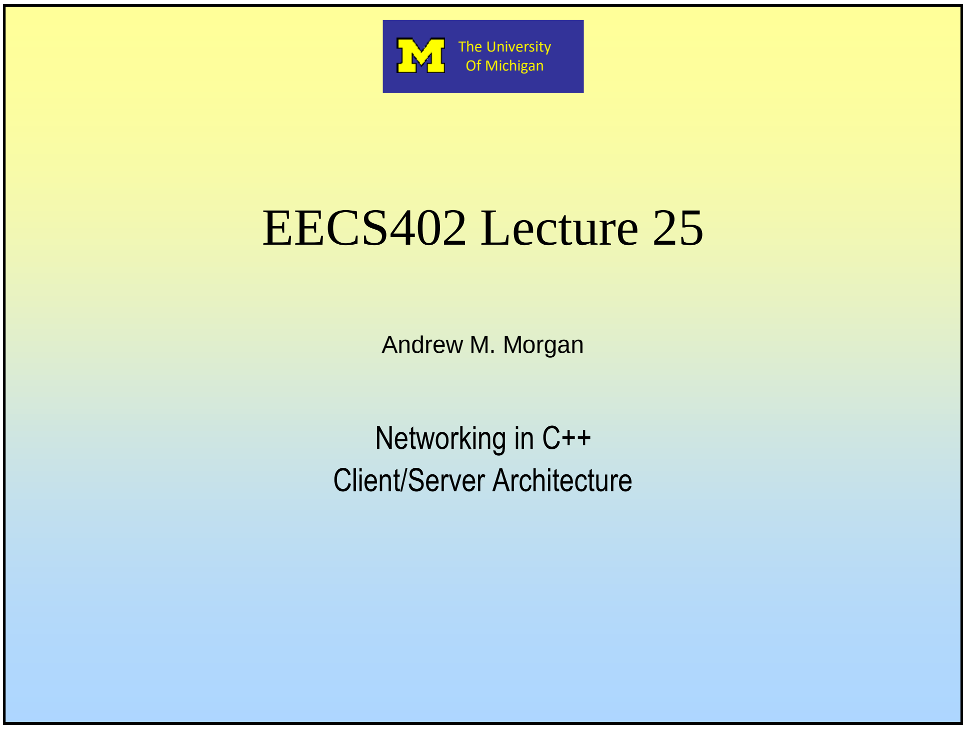
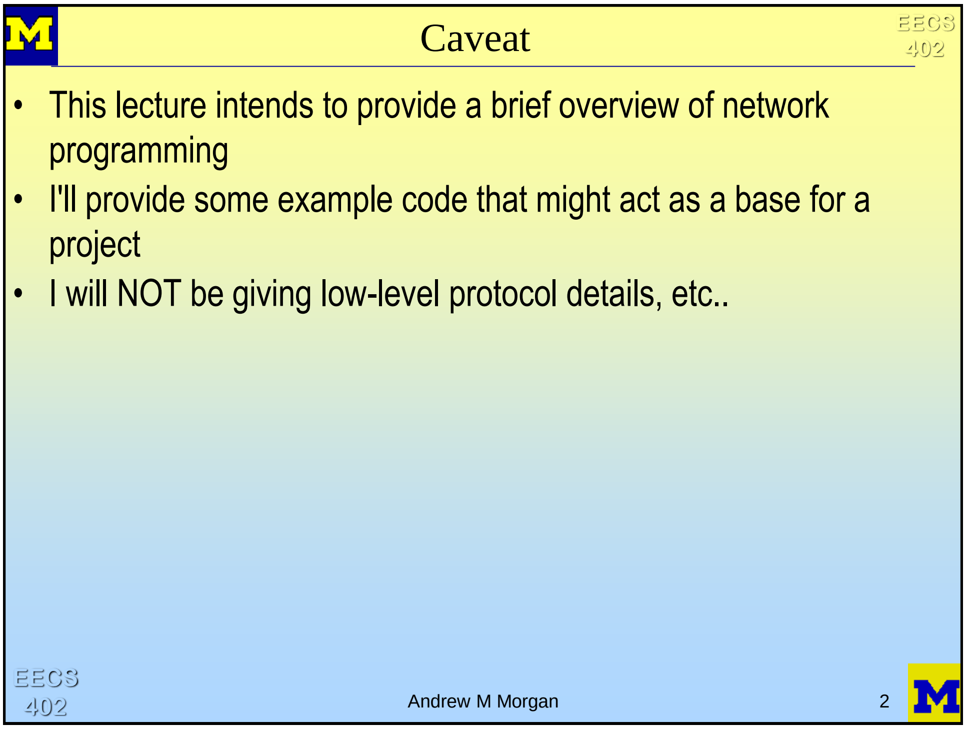


The University Of Michigan

EECS402 Lecture 25

Andrew M. Morgan

Networking in C++
Client/Server Architecture



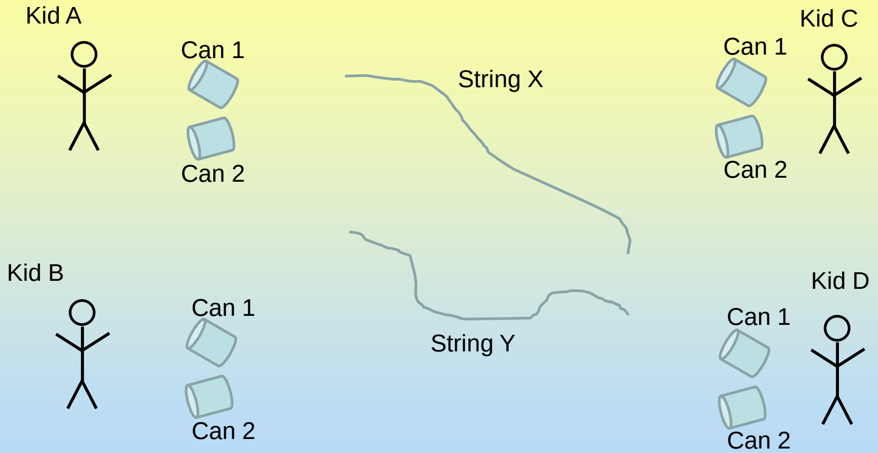
Caveat

- This lecture intends to provide a brief overview of network programming
- I'll provide some example code that might act as a base for a project
- I will NOT be giving low-level protocol details, etc..

Communicating via Network: Analogy

EECS 402

- Think of this situation:

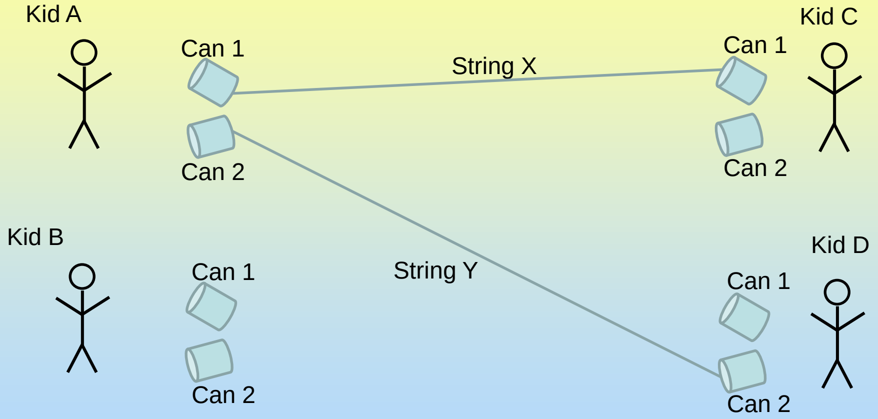


No one can communicate with each other in this case.. The strings are not connecting any cans!

