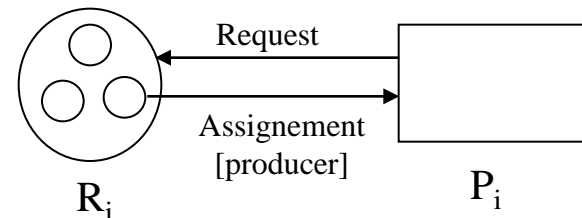


Deadlock analysis using the GRG

- The GRG is a powerful theoretical tool
 - simplified view of the state of the system
 - we are going to see theorems linking graph properties to presence/absence of deadlock
- A powerful analysis tool is the **graph reduction method**
 - the idea is to use a graph to test sets of operations on the system
 - a **reduction** is the most optimistic set of operations that unblocks one or more blocked processes

The Graph Reduction Method

- Reduction of a graph by a process P_i (unblocked):
 - for each reusable resource R_j
 - delete all request and assignment edges from/to P_i
 - increase r_j count by one for each assignment edge deleted
 - for each consumable resource R_j
 - delete all request and production edges from/to P_i
 - decrease r_j count by one for each request edge deleted
 - if P_i is a producer of R_j then set r_j to ∞



First Theorem on GRGs

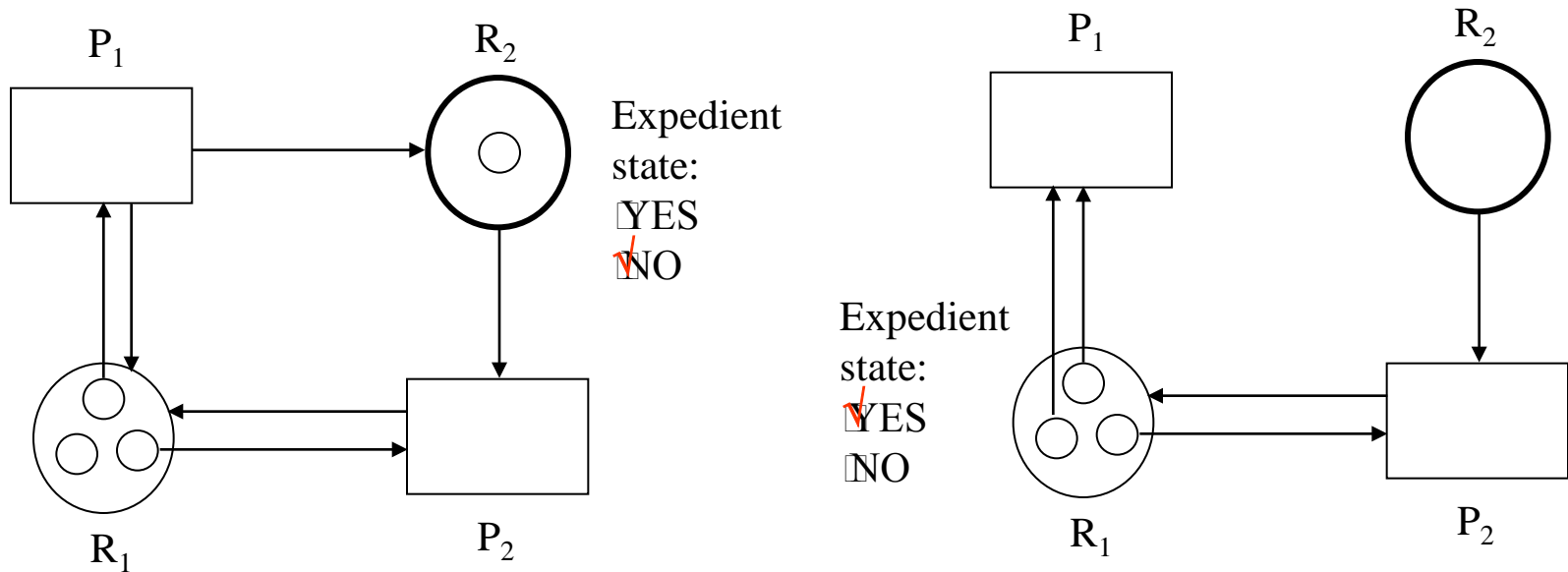
- Th. 3.1: “A process P_i is not deadlocked in a grg if and only if a sequence of reductions leaves the graph in a state in which P_i is not blocked”
- Definition:
 - A general resource graph is *completely reducible* if a sequence of reductions deletes all edges in the graph
- Corollary: “A system state is deadlock free if its general resource graph is completely reducible”

