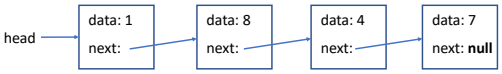
Linked list visualized

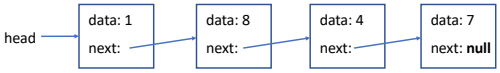
[1, 8, 4, 7]



Look at code

1

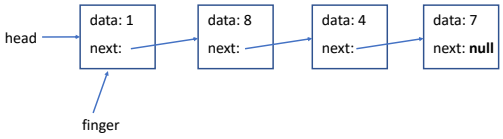
Iterating through a linked list



How can we iterate through a linked list?

2

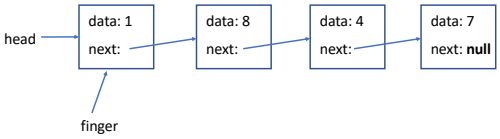
Iterating through a linked list



Use a variable starting at the head and move it down the list

3

Iterating through a linked list

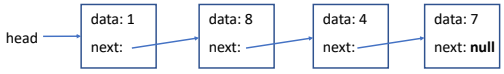


Use a variable starting at the head and move it down the list

How will we know when we're at the end?

4

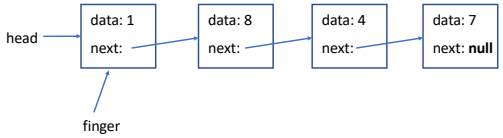
Iterating through a linked list: printing it




```
Node finger = head;

while (finger != null ) {
    System.out.println(finger.value());
    finger = finger.next();
}
```

5

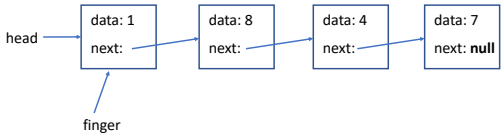
Iterating through a linked list: printing it




```
Node finger = head;

while (finger != null ) {
    System.out.println(finger.value());
    finger = finger.next();
}
```

6

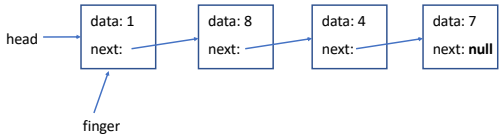
Iterating through a linked list: printing it




```
Node finger = head;

while (finger != null ) {
    System.out.println(finger.value());
    finger = finger.next();
}
```

7

Iterating through a linked list: printing it

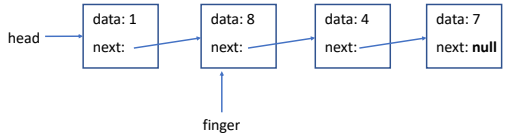


1

```
Node finger = head;

while (finger != null ) {
    System.out.println(finger.value());
    finger = finger.next();
}
```

8

Iterating through a linked list: printing it

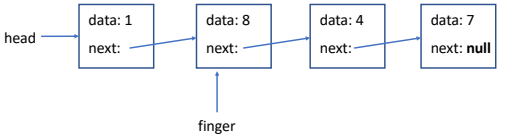


1

```
Node finger = head;

while (finger != null ) {
    System.out.println(finger.value());
    finger = finger.next();
}
```

9

Iterating through a linked list: printing it

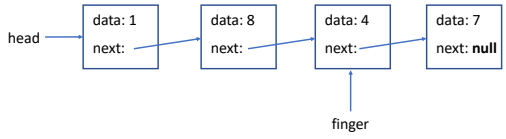


1
8

```
Node finger = head;

while (finger != null ) {
    System.out.println(finger.value());
    finger = finger.next();
}
```

10

Iterating through a linked list: printing it

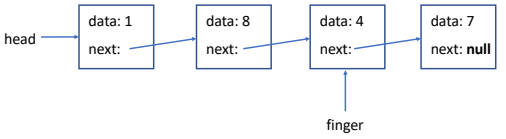


1
8

```
Node finger = head;

while (finger != null ) {
    System.out.println(finger.value());
    finger = finger.next();
}
```

