

SER 321 B Session

SI Session

Tuesday, April 22nd 2025

10:00 am - 11:00 am MST

Agenda



Process Flow Examination

Consensus!

Types and Algorithms

RAFT

Peer to Peer Differences

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

Assignment 5 PSA

No starter code for this assignment



Don't panic - you have options!

SER 321

Assignment 5 PSA

No starter code for this assignment

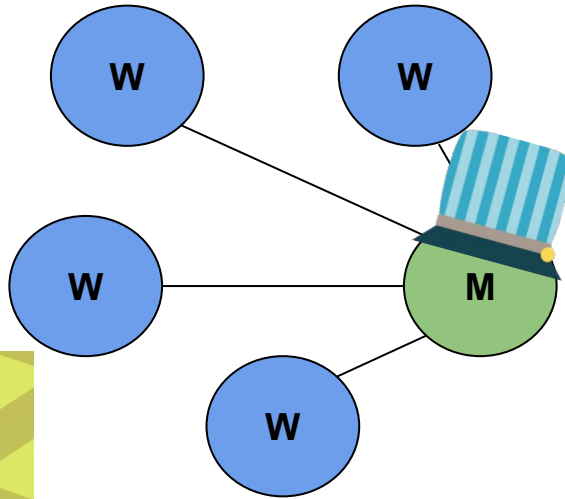
Use a previous
assignment as a
starting point

Use a repo
example as a
starting point

Build from scratch

SER 321

Communication



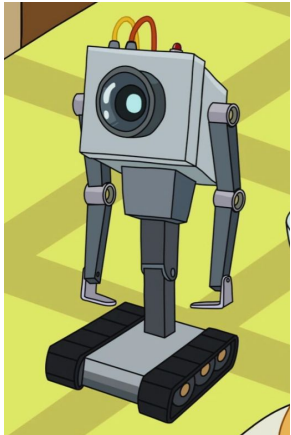
Workers
only do
their task
then report
back

Think worker bees
or
specialized machinery



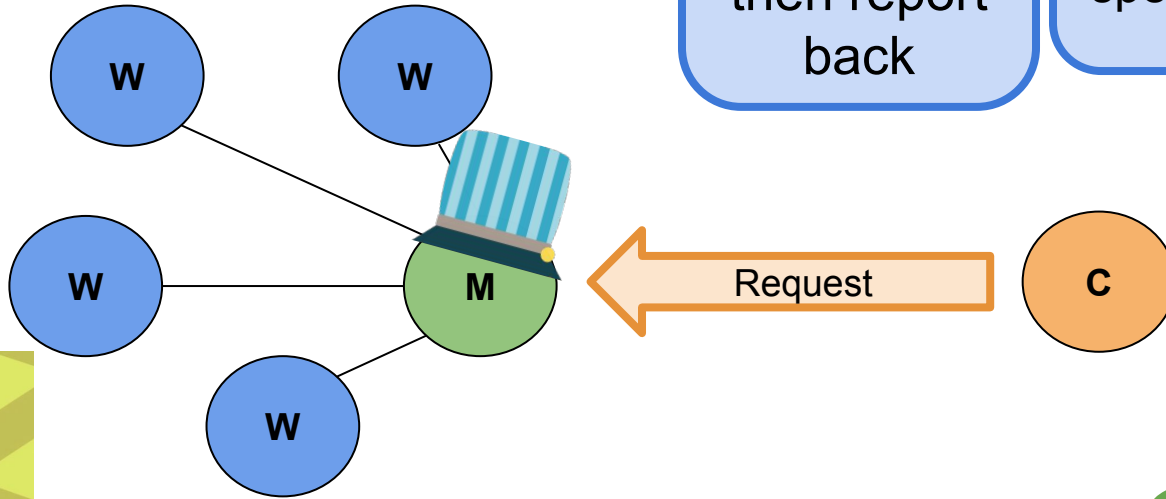
Main is in charge of everything

Similar to _____
in our previous
implementations



SER 321

Communication



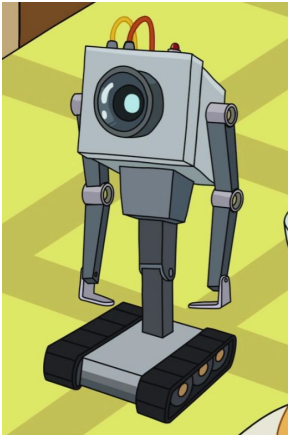
Workers
only do
their task
then report
back

Think worker bees
or
specialized machinery

What about
Client
Requests?
Where do those
get handled?

Main is in charge of everything

Similar to _____
in our previous
implementations

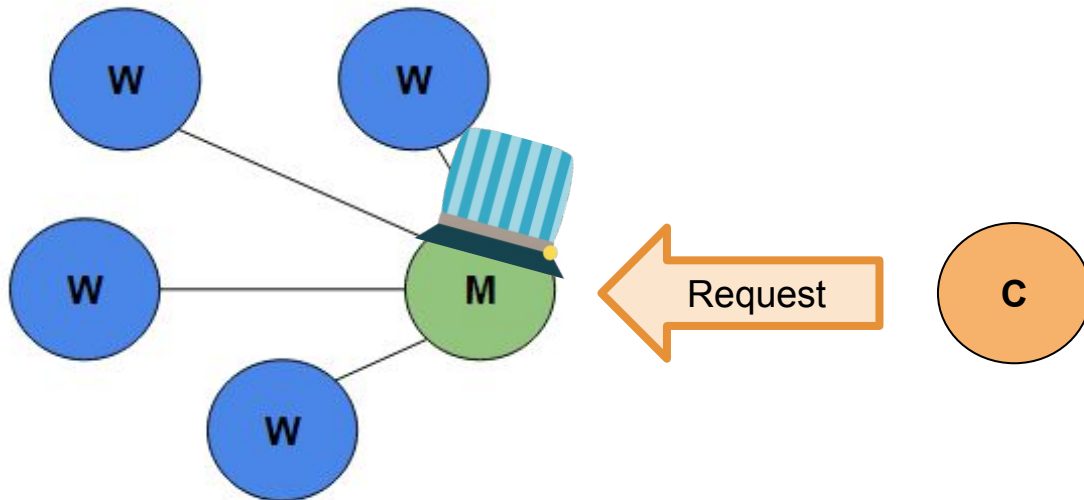


SER 321

Distributed Systems

Process Flow!

DATA

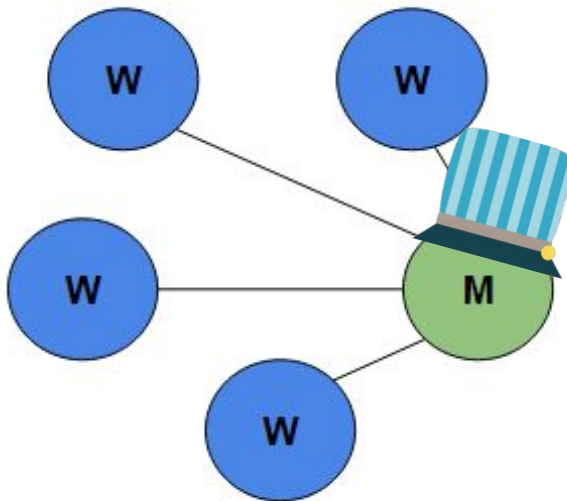


SER 321

Distributed Systems

Process Flow!

