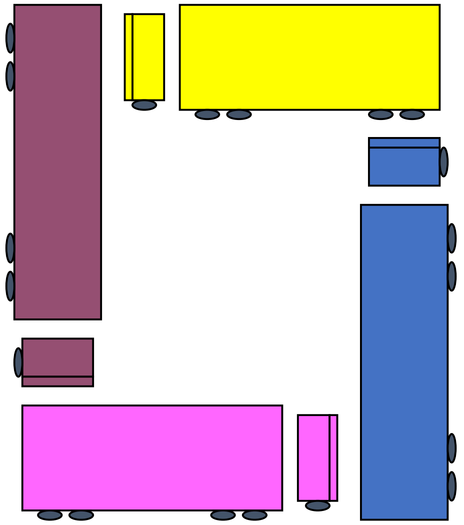
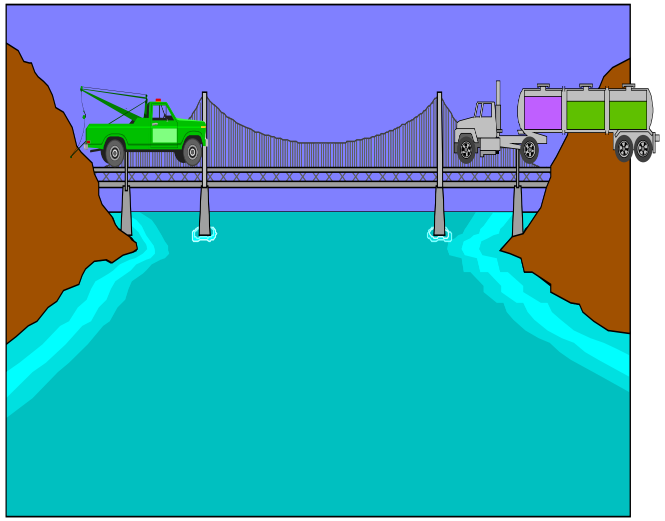
Operating Systems

Jinghui Zhong （钟竟辉）
Office：B3-515
Email ： jinghuizhong@scut.edu.cn

Computer Deadlock: What it is and how to model it

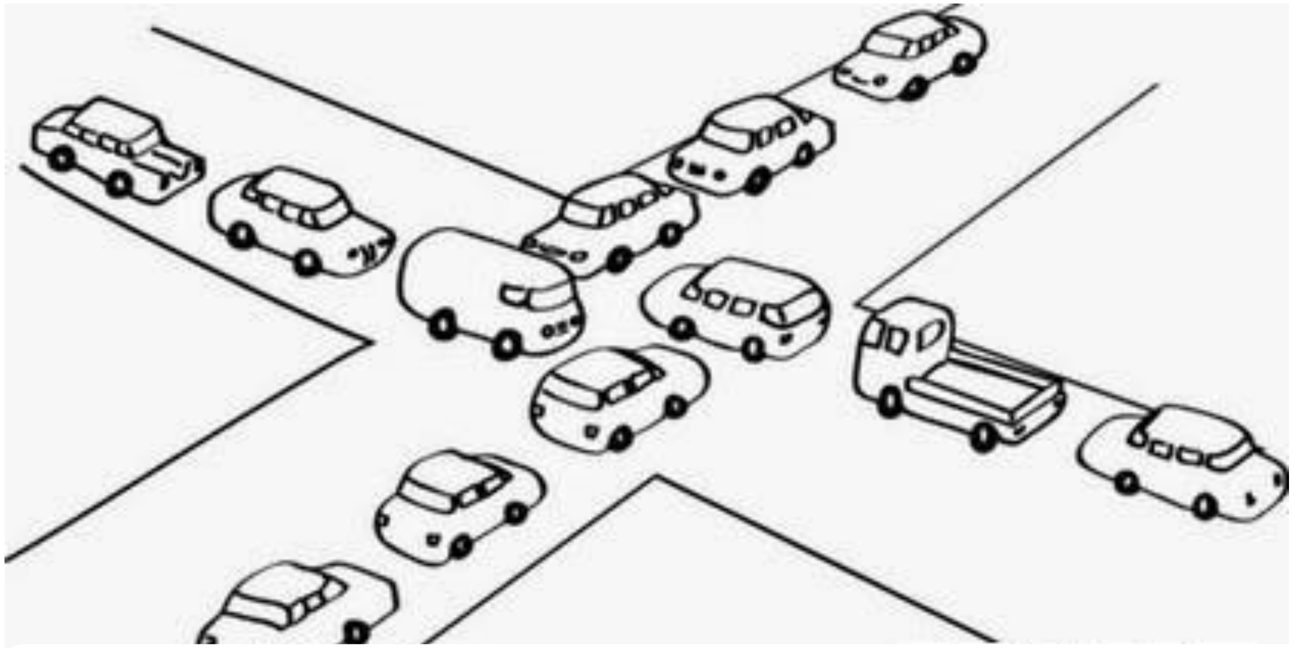




What is deadlock?

