Deadlock

David Ferry
CSCI 2510 – Principles of Computing Systems
Saint Louis University
St. Louis, MO 63103

Blocking Operations

Recall- Some operations cause a program to wait.

Quiz: Which of the following operations could cause blocking?

A. Locking a mutex
B. Opening a file
C. Reading a file
D. Reading from a pipe
E. Writing to a pipe

Blocking Operations

Recall- Some operations cause a program to wait.

Quiz: Which of the following operations could cause blocking?

A. Locking a mutex (already locked)
B. Opening a file (disk I/O)
C. Reading a file (disk I/O)
D. Reading from a pipe (pipe is empty)
E. Writing to a pipe (pipe is full)

Deadlock

A specific hazard occurs with blocking operations:

- Could an adversary scheduler break our code?

```
mutex a, b;
```

Thread 1:

```
lock(a)
lock(b)
//Critical section
unlock(b)
unlock(a)
```

Thread 2:

```
lock(b)
lock(a)
//Critical section
unlock(a)
unlock(b)
```

Deadlock

A specific hazard occurs with blocking operations:

- Could an adversary scheduler break our code?

```
mutex a, b;
```

Thread 1:

1. lock(a)
4. lock(b)

```
   //Critical section
   unlock(b)
   unlock(a)
```

Thread 2:

2. lock(b)
3. lock(a)

```
   //Critical section
   unlock(a)
   unlock(b)
```

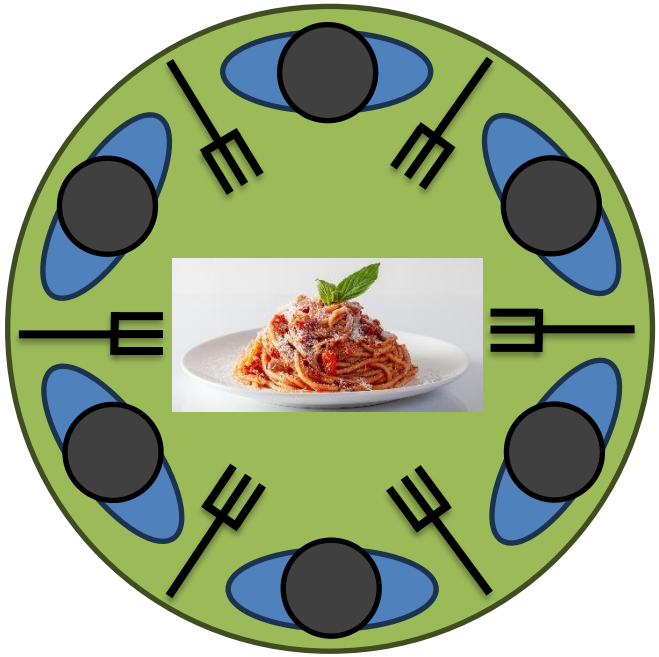
Classic Deadlock Example: Dining Philosophers

A group of philosophers are sharing a delicious spaghetti meal.

Six thinkers, six forks, each needs two forks to eat.

Each philosopher follows the procedure:

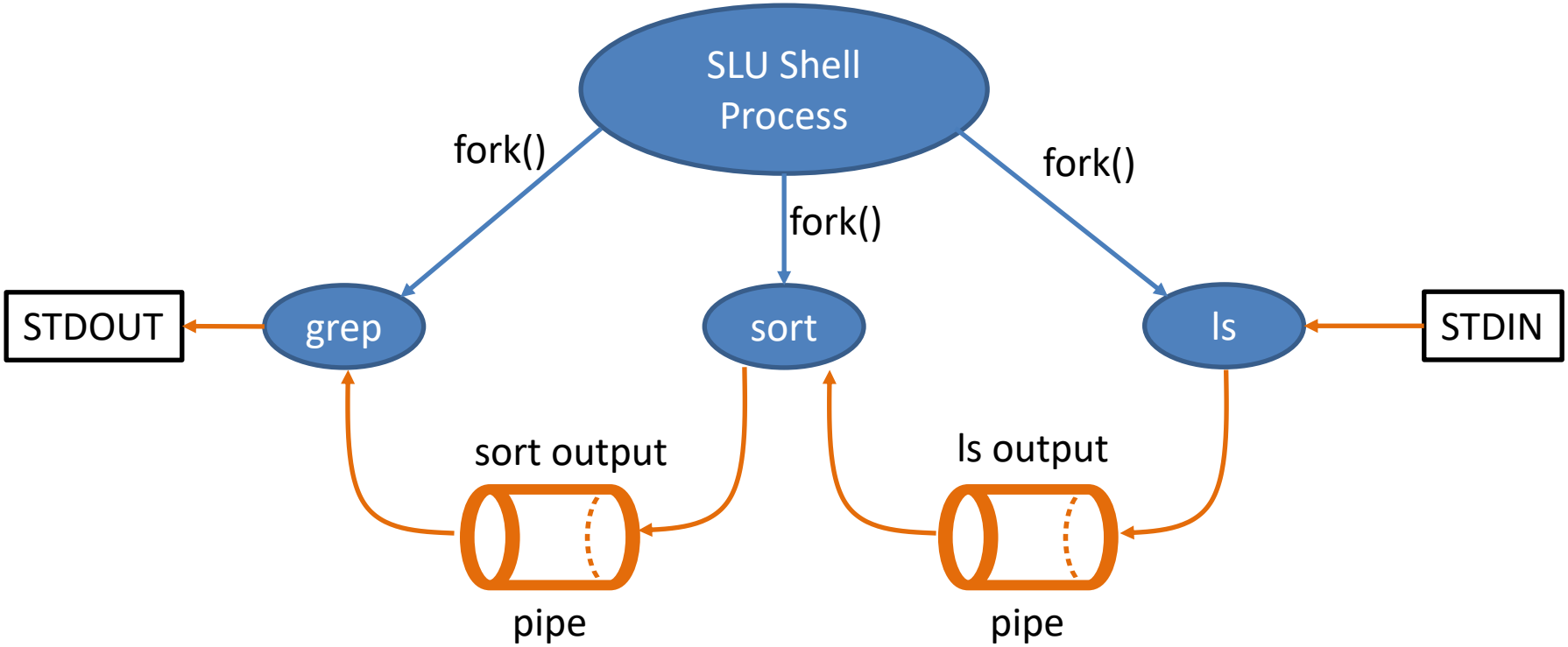
```
while( 1 ){
   1. Ponder mysteries for a while
   2. Grab fork to their left (or wait)
   3. Grab fork to their right (or wait)
   4. Eat
   5. Release forks
}
```