11

Iterating through a linked list: printing it

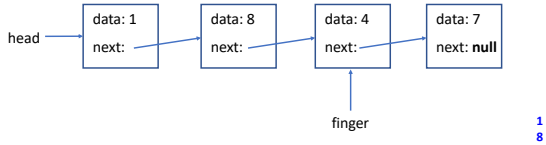


1
8
4

```
Node finger = head;

while (finger != null ) {
    System.out.println(finger.value());
    finger = finger.next();
}
```

12

Iterating through a linked list: printing it

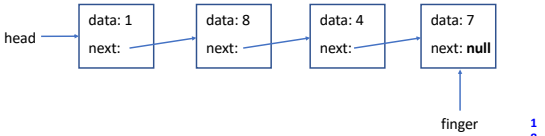


1
8
4

```
Node finger = head;

while (finger != null ) {
    System.out.println(finger.value());
    finger = finger.next();
}
```

13

Iterating through a linked list: printing it

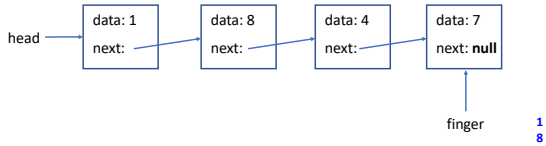


1
8
4

```
Node finger = head;

while (finger != null ) {
    System.out.println(finger.value());
    finger = finger.next();
}
```

14

Iterating through a linked list: printing it

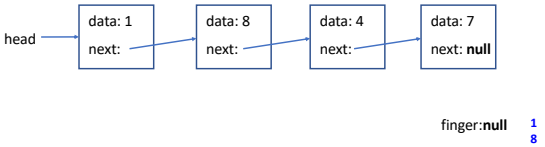


1
8
4
7

```
Node finger = head;

while (finger != null ) {
    System.out.println(finger.value());
    finger = finger.next();
}
```

15

Iterating through a linked list: printing it

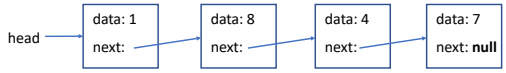


1
8
4
7

```
Node finger = head;

while (finger != null ) {
    System.out.println(finger.value());
    finger = finger.next();
}
```

16

Iterating through a linked list: printing it



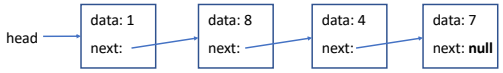
finger:**null**

1
8
4
7

```
Node finger = head;

while (finger != null ) {
    System.out.println(finger.value());
    finger = finger.next();
}
```

17

Checking if a value is in the list



Write a method contains that takes a value as input and returns a boolean indicating whether or not the linked list contains that value:

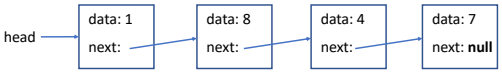
```
public boolean contains(E value){
```

```
Node finger = head;

while (finger != null ) {
    System.out.println(finger.value());
    finger = finger.next();
}
```

18

Adding to the end

[1, 8, 4, 7]

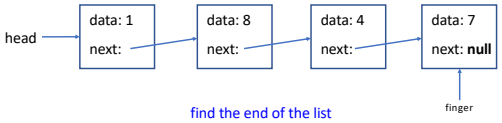


How can we add a value to the end of the list?

19

Adding to the end: add 9 to the end

[1, 8, 4, 7]



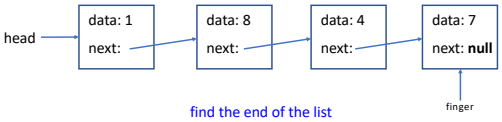
find the end of the list

How do we know we're at the end?

