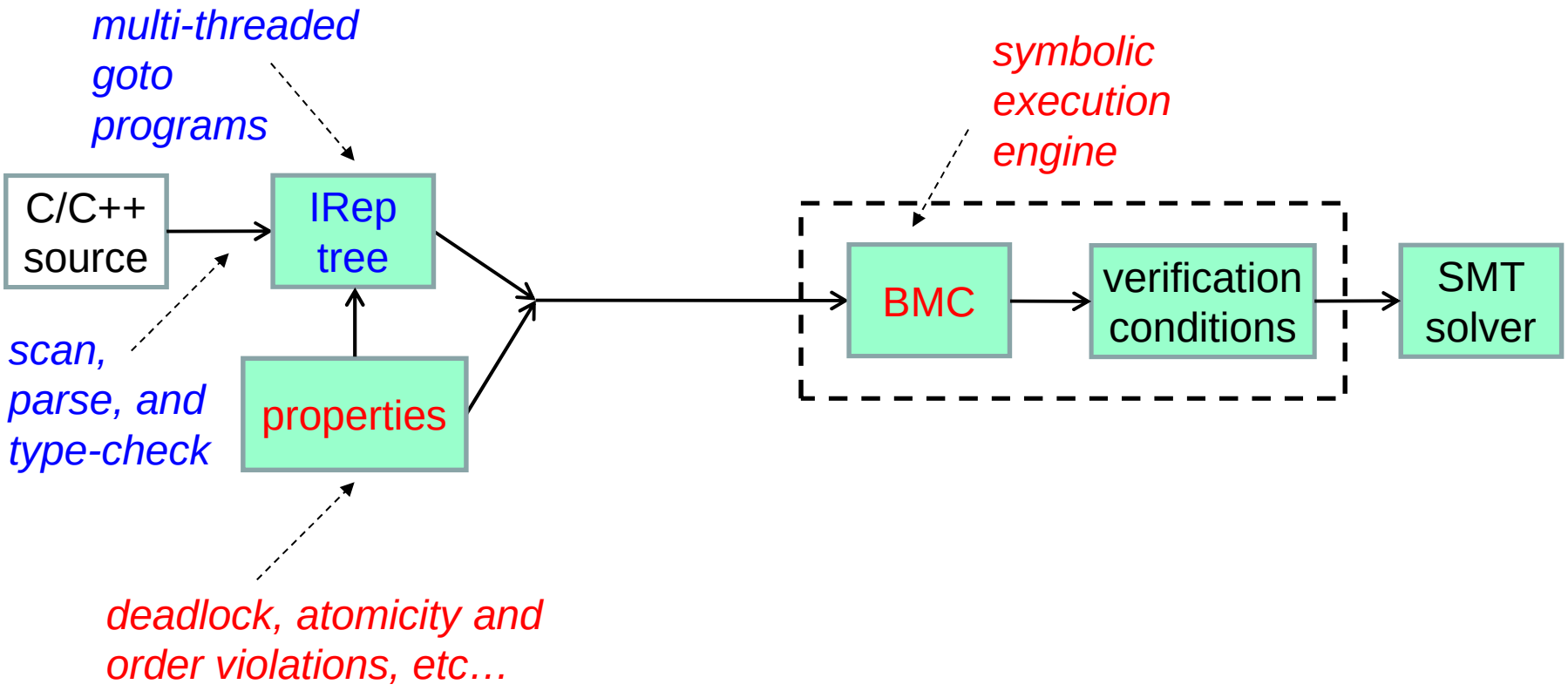
SMT-based Bounded Model Checking for Multi-threaded Software in Embedded Systems

Lucas Cordeiro

lucascordeiro@ufam.edu.br

Lazy exploration of interleavings

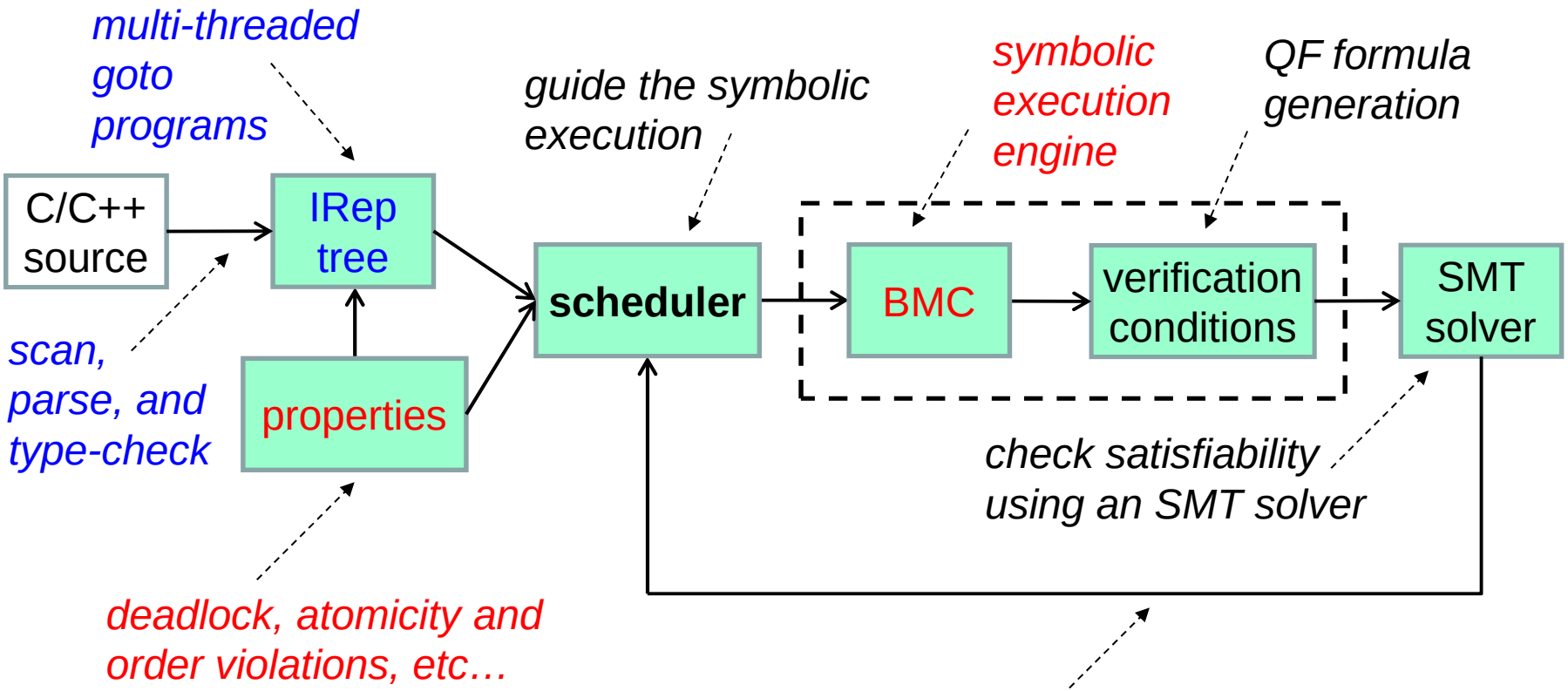
Idea: iteratively generate all possible interleavings and call the BMC procedure on each interleaving



reused/extended from the Cprover framework

Lazy exploration of interleavings

Idea: iteratively generate all possible interleavings and call the BMC procedure on each interleaving



reused/extended from the Cprover framework

Running Example

- the program has sequences of operations that need to be protected together to avoid atomicity violation
  - requirement: the region of code (*val1* and *val2*) should execute atomically

```
Thread twoStage
1:  lock(m1);
2:  val1 = 1;
3:  unlock(m1);
4:  lock(m2);
5:  val2 = val1 + 1;
6:  unlock(m2);
```

```
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));
```

A state $s \in S$ consists of the value of the program counter *pc* and the values of all program variables

program counter: 0
mutexes: m1=0; m2=0;
global variables: val1=0; val2=0;
local variabes: t1= -1; t2= -1;

Lazy exploration: interleaving I_s

statements:

val1-access:

val2-access:

```
Thread twoStage
1:  lock(m1);
2:  val1 = 1;
3:  unlock(m1);
4:  lock(m2);
5:  val2 = val1 + 1;
6:  unlock(m2);
```

```
Thread reader
7:  lock(m1);
8:  if (val1 == 0) {
9:    unlock(m1);
10:  return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));
```

program counter: 0
mutexes: m1=0; m2=0;
global variables: val1=0; val2=0;
local variabes: t1= -1; t2= -1;

Lazy exploration: interleaving I_s

statements: 1

val1-access:

val2-access:

```
Thread twoStage
1:  lock(m1);
2:  val1 = 1;
3:  unlock(m1);
4:  lock(m2);
5:  val2 = val1 + 1;
6:  unlock(m2);
```

```
Thread reader
7:  lock(m1);
8:  if (val1 == 0) {
9:    unlock(m1);
10:  return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));
```

program counter: 1
mutexes: ***m1=1****; m2=0;*
global variables: val1=0; val2=0;
local variabes: t1= -1; t2= -1;

Lazy exploration: interleaving I_s

statements: 1-2

val1-access: $W_{twoStage,2}$

val2-access:

write access to the shared variable *val1* in statement *2* of the thread *twoStage*

```
Thread twoStage
1:  lock(m1);
2:  val1 = 1;
3:  unlock(m1);
4:  lock(m2);
5:  val2 = val1 + 1;
6:  unlock(m2);
```

```
Thread reader
7:  lock(m1);
8:  if (val1 == 0) {
9:    unlock(m1);
10:  return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));
```

program counter: 2
mutexes: m1=1; m2=0;
global variables: ***val1=1****; val2=0;*
local variabes: t1= -1; t2= -1;

Lazy exploration: interleaving I_s

statements: 1-2-3

val1-access: $W_{twoStage,2}$

val2-access:

```
Thread twoStage
1:  lock(m1);
2:  val1 = 1;
3:  unlock(m1);
4:  lock(m2);
5:  val2 = val1 + 1;
6:  unlock(m2);
```

```
Thread reader
7:  lock(m1);
8:  if (val1 == 0) {
9:    unlock(m1);
10:  return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));
```

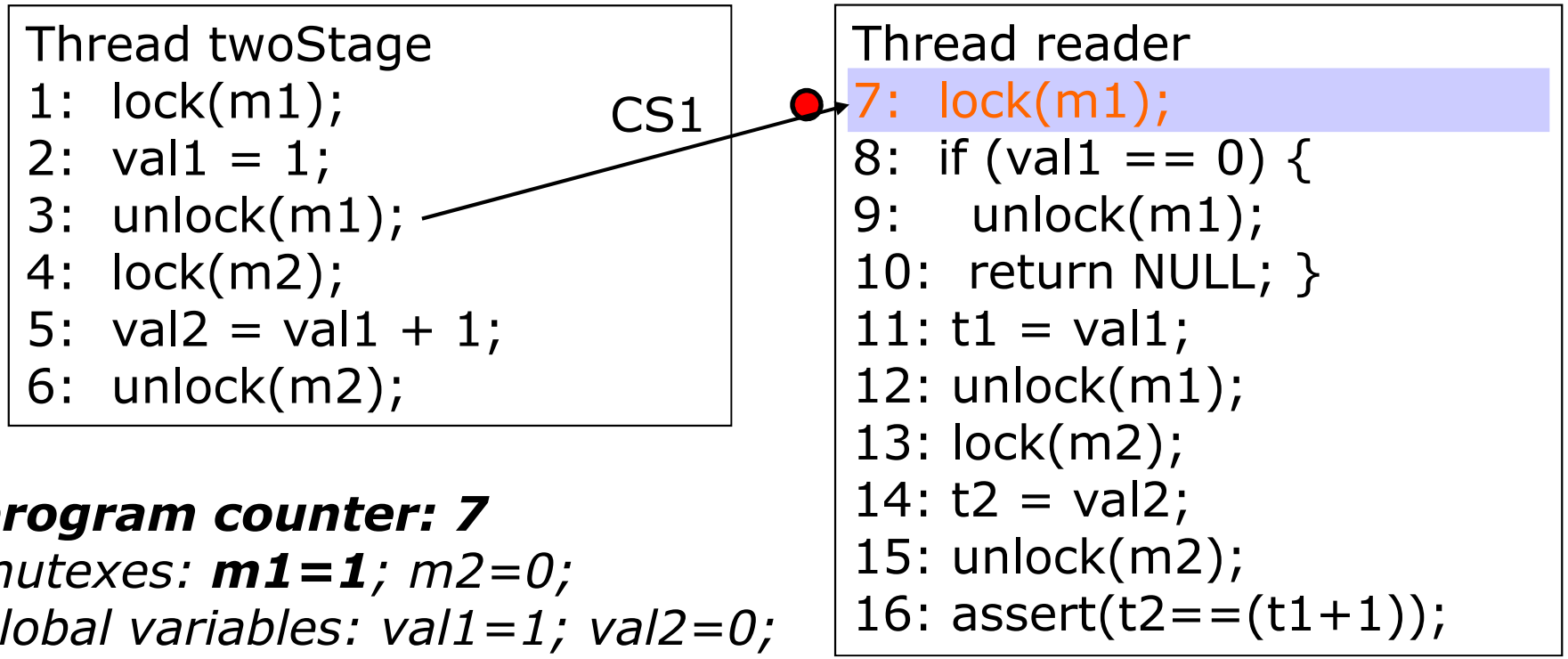
program counter: 3
mutexes: ***m1=0****; m2=0;*
global variables: val1=1; val2=0;
local variabes: t1= -1; t2= -1;

Lazy exploration: interleaving I_s

statements: 1-2-3-7

val1-access: $W_{twoStage,2}$

val2-access:



program counter: 7
mutexes: ***m1=1****; m2=0;*
global variables: val1=1; val2=0;
local variabes: t1= -1; t2= -1;

Lazy exploration: interleaving I$_6$

statements: 1-2-3-7-8

val1-access: $W_{twoStage,2}$ - $R_{reader,8}$

val2-access:

read access to the shared variable *val1* in statement *8* of the thread *reader*

```
Thread twoStage
1:  lock(m1);
2:  val1 = 1;
3:  unlock(m1);
4:  lock(m2);
5:  val2 = val1 + 1;
6:  unlock(m2);
```

CS1

```
Thread reader
7:  lock(m1);
8:  if (val1 == 0) {
9:    unlock(m1);
10:  return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));
```

program counter: 8

mutexes: m1=1; m2=0;

global variables: val1=1; val2=0;

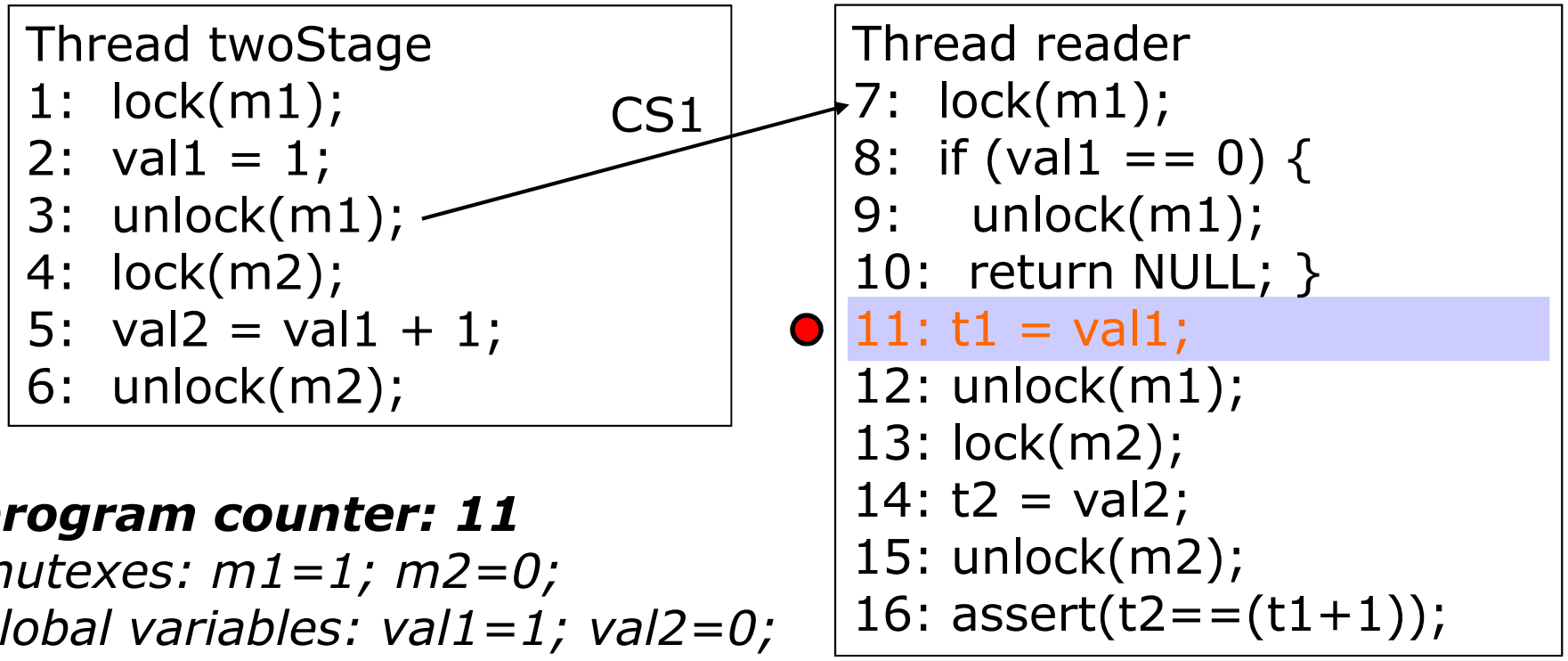
local variabes: t1= -1; t2= -1;

Lazy exploration: interleaving I_s

statements: 1-2-3-7-8-11

val1-access: $W_{twoStage,2}$ - $R_{reader,8}$ - $R_{reader,11}$

val2-access:



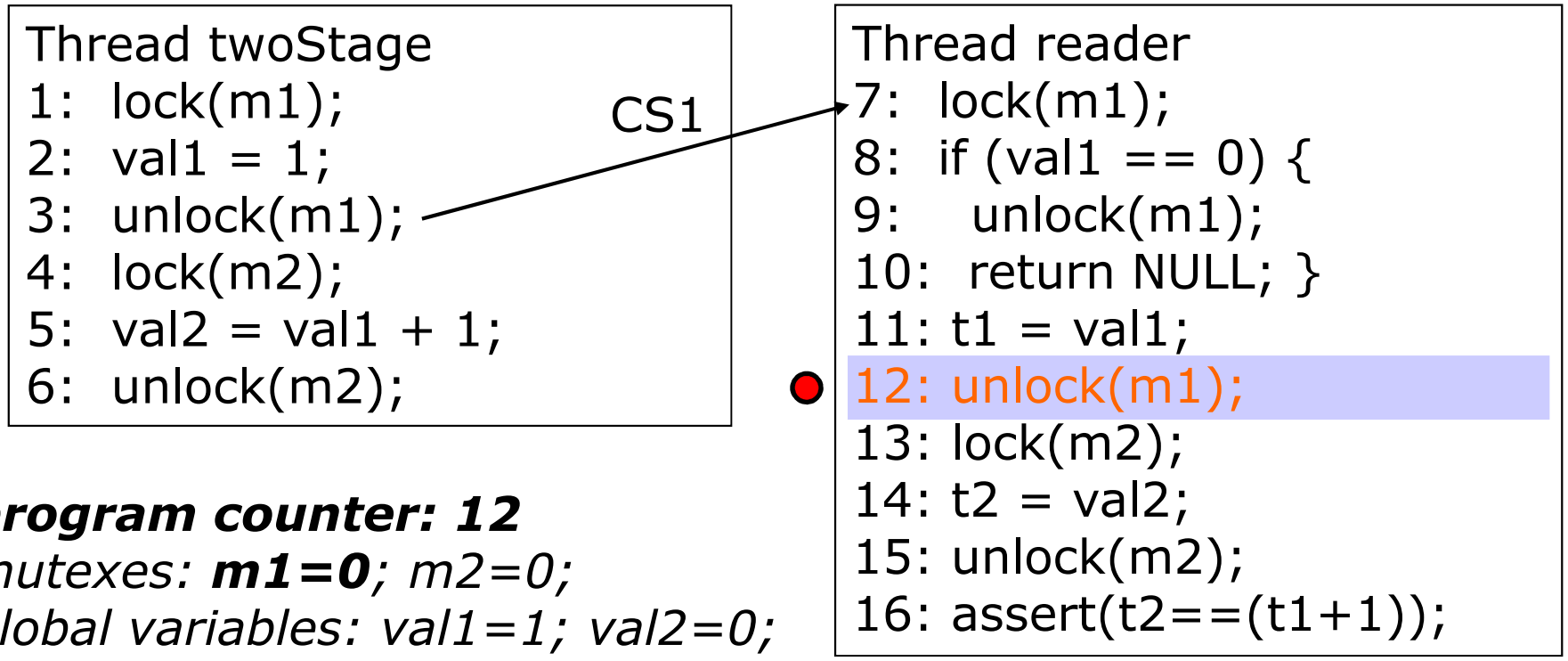
program counter: 11
mutexes: m1=1; m2=0;
global variables: val1=1; val2=0;
local variabes: ***t1= 1****; t2= -1;*

