On the Semantics of Snapshot Isolation

Azalea Raad Ori Lahav Viktor Vafeiadis

MPI-SWS
Tel Aviv University

Tuesday 15 January
VMCAI 2019
Lisbon, Portugal

azalea@mpi-sws.org SoundAndComplete.org @azalearaad

What is STM?

Concurrency Control via ***transactions***

- ***atomic*** unit of work (set of operations) on shared data
- ***all-or-nothing***

```
// x = y = 0

T: [ x := 1;
     y := 1;

// x = y = 0     OR     x = y = 1
```

Which STM?

Strong consistency - inefficient

- serialisability
- strict serialisability
- …

Weak consistency

- parallel snapshot isolation (PSI)
- snapshot isolation (SI)
- …

Which STM?

Strong consistency - inefficient

- serialisability
- strict serialisability
- …

Weak consistency

- parallel snapshot isolation (PSI)
  - our earlier ESOP paper
- **snapshot isolation (SI)**
  - efficient, popular ⇒ this talk

Which STM?

Strong consistency - inefficient

- serialisability
- strict serialisability
- …

Weak consistency

- parallel snapshot isolation (PSI)
  - our earlier ESOP paper
- **snapshot isolation (SI)**
  - efficient, popular ⇒ this talk

STM Context

- Shared memory setting

  (with ***weak*** memory consistency)

- ***Mixed*** accesses to shared data

  (transactional and non-transactional)

- ***Cannot instrument*** non-transactional accesses

  (weak isolation)

SI STM Desiderata

- *Declarative* semantics

- Lock-based reference implementation (*operational* semantics)
  - ➡ ***Sound***: Behaviours(imp) ⊆ Behaviours(spec)
  - ➡ ***Complete***: Behaviours(spec) ⊆ Behaviours(imp)

SI STM Desiderata

- *Declarative* semantics

- Lock-based reference implementation (*operational* semantics)
  - ➡ ***Sound***: Behaviours(imp) ⊆ Behaviours(spec)
  - ➡ ***Complete***: Behaviours(spec) ⊆ Behaviours(imp)

- *Declarative* semantics with ***mixed*** accesses

- Reference implementation with ***mixed*** accesses (*operational* semantics)
  - ➡ ***Sound***: Behaviours(imp) ⊆ Behaviours(spec)
  - ➡ ***Complete***: Behaviours(spec) ⊆ Behaviours(imp)

SI STM Desiderata: State of Art

✓ *Declarative* semantics

✘ Lock-based reference implementation (*operational* semantics)

- ➡ ***Sound***: Behaviours(imp) ⊆ Behaviours(spec)
- ➡ ***Complete***: Behaviours(spec) ⊆ Behaviours(imp)

✘ *Declarative* semantics with ***mixed*** accesses

✘ Reference implementation with ***mixed*** accesses (operational semantics)

- ➡ ***Sound***: Behaviours(imp) ⊆ Behaviours(spec)
- ➡ ***Complete***: Behaviours(spec) ⊆ Behaviours(imp)

SI STM Desiderata: Our Work

✓ *Declarative* semantics

✓ Lock-based reference implementation (*operational* semantics)

- ➡ ***Sound***: Behaviours(imp) ⊆ Behaviours(spec)
- ➡ ***Complete***: Behaviours(spec) ⊆ Behaviours(imp)

✓ *Declarative* semantics with ***mixed*** accesses ⟹ ***RSI*** (‘robust’ SI)

✓ Reference implementation with ***mixed*** accesses (operational semantics)

- ➡ ***Sound***: Behaviours(imp) ⊆ Behaviours(spec)
- ➡ ***Complete***: Behaviours(spec) ⊆ Behaviours(imp)

SI STM Desiderata: Our Work

✓ *Declarative* semantics

✓ Lock-based reference implementation (*operational* semantics)

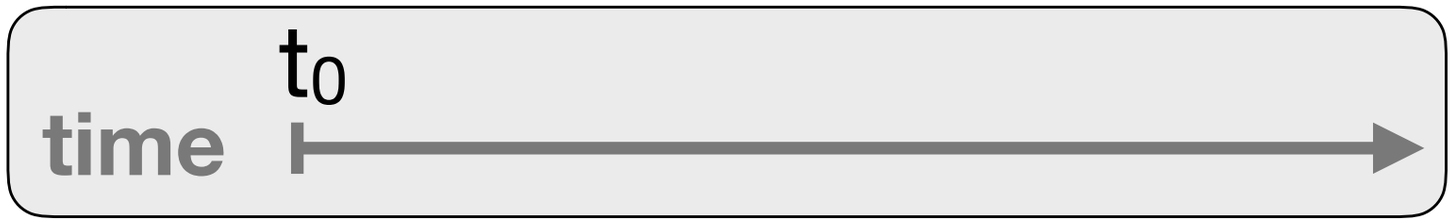
- ➡ ***Sound***: Behaviours(imp) ⊆ Behaviours(spec)
- ➡ ***Complete***: Behaviours(spec) ⊆ Behaviours(imp)

✓ *Declarative* semantics with ***mixed*** accesses ⟹ ***RSI*** (‘robust’ SI)

✓ Reference implementation with ***mixed*** accesses (operational semantics)

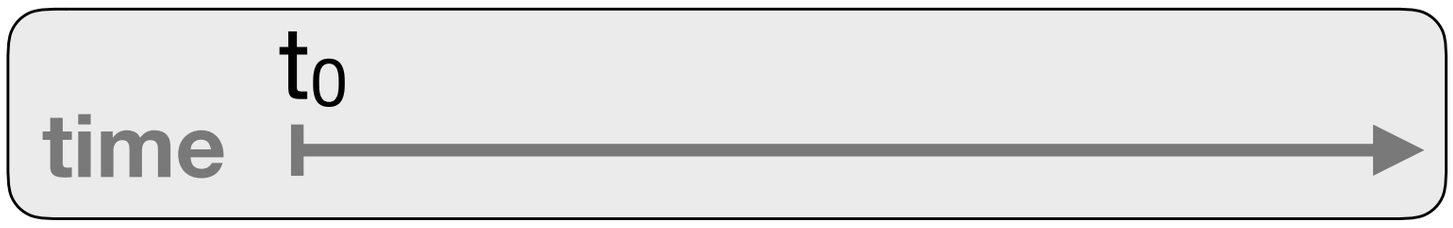
- ➡ ***Sound***: Behaviours(imp) ⊆ Behaviours(spec)
- ➡ ***Complete***: Behaviours(spec) ⊆ Behaviours(imp)

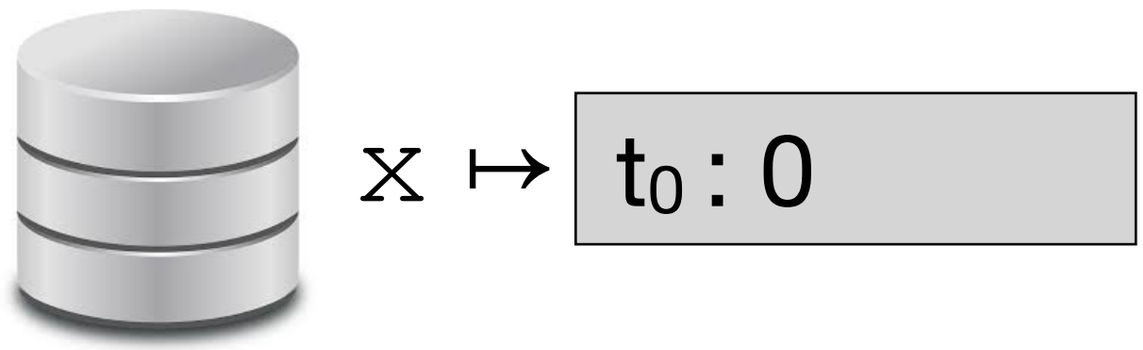
What is SI?



$x \mapsto t_0 : 0$

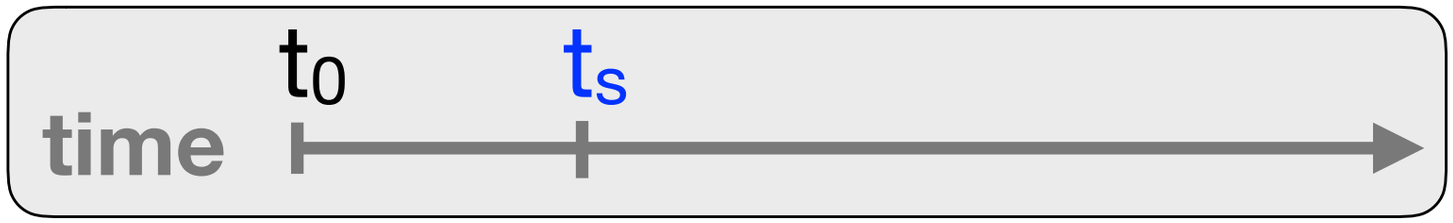
What is SI?

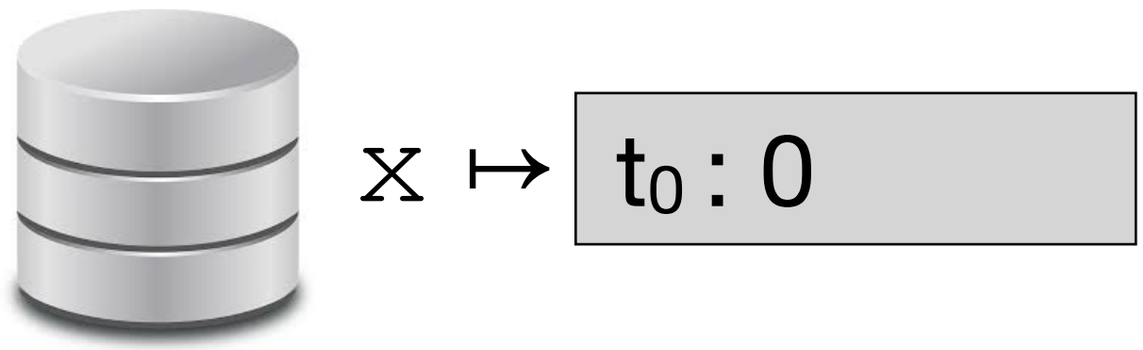





```
[ x := 1;
```

What is SI?

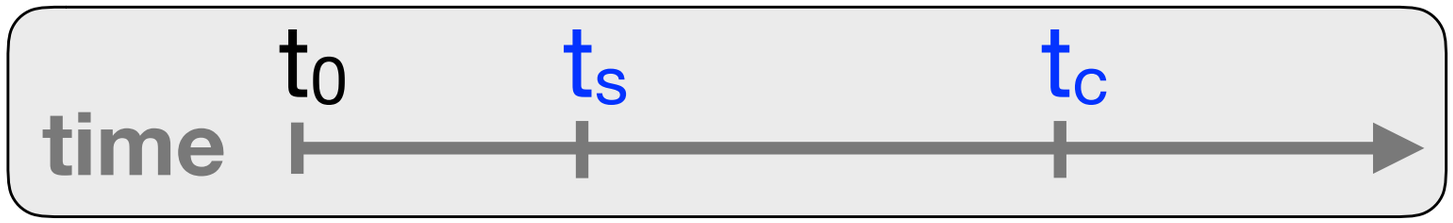


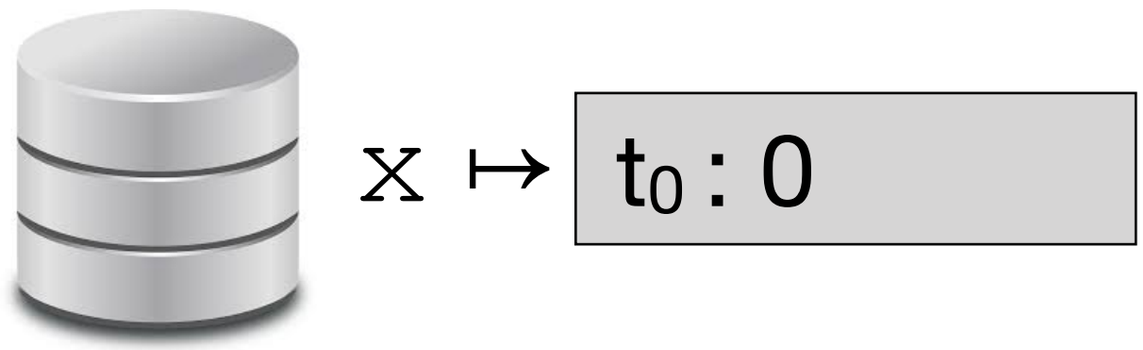


t_s: **S** ≜ x=0

[x := 1;

What is SI?



