# Verification

Lecture 34

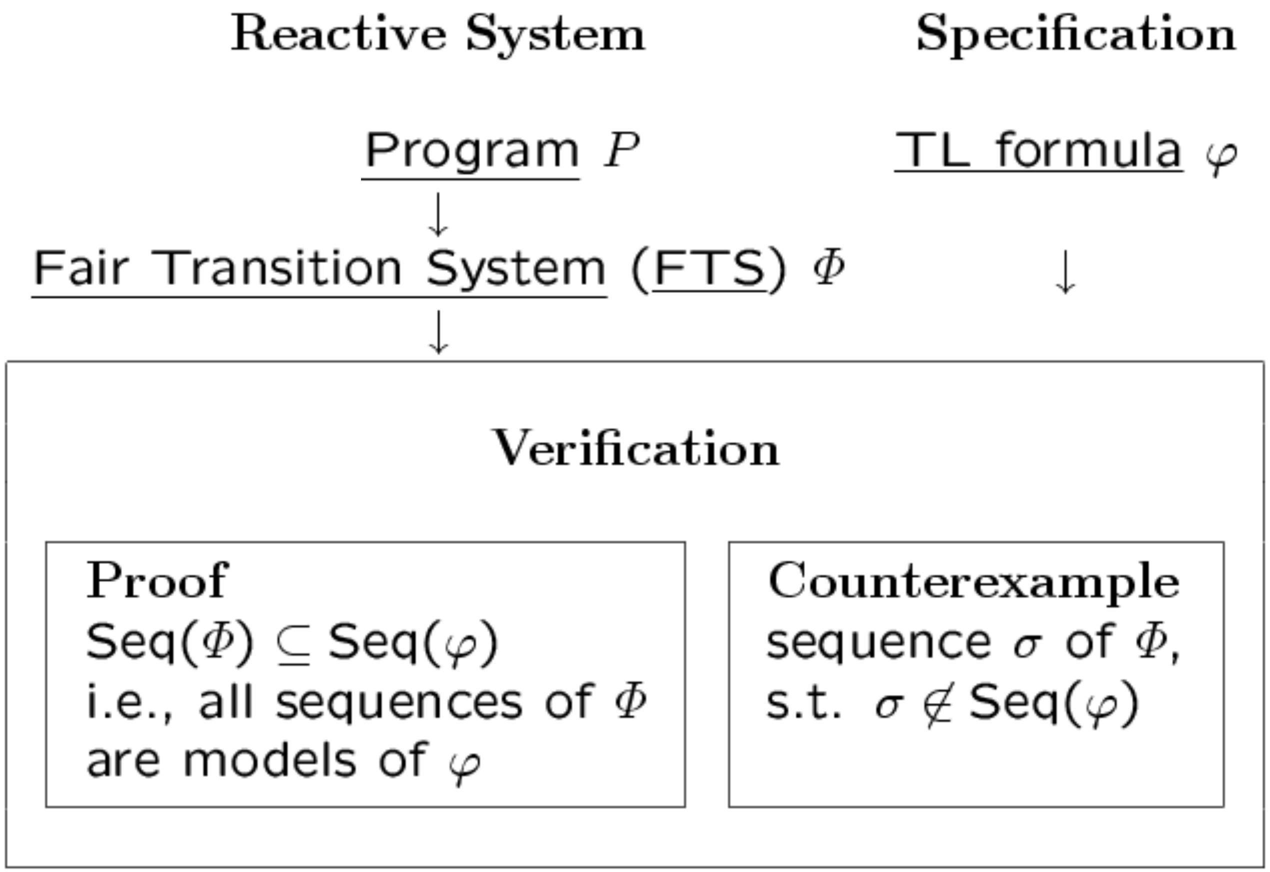
Andrey Kupriyanov, Martin Zimmermann

UNIVERSITÄT
DES
SAARLANDES

# Plan for today

- Deductive verification
    - The SLAB Model Checker

# Deductive verification of reactive systems



# Symbolic Transition Systems

- A (finite) set of variables $V \subseteq \mathcal{V}$
  System variables: data variables + control variables

- Initial condition $\theta$
  first-order assertion over $V$
  that characterizes all initial states

- A (finite) set of transitions $\mathcal{T}$

For each $\tau \in$ : $\tau$: $\Sigma \mapsto 2^{\Sigma}$

$\tau$ is represented by the transition relation $\rho(\tau)$
(next-state relation)

# Inductive Assertions

For assertion $q$,

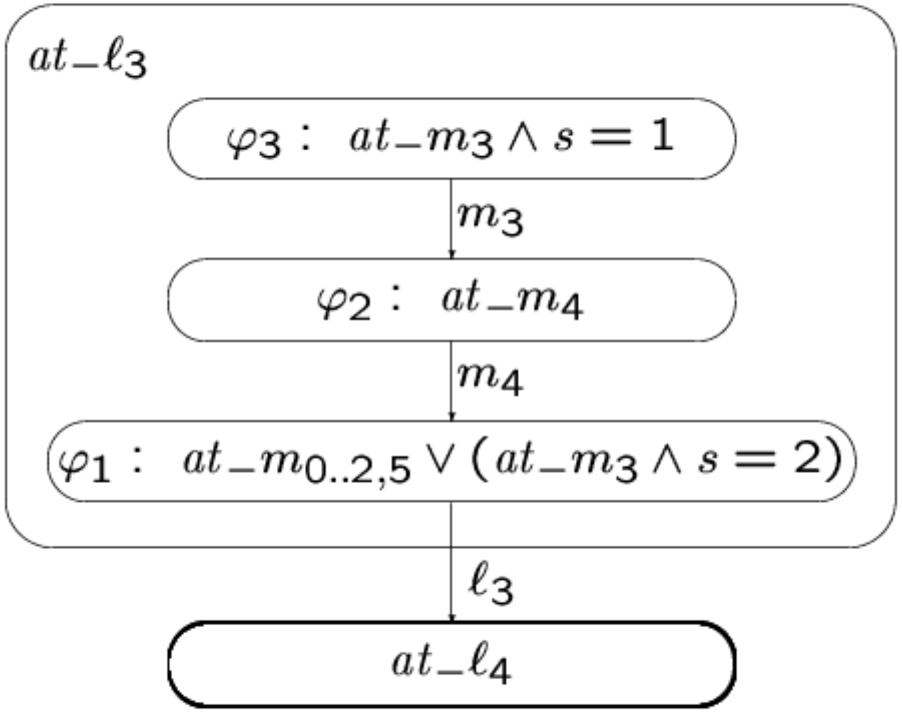
$$\frac{\begin{array}{ll}\text{B1.} & P \Vdash \Theta \rightarrow q \\ \text{B2.} & P \Vdash \{q\}\ \mathcal{T}\ \{q\}\end{array}}{P \vDash \square q} \quad \text{B-INV}$$

- $q$ is <u>inductive</u> if B1 and B2 are (state) valid
- By rule B-INV,
  <u>every inductive assertion $q$ is $P$-invariant</u>
- <u>The converse is not true</u>

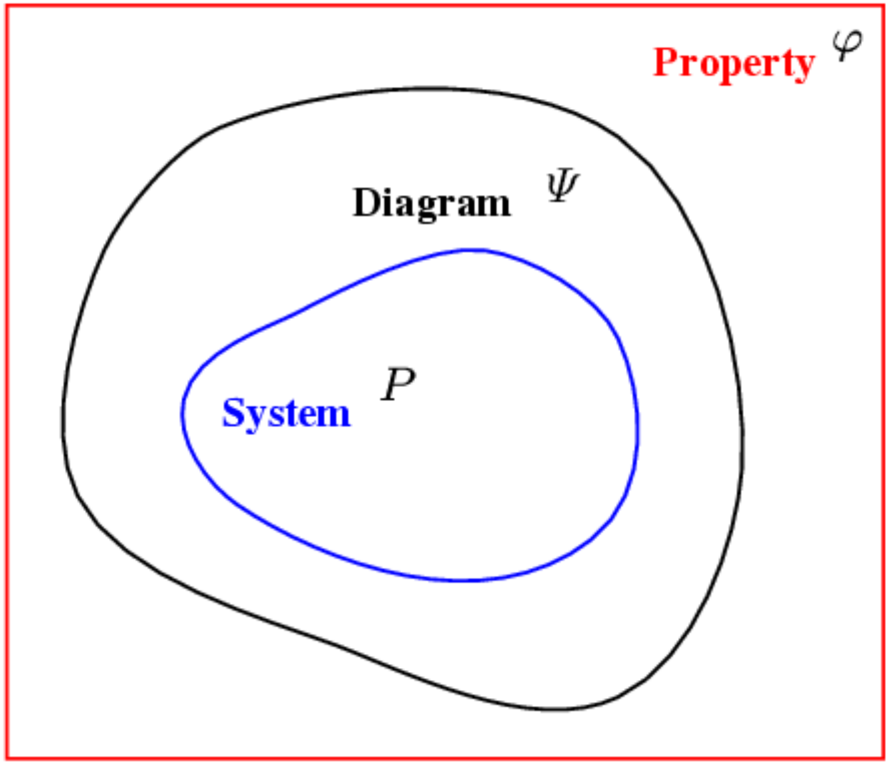
# Verification Diagrams

Verification diagrams allow a graphical representation of a proof of a temporal property.

Example:



# Idea




$\mathcal{L}(P) \subseteq \mathcal{L}(\Psi)$ proved by verification conditions.

$\mathcal{L}(\Psi) \subseteq \mathcal{L}(\varphi)$ follows from well-formedness of diagram

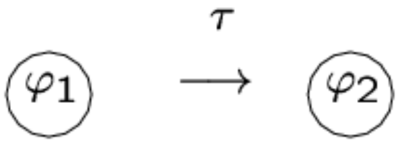
# P-Valid Verification Diagrams

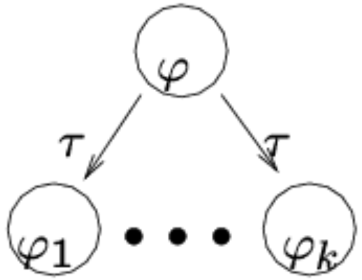
Directed labeled graph with

Verification conditions

Nodes – labeled by assertions


Edges – labeled by names of transitions




$$\Rightarrow \ \{\varphi\} \ \tau \ \{\varphi \vee \varphi_1 \vee \ldots \vee \varphi_k\}$$

Terminal Node ("goal") – no edges depart from it


Definition: VD is $P$-valid iff all VCs associated with nodes in the diagram are $P$-state valid

# Invariance Diagrams

VDs with no terminal nodes (cycles OK)

Claim (invariance diagram):

A $P$-valid INVARIANCE diagram establishes that

$$\bigvee_{j=1}^{m} \varphi_j \quad \Rightarrow \quad \Box(\bigvee_{j=1}^{m} \varphi_j)$$

is $P$-valid.

If, in addition,

$$(I1) \quad \bigvee_{j=1}^{m} \varphi_j \rightarrow q$$

$$(I2) \quad \Theta \rightarrow \bigvee_{j=1}^{m} \varphi_j$$

are $P$-state valid, then

$$\Box q \text{ is } P\text{-valid}$$

# Example

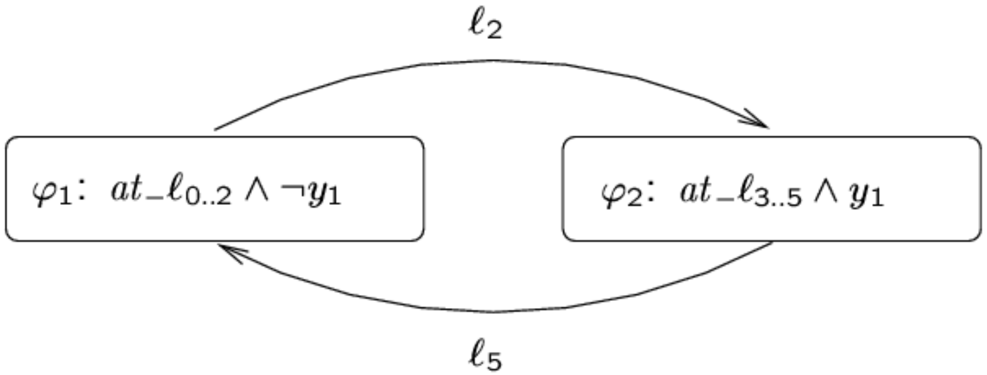
local $y_1, y_2$: boolean where $y_1 = \text{F}, y_2 = \text{F}$
$s$ : integer where $s = 1$

$P_1 ::$ $\ell_0$ : loop forever do
$\left[\begin{array}{ll} \ell_1: & \text{noncritical} \\ \ell_2: & (y_1,\ s) := (\text{T},\ 1) \\ \ell_3: & \text{await } (\neg y_2) \lor (s = 2) \\ \ell_4: & \text{critical} \\ \ell_5: & y_1 := \text{F} \end{array}\right]$

||

$P_2 ::$ $m_0$ : loop forever do
$\left[\begin{array}{ll} m_1: & \text{noncritical} \\ m_2: & (y_2,\ s) := (\text{T},\ 2) \\ m_3: & \text{await } (\neg y_1) \lor (s = 1) \\ m_4: & \text{critical} \\ m_5: & y_2 := \text{F} \end{array}\right]$

- INVARIANCE diagram valid for program MUX-PET1




- Also,

(I2) $\underbrace{at\_\ell_0 \ \land\ \neg y_1 \ \land\ \cdots}_{\Theta} \ \rightarrow$
$\underbrace{at\_\ell_{0..2} \ \land\ \neg y_1}_{\varphi_1} \ \lor\ \underbrace{\cdots}_{\varphi_2}$

(I1) $\underbrace{at\_\ell_{0..2} \ \land\ \neg y_1}_{\varphi_1} \ \rightarrow\ \underbrace{y_1 \ \leftrightarrow\ at\_\ell_{3..5}}_{q}$

$\underbrace{at\_\ell_{3..5} \ \land\ y_1}_{\varphi_2} \ \rightarrow\ \underbrace{y_1 \ \leftrightarrow\ at\_\ell_{3..6}}_{q}$

Therefore

$$\Box(y_1 \ \leftrightarrow\ at\_\ell_{3..5})$$

# Abstraction

**local** $y_1, y_2$ : **integer** **where** $y_1 = y_2 = 0$

**loop forever do**

- $\ell_0$: **noncritical**
- $\ell_1$: $y_1 := y_2 + 1$
- $\ell_2$: **await** $(y_2 = 0 \lor y_1 \leq y_2)$
- $\ell_3$: **critical**
- $\ell_4$: $y_1 := 0$

||

**loop forever do**

- $m_0$: **noncritical**
- $m_1$: $y_2 := y_1 + 1$
- $m_2$: **await** $(y_1 = 0 \lor y_2 < y_1)$
- $m_3$: **critical**
- $m_4$: $y_2 := 0$

**local** $b_1, b_2, b_3$ : **boolean** **where** $b_1, b_2, b_3$

**loop forever do**

- $\ell_0$: **noncritical**
- $\ell_1$: $(b_1, b_3) := (\mathit{false}, \mathit{false})$
- $\ell_2$: **await** $(b_2 \lor b_3)$
- $\ell_3$: **critical**
- $\ell_4$: $(b_1, b_3) := (\mathit{true}, \mathit{true})$

||

**loop forever do**

- $m_0$: **noncritical**
- $m_1$: $(b_2, b_3) := (\mathit{false}, \mathit{true})$
- $m_2$: **await** $(b_1 \lor \neg b_3)$
- $m_3$: **critical**
- $m_4$: $(b_2, b_3) := (\mathit{true}, b_1)$

# REVIEW: Simulation order

Let $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP, L_i)$ , $i$=1, 2, be two transition systems over $AP$.

A simulation for $(TS_1, TS_2)$ is a binary relation $\mathcal{R} \subseteq S_1 \times S_2$ such that:

1. $\forall q_1 \in I_1 \; \exists q_2 \in I_2. \; (q_1, q_2) \in \mathcal{R}$
2. for all $(q_1, q_2) \in \mathcal{R}$ it holds:

   2.1 $L_1(q_1) = L_2(q_2)$

   2.2 if $q_1' \in Post(q_1)$
   then there exists $q_2' \in Post(q_2)$ with $(q_1', q_2') \in \mathcal{R}$

# REVIEW: Simulation order and $\forall$CTL*

Let $TS$ be a finite transition system (without terminal states) and $q, q'$ states in $TS$.

