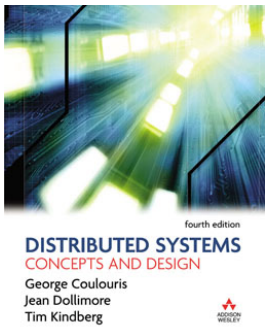
Slides for Chapter 11: Time and Global State

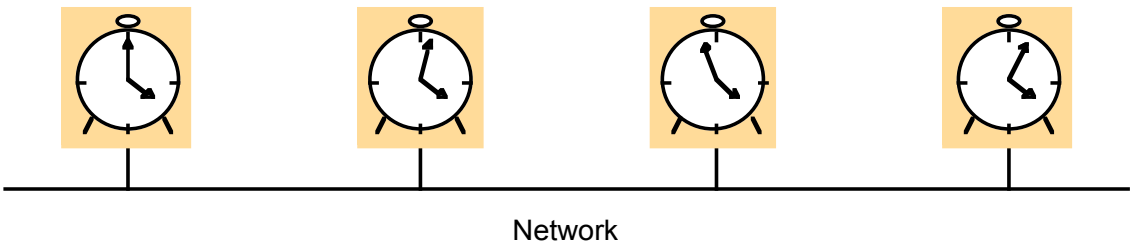


From **Coulouris, Dollimore and Kindberg**
Distributed Systems: Concepts and Design

Edition 4, © Pearson Education 2005

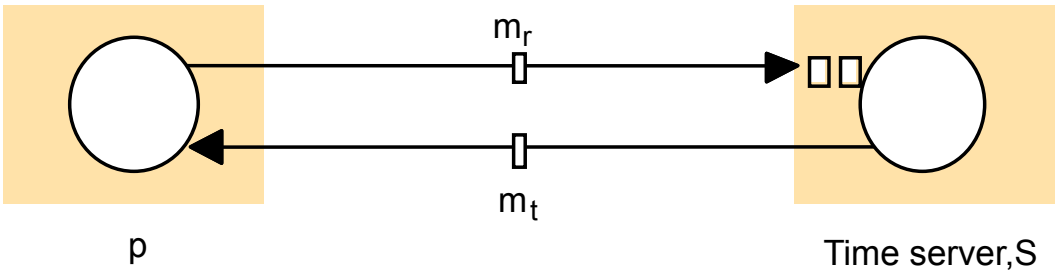
Edited and supplemented by Jonne Itkonen, for TIE427 (2011)

Figure 11.1
Skew between computer clocks in a distributed system



Kuva yrittää vain kertoa, että hajautetun järjestelmän kellot ovat aina eri ajassa. Nykyään, kun kaikki on nopeaa ja tehokasta, on järjettömän pieneltä tuntuvalla aikaerollakin merkitys. On jopa jouduttu siirtämään etäkoneita lähemmäs palvelimia, jotta bitin kulkuaika piuhassa pienenisi, latenssi vähenisi.

