

SER 321 C Session

SI Session

Thursday, June 20th 2024

6:00 pm - 7:00 pm MST

Agenda



Gradle Review

Threading Your Server

How-To

Tracing Execution

Distributed Systems

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

Gradle Review

Which of the following will run the main method in `/java/taskone/Server.java` with `gradle runTask1` ?

```
task runServer(type: JavaExec) {
    group 'server'
    description 'Creates Server socket waits for messages'

    classpath = sourceSets.main.runtimeClasspath

    main = 'taskone.Server.runTask1'
    standardInput = System.in

    args 8000;
    if (project.hasProperty('port')) {
        args(project.getProperty('port'));
    }
}
```

A.

```
task runServer(type: JavaExec) {
    group 'server'
    description 'Creates Server socket waits for messages'

    classpath = sourceSets.main.runtimeClasspath

    main = 'taskone.Server'
    standardInput = System.in

    args 8000;
    if (project.hasProperty('port')) {
        args(project.getProperty('port'));
    }
}
```

B.

```
task1 runServer(type: JavaExec) {
    group 'server'
    description 'Creates Server socket waits for messages'

    classpath = sourceSets.main.runtimeClasspath

    main = 'taskone.Server'
    standardInput = System.in

    args 8000;
    if (project.hasProperty('port')) {
        args(project.getProperty('port'));
    }
}
```

C.

```
task runTask1(type: JavaExec) {
    group 'server'
    description 'Creates Server socket waits for messages'

    classpath = sourceSets.main.runtimeClasspath

    main = 'taskone.Server'
    standardInput = System.in

    args 8000;
    if (project.hasProperty('port')) {
        args(project.getProperty('port'));
    }
}
```

D.

SER 321

Gradle Review

Which of the following will run the main method in `/java/taskone/Server.java` with `gradle runTask1` ?

```
task runServer(type: JavaExec) {
    group 'server'
    description 'Creates Server socket waits for messages'

    classpath = sourceSets.main.runtimeClasspath

    main = 'taskone.Server.runTask1'
    standardInput = System.in

    args 8000;
    if (project.hasProperty('port')) {
        args(project.getProperty('port'));
    }
}
```

A.

```
task runServer(type: JavaExec) {
    group 'server'
    description 'Creates Server socket waits for messages'

    classpath = sourceSets.main.runtimeClasspath

    main = 'taskone.Server'
    standardInput = System.in

    args 8000;
    if (project.hasProperty('port')) {
        args(project.getProperty('port'));
    }
}
```

B.

```
task1 runServer(type: JavaExec) {
    group 'server'
    description 'Creates Server socket waits for messages'

    classpath = sourceSets.main.runtimeClasspath

    main = 'taskone.Server'
    standardInput = System.in

    args 8000;
    if (project.hasProperty('port')) {
        args(project.getProperty('port'));
    }
}
```

C.

```
task runTask1(type: JavaExec) {
    group 'server'
    description 'Creates Server socket waits for messages'

    classpath = sourceSets.main.runtimeClasspath

    main = 'taskone.Server'
    standardInput = System.in

    args 8000;
    if (project.hasProperty('port')) {
        args(project.getProperty('port'));
    }
}
```

D.

SER 321

Gradle Review

Which of the following will run the main method in `/java/tasktwo/Server.java` with `gradle runTask2` ?

```
task runTask2(type: JavaExec) {
    group 'server'
    description 'Creates Server socket waits for messages'

    classpath = sourceSets.main.runtimeClasspath

    main = 'taskone.Server'
    standardInput = System.in

    args 8000;
    if (project.hasProperty('port')) {
        args(project.getProperty('port'));
    }
}
```

A.

```
task runTask2(type: JavaExec) {
    group 'server'
    description 'Creates Server socket waits for messages'

    classpath = sourceSets.main.runtimeClasspath

    main = 'tasktwo.Server'
    standardInput = System.in

    args 8000;
    if (project.hasProperty('port')) {
        args(project.getProperty('port'));
    }
}
```

B.

```
task2 runServer(type: JavaExec) {
    group 'server'
    description 'Creates Server socket waits for messages'

    classpath = sourceSets.main.runtimeClasspath

    main = 'tasktwo.Server'
    standardInput = System.in

    args 8000;
    if (project.hasProperty('port')) {
        args(project.getProperty('port'));
    }
}
```

C.

```
task runServer(type: JavaExec) {
    group 'server'
    description 'Creates Server socket waits for messages'

    classpath = sourceSets.main.runtimeClasspath

    main = 'tasktwo.Server'
    standardInput = System.in

    args 8000;
    if (project.hasProperty('port')) {
        args(project.getProperty('port'));
    }
}
```

D.

