Ogre et Pythia: An invariance proof method for weak consistency models

Jade Alglave (MSR-Cambridge, UCL, UK)
Patrick Cousot (NYU, Emer. ENS, PSL)

POPL 2017
18 January 2017

Objective

Example (Peterson)

```
0:{ w F1 false; w F2 false; w T 0; }
```

```
1: w[] F1 true
2: w[] T 2
3: do
5:     r[] R1 F2
6:     r[] R2 T
7: while R1 ∧ R2 ≠ 1
8: ¬at 28
9: w[] F1 false
10:
```

∥

```
21:w[] F2 true;
22:w[] T 1;
23:do
25:    r[] R3 F1;
26:    r[] R4 T;
27:while R3 ∧ R4 ≠ 2;
28:¬at 8
29:w[] F2 false;
39:
```

critical section

An invariance proof method for WCMs

- Extend Lamport's invariance proof method for parallel programs from sequentially consistent to weak consistency models so that
  - The weak consistency model is a *parameter* of the proof
  - We don't have to redo the whole proof when *changing the consistency model*

Note: Owicki & Gries is Lamport with auxiliary variables instead of programs counters

Separating invariance from WCM

- The invariance proof (that a specification S_{inv} is invariant for a program):
  - Done for a program consistency hypothesis S_{com}:
    - Sufficient for the program to be correct
    - Or better, also necessary for correctness (weakest consistency model)
  - This program consistency hypothesis S_{com} is expressed as an invariant
  - sound and (relatively) complete

Separating invariance from WCM

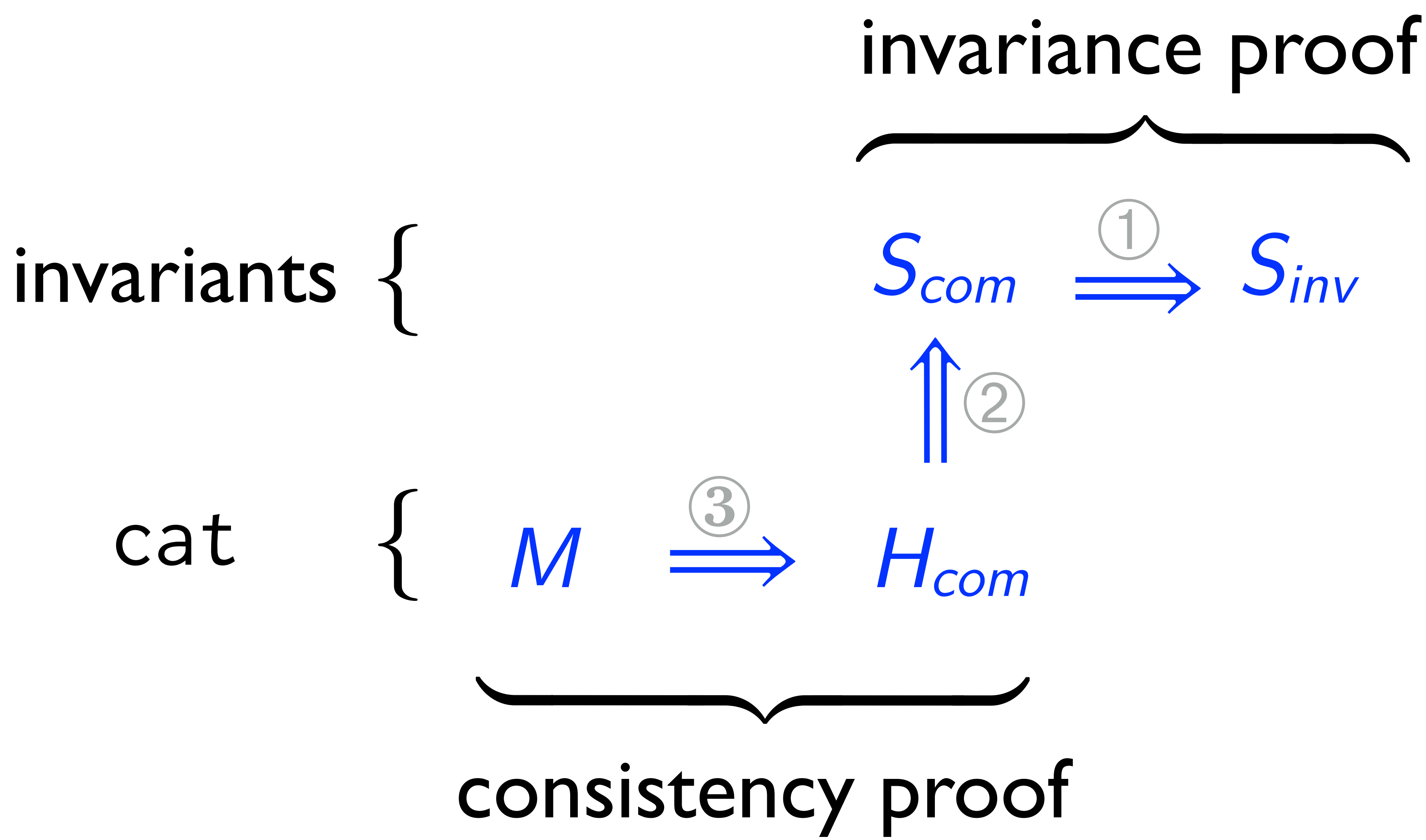
- Consistency proof:

  a. The program consistency hypothesis S_{com} is strengthen into H_{com} written in a consistency specification language (e.g. `cat`)

  b. A `cat` architecture consistency model M is shown to imply the `cat` program consistency model H_{com}

- only b. to be redone when changing the architecture
- sound but possibly incomplete

Methodology



The invariance proof method is designed by abstract interpretation of an *analytic* semantics