- **Examples**

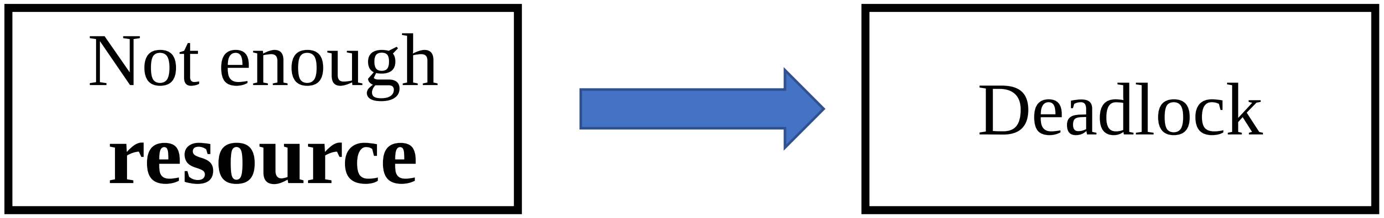
You can't get a job without experience;
You can't get experience without a job.



What is deadlock?

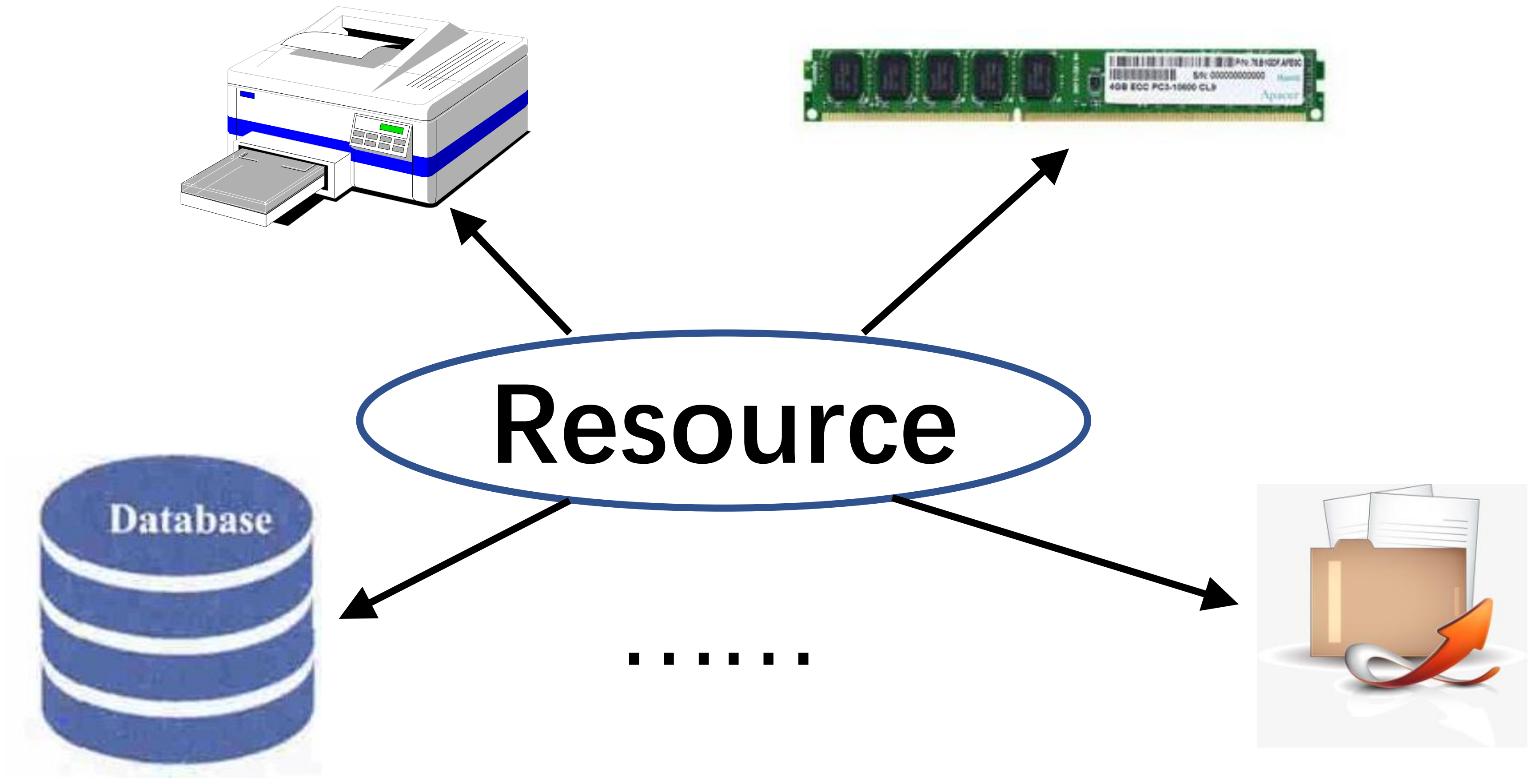
- **Definition**

A set of processes(e.g.,) is in a ***deadlock state*** when every process in the set is waiting for a **resource** that can only be released by another process in the set.



Resources

A **resource** is anything that can be acquired, used, and released over the course of time.



Two types of Resources

- **Preemptable resources**

  can be taken away from a process with no ill effects (e.g. memory)

- **Nonpreemptable resources**

  will cause the process to fail if taken away (e.g. CD recorder)

Potential deadlocks that involve Preemptable resources can usually be resolved by reallocating resources from one process to another.

Resource Usage

- Sequence of events required to use a resource
  - ✓request the resource
  - ✓use the resource
  - ✓release the resource

- Must wait if request is denied
  - ✓requesting process may be blocked
  - ✓may fail with error code

Resource Acquisition

- Associate a semaphore with each resource :

```
typedef int semaphore;
semaphore resource_1;
semaphore resource_2;
```

All semaphores are initialized to 1.

