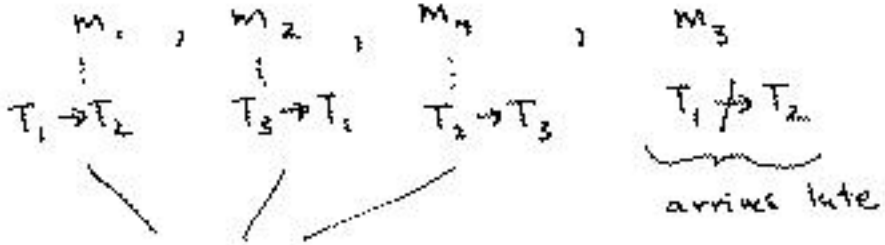
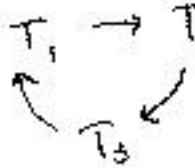
Let's look at another example:

Host A	Host B	Host C
T_1 lock X		
	T_2 lock Y	
		T_3 lock Z
	T_1 lock Y (send m_1 to detector)	
T_3 lock X (send m_2 to detector)		
	T_2 unlock Y (T_1 gets Y, send m_3 to detector)	
		T_2 lock Z (send m_4 to detector)

time ↓

Detector could receive:





deadlock!

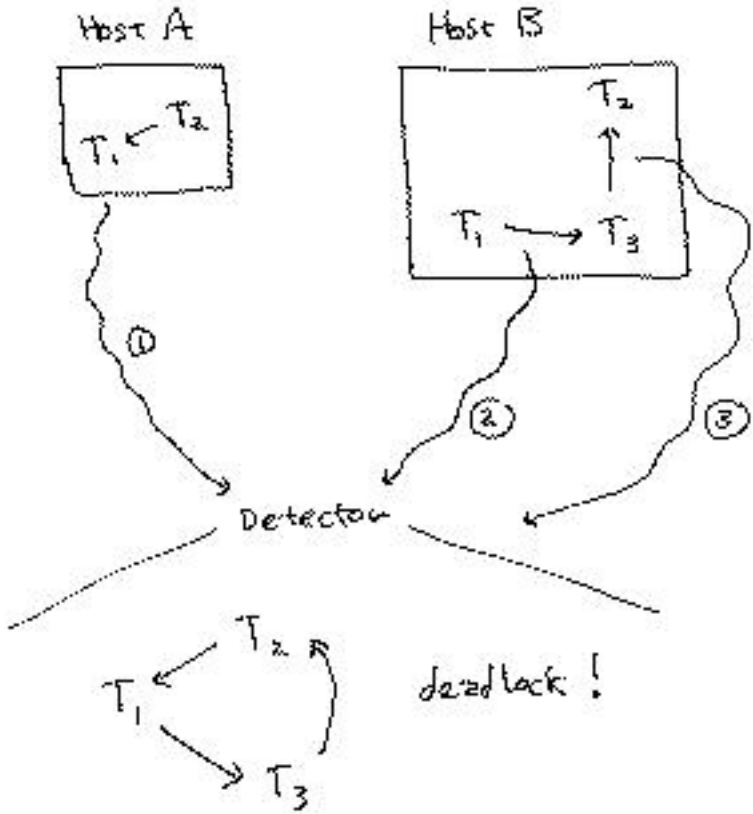
BUT THIS IS FALSE!!

Proposed Solution:

Assign one host to be "deadlock detector"

- Each time a (local) wait-for graph is changed (edge added or removed), send a message to the deadlock detector

Ex:



Distributed Deadlock Detection

A first proposal: instead of having one detector, let each host communicate its WF-graph to the others.

Observe: if $T_1 \to T_2 \to T_3$, then "indirectly" $T_1 \overset{*}{\to} T_3$ (waiting for), so we can add indirect edges to detect deadlock cycles in WF Graph.

New, proposed protocol:

(1) if T at host A cannot lock some item R owned by $T_1, T_2, \ldots T_k$ add $T \to T_i$ $i = 1, 2, \ldots k$ to WF graph & send messages to coordinator(T), coordinator(T_i), $i = 1, 2 \ldots k$

(2) when coordinator receives $(T \to T')$ message, add $T \to T'$ to WF graph; if T' is known to be blocked, and $T \overset{*}{\to} T''$ in the local WF graph, coordinator sends $T \to T''$ to coordinator(T'') - for each such T''

Note: obviously, we also need similar rules to deal with cancelling edges.

EXAMPLE

S_1 S_2 S_3 (Hosts)

T_1 locks R_1

T_2 locks R_2

T_3 locks R_3

// note: coordinator (T_i) is at host S_i

T_1 tries to lock R_2
⇒ add $T_1 \to T_2$
send m_1 to S_1

T_2 tries to lock R_3
⇒ add $T_2 \to T_3$
send m_2 to S_2

T_3 tries to lock R_1
⇒ add $T_3 \to T_1$
send m_3 to S_3

m_2 arrives, so
add $T_2 \to T_3$
but since T_3 was blocked, no send

m_1 arrives, so
add $T_1 \to T_2$

m_3 arrives,
add $T_3 \to T_1$

RESULT

$T_3 \to T_1 \to T_2$ $T_1 \to T_2 \to T_3$ $T_2 \to T_3 \to T_1$

NO DEADLOCK DETECTED!

Lesson:

Distributed Deadlock Detection is a difficult problem.

Edge Chasing Protocols

Idea: send special messages called probes that "follow" the direction of edges in a distributed WF graph; if a probe travels in a cycle, we have deadlock.

Complications

(1) Can be many concurrent "probe computations" ⇒ probe messages and related variables should be tagged with unique "incarnation numbers" to tell them apart

(2) WF edges are dynamic - being added & removed during transaction processing

(3) Danger of error in detection if probe messages are "out of order"

The good news:

Probe protocols have been devised so that:

(1) they never erroneously detect deadlock,

(2) if there exists a deadlock before a "probe computation" is started, then the probe will detect the deadlock.