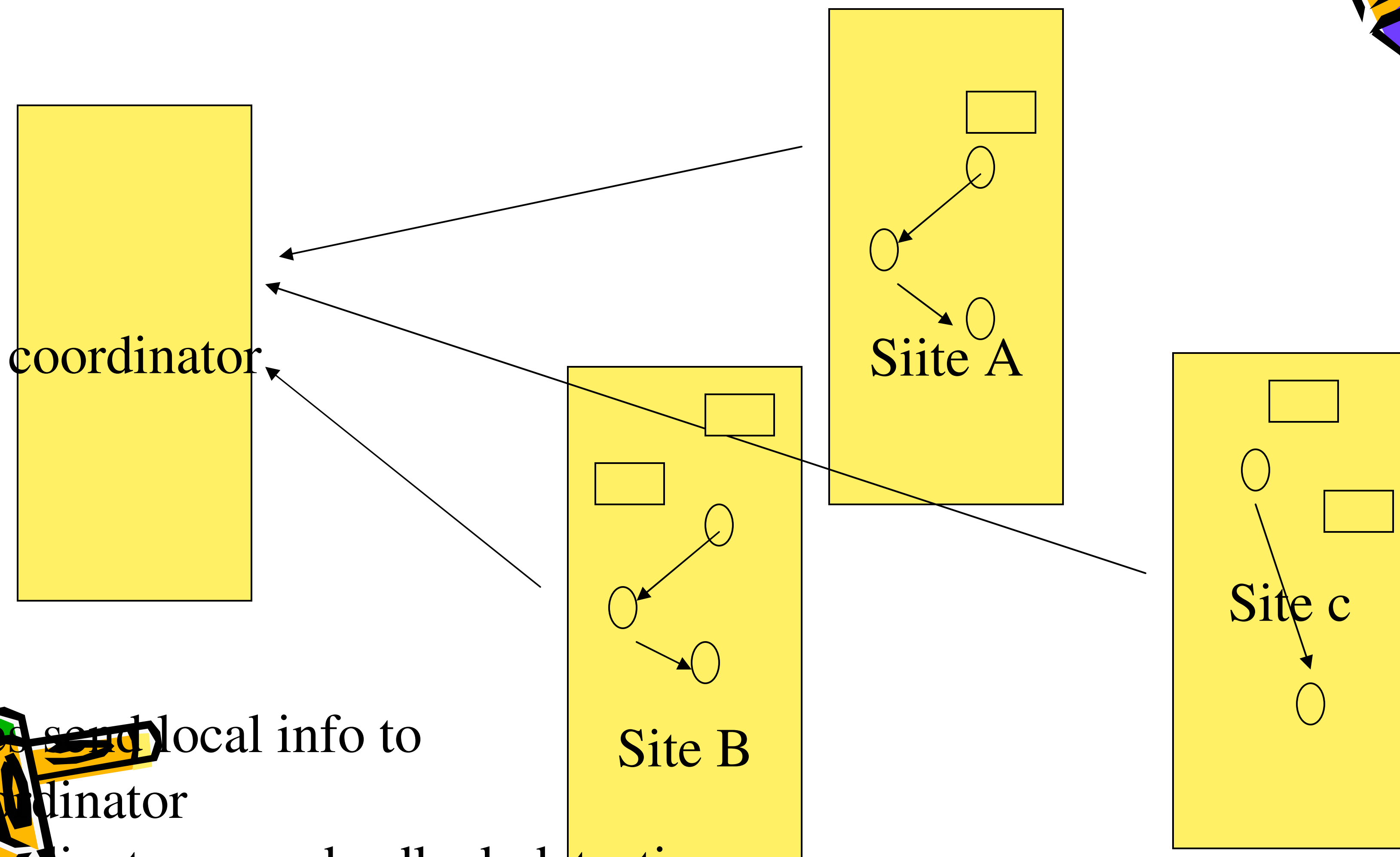
Distributed Deadlock detection

CS 451 offering – 2003-2004

Prof. R.K. Joshi
Dept of Computer Science and Engineering
IIT Bombay

Central coordinator based



Sites send local info to Coordinator

Coordinator runs deadlock detection

The Alternatives

- What info should be sent?
- When?
- Who initiates?

Event Echo

- What: *Every event* echoed to coordinator
  - Request
  - Allocation
  - release
- When: when event arises
- Who initiates: participants/sites
  - Request: sender
  - Allocation: resource site
  - Release: resource user

Release first and then echo

- Coordinator may see 2 allocations of a resource
  - Allocation echoed before release echo is recd by coordinator
  - Coordinator can tolerate boundary error (based no. of instances of each resource)

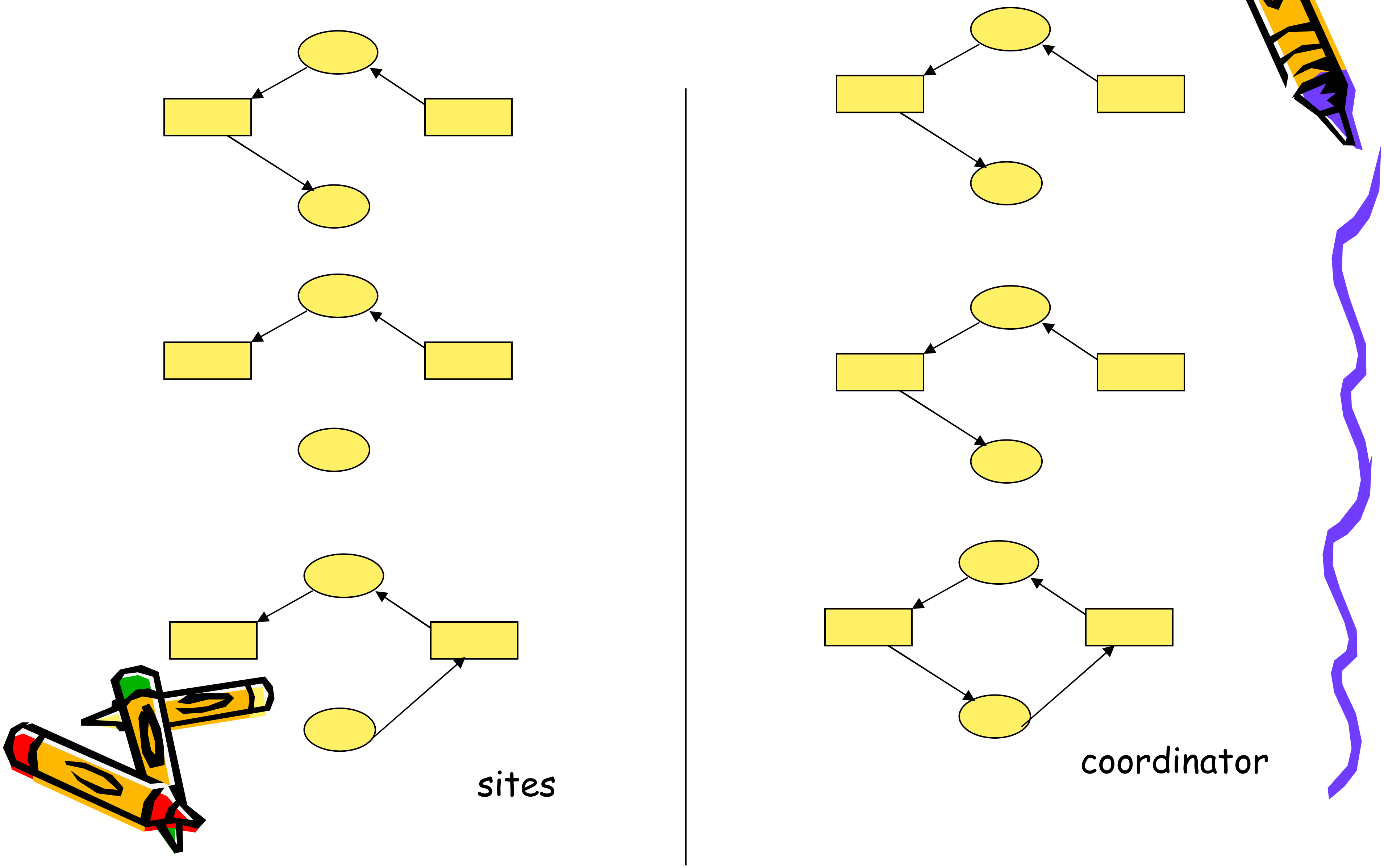
Our model - 1

- Resource site communicates to coordinator:
  - Request edge (blocked)
  - Allocation edge
- Process site communicates to coordinator
  - Release before sending it to the resource site

Our model - 2

- Resource site communicates to coordinator:
  - Request edge (blocked)
  - Allocation edge
  - Release