20

Adding to the end: add 9 to the end

[1, 8, 4, 7]



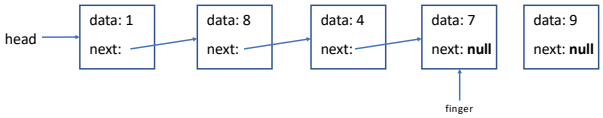
find the end of the list

finger.next() == null

21

Adding to the end: add 9 to the end

[1, 8, 4, 7]

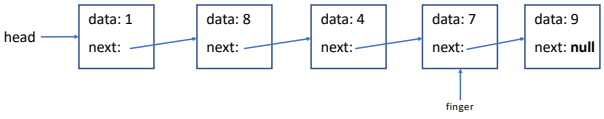


create a new node and add it to the end

22

Adding to the end: add 9 to the end

[1, 8, 4, 7]

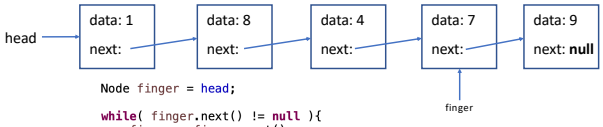


create a new node and add it to the end

23

Adding to the end: add 9 to the end

[1, 8, 4, 7]

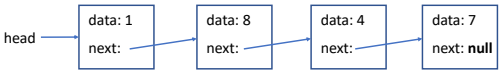



```
Node finger = head;

while( finger.next() != null ){
    finger = finger.next();
}

finger.setNext(new Node(value));
```

24

Adding to the end

[1, 8, 4, 7]




```
Node finger = head;

while( finger.next() != null ){
    finger = finger.next();
}

finger.setNext(new Node(value));
```

Any special cases we need to handle?

25

Adding to the end

[1, 8, 4, 7]

head: **null**

What would happen here?

```
Node finger = head;

while( finger.next() != null ){
    finger = finger.next();
}

finger.setNext(new Node(value));
```

26

Adding to the end

[1, 8, 4, 7]

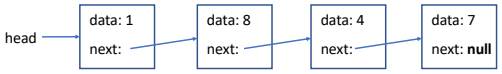
head: **null**

```
public void addLast(E value){
    if( head == null ){
        head = new Node(value);
    }else{
        Node finger = head;

        while( finger.next() != null ){
            finger = finger.next();
        }

        finger.setNext(new Node(value));
    }
}
```

27

Removing a value

[1, 8, 4, 7]

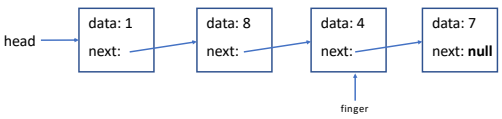


How can we we remove a value from the list?

28

Removing a value: remove 4

[1, 8, 4, 7]

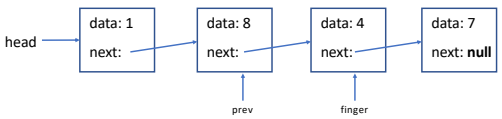


Find it

29

Removing a value: remove 4

[1, 8, 4, 7]

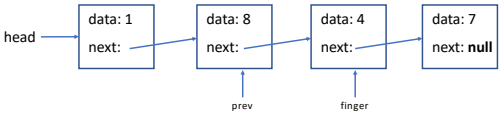


Need the node before it too!

30

Removing a value: remove 4

[1, 8, 4, 7]

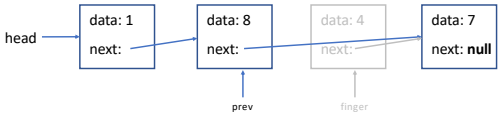


splice it out

31

Removing a value: remove 4

[1, 8, 4, 7]

