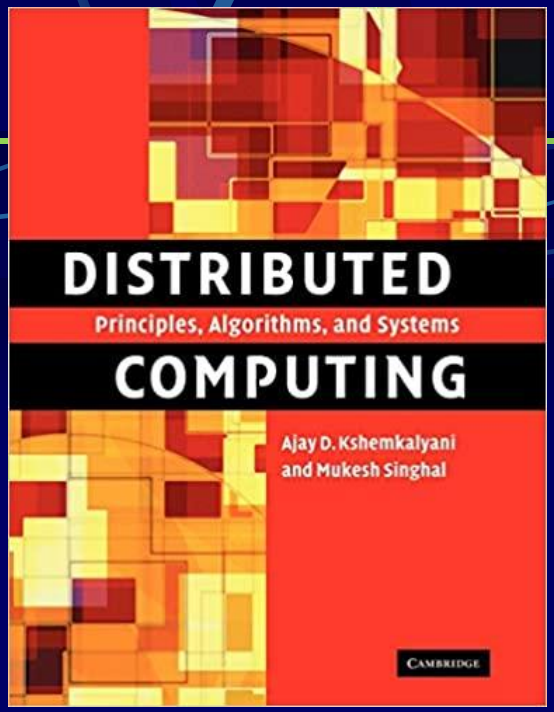
Deadlock Detection



Introduction

A process may request resources in any order, which may not be known a priori and a process can request resources while holding others.

A deadlock is a state where a set of processes request resources that are held by other processes in the set.

Process states

A process can be in two states: *running* or *blocked*.

- In the running (active) state, a process has all the needed resources and is either executing or is ready for execution.
- In the blocked state, a process is waiting to acquire some resource.

WFG

The state of the system can be modeled by directed graph, called a *wait for graph*.

In a WFG , nodes are processes and there is a directed edge from node P1 to node P2 if P1 is blocked and is waiting for P2 to release some resource.

A system is deadlocked if and only if there exists a directed cycle or knot in the WFG.

Correctness criteria

1. Liveness
   - The algorithm must detect all existing deadlocks in finite time.

Correctness criteria

2. Safety
   - The algorithm should not report deadlocks which do not exist (called *phantom or false* deadlocks).

Resolution of a deadlock

Deadlock resolution involves breaking existing wait-for dependencies between the processes to resolve the deadlock.

It involves rolling back one or more deadlocked processes and assigning their resources to blocked processes so that they can resume execution.

Models of deadlocks

Distributed systems allow several kinds of resource requests.

- Single resource model
- AND model (resource, all)
- OR model (communication, any)

Single resource model

A process can have at most one outstanding request for only one unit of a resource.

The presence of a cycle in the WFG indicates a deadlock.

AND model

A process can request for more than one resource simultaneously and the request is satisfied only after all the requested resources are granted to the process.

The presence of a cycle in the WFG indicates a deadlock.

OR model

A process can make a request for numerous resources simultaneously and the request is satisfied if any one of the requested resources is granted.

The presence of a cycle in the WFG does not imply a deadlock. The presence of a knot indicates a deadlock.

Knapp's classification

Distributed deadlock detection algorithms can be divided into four classes:

1. Path-pushing
2. Edge-chasing
3. Diffusing computation
4. Global state detection

Edge-Chasing

- The presence of a cycle in a distributed graph structure is verified by propagating special messages called probes, along the edges of the graph.
- Whenever a running process receives a probe message, it discards this message and continues. Only blocked processes propagate probe messages along their outgoing edges.
- The formation of a cycle can be detected by a site if it receives the matching probe sent by it previously.

Diffusing Computation

- Deadlock detection computation is diffused through the WFG of the system.
- This computation is superimposed on the underlying distributed computation.
- If this computation terminates, the initiator declares a deadlock.

Diffusing Computation

- To detect a deadlock, a process sends out query messages along all the outgoing edges in the WFG.
- These queries are successively propagated (i.e., diffused) through the edges of the WFG.

