Announcements

- Midterm is next 3/9/04
- Reading
  - Chapter 7 – can skip 7.7 & 7.9
  - Today Chapter 8
- Project #3 is available on the web
- Suggested problems:
  - 7.1, 7.2, 7.6, 7.8, 7.9, 7.15, 7.18

Banker's Algorithm - Example

Three resources: A, B, C (10, 5, 7 instances each)

Consider the snapshot of the system at this time

	Alloc	Max	Avail	Max - alloc Need
	A B C	A B C	A B C	A B C
P0	0 1 0	7 5 3	3 3 2	7 4 3
P1	2 0 0	3 2 2		1 2 2
P2	3 0 2	9 0 2		6 0 0
P3	2 1 1	2 2 2		0 1 1
P4	0 0 2	4 3 3		4 3 1

System is in a safe state, since the sequence <P1, P3, P4, P2, P0> satisfy the safety criteria.

Resource Request Algorithm

(1) If $Request_i <= Need_i$ then goto 3

- otherwise - the process has exceeded its maximum claim

(2) If $Request_i <= Available$ then goto 3

- otherwise process must wait since resources are not available

(3) Check request by having the system pretend that it has allocated the resources by modifying the state as follows:

- $Available = Available - Request_i$
- $Allocation = Allocation + Request_i$
- $Need_i = Need_i - Request_i$

- Find out if resulting resource allocation state is safe, otherwise the request must wait.

Deadlock Detection

- Resource Allocation Graph
  - Graph consists of vertices
    - type $P = \{P_1,..,P_n\}$ represent processes
    - type $R = \{R_1,..,R_m\}$ represent resources
  - Directed edge from process P_i to resource type R_j signifies that a process i has requested resource type j
  - *request edge*
  - A directed edge from R_j to P_i indicates that resource R_j has been allocated to process P_i
  - *assignment edge*

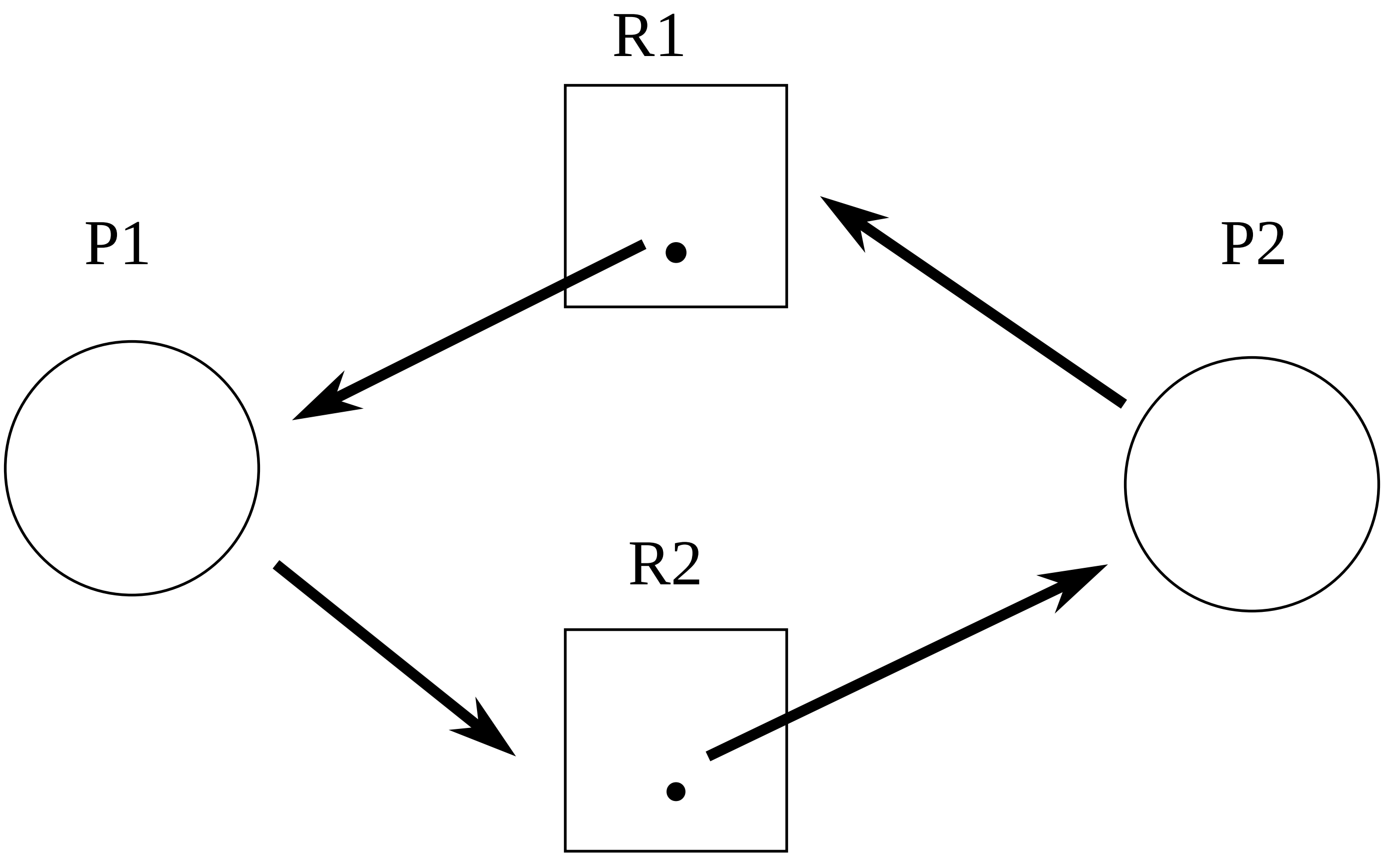
Deadlock Detection (cont.)

- Resource types may have more than one instance
- Each resource vertex represents a resource type.
- Each resource instance is of a unique resource type, each resource instance is represented by a "subvertex" associated with a resource vertex
  - (Silverschatz represents resource vertices by squares, resource instance "subvertices" by dots in the square. Process vertices are represented by circles)
- A request edge points to a resource vertex
- An assignment edge points from a resource "subvertex" to a process vertex

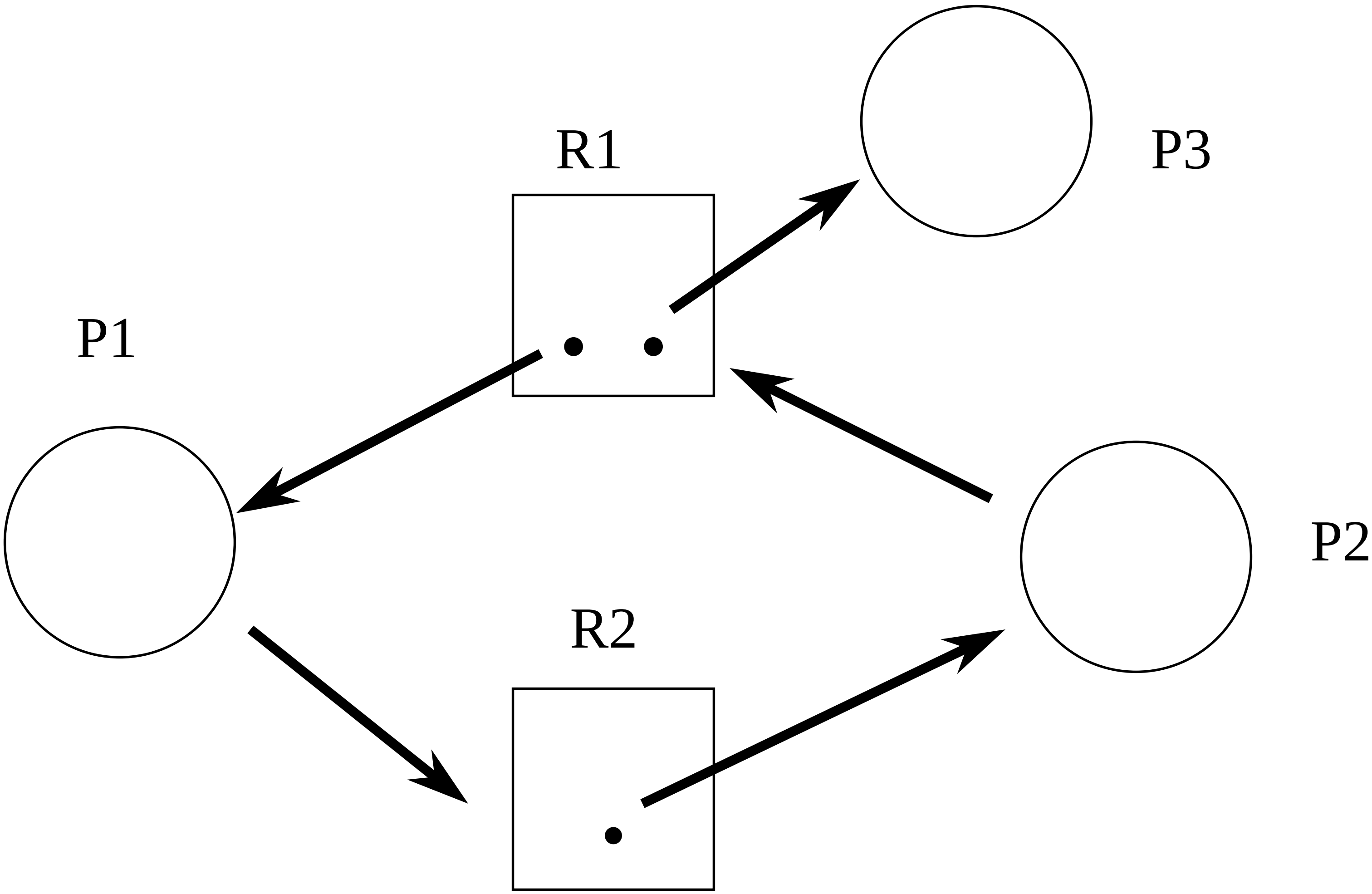
Resource Allocation Graph

- When a process P_i requests an instance of resource type R_j, a request edge is inserted into the resource allocation graph
- When the request can be fulfilled, the request edge is transformed into an assignment edge
- When the process is done using the resource, the assignment edge is deleted
- If the graph contains no cycles, no deadlock can exist

Deadlock!

