SER 321 B Session

SI Session

Tuesday, December 3rd 2024

10:00 am - 11:00 am MST

Agenda

- Assignment 6
  - Registry
  - RPC
- Middleware Review
- Distribution 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

Assign 6

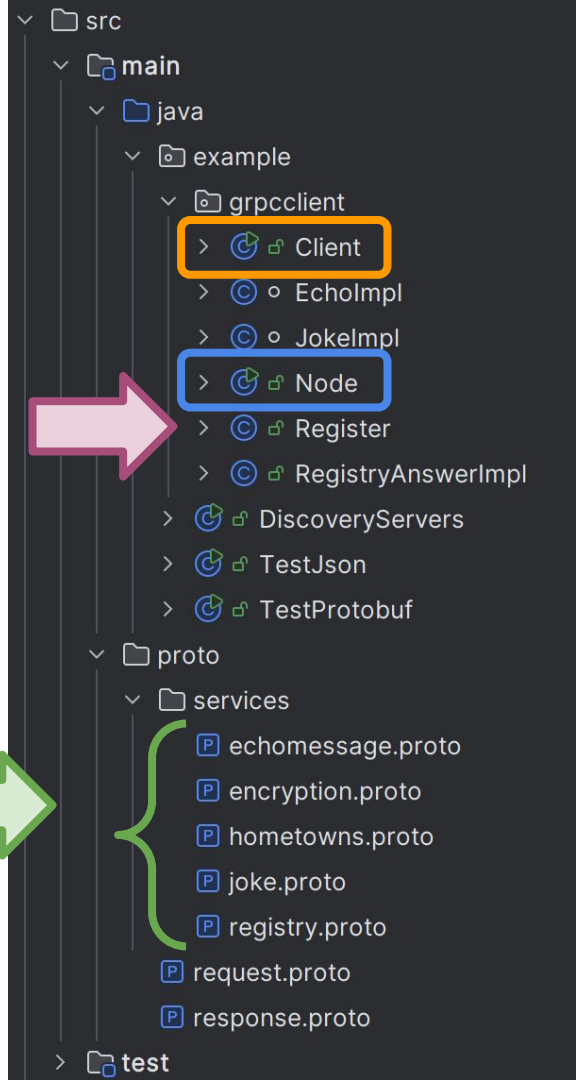
Client

Node

Registry

Service

Protocol Buffers!

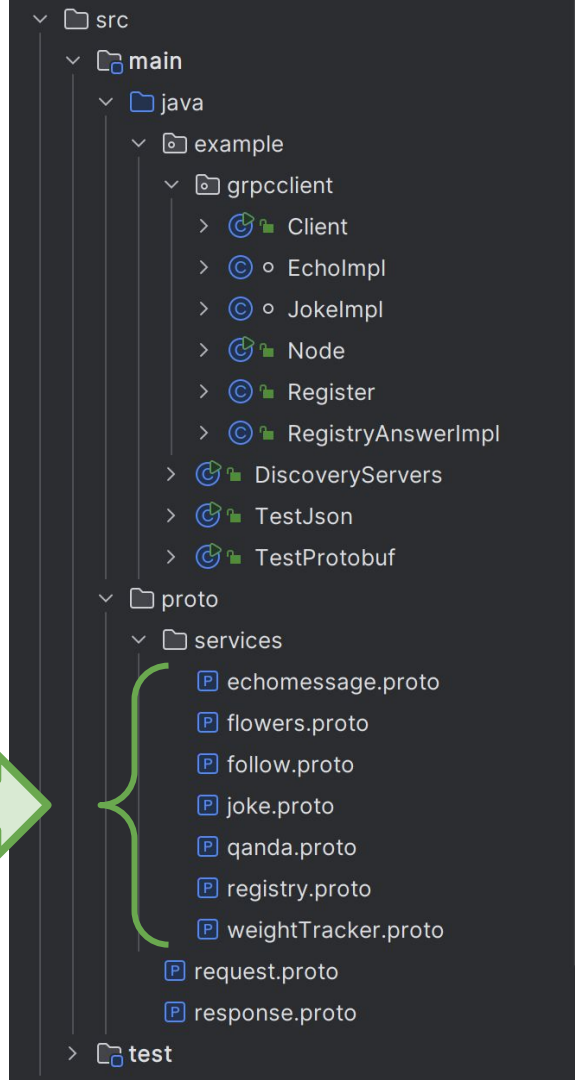


SER 321
Protobuf Review

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

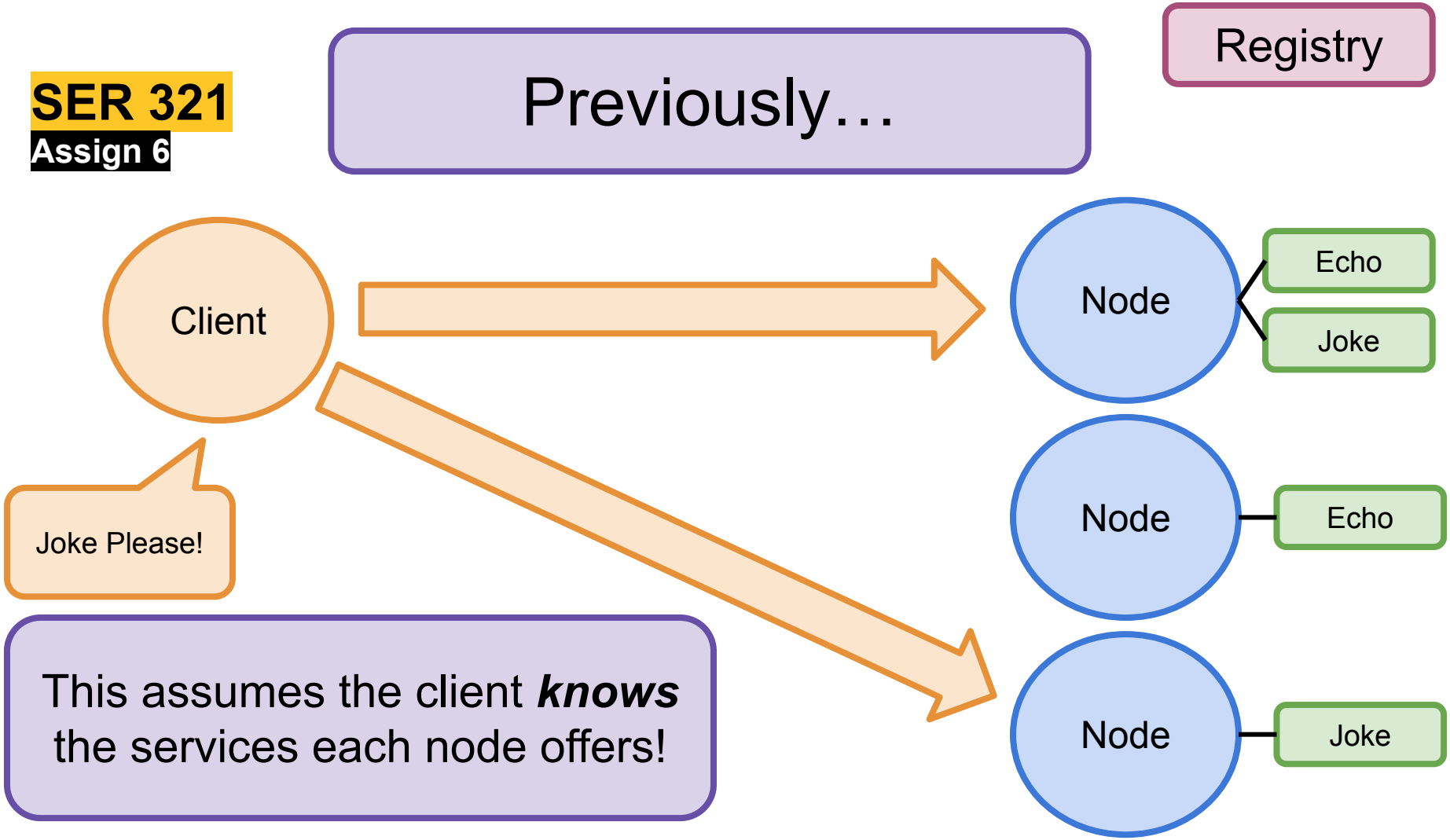
Two new concepts!

