The Semantics of Shared Variable Concurrency in Relaxed Memory Models

Colloque Anniversaire

en l'honneur de

Gérard Berry & Jean-Jacques Lévy

Claim.

☞ relaxed memory = a truly concurrent setting for program execution

Claim.

☞ relaxed memory = a truly concurrent setting for program execution

✌ true concurrency = Berry & Lévy's equivalence by permutation of transitions (POPL 1977)

Claim.

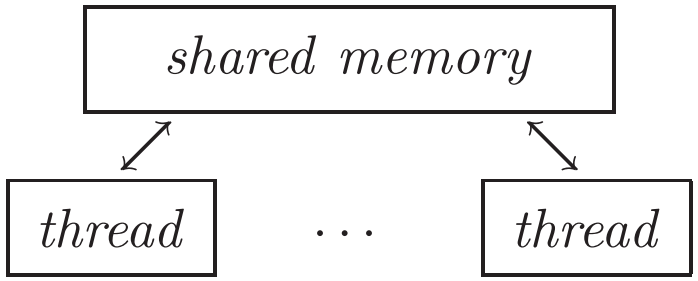
☞ relaxed memory = a truly concurrent setting for program execution

✌ true concurrency = Berry & Lévy's equivalence by permutation of transitions (POPL 1977)

(I came to know about that at the weekly Thursday Seminar organized by Maurice Nivat, who managed to attract GG & JJ – and some others – out of the pure light of their Bat 8 Sanctuary, down to some obscure room at Paris 7 University – thanks Maurice.)

Shared Memory Concurrency

- sequential programs (threads) running in parallel, sharing a global memory



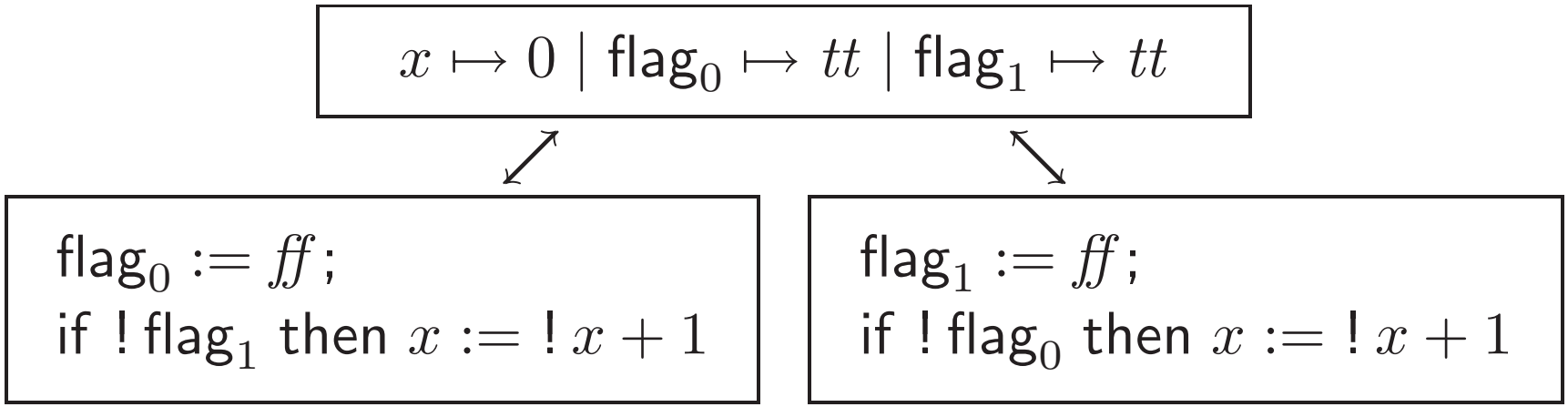
- semantics: small step transitions, interleaved, i.e. one thread at a time, non deterministically chosen, for one step

  Hennessy & Plotkin 1979
  precursors: Bekič 1971, Milner 1973, Cadiou & Lévy 1973

- aka sequential consistency (SC), Lamport 1979

Synchronization (1/2)

[Basic idea for] mutual exclusion, in a ML-like syntax:

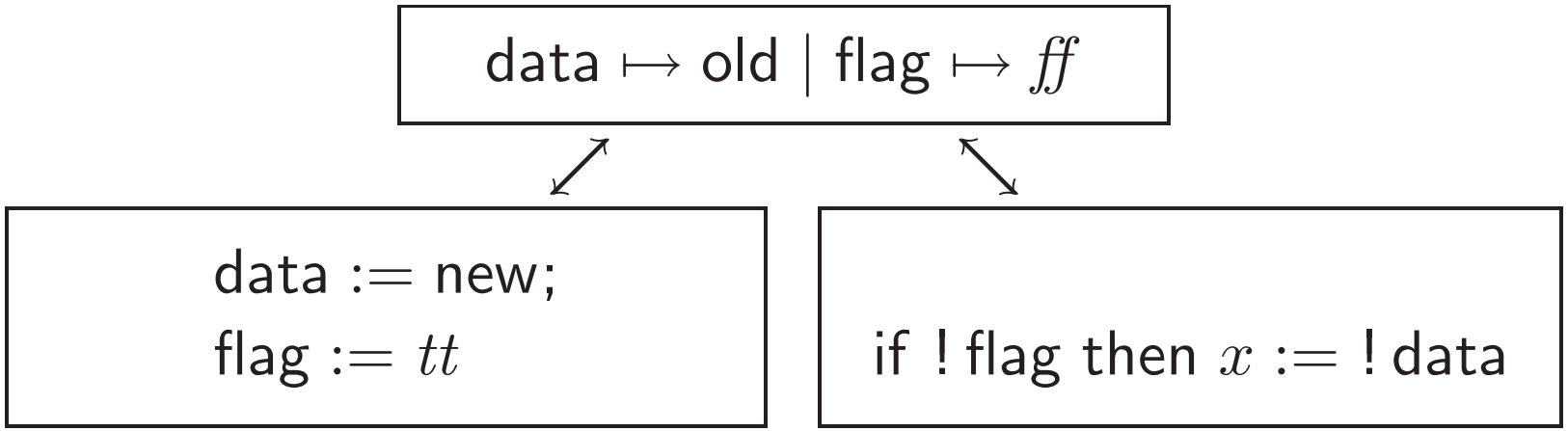


The computation must start with updating flag_0 or flag_1

➥ at most one thread, but not both, can update x

Synchronization (2/2)

A producer/consumer scenario, or safe publication:



To update x the consumer must wait for **flag** being updated

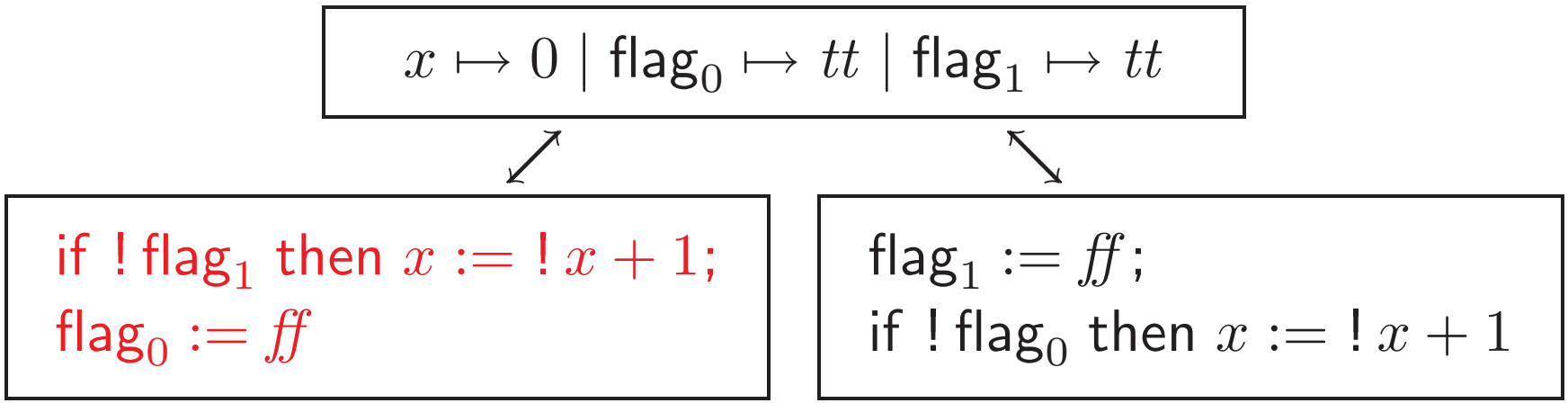
➥ x cannot get the **old** value

Unfortunately... (1/2)

The standard, input/output equivalence of sequential programs is not compatible with parallel composition

$$\begin{array}{l}\mathsf{flag}_0 := \mathit{ff}\,; \\ \text{if } !\,\mathsf{flag}_1 \text{ then } x := !\,x + 1\end{array} \quad \approx \quad \begin{array}{l}\text{if } !\,\mathsf{flag}_1 \text{ then } x := !\,x + 1; \\ \mathsf{flag}_0 := \mathit{ff}\end{array}$$

but cannot replace each other in a concurrent context:



may end up with $x = 2$

Unfortunately... (2/2)

- standard equivalence of sequential program is the basis for optimizing compilers, and hardware, especially relaxed memory models
- correct – and efficient! – for sequential programs, but

↪ **do not implement SC for concurrent threads**

Relaxed Memory Models

memory model = reordering, i.e. relaxation of program order, for
memory accesses (write/read)
+ write visiblity
+ means to maintain program order in some cases
(fences, memory barriers, acquire/release...)

Some examples:

- TSO: relaxing write/read (on different memory locations)
+ read own writes
- PSO: TSO + write/write (idem)
- RMO: PSO + read/write + read/read

Question

How to formalize the semantics of concurrent programs in the context of a relaxed memory model?