```
void process_A(void) {
   down(resource_1);
   down(resource_2);
   use_both_resources();
   up(resource_2);
   up(resource_1);
}
```

```
void process_B(void) {
   down(resource_1);
   down(resource_2);
   use_both_resources();
   up(resource_2);
   up(resource_1);
}
```

Resource Acquisition

- Code with a potential deadlock :

```
typedef int semaphore;
semaphore resource_1;
semaphore resource_2;
```

```
void process_A(void) {
   down(resource_1);
   down(resource_2);
   use_both_resources();
   up(resource_2);
   up(resource_1);
}
```

```
void process_B(void) {
   down(resource_2);
   down(resource_1);
   use_both_resources();
   up(resource_1);
   up(resource_2);
     }
```

Conditions for Deadlocks

Mutual exclusion

Resources are held by processes in a non-sharable (exclusive) mode.

Hold and Wait

A process holds a resource while waiting for another resource.

No Preemption

There is only voluntary release of a resource - nobody else can make a process give up a resource.

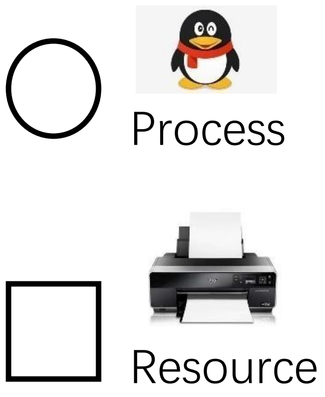
Circular Wait

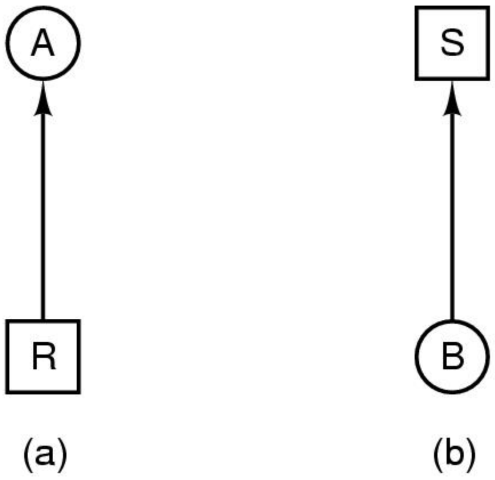
Process A waits for Process B waits for Process C .... waits for Process A.

ALL of these four conditions **must** happen simultaneously for a deadlock to occur.