SER 321

Gradle Review

Which of the following will run the main method in `/java/tasktwo/Server.java` with `gradle runTask2` ?

```
task runTask2(type: JavaExec) {
    group 'server'
    description 'Creates Server socket waits for messages'

    classpath = sourceSets.main.runtimeClasspath

    main = 'taskone.Server'
    standardInput = System.in

    args 8000;
    if (project.hasProperty('port')) {
        args(project.getProperty('port'));
    }
}
```

A.

```
task runTask2(type: JavaExec) {
    group 'server'
    description 'Creates Server socket waits for messages'

    classpath = sourceSets.main.runtimeClasspath

    main = 'tasktwo.Server'
    standardInput = System.in

    args 8000;
    if (project.hasProperty('port')) {
        args(project.getProperty('port'));
    }
}
```

B.

```
task2 runServer(type: JavaExec) {
    group 'server'
    description 'Creates Server socket waits for messages'

    classpath = sourceSets.main.runtimeClasspath

    main = 'tasktwo.Server'
    standardInput = System.in

    args 8000;
    if (project.hasProperty('port')) {
        args(project.getProperty('port'));
    }
}
```

C.

```
task runServer(type: JavaExec) {
    group 'server'
    description 'Creates Server socket waits for messages'

    classpath = sourceSets.main.runtimeClasspath

    main = 'tasktwo.Server'
    standardInput = System.in

    args 8000;
    if (project.hasProperty('port')) {
        args(project.getProperty('port'));
    }
}
```

D.

SER 321

Gradle Review

Which of the following will run the main method in `/java/taskone/Client.java` with `gradle runClient` ?

```
task runClient(type: JavaExec) {
    group 'client'
    description 'Creates client socket sends a message to the server'

    classpath = sourceSets.main.runtimeClasspath
    standardInput = System.in

    main = 'taskone.Client'
    standardInput = System.in

    if (project.hasProperty("host") && project.hasProperty('port')) {
        args(project.getProperty('host'), project.getProperty('port'));
    }
}
```

A.

```
task runClient(type: JavaExec) {
    group 'client'
    description 'Creates client socket sends a message to the server'

    classpath = sourceSets.main.runtimeClasspath
    standardInput = System.in

    main = 'taskone.Client'
    standardInput = System.in

    args("localhost", 8000);
    if (project.hasProperty("host") && project.hasProperty('port')) {
        args(project.getProperty('host'), project.getProperty('port'));
    }
}
```

B.

```
task runClient(type: JavaExec) {
    group 'client'
    description 'Creates client socket sends a message to the server'

    classpath = sourceSets.main.runtimeClasspath
    standardInput = System.in

    main = 'taskone.Client'
    standardInput = System.in

    if (project.hasProperty("host") && project.hasProperty('port')) {
        args(project.getProperty('host'), project.getProperty('port'));
    } else if (project.hasProperty("host")) {
        args(project.getProperty('host'), 8000);
    } else if (project.hasProperty("port")) {
        args("localhost", project.getProperty('port'))
    } else {
        args("localhost", 8000);
    }
}
```

C.

```
task runClient(type: JavaExec) {
    group 'client'
    description 'Creates client socket sends a message to the server'

    classpath = sourceSets.main.runtimeClasspath
    standardInput = System.in

    main = 'taskone.Client'
    standardInput = System.in

    if (project.hasProperty('host') && project.hasProperty('port')) {
        args(project.getProperty('host'), project.getProperty('port'));
    } else if (project.hasProperty('host')) {
        args(project.getProperty('host'), 8000);
    } else if (project.hasProperty('port')) {
        args('localhost', project.getProperty('port'));
    }
}
```

D.

SER 321

Gradle Review

Which of the following will run the main method in `/java/taskone/Client.java` with `gradle runClient` ?

```
task runClient(type: JavaExec) {
    group 'client'
    description 'Creates client socket sends a message to the server'

    classpath = sourceSets.main.runtimeClasspath
    standardInput = System.in

    main = 'taskone.Client'
    standardInput = System.in

    if (project.hasProperty("host") && project.hasProperty('port')) {
        args(project.getProperty('host'), project.getProperty('port'));
    }
}
```

A.

```
task runClient(type: JavaExec) {
    group 'client'
    description 'Creates client socket sends a message to the server'

    classpath = sourceSets.main.runtimeClasspath
    standardInput = System.in

    main = 'taskone.Client'
    standardInput = System.in

    args("localhost", 8000);
    if (project.hasProperty("host") && project.hasProperty('port')) {
        args(project.getProperty('host'), project.getProperty('port'));
    }
}
```

