Relaxed Systems Architecture: Instruction Fetching

Ben Simner

University of Cambridge

In collaboration with Shaked Flur, Christopher Pulte, Alasdair Armstrong, Jean Pichon, Luc Maranget[1] and Peter Sewell

[1]INRIA Paris

Motivation

Why?

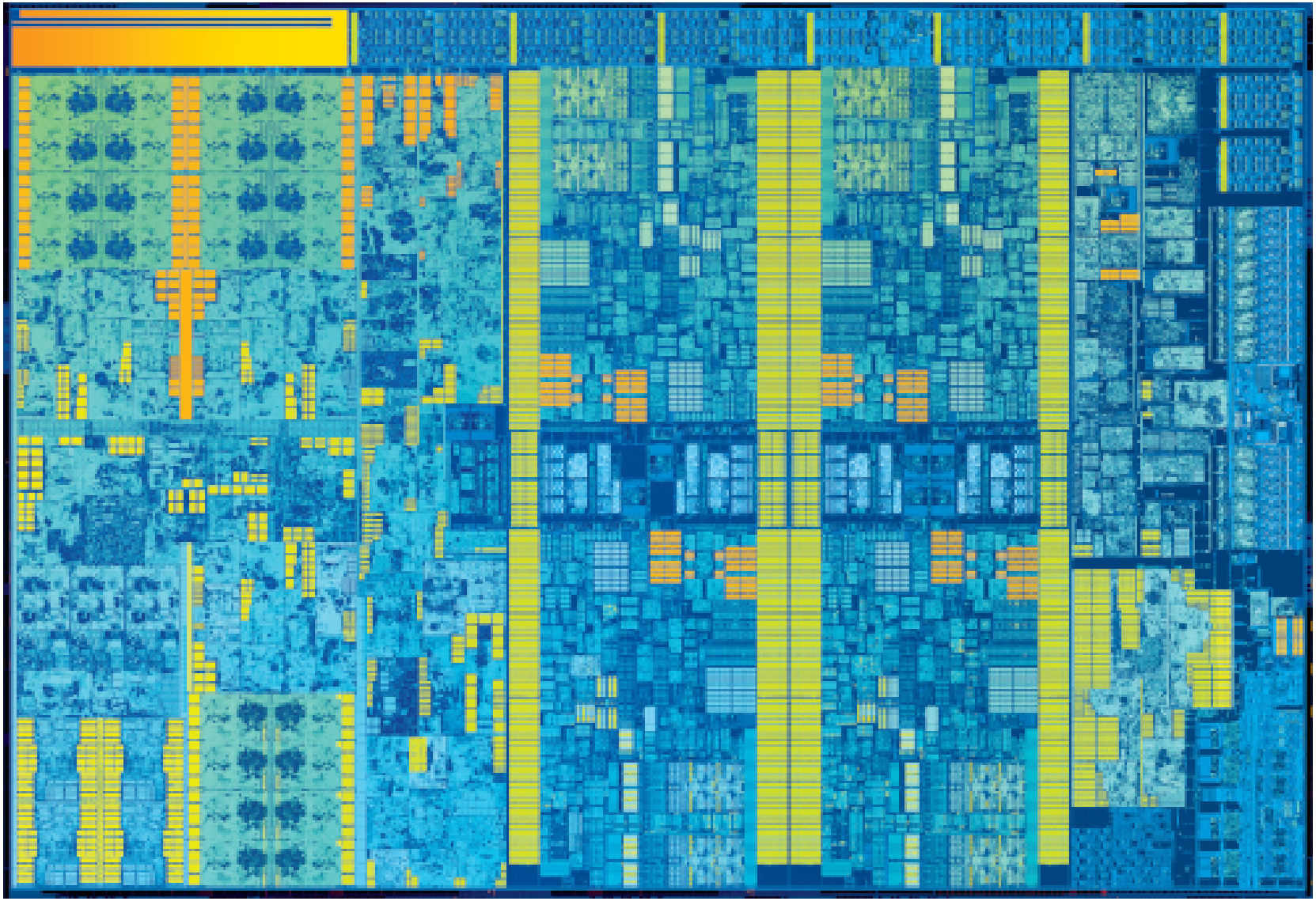
Want to understand: TLBs, Instruction Caches, Interrupts
Want to prove: Operating Systems, JITs, Hypervisors

But first...

Computers are fast...

...but **terrible**!

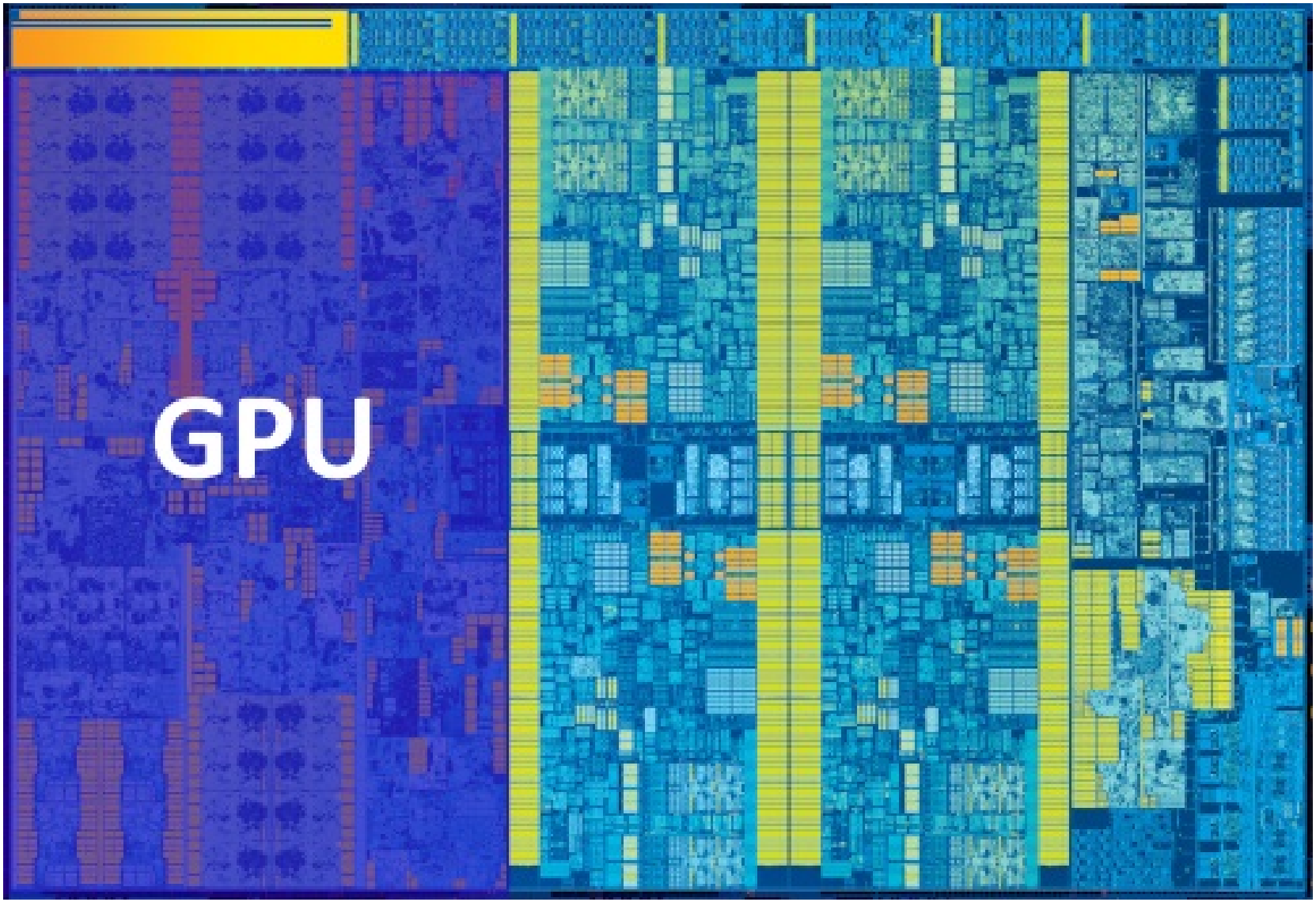
Intel (Skylake) die



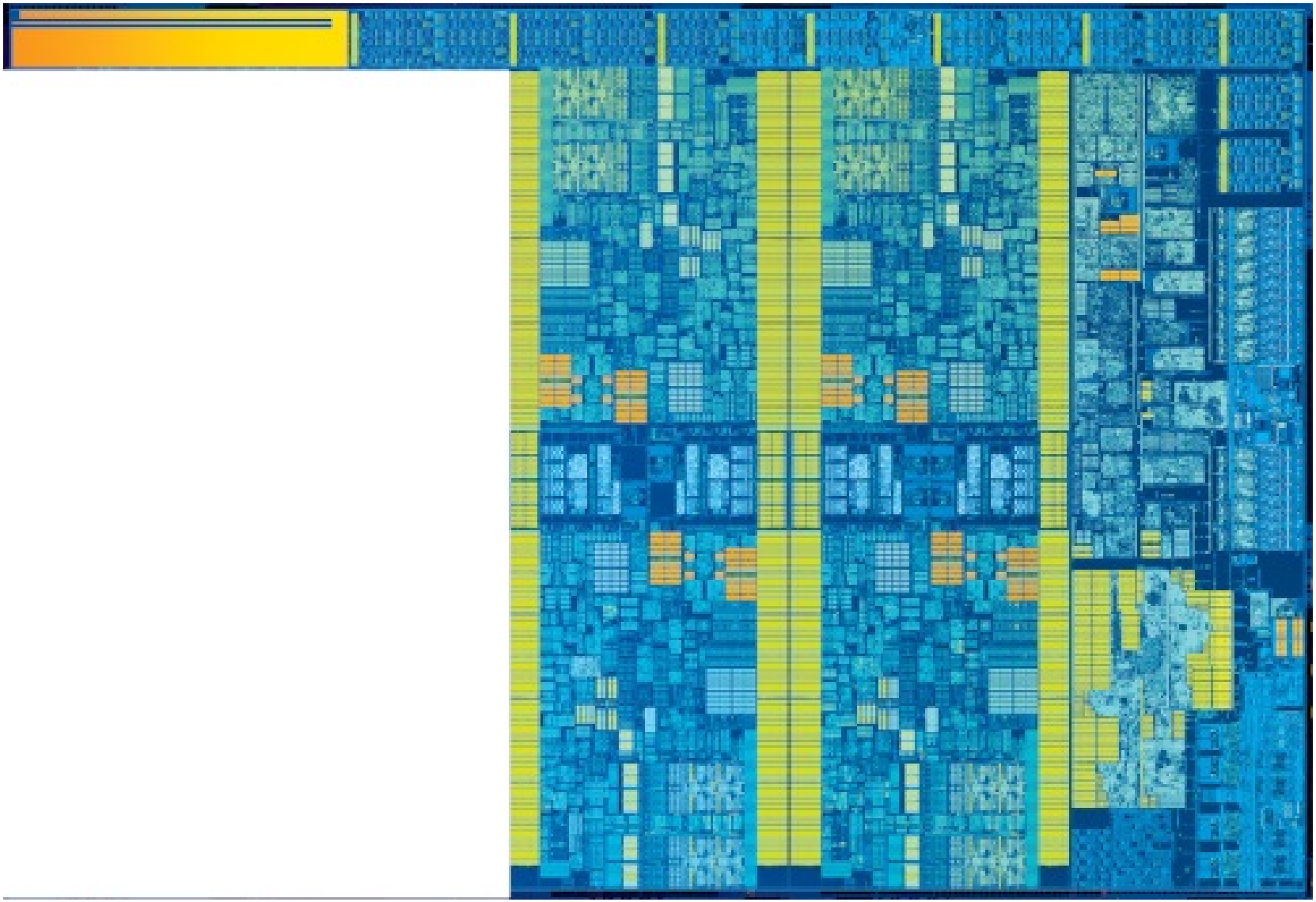
2

[2] Source: https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(client)

