SER 321 B Session

SI Session

Tuesday, April 29th 2025

10:00 am - 11:00 am MST

Agenda

Requested Material

Continued Review!

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
Exam Information

Exam Info Page

80 minutes

Similar to the quizzes

Make sure to look at the Study Guide!

Opens: Wednesday April 30th @ 12:30 AM

Closes: Friday May 2nd @ 11:59:59 PM

SER 321

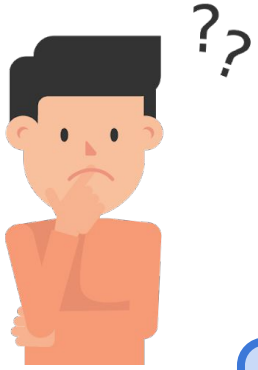
Review Requests

It's ***not*** too late to make a topic request!

Drop a concept in the chat and we can cover it next!

SER 321

HTTP(s)



Stateful OR Stateless

Synchronous OR Asynchronous

Can we recall some of the formats?

JSON

Java Object Serialization

Protocol Buffers

XML

SER 321
Serialization

Binary

Text

Two main approaches for storing the content…

What about the data format?

JSON

Java Object Serialization

Protocol Buffers

XML

SER 321
Serialization

Binary

Text

Who uses ***TEXT***?

Text

JSON

Java Object Serialization

Protocol Buffers

Text

XML

SER 321
Serialization

Binary

Text

What does this imply?

Who uses ***BINARY***?

Text — JSON

Binary — Java Object Serialization

Binary — Protocol Buffers

Text — XML

org.json Docs

JSON Guide

SER 321

JSON Recognition

How many Objects?

How many Arrays?

How many Members?


```
{
  "lat": 42.3434,
  "lon": -88.0412,
  "timezone": "America/Chicago",
  "timezone_offset": -21600,
  "current": {
    "dt": 1733070576,
    "sunrise": 1733058144,
    "sunset": 1733091649,
    "temp": 18.57,
    "feels_like": 5.97,
    "pressure": 1025,
    "humidity": 63,
    "dew_point": 9.21,
    "uvi": 0.79,
    "clouds": 0,
    "visibility": 10000,
    "wind_speed": 14.97,
    "wind_deg": 280,
    "wind_gust": 21.85,
    "weather": [
      {
        "id": 800,
        "main": "Clear",
        "description": "clear sky",
        "icon": "01d"
      }
    ]
  }
}
```

SER 321
Serialization

Streams and their types

Generic Superclass

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

Buffered Stream

Bytes

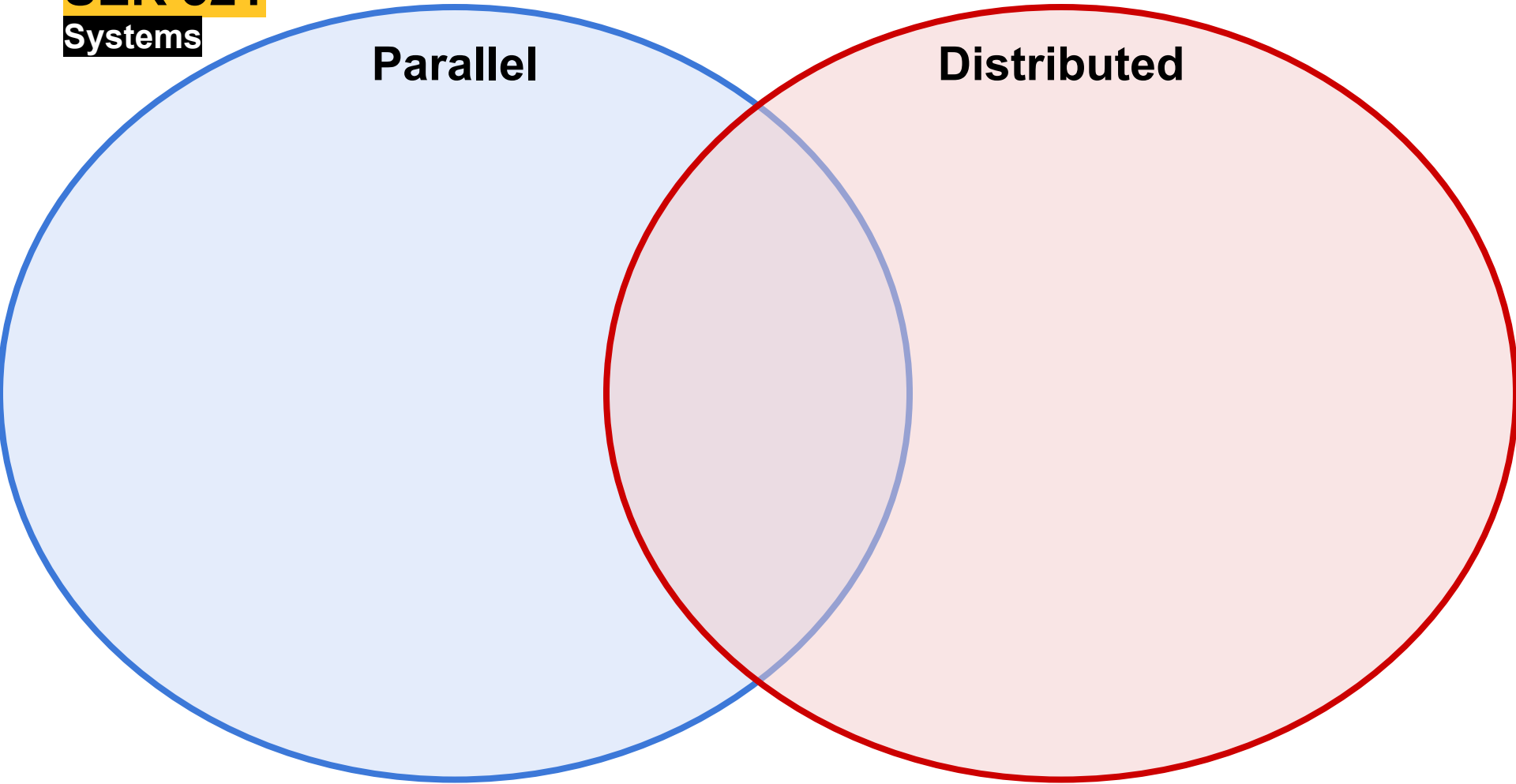
Data Stream

Primitive DATA Types

Object Stream

Java Objects

SER 321
Systems



SER 321
Systems

Parallel

- Single computer
- Work split among different *processors*
- Memory is shared ***or*** distributed
- Communicate through *bus*
- Latency while waiting for resources

Distributed

- Many computers
- Work split among different *locations*
- Memory is distributed
- Communicate through *message passing*
- Experience latency both between nodes and within nodes

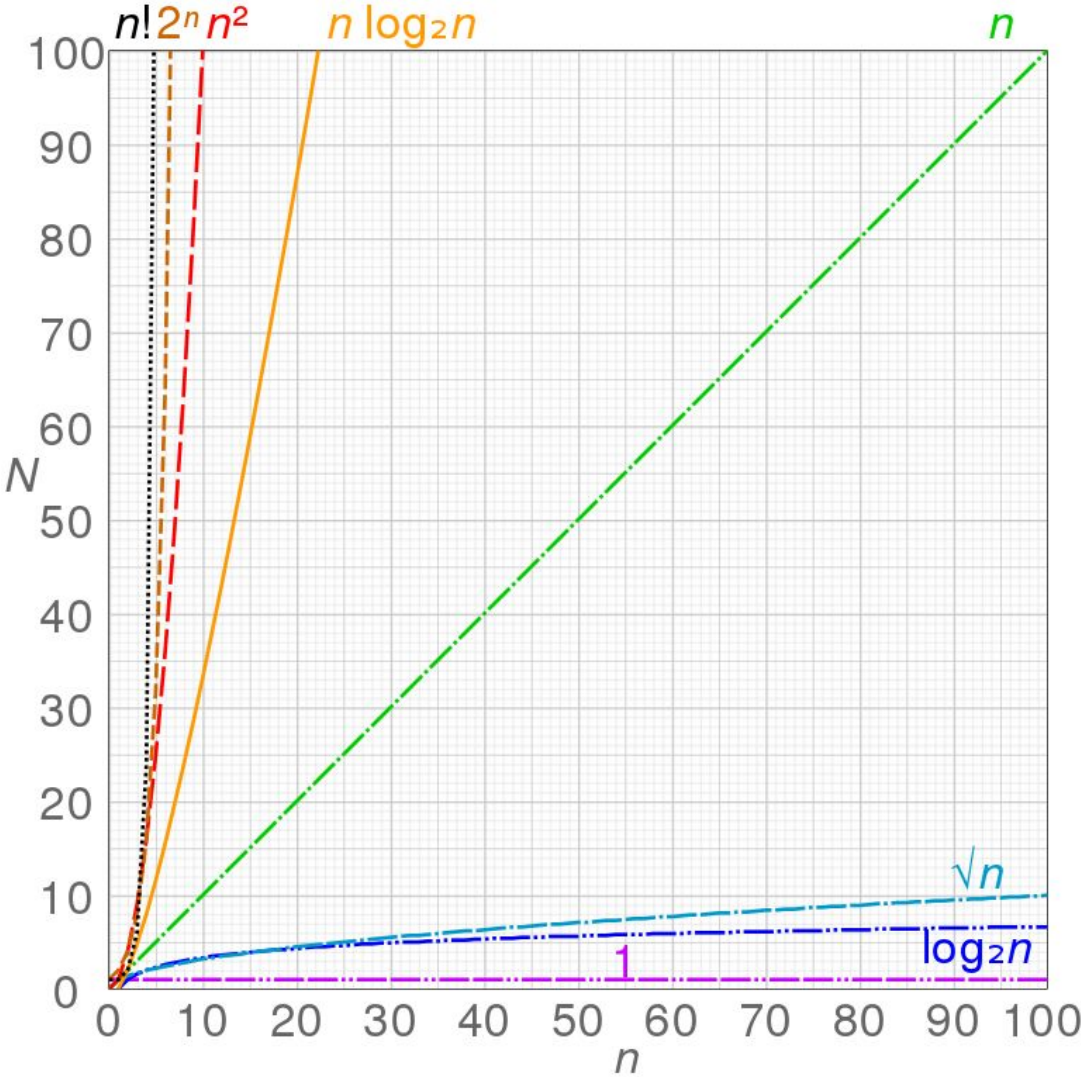
Parallel / Distributed

- Work is partitioned
- Partitions processed individually
- ***Can*** improve performance
- ***Can*** improve speed
- Experience Latency

SER 321

When to Distribute

When should we *consider* distributing?

