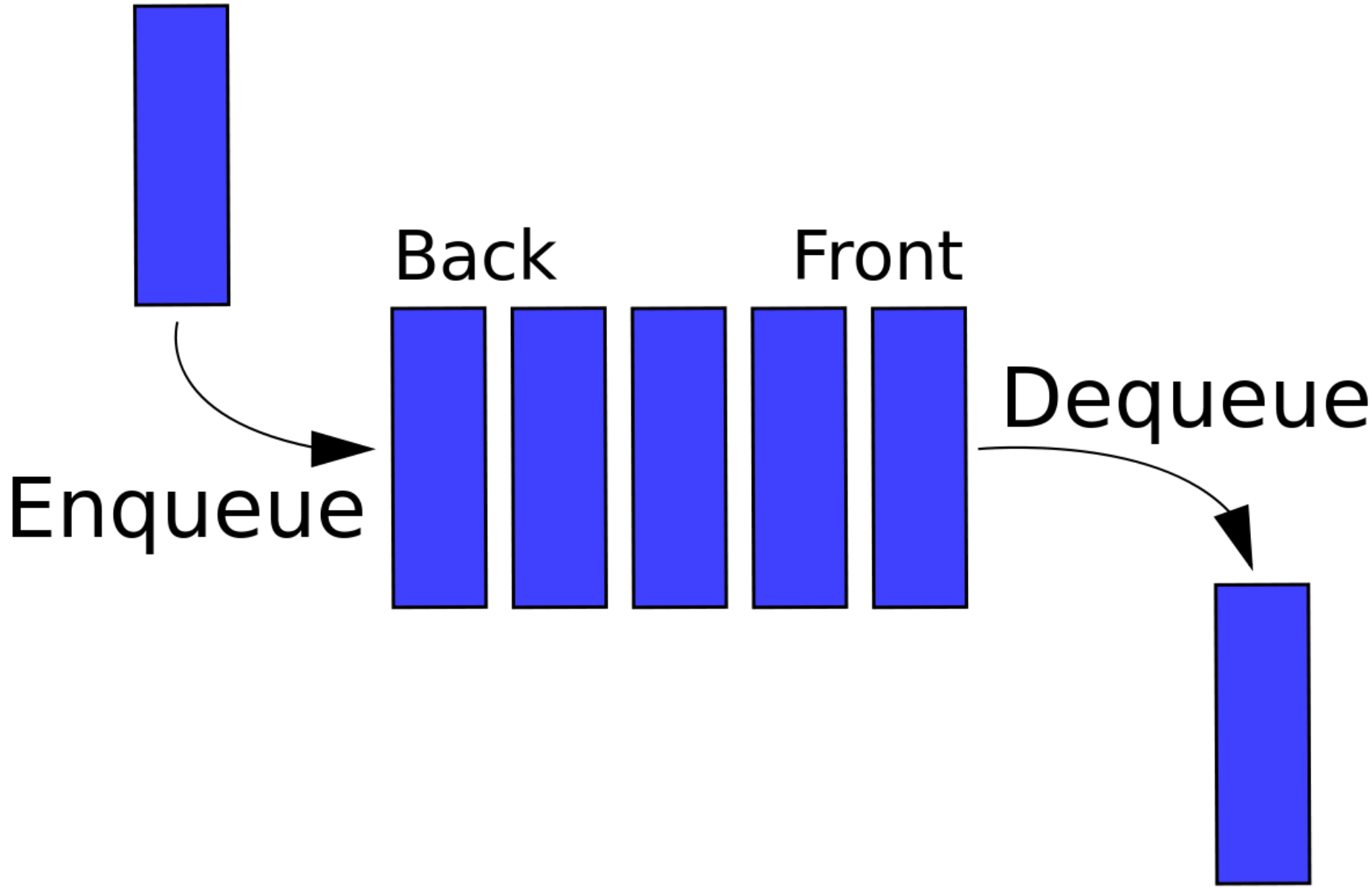
Queue

An ADT for a Line

A queue is like a line.