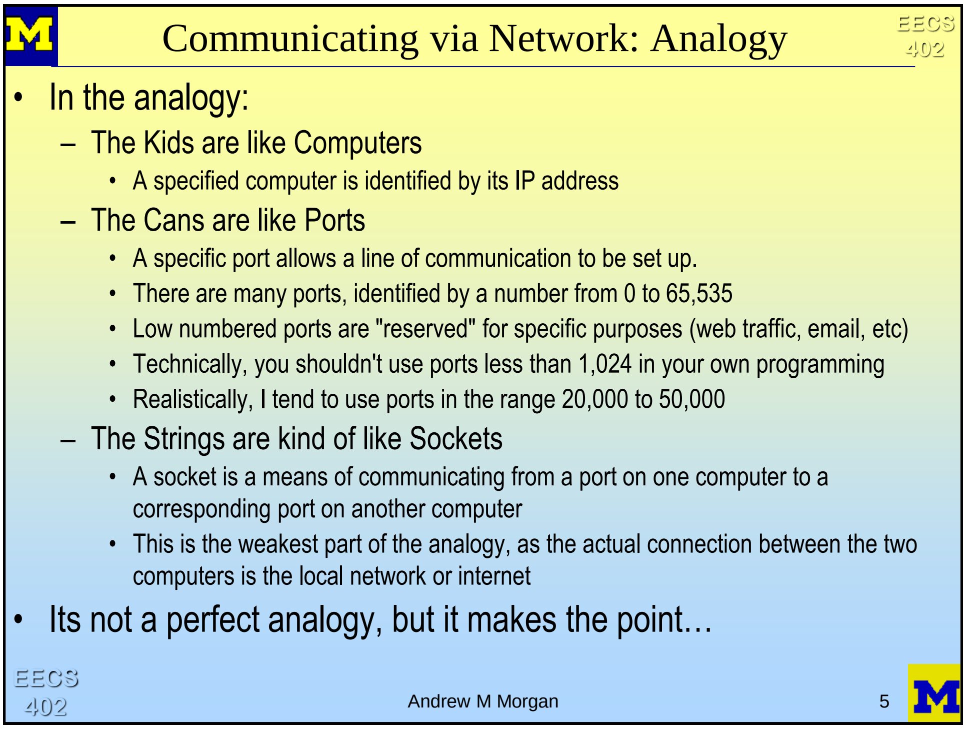
Communicating via Network: Analogy

EECS 402

- At school, Kid A tells Kid D, "at 6:30 tonight, attach the Y string to your can #2"
- Kid A also tells Kid C, "at 6:30 tonight, attach the X string to your can #1"
- At 6:30 that night …

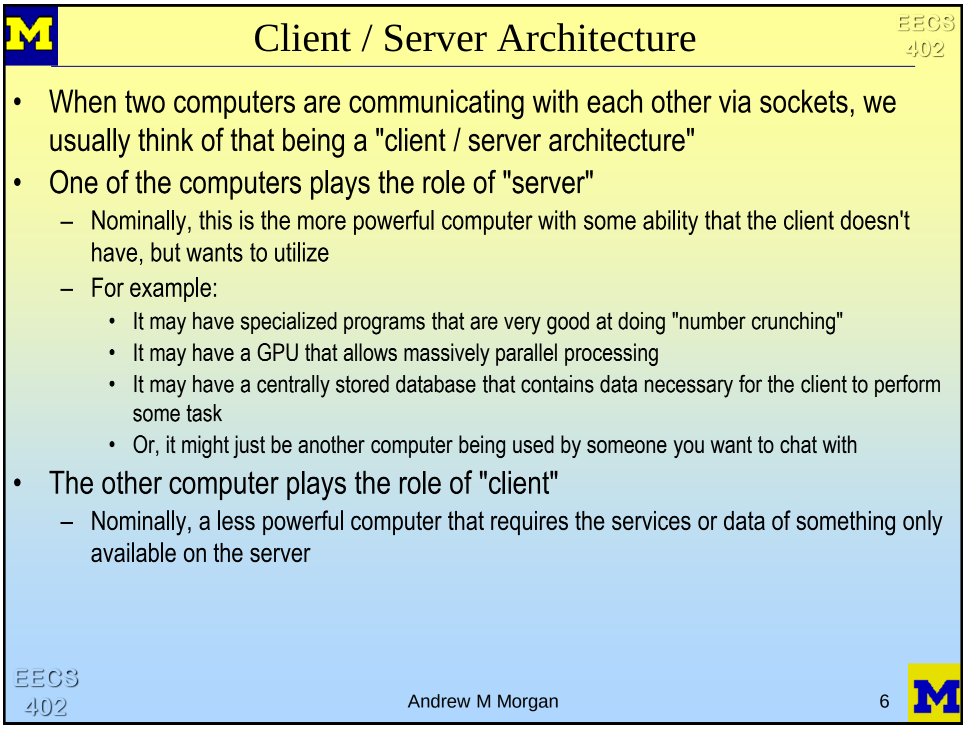


Now, Kid A can communicate with Kid D via Can #2. Kid A can also communicate with Kid C via Can #1.



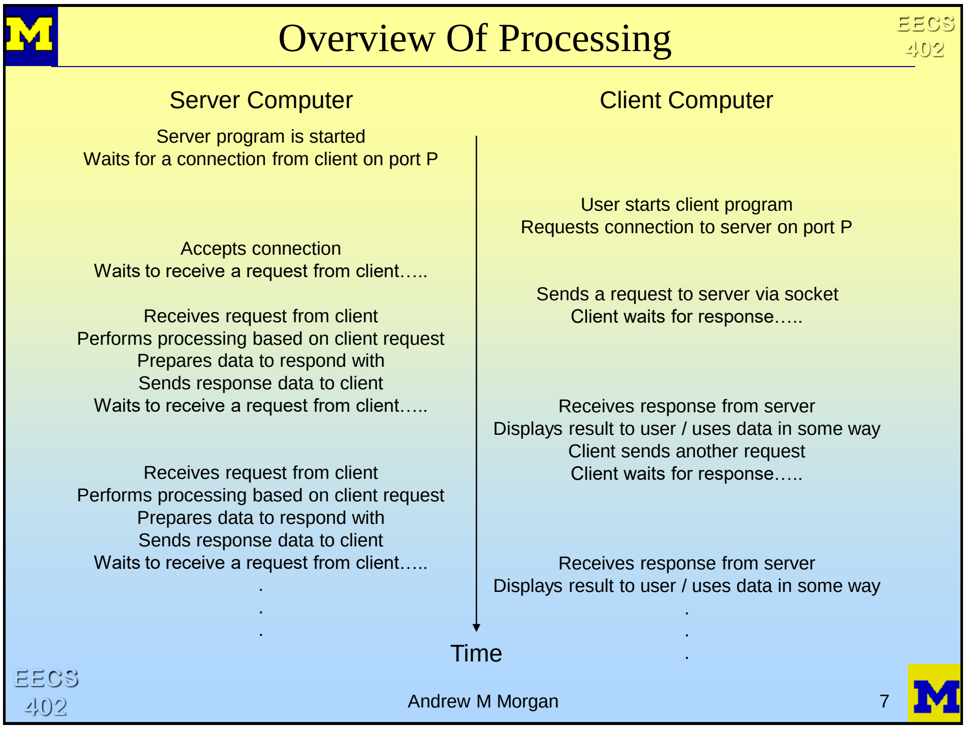
Communicating via Network: Analogy

- In the analogy:
  - The Kids are like Computers
    - A specified computer is identified by its IP address
  - The Cans are like Ports
    - A specific port allows a line of communication to be set up.
    - There are many ports, identified by a number from 0 to 65,535
    - Low numbered ports are "reserved" for specific purposes (web traffic, email, etc)
    - Technically, you shouldn't use ports less than 1,024 in your own programming
    - Realistically, I tend to use ports in the range 20,000 to 50,000
  - The Strings are kind of like Sockets
    - A socket is a means of communicating from a port on one computer to a corresponding port on another computer
    - This is the weakest part of the analogy, as the actual connection between the two computers is the local network or internet
- Its not a perfect analogy, but it makes the point…



Client / Server Architecture

- When two computers are communicating with each other via sockets, we usually think of that being a "client / server architecture"
- One of the computers plays the role of "server"
  - Nominally, this is the more powerful computer with some ability that the client doesn't have, but wants to utilize
  - For example:
    - It may have specialized programs that are very good at doing "number crunching"
    - It may have a GPU that allows massively parallel processing
    - It may have a centrally stored database that contains data necessary for the client to perform some task
    - Or, it might just be another computer being used by someone you want to chat with
- The other computer plays the role of "client"
  - Nominally, a less powerful computer that requires the services or data of something only available on the server



Overview Of Processing

Server Computer

Server program is started
Waits for a connection from client on port P

Accepts connection
Waits to receive a request from client…..

