# Software verification under weak memory consistency

Viktor Vafeiadis

Max Planck Institute for Software Systems (MPI-SWS)

Dagstuhl, 2015-05-04

# Introduction

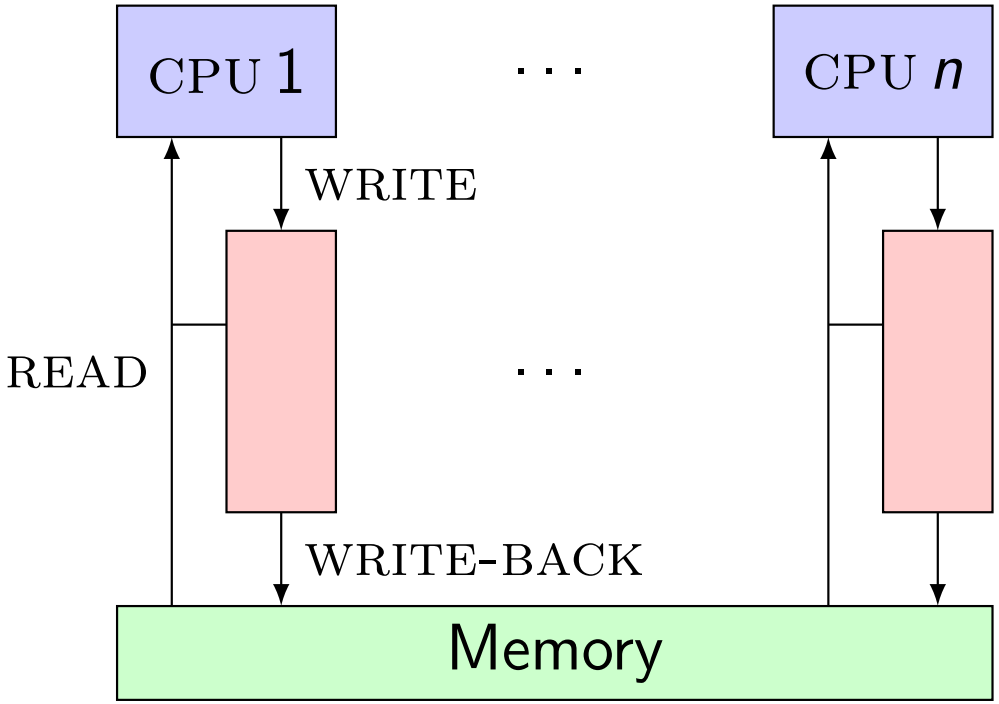
**Goal:**

- Understand & exploit weak memory consistency (using PLV techniques)

**Questions:**

- How do the different models relate?
- What program transformations are allowed? (Compiler optimisations)
- What proof techniques are sound? (Robustness theorems, program logics)
- Is there a good weak memory model?

# Store buffering in x86-TSO




Initially, $x = y = 0$.

$$\begin{array}{l} x := 1; \\ print(y); \end{array} \quad \Big\| \quad \begin{array}{l} y := 1; \\ print(x); \end{array}$$

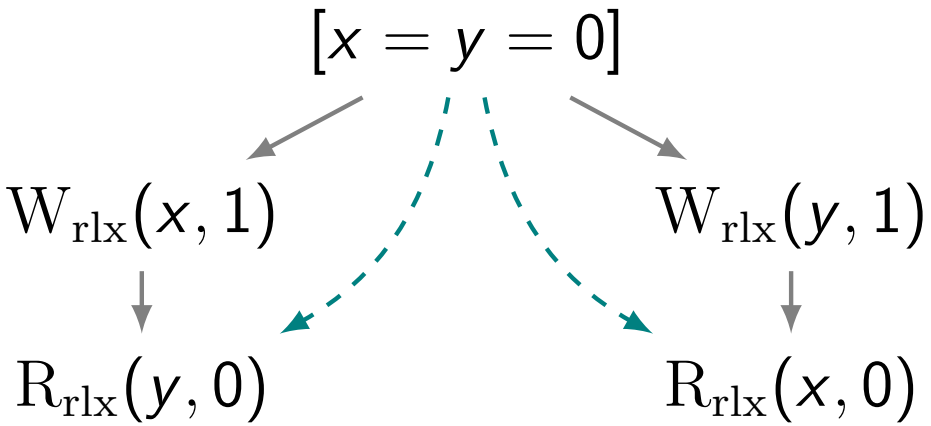
This program can also print 00.

