SER 321 A Session

SI Session

Tuesday, February 4th 2025

11:00 am - 12:00 pm MST

Agenda

- Sockets Review
  - Steps & Handling the Client
  - Port Examination
- Serialization
- Threading the System

SI Session Expectations

Thanks for coming to the **SER 321** SI session. We have a packed agenda and we are going to try to get through as many of our planned example problems as possible. This session will be recorded and shared with others.

- If after this you want to see additional examples, please visit the drop-in tutoring center.
- We will post the link in the chat now and at the end of the session.
  - tutoring.asu.edu
- Please keep in mind we are recording this session and it will be made available for you to review 24-48 hours after this session concludes.
- Finally, please be respectful to each other during the session.

Interact with us:

Zoom Features

Zoom Chat

- Use the chat feature to interact with the presenter and respond to presenter's questions.
- Annotations are encouraged

SER 321
Client Socket

Put the Steps for the **Client Socket** in the correct order:

1.
2.
3.
4.
5.
6.
7.
8.

A. Send Message
B. Close Socket
C. Define Params
D. Create Param Struct
E. Receive Message
F. Create Socket
G. Repeat
H. Establish Connection

SER 321
Client Socket

Put the Steps for the **Client Socket** in the correct order:

C	1. Define Params
F	2. Create Socket
D	3. **C ONLY** Create a struct for the address
H	4. Establish Connection
A	5. Send Message
E	6. Receive Message
G	7. Repeat #5 and #6 as needed
B	8. Close Socket

A. Send Message
B. Close Socket
C. Define Params
D. Create Param Struct
E. Receive Message
F. Create Socket
G. Repeat
H. Establish Connection

SER 321

Server Socket

Put the Steps for the **Server Socket** in the correct order:

1.

2.

3.

4.

5.

6.

7.

8.

9.

A. Mark Socket to Listen

B. Close Socket

C. Define Params

D. Create Param Struct

E. Continue Listening

F. Handle Client

G. Wait for Connection

H. Bind Socket to Address

I. Create Socket

SER 321

Server Socket

Put the Steps for the **Server Socket** in the correct order:

F	1. Define Params
I	2. Create Socket
E	3. **C ONLY** Create a struct for the address
H	4. Bind Socket to Local Address
A	5. Mark Socket to Listen for Connections
C	6. Wait for Connection
B	7. Handle Client Connection
G	8. Close Client Connection
D	9. Continue Listening for Connections

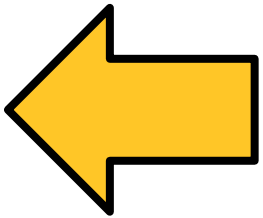
A. Mark Socket to Listen
B. Handle Client
C. Wait for Connection
D. Continue Listening
E. Create Param Struct
F. Define Params
G. Close Socket
H. Bind Socket to Address
I. Create Socket

Assign 3-1 Starter Code

SER 321
Server Socket

What needs to be done here?

1. Define Params
2. Create Socket
3-5. Mark Socket to Listen
6. Wait for Connection
7. Handle Client Connection
8. Close Client Connection
9. Continue Listening



1

2

3

4

5

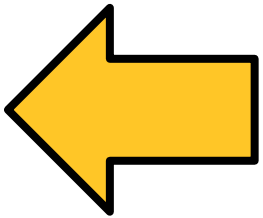
Assign 3-1 Starter Code

SER 321
Server Socket

What needs to be done here?

```
sock = serv.accept(); // blocking wait
System.out.println("Client connected");
```

1. Define Params
2. Create Socket
3-5. Mark Socket to Listen
6. Wait for Connection
7. Handle Client Connection
8. Close Client Connection
9. Continue Listening



1

2

3

4

5

Assign 3-1 Starter Code

Echo_Java

SER 321

Server Socket

What needs to be done here?

Is input
from the client
or
to the client ?

1. Define Params

```
// setup the object reading channel
in = new ObjectInputStream(sock.getInputStream());

// get output channel
OutputStream out = sock.getOutputStream();

// create an object output writer (Java only)
os = new DataOutputStream(out);
```

1

2

3

4

5

```
clientSock = sock.accept(); // blocking wait
PrintWriter out = new PrintWriter(clientSock.getOutputStream(), autoFlush: true);
InputStream input = clientSock.getInputStream();
System.out.println("Server connected to client");
```

Assign 3-1 Starter Code

SER 321
Server Socket

What needs to be done here?

```
static void overandout() {
  try {
    os.close();
    in.close();
    sock.close();
  } catch(Exception e) {e.printStackTrace();}

}
```

```
try {
  s = (String) in.readObject();
} catch (Exception e) {
  System.out.println("Client disconnect");
  connected = false;
  continue;
}
```

1. Create input/output streams
2.
3.
4.
5.

Assign 3-1 Starter Code

SER 321
Server Socket

What needs to be done here?

```
JSONObject res = isValid(s);

if (res.has( key: "ok")) {
  writeOut(res);
  continue;
}

JSONObject req = new JSONObject(s);

res = testField(req, key: "type");
if (!res.getBoolean( key: "ok")) {
  res = noType(req);
  writeOut(res);
  continue;
}
```

```
public static JSONObject isValid(String json) {
  try {
static JSONObject testField(JSONObject req, String key){
  JSONObject res = new JSONObject();

  // field does not exist
  if (!req.has(key)){
    res.put("ok", false);
    res.put("message", "Field " + key + " does not exist in request");
    return res;
  }
  return res.put("ok", true);
}
      return res;
    }
  }
  return new JSONObject();
}
```

Assign 3-1 Starter Code

Echo_Java

SER 321

Server Socket

What needs to be done here?

```
int numr = input.read(clientInput,  off: 0, bufLen);
String received = new String(clientInput,  offset: 0, numr);
System.out.println("read from client: " + received);
out.println(received);
```

```
if (req.getStri
  res = echo(r
} else if (req                                    {
  res = add(re
} else if (req.getString( key: "type").equals("addmany")) {
  res = addmany(req);
} else {
  res = wrongType(req);
}
writeOut(res);
```

Just grabbed the input and printed to the console

1 Create input/output streams

2 Check for disconnect

3 Check Protocol

4

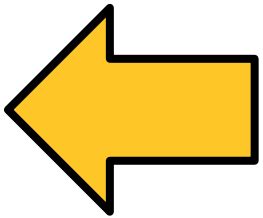
5

Assign 3-1 Starter Code

SER 321
Server Socket

What needs to be done here?

1. Define Params
2. Create Socket
3-5. Mark Socket to Listen
6. Wait for Connection
7. Handle Client Connection
8. Close Client Connection
9. Continue Listening