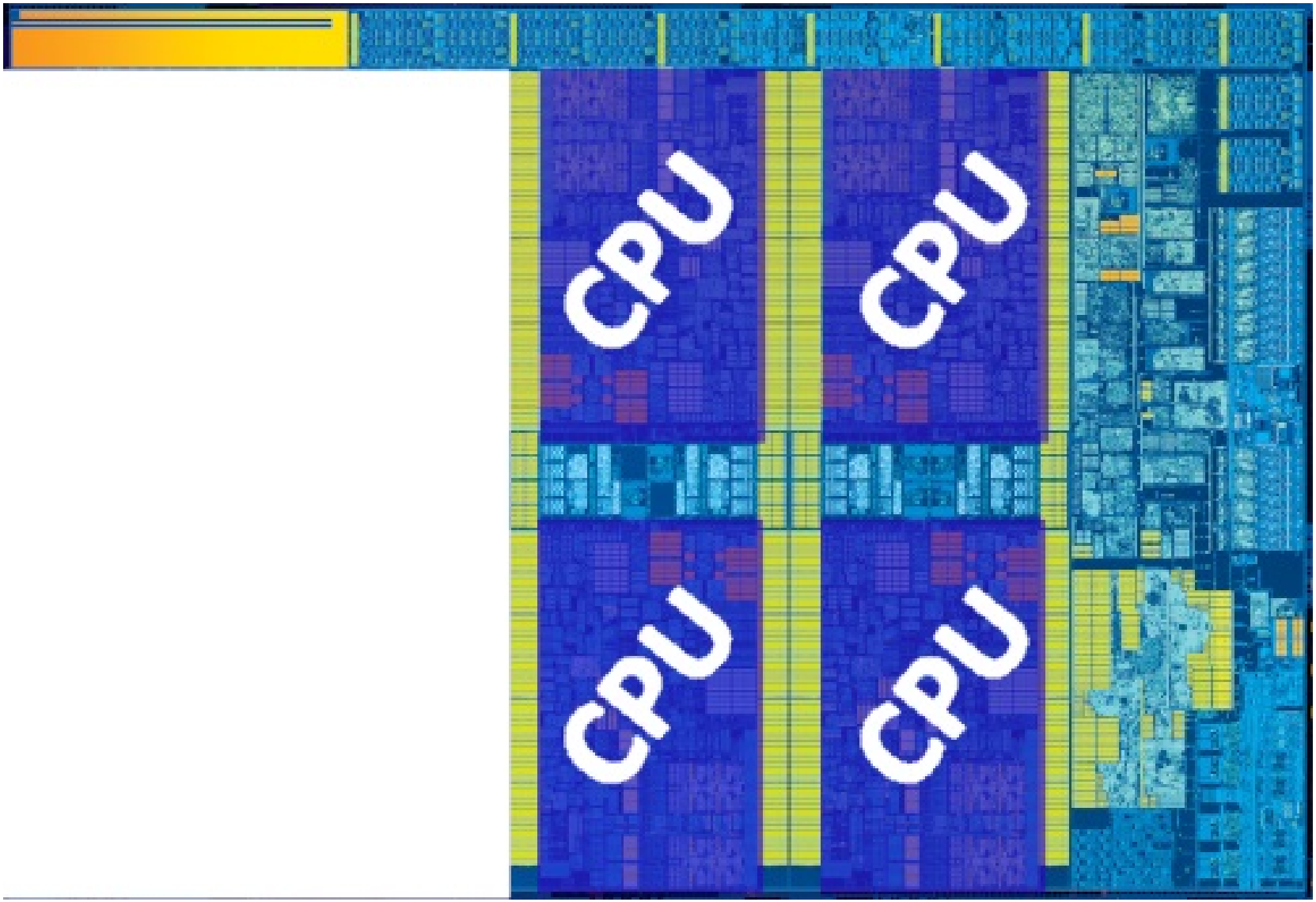
Intel (Skylake) die



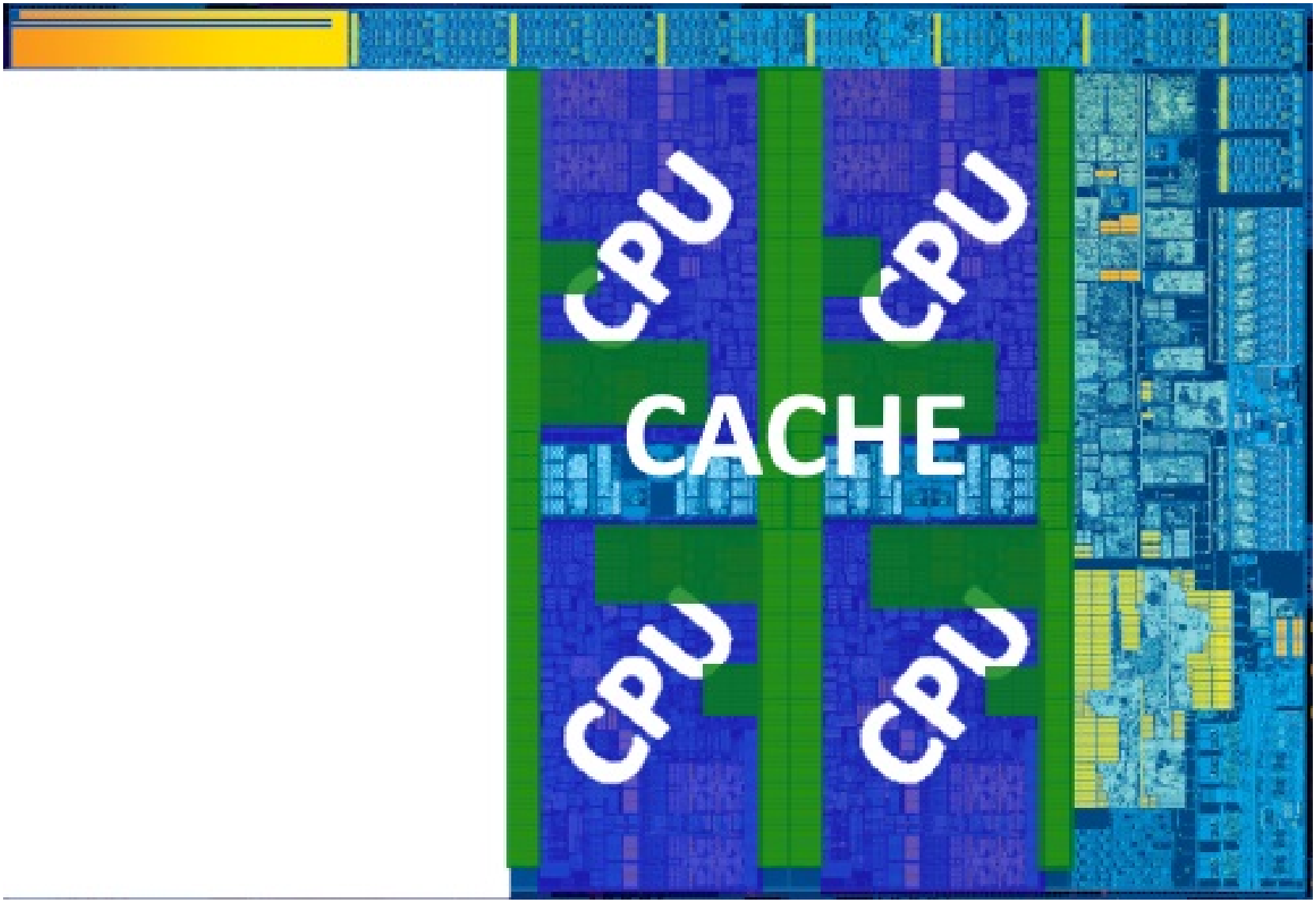
Intel (Skylake) die



Intel (Skylake) die



Intel (Skylake) die



x86: Observable complexity

Dekker's/Peterson's mutual exclusion algorithm (extract)

Thread A	Thread B
`flagA` $\leftarrow$ 1;	`flagB` $\leftarrow$ 1;
`while flagB` {};	`while flagA` {};
`print("A")`	`print("B")`

x86 hardware can execute both prints!

x86: TSO Architecture

Source Code

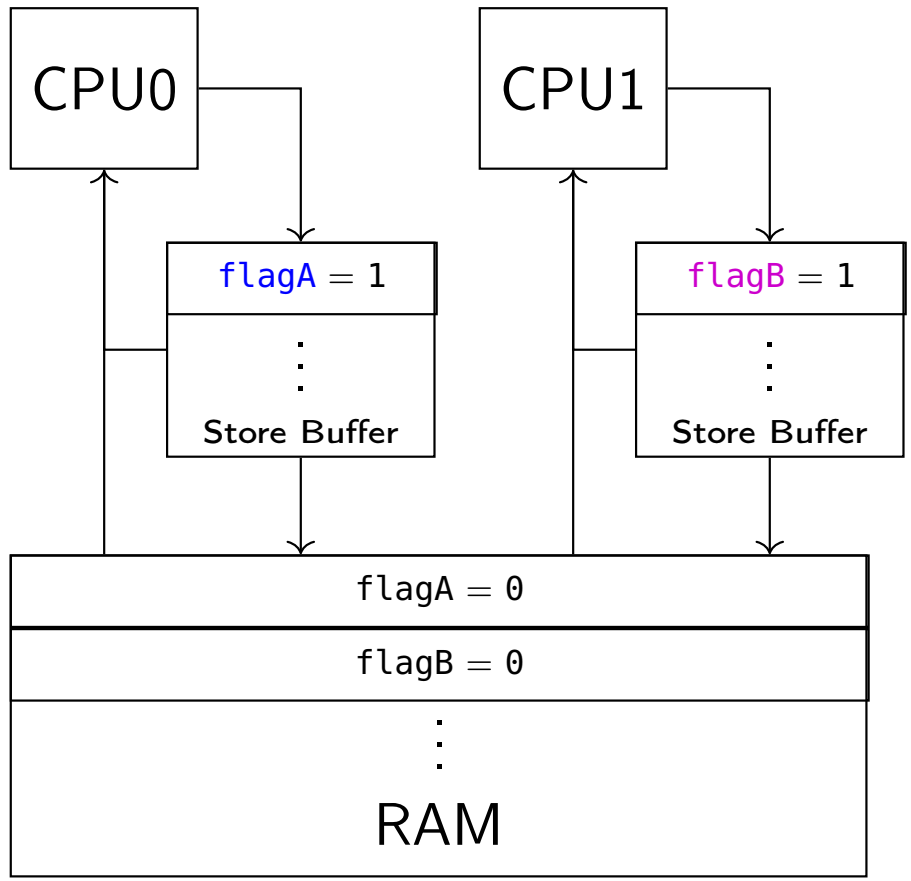
Thread A

```
flagA ← 1;
print(flagB)
```

Thread B

```
flagB ← 1;
print(flagA)
```

Model



State of the Art

Models:

- Abstract Hardware Operational
- Axiomatic-Style

x86-TSO: Operational Semantics

- State = Abstracted Machine State

$$m : \langle M : addr \to value;$$
$$B : tid \to (addr \times value)\ list; \rangle$$

- Structural Operational Semantics

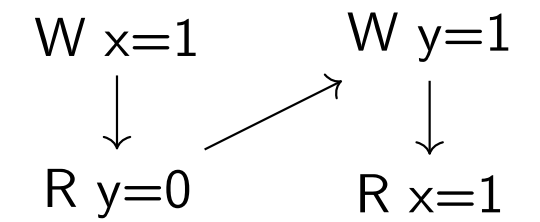
$$\frac{m' = \langle m \text{ with } B := m.B \oplus (t \mapsto ((x, v) : m.B\ t))}{m \xrightarrow{t : Wx = v} m'}\ \textbf{WB}$$

x86-TSO: Axiomatic-Style

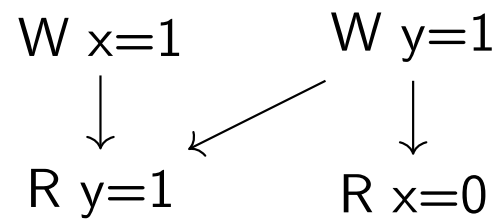
Source Code

```
x ← 1;      ‖  y ← 1;
print(y)    ‖  print(x)
```

Potential Execution #1



Potential Execution #2



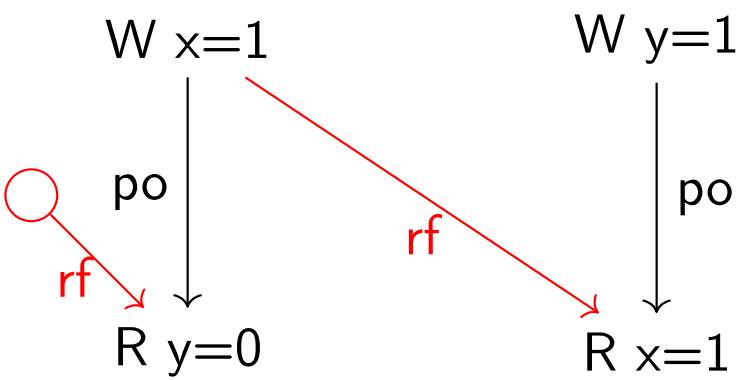
. . .