# Store buffering in C11

Initially $x = y = 0$.

$$
\begin{array}{l|l}
x.\mathsf{store}(1, \mathit{rlx}); & y.\mathsf{store}(1, \mathit{rlx}); \\
\mathit{print}(y.\mathsf{load}(\mathit{rlx})); & \mathit{print}(x.\mathsf{load}(\mathit{rlx}));
\end{array}
$$

Can print 00 with the following execution:

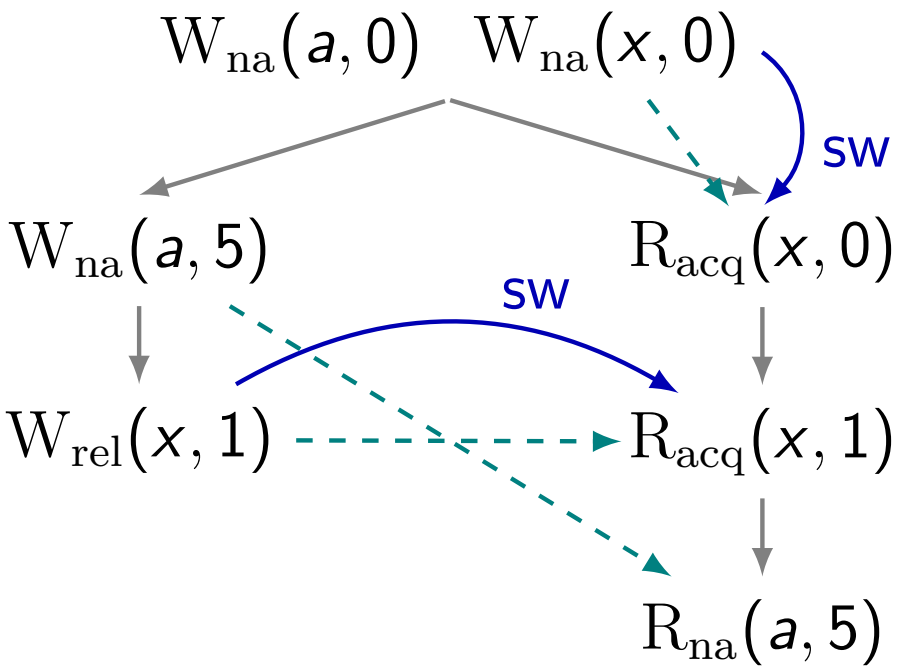


# Release-acquire synchronization

Initially $a = x = 0$.

| | | |
|---|---|---|
| $a = 5;$ | $\Vert$ | **while** $(x.\text{load}(acq) == 0);$ |
| $x.\text{store}(1, release);$ | $\Vert$ | $\text{print}(a);$ |

One possible execution:

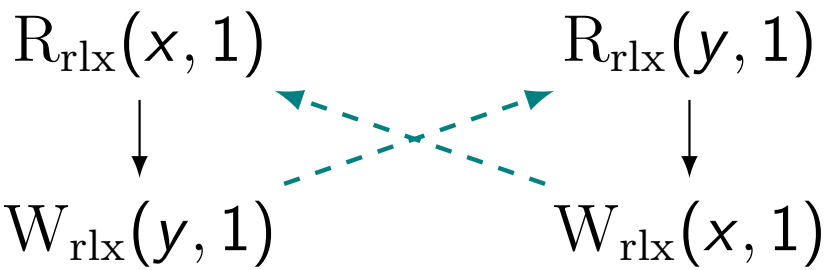


# Causality cycles

Initially $x = y = 0$.

$$\begin{array}{l} \textbf{if } (x.load(rlx) == 1) \\ \quad y.store(1, rlx); \end{array} \quad \Big\| \quad \begin{array}{l} \textbf{if } (y.load(rlx) == 1) \\ \quad x.store(1, rlx); \end{array}$$

C11 allows the outcome $x = y = 1$.




Relaxed accesses don't synchronize.