The following statements are equivalent:

(1) $q \preceq_{TS} q'$

(2) for all $\forall$CTL$^*$-formulas $\Phi$: $q' \models \Phi$ implies $q \models \Phi$

(3) for all $\forall$CTL-formulas $\Phi$: $q' \models \Phi$ implies $q \models \Phi$

# Predicate Abstraction

- Abstraction is determined by a set of predicates,

  $$P=\{\phi_1, \phi_2, \ldots \phi_N\}$$

- Abstract state space: subsets of $P$

- Abstraction function $f(q) = \{\phi_i \mid q \models \phi_i\}$

# Example

**local** $y_1, y_2$ : **integer**
**where** $y_1 = y_2 = 0$

**loop forever do**

- $\ell_0$: **noncritical** 
- $\ell_1$: $y_1 := y_2 + 1$
- $\ell_2$: **await** $(y_2 = 0 \lor y_1 \leq y_2)$
- $\ell_3$: **critical** 
- $\ell_4$: $y_1 := 0$

||

**loop forever do**

- $m_0$: **noncritical** 
- $m_1$: $y_2 := y_1 + 1$
- $m_2$: **await** $(y_1 = 0 \lor y_2 < y_1)$
- $m_3$: **critical** 
- $m_4$: $y_2 := 0$

Predicates:

guards of transitions

$P = \{b_1, b_2, b_3\}$ +
control predicates

with

$b_1$: y1 = 0

$b_2$: y2 = 0

$b_3$: y1 $\leq$ y2

# Example

**local** $y_1, y_2$ : **integer** **where** $y_1 = y_2 = 0$

**loop forever do**

- $\ell_0$: **noncritical**
- $\ell_1$: $y_1 := y_2 + 1$
- $\ell_2$: **await** $(y_2 = 0 \lor y_1 \leq y_2)$
- $\ell_3$: **critical**
- $\ell_4$: $y_1 := 0$

||

**loop forever do**

- $m_0$: **noncritical**
- $m_1$: $y_2 := y_1 + 1$
- $m_2$: **await** $(y_1 = 0 \lor y_2 < y_1)$
- $m_3$: **critical**
- $m_4$: $y_2 := 0$

**local** $b_1, b_2, b_3$ : **boolean** **where** $b_1, b_2, b_3$

**loop forever do**

- $\ell_0$: **noncritical**
- $\ell_1$: $(b_1, b_3) := (\mathit{false}, \mathit{false})$
- $\ell_2$: **await** $(b_2 \lor b_3)$
- $\ell_3$: **critical**
- $\ell_4$: $(b_1, b_3) := (\mathit{true}, \mathit{true})$

||

**loop forever do**

- $m_0$: **noncritical**
- $m_1$: $(b_2, b_3) := (\mathit{false}, \mathit{true})$
- $m_2$: **await** $(b_1 \lor \neg b_3)$
- $m_3$: **critical**
- $m_4$: $(b_2, b_3) := (\mathit{true}, b_1)$

# Example

This abstraction allows us to prove

- mutual exclusion
- bounded overtaking

using a model checker, since it is a finite-state program.

**local** $b_1, b_2, b_3$ : **boolean** **where** $b_1, b_2, b_3$

**loop forever do**

$\ell_0$: **noncritical**

$\ell_1$: $(b_1, b_3) := (\mathit{false}, \mathit{false})$

$\ell_2$: **await** $(b_2 \vee b_3)$

$\ell_3$: **critical**

$\ell_4$: $(b_1, b_3) := (\mathit{true}, \mathit{true})$

||

**loop forever do**

$m_0$: **noncritical**

$m_1$: $(b_2, b_3) := (\mathit{false}, \mathit{true})$

$m_2$: **await** $(b_1 \vee \neg b_3)$

$m_3$: **critical**

$m_4$: $(b_2, b_3) := (\mathit{true}, b_1)$

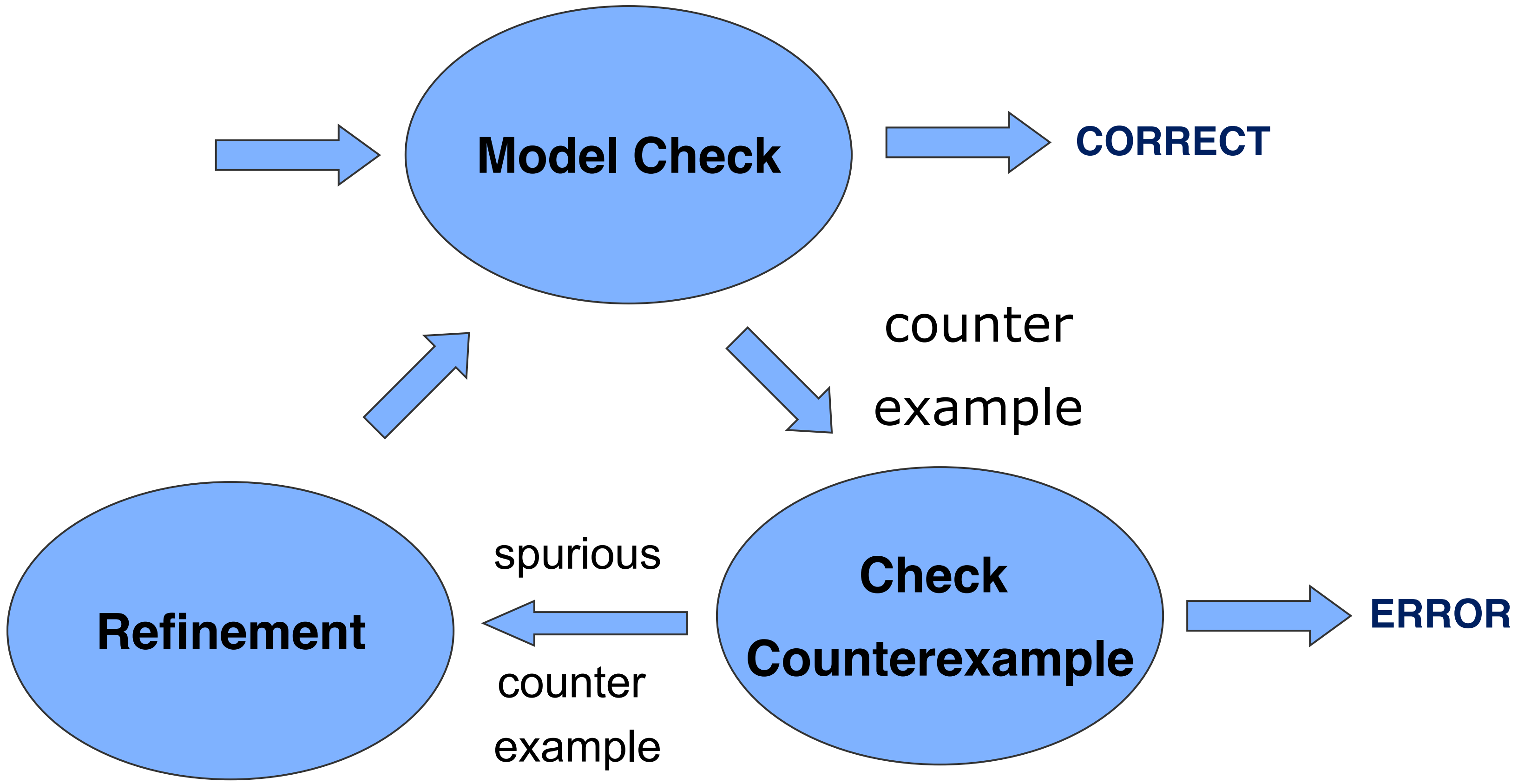
# How To Determine the Basis?

A good starting set:

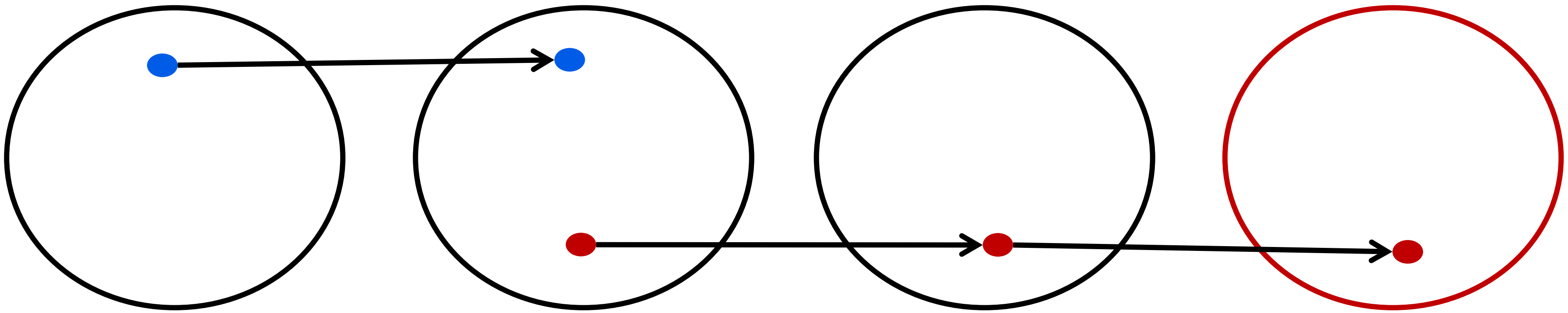
- The atomic assertions appearing in the guards of the transitions ($\rightarrow$ enabling conditions can be represented exactly, and thus fairness carries over)
- The atomic assertions appearing in the property to be proven ($\rightarrow$ the property abstraction is exact)

Analysis of counterexamples may lead to refinement of the abstraction by adding more assertions to the basis.

# Counter Example Guided Abstraction Refinement (CEGAR)



# Spurious counter examples



# Checking abstract error paths

Let $E$ be an assertion indicating an error state.

An abstract counter example $x_0\ x_1\ \ldots\ x_k$ is **concretizable** if there exists a sequence of concrete states $s_0\ s_1\ \ldots\ s_k$ such that

1. For each $0 \leq i \leq k$, $f(s_i)=x_k$.
2. $s_0 \models \Theta$ and $s_k \models E$
3. For each $0 \leq i < k$, $(s_i, s_{i+1}) \models \rho$

# Checking abstract error paths

1. For each $0 \leq i \leq k$, $f(s_i)=x_k$.
2. $s_0 \models \Theta$ and $s_k \models E$
3. For each $0 \leq i < k$, $(s_i,s_{i+1}) \models \rho$

represented as a formula:

$$\Theta(V^0) \wedge \bigwedge_{i=0..k} \bigwedge_{\phi \in x_i} \phi\,(V^i) \wedge \bigwedge_{i=0..k-1} \rho(V^i,V^{i+1}) \wedge E(V^k)$$

# Craig Interpolation

For a given pair of formulas $\varphi(X)$ and $\psi(Y)$
such that $\varphi \wedge \psi$ is unsatisfiable,

a **Craig interpolant** $\Delta(X \cap Y)$ is a formula
over the common variables
such that

$\varphi$ implies $\Delta$ and
$\Delta \wedge \psi$ is unsatisfiable.

Craig interpolants can be automatically generated for many
first-order theories.

# Path cutting

Split formula

$$\Theta(V^0) \wedge \bigwedge_{i=0..k} \bigwedge_{\phi \in x_i} \phi\,(V^i) \quad \wedge \bigwedge_{i=0..k-1} \rho(V^i,V^{i+1}) \quad \wedge\; E(V^k)$$

into two parts:

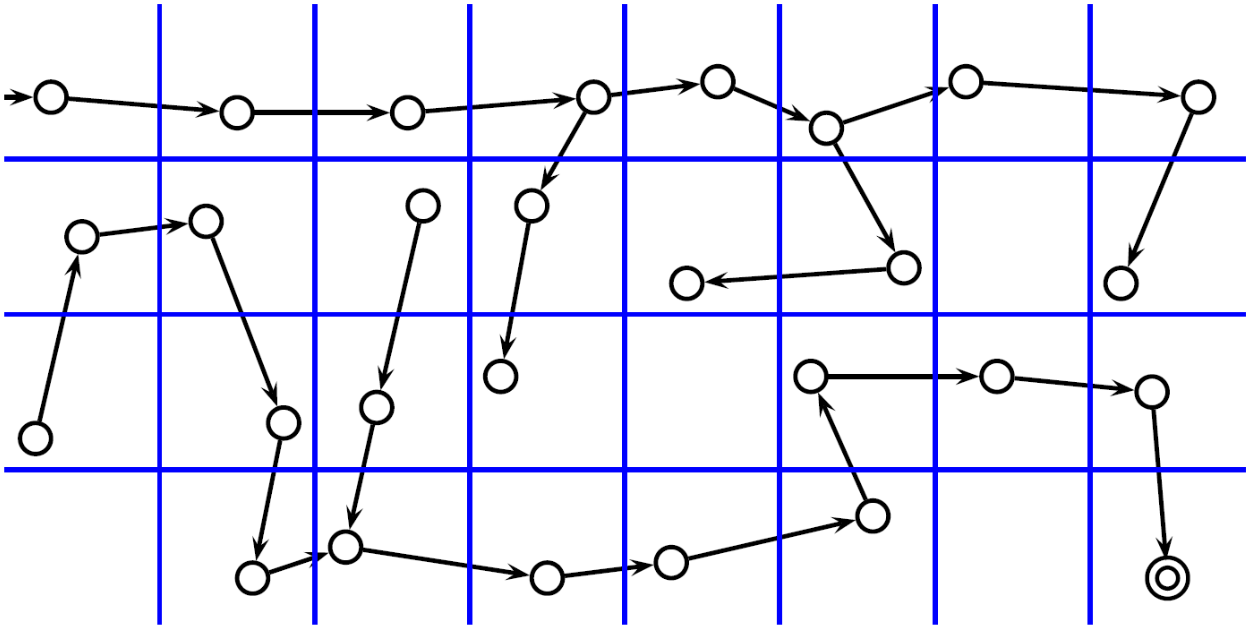
$$\phi_1 = \Theta(V^0) \wedge \bigwedge_{i=0..j-1} \bigwedge_{\phi \in x_i} \phi\,(V^i) \quad \wedge \bigwedge_{i=0..j-2} \rho(V^i,V^{i+1})$$

$$\phi_2 = \bigwedge_{i=j..k} \bigwedge_{\phi \in x_i} \phi\,(V^i) \quad \wedge \bigwedge_{i=j-1..k-1} \rho(V^i,V^{i+1}) \quad \wedge\; E(V^k)$$

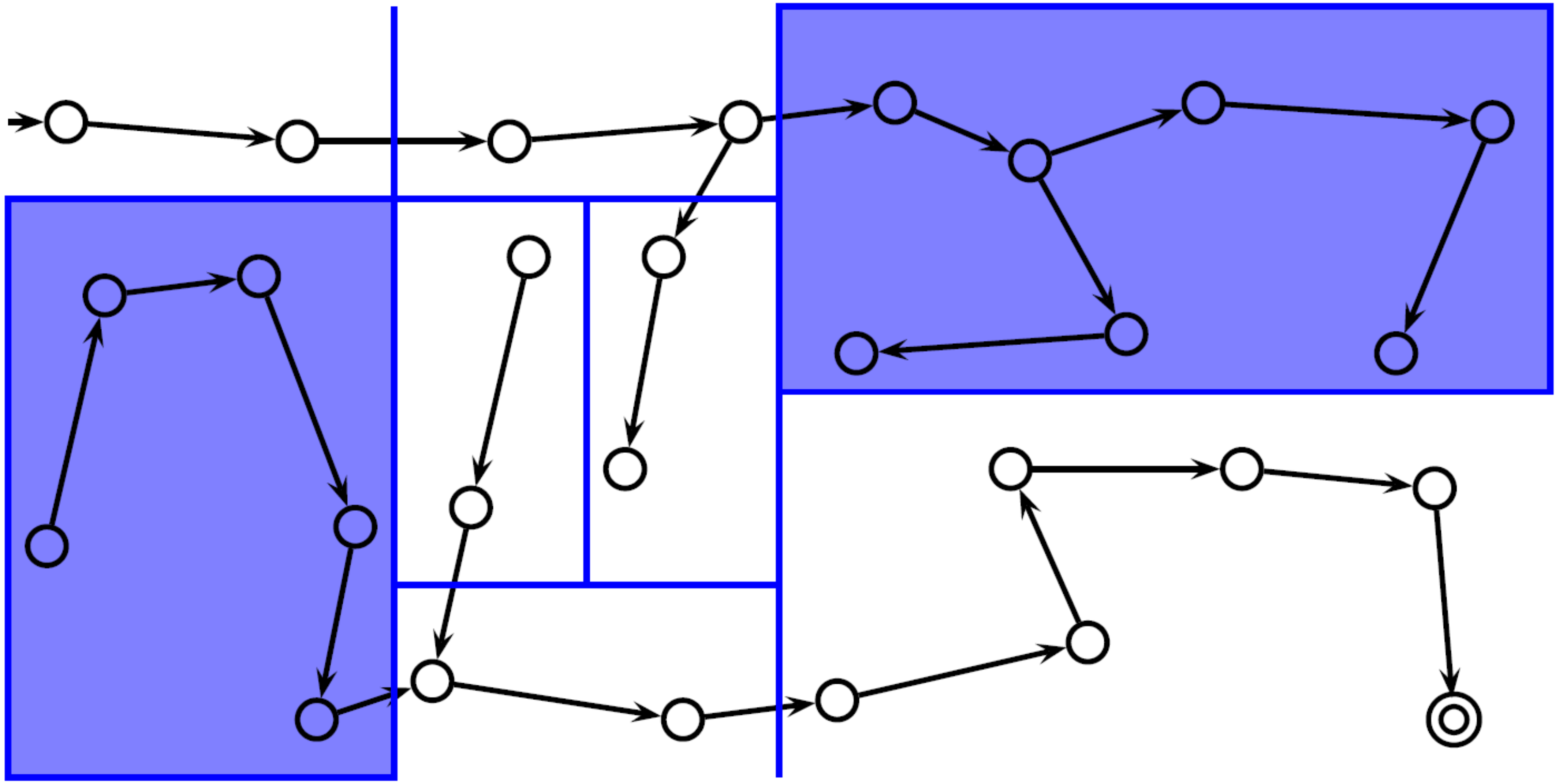
Use interpolant of $\phi_1$ and $\phi_2$ as new predicate.