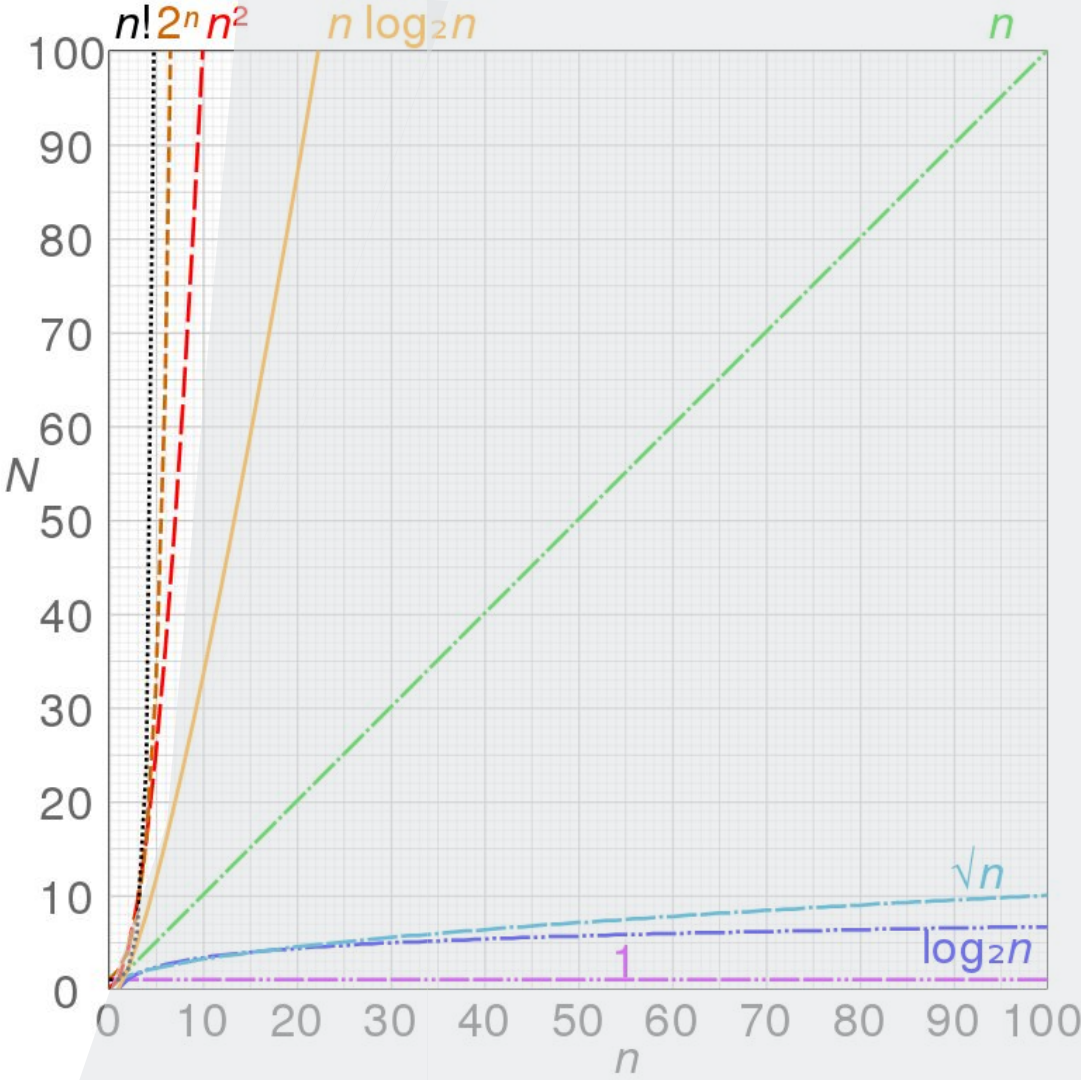


SER 321

When to Distribute

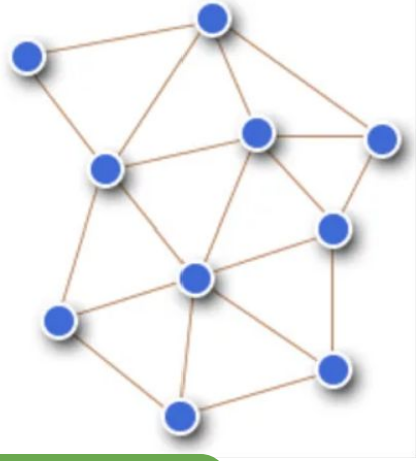
When should we *consider* distributing?

Super Duper Extra Extra Large Orders of Magnitude!



SER 321
Distributed Systems

Distributed System Properties



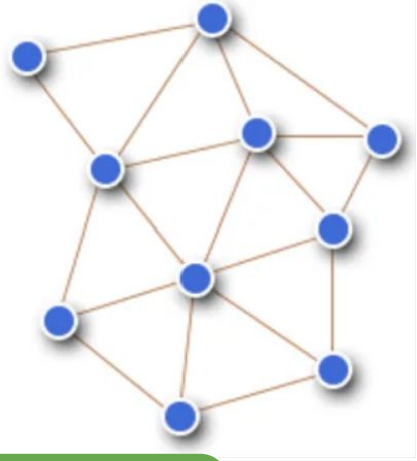
Global Clock

No! 👎

Yes! 👍

SER 321
Distributed Systems

Distributed System Properties



Nodes Fail

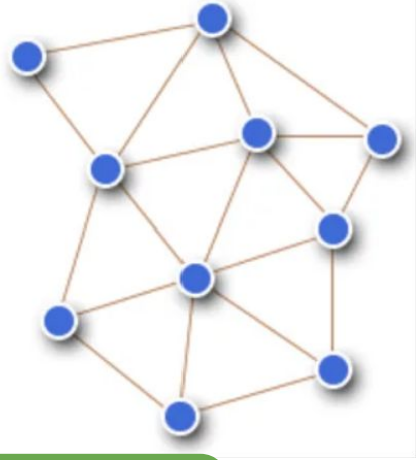
No! 👎

Yes! 👍

Global Clock

SER 321
Distributed Systems

Distributed System Properties



Cluster Changes

No! 👎

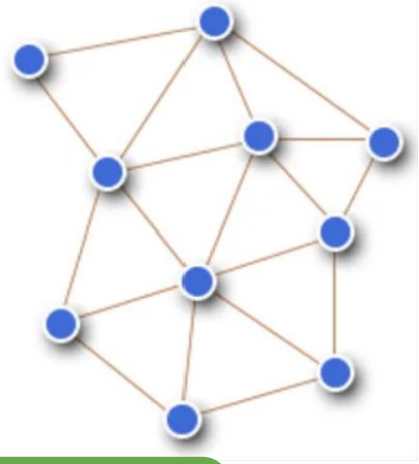
Global Clock

Yes! 👍

Nodes Fail

SER 321
Distributed Systems

Distributed System Properties



Network is Reliable

No! 👎

Global Clock

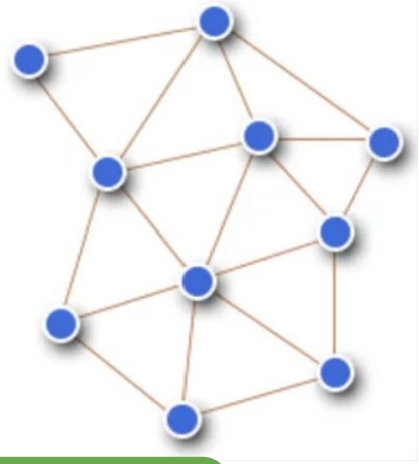
Yes! 👍

Nodes Fail

Cluster Changes

SER 321
Distributed Systems

Distributed System Properties



Latency Never Exists

No! 👎

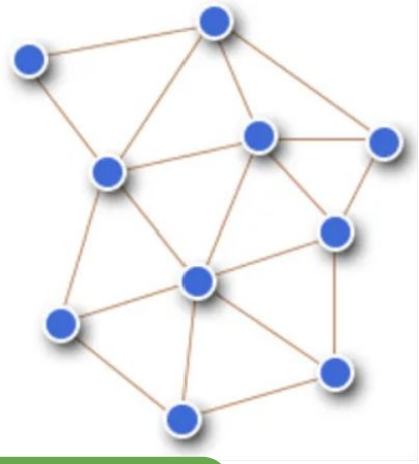
- Global Clock
- Network is Reliable

Yes! 👍

- Nodes Fail
- Cluster Changes

SER 321
Distributed Systems

Distributed System Properties



Path taken Changes

No! 👎

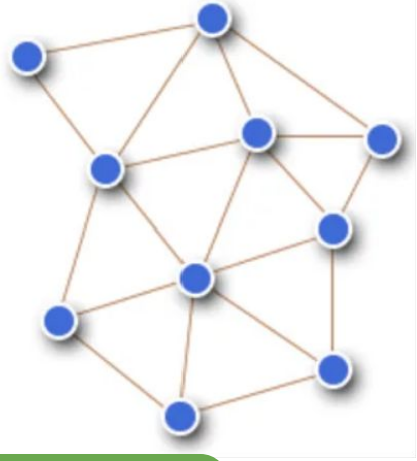
- Global Clock
- Network is Reliable
- Latency Never Exists

Yes! 👍

- Nodes Fail
- Cluster Changes

SER 321
Distributed Systems

Distributed System Properties



Share Common Resources

No! 👎

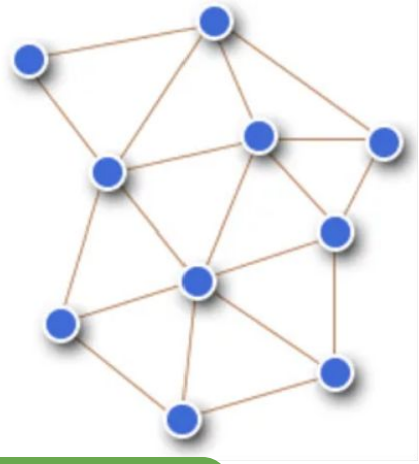
- Global Clock
- Network is Reliable
- Latency Never Exists

Yes! 👍

- Nodes Fail
- Cluster Changes
- Path taken Changes

SER 321
Distributed Systems

Distributed System Properties



Pitfalls handled inherently

No! 👎

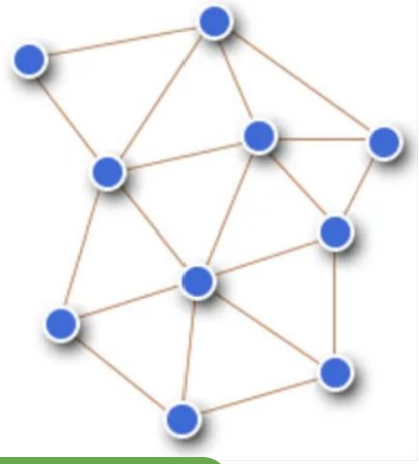
- Global Clock
- Network is Reliable
- Latency Never Exists

Yes! 👍

- Nodes Fail
- Cluster Changes
- Path taken Changes
- Share Common Resources

SER 321
Distributed Systems

Distributed System Properties



No! 👎

- Global Clock
- Network is Reliable
- Latency Never Exists
- Pitfalls handled inherently

Yes! 👍

- Nodes Fail
- Cluster Changes
- Path taken Changes
- Share Common Resources

SER 321
Consensus

What is Consensus?

"General agreement or trust amongst a group"

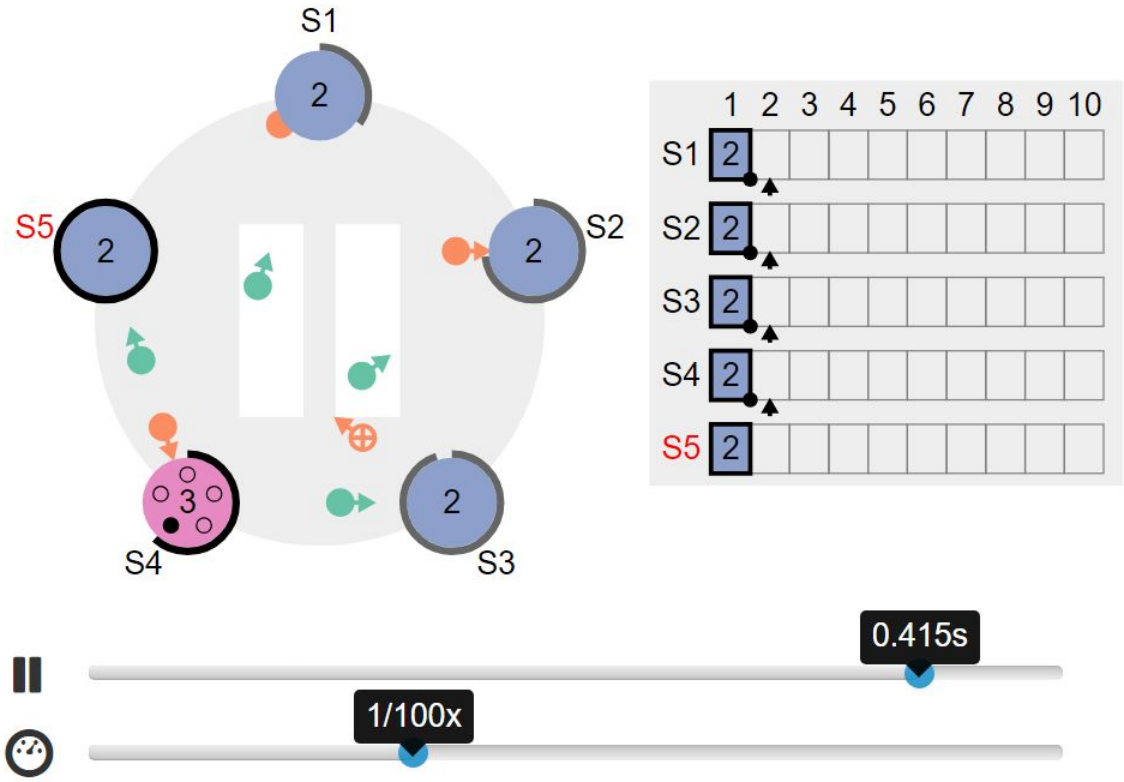
Who's in charge or keeping the beat	Leader Election
Check your work with a neighbor	Result Verification
Verify and maintain my copy of the data	Log Replication
Do I want to let you into my network	Node Validation

SER 321

RAFT

How do we feel about Consensus?