Receives request from client
Performs processing based on client request
Prepares data to respond with
Sends response data to client
Waits to receive a request from client…..

Receives request from client
Performs processing based on client request
Prepares data to respond with
Sends response data to client
Waits to receive a request from client…..
.
.
.

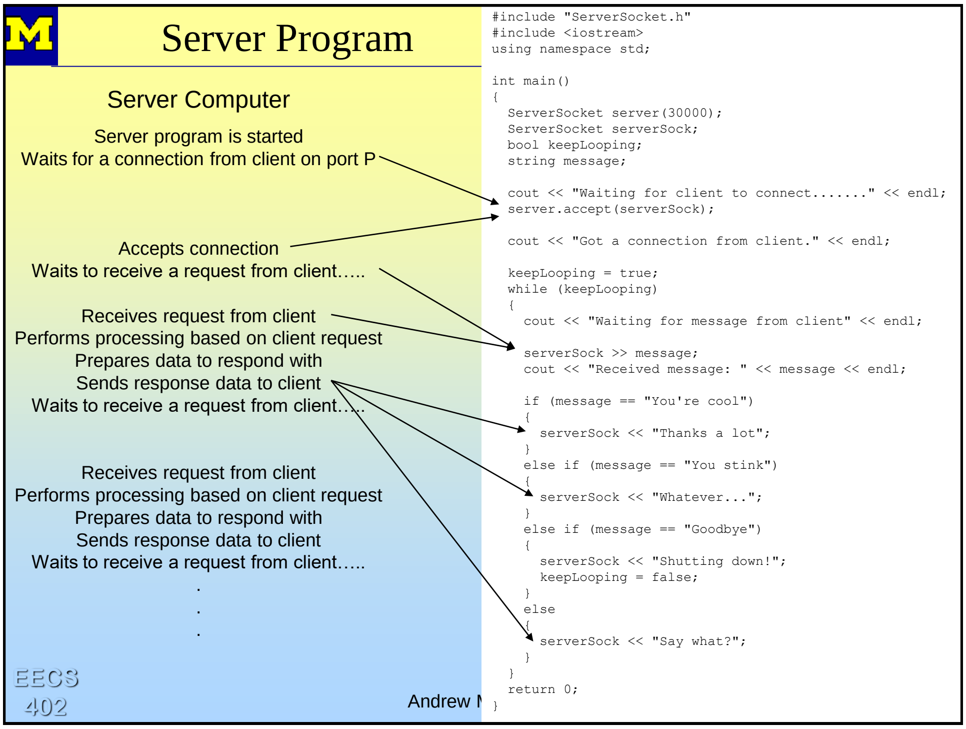
Client Computer

User starts client program
Requests connection to server on port P

Sends a request to server via socket
Client waits for response…..

Receives response from server
Displays result to user / uses data in some way
Client sends another request
Client waits for response…..

Receives response from server
Displays result to user / uses data in some way
.
.
.



Server Program

Server Computer

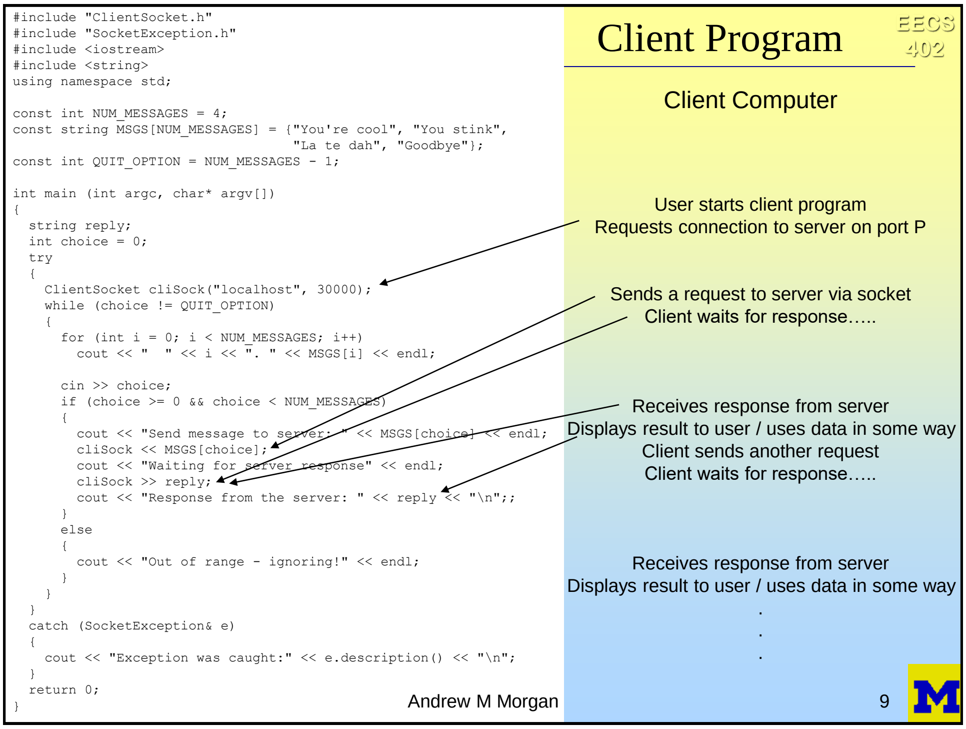
Server program is started
Waits for a connection from client on port P

Accepts connection
Waits to receive a request from client…..

Receives request from client
Performs processing based on client request
Prepares data to respond with
Sends response data to client
Waits to receive a request from client…..

Receives request from client
Performs processing based on client request
Prepares data to respond with
Sends response data to client
Waits to receive a request from client…..
.
.
.

```
#include "ServerSocket.h"
#include <iostream>
using namespace std;

int main()
{
  ServerSocket server(30000);
  ServerSocket serverSock;
  bool keepLooping;
  string message;

  cout << "Waiting for client to connect......." << endl;
  server.accept(serverSock);

  cout << "Got a connection from client." << endl;

  keepLooping = true;
  while (keepLooping)
  {
    cout << "Waiting for message from client" << endl;

    serverSock >> message;
    cout << "Received message: " << message << endl;

    if (message == "You're cool")
    {
      serverSock << "Thanks a lot";
    }
    else if (message == "You stink")
    {
      serverSock << "Whatever...";
    }
    else if (message == "Goodbye")
    {
      serverSock << "Shutting down!";
      keepLooping = false;
    }
    else
    {
      serverSock << "Say what?";
    }
  }
  return 0;
}
```