B.

```
task runClient(type: JavaExec) {
    group 'client'
    description 'Creates client socket sends a message to the server'

    classpath = sourceSets.main.runtimeClasspath
    standardInput = System.in

    main = 'taskone.Client'
    standardInput = System.in

    if (project.hasProperty("host") && project.hasProperty('port')) {
        args(project.getProperty('host'), project.getProperty('port'));
    } else if (project.hasProperty("host")) {
        args(project.getProperty('host'), 8000);
    } else if (project.hasProperty("port")) {
        args("localhost", project.getProperty('port'))
    } else {
        args("localhost", 8000);
    }
}
```

C.

```
task runClient(type: JavaExec) {
    group 'client'
    description 'Creates client socket sends a message to the server'

    classpath = sourceSets.main.runtimeClasspath
    standardInput = System.in

    main = 'taskone.Client'
    standardInput = System.in

    if (project.hasProperty('host') && project.hasProperty('port')) {
        args(project.getProperty('host'), project.getProperty('port'));
    } else if (project.hasProperty('host')) {
        args(project.getProperty('host'), 8000);
    } else if (project.hasProperty('port')) {
        args('localhost', project.getProperty('port'));
    }
}
```

D.

SER 321

Threaded Server

Given the standard server socket steps...

In which step do we introduce **threads**?

1. Define Params

2. Create Socket

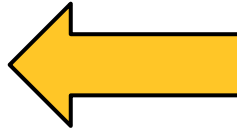
3-5. Mark Socket to Listen

6. Wait for Connection

7. Handle Client Connection

8. Close Client Connection

9. Continue Listening



7. Send Client Socket to thread

SER 321 Threads

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



1

2 & 3-5

9

6

7

8

```
public static void main(String args[]) throws IOException {
    Socket sock = null;
    int id = 0;
    try {
        if (args.length != 1) {
            System.out.println
                ("Usage: gradle ThreadedSockServer --args=<port num>");
            System.exit(0);
        }
        int portNo = Integer.parseInt(args[0]);
        if (portNo <= 1024)
            portNo = 8888;
        ServerSocket serv = new ServerSocket(portNo);

        while (true) {
            System.out.println
                ("Threaded server waiting for connects on port " + portNo);
            sock = serv.accept();
            System.out.println
                ("Threaded server connected to client-" + id);
            // create thread
            ThreadedSockServer myServerThread =
                new ThreadedSockServer(sock, id++);
            // run thread and don't care about managing it
            myServerThread.start();
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        if (sock != null) sock.close();
    }
}
```

SER 321 Threads

```
public void run() {
    try {
        // setup read/write channels for connection
        ObjectInputStream in = new ObjectInputStream(conn.getInputStream());
        ObjectOutputStream out = new ObjectOutputStream(conn.getOutputStream());

        // read the digit being send
        String s = (String) in.readObject();
        int index;
        // while client hasn't ended
        while (!s.equals("end")) {
            Boolean validInput = true;

            // checks if input only contains digits
            if (!s.matches(expr: "\\d+")) {
                validInput = false;
                out.writeObject("Not a number: https://gph.is/2yDymkn");
            }
        }
    }
}
```

Client
A

Server

```
// if it contains only numbers
if (validInput) {
    // convert to an integer
    index = Integer.valueOf(s);
    System.out.println("From client " + id + " get string " + index);
    if (index > -1 & index < buf.length) {
        // if valid, pull the line from the buffer array above and write it to socket
        out.writeObject(buf[index]);
    } else if (index == 5) {
        // fun surprise for mostly correct
        out.writeObject("Close but out of range: https://youtu.be/dQw4w9WgXcQ");
    } else {
        // really wrong
        out.writeObject("index out of range");
    }
}

// wait for next token from the user
s = (String) in.readObject();
}

// on close, clean up
System.out.println("Client " + id + " closed connection.");
in.close();
out.close();
conn.close();
} catch (Exception e) {
    e.printStackTrace();
}
}
```

```
public static void main(String args[]) throws IOException {
    Socket sock = null;
    int id = 0;
    try {
        if (args.length != 1) {
            System.out.println
                ("Usage: gradle ThreadedSockServer --args=<port num>");
            System.exit( code: 0);
        }
        int portNo = Integer.parseInt(args[0]);
        if (portNo <= 1024)
            portNo = 8888;
        ServerSocket serv = new ServerSocket(portNo);

        while (true) {
            System.out.println
                ("Threaded server waiting for connects on port " + portNo);
            sock = serv.accept();
            System.out.println
                ("Threaded server connected to client-" + id);
            // create thread
            ThreadedSockServer myServerThread =
                new ThreadedSockServer(sock, id++);
            // run thread and don't care about managing it
            myServerThread.start();
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        if (sock != null) sock.close();
    }
}
}
```

