Weak memory consistency

Lecture 1

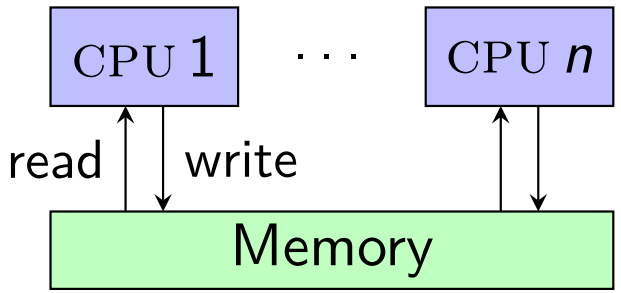
Viktor Vafeiadis
MPI-SWS

2022-11-14

The illusion of sequential consistency

Sequential consistency (SC)

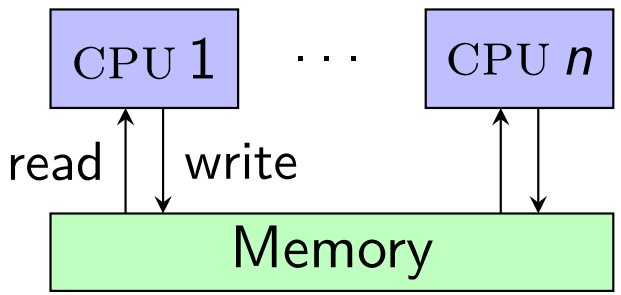
- The standard simplistic concurrency model.
- Threads access shared memory in an interleaved fashion.



The illusion of sequential consistency

Sequential consistency (SC)

- The standard simplistic concurrency model.
- Threads access shared memory in an interleaved fashion.



But. . .

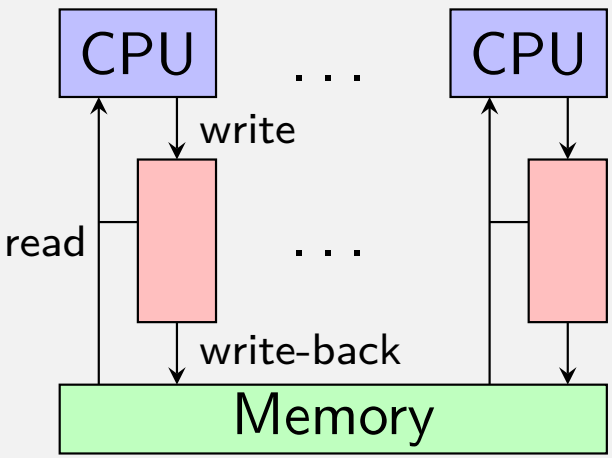
- No multicore processor implements SC.
- Compiler optimizations invalidate SC.

Weak consistency

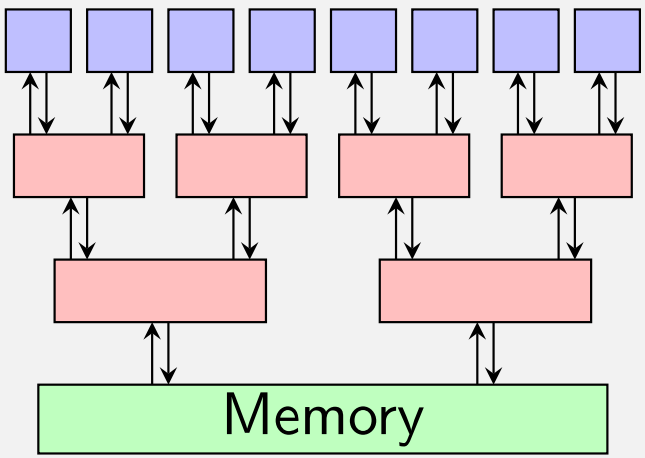
Hardware provides weak consistency.

- Weak memory models $\rightsquigarrow$ semantics of shared memory.
- Every hardware architecture has its own WMM: x86-TSO, ARM, Power, Itanium.

x86-TSO model (2010)



ARMv8 model (2016)

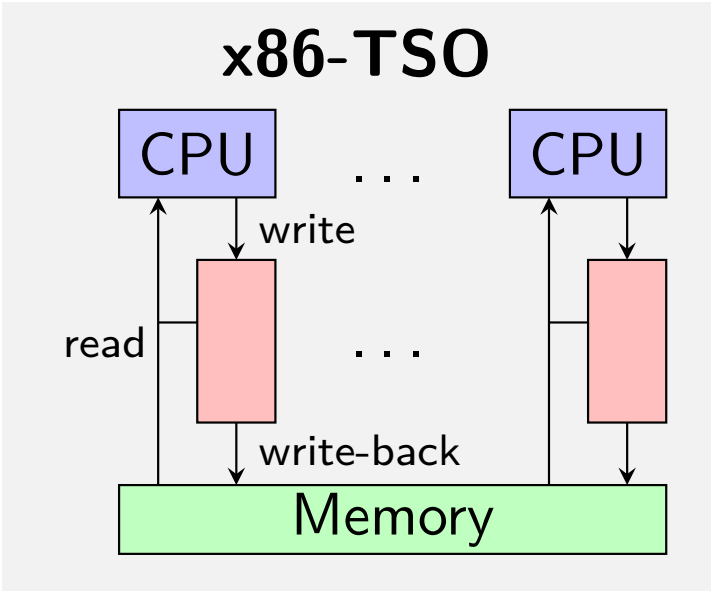


Weak consistency examples

Store buffering (SB)

Initially, $x = y = 0$

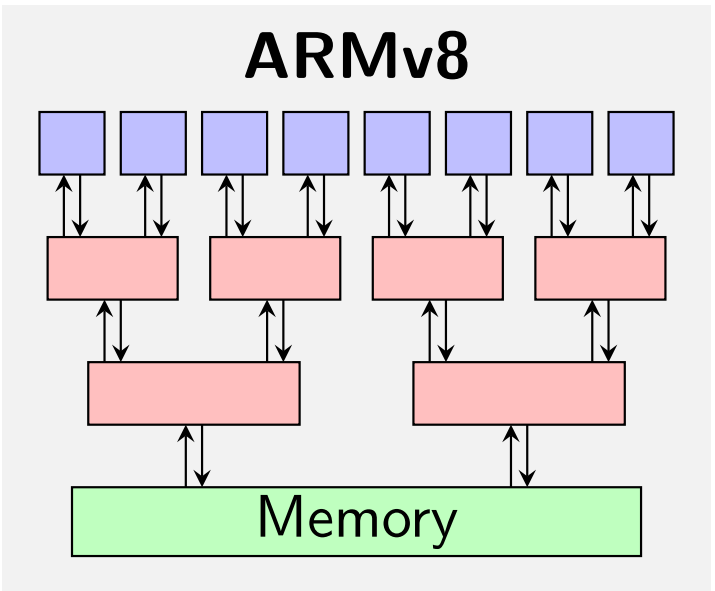
$$\begin{array}{l|l} x := 1; & y := 1; \\ a := y \quad /\!/ 0 & b := x \quad /\!/ 0 \end{array}$$



Load buffering (LB)

Initially, $x = y = 0$

$$\begin{array}{l|l} a := y; \quad /\!/ 1 & b := x; \quad /\!/ 1 \\ x := 1 & y := 1 \end{array}$$



Weak consistency in “real life”

- Messages may be delayed.

$$MsgX := 1; \quad a := MsgY; \quad //0 \;\Big\|\; MsgY := 1; \quad b := MsgX; \quad //0$$

- Messages may be sent/received out of order.

$$Email := 1; \quad Sms := 1; \;\Big\|\; a := Sms; \quad //1 \quad b := Email; \quad //0$$

Operational memory models

A simple concurrent programming language

Basic domains:

$r \in \mathrm{Reg}$ – Registers (local variables)

