Recapitulation:

Goal: Establish locality

In a minimal violation, only a single thread delays stores.

Tool: Happens-before through

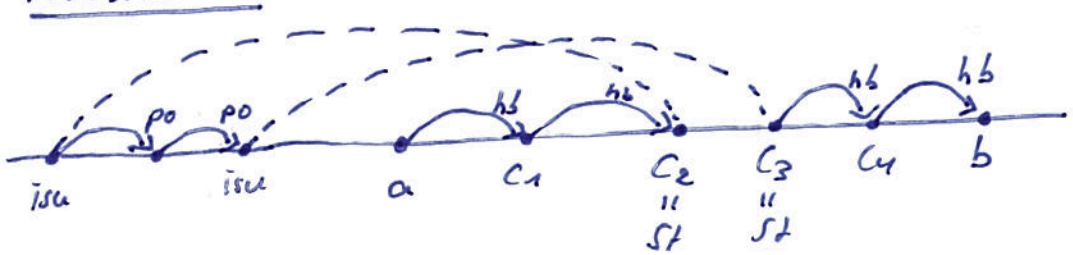
Consider $\bar{\tau}_1\, a\, \bar{\tau}\, b\, \bar{\tau}_2 \in C_{TSO}(P)$.

Then **a happens-before b through $\bar{\tau}$** if there is a subsequence $c_1, \dots, c_n$ of $\bar{\tau}$ so that

$$c_i \xrightarrow[\text{src, cf, st}]{} c_{i+1} \quad \text{or} \quad c_i \xrightarrow{+}_{po} c_{i+1}$$

for all $0 \le i \le n$ with $c_0 := a$ and $c_{n+1} := b$.

Illustration:



Proposition (Dichotomy):

Consider a minimal violation $\bar{\tau} = \bar{\tau}_1\, a\, \bar{\tau}_2\, b\, \bar{\tau}_3 \in C_{TSO}(P)$

Then

(1) $a \xrightarrow{+}_{hb} b$ through $\bar{\tau}_2$

or

(2) There is $\bar{\tau}' = \bar{\tau}_1\, \bar{\tau}_{21}\, b\, a\, \bar{\tau}_{22}\, \bar{\tau}_3 \in C_{TSO}(P)$

with $Tr(\bar{\tau}) = Tr(\bar{\tau}')$

and $\bar{\tau} \downarrow t = \bar{\tau}' \downarrow t$ for all $t \in TID$.

The proposition gives rise to the following principle for proving cycles:

"In a minimal violation, whenever a store is delayed past a load there is a happens-before cycle (that includes the two)."

Corollary:
Consider a minimal violation $\tau = \tau_1.\mathrm{isu}.\tau_2.\mathrm{ld}.\tau_3.\mathrm{st}.\tau_4 \in L_{tso}(P)$, where st is the store corresponding to isu.
Then $Tr(\tau)$ is cyclic.

Proof:
We show $\mathrm{st} \rightarrow^{+}_{po} \mathrm{ld} \rightarrow^{+}_{hs} \mathrm{st}$.

- The former dependence holds since isu is issued before ld.
- For $\mathrm{ld} \rightarrow^{+}_{hs} \mathrm{st}$ we argue that $\mathrm{ld} \rightarrow^{+}_{hs} \mathrm{st}$ through τ_3.

Indeed, with dichotomy we have
(1) $\mathrm{ld} \rightarrow^{+}_{hs} \mathrm{st}$ through τ_3
or (2) a reordering of ld and st that does not change the trace and the threads' computations.

If we reordered ld and st using (2), then we would change $\tau \downarrow t$ with $t = \mathrm{thread}(\mathrm{ld}) = \mathrm{thread}(\mathrm{st})$.
Hence, case (1) applies and gives hb-through. □

Theorem (Locality, Bouajjani, M., Mohlmann, ICALP'11):
In a minimal violation, only a single thread reorders its actions.

Proof:

Consider a minimal violation $\tau \in C_{TSO}(P)$.

Towards a contradiction,

suppose <u>at least two</u> threads delay stores.

- By lemma from last lecture,
  each store is delayed past a load of the same thread.
  Let st_2 be the overall last store
  that was delayed in τ.
  Let st_2 be from thread t_2.
  Let ld_2 be the overall last load
  that was overtaken by st_2.
  Similarly, let st_1 be the overall last store
  delayed in a thread $t_1 \neq t_2$.
  Let ld_1 be the last load overtaken by st_1.

- There are the following three situations:

(1) $\tau = \tau_1 . ld_1 . \tau_2 . st_1 . \tau_3 \, ld_2 . \tau_4 \, st_2 \, \tau_5 \in C_{TSO}(P)$

(2) $\tau = \tau_1 . ld_2 . \tau_2 \, ld_1 \, \tau_3 \, st_1 \, \tau_4 \, st_2 \, \tau_5 \in C_{TSO}(P)$

(3) $\tau = \tau_1 . ld_1 \, \tau_2 \, ld_2 \, \tau_3 \, st_1 \, \tau_4 \, st_2 \, \tau_5 \in C_{TSO}(P)$.

