

CS 106B

Lecture 16: Linked Lists

Monday, May 7, 2018

Programming Abstractions
Spring 2018
Stanford University
Computer Science Department

Lecturer: Chris Gregg

reading:
Programming Abstractions in C++, Chapter 11



(Linked Wrists ↑)



Today's Topics

- Logistics
 - Midterm graded — regrade requests due by next Monday
 - You should have received an email to check Gradescope for your grade.
- 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



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

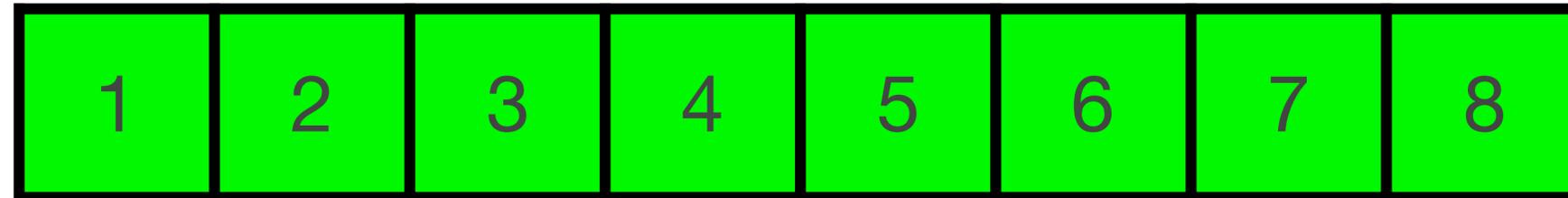
private:
    Vector<int> data; // member variables
};
```



You're next!

back

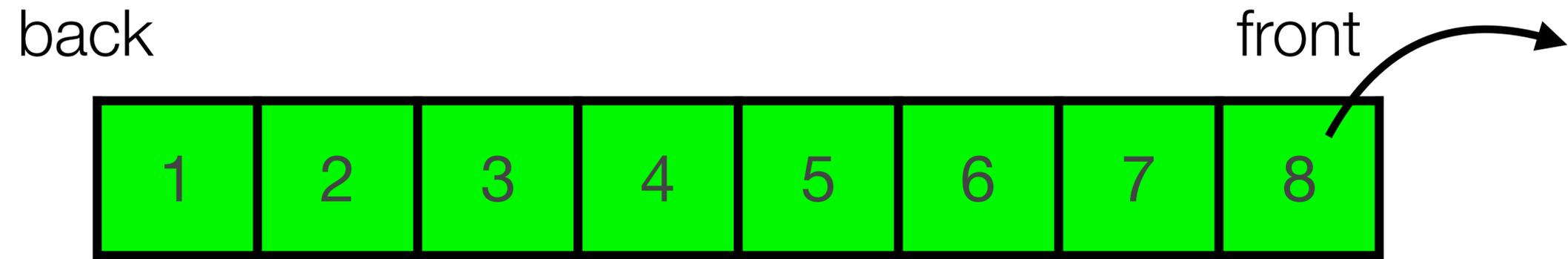
front



dequeue()



You're next!



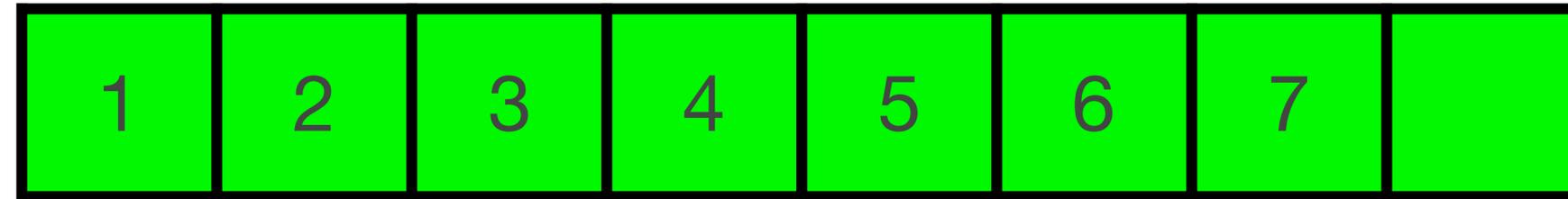
dequeue()



Excuse Me, Coming Through

back

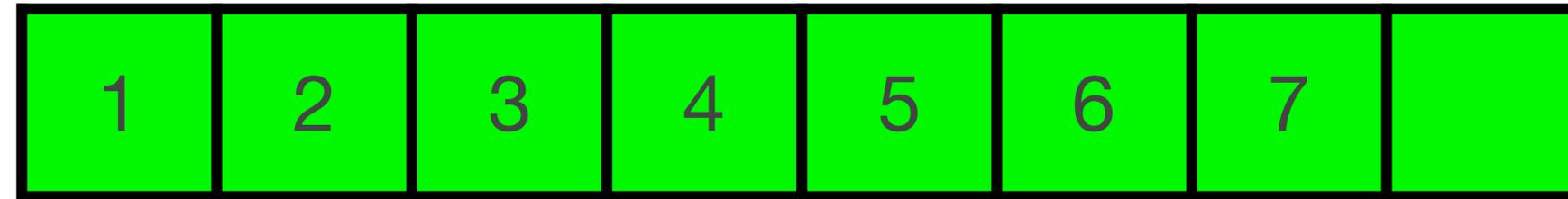
front



Excuse Me, Coming Through

back

front



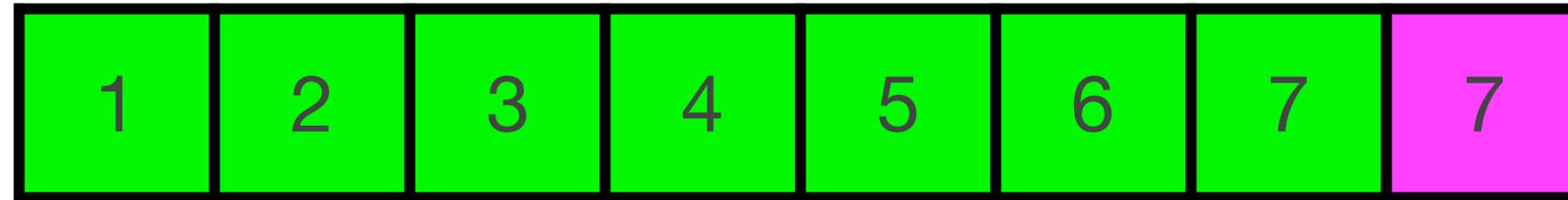
enqueue(42)



Excuse Me, Coming Through

back

front



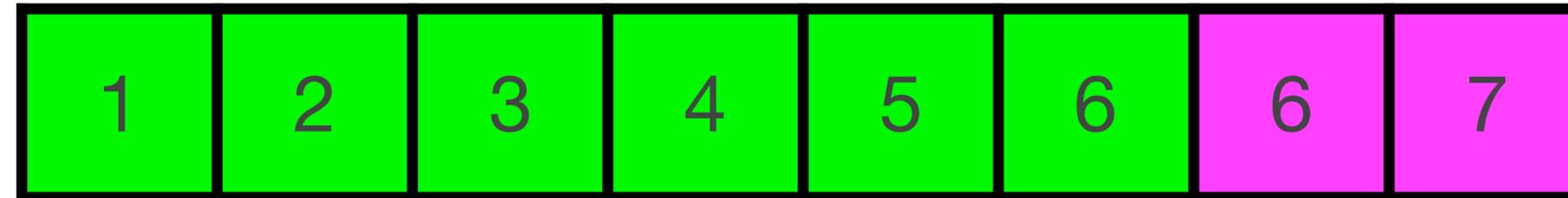
enqueue(42)



Excuse Me, Coming Through

back

front



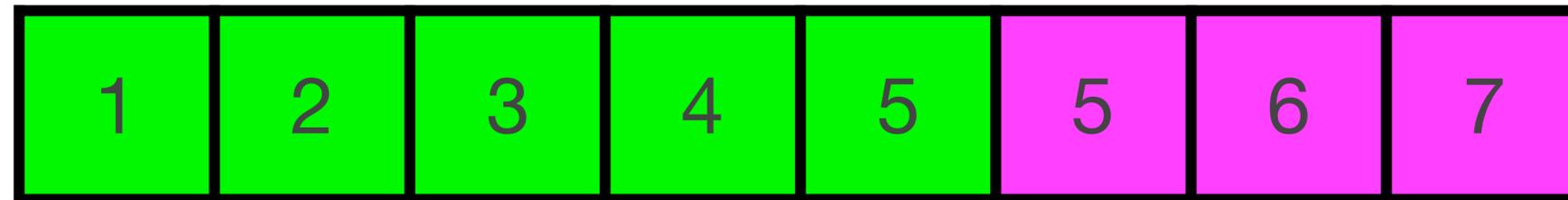
enqueue(42)



Excuse Me, Coming Through

back

front



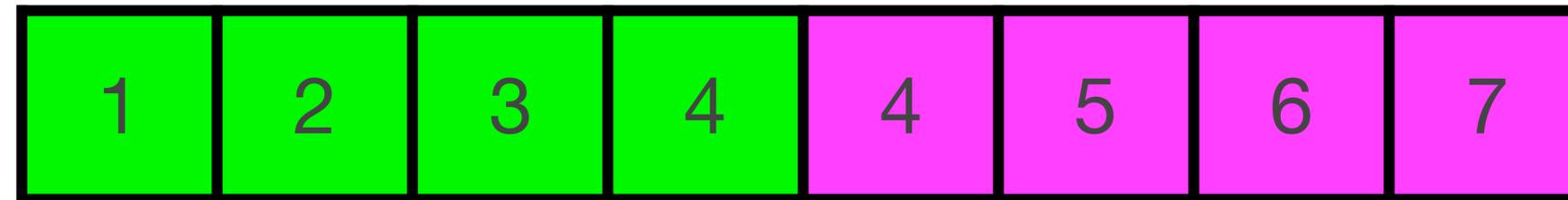
enqueue(42)



Excuse Me, Coming Through

back

front



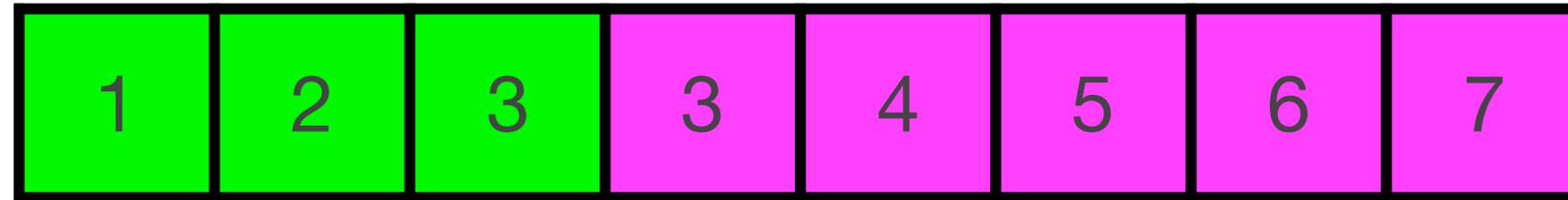
enqueue(42)



Excuse Me, Coming Through

back

front



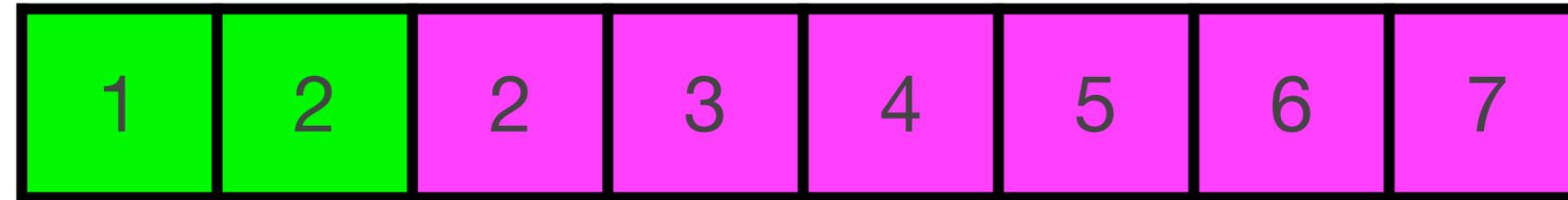
enqueue(42)



Excuse Me, Coming Through

back

front



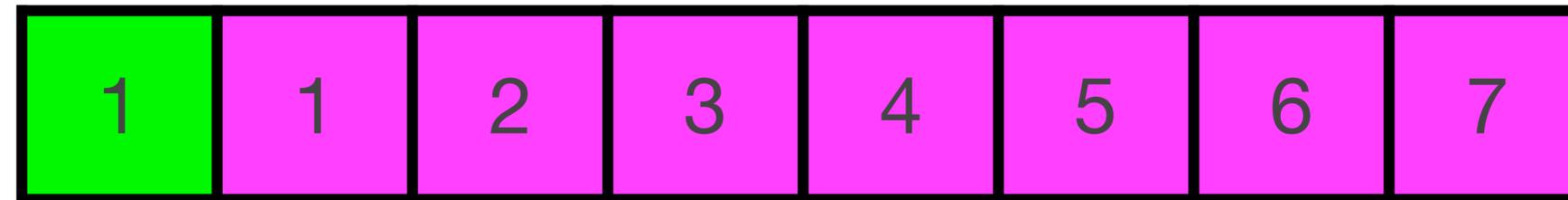
enqueue(42)



Excuse Me, Coming Through

back

front



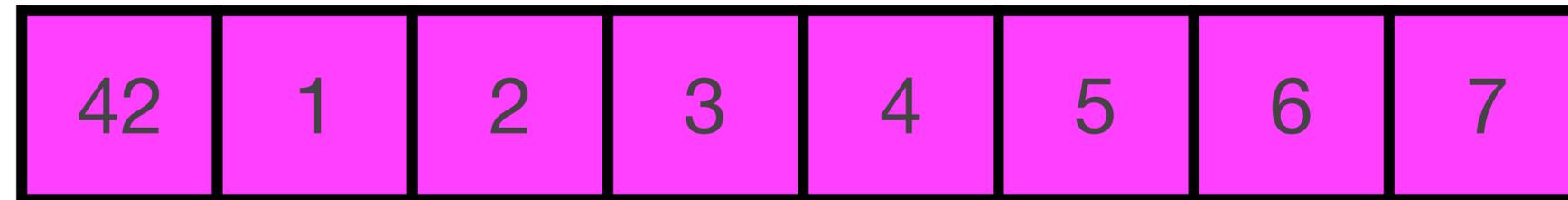
enqueue(42)



Excuse Me, Coming Through

back

front



enqueue(42)



Queue as Vector: Big O

Enqueue: $O(n)$

Dequeue: $O(1)$

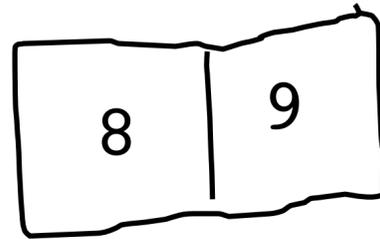
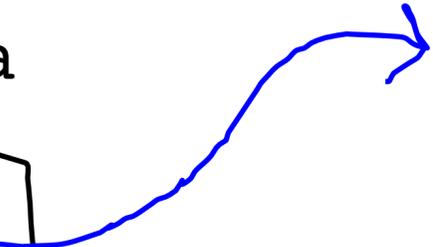
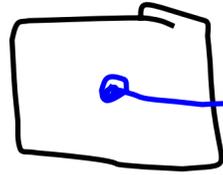


**WE CAN DO
BETTER**

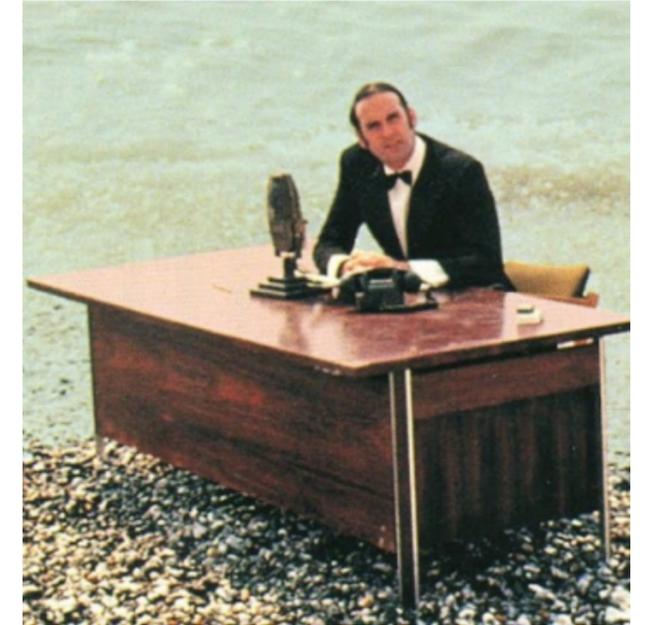


And Now for Something Completely Different

```
int *  
data
```

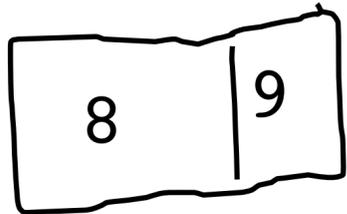
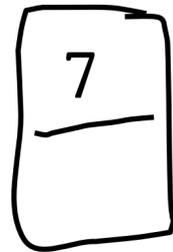
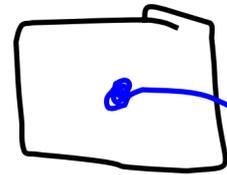


```
enqueue(7)
```



And Now for Something Completely Different

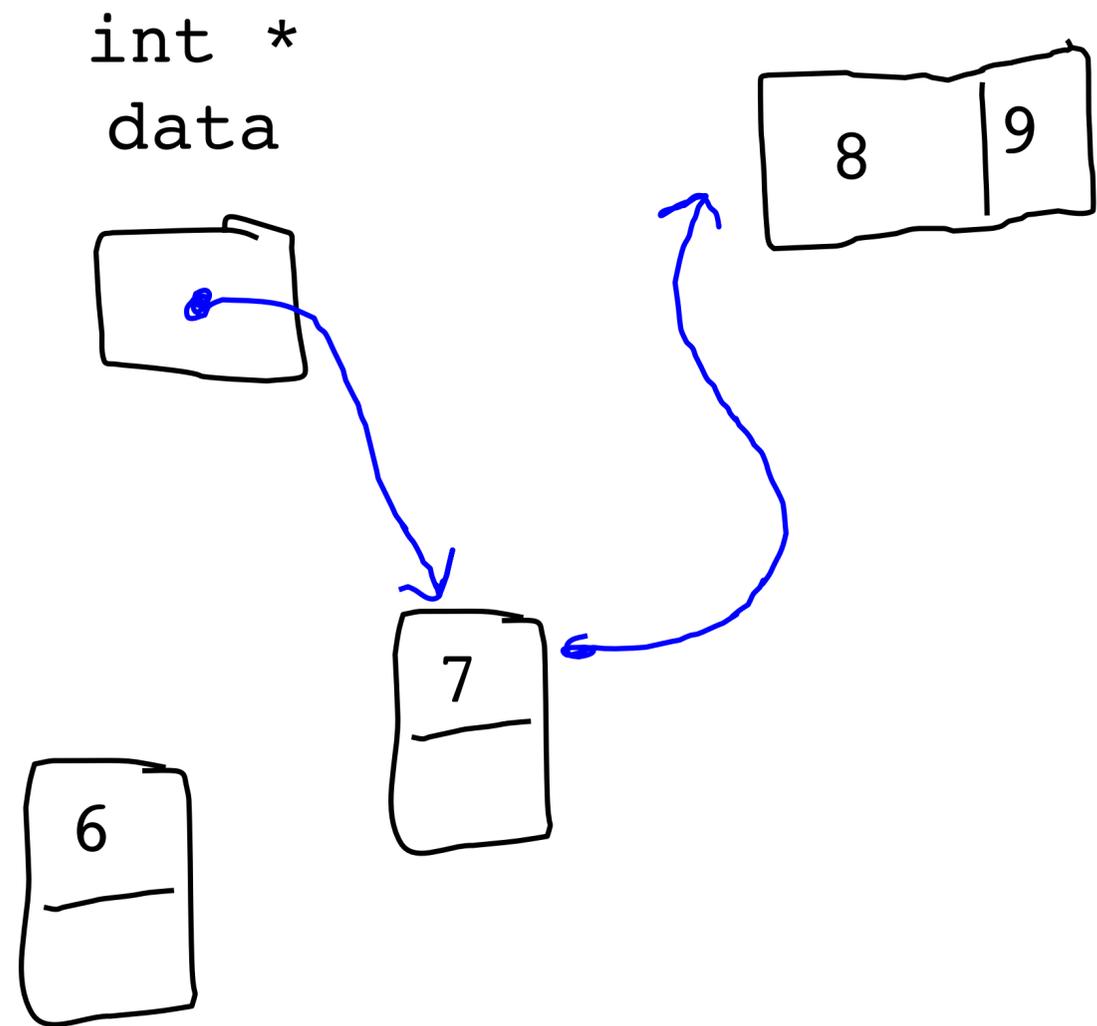
```
int *  
data
```