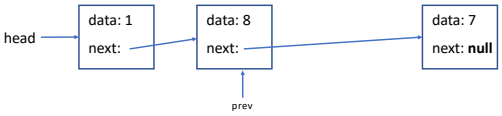


splice it out

32

Removing a value: remove 4

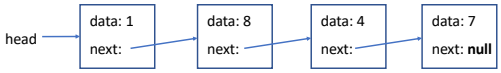
[1, 8, 4, 7]



splice it out

33

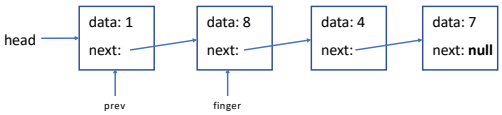
Removing a value




```
Node finger = head.next();
Node prev = head;

while (finger != null && !finger.value().equals(value)){
    prev = finger;
    finger = finger.next();
}

if( finger != null ){
    prev.setNext(finger.next());
}
```

34

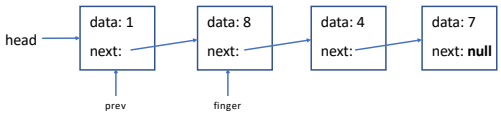
Removing a value




```
Node finger = head.next();
Node prev = head;

while (finger != null && !finger.value().equals(value)){
    prev = finger;
    finger = finger.next();
}

if( finger != null ){
    prev.setNext(finger.next());
}
```

35

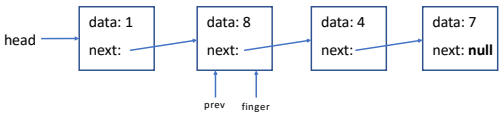
Removing a value




```
Node finger = head.next();
Node prev = head;

while (finger != null && !finger.value().equals(value)){
    prev = finger;
    finger = finger.next();
}

if( finger != null ){
    prev.setNext(finger.next());
}
```

36

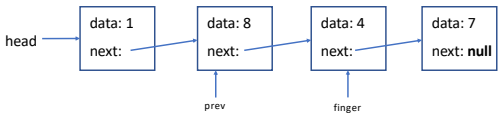
Removing a value




```
Node finger = head.next();
Node prev = head;

while (finger != null && !finger.value().equals(value)){
    prev = finger;
    finger = finger.next();
}

if( finger != null ){
    prev.setNext(finger.next());
}
```

37

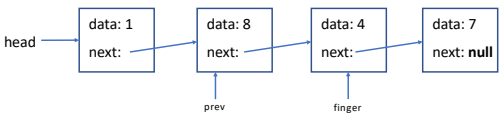
Removing a value




```
Node finger = head.next();
Node prev = head;

while (finger != null && !finger.value().equals(value)){
    prev = finger;
    finger = finger.next();
}

if( finger != null ){
    prev.setNext(finger.next());
}
```

38

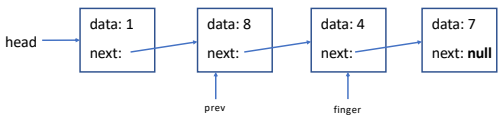
Removing a value




```
Node finger = head.next();
Node prev = head;

while (finger != null && !finger.value().equals(value)){
    prev = finger;
    finger = finger.next();
}

if( finger != null ){
    prev.setNext(finger.next());
}
```

39

Removing a value

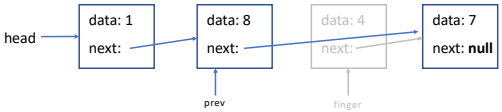



```
Node finger = head.next();
Node prev = head;

while (finger != null && !finger.value().equals(value)){
    prev = finger;
    finger = finger.next();
}

if( finger != null ){
    prev.setNext(finger.next());
}
```

When would finger be null?

40

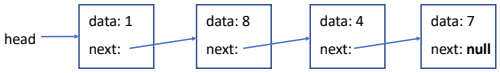
Removing a value




```
Node finger = head.next();
Node prev = head;

while (finger != null && !finger.value().equals(value)){
    prev = finger;
    finger = finger.next();
}

if( finger != null ){
    prev.setNext(finger.next());
}
```

41

Removing a value




```
Node finger = head.next();
Node prev = head;

while (finger != null && !finger.value().equals(value)){
    prev = finger;
    finger = finger.next();
}

if( finger != null ){
    prev.setNext(finger.next());
}
```

Any special cases we need to handle?