Registry

RPC

SER 321
Assign 6

Previously…



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

Registry

SER 321
Assign 6

With the 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

Node

Echo

Node

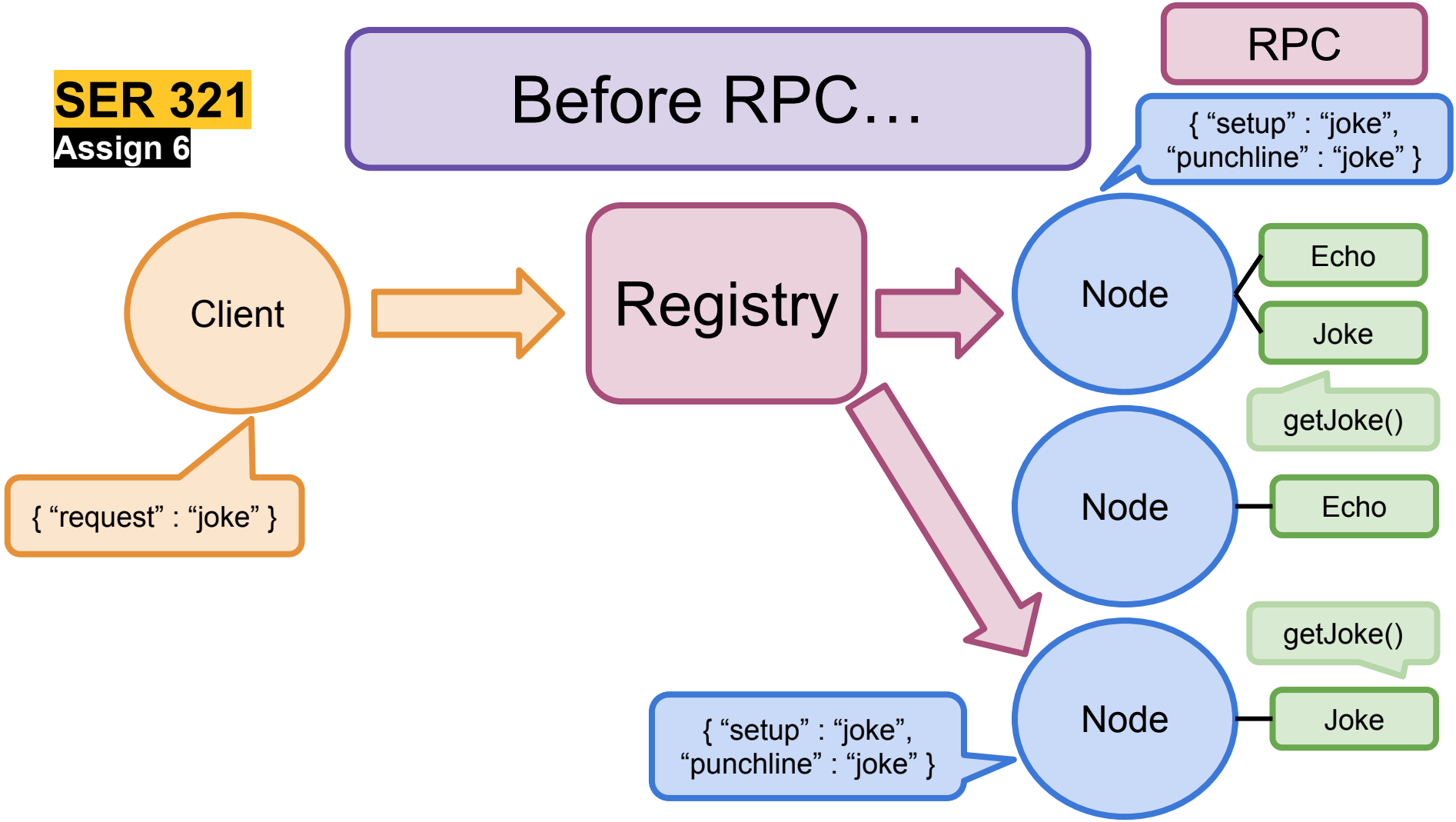
Joke

Joke Please!

