Deadlock

Basic Definitions
Different Resource Types
Graph Theoretical Treatment
Detection, Prevention, and Avoidance

The Deadlock Problem

Definition: Deadlock is the permanent blocking of a set of processes that either compete for system resources or communicate with each other!

Formal treatment of deadlocks goes back over 30 years:

- Dijkstra (1968)
- Holt (1971, 1972)

We distinguish 3 different types of approaches to the deadlock problem:

1. Detection and Recovery;
2. Prevention;
3. Avoidance;

We consider two different types of system resources:

1. Reusable Resources → permanent objects with two properties:
   a) # of units is constant
   b) each unit is available or allocated to exactly one process;
2. Consumable Resources → produced and consumed dynamically:
   a) # of units will vary over time;
   b) system can create new units
   c) process may request, acquire, and consume resource units

Examples

Examples of reusable resources are:

- Devices → printers, tapes, disk, etc.
- Memory
- Service Routines (critical sections!!)

Examples of consumable resources are:

- messages;
- signals;
- interrupts;
- events;
- data structures (dynamic);

Deadlock Example 1:

P1:	P2:
…	…
request(Disk);	request(Tape);
…	…
request(Tape);	request(Disk);
….	…
release(Tape);	release(Disk);
release(Disk);	release(Tape);
…	…

Let's analyze the problem…

What causes the deadlock condition?

… more examples

Deadlock Example 2:

Available: 100 blocks of memory!

P1:	P2:
…	…
request(40Blocks);	request(30Blocks);
…	….
request(40Blocks);	request(50Blocks);

Even though neither P1 nor P2 request more than the available amount, deadlock can occur.

Note: their first request can easily be granted!

Deadlock Example 3:

Consider a message exchange between P1 and P2.

P1:	P2:
…	…
receive(P2, M1);	receive(P1, M2));
…	…
send(P2, M1');	send(P1, M2');

Note: Messages are consumable, yet both processes are waiting for a message to be sent by the other process.

Processes and States…

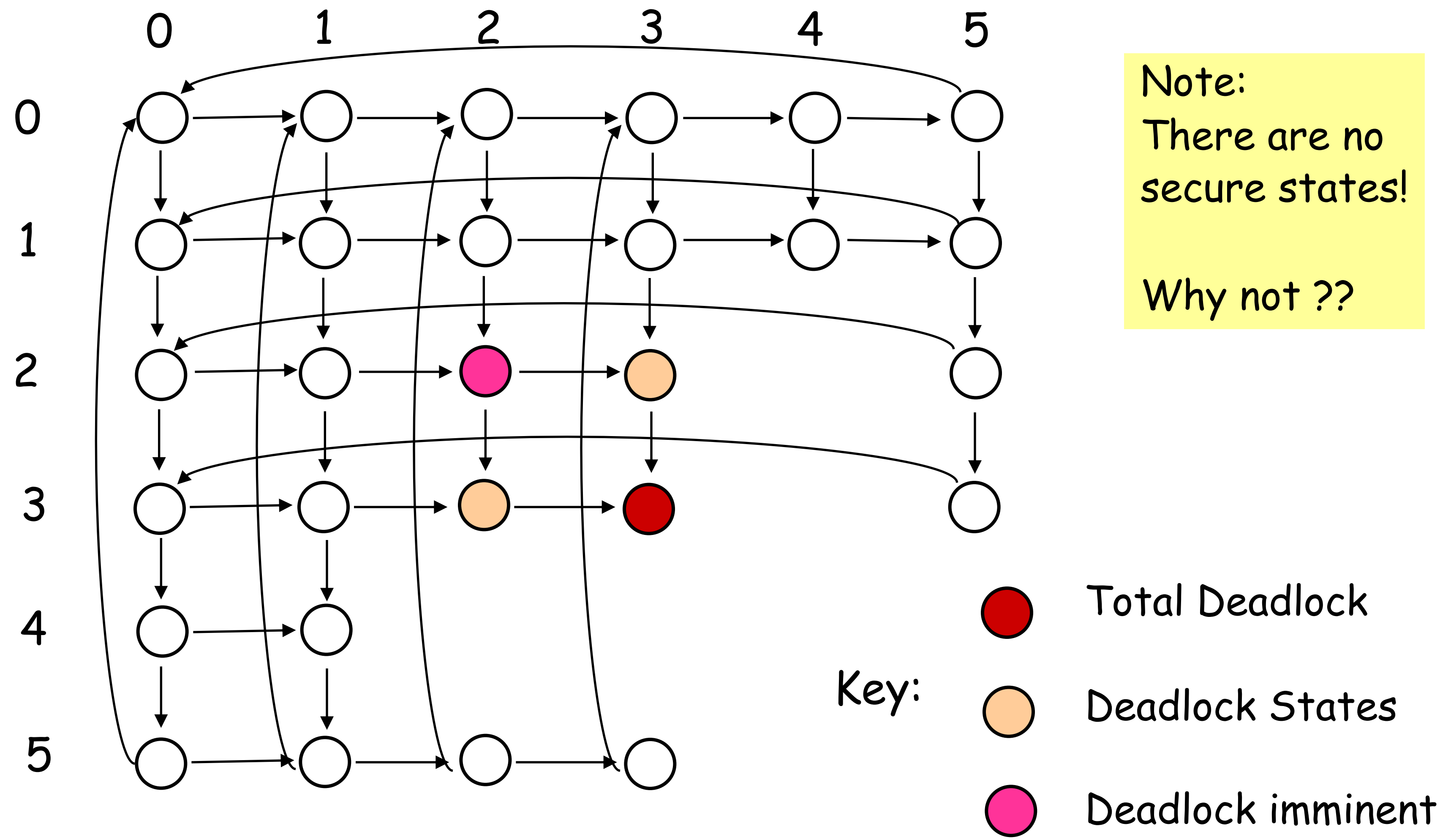
The following actions by P1 and P2 will result in state changes.