Figure 11.2
Clock synchronization using a time server

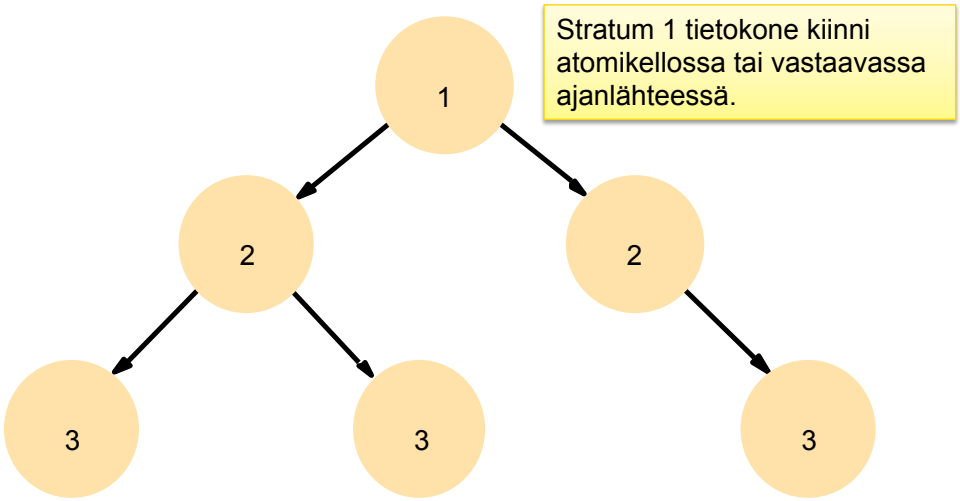


Cristian's algorithm:

1. *p* requests time in message m_r.
2. *S* returns time *t* in message m_t.
3. *p* records round-trip time T_{round}
4. *p* estimates time to change as $t+T_{round}/2$, accuracy being $\pm (T_{round}/2\text{-}min)$, where *min* is the time between sending m_r and inserting time to m_t.

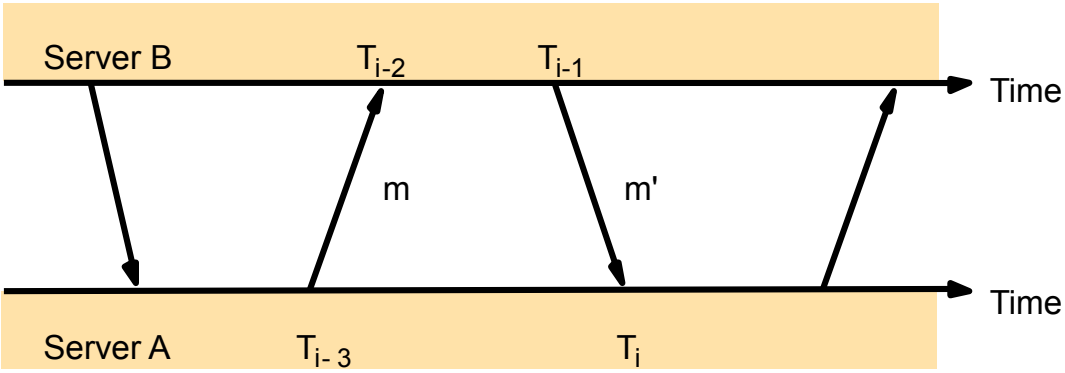
Algoritmi on arka palvelimen rikkoutumiselle ja huijaavalle palvelimelle.

Figure 11.3
An example synchronization subnet in an NTP implementation



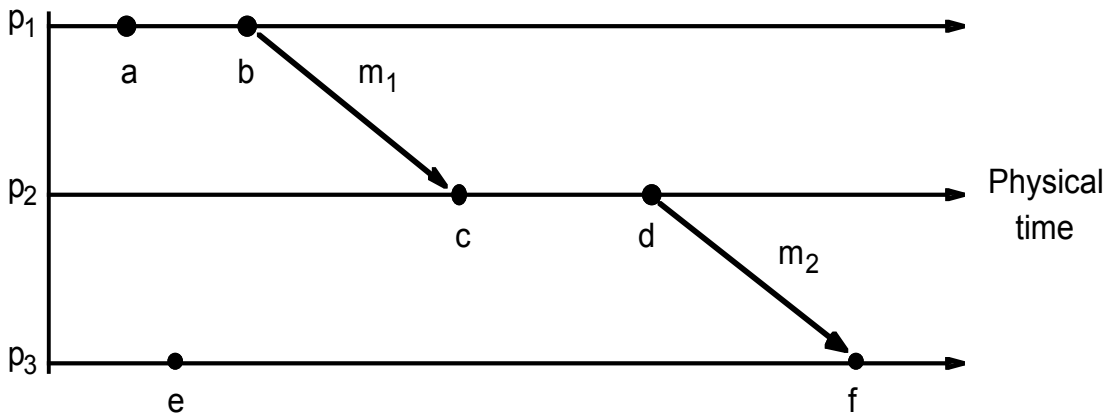
Note: Arrows denote synchronization control, numbers denote strata.

