Robustness of Concurrent Programs on Weak Memory Models

Parosh Aziz Abdulla  Phong Ngo  Mohamed Faouzi Atig

Uppsala University  Sweden
Outline

- Concurrent Programs
- Total Store Order (TSO)
- Correctness - Robustness
- Program Repair
- Perspectives
- Conclusions
Based on:

- Parosh Aziz Abdulla, Mohamed Faouzi Atig, Rojin Rezvan. Parameterized verification under TSO is PSPACE-complete. POPL 2020.
- Parosh Aziz Abdulla, Jatin Arora, Mohamed Faouzi Atig, Shankara Narayanan Krishna. Verification of programs under the release-acquire semantics. PLDI 2019.
Outline

• Concurrent Programs
  • Total Store Order (TSO)
  • Correctness - Robustness
  • Program Repair
  • Perspectives
  • Conclusions
Concurrent Programs

1. \( y = 1; \)
2. \( b = x; \)
3. \( \text{if} \ (b == 0) \{
   \quad \text{//cs2;}
\}
4. \( x = 1; \)
5. \( a = y; \)
6. \( \text{if} \ (a == 0) \{
   \quad \text{//cs1;}
\}

Process 1

Process 2

x=0  y=0

Shared memory
Concurrent Programs

Process 1
1. x=1;
2. a=y;
3. if (a==0) {
4. //cs1;
5. }

Process 2

Shared memory

x=0  y=0

shared variables
Concurrent Programs

Process 1
1. x=1;
2. a=y;
3. if (a==0) {
   4. //cs1;
   5. }

Process 2
1. y=1;
2. b=x;
3. if (b==0) {
   4. //cs2;
   5. }

x=0  y=0

Shared memory

1. y=1;
2. b=x;
3. if (b==0) {
   4. //cs2;
   5. }

Process 1
1. x=1;
2. a=y;
3. if (a==0) {
   4. //cs1;
   5. }

Concurrent Programs

Processes

Shared variables

x=0  y=0

Shared memory

1. x=1;
2. a=y;
3. if (a==0) {
   4. //cs1;
   5. }

1. y=1;
2. b=x;
3. if (b==0) {
   4. //cs2;
   5. }

x=0  y=0

Shared memory
Concurrent Programs

Init: \( x=y=0 \)

**Process 1**
1. \( x=1; \)
2. \( a=y; \)
3. if \( (a==0) \) {
4.  \( //cs1; \)
5. }

**Process 2**
1. \( y=1; \)
2. \( b=x; \)
3. if \( (b==0) \) {
4.  \( //cs2; \)
5. }

Shared variables

**Shared memory**

\( x=0 \quad y=0 \)
Concurrent Programs

Init: $x = y = 0$

Process 1
1. $x = 1$
2. $a = y$
3. if ($a == 0$) {
4. //cs1;
5. }

Process 2
1. $y = 1$
2. $b = x$
3. if ($b == 0$) {
4. //cs2;
5. }

Shared variables

$x = 0 \quad y = 0$

Shared memory
Concurrent Programs

Init: x=y=0

Process 1
1. x=1;
2. a=y;
3. if (a==0) {
   4. //cs1;
   5. }

Process 2
1. y=1;
2. b=x;
3. if (b==0) {
   4. //cs2;
   5. }

Processes

Initial values

Local variables

Shared variables

Shared memory

x=0  y=0
Concurrent Programs

**Init**: \(x = y = 0\)

<table>
<thead>
<tr>
<th>Process 1</th>
<th>Process 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. (x = 1);</td>
<td>1. (y = 1);</td>
</tr>
<tr>
<td>2. (a = y);</td>
<td>2. (b = x);</td>
</tr>
<tr>
<td>3. if ((a == 0)) {</td>
<td>3. if ((b == 0)) {</td>
</tr>
<tr>
<td>4. //cs1;</td>
<td>4. //cs2;</td>
</tr>
<tr>
<td>5. }</td>
<td>5. }</td>
</tr>
</tbody>
</table>

**Program counters**: Initial values

**Processes**: Local variables

**Shared variables**: Shared memory

\(x = 0 \quad y = 0\)
Concurrent Programs

Init: $x = y = 0$

Process 1
1. $x = 1$
2. $a = y$
3. if ($a == 0$) {
   4. // cs1;
   5. }

Process 2
1. $y = 1$
2. $b = x$
3. if ($b == 0$) {
   4. // cs2;
   5. }

Shared memory
$x = 0 \quad y = 0$
Dekker Mutual Exclusion Protocol

Init: \( x=y=0 \)

**Process 1**
1. \( x=1; \)
2. \( a=y; \)
3. if \( (a==0) \) {
   4. //cs1;
   5. }

**Process 2**
1. \( y=1; \)
2. \( b=x; \)
3. if \( (b==0) \) {
   4. //cs2;
   5. }

**Shared memory**
\( x=0 \quad y=0 \)
Dekker Mutual Exclusion Protocol

Init: $x = y = 0$

Process 1
1. $x = 1$;
2. $a = y$;
3. if ($a == 0$) {
   4. //cs1;
   5. }

Process 2
1. $y = 1$;
2. $b = x$;
3. if ($b == 0$) {
   4. //cs2;
   5. }

Assertion: not (cs1 && cs2)

$x = 0$  $y = 0$

Shared memory
Dekker Mutual Exclusion Protocol

Init: \( x = y = 0 \)

1. \( x = 1; \)
2. \( a = y; \)
3. if \( (a == 0) \) {
4.   \(/ / cs1; \)
5. }

1. \( y = 1; \)
2. \( b = x; \)
3. if \( (b == 0) \) {
4.   \(/ / cs2; \)
5. }

Sequential Consistency (SC)

- Shared Memory
- Processes perform read/write **atomically**
- execute instructions of one process in **program-order**
- **interleave** instructions of different processes
- read-from the latest write

\( x = 0 \quad y = 0 \)

Shared memory
Sequential Consistency (SC)
- Shared Memory
- Processes perform read/write atomically
- execute instructions of one process in program-order
- interleave instructions of different processes
- read-from the latest write

Dekker Mutual Exclusion Protocol

Init: x=y=0

Process 1
1. x=1;
2. a=y;
3. if (a==0) {
4. //cs1;
5. }

Process 2
1. y=1;
2. b=x;
3. if (b==0) {
4. //cs2;
5. }

An SC execution
x=0 y=0

Shared memory
Dekker Mutual Exclusion Protocol

Initial state: $x=y=0$

Process 1:
1. $x=1$;
2. $a=y$;
3. if ($a==0$) {
4. //cs1;
5. }

Process 2:
1. $y=1$;
2. $b=x$;
3. if ($b==0$) {
4. //cs2;
5. }

Sequential Consistency (SC)
- Shared Memory
- Processes perform read/write atomically
- Execute instructions of one process in program-order
- Interleave instructions of different processes
- Read-from the latest write
Dekker Mutual Exclusion Protocol

**Sequential Consistency (SC)**
- Shared Memory
- Processes perform read/write atomically
- execute instructions of one process in program-order
- interleave instructions of different processes
- read-from the latest write

**Init:** $x = y = 0$

**Process 1**
1. $x = 1$
2. $a = y$
3. if ($a == 0$) {
4. //cs1;
5. }

**Process 2**
1. $y = 1$
2. $b = x$
3. if ($b == 0$) {
4. //cs2;
5. }

An SC execution

**Init:** $x = y = 0$

**Process 1**
1. $x = 1$
2. $a = y$
3. if ($a == 0$) {
4. //cs1;
5. }

**Process 2**
1. $y = 1$
2. $b = x$
3. if ($b == 0$) {
4. //cs2;
5. }
Dekker Mutual Exclusion Protocol

Sequential Consistency (SC)

- Shared Memory
- Processes perform read/write atomically
- execute instructions of one process in program-order
- interleave instructions of different processes
- read-from the latest write

An SC execution

Init: $x=y=0$

- Process 1:
  1. $x=1$
  2. $a=y$
  3. if ($a==0$) {
     4. //cs1;
     5. }

- Process 2:
  1. $y=1$
  2. $b=x$
  3. if ($b==0$) {
     4. //cs2;
     5. }

$w(x,1)$

$x=1$  $y=0$
Dekker Mutual Exclusion Protocol

**Sequential Consistency (SC)**
- Shared Memory
- Processes perform read/write atomically
- execute instructions of one process in program-order
- interleave instructions of different processes
- read-from the latest write

**Init:** $x = y = 0$

**Process 1**
1. $x = 1$
2. $a = y$
3. if ($a == 0$) {
   4. //cs1;
   5. }

**Process 2**
1. $y = 1$
2. $b = x$
3. if ($b == 0$) {
   4. //cs2;
   5. }

**An SC execution**
- Memory event: $w(x,1)$

**Shared memory**
- $x = 1$
- $y = 0$
Sequential Consistency (SC)
- Shared Memory
- Processes perform read/write atomically
- execute instructions of one process in program-order
- interleave instructions of different processes
- read-from the latest write

Dekker Mutual Exclusion Protocol

Process 1
1. x=1;
2. a=y; //o
3. if (a==0) {
   4. //cs1;
   5. }

Process 2
1. y=1;
2. b=x;
3. if (b==0) {
   4. //cs2;
   5. }

An SC execution

Init: x=y=0

memory event
w(x,1)
memory event
r(y,0)

x=1
y=0

Shared memory
Sequential Consistency (SC)

- Shared Memory
- Processes perform read/write atomically
- execute instructions of one process in program-order
- interleave instructions of different processes
- read-from the latest write

Dekker Mutual Exclusion Protocol

Init: $x = y = 0$

Process 1
1. $x = 1$
2. $a = y$ // 0
3. if ($a = 0$) {
   4. // cs1;
   5. }

Process 2
1. $y = 1$
2. $b = x$
3. if ($b = 0$) {
   4. // cs2;
   5. }

An SC execution

$w(x, 1)$

$r(y, 0)$
Sequential Consistency (SC)

- **Shared Memory**
- Processes perform read/write **atomically**
- execute instructions of one process in **program-order**
- **interleave** instructions of different processes
- **read-from** the latest write
Dekker Mutual Exclusion Protocol

Init: \( x=y=0 \)

Process 1
1. \( x=1; \)
2. \( a=y; \) //0
3. if (a==0) {
   4. //cs1;
   5. }

Process 2
1. \( y=1; \)
2. \( b=x; \)
3. if (b==0) {
   4. //cs2;
   5. }

Sequential Consistency (SC)
- Shared Memory
- Processes perform read/write atomically
- execute instructions of one process in program-order
- interleave instructions of different processes
- read-from the latest write

An SC execution
- \( w(x,1) \)
- \( r(y,0) \)
**Dekker Mutual Exclusion Protocol**

**Sequential Consistency (SC)**
- Shared Memory
- Processes perform read/write **atomically**
- execute instructions of one process in **program-order**
- **interleave** instructions of different processes
- **read-from** the latest write

**Init: x=y=0**

**Process 1**
1. x=1;
2. a=y; //0
3. if (a==0) {
4. //cs1;
5. }

**Process 2**
1. y=1;
2. b=x;
3. if (b==0) {
4. //cs2;
5. }

**An SC execution**
- w(x,1)
- r(y,0)

**Shared memory**
- x=1 y=0
Sequential Consistency (SC)

- Shared Memory
- Processes perform read/write atomically
- execute instructions of one process in program-order
- interleave instructions of different processes
- read-from the latest write