Client Program

EECS 402

```
#include "ClientSocket.h"
#include "SocketException.h"
#include <iostream>
#include <string>
using namespace std;

const int NUM_MESSAGES = 4;
const string MSGS[NUM_MESSAGES] = {"You're cool", "You stink",
                                   "La te dah", "Goodbye"};
const int QUIT_OPTION = NUM_MESSAGES - 1;

int main (int argc, char* argv[])
{
  string reply;
  int choice = 0;
  try
  {
    ClientSocket cliSock("localhost", 30000);
    while (choice != QUIT_OPTION)
    {
      for (int i = 0; i < NUM_MESSAGES; i++)
        cout << "  " << i << ". " << MSGS[i] << endl;

      cin >> choice;
      if (choice >= 0 && choice < NUM_MESSAGES)
      {
        cout << "Send message to server: " << MSGS[choice] << endl;
        cliSock << MSGS[choice];
        cout << "Waiting for server response" << endl;
        cliSock >> reply;
        cout << "Response from the server: " << reply << "\n";;
      }
      else
      {
        cout << "Out of range - ignoring!" << endl;
      }
    }
  }
  catch (SocketException& e)
  {
    cout << "Exception was caught:" << e.description() << "\n";
  }
  return 0;
}
```

Client Computer

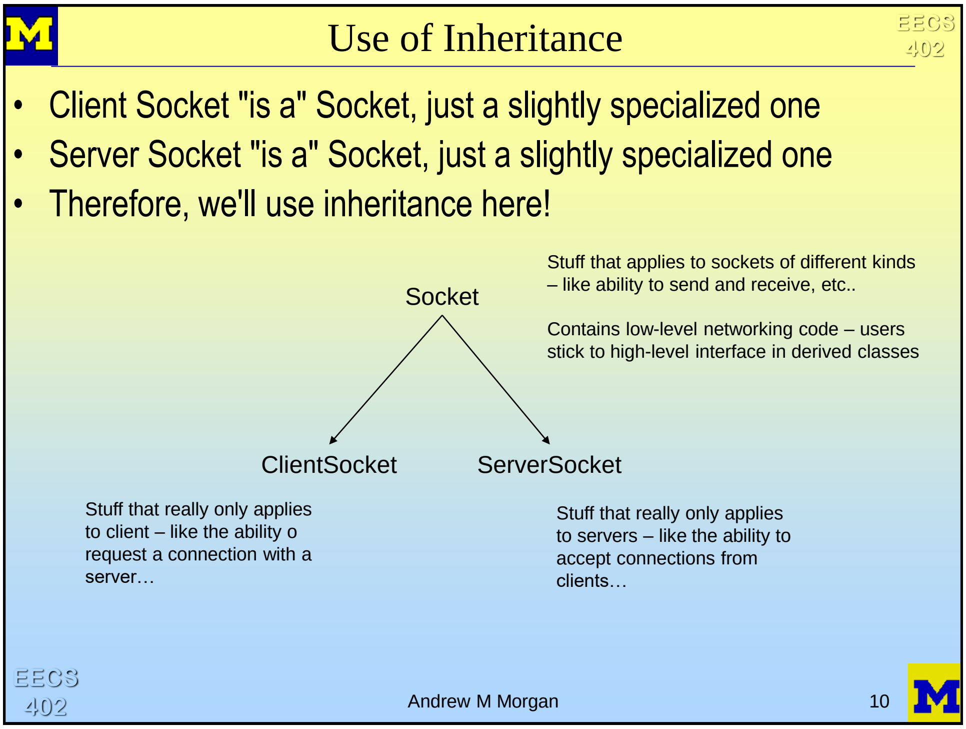
User starts client program
Requests connection to server on port P

Sends a request to server via socket
Client waits for response…..

Receives response from server
Displays result to user / uses data in some way
Client sends another request
Client waits for response…..

Receives response from server
Displays result to user / uses data in some way
.
.
.

Andrew M Morgan 9



Use of Inheritance

EECS 402

- Client Socket "is a" Socket, just a slightly specialized one
- Server Socket "is a" Socket, just a slightly specialized one
- Therefore, we'll use inheritance here!

Stuff that applies to sockets of different kinds – like ability to send and receive, etc..

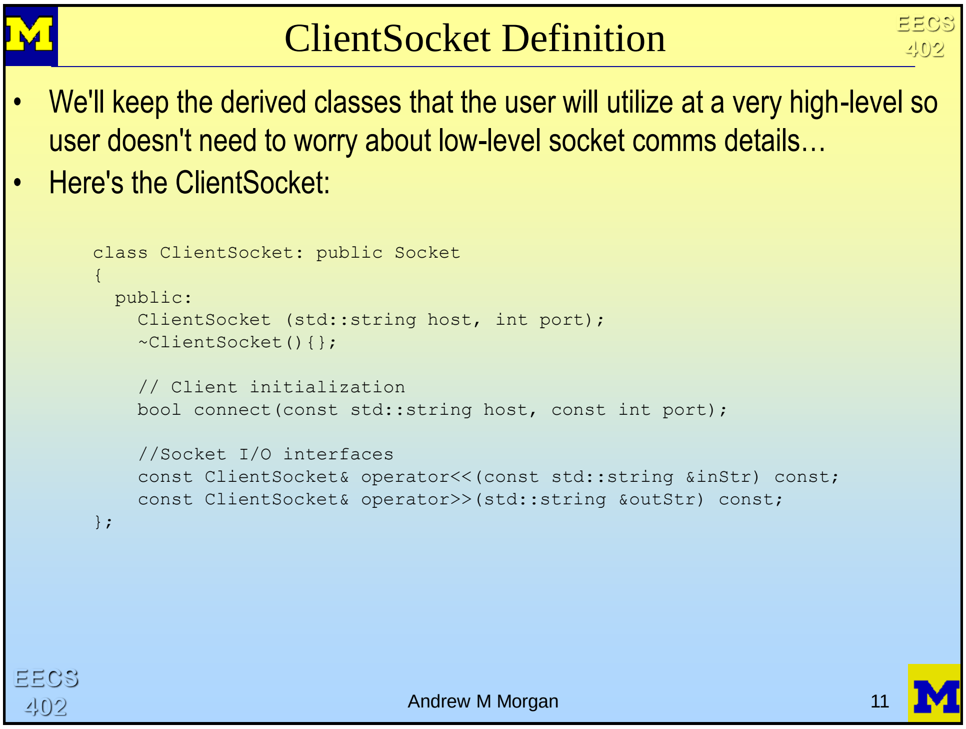
Contains low-level networking code – users stick to high-level interface in derived classes

Stuff that really only applies to client – like the ability o request a connection with a server…

Stuff that really only applies to servers – like the ability to accept connections from clients…

EECS 402

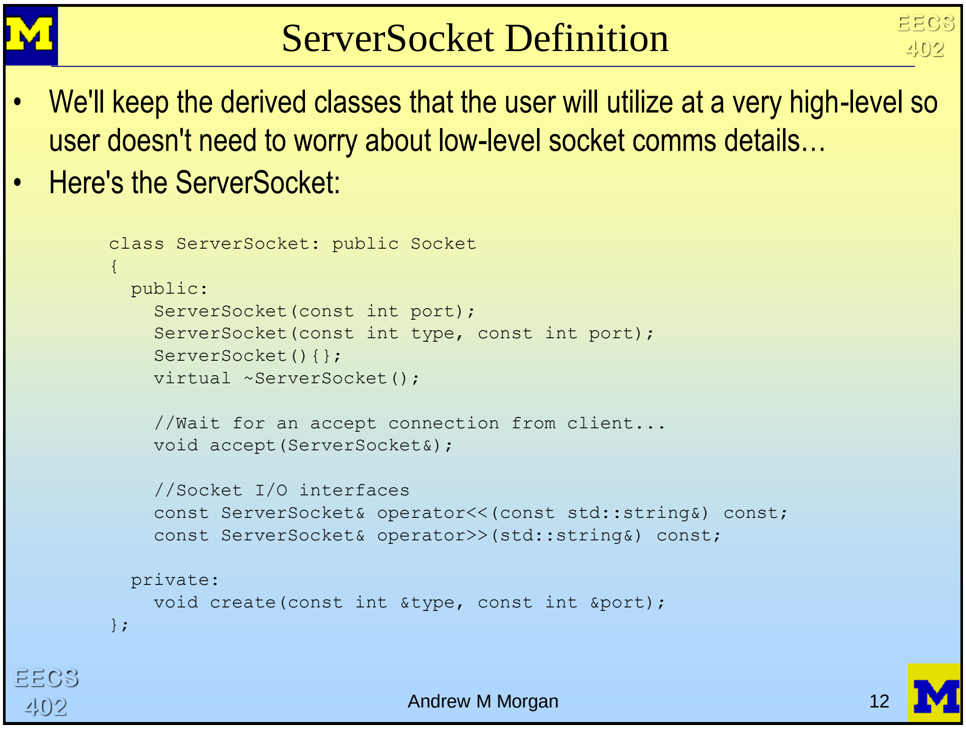
Andrew M Morgan 10



ClientSocket Definition

- We'll keep the derived classes that the user will utilize at a very high-level so user doesn't need to worry about low-level socket comms details…
- Here's the ClientSocket:

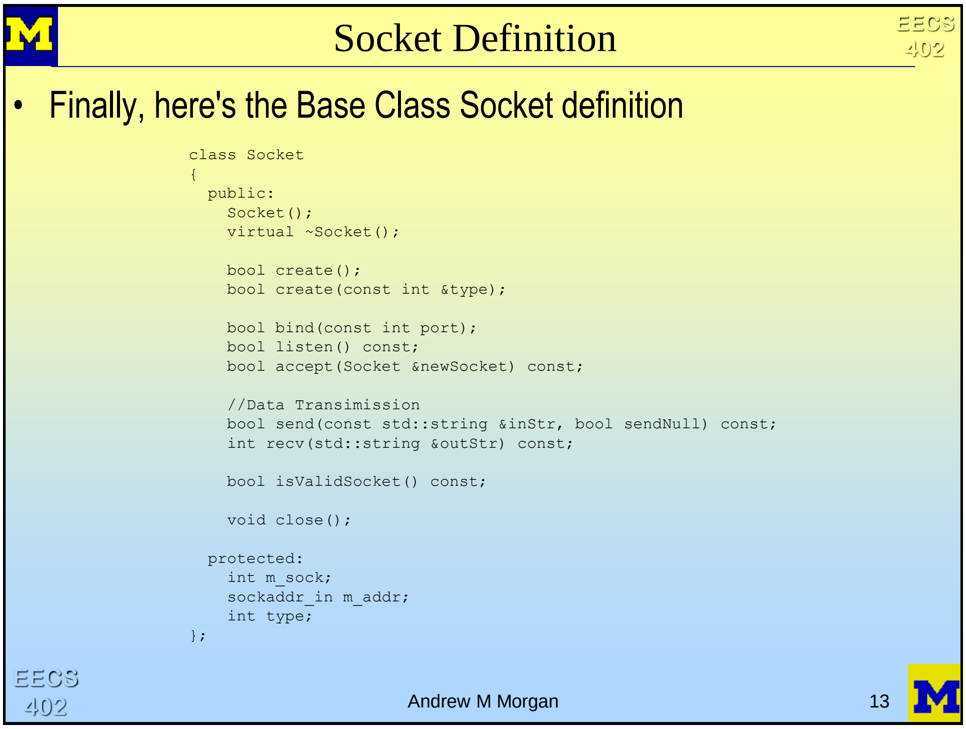
```
class ClientSocket: public Socket
{
  public:
    ClientSocket (std::string host, int port);
    ~ClientSocket(){};

    // Client initialization
    bool connect(const std::string host, const int port);

    //Socket I/O interfaces
    const ClientSocket& operator<<(const std::string &inStr) const;
    const ClientSocket& operator>>(std::string &outStr) const;
};
```



ServerSocket Definition

- We'll keep the derived classes that the user will utilize at a very high-level so user doesn't need to worry about low-level socket comms details…
- Here's the ServerSocket:

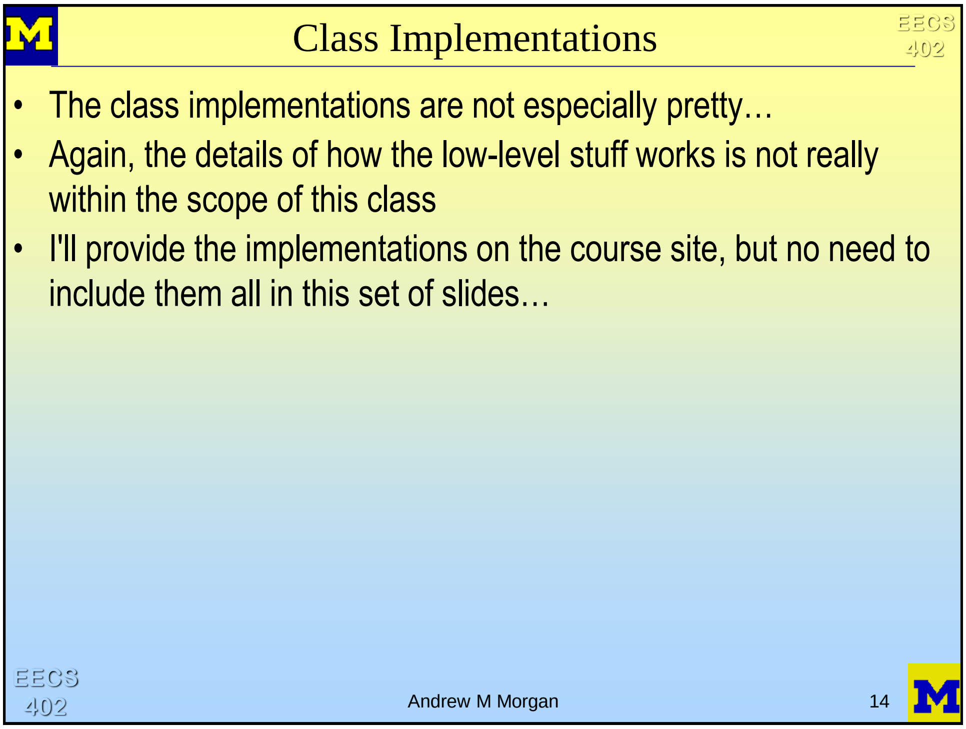
```
class ServerSocket: public Socket
{
  public:
    ServerSocket(const int port);
    ServerSocket(const int type, const int port);
    ServerSocket(){};
    virtual ~ServerSocket();

    //Wait for an accept connection from client...
    void accept(ServerSocket&);

    //Socket I/O interfaces
    const ServerSocket& operator<<(const std::string&) const;
    const ServerSocket& operator>>(std::string&) const;

  private:
    void create(const int &type, const int &port);
};
```



Socket Definition

- Finally, here's the Base Class Socket definition

```
class Socket
{
  public:
    Socket();
    virtual ~Socket();

    bool create();
    bool create(const int &type);

    bool bind(const int port);
    bool listen() const;
    bool accept(Socket &newSocket) const;

    //Data Transimission
    bool send(const std::string &inStr, bool sendNull) const;
    int recv(std::string &outStr) const;

    bool isValidSocket() const;

    void close();

  protected:
    int m_sock;
    sockaddr_in m_addr;
    int type;
};
```