Workers
only do
their task
then report
back



DATA



D1

D2

D3

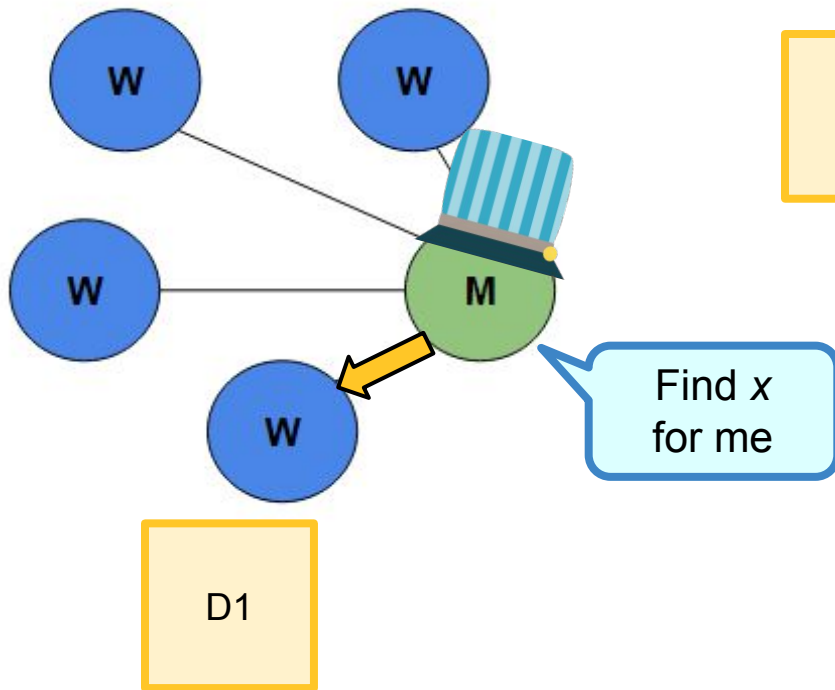
D4

SER 321

Distributed Systems

Process Flow!

Workers
only do
their task
then report
back



DATA



D1

D2

D3

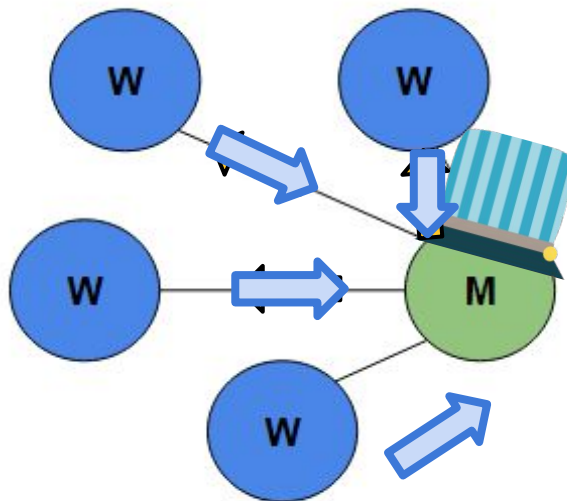
D4

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

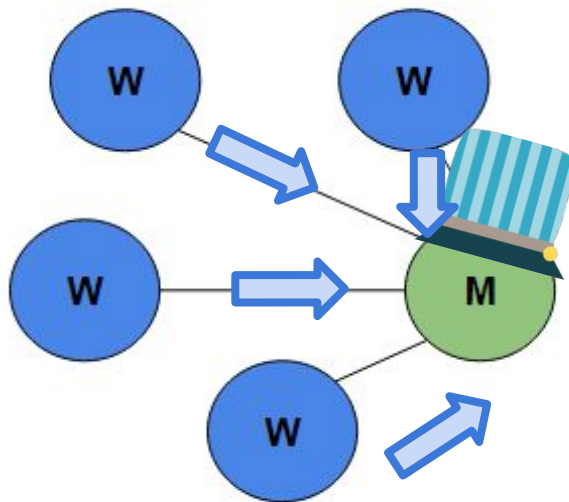
D4
Result

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



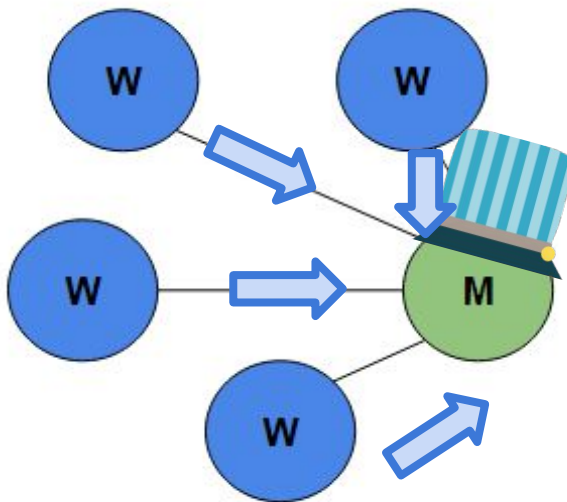
RESULTS

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?



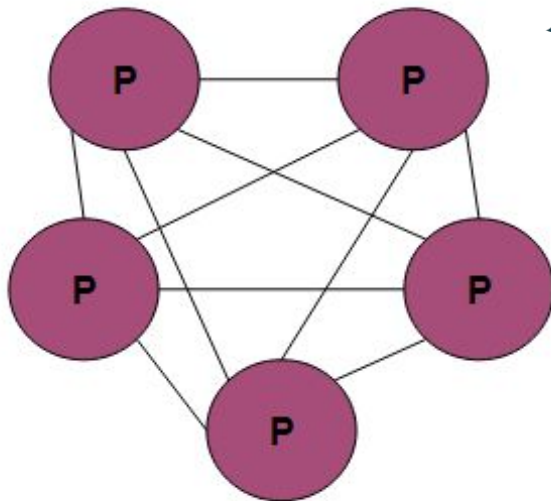
SER 321

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

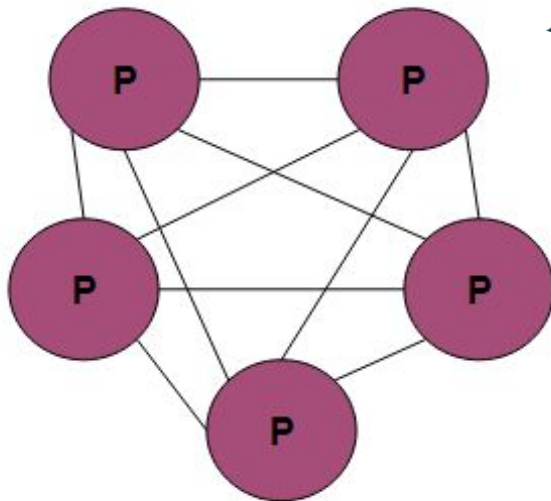
SER 321

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

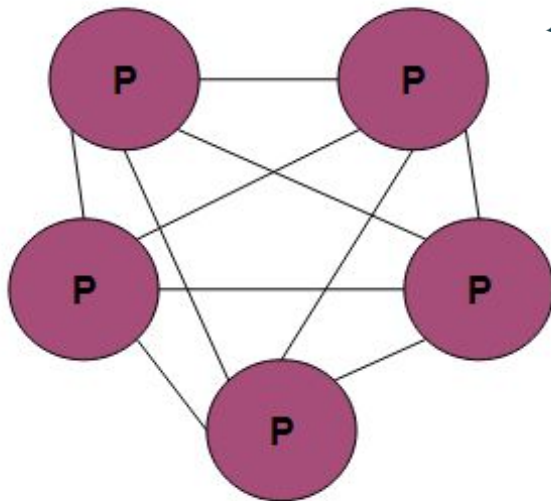
SER 321

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



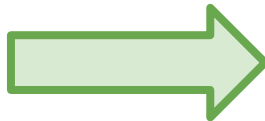
Who's in charge or keeping the beat

Result Verification



Check your work with a neighbor

Log Replication



Verify and maintain my copy of the data

Node Validation



Do I want to let you into my network

SER 321

Consensus

Match the Consensus Algorithm to its Description!

