Collections with Iteration Order: Lists

Richard Warburton
JAVA CHAMPION, AUTHOR AND PROGRAMMER

@richardwarburto   www.monotonic.co.uk

Outline

Key Features

Shipment Example

Implementations

Key Features

Lists are collections with iteration order

```java
void add(int index, E e);

E get(int index);

E remove(int index);

E set(int index, E element);

boolean addAll(int index, Collection c);
```

Each element has an index

An index is an int representing its position in the List.

We can modify Lists using indices

```java
int indexOf(Object o);

int lastIndexOf(Object o);
```

You can also lookup indices by value

Sublists are views over ranges of lists.

Modifying the view modifies the List.

```
List subList(int fromIndex, int toIndex);
```

Shipments Example

Light Products

Heavy Products

Shipments Example (2)

Shipments Example (3)

Implementations

Interfaces define behavior.

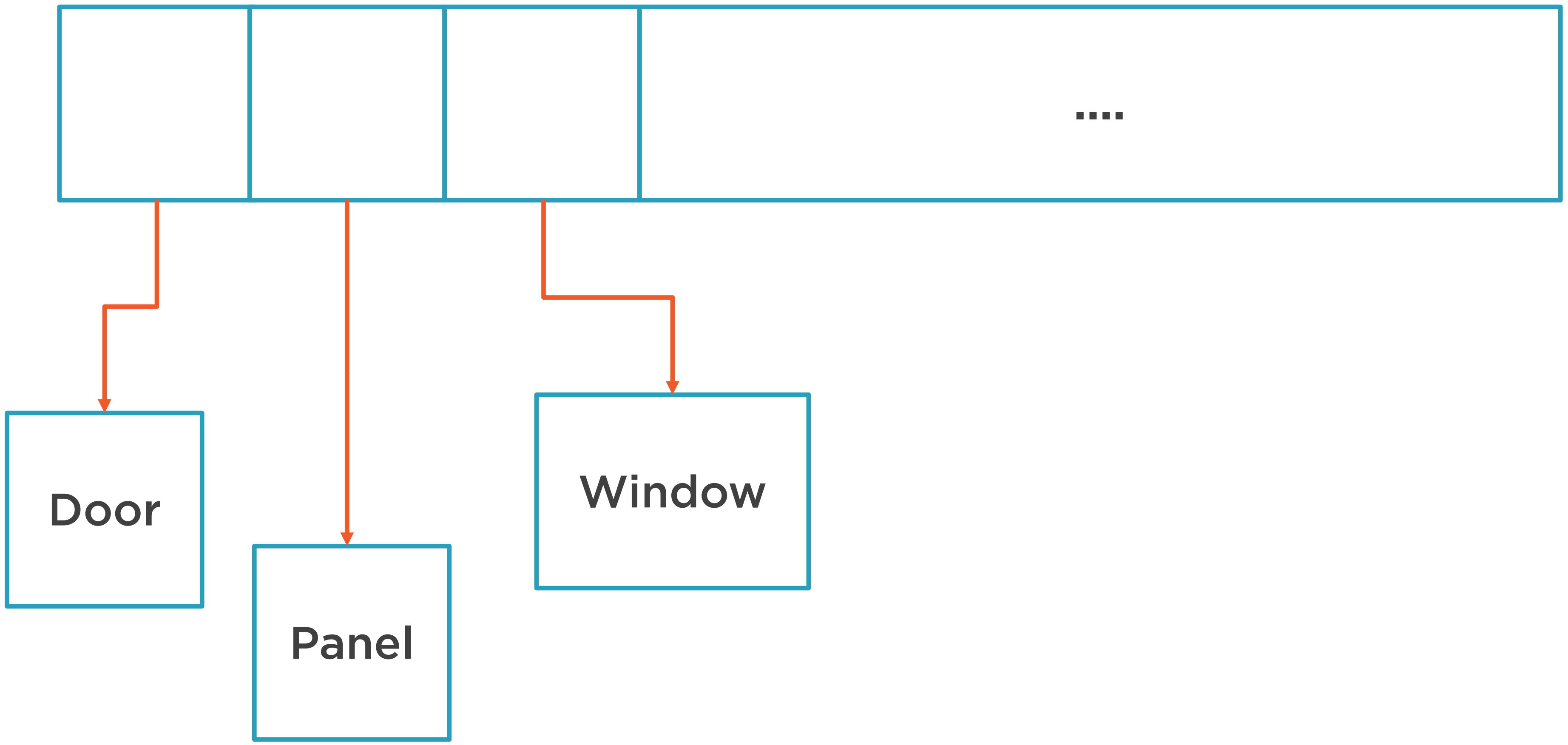
Implementations determine performance.

