COMP-SCI-431
Intro Operating Systems

Lecture 5 – Deadlock

Adu Baffour, PhD

University of Missouri-Kansas City
Division of Computing, Analytics, and Mathematics
School of Science and Engineering
aabnxq@umkc.edu

Lecture Objectives

- Understand the concept of a system model for deadlocks in computer systems.
- Explore the techniques and methods used for deadlock detection within a system.
- Analyze the principles of dynamic deadlock avoidance strategies like Banker's Algorithm.
- Examine the mechanisms and principles of static deadlock prevention in system design.

Outline

5.1 A system model for deadlocks

5.2 Deadlock detection

5.3 Static deadlock prevention

5.4 Dynamic deadlock avoidance

Outline

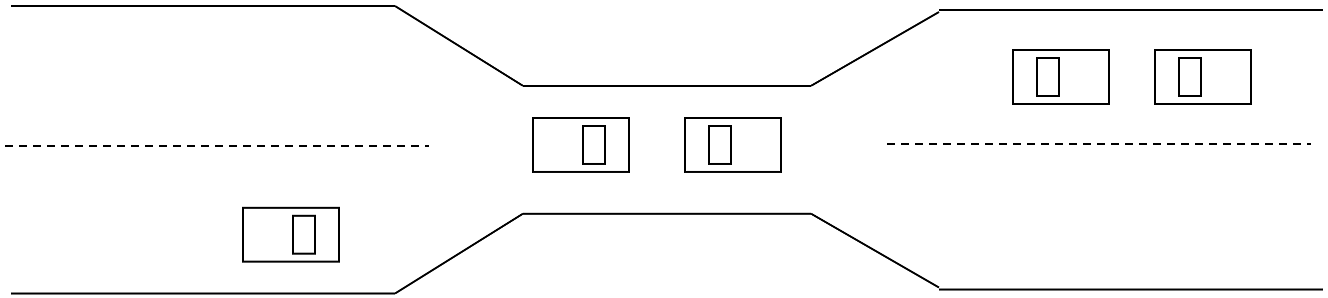
5.1 A system model for deadlocks

5.2 Deadlock detection

5.3 Static deadlock prevention

5.4 Dynamic deadlock avoidance

5.1 A system model for deadlocks



- Traffic in one direction
- Each section of a bridge can be viewed as a resource
- If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback)
- Several cars may have to be backed up if a deadlock occurs
- Starvation is possible
- Note – Most OSes do not prevent or deal with deadlocks

5.1 A system model for deadlocks

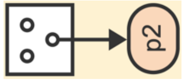
Deadlock can arise if four conditions hold simultaneously

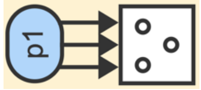
- **Mutual exclusion:** only one process at a time can use a resource
- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes
- **No preemption:** a resource can be released only voluntarily by the process holding it after that process has completed its task
- **Circular wait:** there exists a set $\{P_0, P_1, \ldots, P_n\}$ 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, …, 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

5.1 A system model for deadlocks

Resource allocation graphs

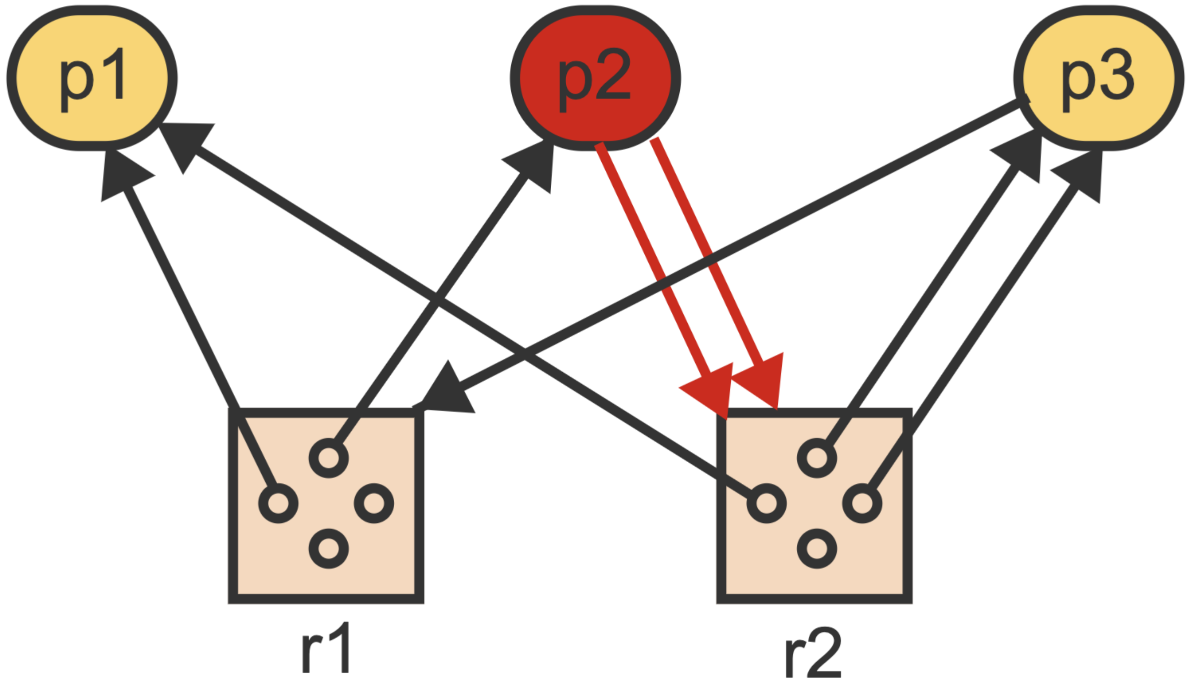
- A ***resource allocation graph*** shows the current allocation of resources to processes and the current requests by processes for new resources.
- Circles represent processes. 