Lazy exploration: interleaving I_s

statements: 1-2-3-7-8-11-12

val1-access: $W_{twoStage,2}$ - $R_{reader,8}$ - $R_{reader,11}$

val2-access:



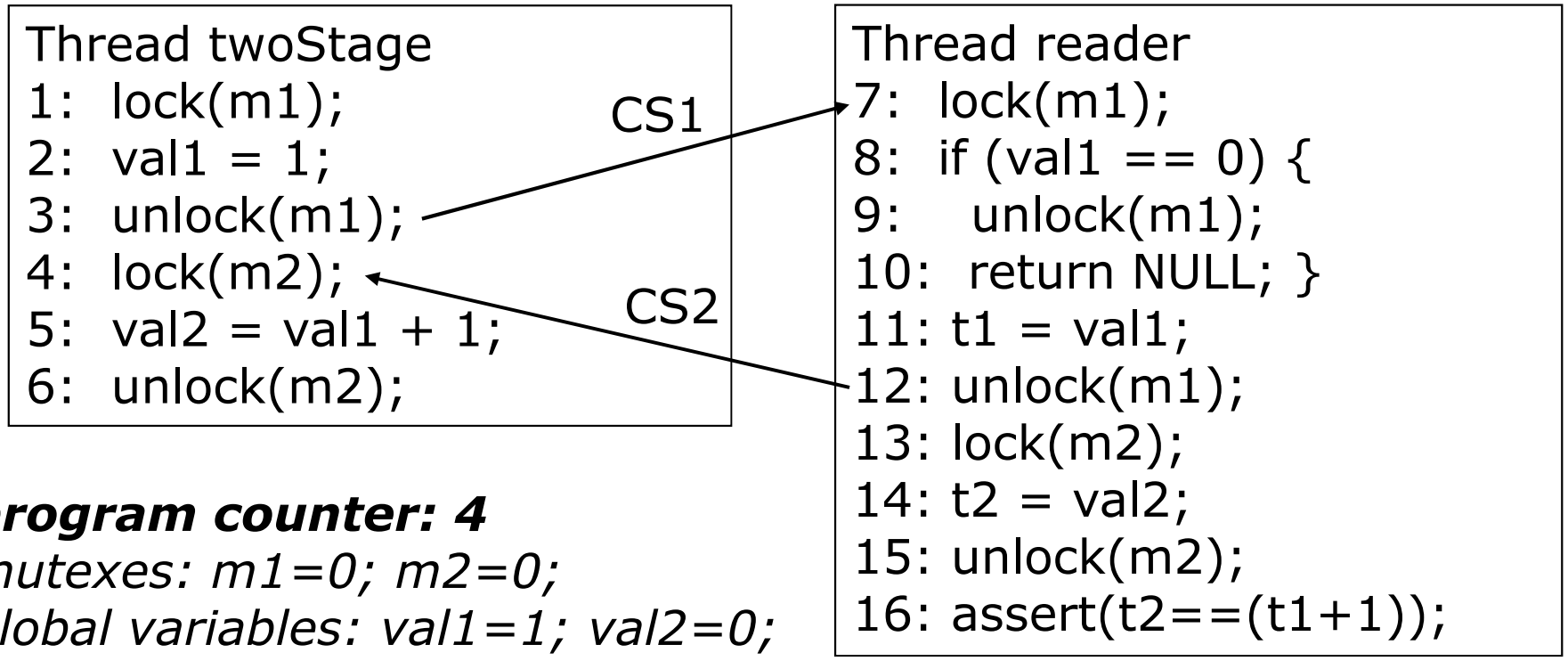
program counter: 12
mutexes: ***m1=0****; m2=0;*
global variables: val1=1; val2=0;
local variabes: t1= 1; t2= -1;

Lazy exploration: interleaving I_s

statements: 1-2-3-7-8-11-12

val1-access: $W_{twoStage,2}$ - $R_{reader,8}$ - $R_{reader,11}$

val2-access:



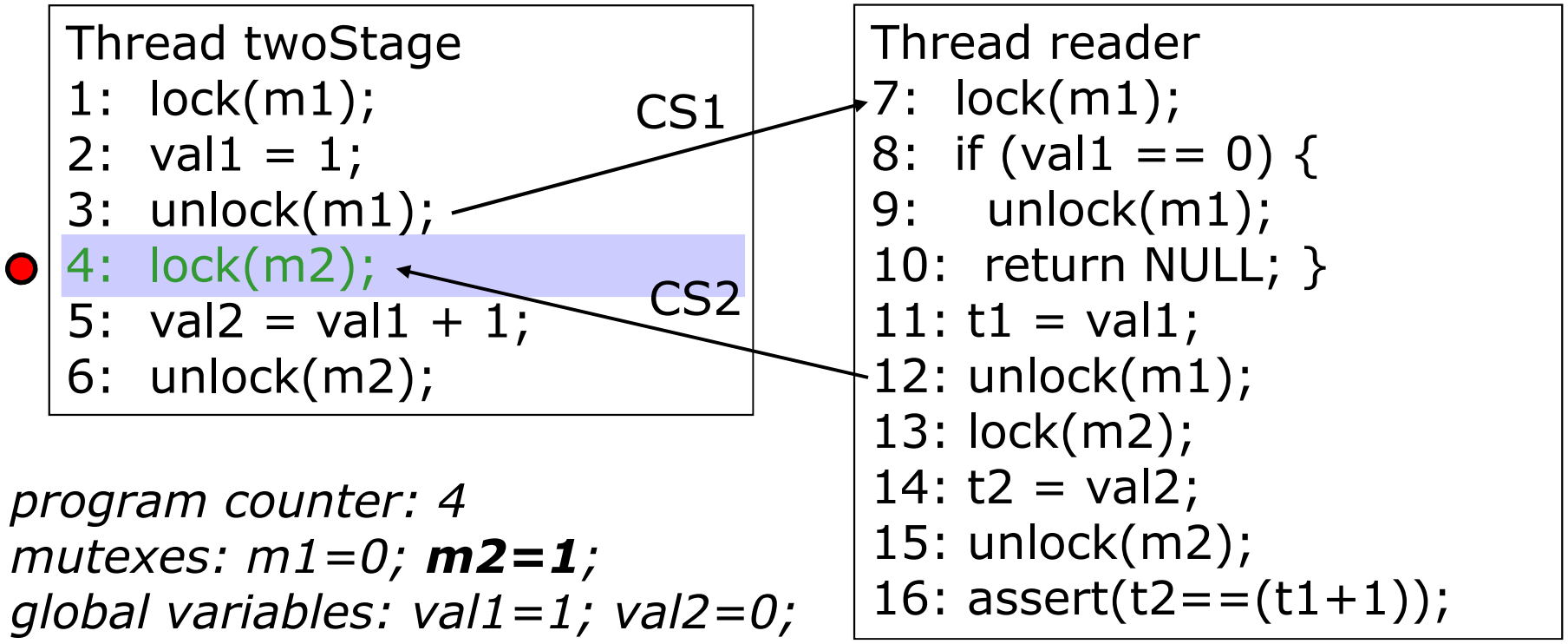
program counter: 4
mutexes: m1=0; m2=0;
global variables: val1=1; val2=0;
local variabes: t1= 1; t2= -1;

Lazy exploration: interleaving I_s

statements: 1-2-3-7-8-11-12-4

val1-access: $W_{twoStage,2}$ - $R_{reader,8}$ - $R_{reader,11}$

val2-access:



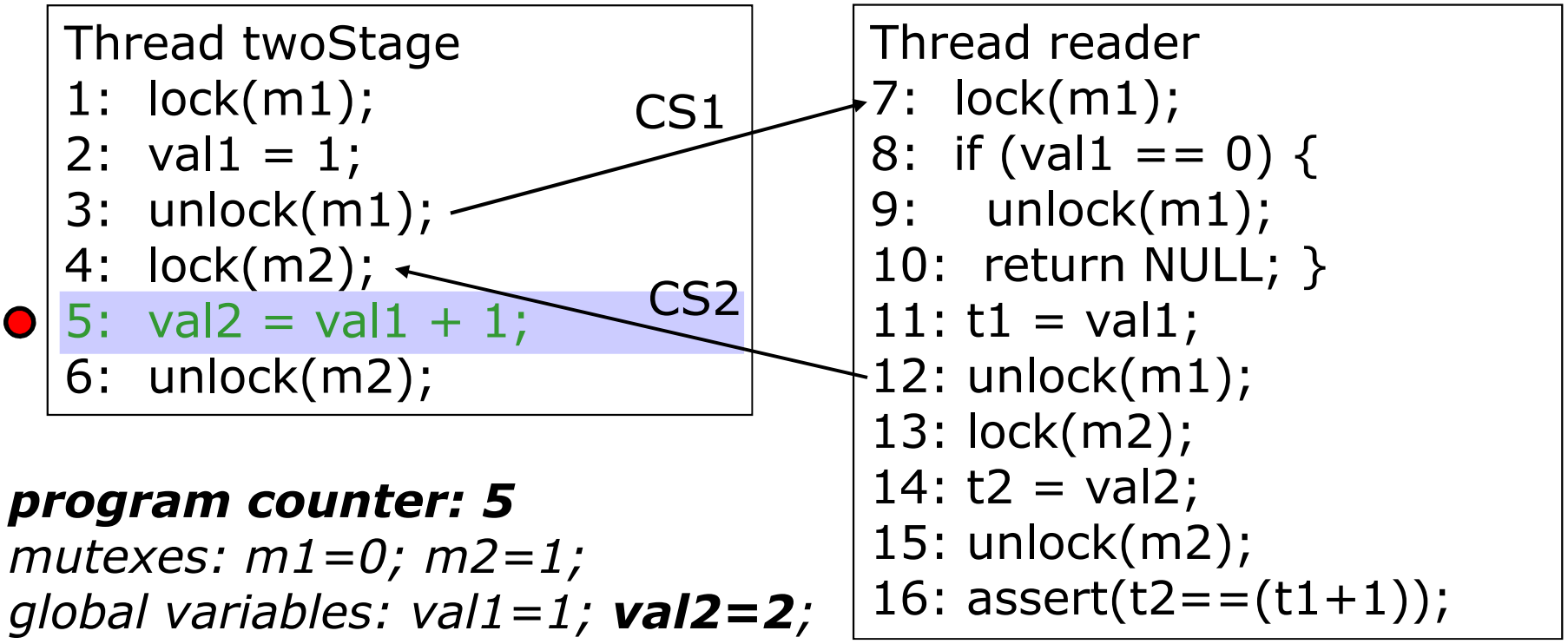
program counter: 4
*mutexes: m1=0; **m2=1**;*
global variables: val1=1; val2=0;
local variabes: t1= 1; t2= -1;

Lazy exploration: interleaving I_s

statements: 1-2-3-7-8-11-12-4-5

val1-access: $W_{twoStage,2}$ - $R_{reader,8}$ - $R_{reader,11}$ - $R_{twoStage,5}$

val2-access: $W_{twoStage,5}$



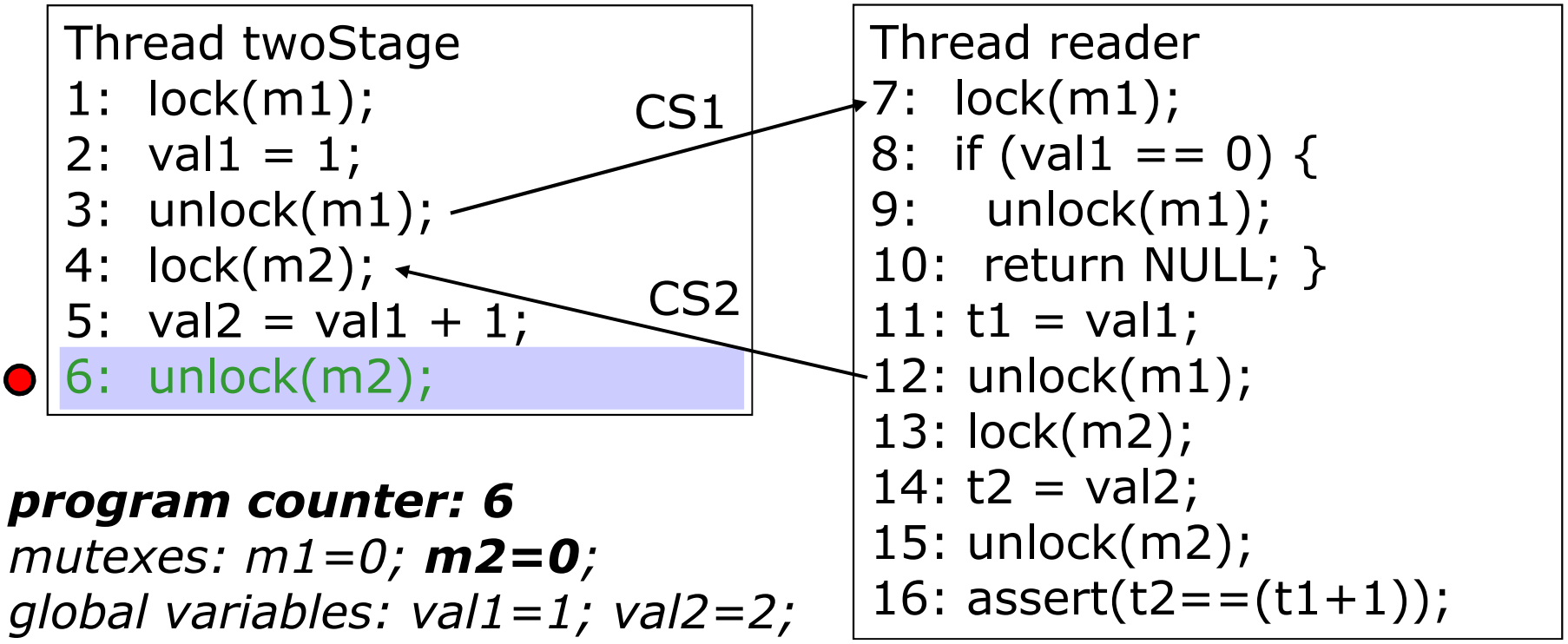
program counter: 5
mutexes: m1=0; m2=1;
global variables: val1=1; ***val2=2****;*
local variabes: t1= 1; t2= -1;

Lazy exploration: interleaving I_s

statements: 1-2-3-7-8-11-12-4-5-6

val1-access: $W_{twoStage,2}$ - $R_{reader,8}$ - $R_{reader,11}$ - $R_{twoStage,5}$

val2-access: $W_{twoStage,5}$



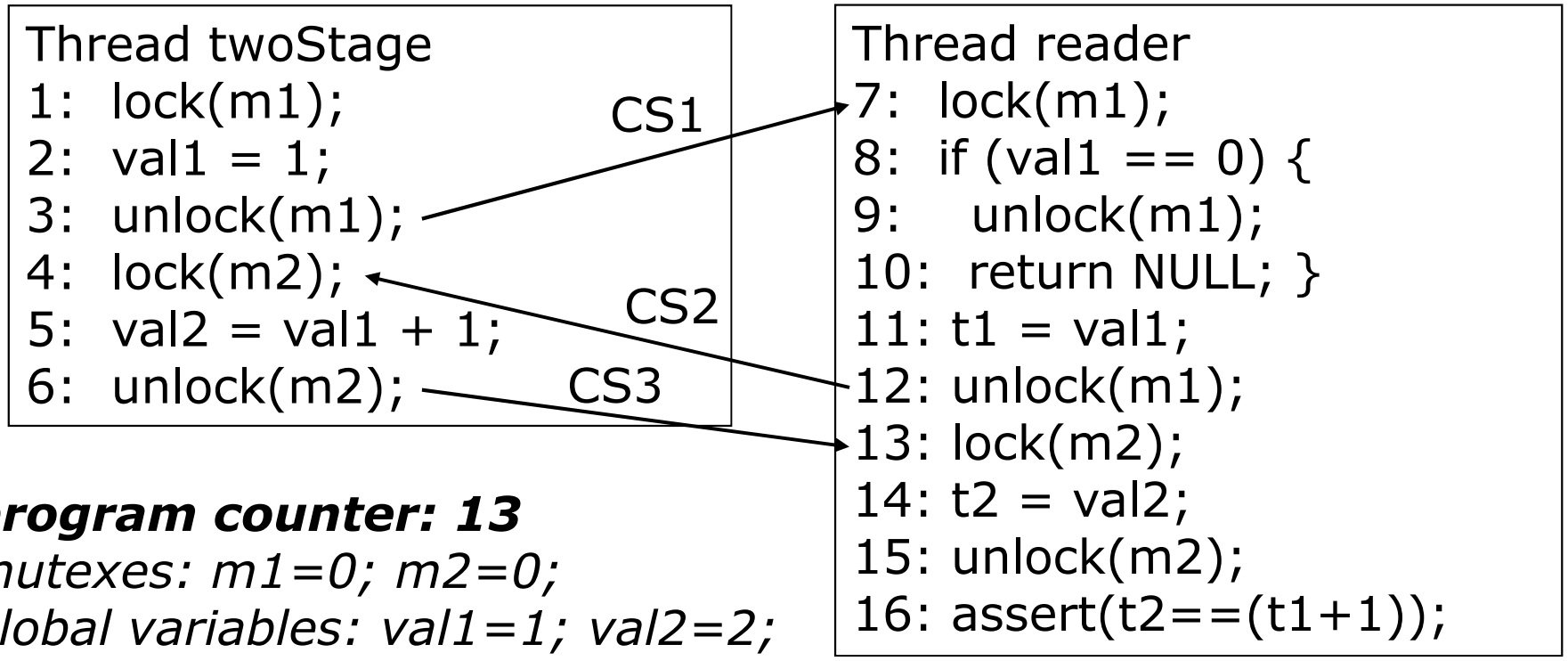
program counter: 6
*mutexes: m1=0; **m2=0**;*
global variables: val1=1; val2=2;
local variabes: t1= 1; t2= -1;