Dekker Mutual Exclusion Protocol

### Init: x=y=0

<table>
<thead>
<tr>
<th>Process 1</th>
<th>Process 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. x=1;</td>
<td>1. y=1;</td>
</tr>
<tr>
<td>2. a=y; //0</td>
<td>2. b=x;</td>
</tr>
<tr>
<td>3. if (a==0) {</td>
<td>3. if (b==0) {</td>
</tr>
<tr>
<td>4. //cs1;</td>
<td>4. //cs2;</td>
</tr>
<tr>
<td>5. }</td>
<td>5. }</td>
</tr>
</tbody>
</table>

An SC execution

**memory event**

- `w(x,1)`
- `r(y,0)`
- `w(y,1)`

Shared memory

x=1  y=1

x=1

y=1
**Dekker Mutual Exclusion Protocol**

**Init:** $x = y = 0$

**Process 1**
1. $x = 1$
2. $a = y$ // 0
3. if (a == 0) {
   4. // cs1;
   5. }

**Process 2**
1. $y = 1$
2. $b = x$
3. if (b == 0) {
   4. // cs2;
   5. }

**Sequential Consistency (SC)**
- Shared Memory
- Processes perform read/write **atomically**
- execute instructions of one process in **program-order**
- interleave instructions of different processes
- read-from the latest write

**An SC execution**
- $w(x, 1)$
- $r(y, 0)$
- $w(y, 1)$
- $w(y, 1)$

**Shared memory**

$x = 1 \quad y = 1$
Shared Memory

Processes perform read/write atomically
execute instructions of one process in program-order
interleave instructions of different processes
read-from the latest write

Sequential Consistency (SC)

Dekker Mutual Exclusion Protocol

Process 1
1. x=1;
2. a=y;  // 0
3. if (a==0) {
4. //cs1;
5. }

Process 2
1. y=1;
2. b=x;
3. if (b==0) {
4. //cs2;
5. }

An SC execution

Init: x=y=0

x=1  y=1

Shared memory

memory event
w(x,1)

memory event
r(y,0)

memory event
w(y,1)
Dekker Mutual Exclusion Protocol

**Sequential Consistency (SC)**

- Shared Memory
- Processes perform read/write **atomically**
- execute instructions of one process in **program-order**
- **interleave** instructions of different processes
- **read-from** the latest write

**Process 1**
1. `x=1;`
2. `a=y; //0`
3. `if (a==0) {`
   4. `//cs1;`
5. `}`

**Process 2**
1. `v=1;`
2. `b=x; //1`
3. `if (b==0) {`
   4. `//cs2;`
5. `}`

**Init: x=y=0**

**An SC execution**
- `w(x,1)`
- `r(y,0)`
- `w(y,1)`
- `w(y,1)`
- `r(x,1)`
- `x=1` `y=1`
Dekker Mutual Exclusion Protocol

Init: $x = y = 0$

1. $x = 1$
2. $a = y$；//0
3. if ($a = 0$) {
   4. //cs1;
   5. }

Process 1

1. $y = 1$
2. $b = x$；//1
3. if ($b = 0$) {
   4. //cs2;
   5. }

Process 2

Sequential Consistency (SC)

- Shared Memory
- Processes perform read/write atomically
- execute instructions of one process in program-order
- interleave instructions of different processes
- read-from the latest write

An SC execution

$x = 1$ $y = 1$

Shared memory
Dekker Mutual Exclusion Protocol

Sequential Consistency (SC)
- Shared Memory
- Processes perform read/write atomically
- execute instructions of one process in program-order
- interleave instructions of different processes
- read-from the latest write

Init: $x=y=0$

Process 1
1. $x=1$;
2. $a=y$; // 0
3. if ($a==0$) {
4. // cs1;
5. }

Process 2
1. $y=1$;
2. $b=x$; // 1
3. if ($b==0$) {
4. // cs2;
5. }

An SC execution
- $w(x,1)$
- $r(y,0)$
- $w(y,1)$
- $r(x,1)$

$x=1$  $y=1$

Shared memory
Dekker Mutual Exclusion Protocol

**Process 1**
1. \(x=1;\)
2. \(a=y; \quad //^0\)
3. if \((a==0)\) {
4. \ ///cs1;
5. }

**Process 2**
1. \(y=1;\)
2. \(b=x; \quad //^1\)
3. if \((b==0)\) {
4. \ ///cs2;
5. }

**Init:** \(x=y=0\)

**An SC execution**

- Memory event \(w(x,1)\)
- Memory event \(r(y,0)\)
- Memory event \(w(y,1)\)
- Memory event \(w(y,1)\)
- Memory event \(r(x,1)\)

Shared memory

\(x=1 \quad y=1\)
Dekker Mutual Exclusion Protocol

Init: $x=y=0$

Process 1
1. $x=1$;
2. $a=y$;  // 0
3. if ($a==0$) {
4.  // cs1;
5. }

Process 2
1. $y=1$;
2. $b=x$;  // 1
3. if ($b==0$) {
4.  // cs2;
5. }

Assertion A: not (cs1 && cs2)

An SC execution

Shared memory

$x=1$  $y=1$
Dekker Mutual Exclusion Protocol

1. Process 1:
   1. x=1;
   2. a=y;  //0
   3. if (a==0) {
      4. //cs1;
      5. }
   4. //cs2;

2. Process 2:
   1. y=1;
   2. b=x;  //1
   3. if (b==0) {
      4. //cs1;
      5. }
   4. //cs2;

Init: x=y=0

Assertion A: not (cs1 && cs2)

An SC execution:
- x=1
- y=1

Shared memory

w(x,1) r(y,0) w(y,1) r(x,1)

memory event memory event memory event memory event
Sequential Consistency

Sequential Consistency (SC)

- Shared Memory
- Processes perform read/write **atomically**
- execute instructions of one process in **program-order**
- **interleave** instructions of different processes
- **read-from** the latest write
Sequential Consistency

- Simple
- Intuitive

Sequential Consistency (SC)

- Shared Memory
- Processes perform read/write atomically
- execute instructions of one process in program-order
- interleave instructions of different processes
- read-from the latest write
Sequential Consistency

- Shared Memory
- Processes perform read/write **atomically**
- execute instructions of one process in **program-order**
- **interleave** instructions of different processes
- **read-from** the latest write

Sequential Consistency (SC)

- **simple**
- **intuitive**
- **costly**
- **unrealistic**
Sequential Consistency (SC)

- Shared Memory
- Processes perform read/write atomically
- execute instructions of one process in program-order
- interleave instructions of different processes
- read-from the latest write
Sequential Consistency

Weakly Consistent Systems

- **Microprocessors:**
  - TSO, POWER, ARM, ...
- **Weak Cache Protocols:**
  - TSO-CC, Racer, SISD, ...
- **Programming Languages:**
  - C11, Java, ...
- **Distributed Data Stores:**
  - Amazon, Facebook, Google, ...
Weakly Consistent Systems

- Microprocessors:
  - TSO, POWER, ARM, ...
- Weak Cache Protocols:
  - TSO-CC, Racer, SISD, ...
- Programming Languages:
  - C11, Java, ...
- Distributed Data Stores:
  - Amazon, Facebook, Google, ...
Outline

- Concurrent Programs
- **Total Store Order (TSO)**
- Correctness - Robustness
- Program Repair
- Perspectives
- Conclusions
Weak Memory Models

- Modern processors and/or compilers:
  - Reorder instructions
  - Use caches and buffers
Weak Memory Models

- Modern processors and/or compilers:
  - Reorder instructions
  - Use caches and buffers

- Behaviors described by weak memory models:
  - Here: Total Store Ordering (TSO) memory model
Weak Memory Models

• Modern processors and/or compilers:
  • Reorder instructions
  • Use caches and buffers

• Behaviors described by **weak memory models**:
  • Here: Total Store Ordering (**TSO**) memory model

\[ w(y,1) \]
\[ r(x,0) \]
Weak Memory Models

- Modern processors and/or compilers:
  - Reorder instructions
  - Use caches and buffers

- Behaviors described by weak memory models:
  - Here: Total Store Ordering (TSO) memory model
Weak Memory Models

- Modern processors and/or compilers:
  - Reorder instructions
  - Use caches and buffers

- Behaviors described by weak memory models:
  - Here: Total Store Ordering (TSO) memory model
Weak Memory Models

• Modern processors and/or compilers:
  • Reorder instructions
  • Use caches and buffers

• Behaviors described by weak memory models:
  • Here: Total Store Ordering (TSO) memory model

\[ w(y,1) \rightarrow r(x,0) \rightarrow w(y,1) \rightarrow r(x,0) \]

write-read reordering
Weak Memory Models

• Modern processors and/or compilers:
  • Reorder instructions
  • Use caches and buffers

• Behaviors described by **weak memory models**:
  • Here: Total Store Ordering (TSO) memory model
Potential Bad Behaviours

Init: \(x=y=0\)

Process 1
1. \(x=1;\)
2. \(a=y;\)
3. if \((a==0)\) {
4. //cs1;
5. }

Process 2
1. \(y=1;\)
2. \(b=x;\)
3. if \((b==0)\) {
4. //cs2;
5. }

Specification S: \(\text{not (cs1 && cs2)}\)

A TSO execution

\(x=0\) \(\quad y=0\)

Shared memory
Potential Bad Behaviours

[Diagram showing a TSO execution]

Init: \(x=0, y=0\)

Process 1:
1. \(x=1;\)
2. \(a=y;\)
3. if \((a==0)\) {
   4. //cs1;
   5. }

Process 2:
1. \(b=x;\)
2. \(y=1;\)
3. if \((b==0)\) {
   4. //cs2;
   5. }

Specification S: not (cs1 && cs2)

Shared memory
\(x=0, y=0\)
Potential Bad Behaviours

**Specification S:** not (cs1 && cs2)
Potential Bad Behaviours

Init: $x=y=0$

Process 1
1. $x=1$;
2. $a=y$;
3. if $(a==0)$ {
4. //cs1;
5. }

Process 2
1. $b=x$;
2. $y=1$;
3. if $(b==0)$ {
4. //cs2;
5. }

Specification S: not (cs1 && cs2)

A TSO execution

$x=0$  $y=0$

Shared memory
Potential Bad Behaviours

Init: \(x=y=0\)

Process 1
1. \(x=1;\)
2. \(a=y;\)
3. if \((a==0)\) {
4. //cs1;
5. }

Process 2
1. \(b=x; \quad //0\)
2. \(y=1;\)
3. if \((b==0)\) {
4. //cs2;
5. }

A TSO execution
\(r(x,0)\)

Specification S: not (cs1 && cs2)
Potential Bad Behaviours

Init: $x=y=0$

Process 1
1. $x=1$
2. $a=y$
3. if ($a==0$) {
   4. //cs1;
5. }

Process 2
1. $b=x$; //0
2. $y=1$
3. if ($b==0$) {
   4. //cs2;
5. }

Specification S: not (cs1 && cs2)

A TSO execution

$x=0$  $y=0$

Shared memory
Potential Bad Behaviours

Init: x=y=0

Process 1
1. x=1;
2. a=y;
3. if (a==0) {
4. //cs1;
5. }

Process 2
1. b=x;   //0
2. y=1;
3. if (b==0) {
4. //cs2;
5. }

A TSO execution

r(x,0)