Analytic semantics

=

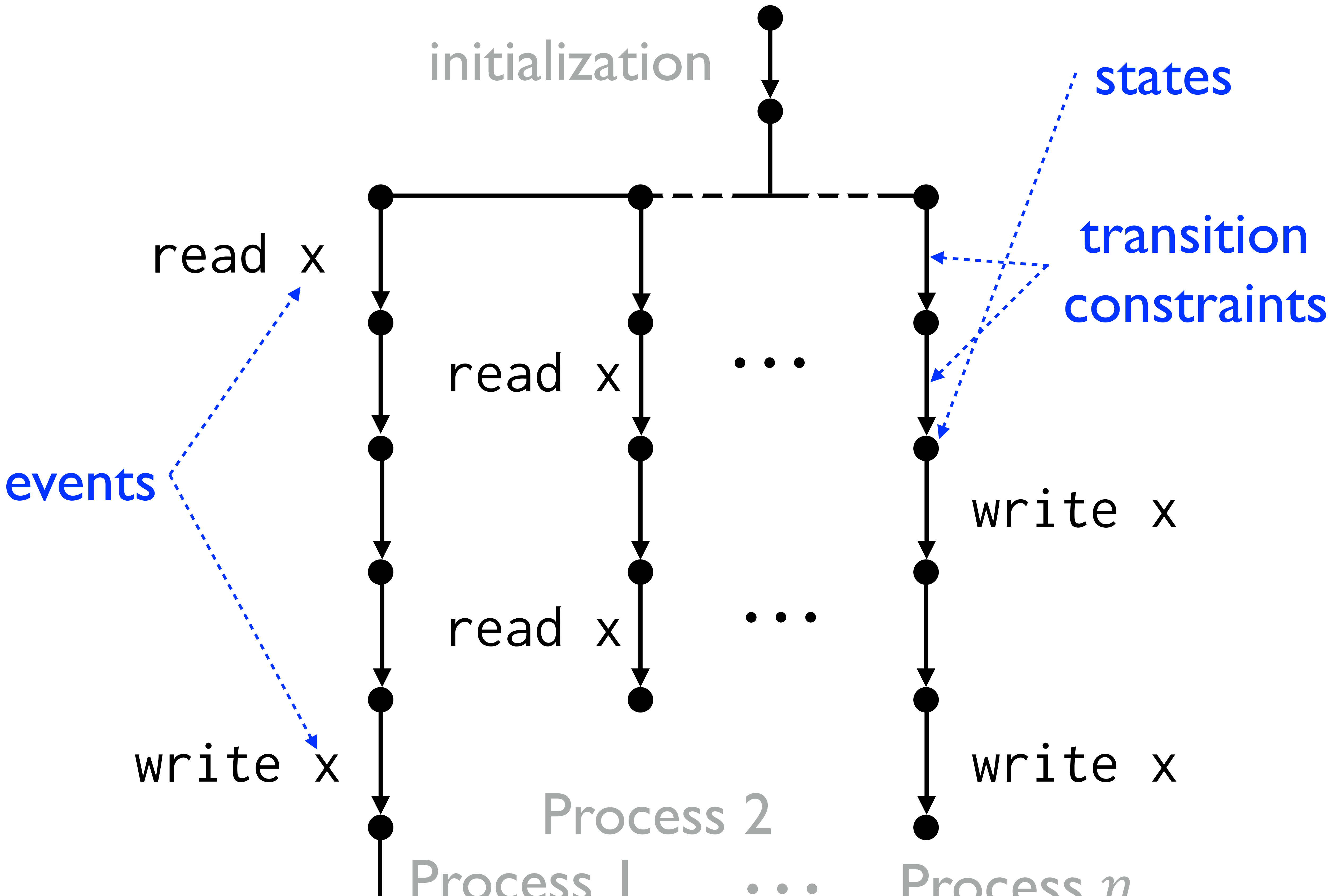
Anarchic semantics

$\lceil$

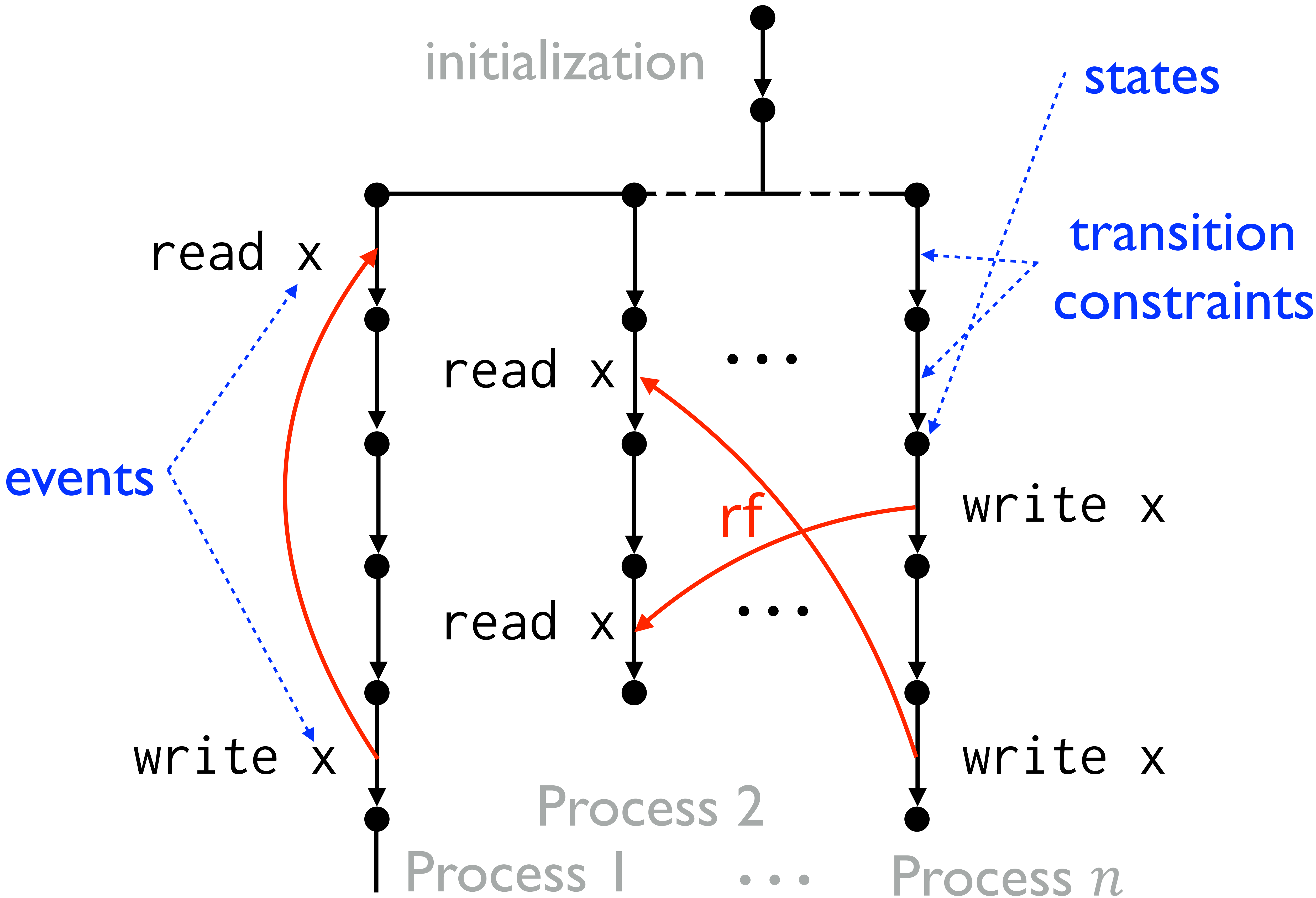
Weak consistency model

The anarchic semantics

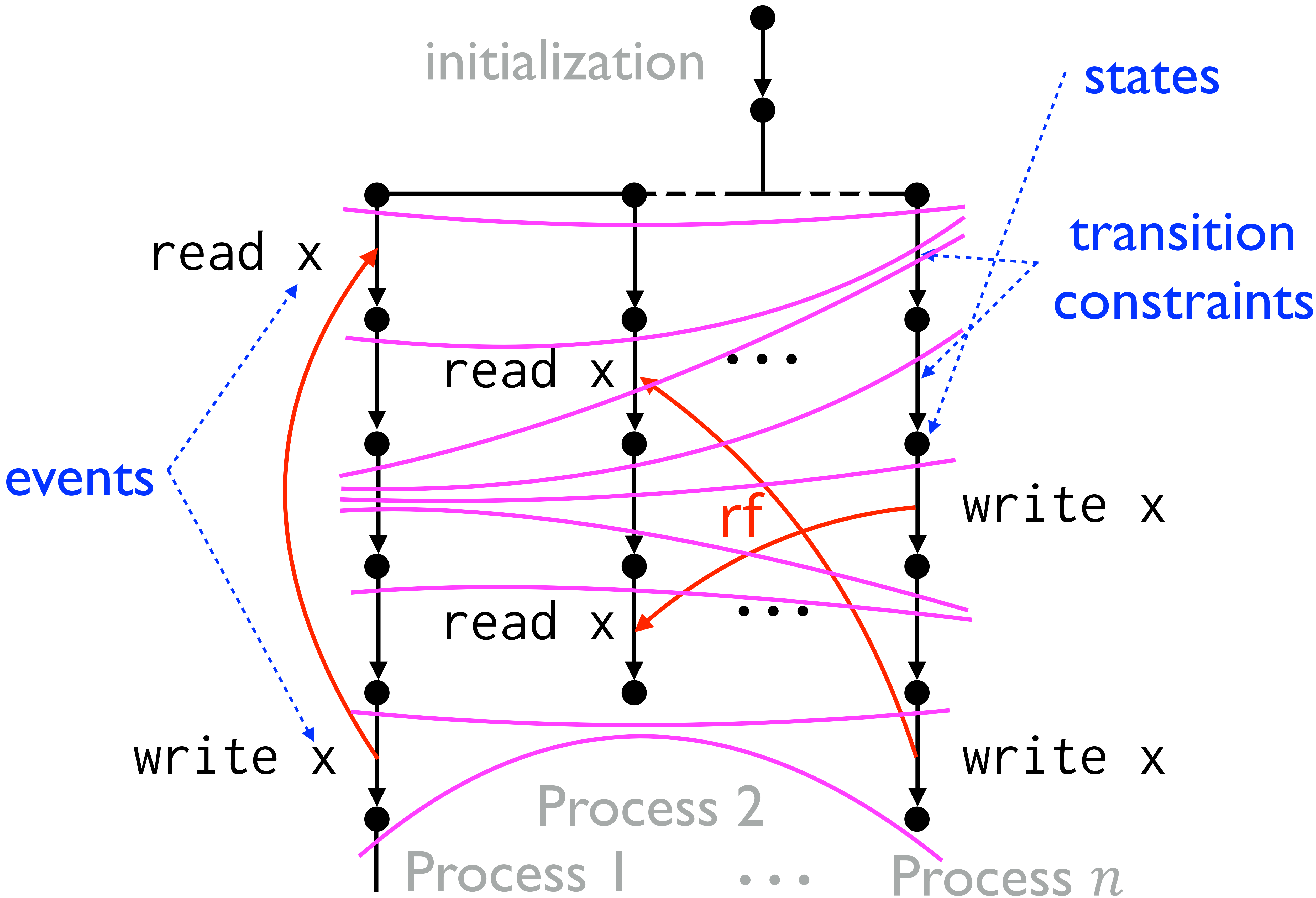
The anarchic semantics



The read-from relation rf



Cuts


initialization
states
transition
constraints
events
read x
read x
read x
write x
write x
write x
rf
...
...
Process 1
Process 2
...
Process n

