Global State Collection

Global state collection

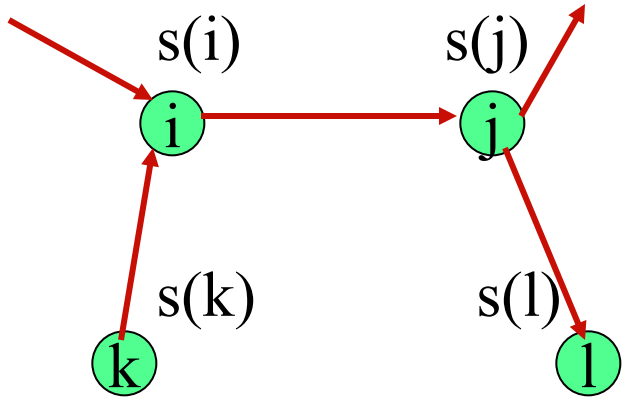
Some applications

- computing network topology
- termination detection
- deadlock detection

Chandy-Lamport algorithm does a partial job. Each process generates a ***fragment*** of the global state, but these pieces have to be "stitched together" to form a global state.

A simple exercise

Once the pieces of a consistent global state become available, consider collecting the global state via *all-to-all broadcast*



At the end, each process will compute a set V, where

$V = \{s(i): 0 \leq i \leq N-1\}$

All-to-all broadcast

Assume that the topology is a strongly connected graph

program broadcast (for process i}

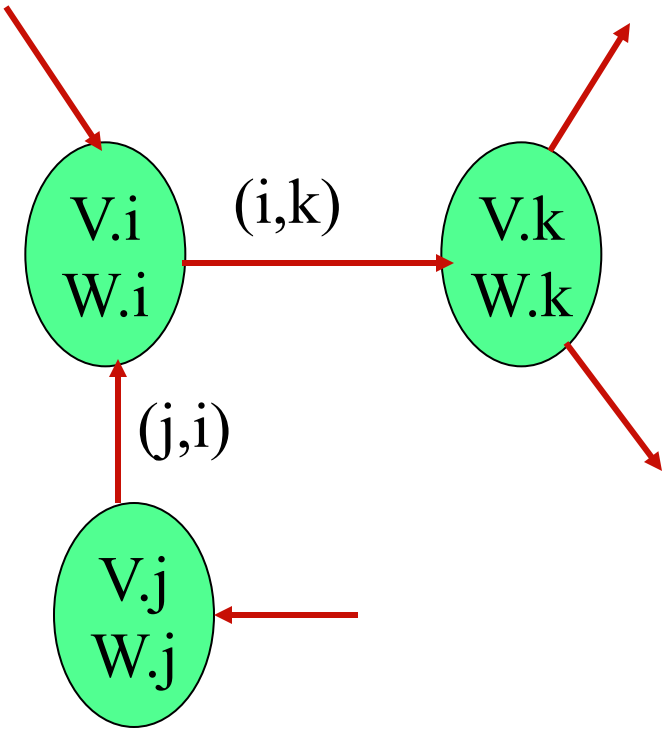
define V_i, W_i: **set of values**;

initially $V_i = \{s(i)\}$, $W_i = \emptyset$ {and every channel is empty}

1 **do** $V_i \neq W_i$ → send $(V_i \setminus W_i)$ to every outgoing channel;
$W_i := V_i$

2 [] $\neg empty(k,i)$ → receive X from channel (k,i);
$V_i := V_i \cup X$

od



Acts like a "pump"

