The challenges of non-stable predicates

- Consider a non-stable predicate Φ encoding, say, a safety property. We want to determine whether Φ holds for our program.

The challenges of non-stable predicates

- Consider a non-stable predicate Φ encoding, say, a safety property. We want to determine whether Φ holds for our program.
- Suppose we apply Φ to Σ^s

The challenges of non-stable predicates

- Consider a non-stable predicate Φ encoding, say, a safety property. We want to determine whether Φ holds for our program.
- Suppose we apply Φ to Σ^s
- Φ holding in Σ^s does not preclude the possibility that our program violates safety!

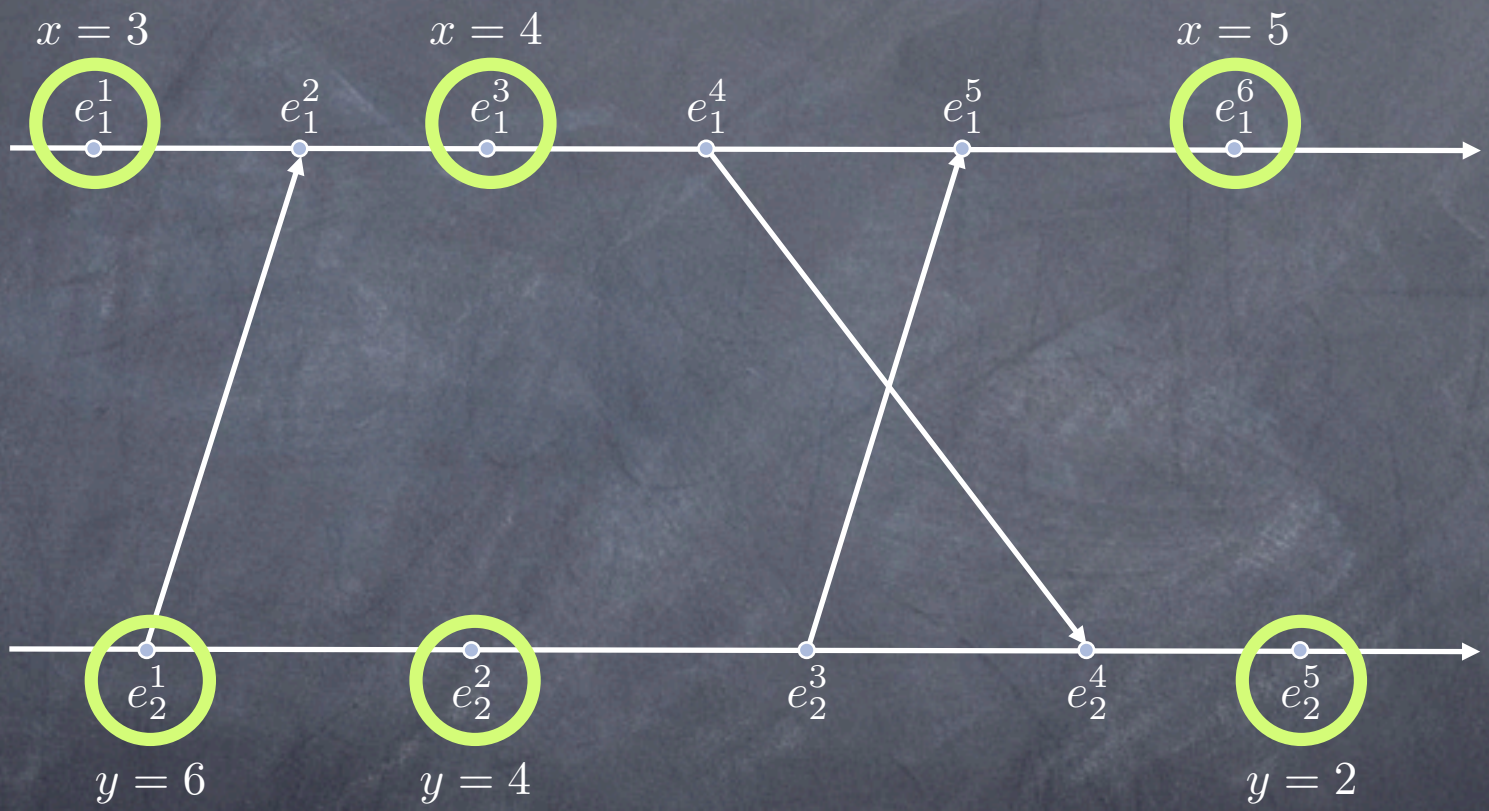
The challenges of non-stable predicates

- Consider now a different non-stable predicate Φ. We want to determine whether Φ ever holds during a particular computation.
- Suppose we apply Φ to Σ^s

The challenges of non-stable predicates

- Consider now a different non-stable predicate Φ. We want to determine whether Φ ever holds during a particular computation.
- Suppose we apply Φ to Σ^s
- Φ holding in Σ^s does not imply that Φ ever held during the actual computation!

Example



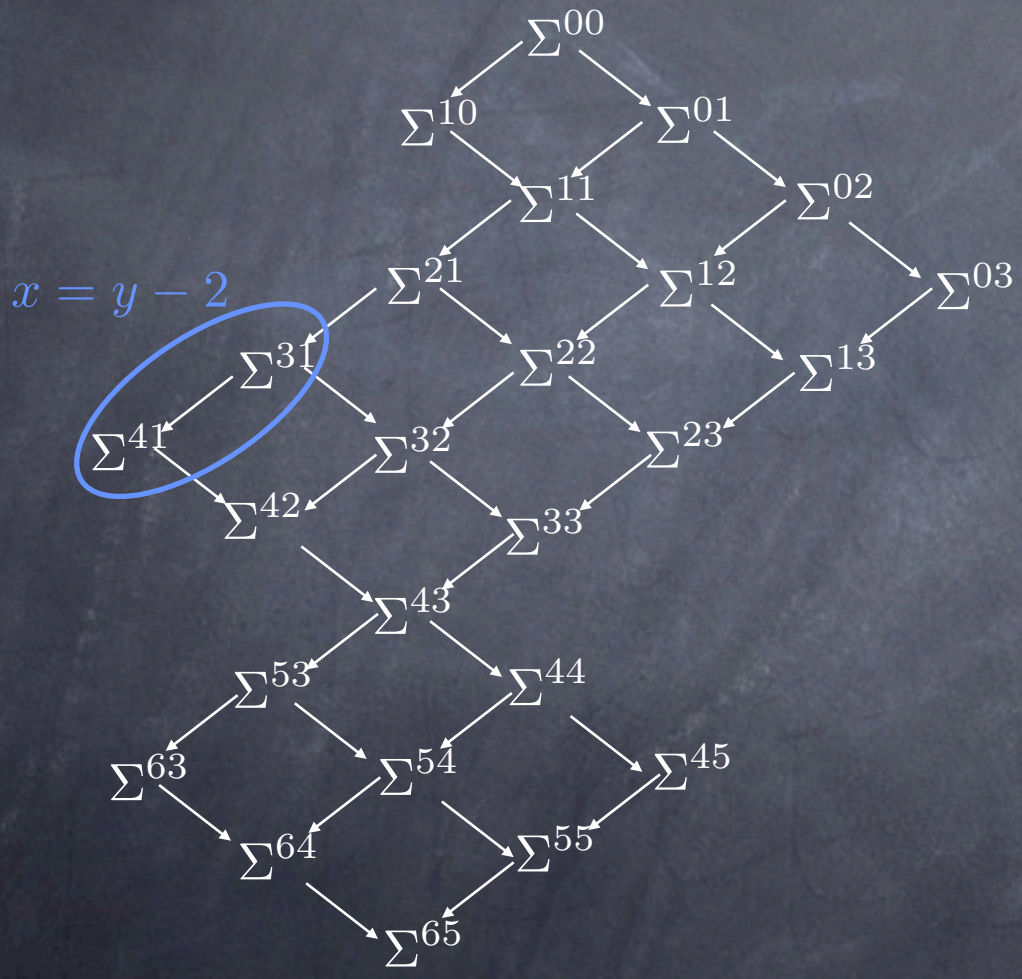
Detect whether the following predicates hold:

$x = y$ $x = y - 2$

Assume that initially:

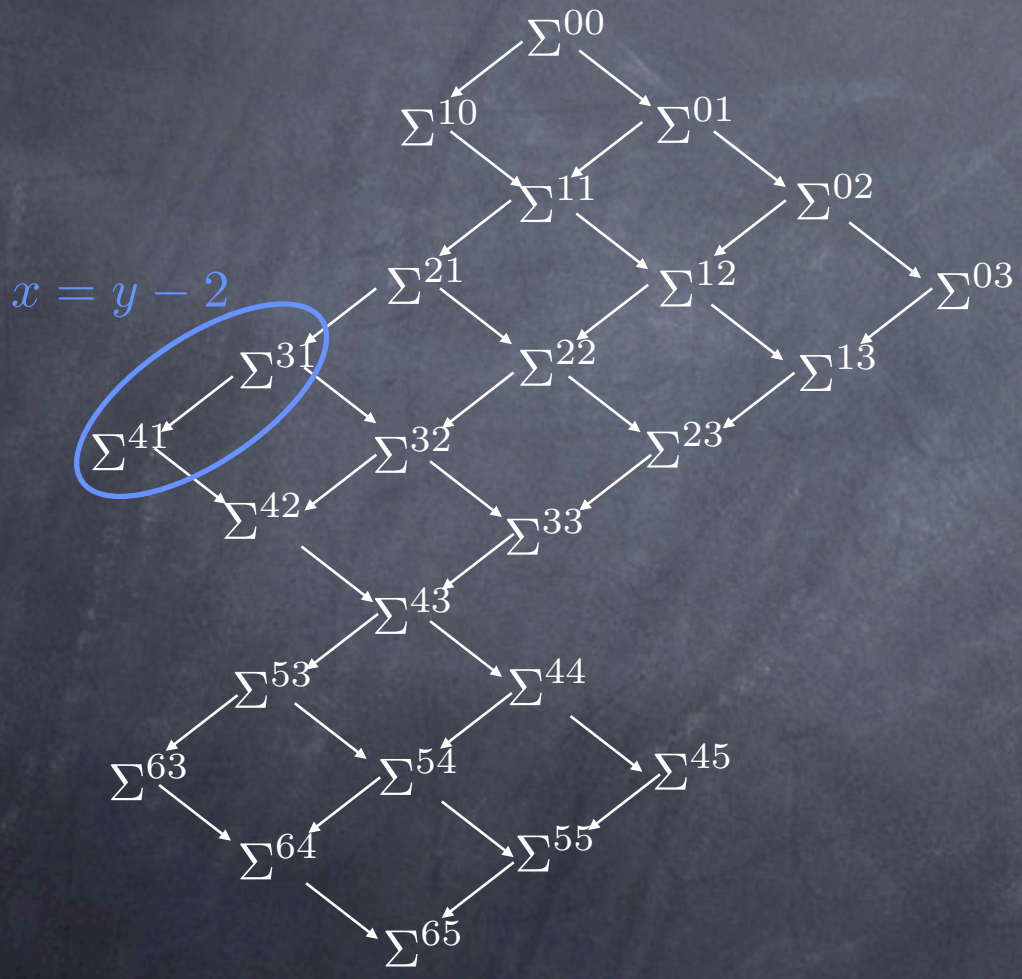
$x = 0; y = 10$

Possibly



- If Σ^s is Σ^{31} or Σ^{41}, $x = y - 2$ is detected, but it may never have occurred

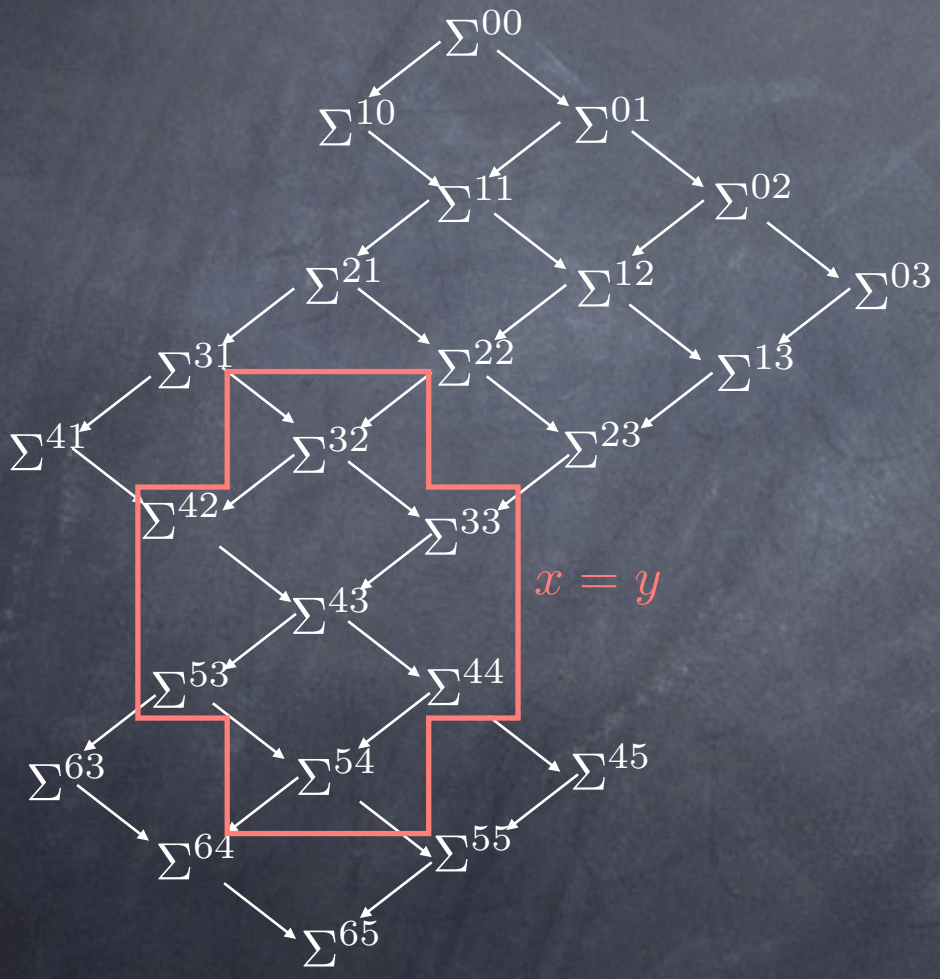
Possibly



- If Σ^s is Σ^{31} or Σ^{41}, $x = y - 2$ is detected, but it may never have occurred

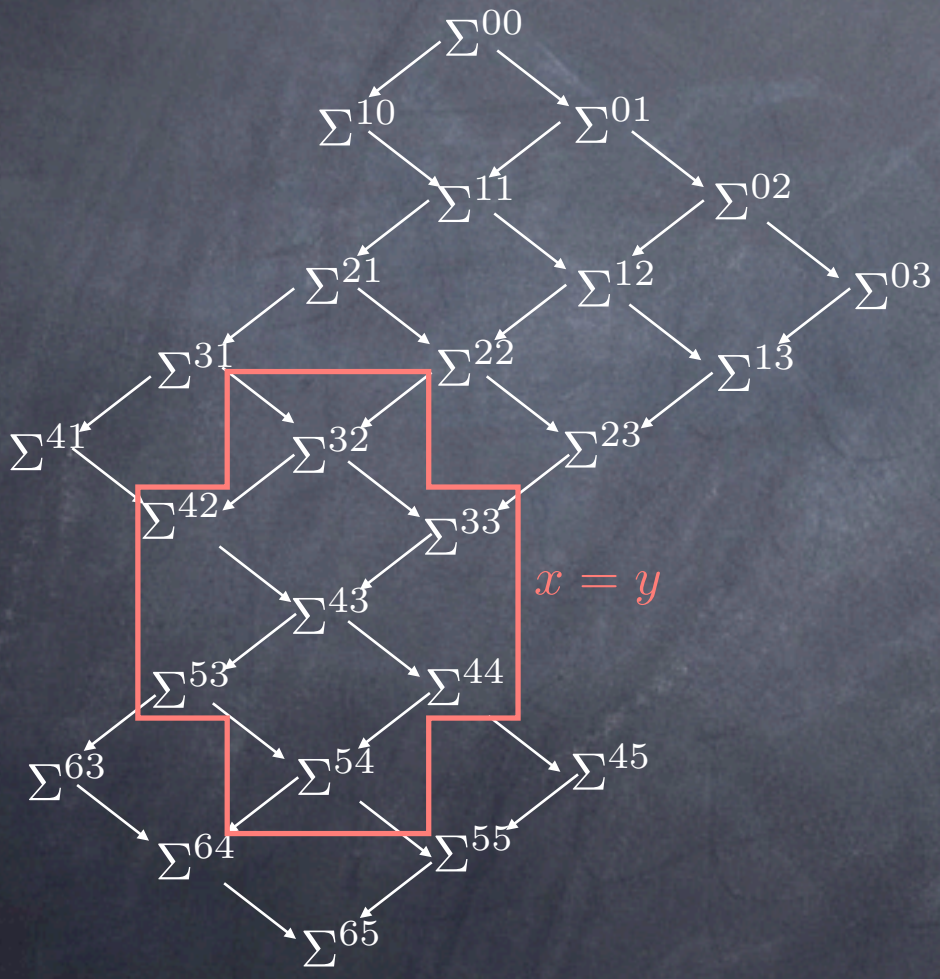
- Possibly(Φ)
  There exists a consistent observation of the computation O such that Φ holds in a global state of O

Definitely



- We know that $x = y$ has occurred, but it may not be detected if tested before Σ^{32} or after Σ^{54}

