SER 321 B Session

SI Session

Thursday November 9th 2023

7:00 - 8:00 pm MST

Agenda

- Threading Pitfalls & Concurrency
- Distributed Systems
- Different 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
Review Session PSA

Scheduling Poll is live in #si_channel

We have roughly 5 sessions remaining, followed by the Review Session

If the Review Session is on Monday (11/27), we can have our regularly scheduled session the day before on Sunday, November 26th

Thursday, November 23rd (Thanksgiving!)
Session is Cancelled for the Holiday

Austin Walter's Traffic Comparison

SER 321

Threading Pitfalls

- Race Condition → More than one thread accesses a single resource at one time

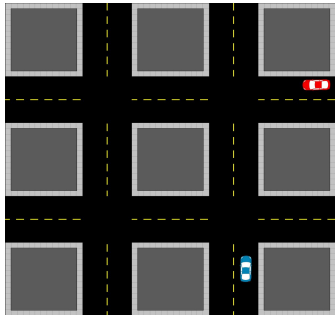
  Crash

- Starvation → One thread never gets access to the resource it needs

  Cross Traffic

- Deadlock → A thread is only able to acquire access to part of its resources

  Gridlock



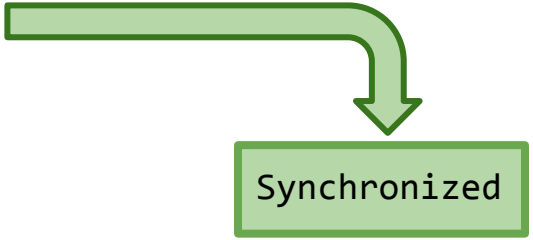
Handling Threaded Pitfalls

Need to prevent threads from stepping on each other!

Mutual Exclusion

Like using a talking stick!

- Locks and semaphores
  - aka Mutex
  - Different types with different capabilities
- Atomic Variables
- Monitor



SER 321

Handling Concurrency

Bow and BowBack are both synchronized

The Synchronized keyword marks that method as a *critical section*

So we should be fine, right?

```
Deadlock [:Deadlock.main()]
Deadlock [:Deadlock.main()]: Running 8 sec
:Deadlock.main() 8 sec

9:44:55 PM: Executing ':Deadlock.main()'...

> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes

> Task :Deadlock.main()
Alphonse: Gaston  has bowed to me!
Gaston: waiting to bow back
Gaston: Alphonse  has bowed to me!
Alphonse: waiting to bow back
```

Deadlock from the examples repo

```java
public class Deadlock {
    6 usages  amehlhase316 *
    static class Friend {
        5 usages
        private final String name;
        2 usages  amehlhase316
        public Friend(String name) { this.name = name; }
        amehlhase316
        public String getName() { return this.name; }
        /* See the README.md for a reference on 'synchronized' methods */
        2 usages  amehlhase316 *
        public synchronized void bow(Friend bower) {
            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);
        }
        1 usage  amehlhase316 *
        public synchronized void bowBack(Friend bower) {
            System.out.format("%s: waiting", this.name);
            System.out.format("%s: %s" + " has bowed back to me!%n", this.name, bower.getName());
        }
    }
    amehlhase316 *
    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 */
        amehlhase316
        new Thread(new Runnable() {
            amehlhase316
            public void run() { alphonse.bow(gaston); }
        }).start();
        amehlhase316
        new Thread(new Runnable() {
            amehlhase316
            public void run() { gaston.bow(alphonse); }
        }).start();
    }
}
```

SAME OBJECT
Cannot both be called at once

SER 321

Handling Concurrency

How do we fix this?

```
Deadlock [:Deadlock.main()]
Deadlock [:Deadlock.main()]: success 626 ms
> Task :Deadlock.main()
Alphonse: Gaston  has bowed to me!
Gaston: waiting to bow back
Gaston: Alphonse  has bowed to me!
Alphonse: waiting to bow back
Alphonse: waiting
Alphonse: Gaston has bowed back to me!
Gaston: waiting
Gaston: Alphonse has bowed back to me!
```

We can synchronize the `run` method of both threads

or

```java
public class Deadlock {
    6 usages  amehlhase316 *
    static class Friend {
        5 usages
        private final String name;
        2 usages  amehlhase316
        public Friend(String name) { this.name = name; }
        amehlhase316
        public String getName() { return this.name; }
        /* See the README.md for a reference on 'synchronized' methods */
        2 usages  amehlhase316 *
        public synchronized void bow(Friend bower) {
            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);
        }
        1 usage  amehlhase316 *
        public synchronized void bowBack(Friend bower) {
            System.out.format("%s: waiting", this.name);
            System.out.format("%s: %s" + " has bowed back to me!%n", this.name, bower.getName());
        }
    }
    amehlhase316 *
    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 */
        amehlhase316 *
        new Thread(new Runnable() {
            new *
            public synchronized void run() { alphonse.bow(gaston); }
        }).start();
        amehlhase316 *
        new Thread(new Runnable() {
            new *
            public synchronized void run() { gaston.bow(alphonse); }
        }).start();
    }
}
```

Deadlock from the examples repo

SER 321
Handling Concurrency

How do we fix this?