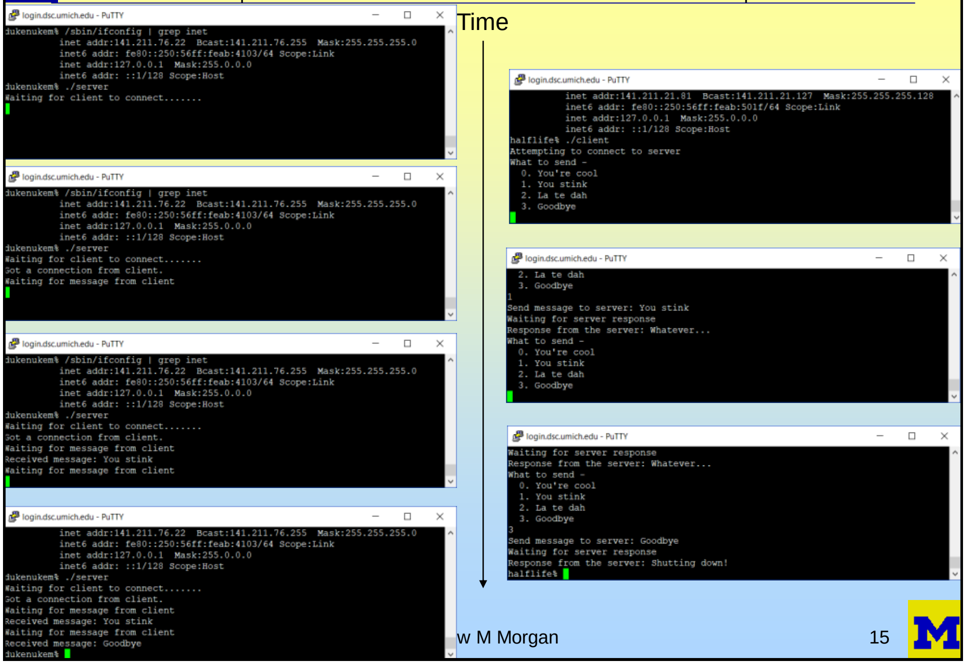
Class Implementations

- The class implementations are not especially pretty…
- Again, the details of how the low-level stuff works is not really within the scope of this class
- I'll provide the implementations on the course site, but no need to include them all in this set of slides…

Execution of Example

Server Computer

Client Computer



```
dukenukem% /sbin/ifconfig | grep inet
          inet addr:141.211.76.22  Bcast:141.211.76.255  Mask:255.255.255.0
          inet6 addr: fe80::250:56ff:feab:4103/64 Scope:Link
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
dukenukem% ./server
Waiting for client to connect.......
```

```
dukenukem% /sbin/ifconfig | grep inet
          inet addr:141.211.76.22  Bcast:141.211.76.255  Mask:255.255.255.0
          inet6 addr: fe80::250:56ff:feab:4103/64 Scope:Link
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
dukenukem% ./server
Waiting for client to connect.......
Got a connection from client.
Waiting for message from client
```

```
dukenukem% /sbin/ifconfig | grep inet
          inet addr:141.211.76.22  Bcast:141.211.76.255  Mask:255.255.255.0
          inet6 addr: fe80::250:56ff:feab:4103/64 Scope:Link
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
dukenukem% ./server
Waiting for client to connect.......
Got a connection from client.
Waiting for message from client
Received message: You stink
Waiting for message from client
```

```
          inet addr:141.211.76.22  Bcast:141.211.76.255  Mask:255.255.255.0
          inet6 addr: fe80::250:56ff:feab:4103/64 Scope:Link
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
dukenukem% ./server
Waiting for client to connect.......
Got a connection from client.
Waiting for message from client
Received message: You stink
Waiting for message from client
Received message: Goodbye
dukenukem% 
```

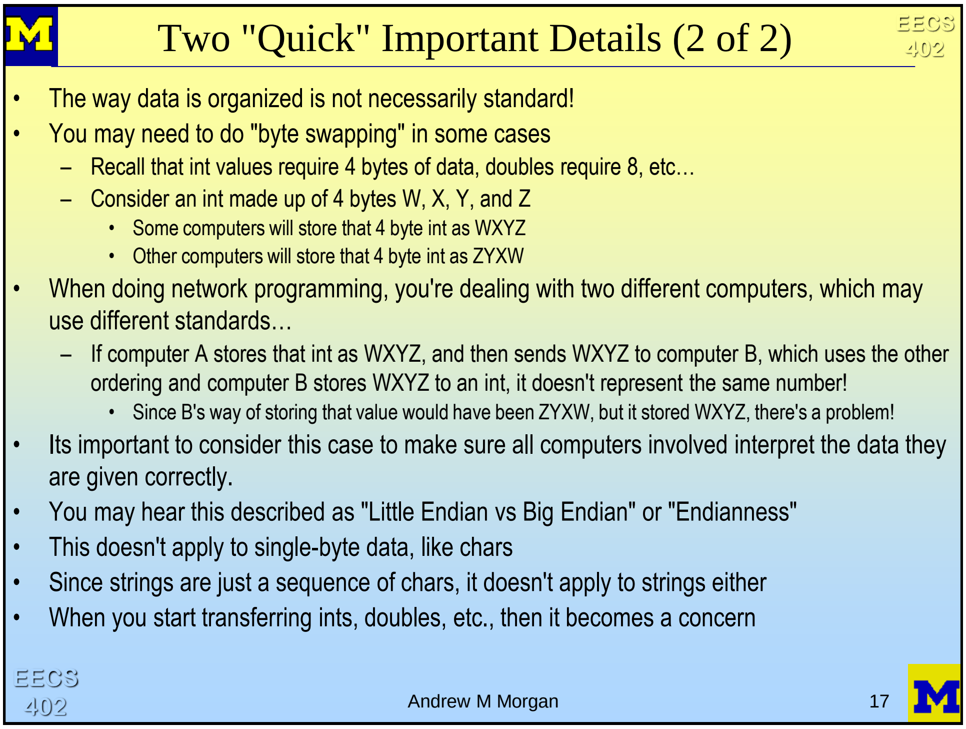
```
          inet addr:141.211.21.81  Bcast:141.211.21.127  Mask:255.255.255.128
          inet6 addr: fe80::250:56ff:feab:501f/64 Scope:Link
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
halflife% ./client
Attempting to connect to server
What to send -
  0. You're cool
  1. You stink
  2. La te dah
  3. Goodbye
```

```
  2. La te dah
  3. Goodbye
1
Send message to server: You stink
Waiting for server response
Response from the server: Whatever...
What to send -
  0. You're cool
  1. You stink
  2. La te dah
  3. Goodbye
```

```
Waiting for server response
Response from the server: Whatever...
What to send -
  0. You're cool
  1. You stink
  2. La te dah
  3. Goodbye
3
Send message to server: Goodbye
Waiting for server response
Response from the server: Shutting down!
halflife% 
```

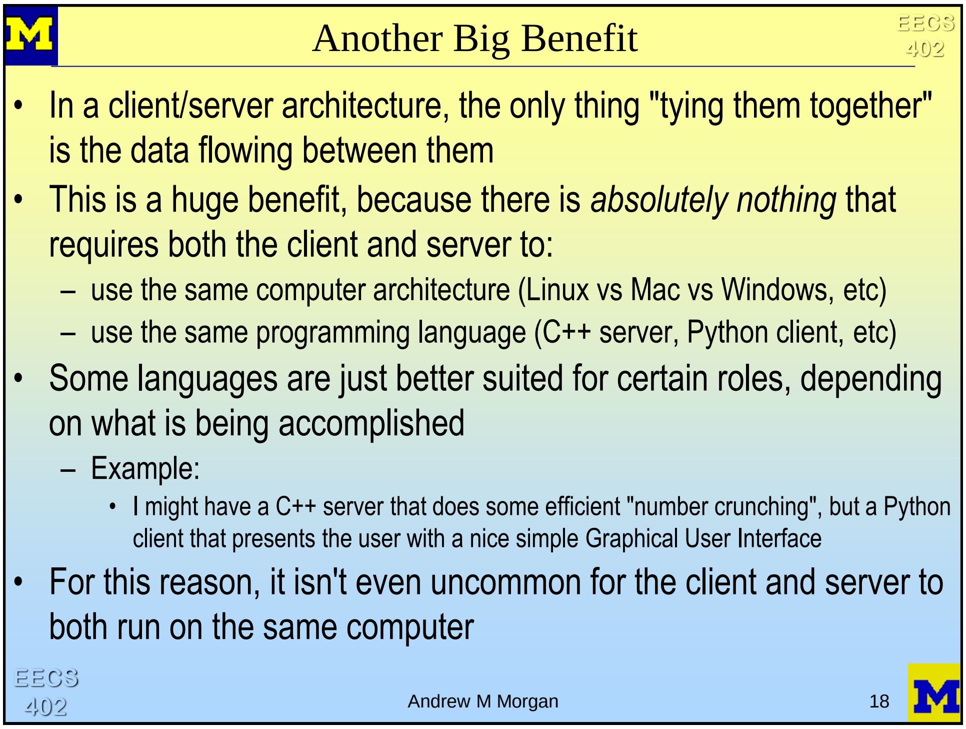
Two "Quick" Important Details (1 of 2)

