Deadlock

Deadlock is only possible if certain *necessary* conditions are met: the *Coffman Conditions*

Deadlock

Deadlock is only possible if certain *necessary* conditions are met: the *Coffman Conditions*

1. **Mutual exclusion** Only one process can use a resource at a time

Deadlock

Deadlock is only possible if certain *necessary* conditions are met: the *Coffman Conditions*

1. **Mutual exclusion** Only one process can use a resource at a time
2. **Hold-and-wait** A process continues to hold a resource while waiting for other resources

Deadlock

Deadlock is only possible if certain *necessary* conditions are met: the *Coffman Conditions*

1. **Mutual exclusion** Only one process can use a resource at a time
2. **Hold-and-wait** A process continues to hold a resource while waiting for other resources
3. **No preemption** No resource can forcibly be removed from a process holding it

Deadlock

Deadlock is only possible if certain *necessary* conditions are met: the *Coffman Conditions*

1. **Mutual exclusion** Only one process can use a resource at a time
2. **Hold-and-wait** A process continues to hold a resource while waiting for other resources
3. **No preemption** No resource can forcibly be removed from a process holding it

All of these must hold for it to be *possible* to deadlock

Deadlock

It might seem easy to avoid these, but in practice it's harder than you think

Deadlock

It might seem easy to avoid these, but in practice it's harder than you think

Suppose we ensure Hold-and-wait never happens, e.g., requiring a process to drop other resources it holds whenever it gets blocked on a new request

Deadlock

It might seem easy to avoid these, but in practice it's harder than you think

Suppose we ensure Hold-and-wait never happens, e.g., requiring a process to drop other resources it holds whenever it gets blocked on a new request

When it gets the new resource it will have to go back and pick up the other resources again

Deadlock

It might seem easy to avoid these, but in practice it's harder than you think

Suppose we ensure Hold-and-wait never happens, e.g., requiring a process to drop other resources it holds whenever it gets blocked on a new request

When it gets the new resource it will have to go back and pick up the other resources again

Which may require it to drop the new resource while waiting. . .

Deadlock

It might seem easy to avoid these, but in practice it's harder than you think

Suppose we ensure Hold-and-wait never happens, e.g., requiring a process to drop other resources it holds whenever it gets blocked on a new request

When it gets the new resource it will have to go back and pick up the other resources again

Which may require it to drop the new resource while waiting. . .

It is easy to get into a situation where the process never manages to get all the resources it needs: called *indefinite postponement*

Deadlock

A deadlock may be possible but will only actually happen if

4. **Circular Wait** There is a circular chain of processes where each holds a resource that is needed by the next in the circle

Deadlock

A deadlock may be possible but will only actually happen if

4. **Circular Wait** There is a circular chain of processes where each holds a resource that is needed by the next in the circle

This says that deadlock is happening as in the formal definition

Deadlock

Dining Philosophers

A popular illustration of deadlock is *The Dining Philosophers*

Deadlock

Dining Philosophers

A popular illustration of deadlock is *The Dining Philosophers*

Some Philosophers wish to share a plate of spaghetti, but they have only been provided with chopsticks

Deadlock

Dining Philosophers

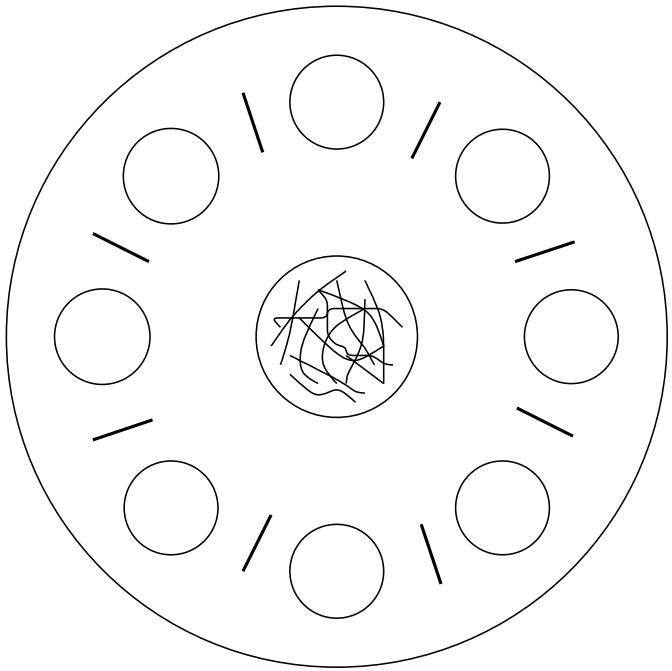
A popular illustration of deadlock is *The Dining Philosophers*

Some Philosophers wish to share a plate of spaghetti, but they have only been provided with chopsticks

Unfortunately, there is not quite enough chopsticks to go around

Deadlock

Dining Philosophers



Dining Philosophers

Deadlock

Dining Philosophers

Each Philosopher needs two chopsticks to eat, one from each side of their plate

Deadlock

Dining Philosophers

Each Philosopher needs two chopsticks to eat, one from each side of their plate

We have

Deadlock

Dining Philosophers

Each Philosopher needs two chopsticks to eat, one from each side of their plate

We have

1. Mutual exclusion. Only one Philosopher can use a chopstick at a time

Deadlock

Dining Philosophers

Each Philosopher needs two chopsticks to eat, one from each side of their plate

We have

1. Mutual exclusion. Only one Philosopher can use a chopstick at a time
2. Hold-and-wait. Each Philosopher wants to eat and won't let go of a chopstick until they have eaten

Deadlock

Dining Philosophers

Each Philosopher needs two chopsticks to eat, one from each side of their plate

We have

1. Mutual exclusion. Only one Philosopher can use a chopstick at a time
2. Hold-and-wait. Each Philosopher wants to eat and won't let go of a chopstick until they have eaten
3. No preemption. No-one is going to tell a Philosopher what to do!

Deadlock

Dining Philosophers

And if they all grab the left chopstick simultaneously

Deadlock

Dining Philosophers

And if they all grab the left chopstick simultaneously

4. Circular Wait. There is a circular chain of Philosophers where each holds a chopstick that is needed by the next in the circle

Deadlock

Dining Philosophers

And if they all grab the left chopstick simultaneously

4. Circular Wait. There is a circular chain of Philosophers where each holds a chopstick that is needed by the next in the circle

Of course, if the Philosophers were a bit more friendly, or polite, there would not be a problem

Deadlock

Dining Philosophers

Exercise. Identify the conditions in the car gridlock scenarios

□