Operating Systems:

Chap 6: DeadLocks

CS400

Week 13: 8th April

Spring 2020

Oliver BONHAM-CARTER

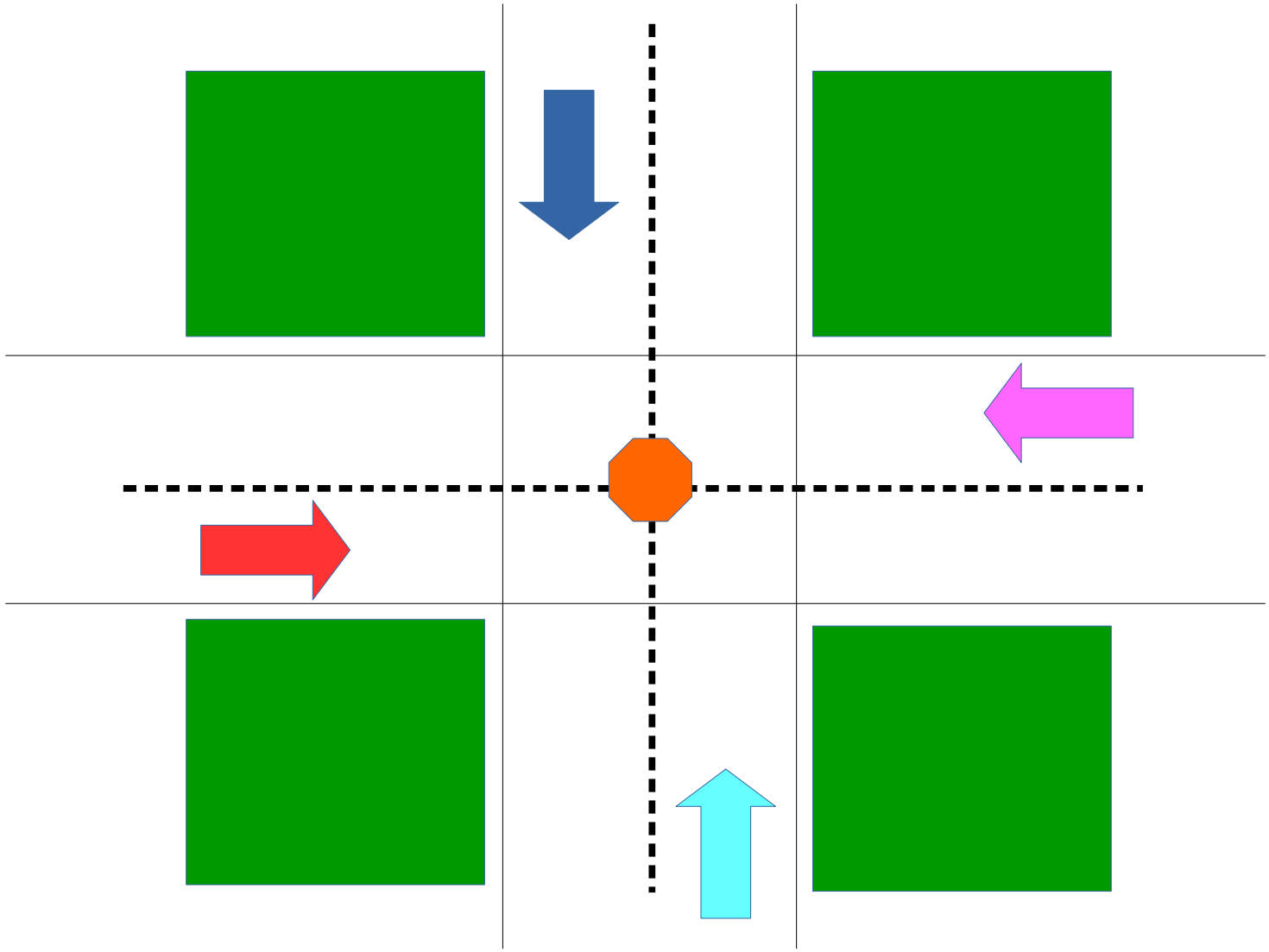
DeadLocks

- A deadlock is a *state* in which each member of a group of actions, is waiting for some other member to release a lock.
- Deadlock is a common problem in multiprocessing systems, parallel computing, and distributed systems, where software and hardware locks are used to handle shared resources and implement process synchronization.