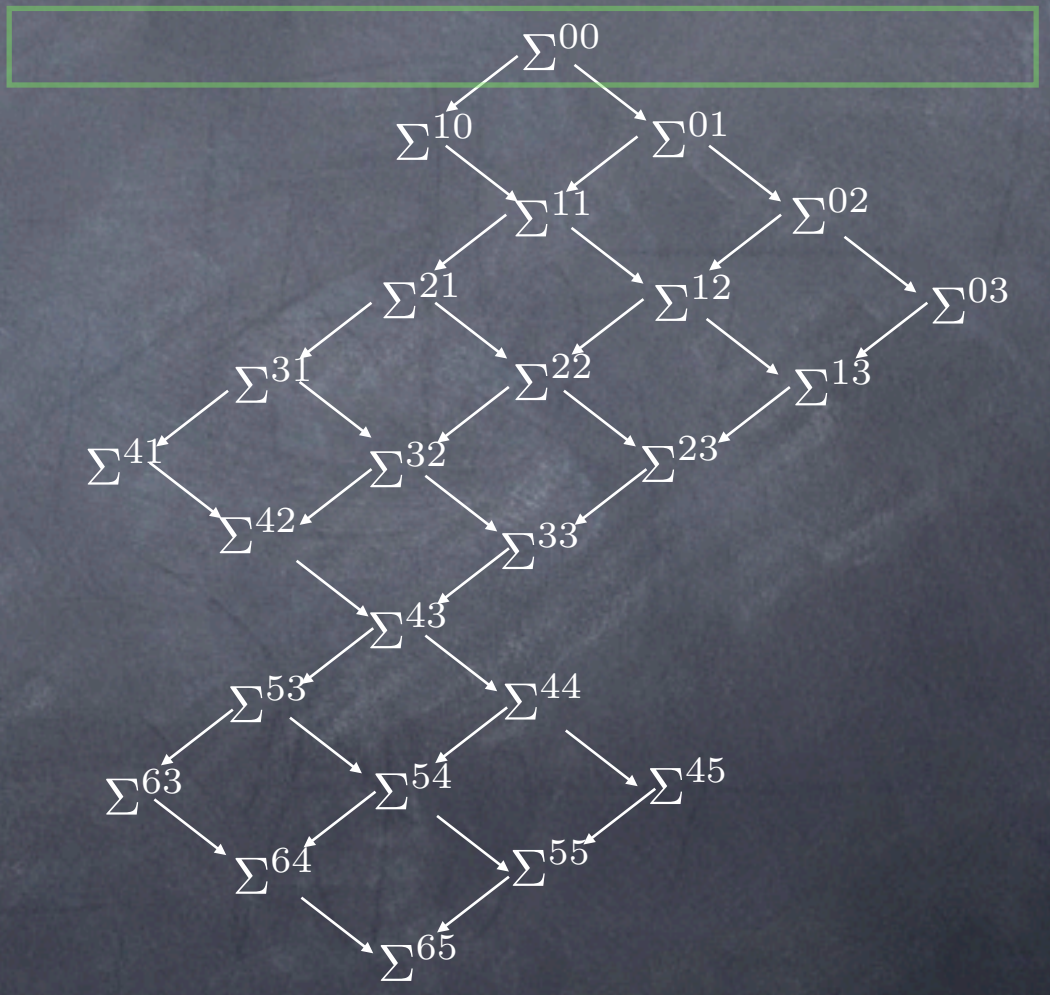
Definitely



- We know that $x = y$ has occurred, but it may not be detected if tested before Σ^{32} or after Σ^{54}
- Definitely(Φ)
  For every consistent observation O of the computation, there exists a global state of O in which Φ holds

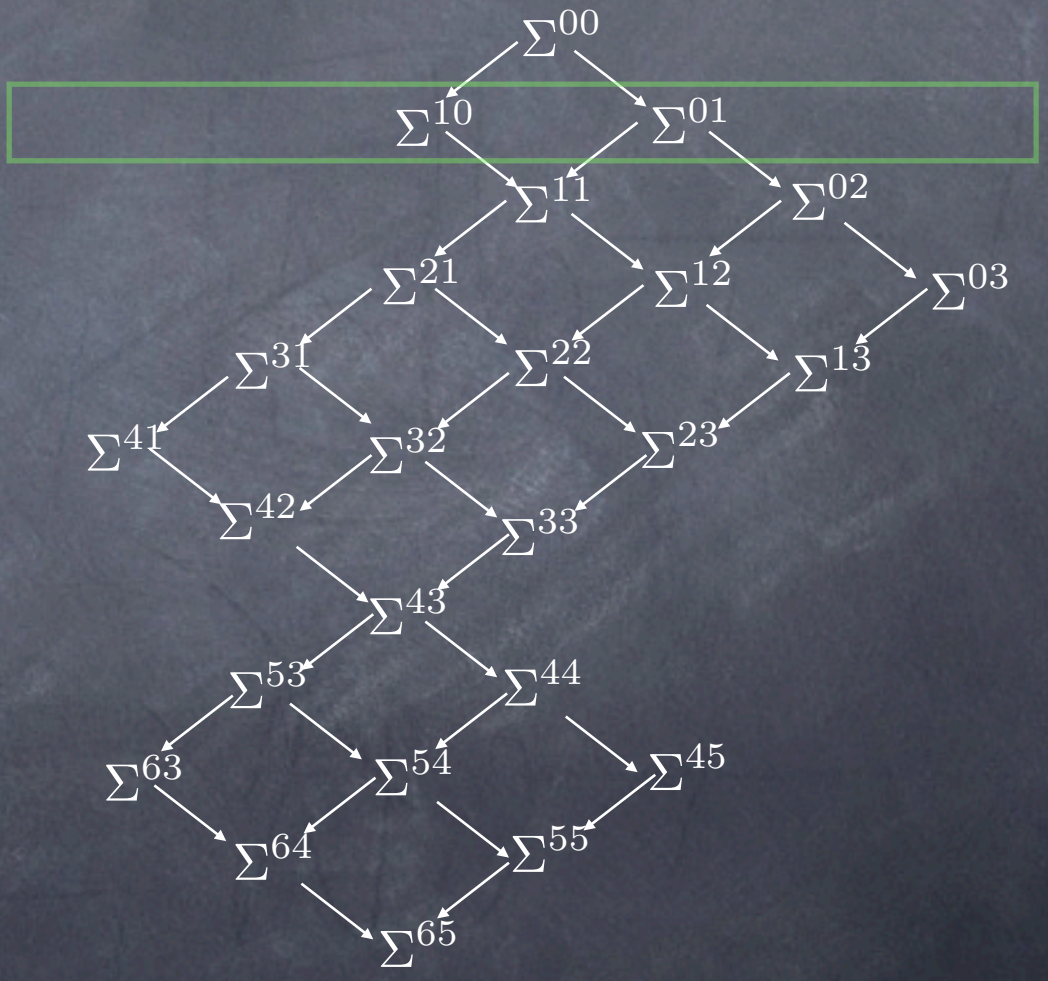
Computing Possibly

- Scan lattice, level after level
- If Φ holds in one global state, then Possibly(Φ)



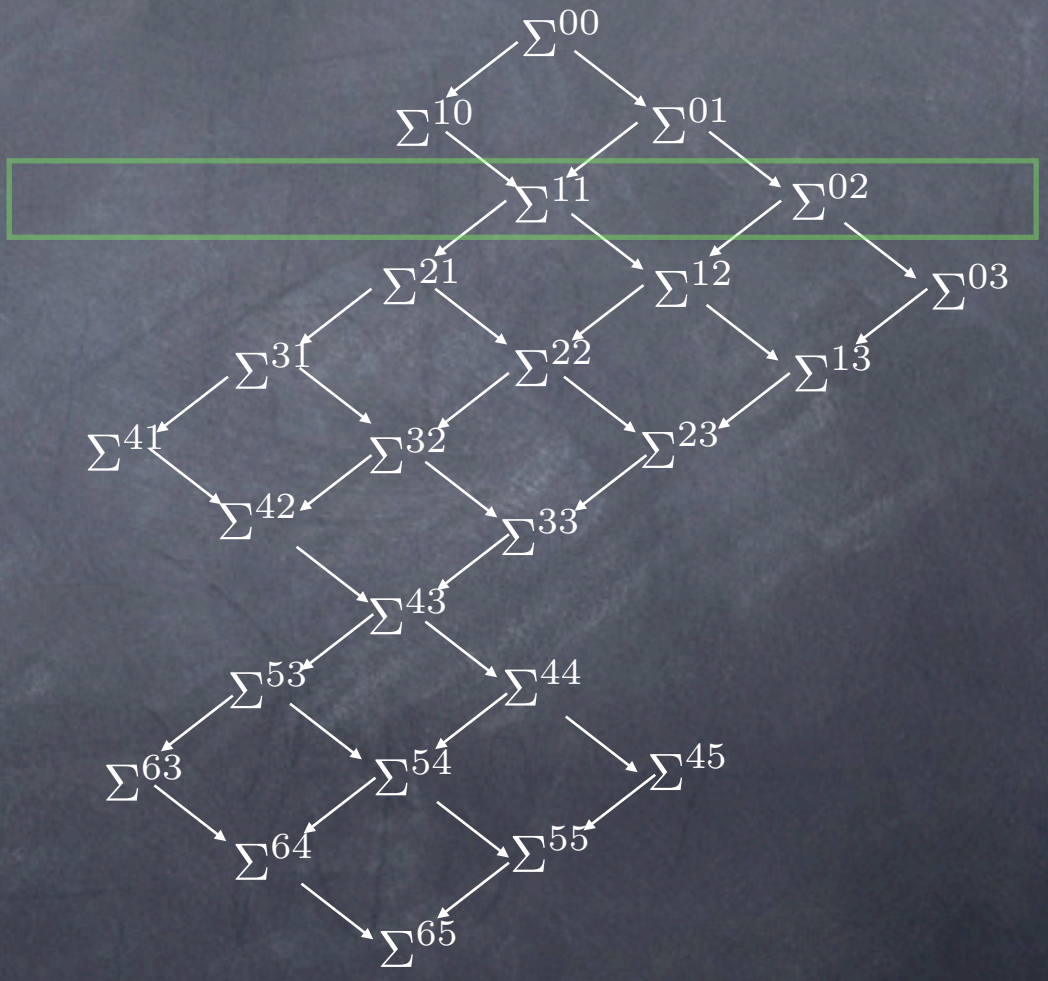
Computing Possibly

- Scan lattice, level after level
- If Φ holds in one global state, then Possibly(Φ)



