# Centralized Deadlock-Detection Algorithms

- The Ho-Ramamoorthy Algorithms: In class of path-pushing algorithms
- The Two-Phase Algorithm

  Deadlocks don't go away unless broken. Collect status twice. Take only those dependencies which are present in both. Any issues?

  Ho-Ramamoorthys Algorithm Link

- The One-phase Algorithm

  Uses process status table and resource status table . process status table contains resources that are locked or waited on for each process. resource status table contains processes using or waiting on each resource. Takes a dependency as valid only if both tables indicate it.

# Centralized Deadlock-Detection Algorithms

- The One-phase Algorithm

  process status table: the process' information where for each process gives information about resources held or seeked after.

  Resource status table: Processes to which the resource is allocated to or seeking it

Gather this from all sites and create WFG if tables gathered from both sites say the same information. That is site 1 says P waits for R which is on site 2. Site2 also says P requested for R

Can we have false deadlock?

# Distributed Edge-Chasing Algorithm

Chandy Misra Haas Algorithm

- A probe is sent by a blocked process to the process holding the resource.

- Probe is a triplet (I, J, k)

I: ID of blocked process

J: ID of process sending the message

K: ID to which message is sent

- If probe is received by a blocked process, it edits and forwards the probe to the processes holding the resource it requires
- If initial blocked process receives back its probe; it declares deadlock

# Distributed Edge-Chasing Algorithm

Chandy Misra Haas Algorithm

To determine if a blocked process is deadlocked

if $P_i$ is locally dependent on itself

then declare a deadlock

else for all $P_j$ and $P_k$ such that

(a) $P_i$ is locally dependent upon $P_j$, and

(b) $P_j$ is waiting on $P_k$, and

(c) $P_j$ and $P_k$ are on different sites,

send *probe (i, j, k)* to the home site of $P_k$

On the receipt of *probe (i, j, k),* the site takes the foll. actions:

if (a) $P_k$ is blocked, and

(b) *dependent$_k$(i)* is false, and //Pk knows of I's dependency on it.

(c) $P_k$ has not replied to all requests of $P_j$,

then begin

*dependent$_k$(i)* = true;

if k = i then declare that $P_i$ is deadlocked

else for all $P_m$ and $P_n$ such that

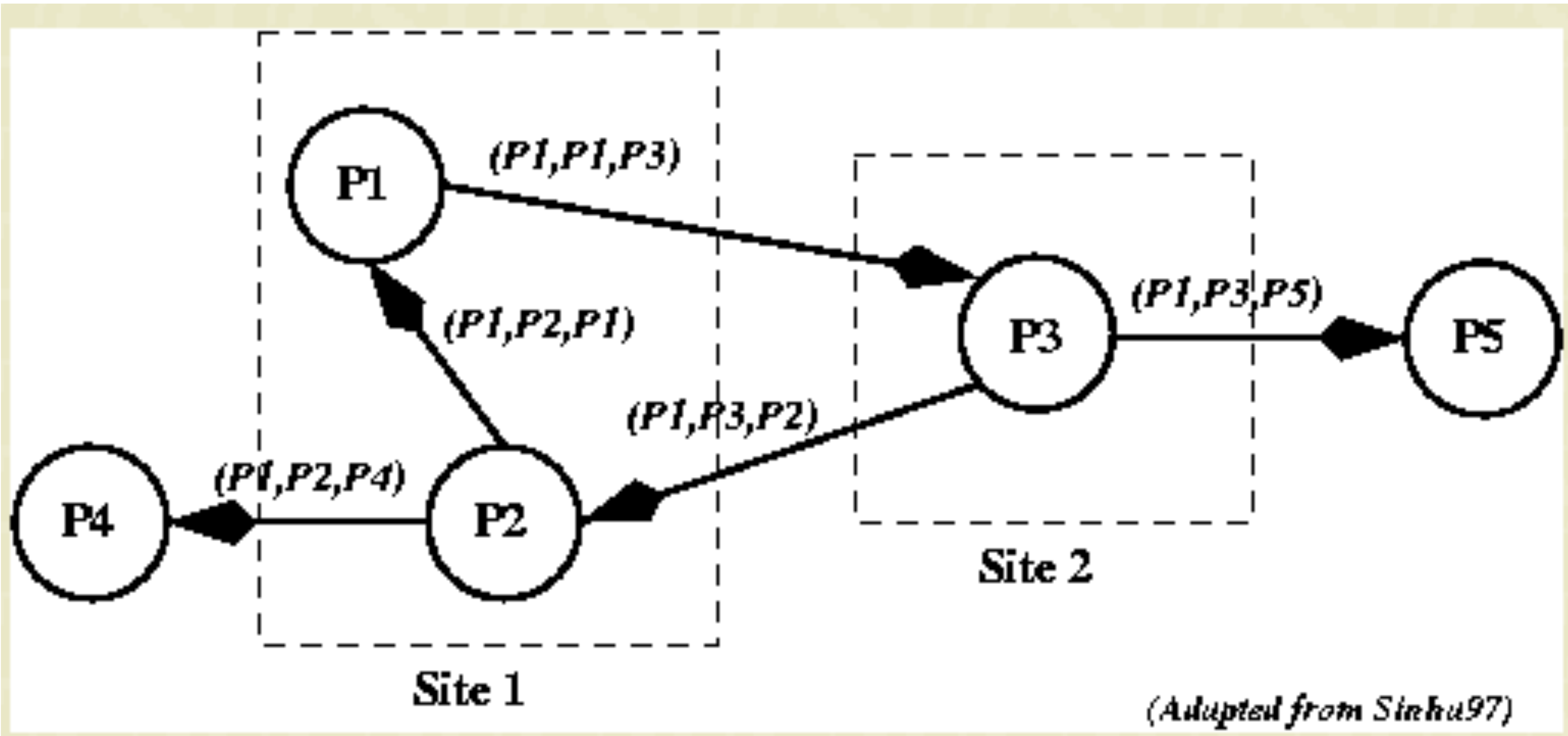
(i) $P_k$ is locally dependent upon $P_m$, and

(ii) $P_m$ is waiting on $P_n$, and

(iii) $P_m$ and $P_n$ are on different sites,

send *probe (i, m, n)* to the home site of $P_n$

end.


(P1,P1,P3)
P1
(P1,P2,P1)
P3
(P1,P3,P5)
P5
(P1,P3,P2)
P4
(P1,P2,P4)
P2
Site 2
Site 1
(Adapted from Sinha97)

# Homework

1. What happens if there is no deadlock?
2. How will Pi conclude that there is no deadlock?
3. This algorithm works for what kind of a model-- AND, OR , single resource request?