# Sequentialisation is invalid

Initially $a = x = y = 0$.

$$a = 1; \;\Big\|\; \textbf{if } (x.\text{load}(rlx) == 1) \;\; \textbf{if } (a == 1) \;\; y.\text{store}(1, rlx); \;\Big\|\; \textbf{if } (y.\text{load}(rlx) == 1) \;\; x.\text{store}(1, rlx);$$

**Remarks:**

- Non-atomic read axiom: $rf \cap (na \times na) \subseteq hb$.
- This program is not racy according to C11.
- The only possible output is $a = 1$ and $x = y = 0$.

**Consequences:**

- The model is not monotone.
- Sequentialization $C_1 \| C_2 \rightsquigarrow C_1; C_2$ is invalid.

# Data race freedom

**Theorem (DRF for a model $M$)**

*If $[\![C]\!]_{\mathsf{sf}}$ is data race free, then $[\![C]\!]_M = [\![C]\!]_{\mathsf{SC}}$.*

- Holds for all 'reasonable' memory models $M$.
- But not for C11 with RLX accesses due to causality cycles.

  Recall the example. Initially $x = y = 0$.

$$\begin{array}{l} \textbf{if } (x.load(rlx) == 1) \\ \quad y.store(1, rlx); \end{array} \;\Big\|\; \begin{array}{l} \textbf{if } (y.load(rlx) == 1) \\ \quad x.store(1, rlx); \end{array}$$

- Strengthen model for relaxed accesses...

# Program logics for WM

- Standard SC logics are unsound.
- What can be done.

# Owicki-Gries method (1976)

OG = Hoare logic + rule for parallel composition

$$\frac{\{P_1\}\, c_1\, \{Q_1\} \qquad \{P_2\}\, c_2\, \{Q_2\} \quad \text{the two proofs are } \textit{non-interfering}}{\{P_1 \wedge P_2\}\, c_1 \parallel c_2\, \{Q_1 \wedge Q_2\}}$$

## Non-interference

$R \wedge P \vdash R\{u/x\}$ for every:

- assertion $R$ in the proof outline of one thread
- assignment $x := u$ with precondition $P$ in the proof outline of the other thread

# Standard OG is unsound for WM

$$\{a \neq 0\}$$

$$x := 1; \quad \| \quad y := 1;$$

$$a := y \quad \| \quad b := x$$

$$\{a \neq 0 \vee b \neq 0\}$$

# Standard OG is unsound for WM

$$
\begin{array}{c}
\{a \neq 0\} \\
\begin{array}{l|l}
\{a \neq 0\} & \{\top\} \\
x := 1; & y := 1; \\
\{x \neq 0\} & \{y \neq 0\} \\
a := y & b := x \\
\{x \neq 0\} & \{y \neq 0 \wedge (a \neq 0 \vee b = x)\}
\end{array} \\
\{a \neq 0 \vee b \neq 0\}
\end{array}
$$

# Program logics for WM

**Program logics for C11 release-acquire:**

- Relaxed separation logic (V. & Narayan, OOPSLA'13)
- GPS (Turon et al., OOPSLA'14)
- OGRA (Lahav & V., ICALP'15)

**Program logics for TSO:**

- Rely/guarantee for TSO (Ridge, VSTTE'10)
- Verifying TSO programs (Jacobs, 2014)
- iCAP-TSO (Sieczkowski et al., ESOP'15)

# Relaxed separation logic (simplified)

Joint work with Chinmay Narayan, published at OOPSLA'13.

Ownership transfer by rel-acq synchronizations.

- Atomic allocation $\rightsquigarrow$ pick loc. invariant $\mathcal{Q}$.

$$\{\mathcal{Q}(v)\}\ x = \mathit{alloc}(v);\ \{\mathbf{W}_{\mathcal{Q}}(x) * \mathbf{R}_{\mathcal{Q}}(x)\}$$

- Release write $\rightsquigarrow$ give away permissions.

$$\{\mathcal{Q}(v) * \mathbf{W}_{\mathcal{Q}}(x)\}\ x.\mathit{store}(v, \mathit{release});\ \{\top\}$$

- Acquire read $\rightsquigarrow$ gain permissions.

$$\{\mathbf{R}_{\mathcal{Q}}(x)\}\ t = x.\mathit{load}(\mathit{acq});\ \{\mathcal{Q}(t)\}$$

