

SER 321 A Session

SI Session

Tuesday, February 11th 2025

11:00 am - 12:00 pm MST

Agenda



Threading the Server

Where and How

Thread Tracing

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

Single Threaded Server

What will happen if there are two clients?

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sockets
\JavaSimpleSock> gradle socketServer
```

```
> Task :SocketServer
Server ready for a connection
Server waiting for a connection
<=====--> 75% EXECUTING [20s]
```

```
> :SocketServer
```

Server

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sock
ets\JavaSimpleSock> gradle socketClient
```

```
> Task :SocketClient
Please enter a String to send to the Server (enter
"exit" to quit):
```

```
<=====--> 75% EXECUTING [14s]
```

```
> :SocketClient
```

Client 1

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sock
ets\JavaSimpleSock> gradle socketClient
```

Client 2

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sockets
\JavaSimpleSock> gradle socketServer
```

```
> Task :SocketServer
Server ready for a connection
Server waiting for a connection
<=====--> 75% EXECUTING [53s]
```

```
> :SocketServer
```

Server

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sock
ets\JavaSimpleSock> gradle socketClient
```

```
> Task :SocketClient
Please enter a String to send to the Server (enter
"exit" to quit):
```

```
<=====--> 75% EXECUTING [47s]
```

```
> :SocketClient
```

```
Hello!
```

Client 1

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sock
ets\JavaSimpleSock> gradle socketClient
```

```
Starting a Gradle Daemon, 2 busy and 4 stopped Dae
mons could not be reused, use --status for details
<=====--> 75% EXECUTING [15s]
```

```
> :SocketClient
```

Client 2

SER 321

Single Threaded Server

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sockets\JavaSimpleSock> gradle socketServer
```

```
> Task :SocketServer
Server ready for a connection
Server waiting for a connection
Received the String Hello!
Received the Integer 9
<=====--> 75% EXECUTING [1m 27s]
> :SocketServer
█
```

Server

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sockets\JavaSimpleSock> gradle socketClient
```

```
> Task :SocketClient
Please enter a String to send to the Server (enter "exit" to quit):
<=====--> 75% EXECUTING [59s]      P
Please enter a Number to send to the Server (enter 0 to quit):
<=====--> 75% EXECUTING [1m 18s]    9
and Hello! ... Got it!
Please enter a String to send to the Server (enter "exit" to quit):
<=====--> 75% EXECUTING [1m 21s]
> :SocketClient
█
```

Client 1

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sockets\JavaSimpleSock> gradle socketClient
```

```
Starting a Gradle Daemon, 2 busy and 4 stopped Daemons could not be reused, use --status for details
<=====--> 75% EXECUTING [49s]
> :SocketClient
█
```

Client 2

SER 321

Single Threaded Server

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sockets\JavaSimpleSock> gradle socketServer
```

```
> Task :SocketServer
Server ready for a connection
Server waiting for a connection
Received the String Hello!
Received the Integer 9
<=====----> 75% EXECUTING [1m 55s]
> :SocketServer
█
```

Server

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sockets\JavaSimpleSock> gradle socketClient
```

```
> Task :SocketClient
Please enter a String to send to the Server (enter "exit" to quit):
<=====----> 75% EXECUTING [59s] P
Please enter a Number to send to the Server (enter 0 to quit):
<=====----> 75% EXECUTING [1m 18s] 9
and Hello! ... Got it!
Please enter a String to send to the Server (enter "exit" to quit):
<=====----> 75% EXECUTING [1m 49s]
> :SocketClient
exit█
```

Client 1

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sockets\JavaSimpleSock> gradle socketClient
```

```
Starting a Gradle Daemon, 2 busy and 4 stopped Daemons could not be reused, use --status for details
<=====----> 75% EXECUTING [1m 18s]
> :SocketClient
```

Client 2

What do we think will happen?

SER 321

Single Threaded Server

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sockets
\JavaSimpleSock> gradle socketServer
```

```
> Task :SocketServer
Server ready for a connection
Server waiting for a connection
Received the String Hello!
Received the Integer 9
Received the String exit
Received the Integer 0
Server waiting for a connection
<=====--> 75% EXECUTING [2m 15s]
> :SocketServer
█
```

Server

```
and Hello! ... Got it!
Please enter a String to send to the Server (enter
"exit" to quit):
<=====--> 75% EXECUTING [2m 3s]
xitingketClient
```

Deprecated Gradle features were used in this build
, making it incompatible with Gradle 8.0.

You can use '--warning-mode all' to show the individ-
ual deprecation warnings and determine if they come
from your own scripts or plugins.

See https://docs.gradle.org/7.4.2/userguide/command_line_interface.html#sec:command_line_warnings

```
BUILD SUCCESSFUL in 2m 5s
2 actionable tasks: 1 executed, 1 up-to-date
PS C:\ASU\SER321\examples_repo\ser321examples\Sock
ets\JavaSimpleSock>
```

Client 1

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sock
ets\JavaSimpleSock> gradle socketClient
Starting a Gradle Daemon, 2 busy and 4 stopped Dae
mons could not be reused, use --status for details
```

```
> Task :SocketClient
Please enter a String to send to the Server (enter
"exit" to quit):
<=====--> 75% EXECUTING [1m 37s]
> :SocketClient
█
```