# Problem: abstract state space explosion

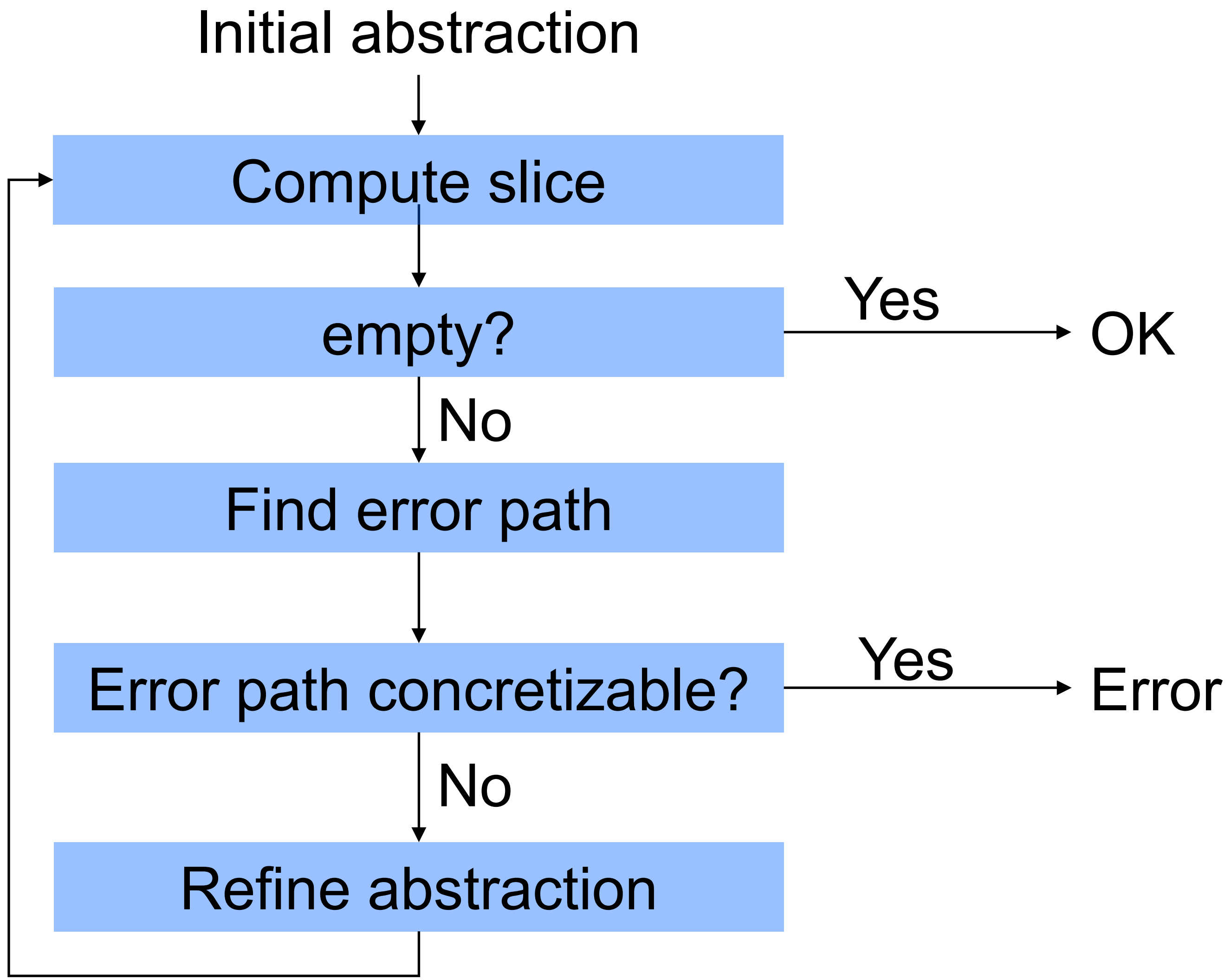
- Abstract state space grows exponentially with number of predicates



# Slicing Abstractions

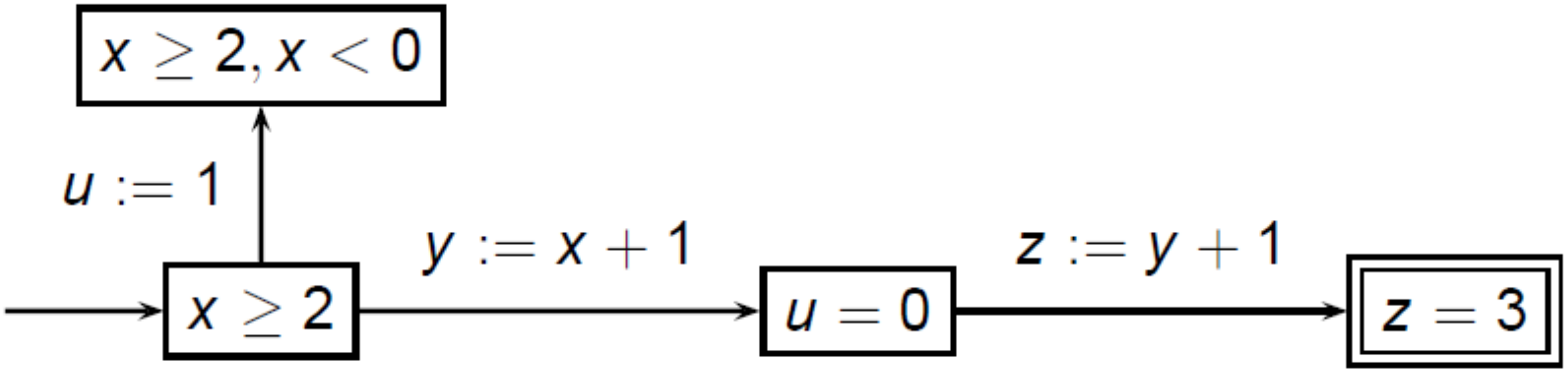


# Slicing Abstractions (SLAB)



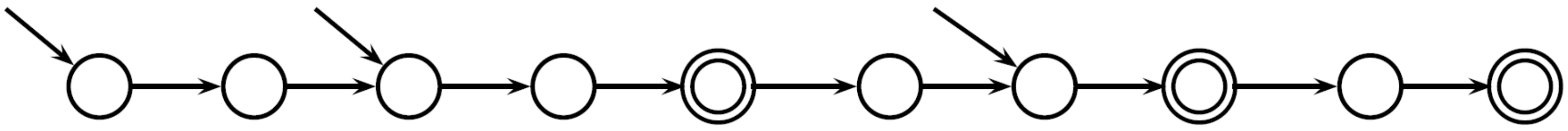
# SLAB abstractions

- Finite graphs
- Nodes labeled with sets of literals
- Edges labeled with sets of transitions
- Initial node, error node

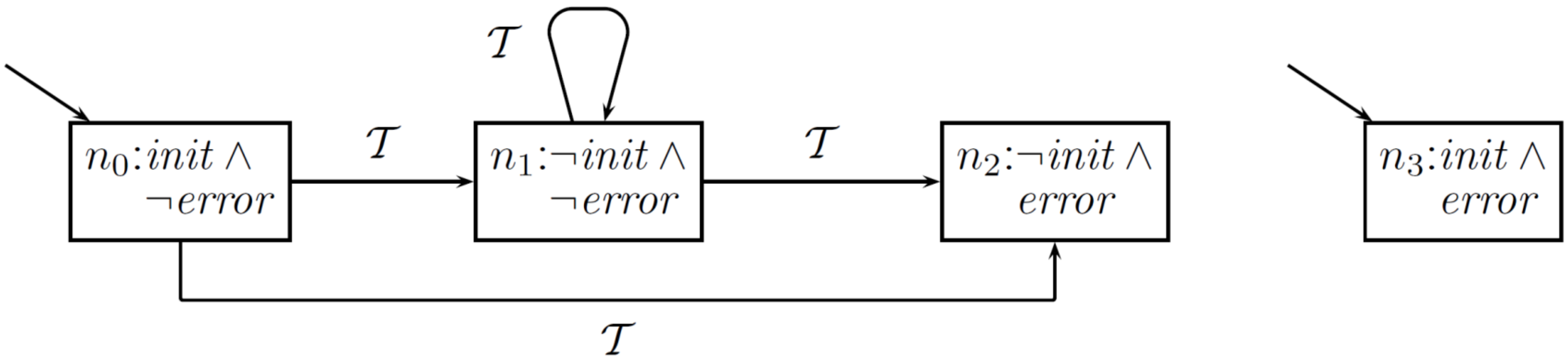


# Initial abstraction

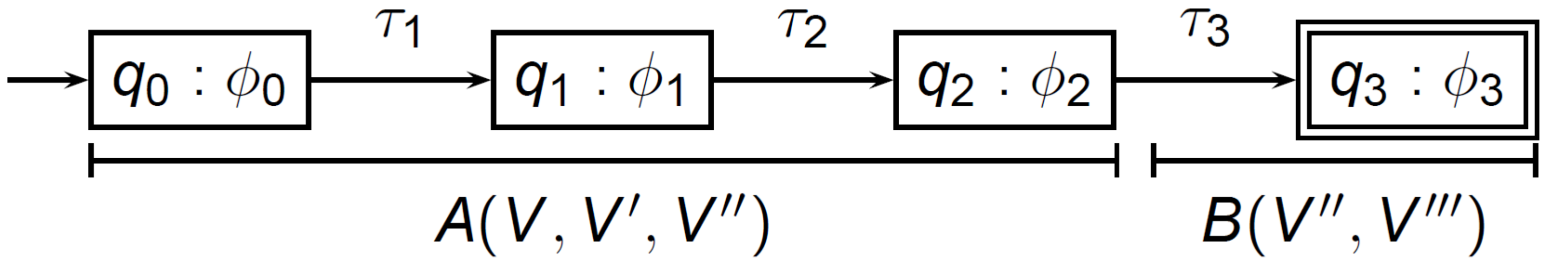
- need only *irreducible* error paths



- Initial abstraction:

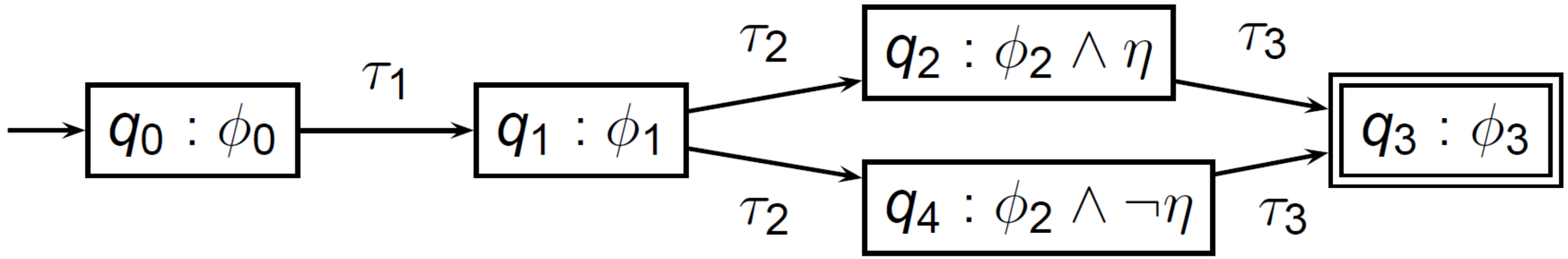


# Local refinement by node splitting



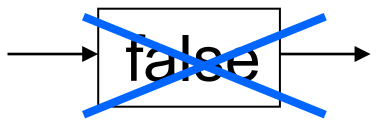

- $A \wedge B$ unsat, but $A, B$ sat $\rightsquigarrow$ Craig interpolant $\eta$:
  - $A \models \eta$, $B \models \neg\eta$
  - $Var(\eta) \subseteq Var(A) \cap Var(B)$, i.e. values at $q_2$

$\rightsquigarrow$ split $q_2$ with $\eta, \neg\eta$:

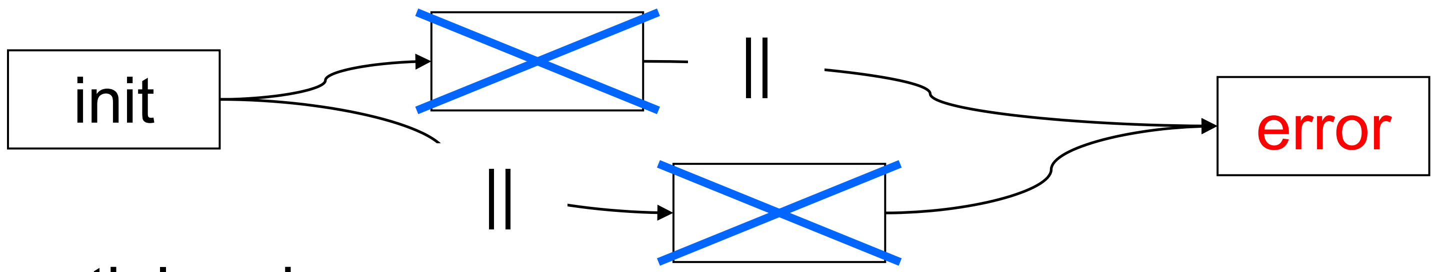


# Slicing: Eliminating Nodes

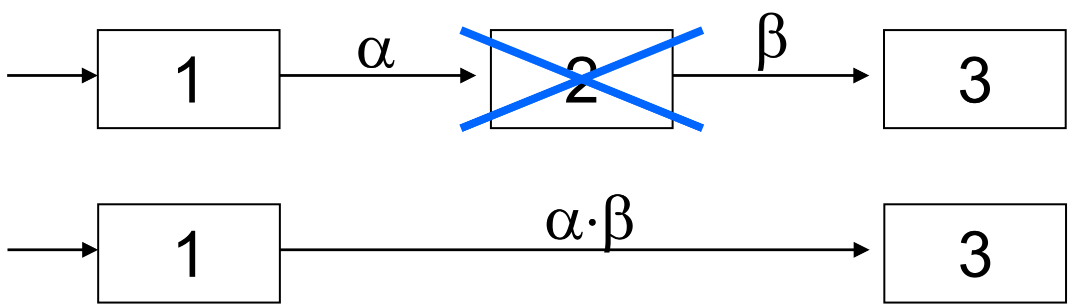
- Inconsistent nodes




- Unreachable nodes

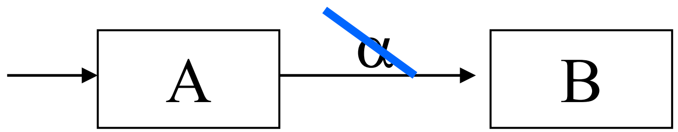



- Sequential nodes



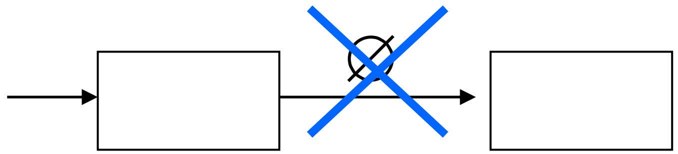
# Slicing: Eliminating transitions

- Inconsistent transitions

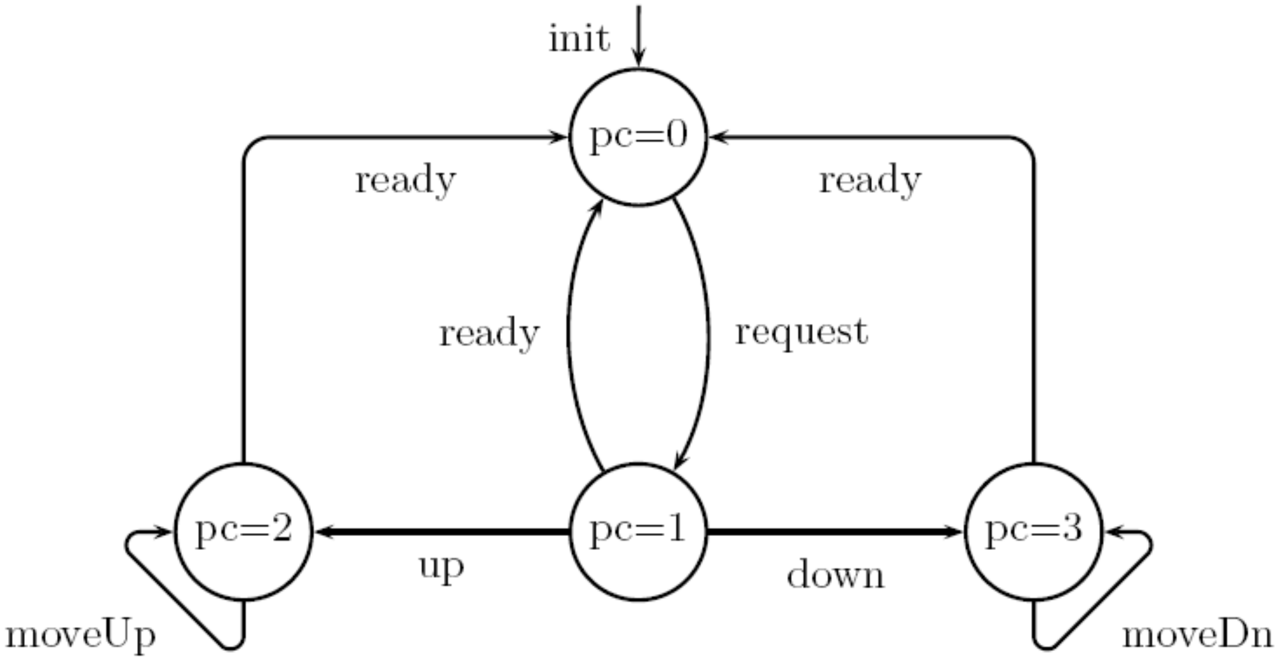



$A(V) \wedge \alpha(V,V') \wedge B(V')$ <u>unsatisfiable</u>

- Empty Edges

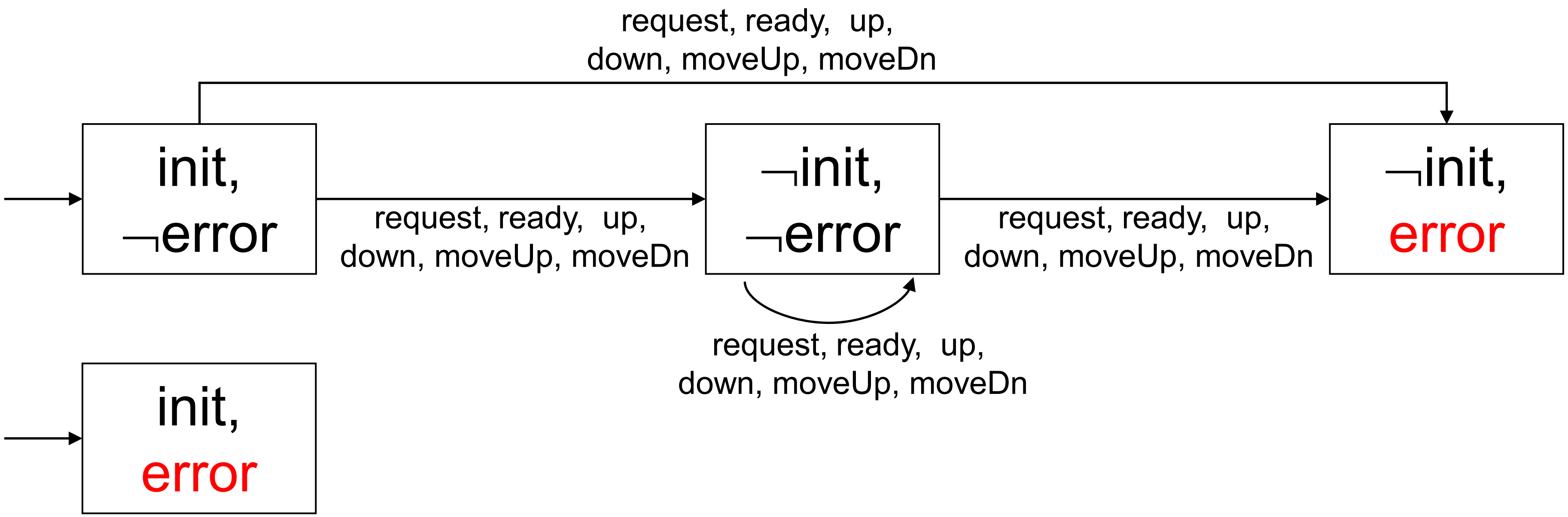


# Example

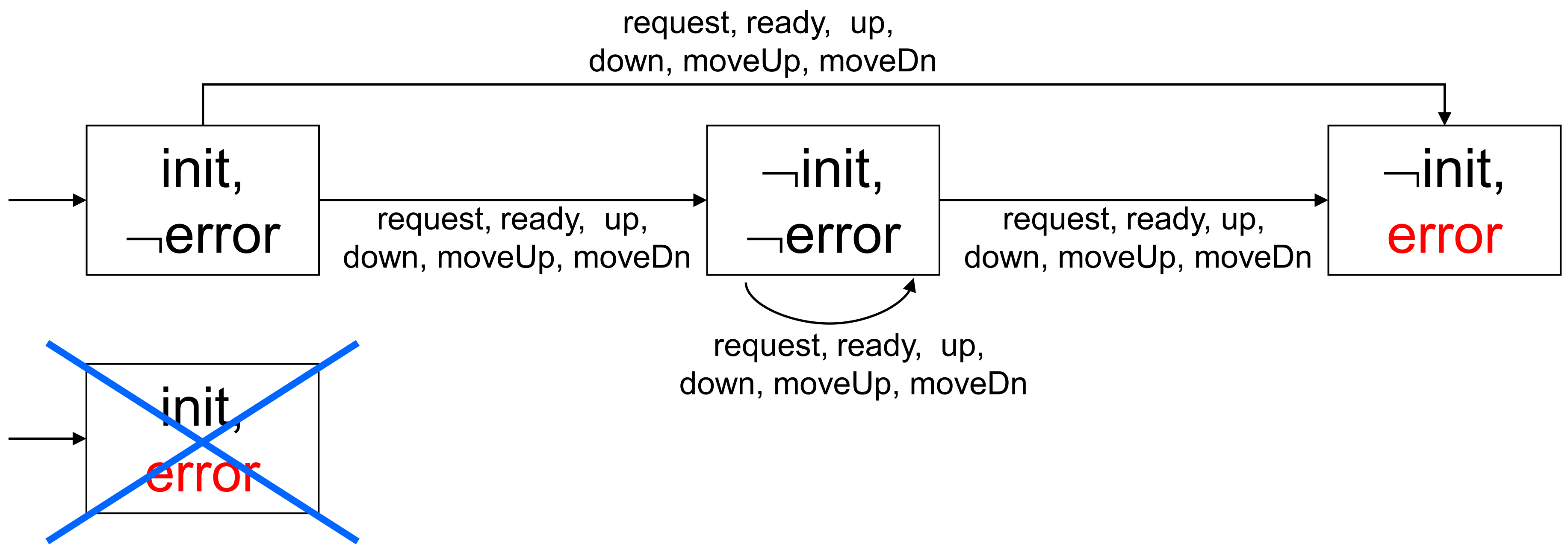



| | |
|---|---|
| *init* | $pc{=}0 \wedge current{\leq}Max \wedge input{\leq}Max$ |
| *error* | $current{>}Max$ |
| request | $pc{=}0 \wedge pc'{=}1 \wedge current'{=}current \wedge req'{=}input \wedge input{\leq}Max$ |
| ready | $pc \geq 1 \wedge req{=}current \wedge pc'{=}0 \wedge current'{=}current \wedge req'{=}req \wedge input'{\leq}Max$ |
| up | $pc{=}1 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current \wedge req'{=}req$ |
| down | $pc{=}1 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current \wedge req'{=}req$ |
| moveUp | $pc{=}2 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current + 1 \wedge req'{=}req$ |
| moveDn | $pc{=}3 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current - 1 \wedge req'{=}req$ |

# Initial Abstraction

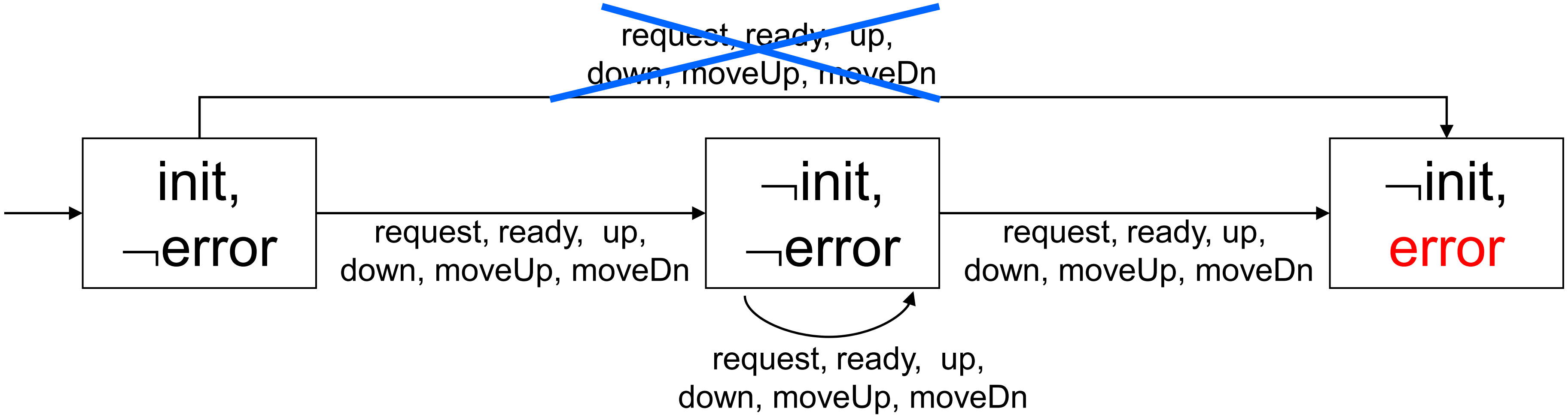


# Slicing



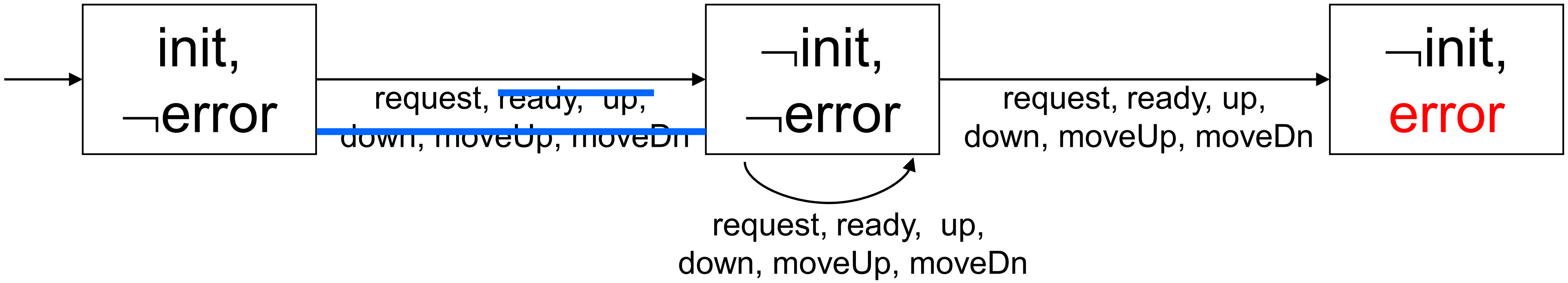

| | |
|---|---|
| $init$ | $pc{=}0 \wedge current{\leq}Max \wedge input{\leq}Max$ |
| $error$ | $current{>}Max$ |
| request | $pc{=}0 \wedge pc'{=}1 \wedge current'{=}current \wedge req'{=}input \wedge input{\leq}Max$ |
| ready | $pc \geq 1 \wedge req{=}current \wedge pc'{=}0 \wedge current'{=}current \wedge req'{=}req \wedge input'{\leq}Max$ |
| up | $pc{=}1 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current \wedge req'{=}req$ |
| down | $pc{=}1 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current \wedge req'{=}req$ |
| moveUp | $pc{=}2 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current + 1 \wedge req'{=}req$ |
| moveDn | $pc{=}3 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current - 1 \wedge req'{=}req$ |

# Slicing



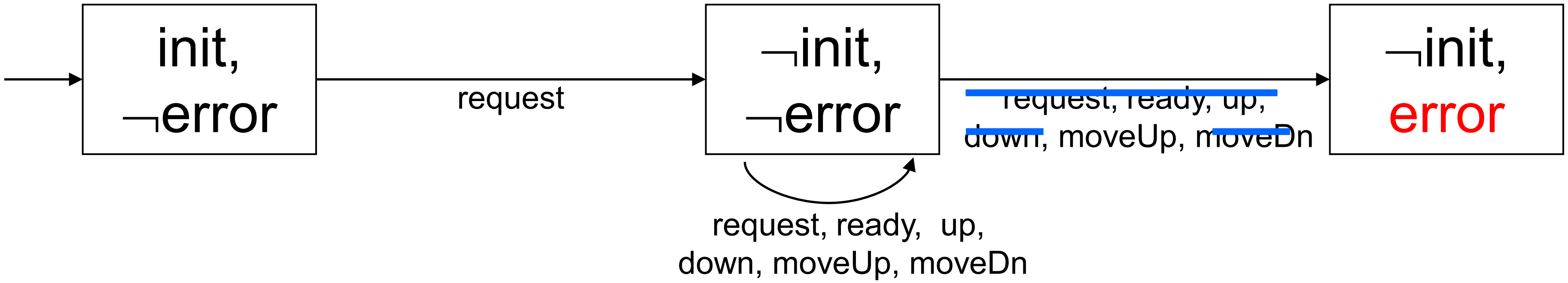

| | |
|---|---|
| *init* | $pc{=}0 \wedge current{\leq}Max \wedge input{\leq}Max$ |
| *error* | $current{>}Max$ |
| request | $pc{=}0 \wedge pc'{=}1 \wedge current'{=}current \wedge req'{=}input \wedge input{\leq}Max$ |
| ready | $pc \geq 1 \wedge req{=}current \wedge pc'{=}0 \wedge current'{=}current \wedge req'{=}req \wedge input'{\leq}Max$ |
| up | $pc{=}1 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current \wedge req'{=}req$ |
| down | $pc{=}1 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current \wedge req'{=}req$ |
| moveUp | $pc{=}2 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current + 1 \wedge req'{=}req$ |
| moveDn | $pc{=}3 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current - 1 \wedge req'{=}req$ |

# Slicing



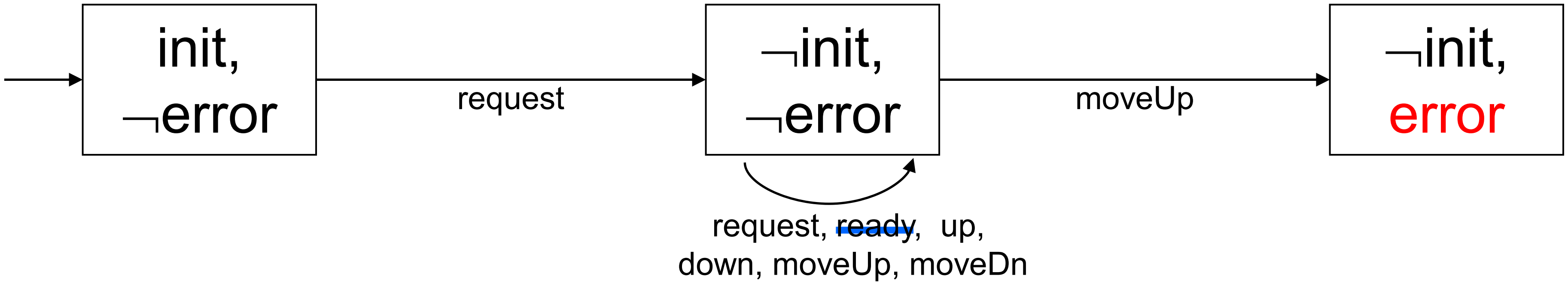

| | |
|---|---|
| *init* | $pc{=}0 \wedge current{\leq}Max \wedge input{\leq}Max$ |
| *error* | $current{>}Max$ |
| request | $pc{=}0 \wedge pc'{=}1 \wedge current'{=}current \wedge req'{=}input \wedge input{\leq}Max$ |
| ready | $pc \geq 1 \wedge req{=}current \wedge pc'{=}0 \wedge current'{=}current \wedge req'{=}req \wedge input'{\leq}Max$ |
| up | $pc{=}1 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current \wedge req'{=}req$ |
| down | $pc{=}1 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current \wedge req'{=}req$ |
| moveUp | $pc{=}2 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current + 1 \wedge req'{=}req$ |
| moveDn | $pc{=}3 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current - 1 \wedge req'{=}req$ |

# Slicing




| | |
|---|---|
| *init* | $pc{=}0 \wedge current{\leq}Max \wedge input{\leq}Max$ |
| *error* | $current{>}Max$ |
| request | $pc{=}0 \wedge pc'{=}1 \wedge current'{=}current \wedge req'{=}input \wedge input{\leq}Max$ |
| ready | $pc \geq 1 \wedge req{=}current \wedge pc'{=}0 \wedge current'{=}current \wedge req'{=}req \wedge input'{\leq}Max$ |
| up | $pc{=}1 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current \wedge req'{=}req$ |
| down | $pc{=}1 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current \wedge req'{=}req$ |
| moveUp | $pc{=}2 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current + 1 \wedge req'{=}req$ |
| moveDn | $pc{=}3 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current - 1 \wedge req'{=}req$ |

