SER 321 B Session

SI Session

Thursday, April 17th 2025

7:00 pm - 8:00 pm MST

Agenda

- Rapid Concurrency Structures
- Distributed Systems
  - When to Distribute
  - Parallel vs. Distributed
  - Distributed Structures
- Consensus

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

Concurrency Structures

Can we name some concurrency structures?

Atomic Operations & Variables

Locks

Semaphores

Monitors

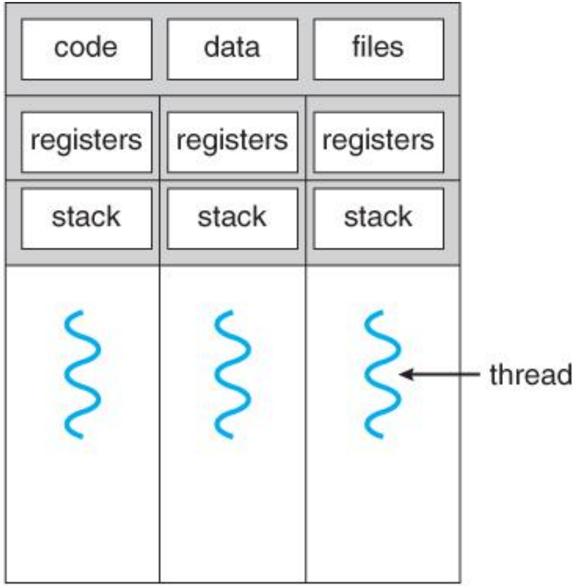
SER 321

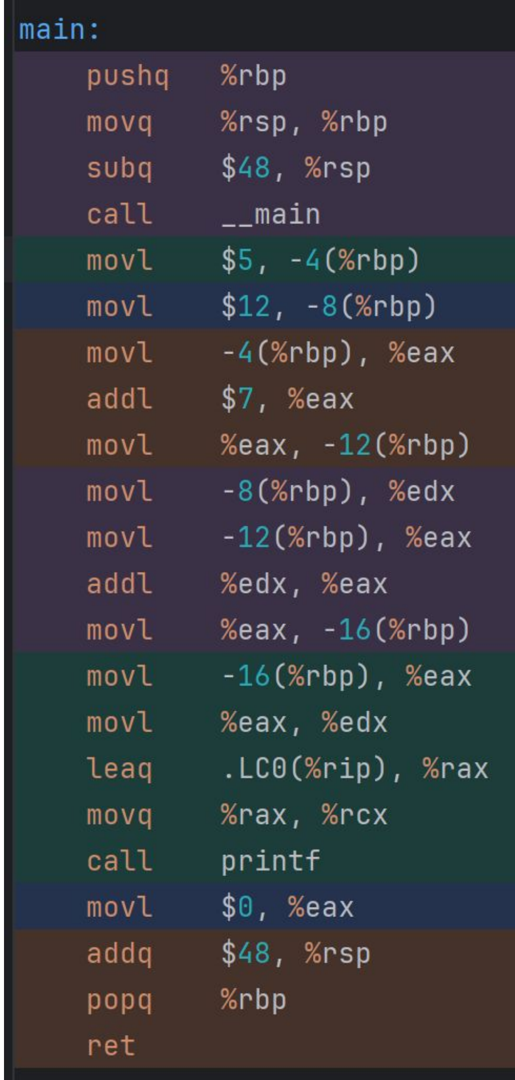
Concurrency Structures

Atomic Operations & Variables

Recall *registers*…

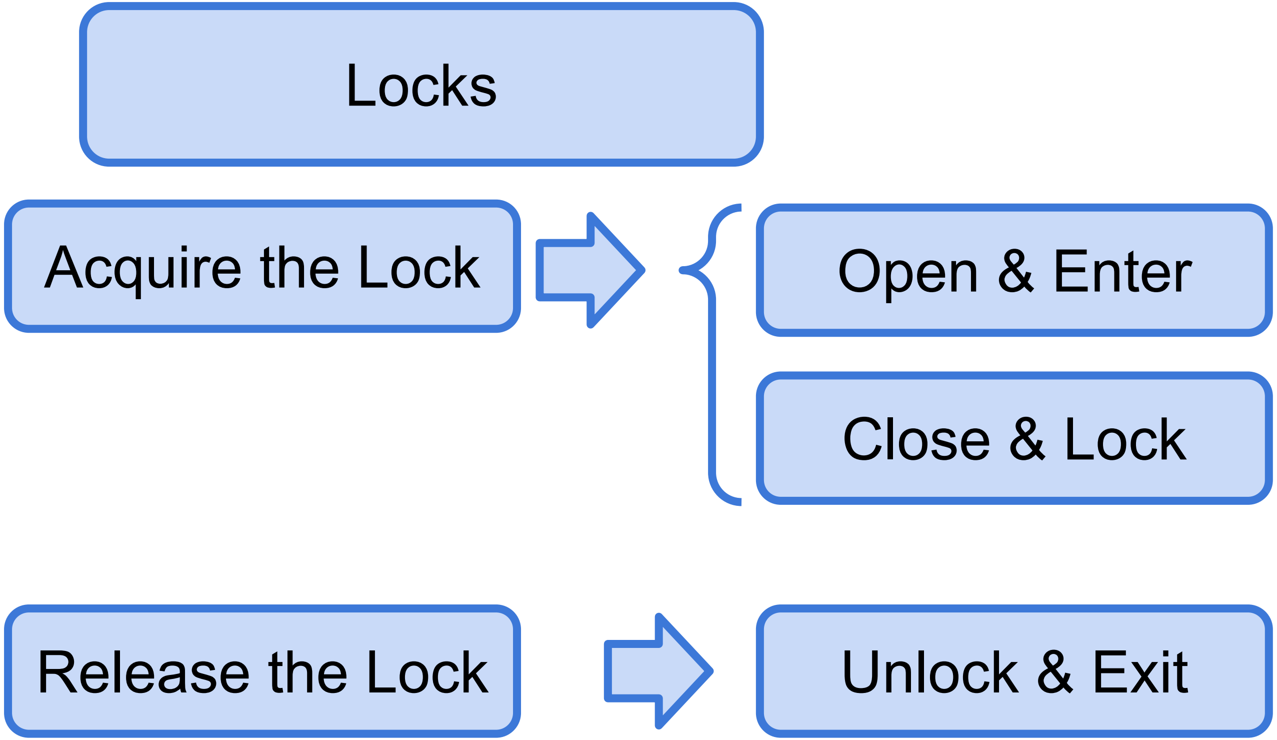
Ensures updates are immediately visible for the local copy in *each thread*





```
main:
    pushq   %rbp
    movq    %rsp, %rbp
    subq    $48, %rsp
    call    __main
    movl    $5, -4(%rbp)
    movl    $12, -8(%rbp)
    movl    -4(%rbp), %eax
    addl    $7, %eax
    movl    %eax, -12(%rbp)
    movl    -8(%rbp), %edx
    movl    -12(%rbp), %eax
    addl    %edx, %eax
    movl    %eax, -16(%rbp)
    movl    -16(%rbp), %eax
    movl    %eax, %edx
    leaq    .LC0(%rip), %rax
    movq    %rax, %rcx
    call    printf
    movl    $0, %eax
    addq    $48, %rsp
    popq    %rbp
    ret
```

Pros and Cons?



Locks

Acquire the Lock → Open & Enter, Close & Lock

Release the Lock → Unlock & Exit

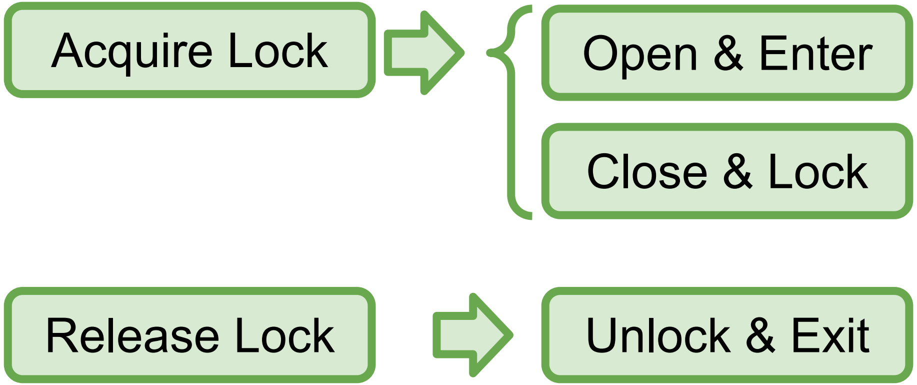
SER 321

Concurrency Structures

How am I different from a lock?

Semaphores

More than one stall!



Acquire Lock

Open & Enter

Close & Lock

Release Lock

Unlock & Exit

Semaphores support ***more than one*** acquirer

When would that be beneficial?

Pros and Cons?

Monitors

You lock the main door instead!

Acquire Lock → Open & Enter, Close & Lock

Release Lock → Unlock & Exit

Covers the ***entire object***

SER 321

Concurrency Structures

RECAP

Atomic Operations & Variables

YOU control the locks directly

Locks

YOU control the locks directly

Semaphores

YOU control the locks directly

Monitors

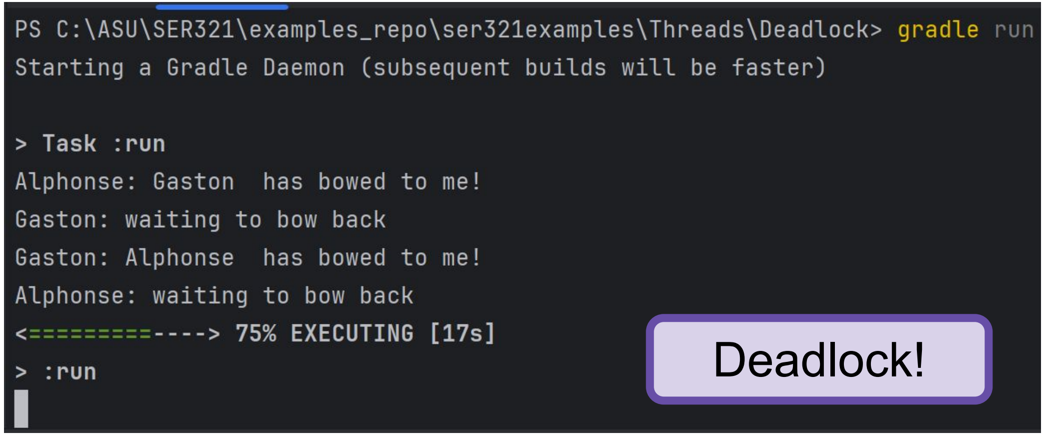
Locks managed for you

SER 321

Concurrency Structures

Monitors

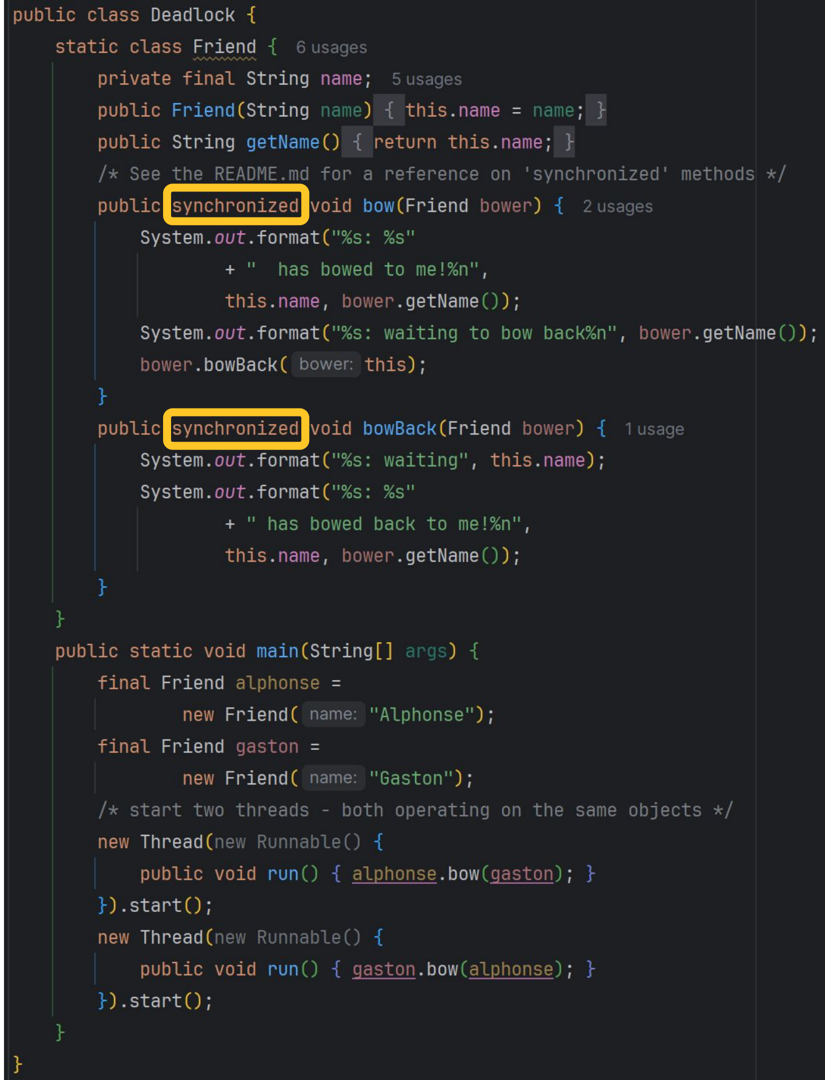
Both *bow()* and *bowBack()* are synchronized → are we good?



```
PS C:\ASU\SER321\examples_repo\ser321examples\Threads\Deadlock> gradle run
Starting a Gradle Daemon (subsequent builds will be faster)

> Task :run
Alphonse: Gaston  has bowed to me!
Gaston: waiting to bow back
Gaston: Alphonse  has bowed to me!
Alphonse: waiting to bow back
<=========----> 75% EXECUTING [17s]
> :run
```

Deadlock!



```
public class Deadlock {
    static class Friend {  6 usages
        private final String name;  5 usages
        public Friend(String name) { this.name = name; }
        public String getName() { return this.name; }
        /* See the README.md for a reference on 'synchronized' methods */
        public synchronized void bow(Friend bower) {  2 usages
            System.out.format("%s: %s"
                    + "  has bowed to me!%n",
                    this.name, bower.getName());
            System.out.format("%s: waiting to bow back%n", bower.getName());
            bower.bowBack( bower: this);
        }
        public synchronized void bowBack(Friend bower) {  1 usage
            System.out.format("%s: waiting", this.name);
            System.out.format("%s: %s"
                    + " has bowed back to me!%n",
                    this.name, bower.getName());
        }
    }
    public static void main(String[] args) {
        final Friend alphonse =
                new Friend( name: "Alphonse");
        final Friend gaston =
                new Friend( name: "Gaston");
        /* start two threads - both operating on the same objects */
        new Thread(new Runnable() {
            public void run() { alphonse.bow(gaston); }
        }).start();
        new Thread(new Runnable() {
            public void run() { gaston.bow(alphonse); }
        }).start();
    }
}
```

SER 321

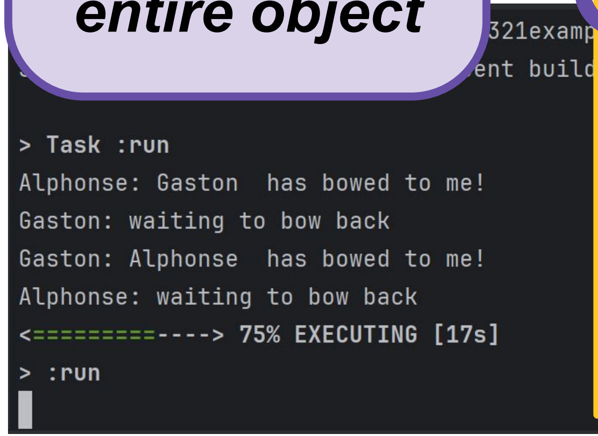
Concurrency Structures

Monitors manage locks for us by ***locking the entire object***

d bo

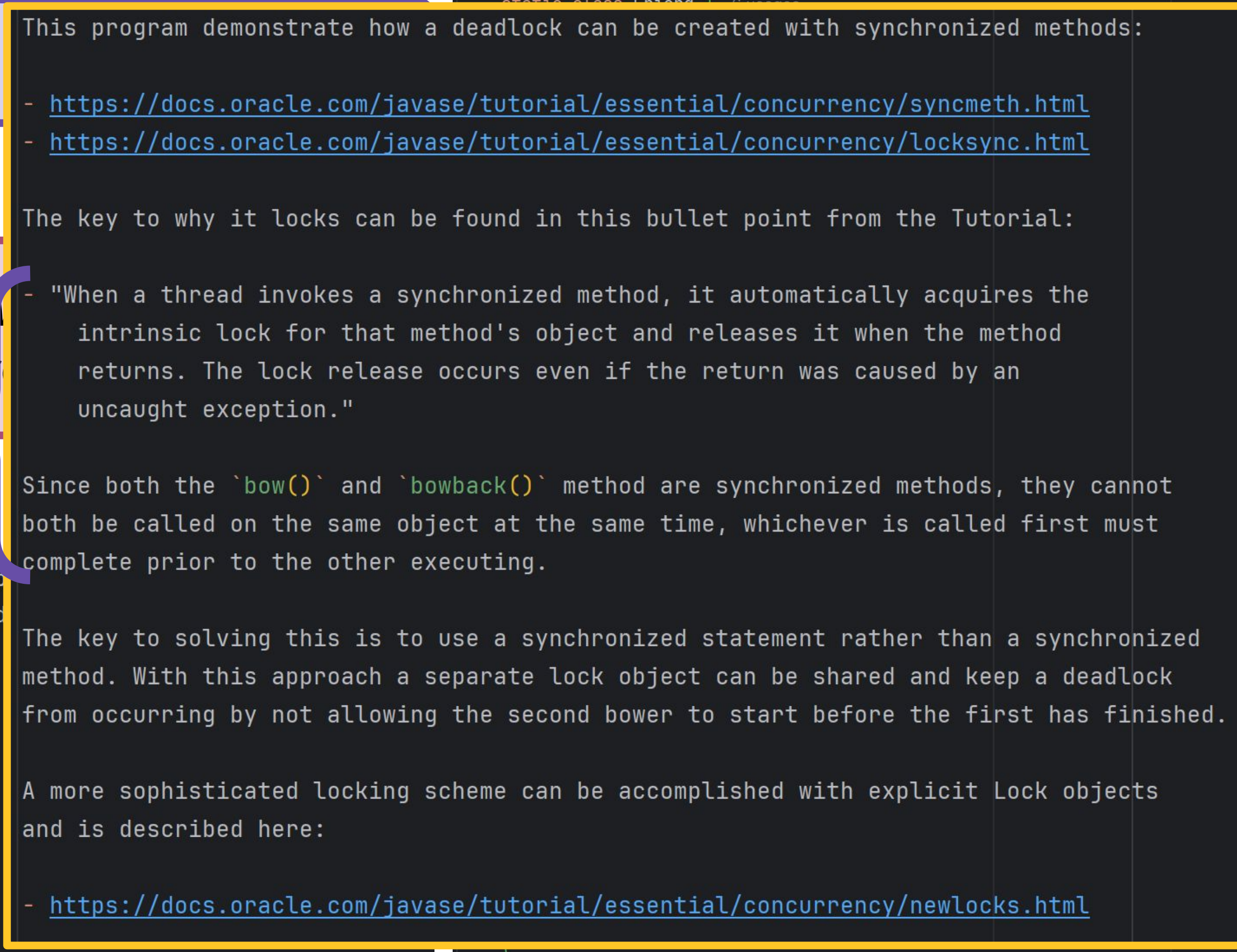
→ a

```
public class Deadlock {
    static class Friend {
```



```
321examp
ent build

> Task :run
Alphonse: Gaston  has bowed to me!
Gaston: waiting to bow back
Gaston: Alphonse  has bowed to me!
Alphonse: waiting to bow back
<=========----> 75% EXECUTING [17s]
> :run
```



```
This program demonstrate how a deadlock can be created with synchronized methods:

- https://docs.oracle.com/javase/tutorial/essential/concurrency/syncmeth.html
- https://docs.oracle.com/javase/tutorial/essential/concurrency/locksync.html

The key to why it locks can be found in this bullet point from the Tutorial:

- "When a thread invokes a synchronized method, it automatically acquires the
   intrinsic lock for that method's object and releases it when the method
   returns. The lock release occurs even if the return was caused by an
   uncaught exception."

Since both the `bow()` and `bowback()` method are synchronized methods, they cannot
both be called on the same object at the same time, whichever is called first must
complete prior to the other executing.

The key to solving this is to use a synchronized statement rather than a synchronized
method. With this approach a separate lock object can be shared and keep a deadlock
from occurring by not allowing the second bower to start before the first has finished.

A more sophisticated locking scheme can be accomplished with explicit Lock objects
and is described here:

- https://docs.oracle.com/javase/tutorial/essential/concurrency/newlocks.html
```

```
}
```

SER 321

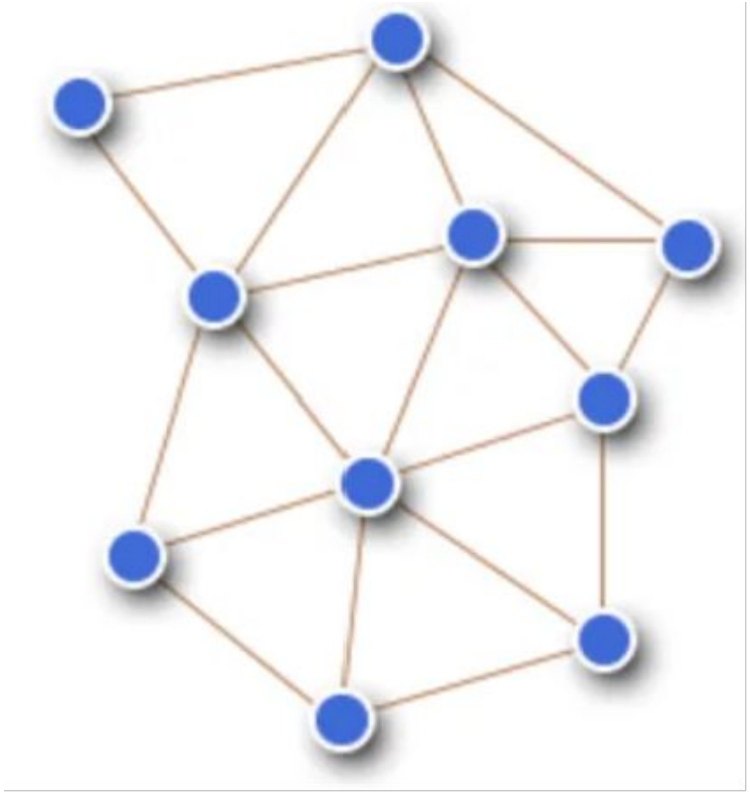
Distributed Systems

What do we mean by

“Distributed Systems”

or

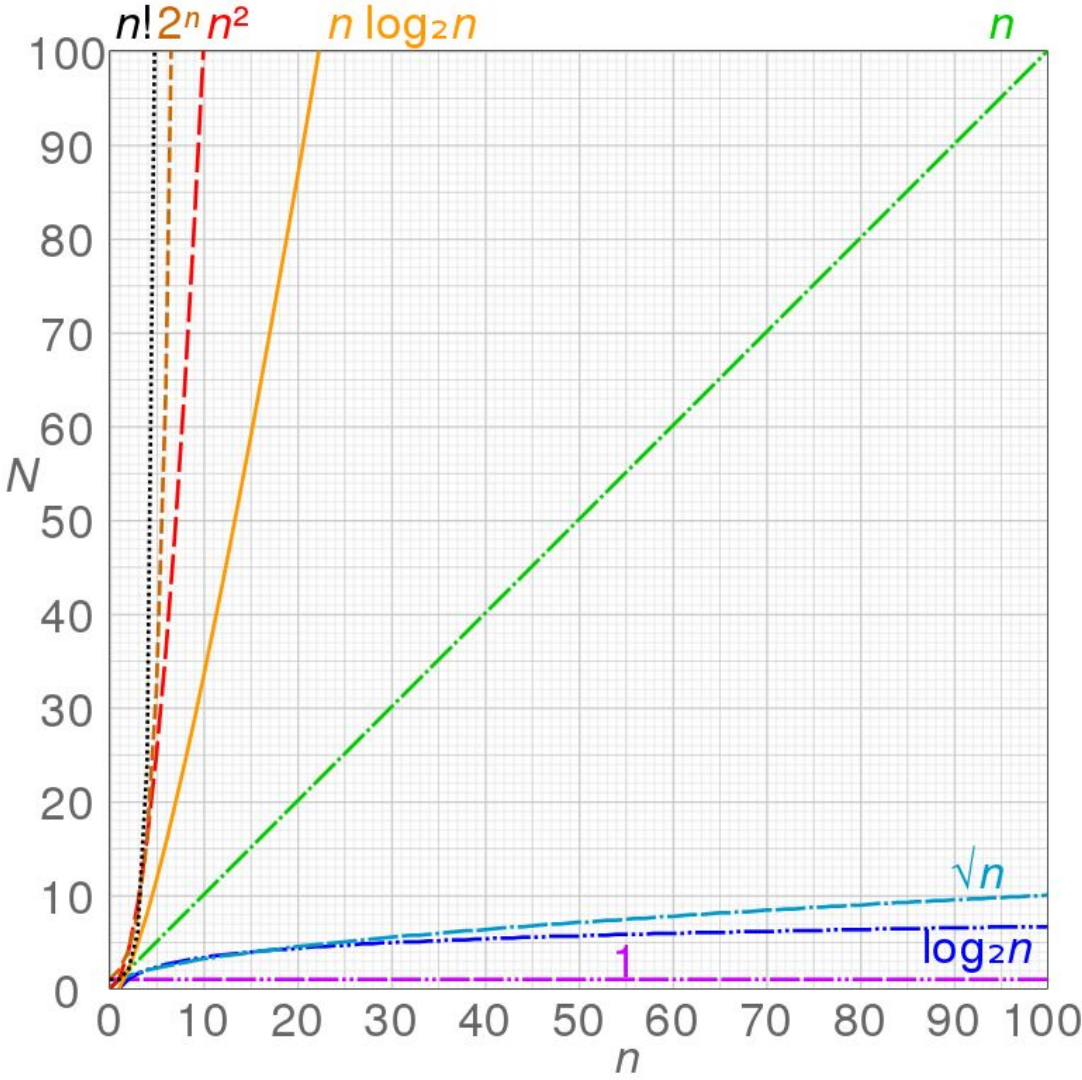
“Distributed Algorithms”?



SER 321

Distributed Systems

When should we *consider* distributing?

