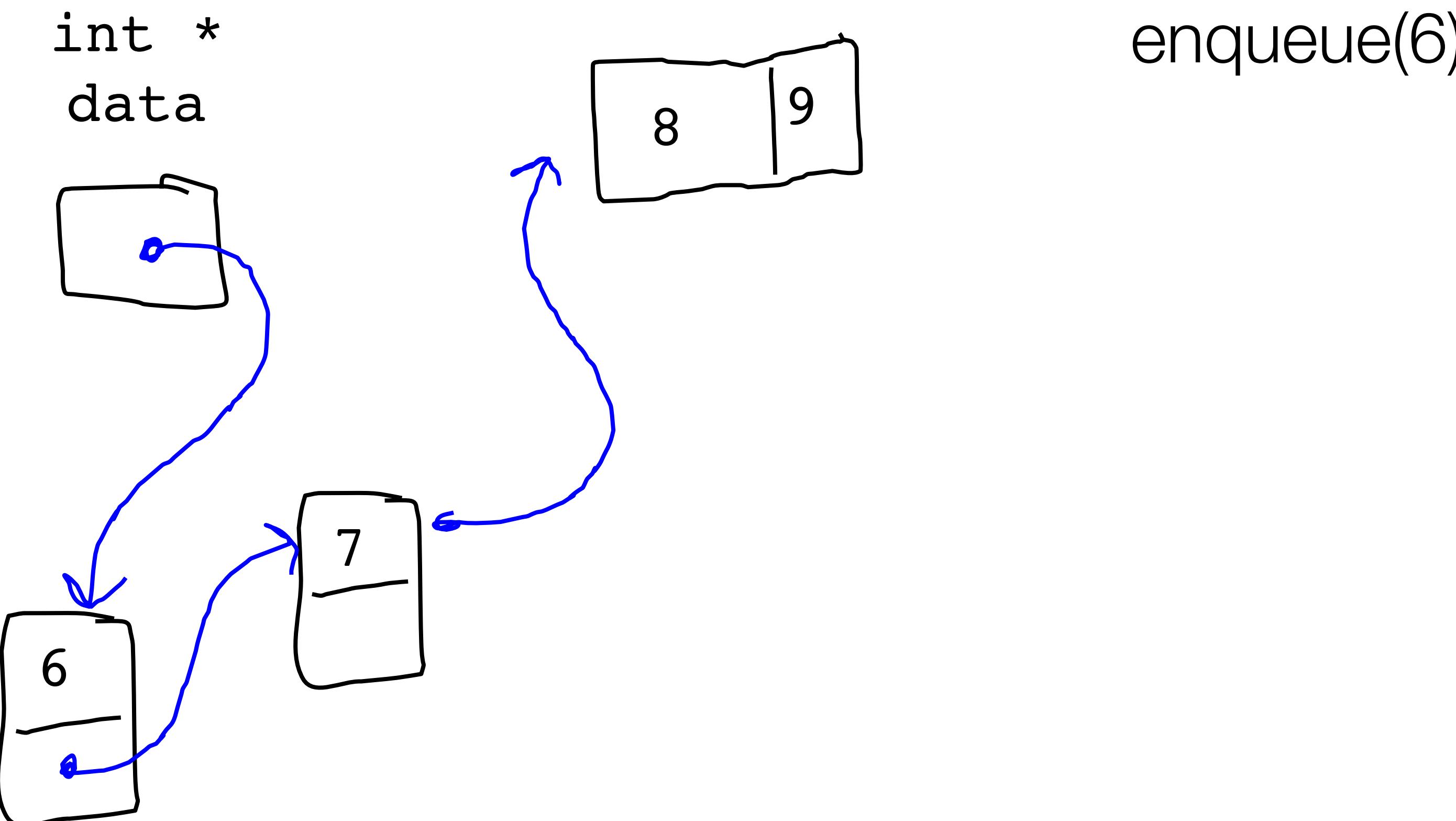
And Now for Something Completely Different



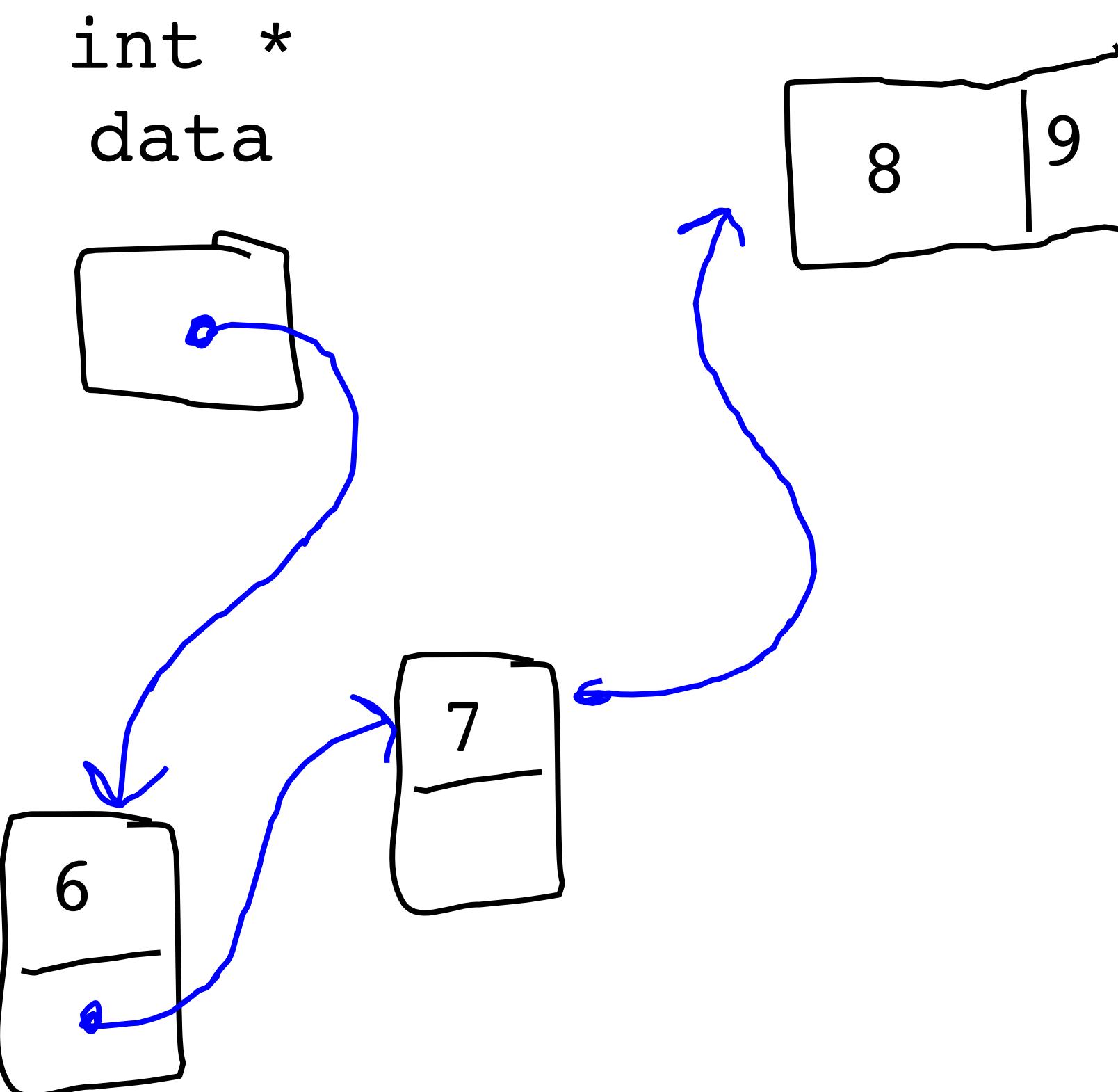
And Now for Something Completely Different



And Now for Something Completely Different



And Now for Something Completely Different

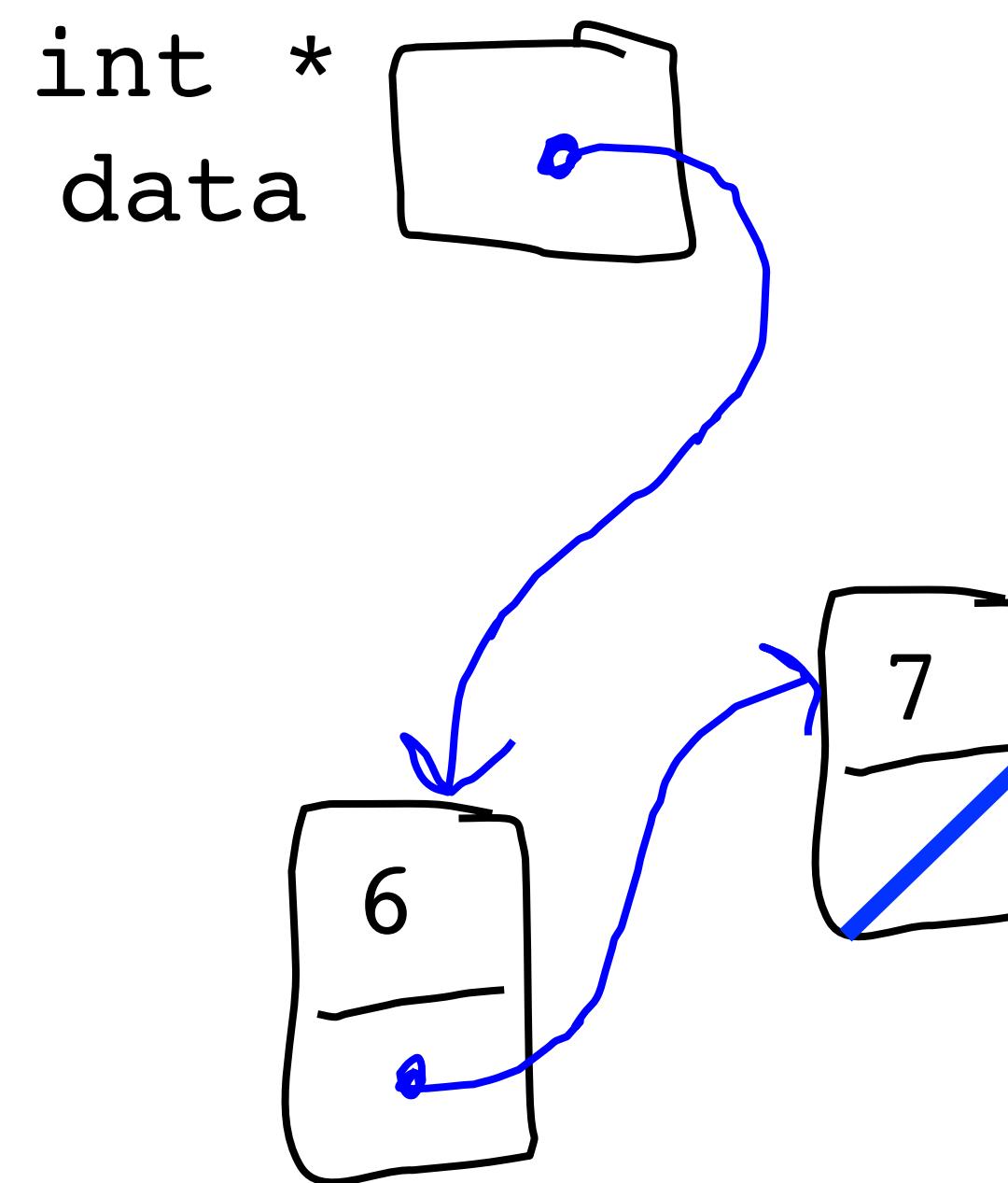
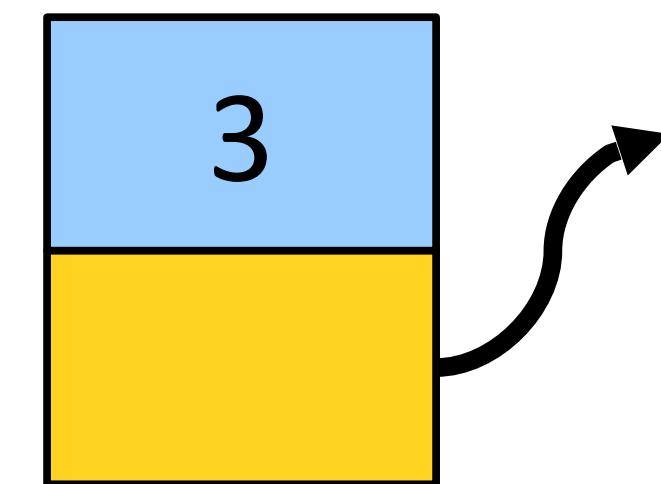


Now we have a way to add
to the front in O(1) time!



Linked List

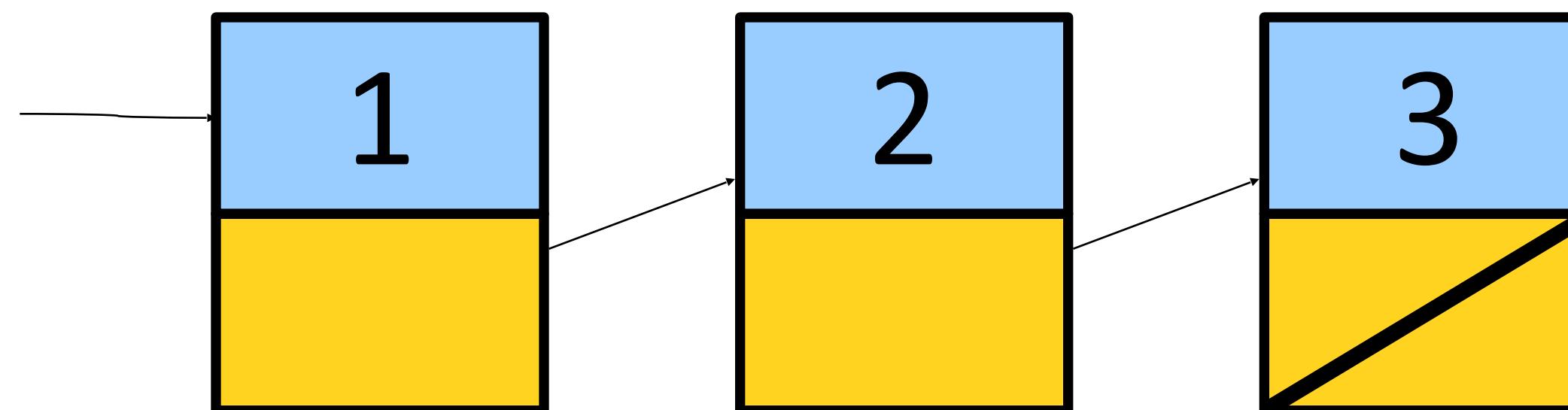
- A linked list is a chain of **nodes**.
- Each node contains two pieces of information:
 - Some piece of data that is stored in the sequence
 - A **link** to the next node in the list.
- We can traverse the list by starting at the first cell and repeatedly following its link.



Linked Lists

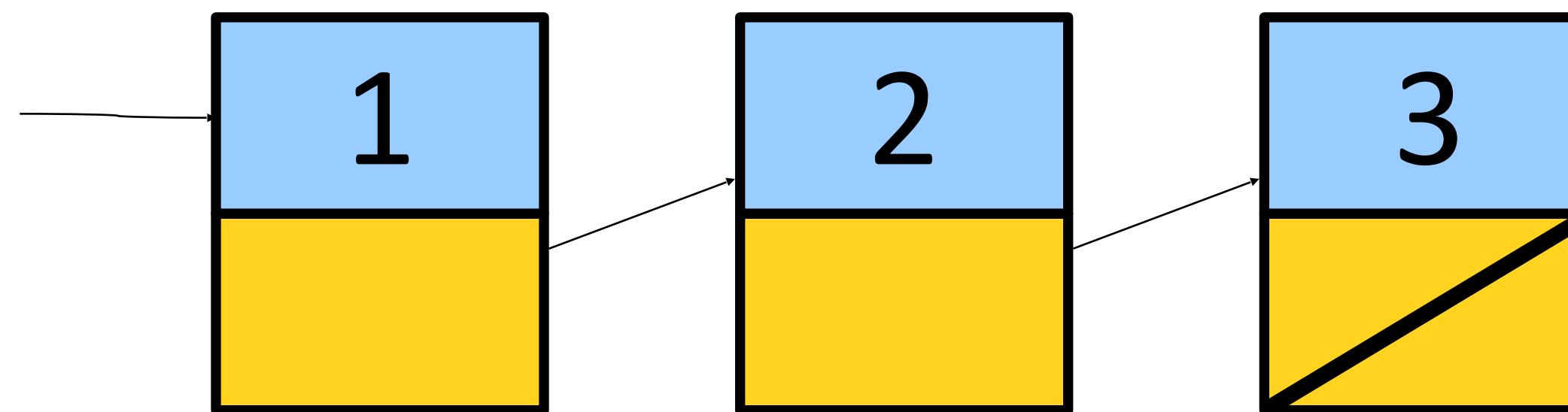


- A **linked list** is a data structure for storing a sequence of elements.
- Each element is stored separately from the rest.
- The elements are then chained together into a sequence.



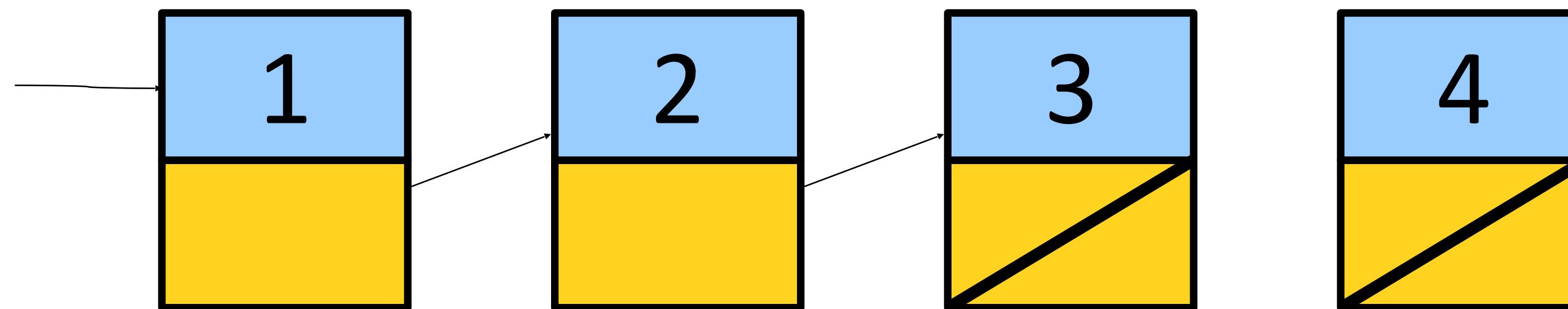
Linked Lists

- 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.



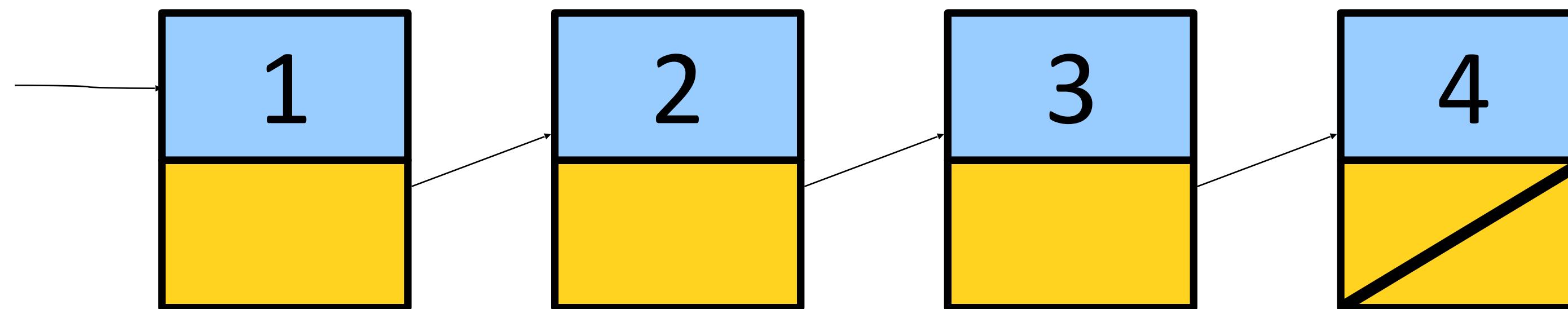
Linked Lists

- 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.



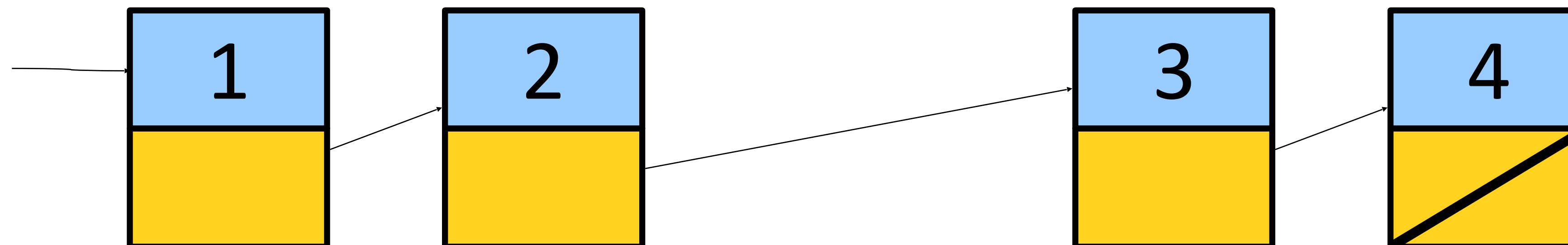
Linked Lists

- 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.



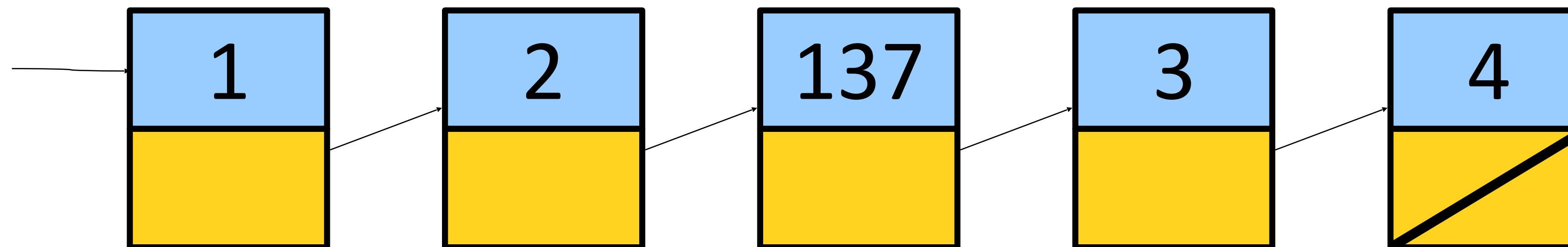
Linked Lists

- 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.



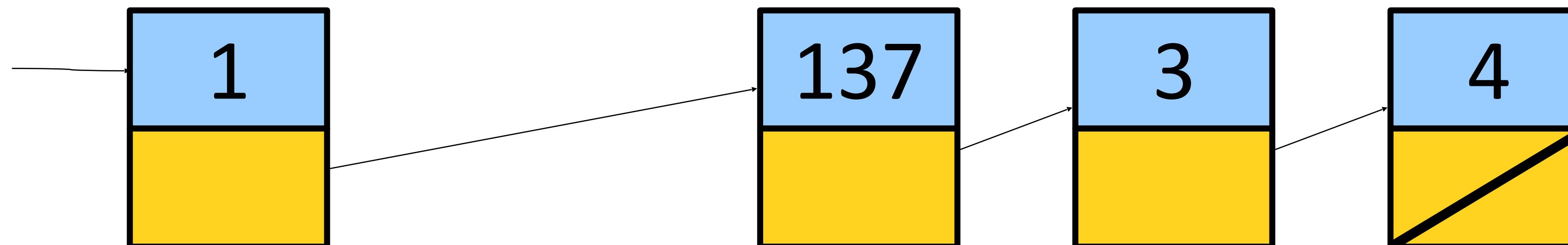
Linked Lists

- 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.



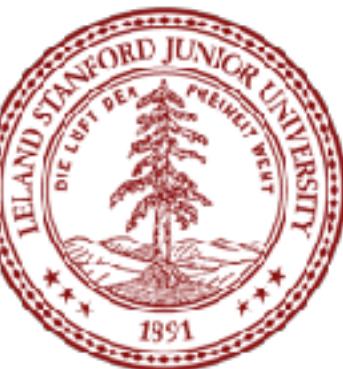
Linked Lists

- 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?

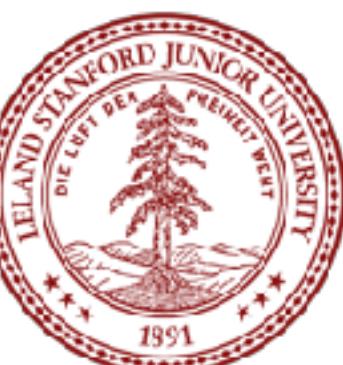
- Can efficiently splice new elements into the list or remove existing elements anywhere in the list.
- Never have to do a massive copy step;
- Has some tradeoffs; we'll see this later.



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:

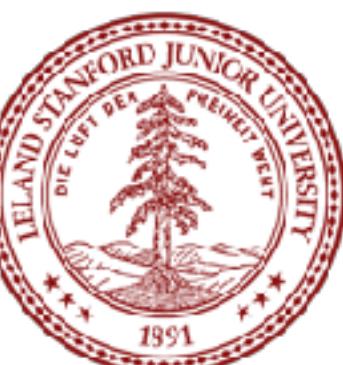
```
struct Node {  
    string value;  
    /* ? */ next;  
};
```



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:

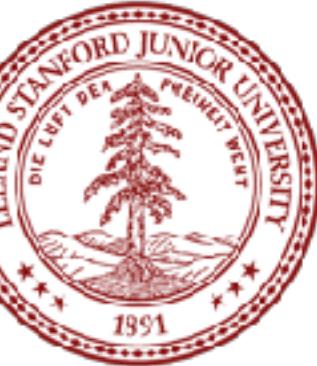
```
struct Node {  
    string value;  
    Node* next;  
};
```

- The structure is defined recursively!