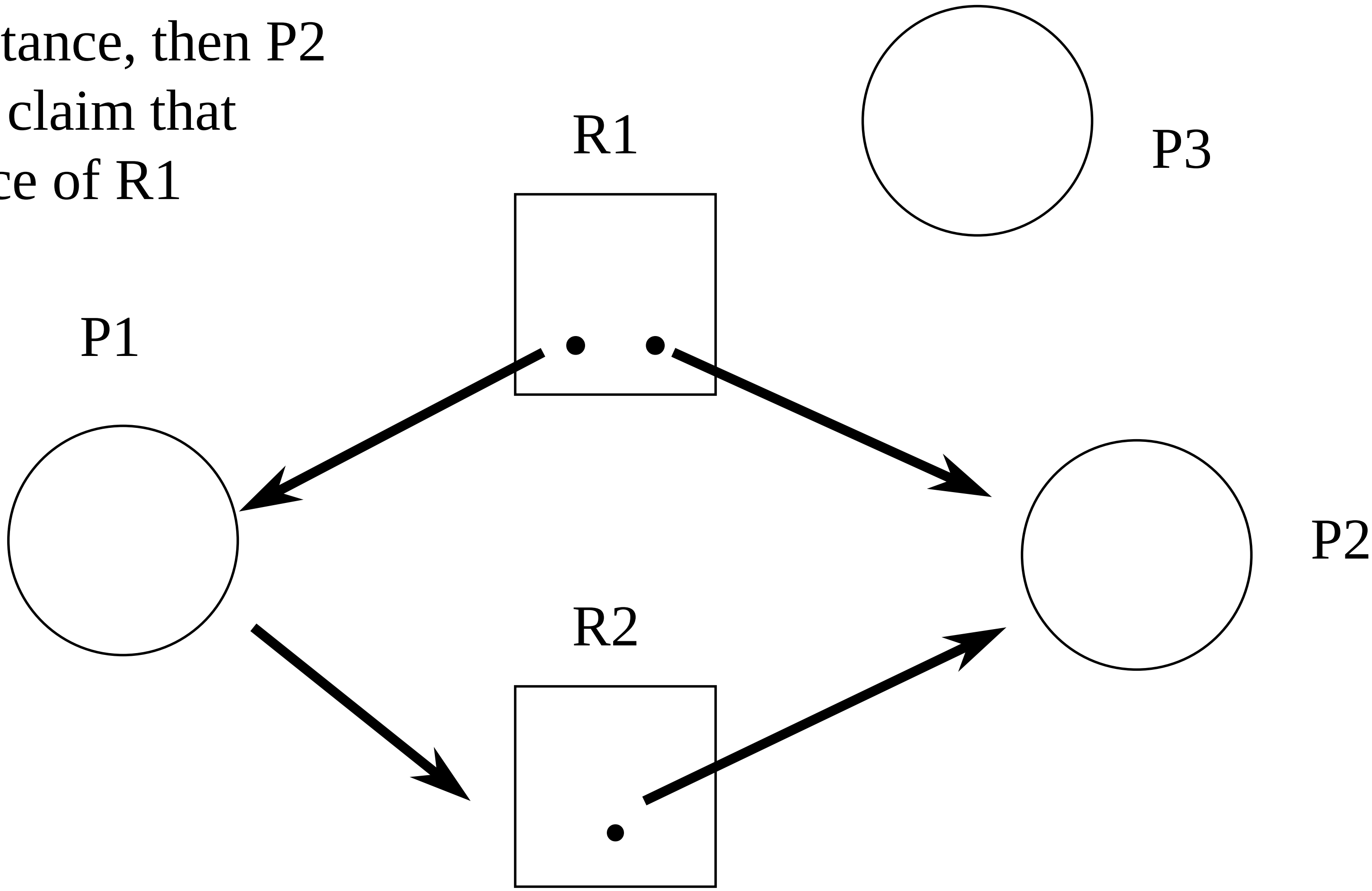


Deadlock??

