

CSC 256/456: Operating Systems

Deadlock

John Criswell
University of Rochester



Outline

- ❖ Basics of deadlock
- ❖ Three approaches for handling deadlock

The Deadlock Problem

- ❖ Definition:

- ❖ A set of blocked processes each holding some resources and waiting to acquire a resource held by another process in the set
- ❖ None of the processes can proceed or back-off (release resources it owns)

- ❖ Examples:

- ❖ Dining philosopher problem
- ❖ System has 2 memory pages (unit of memory allocation); P1 and P2 each hold one page and each needs another one
- ❖ Semaphores A and B, initialized to 1

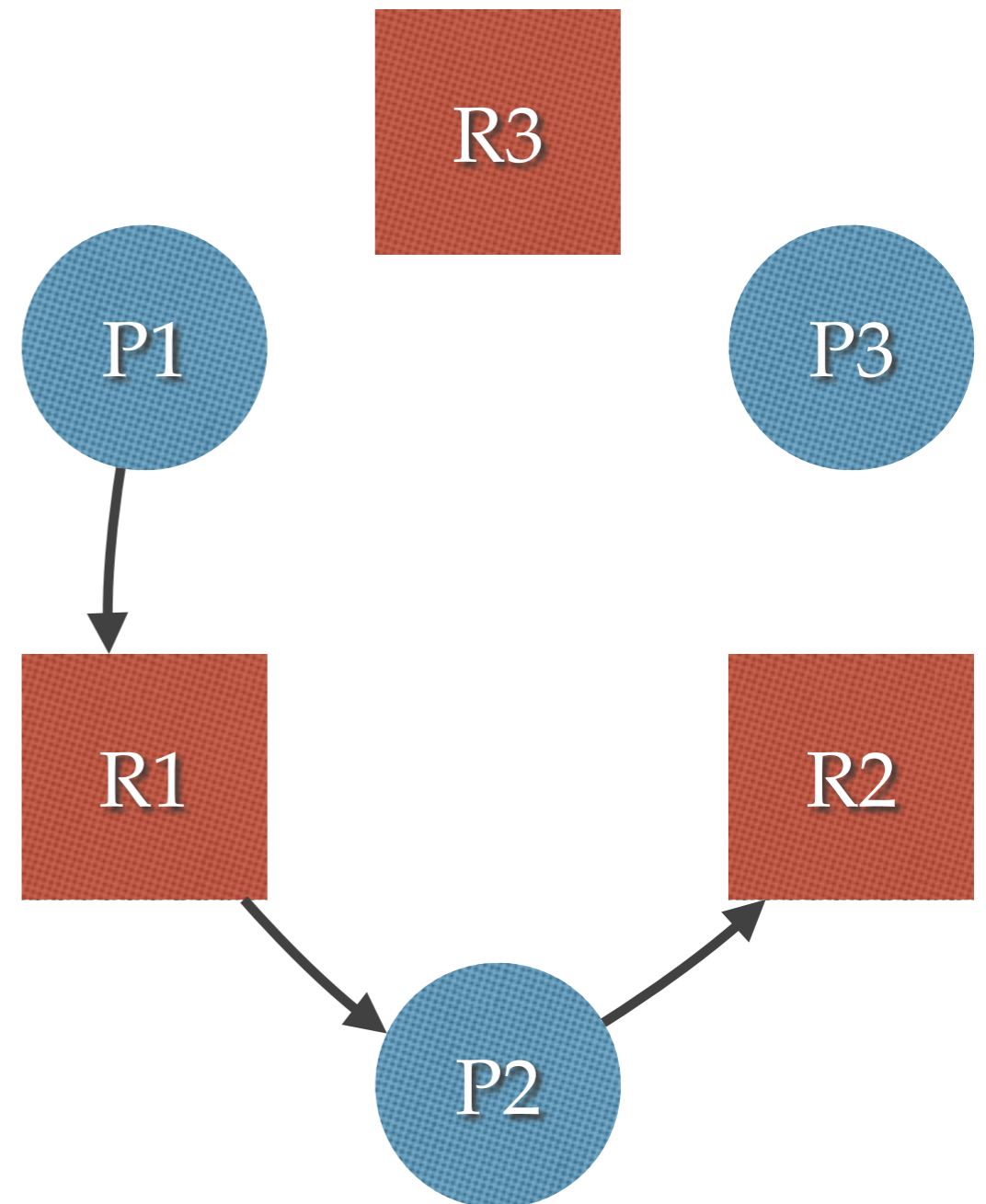
- | | | |
|---|-----------|---------|
| ❖ | P1 | P2 |
| ❖ | wait (A); | wait(B) |
| ❖ | wait (B); | wait(A) |