False deadlock



Model 3

- Processes echo
  - Allocated edge
  - Release edge
  - Requesting edge
- Resources echo
  - Allocated edge
  - Release edge
  - Blocked request

Model 4

- Resources echo
  - Allocated edge
  - Release edge
  - Blocked request
- Processes echo: release
  - And wait for an ack from coordinator

2 Phase model

- Model 2 + Model 2
  - On request of coordinator

2 Phase model with sequence ids

- Model 2 + Model 2
  - On request of coordinator
- Every site keeps a sequence number associated with every event
  - - associate with events
  - Keep a event count on the site

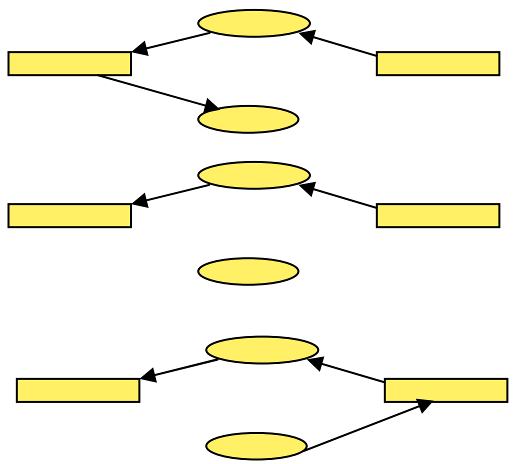
2 Phase model with event count

- If events occurred in phase 2 and phase1 reports a deadlock --> no deadlock in phase 1
- Take only those processes on which no new events are reported in phase 2

2 phase model

2. Coordinator asks R1

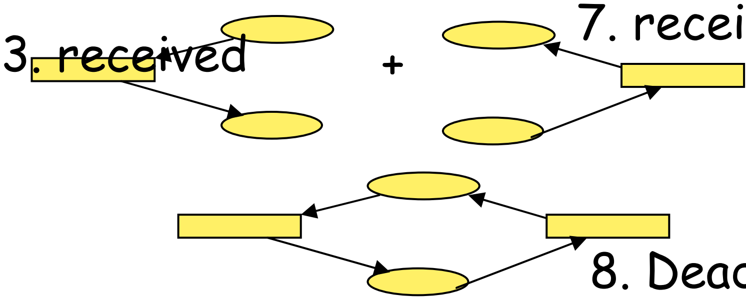
6. Coordinator asks R2



1

4

5



3. received

\+

7. received

8. Deadlock reported

Withdraw, and

Every thing repeats all over → false deadlock

sites

coordinator

A coordinated detection algorithm

- Resource sites communicate local resource status table
- Process sites communicate local process status table

- Coordinator asks for local grpahs
- Considers an entry if it's present in both resource table and corresponding process table
- Inconsistency is eliminated
- Use unique sequence number stamps for edges

Any other ideas?

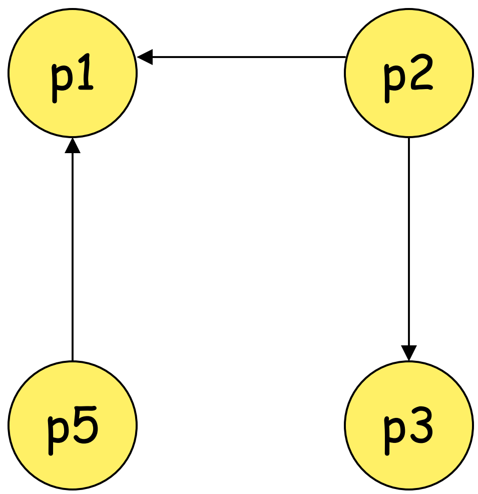
Fully Distributed deadlock detection

- If there is a deadlock, at least one site sees a cycle in its local graph

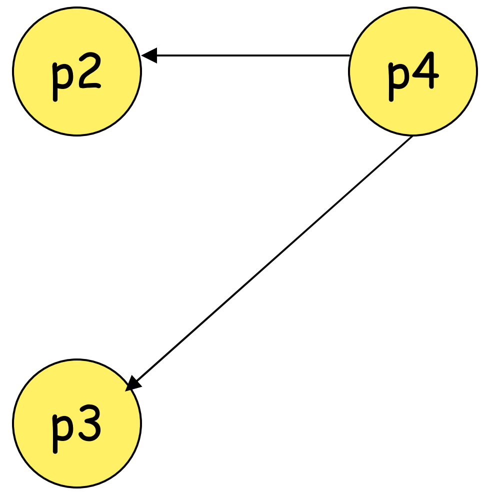
- Each site has one additional node Pex
- Pi → Pex exists if Pi is waiting for data in another site held by any other process

- Pex → Pj exists if there exists a process at another site that is waiting to acquire a resource held by Pj