DeadLocks and Threads

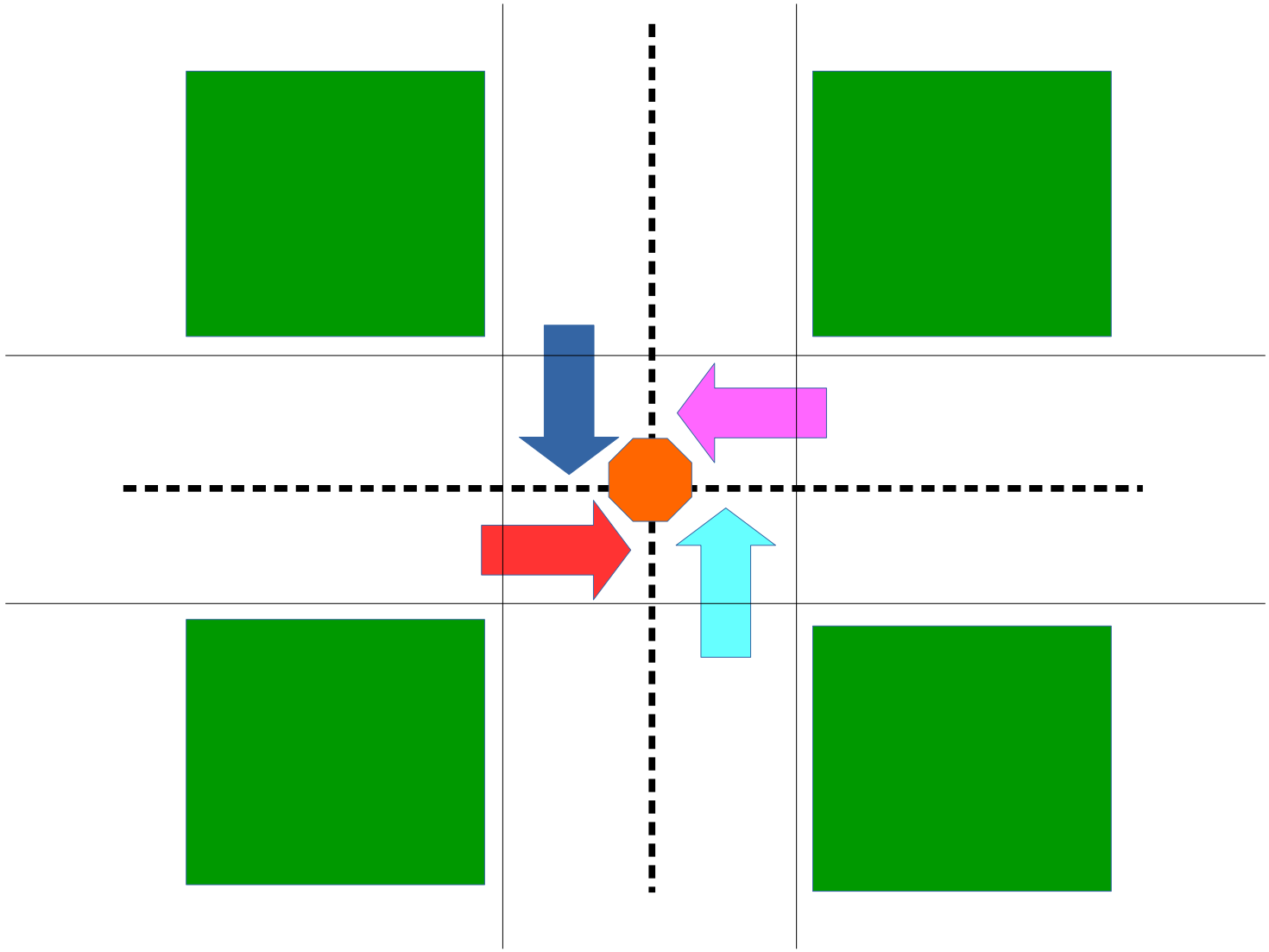
- Occurs when a process or thread enters a waiting state because a requested system resource is held by another waiting process, which, in turn, is waiting for another resource held by another waiting process.
- If a process is unable to change its state indefinitely because the resources requested by it are being used by another waiting process, then the system is said to be in a deadlock.
- No one can move.

Visual Description (1)



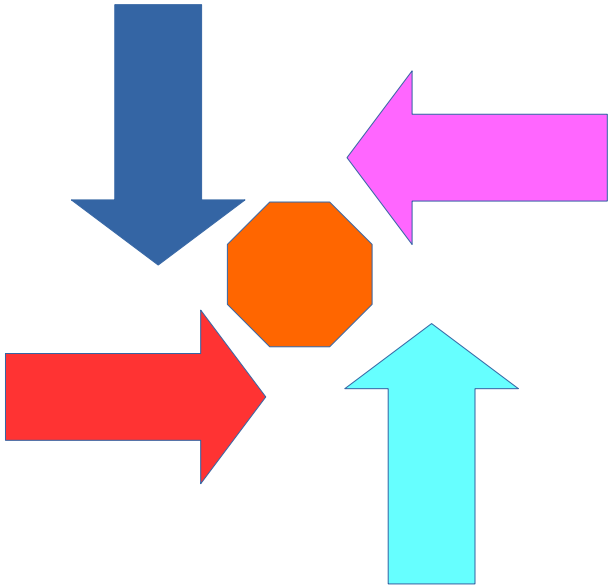
ALLEGHENY COLLEGE

Visual Description (2)



ALLEGHENY COLLEGE

No Car Can Move

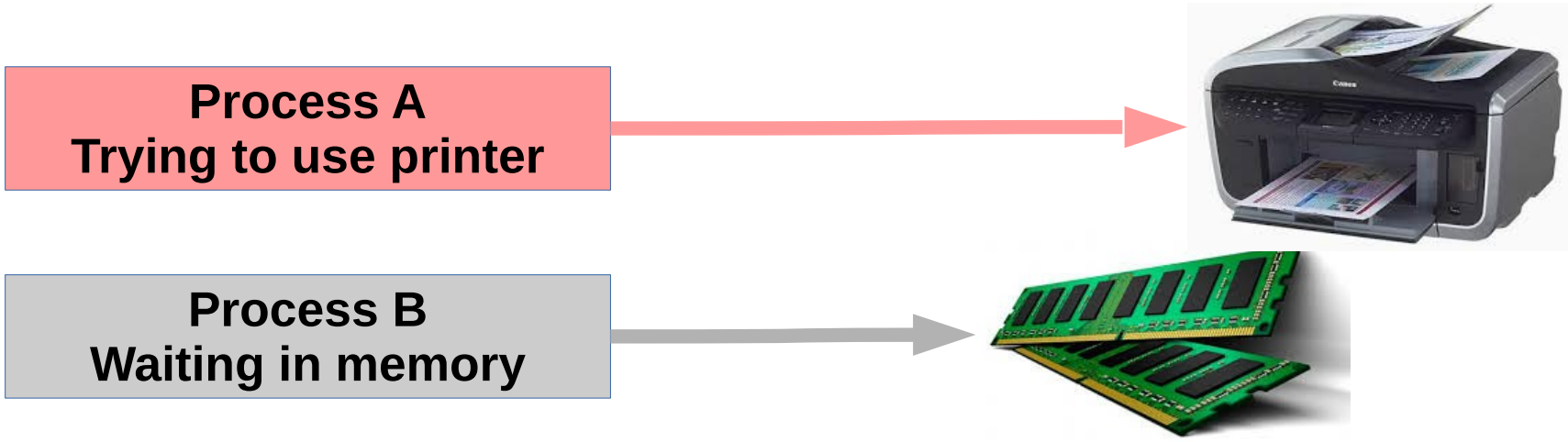


- Blue cannot go, blocked by Red
- Red cannot go, blocked by SkyBlue
- SkyBlue cannot go, blocked by Pink
- Pink cannot go, blocked by NavyBlue