1 Create input/output streams

2 Check for disconnect

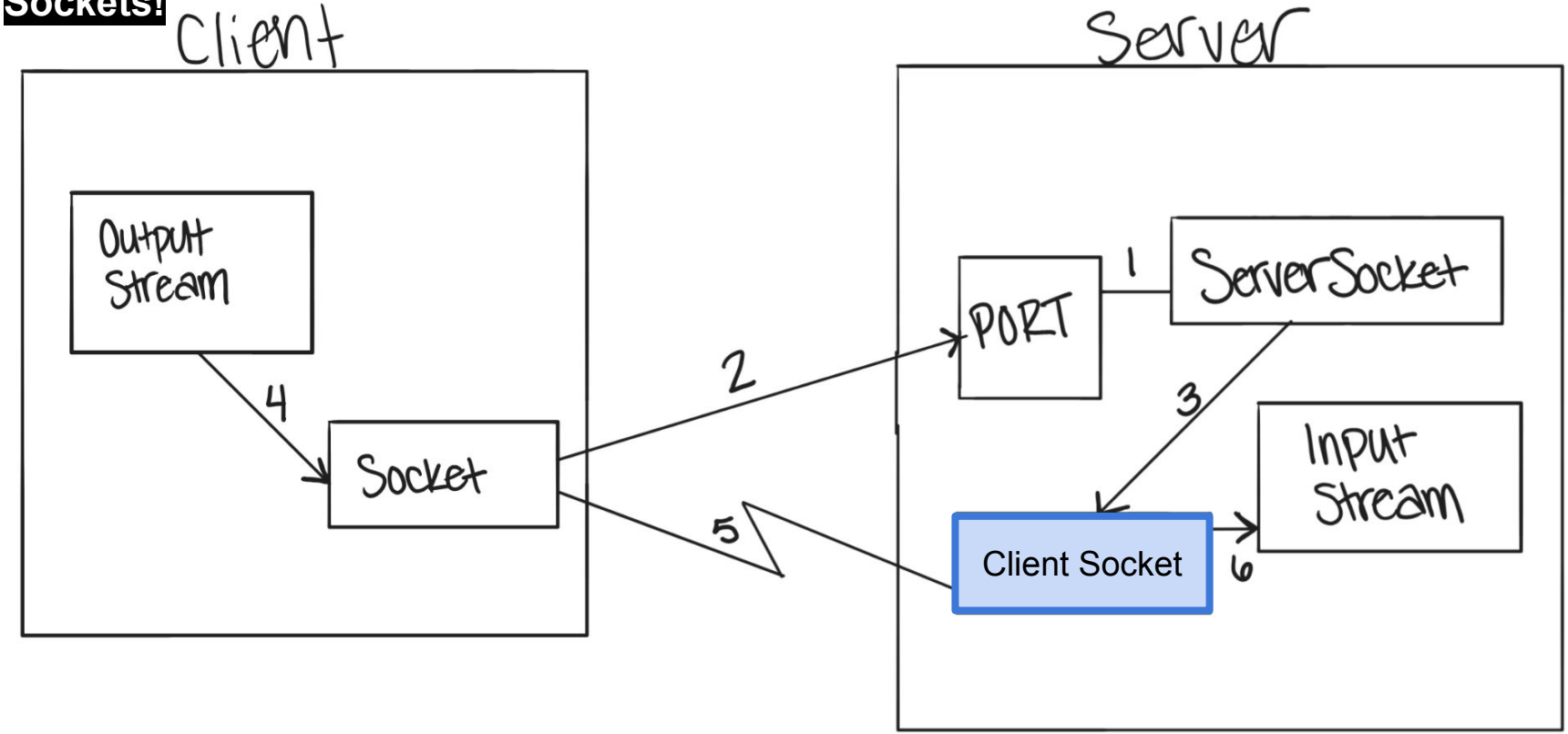
3 Check Protocol

4 Read Headers

5 Handle Accordingly

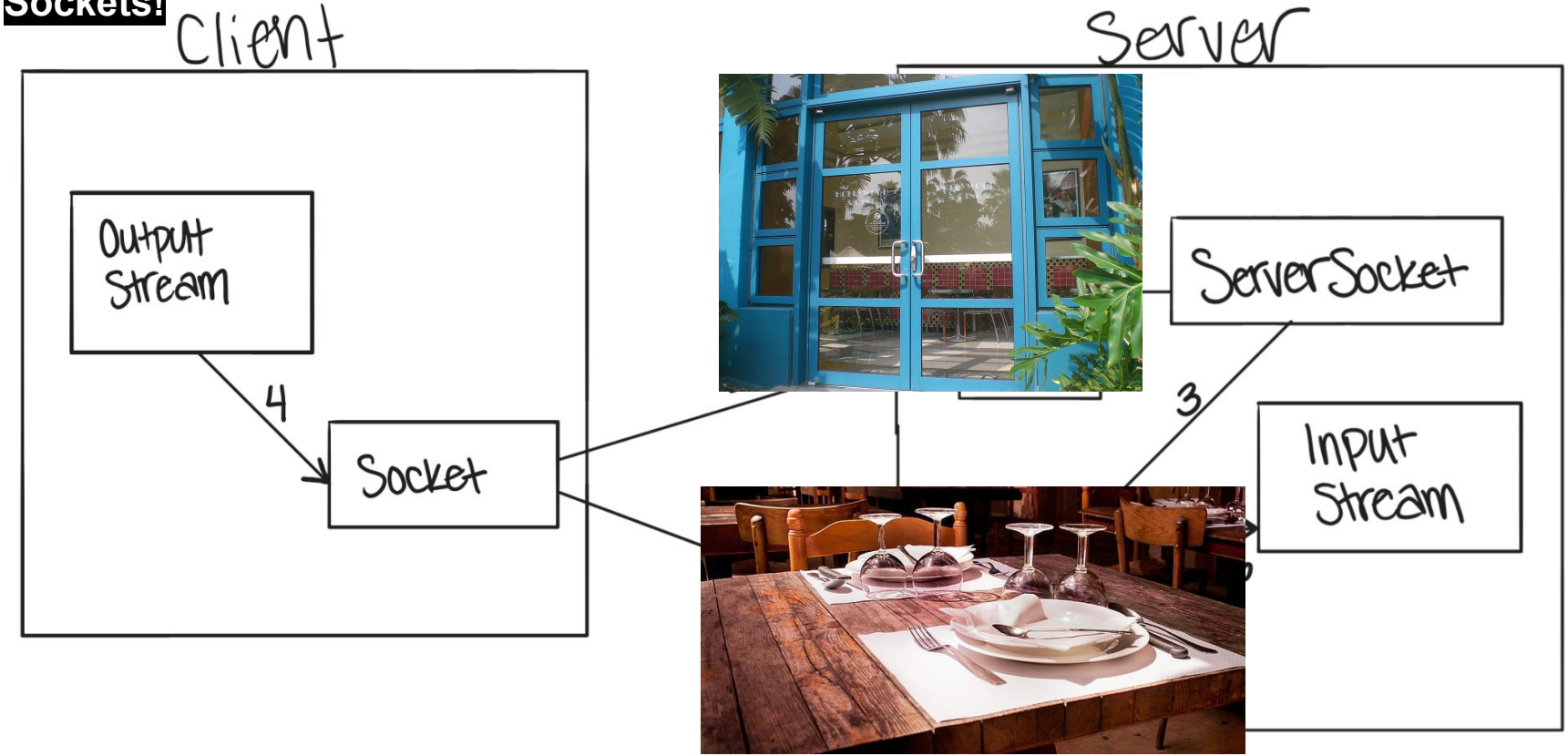
SER 321

Sockets!



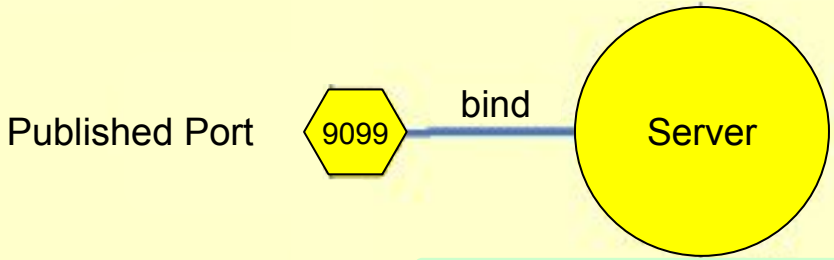
SER 321

Sockets!



SER 321
Sockets!

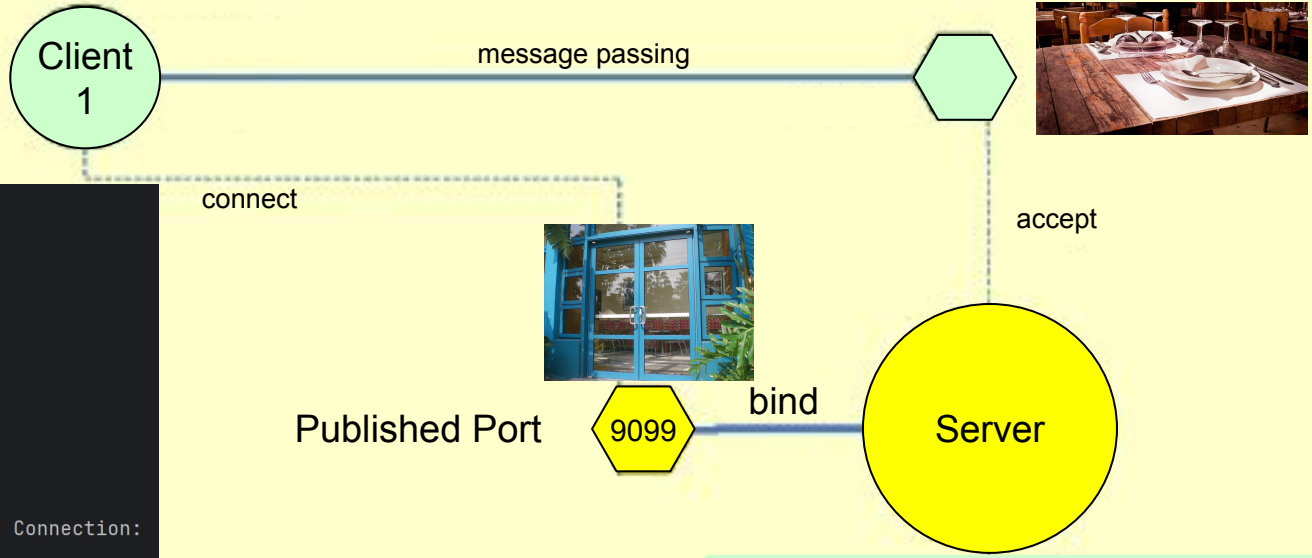
```
> Task :runServer
Server ready for connections
Server is listening on port: 9099
-----
Values of the ServerSocket Object:
Inet Address: 0.0.0.0/0.0.0.0
Local Port: 9099
Server waiting for a connection
Server connected to client
-----
Values of the Client Socket Object after Connection:
        Inet Address: /127.0.0.1
        Local Address: /127.0.0.1
        Local Port: 9099
        Allocated Client Socket (Port): 60296
<==========----> 75% EXECUTING [2m 36s]
> :runServer
```




```
> Task :runClient
Connected to server at localhost:9099
Values of the Socket Object for the Server:
        Host: /127.0.0.1
        Port: 9099
        Local Port: 60296
String to send>
<==========----> 75% EXECUTING [2m 18s]s]
> :runClient
```