Figure 11.4
Messages exchanged between a pair of NTP peers



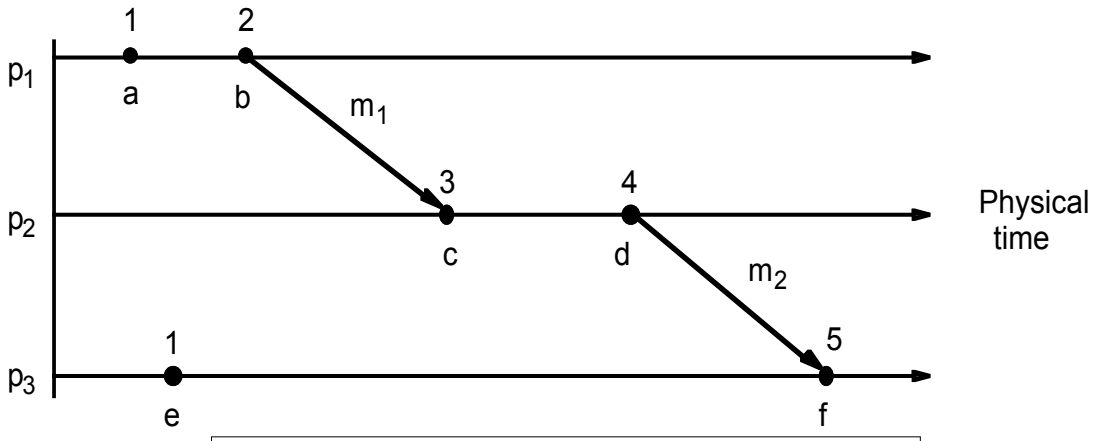
NTP laskee näiden perusteella siirtymät (offset) o_i ja viipeet d_i jokaiselle viestiparille. Näitä suodattamalla, valikoimalla ja tallentamalla saadaan lasketuksi siirtymä o palvelinten kellojen välillä.

Figure 11.5
Events occurring at three processes



Tapahtumien *a* ja *e* järjestystä ei voida määrittää *tapahtui ennen -relaatiolla*, joten niiden sanotaan olevan *samanaikaisia* (concurrent).

Figure 11.6
Lamport timestamps[1978] for the events shown in Figure 11.5



LC1:
L_i is incremented before each event is issued at process p_i: $L_i := L_i + 1$.
LC2:
(a) When a process p_i sends a message m, it piggybacks on m the value $t = L_i$.
(b) On receiving (m,t), a process p_j computes $L_j := max(L_j,t)$ and
then applies LC1 before timestamping the event *receive(m)*.

Lamportin mallin ongelmat ja vektorikellot

- Lamport: e → e' ⇒ L(e) < L(e'), mutta ei toisinpäin
- Vektorikelloilla hoituu (Mattern [1989], Fidge [1991]):
  - VC1: Initially, $V_i[j] = 0$, for i,j=1,2,…,N.
  - VC2: Just before p_i timestamps an event, it sets $V_i[i]:=V_i[i]+1$
  - VC3: p_i includes the value $t=V_i$ in every message it sends.
  - VC4: Whe *pi* receives a timestamp t in a message, it sets $V_i[j]:=max(V_i[j],t[j])$, for j=1,2,…,N. Taking the component-wise maximum of two vector timestamps in this way is known as a merge operator.
- V:n koko on huima isoilla prosessimäärillä…
  - Korjaus: muutoksen siirto, Lamport-riittävä