Specification S: not (cs1 && cs2)

x=0  y=0

Shared memory
Potential Bad Behaviours

Init: x=y=0

Process 1
1. x=1;
2. a=y;
3. if (a==0) {
4. //cs1;
5. }

Process 2
1. b=x;  //0
2. y=1;
3. if (b==0) {
4. //cs2;
5. }

Specification S: not (cs1 && cs2)

A TSO execution

r(x,0)
w(x,1)

x=1  y=0

Shared memory
Potential Bad Behaviours

**Init:** \( x = y = 0 \)

**Process 1**
1. \( x = 1; \)
2. \( a = y; \)
3. if \( (a == 0) \) {
   4. //cs1;
5. }

**Process 2**
1. \( b = x; \) //0
2. \( y = 1; \)
3. if \( (b == 0) \) {
   4. //cs2;
5. }

**Specification S:** \( \text{not} \ (cs1 \ & \& \ cs2) \)

**A TSO execution**
- \( w(x, 1) \)
- \( r(x, 0) \)

**Shared memory:** \( x = 1 \quad y = 0 \)
Potential Bad Behaviours

**Specification S:** not (cs1 && cs2)

**Init:** \( x=y=0 \)

**Process 1**
1. \( x=1 \);
2. \( a=y \);
3. if (a==0) {
4. //cs1;
5. }

**Process 2**
1. \( b=x \); //0
2. \( y=1 \);
3. if (b==0) {
4. //cs2;
5. }

A TSO execution

\( r(x,0) \)
\( w(x,1) \)

\( x=1 \) \( y=0 \)

Shared memory
Potential Bad Behaviours

Specification S: not (cs1 && cs2)

Init: x=y=0

Process 1
1. x=1;
2. a=y; //0
3. if (a==0) {
4. //cs1;
5. }

Process 2
1. b=x; //0
2. y=1;
3. if (b==0) {
4. //cs2;
5. }

A TSO execution

w(x,1)
r(x,0)
r(y,0)

shared memory

x=1 y=0
Potential Bad Behaviours

Init: $x = y = 0$

Process 1
1. $x = 1$
2. $a = y$  
3. if ($a = 0$) {
   4.  //cs1;
   5. }

Process 2
1. $b = x$  
2. $y = 1$
3. if ($b = 0$) {
   4.  //cs2;
   5. }

Specification S: not (cs1 && cs2)

A TSO execution

Shared memory

$x = 1$  $y = 0$
Potential Bad Behaviours

Init: x=y=0

Process 1
1. x=1;
2. a=y;  //0
3. if (a==0) {
4.   //cs1;
5. }

Process 2
1. b=x;  //0
2. y=1;
3. if (b==0) {
4.   //cs1;
5. }

Specification S: not (cs1 && cs2)

A TSO execution

\[r(x,0)\]
\[w(x,1)\]
\[r(y,0)\]

x=1  y=0

Shared memory
Potential Bad Behaviours

Init: \( x=y=0 \)

**Process 1**
1. \( x=1; \)
2. \( a=y; \quad //^0 \)
3. if (\( a==0 \)) {
   4. \( //cs1; \)
   5. }

**Process 2**
1. \( b=x; \quad //^0 \)
2. \( y=1; \)
3. if (\( b==0 \)) {
   4. \( //cs1; \)
   5. }

**Specification S:** not (\( cs1 \&\& cs2 \))

A TSO execution:
- \( w(x,1) \)
- \( r(y,0) \)
- \( r(x,0) \)

Shared memory: \( x=1 \quad y=1 \)
Potential Bad Behaviours

Init: $x=y=0$

Process 1
1. $x=1$;
2. $a=y$;  // 0
3. if ($a==0$) {
4.    //cs1;
5. }

Process 2
1. $b=x$;  // 0
2. $y=1$;
3. if ($b==0$) {
4.    //cs2;
5. }  

A TSO execution

Specification S: not (cs1 && cs2)

Shared memory

$x=1$  $y=1$
Potential Bad Behaviours

Init: x=y=0

Process 1
1. x=1;
2. a=y;  //0
3. if (a==0) {
4.  //cs1;
5. }

Process 2
1. b=x;  //0
2. y=1;
3. if (b==0) {
4.  //cs2;
5. }

A TSO execution

x=1  y=1

Shared memory

Specification S: not (cs1 && cs2)
Potential Bad Behaviours

Init: x=y=0

Process 1
1. x=1;
2. a=y;  //0
3. if (a==0) {
4. //cs1;
5. }

Process 2
1. b=x;  //0
2. y=1;
3. if (b==0) {
4. //cs2;
5. }

A TSO execution

\[ r(x,0) \]
\[ w(x,1) \]
\[ r(y,0) \]
\[ w(y,1) \]

Specification S: \textbf{not (cs1 && cs2)}
Potential Bad Behaviours

Init: x=y=0

Process 1
1. x=1;
2. a=y;  //0
3. if (a==0) {
   4. //cs1;
   5. }

Process 2
1. b=x;  //0
2. y=1;
3. if (b==0) {
   4. //cs2;
   5. }

A TSO execution

r(x,0) → w(x,1) → r(y,0) → w(y,1)

x=1  y=1

Speciﬁcation S: not (cs1 && cs2)
Potential Bad Behaviours

Init: x=y=0

Process 1
1. x=1;
2. a=y; //0
3. if (a==0) {
4. //cs1;
5. }

Process 2
1. b=x; //0
2. y=1;
3. if (b==0) {
4. //cs2;
5. }

Specification S: not (cs1 && cs2)

A TSO execution

x=1 y=1

Shared memory
Potential Bad Behaviours

**Init:** \( x = y = 0 \)

**Process 1**
1. \( x = 1; \)
2. \( a = y; \)  // 0
3. if \((a == 0)\) {
   4.  // cs1;
   5.  }

**Process 2**
1. \( b = x; \)  // 0
2. \( y = 1; \)
3. if \((b == 0)\) {
   4.  // cs2;
   5.  }

**Specification S:** not (cs1 \&\& cs2)

**A TSO execution**
- \( w(x, 1) \)
- \( r(x, 0) \)
- \( w(y, 1) \)
- \( r(y, 0) \)
- \( w(x, 1) \)

**Shared memory**
- \( x = 1 \)
- \( y = 1 \)
Potential Bad Behaviours
Potential Bad Behaviours

Challenge

Program Repair
Potential Bad Behaviours

Challenge

Program Repair

retrieve correctness
but
maintain efficiency
Potential Bad Behaviours

Program Repair

retrieving \textit{correctness}
but
maintain \textit{efficiency}

use synchronization primitives
e.g., fences
Potential Bad Behaviours

Program Repair

- retrieve **correctness**
- but
- maintain **efficiency**

- use synchronization primitives

- synthesize the necessary set of fences

Challenge
e.g., fences
Correctness Problem

• How to make a program run correctly under TSO?
Correctness Problem

• How to make a program run correctly under TSO?
  • Automatic inference of memory **fences**
Correctness Problem

• How to make a program run correctly under TSO?
  • Automatic inference of memory fences
Naive Solution

- Insert a fence after each instruction

Init: $x=y=0$

Process 1
1. $x=1$;
2. fence;
3. $ry=y$;
4. fence;
5. if ($ry==0$) {
6. fence;
7. //cs1;
8. fence;
9. }

Process 2
1. $y=1$;
2. fence;
3. $rx=x$;
4. fence;
5. if ($rx==0$) {
6. fence;
7. //cs2;
8. fence;
9. }

Specification S: $\neg (cs1 \&\& cs2)$
Naive Solution

• Insert a fence after each instruction

**Init: x=y=0**

**Specification S:** not (cs1 && cs2)
Naive Solution

- Insert a fence after each instruction

**Specification S:** not (cs1 && cs2)
Naive Solution

- Insert a fence after each instruction

Init: x=y=0

Process 1
1. x=1;
2. fence;
3. ry=y;
4. fence;
5. if (ry==0) {
6. fence;
7. //cs1;
8. fence;
9. }

Process 2
1. v=1;
2. fence;
3. rx=x;
4. fence;
5. if (rx==0) {
6. fence;
7. //cs2;
8. fence;
9. }

Specification S: not (cs1 && cs2)
Naive Solution

- Insert a fence after each instruction

Init: x=y=0

Specification S: not (cs1 && cs2)

Process 1
1. x=1;
2. fence;
3. ry=y;
4. fence;
5. if (ry==0) {
6. fence;
7. //cs1;
8. fence;
9. }

Process 2
1. v=1;
2. fence;
3. rx=x;
4. fence;
5. if (rx==0) {
6. fence;
7. //cs2;
8. fence;
9. }
Naive Solution

• Insert a fence after each instruction

Disadvantage

• Fences are expensive [Herlihy&Shavit]:
  • Normal instructions: 1-6 cycles
  • Fence instructions > 100 cycles

Challenge

• Insert a minimal set of fences to repair the program
Naive Solution

- Insert a fence after each instruction

**Disadvantage**

- Fences are expensive [Herlihy&Shavit]:
  - Normal instructions: 1-6 cycles
  - Fence instructions > 100 cycles

**Challenge**

- Insert a minimal set of fences to repair the program
Naive Solution

- Insert a fence after each instruction

Disadvantage

- Fences are expensive [Herlihy&Shavit]:
  - Normal instructions: 1-6 cycles
  - Fence instructions > 100 cycles

Challenge

- Insert a minimal set of fences to make the program correct
Outline

- Concurrent Programs
- Total Store Order (TSO)
- Correctness - Robustness
- Program Repair
- Perspectives
- Conclusions
Robustness

Challenge

- Insert a minimal set of fences to repair the program

optimality

make the program correct
Robustness

Requirements

- Optimal
- Automatic
- Efficient

Challenge

- Insert a minimal set of fences to repair the program

make the program correct

optimality
Robustness

Challenge

- Insert a minimal set of fences to repair the program

Requirements

- Optimal
- Automatic
- Efficient

make the program correct

Specification S: not (cs1 && cs2)

Process 1

1. x=1;
2. fence;
3. ry=y;
4. if (ry==0) {
5. //cs1;
6. }

Process 2

1. y=1;
2. fence;
3. rx=x;
4. if (rx==0) {
5. //cs2;
6. }

Init: x=y=0
Robustness

**Challenge**
- Insert a minimal set of fences to repair the program

**Requirements**
- Optimal
- Automatic
- Efficient

**Specification S:** \( \text{not (cs1 && cs2)} \)

**Process 1**
1. \( x=1; \)
2. fence;
3. \( ry=y; \)
4. if (\( ry==0 \)) {
5. //cs1;
6. }

**Process 2**
1. \( y=1; \)
2. fence;
3. \( rx=x; \)
4. if (\( rx==0 \)) {
5. //cs2;
6. }

**Init:** \( x=y=0 \)
Robustness

**Challenge**

- Insert a minimal set of fences to repair the program

**Requirements**

- Optimal
- Automatic
- Efficient

make the program correct
Robustness

Challenge

- Insert a minimal set of fences to repair the program

Requirements

- Optimal
- Automatic
- Efficient

make the program correct
**Robustness**

**Challenge**
- Insert a minimal set of fences to repair the program

**Requirements**
- Optimal
- Automatic
- Efficient
Robustness

Challenge

• Insert a minimal set of fences to repair the program

Requirements

• Optimal
• Automatic
• Efficient

P: Program

make the program correct
Robustness makes the program correct

Challenge

- Insert a minimal set of fences to repair the program

Requirements

- Optimal
- Automatic
- Efficient

P: Program

P is robust
Robustness

Challenge

• Insert a minimal set of fences to repair the program

Requirements

• Optimal
• Automatic
• Efficient

P: Program

P is robust

P has identical behaviors under SC & TSO
Robustness

Challenge

- Insert a minimal set of fences to repair the program

Requirements

- Optimal
- Automatic
- Efficient

P: Program

P is robust

P has identical behaviors under SC & TSO
Robustness

P has identical behaviors under SC & TSO
Robustness

P has identical behaviors under SC & TSO

Init: x=y=0

Process 1
1. x=1;
2. a=y;  //0
3. if (a==0) {
   4. //cs1;
   5. }

Process 2
1. b=x;  //0
2. y=1;
3. if (b==0) {
   4. //cs2;
   5. }

Specification S: not (cs1 && cs2)