SER 321 Threads

```
public void run() {
    try {
        // setup read/write channels for connection
        ObjectInputStream in = new ObjectInputStream(conn.getInputStream());
        ObjectOutputStream out = new ObjectOutputStream(conn.getOutputStream());

        // read the digit being send
        String s = (String) in.readObject();
        int index;
        // while client hasn't ended
        while (!s.equals("end")) {
            Boolean validInput = true;

            // checks if input only contains digits
            if (!s.matches(expr: "\\d+")) {
                validInput = false;
                out.writeObject("Not a number: https://gph.is/2yDymkn");
            }
        }
    }
}
```

Client
A

Server

Client
B

```
// if it contains only numbers
if (validInput) {
    // convert to an integer
    index = Integer.valueOf(s);
    System.out.println("From client " + id + " get string " + index);
    if (index > -1 & index < buf.length) {
        // if valid, pull the line from the buffer array above and write it to socket
        out.writeObject(buf[index]);
    } else if (index == 5) {
        // fun surprise for mostly correct
        out.writeObject("Close but out of range: https://youtu.be/dQw4w9WgXcQ");
    } else {
        // really wrong
        out.writeObject("index out of range");
    }
}

// wait for next token from the user
s = (String) in.readObject();

// on close, clean up
System.out.println("Client " + id + " closed connection.");
in.close();
out.close();
conn.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

```
public static void main(String args[]) throws IOException {
    Socket sock = null;
    int id = 0;
    try {
        if (args.length != 1) {
            System.out.println
                ("Usage: gradle ThreadedSockServer --args=<port num>");
            System.exit(0);
        }

        int portNo = Integer.parseInt(args[0]);
        if (portNo <= 1024)
            portNo = 8888;
        ServerSocket serv = new ServerSocket(portNo);

        while (true) {
            System.out.println
                ("Threaded server waiting for connects on port " + portNo);
            sock = serv.accept();
            System.out.println
                ("Threaded server connected to client-" + id);
            // create thread
            ThreadedSockServer myServerThread =
                new ThreadedSockServer(sock, id++);
            // run thread and don't care about managing it
            myServerThread.start();
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        if (sock != null) sock.close();
    }
}
```



SER 321 Threads

```
public void run() {
    try {
        // setup read/write channels for connection
        ObjectInputStream in = new ObjectInputStream(conn.getInputStream());
        ObjectOutputStream out = new ObjectOutputStream(conn.getOutputStream());

        // read the digit being send
        String s = (String) in.readObject();
        int index;
        // while client hasn't ended
        while (!s.equals("end")) {
            Boolean validInput = true;

            // checks if input only contains digits
            if (!s.matches(expr: "\\d+")) {
                validInput = false;
                out.writeObject("Not a number: https://gph.is/2yDymkn");
            }
        }
    }
}
```



Client
A

```
// if it contains only numbers
if (validInput) {
    // convert to an integer
    index = Integer.valueOf(s);
    System.out.println("From client " + id + " get string " + index);
    if (index > -1 & index < buf.length) {
        // if valid, pull the line from the buffer array above and write it to socket
        out.writeObject(buf[index]);
    } else if (index == 5) {
        // fun surprise for mostly correct
        out.writeObject("Close but out of range: https://youtu.be/dQw4w9WgXcQ");
    } else {
        // really wrong
        out.writeObject("index out of range");
    }
}

// wait for next token from the user
s = (String) in.readObject();

// on close, clean up
System.out.println("Client " + id + " closed connection.");
in.close();
out.close();
conn.close();
} catch (Exception e) {
    e.printStackTrace();
}
}
```



Server

Client
B



```
public static void main(String args[]) throws IOException {
    Socket sock = null;
    int id = 0;
    try {
        if (args.length != 1) {
            System.out.println
                ("Usage: gradle ThreadedSockServer --args=<port num>");
            System.exit(0);
        }

        int portNo = Integer.parseInt(args[0]);
        if (portNo <= 1024)
            portNo = 8888;
        ServerSocket serv = new ServerSocket(portNo);

        while (true) {
            System.out.println
                ("Threaded server waiting for connects on port " + portNo);
            sock = serv.accept();
            System.out.println
                ("Threaded server connected to client-" + id);
            // create thread
            ThreadedSockServer myServerThread =
                new ThreadedSockServer(sock, id++);
            // run thread and don't care about managing it
            myServerThread.start();
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        if (sock != null) sock.close();
    }
}
}
```



SER 321

Threaded Server

Now What?

Handle the Client just like before!

1. Define Params

2. Create Socket

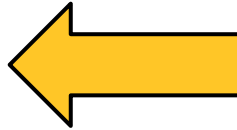
3-5. Mark Socket to Listen

6. Wait for Connection

7. Send Client **Socket** to Thread

8. Close Client Connection

9. Continue Listening



1

2

3

4

5

SER 321

Threaded Server

Now What?

Handle the Client just like before!

1. Define Params

```
public SockBaseServer(Socket sock, Game game){ 1 usage
    this.clientSocket = sock;
    this.game = game;
    try {
        in = clientSocket.getInputStream();
        out = clientSocket.getOutputStream();
    } catch (Exception e){
        System.out.println("Error in constructor: " + e);
    }
}
```

9. Continue Listening

1 Create input/output streams

2

3

4

5

SER 321

Threaded Server

Now What?

Handle the Client just like before!