```
enqueue(7)
```



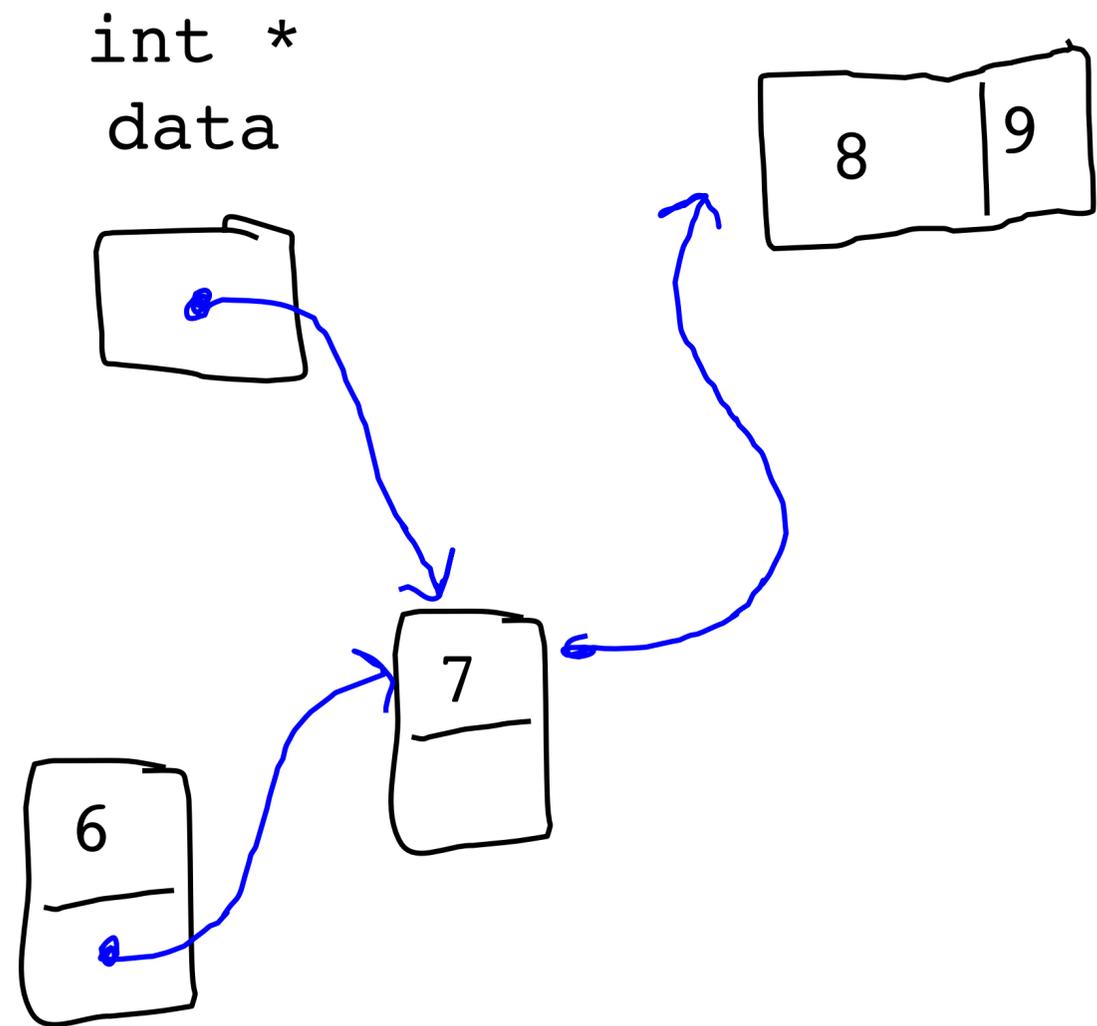
And Now for Something Completely Different



enqueue(6)



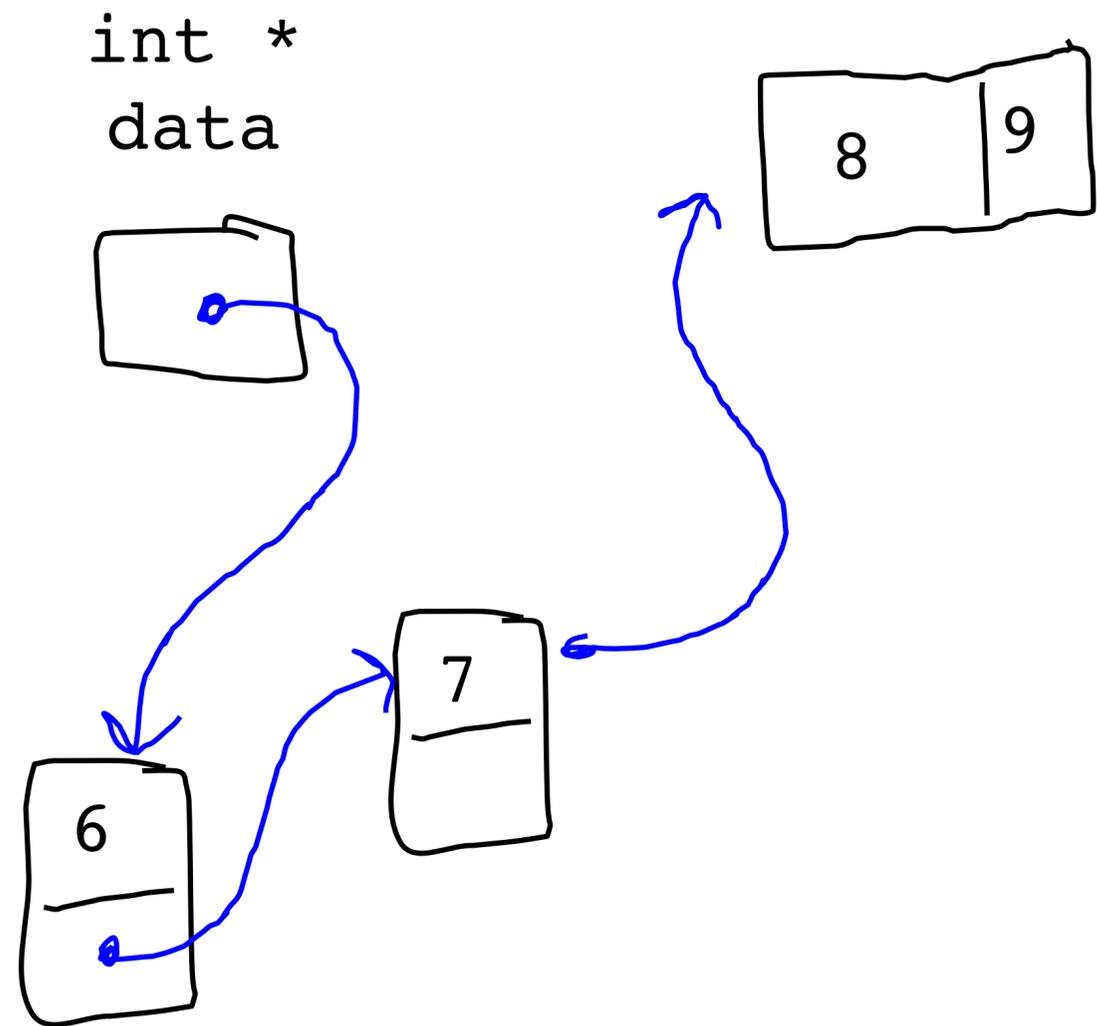
And Now for Something Completely Different



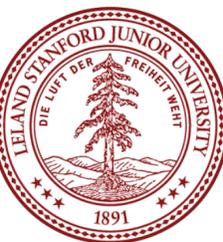
enqueue(6)



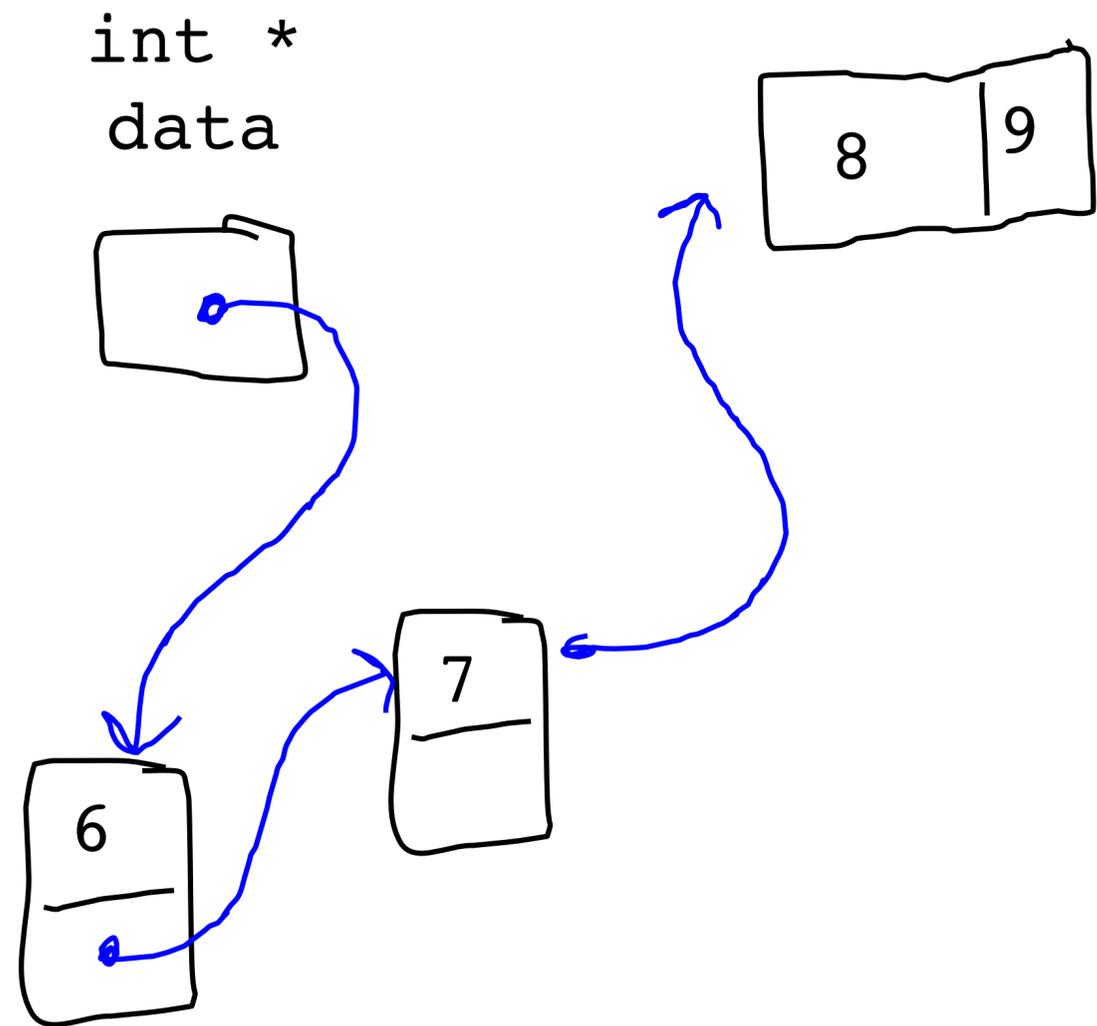
And Now for Something Completely Different



enqueue(6)



And Now for Something Completely Different

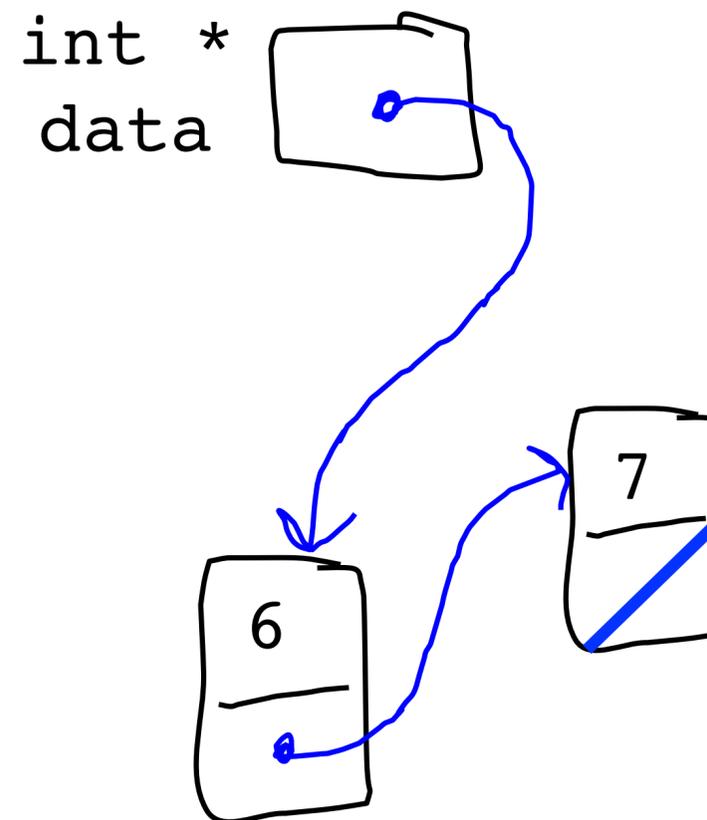
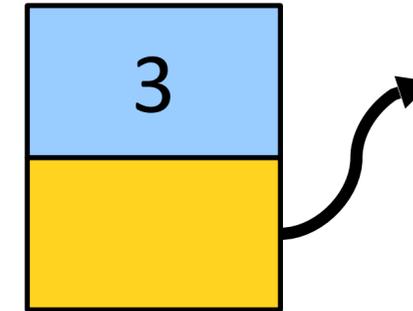


Now we have a way to add
to the front in $O(1)$ time!



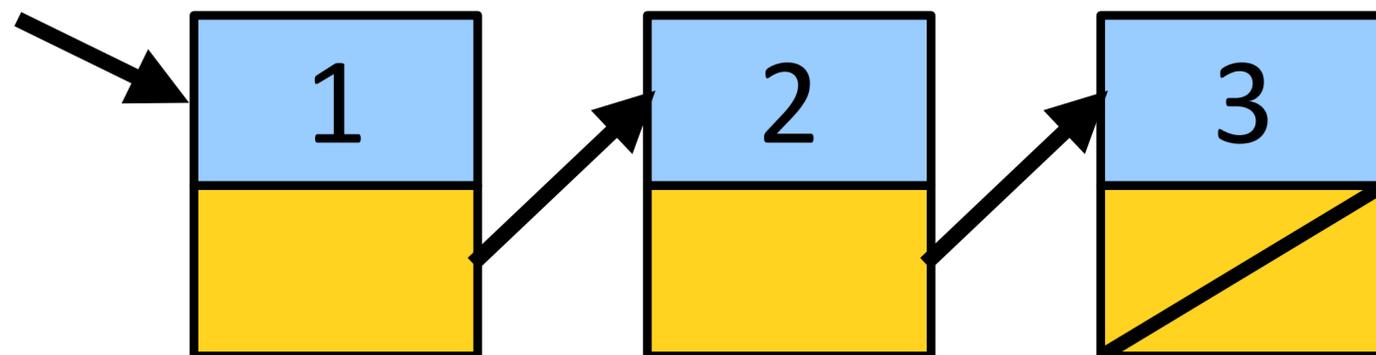
Linked List

- 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.



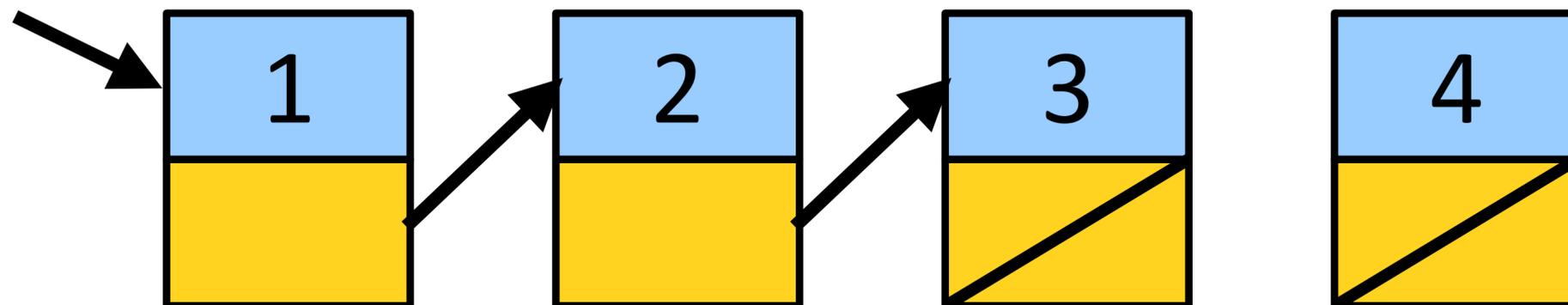
Linked Lists

- 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.



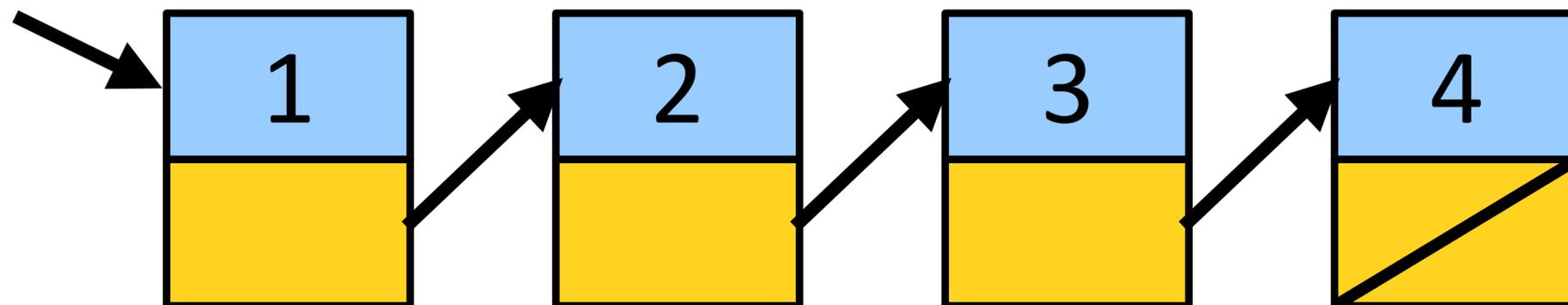
Linked Lists

- 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.
- To add a node at the end, we chain the last to the end



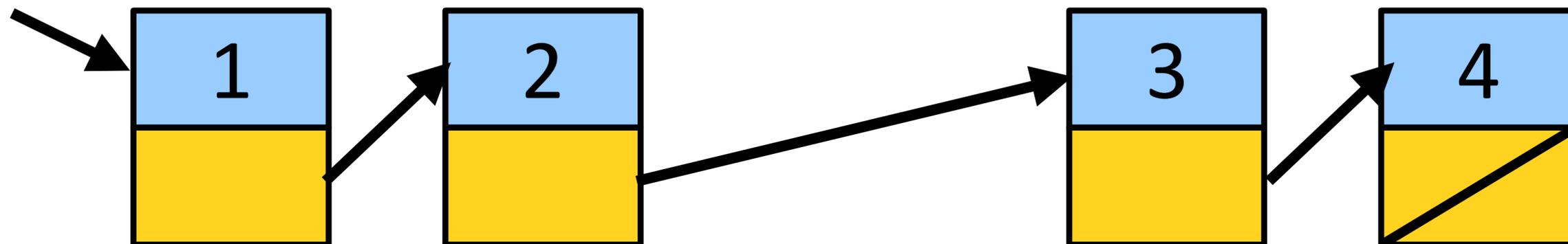
Linked Lists

- 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.
- To add a node at the end, we chain the last to the end



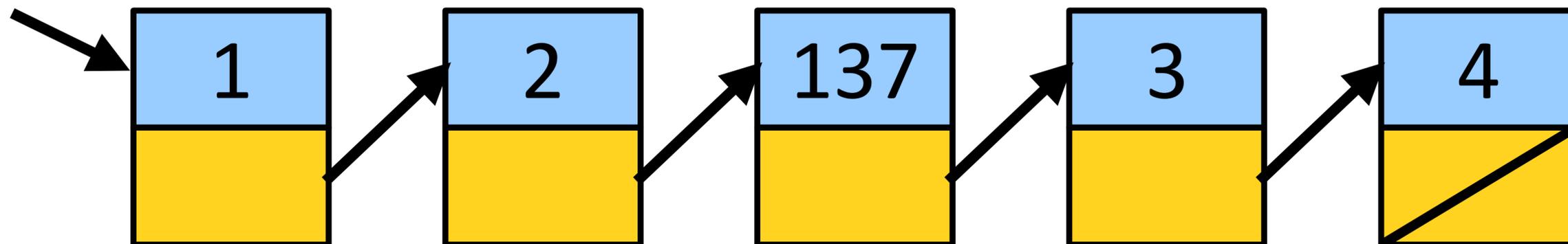
Linked Lists

- 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.
- To add a node in the middle, we simply add one to the middle (we have to find that location, though!)



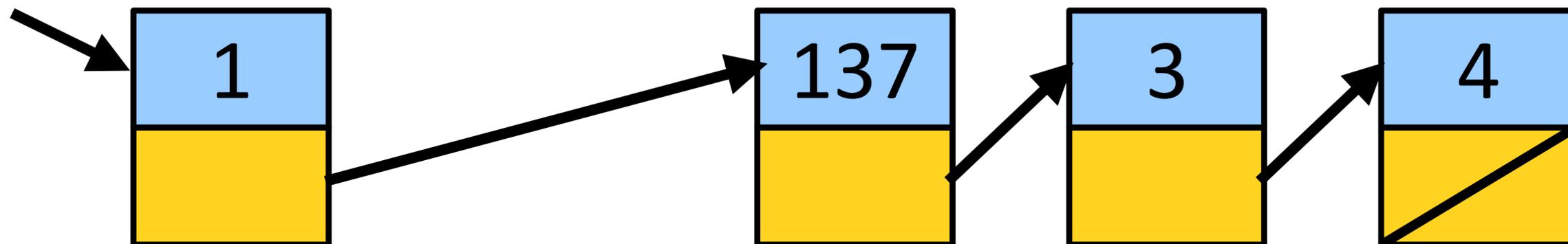
Linked Lists

- 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.
- To add a node in the middle, we simply add one to the middle (we have to find that location, though!)



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.
- To remove a node, we connect the previous node to the next node.



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!



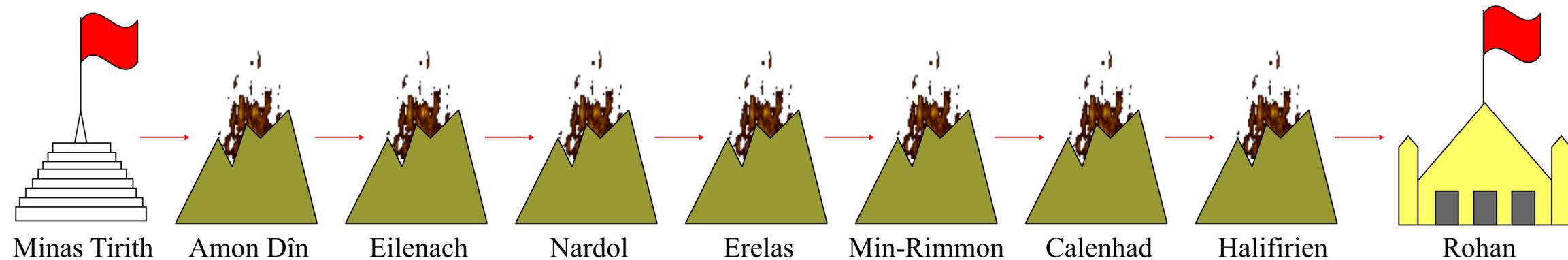
Always!

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.





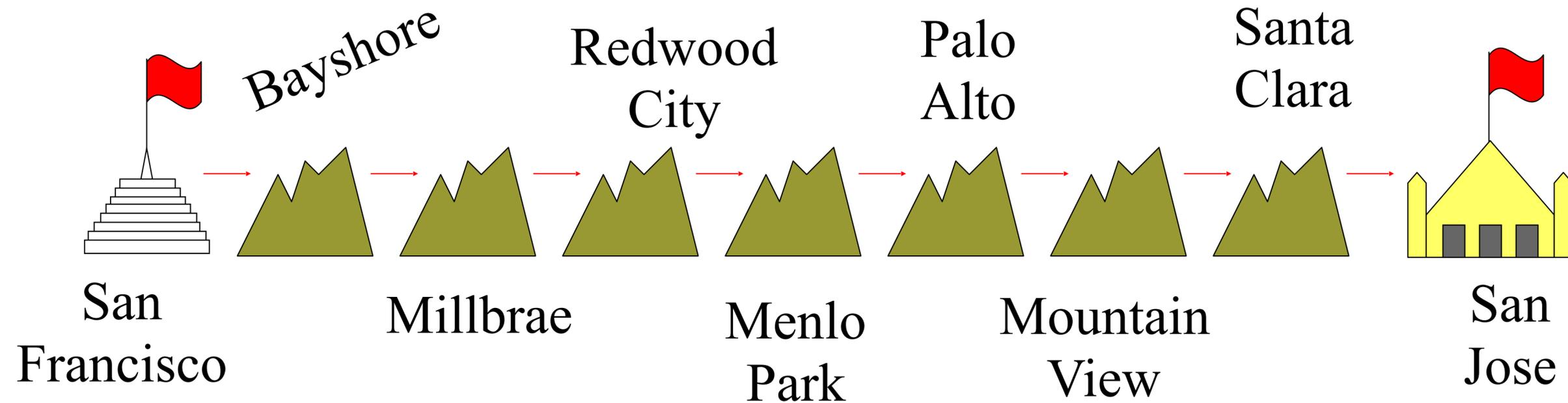
THE
LORD OF THE RINGS
THE RETURN OF THE KING



<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;
}
```



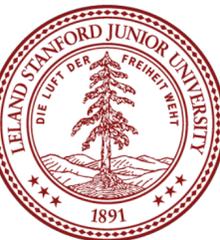
Linked List Trace

```
// 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);  
    }  
}
```

