Shared Variable
Concurrency
Shared Variable
Concurrency

• Sequential Consistency (SC)
• Total Store Ordering (TSO)
• Memory Fences
• Non-monotonicity
• Load-Buffer Semantics
Shared Variable
Concurrency

• Sequential Consistency (SC)
• Total Store Ordering (TSO)
• Memory Fences
• Non-monotonicity
• Load-Buffer Semantics
Shared Variable
Concurrency

- Sequential Consistency (SC)
- Total Store Ordering (TSO)
- Memory Fences
- Non-monotonicity
- Load-Buffer Semantics
Shared Variable
Concurrency
Shared Variable
Concurrency

process $P_1$
Shared Variable
Concurrency

process $P_1$
process $P_2$
Shared Variable
Concurrency

- Process $P_1$
- Process $P_2$
- Memory

Diagram showing the interaction between processes and memory in a concurrency context.
Shared Variable
Concurrency

- Process P₁ connects to the shared variable x=1.
- Process P₂ also connects to the shared variable x=1.
- Both processes access the shared variable through memory.
Shared Variable
Concurrency

- Process $P_1$
- Process $P_2$
- Shared variable $x=1$
- Shared variable $y=1$
- Memory

Wallpaperswide.com
Shared Variable
Concurrency

- process \( P_1 \)
- process \( P_2 \)
- \( x = 1 \) (shared variable)
- \( y = 1 \) (shared variable)
- memory
Shared Variable
Concurrency

Sequential Consistency (SC)

process $P_1$

process $P_2$

variable $x=1$

variable $y=1$

memory

shared variable

shared variable
Shared Variable
Concurrency

Sequential
Consistency
(SC)

atomic
operations

(processes)
P_1

(processes)
P_2

(shared variable)
x=1

(shared variable)
y=1

(memory)
Shared Variable
Concurrency

Sequential Consistency (SC)

- atomic writes
- read from memory

atomic operations

process \( P_1 \)
\( x = 1 \)
shared variable

process \( P_2 \)
\( y = 1 \)
shared variable

memory
Shared Variable
Concurrency
Sequential Consistency (SC)

- atomic writes
- read from memory

atomic operations

process $P_1$

process $P_2$

memory

shared variable $x=1$

shared variable $y=1$
Shared Variable
Concurrency

Sequential
Consistency
(SC)

- atomic writes
- read from memory

atomic
operations

\( x:=3 \)

process
\( P_1 \)

shared
variable
\( x=1 \)

process
\( P_2 \)

shared
variable
\( y=1 \)

memory
Shared Variable

Concurrency

Sequential Consistency (SC)

atomic writes

atomic operations

• atomic writes
• read from memory

process P

shared variable x=1
y=1

P_1

P_2

memory

x:=3
Shared Variable Concurrency

Sequential Consistency (SC)

- Atomic writes
- Read from memory

Atomic operations

Process P1

Process P2

x := 3

y := 1

Shared variable

Memory

Shared variable

x := 3

y := 1
Shared Variable Concurrency

Sequential Consistency (SC)

- atomic writes
- read from memory

atomic operations

- process \( P_1 \)
- process \( P_2 \)

- shared variable \( x=3 \)
- shared variable \( y=1 \)

memory
Shared Variable Concurrency

Sequential Consistency (SC)

- atomic writes
- read from memory

atomic operations

process

P₁

x=3

shared variable

P₂

y=1

shared variable

memory

a:=x
Shared Variable
Concurrency

Sequential
Consistency (SC)

- atomic writes
- read from memory

atomic operations

process $P_1$

process $P_2$

$a := x$

$x = 3$

$y = 1$

shared variable

shared variable

memory
Shared Variable
Concurrency

Sequential Consistency (SC)

- atomic writes
- read from memory

atomic operations

process P1

x=3

shared variable

process P2

y=1

shared variable

memory

a:=3
Shared Variable

Concurrency

Sequential Consistency (SC)

- atomic writes
- read from memory

atomic operations

• process $P_1$
• process $P_2$

$a := 3$

$x = 3$
$y = 1$

memory

shared variable

shared variable
Shared Variable Concurrency

Sequential Consistency (SC)
- atomic writes
- read from memory

• atomic operations

Sequential Consistency (SC)
+ simple & intuitive
- expensive
Shared Variable Concurrency

Sequential Consistency (SC)

- atomic writes
- read from memory

Dekker Protocol

\[
x := 1 \\
a := y \\
\text{if } a = 0 \text{ then } \\
\text{CS}_1 \\
\]

\[
y := 1 \\
b := x \\
\text{if } b = 0 \text{ then } \\
\text{CS}_2 \\
\]

not (CS\textsubscript{1} and CS\textsubscript{2})

assertion

\[
\text{P}_1 \quad x = 0 \\
\text{P}_2 \quad y = 0
\]
Dekker Protocol

Shared Variable
Concurrency

Sequential
Consistency
(SC)

- atomic writes
- read from memory

atomic operations

\[ x := 1 \]
\[ a := y \]
\[ \text{if } a = 0 \text{ then} \]
\[ CS_1 \]

\[ y := 1 \]
\[ b := x \]
\[ \text{if } b = 0 \text{ then} \]
\[ CS_2 \]

not (CS_1 and CS_2)

assertion

\[ x = 0 \]
\[ y = 0 \]
**Shared Variable**

**Concurrency**

**Sequential Consistency (SC)**

- atomic writes
- read from memory

**Atomic operations**

```
x:=1
a:=y
if a=0 then CS1
y:=1
b:=x
if b=0 then CS2
not (CS1 and CS2)
```

**Dekker Protocol**
Dekker Protocol

assertion

\[ x := 1 \]
\[ a := y \]
\[ \text{if } a = 0 \text{ then } CS_1 \]
\[ y := 1 \]
\[ b := x \]
\[ \text{if } b = 0 \text{ then } CS_2 \]

\[ \neg (CS_1 \text{ and } CS_2) \]

Sequential Consistency (SC)

- atomic writes
- read from memory

atomic operations

\[ x = 1 \]
\[ y = 0 \]
Shared Variable
Concurrency

Sequential Consistency (SC)
• atomic writes
• read from memory

atomic operations

Dekker Protocol

\[
\begin{align*}
&x := 1 \\
&a := y \\
&\text{if } a = 0 \text{ then} \\
&\text{CS}_1 \\
&y := 1 \\
&b := x \\
&\text{if } b = 0 \text{ then} \\
&\text{CS}_2 \\
\end{align*}
\]

assertion

P₁ \rightarrow x = 1

P₂ \rightarrow y = 0

not (CS₁ and CS₂)
Shared Variable
Concurrency

Sequential
Consistency (SC)

• atomic writes
• read from memory

atomic
operations

Dekker Protocol

P1

x=1

a:=y
if a=0 then
CS1

P2

y:=1
b:=x
if b=0 then
CS2

not (CS1 and CS2)

assertion
Shared Variable
Concurrency
Sequential Consistency (SC)

- atomic writes
- read from memory

atomic operations

Dekker Protocol

assertion

P1 -> x=1
if a=0 then
CS1
P1

P2 -> y=1
b:=x
if b=0 then
CS2
P2

not (CS1 and CS2)
Shared Variable Concurrency

Sequential Consistency (SC)

- atomic writes
- read from memory

atomic operations

Dekker Protocol

x:=1
a:=0
if a=0 then
CS1

y:=1
b:=x
if b=0 then
CS2

not (CS1 and CS2)

assertion

P1
x=1

P2
y=0

0

//0

P1

P2

x=1
y=0

P1

P2

x=1
y=0

P1

P2

x=1
y=0
**Shared Variable Concurrency**

**Sequential Consistency (SC)**
- atomic writes
- read from memory

**Dekker Protocol**

```
//0 not (CS1 and CS2)
```

**Assertion**

```
x:=1
a:=y //0
if a=0 then
  CS1
y:=1
b:=x
if b=0 then
  CS2
```

```
x=1
y=0
```
Shared Variable
Concurrency

Sequential
Consistency (SC)

- atomic writes
- read from memory

atomic operations

Dekker Protocol

\[ x := 1 \]
\[ a := y // 0 \]
\[ \text{if } a = 0 \text{ then } \]
\[ CS_1 \]
\[ y := 1 \]
\[ b := x \]
\[ \text{if } b = 0 \text{ then } \]
\[ CS_2 \]

\[ \text{not (} CS_1 \text{ and } CS_2 \text{)} \]

assertion
Shared Variable
Concurrency

Sequential Consistency (SC)

- atomic writes
- read from memory

atomic operations

Dekker Protocol

\[ x := 1 \]
\[ a := y \text{ //0} \]
\[ \text{if } a = 0 \text{ then} \]
\[ \text{CS}_1 \]
\[ y := 1 \]
\[ b := x \]
\[ \text{if } b = 0 \text{ then} \]
\[ \text{CS}_2 \]

not (CS\(_1\) and CS\(_2\))

assertion
Shared Variable
Concurrency

Sequential Consistency (SC)
- atomic writes
- read from memory

atomic operations

Dekker Protocol

```
x := 1
a := y //0
if a = 0 then
    CS1
```

```
y := 1
b := x
if b = 0 then
    CS2
```

\[ \text{not (CS1 and CS2)} \]

assertion

\[ P_1 \xrightarrow{\text{P1}} x = 1 \]
\[ P_2 \xrightarrow{\text{P2}} y = 1 \]
Shared Variable
Concurrency

Sequential Consistency (SC)

• atomic writes
• read from memory

atomic operations

Dekker Protocol

x:=1
a:=y //0
if a=0 then
CS_1

y:=1
b:=x
if b=0 then
CS_2

not (CS_1 and CS_2)

assertion
Shared Variable
Concurrency

Sequential
Consistency (SC)

• atomic writes
• read from memory

atomic operations

Dekker Protocol

P1

P2

x:=1
a:=y //0
if a=0 then
CS1

y:=1
b:=x
if b=0 then
CS2

not (CS1 and CS2)

assertion

x=1
y=1

//0

//1

not (CS1 and CS2)
Shared Variable
Concurrency

Sequential
Consistency
(SC)

- atomic writes
- read from memory

atomic operations

Dekker Protocol

assertion

\[ x := 1 \]
\[ a := y \]
\[ \text{if } a = 0 \text{ then } \]
\[ CS_1 \]
\[ y := 1 \]
\[ b := x \]
\[ \text{if } b = 0 \text{ then } \]
\[ CS_2 \]
\[ \text{not (} CS_1 \text{ and } CS_2 \text{)} \]
Shared Variable

Concurrency

Sequential Consistency (SC)

- atomic writes
- read from memory

atomic operations

Dekker Protocol

\[
\begin{align*}
    &x := 1 \\
    &a := y \quad // 0 \\
    &\text{if } a = 0 \text{ then} \\
    &\quad CS_1 \\
    &y := 1 \\
    &b := x \quad // 1 \\
    &\text{if } b = 0 \text{ then} \\
    &\quad CS_2 \\
\end{align*}
\]

not (CS_1 and CS_2)

assertion
Shared Variable
Concurrency

Sequential Consistency (SC)
- atomic writes
- read from memory

atomic operations

Dekker Protocol

P₁ ➔ x=1
a:=y //0
if a=0 then
  CS₁

P₂ ➔ y=1
b:=x //1
if b=0 then
  CS₂

not (CS₁ and CS₂)

assertion

x=1
y=1

//0
//1

P₁

P₂

//0
//1

assertion
Shared Variable
Concurrency

Sequential
Consistency (SC)

- atomic writes
- read from memory

atomic operations

Dekker Protocol

P1

x:=1
a:=y //0
if a=0 then
CS1

y:=1
b:=x //1
if b=0 then
CS2

not (CS1 and CS2)

P2

x=1
y=1

assertion
Shared Variable Concurrency

Sequential Consistency (SC)

- atomic writes
- read from memory

atomic operations

Dekker Protocol

P1

\[ x := 1 \]
\[ a := y \quad // 0 \]
\[ \text{if } a = 0 \text{ then} \]
\[ \text{CS}_1 \]

P2

\[ y := 1 \]
\[ b := x \quad // 1 \]
\[ \text{if } b = 0 \text{ then} \]
\[ \text{CS}_2 \]

not (CS\(_1\) and CS\(_2\))

assertion

\[ x = 1 \]
\[ y = 1 \]
Shared Variable
Concurrency
Sequential
Consistency (SC)
- atomic writes
- read from memory

atomic operations

Dekker Protocol

assertion

P1
x:=1
a:=y  //0
if a=0 then
    CS1

P2
y:=1
b:=x  //1
if b=0 then
    CS2

not (CS1 and CS2)

x=1
y=1

//0
//1

✓
**Shared Variable Concurrency**

**Sequential Consistency (SC)**
- atomic writes
- read from memory

**Dekker Protocol**

1. \( x := 1 \)
2. \( a := y \) //0
3. if \( a = 0 \) then
   - \( CS_1 \)
4. \( y := 1 \)
5. \( b := x \) //1
6. if \( b = 0 \) then
   - \( CS_2 \)

**Assertion**

\[ \text{not (CS}_1 \text{ and CS}_2 \text{)} \] under SC

\( x = 1 \)
\( y = 1 \)
Shared Variable
Concurrency

- Sequential Consistency (SC)
- Total Store Ordering (TSO)
- Memory Fences
- Non-monotonicity
- Load-Buffer Semantics
Shared Variable
Concurrency

- Sequential Consistency (SC)
- Total Store Ordering (TSO)
- Memory Fences
- Non-monotonicity
- Load-Buffer Semantics
Shared Variable
Concurrency
Shared Variable
Concurrency

Total Store
Order (TSO)
<table>
<thead>
<tr>
<th>Shared Variable Concurrency</th>
<th>Total Store Order (TSO)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• non-atomic writes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• read locally or from memory</td>
<td></td>
</tr>
<tr>
<td>Shared Variable</td>
<td>Total Store Order (TSO)</td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------</td>
<td>---</td>
</tr>
<tr>
<td>Concurrency</td>
<td>• non-atomic writes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• read locally or from memory</td>
<td></td>
</tr>
</tbody>
</table>
Shared Variable
Concurrency

Total Store
Order (TSO)

- non-atomic writes
- read locally or from memory

process $P_1$

process $P_2$

memory

shared variable

$x=1$

$y=1$
Shared Variable

Concurrency

Total Store Order (TSO)

- non-atomic writes
- read locally or from memory

process $P_1$ → $x=0$

process $P_2$ → $y=1$

shared variable

memory
Shared Variable
Concurrency

Total Store
Order (TSO)

• non-atomic writes
• read locally or from memory

process \( P_1 \)

\[ x = 0 \]

shared variable

store buffer

memory

process \( P_2 \)

\[ y = 1 \]
Shared Variable Concurrency

Total Store Order (TSO)

• non-atomic writes
• read locally or from memory

\[
\begin{align*}
\text{process } P_1 & \quad \text{store buffer} \\
\text{shared variable } x &= 0 \\
\text{shared variable } y &= 1 \\
\text{process } P_2 & \quad \text{store buffer} \quad \text{memory}
\end{align*}
\]
Shared Variable

Concurrency

Total Store
Order (TSO)

• non-atomic writes
• read locally or from memory

P1

store buffer

x=0

shared variable

store buffer

P2

y=1

shared variable

memory
Shared Variable
Concurrency

Total Store
Order (TSO)

- non-atomic writes
- read locally or from memory

\[ x := 1 \]
\[ x := 2 \]
\[ a := x \]
\[ b := y \]

\[ x = 0 \]
\[ y = 1 \]
Shared Variable
Concurrency

Total Store
Order (TSO)

- non-atomic writes
- read locally or from memory

x := 1
x := 2
a := x
b := y

P₁
P₂

store buffer

x = 1

store buffer

y = 1

shared variable

memory

shared variable

shared variable
Shared Variable
Concurrency
Total Store
Order (TSO)

• non-atomic writes
• read locally or from memory

\[ x = 0 \]
\[ x = 1 \]
\[ x := 1 \]
\[ x := 2 \]
\[ a := x \]
\[ b := y \]
\[ P_1 \]
\[ P_2 \]
Shared Variable
Concurrency

Total Store
Order (TSO)

- non-atomic writes
- read locally or from memory

\[ \begin{align*}
  x &= 0 \\
  y &= 1
\end{align*} \]

\[ \begin{align*}
  P_1 &\rightarrow x = 1 \\
  P_2 &\rightarrow \text{write-to-buffer}
\]
Shared Variable Concurrency

Total Store Order (TSO)

- non-atomic writes
- read locally or from memory

```
x=0
shared variable
y=1
shared variable
```

```
P_1
store buffer
write-to-buffer
x=1
append write-message to buffer
P_2
store buffer
write-to-buffer
x:=1
a:=x
b:=y
```
Shared Variable Concurrency

Total Store Order (TSO)

- non-atomic writes
- read locally or from memory

P1

P2

x:=1
x:=2
a:=x
b:=y

x=1
store buffer

x=0
shared variable
y=1
shared variable

store buffer
memory
Shared Variable
Concurrency

Total Store Order (TSO)

- non-atomic writes
- read locally or from memory

P1
x:=1
x:=2
a:=x
b:=y

P2

x=2
x=1

store buffer

x=0
y=1

shared variable

store buffer

memory

shared variable

store buffer
Shared Variable Concurrency

Total Store Order (TSO)

• non-atomic writes
• read locally or from memory

P₁

x:=1
x:=2
a:=x
b:=y

write-to-buffer

store buffer

x=2 x=1

P₂

x:=0
y:=1

shared variable

memory

shared variable

store buffer
Shared Variable
Concurrency

Total Store
Order (TSO)

- non-atomic writes
- read locally or from memory

x=0
shared variable

y=1
shared variable

\[ x := 1 \]
\[ x := 2 \]
\[ a := x \]
\[ b := y \]

\[ P_1 \]
\[ P_2 \]

store buffer

store buffer

memory
Shared Variable

Concurrency

Total Store Order (TSO)

- non-atomic writes
- read locally or from memory

P1

x:=1
x:=2
a:=x
b:=y

P2

x:=1

x:=2

a:=x

b:=y

store buffer

shared variable

read-own-write

store buffer

memory

x=2
x=1

x=0
y=1
Shared Variable Concurrency

Total Store Order (TSO)

- non-atomic writes
- read locally or from memory

read your most recent write on x

read-own-write

\[
\begin{align*}
x &:= 1 \\
x &:= 2 \\
a &:= x \\
b &:= y
\end{align*}
\]

\[
\begin{align*}
x &:= 1 \\
x &:= 2 \\
a &:= x \\
b &:= y
\end{align*}
\]

\[
\begin{align*}
x &:= 0 \\
y &:= 1
\end{align*}
\]
Shared Variable

Concurrency

Total Store Order (TSO)

- non-atomic writes
- read locally or from memory

P1

x := 1
x := 2
a := 2
b := y

read your most recent write on x

read-own-write

store buffer

x := 1

P2

store buffer

y := 1

x := 0

shared variable

memory

shared variable
Shared Variable Concurrency

Total Store Order (TSO)