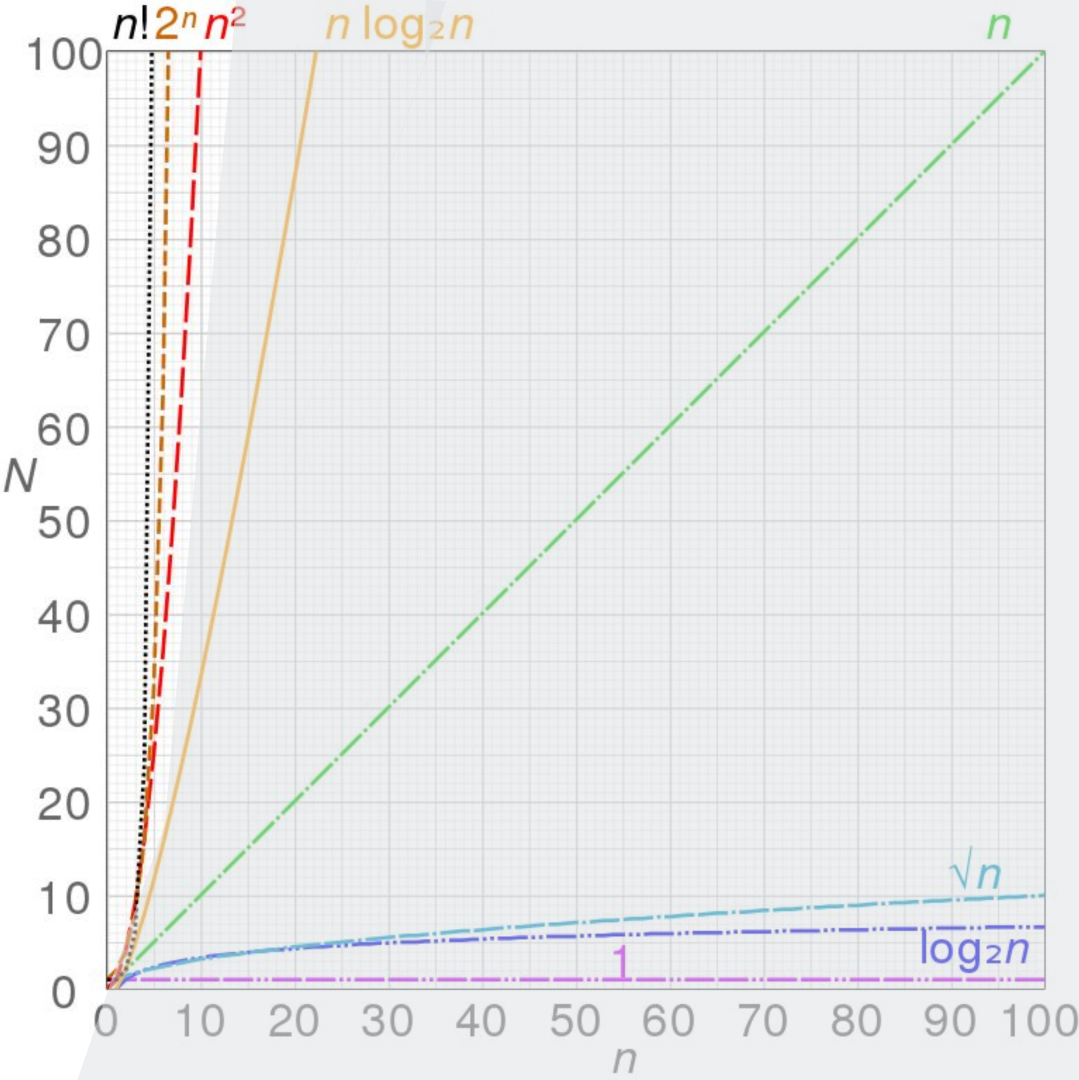


SER 321

Distributed Systems

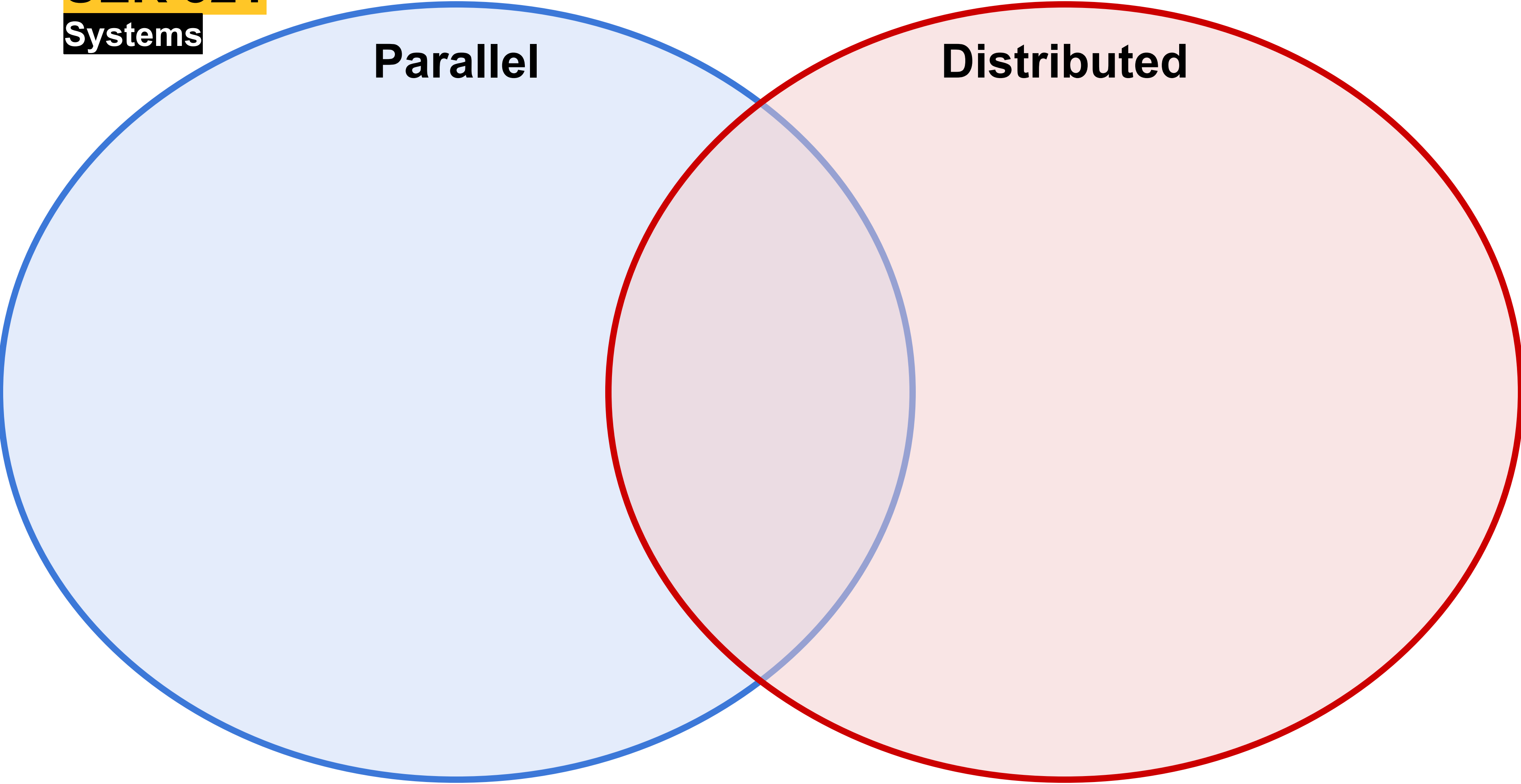
When should we *consider* distributing?

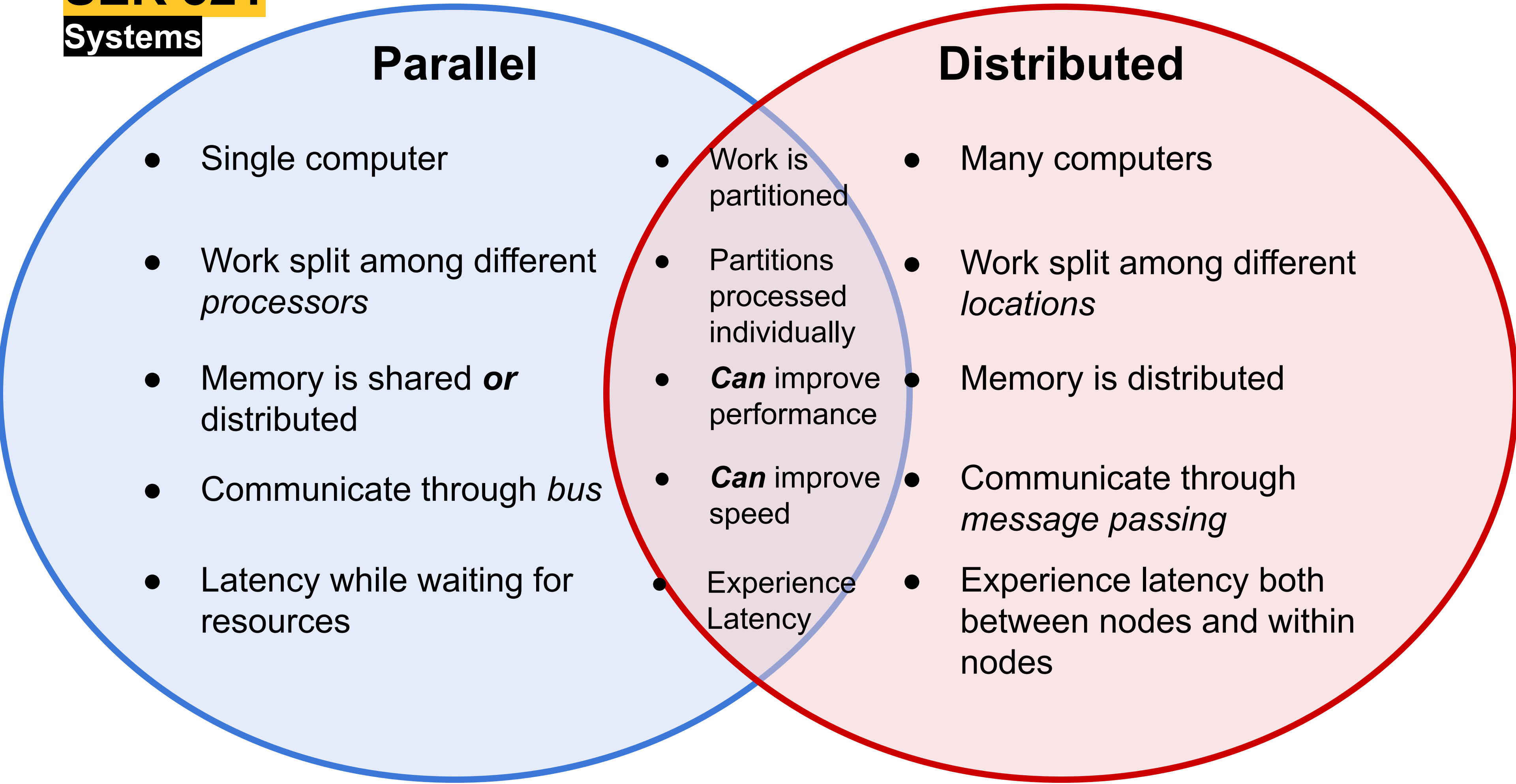
Super Duper Extra Extra Large Orders of Magnitude!



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

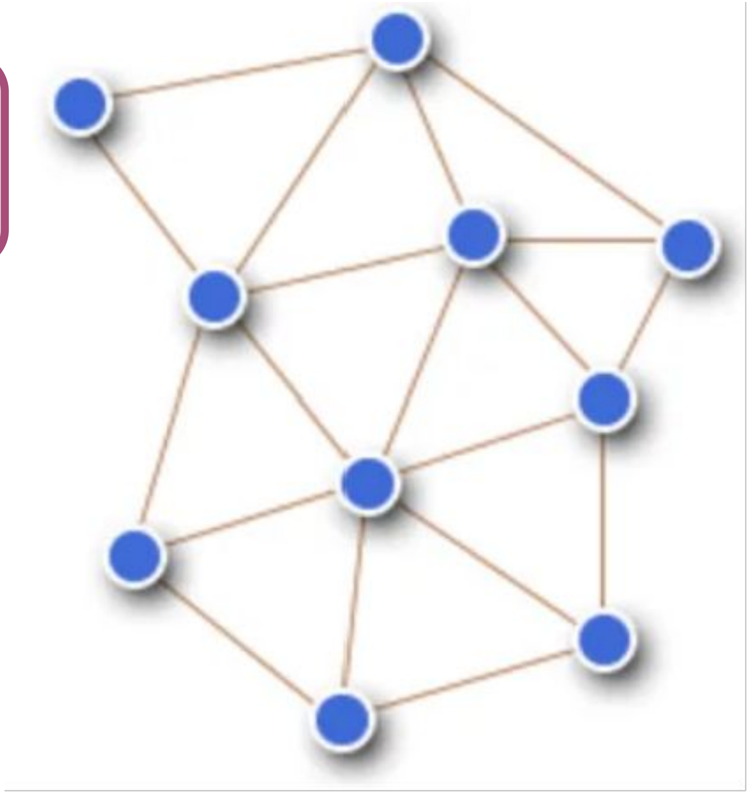
- Many computers
- Work split among different *locations*
- Memory is distributed
- Communicate through *message passing*
- Experience latency both between nodes and within nodes

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

SER 321
Distributed Systems

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

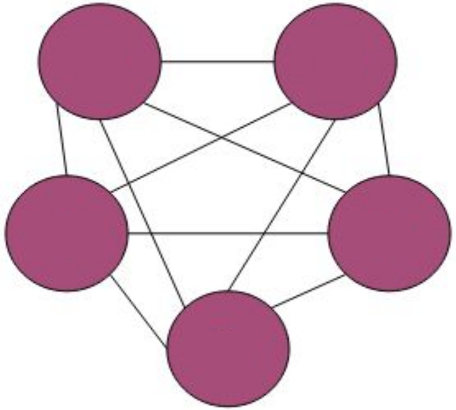
- *No* global clock
- Nodes *will* fail
- Web of nodes *will constantly* change
- Network is not *always* reliable
- Latency is *always present*
- The path traversed *changes*
- Some resources *must be shared*
- *You* need to prevent the pitfalls!
  - No deadlocks
  - No starvation
  - No error states

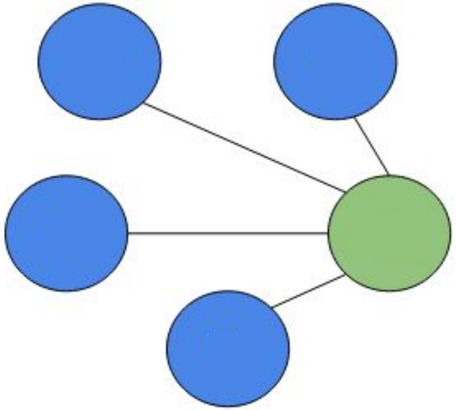


Main and Worker

Peer to Peer

Which is which?



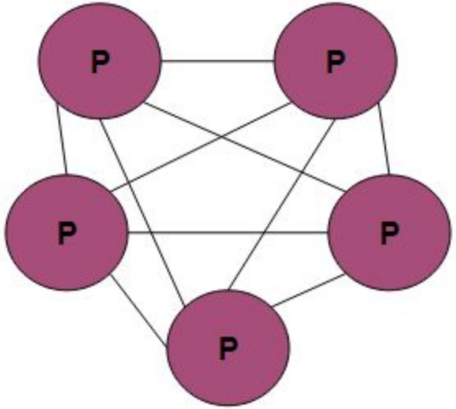


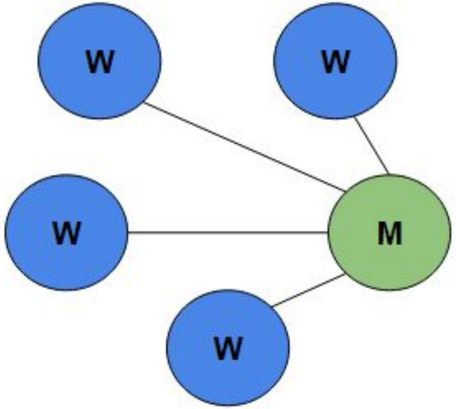
Main and Worker

Peer to Peer

Which is which?

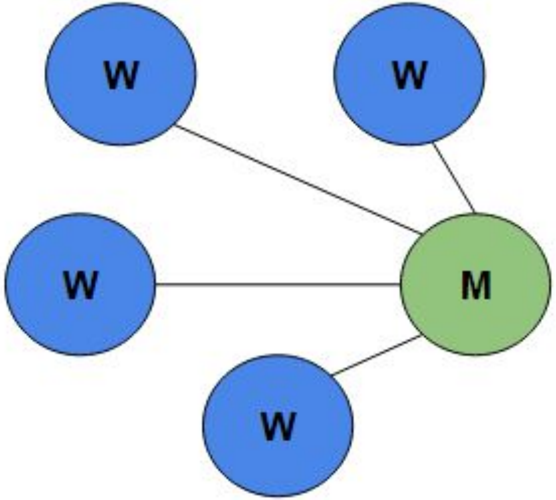
Peer to Peer





Main and Worker

Pros and Cons



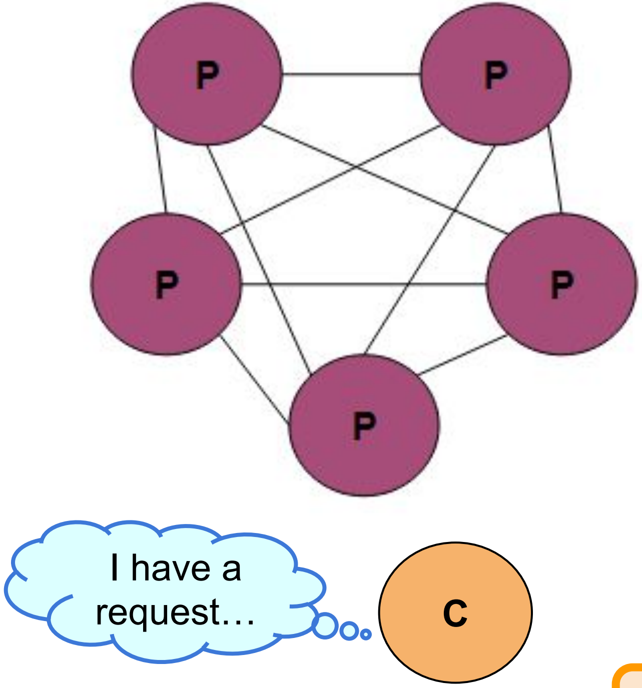
Pros:

- Straightforward setup
- Logic is centralized
- Communication is linear

Cons:

- Single point of failure

Pros and Cons



Pros:

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

Cons:

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

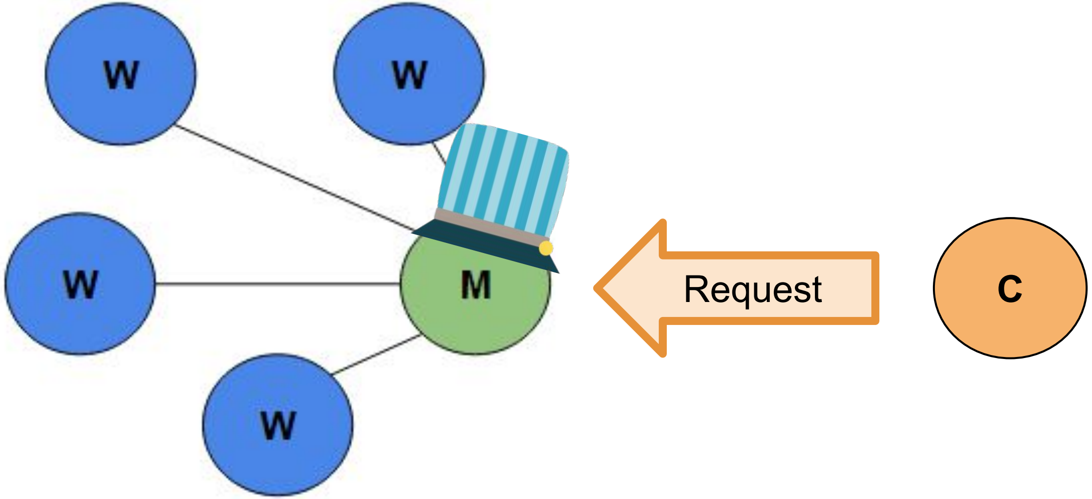
We will cover this in a little bit!