Deadlock [:Deadlock.main()]

Deadlock [:Deadlock.main()]: success 480 ms

```
> Task :Deadlock.main()
Alphonse: Gaston  has bowed to me!
Gaston: waiting to bow back
Gaston: waiting
Gaston: Alphonse has bowed back to me!
Gaston: Alphonse  has bowed to me!
Alphonse: waiting to bow back
Alphonse: waiting
Alphonse: Gaston has bowed back to me!
```

or

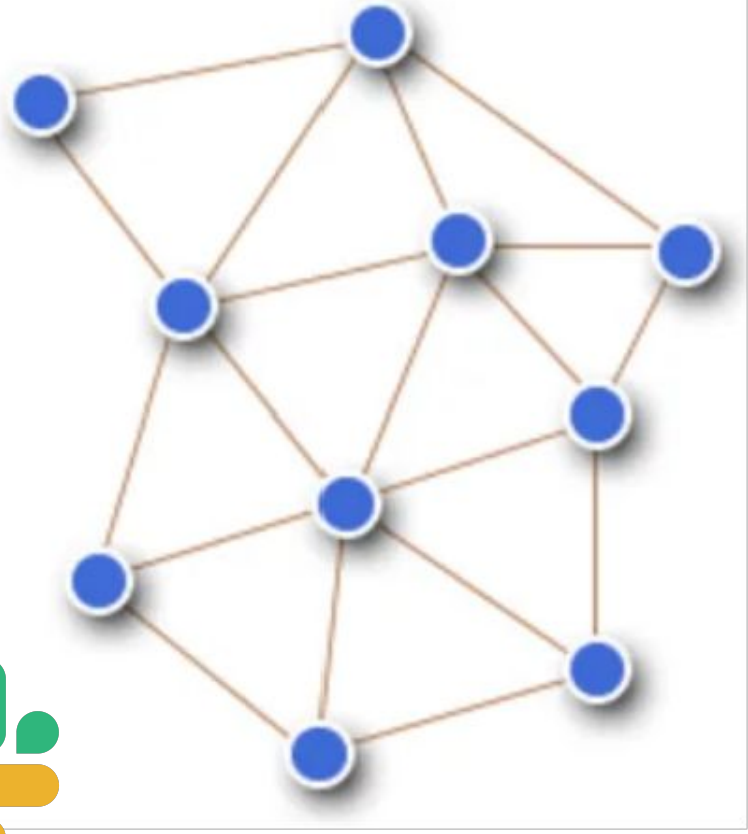
Or we can synchronize the `bowBack` method call

```java
public class Deadlock {
    6 usages  amehlhase316 *
    static class Friend {
        5 usages
        private final String name;
        2 usages  amehlhase316
        public Friend(String name) { this.name = name; }
        amehlhase316
        public String getName() { return this.name; }
        /* See the README.md for a reference on 'synchronized' methods */
        2 usages  amehlhase316 *
        public synchronized void bow(Friend bower) {
            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());
            synchronized(this) {
                bower.bowBack( bower: this);
            }
        }
        1 usage  amehlhase316 *
        public synchronized void bowBack(Friend bower) {
            System.out.format("%s: waiting%n", this.name);
            System.out.format("%s: %s" + " has bowed back to me!%n", this.name, bower.getName());
        }
    }
    amehlhase316 *
    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 */
        amehlhase316
        new Thread(new Runnable() {
            amehlhase316
            public void run() { alphonse.bow(gaston); }
        }).start();
        amehlhase316
        new Thread(new Runnable() {
            amehlhase316
            public void run() { gaston.bow(alphonse); }
        }).start();
    }
}
```

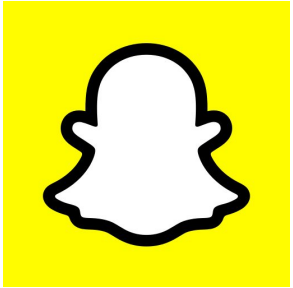
Deadlock from the examples repo

SER 321
Distributed Algorithms

What's a distributed system again?



