DSSE Dependable Systems & Software Engineering

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Bounded Model Checking of Multi-threaded Software using SMT solvers

Lucas Cordeiro and Bernd Fischer

lcc08r@ecs.soton.ac.uk

DSSE Dependable Systems & Software Engineering

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Bounded Model Checking of ***Multi-threaded Software*** *using SMT solvers*

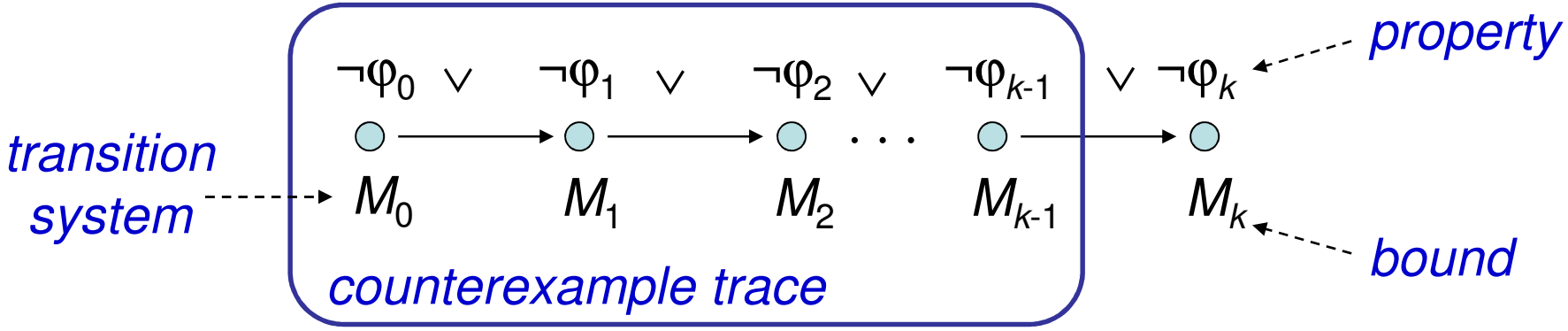
Lucas Cordeiro and Bernd Fischer

lcc08r@ecs.soton.ac.uk

Bounded Model Checking (BMC)

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Basic Idea: check negation of given property up to given depth



- transition system *M* unrolled *k* times
  - for programs: unroll loops, unfold arrays, …
- translated into verification condition ψ such that

  ψ satisfiable iff φ has counterexample of max. depth *k*
- has been applied successfully to verify (embedded) software

BMC of Multi-threaded Software

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- concurrency bugs are tricky to **reproduce/debug** because they usually occur under specific thread interleavings
  - most common errors: *67% related to atomicity and order violations, 30% related to deadlock* [Lu et al.'08]
- problem: the number of interleavings grows exponentially with the number of threads (n) and program statements (s)
  - *number of executions:* $O(n^s)$
  - *context switches among threads increase the number of possible executions*
- two important observations help us:
  - concurrency bugs are shallow [Qadeer&Rehof'05]
  - SAT/SMT solvers produce unsatisfiable cores that allow us to remove logic that is not relevant

Objective of this work

UNIVERSITY OF Southampton
School of Electronics and Computer Science

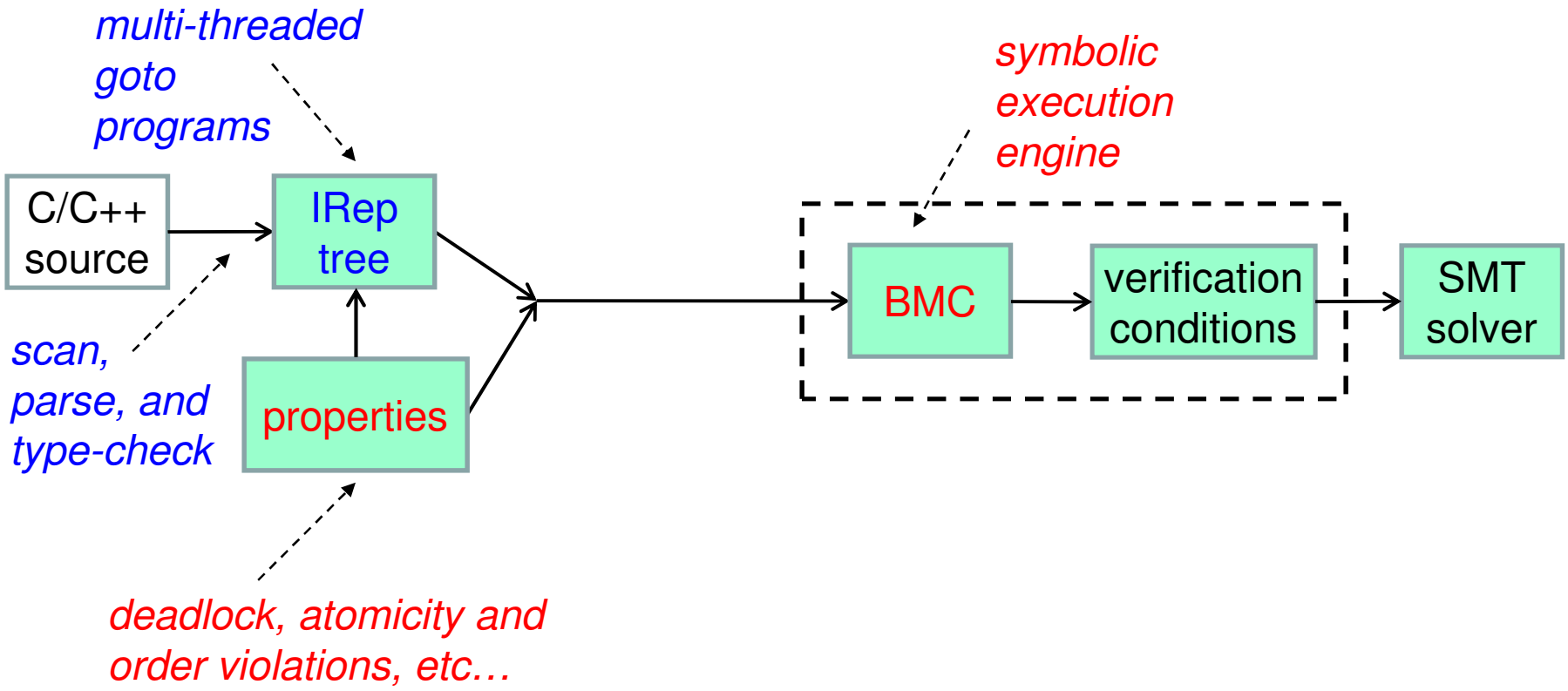
Exploit SMT to improve BMC of multi-threaded software

- exploit SMT solvers to:
  - prune the *property and data dependent* search space (non-chronological backtracking and conflict clauses learning)
  - remove interleavings that are not relevant by analyzing the proof of unsatisfiability [***NOT** Craig Interpolants **yet***]
- propose three approaches to SMT-based BMC:
  - *lazy exploration* of the interleavings
  - *schedule guards* to encode all interleavings
  - *underapproximation and widening (UW) [Grumberg&et al.'05]*
- evaluate our approaches over multi-threaded applications

Lazy exploration of interleavings

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

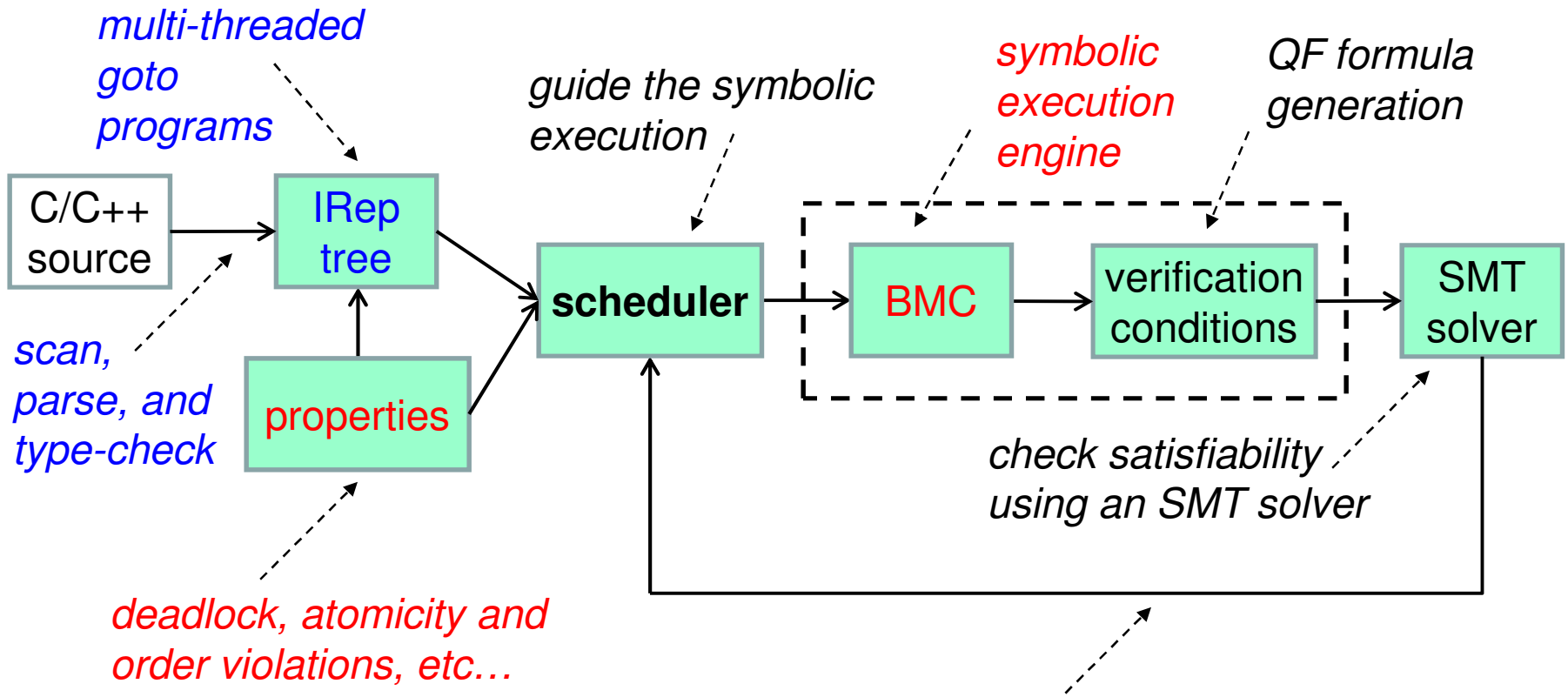


reused/extended from the CBMC model checker

Lazy exploration of interleavings

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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



reused/extended from the CBMC model checker

Lazy exploration: Scheduler

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- the scheduler … at a given time (emulate the …

Step 1: expa… thread with … smallest sub…

Step 2: generate interleavings starting with T_2

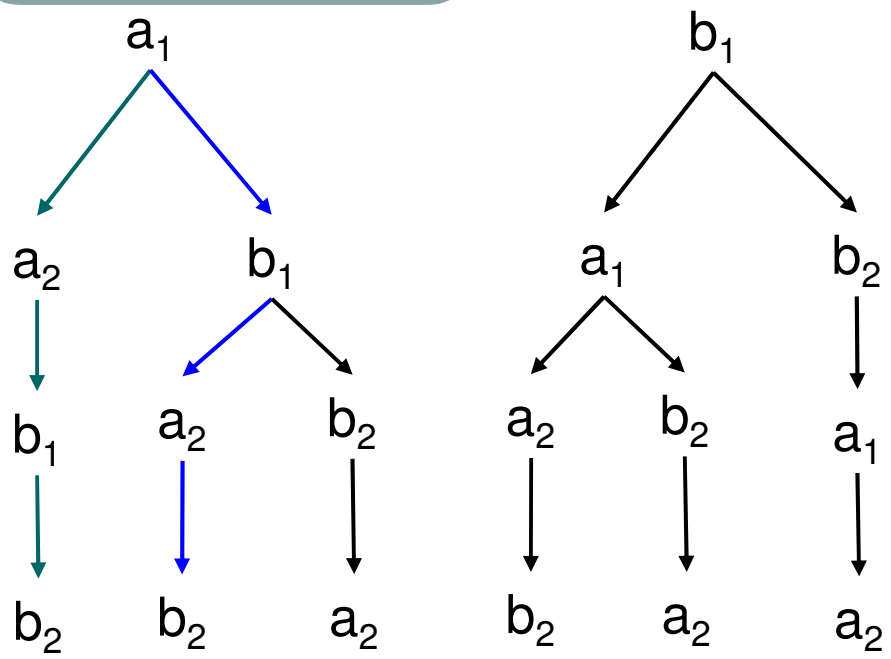
Thread T_1	Thread T_2
a_1	b_1
a_2	b_2

Thread interleavings:

$a_1; a_2; b_1; b_2$

$a_1; b_1; a_2; b_2$

…



- allow preemptions only before visible statements (global variables and synchronization points)