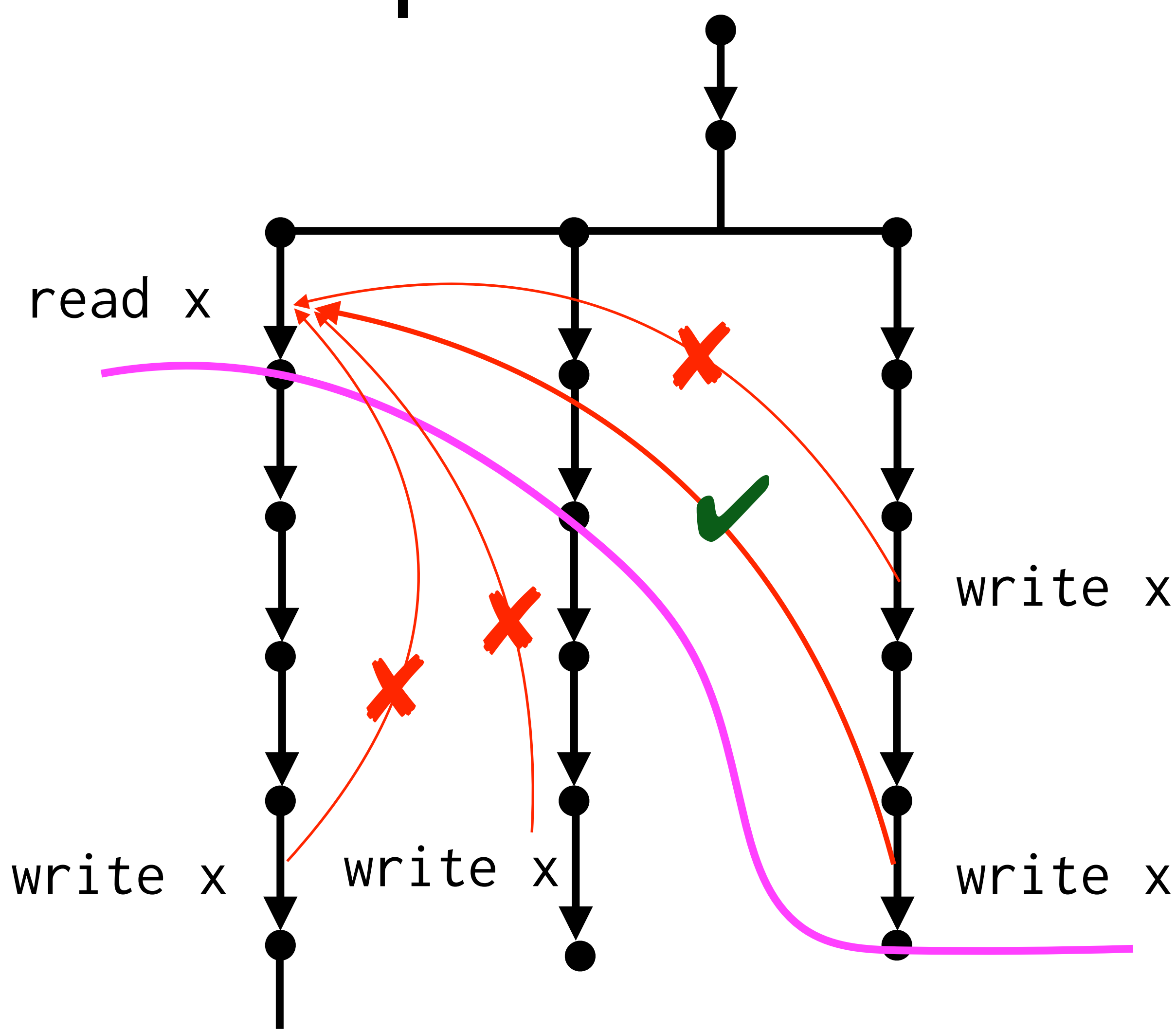
Anarchic semantics of fences

- The anarchic semantics of (localized) fences is `skip` (the state is unmodified)
- Fences are static marker events used by the WCM in `cat` to restrict the read-from relation rf

The weak consistency model

Weak consistency models

- Put restrictions on the read-from relation rf
- e.g. sequential consistency: a read at a cut reads from that last write in a process before that cut



Difficulties

Naming entities

- Invariants are logical formulæ
- can only describe entities that they name
- L/O-G use the name of shared variables to designate their current value in invariants

Naming entities

- Invariants are logical formulæ
- can only describe entities that they name
- L/O-G use the name of shared variables to designate their current value in invariants

Difficulty

- Meaningless with WCMs since there is no notion of ``the current value of a shared variable"

What is known on communications?

- Each process only knows the value of the shared variables from its last read
- Need to be named $\longrightarrow$ Pythia Variables

What we know on communications?

- Each process only knows the value of the shared variables from its last read
- Need to be named $\longrightarrow$ Pythia Variables

Difficulty

- Its dynamic, not static!
- A program read action can read from a different write each time it is executed $\longrightarrow$ Stamps (abstraction of local time)

Back to the anarchic semantics

State

- Per process:
  - A stamp (local time, no global time)
  - A program counter
  - The value of the local variables (registers) of the process
  - The stamped pythia variables (uniquely identifying all reads along a trace)
  - The value of the pythia variables (what was read)
- The read-from relation (rf)

Example (Peterson)

```
0:{ w F1 false; w F2 false; w T 0; }
```

P0:	P1:
1:w[] F1 true	10:w[] F2 true;
2:w[] T 2	11:w[] T 1;
3:do $\{i\}$	12:do $\{j\}$
4: r[] R1 F2 $\{\leadsto F2^i_4\}$	13: r[] R3 F1; $\{\leadsto F1^j_{13}\}$
5: r[] R2 T $\{\leadsto T^i_5\}$	14: r[] R4 T; $\{\leadsto T^j_{14}\}$
6:while R1 $\wedge$ R2 $\neq$ 1 $\{i_{\mathrm{end}}\}$	15:while R3 $\wedge$ R4 $\neq$ 2; $\{j_{\mathrm{end}}\}$
7:skip (* CS1 *)	16:skip (* CS2 *)
8:w[] F1 false	17:w[] F2 false;

Stamps (loop counters)

Stamps (on loop exit)

Example (Peterson)

0:{ w F1 false; w F2 false; w T 0; }

P0:
1:w[] F1 true
2:w[] T 2
3:do $\{i\}$
4:    r[] R1 F2 $\{\leadsto \mathtt{F2}^{i}_{4}\}$
5:    r[] R2 T $\{\leadsto \mathtt{T}^{i}_{5}\}$
6:while R1 $\wedge$ R2 $\neq$ 1 $\{i_{\mathrm{end}}\}$
7:skip (* CS1 *)
8:w[] F1 false

P1:
10:w[] F2 true;
11:w[] T 1;
12:do $\{j\}$
13:    r[] R3 F1; $\{\leadsto \mathtt{F1}^{j}_{13}\}$
14:    r[] R4 T; $\{\leadsto \mathtt{T}^{j}_{14}\}$
15:while R3 $\wedge$ R4 $\neq$ 2; $\{j_{\mathrm{end}}\}$
16:skip (* CS2 *)
17:w[] F2 false;

Pythia variables

Stamps (loop counters)

Stamps (on loop exit)

The abstraction

The invariance abstraction

- For each process

The invariance abstraction

- For each process
    - For each program point of that process

The invariance abstraction

- For each process
  - For each program point of that process
    - For each execution of the program

The invariance abstraction

- For each process
  - For each program point of that process
    - For each execution of the program
      - For each cut of that execution going through the program point of that process

The invariance abstraction

- For each process
  - For each program point of that process
    - For each execution of the program
      - For each cut of that execution going through the program point of that process

        collect:

The invariance abstraction

- For each process
  - For each program point of that process
    - For each execution of the program
      - For each cut of that execution going through the program point of that process

        collect:

        - The states of all processes, and

The invariance abstraction

- For each process
  - For each program point of that process
    - For each execution of the program
      - For each cut of that execution going through the program point of that process

        collect:

        - The states of all processes, and
        - The read-from relation rf

Example: Peterson

0: { w F1 false; w F2 false; w T 0; }
$\{F1=false \wedge F2=false \wedge T=0\}$ }