example



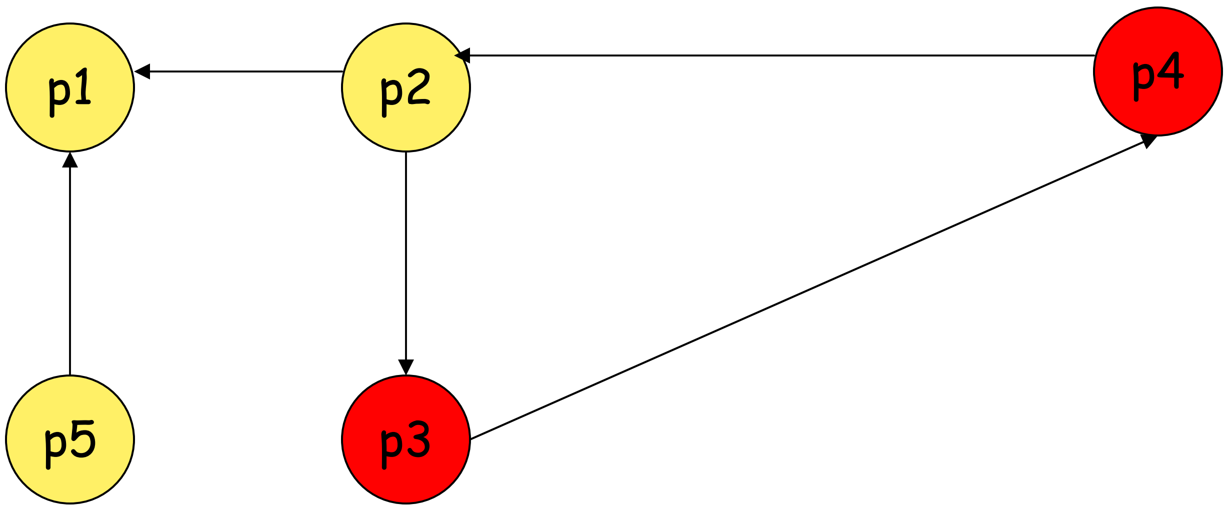
Site 1



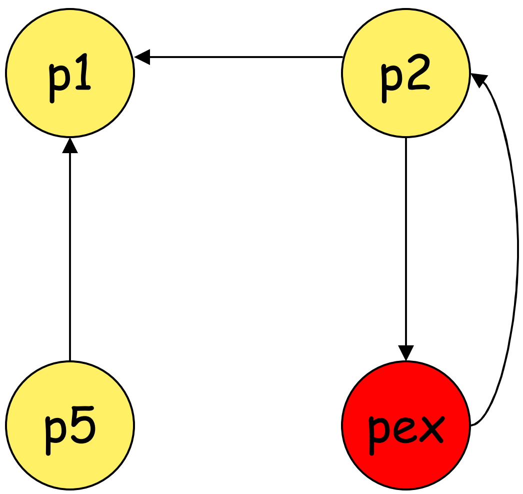
Site 2

No deadlock

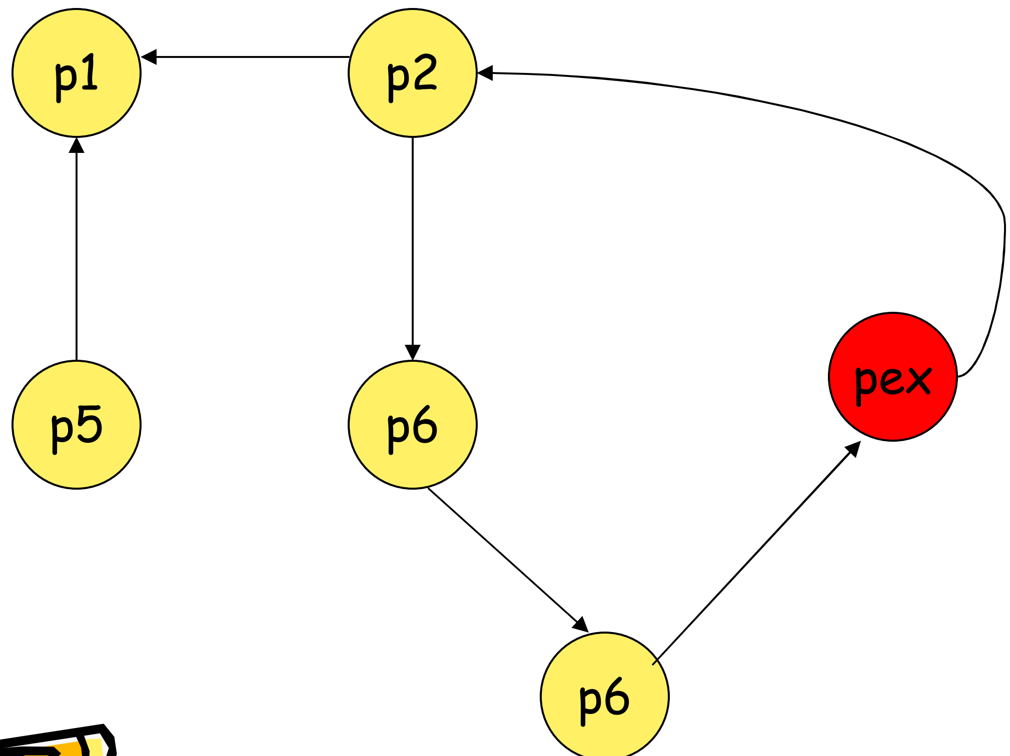
example



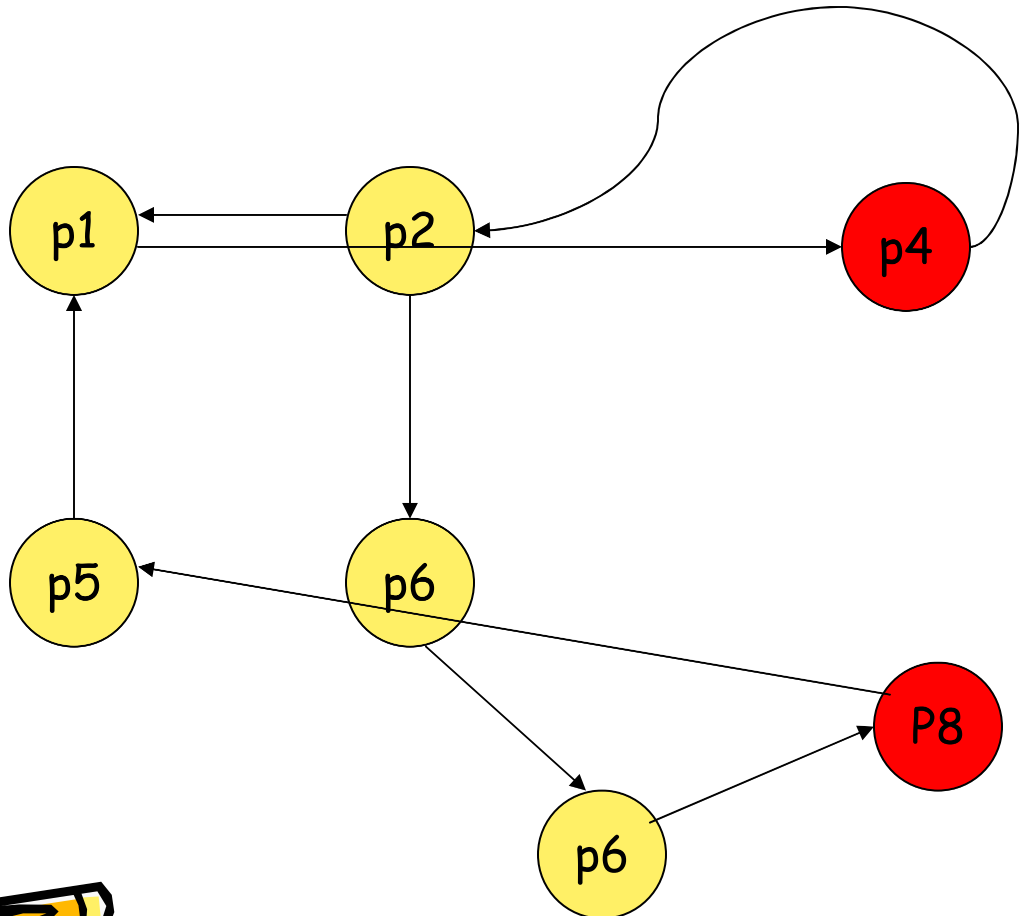
Collapse the external world



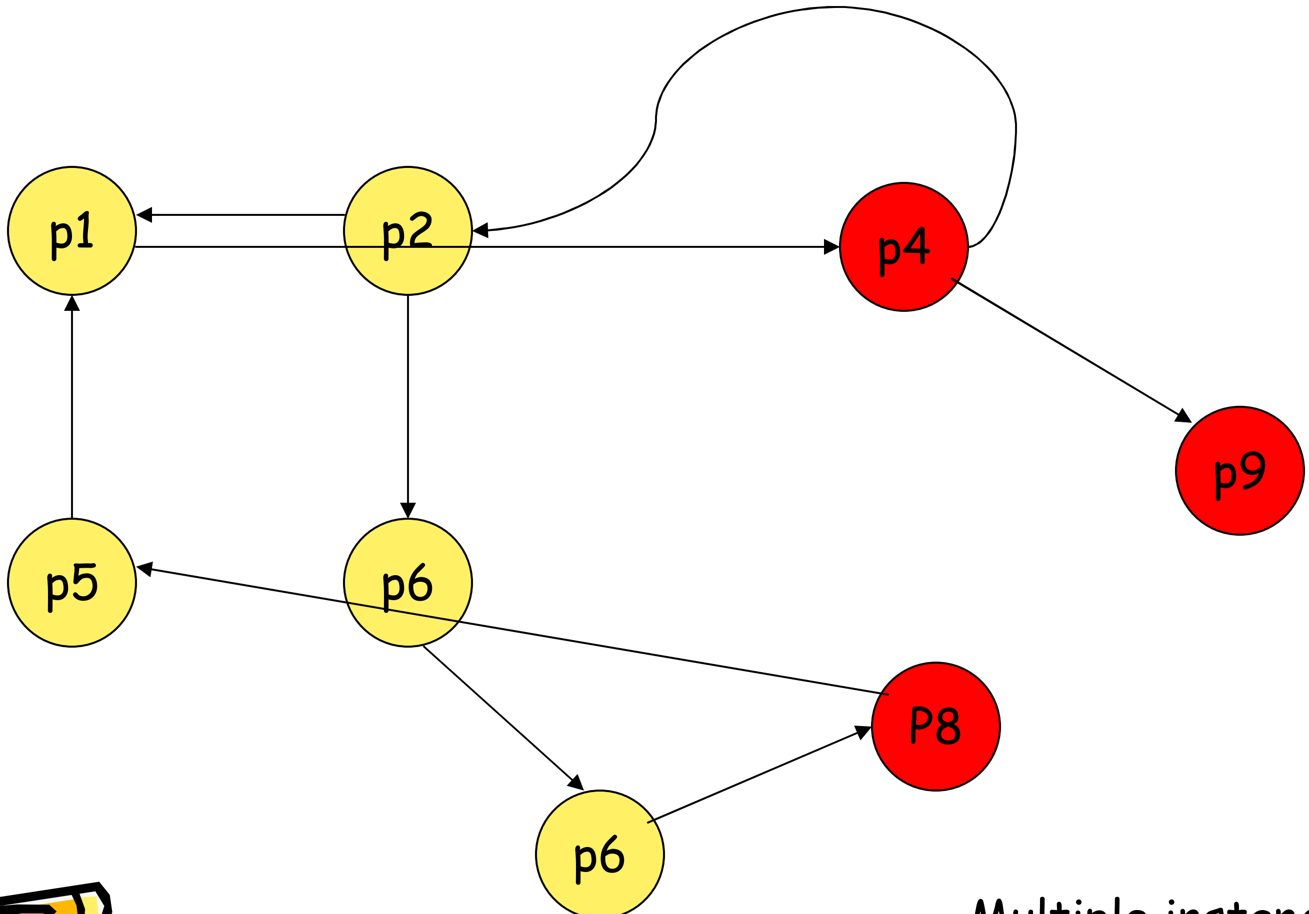
Collapse the external world - another example