- There are two commonly used comms protocols used
  - TCP: Transmission Control Protocol
    - Performs some internal "handshaking" and can ensure that data is received, and is received in the exact order it was sent
    - If unsure, use TCP!
    - Can be slower than UDP, but comes with guarantees you don't get with UDP
  - UDP: User Datagram Protocol
    - Does not guarantee data is received, or is received in correct order
    - One computer "broadcasts" data out, and one or more other computers may be listening for that data
    - Originating computer does NOT "re-broadcast" or attempt to ensure the intended recipient receives it
    - Often used in high-bandwidth streaming applications, where if the receiving computer can't keep up with the data or otherwise loses some of it, its no big deal



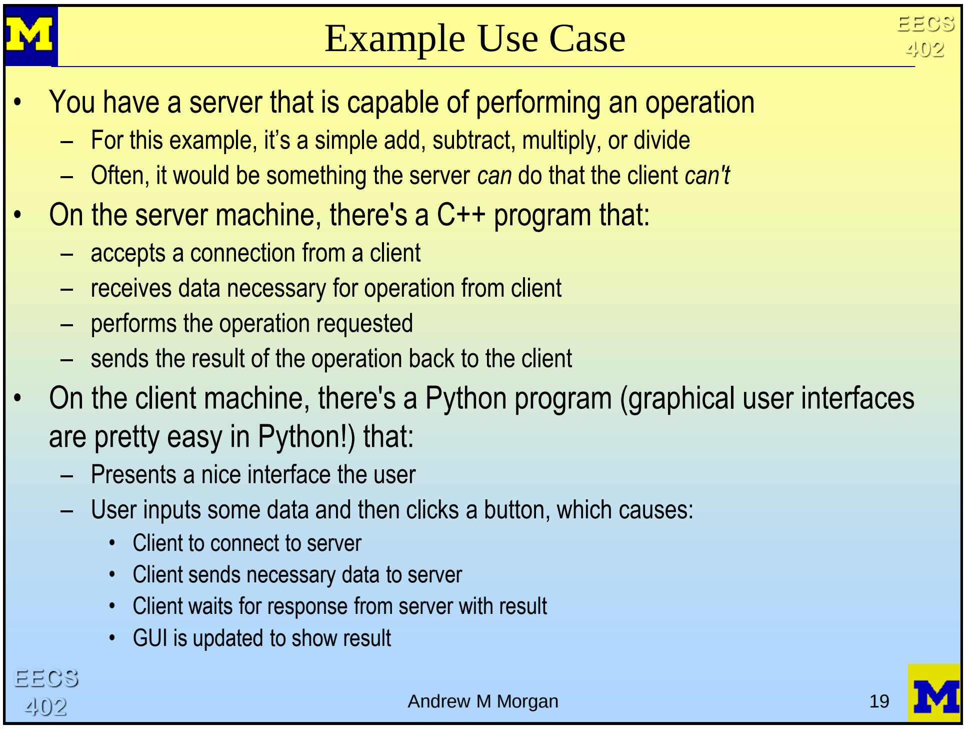
Two "Quick" Important Details (2 of 2)

- The way data is organized is not necessarily standard!
- You may need to do "byte swapping" in some cases
  - Recall that int values require 4 bytes of data, doubles require 8, etc…
  - Consider an int made up of 4 bytes W, X, Y, and Z
    - Some computers will store that 4 byte int as WXYZ
    - Other computers will store that 4 byte int as ZYXW
- When doing network programming, you're dealing with two different computers, which may use different standards…
  - If computer A stores that int as WXYZ, and then sends WXYZ to computer B, which uses the other ordering and computer B stores WXYZ to an int, it doesn't represent the same number!
    - Since B's way of storing that value would have been ZYXW, but it stored WXYZ, there's a problem!
- Its important to consider this case to make sure all computers involved interpret the data they are given correctly.
- You may hear this described as "Little Endian vs Big Endian" or "Endianness"
- This doesn't apply to single-byte data, like chars
- Since strings are just a sequence of chars, it doesn't apply to strings either
- When you start transferring ints, doubles, etc., then it becomes a concern



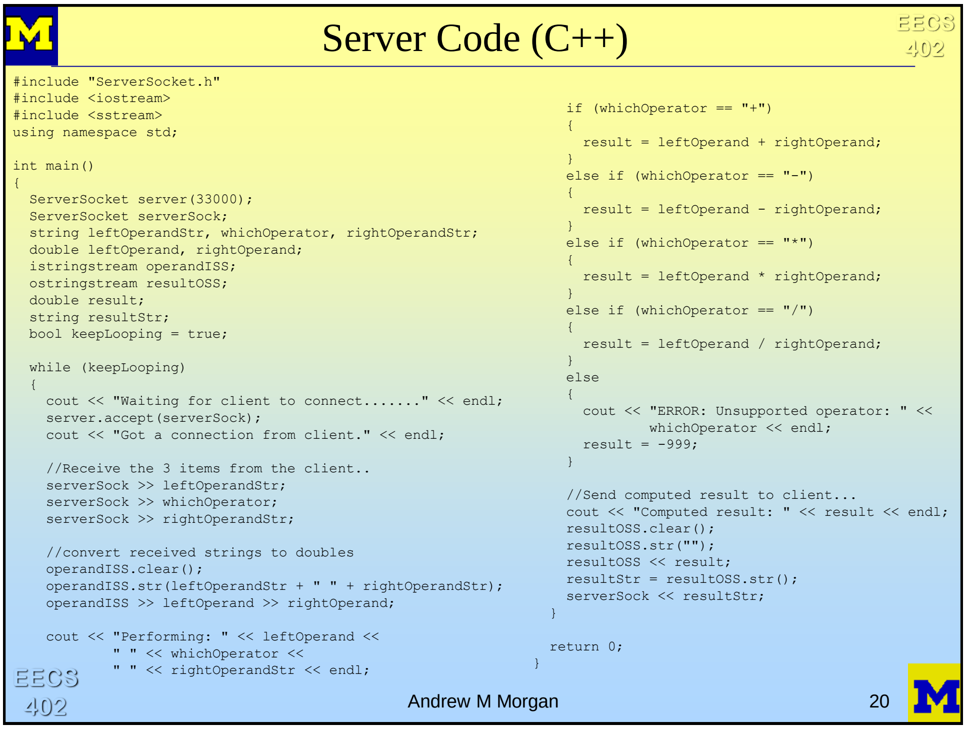
Another Big Benefit

- In a client/server architecture, the only thing "tying them together" is the data flowing between them
- This is a huge benefit, because there is *absolutely nothing* that requires both the client and server to:
  - use the same computer architecture (Linux vs Mac vs Windows, etc)
  - use the same programming language (C++ server, Python client, etc)
- Some languages are just better suited for certain roles, depending on what is being accomplished
  - Example:
    - I might have a C++ server that does some efficient "number crunching", but a Python client that presents the user with a nice simple Graphical User Interface
- For this reason, it isn't even uncommon for the client and server to both run on the same computer