```
public void start() throws IOException { 1 usage
    String name = "";
    System.out.println("Ready...");
    > try {...} catch (Exception ex) {
        ex.printStackTrace();
    } finally {
        if (out != null) out.close();
        if (in != null) in.close();
        if (clientSocket != null) clientSocket.close();
    }
}
```

9. Continue Listening

1 Create input/output streams

2 Check for disconnect

3

4

5

SER 321

Threaded Server

Now What?

Handle the Client just like before!

1. Define Params

2. Create Socket

```
Request op = Request.parseDelimitedFrom(in);
```

6. Wait for Connection

7. Send Client **Socket** to Thread

1 Create input/output streams

2 Check for disconnect

3 Check Protocol

```
public static buffers.RequestProtos.Request parseDelimitedFrom(java.io.InputStream input)
    throws java.io.IOException {
    return com.google.protobuf.GeneratedMessageV3
        .parseDelimitedWithIOException(PARSER, input);
}
```

SER 321

Threaded Server

Now What?

Handle the Client just like before!

1. Define Params

2. Create Socket

3-5. Mark Socket to Listen

```
Request op = Request.parseDelimitedFrom(in);  
String result = null;
```

```
if (op.getOperationType() == Request.OperationType.NAME) {...}
```

8. Close Client Connection

9. Continue Listening

1 Create input/output streams

2 Check for disconnect

3 Check Protocol

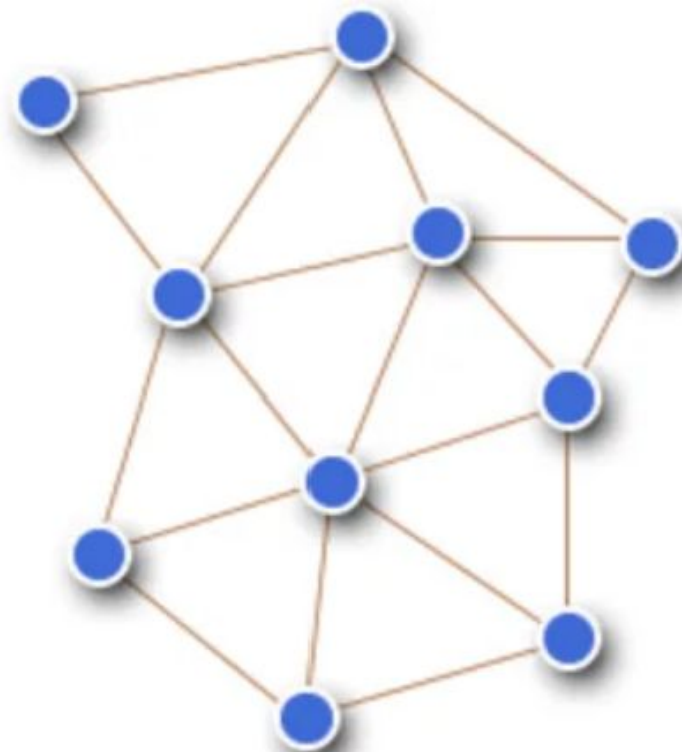
4 Read Headers

5

SER 321

Distributed Systems

What do we mean by
“Distributed Systems”