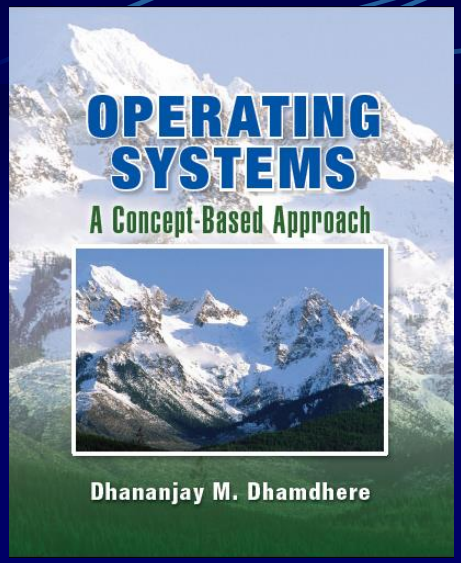
Diffusing Computation

- When a blocked process receives the first query message for a particular deadlock detection initiation, it does not send a reply message until it has received a reply message for every query it sent.
- For all subsequent queries for this deadlock detection initiation, it immediately sends back a reply message.

Diffusing Computation

- The initiator of a deadlock detection computation detects a deadlock when it receives reply for every query it had sent out.

A distributed algorithm for deadlock detection and resolution



Mitchel, M. and Merrit, M. (1984).

Proceedings of the ACM Symposium on Principles of Distributed Computing, pp. 282-284.

Chapter 18

Distributed Control Algorithms

Deadlock occurs when four conditions hold

- The access to each resource is mutually exclusive
- Requesting processes hold ressources while requesting for more (*hold and wait*)
- Resource scheduling is nonpreemptive
- Each process waits for another process to release a ressource (*circular waiting*)

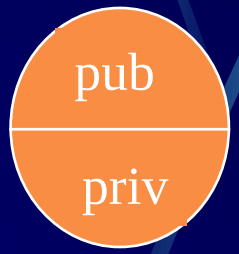
This is a very simple algorithm

- Single-resource model
- *Edge chasing* algorithm - control messages (**probes**) are *sent* over WFG edges to detect cycles
- Only one process detects deadlock

Each process is assigned a public label and a private label

The private label of each process is

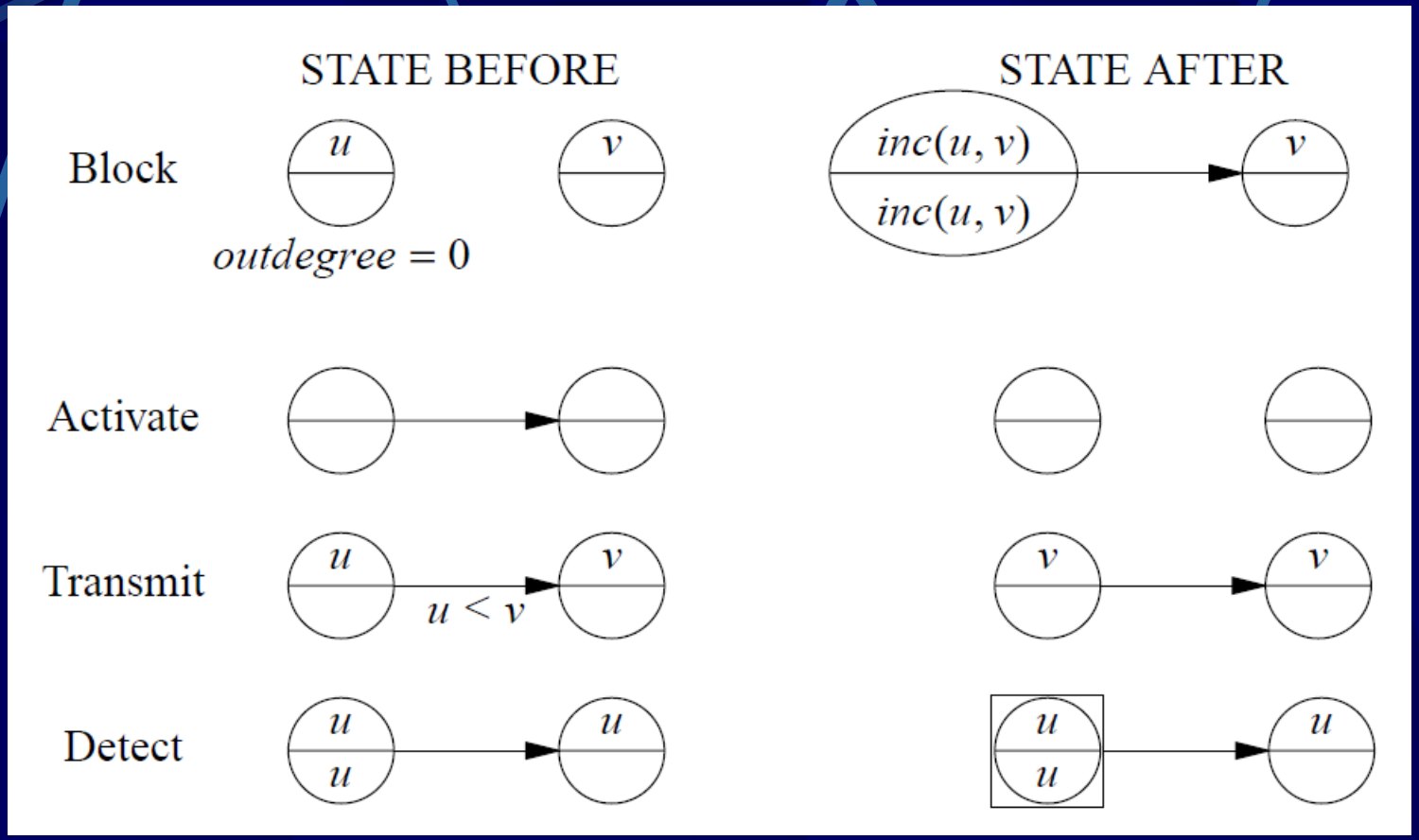
- always unique to that process, and
- non-decreasing over time