Deadlock Modeling
(Resource with single instance)

- Modeled with **directed graphs**



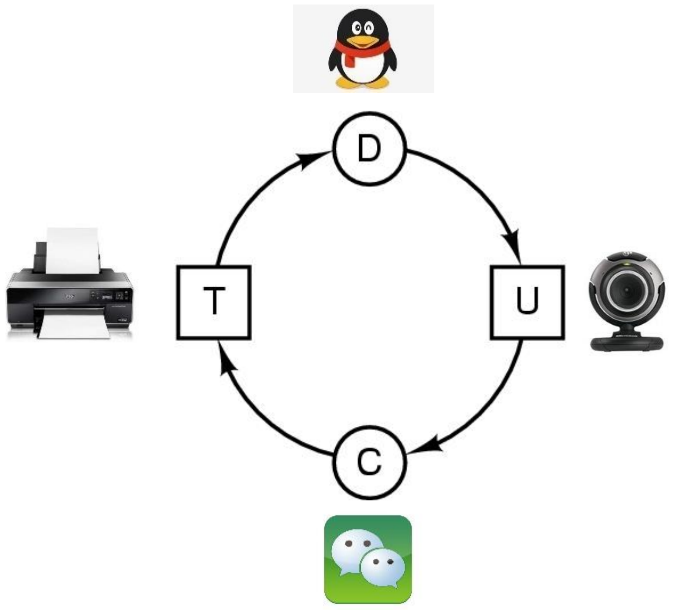


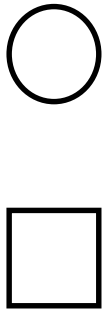
(a) resource R assigned to process A

(b) process B is requesting/waiting for resource S

Deadlock Modeling (Resource with single instance)

- Modeled with **directed graphs**





Process C and D are in deadlock over resources T and U

Resource-Allocation Graph

- A set of vertices V and a set of edges E.

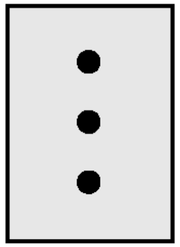
✓ V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the processes in the system.
  - $R = \{R_1, R_2, \ldots, R_m\}$, the resource types.

✓ Request edge: directed edge $P_1 \rightarrow R_j$

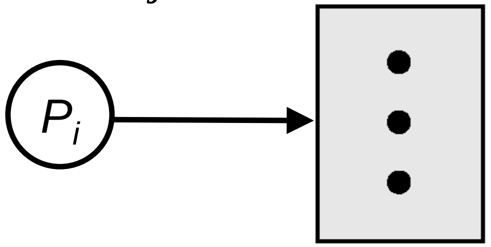
✓ Assignment edge: directed edge $R_j \rightarrow P_i$

Deadlock Modeling (Resource with multiple instances)