- non-atomic writes
- read locally or from memory

read your most recent write on x

read-own-write

store buffer

store buffer

memory

x=1
x=2
x=0
y=1

P₁

P₂

a:=x
b:=y

x:=1
x:=2
Shared Variable

Concurrency

Total Store Order (TSO)

- non-atomic writes
- read locally or from memory

\[ \text{P}_1 \]
\[ x := 1 \]
\[ x := 2 \]
\[ a := x \]
\[ b := y \]

\[ \text{P}_2 \]
\[ x := 1 \]
\[ x := 2 \]
\[ a := x \]
\[ b := y \]

\[ \text{Shared variable} \]
\[ \text{memory} \]
Shared Variable Concurrency

Total Store Order (TSO)

- non-atomic writes
- read locally or from memory

```
x=0
shared variable

y=1
shared variable
```

```
P1
x:=1
x:=2
a:=x //2
b:=y

P2
```

read-from-memory

store buffer

x=2
x=1

store buffer

memory

x=0
shared variable

y=1
shared variable

//2

read-from-memory

Shared Variable
Concurrency
Total Store Order (TSO)
• non-atomic writes
• read locally or from memory

\[ x = 1 \]
\[ x = 2 \]
\[ a := x // 2 \]
\[ b := y \]

\[ P_1 \rightarrow x = 2 \]
\[ x = 1 \]

\[ P_2 \rightarrow x = 0 \]
\[ y = 1 \]

no pending writes on \( y \)
read-from-memory
store buffer

\[ \text{store buffer} \]
\[ \text{memory} \]
\[ \text{shared variable} \]
\[ \text{shared variable} \]
Shared Variable
Concurrency

Total Store
Order (TSO)

• non-atomic writes
• read locally or from memory

no pending writes on y
read-from-memory

store buffer

store buffer

memory

shared variable

shared variable

\[
\begin{align*}
x &:= 1 \\
x &:= 2 \\
a &:= x \div 2 \\
b &:= 1
\end{align*}
\]

\[
\begin{align*}
P_1 &\rightarrow x = 2 \\
P_2 &\rightarrow x = 1
\end{align*}
\]

\[
\begin{align*}
x &:= 1 \\
x &:= 2 \text{\hspace{1cm} \text{//2}}
\end{align*}
\]

\[
\begin{align*}
\text{P}\_1 &\rightarrow a := x \\
\text{P}\_2 &\rightarrow b := y \\
\end{align*}
\]
Shared Variable

Concurrency

Total Store Order (TSO)

- non-atomic writes
- read locally or from memory

\[
\begin{align*}
x &:= 1 \\
x &:= 2 \\
a &:= x/2 \\
b &:= 1/1
\end{align*}
\]

\[
\begin{align*}
P_1 &\rightarrow x=2, x=1 \\
P_2 &\rightarrow x=0, y=1
\end{align*}
\]

store buffer

no pending writes on y

read-from-memory

x=1

x=2

1

1

memory

store buffer

shared variable

shared variable
Shared Variable

Concurrency

Total Store Order (TSO)

• non-atomic writes
• read locally or from memory

\[
\begin{align*}
  x &= 0 \\
  y &= 1 \\
  x &= 1 \\
  y &= 1 \\
  x &= 2 \\
  a &= x / 2 \\
  b &= y / 1
\end{align*}
\]
Shared Variable
Concurrency

Total Store
Order (TSO)

- non-atomic writes
- read locally or from memory

\[
\begin{align*}
x &= 0 \\
y &= 1
\end{align*}
\]

\[
\begin{align*}
x &= 1 \\
a &= x/2 \\
b &= y/1
\end{align*}
\]

\[
\begin{align*}
x &= 2 \\
x &= 1
\end{align*}
\]

\[
\begin{align*}
\text{store buffer} \\
\text{store buffer} \\
\text{memory update} \\
\text{memory variable} \\
\text{shared variable}
\end{align*}
\]
Shared Variable
Concurrency

Total Store Order (TSO)

- non-atomic writes
- read locally or from memory

\begin{align*}
x &:= 1 \\
x &:= 2 \\
a &:= x / 2 \\
b &:= y / 1 \\
\end{align*}
**Shared Variable Concurrency**

**Total Store Order (TSO)**
- non-atomic writes
- read locally or from memory

```
x=1
P_1
x:=1
a:=x//2
b:=y//1
P_2
x:=2
y:=1
```

```
//1
memory update
memory
shared variable
store buffer

//2
```
Shared Variable
Concurrency
Shared Variable
Concurrency

Sequential
Consistency (SC)
Shared Variable
Concurrency

Sequential
Consistency
(SC)

- atomic writes
- read from memory
Shared Variable
Concurrency

Sequential Consistency (SC)

- atomic writes
- read from memory

P1 → x=1
P2 → y=0
Shared Variable
Concurrency

Sequential Consistency (SC)

- atomic writes
- read from memory

```
P_1 → x=1
P_2 → y=0
```

- write-to-memory
- read-from-memory
Shared Variable
Concurrency

Sequential Consistency (SC)

- atomic writes
- read from memory

\[
\begin{array}{ll}
P_1 & x = 1 \\
P_2 & y = 0
\end{array}
\]

- write-to-memory
- read-from-memory
Shared Variable
Concurrency

Sequential Consistency (SC)

- atomic writes
- read from memory

$P_1 \rightarrow x=1$
$P_2 \rightarrow y=0$

- write-to-memory
- read-from-memory

sc
Shared Variable Concurrency

Total Store Order (TSO)

Sequential Consistency (SC)

- atomic writes
- read from memory

P₁ → x = 1
P₂ → y = 0

- write-to-memory
- read-from-memory
Shared Variable Concurrency

Total Store Order (TSO)

- non-atomic writes
- read locally or from memory

Sequential Consistency (SC)

- atomic writes
- read from memory

\[ \begin{align*}
P_1 & \rightarrow x = 1 \\
P_2 & \rightarrow y = 0
\end{align*} \]

- write-to-memory
- read-from-memory

\( sc \)
Shared Variable
Concurrent

Total Store
Order (TSO)

- non-atomic writes
- read locally or from memory

Sequential
Consistency
(SC)

- atomic writes
- read from memory

- write-to-memory
- read-from-memory

P1
x=1
y=0

P2
x=2
y=1

P1
x=1

P2
y=0

x:=1
x:=2
a:=x \div 2
b:=y \div 1
Shared Variable Concurrency

Total Store Order (TSO)

- non-atomic writes
- read locally or from memory

Sequential Consistency (SC)

- atomic writes
- read from memory

P1
- store buffer
- write-to-memory

P2
- store buffer
- read-from-memory

x := 1
x := 2
a := x // 2
b := y // 1

x := 2
y := 1

x := 2
y := 0

SC
### Shared Variable

**Concurrency**

- non-atomic writes
- read locally or from memory

**Total Store Order (TSO)**

- write-to-buffer
- read-own-write
- read-from-memory
- update

---

### Sequential Consistency (SC)

- atomic writes
- read from memory

**P1** → **x=1**

**P2** → **y=0**
Shared Variable Concurrency

- non-atomic writes
- read locally or from memory

Total Store Order (TSO)

- non-atomic writes
- read locally or from memory

Sequential Consistency (SC)

- atomic writes
- read from memory

• write-to-buffer
• read-own-write
• read-from-memory
• update

P<sub>1</sub>
x<sub>1</sub> := 1
x<sub>2</sub> := 2
a := x<sub>1</sub> / 2
b := y / 1

P<sub>2</sub>

store buffer

x<sub>1</sub> := 1
y<sub>0</sub> := 0

x<sub>2</sub> := 2
y<sub>1</sub> := 1

store buffer

TSO

SC
Shared Variable Concurrency

Total Store Order (TSO)
- non-atomic writes
- read locally or from memory

Sequential Consistency (SC)
- atomic writes
- read from memory

- write-to-buffer
- read-own-write
- read-from-memory
- update

Shared variable:
- \( x = 1 \)
- \( x = 2 \)
- \( a = \frac{x}{2} \)
- \( b = \frac{y}{1} \)

Concurrent execution:

- \( P_1 \):
  - Store buffer: \( x = 1 \)
  - Store buffer: \( x = 2 \)
  - Shared variable: \( a = \frac{x}{2} \)
  - Shared variable: \( b = \frac{y}{1} \)

- \( P_2 \):
  - Store buffer
  - Shared variable

Sequential execution:

- \( P_1 \):
  - \( x = 1 \)
  - \( y = 0 \)

- \( P_2 \):
  - \( x = 1 \)
  - \( y = 0 \)

Weak memory model:
- TSO
- Update

SC weak memory model:
Shared Variable Concurrency

Total Store Order (TSO)

- non-atomic writes
- read locally or from memory

Sequential Consistency (SC)

- atomic writes
- read from memory

extra behaviors

weak memory model

 TS O

write-to-memory

read-from-memory

write-to-buffer

read-own-write

read-from-memory

update
### Shared Variable Concurrency
- Total Store Order (TSO)
  - non-atomic writes
  - read locally or from memory

### Sequential Consistency (SC)
- atomic writes
- read from memory

#### Example
- P1: `x:=1` → `x:=2` → `a:=x//2` → `b:=y//1`
- P2: `x=2` → `y=1` → `x=2` → `y=1`
- P1: `x=1` → `y=0`
Dekker Protocol

\[ \begin{align*}
    &x:=1 \\
    &a:=y \\
    &\text{if } a=0 \text{ then } \text{CS}_1
\end{align*} \]

\[ \begin{align*}
    &y:=1 \\
    &b:=x \\
    &\text{if } b=0 \text{ then } \text{CS}_2
\end{align*} \]

not (CS\(_1\) and CS\(_2\))

\[ P_1 \rightarrow x=0 \quad y=0 \]

\[ P_2 \]

\[ \text{assertion} \]

\[ \text{potentially bad behaviors} \]

\[ \text{extra behaviors} \]
Dekker Protocol

\[ x := 1 \]
\[ a := y \]
\[ \text{if } a = 0 \text{ then } \]
\[ \text{CS}_1 \]
\[ y := 1 \]
\[ b := x \]
\[ \text{if } b = 0 \text{ then } \]
\[ \text{CS}_2 \]
\[ \text{not } (\text{CS}_1 \text{ and } \text{CS}_2) \]

\[ \text{assertion} \]

\[ \text{P}_1 \rightarrow \text{CS}_1 \rightarrow x = 0 \]
\[ \text{P}_2 \rightarrow \text{CS}_2 \rightarrow y = 0 \]

Shared Variable Concurrency

Total Store Order (TSO)

potentially bad behaviors
extra behaviors

Shared Variable
Concurrency

Total Store
Order (TSO)
\[
\begin{align*}
&x := 1 \\
a := y \\
&\text{if } a = 0 \text{ then} \\
&\text{CS}_1
\end{align*}
\]

\[
\begin{align*}
y := 1 \\
b := x \\
&\text{if } b = 0 \text{ then} \\
&\text{CS}_2
\end{align*}
\]

\[
\text{assertion} \\
\text{not (CS}_1 \text{ and CS}_2\text{)}
\]
Dekker Protocol

\[ x := 1 \]
\[ a := y \]
\[ \text{if } a = 0 \text{ then } \text{CS}_1 \]
\[ y := 1 \]
\[ b := x \]
\[ \text{if } b = 0 \text{ then } \text{CS}_2 \]

not (CS\(_1\) and CS\(_2\))

assertion

\[ x = 0 \]
\[ y = 0 \]

Shared Variable
Concurrency

Total Store
Order (TSO)
\textbf{Dekker Protocol} \\
\textbf{Shared Variable Concurrency} \hspace{1cm} \textbf{Total Store Order (TSO)} \\

\begin{itemize}
  \item $x := 1$
  \item $a := 0$
  \item if $a = 0$ then $CS_1$
  \item $y := 1$
  \item $b := x$
  \item if $b = 0$ then $CS_2$
  \item not ($CS_1$ and $CS_2$)
\end{itemize}

Assertion $P_1$:

- $x = 0$
- $y = 0$

\begin{itemize}
  \item potentially bad behaviors
  \item extra behaviors
\end{itemize}
\begin{align*}
  x &:= 1 \\
a &:= 0
\end{align*}

if $a = 0$ then $CS_1$

\begin{align*}
y &:= 1 \\
b &:= x
\end{align*}

if $b = 0$ then $CS_2$

not ($CS_1$ and $CS_2$)

Dekker Protocol

\begin{align*}
P_1 &\xrightarrow{x=1} P_2 & x &= 0 \\
P_2 &\xrightarrow{y=1} & y &= 0
\end{align*}

assertion

\textbf{Shared Variable Concurrency}

\textbf{Total Store Order (TSO)}

\textbf{Shared Variable}

\textbf{Concurrency}

\textbf{Total Store Order (TSO)}

\textbf{Dekker Protocol}

\textbf{Not (CS$_1$ and CS$_2$)}

extra behaviors

potentially bad behaviors
Dekker Protocol

- x := 1
- a := y // 0
- if a = 0 then
  - CS1

- y := 1
- b := x
- if b = 0 then
  - CS2

- not (CS1 and CS2)

assertion

Shared Variable
Concurrency

Total Store
Order (TSO)

P1

x = 1

x = 0

P2

y = 0

potentially bad behaviors

extra behaviors
Dekker Protocol

\[ x := 1 \]
\[ a := y \]  
\[ \text{if } a = 0 \text{ then} \]
\[ y := 1 \]
\[ b := x \]  
\[ \text{if } b = 0 \text{ then} \]
\[ \text{not } (CS_1 \text{ and } CS_2) \]

assertion

P1: \[ x = 1 \]  \[ x = 0 \]
P2: \[ x = 1 \]  \[ y = 0 \]

Shared Variable
Concurrency

Total Store
Order (TSO)

Extra behaviors

Potentially bad behaviors
Dekker Protocol

\[ x := 1 \]
\[ a := y \]
\[ \text{if } a = 0 \text{ then } CS_1 \]
\[ y := 1 \]
\[ b := x \]
\[ \text{if } b = 0 \text{ then } CS_2 \]
\[ \text{not } (CS_1 \text{ and } CS_2) \]

Assertion:

\[ P_1 \rightarrow x = 1 \rightarrow x = 0 \]
\[ P_2 \rightarrow y = 0 \]

Extra behaviors:

Potentially bad behaviors:
Dekker Protocol

\[ x := 1 \]
\[ a := y \]
\[ \text{if } a = 0 \text{ then} \]
\[ CS_1 \]
\[ y := 1 \]
\[ b := x \]
\[ \text{if } b = 0 \text{ then} \]
\[ CS_2 \]
\[ \text{not } (CS_1 \text{ and } CS_2) \]

assertion

P1: x = 1 → x = 0

P2: y = 1 → y = 0

potential bad behaviors

extra behaviors
Dekker Protocol

\[ x:=1 \]
\[ a:=y \]
\[ \text{if } a=0 \text{ then} \]
\[ \text{CS}_1 \]
\[ y:=1 \]
\[ b:=x \]
\[ \text{if } b=0 \text{ then} \]
\[ \text{CS}_2 \]
\[ \text{not (CS}_1 \text{ and CS}_2) \]

Assertion

\[ x=0 \]
\[ y=0 \]
Dekker Protocol

- $x := 1$
- $a := y // 0$
- if $a = 0$ then
  - CS

- $y := 1$
- $b := 0$
- if $b = 0$ then
  - CS

not (CS\text{1} and CS\text{2})

assertion

P\text{1} \rightarrow x = 1 \rightarrow x = 0

P\text{2} \rightarrow y = 1 \rightarrow y = 0

potential bad behaviors

extra behaviors

Shared Variable Concurrency

Total Store Order (TSO)
Dekker Protocol

\[
\begin{align*}
\text{x:=1} \\
\text{a:=y} //0 \\
\text{if a=0 then} \\
\text{CS}_1
\end{align*}
\]

\[
\begin{align*}
\text{y:=1} \\
\text{b:=x} //0 \\
\text{if b=0 then} \\
\text{CS}_2
\end{align*}
\]

\[
\text{not (CS}_1 \text{ and CS}_2)
\]

\[
\text{assertion}
\]

\[
\begin{align*}
\text{P}_1 & \quad \xrightarrow{\text{x=1}} \quad \xrightarrow{\text{x=0}} \\
\text{P}_2 & \quad \xrightarrow{\text{y=1}} \quad \xrightarrow{\text{y=0}} \\
\end{align*}
\]
Dekker Protocol

$x := 1$
$a := y // 0$
if $a = 0$ then
  $\text{CS}_1$

$y := 1$
$b := x // 0$
if $b = 0$ then
  $\text{CS}_2$

not ($\text{CS}_1$ and $\text{CS}_2$)

assertion

$P_1 \\ x=1 \rightarrow x=0$

$P_2 \\ y=1 \rightarrow y=0$

Dekker Protocol: Shared Variable Concurrency and Total Store Order (TSO)

potentially bad behaviors

extra behaviors
x := 1  
a := y  // 0  
if a = 0 then  
  → CS\textsubscript{1}

y := 1  
b := x  // 0  
if b = 0 then  
  → CS\textsubscript{2}

P\textsubscript{1} ↓ x = 1  
P\textsubscript{2} ↓ y = 1

x = 1  
y = 1  
// 0

\text{not (CS\textsubscript{1} and CS\textsubscript{2})}

\text{assertion}

\text{Dekker Protocol}
Dekker Protocol

\[
x := 1 \\
a := y //0 \\
\text{if } a = 0 \text{ then} \\
\downarrow CS_1
\]

\[
y := 1 \\
b := x //0 \\
\text{if } b = 0 \text{ then} \\
\downarrow CS_2
\]

not (CS_1 and CS_2)

assertion

\[
P_1 \rightarrow x = 1 \rightarrow x = 0 \\
P_2 \rightarrow y = 1 \rightarrow y = 0
\]
Dekker Protocol

\[ x := 1 \]
\[ a := y \]
\[ \text{if } a = 0 \text{ then} \]
\[ \text{CS}_1 \]
\[ y := 1 \]
\[ b := x \]
\[ \text{if } b = 0 \text{ then} \]
\[ \text{CS}_2 \]
\[ \text{assertion} \]

\[ \text{not (CS}_1 \text{ and CS}_2) \]

\[ x = 1 \]
\[ y = 1 \]
\[ x = 0 \]
\[ y = 0 \]

Shared Variable
Concurrency

Total Store
Order (TSO)

extra behaviors

potentially bad behaviors

weak memory model

extra behaviors

potentially bad behaviors
Dekker Protocol

```
x := 1
a := y // 0
if a = 0 then
    ▶ CS1

y := 1
b := x // 0
if b = 0 then
    ▶ CS2
```