Design of an RFID Vehicle Authentication System: A Case Study for Al-Nahrain University Campus - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Client-and-Server-Socket-Ports_fig4_282671198

SER 321
Sockets!




```
> Task :runServer
Server ready for connections
Server is listening on port: 9099
-----
Values of the ServerSocket Object:
Inet Address: 0.0.0.0/0.0.0.0
Local Port: 9099
Server waiting for a connection
Server connected to client
-----
Values of the Client Socket Object after Connection:
        Inet Address: /127.0.0.1
        Local Address: /127.0.0.1
        Local Port: 9099
        Allocated Client Socket (Port): 60296
<==========----> 75% EXECUTING [2m 36s]
> :runServer
```

```
> Task :runClient
Connected to server at localhost:9099
Values of the Socket Object for the Server:
        Host: /127.0.0.1
        Port: 9099
        Local Port: 60296
String to send>
<==========----> 75% EXECUTING [2m 18s]s]
> :runClient
```

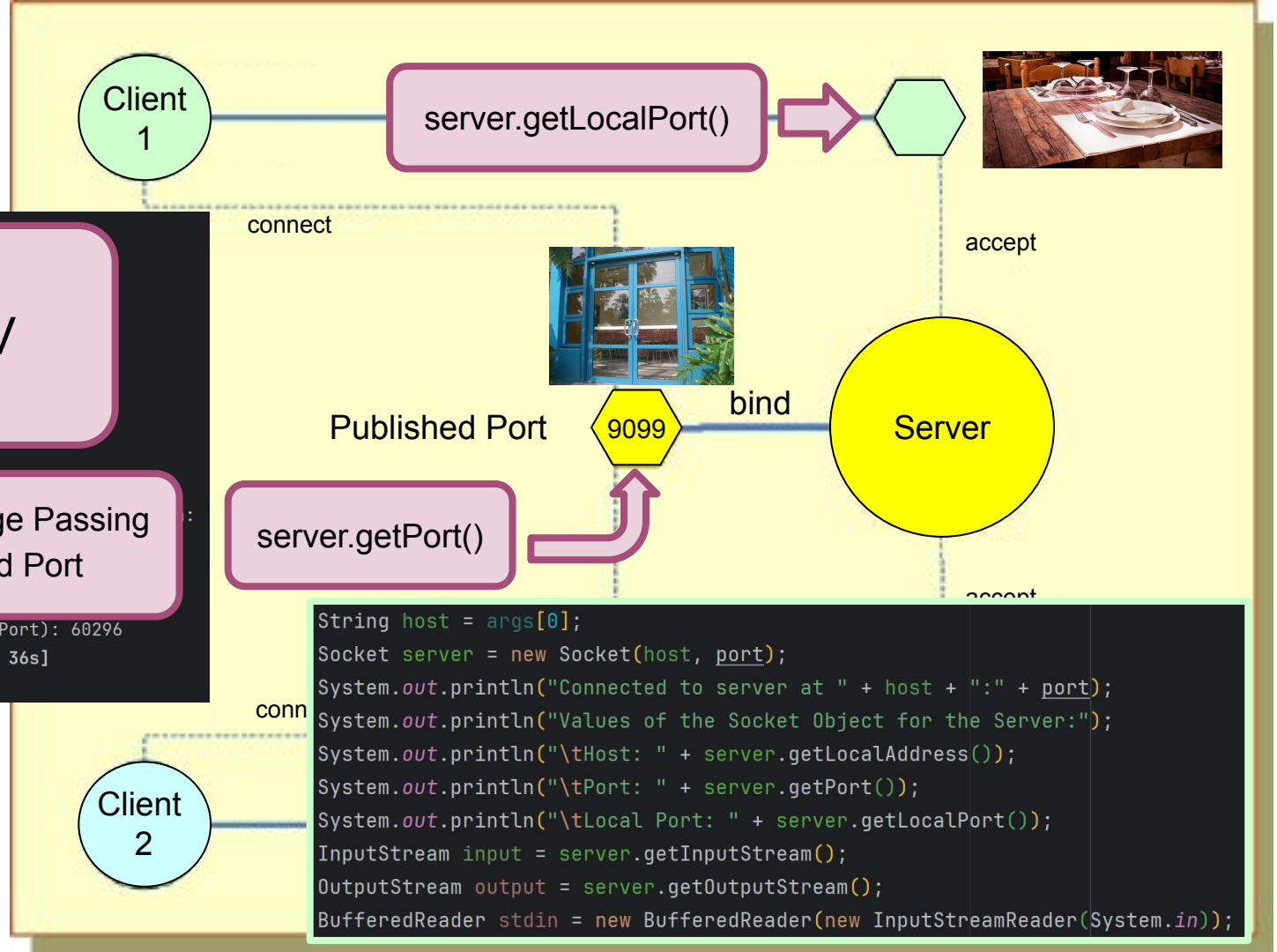
Design of an RFID Vehicle Authentication System: A Case Study for Al-Nahrain University Campus - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Client-and-Server-Socket-Ports_fig4_282671198

SER 321
Sockets!

Client POV

Local Port → Message Passing
Port → Published Port

```
Server connected to client
Allocated Client Socket (Port): 60296
<==========----> 75% EXECUTING [2m 36s]
> :runServer
```




```
String host = args[0];
Socket server = new Socket(host, port);
System.out.println("Connected to server at " + host + ":" + port);
System.out.println("Values of the Socket Object for the Server:");
System.out.println("\tHost: " + server.getLocalAddress());
System.out.println("\tPort: " + server.getPort());
System.out.println("\tLocal Port: " + server.getLocalPort());
InputStream input = server.getInputStream();
OutputStream output = server.getOutputStream();
BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
```

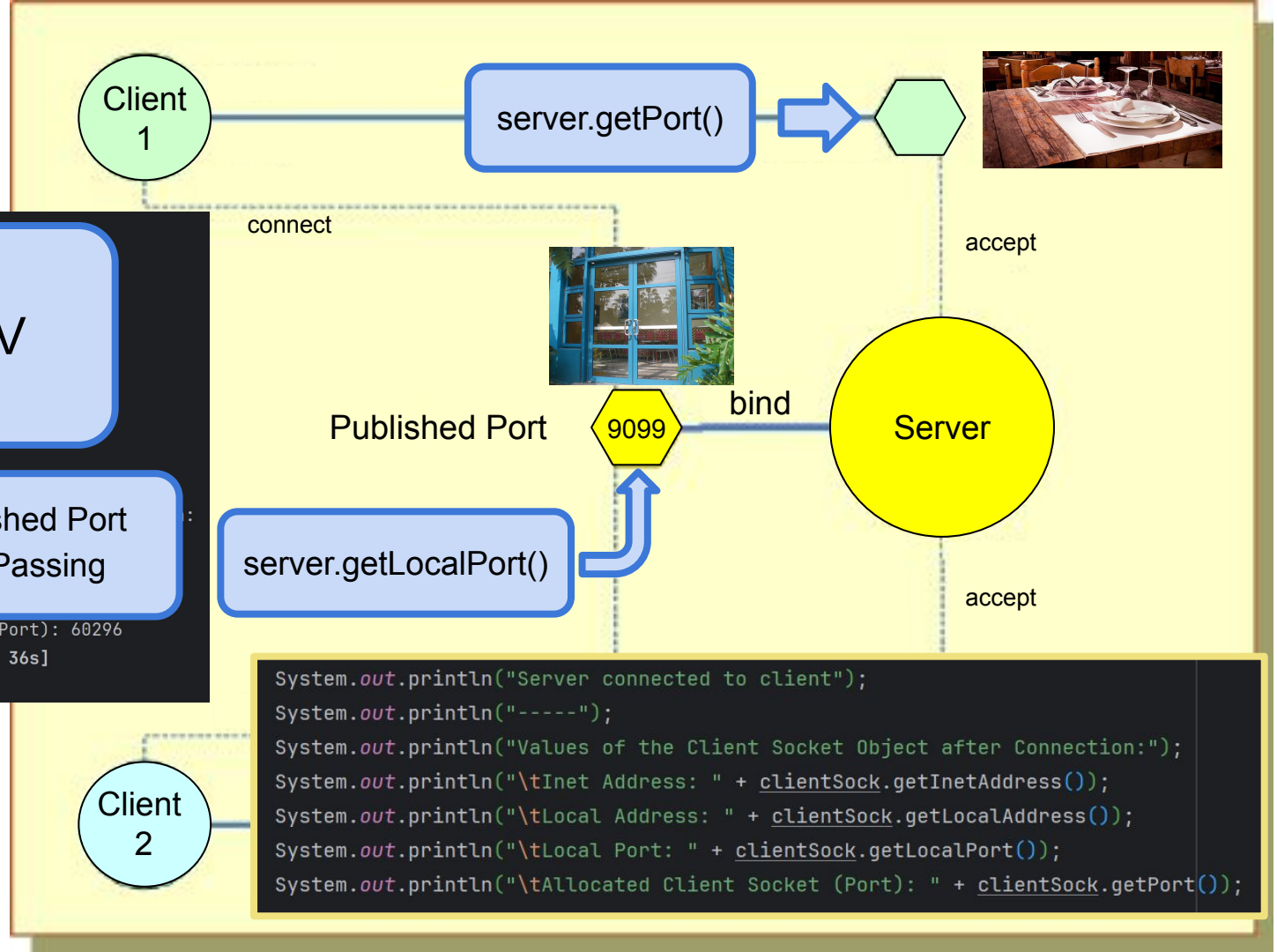
Design of an RFID Vehicle Authentication System: A Case Study for Al-Nahrain University Campus - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Client-and-Server-Socket-Ports_fig4_282671198

SER 321
Sockets!

