CS 370

Multithreaded Virtual Network Simulation

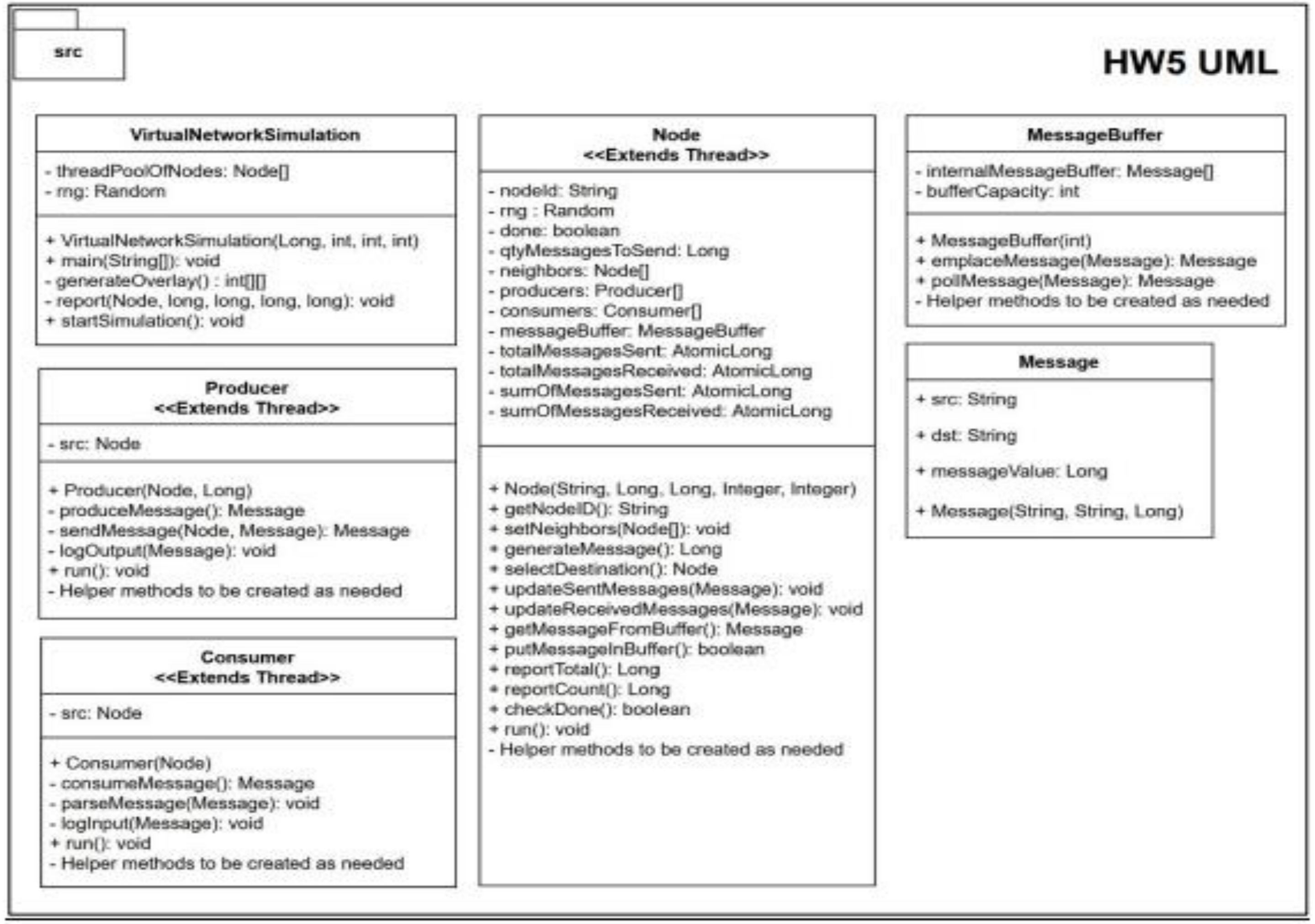
Homework 5

Assignment Review

- You will implement a solution to a version of the Producer and Consumer problem, using a circular buffer.
- High Level: we create a network of Nodes that shall exchange messages with its direct neighboring nodes
- Your program will take a Seed, (N) nodes with (K) neighbors, and (B) buffer size as parameters.
- VirtualNetworkSimulation is responsible for creating Nodes and initiating the simulation.
- The Simulation class shall create a
  - If K (num of neighbors) < N (total nodes) – 1, then each node is not connected to every other node. This means at least one pair of nodes does not have a direct connection, and in this simulation they will not communicate.
  - If K == N – 1 then each node is a neighbor to each node
- Each node will exchange messages with neighboring nodes
- Each node has a MessageBuffer where messages from neighboring nodes Producer Thread are sent
- Then the Node can consume using its Consumer Thread from its MessageBuffer

Which files are required?

- VirtualNetworkSimulation.java
- MessageBuffer.java
- Consumer.java
- Producer.java
- Node.java
- Message.java
- Makefile
- README.txt



src

HW5 UML

VirtualNetworkSimulation

- threadPoolOfNodes: Node[]
- rng: Random

+ VirtualNetworkSimulation(Long, int, int, int)
+ main(String[]): void
- generateOverlay() : int[][]
- report(Node, long, long, long, long): void
+ startSimulation(): void

Producer
<<Extends Thread>>

- src: Node

+ Producer(Node, Long)
- produceMessage(): Message
- sendMessage(Node, Message): Message
- logOutput(Message): void
+ run(): void
- Helper methods to be created as needed

Consumer
<<Extends Thread>>

- src: Node

+ Consumer(Node)
- consumeMessage(): Message
- parseMessage(Message): void
- logInput(Message): void
+ run(): void
- Helper methods to be created as needed

Node
<<Extends Thread>>

- nodeId: String