1: $\{R1=0 \wedge R2=0\}$
w[] F1 true
2: $\{R1=0 \wedge R2=0\}$
w[] T 2
3: $\{R1=0 \wedge R2=0\}$
do $\{i\}$
4: $\{(i=0 \wedge R1=0 \wedge R2=0) \vee (i>0 \wedge R1=F2_4^{i-1} \wedge R2=T_5^{i-1})\}$
r[] R1 F2 $\{\leadsto F2_4^i\}$
5: $\{R1=F2_4^i \wedge (i=0 \wedge R2=0) \vee (i>0 \wedge R2=T_5^{i-1})\}$
r[] R2 T $\{\leadsto T_5^i\}$
6: $\{R1=F2_4^i \wedge R2=T_5^i\}$
while R1 ∧ R2≠1 $\{i_{end}\}$
7: $\{\neg F2_4^{i_{end}} \vee T_5^{i_{end}}=1\}$
skip (* CS1 *)
8: $\{\neg F2_4^{i_{end}} \vee T_5^{i_{end}}=1\}$
w[] F1 false
9: $\{\neg F2_4^{i_{end}} \vee T_5^{i_{end}}=1\}$

||

10: $\{R3=0 \wedge R4=0\}$
w[] F2 true;
11: $\{R3=0 \wedge R4=0\}$
w[] T 1;
12: $\{R3=0 \wedge R4=0\}$
do $\{j\}$
13: $\{(j=0 \wedge R3=0 \wedge R4=0) \vee (j>0 \wedge R3=F1_{13}^{j-1} \wedge R4=T_{14}^{j-1})\}$
r[] R3 F1 $\{\leadsto F1_{13}^j\}$;
14: $\{R3=F1_{13}^j \wedge (j=0 \wedge R4=0) \vee (j>0 \wedge R4=T_{14}^{j-1})\}$
r[] R4 T; $\{\leadsto T_{14}^j\}$
15: $\{R3=F1_{13}^j \wedge R4=T_{14}^j)\}$
while R3 ∧ R4≠2 $\{j_{end}\}$;
16: $\{\neg F1_{13}^{j_{end}} \vee T_{14}^{j_{end}}=2\}$
skip (* CS2 *)
17: $\{\neg F1_{13}^{j_{end}} \vee T_{14}^{j_{end}}=2\}$
w[] F2 false;
18: $\{\neg F1_{13}^{j_{end}} \vee T_{14}^{j_{end}}=2\}$

Example: Peterson

0: { w F1 false; w F2 false; w T 0; }

$\{F1=false \wedge F2=false \wedge T=0\}$ }

1: $\{R1=0 \wedge R2=0\}$

w[] F1 true

2: $\{R1=0 \wedge R2=0\}$

w[] T 2

3:

4:

5: $\{R1=F2^{i}_{4} \wedge (i=0 \wedge R2=0) \vee (i>0 \wedge R2=T^{i-1}_{5})\}$

r[] R2 T $\{\rightsquigarrow T^{i}_{5}\}$

6: $\{R1=F2^{i}_{4} \wedge R2=T^{i}_{5}\}$

while R1 $\wedge$ R2$\neq$1 $\{i_{end}\}$

7: $\{\neg F2^{i_{end}}_{4} \vee T^{i_{end}}_{5}=1\}$

skip (* CS1 *)

8: $\{\neg F2^{i_{end}}_{4} \vee T^{i_{end}}_{5}=1\}$

w[] F1 false

9: $\{\neg F2^{i_{end}}_{4} \vee T^{i_{end}}_{5}=1\}$

10: $\{R3=0 \wedge R4=0\}$

w[] F2 true;

11: $\{R3=0 \wedge R4=0\}$

w[] T 1;

14: $\{R3=F1^{j}_{13} \wedge (j=0 \wedge R4=0) \vee (j>0 \wedge R4=T^{j-1}_{14})\}$

r[] R4 T; $\{\rightsquigarrow T^{j}_{14}\}$

15: $\{R3=F1^{j}_{13} \wedge R4=T^{j}_{14})\}$

while R3 $\wedge$ R4$\neq$2 $\{j_{end}\}$;

16: $\{\neg F1^{j_{end}}_{13} \vee T^{j_{end}}_{14}=2\}$

skip (* CS2 *)

17: $\{\neg F1^{j_{end}}_{13} \vee T^{j_{end}}_{14}=2\}$

w[] F2 false;

18: $\{\neg F1^{j_{end}}_{13} \vee T^{j_{end}}_{14}=2\}$

4: $\{(i=0 \wedge R1=0 \wedge R2=0) \vee (i>0 \wedge R1=F2^{i-1}_{4} \wedge R2=T^{i-1}_{5})\}$

The calculational design of the verification conditions by abstract interpretation

The induction principle

- Given an invariance specification S_{inv} find a stronger inductive invariant S_{ind}
- Prove that S_{ind} satisfy verification conditions
  - Holds after initialization
  - Remains true after a computation step
  - Remains true after a communication
- Assuming S_{com} / H_{com}

- Given an ir … nd a stronger inductive ir …
- Prove that … itions
  - Holds aft …
  - Remains … ep
  - Remains …
- Assuming S_{com} … I_{com}

Verification conditions = abstraction of the concrete transformer for one computation step

Calculational design of the verification conditions

$$\alpha_{\mathsf{inv}}(\alpha_{\mathsf{ana}}[\![H_{com}]\!](S^{\mathsf{a}}[\![\mathtt{P}]\!])) \mathrel{\dot{\subseteq}} S_{inv}$$

$$\Leftrightarrow \alpha_{\mathsf{inv}}(\{\xi \in S^{\mathsf{a}}[\![\mathtt{P}]\!] \mid S[\![H_{com}]\!]\xi = \mathtt{allowed}\}) \mathrel{\dot{\subseteq}} S_{inv} \quad \wr\text{def. } \alpha_{\mathsf{ana}}[\![H_{com}]\!]\int$$

$$\Leftrightarrow \alpha_{\mathsf{inv}}(S^{\mathsf{a}}[\![\mathtt{P}]\!] \cap \{\xi \in S^{\mathsf{a}}[\![\mathtt{P}]\!] \mid S[\![H_{com}]\!]\xi = \mathtt{allowed}\}) \mathrel{\dot{\subseteq}} S_{inv} \quad \wr\text{def. } \cap\int$$

$$\Leftrightarrow \alpha_{\mathsf{inv}}(S^{\mathsf{a}}[\![\mathtt{P}]\!]) \cap \alpha_{\mathsf{inv}}(\{\xi \in \Xi \mid S[\![H_{com}]\!]\xi = \mathtt{allowed}\}) \mathrel{\dot{\subseteq}} S_{inv}$$