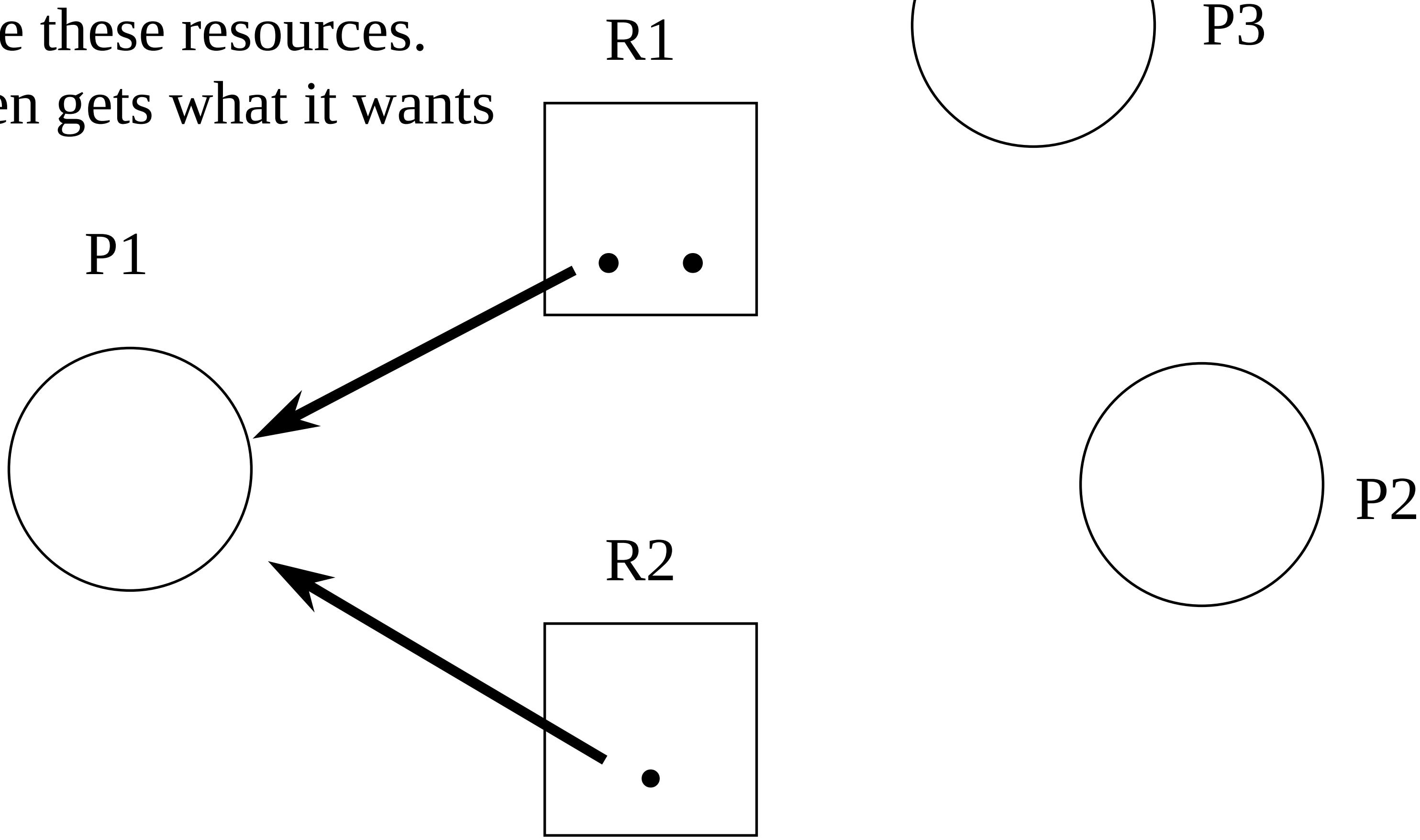


No!!

P3 could finish with
its instance of R1, release
the instance, then P2
would claim that
instance of R1



Then, P2 could finish with its instances of R1 and R2 and release these resources. P1 then gets what it wants



Detecting Deadlock

Work is a vector of length m (resources)

Finish is a vector of length n (processes)

- Allocation is an n x m matrix indicating the number of each resource type held by each process
- Request is an m x n matrix indicating the number of additional resources requested by each process

This is the difference from the Banker's algorithm.

1. Work = Available;

   if Allocation[i] != 0 Finish = false else Finish = true;

2. Find an *i* such that Finish[i] = false and Request_i <= Work if no such i, go to 4
3. Work += Allocation ; Finish[i] = true; goto step 2
4. If Finish[i] = false for some i, system is in deadlock

Note: this requires m x n^2 steps

Recovery from deadlock

- Must free up resources by some means
- Process termination
  - kill all deadlocked processes
  - select one process and kill it
    - must re-run deadlock detection algorithm again to see if it is freed.
- Resource Preemption
  - select a process, resource and de-allocate it
  - rollback the process
    - needs to be reset the process to a safe state
    - this requires additional state
  - starvation
    - what prevents a process from never finishing?

Sample Synchronization Problem

- Class Exercise:
  - **CMSC 412 Midterm #1 (Spring 1998)** Q#3