Client 2

SER 321

Single Threaded Server



<pre>PS C:\ASU\SER321\examples_repo\ser321examples\Sockets \JavaSimpleSock> gradle socketServer > Task :SocketServer Server ready for a connection Server waiting for a connection Received the String Hello! Received the Integer 9 Received the String exit Received the Integer 0 Server waiting for a connection Received the String Hello! <=====--> 75% EXECUTING [3m 7s] > :SocketServer █</pre>	<pre>and Hello! ... Got it! Please enter a String to send to the Server (enter "exit" to quit"): <=====--> 75% EXECUTING [2m 3s] e xitingketClient Deprecated Gradle features were used in this build , making it incompatible with Gradle 8.0. You can use '--warning-mode all' to show the individ ual deprecation warnings and determine if they c ome from your own scripts or plugins. See https://docs.gradle.org/7.4.2/userguide/command_line_interface.html#sec:command_line_warnings BUILD SUCCESSFUL in 2m 5s 2 actionable tasks: 1 executed, 1 up-to-date PS C:\ASU\SER321\examples_repo\ser321examples\Sockets\JavaSimpleSock> █</pre>	<pre>PS C:\ASU\SER321\examples_repo\ser321examples\Sockets\JavaSimpleSock> gradle socketClient Starting a Gradle Daemon, 2 busy and 4 stopped Daemons could not be reused, use --status for details > Task :SocketClient Please enter a String to send to the Server (enter "exit" to quit"): <=====--> 75% EXECUTING [2m 24s] P lease enter a Number to send to the Server (enter 0 to quit"): <=====--> 75% EXECUTING [2m 30s] > :SocketClient 77 █</pre>
---	--	--

Server

Client 1

Client 2

SER 321

Single Threaded Server

Why?



```
PS C:\ASU\SER321\examples_repo\ser321examples\Sockets\JavaSimpleSock> gradle socketServer

> Task :SocketServer
Server ready for a connection
Server waiting for a connection
Received the String Hello!
Received the Integer 9
Received the String exit
Received the Integer 0
Server waiting for a connection
Received the String Hello!
<=====--> 75% EXECUTING [3m 7s]
> :SocketServer
█
```

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.

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sockets\JavaSimpleSock> gradle socketClient
Starting a Gradle Daemon, 2 busy and 4 stopped Daemons could not be reused, use --status for details