- rng : Random
- done: boolean
- qtyMessagesToSend: Long
- neighbors: Node[]
- producers: Producer[]
- consumers: Consumer[]
- messageBuffer: MessageBuffer
- totalMessagesSent: AtomicLong
- totalMessagesReceived: AtomicLong
- sumOfMessagesSent: AtomicLong
- sumOfMessagesReceived: AtomicLong

+ Node(String, Long, Long, Integer, Integer)
+ getNodeID(): String
+ setNeighbors(Node[]): void
+ generateMessage(): Long
+ selectDestination(): Node
+ updateSentMessages(Message): void
+ updateReceivedMessages(Message): void
+ getMessageFromBuffer(): Message
+ putMessageInBuffer(): boolean
+ reportTotal(): Long
+ reportCount(): Long
+ checkDone(): boolean
+ run(): void
- Helper methods to be created as needed

MessageBuffer

- internalMessageBuffer: Message[]
- bufferCapacity: int

+ MessageBuffer(int)
+ emplaceMessage(Message): Message
+ pollMessage(Message): Message
- Helper methods to be created as needed

Message

+ src: String
+ dst: String
+ messageValue: Long
+ Message(String, String, Long)

+ denotes public, - denotes private. Format: (public/private) name(arg_types): return_type

VirtualNetworkSimulation.java

- Creates your Network: A network is a graph of Nodes with some ordering
- Given N Nodes each with K neighboring Nodes, creates an overlay of connections.
    - This Overlay represents which Nodes can pass messages between each other.
- Determines number of messages to pass globally.
- At the end of the Simulation (once all Nodes signal that they are done), this class shall collect and display the sums and counts of each Node and Total

Node.java

- A Node is a vertice in the network graph and is responsible for producing M/N messages distributed among its K neighboring Nodes
- Has one private MessageBuffer instance accessed via public methods getMessageFromBuffer() and putMessageInBuffer(Message).
- Has K Producers and K Consumer threads
- Responsible for keeping track of messages sent and received Atomically.