Server POV

Local Port → Published Port
Port → Message Passing

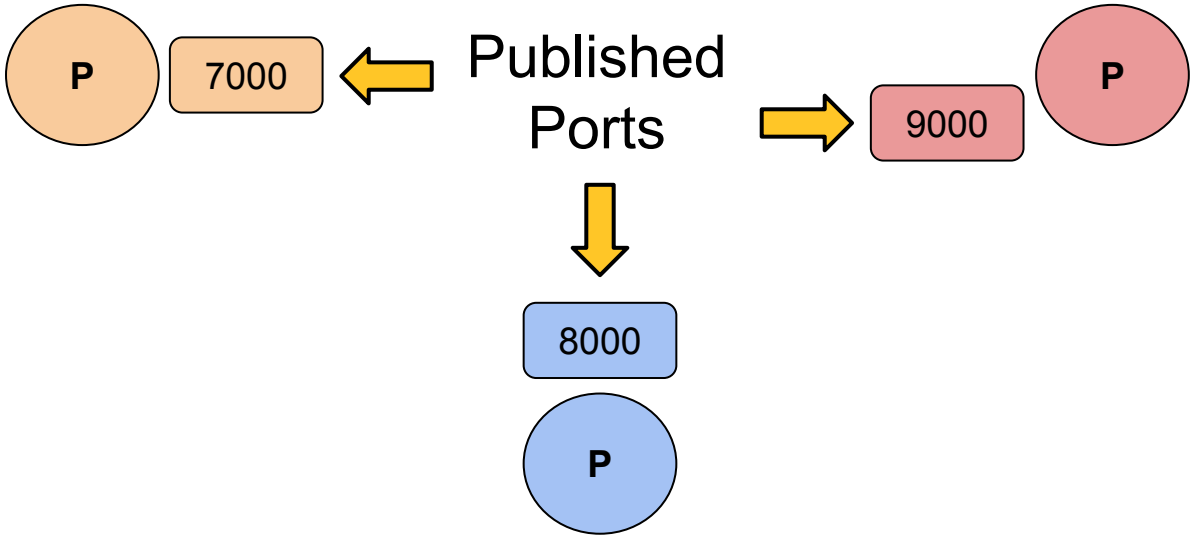
```
Server connected to client
        Allocated Client Socket (Port): 60296
<==========----> 75% EXECUTING [2m 36s]
> :runServer
```




```
System.out.println("Server connected to client");
System.out.println("-----");
System.out.println("Values of the Client Socket Object after Connection:");
System.out.println("\tInet Address: " + clientSock.getInetAddress());
System.out.println("\tLocal Address: " + clientSock.getLocalAddress());
System.out.println("\tLocal Port: " + clientSock.getLocalPort());
System.out.println("\tAllocated Client Socket (Port): " + clientSock.getPort());
```

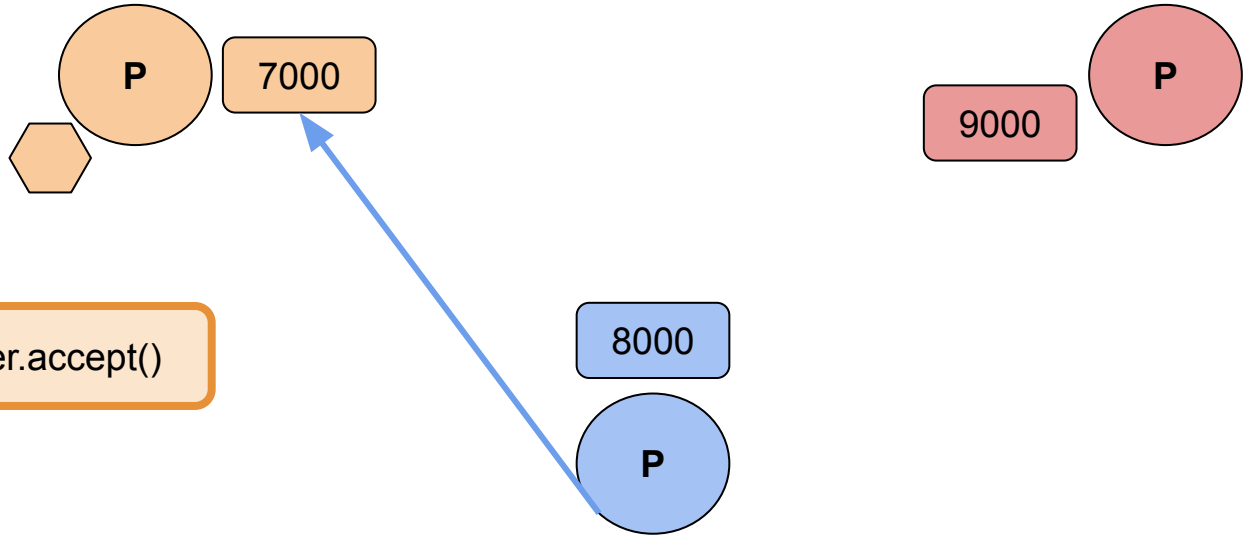
Design of an RFID Vehicle Authentication System: A Case Study for Al-Nahrain University Campus - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Client-and-Server-Socket-Ports_fig4_282671198

Remember that the OS allocates a new port for the client socket!



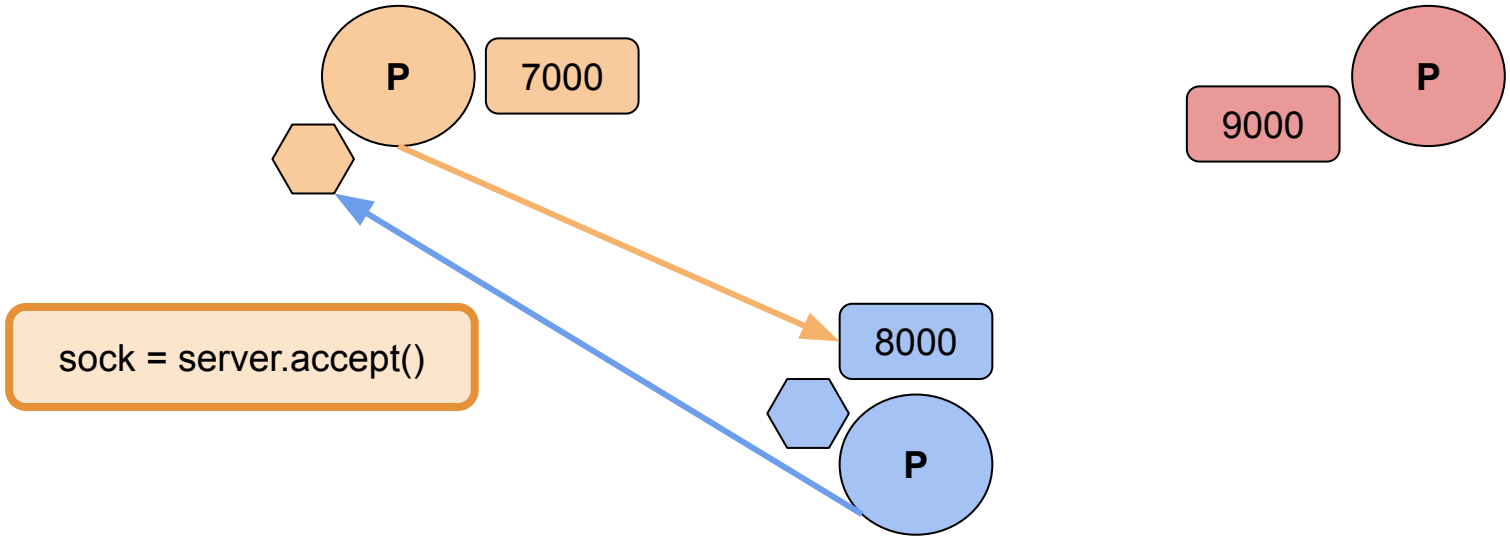
SER 321
Communication

Remember that the OS allocates a new port for the client socket!

