SER 321 B Session

SI Session

Tuesday, November 26th 2024

10:00 am - 11:00 am MST

Agenda

- Distributed Structure Communication
  - Main and Worker
  - Peer to Peer
- Middleware
  - What is it?
  - Why we care

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.

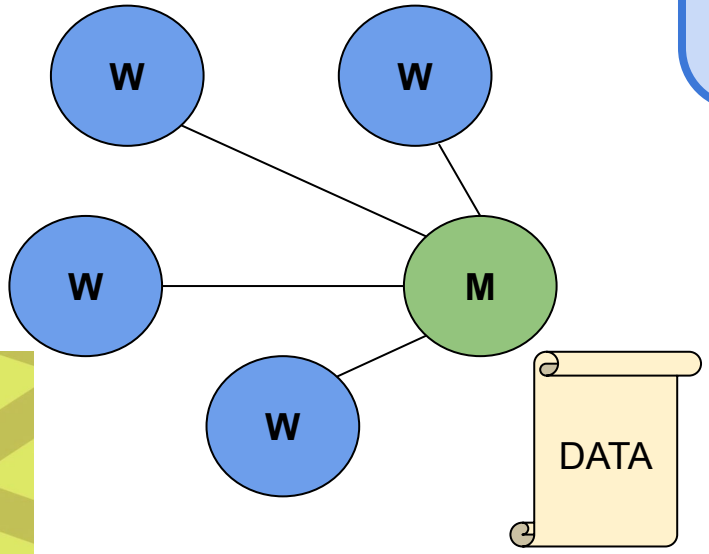
Interact with us:

Zoom Features

Zoom Chat

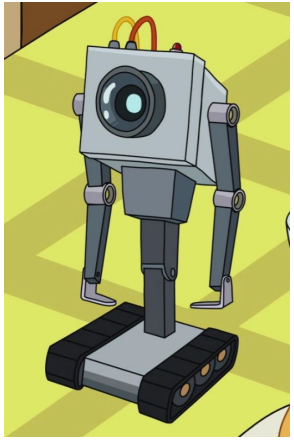
- Use the chat feature to interact with the presenter and respond to presenter's questions.
- Annotations are encouraged

SER 321
Communication



Workers only do their task then report back

Think worker bees or specialized machinery



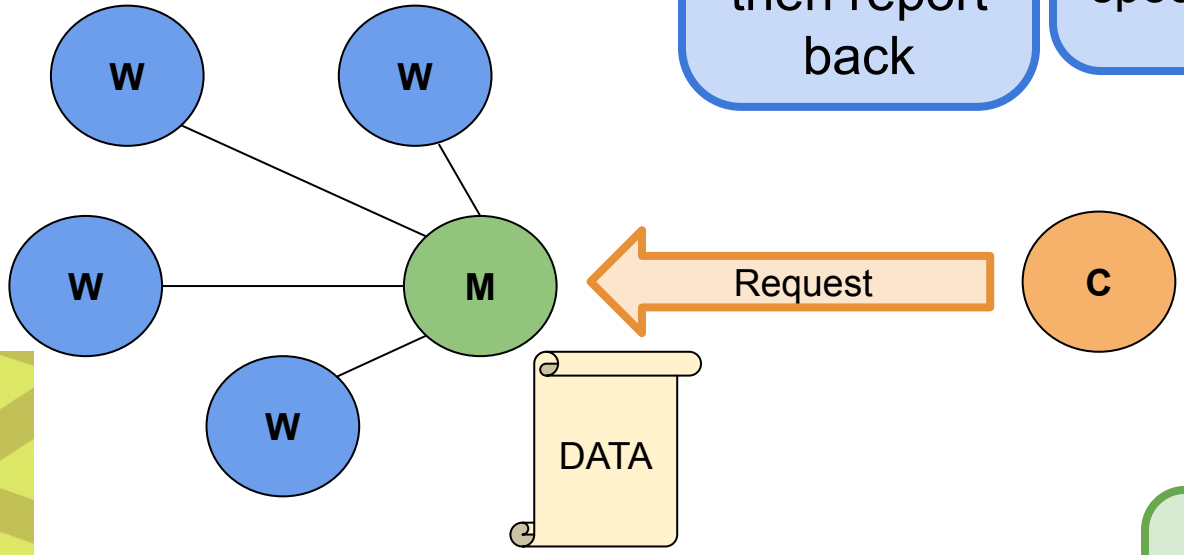
Main is in charge of everything

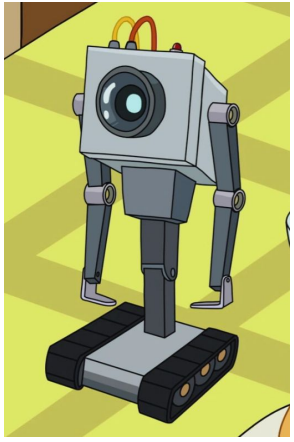
Similar to _____ in our previous implementations

