Introduction to weak memory consistency

Ori Lahav Viktor Vafeiadis

28 August 2017

Weak memory consistency

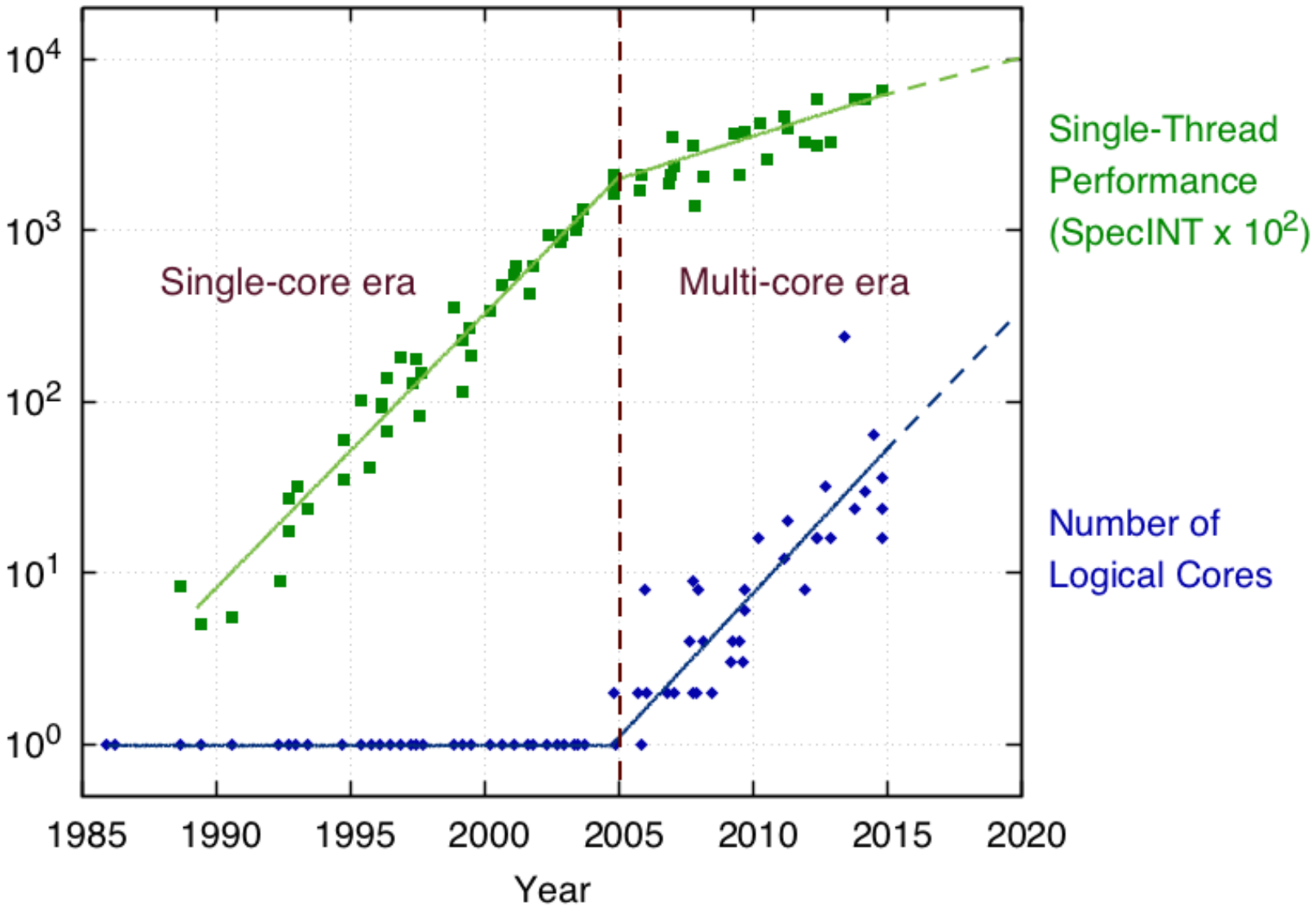
is about the

semantics of concurrent programs

taking into account the effects of:

- multicore hardware implementations
- and compiler optimizations.