Back

Front


Back
Front
Enqueue
Dequeue

Queue - FIFO

- An standard object type (an ADT)
- First in, first out (FIFO)
- Add to the back, remove from the front
- Used to simulate lines: Printer Queue, Banking operations, Order Placement

Object = data + methods

Public

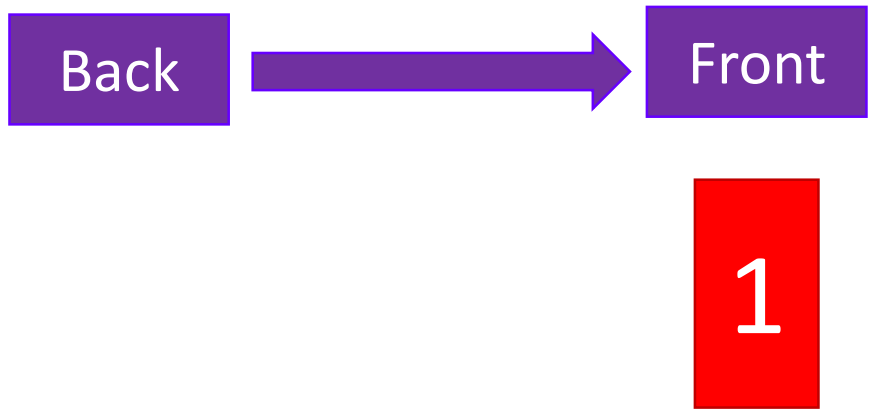
Private

Queue = Array to hold stuff
Int to track front
Int to track size + Peek
Enqueue
Dequeue
Size
isFull
isEmpty
toString

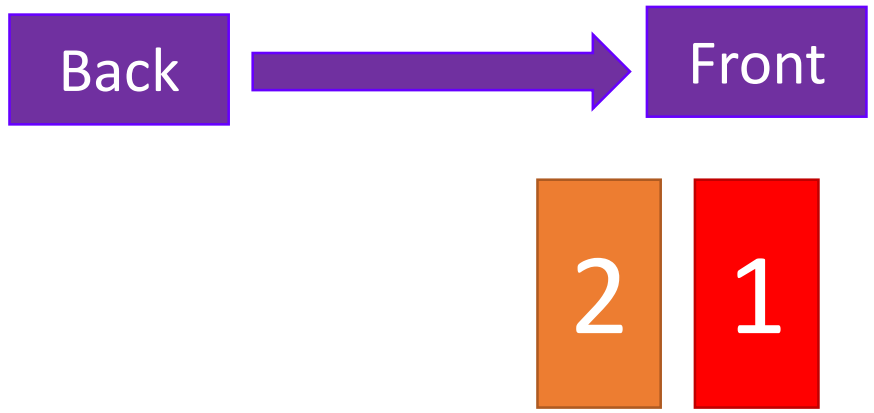
```
1. Queue q = new Queue ();
2. q.enqueue (1);
3. q.enqueue (2);
4. System.out.println (q.dequeue ());
5. q.enqueue (3);
6. q.enqueue (4);
7. System.out.println (q.size ());
8. System.out.println (q.dequeue ());
9. q.enqueue (5);
10. System.out.println (q.dequeue ());
```

```
1. Queue q = new Queue ();
2. q.enqueue (1);
3. q.enqueue (2);
4. System.out.println (q.dequeue ());
5. q.enqueue (3);
6. q.enqueue (4);
7. System.out.println (q.size ());
8. System.out.println (q.dequeue ());
9. q.enqueue (5);
10. System.out.println (q.dequeue ());
```