A Candidate Execution

Pre-execution = Set of Events + Induced Binary Relations (po/data/addr)

Candidate = Pre-execution + Existentially Quantified Relations (co/rf)

Allowed Execution



Definition of a *valid* Candidate ("Axiomatic Model"):

$\mathtt{poWR} = \mathtt{po} \cap (\mathtt{W} \times \mathtt{R})$

$\mathtt{uniproc} = \mathtt{po\text{-}loc} \cup (\mathtt{po} \setminus \mathtt{poWR})$

$\mathtt{fr} = \mathtt{rf}^{-1} ; \mathtt{co}$

$\mathtt{tso} = \mathtt{rf} \cup \mathtt{fr} \cup \mathtt{co}$

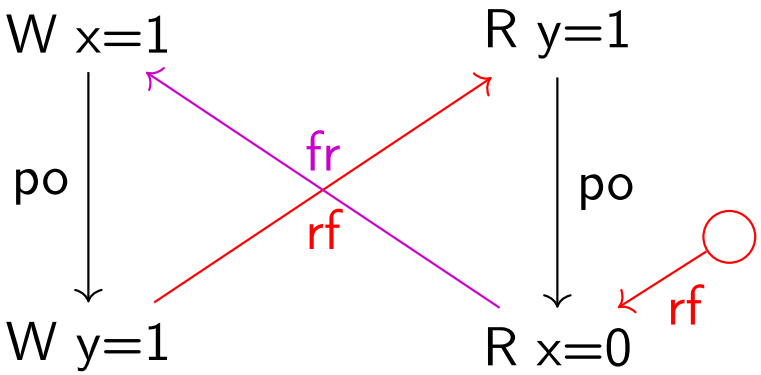
axiom : $\mathtt{acyclic}\,(\mathtt{uniproc} \cup \mathtt{tso})$

po = Program-Order

rf = Reads-From

TSO: Forbidden Execution

Forbidden Execution



Axiomatic Model:

$\texttt{poWR} = \texttt{po} \cap (\texttt{W} \times \texttt{R})$

$\texttt{uniproc} = \texttt{po-loc} \cup (\texttt{po} \setminus \texttt{poWR})$

$\texttt{fr} = \texttt{rf}^{-1} ; \texttt{co}$

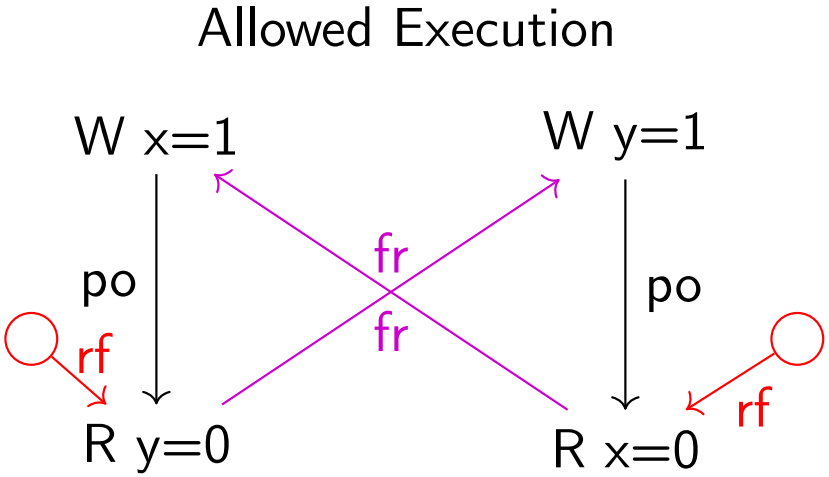
$\texttt{tso} = \texttt{rf} \cup \texttt{fr} \cup \texttt{co}$

axiom : $\texttt{acyclic}\,(\texttt{uniproc} \cup \texttt{tso})$

po = Program-Order

rf = Reads-From

TSO: Allowed Execution



Axiomatic Model:

$\mathtt{poWR} = \mathtt{po} \cap (\mathtt{W} \times \mathtt{R})$

$\mathtt{uniproc} = \mathtt{po\text{-}loc} \cup (\mathtt{po} \setminus \mathtt{poWR})$

$\mathtt{fr} = \mathtt{rf}^{-1} ; \mathtt{co}$

$\mathtt{tso} = \mathtt{rf} \cup \mathtt{fr} \cup \mathtt{co}$

axiom : $\mathtt{acyclic}\,(\mathtt{uniproc} \cup \mathtt{tso})$

po = Program-Order

rf = Reads-From

fr = From-Reads

"user-mode" concurrency

Much work not covered here:

- Fences
- Atomics
- Mixed-size
- Multi-copy atomicity
- Other Architectures: IBM Power, Arm, RISC-V

Systems Architecture Semantics

Instruction Fetch

↘

ESOP2020

Exceptions and Interrupts

↘

with Ohad Kammar

Pagetables and TLBs

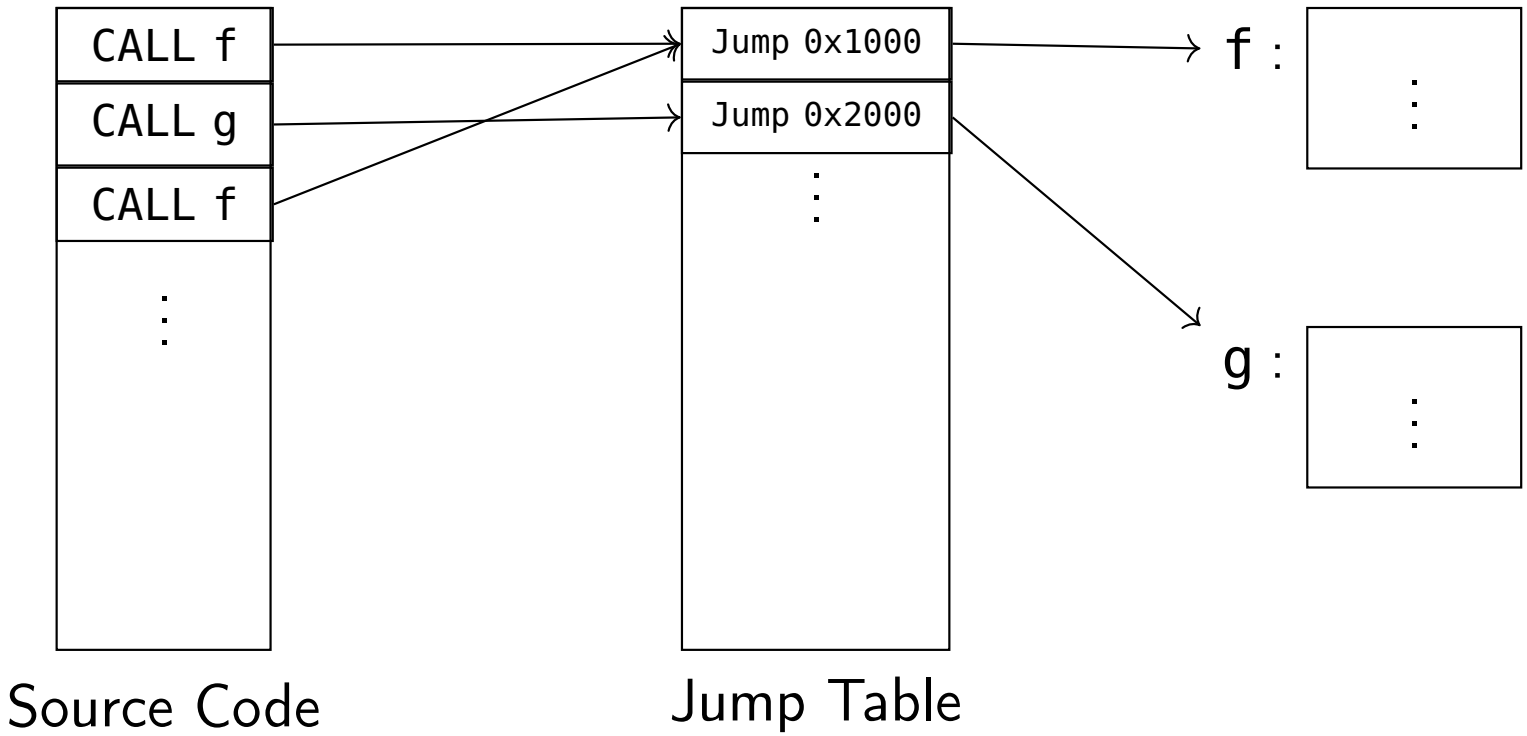
Devices and NVME

↘

Future Work . . .