# Message passing in RSL

Let $\mathcal{Q}(v) \triangleq v = 0 \vee \&a \mapsto 7$.

$$
\begin{array}{l}
\{\top\} \\
\textbf{atomic\_int}\ x = 0;\ \textbf{int}\ a = 0; \\
\{\&a \mapsto 0 * \mathbf{W}_{\mathcal{Q}}(x) * \mathbf{R}_{\mathcal{Q}}(x)\} \\
\left(\begin{array}{l|l}
\{\&a \mapsto 0 * \mathbf{W}_{\mathcal{Q}}(x)\} & t = x.\mathsf{load}(acq); \\
a = 7; & \{t = 0 \vee \&a \mapsto 7)\} \\
\{\&a \mapsto 7 * \mathbf{W}_{\mathcal{Q}}(x)\} & \textbf{if}\ (t \neq 0)\{\&a \mapsto 7\} \\
x.\mathsf{store}(1, release); & \quad \textbf{print}(a); \\
\{\top\} & \{\top\}
\end{array}\right) \\
\{\top\}
\end{array}
$$

# GPS: A better logic for release-acquire

Joint work with Aaron Turon and Derek Dreyer, at OOPSLA'14.

Three key features:

- Location protocols
- Ghost state/tokens 
- Escrows for ownership transfer

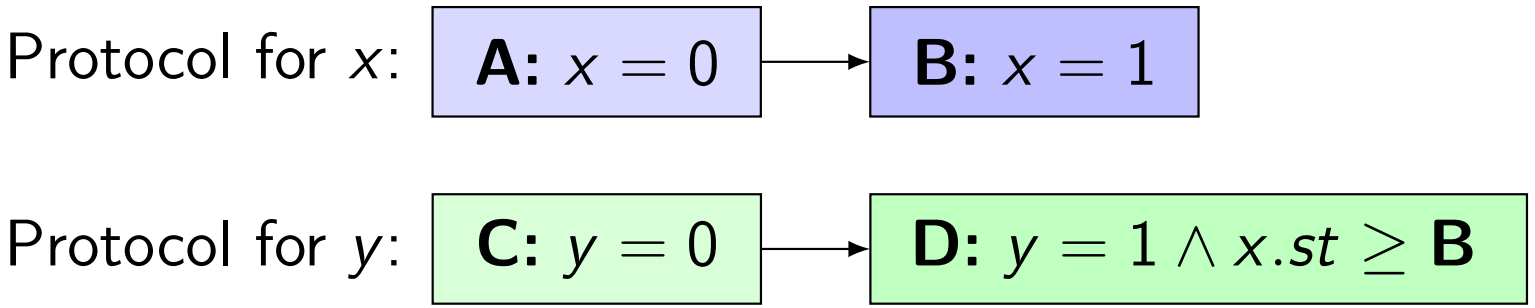
**Example (Racy message passing)**

Initially, $x = y = 0$.

$$\begin{array}{l|l|l} x.\mathsf{store}(1, \mathit{rlx}); & x.\mathsf{store}(1, \mathit{rlx}); & t = y.\mathsf{load}(\mathit{acq}); \\ y.\mathsf{store}(1, \mathit{rel}); & y.\mathsf{store}(1, \mathit{rel}); & t' = x.\mathsf{load}(\mathit{rlx}); \end{array}$$

Cannot get $t = 1 \wedge t' = 0$.

# Racy message passing in GPS




Acquire reads gain knowledge, not ownership.

$$
\begin{array}{l}
\{x.st \geq \mathbf{A} \wedge y.st \geq \mathbf{C}\} \\
x.\mathsf{store}(1, \mathit{rlx}); \\
\{x.st \geq \mathbf{B} \wedge y.st \geq \mathbf{C}\} \\
y.\mathsf{store}(1, \mathit{rel}); \\
\{x.st \geq \mathbf{B} \wedge y.st \geq \mathbf{D}\}
\end{array}
\quad \Big\| \quad
\begin{array}{l}
\{x.st \geq \mathbf{A} \wedge y.st \geq \mathbf{C}\} \\
t = y.\mathsf{load}(\mathit{acq}); \\
\left\{ \begin{array}{l} t = 0 \wedge x.st \geq \mathbf{A} \\ \vee\ t = 1 \wedge x.st \geq \mathbf{B} \end{array} \right\} \\
t' = x.\mathsf{load}(\mathit{rlx}); \\
\{t = 0 \vee (t = 1 \wedge t' = 1)\}
\end{array}
$$

# Some unresolved questions

- OOTA
- LLVM memory model
- MM classification

# Resolving out of thin air (OOTA) reads

Current status:

- C11 permits OOTA because of relaxed accesses.
- OOTA invalidates type safety & DRF.
- Hardware implementations don't generate OOTA.