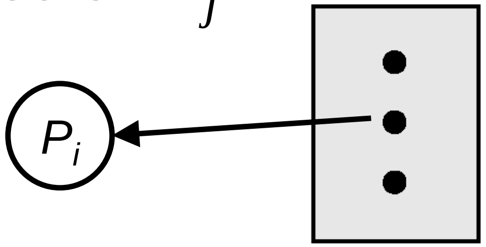
- Resource Type with 3 instances



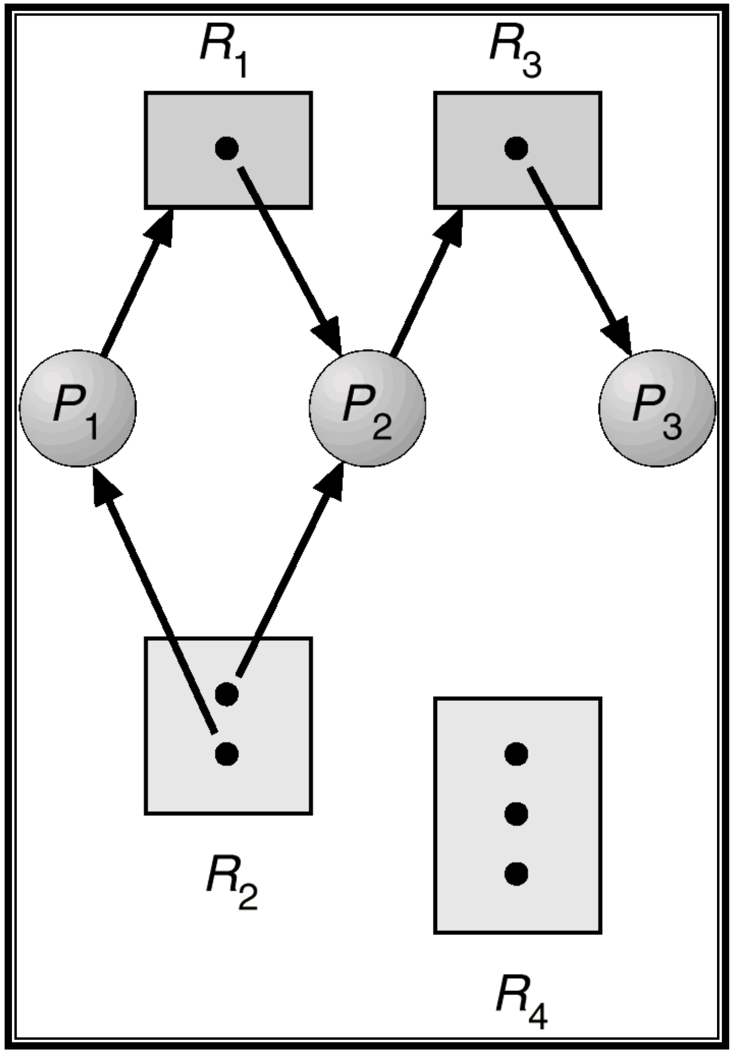
- P_i requests an instance of R_j



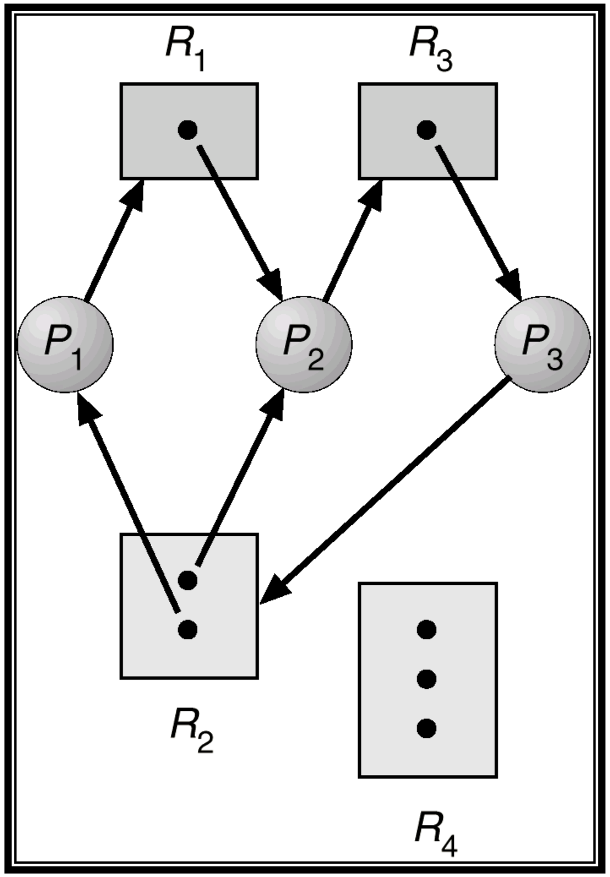
- P_i is holding an instance of R_j

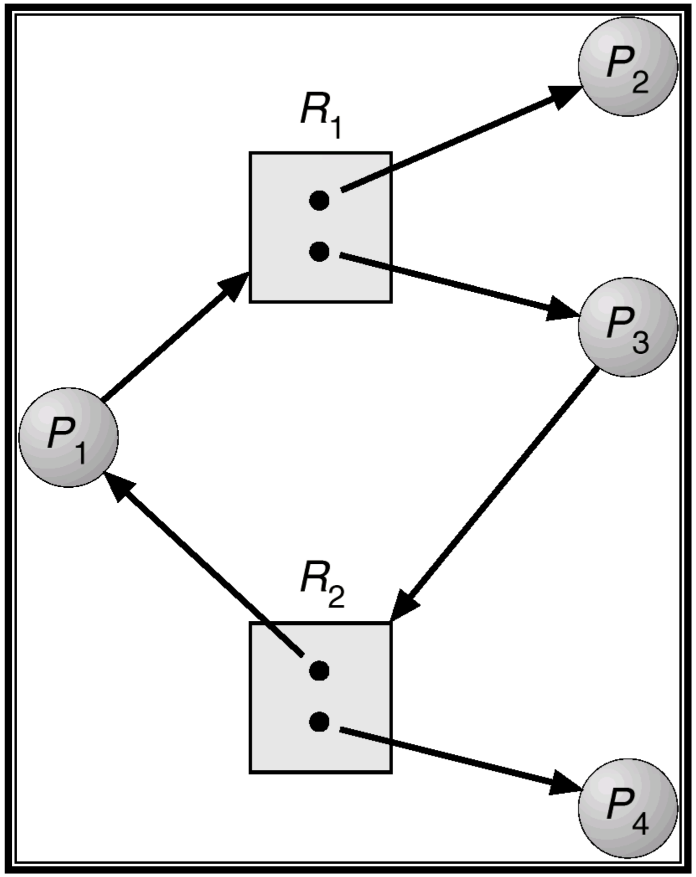


Example (1)