Lazy exploration: interleaving I_s

statements: 1-2-3-7-8-11-12-4-5-6

val1-access: $W_{twoStage,2}$ - $R_{reader,8}$ - $R_{reader,11}$ - $R_{twoStage,5}$

val2-access: $W_{twoStage,5}$



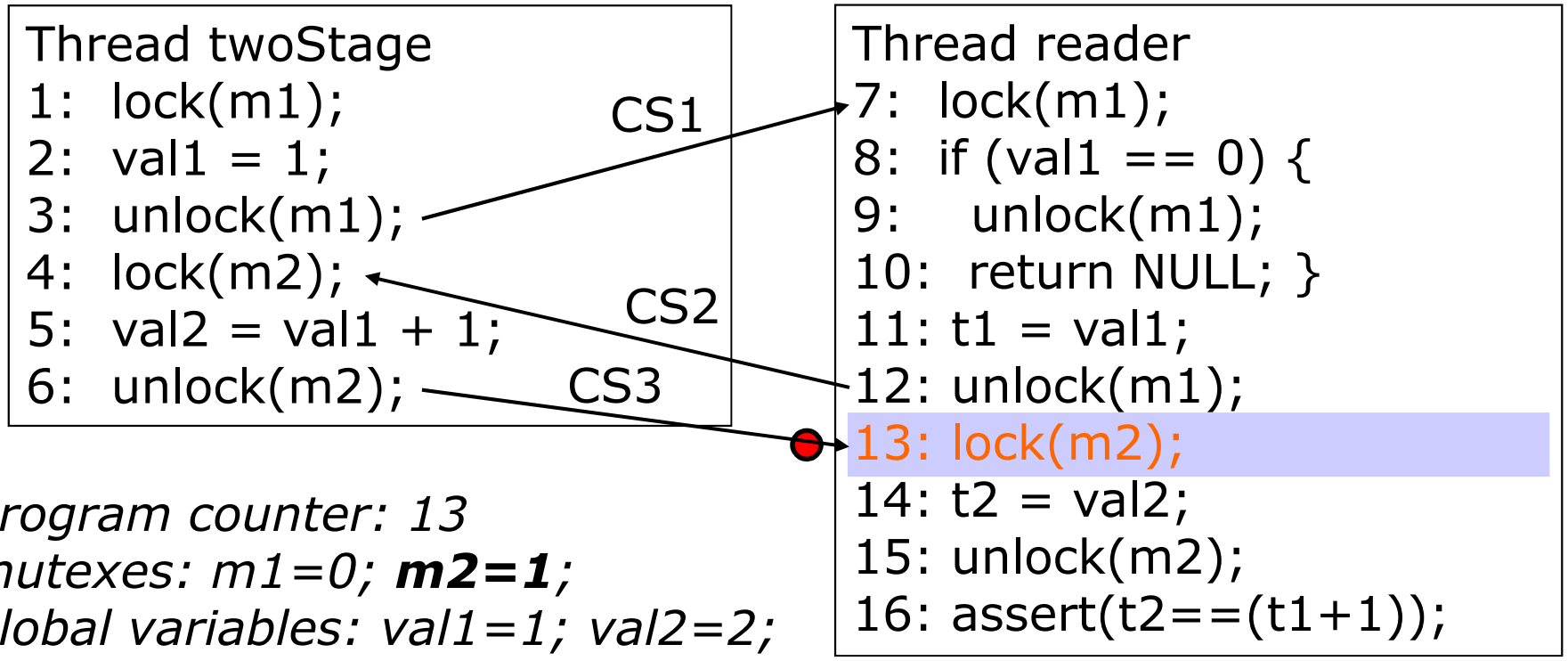
program counter: 13
mutexes: m1=0; m2=0;
global variables: val1=1; val2=2;
local variabes: t1= 1; t2= -1;

Lazy exploration: interleaving I_s

statements: 1-2-3-7-8-11-12-4-5-6-13

val1-access: $W_{twoStage,2}$ - $R_{reader,8}$ - $R_{reader,11}$ - $R_{twoStage,5}$

val2-access: $W_{twoStage,5}$



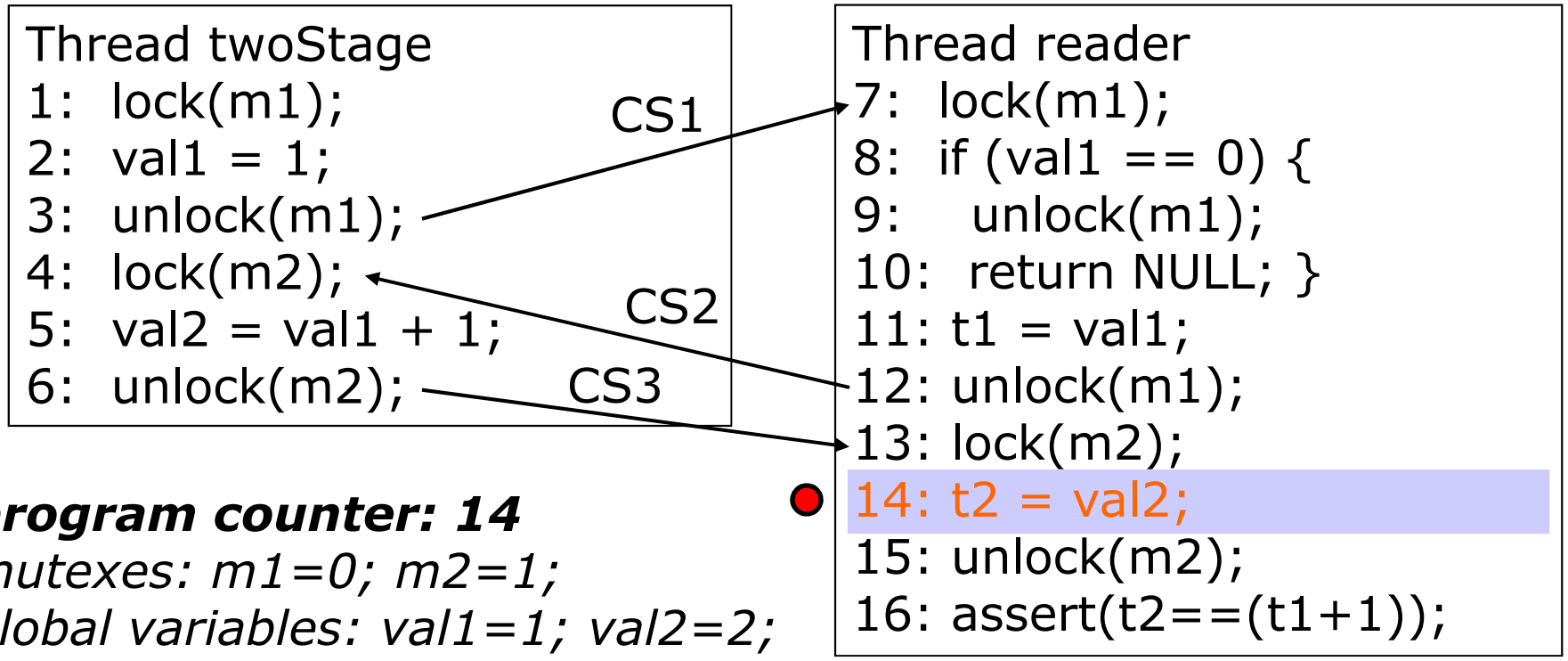
program counter: 13
*mutexes: m1=0; **m2=1**;*
global variables: val1=1; val2=2;
local variabes: t1= 1; t2= -1;

Lazy exploration: interleaving I_s

statements: 1-2-3-7-8-11-12-4-5-6-13-14

val1-access: $W_{twoStage,2}$ - $R_{reader,8}$ - $R_{reader,11}$ - $R_{twoStage,5}$

val2-access: $W_{twoStage,5}$ - $R_{reader,14}$



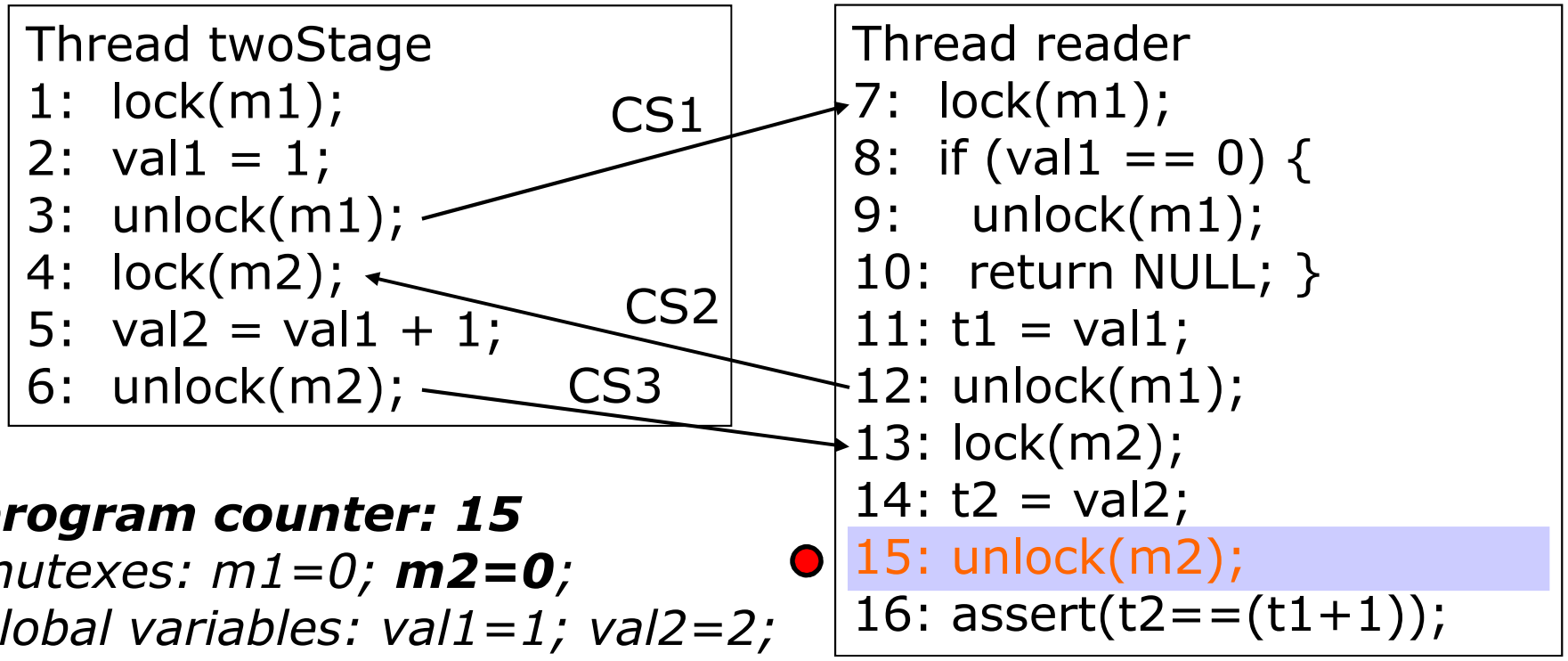
program counter: 14
mutexes: m1=0; m2=1;
global variables: val1=1; val2=2;
local variabes: t1= 1; ***t2= 2;***

Lazy exploration: interleaving I_s

statements: 1-2-3-7-8-11-12-4-5-6-13-14-15

val1-access: $W_{twoStage,2}$ - $R_{reader,8}$ - $R_{reader,11}$ - $R_{twoStage,5}$

val2-access: $W_{twoStage,5}$ - $R_{reader,14}$



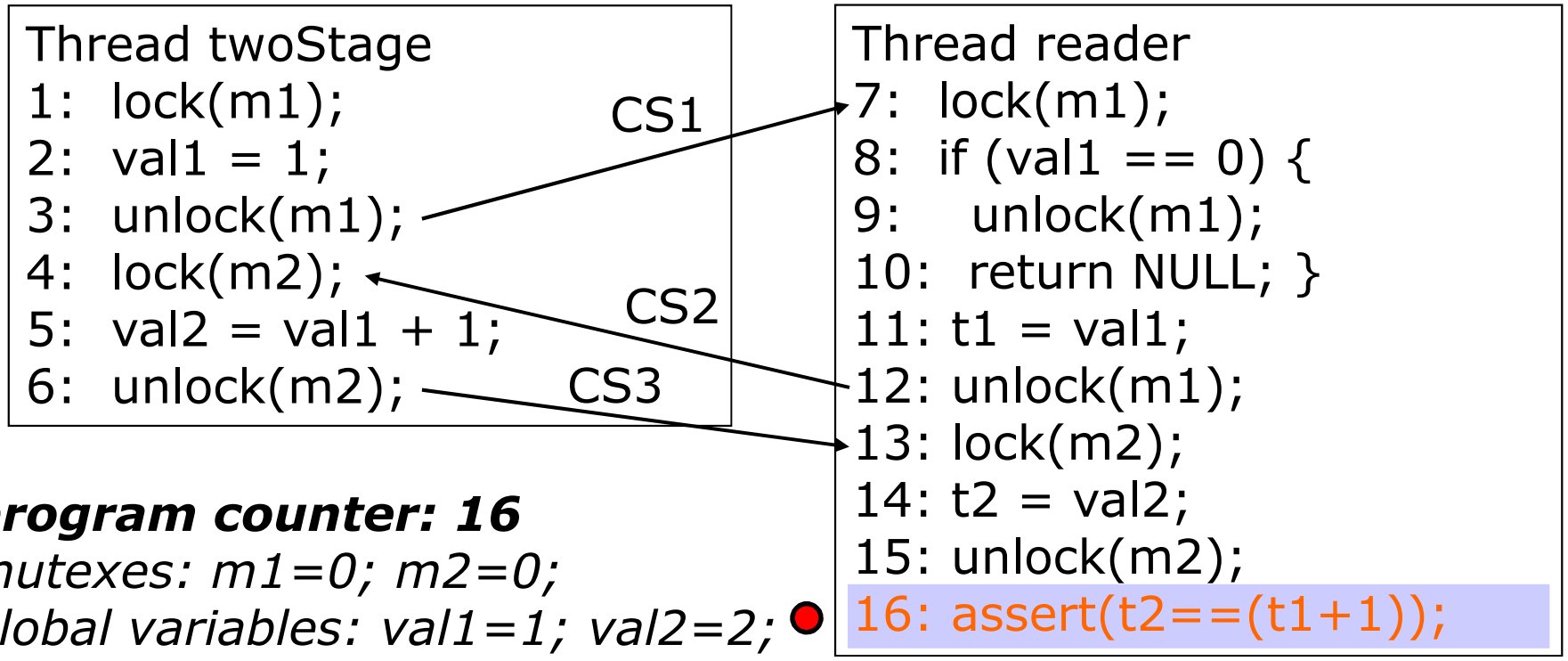
program counter: 15
*mutexes: m1=0; **m2=0**;*
global variables: val1=1; val2=2;
local variabes: t1= 1; t2= 2;

Lazy exploration: interleaving I_s

statements: 1-2-3-7-8-11-12-4-5-6-13-14-15-16

val1-access: $W_{twoStage,2}$ - $R_{reader,8}$ - $R_{reader,11}$ - $R_{twoStage,5}$

val2-access: $W_{twoStage,5}$ - $R_{reader,14}$



program counter: 16

mutexes: m1=0; m2=0;

global variables: val1=1; val2=2;

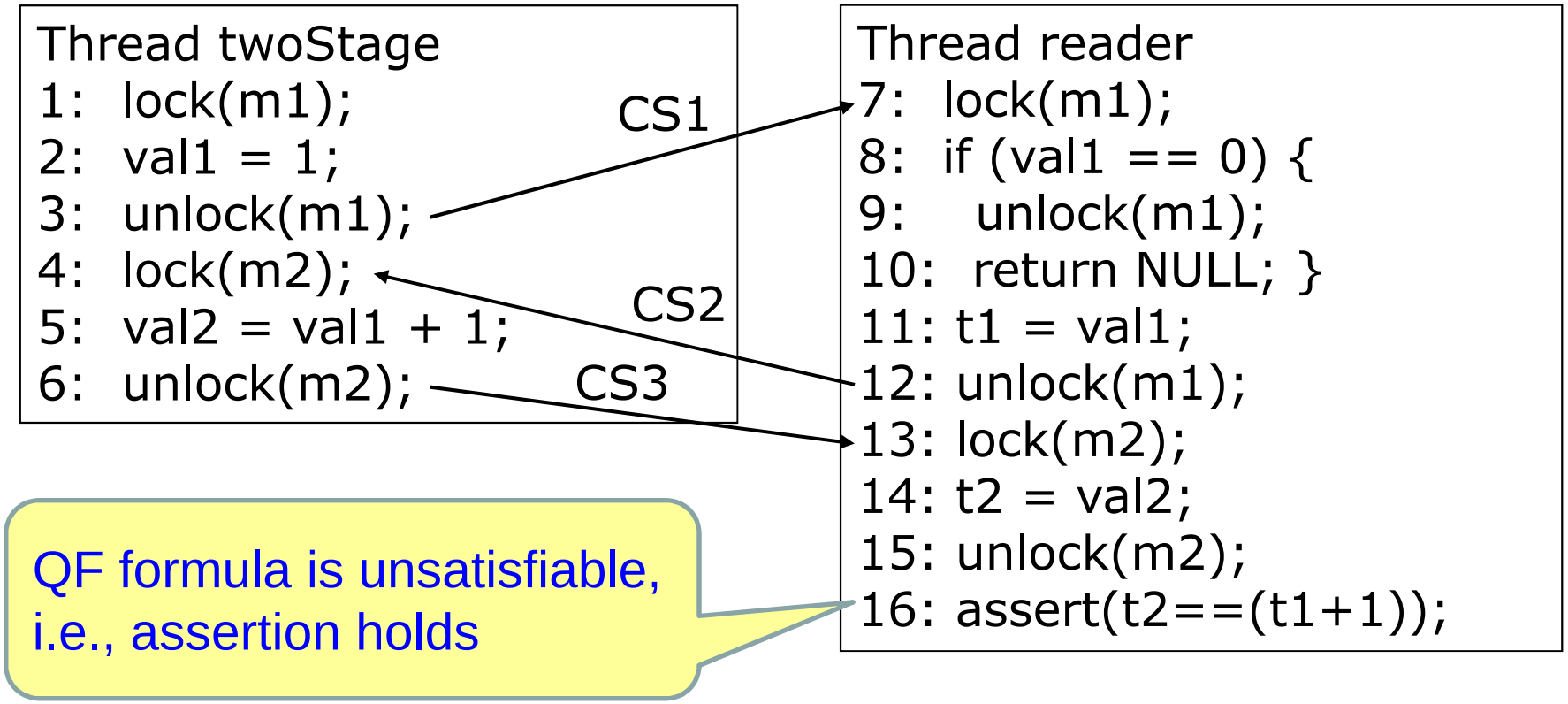
local variabes: t1= 1; t2= 2;

Lazy exploration: interleaving I_s

statements: 1-2-3-7-8-11-12-4-5-6-13-14-15-16

val1-access: $W_{twoStage,2}$ - $R_{reader,8}$ - $R_{reader,11}$ - $R_{twoStage,5}$

val2-access: $W_{twoStage,5}$ - $R_{reader,14}$



Lazy exploration: interleaving I_f

statements:

val1-access:

val2-access:

```
Thread twoStage
1:  lock(m1);
2:  val1 = 1;
3:  unlock(m1);
4:  lock(m2);
5:  val2 = val1 + 1;
6:  unlock(m2);
```

```
Thread reader
7:  lock(m1);
8:  if (val1 == 0) {
9:    unlock(m1);
10:  return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));
```

program counter: 0
mutexes: m1=0; m2=0;
global variables: val1=0; val2=0;
local variabes: t1= -1; t2= -1;

Lazy exploration: interleaving I_f

statements: 1-2-3

val1-access: $W_{twoStage,2}$

val2-access:

```
Thread twoStage
1:  lock(m1);
2:  val1 = 1;
3:  unlock(m1);
4:  lock(m2);
5:  val2 = val1 + 1;
6:  unlock(m2);
```

```
Thread reader
7:  lock(m1);
8:  if (val1 == 0) {
9:    unlock(m1);
10:  return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));
```

program counter: 3
mutexes: m1=0; m2=0;
global variables: ***val1=1****; val2=0;*
local variabes: t1= -1; t2= -1;

Lazy exploration: interleaving I_f

statements: 1-2-3

val1-access: $W_{twoStage,2}$

val2-access:

```
Thread twoStage
1:  lock(m1);                CS1
2:  val1 = 1;
3:  unlock(m1);
4:  lock(m2);
5:  val2 = val1 + 1;
6:  unlock(m2);
```

```
Thread reader
7:  lock(m1);
8:  if (val1 == 0) {
9:    unlock(m1);
10:  return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));
```

program counter: 7
mutexes: m1=0; m2=0;
global variables: val1=1; val2=0;
local variabes: t1= -1; t2= -1;

Lazy exploration: interleaving I_f

statements: 1-2-3-7-8-11-12-13-14-15-16

val1-access: $W_{twoStage,2}$- $R_{reader,8}$- $R_{reader,11}$

val2-access: $R_{reader,14}$

```
Thread twoStage
1:  lock(m1);                    CS1
2:  val1 = 1;
3:  unlock(m1);
4:  lock(m2);
5:  val2 = val1 + 1;
6:  unlock(m2);
```

```
Thread reader
7:  lock(m1);
8:  if (val1 == 0) {
9:    unlock(m1);
10:  return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));
```