Proof outline

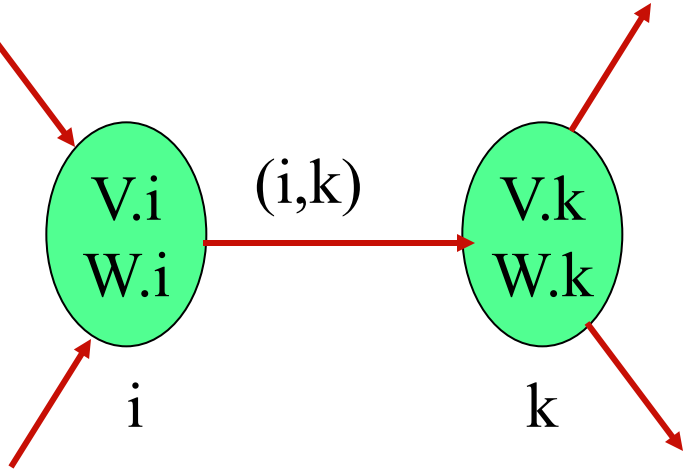
Lemma. empty (i, k) ⇒ W.i ⊆ V.k.

Use a simple inductive argument.

(Base case) W.i = null

(Induction hypothesis) Assume it is true now.

(Inductive step) Show that when (i,k) becomes empty next time, the condition holds. Easy!



(Upon termination) ∀**i**: **V.i = W.i,**
and all channels are empty. So, **V.i ⊆ V.k.**

In a strongly connected graph, every node is in a cycle On a cyclic path, **V.i = V.k** must be true. This means ∀**i**,,j: V.i=V.j Since **s(i) ∈V.i,** ∀**j: s(i) ∈V.j.**

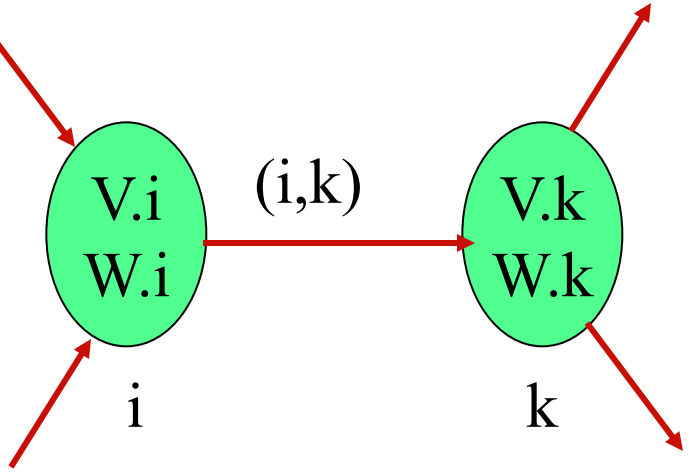
Proof outline

Lemma. The algorithm will terminate in a bounded number of steps.

Consider the variant function

$$Y = (V_0, V_1, V_2, \ldots, V_{n-1}, c_0, c_1, c_2, \ldots, c_{m-1})$$

Channel states



Statement 1 and Statement 2 increase Y lexicographically, until the V's reach their largest value {s(0), s(1), s(2), …, s(n-1)}.

So the algorithm terminates in a bounded number of steps.

Termination detection

During the progress of a distributed computation, processes may periodically turn **active** or **passive**.