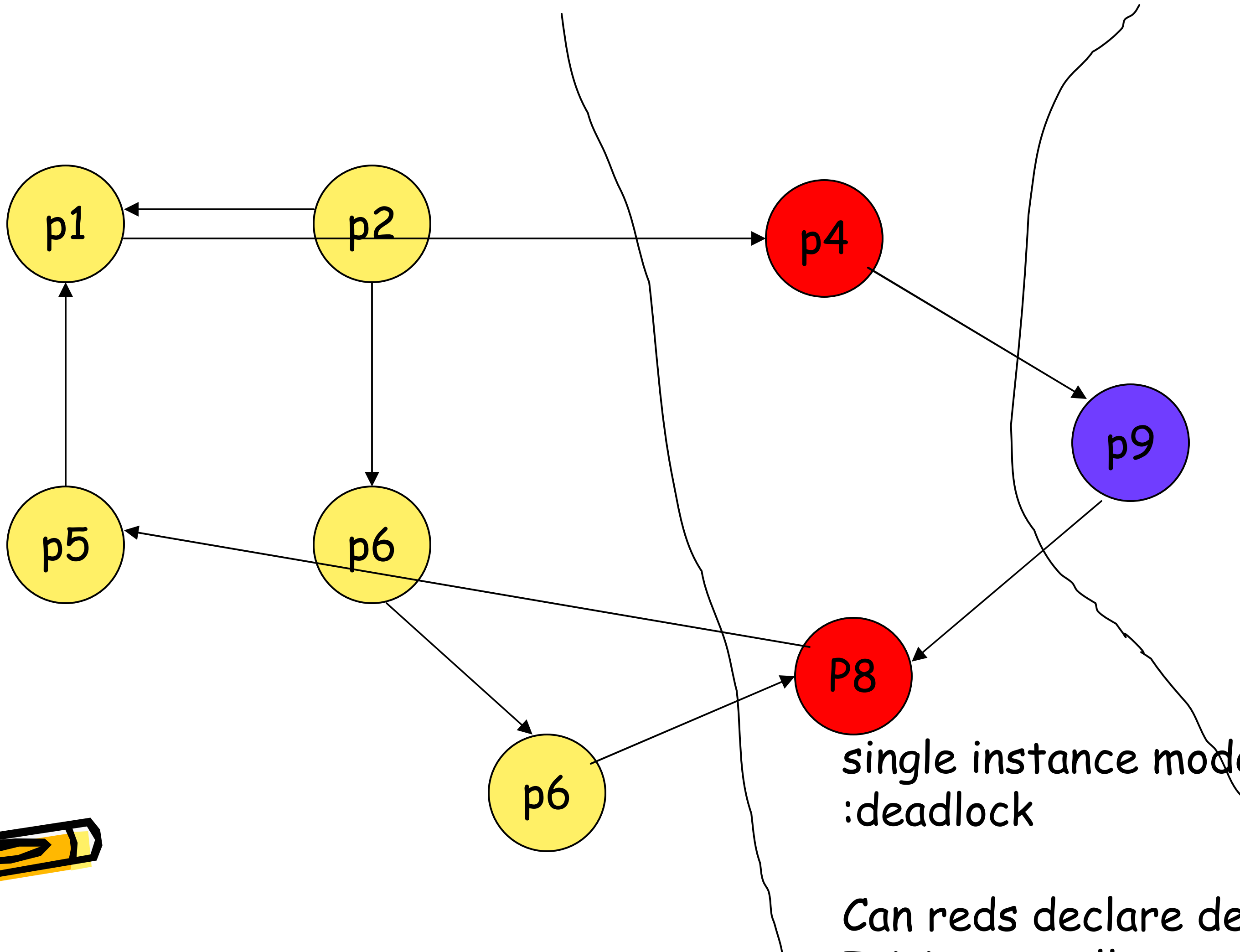
- If you see a local deadlock (cycle/knot) involving only local nodes → system deadlock
- Can you report a deadlock on a locally visible cycle/knot involving external nodes?
  - Yes provided that external resources are single instance resources