# Slicing

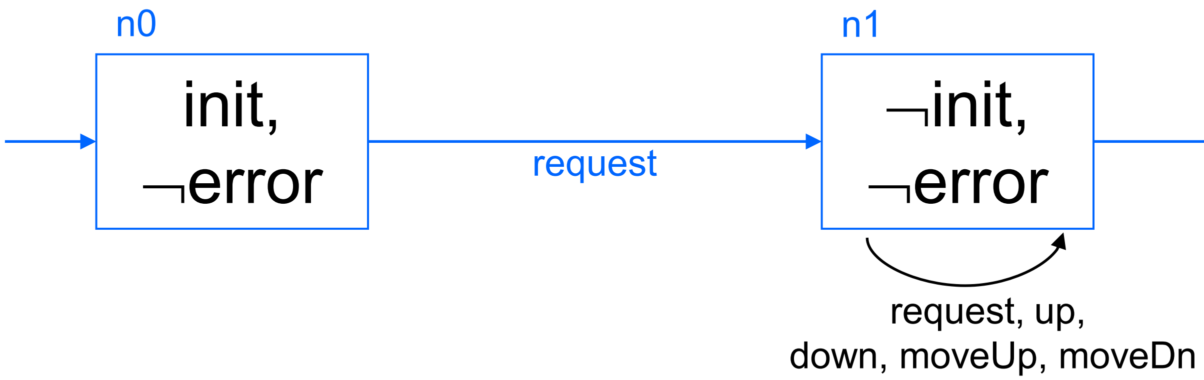



| | |
|---|---|
| *init* | $pc{=}0 \wedge current{\leq}Max \wedge input{\leq}Max$ |
| *error* | $current{>}Max$ |
| request | $pc{=}0 \wedge pc'{=}1 \wedge current'{=}current \wedge req'{=}input \wedge input{\leq}Max$ |
| ready | $pc \geq 1 \wedge req{=}current \wedge pc'{=}0 \wedge current'{=}current \wedge req'{=}req \wedge input'{\leq}Max$ |
| up | $pc{=}1 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current \wedge req'{=}req$ |
| down | $pc{=}1 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current \wedge req'{=}req$ |
| moveUp | $pc{=}2 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current + 1 \wedge req'{=}req$ |
| moveDn | $pc{=}3 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current - 1 \wedge req'{=}req$ |

# Error Path Analysis

1. Error Path concretizable?
2. If yes: System incorrect
3. If no: Node split
   - Find minimal error path
   - Determine node to split
   - Determine splitting predicate

# Error Path Analysis

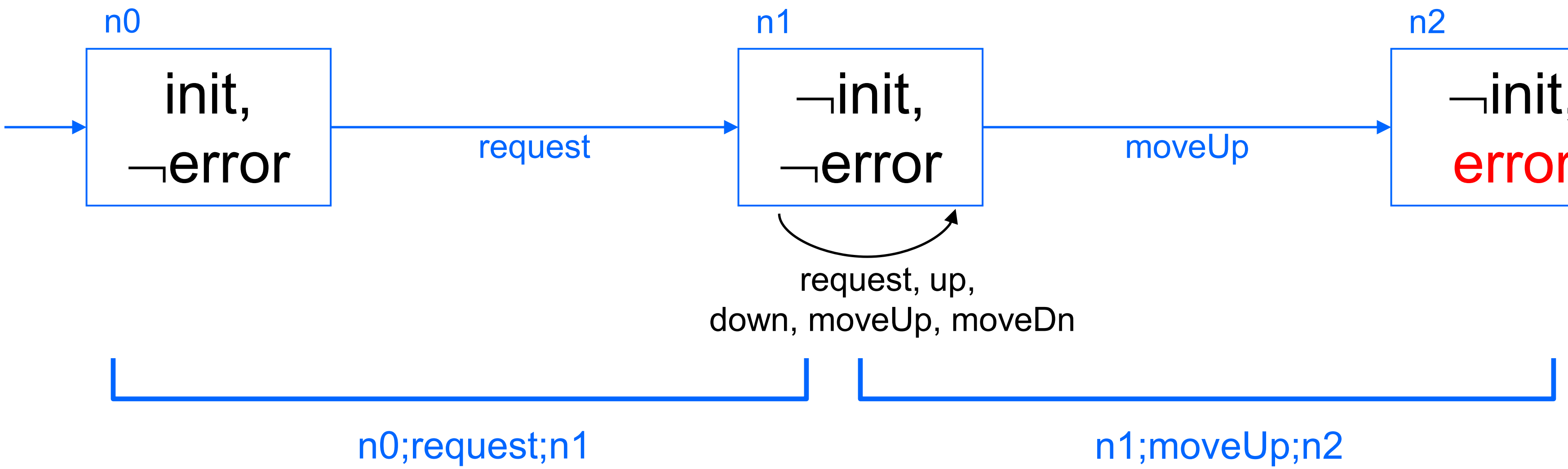



Error path concretizable?

**$\Phi$(n0;request;n1;moveUp;n2) =**
**n0($V^0$) $\wedge$ request($V^0$,$V^1$) $\wedge$ n1($V^1$) $\wedge$ moveUp($V^1$,$V^2$) $\wedge$ n2($V^2$)**

**is unsatisfiable $\Rightarrow$ n0;request;n1;moveUp;n2 is not concretizable.**

# Error Path Analysis



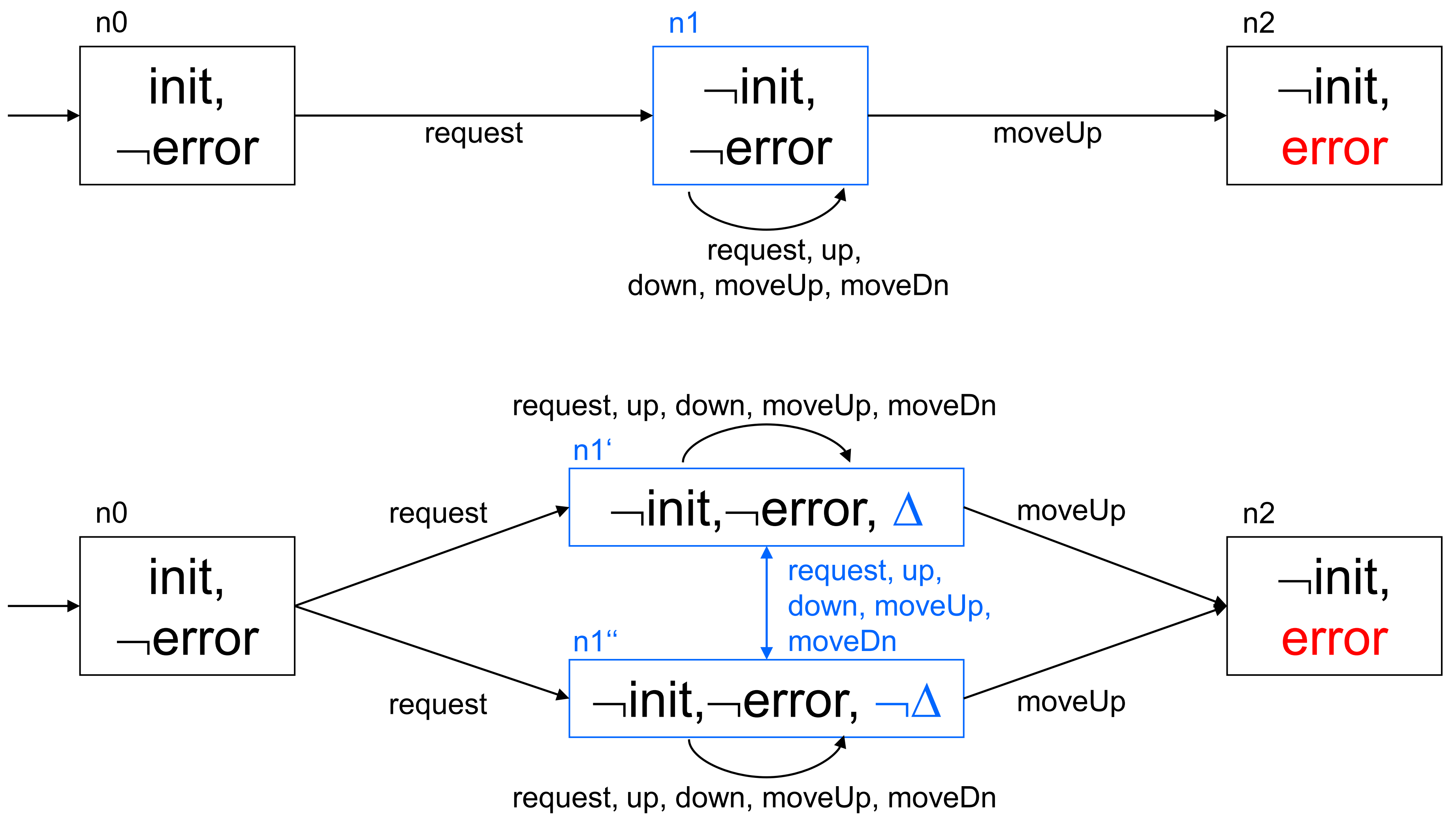

Error path minimal?

**Φ(n0;request;n1) is satisfiable. Φ(n1;moveUp;n2) is satisfiable.**

⇒ **n0;request;n1;moveUp;n2 is minimal.**

⇒ **Split node n1.**

# Node Split



n0
init,
¬error
request
n1
¬init,
¬error
moveUp
n2
¬init,
error
request, up,
down, moveUp, moveDn
request, up, down, moveUp, moveDn
n1‘
¬init,¬error, Δ
n0
init,
¬error
request
moveUp
n2
¬init,
error
request, up,
down, moveUp,
moveDn
n1“
¬init,¬error, ¬Δ
request
moveUp
request, up, down, moveUp, moveDn

# Interpolation

$\Phi$(n0;request;n1) = n0($V^0$) $\wedge$ request($V^0$,$V^1$) $\wedge$ n1($V^1$)  **satisfiable**

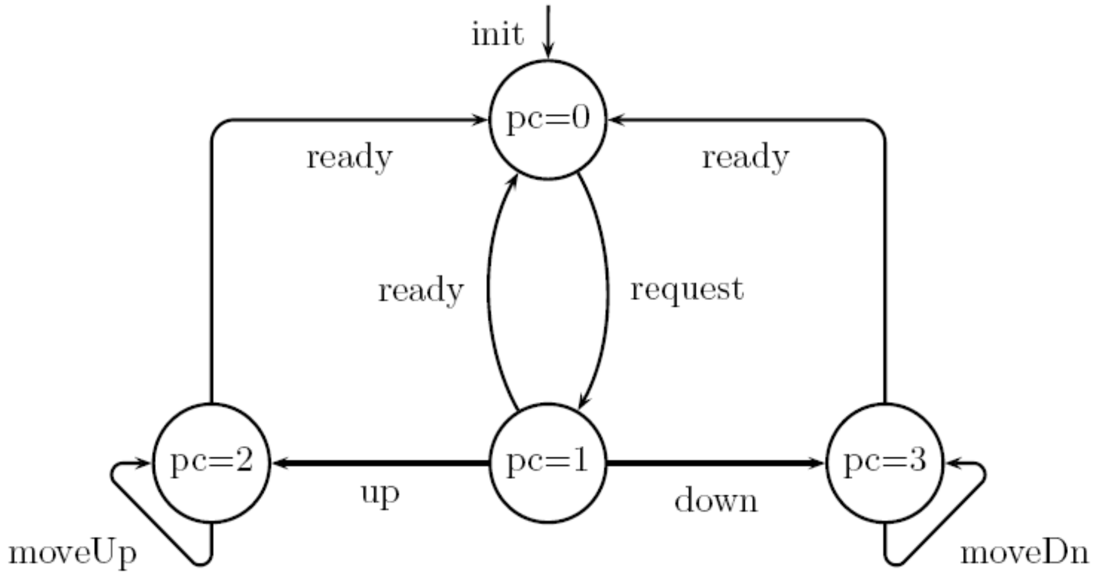
$\Phi$(moveUp;n2) = moveUp($V^1$,$V^2$) $\wedge$ n1($V^2$)  **satisfiable**

$\Phi$(n0;request;n1;moveUp;n2) = $\Phi$(n0;request;n1) $\wedge$ $\Phi$(moveUp;n2)
**unsatisfiable**

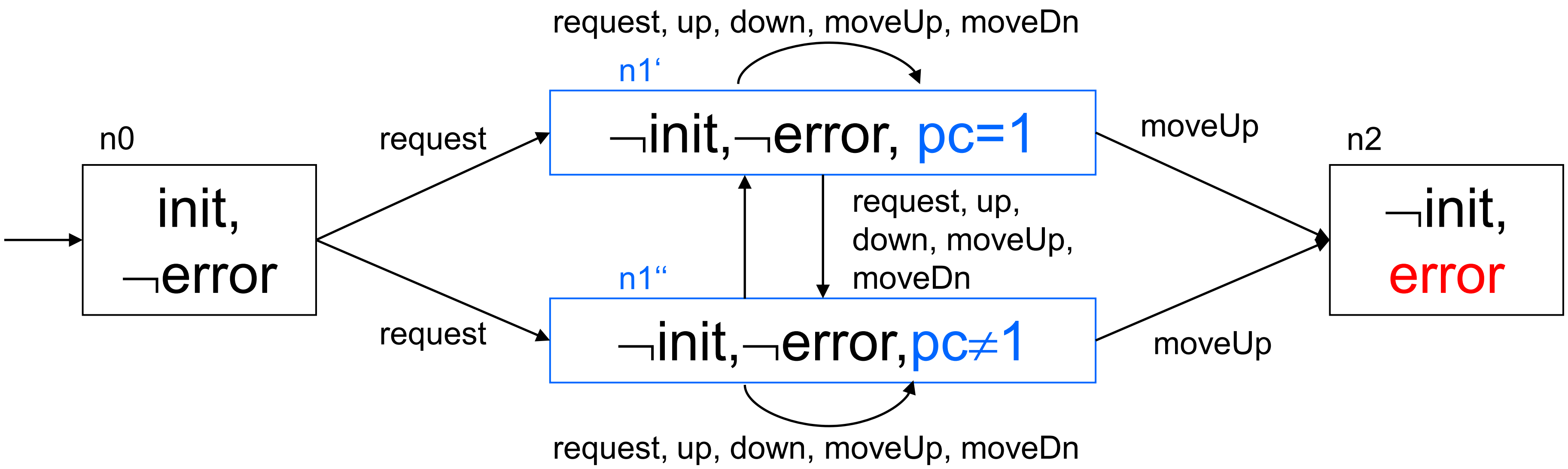
$\Rightarrow$ There exists a Craig interpolant $\Delta^1$, such that

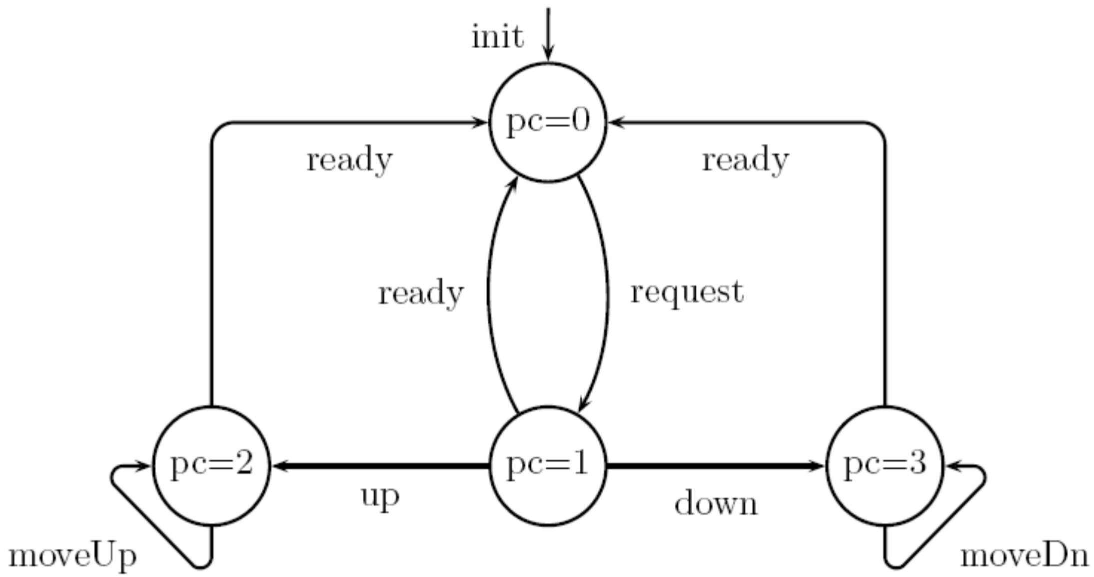
- $\Phi$(n0;request;n1) $\Rightarrow \Delta^1$
- $\Phi$(moveUp;n2) $\Rightarrow \neg\Delta^1$
- Variables($\Delta^1$) $\subseteq$ $V^1$