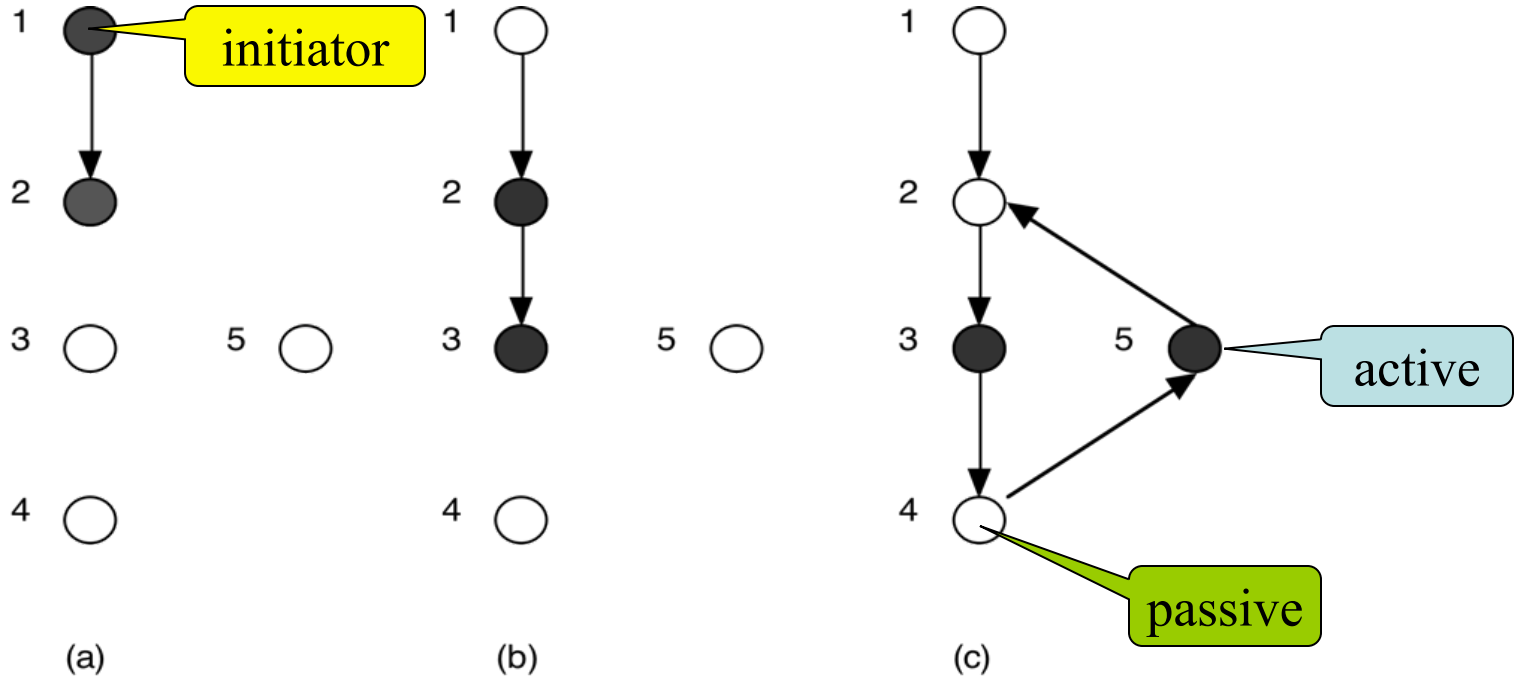
A distributed computation terminates when:

(a) every process is **passive**,

(b) all channels are **empty**, and

(c) the global state satisfies the **desired postcondition**

Visualizing diffusing computation



Notice how one process ***engages*** another process. Eventually all processes turn white, and no message is in transit -- this signals termination.
How to develop a *signaling mechanism* to detect termination?

Dijkstra-Scholten algorithm

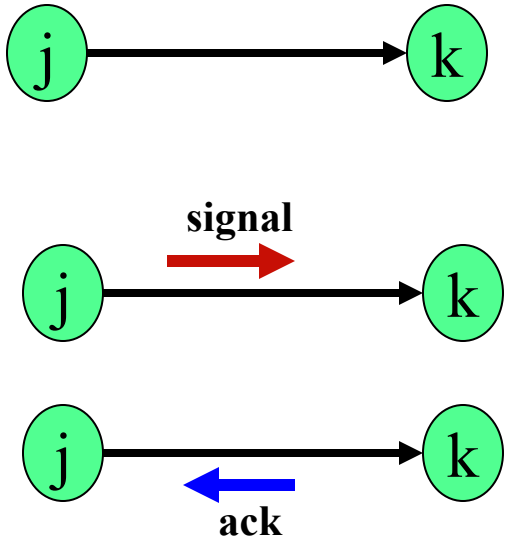
The basic scheme

An initiator initiates termination detection by sending signals (messages) down the edges via which it engages other nodes.

At a "suitable time," the recipient sends an ack back.

When the initiator receives ack from every node that it engaged, it detects termination.

Node j **engages** node k.