SER 321
Communication

Workers only do their task then report back

Think worker bees or specialized machinery

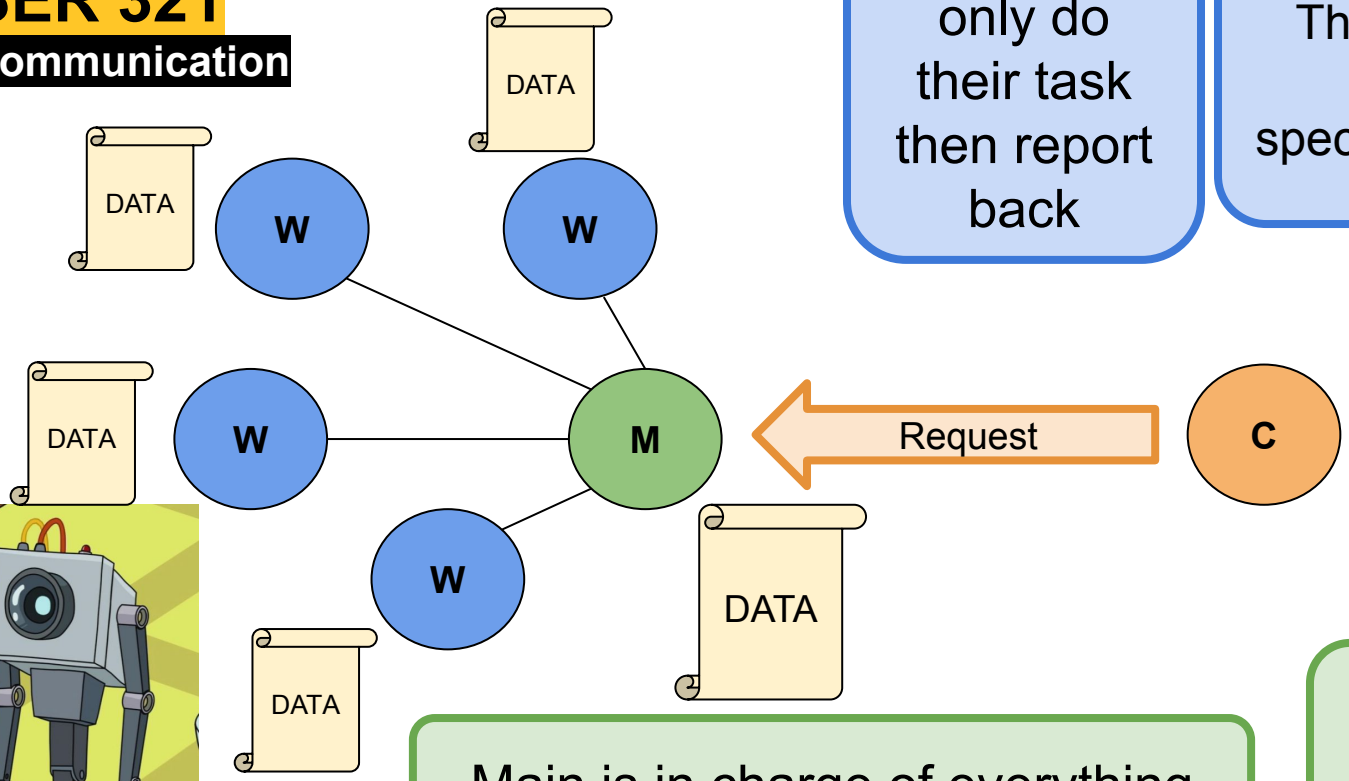


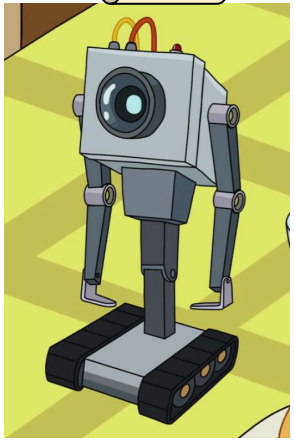


Main is in charge of everything

Similar to _____ in our previous implementations

SER 321
Communication





Workers only do their task then report back

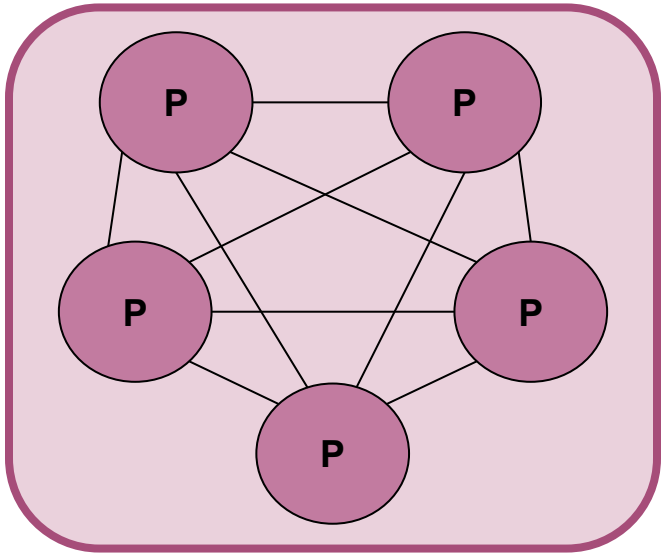
Think worker bees or specialized machinery

Main is in charge of everything

Similar to _____ in our previous implementations

SER 321
Communication

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



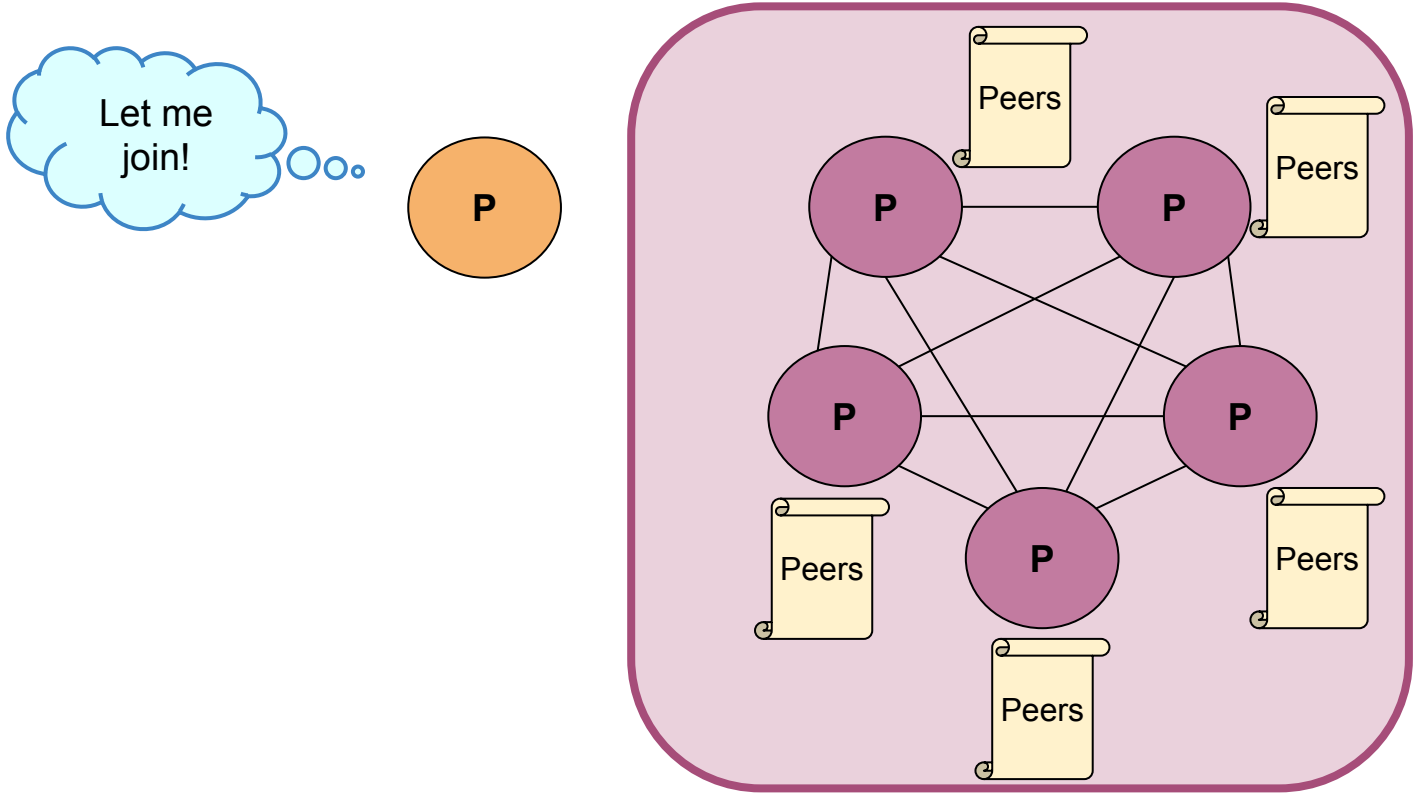
Request is sent to the *current leader*

or

Peer that received the request *acts as the leader*

SER 321
Communication

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

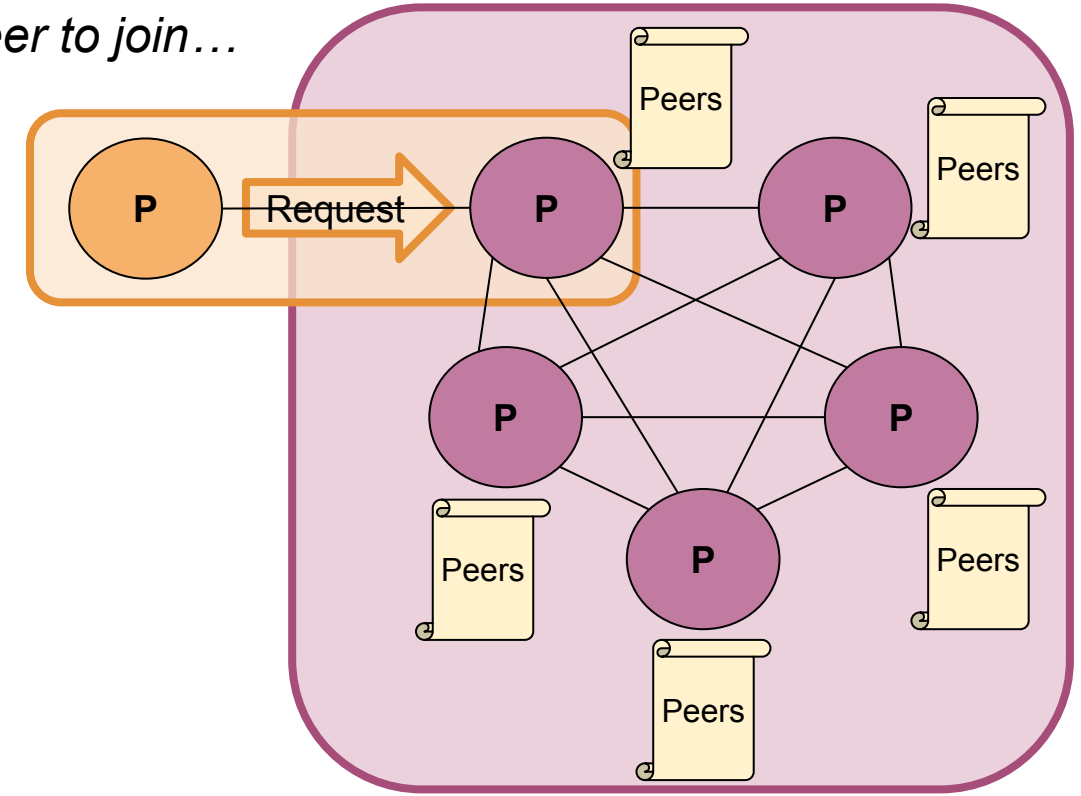


SER 321
Communication

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

Assuming we want to allow the peer to join…

Is that all?