Figure 11.7
Vector timestamps for the events shown in Figure 11.5

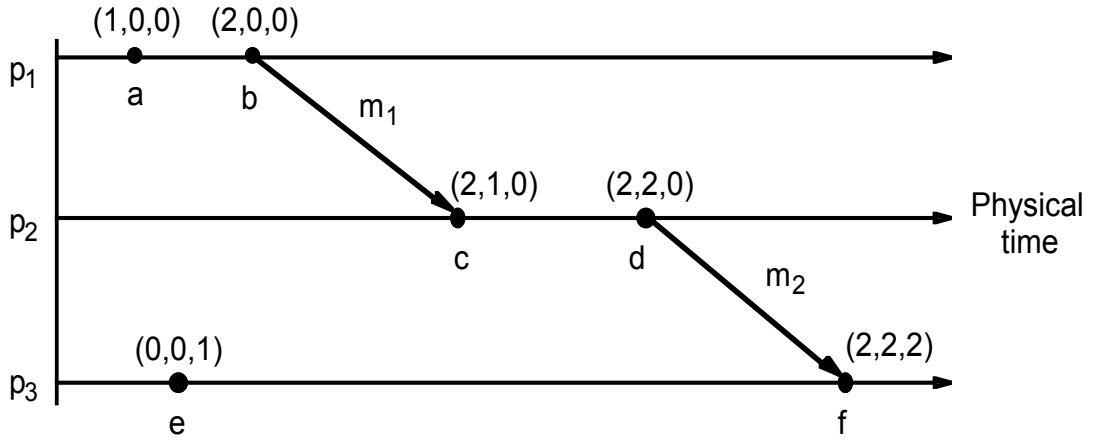
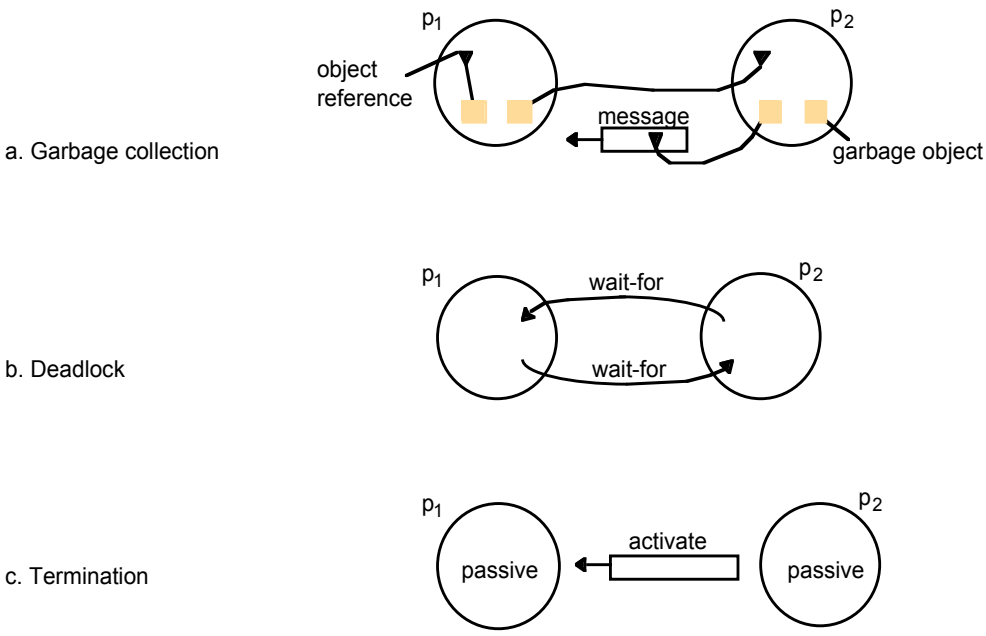
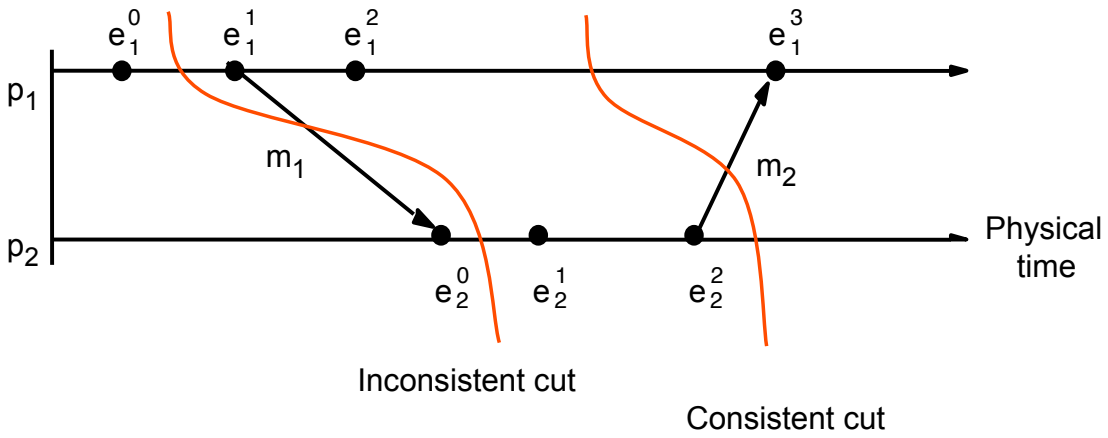