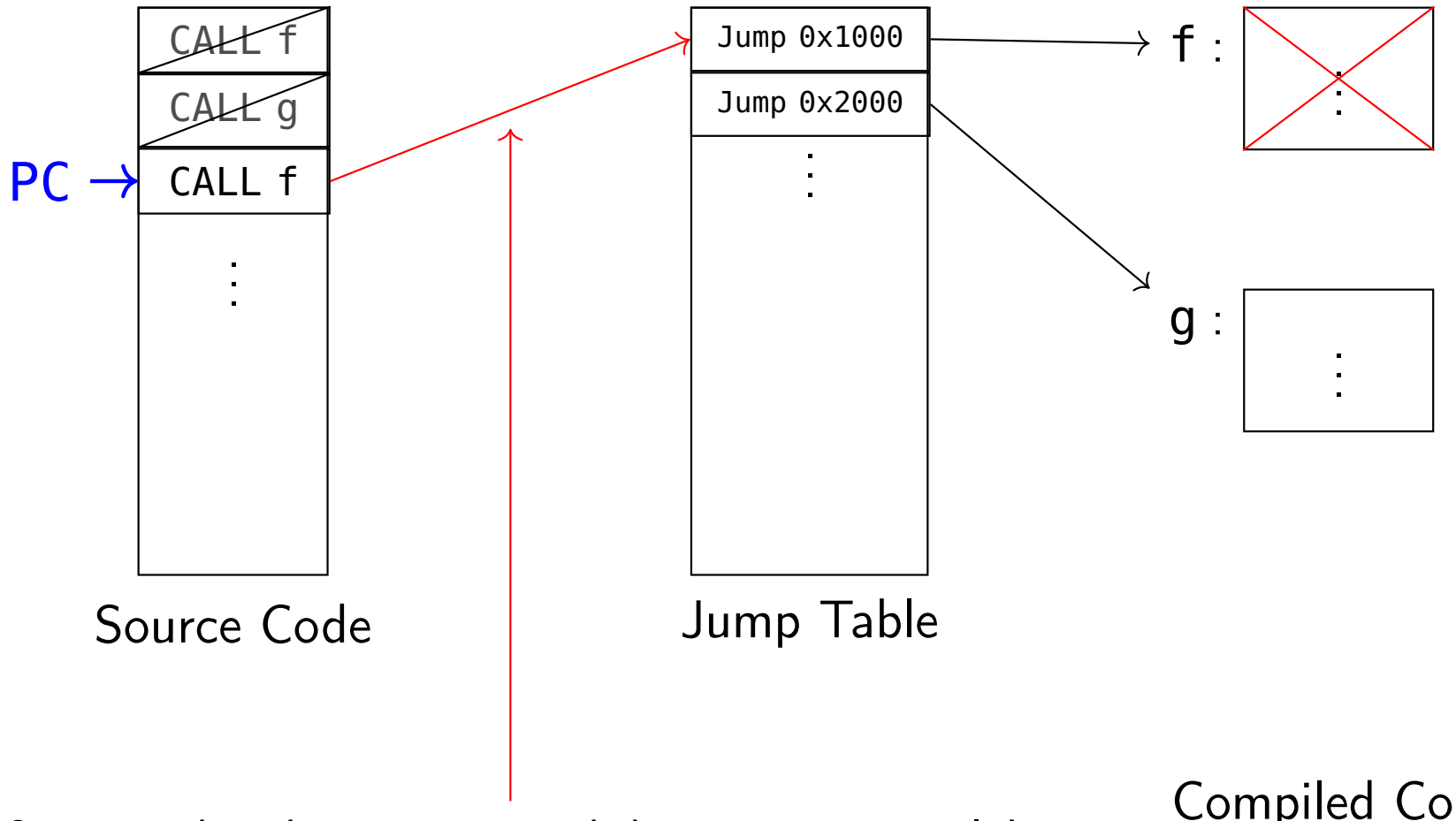
JITs

Just-In-Time Compilation



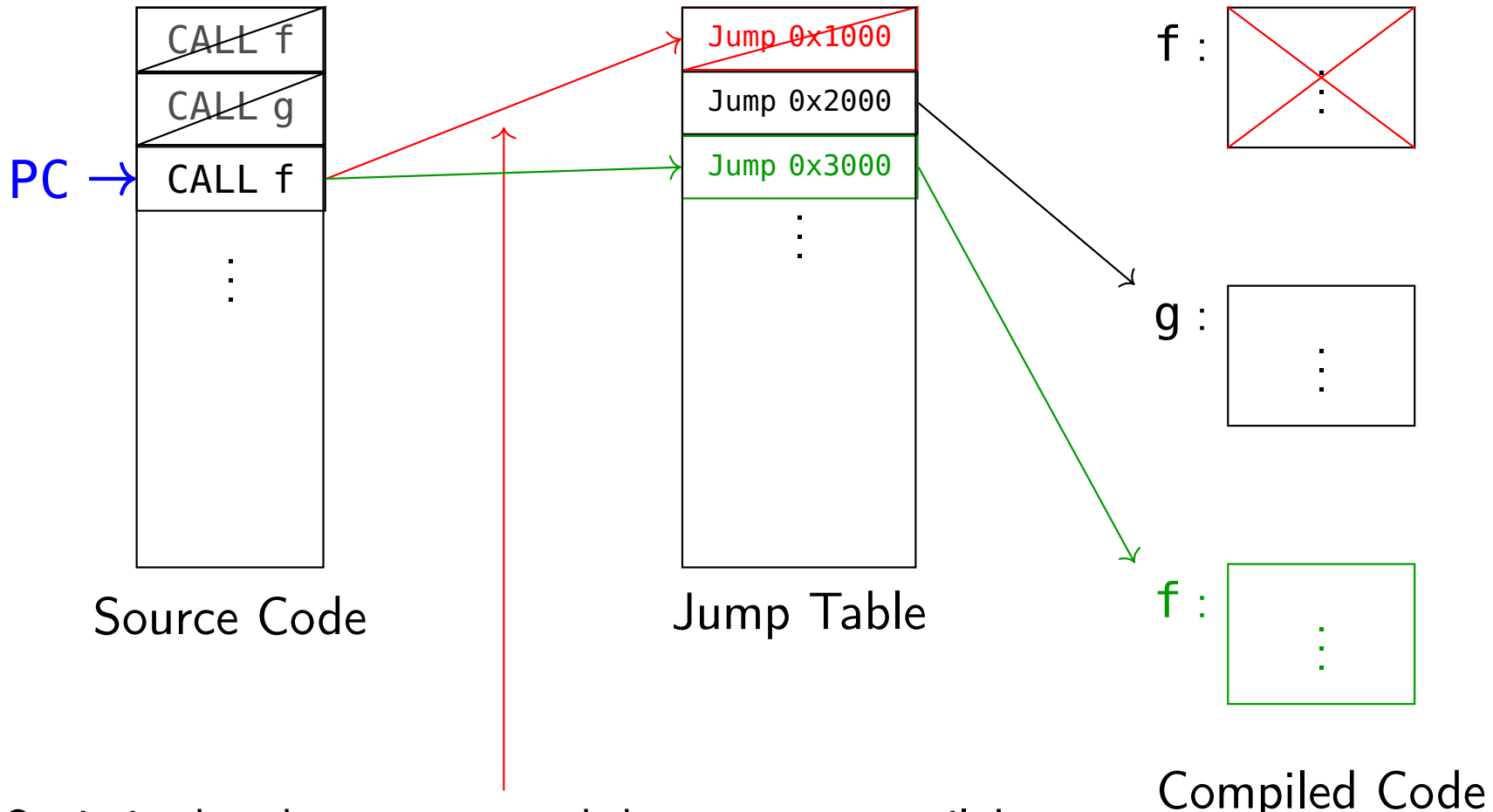
JITs

JIT: de-opt after executing g



JITs

JIT: re-compile



ARMv8: How to safely modify code?

The write is complete for the shareability domain.

If software requires coherency between instruction execution and memory, it must manage this coherency us
*Context synchronization event*s and cache maintenance instructions. The following code sequence can be us
allow a PE to execute code that the same PE has written.

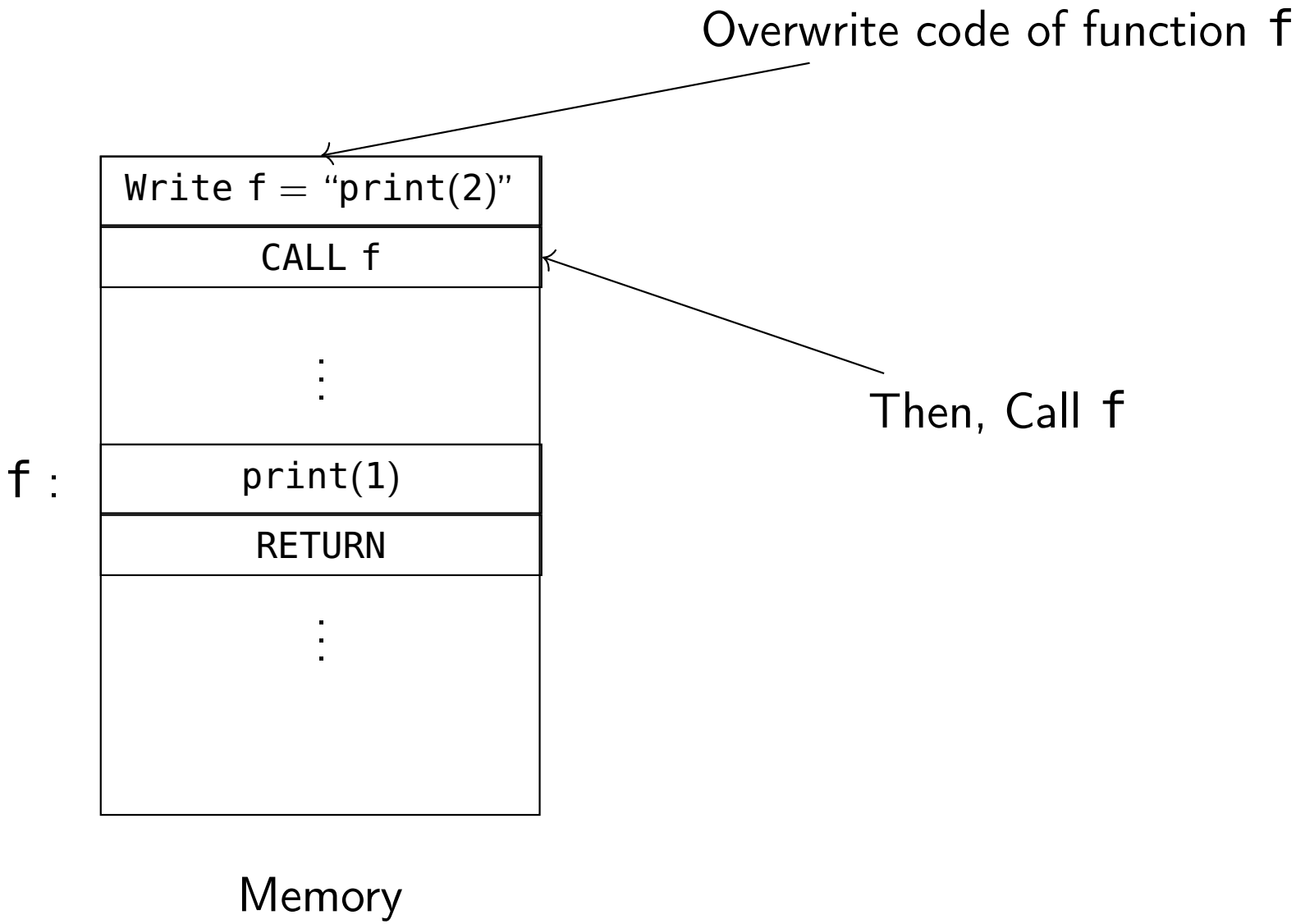
```
; Coherency example for data and instruction accesses within the same Inner Shareable domain.
; Enter this code with <Wt> containing a new 32-bit instruction,
; to be held in Cacheable space at a location pointed to by Xn.
        STR Wt, [Xn]
        DC CVAU, Xn          ; Clean data cache by VA to point of unification (PoU)
        DSB ISH              ; Ensure visibility of the data cleaned from cache
        IC IVAU, Xn          ; Invalidate instruction cache by VA to PoU
        DSB ISH              ; Ensure completion of the invalidations
        ISB                  ; Synchronize the fetched instruction stream
```


The AArch64 Application Level Memory
B2.4 Caches and memory hie

RISC-V/x86/Power: How to?

Similar for IBM Power
Much easier on x86
RISC-V not decided yet ...
Focus on **ARMv8-A** for rest of talk...