2-Phase Commit

Blockchain

Proof of Work

RAFT

If you solve this
resource-intensive problem, you
may make a request

Leader Election and Log
Replication coordinate
transactions

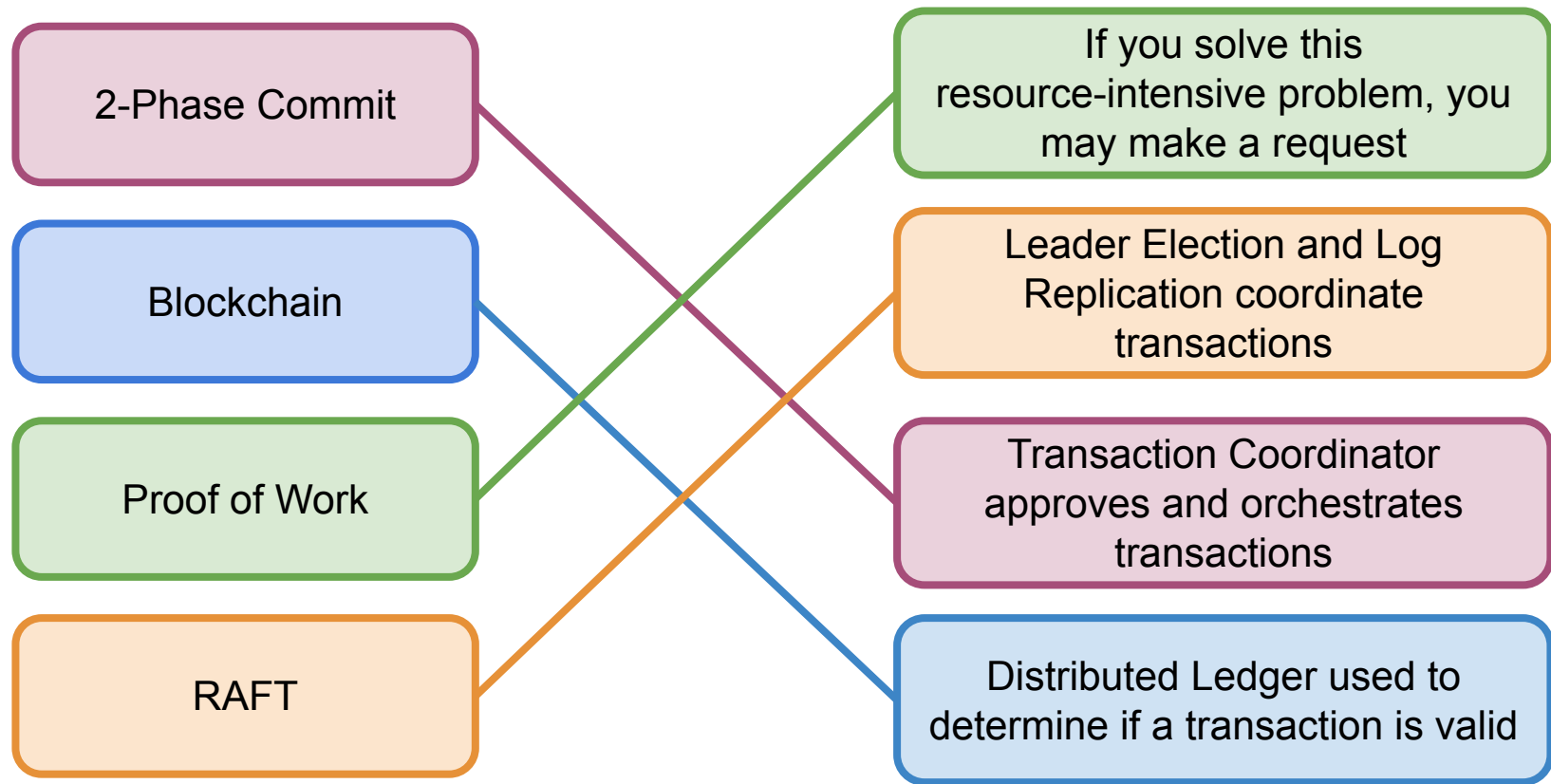
Transaction Coordinator
approves and orchestrates
transactions

Distributed Ledger used to
determine if a transaction is valid

SER 321

Consensus

Match the Consensus Algorithm to its Description!



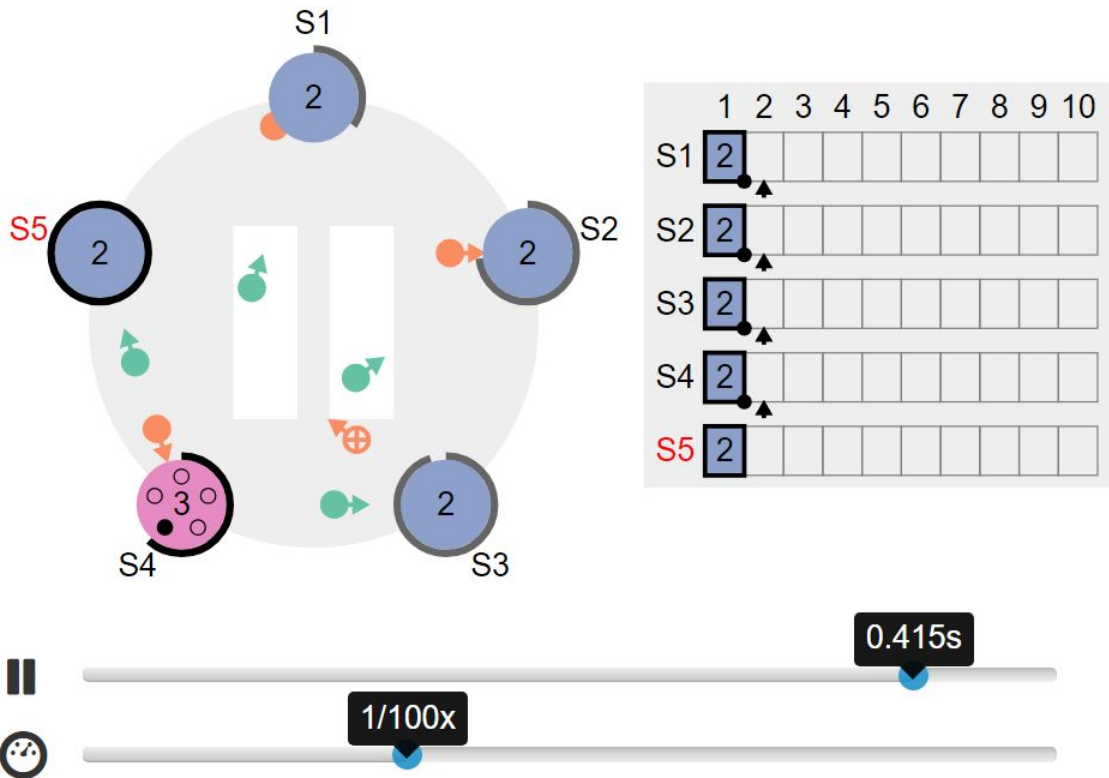
SER 321

RAFT

RAFT is a
great
consensus
example!

Leader Election

Log Replication



[The Secret Lives of Data](#) is a different visualization of Raft. It's more guided and less interactive, so it may be a gentler starting point.