Deadlock if p4 and p8 are single instances

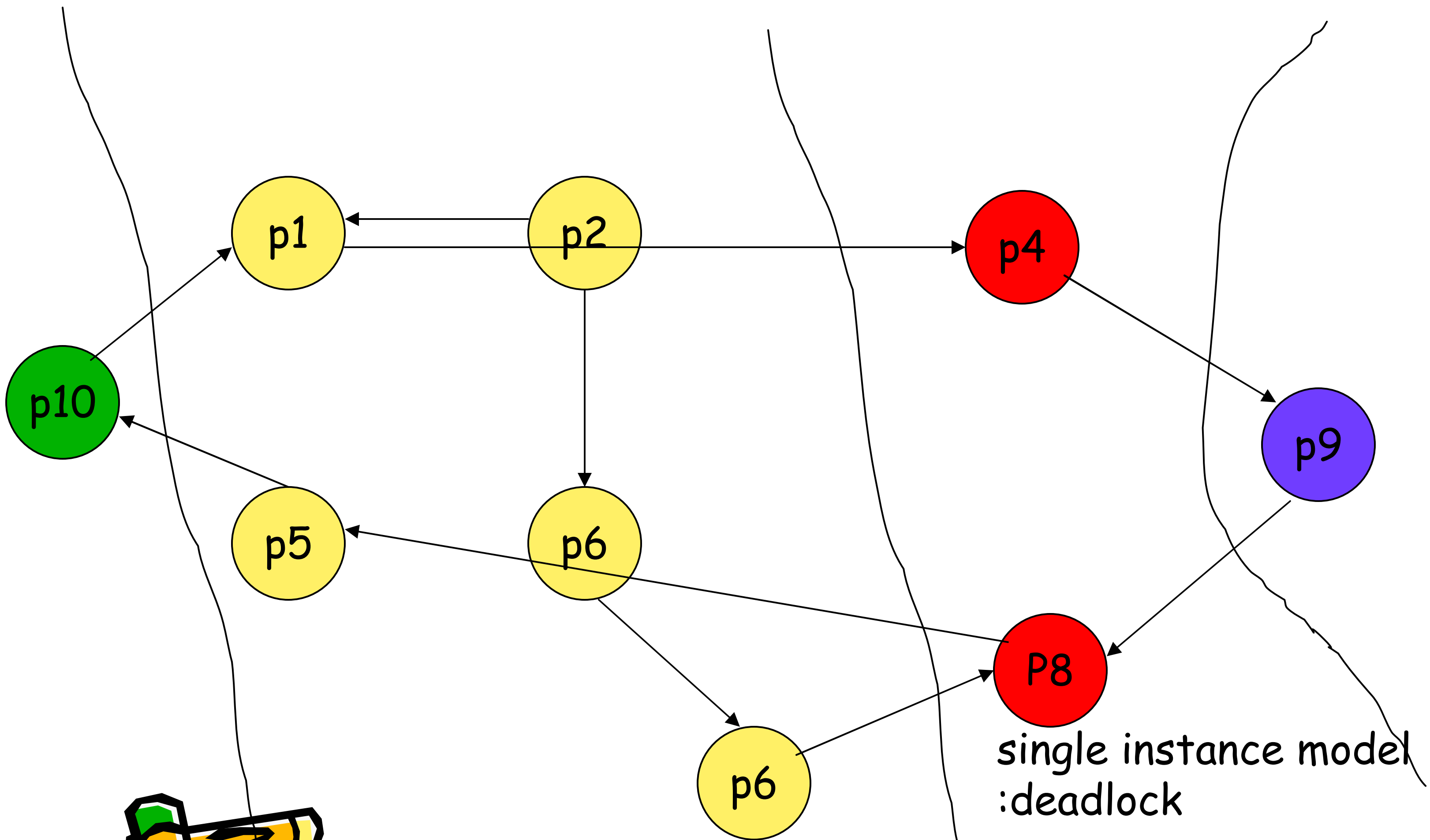


Multiple instance model
No deadlock



single instance model
:deadlock

Can reds declare deadlock?
Initiator: yellows
Red recs. From yellows

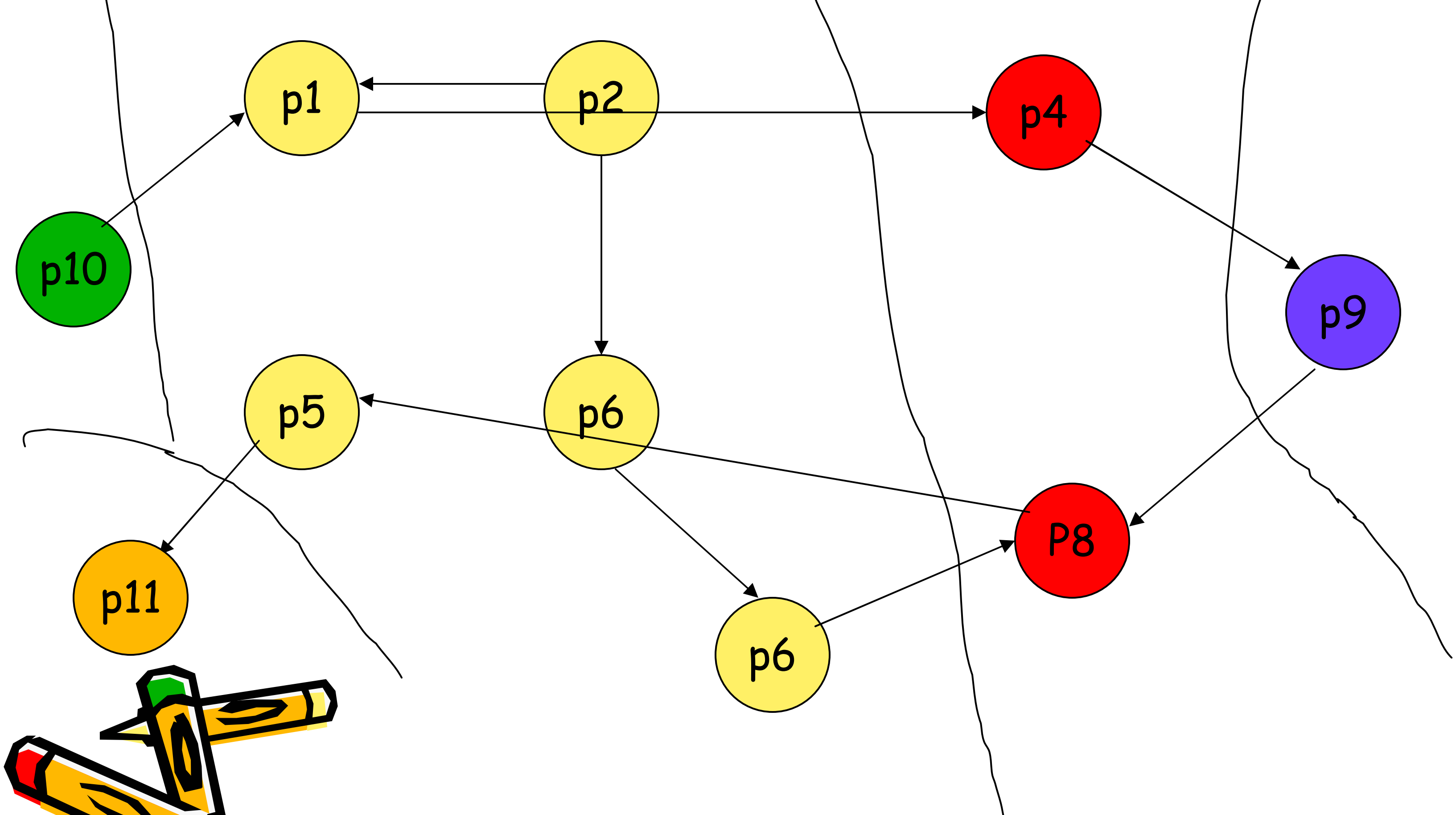


single instance model
:deadlock

Can reds declare deadlock?
Initiator : yellows
Red receives from yellows

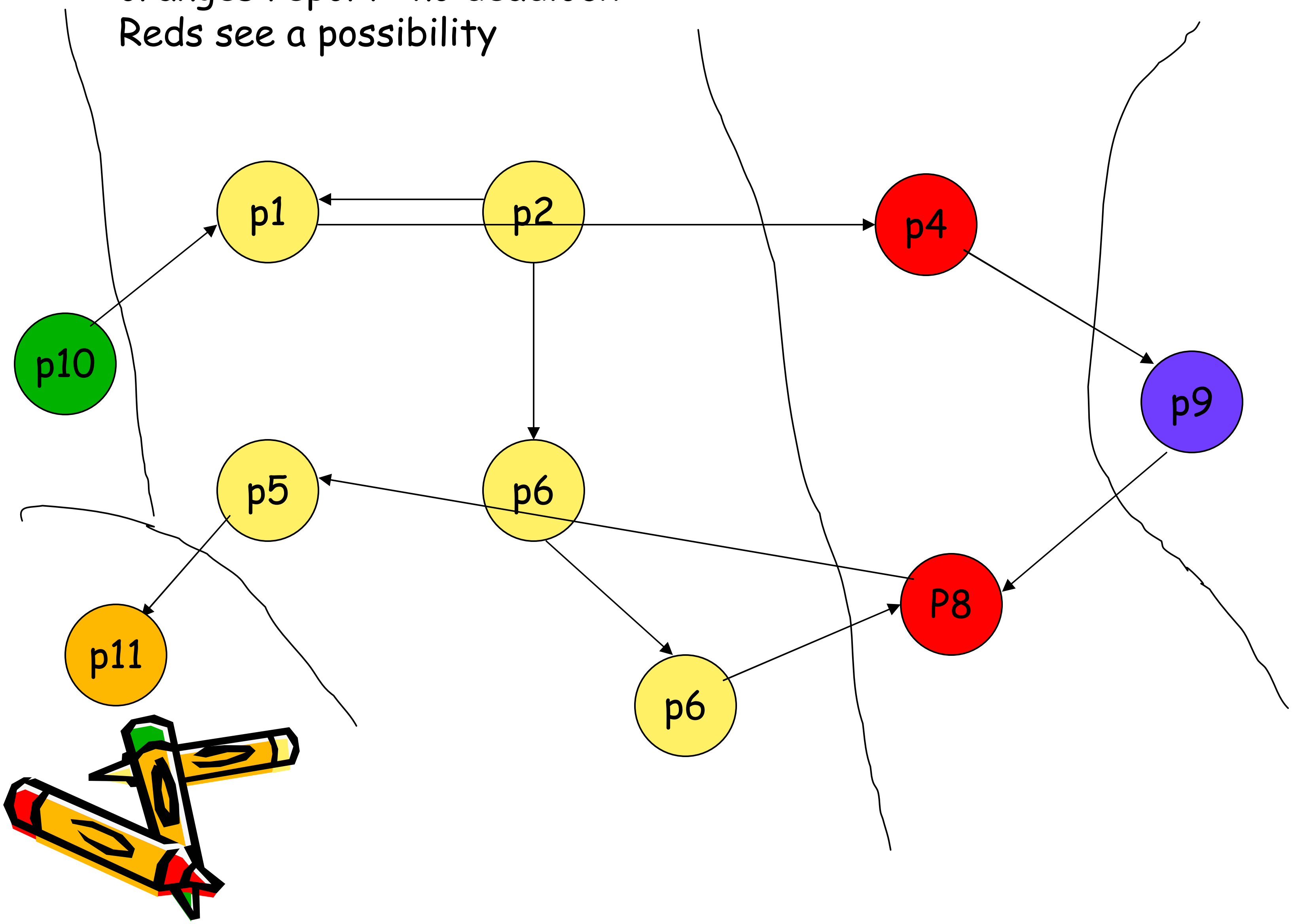
single instance model

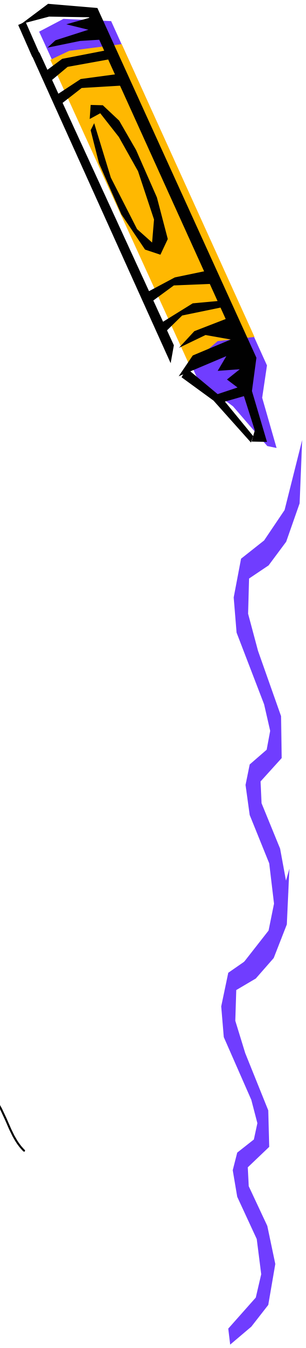
Can reds declare deadlock? - no
Initiator : yellows
Red receives from yellows
Do reds report a possibility of deadlock? – yes; what next?