> Task :SocketClient
Please enter a String to send to the Server (enter "exit" to quit"):
<=====--> 75% EXECUTING [2m 24s]
Please enter a Number to send to the Server (enter 0 to quit"):
<=====--> 75% EXECUTING [2m 30s]
> :SocketClient
77█
```

Server

Client 1

Client 2

SER 321

Threaded Server

Given the standard server socket steps...

Ideas on how we could introduce threads?

1. Define Params

2. Create Socket

3-5. Mark Socket to Listen

6. Wait for Connection

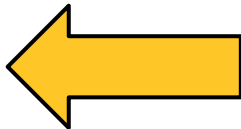
7. Handle Client Connection

8. Close Client Connection

9. Continue Listening

Why do we send the *client socket* to the thread?

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);
        }

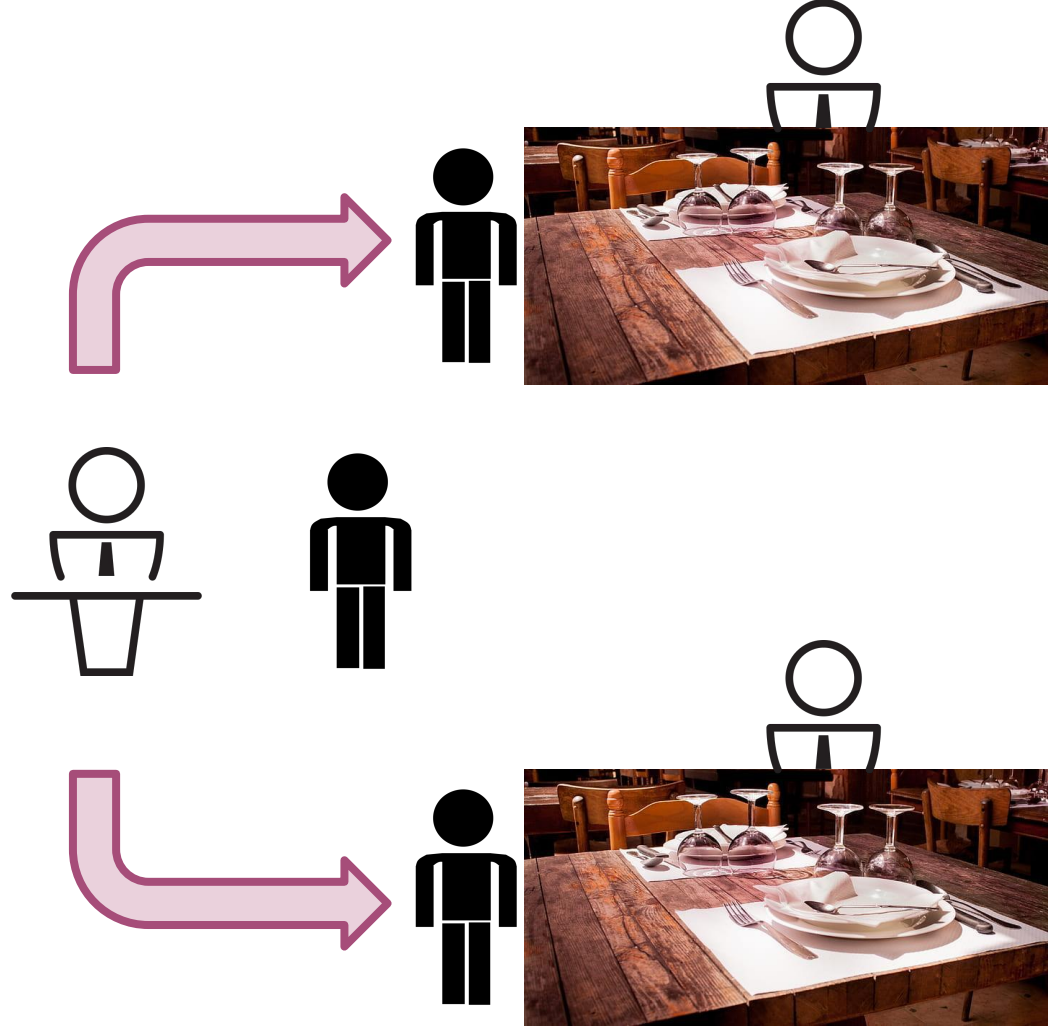
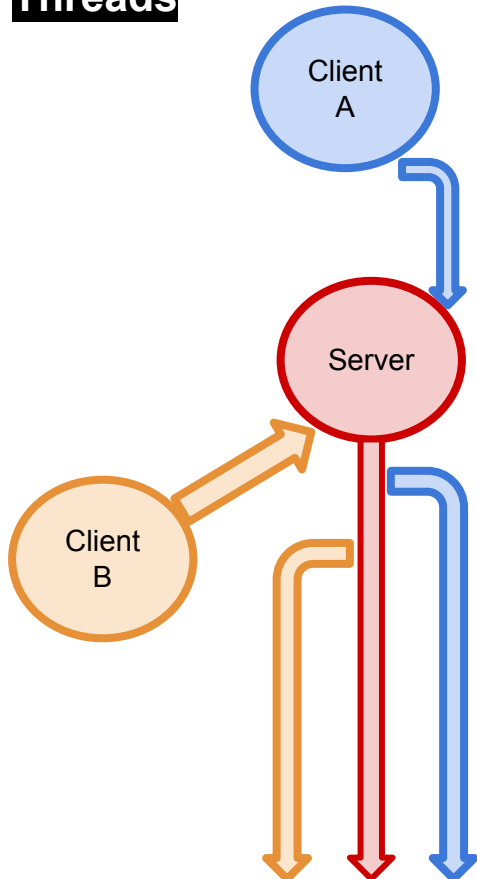
        int portNo = Integer.parseInt(args[0]);
        if (portNo <= 1024)
            portNo = 8888;