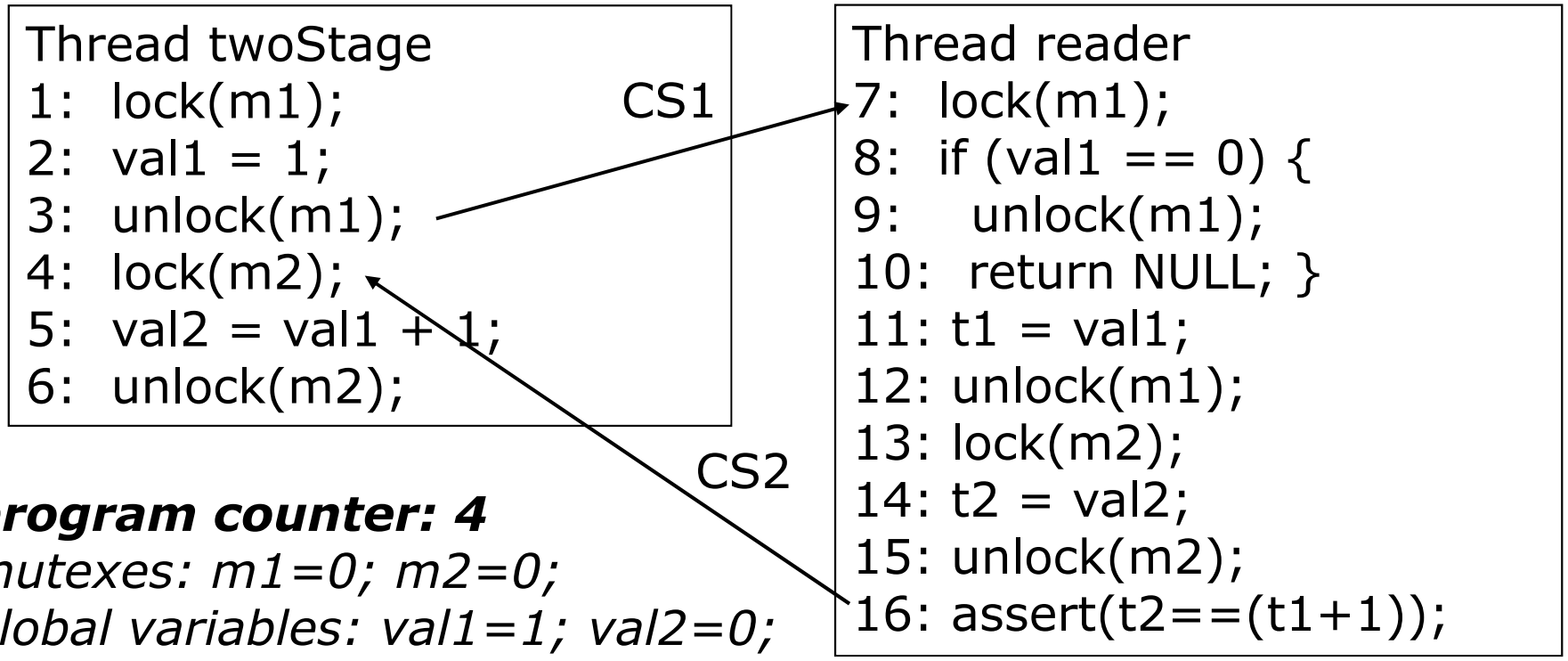
program counter: 16
mutexes: m1=0; m2=0;
global variables: val1=1; val2=0;
local variabes: ***t1= 1; t2= 0;***

Lazy exploration: interleaving I_f

statements: 1-2-3-7-8-11-12-13-14-15-16

val1-access: $W_{twoStage,2}$- $R_{reader,8}$- $R_{reader,11}$

val2-access: $R_{reader,14}$



program counter: 4

mutexes: m1=0; m2=0;

global variables: val1=1; val2=0;

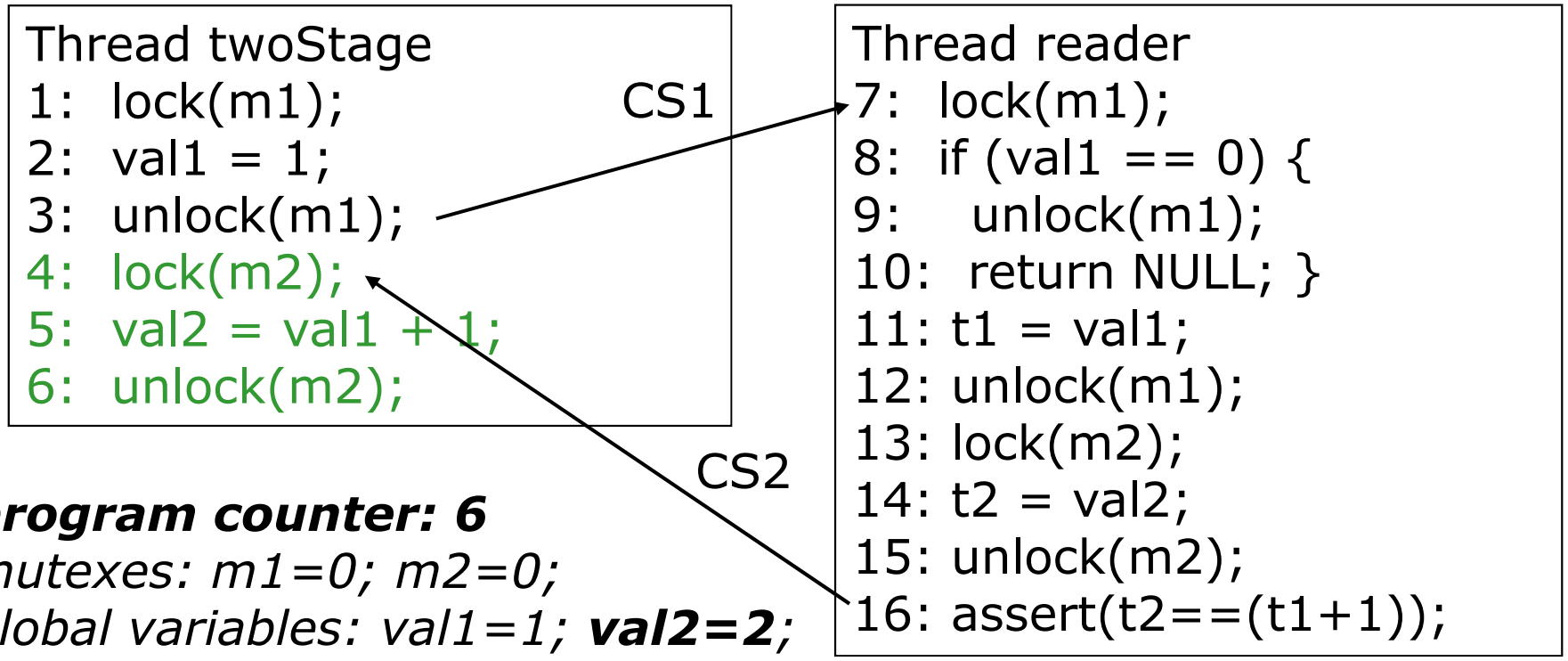
local variabes: t1= 1; t2= 0;

Lazy exploration: interleaving I_f

statements: 1-2-3-7-8-11-12-13-14-15-16-4-5-6

val1-access: $W_{twoStage,2}$- $R_{reader,8}$- $R_{reader,11}$ - $R_{twoStage,5}$

val2-access: $R_{reader,14}$- $W_{twoStage,5}$



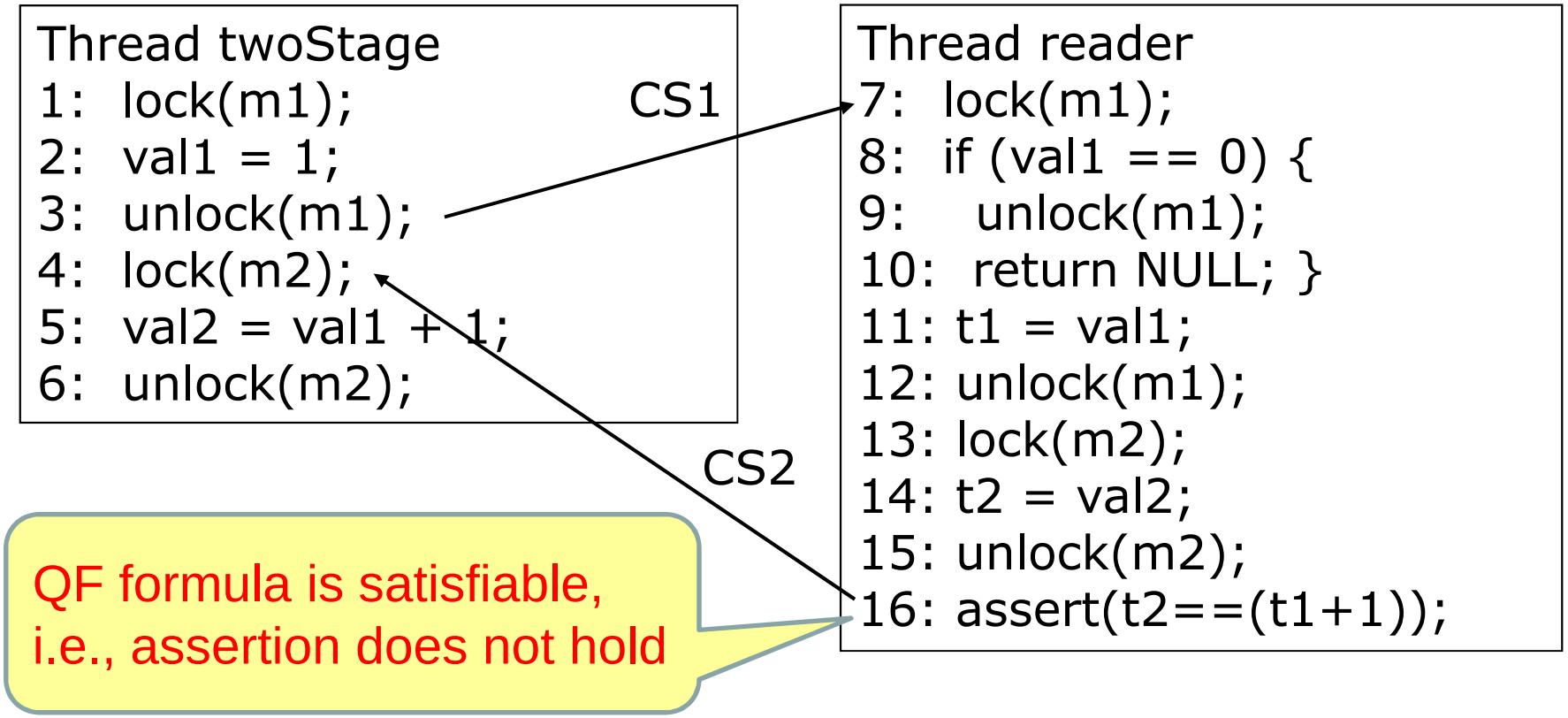
program counter: 6
mutexes: m1=0; m2=0;
global variables: val1=1; ***val2=2****;*
local variabes: t1= 1; t2= 0;

Lazy exploration: interleaving I_f

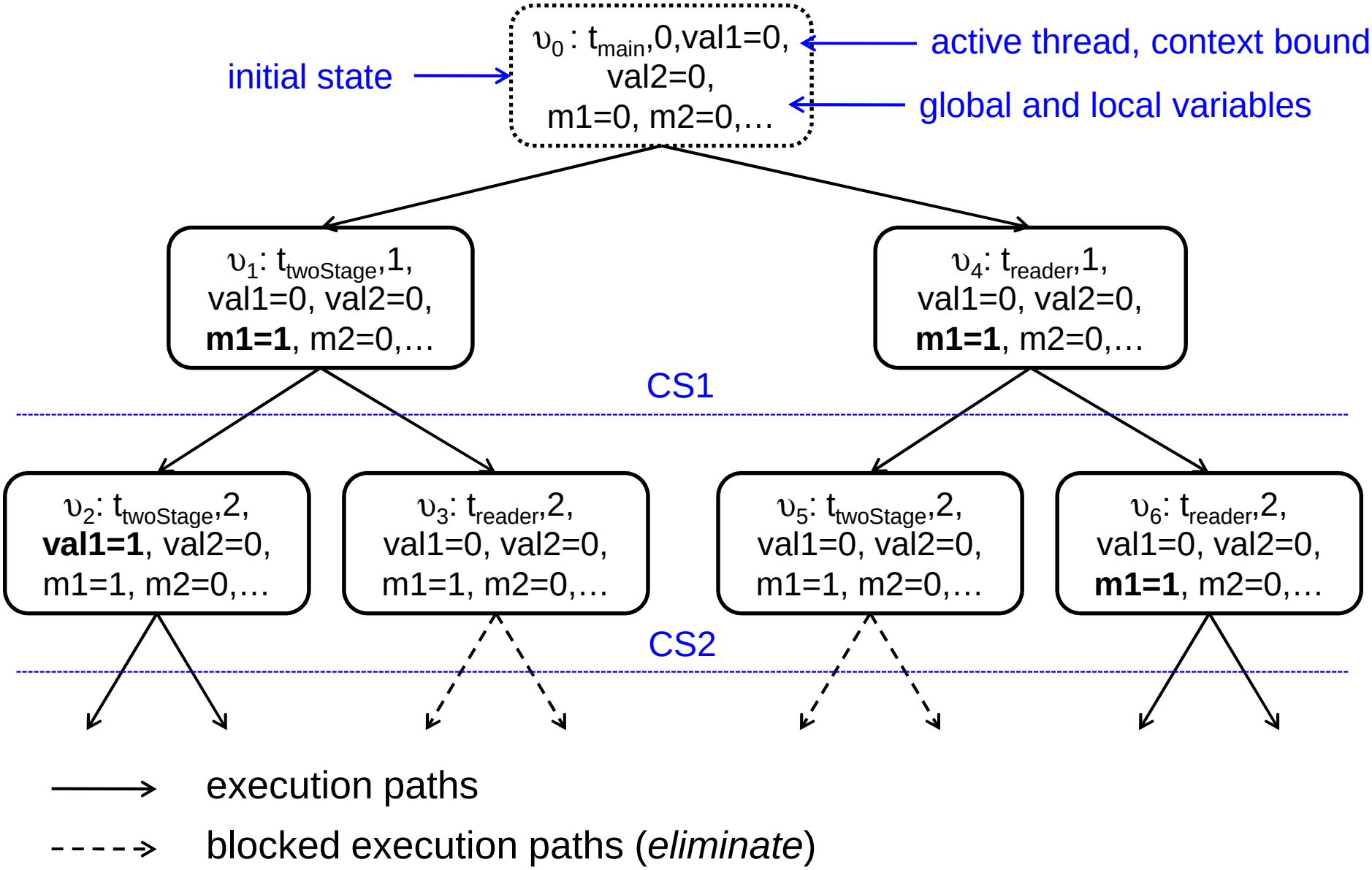
statements: 1-2-3-7-8-11-12-13-14-15-16-4-5-6

val1-access: $W_{twoStage,2}$- $R_{reader,8}$- $R_{reader,11}$ - $R_{twoStage,5}$

val2-access: $R_{reader,14}$- $W_{twoStage,5}$



Lazy Approach: State Transitions



Exploring the Reachability Tree

- use a reachability tree (RT) to describe reachable states of a multi-threaded program
- each node in the RT is a tuple $v = \left(A_i, C_i, s_i, \left\langle l_i^j, G_i^j \right\rangle_{j=1}^{n} \right)_i$ for a given time step i, where:
  - A_i represents the currently active thread
  - C_i represents the context switch number
  - s_i represents the current state
  - l_i^j represents the current location of thread *j*
  - G_i^j represents the control flow guards accumulated in thread *j* along the path from l_0^j to l_i^j
- expand the RT by executing symbolically each instruction of the multi-threaded program

Expansion Rules of the RT

R1 (assign): If *l* is an assignment, we execute *l*, which generates s_{i+1}. We add as child to υ a new node υ'

$$\upsilon' = \left(A_i, C_i, s_{i+1}, \left\langle l_{i+1}^{j}, G_i^{j} \right\rangle\right)_{i+1} \qquad l_{i+1}^{A_i} = l_i^{A_i} + 1$$

- we have fully expanded υ if
  - I within an atomic block; or
  - I contains no global variable; or
  - the upper bound of context switches ($C_i = C$) is reached
- if υ is not fully expanded, for each thread $j \neq A_i$ where G_i^j is enabled in s_{i+1}, we thus create a new child node

$$\upsilon_j^{'} = \left(j, C_i + 1, s_{i+1}, \left\langle l_i^j, G_i^j \right\rangle \right)_{i+1}$$

Expansion Rules of the RT

R2 (skip): If *l* is a *skip*-statement with target *l*, we increment the location of the current thread and continue with it. We explore no context switches:

$$v' = \left(A_i, C_i, s_i, \left\langle \underline{l_{i+1}^{j}}, G_i^{j} \right\rangle\right)_{i+1} \longrightarrow l_{i+1}^{j} = \begin{cases} l_i^{j} + 1 & : \quad j = A_i \\ l_i^{j} & : \quad otherwise \end{cases}$$

R3 (unconditional goto): If *l* is an unconditional *goto*-statement with target *l*, we set the location of the current thread and continue with it. We explore no context switches:

$$v' = \left(A_i, C_i, s_i, \left\langle \underline{l_{i+1}^{j}}, G_i^{j} \right\rangle\right)_{i+1} \longrightarrow l_{i+1}^{j} = \begin{cases} l & : \quad j = A_i \\ l_i^{j} & : \quad otherwise \end{cases}$$

Expansion Rules of the RT

R4 (conditional goto): If *l* is a conditional *goto*-statement with test *c and* target *l*, we create two child nodes υ' and υ''.

– for υ' , we assume that *c* is *true* and proceed with the target instruction of the jump:

$$\upsilon' = \left(A_i, C_i, s_i, \left\langle l_{i+1}^{j}, c \wedge G_i^{j} \right\rangle\right)_{i+1} \longrightarrow l_{i+1}^{j} = \begin{cases} l & : \quad j = A_i \\ l_i^{j} & : \quad otherwise \end{cases}$$

– for υ'', we add $\neg c$ to the guards and continue with the next instruction in the current thread

$$\upsilon'' = \left(A_i, C_i, s_i, \left\langle l_{i+1}^{j}, \neg c \wedge G_i^{j} \right\rangle\right)_{i+1} \longrightarrow l_{i+1}^{j} = \begin{cases} l_i^{j} + 1 & : \quad j = A_i \\ l_i^{j} & : \quad otherwise \end{cases}$$

– prune one of the nodes if the condition is determined statically

Expansion Rules of the RT

R5 (assume): If *I* is an *assume*-statement with argument *c*, we proceed similar to R1

- we continue with the unchanged state s_i but add *c* to all guards, as described in R4
- If $c \wedge G_i^j$ evaluates to *false*, we prune the execution path

R6 (assert): If *I* is an *assert*-statement with argument *c*, we proceed similar to R1.