Attempts at alternative models:

- Rule out RLX accesses and/or strengthen C11 axioms.
- Local transformations + RA.
- Operational
- Event structures

# The LLVM memory model

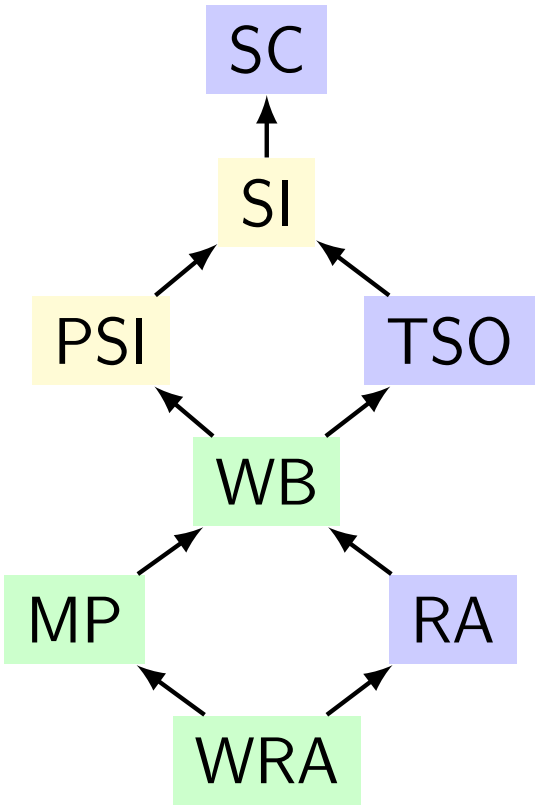
**Almost C11, but not exactly!**

- C11: non-atomic races $\rightsquigarrow$ program error.
- Irrelevant read introduction is unsound.
- LLVM, however, does introduce reads.
- LLVM: read-write race $\rightsquigarrow$ read gets bogus value.

**Example (Unsound sequence of transformations)**

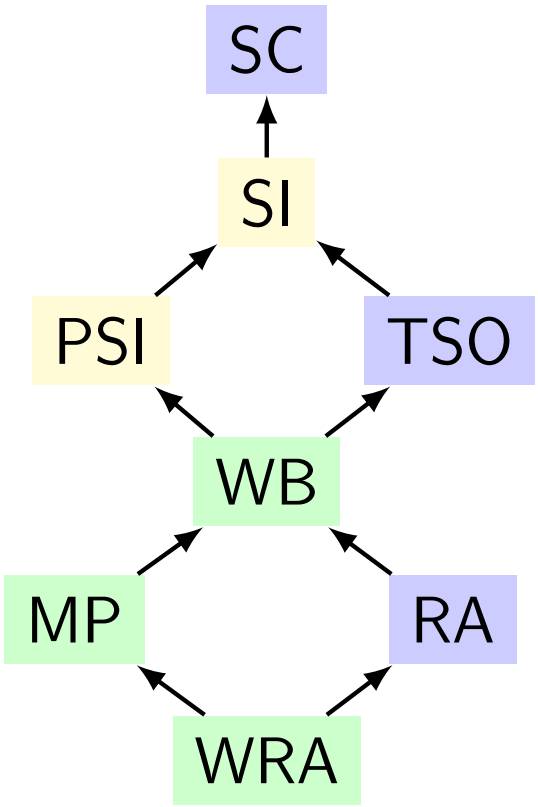
| | | | | |
|---|---|---|---|---|
| **if** (*flag*) | | | | |
| $a = X;$ | | $t = X;$ | | $t = X;$ |
| **else** | $\overset{(1)}{\rightsquigarrow}$ | $a = flag\ ?\ t : 0;$ | $\overset{(2)}{\rightsquigarrow}$ | $a = flag\ ?\ t : 0;$ |
| $a = 0;$ | | lock(); | | lock(); |
| lock(); | | $b = X;$ | | $b = t;$ |
| $b = X;$ | | | | |