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

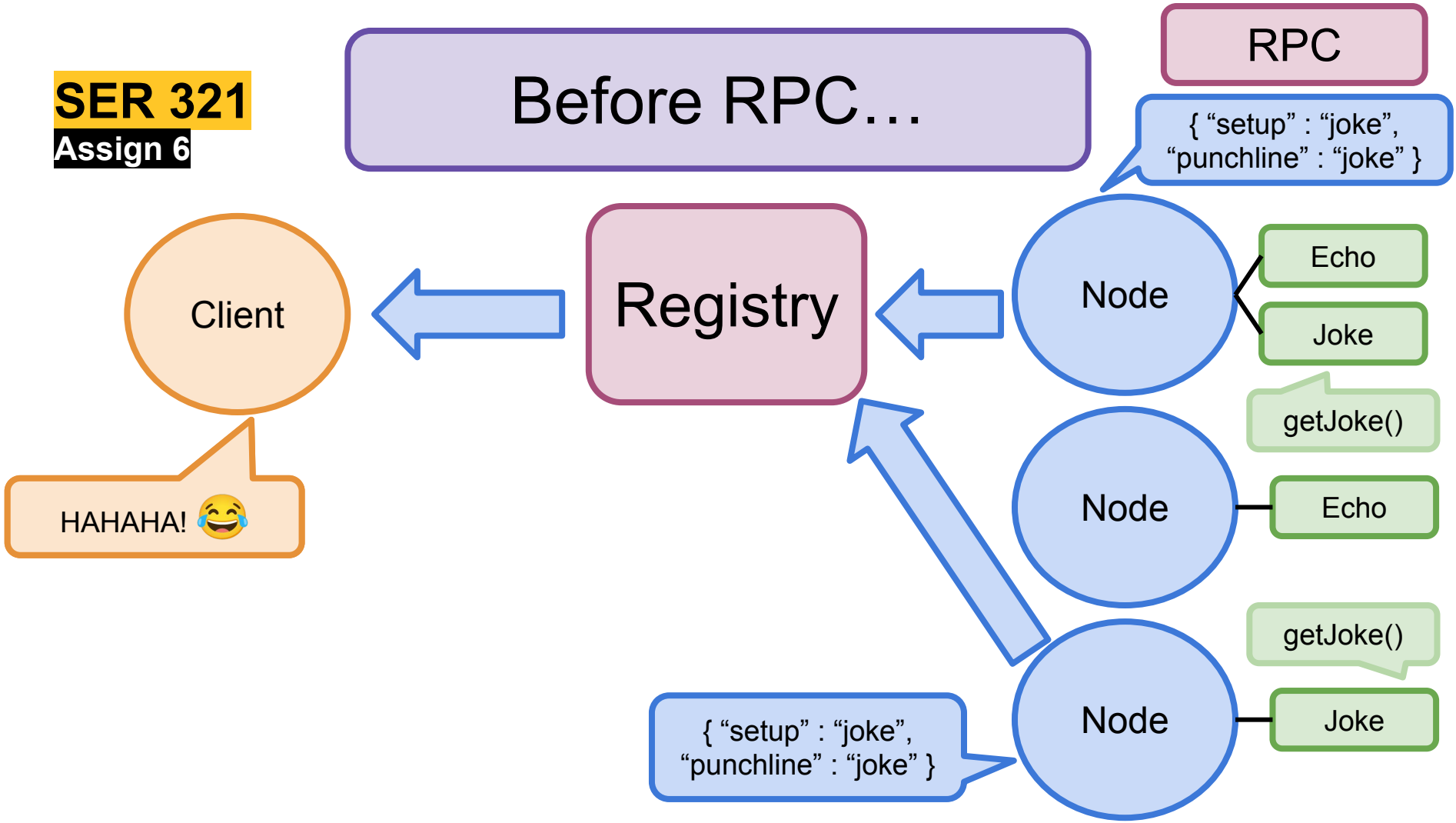
SER 321
Assign 6

Before RPC…



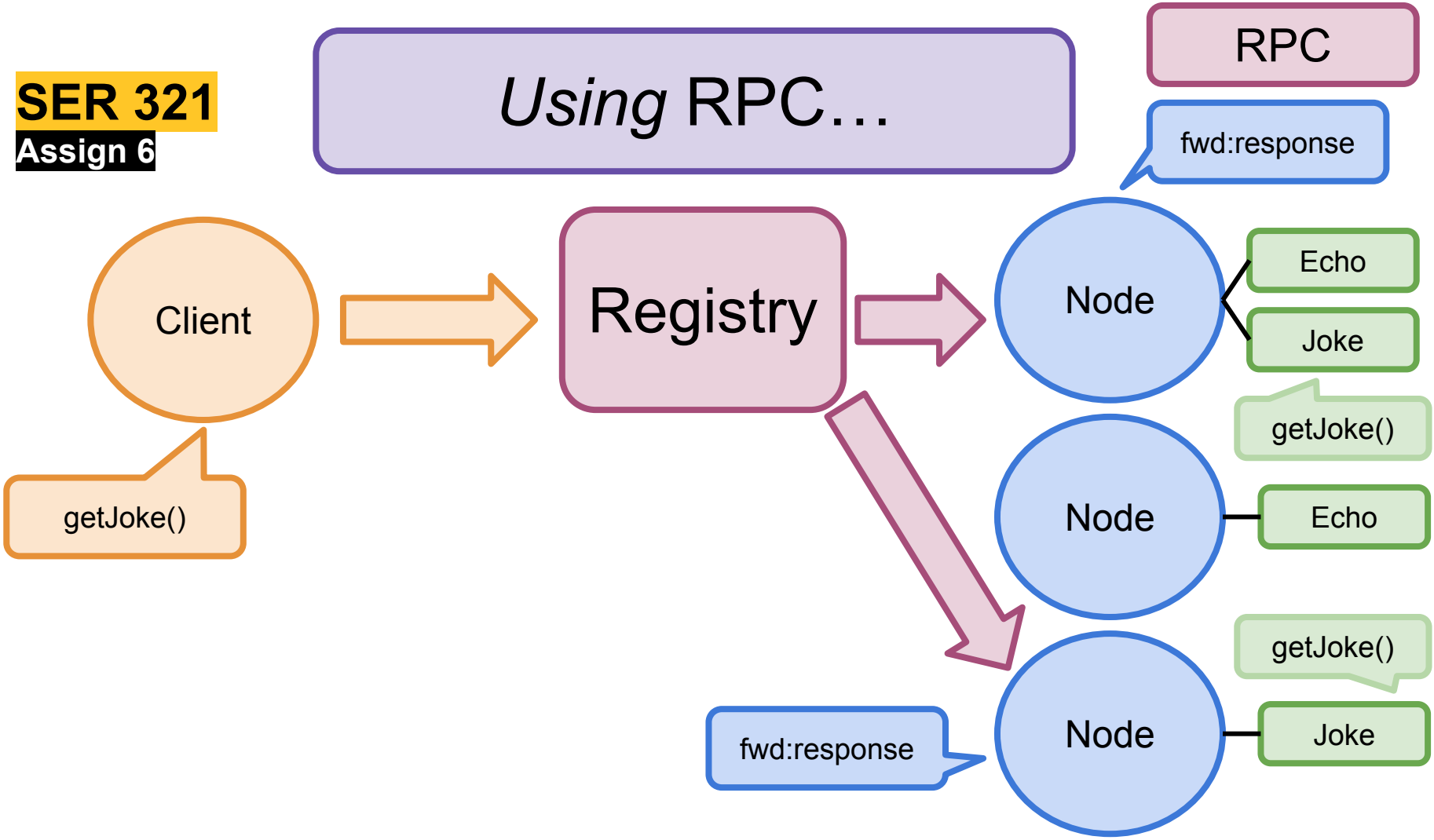
SER 321
Assign 6

Before RPC…



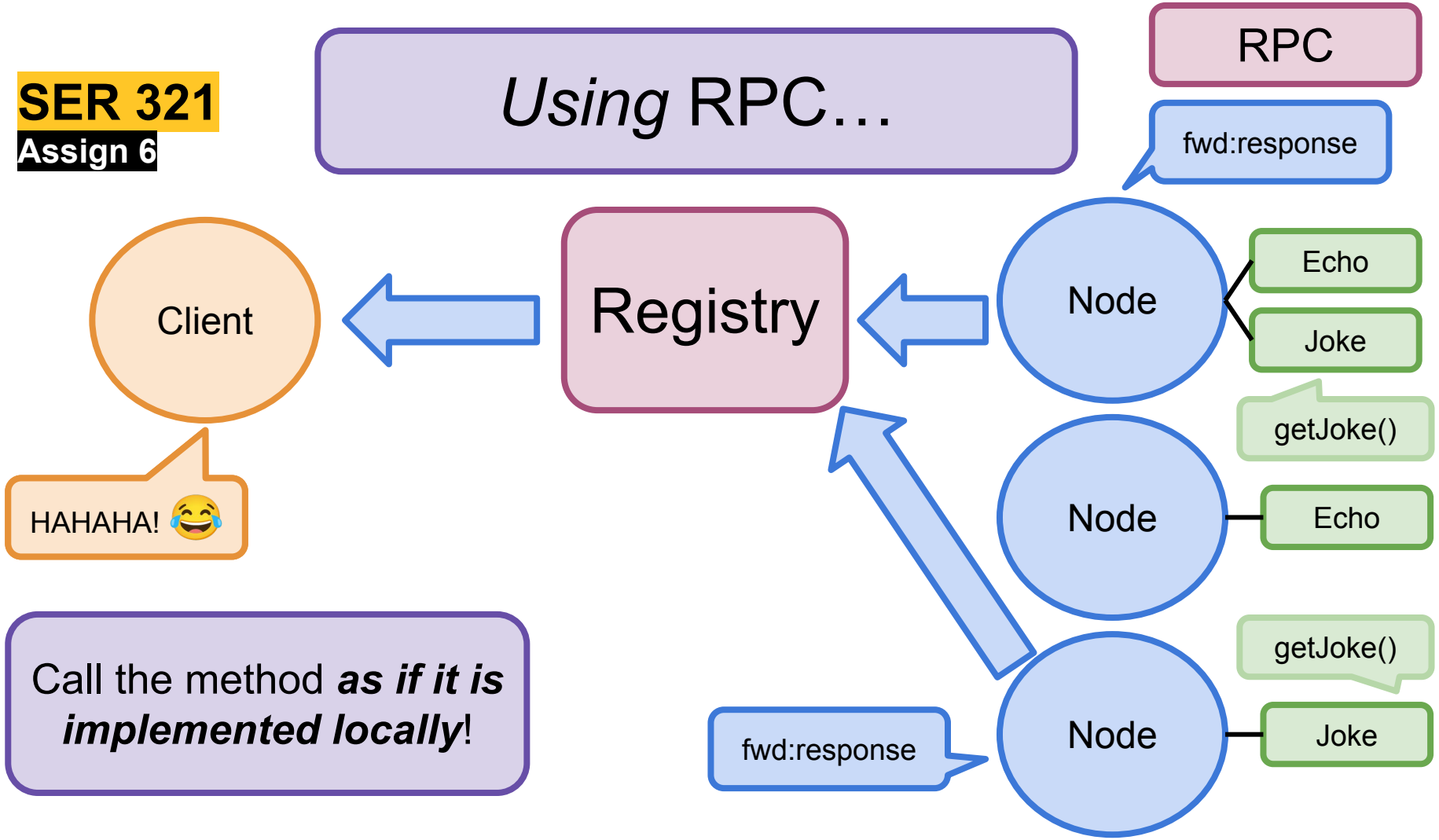
SER 321
Assign 6

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?

```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

```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);
    }

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

JokeImpl.java

Looking at **SetJoke**

SER 321
Assign 6

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

```
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

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

```
// 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

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

```
@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();
}
```