(count, process id)

The two labels are identical when a process is created

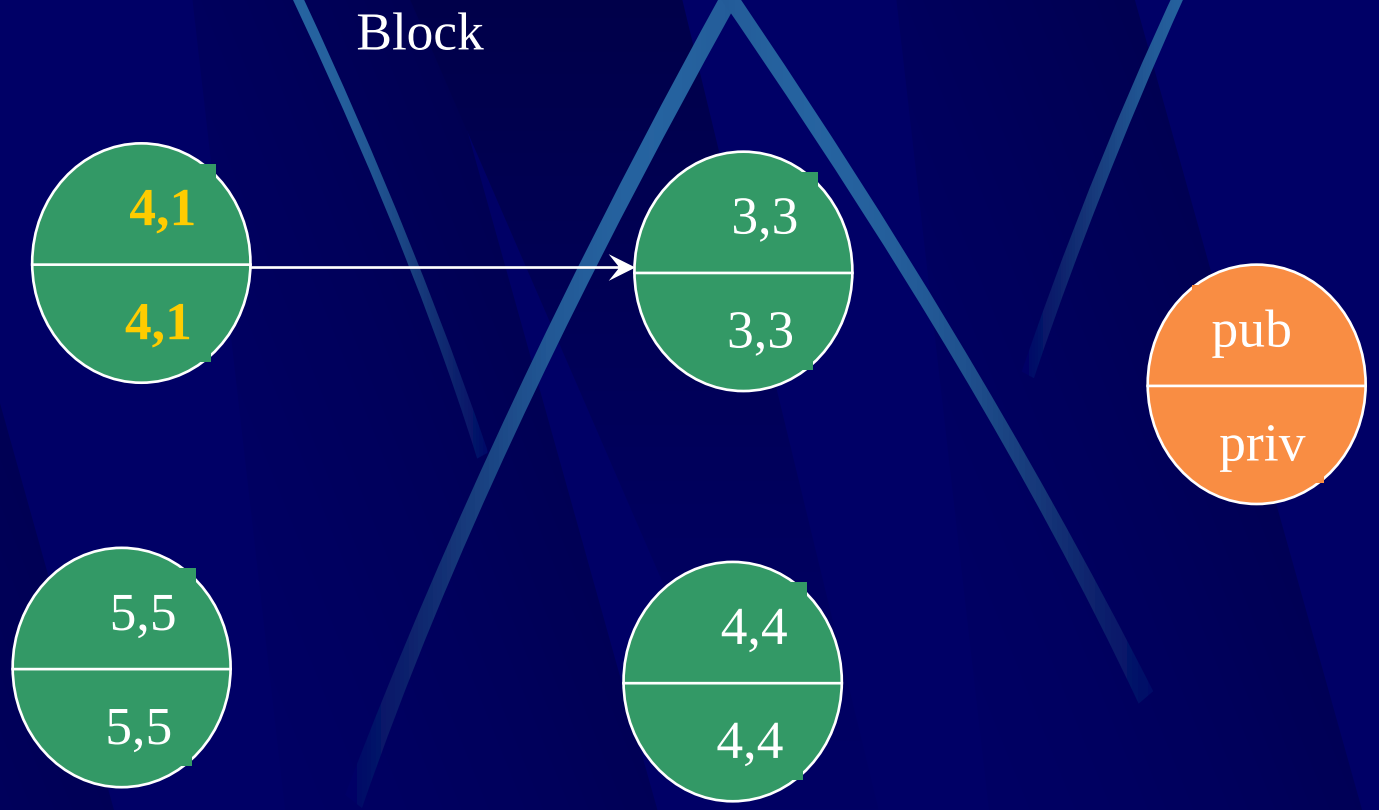
The algorithm has four nondeterministic rules