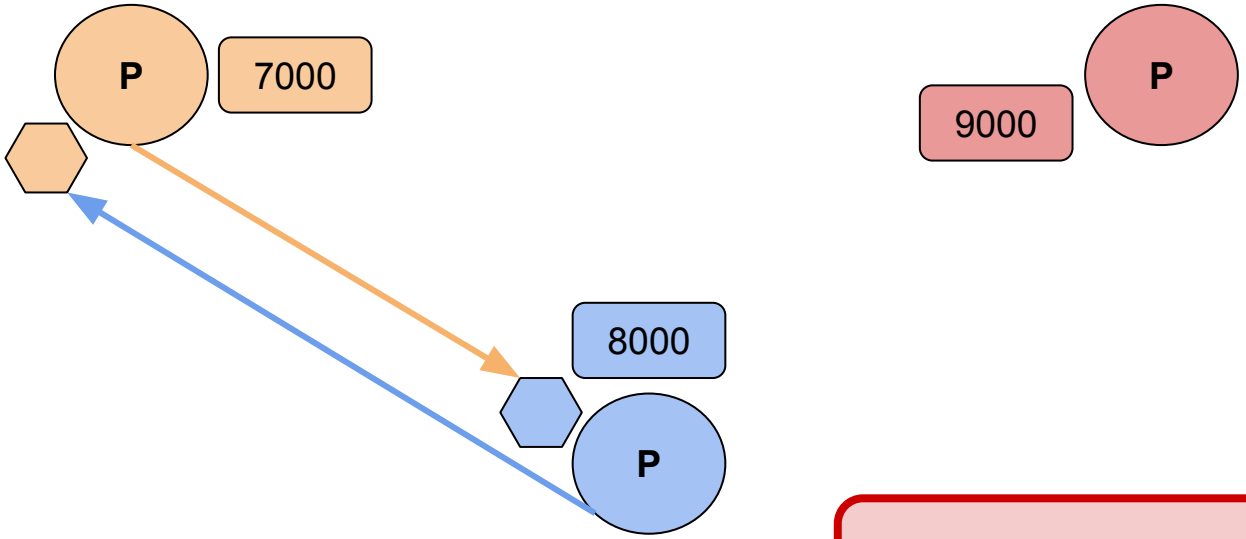


SER 321
Communication

Remember that the OS allocates a new port for the client socket!

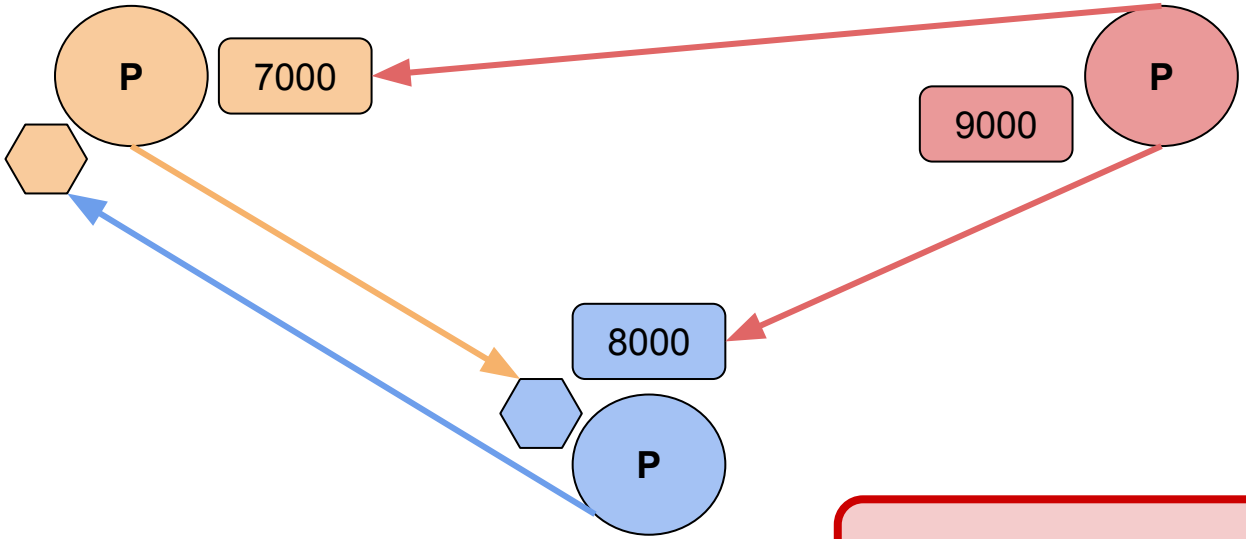


Remember that the OS allocates a new port for the client socket!



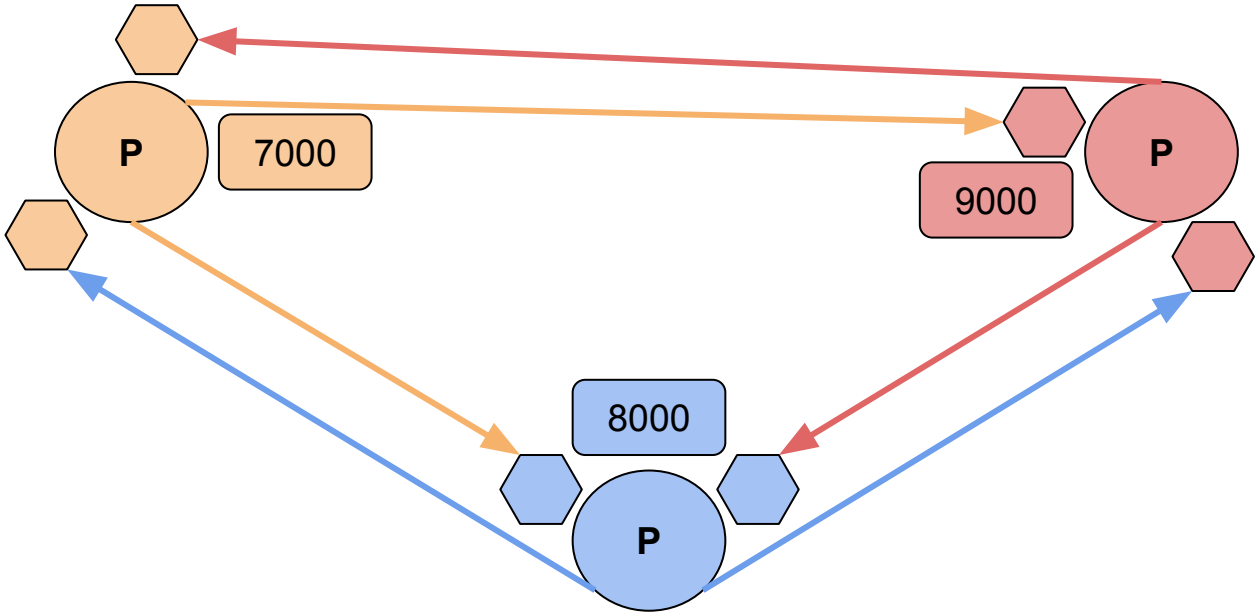
What about Peer 9000?

Remember that the OS allocates a new port for the client socket!



What about Peer 9000?

Remember that the OS allocates a new port for the client socket!



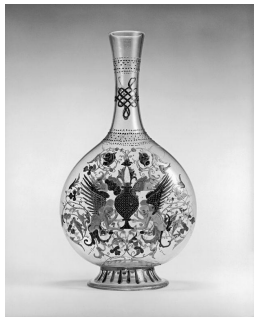
SER 321
Serialization

What is *serialization*?

"Translating data structures or object states for storage or transmission"

SER 321
Serialization

What is *serialization*?



Data

"Translating data structures or object states for storage or transmission"

SER 321
Serialization

What is *serialization*?

Serialized Data

"Translating data structures or object states for storage or transmission"

SER 321
Serialization

Can we recall some of the formats?

JSON

Java Object Serialization

Protocol Buffers

XML

SER 321
Serialization

Binary

Text

Two main approaches for storing the content…

What about the data format?

JSON

Java Object Serialization

Protocol Buffers

XML

SER 321
Serialization

Binary

Text

Who uses *TEXT*?

Text

JSON

Java Object Serialization

Protocol Buffers

Text

XML

SER 321
Serialization

Binary

Text

What does this imply?

Who uses ***BINARY***?

Text JSON

Binary Java Object Serialization

Binary Protocol Buffers

Text XML

SER 321
Serialization

Streams and their types

Generic Superclass

```
OutputStream out = sock.getOutputStream();
```

Buffered Stream

Bytes

Data Stream

Primitive DATA Types

Object Stream

Java Objects

SER 321
System Layout

You have two systems…

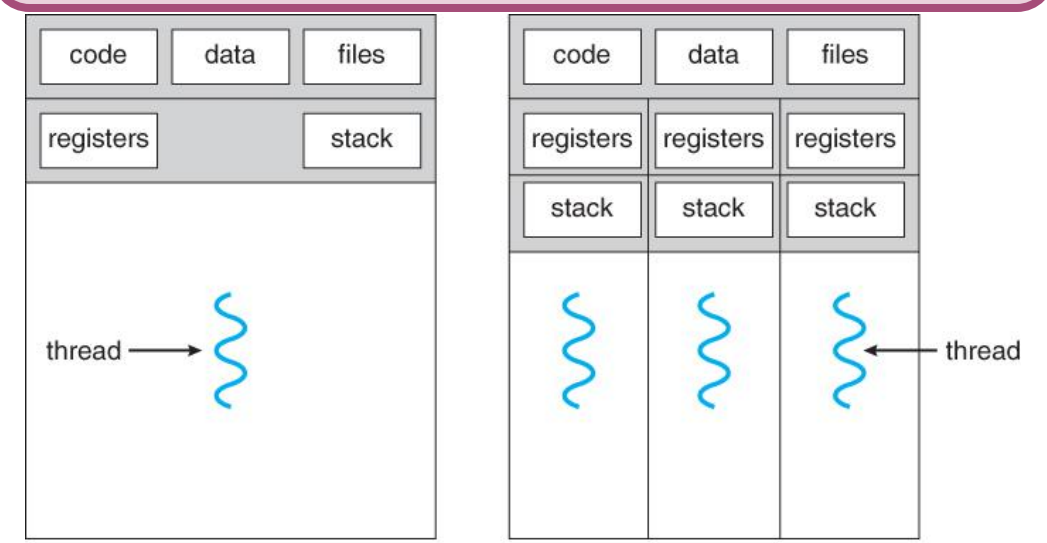
How can we test our server with multiple clients?

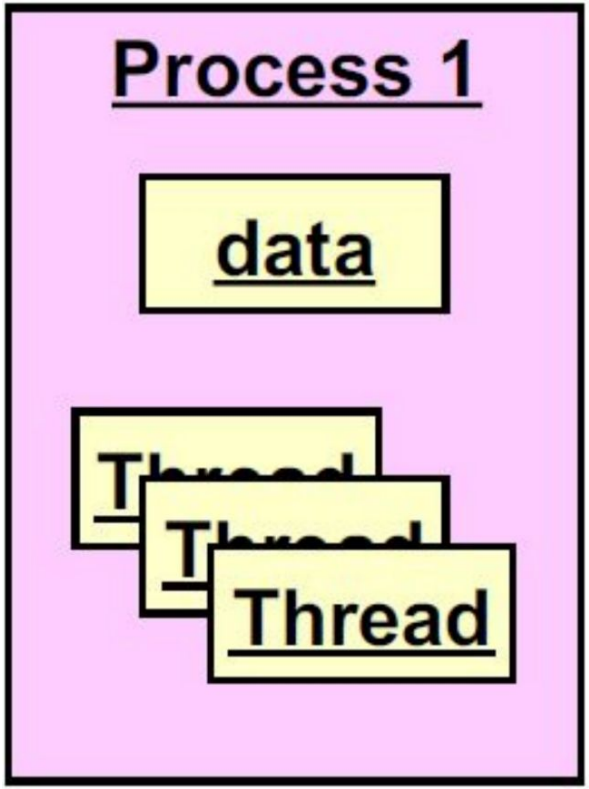
?

SER 321
Threads

What does that imply?

Remember that they exist *within* the parent process





SER 321
Threading Pitfalls

Race Condition	A thread never gains access to the resource it needs
Starvation	A thread is only able to acquire some of the resources it needs
Deadlock	More than one thread accesses a single resource at the same time

SER 321
Threading Pitfalls

Race Condition	A thread never gains access to the resource it needs
Starvation	A thread is only able to acquire some of the resources it needs
Deadlock	More than one thread accesses a single resource at the same time

NetworkDeadlock

SER 321
Threading Pitfalls

As the project name implies, we encounter a ***deadlock***.

But what happened?

Client

```
class SockClient {
    public static void main (String args[]) throws Exception {
        Socket      sock = new Socket( host: "localhost", port: 8888);    //Any IP name

        ObjectInputStream in = new ObjectInputStream(sock.getInputStream());
        ObjectOutputStream out = new ObjectOutputStream(sock.getOutputStream());

        String s = (String) in.readObject();
        out.writeObject("Back at you");

        in.close();
        out.close();
        sock.close();
    }
}
```

Server

```
class SockServer {
    public static void main (String args[]) throws Exception {

        int count = 0;
        ServerSocket    serv = new ServerSocket( port: 8888);

        Socket sock = serv.accept();

        ObjectInputStream in = new ObjectInputStream(sock.getInputStream());
        ObjectOutputStream out = new ObjectOutputStream(sock.getOutputStream());

        String s = (String) in.readObject();
        System.out.println("Received " + s);
        out.writeObject("Back at you");
        System.out.println("Received " + s);

        in.close();
        out.close();
        sock.close();
    }
}
```

```
PS C:\ASU\SER321\examples_repo\ser321examples\Threads\NetworkDeadlock> gradle
server
<=========----> 75% EXECUTING [1m 33s]
> :server
```

```
PS C:\ASU\SER321\examples_repo\ser321examples\Threads\NetworkDeadlock> gradle
client
Starting a Gradle Daemon, 1 busy and 1 stopped Daemons could not be reused, us
e --status for details
<=========----> 75% EXECUTING [53s]
> :client
```

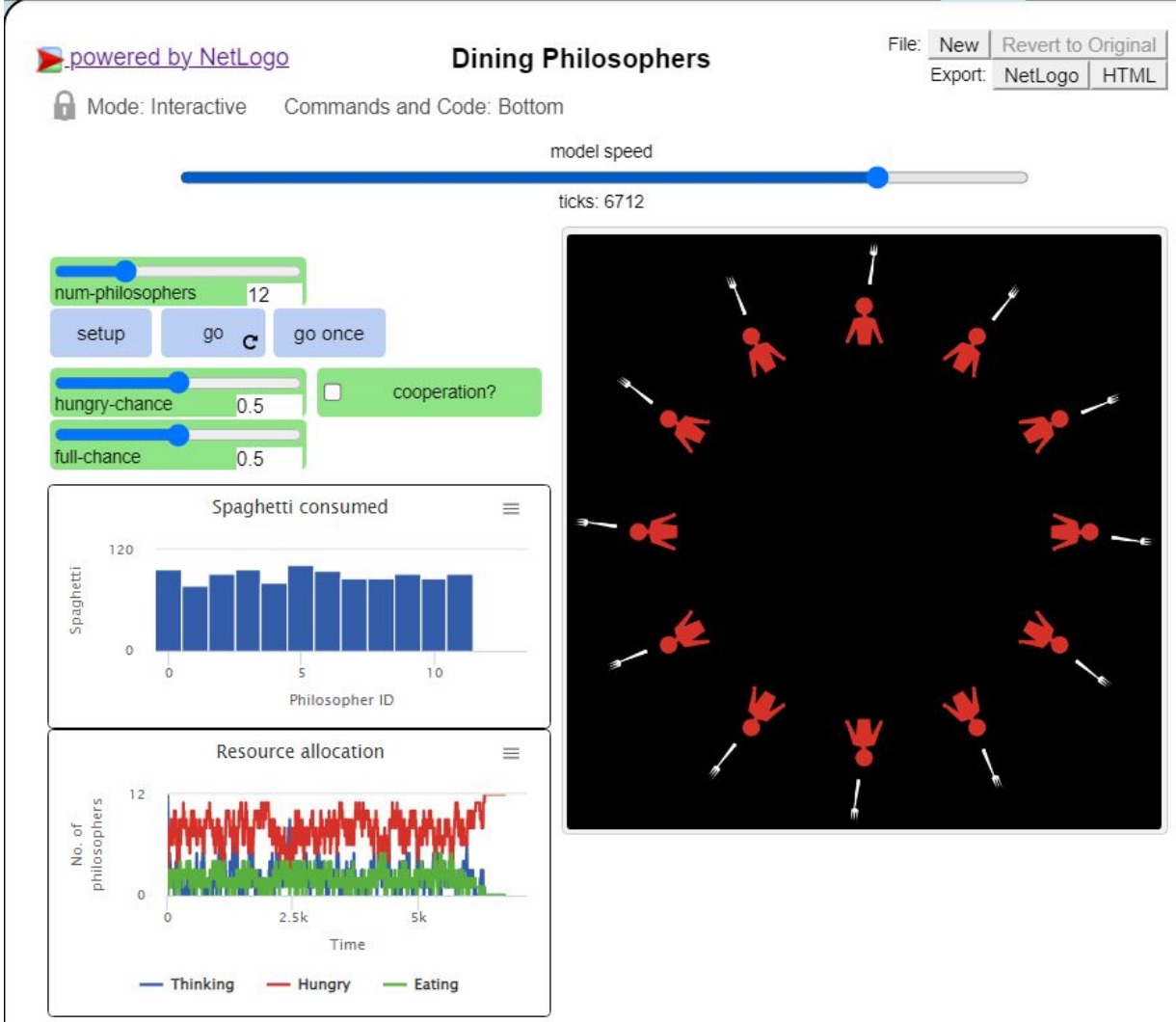
Dining Philosophers

SER 321
Threading Pitfalls

What does *Spaghetti Consumed* represent?