Two approaches:

- axiomatic: relations between memory events, satisfying some properties, see Francesco & Luc & Jade
- operational: transitions between configurations, as with programming languages (abstract machines)

An Example: Write Buffering (1/2)

Boudol & Petri POPL 2009, Sewell & al. CACM 2010

- configurations:

$$C = (S, (B_1, P_1), \ldots, (B_n, P_n))$$

S = shared memory
B_i = write buffers
= fifo sequences of writes $\langle x := v \rangle$
P_i = processors = multisets of threads

- labelled transitions: action from a component i

- actions: read/write memory locations $\mathsf{rd}_{x,v}$ and $\mathsf{wr}_{y,w}$ (and more)

An Example: Write Buffering (2/2)

Some rules:

$$\frac{P_i \xrightarrow{\mathsf{wr}_{x,v}} P_i'}{(S, \ldots, (B_i, P_i), \ldots) \xrightarrow{i,\tau} (S, , \ldots, (B_i \cdot \langle x := v \rangle, P_i'), \ldots)}$$

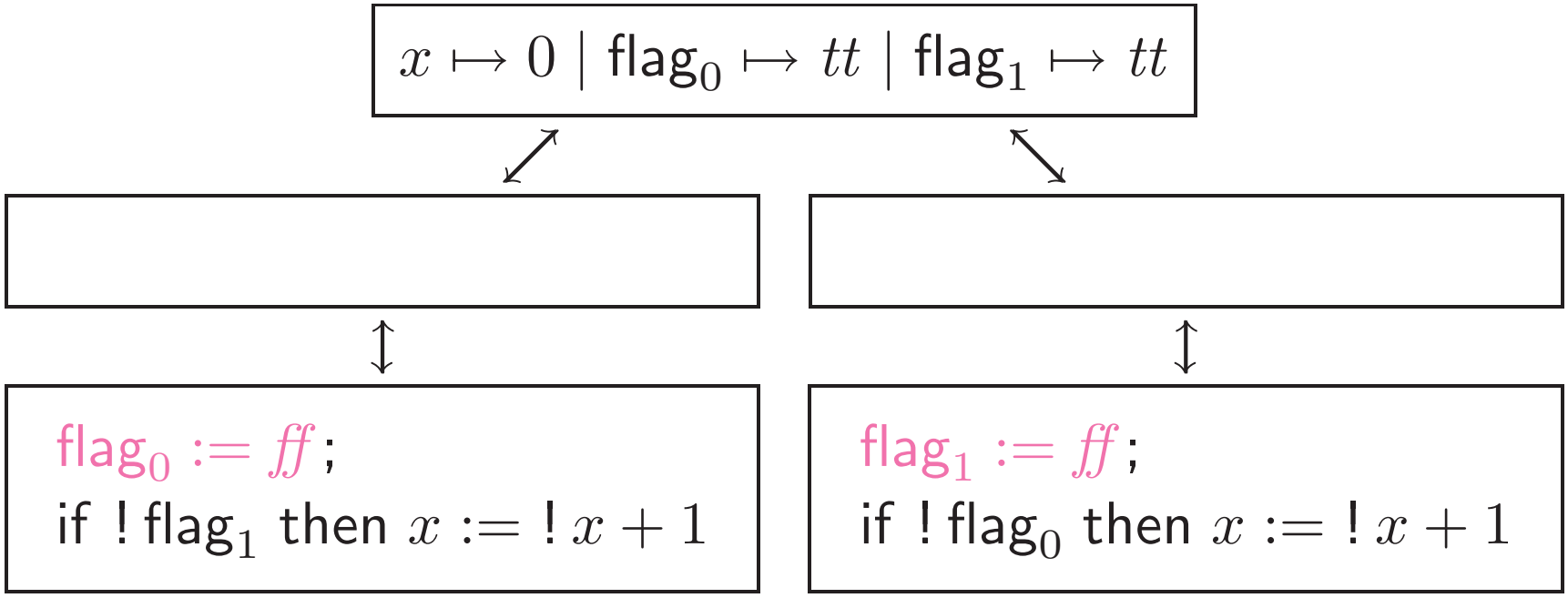
$$\frac{P_i \xrightarrow{\mathsf{rd}_{x,v}} P_i' \qquad B_i(x) = v}{(S, \ldots, (B_i, P_i), \ldots) \xrightarrow{i,\tau} (S, \ldots, (B_i, P_i'), \ldots)}$$

$$\frac{P_i \xrightarrow{\mathsf{rd}_{x,v}} P_i' \qquad S(x) = v \qquad x \notin \mathsf{dom}(B_i)}{(S, \ldots, (B_i, P_i), \ldots) \xrightarrow{i,\mathsf{rd}_{x,v}} (S, \ldots, (B_i, P_i'), \ldots)}$$

$$\frac{B_i \equiv_{\mathsf{TSO/PSO}} \langle x := v \rangle \cdot B_i'}{(S, \ldots, (B_i, P_i), \ldots) \xrightarrow{i,\mathsf{wr}_{x,v}} (S[x := v], \ldots, (B_i', P_i,), \ldots)}$$