Do we want to look at RAFT again?



SER 321
Middleware

We have been:

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

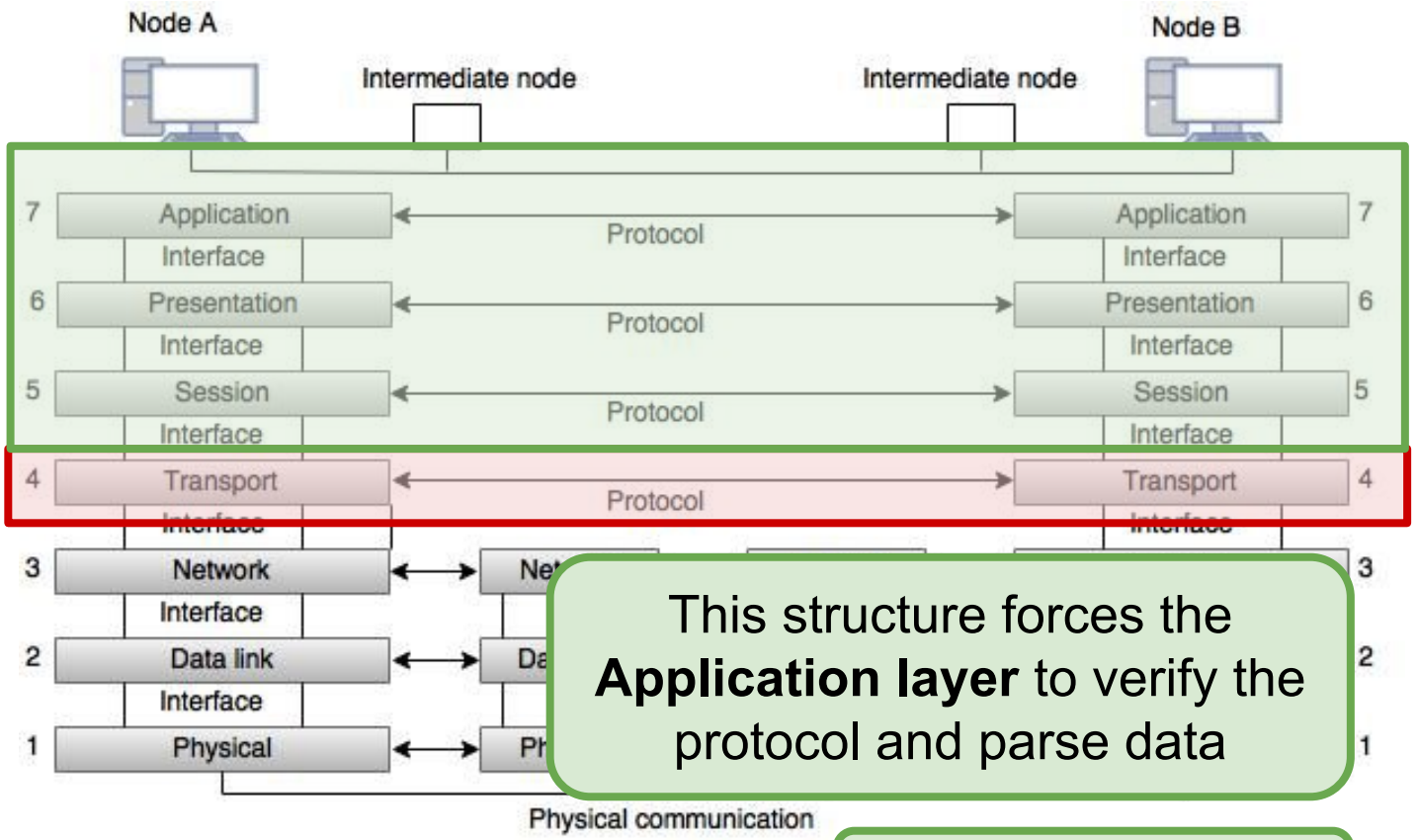


Fig: OSI Model

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

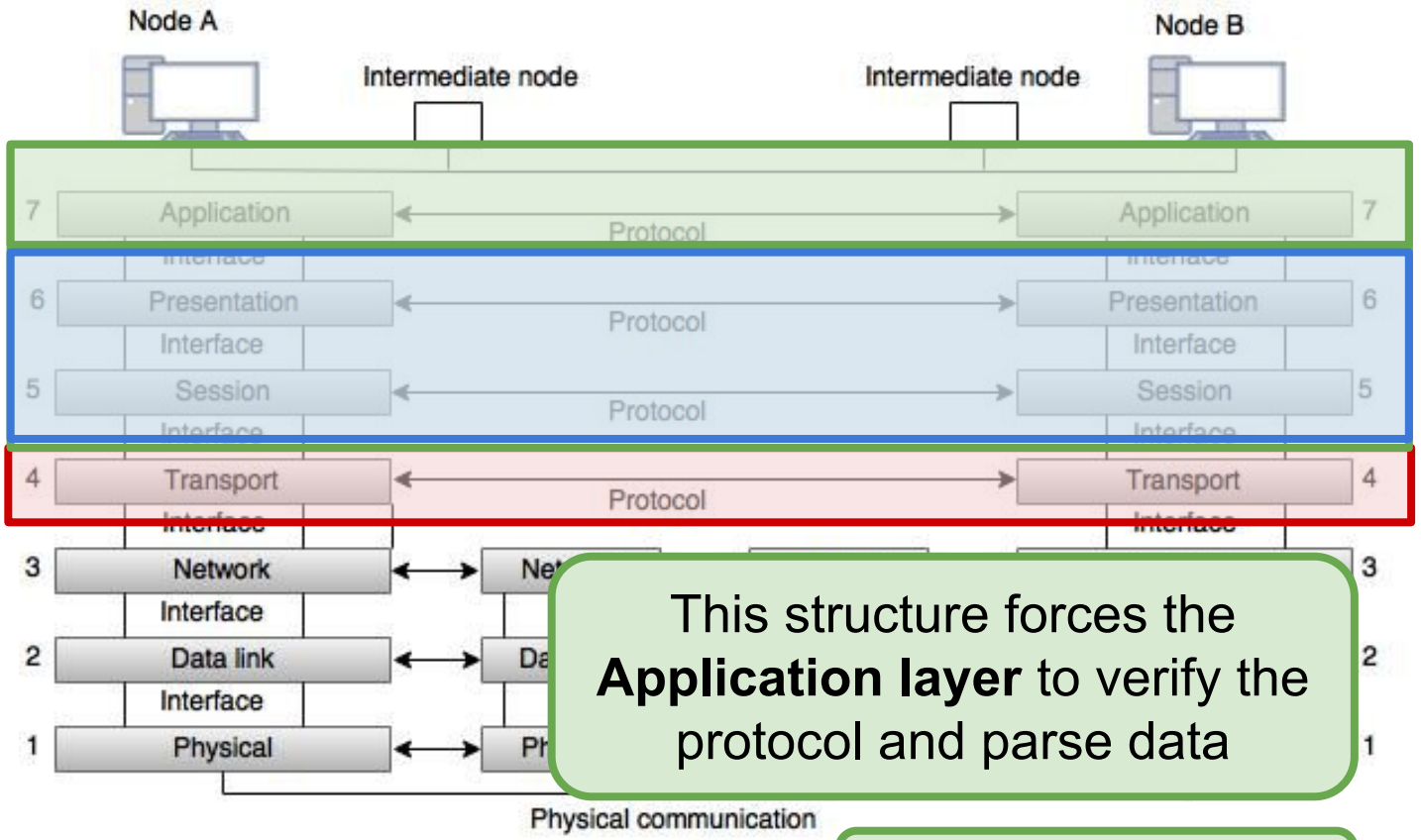


Fig: OSI Model

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

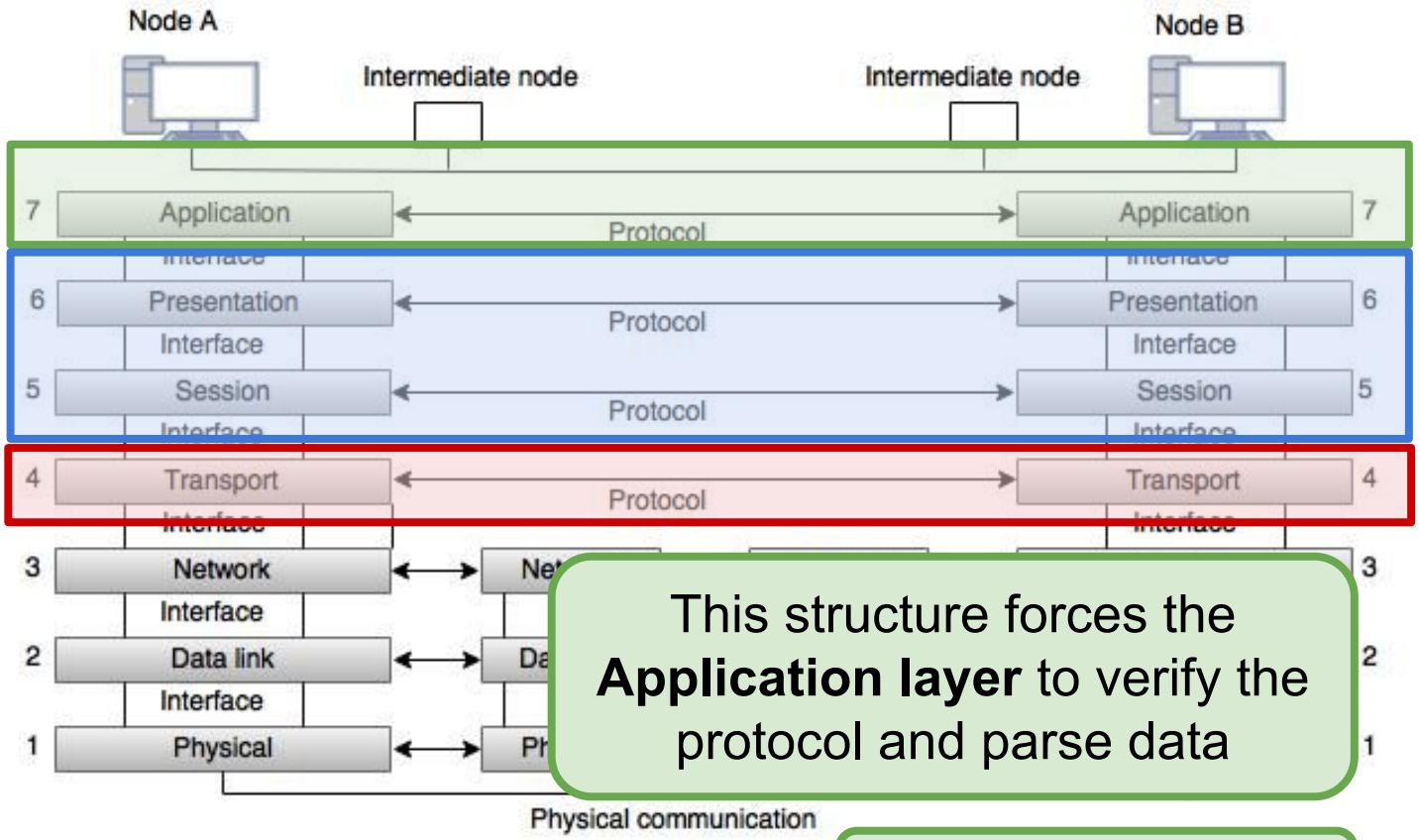


Fig: OSI Model

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

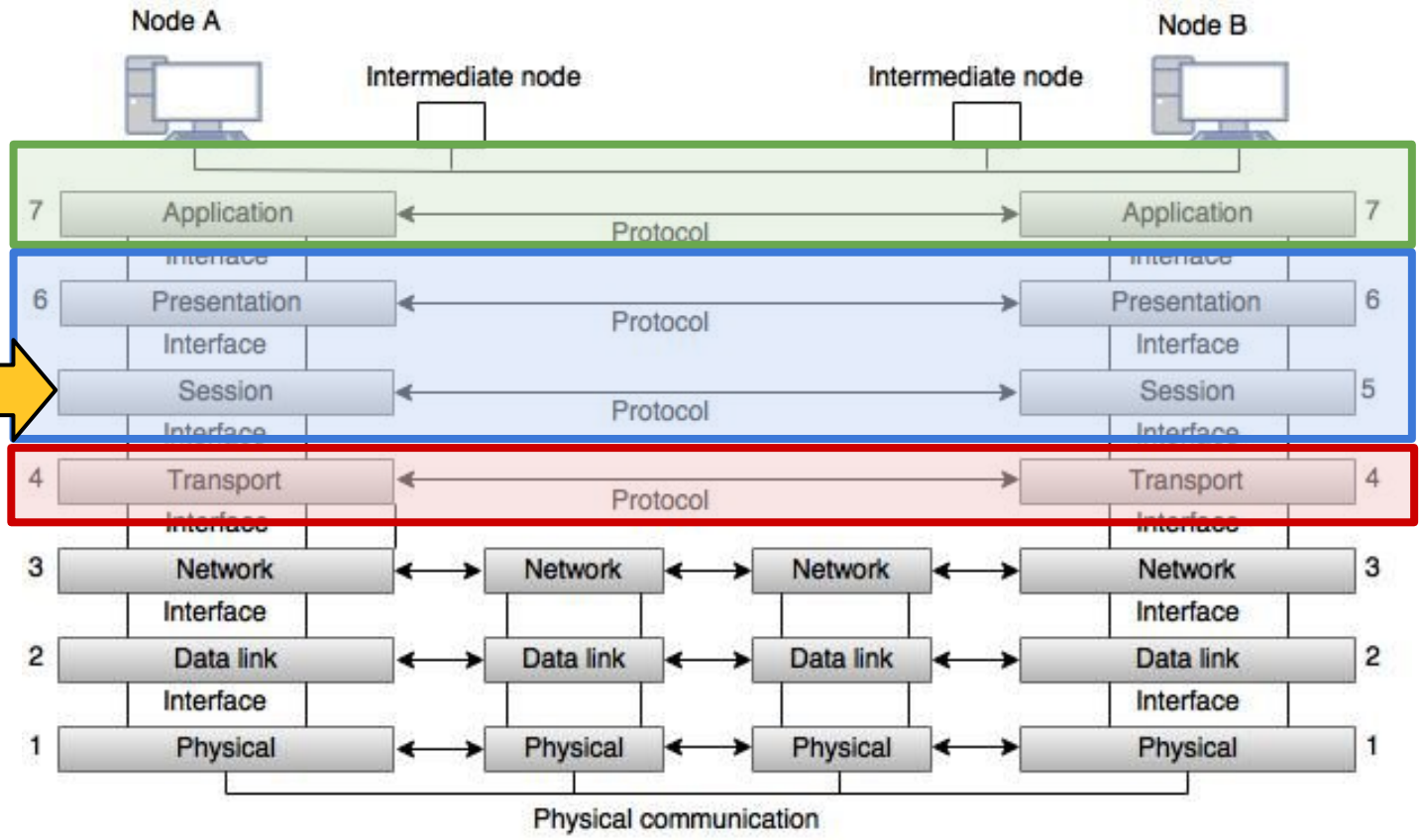


Fig: OSI Model

SER 321
Middleware

Middleware:

Presentation Layer Responsibilities:

Translation

Compression

Encryption

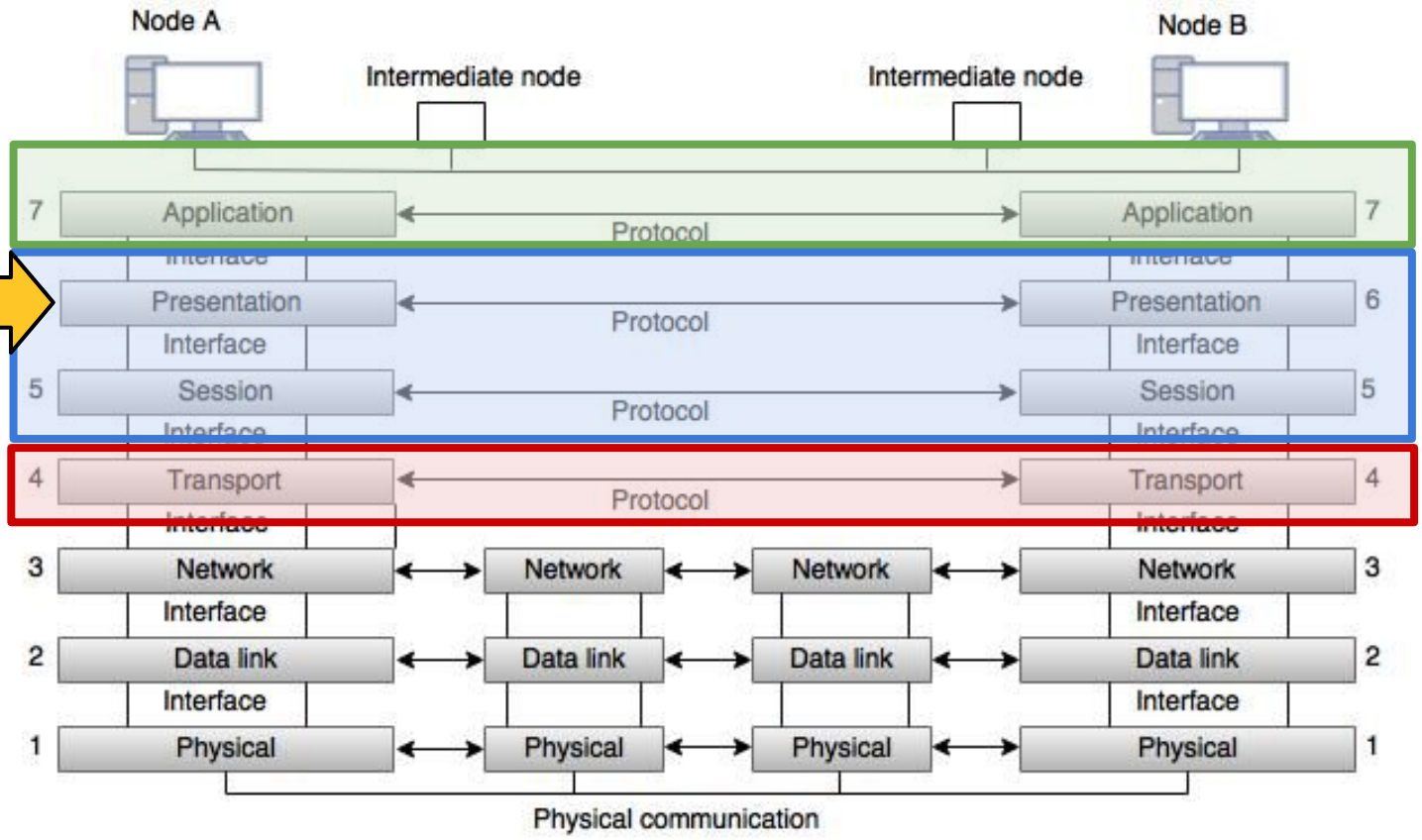


Fig: OSI Model