Example Use Case

- You have a server that is capable of performing an operation
  - For this example, it's a simple add, subtract, multiply, or divide
  - Often, it would be something the server *can* do that the client *can't*
- On the server machine, there's a C++ program that:
  - accepts a connection from a client
  - receives data necessary for operation from client
  - performs the operation requested
  - sends the result of the operation back to the client
- On the client machine, there's a Python program (graphical user interfaces are pretty easy in Python!) that:
  - Presents a nice interface the user
  - User inputs some data and then clicks a button, which causes:
    - Client to connect to server
    - Client sends necessary data to server
    - Client waits for response from server with result
    - GUI is updated to show result



Server Code (C++)

```
#include "ServerSocket.h"
#include <iostream>
#include <sstream>
using namespace std;

int main()
{
  ServerSocket server(33000);
  ServerSocket serverSock;
  string leftOperandStr, whichOperator, rightOperandStr;
  double leftOperand, rightOperand;
  istringstream operandISS;
  ostringstream resultOSS;
  double result;
  string resultStr;
  bool keepLooping = true;

  while (keepLooping)
  {
    cout << "Waiting for client to connect......." << endl;
    server.accept(serverSock);
    cout << "Got a connection from client." << endl;

    //Receive the 3 items from the client..
    serverSock >> leftOperandStr;
    serverSock >> whichOperator;
    serverSock >> rightOperandStr;

    //convert received strings to doubles
    operandISS.clear();
    operandISS.str(leftOperandStr + " " + rightOperandStr);
    operandISS >> leftOperand >> rightOperand;

    cout << "Performing: " << leftOperand <<
            " " << whichOperator <<
            " " << rightOperandStr << endl;

    if (whichOperator == "+")
    {
      result = leftOperand + rightOperand;
    }
    else if (whichOperator == "-")
    {
      result = leftOperand - rightOperand;
    }
    else if (whichOperator == "*")
    {
      result = leftOperand * rightOperand;
    }
    else if (whichOperator == "/")
    {
      result = leftOperand / rightOperand;
    }
    else
    {
      cout << "ERROR: Unsupported operator: " <<
              whichOperator << endl;
      result = -999;
    }

    //Send computed result to client...
    cout << "Computed result: " << result << endl;
    resultOSS.clear();
    resultOSS.str("");
    resultOSS << result;
    resultStr = resultOSS.str();
    serverSock << resultStr;
  }

  return 0;
}
```

Client Code (Python)

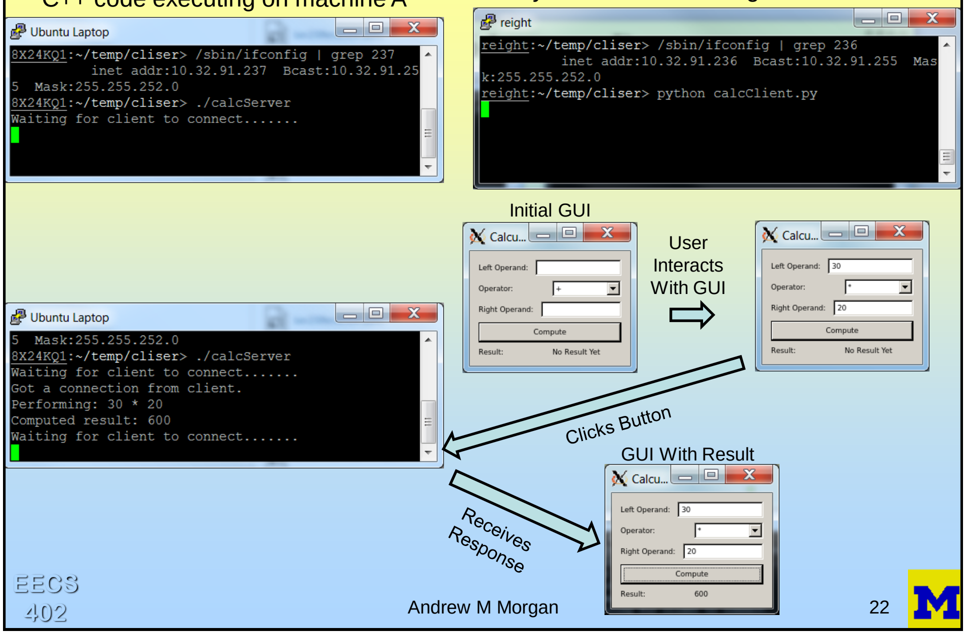
```
from PyQt4 import QtCore
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import socket

class CalculatorWidget(QDialog):
  def __init__(self):
    super(CalculatorWidget, self).__init__(None)

    fullLayout = QVBoxLayout()

    leftOperandBox = QHBoxLayout()
    leftOperandBox.addWidget(QLabel("Left Operand: "))
    self.leftOperandField = QLineEdit()
    leftOperandBox.addWidget(self.leftOperandField)

    operatorBox = QHBoxLayout()
    operatorBox.addWidget(QLabel("Operator: "))
    self.operatorCombo = QComboBox()
    self.operatorCombo.addItem("+")
    self.operatorCombo.addItem("-")
    self.operatorCombo.addItem("*")
    self.operatorCombo.addItem("/")
    operatorBox.addWidget(self.operatorCombo)

    rightOperandBox = QHBoxLayout()
    rightOperandBox.addWidget(QLabel("Right Operand: "))
    self.rightOperandField = QLineEdit()
    rightOperandBox.addWidget(self.rightOperandField)

    resultBox = QHBoxLayout()
    resultBox.addWidget(QLabel("Result: "))
    self.resultField = QLabel("No Result Yet")
    resultBox.addWidget(self.resultField)

    //function continued next column
```

```
    fullLayout.addLayout(leftOperandBox)
    submitButton = QPushButton("Compute")
    submitButton.clicked.connect(self.getResult)

    fullLayout.addLayout(operatorBox)
    fullLayout.addLayout(rightOperandBox)
    fullLayout.addWidget(submitButton)
    fullLayout.addLayout(resultBox)

    self.setLayout(fullLayout)

  def getResult(self):
    cliSock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    cliSock.connect(("10.32.91.237", 33000))

    #Send request info
    cliSock.send(str(self.leftOperandField.text()) + chr(0))
    cliSock.send(str(self.operatorCombo.currentText()) + chr(0))
    cliSock.send(str(self.rightOperandField.text()) + chr(0))

    #Receive resposne string
    responseStr = ""
    responseChar = cliSock.recv(1)
    while (ord(responseChar[0]) != 0):
      responseStr += responseChar
      responseChar = cliSock.recv(1)

    #Update GUI to show response
    self.resultField.setText(responseStr)

if __name__ == '__main__':
  qapp = QApplication(["Calculator Client"])
  guiWidget = CalculatorWidget()
  guiWidget.show()
  qapp.exec_()
```

Multi-Language Demo

C++ code executing on machine A

Python code executing on machine B



Ubuntu Laptop

```
8X24KQ1:~/temp/cliser> /sbin/ifconfig | grep 237
          inet addr:10.32.91.237  Bcast:10.32.91.25
5  Mask:255.255.252.0
8X24KQ1:~/temp/cliser> ./calcServer
Waiting for client to connect.......
```

reight

```
reight:~/temp/cliser> /sbin/ifconfig | grep 236
          inet addr:10.32.91.236  Bcast:10.32.91.255  Mas
k:255.255.252.0
reight:~/temp/cliser> python calcClient.py
```

Initial GUI

Left Operand:
Operator: +
Right Operand:
Result: No Result Yet

User Interacts With GUI

Left Operand: 30
Operator: *
Right Operand: 20
Result: No Result Yet

Clicks Button

Ubuntu Laptop

```
5  Mask:255.255.252.0
8X24KQ1:~/temp/cliser> ./calcServer
Waiting for client to connect.......
Got a connection from client.
Performing: 30 * 20
Computed result: 600
Waiting for client to connect.......
```

Receives Response

GUI With Result

Left Operand: 30
Operator: *
Right Operand: 20
Result: 600