Dijkstra-Scholten algorithm

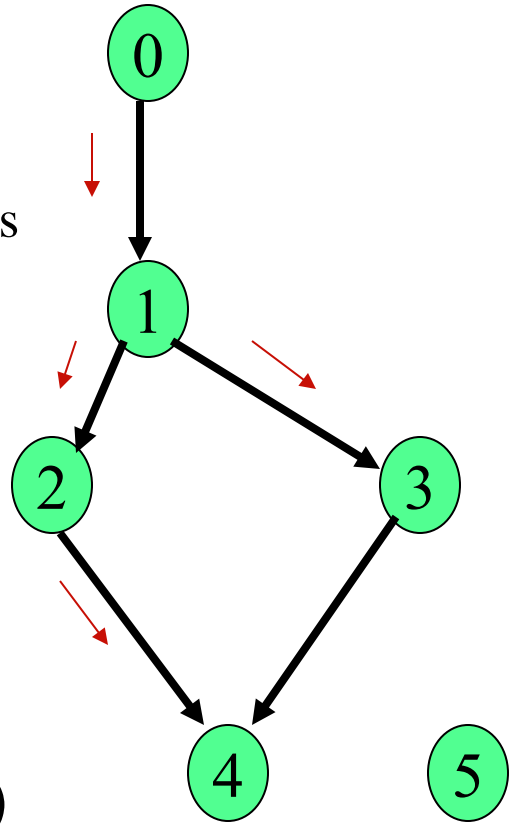
Deficit (e) = # of signals on edge e - # of acks on edge e

For any node, C = total deficit along ***incoming*** edges

and D = total deficit along ***outgoing*** edges edges

For the initiator, by definition, C = 0

Dijkstra-Scholten algorithm used the following two ***invariants*** to develop their algorithm:



Invariant 1. **(C ≥ 0) ∧ (D ≥ 0) (obvious, deficit ≥ 0)**

Invariant 2. **(C > 0) ∨ (D = 0) (proposed invariant)**

To be observed by the signaling scheme

Dijkstra-Scholten algorithm

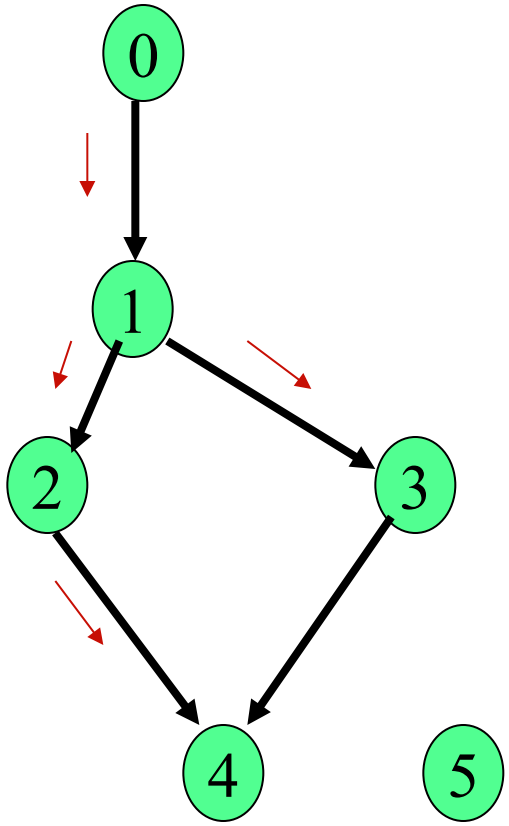
The invariants must hold when an interim node sends an ack.
So, acks will be sent when

$$(C\text{-}1 \geq 0) \wedge (C\text{-}1 > 0 \vee D = 0)$$

{follows from **INV1** and **INV2**}

$$= (C > 1) \vee (C \geq 1 \wedge D = 0)$$

$$= (C > 1) \vee (C = 1 \wedge D = 0)$$



Dijkstra-Scholten algorithm

program detect {for an internal node i}

define C, D : integer

m : (signal, ack) {represents the type of message received}

$state$: (active, passive)