single instance model

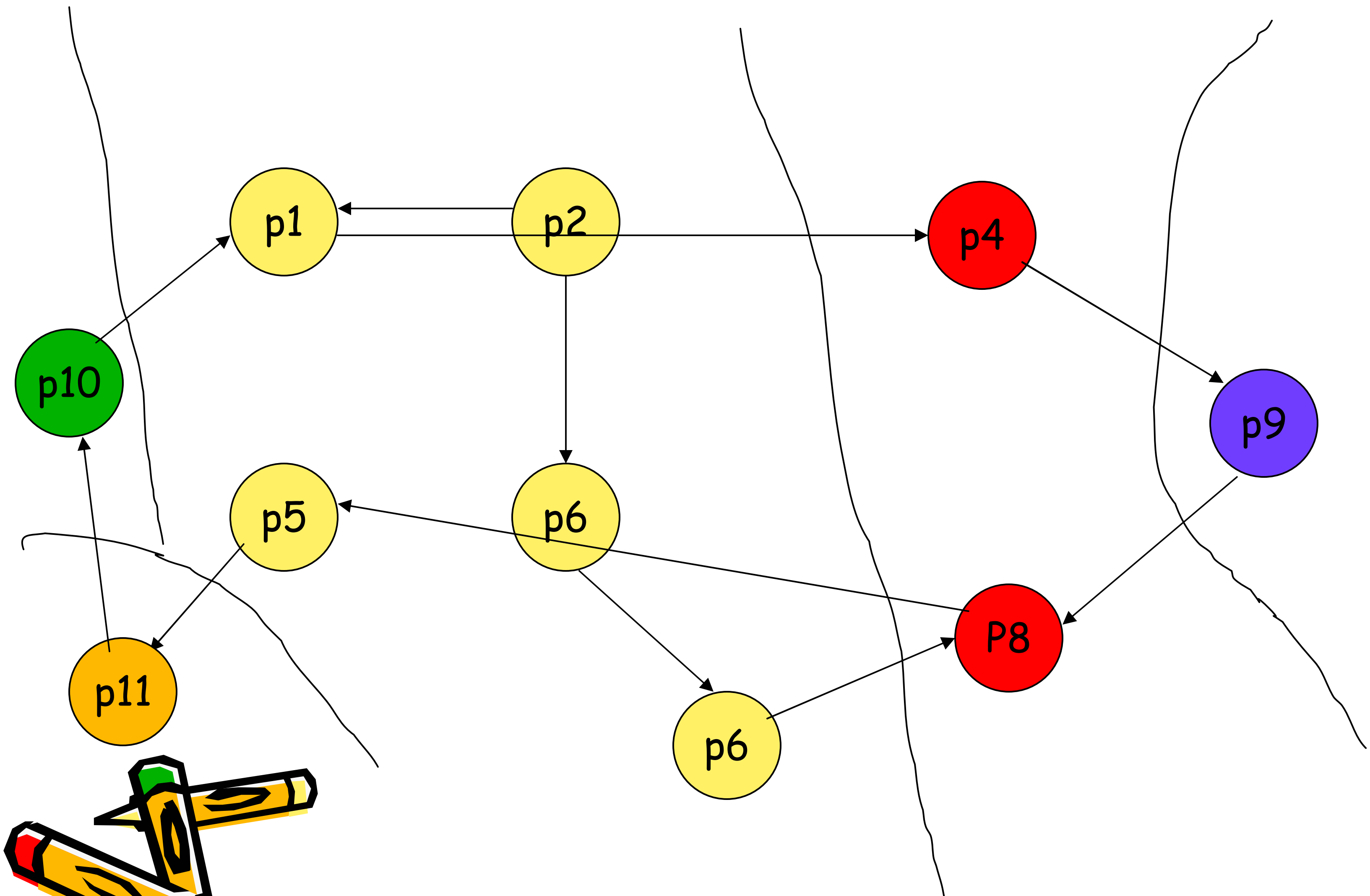
yellows initiate
oranges report 'no deadlock'
Reds see a possibility