A TSO execution

Shared memory

x=1  y=1
Robustness

P has identical behaviors under SC & TSO

A TSO execution

\[ r(x,0) \]
\[ w(x,1) \]
\[ r(y,0) \]
\[ w(y,1) \]
Robustness

$P$ has **identical behaviors** under SC & TSO

A TSO execution:

$r(x,0) 
\downarrow
w(x,1) 
\downarrow
r(y,0) 
\downarrow
w(y,1) 
\rightarrow$

$x=0 \quad y=0$

trace

graph
Robustness

P has identical behaviors under SC & TSO

A TSO execution

\[ r(x,0) \]
\[ w(x,1) \]
\[ r(y,0) \]
\[ w(y,1) \]

x=0 y=0
Robustness

P has **identical behaviors** under SC & TSO

A TSO execution

\[ r(x,0) \rightarrow w(x,1) \rightarrow r(y,0) \rightarrow w(y,1) \rightarrow r(x,0) \]

\[ w(x,1) \rightarrow w(y,1) \]

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

trace

graph
Robustness

P has identical behaviors under SC & TSO

A TSO execution

x=0  y=0
Robustness

P has **identical behaviors** under SC & TSO

A TSO execution

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

\[
\begin{align*}
r(x, 0) \\
w(x, 1) \\
r(y, 0) \\
w(y, 1)
\end{align*}
\]

\[
\begin{align*}
r(y, 0) \\
w(x, 1) \\
r(x, 0) \\
w(y, 1)
\end{align*}
\]
Robustness

P has **identical behaviors** under SC & TSO

A TSO execution

Initial value

Trace

Graph

Memory event

$x=0, y=0$
P has identical behaviors under SC & TSO

A TSO execution

\[ \begin{align*}
    &w(x,1) \\
    &r(y,0) \\
    &w(y,1) \\
    &r(x,0) \\
\end{align*} \]

\[ \begin{align*}
    &w(x,1) \\
    &r(y,0) \\
    &w(y,1) \\
\end{align*} \]

po: program-order

trace

graph

memory event

initial value

x=0 y=0

Robustness
P has **identical behaviors** under SC & TSO

A TSO execution

\[ \begin{align*}
  & r(x,0) \\
  & w(x,1) \\
  & r(y,0) \\
  & w(y,1)
\end{align*} \]

\[ \begin{align*}
  & w(x,1) \\
  & r(y,0) \\
  & w(y,1) \\
  & r(x,0)
\end{align*} \]

- Initial value
- Program-order (po)
- Order of events per process
- Trace
- Memory event
- Graph

**x=0  y=0**
Robustness

P has **identical behaviors** under SC & TSO

A TSO execution

\[ \begin{align*}
    r(x,0) &\quad w(x,0) &\quad w(x,1) &\quad r(x,0) \\
    w(y,1) &\quad r(y,0) &\quad w(y,1) &\quad r(y,0)
\end{align*} \]

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

- initial value
- program-order
- order of events per process
- memory event
- trace
- graph
Robustness

P has identical behaviors under SC & TSO

A TSO execution

rf: read-from

x=0 y=0

initial value

order of events per process

po: program-order

memory event

trace

graph

w(x,1) → r(y,0) → w(y,1) → r(x,0)
Robustness

P has identical behaviors under SC & TSO

A TSO execution

\[ r(x,0) \]
\[ w(x,1) \]
\[ r(y,0) \]
\[ w(y,1) \]

rf: read-from

which write is the source of a read

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

\[ w(x,1) \]
\[ w(y,1) \]

\[ r(y,0) \]
\[ r(x,0) \]

initial value

po: program-order

order of events per process

memory event

trace

graph

x=0  y=0
Robustness

P has identical behaviors under SC & TSO

A TSO execution

\[
\begin{align*}
&\text{Initial value} \\
&\text{Memory event} \\
&\text{Order of events per process} \\
&\text{Program-order} \\
&\text{Trace} \\
&\text{Graph}
\end{align*}
\]
Robustness

P has identical behaviors under SC & TSO

A TSO execution

\[ \begin{align*}
& r(x,0) \\
& w(x,1) \\
& r(y,0) \\
& w(y,1)
\end{align*} \]

which write is the source of a read

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

initial value

coherece order

initial value

po: program-order

order of events per process

memory event

trace

graph
Robustness

P has identical behaviors under SC & TSO

A TSO execution

\begin{align*}
r(x,0) & \rightarrow w(x,1) & w(y,1) & \rightarrow r(y,0) \\
\end{align*}

which write is the source of a read

\begin{align*}
\text{rf: read-from} & \\
\text{which write is the source of a read} & \\
\end{align*}

initial value

coherence order

order of writes

order of events per process

program-order

trace

graph

memory event
Robustness

P has **identical behaviors** under SC & TSO

A TSO execution

- Initial value
- Coherence order
- Order of writes
- Program-order
- Order of events per process
- Trace
- Memory event
- Graph

```
\text{rf: read-from}
\text{initial value}
\text{coherence order}
\text{order of writes}
\text{po: program-order}
\text{order of events per process}
\text{trace}
\text{memory event}
\text{graph}
```

\text{w(x,1)} \rightarrow \text{w(y,1)}

A TSO execution:

- \text{r(x,0)} \rightarrow \text{w(x,1)}
- \text{w(x,1)} \rightarrow \text{r(y,0)}
- \text{r(y,0)} \rightarrow \text{w(y,1)}
- \text{w(y,1)} \rightarrow \text{r(x,0)}

\text{x=0} \quad \text{y=0}
P has identical behaviors under SC & TSO

A TSO execution

\[
\begin{align*}
\mathbf{r}(x,0) & \xrightarrow{\text{rf: read-from}} \mathbf{w}(x,1) \\
\mathbf{w}(x,1) & \xrightarrow{\text{rf: read-from}} \mathbf{r}(y,0) \\
\mathbf{r}(y,0) & \xrightarrow{\text{rf: read-from}} \mathbf{w}(y,1) \\
\mathbf{w}(y,1) & \xrightarrow{\text{rf: read-from}} \mathbf{r}(x,0)
\end{align*}
\]

Initial value

Coherence order

Order of writes

Program-order

Order of events per process

Trace

Memory event

Graph

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

P has **identical behaviors** under SC & TSO

A TSO execution

- **rf:** read-from
- **w(x,1)**
- **r(y,0)**
- **w(y,1)**
- **r(x,0)**

Initial value
- coherence order
- order of writes
- program-order
- order of events per process

*initial value*
*coherence order*
*order of writes*
*program-order*
*order of events per process*

Trace
Graph
Memory event

**x=0 y=0**
Robustness

P has **identical behaviors** under SC & TSO
Robustness

Program P

P has **identical behaviors** under SC & TSO
Robustness

P has identical behaviors under SC & TSO
Robustness

Program P

TSO execution

r(x,0)

w(x,1)

r(y,0)

w(y,1)

P has **identical behaviors** under SC & TSO
Robustness

P has **identical behaviors** under SC & TSO

Program P

TSO execution:
- $r(x,0)$
- $w(x,1)$
- $r(y,0)$
- $w(y,1)$

Trace:

Init
- $w(x,1)$
- $w(y,1)$
- $r(y,0)$
- $r(x,0)$
Program P

P has **identical behaviors** under SC & TSO

Robustness
Program P

TSO executions

Traces

Init

\begin{align*}
    w(x,1) & \rightarrow r(y,0) \\
    w(y,1) & \rightarrow r(x,0) \\
\end{align*}
Robustness

P has identical behaviors under SC & TSO

Program P

TSO executions

w(x,1) r(y,0) w(y,1) r(x,0)

traces

w(x,1) r(y,0) w(y,1) r(x,0)

Init
Robustness

P has identical behaviors under SC & TSO
Program P

TSO executions

SC executions

P has identical behaviors under SC & TSO

Robustness
Robustness

Program P

TSO executions

scenarios:

Init

w(x,0)

w(y,0)

w(x,1)

w(y,1)

r(x,0)

r(y,0)

SC executions

scenarios:

Init

w(x,1)

w(y,1)

w(x,0)

w(y,0)

r(x,0)

r(y,0)

P has identical behaviors under SC & TSO

= ?
Program P

Robustness

P has identical behaviors under SC & TSO

TSO executions

SC executions

traces

?
Robustness

P has **identical behaviors** under SC & TSO

Program P

TSO executions

SC executions

traces

?
Robustness

P has identical behaviors under SC & TSO

Init: x=y=0

Process 1
1. x=1;
2. a=y;
3. if (a==0) {
   4. //cs1;
   5. }

Process 2
1. y=1;
2. b=x;
3. if (b==0) {
   4. //cs2;
   5. }

1. x=1;
2. a=y;
3. if (a==0) {
   4. //cs1;
   5. }

SC executions

w(x,1) → r(y,0) → w(y,1)

traces

w(x,1) → w(y,1) → r(y,0) → r(x,0)
Robustness

P has identical behaviors under SC & TSO

Init: x=y=0

Process 1
1. x=1;
2. a=y;
3. if (a==0) {
4. //cs1;
5. }

Process 2
1. y=1;
2. x=x;
3. if (b==0) {
4. //cs2;
5. }

Not Robust

1. x=1;
2. a=y;
3. if (a==0) {
4. //cs1;
5. }

SC executions
traces

1. x=1;
2. a=y;
3. if (a==0) {
4. //cs1;
5. }

Not Robust
Robustness

P has identical behaviors under SC and TSO.

