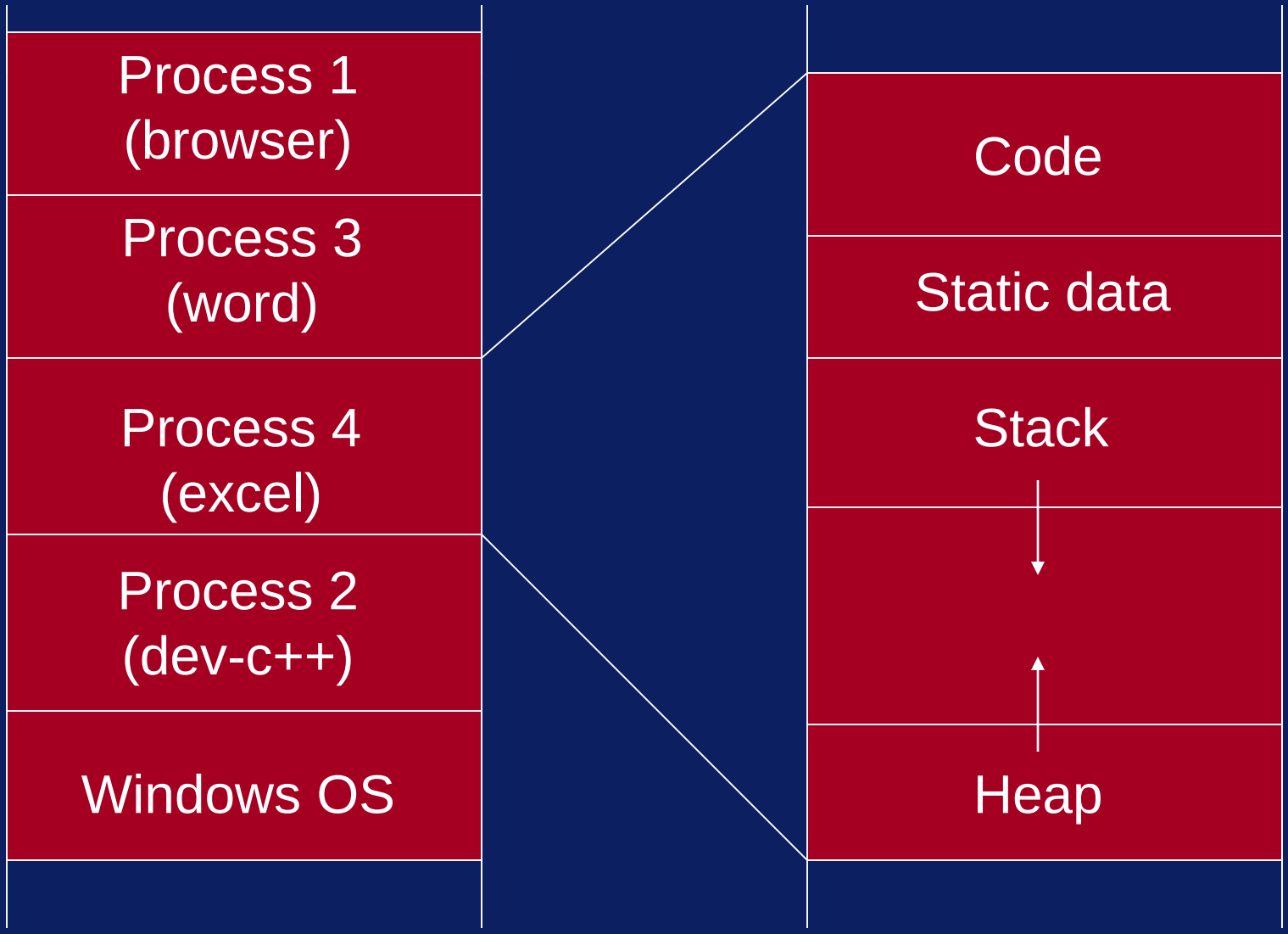
Lecture No.09

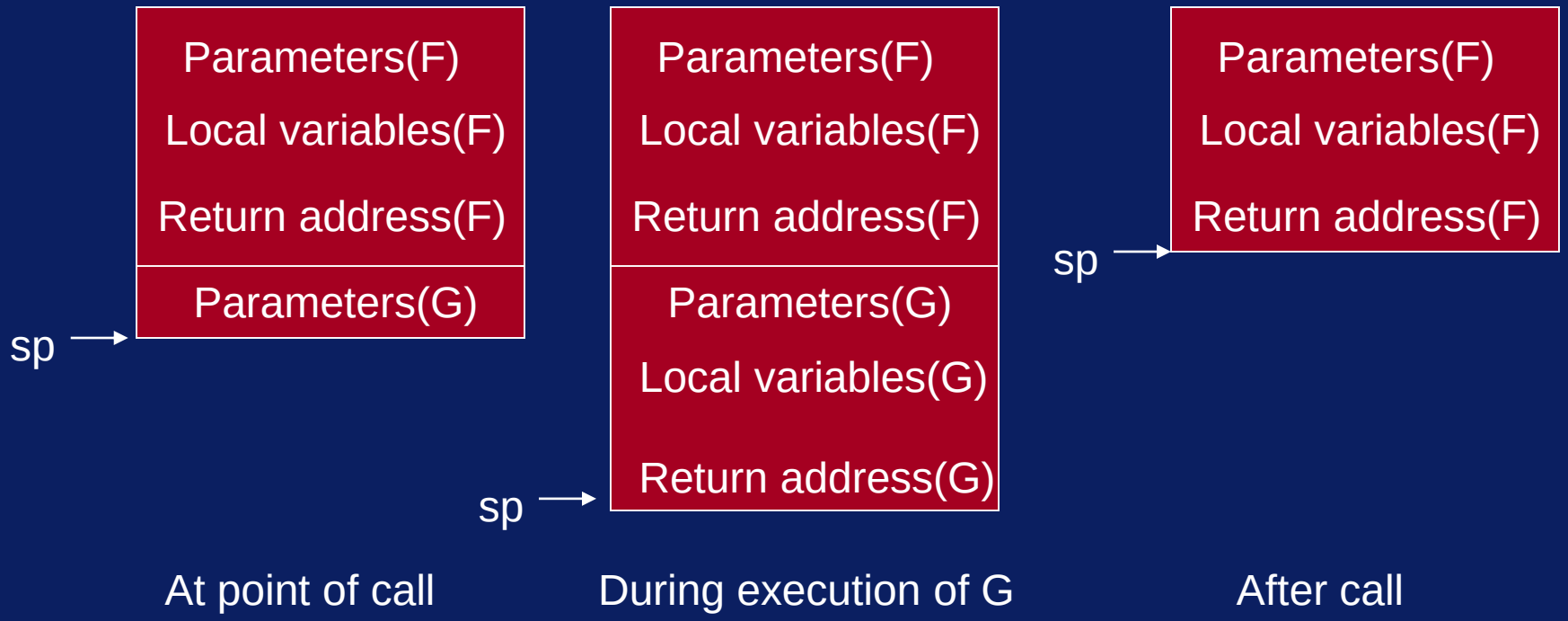
Data Structures

Memory Organization



Stack Layout during a call

- Here is stack layout when function F calls function G:



Queues

- A stack is LIFO (Last-In First Out) structure.
- In contrast, a *queue* is a FIFO (First-In First-Out) structure.
- A queue is a linear structure for which items can be only inserted at one end and removed at another end.

Queue Operations

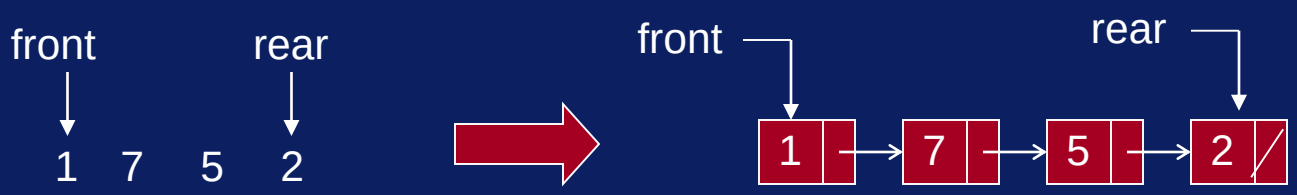
Enqueue(X) –	place X at the *rear* of the queue.
Dequeue() --	remove the *front* element and return it.
Front() --	return front element without removing it.
IsEmpty() --	return TRUE if queue is empty, FALSE otherwise

Implementing Queue

- Using linked List: *Recall*
- Insert works in constant time for either end of a linked list.
- Remove works in constant time only.
- Seems best that head of the linked list be the front of the queue so that all removes will be from the front.
- Inserts will be at the end of the list.