        ServerSocket serv = new ServerSocket(portNo);

        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

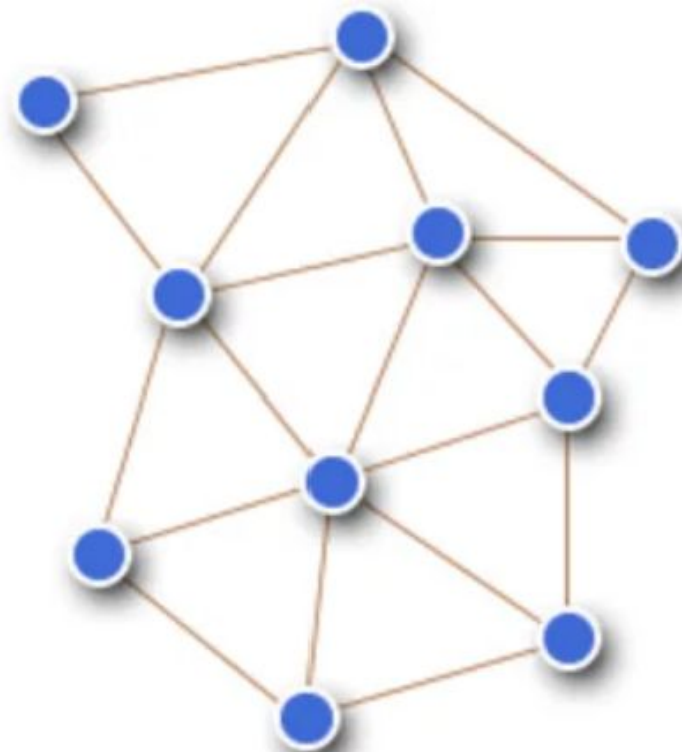
Threads



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*

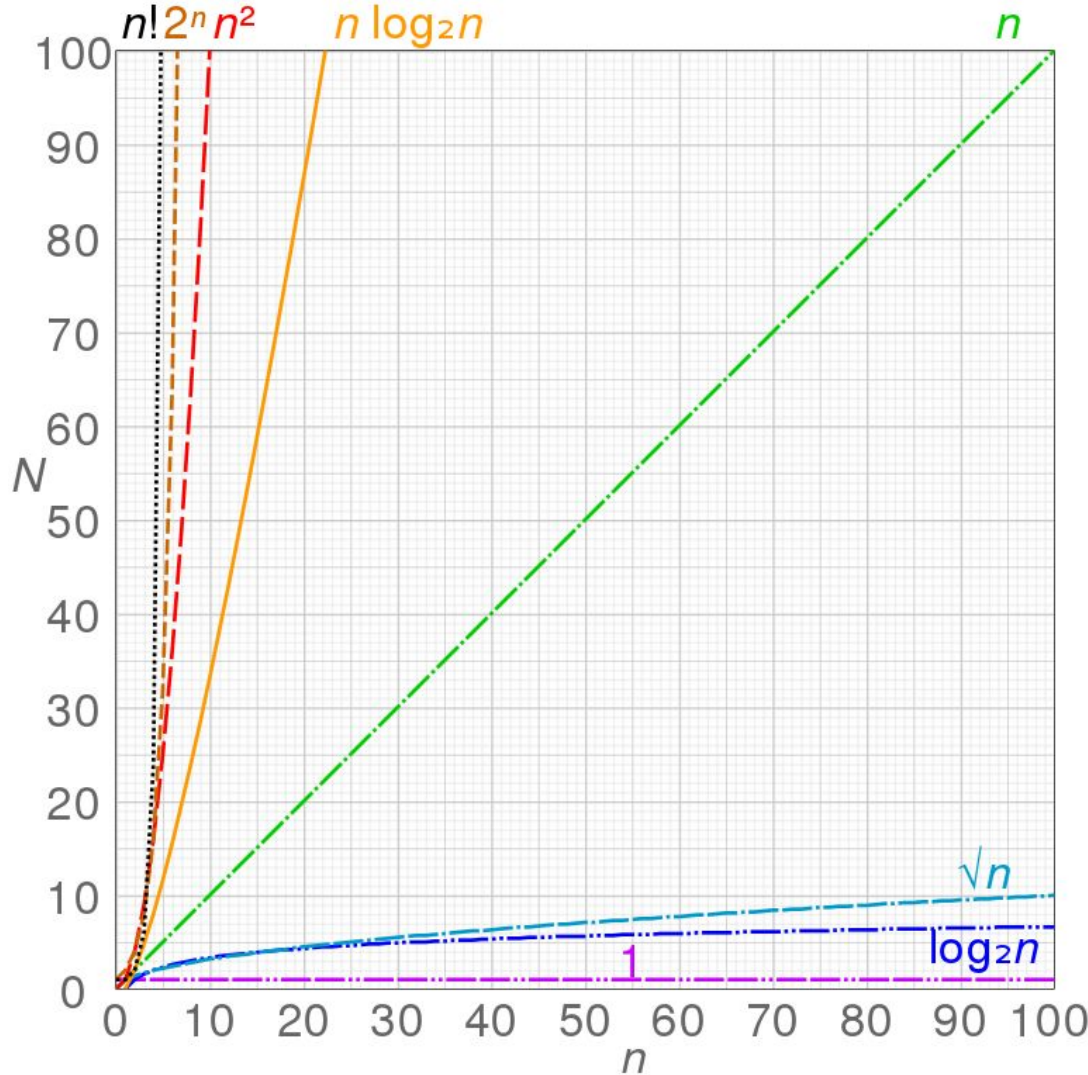
Distributed

- 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

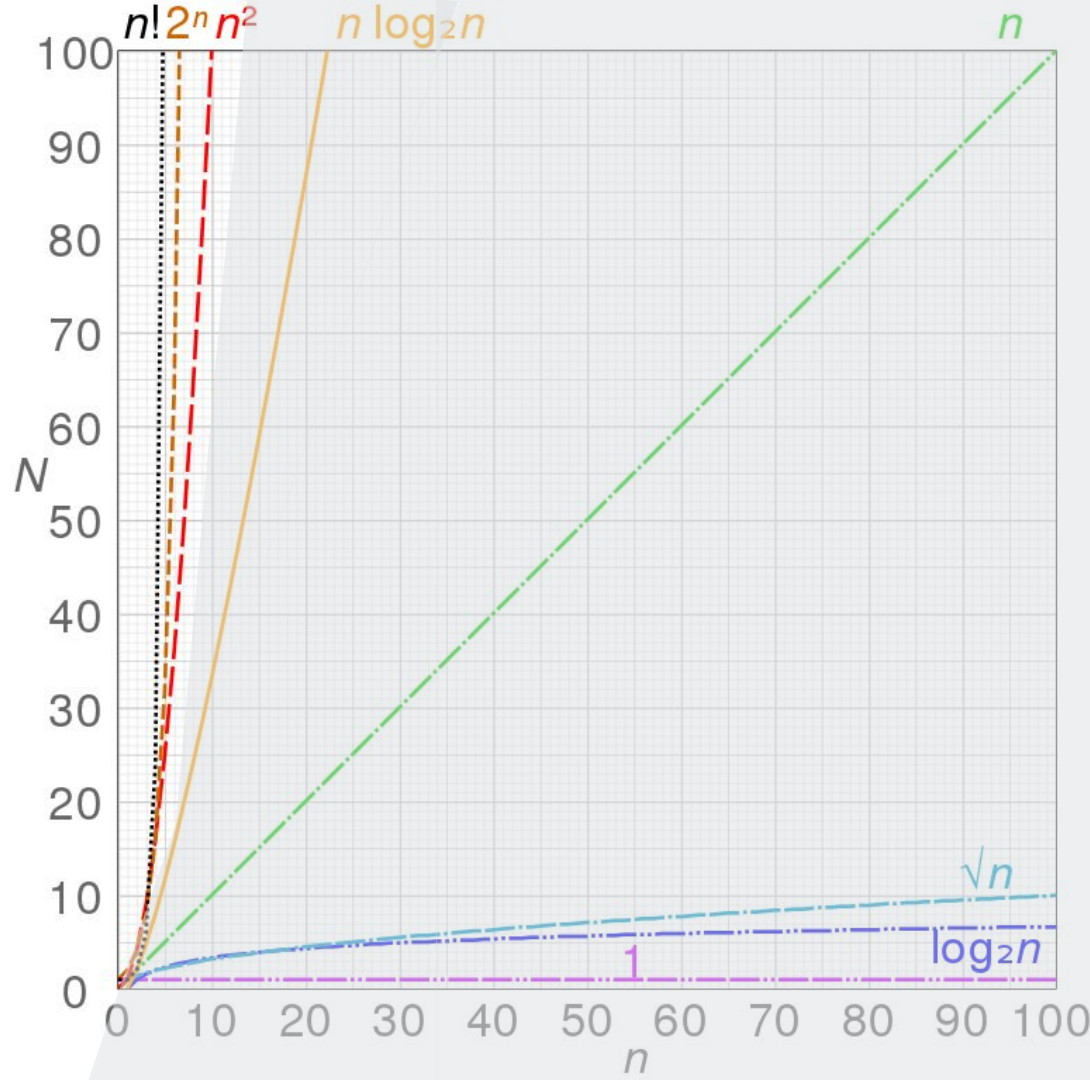
When should
we *consider*
distributing?



Distributed Systems

When should
we *consider*
distributing?

Super Duper Extra Extra
Large Orders of Magnitude!

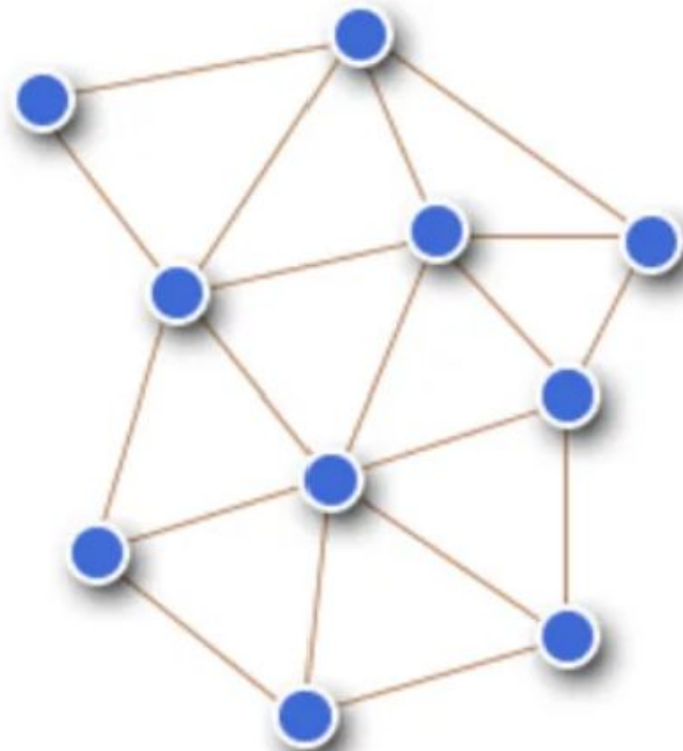


SER 321

Distributed Systems

Remember that we are operating in *reality*

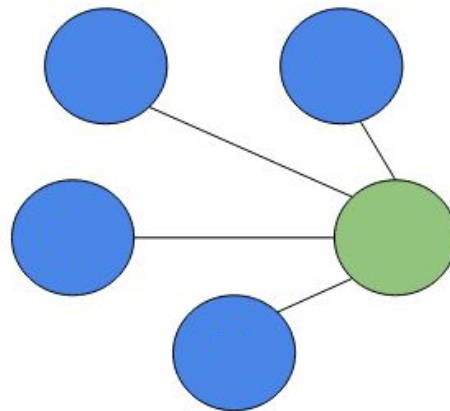
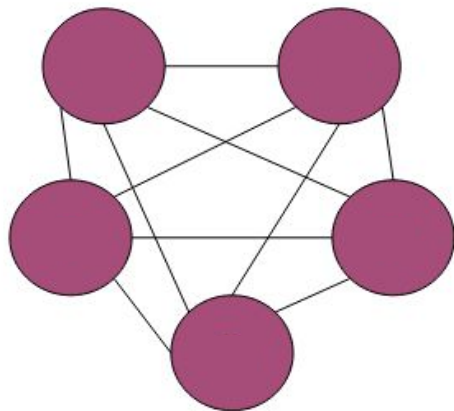
- Nodes *will* fail
- Web of nodes *will constantly* change
- Network is not *always* reliable
- Latency is *always present*
- The path traversed *changes*
- Some resources *must be shared*