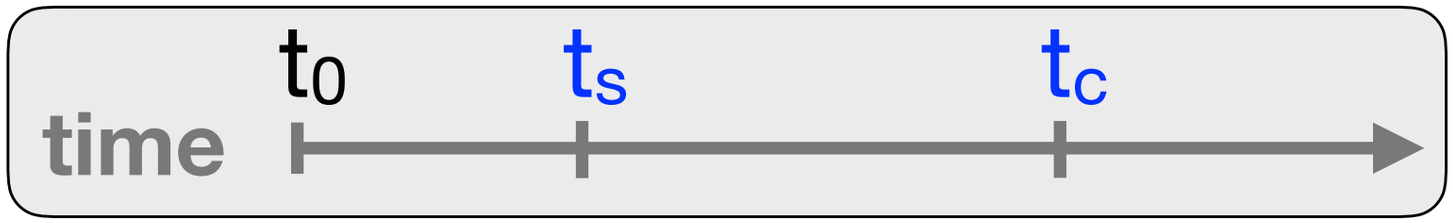


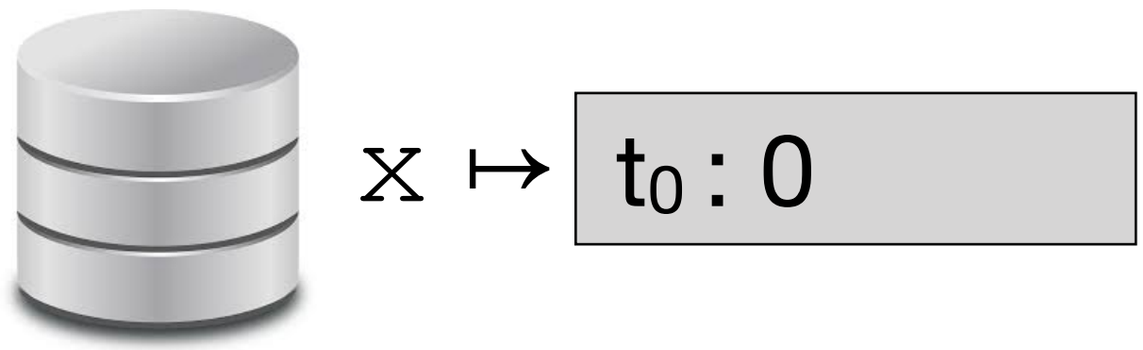
t_s: **S** ≜ x=0

[x := 1;

t_c: **C** ≜ x=1

What is SI?



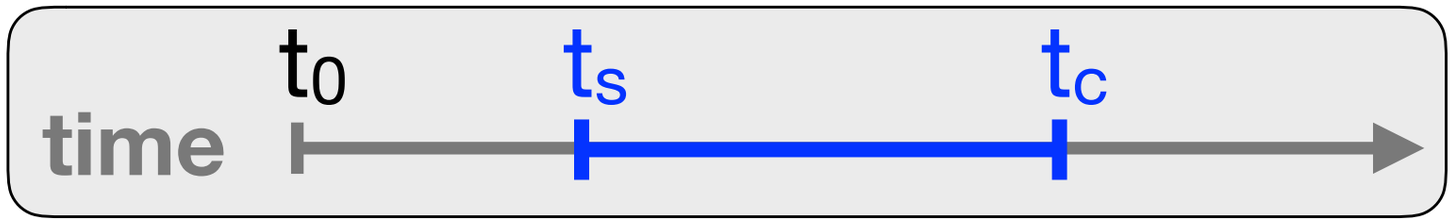


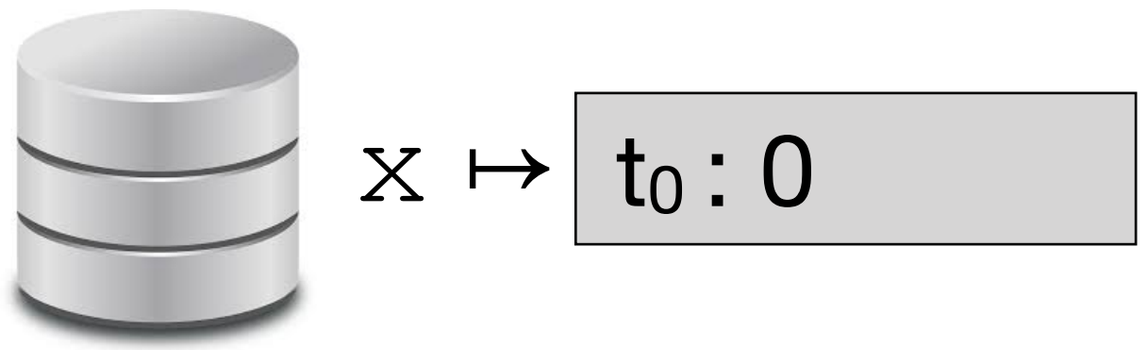
t_s : **S** ≜ x=0
[x := 1;
t_c : **C** ≜ x=1

ww-conflict(**C**,t_s,t_c)?

ww-conflict(**C**,t_s,t_c): $\exists x \in \mathbf{C}$. a committed trans. updates x during [t_s,t_c]

What is SI?



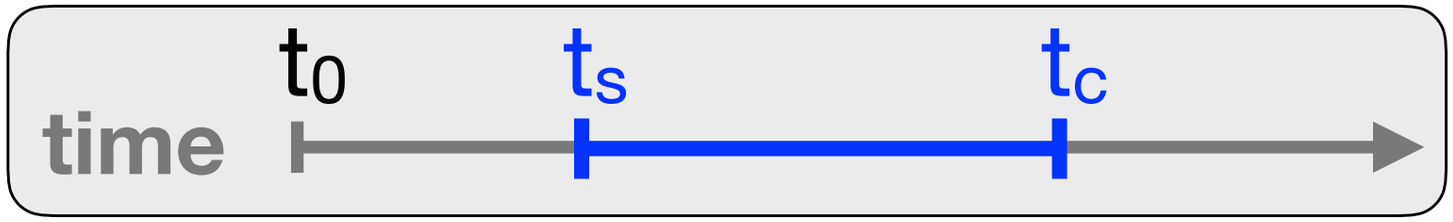


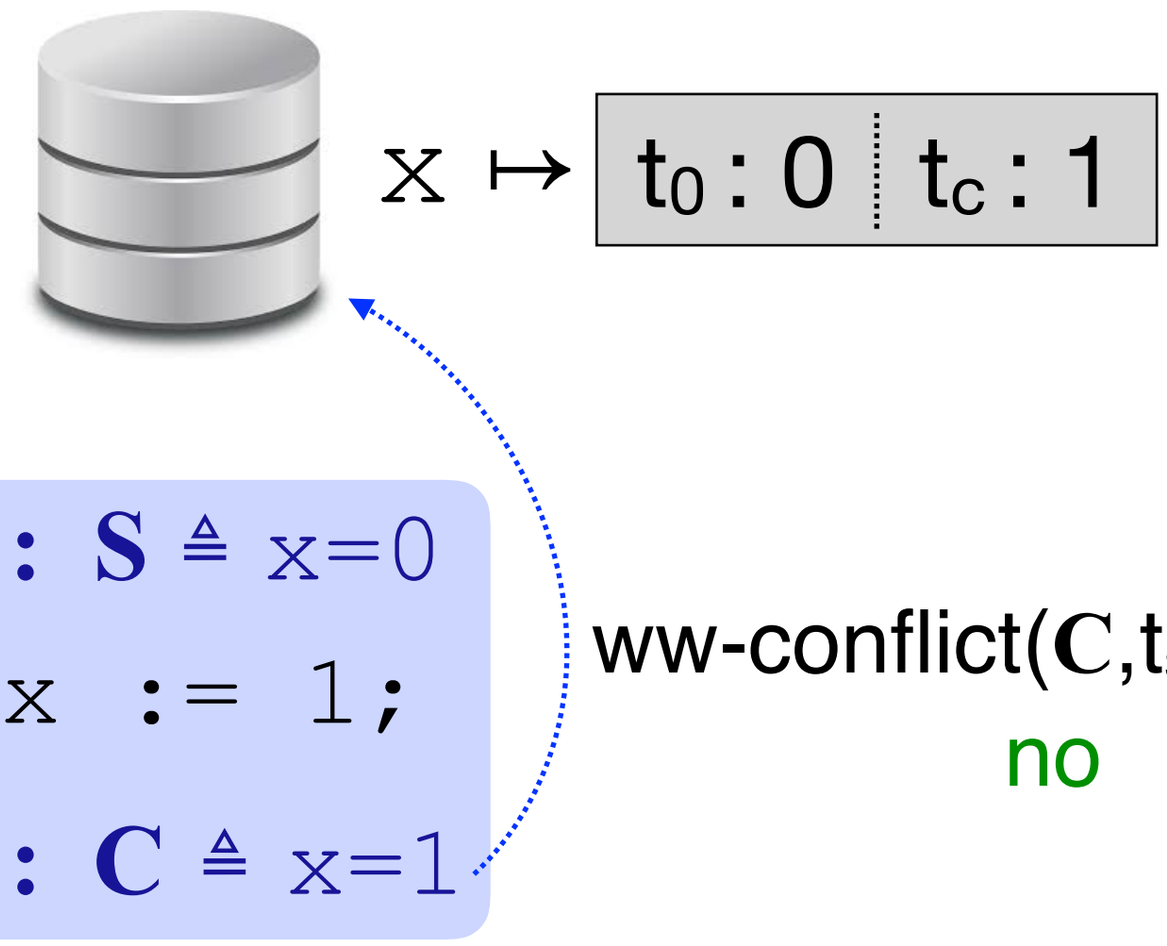
t_s : **S** ≜ x=0
[x := 1;
t_c : **C** ≜ x=1

ww-conflict(**C**,t_s,t_c)?
no

ww-conflict(**C**,t_s,t_c): ∃x ∈ **C**. a committed trans. updates x during [t_s,t_c]

What is SI?





ww-conflict($\mathbf{C}$,t_s,t_c): $\exists x \in \mathbf{C}$. a committed trans. updates x during [t_s,t_c]

SI: ***Allowed*** Anomalies

write skew (WS) / store buffering (SB)

```
T1 : [ x := 1;          ||  T2 : [ y := 1;
       a := y; // 0     ||         b := x; // 0
```

// v denotes that v was read

SI: ***Allowed*** Anomalies

write skew (WS) / store buffering (SB)

```
T1 : [ x := 1;        ||  T2 : [ y := 1;
       a := y; // 0   ||         b := x; // 0
```

// v denotes that v was read

Not allowed under ***serialisability***!

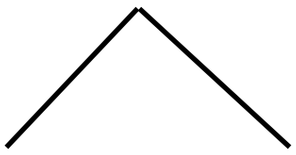
SI: ***Allowed*** Anomalies

write skew (WS) / store buffering (SB)

```
T1 : [ x := 1;        ||  T2 : [ y := 1;
       a := y; // 0   ||         b := x; // 0
```

// v denotes that v was read

Not allowed under ***serialisability***!