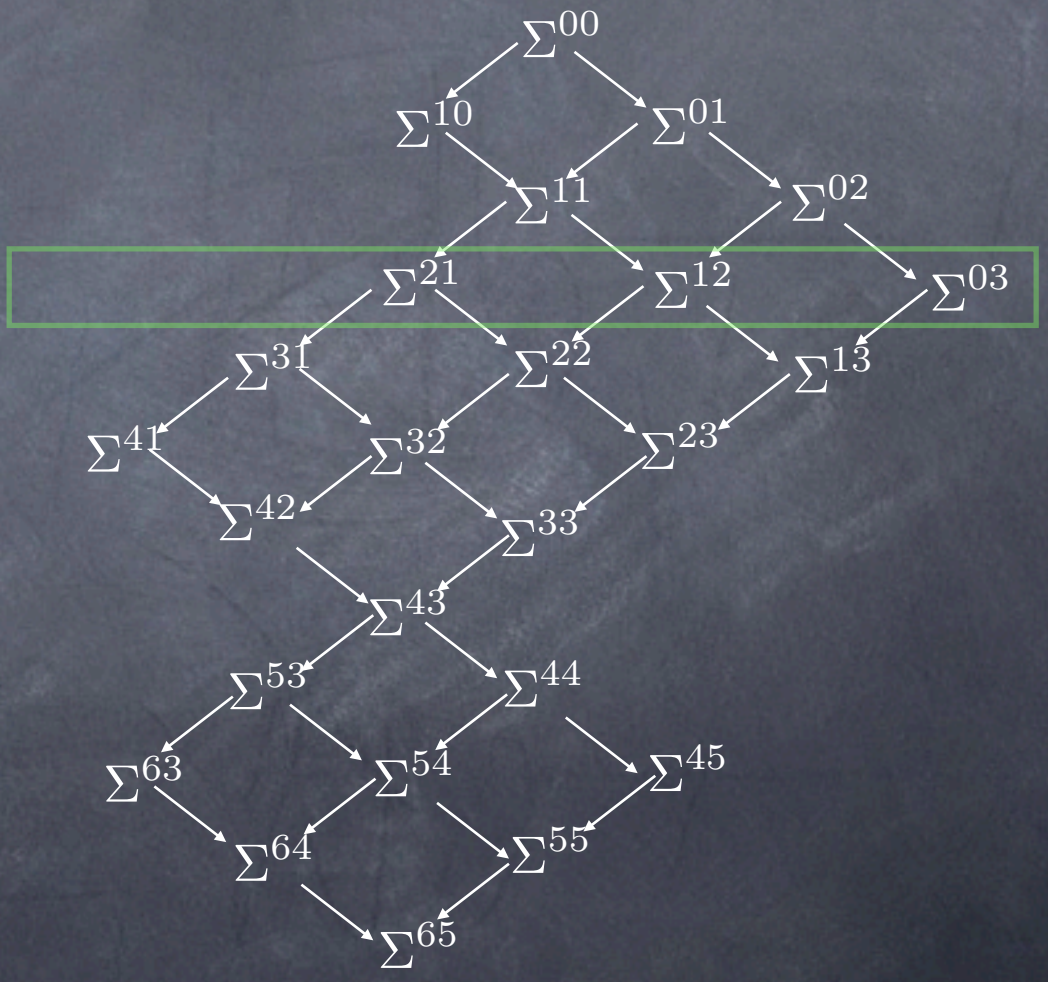
Computing Possibly

- Scan lattice, level after level
- If Φ holds in one global state, then Possibly(Φ)



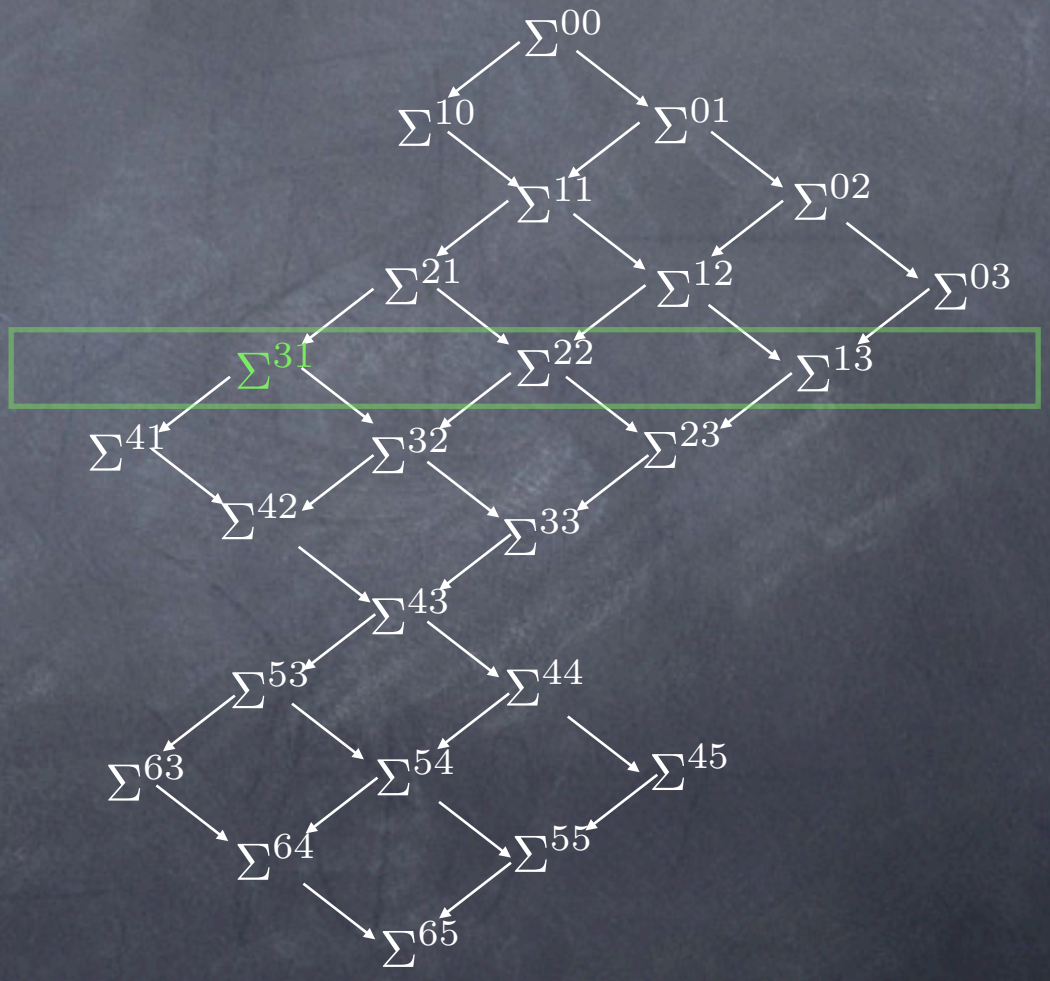
Computing Possibly

- Scan lattice, level after level
- If Φ holds in one global state, then Possibly(Φ)



Computing Possibly

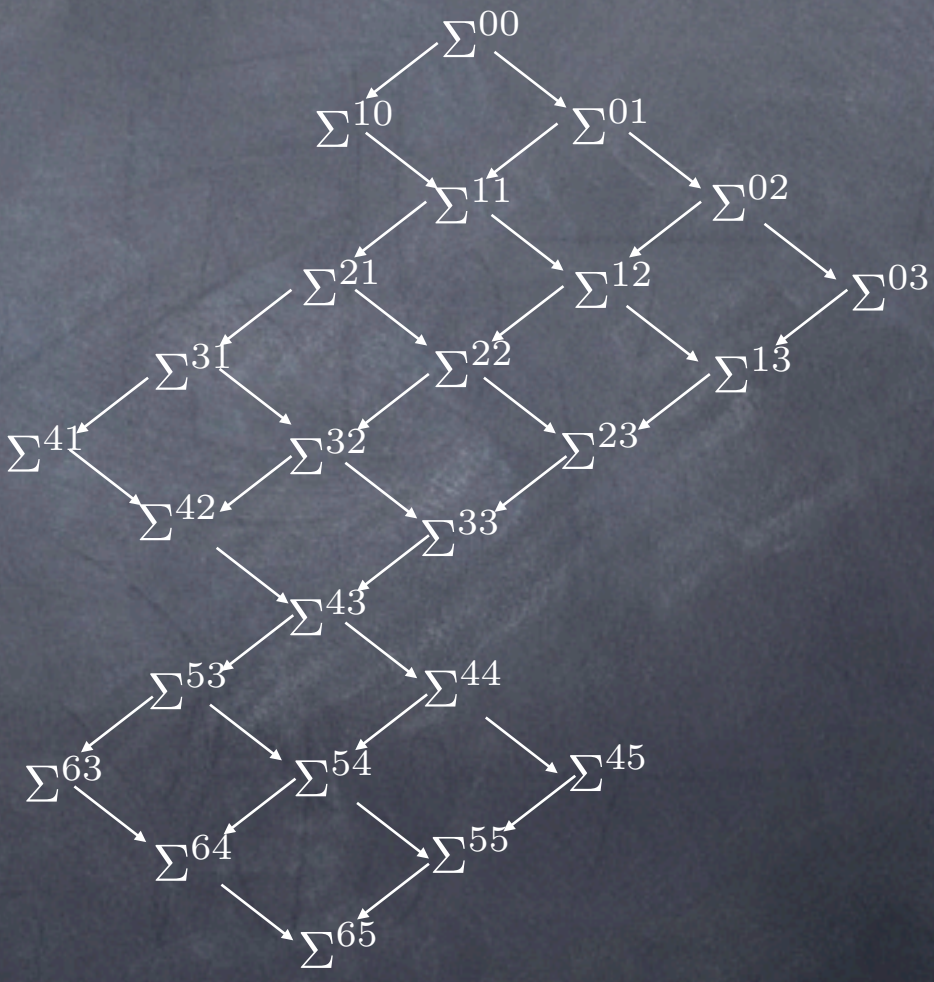
- Scan lattice, level after level
- If Φ holds in one global state, then Possibly(Φ)