$$\wr\text{since } \alpha_{\mathsf{inv}} \text{ preserves intersections}\int$$

$$\Leftrightarrow \alpha_{\mathsf{inv}}(S^{\mathsf{a}}[\![\mathtt{P}]\!]) \mathrel{\dot{\cap}} \alpha_{\mathsf{inv}}(\alpha_{\mathsf{ana}}[\![H_{com}]\!](S^{\mathsf{a}}[\![\mathtt{P}]\!])) \mathrel{\dot{\subseteq}} S_{inv} \quad \wr\text{def. } \alpha_{\mathsf{ana}}[\![H_{com}]\!]\int$$

$$\Leftrightarrow \exists S_{com} \;.\; \alpha_{\mathsf{inv}}(S^{\mathsf{a}}[\![\mathtt{P}]\!]) \mathrel{\dot{\cap}} S_{com} \mathrel{\dot{\subseteq}} S_{inv} \wedge \alpha_{\mathsf{inv}}(\alpha_{\mathsf{ana}}[\![H_{com}]\!](S^{\mathsf{a}}[\![\mathtt{P}]\!])) \mathrel{\dot{\subseteq}} S_{com}$$

≀(⇐) For soundness, we have $\alpha_{\mathsf{inv}}(S^{\mathsf{a}}[\![\mathtt{P}]\!]) \mathrel{\dot{\cap}} \alpha_{\mathsf{inv}}(\alpha_{\mathsf{ana}}[\![H_{com}]\!](S^{\mathsf{a}}[\![\mathtt{P}]\!])) \mathrel{\dot{\subseteq}} \alpha_{\mathsf{inv}}(S^{\mathsf{a}}[\![\mathtt{P}]\!]) \mathrel{\dot{\cap}} S_{com} \mathrel{\dot{\subseteq}} S_{inv}$;

(⇒) For completeness, we choose to describe exactly the communications that is $S_{com} = \alpha_{\mathsf{inv}}(\alpha_{\mathsf{ana}}[\![H_{com}]\!](S^{\mathsf{a}}[\![\mathtt{P}]\!]))$.∫

$$\Leftrightarrow \exists S_{com} \;.\; (S_{com} \Rightarrow S_{inv}) \wedge (H_{com} \Rightarrow S_{com})$$

by defining the conditional invariance proof $S_{com} \Rightarrow S_{inv}$ to be $\alpha_{\mathsf{inv}}(S^{\mathsf{a}}[\![\mathtt{P}]\!]) \mathrel{\dot{\cap}} S_{com} \mathrel{\dot{\subseteq}} S_{inv}$ and the inclusion proof $H_{com} \Rightarrow S_{com}$ to be $\alpha_{\mathsf{inv}}(\alpha_{\mathsf{ana}}[\![H_{com}]\!](S^{\mathsf{a}}[\![\mathtt{P}]\!])) \mathrel{\dot{\subseteq}} S_{com}$.

• • •

• • •

• • •

Calculational design of the verification conditions

$\alpha_{\text{inv}}(\alpha_{\text{ana}}[\![H_{com}]\!](S^{\text{a}}[\![\text{P}]\!])) \mathrel{\dot{\subseteq}} S_{inv}$

$\Leftrightarrow \alpha_{\text{inv}}(\{\xi \in S^{\text{a}}[\![\text{P}]\!] \mid S[\![H_{com}]\!]\xi = \texttt{allowed}\}) \mathrel{\dot{\subseteq}} S_{inv}$ ⟨def. $\alpha_{\text{ana}}[\![H_{com}]\!]$⟩

$\Leftrightarrow \alpha_{\text{inv}}(S^{\text{a}}[\![\text{P}]\!] \cap \{\xi \in S^{\text{a}}[\![\text{P}]\!] \mid S[\![H_{com}]\!]\xi = \texttt{allowed}\}) \mathrel{\dot{\subseteq}} S_{inv}$ ⟨def. $\cap$⟩

$\Leftrightarrow \alpha_{\text{inv}}(S^{\text{a}}[\![\text{P}]\!]) \cap \alpha_{\text{inv}}(\{\xi \in \Xi \mid S[\![H_{com}]\!]\xi = \texttt{allowed}\}) \mathrel{\dot{\subseteq}} S_{inv}$

$$\Leftrightarrow \exists S_{com} \,.\, (S_{com} \Rightarrow S_{inv}) \wedge (H_{com} \Rightarrow S_{com})$$

⟨($\Rightarrow$) For completeness, we choose to describe exactly the communications that is $S_{com} = \alpha_{\text{inv}}(\alpha_{\text{ana}}[\![H_{com}]\!](S^{\text{a}}[\![\text{P}]\!]))$.⟩

$\Leftrightarrow \exists S_{com} \,.\, (S_{com} \Rightarrow S_{inv}) \wedge (H_{com} \Rightarrow S_{com})$

by defining the conditional invariance proof $S_{com} \Rightarrow S_{inv}$ to be $\alpha_{\text{inv}}(S^{\text{a}}[\![\text{P}]\!]) \mathrel{\dot{\cap}} S_{com} \mathrel{\dot{\subseteq}} S_{inv}$ and the inclusion proof $H_{com} \Rightarrow S_{com}$ to be $\alpha_{\text{inv}}(\alpha_{\text{ana}}[\![H_{com}]\!](S^{\text{a}}[\![\text{P}]\!])) \mathrel{\dot{\subseteq}} S_{com}$.

...

...

...

Verification conditions

- Sequential proof
- Non-interference proof
  (like L/O-G but for different kind of invariants)
- Communication proof
  - a read event reading from a write event must be in rf
  - the value read for a variable is the one written
  - reading is fair in rf (cannot be delayed indefinitely)
  - …

  (useless in L/O-G since rf is fixed)