T1 executes before T2

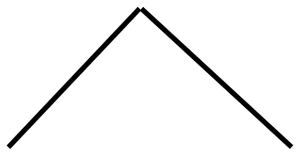
SI: ***Allowed*** Anomalies

write skew (WS) / store buffering (SB)

```
T1 : [ x := 1;          ||  T2 : [ y := 1;
       a := y; // 0     ||         b := x; // 0
```

// v denotes that v was read

Not allowed under ***serialisability***!



T1 executes before T2

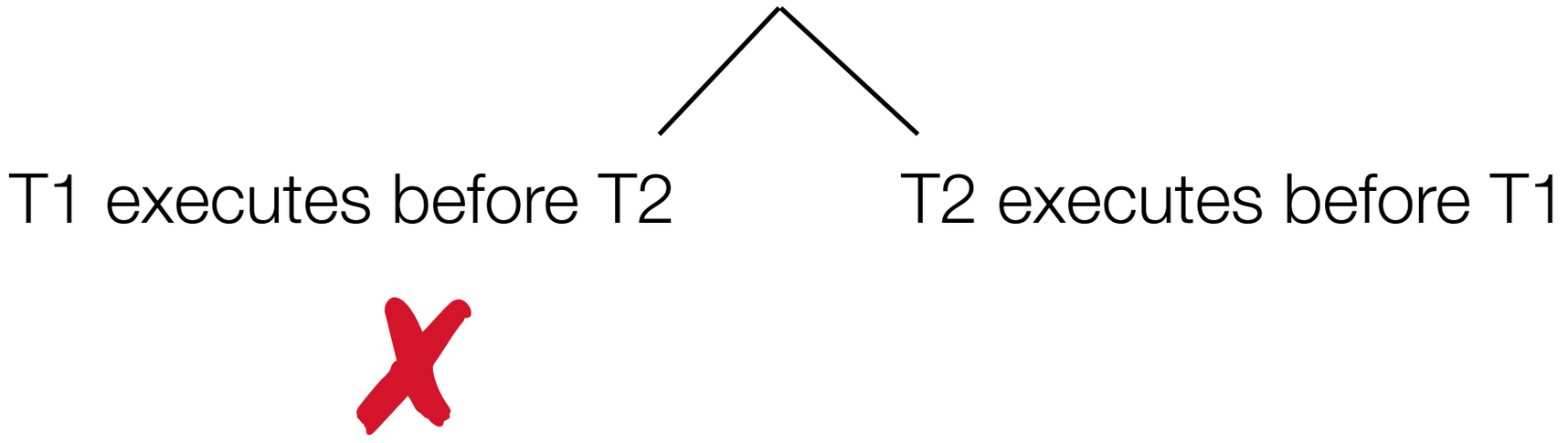
SI: ***Allowed*** Anomalies

write skew (WS) / store buffering (SB)

```
T1 : [ x := 1;        ||  T2 : [ y := 1;
       a := y; // 0   ||         b := x; // 0
```

// v denotes that v was read

Not allowed under ***serialisability***!



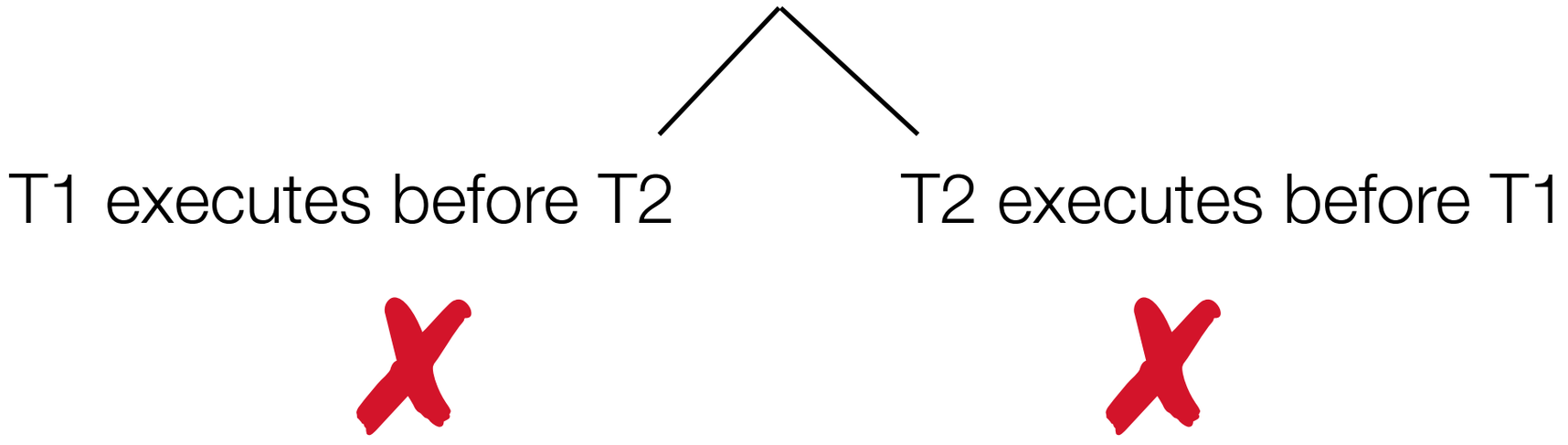
SI: ***Allowed*** Anomalies

write skew (WS) / store buffering (SB)

```
T1 : [ x := 1;        ||  T2 : [ y := 1;
       a := y; // 0 ✗   ||       b := x; // 0
```

// v denotes that v was read

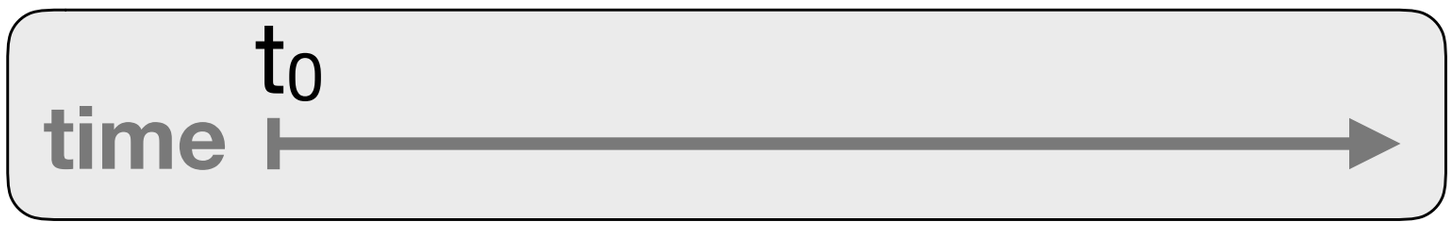
Not allowed under ***serialisability***!



SI: ***Allowed*** Anomalies

write skew (WS) / store buffering (SB)

```
T1 : [ x := 1;      ||  T2 : [ y := 1;
       a := y; // 0 ||         b := x; // 0
```



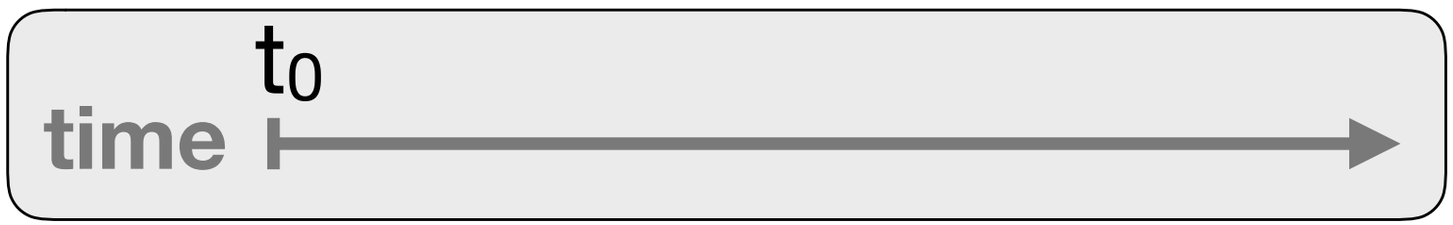
$x \mapsto$ $t_0 : 0$

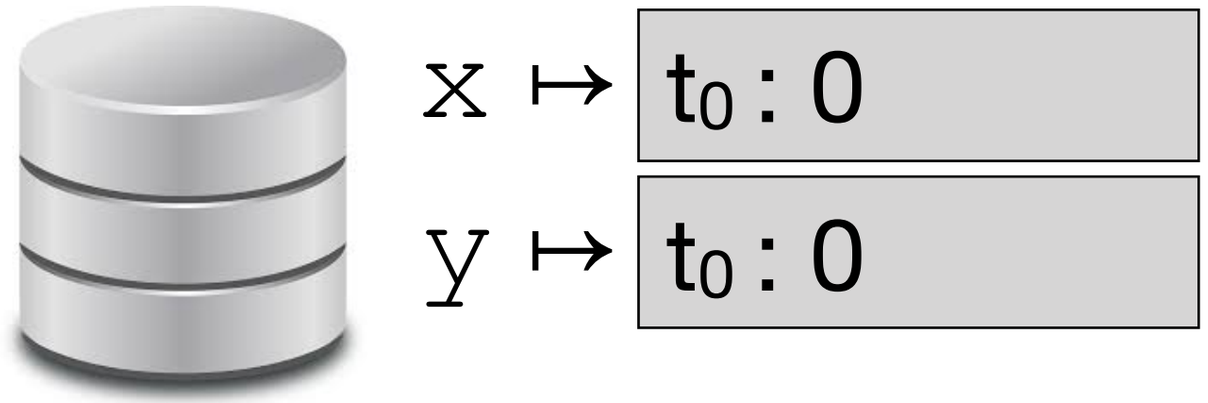
$y \mapsto$ $t_0 : 0$

SI: ***Allowed*** Anomalies

write skew (WS) / store buffering (SB)

T1 :
```
x := 1;
a := y; // 0
```
‖ **T2 :**
```
y := 1;
b := x; // 0
```



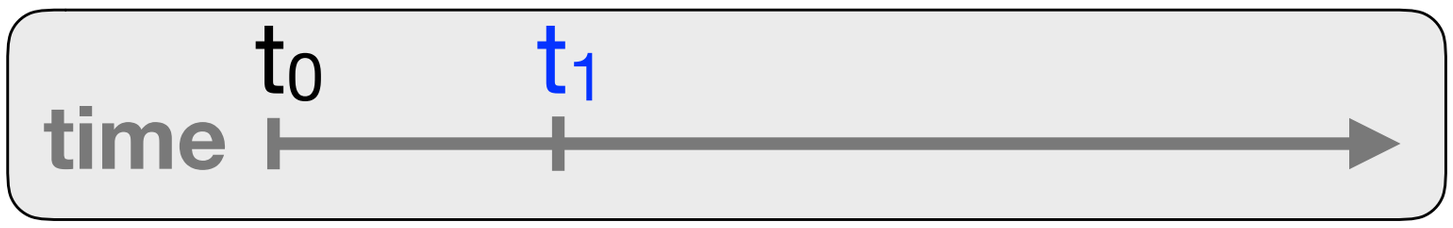


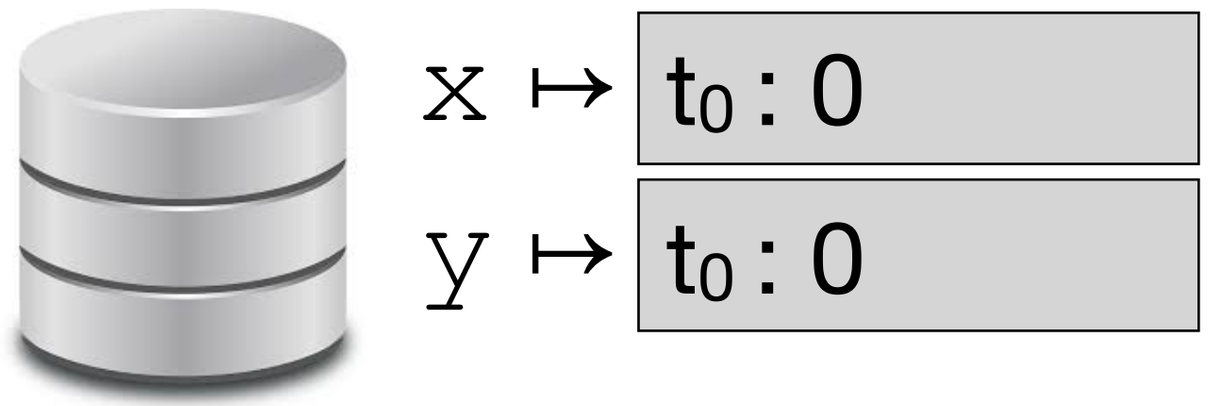

```
x := 1;
a := y;
```

SI: ***Allowed*** Anomalies

write skew (WS) / store buffering (SB)

```
T1 : [ x := 1;        ||  T2 : [ y := 1;
       a := y; //0    ||         b := x; //0
```