Init: \( x=y=0 \)

Process 1

1. \( x=1 \);
2. \( a=y \);
3. if \( (a==0) \) {
   4. \// cs1;
   5. }

Process 2

1. \( y=1 \);
2. \( b=x \);
3. if \( (b==0) \) {
   4. \// cs2;
   5. }

Not Robust

\[
\begin{align*}
1. \ x&=1; \\
2. \ a&=y; \\
3. \text{if } (a==0) \
   &\{ \\
   4. \ //cs1; \\
   5. \ }
\end{align*}
\]

\[
\begin{align*}
1. \ y&=1; \\
2. \ b&=x; \\
3. \text{if } (b==0) \
   &\{ \\
   4. \ //cs2; \\
   5. \ }
\end{align*}
\]

SC executions

traces

\[
\begin{align*}
\text{Init: } x&=y=0 \\
&w(x,1) \\
r(y,0) \\
&w(y,1)
\end{align*}
\]

\[
\begin{align*}
\text{Init: } x&=y=0 \\
&w(x,1) \\
r(y,0) \\
&w(y,1)
\end{align*}
\]
Robustness

P has identical behaviors under SC & TSO

Init: \(x=y=0\)

Process 1
1. \(x=1;\)
2. \(a=y;\)
3. if (\(a==0\)) {
4. //cs1;
5. }

Process 1
1. \(x=1;\)
2. fence;
3. \(ry=y;\)
4. if (\(ry==0\)) {
5. //cs1;
6. }

Process 2
1. \(y=1;\)
2. fence;
3. \(rx=x;\)
4. if (\(rx==0\)) {
5. //cs2;
6. }

Init: \(x=y=0\)

Not Robust
Robustness

P has identical behaviors under SC & TSO

Init: $x=y=0$

Process 1
1. $x=1$;
2. $a=y$;
3. if ($a==0$) {
4. //cs1;
5. }

Process 2
1. $y=1$;
2. fence;
3. $ry=y$;
4. if ($ry==0$) {
5. //cs1;
6. }

Not Robust

Init: $x=y=0$

Process 1
1. $y=1$;
2. fence;
3. $rx=x$;
4. if ($rx==0$) {
5. //cs2;
6. }

Process 2
1. $x=1$;
2. $a=y$;
3. if ($a==0$) {
4. //cs1;
5. }

Robust
Robustness

P has identical behaviors under SC & TSO

Init: \(x=y=0\)

Process 1:
1. \(x=1;\)
2. \(a=y;\)
3. if (a==0) {
4. //cs1;
5. }

Not Robust

Process 2:
1. \(y=1;\)
2. fence;
3. \(ry=y;\)
4. if (ry==0) {
5. //cs1;
6. }

Process 1:
1. \(y=1;\)
2. fence;
3. \(rx=x;\)
4. if (rx==0) {
5. //cs2;
6. }

Init: \(x=y=0\)

Robust

How to check?
Robustness

Program P

NO/witness (TSO)

Robust?

How to check?
How to check? [Robustness]

Program P

NO/witness (TSO)

YES
Robustness

How to check?

Program P

NO/witness (TSO)

YES
Robustness

How to check?

Program P

NO/witness (TSO)

compiler

YES
Robustness

How to check?

Program P

NO/witness (TSO)

new program

P' compiler

P
Robustness

How to check?

Program P

NO/witness (TSO)

new program

compiler

assertion
Robustness

How to check?

Program P

NO/witness (TSO)

new program

compiler

assertion

P'

P' |= \phi?

P
How to check?

Robustness

Program P

NO/witness (TSO)

P

P' |= \phi?

compiler

new program

assertion

under SC

\phi

P'

YES
Robustness

How to check?

Program P

NO/witness (TSO)

P

P'

P' |= \phi?

new program

compiler

assertion

under SC
Robustness

How to check?

Program P

NO/witness (TSO)

NO/witness (SC)

P' \models \phi?

P'

P

new program

compiler

assertion

under SC

YES

YES
Robustness

How to check?

Program P

NO/witness (TSO)

NO/witness (SC)

P' \models \phi?

new program

compiler

compiler

assertion

under SC

P'

P
Program Repair (based on Robustness)
Program Repair (based on Robustness)
Program Repair (based on Robustness)

TSO execution

1. x=1;
2. a=y;
3. if (a==0) {
4.   //cs1;
5. }

1. y=1;
2. b=x;
3. if (b==0) {
4.   //cs2;
5. }

r(x,0) → w(x,1) → r(y,0) → r(y,0) → w(y,1)
Program Repair (based on Robustness)

1. x=1;
2. a=y;
3. if (a==0) {
   4. //cs1;
   5. }

1. y=1;
2. b=x;
3. if (b==0) {
   4. //cs2;
   5. }

w(x,1)

w(y,1)

r(y,0)

w(y,1)

w(x,1)

r(x,0)

TSO execution
Program Repair (based on Robustness)

TSO execution

1. \( x=1; \)
2. \( a=y; \)
3. if (a==0) {
4.    \(/ /cs1; \)
5. }

1. \( y=1; \)
2. \( b=x; \)
3. if (b==0) {
4.    \(/ /cs2; \)
5. }

w(x,1) r(y,0) w(y,1) r(x,0)

Re-ordering
Program Repair (based on Robustness)

1. \texttt{x=1;}
2. \texttt{a=y;}
3. \texttt{if (a==0) {}
4. \texttt{if (b==0) { //cs2;}
5. \texttt{}}

\textbf{TSO execution}

\begin{itemize}
  \item \texttt{w(x,1)}
  \item \texttt{r(y,0)}
  \item \texttt{w(y,1)}
  \item \texttt{r(x,0)}
\end{itemize}

\textbf{witness analysis}

\textbf{repair}

\textbf{Re-ordering}
Program Repair (based on Robustness)

1. \( y = 1; \)
2. \( b = x; \)
3. if (\( a = 0 \)) {
   4. //cs1;
   5. }

TSO execution

1. \( x = 1; \)
2. \( a = y; \)
3. if (\( a = 0 \)) {
   4. //cs1;
   5. }

witness analysis

1. \( y = 1; \)
2. fence
3. \( b = x; \)
4. if (\( b = 0 \)) {
   4. //cs2;
   5. }

re-ordering

1. \( x = 1; \)
2. \( a = y; \)
3. if (\( a = 0 \)) {
   4. //cs1;
   5. }

repair
Program Repair (based on Robustness)

**TSO execution**

1. `x=1;`
2. `a=y;`
3. if `(a==0)` {
   4. //cs1;
   5. }

**witness analysis**

1. `y=1;`
2. `b=x;`
3. if `(b==0)` {
   4. //cs2;
   5. }

**Re-ordering**

1. `x=1;`
2. `a=y;`
3. if `(a==0)` {
   4. //cs1;
   5. }

**repair**

1. `y=1;`
2. `fence`
3. `b=x;`
4. if `(b==0)` {
   4. //cs1;
   5. //cs2;
   6. }

w(x,1)
r(y,0)
w(y,1)
r(x,0)

w(x,1)
r(y,0)
w(y,1)
Outline

• Concurrent Programs
• Total Store Order (TSO)
• Correctness - Robustness

• **Program Repair**
• Perspectives
• Conclusions
Program Repair (based on Robustness)
Program Repair (based on Robustness)

**CEGAR:**
Counter-Example Guided Abstraction Refinement
Program Repair (based on Robustness)

CEGAR:
Counter-Example Guided
Abstraction Refinement

1. x = 1;
2. a = y;
3. if (a == 0) {
4. // cs1;
5. }

1. y = 1;
2. b = x;
3. if (b == 0) {
4. // cs2;
5. }

Yes

witness
(TSO)

witness
analysis

robust?

Fences

Re-orderings

repair
Program Repair (based on Robustness)

CEGAR:
Counter-Example Guided
Abstraction Refinement

1. x=1;
2. a=y;
3. if (a==0) {
   4. //cs1;
   5. }
4. //cs2;
5. }

1. y=1;
2. b=x;
3. if (b==0) {
   4. //cs1;
   5. }

w(x,1)
r(x,0)
w(x,0)

r(y,0)
w(y,1)

witness (TSO)

Yes

robust?

repair

Fences

Re-orderings

witness analysis
Program Repair (based on Robustness)

CEGAR:
Counter-Example Guided
Abstraction Refinement

1. x=1;
2. a=y;
3. if (a==0) {
   4. //cs1;
   5. }
4. //cs2;
5. }

1. y=1;
2. b=x;
3. if (b==0) {
   4. //cs2;
   5. }

w(x,1)
r(y,0)
w(y,1)
r(x,0)

witness analysis

Yes

Fences

re-orderings

repair

witness (TSO)
Program Repair (based on Robustness)

**CEGAR:**
Counter-Example Guided Abstraction Refinement

1. `x=1;`
2. `a=y;`
3. if `(a==0)` {
   4. `//cs1;`
   5. }
4. `//cs2;`
5. }

1. `y=1;`
2. `b=x;`
3. if `(b==0)` {
   4. `//cs;`
   5. }

Yes

Fences

Re-orderings

robust?

witness (TSO)

witness analysis

repair
Program Repair (based on Robustness)

CEGAR:
Counter-Example Guided
Abstraction Refinement

1. x=1;
2. a=y;
3. if (a==0) {
4. //cs1;
5. }

1. y=1;
2. b=x;
3. if (b==0) {
4. //cs2;
5. }

w(x,1)
r(y,0)
w(y,1)
r(x,0)
w(x,1)
r(y,0)
w(y,1)
Program Repair (based on Robustness)

CEGAR:
Counter-Example Guided
Abstraction Refinement

1. x=1;
2. a=y;
3. if (a==0) {
   4. //cs1;
   5. }
4. //cs2;
5. }

1. y=1;
2. b=x;
3. if (b==0) {
   4. //cs1;
   5. }
6. //cs2;

w(x,1) r(y,0) w(y,1) r(x,0)

Repair

Fences

Re-orderings

witness (TSO)

witness analysis

Yes
Program Repair (based on Robustness)

CEGAR:
Counter-Example Guided
Abstraction Refinement

1. x=1;
2. a=y;
3. if (a==0) {
4.   //cs1;
5. }

1. y=1;
2. b=x;
3. if (b==0) {
4.   //cs2;
5. }

1. y=1;
2. fence
3. b=x;
4. if (b==0) {
5.   //cs2;
6. }

w(x,1) r(y,0)

1. y=1;
2. fence
3. b=x;
4. if (b==0) {
5.   //cs2;
6. }

w(x,1) r(y,0)

r(x,0) w(y,1)
Program Repair (based on Robustness)

CEGAR:
Counter-Example Guided Abstraction Refinement

```
1. x=1;
2. a=y;
3. if (a==0) {
4.   //cs1;
5. }
1. y=1;
2. b=x;
3. if (b==0) {
4.   //cs2;
5. }
```

1. y=1;
2. fence
3. b=x;
4. if (b==0) {
5.   //cs2;
6. }

w(x,1) r(y,0) w(y,1) r(x,0)

Program Repair (based on Robustness)

CEGAR:
Counter-Example Guided Abstraction Refinement

1. x=1;
2. a=y;
3. if (a==0) {
4.   //cs1;
5. }
1. y=1;
2. b=x;
3. if (b==0) {
4.   //cs2;
5. }

1. y=1;
2. b=x;
3. if (b==0) {
4.   //cs2;
5. }

w(x,1) r(y,0) w(y,1) r(x,0)
Program Repair (based on Robustness)

**CEGAR:**
Counter-Example Guided
Abstraction Refinement

```
1. x=1;
2. a=y;
3. if (a==0) {
4.   //cs1;
5. }
```

```
1. y=1;
2. fence
3. b=x;
4. if (b==0) {
5.   //cs2;
6. }
```

Yes

Fences

Re-orderings

witness

witness (TSO)

repair

---

`1. y=1;
2. fence`
Program Repair (based on Robustness)

CEGAR:
Counter-Example Guided Abstraction Refinement

1. x=1;
2. a=y;
3. if (a==0) {
4. //cs1;
5. }
1. y=1;
2. fence
3. b=x;
4. if (b==0) {
5. //cs2;
6. }

robust?

Yes

witness (TSO)

witness analysis

repair

Fences

Re-orderings
Program Repair (based on Robustness)

CEGAR: Counter-Example Guided Abstraction Refinement

1. x=1;
2. a=y;
3. if (a==0) {
4.   //cs1;
5. }

1. y=1;
2. fence
3. b=x;
4. if (b==0) {
5.   //cs2;
6. }

w(x,1)
r(y,0)
w(y,1)
r(x,0)
w(x,1)

witness (TSO)

1. y=1;
2. fence
3. b=x;
4. if (b==0) {
5.   //cs2;
6. }

Fences
Re-orderings

repair

witness analysis

Yes
Program Repair (based on Robustness)

CEGAR:
Counter-Example Guided Abstraction Refinement

1. x=1;
2. a=y;
3. if (a==0) {
   4.  //cs1;
   5. }
4. y=1;
5. fence
6. b=x;
7. if (b==0) {
   8.  //cs2;
   9. }

Yes

robust?