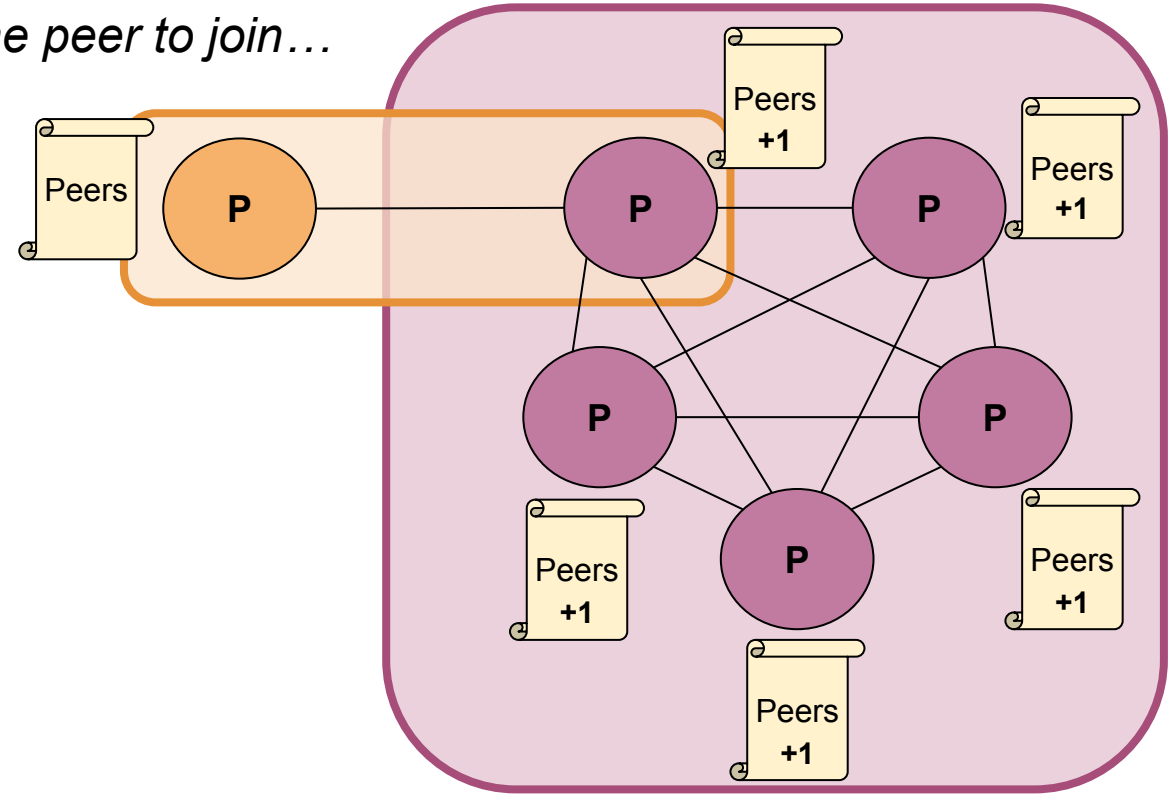
SER 321
Communication

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

Assuming we want to allow the peer to join…

Three Additional Steps:

1.
2.
3.



SimplePeerToPeer

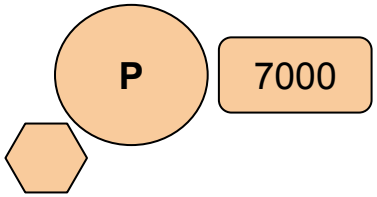
SER 321

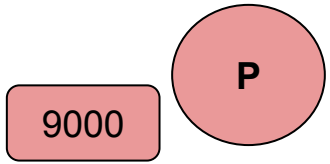
Communication

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

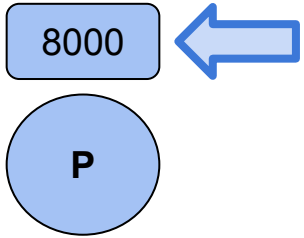
Run With:

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





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



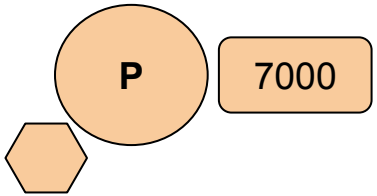

```
gradle runPeer --args "Peer8000 8000"
```

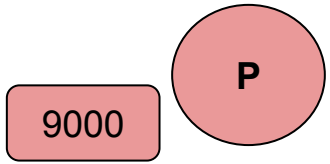
SimplePeerToPeer

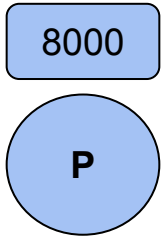
SER 321
Communication

```
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
```








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

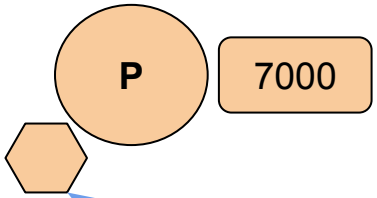
```
> Task :runPeer
Hello Peer8000 and welcome! Your port will be 8000
> Who do you want to listen to? Enter host:port
<==========----> 75% EXECUTING [21s]
> :runPeer
```

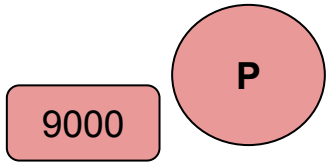
SimplePeerToPeer

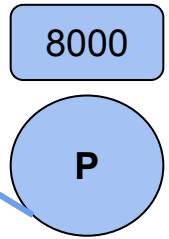
SER 321
Communication

```
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
```

SimplePeerToPeer

SER 321
Communication

What will happen?

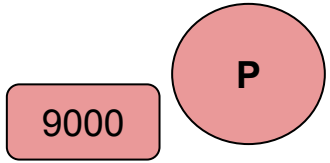
```
> Task :runPeer
Hello Peer7000 and welcome! Your port will be 7000
> Who do you want to listen to? Enter host:port
<==========----> 75% EXECUTING [21s]
> :runPeer
```

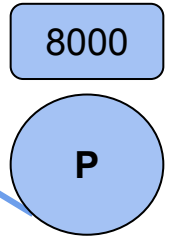
```
> Task :runPeer
Hello Peer8000 and welcome! Your port will be 8000
> Who do you want to listen to? Enter host:port
<<===<==<<==<=<==========----> 75% EXECUTING [1m 56s]
> You can now start chatting (exit to exit)
<<<==<=<==========----> 75% EXECUTING [3m 33s]
<==========----> 75% EXECUTING [3m 37s]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