Resources Can Be Anything!

- ❖ Devices!
- ❖ Files!
- ❖ Networks!
- ❖ Mutexs!
- ❖ Sections of Road!
- ❖ Time itself!

Resource Allocation Graph

- ❖ Nodes
 - ❖ Processes
 - ❖ Resources (or resource classes)
- ❖ Request Edge
 - ❖ $P \rightarrow R$: Process Wants Resource
- ❖ Assignment Edge
 - ❖ $R \rightarrow P$: Resource held by process



The Four Conditions of Deadlock

- ❖ Mutual exclusion
- ❖ Hold and wait
- ❖ No preemption
- ❖ Circular wait

Mutual Exclusion

- ❖ Only one process at a time can use a resource

Hold and Wait

- ❖ Process can hold one resource while waiting to acquire additional resources held by other processes

No Preemption

- ❖ Resource can be released only voluntarily by the process holding it, after that process has completed its task

Circular Wait

- ❖ There exists a set $\{P_0, P_1, \dots, P_n\}$ of waiting processes such that:
 - ❖ P_0 is waiting for a resource that is held by P_1 ,
 - ❖ P_1 is waiting for a resource that is held by P_2 ,
 - ❖ \dots ,
 - ❖ P_{n-1} is waiting for a resource that is held by P_n ,
 - ❖ and P_n is waiting for a resource that is held by P_0 .

Methods for Handling Deadlocks

- ❖ Ignore the problem!
- ❖ Ensure that the system will never enter a deadlock state
 - ❖ Deadlock Prevention
 - ❖ Deadlock Avoidance
- ❖ Detect and recover from deadlock state

Of Ostriches and Resources

The “Ostrich” Algorithm

- ❖ Pretend there is no problem
 - ❖ unfortunately they can occur
- ❖ Reasonable if
 - ❖ deadlocks occur very rarely
 - ❖ cost of prevention is high
- ❖ Typical OS take this approach for processes and devices
- ❖ Trade-off between convenience and correctness

Deadlock Prevention

Deadlock Prevention

- ❖ Establish conventions that prevent deadlock
- ❖ Prevents one of the four conditions from happening

Attack the Mutual Exclusion Condition!

- ❖ Don't permit mutual exclusion
 - ❖ Read-only files
 - ❖ Spooled devices (e.g., printer)
- ❖ Can't be done in most cases

Attack the Hold and Wait Condition!

- ❖ Require processes to request all resources before starting
- ❖ Problems
 - ❖ May not know required resources at start of run
 - ❖ Ties up resources other processes could be using
- ❖ Variation:
 - ❖ before a process requests a new resource, it must give up all resources and then request all resources needed

Attack the No Preemption Condition!

- ❖ Preemption

- ❖ when a process is holding some resources and waiting for others, its resources may be preempted to be used by others

- ❖ Problem

- ❖ Many resources may not allow preemption; i.e., preemption will cause process to fail

Attack the Circular Wait Condition!

