Process 1 Outgoing channels 2 sent 1, 2, 3, 4, 5, 6 3 sent 1, 2, 3, 4, 5, 6 Incoming channels	**Process 3** Outgoing channels 2 sent 1, 2, 3, 4, 5, 6, 7, 8 Incoming channels 1 received 1, 2, 3 stored 4, 5, 6 2 received 1, 2, 3 stored 4 4 received 1, 2, 3
Process 2 Outgoing channels 3 sent 1, 2, 3, 4 4 sent 1, 2, 3, 4 Incoming channels 1 received 1, 2, 3, 4 stored 5, 6 3 received 1, 2, 3, 4, 5, 6, 7, 8	**Process 4** Outgoing channels 3 sent 1, 2, 3 Incoming channels 2 received 1, 2 stored 3, 4

Figure 15.6 An Example of a Snapshot

```
if (!token_present)
{
    clock++;                                      /* Prelude */
    broadcast (Request, clock, i);
    wait (access, token);
    token_present = true;
}

token_held = true;
<critical section>;

token[i] = clock;                                /* Postlude */
token_held = false;
for (int j = i + 1; j < n; j++)
{
    if (request(j) > token[j] && token_present)
    {
        token_present = false;
        send (access, token[j]);
    }
}
for (j = 1; j <= i-1; j++)
{
    if (request(j) > token[j] && token_present)
    {
        token_present = false;
        send(access, token[j]);
    }
}
```

(a) First Part

```
if (received (Request, k, j))
{
    request (j) = max(request(j), k);
    if (token_present && !token_held)
      <text of postlude>;
}
```

(b) Second Part

Notation

send (j, access, token) — end message of type access, with token, by process j

broadcast (request, clock, i) — send message from process i of type request, with timestamp clock, to all other processes

received (request, t, j) — receive message from process j of type request, with timestamp t

Figure 15.11 Token-Passing Algorithm (for process P*i*)

```
if (e(T2) < e(T1))
    halt_T2 ('wait');
else
    kill_T2 ('die');
```

(a) Wait-die method

```
if (e(T2) < e(T1))
    kill_T1 ('wound');
else
    halt_T2 ('wait');
```

(b) Wound-wait method

Figure 15.13 Deadlock Prevention Methods