Running Example

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- 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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

statements: 1-2-3-7

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);
```

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

Lazy exploration: interleaving I$_s$

UNIVERSITY OF Southampton
School of Electronics

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;

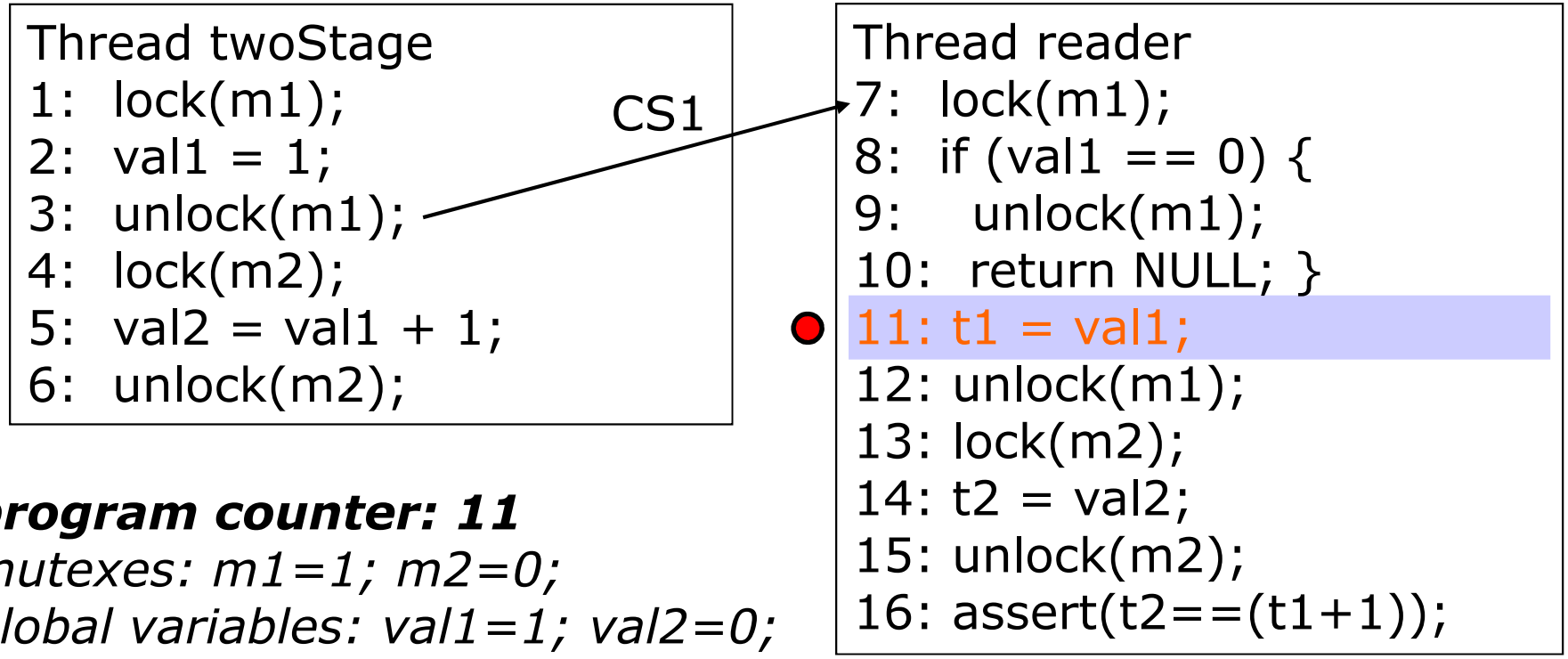
Lazy exploration: interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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;*

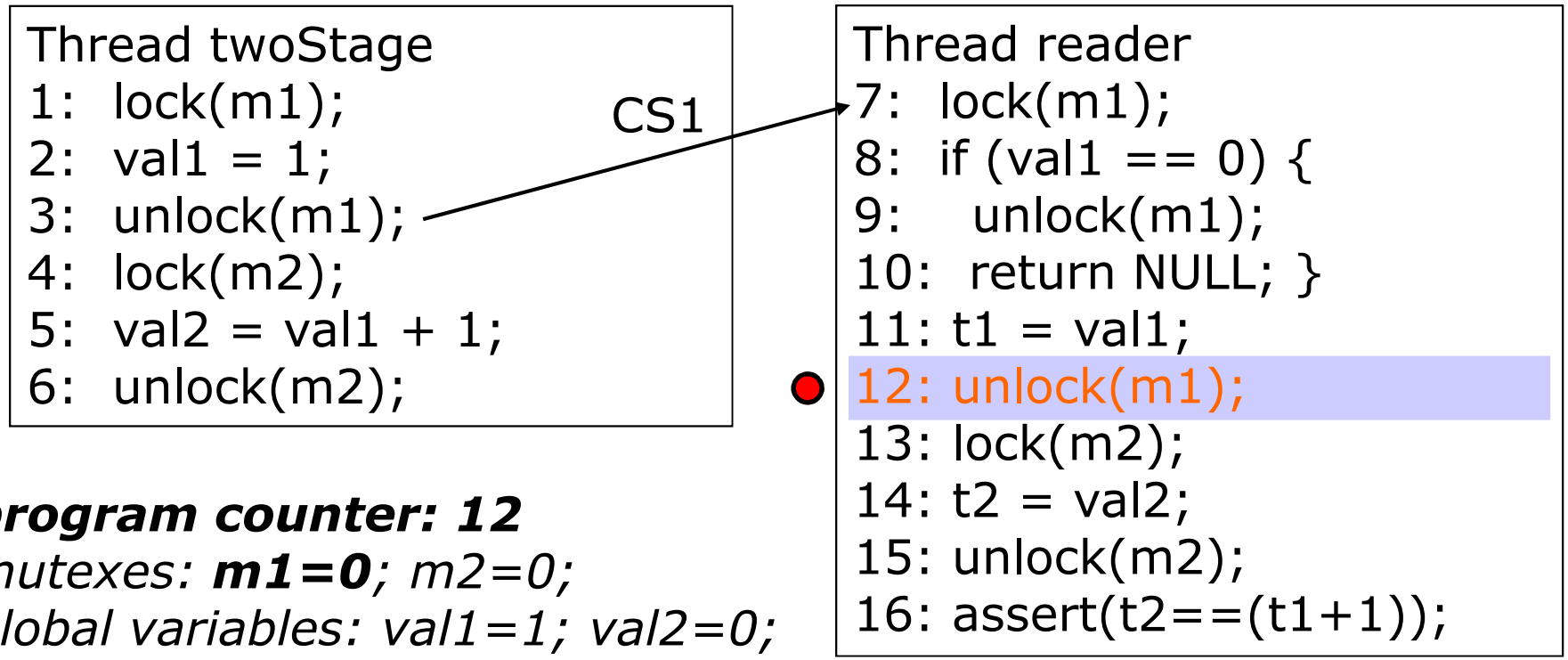
Lazy exploration: interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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;

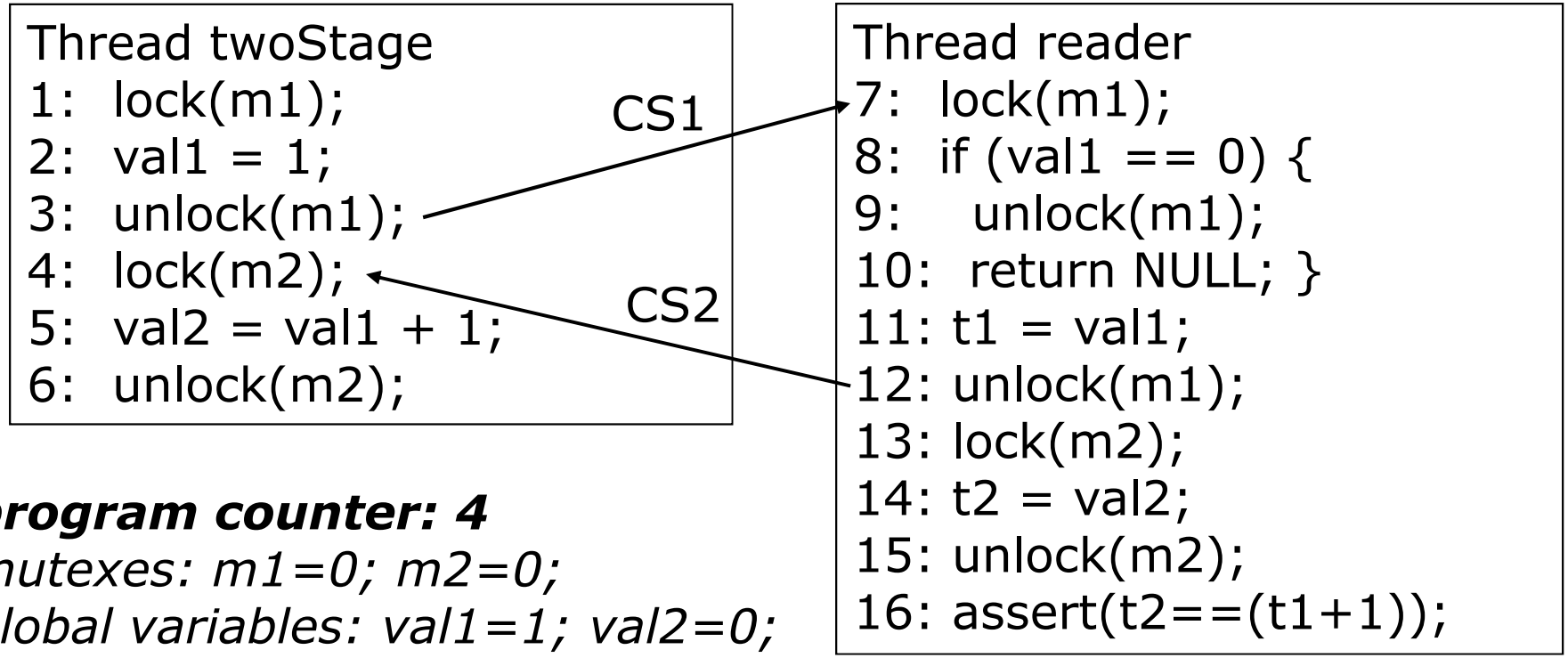
Lazy exploration: interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

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

val2-access:

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

CS1

CS2

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

Lazy exploration: interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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: $W_{twoStage,5}$

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

CS1

CS2

```
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: 5

mutexes: m1=0; m2=1;

global variables: val1=1; ***val2=2****;*

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

Lazy exploration: interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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: $W_{twoStage,5}$

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

CS1

CS2

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

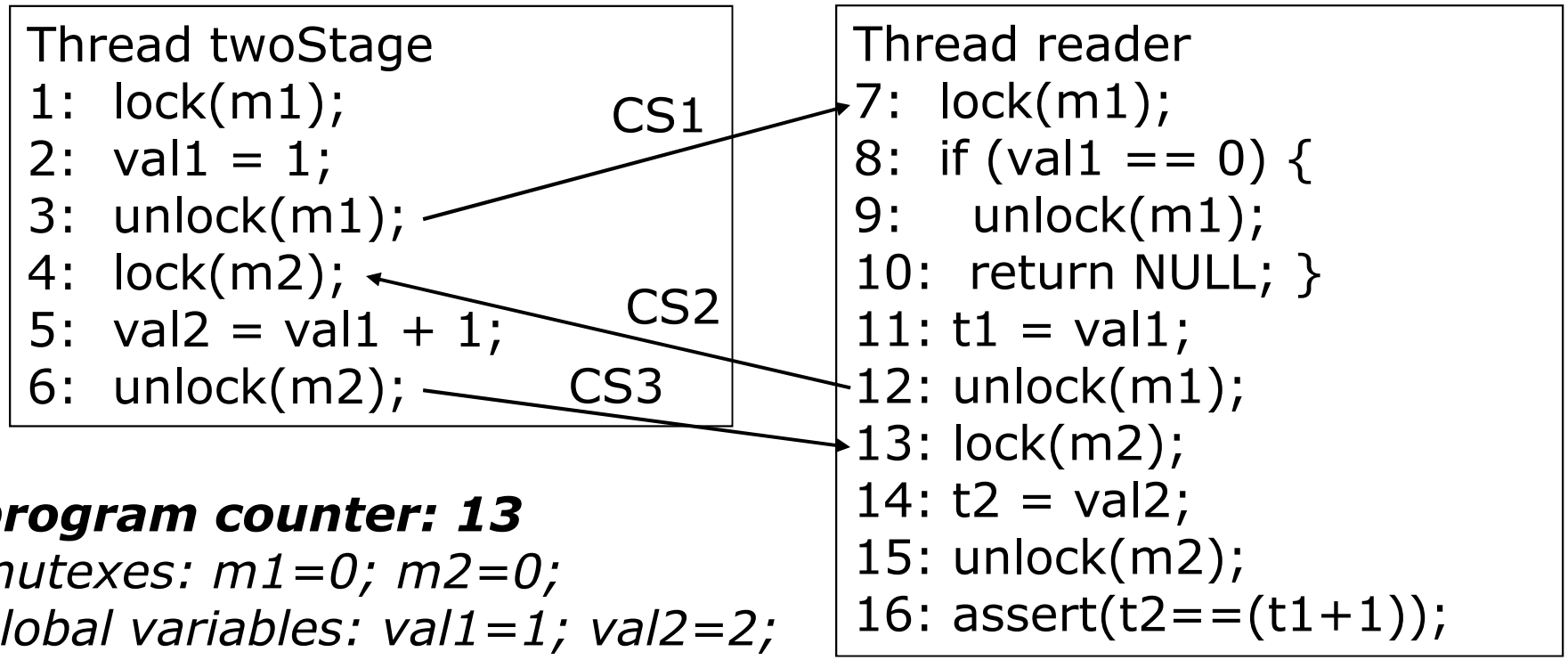
Lazy exploration: interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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;