initially $C = 0,\ D = 0,\ parent(i) = i$

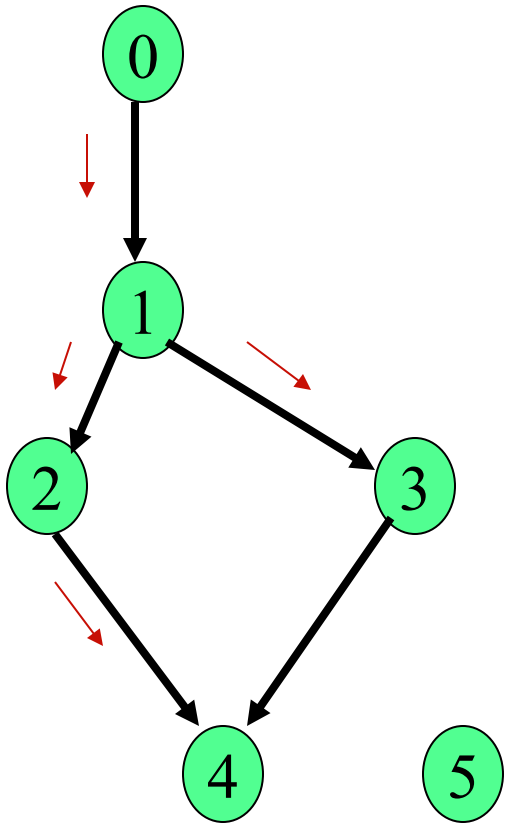
do $(m = \text{signal}) \wedge (C = 0) \rightarrow$ $C := 1$; $state :=$ active;
$parent := $ sender
{Read *Note 1*}

[] $m = \text{ack} \rightarrow$ $D := D - 1$

[] $(C = 1 \wedge D = 0) \wedge (state = \text{passive}) \rightarrow$ send ack to parent;
$C := 0$; $parent(i) = i$
{Read *Note 2*}

[] $(m = \text{signal}) \wedge (C = 1) \rightarrow$ send ack to the sender

od



Note 1. The node will forward signals to its successors (whenever they exist)
Note 2. The node now disappears from the computation graph.
The ***parent*** relation induces a spanning tree

Distributed deadlock

When each process waits for some other process (to do something), a deadlock occurs.

Assume each process owns a few resources. Review how resources are allocated, and how a deadlock is created.

Three criteria for the occurrence of deadlock

- Exclusive use of resources
- Non-preemptive scheduling
- Circular waiting by all (or a subset of) processes

Distributed deadlock

Three aspects of deadlock

- deadlock detection
- deadlock prevention
- deadlock recovery

Distributed deadlock

- May occur due to bad designs/bad strategy

  [Sometimes prevention is more expensive than detection and recovery. So designs may not care about deadlocks, particularly if it is rare.]

- Caused by failures or perturbations in the system

Distributed Deadlock Prevention uses pessimistic strategies

An example from banker's problem (Dijkstra)

A banker has $10,000. She approves a *credit line* of $6,000 to each of the three customers A, B, C. since the requirement of each is less than the available funds.

1. The customers can pay back any portion of their loans at any time. Note that no one is required to pay any part of the loan unless (s)he has borrowed up to the entire credit line.
2. However, after the customer has borrowed up to the entire credit line ($6000) (s)he must return the *entire money* is a finite time.
3. Now, assume that A, B, C borrowed $3000 each.

The state is unsafe since there is a "potential for deadlock." Why?