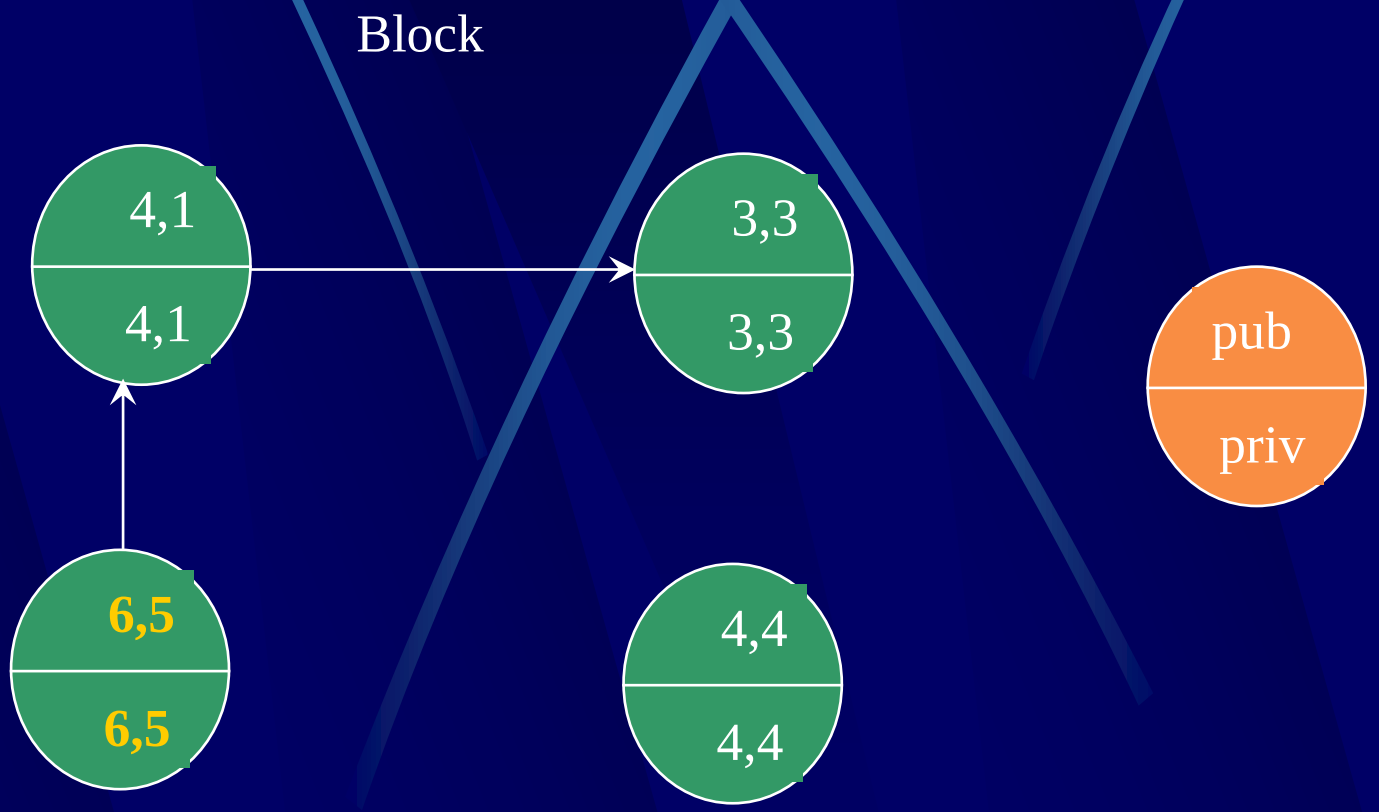
inc(u,v) means a value larger than both *u* and *v* that is unique to that node