JokeImpl.java

Everything else we have had to do is handled 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!

```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

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

Implementations need to be robust and thorough!

Check out the recording for the discussion!

SER 321
Middleware

We have been:

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

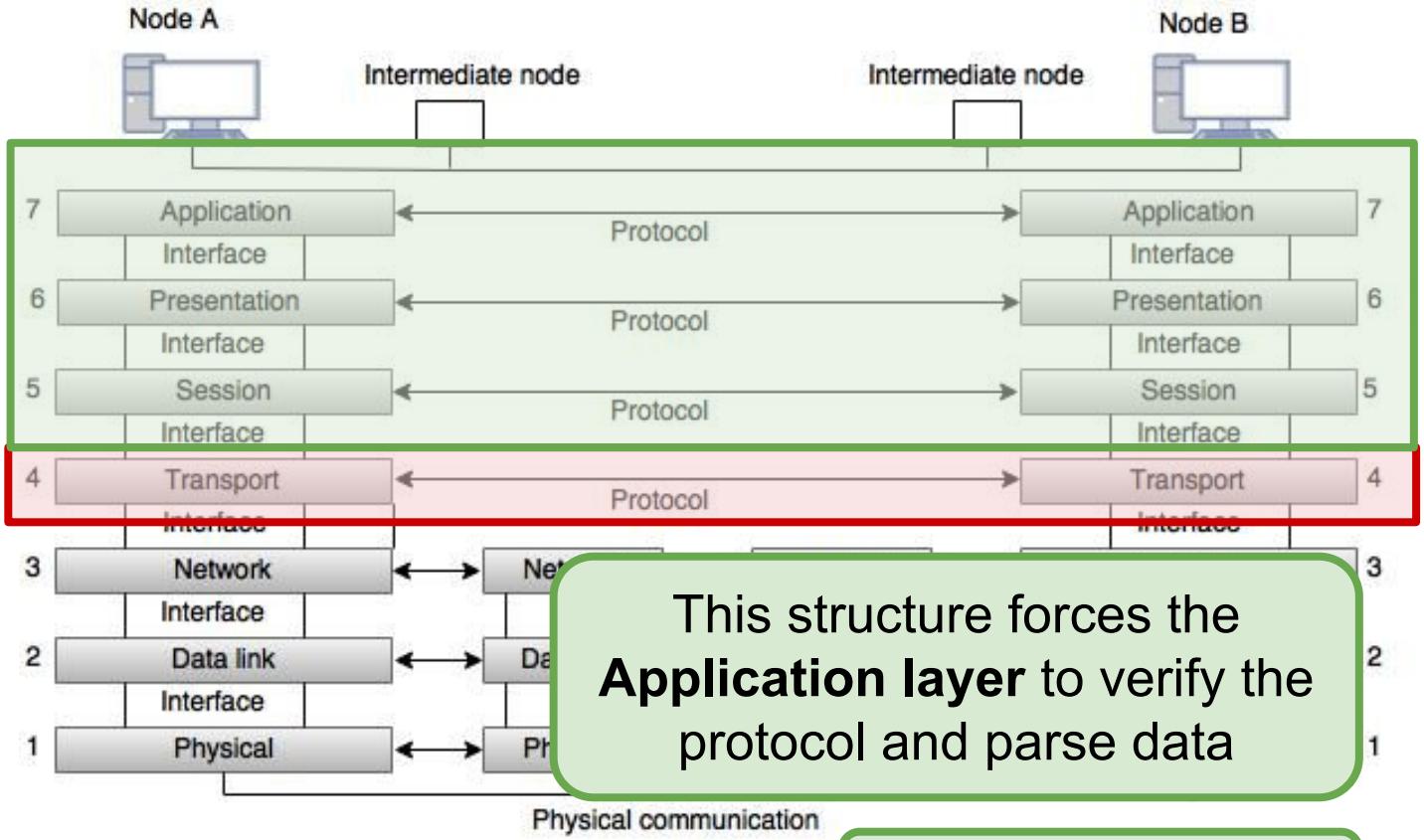


Fig: OSI Model

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

Not really its job…

Check out the recording for the discussion!

SER 321
Middleware

With Middleware:

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

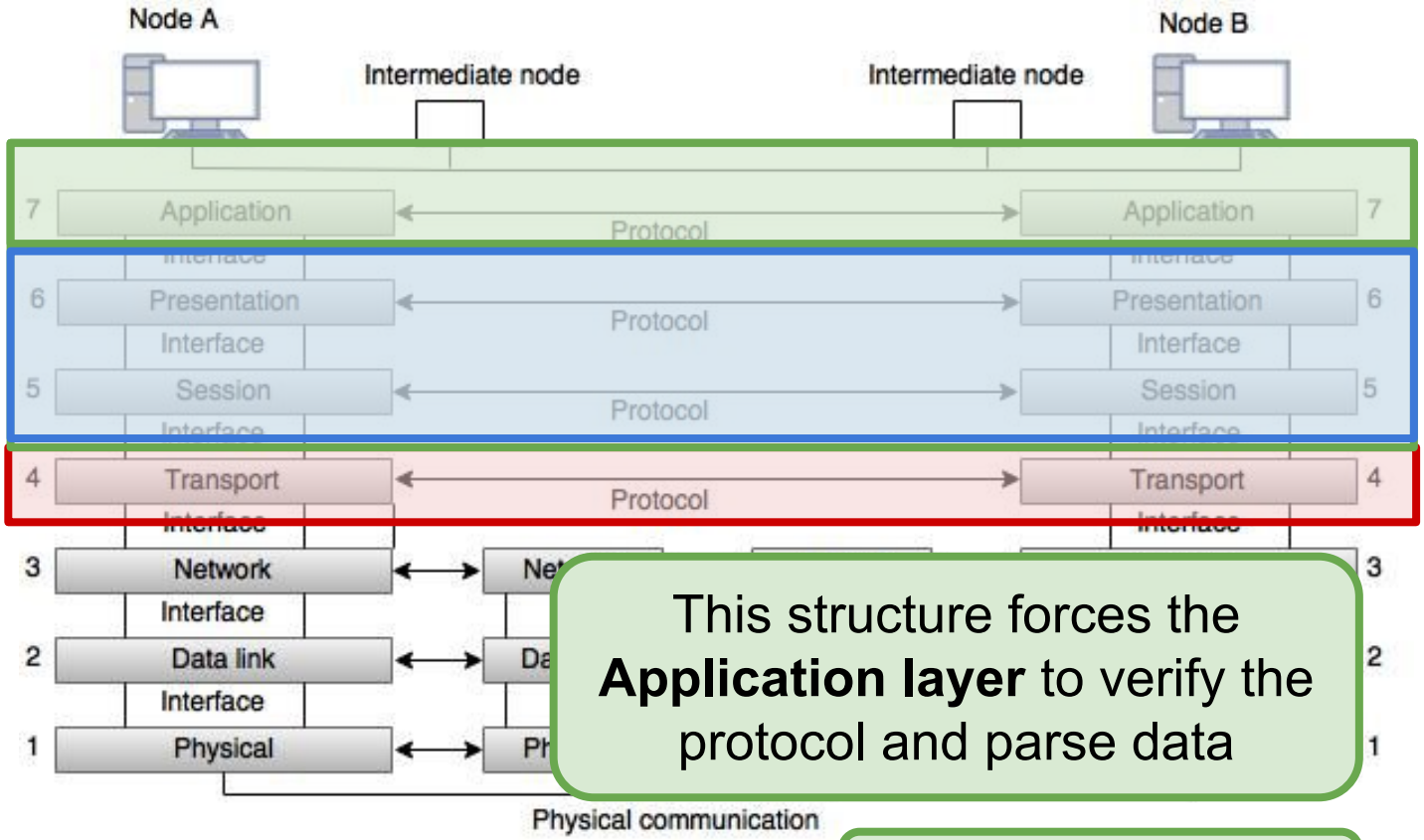