Banker's Problem

Questions for the banker

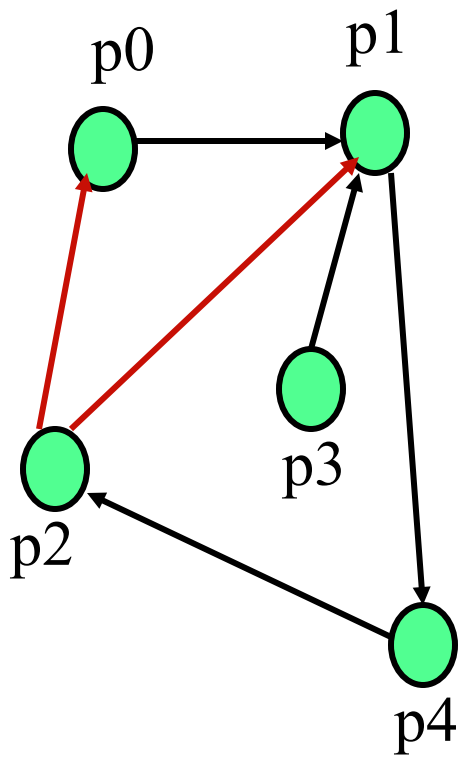
Let the current allocations be A = $2000, B = $2400, $C=$1800.

1. Now, if A asks for an additional $1500, then will the banker give the money immediately?

2. Instead, if B asks for $1500 then will the banker give the money immediately?

Wait-for Graph (WFG)

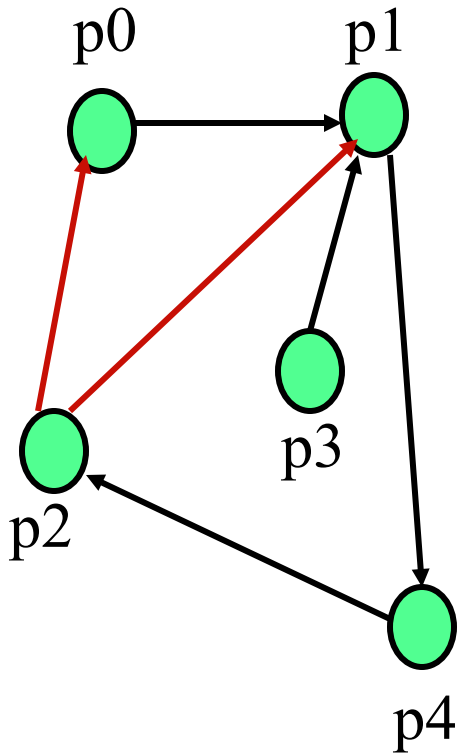
- Represents who waits for whom.
- No single process can see the WFG.
- Review how the WFG is formed.



Another classification

- **Resource deadlock**
  [R1 AND R2 AND R3 …]
  also known as AND deadlock

- **Communication deadlock**
  [R1 OR R2 OR R3 …]
  also known as OR deadlock



Detection of resource deadlock

Notations

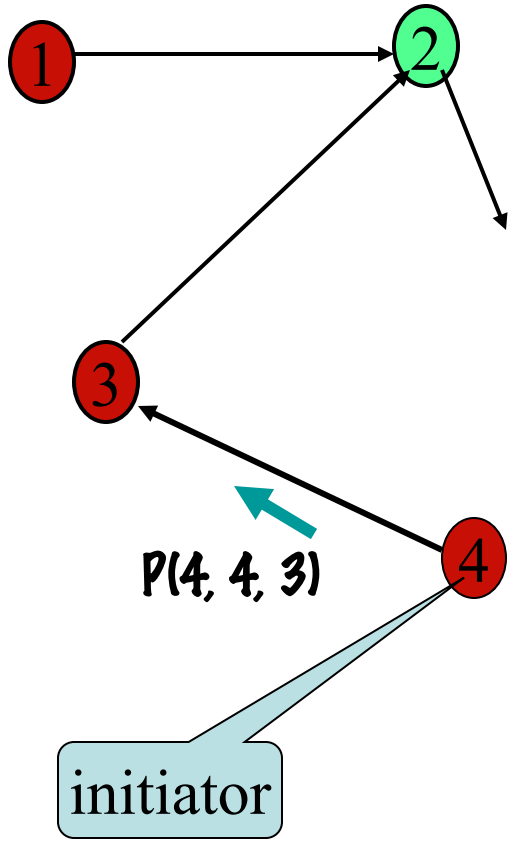
w(j) = true $\Rightarrow$ (**j** is waiting)

depend [j,i] = true $\Rightarrow j \in succ^n(i)$ (n>0)