Examples (2)





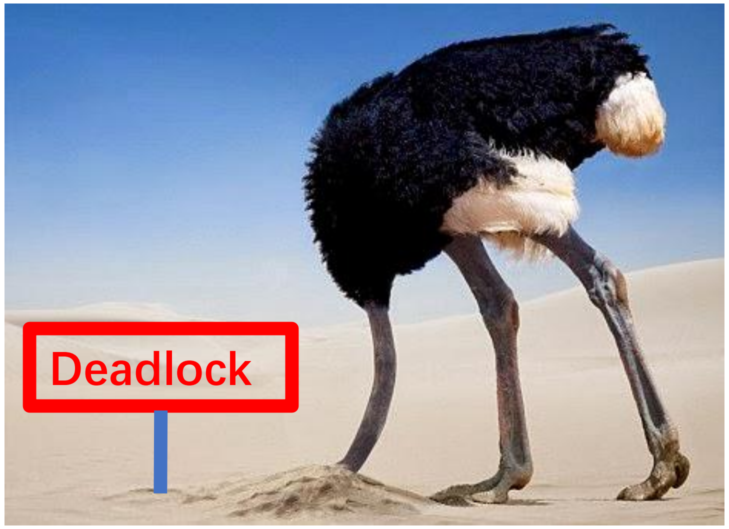
Basic Fact

- If graph contains **no cycles** ⇒ **no** deadlock.

- If graph contains **a cycle** ⇒
  - ✓if only one instance per resource type, then deadlock.
  - ✓if several instances per resource type, possibility of deadlock.