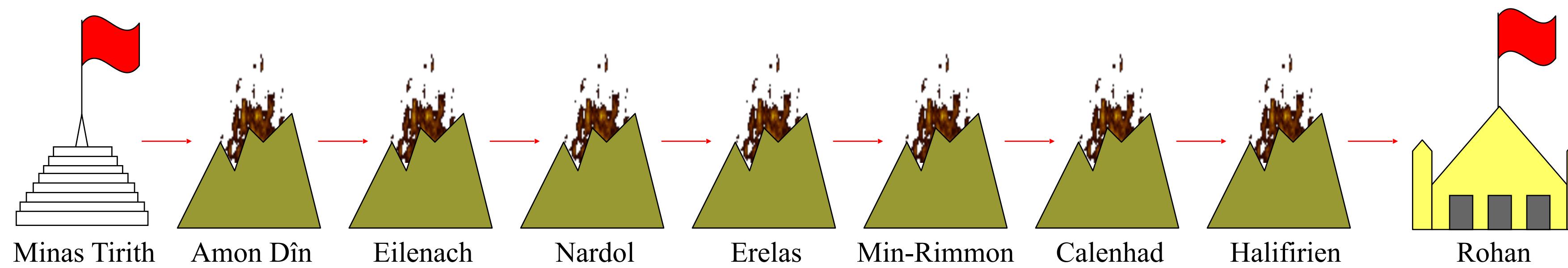
Always!

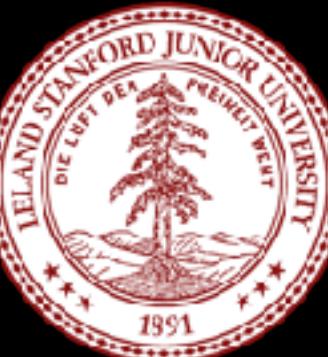
Draw a picture



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.

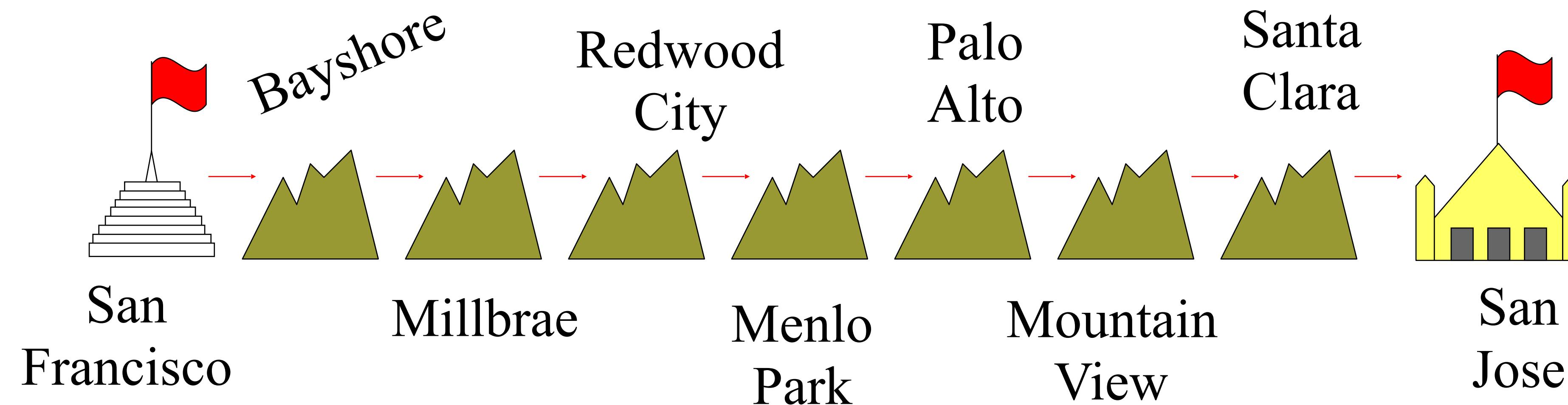




<https://www.youtube.com/watch?v=i6LGJ7evrAg>

Lord of the Linked Lists

Step 1: Make this linked list



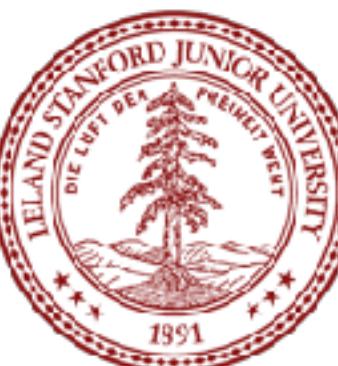
Step 2: Light the fires....

Lighting the fire of San Francisco!



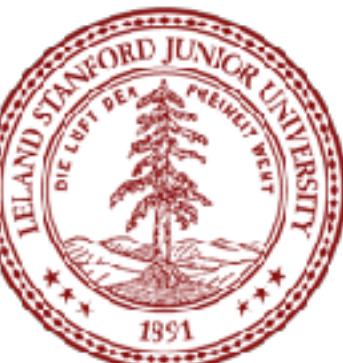
Lord of the Linked Lists

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



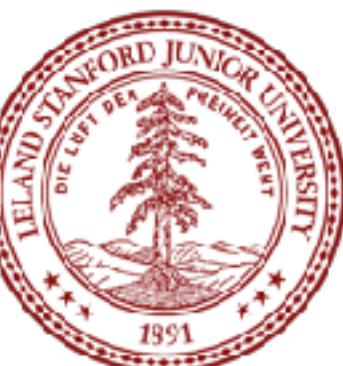
Lord of the Linked Lists

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



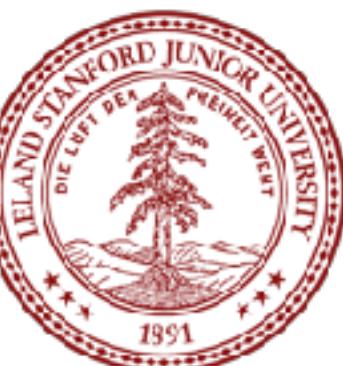
Lord of the Linked Lists

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



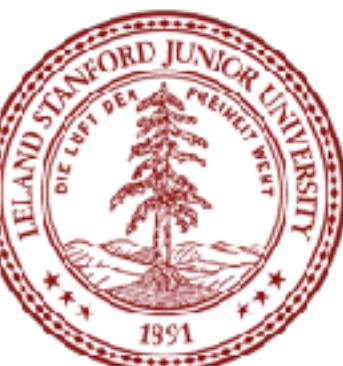
Lord of the Linked Lists

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

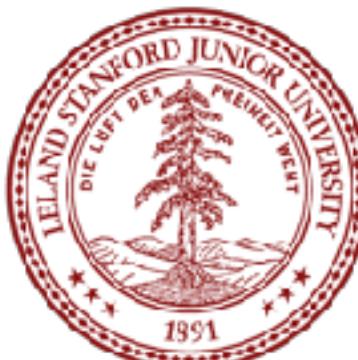


Lord of the Linked Lists

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



Lord of the Linked Lists



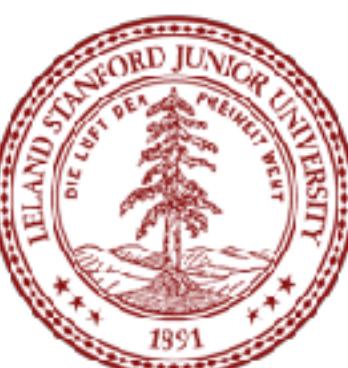
Linked List Trace

```
// 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;  
}
```



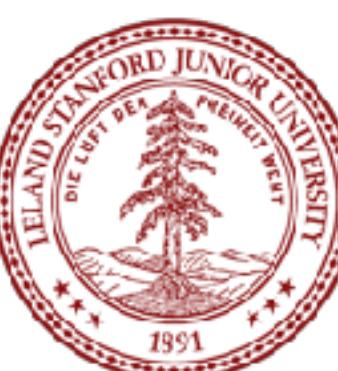
Linked List Trace

```
// 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;
}
```



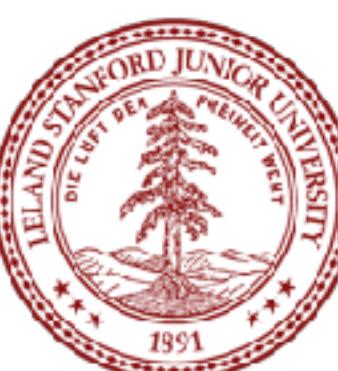
Linked List Trace

```
// 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;
}
```



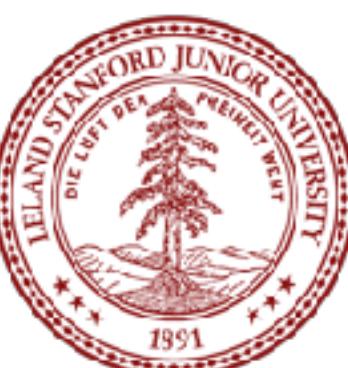
Linked List Trace

```
// 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;
}
```



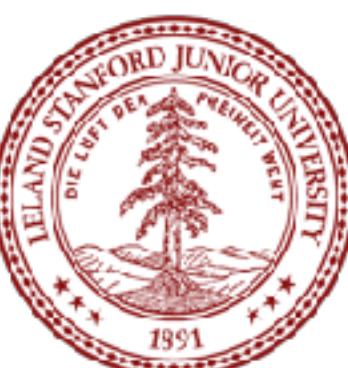
Linked List Trace

```
// 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;
}
```



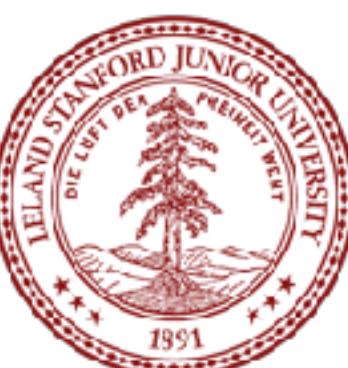
Linked List Trace

```
// 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;
}
```



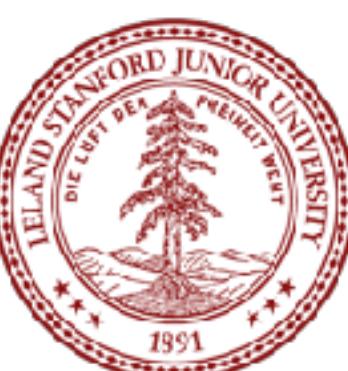
Linked List Trace

```
// 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;
}
```



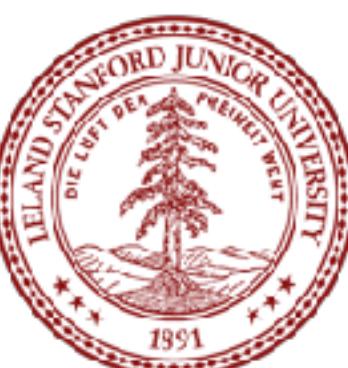
Linked List Trace

```
// 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;
}
```



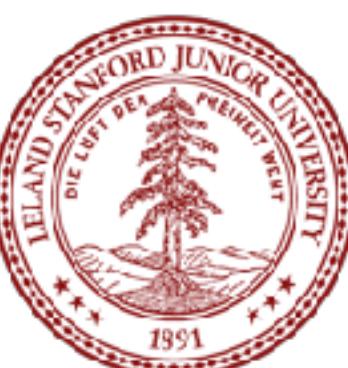
Linked List Trace

```
// 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;
}
```



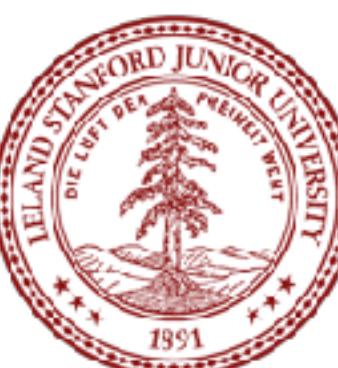
Linked List Trace

```
// 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); // Line 10
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("Dayshore", 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;
}
```



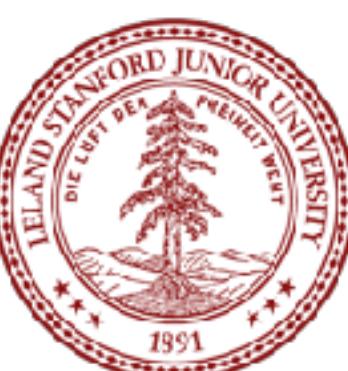
Linked List Trace

```
// 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);
```

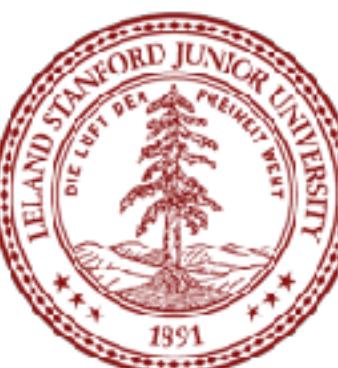
```
struct Tower{
    string name;
    Tower * link;
};
```

```
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);  
    }  
}
```

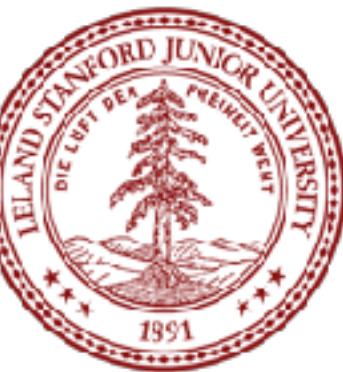
```
signal(head);
```

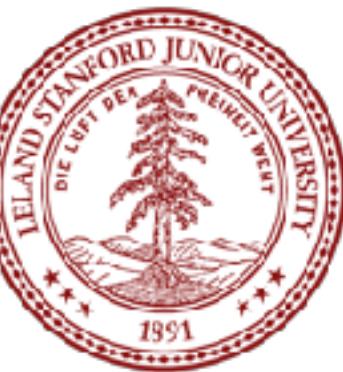
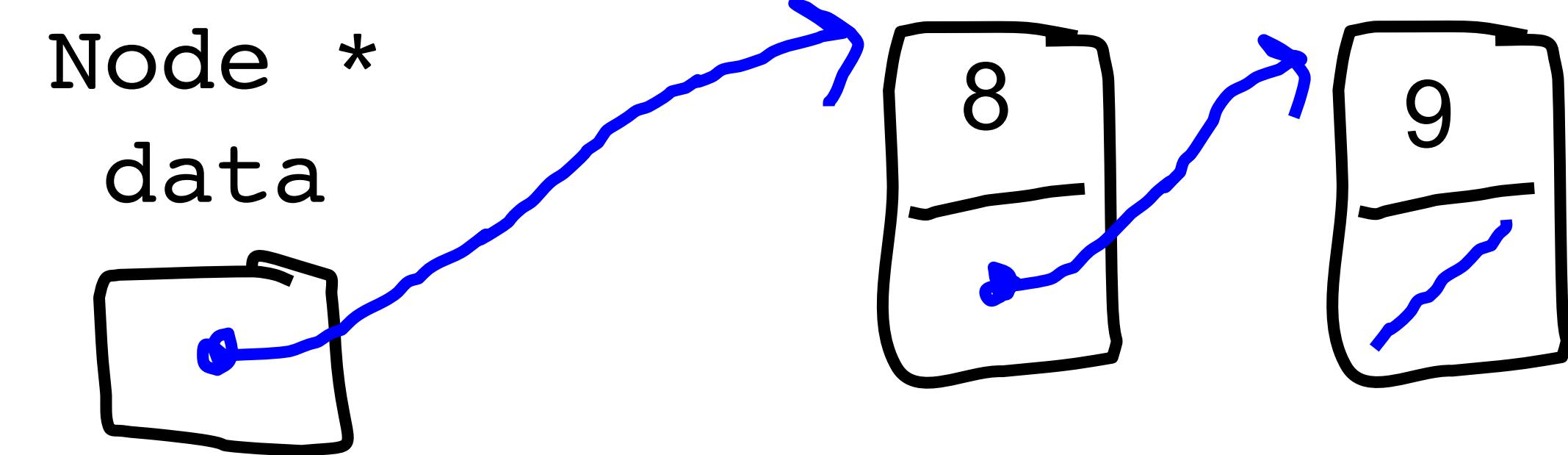


How is the Stack Implemented?



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





Stack

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

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

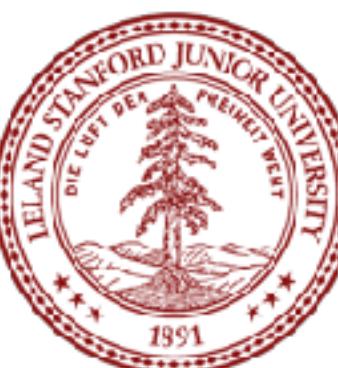
private:
    struct Node {
        int value;
        Node * link;
    };
    Node * data;          // member variables
};
```

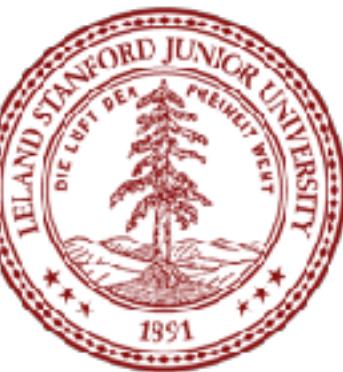
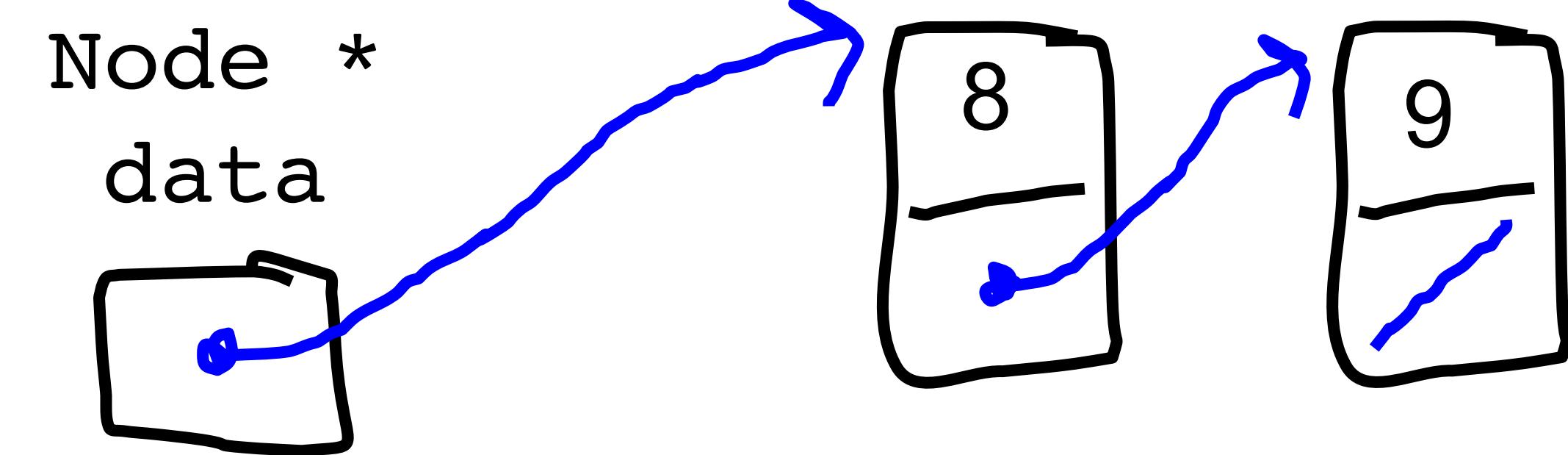


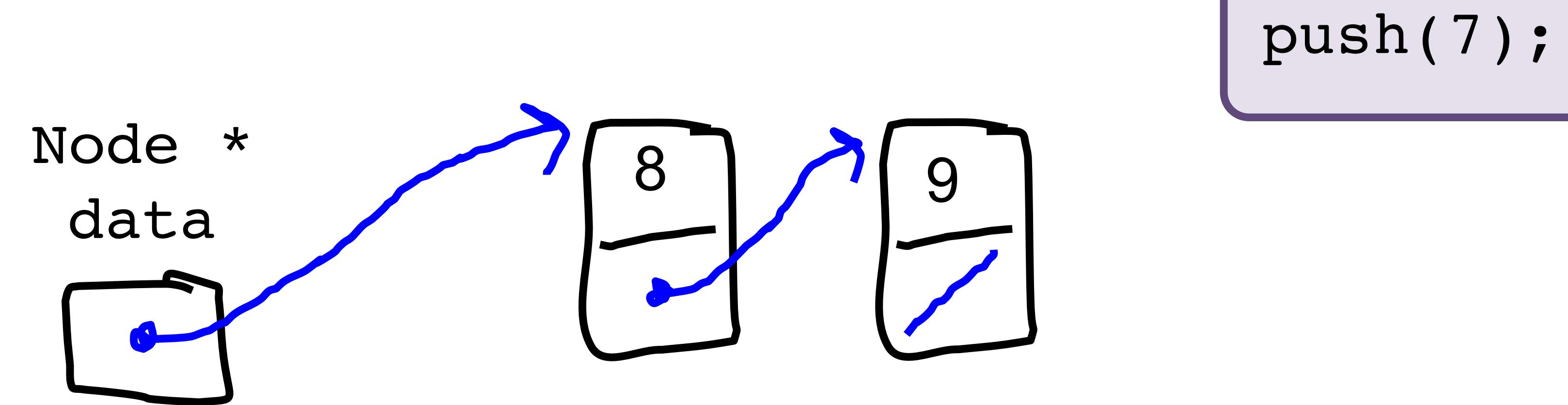
Stack Implementation

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

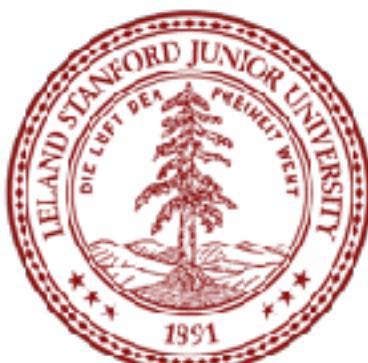
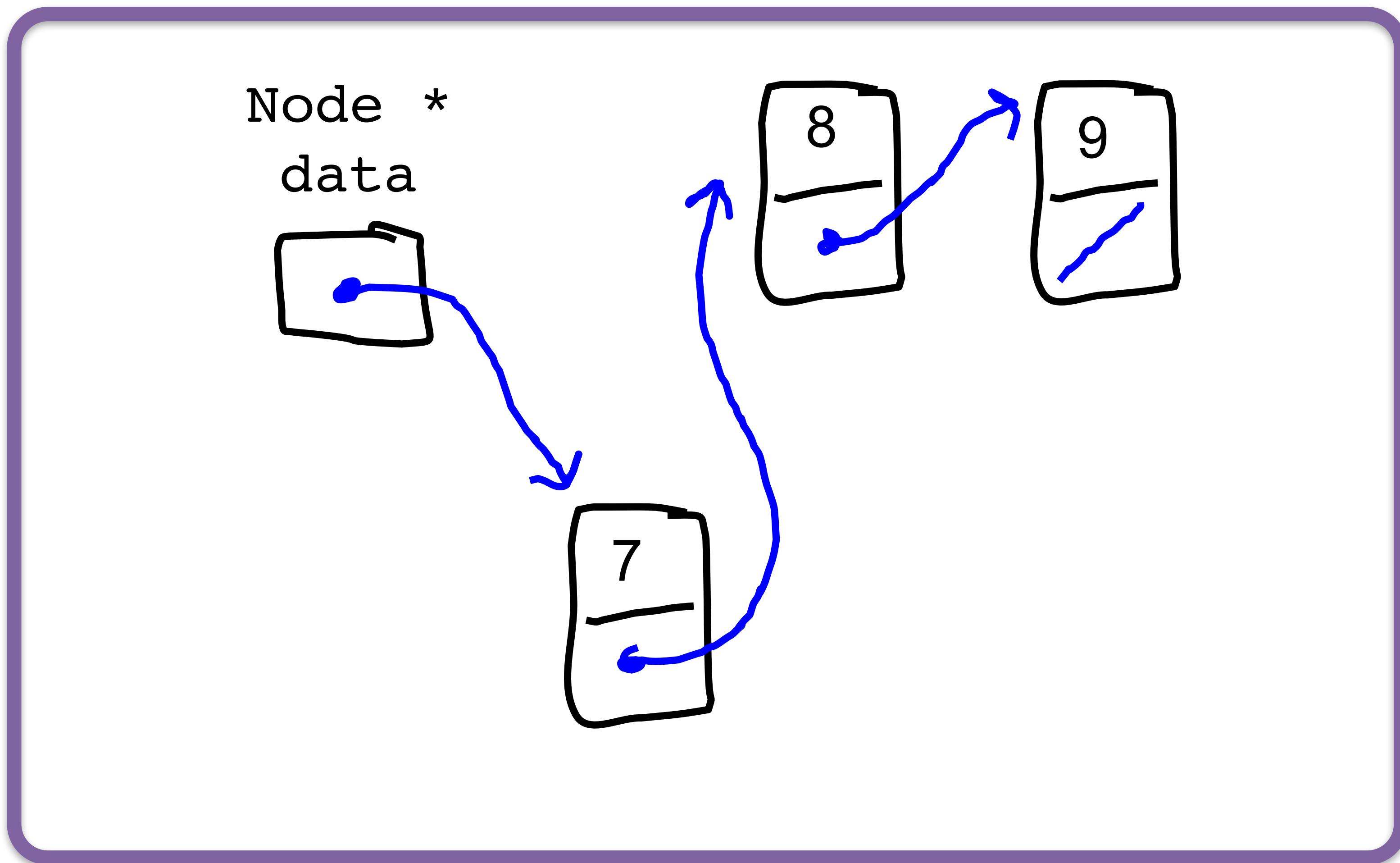
int StackInt::pop() {
    int toReturn = data->value;
    Node * temp = data;
    data = temp->link;
    delete temp;
    return toReturn;
}
```