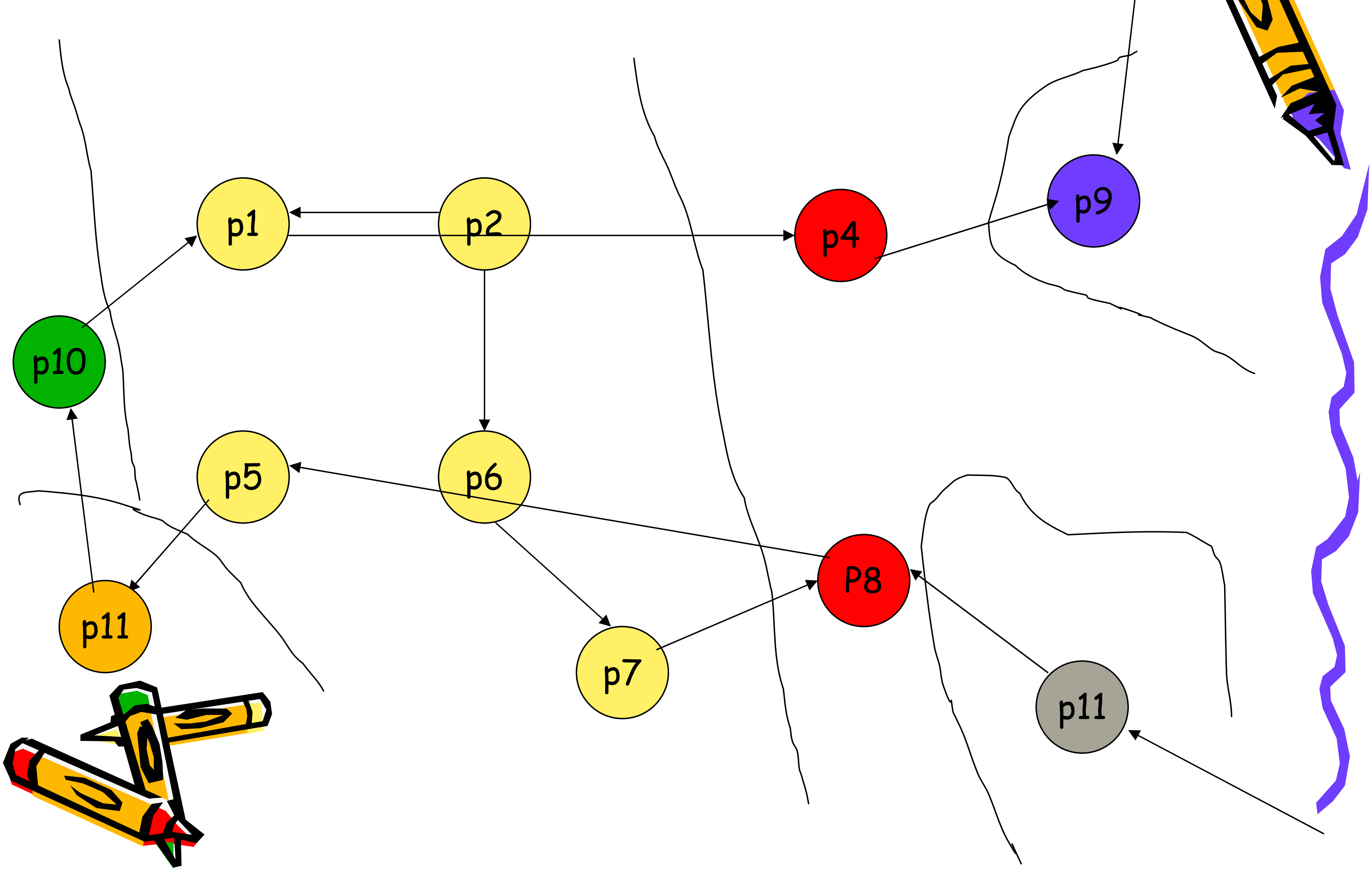
single instance model

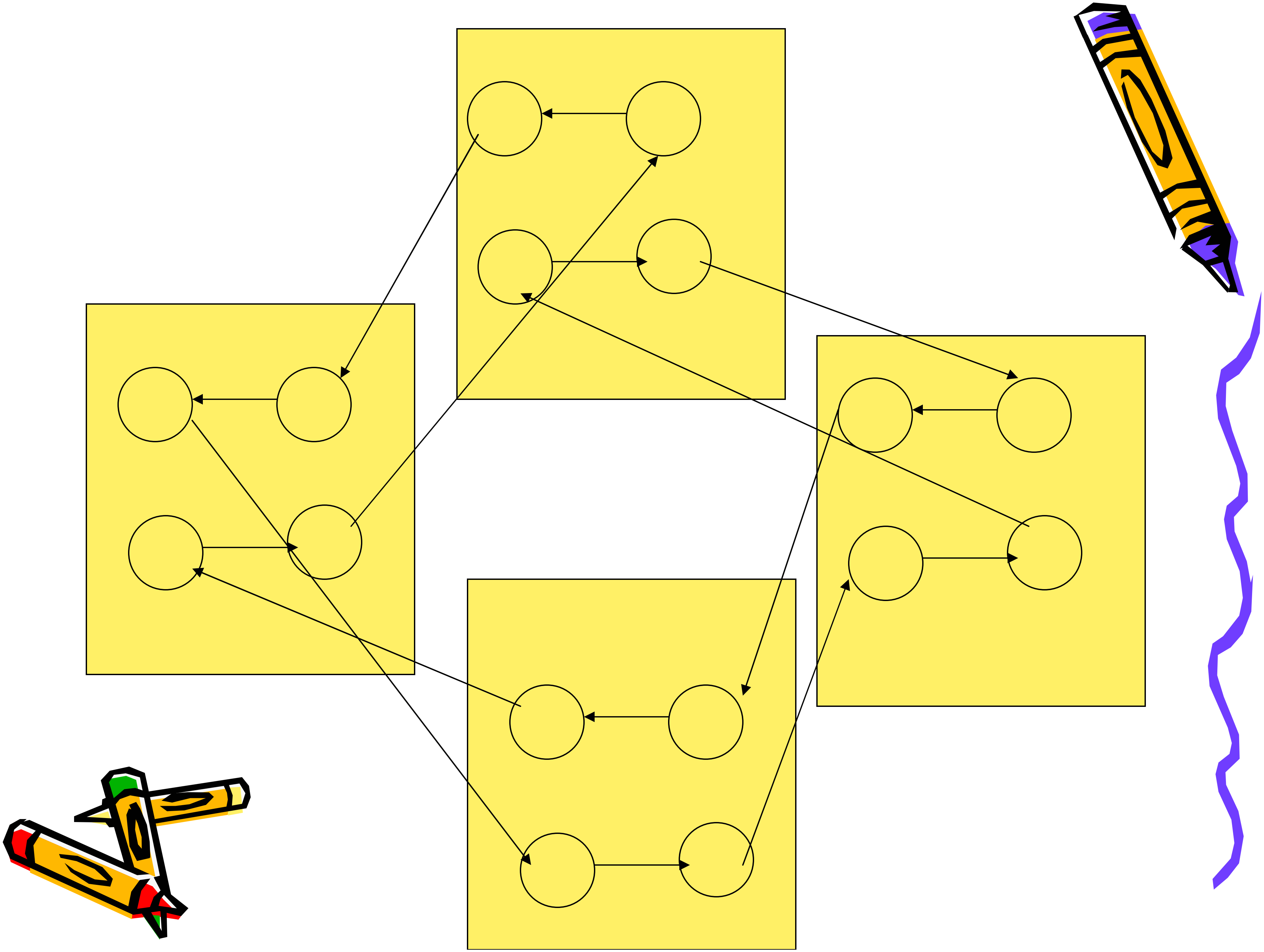
yellows initiate'

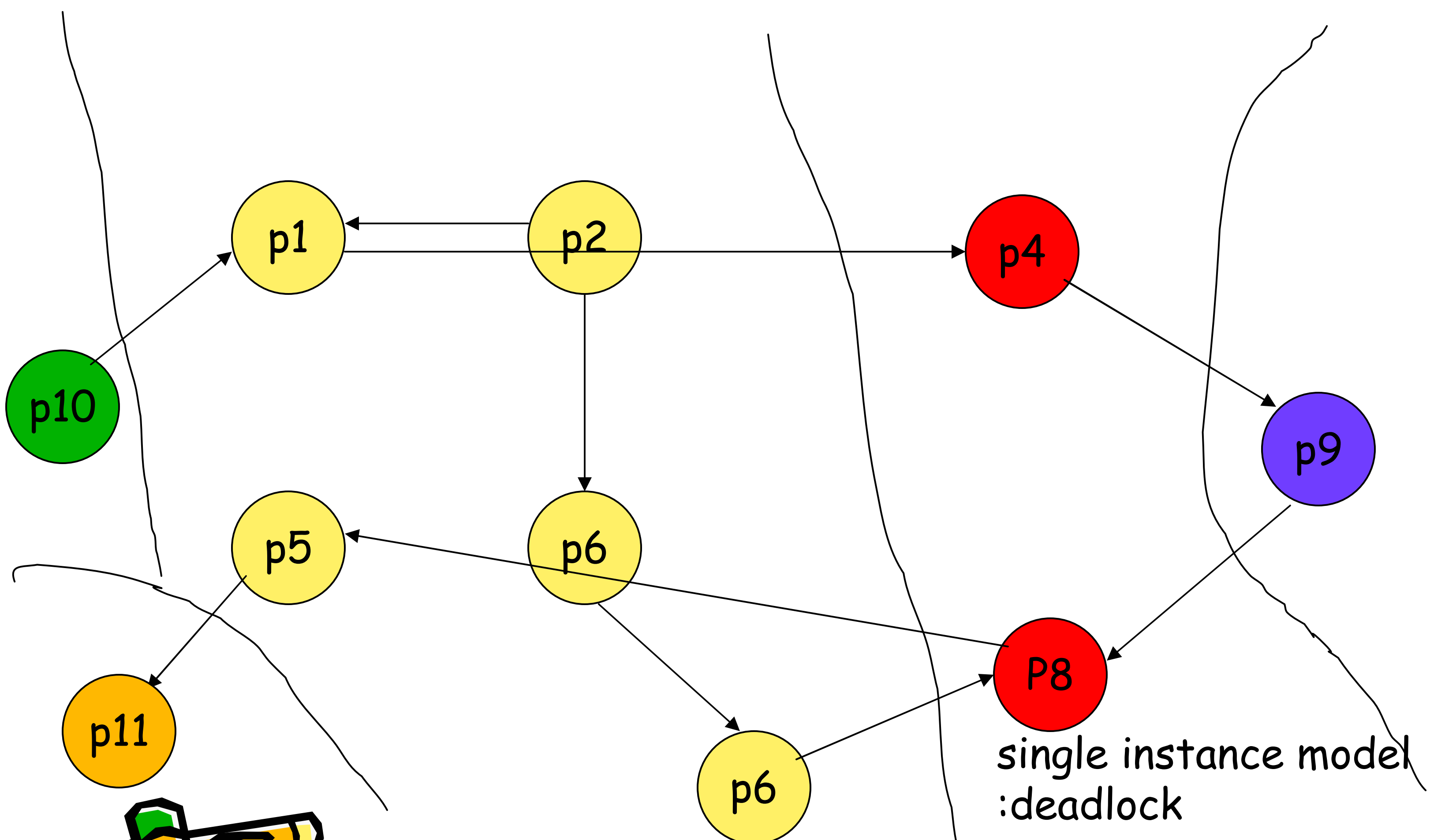


single instance model

yellows initiate'







single instance model
:deadlock

Can reds declare deadlock?
Initiator : yellows
Red receives from yellows

- If local cycle does not involve Pex, deadlock is detected
- If Pex is involved → deadlock is possible
  - Invoke distributed deadlock detection algorithm

- Example: Pex→Px1→Px2→.....->Pxn → Pex

Site si sends its WFG to site sj on which Si is blocked

On receiving the WFG, Sj updates its WFG

If sj finds a deadlock in its new WFG, not involving its Pex, deadlock is reported

Else if a cycle involving its Pex is found, Sj transmits the WFG to appropriate site Sk

After finite number of rounds, either deadlock is detected or detection halts (no deadlock).