SER 321
Distributed Systems

Process Flow!

DATA

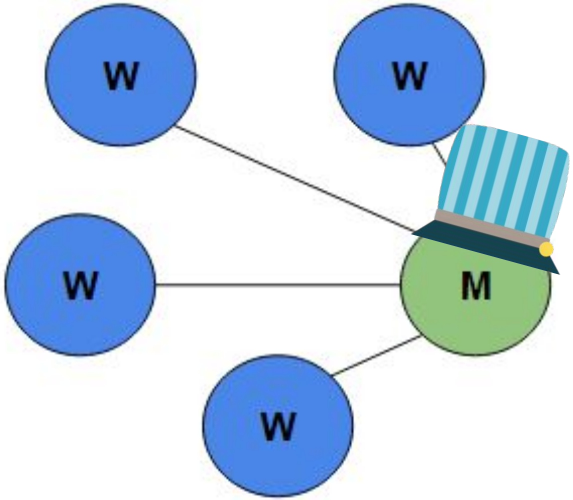
Workers only do their task then report back

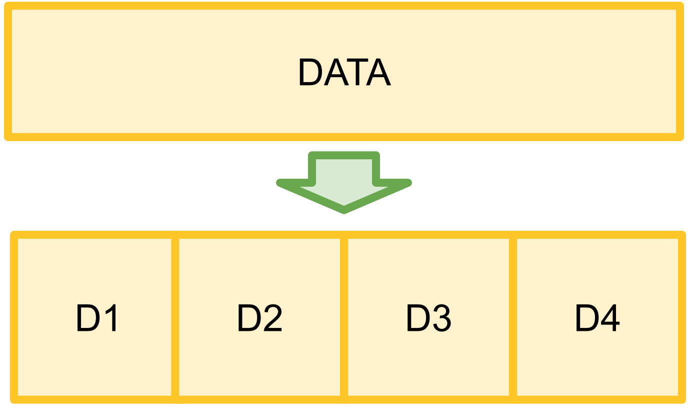


Main is like our server

Process Flow!

Workers only do their task then report back



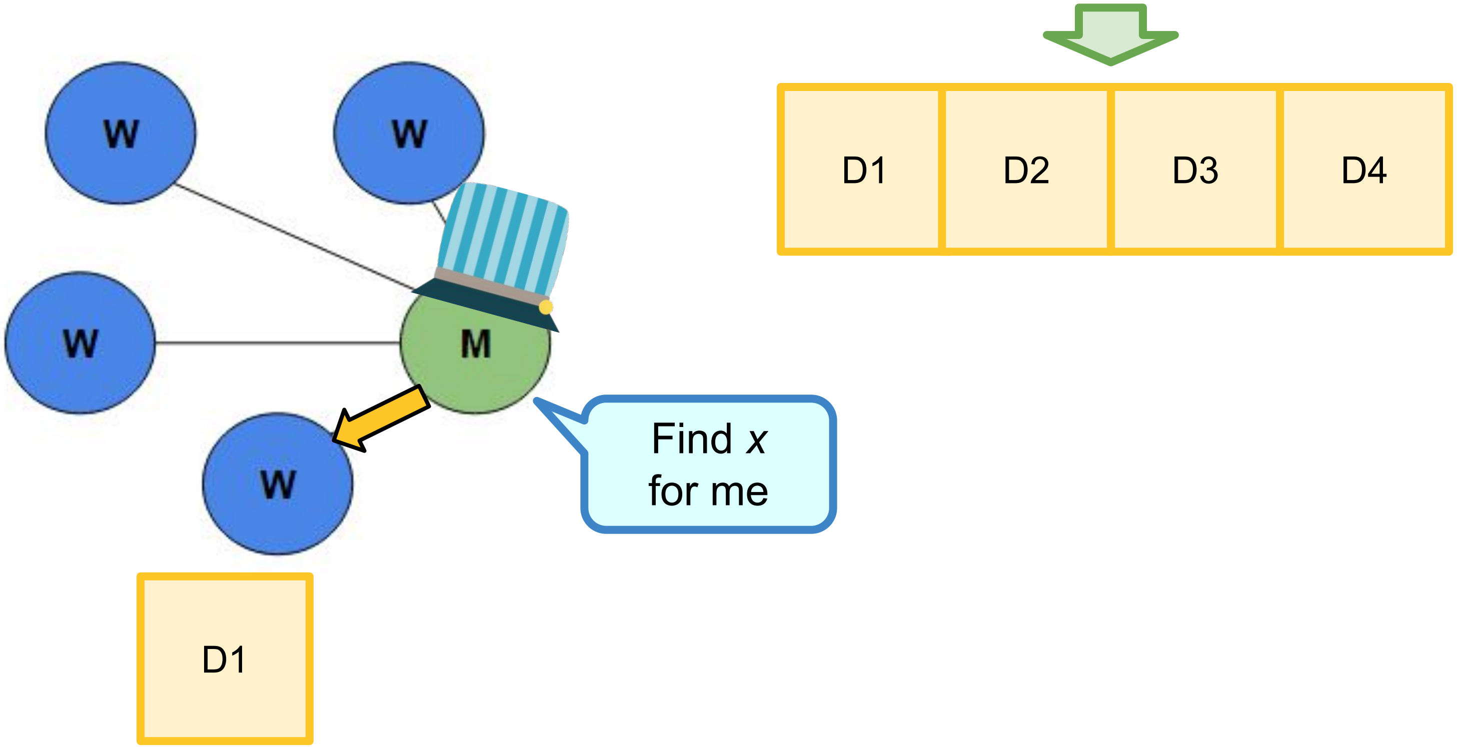

DATA

D1 D2 D3 D4

SER 321
Distributed Systems

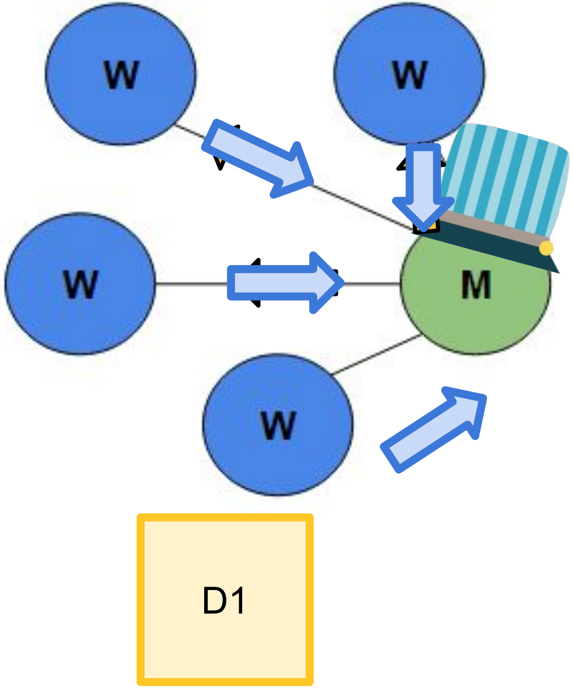
Process Flow!

Workers only do their task then report back



Process Flow!

Workers only do their task then report back

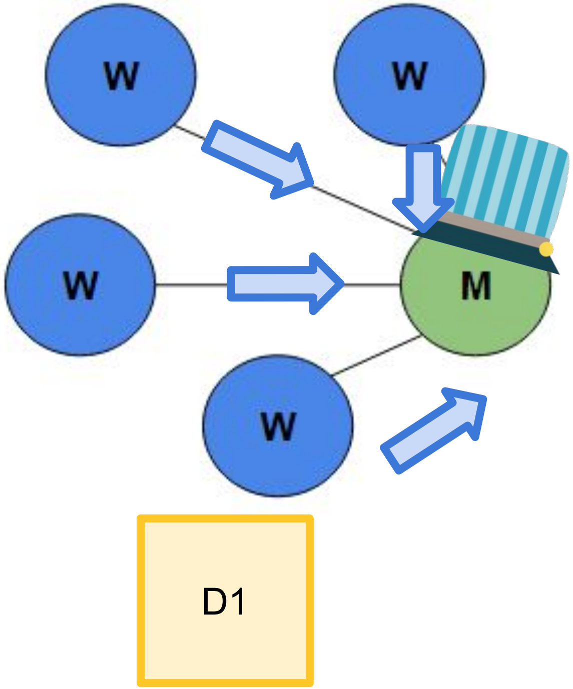


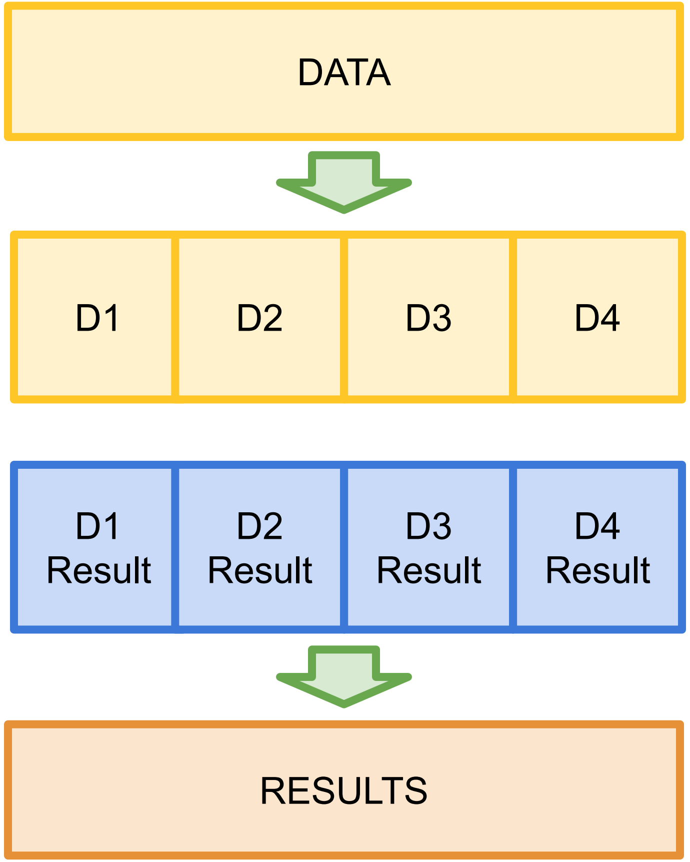
DATA

D1	D2	D3	D4
D1 Result	D2 Result	D3 Result	D4 Result

Process Flow!

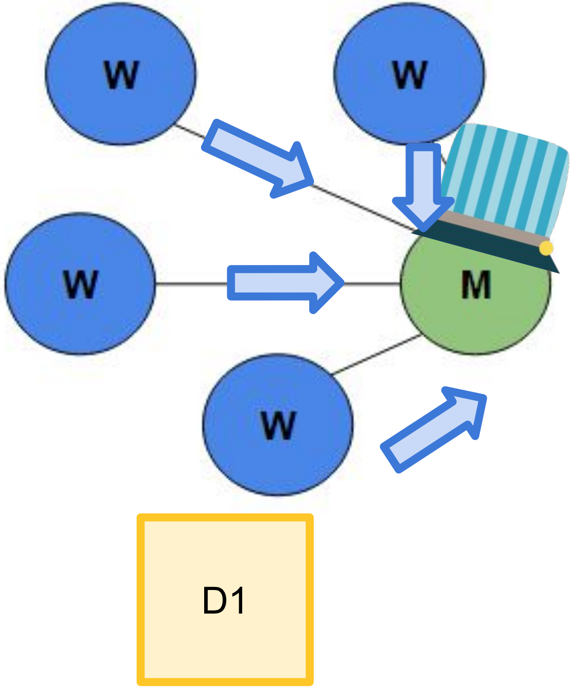
Workers only do their task then report back

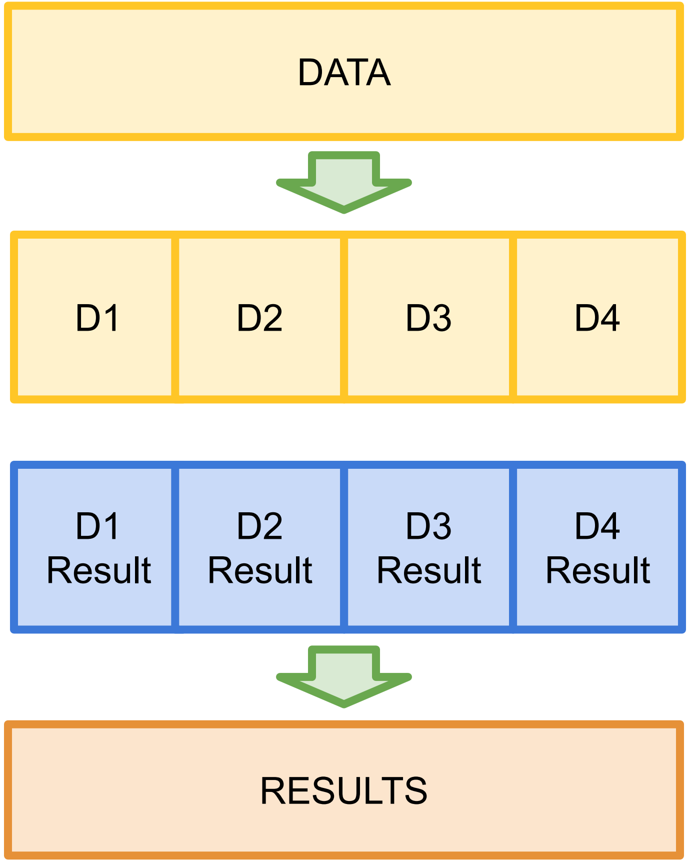




Does this look familiar?

How is this different from a parallel processing model?