The program consistency hypothesis S_{com}

Communication hypothesis S_{com}

- A sufficient communication hypothesis can be discovered by calculational design:

$$S_{com} \Longrightarrow (S_{ind} \Longrightarrow S_{inv})$$

Communication hypothesis

Program inductive anarchic invariant

Program invariance specification

- i.e. (Sind ∧ ¬Sinv) ⟹ ¬Scom

- Necessary: by counter examples

Proving Consistency

$$H_{com} \Longrightarrow S_{com}$$

`cat` invariant

Proof method

- **Obtained by calculational design:**

$$\alpha_{\mathsf{inv}}(\alpha_{\mathsf{ana}}[\![H_{com}]\!](S^{\mathsf{a}}[\![\mathtt{P}]\!])) \mathrel{\dot\subseteq} S_{com}$$

$$\Leftrightarrow \alpha_{\mathsf{inv}}(S^{\mathsf{ana}}[\![H_{com}]\!]\mathtt{P}) \mathrel{\dot\subseteq} S_{com} \qquad \wr\text{def. } S^{\mathsf{ana}}[\![H_{com}]\!]\mathtt{P}\int$$

$$\Leftrightarrow \forall \xi \in S^{\mathsf{ana}}[\![H_{com}]\!]\mathtt{P} \,.\, \alpha_{\mathsf{inv}}(\{\xi\}) \mathrel{\dot\subseteq} S_{com} \qquad \wr\alpha_{\mathsf{inv}} \text{ preserves } \cup\int$$

$$\Leftrightarrow \forall \xi \in S^{\mathsf{ana}}[\![H_{com}]\!]\mathtt{P} \,.\, \dot{\bigcup_{p=1}^{n}} \dot{\bigcup_{\mathtt{L}\in\mathtt{P}_p}} \{\alpha_{\mathsf{inv}}(\xi')_p(\mathtt{L}) \mid \xi' \in \{\xi\}\} \mathrel{\dot\subseteq} S_{com}$$

$$\wr\text{def. (19) of } \alpha_{\mathsf{inv}}\int$$

$$\Leftrightarrow \forall(\tau_{\mathsf{start}} \times \prod_{p=0}^{n-1} \tau_p \times \pi \times \mathsf{rf}) \in S^{\mathsf{ana}}[\![H_{com}]\!]\mathtt{P} \,.\, \forall p \in [1,n] \,.\, \forall \mathtt{L} \in \mathtt{P}_p \,.$$
$$\alpha_{\mathsf{inv}}(\tau_{\mathsf{start}} \times \prod_{p=0}^{n-1} \tau_p \times \pi \times \mathsf{rf})_p(\mathtt{L}) \subseteq {S_{com}}_p(\mathtt{L})$$

$\wr$def. $\in$, $\dot\bigcup$, $\dot\subseteq$, and $S^{\mathsf{ana}}[\![H_{com}]\!]\mathtt{P}$ so that ξ has the form $\xi = \tau_{\mathsf{start}} \times \prod_{p=0}^{n-1} \tau_p \times \pi \times \mathsf{rf}$. By def. (19) of α_{inv} and $\subseteq$, we get$\int$

$$\Leftrightarrow \forall(\tau_{\mathsf{start}} \times \prod_{p=0}^{n-1} \tau_p \times \pi \times \mathsf{rf}) \in S^{\mathsf{ana}}[\![H_{com}]\!]\mathtt{P} \,.\, \forall i \in \tag{20}$$
$$[1,n] \,.\, \forall \mathtt{L} \in \mathtt{P}_p \,.\, \forall q \in [0,n[\,.\, \forall k_q < |\tau_q| \,.$$
$$(\underline{\tau_q}_{k_q} = \mathfrak{s}\langle \kappa_{q,k_q}, \theta_{q,k_q}, \rho_{q,k_q}, \nu_{q,k_q}\rangle \wedge \kappa_{p,k_p} = \mathtt{L}) \Rightarrow$$
$$\langle \kappa_{0,k_0}, \theta_{0,k_0}, \rho_{0,k_0}, \nu_{0,k_0}, \ldots, \nu_{p-1,k_{p-1}}, \theta_{p,k_p}, \rho_{p,k_p}, \nu_{p,k_p},$$
$$\kappa_{p+1,k_{p+1}}, \ldots, \kappa_{n-1,k_{n-1}}, \theta_{n-1,k_{n-1}}, \rho_{n-1,k_{n-1}}, \nu_{n-1,k_{n-1}}, \mathsf{rf}\rangle$$
$$\in {S_{com}}_i(\mathtt{L})$$

Proof method

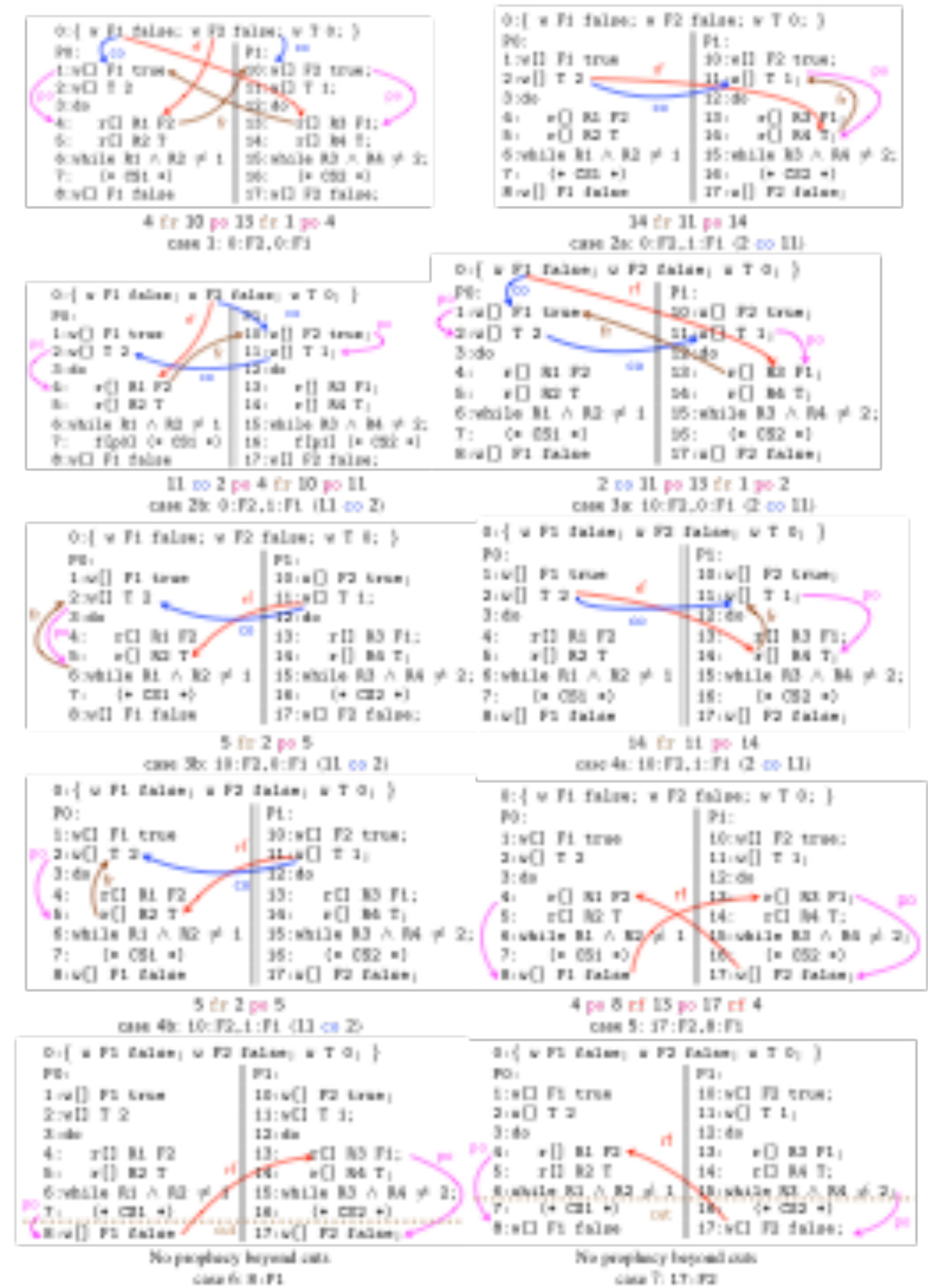
- The anarchic invariants can be used to calculate all communication scenarios violating S_{com}
- These scenarios must be forbidden by the cat specification H_{com}