CPU trends: Parallelism is here!



Concurrent programming is hard!

> *If you can get away with it, avoid using threads. Threads can be difficult to use, and they make programs harder to debug.*
>
> (Java documentation, $\approx$ 15 years ago)

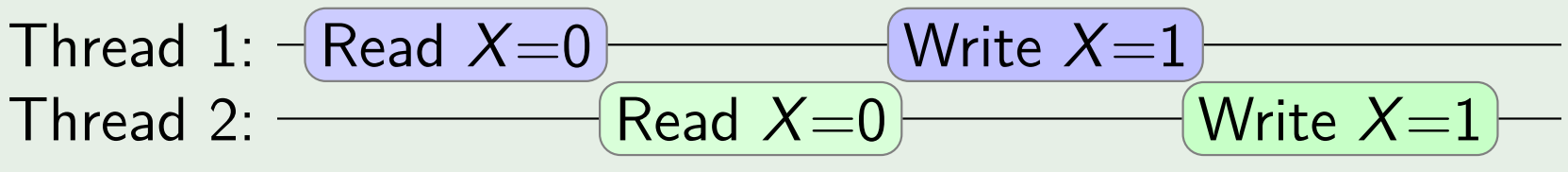
"Difficult to use"

- Requires a fundamentally different way of thinking.
- Interference among threads.

"Harder to debug"

- Huge non-determinism $\rightsquigarrow$ testing is ineffective.

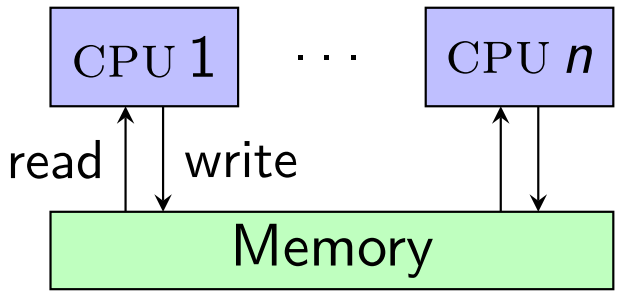
$X := X + 1 \parallel X := X + 1$ might increment X only once.



The illusion of sequential consistency

Sequential consistency (SC)

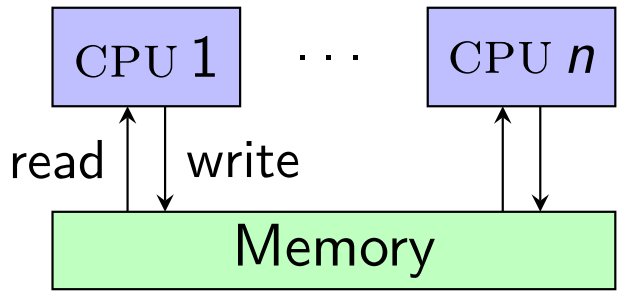
- The standard simplistic concurrency model.
- Threads access shared memory in an interleaved fashion.



The illusion of sequential consistency

Sequential consistency (SC)

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



But. . .

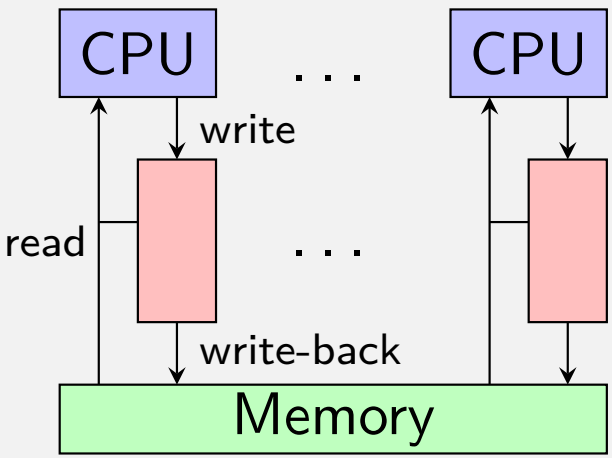
- No multicore processor implements SC.
- Compiler optimizations invalidate SC.