List Implementations

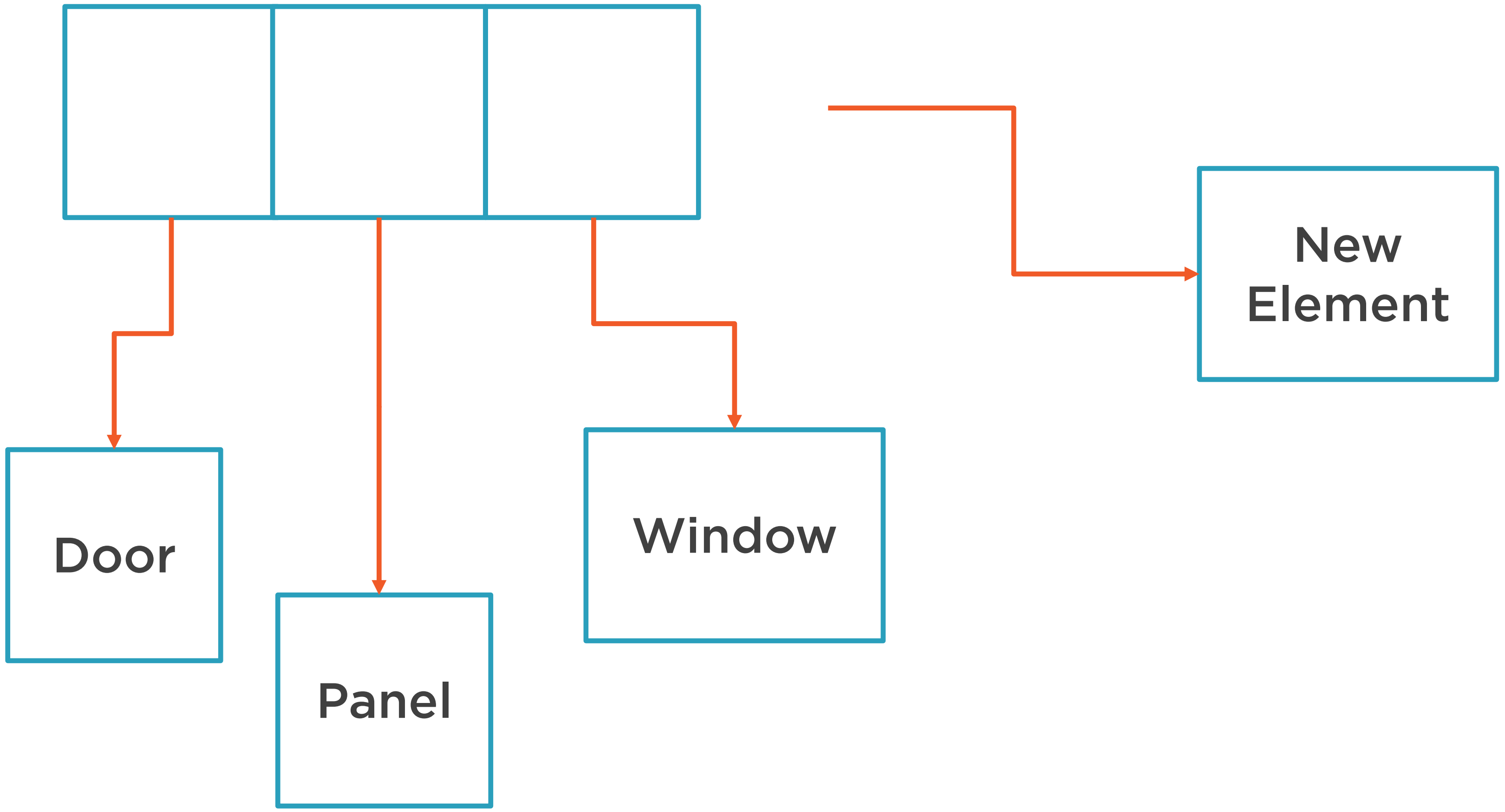
ArrayList

LinkedList

ArrayList



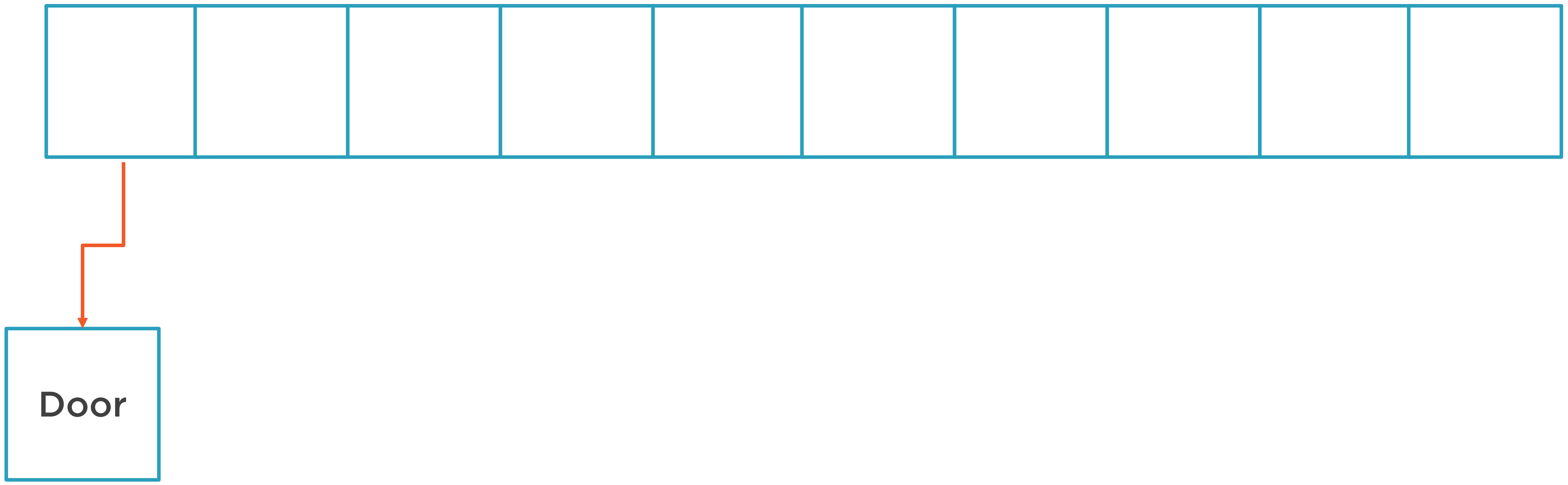
ArrayList



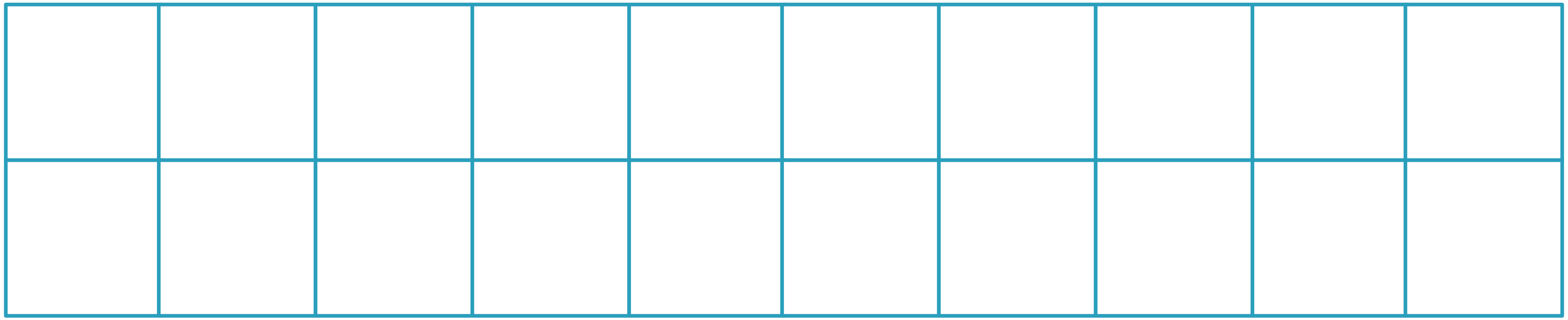
Empty ArrayList

```
null
```

Initialised ArrayList



Growing ArrayList



