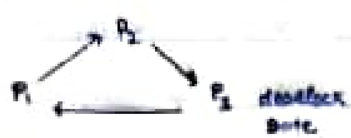
DEADLOCK DETECTION IN DISTRIBUTED SYSTEMS

consider a set of processes using shared resources. if none of them have all the resources they need to proceed and (thus) are waiting for each other to release some resource, deadlock occurs.

process P_i - running | blocked (busy)

$P_1 \longrightarrow P_2$
waiting
{waiting for resource held by P_2}



wait-for graph (WFG)

deadlock if directed cycle | knot in WFG

how to deal with it?

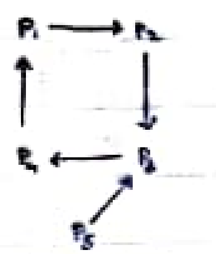
- prevention (proper driving)
  processes acquire all needed resources at once. or, pre-empt process holding the resource.
- avoidance (clear mind)
  resource is granted to a process if resulting global state is safe.
- detection (accidents)
  check if deadlock occurred.
  resolve: abort a deadlocked process

(best approach in distributed systems)

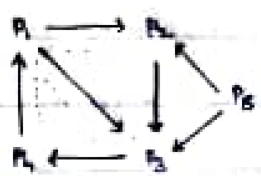
correctness criteria: (of detection algorithm)

- progress: no undetected deadlocks
- safety: no false (phantom) deadlocks

(difficult due to outdated / inconsistent info)



single resource:
- each process has at most one outstanding request
(cycle → deadlock)

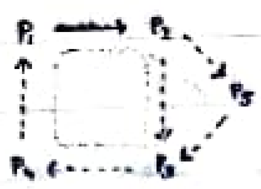


AND:
- each process can have more than one requests. all necessary to proceed.
(cycle → deadlock)

«resource models»

$\binom{q}{q}$ → AND $\quad \binom{1}{q}$ → OR

$\binom{p}{q}$:
another form of AND-OR model. more compact request.



OR:
- each process can have more than one requests.
only one enough to proceed.
(knot → deadlock)
(if $P_5 \to P_3$ didn't exist, there would be no knot)

AND-OR:
- requests can be of the form $x \wedge (y \vee z)$
(no familiar WFG construct)
repeated OR model application / knot detection is a way to detect deadlock.

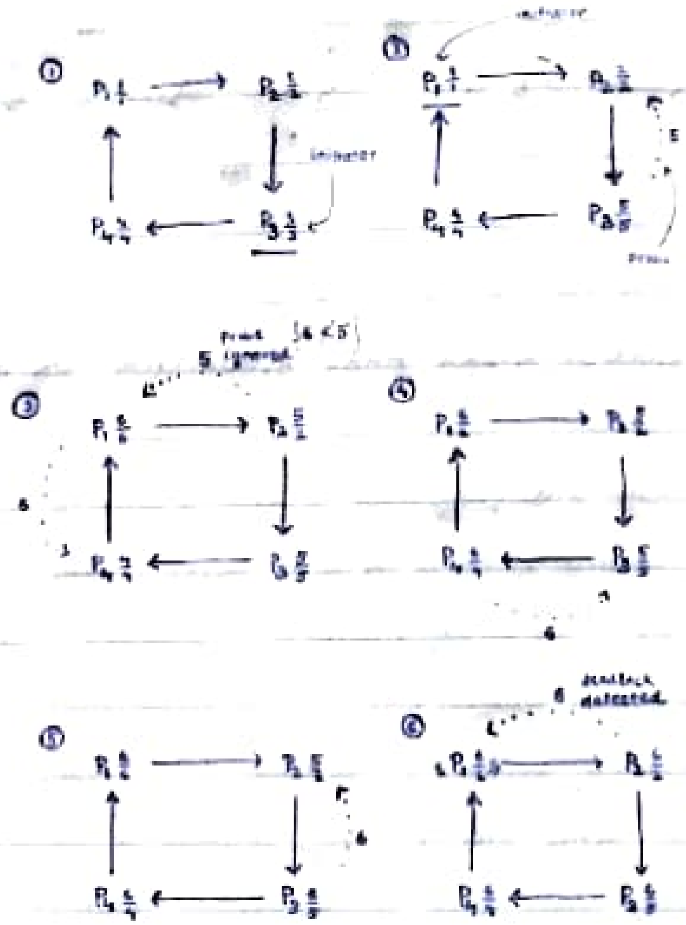
unrestricted:
no assumptions made regarding underlying structure of requests.
(high overhead)