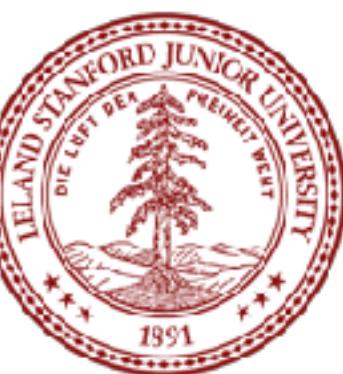
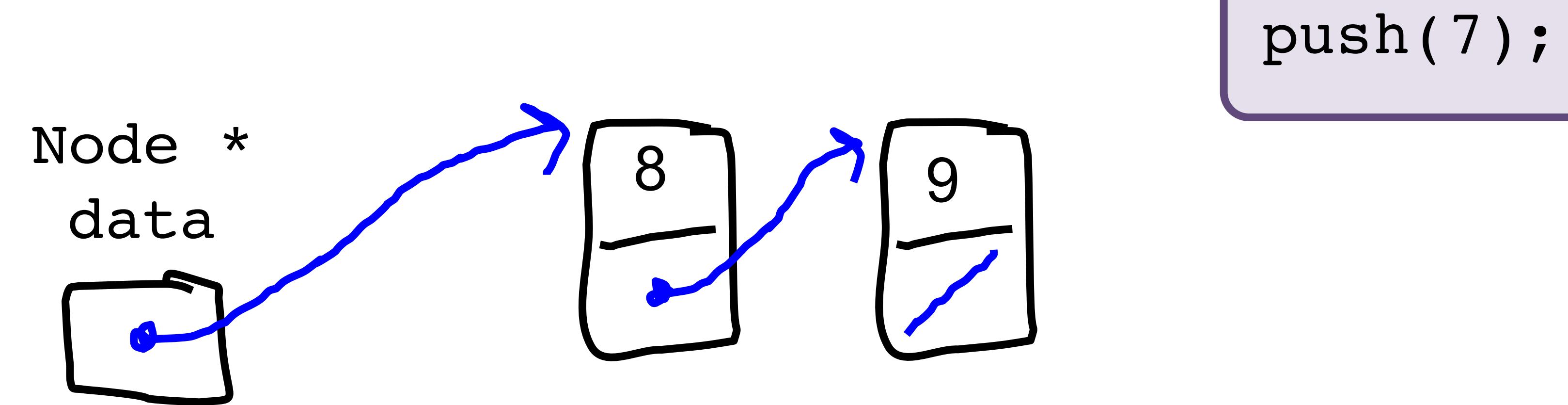






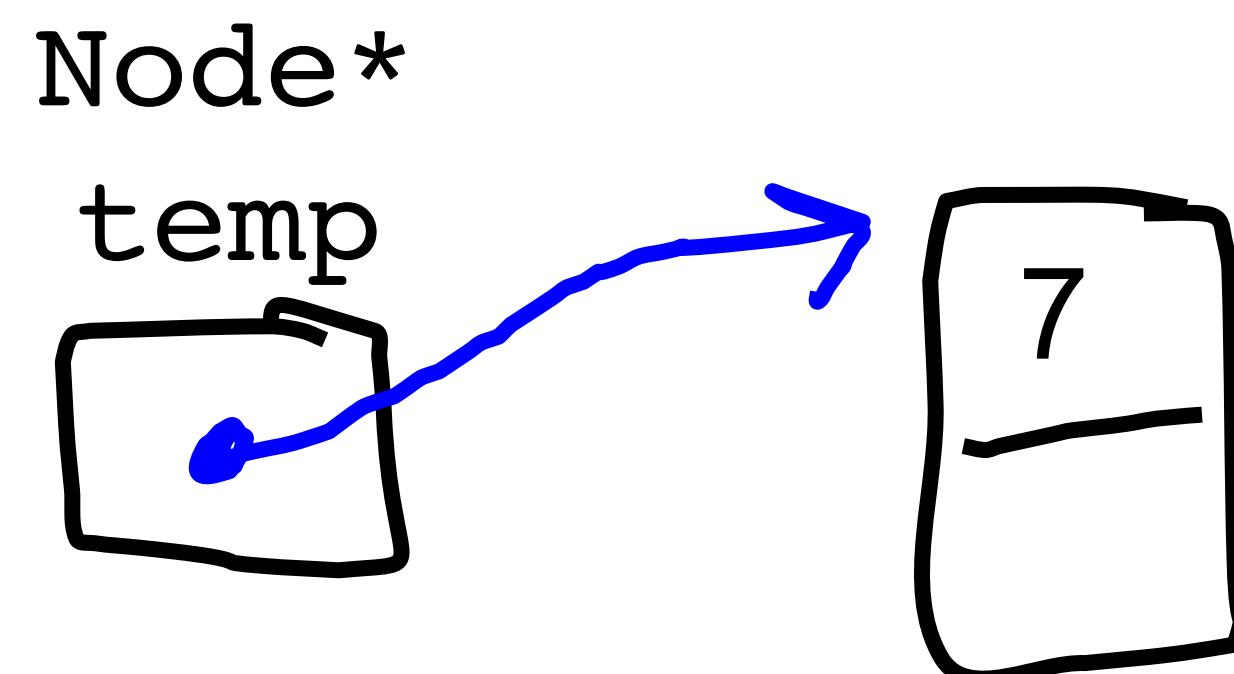
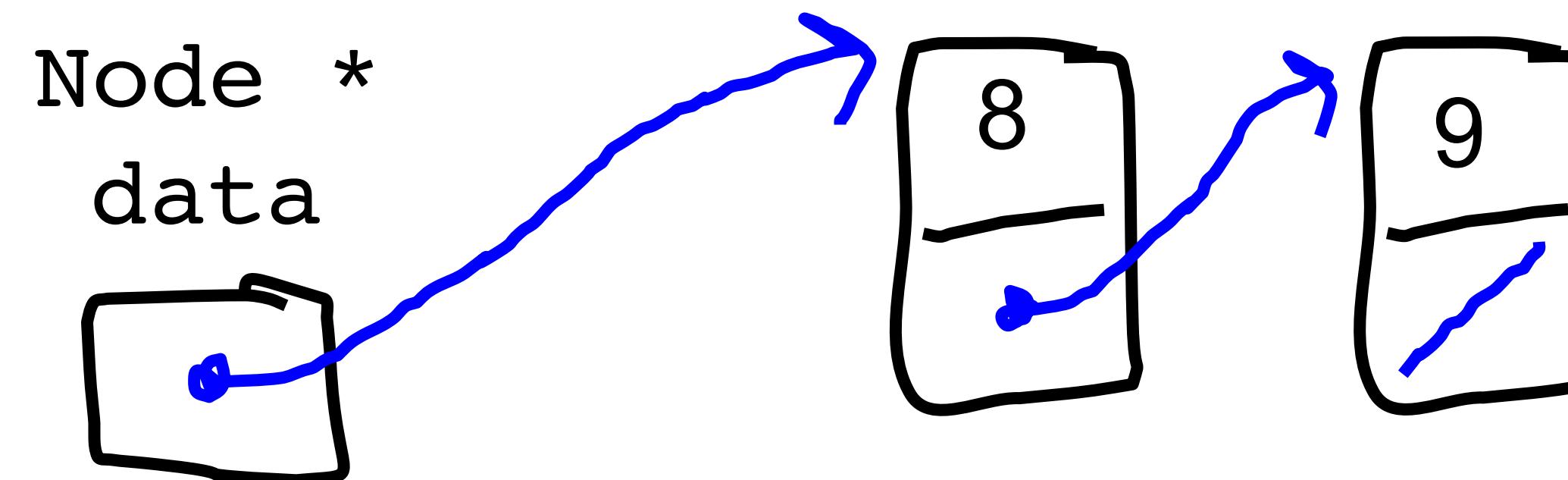
Goal of Push





Stack is a Linked List

push(7);

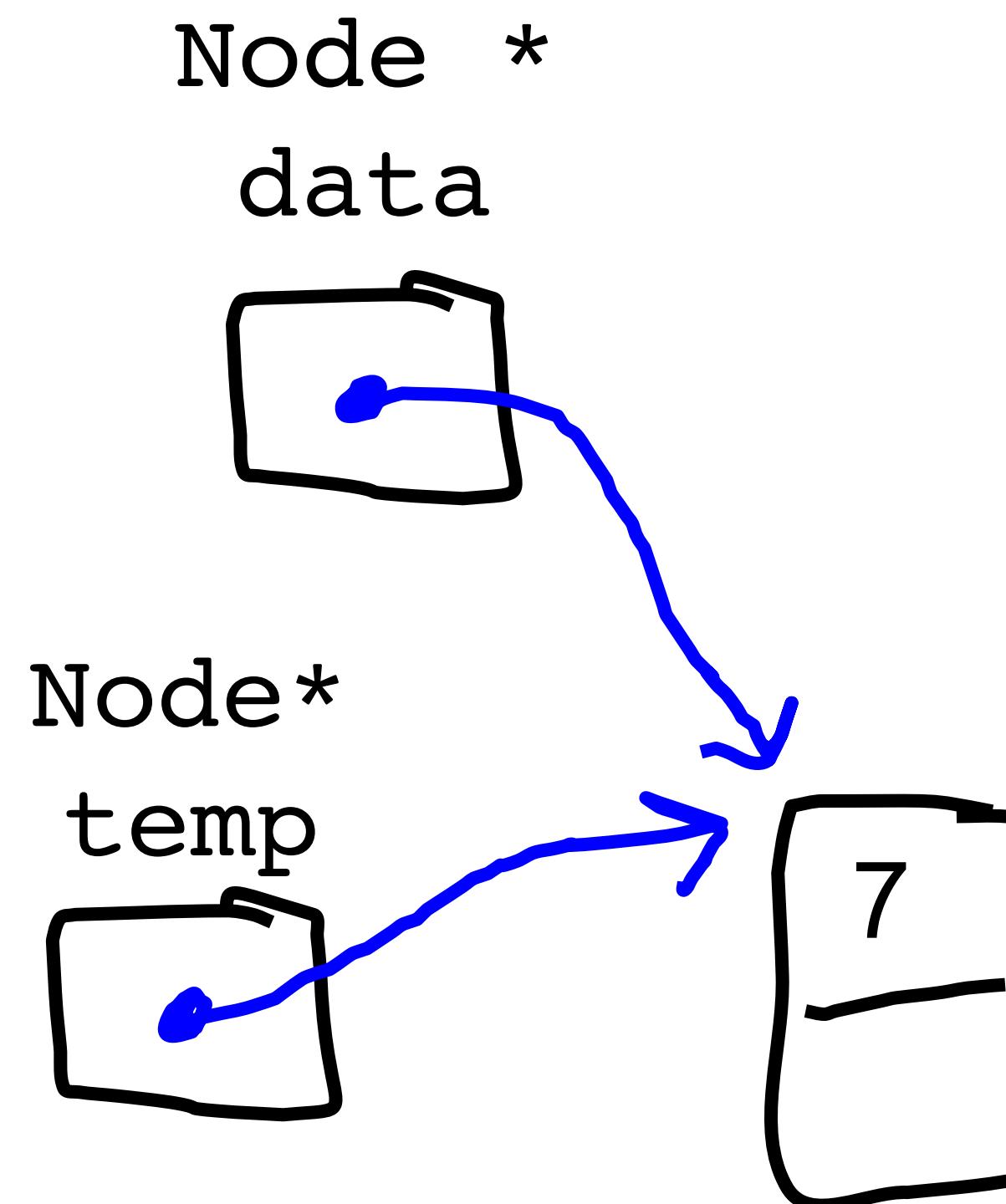


```
Node * temp = new Node;  
temp -> value = 7;
```

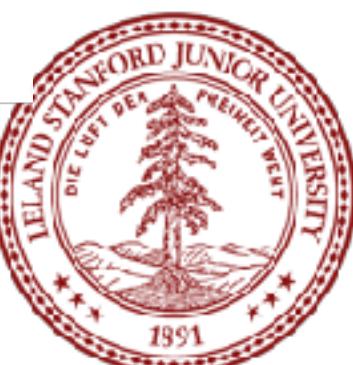
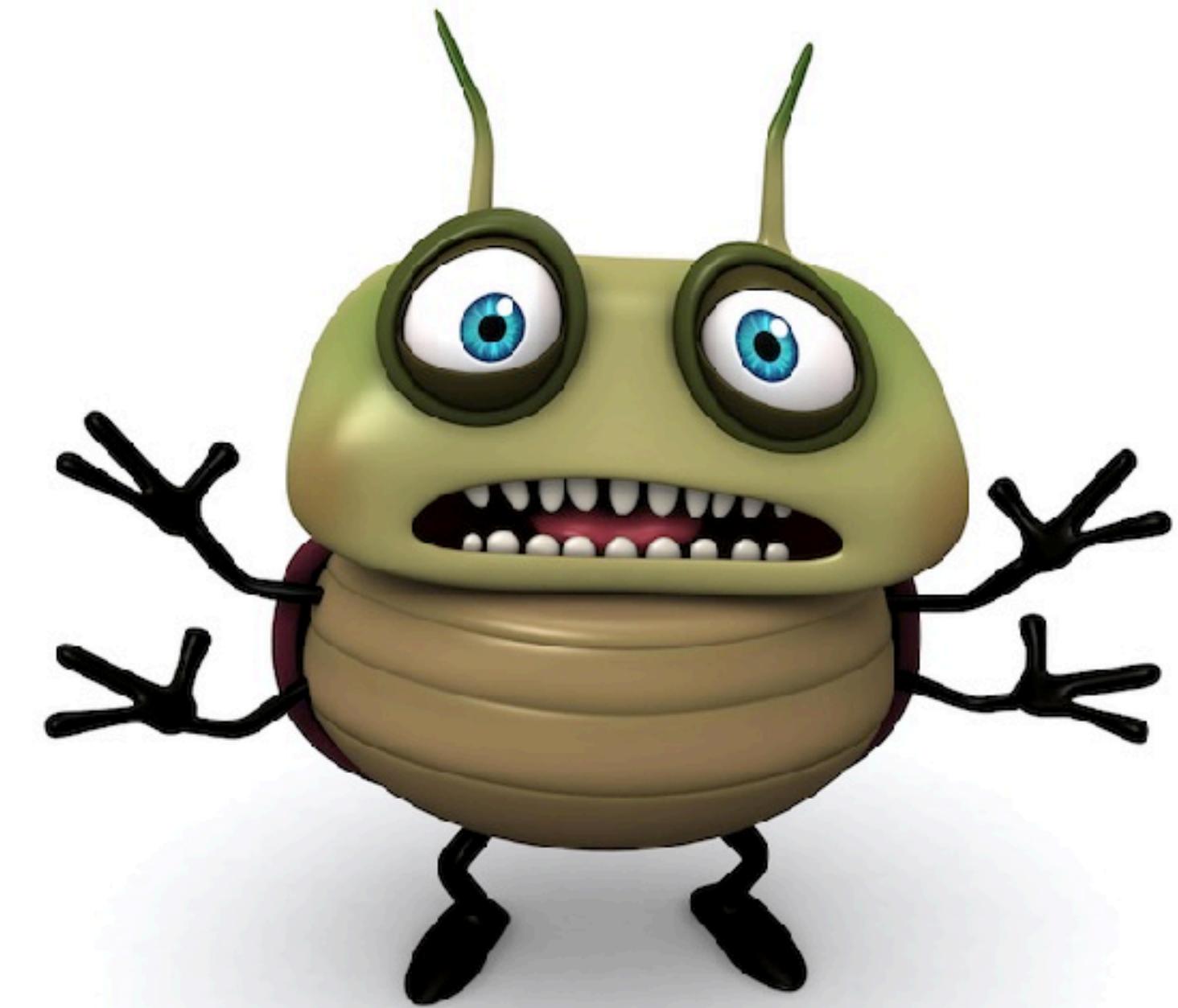


Stack is a Linked List

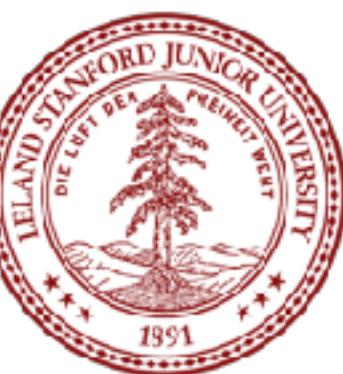
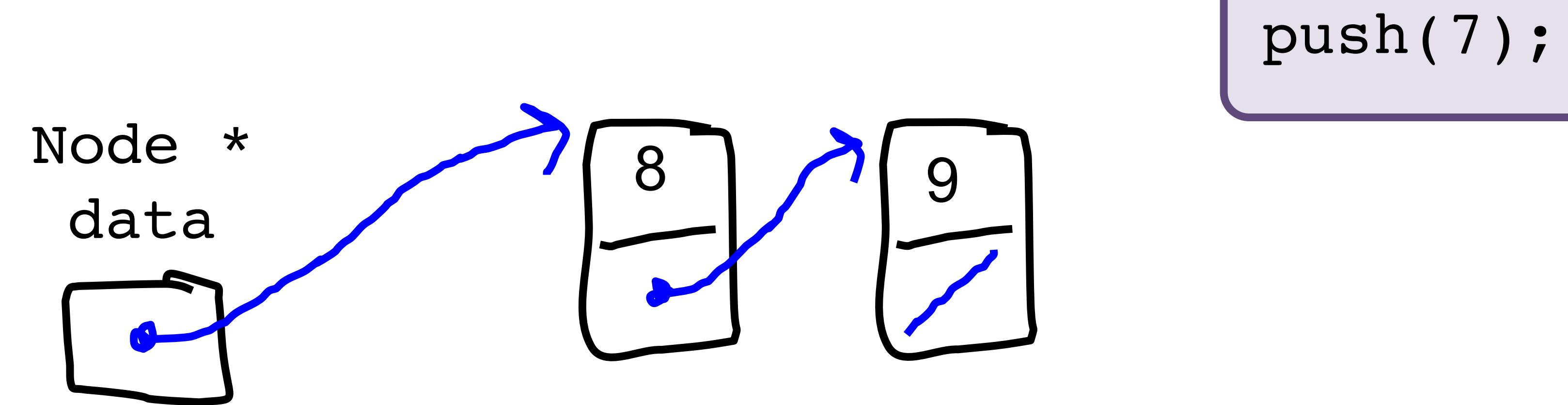
push(7);



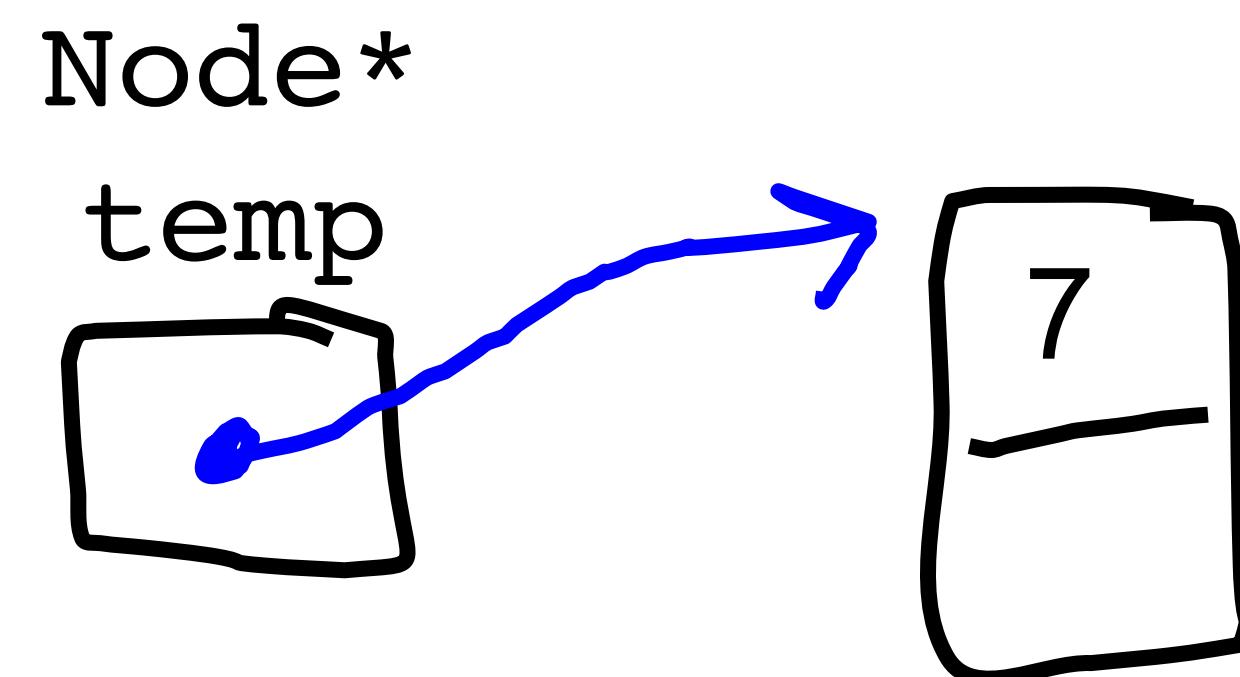
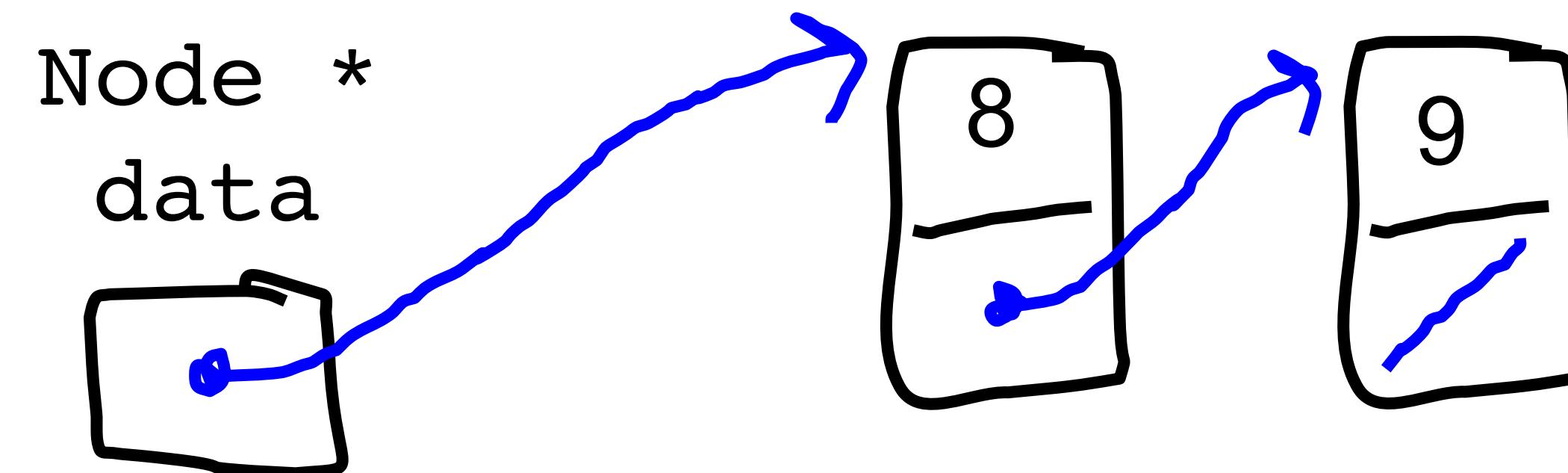
data = temp;



BRAAWWRRRR!



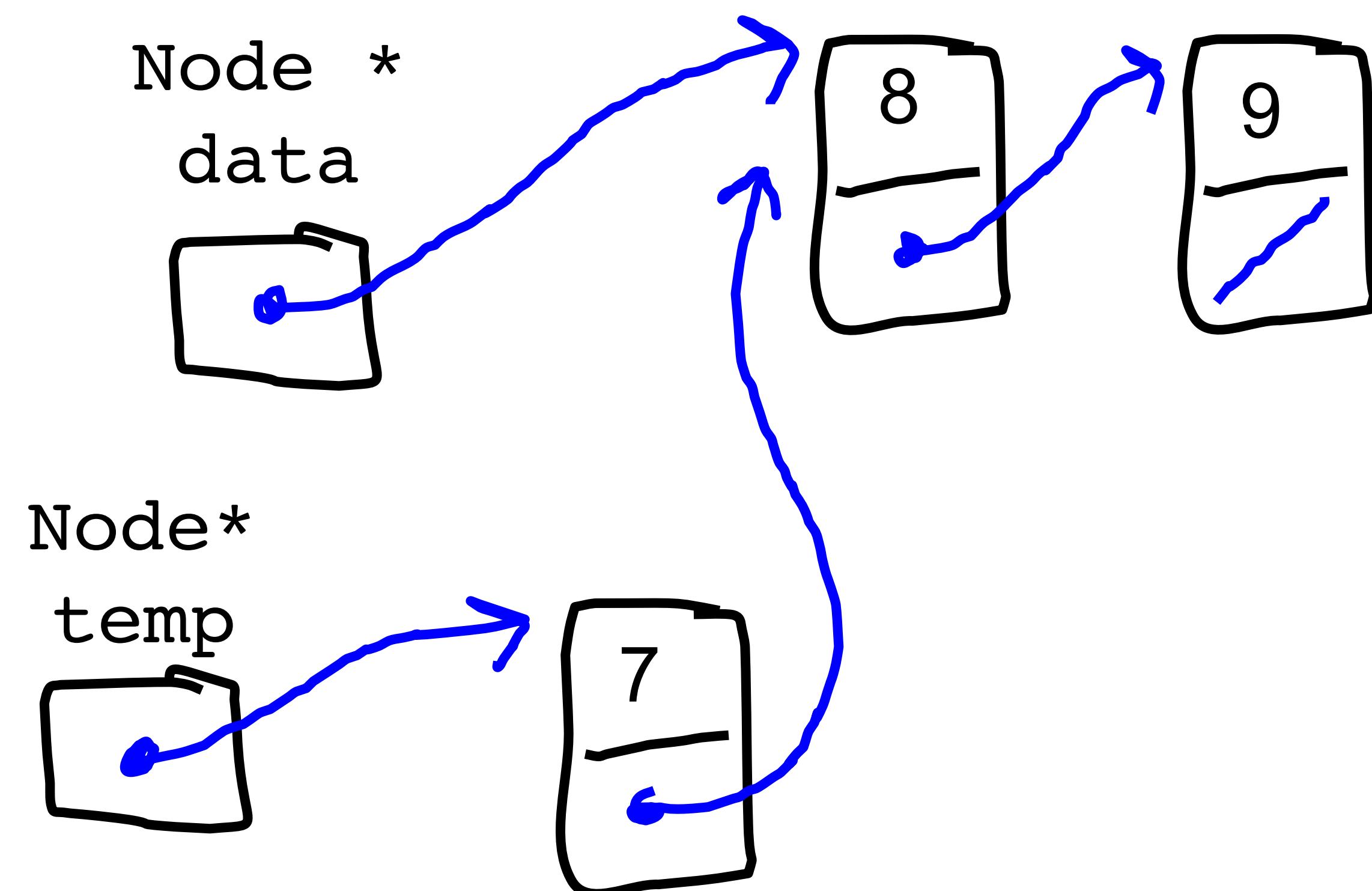
`push(7);`



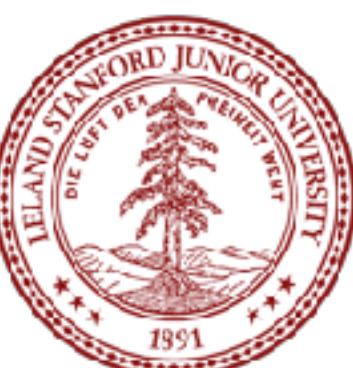
```
Node * temp = new Node;  
temp -> value = 7;
```