SER 321

RAFT

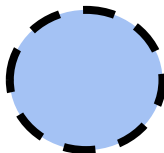
Leader Election

Nodes have 3 states:

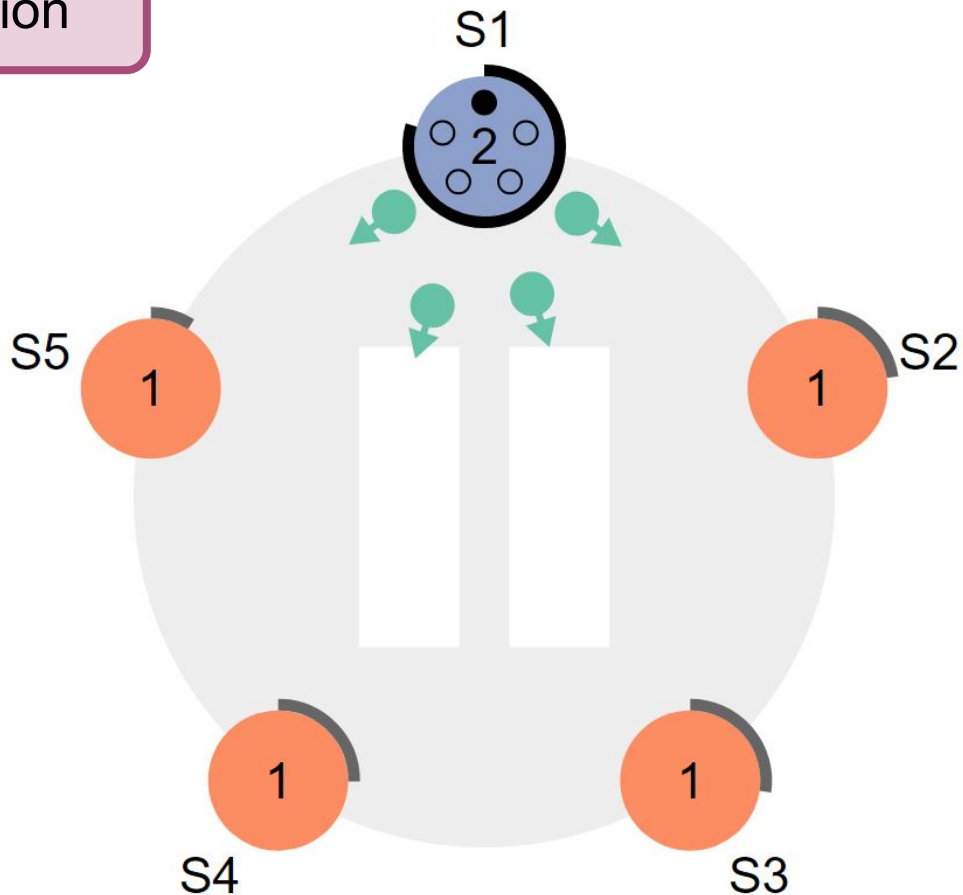
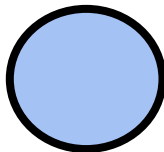
Follower