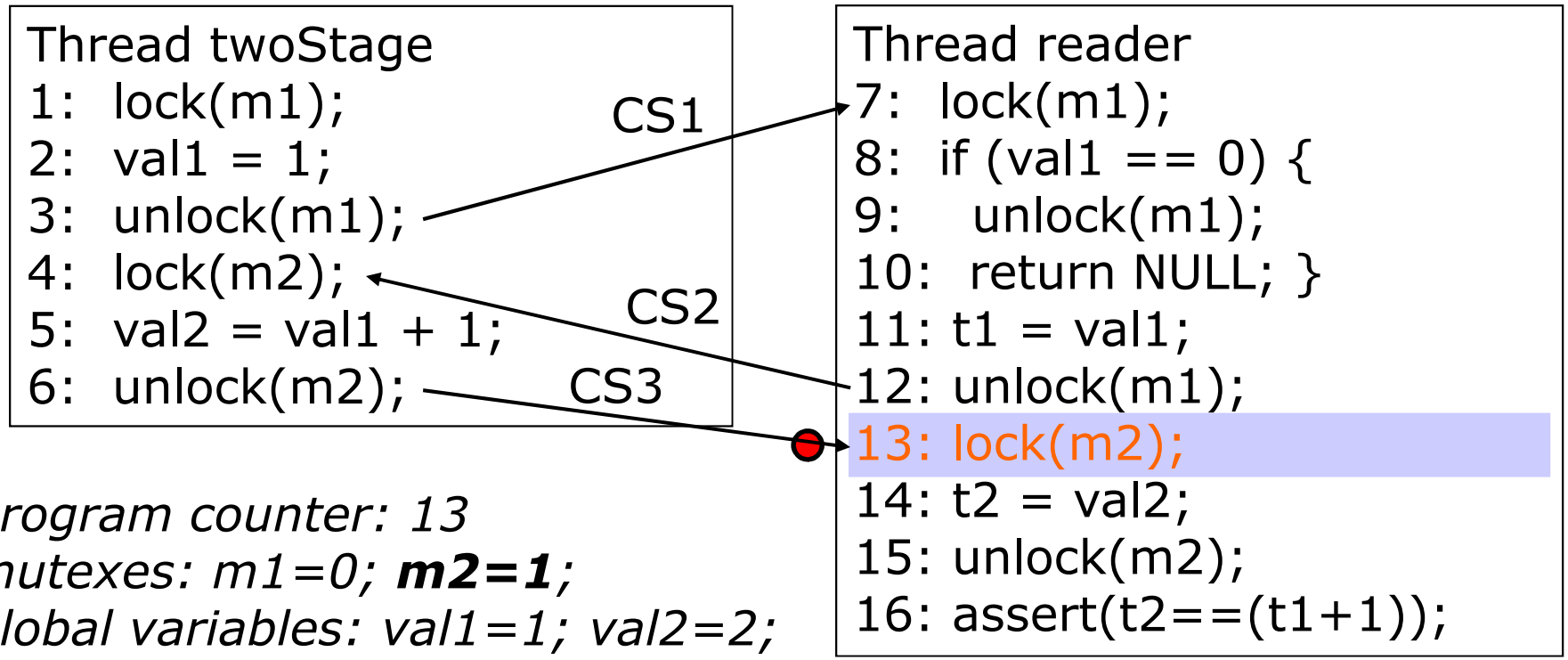
Lazy exploration: interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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;

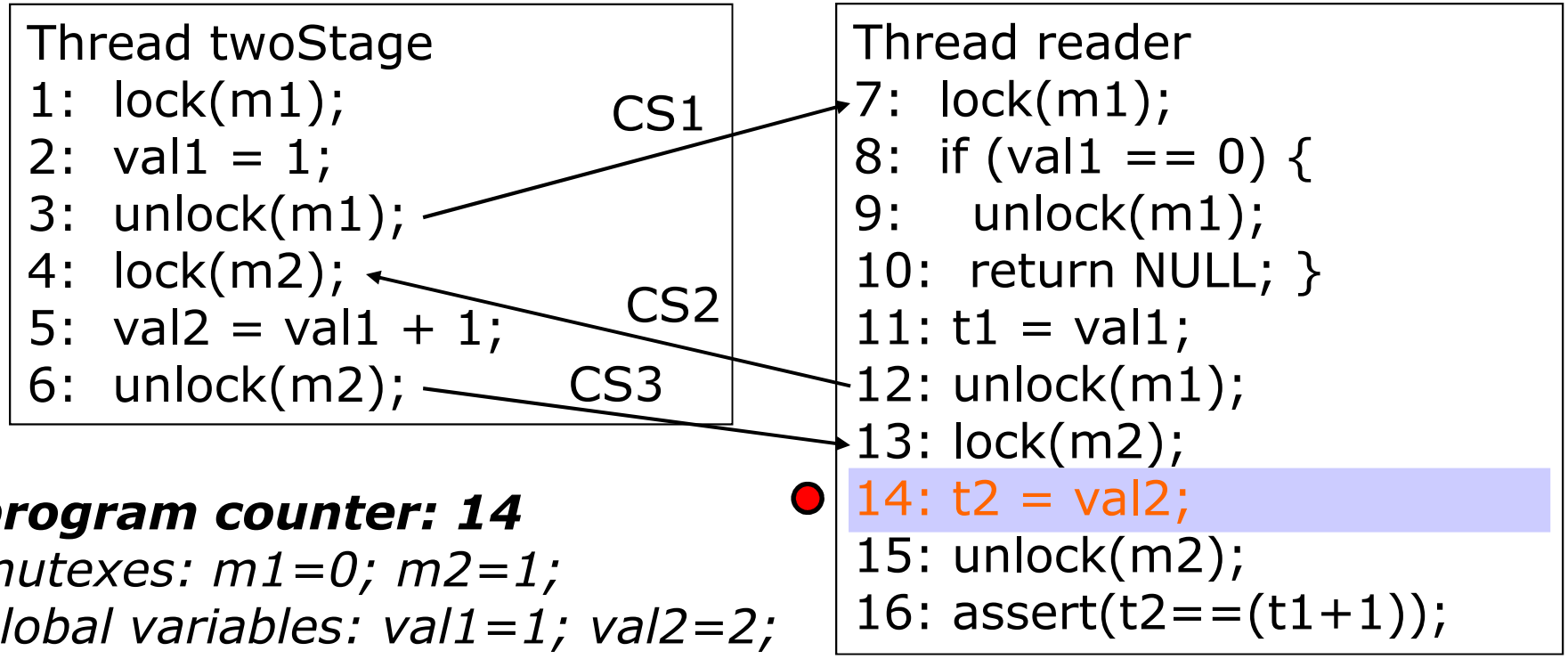
Lazy exploration: interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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;***

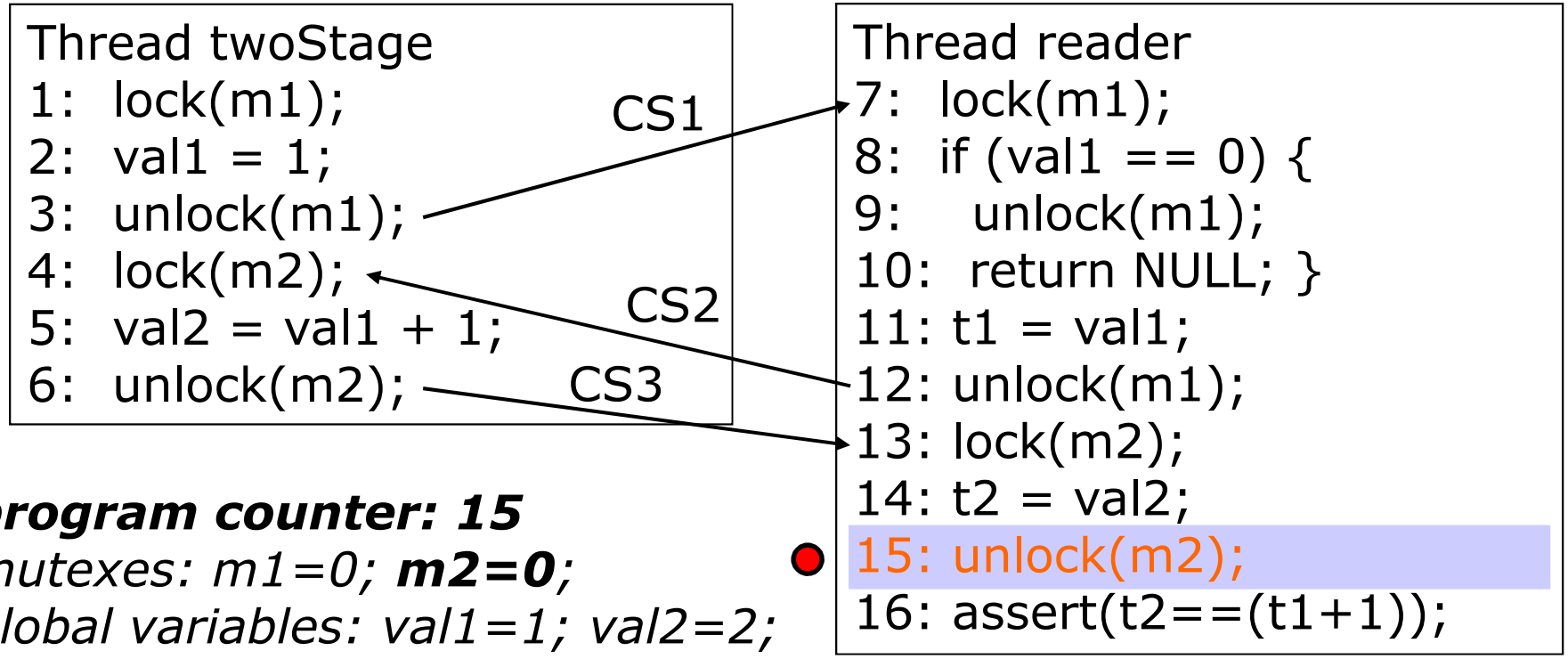
Lazy exploration: interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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;

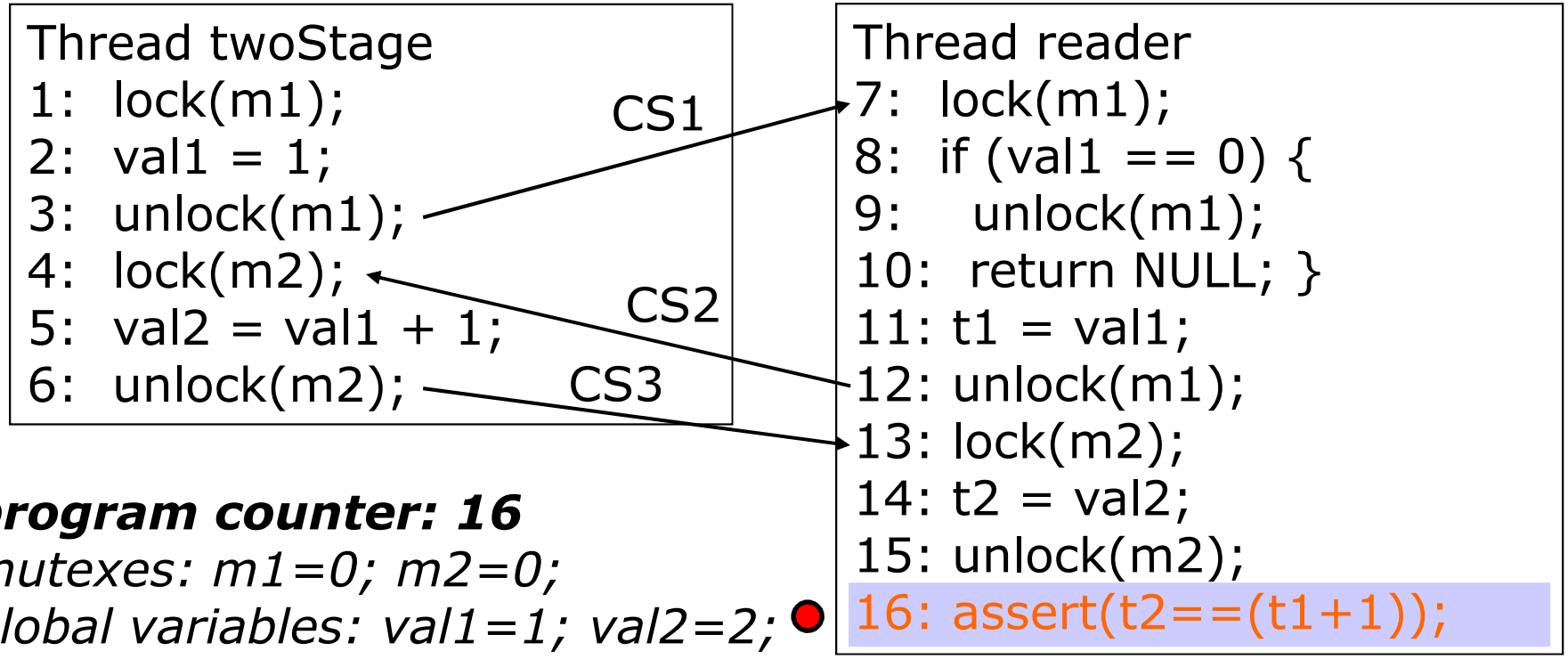
Lazy exploration: interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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;