```
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
```

Why?



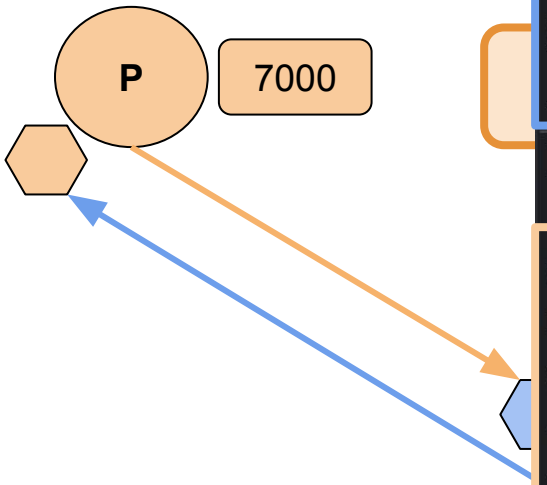



```
> Task :runPeer
Hello Peer8000 and welcome! Your port will be 8000
> Who do you want to listen to? Enter host:port
<<===<==<<==<=<==========----> 75% EXECUTING [1m 56s]
> You can now start chatting (exit to exit)
<<<==<=<==========----> 75% EXECUTING [3m 13s]
> :runPeer
hi 7000
```

SER 321
Communication

```
> Task :runPeer
Hello Peer7000 a
> Who do you wan
<=<====<==<===<=
> :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
```

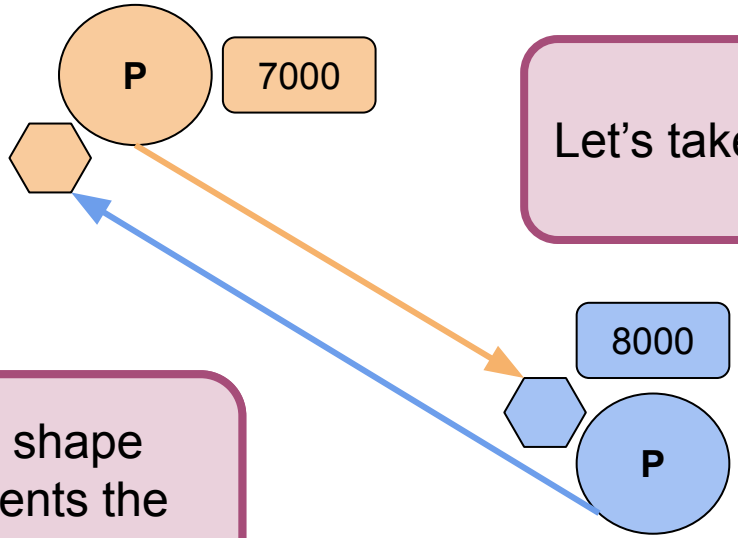



```
PS C:\ASU\SER321\examples_repo\ser321examples\Sockets\
```

```
> 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!
```

SER 321
Communication

```
> 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
```



Let's take a closer look at the Code!

What shape represents the **ClientThread?**

```
> 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
```

SimplePeerToPeer

SER 321
Communication

```java
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();
```

ServerThread

Peer

```java
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));
    }

    Starting the thread, we are waiting for clients wanting to talk to us, then save the 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) {...}
    }
}
```

ClientThread