or
“Distributed Algorithms”?



SER 321

Systems

Parallel



A Venn diagram with two overlapping circles. The left circle is light blue with a blue outline and is labeled 'Parallel'. The right circle is light red with a red outline and is labeled 'Distributed'. The intersection of the two circles is shaded with a mix of blue and red. The text 'SER 321' is in a yellow box at the top left, and 'Systems' is in a black box below it.

Distributed

Parallel

- Single computer
- Work split among different *processors*
- Memory is shared **or** distributed
- Communicate through *bus*

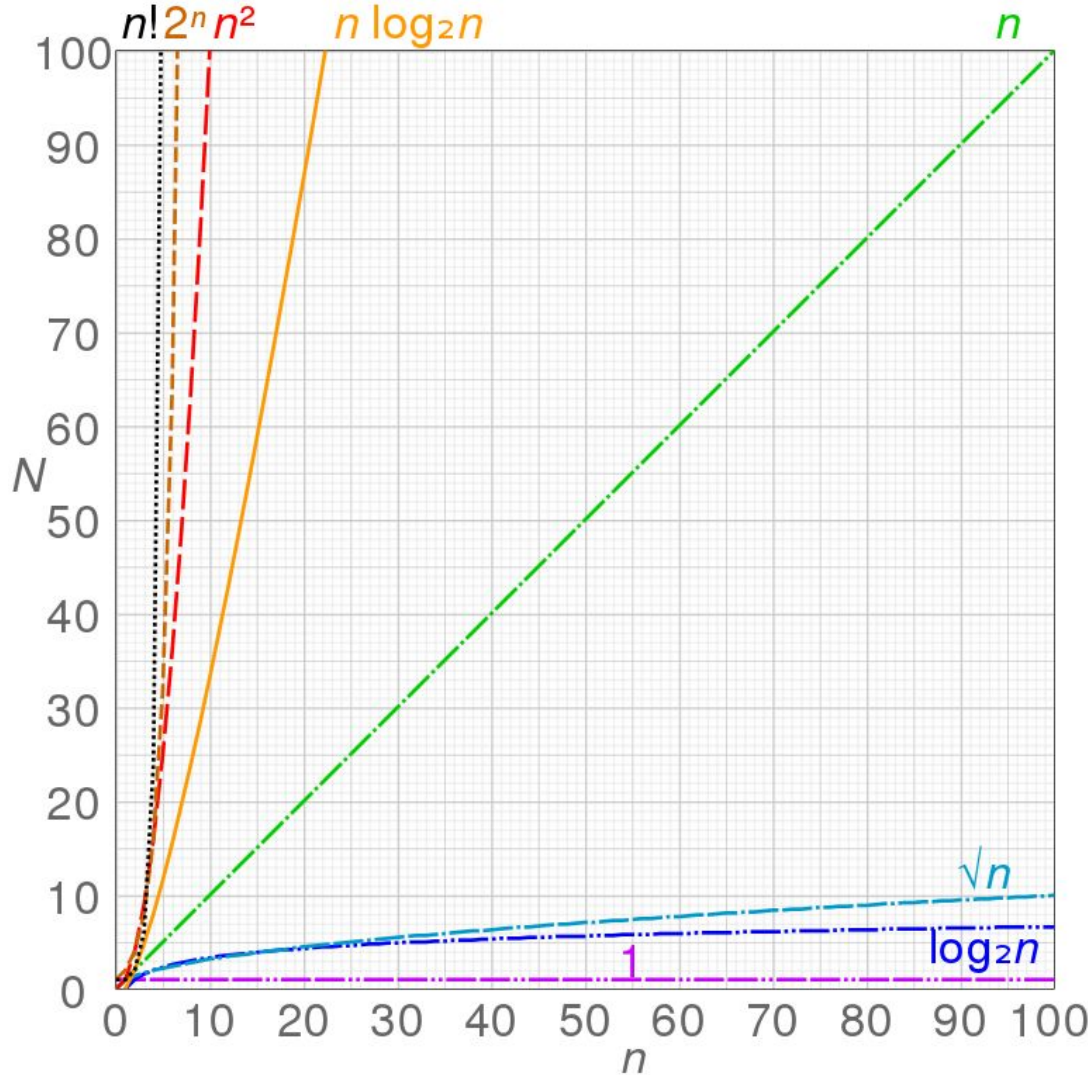
Distributed

- Many computers
 - Work split among different *locations*
 - Memory is distributed
 - Communicate through *message passing*
- Work is partitioned
 - Partitions processed individually
 - **Can** improve performance
 - **Can** improve speed

SER 321

Distributed Systems

When should
we *consider*
distributing?

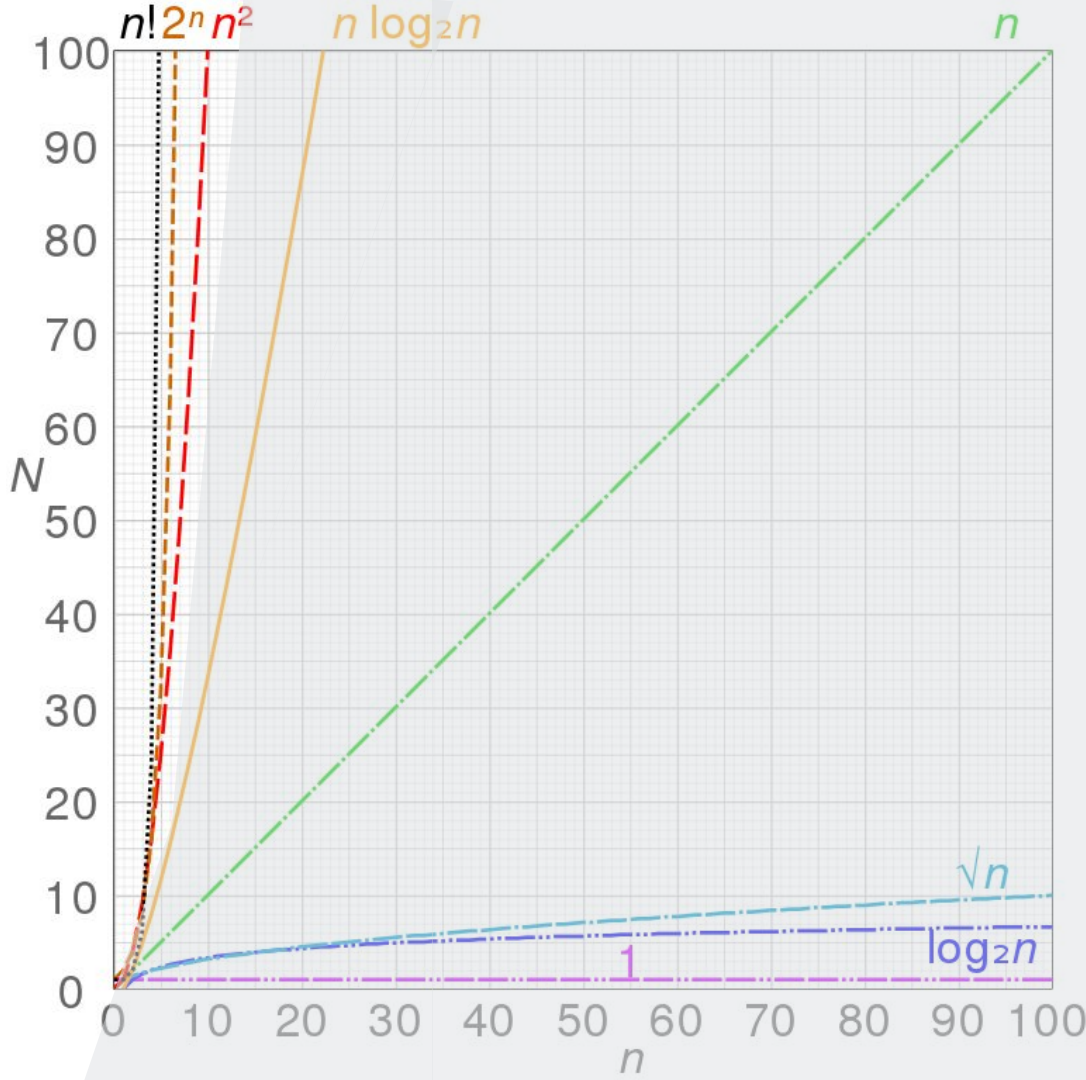


SER 321

Distributed Systems

When should
we *consider*
distributing?

Super Duper Extra Extra
Large Orders of Magnitude!

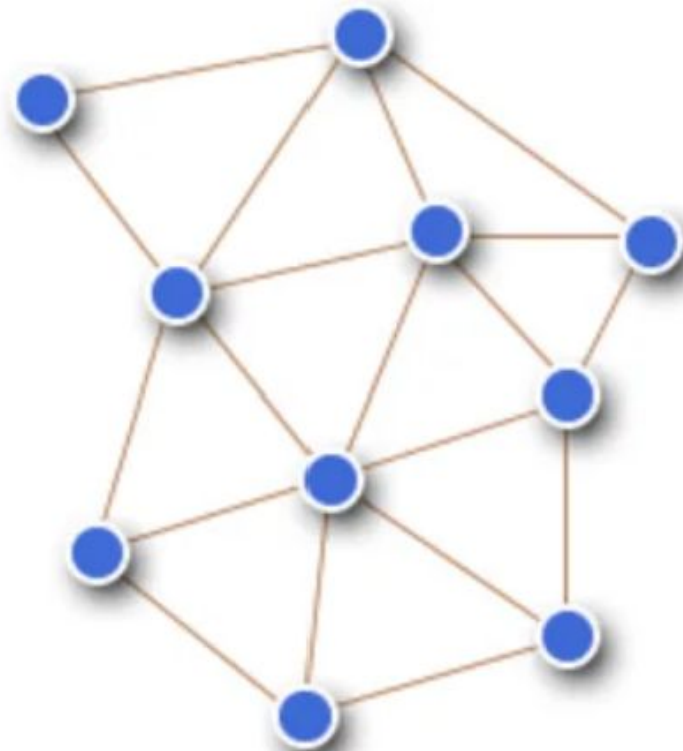


SER 321

Distributed Systems

Remember that we are operating in *reality*

- Nodes *will* fail
- Web of nodes *will constantly* change
- Network is not *always* reliable
- Latency is *always present*
- The path traversed *changes*
- Some resources *must be shared*
- *You* need to prevent the pitfalls!
 - No deadlocks
 - No starvation