(no need to reason at the level of traces of the anarchic semantics)

Example (Peterson)

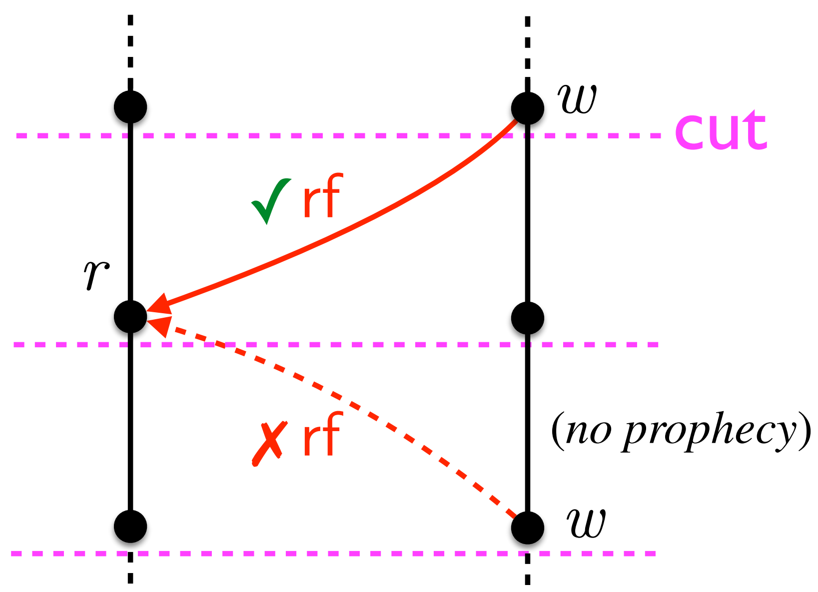
Communication scenarios violating S_{com} for Peterson

$$S_{com} \triangleq \neg[\exists i, j.[\mathit{rf}\langle \mathtt{F2}_4^i,\ 0,\ \mathsf{false}\rangle \vee \mathit{rf}\langle \mathtt{F2}_4^i,\ 17,\ \mathsf{false}\rangle \vee \mathit{rf}\langle \mathtt{T}_5^i,\ 11,\ 1\rangle] \wedge [\mathit{rf}\langle \mathtt{F1}_{13}^j,\ 0,\ \mathsf{false}\rangle \vee \mathit{rf}\langle \mathtt{F1}_{13}^j,\ 8,\ \mathsf{false}\rangle \vee \mathit{rf}\langle \mathtt{T}_{14}^j,\ 2,\ 2\rangle]]$$



Incompleteness

- In general you have to add fences for H_{com} (do not change the invariants, S_{inv}, S_{ind}, and S_{com} remain valid)
- S_{com} can refer to communicated values not H_{com} in `cat` (redesign your algorithm without assuming that the hardware does know about communicated values)
- `cat` may not be expressive enough:



No read beyond cut

Proving Architectural Consistency

$$M \Longrightarrow H_{com}$$

cat   cat

$H_{com} \Longrightarrow M$ *in* cat

- sound and complete proof method
- unpublished paper of JA and PC with Luc Maranget

Beyond L/O-G: non-starvation

Reasoning on one execution only

- A particular execution can be uniquely characterized by its read-from relation rf
- We can reason on one execution only (Scom for this execution + Sind)
- Not directly possible with L/O-G
- Can be used to prove non-starvation

Non-starvation (e.g. PostgrSQL)

- Consider all traces that may starve (for an appropriate S'_{com} for each trace)
- Prove each of them to be infeasible:
  - the inductive invariant S_{ind} under the program communication hypothesis S_{com} is unsatisfied
  - or, by strengthening the program communications S_{com} (maybe implemented by adding fences in H_{com})
  - or, by a fairness hypothesis.

Communication fairness hypothesis(*)

- All writes eventually hit the memory:
  - If, at a cut of the execution, all the processes infinitely often write the same value υ to a shared variable $\mathtt{x}$ and only that value υ
  - and from a later cut point of that execution, a process infinitely often repeats reads to that variable $\mathtt{x}$
  - then the reads will end up reading that value υ

(*) The SPARC Architecture Manual, Version 8, Section K2, p. 283: ``if one processor does an *S* , and another processor repeatedly does *L* 's to the same location, then there is an *L* that will be after the *S*''.

Conclusion

Conclusion

- To design a correct parallel algorithm, specify:
  - the algorithm
  - the invariance specification S_{inv}
  - the required program consistency model S_{com}
- Find an anarchic inductive invariant S_{ind} satisfying the verification conditions such that $(S_{com} \wedge S_{inv}) \Longrightarrow S_{inv}$

Conclusion

- To implement a parallel algorithm correctly:
  - Implement the program consistency model on an architecture consistency model M (possibly adding fences)
  - Prove $M \Longrightarrow S_{com}$
- Or better
  - Find a minimal/weakest H_{com} such that $H_{com} \Longrightarrow S_{com}$
  - $M \Longrightarrow H_{com}$

More work needed

- Specification of parallel/distributed program consistency models (more refined than architecture consistency models, e.g. cuts needed)
- Liveness (beyond non-starvation)
- Collection of certified algorithms for WCM (e.g. transactional memory, databases, etc)
- Static analysis (by abstract interpretation of the analytic semantics parameterized by a WCM)

The End, Thank You