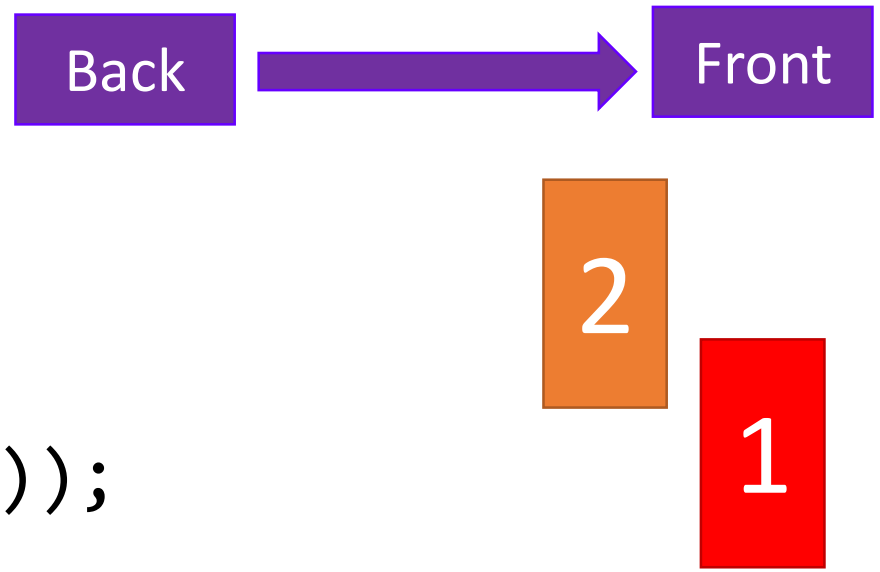
```
1. Queue q = new Queue ();
2. q.enqueue (1);
3. q.enqueue (2);
4. System.out.println (q.dequeue ());
5. q.enqueue (3);
6. q.enqueue (4);
7. System.out.println (q.size ());
8. System.out.println (q.dequeue ());
9. q.enqueue (5);
10. System.out.println (q.dequeue ());
```



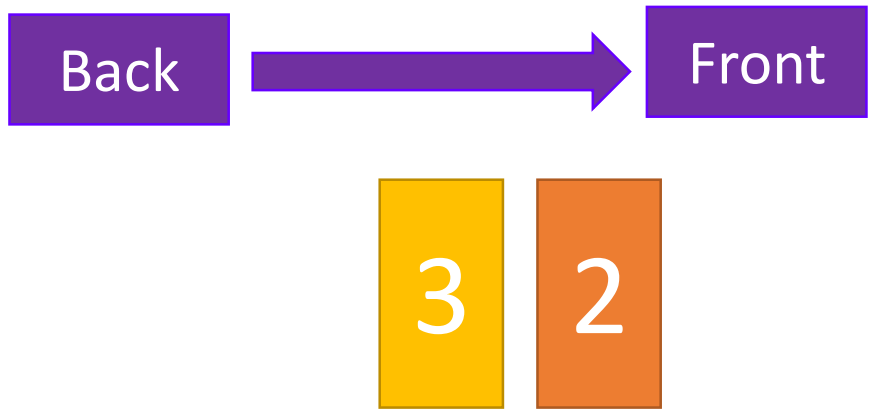
```
1. Queue q = new Queue ();
2. q.enqueue (1);
3. q.enqueue (2);
4. System.out.println (q.dequeue ());
5. q.enqueue (3);
6. q.enqueue (4);
7. System.out.println (q.size ());
8. System.out.println (q.dequeue ());
9. q.enqueue (5);
10. System.out.println (q.dequeue ());
```



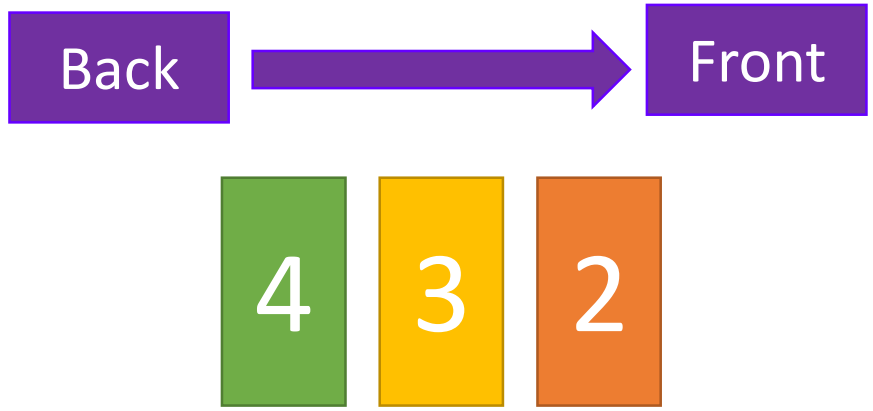
```
1. Queue q = new Queue ();
2. q.enqueue (1);
3. q.enqueue (2);
1 4. System.out.println (q.dequeue ());
5. q.enqueue (3);
6. q.enqueue (4);
7. System.out.println (q.size ());
8. System.out.println (q.dequeue ());
9. q.enqueue (5);
10. System.out.println (q.dequeue ());
```