Two Types of System Resources

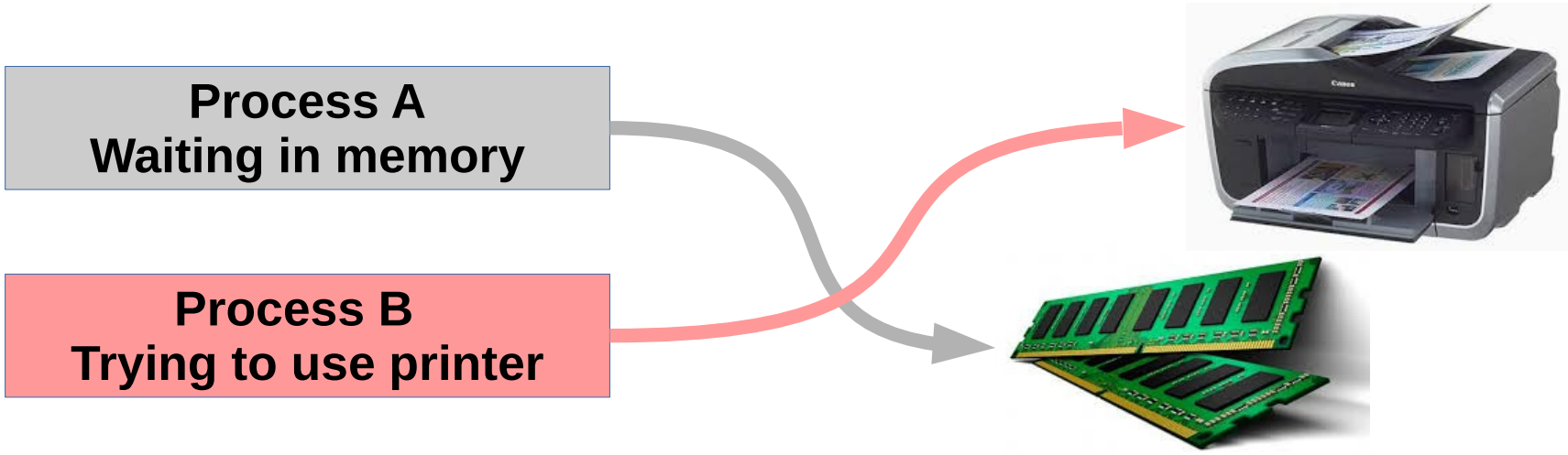
- **Preemptable** resources can be taken away from its current owner (and given back later) without (bad) consequences.
  - Memory: allocated by discretion of the OS
- **Non-preemptable** resources ***cannot*** be taken away.
  - Printers: when removed from system, the task cannot be completed
- A non-preemptable unit of work can be interrupted, but must receive control after the interrupt is processed.
- If it is interrupted, control returns to the operating system when the interrupt handling completes, the device must be allowed to complete.

Preemptable Resources and Deadlock



- Process A requests and is granted use of the printer
  - Begins to print but is interrupted and stopped before its completion (due to taking up too much time to complete the printing task)

Preemptable Resources and Deadlock



- Process B is run and tries unsuccessfully to use the printer (which is held by process A)
- **Can process B use this resource?**
- **What will happen to its printing task?**

ALLEGHENY COLLEGE

Preemptable Resources and Deadlock

Process A **Waiting in memory**	**Process A** **Trying to use printer**
Process B **Trying to use printer**	**Process B** **Waiting in memory**

- Process A has locked the printer: waiting for Process B.
- Process B has locked the memory: waiting for process A
- To break deadlock: each processes needs to give up its resources to the other
- Something needs to step in to make this happen

ALLEGHENY COLLEGE

Non-Preemptable Resources and Deadlock

- **Non-preemptable** resources ***cannot*** be taken away.
- Another example:
  - Burning data to a Blu-ray disk
  - If the Blu-ray burner is interrupted and paused,
  - Then the burning task will fail.
  - Blu-ray recorders are not pre-emptable at an arbitrary moment.

ALLEGHENY COLLEGE

Deadlocks and Non-Preemptive Resources

- In general, deadlocks involve nonpreemptable resources.
- Potential deadlocks that involve preemptable resources can usually be resolved by reallocating resources from one process to another.
- If the resource is not available when requested, the requesting process will wait.