MessageBuffer.java

- Each Node has a MessageBuffer which is a FIFO circular bounded buffer
- This array contains Messages instances
- This Buffer must be implemented in a thread safe manner for this to work

Producer.java

- Extends Thread, override public void run(){ … Thread Logic … }
- Each node has K Producer threads that can send to any Neighbor
- An instance of this class produces messages to be send to other nodes MessageBuffer using that Nodes public methods
- Waits when the MessageBuffer is full

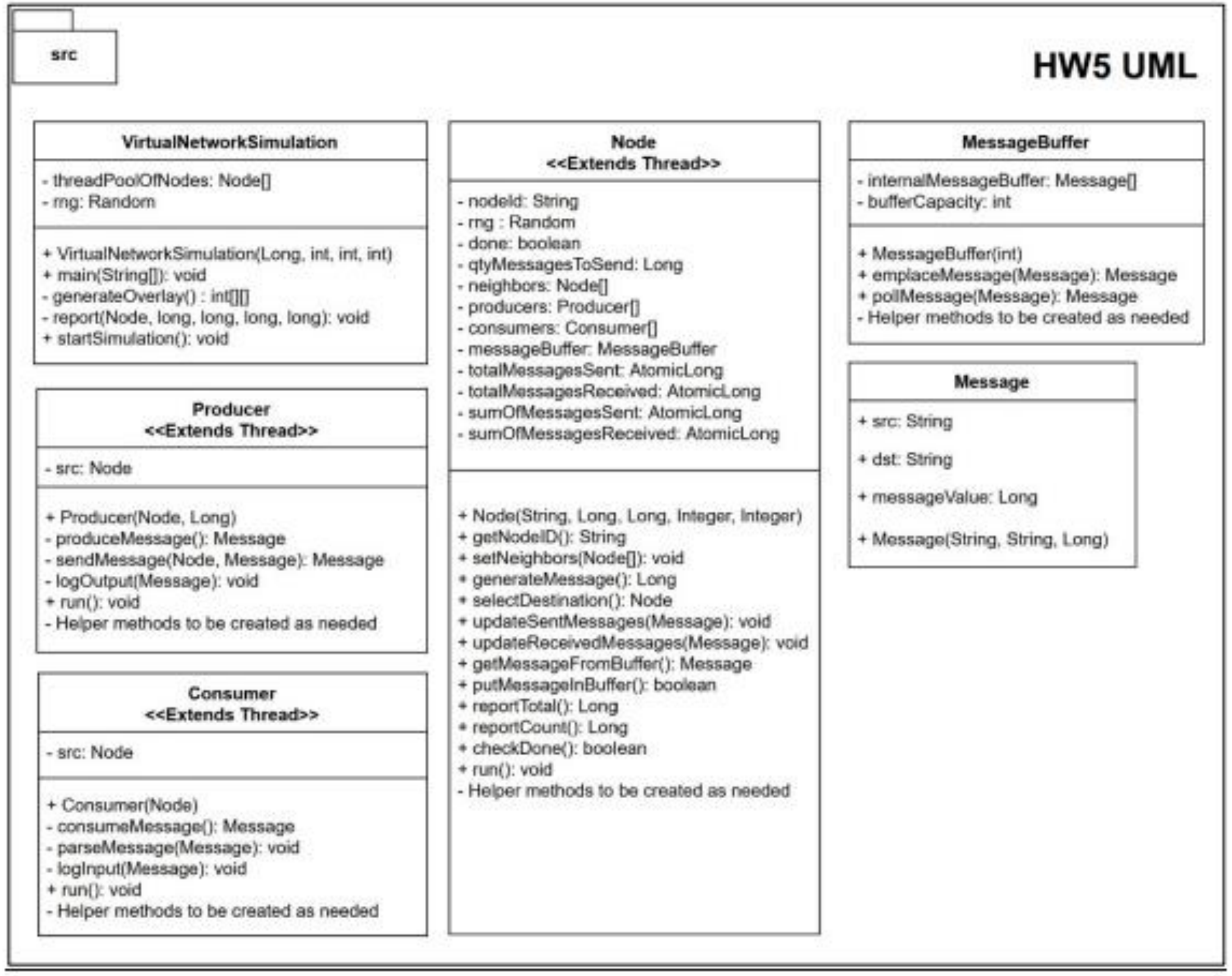
Consumer.java

- Extends Thread, @Override public void run() { … Thread Logic … }
- Each Node has K consumer Threads
- This is where our messages (instances of the Message class) are consumed
- These consumed messages come from Producers of neighboring nodes
- Consumer must inform its src Node of messages consumed

Message.java

1. This is a data object Class that represents the Messages passed between Nodes via Producers and Message Buffers and processed by Consumers.
   - Contains only three public class vars
     - A string src
     - A string dst
     - A string messageValue
   - Uses built in Java Object Serialization

UML Diagram



src

HW5 UML

VirtualNetworkSimulation

- threadPoolOfNodes: Node[]
- rng: Random

+ VirtualNetworkSimulation(Long, int, int, int)
+ main(String[]): void
- generateOverlay() : int[][]
- report(Node, long, long, long, long): void
+ startSimulation(): void