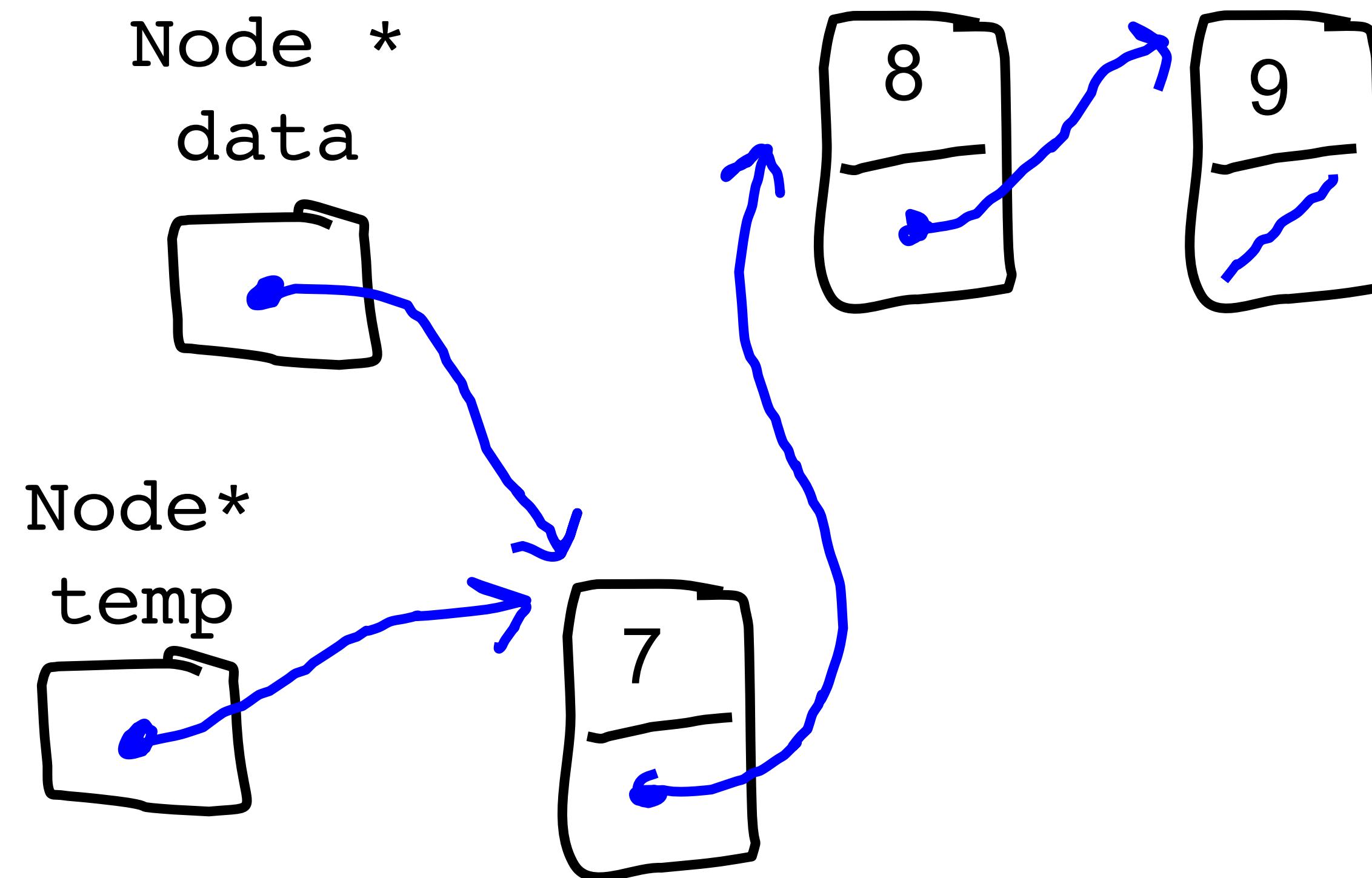
`push(7);`



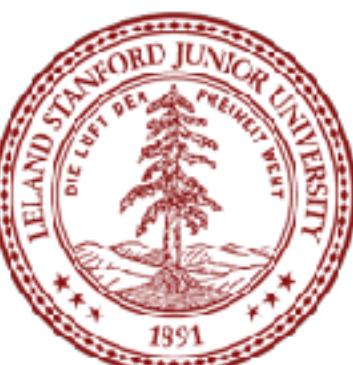
`temp -> link = data;`



`push(7);`

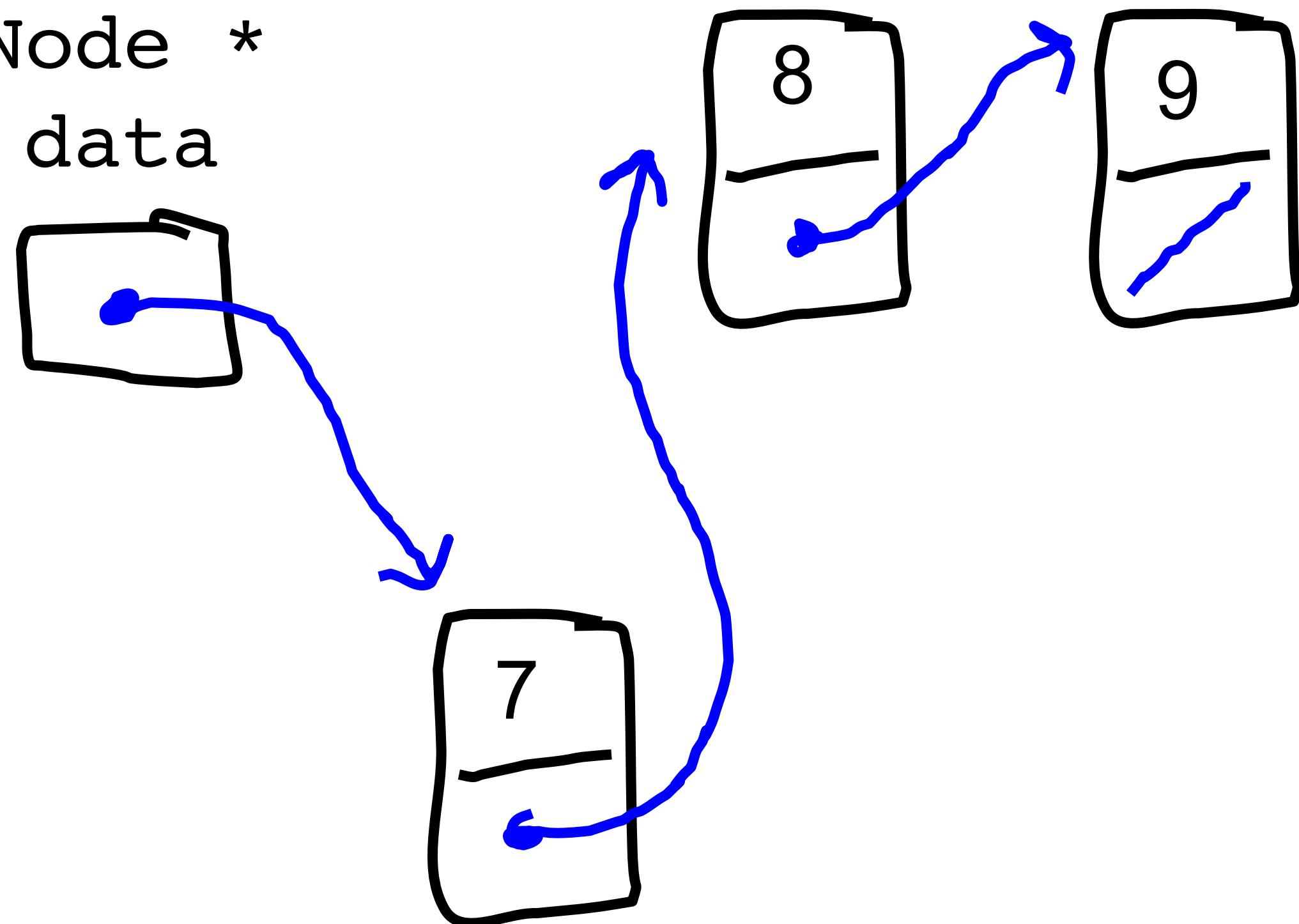


`data = temp;`

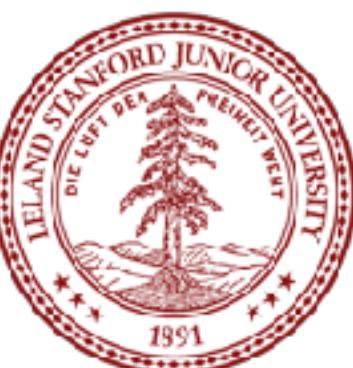


`push(7);`

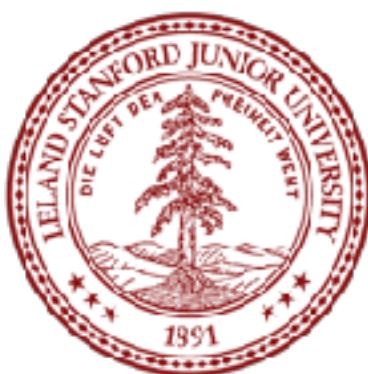
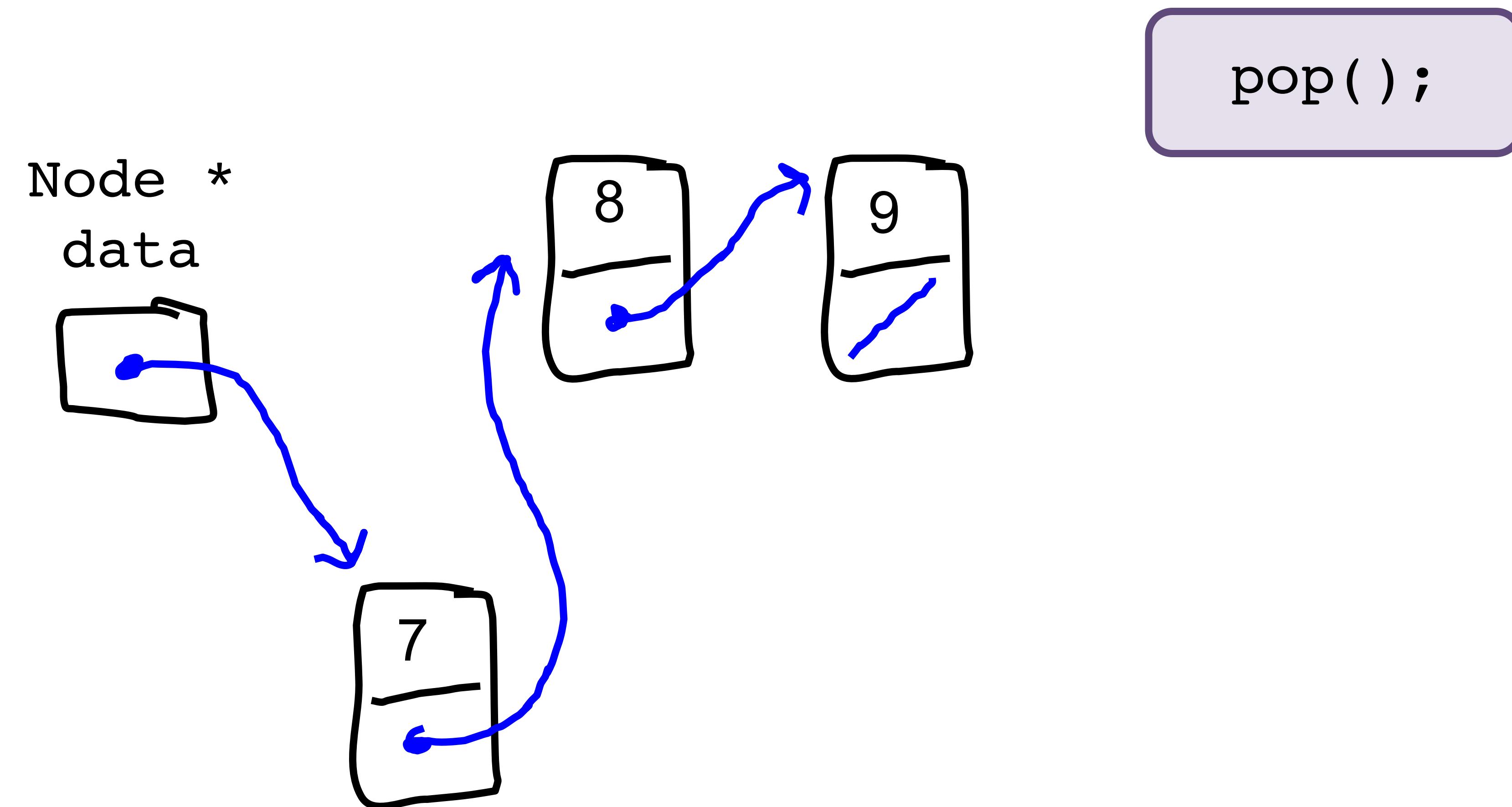
`Node *`
`data`



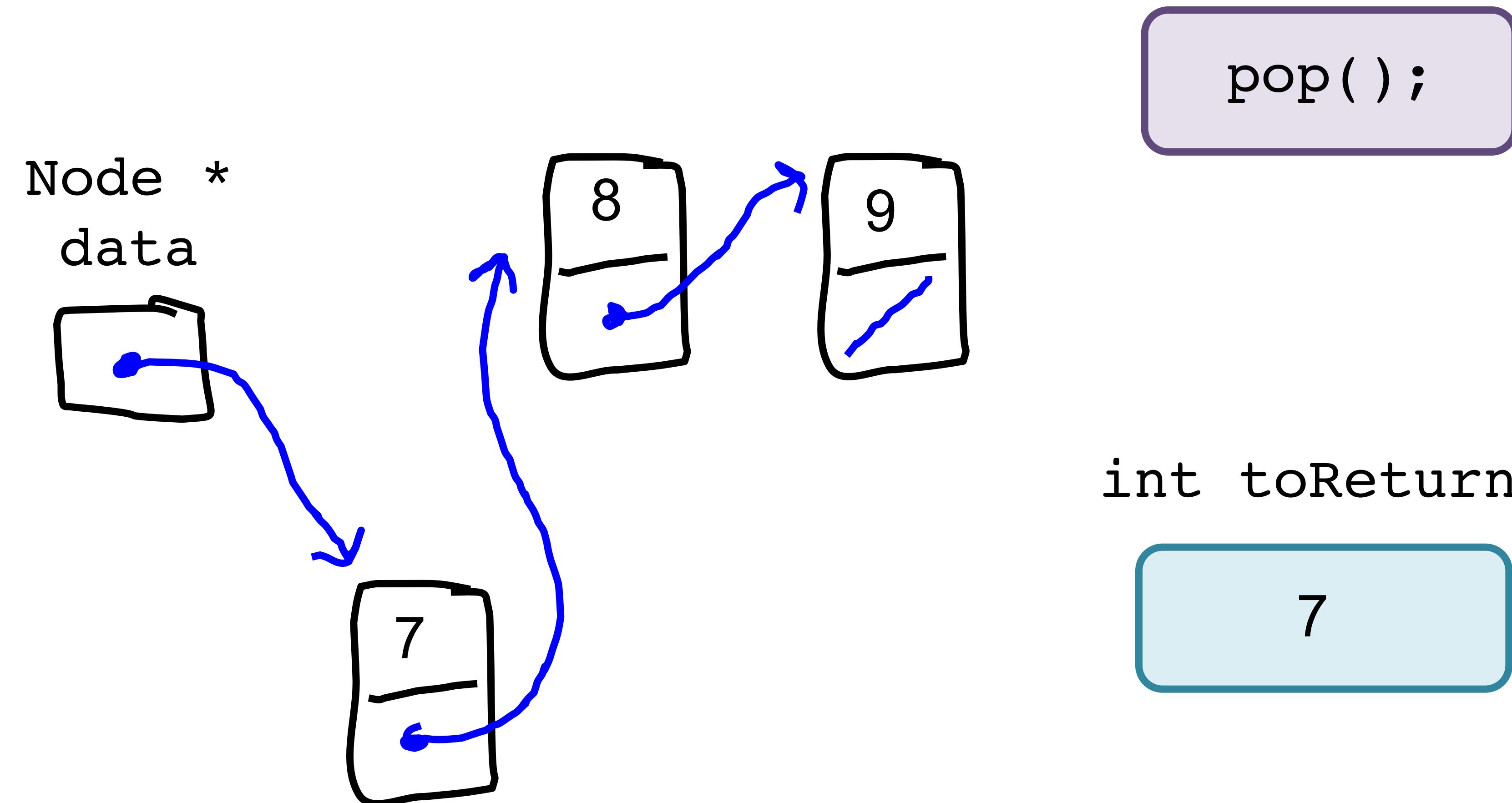
exit function



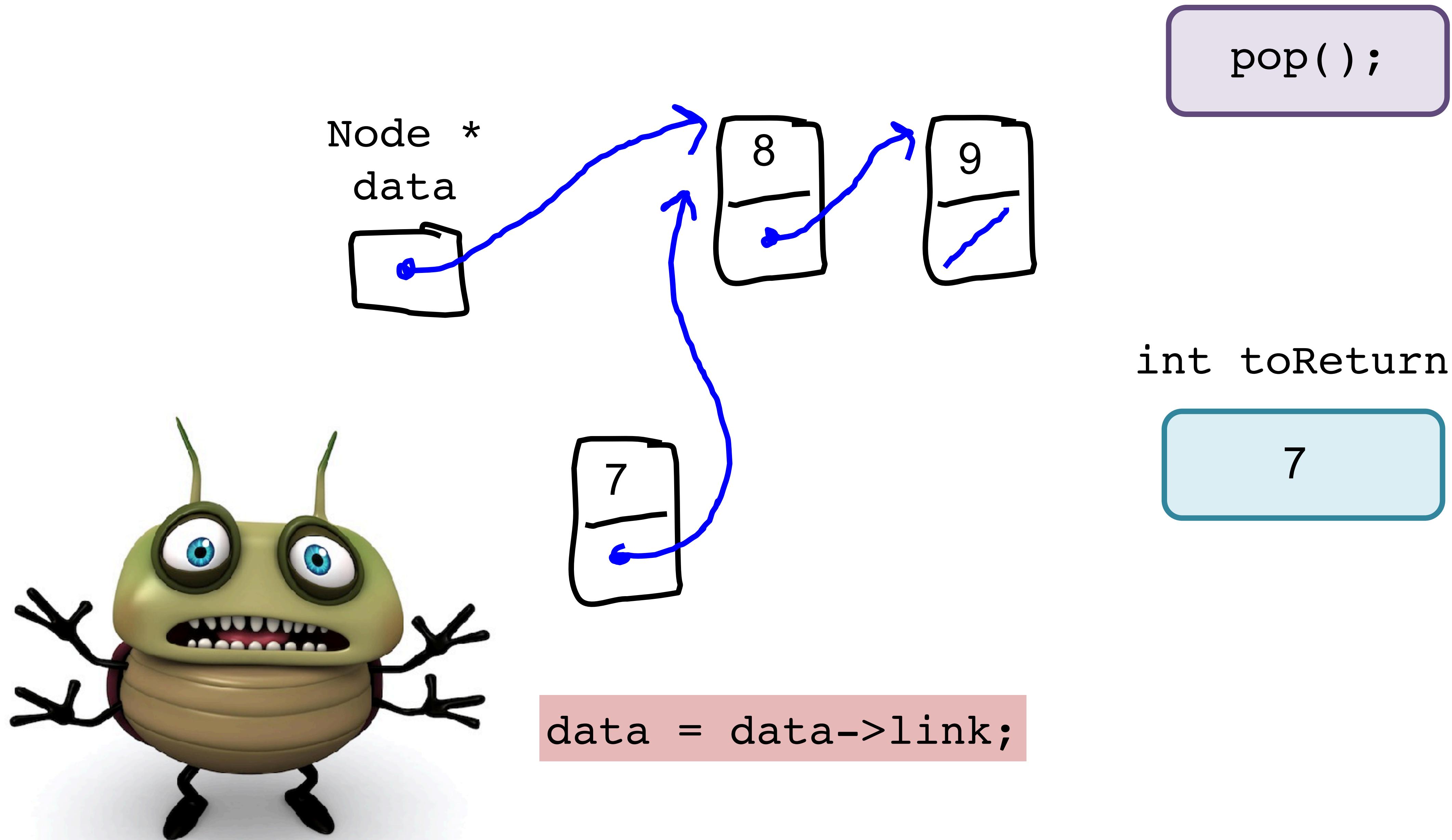
Stack is a Linked List



Stack is a Linked List

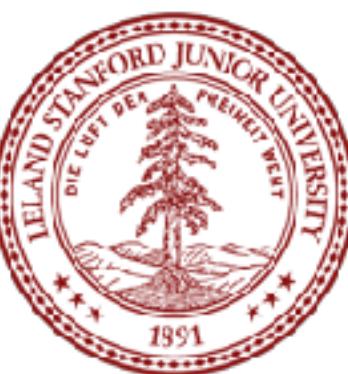
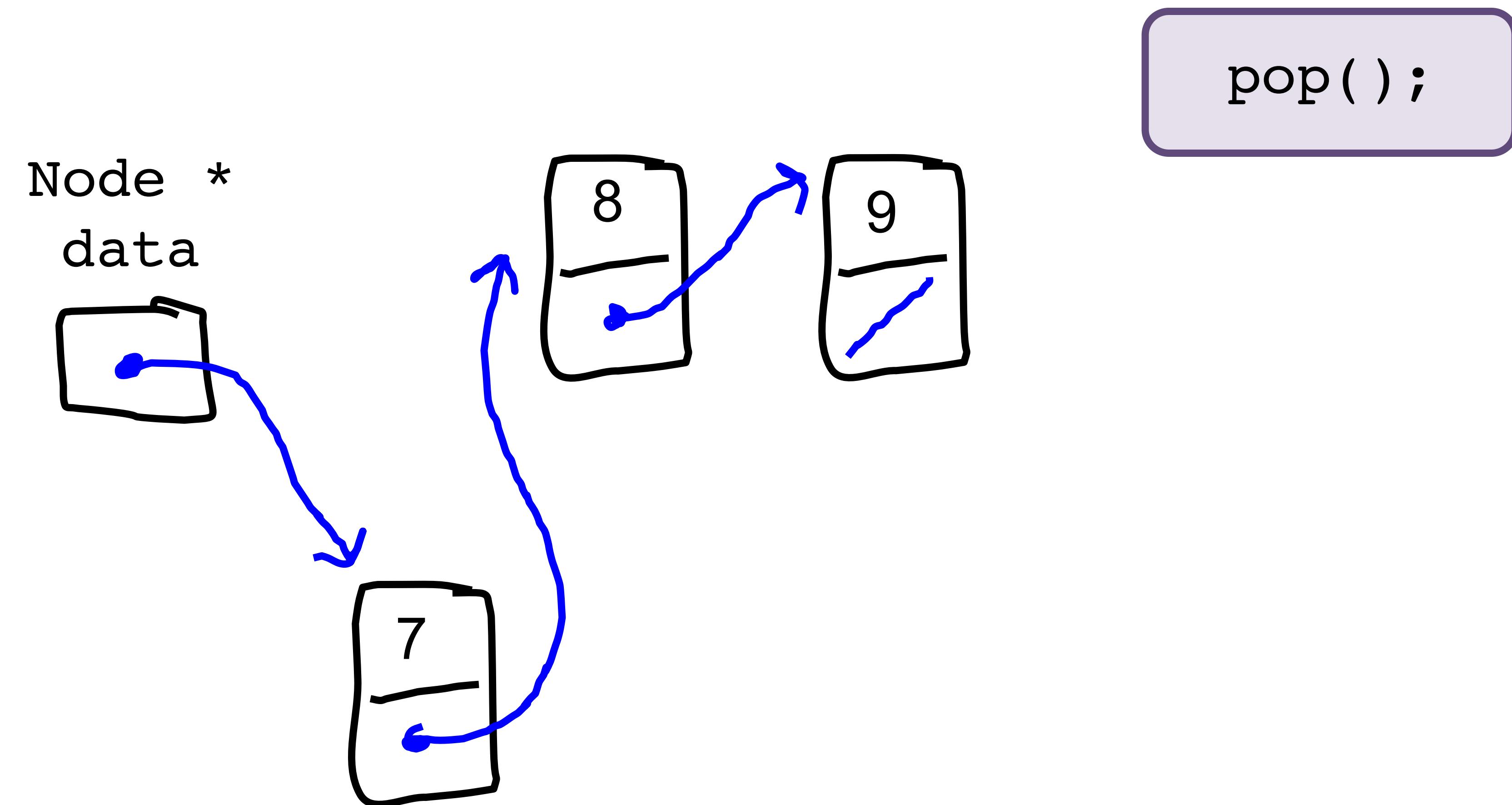


Stack is a Linked List

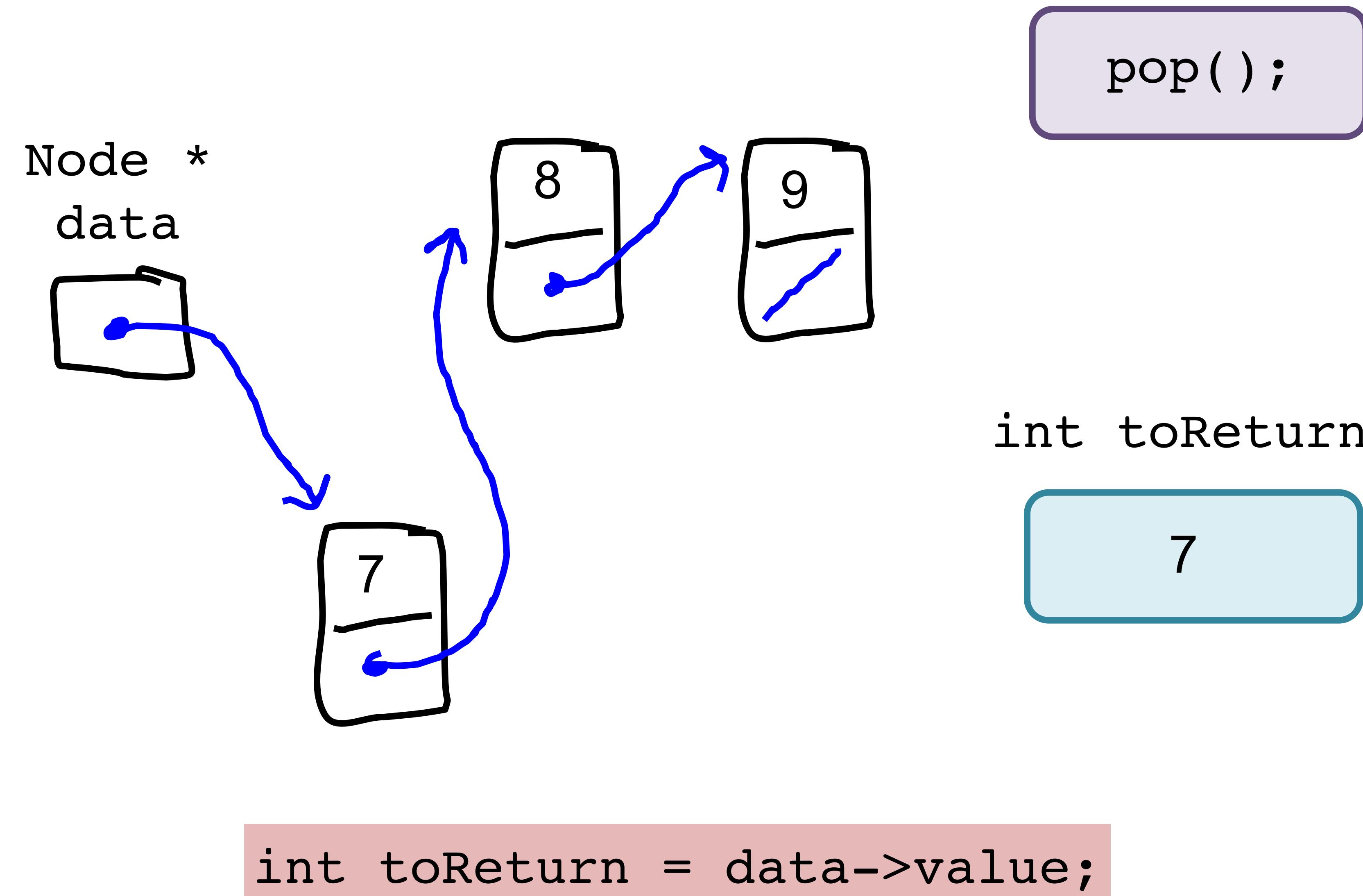


That didn't work. Let's try again...