An Instruction Fetching Test



Real A64 Assembly

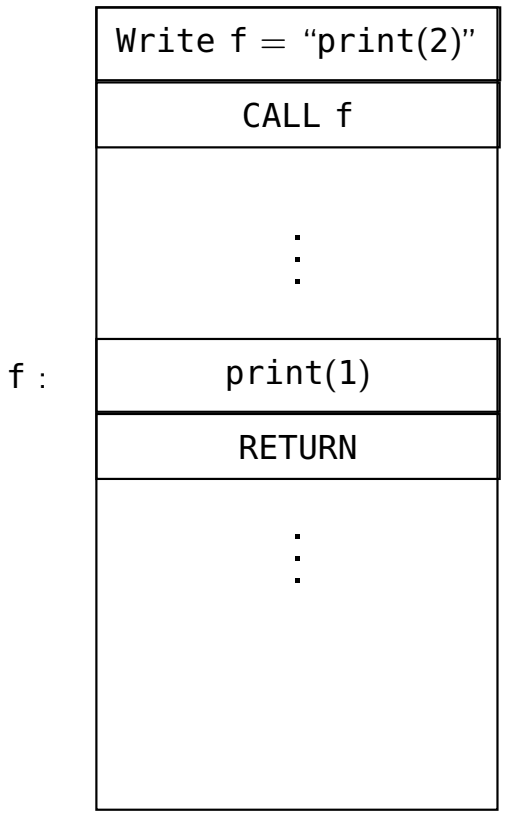
Initial state: `0:W0="B l1", 0:X1=f`	
Thread 0	**f**

`STR W0,[X1]`
`BL f`
	`f:  B l0`
	`l1: MOV X0,#2`
	`RET`
	`l0: MOV X0,#1`
	`RET`

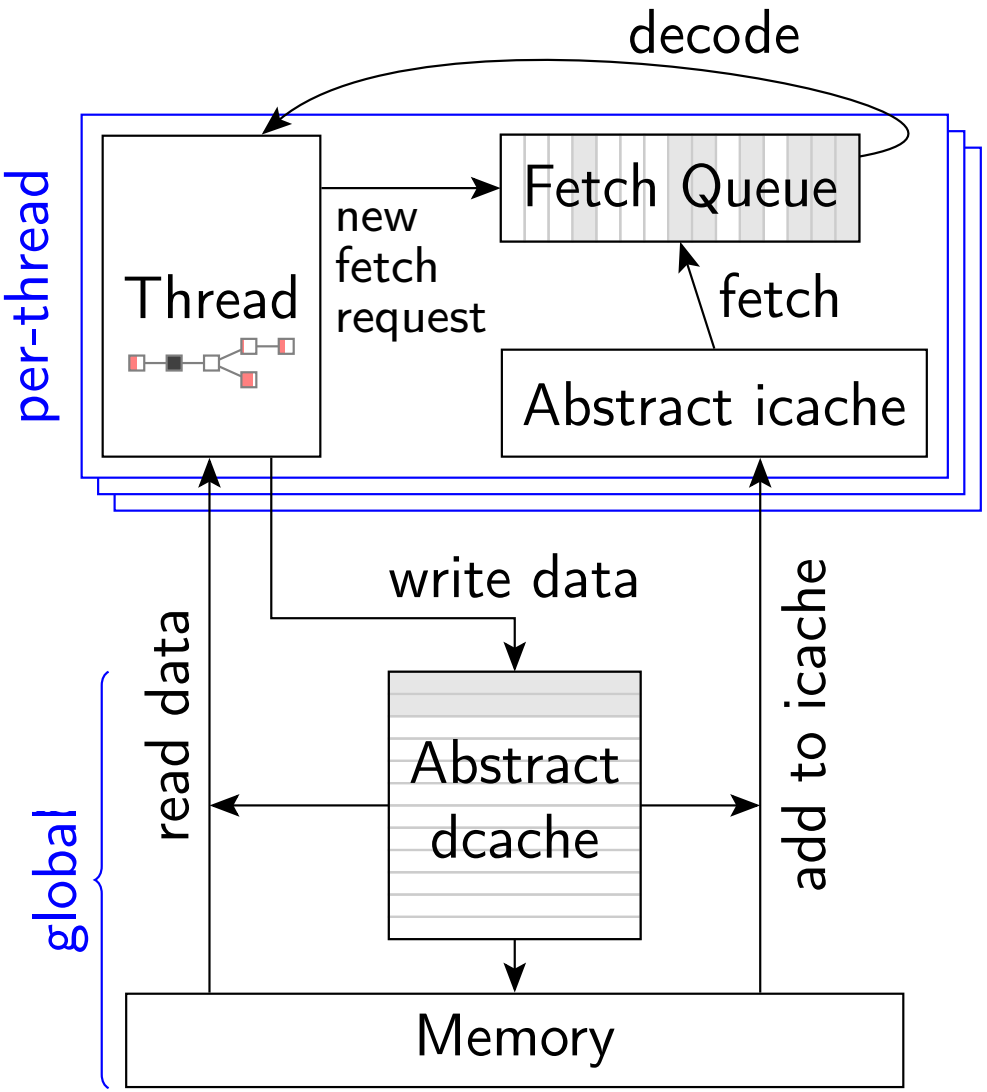
Allowed: `1:X0=1`	

Relaxed Result Observed in ~99% of experimental runs on multiple devices.

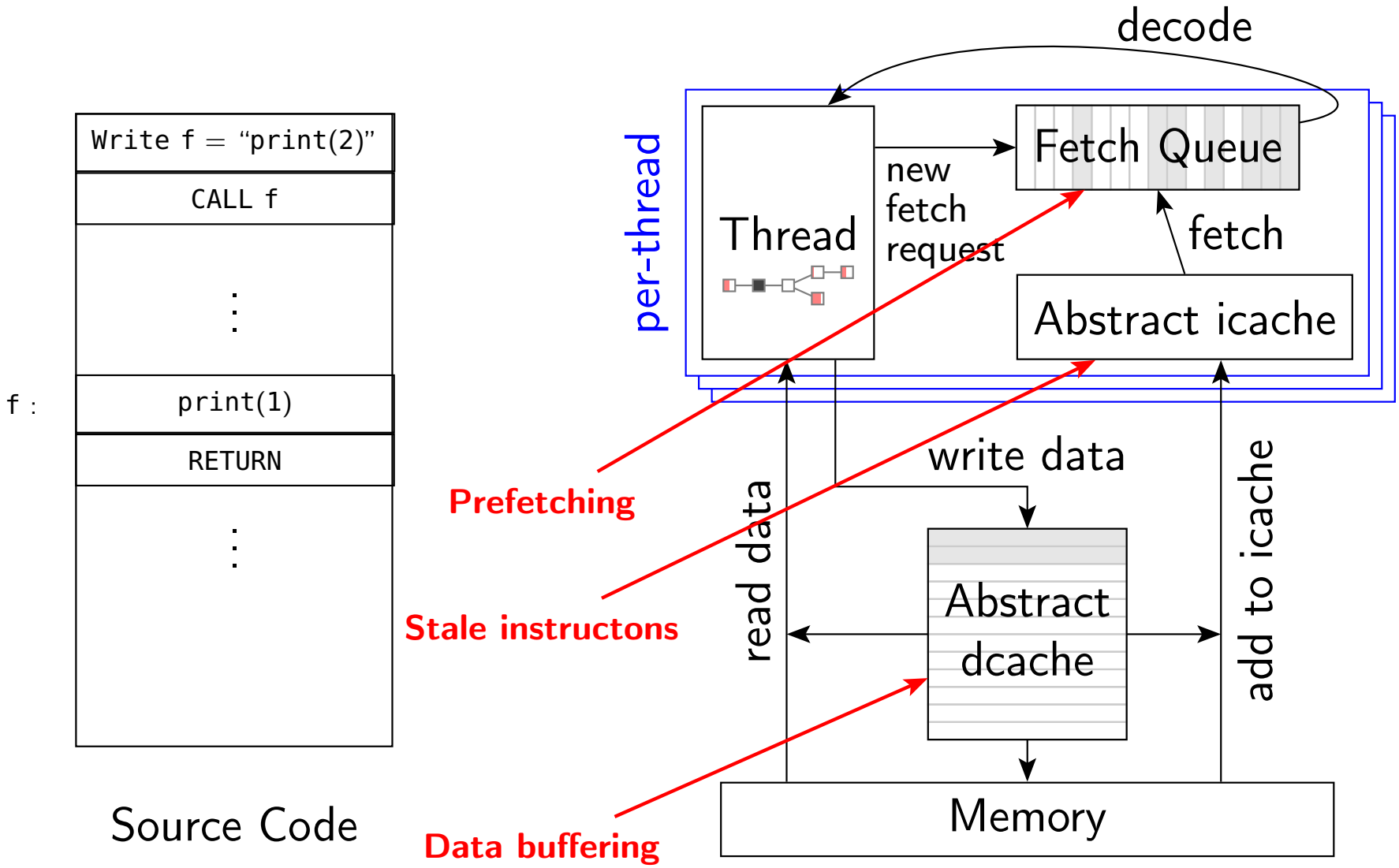
An Architectural Model!



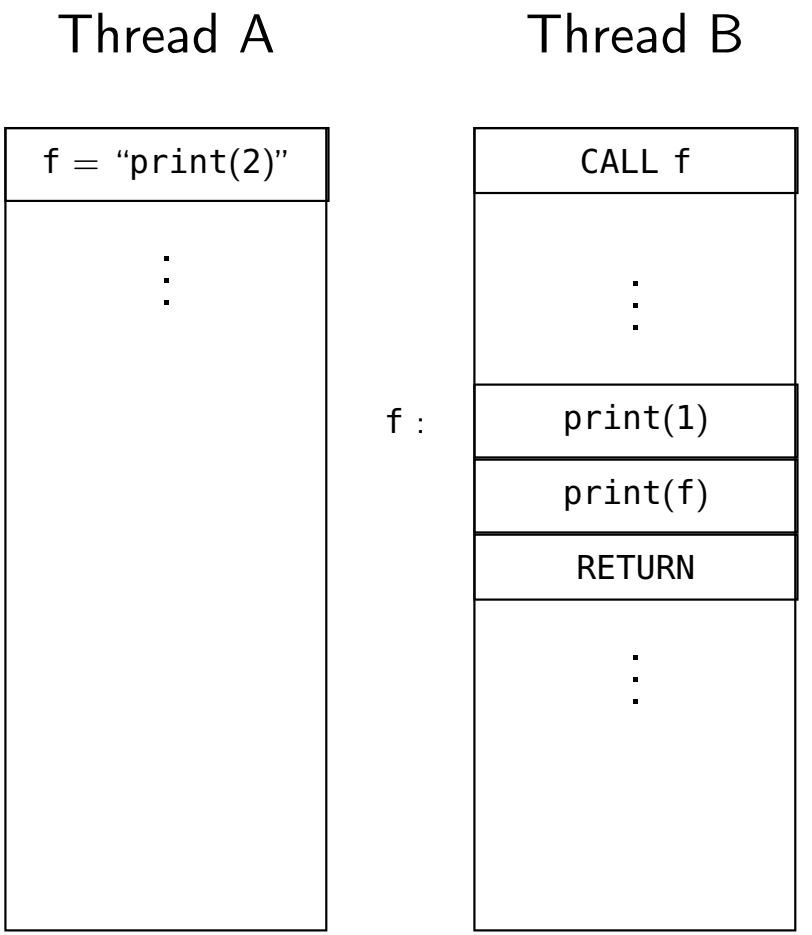
Source Code



An Architectural Model!



Unexpected Coherence!



If `f` **executes** `print(2)`
Then `print(f)` ***must*** **print the updated memory** (2).

Real A64 Assembly

Initial state: `0:W0="B l1", 0:X1=f, 1:X2=f`		
Thread 0	**Thread 1**	**f**

`STR W0,[X1]`	`BL f`
	`LDR X1,[X2]`
		`f: B l0`
		`l1: MOV X0,#2`
		`RET`
		`l0: MOV X0,#1`
		`RET`

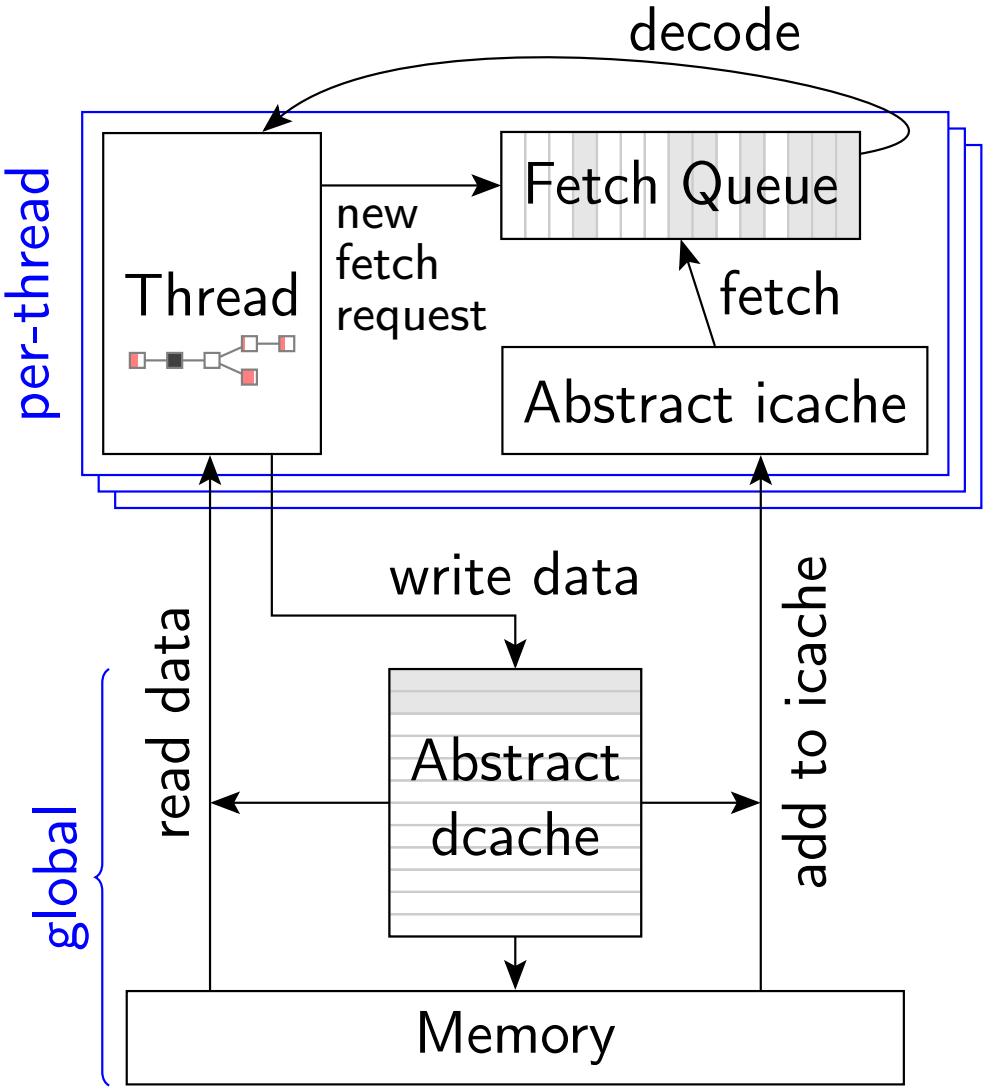
Forbidden: `1:X0=2, 1:X1="B l0"`		

Other Phenomena

Not Mentioned Here:

- (In)coherence
- Multiple images in I-cache
- Multiple images in D-cache(s)
- Direct Data Intervention
- Speculating cache maintenance
- O/S Migration
- and others . . .

Operational Model

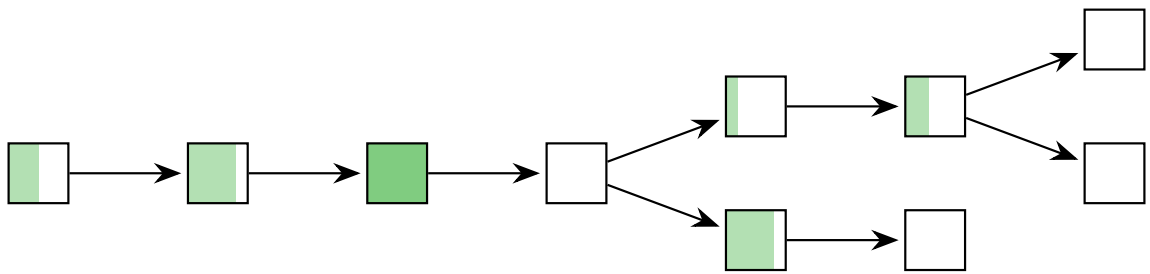


Operational State

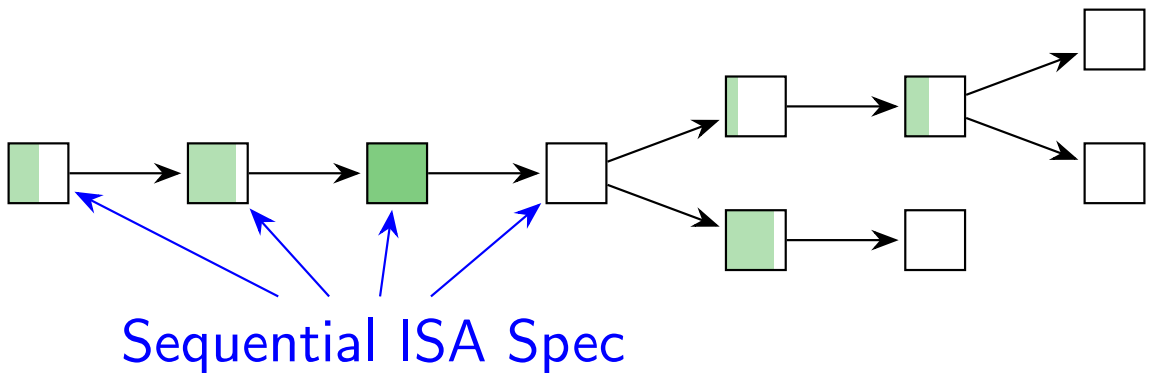
$$m : \langle ts : tid \mapsto \texttt{instruction\_tree}$$
$$ss : \texttt{storage\_subsystem} \rangle$$

$$storage\_subsystem : \langle mem : \texttt{write}\ list$$
$$icache : tid \mapsto \texttt{write}\ set$$
$$dcache : \texttt{write}\ list$$
$$\ldots \rangle$$

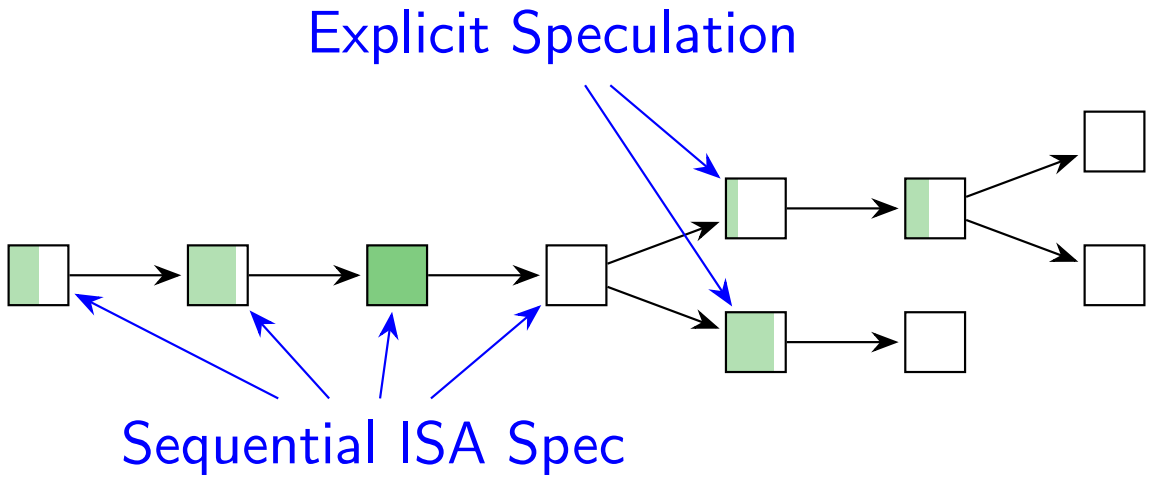
Thread State



Thread State



Thread State



Operational: Transitions

Transitions:

- Step ISA Spec
- Memory Read/Write
- ...
- Fetch Request
- Fetch Instruction (from icache)
- Decode Instruction
- ...
- Update Instruction Cache
- Flow Writes into Memory
- Reset Instruction

New!

*exact names my vary

Operational Rule (prose)

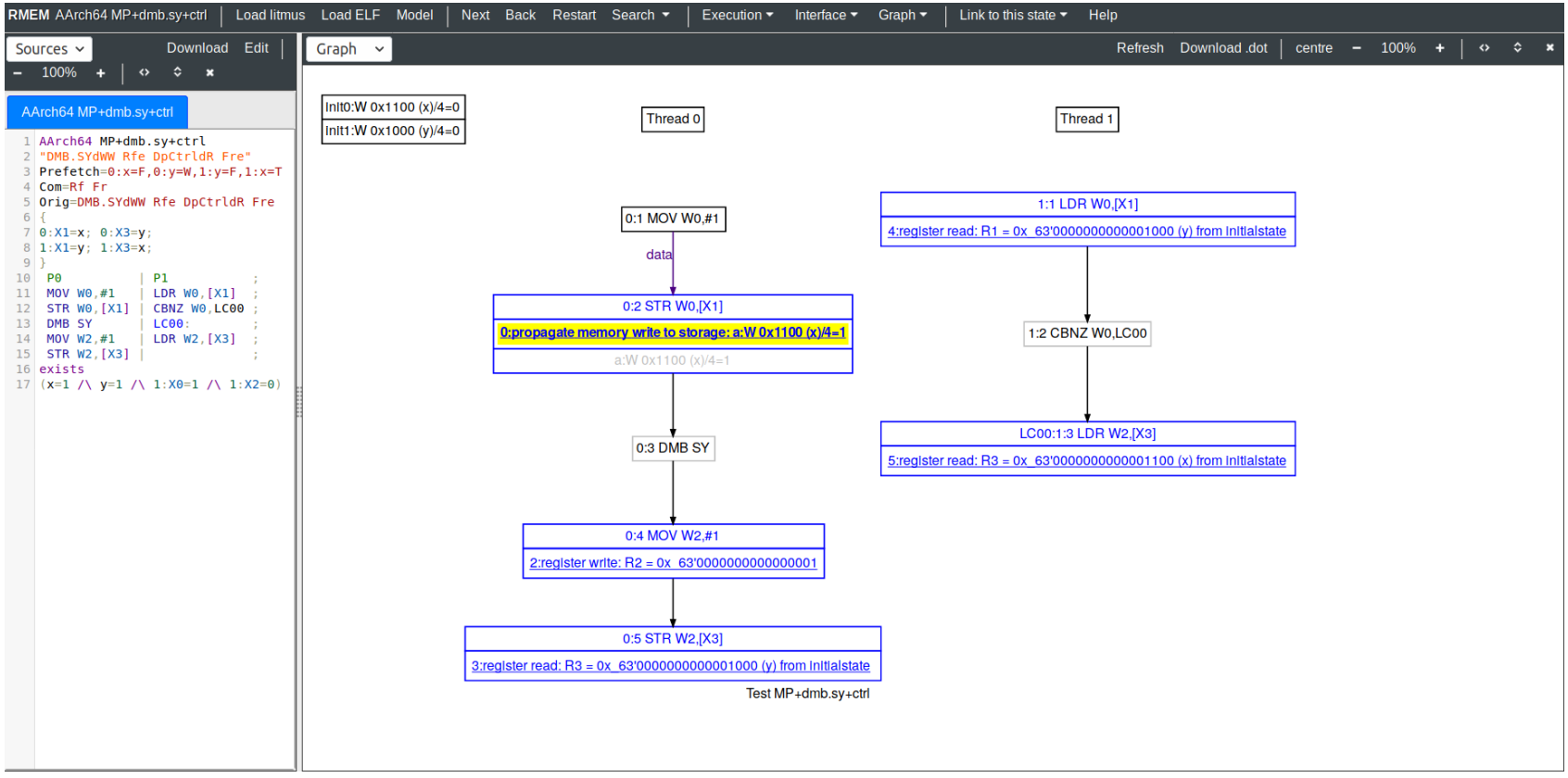
Flow Writes into Memory An instruction *i* in the state Perform_DC*(address, state_cont)* can complete if all po-previous `DMB ISH` and `DSB ISH` instructions have finished. Action:

1. For the most recent writes *ws* which are in the same data cache line of minimum size in the abstract data cache as *address*, update the memory with *ws*;
2. Remove all those writes from the abstract data cache.
3. Set the state of *i* to Plain*(state_cont)*.

Operational Rule (lem)

```
let flat_propagate_dc params state _cmr addr =
    (* remove all to that cacheline from buffer *)
    let (overlapping, fetch_buf) =
      List.partition
        (write_overlaps_with_addr (cache_line_fp addr))
        state.flat_ss_fetch_buf
    in
    (* flow the overlapping writes into memory *)
    List.foldr
      (fun write state ->
        flat_write_to_memory params state write)
      (<| state with flat_ss_fetch_buf = fetch_buf |>)
      overlapping
```

RMEM



https://www.cl.cam.ac.uk/~pes20/rmem/

Axiomatic-Style Model

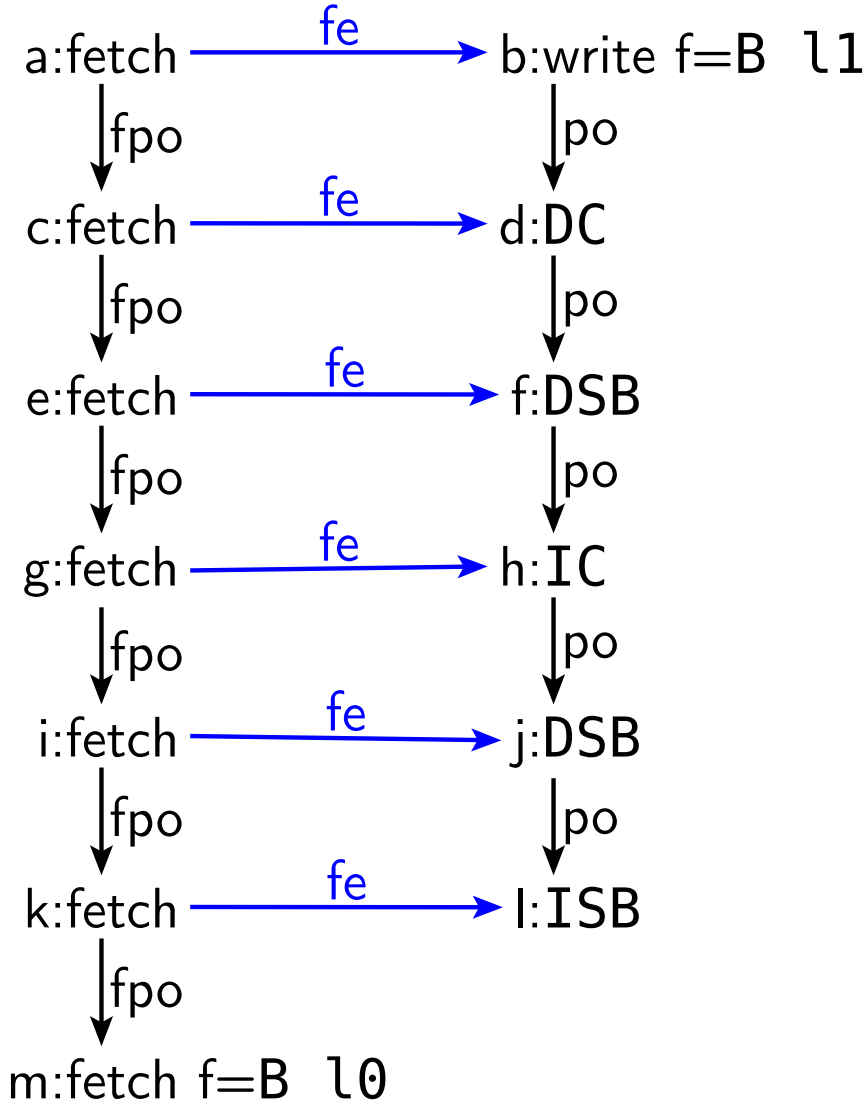
```
let iseq = [W];(wco&scl);[DC];
               (wco&scl);[IC]

(* Observed-by *)
let obs = rfe | fr | wco
        | irf | (ifr;iseq)

(* Fetch-ordered-before *)
let fob = [IF]; fpo; [IF]
  | [IF]; fe
  | [ISB]; fe^-1; fpo

(* Dependency-ordered-before *)
let dob = addr | data
  | ctrl; [W]
  | (ctrl | (addr; po)); [ISB]
  | addr; po; [W]
  | (addr | data); rfi

(* Atomic-ordered-before *)
let aob = rmw
  | [range(rmw)]; rfi; [A|Q]

(* Barrier-ordered-before *)
let bob = [R|W]; po; [dmb.sy]
  | [dmb.sy]; po; [R|W]
  | [L]; po; [A]
  | [R]; po; [dmb.ld]
  | [dmb.ld]; po; [R|W]
  | [A|Q]; po; [R|W]
  | [W]; po; [dmb.st]
  | [dmb.st]; po; [W]
  | [R|W]; po; [L]
  | [R|W|F|DC|IC]; po; [dsb.ish]
  | [dsb.ish]; po; [R|W|F|DC|IC]
  | [dmb.sy]; po; [DC]

(* Cache-op-ordered-before *)
let cob = [R|W]; (po&scl); [DC]
  | [DC]; (po&scl); [DC]

(* Ordered-before *)
let ob = obs|fob|dob|aob|bob|cob

(* Internal visibility requirement *)
acyclic (po-loc|fr|co|rf) as internal

(* External visibility requirement *)
acyclic ob as external

(* Atomic *)
empty rmw & (fre; coe) as atomic

(* Constrained unpredictable *)
let cff = ([W];loc;[IF]) \
            ob+^-1 \ (co;iseq;ob+)
cff_bad cff ≡ CU
```

Axiomatic-Style Model

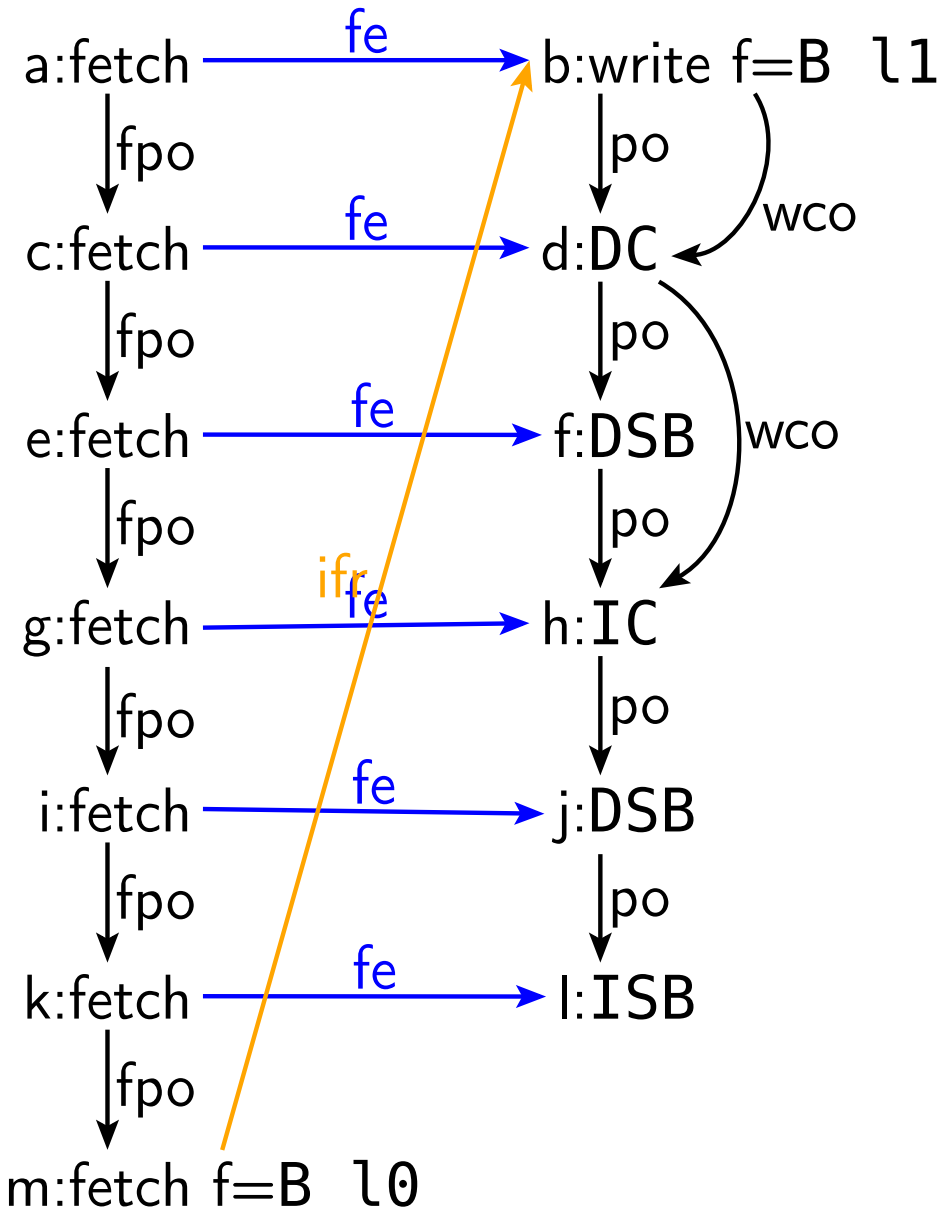
```
let iseq = [W];(wco&scl);[DC];
               (wco&scl);[IC]

(* Observed-by *)
let obs = rfe | fr | wco
        | irf | (ifr;iseq)

(* Fetch-ordered-before *)
let fob = [IF]; fpo; [IF]
  | [IF]; fe
  | [ISB]; fe^-1; fpo

(* Dependency-ordered-before *)
let dob = addr | data
  | ctrl; [W]
  | (ctrl | (addr; po)); [ISB]
  | addr; po; [W]
  | (addr | data); rfi

(* Atomic-ordered-before *)
let aob = rmw
  | [range(rmw)]; rfi; [A|Q]

(* Barrier-ordered-before *)
let bob = [R|W]; po; [dmb.sy]
  | [dmb.sy]; po; [R|W]
  | [L]; po; [A]
  | [R]; po; [dmb.ld]
  | [dmb.ld]; po; [R|W]
  | [A|Q]; po; [R|W]
  | [W]; po; [dmb.st]
  | [dmb.st]; po; [W]
  | [R|W]; po; [L]
  | [R|W|F|DC|IC]; po; [dsb.ish]
  | [dsb.ish]; po; [R|W|F|DC|IC]
  | [dmb.sy]; po; [DC]

(* Cache-op-ordered-before *)
let cob = [R|W]; (po&scl); [DC]
  | [DC]; (po&scl); [DC]

(* Ordered-before *)
let ob = obs|fob|dob|aob|bob|cob

(* Internal visibility requirement *)
acyclic (po-loc|fr|co|rf) as internal

(* External visibility requirement *)
acyclic ob as external

(* Atomic *)
empty rmw & (fre; coe) as atomic

(* Constrained unpredictable *)
let cff = ([W];loc;[IF]) \
            ob+^-1 \ (co;iseq;ob+)
cff_bad cff ≡ CU
```

Axiomatic ifetch: an example

Initial state: W0="B l1" X1=f
STR W0,[X1] // (b) **DC** CVAU,X1 // (d) DSB ISH **IC** IVAU,X1 // (h) DSB ISH **ISB** // (l) BL f // (m)
Forbidden: X0=1