# Hierarchy of relatively strong WMM




**Theorems & Conjectures:**

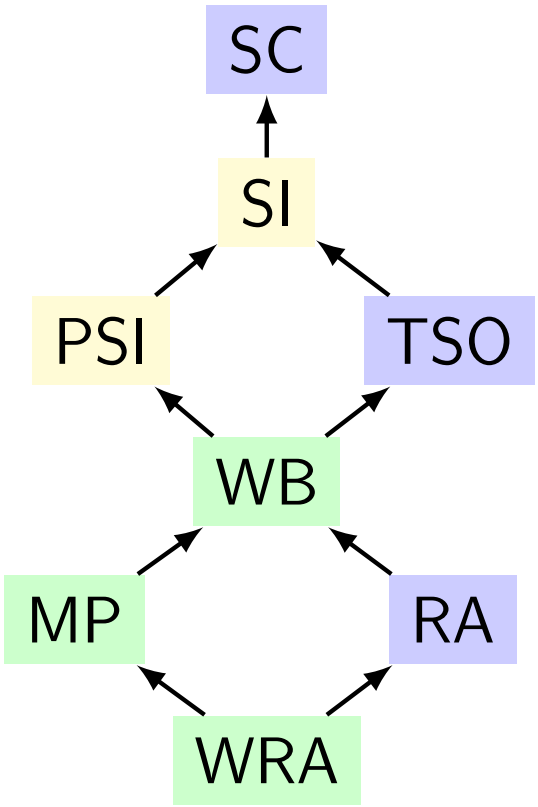
# Hierarchy of relatively strong WMM




**Theorems & Conjectures:**

- WRA = PSI if no WW races.

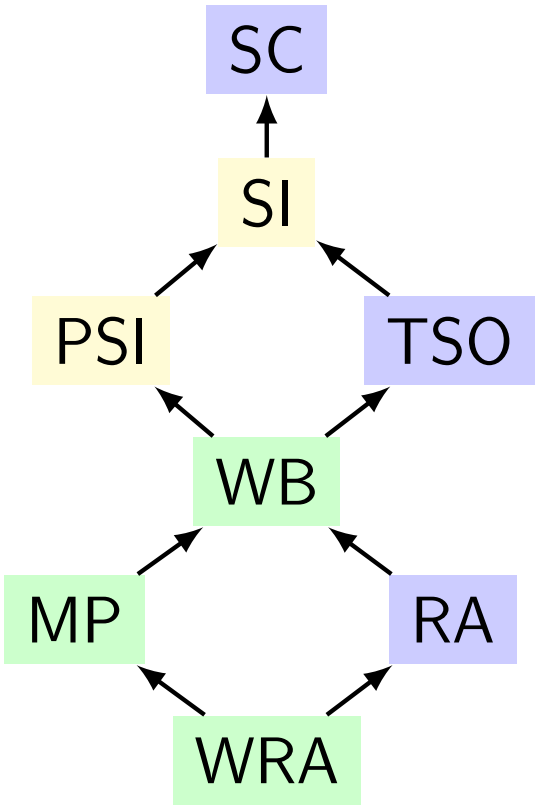
# Hierarchy of relatively strong WMM




**Theorems & Conjectures:**

- WRA = PSI if no WW races.
- TSO = WB and PSI = SI for server-client programs.

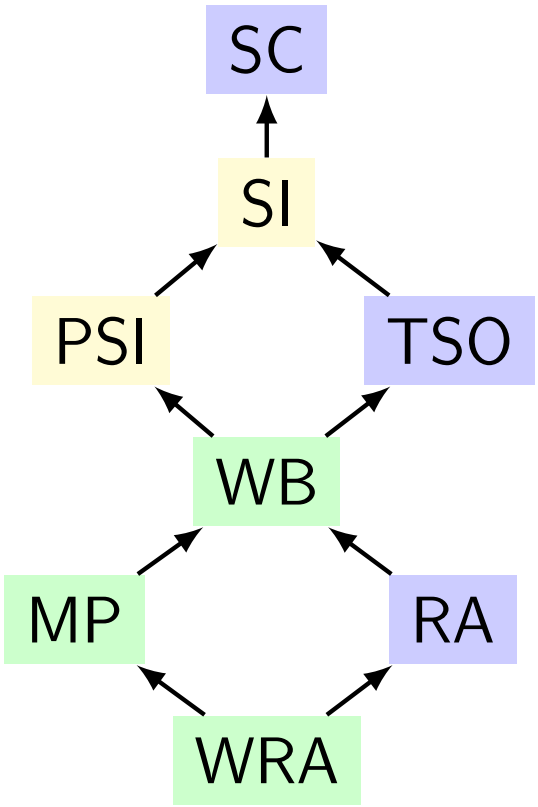
# Hierarchy of relatively strong WMM




**Theorems & Conjectures:**

- WRA = PSI if no WW races.
- TSO = WB and PSI = SI for server-client programs.
- Power-RA = WB.

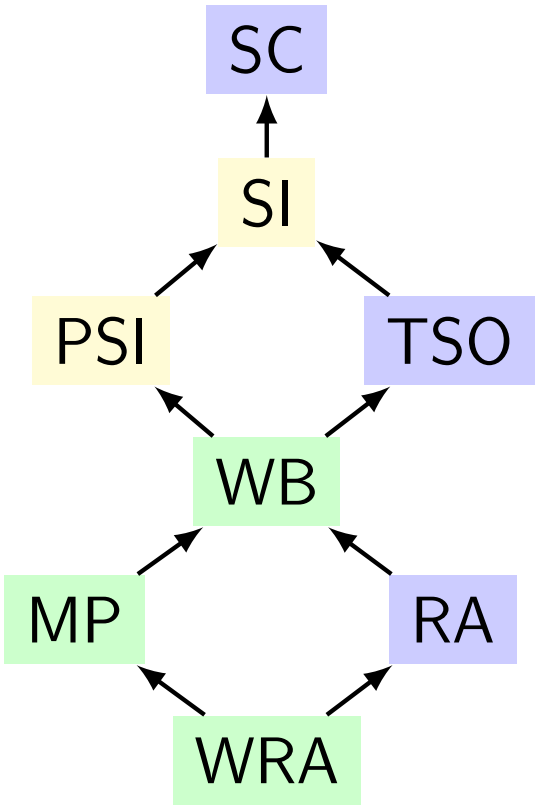
# Hierarchy of relatively strong WMM




**Theorems & Conjectures:**

- WRA = PSI if no WW races.
- TSO = WB and PSI = SI for server-client programs.
- Power-RA = WB.
- Operational model for MP?

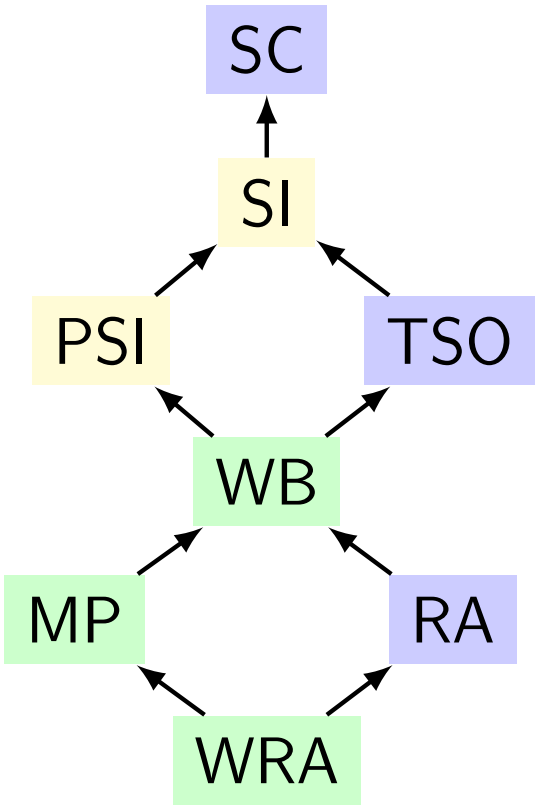
# Hierarchy of relatively strong WMM




**Theorems & Conjectures:**

- WRA = PSI if no WW races.
- TSO = WB and PSI = SI for server-client programs.
- Power-RA = WB.
- Operational model for MP?
- OGRA sound for RA, not for WRA.