(i's progress depends on j's progress)

P(i,s,k) is a probe

(i=initiator, s= sender, k3=receiver)



Detection of resource deadlock

{Program for process k}

do P(i,s,k) received ∧

w[k] ∧ (k ≠ i) ∧¬ depend[k, i] →

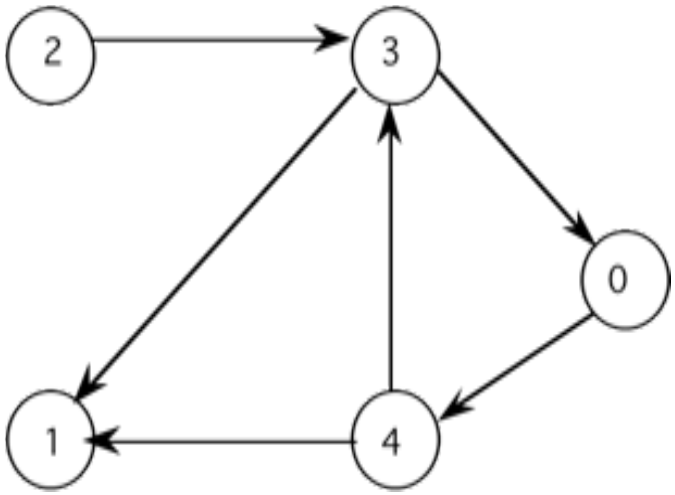
send P(i,k,j) to each successor j;