Candidate



Leader

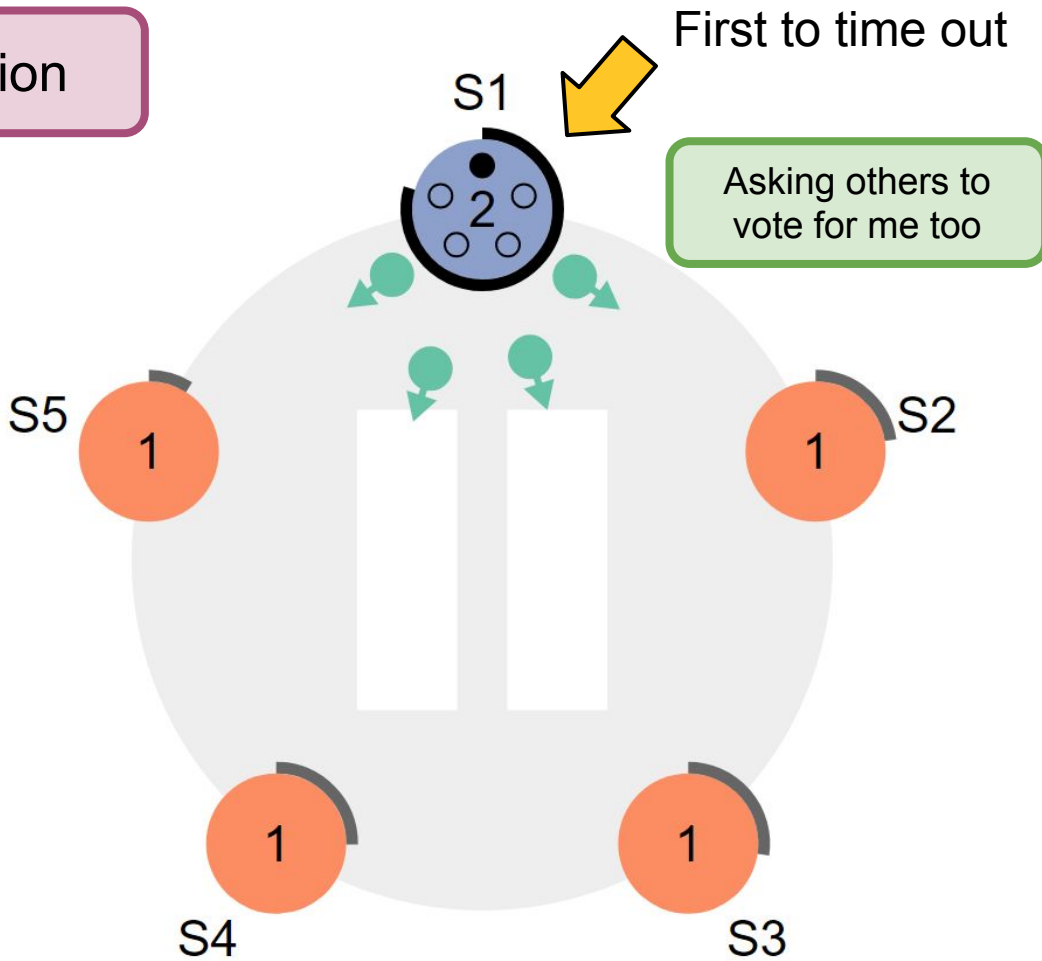


SER 321

RAFT

Leader Election

This is the first election

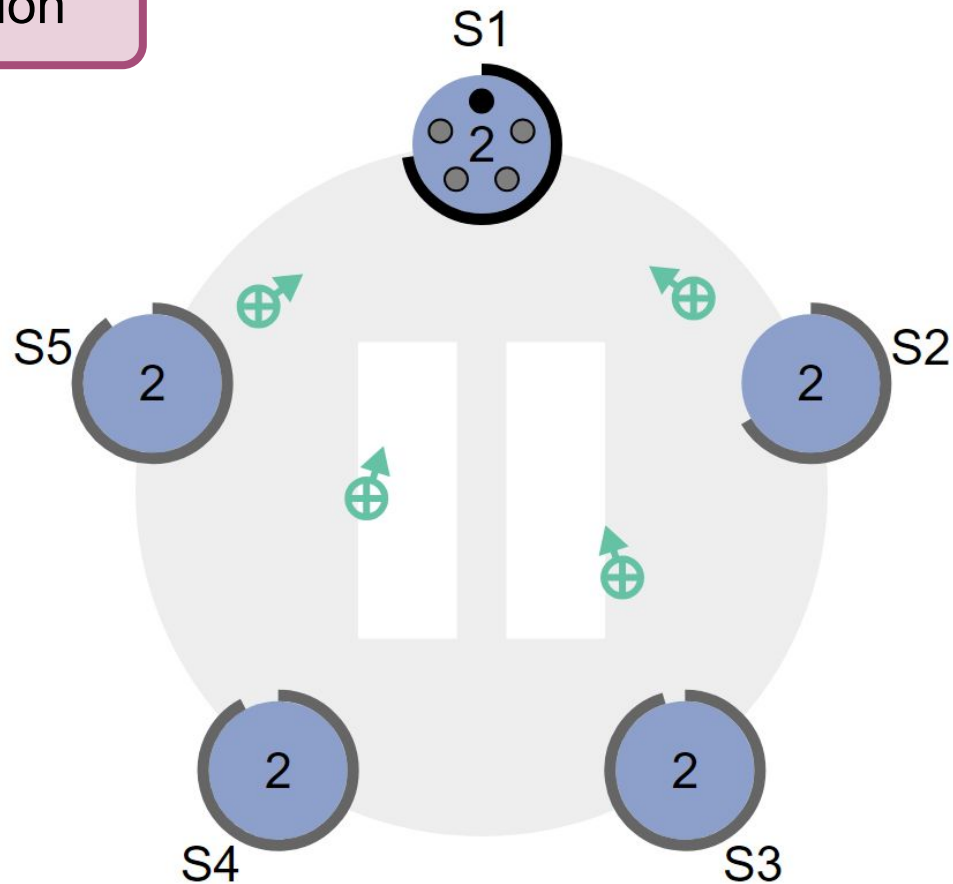


SER 321

RAFT

Leader Election

Other nodes said
sure whatever

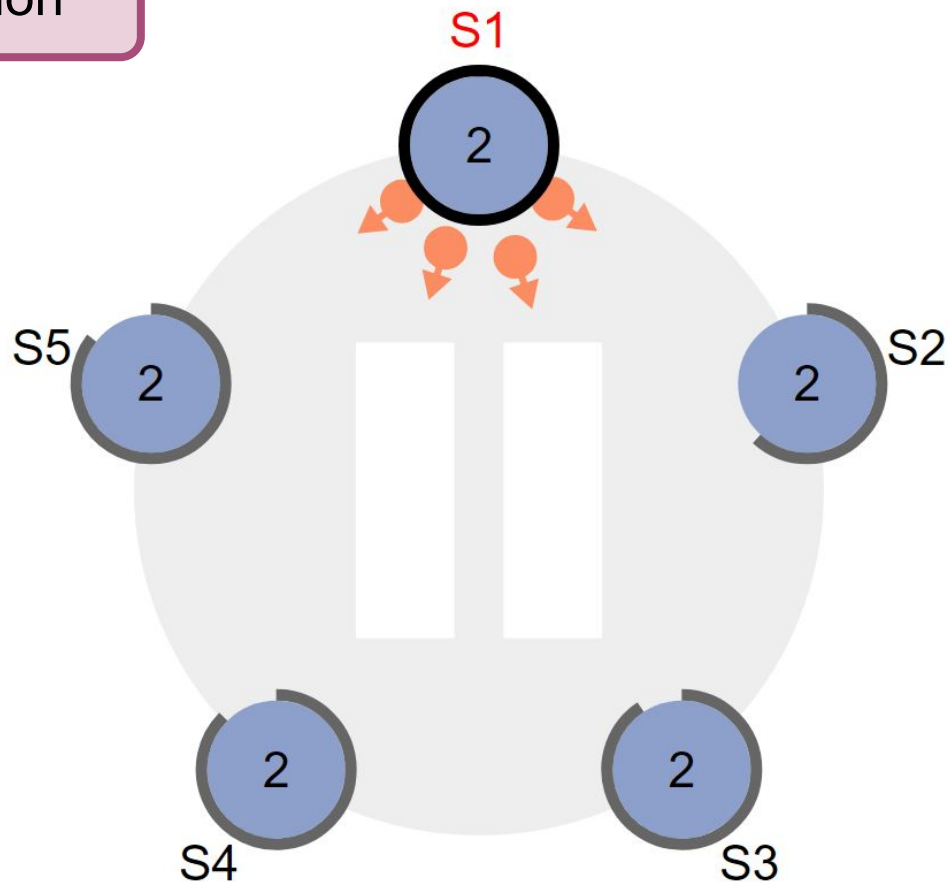


SER 321

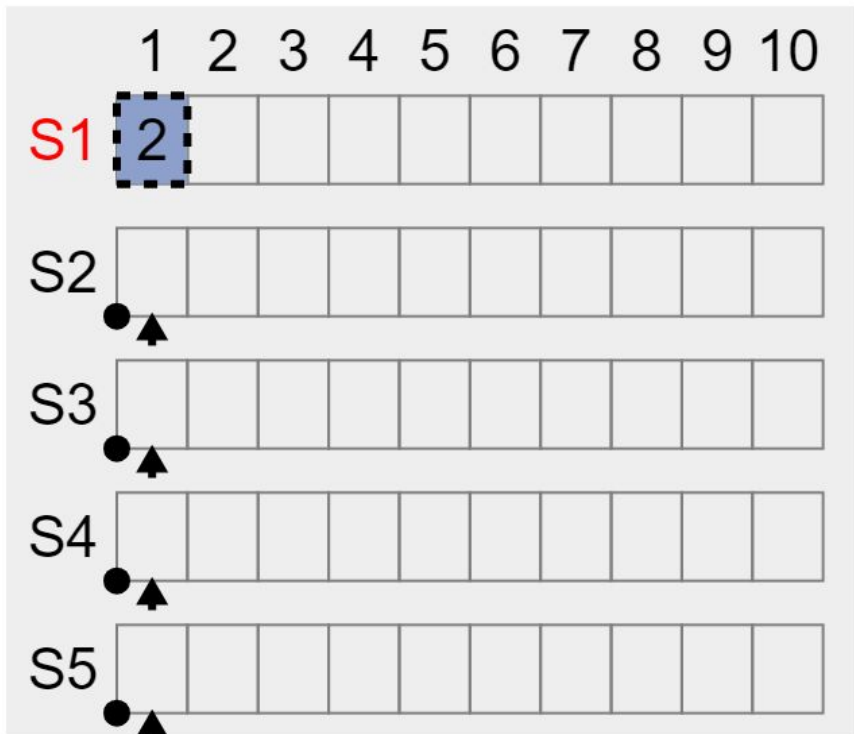

RAFT

Leader Election

Now confirmed
as Leader



Same Pattern!



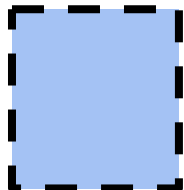
SER 321

RAFT

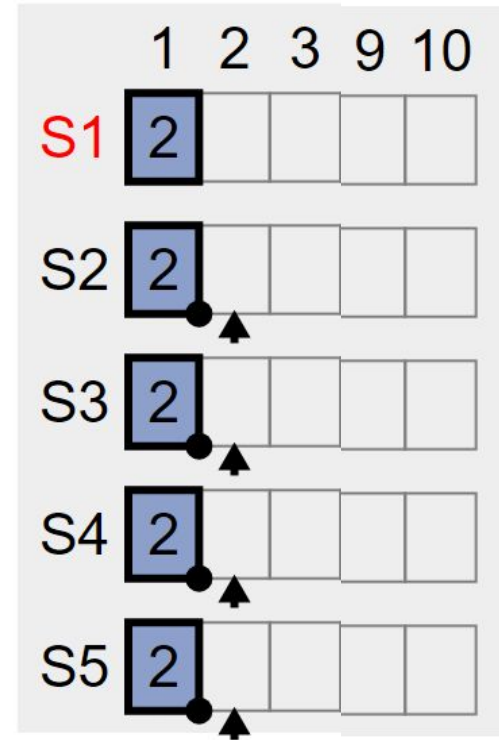
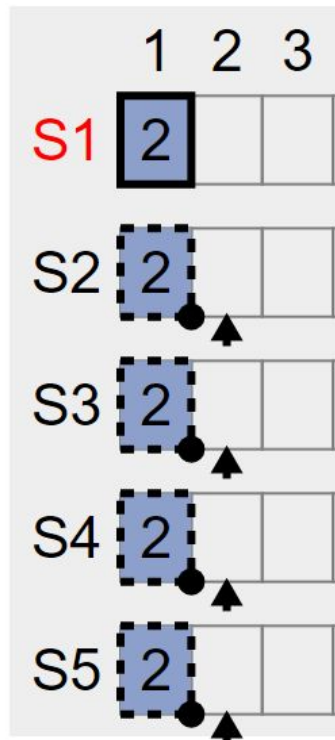
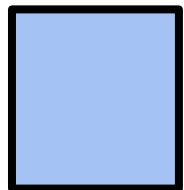
Log Replication

Same Pattern!