**assertion**

```
not (CS1 and CS2)
```

**Shared Variable**

**Concurrency**

**Total Store Order (TSO)**

**extra behaviors**

**potentially bad behaviors**

**TSO**

**weak memory model**

**extra behaviors**

**potentially bad behaviors**
Dekker Protocol

\[x := 1\]
\[a := y \text{//0}\]
\[\text{if } a = 0 \text{ then} \]
\[\text{CS}_1\]

\[y := 1\]
\[b := x \text{//0}\]
\[\text{if } b = 0 \text{ then} \]
\[\text{CS}_2\]

\[\neg (\text{CS}_1 \text{ and } \text{CS}_2)\]

\[\text{assertion}\]

\[x = 1\]
\[y = 1\]

\[\text{P}_1\]

\[x = 0\]
\[y = 0\]

\[\text{P}_2\]

\[\text{TSO}\]

\[\text{weak memory model}\]

\[\text{extra behaviors}\]

\[\text{potentially bad behaviors}\]

\[\text{extra behaviors}\]
Dekker Protocol

x := 1
a := y // 0
if a = 0 then
  \(CS_1\)

y := 1
b := x // 0
if b = 0 then
  \(CS_2\)

not (\(CS_1\) and \(CS_2\))

assertion

SC

TSO

weak memory model

extra behaviors

potentially bad behaviors

Shared Variable Concurrency

Total Store Order (TSO)

x = 0
y = 0

//0

x = 1
y = 1

//0

TSO

potentially bad behaviors

extra behaviors

Shared Variable

Concurrency

Total Store

Order (TSO)
Dekker Protocol

x:=1
a:=y //0
if a=0 then
   CS1

y:=1
b:=x //0
if b=0 then
   CS2

not (CS1 and CS2)

assertion

P1: x=1 → x=0
P2: y=1 → y=0

SC: Total Store Order (TSO)

program repair

potentially bad behaviors

weak memory model

extra behaviors

extra behaviors

potentially bad behaviors
Dekker Protocol

\[ x := 1 \]
\[ a := y // 0 \]
\[ \text{if } a = 0 \text{ then } \]
\[ \triangleright CS_1 \]
\[ y := 1 \]
\[ b := x // 0 \]
\[ \text{if } b = 0 \text{ then } \]
\[ \triangleright CS_2 \]
\[ \text{not } (CS_1 \text{ and } CS_2) \]

Program repair

\[ \text{assertion} \]

\[ P_1 \]
\[ x = 1 \]
\[ P_2 \]
\[ y = 1 \]

\[ x = 0 \]
\[ y = 0 \]

Total Store Order (TSO)

Extra behaviors

Weak memory model

Potentially bad behaviors

Memory fences

SC

TSO

Extra behaviors

Potentially bad behaviors
Dekker Protocol

x := 1
a := y // 0
if a = 0 then
  CS1

y := 1
b := x // 0
if b = 0 then
  CS2

not (CS1 and CS2)

assertion

P1
x = 1
x = 0

P2
y = 1
y = 0

P1

P2

not (CS1 and CS2)

program repair

mem Fences

potentially bad behaviors

weak memory model

extra behaviors

extra behaviors

"strengthen the program behavior"

SC

TSO

TSO

weak memory model

potentially bad behaviors

SC

TSO

extra behaviors

extra behaviors

extra behaviors
Shared Variable
Concurrency

- Sequential Consistency (SC)
- Total Store Ordering (TSO)
- Memory Fences
- Non-monotonicity
- Load-Buffer Semantics
Shared Variable
Concurrency

- Sequential Consistency (SC)
- Total Store Ordering (TSO)
- Memory Fences
- Non-monotonicity
- Load-Buffer Semantics
Shared Variable

Concurrency

• Sequential Consistency (SC)
• Total Store Ordering (TSO) (Examples)
• Memory Fences
• Non-monotonicity
• Load-Buffer Semantics
Message Passing

\[ a = 1 \quad \text{and} \quad b = 0 \]

**Shared Variable Concurrency**

**Total Store Order (TSO)**
Message Passing

\[ x := 1 \]
\[ y := 1 \]
\[ a := y \]
\[ b := x \]

\[ a = 1 \text{ and } b = 0 \]

\[ P_1 \rightarrow x = 1 \rightarrow x = 0 \]
\[ P_2 \rightarrow y = 0 \]

Shared Variable Concurrency

Total Store Order (TSO)

Extra behaviors

Potentially bad behaviors
Shared Variable
Concurrency

Total Store
Order (TSO)

Message Passing

\[ a = 1 \text{ and } b = 0 \]

\[ x := 1 \]
\[ y := 1 \]
\[ a := y \]
\[ b := x \]

\[ P_1 \rightarrow x = 1 \rightarrow x = 0 \]
\[ P_2 \rightarrow x = 1 \rightarrow y = 0 \]

potentially bad behaviors
extra behaviors
Message Passing

x := 1
y := 1
a := y
b := x

P₁
y = 1  x = 1

P₂

x = 0
y = 0

a = 1 and b = 0
**Shared Variable Concurrency**

**Total Store Order (TSO)**

Message Passing

\[ a = 1 \] and \[ b = 0 \]

\[ P_1 \]

\[ P_2 \]

\[ x := 1 \]
\[ y := 1 \]
\[ a := y \]
\[ b := x \]

\[ x = 1 \]
\[ y = 1 \]

\[ a = 1 \text{ and } b = 0 \]

Potentially bad behaviors

Extra behaviors
Message Passing

Shared Variable
Concurrency

Total Store
Order (TSO)

a = 1
b = 0

P_1
P_2

x := 1
y := 1

x := y // 1
b := x

x = 1
y = 1

a = 1 and b = 0

potentially bad behaviors
extra behaviors
Message Passing

\[ a := y \]
\[ b := x \]
\[ x := 1 \]
\[ y := 0 \]
\[ y := 1 \]

\( a = 1 \) and \( b = 0 \)

**Shared Variable**

**Concurrency**

**Total Store**

**Order (TSO)**
Message Passing

a := 1
b := x

P1

x := 1
y := 1

P2

a := y
b := x

a = 1 and b = 0

x = 1
y = 1

Shared Variable
Concurrency

Total Store
Order (TSO)

extra behaviors
potentially bad behaviors
**Message Passing**

- $x := 1$
- $y := 1$
- $a := y ; x$
- $b := x \mid 1$

- $a = 1$ and $b = 0$
- Not allowed

**Total Store Order (TSO)**

- $P_1$
- $P_2$
- $x := 1$
- $y := 1$

**Potential Behaviors**

- Extra behaviors
- Potentially bad behaviors
Shared Variable
Concurrency

- Sequential Consistency (SC)
- Total Store Ordering (TSO)
- Memory Fences
- Non-monotonicity
- Load-Buffer Semantics
Shared Variable
Concurrency

- Sequential Consistency (SC)
- Total Store Ordering (TSO)
- Memory Fences
- Non-monotonicity
- Load-Buffer Semantics
Dekker Protocol

\[ x := 1 \]
\[ a := y \quad // 0 \]
\[ \text{if } a = 0 \text{ then} \quad \triangleright \text{CS}_1 \]
\[ y := 1 \]
\[ b := x \quad // 0 \]
\[ \text{if } b = 0 \text{ then} \quad \triangleright \text{CS}_2 \]

\[ P_1 \quad x = 1 \quad \rightarrow \quad x = 0 \quad P_2 \quad y = 1 \quad \rightarrow \quad y = 0 \]

not (CS\(_1\) and CS\(_2\))

assertion

SC

Total Store Order (TSO)

program repair

memory fences

potentially bad behaviors

weak memory model

extra behaviors

potentially bad behaviors

extra behaviors

"strengthen the program behavior"
Shared Variable
Concurrency

Total Store
Order (TSO)

Dekker Protocol

\[
\begin{align*}
&\text{x:=1} \\
&\text{mfence} \\
&\text{a:=y} \\
&\text{if a=0 then} \\
&\text{CS}_1
\end{align*}
\]

\[
\begin{align*}
&\text{P}_1 \\
&\text{x=0}
\end{align*}
\]

\[
\begin{align*}
&\text{y:=1} \\
&\text{mfence} \\
&\text{b:=x} \\
&\text{if b=0 then} \\
&\text{CS}_2
\end{align*}
\]

\[
\begin{align*}
&\text{P}_2 \\
&\text{y=0}
\end{align*}
\]

not (CS\textsubscript{1} and CS\textsubscript{2})

assertion

potentially bad behaviors

extra behaviors
Dekker Protocol

Shared Variable
Concurrency

Total Store
Order (TSO)

x:=1
mfence
a:=y
if a=0 then
CS1

y:=1
mfence
b:=x
if b=0 then
CS2

P1

memory fence

P2

x=0
y=0

not (CS1 and CS2)

assertion

potentially bad behaviors

extra behaviors
**Shared Variable Concurrency**

**Total Store Order (TSO)**

- `x := 1`  
- `y := 0`  
- `x := 1`  
- `y := 0`

**Dekker Protocol**

- `x := 1`  
- `mfence`  
- `a := y`  
- `if a = 0 then CS_1`  
- `y := 1`  
- `b := x`  
- `if b = 0 then CS_2`  
- `not (CS_1 and CS_2)`  
- `assertion`

- `P_1`  
- `P_2`  
- `x = 0`  
- `y = 0`  
- `memory fence`

**Enabled only on empty buffers**

**Extra behaviors**

**Potentially bad behaviors**
Shared Variable
Concurrency

Total Store
Order (TSO)

Dekker Protocol

\[ x := 1 \]
\[ mfence \]
\[ a := y \]
\[ if \ a = 0 \ then \ CS_1 \]

\[ y := 1 \]
\[ b := x \]
\[ if \ b = 0 \ then \ CS_2 \]

\[ not \ (CS_1 \ and \ CS_2) \]

assertion

\[ x = 0 \]
\[ y = 0 \]

memory fence

enabled only on empty buffers

potential bad behaviors

extra behaviors

enabled only on empty buffers
Shared Variable
Concurrency

Total Store
Order (TSO)

Dekker Protocol

\[
x := 1 \\
mfence \\
a := y \\
if a = 0 then \\
CS_1
\]

\[
y := 1 \\
mfence \\
b := x \\
if b = 0 then \\
CS_2
\]

 assertion

memory fence

enabled only on empty buffers

x = 0

y = 0

extra behaviors

potentially bad behaviors

not (CS_1 and CS_2)

memory fence enabled only on empty buffers
Shared Variable
Concurrency

Total Store
Order (TSO)

Dekker Protocol

\[ x := 1 \]
\[ \text{mfence} \]
\[ a := y \]
\[ \text{if } a = 0 \text{ then } \]
\[ \text{CS}_1 \]

\[ y := 1 \]
\[ \text{mfence} \]
\[ b := x \]
\[ \text{if } b = 0 \text{ then } \]
\[ \text{CS}_2 \]

\[ \text{not } (\text{CS}_1 \text{ and } \text{CS}_2) \]

\[ \text{assertion} \]

memory fence

enabled only on empty buffers

extra behaviors

potentially bad behaviors
Shared Variable
Concurrency

Total Store
Order (TSO)

Dekker Protocol

\[ x := 1 \]
\[ \text{mfence} \]
\[ a := y \]
\[ \text{if } a = 0 \text{ then } \]
\[ CS_1 \]

\[ P_1 \]
\[ x = 1 \]
\[ \rightarrow \]
\[ x = 0 \]
\[ y = 0 \]

\[ y := 1 \]
\[ \text{mfence} \]
\[ b := x \]
\[ \text{if } b = 0 \text{ then } \]
\[ CS_2 \]

\[ P_2 \]

not (\( CS_1 \) and \( CS_2 \))

memory fence

enabled only on empty buffers

extra behaviors

potentially bad behaviors

assertion

memory fence

enabled only on empty buffers
Dekker Protocol

```plaintext
x := 1
mfence
if a = 0 then
  P1
  x = 0
  y = 0
  mfence
  P2
  y := 1
  b := x
  if b = 0 then
    CS2
  else
    CS1

not (CS1 and CS2)

assertion
```

extra behaviors

potentially bad behaviors

memory fence

enabled only on empty buffers

Shared Variable

Concurrency

Total Store

Order (TSO)

x = 0

y = 0

extra behaviors
Dekker Protocol

```
x := 1
mfence
if a = 0 then
  CS
P1
  x := 0
y := 1
b := x
if b = 0 then
  CS
P2
  y := 0
```

assertion

not (CS₁ and CS₂)

memory fence

enabled only on empty buffers

Potential bad behaviors

Extra behaviors

Shared Variable

Concurrency

Total Store

Order (TSO)

x = 0

y = 0

x = 0

y = 0

extra behaviors

memory fence

enabled only on empty buffers

Potentially bad behaviors

Extra behaviors

Shared Variable

Concurrency

Total Store

Order (TSO)
Shared Variable
Concurrency

Total Store
Order (TSO)

Dekker Protocol

\[ x := 1 \]
\[ mfence \]
\[ a := y \]
\[ if \ a = 0 \ then \]
\[ CS_1 \]

\[ P_1 \]
\[ x = 1 \]
\[ y = 0 \]

\[ y := 1 \]
\[ mfence \]
\[ b := x \]
\[ if \ b = 0 \ then \]
\[ CS_2 \]

\[ P_2 \]

\[ not \ (CS_1 \ and \ CS_2) \]

memory fence

enabled only on empty buffers

not (CS_1 and CS_2)

assertion

potentially bad behaviors

extra behaviors

extra behaviors
Shared Variable
Concurrency

Total Store
Order (TSO)

Dekker Protocol

\[
x := 1 \\
mfence \\
a := y \\
if a = 0 then \\
CS_1 \\
\]

\[
y := 1 \\
mfence \\
b := x \\
if b = 0 then \\
CS_2 \\
\]

not \( (CS_1 \text{ and } CS_2) \)

memory fence

enabled only on empty buffers

Assertion

\text{potentially bad behaviors}

\text{extra behaviors}
Shared Variable Concurrency

Total Store Order (TSO)

Dekker Protocol

\[
x := 1 \\
mfence \\
a := 0 \\
if a = 0 then \\
CS_1 \\
P_1 \\
y := 1 \\
mfence \\
b := x \\
if b = 0 then \\
CS_2 \\
P_2 \\
not (CS_1 and CS_2) \\
assertion \\
\]

potential bad behaviors

extra behaviors

memory fence

enabled only on empty buffers

x = 1

y = 0
Shared Variable
Concurrency

Total Store
Order (TSO)

Dekker Protocol

\[
x := 1
\]

\[
fence
\]

If \( a = 0 \) then

\[
CS_1
\]

If \( b = 0 \) then

\[
CS_2
\]

\[
not (CS_1 \text{ and } CS_2)
\]

\[
\text{assertion}
\]

Memory fence

Enabled only on empty buffers

Extra behaviors

Potentially bad behaviors
**Shared Variable Concurrency**

**Total Store Order (TSO)**

**Dekker Protocol**

\[ x := 1 \]
\[ \text{mfence} \]
\[ \text{if } a = 0 \text{ then } \text{CS}_1 \]
\[ y := 1 \]
\[ \text{mfence} \]
\[ \text{if } b = 0 \text{ then } \text{CS}_2 \]

not (CS\(_1\) and CS\(_2\))

assertion

**Memory fence**

enabled only on empty buffers

**Extra behaviors**

potentially bad behaviors

x=1

y=0

P\(_1\) \rightarrow y=1

P\(_2\) \rightarrow x=1
Shared Variable Concurrency

Total Store Order (TSO)

Dekker Protocol

\[ x := 1 \]
\[ mfence \]
\[ a := y \] //0
\[ if \ a = 0 \ then \]
\[ CS_1 \]
\[ P_1 \] \[ y = 1 \]
\[ y := 1 \]
\[ b := x \]
\[ if \ b = 0 \ then \]
\[ CS_2 \]
\[ P_2 \]

\[ not (CS_1 \ and \ CS_2) \]

\[ assertion \]

Extra behaviors

potentially bad behaviors

memory fence

enabled only on empty buffers

\[ x = 1 \]

\[ y = 0 \]

\[ //0 \]

memory fence

enabled only on empty buffers
Dekker Protocol

Shared Variable Concurrency

Total Store Order (TSO)

\[ x := 1 \]
\[ m fence \]
\[ a := y \quad //0 \]
\[ \text{if } a = 0 \text{ then} \]
\[ \text{CS}_1 \]

\[ y := 1 \]
\[ m fence \]
\[ b := x \]
\[ \text{if } b = 0 \text{ then} \]
\[ \text{CS}_2 \]

\[ \text{not } (\text{CS}_1 \text{ and } \text{CS}_2) \]

assertion

memory fence

enabled only on empty buffers

\[ x = 1 \]

\[ y = 1 \]

extra behaviors

potentially bad behaviors
Dekker Protocol

\[ \begin{align*}
\text{x:=1} \\
mfence \\
a:=y //0 \\
\text{if a=0 then} \\
\text{CS}_1 \\
p_1 \rightarrow x=1 \\
y:=1 \\
mfence \\
\text{if b=0 then} \\
\text{CS}_2 \\
b:=x \\
p_2 \rightarrow y=1 \\
\text{not (CS}_1 \text{ and CS}_2) \\
\text{assertion}
\end{align*} \]
Shared Variable
Concurrency

Total Store
Order (TSO)

Dekker Protocol

\[ x := 1 \]
\[ mfence \]
\[ a := y \quad / / 0 \]
\[ \text{if } a = 0 \text{ then} \]
\[ \text{CS}_1 \]

\[ y := 1 \]
\[ mfence \]
\[ b := x \]
\[ \text{if } b = 0 \text{ then} \]
\[ \text{CS}_2 \]

\[ \text{not } (\text{CS}_1 \text{ and } \text{CS}_2) \]

Extra behaviors

Potentially bad behaviors

Assertion

Memory fence enabled only on empty buffers
**Shared Variable Concurrency**

**Total Store Order (TSO)**

**Dekker Protocol**

```plaintext
x := 1
mfence
if a = 0 then
  CS1
  P1
  y := 1
  mfence
  if b = 0 then
    CS2
    P2
    b := x

memory fence

enabled only on empty buffers

not (CS1 and CS2)

assertion
```

- **extra behaviors**
- **potentially bad behaviors**
Dekker Protocol

\[ x := 1 \]
\[ mfence \]
\[ a := y \quad // 0 \]
\[ if \ a = 0 \ then \]
\[ CS_1 \]
\[ \]
\[ y := 1 \]
\[ mfence \]
\[ b := x \quad // 1 \]
\[ if \ b = 0 \ then \]
\[ CS_2 \]

\[ not \ (CS_1 \ and \ CS_2) \]

\[ assertion \]

Extra behaviors

Potentially bad behaviors

Total Store Order (TSO)

Shared Variable

Concurrency

memory fence

enabled only on empty buffers

x\(\neq 1\)
y\(\neq 1\)
Shared Variable

Concurrency

Total Store Order (TSO)

Dekker Protocol

1. $x := 1$
2. $y := 1$
3. $y := 1$
4. $b := x$ // 1
5. if $b = 0$ then
   - CS$_2$
6. $a := y$ // 0
7. if $a = 0$ then
   - CS$_1$
8. $x := 1$

memory fence

enabled only on empty buffers

not (CS$_1$ and CS$_2$)

assertion

weak behaviors

potentially bad behaviors

extra behaviors

TSO

weak memory model

extra behaviors
Shared Variable Concurrency

Total Store Order (TSO)