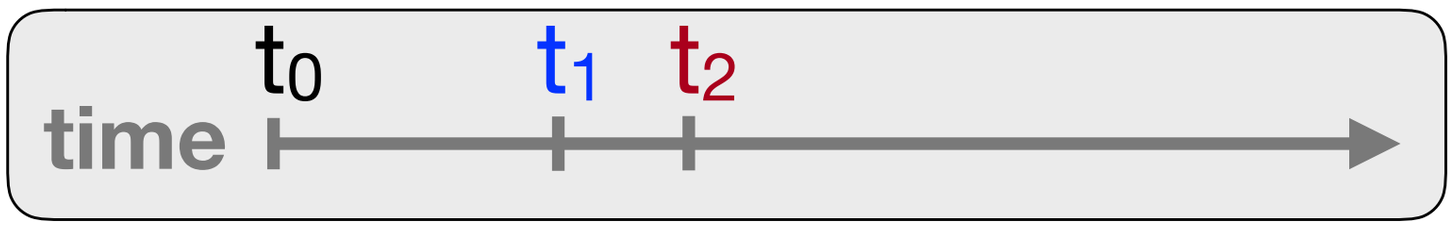
t_1 : **S1:** x=y=0

```
[ x := 1;
  a := y; //0
```

SI: ***Allowed*** Anomalies

write skew (WS) / store buffering (SB)

T1 :
```
x := 1;
a := y; // 0
```
‖ **T2 :**
```
y := 1;
b := x; // 0
```



$x \mapsto$ $t_0 : 0$

$y \mapsto$ $t_0 : 0$

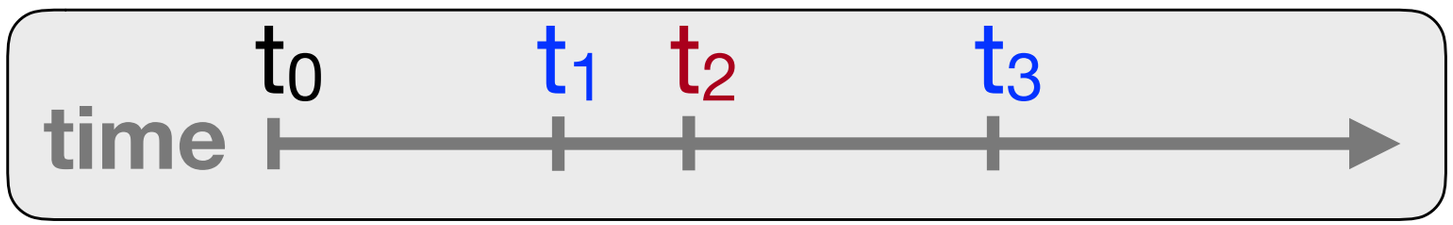
t_1 : S1: x=y=0
```
x := 1;
a := y; // 0
```

t_2 : S2: x=y=0
```
y := 1;
b := x; // 0
```

SI: ***Allowed*** Anomalies

write skew (WS) / store buffering (SB)

T1 :
```
x := 1;
a := y; // 0
```
‖ **T2 :**
```
y := 1;
b := x; // 0
```



$x \mapsto$ $t_0 : 0$

$y \mapsto$ $t_0 : 0$

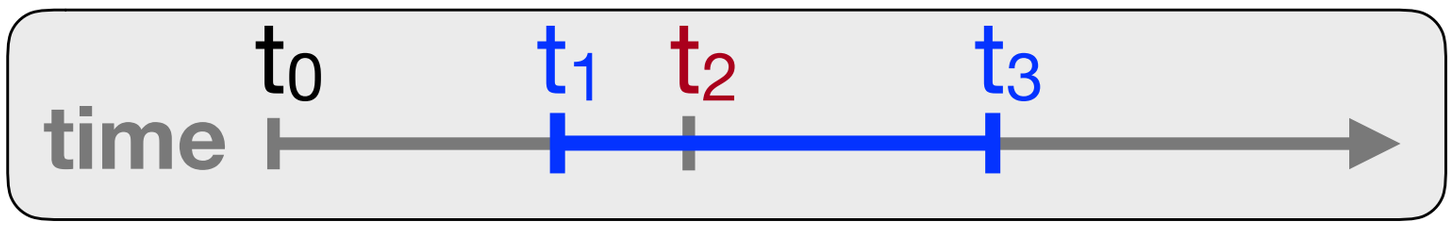
t_1 : **S1:** x=y=0
```
x := 1;
a := y; // 0
```
t_3 : **C1:** x=1

t_2 : **S2:** x=y=0
```
y := 1;
b := x; // 0
```

SI: ***Allowed*** Anomalies

write skew (WS) / store buffering (SB)

```
T1: [ x := 1;        ||  T2: [ y := 1;
      a := y; // 0   ||        b := x; // 0
```



x ↦ t_0 : 0

y ↦ t_0 : 0

t_1 : **S1:** x=y=0

```
[ x := 1;
  a := y; // 0
```

t_3 : **C1:** x=1

ww-conflict? no

t_2 : **S2:** x=y=0

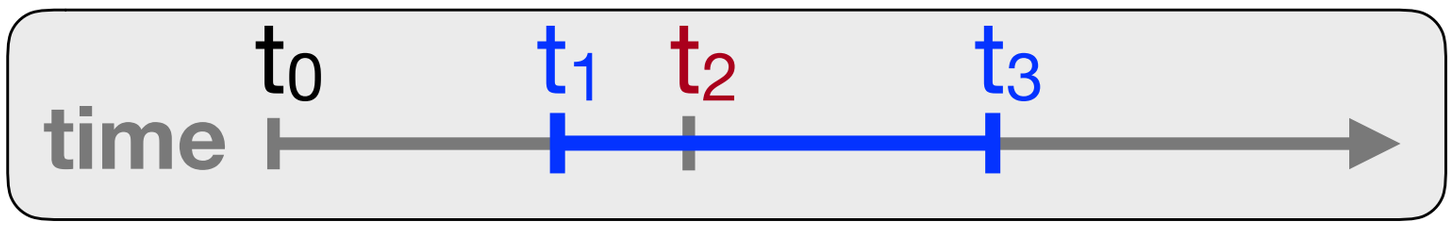
```
[ y := 1;
  b := x; // 0
```

ww-conflict(**C**,t_s,t_c): $\exists x \in \mathbf{C}$. a committed trans. updates x during [t_s,t_c]

SI: ***Allowed*** Anomalies

write skew (WS) / store buffering (SB)

```
T1 : [ x := 1;        ||  T2 : [ y := 1;
       a := y; // 0   ||         b := x; // 0
```



x ↦ | t0 : 0 | t3 : 1 |

y ↦ | t0 : 0 |

t1 : **S1:** x=y=0

```
[ x := 1;
  a := y; // 0
```

t3 : **C1:** x=1

ww-conflict? no

t2 : **S2:** x=y=0

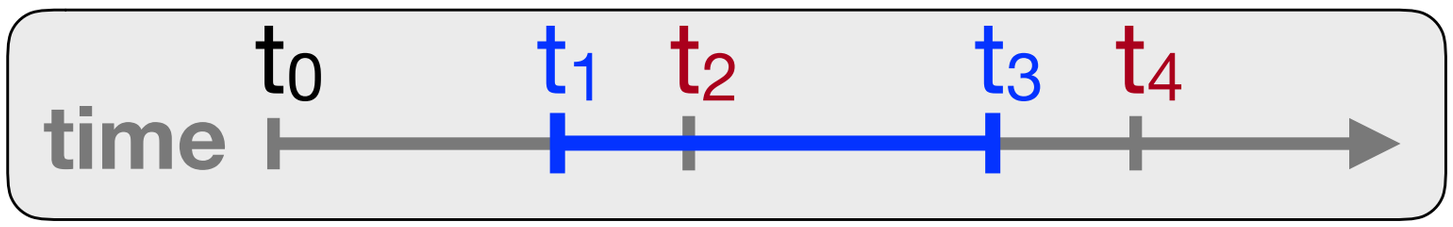
```
[ y := 1;
  b := x; // 0
```

ww-conflict(**C**,t_s,t_c): $\exists x \in \mathbf{C}$. a committed trans. updates x during [t_s,t_c]

SI: ***Allowed*** Anomalies

write skew (WS) / store buffering (SB)

T1 :
```
x := 1;
a := y; // 0
```
‖
T2 :
```
y := 1;
b := x; // 0
```



x ↦ | $t_0 : 0$ | $t_3 : 1$ |

y ↦ | $t_0 : 0$ |

t_1 : **S1:** x=y=0
```
x := 1;
a := y; // 0
```
t_3 : **C1:** x=1

ww-conflict? no

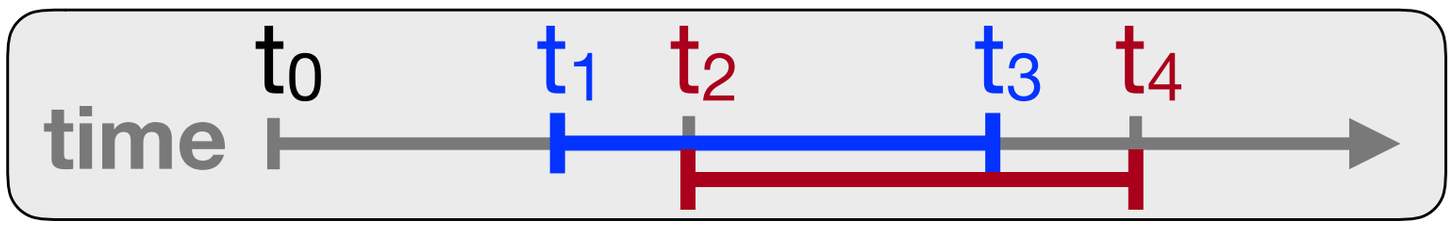
t_2 : **S2:** x=y=0
```
y := 1;
b := x; // 0
```
t_4 : **C2:** y=1

ww-conflict(**C**,t_s,t_c): $\exists x \in \mathbf{C}$. a committed trans. updates x during [t_s,t_c]

SI: ***Allowed*** Anomalies

write skew (WS) / store buffering (SB)