<u>Case (1):</u>

We argue that in this case τ is not minimal
and therefore the case does not apply.
Indeed, consider
$\tau' := \tau_1 . ld_1 . \tau_2 . st_2 . \tau_{5'} \in C_{TSO}(P)$.

Here, $\bar{\tau}_{st}$ contains stores of thread t_2 that were issued before st_1.

- Then:

$$\#(\bar{\tau}') < \#(\bar{\tau})$$ as the delay of st_2 over ld_2 is missing.

Moreover,

$Tr(\bar{\tau}')$ is cyclic.

The reason is that

$ld_1 \rightarrow^{+}_{hb} st_1$ through $\bar{\tau}_2$

continues to hold.

The only critical check here is

$c_i \rightarrow^{+}_{po} c_{i+1}$.

One can show that

- as long as $\bar{\tau}_2$ is not changed
- and the resulting computation $\bar{\tau}'$ is feasible ($\in C_{TSO}(P)$),

then

hb-through continues to hold.

Together, $\#(\bar{\tau}') < \#(\bar{\tau})$

and $Tr(\bar{\tau}')$ cyclic contradicts minimality of $\bar{\tau}$.

Case (2):

Again, the case would imply that $\bar{\tau}$ is not minimal and can therefore not occur.

- Starting from ld_2, thread t_2 does not do actions except delayed stores, until st_2 (this was a lemma last time).

Therefore, ld_2 and all program-order later actions of t_2 can be removed without affecting feasibility of the computation.

To be precise, we also have to remove $\bar{\tau}_5$, because other threads may load from stores of t_2 that are program-order later than ld_2:

$$\bar{\tau}' := \bar{\tau}_1 . \bar{\tau}_2 . ld_1 . \bar{\tau}_3 . st_1 . \bar{\tau}_4 . st_2 \in C_{TSO}(P).$$

- Then
$$\#(\bar{\tau}') < \#(\bar{\tau}).$$

Moreover,

$Tr(\bar{\tau}')$ is cyclic

due to $ld_1 \to_{hb}^{+} st_1$ through $\bar{\tau}_3$.

$\lightning$ minimality of $\bar{\tau}$.

Case (3):

Again we derive a contradiction to minimality of $\bar{\tau}$.

Remember

- First, we delete $\bar{\tau}_5$.
Then we delete all actions from $\bar{\tau}_4$ that do not belong to thread t_2:

$$\bar{\tau}' := \bar{\tau}_1 . ld_1 . \bar{\tau}_2 \; ld_2 \; \bar{\tau}_3 \; st_1 \; (\bar{\tau}_4 \downarrow t_2) . st_2 \in C_{TSO}(P).$$

As this does not change $\bar{\tau}_2 \; ld_2 \; \bar{\tau}_3$,
we still have

$ld_1 \to_{hb}^{*} st_1$ through $\bar{\tau}_2 \; ld_2 \; \bar{\tau}_3$

and $st_1 \to_{po}^{+} ld_2$.

So

$Tr(\bar{\tau}')$ is cyclic.

- Moreover, deleting actions does not introduce reorderings.
  Therefore,
  $$\#(\bar{\tau}') \leq \#(\bar{\tau}).$$
  Together, also $\bar{\tau}'$ is a minimal violation.

- We apply dichotomy again
  and get
  $$ld_2 \to_{hb}^{+} st_2 \quad \text{through } \bar{\tau}_3.st_2.(\tau_4 \downarrow t_2).$$
  Moreover,
  $$st_2 \to_{po}^{+} ld_2.$$

- Now, ld_2 is the program-order last action
  of thread t_2 in $\bar{\tau}'$.
  We delete it and get
  $$\bar{\tau}'' := \bar{\tau}_1\, \bar{\tau}_2\, ld_2\, \bar{\tau}_3\, st_2\, (\bar{\tau}_4 \downarrow t_2)\, st_2 \in G_{TIO}(P).$$
  Then
  $$\#(\bar{\tau}'') < \#(\bar{\tau}') \leq \#(\bar{\tau}).$$
  Moreover,
  $$Tr(\bar{\tau}'') \text{ is cyclic as the cycle}$$
  $$st_2 \to_{po}^{+} ld_2 \to_{hb}^{+} st_2 \text{ remains.}$$
  $\lightning$ minimality of $\bar{\tau}$. $\square$

10.3 Attacks on Robustness

Know: If robustness breaks,
then one thread delays actions.

Show: There are two instructions (a store and a load)
where a delay yields a cycle.