$\Delta^1$ = $pc^1$=1

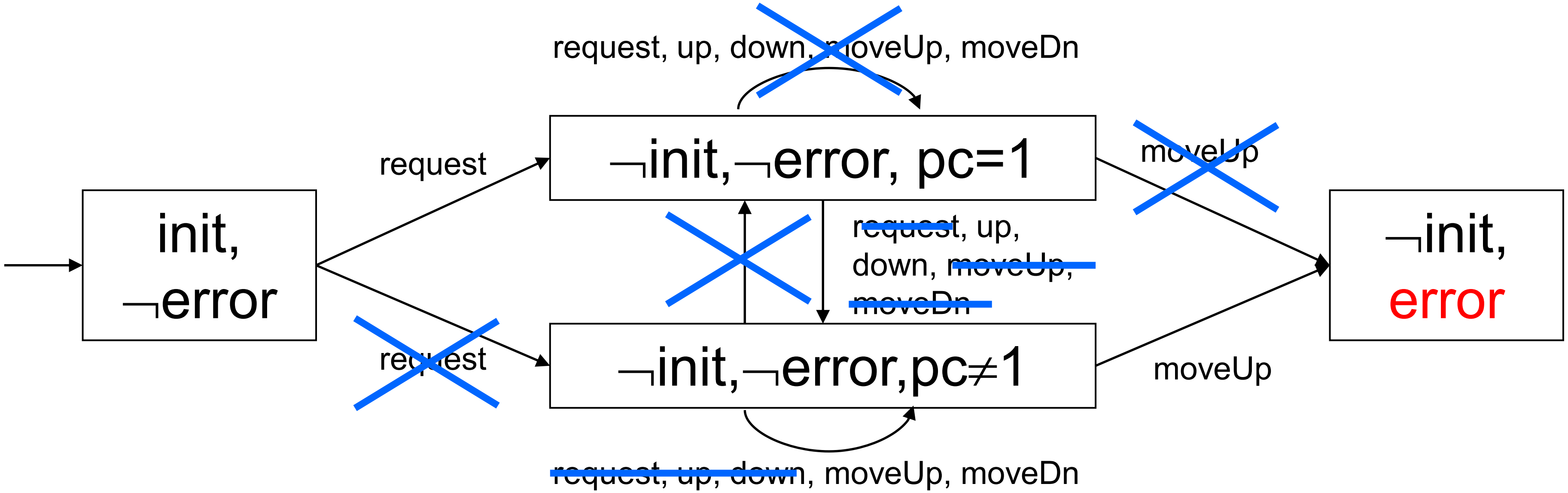


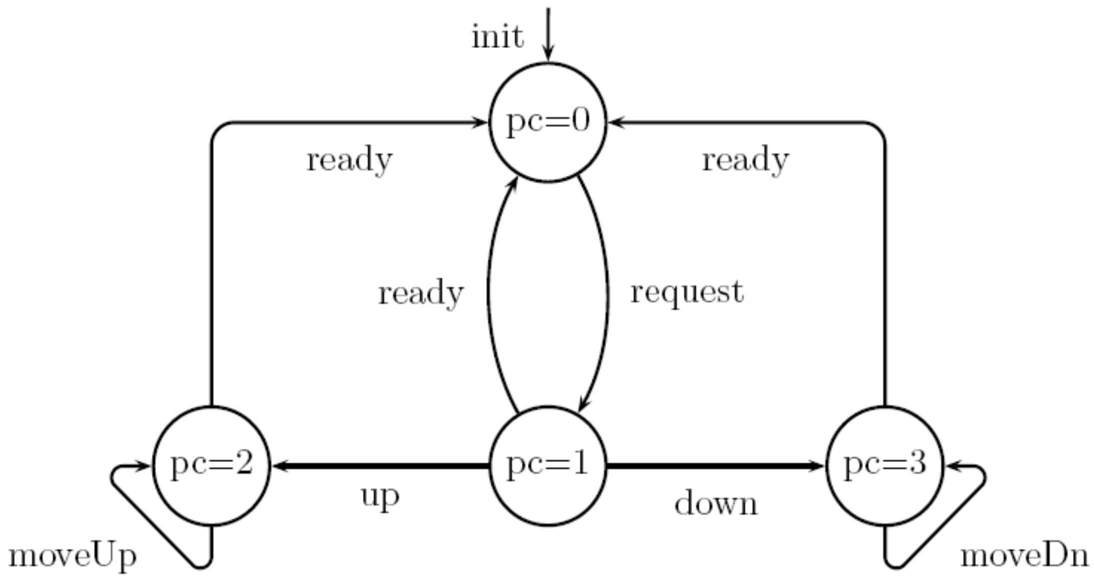
# Splitting


request, up, down, moveUp, moveDn
n1'
n0
request
¬init,¬error, pc=1
moveUp
n2
init,
¬error
request, up,
down, moveUp,
moveDn
¬init,
error
n1"
request
¬init,¬error,pc≠1
moveUp
request, up, down, moveUp, moveDn

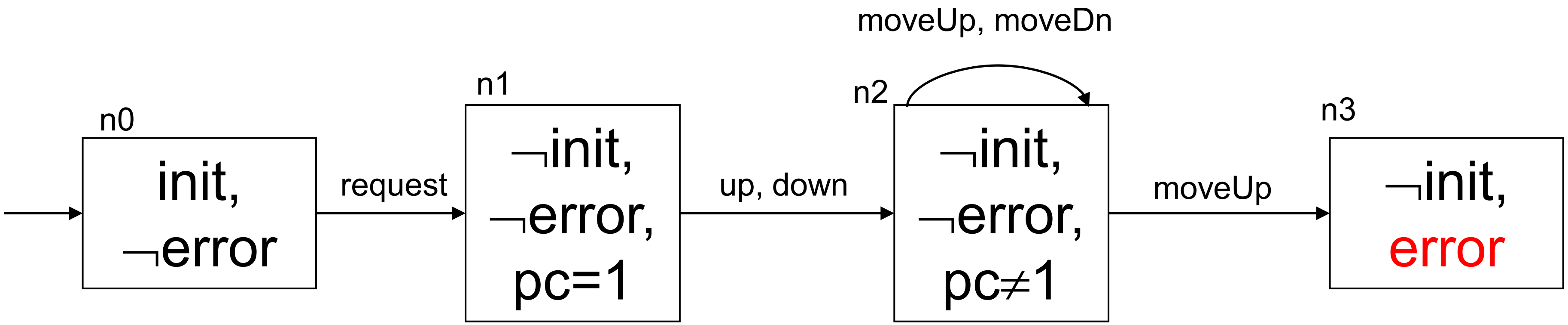

init
pc=0
ready
ready
ready
request
pc=2
pc=1
pc=3
up
down
moveUp
moveDn

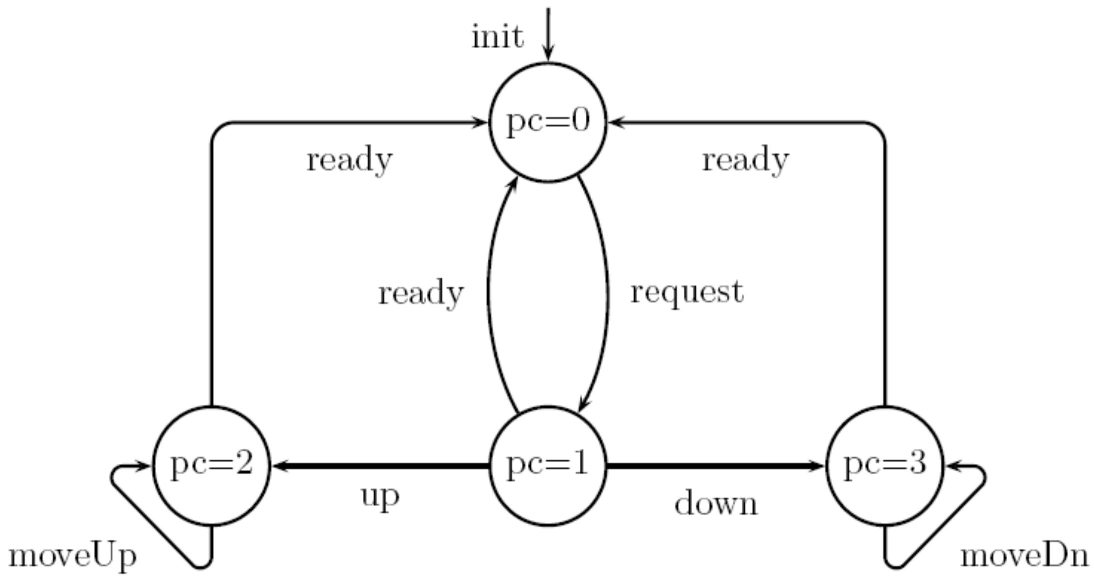
# Slicing


request, up, down, moveUp, moveDn
request
init,
¬error
¬init,¬error, pc=1
moveUp
request, up,
down, moveUp,
moveDn
¬init,
error
request
¬init,¬error,pc≠1
moveUp
request, up, down, moveUp, moveDn