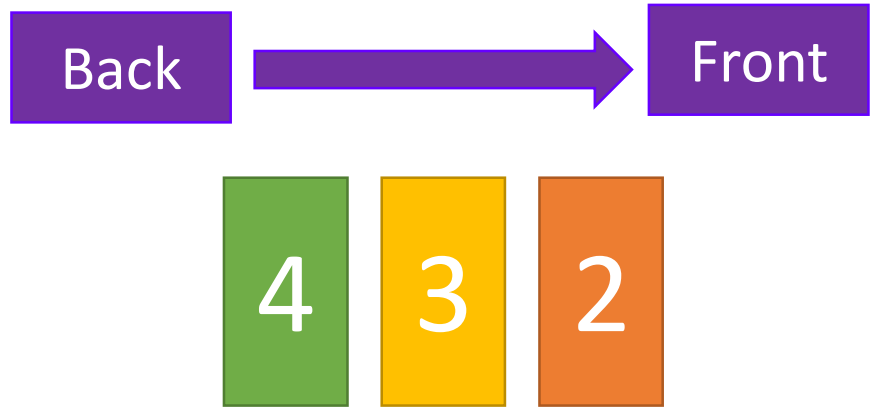


```
1. Queue q = new Queue ();
2. q.enqueue (1);
3. q.enqueue (2);
1 4. System.out.println (q.dequeue ());
5. q.enqueue (3);
6. q.enqueue (4);
7. System.out.println (q.size ());
8. System.out.println (q.dequeue ());
9. q.enqueue (5);
10. System.out.println (q.dequeue ());
```



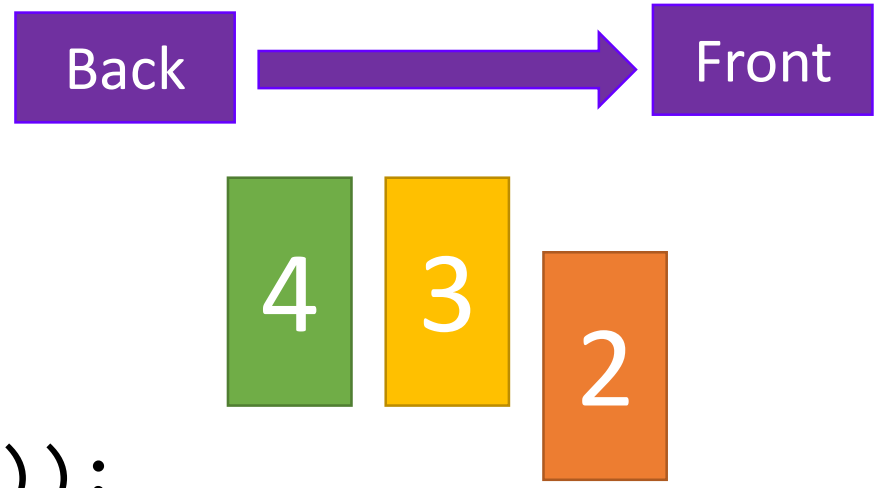
```
1. Queue q = new Queue ();
2. q.enqueue (1);
3. q.enqueue (2);
1 4. System.out.println (q.dequeue ());
5. q.enqueue (3);
6. q.enqueue (4);
7. System.out.println (q.size ());
8. System.out.println (q.dequeue ());
9. q.enqueue (5);
10. System.out.println (q.dequeue ());
```