```java
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) {...}
        }
    }
}
```

SimplePeerToPeer

SER 321
Communication

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

    BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));
    Strin
    Syste

    // st
```

```
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 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
Middleware

We have been:

- Serializing Messages
- Sending Messages
- Parsing Messages
- Handle Messages

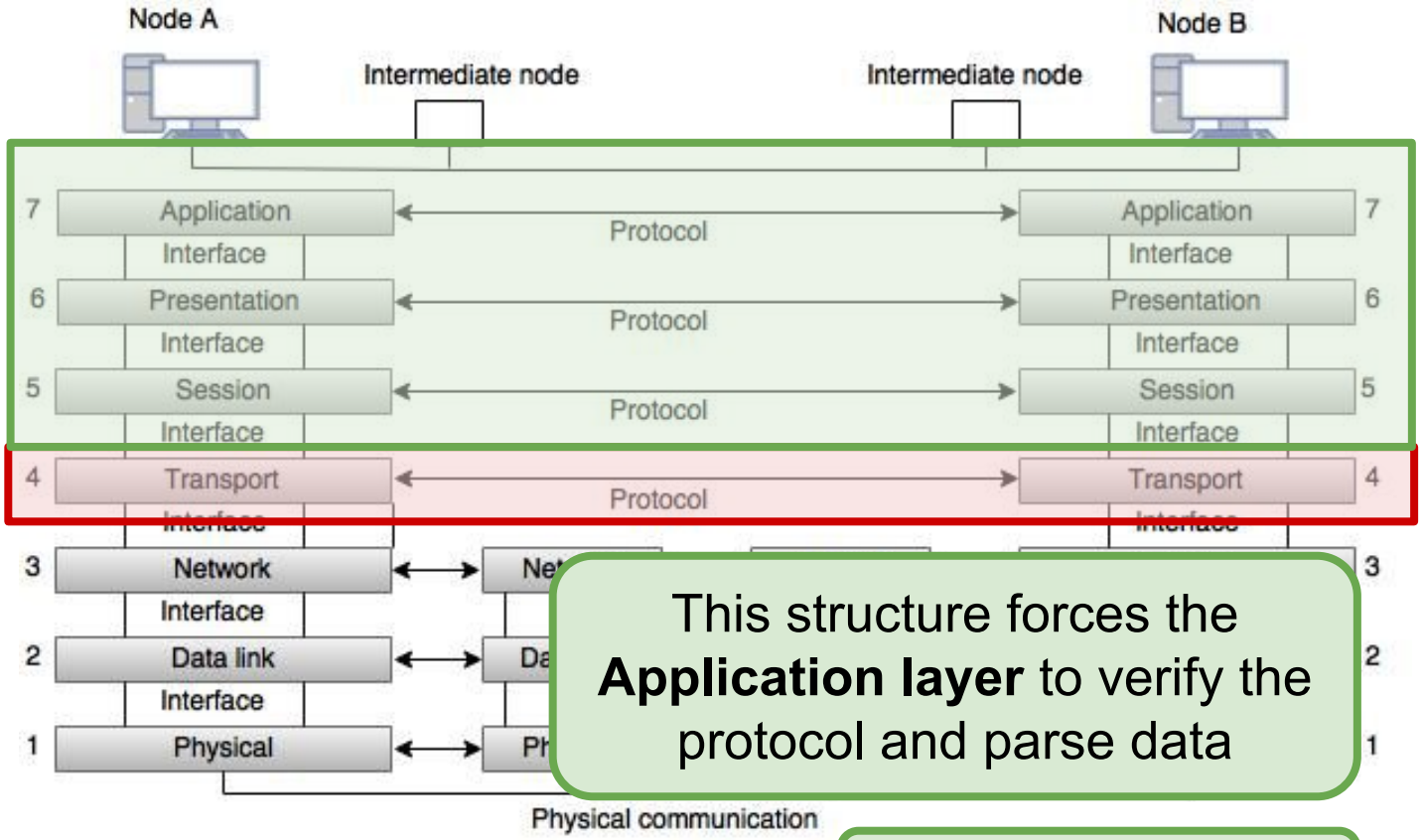


Fig: OSI Model

This structure forces the **Application layer** to verify the protocol and parse data

Not really its job…

SER 321
Middleware

With Middleware:

- Serializing Messages
- Sending Messages
- Parsing Messages
- Handle Messages

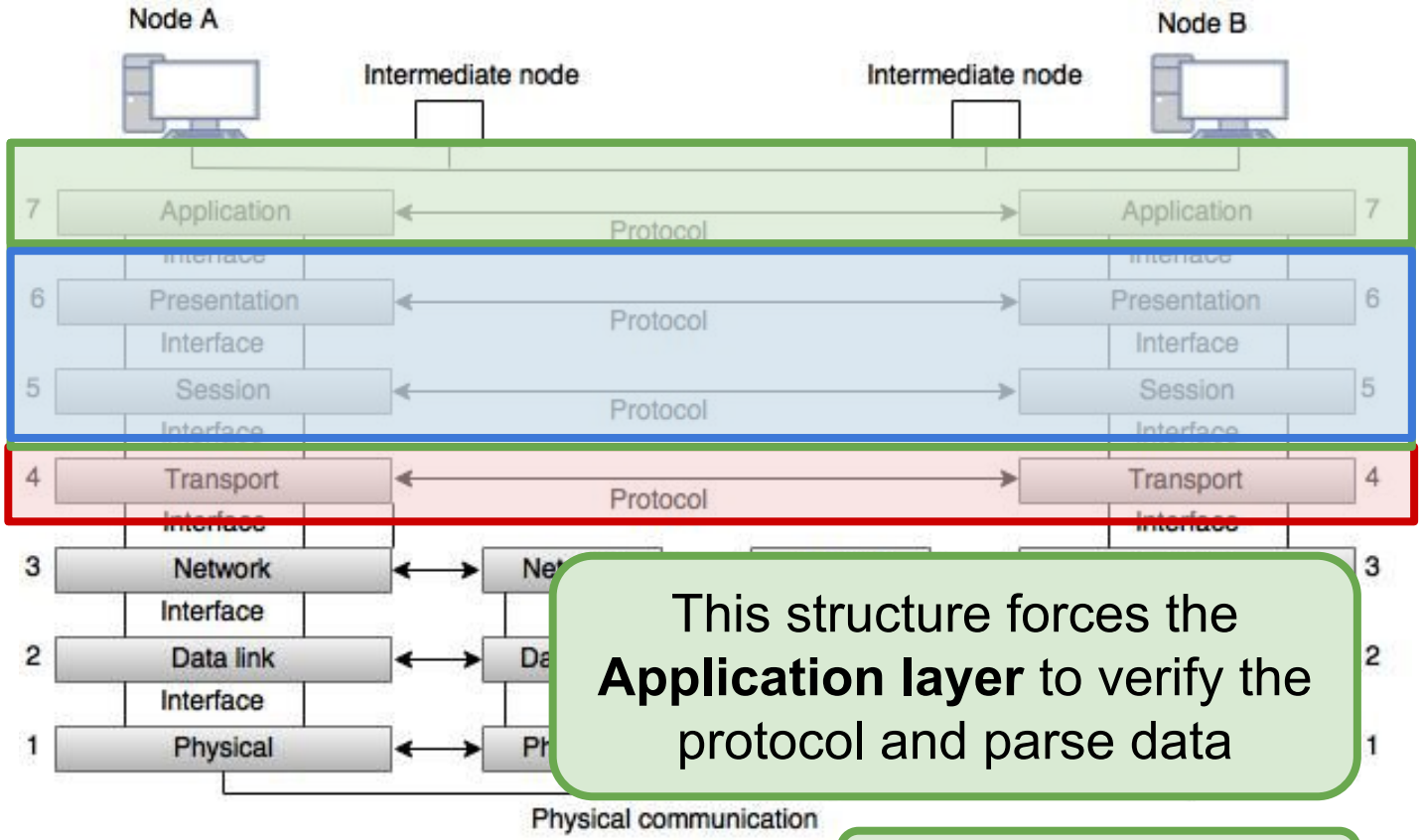


Fig: OSI Model

This structure forces the **Application layer** to verify the protocol and parse data

Not really its job…

SER 321
Middleware

With Middleware:

Serializing Messages

Sending Messages

Parsing Messages

Handle Messages

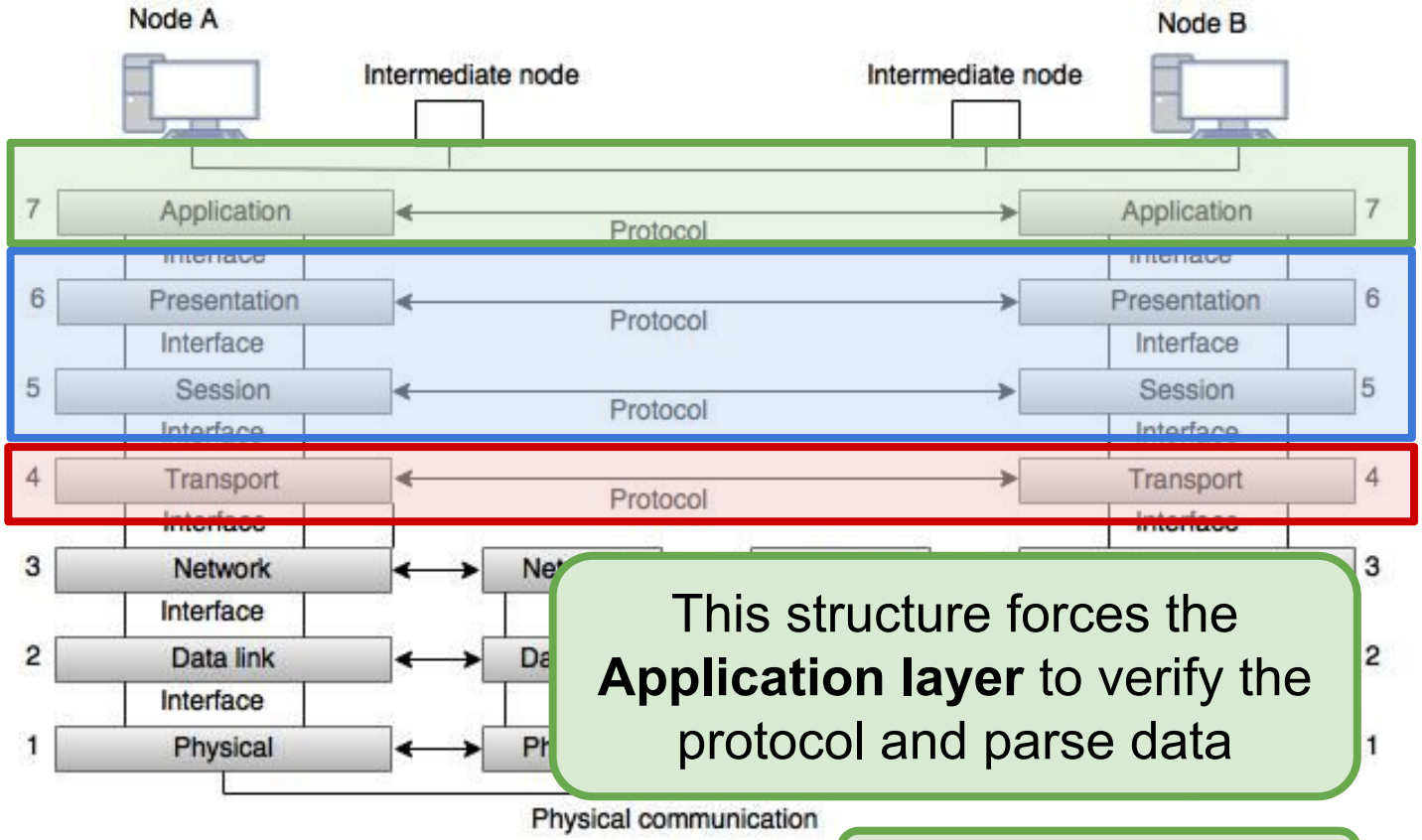


Fig: OSI Model

This structure forces the **Application layer** to verify the protocol and parse data

Not really its job…

SER 321
Middleware

Middleware:

Session Layer Responsibilities:

Authentication

Authorization

Session Management

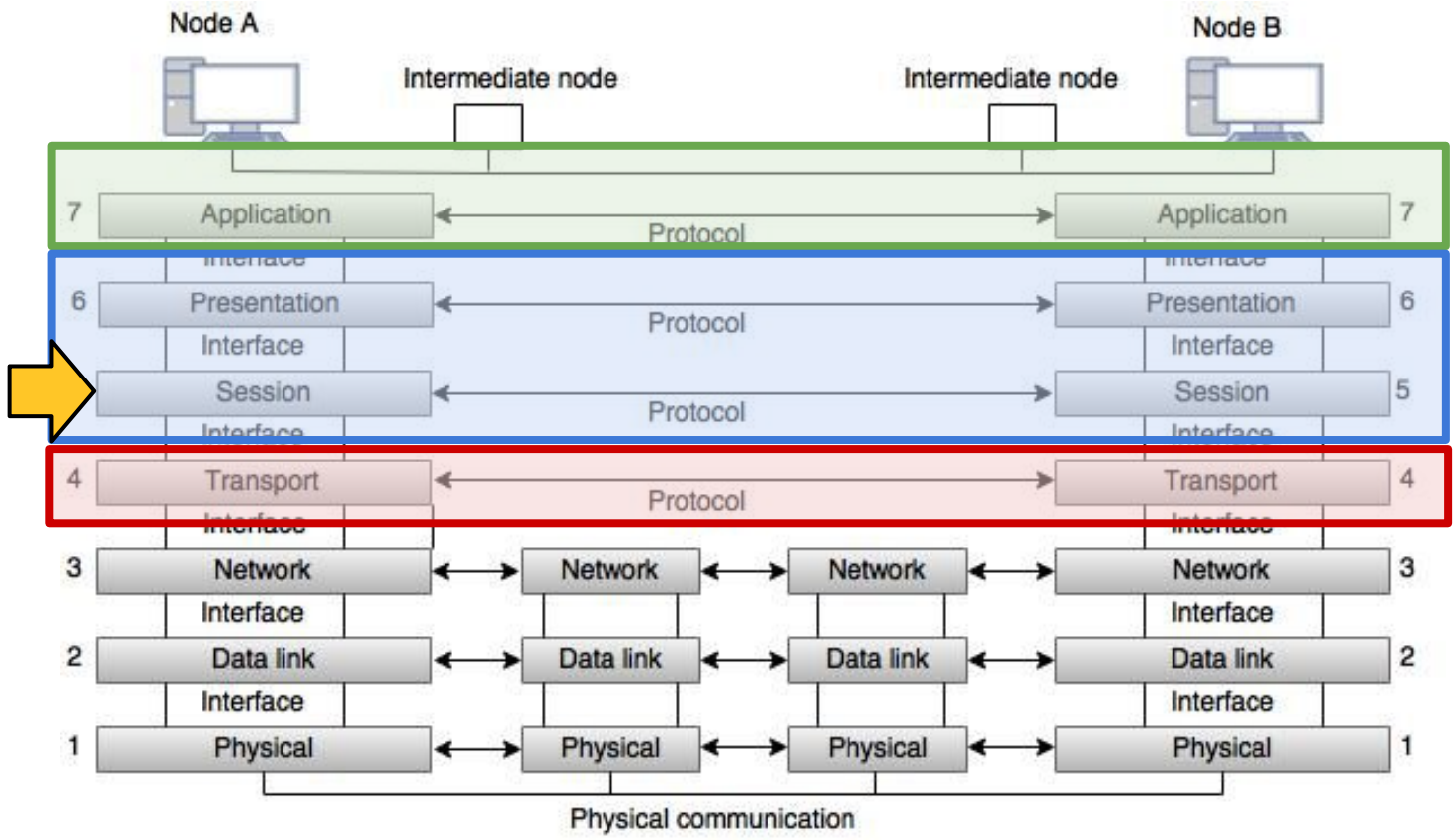


Fig: OSI Model

SER 321
Middleware

Middleware:

Presentation Layer Responsibilities:

Translation

Compression

Encryption

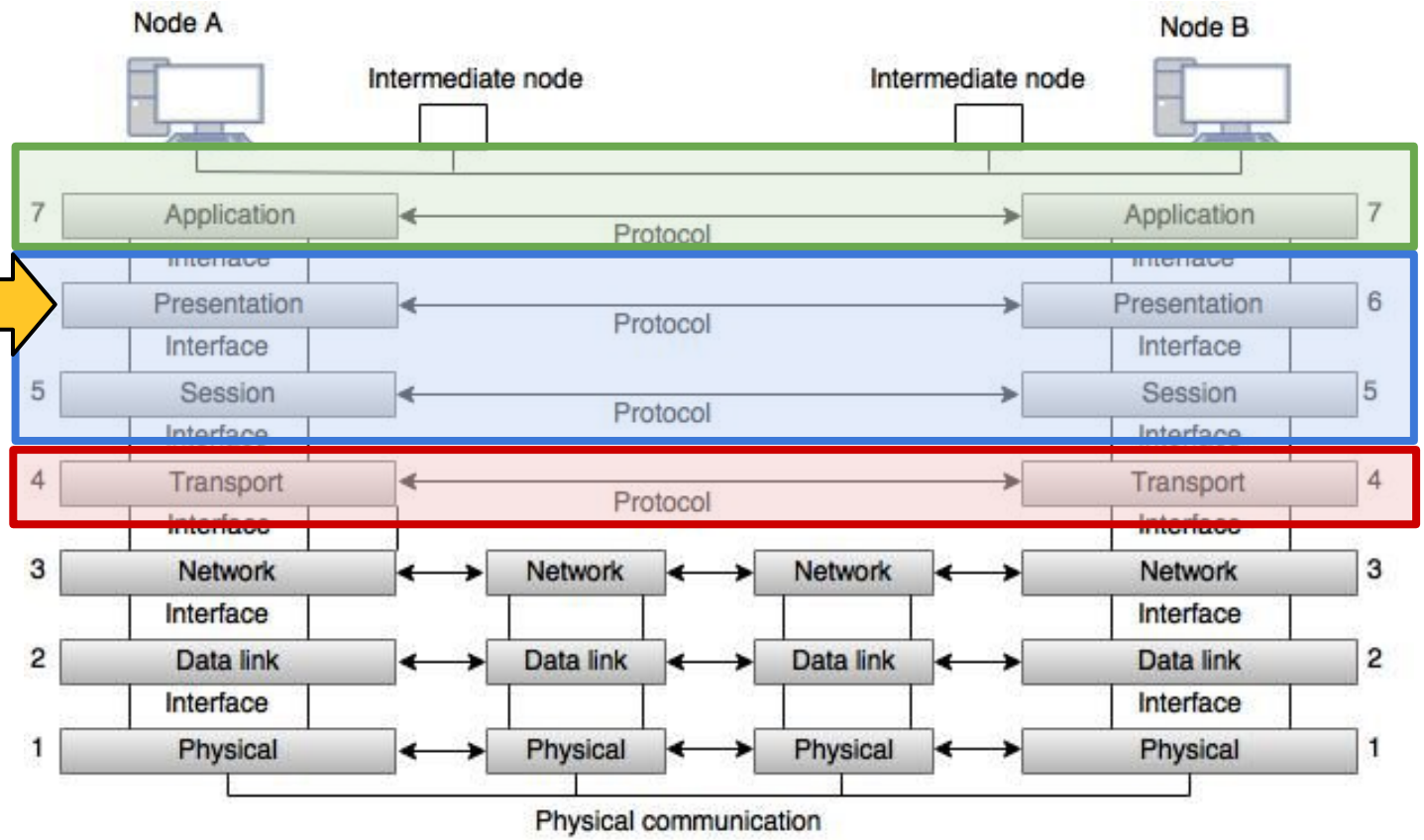


Fig: OSI Model

SER 321
Middleware

Examples?

Message Oriented Middleware (MOM)

Web Frameworks

Remote Procedure Calls (RPC)

App. Programming Interface (API)

SER 321
Middleware

Why do we care?

- Agility
- Reusability
- Efficiency
- Cost Effectiveness
- Portability

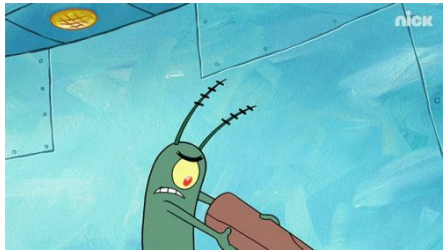
SER 321
Middleware

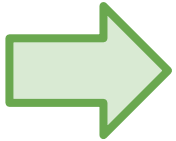
Why do we care?

Sort of like publishing a contract

“If you follow these rules, I will handle your request.”

SER 321
Middleware

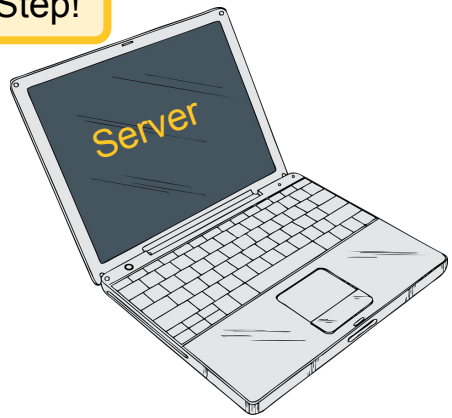