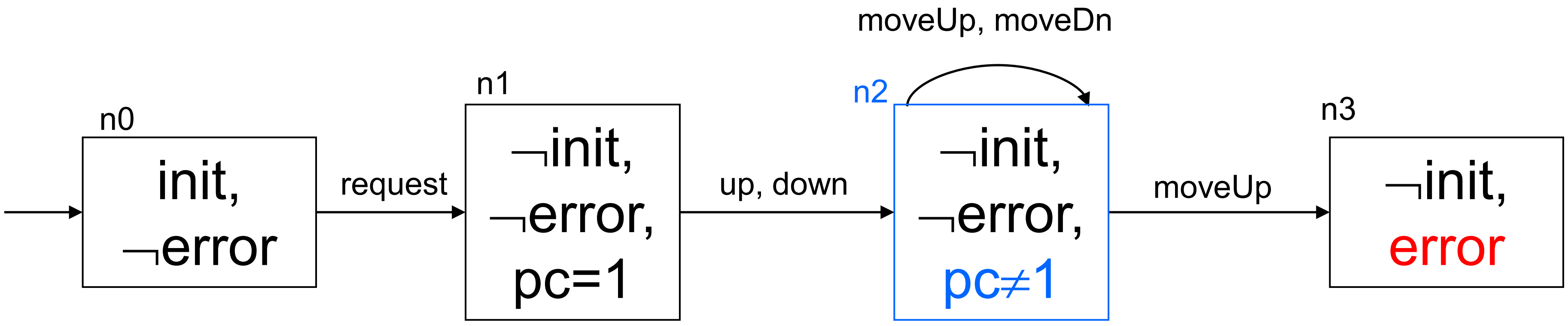

init
pc=0
ready
ready
ready
request
pc=2
pc=1
pc=3
up
down
moveUp
moveDn

# Error Path Analysis

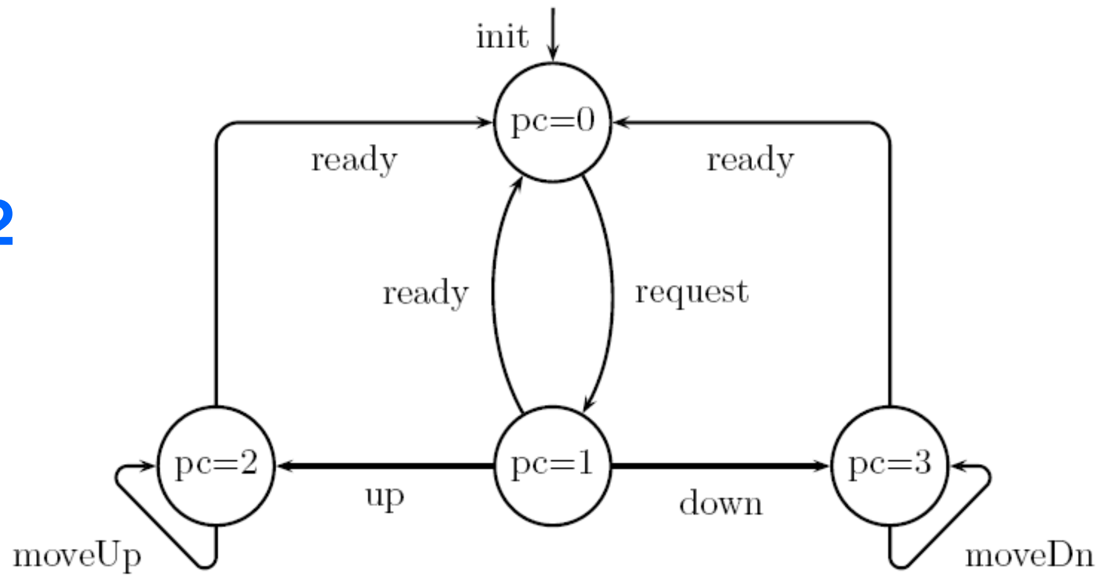


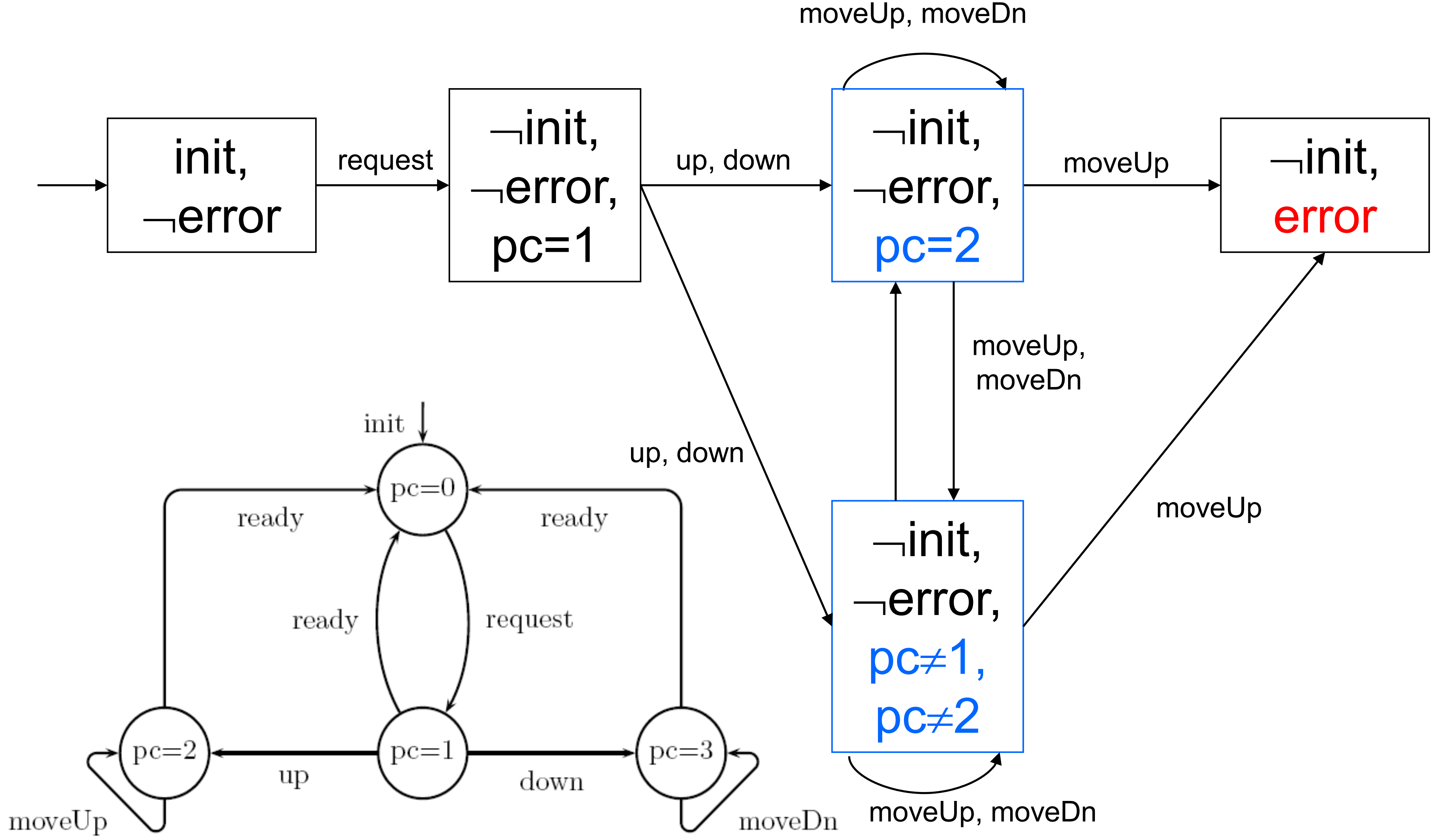
# Error Path Analysis



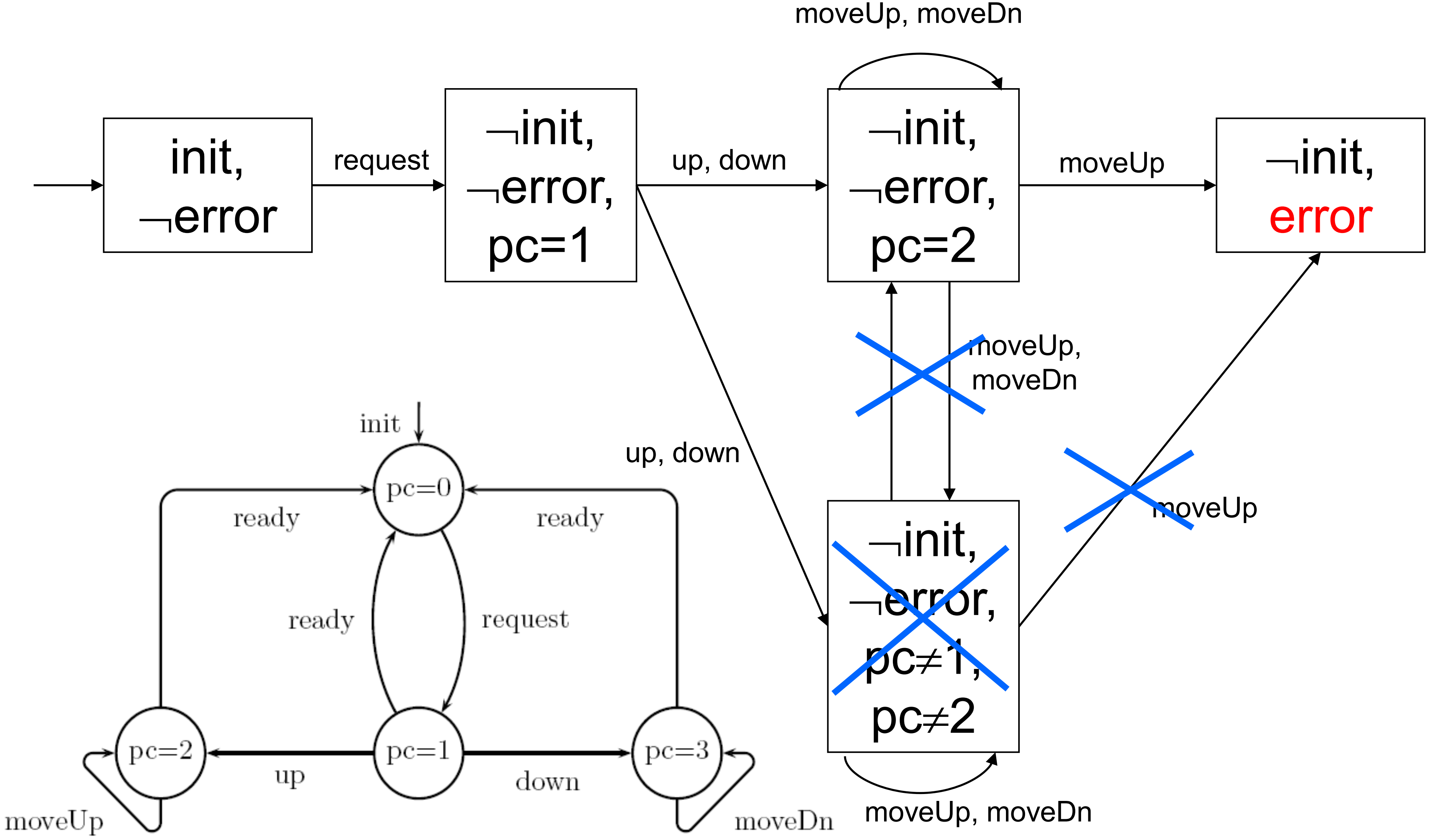

**Split node n2 with pc=2**



# Splitting

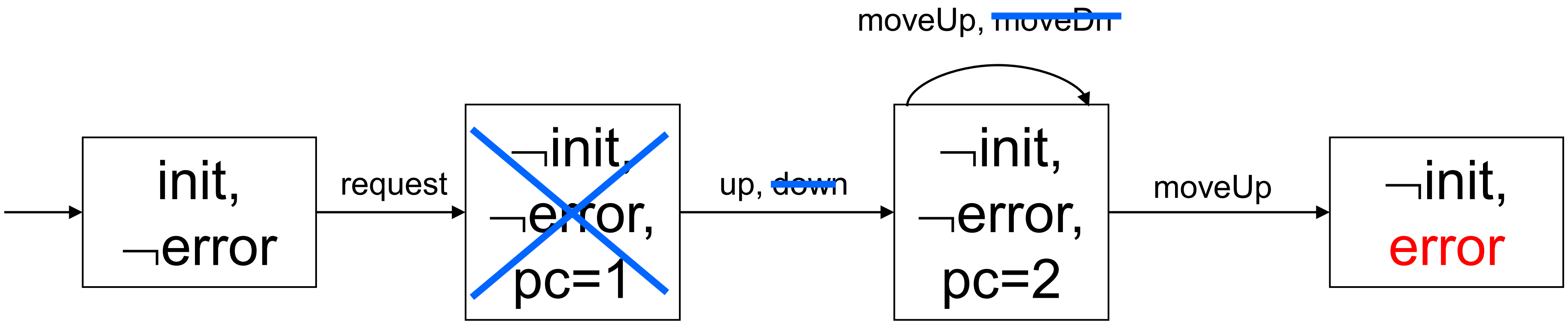

moveUp, moveDn
init,
¬error
request
¬init,
¬error,
pc=1
up, down
¬init,
¬error,
pc=2
moveUp
¬init,
error
moveUp,
moveDn
up, down
moveUp
¬init,
¬error,
pc≠1,
pc≠2
moveUp, moveDn
init
pc=0
ready
ready
ready
request
pc=2
pc=1
pc=3
up
down
moveUp
moveDn

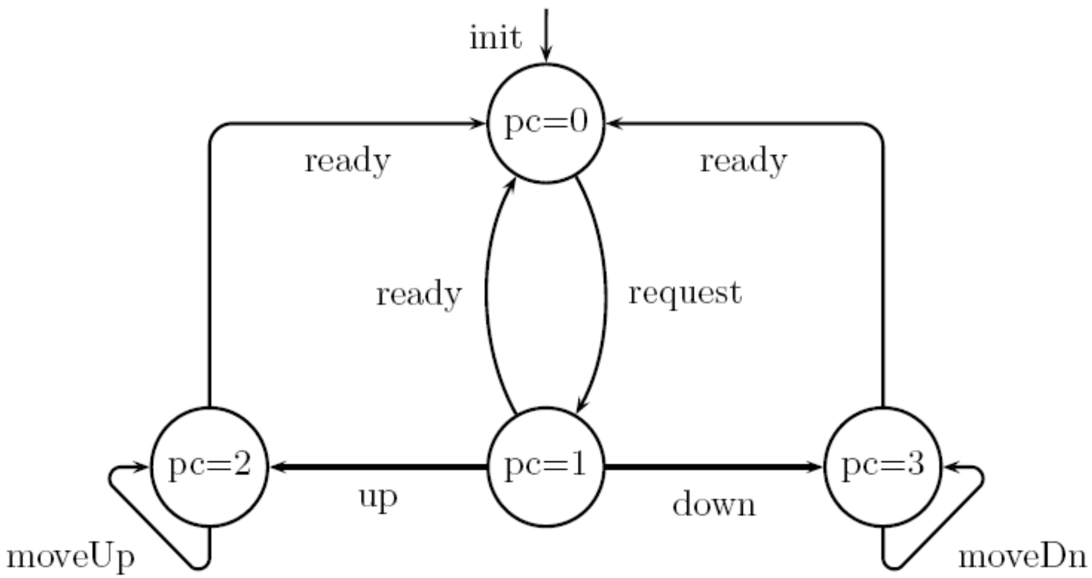
# Slicing



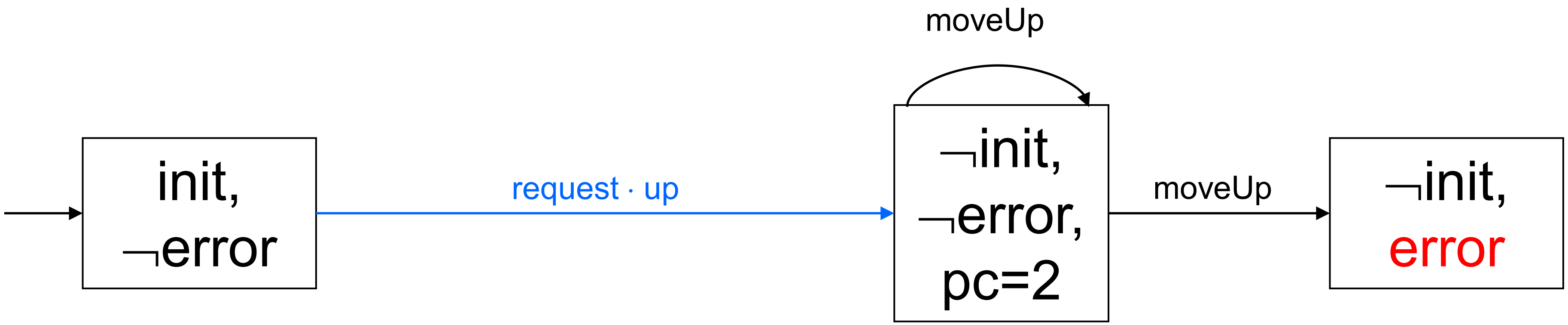
init, ¬error
request
¬init, ¬error, pc=1
up, down
moveUp, moveDn
¬init, ¬error, pc=2
moveUp
¬init, error
moveUp, moveDn
up, down
moveUp
¬init, ¬error, pc≠1, pc≠2
moveUp, moveDn
init
pc=0
ready
ready
ready
request
pc=2
pc=1
pc=3
up
down
moveUp
moveDn

# Slicing

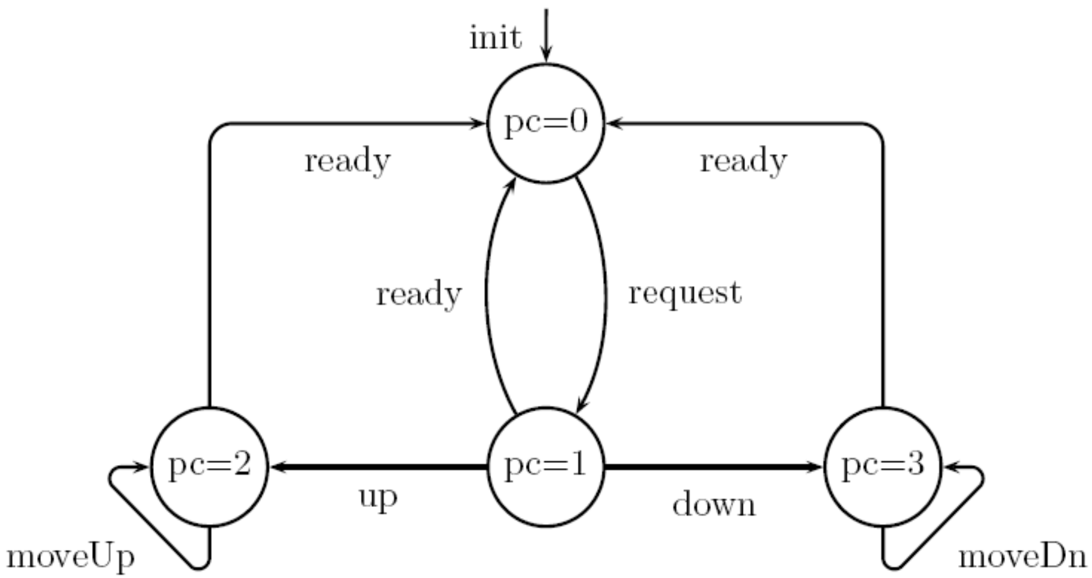

moveUp, ~~moveDn~~
init,
¬error
request
¬init,
¬error,
pc=1
up, ~~down~~
¬init,
¬error,
pc=2
moveUp
¬init,
error


init
pc=0
ready
ready
ready
request
pc=2
pc=1
pc=3
up
down
moveUp
moveDn

# Slicing

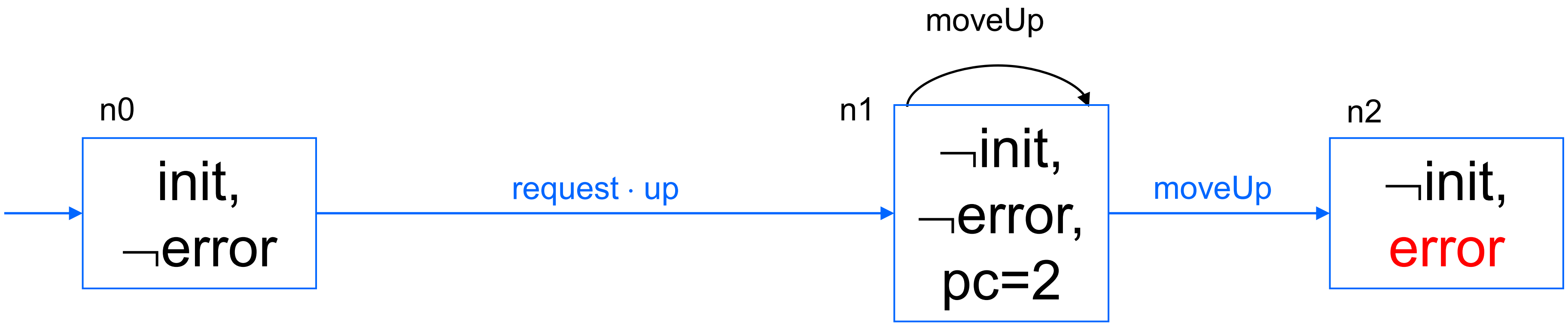


moveUp
init,
¬error
request · up
¬init,
¬error,
pc=2
moveUp
¬init,
error

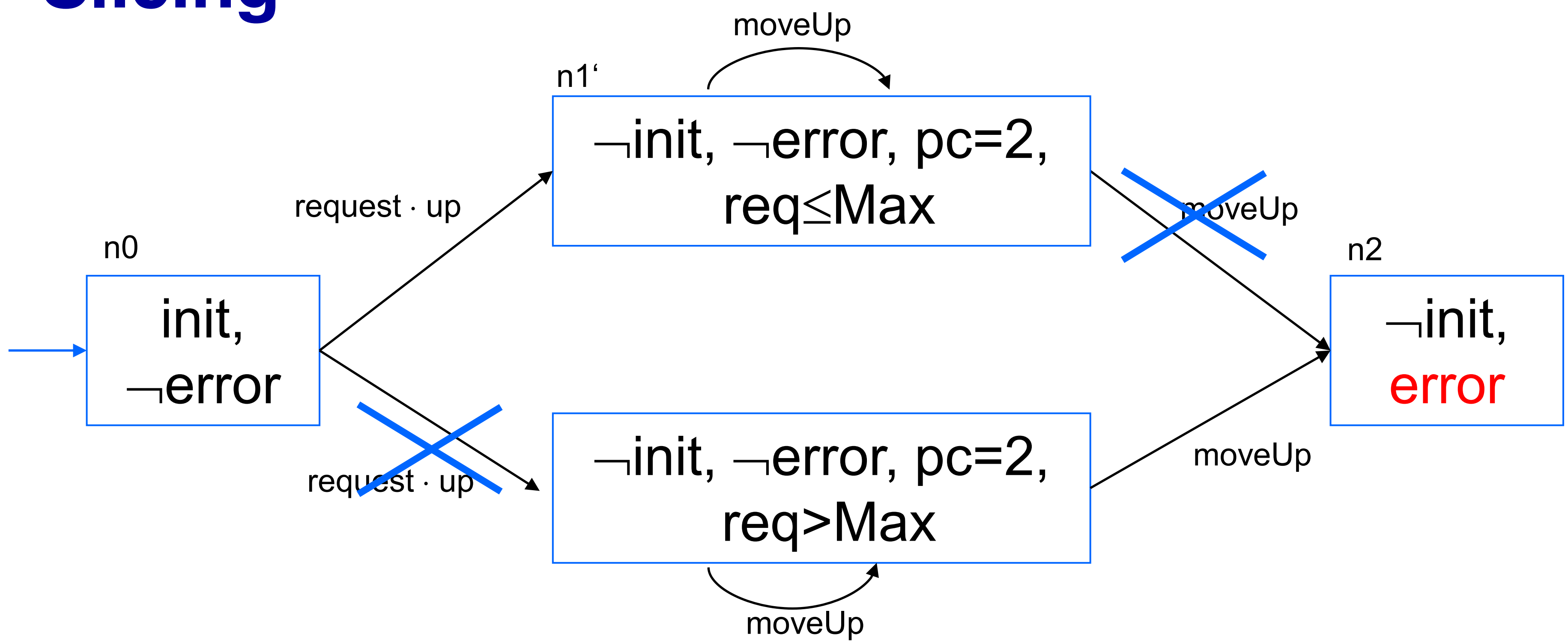
init
pc=0
ready
ready
ready
request
pc=2
pc=1
pc=3
up
down
moveUp
moveDn