Possibly$(x = y - 2)$

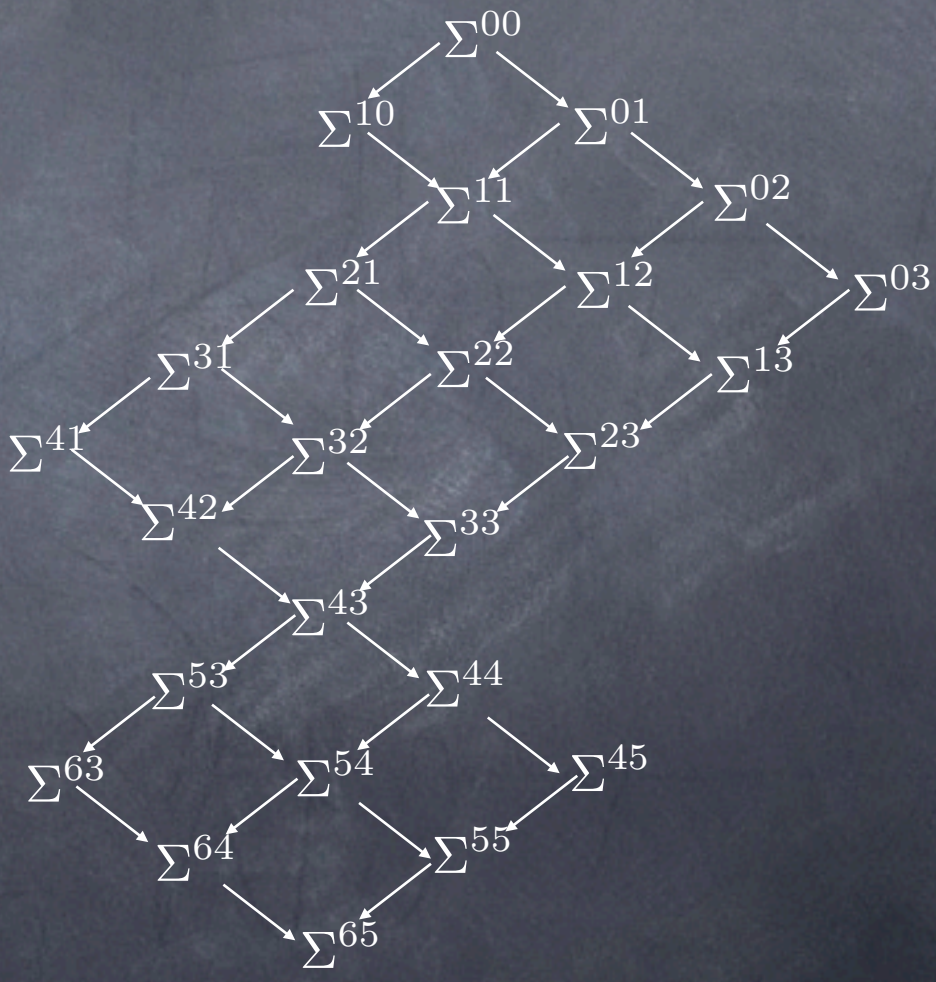
Computing Definitely

- Scan lattice, level after level



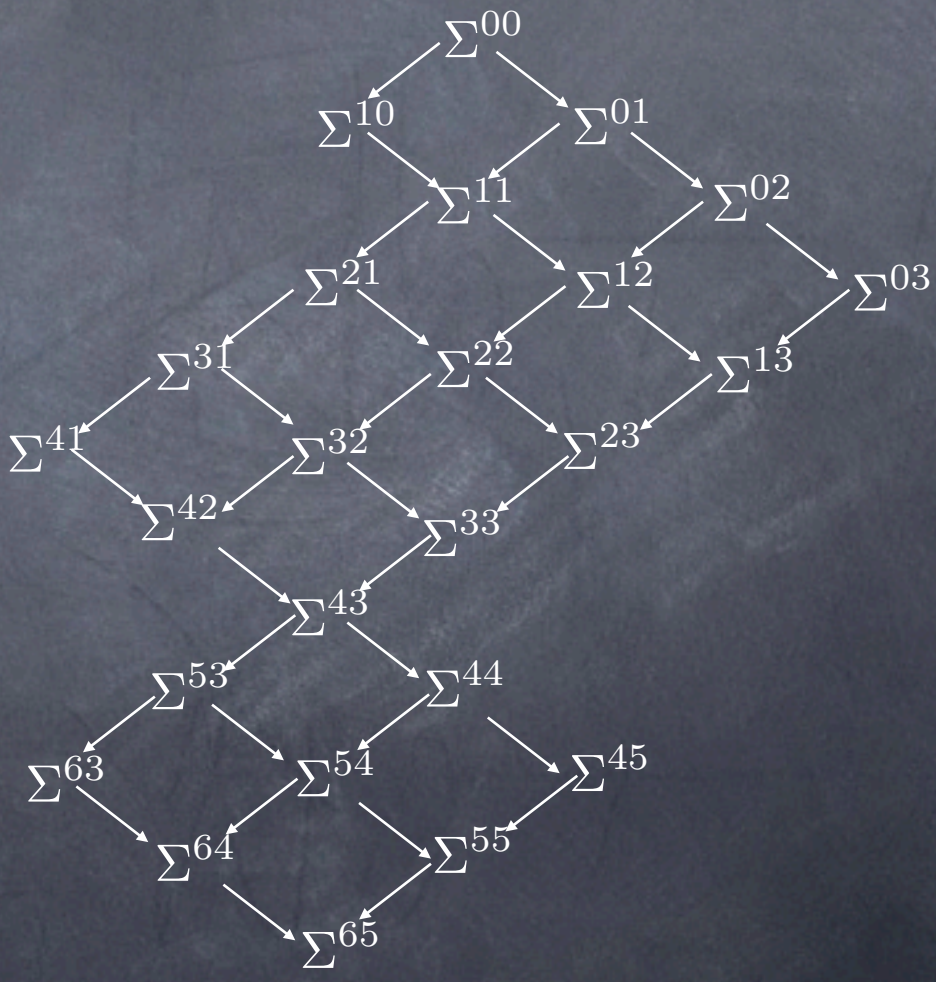
Computing Definitely

- Scan lattice, level after level
- Given a level, only expand nodes that correspond to states for which $\neg\Phi$



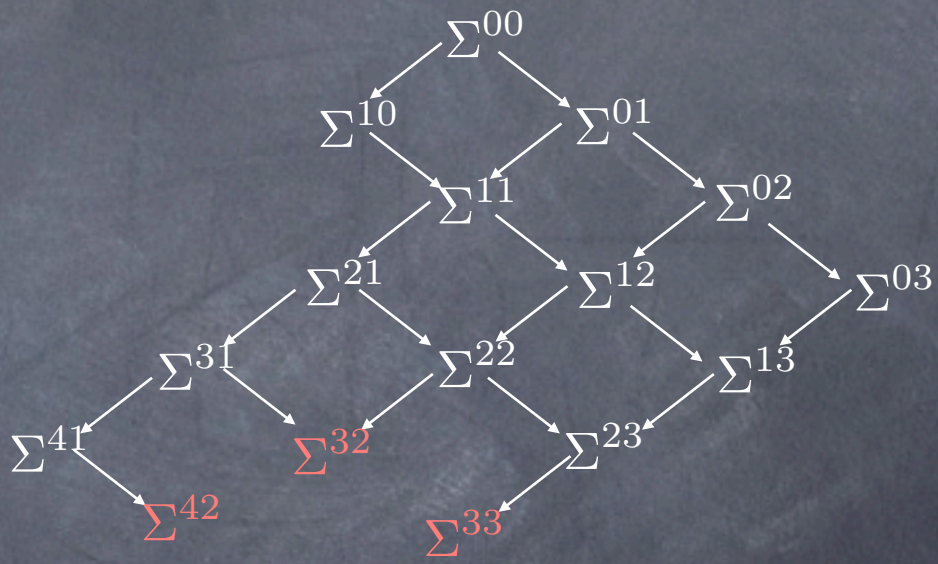
Computing Definitely

- Scan lattice, level after level
- Given a level, only expand nodes that correspond to states for which $\neg\Phi$
- If no such state, then Definitely(Φ)
- If reached last state Σ^l, and $\Phi(\Sigma^l)$, then ¬Definitely(Φ)



Computing Definitely

- Scan lattice, level after level
- Given a level, only expand nodes that correspond to states for which $\neg\Phi$
- If no such state, then Definitely(Φ)
- If reached last state Σ^l, and $\Phi(\Sigma^l)$, then ¬Definitely(Φ)



Definitely $(x = y)$

Building the lattice: collecting local states

- To build the global states in the lattice, p_0 collects local states from each process.
- p_0 keeps the set of local states received from p_i in a FIFO queue Q_i

Key questions:

1. when is it safe for p_0 to discard a local state σ_i^k of p_i?
2. Given level i of the lattice, how does one build level $i+1$?