Figure 11.8
Detecting global properties



- Seuraavat kalvot käsittelevät tarkemmin globaalin tilan määrittelyä. Ne ovat mukana mielenkiinnon vuoksi, eivätkä kuulu tenttiin.

Figure 11.9
Cuts



Viestin m_1:n lähettäminen ei sisälly leikkaukseen, joten leikkaus on epäyhtenäinen.
Viestin m_2:n vastaanotto ei sisälly leikkaukseen, mutta leikkaus on silti yhtenäinen.

Figure 11.10
Chandy and Lamport's 'snapshot' algorithm

Marker receiving rule for process p_i

On p_i's receipt of a *marker* message over channel c:

if (p_i has not yet recorded its state) it

records its process state now;

records the state of c as the empty set;

turns on recording of messages arriving over other incoming channels;

else

p_i records the state of c as the set of messages it has received over c since it saved its state.

end if

Marker sending rule for process p_i

After p_i has recorded its state, for each outgoing channel c:

p_i sends one marker message over c (before it sends any other message over c).

Figure 11.11
Two processes and their initial states

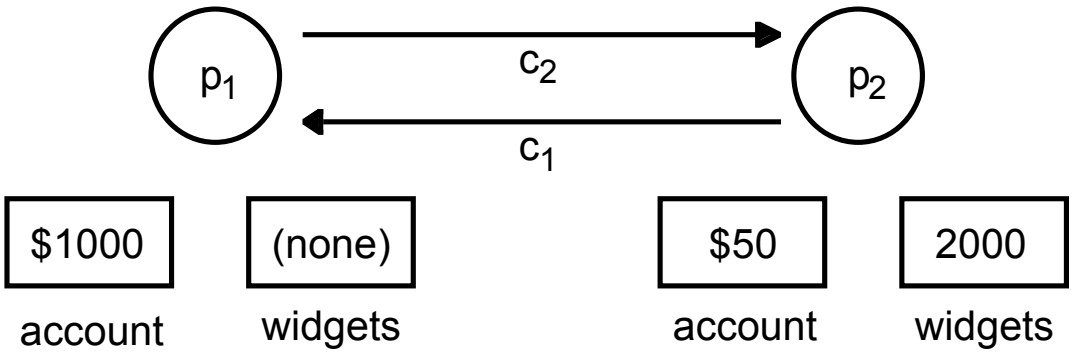


Figure 11.12
The execution of the processes in Figure 11.11

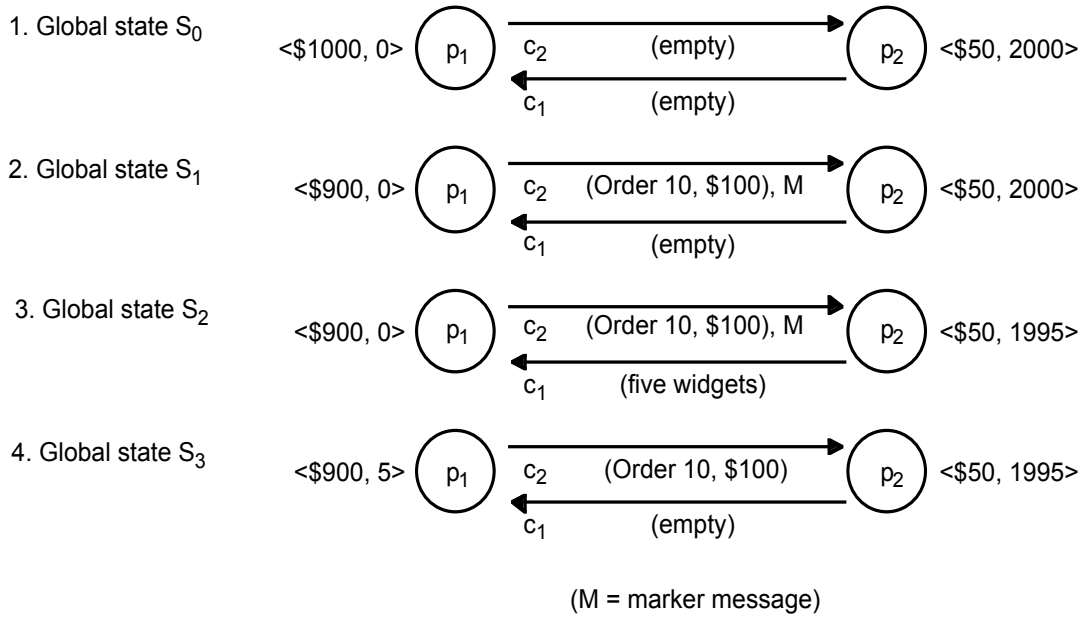


Figure 11.13
Reachability between states in the snapshot algorithm

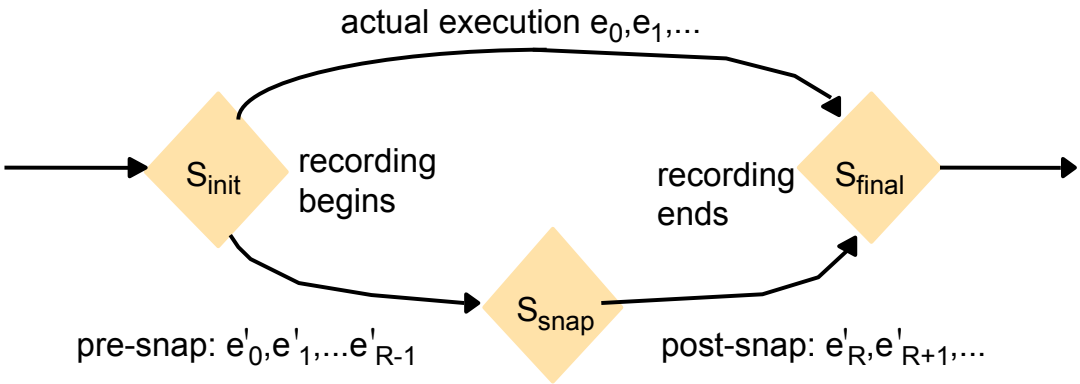


Figure 11.14
Vector timestamps and variable values for the execution of Figure 11.9