What does *Thinking* represent?

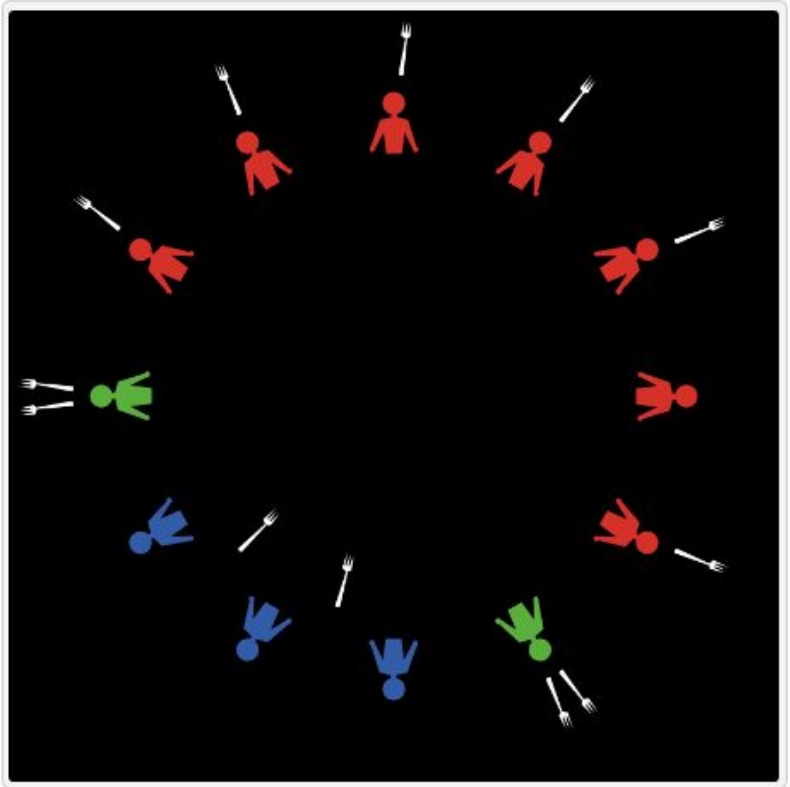
What does *Hungry* represent?



SER 321

Threading Pitfalls

Can we take a guess at what is happening here?



Austin Walter's Traffic Comparison

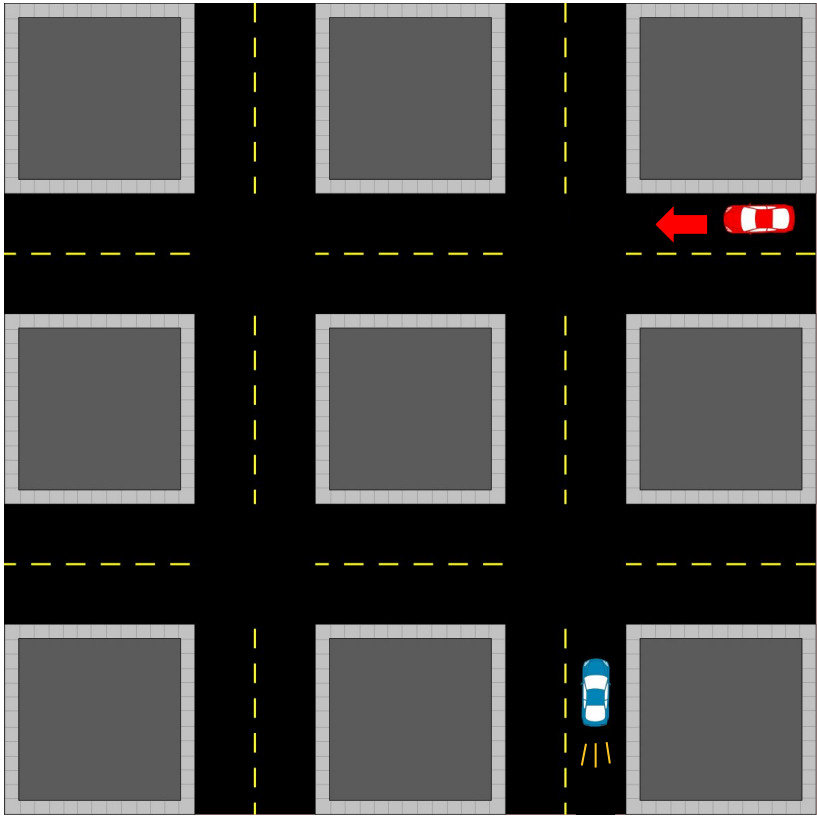
SER 321

Threading Pitfalls

Race Condition

Crash

More than one thread accesses a single resource at once



Austin Walter's Traffic Comparison

SER 321

Threading Pitfalls

Race Condition

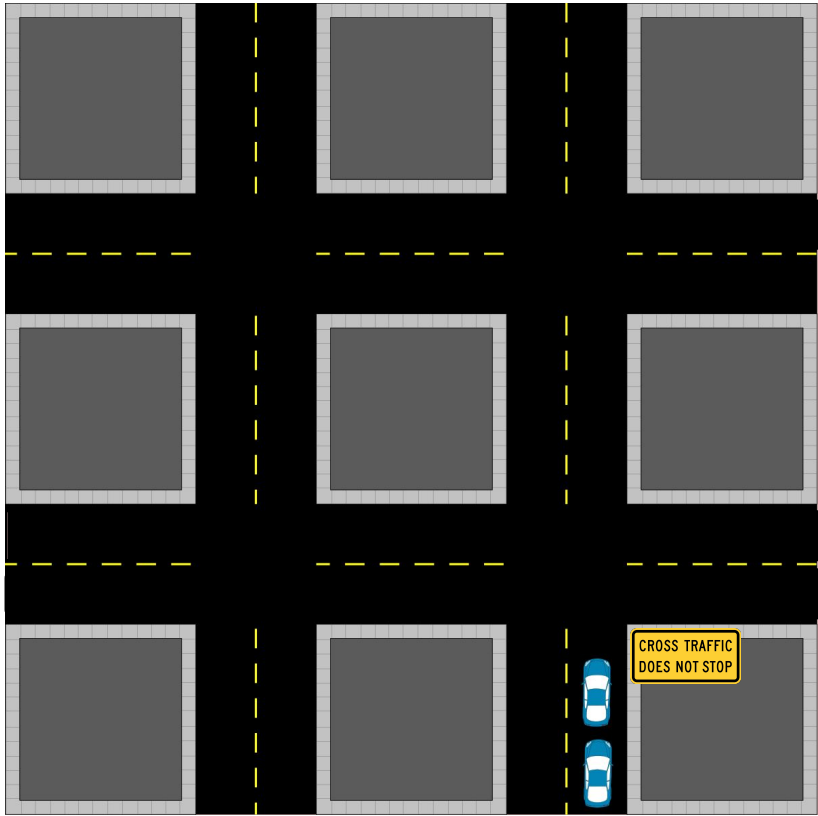
Crash

More than one thread accesses a single resource at once

Starvation

Cross Traffic

A thread never gains access to the resource it needs



Austin Walter's Traffic Comparison

SER 321

Threading Pitfalls

Race Condition — **Crash**

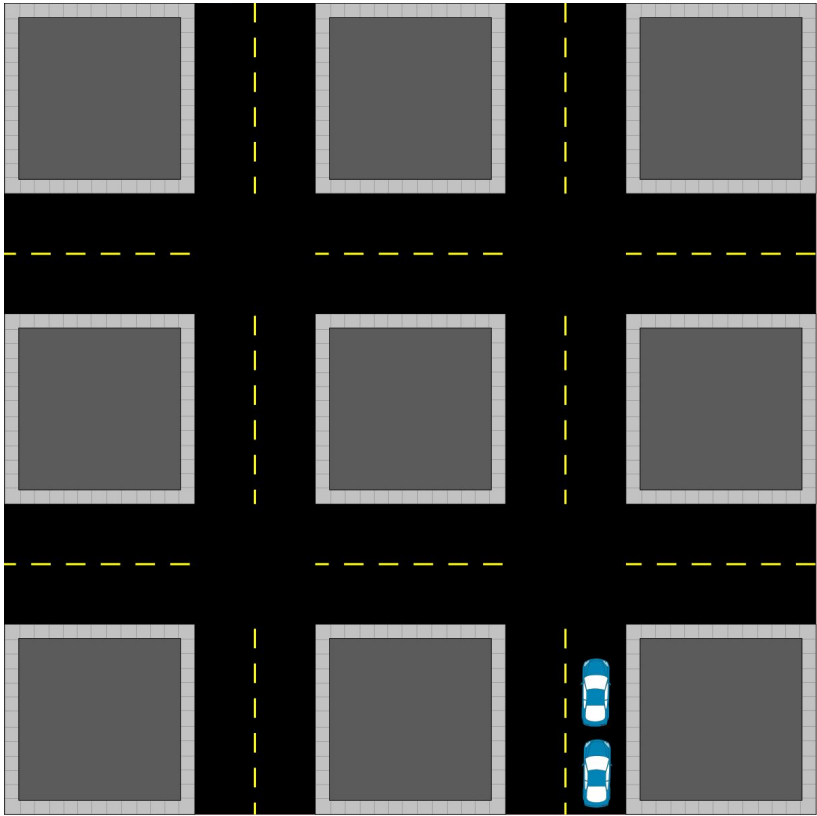
More than one thread accesses a single resource at once

Starvation — **Cross Traffic**

A thread never gains access to the resource it needs

Deadlock — **Gridlock**

A thread is only able to acquire some of the needed resources



SER 321

Scratch Space

Upcoming Events

SI Sessions:

- Thursday, February 6th at 7:00 pm MST
- Sunday, February 9th at 7:00 pm MST
- Tuesday, February 11th at 11:00 am MST

Review Sessions:

- Tuesday, February 25th at 11:00 am MST - **Q&A Session**
- Thursday, February 27th at 7:00 pm MST - **Exam Review Session (2hrs)**

Questions?