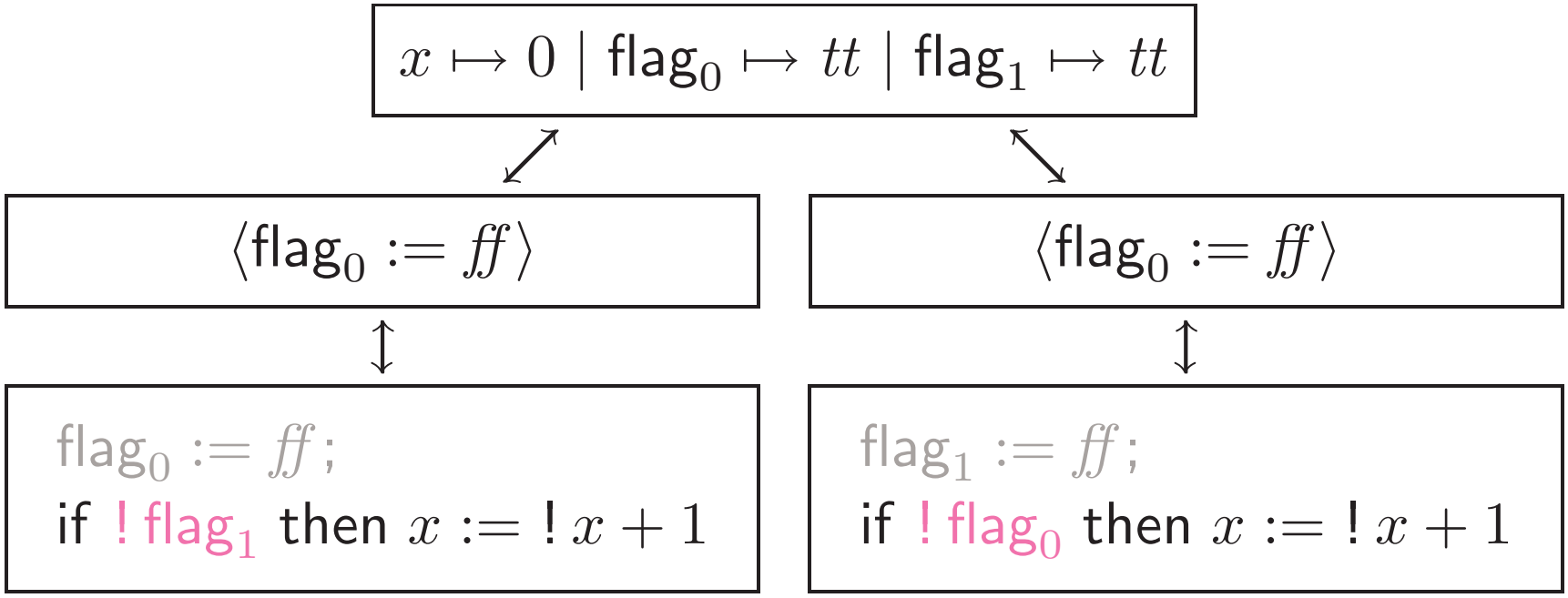
For Instance – TSO

The "mutual exclusion" example:



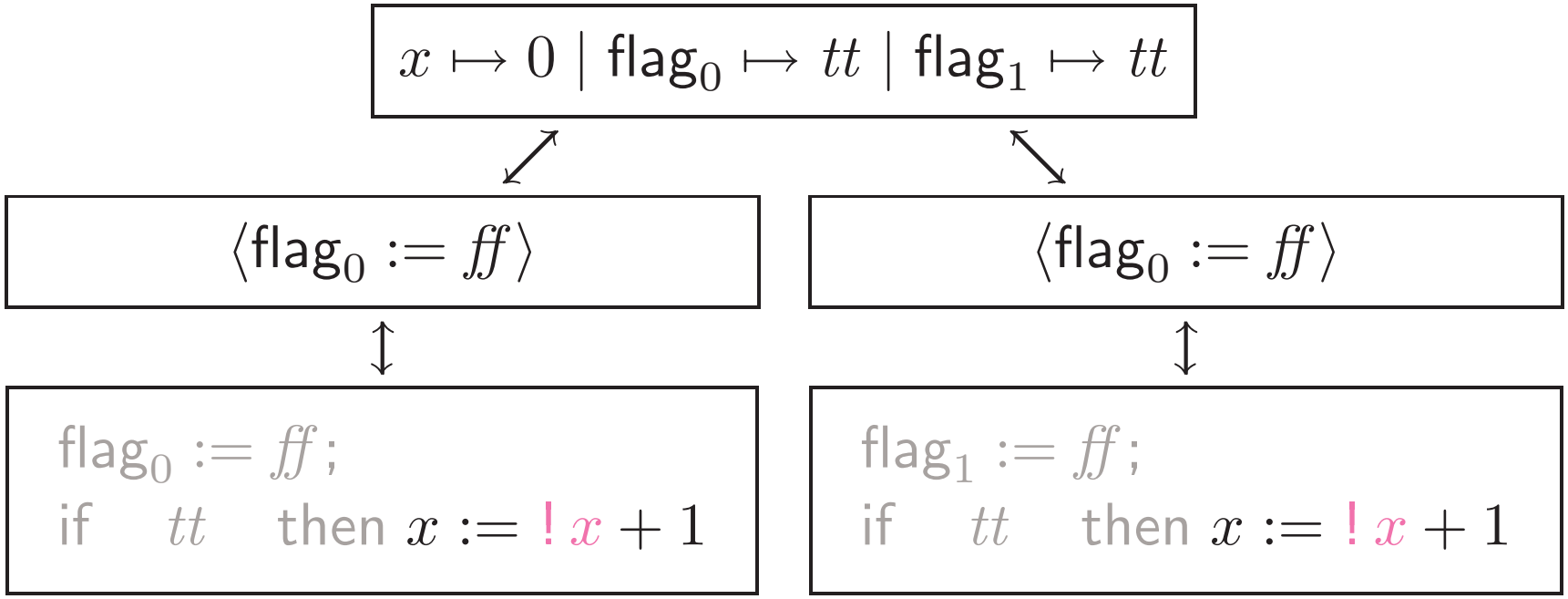
For Instance – TSO

The writes are issued and put in the buffers:



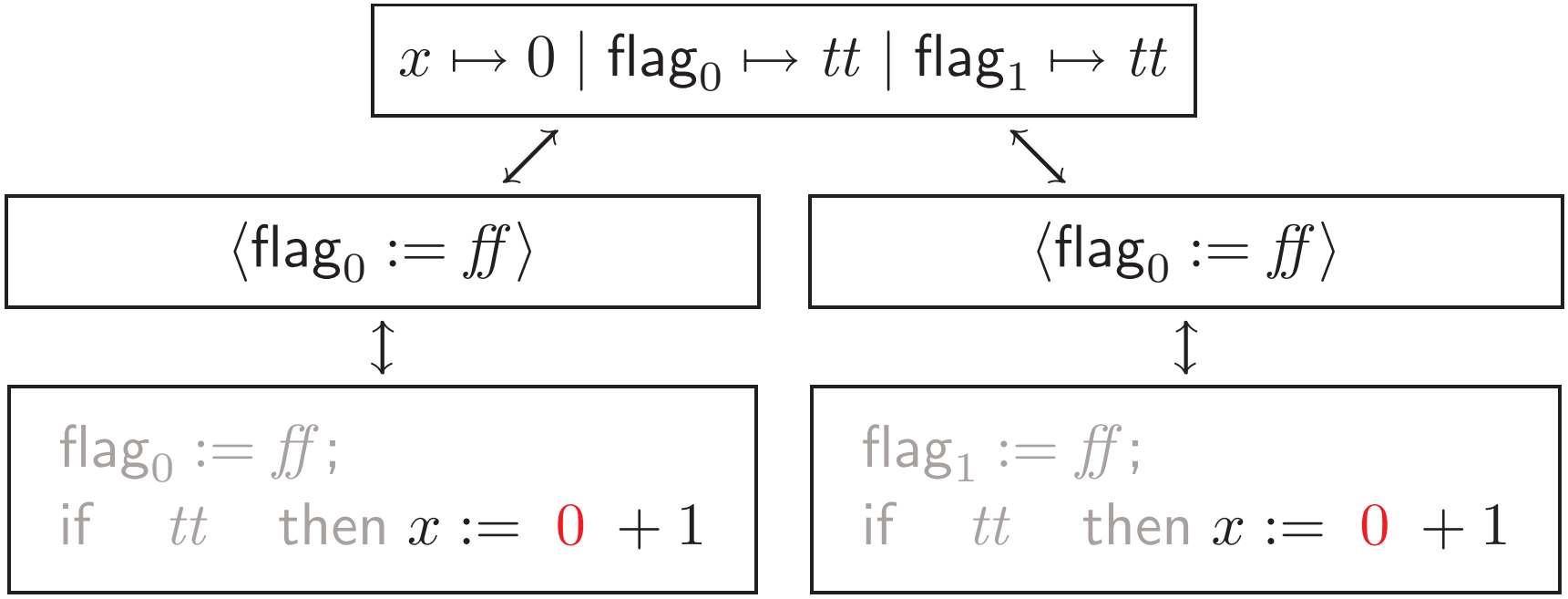
For Instance – TSO

The flags are read (tt) from the shared memory, **then** branches selected:



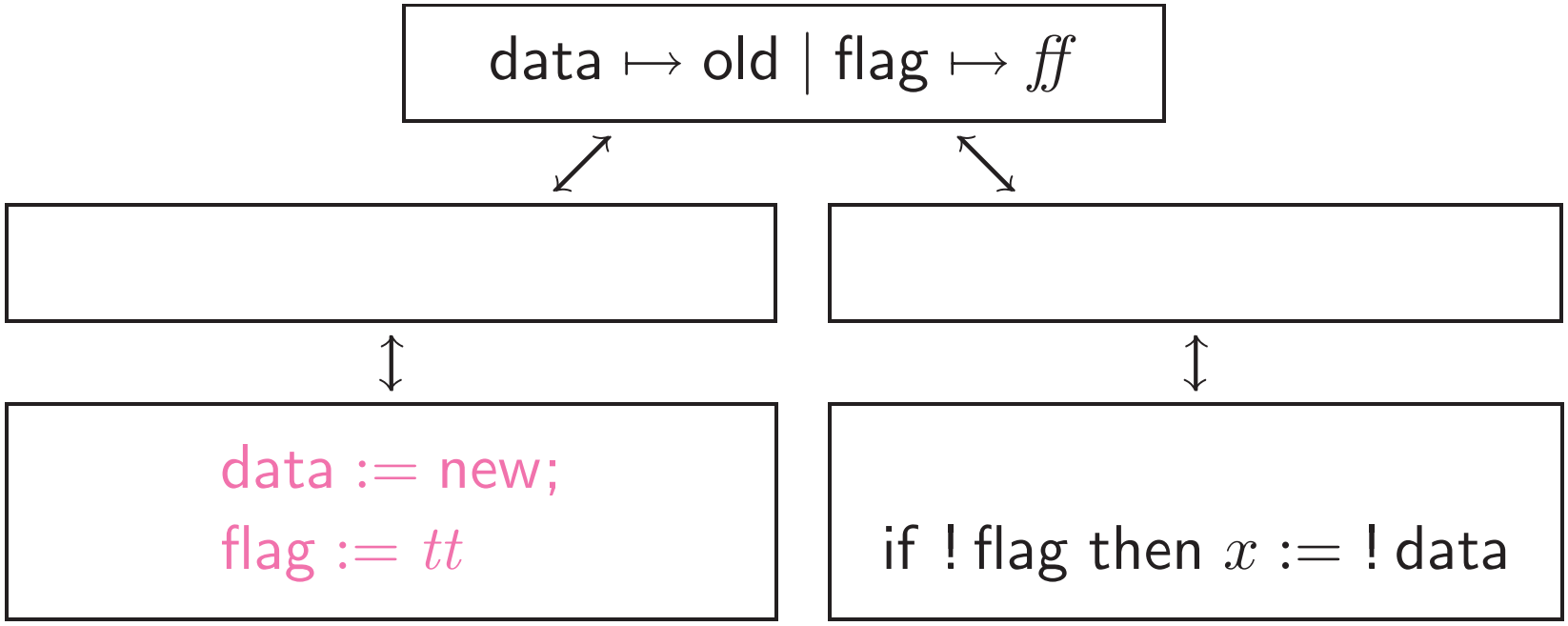
For Instance – TSO

Value of x taken from the shared memory – not in the buffers:



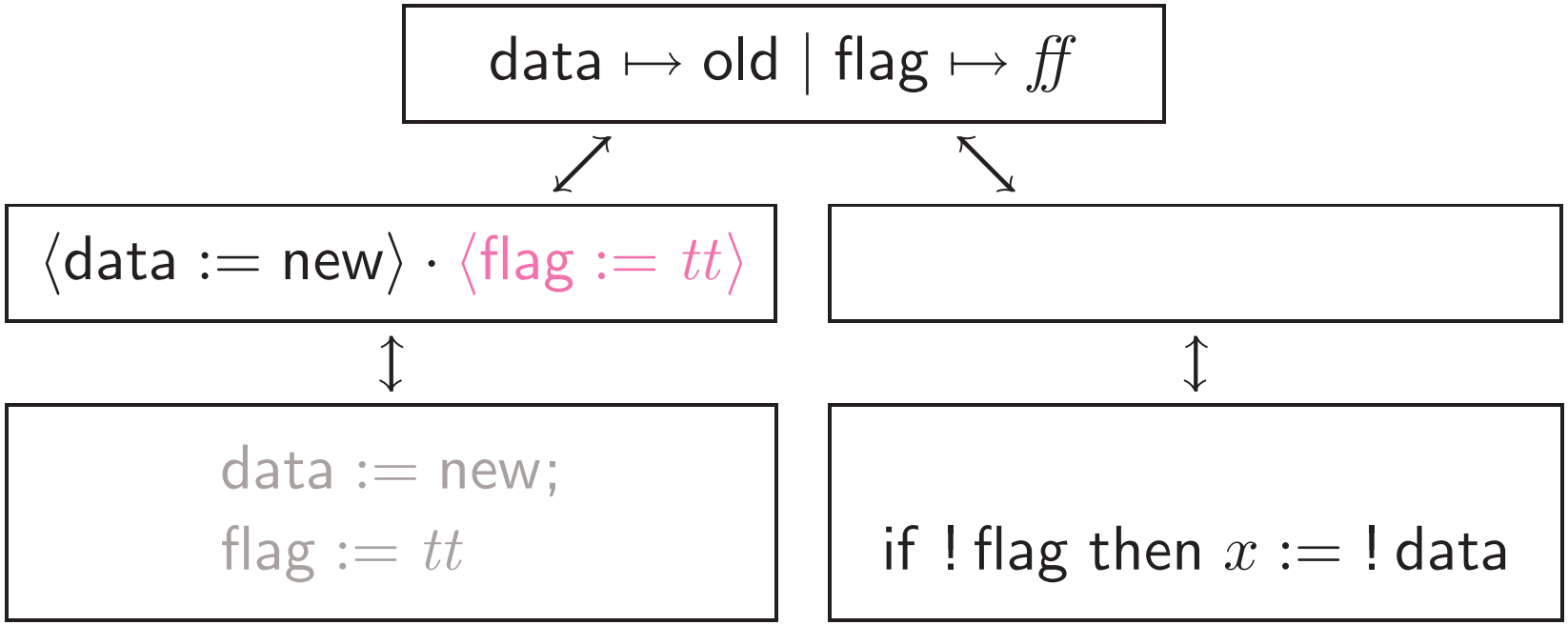
For Instance – PSO

The producer/consumer example



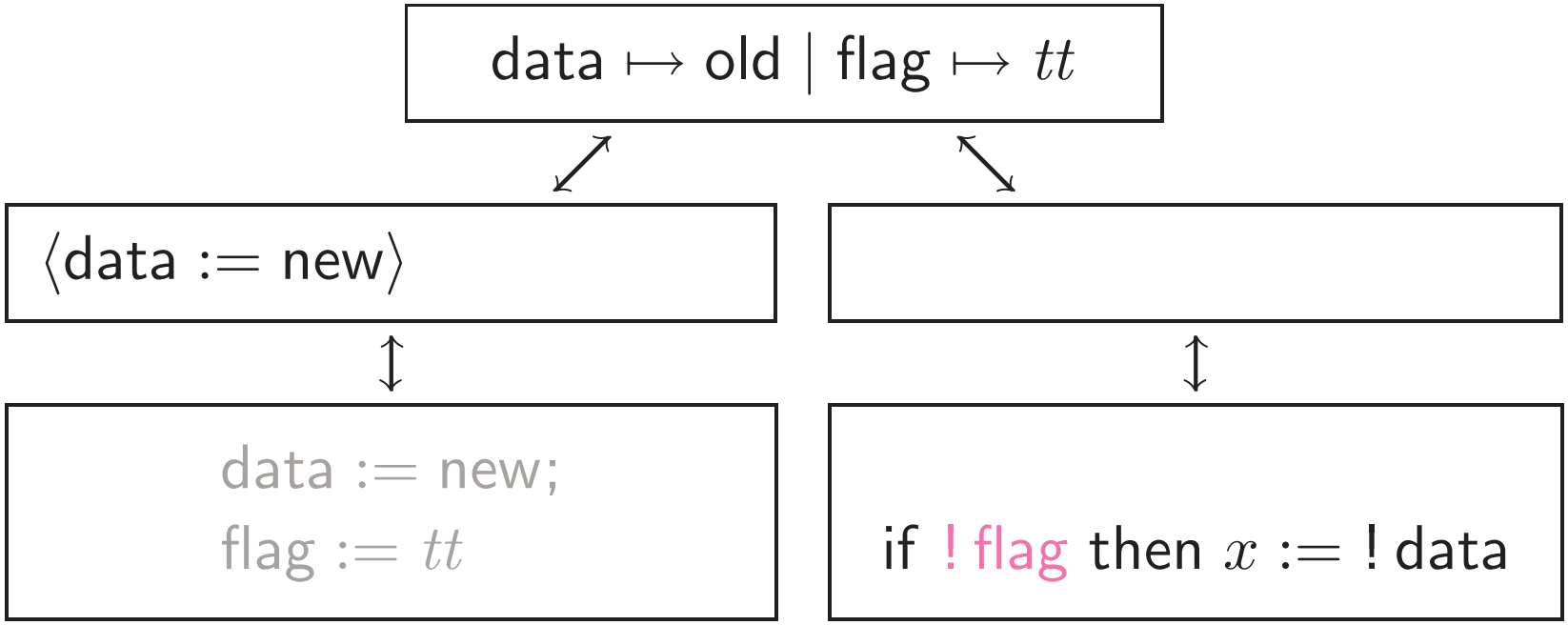
For Instance – PSO

The writes are issued:



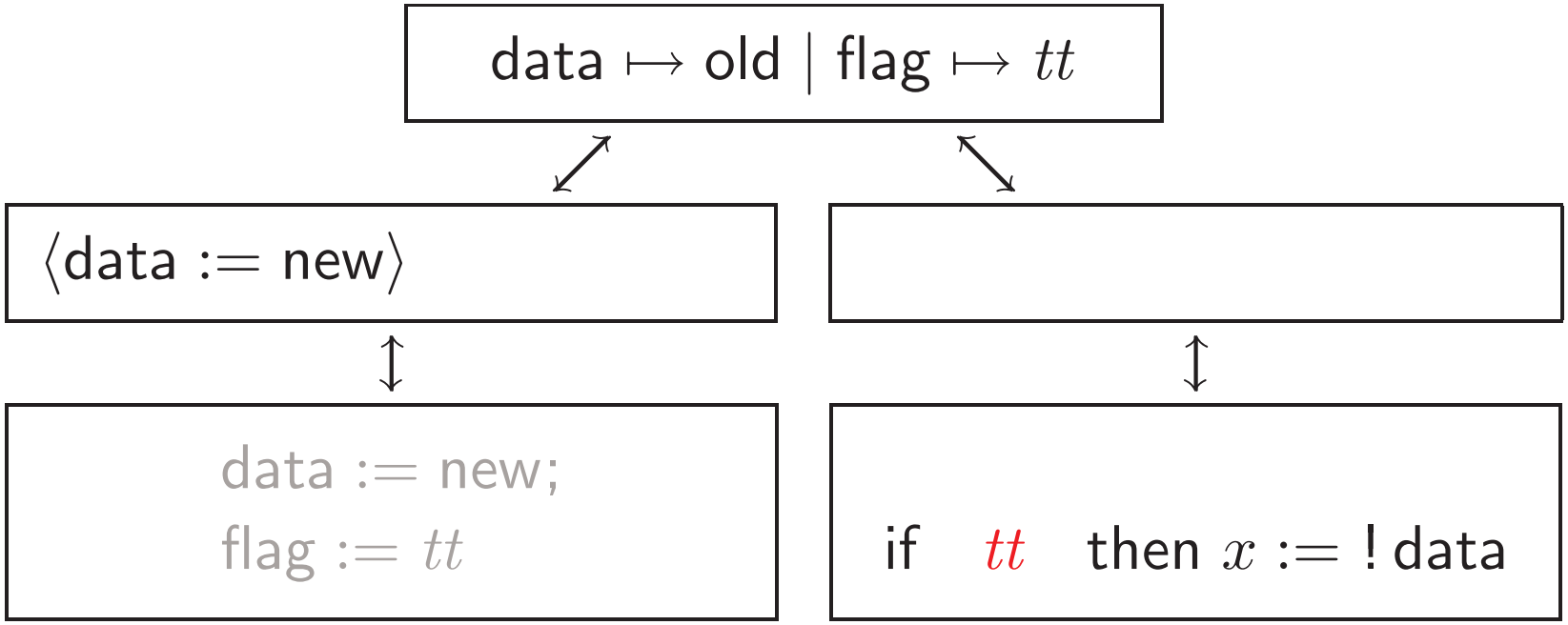
For Instance – PSO

The memory is partially updated:



For Instance – PSO

The **flag** is read from the memory:



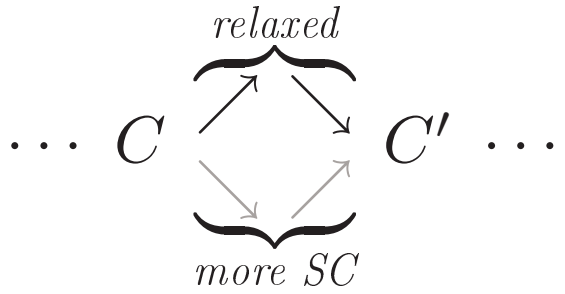
then x may take the **old** value

A Metatheoretical Question

How to prove that a program appears to have a sequentially consistent semantics?

➥ an ideal tool: **GGJJ**'s equivalence by permutation of independent steps

Why? Under certain conditions (e.g. DRF) one can turn a weak computation into an interleaving computation, by permuting independent steps:



Some Results

[Boudol & Petri 2009]: a proof of the "DRF guarantee" for the TSO/PSO weak memory models using GGJJ's equivalence:

- DRF programs are "well-synchronized"
- DRF $\Rightarrow$ (weak) computations are equivalent (by permutations) to SC (i.e. interleaving) computations

Speculative Computations (1/2)

Relaxation not allowed with write buffering: read/write order (RMO, PowerPC...)

↪ a more general approach: speculative computations

= computing (almost) anywhere in the code, introducing speculation contexts, e.g.

- $e;[]$, or more generally $(\mathsf{let}\ x = e\ \mathsf{in}\ []) = (\lambda x[]e)$: out of order evaluation
- $(\mathsf{if}\ e\ \mathsf{then}\ []\ \mathsf{else}\ e_1)$ and $(\mathsf{if}\ e\ \mathsf{then}\ e_0\ \mathsf{else}\ [])$: branch prediction

Speculative Computations (2/2)

Speculations can go wrong:

- violation of data dependency – read/write on the same location
- violation of control dependency – computing in the wrong branch

↪ validity criterion: a speculation is valid iff it is, up to the reorderings allowed in a given memory model, a reordering – à la **GGJJ** – of a normal computation

A result: write buffering coincides with speculations where write/read (TSO) and write/write (PSO) relaxations are allowed.

More in the PhD Thesis of **Gustavo Petri** (November 2010)

thanks again, GG & JJ

and thanks to Pierre-Louis and Paul André