Spaghetti: Powerpoint Stock Photos

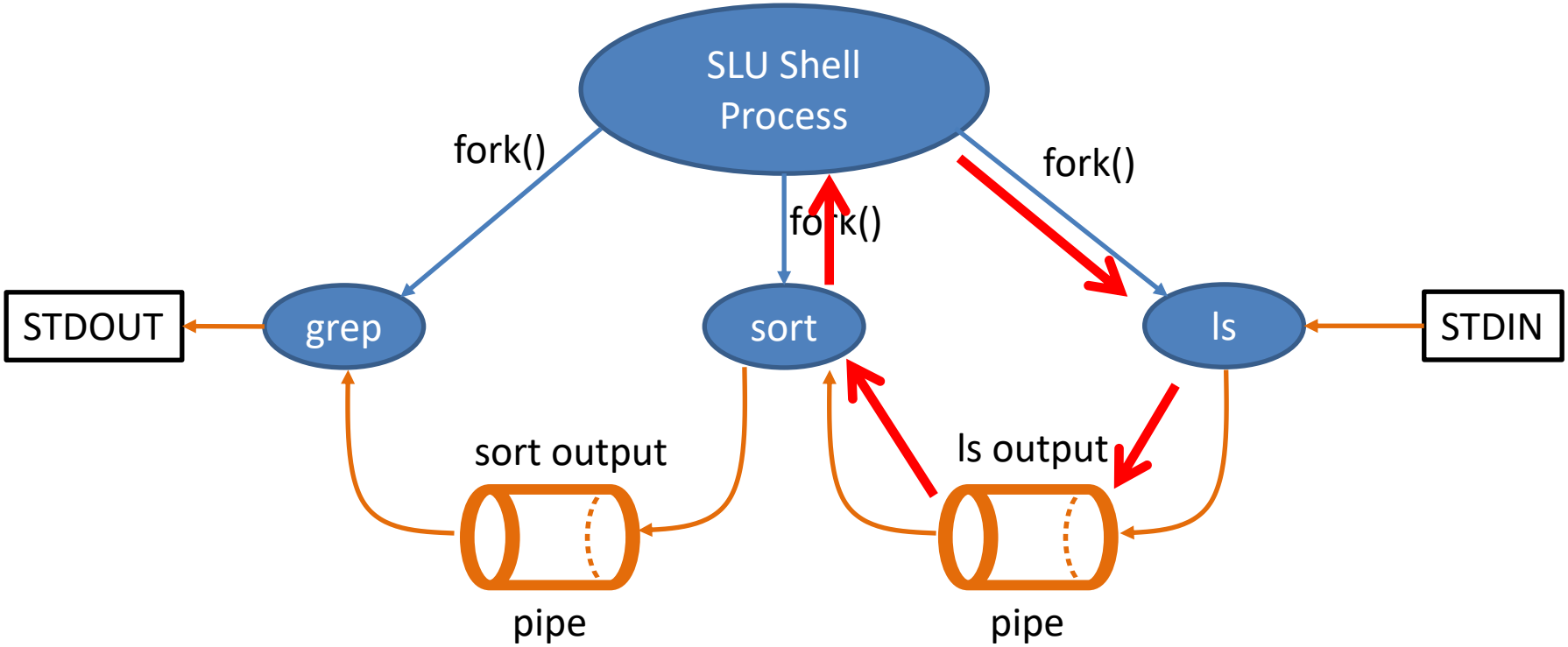
Recall – Lab 2

How could deadlock occur if the parent shell process *waits* on each child after forking instead of just the last?



Recall – Lab 2

How could deadlock occur if the parent shell process *waits* on each child after forking instead of just the last?



Necessary Conditions for Deadlock

Which of the following elements occurred in all three examples?

- Philosophers
- Graph cycles
- Processes
- Threads
- Resource exclusion
- Locking
- Mutex variables
- Waiting
- Pipes
- Spaghetti

Necessary Conditions for Deadlock

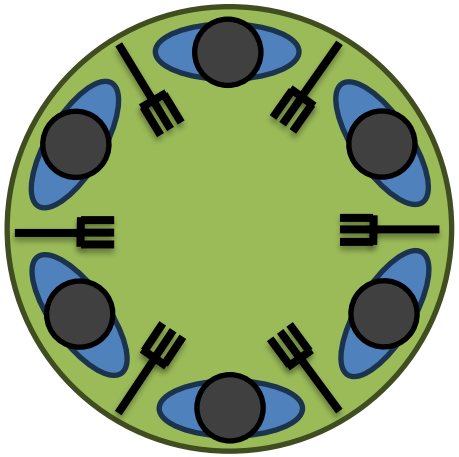
Which of the following elements occurred in all three examples?

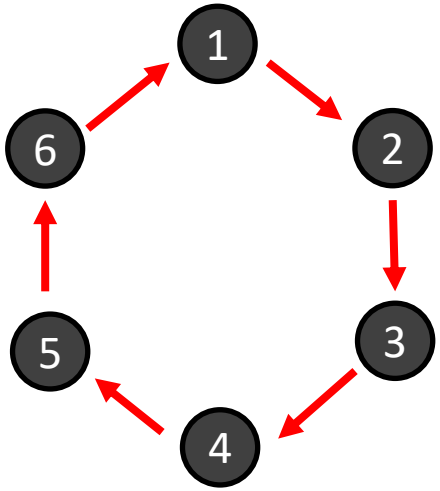
Philosophers	Locking
Graph cycles	Mutex variables
Processes	Waiting
Threads	Pipes
Resource exclusion	Spaghetti

Deadlock is not just a property of threads, locks, and mutexes- it is a general systems problem.