Candidate



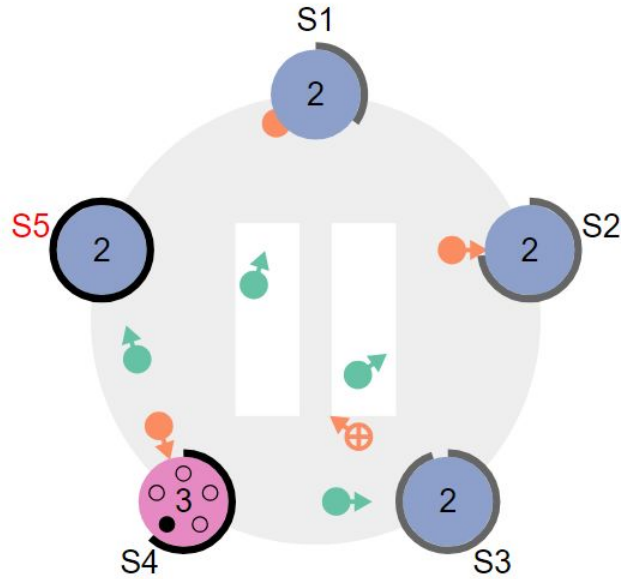
Added



SER 321

RAFT

RAFT



	1	2	3	4	5	6	7	8	9	10
S1	2									
S2	2									
S3	2									
S4	2									
S5	2									



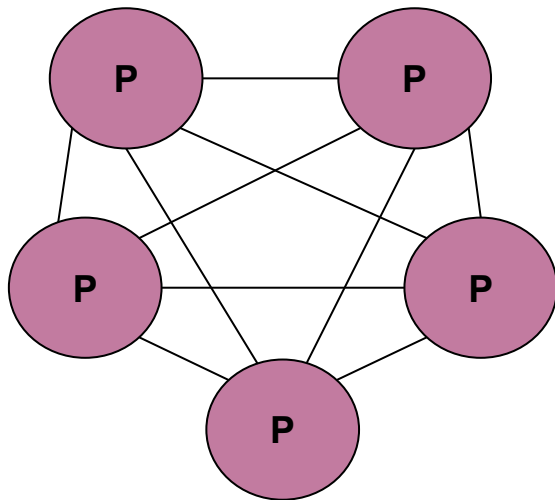
1/100x

0.415s

SER 321

Communication

How do we handle the **client** in a Peer to Peer system?



Request is sent/forwarded
to the *current leader*

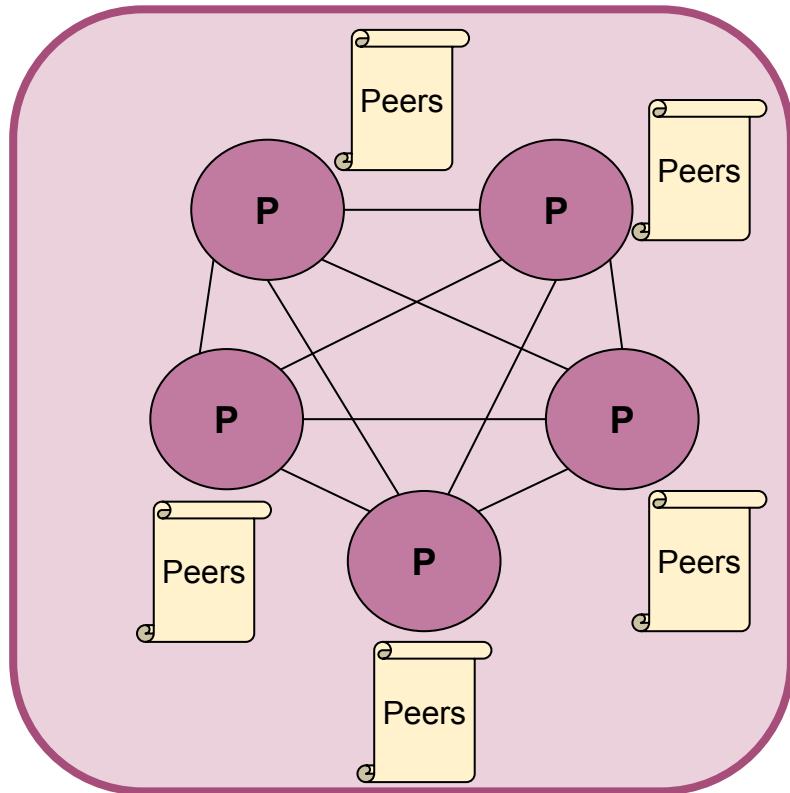
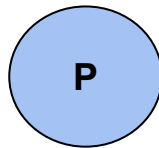
or

Peer that received the
request *acts as the leader*

SER 321

Communication

What about *adding* a Peer to the Cluster?



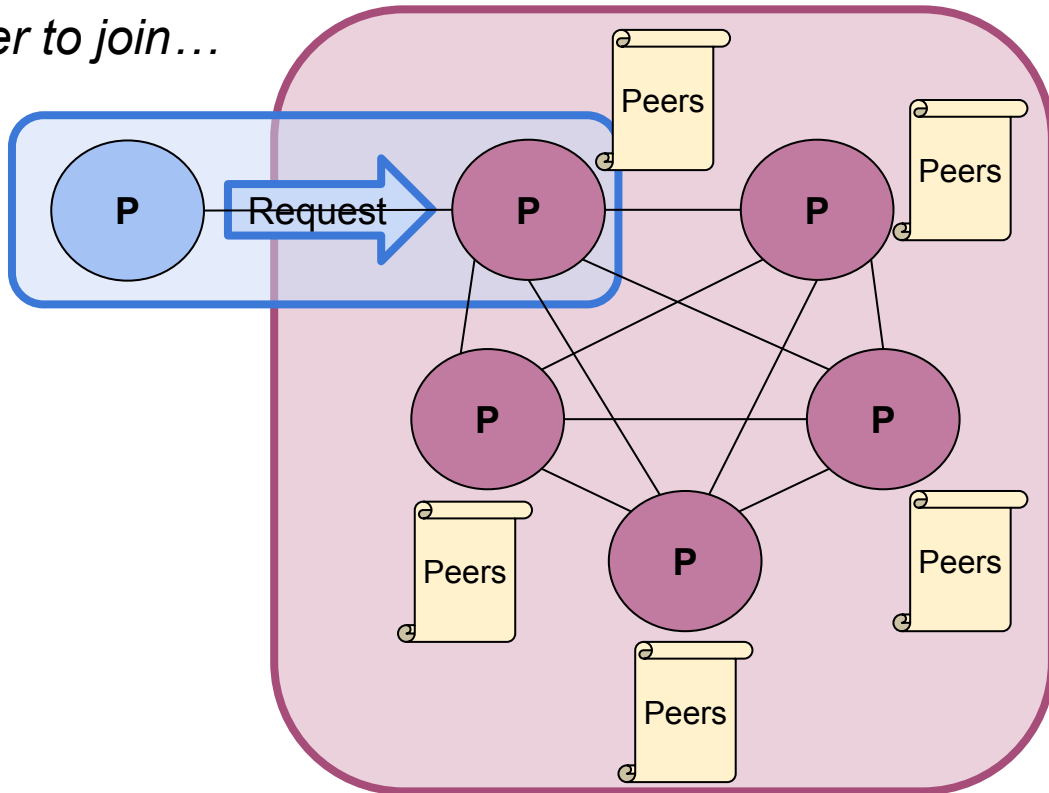
SER 321

Communication

What about **adding** a Peer to the Cluster?

Assuming we want to allow the peer to join...

Is that all?