Doubling Strategy

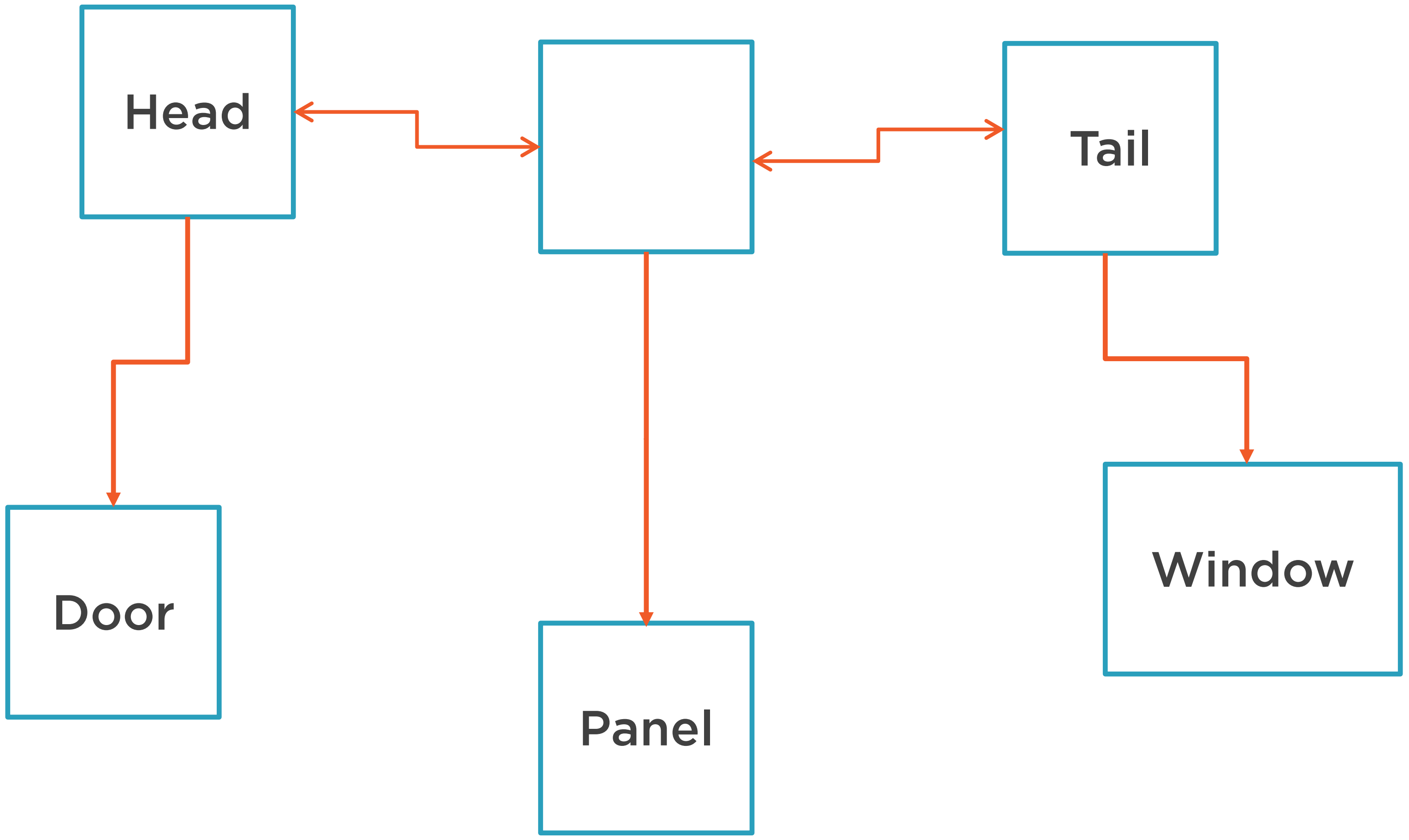
ArrayList

Good General Purpose Implementation

Use as Default

CPU Cache Sympathetic

LinkedList



LinkedList

Worse performance in most cases

Use when adding elements at start

Or when adding / remove a lot

Performance Comparison

	get	add	contains	next	remove
ArrayList	O(1)	O(N), Ω(1)	O(N)	O(1)	O(N)
LinkedList	O(N)	O(1)	O(N)	O(1)	O(N)

Conclusions

Summary

Demonstrated key List features

Looked at different performance tradeoffs

Lists are really commonly used