DATA

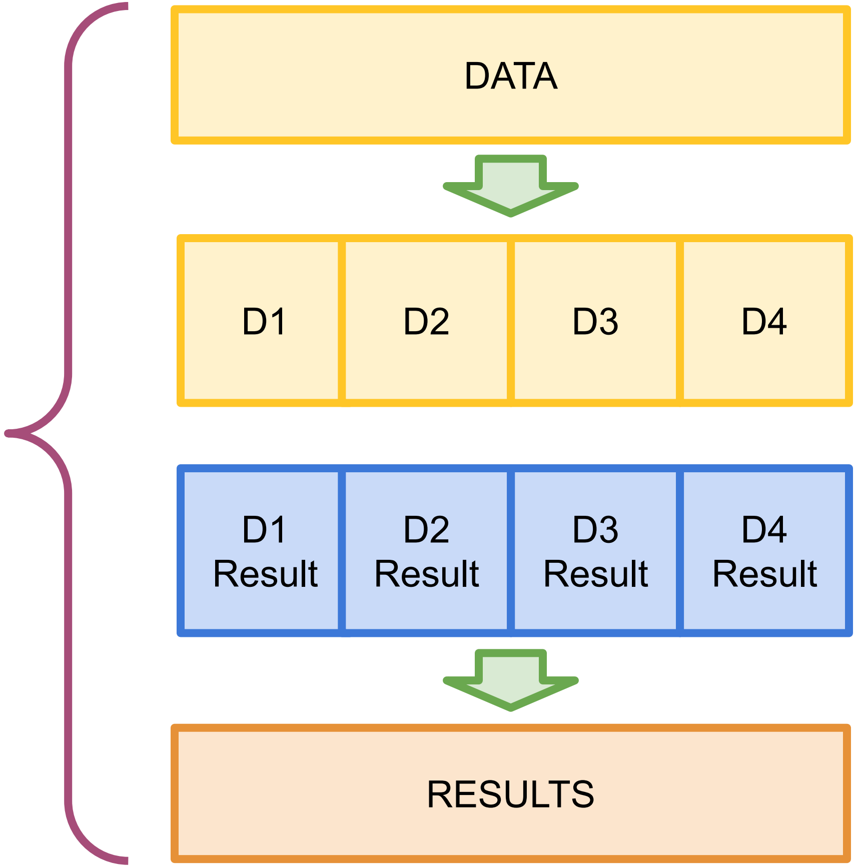
D1 D2 D3 D4

D1 Result D2 Result D3 Result D4 Result

RESULTS

What about Peer to Peer?

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



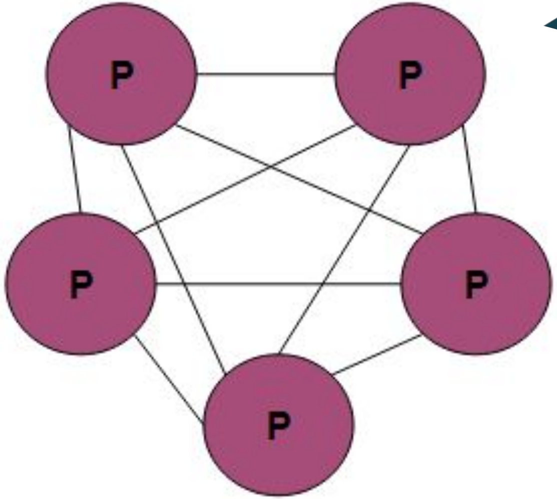
SER 321

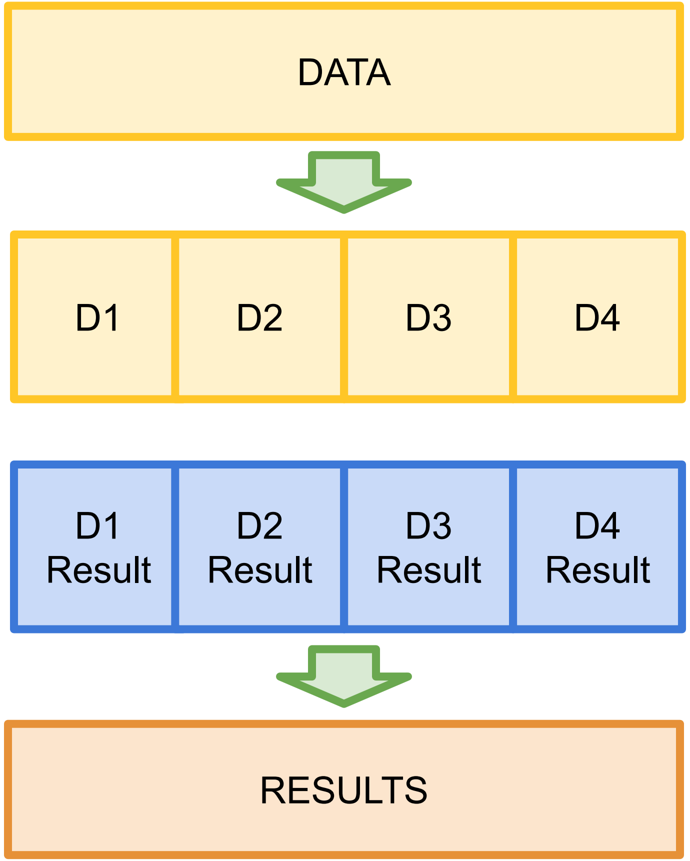
Distributed Systems

What about Peer to Peer?

We want someone to wear the conductor hat!

A ***LEADER***



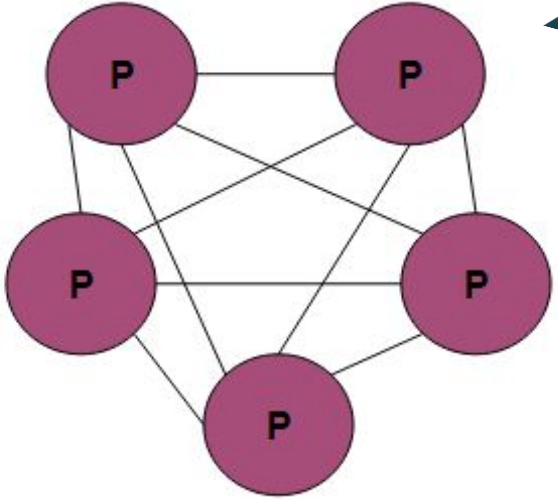


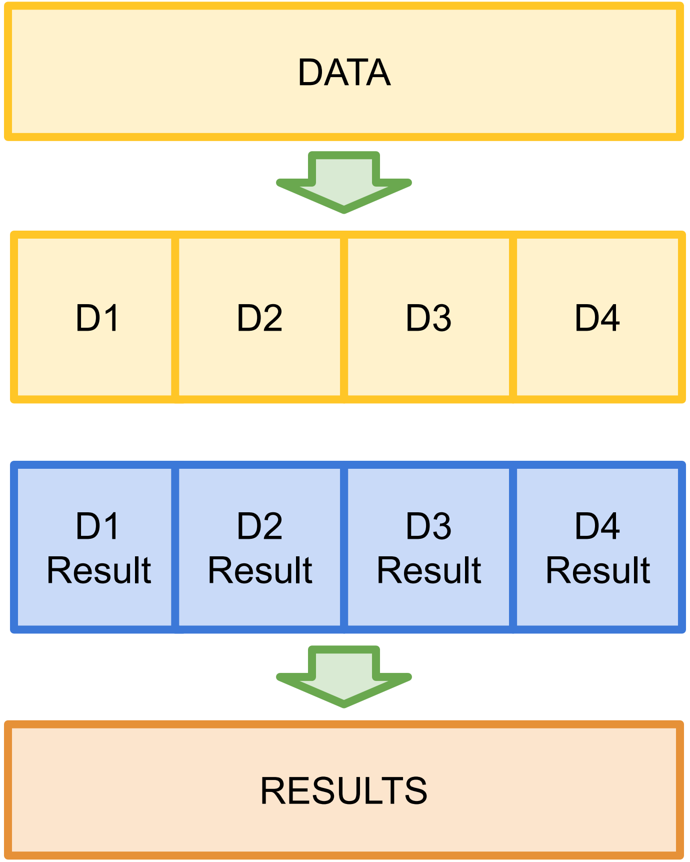
How do we choose a leader?

What about Peer to Peer?

We want someone to wear the conductor hat!

A ***LEADER***



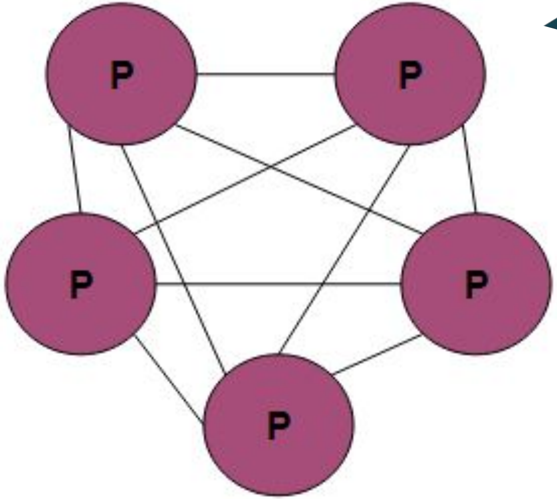


Leader Election!

What about Peer to Peer?

We want someone to wear the conductor hat!

A ***LEADER***



Leader Election!

Type of ***CONSENSUS***

What's consensus?

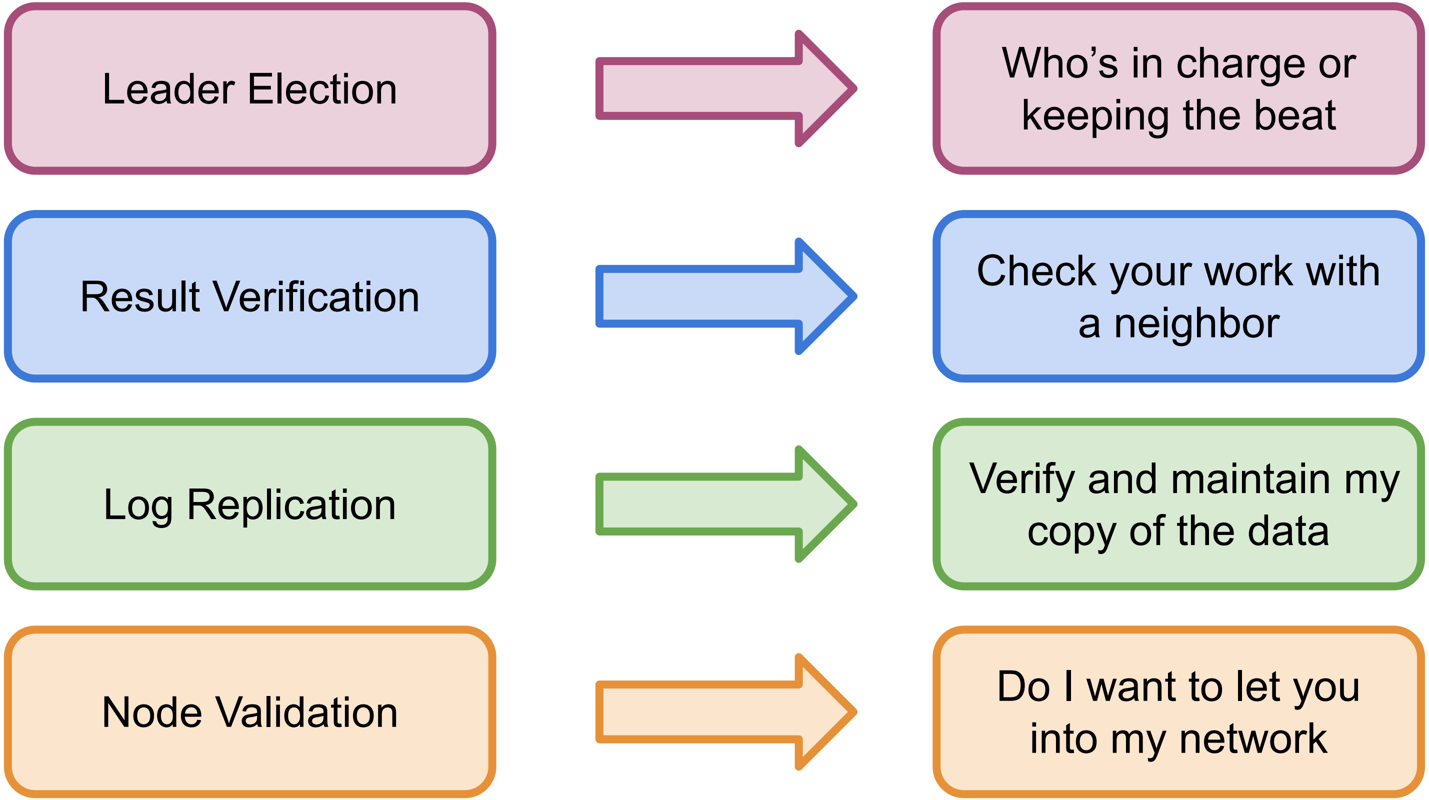
"General agreement or trust amongst a group"

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

Match the Consensus Algorithm to its Description!

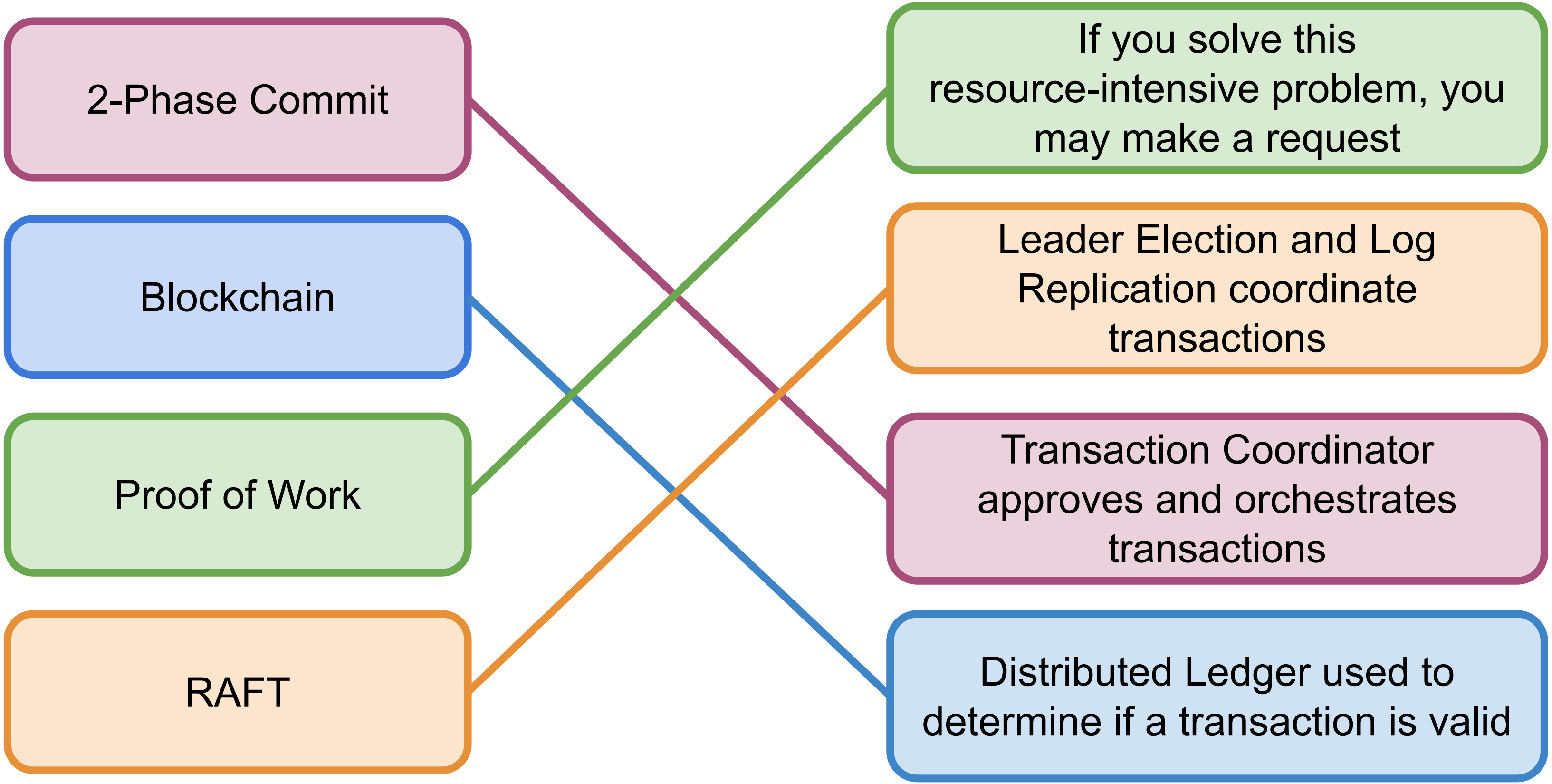
- 2-Phase Commit
- Blockchain
- Proof of Work
- RAFT

- 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!