- Rectangles represent resources. 
- If a resource contains multiple units, then each unit is represented by a small circle.
- Resource allocations are represented by edges directed from a resource to a process. 

- Resource requests are represented by edges directed from a process to a resource. 

5.1 A system model for deadlocks

- A process p is ***blocked*** on a resource r if one or more request edges directed from p to r exist, and r does not contain sufficient free units to satisfy all requests.



5.1 A system model for deadlocks

Modeling Deadlocks

- A resource r contains one or more identical units, each of which may be requested and used by a process on a non-shared basis.
- A deadlock involves at least 2 processes and 2 resources (or resource units), where each process holds one resource and is blocked indefinitely on another resource held by another process.

Outline

5.1 A system model for deadlocks

5.2 Deadlock detection

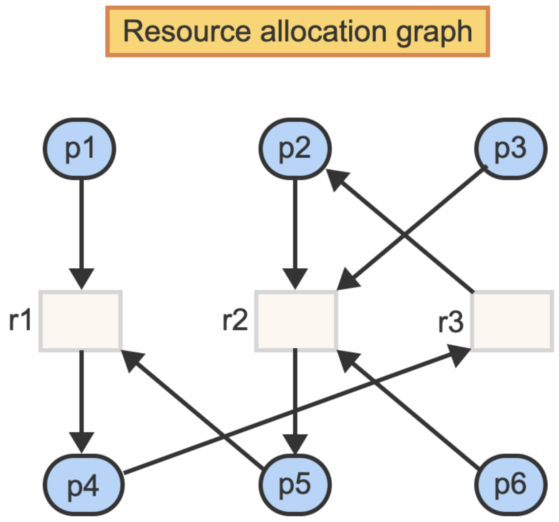
5.3 Static deadlock prevention

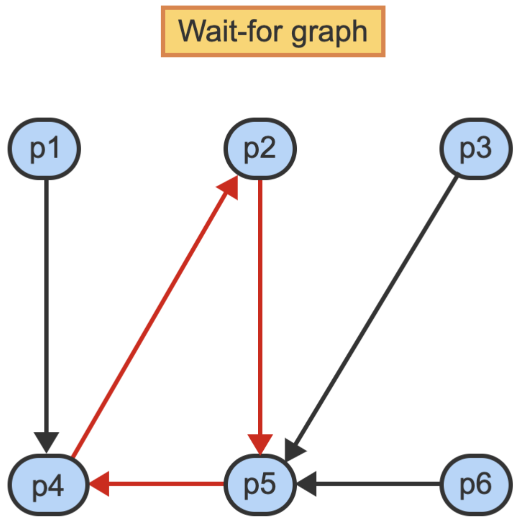
5.3 Dynamic deadlock avoidance

5.2 Deadlock detection

Deadlock detection with single-unit resources

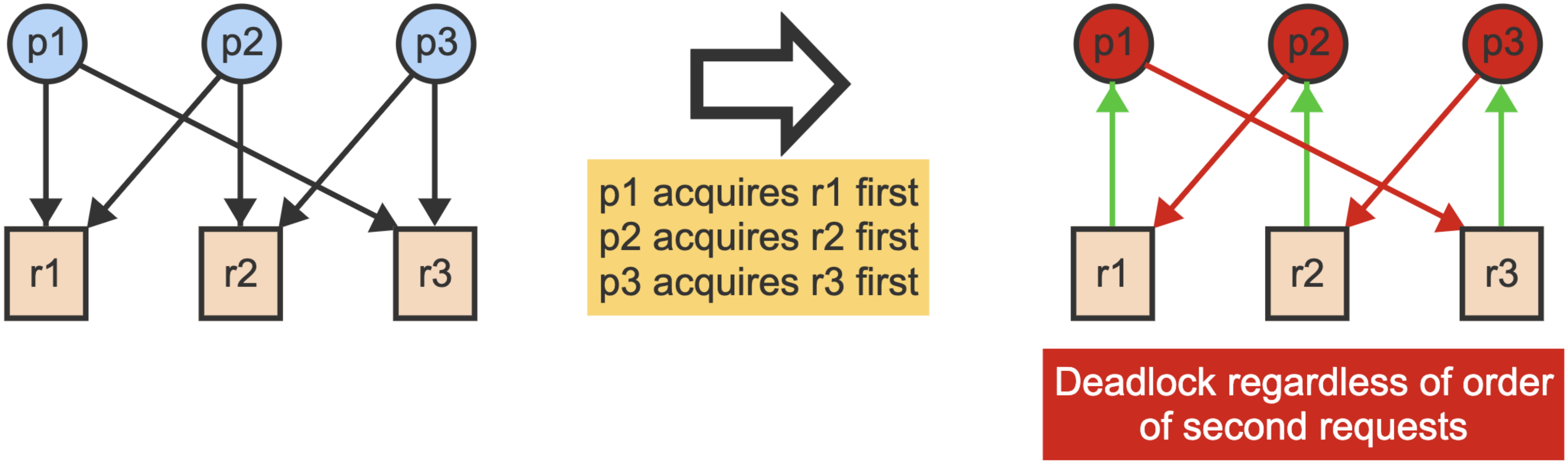
- With a single-unit resource r, only a single allocation edge can exist between r and a process p.
- A ***wait-for graph*** is a resource allocation graph containing only processes where each process can have multiple incoming resource allocation edges but only one outgoing resource request edge.