Deadlock Conditions

1. Resource exclusion – Processes can hold resources in a non-shareable state
2. Hold-while-waiting – Processes can hold a resource while waiting for another resource
3. No preemption – Only the process holding a resource can release it
4. Circular wait – There can exist a set of waiting processes $P_1 \ldots P_N$ such that P_1 waits on P_2, P_2 waits on P_3, … P_{N-1} waits on P_N, and P_N waits on P_1

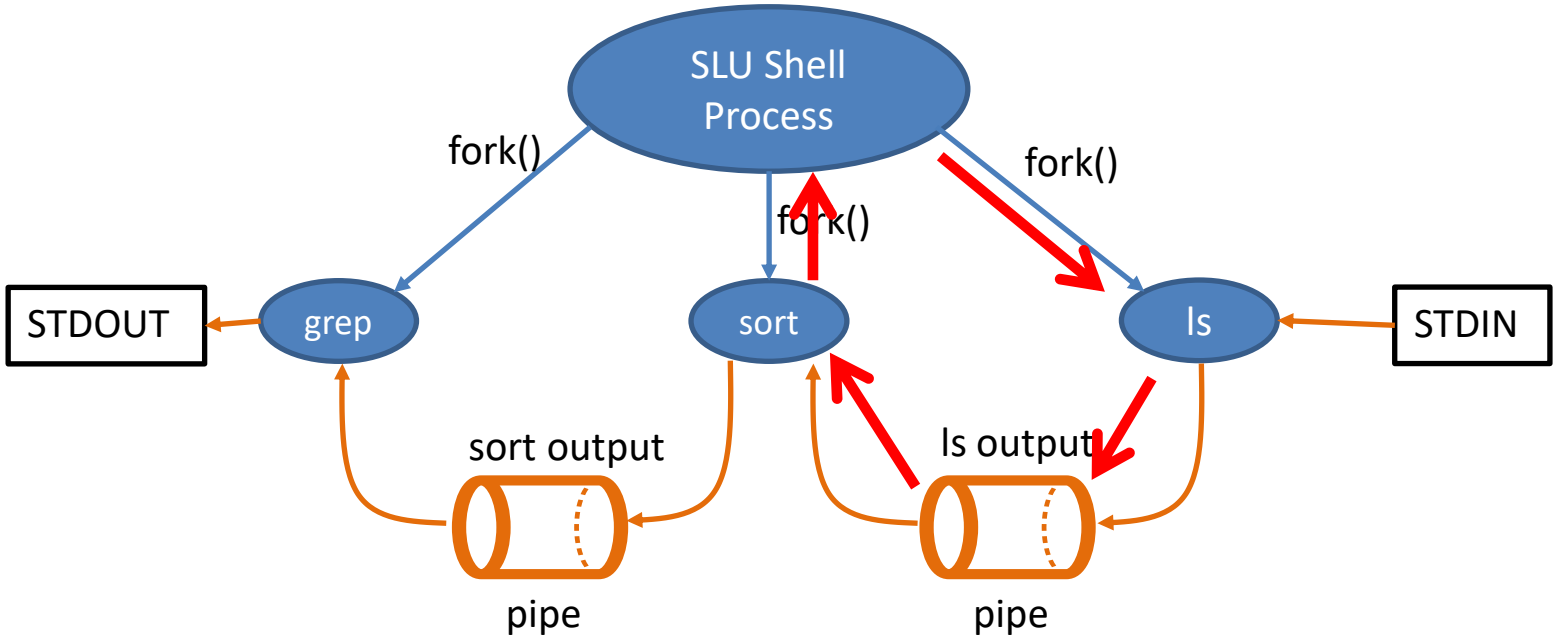




Deadlock does not require explicit resource acquisition

The shell process calls wait() on the first child process, the child process calls write() on the pipe. The pipe is full. No process ever explicitly acquires a resource.