Dekker Protocol

\[ x := 1 \]
\[ mfence \]
\[ \text{if } a = 0 \text{ then} \]
\[ CS_1 \]
\[ y := 1 \]
\[ mfence \]
\[ b := x \]
\[ \text{if } b = 0 \text{ then} \]
\[ CS_2 \]

not \((CS_1 \text{ and } CS_2)\)

assertion

enabled only on empty buffers

memory fence

program repair

weak memory model

TSO

potentially bad behaviors

extra behaviors

extra behaviors

potentially bad behaviors

weak memory model

program repair

memory fence

\[ x = 1 \]
\[ y = 1 \]
while true

x := 1
while true

\[ x := 1 \]

\[ y := 0 \]
while true

\[ x := 1 \]

\[ y := 0 \]
while true
p1

x:=1

p2

x:=0

y:=0
while true
  x := 1
  x := 1
  x := 1

while true
x:=1
x:=1
x:=1
x:=1
x:=0
y:=0
while true
x:=1

\(x=0\)
\(y=0\)
while true

P₁

P₂

x := 1

x := 1

y := 0

x := 0
while true

x := 1

y := 0

P₁

x := 1

P₂

x := 1

x := 1

x := 1

x := 1

y := 0

x := 0
while true

\[ x := 1 \]

\[ y := 0 \]

- unbounded
- FIFO
Shared Variable

Concurrency

- Sequential Consistency (SC)
- Total Store Ordering (TSO)
- Memory Fences
- Non-monotonicity
- Load-Buffer Semantics
Shared Variable

Concurrency

- Sequential Consistency (SC)
- Total Store Ordering (TSO)
- Memory Fences
- Non-monotonicity
- Load-Buffer Semantics
\text{Shared Variable Concurrency} \quad \text{Total Store Order (TSO)}

\begin{align*}
\text{extra behaviors} & \quad \text{potentially bad behaviors} \\
\text{assertion} & \end{align*}

\begin{align*}
a &= x \\
b &= y \\
\text{if } a &= 1 \text{ and } b = 0 \text{ then} \\
& z = 1 \\
\text{else} \\
& z = 2
\end{align*}

\begin{align*}
P_1 & \xrightarrow{x=1} P_2 \xrightarrow{y=0} x=0 \\
z = 1
\end{align*}
Shared Variable
Concurrency

Total Store
Order (TSO)

\[ x = 0 \]
\[ y = 0 \]

extra

potentially bad behaviors

extra behaviors

z = 1

assertion

\[ a := x \]
\[ b := y \]
\[ \text{if } a = 1 \text{ and } b = 0 \text{ then} \]
\[ z := 1 \]
\[ \text{else} \]
\[ z := 2 \]

P₁

P₂

x = 1

x = 0

y = 0
**Shared Variable Concurrency**

**Total Store Order (TSO)**

```plaintext
x = 0
y = 0

extra behaviors potentially bad behaviors

z = 1

a := x
b := y
if a = 1 and b = 0 then
  z := 1
else
  z := 2

P1

P2

z = 1

assertion

 potentials bad behaviors

extra behaviors
```

x = 1

y = 0
extra behaviors
potentially bad behaviors

```
a:=x  //1
b:=y
if a=1 and b=0 then
  z:=1
else
  z:=2
```

```
if a=1 and b=0 then
  z:=1
else
  z:=2
```
extra behaviors

potentially bad behaviors

a := x // 1
b := y
if a = 1 and b = 0 then
  z := 1
else
  z := 2

if a = 1 and b = 0 then
  z := 1
else
  z := 2

//1
assertion

$z = 1$

$\text{if } a = 1 \text{ and } b = 0 \text{ then}$

$\text{else}$

$z := 2$

$x := x$ // 1

$b := y$ // 0
x=1
y=0

extra behaviors
potentially bad behaviors

 assertion

P1

P2

x=1
y=0

z=1

if a=1 and b=0 then
  z:=1
else
  z:=2

a:=x
b:=y
a := x // 1
b := y // 0
if a = 1 and b = 0 then
  z := 1
else
  z := 2

if a = 1 and b = 0 then
  z := 1
x=1
y=0

z=1

P₁

a:=x //1
b:=y //0
if a=1 and b=0 then
  z:=1
else
  z:=2

P₂

z=1

assertion

potentially bad behaviors

extra behaviors
P_1: a:=x //1
    b:=y //0
    if a=1 and b=0 then
        z:=1
    else
        z:=2

P_2: x=0
    y=0

P_1: z=1

x=1

y=0

z=1

potentially bad behaviors

extra behaviors

assertion
a := x // 1
b := y // 0
if a = 1 and b = 0 then
  z := 1
else
  z := 2

if a = 1 and b = 0 then
  z := 1
else
  z := 2

assertion

potentially bad behaviors
extra behaviors
Shared Variable
Concurrency
Total Store Order (TSO)

x=0
y=0

extra behaviors potentially bad behaviors
assertion

\[ a := x \]
\[ b := y \]
\[ \text{if } a = 1 \text{ and } b = 0 \text{ then} \]
\[ \quad z := 1 \]
\[ \text{else} \]
\[ \quad z := 2 \]

\[ x = 1 \]
\[ y = 1 \]

\[ z = 1 \]

\[ x = 0 \]
\[ y = 0 \]

\[ z = 1 \]

assertion

\[ x = 1 \]
\[ y = 1 \]

\[ x = 1 \]
\[ y = 1 \]

\[ ⊑ \]
Shared Variable
Concurrency

Total Store
Order (TSO)

\[ x = 0 \]
\[ y = 0 \]

\[
\begin{align*}
& a := x \quad // 1 \\
& b := y \quad // 0 \\
& \text{if } a = 1 \text{ and } b = 0 \text{ then} \\
& \quad z := 1 \\
& \text{else} \\
& \quad z := 2
\end{align*}
\]

\[ z = 1 \]

//1
//0

\[
\begin{align*}
& P_1 \quad x = 1 \quad y = 1 \\
& P_2 \quad x = 0 \quad y = 0
\end{align*}
\]

\[ x = 0 \quad y = 0 \]

\[ x = 1 \quad y = 1 \]

\[ x = 1 \quad y = 1 \]

potentially bad behaviors

extra behaviors

assertion
Shared Variable

Concurrency

Total Store Order (TSO)

x=1

y=0

extra behaviors

potentially bad behaviors

assertion

P_1

P_2

a:=x
b:=y
if a=1 and b=0 then
  z:=1
else
  z:=2

z=1

a:=x
b:=y
if a=1 and b=0 then
  z:=1
else
  z:=2

z=1

assertion

x=1

y=0

x=1

y=1

x=0

y=0

x=1

y=1

⊑
x=0  y=0

extra behaviors potentially bad behaviors

z=1

a:=x
b:=y
if a=1 and b=0 then
  z:=1
else
  z:=2

P1
P2

z=1

assertion

x=1  y=1

P
P2

x=0  y=1

x=1  y=1
```
\begin{align*}
    & a := x \\
    & b := y \\
    & \text{if } a = 1 \text{ and } b = 0 \text{ then} \\
    & \quad z := 1 \\
    & \text{else} \\
    & \quad z := 2 \\
\end{align*}
```
x=1

y=1

z=1

a:=x
b:=y
if a=1 and b=0 then
  z:=1
else
  z:=2

assertion

P1

P2

x=1

y=1

z=1

potentially bad behaviors

extra behaviors
Shared Variable
Concurrency

Total Store
Order (TSO)

\[ x = 1 \quad y = 1 \]

\[ z = 1 \]

Extra behaviors potentially bad behaviors

\[ a := x \quad \text{//1} \]
\[ b := y \]
\[ \text{if } a = 1 \text{ and } b = 0 \text{ then} \]
\[ z := 1 \]
\[ \text{else} \]
\[ z := 2 \]

\[ z = 1 \]

assertion
Shared Variable
Concurrency

Total Store
Order (TSO)

\[ x=1 \]

\[ y=1 \]

\[ z=1 \]

\[ a := x \]
\[ b := y \]
\[ \text{if } a=1 \text{ and } b=0 \text{ then} \]
\[ z := 1 \]
\[ \text{else} \]
\[ z := 2 \]

\[ z=1 \]

\[ \text{assertion} \]

potentially bad behaviors

extra behaviors
extra behaviors

potentially bad behaviors

z := 1

P₁

P₂

a := x // 1
b := y // 1
if a = 1 and b = 0 then
  z := 1
else
  z := 2

if a = 1 and b = 0 then
  z := 1
else
  z := 2

assertion

\[
\begin{align*}
x &= 1 & y &= 1 \\
z &= 1 \\
\end{align*}
\]
Shared Variable
Concurrency

Total Store
Order (TSO)

\[ x = 1 \quad \subseteq \quad x = 1, y = 1 \]

\[ z = 1 \]

\[ \text{P}_1 \quad \rightarrow \quad x = 1 \]
\[ \text{P}_2 \quad \rightarrow \quad y = 1 \]

\[ a := x \quad // 1 \]
\[ b := y \quad // 1 \]

\[ \text{if } a = 1 \text{ and } b = 0 \text{ then} \]
\[ \quad z := 1 \]
\[ \text{else} \]
\[ \quad z := 2 \]

\[ z = 1 \]

\[ \text{assertion} \]

potentially bad behaviors

extra behaviors
Shared Variable
Concurrency

Total Store
Order (TSO)

\[ x = 1 \]
\[ y = 1 \]
\[ z = 1 \]

```
a := x; // 1
b := y; // 1
if a = 1 and b = 0 then
  z := 1
else
  z := 2
```

\[ // 1 \]
\[ x = 1 \]
\[ y = 1 \]

\[ z = 1 \]

assertion

tenatively bad behaviors

extra behaviors
```plaintext
P_1: a:=x //1
    b:=y //1
    if a=1 and b=0 then
        z:=1
    else
        z:=2

P_2: x:=1
y:=1

z:=1
```

**Shared Variable Concurrency**

**Total Store Order (TSO)**

- **x=1**
- **y=1**
- **z=1**

**Extra Behaviors**

**Potentially Bad Behaviors**
Shared Variable Concurrency

Total Store Order (TSO)

\[ x = 1 \]
\[ y = 1 \]

\[ x = 1 \]
\[ y = 1 \]

\[ z = 1 \]

\[ P_1 \]
\[ a := x \]
\[ b := y \]
\[ \text{if } a = 1 \text{ and } b = 0 \text{ then} \]
\[ z := 1 \]
\[ \text{else} \]
\[ z := 2 \]

\[ P_2 \]

\[ x = 1 \]
\[ y = 1 \]

non-monotonicity

potentially bad behaviors

extra behaviors

assertion

x=1

y=1

z=1

Shared Variable
Concurrency

- Sequential Consistency (SC)
- Total Store Ordering (TSO)
- Memory Fences
- Non-monotonicity
- Load-Buffer Semantics
Shared Variable
Concurrency

- Sequential Consistency (SC)
- Total Store Ordering (TSO)
- Memory Fences
- Non-monotonicity
- Load-Buffer Semantics
Shared Variable
Concurrency

Total Store
Order (TSO)

process $P_1$
load buffer
shared variable
load buffer
memory

process $P_2$
load buffer
shared variable
load buffer
memory

$x=0$
$y=0$
Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

Process $P_1$
Load Buffer
$x = 0$
Shared Variable

Process $P_2$
Load Buffer
$y = 0$
Shared Variable

Memory
Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

process $P_1$
load buffer
process $P_2$
load buffer

load buffer
memory
shared variable

$x = 0$
$y = 0$
Shared Variable Concurrency

Total Store Order (TSO)

Load-Buffer Semantics

process $P_1$

process $P_2$

Load Buffer

store buffer

load buffer

memory

shared variable

$x = 0$

$y = 0$
Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

process $P_1$

load buffer

load buffer

process $P_2$

x=0

y=0

shared variable

memory

shared variable

shared variable
Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

{x:=1
a:=x
b:=y
x:=2
c:=x
}

P_1

load
buffer

P_2

load
buffer

x=0
y=0

shared
variable

memory

load
buffer

shared
variable

shared
variable

Load-Buffer
Semantics
Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

\[ x := 1 \]
\[ a := x \]
\[ b := y \]
\[ x := 2 \]
\[ c := x \]

\[ P_1 \]
\[ P_2 \]

load buffer

write-to-memory

load buffer

memory

load buffer

shared variable

shared variable

shared variable

x=0

y=0

x=0

shared variable

shared variable

write-to-memory

Load-Buffer
Semantics
Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

- update memory
- propagate-self-message

• write-to-memory

memory

x=0

shared
variable

y=0

shared
variable

P1

P2

load
buffer

load
buffer

x:=1

a:=x

b:=y

x:=2

c:=x

load
buffer

memory

load
buffer

Update memory

Propagate self-message

• Load-Buffer Semantics

• Total Store Order (TSO)

• Shared Variable Concurrency
Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

\[\begin{align*}
\text{x:=1} & \\
\text{a:=x} & \\
\text{b:=y} & \\
\text{x:=2} & \\
\text{c:=x} & \\
\end{align*}\]

\[\begin{align*}
\text{P}_1 & \quad \text{load buffer} \\
\text{P}_2 & \quad \text{load buffer} \\
\text{x=1} & \quad \text{write-to-memory} \\
\text{y=0} & \quad \text{update memory} \\
\end{align*}\]
Shared Variable Concurrency

Total Store Order (TSO)

Load-Buffer Semantics

- update memory
- propagate-self-message

load buffer

write-to-memory

x=1
y=0

P₁

P₂

x:=1
a:=x
b:=y
x:=2
c:=x

load buffer

shared variable

memory

shared variable

shared variable

Load-Buffer Semantics
**Shared Variable Concurrency**

**Total Store Order (TSO)**

**Load-Buffer Semantics**

- `x=1`  
  `y=0`  

- `P_1`  
  `P_2`  

- `x:=1`  
  `a:=x`  
  `b:=y`  
  `x:=2`  
  `c:=x`  

- `load`  
  `buffer`  

- `write-to-memory`  
  `update memory`  
  `propagate-self-message`  

- `memory`  
  `shared variable`  

- `load buffer`  
  `load buffer`  

- `x=1`  
  `y=0`  

- `a:=x`  
  `b:=y`  
  `x:=2`  
  `c:=x`
**Shared Variable**

- Concurrency

**Total Store Order (TSO)**

- Total Store Order

**Load-Buffer Semantics**

- Update memory
- Propagate self-message

- Write-to-memory

- Write-to-memory

```
P1
x:=1
a:=x
b:=y
x:=2
c:=x
```

```
P2
x:=1
y:=0
```

- Load buffer
- Memory
- Shared variable
**Shared Variable Concurrency**

**Total Store Order (TSO)**

**Load-Buffer Semantics**

- Write-to-memory

```
\[
\begin{align*}
\text{x} &:= 1 \\
a &:= \text{x} \\
b &:= \text{y} \\
x &:= 2 \\
c &:= \text{x}
\end{align*}
\]
**Shared Variable Concurrency**

**Total Store Order (TSO)**

**Load-Buffer Semantics**

- **write-to-memory**

```
x:=1
a:=x
b:=y
x:=2
c:=x
```

```
x:=1
y:=0
s: self
```

- **P1**
  - x=1
  - P2
    - x=2
    - c:=x

- **P1**
  - load buffer
  - s: self

- **P2**
  - load buffer
- **Shared Variable**
  - `x` and `y` are shared variables.

- **Concurrency**
  - Processes `P_1` and `P_2` access the shared variables.

- **Total Store Order (TSO)**
  - Load-buffer semantics:
    - `x=1` and `y=0` are stored in the load buffer.
    - `x` is updated to 1 by `P_1`.
    - `y` is updated to 0 by `P_2`.
    - `x` is updated to 2 by `P_1`.
    - `c=x` is performed by `P_2`.

- **Load-Buffer Semantics**
  - `s: self` indicates a self-update operation.
  - `propagate-other-message` signals the completion of the write-to-memory operation.
Shared Variable Concurrency

Total Store Order (TSO)

Load-Buffer Semantics

- `x:=1`  
  - `a:=x`
  - `b:=y`
  - `x:=2`
  - `c:=x`

- `x=1`  
  - `s: self`

- `y=0`
  - `shared variable`
  - `propagate-other-message`

- `write-to-memory`

**Diagram:**

- `P_1`  
  - `load buffer`
  - `x=1`  
    - `a:=x`  
    - `b:=y`
  - `x:=2`
  - `c:=x`

- `P_2`  
  - `load buffer`
  - `y=0`
    - `shared variable`
    - `memory`
**Shared Variable**

**Concurrency**

**Total Store Order (TSO)**

**Load-Buffer Semantics**

- **write-to-memory**
Shared Variable
Concurrency
Total Store Order (TSO)
Load-Buffer Semantics

\[ \begin{align*}
x &:= 1 \\
a &:= x \\
b &:= y \\
x &:= 2 \\
c &:= x
\end{align*} \]

\[ \begin{align*}
x &:= 1^s \\
 y &:= 0^o
\end{align*} \]

\[ \begin{align*}
\text{load buffer} \\
\text{memory} \\
\text{propagate-other-message}
\end{align*} \]

\[ \begin{align*}
\alpha &: \text{other}
\end{align*} \]

\textit{write-to-memory}
Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

- write-to-memory
- propagate-other-message
**Shared Variable**

**Concurrency**

**Total Store Order (TSO)**

**Load-Buffer Semantics**

- **write-to-memory**
- **propagate-other-message**
• write-to-memory
• propagate-other-message
• write-to-memory
• propagate-other-message
• write-to-memory
• propagate-other-message
Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

- write-to-memory
- propagate-other-message
- write-to-memory
- propagate-other-message
**Shared Variable**

**Concurrency**

**Total Store Order (TSO)**

**Load-Buffer Semantics**

- read the most recent one
- read-own-write
- load buffer
- write-to-memory
- propagate-other-message
**Shared Variable**

**Concurrency**

**Total Store Order (TSO)**

**Load-Buffer Semantics**

- read the most recent one

- read-own-write

- write-to-memory

- propagate-other-message

---

**Load-Buffer Semantics**

- load buffer
- memory

- shared variable

---

**Example Sequence**

1. $x := 1$
2. $a := x$
3. $b := y$
4. $x := 2$
5. $c := x$
6. $y := 0$
7. loads buffer
8. loads buffer
9. $x := 1$
10. $y := 0$
11. loads buffer
12. loads buffer
13. shared variable
14. shared variable

---

**Execution Timeline**

- $P_1$
  - $x := 1$
  - $a := x$
  - $b := y$
  - $x := 2$
  - $c := x$
  - $y := 0$
  - loads buffer
  - loads buffer
- $P_2$
  - reads $x = 1$
  - reads $y = 0$
  - reads $x = 1$
  - reads $y = 0$
  - loads buffer
  - loads buffer
  - shared variable
  - shared variable
Shared Variable

Concurrency

Total Store Order (TSO)

Load-Buffer Semantics

- write-to-memory
- propagate-other-message

• read the most recent one

read-own-write

- load buffer
- write-to-memory
- propagate-other-message

\[\begin{align*}
\text{P}_1 & \quad x:=1, a:=x, //1 \\
\text{P}_2 & \quad x:=2, c:=x
\end{align*}\]
Shared Variable

Concurrency

Total Store

Order (TSO)

Load-Buffer

Semantics

- read the most recent one
- read-own-write

\[ x=1 \]

\[ a=x \]

\[ b=y \]

\[ x=2 \]

\[ c=x \]

\[ y=0 \]

\[ y=0 \]

\[ x=1 \]

\[ y=0 \]

\[ y=0 \]

- write-to-memory
- propagate-other-message
- read-own-write

Load-Buffer Semantics
Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

- write-to-memory
- propagate-other-message
- read-own-write
Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

- write-to-memory
- propagate-other-message
- read-own-write

\[
\begin{align*}
x &:= 1 \\
a &:= x \quad \text{//1} \\
b &:= y \\
x &:= 2 \\
c &:= x
\end{align*}
\]
Shared Variable Concurrency

Total Store Order (TSO)

Load-Buffer Semantics

- at the head
- delete-message
- load buffer
- load buffer
- memory

- write-to-memory
- propagate-other-message
- read-own-write

\[ x := 1 \]
\[ a := x \]
\[ b := y \]
\[ x := 2 \]
\[ c := x \]

\[ y := 0 \]

\[ x := 1 \]
\[ y := 0 \]
- write-to-memory
- propagate-other-message
- read-own-write
- delete
Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

- write-to-memory
- propagate-other-message
- read-own-write
- delete