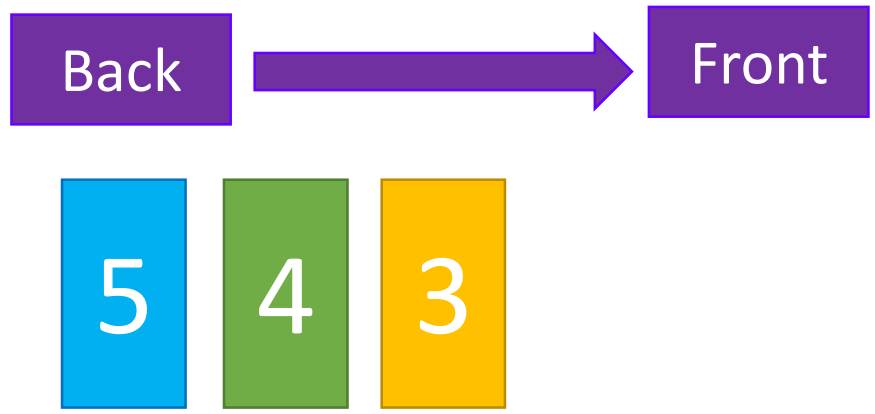




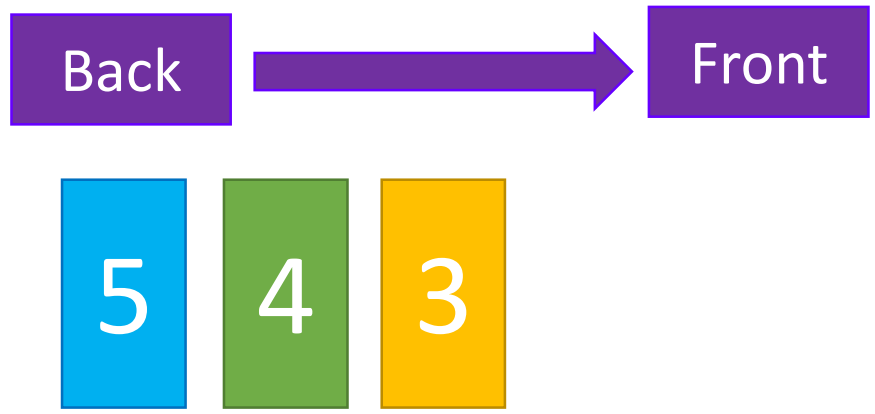

```
1. Queue q = new Queue ();
2. q.enqueue (1);
3. q.enqueue (2);
1  4. System.out.println (q.dequeue ());
5. q.enqueue (3);
6. q.enqueue (4);
3  7. System.out.println (q.size ());
8. System.out.println (q.dequeue ());
9. q.enqueue (5);
10. System.out.println (q.dequeue ());
```

```
1. Queue q = new Queue ();
2. q.enqueue (1);
3. q.enqueue (2);
1  4. System.out.println (q.dequeue ());
5. q.enqueue (3);
6. q.enqueue (4);
3  7. System.out.println (q.size ());
2  8. System.out.println (q.dequeue ());
9. q.enqueue (5);
10. System.out.println (q.dequeue ());
```






```
1. Queue q = new Queue ();
2. q.enqueue (1);
3. q.enqueue (2);
1  4. System.out.println (q.dequeue ());
5. q.enqueue (3);
6. q.enqueue (4);
3  7. System.out.println (q.size ());
2  8. System.out.println (q.dequeue ());
9. q.enqueue (5);
10. System.out.println (q.dequeue ());
```




```
    1. Queue q = new Queue ();
    2. q.enqueue (1);
    3. q.enqueue (2);
1   4. System.out.println (q.dequeue ());
    5. q.enqueue (3);
    6. q.enqueue (4);
3   7. System.out.println (q.size ());
2   8. System.out.println (q.dequeue ());
    9. q.enqueue (5);
3   10. System.out.println (q.dequeue ());
```

Declaration

Setting aside RAM

“JButton b;”

Constructor

Sets up dynamic memory

to be ready for use

“new”

Facilitator

Anything that isn’t construction, accessor, mutator

Complex functions

“equals”

“compareTo”

Mutator

Changing

dynamic memory

“set” - setForeground

Accessor

Checking dynamic memory

“get” – getText

“toString”

“is” – isVisible

Classifying the Method Types in the Queue Class

```
Queue ()
void enqueue (String value)
String dequeue ()
String peek ()
int size ()
boolean isEmpty ()
boolean isFull ()
void clear()
```