Example

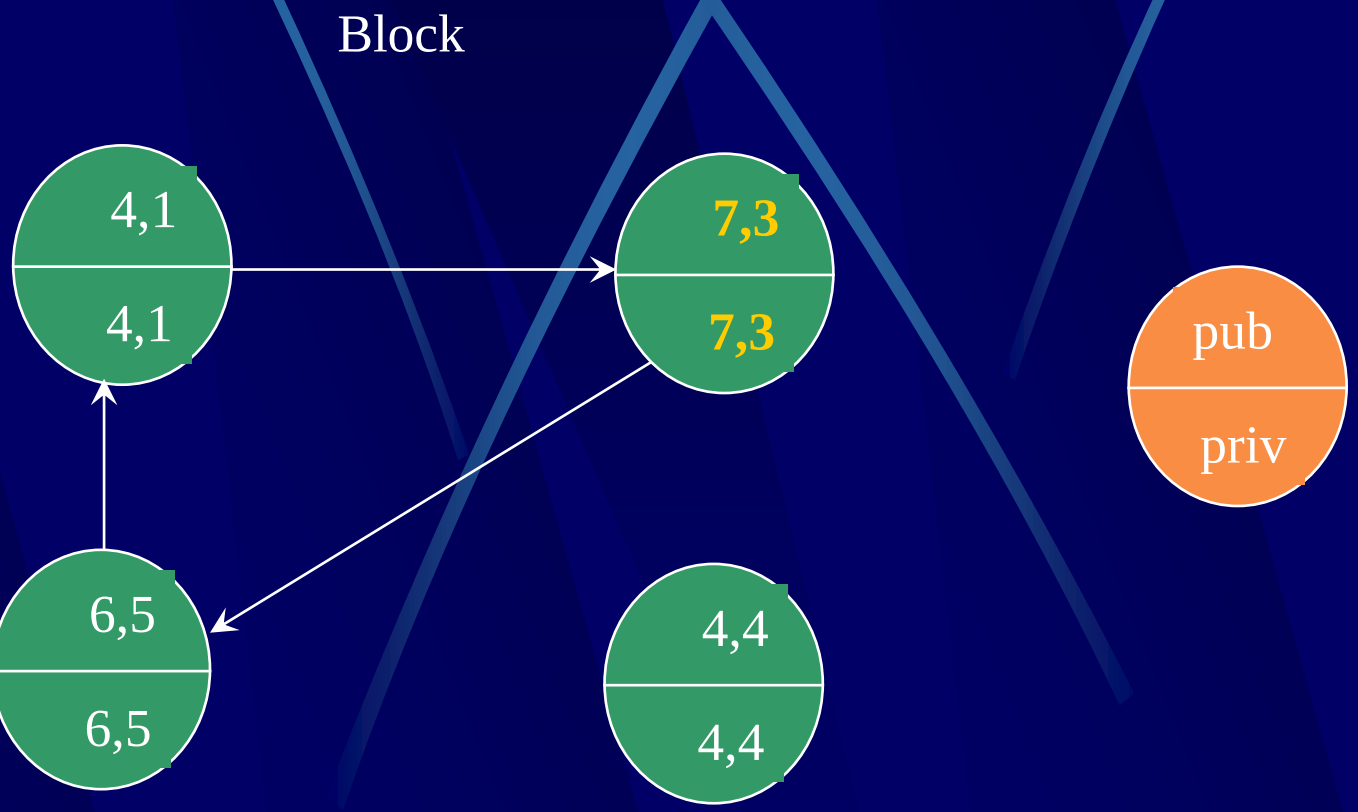
Example



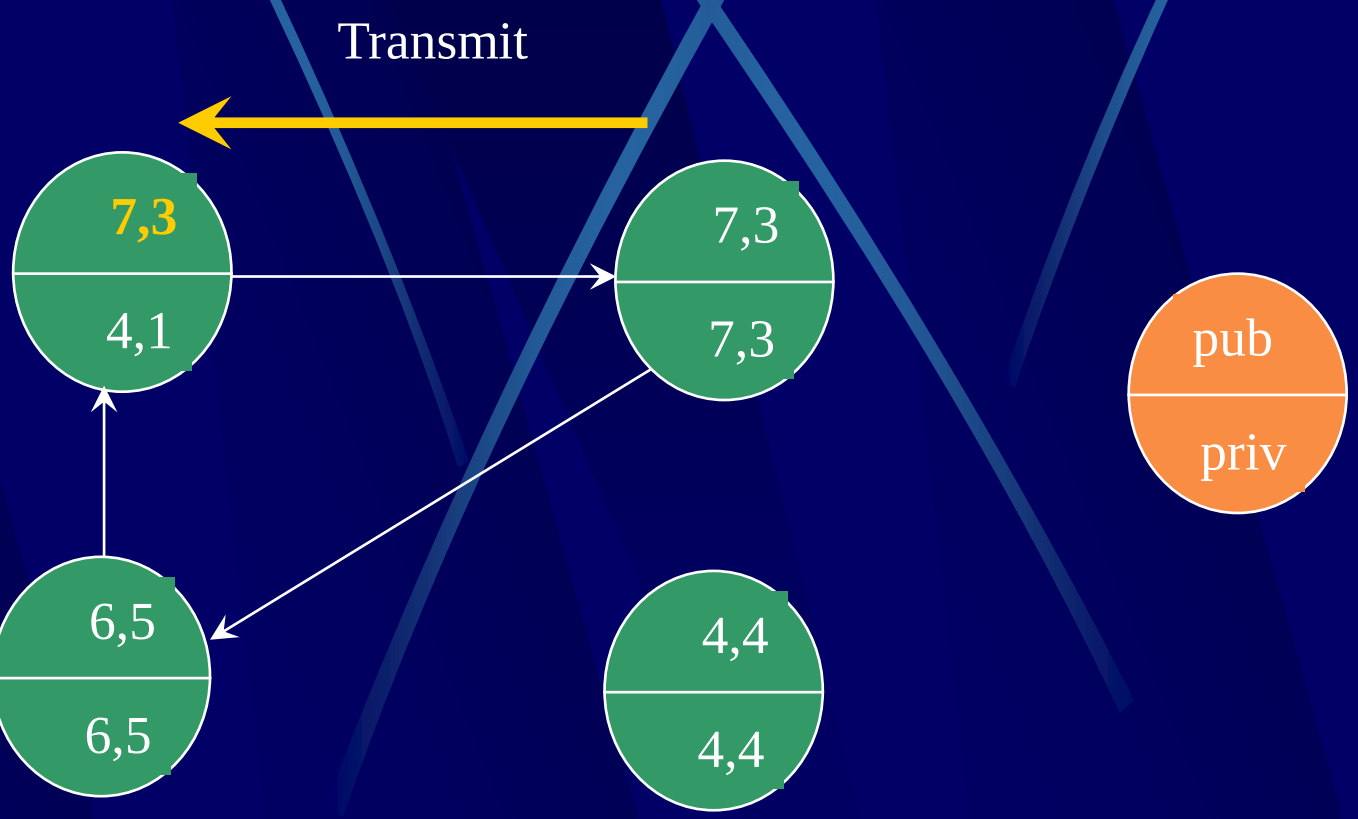