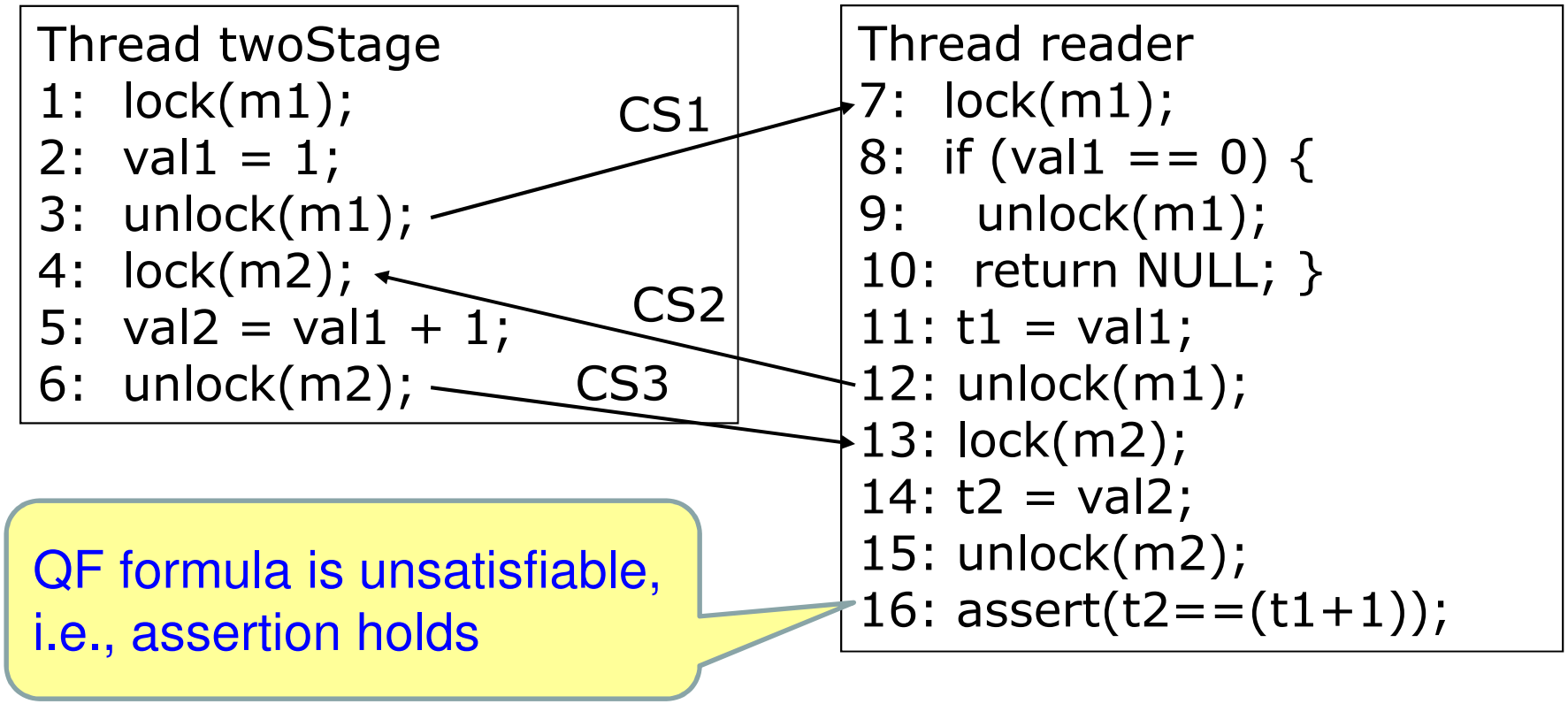
Lazy exploration: interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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****;*

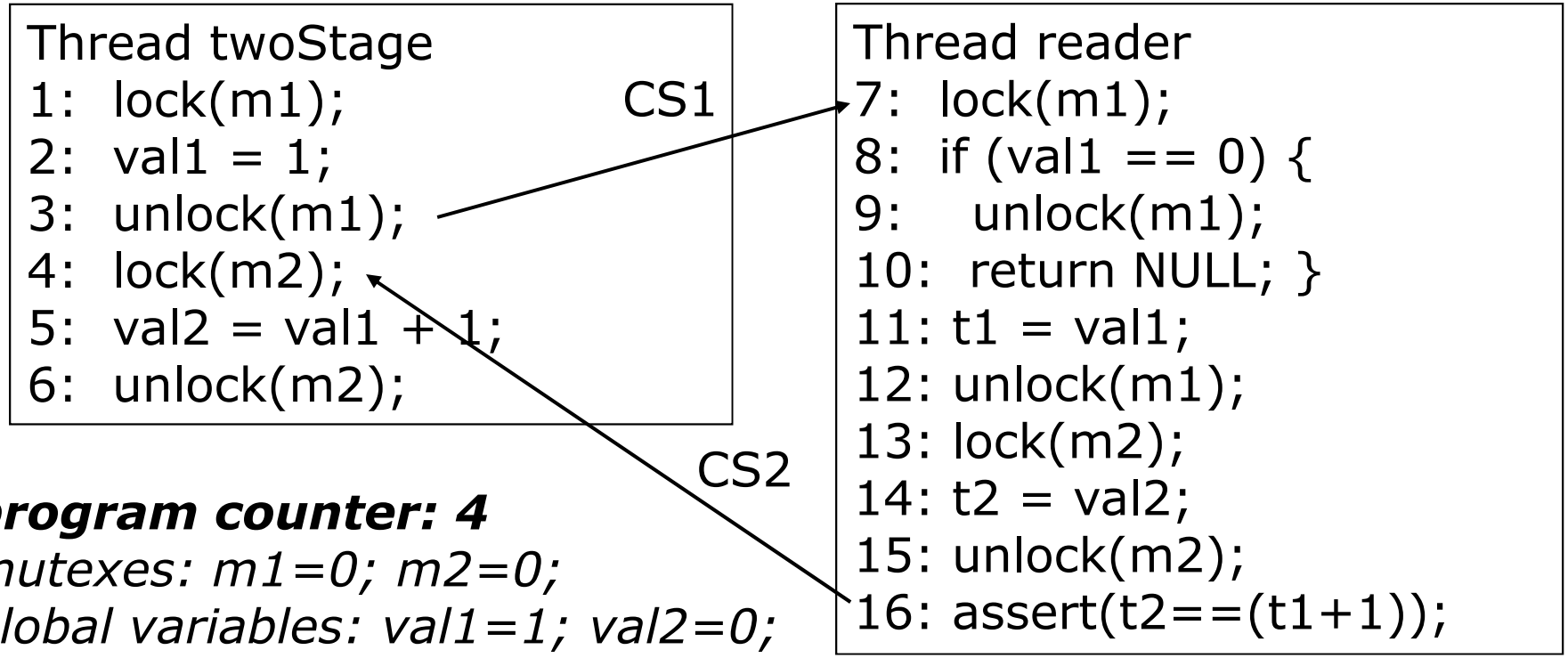
Lazy exploration: interleaving I_f

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

```
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));
```

CS2

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

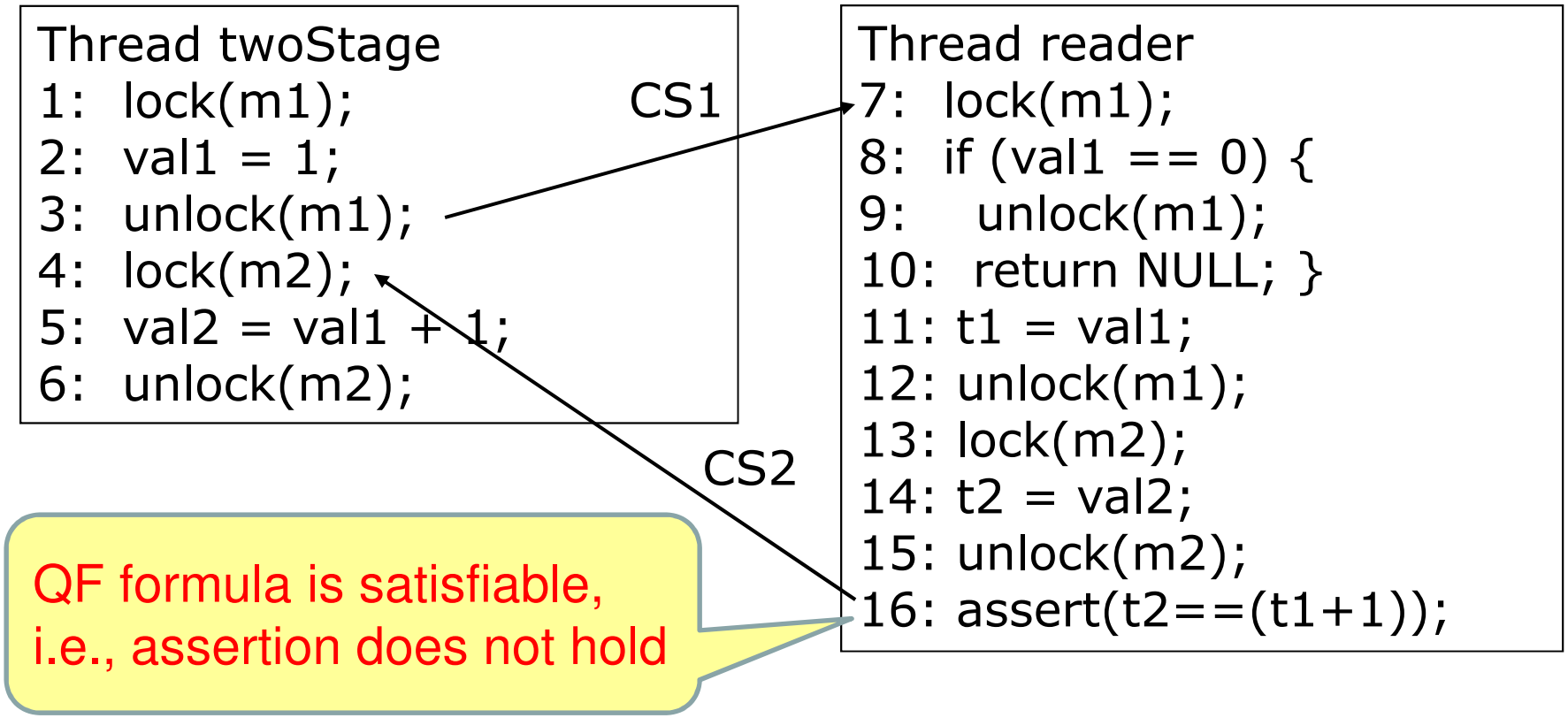
Lazy exploration: interleaving I_f

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

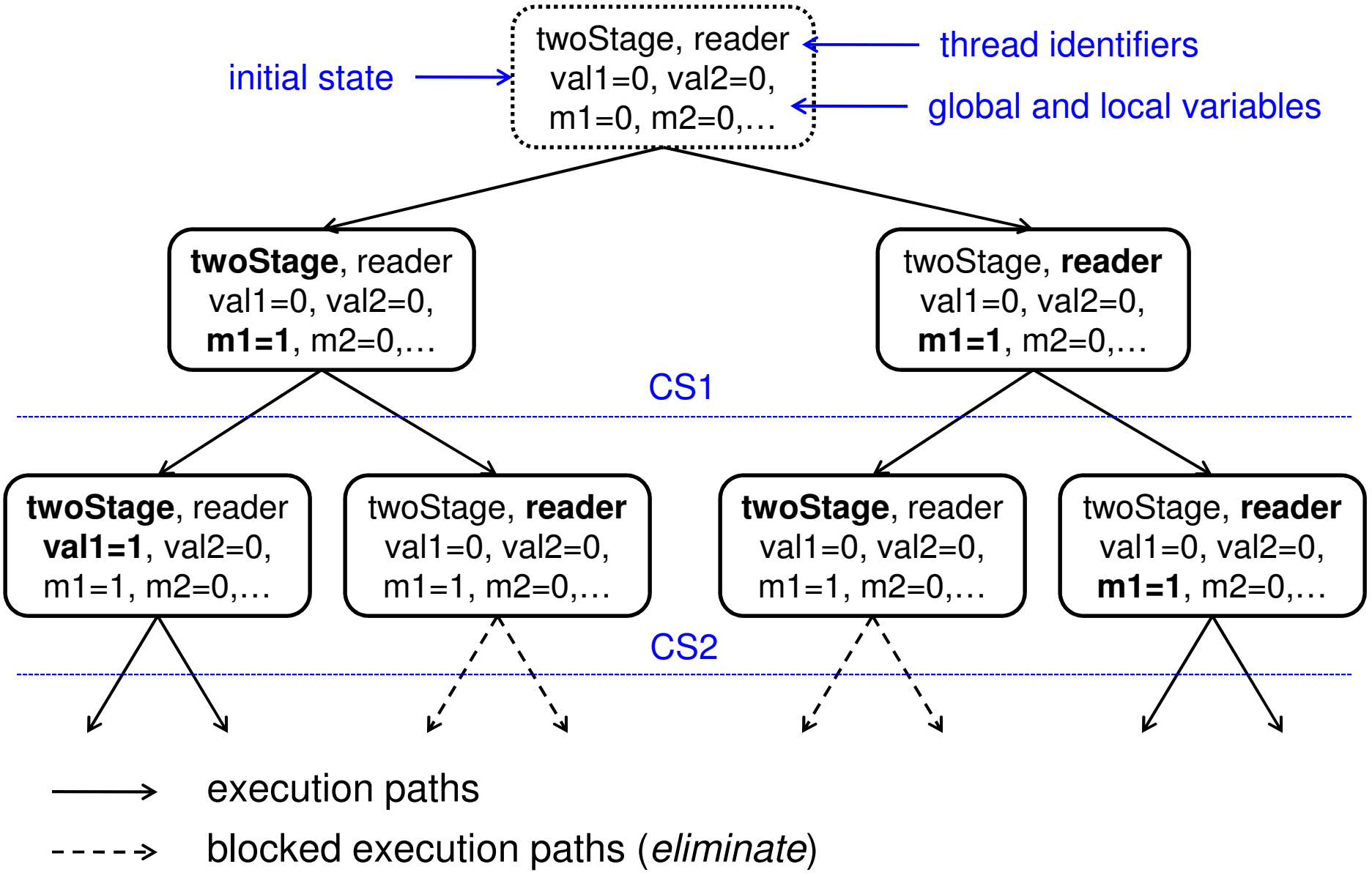
val1-access: $W_{twoStage,2}$- $R_{reader,8}$- $R_{reader,11}$ - $R_{twoStage,5}$

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



Lazy Approach: State Transitions

UNIVERSITY OF Southampton
School of Electronics and Computer Science



Observations about the lazy approach

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- 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