Producer
<<Extends Thread>>

- src: Node

+ Producer(Node, Long)
- produceMessage(): Message
- sendMessage(Node, Message): Message
- logOutput(Message): void
+ run(): void
- Helper methods to be created as needed

Consumer
<<Extends Thread>>

- src: Node

+ Consumer(Node)
- consumeMessage(): Message
- parseMessage(Message): void
- logInput(Message): void
+ run(): void
- Helper methods to be created as needed

Node
<<Extends Thread>>

- nodeId: String
- rng : Random
- done: boolean
- qtyMessagesToSend: Long
- neighbors: Node[]
- producers: Producer[]
- consumers: Consumer[]
- messageBuffer: MessageBuffer
- totalMessagesSent: AtomicLong
- totalMessagesReceived: AtomicLong
- sumOfMessagesSent: AtomicLong
- sumOfMessagesReceived: AtomicLong

+ Node(String, Long, Long, Integer, Integer)
+ getNodeID(): String
+ setNeighbors(Node[]): void
+ generateMessage(): Long
+ selectDestination(): Node
+ updateSentMessages(Message): void
+ updateReceivedMessages(Message): void
+ getMessageFromBuffer(): Message
+ putMessageInBuffer(): boolean
+ reportTotal(): Long
+ reportCount(): Long
+ checkDone(): boolean
+ run(): void
- Helper methods to be created as needed

MessageBuffer

- internalMessageBuffer: Message[]
- bufferCapacity: int

+ MessageBuffer(int)
+ emplaceMessage(Message): Message
+ pollMessage(Message): Message
- Helper methods to be created as needed

Message

+ src: String

+ dst: String

+ messageValue: Long

+ Message(String, String, Long)