Survey:

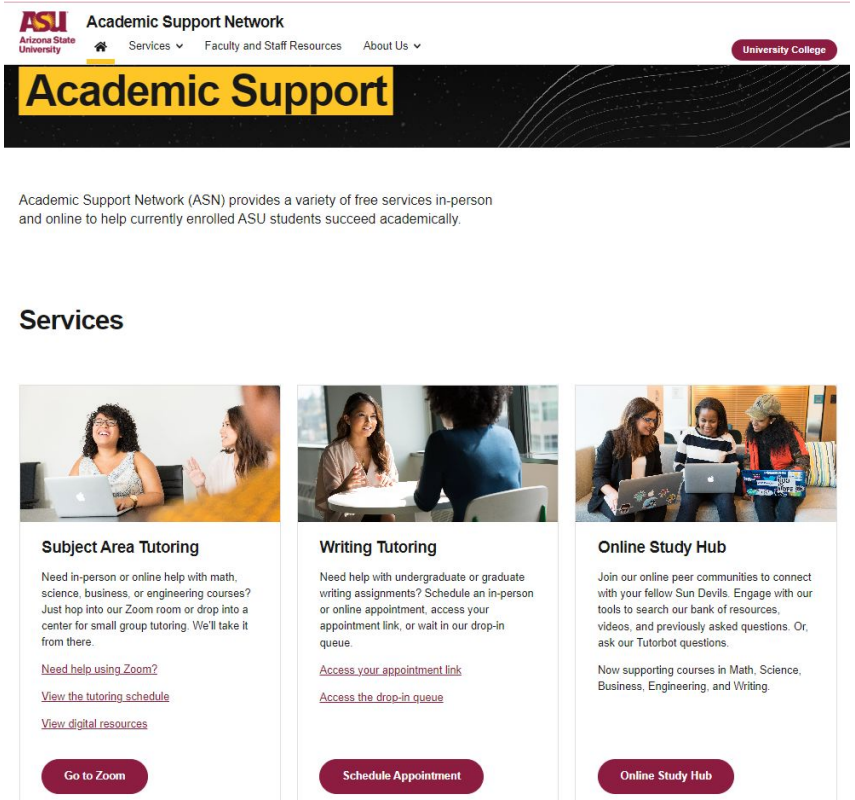
https://asuasn.info/ASNSurvey

48

More Questions?

Check out our other resources!

tutoring.asu.edu

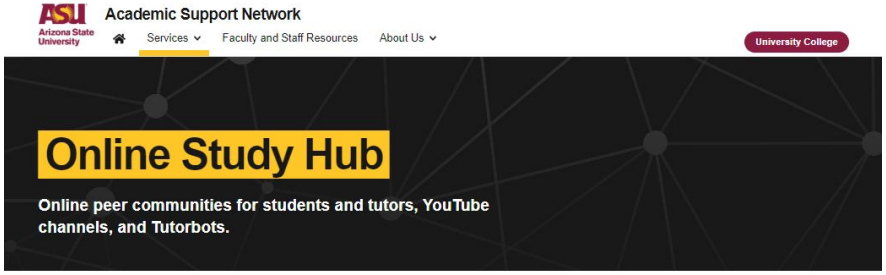


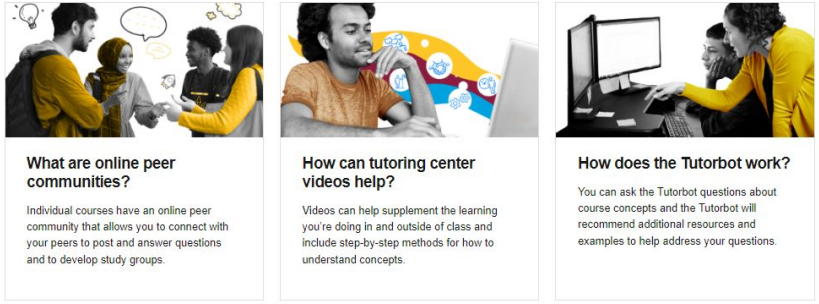
1. **Click on ‘Go to Zoom’ to log onto our Online Tutoring Center.**
2. **Click on ‘View the tutoring schedule’ to see when tutors are available for specific courses.**

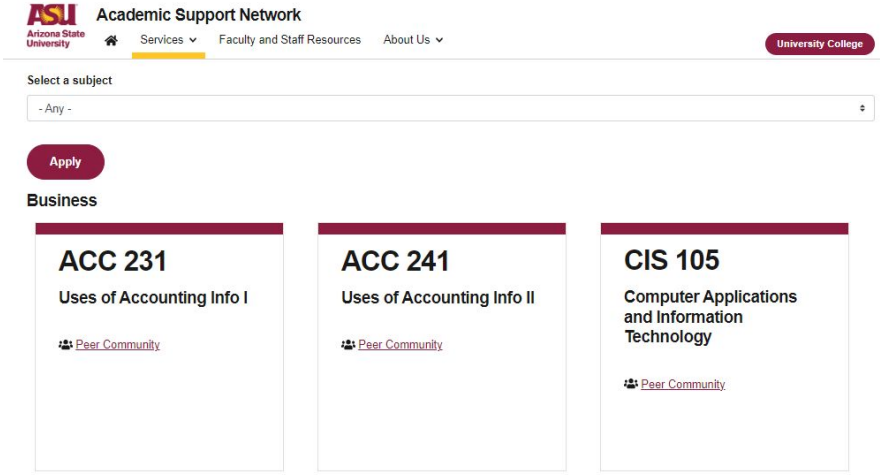
More Questions?

Check out our other resources!

tutoring.asu.edu/online-study-hub



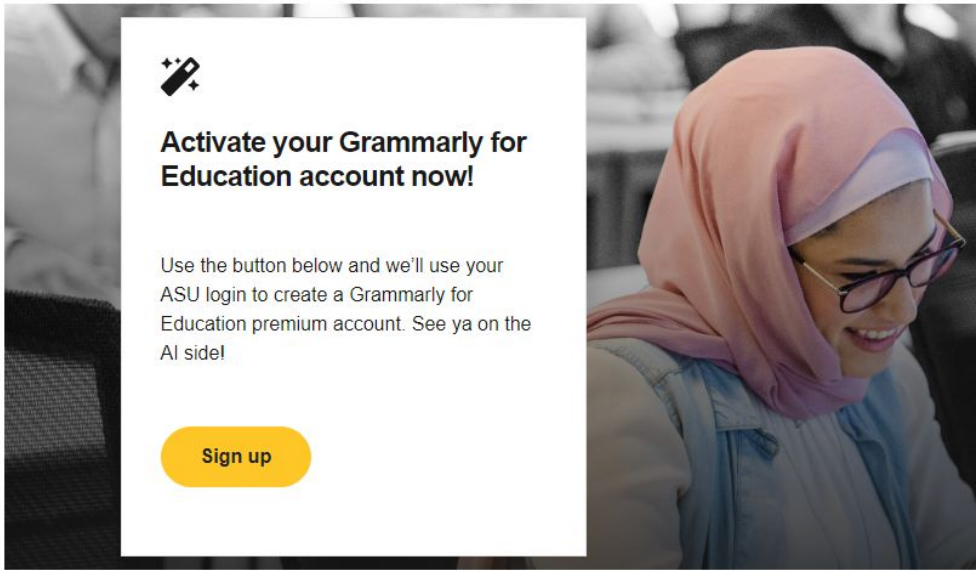




Don't forget to check out the Online Study Hub for additional resources!

Expanded Writing Support Available

Including Grammarly for Education, at no cost!



tutoring.asu.edu/expanded-writing-support

*Available slots for this pilot are limited

Additional Resources

- Course Repo
- Gradle Documentation
- GitHub SSH Help
- Linux Man Pages
- OSI Interactive
- MDN HTTP Docs
  - Requests
  - Responses
- JSON Guide
- org.json Docs
- javax.swing package API
- Swing Tutorials
- Dining Philosophers Interactive
- Austin G Walters Traffic Comparison