- *You* need to prevent the pitfalls!
 - No deadlocks
 - No starvation
 - No error states



Main and Worker

Peer to Peer

Which is which?



SER 321

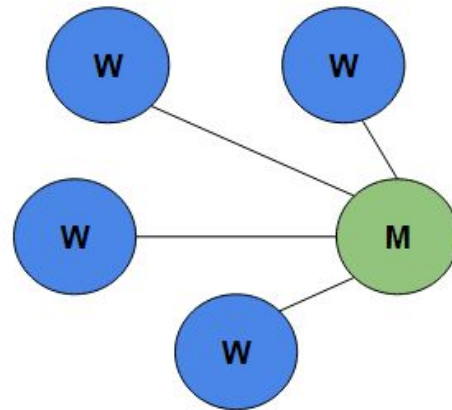
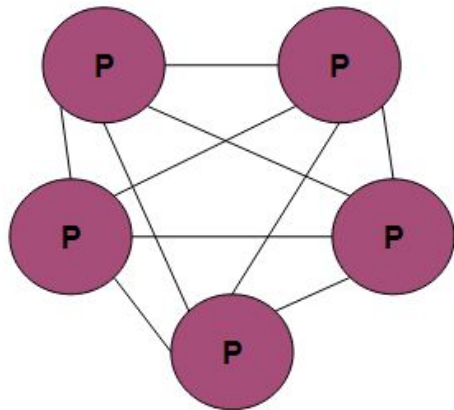
Distributed Systems

Main and Worker

Peer to Peer

Which is which?

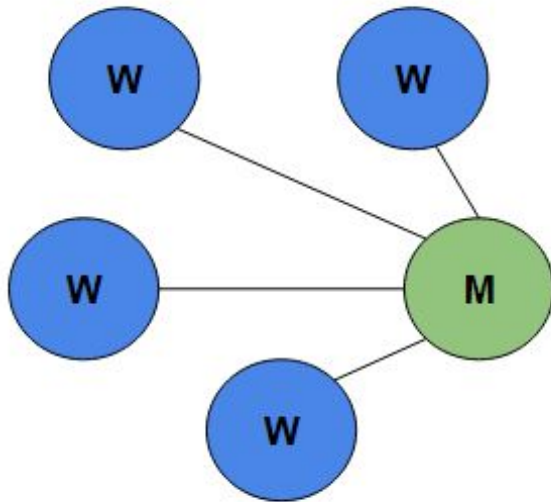
Peer to Peer



Main and
Worker

SER 321

Distributed Systems



Pros and Cons

Pros:

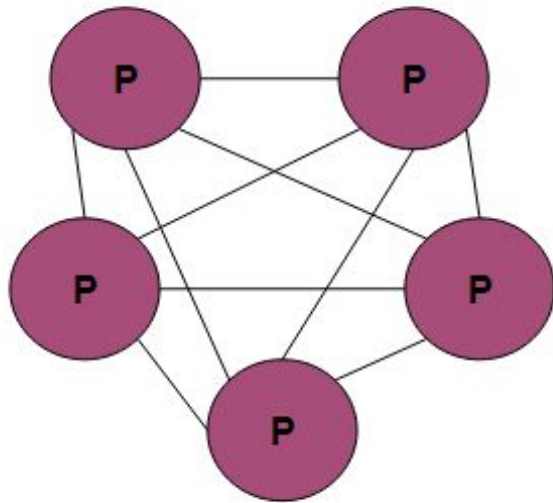
- Straightforward setup
- Logic is centralized
- Communication is linear

Cons:

- Single point of failure

SER 321

Distributed Systems



Pros and Cons

Pros:

- Peers can join or leave as needed
- Robust - no single point of failure

Cons:

- Communication is more *complex*
- Setup is not as straightforward
- Client connections are handled *differently*

We will cover this in a moment!

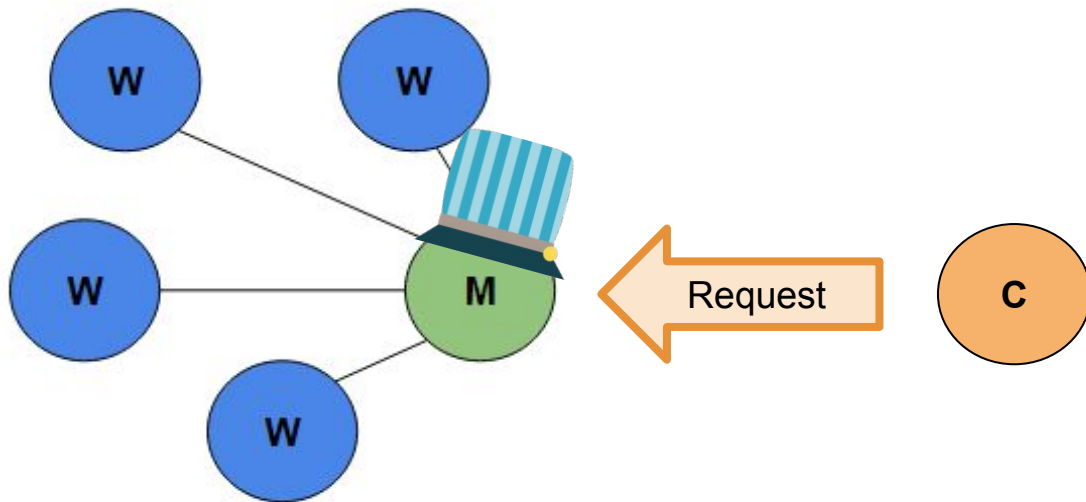
SER 321

Distributed Systems