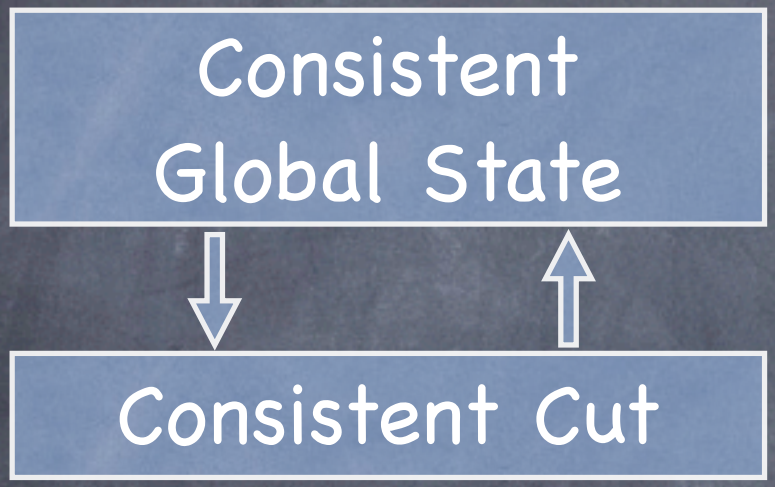
Garbage-collecting local states

- For each local state σ_i^k, we need to determine:
  - $\Sigma_{min}(\sigma_i^k)$), the earliest consistent state that σ_i^k can belong to
  - $\Sigma_{max}(\sigma_i^k)$), the latest consistent state that σ_i^k can belong to

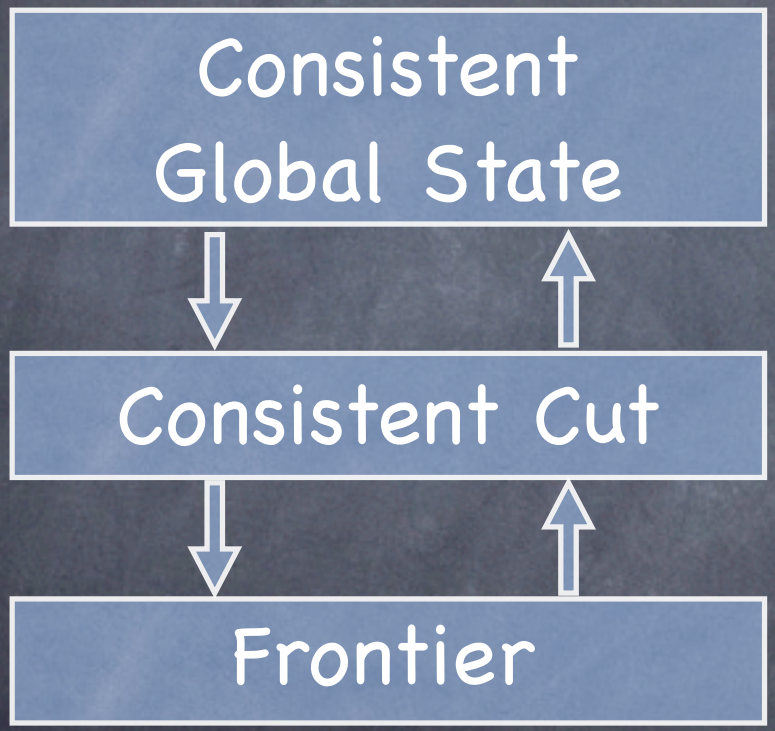
Defining "earliest" and "latest"

Consistent Global State

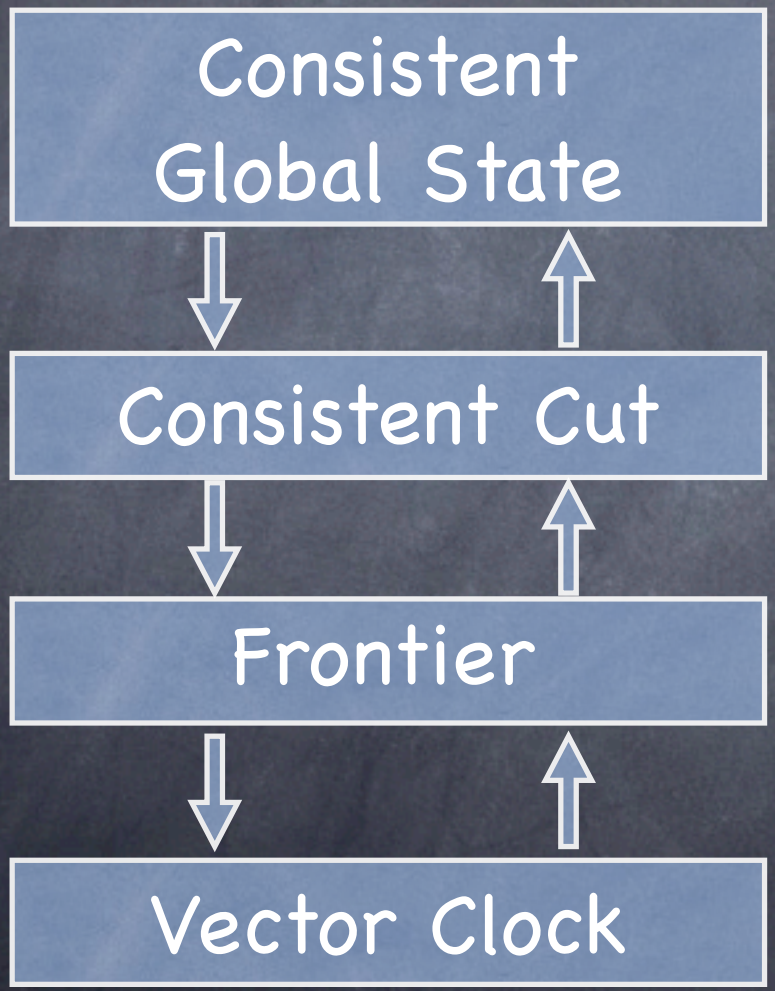
Defining "earliest" and "latest"



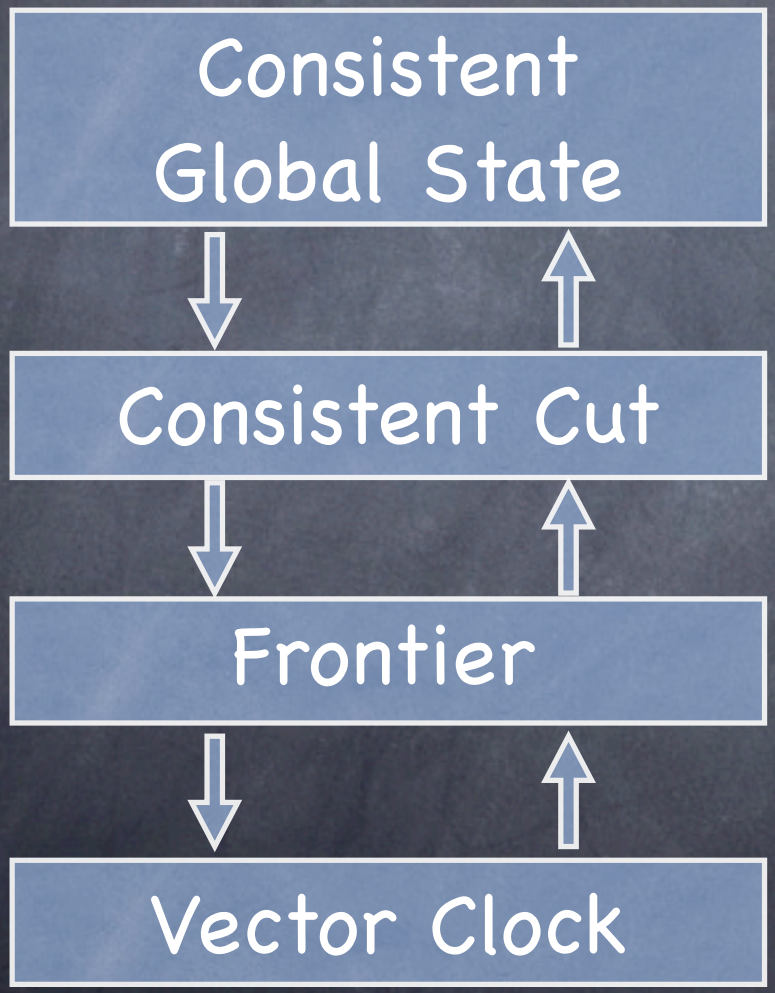
Defining "earliest" and "latest"



Defining "earliest" and "latest"