Examples?

Message Oriented Middleware (MOM)

Web Frameworks

Remote Procedure Calls (RPC)

App. Programming Interface (API)

SER 321
Middleware Benefits

Why do we care?

- Agility
- Reusability
- Efficiency
- Cost Effectiveness
- Portability

TL;DR → It's the "glue" between the client and server

SER 321
Middleware

Why do we care?

Sort of like publishing a contract

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

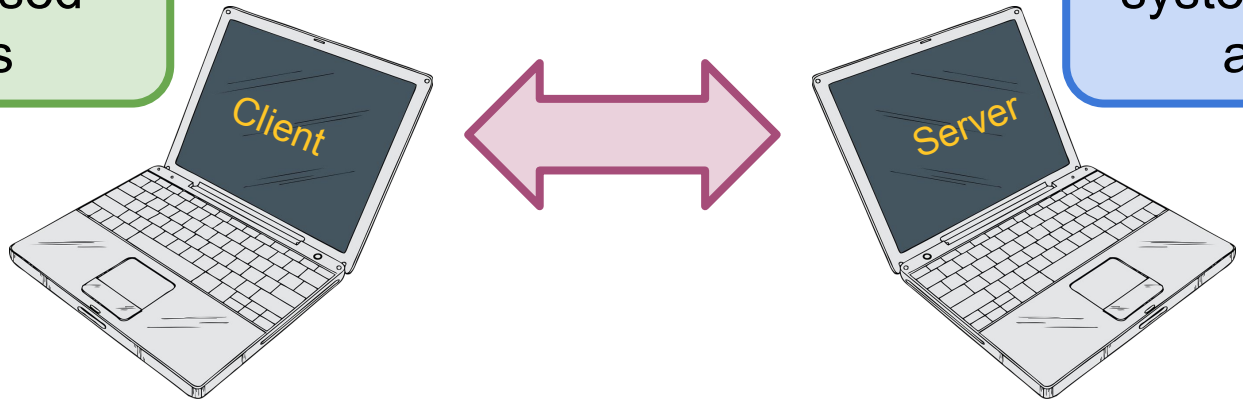
Separation of Concerns!

SER 321
Middleware

Why do we care?

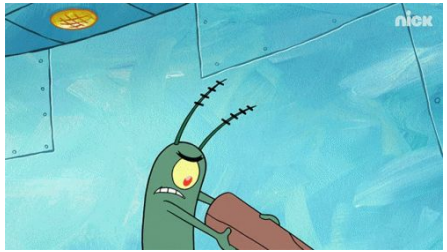
Client handles user-focused actions

Server handles system-focused actions



Middleware handles the in-between!