Weak consistency

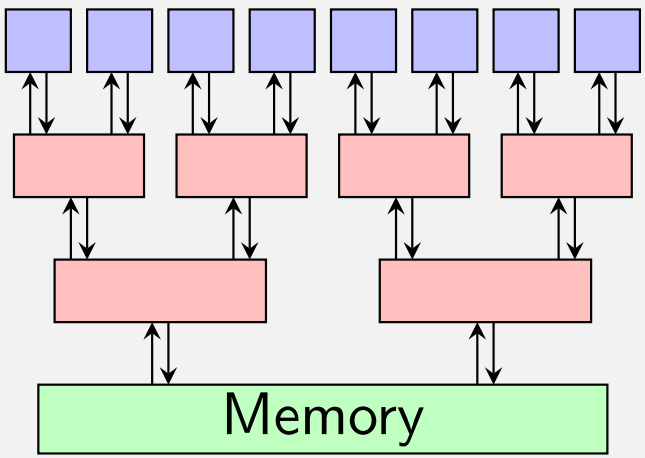
Hardware provides weak consistency.

- 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)

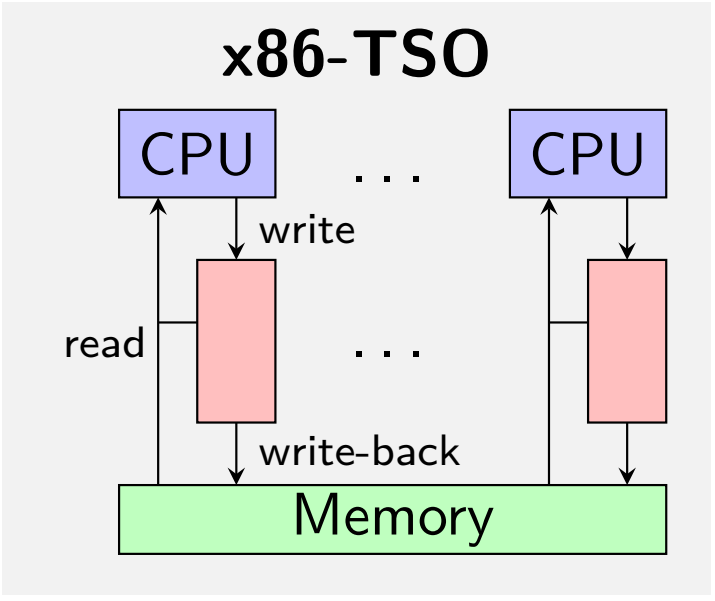


Weak consistency examples

Store buffering (SB)

Initially, $x = y = 0$

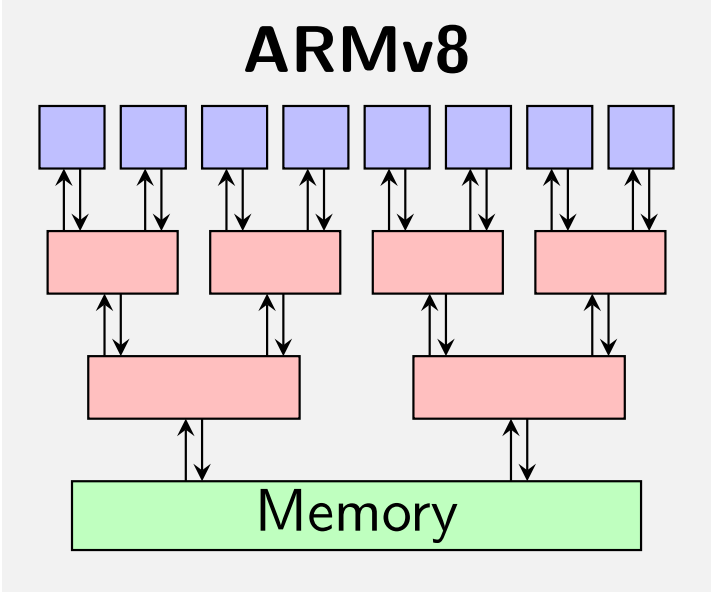
$$\begin{array}{l|l} x := 1; & y := 1; \\ a := y \quad /\!/ 0 & b := x \quad /\!/ 0 \end{array}$$



Load buffering (LB)

Initially, $x = y = 0$

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

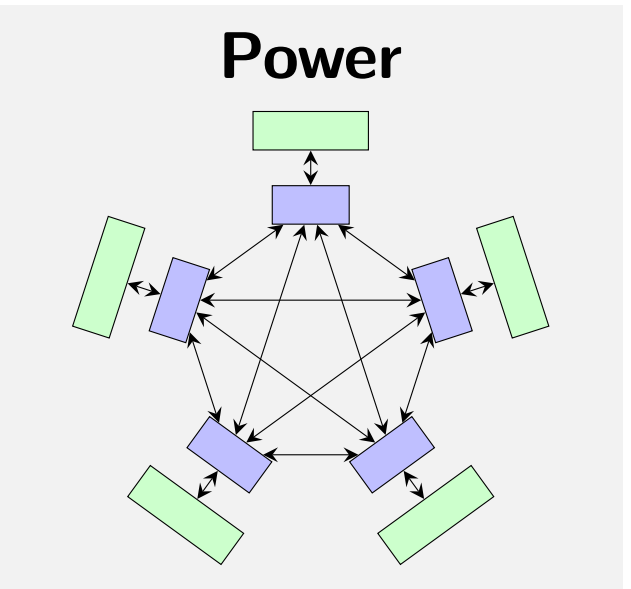


Independent reads of independent writes (IRIW)