SER 321

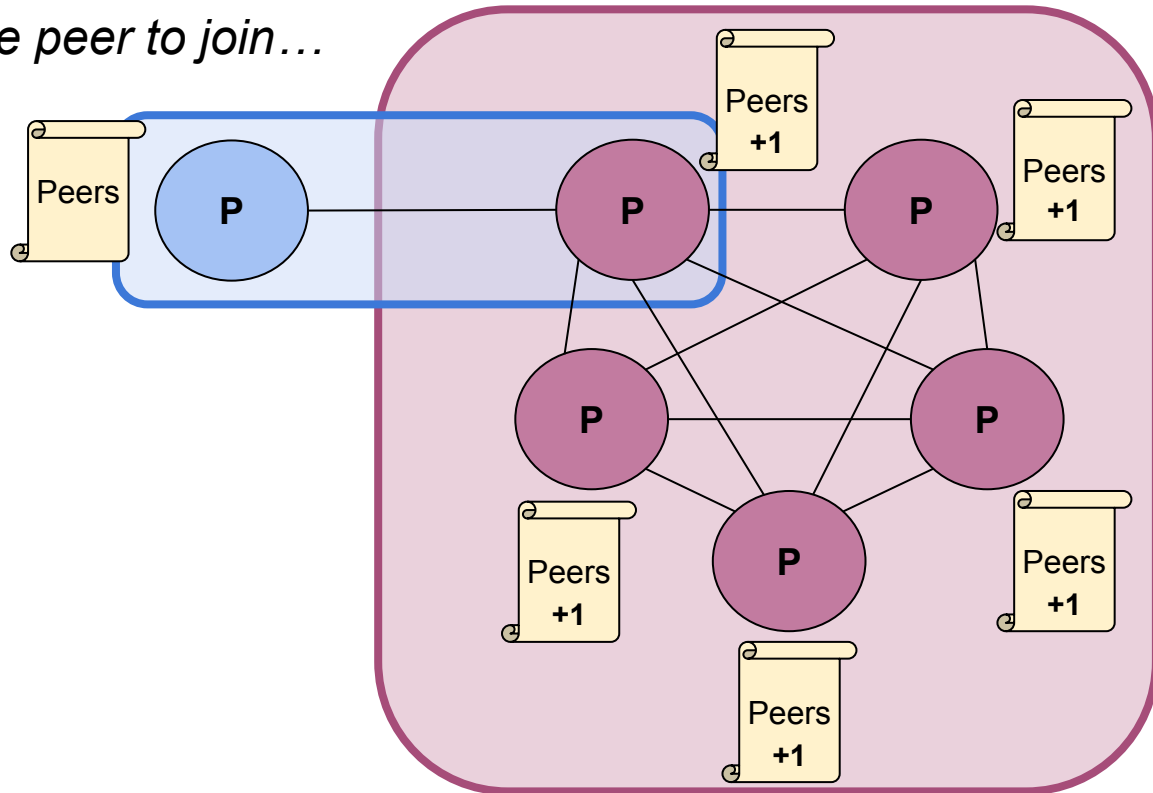
Communication

What about **adding** a Peer to the Cluster?

Assuming we want to allow the peer to join...

Three Additional Steps:

- 1.
- 2.
- 3.





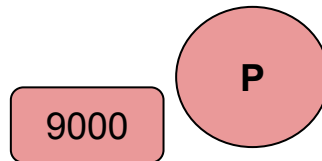
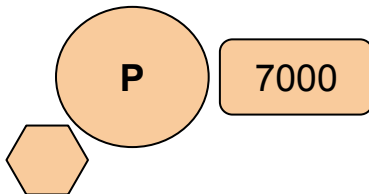
SER 321

Communication

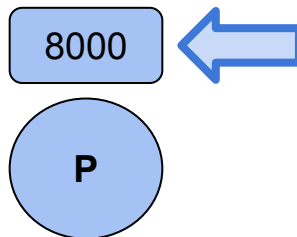
Remember that the OS allocates a new port for the client socket!

Run With:

```
gradle runPeer --args "Name Port"
```



We are going to take a closer look at the code in a moment!

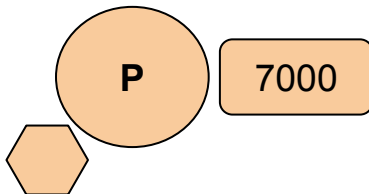


```
gradle runPeer --args "Peer8000 8000"
```

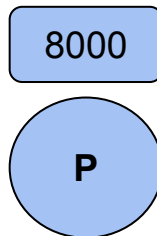
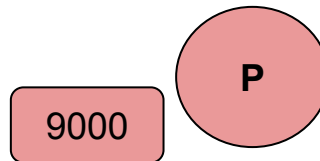
SER 321

Communication

```
gradle runPeer --args "Peer7000 7000"
```



```
> Task :runPeer
Hello Peer7000 and welcome! Your port will be 7000
> Who do you want to listen to? Enter host:port
<=====--> 75% EXECUTING [21s]
> :runPeer
█
```



```
> Task :runPeer
Hello Peer8000 and welcome! Your port will be 8000
> Who do you want to listen to? Enter host:port
<=====--> 75% EXECUTING [21s]
> :runPeer
█
```

```
gradle runPeer --args "Peer8000 8000"
```

SER 321

Communication

```
gradle runPeer --args "Peer7000 7000"
```

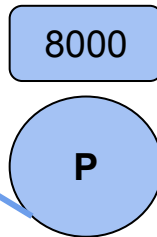
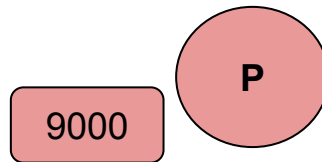
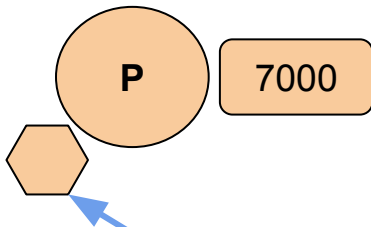
```
> Task :runPeer
```

```
Hello Peer7000 and welcome! Your port will be 7000
```

```
> Who do you want to listen to? Enter host:port
```

```
<=====--> 75% EXECUTING [21s]
```

```
> :runPeer
```



```
> Task :runPeer
```

```
Hello Peer8000 and welcome! Your port will be 8000
```

```
> Who do you want to listen to? Enter host:port
```

```
<<==<=<=<=<=<=====----> 75% EXECUTING [1m 56s]
```

```
> You can now start chatting (exit to exit)
```

```
<=====--> 75% EXECUTING [2m 3s]
```

```
> :runPeer
```

SER 321 Communication

What will
happen?

```
> Task :runPeer
Hello Peer7000 and welcome! Your port will be 7000
> Who do you want to listen to? Enter host:port
<===== > 75% EXECUTING [21s]
> :runPeer
█
```

```
> Task :runPeer
Hello Peer8000 and welcome! Your port will be 8000
> Who do you want to listen to? Enter host:port
<===== > 75% EXECUTING [1m 56s]
> You can now start chatting (exit to exit)
<===== > 75% EXECUTING [3m 33s]
<===== > 75% EXECUTING [3m 37s]
hi 7000
```

8000

Why?