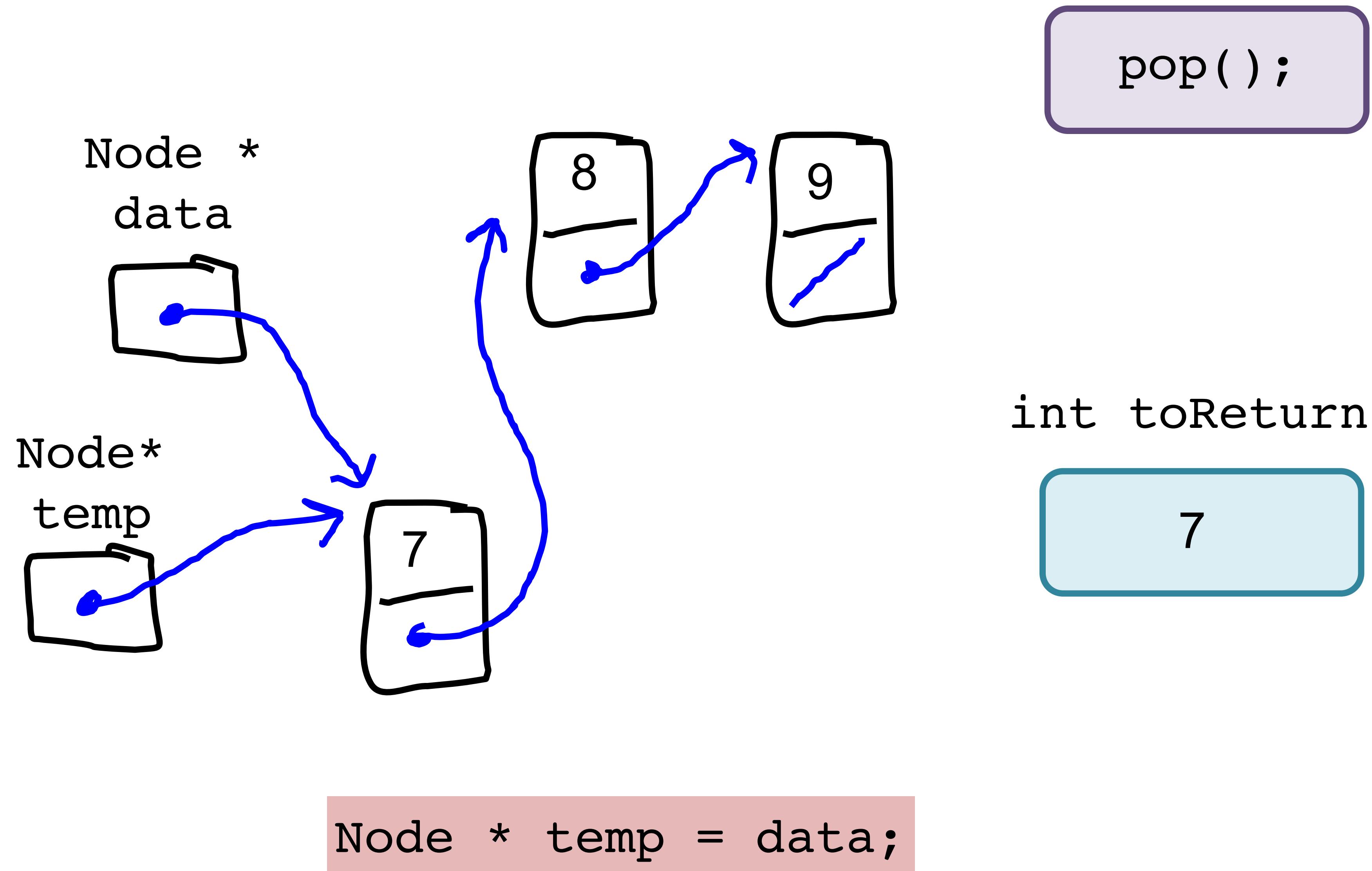
Stack is a Linked List



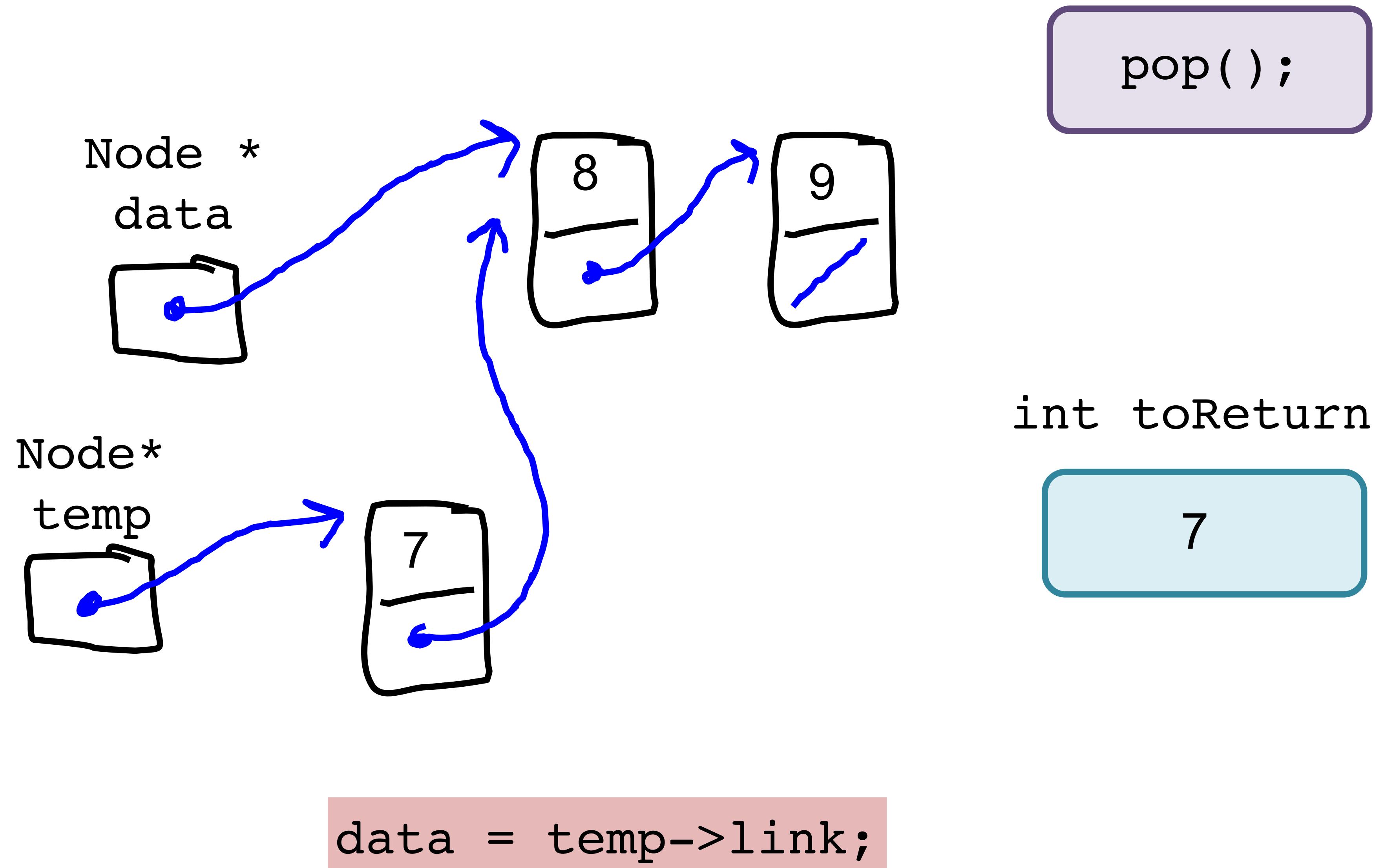
Stack is a Linked List



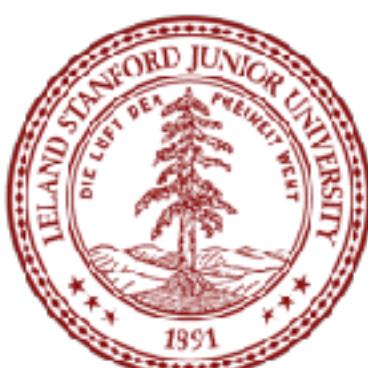
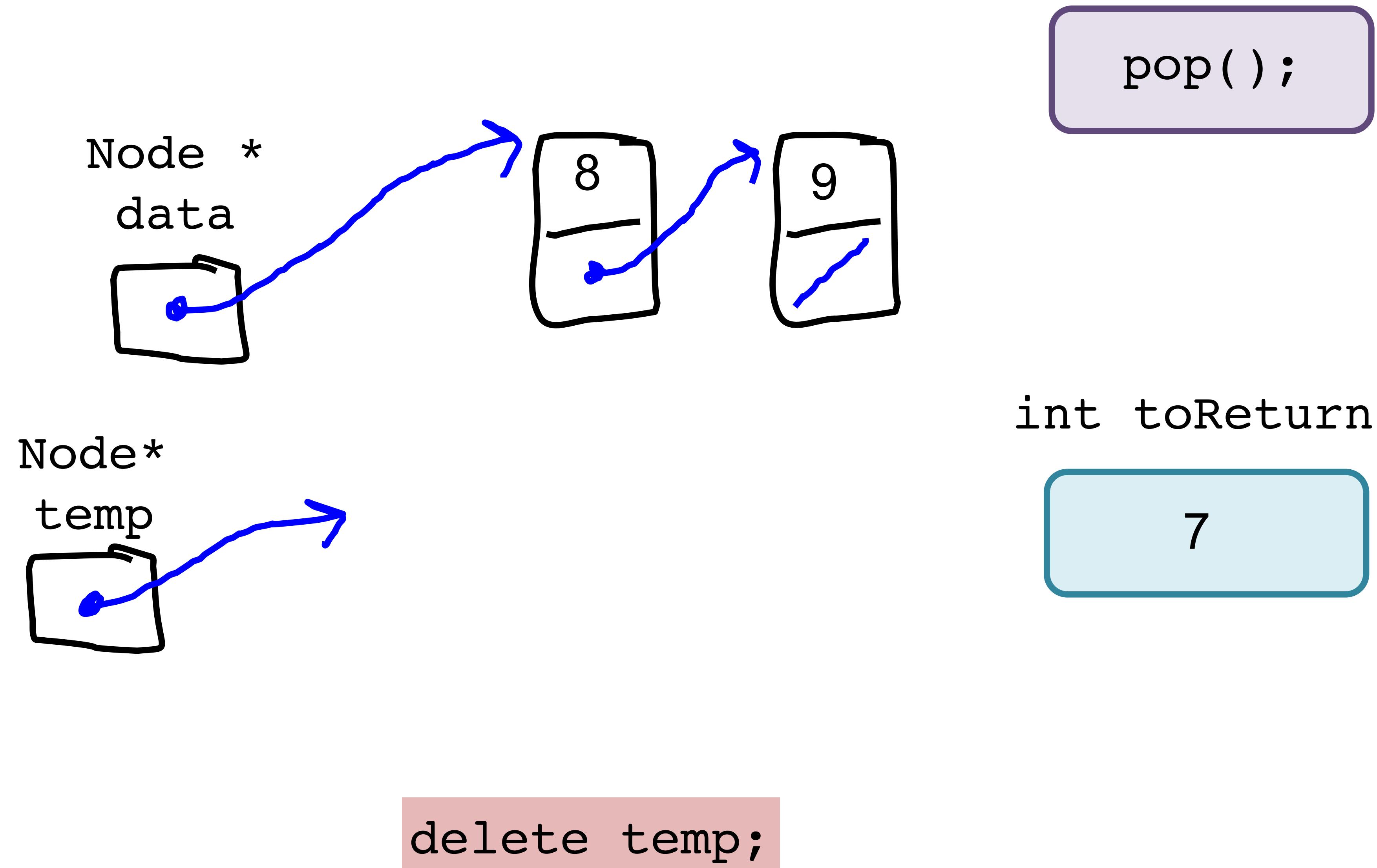
Stack is a Linked List



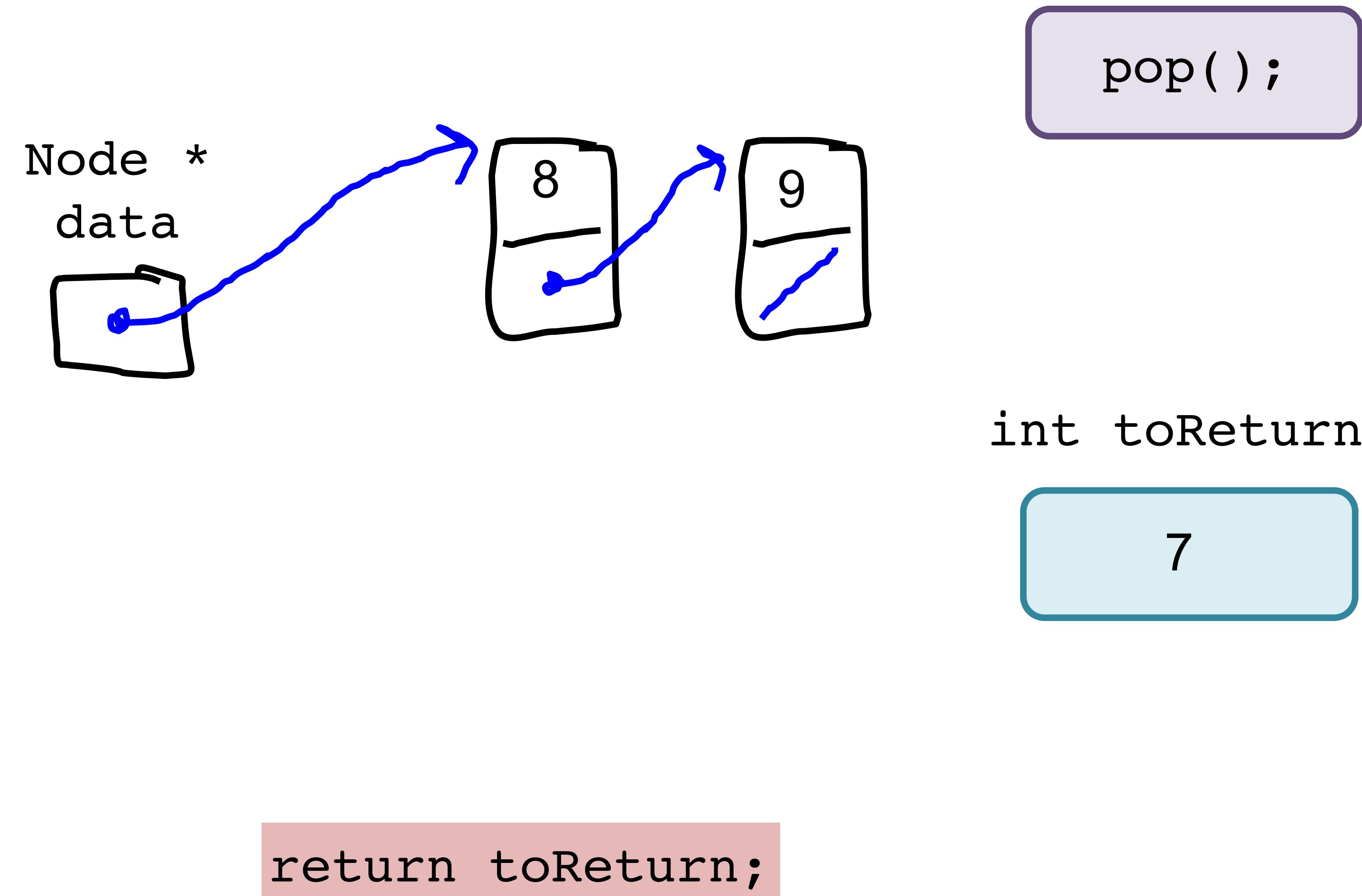
Stack is a Linked List



Stack is a Linked List



Stack is a Linked List



Stack

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

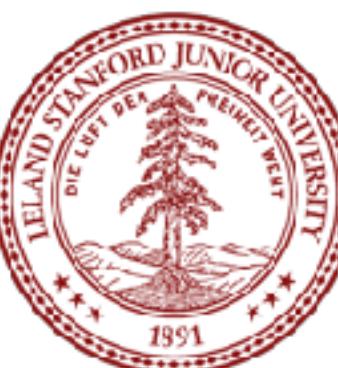
    void push(value);      // append a value
    int pop();            // return the first-in value