- we continue with the unchanged state s_i but add *c* to all guards, as described in R4
- we generate a verification condition to check the validity of *c*

Expansion Rules of the RT

R5 (start_thread): If *l* is a *start_thread* instruction, we add the indicated thread to the set of active threads:

$$v' = \left(A_i, C_i, s_i, \left\langle l_{i+1}^{j}, G_{i+1}^{j} \right\rangle_{j=1}^{n+1} \right)_{i+1}$$

- where l_{i+1}^{n+1} is the initial location of the thread and $G_{i+1}^{n+1} = G_i^{A_i}$
- the thread starts with the guards of the currently active thread

R6 (join_thread): If *l* is a *join_thread* instruction with argument *Id*, we add a child node:

$$v' = \left(A_i, C_i, s_i, \left\langle l_{i+1}^{j}, G_i^{j} \right\rangle \right)_{i+1}$$

- where $l_{i+1}^{j} = l_i^{A_i} + 1$ only if the joining thread Id has exited

Lazy exploration of interleavings

- Main steps of the algorithm:

1. Initialize the stack with the initial node ν_0 and the initial path $\pi_0 = \langle \upsilon_0 \rangle$
2. If the stack is empty, terminate with “no error”.
3. Pop the current node υ and current path π off the stack and compute the set υ’ of successors of υ using rules R1-R8.
4. If υ’ is empty, derive the VC φ_k^π for π and call the SMT solver on it. If φ_k^π is satisfiable, terminate with “error”; otherwise, goto step 2.
5. If υ’ is not empty, then for each node $\upsilon \in \upsilon$’, add ν to π, and push node and extended path on the stack. goto step 3.

computation path

$$\pi = \{\upsilon_1, \ldots \upsilon_n\}$$

$$\varphi_k^\pi = \overbrace{I(s_0) \wedge R(s_0, s_1) \wedge \ldots \wedge R(s_{k-1}, s_k)}^{\text{constraints}} \wedge \overbrace{\neg \phi_k}^{\text{property}}$$

bound

Observations about the lazy approach

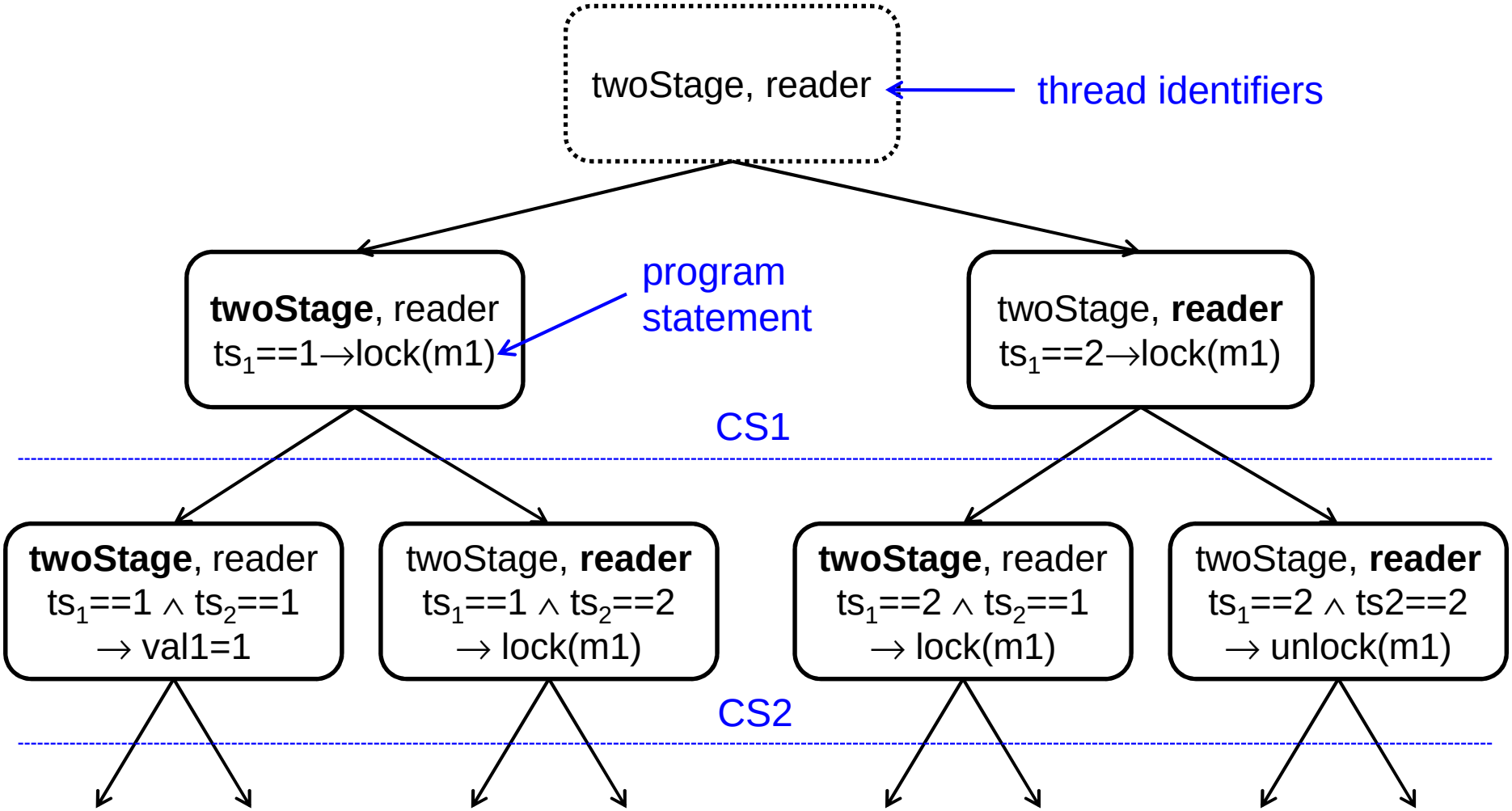
- naïve but useful:
  - bugs usually manifest with few context switches [Qadeer&Rehof'05]
  - keep in memory the parent nodes of all unexplored paths only
  - exploit which transitions are enabled in a given state
  - bound the number of preemptions (C) allowed per threads
    - ▷ *number of executions: $O(n^c)$*
  - as each formula corresponds to one possible path only, its size is relatively small
- can suffer performance degradation:
  - in particular for correct programs where we need to invoke the SMT solver once for each possible execution path

Schedule Recording

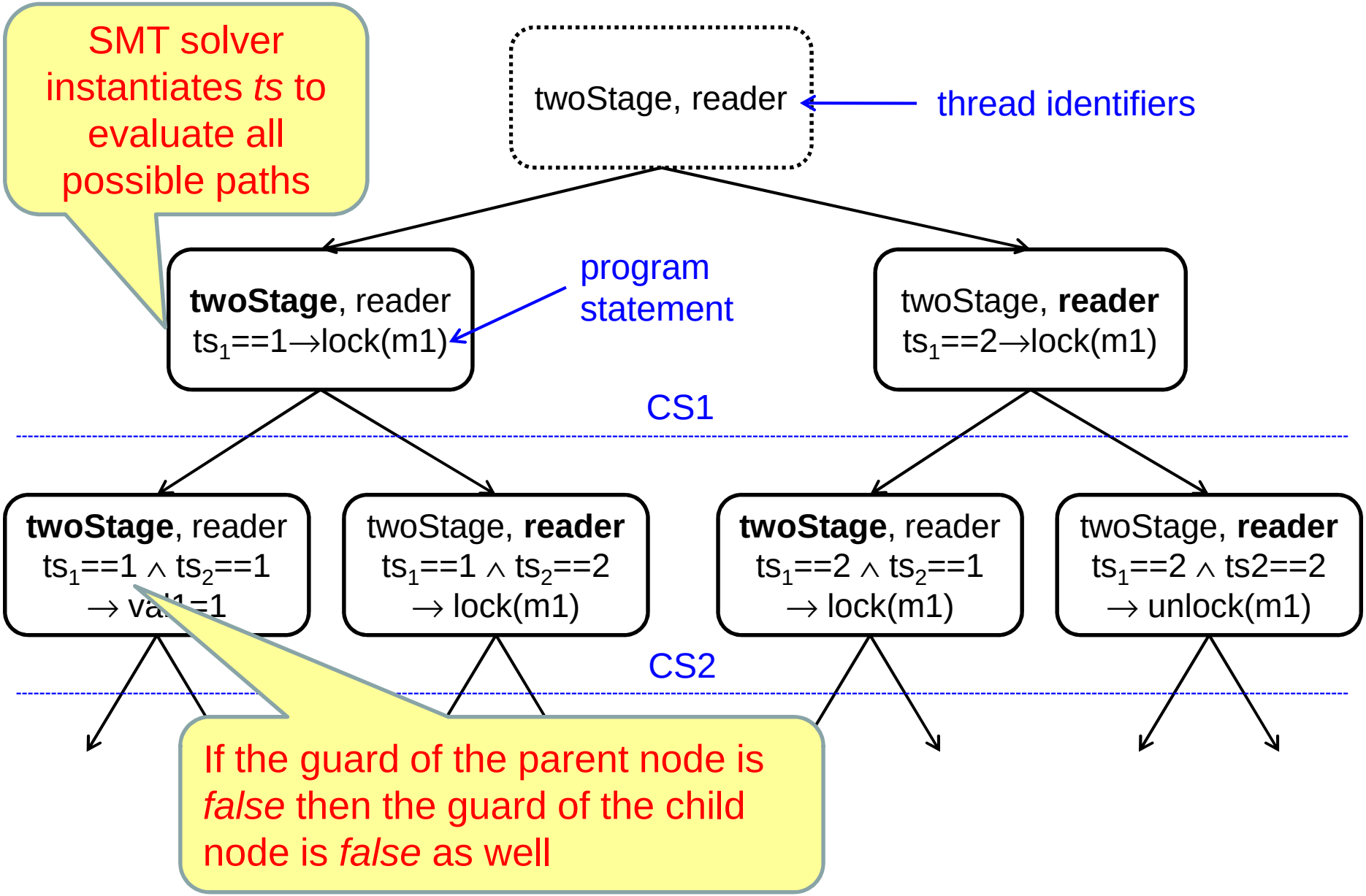
Idea: systematically encode all possible interleavings into one formula

- add a ***fresh variable*** *(ts)* for each context switch block *(i)* so that $0 < ts_i \leq$ number of threads
  - record in which order the *scheduler* has executed the program (*aka* **scheduler guards**)
  - SMT solver determines the order in which threads are simulated
- add scheduler guards only to ***effective statements*** (assignments and assertions)
  - record ***effective context switches (ECS)***
    - ▷ *context switches to an effective statement*
  - *ECS block*: sequence of program statements that are executed with no intervening ECS

Schedule Recording: Execution Paths



Schedule Recording: Execution Paths



Schedule Recording: Interleaving I_s

statements:

twoStage-ECS:

reader-ECS:

```
Thread twoStage
1: lock(m1);
2: val1 = 1;
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);
```

ECS block: sequence of program statements that are executed with no intervening ECS

```
10:  return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));
```

Schedule Recording: Interleaving I_s

statements: 1

twoStage-ECS: $ts_{1,1}$

reader-ECS:

guarded statement can only be executed if statement *1* is scheduled in the *ECS block 1*

```
Thread twoStage
1: lock(m1);          ts1 == 1
2: val1 = 1;
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);
```

```
Thread reader
7: lock(m1);
15: unlock(m2);
16: assert(t2==(t1+1));
```

each program statement is then prefixed by a *schedule guard* $ts_i = j$, where:

- *i* is the *ECS block number*
- *j* is the *thread identifier*

Schedule Recording: Interleaving I_s

statements: 1-2

twoStage-ECS: $ts_{1,1}$-$ts_{2,2}$

reader-ECS:

```
Thread twoStage
1: lock(m1);          ts1 == 1
2: val1 = 1;          ts2 == 1
3: unlock(m1);
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);
```

```
Thread reader
7:  lock(m1);
8:  if (val1 == 0) {
9:    unlock(m1);
10:  return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));
```

Schedule Recording: Interleaving I_s

statements: 1-2-3

twoStage-ECS: $ts_{1,1}$-$ts_{2,2}$-$ts_{3,3}$

reader-ECS:

```
Thread twoStage
1: lock(m1);          ts1 == 1
2: val1 = 1;          ts2 == 1
3: unlock(m1);        ts3 == 1
4: lock(m2);
5: val2 = val1 + 1;
6: unlock(m2);
```

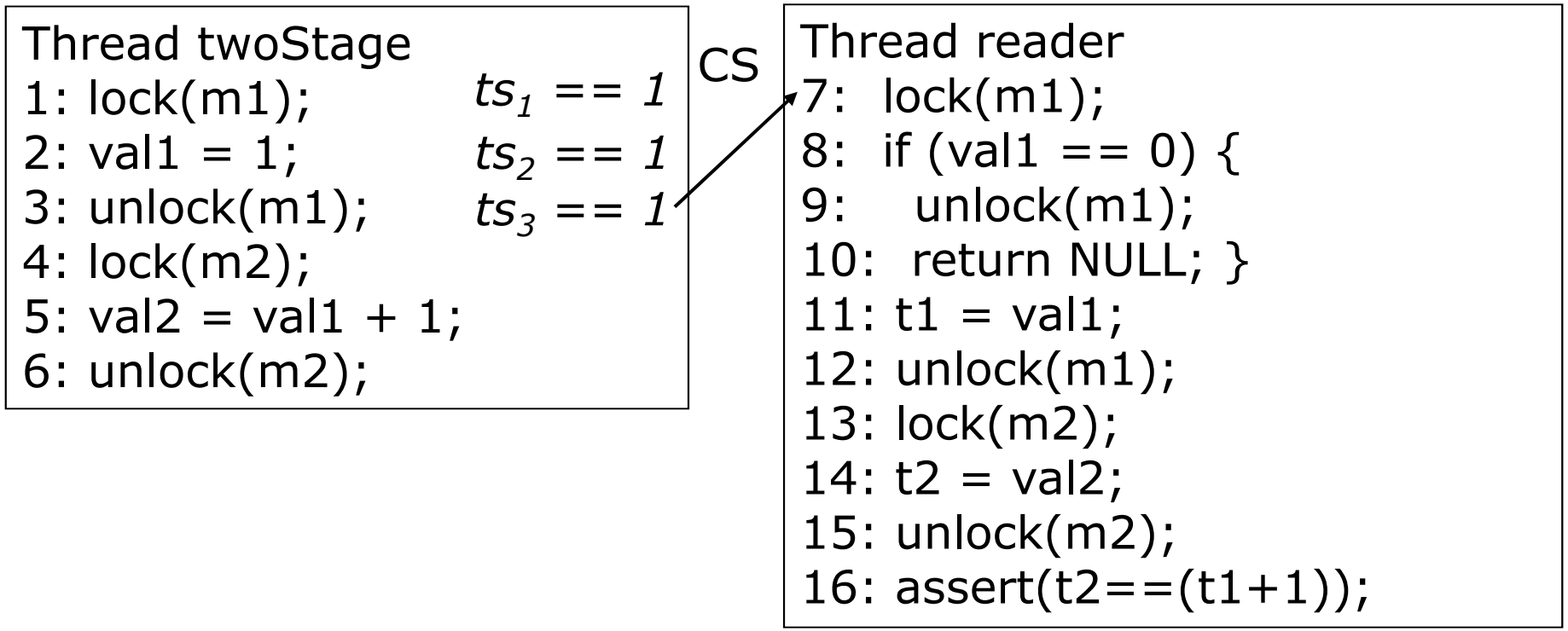
```
Thread reader
7:  lock(m1);
8:  if (val1 == 0) {
9:    unlock(m1);
10:  return NULL; }
11: t1 = val1;
12: unlock(m1);
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));
```

Schedule Recording: Interleaving I_s

statements: 1-2-3

twoStage-ECS: $ts_{1,1}$-$ts_{2,2}$-$ts_{3,3}$

reader-ECS:

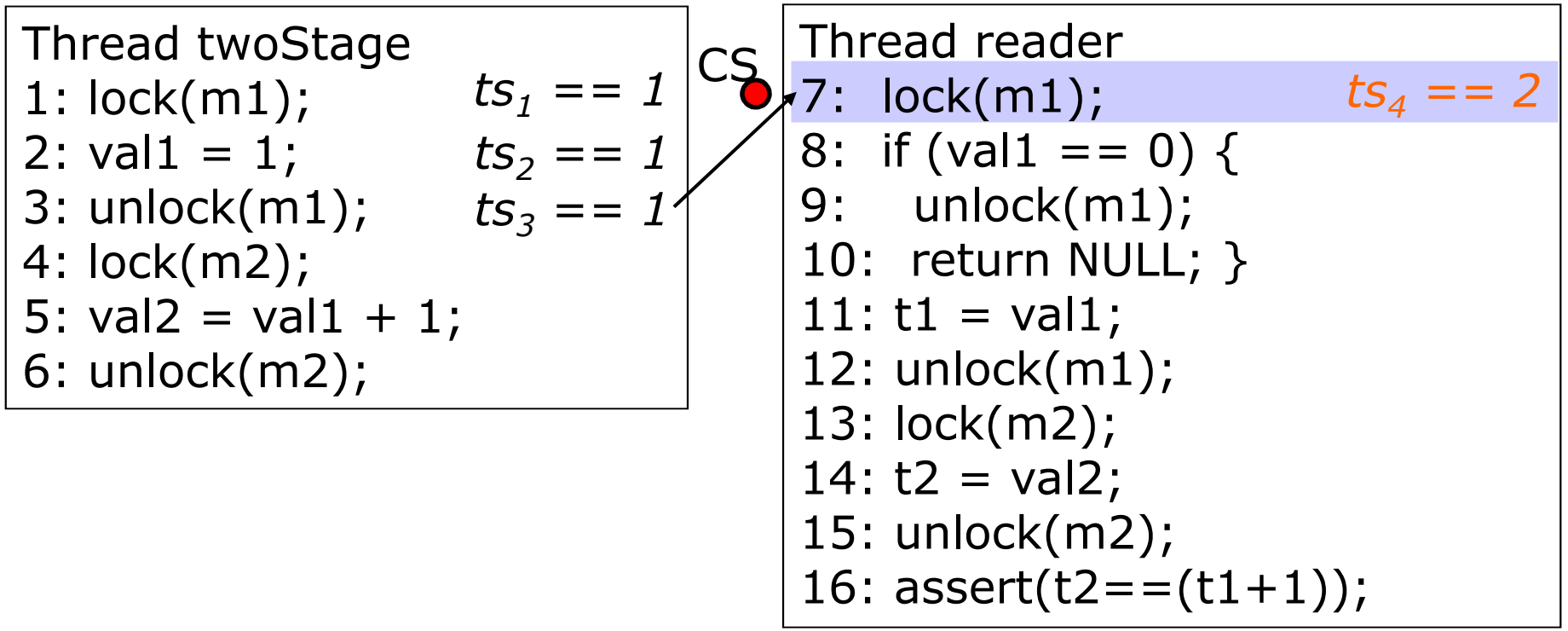


Schedule Recording: Interleaving I_s

statements: 1-2-3-7

twoStage-ECS: $ts_{1,1}$-$ts_{2,2}$-$ts_{3,3}$

reader-ECS: $ts_{7,4}$

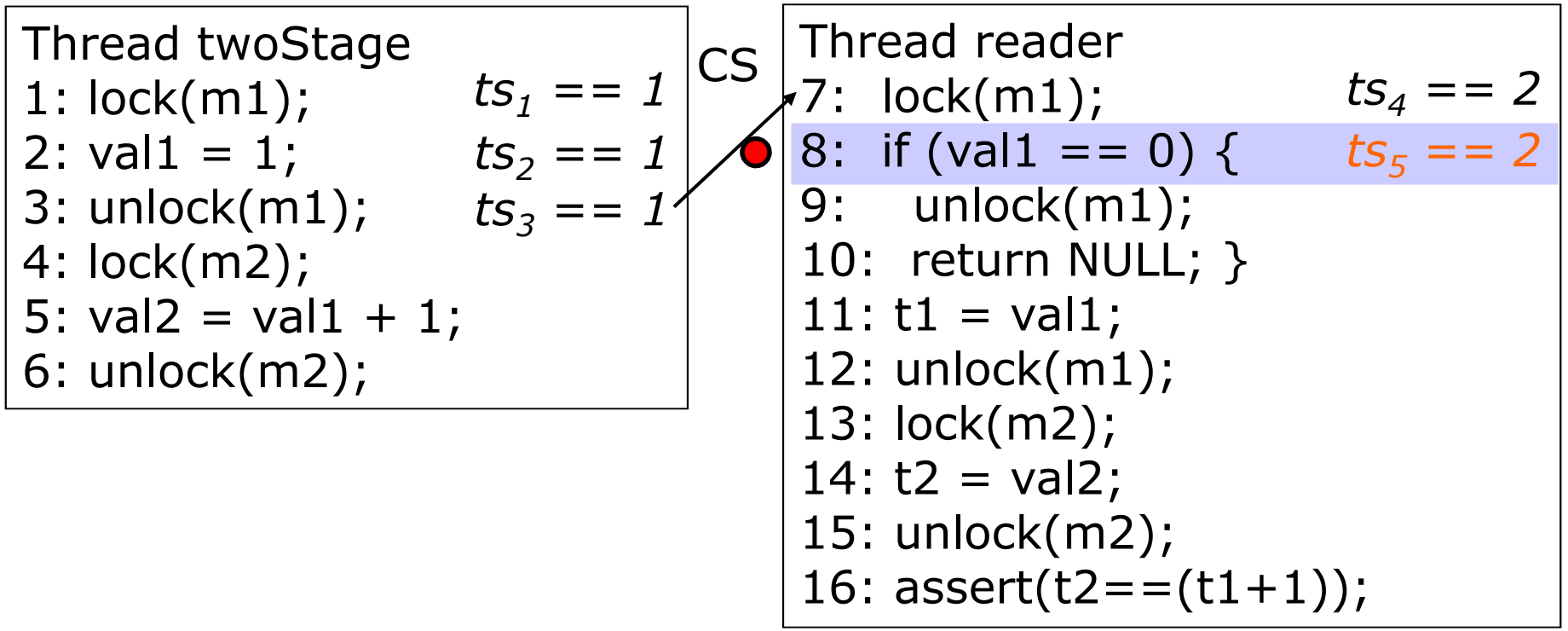


Schedule Recording: Interleaving I_s

statements: 1-2-3-7-8

twoStage-ECS: $ts_{1,1}$-$ts_{2,2}$-$ts_{3,3}$

reader-ECS: $ts_{7,4}$- $ts_{8,5}$

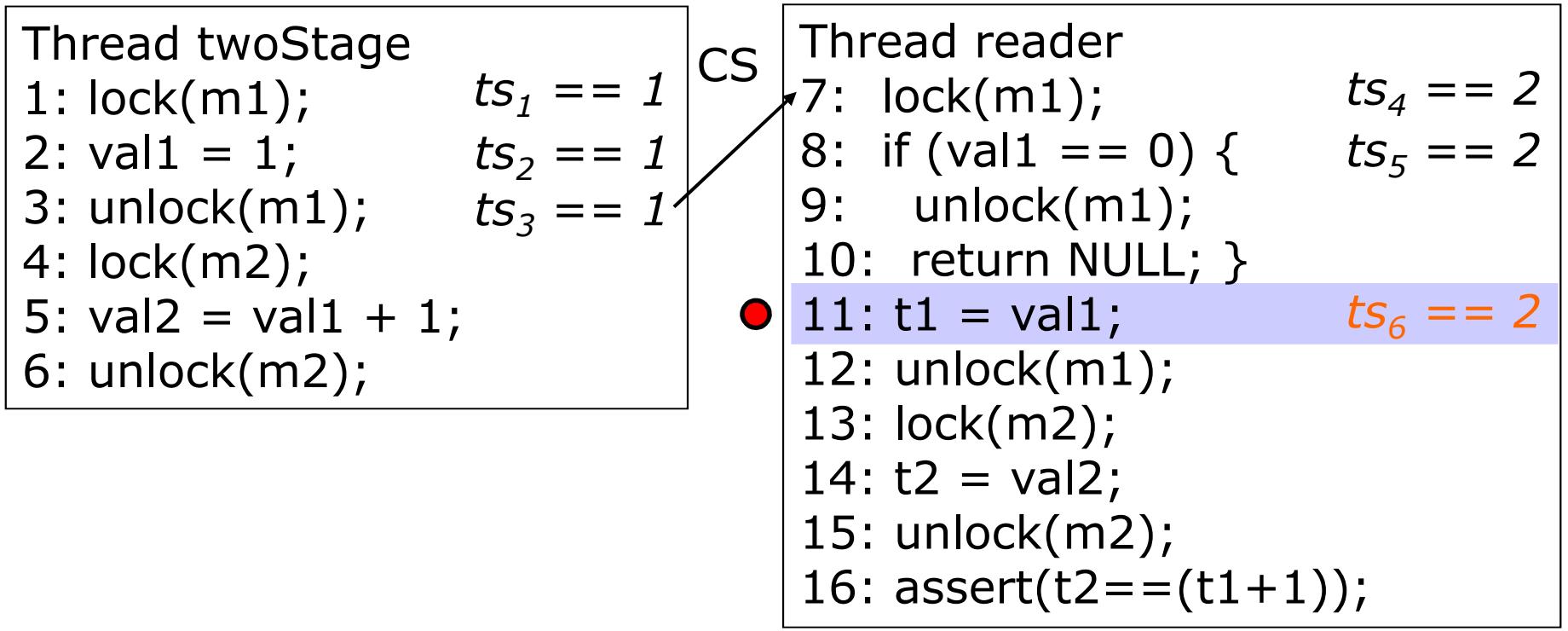


Schedule Recording: Interleaving I_s

statements: 1-2-3-7-8-11

twoStage-ECS: $ts_{1,1}$-$ts_{2,2}$-$ts_{3,3}$

reader-ECS: $ts_{7,4}$- $ts_{8,5}$-$ts_{11,6}$

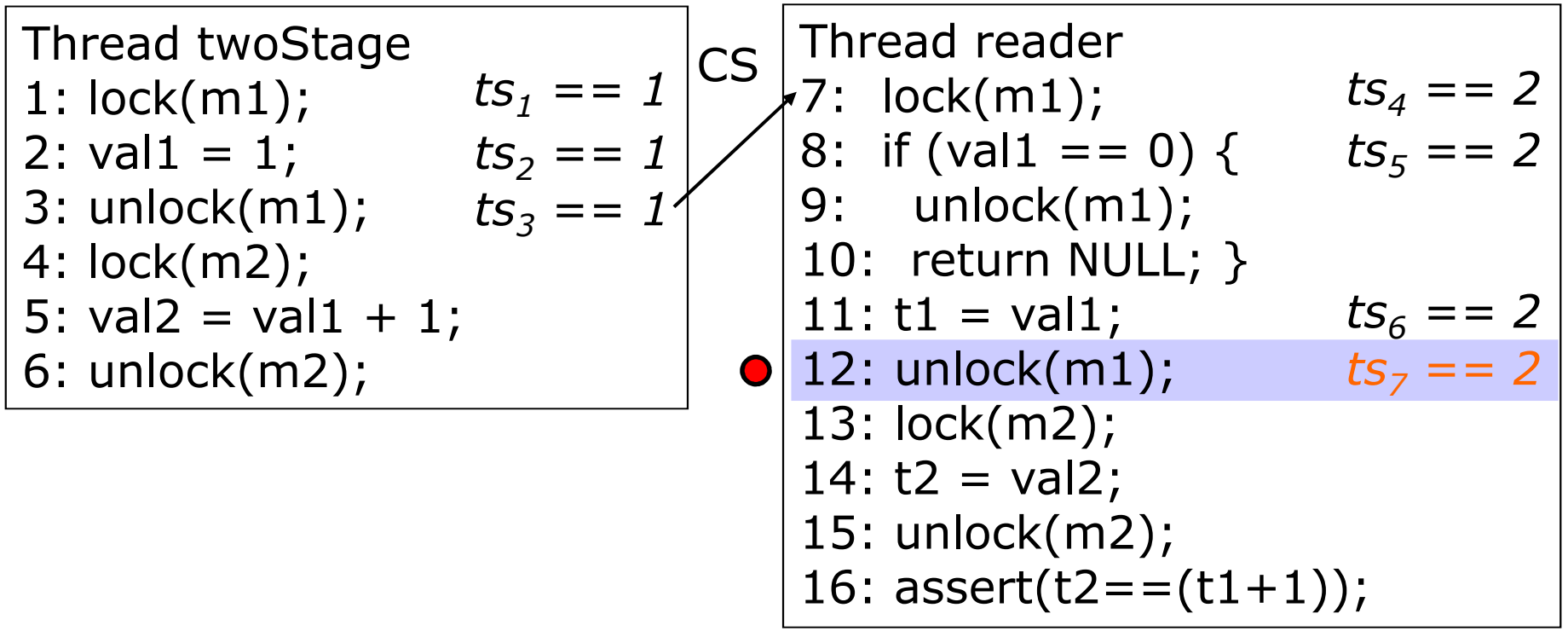


Schedule Recording: Interleaving I_s

statements: 1-2-3-7-8-11-12

twoStage-ECS: $ts_{1,1}$-$ts_{2,2}$-$ts_{3,3}$

reader-ECS: $ts_{7,4}$- $ts_{8,5}$-$ts_{11,6}$-$ts_{12,7}$

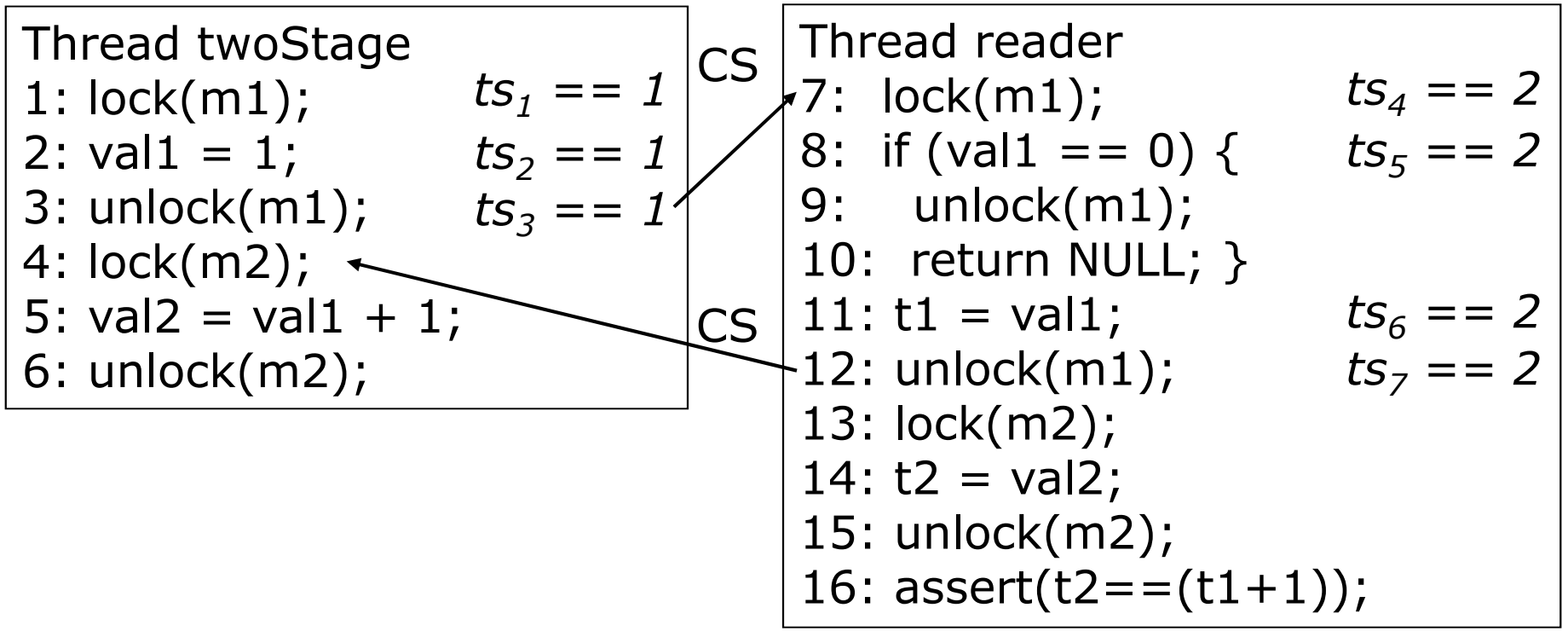


Schedule Recording: Interleaving I_s

statements: 1-2-3-7-8-11-12

twoStage-ECS: $ts_{1,1}$-$ts_{2,2}$-$ts_{3,3}$

reader-ECS: $ts_{7,4}$- $ts_{8,5}$-$ts_{11,6}$-$ts_{12,7}$

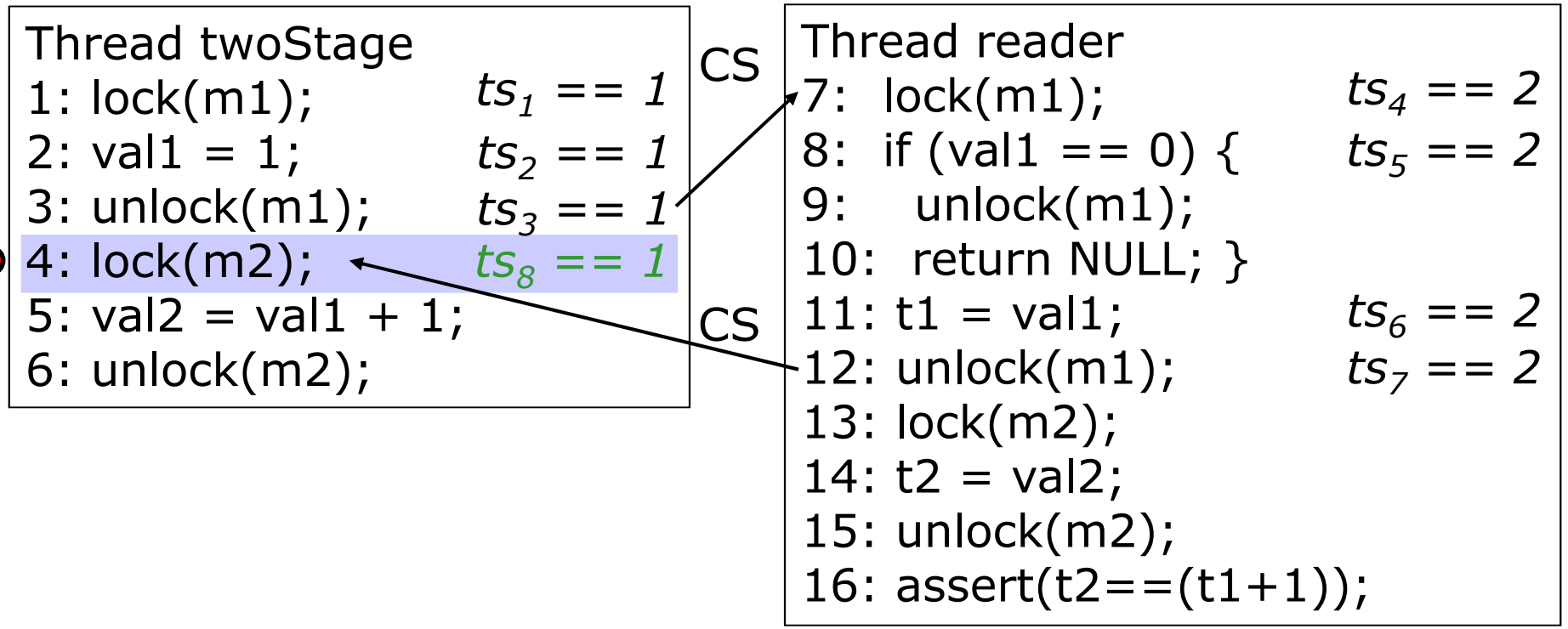


Schedule Recording: Interleaving I_s

statements: 1-2-3-7-8-11-12-4

twoStage-ECS: $ts_{1,1}$-$ts_{2,2}$-$ts_{3,3}$-$ts_{4,8}$

reader-ECS: $ts_{7,4}$- $ts_{8,5}$-$ts_{11,6}$-$ts_{12,7}$

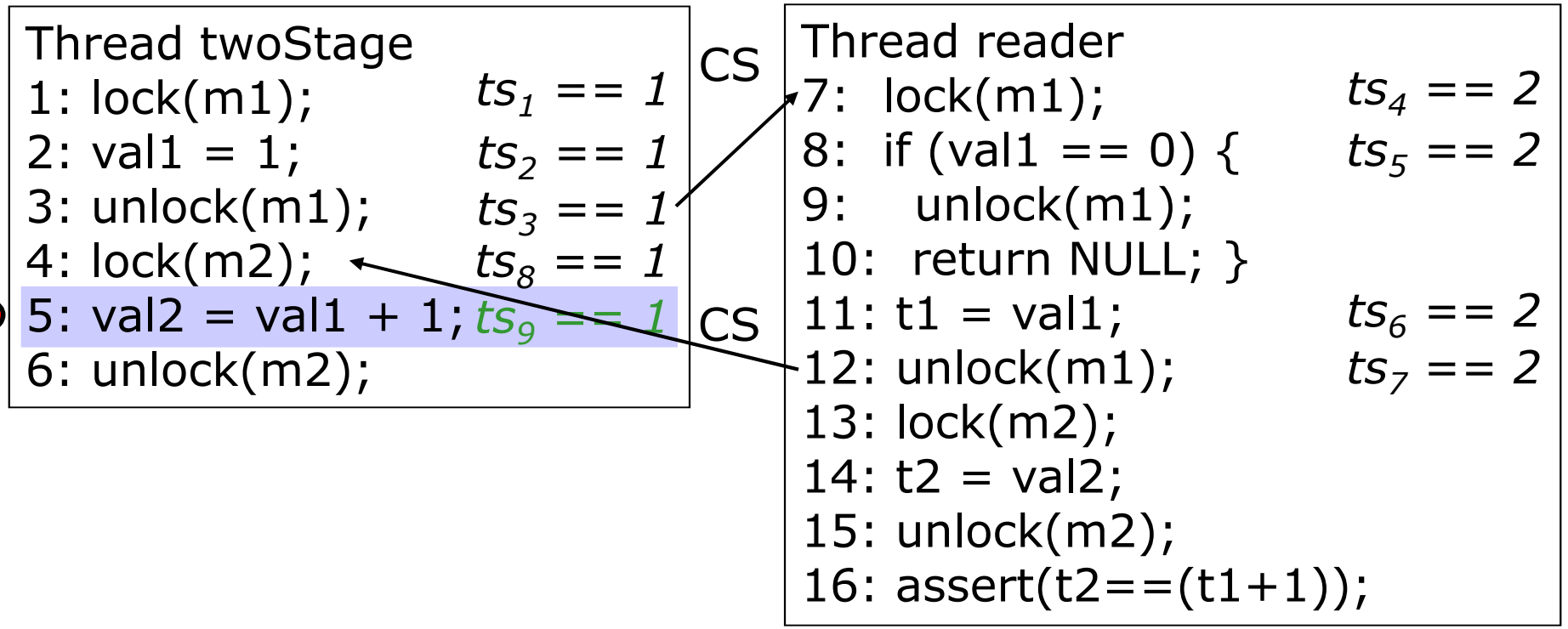


Schedule Recording: Interleaving I_s

statements: 1-2-3-7-8-11-12-4-5

twoStage-ECS: $ts_{1,1}$-$ts_{2,2}$-$ts_{3,3}$-$ts_{4,8}$-$ts_{5,9}$

reader-ECS: $ts_{7,4}$- $ts_{8,5}$-$ts_{11,6}$-$ts_{12,7}$

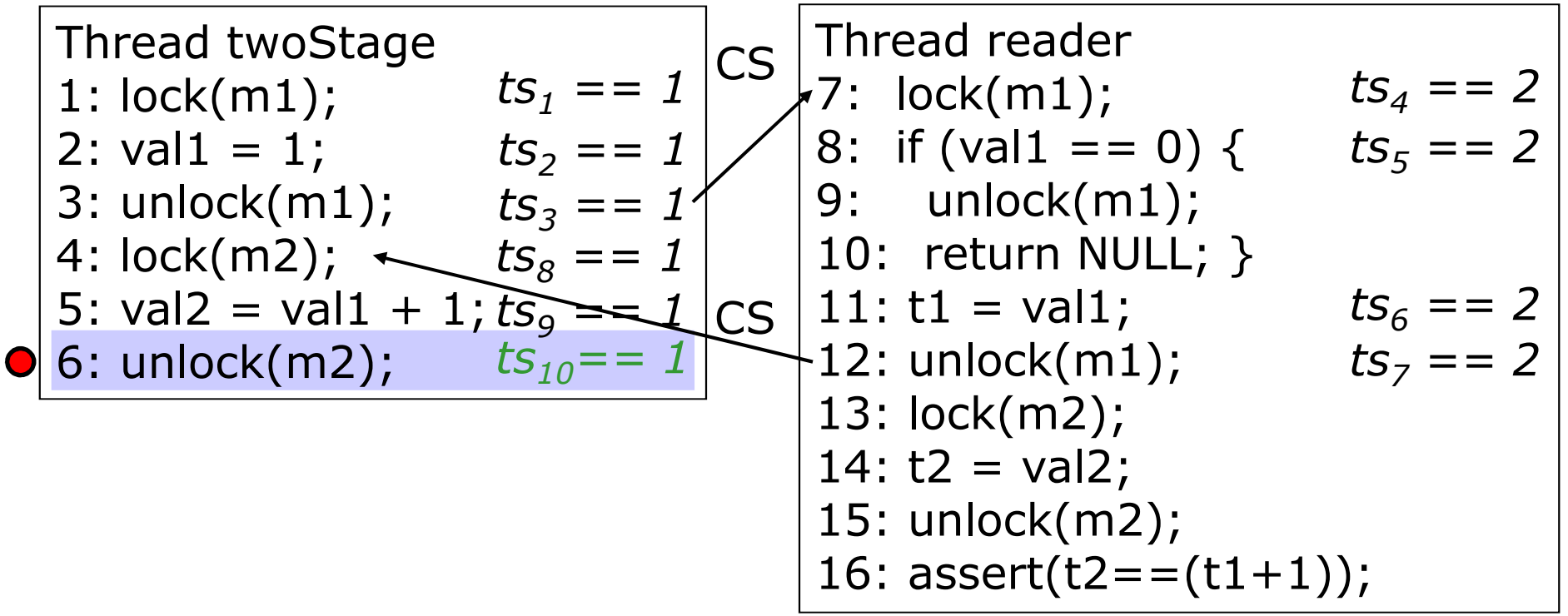


Schedule Recording: Interleaving I_s

statements: 1-2-3-7-8-11-12-4-5-6