42

Removing a value

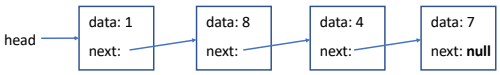
head: **null**

What would happen here?

```
Node finger = head.next();
Node prev = head;

while (finger != null && !finger.value().equals(value)){
    prev = finger;
    finger = finger.next();
}

if( finger != null ){
    prev.setNext(finger.next());
}
```

43

Removing a value

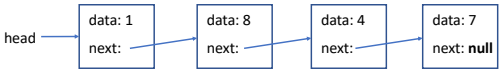



```
Node finger = head.next();
Node prev = head;

while (finger != null && !finger.value().equals(value)){
    prev = finger;
    finger = finger.next();
}

if( finger != null ){
    prev.setNext(finger.next());
}
```

Any *other* special cases we need to handle?

44

Removing a value

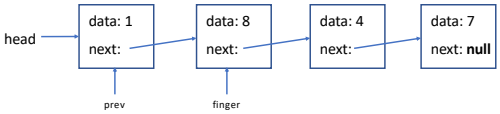



```
Node finger = head.next();
Node prev = head;

while (finger != null && !finger.value().equals(value)){
    prev = finger;
    finger = finger.next();
}

if( finger != null ){
    prev.setNext(finger.next());
}
```

Delete 1. Does it work?

45

Removing a value




```
Node finger = head.next();
Node prev = head;

while (finger != null && !finger.value().equals(value)){
    prev = finger;
    finger = finger.next();
}

if( finger != null ){
    prev.setNext(finger.next());
}
```

Delete 1. Does it work?

46

Removing a value

```
public void remove(E value){
    if( head != null ) {
        if( head.value().equals(value) ){
            head = head.next();
        }else{
            Node finger = head.next();
            Node prev = head;

            while (finger != null && !finger.value().equals(value)){
                prev = finger;
                finger = finger.next();
            }

            if( finger != null ){
                prev.setNext(finger.next());
            }
        }
    }
}
```

47

Removing a value

```
public void remove(E value){
    if( head != null ) {
        if( head.value().equals(value) ){
            head = head.next();
        }else{
            Node finger = head.next();
            Node prev = head;

            while (finger != null && !finger.value().equals(value)){
                prev = finger;
                finger = finger.next();
            }

            if( finger != null ){
                prev.setNext(finger.next());
            }
        }
    }
}
```

handle empty list

removing first element

all other elements

48

Linked lists: fast or slow?

add to the end
add to the front
contains
get
insert at an index
remove an element
set the value of an existing element
size

49

Linked lists: fast or slow?

add to the end	slow
add to the front:	fast
contains	slow
get	slow
insert at an index	slow
remove an element	slow
remove from the front	fast
set the value of an existing element	slow
size	fast

50

Linked lists: fast or slow?

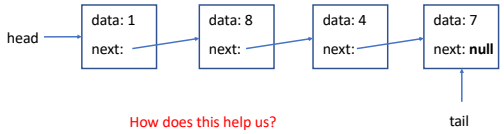
add to the end	slow
add to the front:	fast
contains	slow
get	slow
insert at an index	slow
remove an element	slow
remove from the front	fast
set the value of an existing element	slow
size	fast

Can we make any of these faster?

51

Linked list visualized

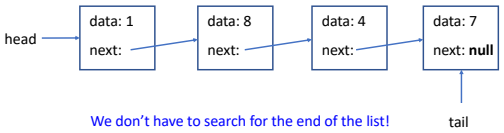
[1, 8, 4, 7]



How does this help us?