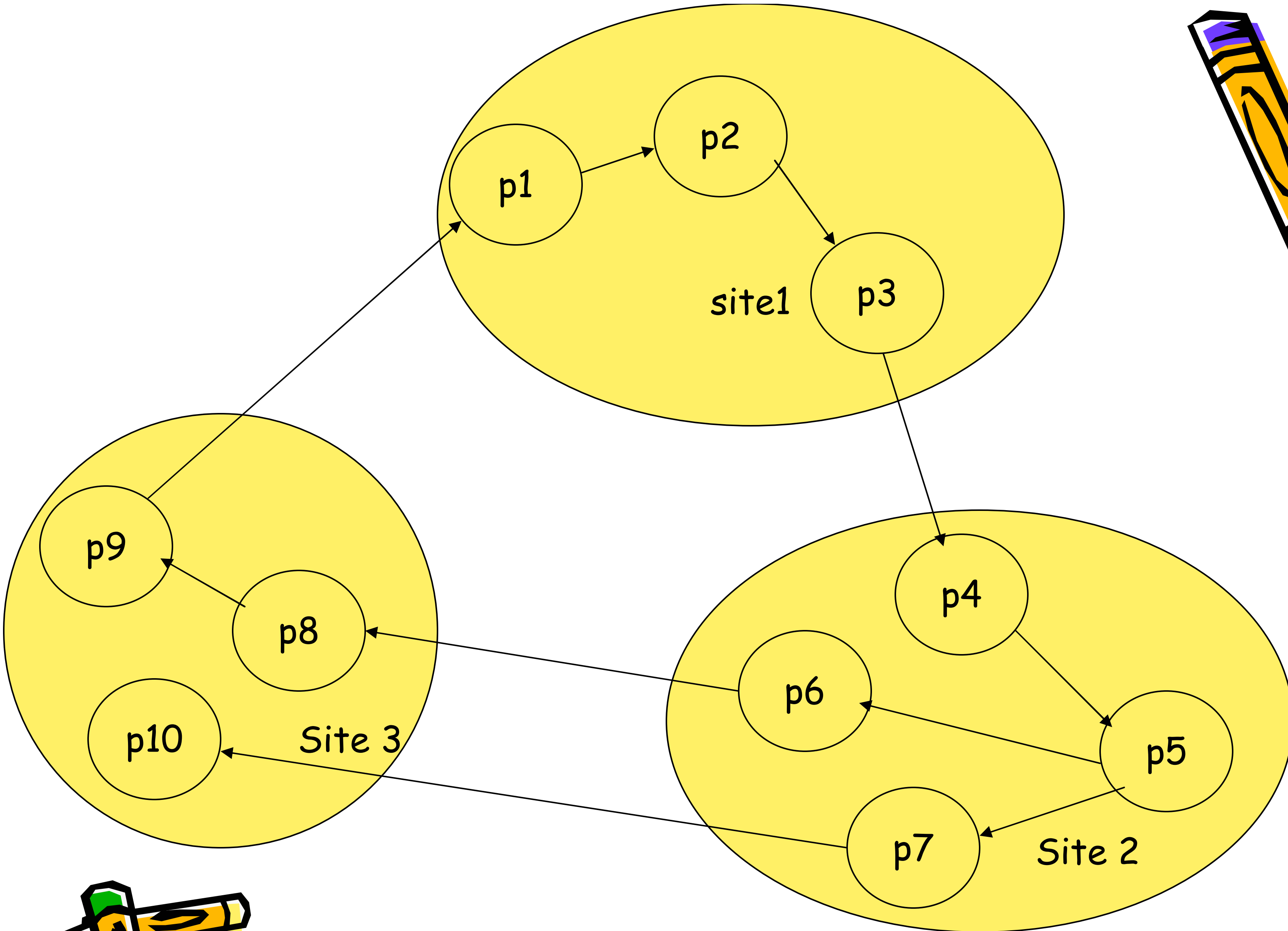
Obermarck's Path pushing Algorithm in ACM ToDS 1982

Edge chasing

- If the process is blocked on another process at another site, chase the edge by sending probe message

- If probe returns, deadlock is detected

Chandy and Mishra ACM ToCS May 83


p1
p2
p3
site1
p9
p8
p10
Site 3
p4
p6
p5
p7
Site 2

Site that sends a probe

- If Pi is locally dependent on itself
  - Deadlock is detected, terminate
- For all Pj and Pk such that
  - Pj is local
  - Pi depends on Pj
  - Pk is non-local
  - Pj depends on Pk

  Send probe (i, j, k) to site of Pk

Site that receives a probe (i, j, k)

??

Site that receives a probe (i, j, k)

If Pk is blocked, dependent (k←i) is false, Pk has not replied to all requests of Pj

set dependent (k←i) = true

if k=i declare deadlock

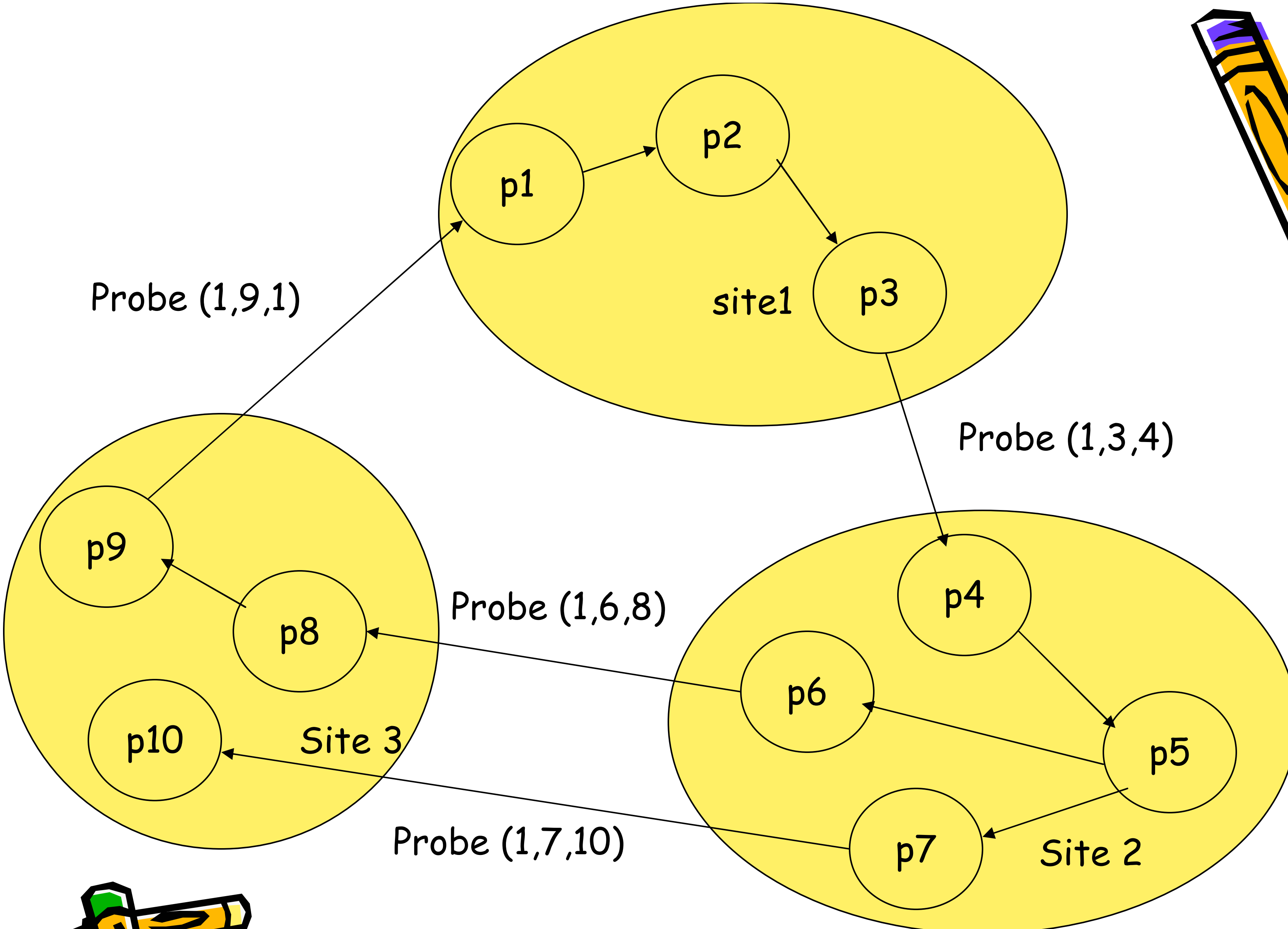
else for all Pm and Pn such that

Pk is locally dependent on Pm

Pm is waiting on Pn

Pn is on different site

send probe (i,m,n) to site of Pn



p1
p2
p3
site1
Probe (1,9,1)
Probe (1,3,4)
p9
p8
p10
Site 3
Probe (1,6,8)
p4
p6
p5
p7
Site 2
Probe (1,7,10)

Diffusing computation based algorithm

- Deadlock detection is diffused through the global WFG
- When there's a deadlock, the diffusing computation terminates

- A query (i,j,k) is sent
  - [initiator:i, currently from j, to k]
- An active process ignores an incoming query.
- A blocked process on receiving a query does the following:
  - If this is the first time it receives a query for i (engaging query)
    - propagate query to all processes in its dependent set
    - set $count_k$ (i) = no of query messages sent
  - If not an engaging query
    - If Pk remained blocked since it received the engaging query
      - Send reply
    - Else discard message
- A blocked process on receiving a reply (i, k, j)
  - If Pk remained blocked since it received engaging query
    - Decrement $count_k$ (i) by 1.
    - send response to engaging query for i only after the count reaches 0
  - Else discard
- When initiator receives all replies → detects a deadlock

Readings

- Knapp: deadlock detection in distributed databases, ACM Computing surveys, Dec 1987
  - Recommended reading for CS 451