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;  
}
```

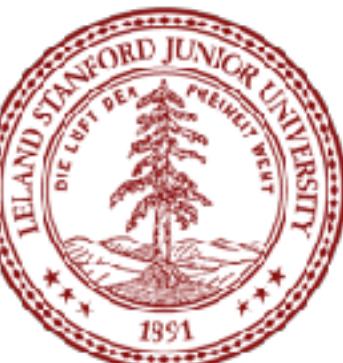


Stack Implementation: Big O?

Big O of push()? O(1)

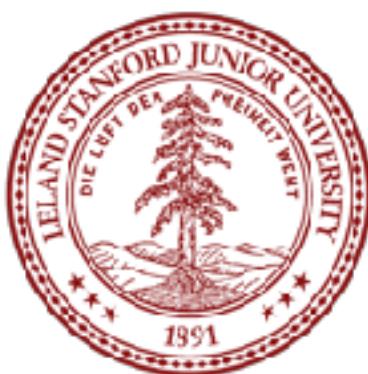
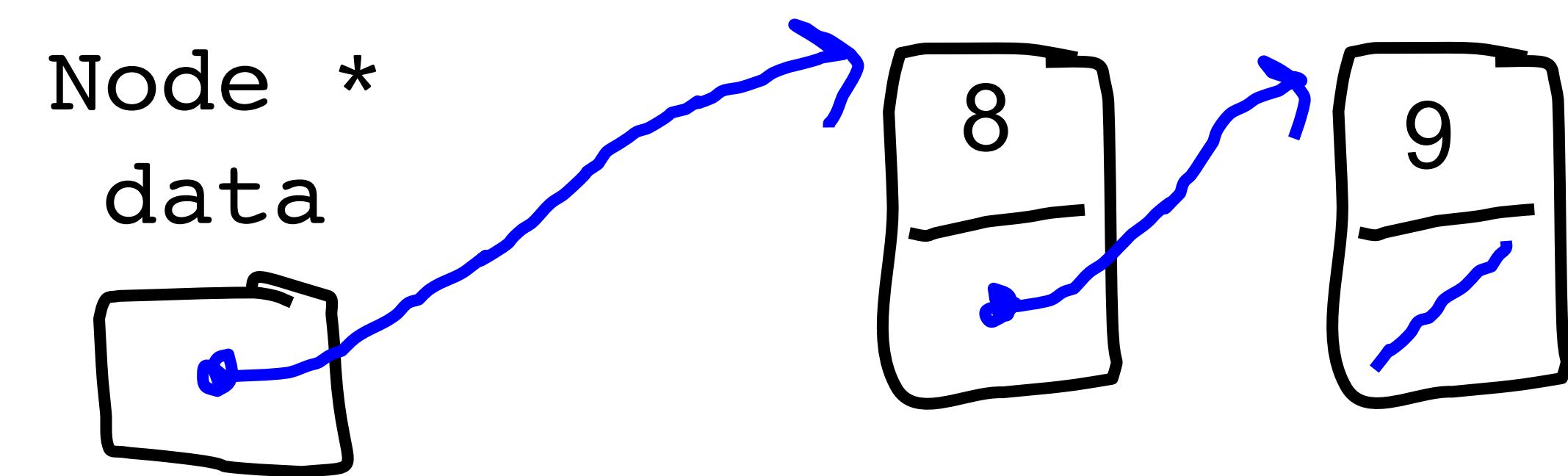
Big O of pop()? O(1)

Yay!

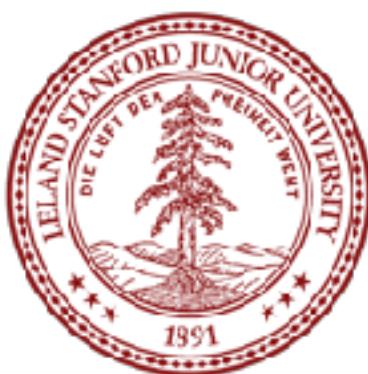
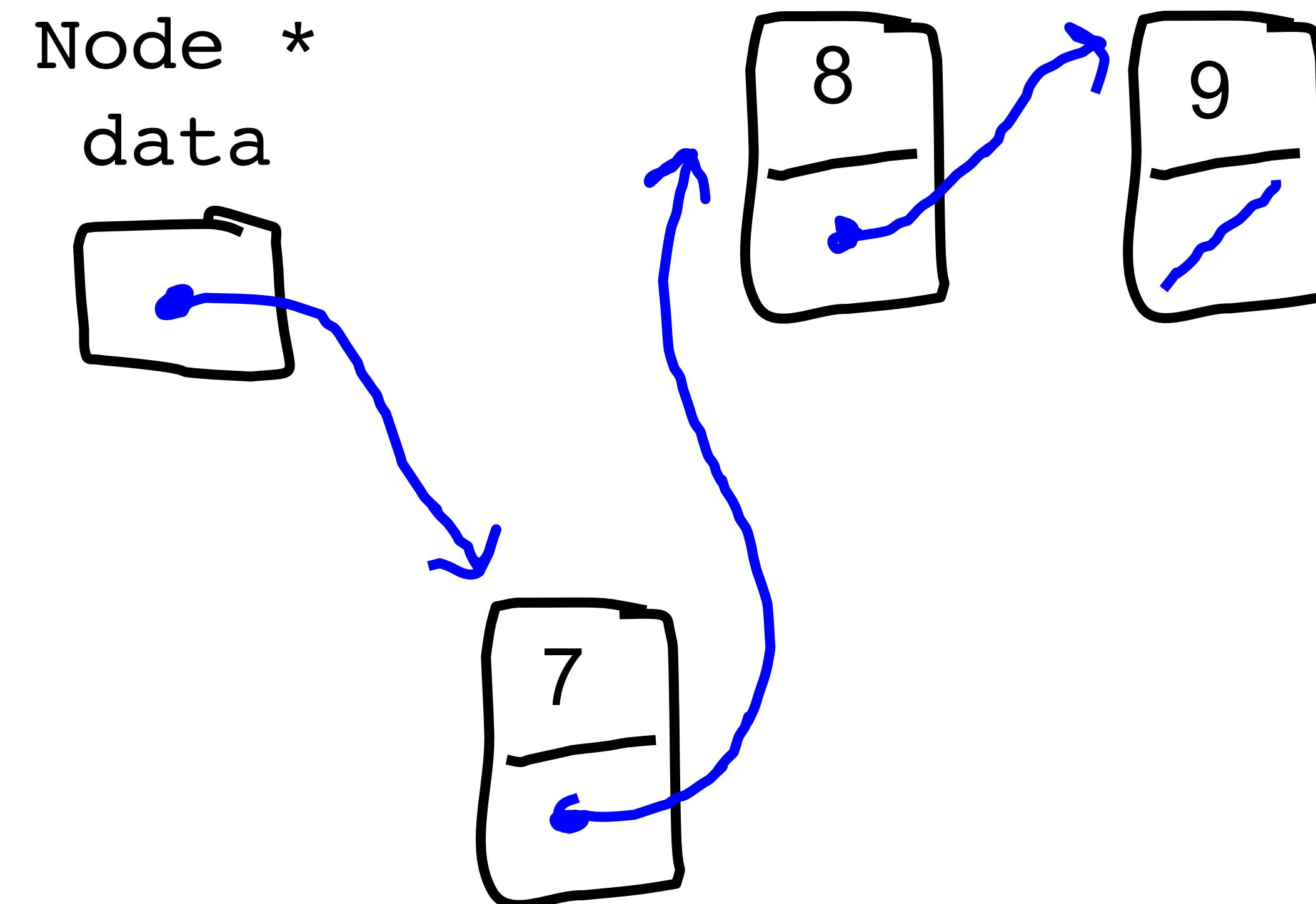




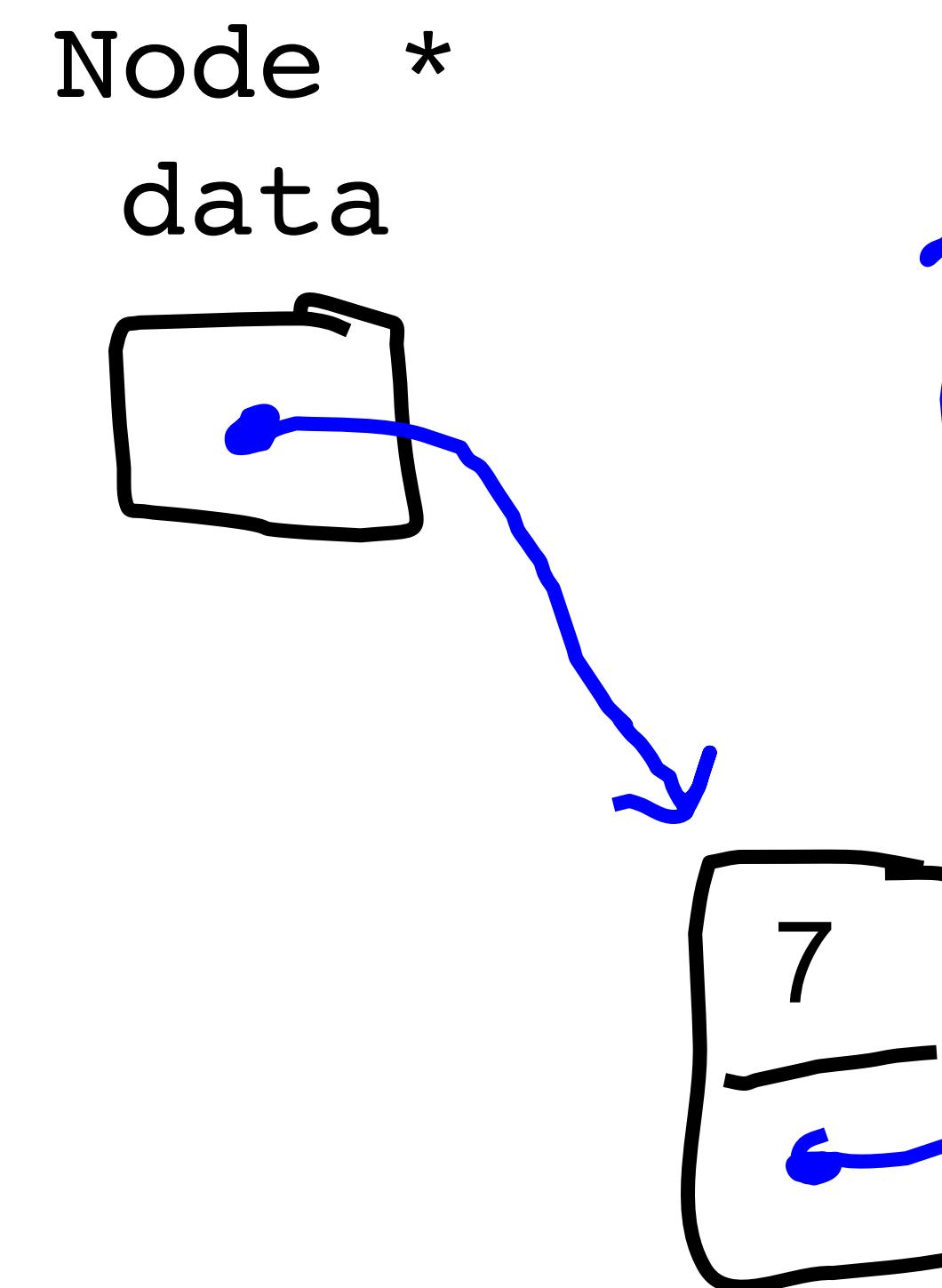
Queue?



Queue Enqueue?



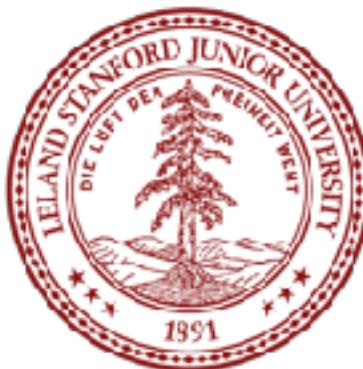
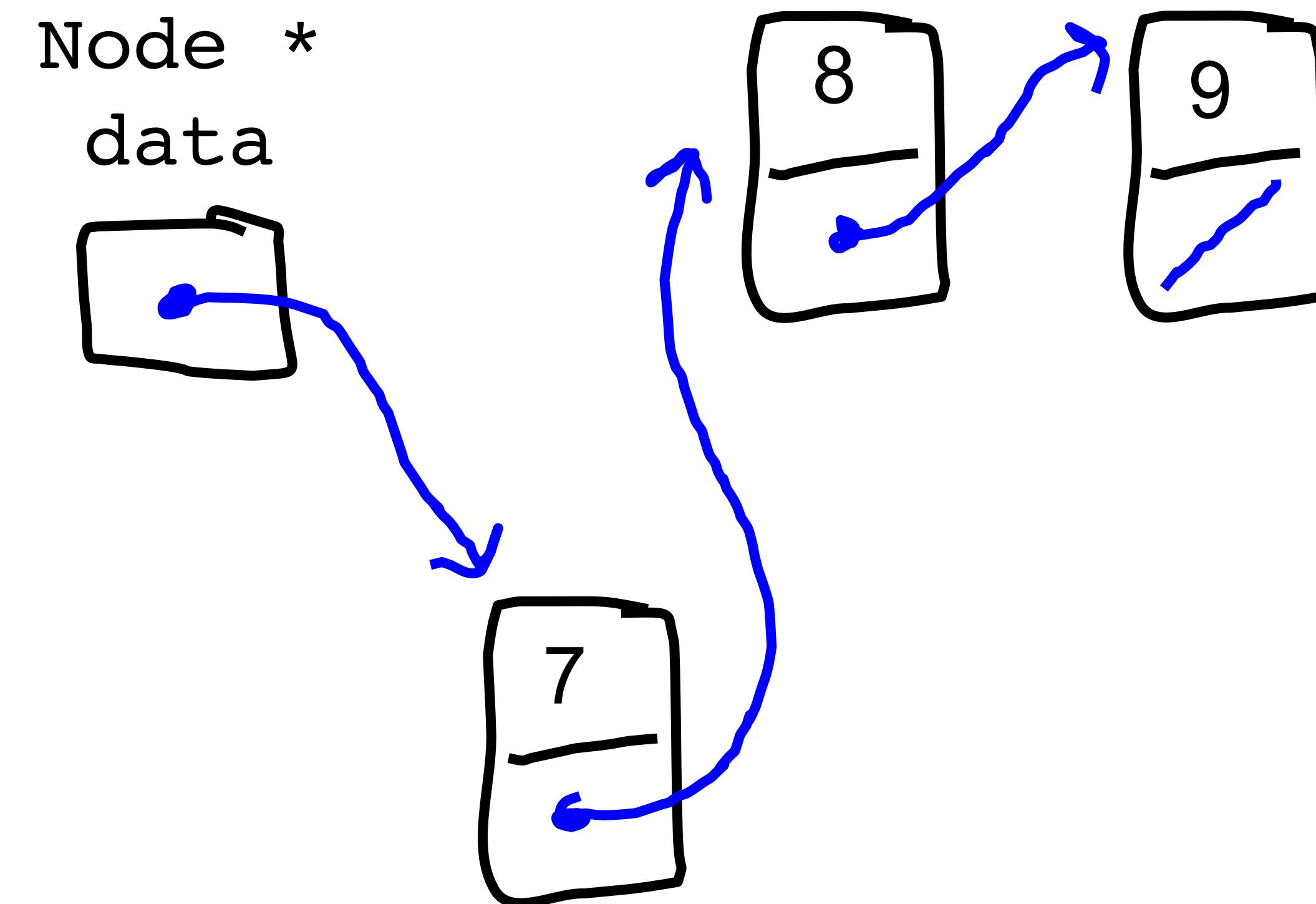
Queue Enqueue?



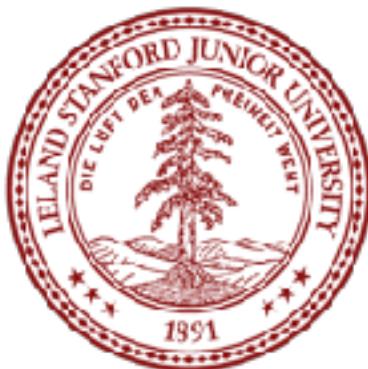
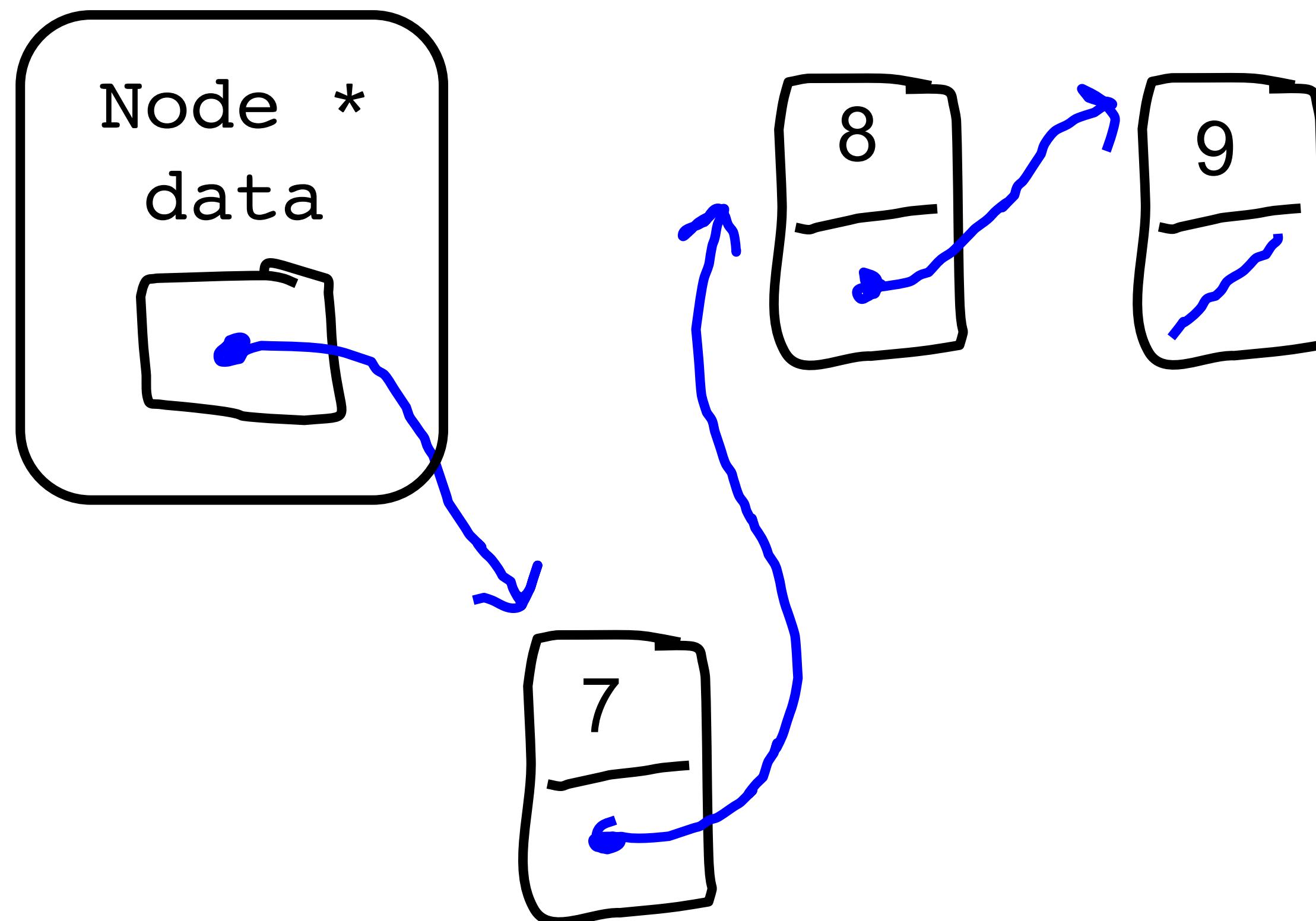
O(1)



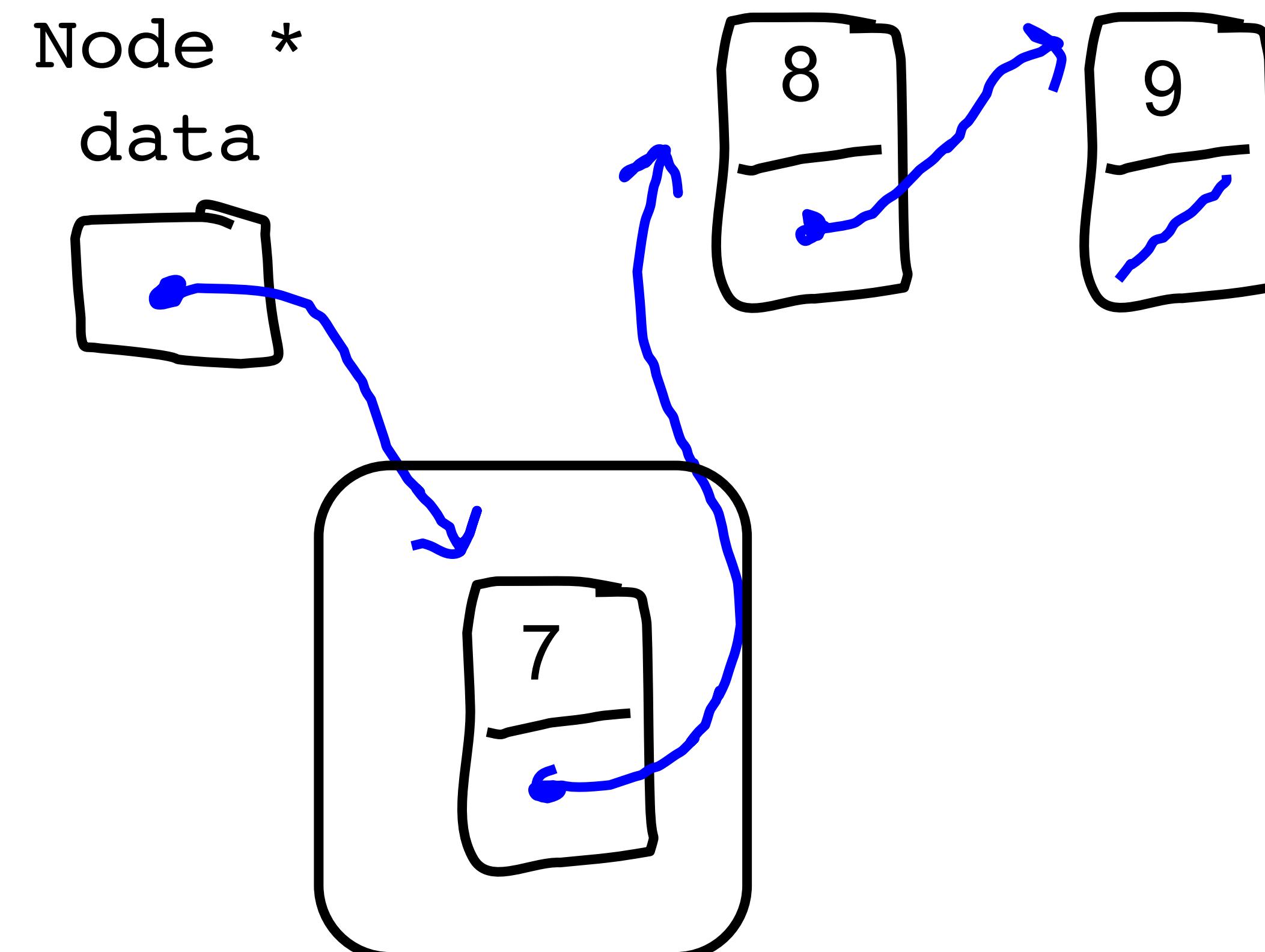
Queue Dequeue?



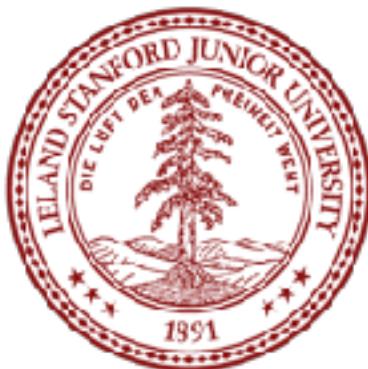
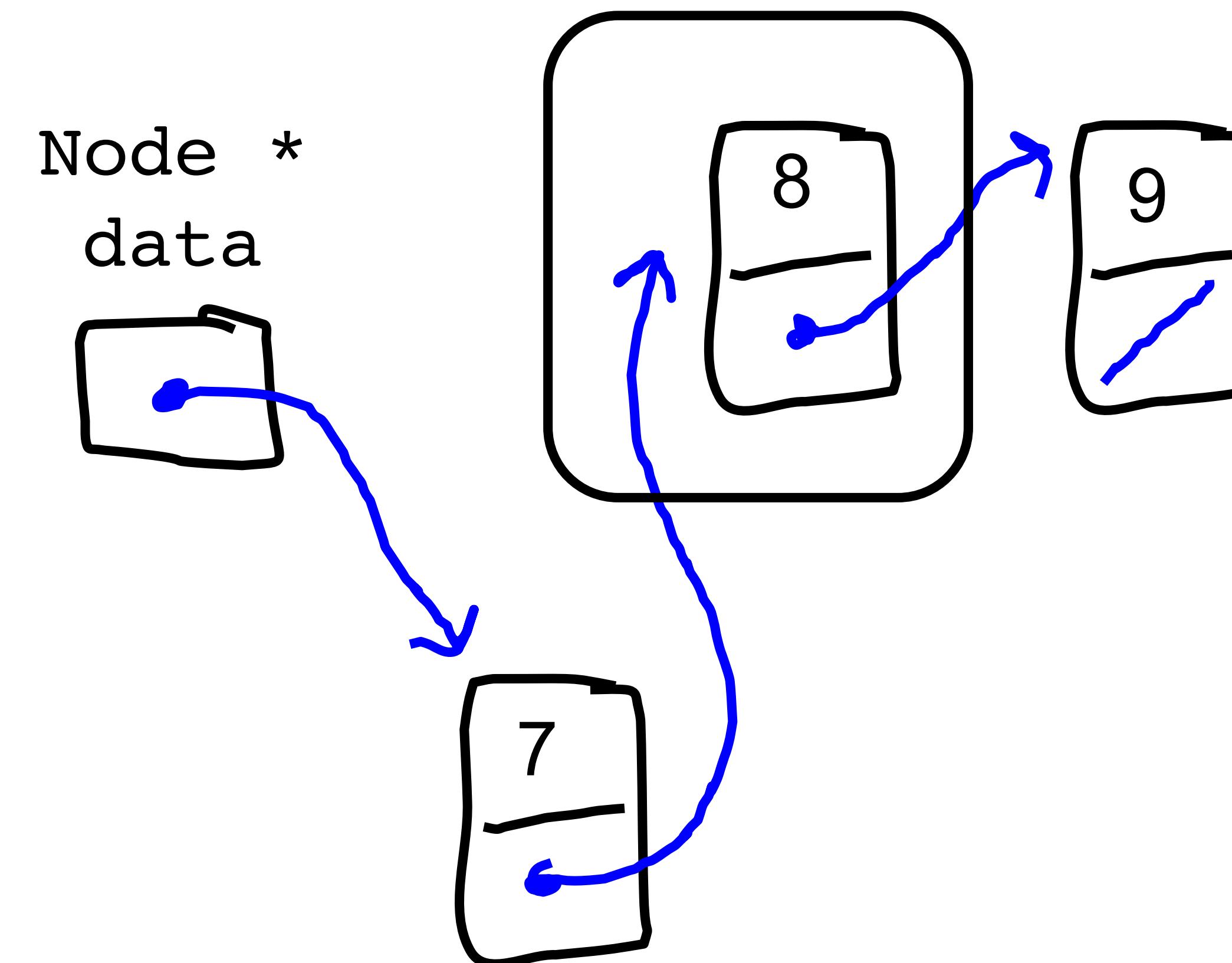
Queue Dequeue?



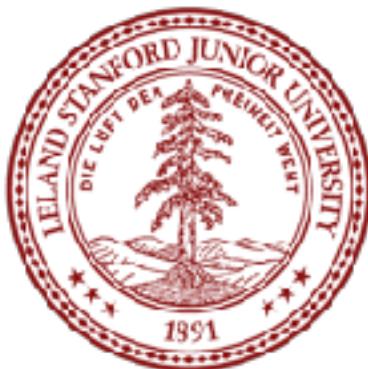
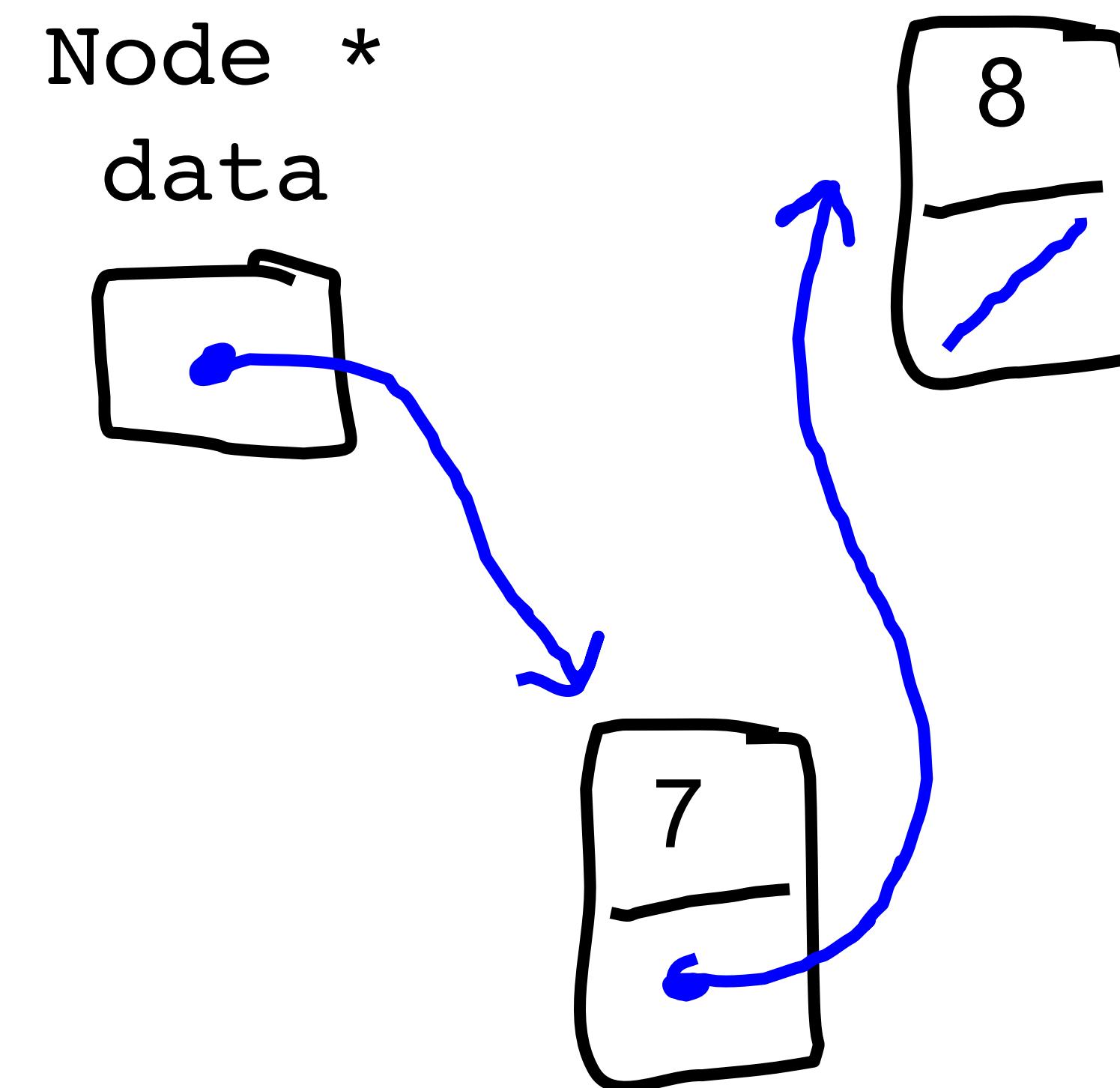
Queue Dequeue?



Queue Dequeue?



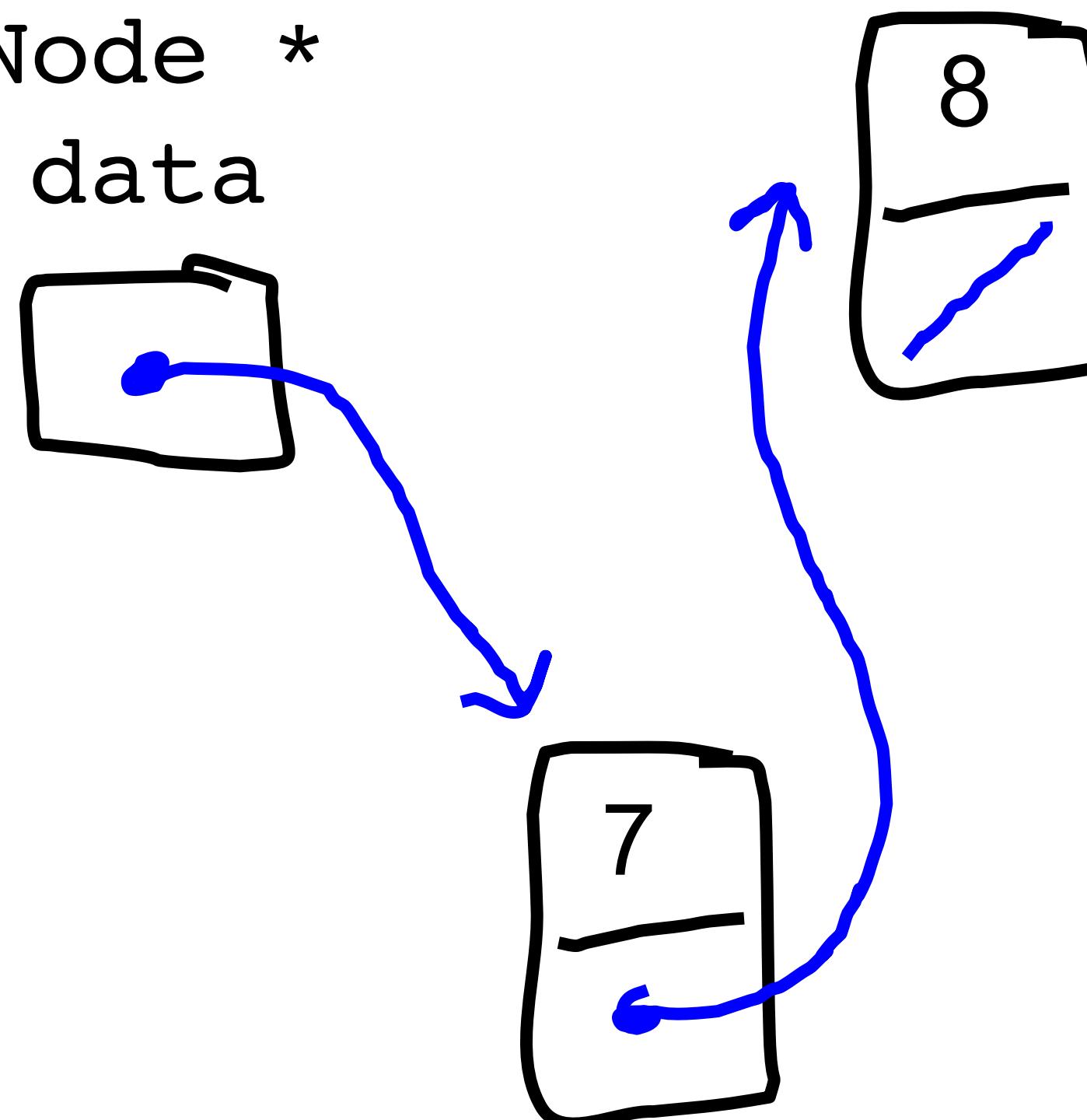
Queue Dequeue?



Queue Dequeue?

Node *

data

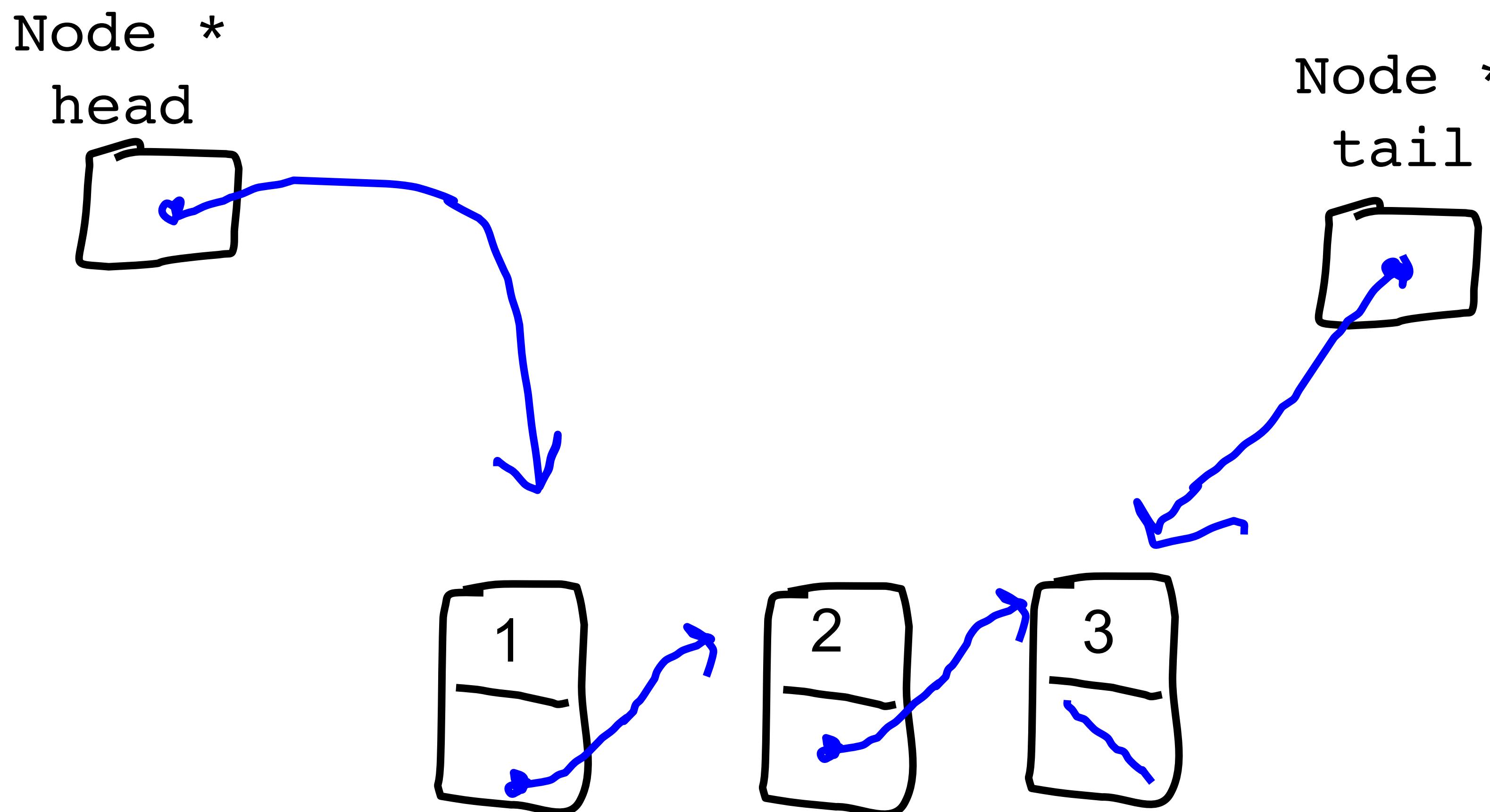


$O(n)$

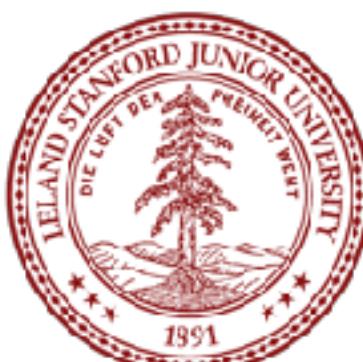
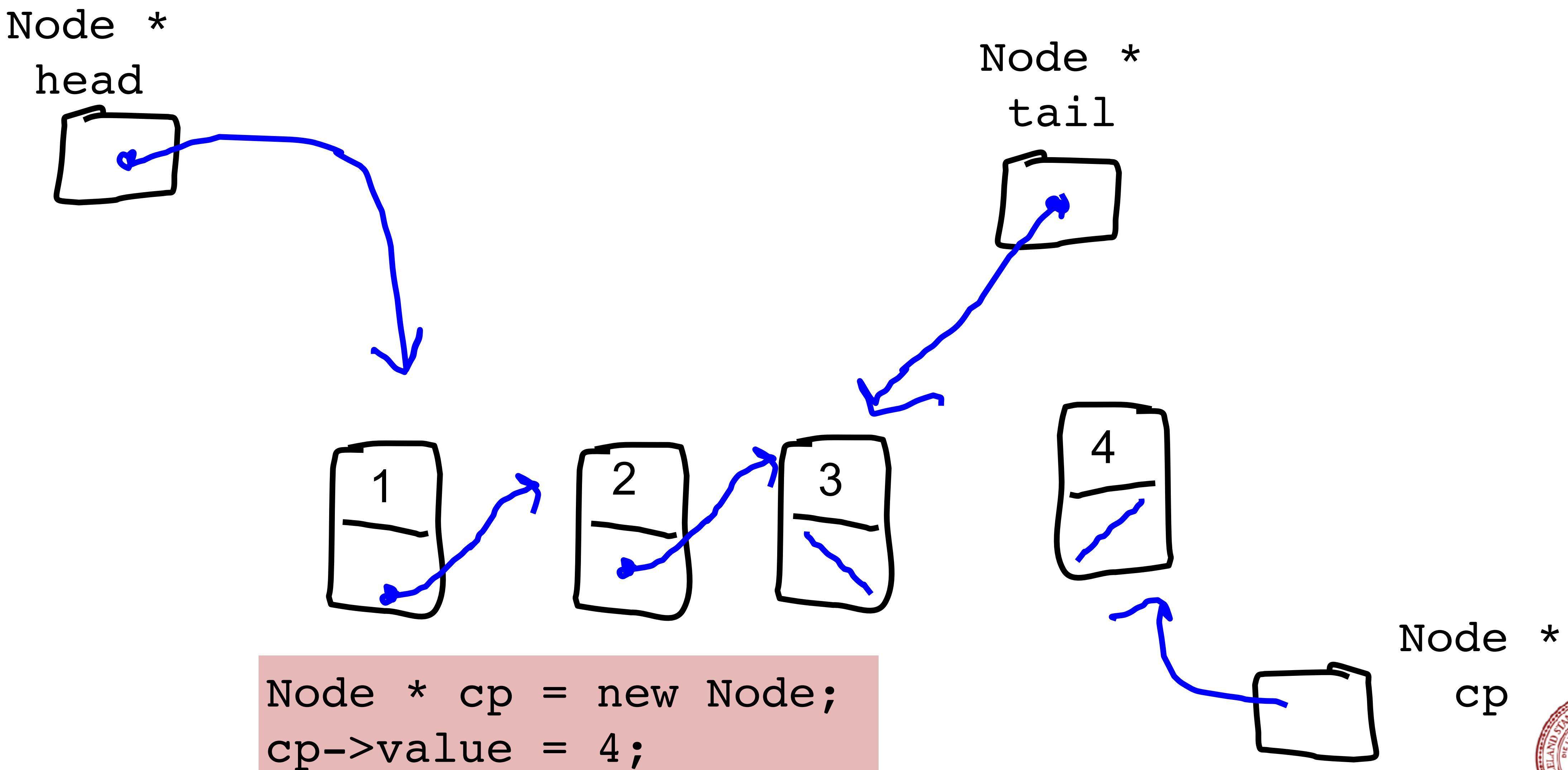