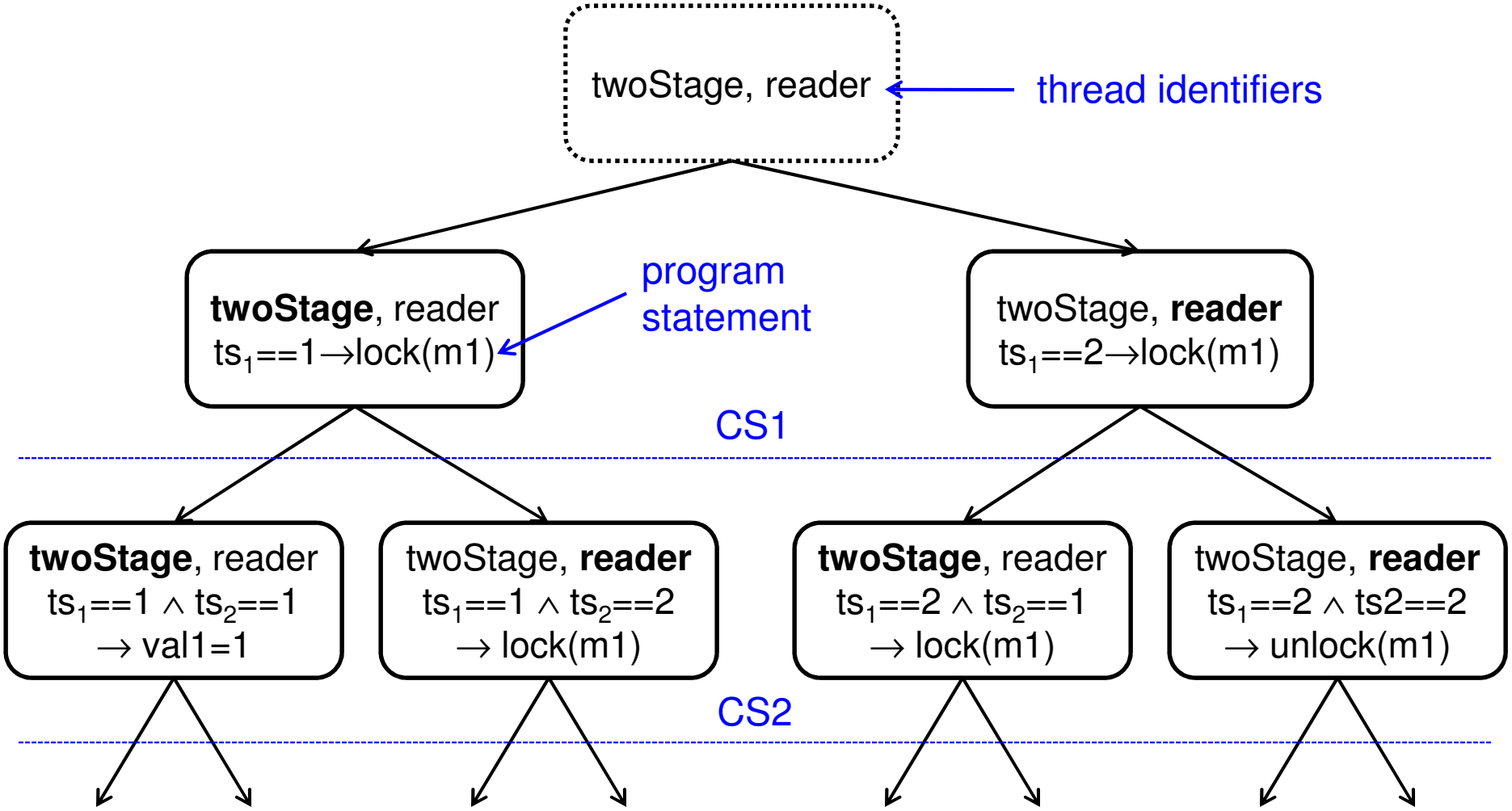
UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

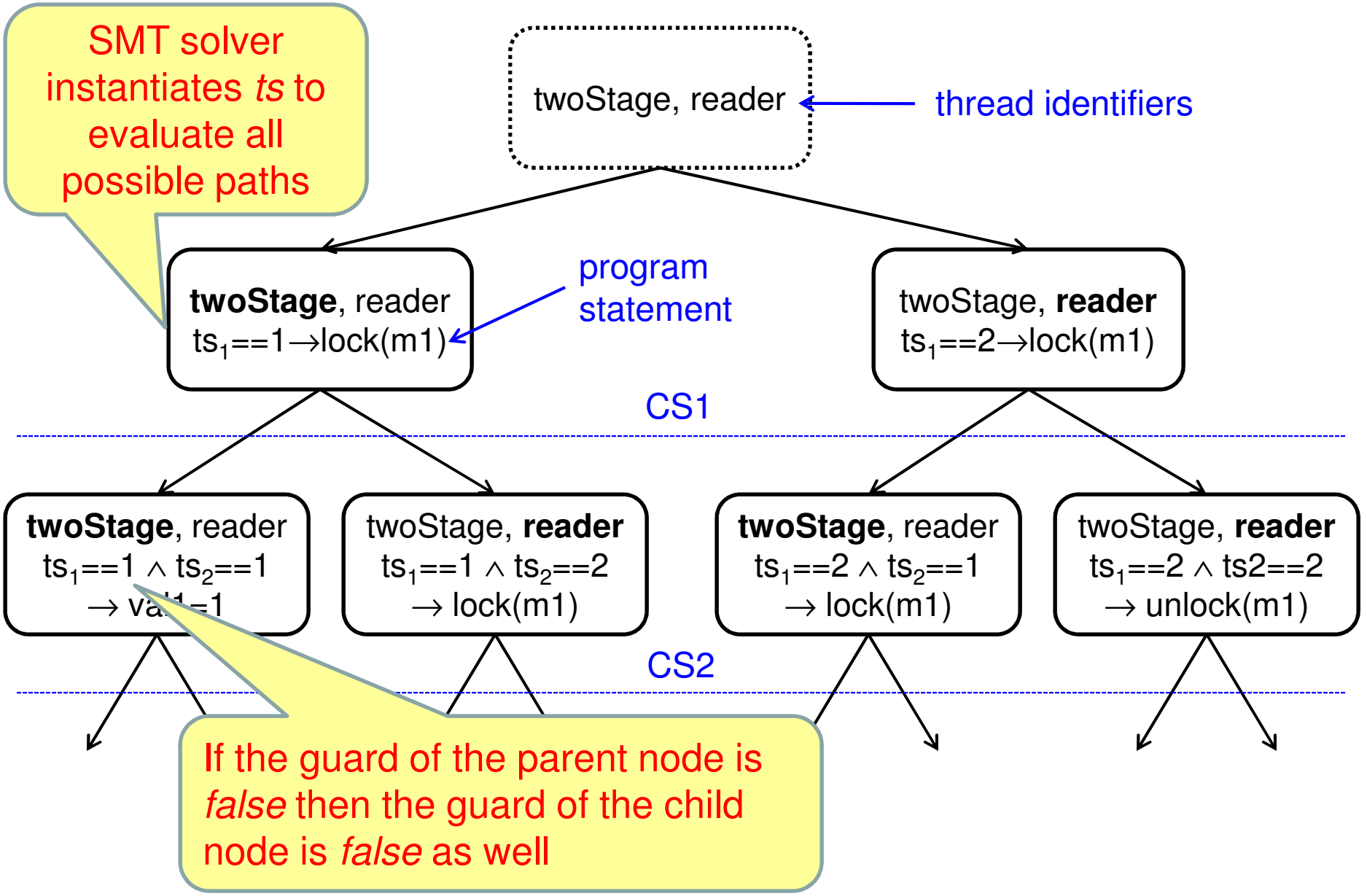
Schedule Recording: Execution Paths

UNIVERSITY OF Southampton
School of Electronics and Computer Science



Schedule Recording: Execution Paths

UNIVERSITY OF Southampton
School of Electronics and Computer Science



Schedule Recording: Interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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);
14: t2 = val2;
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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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);
```

CS

```
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));
```

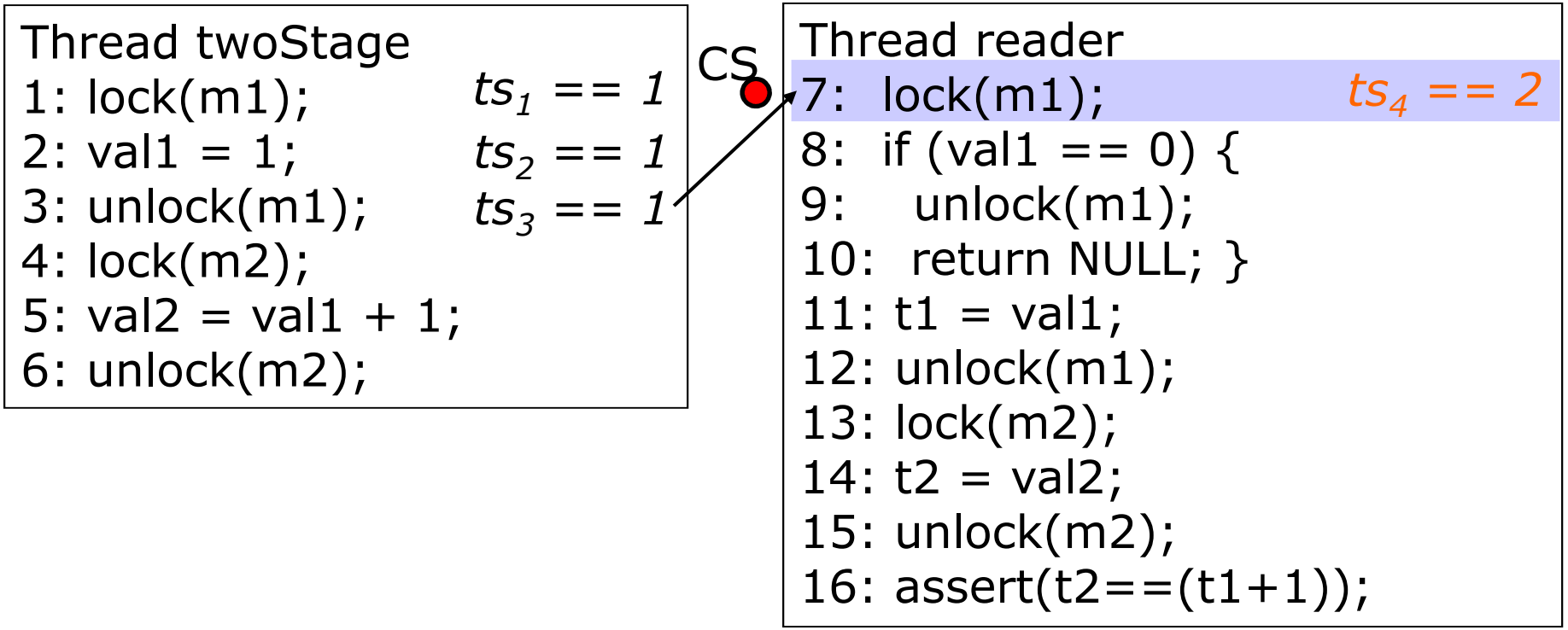
Schedule Recording: Interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

statements: 1-2-3-7-8

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

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

```
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);
```

CS

```
Thread reader
7:  lock(m1);                 ts4 == 2
8:  if (val1 == 0) {          ts5 == 2
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));
```

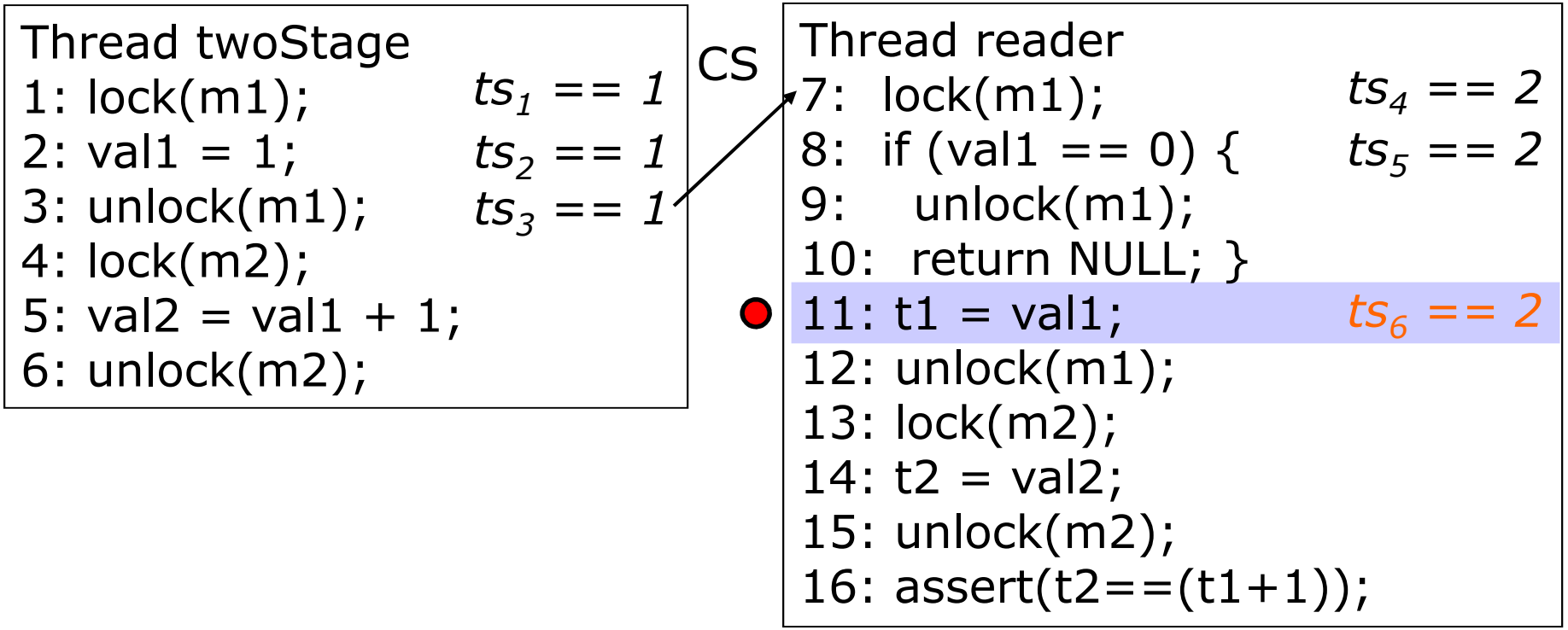
Schedule Recording: Interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

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

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