T1 :
```
x := 1;
a := y; // 0
```
‖ **T2 :**
```
y := 1;
b := x; // 0
```



x ↦ | $t_0 : 0$ | $t_3 : 1$ |

y ↦ | $t_0 : 0$ |

t_1 : **S1:** x=y=0
```
x := 1;
a := y; // 0
```
t_3 : **C1:** x=1

ww-conflict? no

ww-conflict? no

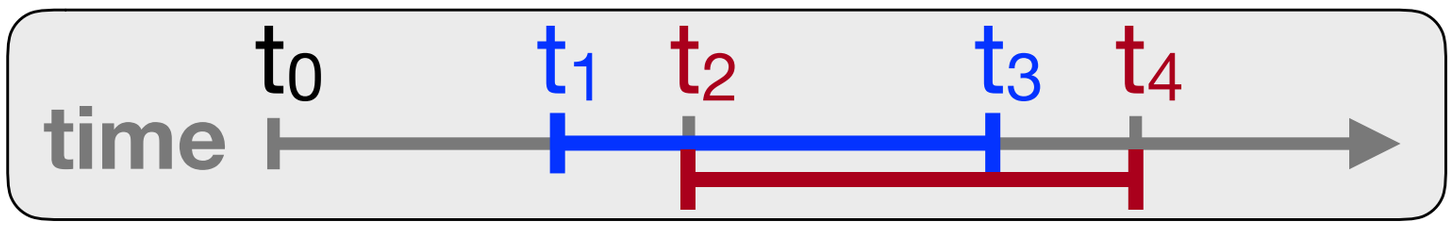
t_2 : **S2:** x=y=0
```
y := 1;
b := x; // 0
```
t_4 : **C2:** y=1

ww-conflict(**C**,t_s,t_c): $\exists x \in \mathbf{C}$. a committed trans. updates x during [t_s,t_c]

SI: ***Allowed*** Anomalies

write skew (WS) / store buffering (SB)

T1 :
```
x := 1;
a := y; // 0
```
‖ **T2 :**
```
y := 1;
b := x; // 0
```



$x \mapsto$ | $t_0 : 0$ | $t_3 : 1$

$y \mapsto$ | $t_0 : 0$ | $t_4 : 1$

t_1 : **S1:** x=y=0
```
x := 1;
a := y; // 0
```
t_3 : **C1:** x=1

ww-conflict? no

ww-conflict? no

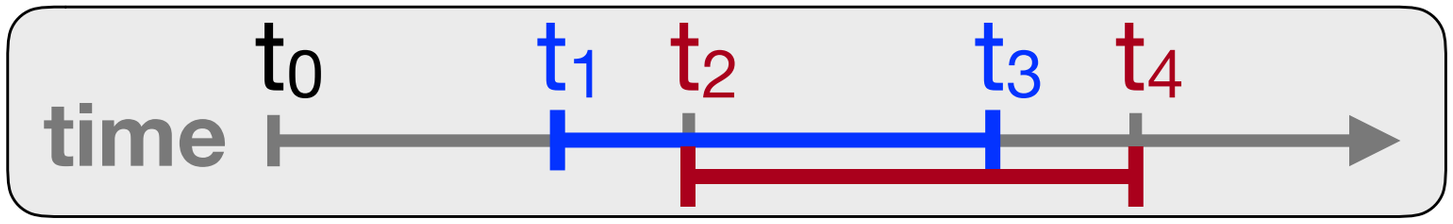
t_2 : **S2:** x=y=0
```
y := 1;
b := x; // 0
```
t_4 : **C2:** y=1

ww-conflict(**C**,t_s,t_c): $\exists x \in \mathbf{C}$. a committed trans. updates x during $[t_s,t_c]$

SI: ***Allowed*** Anomalies

write skew (WS) / store buffering (SB)

T1 :
```
x := 1;
a := y; // 0
```
‖ **T2 :**
```
y := 1;
b := x; // 0
```



Interval Order used to detect conflicts:

if: two intervals are ***unordered*** (they overlap)
then: they have ***no write*** on the ***same location***

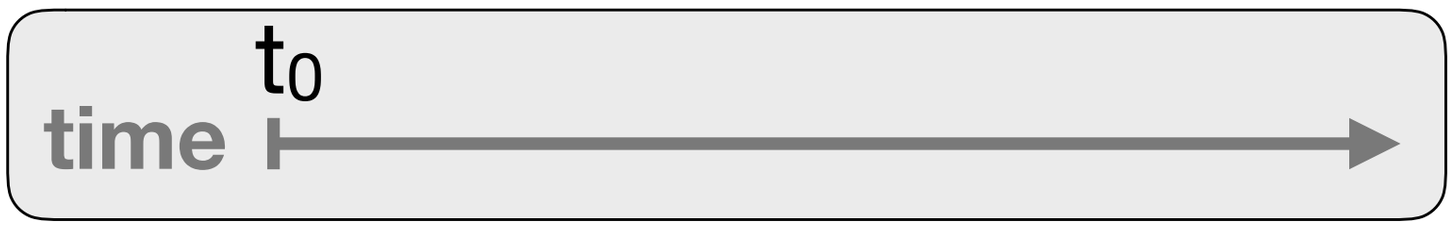
t_3 : **C1:** x=1 — ww-conflict? no — ww-conflict? no — t_4 : **C2:** y=1

ww-conflict($\mathbf{C}$,t_s,t_c): $\exists x \in \mathbf{C}$. a committed trans. updates x during $[t_s,t_c]$

SI: ***Disallowed*** Anomalies

lost update (LU)

```
T1: [ a := x; //0          ||  T2: [ b := x; //0
    [ x := a+1;                    [ x := b+1;
```

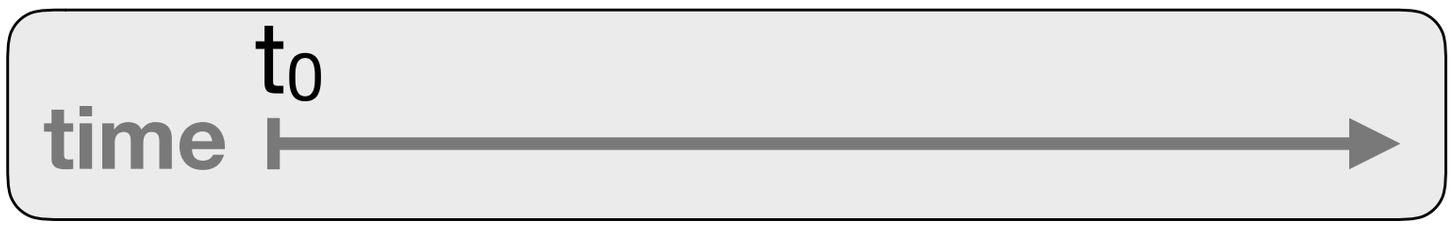


$x \mapsto$ $t_0 : 0$

SI: ***Disallowed*** Anomalies

lost update (LU)

T1:
```
a := x; //0
x := a+1;
```
‖ **T2:**
```
b := x; //0
x := b+1;
```



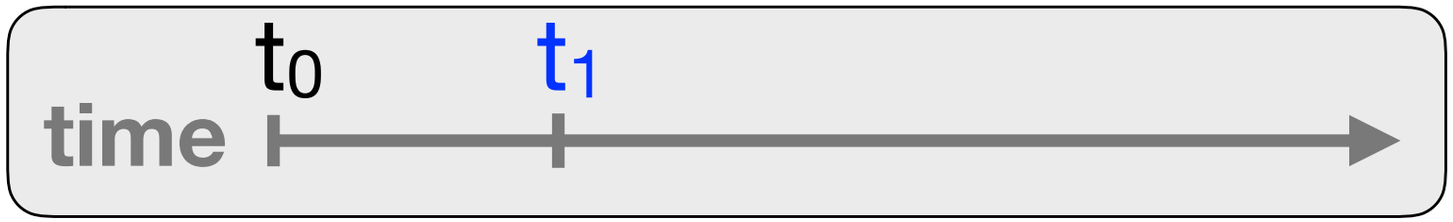
$x \mapsto$ $t_0 : 0$

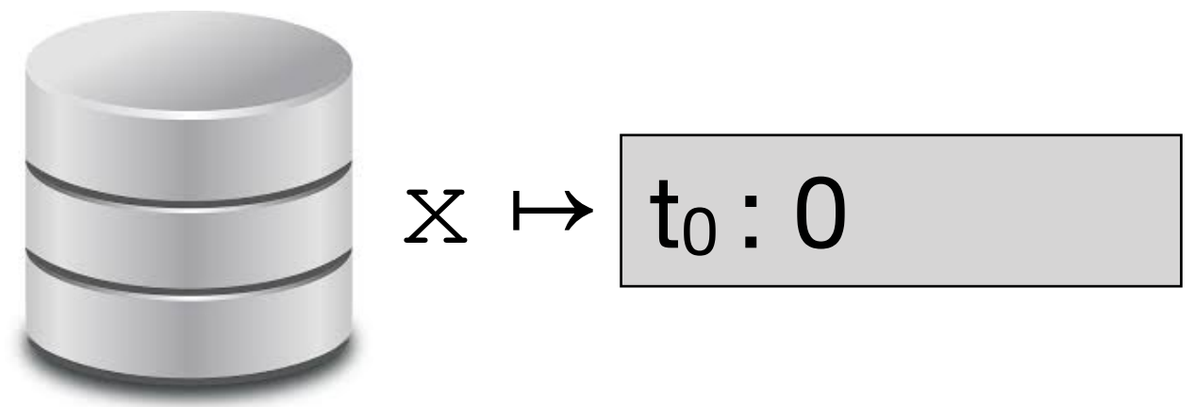
```
a := x;
x := a+1;
```

SI: ***Disallowed*** Anomalies

lost update (LU)

```
T1: [ a := x; //0          ||  T2: [ b := x; //0
      x := a+1;                      x := b+1;
```





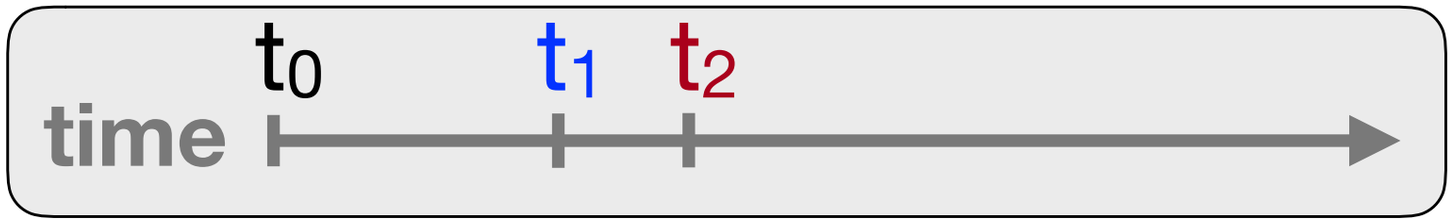
t_1 : **S1:** `x=0`

```
[ a := x; //0
  x := a+1;
```

SI: ***Disallowed*** Anomalies

lost update (LU)

```
T1: [ a := x; //0          ||  T2: [ b := x; //0
      x := a+1;                      x := b+1;
```



$x \mapsto$ $t_0 : 0$

t_1 : **S1:** x=0

```
[ a := x; //0
  x := a+1;
```

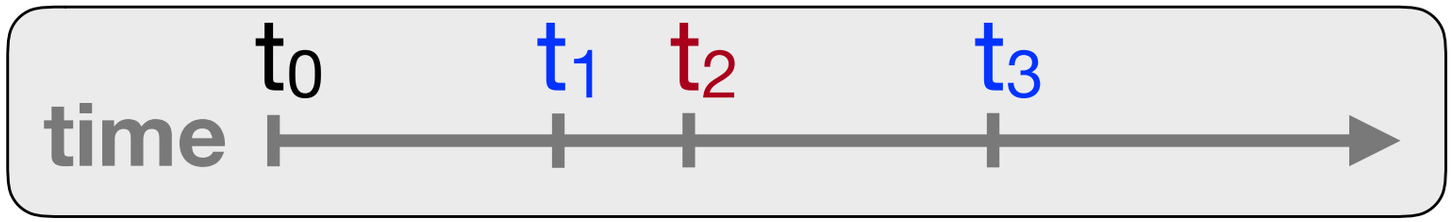
t_2 : **S2:** x=0

```
[ a := x; //0
  x := a+1;
```

SI: ***Disallowed*** Anomalies

lost update (LU)

```
T1: [ a := x; //0          ||  T2: [ b := x; //0
      x := a+1;                      x := b+1;
```