```
shared variable
x=1
y=0

P1
x:=1
a:=x //1
b:=y
x:=2
c:=x

load buffer

P2
y=0

x=1
y=0

delete
message

load
buffer

memory

shared variable
```

at the head
Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

- write-to-memory
- propagate-other-message
- read-own-write
- delete
- write-to-memory
- propagate-other-message
- read-own-write
- delete
• write-to-memory
• propagate-other-message
• read-own-write
• delete
Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

- at the head
- read-message
- load buffer
- y=0
- load buffer
- memory
- shared variable
- shared variable

- write-to-memory
- propagate-other-message
- read-own-write
- delete
Shared Variable

Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

\[ \begin{align*}
    \text{at the head} \\
    \text{read-message} \\
    \text{load buffer} \\
    \text{shared variable} \\
    \text{write-to-memory} \\
    \text{propagate-other-message} \\
    \text{read-own-write} \\
    \text{delete} \\
    \text{read-buffer-head}
\end{align*} \]
Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

- write-to-memory
- propagate-other-message
- read-own-write
- delete
- read-buffer-head
Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

x=1
shared variable
y=0
shared variable

P1
load buffer
y=0

P2
load buffer
x=1

x:=1
a:=x //1
b:=y //0
x:=2
c:=x

• write-to-memory
• propagate-other-message
• read-own-write
• delete
• read-buffer-head
**Shared Variable**

- **Concurrency**

  - **Total Store Order (TSO)**

  - **Load-Buffer Semantics**

- **Example**

  ```python
  x:=1
  a:=x //1
  b:=y //0
  x:=2
  c:=x
  ```

  - **P1**
    - load buffer
    - y=0
  - **P2**
    - load buffer
    - y=0

- **Semantics**

  - **write-to-memory**
  - **propagate-other-message**
  - **read-own-write**
  - **delete**
  - **read-buffer-head**
**Shared Variable Concurrency**

**Total Store Order (TSO)**

**Load-Buffer Semantics**

- write-to-memory
- propagate-other-message
- read-own-write
- delete
- read-buffer-head
**Shared Variable Concurrency**

**Total Store Order (TSO)**

**Load-Buffer Semantics**

1. `x:=1`  
2. `a:=x //1`  
3. `b:=y //0`  
4. `x:=2`  
5. `c:=x`

**Process P1:**
- `y:=0`
- `x:=2`

**Process P2:**
- `y:=0`
- `x:=2`

- **write-to-memory**
- **propagate-other-message**
- **read-own-write**
- **delete**
- **read-buffer-head**
Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

- write-to-memory
- propagate-other-message
- read-own-write
- delete
- read-buffer-head
### Shared Variable Concurrency

#### Total Store Order (TSO)

#### Load-Buffer Semantics

- write-to-memory
- propagate-other-message
- **read-own-write**
- delete
- read-buffer-head

---

**Example:**

```plaintext
P1
x:=1
a:=x  //1
b:=y  //0
dx:=2
c:=x  //2

P2
y:=0
x:=2
y:=0
```

**Notes:**

- `x:=1`: Load from memory, propagates to `P2`
- `a:=x`: Propagates to `P2`
- `b:=y`: Reads own write, propagates to `P2`
- `x:=2`: Load from memory, writes to memory, propagates to `P2`
- `c:=x`: Reads own write, propagates to `P2`
**Shared Variable Concurrency**

- **Total Store Order (TSO)**

- **Load-Buffer Semantics**

- **Example Code**

  ```plaintext
  x:=1
  a:=x //1
  b:=y //0
  x:=2
  c:=x //2
  ```

- **Load-Buffer Operations**
  - write-to-memory
  - propagate-other-message
  - read-own-write
  - delete
  - read-buffer-head
**Shared Variable**

- **Concurrency**

**Total Store Order (TSO)**

**Load-Buffer Semantics**

- write-to-memory
- propagate-other-message
- read-own-write
- **delete**
- read-buffer-head
**Shared Variable Concurrency**

**Total Store Order (TSO)**

**Load-Buffer Semantics**

- write-to-memory
- propagate-other-message
- read-own-write
- delete
- read-buffer-head
**Shared Variable Concurrency**

**Total Store Order (TSO)**

**Load-Buffer Semantics**

- write-to-memory
- propagate-other-message
- read-own-write
- delete
- read-buffer-head
Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

- write-to-memory
- propagate-other-message
- read-own-write
- **delete**
- read-buffer-head
- write-to-memory
- propagate-other-message
- read-own-write
- delete
- read-buffer-head
<table>
<thead>
<tr>
<th>Total Store Order (TSO)</th>
<th>Load-Buffer Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>write-to-buffer</td>
<td>write-to-memory</td>
</tr>
<tr>
<td>read-own-write</td>
<td>propagate-other-message</td>
</tr>
<tr>
<td>read-from-memory</td>
<td>read-own-write</td>
</tr>
<tr>
<td>update</td>
<td>delete</td>
</tr>
<tr>
<td></td>
<td>read-buffer-head</td>
</tr>
</tbody>
</table>
Shared Variable
Concurrency

- Sequential Consistency (SC)
- Total Store Ordering (TSO)
- Memory Fences
- Non-monotonicity
- Load-Buffer Semantics
Shared Variable
Concurrency

- Sequential Consistency (SC)
- Total Store Ordering (TSO)
- Memory Fences
- Non-monotonicity
- Load-Buffer Semantics
Shared Variable
Concurrency

- Sequential Consistency (SC)
- Total Store Ordering (TSO)
- Memory Fences
- Non-monotonicity
- Load-Buffer Semantics (Examples)
Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

\[ x = 0 \]
\[ y = 0 \]

\textbf{Dekker Protocol}

\[ x := 1 \]
\[ a := y \]
\[ \text{if } a = 0 \text{ then } \]
\[ \text{CS}_1 \]

\[ y := 1 \]
\[ b := x \]
\[ \text{if } b = 0 \text{ then } \]
\[ \text{CS}_2 \]

\text{not (CS}_1 \text{ and CS)}

\textbf{assertion}

\textbf{potentially bad behaviors}

\textbf{extra behaviors}
**Shared Variable Concurrency**

**Total Store Order (TSO)**

**Load-Buffer Semantics**

---

**Dekker Protocol**

\[
x := 1 \\
y := 1 \\
\text{if } a = 0 \text{ then } \\
\text{CS}_1 \\
\text{if } b = 0 \text{ then } \\
\text{CS}_2 \\
\text{not (CS}_1 \text{ and CS)}
\]

---

**Assertion**  

\[
P_1 \\
\text{P}_2 \\
x = 0 \\
y = 0
\]
Dekker Protocol

\[ x := 1 \]
\[ a := y \]
\[ \text{if } a = 0 \text{ then } \]
\[ \text{CS}_1 \]

\[ y := 1 \]
\[ b := x \]
\[ \text{if } b = 0 \text{ then } \]
\[ \text{CS}_2 \]

\[ \text{not (CS}_1 \text{ and CS)} \]

assertion

\[ P_1 \Rightarrow y = 0, x = 0 \]

\[ P_2 \Rightarrow x = 0, y = 0 \]

potentially bad behaviors
extra behaviors
Dekker Protocol

\[ x := 1 \]
\[ a := y \]
\[ \text{if } a = 0 \text{ then } CS_1 \]

\[ y := 1 \]
\[ b := x \]
\[ \text{if } b = 0 \text{ then } CS_2 \]

not (CS₁ and CS₂)

assertion

\[ P_1 \]
\[ y = 0^0 \]
\[ x = 0 \]

\[ P_2 \]
\[ x = 0^0 \]
\[ y = 0 \]

potentially bad behaviors

extra behaviors
Dekker Protocol

\[\begin{align*}
\text{x} &:= 1 \\
\text{a} &:= \text{y} \\
\text{if a} &= 0 \text{ then} \\
&\text{CS}\_1 \\
\text{y} &:= 1 \\
\text{b} &:= \text{x} \\
\text{if b} &= 0 \text{ then} \\
&\text{CS}\_2 \\
&\text{not (CS}\_1\text{ and CS}\_2) \\
\end{align*}\]
Dekker Protocol

\[ x := 1 \]
\[ a := y \]
\[ \text{if } a = 0 \text{ then } \]
\[ \text{CS}_1 \]

\[ y := 1 \]
\[ b := x \]
\[ \text{if } b = 0 \text{ then } \]
\[ \text{CS}_2 \]

not (\text{CS}_1 \text{ and } \text{CS})

\[ x = y^0 \]
\[ y = 0 \]
\[ x = 1 \]

potentially bad behaviors

extra behaviors

assertion
The Dekker protocol is a solution to the mutual exclusion problem in concurrent programming. It uses two shared variables, `x` and `y`, and two critical sections, `CS1` and `CS2`, to ensure that only one process can enter a critical section at a time.

Here is the Dekker protocol in detail:

1. **P1**: Sets `x := 1`, `a := y`, and then checks if `a = 0`. If true, enters `CS1`.
2. **P2**: Sets `y := 1`, `b := x`, and then checks if `b = 0`. If true, enters `CS2`.

The assertions are:

- `P1`: `x = 1` and `y = 0` at the end.
- `P2`: `y = 1` and `x = 0` at the end.

The key property is that `not (CS1 and CS2)` ensures mutual exclusion.

Extra behaviors: The protocol can potentially lead to deadlocks or other undesirable behaviors if not properly implemented.
Dekker Protocol

\[
x := 1
\]
\[
a := y
\]
\[
\text{if } a = 0 \text{ then } CS_1
\]
\[
y := 1
\]
\[
b := x
\]
\[
\text{if } b = 0 \text{ then } CS_2
\]
\[
\text{not } (CS_1 \text{ and } CS_2)
\]

Assertion

Extra behaviors

Potentially bad behaviors
Dekker Protocol

1. $x := 1$
2. $a := y$
3. if $a = 0$ then
   1. $y := 1$
   2. $b := x$
   3. if $b = 0$ then
      1. $x := 1$

Assertion $P_1$ and $P_2$

Extra behaviors

Potentially bad behaviors

Not $(CS_1$ and $CS_2$)
**Dekker Protocol**

- **Shared Variable**
- **Concurrency**
- **Total Store Order (TSO)**
- **Load-Buffer Semantics**

**Dekker Protocol**

1. **P₁**
   - `x := 1`
   - `a := y`
   - `if a = 0 then
     - CS₁`
   - `y := 1`
   - `b := x`
   - `if b = 0 then
     - CS₂`

2. **P₂**
   - `x := 0`
   - `y := 1`

**Assertion**

- `not (CS₁ and CS₂)`

**Potential**

- Bad behaviors
- Extra behaviors
Dekker Protocol

\[
\begin{align*}
&x := 1 \\
&a := y \\
&\text{if } a = 0 \text{ then} \\
&\text{CS}_1 \\
&y := 1 \\
&b := x \\
&\text{if } b = 0 \text{ then} \\
&\text{CS}_2
\end{align*}
\]

\[
\text{not (CS}_1 \text{ and CS}_2)
\]

Assertion

P1

P2

\[
\begin{align*}
&y = 0^o \\
&x = 1^s \\
&x = 1 \\
&y = 1 \\
&x = 0^o \\
&y = 1^s \\
&x = 0 \\
&y = 1
\end{align*}
\]

potentially bad behaviors

extra behaviors
Dekker Protocol

\[ x := 1 \]
\[ a := y \]
\[ \text{if } a = 0 \text{ then } CS_1 \]

\[ y := 1 \]
\[ b := x \]
\[ \text{if } b = 0 \text{ then } CS_2 \]

\[ \text{not } (CS_1 \text{ and } CS) \]

\[ \text{assertion} \]
Dekker Protocol

```
x:=1
a:=y
if a=0 then
   CS1
y:=1
b:=x
if b=0 then
   CS2
```

Assertion

```
not (CS1 and CS)
```

Shared Variable

Concurrency

Total Store

Order (TSO)

Load-Buffer

Semantics

Load-Buffer Semantics

`x=1`

`y=0`

`s`

`y=0`

`x=1`

`s`

`x=1`

`y=1`

`s`

extra behaviors

potentially bad behaviors
x := 1
a := y // 0
if a = 0 then
   CS
1
y := 1
b := x
if b = 0 then
   CS
2
not (CS
1
and
CS
2)
assertion

Dekker Protocol

Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

P1
y := 0
x := 1s
P2
x := 0
y := 1s
x := 1
y := 1

potential bad behaviors

extra behaviors

Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

x := 1
y := 1

extra behaviors
x := 1
a := y // 0
if a = 0 then
  CS1
y := 1
b := x
if b = 0 then
  CS2

not (CS1 and CS)

assertion

P1

P2

x = 1
y = 1

y = 0
x = 1s

x = 0
y = 1

x = 0
y = 0

Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

Dekker Protocol

potentially bad behaviors

extra behaviors
Dekker Protocol

Shared Variable Concurrency
Total Store Order (TSO)
Load-Buffer Semantics

$x := 1$
$a := y$ if $a = 0$ then $CS_1$

$y := 1$
$b := x$ if $b = 0$ then $CS_2$

not ($CS_1$ and $CS_2$)

assertion

$P_1$ $y = 0$, $x = 1$ (s)

$P_2$ $x = 0$, $y = 1$ (s)

$x = 1$, $y = 1$

potentially bad behaviors
extra behaviors
Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

Dekker Protocol

\(x := 1\)
\(a := y \text{ //0}\)

if \(a = 0\) then
\(CS_1\)

\(y := 1\)
\(b := x \text{ //0}\)

if \(b = 0\) then
\(CS_2\)

not (\(CS_1\) and \(CS_2\))

assertion

extra behaviors

potentially bad behaviors

\(P_1\)
\(y = 0^o\) \(x = 1^s\)

\(x = 1\)

\(P_2\)
\(x = 0^o\) \(y = 1^s\)

\(x = 0\) \(s\)

\(y = 1\) \(s\)
Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

Dekker Protocol

x := 1
a := y // 0
if a = 0 then
  ▶ CS1

y := 1
b := x // 0
if b = 0 then
  ▶ CS2

not (CS1 and CS)

assertion

P1

y = 0^o  x = 1^s

P2

x = 0^o  y = 1^s

x = 1
y = 1

potentially bad behaviors

extra behaviors
**Dekker Protocol**

```
x := 1
a := y // 0
if a = 0 then
  → CS

y := 1
b := x // 0
if b = 0 then
  → CS
```

**Assertion**

```
not (CS1 and CS)
```

**Extra Behaviors**

- Potentially bad behaviors
- Extra behaviors
Shared Variable
Concurrency

- Sequential Consistency (SC)
- Total Store Ordering (TSO)
- Memory Fences
- Non-monotonicity
- Load-Buffer Semantics
Shared Variable
Concurrency

- Sequential Consistency (SC)
- Total Store Ordering (TSO)
- Memory Fences
- Non-monotonicity
- Load-Buffer Semantics
Shared Variable
Concurrency

- Sequential Consistency (SC)
- Total Store Ordering (TSO)
- Memory Fences
- Non-monotonicity
- Load-Buffer Semantics (Examples)
**Message Passing**

- \( x := 1 \)
- \( y := 1 \)
- \( a := y \)
- \( b := x \)
- \( a = 1 \) and \( b = 0 \)

**Shared Variable Concurrency**

**Total Store Order (TSO)**

**Load-Buffer Semantics**

- \( P_1 \)
- \( P_2 \)
- \( x = 0 \)
- \( y = 0 \)

**Extra behaviors**

**Potentially bad behaviors**
Message Passing

x := 1
y := 1

a := y
b := x

a = 1 and b = 0

P1
P2

x := 1
y := 0

Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

potentially bad behaviors
extra behaviors
Message Passing

Shared Variable Concurrency

Total Store Order (TSO)

Load-Buffer Semantics

\[ x := 1 \]
\[ y := 1 \]
\[ a := y \]
\[ b := x \]

\[ a = 1 \text{ and } b = 0 \]

\[ x = 1 \]
\[ y = 0 \]

Potentially bad behaviors

Extra behaviors
Message Passing

\[a := y\]
\[b := x\]

\[x := 1\]
\[y := 0\]

\[a = 1 \text{ and } b = 0\]
Message Passing

Shared Variable Concurrency

Total Store Order (TSO)

Load-Buffer Semantics

\[ x := 1 \]
\[ y := 1 \]
\[ a := y \]
\[ b := x \]

\[ a = 1 \] and \[ b = 0 \]

\[ P_1 \]
\[ P_2 \]

\[ x = 1 \] and \[ y = 0 \]

potentially bad behaviors

extra behaviors
Message Passing

\begin{align*}
x & := 1 \\
y & := 1 \\
a & := y \\
b & := x
\end{align*}

\begin{itemize}
  \item \( a = 1 \) and \( b = 0 \)
\end{itemize}

\textbf{Shared Variable Concurrency}

\textbf{Total Store Order (TSO)}

\textbf{Load-Buffer Semantics}

\textit{potentially bad behaviors}

\textit{extra behaviors}
Message Passing

Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

a := y
b := x

x := 1
y := 1

P₁
P₂

x = 1
y = 1

s

a = 1 and b = 0

potentially bad behaviors

extra behaviors

Message Passing

x := 1
y := 1

P₁
P₂

x = 1
y = 1

s

a = 1 and b = 0

potentially bad behaviors

extra behaviors

Message Passing

x := 1
y := 1

P₁
P₂

x = 1
y = 1

s

a = 1 and b = 0

potentially bad behaviors

extra behaviors
Shared Variable Concurrency
Total Store Order (TSO)
Load-Buffer Semantics

Message Passing

\[ x := 1 \]
\[ y := 1 \]
\[ a := y \]
\[ b := x \]

\( a = 1 \) and \( b = 0 \)

\( P_1 \):

\[ x = 1 \]
\[ y = 1 \]

\( P_2 \):

\[ a = y \]
\[ b = x \]

\( P_1 \) sends \( x = 1 \) and \( y = 1 \) to \( P_2 \).

\( P_2 \) receives \( x = 1 \) and \( y = 1 \) from \( P_1 \).

\( P_1 \) and \( P_2 \) synchronize.

Concurrent behaviors:
- \( a = 1 \) and \( b = 0 \)

Load-Buffer Semantics:
- Potentially bad behaviors
- Extra behaviors
**Shared Variable Concurrency**

**Total Store Order (TSO)**

**Load-Buffer Semantics**

---

**Message Passing**

- \(a := y\)
- \(b := x\)

\(a = 1\) and \(b = 0\)

**P1**

- \(x := 1\)
- \(y := 1\)

**P2**

- \(a := y\)
- \(b := x\)

- \(x = 1\)
- \(y = 1\)

---

- Potentially bad behaviors
- Extra behaviors
Message Passing

\[ x := 1 \]
\[ y := 1 \]
\[ a := y \]
\[ b := x \]

\[ x = 1 \]
\[ y = 1 \]
\[ y = 1^o \]

\[ x = 1^s \]
\[ y = 1^s \]

\[ a = 1 \] and \[ b = 0 \]
Message Passing

Shared Variable Concurrency

Total Store Order (TSO)

Load-Buffer Semantics

\[ x := 1 \]
\[ y := 1 \]
\[ a := y \]
\[ b := x \]

\[ a = 1 \text{ and } b = 0 \]

\[ x = 1 \]
\[ y = 1 \]

\[ x = 1 \]
\[ y = 1 \]

potentially bad behaviors

extra behaviors
Message Passing

\[ a = 1 \quad \text{and} \quad b = 0 \]

\[ x := 1 \]
\[ y := 1 \]
\[ a := y \]
\[ b := x \]

**Shared Variable Concurrency**

**Total Store Order (TSO)**

**Load-Buffer Semantics**

**extra behaviors**

**potentially bad behaviors**
Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

Message Passing

P1
\[ \begin{align*}
x &= 1 \\
y &= 1
\end{align*} \]
P2
\[ \begin{align*}
a &= y \quad \text{//1} \\
b &= x
\end{align*} \]
\[ \begin{align*}
x_1 &= 1 \\
y_1 &= 1
\end{align*} \]
\[ \begin{align*}
x_0 &= 1 \\
y_0 &= 0
\end{align*} \]

\[ a = 1 \quad \text{and} \quad b = 0 \]
Message Passing

