Data races – easy explanation of Djit+
Simple version

- **LTF** (local time frame) counts local *release*(m)
- **Time vector**: local version of LTFs of other threads
- Per *release*(m):
  - LTF++
  - Send time vector to the next locker thread
- Per *lock*(m):
  - *Receive* time vector from the most recent releaser
  - Update local time vector as in vector clocks
- Per memory access – add vector time stamp
- Check HB condition with all previous accesses
### Example

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
<th>Thread 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 1 1)</td>
<td>(1 1 1)</td>
<td>(1 1 1)</td>
</tr>
<tr>
<td>Lock( m1 )</td>
<td>acquire( m1 )</td>
<td>write X</td>
</tr>
<tr>
<td>Write x</td>
<td>read X</td>
<td></td>
</tr>
<tr>
<td>release( m1 )</td>
<td>release( m2 )</td>
<td></td>
</tr>
<tr>
<td>(1 1 1)</td>
<td>(2 1 1)</td>
<td>(1,1,1)</td>
</tr>
</tbody>
</table>

#### History

<table>
<thead>
<tr>
<th>X: (1,1,1), ltf=1</th>
<th>X:(2,1,1) ltf=1</th>
<th>X:(1,1,1) ltf=1</th>
<th>X: (1,1,1), ltf=1</th>
<th>X:(2,1,1) ltf=1</th>
<th>X:(1,1,1) ltf=1</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
</tbody>
</table>
# Example 2

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
<th>Thread 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 1 1)</td>
<td>(1 1 1)</td>
<td>(1 1 1)</td>
</tr>
<tr>
<td>Lock( m1 )</td>
<td>(1 1 1)</td>
<td>write X</td>
</tr>
<tr>
<td>Write x</td>
<td>(2 1 1)</td>
<td>(1,1,1)</td>
</tr>
<tr>
<td>release( m1 )</td>
<td>acquire( m1 )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>read X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>release( m2 )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(2 1 1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(2 1 1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(2 2 1)</td>
<td></td>
</tr>
</tbody>
</table>

**History**

| X: (1,1,1), ltf=1 | X:(2,1,1) ltf=1 | X:(1,1,1) ltf=1 |
Check HB?

- Which accesses to check?
  - All accesses in all threads

- Condition?
  - We compare as in vector clocks
  - Reminder: 
    - \( u \) did not happen after \( v \):
      \[
      u \leq v \iff \forall i : u[i] \leq v[i]
      \]
    - \( u \) happened before \( v \):
      \[
      u < v \iff u \leq v \& u \neq v
      \]
Check HB

• Take **ANY** two accesses a and b
• For them to be data races –
  • at least one for write
• For them to be concurrent?
  • a !hb b AND b !hb a
a HB b?

• How do we know?
• If both accesses used lock?

• a.timestamp[a.thread] ? b.timestamp[a.thread]
a HB b?

- How do we know?
- If both accesses used lock?

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a: (1,1,1)</td>
<td>b: (2,1,1)</td>
</tr>
</tbody>
</table>

- a.timestamp[a.thread] ? b.timestamp[a.thread]

This was BEFORE release of lock

This was AFTER release of lock
a HB b?

- \( a.\text{timestamp}[a.\text{thread}] < b.\text{timestamp}[a.\text{thread}] \)
Putting it all together

\[ P(a,b) \triangleq ( a, b - \text{potential data race} ) \wedge \wedge ( a \text{!HB} b ) \wedge ( b \text{!HB} a) \]

\[ P(a,b) \triangleq ( a\.type = \text{write} \lor b\.type = \text{write} ) \wedge \wedge ( a\.ltf \geq b\.timestamp[a\.thread_id] ) \wedge ( b\.ltf \geq a\.timestamp[b\.thread_id] ) \]
Optimization 1

- If we KNOW b was LOGGED after a, what do we know?
  - b happened in physical time later than a
  - Hence b hb a is NOT possible
- => why to check b !hb a?

\[ P(a,b) \triangleq (a\.type = \text{write} \lor b\.type = \text{write}) \land \\
\land (a\.ltf \geq b\.timestamp[a\.thread_id]) \land \\
\land (b\.ltf \geq a\.timestamp[b\.thread_id]) \]

- => No need to log full vector clock!
Optimization 1

• If we know \( b \) was \textbf{LOGGED} after \( a \), what do we know?
  - \( b \) happened in physical time later than \( a \)
  - Hence, \( b \not\geq_h b a \) is \textbf{NOT} possible
• \( \Rightarrow \) why to check \( b \not\geq_h b a \)?

\[
P(a,b) \triangleq (a\.\text{type} = \text{write} \lor b\.\text{type} = \text{write}) \land \not\land (a\.ltf \geq b\.timestamp[a\.thread_id]) \land (b\.ltf \geq a\.timestamp[b\.thread_id])
\]

Works only if logging is sequentially consistent
Optimization 2

- Which accesses to log?
- Observation: if b and c are in the same thread and in the same LTF:
  - b \textbf{hb} a \Rightarrow c \textbf{hb} a
- => only first read and first write to X are logged in each LTF
Optimization 3

- Which accesses to compare the current access with? (recall – we started with all accesses)
- Observation:
- Observation: if b and c are in the same thread and \( b \text{ hb } c \)
- \( \Rightarrow c \text{ hb } a \Rightarrow b \text{ hb } a \)
- \( \Rightarrow \) only the most recent accesses should be compared with