$x \mapsto t_0 : 0$

t_1 : **S1:** x=0

```
[ a := x; //0
  x := a+1;
```

t_3 : **C1:** x=1

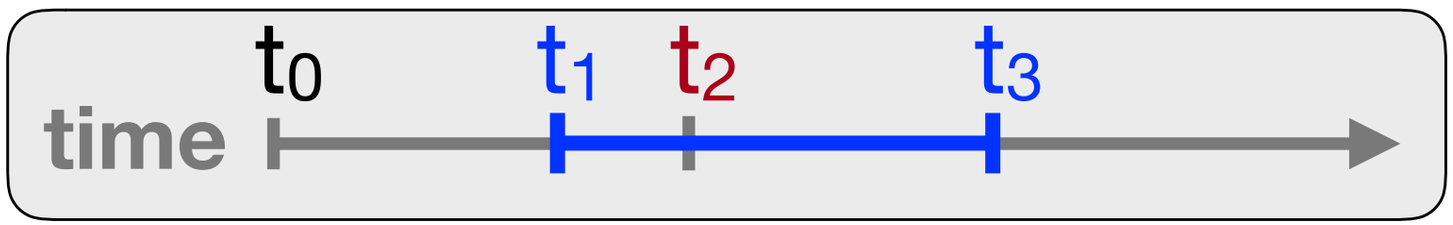
t_2 : **S2:** x=0

```
[ a := x; //0
  x := a+1;
```

SI: ***Disallowed*** Anomalies

lost update (LU)

T1:
```
a := x; //0
x := a+1;
```
‖ **T2:**
```
b := x; //0
x := b+1;
```



x ↦ t_0 : 0

t_1 : **S1:** x=0
```
a := x; //0
x := a+1;
```
t_3 : **C1:** x=1

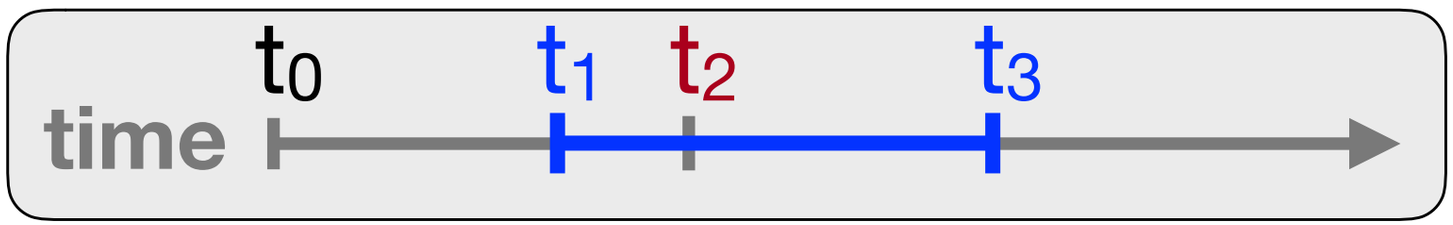
ww-conflict? no

t_2 : **S2:** x=0
```
a := x; //0
x := a+1;
```

SI: ***Disallowed*** Anomalies

lost update (LU)

T1:
```
a := x; //0
x := a+1;
```
‖ **T2:**
```
b := x; //0
x := b+1;
```



x ↦ | $t_0 : 0$ | $t_3 : 1$ |

t_1: **S1:** x=0
```
a := x; //0
x := a+1;
```
t_3: **C1:** x=1

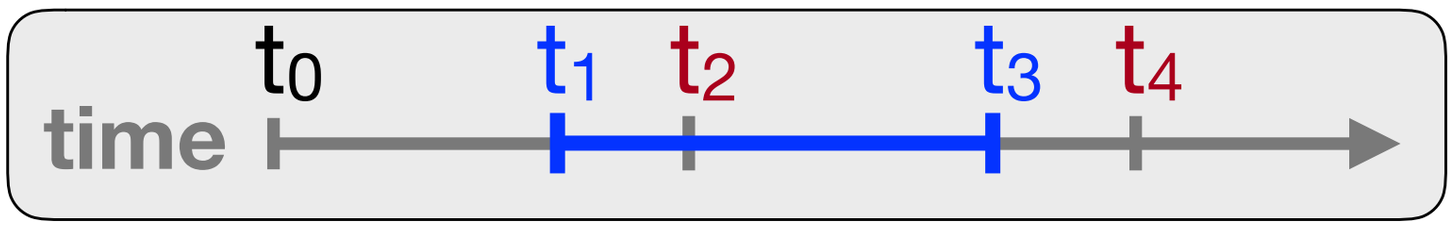
ww-conflict? no

t_2: **S2:** x=0
```
a := x; //0
x := a+1;
```

SI: ***Disallowed*** Anomalies

lost update (LU)

T1:
```
a := x; //0
x := a+1;
```
‖
T2:
```
b := x; //0
x := b+1;
```



$x \mapsto$ | $t_0 : 0$ | $t_3 : 1$

t_1 : **S1:** x=0
```
a := x; //0
x := a+1;
```
t_3 : **C1:** x=1

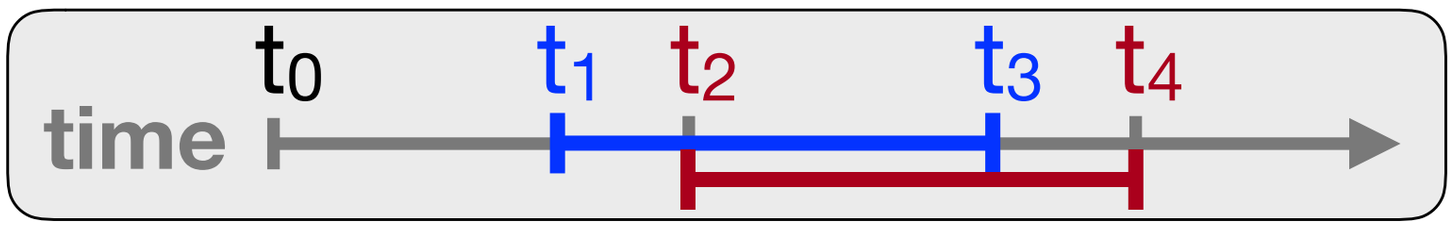
ww-conflict? no

t_2 : **S2:** x=0
```
a := x; //0
x := a+1;
```
t_4 : **C2:** x=1

SI: ***Disallowed*** Anomalies

lost update (LU)

T1:
```
a := x; //0
x := a+1;
```
‖ **T2:**
```
b := x; //0
x := b+1;
```



$x \mapsto$ | $t_0 : 0$ | $t_3 : 1$ |

t_1 : **S1:** `x=0`
```
a := x; //0
x := a+1;
```
t_3 : **C1:** `x=1`

ww-conflict? no

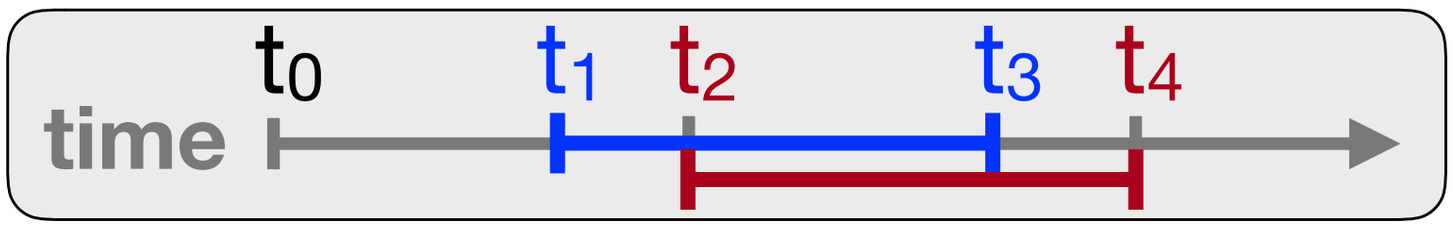
ww-conflict? yes

t_2 : **S2:** `x=0`
```
a := x; //0
x := a+1;
```
t_4 : **C2:** `x=1`

SI: ***Disallowed*** Anomalies

lost update (LU)

T1:
```
a := x; //0
x := a+1;
```
‖ **T2:**
```
b := x; //0
x := b+1;
```



$x \mapsto$ | $t_0 : 0$ | $t_3 : 1$

t_1 : **S1:** `x=0`
```
a := x; //0
x := a+1;
```
t_3 : **C1:** `x=1`

ww-conflict? no

ww-conflict? yes

t_2 : **S2:** `x=0`
```
a := x; //0
x := a+1;
```
t_4 : **C2:** `x=1`

Reasoning about
timestamps and **interval order**
is difficult

Reasoning about

timestamps and **interval order**

is difficult

⇒

Lock-based

SI reference implementation

Which Locks for SI?

✗ ***Global*** lock

➡ *disjoint* accesses allowed

✗ ***Per-location*** locks

➡ concurrent reads allowed

✓ ***Per-location MRSW*** (multiple-readers-single-writer) locks

SI Reference Implementation Pattern

```
SI(T) ≜
  // must hold read locks on RS
  s := snapshot(RS)
  // must hold write locks on WS
  ⟦ T ⟧ ;
```

```
snapshot(RS) ≜
  for(x∈RS) s[x]:=x

⟦ a := x ⟧ ≜ a := s[x]

⟦ x := a ⟧ ≜ x := a; s[x] := a

⟦ S1;S2 ⟧ ≜ ⟦ S1 ⟧ ; ⟦ S2 ⟧

...
```

SI Reference Implementation Pattern

```
SI(T) ≜
  // must hold read locks on RS
  s := snapshot(RS)
  // must hold write locks on WS
  ⟦ T ⟧ ;
```

```
snapshot(RS) ≜
  for(x∈RS) s[x]:=x

⟦ a := x ⟧ ≜ a := s[x]

⟦ x := a ⟧ ≜ x := a; s[x] := a

⟦ S1;S2 ⟧ ≜ ⟦ S1 ⟧ ; ⟦ S2 ⟧

...
```

➡ ***sound***: behaviours(imp) ⊆ behaviours(SI_spec)

e.g. disallow lost update (LU)

➡ ***complete***: behaviours(SI_spec) ⊆ behaviours(imp)

e.g. allow write skew (WS)

SI Reference Implementation: Attempt 1

```
SI(T) ≜
 // must hold read locks on RS
 s := snapshot(RS)
 // must hold write locks on WS
 ⟦ T ⟧ ;
```

sound: disallow lost update (LU)

complete: allow write skew (WS)

SI Reference Implementation: Attempt 1

```
SI(T) ≜
 r_lock(RS);
 s := snapshot(RS);
 r_unlock(RS);
 // must hold write locks on WS
 ⟦ T ⟧
```

sound: disallow lost update (LU)

complete: allow write skew (WS)

SI Reference Implementation: Attempt 1

```
SI(T) ≜
 r_lock(RS);
 s := snapshot(RS);
 r_unlock(RS);
 w_lock(WS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU)

complete: allow write skew (WS)

SI Reference Implementation: Attempt 1

```
SI(T) ≜
 r_lock(RS);
 s := snapshot(RS);
 r_unlock(RS);
 w_lock(WS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU) ✗

complete: allow write skew (WS)

SI Reference Implementation: Attempt 1

```
SI(T) ≜
 r_lock(RS);
 s := snapshot(RS);
 r_unlock(RS);
 w_lock(WS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU) ✗

T1: `a:=x; // 0` `x:=a+1;` ‖ **T2:** `b:= x; // 0` `x:= b+1;`

(LU)

SI Reference Implementation: Attempt 1

```
SI(T) ≜
 r_lock(RS);
 s := snapshot(RS);
 r_unlock(RS);
 w_lock(WS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU) ✗

T1: `a:=x; // 0` `x:=a+1;` ‖ **T2:** `b:= x; // 0` `x:= b+1;`

(LU)

SI Reference Implementation: Attempt 1

```
SI(T) ≜
 r_lock(RS);
 s := snapshot(RS);
 r_unlock(RS);
 w_lock(WS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU) ✗

T1: `a:=x; // 0` `x:=a+1;` ‖ **T2:** `b:= x; // 0` `x:= b+1;`

(LU)

SI Reference Implementation: Attempt 1