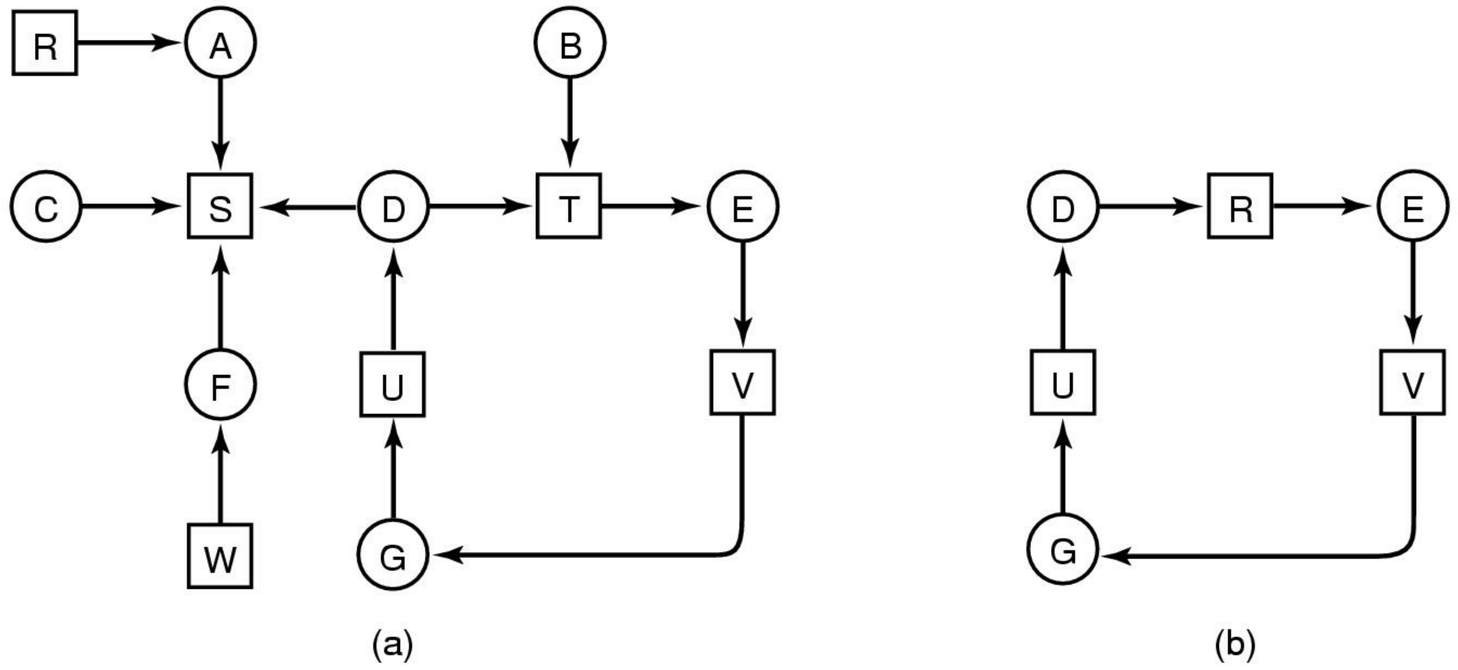
The Ostrich Algorithm

- Pretend that there is no problem



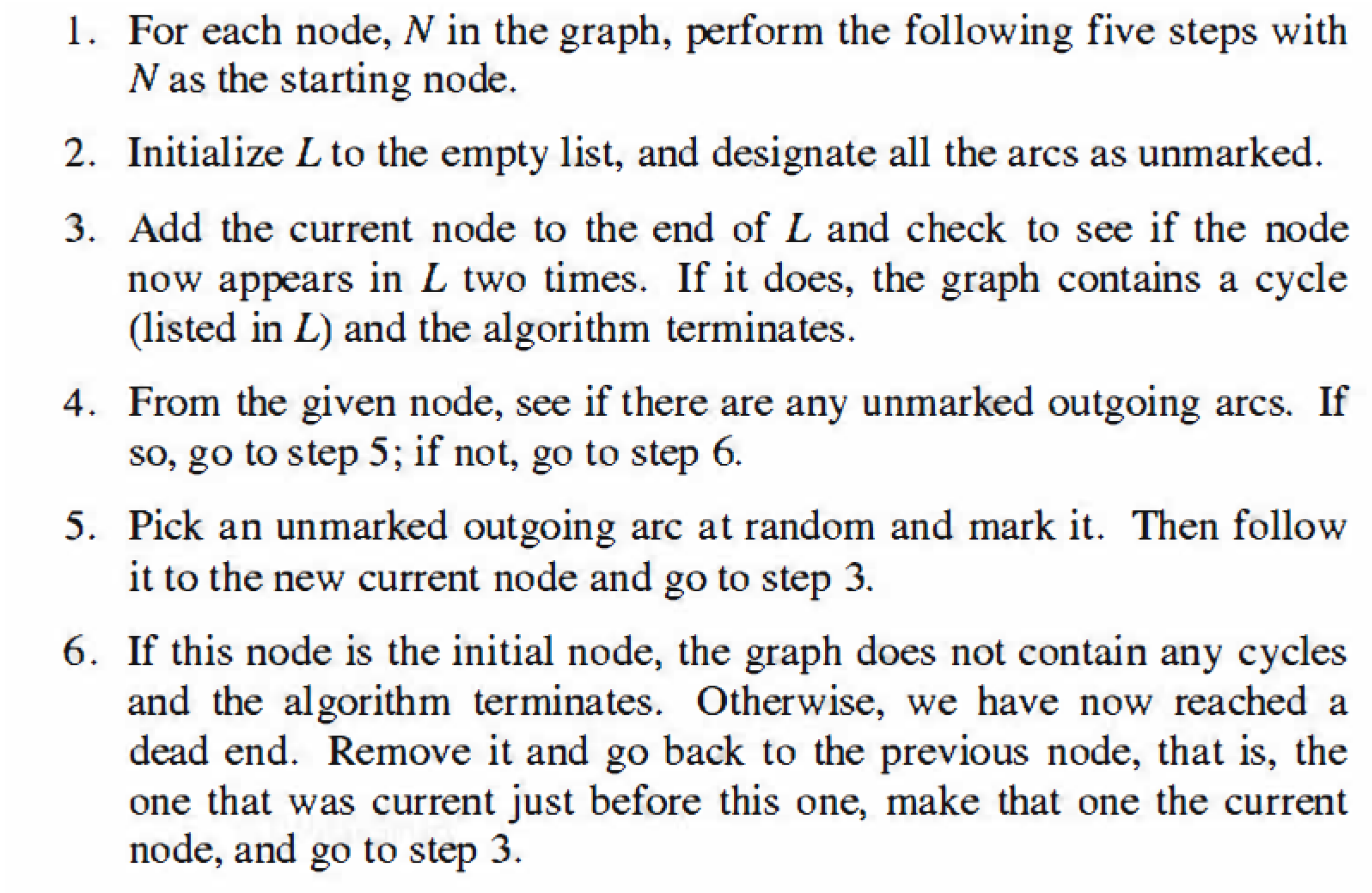
- Reasonable if
  ① deadlocks occur very rarely
  ② cost of prevention is high