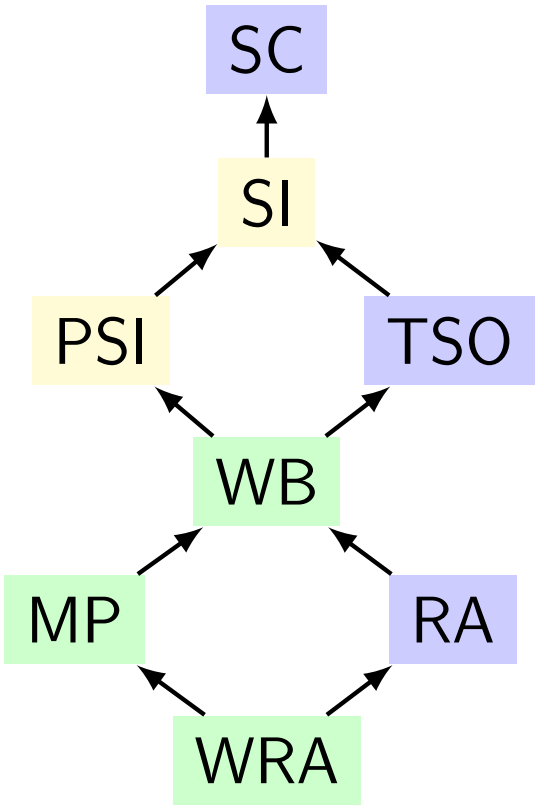
# Hierarchy of relatively strong WMM




**Theorems & Conjectures:**

- WRA = PSI if no WW races.
- TSO = WB and PSI = SI for server-client programs.
- Power-RA = WB.
- Operational model for MP?
- OGRA sound for RA, not for WRA.
- RSL and GPS sound for WRA?

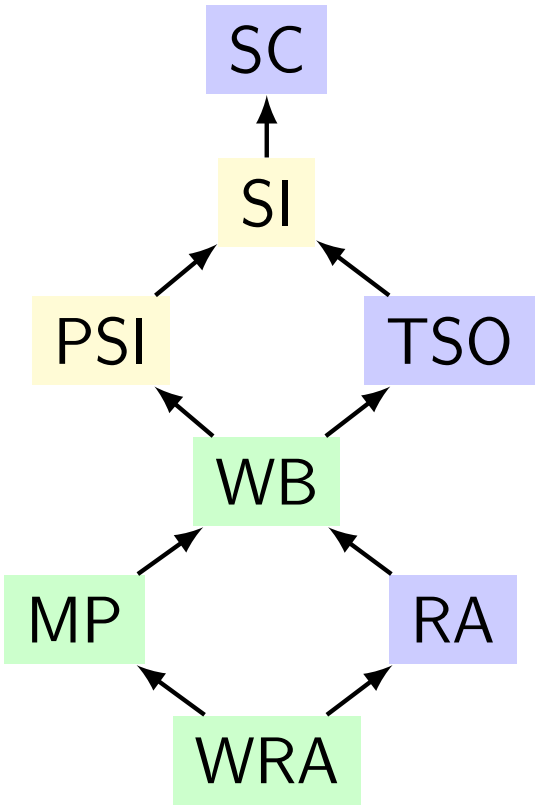
# Hierarchy of relatively strong WMM




**Theorems & Conjectures:**

- WRA = PSI if no WW races.
- TSO = WB and PSI = SI for server-client programs.
- Power-RA = WB.
- Operational model for MP?
- OGRA sound for RA, not for WRA.
- RSL and GPS sound for WRA?
- Restricted forms of auxiliary state sound for (W)RA and WB.

# Hierarchy of relatively strong WMM




**Theorems & Conjectures:**

- WRA = PSI if no WW races.
- TSO = WB and PSI = SI for server-client programs.
- Power-RA = WB.
- Operational model for MP?
- OGRA sound for RA, not for WRA.
- RSL and GPS sound for WRA?
- Restricted forms of auxiliary state sound for (W)RA and WB.

**Q:** Explain the MM differences with program logic?

# Conclusion

**Summary:**

- C11 is broken, but largely fixable.
- OG is unsound for TSO.

**Further work:**

- Resolving OOTA
- Formalizing the LLVM model
- MM hierarchy