- ❖ Impose a total order of all resource types
- ❖ Processes *must* request resources in the same order
- ❖ Often used for mutexes

Deadlock Avoidance

Prevention vs. Avoidance

- ❖ Prevention prevents conditions under which deadlock can occur
 - ❖ Enforces conventions that break deadlock conditions
- ❖ Avoidance refuses resource allocation if deadlock would be possible in the future
 - ❖ Examines state of allocation to check for future deadlock

Deadlock Avoidance

- ❖ Processes declare max amount of resources ever needed
- ❖ How to determine if allocation state is deadlock free:
 - ❖ Line up processes in some order such that for all P_i :
 - ❖ Can allocate all future resources from either:
 - ❖ Free resources
 - ❖ Resources held by all P_j such that $j < i$
- ❖ State of system is called a safe state

Deadlock Avoidance Example 1

Resources Unallocated: 1

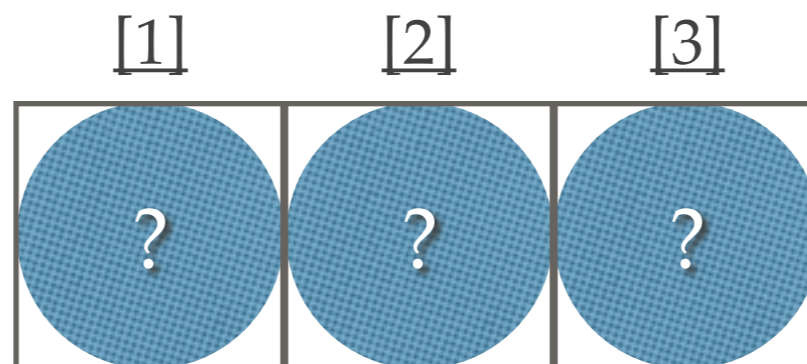
Holds: 5
Needs: 1



Holds: 7