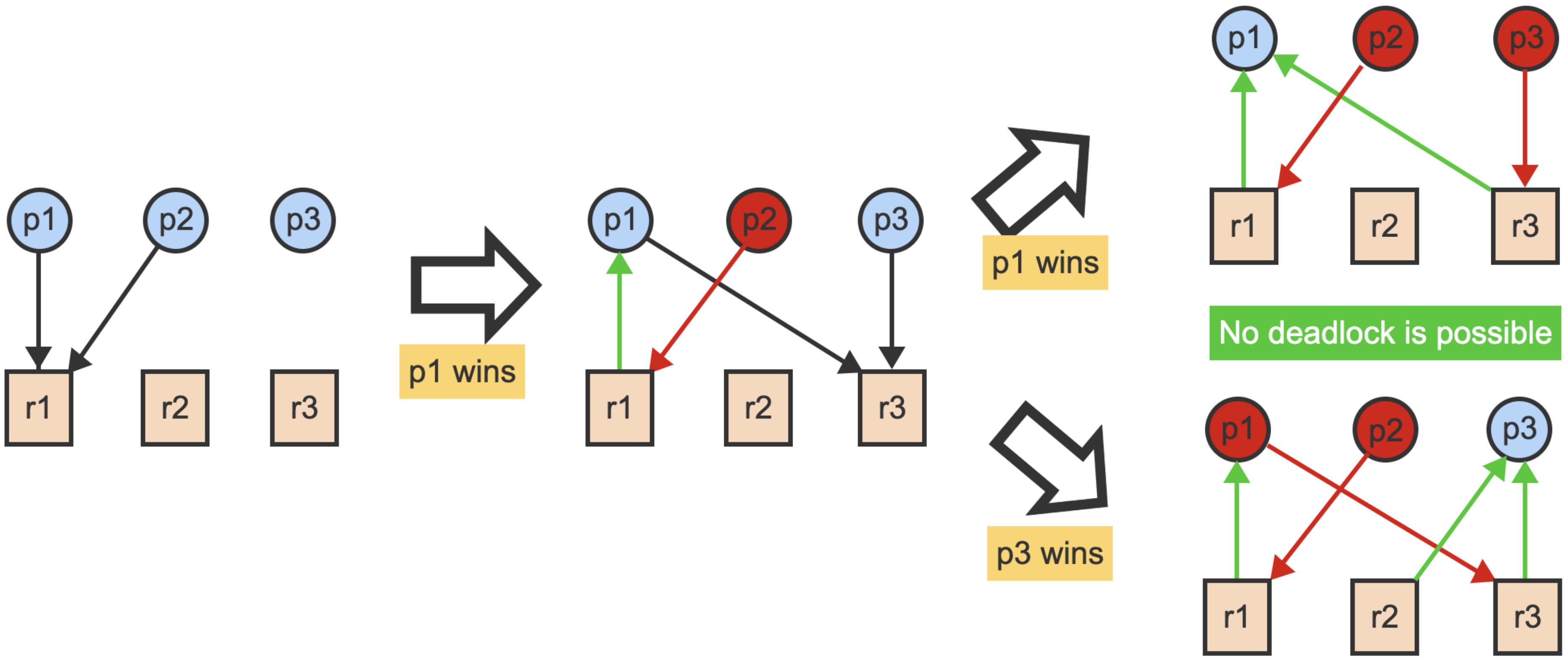
5.2 Deadlock detection

Possible deadlock with 3 processes and 3 single-unit resources

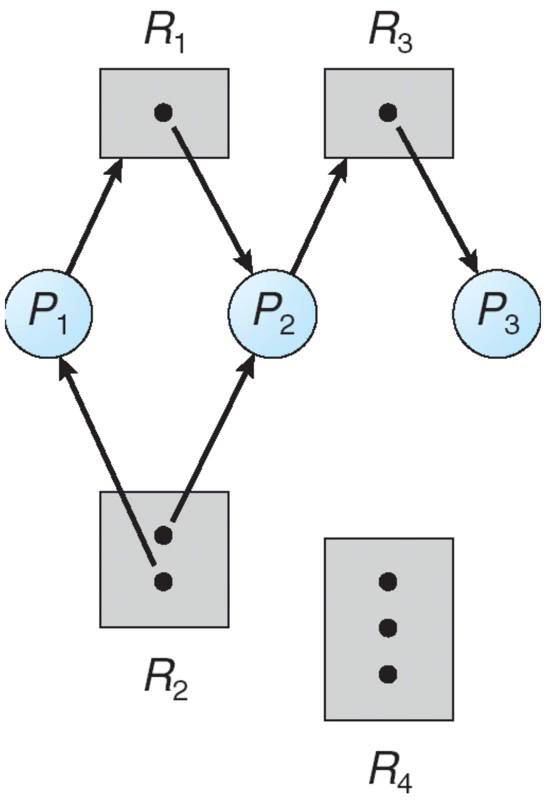


5.2 Deadlock detection

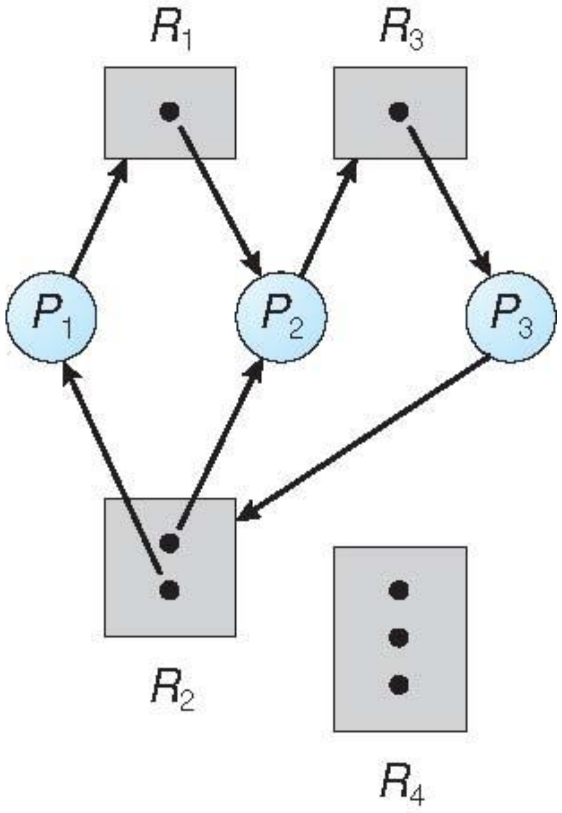
Possible deadlock with 3 processes and 3 single-unit resources



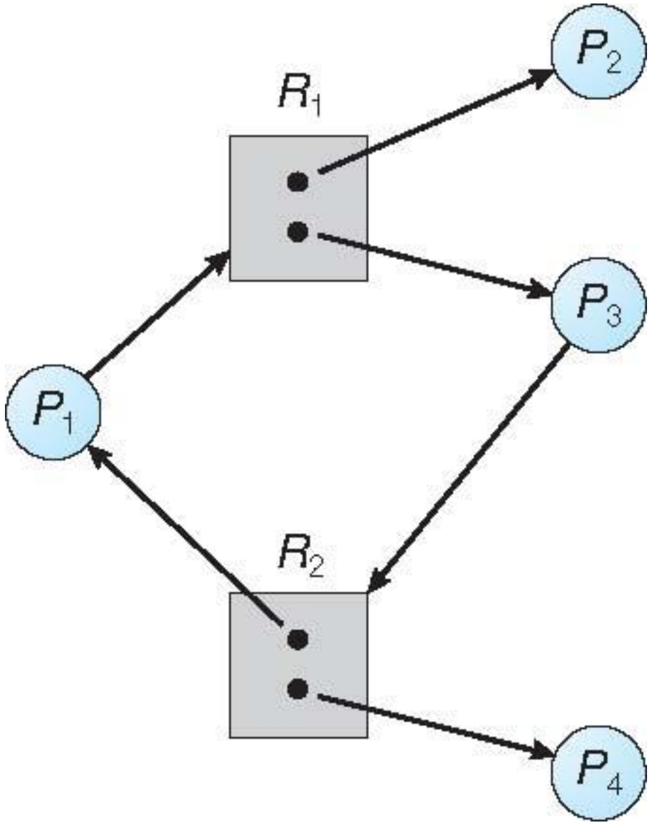
5.2 Deadlock detection



No deadlock



Deadlock



No deadlock

5.2 Deadlock detection

Basic facts

- If the graph contains no cycles ⇒ no deadlock
- If the graph contains a cycle ⇒
  - if only one instance per resource type, then deadlock
  - if there are several units per resource type, there is a possibility of deadlock

- How do we deal with deadlocks?
  - Ensure that the system will *never* enter a deadlock state.
  - Allow the system to enter a deadlock state and then recover.
  - Ignore the problem and pretend that deadlocks never occur in the system, which is used by most operating systems.