 - No error states



SER 321

Scratch Space

Questions?



Survey:

<http://bit.ly/ASN2324>



Upcoming Events

SI Sessions:

- Sunday, June 23rd at 6:00 pm MST
- Monday, June 24th at 6:00 pm MST
- Thursday, June 27th at 6:00 pm MST

Review Sessions:

- Review Session - **Wednesday**, July 3rd at 6:00 pm MST (2 hr Session)
- Q&A Session - Sunday, July 7th at 6:00 pm MST (Final Session)

More Questions?

Check out our other resources!

tutoring.asu.edu



Academic Support

Academic Support Network (ASN) provides a variety of free services in-person and online to help currently enrolled ASU students succeed academically.

Services



Subject Area Tutoring

Need in-person or online help with math, science, business, or engineering courses? Just hop into our Zoom room or drop into a center for small group tutoring. We'll take it from there.

[Need help using Zoom?](#)

[View the tutoring schedule](#)

[View digital resources](#)

Go to Zoom



Writing Tutoring

Need help with undergraduate or graduate writing assignments? Schedule an in-person or online appointment, access your appointment link, or wait in our drop-in queue.

[Access your appointment link](#)

[Access the drop-in queue](#)

Schedule Appointment



Online Study Hub

Join our online peer communities to connect with your fellow Sun Devils. Engage with our tools to search our bank of resources, videos, and previously asked questions. Or, ask our Tutorbot questions.

Now supporting courses in Math, Science, Business, Engineering, and Writing.