```
{
        “type” : “addUser”,
        “name” : “katie”,
        “password” : “password”
}
```

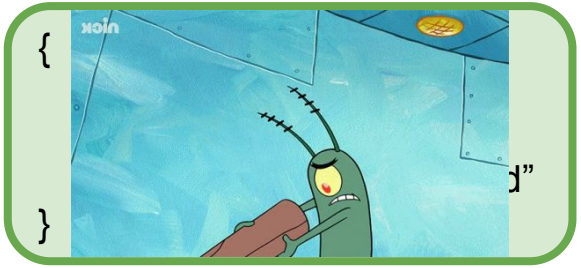
- Get data from user
- Validate data
- Determine Request Format
- Construct Valid Request
- Establish Connection
- Send Request
- Wait for Response
  - Read Response from Stream
  - Parse Response
  - Display Response to User

Critical Step!



SER 321
Middleware

- Get data from user
- Validate data
- Determine Request Format
- Construct Valid Request
- Establish Connection
- Send Request
- Wait for Response
  - Read Response from Stream
  - Parse Response
  - Display Response to User



How do we know?

Add User

- Read data from Stream
- Parse Data
- Validate Request Format
- Determine Request Type
- *Perform Requested Action*
- Determine Response Format
- Construct Valid Response
- [Re-establish connection]
- Send Response

Done!

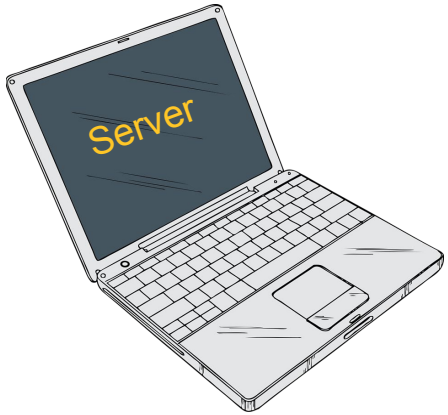
Wants to Add User

SER 321
Middleware

With Middleware:

Add User:
Name = Katie
Password = password

API



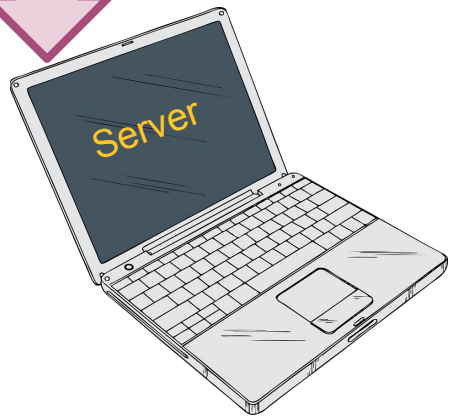
SER 321
Middleware

With Middleware:

Add User:
Name = Katie
Password = password

API