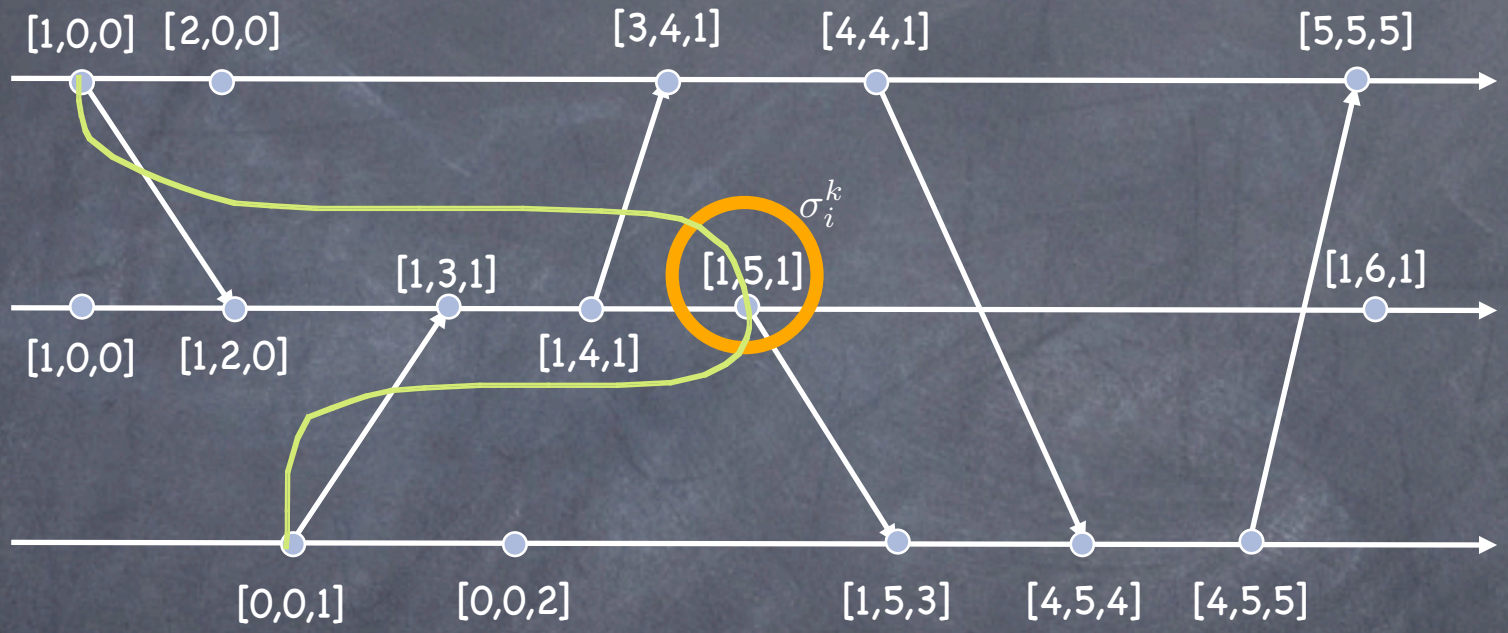
Defining "earliest" and "latest"



Associate a vector clock with each consistent global state

- $\Sigma_{min}(\sigma_i^k))$ is the consistent global state with the lowest vector clock that has σ_i^k on its frontier
- $\Sigma_{max}(\sigma_i^k))$ is the one with the highest

Computing Σ_{min}

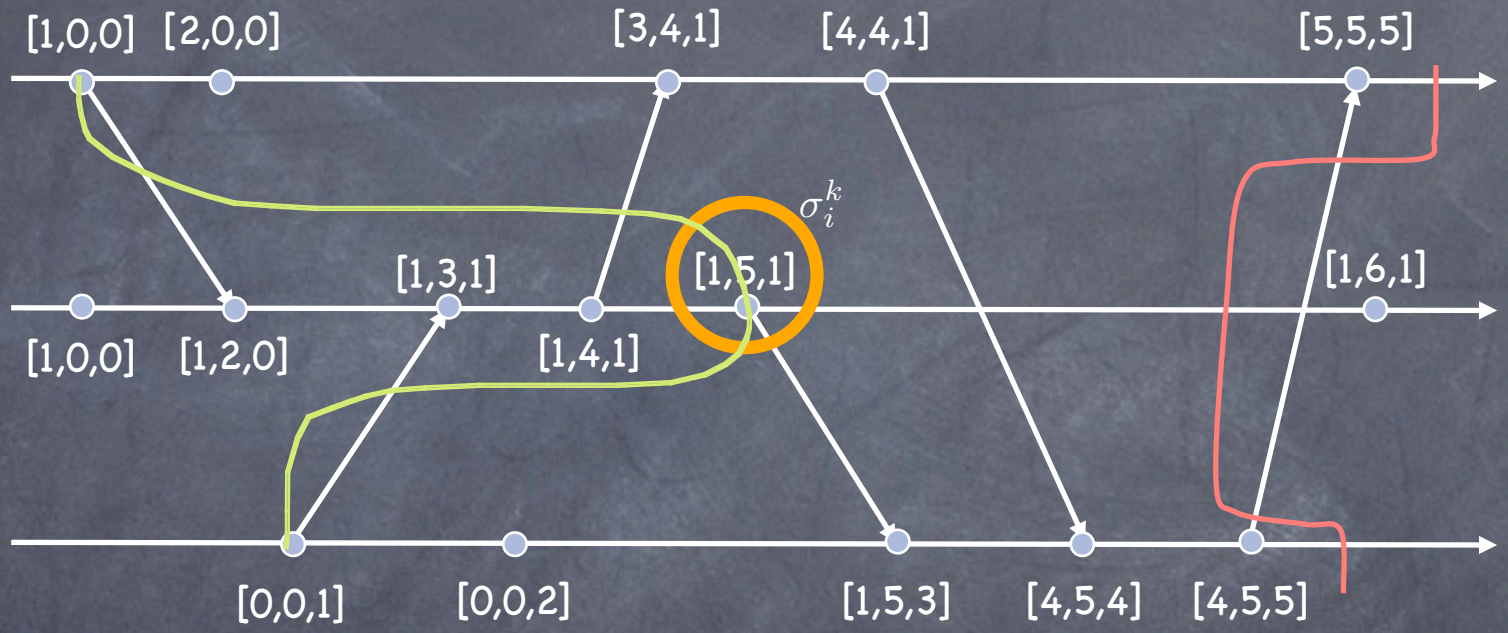


- Label σ_i^k with $VC(e_i^k)$

$$\Sigma_{min}(\sigma_i^k) = (\sigma_1^{c_1}, \sigma_2^{c_2}, \ldots, \sigma_n^{c_n}) : \forall j : c_j = VC(\sigma_i^k)[j]$$

- $\Sigma_{min}(\sigma_i^k)$ and σ_i^k have the same vector clock!

Computing Σ_{max}

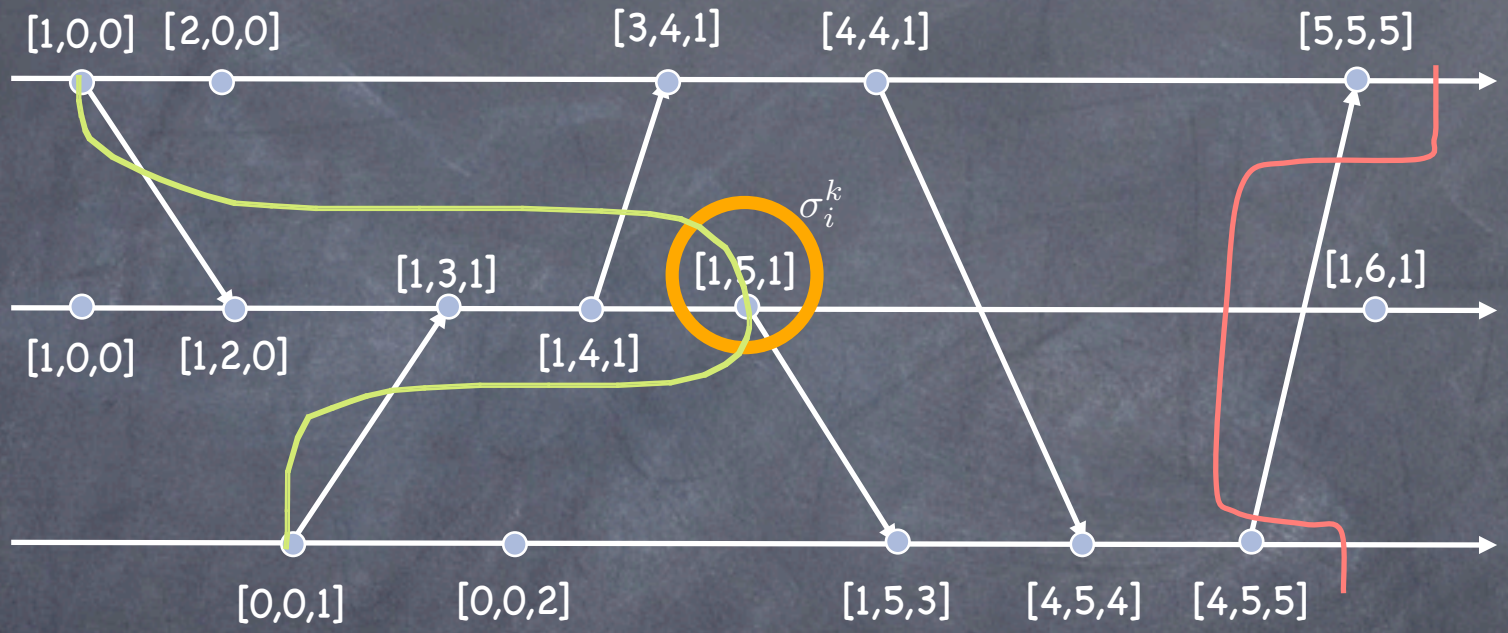


$$\Sigma_{max}(\sigma_i^k) = (\sigma_1^{c_1}, \sigma_2^{c_2}, \ldots \sigma_n^{c_n}) :$$

$$\wedge \forall j : VC(\sigma_j^{c_j})[i] \leq VC(\sigma_i^k)[i]$$

$$\wedge((\sigma_j^{c_j} = \sigma_j^{c_f}) \vee VC(\sigma_j^{c_j+1})[i] > VC(\sigma_i^k)[i]))$$