Needs: 2



Holds: 2
Needs: 7



Deadlock Avoidance Example 1

Resources Unallocated: 1

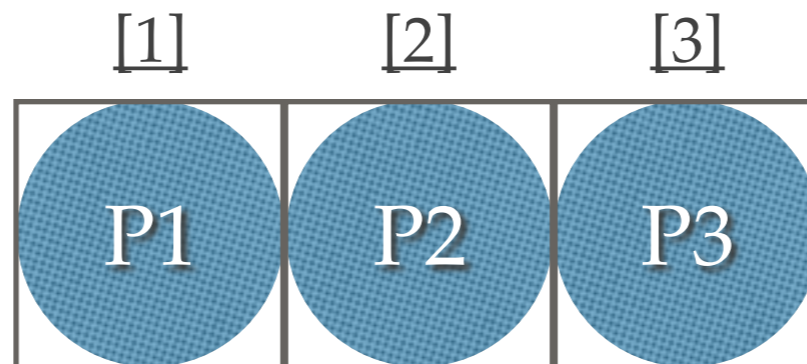
Holds: 5
Needs: 1



Holds: 7
Needs: 2



Holds: 2
Needs: 7



Deadlock Avoidance Example 2

Resources Unallocated: 1

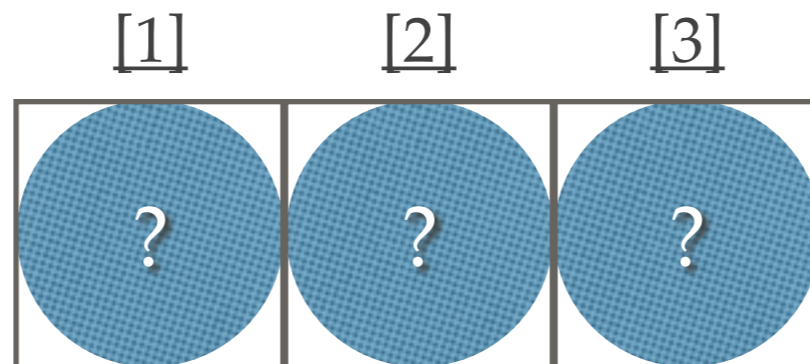
Holds: 1
Needs: 1



Holds: 1
Needs: 4



Holds: 1
Needs: 7



One Resource Per Resource Class

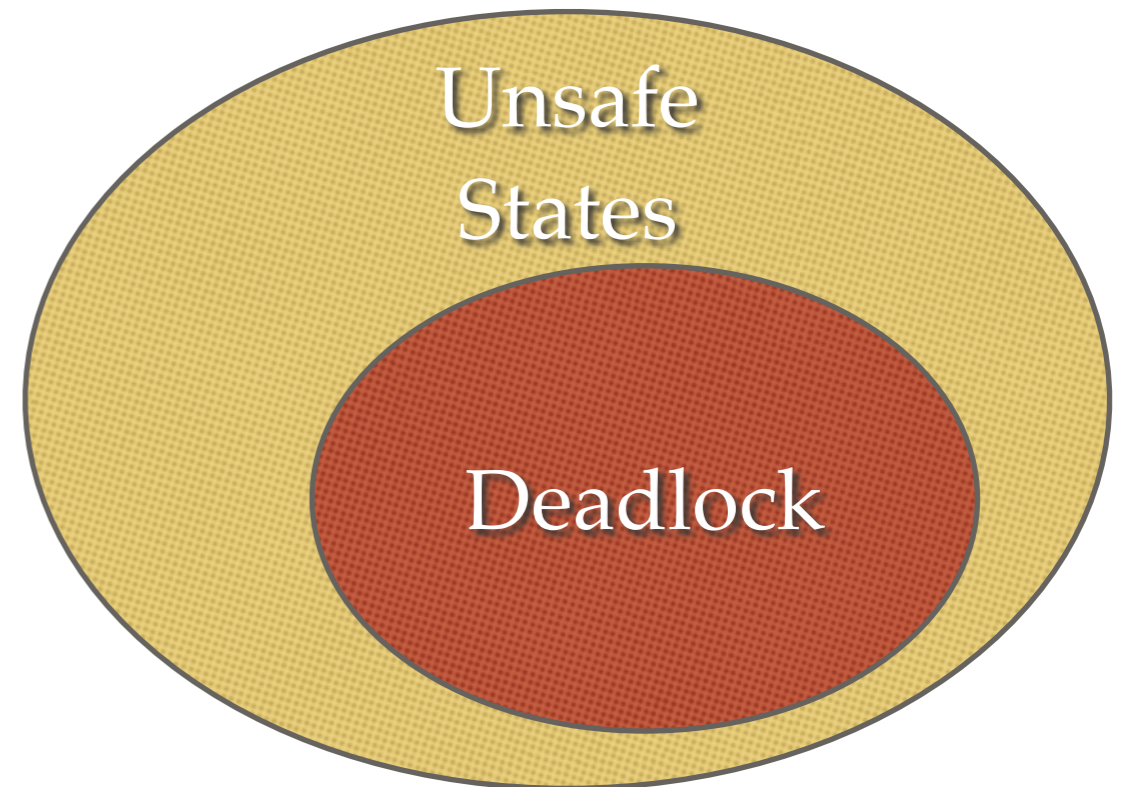
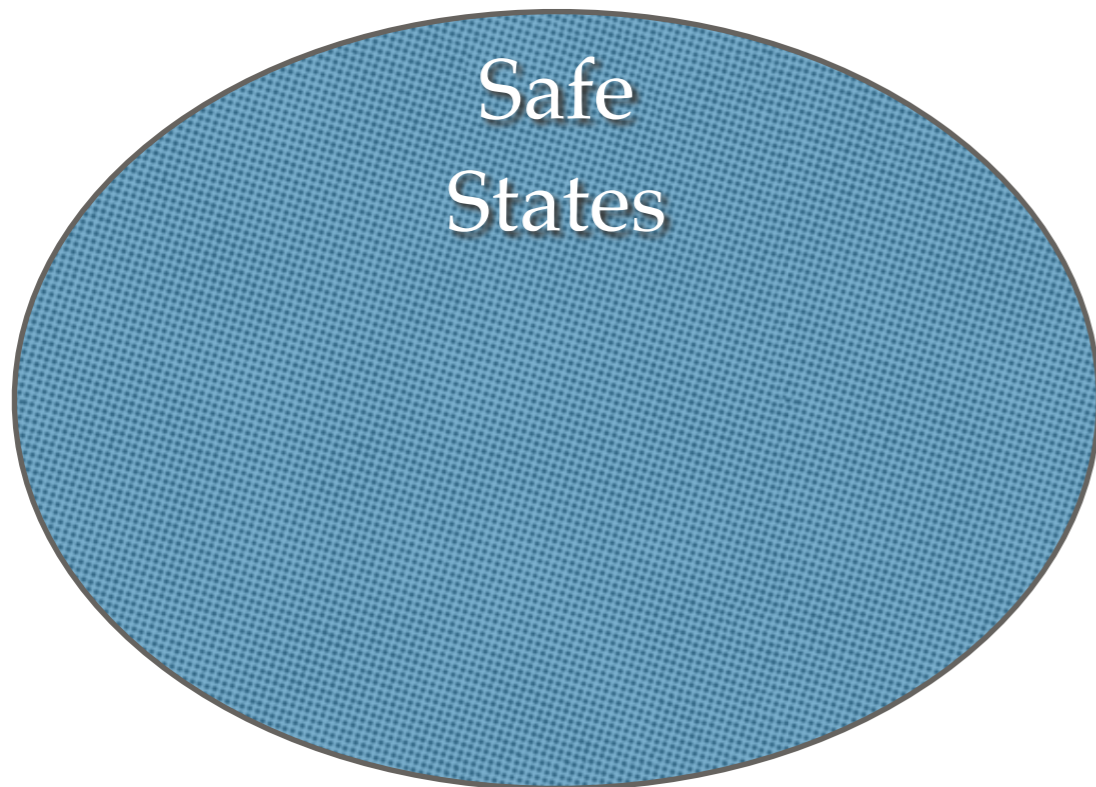