# Error Path Analysis



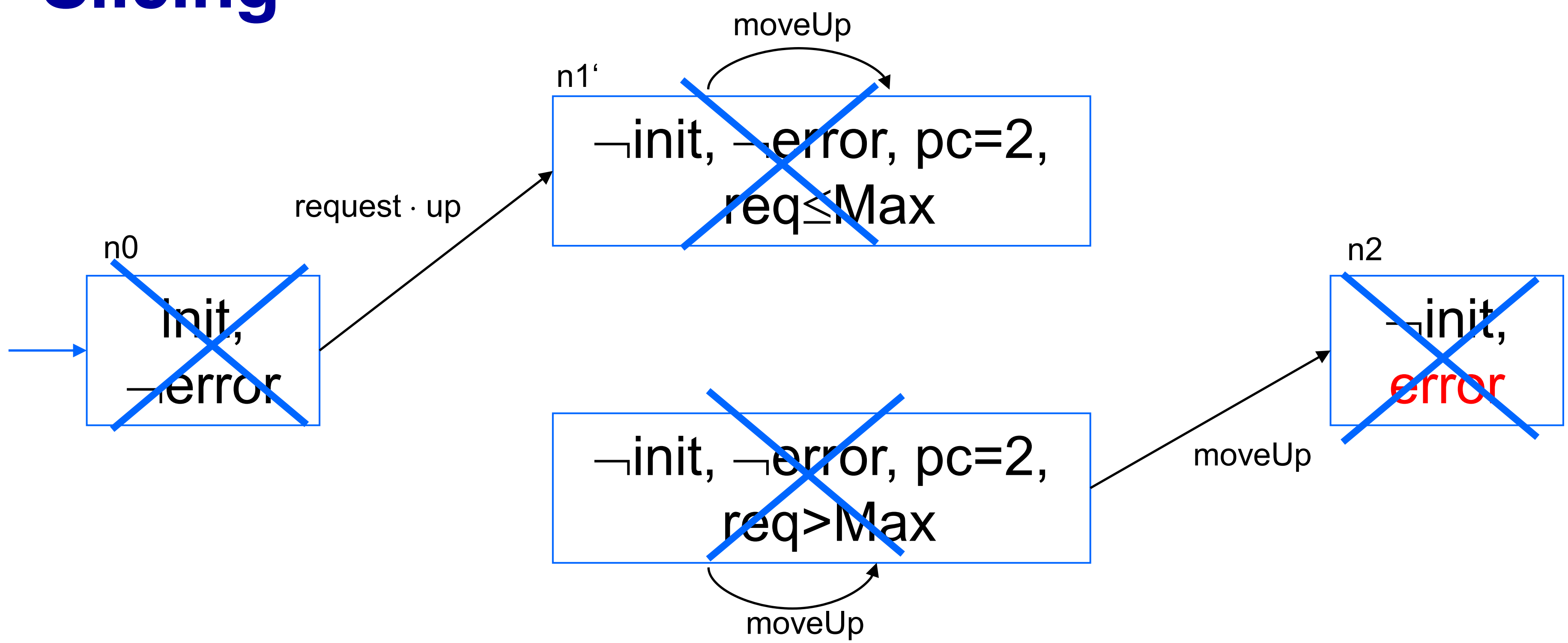

**Split node n1 with req≤Max**

# Slicing



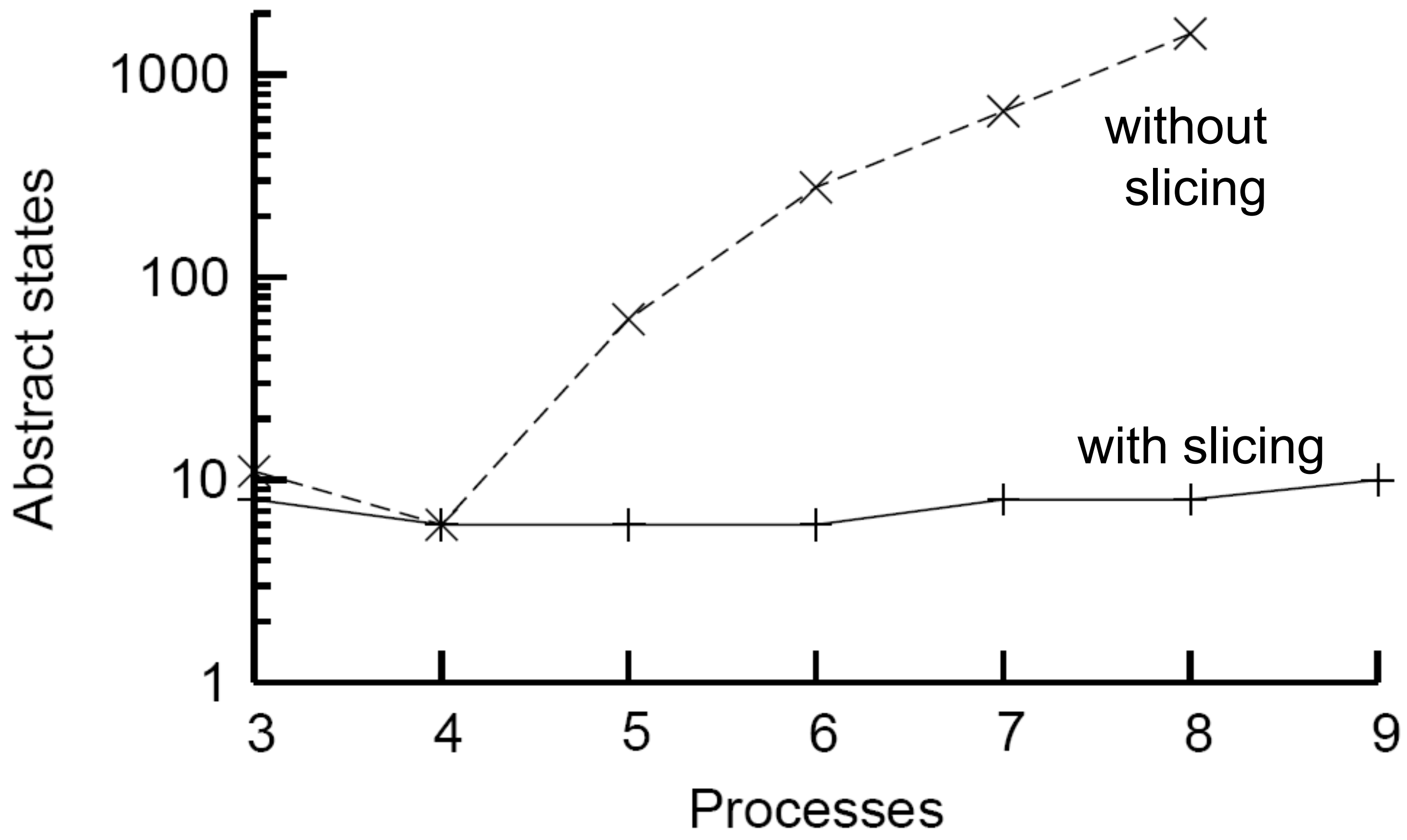

| | |
|---|---|
| $init$ | $pc{=}0 \wedge current{\leq}Max \wedge input{\leq}Max$ |
| $error$ | $current{>}Max$ |
| request | $pc{=}0 \wedge pc'{=}1 \wedge current'{=}current \wedge req'{=}input$ |
| ready | $pc \geq 1 \wedge req{=}current \wedge pc'{=}0 \wedge current'{=}current \wedge req'{=}req \wedge input'{\leq}Max$ |
| up | $pc{=}1 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current \wedge req'{=}req$ |
| down | $pc{=}1 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current \wedge req'{=}req$ |
| moveUp | $pc{=}2 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current + 1 \wedge req'{=}req$ |
| moveDn | $pc{=}3 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current - 1 \wedge req'{=}req$ |

# Slicing

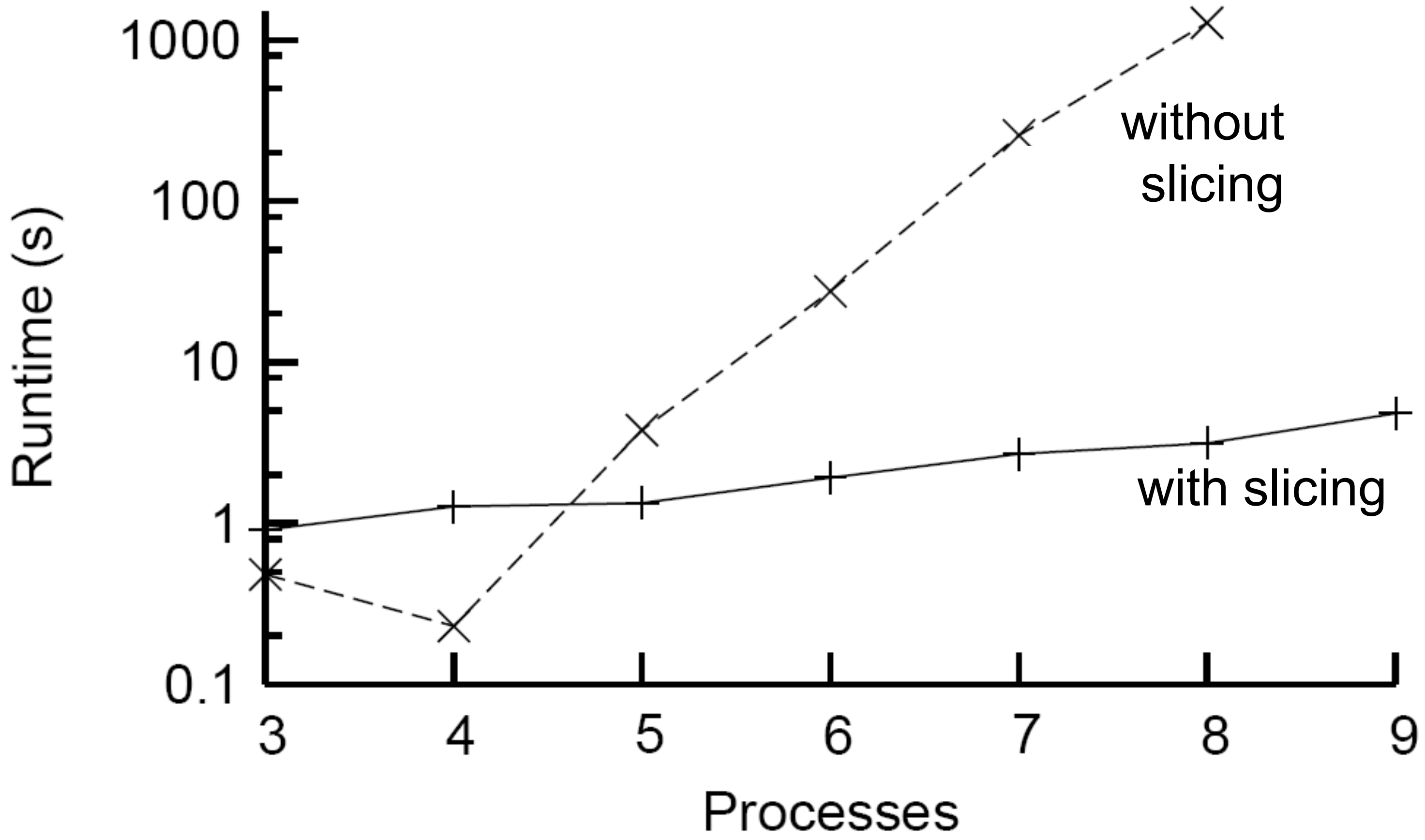



| | |
|---|---|
| *init* | $pc{=}0 \wedge current{\leq}Max \wedge input{\leq}Max$ |
| *error* | $current{>}Max$ |
| request | $pc{=}0 \wedge pc'{=}1 \wedge current'{=}current \wedge req'{=}input$ |
| ready | $pc \geq 1 \wedge req{=}current \wedge pc'{=}0 \wedge current'{=}current \wedge req'{=}req \wedge input'{\leq}Max$ |
| up | $pc{=}1 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current \wedge req'{=}req$ |
| down | $pc{=}1 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current \wedge req'{=}req$ |
| moveUp | $pc{=}2 \wedge req > current \wedge pc'{=}2 \wedge current'{=}current + 1 \wedge req'{=}req$ |
| moveDn | $pc{=}3 \wedge req < current \wedge pc'{=}3 \wedge current'{=}current - 1 \wedge req'{=}req$ |

# Experiments: State Space

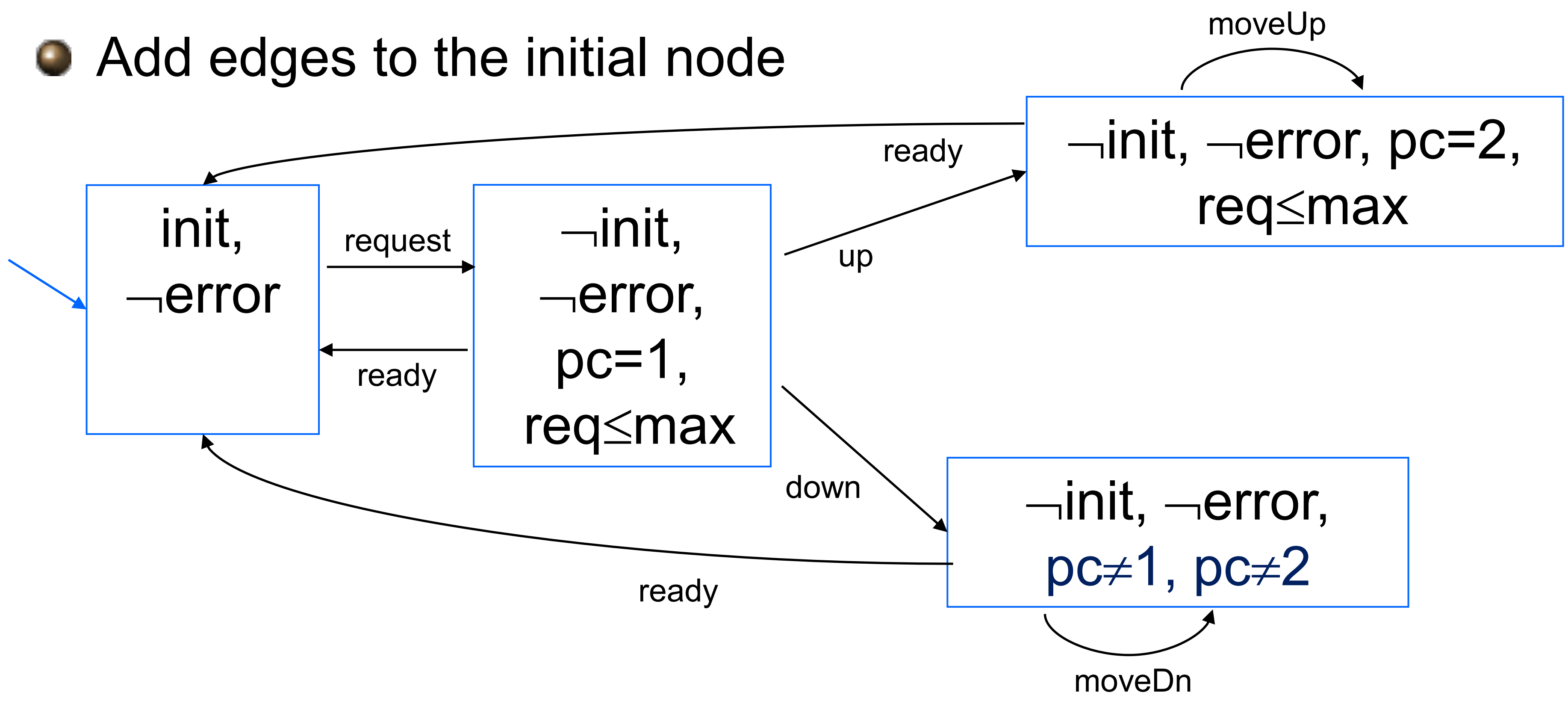


# Experiments: Runtime



# Verification diagrams as certificates

- Add intermediate nodes for composite transitions (using strongest postcondition)
- Do not remove nodes that are not backward reachable but still forward-reachable
- Add edges to the initial node



# Exams

- **Exam**: 09.10.2013, 9am at E2 2, Günter-Hotz-Hörsaal.
- **Backup Exam**: 25.11.2013, 10am at E1 3, HS 002.
- Each $2\frac{1}{2}$ hours.

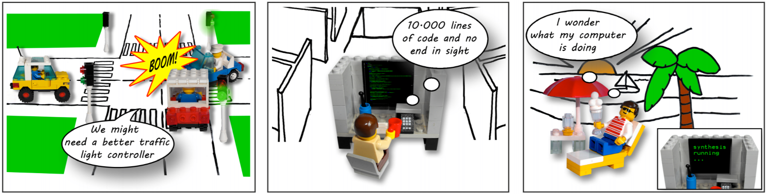
- Review session: 07.10.2013, 2pm at E1 3, HS 001.

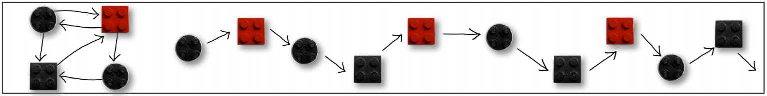
• Your grade solely depends on your performance in the exam.

• We inform you over the weekend, whether you are admitted to the exam (reach at least 50% of the total points in the assignments).

• The exams are open-book: bring books, hand-written, notes, etc., but no cell phones, laptops, tabs, etc.

# Advertisement



- Advanced lecture: **Infinite Games**
- Lectures: Th. 10.15am; Tutorials: Tu. 10am or 4pm
- Learn how infinite games...
  - allow you to automatically generate correct programs,
  - decide logics stronger than everything considered here, and
  - play, play, play...
- `www.react.uni-saarland.de/teaching/infinite-games-13-14/`

# The last slide

Thank you and good luck for the exam.