Fig: OSI Model

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

Not really its job…

Check out the recording for the discussion!

SER 321
Middleware

With Middleware:

Serializing Messages

Sending Messages

Parsing Messages

Handle Messages

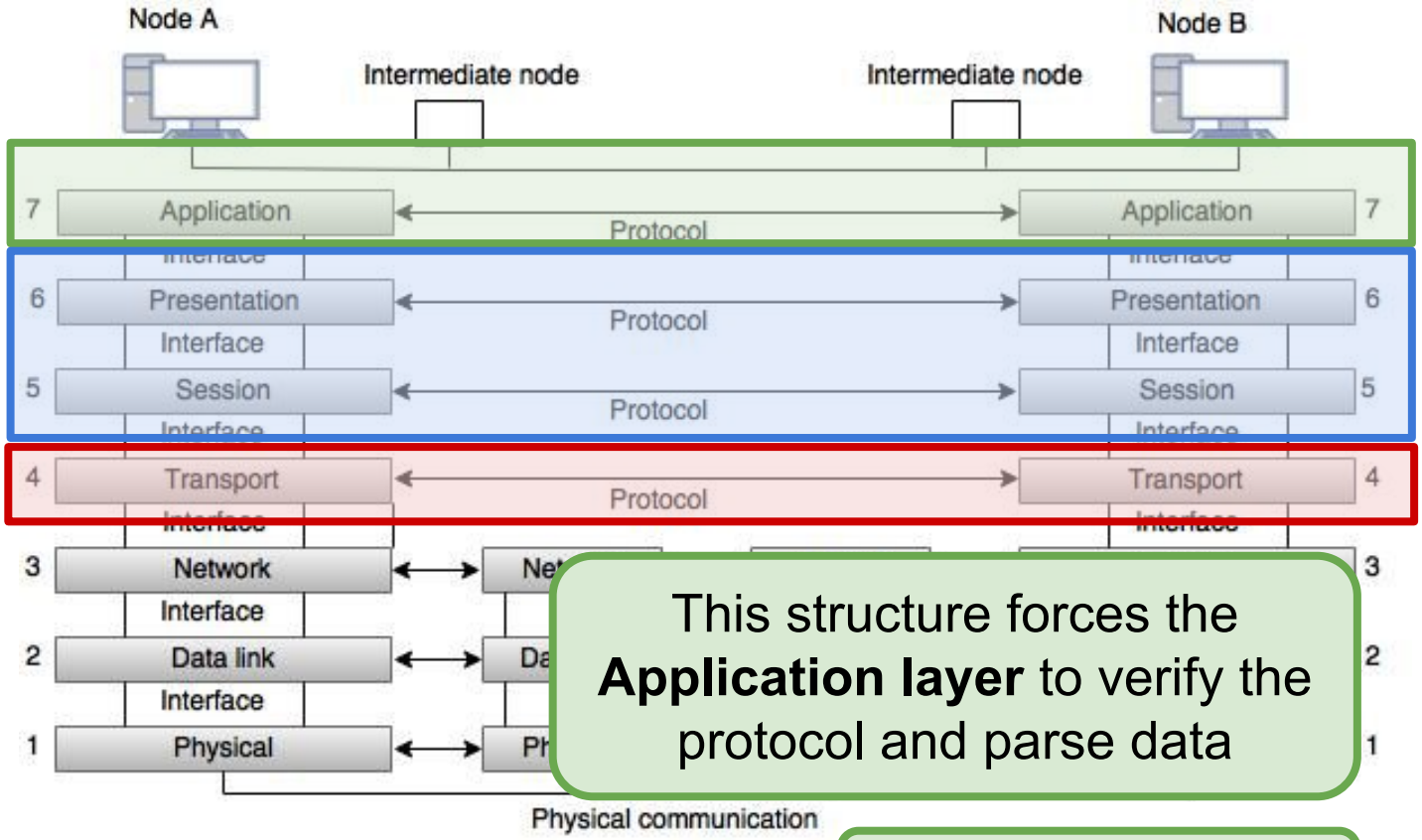


Fig: OSI Model

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

Not really its job…

Check out the recording for the discussion!

SER 321
Middleware

Middleware:

Session Layer Responsibilities:

Authentication

Authorization

Session Management

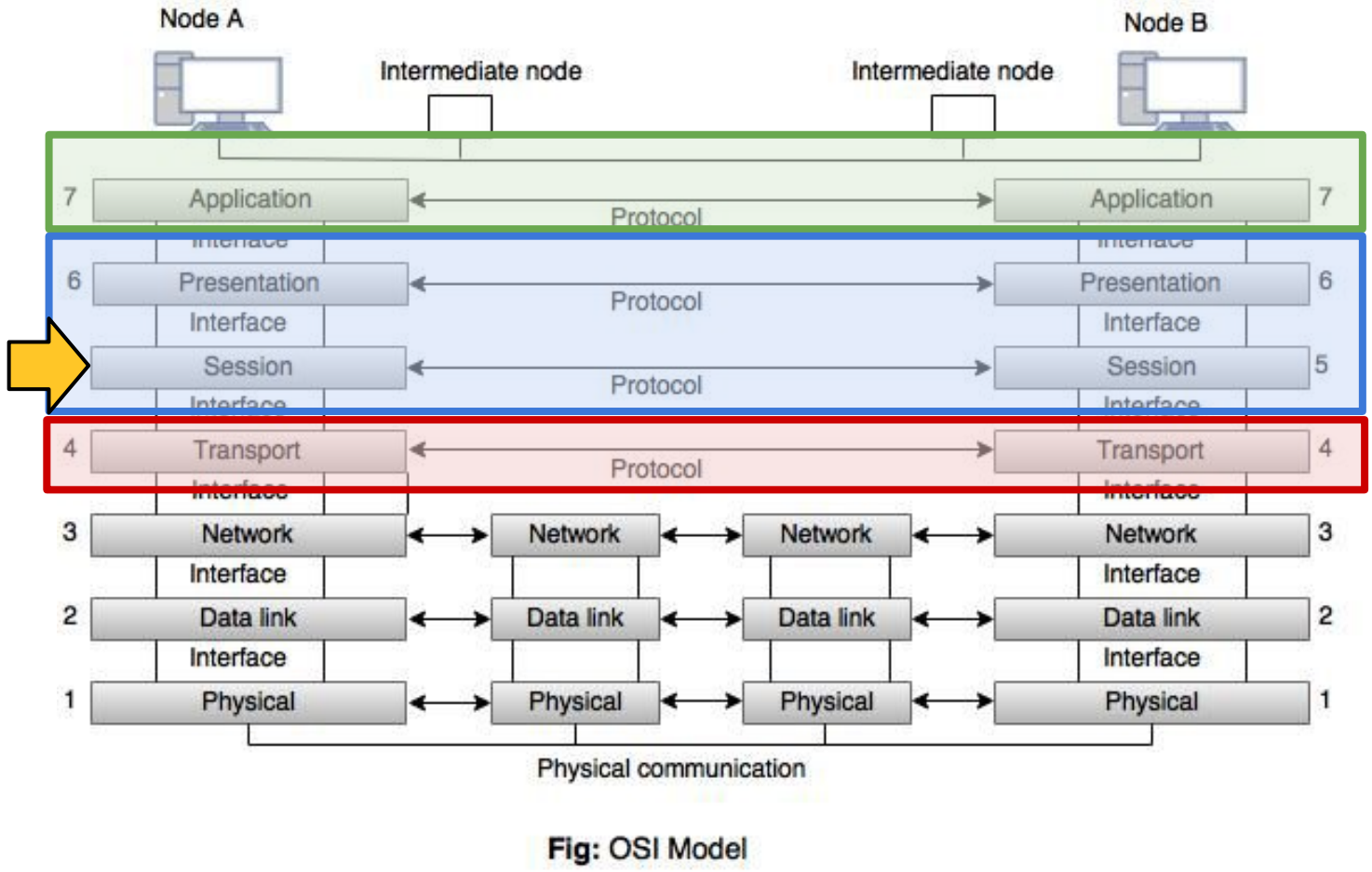


Fig: OSI Model

Check out the recording for the discussion!

SER 321
Middleware

Middleware:

Presentation Layer Responsibilities:

Translation

Compression

Encryption

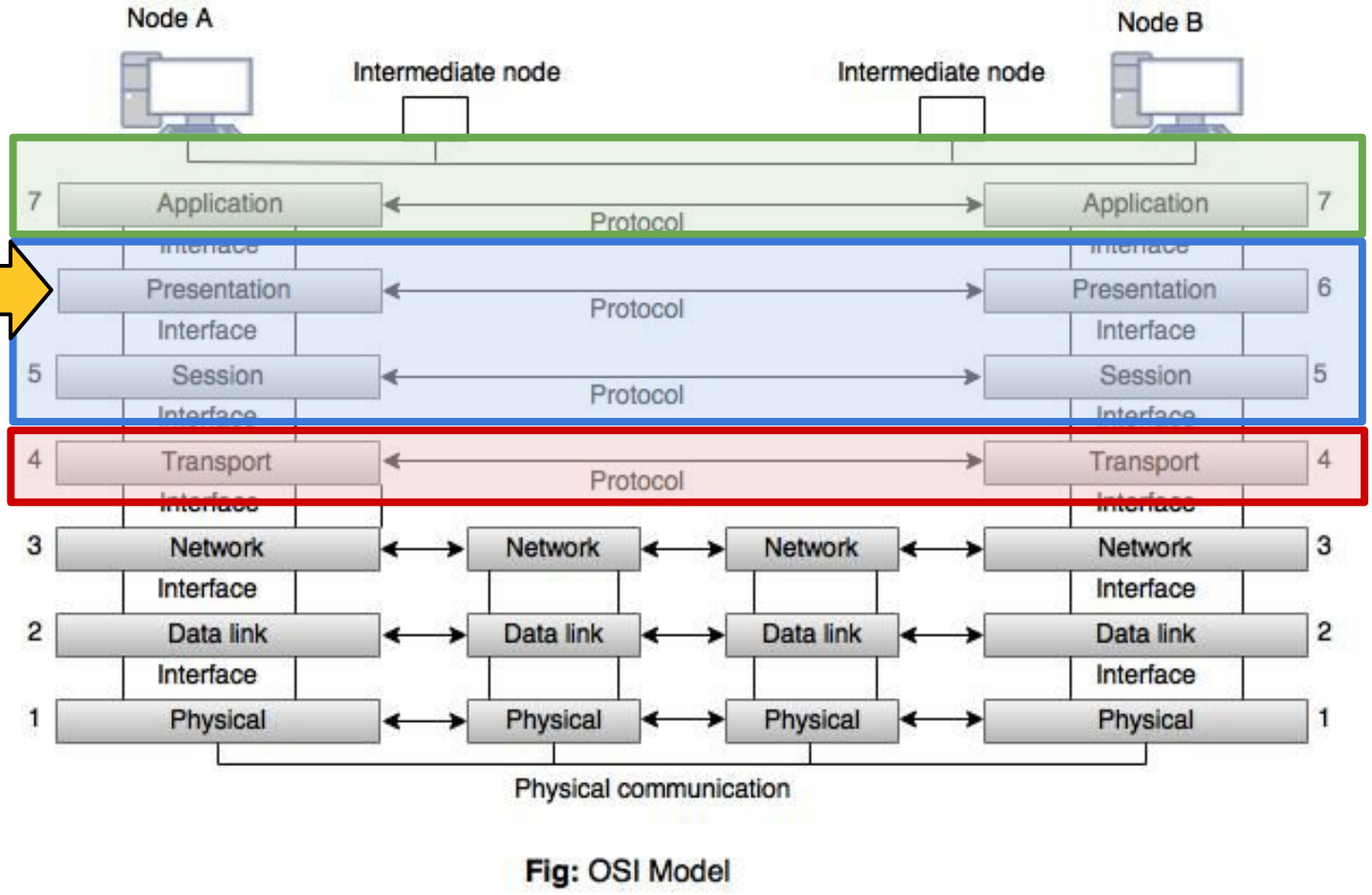


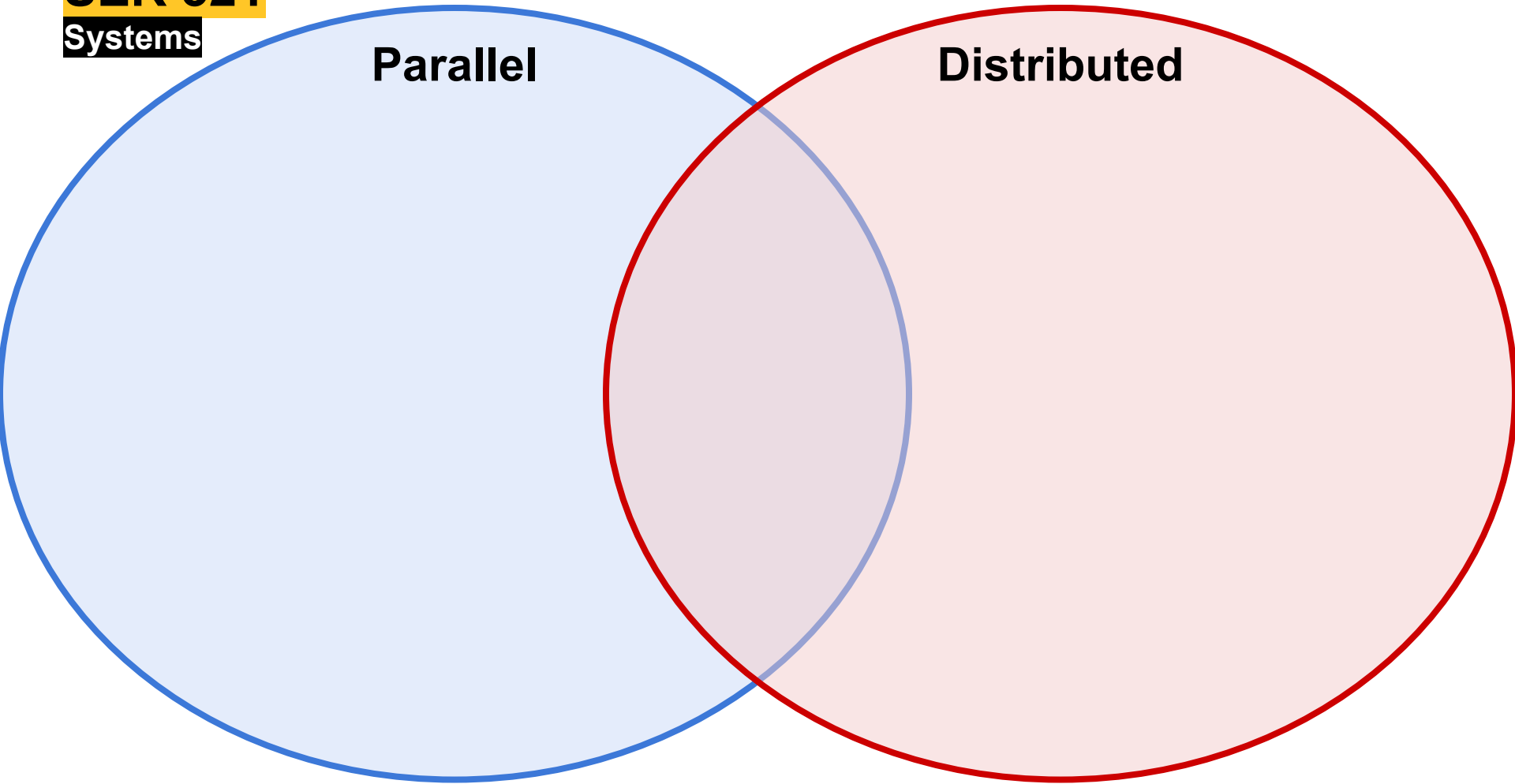
Fig: OSI Model

SER 321
Middleware Benefits

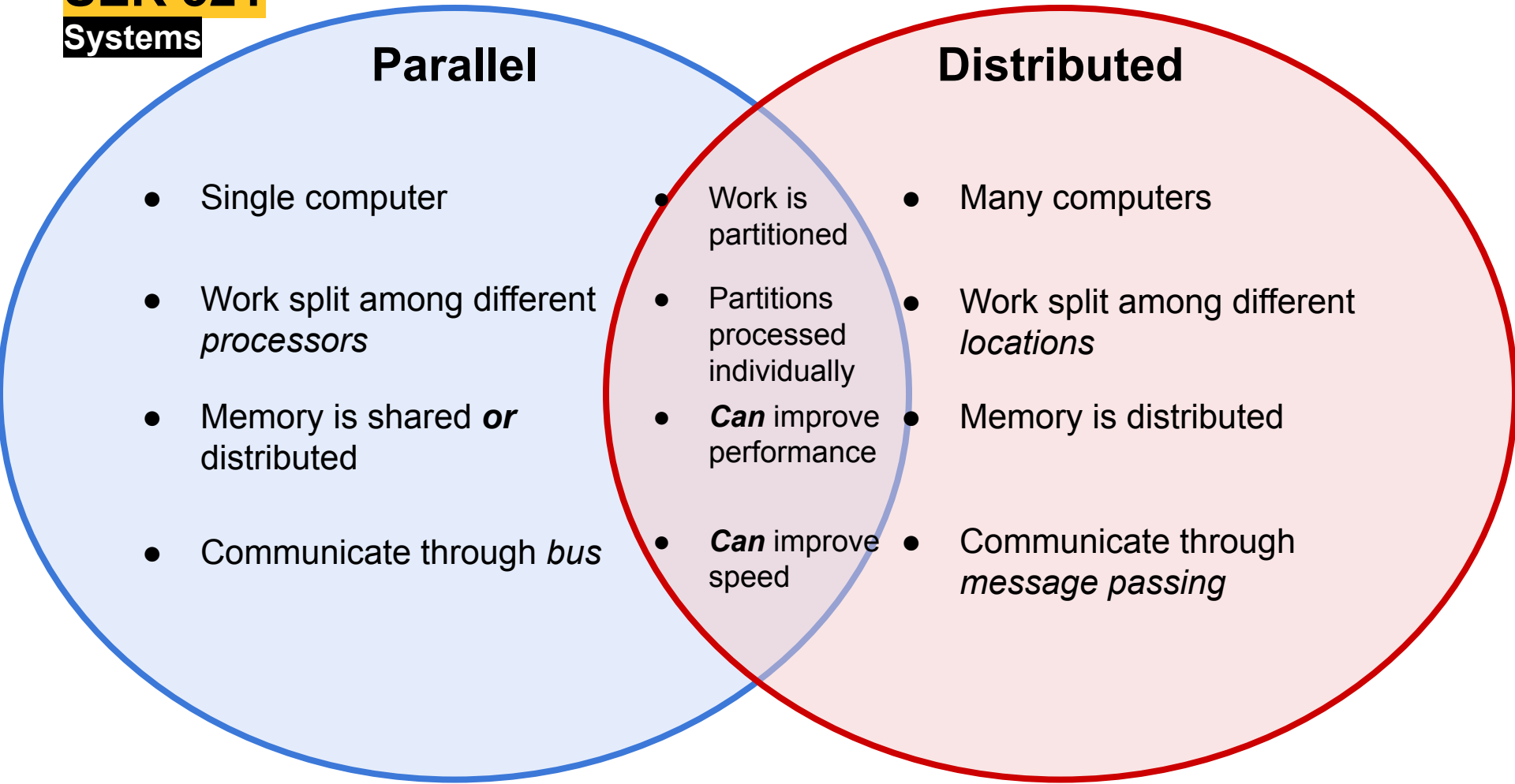
Why do we care?

- Agility
- Reusability
- Efficiency
- Cost Effectiveness
- Portability

SER 321
Systems



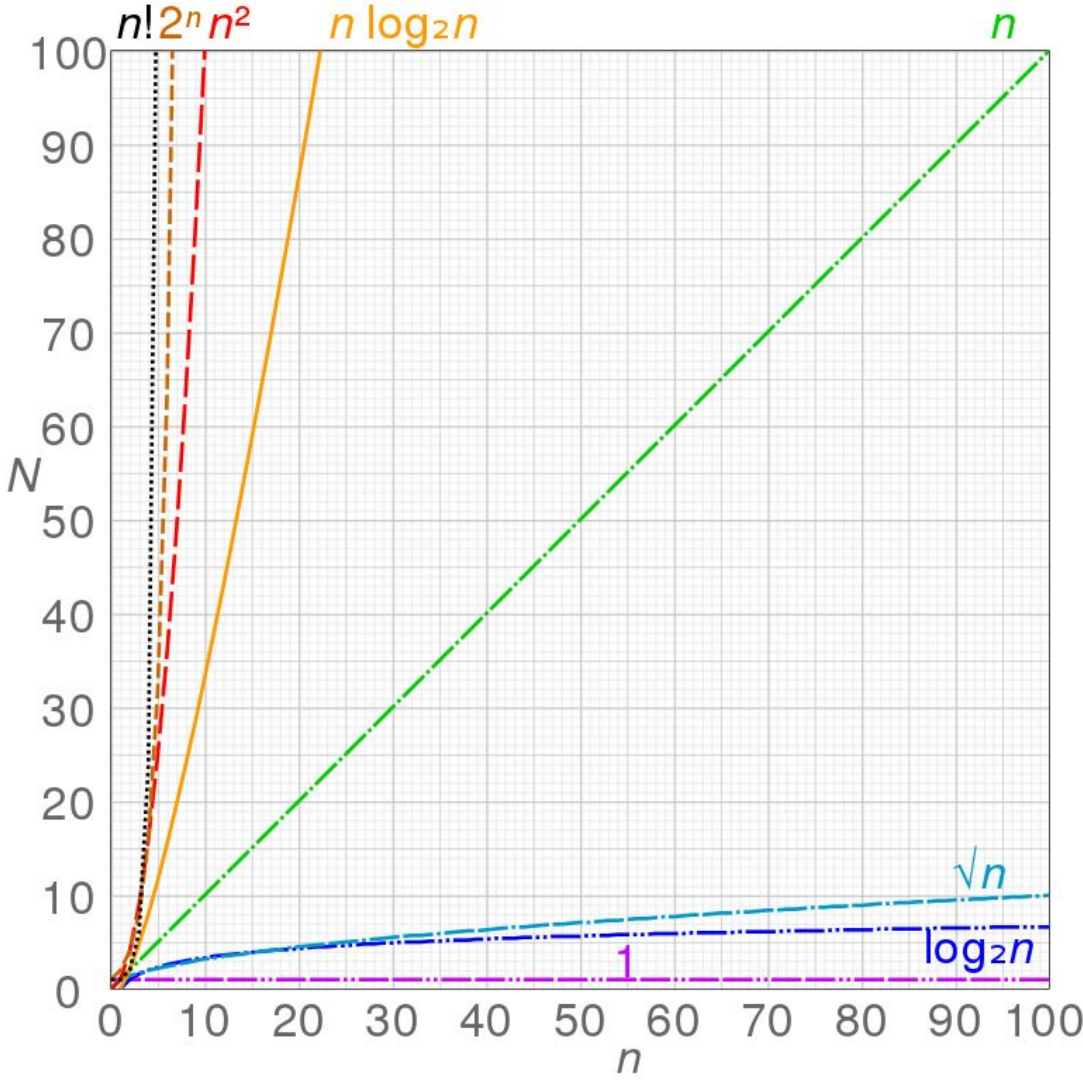
SER 321
Systems



SER 321

When to Distribute

When should we *consider* distributing?

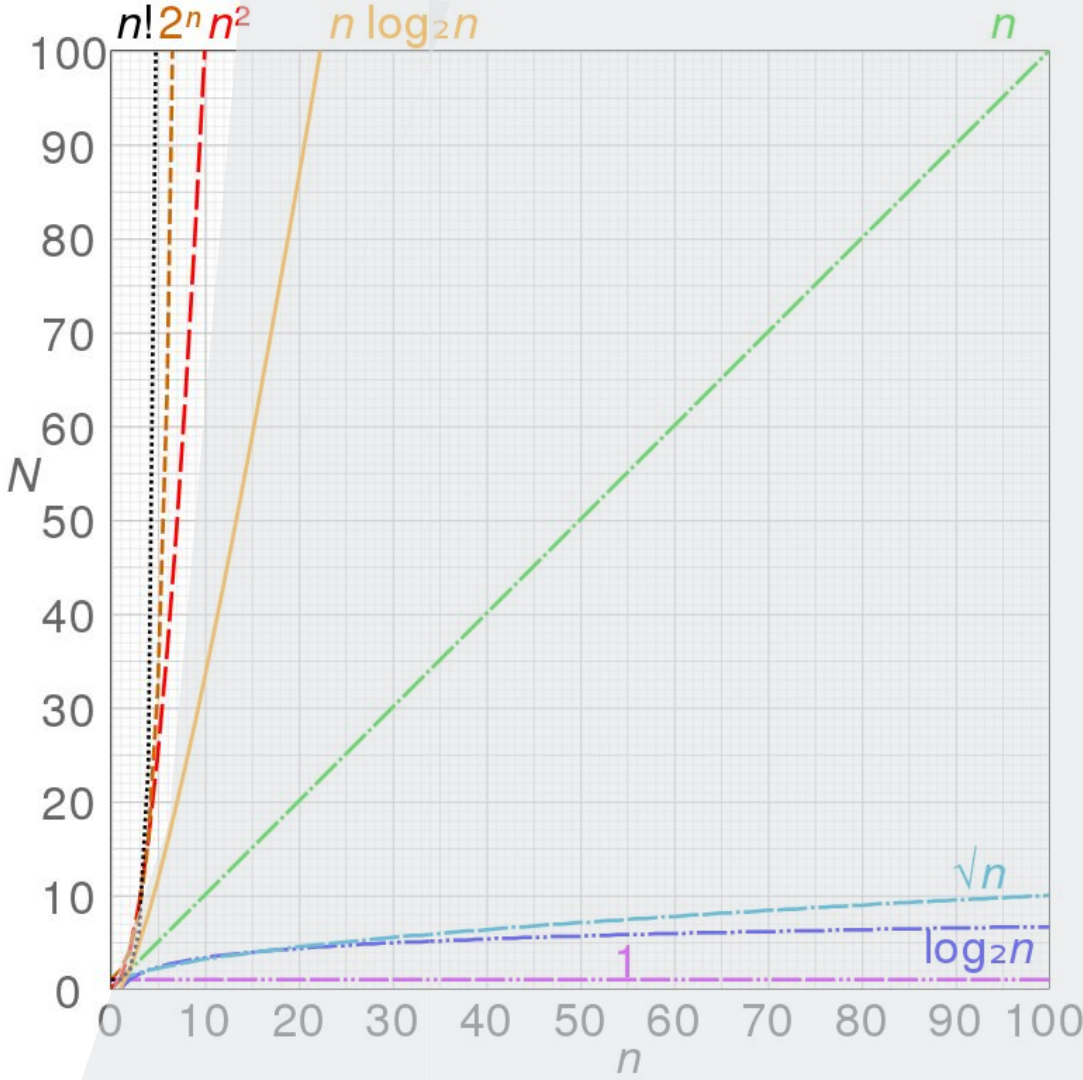


SER 321

When to Distribute

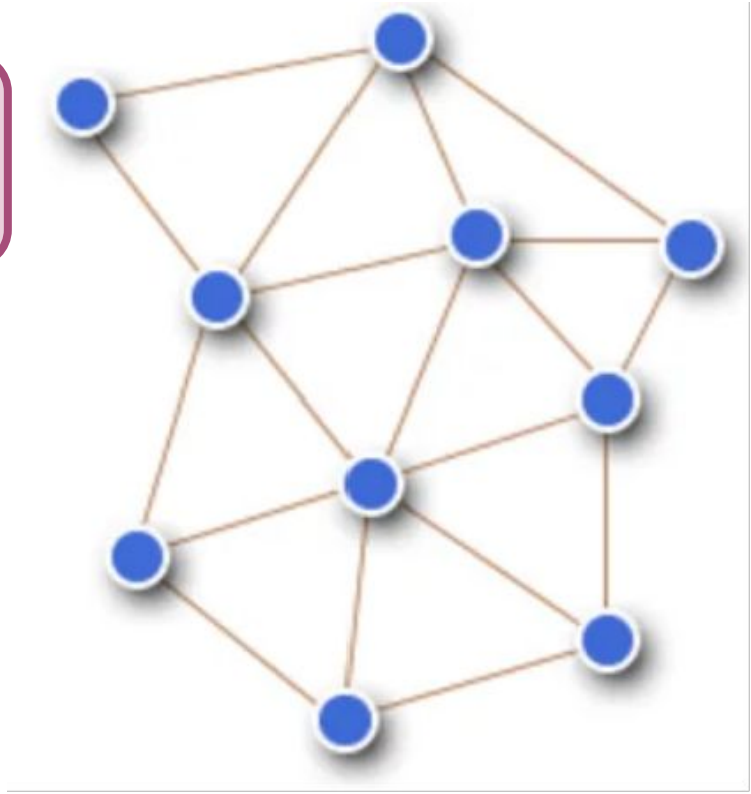
When should we *consider* distributing?

Super Duper Extra Extra Large Orders of Magnitude!



SER 321
Distributed Issues

Remember that we are operating in ***reality***