twoStage-ECS: $ts_{1,1}$-$ts_{2,2}$-$ts_{3,3}$-$ts_{4,8}$-$ts_{5,9}$-$ts_{6,10}$

reader-ECS: $ts_{7,4}$- $ts_{8,5}$-$ts_{11,6}$-$ts_{12,7}$

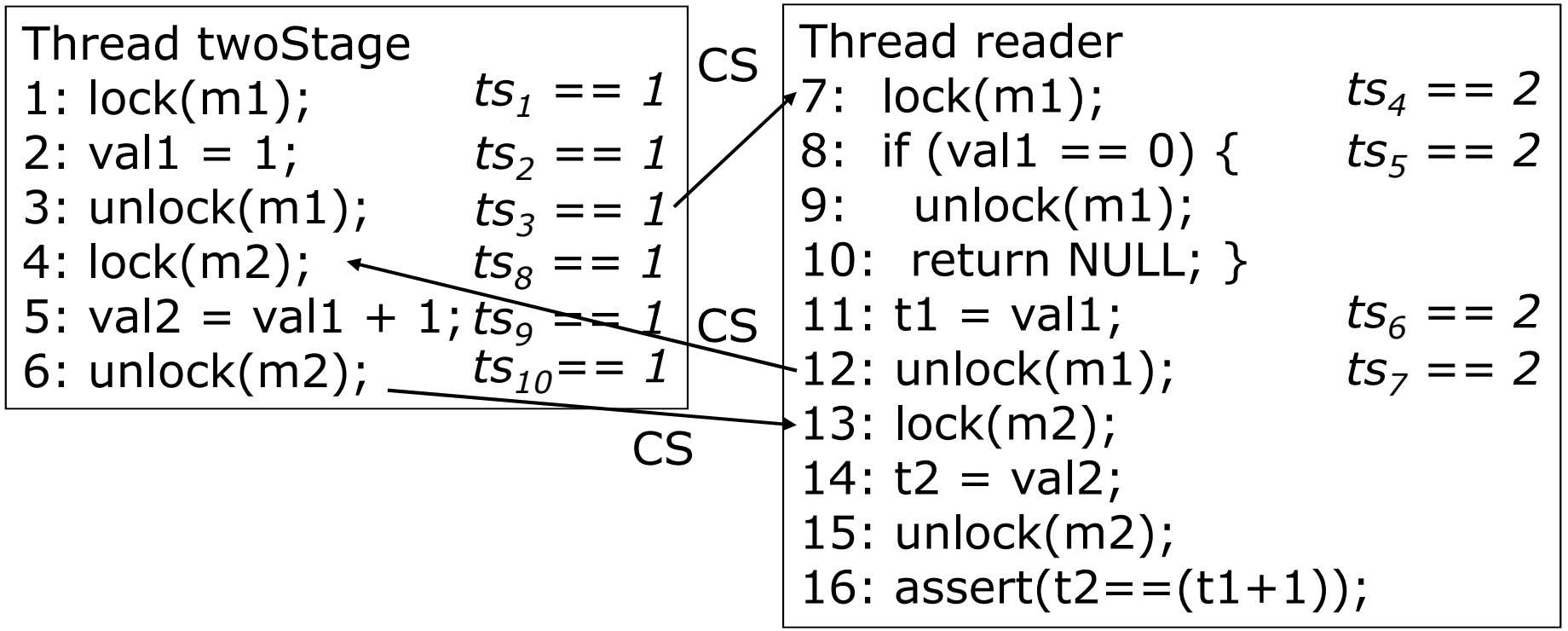


Schedule Recording: Interleaving I_s

statements: 1-2-3-7-8-11-12-4-5-6

twoStage-ECS: $ts_{1,1}$-$ts_{2,2}$-$ts_{3,3}$-$ts_{4,8}$-$ts_{5,9}$-$ts_{6,10}$

reader-ECS: $ts_{7,4}$- $ts_{8,5}$-$ts_{11,6}$-$ts_{12,7}$

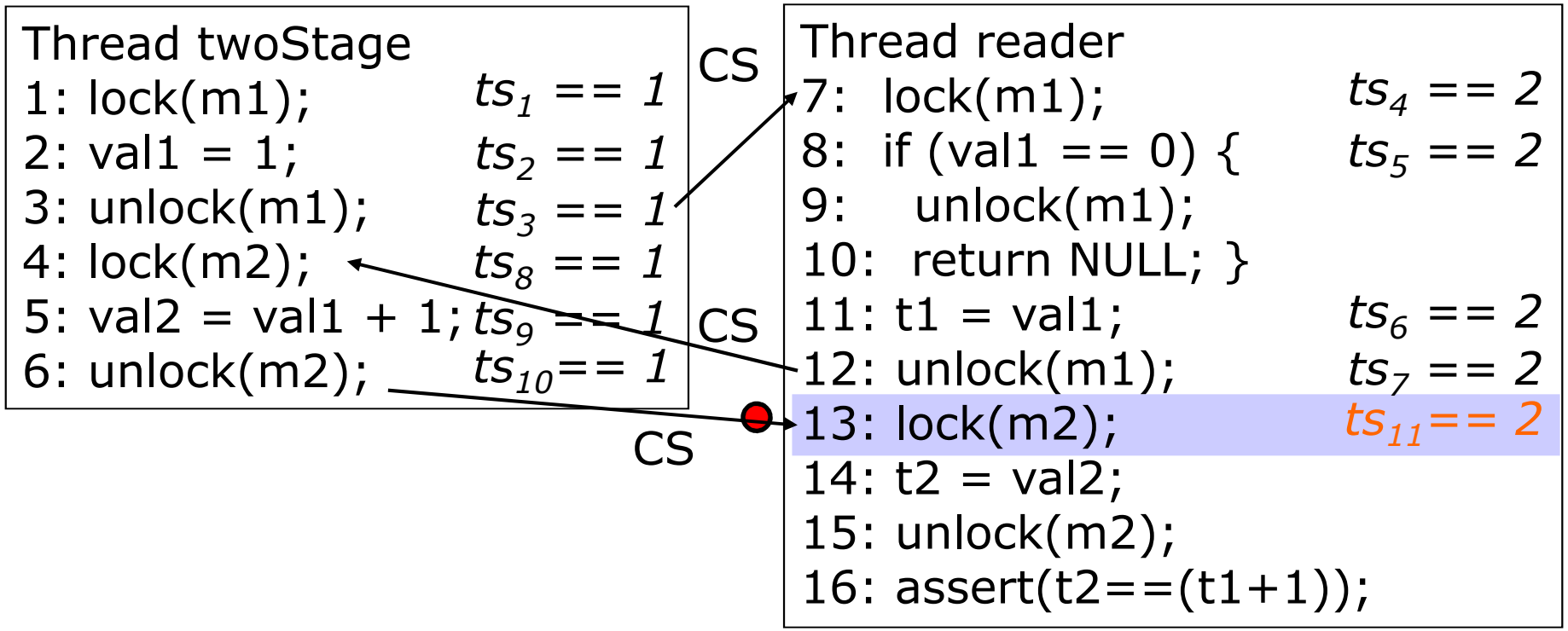


Schedule Recording: Interleaving I_s

statements: 1-2-3-7-8-11-12-4-5-6-13

twoStage-ECS: $ts_{1,1}$-$ts_{2,2}$-$ts_{3,3}$-$ts_{4,8}$-$ts_{5,9}$-$ts_{6,10}$

reader-ECS: $ts_{7,4}$- $ts_{8,5}$-$ts_{11,6}$-$ts_{12,7}$-$ts_{13,11}$

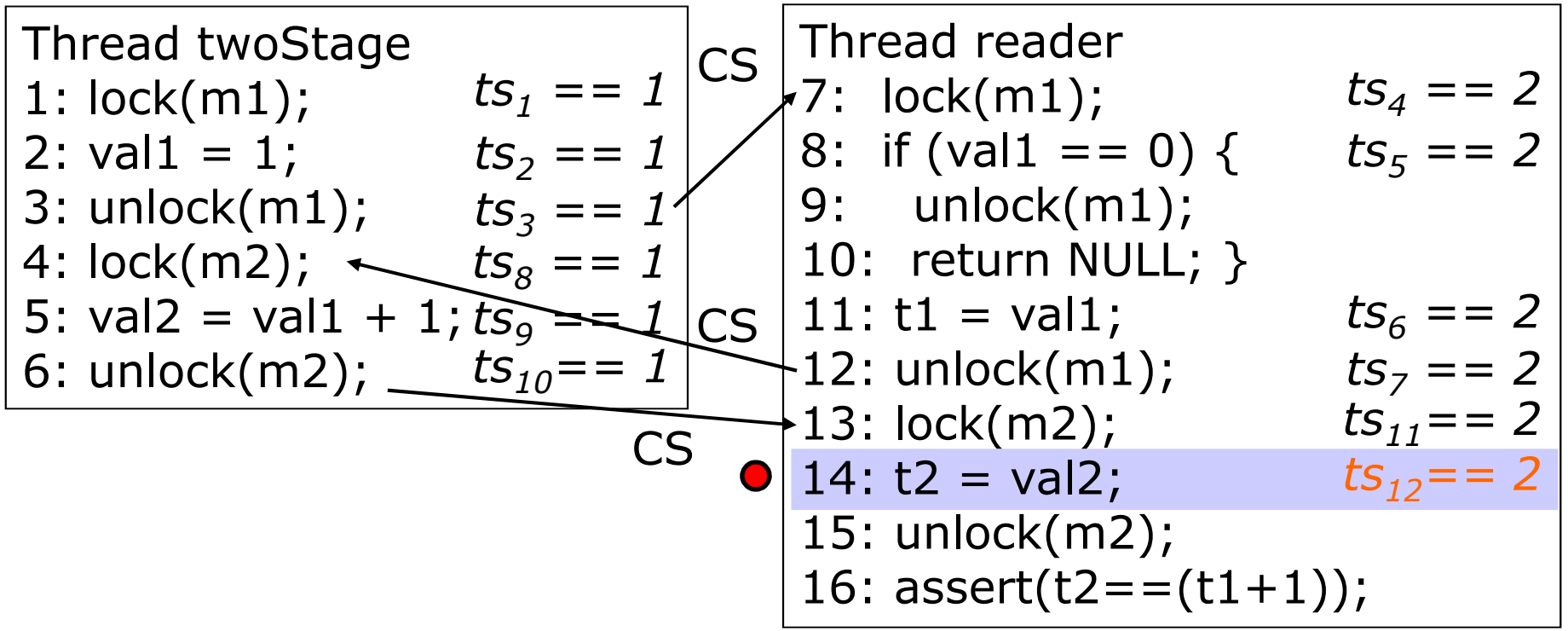


Schedule Recording: Interleaving I_s

statements: 1-2-3-7-8-11-12-4-5-6-13-14

twoStage-ECS: $ts_{1,1}$-$ts_{2,2}$-$ts_{3,3}$-$ts_{4,8}$-$ts_{5,9}$-$ts_{6,10}$

reader-ECS: $ts_{7,4}$- $ts_{8,5}$-$ts_{11,6}$-$ts_{12,7}$-$ts_{13,11}$-$ts_{14,12}$

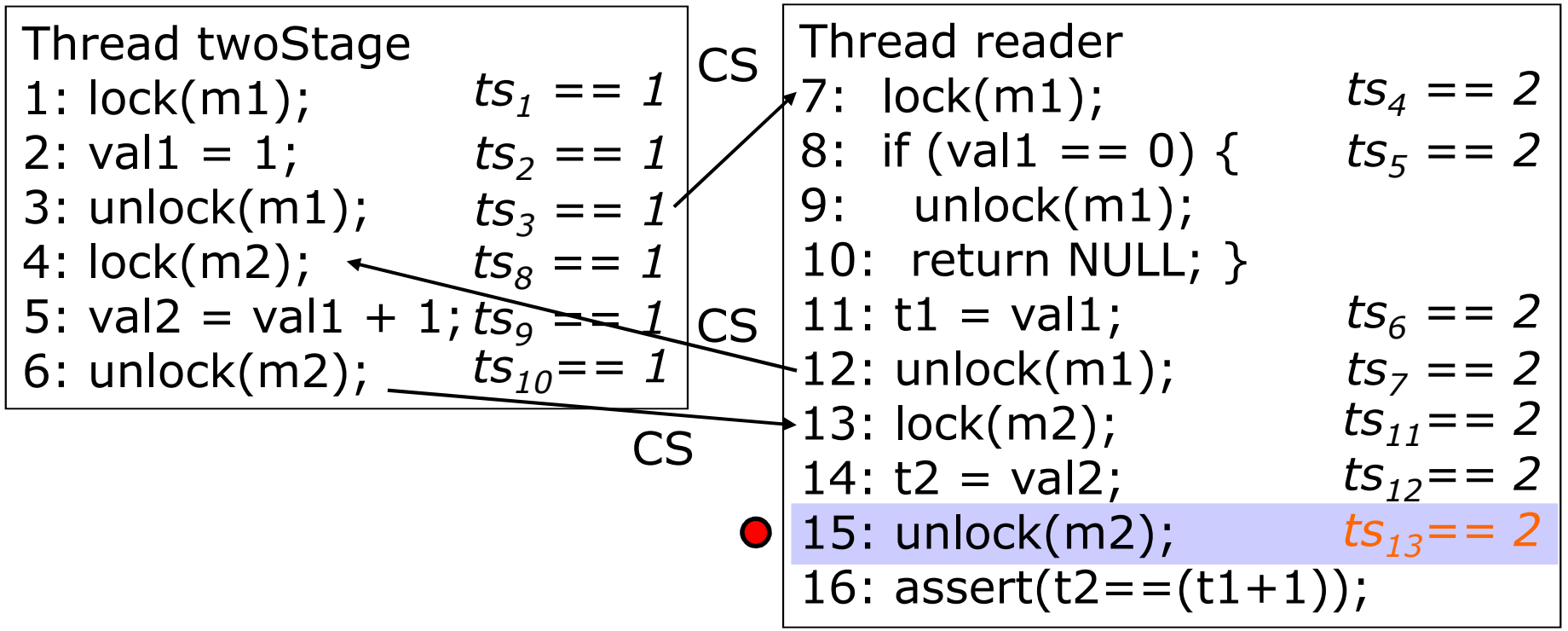


Schedule Recording: Interleaving I_s

statements: 1-2-3-7-8-11-12-4-5-6-13-14-15

twoStage-ECS: $ts_{1,1}$-$ts_{2,2}$-$ts_{3,3}$-$ts_{4,8}$-$ts_{5,9}$-$ts_{6,10}$

reader-ECS: $ts_{7,4}$- $ts_{8,5}$-$ts_{11,6}$-$ts_{12,7}$-$ts_{13,11}$-$ts_{14,12}$-$ts_{15,13}$

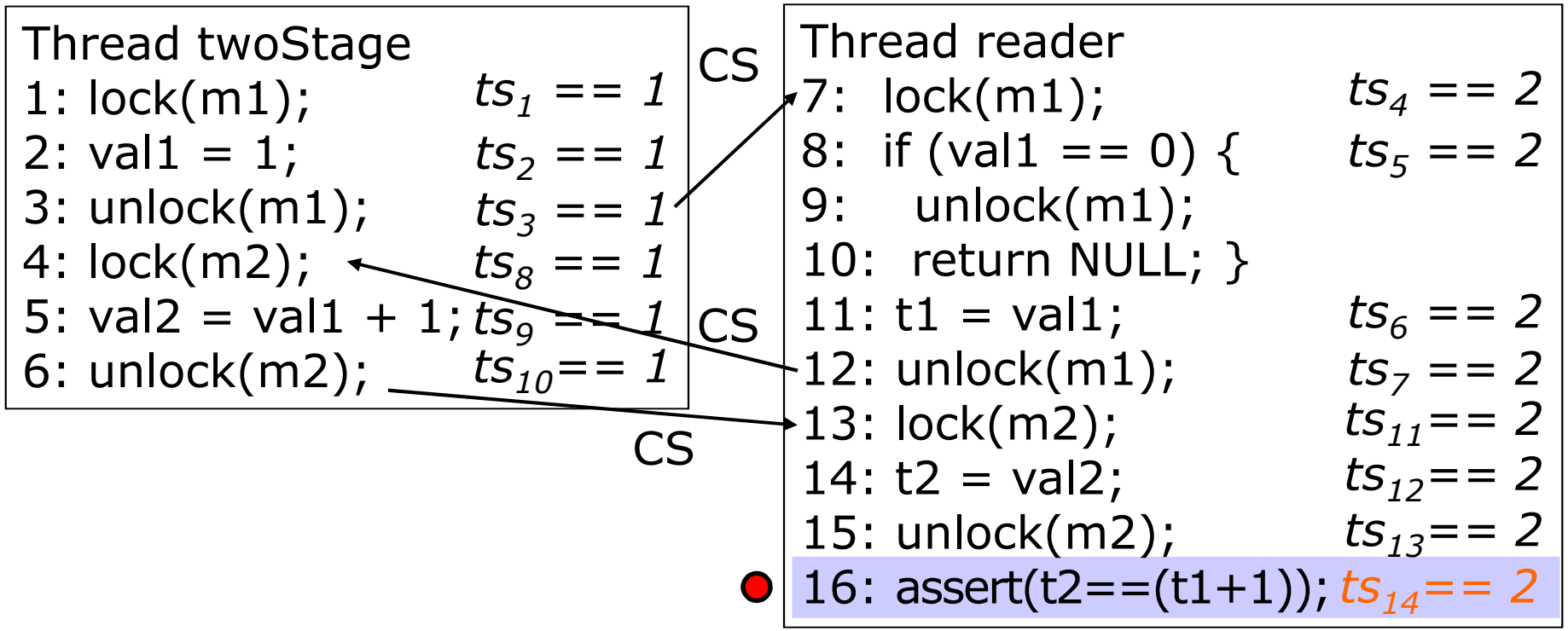


Schedule Recording: Interleaving I_s

statements: 1-2-3-7-8-11-12-4-5-6-13-14-15-16

twoStage-ECS: $ts_{1,1}$-$ts_{2,2}$-$ts_{3,3}$-$ts_{4,8}$-$ts_{5,9}$-$ts_{6,10}$

reader-ECS: $ts_{7,4}$- $ts_{8,5}$-$ts_{11,6}$-$ts_{12,7}$-$ts_{13,11}$-$ts_{14,12}$-$ts_{15,13}$ -$ts_{16,14}$