Computing Σ_{max}



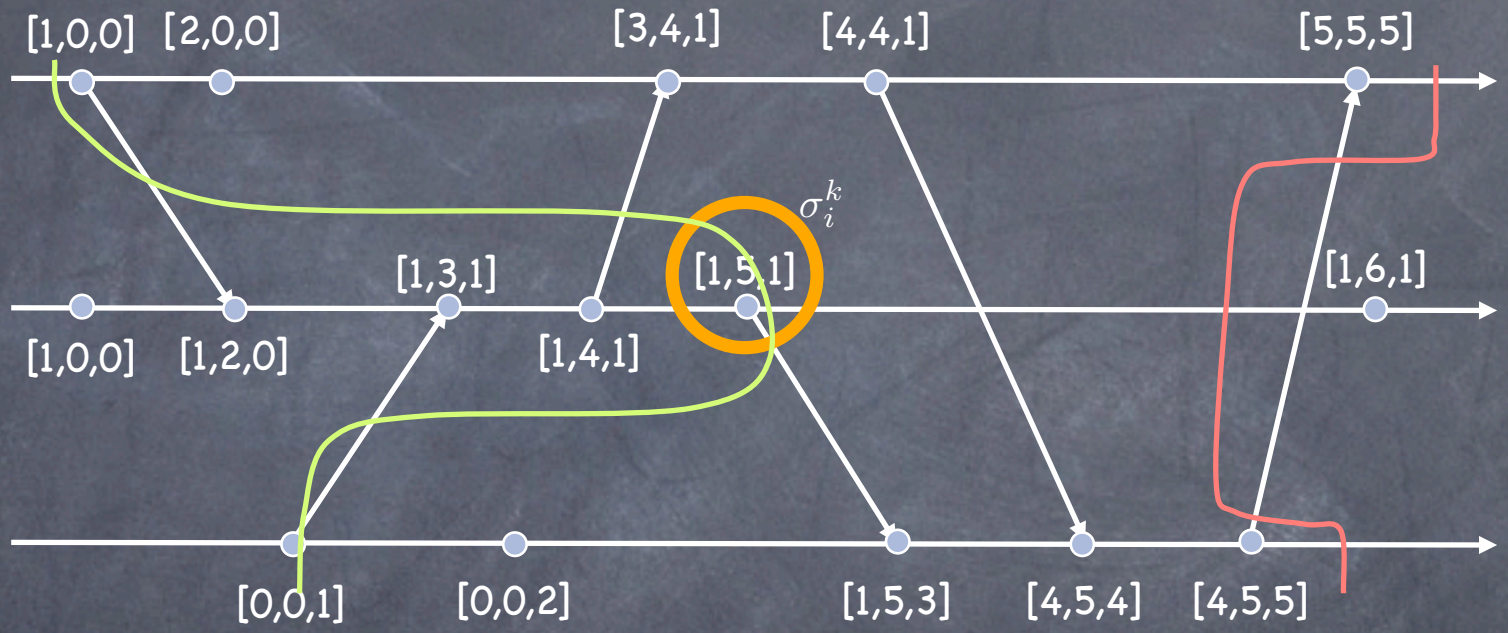
$$\Sigma_{max}(\sigma_i^k) = (\sigma_1^{c_1}, \sigma_2^{c_2}, \ldots \sigma_n^{c_n}) :$$

$$\wedge \forall j : VC(\sigma_j^{c_j})[i] \leq VC(\sigma_i^k)[i]$$

$$\wedge((\sigma_j^{c_j} = \sigma_j^{c_f}) \vee VC(\sigma_j^{c_j+1})[i] > VC(\sigma_i^k)[i]))$$

set of local states one for each process, s.t.

Computing Σ_{max}



$$\Sigma_{max}(\sigma_i^k) = (\sigma_1^{c_1}, \sigma_2^{c_2}, \ldots \sigma_n^{c_n}) :$$

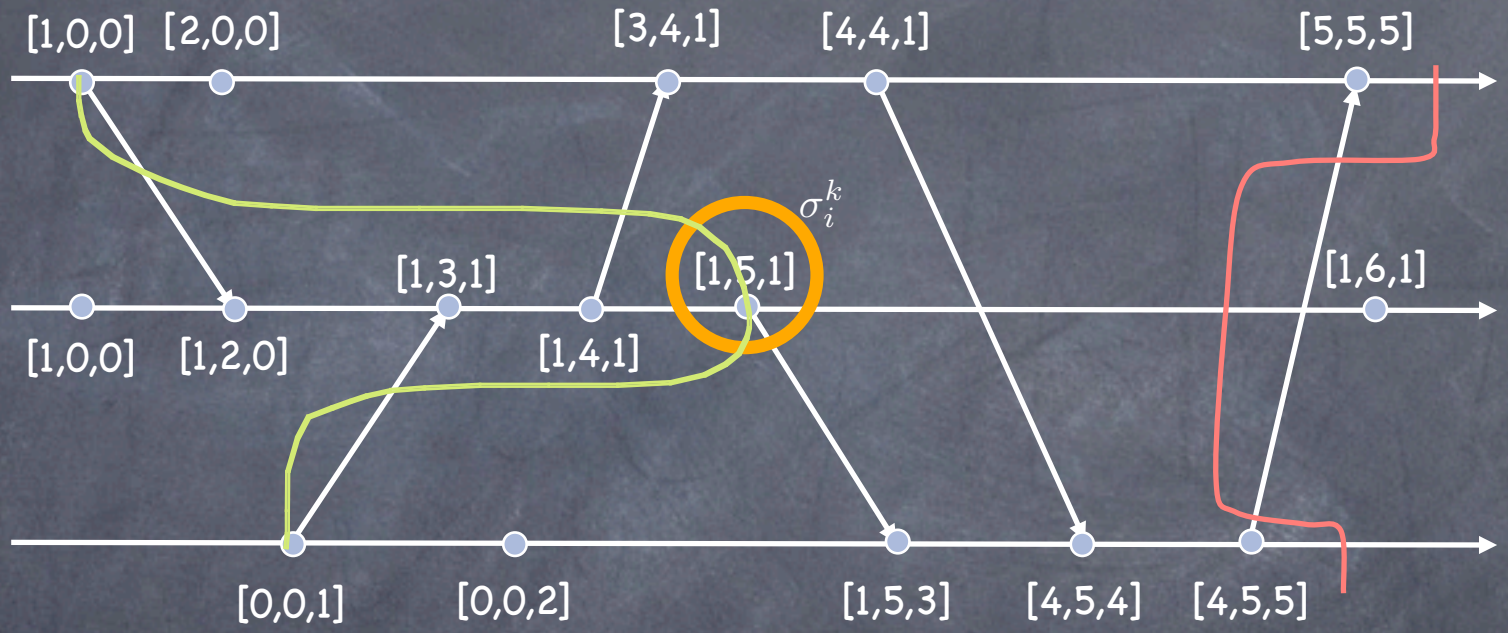
$$\wedge \forall j : VC(\sigma_j^{c_j})[i] \leq VC(\sigma_i^k)[i]$$

$$\wedge((\sigma_j^{c_j} = \sigma_j^{c_f}) \vee VC(\sigma_j^{c_j+1})[i] > VC(\sigma_i^k)[i]))$$

set of local states one for each process, s.t.

all local states are pair-wise consistent with σ_i^k

Computing Σ_{max}



$$\Sigma_{max}(\sigma_i^k) = (\sigma_1^{c_1}, \sigma_2^{c_2}, \ldots \sigma_n^{c_n}) :$$

$$\wedge \forall j : VC(\sigma_j^{c_j})[i] \leq VC(\sigma_i^k)[i]$$

$$\wedge((\sigma_j^{c_j} = \sigma_j^{c_f}) \vee VC(\sigma_j^{c_j+1})[i] > VC(\sigma_i^k)[i]))$$

set of local states one for each process, s.t.

all local states are pair-wise consistent with σ_i^k

and they are the last such state

Assembling the levels

- To build level l
  - wait until each Q_i contains a local state for whose vector clock:

$$\sum_{i=1}^{n} VC[i] \geq l$$

- To build level $l+1$
  - For each global state $\sum^{i_1,i_2,\ldots,i_n}$ on level l, build

$$\sum^{i_1+1,i_2,\ldots,i_n}, \sum^{i_1,i_2+1,\ldots,i_n}, \ldots, \sum^{i_1,i_2,\ldots,i_n+1}$$

  - Using VCs, check whether these global states are consistent