```
SI(T) ≜
 r_lock(RS);
 s := snapshot(RS);
 r_unlock(RS);
 w_lock(WS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU) ✗

T1: `a:=x; // 0` `x:=a+1;` ‖ **T2:** `b:= x; // 0` `x:= b+1;`

(LU)

SI Reference Implementation: Attempt 1

```
SI(T) ≜
 r_lock(RS);
 s := snapshot(RS);
 r_unlock(RS);
 w_lock(WS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU) ✗

T1: `a:=x; // 0` `x:=a+1;` ‖ **T2:** `b:= x; // 0` `x:= b+1;`

(LU)

SI Reference Implementation: Attempt 1

```
SI(T) ≜
 r_lock(RS);
 s := snapshot(RS);
 r_unlock(RS);
 w_lock(WS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU) ✗

T1: `a:=x; // 0` / `x:=a+1;` ‖ **T2:** `b:= x; // 0` / `x:= b+1;`

(LU)

SI Reference Implementation: Attempt 1

```
SI(T) ≜
 r_lock(RS);
 s := snapshot(RS);
 r_unlock(RS);
 w_lock(WS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

w_locks are acquired ***too late***!
acquire them ***before releasing*** r_locks!

(LU)

SI Reference Implementation: Attempt 2

```
SI(T) ≜
 w_lock(WS);
 r_lock(RS\WS);
 s := snapshot(RS);
 r_unlock(RS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

SI Reference Implementation: Attempt 2

```
SI(T) ≜
 w_lock(WS);
 r_lock(RS\WS);
 s := snapshot(RS);
 r_unlock(RS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU)

complete: allow write skew (WS)

SI Reference Implementation: Attempt 2

```
SI(T) ≜
 w_lock(WS);
 r_lock(RS\WS);
 s := snapshot(RS);
 r_unlock(RS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU) ✔

complete: allow write skew (WS)

SI Reference Implementation: Attempt 2

```
SI(T) ≜
 w_lock(WS);
 r_lock(RS\WS);
 s := snapshot(RS);
 r_unlock(RS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU) ✔

complete: allow write skew (WS) ✗

T1: `x:=1; a:=y;//0` ‖ **T2:** `y:= 1; b:= x;//0`

(WS)

SI Reference Implementation: Attempt 2

```
SI(T) ≜
 w_lock(WS);
 r_lock(RS\WS);
 s := snapshot(RS);
 r_unlock(RS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU) ✔

complete: allow write skew (WS) ✘

T1: `x:=1; a:=y;//0` ‖ **T2:** `y:= 1; b:= x;//0`

(WS)

T1

```
1. w_lock({x});
2. r_lock({y});
3. s[y] := y;
4. r_unlock({y});
5. x := 1; s[x] := 1;
6. a := s[y];
7. w_unlock({x});
```

T2

```
8. w_lock({y});
9. r_lock({x});
10. s[x] := x;
11. r_unlock({x});
12. y := 1; s[y] := 1
13. b := s[x];
14. w_unlock({y});
```

SI Reference Implementation: Attempt 2

```
SI(T) ≜
 w_lock(WS);
 r_lock(RS\WS);
 s := snapshot(RS);
 r_unlock(RS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU) ✔

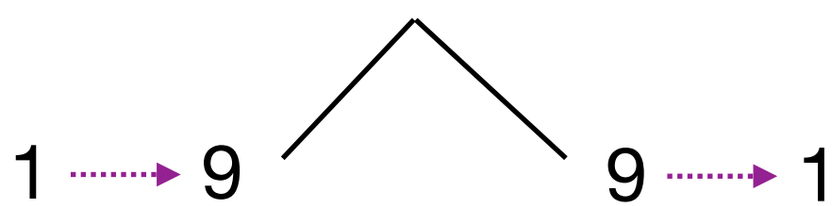
complete: allow write skew (WS) ✘

T1: `x:=1; a:=y;//0` ‖ **T2:** `y:= 1; b:= x;//0`

(WS)

```
1. w_lock({x});          T1
2. r_lock({y});
3. s[y] := y;
4. r_unlock({y});
5. x := 1; s[x] := 1;
6. a := s[y];
7. w_unlock({x});
```

```
8.  w_lock({y});         T2
9.  r_lock({x});
10. s[x] := x;
11. r_unlock({x});
12. y := 1; s[y] := 1
13. b := s[x];
14. w_unlock({y});
```



⟶ : execution order

SI Reference Implementation: Attempt 2

```
SI(T) ≜
 w_lock(WS);
 r_lock(RS\WS);
 s := snapshot(RS);
 r_unlock(RS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU) ✔

complete: allow write skew (WS) ✘

T1: `x:=1; a:=y;//0` ‖ **T2:** `y:= 1; b:= x;//0`

(WS)

```
1. w_lock({x});          T1
2. r_lock({y});
3. s[y] := y;
4. r_unlock({y});
5. x := 1; s[x] := 1;
6. a := s[y];
7. w_unlock({x});
```

```
8.  w_lock({y});         T2
9.  r_lock({x});
10. s[x] := x;
11. r_unlock({x});
12. y := 1; s[y] := 1
13. b := s[x];
14. w_unlock({y});
```

1 ⇢ 9 9 ⇢ 1

⇢ : execution order

SI Reference Implementation: Attempt 2

```
SI(T) ≜
 w_lock(WS);
 r_lock(RS\WS);
 s := snapshot(RS);
 r_unlock(RS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU) ✔

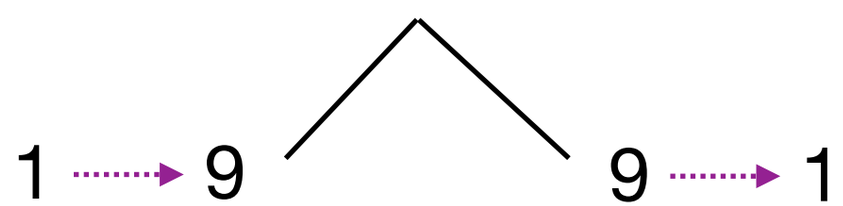
complete: allow write skew (WS) ✘

T1: `x:=1; a:=y;//0` ‖ **T2:** `y:= 1; b:= x;//0`

(WS)

```
1. w_lock({x});          T1
2. r_lock({y});
3. s[y] := y;
4. r_unlock({y});
5. x := 1; s[x] := 1;
6. a := s[y];
7. w_unlock({x});
```

```
8.  w_lock({y});         T2
9.  r_lock({x});
10. s[x] := x;
11. r_unlock({x});
12. y := 1; s[y] := 1
13. b := s[x];
14. w_unlock({y});
```



⟶ : execution order

SI Reference Implementation: Attempt 2

```
SI(T) ≜
 w_lock(WS);
 r_lock(RS\WS);
 s := snapshot(RS);
 r_unlock(RS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU) ✔

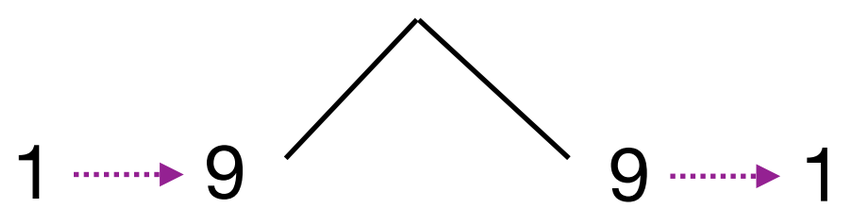
complete: allow write skew (WS) ✘

T1: `x:=1; a:=y;//0` ‖ **T2:** `y:= 1; b:= x;//0`

(WS)

```
1. w_lock({x});          T1
2. r_lock({y});
3. s[y] := y;
4. r_unlock({y});
5. x := 1; s[x] := 1;
6. a := s[y];
7. w_unlock({x});
```

```
8.  w_lock({y});         T2
9.  r_lock({x});
10. s[x] := x;
11. r_unlock({x});
12. y := 1; s[y] := 1
13. b := s[x];
14. w_unlock({y});
```



⟶ : execution order

SI Reference Implementation: Attempt 2

```
SI(T) ≜
 w_lock(WS);
 r_lock(RS\WS);
 s := snapshot(RS);
 r_unlock(RS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU) ✔

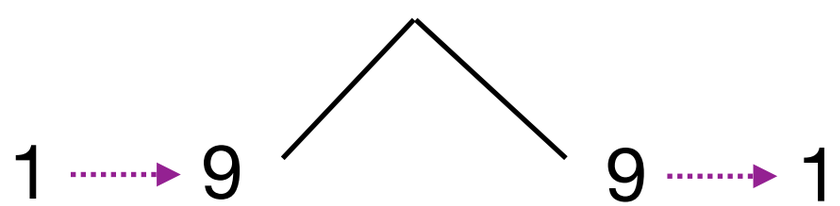
complete: allow write skew (WS) ✘

T1: `x:=1; a:=y;//0` ‖ **T2:** `y:= 1; b:= x;//0`

(WS)

```
1. w_lock({x});          T1
2. r_lock({y});
3. s[y] := y;
4. r_unlock({y});
5. x := 1; s[x] := 1;
6. a := s[y];
7. w_unlock({x});
```

```
8.  w_lock({y});         T2
9.  r_lock({x});
10. s[x] := x;    //1
11. r_unlock({x});
12. y := 1; s[y] := 1
13. b := s[x];    //1
14. w_unlock({y});
```



⋯→ : execution order

SI Reference Implementation: Attempt 2

```
SI(T) ≜
 w_lock(WS);
 r_lock(RS\WS);
 s := snapshot(RS);
 r_unlock(RS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU) ✔

complete: allow write skew (WS) ✗

T1: `x:=1; a:=y; // 0` ‖ **T2:** `y:= 1; b:= x; // 0`

(WS)

T1

```
1. w_lock({x});
2. r_lock({y});
3. s[y] := y;
4. r_unlock({y});
5. x := 1; s[x] := 1;
6. a := s[y];
7. w_unlock({x});
```

T2

```
8.  w_lock({y});
9.  r_lock({x});
10. s[x] := x;   // 1
11. r_unlock({x});
12. y := 1; s[y] := 1
13. b := s[x];   // 1
14. w_unlock({y});
```

✗ 1 ⇢ 9 ⇢ 1

9 ⇢ 1

⇢ : execution order

SI Reference Implementation: Attempt 2

```
SI(T) ≜
 w_lock(WS);
 r_lock(RS\WS);
 s := snapshot(RS);
 r_unlock(RS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU) ✔

complete: allow write skew (WS) ✘

T1: `x:=1; a:=y;//0` ‖ **T2:** `y:= 1; b:= x;//0`

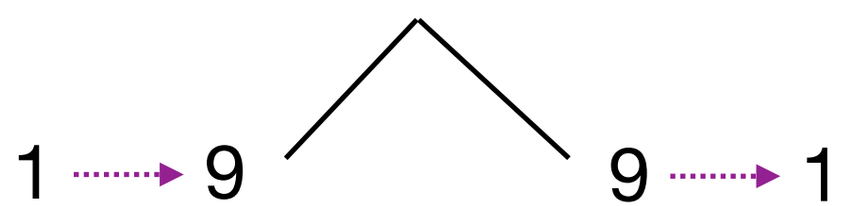
(WS)

T1

```
1. w_lock({x});
2. r_lock({y});
3. s[y] := y;
4. r_unlock({y});
5. x := 1; s[x] := 1;
6. a := s[y];
7. w_unlock({x});
```

T2

```
8.  w_lock({y});
9.  r_lock({x});
10. s[x] := x;    //1
11. r_unlock({x});
12. y := 1; s[y] := 1
13. b := s[x];    //1
14. w_unlock({y});
```



⋯▸ : execution order

SI Reference Implementation: Attempt 2