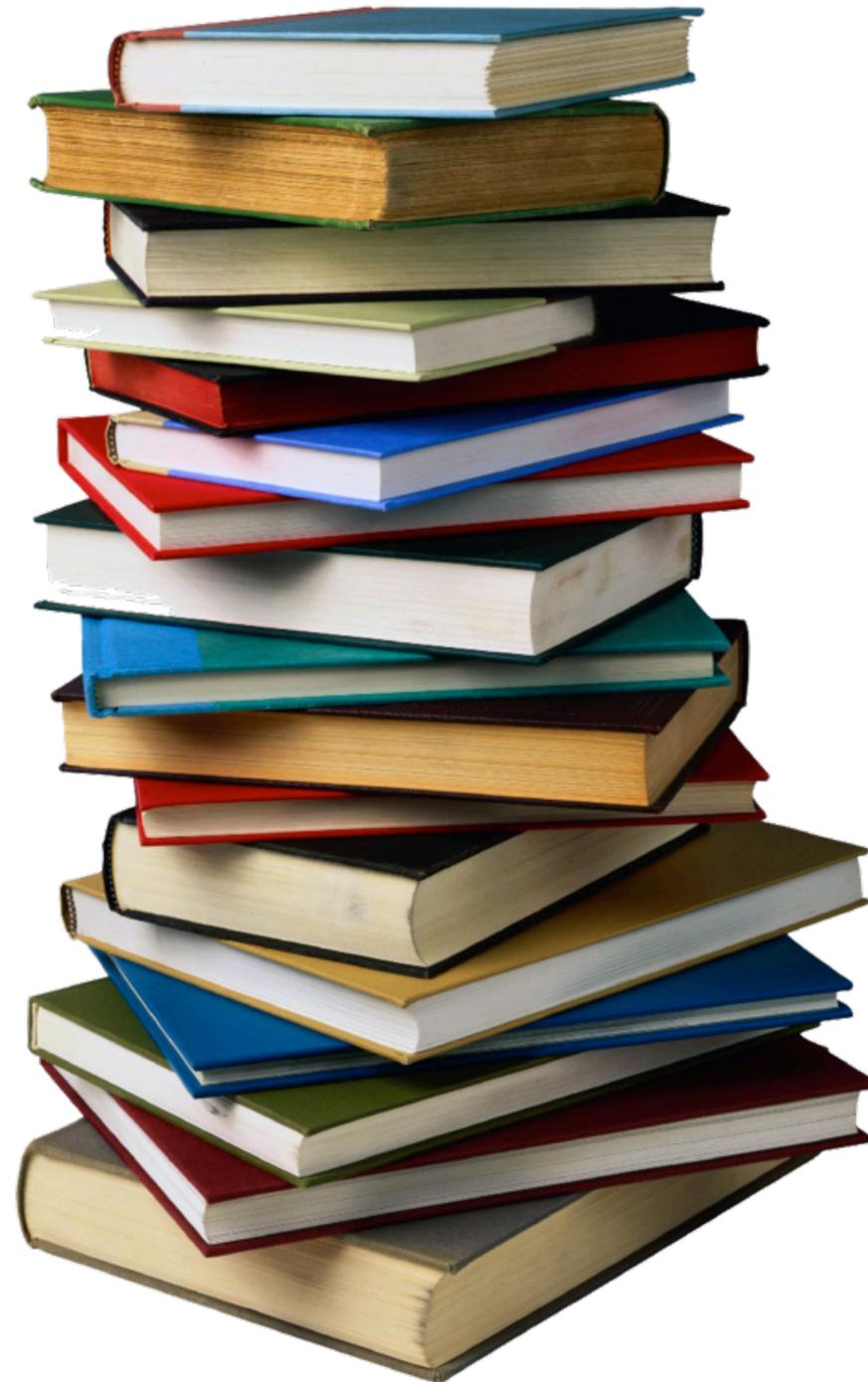
```
signal(head);
```



Lord of the Linked Lists



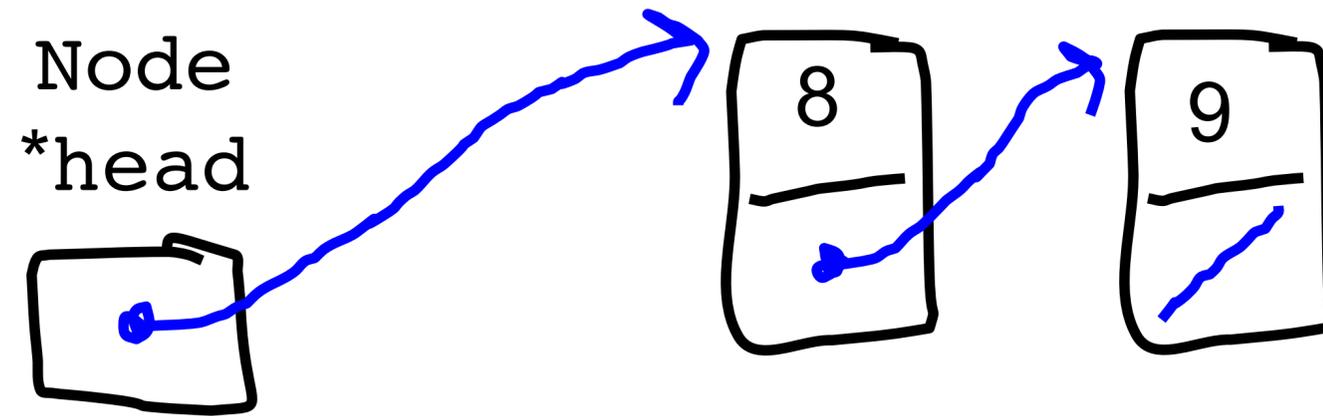
How is the Stack Implemented?



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

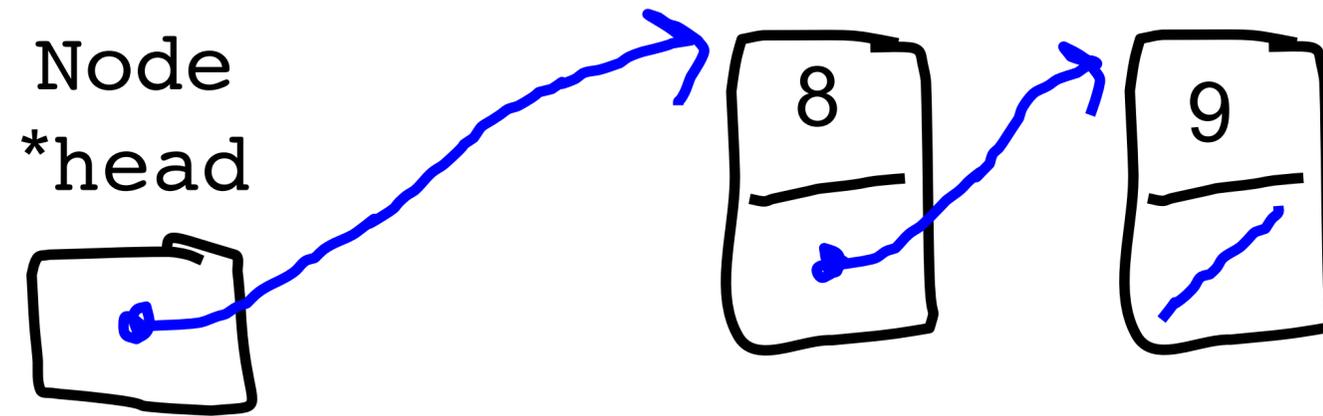


Stack

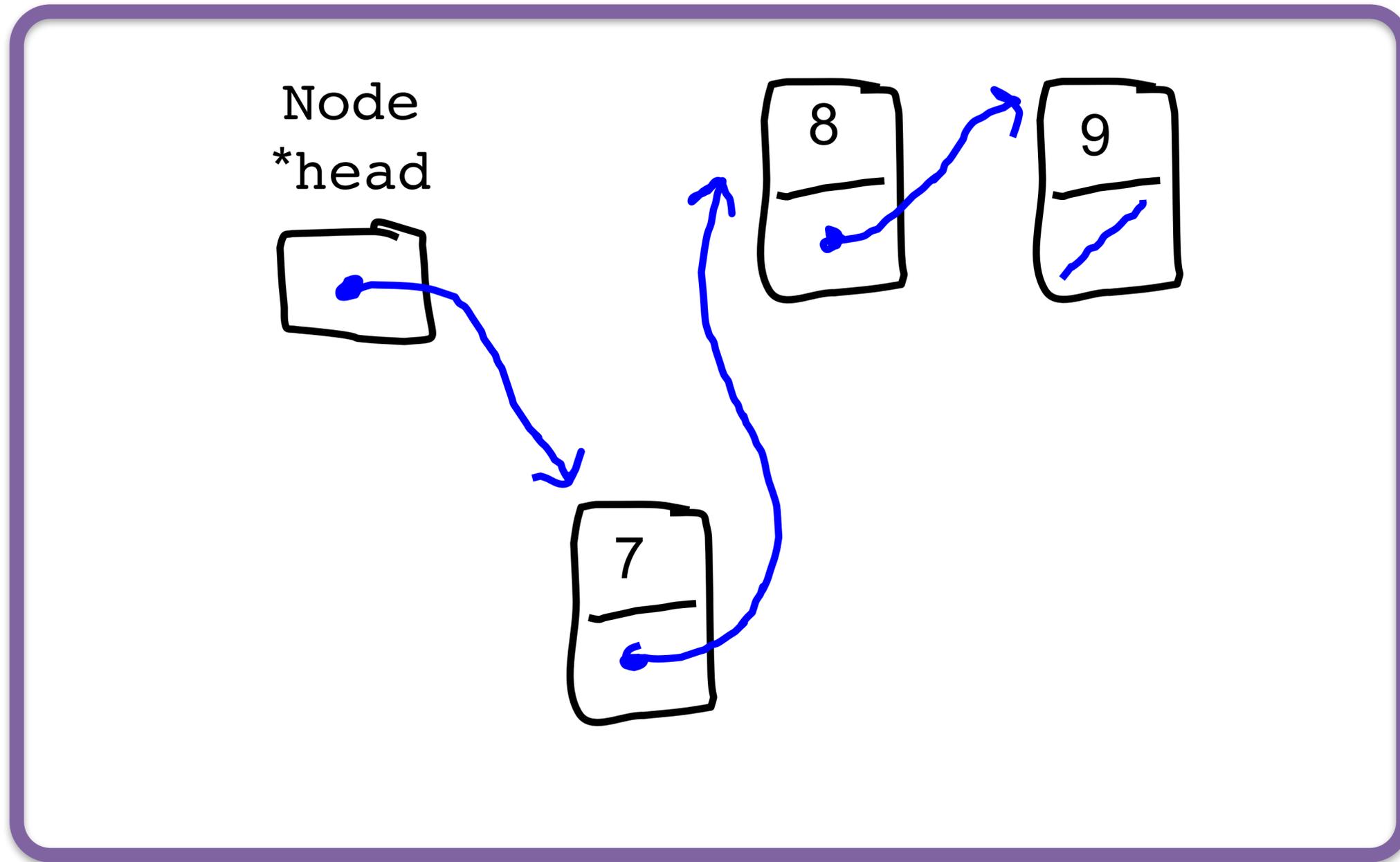


Stack

```
push(7);
```

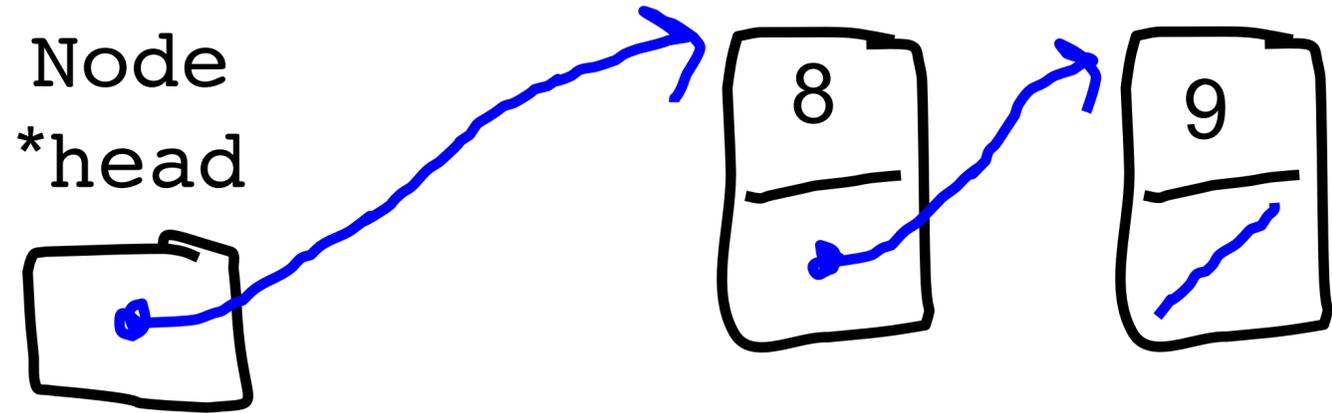


Goal of Push

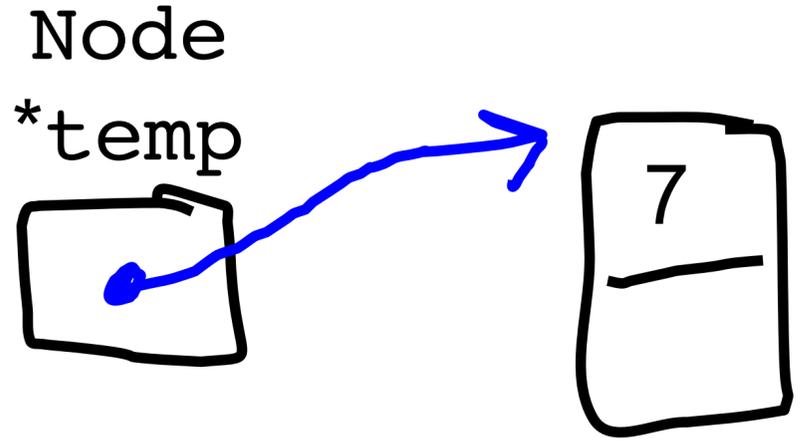
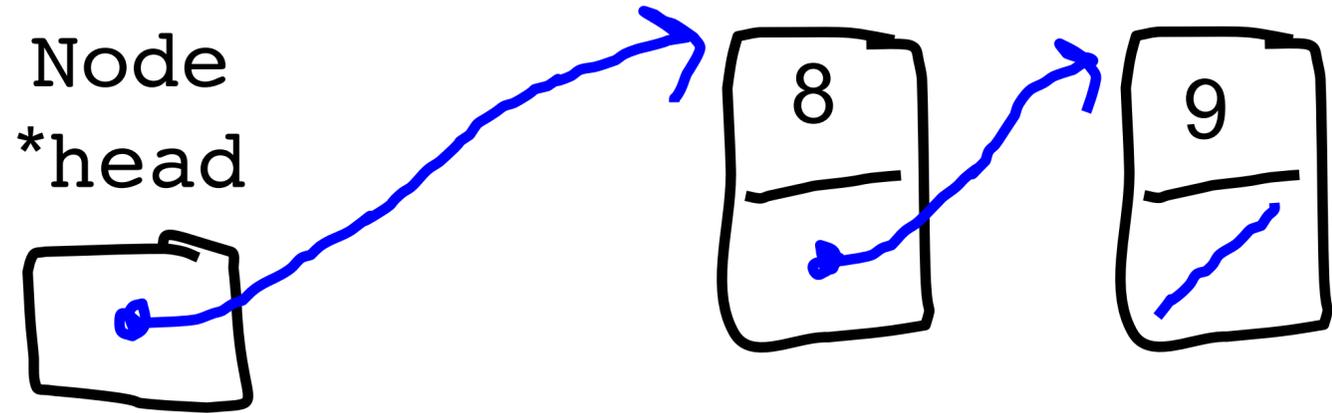


Push Attempt

```
push(7);
```



push(7);

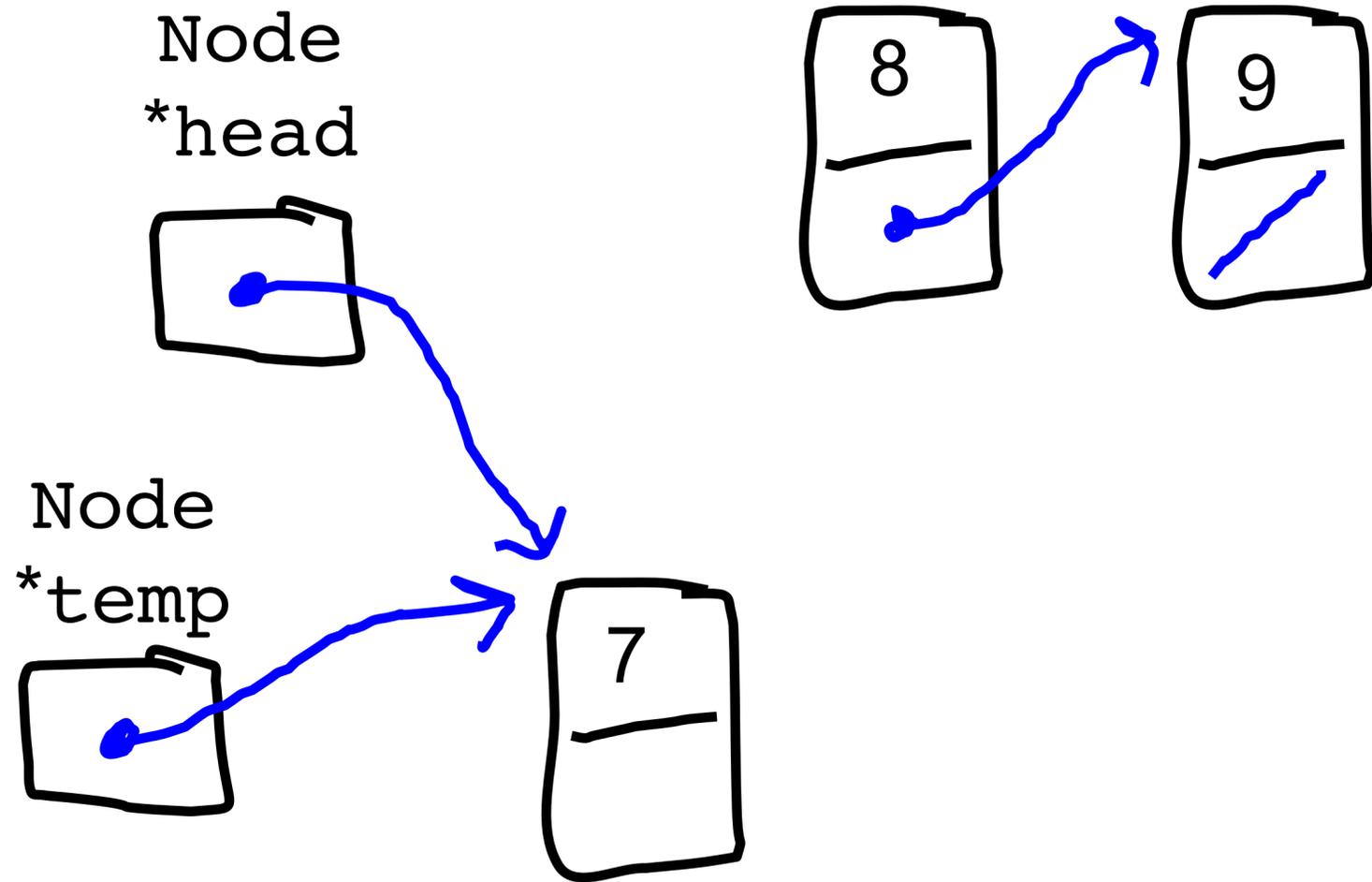


```
Node * temp = new Node;  
temp -> value = 7;
```

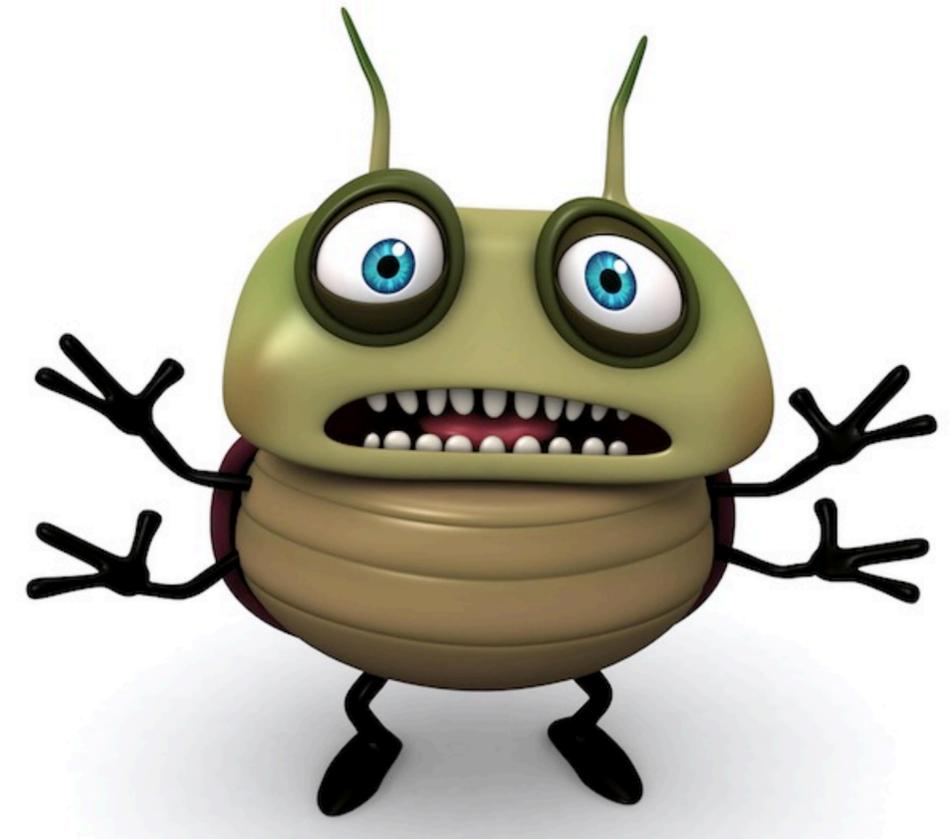


Stack is a Linked List

```
push(7);
```

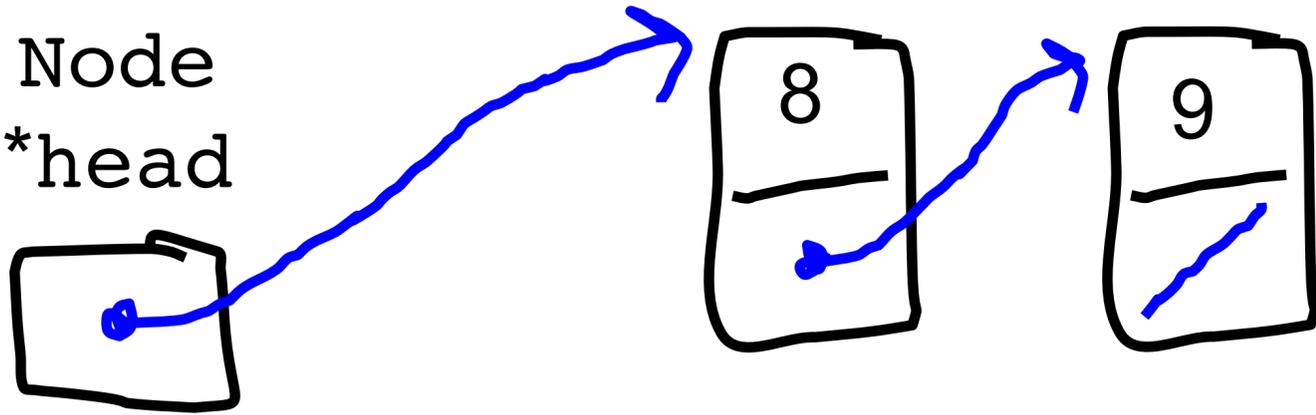


```
head = temp;
```



BRAAWWRRRR!