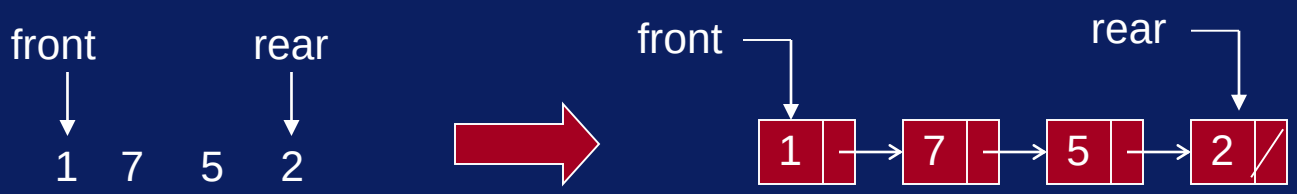
Implementing Queue

- Using linked List:

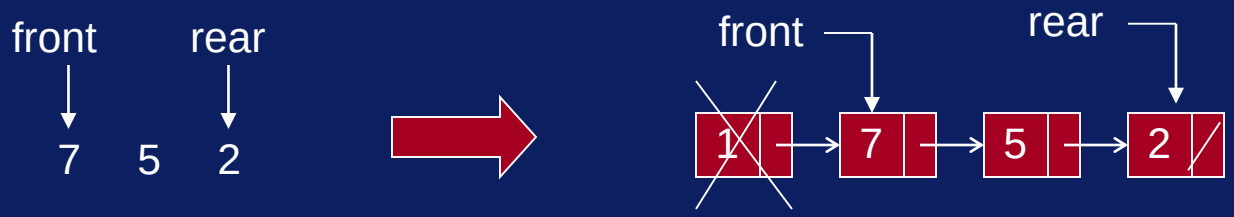


Implementing Queue

- Using linked List:

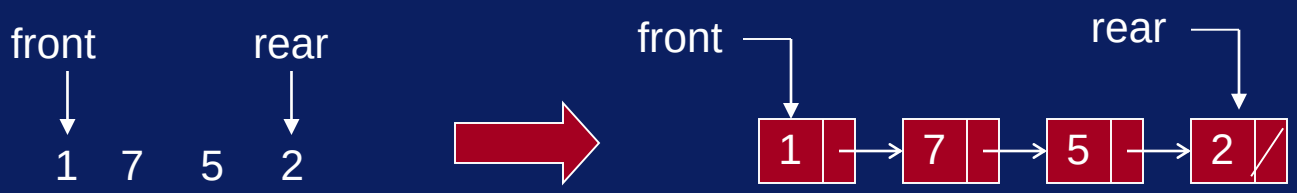


dequeue()



Implementing Queue

- Using linked List:



enqueue(9)

Implementing Queue

```
int dequeue()
{
    int x = front->get();
    Node* p = front;
    front = front->getNext();
    delete p;
    return x;
}
void enqueue(int x)
{
    Node* newNode = new Node();
    newNode->set(x);
    newNode->setNext(NULL);
    rear->setNext(newNode);
    rear = newNode;
}
```