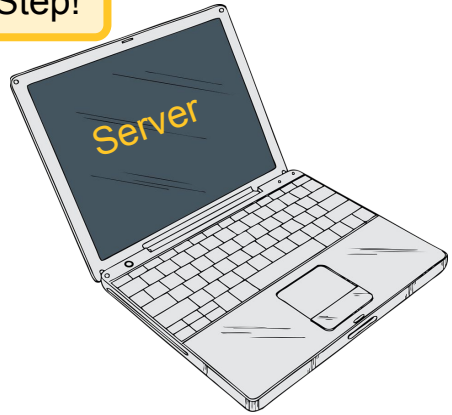
SER 321
Middleware




```
{
        "type" : "addUser",
        "name" : "katie",
        "password" : "password"
}
```

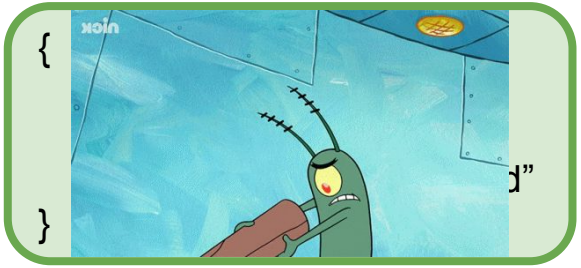
- Get data from user
- Validate data
- Determine Request Format
- Construct Valid Request
- Establish Connection
- Send Request
- Wait for Response
  - Read Response from Stream
  - Parse Response
  - Display Response to User

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

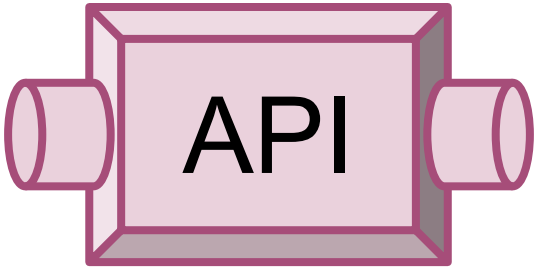
Done!

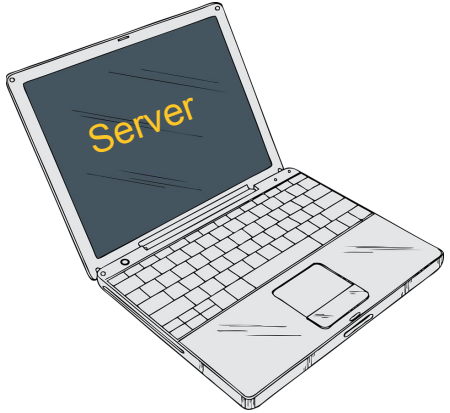
Wants to Add User

SER 321
Middleware

With Middleware:

Add User:
Name = Katie
Password = password





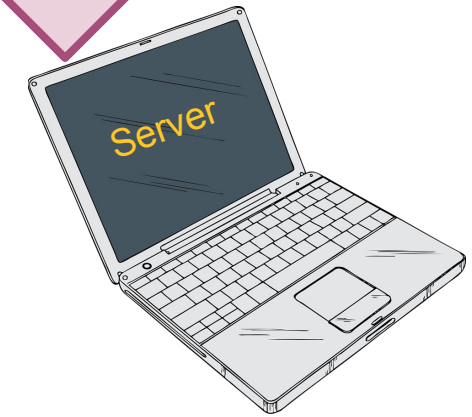
SER 321
Middleware

With Middleware:

Add User:
Name = Katie
Password = password

API

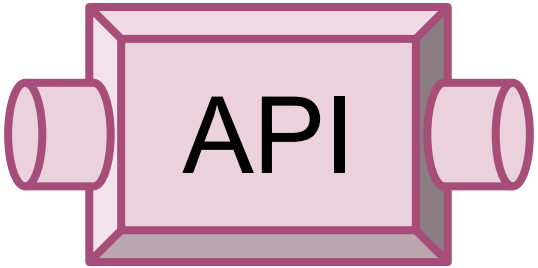
```
{
        "type" : "addUser",
        "name" : "katie",
        "password" : "password"
}
```



SER 321
Middleware

With Middleware:

Get repositories for a specific user



GitHub REST API

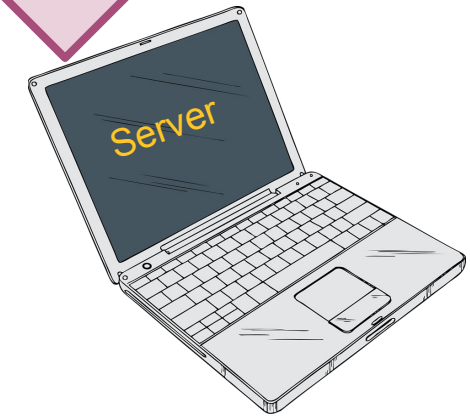
Code samples for "List repositories for a user"

Request example

GET /users/{username}/repos

cURL | JavaScript | GitHub CLI

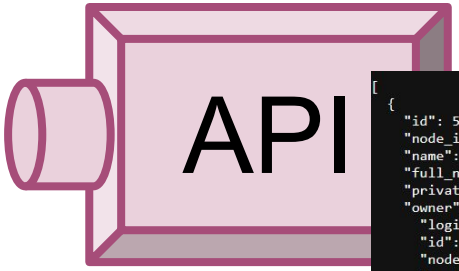
```
curl -L \
  -H "Accept: application/vnd.github+json" \
  -H "Authorization: Bearer <YOUR-TOKEN>" \
  -H "X-GitHub-Api-Version: 2022-11-28" \
  https://api.github.com/users/USERNAME/repos
```



SER 321
Middleware

With Middleware:

Get repositories for a specific user



GitHub REST API

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/US
```

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

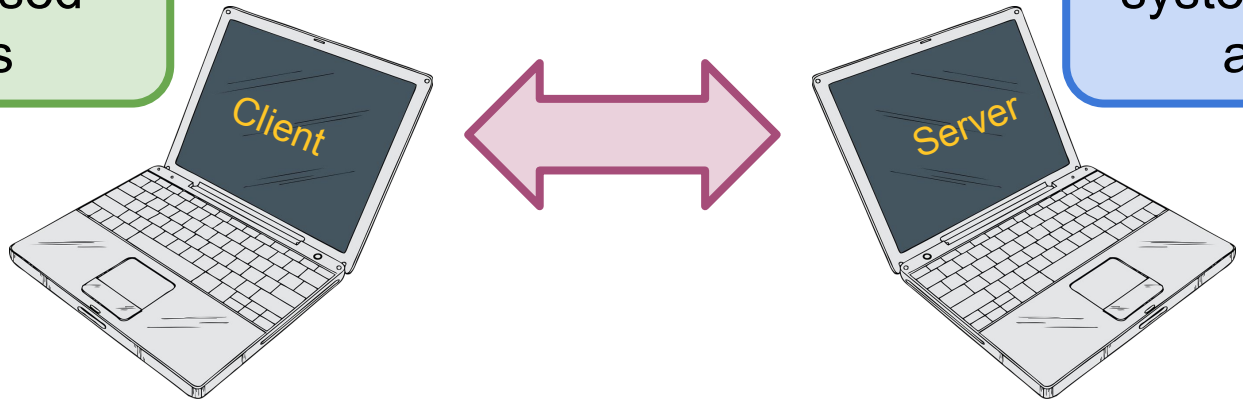

```
[
  {
    "id": 550568457,
    "node_id": "R_kgDOINECCQ",
    "name": "assign1git",
    "full_name": "kgrinne3/assign1git",
    "private": false,
    "owner": {
      "login": "kgrinne3",
      "id": 115493885,
      "node_id": "U_kgDOBuJL_Q",
      "avatar_url": "https://avatars.githubusercontent.com/u/115493885?v=4",
      "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
Middleware

Why do we care?

Client handles user-focused actions

Server handles system-focused actions



Middleware handles the in-between!

SER 321
Assign 6

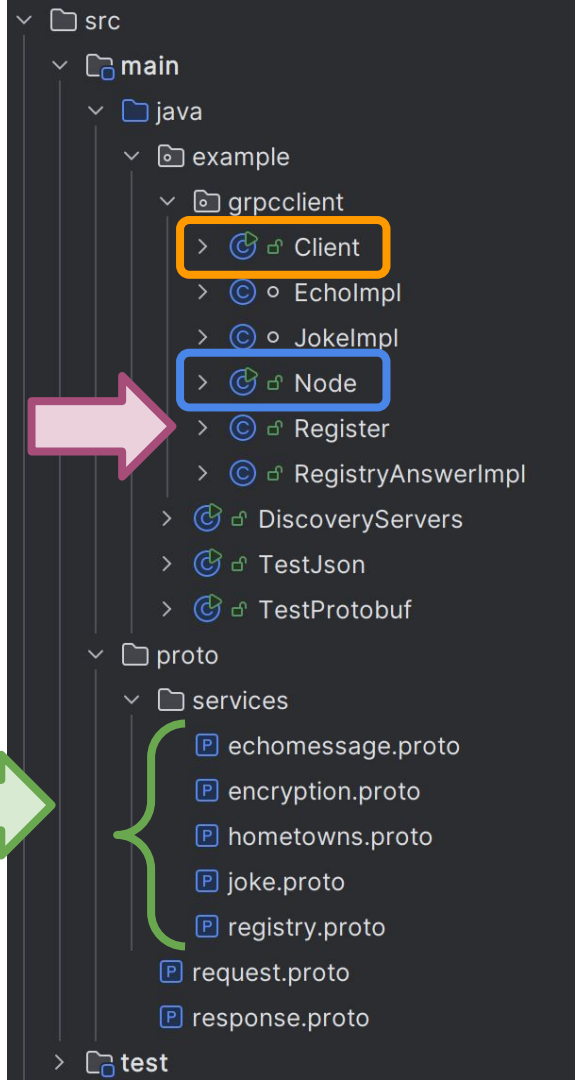
Client

Node

Registry

Service

Protocol Buffers!

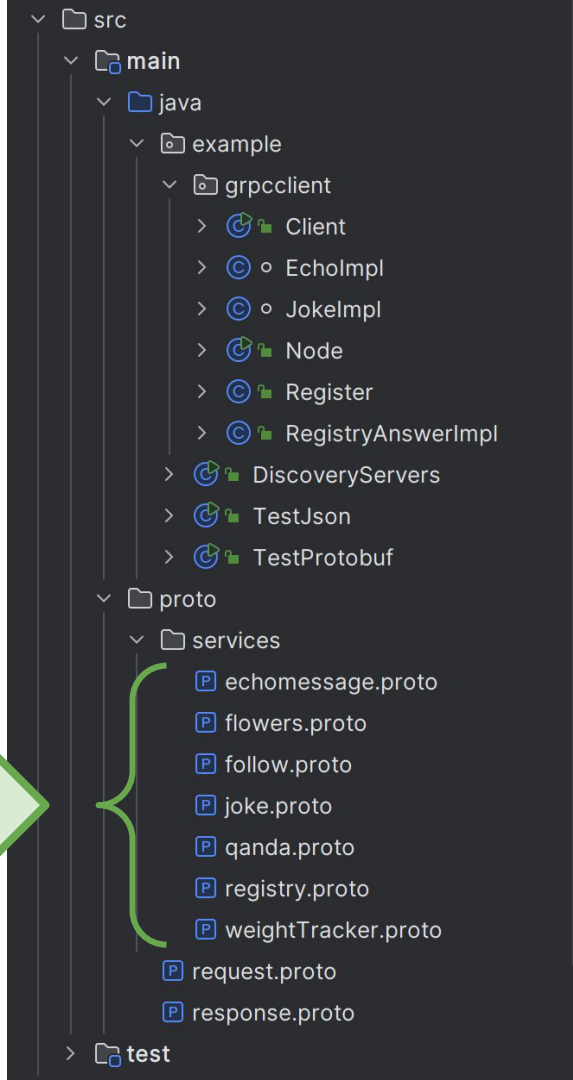


SER 321
Protobuf Review

All nodes and clients have agreed to these contracts

So ***DON'T CHANGE THEM!***

Think of these as a *contract*



SER 321

Protobuf Review

joke.proto

```
syntax = "proto3";

option java_multiple_files = true;
option java_package = "service";
option java_outer_classname = "JokeProto";

package services;

service Joke {
  rpc getJoke (JokeReq) returns (JokeRes) {}
  rpc setJoke (JokeSetReq) returns (JokeSetRes) {}
}

// The request message
message JokeReq {
  int32 number = 1;
```

```
// We are reading how many jokes the clients wants and put them in a list to send back to client
@Override  1 usage
public void getJoke(JokeReq req, StreamObserver<JokeRes> responseObserver) {
    System.out.println("Received from client: " + req.getNumber());
    JokeRes.Builder response = JokeRes.newBuilder();
    for (int i=0; i < req.getNumber(); i++){
        if(!jokes.empty()) {
            // yes, I take the joke out when it was used already,
            // should probably be done differently since this way
            // a joke cannot be told twice even to different clients
            response.addJoke(jokes.pop());
        }
        else {
            // this is more of a hack, better would be to either
            // check the number at the beginnig and say right away
            // if you do not have enough. Or send an error code or
            // similar as well.
            response.addJoke( value: "I am out of jokes...");
            break;
        }
    }
    JokeRes resp = response.build();
    responseObserver.onNext(resp);
    responseObserver.onCompleted();
}
```

```
// We take the joke the user wants to set and put it in our set of jokes
@Override  1 usage
public void setJoke(JokeSetReq req, StreamObserver<JokeSetRes> responseObserver) {

    System.out.println("Received from client: " + req.getJoke());
    JokeSetRes.Builder response = JokeSetRes.newBuilder();
    if (req.getJoke().isEmpty()) { // we do not want to add empty jokes
        response.setOk(false);
    } else {
        jokes.add(req.getJoke());
        response.setOk(true);
    }

    JokeSetRes resp = response.build();
    responseObserver.onNext(resp);
    responseObserver.onCompleted();
}
```