Always a Better Way

Actual Queue: Enqueue

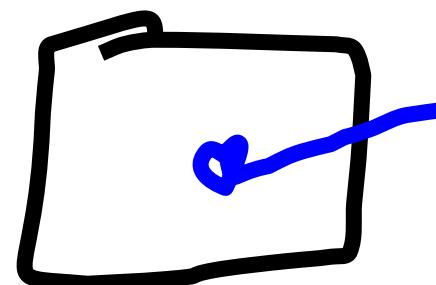


Actual Queue: Enqueue

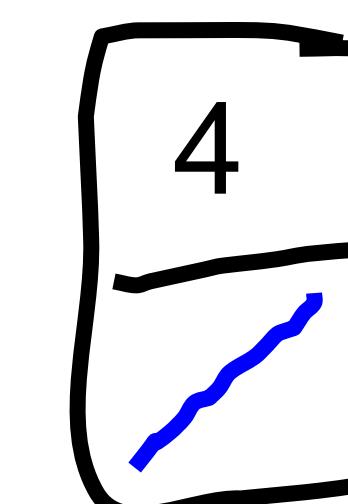
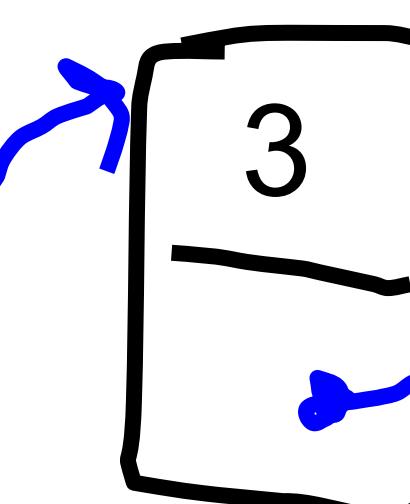
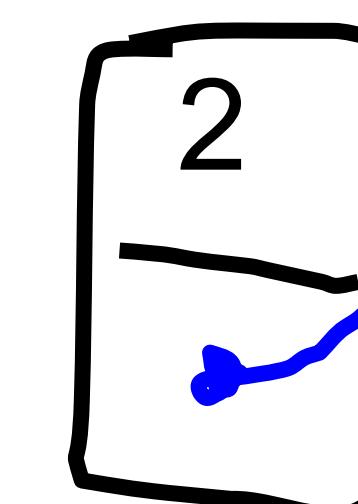
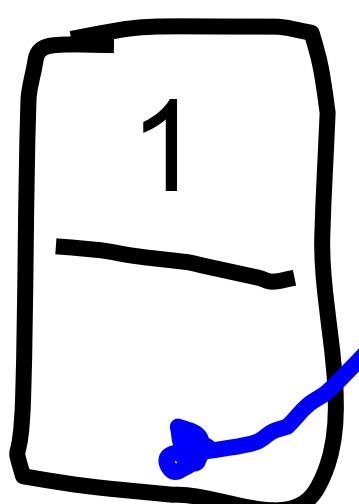
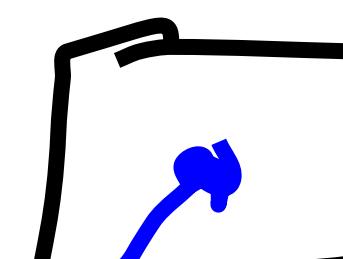


Actual Queue: Enqueue

Node *
head

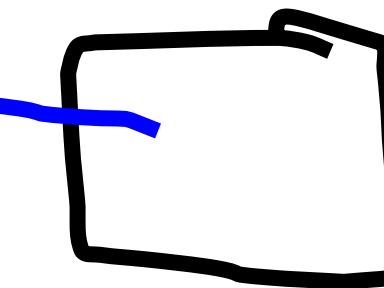


Node *
tail



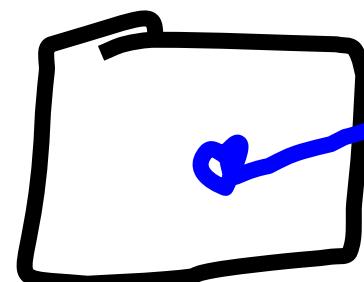
tail->link = cp;

Node *
cp

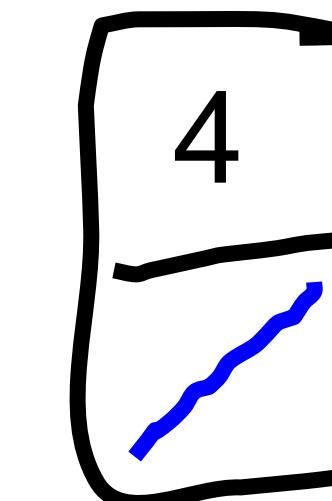
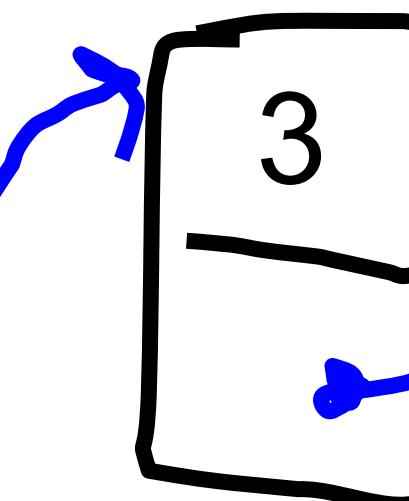
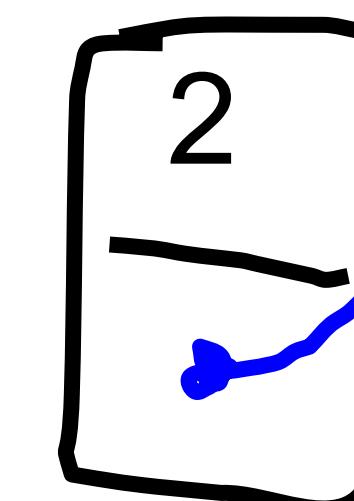
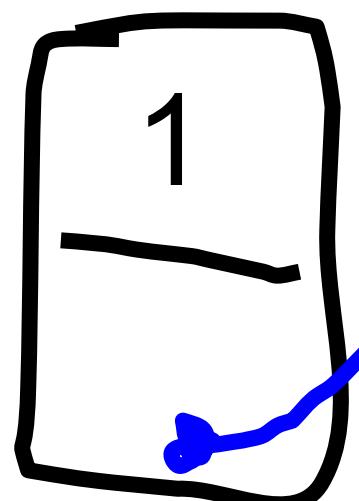
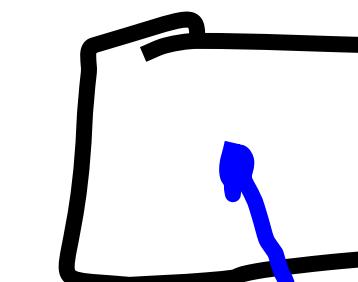


Actual Queue: Enqueue

Node *
head

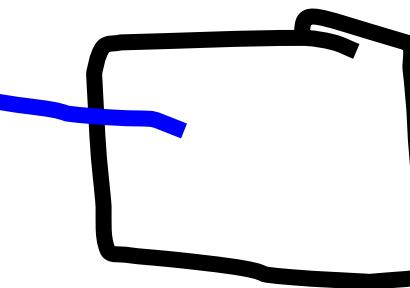


Node *
tail

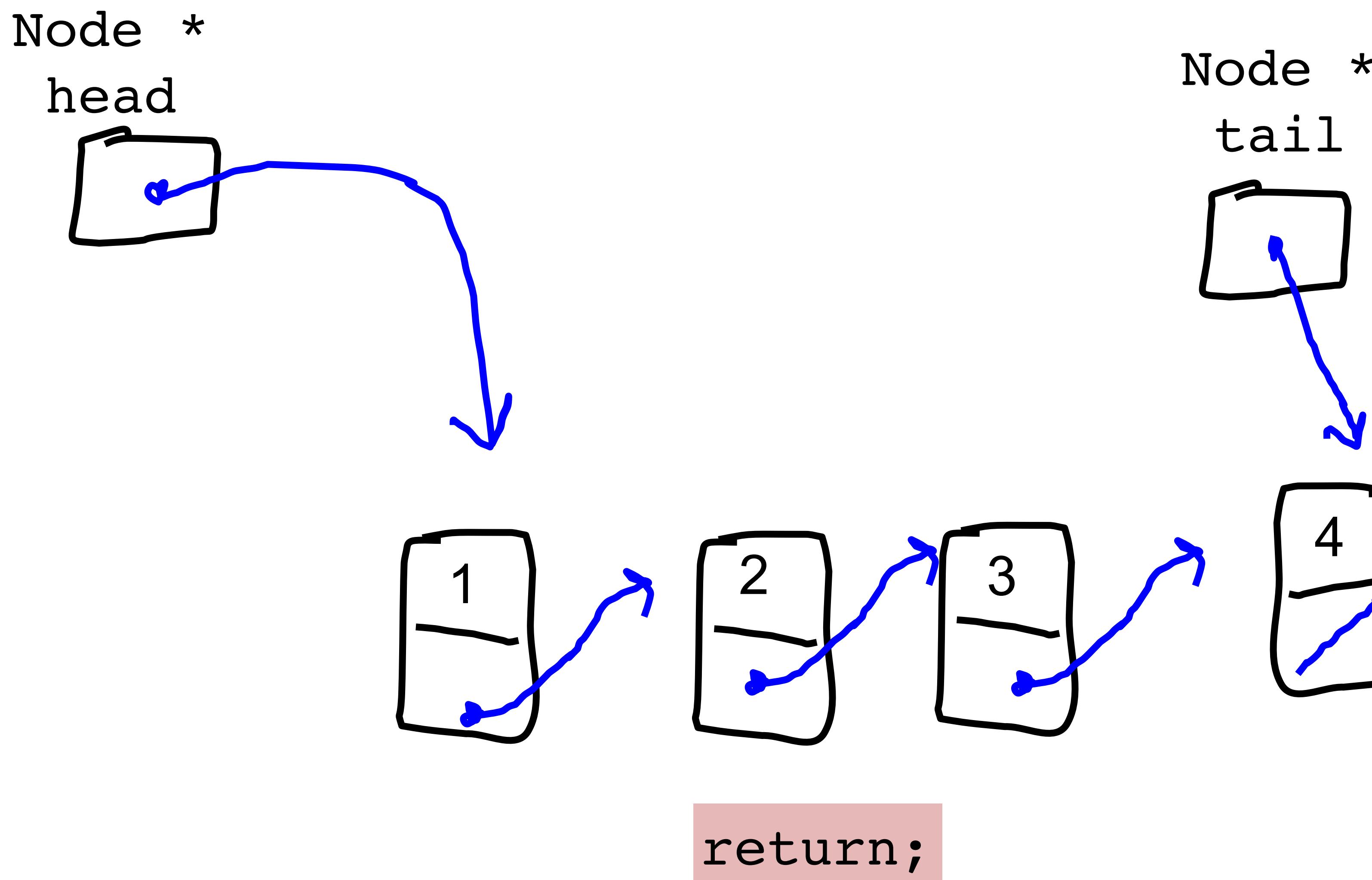


tail = cp;

Node *
cp

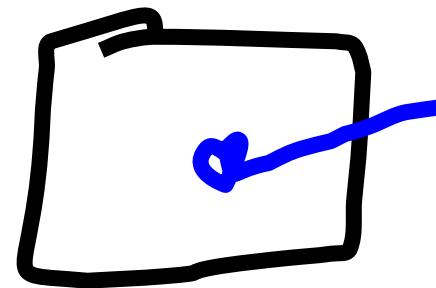


Actual Queue: Enqueue

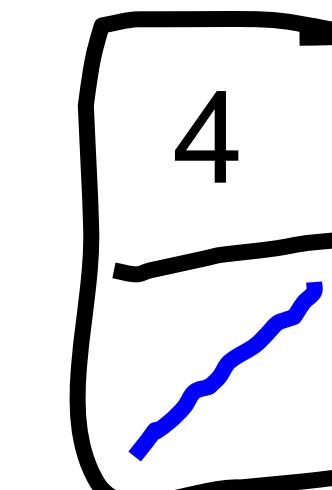
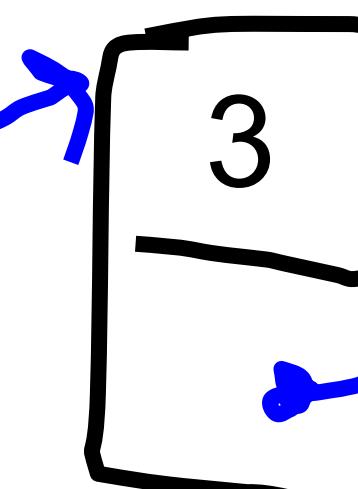
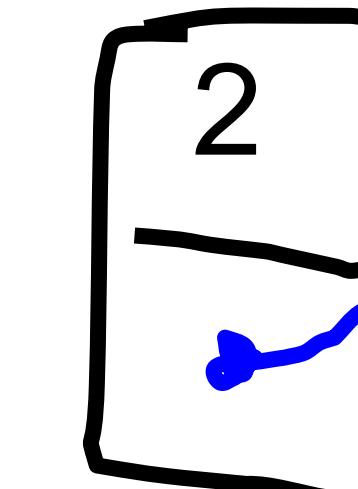
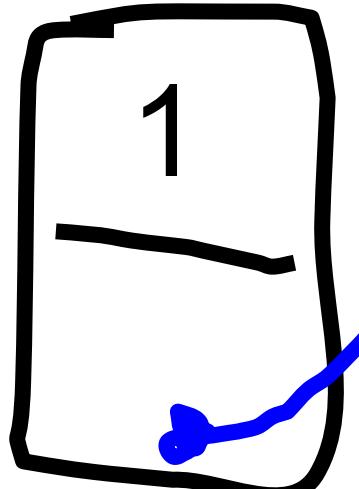
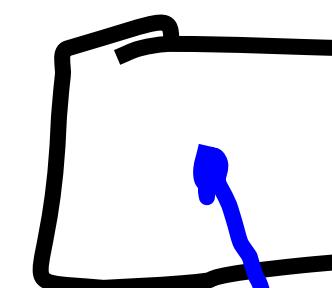