Implementing Queue

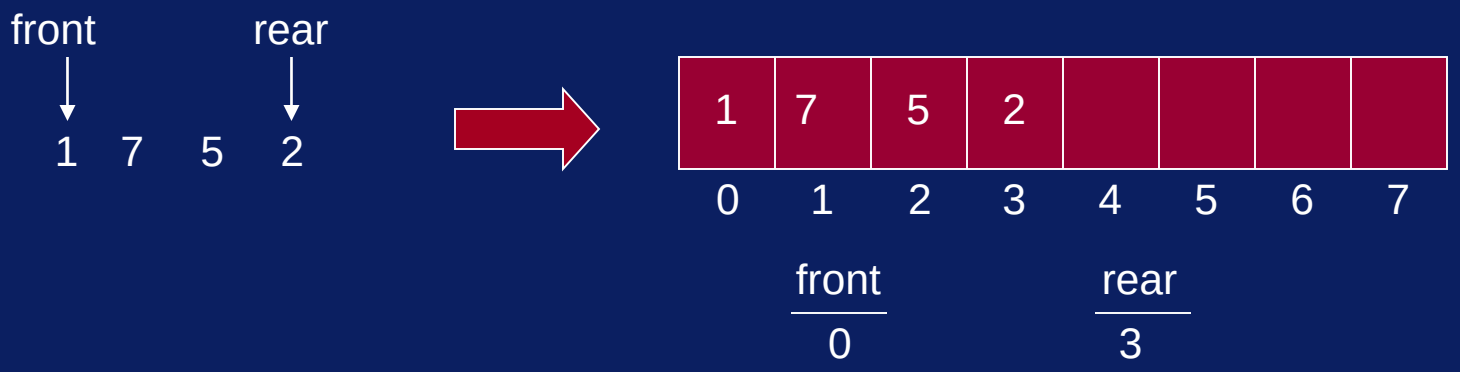
```
int front()
{
    return front->get();
}


int isEmpty()
{
    return ( front == NULL );
}
```

Queue using Array

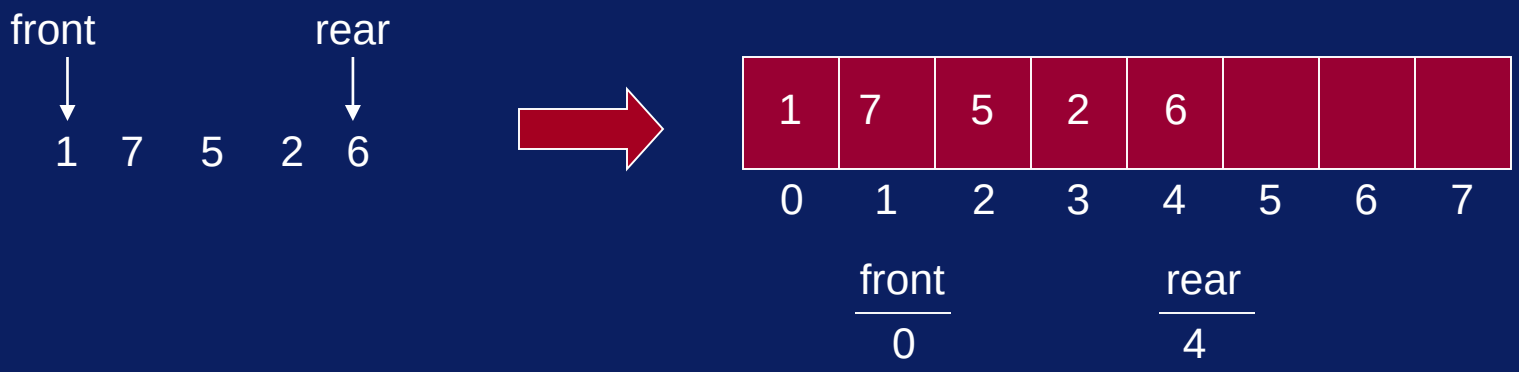
- If we use an array to hold queue elements, both insertions and removal at the front (start) of the array are expensive.
- This is because we may have to shift up to “n” elements.
- For the stack, we needed only one end; for queue we need both.
- To get around this, we will not shift upon removal of an element.