```
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);
```

CS

```
Thread reader
7:  lock(m1);                 ts4 == 2
8:  if (val1 == 0) {          ts5 == 2
9:    unlock(m1);
10:  return NULL; }
11: t1 = val1;                ts6 == 2
12: unlock(m1);               ts7 == 2
13: lock(m2);
14: t2 = val2;
15: unlock(m2);
16: assert(t2==(t1+1));
```

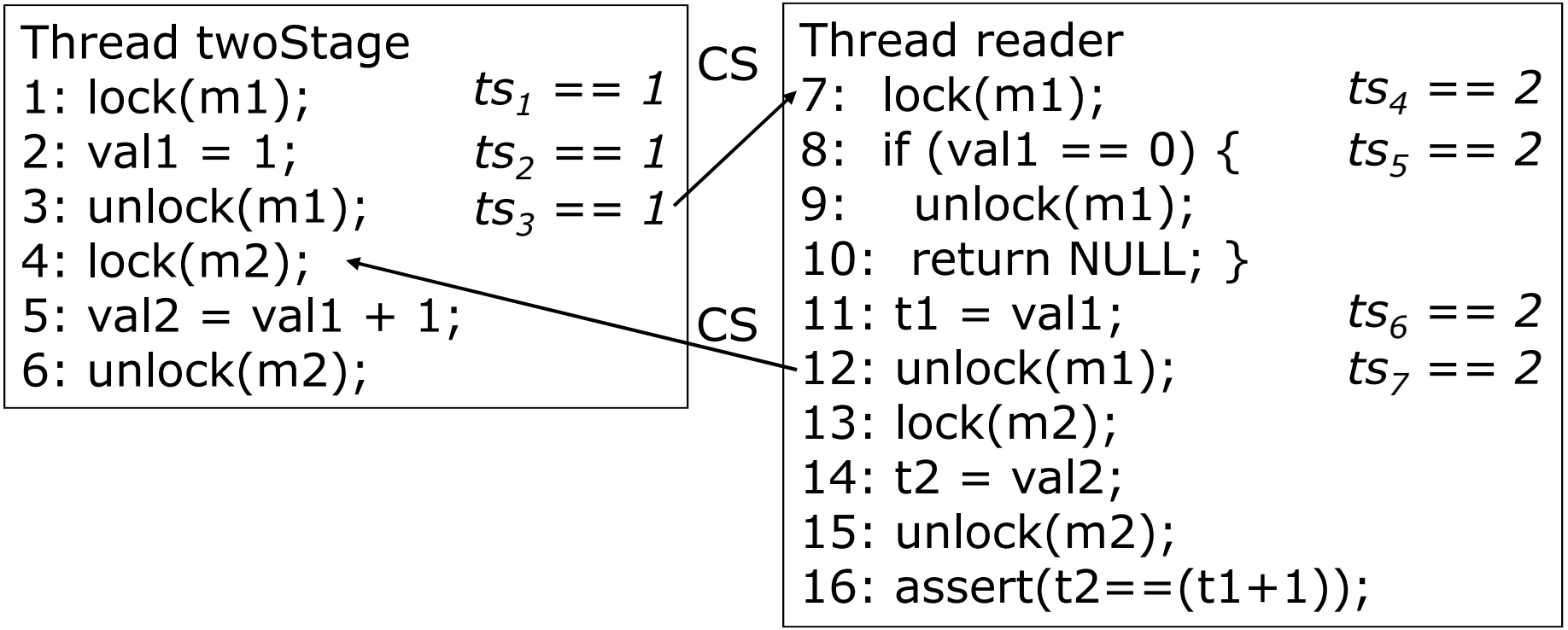
Schedule Recording: Interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

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

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



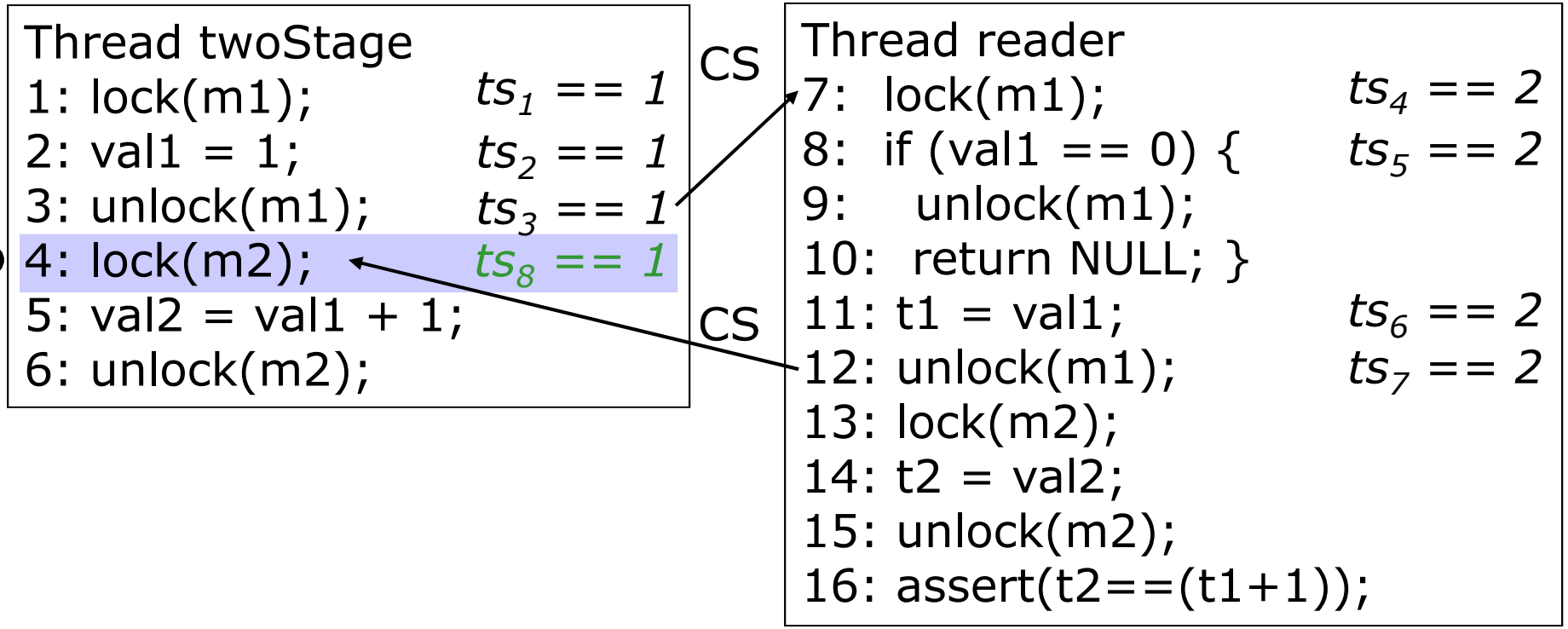
Schedule Recording: Interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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



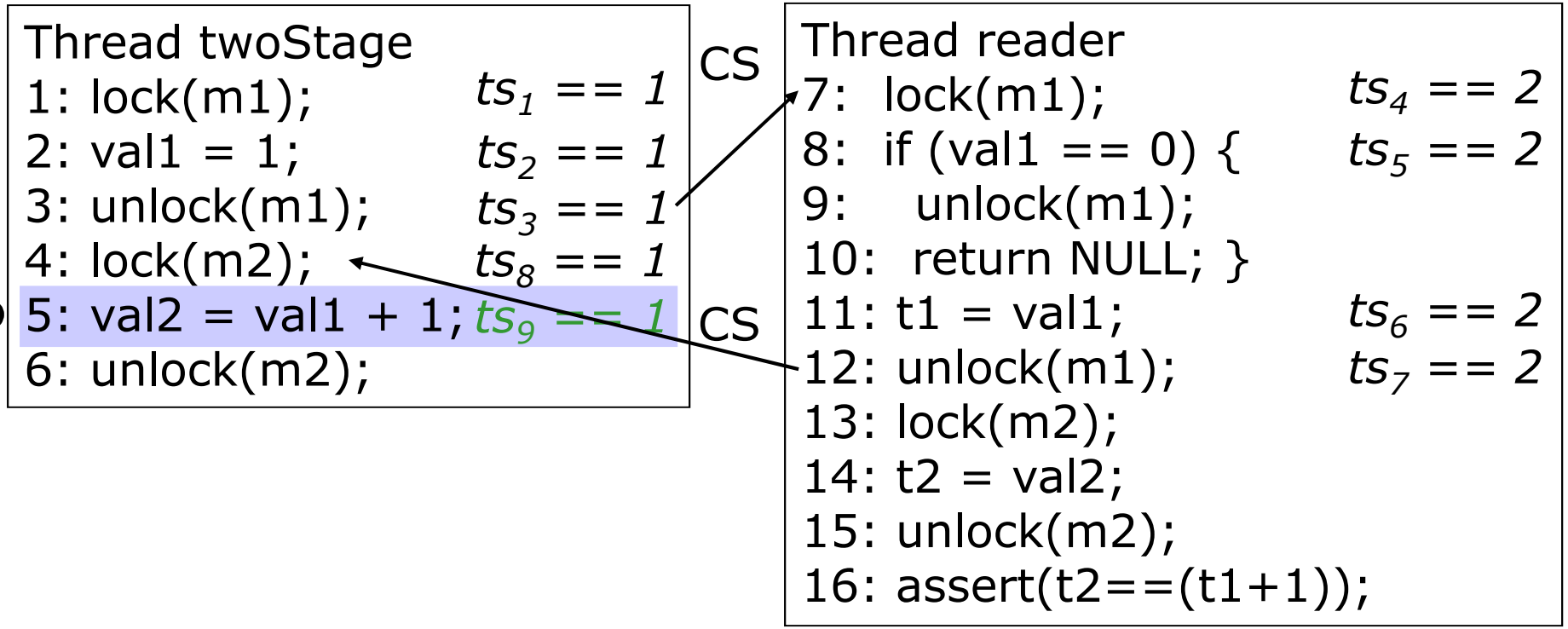
Schedule Recording: Interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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



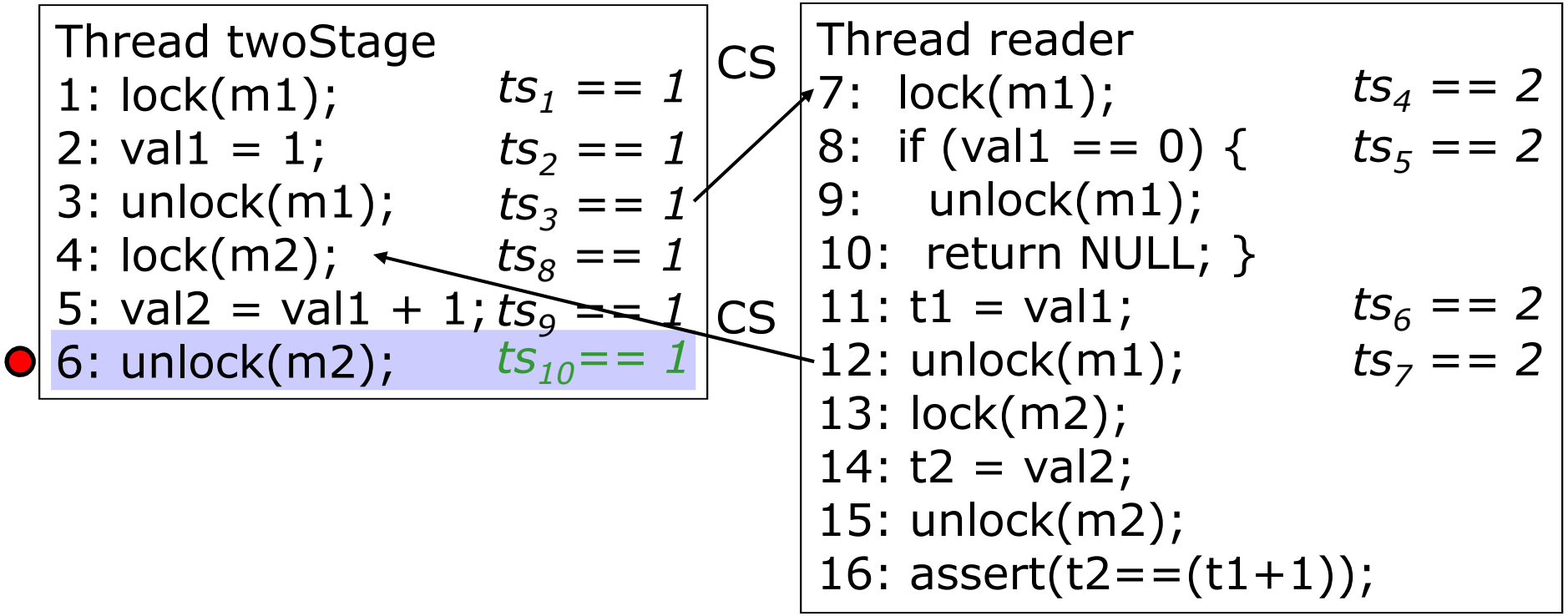
Schedule Recording: Interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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



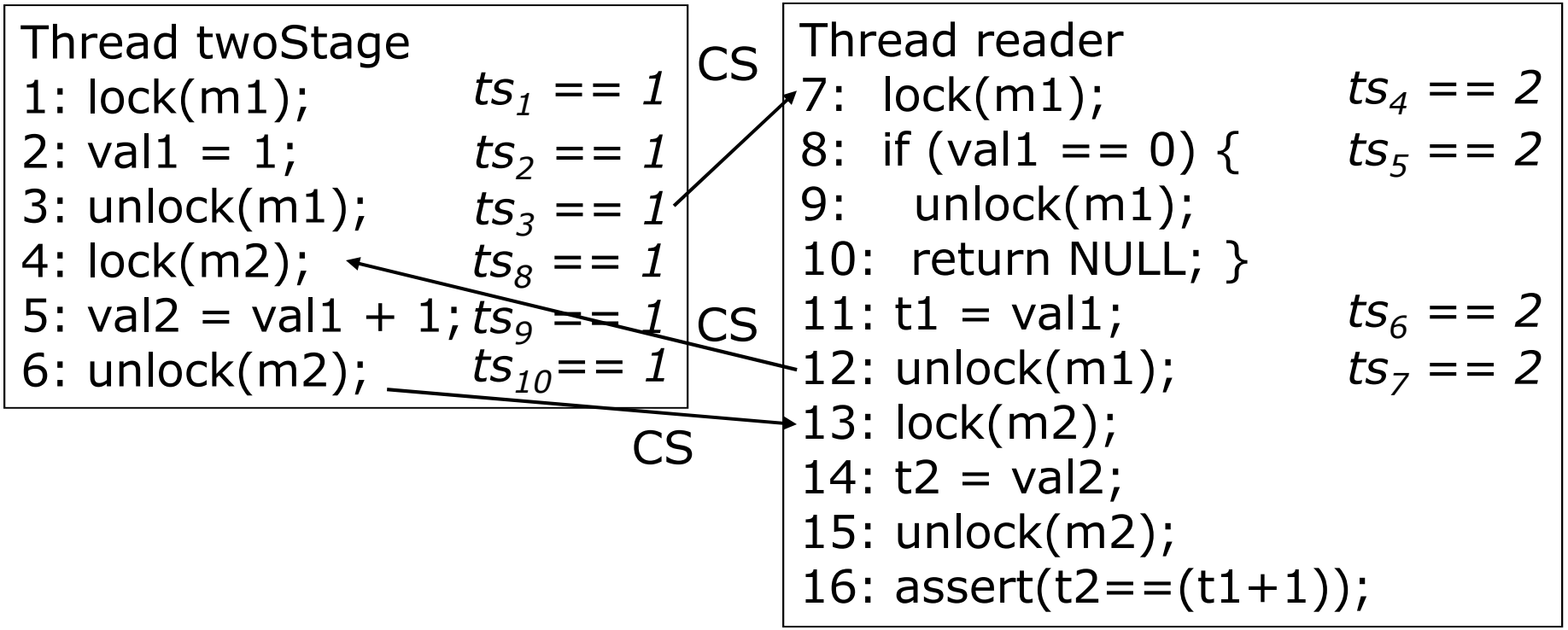
Schedule Recording: Interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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



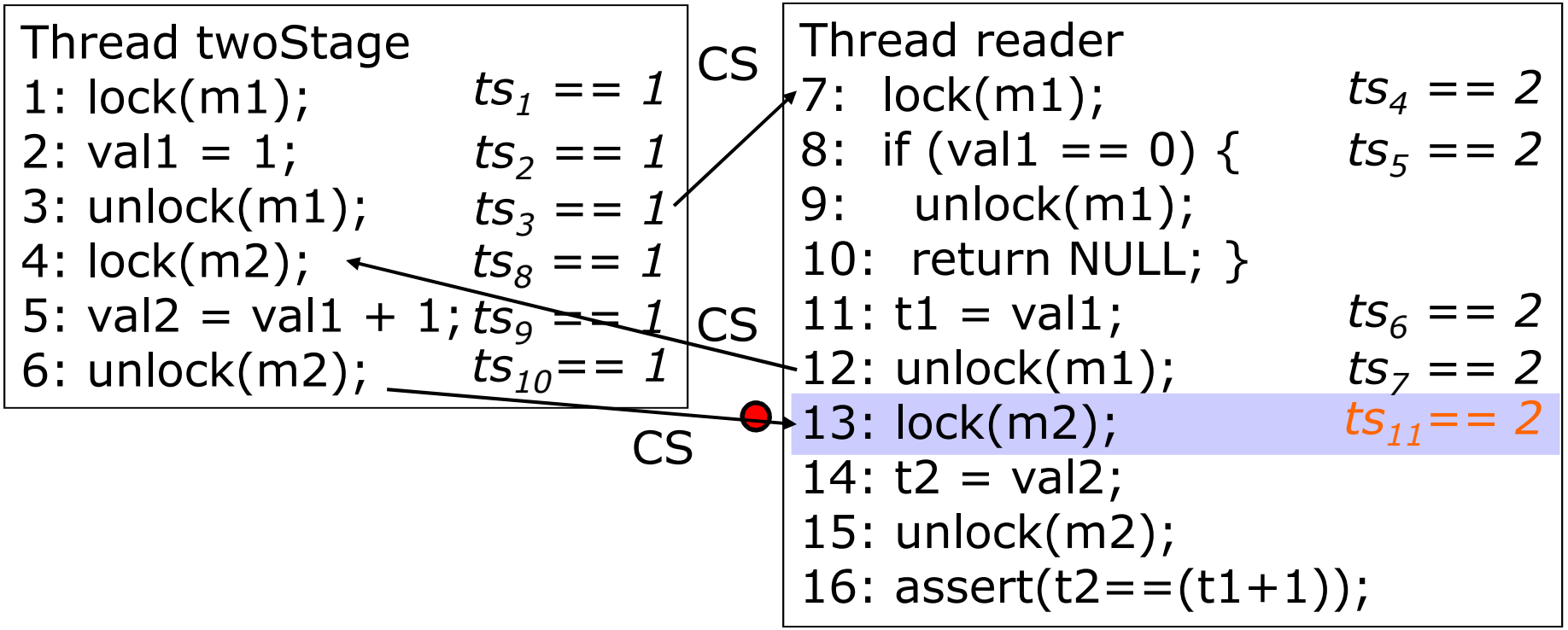
Schedule Recording: Interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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



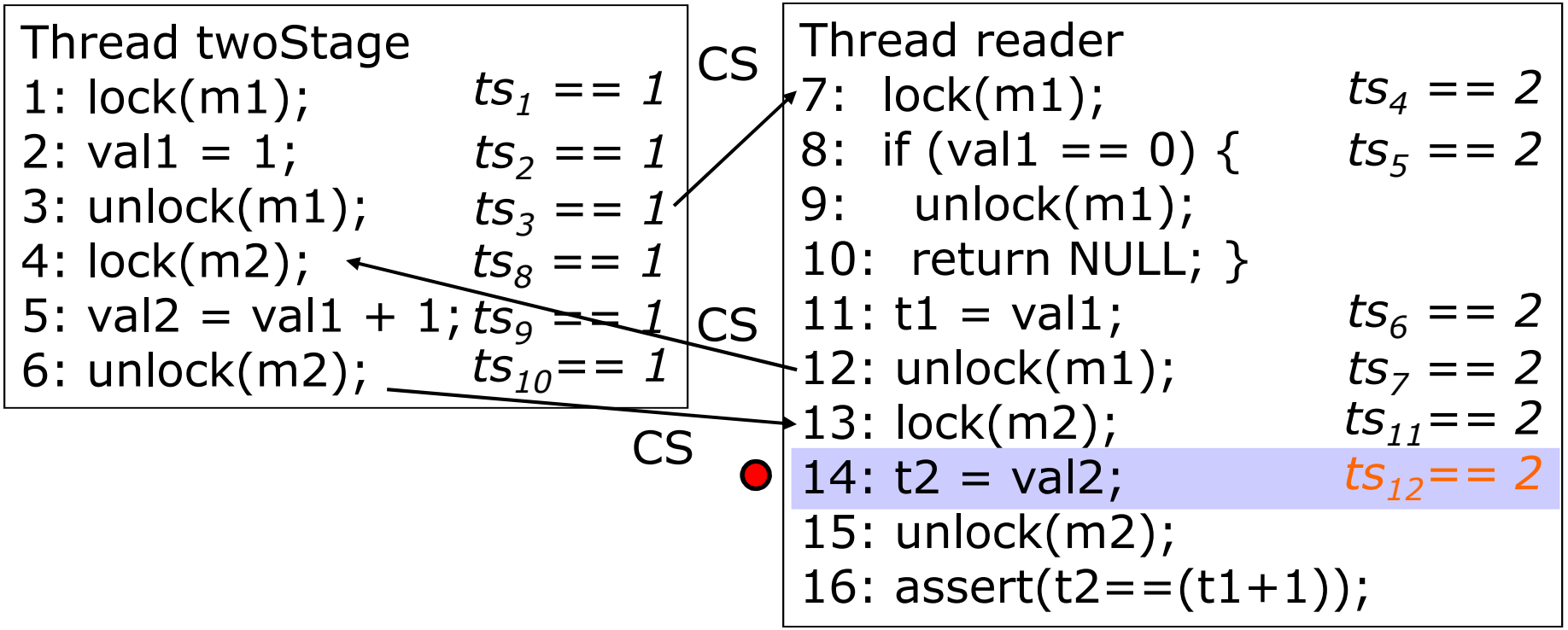
Schedule Recording: Interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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



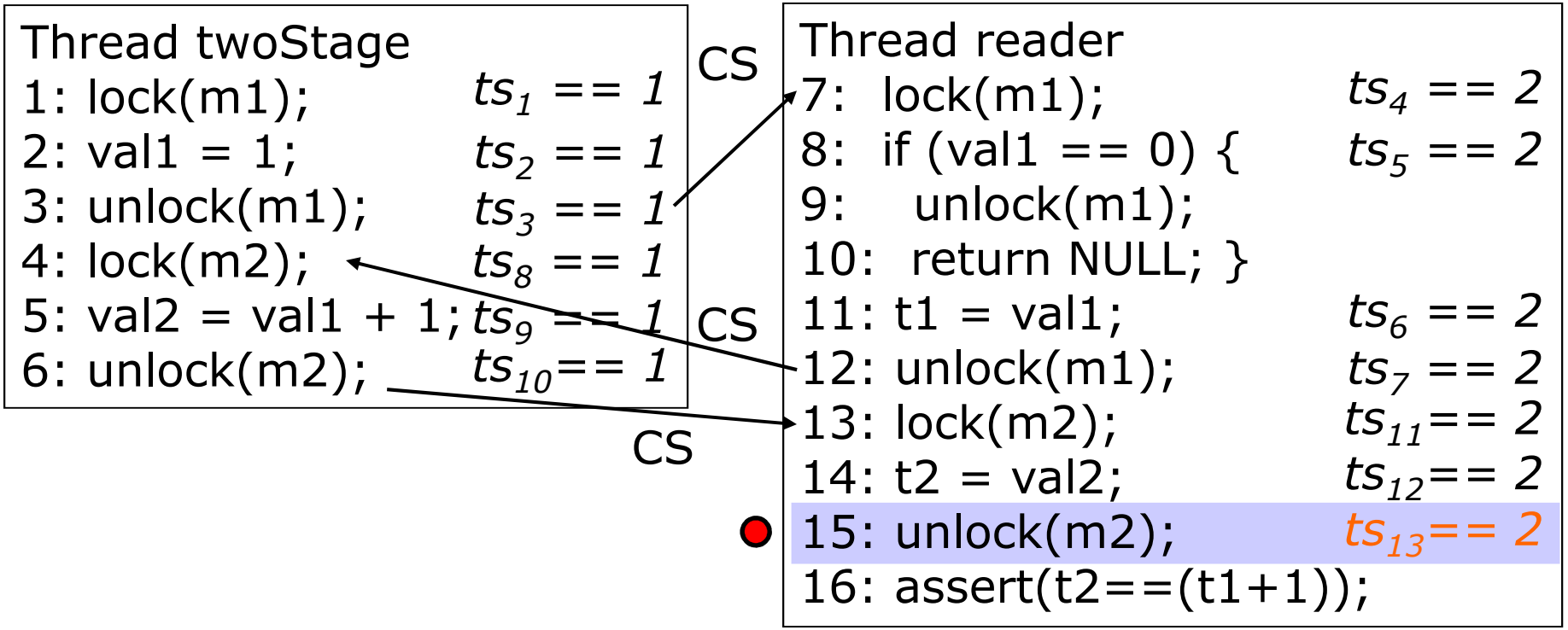
Schedule Recording: Interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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



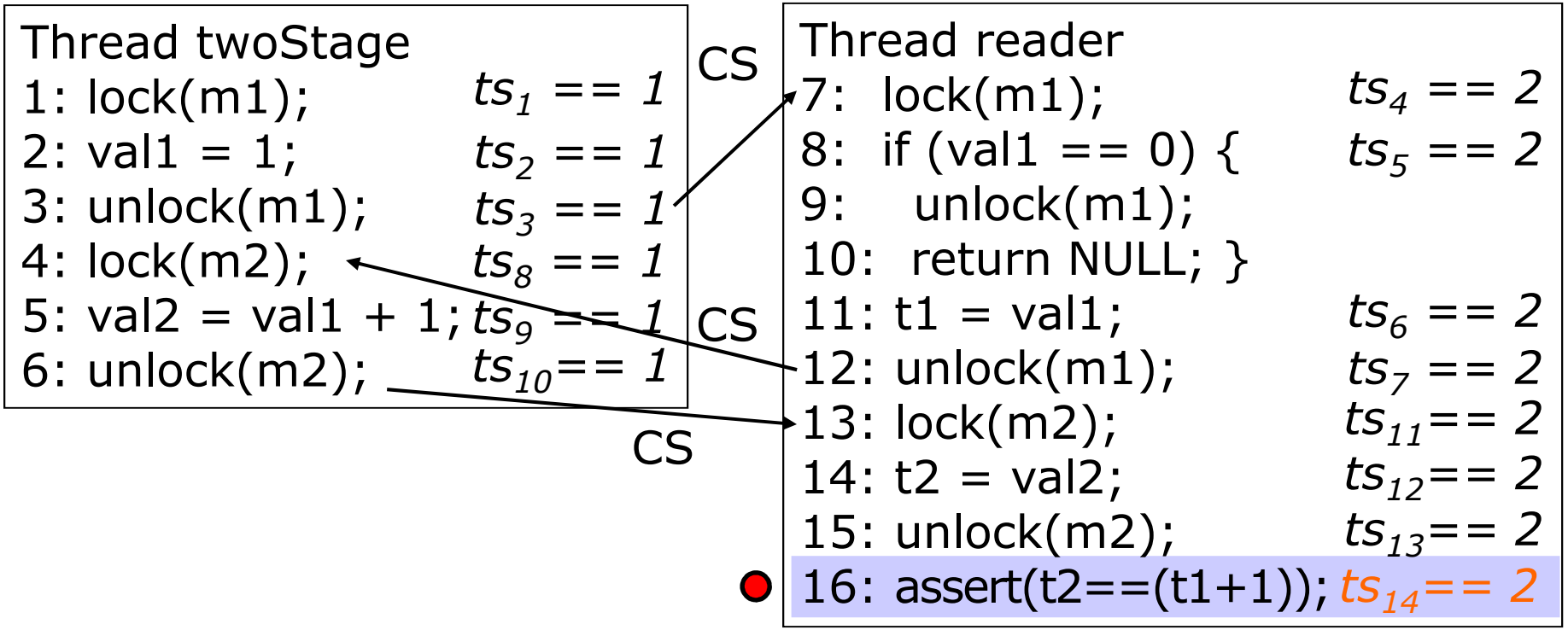
Schedule Recording: Interleaving I_s

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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



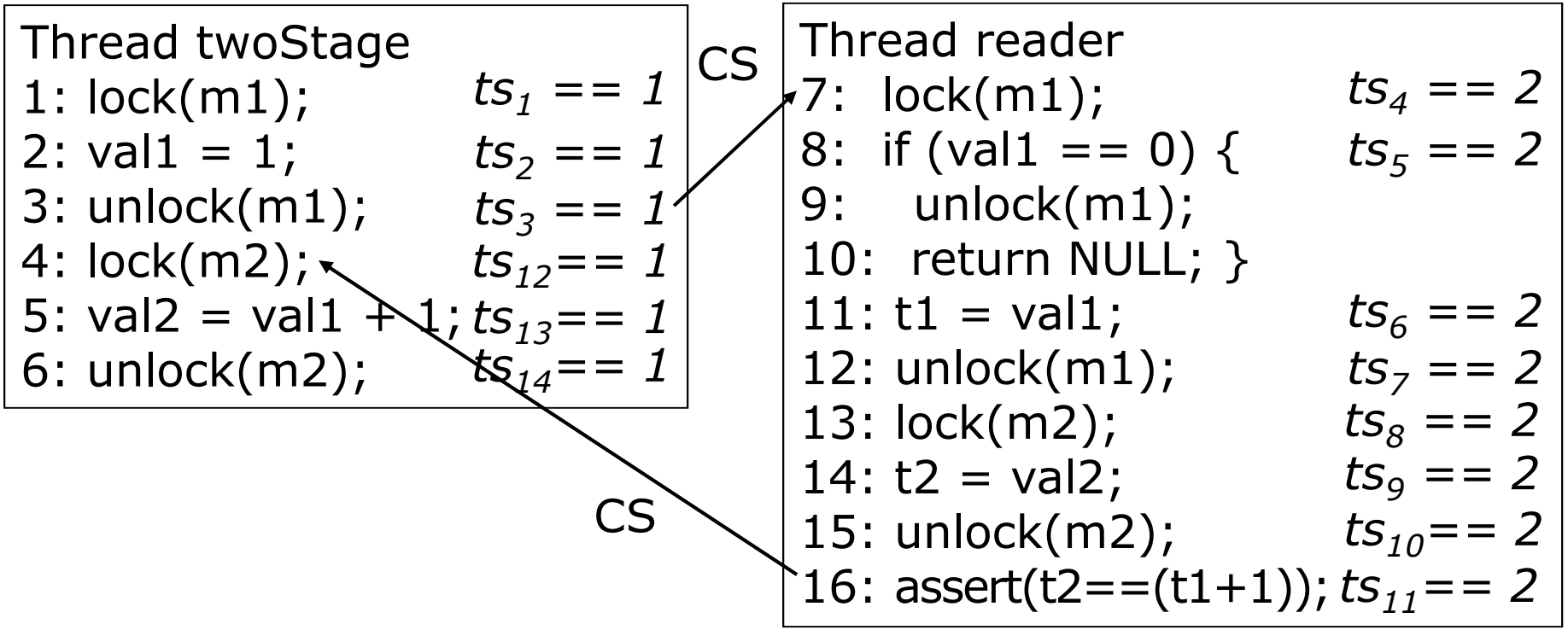
Schedule Recording: Interleaving I_f

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- 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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- 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

UNIVERSITY OF
Southampton
School of Electronics
and Computer Science

Evaluation

Comparison of the Approaches

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- Goal: compare efficiency of the proposed approaches
  - lazy exploration
  - schedule recording
  - under-approximation and widening
- Set-up:
  - ESBMC v1.6 together with the SMT solver Z3 v2.7
  - support the logics *QF_AUFBV* and *QF_AUFLIRA*
  - standard desktop PC, time-out 3600 seconds

About the benchmarks

UNIVERSITY OF Southampton
School of Electronics and Computer Science

	Module	#L	#P	k	#T	#C	
		81	47	7	26	2	Fran
2	fsbench_bad		48		27	2	Fra ipani file system with array o
3	in		1	1	13	4	ir to a hash table c
4	aget-0.4_bad	1233	1				aded download or
5	reorder_bad	84					a data race
6	twostage_bad	128	8	16	30	3	contains an atomicity violation