SER 321
Protobuf Review

joke.proto

```
syntax = "proto3";

option java_multiple_files = true;
option java_package = "service";
option java_outer_classname = "JokeProto";

package services;

service Joke {
  rpc getJoke (JokeReq) returns (JokeRes) {}
  rpc setJoke (JokeSetReq) returns (JokeSetRes) {}
}

// The request message
message JokeReq {
  int32 number = 1;
```

How do we use Protobufs again?

Use a ***Builder*** to construct the proto object

Fill with *setters*

Build when done!

```
// We take the joke the user wants to set and put it in our set of jokes
@Override  1 usage
public void setJoke(JokeSetReq req, StreamObserver<JokeSetRes> responseObserver) {

    System.out.println("Received from client: " + req.getJoke());
    JokeSetRes.Builder response = JokeSetRes.newBuilder();
    if (req.getJoke().isEmpty()) { // we do not want to add empty jokes
        response.setOk(false);
    } else {
        jokes.add(req.getJoke());
        response.setOk(true);
    }

    JokeSetRes resp = response.build();
    responseObserver.onNext(resp);
    responseObserver.onCompleted();
}
```

SER 321
Assign 6

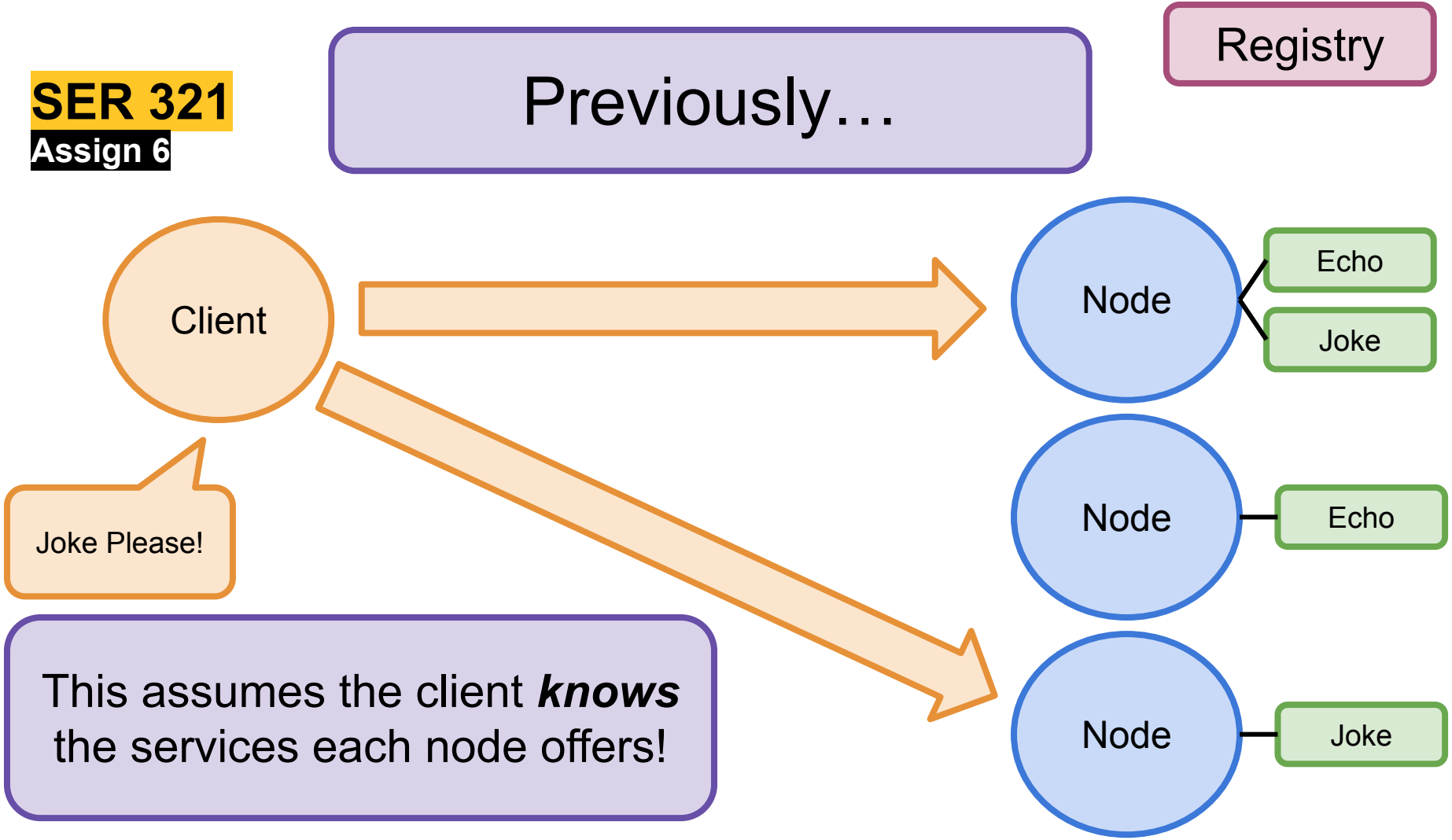
Two new concepts!

Registry

RPC

SER 321
Assign 6

Previously…



This assumes the client ***knows*** the services each node offers!

SER 321
Assign 6

With the Registry…

Registry

Client

Joke

Registry

Joke Please!

Node — Echo, Joke

Node — Echo

Node — Joke

SER 321
Assign 6

With the Registry…

Registry

Client

Joke

Registry

Node

Echo

Joke

Node

Echo

Node

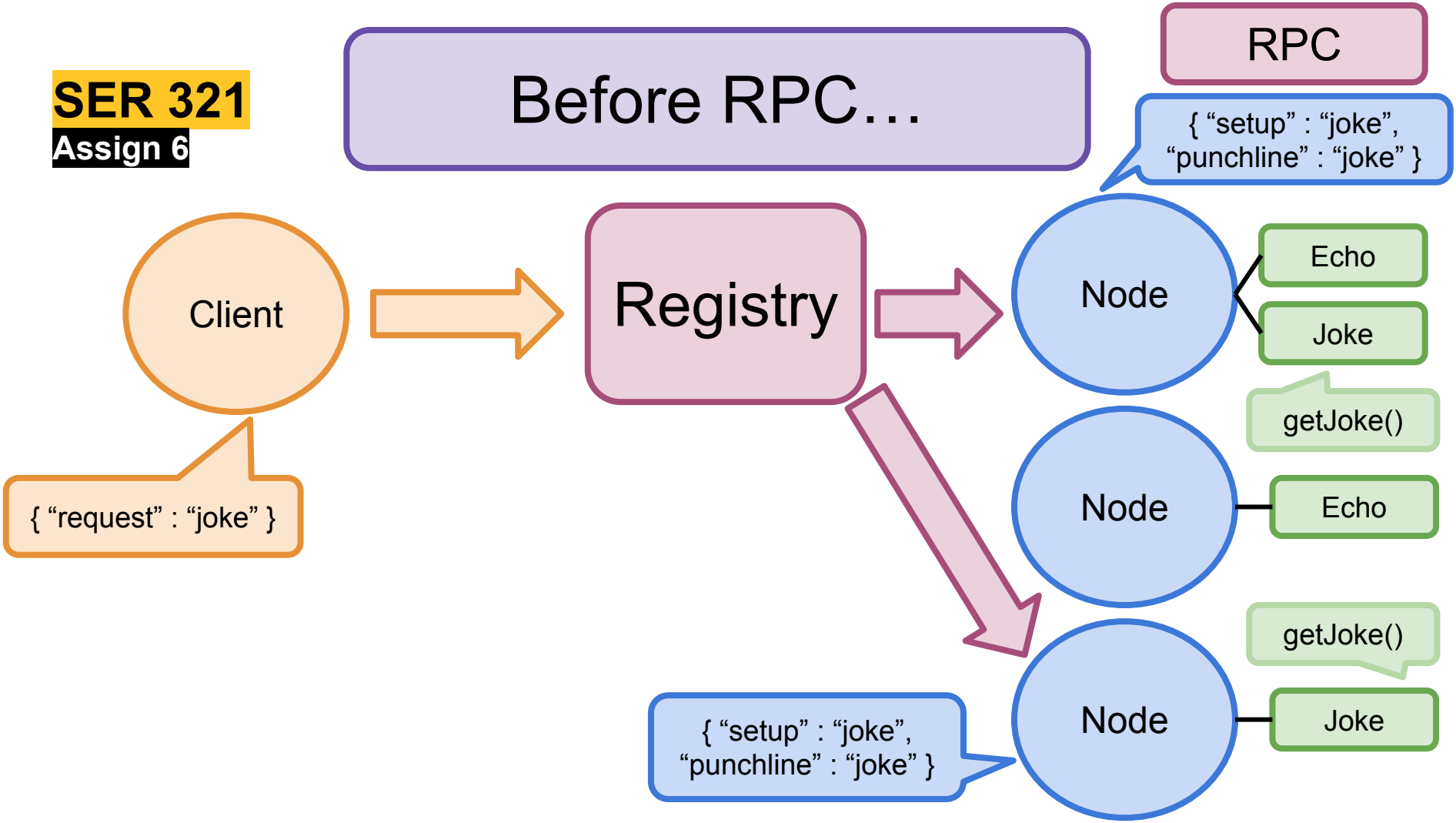
Joke

Joke Please!

Registry directs us to a node that can handle our request!