Queue using Array



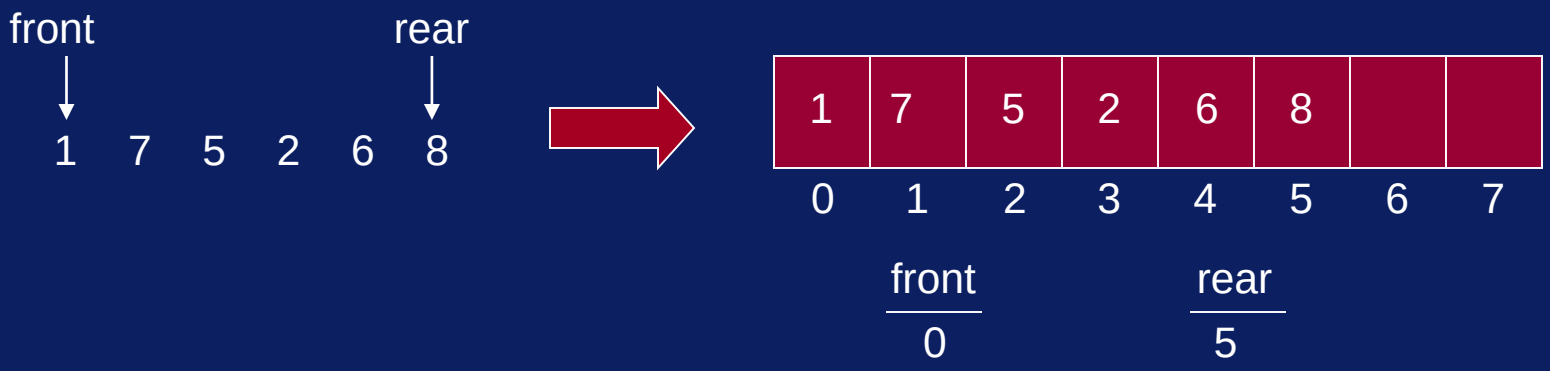
Queue using Array

enqueue(6)



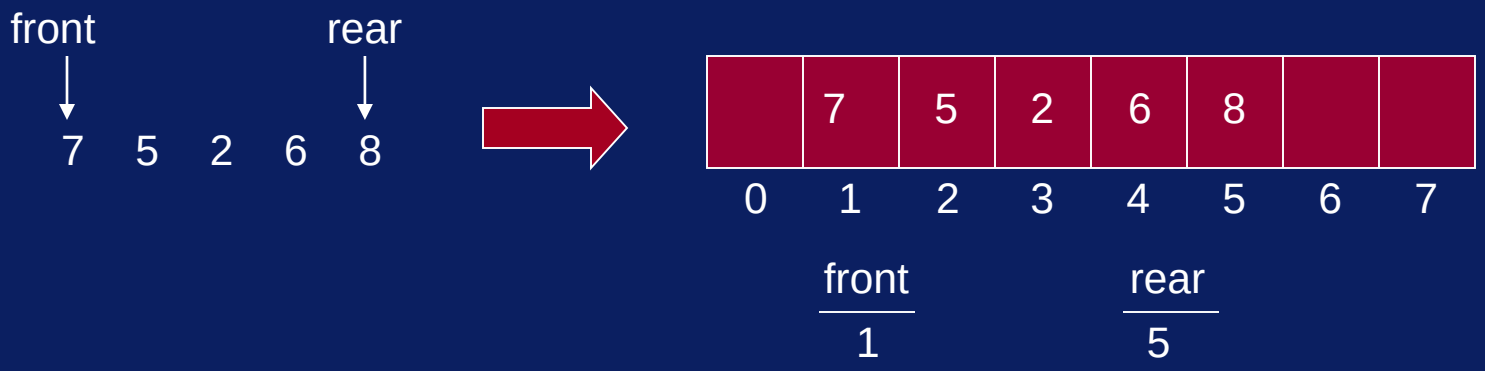
Queue using Array

enqueue(8)