```
\text{a:=1 and b:=0}
```

**Shared Variable Concurrency**

**Total Store Order (TSO)**

**Load-Buffer Semantics**
Message Passing

Shared Variable
Concurrency

Total Store
Order (TSO)

Load-Buffer
Semantics

\[ x := 1 \]
\[ y := 1 \]
\[ a := y \]
\[ b := x \]

\[ a = 1 \text{ and } b = 0 \]

Extra behaviors

Potentially bad behaviors
Message Passing

Shared Variable Concurrency
Total Store Order (TSO)
Load-Buffer Semantics

a:=y // 1
b:=x // 1

a=1 and b=0

P1

P2

x=1
y=1
x=1
y=1
x=1s
y=1s
x=1o

extra behaviors
potentially bad behaviors
Shared Variable Concurrency

Total Store Order (TSO)

Load-Buffer Semantics

Message Passing

\[ x := 1 \]
\[ y := 1 \]

\[ a := y \]
\[ b := x \]

\[ a = 1 \text{ and } b = 0 \]

\[ p_1 \]
\[ p_2 \]

\[ x = 1^s \]
\[ y = 1^s \]

\[ x = 1 \]
\[ y = 1 \]

potentially bad behaviors

extra behaviors
<table>
<thead>
<tr>
<th>Shared Variable</th>
<th>Concurrency</th>
<th>Load-Buffer Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Sequential Consistency (SC)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Total Store Ordering (TSO)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Memory Fences</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Non-monotonicity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Load-Buffer Semantics</td>
</tr>
</tbody>
</table>
Shared Variable
Concurrency

Load-Buffer Semantics

- Sequential Consistency (SC)
- Total Store Ordering (TSO)
- Memory Fences
- Non-monotonicity
- Load-Buffer Semantics
Shared Variable

Concurrency

Load-Buffer

Semantics

- Sequential Consistency (SC)
- Total Store Ordering (TSO)
- Memory Fences
- Non-monotonicity

- Load-Buffer Semantics (Verification)
Shared Variable
Concurrency

Load-Buffer
Semantics

- Sequential Consistency (SC)
- Total Store Ordering (TSO)
- Memory Fences
- Non-monotonicity

- Load-Buffer Semantics (Verification)
  - Equivalence with Store-Buffer Semantics
  - Monotonicity and WQO
• Sequential Consistency (SC)
• Total Store Ordering (TSO)
• Memory Fences
• Non-monotonicity

Load-Buffer Semantics (Verification)

• Equivalence with Store-Buffer Semantics
• Monotonicity and WQO
Load-Buffer Semantics

Store-Buffer Semantics
Store-Buffer Semantics

\[ P_1: w(x,2) \]

Load-Buffer Semantics

\[ x=0 \]
\[ y=0 \]
Store-Buffer Semantics

P1: w(x,2)

Load-Buffer Semantics
Store-Buffer Semantics

\[
\begin{align*}
P_1 & \quad x=2 \quad x=0 \\
P_2 & \quad y=0
\end{align*}
\]

Load-Buffer Semantics

\[
\begin{align*}
P_1 & \quad \text{w(x,2)} \\
P_1 & \quad \text{r(y,0)}
\end{align*}
\]
Store-Buffer Semantics

P₁ \text{ w}(x, 2) \quad P₁ \text{ r}(y, 0)

P₁ → x=2 → x=0
P₂ → y=0

Load-Buffer Semantics

P₁ \quad P₁ → x=0 \quad y=0
P₂
Store-Buffer Semantics

P₁: w(x,2)  P₂: w(y,1)

| P₁ | x=2 | x=0 |
| P₂ | y=1 | y=0 |

Load-Buffer Semantics

P₁: r(y,0)  P₂: w(y,1)

| P₁ | x=0 |
| P₂ | y=0 |
Store-Buffer Semantics

P₁ \( x=2 \) \( x=0 \)

P₂ \( y=1 \) \( y=0 \)

Load-Buffer Semantics

P₁ \( w(x,2) \) \( r(y,0) \) \( w(y,1) \) \( w(x,1) \)

P₁ \( x=0 \) \( x=0 \)

P₂ \( y=0 \) \( y=0 \)
Store-Buffer Semantics

P₁: w(x,2)  P₂: w(x,1)  P₁: r(y,0)  P₂: w(y,1)

Load-Buffer Semantics

P₁: w(x)  P₂: w(y)

x=0  y=0

x=2  y=1
Store-Buffer Semantics

Load-Buffer Semantics
Store-Buffer Semantics

P₁: \text{w}(x,2)\quad P₂: \text{w}(x,1)\quad P₁: \text{r}(y,0)\quad P₂: \text{w}(y,1)

Load-Buffer Semantics

P₁: w(x,2)\quad P₁: r(y,0)\quad P₂: w(y,0)
Load-Buffer Semantics

P1: w(x,2)  P2: w(y,1)  P2: w(x,1)

Store-Buffer Semantics

P1: r(y,0)  P2: w(y,0)  P2: w(x,1)
Load-Buffer Semantics

Store-Buffer Semantics

Load-Buffer Semantics

Store-Buffer Semantics
Store-Buffer Semantics

\[ \begin{align*}
P_1 & \quad x=2 \\
P_2 & \quad y=1
\end{align*} \]

Load-Buffer Semantics

\[ \begin{align*}
P_1 & \quad x=0 \\
P_2 & \quad y=0
\end{align*} \]
Store-Buffer Semantics

Load-Buffer Semantics

P₁: w(x, 2)
P₁: r(y, 0)

P₂: w(y, 1)
P₂: w(x, 1)
P₂: r(x, 2)
Store-Buffer Semantics

P₁ → x=2 → P₂
P₁ → y=0

P₂ → w(y,1) → P₁
P₂ → w(x,1) → P₁
P₂ → r(x,2) → P₁

Load-Buffer Semantics

P₁ ← x=0 → P₂
P₁ ← y=0

P₂ ← w(x,2) → P₁
P₂ ← r(y,0) → P₁
Store-Buffer Semantics

- P1: w(x, 2)
- P1: r(y, 0)
- P2: w(y, 1)
- P2: w(x, 1)
- P2: r(x, 2)

Load-Buffer Semantics

- P2: w(y, 1)
Store-Buffer Semantics

P₁: w(x,2)  P₁: r(y,0)  P₂: w(y,1)  P₂: w(x,1)  P₂: r(x,2)

Load-Buffer Semantics

P₁: y=0  P₁: x=0  P₂: y=1  P₂: x=2  P₂: y=1

P₂: w(y,1)
**Load-Buffer Semantics**

- P1 → y = 0° → x = 0
- P2 → y = 1° → y = 1

**Store-Buffer Semantics**

- P1 → x = 2
- P2 → y = 1

**Operations**

- P1: w(x, 2)
- P1: r(y, 0)
- P2: w(y, 1)
- P2: w(x, 1)
- P2: r(x, 2)
Store-Buffer Semantics

P₁ → x=2
P₂ → y=1

Load-Buffer Semantics

P₁ → y=0
P₂ → y=1

P₁: w(x,2)  P₁: r(y,0)  P₂: w(y,1)  P₂: w(x,1)  P₂: r(x,2)
**Store-Buffer Semantics**

- $P_1$: val($x$) = 2
- $P_2$: val($y$) = 1

**Load-Buffer Semantics**

- $P_1$: val($y$) = 0
- $P_2$: val($x$) = 1

Transactions:
- $P_1$: write($x$, 2) _red_
- $P_1$: read($y$, 0) _orange_
- $P_2$: write($y$, 1) _green_
- $P_2$: read($x$, 2) _blue_
- $P_2$: write($x$, 1) _green_
- $P_2$: write($y$, 1) _green_
Store-Buffer Semantics

P₁: w(x,2)  P₁: r(y,0)  P₂: w(y,1)  P₂: w(x,1)  P₂: r(x,2)

Load-Buffer Semantics

P₁: y=0  P₁: y=1  P₂: x=1  P₂: y=1
P₂: w(y,1)  P₂: w(x,1)

y=1  x=1
y=0  x=2
Store-Buffer Semantics

P₁: w(x,2)  P₁: r(y,0)

P₂: w(y,1)  P₂: w(x,1)  P₂: r(x,2)

Load-Buffer Semantics

P₁: w(x,2)  P₁: w(x,2)  P₂: w(y,1)

P₂: w(x,1)
Store-Buffer Semantics

Load-Buffer Semantics

P1: w(x, 2)  P1: r(y, 0)  P2: w(y, 1)  P2: w(x, 1)  P2: r(x, 2)

P1: w(x, 2)  P2: w(y, 1)  P2: w(x, 1)  P1: w(x, 2)

x = 2  y = 1

x = 2  y = 1

y = 0  x = 2

x = 2  y = 1

x = 2  y = 1
Store-Buffer Semantics

Load-Buffer Semantics
Store-Buffer Semantics

P1
P2

P1: w(x,2) P1: r(y,0) P2: w(y,1) P2: w(x,1) P2: r(x,2)

Load-Buffer Semantics

P1
P2

P2: w(y,1) P2: w(x,1) P1: w(x,2) P2: r(x,2) P1: r(y,0)
“processes perform identical transition sequences”
Store-Buffer Semantics

\[ P_1 \rightarrow \text{w}(x,2) \leftarrow P_2 \rightarrow \text{w}(x,1) \]

Load-Buffer Semantics

\[ P_1 \rightarrow \text{w}(x,2) \leftarrow P_2 \rightarrow \text{w}(x,1) \]

\[ P_1 \rightarrow \text{r}(y,0) \leftarrow P_2 \rightarrow \text{r}(y,0) \]

\[ P_1 \rightarrow \text{w}(y,1) \leftarrow P_2 \rightarrow \text{w}(y,1) \]

\[ P_1 \rightarrow \text{r}(x,2) \leftarrow P_2 \rightarrow \text{r}(x,2) \]

“processes perform identical transition sequences”
Store-Buffer Semantics

Load-Buffer Semantics

"processes perform identical transition sequences"
"processes perform identical transition sequences"
Store-Buffer Semantics

P₁: w(x,2) P₂: w(y,1) P₁: r(y,0) P₂: w(x,1) P₂: r(x,2)

Load-Buffer Semantics

P₁: w(y,1) P₂: w(x,1) P₁: w(x,2) P₂: r(x,2) P₁: r(y,0)

“processes perform identical transition sequences”
"processes perform identical transition sequences"
Shared Variable
Concurrency

Load-Buffer
Semantics

- Sequential Consistency (SC)
- Total Store Ordering (TSO)
- Memory Fences
- Non-monotonicity
- Load-Buffer Semantics
Shared Variable
Concurrency

Load-Buffer
Semantics

- Sequential Consistency (SC)
- Total Store Ordering (TSO)
- Memory Fences
- Non-monotonicity
- Load-Buffer Semantics
Shared Variable
Concurrency

Load-Buffer
Semantics

- Sequential Consistency (SC)
- Total Store Ordering (TSO)
- Memory Fences
- Non-monotonicity
- Load-Buffer Semantics (Verification)
Shared Variable Concurrency

Load-Buffer Semantics

- Sequential Consistency (SC)
- Total Store Ordering (TSO)
- Memory Fences
- Non-monotonicity
- Load-Buffer Semantics (Verification)
  - Equivalence with Store-Buffer Semantics
  - Monotonicity and WQO
Shared Variable

Concurrency

Load-Buffer Semantics

- Sequential Consistency (SC)
- Total Store Ordering (TSO)
- Memory Fences
- Non-monotonicity

- Load-Buffer Semantics (Verification)
  - Equivalence with Store-Buffer Semantics
  - Monotonicity and WQO
LB-Semantics

Model

Configurations

Ordering

Transitions

Monotonicity

Upward Closed Sets

Computing Predecessors

Backward Reachability
\( x := 1 \)

\( y := 1 \)

\( a := y \)
\( b := x \)

\( a = 1 \) and \( b = 0 \)

\( P_1 \)

\( P_2 \)

\( x = 1^s \)
\( y = 1^s \)

\( x = 1^o \)
\( y = 1 \)
LB-Semantics

Model

Ordering

Monotonicity

Backward Reachability

Upward Closed Sets

Transitions

Computing Predecessors

Configurations
LB-Semantics

Model

Ordering

Monotonicity

Upward Closed Sets

Backward Reachability

Transitions

Computing Predecessors

Configurations
configuration
configuration

\[
\begin{align*}
  x &= 1 \\
  y &= 1 \\
  a &= y \\
  b &= x \\
\end{align*}
\]

\[
\begin{align*}
  P_1 \quad &\quad x = 1^s, \quad y = 1^s \\
  P_2 \quad &\quad y = 1^o \\
  &\quad x = 1, \quad y = 1 \\
\end{align*}
\]
configuration

process state

P₁ → x=1₁s  y=1₁s → P₂

P₂ → y=1₀ → P₁

x:=1
y:=1
a:=y
b:=x
\[x := 1\]
\[y := 1\]
\[a := y\]
\[b := x\]

Process counter

Configuration

State configuration

Program counter

\[P_1\]

\[x := 1\]
\[y := 1\]

\[P_2\]

\[y := 1\]

\[x := 1\]
\[y := 1\]

\[x := 1\]
\[y := 1\]

\[y := 1\]
\[x := 1\]
configuration

program-counter

process state

process state

process state

x:=1
y:=1

x:=1
y:=1

a:=y
b:=x

P1

x=1
y=1

P2

y=1

x=1
y=1

program-counter

process state

configuration
configuration

$P_1 \xrightarrow{x=1, y=1} P_2 \xrightarrow{y=1} x=1, y=1$

process

state

program-counter

program-counter
configuration

program-counter
process state
x:=1
y:=1

process state
da:=y
b:=x

buffer content

memory state

buffer state

program-counter
process state

buffer content

buffer state

process state

program-counter

Transitions
Load-Buffer Semantics

- write-to-memory
- propagate-other-message
- read-own-write
- delete
- read-buffer-head
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - "sub-word in-between"
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - “sub-word in-between”
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - "sub-word in-between"

\[ y = 3^s \]
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - “sub-word in-between”

self-message on y

\( y=3^s \)
ordering on configurations = ordering on process states + ordering on buffers

• self-messages:
  • same order
  • same value
  • “sub-word in-between”

self-message on y

y=3s  x=1s
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - "sub-word in-between"

\[
\begin{align*}
\text{self-message on } y & \quad y = 3^s \\
\text{self-message on } x & \quad x = 1^s
\end{align*}
\]
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - “sub-word in-between”

self-message on y
self-message on x

\( y = 3^s \quad x = 1^s \)
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - “sub-word in-between”

self-message on y
self-message on x

y=3
x=1
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - "sub-word in-between"

self-message on y
self-message on x

y=3s  x=1s
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - "sub-word in-between"

self-message on \( y \)

self-message on \( x \)

\( y=3^s \)

\( x=1^s \)
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - "sub-word in-between"

self-message on y < self-message on x

y = 3
x = 1
ordering on configurations = ordering on process states + ordering on buffers

• self-messages:
  • same order
  • same value
  • “sub-word in-between”

self-message on y < self-message on x

y = 3
x = 1
ordering on configurations = ordering on process states + ordering on buffers

• self-messages:
  • same order
  • same value
  • “sub-word in-between”

self-message on y < self-message on x

\[ y = 3s \quad \text{and} \quad x = 1s \]
ordering on configurations

ordering on process states

ordering on buffers

- self-messages:
  - same order
  - same value