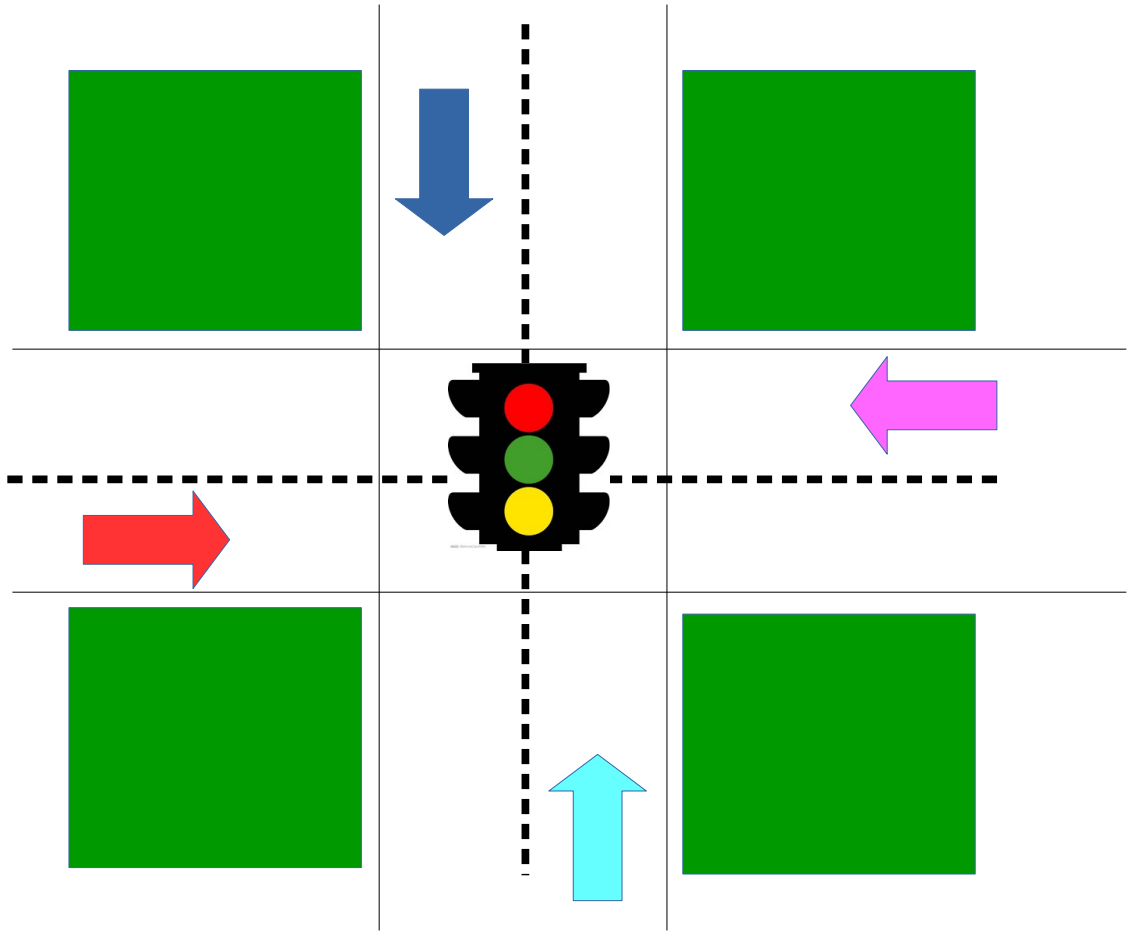
ALLEGHENY COLLEGE

Deadlocks and Non-Preemptive Resources

- The abstract sequence of events required to use a resource is given below.
  - 1. Request the resource.
  - 2. Use the resource.
  - 3. Release the resource.
- In some operating systems, the process is automatically blocked when a resource request fails, and is awakened when it becomes available.
- In other systems, the request fails with an error code, and it is up to the calling process to wait for some time and to try again.
  - Waiting is being *blocked*

How to Avoid Deadlocks?

- Semaphores used to control user-processes which can be interrupted
- Each resource observes rules to go at particular times.



How are traffic lights able to prevent deadlocks when people drive cars?

ALLEGHENY COLLEGE

Pseudocode for Semaphores

- Two semaphores to protect resource usage

```
typedef int semaphore;
semaphore resource_1;


void process_A(void) {
        down(&resource_1);
        use_resource_1( );
        up(&resource_1);
}
```

(a)

```
typedef int semaphore;
semaphore resource_1;
semaphore resource_2;

void process_A(void) {
        down(&resource_1);
        down(&resource_2);
        use_both_resources( );
        up(&resource_2);
        up(&resource_1);
}
```

(b)

Figure 6-1. Using a semaphore to protect resources. (a) One resource. (b) Two resources.