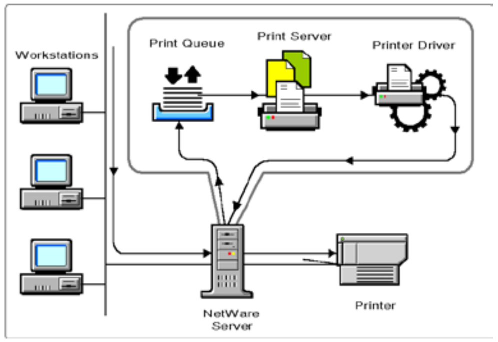
Queues

- Queue Data Container
- STL Overview
- STL Queue Class
- Queue Applications
- Queue Class Implementations
  - Static Array
  - Dynamic Linked List
- Queue Variations

Queue Data Container



- Data structure of ordered entries where entries are inserted at one end (aka rear) and removed from the other end (aka front)
  - enqueue → enter queue
  - de-queue → delete from queue
- Entries removed in order
  - First-In/First-Out (FIFO)

STL Container Class Categories

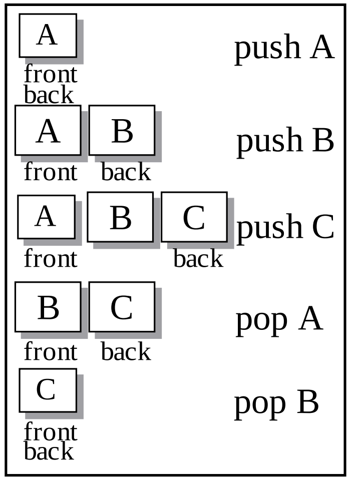
- Sequence
  - stores data by position in linear order
    - vector, deque, list
- Adapter
  - has *another* container as its underlying data structure
  - restricted set of operations on underlying data structure
    - stack, queue, priority_queue
- Associative
  - stores data by key
  - bears no relationship to element location in container
    - set, multiset, map, multimap

STL Containers and Header Files

Sequence	
vector	<vector>
deque	<deque>
list	<list>
Adapters	
stack	<stack>
queue	<queue>
priority_queue	<queue>
Associative	
set	<set>
multiset	<set>
map	<map>
multimap	<map>

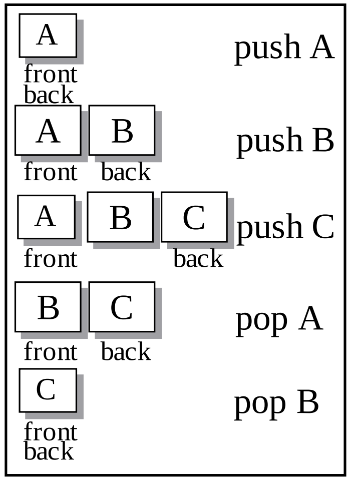
STL Queue Class

- Template-based class that stores elements of *same* data type
  - allows insertions at one end and removal from the other end
- Online API
  - http://www.sgi.com/tech/stl/queue.html
  - http://cppreference.com/cppqueue/index.html
- *Can* be implemented using lists or deques as the underlying data structure. *By default, it uses deque as the base.*
  - `queue<type> name;`
  - `queue<type, list<type> > name;`