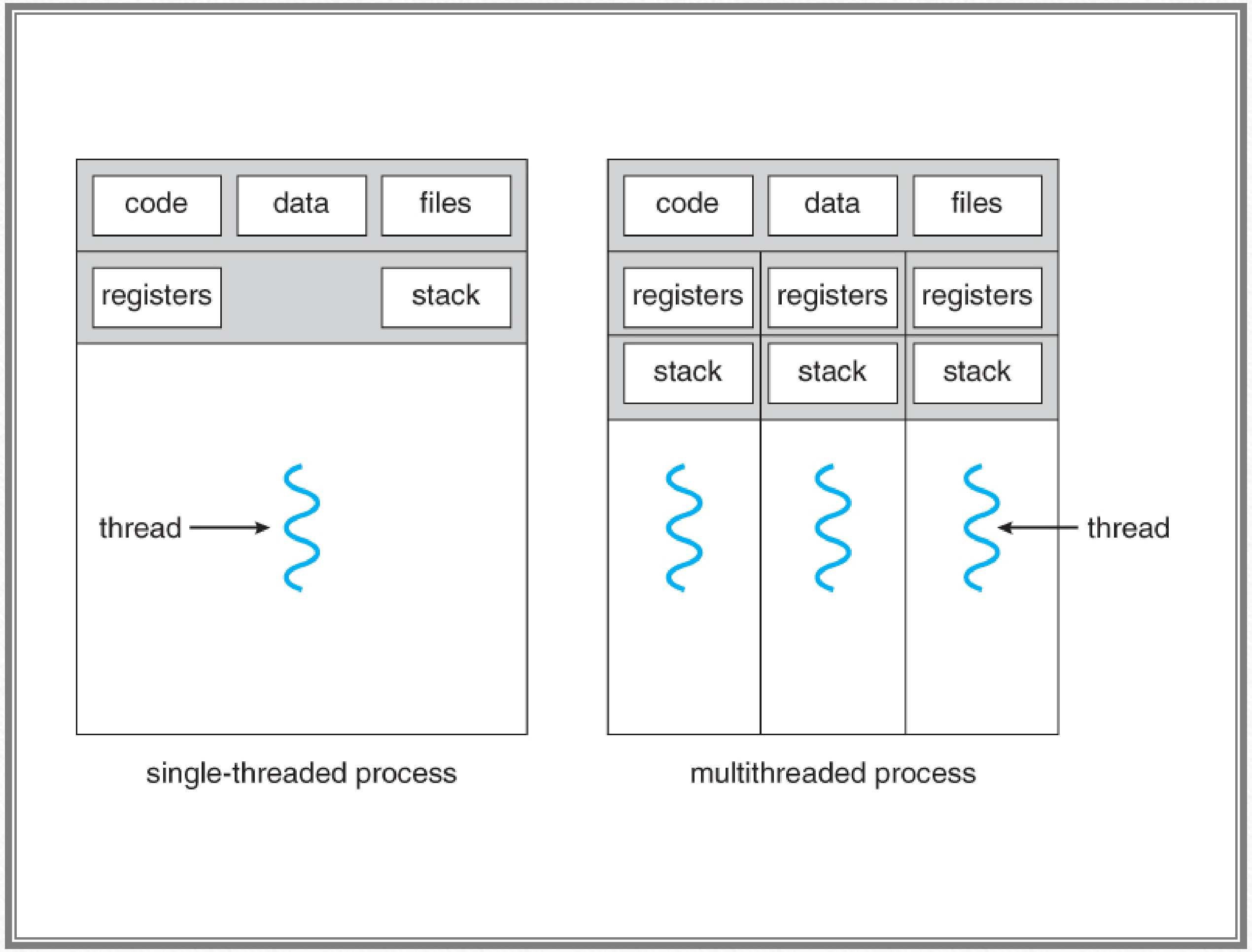
+ denotes public, - denotes private. Format: (public/private) name(arg_types): return_type

- This UML Diagram lists all the required Classes and Methods for you to implement in this program.
- We just went over these

Synchronization in Java

- Java has inbuilt monitors
  - Allows threads to have mutual exclusion
  - Allows threads the ability to wait (block) for a condition to become true
- Signaling is done using
  - wait()
  - notify() or notifyAll()
- Built in thread class can be extended and used
  - Instantiate and use myThread.start()
  - @Override run() to change what a thread does

Threads



```
public class PhilosopherThread
extends Thread
{
    @Override
    public void run()
    {
        // Thread entry point
    }
}
```

Creating and Starting threads

```
public class PhilosopherThread extends Thread {
    @Override
    public void run()  {
        // Thread entry point
    }
}
```

```
PhilosopherThread Socrates = new PhilosopherThread(table, seat);

Socrates.start(); //begins Socrates thread invokes the run() method
```