Theorem implications

- Corollary of Theorem 3.1 gives a sufficient condition for a state to be deadlock free
 - grg completely reducible \Rightarrow state is deadlock free
 - the inverse (\Leftarrow) is not necessarily true
 - thus lack of reducibility does not imply that the state is deadlocked
- Checking for a deadlock requires trying all possible different reduction sequences ($n!$)

Expedient state

- Definition:
 - A state is an **expedient state** if all processes having outstanding requests are blocked
- A state becomes expedient if all possible grantable requests have been granted



Second Theorem on GRGs

- Th. 3.2: “In a general resource graph:
 - a cycle is a necessary condition for a deadlock
 - if the graph is expedient, then a knot is sufficient condition for a deadlock”
- Next two slides describe the implications of this theorem

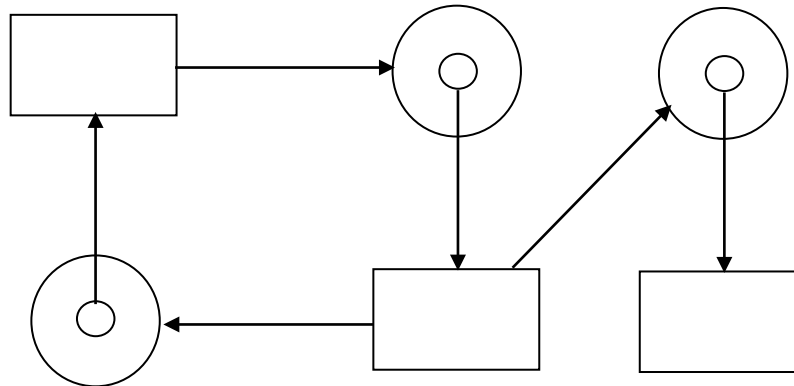
Theorem implications

- Statement #1:
 - a cycle is a necessary condition for a deadlock
- Practical consequence:
 - deadlock presence implies a certain graph property:
deadlock presence \Rightarrow cycle in the graph

Theorem implications (2)

- Statement #2:
 - if the graph is expedient, then a knot is sufficient condition for a deadlock
- Practical consequence:
 - the presence of two graph properties imply deadlock freedom:
expedient graph + knot \Rightarrow state is deadlocked
 - However absence of a knot does not imply absence of a deadlock
 - Example:

Deadlocked state
but no knot in the grg



Systems with only reusable resources

- Reusable resources reminder
 - examples: CPU, memory, I/O devices
 - a process holds the resource until done with it and then releases it to the next requesting process
 - not created nor destroyed, hence fixed number of units in the system
 - not “consumed” by use; contrast with consumable resources like messages, semaphore’s V operations, interrupts

Theorem on deadlock conditions

- Th. 3.5: “Let S be a state of a reusable resource system. Then,
 - if different sequences of reduction bring S into states that cannot be reduced, then all these states are identical
 - S is not a deadlocked state **if and only if** S is completely reducible”
- Next two slides explain implications of this theorem

Theorem implications (1)

- Statement #1
 - if different sequences of reduction bring S into states that cannot be reduced, then all these states are identical
- Practical consequence: we can use any sequence of reduction since they all yield the same result
- In systems with only reusable resources checking a state for deadlock is thus much simpler than in the general case

Theorem implications (2)

- Statement #2
 - S is not a deadlocked state **if and only if** S is completely reducible”
- Practical consequence:
 - deadlock absence is equivalent to a certain graph property:
graph completely reducible \Leftrightarrow state is free from deadlock
- Compare with general case:
 - graph completely reducible \Rightarrow state is free from deadlock
 - cannot say: “if graph not reducible then state is deadlocked”!

System with Single-Unit resources

- Th. 3.6: “If there is only a single unit of every resource, then a *cycle* in an expedient resource graph is a **necessary and sufficient** condition for a deadlock”
- In this particular case, deadlock presence is equivalent to a graph feature - a cycle
 - cycle search complexity is $O(n^2)$ for a n -node graph

Systems with single unit requests

- The General Resource Graph of such systems can have only a single outgoing edge from a process node
- Th. 3.5: “An expedient grg represents a deadlock if and only if it contains a knot”

Systems with single unit requests (2)

- Practical consequence:
 - deadlock presence is equivalent to a certain graph property:
graph expedient, contains a knot \Leftrightarrow state is deadlocked
- Compare with single unit, reusable resources case:
graph expedient, contains a cycle \Leftrightarrow state is deadlocked
- Compare with general case:
graph expedient, contains a knot \Rightarrow state is deadlocked
 - cannot say: “if there are no knots then state is not deadlocked”!

Systems with only consumable resources

- Definition: a *claim-limited graph* is the grg of a particular state, in which:
 - each resource has zero available units
 - there is a request edge (P_i, R_j) iff P_i is a consumer of R_j
- Example:

