

Operational Memory Models: SC, TSO, SRA

September 14, 2026

1 SC

$$\begin{array}{c}
 \text{store} \frac{\sigma_i = (\text{store } \ell \ v; \sigma'_i)}{(\sigma_1 \parallel \dots \sigma_i \dots \parallel \sigma_n, m) \rightarrow (\sigma_1 \parallel \dots \sigma'_i \dots \parallel \sigma_n, m[\ell \mapsto v])} \\
 \\
 \text{load} \frac{\sigma_i = (r = \text{load } \ell; \sigma'_i)}{(\sigma_1 \parallel \dots \sigma_i \dots \parallel \sigma_n, m) \xrightarrow{Rr\ m(\ell)} (\sigma_1 \parallel \dots \sigma'_i \dots \parallel \sigma_n, m)}
 \end{array}$$

2 TSO

$$\begin{array}{c}
 \text{store} \frac{\sigma_i = (\text{store } \ell \ v; s, b) \quad \sigma'_i = (s, b; \ell \mapsto v)}{(\sigma_1 \parallel \dots \sigma_i \dots \parallel \sigma_n, m) \rightarrow (\sigma_1 \parallel \dots \sigma'_i \dots \parallel \sigma_n, m)} \\
 \\
 \text{propagate} \frac{\sigma_i = (s, ell \mapsto v; b) \quad \sigma'_i = (s, b)}{(\sigma_1 \parallel \dots \sigma_i \dots \parallel \sigma_n, m) \rightarrow (\sigma_1 \parallel \dots \sigma'_i \dots \parallel \sigma_n, m[\ell \mapsto v])} \\
 \\
 \text{load-mem} \frac{\sigma_i = (r = \text{load } \ell; s, b) \quad \ell \notin b \quad \sigma'_i = (s, b)}{(\sigma_1 \parallel \dots \sigma_i \dots \parallel \sigma_n, m) \xrightarrow{Rr\ m(\ell)} (\sigma_1 \parallel \dots \sigma'_i \dots \parallel \sigma_n, m)} \\
 \\
 \text{load-buf} \frac{\sigma_i = (r = \text{load } \ell; s, b) \quad \text{last } \ell \text{ in } b \text{ is } \ell \mapsto v \quad \sigma'_i = (s, b)}{(\sigma_1 \parallel \dots \sigma_i \dots \parallel \sigma_n, m) \xrightarrow{Rr\ v} (\sigma_1 \parallel \dots \sigma'_i \dots \parallel \sigma_n, m)} \\
 \\
 \text{fence} \frac{\sigma_i = (\text{fence}; s, \cdot) \quad \sigma'_i = (s, \cdot)}{(\sigma_1 \parallel \dots \sigma_i \dots \parallel \sigma_n, m) \rightarrow (\sigma_1 \parallel \dots \sigma'_i \dots \parallel \sigma_n, m)}
 \end{array}$$

Note that $\cdot$ means an empty list.

Key points:

1. The store rule adds writes to the *end* of the buffer, while the propagate rule removes writes from the *front* of the buffer. Together, these give the write buffers FIFO behavior.
2. The load-buf rule uses the *last* (most recent) write in the buffer.

3 SRA

$$\begin{array}{c}
\text{store} \frac{\sigma_i = (\text{store } \ell \ v; s, V) \quad V(\ell) < t \quad (\ell, t) \notin m \quad V' = V[\ell \mapsto t] \quad \sigma'_i = (s, V')}{(\sigma_1 \parallel \dots \sigma_i \dots \parallel \sigma_n, m) \rightarrow (\sigma_1 \parallel \dots \sigma'_i \dots \parallel \sigma_n, m \uplus (\ell, v, t, V'))} \\
\\
\text{load} \frac{\sigma_i = (r = \text{load } \ell; s, V) \quad V(\ell) \leq t \quad (\ell, v, t, V') \in m \quad \sigma'_i = (s, V \sqcup V')}{(\sigma_1 \parallel \dots \sigma_i \dots \parallel \sigma_n, m) \xrightarrow{R \ r \ v} (\sigma_1 \parallel \dots \sigma'_i \dots \parallel \sigma_n, m)}
\end{array}$$

Key points:

1. The store rule adds a write to the memory (“message pool”) m .
2. In the load rule, a thread can only read a write to ℓ that is at least as recent as the last write to ℓ it synchronized with ($V(\ell)$).
3. Each write in memory carries the view V of the thread that wrote it, and when a thread reads a write it adds that view to its own ($V \sqcup V'$), i.e., it synchronizes with the writing thread.