7	wronglock_bad	110	8	5	8	7	threads do not obtain the same lock instance
8	exStbHDMI_ok	1060	25	16	2	20	set various capabilities in the HDMI device
9	exStbLED_ok	425	45	10	2	-	front panel LED display
10	exStbThumb_bad	1109	243	2	2	1	Demonstrate how thumbnail images can be displayed and layer blending can be used

lines of code

number of properties checked

number of context switches

Number of threads

the number of BMC unrolling steps

About the benchmarks

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Inspect benchmark suite

	Module						Description
1	fsbench_ok					2	Frangipani file system
2	fsbench_bad		48	28	27	2	Frangipani file system with array out of bounds
3	indexer_ok	77	21	129	13	4	insert messages into a hash table concurrently
4	aget-0.4_bad	1233	170	257	3	2	Multi-threaded download accelerator
5	reorder_bad	84	16	11	20	3	contains a data race
6	twostage_bad	128	8	16	30	3	contains an atomicity violation
7	wronglock_bad	110	8	5	8	7	threads do not obtain the same lock instance
8	exStbHDMI_ok	1060	25	16	2	20	set various capabilities in the HDMI device
9	exStbLED_ok	425	45	10	2	-	front panel LED display
10	exStbThumb_bad	1109	243	2	2	1	Demonstrate how thumbnail images can be displayed and layer blending can be used

About the benchmarks

UNIVERSITY OF Southampton
School of Electronics and Computer Science

	Module	#L	#P	k	#T	#C	Description
1	fsbench_ok	81	47	27	26	2	Frangipani file system
2	fsbench_bad					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	12	170	257	3	2	Multi-threaded download accelerator
5	reorder_bad	84	16	11	20	3	contains a data race
6	twostage_bad	128	8	16	30	3	contains an atomicity violation
7	wronglock_bad	110	8	5	8	7	threads do not obtain the same lock instance
8	exStbHDMI_ok	1060	25	16	2	20	set various capabilities in the HDMI device
9	exStbLED_ok	425	45	10	2	-	front panel LED display
10	exStbThumb_bad	1109	243	2	2	1	Demonstrate how thumbnail images can be displayed and layer blending can be used

VV-lab benchmark suite

About the benchmarks

University of Southampton
School of Electronics and Computer Science

	Module	#L	#P	k	#T	#C	Description
1	fsbench_ok	81	47	27	26	2	Frangipani file system
2	fsbench_bad	80	48	28	27	2	Frangipani file system with array out of bounds
3	indexer_ok	77	21	129	13	4	insert messages into a hash table concurrently
4	aget-0.4_bad						Multi-threaded download celerator
5	reorder_bad						ntains a data race
6	twostage_bad						ntains an atomicity violation
7	wronglock_bad		8	5	8	7	threads do not obtain the same lock instance
8	exStbHDMI_ok	1060	25	16	2	20	set various capabilities in the HDMI device
9	exStbLED_ok	425	45	10	2	-	front panel LED display
10	exStbThumb_bad	1109	243	2	2	1	Demonstrate how thumbnail images can be displayed and layer blending can be used

Set-top box applications from NXP semiconductors

Comparison of the approaches (1)

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

	Lazy				UW						
	me	#P		#I		#P					
Module	Total	Fail	Error	#I/ #IF	Total	Fail	Error	Total	Fail	Error	Iter
fsben		1		729/		1	0	1770	1	0	2
fsben		0				0	0				1
indexe		0		0		0	0	234	0	0	1