$x \in \mathrm{Loc}$ – Locations

$v \in \mathrm{Val}$ – Values including 0

$i \in \mathrm{Tid} = \{1, \ldots, N\}$ – Thread identifiers

Expressions and commands:

$$e ::= r \mid v \mid e + e \mid \ldots$$

$$c ::= \textbf{skip} \mid \textbf{if } e \textbf{ then } c \textbf{ else } c \mid \textbf{while } e \textbf{ do } c \mid$$
$$c\,;c \mid r := e \mid r := x \mid x := e \mid$$
$$r := \textbf{FAA}(x, e) \mid r := \textbf{CAS}(x, e, e) \mid \textbf{fence}$$

Programs, $P : \mathrm{Tid} \to \mathrm{Cmd}$, written as $P = c_1 \| \ldots \| c_N$

Basic set up

Thread subsystem

- Thread-local steps: $c, s \xrightarrow{I} c', s'$.
- Interpret sequential programs.
- Lift them to program steps: $P, S \xrightarrow{i:I} P', S'$.

Storage subsystem (defined by the memory model)

- Describe the effect of memory accesses and fences.
- $M \xrightarrow{i:I} M'$ where M is the state of the storage subsystem.

Linking the two

- Either the thread or the storage subsystem make an internal step, ε; or they make matching $i{:}I$ steps.
- $P, S, M \Rightarrow P', S', M'$.

The thread subsystem

Store: $s : \mathrm{Reg} \to \mathrm{Val}$ (Initial store: $s_0 \triangleq \lambda r.\ 0$)

State: $\langle c, s \rangle \in \mathrm{Command} \times \mathrm{Store}$

Transitions:

$$\frac{}{\textbf{skip}; c, s \xrightarrow{\varepsilon} c, s} \qquad \frac{c_1, s \xrightarrow{l} c_1', s'}{c_1; c_2, s \xrightarrow{l} c_1'; c_2, s'} \qquad \frac{s' = s[r \mapsto s(e)]}{r := e, s \xrightarrow{\varepsilon} \textbf{skip}, s'}$$

$$\frac{l = \mathtt{R}(x, v)}{r := x, s \xrightarrow{l} \textbf{skip}, s[r \mapsto v]} \qquad \frac{l = \mathtt{W}(x, s(e))}{x := e, s \xrightarrow{l} \textbf{skip}, s}$$

$$\frac{s(e) \neq 0}{\textbf{if } e \textbf{ then } c_1 \textbf{ else } c_2, s \xrightarrow{\varepsilon} c_1, s} \qquad \frac{s(e) = 0}{\textbf{if } e \textbf{ then } c_1 \textbf{ else } c_2, s \xrightarrow{\varepsilon} c_2, s}$$

$$\frac{}{\textbf{while } e \textbf{ do } c, s \xrightarrow{\varepsilon} \textbf{if } e \textbf{ then } (c; \textbf{while } e \textbf{ do } c) \textbf{ else skip}, s}$$

The thread subsystem: RMW and fence commands

Fetch-and-add:

$$\frac{l = \mathtt{U}(x, v, v + s(e))}{r := \mathbf{FAA}(x, e), s \xrightarrow{l} \mathbf{skip}, s[r \mapsto v]}$$

Compare-and-swap:

$$\frac{l = \mathtt{R}(x, v) \qquad v \neq s(e_r)}{r := \mathbf{CAS}(x, e_r, e_w), s \xrightarrow{l} \mathbf{skip}, s[r \mapsto 0]}$$

$$\frac{l = \mathtt{U}(x, s(e_r), s(e_w))}{r := \mathbf{CAS}(x, e_r, e_w), s \xrightarrow{l} \mathbf{skip}, s[r \mapsto 1]}$$

Fence:

$$\frac{}{\mathbf{fence}, s \xrightarrow{\mathtt{F}} \mathbf{skip}, s}$$

Lifting to concurrent programs

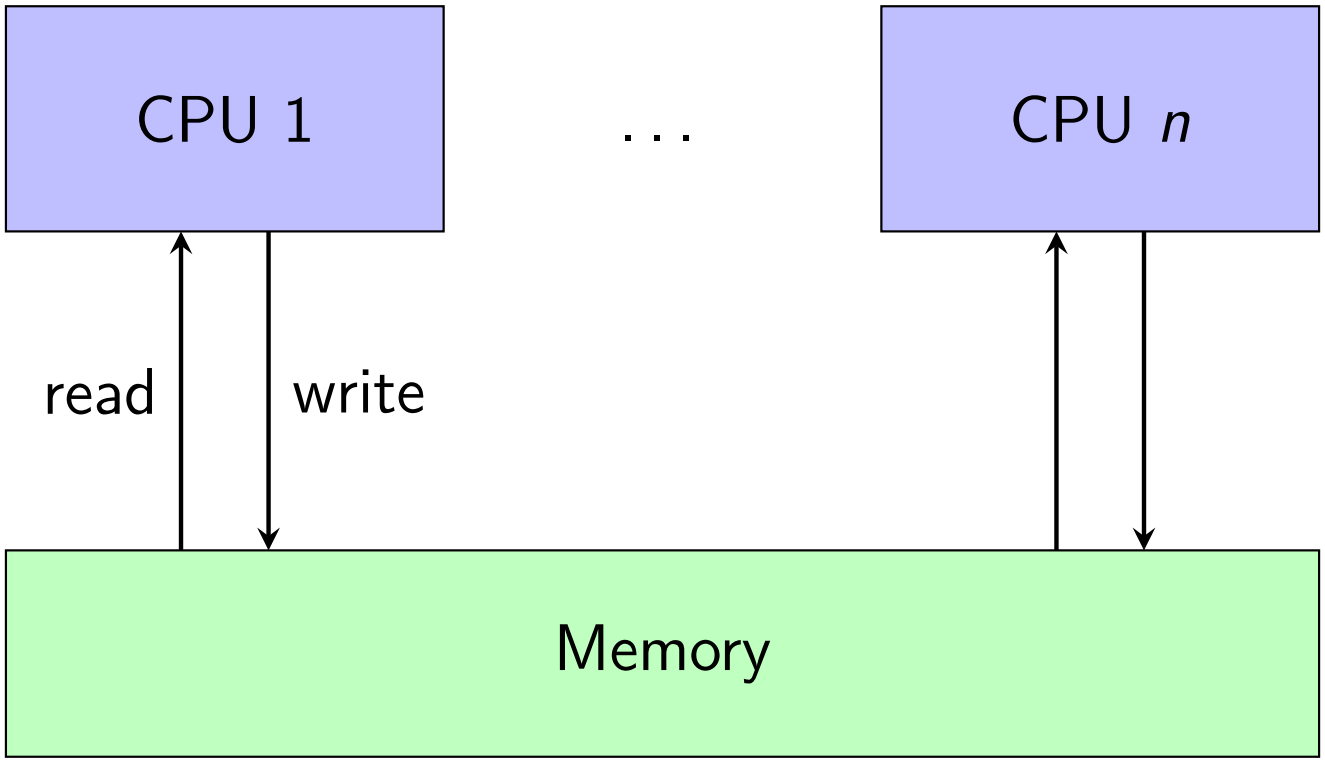
State: $\langle P, S \rangle \in \text{Program} \times (\text{Tid} \to \text{Store})$

- Initial stores: $S_0 \triangleq \lambda i.\ s_0$
- Initial state: $\langle P, S_0 \rangle$

Transition:

$$\frac{P(i), S(i) \xrightarrow{l} c, s}{P, S \xrightarrow{i:l} P[i \mapsto c], S[i \mapsto s]}$$

SC storage subsystem



SC storage subsystem

Machine state: $M : \mathrm{Loc} \to \mathrm{Val}$

- Maps each location to its value.
- Initial state: $M_0 \triangleq \lambda x.\ 0$
  (*i.e.*, the memory that maps every location to 0)

Transitions:

$$\frac{l = \mathtt{W}(x, v)}{M \xrightarrow{i:l} M[x \mapsto v]} \qquad \frac{l = \mathtt{R}(x, v) \qquad M(x) = v}{M \xrightarrow{i:l} M}$$

$$\frac{l = \mathtt{U}(x, v_r, v_w) \qquad M(x) = v_r}{M \xrightarrow{i:l} M[x \mapsto v_w]} \qquad \frac{l = \mathtt{F}}{M \xrightarrow{i:l} M}$$

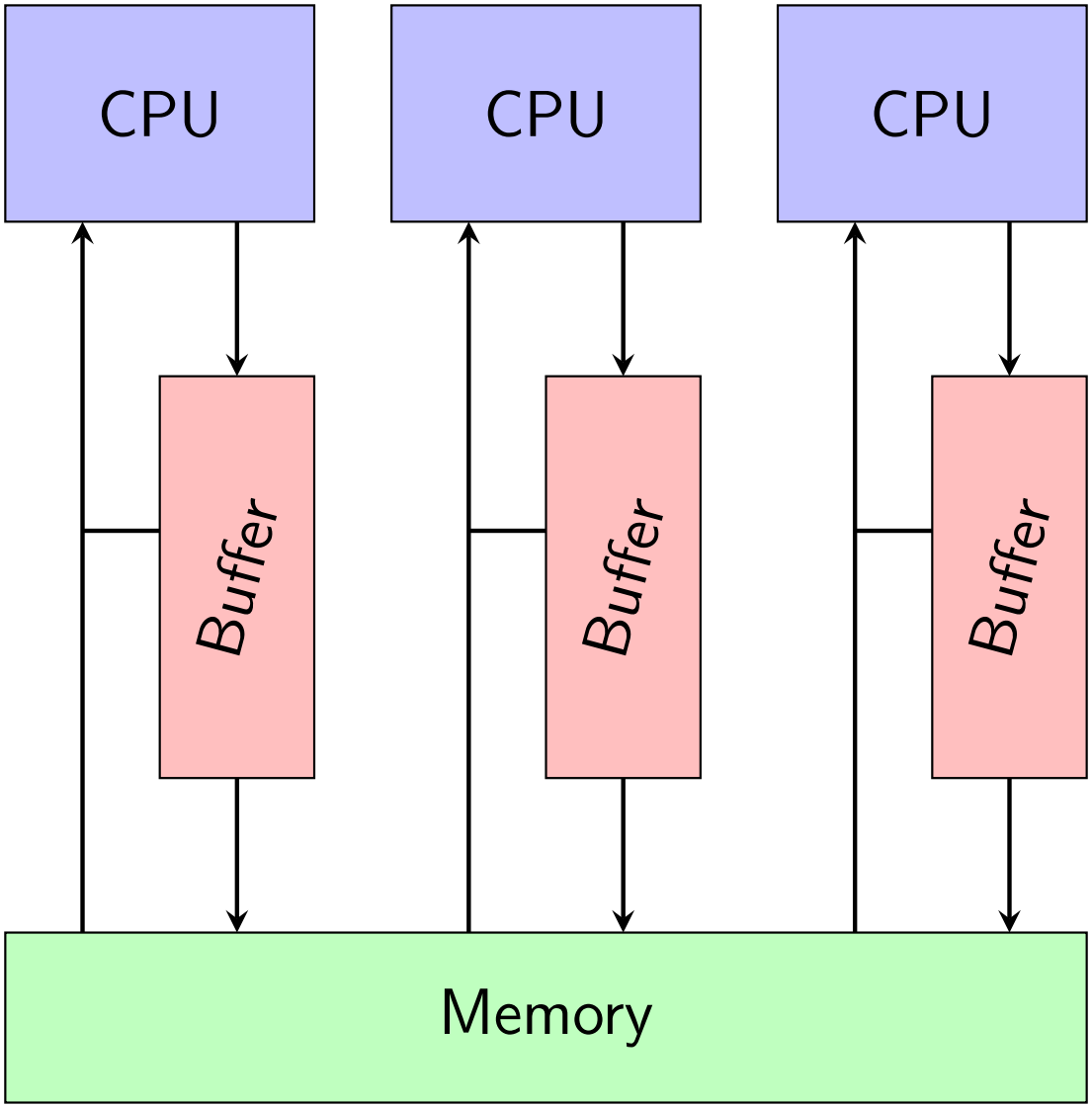
SC: Linking the thread and storage subsystems