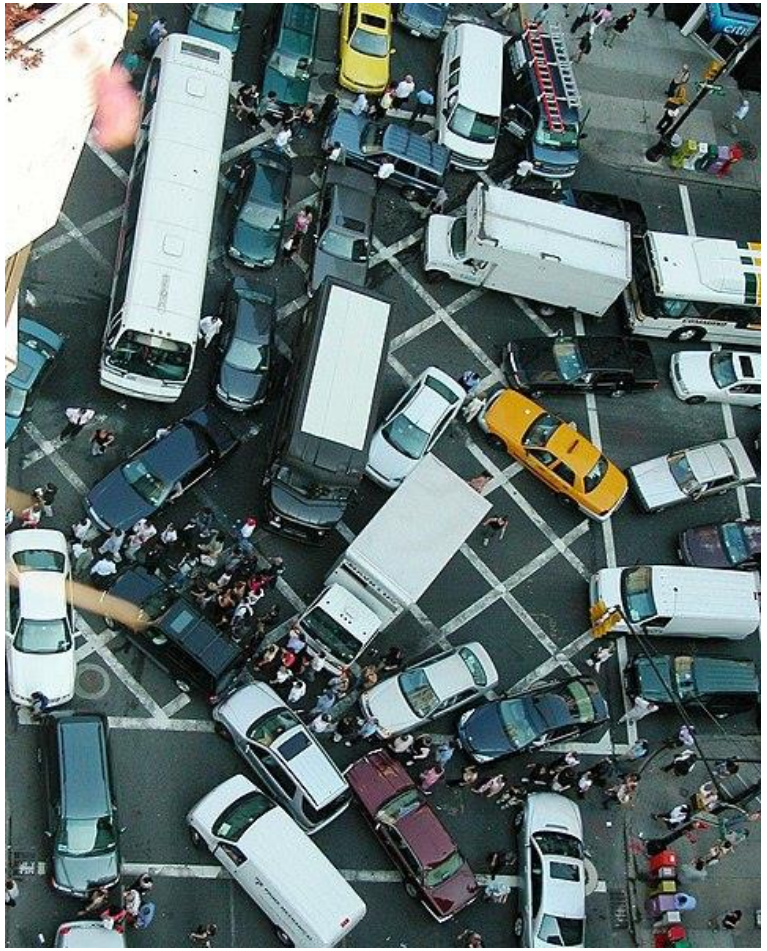
- Possibility of deadlock is an inherent property of a design
- That said, most of the time we talk about explicit acquisition



Real-World Example: Gridlock

Is this deadlock?

- Resource exclusion
- Hold-while-waiting
- No preemption
- Circular wait

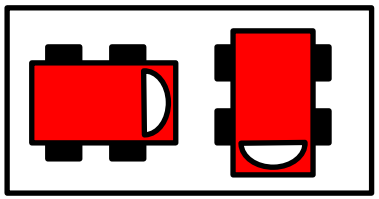


Attribution: Rgoogin at the English Wikipedia

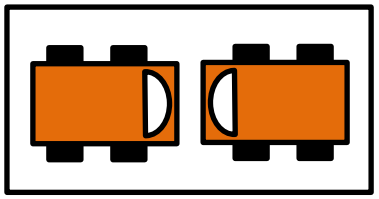
Which of these contain Deadlock?

Rule: Cars must drive in a straight line until they're out of their box.