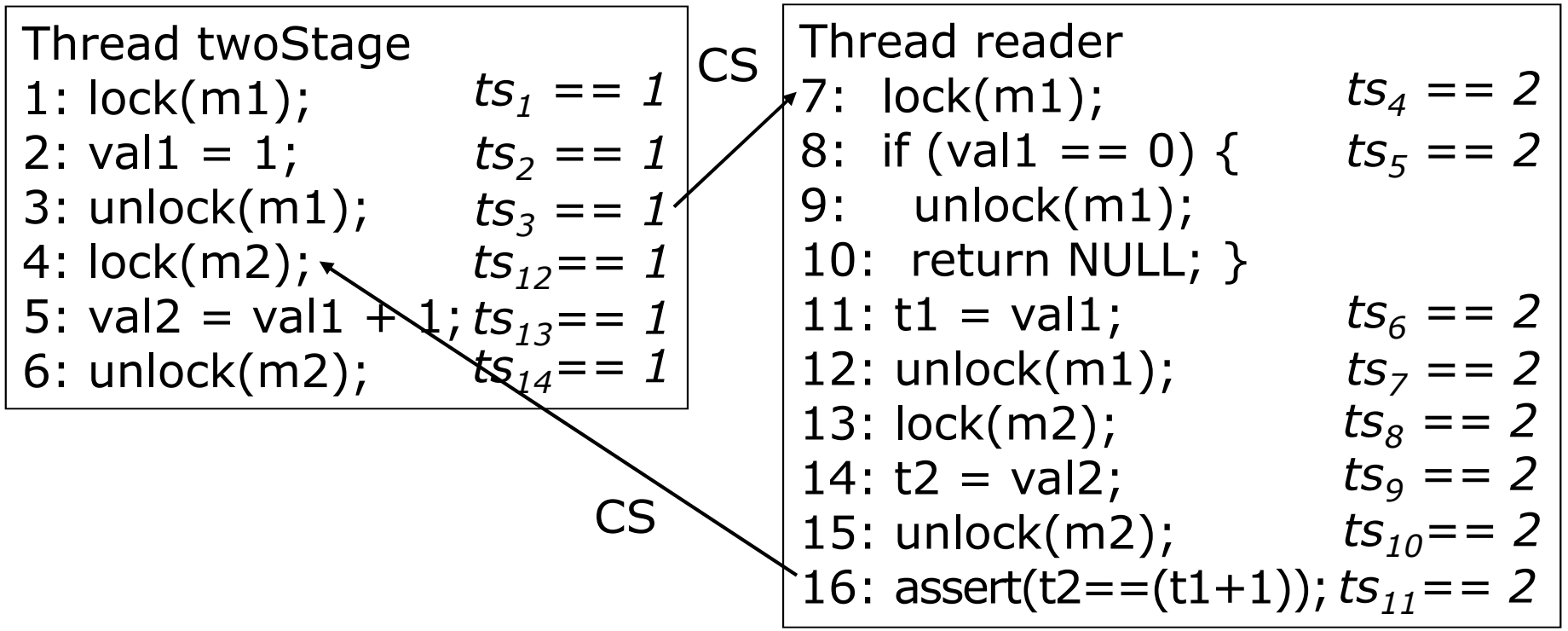


Schedule Recording: Interleaving I_f

statements: 1-2-3-7-8-11-12-13-14-15-16-4-5-6

twoStage-ECS: $ts_{1,1}$-$ts_{2,3}$-$ts_{3,4}$-$ts_{4,12}$-$ts_{5,13}$-$ts_{6,14}$

reader-ECS: $ts_{7,4}$ -$ts_{8,5}$ -$ts_{11,6}$-$ts_{12,7}$-$ts_{13,8}$-$ts_{14,9}$-$ts_{15,10}$-$ts_{16,11}$



Observations about the schedule recoding approach

- we systematically explore the thread interleavings as before, but now:
  - add schedule guards to record in which order the scheduler has executed the program
  - encode all execution paths into one formula
    - ▹ *bound the number of preemptions*
    - ▹ *exploit which transitions are enabled in a given state*
- the number of threads and context switches can grow very large quickly, and easily “blow-up” the solver:
  - there is a clear trade-off between usage of time and memory resources

Under-approximation and Widening

Idea: check models with an increased set of allowed interleavings [Grumberg&et al.'05]

- start from a single interleaving (under-approximation) and widen the model by adding more interleavings incrementally
- main steps of the algorithm:

1. encode control literals ($cl_{i,j}$) into the verification condition ψ
   - ▷ $cl_{i,j}$ where *i* is the ECS block number and *j* is the thread identifier
2. check the satisfiability of ψ (stop if ψ is satisfiable)
3. extract proof objects generated by the SMT solver
4. check whether the proof depends on the control literals (stop if the proof does not depend on the control literals)
5. remove literals that participated in the proof and go to step 2

UW Approach: Running Example

- use the same guards as in the schedule recording approach as control literals
  - but here the schedule is updated based on the information extracted from the proof

Thread twoStage	
1: lock(m1);	$cl_{1,twoStage} \rightarrow ts_1 == 1$
2: val1 = 1;	$cl_{2,twoStage} \rightarrow ts_2 == 1$
3: unlock(m1);	$cl_{3,twoStage} \rightarrow ts_3 == 1$
4: lock(m2);	$cl_{8,twoStage} \rightarrow ts_8 == 1$
5: val2 = val1 + 1;	$cl_{9,twoStage} \rightarrow ts_9 == 1$
6: unlock(m2);	$cl_{10,twoStage} \rightarrow ts_{10} == 1$

- reduce the number of control points from *m* x *n* to *e* x *n*
  - *m* is the number of program statements; *n* is the number of threads, and *e* is the number of ECS blocks

Evaluation

Comparison of the Approaches

- Goal: compare efficiency of the proposed approaches
  - lazy exploration
  - schedule recording
  - underapproximation and widening
- Set-up:
  - ESBMC v1.15.1 together with the SMT solver Z3 v2.11
  - support the logics *QF_AUFBV* and *QF_AUFLIRA*
  - standard desktop PC, time-out 3600 seconds

About the benchmarks

	Module	#L	#T	#P	B	#C	
		81	26	47	26	3	Fra
2	fsbench_bad		27		27	2	Fi… …dani file system with array
3	inc…				29	4	…nto a hash table concurrently
4	aget-0.4_bad	1233					…eaded download …tor
5	bzip2smp_ok	6366					…mpressor
6	reorder_bad	84	10	7	10	11	Contains a data race
7	twostage_bad	128	100	13	100	4	Contains an atomicity violation
8	wronglock_bad	110	8	8	8	8	Contains wrong lock acquisition ordering
9	exStbHDMI_ok	1060	2	24	16	20	Configures the HDMI device
10	exStbLED_ok	425	2	45	10	10	Front panel LED display
11	exStbThumb_bad	1109	2	249	2	1	Demonstrate how thumbnail images can be manipulated
12	micro_10_ok	1171	10	10	1	17	synthetic micro-benchmark

lines of code

number of properties checked

the number of BMC unrolling steps

number of threads

number of context switches

About the benchmarks

Inspect benchmark suite

	Module	#				C	Description
1	**fsbench_ok**					**2**	**Frangipani file system**
2	**fsbench_bad**	**86**		**48**	**27**	**2**	**Frangipani file system with array out of bounds**
3	**indexer_ok**	**77**	**13**	**21**	**129**	**4**	**Insert messages into a hash table concurrently**
4	**aget-0.4_bad**	**1233**	**3**	**279**	**200**	**2**	**Multi-threaded download accelerator**
5	**bzip2smp_ok**	**6366**	**3**	**8568**	**1**	**9**	**Data compressor**
6	reorder_bad	84	10	7	10	11	Contains a data race
7	twostage_bad	128	100	13	100	4	Contains an atomicity violation
8	wronglock_bad	110	8	8	8	8	Contains wrong lock acquisition ordering
9	exStbHDMI_ok	1060	2	24	16	20	Configures the HDMI device
10	exStbLED_ok	425	2	45	10	10	Front panel LED display
11	exStbThumb_bad	1109	2	249	2	1	Demonstrate how thumbnail images can be manipulated
12	micro_10_ok	1171	10	10	1	17	synthetic micro-benchmark

About the benchmarks

	Module	#L	#T	#P	B	#C	Description
1	fsbench_ok	81	26	47	26	2	Frangipani file system
2	fsbench_bad	80	27	48	27	2	Frangipani file system with array out of bounds
3	indexer_ok					4	Insert messages into a hash table concurrently
4	aget-0.4_bad	1				2	Multi-threaded download accelerator
5	bzip2smp_ok	6	3	8568	1	9	Data compressor
6	**reorder_bad**	**84**	**10**	**7**	**10**	**11**	**Contains a data race**
7	**twostage_bad**	**128**	**100**	**13**	**100**	**4**	**Contains an atomicity violation**
8	**wronglock_bad**	**110**	**8**	**8**	**8**	**8**	**Contains wrong lock acquisition ordering**
9	exStbHDMI_ok	1060	2	24	16	20	Configures the HDMI device
10	exStbLED_ok	425	2	45	10	10	Front panel LED display
11	exStbThumb_bad	1109	2	249	2	1	Demonstrate how thumbnail images can be manipulated
12	micro_10_ok	1171	10	10	1	17	synthetic micro-benchmark

VV-lab benchmark suite

About the benchmarks

	Module	#L	#T	#P	B	#C	Description
1	fsbench_ok	81	26	47	26	2	Frangipani file system
2	fsbench_bad	80	27	48	27	2	Frangipani file system with array out of bounds
3	indexer_ok	77	13	21	129	4	Insert messages into a hash table concurrently
4	aget-0.4_bad	1233	3	279	200	2	Multi-threaded download accelerator
5	bzip2smp_ok						ata compressor
6	reorder_bad						ontains a data race
7	twostage_bad						ontains an atomicity violation
8	wronglock_bad						ontains wrong lock acquisition ordering
9	**exStbHDMI_ok**		**2**	**24**	**16**	**20**	**Configures the HDMI device**
10	**exStbLED_ok**	**425**	**2**	**45**	**10**	**10**	**Front panel LED display**
11	**exStbThumb_bad**	**1109**	**2**	**249**	**2**	**1**	**Demonstrate how thumbnail images can be manipulated**
12	micro_10_ok	1171	10	10	1	17	synthetic micro-benchmark

Set-top box applications from NXP semiconductors

About the benchmarks

	Module	#L	#T	#P	B	#C	Description
1	fsbench_ok	81	26	47	26	2	Frangipani file system
2	fsbench_bad	80	27	48	27	2	Frangipani file system with array out of bounds
3	indexer_ok	77	13	21	129	4	Insert messages into a hash table concurrently
4	aget-0.4_bad	1233	3	279	200	2	Multi-threaded download accelerator
5	bzip2smp_ok	6366	3	8568	1	9	Data compressor
6	reorder_bad	84	10	7	10	11	Contains a data race
7	twostage_bad						n atomicity violation
8	wronglock_bad						rong lock acquisition
9	exStbHDMI_ok						the HDMI device
10	exStbLED_ok						l LED display
11	exStbThumb_bad	11	2	249	2	1	Demonstrate how thumbnail images can be manipulated
12	**micro_10_ok**	**1171**	**10**	**10**	**1**	**17**	**synthetic micro-benchmark**

It is used to check the scalability of multi-threaded software verification tools [Ghafari 2010]

Comparison of the appro

encoding and solver time

number of generated and failed interleavings

error detected in module "+" **GOOD THING**

error occurred in tool "–" **BAD THING**

number of iterations

Module	Lazy					M		
	Time	Result	#FI/#I	Time	Result	Time	Result	Iter
fsbench_ok				304	+	301		1
fsbench_ba					+			2
indexer_ok					+			1
aget-0.4_ba				7	+	125	+	1
bzip2smp_ok	1800	+	0/1294	MO	-	MO	-	1
reorder_bad	<1	+	1/154574	MO	-	MO	-	1
twostage_bad	88	+	1/139	93	+	195	+	5
wronglock_bad	90	+	6/104015	MO	-	MO	-	1
exStbHDMI_ok	229	+	0/1	226	+	213	+	1
exStbLED_ok	73	+	0/11	73	+	787	+	1
exStbThumb_bad	95	+	3/3	14	+	12	+	1
micro_10_ok	254	+	0/29260	MO	-	MO	-	1

Comparison of the approaches (1)

lazy encoding often more efficient than schedule recording and UW

	Lazy			Schedule		UW		
	Time	Result	#FI/#I	Time	Result	Time	Result	Iter
fsbench_ok	**282**	+	0/676	**304**	+	**301**	+	1
fsbench_bad	**<1**	+	729/729	**360**	+	**786**	+	2
indexer_ok	595	+	0/17160	220	+	218	+	1
aget-0.4_bad	137	+	1/1	127	+	125	+	1
bzip2smp_ok	**1800**	+	0/1294	**MO**	-	**MO**	-	1
reorder_bad	**<1**	+	1/154574	**MO**	-	**MO**	-	1
twostage_bad	**88**	+	1/139	**93**	+	**195**	+	5
wronglock_bad	**90**	+	6/104015	**MO**	-	**MO**	-	1
exStbHDMI_ok	229	+	0/1	226	+	213	+	1
exStbLED_ok	73	+	0/11	73	+	787	+	1
exStbThumb_bad	95	+	3/3	14	+	12	+	1
micro_10_ok	**254**	+	0/29260	**MO**	-	**MO**	-	1

Comparison of the approaches (2)

lazy encoding often more efficient than schedule recording and UW, ***but not always***

			azy		Schedule		UW		
		sult	#FI/#I	Time	Result	Time	Result	Iter
		+	0/676	304	+	301	+	1
fsbench_	<1	+	729/729	360	+	786	+	2
indexer_ok	**595**	+	0/17160	**220**	+	**218**	+	1
aget-0.4_bad	**137**	+	1/1	**127**	+	**125**	+	1
bzip2smp_ok	1800	+	0/1294	MO	-	MO	-	1
reorder_bad	<1	+	1/154574	MO	-	MO	-	1
twostage_bad	88	+	1/139	93	+	195	+	5
wronglock_bad	90	+	6/104015	MO	-	MO	-	1
exStbHDMI_ok	**229**	+	0/1	**226**	+	**213**	+	1
exStbLED_ok	**73**	+	0/11	**73**	+	**787**	+	1
exStbThumb_bad	**95**	+	3/3	**14**	+	**12**	+	1
micro_10_ok	254	+	0/29260	MO	-	MO	-	1

Comparison of the approaches (3)

lazy encoding is extremely fast for ***satisfiable instances***

Module	Lazy			Schedule		UW		
	Time	Result	#FI/#I	Time	Result	Time	Result	Iter
fsbench_ok	282	+	0/676	304	+	301	+	1
fsbench_bad	**<1**	+	729/729	**360**	+	**786**	+	2
indexer_ok	595	+	0/17160	220	+	218	+	1
aget-0.4_bad	137	+	1/1	127	+	125	+	1
bzip2smp_ok	1800	+	0/1294	MO	-	MO	-	1
reorder_bad	**<1**	+	1/154574	**MO**	-	**MO**	-	1
twostage_bad	**88**	+	1/139	**93**	+	**195**	+	5
wronglock_bad	**90**	+	6/104015	**MO**	-	**MO**	-	1
exStbHDMI_ok	229	+	0/1	226	+	213	+	1
exStbLED_ok	73	+	0/11	73	+	787	+	1
exStbThumb_bad	95	+	3/3	14	+	12	+	1
micro_10_ok	254	+	0/29260	MO	-	MO	-	1

Comparison to CHESS [Musuvathi and Qadeer]

- CHESS (v0.1.30626.0) is a concurrency testing tool for C# programs; also works for C/C++ (Windows API)
  - implements iterative context-bounding
  - requires unit tests that it repeatedly executes in a loop, exploring a different interleaving on each iteration
    - ▷ it is similar to our lazy approach
  - performs state hashing based on a happens-before graph
    - ▷ avoids exploring the same state repeatedly

- Goal: compare efficiency of the approaches
  - on identical verification problems taken from standard benchmark suites of multi-threaded software

ESS [Musuvathi and Qadeer]

CHESS is effective for programs where there are a small number of threads

		B	C	CHESS		Lazy	
				Time	Tests	Time	#FI/#I
reorder_4_bad (3,1)	4	4	5	**98**	130000	**<1**	1/82
reorder_5_bad (4,1)	5	5	6	TO	429000	<1	1/277
reorder_6_bad (5,1)	6	6	7	TO	396000	<1	1/853
reorder_6_bad (5,1)	6	6	8	TO	371000	<1	1/2810
reorder_6_bad (5,1)	6	6	9	TO	367000	≤1	1/8124
twostage_4_bad (3,1)	4	4	4	**215**	27000	**2**	1/42
twostage_5_bad (4,1)	5	5	4	TO	384000	2	1/44
twostage_6_bad (5,1)	6	6	4	TO	366000	2	1/45
wronglock_4_bad (1,3)	4	4	8	**21**	3000	**5**	2/489
wronglock_5_bad (1,4)	5	5	8	**724**	93000	**10**	3/2869
wronglock_6_bad (1,5)	6	6	8	TO	356000	18	4/12106
micro_2_ok (100)	2	1	2	**316**	35855	**<1**	0/4
micro_2_ok (100)	2	1	17	TO	40000	1095	0/131072

SS [Musuvathi and Qadeer]

CHESS is effective for programs where there are a small number of threads, **but it does not scale that well and consistently runs out of time when we increase the number of threads**

				CHESS		Lazy	
				Time	Tests	Time	#FI/#I
			5	98	130000	<1	1/82
reorder_5_bad (4,1)	5	5	6	TO	429000	<1	1/277
reorder_6_bad (5,1)	6	6	7	TO	396000	<1	1/853
reorder_6_bad (5,1)	6	6	8	TO	371000	<1	1/2810
reorder_6_bad (5,1)	6	6	9	TO	367000	≤1	1/8124
twostage_4_bad (3,1)	4	4	4	215	27000	2	1/42
twostage_5_bad (4,1)	5	5	4	TO	384000	2	1/44
twostage_6_bad (5,1)	6	6	4	TO	366000	2	1/45
wronglock_4_bad (1,3)	4	4	8	21	3000	5	2/489
wronglock_5_bad (1,4)	5	5	8	724	93000	10	3/2869
wronglock_6_bad (1,5)	6	6	8	TO	356000	18	4/12106
micro_2_ok (100)	2	1	2	316	35855	<1	0/4
micro_2_ok (100)	2	1	17	TO	40000	1095	0/131072

Comparison to SATABS [D. Kroening]

- SATABS (v2.5) implements predicate abstraction using SAT
  - avoids exponential number of theorem prover calls (for each potential assignment) to construct the Boolean program
  - uses BDD-based model checking (Cadence SMV) to verify the Boolean program
  - supports most ANSI-C constructs (incl. arithmetic overflow) and the verification of multi-threaded software with locks and shared variables

- Goal: compare efficiency of both approaches
  - on identical verification problems taken from standard benchmark suites of multi-threaded software

Comparison to SATABS [D. Kroening]

failed to validate the counterexample

failed to refine the predicate

	SATABS		Lazy		
	Time	Result	Time	Result	#FI/#I
fsbench_ok	†	-	**282**	+	0/676
fsbench_bad	†	-	**<1**	+	729/729
indexer_ok	TO	-	595	+	0/17160
aget-0.4_bad	3346	+	137	+	1/1
bzip2smp_ok	TO	-	1800	+	0/1294
	1	-	<1	+	1/154574
	2	-	88	+	1/139
wrongloc[illegible]d	2	-	90	+	6/104015
exStbHDMI_ok	TO	-	229	+	0/1
exStbLED_ok	RF	-	**73**	+	0/11
exStbThumb_bad	317	+	95	+	3/3
micro_10_ok	TO	-	254	+	0/29260

Comparison to SATABS [D. Kroening]

Module	SATABS		Lazy		
	Time	Result	Time	Result	#FI/#I
fsbench_ok	†	-	282	+	0/676
fsbench_bad	†	-	<1	+	729/729
	TO	-	595	+	0/17160
	3346	+	137	+	1/1
bzip2smp_	TO	-	1800	+	0/1294
reorder_bad	1	-	**<1**	+	1/154574
twostage_bad	2	-	**88**	+	1/139
wronglock_bad	2	-	**90**	+	6/104015
exStbHDMI_ok	TO	-	229	+	0/1
exStbLED_ok	RF	-	73	+	0/11
exStbThumb_bad	317	+	95	+	3/3
micro_10_ok	TO	-	254	+	0/29260

false positives answers

TABS [D. Kroening]

SATABS uses predicate abstraction and refinement and tries to solve a harder problem than ESBMC

	ATABS		Lazy		
		Result	Time	Result	#FI/#I
fsbench_ok	†	-	282	+	0/676
fsbench_bad	†	-	<1	+	729/729
indexer_ok	TO	-	595	+	0/17160
aget-0.4_bad	3346	+	**137**	+	1/1
bzip2smp_ok	TO	-	1800	+	0/1294
reorder_bad	1	-	<1	+	1/154574
twostage_bad	2	-	88	+	1/139
wronglock_bad	2	-	90	+	6/104015
exStbHDMI_ok	TO	-	229	+	0/1
exStbLED_ok	RF	-	73	+	0/11
exStbThumb_bad	317	+	**95**	+	3/3
micro_10_ok	TO	-	254	+	0/29260

SATABS uses predicate abstraction and refinement and tries to solve a harder problem than ESBMC, **but this problem may still be too hard as SATABS is unable to prove the required properties**

. Kroening]

			Lazy		
			Time	Result	#FI/#I
fsbench_ok		-	282	+	0/676
fsbench_bad		-	≤1	+	729/729
indexer_ok	TO	-	**595**	+	0/17160
aget-0.4_bad	3346	+	137	+	1/1
bzip2smp_ok	TO	-	**1800**	+	0/1294
reorder_bad	1	-	<1	+	1/154574
twostage_bad	2	-	88	+	1/139
wronglock_bad	2	-	90	+	6/104015
exStbHDMI_ok	TO	-	**229**	**+**	0/1
exStbLED_ok	RF	-	73	+	0/11
exStbThumb_bad	317	+	95	+	3/3
micro_10_ok	TO	-	**254**	+	0/29260