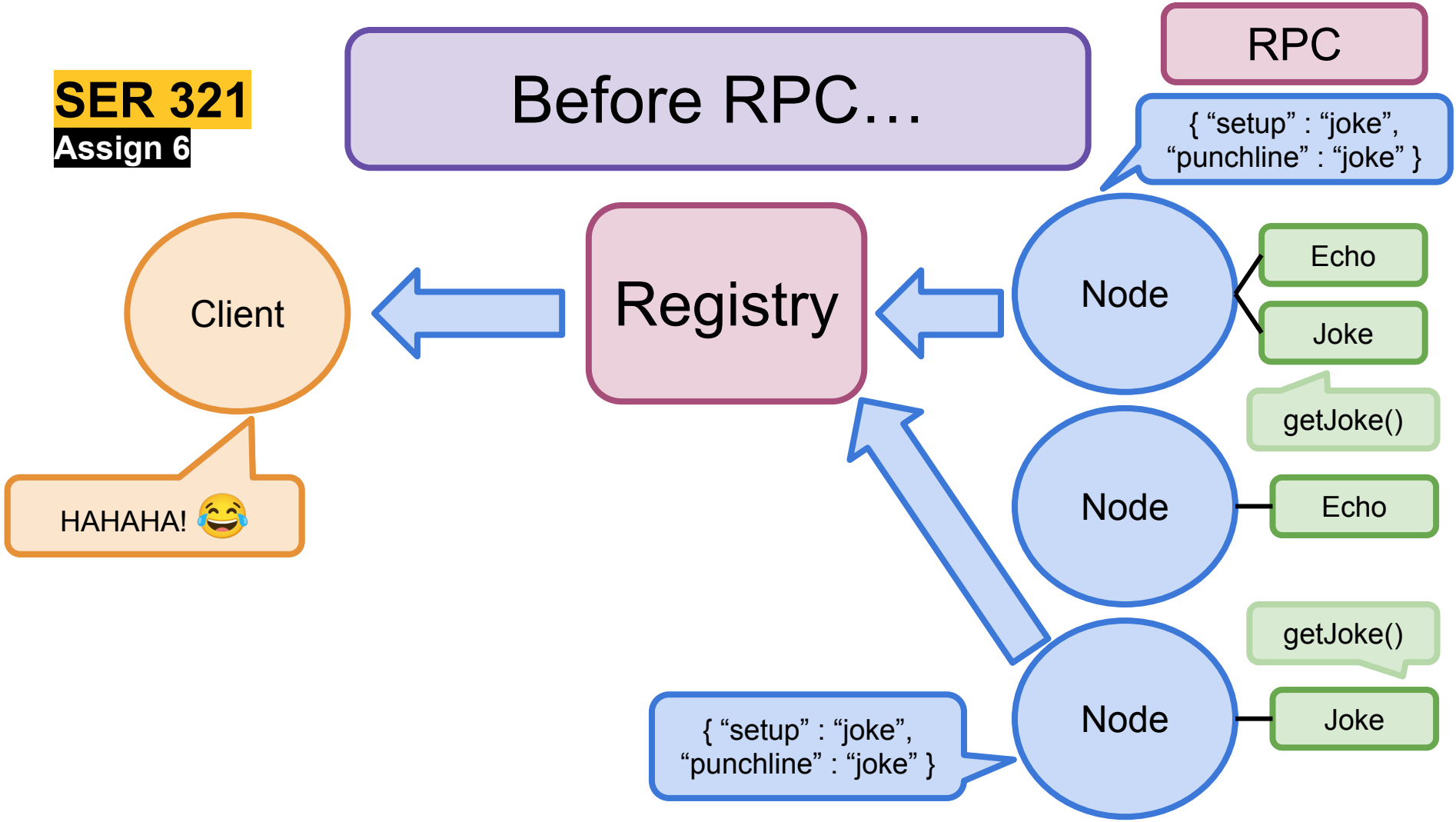
SER 321
Assign 6

Before RPC…



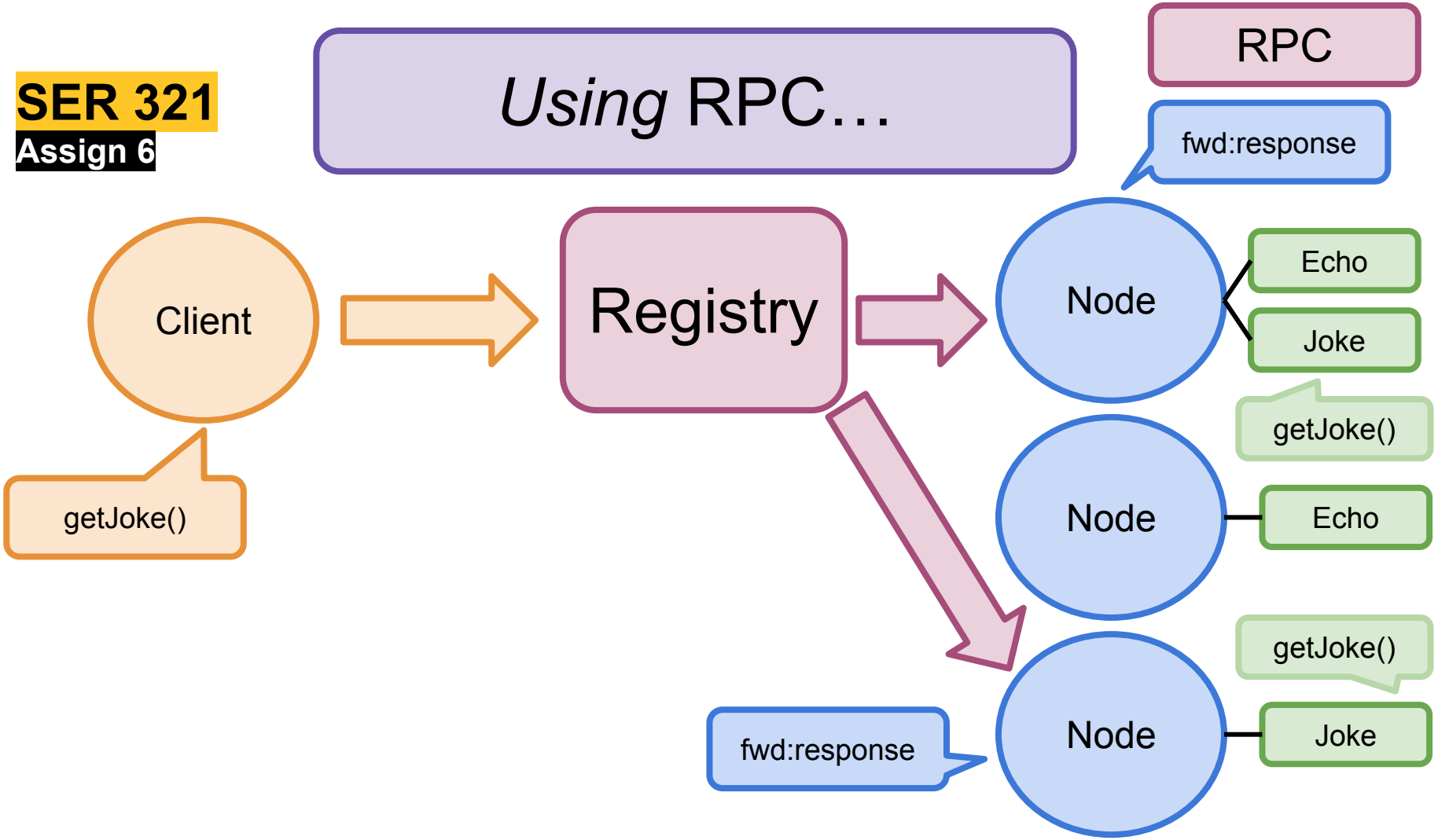
SER 321
Assign 6

Before RPC…



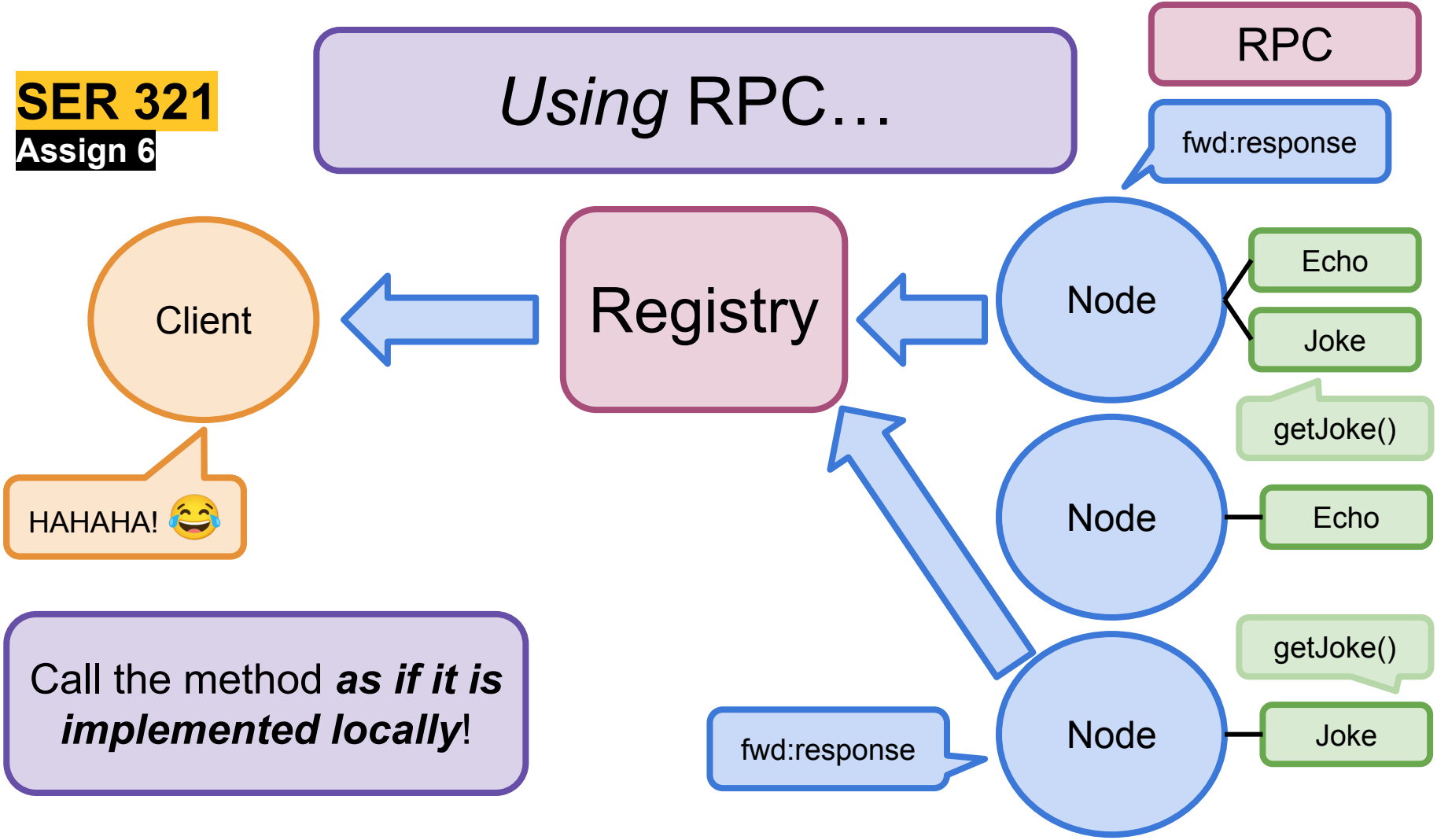
SER 321
Assign 6

Using RPC…



SER 321
Assign 6

Using RPC…



Call the method ***as if it is implemented locally***!

SER 321
Assign 6

Okay so how do we actually *use* this setup?

