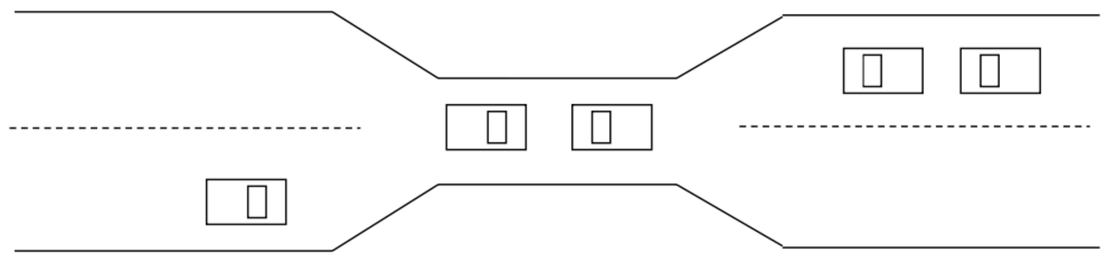
Deadlocks

Deadlock illustration

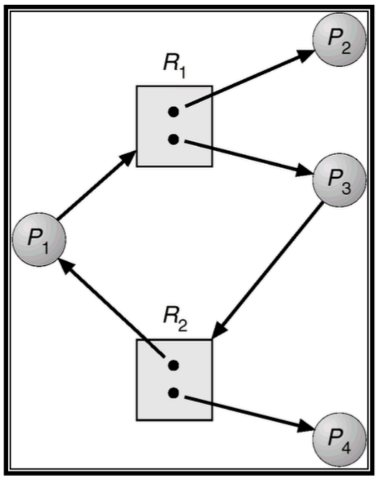


- Bridge is one lane and viewed as resource
- Deadlocks can be resolved if one car backs up
- Several cars may need to back up

Deadlock characterization: the 4 conditions

- Mutual exclusion
    - At least one non-sharable resource
    - Additional requests for resource delayed
- Hold and wait
    - Process holds at least one resource
    - Waiting to acquire resources held by other processes
- No preemption
    - Resources only released by a process after completing its task
- Circular wait
    - Set of processes {P0, P1, …, Pn}
    - P0 waiting for P1, P1 waiting for P2, …, Pn-1 waiting for Pn
    - Pn waiting for P0

Deadlock detection through a visual diagram

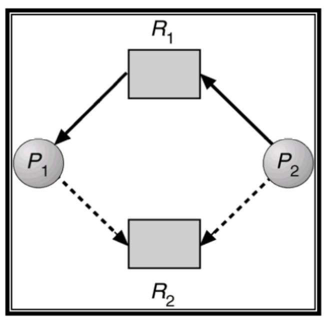


- System Resource Allocation Graphs
    - Processes (circle)
    - Resource (square, dots)
    - Directed Edges
        - Request
            - Pi → Rj
        - Assignment
            - Rj → Pi

Methods for handling deadlocks

- Ensure system cannot have deadlocks (precention or avoidance)
    - Deadlock Prevention
        - Eliminate at least one condition
            - Disadvantages
                - Low resource utliziation
                - Refuced throughput
        - Prevent Mutual Exclusion
            - Needed for non-shareable resources but not for sharable ones
            - Cannot prevent mutex for all resources: some intrinsically non-sharable
        - Prevent Hold and Wait
            - Guarantee that process does not hold a resource when requesting another
            - Example: allocate all resource before execution
        - Prevent no Preemption

                - One possibility: when requested not granted, processes must release all resources it is holding
                - Restarted only if it regains old and new resources
            - Prevent Circular Wait
                - Impose ordering of resources
                - Require process to request resource in increasing number
                    - If process holds R_j, it can request another R_j > R_i
                    - If process holds R_j and requests R_i, it must release R_j
        - Deadlock Avoidance
            - Needs information on how processes will request resources
            - Safe state
                - May allocate resources (up to maximum) in some order with no deadlock
                - Over-allocation: system unsafe
            - Includes *claim edge* (future request) denoted by P → R
            - Request granted only if no cycle is formed
            - Single instance resources only



            - Multiple resource instances: Banker's algorithm
            - Ensure bank does not over-allocate cash
            - Determines if allocating requested resources maintains safe state
            - Less efficient that system = resource allocation graph
- Allow deadlocks then recover
- Ignore deadlocks (ostrich algorithm)
- Deadlock Detection
    - Algorithm to check for deadlocks
    - Algorithm to recover from deadlock
    - Detection algorithm depends on two factors:
        - How often is a deadlock likely to occur?
        - How many processes are affected by the deadlock?
    - Single instance resource
        - Use Wait-For graph
    - Several instances
        - Uses time-varying data structures found in Banker's Algorithm