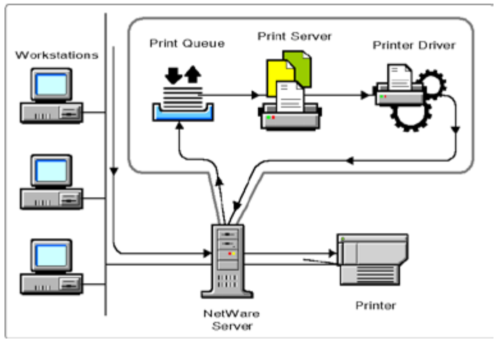
STL Queue Class

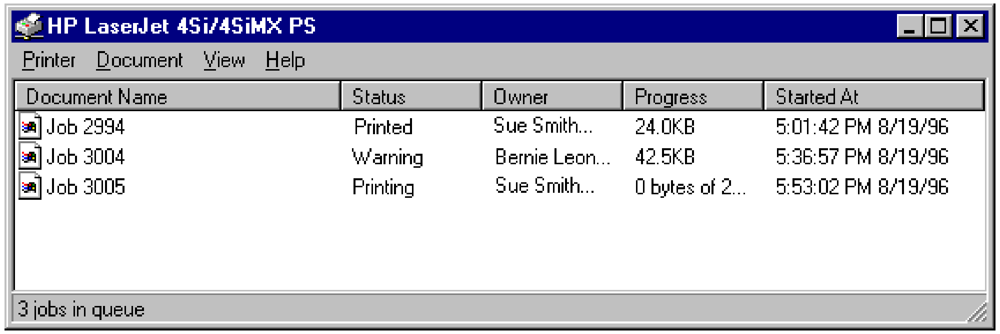
- Required header file
  - `#include <queue>`
- Queue Methods
  - pop → void function that *removes* item at *front* of queue
  - push → void function that *inserts* item at *back* of queue
  - empty → bool function that determines if queue is empty
  - size → returns number of items in queue
  - front → returns reference to item at *front* of queue without removing it
  - back → returns reference to item at *back* of queue without removing it
- Queue Errors
  - Underflow → popping an item from an empty queue
  - Overflow → pushing an item to a full queue



Queue Applications

- User Request for Limited Computer Resources



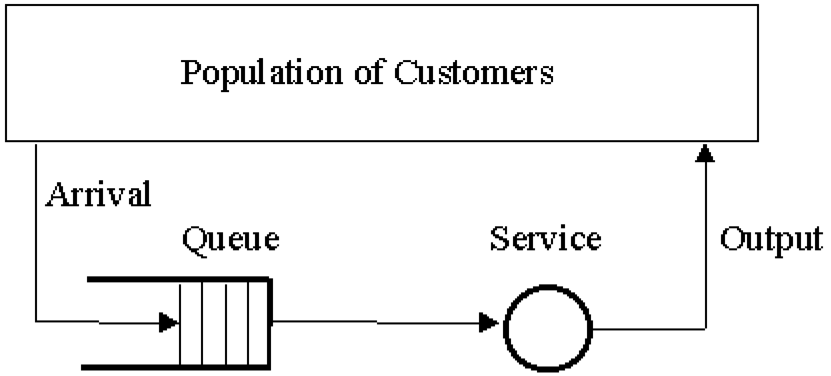


Document Name	Status	Owner	Progress	Started At
Job 2994	Printed	Sue Smith...	24.0KB	5:01:42 PM 8/19/96
Job 3004	Warning	Bernie Leon...	42.5KB	5:36:57 PM 8/19/96
Job 3005	Printing	Sue Smith...	0 bytes of 2...	5:53:02 PM 8/19/96

3 jobs in queue

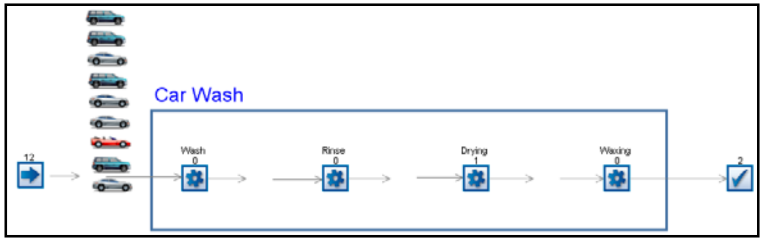
Queue Applications

- Queuing Simulation
  - Analyzes how systems distribute limited resources to elements requesting these resources
    - Arrival time
    - Service time
    - Resource(s) used



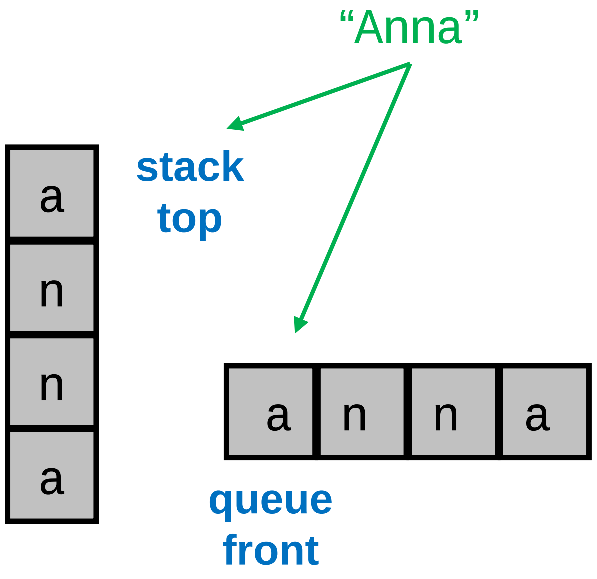
Queue Applications

- Textbook Car Wash Simulation
  - Inputs
    - time to wash car (in seconds)
    - probability of customer arrival during any given second
    - simulation time (in seconds)
  - Outputs
    - number of customers served during simulated time
    - average customer wait time



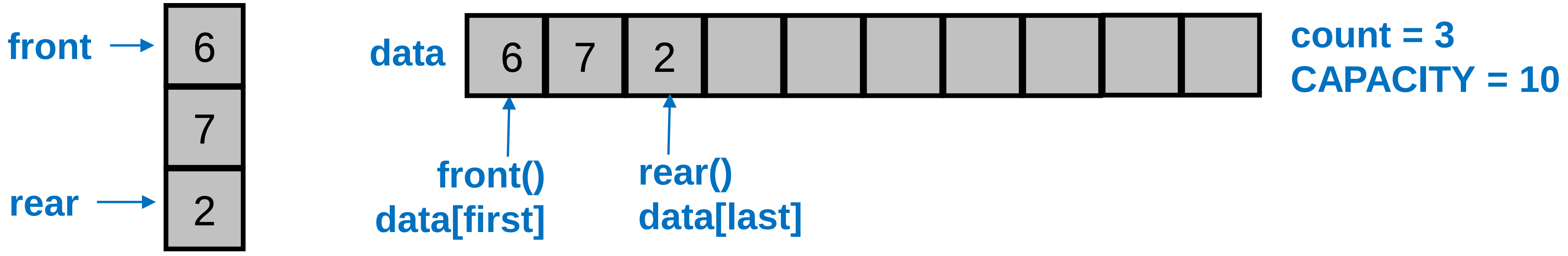
Queue Applications

- Recognizing Palindromes
  - Scan characters from left to right
  - If character is
    - alpha → push character on queue
      → push character on stack
  - End of characters reached
  - Remove characters from queue and stack
    - queue front != stack top
      - increment nonmatches
    - pop character from queue and stack
  - Palindrome if nonmatches == 0



Queue Class Implementations

- Static implementation using fixed-sized array   Array Implementation
- Rules for implementation
  - Array called **`data`** holds up to **`CAPACITY`** items
  - *Number* of items stored in member variable **`count`**
  - If **non-empty**, items stored in a **`circular`** array beginning at **`data[first]`** and continuing through **`data[last]`**
  - If **empty**, **`last`** is valid index and **`first`** is equal to **`next_index(last)`**



Helper Function

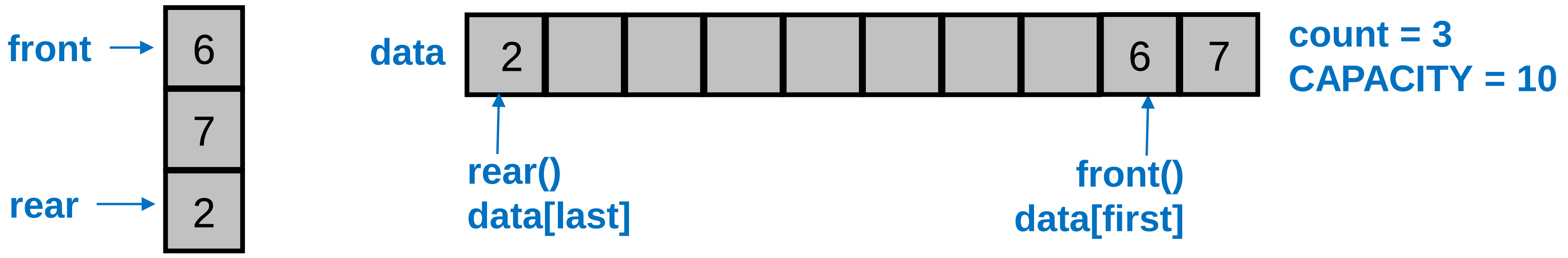
- Private member function useful for implementation but not part of public interface
  - `next_index` function easily steps through array with wraparound at end
    - `return (i + 1) % CAPACITY;`
      - returns `i+1` except when `i = CAPACITY - 1`
        - returns `0`
    - helps in code clarity because it implies the '*next index in the array*'

Queue Class Implementations

- Static implementation using fixed-sized array

Array Implementation

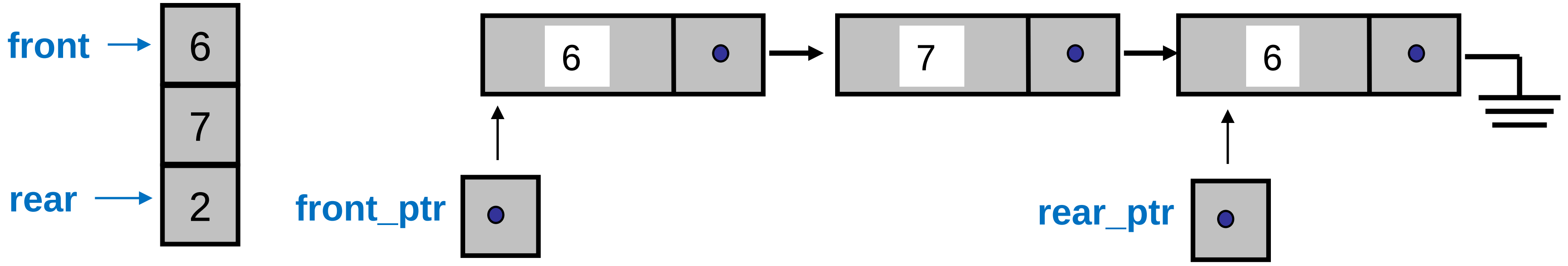
- Rules for implementation
  - Array called `data` holds up to `CAPACITY` items
  - *Number* of items stored in member variable `count`
  - If **non-empty**, items stored in a `circular` array beginning at `data[first]` and continuing through `data[last]`
  - If **empty**, `last` is valid index and `first` is equal to `next_index(last)`



Queue Class Implementations

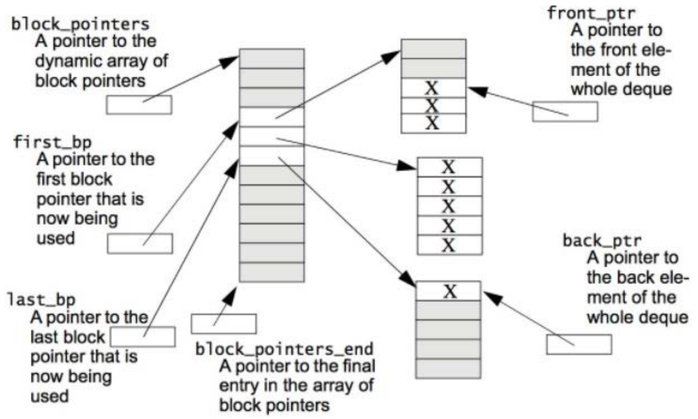
Linked List Implementation

- Dynamic implementation using linked list
- Rules for implementation
  - *Number* of items stored in member variable `count`
  - Items stored in **linked list**, with **front** of queue stored at head node, and **rear** of the queue stored at final node
  - `front_ptr` is head pointer of linked list of items
  - If **non-empty**, member variable `rear_ptr` is tail pointer
  - If **empty**, `rear_ptr` not important



Double Ended Queue (Deque)

- Template-based indexed sequence container class that stores elements of same data type
  - allows insertions and deletions at either end
  - elements not stored contiguously; typical implementation of series of fixed sized arrays
  - expanded as needed
- Header:
  - `#include <deque>`
- Format:
  - `deque<type> name;`
- Online API
  - http://www.sgi.com/tech/stl/Deque.html
  - http://en.cppreference.com/w/cpp/container/deque


block_pointers
A pointer to the dynamic array of block pointers
first_bp
A pointer to the first block pointer that is now being used
last_bp
A pointer to the last block pointer that is now being used
block_pointers_end
A pointer to the final entry in the array of block pointers
front_ptr
A pointer to the front element of the whole deque
back_ptr
A pointer to the back element of the whole deque
X

Priority Queue

- Template-based class that stores elements of same data type
  - allows insertions at one end and deletions from the other end
  - priority ordering (x < y, then y has higher priority)
- *Can* be implemented using vectors or deques as the underlying data structure. *By default, it uses vector as the base.*
  - **`priority_queue<type> name;`**
  - **`priority_queue <type, deque<type> > name;`**
- Online API
  - http://www.sgi.com/tech/stl/priority_queue.html
  - http://cppreference.com/cpppriority_queue/index.html