Fences

reorderings

repair

witness (TSO)

w(x,1)
r(x,0)
r(y,0)
w(y,1)
w(x,0)
r(y,0)

witness analysis

1. y=1;
2. fence
3. b=x;
4. if (b==0) {
5.  //cs2;
6. }
Program Repair (based on Robustness)

CEGAR:
Counter-Example Guided
Abstraction Refinement

```
1. x=1;
2. a=y;
3. if (a==0) {
4.   //cs1;
5. }
```

```
1. y=1;
2. fence
3. b=x;
4. if (b==0) {
5.   //cs2;
6. }
```

---

repair

witness (TSO)

witness analysis

Fences

Re-orderings
Program Repair (based on Robustness)

CEGAR:
Counter-Example Guided Abstraction Refinement

1. x=1;
2. a=y;
3. if (a==0) {
4.  //cs1;
5. }

1. y=1;
2. fence
3. b=x;
4. if (b==0) {
5.  //cs2;
6. }

Yes

Fences

w(x,1)
r(y,0)
w(y,1)
r(x,0)
w(x,1)

witness analysis

robust?

repair

Fences

re-orderings
Program Repair (based on Robustness)

CEGAR:
Counter-Example Guided Abstraction Refinement

1. x=1;
2. a=y;
3. if (a==0) {
4.   //cs1;
5. }

1. y=1;
2. fence
3. b=x;
4. if (b==0) {
5.   //cs2;
6. }

1. x=1;
2. fence;
3. ry=y;
4. if (ry==0) {
5.   //cs1;

w(x,1)
w(y,1)
r(x,0)
r(y,0)

Program Repair (based on Robustness)
Program Repair (based on Robustness)

CEGAR:
Counter-Example Guided Abstraction Refinement

1. x=1;
2. a=y;
3. if (a==0) {
4.   //cs1;
5. }
1. y=1;
2. fence
3. b=x;
4. if (b==0) {
5.   //cs2;
6. }
1. y=1;
2. fence
3. ry=y;
4. if (ry==0) {
5.   //cs1;

r(y,0)  
w(y,1)  
r(x,0)  
w(x,1)
Program Repair (based on Robustness)

**CEGAR:**
Counter-Example Guided Abstraction Refinement

---

**1.** \( x=1; \)
**2.** \( a=y; \)
**3.** if \( (a==0) \) {
**4.** \( //cs1; \)
**5.** }

**1.** \( y=1; \)
**2.** fence
**3.** \( b=x; \)
**4.** if \( (b==0) \) {
**5.** \( //cs2; \)
**6.** }

---

**1.** \( x=1; \)
**2.** fence
**3.** \( ry=y; \)
**4.** if \( (ry==0) \) {
**5.** \( //cs1; \)

---

\[ \text{w(x,1)} \]
\[ \text{r(y,0)} \]
\[ \text{w(y,1)} \]
\[ \text{r(x,0)} \]
\[ \text{w(x,1)} \]
Program Repair (based on Robustness)

CEGAR:
Counter-Example Guided Abstraction Refinement

Yes

robust?

witness (TSO)

witness analysis

re-orderings

repair

Fences

1. x=1;
2. fence;
3. ry=y;
4. if (ry==0) {
5.   //cs1;
}  
1. y=1;
2. fence
3. b=x;
4. if (b==0) {
5.   //cs2;
6. }

r(y,0)
w(y,1)
r(x,0)
w(x,1)
Program Repair (based on Robustness)

CEGAR:
Counter-Example Guided Abstraction Refinement

1. x=1;
2. fence;
3. ry=y;
4. if (ry==0) {
   5. //cs1;
5. } //cs2;
6. }

robust?

No

Yes

witness analysis

w(x,1)
r(y,0)
w(y,1)
r(x,0)
w(x,1)

repair

Fences

Re-orderings
Program Repair (based on Robustness)

CEGAR:
Counter-Example Guided
Abstraction Refinement

1. x=1;
2. fence;
3. ry=y;
4. if (ry==0) {
   1. y=1;
   2. fence
   3. b=x;
   4. if (b==0) {
      5. //cs2;
      6. }
   5. //cs1;
}
Program Repair (based on Robustness)

CEGAR:
Counter-Example Guided Abstraction Refinement

1. x=1;
2. fence;
3. ry=y;
4. if (ry==0) {
   5. //cs1;
5. } //cs2;
6. }

Yes

witness (TSO)

witness analysis

repair

Fences

Re-orderings

r(y,0)
w(y,1)
r(x,0)
w(x,1)
Program Repair (based on Robustness)

CEGAR:
Counter-Example Guided
Abstraction Refinement

1. x=1;
2. fence;
3. ry=y;
4. if (ry==0) {
5.   //cs1;
6.   y=1;
7.   fence
8.   b=x;
9.   if (b==0) {
10.  //cs2;
11.  }

Yes

witness
(TSO)

Fences

Re-orderings

w(x,1)
r(y,0)
w(y,1)
r(x,0)
w(x,1)

witness analysis

repair
Outline

- Concurrent Programs
- Total Store Order (TSO)
- Correctness - Robustness
- Program Repair
- Perspectives
- Conclusions
Program Repair (based on Robustness)
Program Repair (based on Robustness)

Yes

robust?

P

witness analysis
witness (TSO)

Fences

re-orderings

repair
Program Repair (based on Robustness)

- witness analysis
- Fences
- Re-orderings
- repair

Yes

P
Program Repair (based on other criteria)

- Program Repair
  - Witness Analysis
  - Witness (TSO)
  - Re-orderings
  - Fences
  - Repair

Yes
Program Repair (based on other criteria)
Program Repair (based on other criteria)

- Yes
- Fences
- Re-orderings
- witness (TSO)
- witness analysis
- repair
Program Repair (based on other criteria)

- Yes
  - Fences
  - Re-orderings
  - Repair
- P
  - Assertion Violation
  - (TSO) Witness
  - Analysis
## Program Repair (based on other criteria)

<table>
<thead>
<tr>
<th>Data-Race Freedom</th>
<th>Robustness</th>
<th>Assertion Violation</th>
</tr>
</thead>
</table>

---
Program Repair (based on other criteria)

Data-Race Freedom \(\leq\) Robustness \(\leq\) Assertion Violation

efficiency \(\leq\) \# fences
Program Repair (based on other criteria)

Data-Race Freedom ≤ Robustness ≤ Assertion Violation

Assertion Violation

efficiency

# fences
Program Repair (based on other criteria)

\[
\text{Data-Race Freedom} \leq \text{Robustness} \leq \text{Assertion Violation}
\]

Assertion
Violation

efficiency

# fences
Program Repair (based on other criteria)

Data-Race Freedom \leq Robustness \leq Assertion Violation

less efficient

efficiency

Assertion Violation

# fences
Program Repair (based on other criteria)

Data-Race Freedom \(\leq\) Robustness \(\leq\) Assertion Violation

Assertion Violation

- less efficient
- efficiency
- fewer fences
- \# fences
Program Repair (based on other criteria)

Data-Race Freedom \(\leq\) Robustness \(\leq\) Assertion Violation

efficiency

# fences
Program Repair (based on other criteria)

- Data-Race Freedom
- Robustness
- Assertion Violation

- Robustness
- Data-Race Freedom

efficiency

# fences
Program Repair (based on other criteria)

- Data-Race Freedom
- Robustness
- Assertion Violation

Robustness

Data-Race Freedom

efficiency

# fences
Program Repair (based on other criteria)

- Data-Race Freedom
- Robustness
- Assertion Violation

Robustness

Data-Race Freedom

more efficient

efficiency

# fences
Program Repair (based on other criteria)

- Data-Race Freedom
- Robustness
- Assertion Violation

More efficient

Efficiency

More fences

# fences

Robustness

Data-Race Freedom
Program Repair

- Data-Race Freedom
- Robustness
- Assertion Violation

Flowchart:
- P → Assertion Violation
- Yes → Witness Analysis
- No → Fences, Re-orderings
- Repair
Program Repair

Yes

P

Assertion Violation

Fences

Re-orderings

witness (TSO)

witness analysis

repair

efficiency

# fences
Program Repair

Yes

Assertion Violation

witness (TSO)

witness analysis

Fences

Re-orderings

repair

exact methods

efficiency

# fences
Program Repair

Yes

Assertion Violation

witness (TSO)

witness analysis

Fences

Re-orderings

repair

exact methods

model checking

efficiency

# fences
Program Repair

Yes

Assertion Violation

witness (TSO)

witness analysis

repair

Fences

Re-orderings

exact methods

model checking

efficiency

# fences
Program Repair

Yes

Assertion Violation

witness (TSO)

witness analysis

repair

Fences

Re-orderings

exact methods

model checking

less efficient

efficiency

# fences
Program Repair

Yes

Assertion Violation

witness (TSO)

witness analysis

repair

exact methods

model checking

Fences

Re-orderings

less efficient

efficiency

optimal fencing

# fences
Program Repair

- Yes
- No

Assertion (TSO) Violation

witness analysis

exact set of needed fences
optimal fencing

# fences

exact methods
model checking

Fences
Re-orderings

repair

less efficient

efficiency

P
Program Repair

- Yes
- Fences
- Assertion Violation
- witness (TSO)
- Fences
- Re-orderings
- repair
- witness analysis

efficiency

# fences
Program Repair

Yes

Assertion Violation

witness (TSO)

Fences

Re-orderings

witness analysis

repair

under-approximations

efficiency

# fences
Program Repair

- Yes

- Assertion Violation
  - under-approximations
  - testing

- Fences
- Re-orderings

- witness (TSO)
- witness analysis

- repair

- efficiency
- # fences
Program Repair

Yes

Assertion Violation

witness (TSO)

witness analysis

Fences

Re-orderings

repair

under-approximations

stateless model checking

testing

efficiency

# fences
Program Repair

Yes

Assertion Violation

witness (TSO)

Fences

Re-orderings

Fences

Assert

witness

analysis

re-orderings

repair

under-approximations

testing

stateless model checking

bounded model checking

P

efficiency

# fences
Program Repair

- bounded context switching
- under-approximations
- stateless testing
- bounded model checking
- Fences
- Re-orderings
- witness (TSO)
- repair
- witness analysis

- Yes

efficiency

# fences
Program Repair

Assertion Violation

- bounded context switching
- under-approximations
- stateless model checking
- bounded model checking
- testing

Fences

Re-orderings

witness (TSO)

witness analysis

repair

Yes

efficiency

# fences
Program Repair

- bounded context switching
- under-approximations
- stateless model checking
- testing
- bounded model checking
- more efficient
- efficiency

Yes

Assertion Violation

witness analysis

Fences

Re-orderings

repair

witness (TSO)

# fences
Program Repair

- bounded context switching
- under-approximations
- stateless model checking
- more efficient
- efficiency
- bounded model checking
- testing
- Fences
- Re-orderings
- Assertion Violation
- witness (TSO)
- witness analysis
- repair
- Yes
- # fences
- under-fencing
Program Repair

Yes

Assertion (TSO)