Safe States



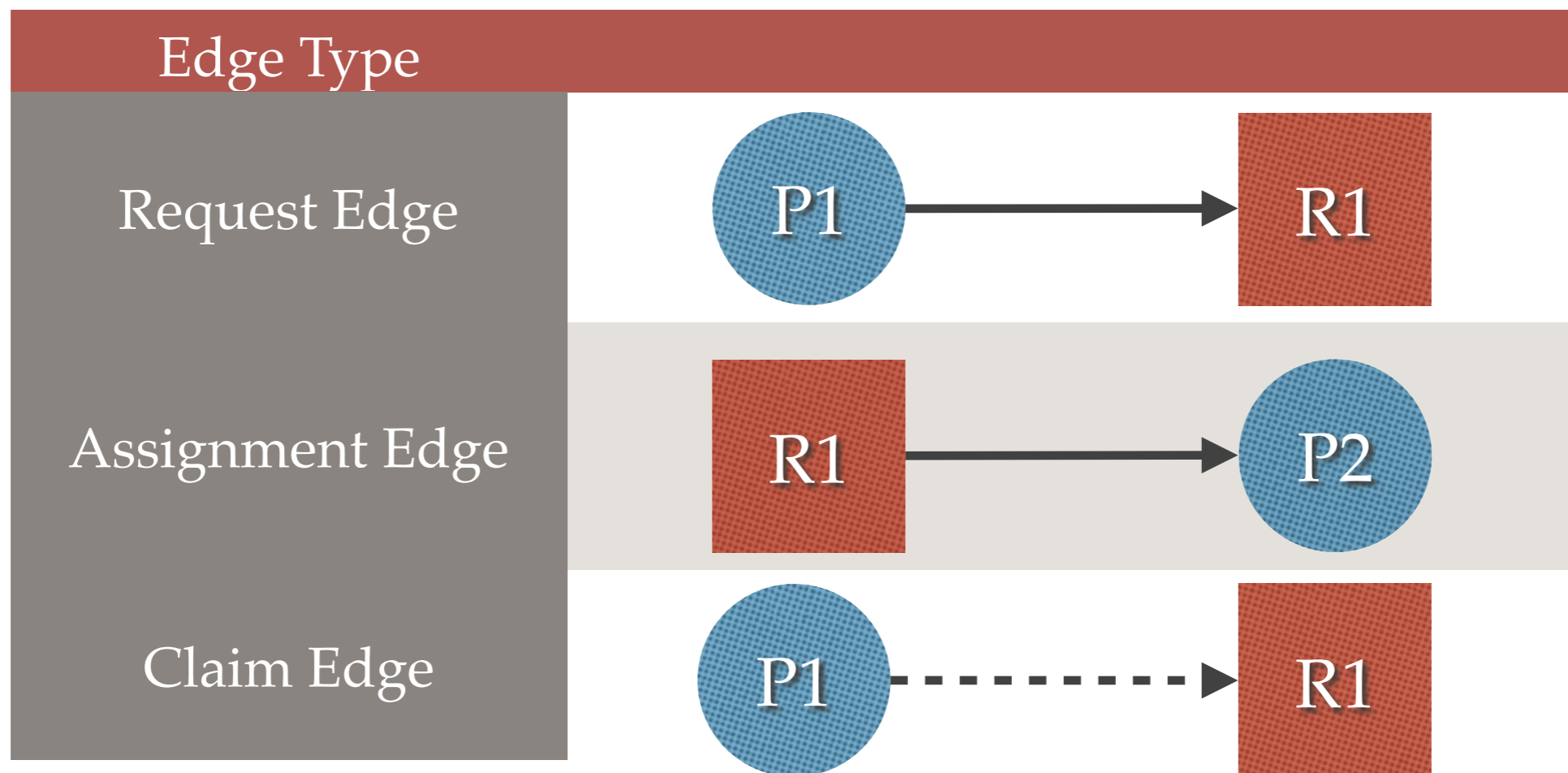
Unsafe States == Deadlock

Multiples Resources Per Resource Class



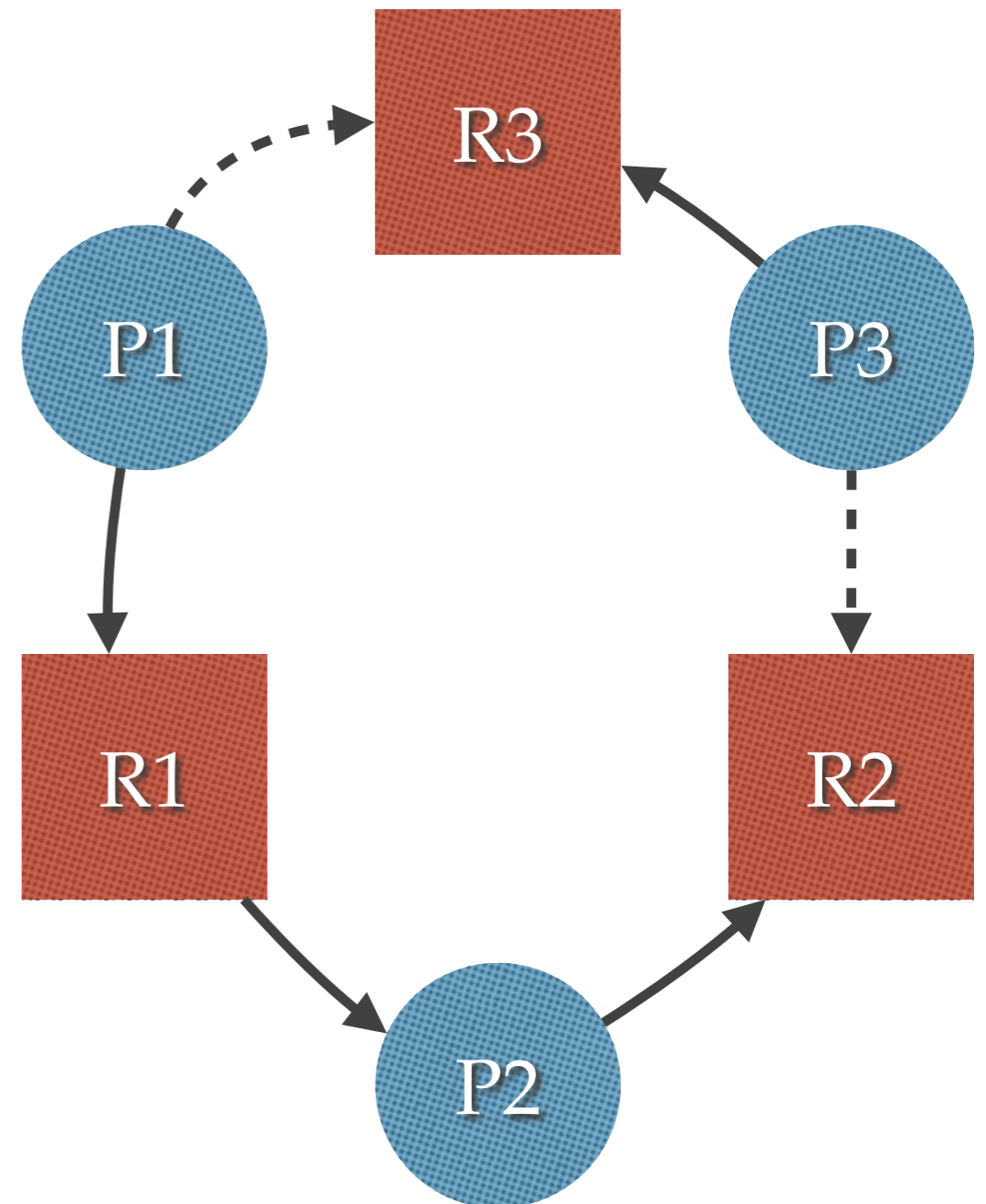
Deadlock Prevention with Resource Allocation Graphs

- ❖ Works with single resource per resource class
- ❖ Enhance allocation graph with three edges



Deadlock Prevention with Resource Allocation Graphs

- ❖ On allocation:
 - ❖ Change claim to assignment edge
 - ❖ Check for cycle
 - ❖ Cycle implies deadlock



What is the execution time for cycle detection?

Banker's Algorithm

- ❖ Each process must a priori claim the maximum set of resources that might be needed in its execution
- ❖ Safety check
 - ❖ repeat
 - ❖ pick any process that can finish with existing available resources; finish it and release all its resources
 - ❖ until no such process exists
 - ❖ all finished \rightarrow safe; otherwise \rightarrow unsafe.
- ❖ When a resource request is made, the process must wait if:
 - ❖ enough available resource is not available for this request
 - ❖ granting the request would result in an unsafe system state

Deadlock Detection

- ❖ Deadlock detection is very similar to the safety check in the Banker's algorithm
- ❖ Replace maximum needs with current requests

Recovery from Deadlock

- ❖ Recovery through preemption
 - ❖ take a resource from some other process
 - ❖ depends on nature of the resource
- ❖ Recovery through rollback
 - ❖ checkpoint a process state periodically
 - ❖ rollback a process to its checkpoint state if it is found deadlocked
- ❖ Recovery through killing processes
 - ❖ kill one or more of the processes in the deadlock cycle
 - ❖ the other processes get its resources
 - ❖ In which order should we choose process to kill?
- ❖ Will recovery lead to starvation?

Disclaimer

- ❖ Parts of the lecture slides contain original work of Abraham Silberschatz, Peter B. Galvin, Greg Gagne, Andrew S. Tanenbaum, and Gary Nutt. The slides are intended for the sole purpose of instruction of operating systems at the University of Rochester. All copyrighted materials belong to their original owner(s).