52

Linked list visualized

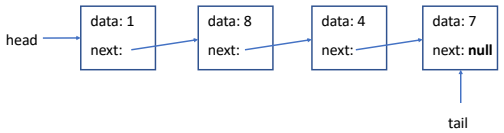


We don't have to search for the end of the list!

```
tail.setNext(new Node(value));
tail = tail.next();
```

53

Linked list visualized

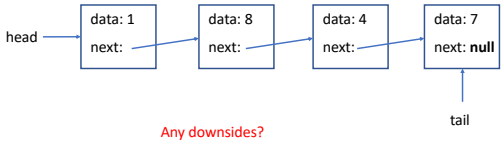



```
public void addLast(E value){
    if( head == null ){
        head = new Node(value);
        tail = head;
    }else{
        tail.setNext(new Node(value));
        tail = tail.next();
    }
}
```

54

Linked list visualized

[1, 8, 4, 7]

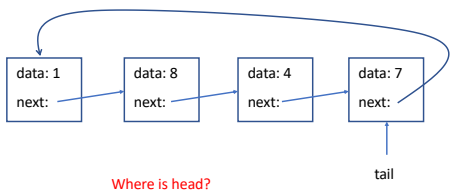


Any downsides?

55

Circularly linked list visualized

[1, 8, 4, 7]

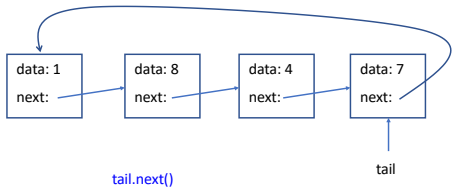


Where is head?

56

Circularly linked list visualized

[1, 8, 4, 7]



57

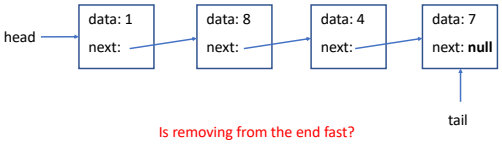
Linked lists: fast or slow?

add to the end	**fast**
add to the front:	fast
contains	slow
get	slow
insert at an index	slow
remove an element	slow
remove from the front	fast
set the value of an existing element	slow
size	fast

58

Linked list visualized

[1, 8, 4, 7]

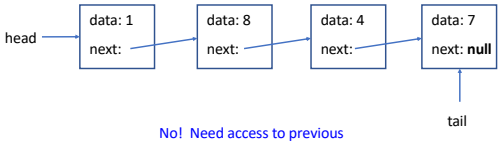


Is removing from the end fast?

59

Linked list visualized

[1, 8, 4, 7]

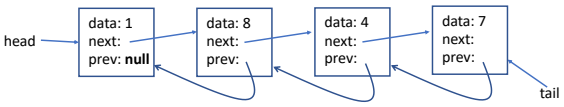


No! Need access to previous

60

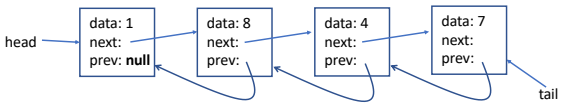
Doubly linked list visualized

Keep track of both the next *and* previous node in the list



61

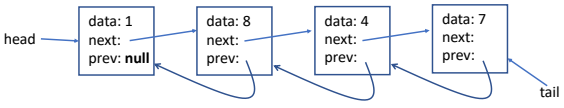
Doubly linked list visualized



Is removing from the end fast?

62

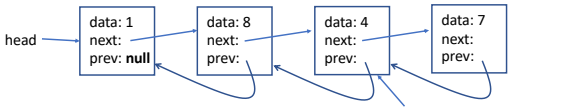
Doubly linked list visualized




```
tail = tail.prev();
tail.setNext(null);
```

63

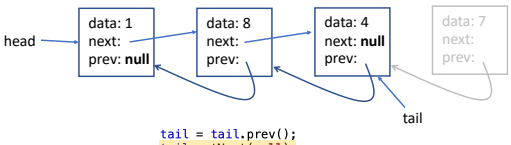
Doubly linked list visualized




```
tail = tail.prev();
tail.setNext(null);
```