Fences

Re-orderings

Witness analysis

repair

bounded context switching

under-approximations

stateless model checking

bounded model checking

more efficient

efficiency

fewer fences than needed

under-fencing

# fences
Program Repair

- Efficiency
- # Fences

- Yes
- Witness (TSO)

- Fences
- Re-orderings

Assertion Violation

Repair
Program Repair

- **P**
- **Assertion Violation**
- **Fences**
- **Witness (TSO)**
- **Witness Analysis**
- **Re-orderings**
- **Repair**
- **Yes**

- **Efficiency**
- **# Fences**

- **Over-approximations**
Program Repair

Yes

Assertion Violation

Fences

Over-approximations

Static analysis

Repair

Witness (TSO)

Witness analysis

Re-orderings

Efficiency

# Fences
Program Repair

Yes

Assertion Violation

witness (TSO)

witness analysis

Fences

over-approximations

abstraction

static analysis

re-orderings

repair

# fences

efficiency
Program Repair

Yes

Fences

Witness (TSO)

Witness analysis

Assertion Violation

Over-approximations

Static analysis

Abstraction

Repair

Efficiency

# Fences
Program Repair

Yes

Assertion Violation

witness (TSO)

witness analysis

Fences

Repair

over-approximations

static analysis

abstraction

more efficient

efficiency

# fences
Program Repair

Assertion Violation

Fences

re-orderings

witness

(TSO)

witness analysis

repair

over-approximations

abstraction

static analysis

more efficient

efficiency

over-fencing

# fences
Program Repair

Yes

Assertion Violation

witness (TSO)

witness analysis

P

re-orderings

Fences

over-approximations

abstraction

static analysis

repair

more fences than needed

over-fencing

# fences

more efficient

efficiency
Program P have identical behaviors under SC & TSO
Robustness

Program P have identical behaviors under SC & TSC
Program P

Robustness

P have identical behaviors under TSO & ARM
Robustness

Program P

P have \textbf{identical behaviors} under TSO & ARM
Robustness

Program P

ARM execution

P have identical behaviors under TSO & ARM
Robustness

P have identical behaviors under TSO & ARM
Program P

ARM executions

traces

Robustness

P have identical behaviors under SC & TSO

w(x,1) r(y,0) w(y,1) r(x,0)

r(x,0)
w(x,1)
r(y,0)
w(y,1)
Robustness

P have **identical behaviors** under SC & TSO

Program P

ARM executions

Init

w(x,1) → r(y,0) → w(y,1) → r(x,0)

Traces
Robustness

Program P

P have identical behaviors under SC & TSO

Init

r(x,0) → w(x,1) → r(y,0) → w(y,1)

ARM executions

traces
Program P

ARM executions

TSO executions

Robustness

P have identical behaviors under SC & TSO

w(x,0)
r(x,0)
w(y,0)
r(y,0)
w(x,1)
r(x,0)
w(y,1)
r(y,0)

w(x,1)
r(y,0)
w(y,1)
Program P

Robustness

P have **identical behaviors** under SC & TSO

**ARM executions**

**TSO executions**

Traces
Robustness

Program P

ARM executions

TSO executions

P have identical behaviors under SC & TSO
Robustness

Program P

P have identical behaviors under SC & TSO

ARM executions

TSO executions

Traces

Init
Tool Implementation

**Nidhugg**
- *Stateless Model Checking*
  - POWER
  - TSO
  - SC
  - C11
  - [https://github.com/nidhugg/nidhugg](https://github.com/nidhugg/nidhugg)

**Memorax**
- *Model Checking*
  - TSO
  - PSO
  - [https://github.com/memorax/memorax](https://github.com/memorax/memorax)

**Persist**
- *Robustness analysis (persistence)*
  - TSO
  - PSO
  - [https://github.com/PhongNgo/persist](https://github.com/PhongNgo/persist)
Persist (Robustness Analysis)

- Open-source: **Persist***

- Properties:
  - Using **Spin** as model-checker backend
  - Working for **finite-state** programs

*: [https://github.com/PhongNgo/persistence](https://github.com/PhongNgo/persistence)
<table>
<thead>
<tr>
<th>Program</th>
<th>#Process</th>
<th>Persist</th>
<th>Trencher</th>
<th>Musketeer</th>
<th>Memorax</th>
<th>Remmex</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>#Fence</td>
<td>#Time</td>
<td>#Fence</td>
<td>#Time</td>
<td>#Fence</td>
</tr>
<tr>
<td>SimDekker</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>163</td>
<td>8</td>
<td>-</td>
</tr>
<tr>
<td>Dekker</td>
<td>4</td>
<td>8</td>
<td>4</td>
<td>14</td>
<td>16</td>
<td>783</td>
</tr>
<tr>
<td>Peterson</td>
<td>4</td>
<td>4</td>
<td>223</td>
<td>-</td>
<td>-</td>
<td>OM</td>
</tr>
<tr>
<td>LamBak</td>
<td>3</td>
<td>6</td>
<td>104</td>
<td>-</td>
<td>-</td>
<td>OM</td>
</tr>
<tr>
<td>LamBak</td>
<td>4</td>
<td>8</td>
<td>286</td>
<td>-</td>
<td>-</td>
<td>OM</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>4</td>
<td>8</td>
<td>30</td>
<td>-</td>
<td>-</td>
<td>OM</td>
</tr>
<tr>
<td>DC-Lock</td>
<td>6</td>
<td>0</td>
<td>7</td>
<td>139</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>SpinLock</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TSpinLock</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>InlinePgsql</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>426</td>
<td>0</td>
<td>TO</td>
</tr>
<tr>
<td>Burns</td>
<td>5</td>
<td>9</td>
<td>119</td>
<td>-</td>
<td>-</td>
<td>OM</td>
</tr>
<tr>
<td>Szymaski</td>
<td>2</td>
<td>8</td>
<td>3</td>
<td>8</td>
<td>642</td>
<td>11</td>
</tr>
<tr>
<td>Szymaski</td>
<td>4</td>
<td>16</td>
<td>88</td>
<td>-</td>
<td>-</td>
<td>OM</td>
</tr>
<tr>
<td>LamFast2</td>
<td>2</td>
<td>8</td>
<td>6</td>
<td>48</td>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>LamFast2</td>
<td>3</td>
<td>12</td>
<td>10</td>
<td>-</td>
<td>-</td>
<td>OM</td>
</tr>
<tr>
<td>CLH Lock</td>
<td>4</td>
<td>4</td>
<td>176</td>
<td>4</td>
<td>674</td>
<td>0</td>
</tr>
<tr>
<td>Parker</td>
<td>6</td>
<td>1</td>
<td>76</td>
<td>-</td>
<td>425</td>
<td>3</td>
</tr>
<tr>
<td>Pgsql</td>
<td>6</td>
<td>7</td>
<td>26</td>
<td>-</td>
<td>260</td>
<td>7</td>
</tr>
<tr>
<td>AltenatingBit</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>IncSequence</td>
<td>6</td>
<td>0</td>
<td>21</td>
<td>-</td>
<td>194</td>
<td>0</td>
</tr>
<tr>
<td>TaskSchedule</td>
<td>10</td>
<td>0</td>
<td>21</td>
<td>-</td>
<td>418</td>
<td>0</td>
</tr>
<tr>
<td>NBW_W_WR</td>
<td>2</td>
<td>0</td>
<td>18</td>
<td>-</td>
<td>595</td>
<td>6</td>
</tr>
<tr>
<td>NBW_W_5R</td>
<td>6</td>
<td>0</td>
<td>3</td>
<td>-</td>
<td>514</td>
<td>0</td>
</tr>
<tr>
<td>SeqLock</td>
<td>4</td>
<td>0</td>
<td>63</td>
<td>-</td>
<td>364</td>
<td>0</td>
</tr>
<tr>
<td>write+r</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>r+deto</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>r+deto</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>write+r+co</td>
<td>6</td>
<td>0</td>
<td>7</td>
<td>4</td>
<td>38</td>
<td>4</td>
</tr>
<tr>
<td>sb+deto</td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>sb+deto+co</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Cilk WSQ</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>107</td>
<td>3</td>
</tr>
<tr>
<td>CL WSQ</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>796</td>
<td>1</td>
</tr>
<tr>
<td>FIFO iWSQ</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>354</td>
<td>1</td>
</tr>
<tr>
<td>LIFO iWSQ</td>
<td>2</td>
<td>1</td>
<td>7</td>
<td>1</td>
<td>558</td>
<td>1</td>
</tr>
<tr>
<td>Anchor iWSQ</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>20</td>
<td>1</td>
</tr>
</tbody>
</table>
## Result without Abstraction

<table>
<thead>
<tr>
<th>Program</th>
<th>#Process</th>
<th>Persist</th>
<th>#Fence</th>
<th>#Time</th>
<th>Trencher</th>
<th>#Fence</th>
<th>#Time</th>
<th>Musketeer</th>
<th>#Fence</th>
<th>#Time</th>
<th>Memorax</th>
<th>#Fence</th>
<th>#Time</th>
<th>Remmex</th>
<th>#Fence</th>
<th>#Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimDekker</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>163</td>
<td>8</td>
<td>1</td>
<td>783</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dekker</td>
<td>4</td>
<td>8</td>
<td>14</td>
<td>TO</td>
<td>166</td>
<td>110</td>
<td>TO</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Peterson</td>
<td>4</td>
<td>4</td>
<td>223</td>
<td>TO</td>
<td>18</td>
<td>110</td>
<td>TO</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>372</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LamBak</td>
<td>3</td>
<td>6</td>
<td>104</td>
<td>TO</td>
<td>8</td>
<td>123</td>
<td>6</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LamBak</td>
<td>4</td>
<td>8</td>
<td>286</td>
<td>TO</td>
<td>8</td>
<td>123</td>
<td>6</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dijkstra</td>
<td>4</td>
<td>8</td>
<td>30</td>
<td>TO</td>
<td>8</td>
<td>123</td>
<td>6</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DC-Lock</td>
<td>6</td>
<td>0</td>
<td>7</td>
<td>139</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SpinLock</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TSpinLock</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>OM</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InlinePgsql</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>426</td>
<td>0</td>
<td>1</td>
<td>OM</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Burns</td>
<td>5</td>
<td>9</td>
<td>119</td>
<td>TO</td>
<td>8</td>
<td>1</td>
<td>3</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>TO</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Szymaski</td>
<td>2</td>
<td>8</td>
<td>3</td>
<td>8</td>
<td>642</td>
<td>11</td>
<td>1</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Szymaski</td>
<td>4</td>
<td>16</td>
<td>88</td>
<td>TO</td>
<td>8</td>
<td>1</td>
<td>3</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LamFast2</td>
<td>2</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>48</td>
<td>15</td>
<td>1</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LamFast2</td>
<td>3</td>
<td>12</td>
<td>10</td>
<td>TO</td>
<td>8</td>
<td>1</td>
<td>3</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLH Lock</td>
<td>4</td>
<td>4</td>
<td>176</td>
<td>4</td>
<td>674</td>
<td>4</td>
<td>1</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parker</td>
<td>6</td>
<td>1</td>
<td>76</td>
<td>1</td>
<td>425</td>
<td>3</td>
<td>1</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pgsq1</td>
<td>6</td>
<td>7</td>
<td>26</td>
<td>7</td>
<td>260</td>
<td>1</td>
<td>4</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>TO</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AlternatingBit</td>
<td>6</td>
<td>2</td>
<td>21</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>TO</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IncSequence</td>
<td>6</td>
<td>2</td>
<td>21</td>
<td>0</td>
<td>194</td>
<td>0</td>
<td>1</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>TO</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TaskSchedule</td>
<td>10</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>418</td>
<td>0</td>
<td>1</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>TO</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NBW_W_WR</td>
<td>2</td>
<td>0</td>
<td>18</td>
<td>1</td>
<td>595</td>
<td>6</td>
<td>1</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NBW_W_5R</td>
<td>6</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>514</td>
<td>0</td>
<td>1</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SeqLock</td>
<td>4</td>
<td>0</td>
<td>63</td>
<td>0</td>
<td>364</td>
<td>0</td>
<td>1</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>write+r</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r+detour</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r+detours</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>write+r+co</td>
<td>6</td>
<td>0</td>
<td>7</td>
<td>4</td>
<td>38</td>
<td>4</td>
<td>1</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sb+detours</td>
<td>6</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sb+detour+co</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cilk WSQ</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>107</td>
<td>3</td>
<td>1</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CL WSQ</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>796</td>
<td>1</td>
<td>1</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIFO iWSQ</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>354</td>
<td>1</td>
<td>1</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIFO iWSQ</td>
<td>2</td>
<td>1</td>
<td>7</td>
<td>1</td>
<td>558</td>
<td>1</td>
<td>1</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Anchor iWSQ</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>20</td>
<td>1</td>
<td>1</td>
<td></td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program</td>
<td>#Fence</td>
<td>#Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>--------</td>
<td>-------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Anchor IWSQ</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIFO IWSQ</td>
<td>TO</td>
<td>TO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLK IWSQ</td>
<td>TO</td>
<td>TO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sb+detoors IWSQ</td>
<td>TO</td>
<td>TO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>write+r+co IWSQ</td>
<td>TO</td>
<td>TO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r+detoors IWSQ</td>
<td>TO</td>
<td>TO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AlternatingBit</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InSequence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TaskSchedule</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NBW-W-5R</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NBW-W-3</td>
<td>TO</td>
<td>TO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LamFast2</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LamFast2</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LamFast2</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Szymaski</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Burns</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SpinLock</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lamlock</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InlinePsql</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Peterson</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dekker</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SimDekker</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memorax</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remmex</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Time</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Fence</td>
<td>OM</td>
<td>OM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Result without Abstraction