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

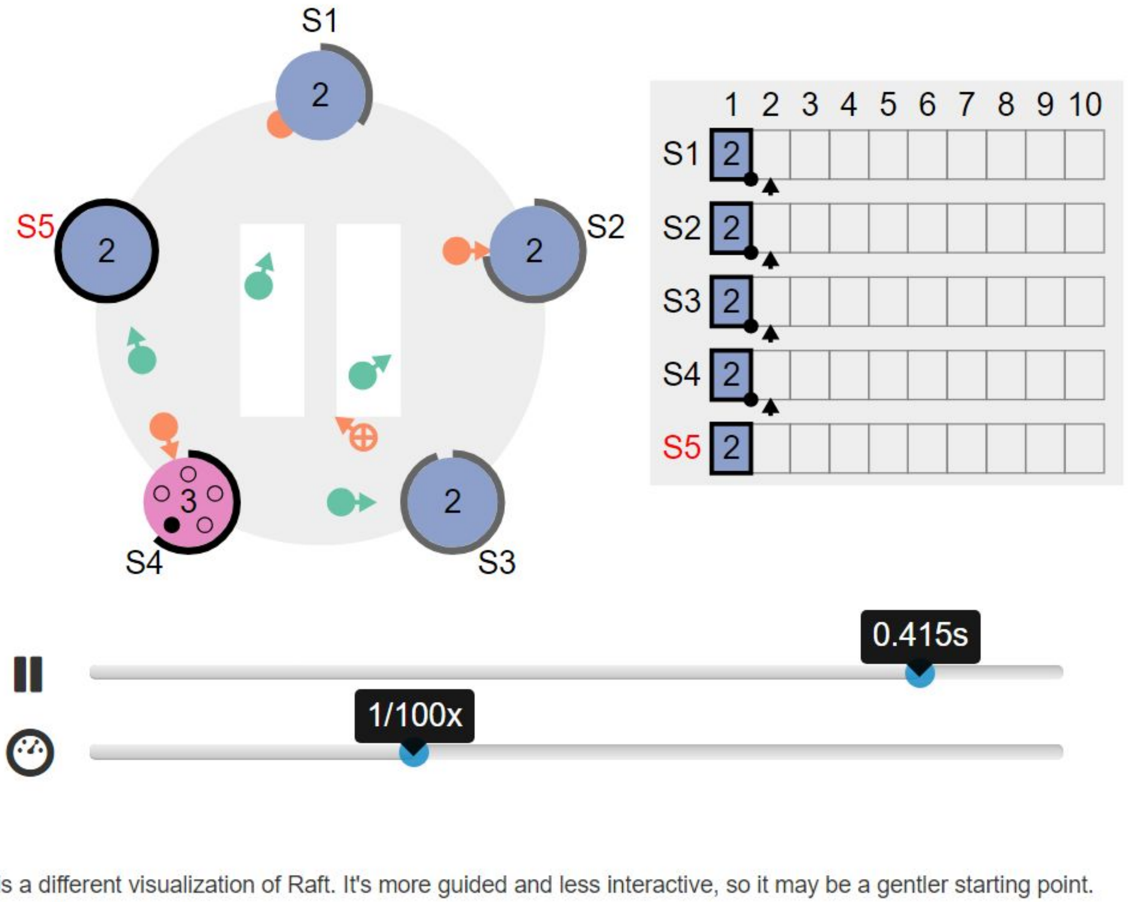
SER 321

RAFT

RAFT is a great consensus example!

Leader Election

Log Replication

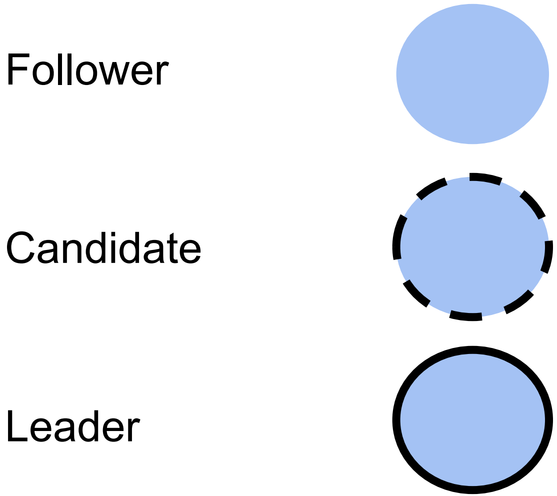


	1	2	3	4	5	6	7	8	9	10
S1	2									
S2	2									
S3	2									
S4	2									
S5	2									

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.

Leader Election

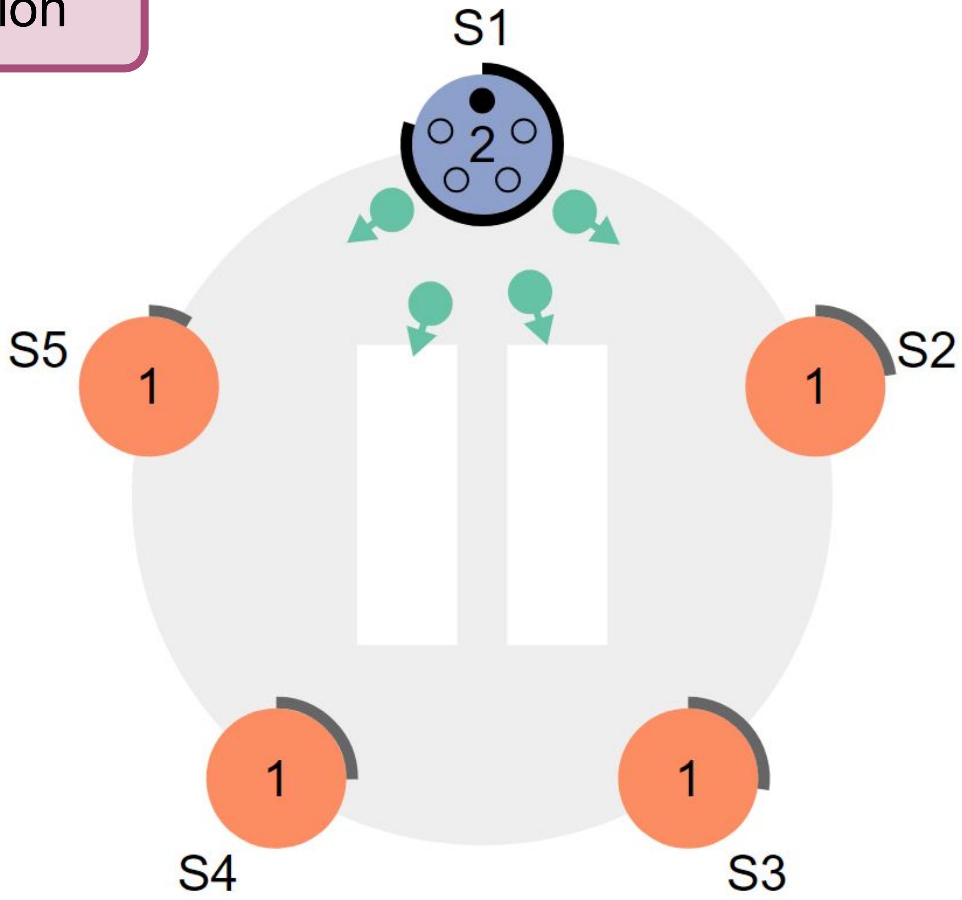
Nodes have 3 states:



Follower

Candidate

Leader

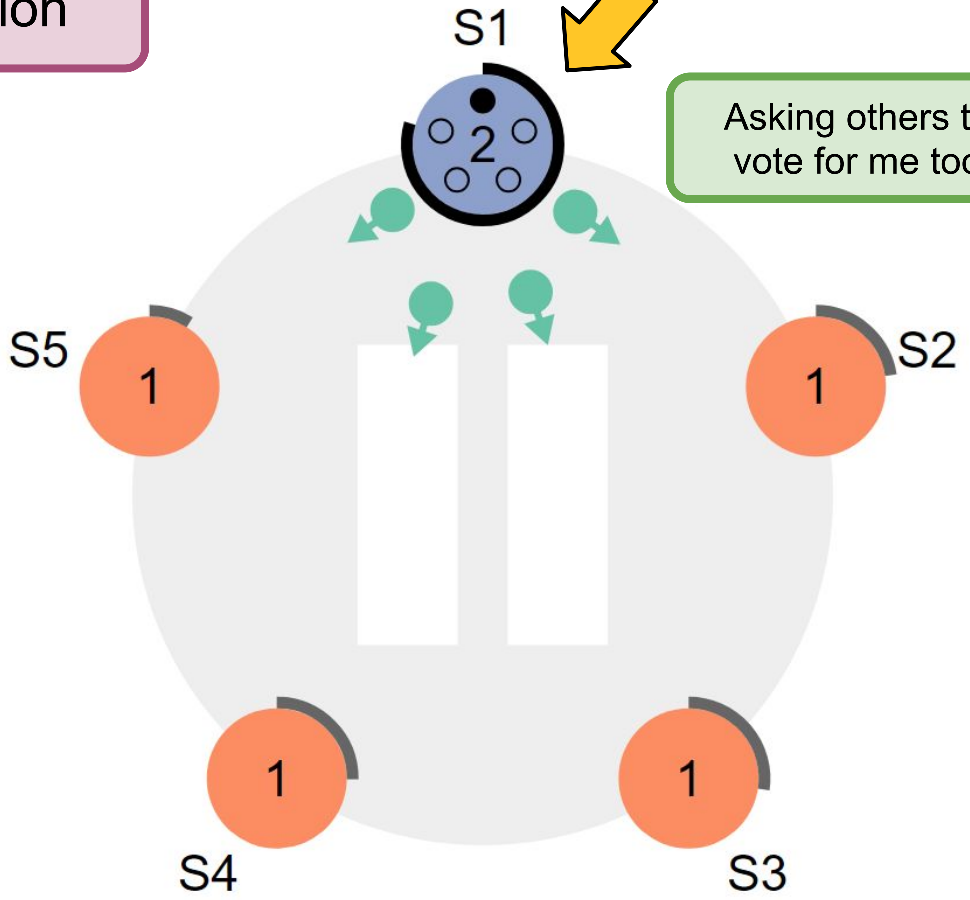


SER 321

RAFT

Leader Election

This is the first election



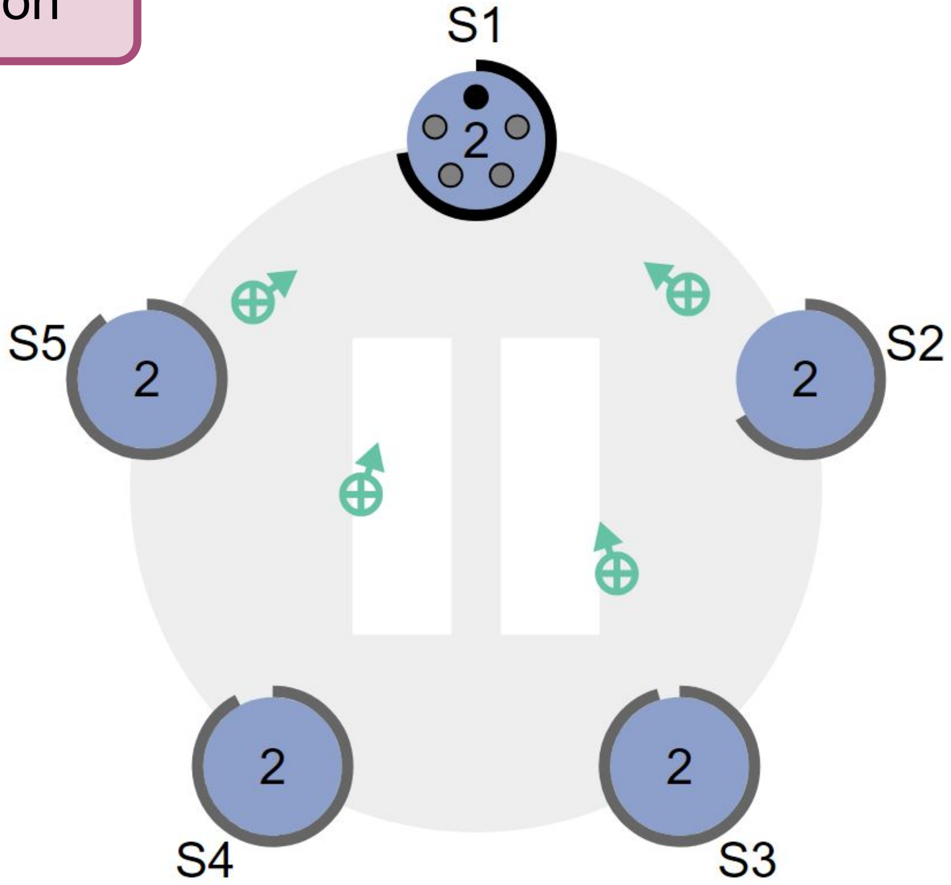
First to time out

Asking others to vote for me too

SER 321

RAFT

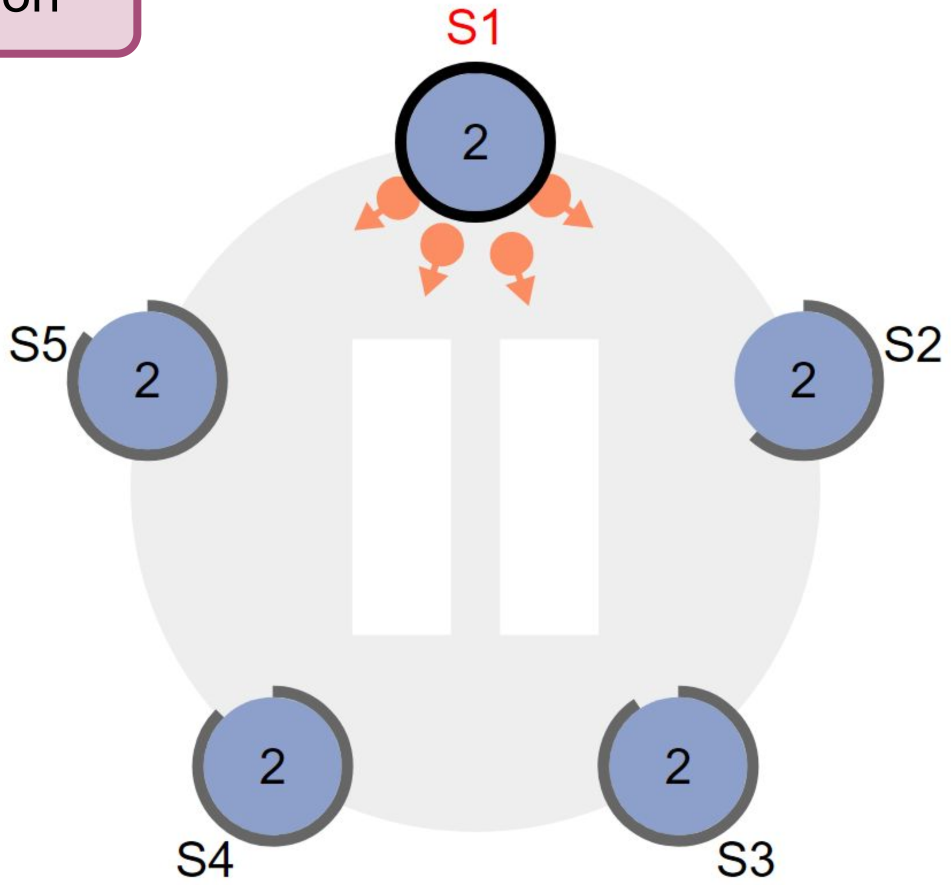
Leader Election



Other nodes said sure whatever

Leader Election

Now confirmed as Leader



SER 321

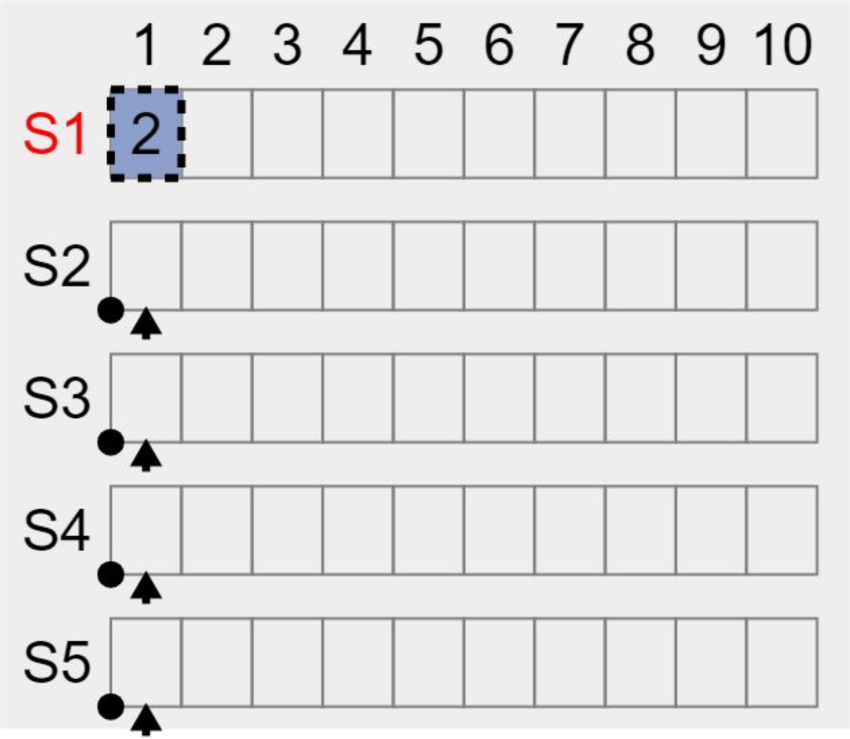
RAFT

Log Replication

Same Pattern!

Candidate ⬚

Added □



SER 321

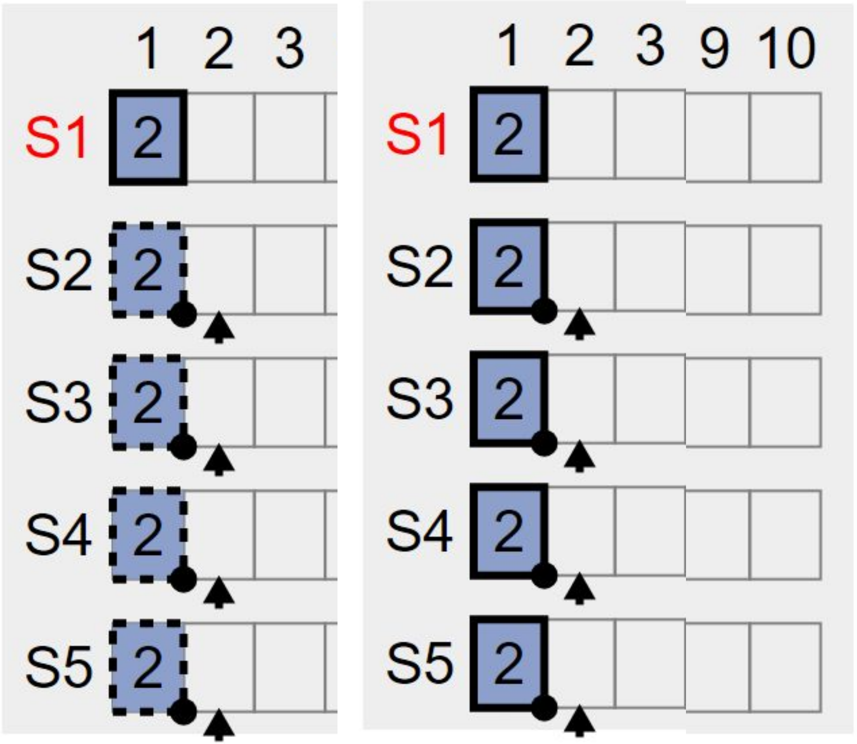
RAFT

Log Replication

Same Pattern!