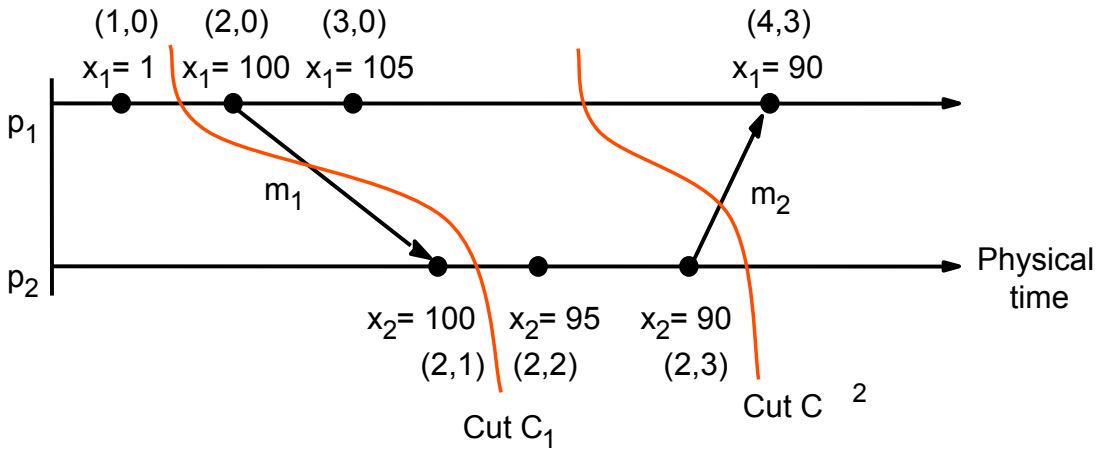
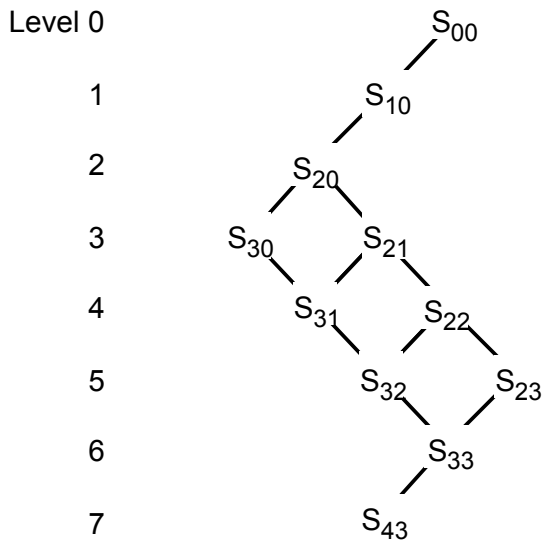


Figure 11.15
The lattice of global states for the execution of Figure 11.14



Sij = global state after *i* events at process 1 and *j* events at process 2

Figure 11.16
Algorithms to evaluate *possibly* ϕ and *definitely* ϕ

1. *Evaluating possibly* ϕ *for global history H of N processes*

$L := 0;$
$States := \{ (s_1^0, s_2^0, \ldots, s_N^0) \};$
while $(\phi(S) = False$ for all $S \in$ States$)$
    $L := L + 1;$
    $Reachable := \{S' : S'$ reachable in H from some $S \in$ States $\wedge$ level$(S') = L\};$
    $States := Reachable$
end while
output "*possibly* ϕ";

2. *Evaluating definitely* ϕ *for global history H of N processes*

$L := 0;$
if $(\phi(s_1^0, s_2^0, \ldots, s_N^0))$ *then* $States := \{\}$ *else* $States := \{ (s_1^0, s_2^0, \ldots, s_N^0) \};$
while $(States \neq \{\})$
    $L := L + 1;$
    $Reachable := \{S' : S'$ reachable in H from some $S \in$ States $\wedge$ level$(S') = L\};$
    $States := \{S \in Reachable : \phi(S) = False\}$
end while
output "*definitely* ϕ";

Figure 11.17
Evaluating *definitely* ϕ