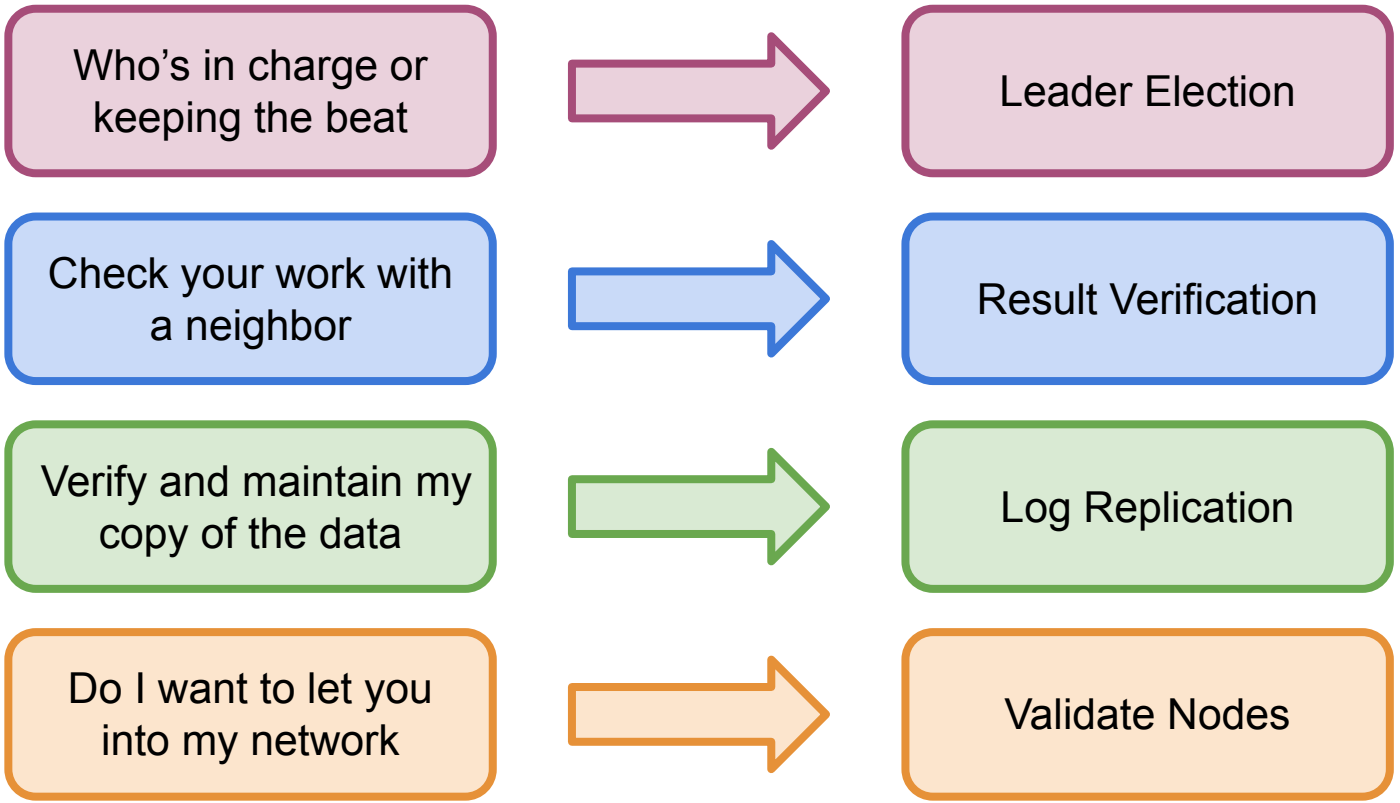


- Nodes *will* fail
- Web of nodes *will constantly* change
- Network is not *always* reliable
- Latency is *always present*
- The path traversed *changes*
- Some resources *must be shared*
- *You* need to prevent the pitfalls!
  - No deadlocks
  - No starvation
  - No error states

SER 321
Consensus

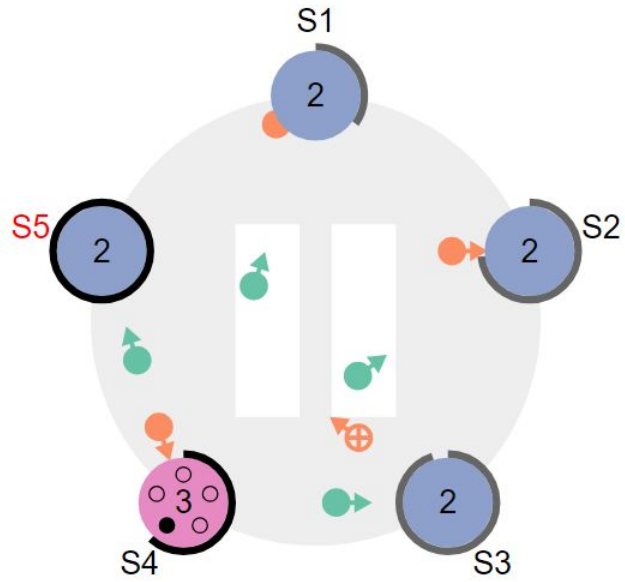
"General agreement or trust amongst a group"

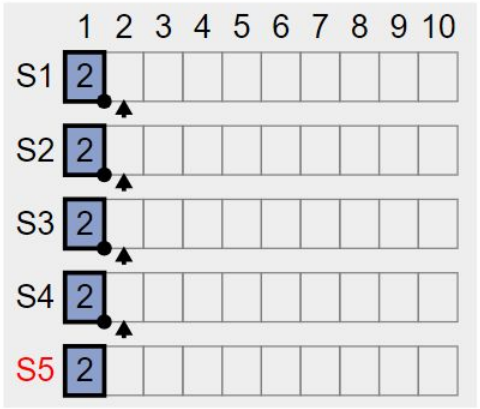
What is Consensus?



SER 321
RAFT

How do we feel about Consensus?





RAFT

SER 321

Scratch Space

Questions?

Survey:

http://bit.ly/ASN2324

34

Upcoming Events

SI Sessions:

- N/A - **Good luck on the final, you've got this!** 