9000

P

8000

P

PS C:\ASU\SER321\examples_repo\ser321examples\Sockets\S
Starting a Gradle Daemon, 1 busy and 1 stopped Daemons

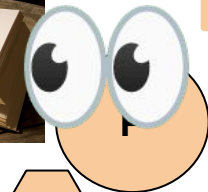
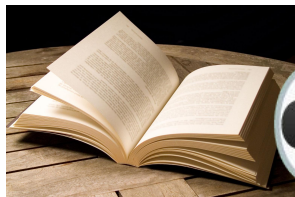
```
> Task :runPeer
Hello Peer7000 and welcome! Your port will be 7000
> Who do you want to listen to? Enter host:port
<===== > 75% EXECUTING [2m 48s]
> :runPeer
█
```

7000

```
> Task :runPeer
Hello Peer8000 and welcome! Your port will be 8000
> Who do you want to listen to? Enter host:port
<===== > 75% EXECUTING [1m 56s]
> You can now start chatting (exit to exit)
<===== > 75% EXECUTING [3m 13s]
> :runPeer
hi 7000
```

SER 321

Communication



7000

Telling Peer7000 about Peer8000

8000

P

HEY DUDE,
LISTEN TO THIS!

```
> Task :runPeer
Hello Peer7000 and welcome! Your port will be 7000
> Who do you want to listen to? Enter host:port
<====<==<==<=====----> 75% EXECUTING [3m 22s]
> :runPeer
localhost:8000
```

```
> Task :runPeer
Hello Peer8000 and welcome! Your port will be 8000
> Who do you want to listen to? Enter host:port
<====<==<==<=====----> 75% EXECUTING [1m 56s]
> You can now start chatting (exit to exit)
<=====----> 75% EXECUTING [2m 3s]
> :runPeer
```

Communication

```
> Task :runPeer  
Hello Peer7000 at 18:26:39  
> Who do you want to connect to?  
  
<<==<==<==<==<  
> :runPeer  
localhost:8000
```

```
> Task :runPeer
Hello Peer8000 and welcome! Your port will be 8000
> Who do you want to listen to? Enter host:port
<====<<====<=<=<=====----> 75% EXECUTING [3m 4s]
> You can now start chatting (exit to exit)
[Peer7000]: Hi Peer8000!
<=====----> 75% EXECUTING [4m 4s]
```

```
> :runPeer
```

8000

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sockets\
```

```
> Task :runPeer  
Hello Peer7000 and welcome! Your port will be 7000  
> Who do you want to listen to? Enter host:port  
<<=====-----> 75% EXECUTING [3m 43s]  
> You can now start chatting (exit to exit)  
<<<=====-----> 75% EXECUTING [3m 58s]  
<=====-----> 75% EXECUTING [4m 1s]  
Hi Peer8000!
```

7000

7000

Let's take a closer look at the Code!


```
public class ServerThread extends Thread {  
    2 usages  
    private ServerSocket serverSocket;  
    2 usages  
    private Set<Socket> listeningSockets = new HashSet<>();  
  
    1 usage  
    public ServerThread(String portNum) throws IOException {  
        serverSocket = new ServerSocket(Integer.valueOf(portNum));  
    }  
}
```

ServerThread

```
    Starting the thread, we are waiting for clients wanting to connect to the server socket in a list  
    public void run() {  
        try {  
            while (true) {  
                Socket sock = serverSocket.accept();  
                listeningSockets.add(sock);  
            }  
        } catch (Exception e) { ... }  
    }  
}
```



```
    Sending the message to the OutputStream for each socket that we saved  
    1 usage  
    void sendMessage(String message) {  
        try {  
            for (Socket s : listeningSockets) {  
                PrintWriter out = new PrintWriter(s.getOutputStream(), true);  
                out.println(message);  
            }  
        } catch (Exception e) { ... }  
    }  
}
```

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

```
    BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));  
    String username = args[0];  
    System.out.println("Hello " + username + " and welcome! Your port will be " + args[1]);
```

```
    // starting the Server Thread, which waits for other peers to want to connect
```

```
    ServerThread serverThread = new ServerThread(args[1]);
```

```
    serverThread.start();
```

```
    Peer peer = new Peer(bufferedReader, args[0], serverThread);
```

```
    peer.updateListenToPeers();
```

Peer

```
public class ClientThread extends Thread {
```

```
    2 usages  
    private BufferedReader bufferedReader;
```

```
    1 usage
```

```
    public ClientThread(Socket socket) throws IOException {  
        bufferedReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
    }  
}
```

```
    public void run() {
```

```
        while (true) {
```

```
            try {
```

```
                JSONObject json = new JSONObject(bufferedReader.readLine());
```

```
                System.out.println "[" + json.getString("username") + ": " + json.getString("message") + " ]";
```

```
            } catch (Exception e) { ... }
```

```
        }  
    }  
}
```

ClientThread




```
public class ClientThread extends Thread {  
    2 usages  
    private BufferedReader bufferedReader;  
  
    1 usage  
    public ClientThread(Socket socket) throws IOException {  
        bufferedReader = new BufferedReader  
            (new InputStreamReader(socket.getInputStream()));  
    }  
    public void run() {  
        while (true) {  
            try {  
                JSONObject json =  
                    new JSONObject(bufferedReader.readLine());  
                System.out.println  
                    ("[" + json.getString("username")+"]: "  
                     + json.getString("message"));  
            } catch (Exception e) {...}  
        }  
    }  
}
```

ClientThread

```
public static void main (String[] args) throws Exception {  
  
    BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));  
    String username = args[0];  
    System.out.println("> Who do you want to listen to? Enter host:port");  
    // st  
    public void updateListenToPeers() throws Exception {  
        System.out.println("> Who do you want to listen to? Enter host:port");  
        String input = bufferedReader.readLine();  
        String[] setupValue = input.split(" ");  
        for (int i = 0; i < setupValue.length; i++) {  
            String[] address = setupValue[i].split(":");  
            Socket socket = null;  
            try {  
                socket = new Socket(address[0], Integer.valueOf(address[1]));  
                new ClientThread(socket).start();  
            } catch (Exception c) {  
                if (socket != null) {  
                    socket.close();  
                } else {  
                    System.out.println("Cannot connect, wrong input");  
                    System.out.println("Exiting: I know really user friendly");  
                    System.exit(0);  
                }  
            }  
        }  
    }  
    askForInput();  
}
```

Peer.updateListenToPeers

SER 321

Scratch Space

Upcoming Events

SI Sessions:

- Thursday, April 24th at 7:00 pm MST
- Sunday, April 27th at **6:00 pm** MST - **2 hour Exam Review Session**
- Tuesday, April 29th, at 10:00 am MST - **Q&A Session**

Review Sessions:

- Sunday, April 27th at **6:00 pm** MST - **2 hour Exam Review Session**
- Tuesday, April 29th, at 10:00 am MST - **Q&A Session**

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

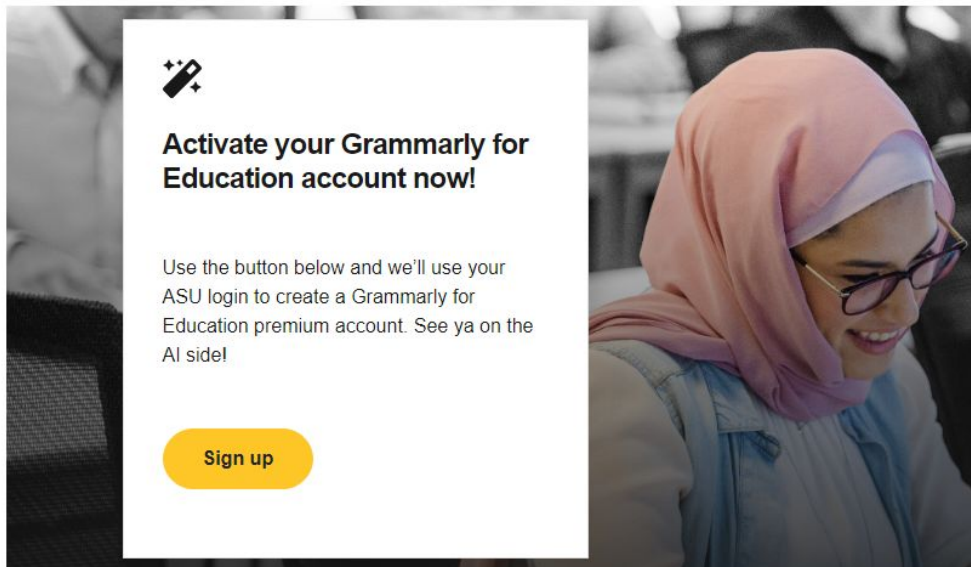
Computer Applications and Information Technology


 [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!





Activate your Grammarly for Education account now!

Use the button below and we'll use your ASU login to create a Grammarly for Education premium account. See ya on the AI side!

[Sign up](#)



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](#)
- [RAFT](#)