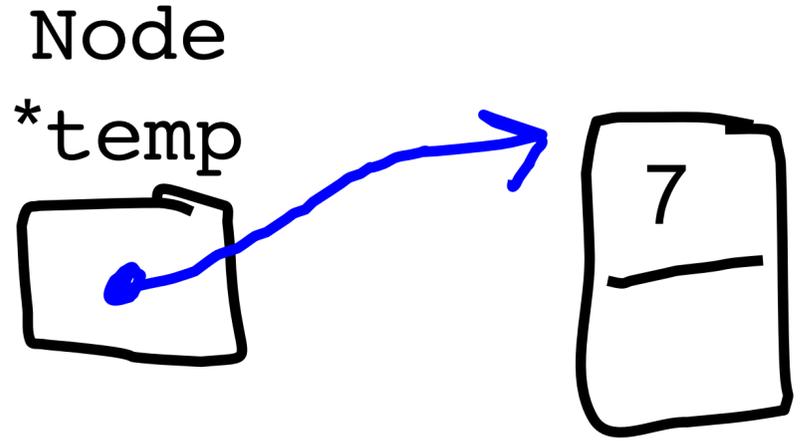
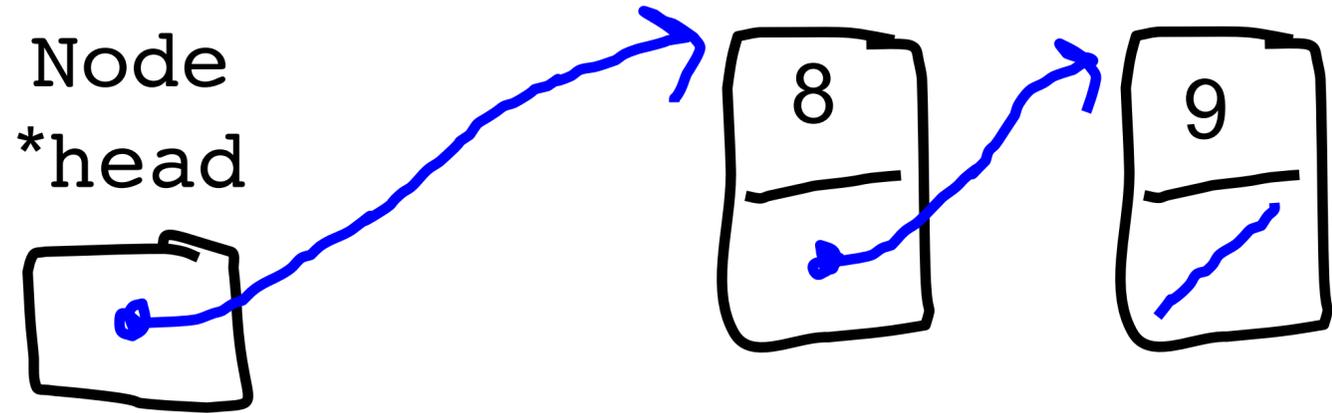
Node
*head



```
push(7);
```



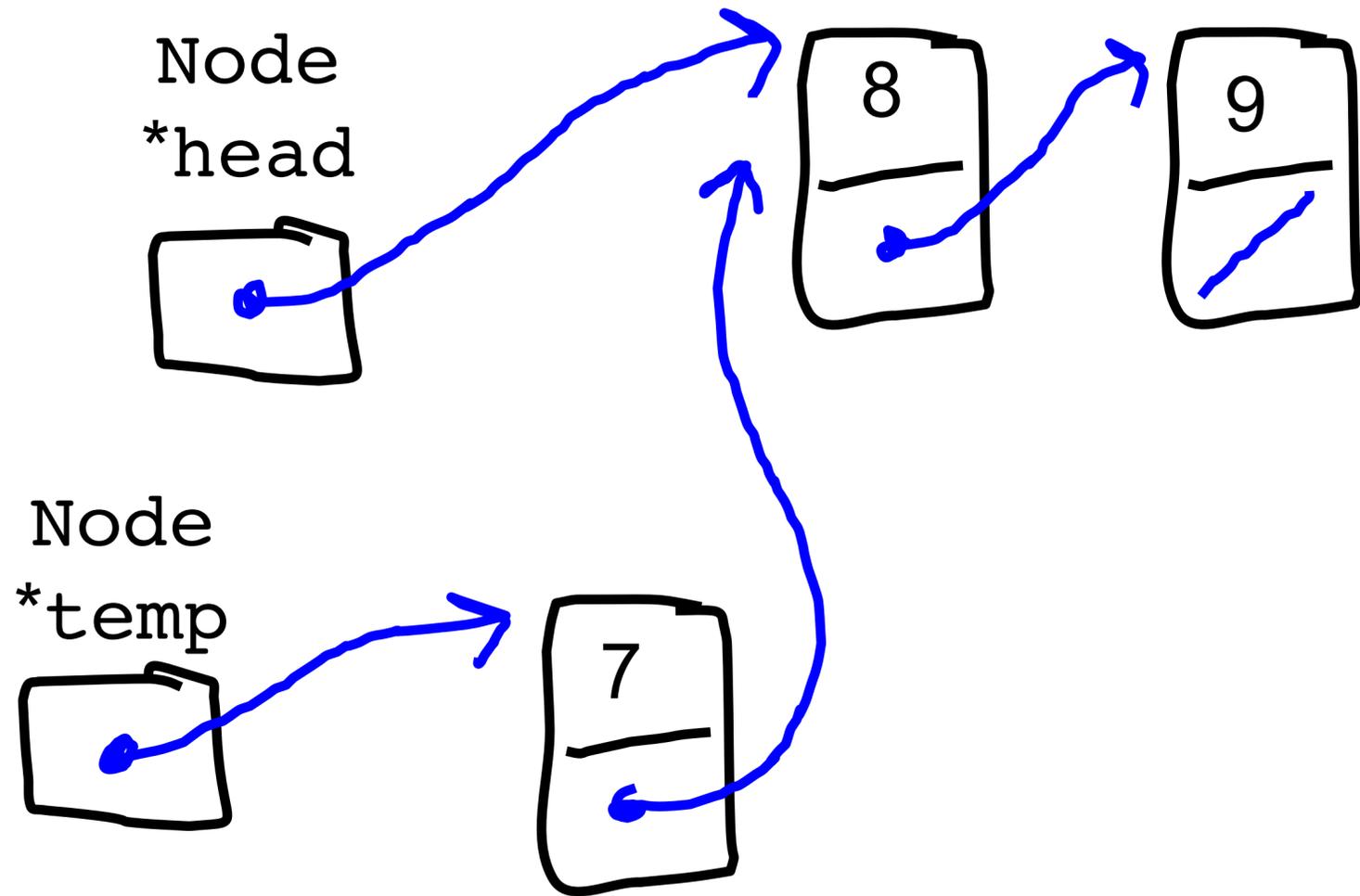
push(7);



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



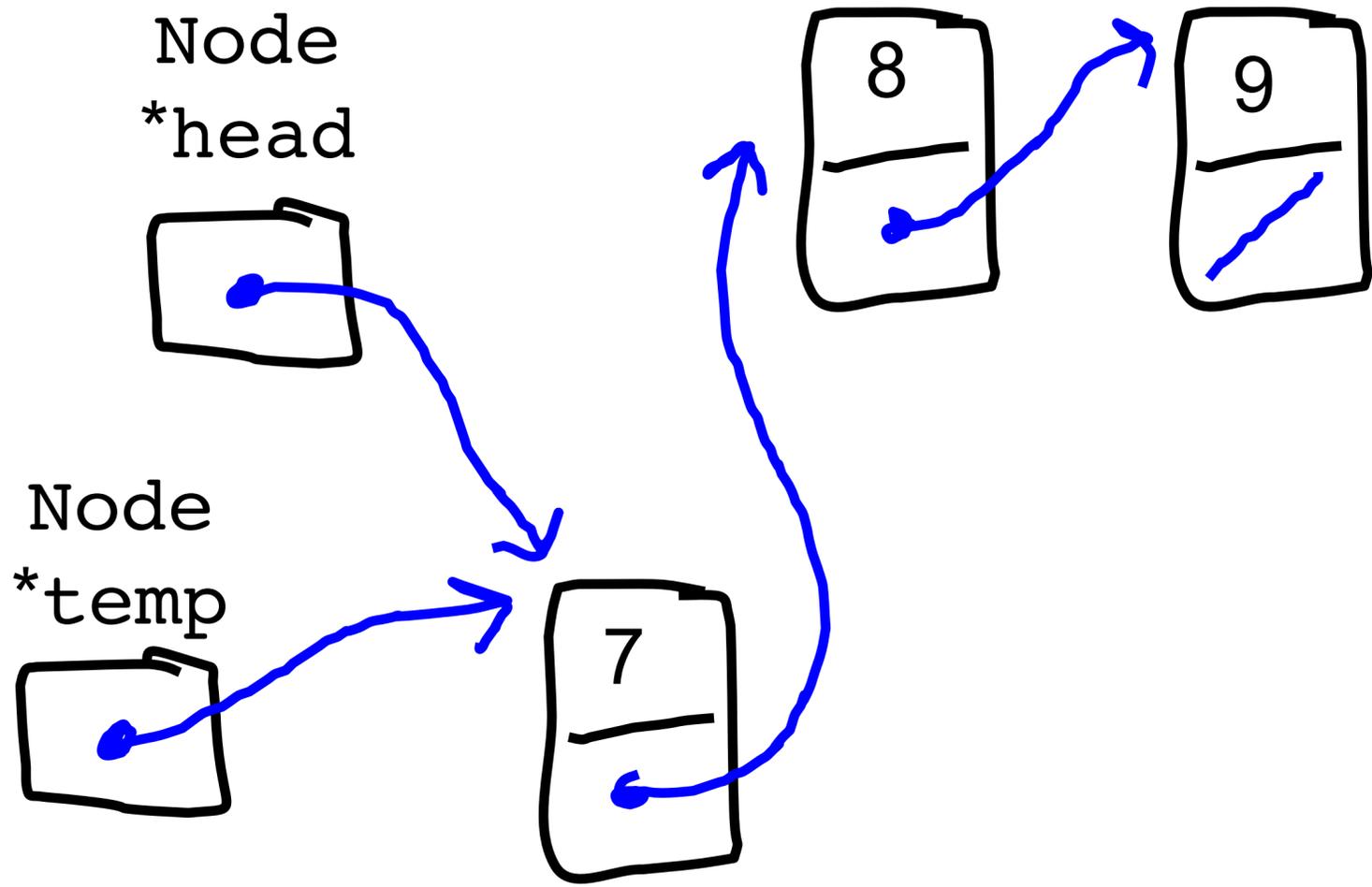
push(7);



`temp -> next = head;`



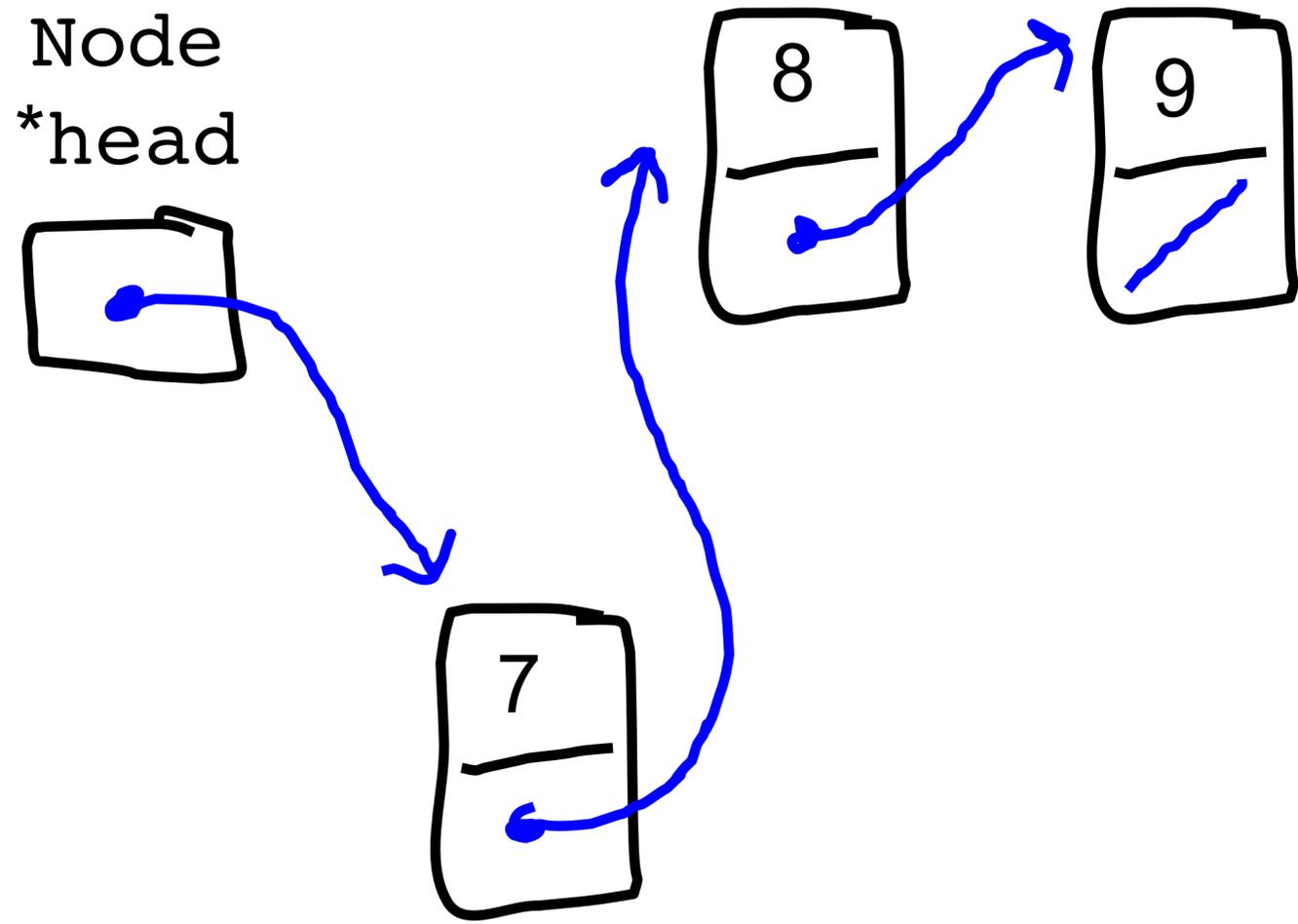
```
push(7);
```



```
head = temp;
```



`push(7);`

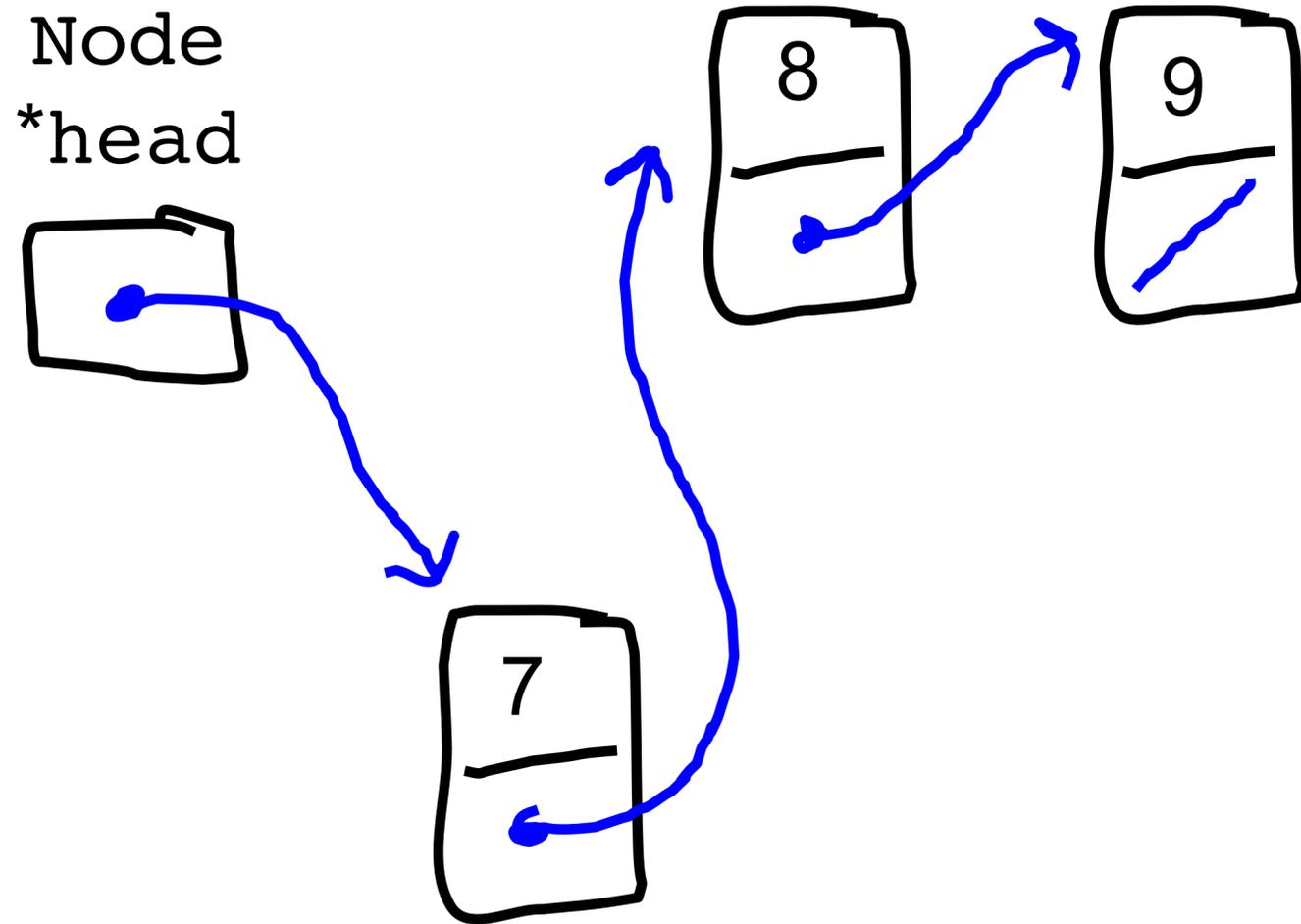


exit function



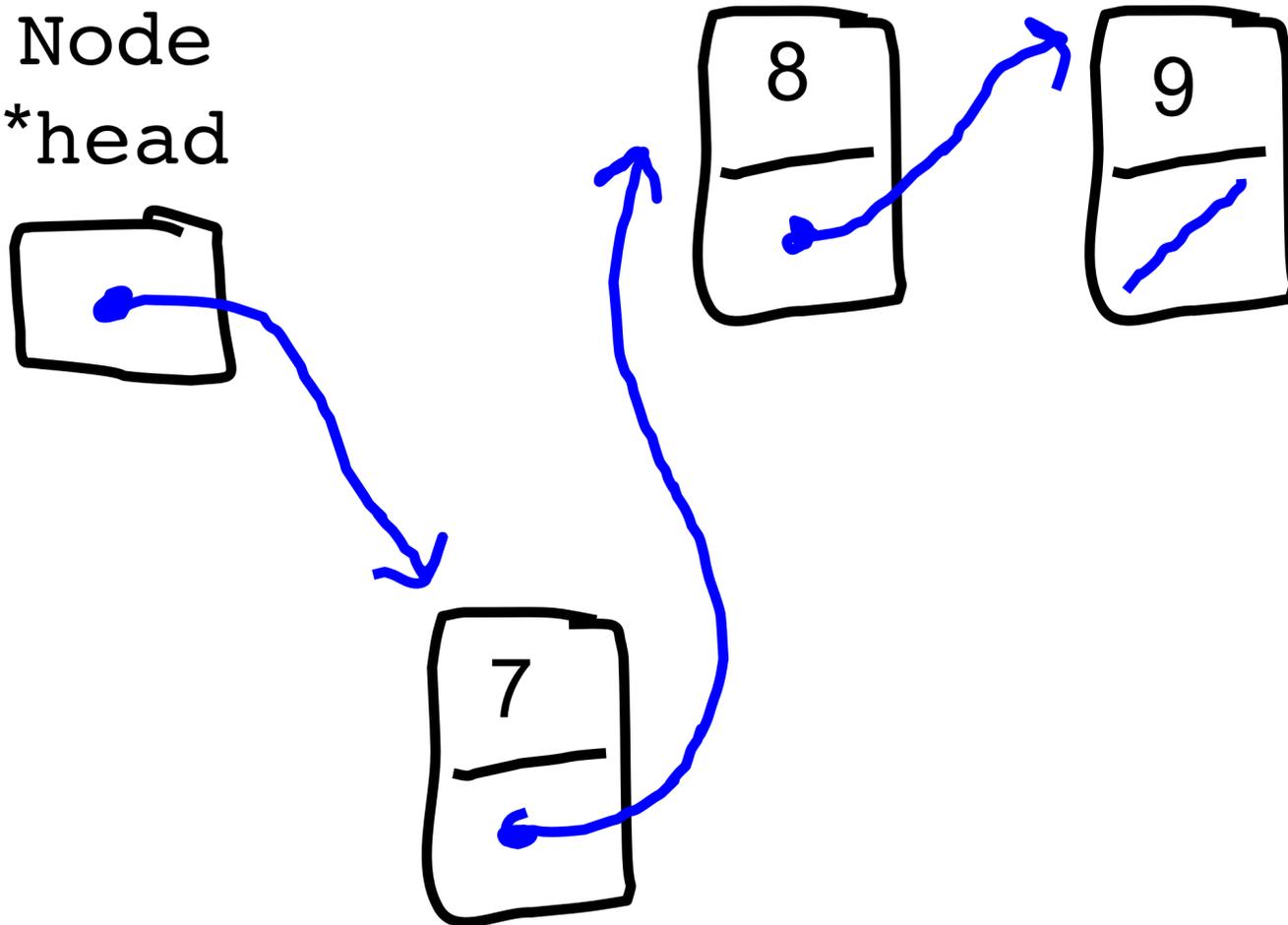
Stack is a Linked List

pop();