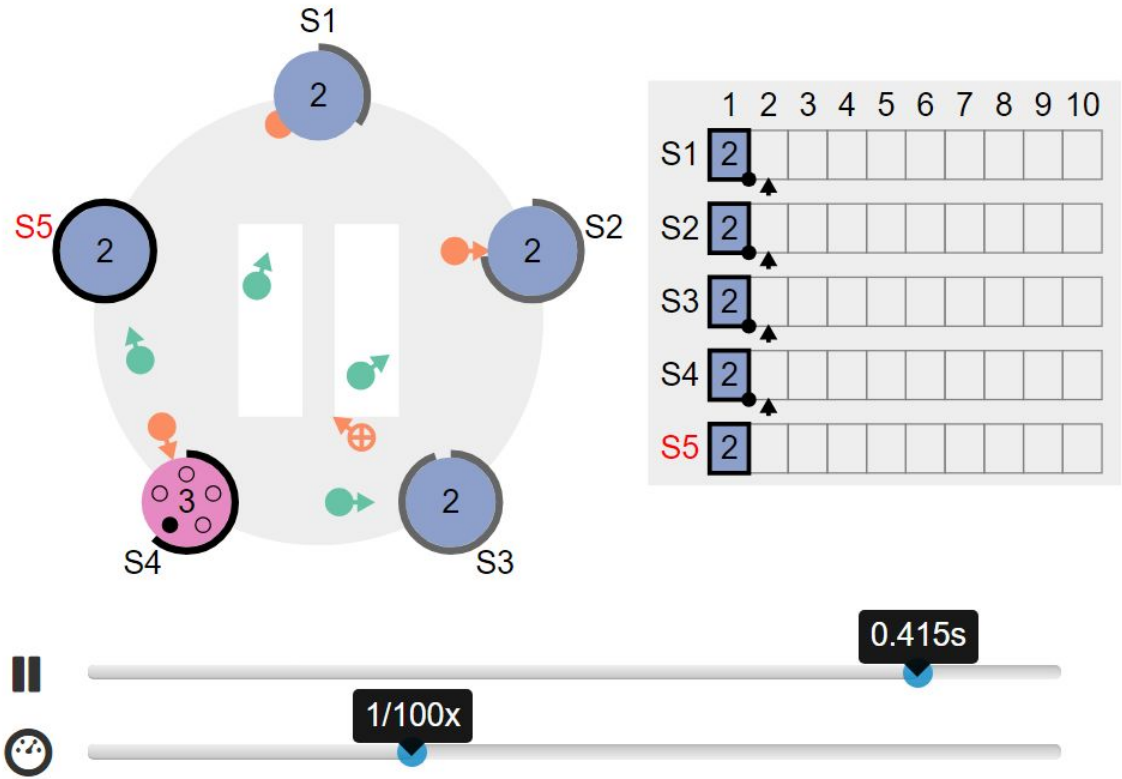
Candidate

Added



SER 321
RAFT

RAFT

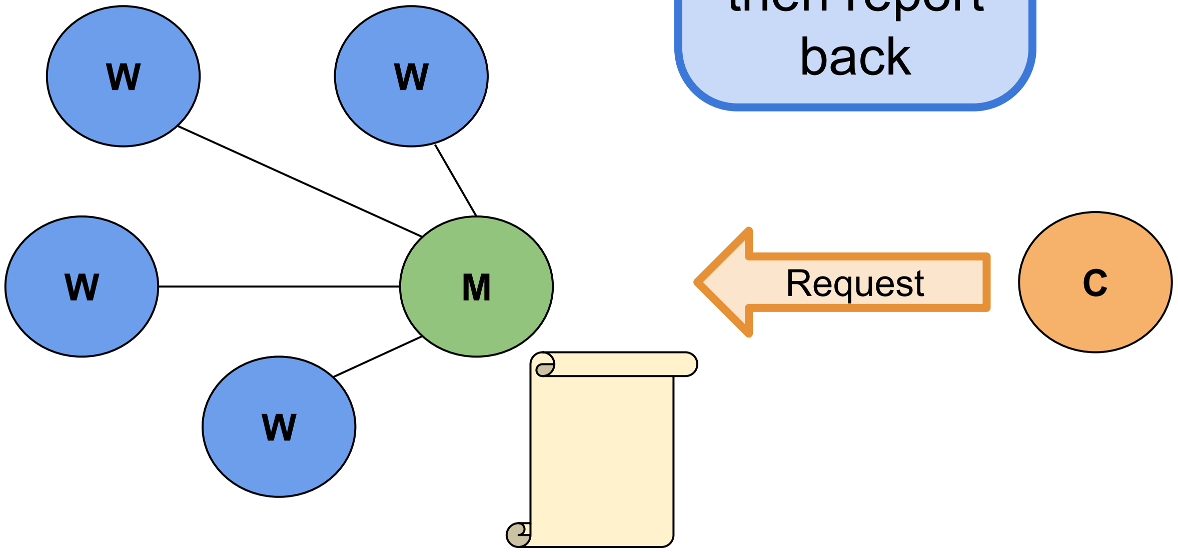


	1	2	3	4	5	6	7	8	9	10
S1	2									
S2	2									
S3	2									
S4	2									
S5	2									

0.415s

1/100x

Workers only do their task then report back

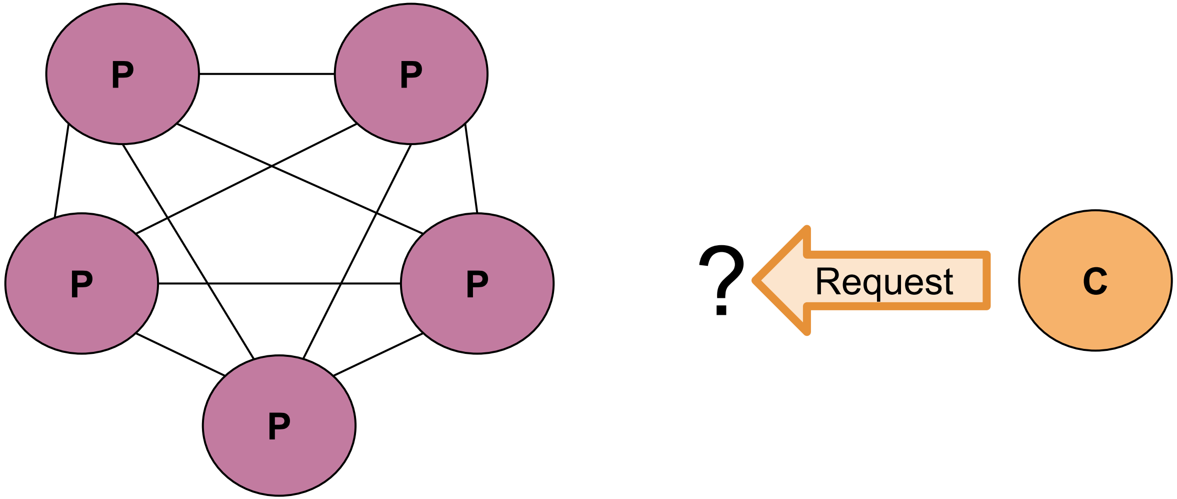


Main is in charge of everything

SER 321

Communication

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



Request is sent to the *current leader*

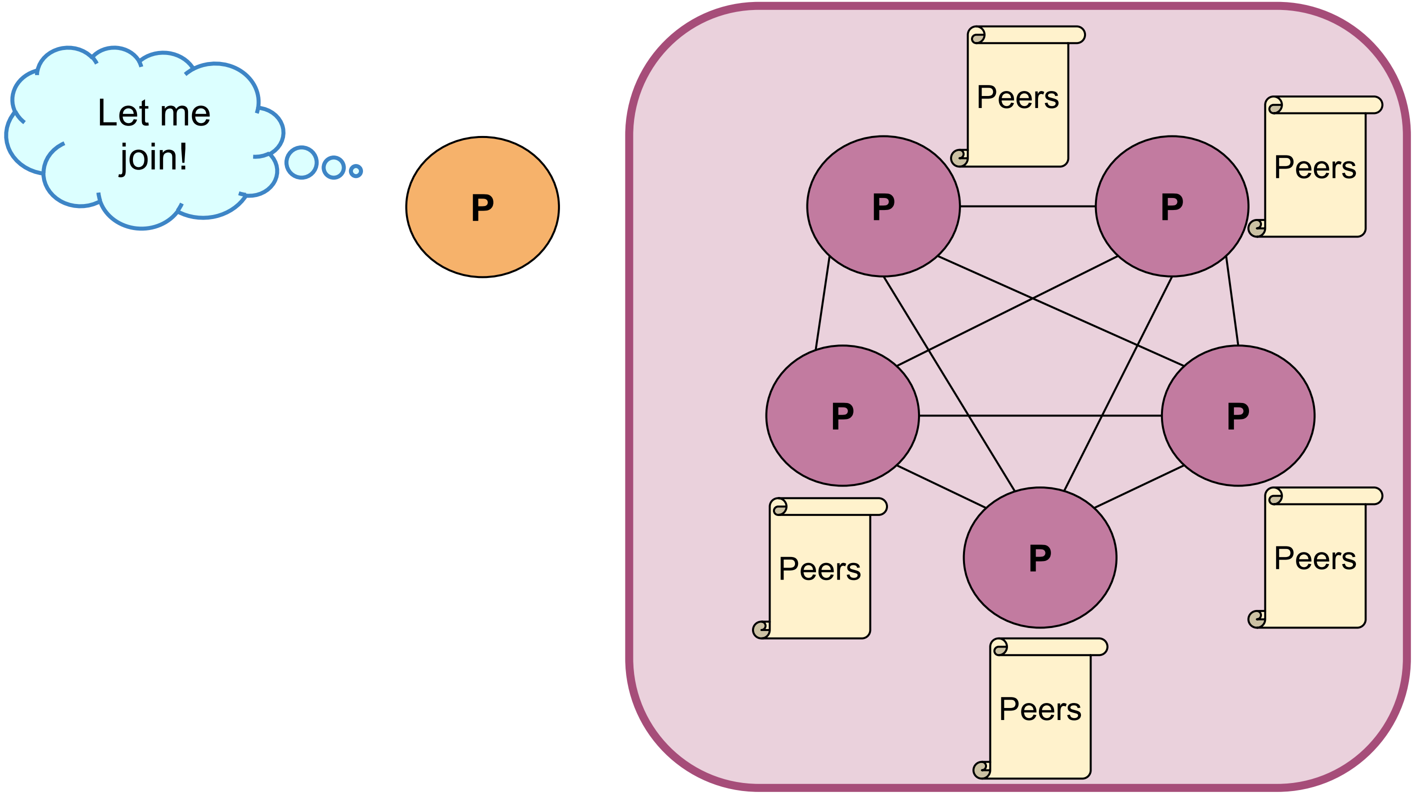
or

Peer that received the request *acts as the leader*

SER 321

Communication

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



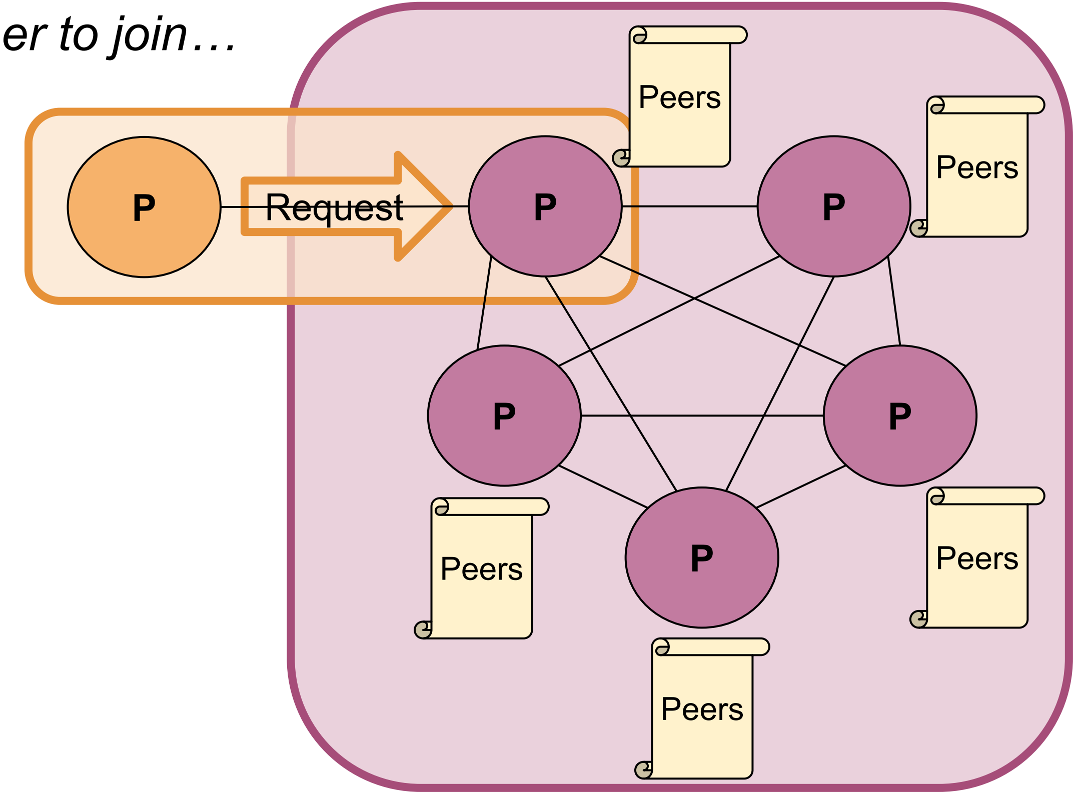
SER 321

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?

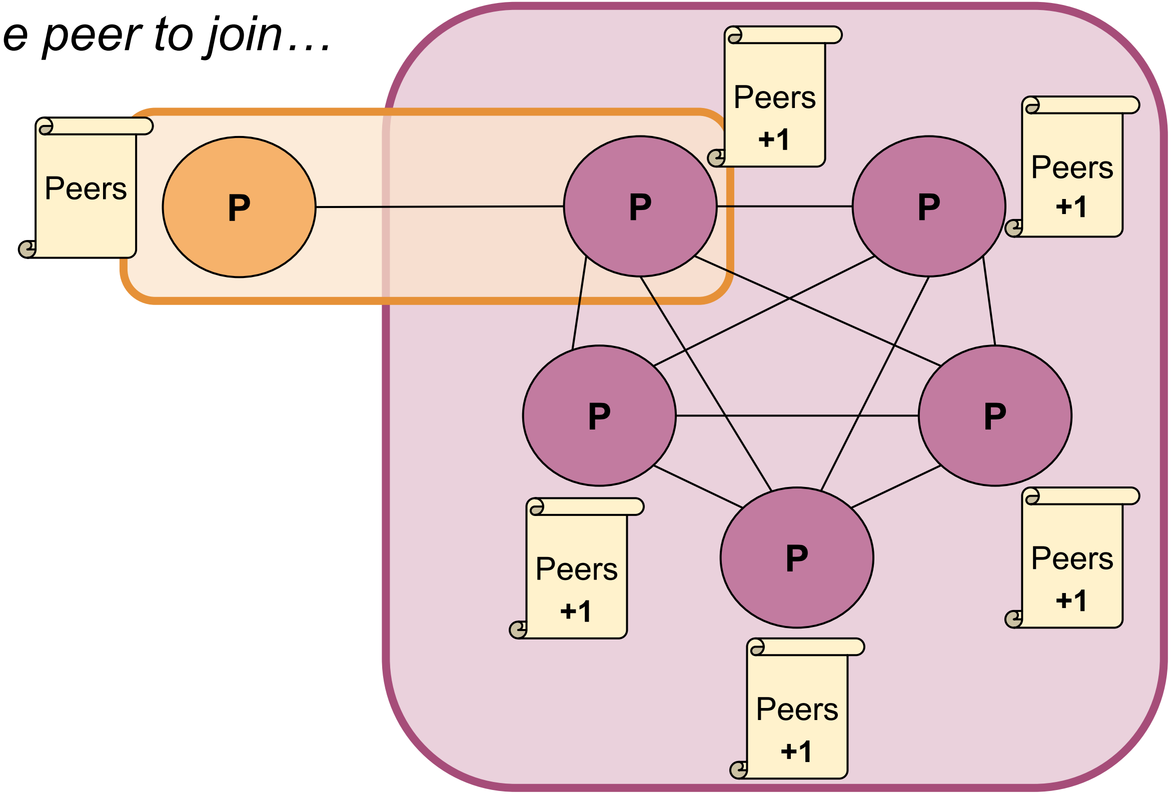
Assuming we want to allow the peer to join…

Three Additional Steps:

1.

2.

3.



SER 321

Scratch Space

Upcoming Events

SI Sessions:

- Sunday, April 20th at 7:00 pm MST
- Tuesday, April 22nd at 10:00 am MST
- Thursday, April 24th at 7:00 pm MST

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?

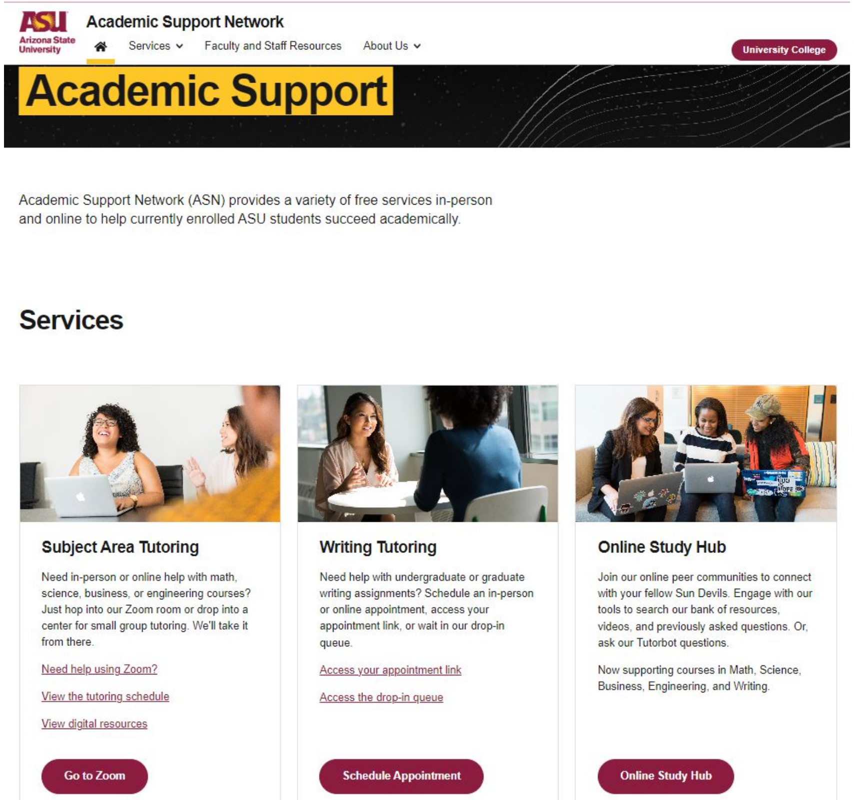
Survey:

https://asuasn.info/ASNSurvey

More Questions?

Check out our other resources!

tutoring.asu.edu



ASU Arizona State University

Academic Support Network

Services ⌄ Faculty and Staff Resources About Us ⌄

University College

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

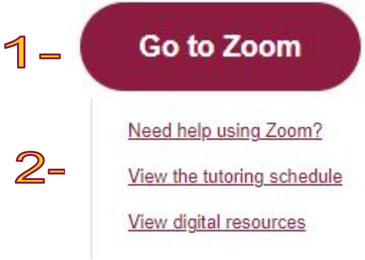
Schedule Appointment

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

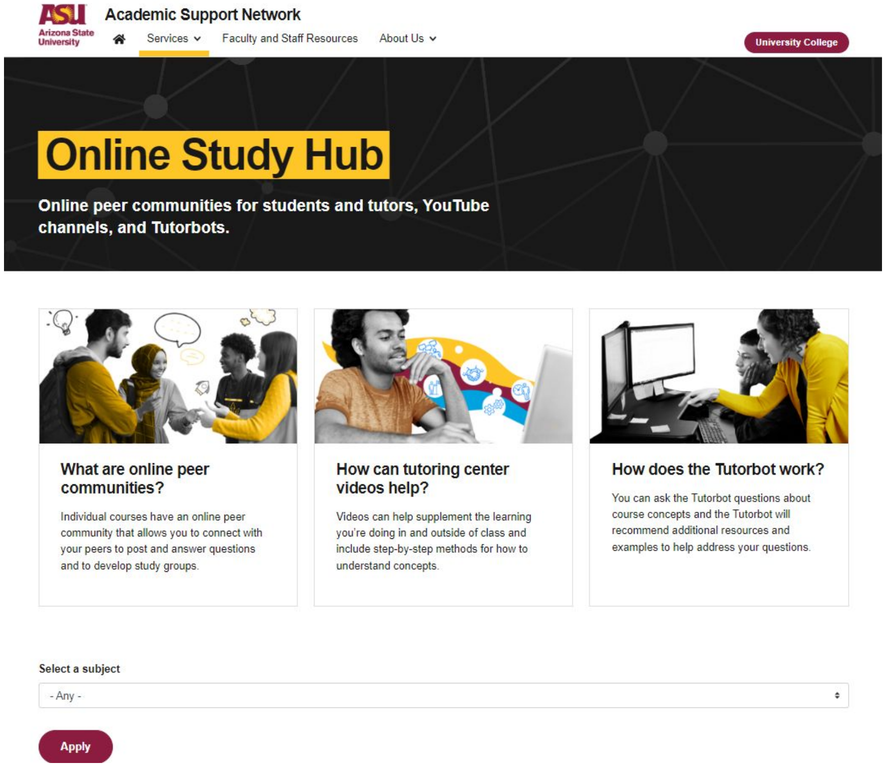


1. **Click on 'Go to Zoom' to log onto our Online Tutoring Center.**
2. **Click on 'View the tutoring schedule' to see when tutors are available for specific courses.**

More Questions?

Check out our other resources!

tutoring.asu.edu/online-study-hub



ASU Arizona State University Academic Support Network

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.

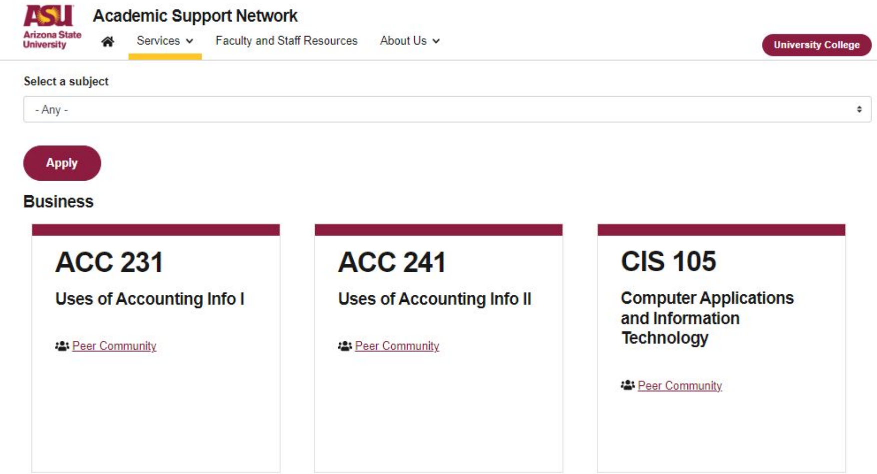
How can tutoring center videos help?

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 -



ASU Arizona State University Academic Support Network

Select a subject: - Any -

Business

ACC 231
Uses of Accounting Info I
Peer Community

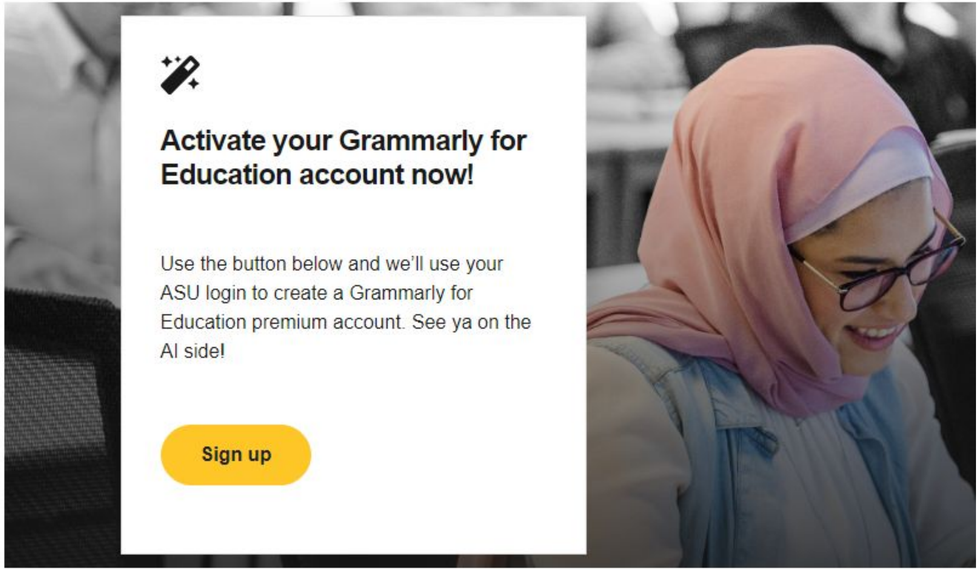
ACC 241
Uses of Accounting Info II
Peer Community

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!

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