```
{
        “type” : “addUser”,
        “name” : “katie”,
        “password” : “password”
}
```



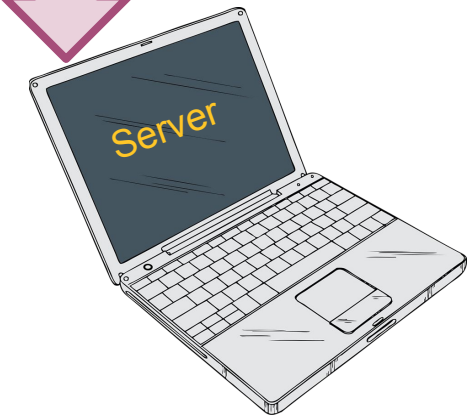
SER 321
Middleware

With Middleware:

Get repositories for a specific user

API

GitHub REST API



Code samples for "List repositories for a user"

Request example

GET /users/{username}/repos

cURL | JavaScript | GitHub CLI

```
curl -L \
  -H "Accept: application/vnd.github+json" \
  -H "Authorization: Bearer <YOUR-TOKEN>" \
  -H "X-GitHub-Api-Version: 2022-11-28" \
  https://api.github.com/users/USERNAME/repos
```

SER 321
Middleware

With Middleware:

Get repositories for a specific user

API

GitHub REST API

https://api.github.com/users/kgrinne3/repos

Code samples for "List reposito

Request example

GET /users/{username}/repos

cURL JavaScript GitHub CL

```
curl -L \
  -H "Accept: application/vnd.gi
  -H "Authorization: Bearer <YOU
  -Api-Version: 2022
  github.com/users/U
```