Stack is a Linked List

Node
*head



```
pop();
```

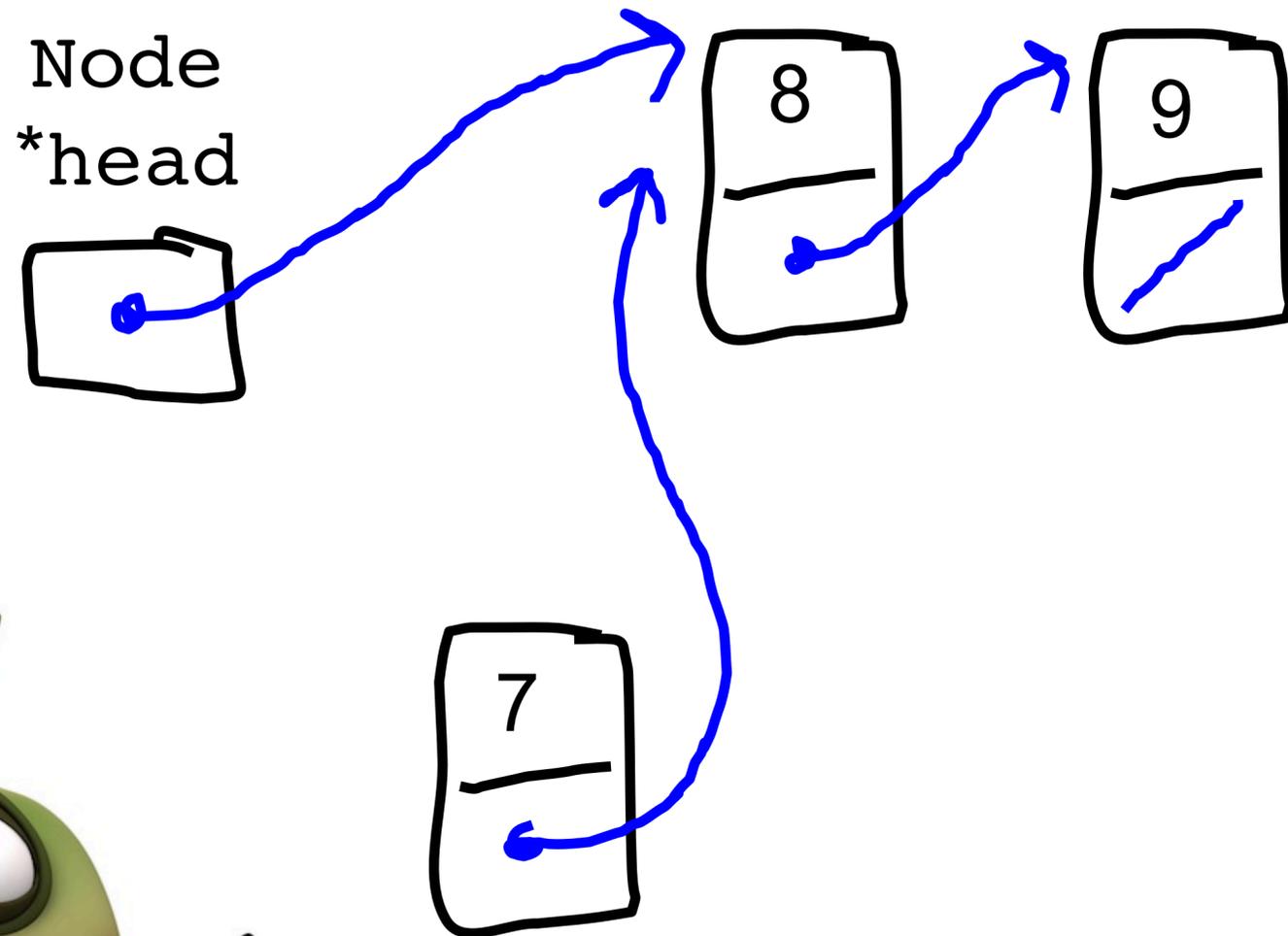
int toReturn

```
7
```

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



Stack is a Linked List

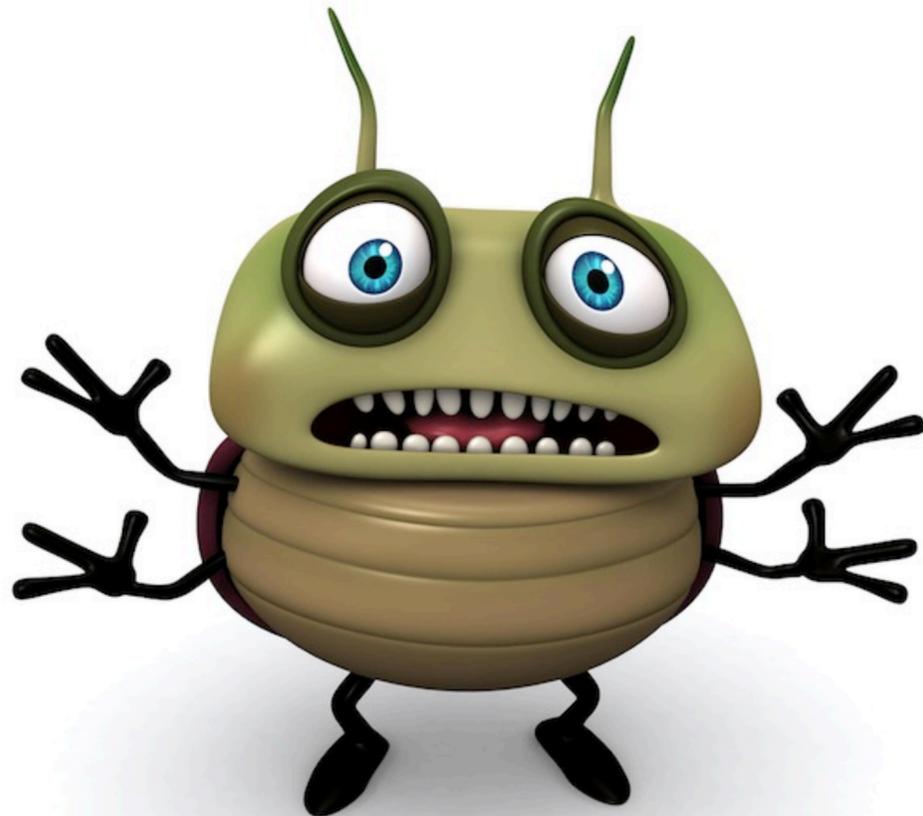


```
pop();
```

int toReturn

```
7
```

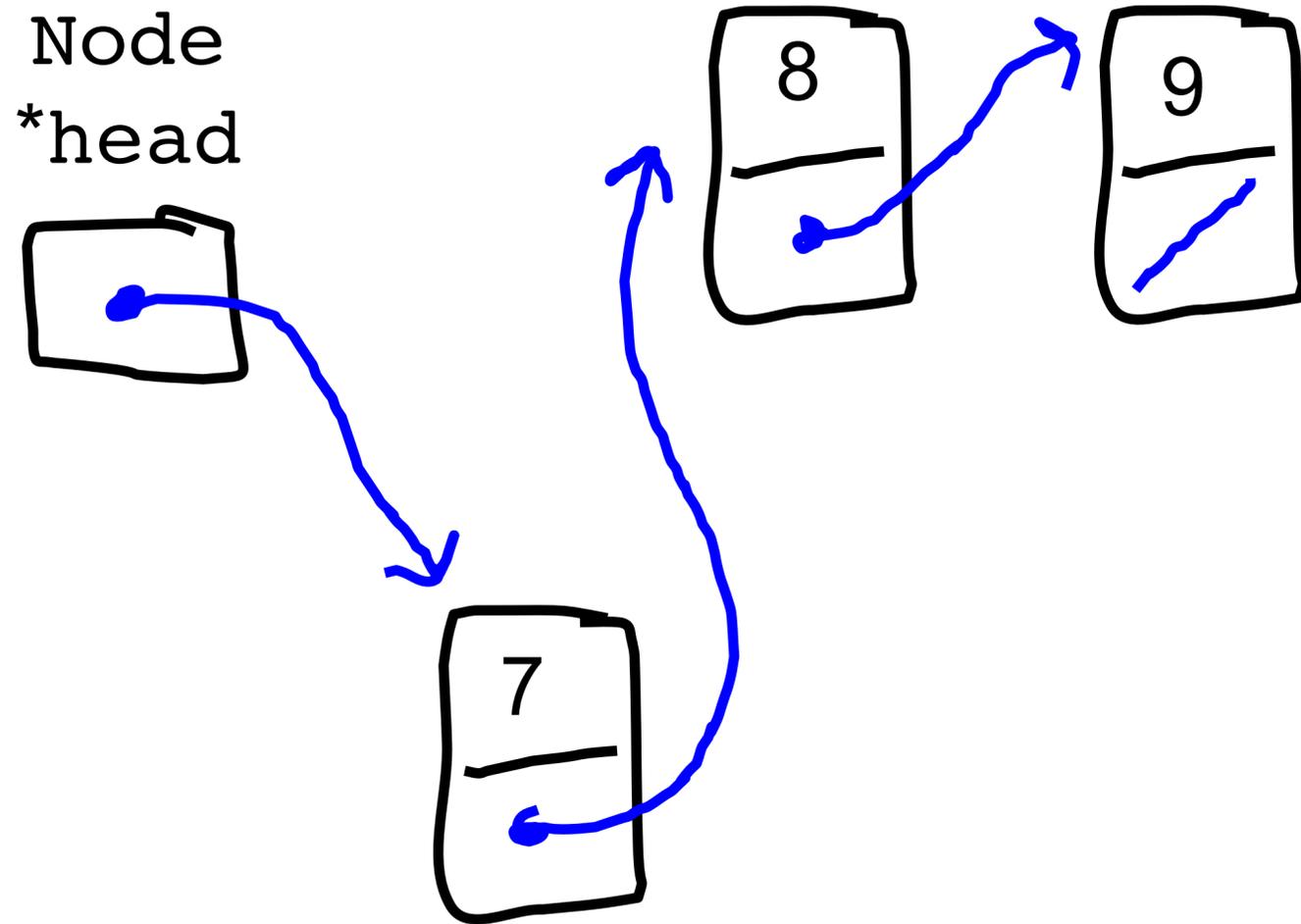
```
head = head->next;
```



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

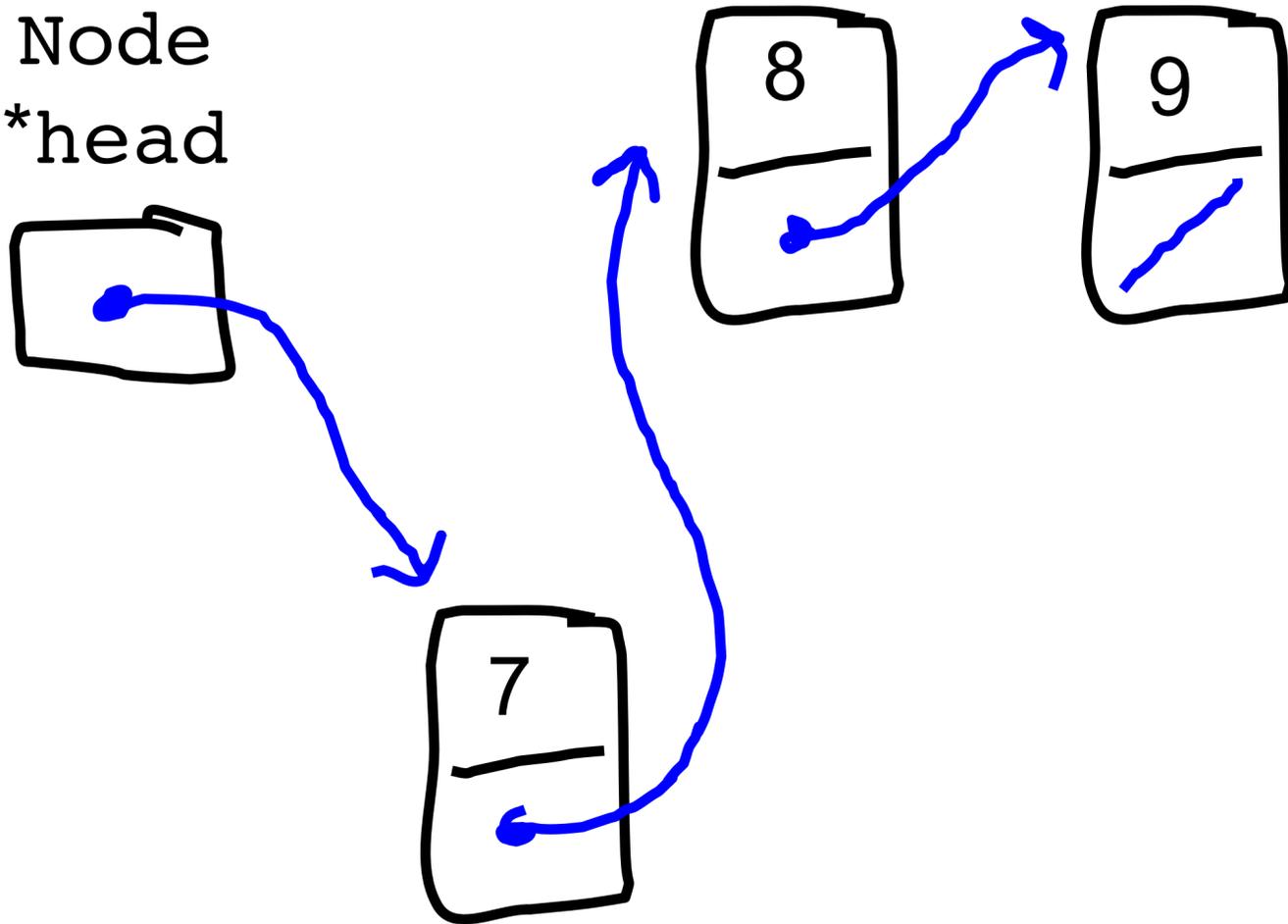
Stack is a Linked List

pop();



Stack is a Linked List

Node
*head



```
pop();
```

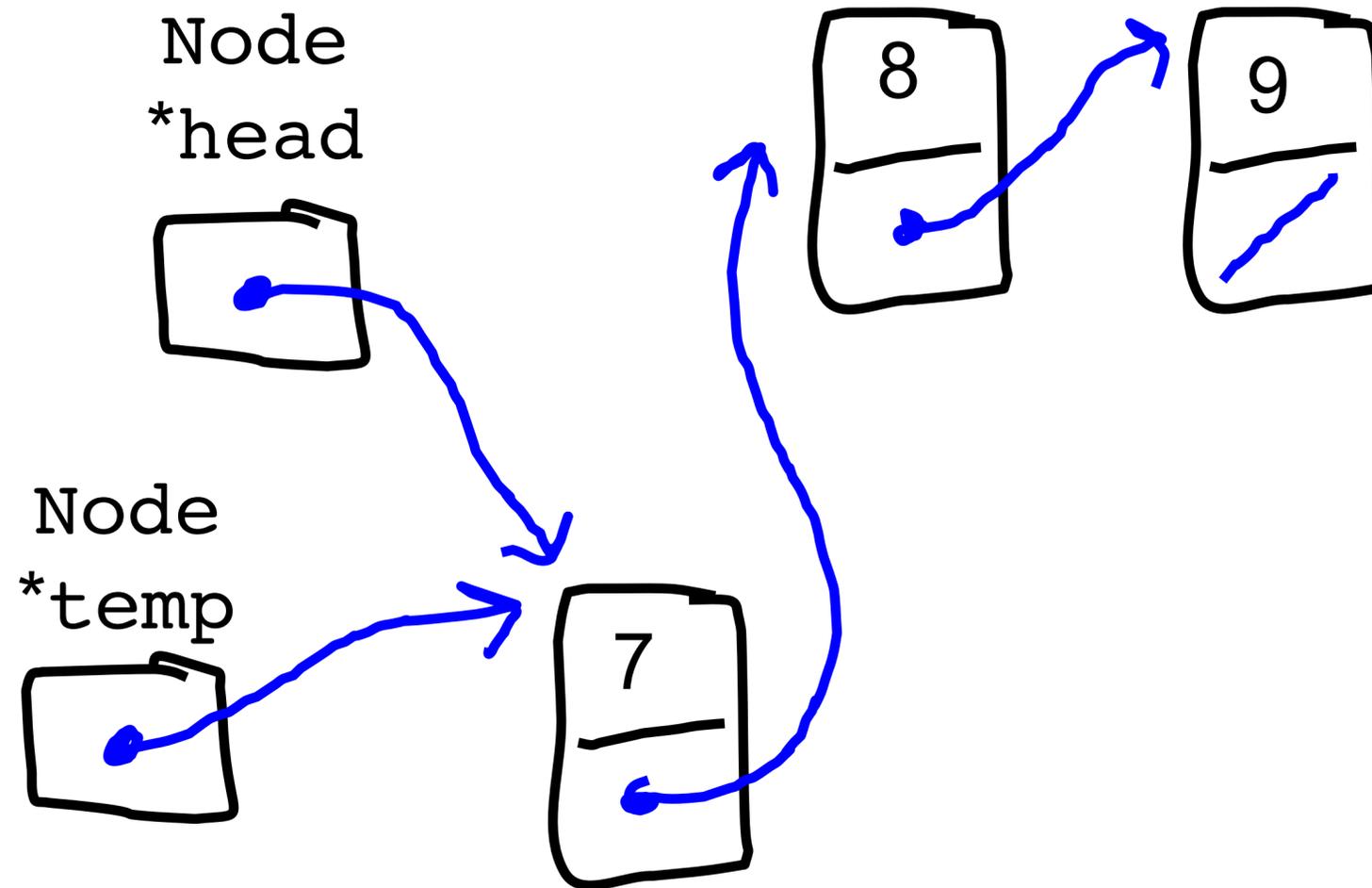
int toReturn

```
7
```

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



Stack is a Linked List



```
pop();
```

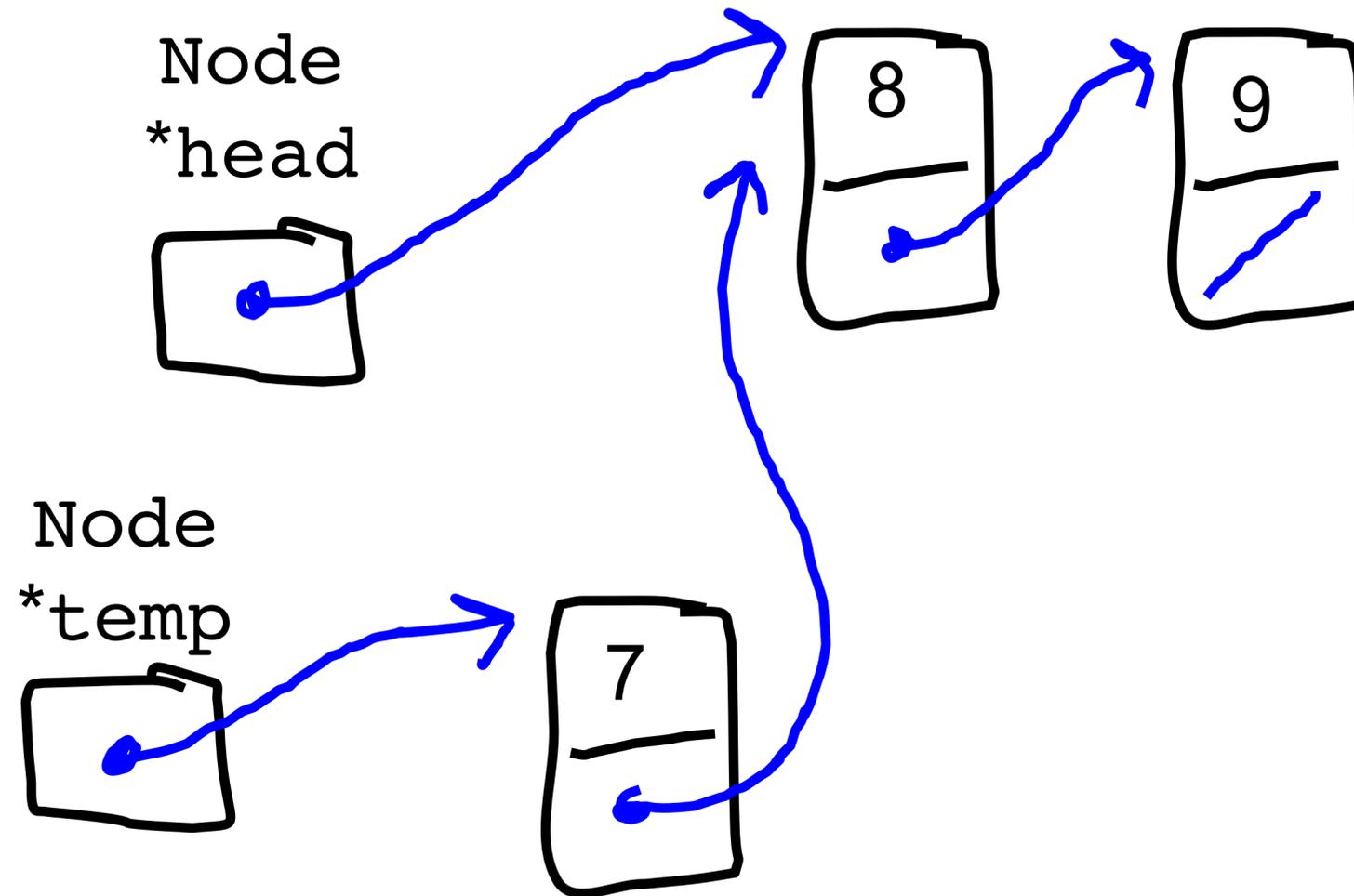
`int toReturn`

```
7
```

```
Node * temp = head;
```



Stack is a Linked List



```
pop();
```

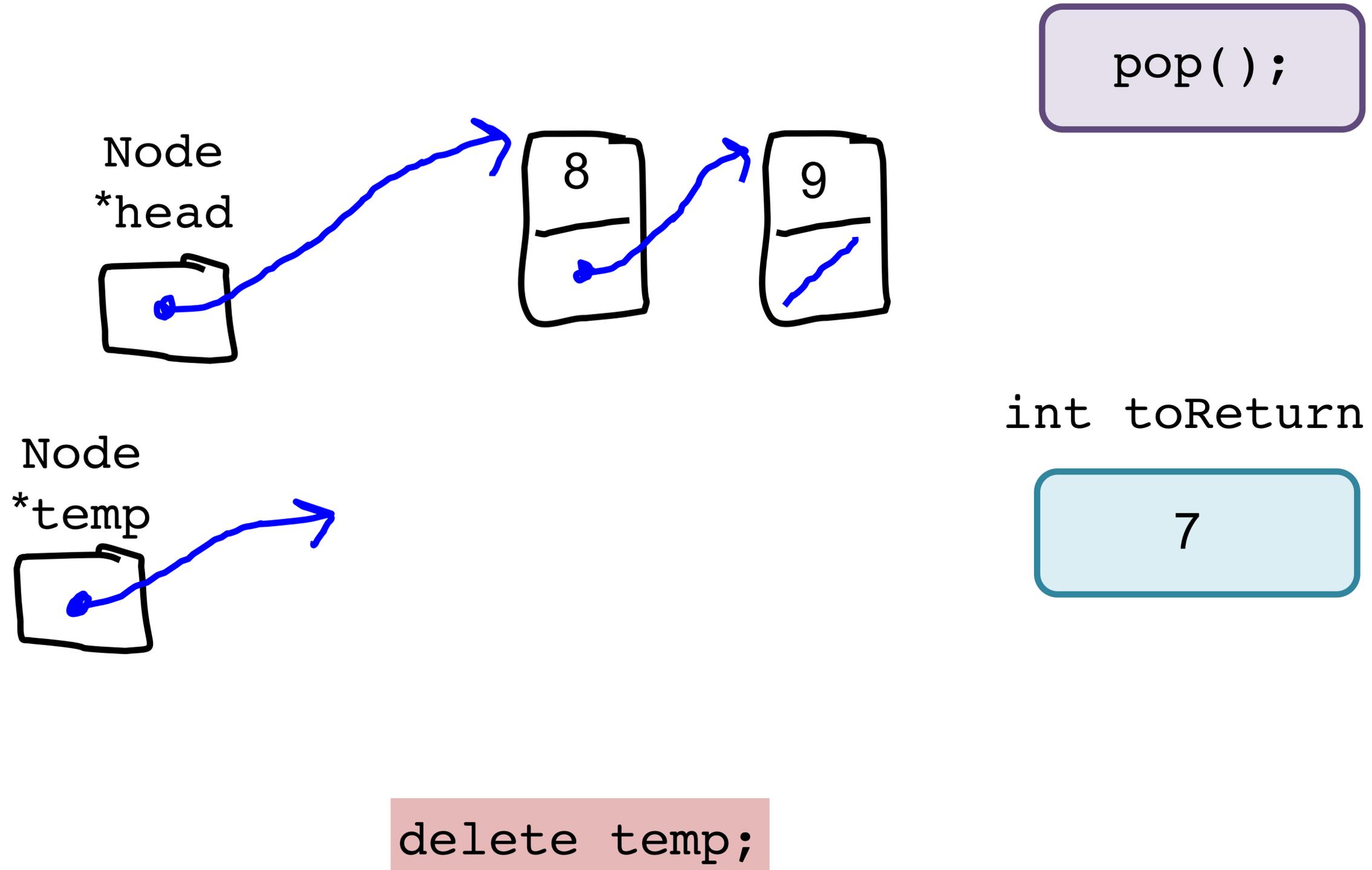
`int toReturn`

```
7
```

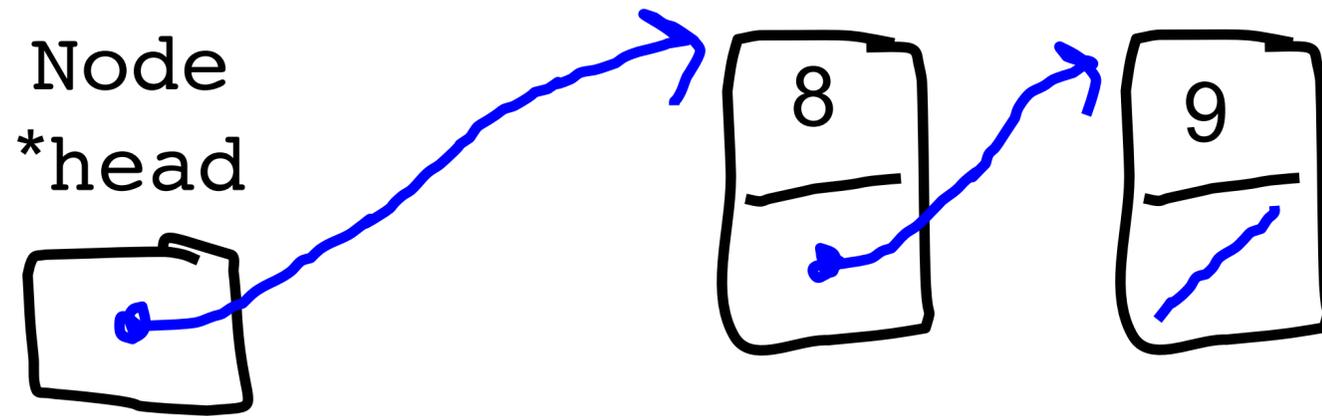
```
head = temp->link;
```



Stack is a Linked List



Stack is a Linked List



```
pop();
```

```
int toReturn
```

```
7
```

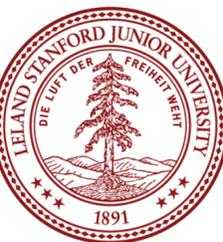
```
return toReturn;
```



Stack

```
#pragma once
```

```
class IntStack {  
public:  
    IntStack(); // constructor  
    ~IntStack();  
  
    bool isEmpty();  
  
    void push(int value);  
    int peek();  
    int pop();  
  
private:  
    struct Node {  
        int value;  
        Node* next;  
    };  
  
    Node* head;  
};
```



Stack Implementation

```
void IntStack::push(int value) {
    Node* node = new Node;
    // pushing "value"
    node->value = value;

    node->next = head;

    head = node;
}

int IntStack::pop() {
    if (isEmpty()) {
        throw "Error! Trying to pop from empty stack!";
    }

    Node* result = head;
    head = head->next;

    int value = result->value;
    delete result;
    return value;
}
```



Stack Implementation: Big O?