Classifying the Method Types in the Queue Class

constructor  `Queue ()`

mutator  `void enqueue (String value)`

mutator  `String dequeue ()`

accessor  `String peek ()`

accessor  `int size ()`

accessor  `boolean isEmpty ()`

accessor  `boolean isFull ()`

mutator  `void clear()`

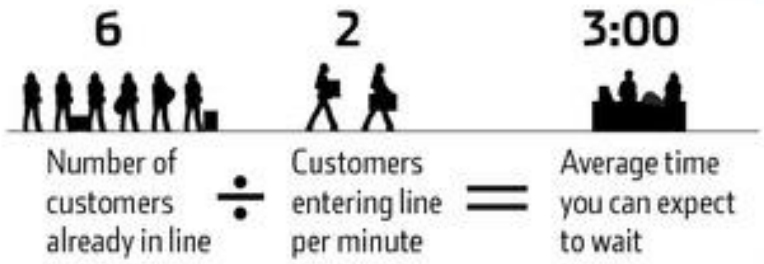
Queue

- A common ADT, An object
- Has a FIFO structure (First in, first out)
- Functions: size, isFull, isEmpty, enqueue (add), dequeue (remove), peek (look at the top)
- All operations are O(1), but you are limited to queue functions. The speed comes from the fact that you restricted the functionality a lot.

The Science of Lines

What's really happening at checkout

A shopper can use this **formula,** by John D.C. Little, to determine expected wait time: Average wait time = average number of people in line divided by their arrival rate.



Clock watching

Once a wait lasts longer than three minutes, the perceived wait time multiplies with each passing minute. Shoppers who actually waited five minutes told surveyors they felt they had waited twice as long.

Impulse buying

Mall retailers are copying grocery stores with items like tiny stuffed animals and gift cards next to lines to distract from the wait.

Line jockeying

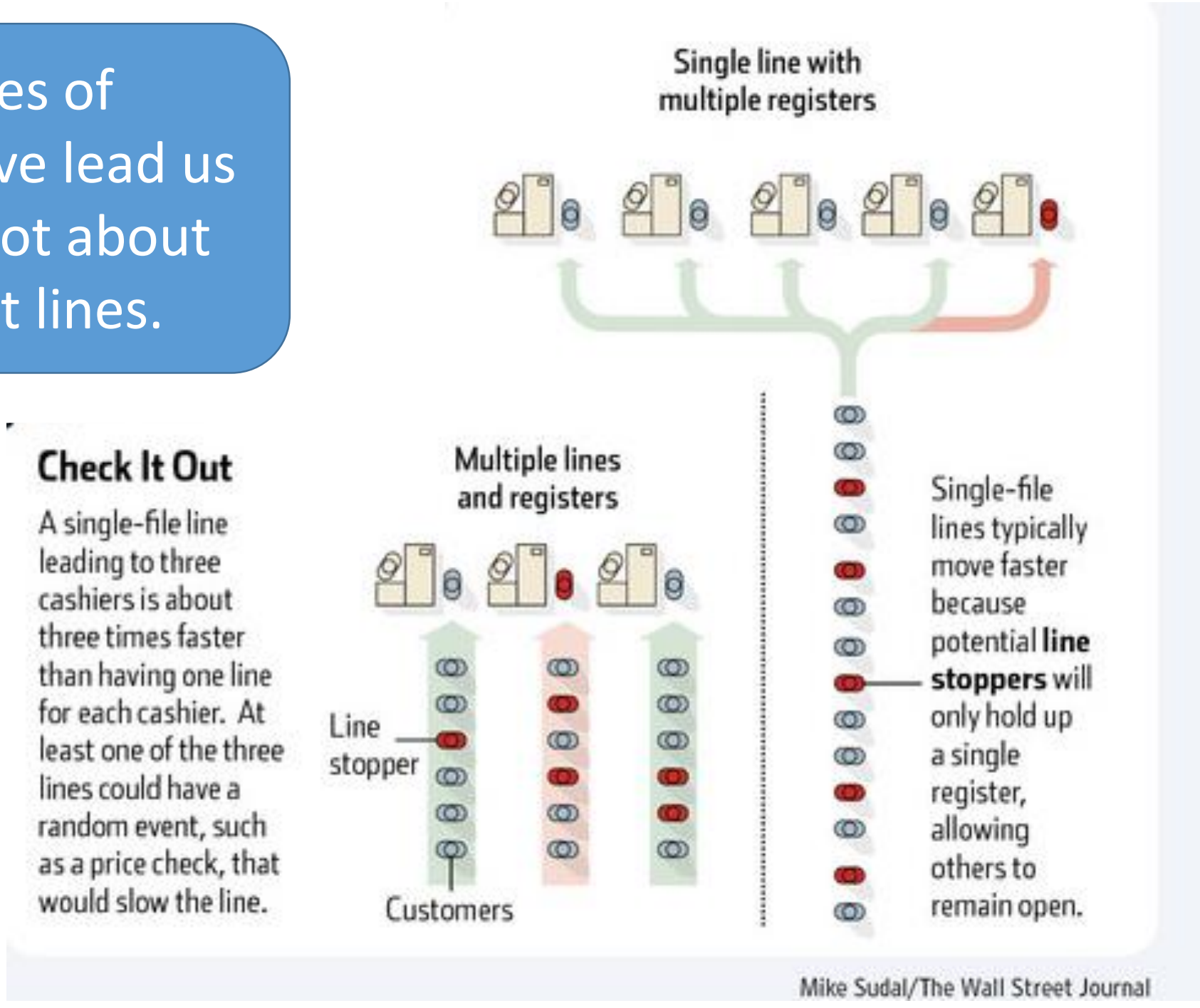
Short lines are usually short for a reason. Other shoppers may have concluded that a short line has an extremely slow or chatty cashier.

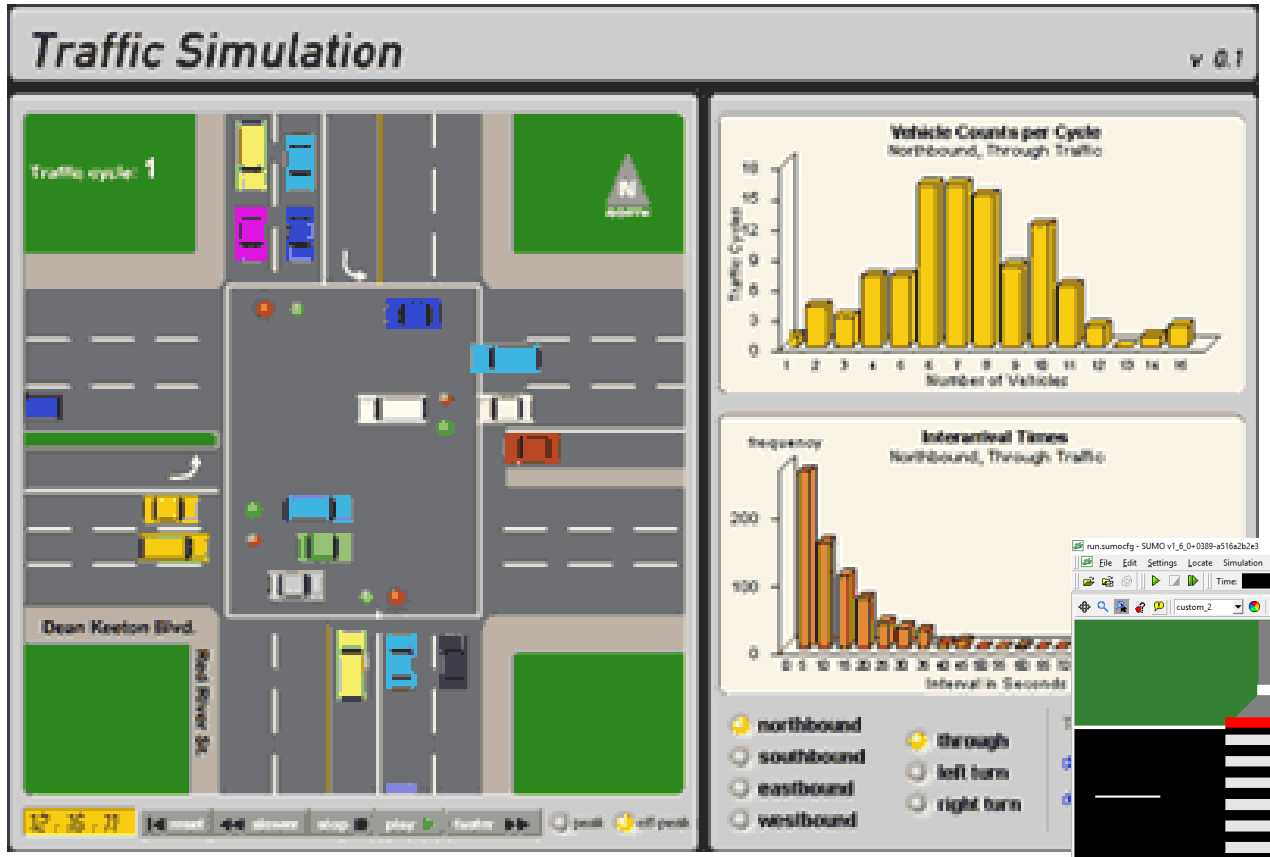
More staff

Some stores employ 'runners' at the holidays to assist cashiers. Old Navy sends out 'line expeditors' and 'super helpers' during peak times.