Online Study Hub

1-

Go to Zoom

2-

[Need help using Zoom?](#)

[View the tutoring schedule](#)

[View digital resources](#)



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

 **Academic Support Network**

 [Services](#)  [Faculty and Staff Resources](#) [About Us](#) 

[University College](#)

Online Study Hub

Online peer communities for students and tutors, YouTube channels, and Tutorbots.



What are online peer communities?

Individual courses have an online peer community that allows you to connect with your peers to post and answer questions and to develop study groups.



How can tutoring center videos help?

Videos can help supplement the learning you're doing in and outside of class and include step-by-step methods for how to understand concepts.



How does the Tutorbot work?

You can ask the Tutorbot questions about course concepts and the Tutorbot will recommend additional resources and examples to help address your questions.

Select a subject

- Any -

[Apply](#)



Academic Support Network



[Services](#) 

[Faculty and Staff Resources](#)

[About Us](#) 

[University College](#)

Select a subject

- Any -

[Apply](#)

Business


ACC 231

Uses of Accounting Info I

 [Peer Community](#)

ACC 241

Uses of Accounting Info II

 [Peer Community](#)

CIS 105

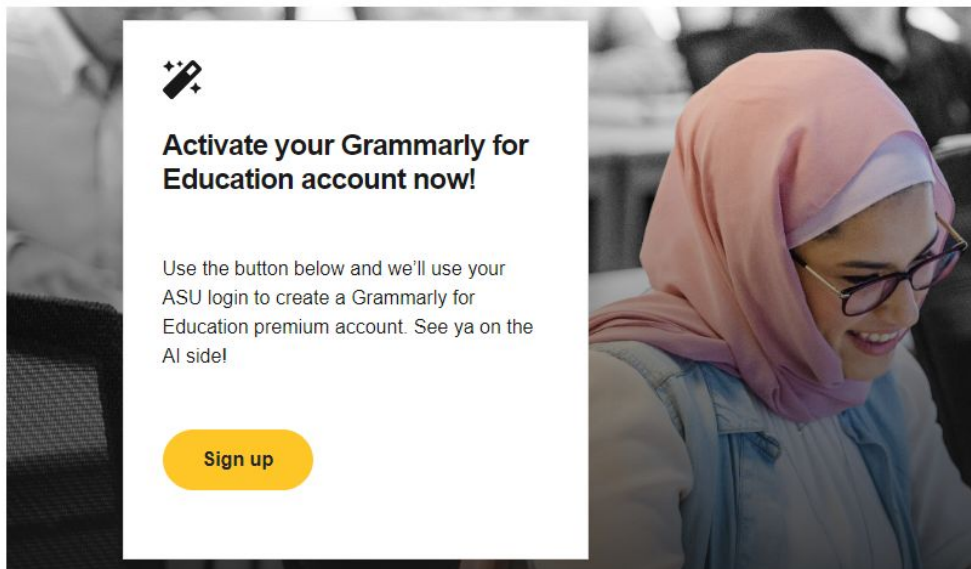
Computer Applications and Information Technology

 [Peer Community](#)

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](#)