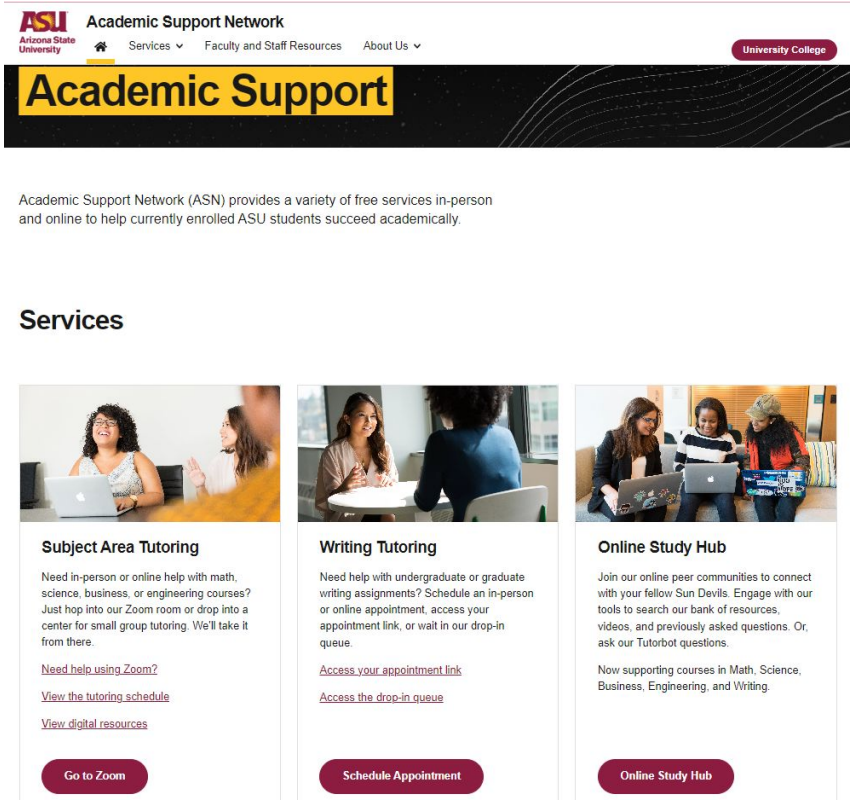
Review Sessions:

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

More Questions?

Check out our other resources!

tutoring.asu.edu

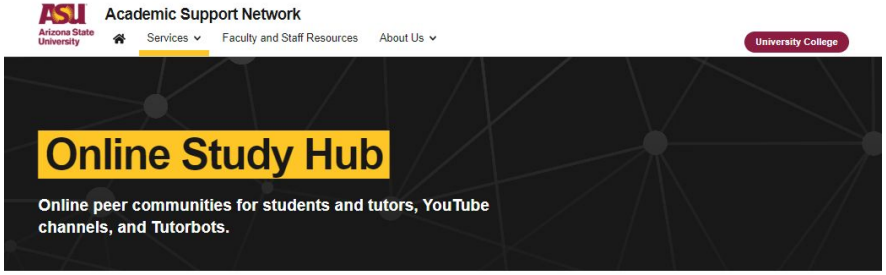


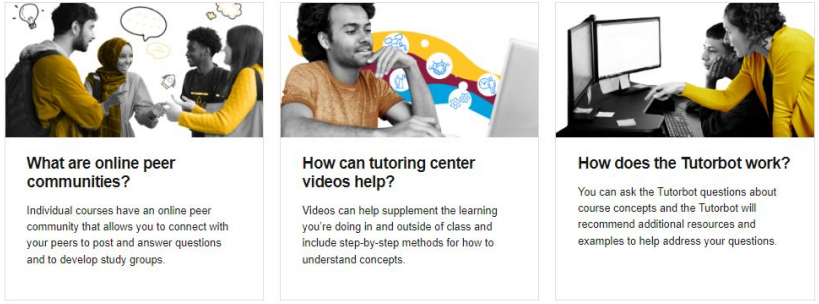
1. **Click on ‘Go to Zoom’ to log onto our Online Tutoring Center.**
2. **Click on ‘View the tutoring schedule’ to see when tutors are available for specific courses.**

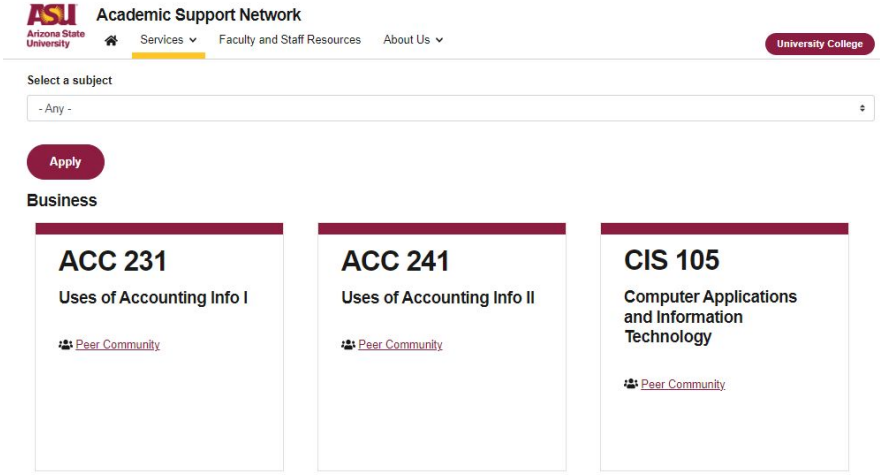
More Questions?

Check out our other resources!

tutoring.asu.edu/online-study-hub



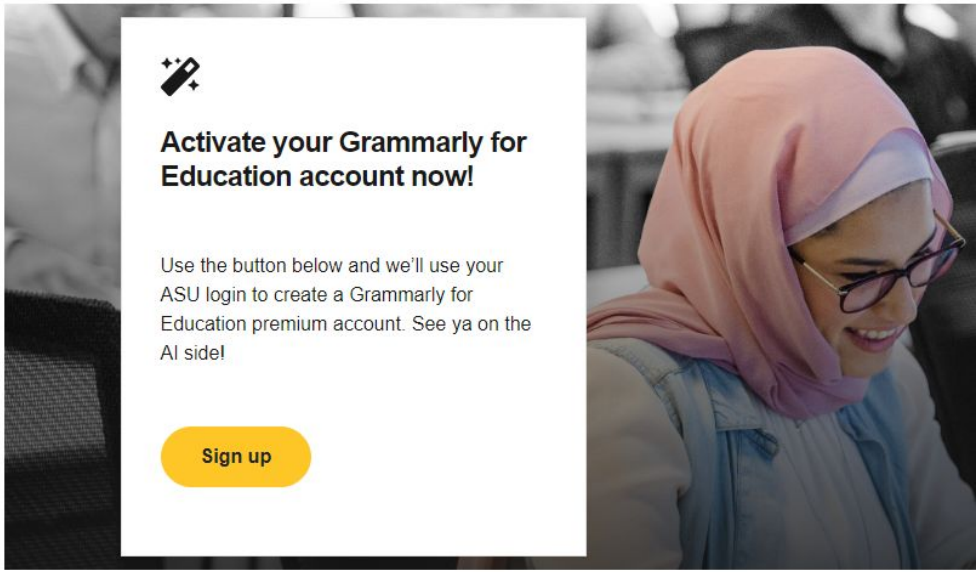




Don't forget to check out the Online Study Hub for additional resources!

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