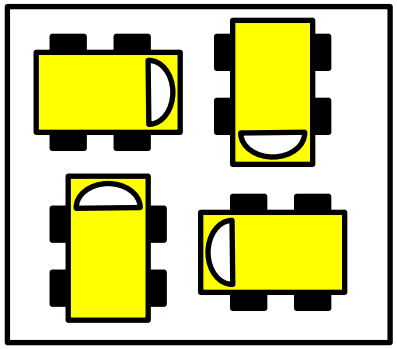
A



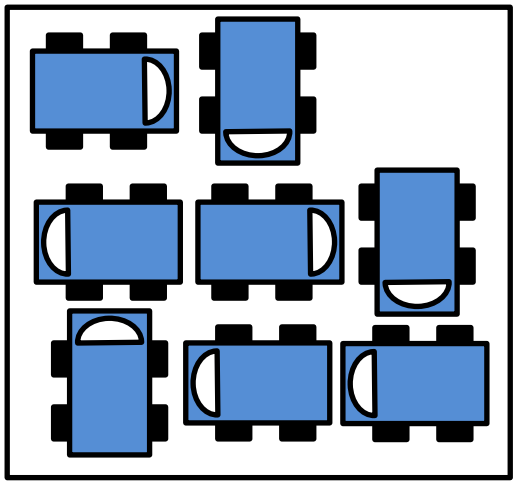
B



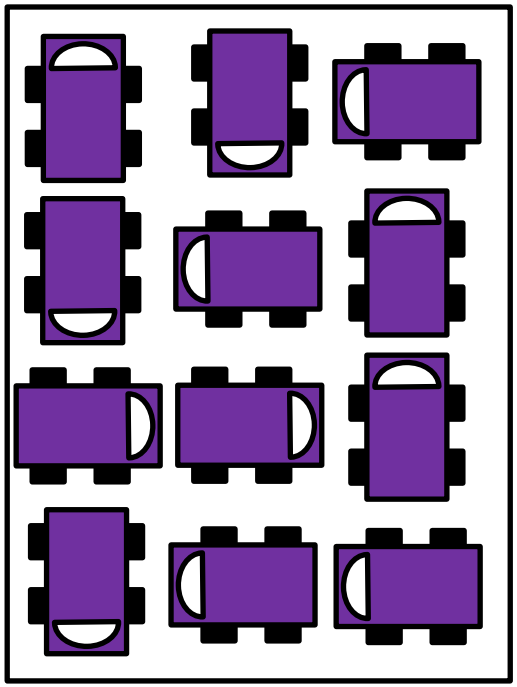
C



D



E

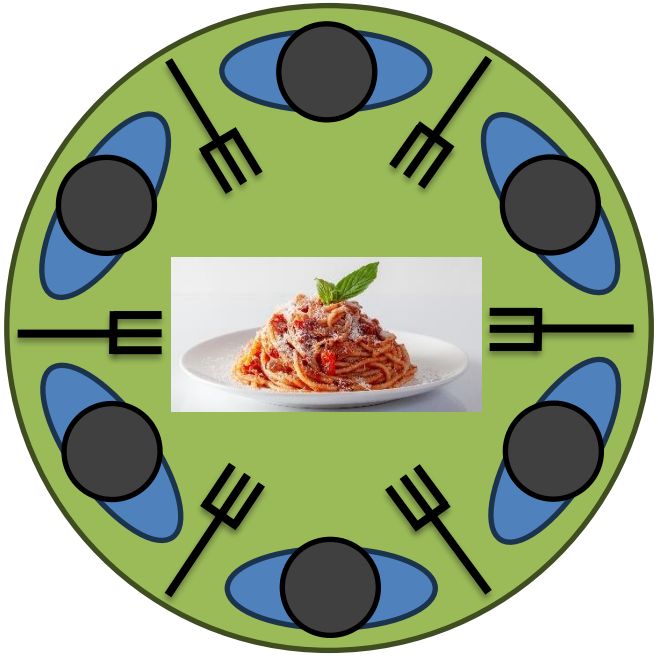


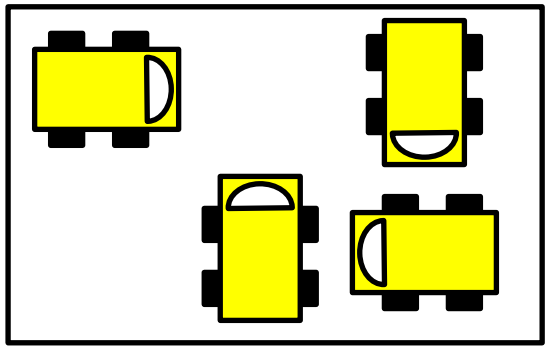
Can Deadlock is not Will Deadlock

Deadlock may or may not occur during any specific execution

- Usually results from a race, so depends on timing
- Just like race conditions, thorough testing is not guarantee of finding deadlock
- Developer must reason about the system and convince themselves the system is deadlock-free
- E.g. Philosophers don't always deadlock

Exercise: Give an execution for Dining Philosophers that doesn't deadlock





Starvation

Related concept, will only introduce here:

- Starvation – Individual processes may be "unlucky" and unable to progress
- Example: Threads race and 1 never gets to lock(a)

mutex a, b

Thread 1
```
lock(a)
lock(b)
//process data
unlock(b)
unlock(a)
```

Thread 2
```
lock(a)
lock(b)
//process data
unlock(b)
unlock(a)
```

Thread 3
```
lock(a)
lock(b)
//process data
unlock(b)
unlock(a)
```

Livelock

Related concept, will only introduce here:

- Livelock – Threads aren't blocked/waiting but also never make progress
- Example: Suppose we have "polite threads" that release their first lock if not able to acquire the second lock.