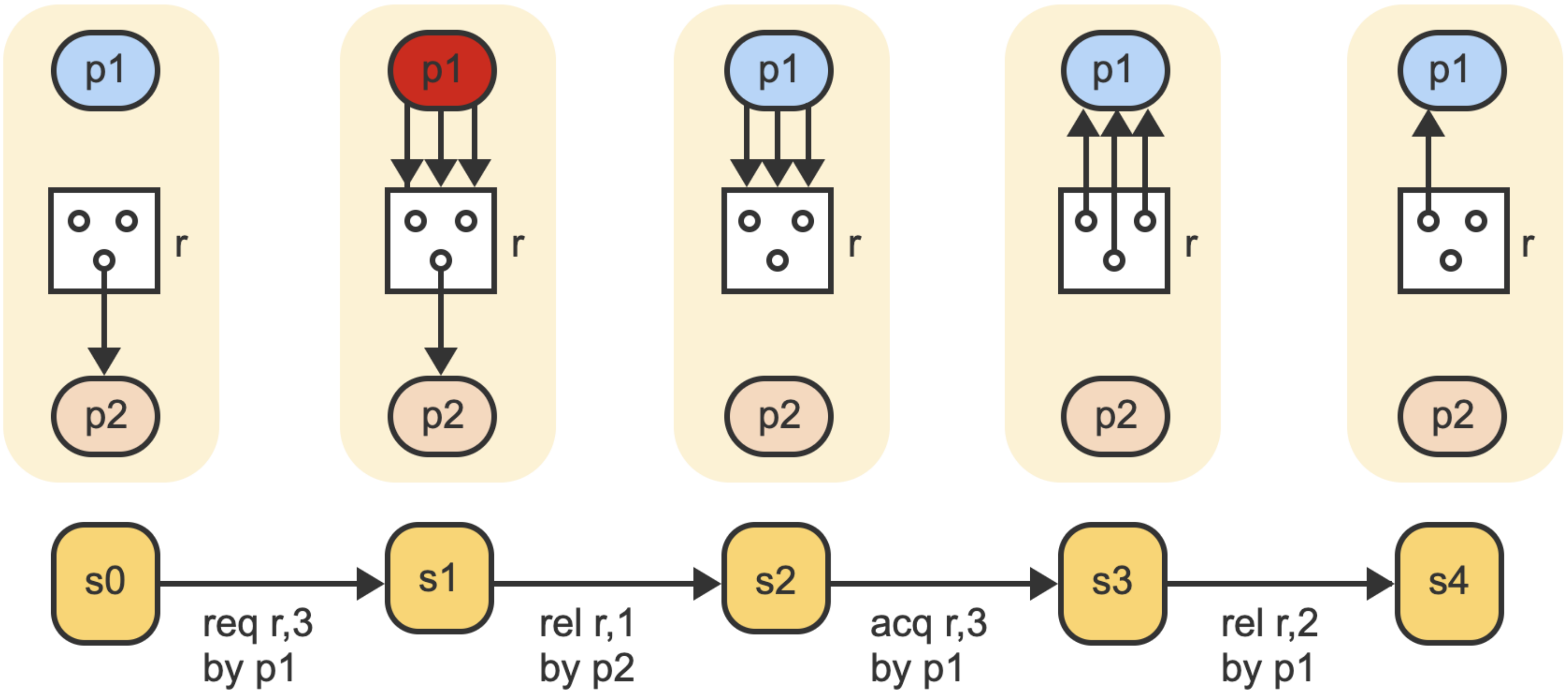
5.2 Deadlock detection

State transitions

- A ***resource request*** (req r, m) by a process p for m units of a resource r creates m new edges directed from p to r.
- A ***resource acquisition*** (acq r, m) by a process p of m units of a resource r reverses the direction of the corresponding request edges to point from the units of r to p.
- A ***resource release*** (rel r, m) operation by a process p of m units of a resource r deletes m allocation edges between p and r.

5.2 Deadlock detection

A sequence of possible state transitions



5.2 Deadlock detection

Deadlock states and safe states

- A process is ***deadlocked*** in a state s if the process is blocked in s, and if no matter what state transitions occur in the future, the process remains blocked.
- A state s is called a ***deadlock state*** if s contains two or more deadlocked processes.
- A state s is a ***safe state*** if no sequence of state transitions exists that would lead from s to a deadlock state.

Outline

5.1 A system model for deadlocks

5.2 Deadlock detection

5.3 Static deadlock prevention

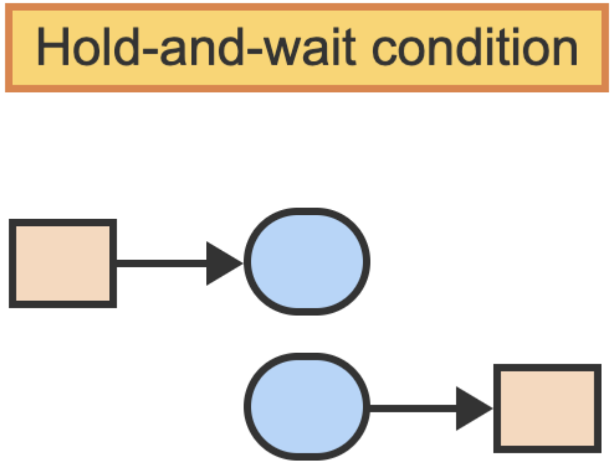
5.4 Dynamic deadlock avoidance

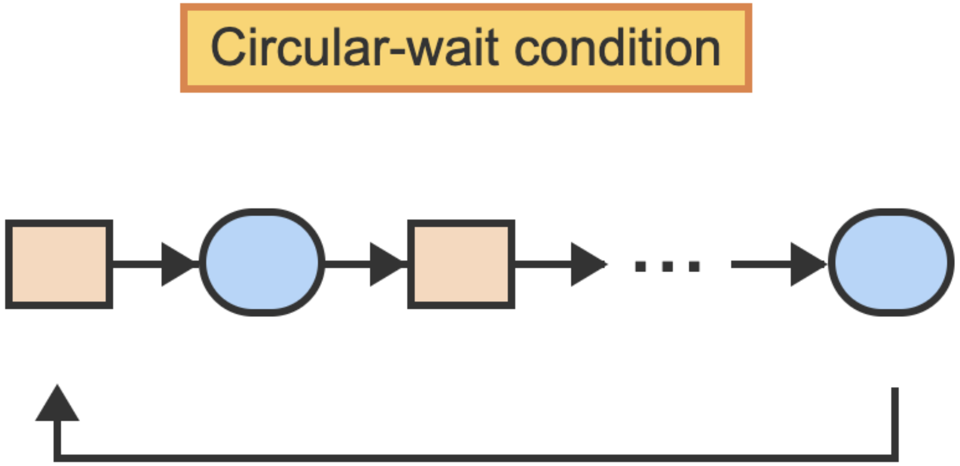
5.3 Static deadlock prevention

Conditions for a deadlock

- Four conditions must hold for a deadlock to occur with reusable resources:

1. Mutual Exclusion
2. Hold and wait
3. No preemption
4. Circular wait





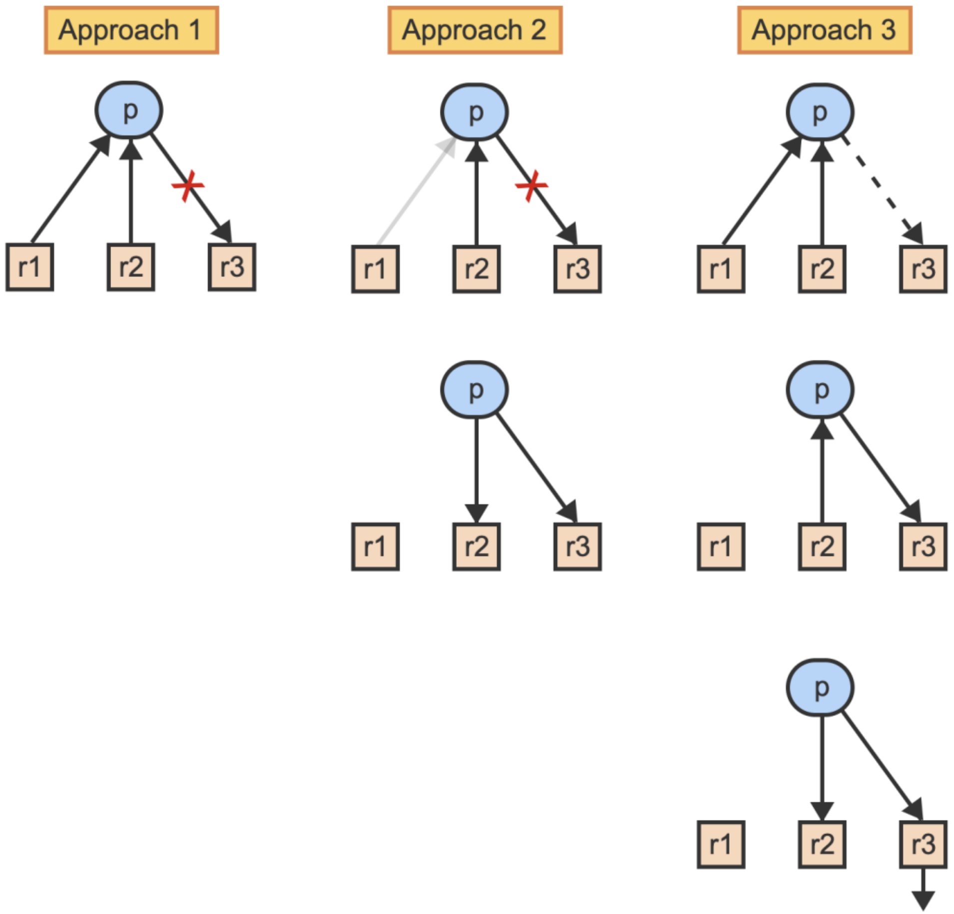
5.3 Static deadlock prevention

Eliminating hold-and-wait

1. Every process must request all resources ever needed at the same time.
2. Every process must release all currently held resources before making any new request.
3. A process can be given the ability to test whether a needed resource is currently available. The process must release all currently held resources if the requested resource is unavailable. Otherwise, the new resource may be requested and allocated immediately.