Detection with One Resource of Each Type



- Note the resource ownership and requests
- A cycle can be found within the graph, denoting deadlock
- Many algorithms for detecting cycles in directed graphs.

Detecting Cycles in Directed Graphs



1. For each node, *N* in the graph, perform the following five steps with *N* as the starting node.
2. Initialize *L* to the empty list, and designate all the arcs as unmarked.
3. Add the current node to the end of *L* and check to see if the node now appears in *L* two times. If it does, the graph contains a cycle (listed in *L*) and the algorithm terminates.
4. From the given node, see if there are any unmarked outgoing arcs. If so, go to step 5; if not, go to step 6.
5. Pick an unmarked outgoing arc at random and mark it. Then follow it to the new current node and go to step 3.
6. If this node is the initial node, the graph does not contain any cycles and the algorithm terminates. Otherwise, we have now reached a dead end. Remove it and go back to the previous node, that is, the one that was current just before this one, make that one the current node, and go to step 3.

Detection with Multiple Resources of Each Type

Resources in existence
$(E_1, E_2, E_3, \ldots, E_m)$

Current allocation matrix

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & \cdots & C_{1m} \\ C_{21} & C_{22} & C_{23} & \cdots & C_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ C_{n1} & C_{n2} & C_{n3} & \cdots & C_{nm} \end{bmatrix}$$

Row n is current allocation to process n

Resources available
$(A_1, A_2, A_3, \ldots, A_m)$

Request matrix

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & \cdots & R_{1m} \\ R_{21} & R_{22} & R_{23} & \cdots & R_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ R_{n1} & R_{n2} & R_{n3} & \cdots & R_{nm} \end{bmatrix}$$

Row 2 is what process 2 needs

Data structures needed by deadlock detection algorithm

$$\sum \boldsymbol{C}_{ij} + \boldsymbol{A}_j = \boldsymbol{E}_j$$

Detection with Multiple Resources of Each Type

$$E = \begin{array}{ccccc} & \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ E = (& 4 & 2 & 3 & 1\ ) \end{array}$$

$$A = \begin{array}{ccccc} & \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ A = (& 2 & 1 & 0 & 0\ ) \end{array}$$

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

Deadlock Detection Algorithm

1. Recovery look for an unmark process P_i s.t. the i-th row of ***R*** is less than ***A***. (P_i 's request can be satisfied)
2. If such a process is found, add the i-th row of ***C*** to A, mark the process and go back to step 1. (When Pi completes, its resources will become available).
3. If no such process exist, the algorithm terminates. The unmarked process, if any, are deadlock.

Deadlock Detection Algorithm

```
Add all nodes to Unfinished;
Do{
    done = true
    foreach node in Unfinished{
        if([$R_{node}$] <= [A]){
            remove node from Unifished;
            [A] = [A] + [$C_{node}$]
            done = false;
        }
    }
}until(done)
```

Current Request Resources of nod

Available Resources

Resources held by node

Deadlock Detection: An Example

$$E = \begin{array}{cccc} \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ (\,4 & 2 & 3 & 1\,) \end{array}$$

$$A = \begin{array}{cccc} \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ (\,2 & 1 & 0 & 0\,) \end{array}$$

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

A = (2 2 2 0)

A = (4 2 2 1)

Recovery from Deadlock

- Recovery through preemption
  - ✓take a resource from some other processes
  - ✓depends on the nature of the resource
- Recovery through rollback
  - ✓checkpoint a process periodically
  - ✓use this saved state
  - ✓restart the process if it is found deadlocked

Recovery from Deadlock

- Recovery through killing processes
  - ✓crudest but simplest way to break a deadlock
  - ✓kill one of the processes in the deadlock cycle
  - ✓the other processes get its resources
  - ✓choose process that can be rerun from the beginning

Check points

- What is deadlock?
- What is resource in computer?
- What are the four conditions for a deadlock to occur?
- How to detect deadlock?
- How to recovery from deadlock?