MITCHELL AND MERITT ALGORITHM - SINGLE RESOURCE MODEL
(edge chasing)

initiator sends probe in opposite direction.

probe comes back to it ⇒ deadlock.

each process → $P_i \frac{u}{v}$ ← public label; v ← private label - unique



activate → resource got from P_j

$P_i \frac{u}{v} \longrightarrow P_j \frac{w}{x} \Rightarrow P_i \frac{u}{v} \quad P_j \frac{w}{x}$

block → P_i creates probe (as its blocked)

$P_i \frac{u}{v} \longrightarrow P_j \frac{w}{x} \Rightarrow P_i \frac{z}{z} \longrightarrow P_j \frac{w}{x}$ { z = inc(u, w); $z > u, w$; z - unique }

transmit → move probe in opp direction

$P_i \frac{u}{v} \longrightarrow P_j \frac{z}{x} \Rightarrow P_i \frac{z}{v} \longrightarrow P_j \frac{z}{x}$

detect → probe reached initiator

$P_i \frac{z}{z} \longrightarrow P_j \frac{z}{x} \Rightarrow \langle P_i \frac{z}{z} \rangle \longrightarrow P_j \frac{z}{x}$

- only 1 process in a cycle detects deadlock — msg complexity - $O(n(n-1)/2)$

(simple deadlock resolution - abort self) (priority may be used for improvement)

- only genuine deadlocks will be detected in absence of spontaneous aborts

CHANDY - MISRA - HAAS ALGORITHM - AND MODEL
(edge chasing)

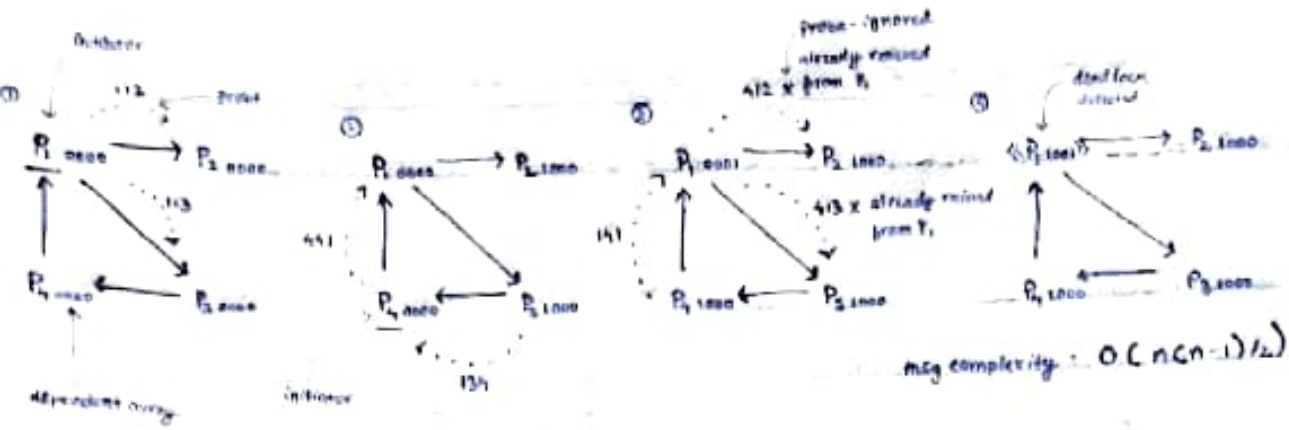
initiator sends probe in forward direction. — probe: $i \; j \; k$ (initiator, from, to)

probe comes back to it ⇒ deadlock.

each process → P_i 00...0 — dependent$_i$ (boolean) (true ⇒ is dependent on P_i)

$P_i \rightarrow P_j \rightarrow P_k \rightarrow P_l$

P_i dependent on P_j, P_k, P_l



CHANDY-MISRA-HAAS ALGORITHM - OR MODEL

(diffusion computation)

initiator sends query probe in forward direction. → query (i j k)

each process maintains the no. of probes it sends → num (i)