aget-0.4_bad	248	1	0	2/1	254	1	0	245	1	0	1
reorder_bad	<1	1	0	32/20	<1	1	0	2	1	0	3
twostage_bad	3	1	0	41/15	2	1	0	7	1	0	4
wronglock_bad	84	1	0	20507/ 16	293	1	0	2089	1	0	3
exStbHDMI_ok	268	0	0	1/0	260	0	0	278	0	0	1
exStbLED_ok	900	0	0	11/0	MO	0	1	MO	0	1	1
exStbThumb_bad	14	1	0	3/1	MO	0	1	MO	0	1	1

Comparison of the approaches (1)

UNIVERSITY OF Southampton
School of Electronics and Computer Science

lazy encoding often more efficient than schedule recording and UW

	Lazy				Schedule			UW			
		#P		#I	Time	#P		Time	#P		
M			Error	#I/ #IF	Total	Fail	Error	Total	Fail	Error	Iter
fsbench_[illegible]	**1**	1	0	729/ 729	**452**	1	0	**1779**	1	0	2
fsbench_ok	**282**	0	0	676/0	**334**	0	0	**335**	0	0	1
indexer_ok	575	0	0	17160/ 0	238	0	0	234	0	0	1
aget-0.4_bad	248	1	0	2/1	254	1	0	245	1	0	1
reorder_bad	<1	1	0	32/20	<1	1	0	2	1	0	3
twostage_bad	3	1	0	41/15	2	1	0	7	1	0	4
wronglock_bad	**84**	1	0	20507/ 16	**293**	1	0	**2089**	1	0	3
exStbHDMI_ok	268	0	0	1/0	260	0	0	278	0	0	1
exStbLED_ok	**900**	0	0	11/0	**MO**	0	1	**MO**	0	1	1
exStbThumb_bad	**14**	1	0	3/1	**MO**	0	1	**MO**	0	1	1

Comparison of the approaches (2)

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

				#I	Schedule			UW			
					Time	#P		Time	#P		
N				#I/ #IF	Total	Fail	Error	Total	Fail	Error	Iter
fsbench_bad		1	0	729/ 729	452	1	0	1779	1	0	2
fsbench_ok	32	0	0	676/0	334	0	0	335	0	0	1
indexer_ok	575	0	0	17160/ 0	238	0	0	234	0	0	1
aget-0.4_bad	248	1	0	2/1	254	1	0	245	1	0	1
reorder_bad	<1	1	0	32/20	<1	1	0	2	1	0	3
twostage_bad	3	1	0	41/15	2	1	0	7	1	0	4
wronglock_bad	84	1	0	20507/ 16	293	1	0	2089	1	0	3
exStbHDMI_ok	268	0	0	1/0	260	0	0	278	0	0	1
exStbLED_ok	900	0	0	11/0	MO	0	1	MO	0	1	1
exStbThumb_bad	14	1	0	3/1	MO	0	1	MO	0	1	1

Comparison of the approaches (3)

UNIVERSITY OF Southampton
School of Electronics and Computer Science

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

	Lazy				Schedule			UW			
	T...	#P		#I	Time	#P		Time	#P		
Module	Total	Fail	Error	#I/ #IF	Total	Fail	Error	Total	Fail	Error	Iter
fsbench_bad	1	1	0	729/ 729	452	1	0	1779	1	0	2
fsbench_ok	282	0	0	676/0	334	0	0	335	0	0	1
indexer_ok	575	0	0	17160/ 0	238	0	0	234	0	0	1
aget-0.4_bad	248	1	0	2/1	254	1	0	245	1	0	1
reorder_bad	<1	1	0	32/20	<1	1	0	2	1	0	3
twostage_bad	3	1	0	41/15	2	1	0	7	1	0	4
wronglock_bad	84	1	0	20507/ 16	293	1	0	2089	1	0	3
exStbHDMI_ok	268	0	0	1/0	260	0	0	278	0	0	1
exStbLED_ok	900	0	0	11/0	MO	0	1	MO	0	1	1
exStbThumb_bad	14	1	0	3/1	MO	0	1	MO	0	1	1

Results

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- **lazy**, **schedule recording**, and **UW** algorithms
  - *lazy:* check constraints lazily is *fast for satisfiable instances*
  - *schedule recording:* the number of threads and context switches can grow quickly (and easily "blow-up" the model checker)
  - *UW:* memory overhead and slowdowns to extract the *unsat core*
- handle more than two threads, detect concurrency bugs (e.g., atomicity and order violations, deadlocks)
- **users.ecs.soton.ac.uk/lcc08r/esbmc/**

Future Work

- partial order reduction (static and dynamic)
- interpolants to prove no interference of context switches
- fault localization in multi-threaded C programs