Many *nodes* working together that *appear* to be a single system from the outside



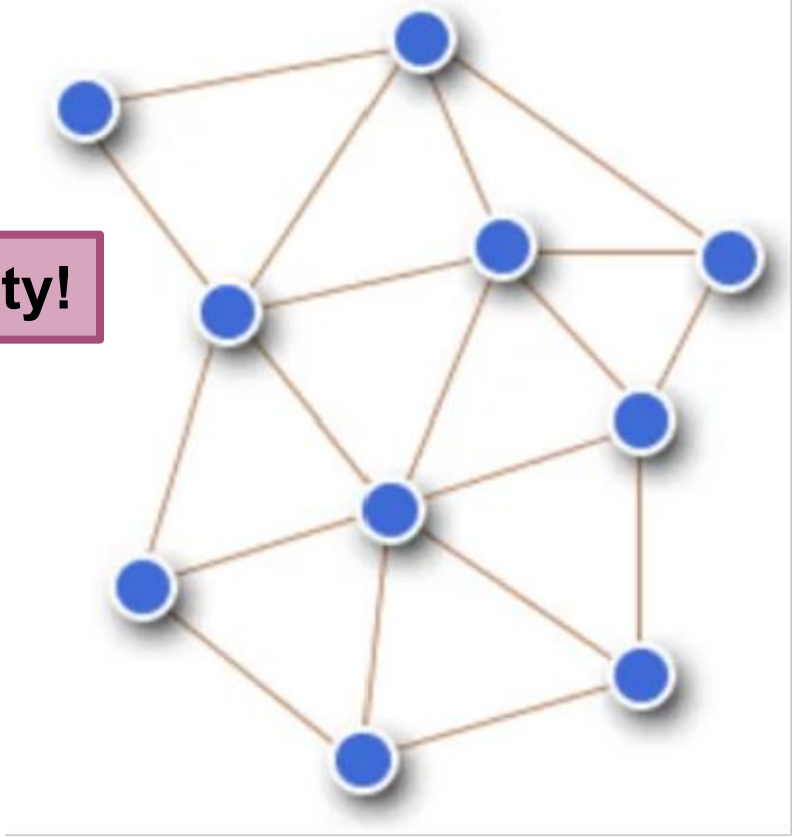
zoom

SER 321

Distributed Algorithms

Common Issues: **aka working in reality!**

- Handle Node failures
- Account for latency
- Account for network failures
- Protection of shared resources
- Prevention of deadlocks
- Execution safety - no errors or gross, bad stuff
- Ensuring liveliness - everyone goes eventually



We look at two main structure forms:

Peer to Peer

Main and Worker

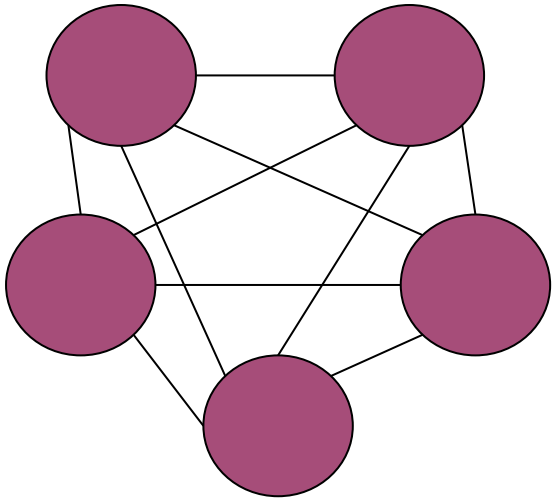
SER 321
Distributed Algorithms

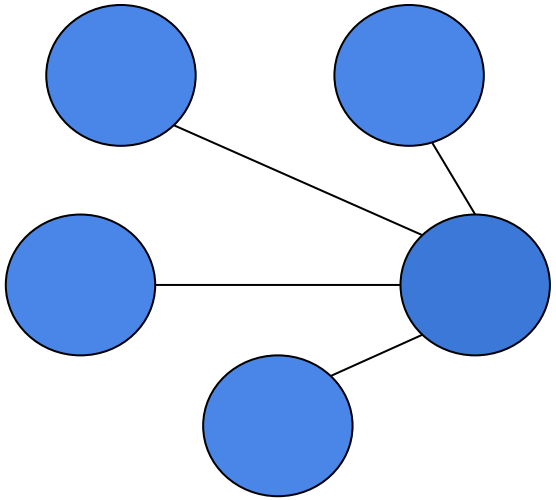
Which is which?

Main and Worker

Peer to Peer

Check out the recording for the solution!