Process Flow!

DATA

Workers
only do
their task
then report
back



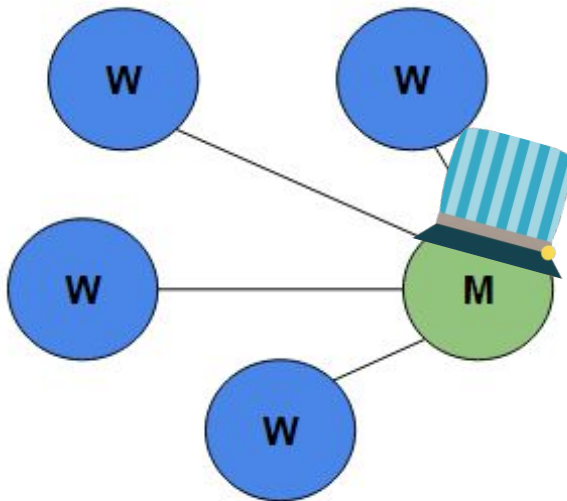
Main is like our server

SER 321

Distributed Systems

Process Flow!

Workers
only do
their task
then report
back



DATA



D1

D2

D3

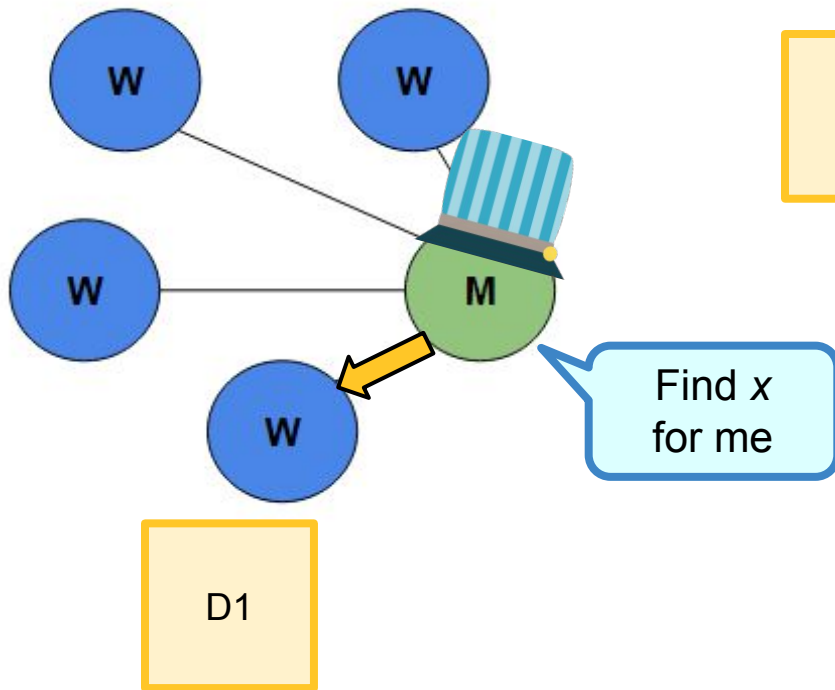
D4

SER 321

Distributed Systems

Process Flow!

Workers
only do
their task
then report
back



DATA



D1

D2

D3

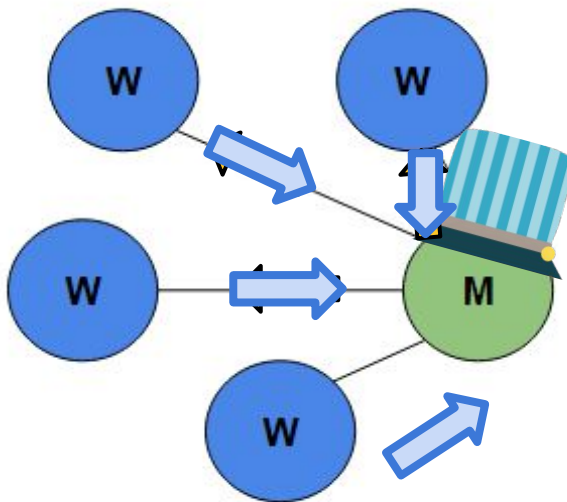
D4

SER 321

Distributed Systems

Process Flow!

Workers
only do
their task
then report
back



D1

DATA



D1

D2

D3

D4

D1
Result

D2
Result

D3
Result

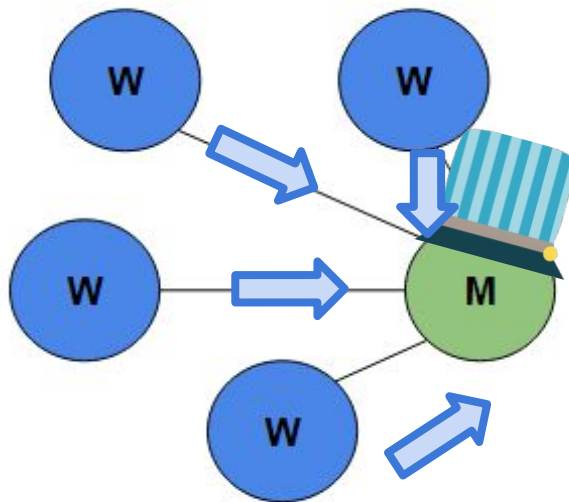
D4
Result

SER 321

Distributed Systems

Process Flow!

Workers
only do
their task
then report
back



D1

DATA



D1

D2

D3

D4

D1
Result

D2
Result

D3
Result

D4
Result



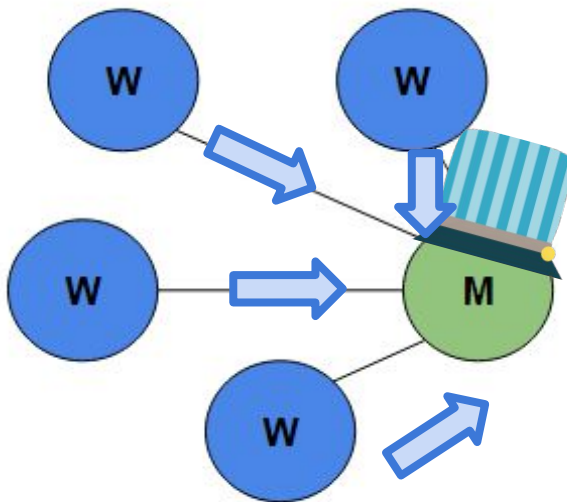
RESULTS

SER 321

Distributed Systems

Does this look familiar?

How is this different from a parallel processing model?



D1

DATA



D1

D2

D3

D4

D1
Result

D2
Result

D3
Result

D4
Result



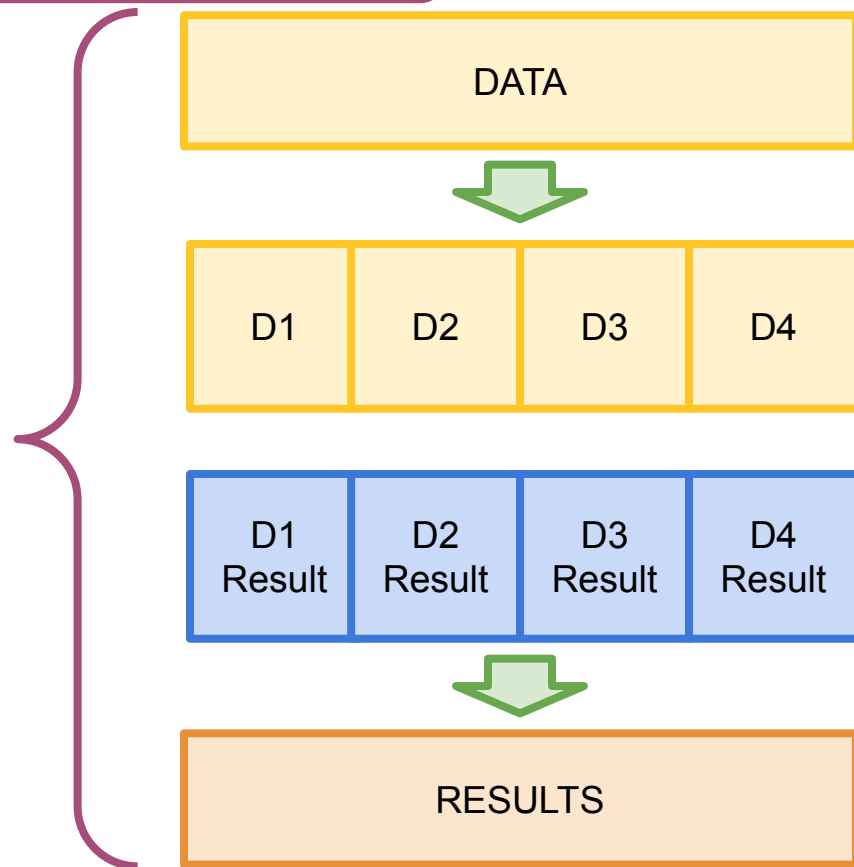
RESULTS

SER 321

Distributed Systems

What about Peer to Peer?

Would this sequence
(*the data handling*) change
in the different structure?



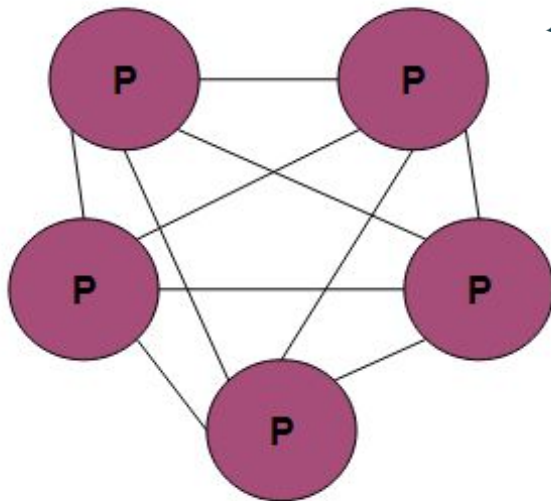
SER 321

Distributed Systems

What about Peer to Peer?

We want
someone to
wear the
conductor
hat!

A **LEADER**



How do we choose a leader?

DATA



D1

D2

D3

D4

D1
Result

D2
Result

D3
Result

D4
Result



RESULTS

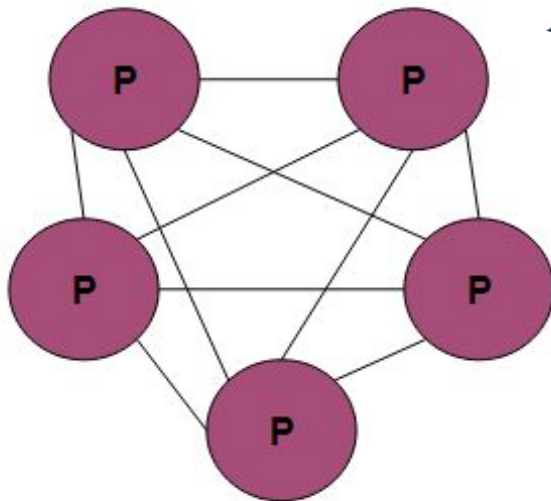
SER 321

Distributed Systems

What about Peer to Peer?

We want
someone to
wear the
conductor
hat!

A *LEADER*



Leader Election!

DATA



D1

D2

D3

D4

D1
Result

D2
Result

D3
Result

D4
Result



RESULTS

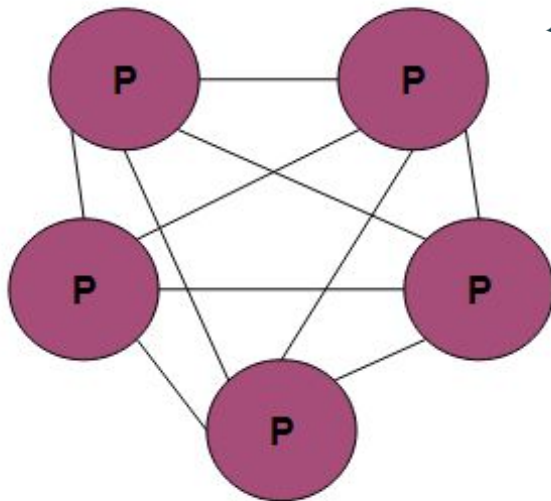
SER 321

Distributed Systems

What about Peer to Peer?

We want
someone to
wear the
conductor
hat!

A **LEADER**



Leader Election!

Type of
CONSENSUS

What's
consensus?



“General agreement or
trust amongst a group”

SER 321

Consensus

“General agreement or trust amongst a group”

Types of Consensus?

Leader Election



Who's in charge or keeping the beat

Verify Results



Check your work with a neighbor

Synchronize Data



Verify and maintain my copy of the data

Validate Nodes



Do I want to let you into my network

SER 321

Scratch Space

Upcoming Events

SI Sessions:

- Thursday, February 13th at 7:00 pm MST
- Sunday, February 16th at 7:00 pm MST
- Tuesday, February 18th 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>



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

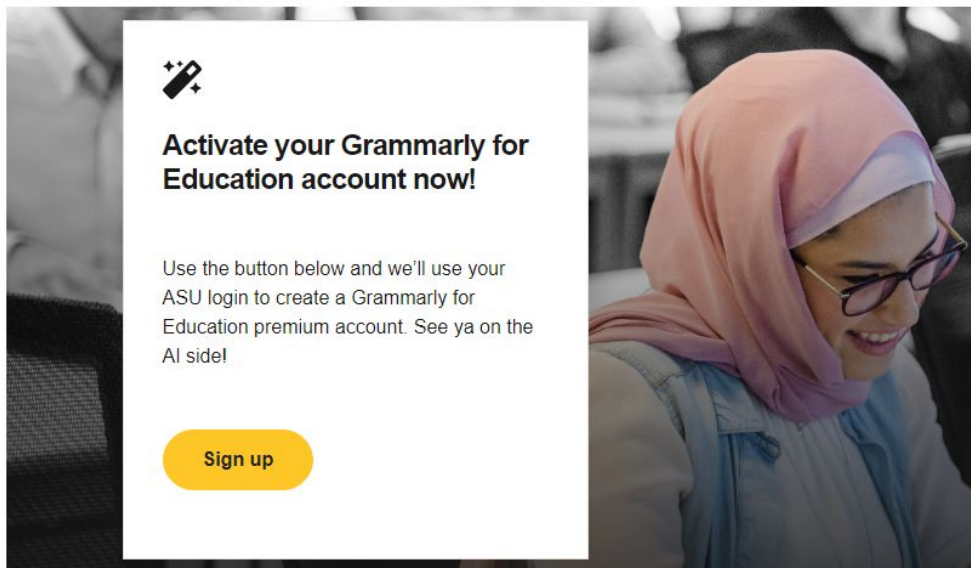
Computer Applications and Information Technology

 [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](#)