```
SI(T) ≜
 w_lock(WS);
 r_lock(RS\WS);
 s := snapshot(RS);
 r_unlock(RS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU) ✔

complete: allow write skew (WS) ✗

T1: `x:=1; a:=y;//0` ‖ **T2:** `y:= 1; b:= x;//0`

(WS)

```
1. w_lock({x});          T1
2. r_lock({y});
3. s[y] := y;
4. r_unlock({y});
5. x := 1; s[x] := 1;
6. a := s[y];
7. w_unlock({x});
```

```
8.  w_lock({y});          T2
9.  r_lock({x});
10. s[x] := x;    //1
11. r_unlock({x});
12. y := 1; s[y] := 1
13. b := s[x];    //1
14. w_unlock({y});
```

1 ⇢ 9 9 ⇢ 1

⇢ : execution order

SI Reference Implementation: Attempt 2

```
SI(T) ≜
 w_lock(WS);
 r_lock(RS\WS);
 s := snapshot(RS);
 r_unlock(RS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU) ✔

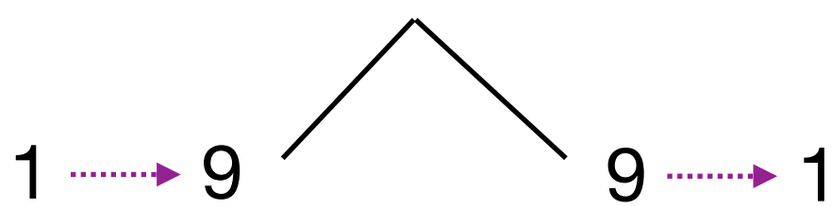
complete: allow write skew (WS) ✘

T1: `x:=1; a:=y;//0` ‖ **T2:** `y:= 1; b:= x;//0`

(WS)

```
1. w_lock({x});          T1
2. r_lock({y});
3. s[y] := y;
4. r_unlock({y});
5. x := 1; s[x] := 1;
6. a := s[y];
7. w_unlock({x});
```

```
8.  w_lock({y});          T2
9.  r_lock({x});
10. s[x] := x;    //1
11. r_unlock({x});
12. y := 1; s[y] := 1
13. b := s[x];    //1
14. w_unlock({y});
```



⟶ : execution order

SI Reference Implementation: Attempt 2

```
SI(T) ≜
 w_lock(WS);
 r_lock(RS\WS);
 s := snapshot(RS);
 r_unlock(RS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU) ✔

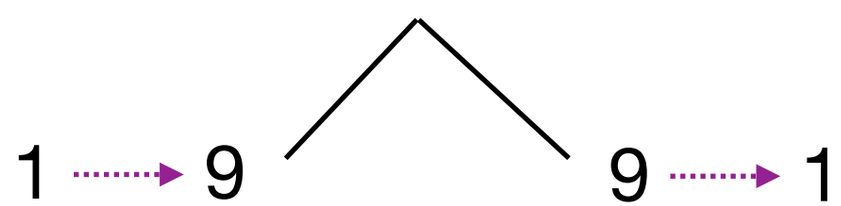
complete: allow write skew (WS) ✘

T1: `x:=1; a:=y;//0` ‖ **T2:** `y:= 1; b:= x;//0`

(WS)

```
1. w_lock({x});          T1
2. r_lock({y});
3. s[y] := y;
4. r_unlock({y});
5. x := 1; s[x] := 1;
6. a := s[y];
7. w_unlock({x});
```

```
8.  w_lock({y});          T2
9.  r_lock({x});
10. s[x] := x;    //1
11. r_unlock({x});
12. y := 1; s[y] := 1
13. b := s[x];    //1
14. w_unlock({y});
```



⟶ : execution order

SI Reference Implementation: Attempt 2

```
SI(T) ≜
 w_lock(WS);
 r_lock(RS\WS);
 s := snapshot(RS);
 r_unlock(RS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU) ✔

complete: allow write skew (WS) ✘

T1: `x:=1; a:=y;//0` ‖ **T2:** `y:= 1; b:= x;//0`

(WS)

```
1. w_lock({x});          T1
2. r_lock({y});
3. s[y] := y;//1
4. r_unlock({y});
5. x := 1; s[x] := 1;
6. a := s[y]; //1
7. w_unlock({x});
```

```
8.  w_lock({y});         T2
9.  r_lock({x});
10. s[x] := x;   //1
11. r_unlock({x});
12. y := 1; s[y] := 1
13. b := s[x];   //1
14. w_unlock({y});
```

1 ⇢ 9 9 ⇢ 1

⇢ : execution order

SI Reference Implementation: Attempt 2

```
SI(T) ≜
 w_lock(WS);
 r_lock(RS\WS);
 s := snapshot(RS);
 r_unlock(RS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU) ✔

complete: allow write skew (WS) ✗

T1: `x:=1; a:=y; // 0` ‖ **T2:** `y:= 1; b:= x; // 0`

(WS)

T1

```
1. w_lock({x});
2. r_lock({y});
3. s[y] := y; // 1
4. r_unlock({y});
5. x := 1; s[x] := 1;
6. a := s[y]; // 1
7. w_unlock({x});
```

T2

```
8.  w_lock({y});
9.  r_lock({x});
10. s[x] := x;   // 1
11. r_unlock({x});
12. y := 1; s[y] := 1
13. b := s[x];   // 1
14. w_unlock({y});
```

1 ⇢ 9 ✗ 9 ⇢ 1

⇢ : execution order

SI Reference Implementation: Attempt 2

```
SI(T) ≜
 w_lock(WS);
 r_lock(RS\WS);
 s := snapshot(RS);
 r_unlock(RS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU) ✔

complete: allow write skew (WS) ✘

T1: `x:=1; a:=y;//0` ‖ **T2:** `y:= 1; b:= x;//0`

(WS)

w_locks are acquired ***too early***!

1. acquire them ***as r_locks*** first;
2. ***promote them*** to writers later

⋯▸ : execution order

SI Reference Implementation: Attempt 3

```
SI(T) ≜
 r_lock(RS∪WS);
 s := snapshot(RS);
 r_unlock(RS\WS);
 promote(WS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU)

complete: allow write skew (WS)

SI Reference Implementation: Attempt 3

```
SI(T) ≜
 r_lock(RS∪WS);
 s := snapshot(RS);
 r_unlock(RS\WS);
 promote(WS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU) ✔

complete: allow write skew (WS) ✔

SI Reference Implementation: Attempt 3

```
SI(T) ≜
 r_lock(RS∪WS);
 s := snapshot(RS);
 r_unlock(RS\WS);
 promote(WS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

sound: disallow lost update (LU) ✔

complete: allow write skew (WS) ✔

1. ***Prescient*** implementation:
   requires prior knowledge of `RS` and `WS`
   ⇒ see paper for ***non-prescient*** variant
2. May ***deadlock***
   ⇒ see paper for ***non-deadlocking*** implementations
   (prescient and non-prescient)

What about **mixed** accesses?

RSI = SI + **mixed** accesses

What about **mixed** accesses?

$$\mathsf{acyclic}(\mathsf{rsi\text{-}hb}_{loc} \cup \mathsf{mo} \cup \mathsf{rb})$$

where

$$\mathsf{rsi\text{-}hb} \triangleq (\mathsf{rsi\text{-}po} \cup \mathsf{rsi\text{-}rf} \cup \mathsf{mo}_{\mathsf{T}} \cup \mathsf{si\text{-}rb})^{+}$$
$$\mathsf{rsi\text{-}po} \triangleq (\mathsf{po} \setminus \mathsf{po}_{\mathrm{I}}) \cup [\mathcal{W}]; \mathsf{po}_{\mathrm{I}}; [\mathcal{W}]$$
$$\mathsf{rsi\text{-}rf} \triangleq (\mathsf{rf}; [\mathcal{NT}]) \cup ([\mathcal{NT}]; \mathsf{rf}; \mathsf{st}) \cup \mathsf{rf}_{\mathsf{T}} \cup (\mathsf{mo}; \mathsf{rf})_{\mathsf{T}}$$
$$\mathsf{si\text{-}rb} \triangleq [\mathcal{R}_{\mathsf{E}}]; \mathsf{rb}_{\mathsf{T}}; [\mathcal{W}] \quad \text{and} \quad \mathcal{R}_{\mathsf{E}} \triangleq \{r \mid \exists w.\ (w, r) \in \mathsf{rf}_{\mathsf{E}}\}$$

RSI = SI + **mixed** accesses

What about **mixed** accesses?

⇒

Lock-based

RSI reference implementation

RSI = SI + **mixed** accesses

RSI: SI + Mixed Accesses

```
x := 1;          ||   T:
y := 1;          ||   [ a := x; // 0
                 ||     b := y; // 1
```

RSI: SI + Mixed Accesses

```
x := 1;          ‖   T:
y := 1;          ‖   [ a := x; // 0
                 ‖     b := y; // 1
```

```
snapshot(RS) ≜
  for(x∈RS)  s[x]:=x
```

RSI: SI + Mixed Accesses

```
x := 1;          ||   T:
y := 1;          ||   [ a := x; // 0
                 ||     b := y; // 1
```

Solution: read every location ***twice***!

```
snapshot(RS) ≜
  for(x∈RS) s[x]:=x
```

```
snapshot_RSI(RS) ≜
  // tentative snapshot
  for(x∈RS) s[x]:=x
  // validate snapshot
  for(x∈RS)
    if (s[x]!=x) snapshot(RS)
```

RSI: SI + Mixed Accesses

Caveat: non-transactional writes with ***same value*** ***cannot race*** with the ***same transaction***

Solution: read every location ***twice***!

```
snapshot(RS) ≜
  for(x∈RS) s[x]:=x
```

```
snapshot_RSI(RS) ≜
  // tentative snapshot
  for(x∈RS) s[x]:=x
  // validate snapshot
  for(x∈RS)
    if (s[x]!=x) snapshot(RS)
```

RSI Reference Implementation

```
SI(T) ≜
 r_lock(RS∪WS);
 s := snapshot(RS);
 r_unlock(RS\WS);
 promote(WS);
 ⟦ T ⟧ ;
 w_unlock(WS);
```

RSI Reference Implementation

```
RSI(T) ≜
  r_lock(RS∪WS);
  s := snapshot_RSI(RS);
  r_unlock(RS\WS);
  promote(WS);
  ⟦ T ⟧ ;
  w_unlock(WS);
```

➡ ***Sound***: behaviours(imp) ⊆ behaviours(RSI_spec)

➡ ***Complete***: behaviours(RSI_spec) ⊆ behaviours(imp)

RSI Reference Implementation

```
RSI(T) ≜
  r_lock(RS∪WS);
  s := snapshot_RSI(RS);
  r_unlock(RS\WS);
  promote(WS);
  ⟦ T ⟧ ;
  w_unlock(WS);
```

➡ ***Sound***: behaviours(imp) ⊆ behaviours(RSI_spec)

➡ ***Complete***: behaviours(RSI_spec) ⊆ behaviours(imp)

1. ***Prescient*** implementation
   ⇒ see paper for ***non-prescient*** variant
2. May ***deadlock***
   ⇒ see our paper for ***non-deadlocking*** implementations (prescient and non-prescient)

Conclusions

✓ ***Sound*** & ***complete*** lock-based reference implementation for ***SI***

- ***prescient*** and ***non-prescient*** variants

✓ *Declarative* ***RSI*** semantics with ***mixed*** accesses

✓ ***Sound*** & ***complete*** reference implementation for ***RSI***

- ***prescient*** and ***non-prescient*** variants

Conclusions

✓ ***Sound*** & ***complete*** lock-based reference implementation for ***SI***

- ***prescient*** and ***non-prescient*** variants

✓ *Declarative* ***RSI*** semantics with ***mixed*** accesses

✓ ***Sound*** & ***complete*** reference implementation for ***RSI***

- ***prescient*** and ***non-prescient*** variants

Thank you for listening!

azalea@mpi-sws.org SoundAndComplete.org @azalearaad