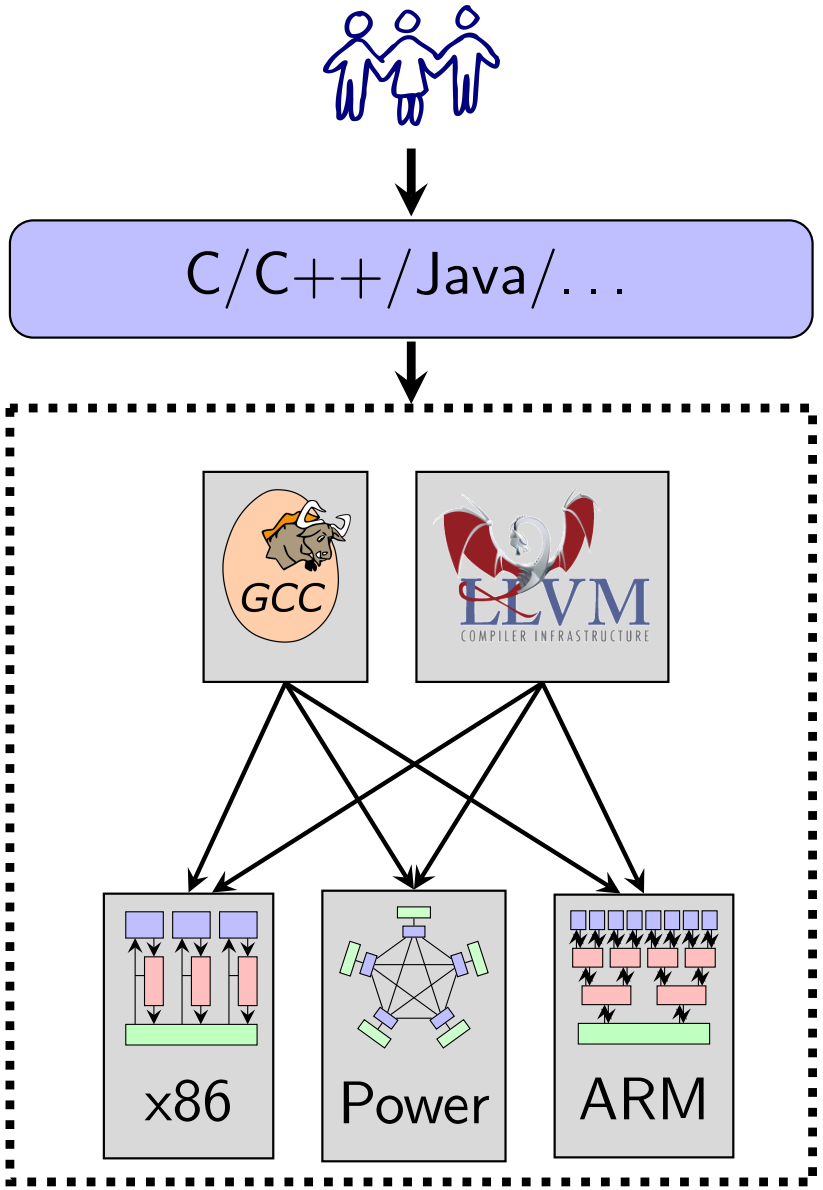
Initially, $x = y = 0$

$$x := 1 \;\Big\|\; \begin{array}{l} a := x; \quad /\!/ 1 \\ \text{lwsync}; \\ b := y \quad /\!/ 0 \end{array} \;\Big\|\; \begin{array}{l} c := y; \quad /\!/ 1 \\ \text{lwsync}; \\ d := x \quad /\!/ 0 \end{array} \;\Big\|\; y := 1$$

- Thread II and III can observe the $x := 1$ and $y := 1$ writes happen in different orders.
- Because of the lwsync fences, no reorderings are possible!



WMC is not just about hardware



Quiz. Should these transformations be allowed?

1. **CSE over acquiring a lock:**

$$\begin{array}{l} a = x; \\ lock(); \\ b = x; \end{array} \quad \rightsquigarrow \quad \begin{array}{l} a = x; \\ lock(); \\ b = a; \end{array}$$

2. **Load hoisting:**

$$\begin{array}{l} \text{if } (c) \\ \quad a = x; \end{array} \quad \rightsquigarrow \quad \begin{array}{l} t = x; \\ a = c \ ? \ t : a; \end{array}$$

[x is a global variable; a, b, c are local; t is a fresh temporary.]

Allowing both is clearly wrong! [CGO'16,CGO'17]

Consider the transformation sequence:

$$\begin{array}{l} \text{if } (c) \\ \quad a = x; \\ lock(); \\ b = x; \end{array} \quad \overset{\text{hoist}}{\rightsquigarrow} \quad \begin{array}{l} t = x; \\ a = c \ ? \ t : a; \\ lock(); \\ b = x; \end{array} \quad \overset{\text{CSE}}{\rightsquigarrow} \quad \begin{array}{l} t = x; \\ a = c \ ? \ t : a; \\ lock(); \\ b = t; \end{array}$$

When c is false, x is moved out of the critical region!

So we have to forbid one transfomation.

- C11 forbids load hoisting, allows CSE over lock().
- LLVM allows load hoisting, forbids CSE over lock().

Weak consistency in “real life”

- Messages may be delayed.

$$\begin{array}{l} MsgX := 1; \\ a := MsgY; \quad // 0 \end{array} \;\Big\|\; \begin{array}{l} MsgY := 1; \\ b := MsgX; \quad // 0 \end{array}$$

- Messages may be sent/received out of order.

$$\begin{array}{l} Email := 1; \\ Sms := 1; \end{array} \;\Big\|\; \begin{array}{l} a := Sms; \quad // 1 \\ b := Email; \quad // 0 \end{array}$$

Embracing weak consistency

Weak consistency is not a threat, but an opportunity.

- Can lead to more scalable concurrent algorithms.
- Several open research problems.
  - What is a good memory model?

Reasoning under WMC is often easier than under SC.

- Avoid thinking about thread interleavings.
- Many/most concurrent algorithms do not need SC!
- Positive *vs* negative knowledge.

Syllabus

Memory model definitions

- Operational memory models
- Axiomatic/declarative memory models
- Promising semantics

WMM metatheory

- Relating memory models
- Correctness of compilation and program transformations
- Programming guarantees: the DRF theorem

Verification techniques for WMM

- Program logics (relaxed separation logic, OGRA)
- Model checking