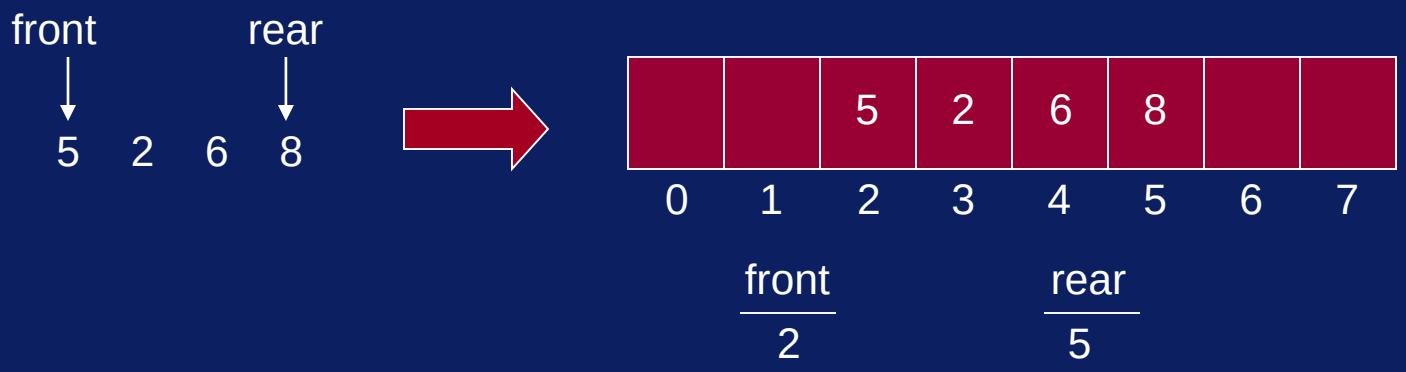
Queue using Array

dequeue()



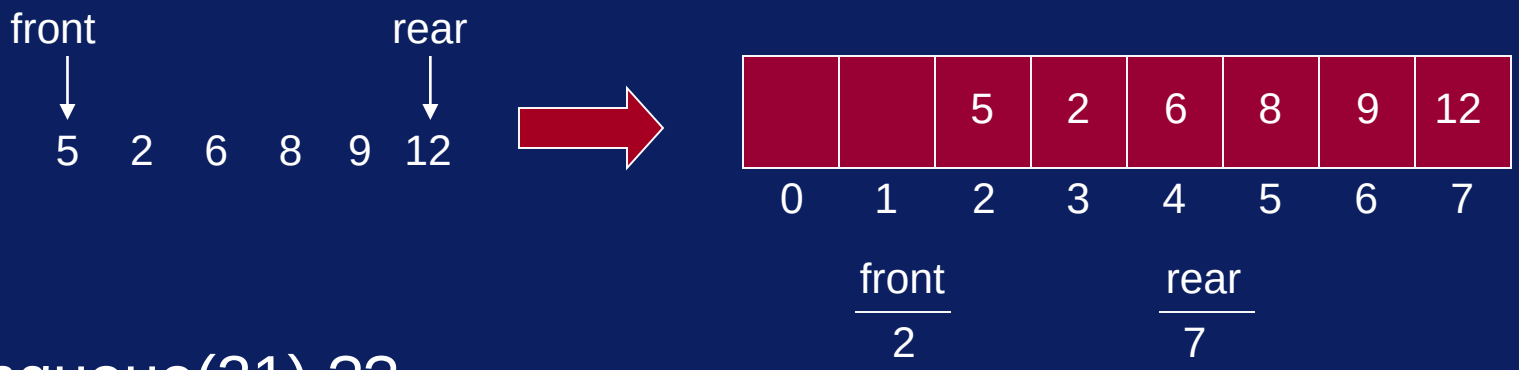
Queue using Array

dequeue()



Queue using Array

enqueue(9)
enqueue(12)