```
[
  {
    "id": 550568457,
    "node_id": "R_kgDOINECCQ",
    "name": "assign1git",
    "full_name": "kgrinne3/assign1git",
    "private": false,
    "owner": {
      "login": "kgrinne3",
      "id": 115493885,
      "node_id": "U_kgDOBuJL_Q",
      "avatar_url": "https://avatars.githubusercontent.com/u/115493885?v=4",
      "gravatar_id": "",
      "url": "https://api.github.com/users/kgrinne3",
      "html_url": "https://github.com/kgrinne3",
      "followers_url": "https://api.github.com/users/kgrinne3/followers",
      "following_url": "https://api.github.com/users/kgrinne3/following{/other_user}",
      "gists_url": "https://api.github.com/users/kgrinne3/gists{/gist_id}",
      "starred_url": "https://api.github.com/users/kgrinne3/starred{/owner}{/repo}",
      "subscriptions_url": "https://api.github.com/users/kgrinne3/subscriptions",
      "organizations_url": "https://api.github.com/users/kgrinne3/orgs",
      "repos_url": "https://api.github.com/users/kgrinne3/repos",
      "events_url": "https://api.github.com/users/kgrinne3/events{/privacy}",
      "received_events_url": "https://api.github.com/users/kgrinne3/received_events",
      "type": "User",
      "site_admin": false
    },
    "html_url": "https://github.com/kgrinne3/assign1git",
    "description": "Katie Grinnell",
    "fork": false,
    "url": "https://api.github.com/repos/kgrinne3/assign1git",
    "forks_url": "https://api.github.com/repos/kgrinne3/assign1git/forks",
    "keys_url": "https://api.github.com/repos/kgrinne3/assign1git/keys{/key_id}",
    "collaborators_url": "https://api.github.com/repos/kgrinne3/assign1git/collaborators{/collaborator}",
    "teams_url": "https://api.github.com/repos/kgrinne3/assign1git/teams",
    "hooks_url": "https://api.github.com/repos/kgrinne3/assign1git/hooks",
    "issue_events_url": "https://api.github.com/repos/kgrinne3/assign1git/issues/events{/number}",
    "events_url": "https://api.github.com/repos/kgrinne3/assign1git/events",
    "assignees_url": "https://api.github.com/repos/kgrinne3/assign1git/assignees{/user}",
    "branches_url": "https://api.github.com/repos/kgrinne3/assign1git/branches{/branch}",
    "tags_url": "https://api.github.com/repos/kgrinne3/assign1git/tags",
    "blobs_url": "https://api.github.com/repos/kgrinne3/assign1git/git/blobs{/sha}",
    "git_tags_url": "https://api.github.com/repos/kgrinne3/assign1git/git/tags{/sha}",
```

SER 321

Scratch Space

Upcoming Events

SI Sessions:

- ~~Thursday, November 28th at 7:00 pm MST~~ **CANCELLED - Happy Thanksgiving!**
- Sunday, December 1st at 7:00 pm MST - **2 hour Review Session**
- Tuesday, December 3rd at 10:00 am MST - **Q&A Session**

Review Sessions:

- Sunday, December 1st at 7:00 pm MST - **2 hour Review Session**
- Tuesday, December 3rd at 10:00 am MST - **Q&A Session**

Questions?

Survey:

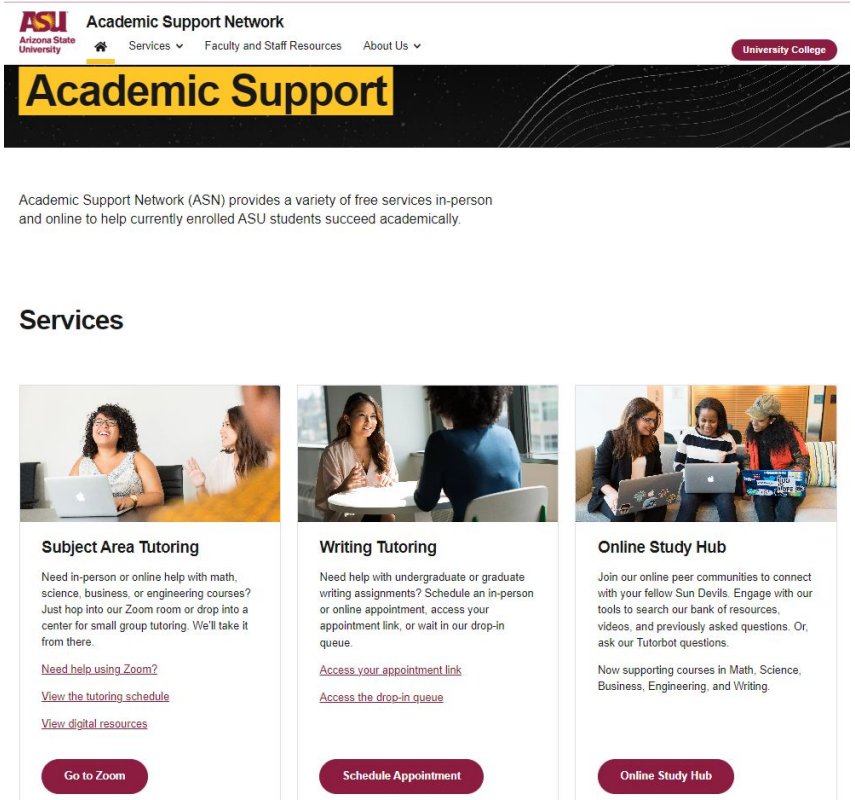
https://asuasn.info/ASNSurvey

36

More Questions?

Check out our other resources!

tutoring.asu.edu

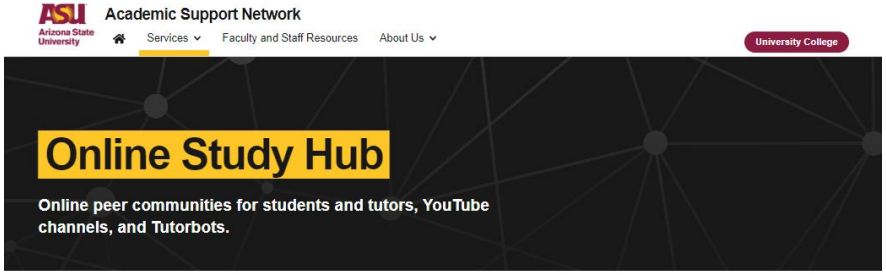


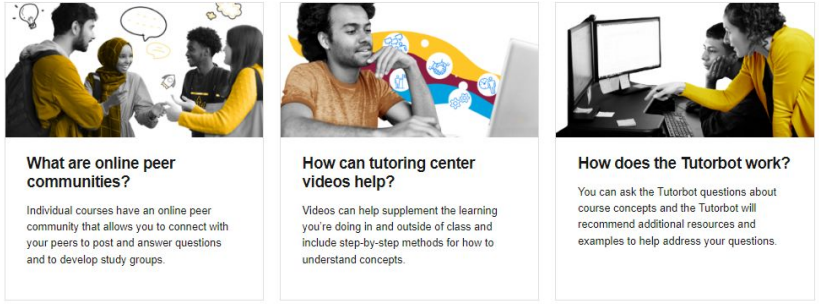
1. **Click on ‘Go to Zoom’ to log onto our Online Tutoring Center.**
2. **Click on ‘View the tutoring schedule’ to see when tutors are available for specific courses.**

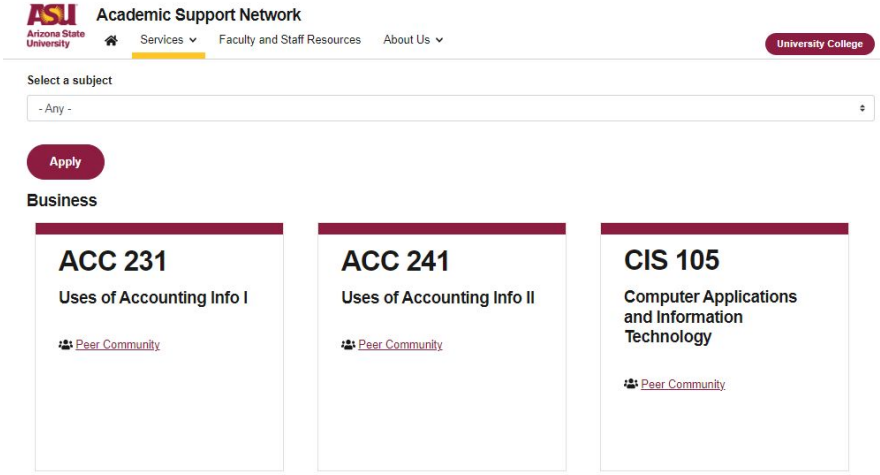
More Questions?

Check out our other resources!

tutoring.asu.edu/online-study-hub



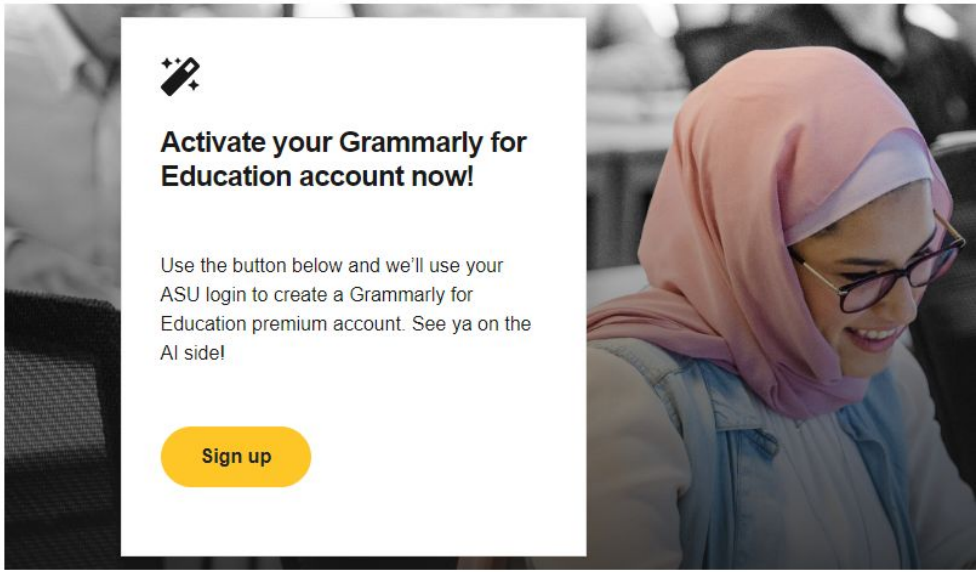




Don't forget to check out the Online Study Hub for additional resources!

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
- RAFT