Synchronized methods

- A piece of logic marked with synchronized becomes a synchronized block, allowing only one thread to execute at any given time.

```
public synchronized void pickup(int i) throws InterruptedException

{

    //Synchronized code goes in here

}
```

wait(), notify() and notifyAll()

- wait()
  - Causes current thread to wait until another thread invokes the notify() or notifyAll() method
- notify()
  - notify() wakes up one thread waiting for the lock
- notifyAll()
  - The notifyAll() method wakes up all the threads waiting for the lock; the JVM selects one of the threads from the list of threads waiting for the lock and wakes that thread up

Java Dining Philosophers Example

- Please see the code for

  Self Exercise 6 Java threads and Synchronization Example

In Teams > Self Exercises

CS 370

Raspberry Pi

Topics

- Intro to Raspberry Pi
- Setting up a Raspberry Pi
- Term Project Requirements
- Term Project Expectations
- Helpful Links

Why Raspberry Pi’s

- Small and Portable
- Cheap
- Well-Documented
- Versatile
- Support for many peripherals (thanks to Linux)

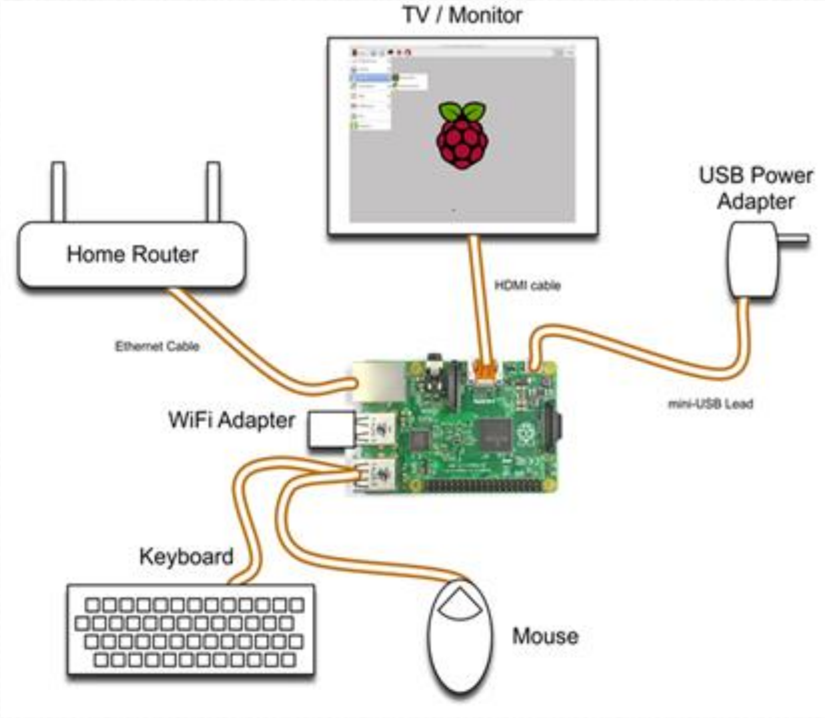
Third Best Selling Computer Brand in the World

Raspberry Pi Models

Raspberry Pi 4 Model B+

- 1.5GHz 64-bit quad-core processor
- dual-band wireless LAN
- Bluetooth 5.0/BLE
- Gigabit Ethernet
- Power-over-Ethernet support (with separate PoE HAT)
- 2 x micro-HDMI ports (up to 4kp60 supported)