$$\textsc{silent}\quad \frac{P, S \xrightarrow{i:\varepsilon} P', S'}{P, S, M \Rightarrow P', S', M} \qquad \textsc{non-silent}\quad \frac{P, S \xrightarrow{i:l} P', S' \qquad M \xrightarrow{i:l} M'}{P, S, M \Rightarrow P', S', M'}$$

Definition (Allowed outcome)

- An *outcome* is a function $O : \text{Tid} \to \text{Store}$.
- An outcome O is *allowed* for a program P under SC if there exists M such that $P, S_0, M_0 \Rightarrow^* \textbf{skip} \| \ldots \| \textbf{skip}, O, M$.

TSO storage subsystem



TSO storage subsystem

The state consists of:

- A memory $M : \mathrm{Loc} \to \mathrm{Val}$
- A function $B : \mathrm{Tid} \to (\mathrm{Loc} \times \mathrm{Val})^*$
  assigning a *store buffer* to every thread.

Initial state: $\langle M_0, B_0 \rangle$ **where**

- $M_0 = \lambda x.\, 0$ (the memory maps 0 to every location)
- $B_0 = \lambda i.\, \epsilon$ (all store buffers are empty)

TSO storage subsystem transitions

WRITE

$$\frac{l = \mathtt{W}(x, v)}{M, B \xrightarrow{i:l} M, B[i \mapsto \langle x, v \rangle \cdot B(i)]}$$

PROPAGATE

$$\frac{B(i) = b \cdot \langle x, v \rangle}{M, B \xrightarrow{i:\varepsilon} M[x \mapsto v], B[i \mapsto b]}$$

READ

$$\frac{\begin{array}{c} l = \mathtt{R}(x, v) \\ B(i) = \langle x_n, v_n \rangle \cdot \ldots \cdot \langle x_2, v_2 \rangle \cdot \langle x_1, v_1 \rangle \\ M[x_1 \mapsto v_1][x_2 \mapsto v_2] \ldots [x_n \mapsto v_n](x) = v \end{array}}{M, B \xrightarrow{i:l} M, B}$$

RMW

$$\frac{l = \mathtt{U}(x, v_r, v_w) \qquad B(i) = \epsilon \qquad M(x) = v_r}{M, B \xrightarrow{i:l} M[x \mapsto v_w], B}$$

FENCE

$$\frac{l = \mathtt{F} \qquad B(i) = \epsilon}{M, B \xrightarrow{i:l} M, B}$$

TSO: linking thread and storage subsystems

SILENT-THREAD

$$\frac{P, S \xrightarrow{i:\varepsilon} P', S'}{P, S, M, B \Rightarrow P', S', M, B}$$

SILENT-STORAGE

$$\frac{M, B \xrightarrow{i:\varepsilon} M', B'}{P, S, M, B \Rightarrow P, S, M', B'}$$

NON-SILENT

$$\frac{P, S \xrightarrow{i:l} P', S' \qquad M, B \xrightarrow{i:l} M', B'}{P, S, M, B \Rightarrow P', S', M', B'}$$

Definition (Allowed outcome)

An outcome O is *allowed* for a program P under TSO if there exists M such that $P, S_0, M_0, B_0 \Rightarrow^* \mathbf{skip} \| \ldots \| \mathbf{skip}, O, M, B_0$.