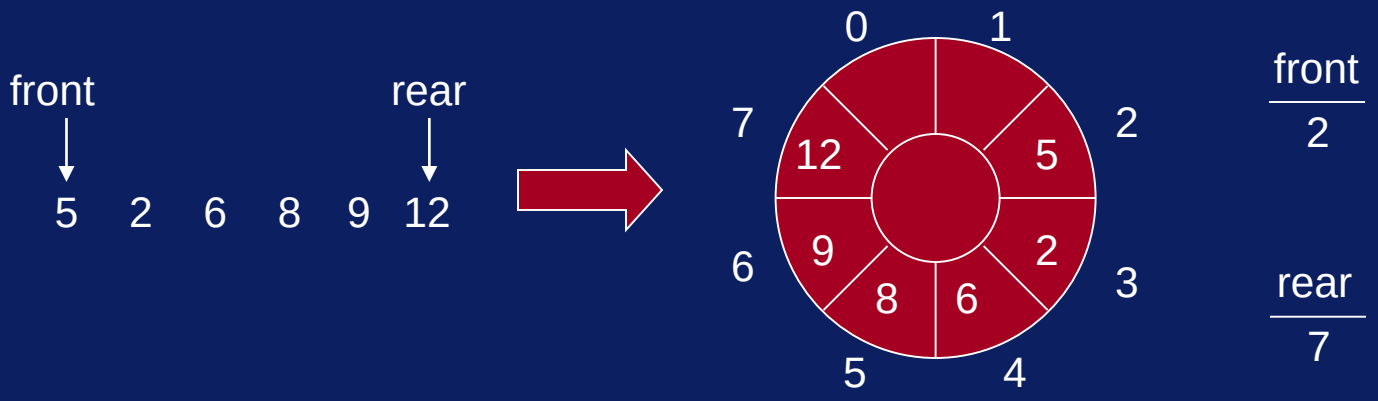
enqueue(21) ??

Queue using Array

- We have inserts and removal running in constant time but we created a new problem.
- Cannot insert new elements even though there are two places available at the start of the array.
- Solution: allow the queue to “wrap around”.

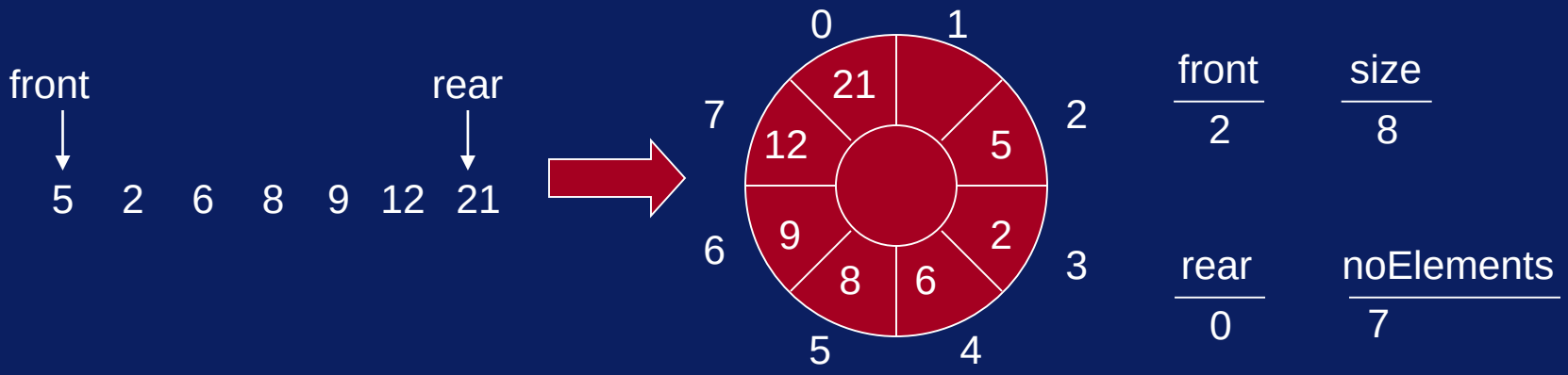
Queue using Array

- Basic idea is to picture the array as a *circular array*.