Raspberry Pi Setup

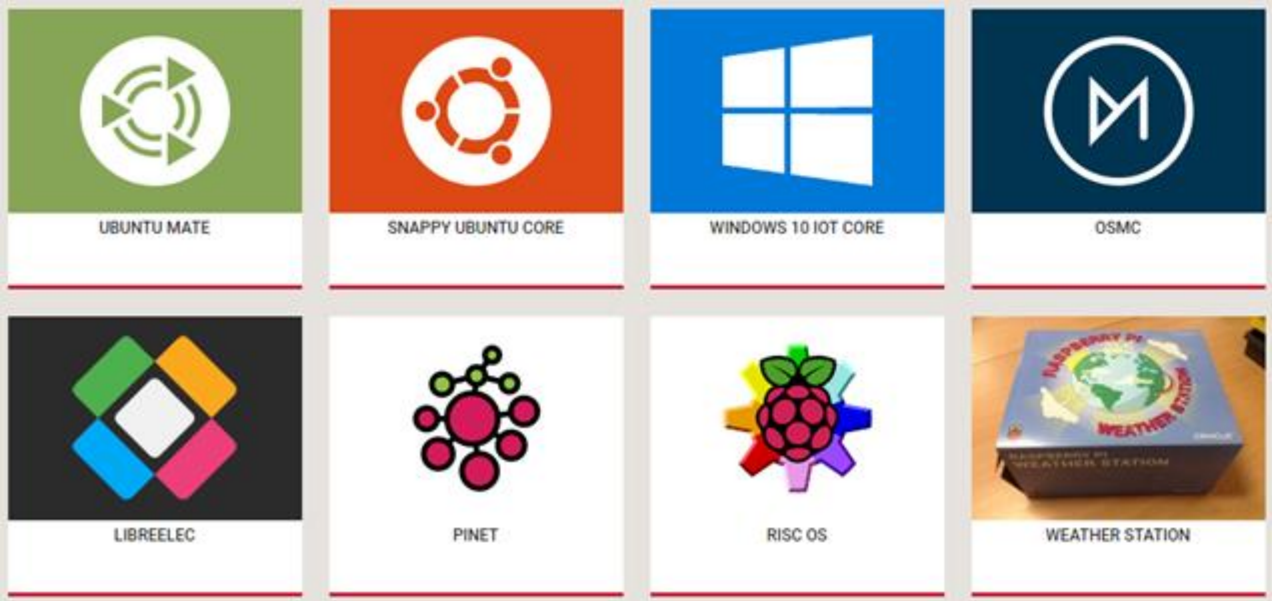


Can connect to monitor, keyboard, mouse

Usable as a normal desktop

Optionally use *ssh* instead of a monitor

Raspberry Pi Operating Systems



Expect most groups to use Raspbian (officially supported OS)

Other options are available - some OS's for specific use cases

Programming Languages

Basically any language will work (Python, C, Java, C++, Javascript, Ruby, Lisp, Rust, R, etc…)

Most projects done in Python or C

GPIO Libraries

Python/C

- RPi.GPIO (Python)
  - RPi.GPIO code samples
- RPIO.GPIO (Python)
- wiringPi (Python/C)
- pigpio (Python/C/Javascript)
- gpiozero (Python)
- bcm2835 (C)

Term Project Requirements

Project must involve:

- A single board computer (Raspberry Pi)
  - With WiFi capability + operating system
- Communication with at least one other computer
  - Another board, desktop, assistant, etc.
- At least one sensing or interacting device
  - Heat sensor, motion detector, camera, motor, controller, etc...

Term Project Expectations

- Originality
    - Several groups with similar projects (temperature sensors, plant waterers, etc...)
    - Come up with a unique selling point
        - Find similar projects online, then do something different
- Thoroughness
    - Think about the evaluations you're performing - design careful experiments and control for variables
    - Try to learn something you couldn't have guessed

Helpful Links

- Help Guides
  - Setup instructions
  - SSH with Raspberry Pi's
  - Help videos
  - FAQ's
  - Embedded Linux wiki

- Forums and Tutorials
  - Raspberry Pi forums / projects
  - Hackaday Projects
  - Adafruit Learning Guides
  - Raspberry Pi subreddit

Thank You

Questions?