  - "sub-word in-between"

self-message on y

self-message on x

$y = 3^s$

$x = 1^s$
ordering on configurations = ordering on process states + ordering on buffers

• self-messages:
  • same order
  • same value
  • "sub-word in-between"

self-message on \( y \)
self-message on \( x \)
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - "sub-word in-between"

self-message on y < self-message on x

\[ y=3^s, \quad x=1^s \]
ordering on configurations = ordering on process states + ordering on buffers

• self-messages:
  • same order
  • same value
  • "sub-word in-between"

self-message on y
self-message on x

y = 3^s
x = 1^s
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - "sub-word in-between"

self-message on y < self-message on x

y = 3s  x = 1s
ordering on configurations = ordering on process states + ordering on buffers

• self-messages:
  • same order
  • same value
  • "sub-word in-between"

self-message on y < self-message on x

y = 3s  x = 1s
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - "sub-word in-between"

self-message on y
self-message on x

\[ y = 3^s \quad x = 1^s \]
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - "sub-word in-between"

\[
\begin{align*}
\text{x}=5 \quad \text{y}=3 \quad \text{y}=2 \quad \text{x}=4 \quad \text{x}=1 \quad \text{y}=2 \quad \text{x}=8 \\
\text{x}=5 \quad \text{y}=3 \quad \text{y}=2 \quad \text{x}=4 \quad \text{x}=1 \quad \text{y}=2 \quad \text{x}=8 \\
\text{y}=4 \quad \text{x}=5 \quad \text{x}=7 \quad \text{y}=3 \quad \text{y}=2 \quad \text{x}=9 \quad \text{x}=4 \quad \text{x}=1 \quad \text{x}=2 \quad \text{y}=2 \quad \text{x}=6 \quad \text{x}=8 \quad \text{y}=8
\end{align*}
\]
ordering on configurations = ordering on process states + ordering on buffers

• self-messages:
  • same order
  • same value
  • “sub-word in-between”

x = 5° y = 3° y = 2° x = 4° x = 1° y = 2° x = 8°

y = 4° x = 5° x = 7° y = 3° y = 2° x = 9° x = 4° x = 1° x = 2° y = 2° x = 6° x = 8° y = 8°
ordering on configurations = ordering on process states + ordering on buffers

• self-messages:
  • same order
  • same value
  • “sub-word in-between”

x = 5°  y = 3°  y = 2°  x = 4°  x = 1°  y = 2°  x = 8°

y = 4°  x = 5°  x = 7°  y = 3°  y = 2°  x = 9°  x = 4°  x = 1°  x = 2°  y = 2°  x = 6°  x = 8°  y = 8°
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - "sub-word in-between"

x=5° y=3° y=2° x=4° x=1° y=2° x=8°

y=4° x=5° x=7° y=3° y=2° x=9° x=4° x=1° x=2° y=2° x=6° x=8° y=8°
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - "sub-word in-between"

x=5° y=3° y=2° x=4° x=1° y=2° x=8°

y=4° x=5° x=7° y=3° y=2° x=9° x=4° x=1° x=2° y=2° x=6° x=8° y=8°
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - “sub-word in-between”

\[
\begin{align*}
x &= 5^o & y &= 3^s & y &= 2^o & x &= 4^o & x &= 1^s & y &= 2^o & x &= 8^o \\
y &= 4^o & x &= 5^o & x &= 7^o & y &= 3^s & y &= 2^o & x &= 9^o & x &= 4^o & x &= 1^s & x &= 2^o & y &= 2^o & x &= 6^o & x &= 8^o & y &= 8^o
\end{align*}
\]
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - “sub-word in-between”

\[
\begin{align*}
x &= 5^o & y &= 3^s & y &= 2^o & x &= 4^o & x &= 1^s & y &= 2^o & x &= 8^o \\
y &= 4^o & x &= 5^o & x &= 7^o & y &= 3^s & y &= 2^o & x &= 9^o & x &= 4^o & x &= 1^s & x &= 2^o & y &= 2^o & x &= 6^o & x &= 8^o & y &= 8^o
\end{align*}
\]
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - “sub-word in-between”

\[ x=5^0 \quad y=3^s \quad y=2^o \quad x=4^o \quad x=1^s \quad y=2^o \quad x=8^o \]

\[ y=4^o \quad x=5^o \quad x=7^o \quad y=3^s \quad y=2^o \quad x=9^o \quad x=4^o \quad x=1^s \quad x=2^o \quad y=2^o \quad x=6^o \quad x=8^o \quad y=8^o \]
ordering on configurations

ordering on process states

ordering on buffers

- self-messages:
- same order
- same value
- “sub-word in-between”

x=5° y=3° y=2° x=4° x=1° y=2° x=8°

y=4° x=5° x=7° y=3° y=2° x=9° x=4° x=1° x=2° y=2° x=6° x=8° y=8°
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - "sub-word in-between"

\[ y = 3 \quad s \quad x = 1 \quad s \quad y = 2 \quad s \quad x = 5 \quad s \quad y = 2 \quad s \quad x = 4 \quad s \quad y = 2 \quad s \quad x = 5 \quad s \quad y = 2 \quad s \quad x = 4 \quad s \quad y = 2 \quad s \quad x = 8 \quad s \quad y = 2 \quad s \quad x = 8 \quad s \quad y = 2 \quad s \quad x = 8 \quad s \quad y = 2 \quad s \quad x = 8 \quad s \quad y = 2 \quad s \quad x = 8 \quad s \quad y = 2 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \quad x = 8 \quad s \quad y = 8 \quad s \Rightarrow WALLPAPERSWIDE.COM
ordering on configurations = ordering on process states + ordering on buffers

• self-messages:
  • same order
  • same value
  • “sub-word in-between”

\[
\begin{align*}
  x &= 5^o, & y &= 3^o, & y &= 2^o, & x &= 4^0, & x &= 1^s, & y &= 2^o, & x &= 8^o, \\
  y &= 4^o, & x &= 5^o, & x &= 7^o, & y &= 3^s, & y &= 2^o, & x &= 9^o, & x &= 4^o, & x &= 1^s, & y &= 2^o, & x &= 8^o, & y &= 8^o
\end{align*}
\]
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - "sub-word in-between"

\[
\begin{align*}
  &x=5^o, y=3^s, y=2^o, x=4^o, x=1^s, y=2^o, x=8^o \\
  &y=4^o, x=5^o, x=7^o, y=3^s, y=2^o, x=9^o, x=4^o, x=1^s, x=2^o, y=2^o, x=6^o, x=8^o, y=8^o
\end{align*}
\]
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - "sub-word in-between"

\[
\begin{align*}
x &= 5^o & y &= 3^s & x &= 2^o & x &= 4^o & x &= 1^s & y &= 2^o & x &= 8^o \\
x &= 7^o & y &= 3^s & x &= 2^o & x &= 4^o & x &= 1^s & x &= 2^o & y &= 2^o & x &= 6^o & x &= 8^o & y &= 8^o
\end{align*}
\]
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - “sub-word in-between”

\[ x=5^{\circ} \quad y=3^{\circ} \quad y=2^{\circ} \quad x=4^{\circ} \quad x=1^{\circ} \quad y=2^{\circ} \quad x=8^{\circ} \]

\[ y=4^{\circ} \quad x=5^{\circ} \quad x=7^{\circ} \quad y=3^{\circ} \quad y=2^{\circ} \quad x=9^{\circ} \quad x=4^{\circ} \quad x=1^{\circ} \quad x=2^{\circ} \quad y=2^{\circ} \quad x=6^{\circ} \quad x=8^{\circ} \quad y=8^{\circ} \]
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - "sub-word in-between"

\[
\begin{align*}
&x=5, y=3 \\
&x=6, y=2, x=4 \quad x=1, y=2, x=8
\end{align*}
\]

\[
\begin{align*}
&y=4, x=5, x=7 \quad y=3 \quad x=2, y=9, x=4 \quad x=1, y=2, x=6, x=8, y=8
\end{align*}
\]
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - "sub-word in-between"

\[
x=5^o, y=3^o, y=2^o, x=4^o, x=1^s, y=2^o, x=8^o
\]

\[
y=4^o, x=5^o, x=7^o, y=3^s, y=2^o, x=9^o, x=4^o, x=1^s, x=2^o, y=2^o, x=6^o, x=8^o, y=8^o
\]
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - "sub-word in-between"

\[ \begin{array}{cccccccc}
  x &=& 5 & y &=& 3 & y &=& 2 & x &=& 4 \\
  x &=& 1 & y &=& 2 & x &=& 8 \end{array} \]
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - “sub-word in-between”

\[
\begin{array}{cccccccc}
  x=5 & y=3 & y=2 & x=4 & x=1 & y=2 & x=8 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
  y=4 & x=5 & x=7 & y=3 & y=2 & x=9 & x=4 & x=1 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
  x=2 & y=2 & x=6 & x=8 & y=8 \\
\end{array}
\]
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - “sub-word in-between”

equality on finite sets

\[
\begin{align*}
x &= 5^o & y &= 3^s & y &= 2^o & x &= 4^o & x &= 1^s & y &= 2^o & x &= 8^o \\
y &= 4^o & x &= 5^o & x &= 7^o & y &= 3^s & y &= 2^o & x &= 9^o & x &= 4^o & x &= 1^s & x &= 2^o & y &= 2^o & x &= 6^o & x &= 8^o & y &= 8^o
\end{align*}
\]
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - “sub-word in-between”

equality on finite sets

\[
\begin{array}{cccccc}
  x = 5^o & y = 3^s & y = 2^o & x = 4^o & x = 1^s & y = 2^o & x = 8^o \\
\end{array}
\]

\[
\begin{array}{cccccc}
  y = 4^o & x = 5^o & x = 7^o & y = 3^s & y = 2^o & x = 9^o & x = 4^o & x = 1^s & x = 2^o & y = 2^o & x = 6^o & x = 8^o & y = 8^o \\
\end{array}
\]
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - “sub-word in-between”

finite data domain

equality on finite sets

equality on finite sets

WQO

x=5° y=3° z=2° x=4° x=1° y=2° x=8°

y=4° x=5° x=7° y=3° y=2° x=9° x=4° x=1° x=2° y=2° x=6° x=8° y=8°
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - "sub-word in-between"

finite data domain

equality on finite sets

equality on finite sets

Higman's theorem

WQO

x = 5° y = 3° x = 2° x = 8° y = 2°

y = 4° x = 5° x = 7° y = 3° y = 2° x = 9° x = 4° x = 1°

y = 8° x = 2° x = 6° x = 8°
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - "sub-word in-between"

equality on finite sets

finite data domain

Higman's theorem

WQO

vectors of words
ordering on configurations = ordering on process states + ordering on buffers

• self-messages:
  • same order
  • same value
  • "sub-word in-between"

equality on finite sets

finite data domain

equality on finite sets

vectors of WQOs

WQO

Higman's theorem

vectors of words

\[ x=5^o \ y=3^s \ y=2^o \ x=4^o \ x=1^s \ y=2^o \ x=8^o \]

\[ y=4^o \ x=5^o \ x=7^o \ y=3^s \ y=2^o \ x=9^o \ x=4^o \ x=1^s \ x=2^o \ y=2^o \ x=6^o \ x=8^o \ y=8^o \]
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - "sub-word in-between"

finite data domain

equality on finite sets

WQO

Higman's theorem

vectors of WQOs

vectors of words

x=5° y=3° x=2° x=4° x=1° y=2° x=8°

y=4° x=5° x=7° y=3° y=2° x=9° x=4° x=1° x=2° y=2° x=6° x=8° y=8°
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory

\[ x := 1 \]
\[ y := 1 \]
\[ a := y \]
\[ b := x \]

\[ P_1 \]
\[ P_2 \]

\[ x = 1 \]
\[ y = 1 \]
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory

identical program counters

x := 1
y := 1
a := y
b := x

P1

P2

x = 1
y = 1
x = 1
y = 1
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory

identical program counters

P₁
x:=1
y:=1
P₂
a:=y
b:=x

x=1
y=1

x:=1
y:=1
a:=y
b:=x

x=1
y=1
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory

identical program counters

\[
\begin{align*}
x &:= 1 \\
y &:= 1 \\
a &:= y \\
b &:= x
\end{align*}
\]

\[
\begin{align*}
x &:= 1 \\
y &:= 1 \\
a &:= y \\
b &:= x
\end{align*}
\]
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory

identical program counters

\[
\begin{align*}
x &:= 1 \\
y &:= 1 \\
a &:= y \\
b &:= x
\end{align*}
\]

\[
\begin{align*}
x &:= 1 \\
y &:= 1 \\
a &:= y \\
b &:= x
\end{align*}
\]
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory

identical program counters

x := 1
y := 1
a := y
b := x

P₁

x := 1
y := 1
a := y
b := x

P₂

x := 1
y := 1
a := y
b := x

P₁

x := 1
y := 1
a := y
b := x

P₂
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory

identical program counters

\[ x := 1 \]
\[ y := 1 \]
\[ a := y \]
\[ b := x \]

\[ x := 1 \]
\[ y := 1 \]
\[ a := y \]
\[ b := x \]
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory

identical program counters

\[ \begin{array}{c}
x := 1 \\
y := 1 \\
a := y \\
b := x
\end{array} \]

\[ \begin{array}{c}
x := 1 \\
y := 1 \\
a := y \\
b := x
\end{array} \]
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory

identical program counters

\[
x := 1 \\
y := 1 \\
a := y \\
b := x
\]

\[
x := 1 \\
y := 1 \\
a := y \\
b := x
\]
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory

identical program counters

\[ x := 1 \]
\[ y := 1 \]
\[ a := y \]
\[ b := x \]

\[ x := 1 \]
\[ y := 1 \]
\[ a := y \]
\[ b := x \]
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory

identical program counters

\[x := 1\]
\[y := 1\]
\[a := y\]
\[b := x\]

\[x = 1\]
\[y = 1\]
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory

identical program counters

x:=1
y:=1
a:=y
b:=x

P1
P2

x=1
y=1
a:=y
b:=x

P1
P2

x=1
y=1
a:=y
b:=x

x=1
y=1

x=1
y=1
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory

identical program counters

$x := 1$
$y := 1$
$a := y$
$b := x$

$x := 1$
$y := 1$
$a := y$
$b := x$

$x = 1$
$y = 1$

$x = 1$
$y = 1$
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory

identical program counters