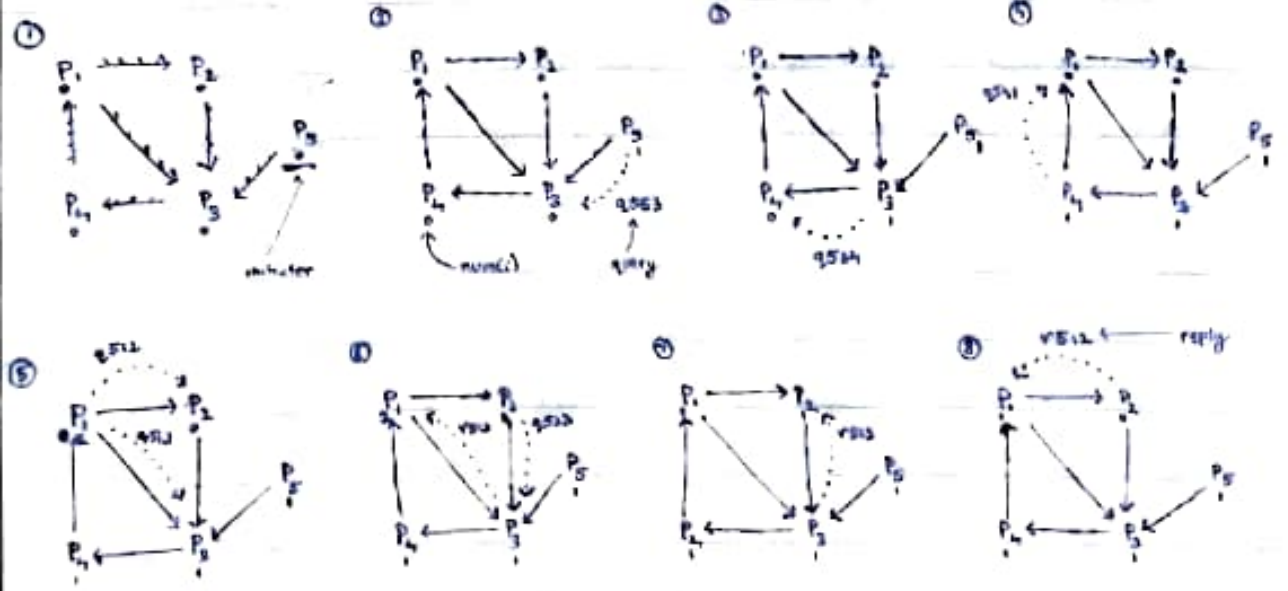
if process has already received a query (probe) it sends a reply → reply (i j k)

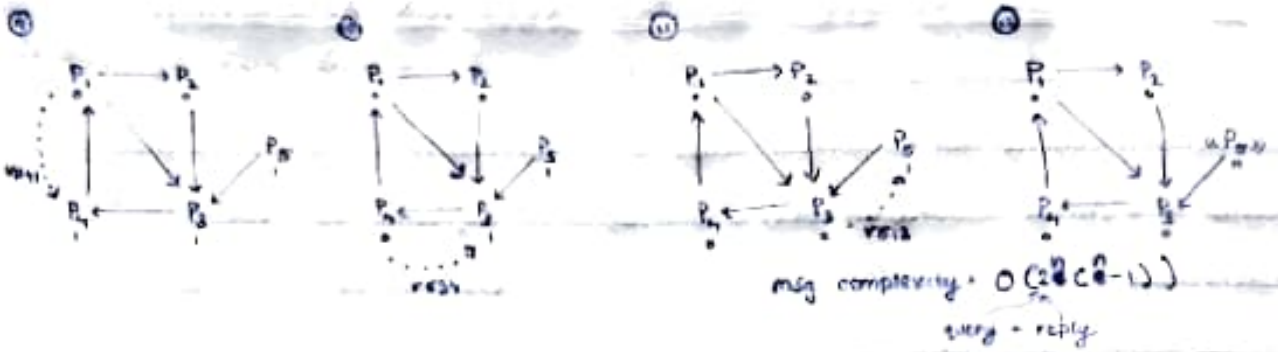
if process receives a reply (ijk) it decrements num (i)

if num(i) becomes 0, it sends back reply, as long as it was blocked since 1st query.

any active process simply discards a query or reply.

reply comes back to initiator → deadlock.





KNAPP'S CLASSIFICATION OF DEADLOCK DETECTION ALGORITHMS

- **path-pushing**

  Each site sends its local WFG to neighbouring sites. procedure repeated until one site has sufficiently complete picture of global state to detect deadlock.

- **edge-chasing**

  Each process sends probe along edges of WFG, and presence of cycle / knot is detected if matching probe is recieved.

- **diffusing computation-based**

  Processes make use of echo algorithms to detect deadlocks. query and reply messages are sent along edges. The initiator detects a deadlock when it has recieved a reply for every query it sent.

- **global state detection-based**

  deadlock can be detected by taking a snapshot of the system, and examining it.