depend[k, i]:= true

[] P(i,s,k) received ∧ w[k] ∧ (k = i) →

process k is deadlocked

od

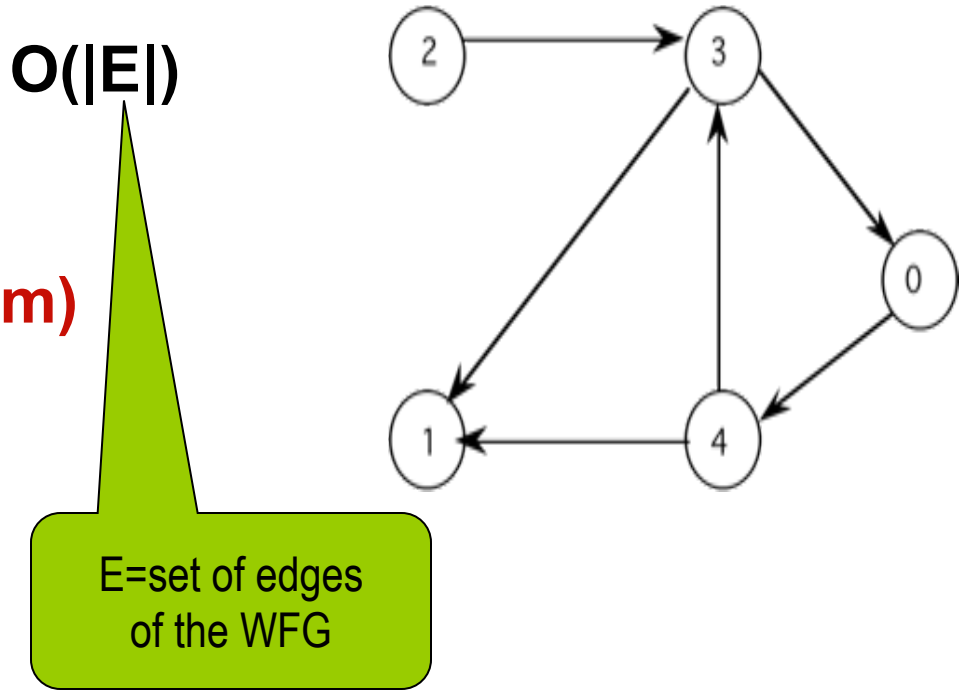


Observations

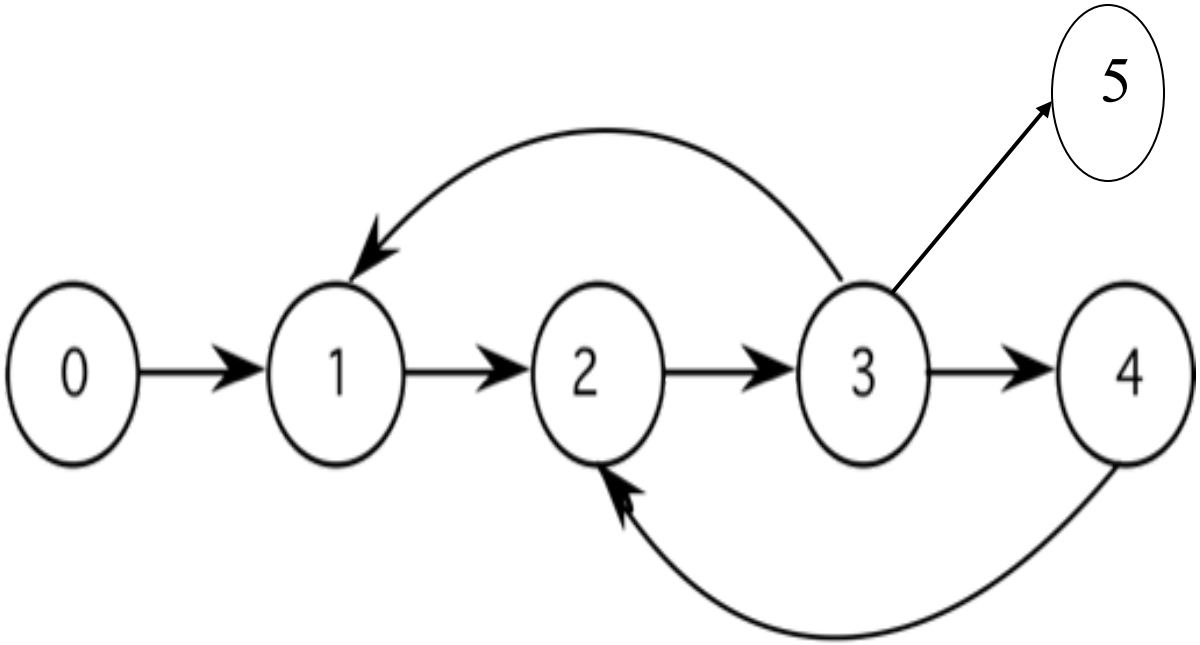
To detect deadlock, the initiator must be in a cycle

Message complexity = O(|E|)

(edge-chasing algorithm)



Communication deadlock



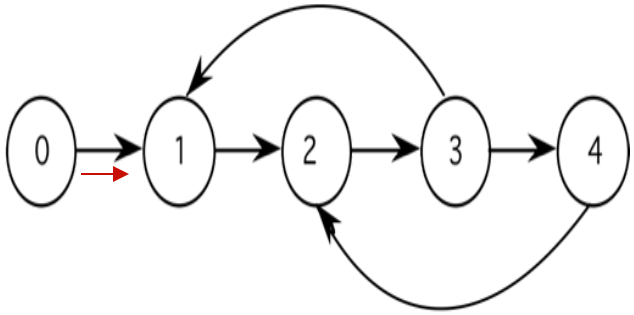
This WFG has a resource deadlock but no communication deadlock

Detection of communication deadlock

A process ignores a probe, if it is not waiting for any process. Otherwise,

- *first* probe →
  mark the sender as *parent*;
  forwards the probe to successors
- Not the first probe →
  Send ack to that sender
- ack received from every successor →
  send ack to the parent

Communication deadlock is detected if the initiator receives ack.



Has many similarities with Dijkstra-Scholten's termination detection algorithm