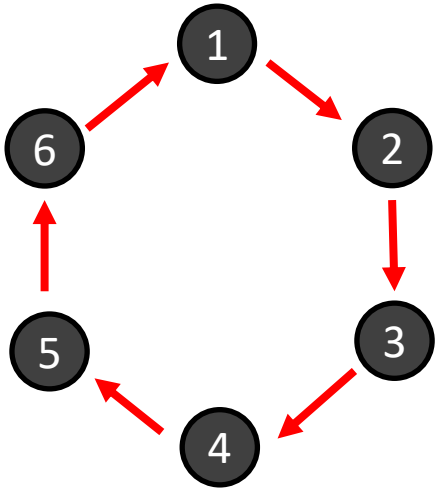
Thread 1: Lock a
Thread 2: Lock b
Thread 1: Try to lock b and fail
Thread 2: Try to lock a and fail
Thread 1: Release a
Thread 2: Release b
Repeat

Deadlock Analysis

Deadlock scenarios are usually analyzed with a *directed graph* called a *resource allocation graph:*

Recall: A directed graph G = (V,E) is a set of vertices V and directed edges E, such that an edge a→b does not imply an edge b→a.

Example:
V = {1, 2, 3, 4, 5, 6}
E = {1→2, 2→3, 3→4,
4→5, 5→6, 6→1}

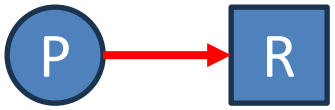


Resource Allocation Graph

The RAG has two sets of vertices-

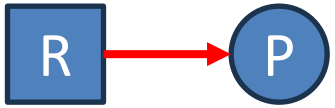
- A set processes/threads drawn as circles
- A set of resources drawn as squares

Edge from P to R: P is trying to acquire R



P is waiting for R

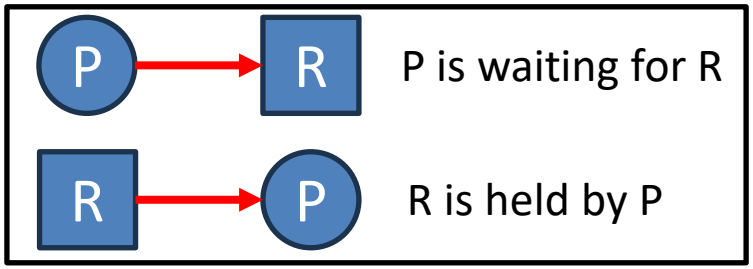
Edge from R to P: P is holding R



R is held by P

The Deadly Embrace

A cycle in the RAG indicates deadlock!



mutex a, b;

Thread P:

1. lock(a)
4. lock(b)

Thread Q:

2. lock(b)
3. lock(a)

Infinite wait

RAG:

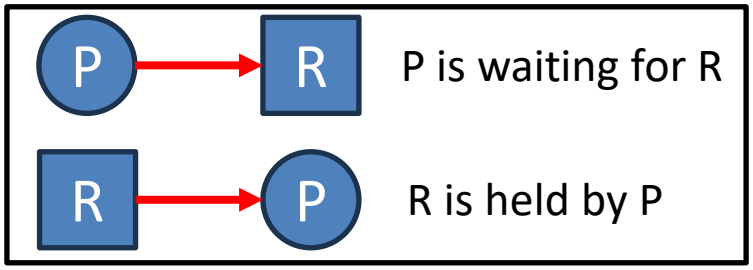
Processes =

Resources =

Edges =

The Deadly Embrace

A cycle in the RAG indicates deadlock!



mutex a, b;

Thread P:

1. lock(a)
4. lock(b)

Thread Q:

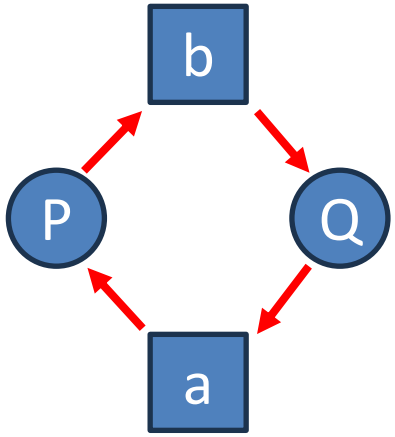
2. lock(b)
3. lock(a)

Infinite wait

RAG:

Processes = { P, Q } Resources = {a, b}

Edges = { a→P, b→Q, Q→a, P→b }



Exercise

Does the following sequence result in deadlock? If so, at which step does the system deadlock?

Processes = { P, Q, R} Resources = {a, b, c, d, e}

1 P.acquire(a)
2 Q.acquire(c)
3 R.acquire(b)
4 P.acquire(c)
5 Q.acquire(d)
6 R.acquire(a)
7 Q.acquire(b)
8 R.acquire(e)