	P1:	P2:
0:	…	…
	while(1){	while(1){
1:	Req(D);	Req(T);
2:	…	…
3:	Req(T);	Req(D);
4:	…	…
	Rel(T);	Rel(D);
5:	…	…
	Rel(D);	Rel(T);
	}	}

P1:
0: Holds none
1: Holds none, requests D
2: Holds D
3: Holds D, requests T
4: Holds D and T
5: Holds D, T is released

P2:
0: Holds none
1: Holds none, requests T
2: Holds T
3: Holds T, requests D
4: Holds T and D
5: Holds T, D is released

A State Diagram



Note:
There are no secure states!

Why not ??

A few definitions

- Let Σ be the set of all possible resource allocation states i.e., $\Sigma=\{S,T,U,V\}$
- Let Π be the set of Processes, e.g., $\Pi =\{P_1, P_2\}$
- *A process P_i is blocked in state S* if the does not exist a state T, such that $S \rightarrow^i T$. i.e., no action by processes P_i will result in a state change.
- *Process P_i is deadlocked* in state S if P_i is blocked in state S and for all states T with $S \rightarrow^* T$, P_i is blocked in T
- If P_i is deadlocked in state S, then S is a *deadlock state*.
- If all processes P_i are deadlocked in state S, Then S is a *total deadlock state*.
- *State S is secure* is S is not a deadlock state and, any state T reachable from S ($S \rightarrow^* T$) is not a deadlock state.
- NOTE: it is possible for a process P_i to be deadlocked while other processes are running.

Necessary Deadlock Conditions

- The following are necessary conditions that must exits for deadlock to occur:

1. Mutual Exclusion → Processes hold resources exclusively, hence making them unavailable to other processes.

2. Nonpreemption → Resources cannot be taken away from a process that is holding them. Only the process can release the resources they hold.

3. Resource Waiting → Processes that request unavailable resources (or units) will block until they become available. (*Circular Wait*)

4. Partial Allocation → Processes may hold some resources when they request additional units of the same resource or other resources. (*Hold and Wait*)

Note: the violation of any one of these conditions will result in a deadlock-free system.

Formal Treatment of Deadlocks

A useful model for the interaction between processes and resources is based on concepts from Graph Theory.

We define a Resource Graph, G(V,E), where V is the set of vertices and E is the set of edges.

The set of vertices (V) consists of process nodes and resource nodes.

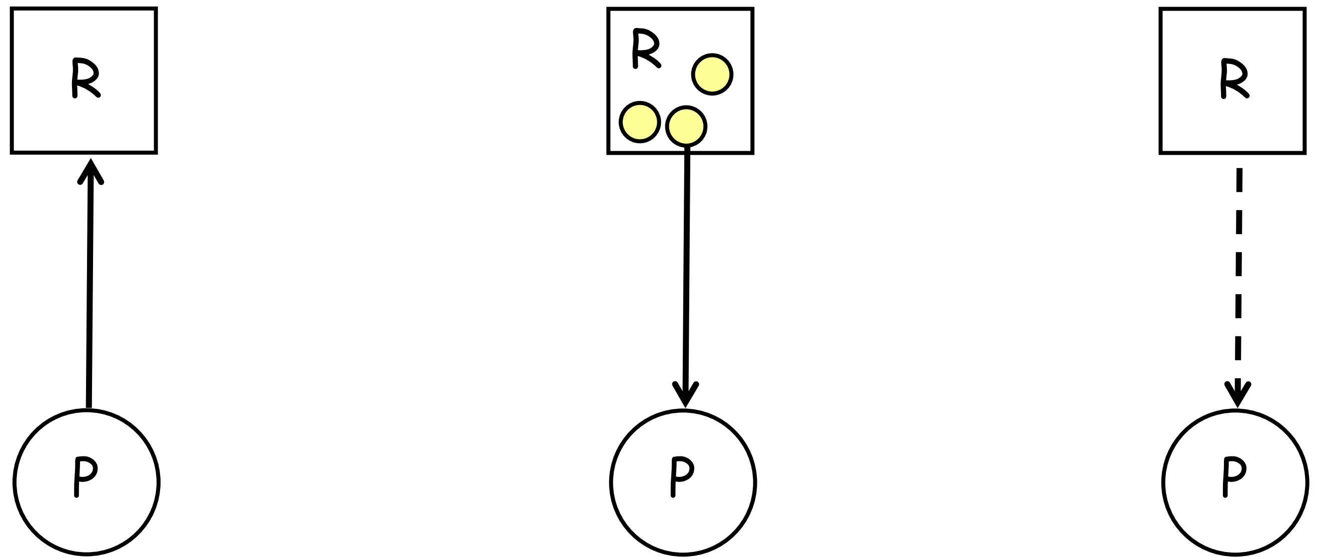
The set of edges (E) consists of

- Request Edges
- Assignment Edges
- Producer Edges

A resource graph, G, is a directed graph (digraph) → edges are directed from processes to resources:

- Request Edges → from process to recourses, signify that a process is requesting that resource (or units thereof).
- Assignment Edges → from resource to process, indicating that a particular resource is assigned to a specific process.
- Producer Edge → connects consumable resources to processes that produce them.

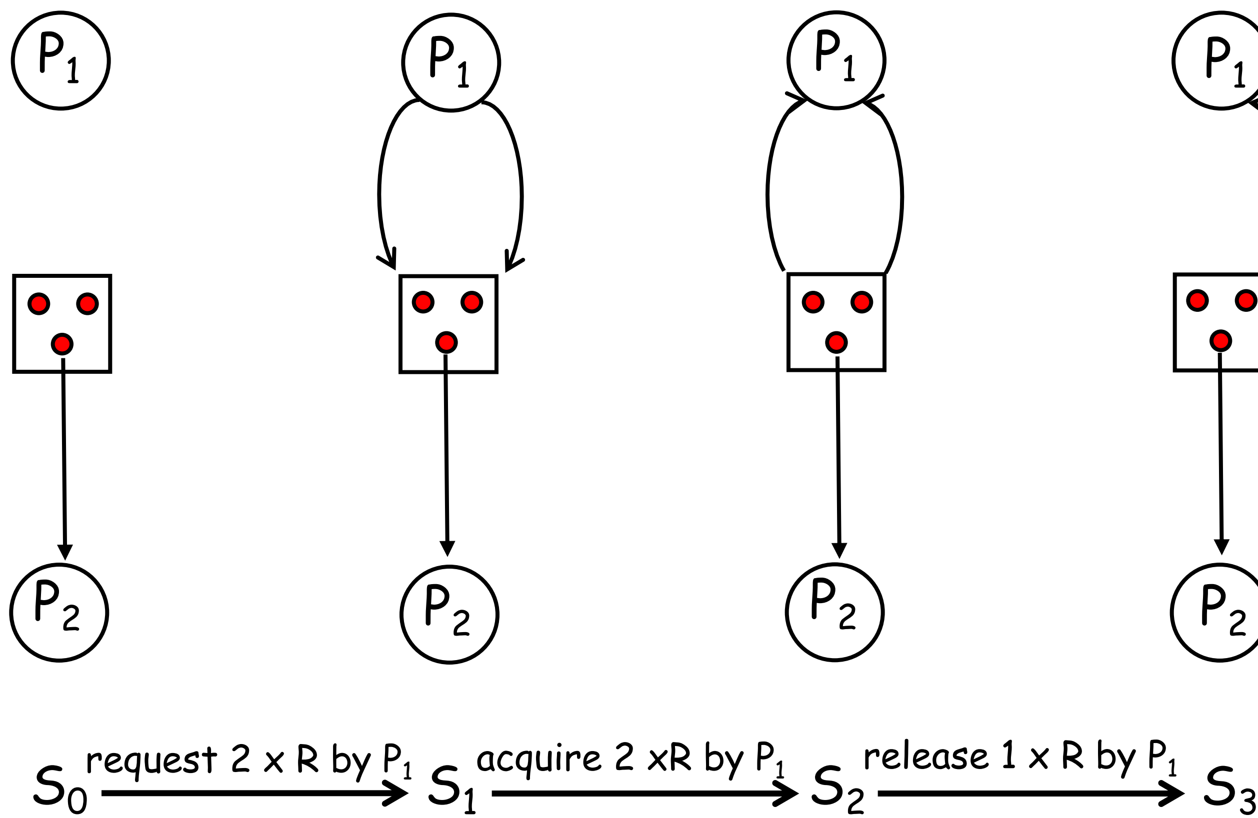
examples



Request Edge | Assignment Edge | Producer Edge

State Transitions

We can use the resource graph to depict state transition as show in the following example:



Requests and Acquisitions

1. Requests → Any process p_i that has no outstanding requests (i.e., it is not blocked) in a given state may request units of any number of resources R_j, R_k, …

    - In the resource graph, this is reflected by adding edges (P_i, R_j), (P_i, R_k), …corresponding to the number of units of each resource requested.
    - For reusable resources, the number of requested must never exceed the number of available resource units.

2. Acquisition → A process p_i may acquire resources that have previously been requested, provided that the number of requested units are available.

    - In the resource graph, this is reflected by reversing the request edges (P_i, R_j) to (R_j, P_i) for reusable resources.
    - For consumable resources, we remove the request edges (P_i, R_j), and remove the number of resource units from R, indicating that they have been consumed.