5.3 Static deadlock prevention

Eliminating hold-and-wait



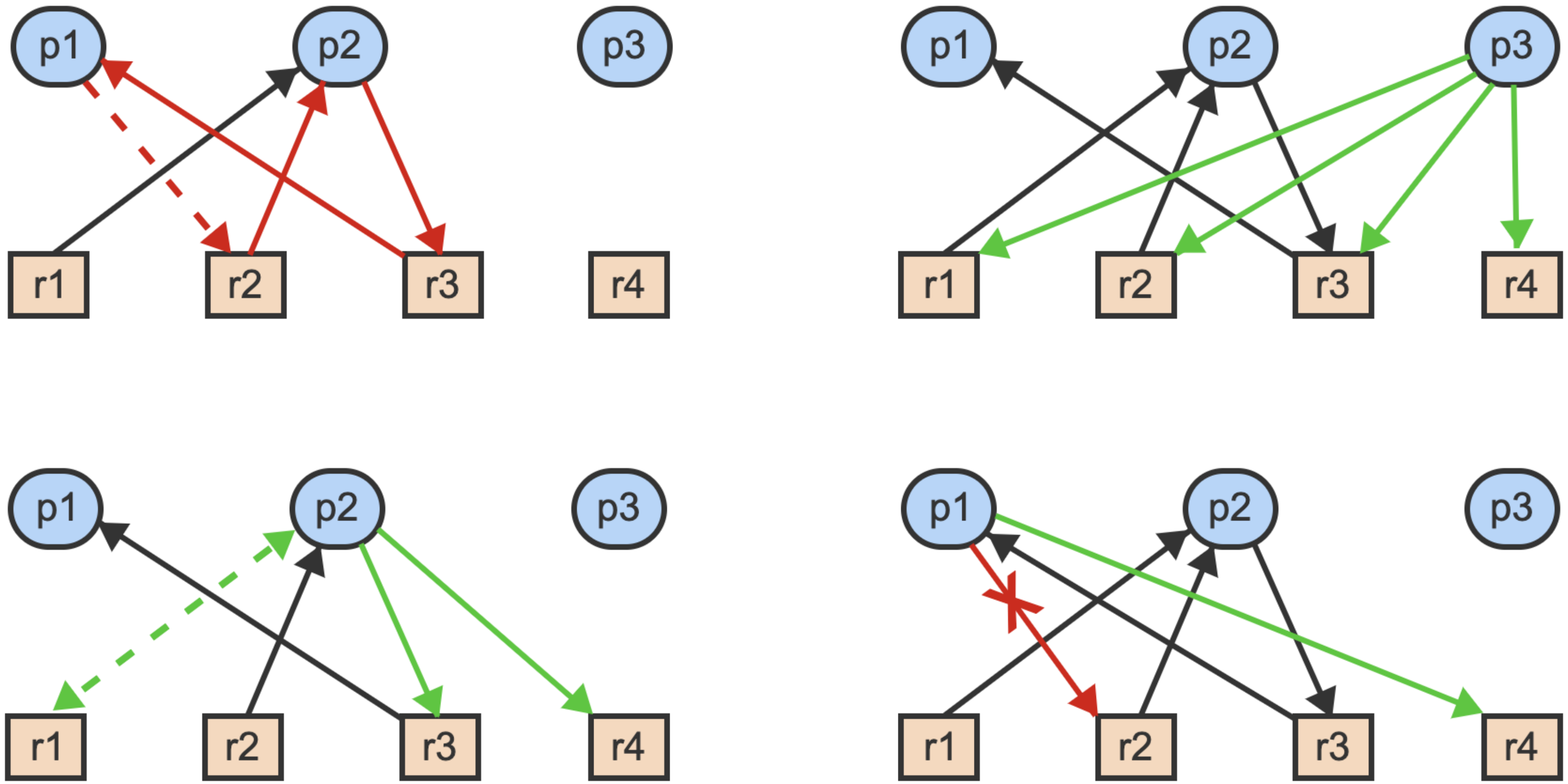
5.3 Static deadlock prevention

Eliminating circular wait

- A circular wait is prevented if all processes must request all resources in the same order.
- One approach is to assign a sequential ordering, seq, to all existing resources, such that $seq(r_i) \neq seq(r_j)$ for all $i \neq j$.
- All processes are then required to request resources in only increasing sequential order.
- A process p already holding a resource ri with the sequence number $seq(r_i)$ may only request resource rj where $seq(r_i) < seq(r_j)$.

5.3 Static deadlock prevention

An ordered resources policy



Outline

5.1 A system model for deadlocks

5.2 Deadlock detection

5.3 Static deadlock prevention

5.4 Dynamic deadlock avoidance

5.4 Dynamic deadlock avoidance

Basic facts

- If a system is in a safe state $\Rightarrow$ no deadlocks
- If a system is in an unsafe state $\Rightarrow$ possibility of deadlock
- Avoidance $\Rightarrow$ ensures that a system will never enter an unsafe state.