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

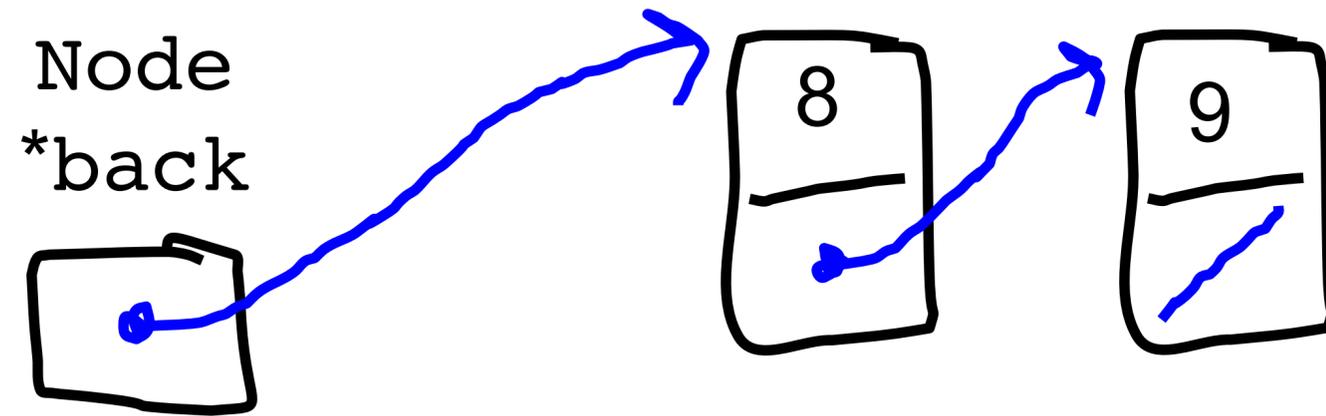
Big O of `pop()`? $O(1)$

Yay!

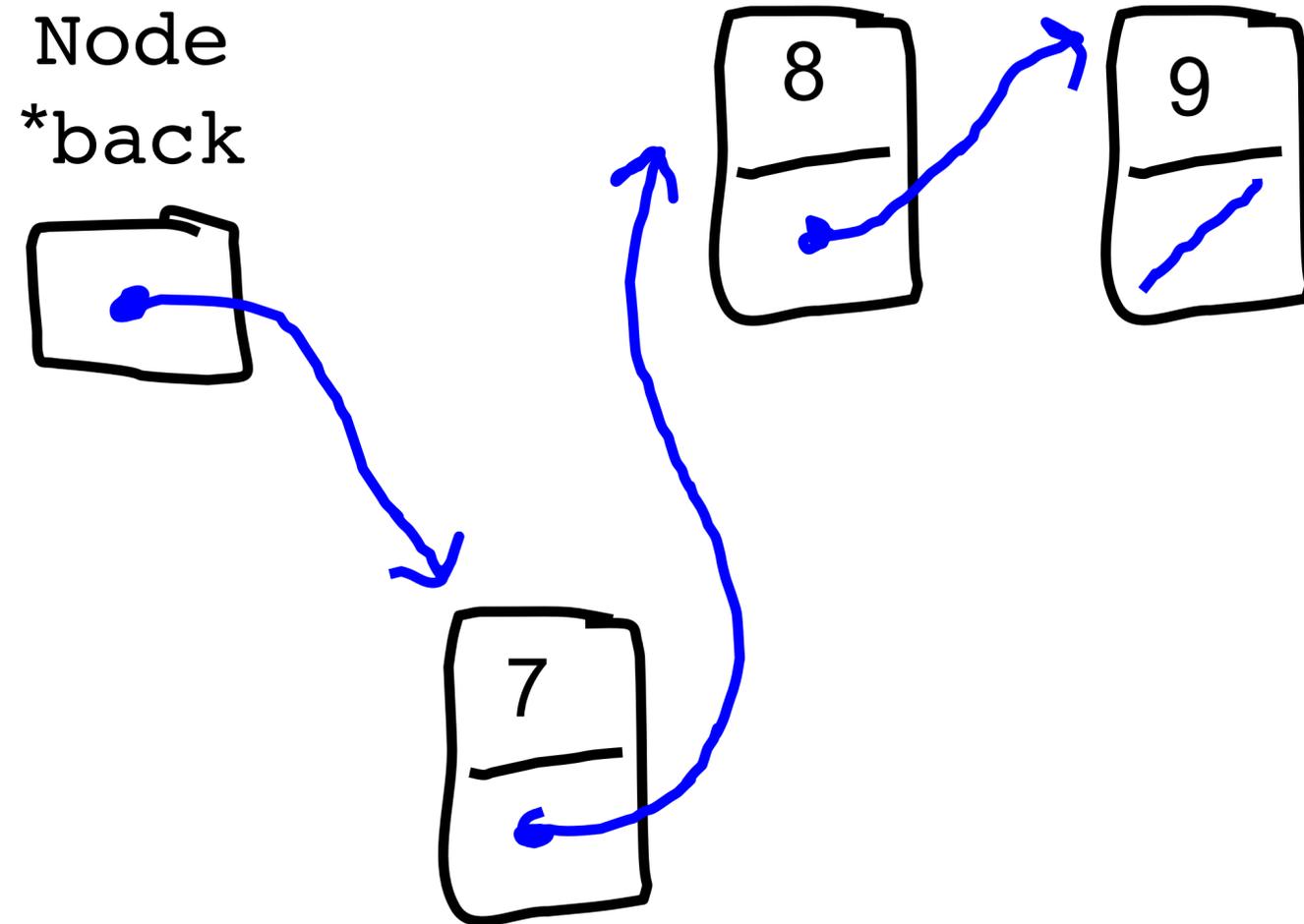




Queue?

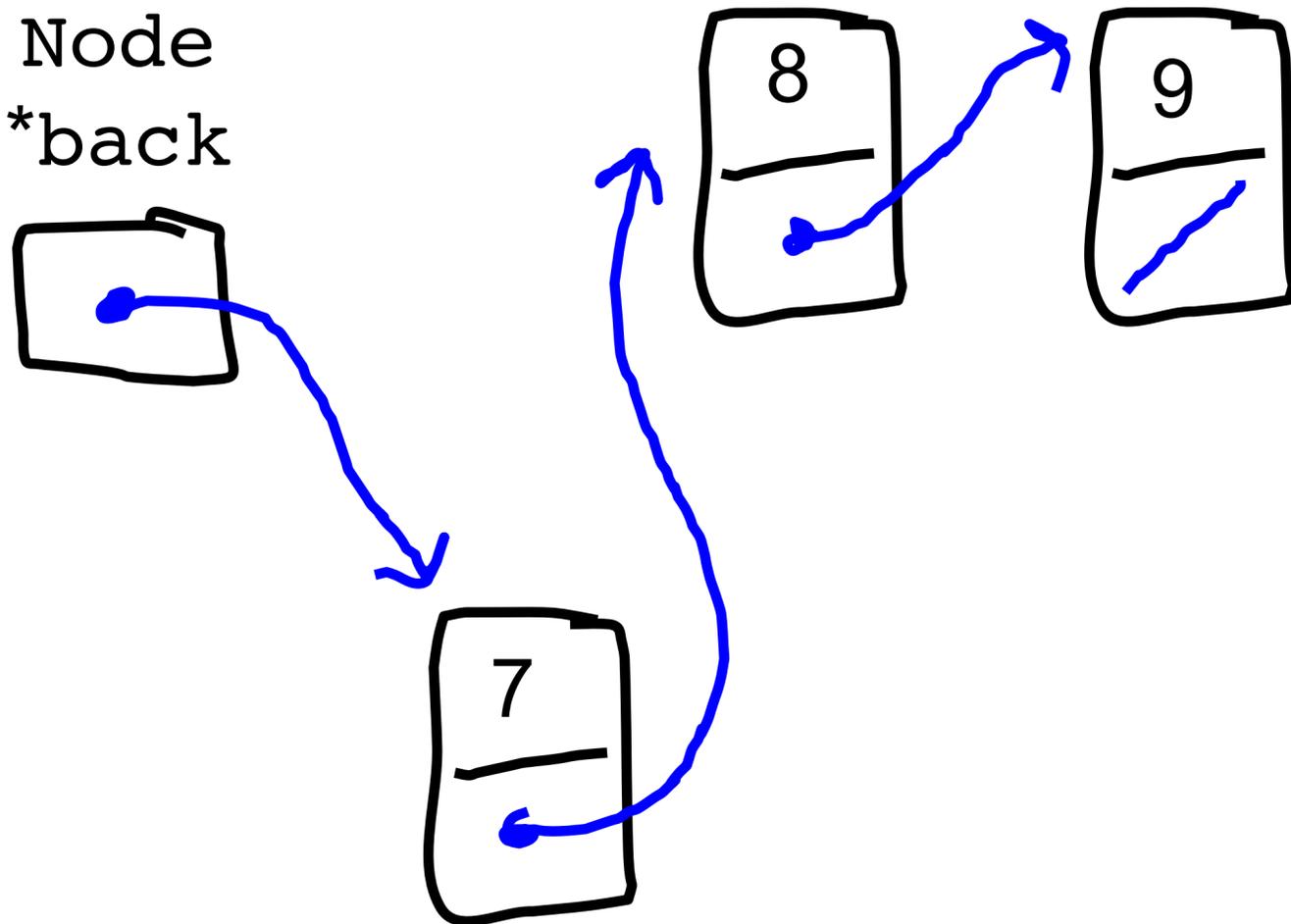


Queue Enqueue?



Queue Enqueue?

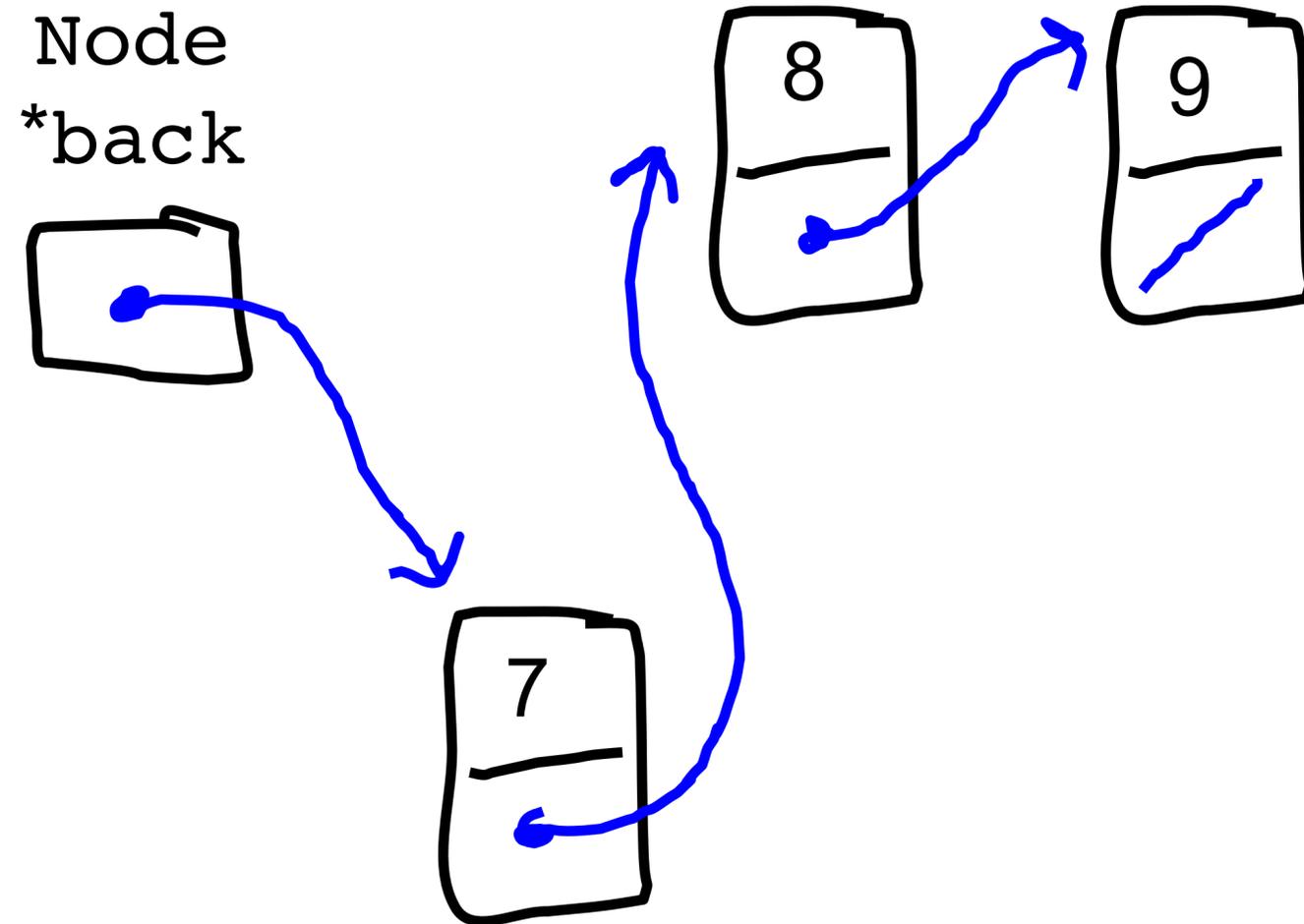
Node
*back



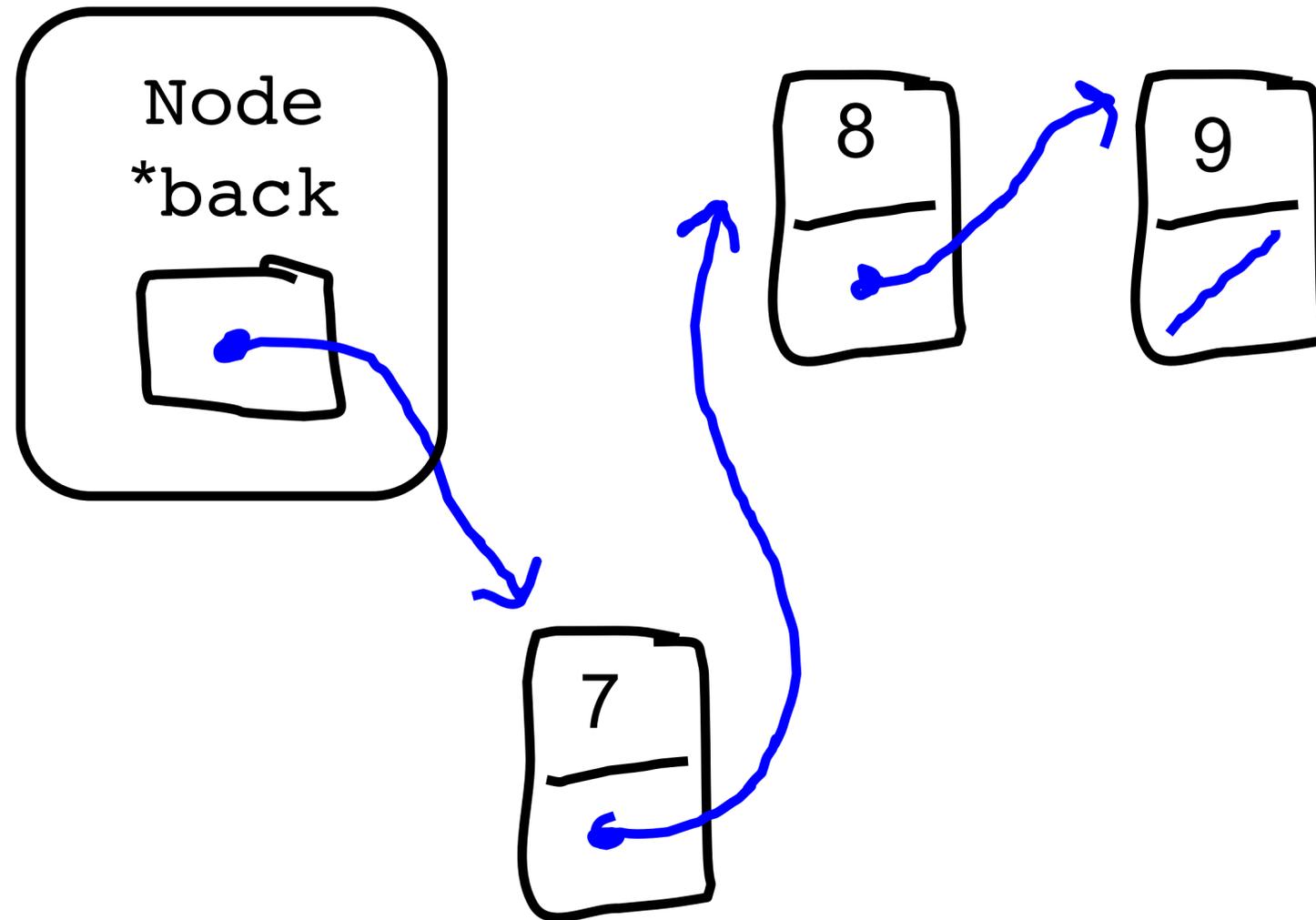
$O(1)$



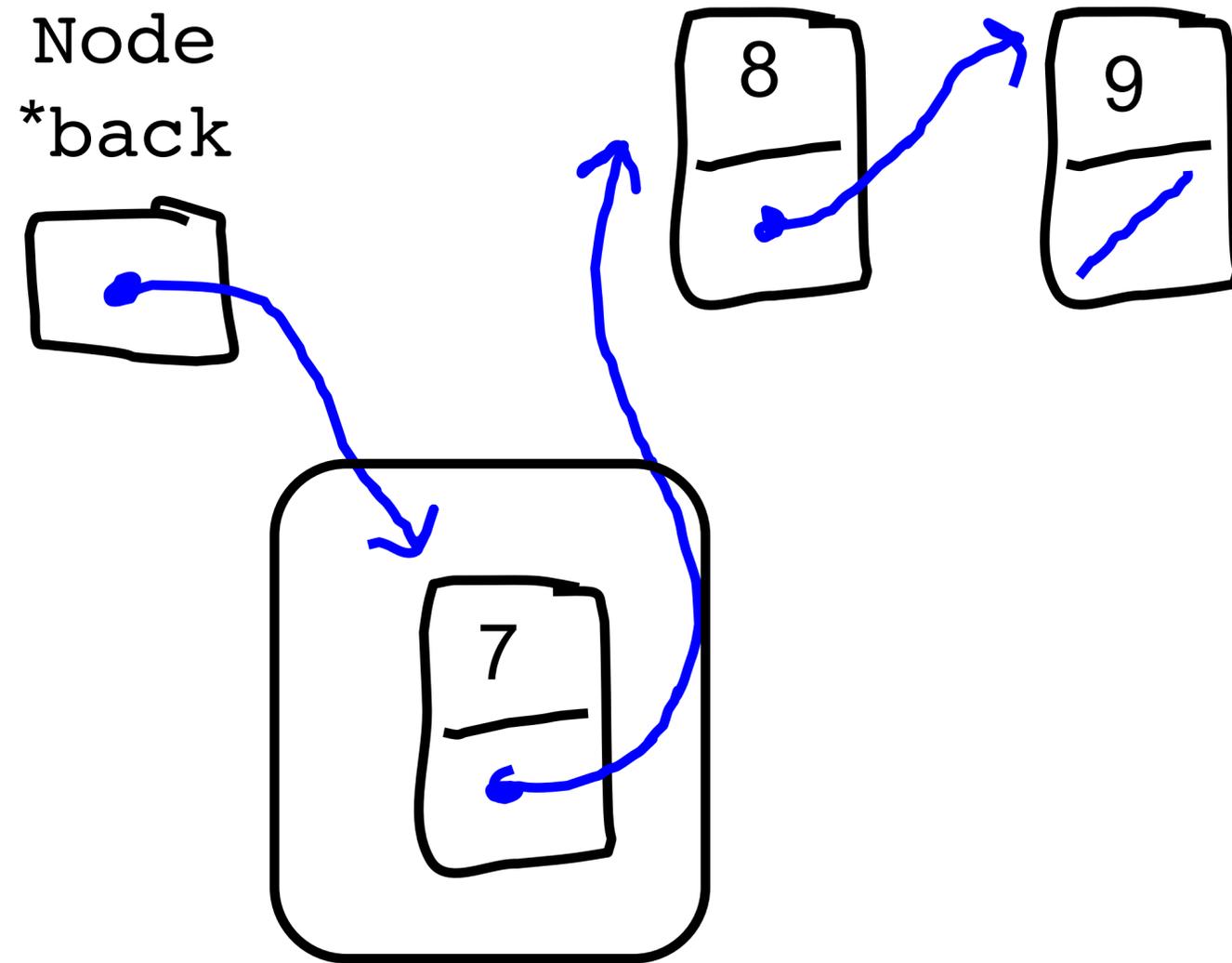
Queue Dequeue?



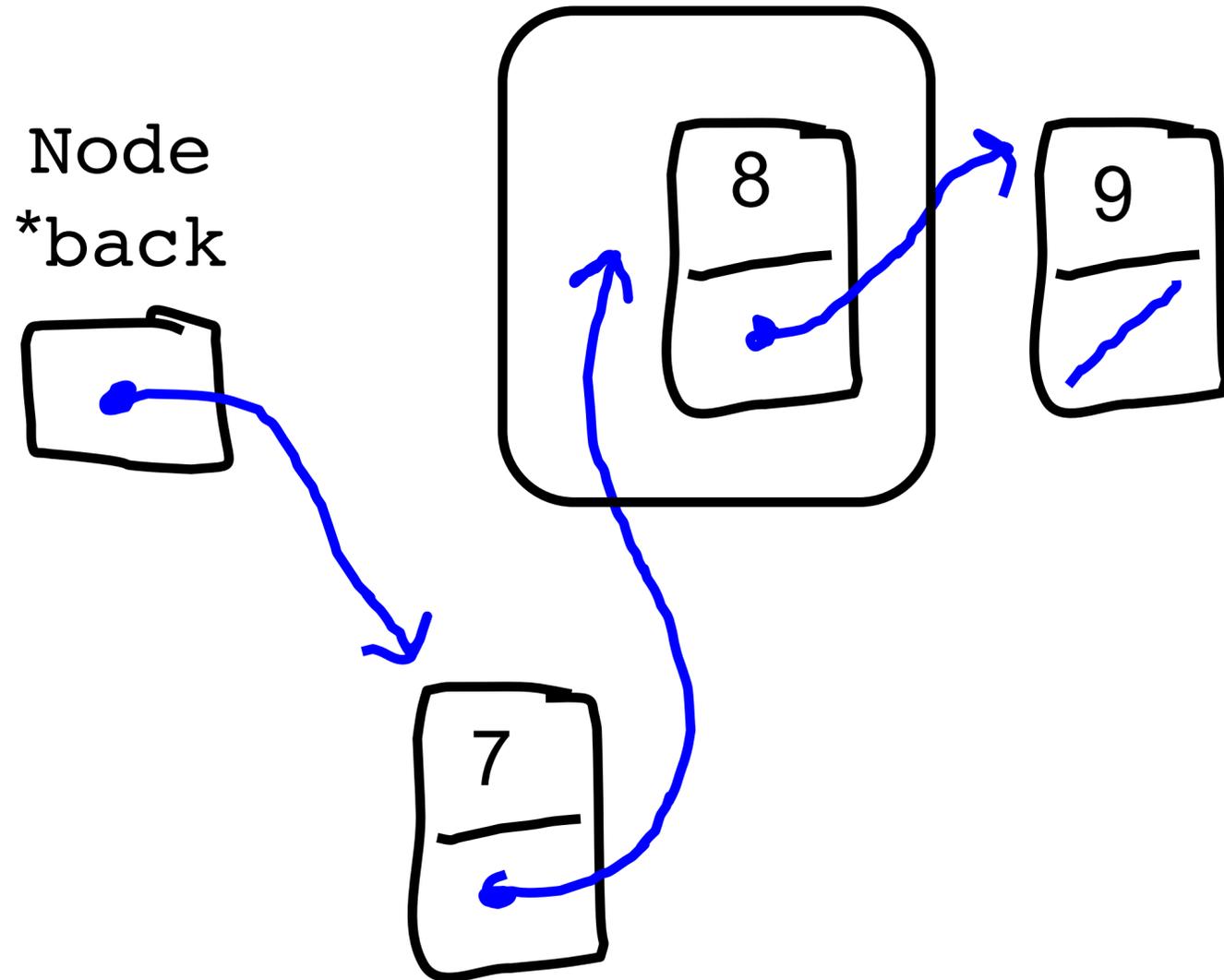
Queue Dequeue?