Client.java

```java
public void setJoke(String joke) {
  JokeSetReq request = JokeSetReq.newBuilder()
      .setJoke(joke).build();
  JokeSetRes response;

  try {
    response = blockingStub2.setJoke(request);
    System.out.println(response.getOk());
  } catch (Exception e) {
    System.err.println("RPC failed: " + e);
    return;
  }
}
```

Client.java (Main)

```java
Client client = new Client(channel, regChannel);

// call the parrot service on the server
client.askServerToParrot(message);

// ask the user for input how many jokes the user wants
BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

// Reading data using readLine
System.out.println("How many jokes would you like?"); // NO ERROR handling of wrong input here.
String num = reader.readLine();

// calling the joked service from the server with num from user input
client.askForJokes(Integer.valueOf(num));

// adding a joke to the server
client.setJoke("I made a pencil with two erasers. It was pointless.");

// showing 6 joked
client.askForJokes(Integer.valueOf(6));
```

JokeImpl.java

```java
// W
@Ove
public void setJoke(JokeSetReq req, StreamObserver<JokeSetRes> responseObserver) {

    System.out.println("Received from client: " + req.getJoke());
    JokeSetRes.Builder response = JokeSetRes.newBuilder();
    if (req.getJoke().isEmpty()) { // we do not want to add empty jokes
        response.setOk(false);
    } else {
        jokes.add(req.getJoke());
        response.setOk(true);
    }

    JokeSetRes resp = response.build();
    responseObserver.onNext(resp);
    responseObserver.onCompleted();
}
```

Looking at **SetJoke**

SER 321
Assign 6

Okay so how do we actually *use* this setup?

```java
public void setJoke(String joke) {
  JokeSetReq request = JokeSetReq.newBuilder()
      .setJoke(joke).build();
  JokeSetRes response;

  try {
    response = blockingStub2.setJoke(request);
    System.out.println(response.getOk());
  } catch (Exception e) {
    System.err.println("RPC failed: " + e);
    return;
  }
}
```

Client.java

Client provides the info

Client creates request by calling the method

```java
Client client = new Client(channel, regChannel);
```

```java
// Implement the joke service. It has two sevices getJokes and setJoke
class JokeImpl extends JokeGrpc.JokeImplBase {  1 usage

    // having a global set of jokes
    Stack<String> jokes = new Stack<~>();  7 usages

    public JokeImpl(){  1 usage
        super();
        // copying some dad jokes
        jokes.add("How do you get a squirrel to like you? Act like a nut.");
        jokes.add("I don't trust stairs. They're always up to something.");
        jokes.add("What do you call someone with no body and no nose? Nobody knows.");
        jokes.add("Did you hear the rumor about butter? Well, I'm not going to spread it!");

    }
```

JokeImpl.java

```java
client.askForJokes(Integer.valueOf(6));
```

```java
public void setJoke(JokeSetReq req, StreamObserver<JokeSetRes> responseObserver) {

    System.out.println("Received from client: " + req.getJoke());
    JokeSetRes.Builder response = JokeSetRes.newBuilder();
    if (req.getJoke().isEmpty()) { // we do not want to add empty jokes
        response.setOk(false);
    } else {
        jokes.add(req.getJoke());
        response.setOk(true);
    }

    JokeSetRes resp = response.build();
    responseObserver.onNext(resp);
    responseObserver.onCompleted();
}
```

JokeImpl.java

Everything else we have had to do is done in the Implementation Class!

SER 321
Assign 6

What does that imply for the system?

```java
public void setJoke(String joke) {
  JokeSetReq request = JokeSetReq.newBuilder()
      .setJoke(joke).build();
  JokeSetRes response;

  try {
    response = blockingStub2.setJoke(request);
    Sys
  } cat
    Sys
    ret
  }
}
```

Client.java

Client Class acts like a middleman!

Everything else we have had to do is done in the Implementation Class!

```java
Client client = new Client(channel, regChannel);

// call the parrot service on the server
client.askServerToParrot(message);

// ask the user for input how many jokes the user wants
BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

// Reading data using readLine
System.out.println("How many jokes would you like?"); // NO ERROR handling of wrong input here.
String num = reader.readLine();

// calling the joked service from the server with num from user input
client.askForJokes(Integer.valueOf(num));

// adding a joke to the server
client.setJoke("I made a pencil with two erasers. It was pointless.");

// showing 6 joked
client.askForJokes(Integer.valueOf(6));
```

Client.java (Main)

```java
// W
@Ove
public void setJoke(JokeSetReq req, StreamObserver<JokeSetRes> responseObserver) {

    System.out.println("Received from client: " + req.getJoke());
    JokeSetRes.Builder response = JokeSetRes.newBuilder();
    if (req.getJoke().isEmpty()) { // we do not want to add empty jokes
        response.setOk(false);
    } else {
        jokes.add(req.getJoke());
        response.setOk(true);
    }

    JokeS
    respo
    respo
}
```

JokeImpl.java

Implementations need to be robust and thorough!

SER 321

Scratch Space

Upcoming Events

SI Sessions:

- ~~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**~~

FINAL EXAM SCHEDULE:

Opens: Wednesday April 30th @ 12:30 AM

Closes: Friday May 2nd @ 11:59:59 PM

Questions?

Survey:

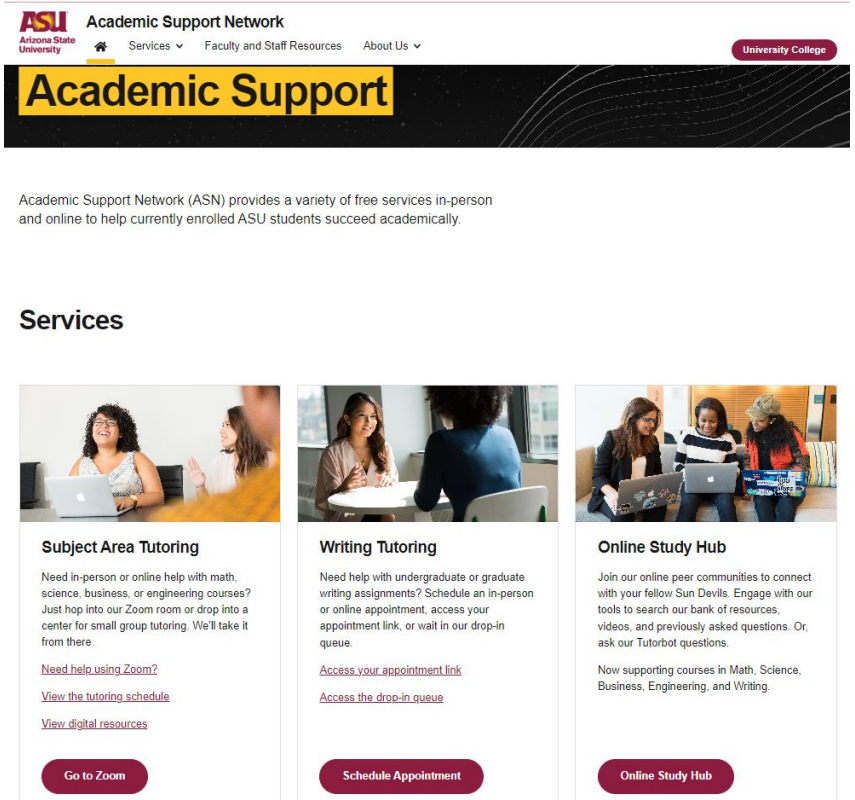
https://asuasn.info/ASNSurvey

62

More Questions?

Check out our other resources!

tutoring.asu.edu



1 – Go to Zoom

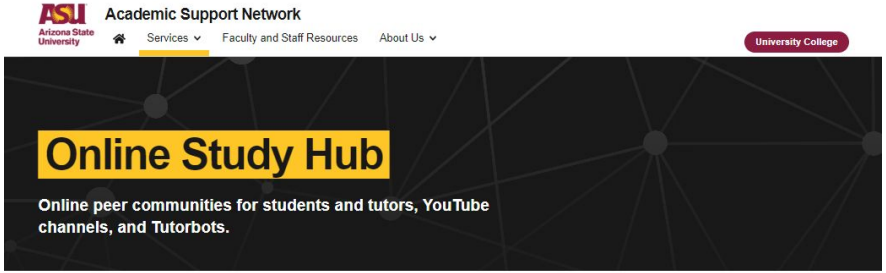
2- Need help using Zoom?
View the tutoring schedule
View digital resources

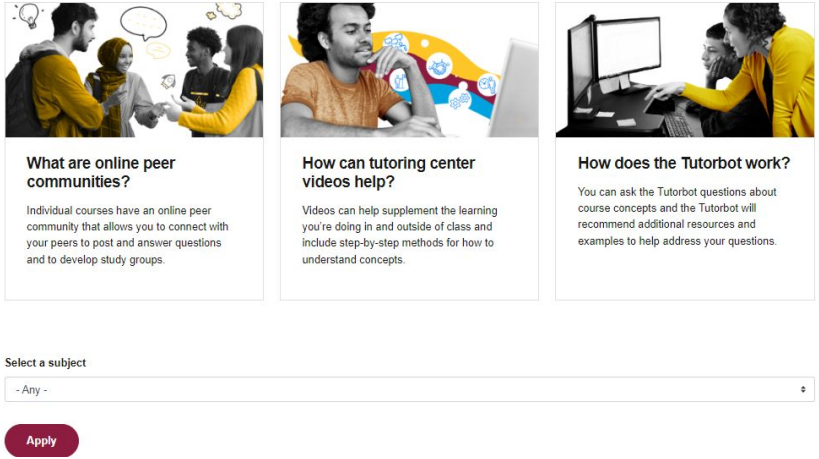
1. **Click on 'Go to Zoom' to log onto our Online Tutoring Center.**
2. **Click on 'View the tutoring schedule' to see when tutors are available for specific courses.**

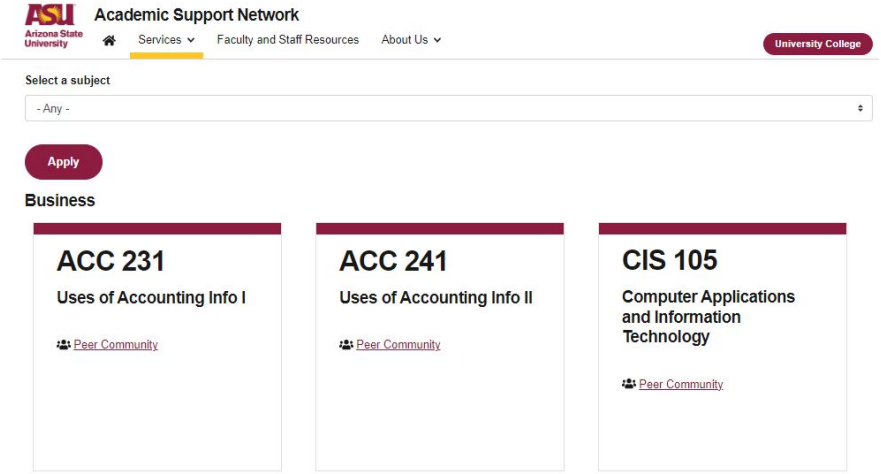
More Questions?

Check out our other resources!

tutoring.asu.edu/online-study-hub



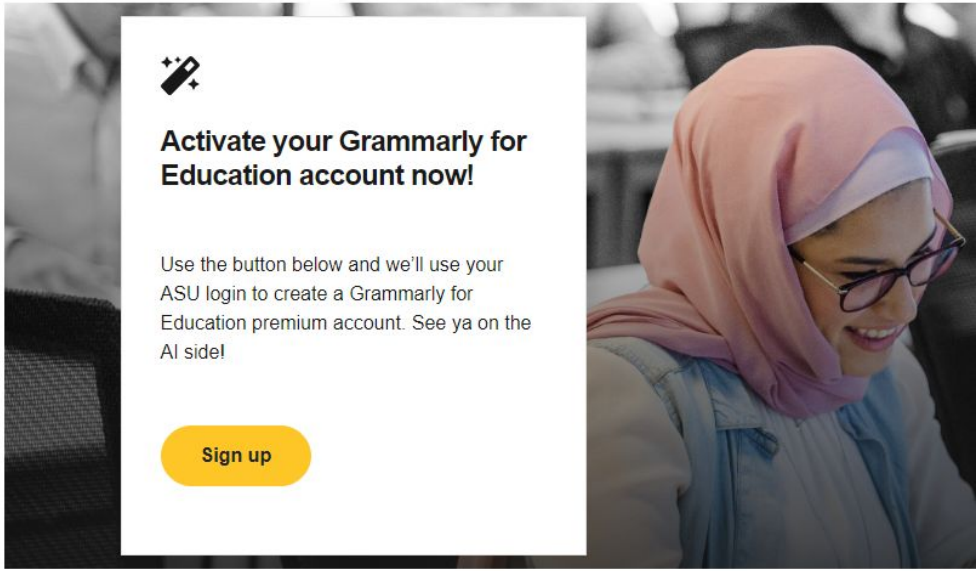




Don't forget to check out the Online Study Hub for additional resources!

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