```
\begin{align*}
x &:= 1 \\
y &:= 1 \\
a &:= y \\
b &:= x
\end{align*}
```

```
\begin{align*}
x &:= 1 \\
y &:= 1 \\
a &:= y \\
b &:= x
\end{align*}
```
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory

identical program counters

identical memory states

\[
\begin{align*}
x &:= 1 \\
y &:= 1 \\
a &:= y \\
b &:= x
\end{align*}
\]

\[
\begin{align*}
\text{P}_1 & \quad \text{P}_2 \\
x &:= 1 \\
y &:= 1 \\
a &:= y \\
b &:= x
\end{align*}
\]
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory

identical program counters

identical memory states

\[
\begin{align*}
x &:= 1 \\
y &:= 1 \\
a &:= y \\
b &:= x
\end{align*}
\]

\[
\begin{align*}
x &:= 1 \\
y &:= 1 \\
a &:= y \\
b &:= x
\end{align*}
\]
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory

identical program counters

identical memory states

x:=1
y:=1
a:=y
b:=x
P_1 P_2 x=1 y=1

x:=1
y:=1
a:=y
b:=x
P_1 P_2 x=1 y=1

identical program counters

identical memory states

x:=1
y:=1
a:=y
b:=x
P_1 P_2 x=1 y=1

identical program counters

identical memory states

x:=1
y:=1
a:=y
b:=x
P_1 P_2 x=1 y=1

identical program counters

identical memory states

x:=1
y:=1
a:=y
b:=x
P_1 P_2 x=1 y=1

identical program counters

identical memory states

x:=1
y:=1
a:=y
b:=x
P_1 P_2 x=1 y=1
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory

 identical program counters

 identical memory states

 x := 1
 y := 1
 a := y
 b := x

 P1

 x := 1
 y := 1
 a := y
 b := x

 P2

 x := 1
 y := 1

 P1

 x := 1
 y := 1
 a := y
 b := x

 P2
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory

identical program counters

identical memory states

\[
x := 1 \\
y := 1 \\
a := y \\
b := x
\]

\[
x := 1 \\
y := 1 \\
a := y \\
b := x
\]

\[
x := 1 \\
y := 1 \\
a := y \\
b := x
\]
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory

identical program counters

identical memory states

wqo

\[
x := 1 \\
y := 1 \\
a := y \\
b := x
\]

P_1 \quad \quad \quad P_2

x = 1 \\
y = 1

x := 1 \\
y := 1 \\
a := y \\
b := x

P_1 \quad \quad \quad P_2

x = 1 \\
y = 1

x := 1 \\
y := 1 \\
a := y \\
b := x

P_1 \quad \quad \quad P_2

x = 1 \\
y = 1
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory

identical program counters

identical memory states

\[
x := 1 \\
y := 1 \\
a := y \\
b := x
\]

\[
x := 1 \\
y := 1 \\
a := y \\
b := x
\]

WQO
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory

identical program counters = WQO

identical memory states = WQO
equality on finite sets = WQO

x := 1
y := 1
a := y
b := x

P_1

P_2
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory

identical program counters = identical memory states

WQO ⊑ WQO

equality on finite sets

finite data domain
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory

identical program counters

equality on finite sets

identical memory states

finite data domain

equality on finite sets

x := 1
y := 1
a := y
b := x

P1

P2

WQO

x = 1
y = 1
ordering on configurations = ordering on process states + ordering on buffers + ordering on memory

finite-state processes → equality on finite sets → identical program counters

WQO ⊑ finite data domain

equality on finite sets
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - "sub-word in-between"

equality on finite sets

finite data domain

equality on finite sets

WQO

Higman's theorem

vectors of WQOs

vectors of words
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - “sub-word in-between”

equality on finite sets

finite data domain

equality on finite sets

vectors of WQOs

Higman's theorem

WQO

"extra messages don't harm"
ordering on configurations = ordering on process states + ordering on buffers

- self-messages:
  - same order
  - same value
  - "sub-word in-between"

equality on finite sets

finite data domain

vectors of WQOs

Higman's theorem

wqos

vectors of words

"process can throw them away"

"extra messages don't harm"
LB-Semantics

Model

Ordering

Monotonicity

Upward Closed Sets

Backward Reachability

Computing Predecessors

Transitions

Configurations
LB-Semantics

Model

Ordering

Monotonicity

Upward Closed Sets

Computing Predecessors

Backward Reachability
LB-Semantics

Model

Ordering

Monotonicity

Upward Closed Sets

Computing Predecessors

Backward Reachability

Configurations

Transitions

Safety properties decidable

Process-state reachability decidable
LB-Semantics

Model

Configurations

Ordering

Transitions

Monotonicity

Upward Closed Sets

Computing Predecessors

Backward Reachability

For the LB-semantics, safety properties are decidable.

Process-state reachability is decidable.
LB-Semantics

Model

Ordering

Configurations

Transitions

Monotonicity

Upward Closed Sets

Computing Predecessors

Backward Reachability

For TSO

Safety properties decidable

Process-state reachability decidable

For the LB-semantics
“Given a finite-state program running under the TSO semantics:
We can check whether the program violates a given safety property”
### Experimental Results

#### Dual-TSO vs Memorax

- **Running time**
- **Memory consumption**

<table>
<thead>
<tr>
<th>Program</th>
<th>#P</th>
<th>Dual-TSO</th>
<th>Memorax</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>#T</td>
<td>#C</td>
</tr>
<tr>
<td>SB</td>
<td>5</td>
<td>0.3</td>
<td>10641</td>
</tr>
<tr>
<td>LB</td>
<td>3</td>
<td>0.0</td>
<td>2048</td>
</tr>
<tr>
<td>WRC</td>
<td>4</td>
<td>0.0</td>
<td>1507</td>
</tr>
<tr>
<td>ISA2</td>
<td>3</td>
<td>0.0</td>
<td>509</td>
</tr>
<tr>
<td>RWC</td>
<td>5</td>
<td>0.1</td>
<td>4277</td>
</tr>
<tr>
<td>W+RWC</td>
<td>4</td>
<td>0.0</td>
<td>1713</td>
</tr>
<tr>
<td>IRIW</td>
<td>4</td>
<td>0.0</td>
<td>520</td>
</tr>
<tr>
<td>NbW_w_wr</td>
<td>2</td>
<td>0.0</td>
<td>222</td>
</tr>
<tr>
<td>Sense_rev_bar</td>
<td>2</td>
<td>0.1</td>
<td>1704</td>
</tr>
<tr>
<td>Dekker</td>
<td>2</td>
<td>0.1</td>
<td>5053</td>
</tr>
<tr>
<td>Dekker_simple</td>
<td>2</td>
<td>0.0</td>
<td>98</td>
</tr>
<tr>
<td>Peterson</td>
<td>2</td>
<td>0.1</td>
<td>5442</td>
</tr>
<tr>
<td>Peterson_loop</td>
<td>2</td>
<td>0.2</td>
<td>7632</td>
</tr>
<tr>
<td>Szymanski</td>
<td>2</td>
<td>0.6</td>
<td>29018</td>
</tr>
<tr>
<td>MP</td>
<td>4</td>
<td>0.0</td>
<td>883</td>
</tr>
<tr>
<td>Ticket_spin_lock</td>
<td>3</td>
<td>0.9</td>
<td>18963</td>
</tr>
<tr>
<td>Bakery</td>
<td>2</td>
<td>2.6</td>
<td>82050</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>2</td>
<td>0.2</td>
<td>8324</td>
</tr>
<tr>
<td>Lamport_fast</td>
<td>3</td>
<td>17.7</td>
<td>292543</td>
</tr>
<tr>
<td>Burns</td>
<td>4</td>
<td>124.3</td>
<td>2762578</td>
</tr>
</tbody>
</table>

https://www.it.uu.se/katalog/tuang296/dual-tso
Experimental Results

Dual-TSO vs Memorax

- Running time
- Memory consumption

<table>
<thead>
<tr>
<th>Program</th>
<th>#P</th>
<th>Dual-TSO</th>
<th>Memorax</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>#T</td>
<td>#C</td>
</tr>
<tr>
<td>SB</td>
<td>5</td>
<td>0.3</td>
<td>10641</td>
</tr>
<tr>
<td>LB</td>
<td>3</td>
<td>0.0</td>
<td>2048</td>
</tr>
<tr>
<td>WRC</td>
<td>4</td>
<td>0.0</td>
<td>1507</td>
</tr>
<tr>
<td>ISA2</td>
<td>3</td>
<td>0.0</td>
<td>509</td>
</tr>
<tr>
<td>RWC</td>
<td>5</td>
<td>0.1</td>
<td>4277</td>
</tr>
<tr>
<td>W+RWC</td>
<td>4</td>
<td>0.0</td>
<td>1713</td>
</tr>
<tr>
<td>IRW</td>
<td>4</td>
<td>0.0</td>
<td>520</td>
</tr>
<tr>
<td>Nbw_w_wr</td>
<td>2</td>
<td>0.0</td>
<td>222</td>
</tr>
<tr>
<td>Sense_rev_bar</td>
<td>2</td>
<td>0.1</td>
<td>1704</td>
</tr>
<tr>
<td>Dekker</td>
<td>2</td>
<td>0.1</td>
<td>5053</td>
</tr>
<tr>
<td>Dekker_simple</td>
<td>2</td>
<td>0.0</td>
<td>98</td>
</tr>
<tr>
<td>Peterson</td>
<td>2</td>
<td>0.1</td>
<td>5442</td>
</tr>
<tr>
<td>Peterson_loop</td>
<td>2</td>
<td>0.2</td>
<td>7632</td>
</tr>
<tr>
<td>Szymanski</td>
<td>2</td>
<td>0.6</td>
<td>29018</td>
</tr>
<tr>
<td>MP</td>
<td>4</td>
<td>0.0</td>
<td>883</td>
</tr>
<tr>
<td>Ticket_spin_lock</td>
<td>3</td>
<td>0.9</td>
<td>18963</td>
</tr>
<tr>
<td>Bakery</td>
<td>2</td>
<td>2.6</td>
<td>82050</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>2</td>
<td>0.2</td>
<td>8324</td>
</tr>
<tr>
<td>Lamport_fast</td>
<td>3</td>
<td>17.7</td>
<td>292543</td>
</tr>
<tr>
<td>Burns</td>
<td>4</td>
<td>124.3</td>
<td>2762578</td>
</tr>
</tbody>
</table>

https://www.it.uu.se/katalog/tuang296/dual-tso
## Experimental Results

**Dual-TSO vs Memorax**

- **Running time**
- **Memory consumption**

### Standard Benchmarks: Litmus Tests and Mutual Exclusion Algorithms

<table>
<thead>
<tr>
<th>Program</th>
<th>#P</th>
<th>Dual-TSO</th>
<th>Memorax</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>#T</td>
<td>#C</td>
</tr>
<tr>
<td>SB</td>
<td>5</td>
<td>0.3</td>
<td>10641</td>
</tr>
<tr>
<td>LB</td>
<td>3</td>
<td>0.0</td>
<td>2048</td>
</tr>
<tr>
<td>WRC</td>
<td>4</td>
<td>0.0</td>
<td>1507</td>
</tr>
<tr>
<td>ISA2</td>
<td>3</td>
<td>0.0</td>
<td>509</td>
</tr>
<tr>
<td>RWC</td>
<td>5</td>
<td>0.1</td>
<td>4277</td>
</tr>
<tr>
<td>W+RWC</td>
<td>4</td>
<td>0.0</td>
<td>1713</td>
</tr>
<tr>
<td>IRIW</td>
<td>4</td>
<td>0.0</td>
<td>520</td>
</tr>
<tr>
<td>Nbw_w_wr</td>
<td>2</td>
<td>0.0</td>
<td>222</td>
</tr>
<tr>
<td>Sense_rev_bar</td>
<td>2</td>
<td>0.1</td>
<td>1704</td>
</tr>
<tr>
<td>Dekker</td>
<td>2</td>
<td>0.1</td>
<td>5053</td>
</tr>
<tr>
<td>Dekker_simple</td>
<td>2</td>
<td>0.0</td>
<td>98</td>
</tr>
<tr>
<td>Peterson</td>
<td>2</td>
<td>0.1</td>
<td>5442</td>
</tr>
<tr>
<td>Peterson_loop</td>
<td>2</td>
<td>0.2</td>
<td>7632</td>
</tr>
<tr>
<td>Szymanski</td>
<td>2</td>
<td>0.6</td>
<td>29018</td>
</tr>
<tr>
<td>MP</td>
<td>4</td>
<td>0.0</td>
<td>883</td>
</tr>
<tr>
<td>Ticket_spin_lock</td>
<td>3</td>
<td>0.9</td>
<td>18963</td>
</tr>
<tr>
<td>Bakery</td>
<td>2</td>
<td>2.6</td>
<td>82050</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>2</td>
<td>0.2</td>
<td>8324</td>
</tr>
<tr>
<td>Lamport_fast</td>
<td>3</td>
<td>17.7</td>
<td>292543</td>
</tr>
<tr>
<td>Burns</td>
<td>4</td>
<td>124.3</td>
<td>2762578</td>
</tr>
</tbody>
</table>
### Experimental Results

#### Running time in seconds

<table>
<thead>
<tr>
<th>Program</th>
<th>#P</th>
<th>Dual-TSO</th>
<th>Memorax</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>#T</td>
<td>#C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#T</td>
<td>#C</td>
</tr>
</tbody>
</table>

- **SB**: 5, #T: 0.3, #C: 10641, #T: 559.7, #C: 10515914
- **LB**: 3, #T: 0.0, #C: 2048, #T: 71.4, #C: 1499475
- **WRC**: 4, #T: 0.0, #C: 1507, #T: 63.3, #C: 1398393
- **ISA2**: 3, #T: 0.0, #C: 509, #T: 21.1, #C: 226519
- **RWC**: 5, #T: 0.1, #C: 4277, #T: 61.5, #C: 1196988
- **W+RWC**: 4, #T: 0.0, #C: 1713, #T: 83.6, #C: 1389009
- **IRIW**: 4, #T: 0.0, #C: 520, #T: 34.4, #C: 358057
- **Nbw_w_wr**: 2, #T: 0.0, #C: 222, #T: 10.7, #C: 200844
- **Sense_rev_bar**: 2, #T: 0.1, #C: 1704, #T: 0.8, #C: 20577
- **Dekker**: 2, #T: 0.1, #C: 5053, #T: 1.1, #C: 19788
- **Dekker_simple**: 2, #T: 0.0, #C: 98, #T: 0.0, #C: 595
- **Peterson**: 2, #T: 0.1, #C: 5442, #T: 5.2, #C: 90301
- **Peterson_loop**: 2, #T: 0.2, #C: 7632, #T: 5.6, #C: 100082
- **Szymanski**: 2, #T: 0.6, #C: 29018, #T: 1.0, #C: 26003
- **MP**: 4, #T: 0.0, #C: 883, #T: TO, #C: •
- **Ticket_spin_lock**: 3, #T: 0.9, #C: 18963, #T: TO, #C: •
- **Bakery**: 2, #T: 2.6, #C: 82050, #T: TO, #C: •
- **Dijkstra**: 2, #T: 0.2, #C: 8324, #T: TO, #C: •
- **Lamport_fast**: 3, #T: 17.7, #C: 292543, #T: TO, #C: •
- **Burns**: 4, #T: 124.3, #C: 2762578, #T: TO, #C: •
### Experimental Results

#### Dual-TSO vs Memorax

- **Running time**
- **Memory consumption**

<table>
<thead>
<tr>
<th>Program</th>
<th>#P</th>
<th>Dual-TSO</th>
<th>Memorax</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>#T</td>
<td>#C</td>
</tr>
<tr>
<td>SB</td>
<td>5</td>
<td>0.3</td>
<td>10641</td>
</tr>
<tr>
<td>LB</td>
<td>3</td>
<td>0.0</td>
<td>2048</td>
</tr>
<tr>
<td>WRC</td>
<td>4</td>
<td>0.0</td>
<td>1507</td>
</tr>
<tr>
<td>ISA2</td>
<td>3</td>
<td>0.0</td>
<td>509</td>
</tr>
<tr>
<td>RWC</td>
<td>5</td>
<td>0.1</td>
<td>4277</td>
</tr>
<tr>
<td>W+RWC</td>
<td>4</td>
<td>0.0</td>
<td>1713</td>
</tr>
<tr>
<td>IRIW</td>
<td>4</td>
<td>0.0</td>
<td>520</td>
</tr>
<tr>
<td>Nbw_w__wr</td>
<td>2</td>
<td>0.0</td>
<td>222</td>
</tr>
<tr>
<td>Sense_rev__bar</td>
<td>2</td>
<td>0.1</td>
<td>1704</td>
</tr>
<tr>
<td>Dekker</td>
<td>2</td>
<td>0.1</td>
<td>5053</td>
</tr>
<tr>
<td>Dekker__simple</td>
<td>2</td>
<td>0.0</td>
<td>98</td>
</tr>
<tr>
<td>Peterson</td>
<td>2</td>
<td>0.1</td>
<td>5442</td>
</tr>
<tr>
<td>Peterson__loop</td>
<td>2</td>
<td>0.2</td>
<td>7632</td>
</tr>
<tr>
<td>Szymanski</td>
<td>2</td>
<td>0.6</td>
<td>29018</td>
</tr>
<tr>
<td>MP</td>
<td>4</td>
<td>0.0</td>
<td>883</td>
</tr>
<tr>
<td>Ticket__spin__lock</td>
<td>3</td>
<td>0.9</td>
<td>18963</td>
</tr>
<tr>
<td>Bakery</td>
<td>2</td>
<td>2.6</td>
<td>82050</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>2</td>
<td>0.2</td>
<td>8324</td>
</tr>
<tr>
<td>Lamport__fast</td>
<td>3</td>
<td>17.7</td>
<td>292543</td>
</tr>
<tr>
<td>Burns</td>
<td>4</td>
<td>124.3</td>
<td>2762578</td>
</tr>
</tbody>
</table>
### Experimental Results

**Dual-TSO vs Memorax**

- **Running time**
- **Memory consumption**

**Dual-TSO is faster and uses less memory in most of examples**

<table>
<thead>
<tr>
<th>Program</th>
<th>#P</th>
<th>Dual-TSO #T</th>
<th>Dual-TSO #C</th>
<th>Memorax #T</th>
<th>Memorax #C</th>
</tr>
</thead>
<tbody>
<tr>
<td>SB</td>
<td>5</td>
<td>0.3</td>
<td>10641</td>
<td>559.7</td>
<td>10515914</td>
</tr>
<tr>
<td>LB</td>
<td>3</td>
<td>0.0</td>
<td>2048</td>
<td>71.4</td>
<td>1499475</td>
</tr>
<tr>
<td>WRC</td>
<td>4</td>
<td>0.0</td>
<td>1507</td>
<td>63.3</td>
<td>1398393</td>
</tr>
<tr>
<td>ISA2</td>
<td>3</td>
<td>0.0</td>
<td>509</td>
<td>21.1</td>
<td>226519</td>
</tr>
<tr>
<td>RWC</td>
<td>5</td>
<td>0.1</td>
<td>4277</td>
<td>61.5</td>
<td>1196988</td>
</tr>
<tr>
<td>W+RWC</td>
<td>4</td>
<td>0.0</td>
<td>1713</td>
<td>83.6</td>
<td>1389009</td>
</tr>
<tr>
<td>IRIW</td>
<td>4</td>
<td>0.0</td>
<td>520</td>
<td>34.4</td>
<td>358057</td>
</tr>
<tr>
<td>Nbw_w_wr</td>
<td>2</td>
<td>0.0</td>
<td>222</td>
<td>10.7</td>
<td>200844</td>
</tr>
<tr>
<td>Sense_rev_bar</td>
<td>2</td>
<td>0.1</td>
<td>1704</td>
<td>0.8</td>
<td>20577</td>
</tr>
<tr>
<td>Dekker</td>
<td>2</td>
<td>0.1</td>
<td>5053</td>
<td>1.1</td>
<td>19788</td>
</tr>
<tr>
<td>Dekker_simple</td>
<td>2</td>
<td>0.0</td>
<td>98</td>
<td>0.0</td>
<td>595</td>
</tr>
<tr>
<td>Peterson</td>
<td>2</td>
<td>0.1</td>
<td>5442</td>
<td>5.2</td>
<td>90301</td>
</tr>
<tr>
<td>Peterson_loop</td>
<td>2</td>
<td>0.2</td>
<td>7632</td>
<td>5.6</td>
<td>100082</td>
</tr>
<tr>
<td>Szymanski</td>
<td>2</td>
<td>0.6</td>
<td>29018</td>
<td>1.0</td>
<td>26003</td>
</tr>
<tr>
<td>MP</td>
<td>4</td>
<td>0.0</td>
<td>883</td>
<td>TO</td>
<td></td>
</tr>
<tr>
<td>Ticket_spin_lock</td>
<td>3</td>
<td>0.9</td>
<td>18963</td>
<td>TO</td>
<td></td>
</tr>
<tr>
<td>Bakery</td>
<td>2</td>
<td>2.6</td>
<td>82050</td>
<td>TO</td>
<td></td>
</tr>
<tr>
<td>Dijkstra</td>
<td>2</td>
<td>0.2</td>
<td>8324</td>
<td>TO</td>
<td></td>
</tr>
<tr>
<td>Lamport_fast</td>
<td>3</td>
<td>17.7</td>
<td>292543</td>
<td>TO</td>
<td></td>
</tr>
<tr>
<td>Burns</td>
<td>4</td>
<td>124.3</td>
<td>2762578</td>
<td>TO</td>
<td></td>
</tr>
</tbody>
</table>
## Experimental Results

### Parameterised Cases

<table>
<thead>
<tr>
<th>Program</th>
<th>Dual-TSO</th>
<th>#T</th>
<th>#C</th>
</tr>
</thead>
<tbody>
<tr>
<td>SB</td>
<td>0.0</td>
<td>147</td>
<td></td>
</tr>
<tr>
<td>LB</td>
<td>0.6</td>
<td>1028</td>
<td></td>
</tr>
<tr>
<td>MP</td>
<td>0.0</td>
<td>149</td>
<td></td>
</tr>
<tr>
<td>WRC</td>
<td>0.8</td>
<td>618</td>
<td></td>
</tr>
<tr>
<td>ISA2</td>
<td>4.3</td>
<td>1539</td>
<td></td>
</tr>
<tr>
<td>RWC</td>
<td>0.2</td>
<td>293</td>
<td></td>
</tr>
<tr>
<td>W+RWC</td>
<td>1.5</td>
<td>828</td>
<td></td>
</tr>
<tr>
<td>IRIW</td>
<td>4.6</td>
<td>648</td>
<td></td>
</tr>
</tbody>
</table>

![Graphs showing experimental results for various programs](chart.png)
Experimental Results
Parameterised Cases

<table>
<thead>
<tr>
<th>Program</th>
<th>Dual-TCO</th>
</tr>
</thead>
<tbody>
<tr>
<td>#T</td>
<td>#C</td>
</tr>
<tr>
<td>SB</td>
<td>0.0</td>
</tr>
<tr>
<td>LB</td>
<td>0.6</td>
</tr>
<tr>
<td>MP</td>
<td>0.0</td>
</tr>
<tr>
<td>WRC</td>
<td>0.8</td>
</tr>
<tr>
<td>ISA2</td>
<td>4.3</td>
</tr>
<tr>
<td>RWC</td>
<td>0.2</td>
</tr>
<tr>
<td>W+RWC</td>
<td>1.5</td>
</tr>
<tr>
<td>IRIW</td>
<td>4.6</td>
</tr>
</tbody>
</table>

unbounded number of processes
Increasing the number of processes

Experimental Results
Parameterised Cases

<table>
<thead>
<tr>
<th>Program</th>
<th>Dual-TSO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#T</td>
</tr>
<tr>
<td>SB</td>
<td>0.0</td>
</tr>
<tr>
<td>LB</td>
<td>0.6</td>
</tr>
<tr>
<td>MP</td>
<td>0.0</td>
</tr>
<tr>
<td>WRC</td>
<td>0.8</td>
</tr>
<tr>
<td>ISA2</td>
<td>4.3</td>
</tr>
<tr>
<td>RWC</td>
<td>0.2</td>
</tr>
<tr>
<td>W+RWC</td>
<td>1.5</td>
</tr>
<tr>
<td>IRIW</td>
<td>4.6</td>
</tr>
</tbody>
</table>
Experimental Results
Parameterised Cases

Dual-TSO is more scalable
Experimental Results
Parameterised Cases

**Dual-TSO is more efficient and scalable**

<table>
<thead>
<tr>
<th>Program</th>
<th>Dual-TSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>SB</td>
<td>0.0</td>
</tr>
<tr>
<td>LB</td>
<td>0.6</td>
</tr>
<tr>
<td>MP</td>
<td>0.0</td>
</tr>
<tr>
<td>WRC</td>
<td>0.8</td>
</tr>
<tr>
<td>ISA2</td>
<td>4.3</td>
</tr>
<tr>
<td>RWC</td>
<td>0.2</td>
</tr>
<tr>
<td>W+RWC</td>
<td>1.5</td>
</tr>
<tr>
<td>IRIW</td>
<td>4.6</td>
</tr>
</tbody>
</table>
Shared Variable
Concurrency

- Sequential Consistency (SC)
- Total Store Ordering (TSO)
- Memory Fences
- Non-monotonicity
- Load-Buffer Semantics