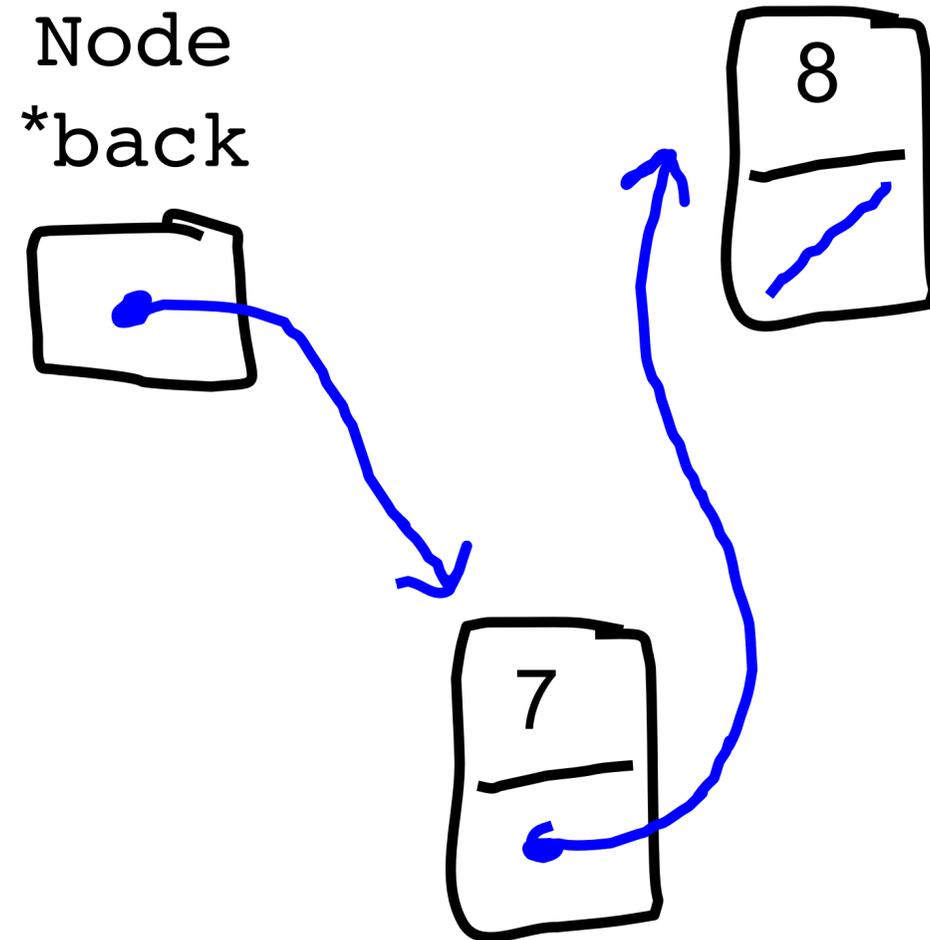
Queue Dequeue?



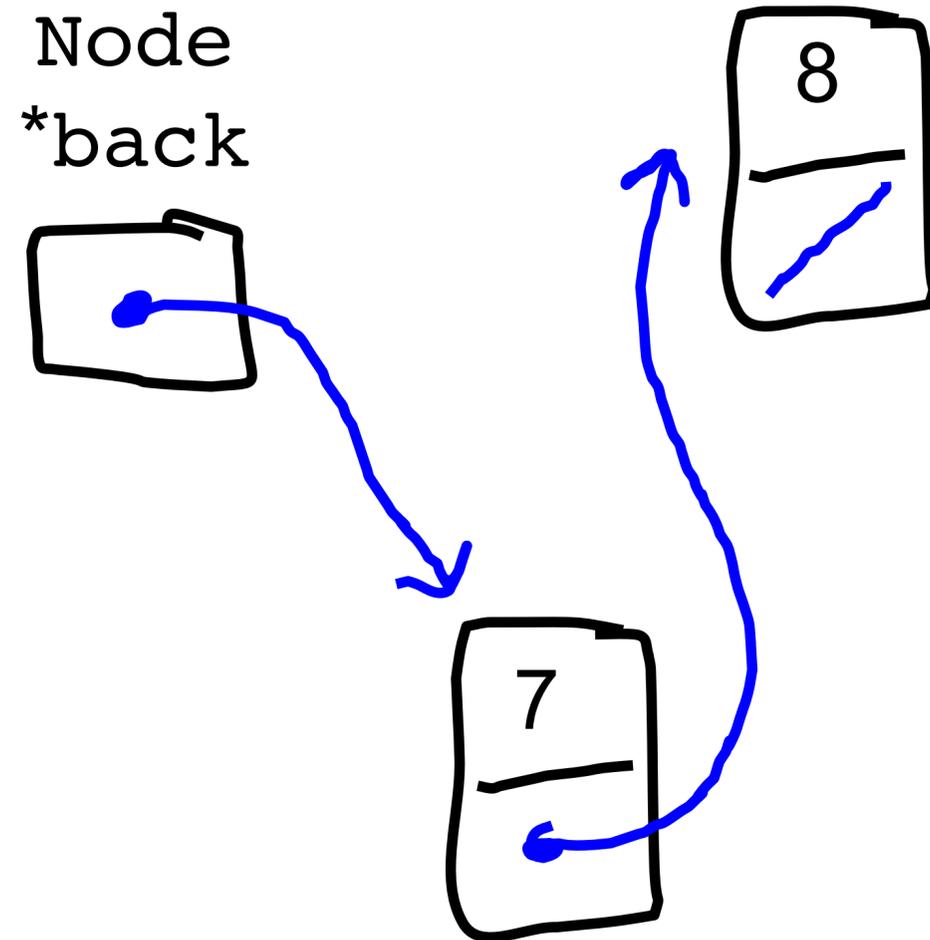
Queue Dequeue?



Queue Dequeue?



Queue Dequeue?



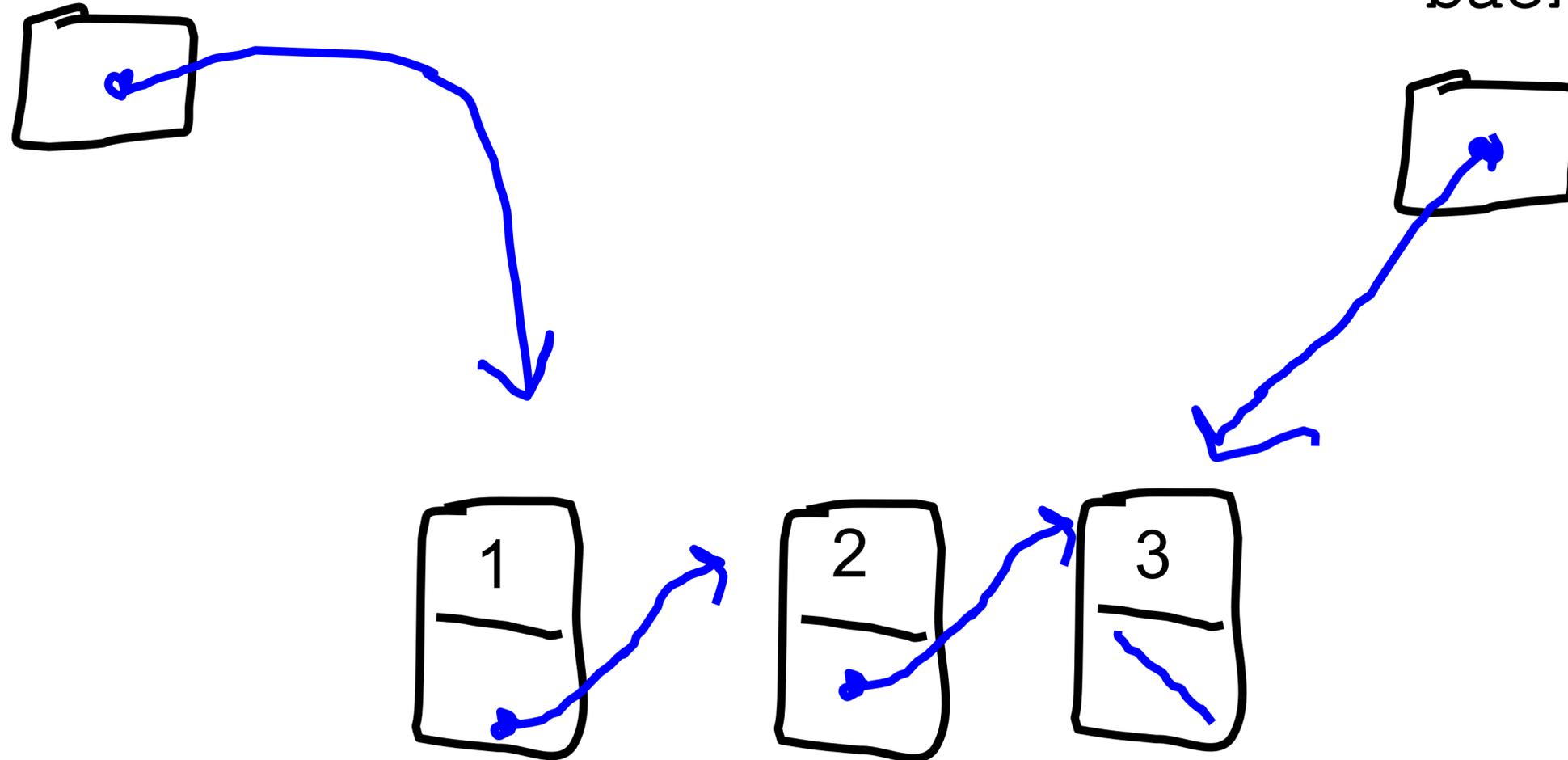
$O(n)$

Always a Better Way

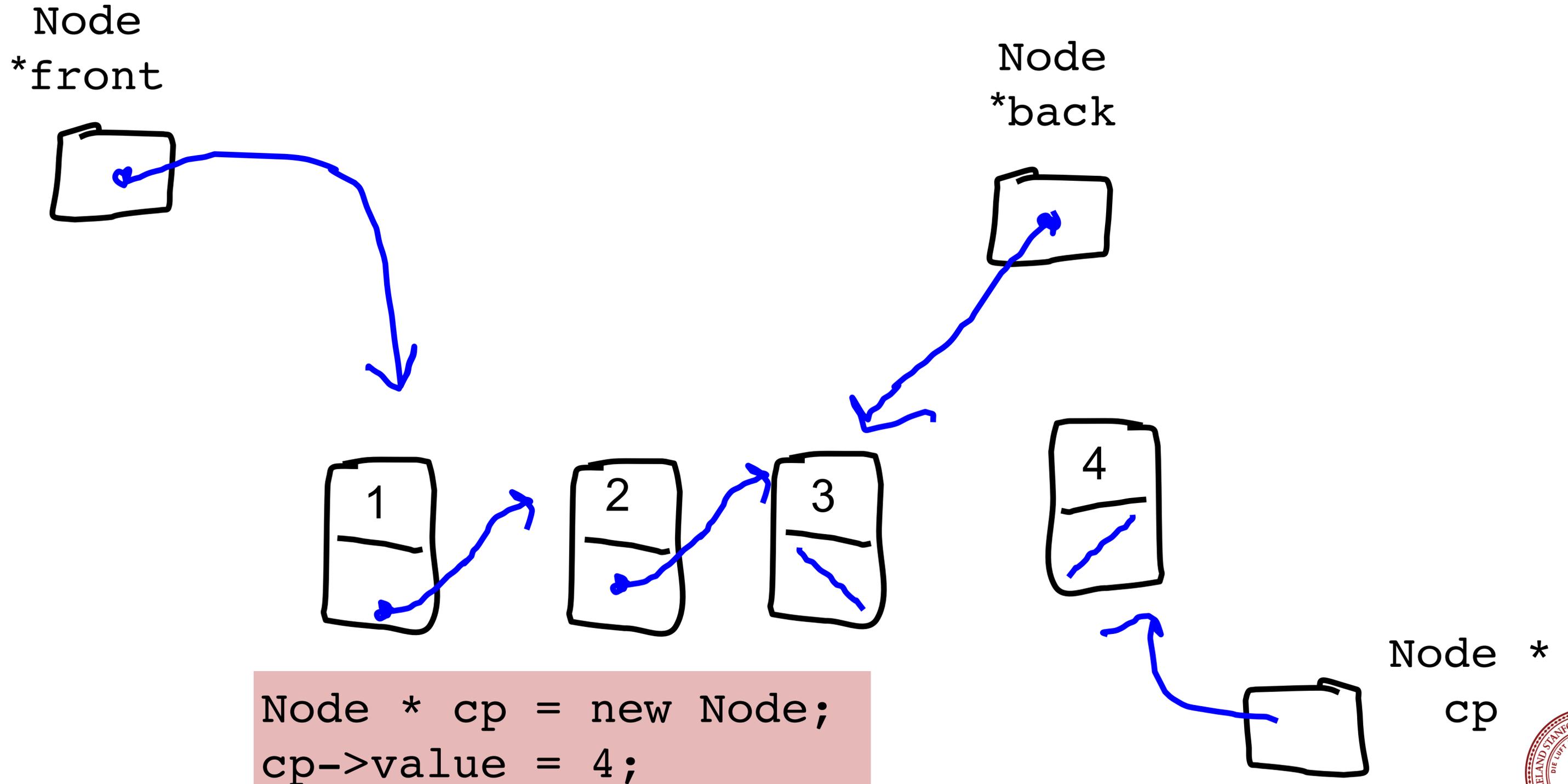
Actual Queue: Enqueue

Node
*front

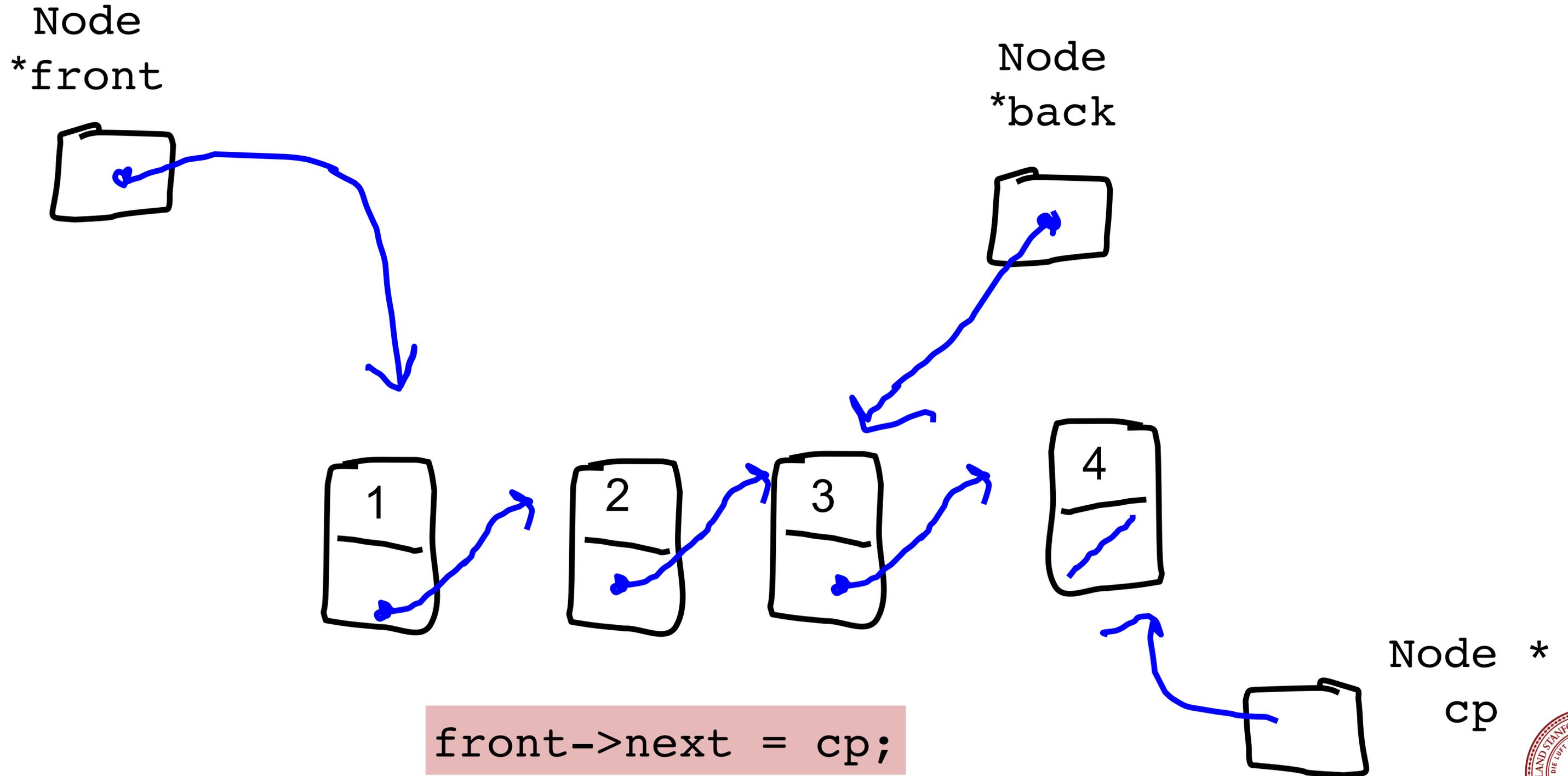
Node
*back



Actual Queue: Enqueue



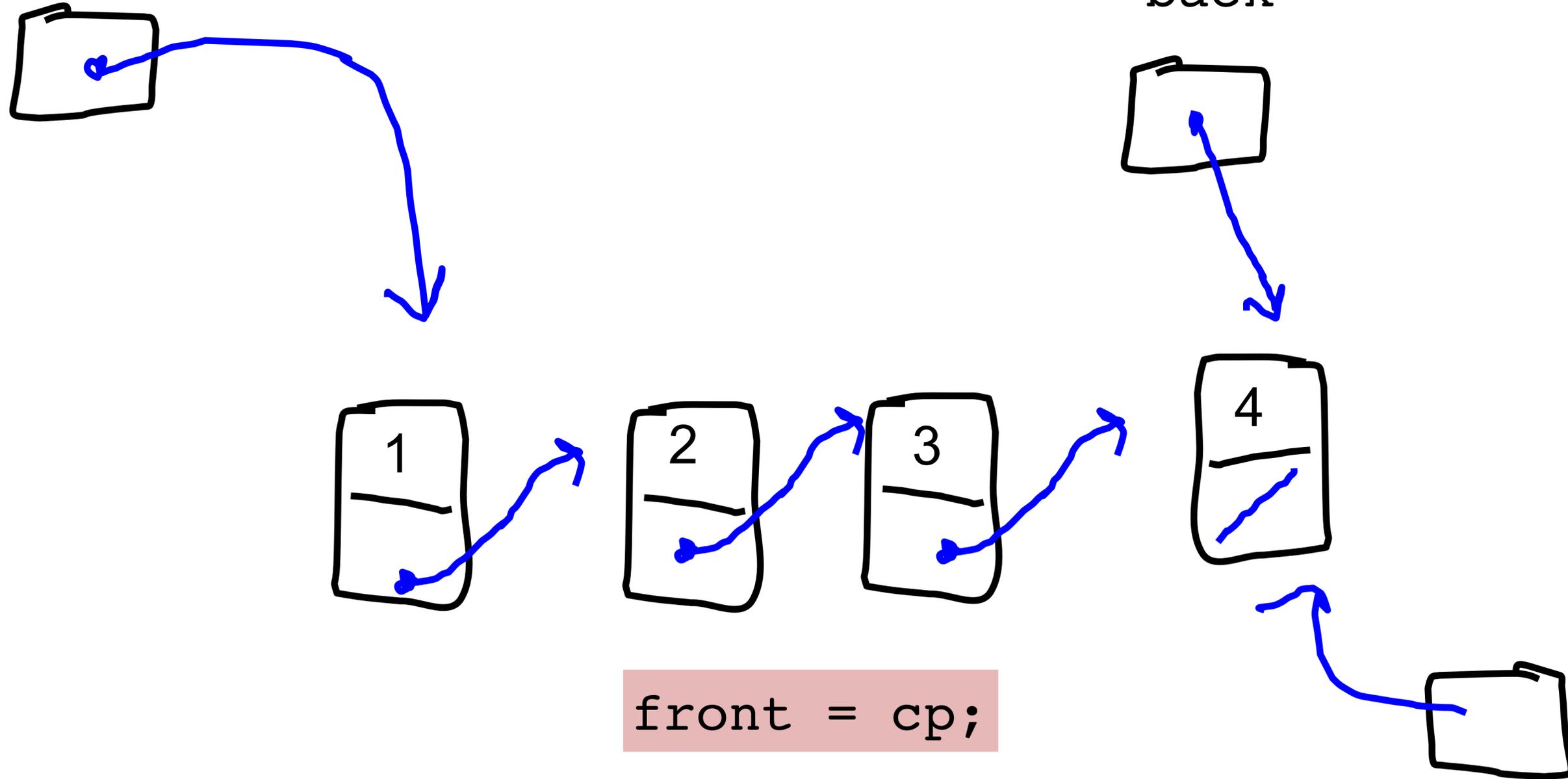
Actual Queue: Enqueue



Actual Queue: Enqueue

Node
*front

Node
*back

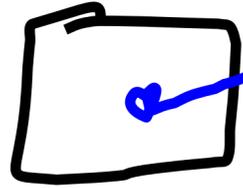


Node *
cp

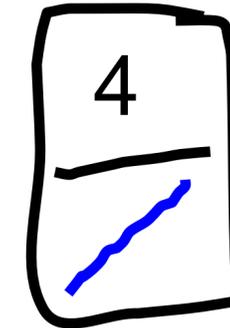
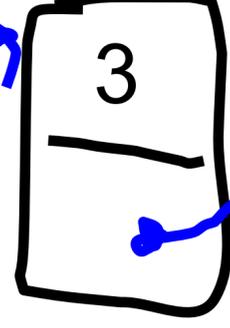
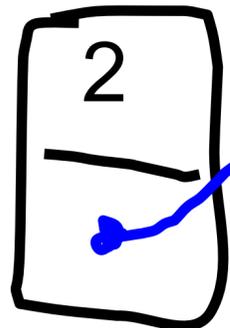
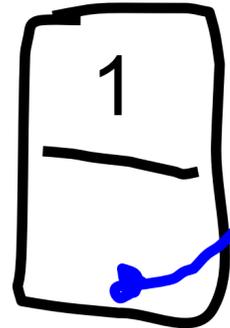