Actual Queue: Enqueue

Node *
head



Node *
tail

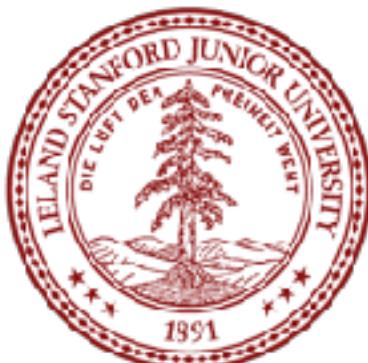
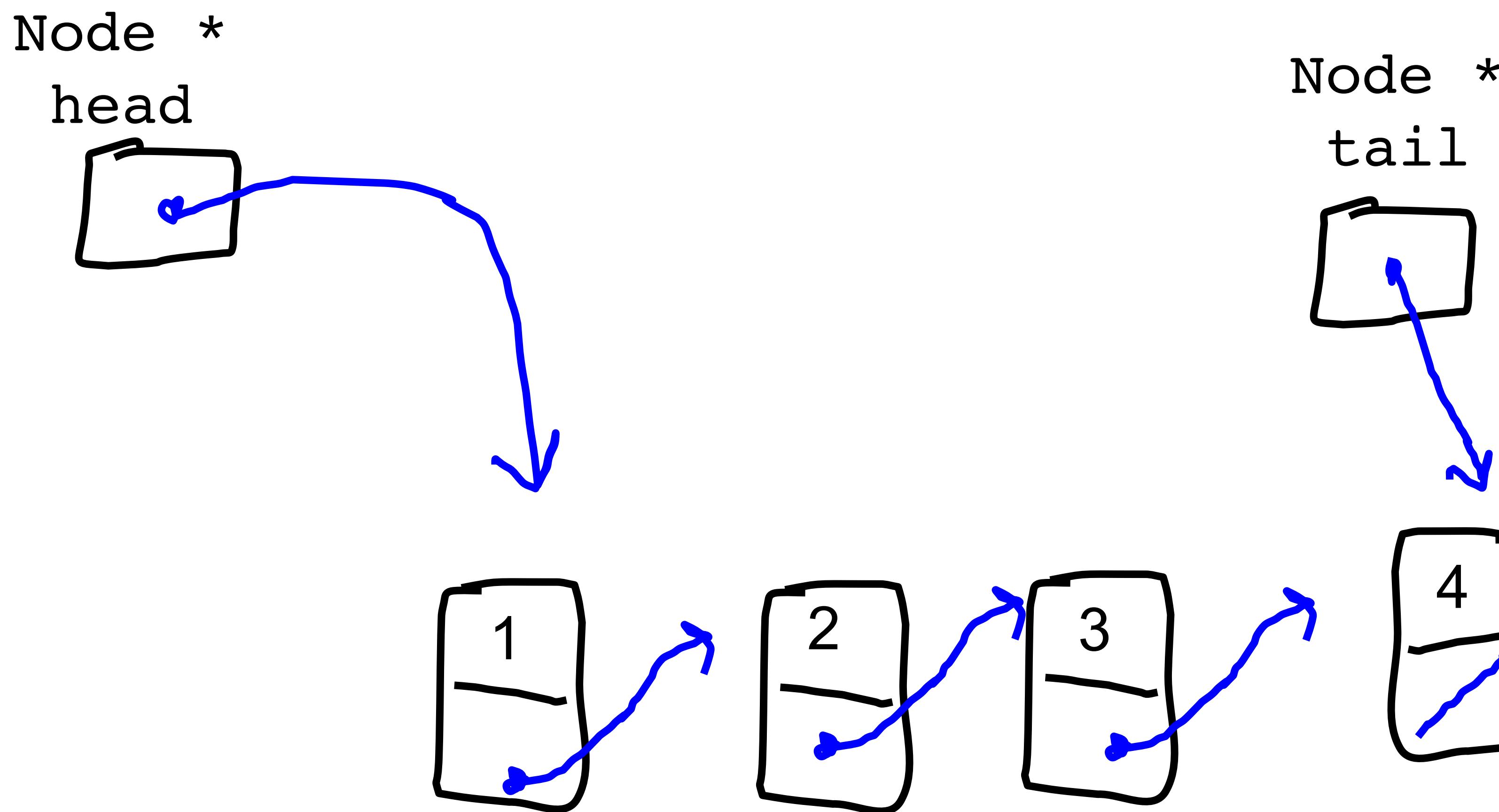


O(1)

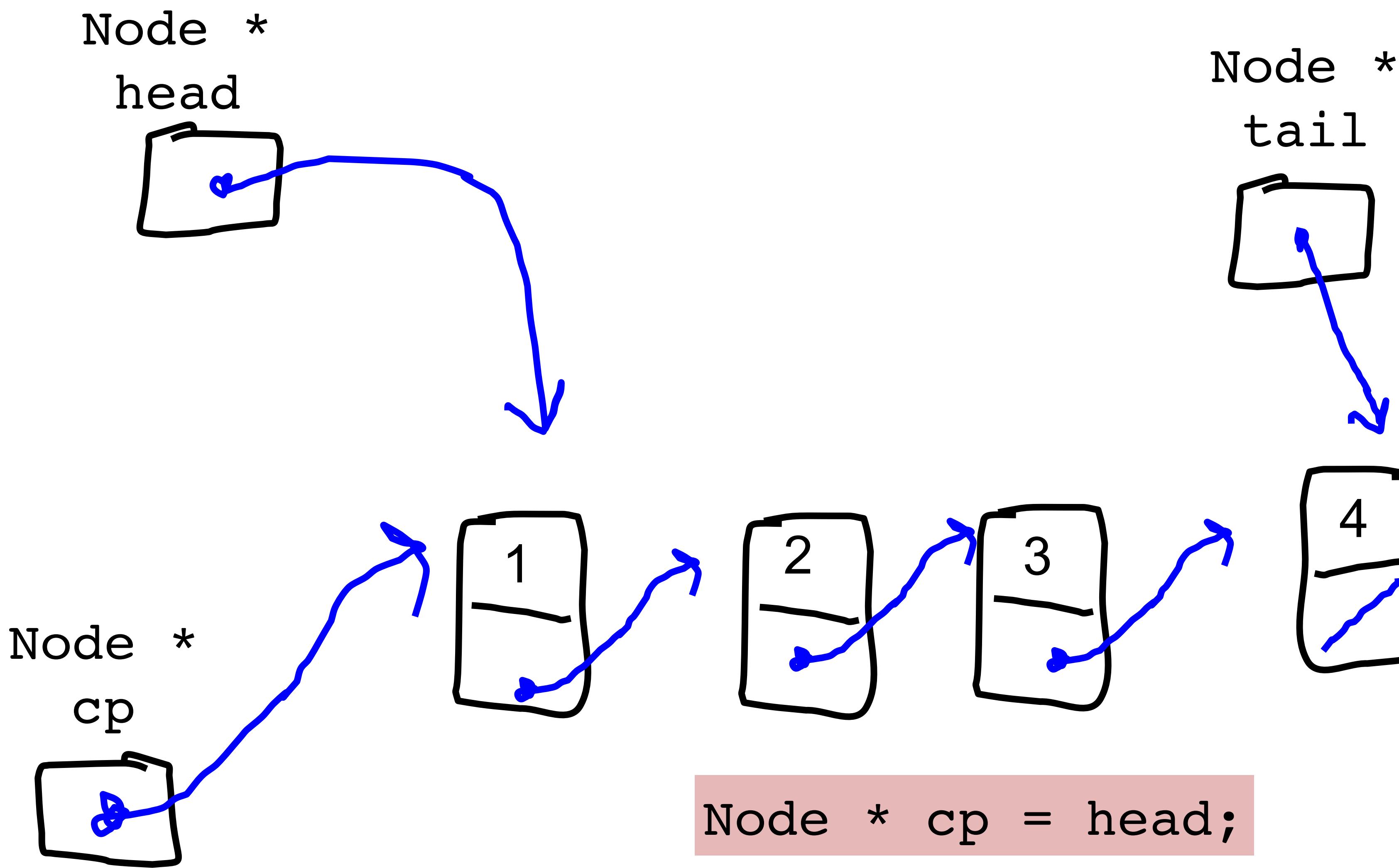


Dequeue

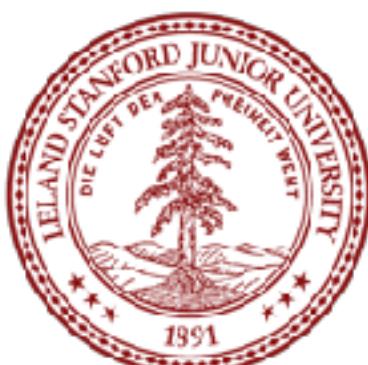
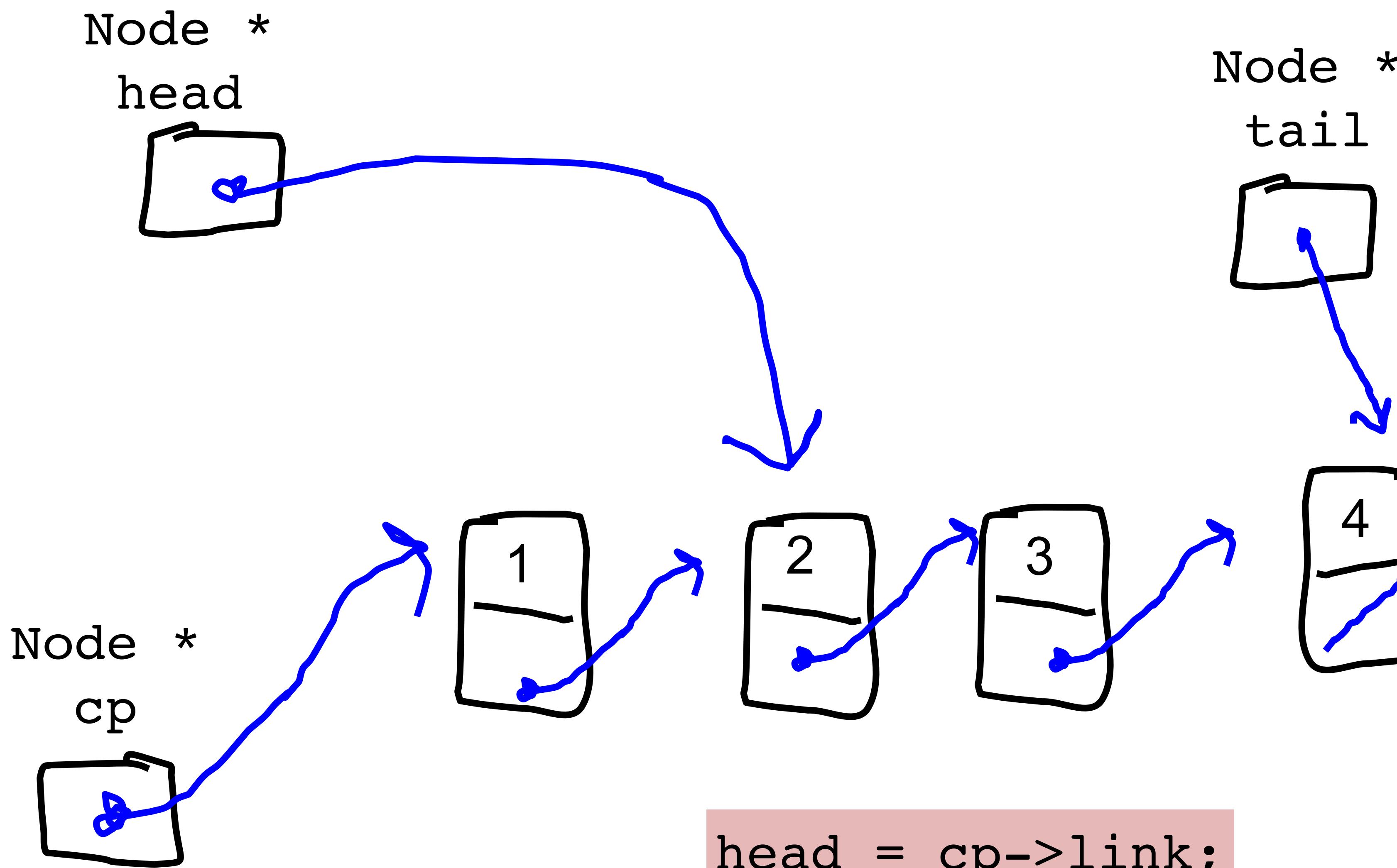
Actual Queue: Dequeue



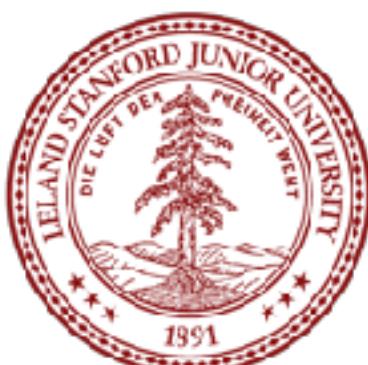
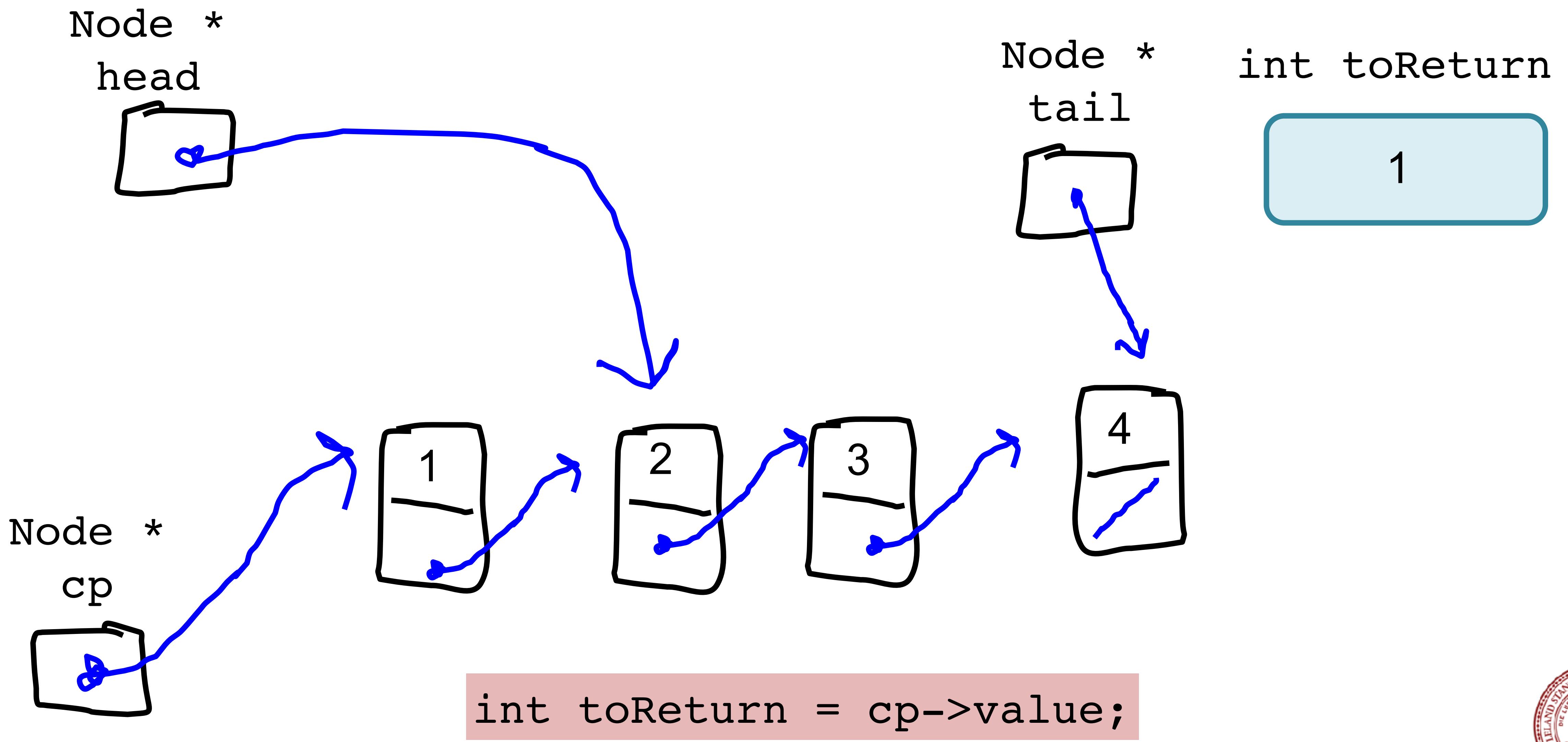
Actual Queue: Dequeue



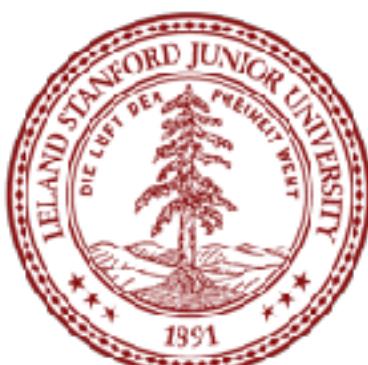
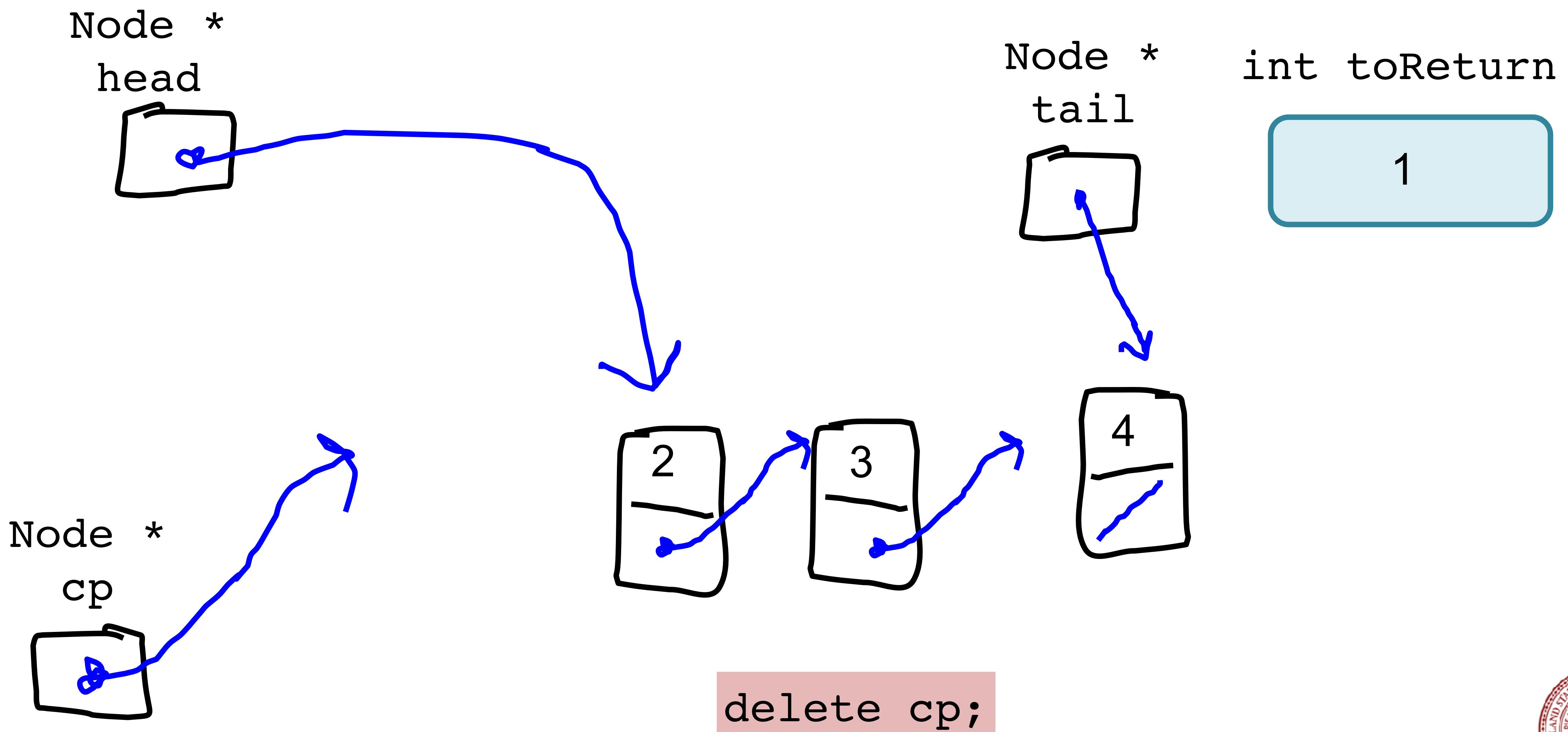
Actual Queue: Dequeue



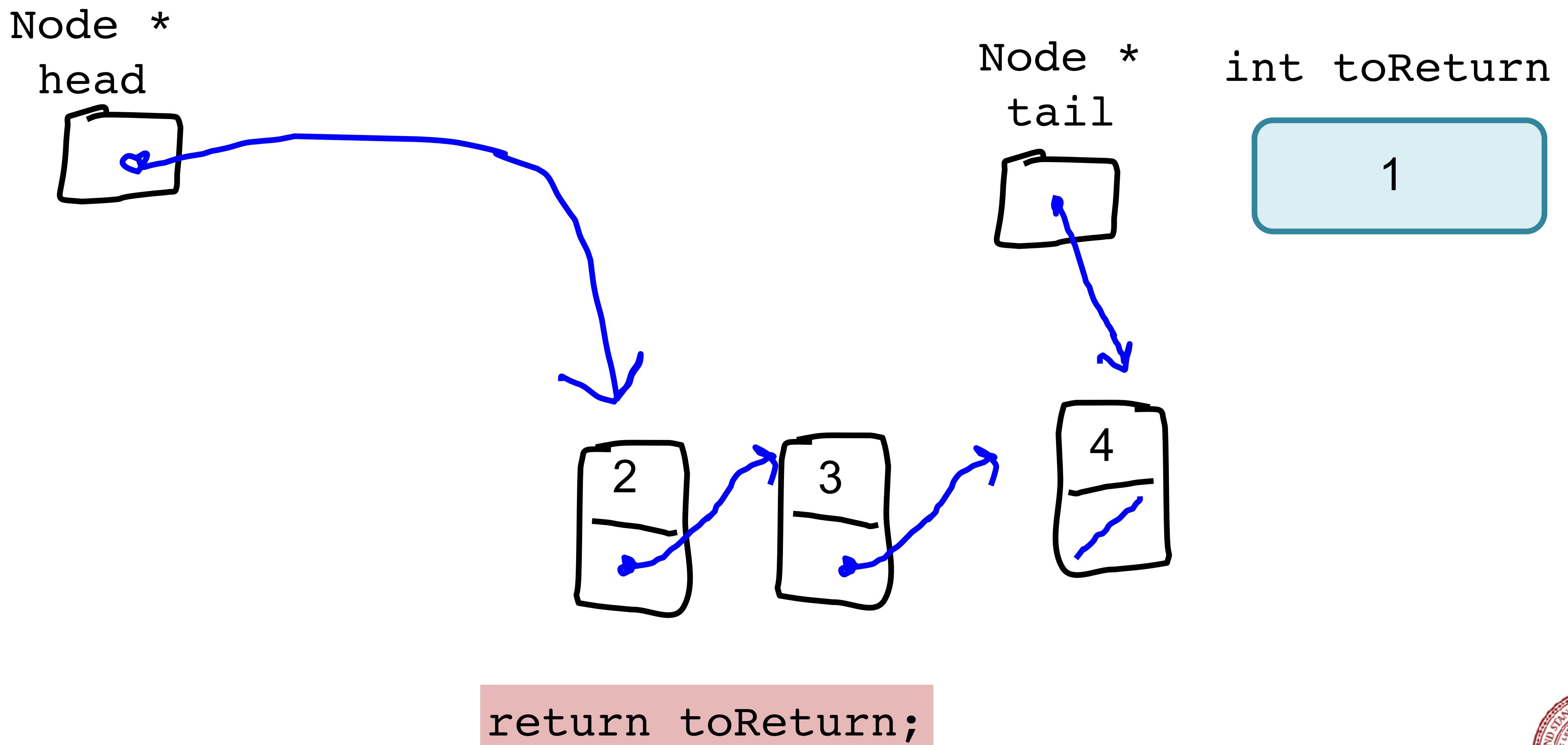
Actual Queue: Dequeue



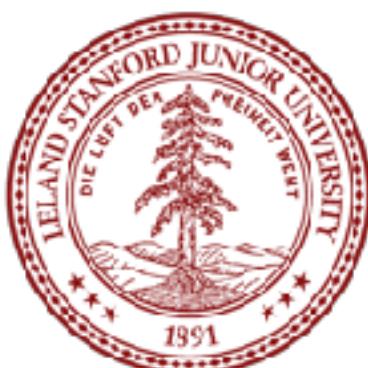
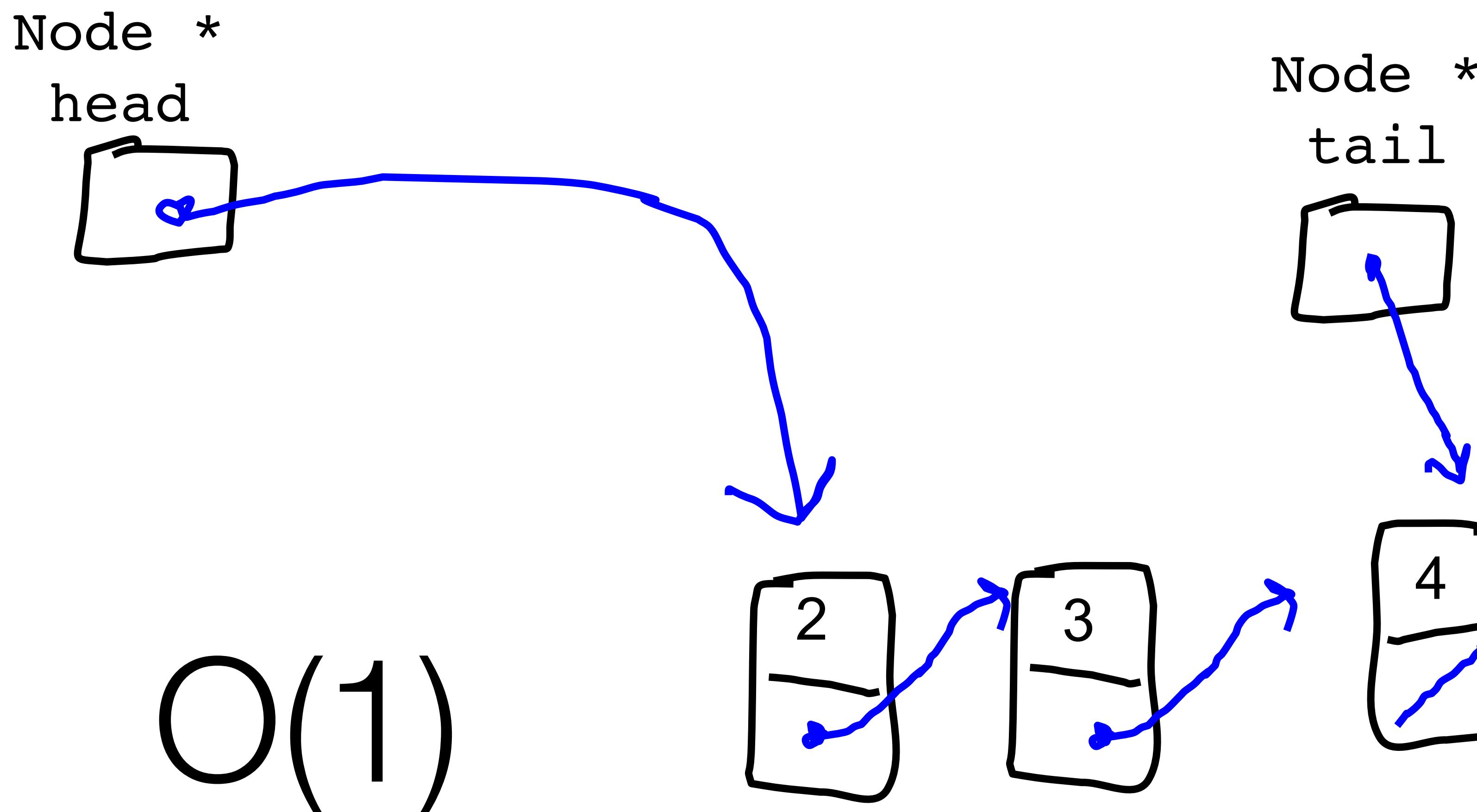
Actual Queue: Dequeue



Actual Queue: Dequeue



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

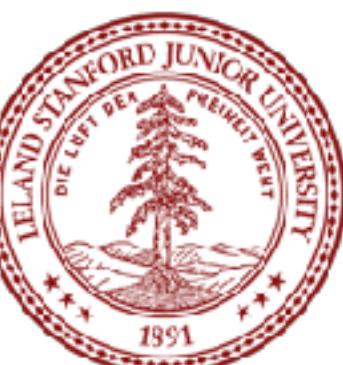
private:
    struct Node {
        int value;
        Node * link;
    };
    Node * head;            // has a pointer to the first node
    Node * tail;            // and a pointer to the last node
};
```



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

$$\mathcal{O}(1)$$