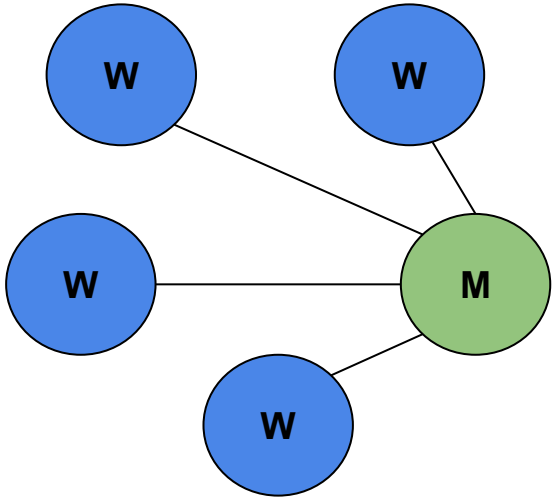




Pros and Cons

SER 321

Distributed Algorithms



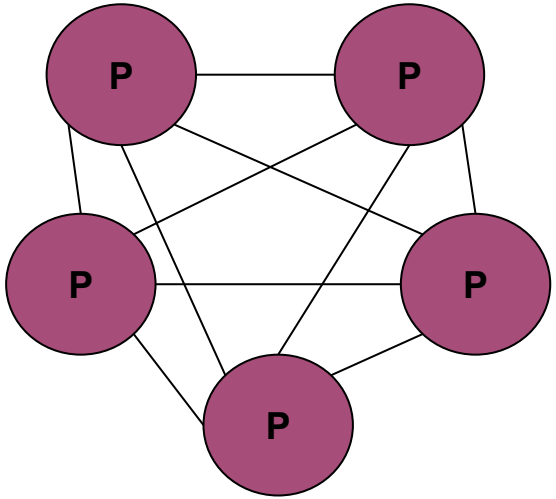
PRO

CON

Check out the recording for the solution!

Pros and Cons

SER 321
Distributed Algorithms



PRO

CON

Check out the recording for the solution!

SER 321
Consensus

What is Consensus?

A. Systematic calculating and recording of information about a given population

B. General agreement or trust amongst a group

C. Controversial; causing or likely to cause an argument

D. No Idea…

Check out the recording for the solution!

Questions?

Survey:

https://bit.ly/asn_survey

18

Upcoming Events

SI Sessions:

- Sunday, November 12th 2023 at 7:00 pm MST
- Monday, November 13th 2023 at 4:00 pm MST
- Thursday, November 16th 2023 at 7:00 pm MST
- Sunday, November 19th 2023 at 7:00 pm MST
- Monday, November 20th 2023 at 4:00 pm MST

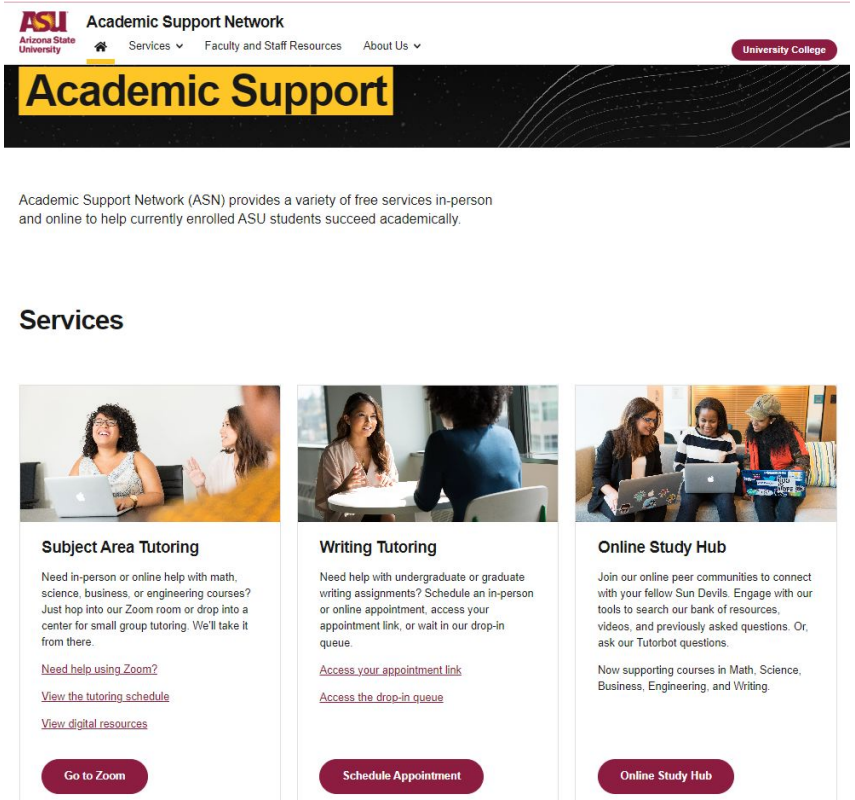
Review Sessions:

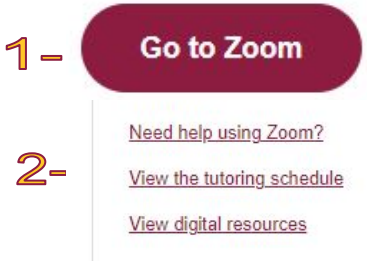
- Survey is LIVE in the #si_channel
- Sunday, November 26th or Monday November 27th

More Questions?

Check out our other resources!

tutoring.asu.edu



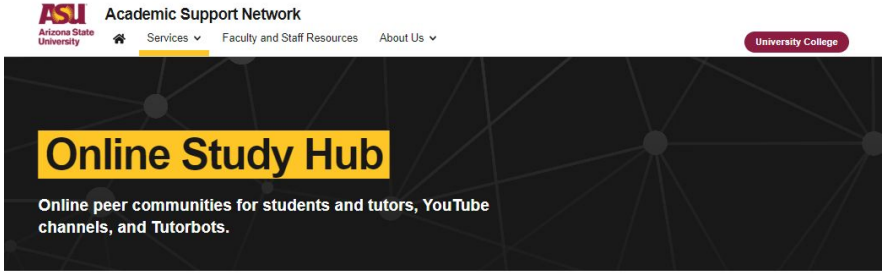


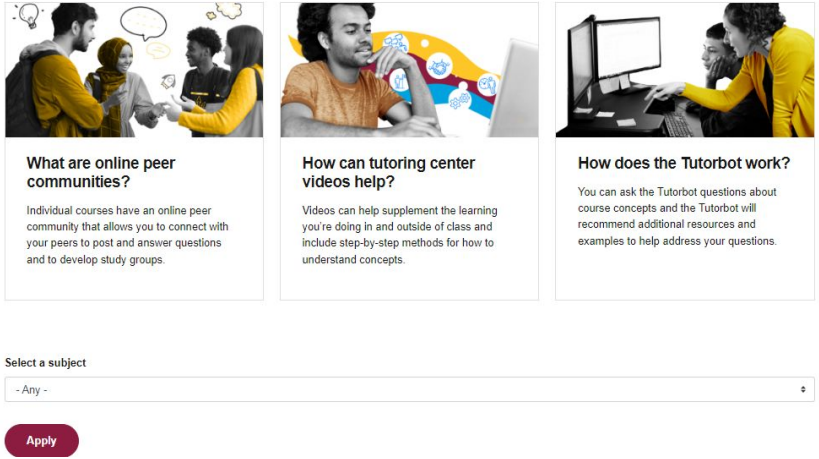
1. **Click on 'Go to Zoom' to log onto our Online Tutoring Center.**
2. **Click on 'View the tutoring schedule' to see when tutors are available for specific courses.**

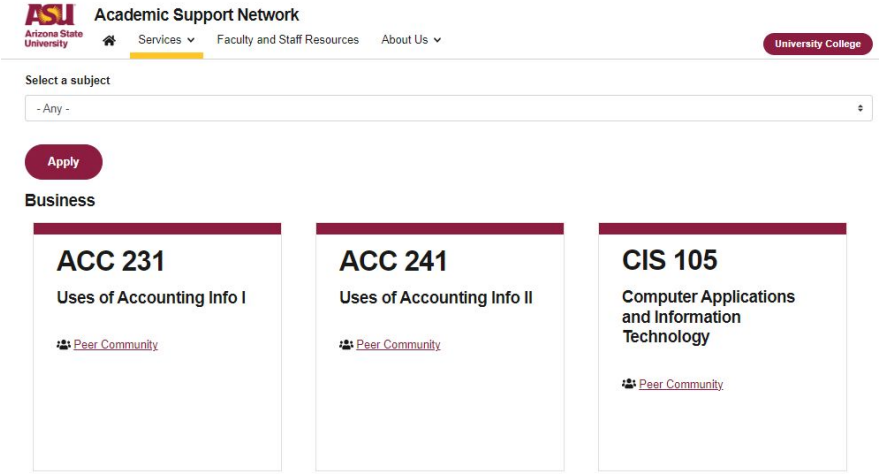
More Questions?

Check out our other resources!

tutoring.asu.edu/online-study-hub







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

Additional Resources

CoureRepo

Dining Philosophers Interactive

Austin Walter’s Traffic Comparison

RAFT