Example



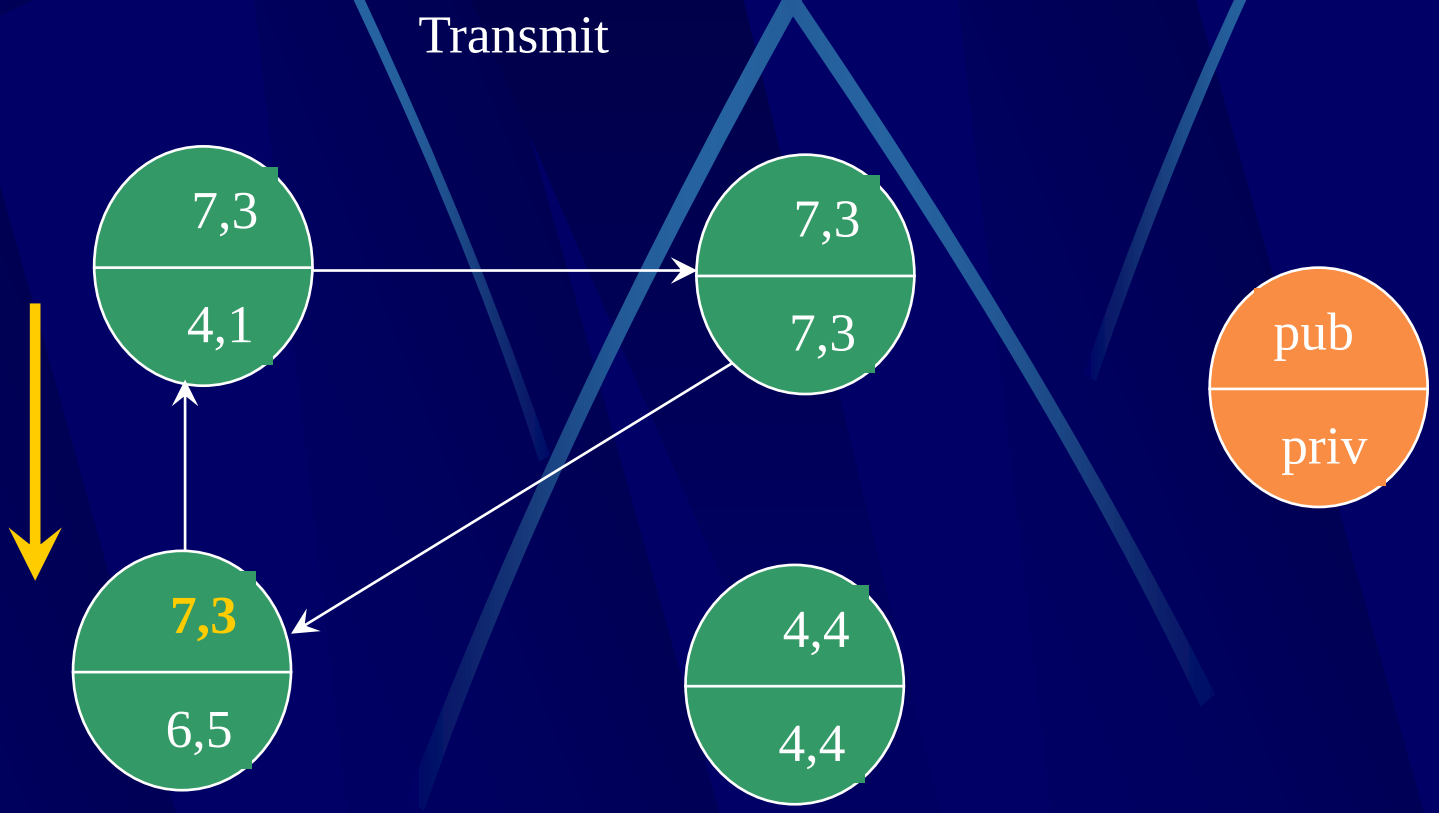
Example