64

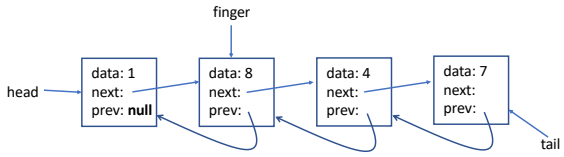
Doubly linked list visualized




```
tail = tail.prev();
tail.setNext(null);
```

65

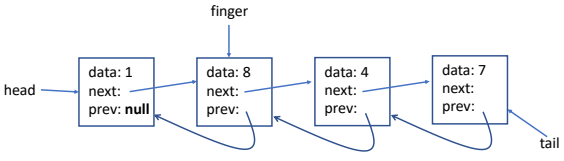
Doubly linked list visualized



Could I remove a node if I had a reference to it?

66

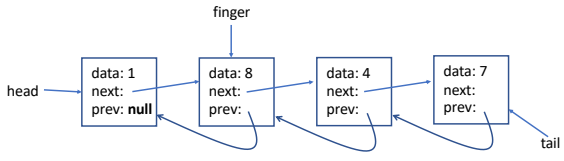
Doubly linked list visualized



Could I remove a node if I had a reference to it?

67

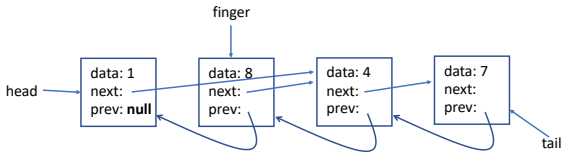
Doubly linked list visualized




```
finger.prev().setNext(finger.next());
finger.next().setPrev(finger.prev());
```

68

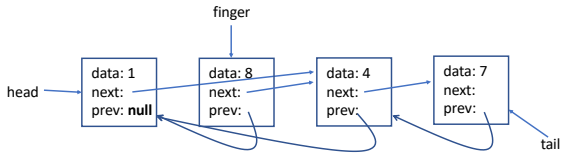
Doubly linked list visualized




```
finger.prev().setNext(finger.next());
finger.next().setPrev(finger.prev());
```

69

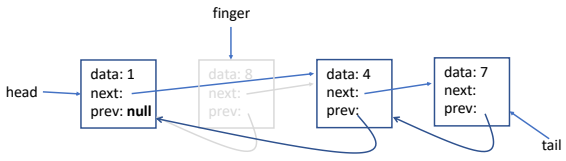
Doubly linked list visualized




```
finger.prev().setNext(finger.next());
finger.next().setPrev(finger.prev());
```

70

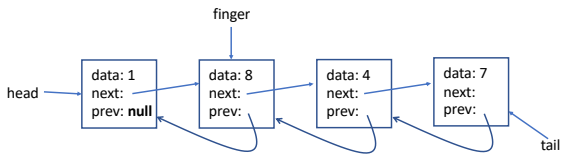
Doubly linked list visualized




```
finger.prev().setNext(finger.next());
finger.next().setPrev(finger.prev());
```

71

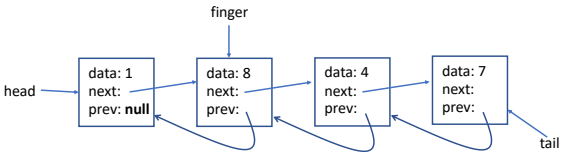
Doubly linked list visualized



How does this help us? Still have to find the node?

72

Doubly linked list visualized



If we're iterating over the data already
Other scenarios (see upcoming assignment ☺)

73

List performance

	ArrayList	Singly linked list	Singly linked list (tail)	Doubly linked list
add to end	fast (amortized)	slow	fast	fast
add to front	slow	fast	fast	fast
insert at index	slow	slow	slow	slow
contains	slow	slow	slow	slow
get	fast	slow	slow	slow
set	fast	slow	slow	slow
remove	slow	slow	slow	slow
remove from end	fast	slow	slow	fast
remove from front	slow	fast	fast	fast
size	fast	fast	fast	fast

74