<table>
<thead>
<tr>
<th>Program</th>
<th>#Process</th>
<th>Persist</th>
<th></th>
<th>Trencher</th>
<th></th>
<th>Musketeer</th>
<th></th>
<th>Memorax</th>
<th></th>
<th>Remmex</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>#Fence</td>
<td>#Time</td>
<td>#Fence</td>
<td>#Time</td>
<td>#Fence</td>
<td>#Time</td>
<td>#Fence</td>
<td>#Time</td>
<td>#Fence</td>
<td>#Time</td>
</tr>
<tr>
<td>SimDekker</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>163</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Dekker</td>
<td>4</td>
<td>8</td>
<td>14</td>
<td>6</td>
<td>104</td>
<td>18</td>
<td>110</td>
<td>6</td>
<td>372</td>
<td>6</td>
<td>372</td>
</tr>
<tr>
<td>Peterson</td>
<td>4</td>
<td>4</td>
<td>223</td>
<td>6</td>
<td>286</td>
<td>8</td>
<td>30</td>
<td>8</td>
<td>30</td>
<td>8</td>
<td>30</td>
</tr>
<tr>
<td>LamBak</td>
<td>3</td>
<td>6</td>
<td>104</td>
<td>6</td>
<td>139</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>LamBak</td>
<td>4</td>
<td>8</td>
<td>286</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>4</td>
<td>8</td>
<td>30</td>
<td>8</td>
<td>139</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Dc-Lock</td>
<td>6</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>SpinLock</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>223</td>
<td>8</td>
<td>223</td>
<td>8</td>
<td>223</td>
</tr>
<tr>
<td>SimDekker</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>163</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Dekker</td>
<td>4</td>
<td>8</td>
<td>14</td>
<td>6</td>
<td>104</td>
<td>18</td>
<td>110</td>
<td>6</td>
<td>372</td>
<td>6</td>
<td>372</td>
</tr>
<tr>
<td>Peterson</td>
<td>4</td>
<td>4</td>
<td>223</td>
<td>6</td>
<td>286</td>
<td>8</td>
<td>30</td>
<td>8</td>
<td>30</td>
<td>8</td>
<td>30</td>
</tr>
<tr>
<td>LamBak</td>
<td>3</td>
<td>6</td>
<td>104</td>
<td>6</td>
<td>139</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>LamBak</td>
<td>4</td>
<td>8</td>
<td>286</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>4</td>
<td>8</td>
<td>30</td>
<td>8</td>
<td>139</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Dc-Lock</td>
<td>6</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>SpinLock</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>223</td>
<td>8</td>
<td>223</td>
<td>8</td>
<td>223</td>
</tr>
<tr>
<td>SimDekker</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>163</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Dekker</td>
<td>4</td>
<td>8</td>
<td>14</td>
<td>6</td>
<td>104</td>
<td>18</td>
<td>110</td>
<td>6</td>
<td>372</td>
<td>6</td>
<td>372</td>
</tr>
<tr>
<td>Peterson</td>
<td>4</td>
<td>4</td>
<td>223</td>
<td>6</td>
<td>286</td>
<td>8</td>
<td>30</td>
<td>8</td>
<td>30</td>
<td>8</td>
<td>30</td>
</tr>
<tr>
<td>LamBak</td>
<td>3</td>
<td>6</td>
<td>104</td>
<td>6</td>
<td>139</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>LamBak</td>
<td>4</td>
<td>8</td>
<td>286</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
## Result without Abstraction

### Table: Comparison of Locking Mechanisms

<table>
<thead>
<tr>
<th>Program</th>
<th>#P</th>
<th>Persist</th>
<th>Trencher</th>
<th>Musketeer</th>
<th>Memorax</th>
<th>Remmex</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Fence</td>
<td>#Time</td>
<td>#Fence</td>
<td>#Time</td>
<td>#Fence</td>
<td>#Time</td>
</tr>
<tr>
<td>SimDekker</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>163</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>Dekker</td>
<td>4</td>
<td>8</td>
<td>14</td>
<td>16</td>
<td>783</td>
<td>1</td>
</tr>
<tr>
<td>Peterson</td>
<td>4</td>
<td>4</td>
<td>223</td>
<td>TO</td>
<td>OM</td>
<td>1</td>
</tr>
<tr>
<td>LamBak</td>
<td>3</td>
<td>6</td>
<td>104</td>
<td>TO</td>
<td>18</td>
<td>110</td>
</tr>
<tr>
<td>LamBak</td>
<td>4</td>
<td>8</td>
<td>286</td>
<td>TO</td>
<td>OM</td>
<td>1</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>4</td>
<td>8</td>
<td>30</td>
<td>TO</td>
<td>OM</td>
<td>1</td>
</tr>
<tr>
<td>Dc-Lock</td>
<td>6</td>
<td>0</td>
<td>7</td>
<td>139</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>SpinLock</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TSpinLock</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>InlinePgsql</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>426</td>
<td>1</td>
<td>OM</td>
</tr>
<tr>
<td>Burns</td>
<td>5</td>
<td>9</td>
<td>119</td>
<td>TO</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>Szymaski</td>
<td>2</td>
<td>8</td>
<td>3</td>
<td>642</td>
<td>11</td>
<td>1</td>
</tr>
</tbody>
</table>

### Notes:
- #P: Program Index
- Persist: M: Must, 0: Optional
- Trencher: M: Must, 0: Optional
- Musketeer: M: Must, 0: Optional
## Result without Abstraction

<table>
<thead>
<tr>
<th>Program</th>
<th>#P</th>
<th>Persist</th>
<th>Trencher</th>
<th>Musketeer</th>
<th>Memorax</th>
<th>Remmex</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>#Fence</td>
<td>#Time</td>
<td>#Fence</td>
<td>#Time</td>
<td>#Fence</td>
</tr>
<tr>
<td>SimDekker</td>
<td>4</td>
<td>8</td>
<td>14</td>
<td>TO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dekker</td>
<td>4</td>
<td>8</td>
<td>14</td>
<td>TO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Peterson</td>
<td>4</td>
<td>6</td>
<td>223</td>
<td>TO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LamBak</td>
<td>3</td>
<td>6</td>
<td>104</td>
<td>TO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LamBak</td>
<td>4</td>
<td>8</td>
<td>286</td>
<td>TO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dijkstra</td>
<td>4</td>
<td>8</td>
<td>30</td>
<td>TO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dc-Lock</td>
<td>6</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>139</td>
<td>0</td>
</tr>
<tr>
<td>SpinLock</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>TSpinLock</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>InlinePgsql</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>426</td>
<td></td>
</tr>
<tr>
<td>Burns</td>
<td>5</td>
<td>9</td>
<td>119</td>
<td>TO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Szymaski</td>
<td>2</td>
<td>8</td>
<td>3</td>
<td>8</td>
<td>642</td>
<td>11</td>
</tr>
</tbody>
</table>

- **Persist**: Number of fences.
- **Trencher**: Time to complete the test.
- **Musketeer**: Time to complete the test.
- **Memorax**: Time to complete the test.
- **Remmex**: Time to complete the test.
## Result without Abstraction

<table>
<thead>
<tr>
<th>Program</th>
<th>#P</th>
<th>Persist</th>
<th>Trencher</th>
<th>Musketeer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Fence</td>
<td>#Time</td>
<td>#Fence</td>
<td>#Time</td>
</tr>
<tr>
<td>SimDekker</td>
<td>4</td>
<td>4</td>
<td>163</td>
<td>8</td>
</tr>
<tr>
<td>Dekker</td>
<td>4</td>
<td>8</td>
<td>14</td>
<td>TO</td>
</tr>
<tr>
<td>Peterson</td>
<td>4</td>
<td>4</td>
<td>223</td>
<td>TO</td>
</tr>
<tr>
<td>LamBak</td>
<td>3</td>
<td>6</td>
<td>104</td>
<td>TO</td>
</tr>
<tr>
<td>LamBak</td>
<td>4</td>
<td>8</td>
<td>286</td>
<td>TO</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>4</td>
<td>8</td>
<td>30</td>
<td>TO</td>
</tr>
<tr>
<td>Dc-Lock</td>
<td>6</td>
<td>0</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>SpinLock</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>TSpinLock</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>InlinePgsql</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>Burns</td>
<td>5</td>
<td>9</td>
<td>119</td>
<td>TO</td>
</tr>
<tr>
<td>Szymaski</td>
<td>2</td>
<td>8</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

Program variations:
- r+detour
- r+detours
- write+r+co
- sb+detours
- sb+detour+co
- Cilk WSQ
- CL WSQ
- FIFO iWSQ
- LIFO iWSQ
- Anchor iWSQ
Result with Abstraction

Figure 1: Persist with/without abstraction compared with Trencher and Musketeer. The x axis is number of processes, the y axis is running time in seconds.
Figure 1: Persist with/without abstraction compared with Trencher and Musketeer. The x axis is number of processes, the y axis is running time in seconds.
Figure 1: Persist with/without abstraction compared with Trencher and Musketeer. 

The x axis is number of processes, the y axis is running time in seconds.
Outline

- Concurrent Programs
- Total Store Order (TSO)
- Correctness - Robustness
- Program Repair
- Perspectives
- Conclusions
Conclusions

- Weak memory models are here to stay
- New difficult challenges:
  - system design
  - program verification