Example

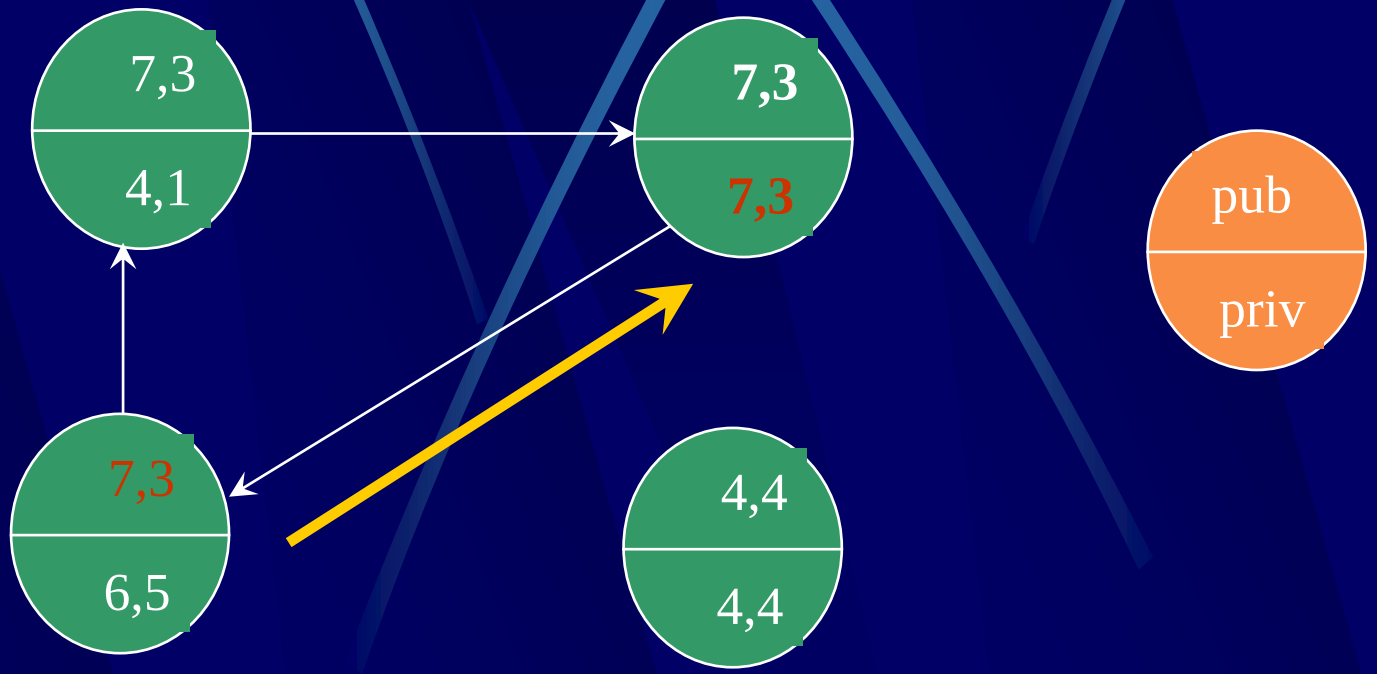


Example

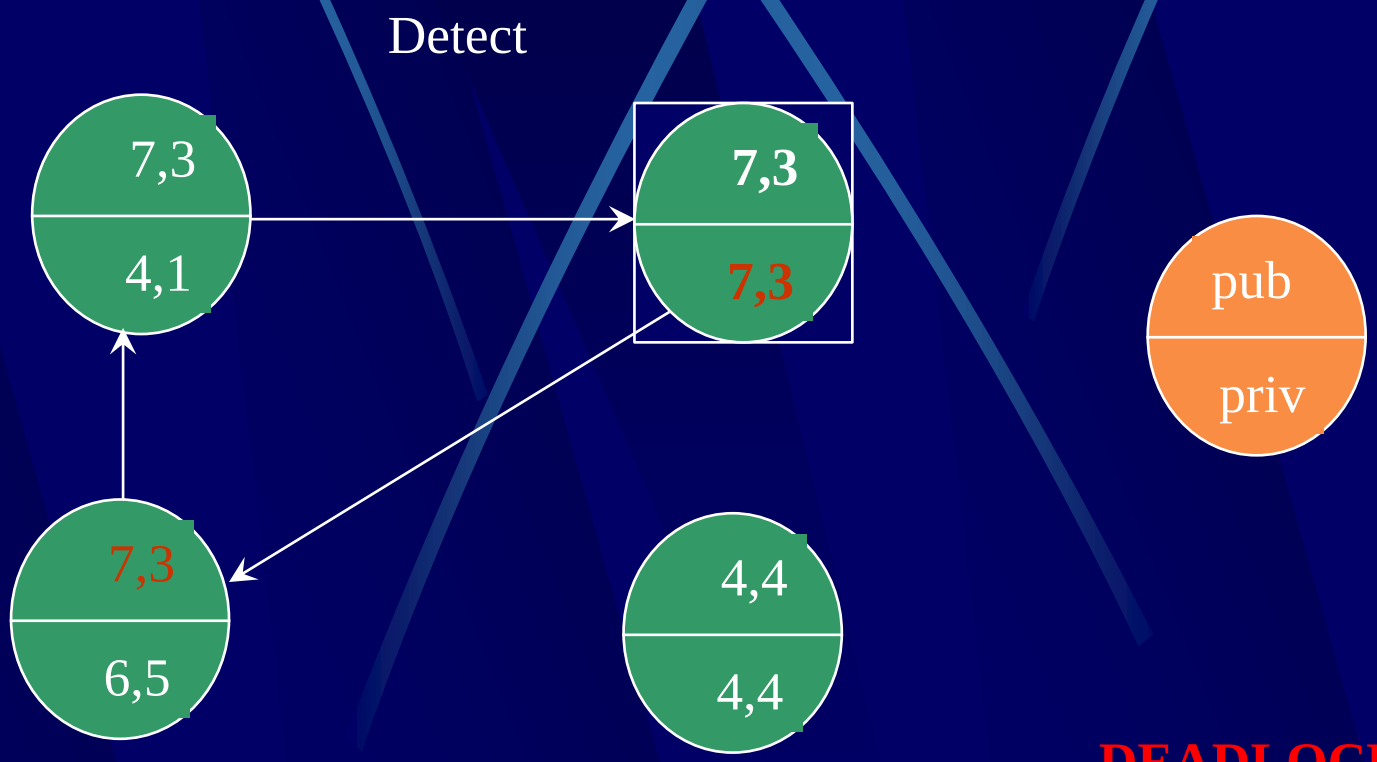


Example

Transmit



Example



Aborting low-priority transactions