Queue using Array

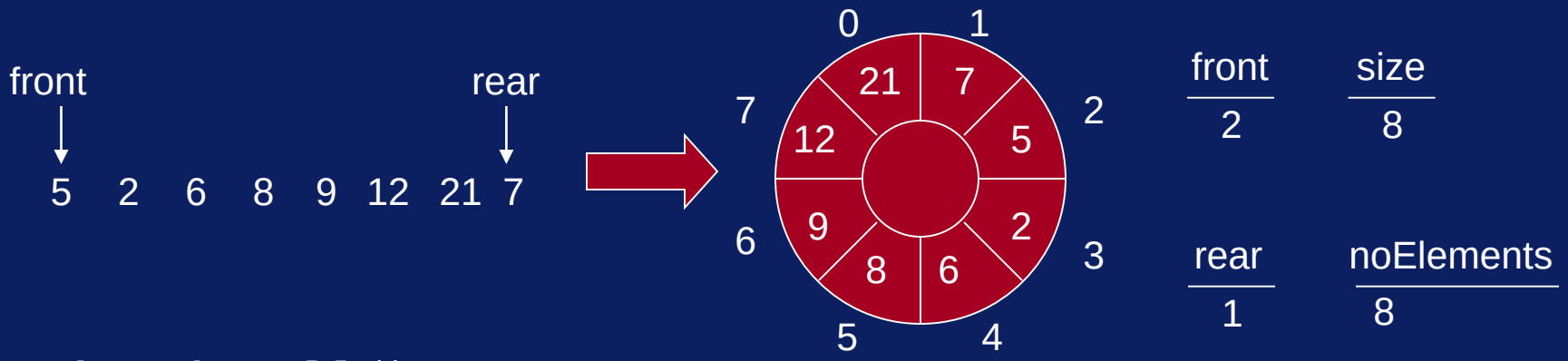
enqueue(21)




```
void enqueue(int x)
{
     rear = (rear+1)%size;
     array[rear] = x;
     noElements = noElements+1;
}
```

Queue using Array

enqueue(7)

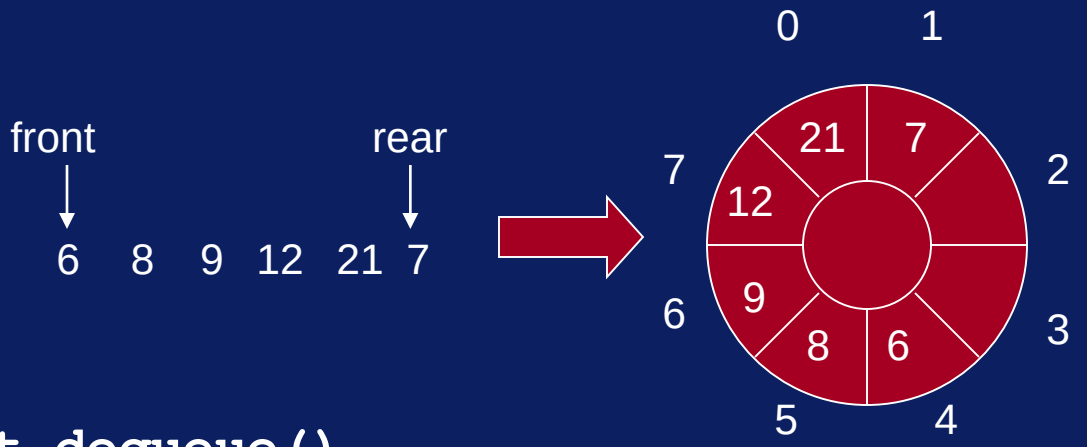



```
int isFull()
{
    return noElements == size;
}

int isEmpty()
{
    return noElements == 0;
}
```

Queue using Array

dequeue()



front
4

size
8

rear
1

noElements
6

```
int dequeue()
{
     int x = array[front];
     front = (front+1)%size;
     noElements = noElements-1;
     return x;
}
```

Use of Queues

- Out of the numerous uses of the queues, one of the most useful is *simulation*.
- A simulation program attempts to model a real-world phenomenon.
- Many popular video games are simulations, e.g., SimCity, FlightSimulator
- Each object and action in the simulation has a counterpart in real world.

Uses of Queues

- If the simulation is accurate, the result of the program should mirror the results of the real-world event.
- Thus it is possible to understand what occurs in the real-world without actually observing its occurrence.
- Let us look at an example. Suppose there is a bank with four tellers.

Simulation of a Bank

- A customer enters the bank at a specific time (t_1) desiring to conduct a transaction.
- Any one of the four tellers can attend to the customer.
- The transaction (withdraw, deposit) will take a certain period of time (t_2).
- If a teller is free, the teller can process the customer's transaction immediately and the customer leaves the bank at t_1+t_2.