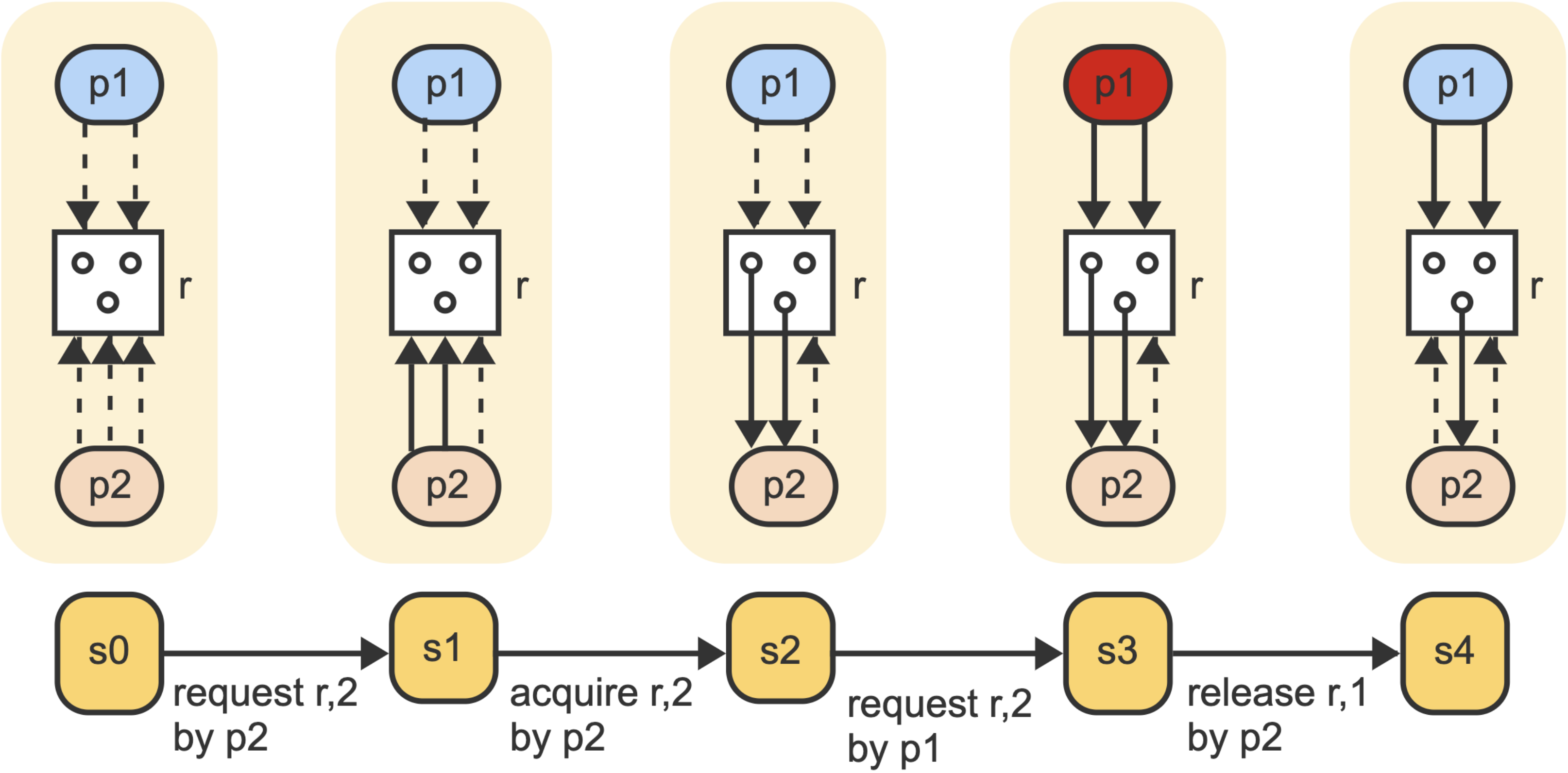
5.4 Dynamic deadlock avoidance

Resource claim graphs

- The ***maximum claim*** of a process is the set of all resources the process may ever request.
- A ***resource claim graph*** is an extension of the general resource allocation graph. The extended graph shows
  - the current allocation of resources to processes and
  - all current and potential future requests by processes for new resources.
- A potential request edge may eventually be transformed into an actual request edge and a resource allocation edge.

5.4 Dynamic deadlock avoidance

A sequence of possible state transitions with a claim graph



5.4 Dynamic deadlock avoidance

The banker's algorithm

1. Given a resource request in a state s, temporarily grant the request by changing the request edges to allocation edges.
2. Execute the safety algorithm on the new state s’.
3. If the graph of state s' keeps the system in a safe state, then accept s’ as the new state. Otherwise, disallow the acquisition by reverting to state s.

5.4 Dynamic deadlock avoidance

The banker's algorithm

- Let n = number of processes, and m = number of resources types.
- **Available**: Vector of length m. If available $[j] = k$, there are k instances of resource type R_j available
- **Max**: n x m matrix. If $Max[i,j] = k$, then process P_i may request at most k instances of resource type R_j
- **Allocation**: n x m matrix. If Allocation$[i,j] = k$ then P_i is currently allocated k instances of R_j
- **Need**: n x m matrix. If $Need[i,j] = k$, then P_i may need k more instances of R_j to complete its task

  $$Need[i,j] = Max[i,j] - Allocation[i,j]$$

5.4 Dynamic deadlock avoidance

The Safety algorithm

1. Let *Work* and *Finish* be vectors of length *m* and *n*, respectively. Initialize:

    Work = *Available*

    Finish [*i*] = *false* for *i* = 0, 1, …, *n*- 1

2. Find *i* such that both:

    (a) *Finish* [*i*] = *false*

    (b) $Need_i \leq Work$

    If no such *i* exists, go to step 4

3. $Work = Work + Allocation_i$
    Finish[*i*] = *true*
    go to step 2

4. If *Finish* [*i*] == true for all *i*, then the system is in a safe state

5.4 Dynamic deadlock avoidance

Resource-Request Algorithm for *P_i*

Request = request vector for process P_i. If $Request_i[j] = k$ then process P_i wants k instances of resource type R_j

1. If $Request_i \leq Need_i$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim
2. If $Request_i \leq Available$, go to step 3. Otherwise, P_i must wait since resources are not available
3. Pretend to allocate requested resources to P_i by modifying the state as follows:

$$Available = Available - Request;$$

$$Allocation_i = Allocation_i + Request_i;$$

$$Need_i = Need_i - Request_i;$$

- *If safe $\Rightarrow$ the resources are allocated to Pi*
- *If unsafe $\Rightarrow$ Pi must wait, and the old resource-allocation state is restored*

5.4 Dynamic deadlock avoidance

Example 1

5 processes P_0 through P_4;

3 resource types:

A (10 instances), *B* (5instances), and *C* (7 instances)

Snapshot at time S_0:

	Allocation	*Max*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

5.4 Dynamic deadlock avoidance

Example 1

- The content of the matrix *Need* is defined to be *Max* – *Allocation*

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

- The system is in a safe state since the sequence < P_1, P_3, P_4, P_2, P_0> satisfies safety criteria

5.4 Dynamic deadlock avoidance

Example 2

5 processes P_0 through P_4;

3 resource types:

A (10 instances), *B* (5instances), and *C* (6 instances)

	Allocation	Need	Available
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 1	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

- Executing safety algorithm shows that sequence < P_1, P_3, P_4, P_0, P_2> satisfies safety requirement
- Can a request for (3,3,0) by P_4 be granted?
- Can a request for (0,2,0) by P_0 be granted?

End of Lecture

Thank you

Any questions?