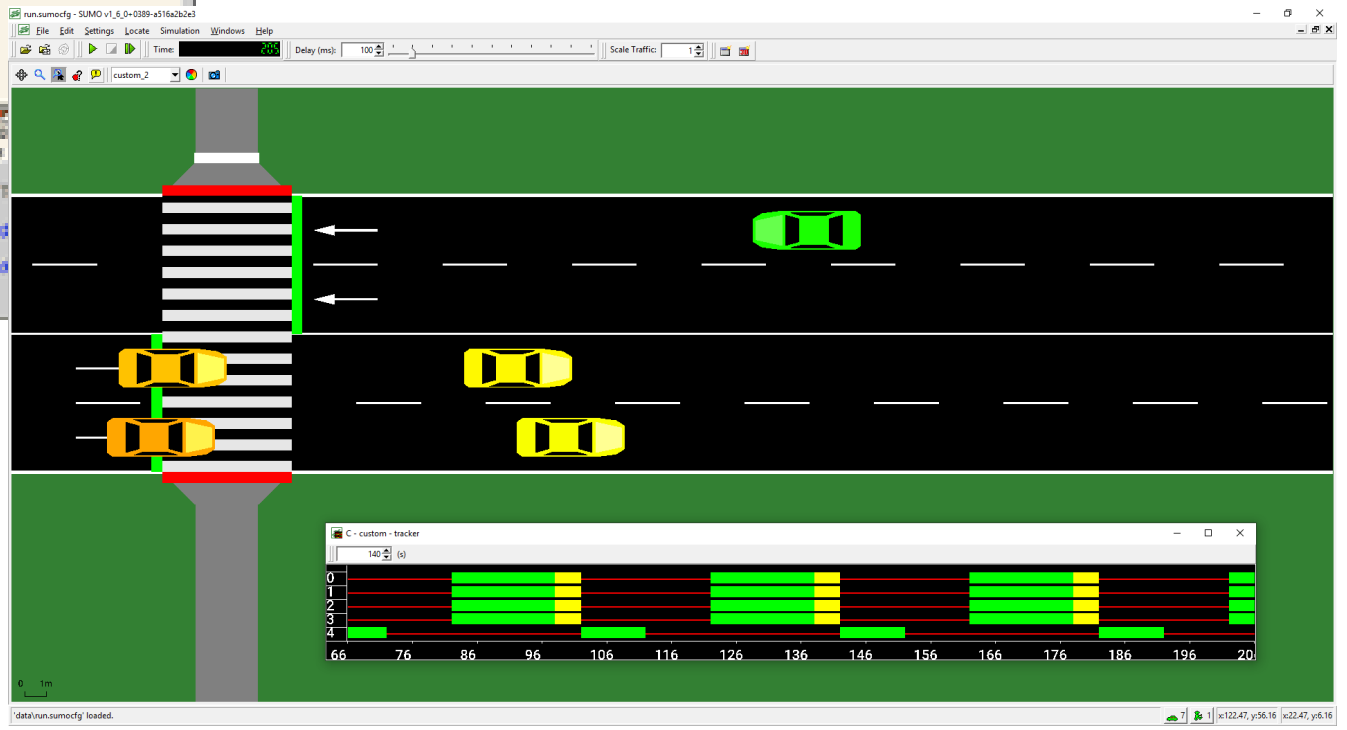
A common introductory programming problem is to use a queue to simulate lines.

These types of simulations have lead us to discover a lot about supermarket lines.





Queues are also used in traffic simulations



https://www.youtube.com/watch?v=oAHbLRjF0vo

An airline boarding queue simulation. Finds a lot of efficiencies.