Actual Queue: Enqueue

Node
*front



Node
*back



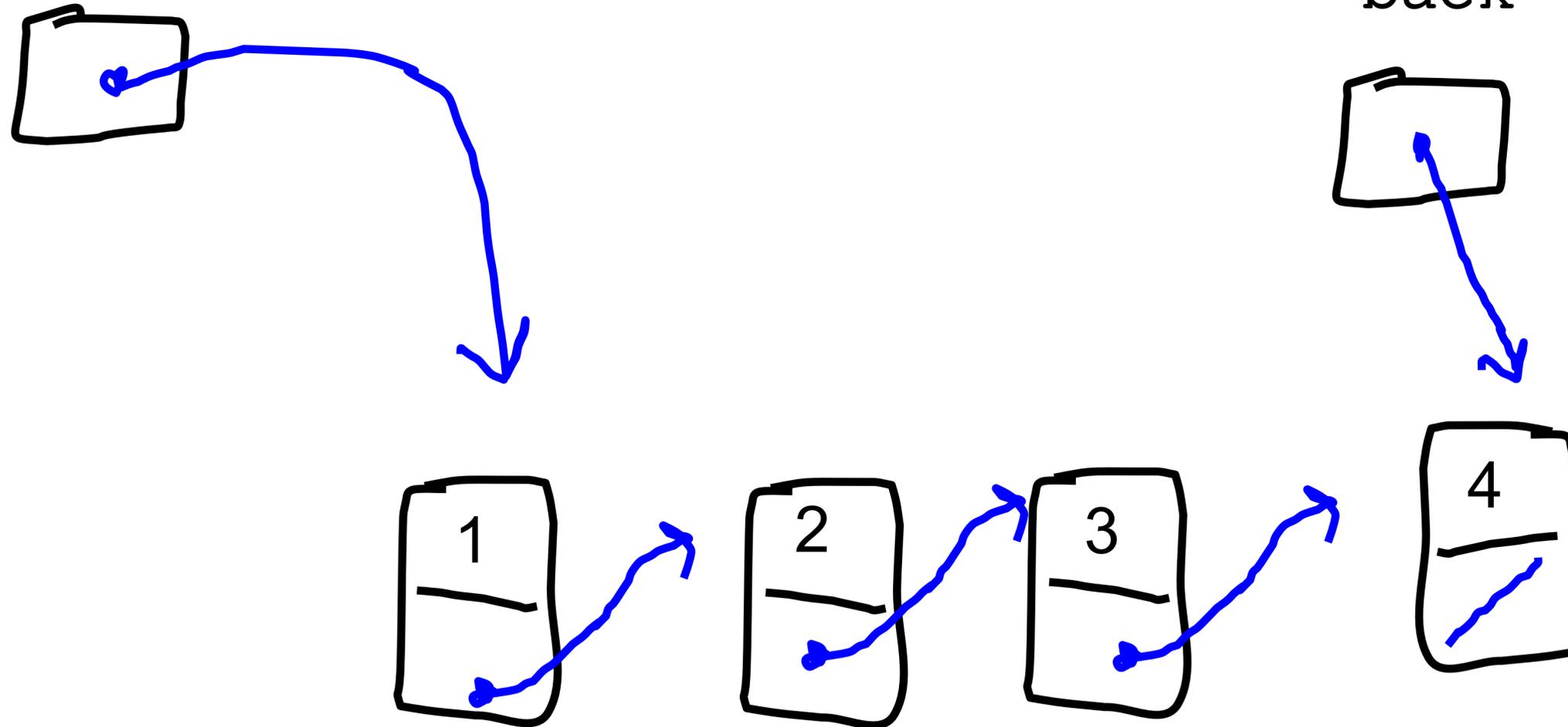
```
return;
```



Actual Queue: Enqueue

Node
*front

Node
*back



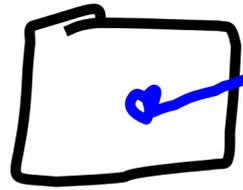
$O(1)$



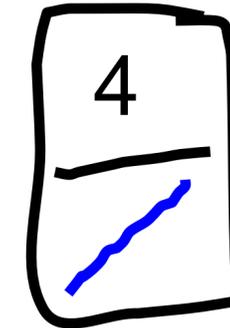
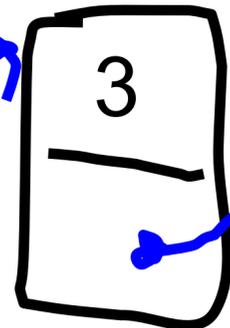
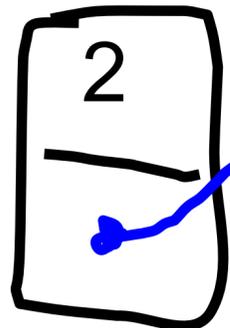
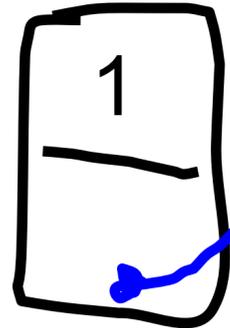
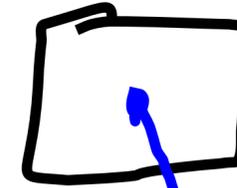
Dequeue

Actual Queue: Dequeue

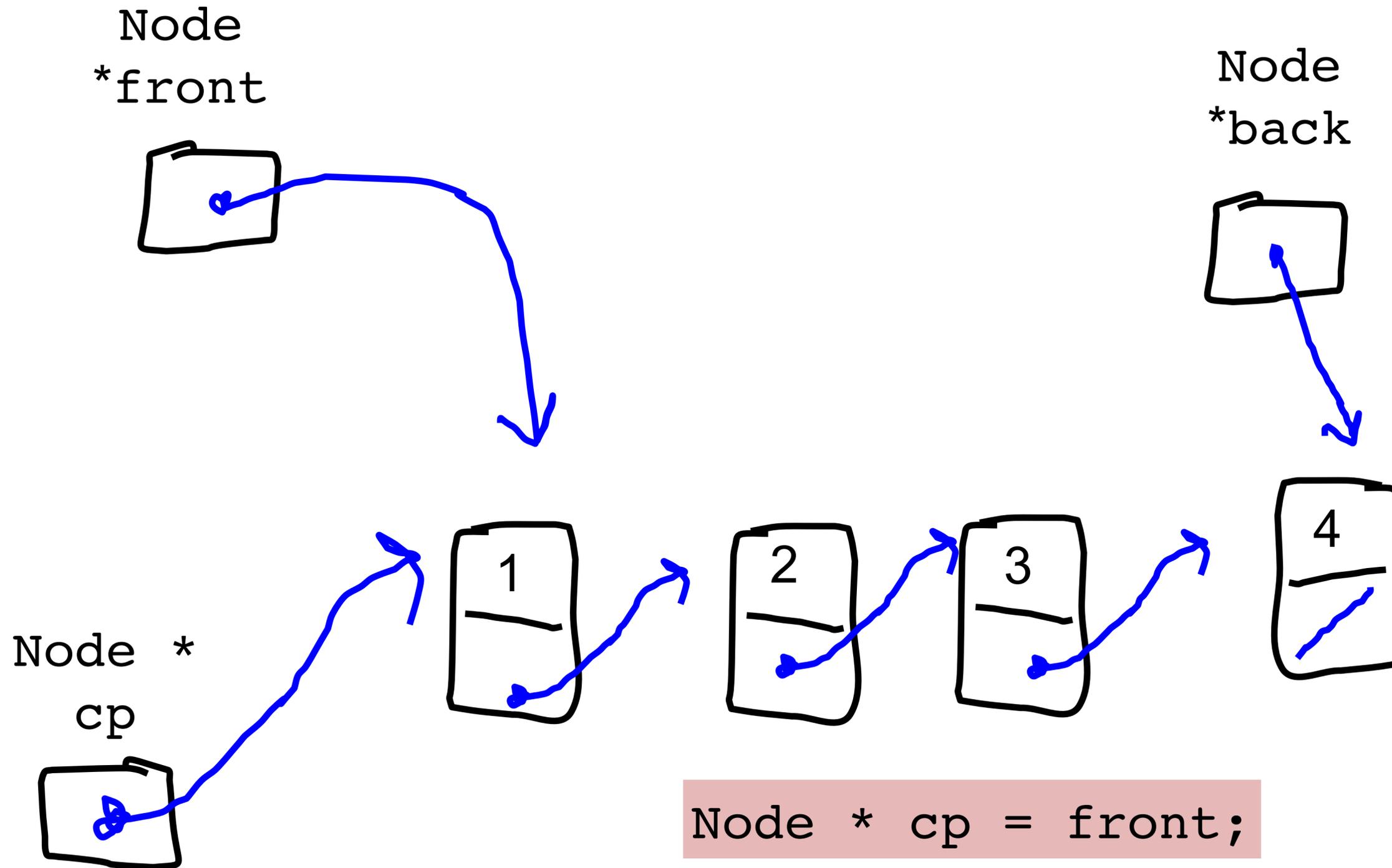
Node
*front



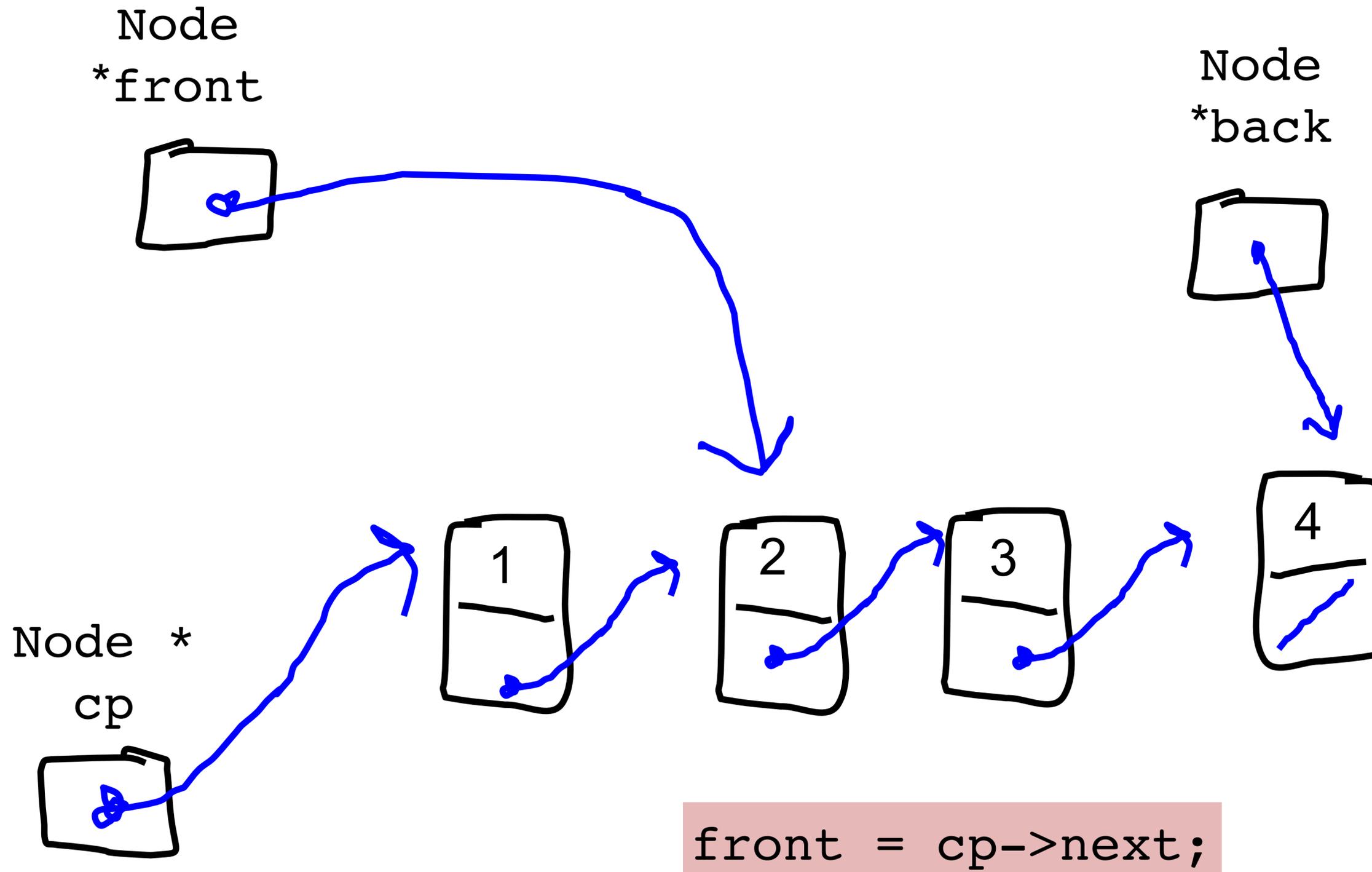
Node
*back



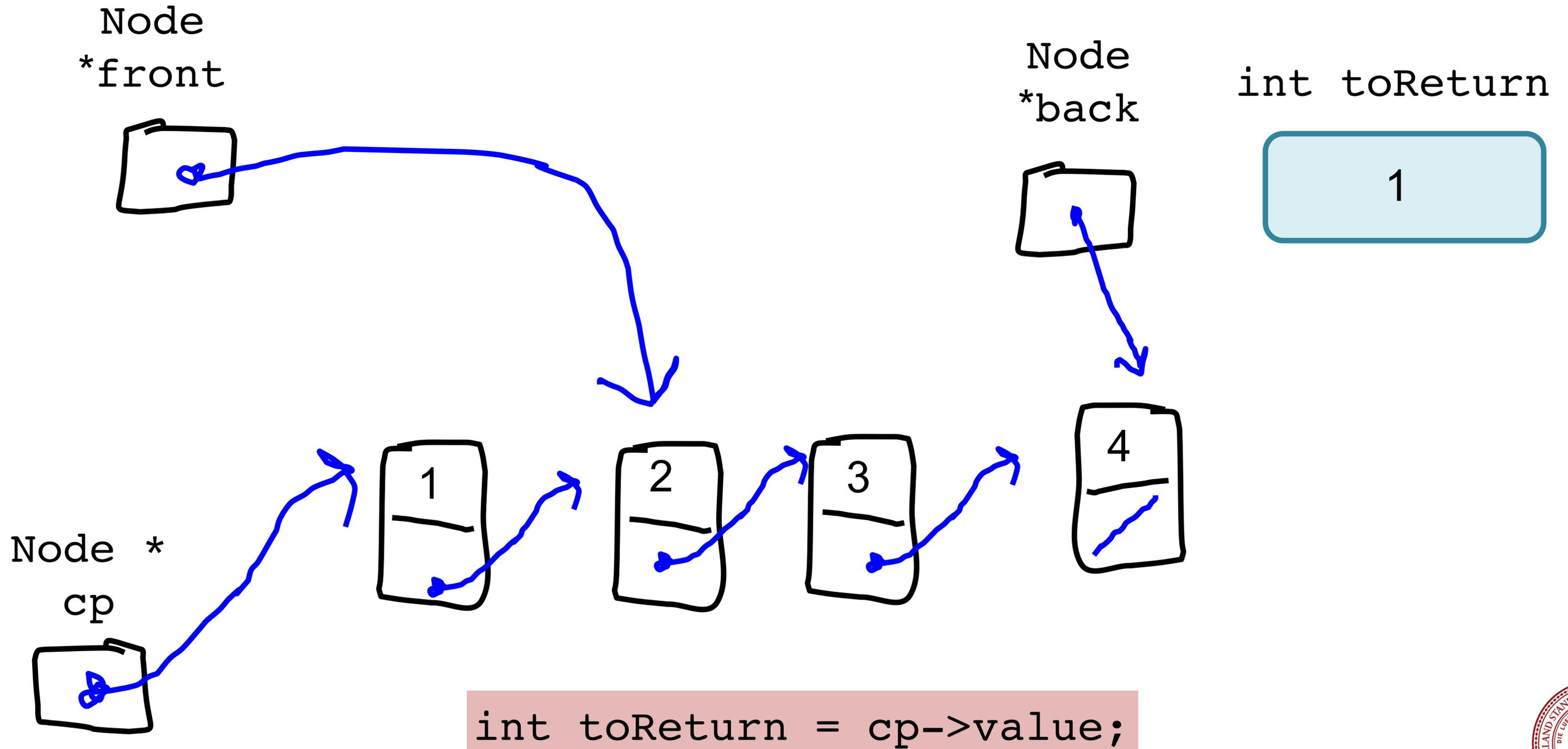
Actual Queue: Dequeue



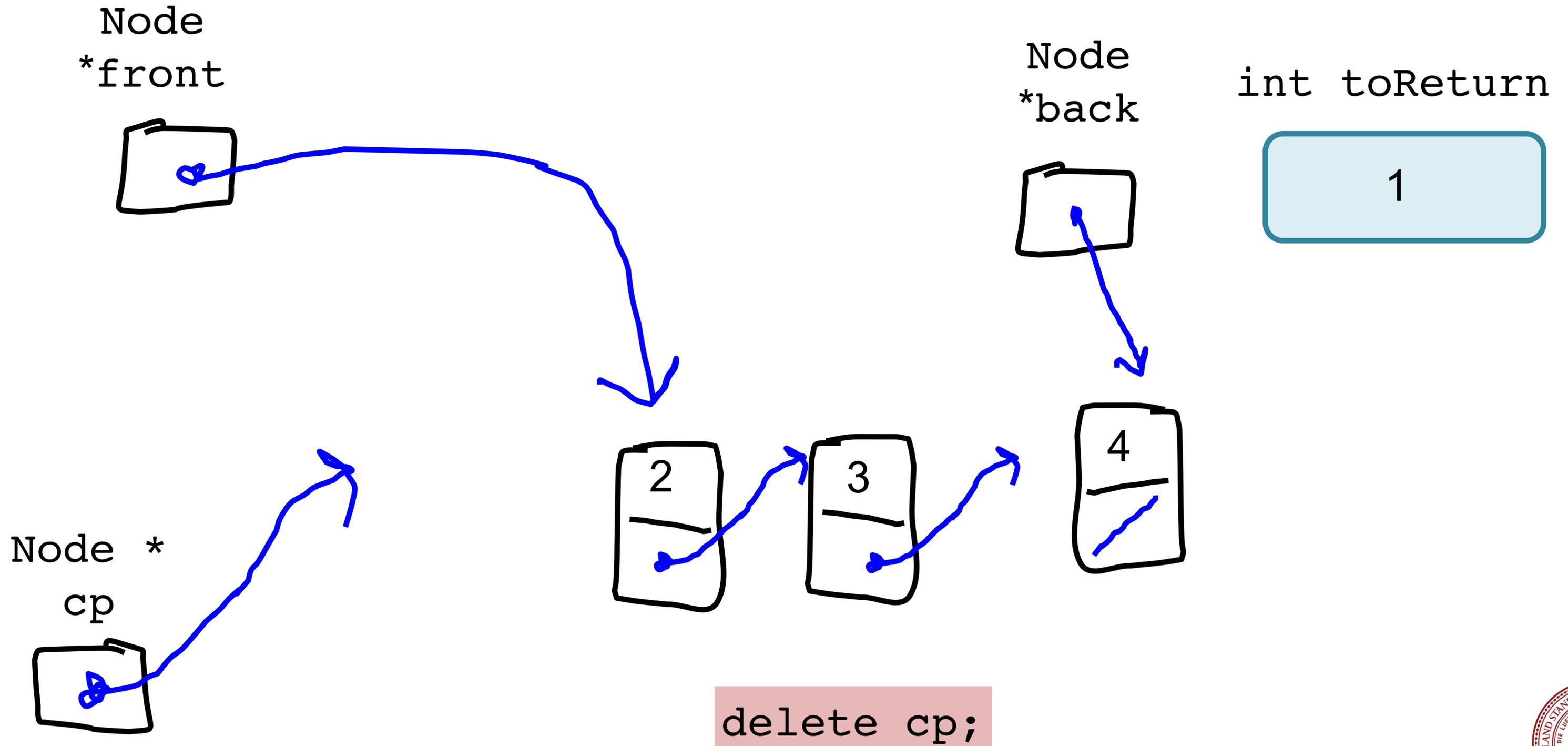
Actual Queue: Dequeue



Actual Queue: Dequeue

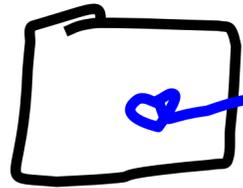


Actual Queue: Dequeue

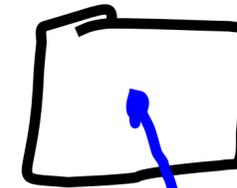


Actual Queue: Dequeue

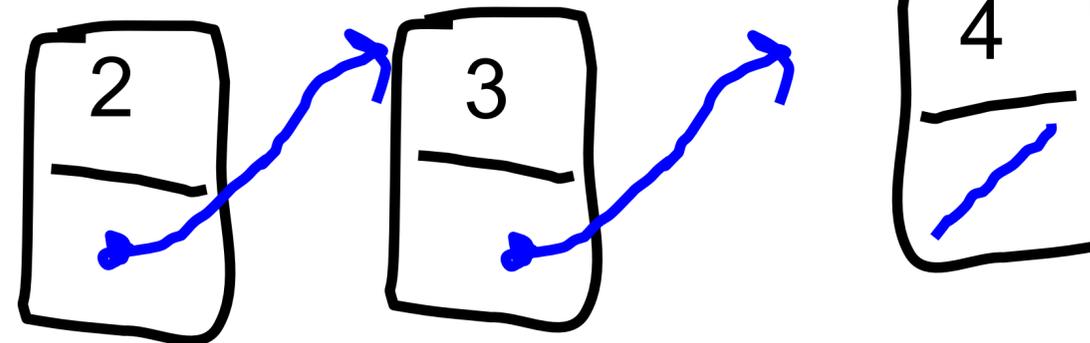
Node
*front



Node
*back



int toReturn

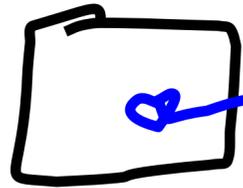


```
return toReturn;
```

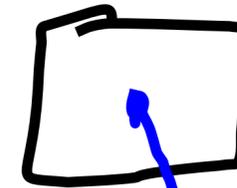


Actual Queue: Dequeue

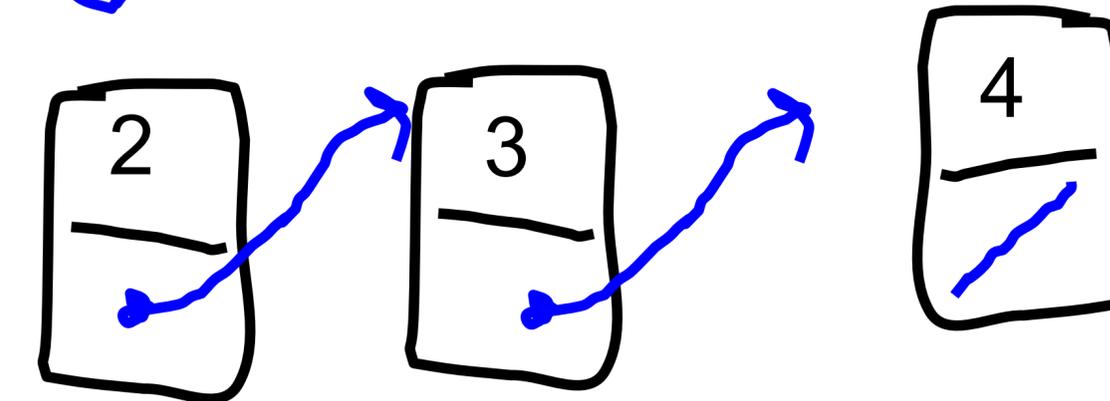
Node
*front



Node
*back



$O(1)$



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 *next;
    };
    Node * back; // has a pointer to the first node
    Node * front; // and a pointer to the last node
};
```



Queue Implementation

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

```
int QueueInt::dequeue() {  
    int toReturn = front->value;  
    Node * temp = front;  
    front = temp->next;  
    if (front == nullptr) {  
        back = nullptr;  
    }  
    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

$\mathcal{O}(1)$