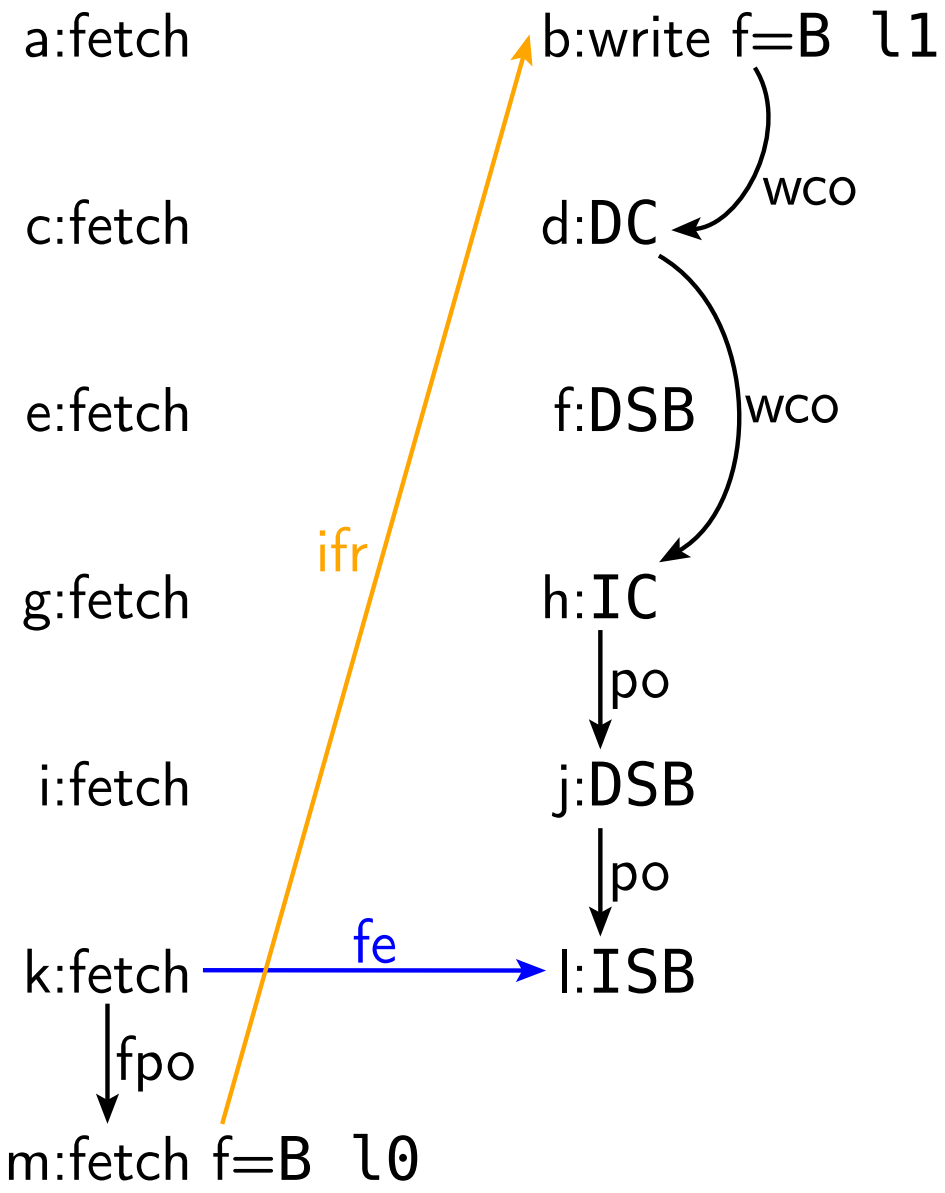


A Forbidden Instuction Fetch



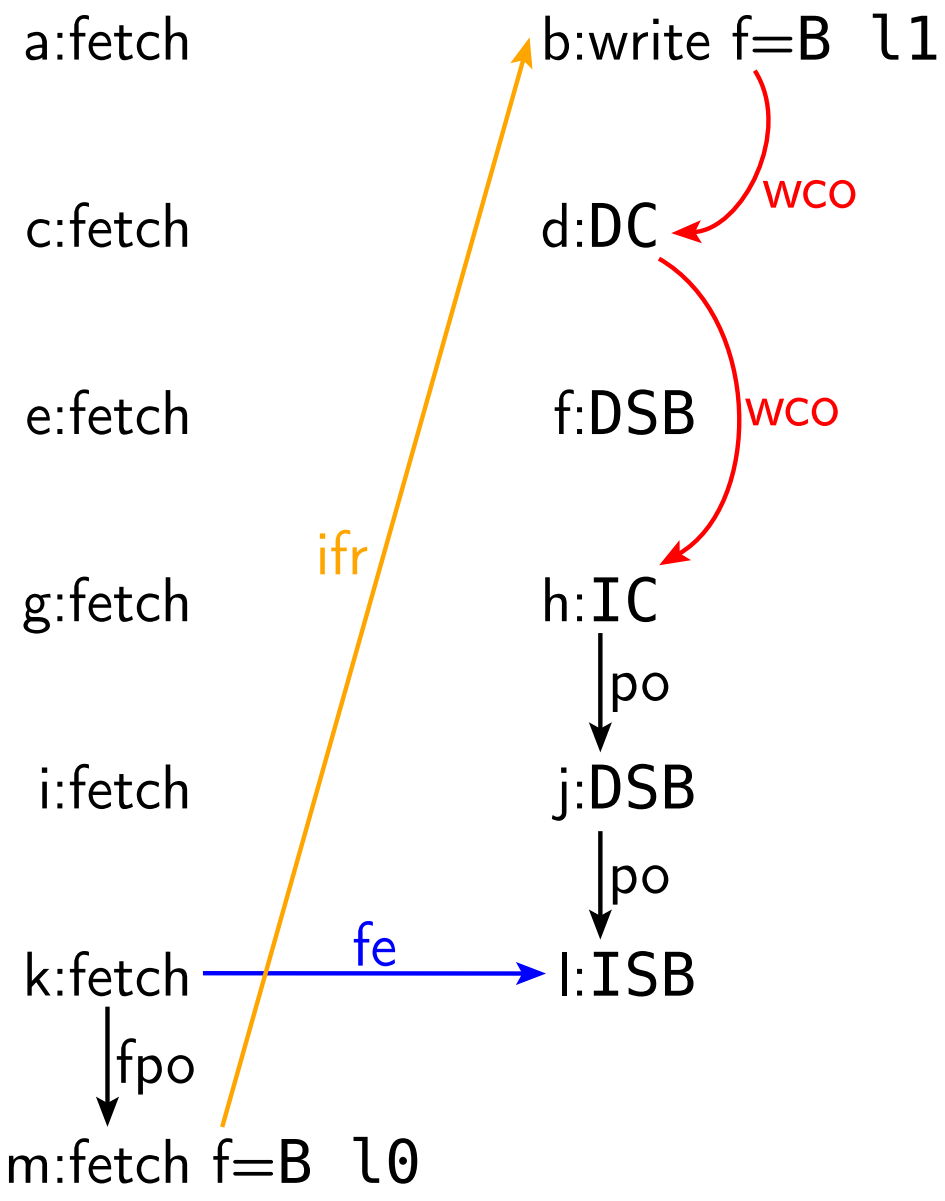

```
let iseq = [W];(wco&scl);[DC];
                (wco&scl);[IC]

(* Observed-by *)
let obs = irf | (ifr;iseq)

(* Fetch-ordered-before *)
let fob = [IF]; fpo; [IF]
    | [IF]; fe
    | [ISB]; fe^-1; fpo

(* Barrier-ordered-before *)
let bob = ...
    | [R|W|F|DC|IC]; po; [dsb.ish]
    | [dsb.ish]; po; [R|W|F|DC|IC]

(* Ordered-before *)
let ob = obs | fob | bob

(* External visibility requirement *)
acyclic ob
```

A Forbidden Instuction Fetch



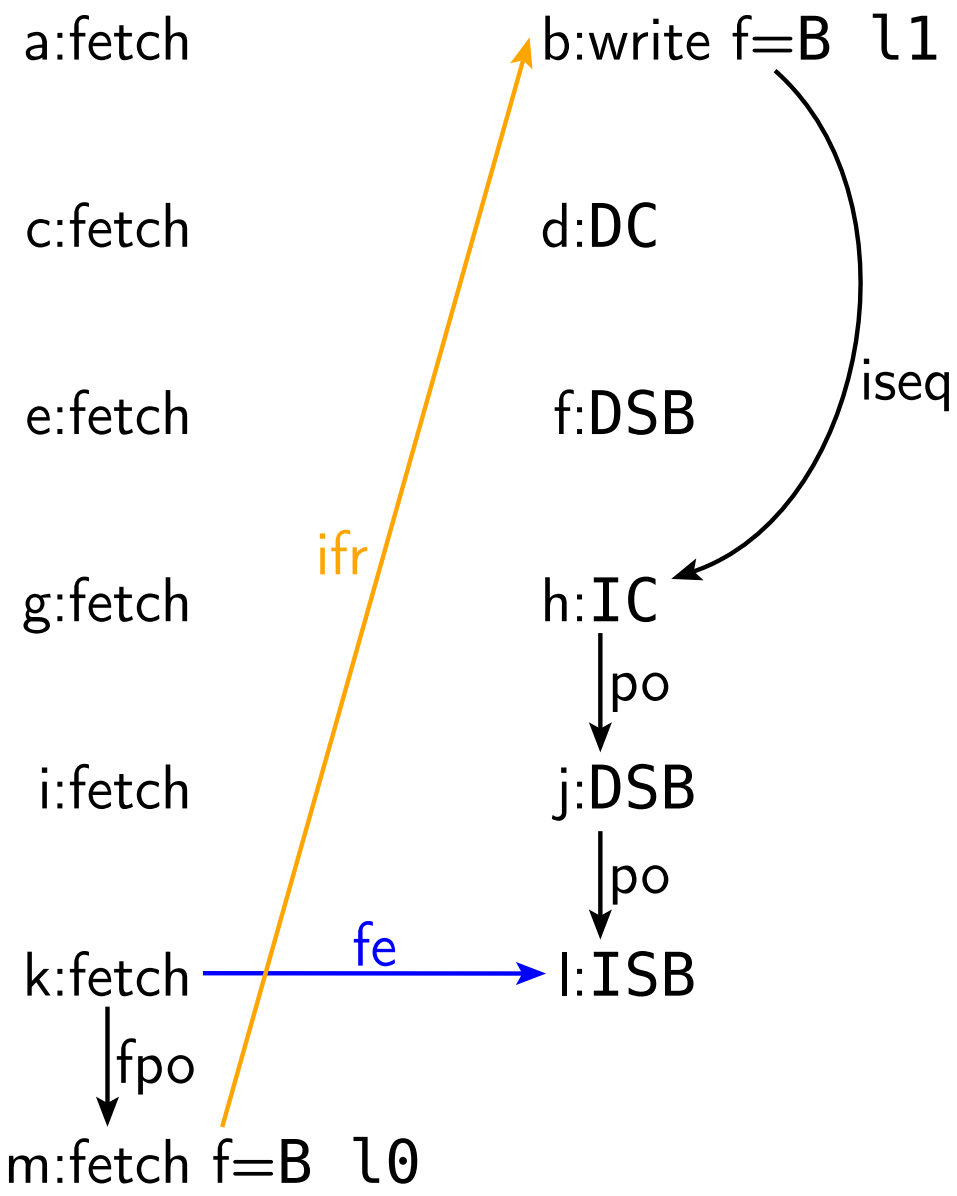

```
let iseq = [W];(wco&scl);[DC];
                (wco&scl);[IC]

(* Observed-by *)
let obs = irf | (ifr;iseq)

(* Fetch-ordered-before *)
let fob = [IF]; fpo; [IF]
    | [IF]; fe
    | [ISB]; fe^-1; fpo

(* Barrier-ordered-before *)
let bob = ...
    | [R|W|F|DC|IC]; po; [dsb.ish]
    | [dsb.ish]; po; [R|W|F|DC|IC]

(* Ordered-before *)
let ob = obs | fob | bob

(* External visibility requirement *)
acyclic ob
```

A Forbidden Instuction Fetch



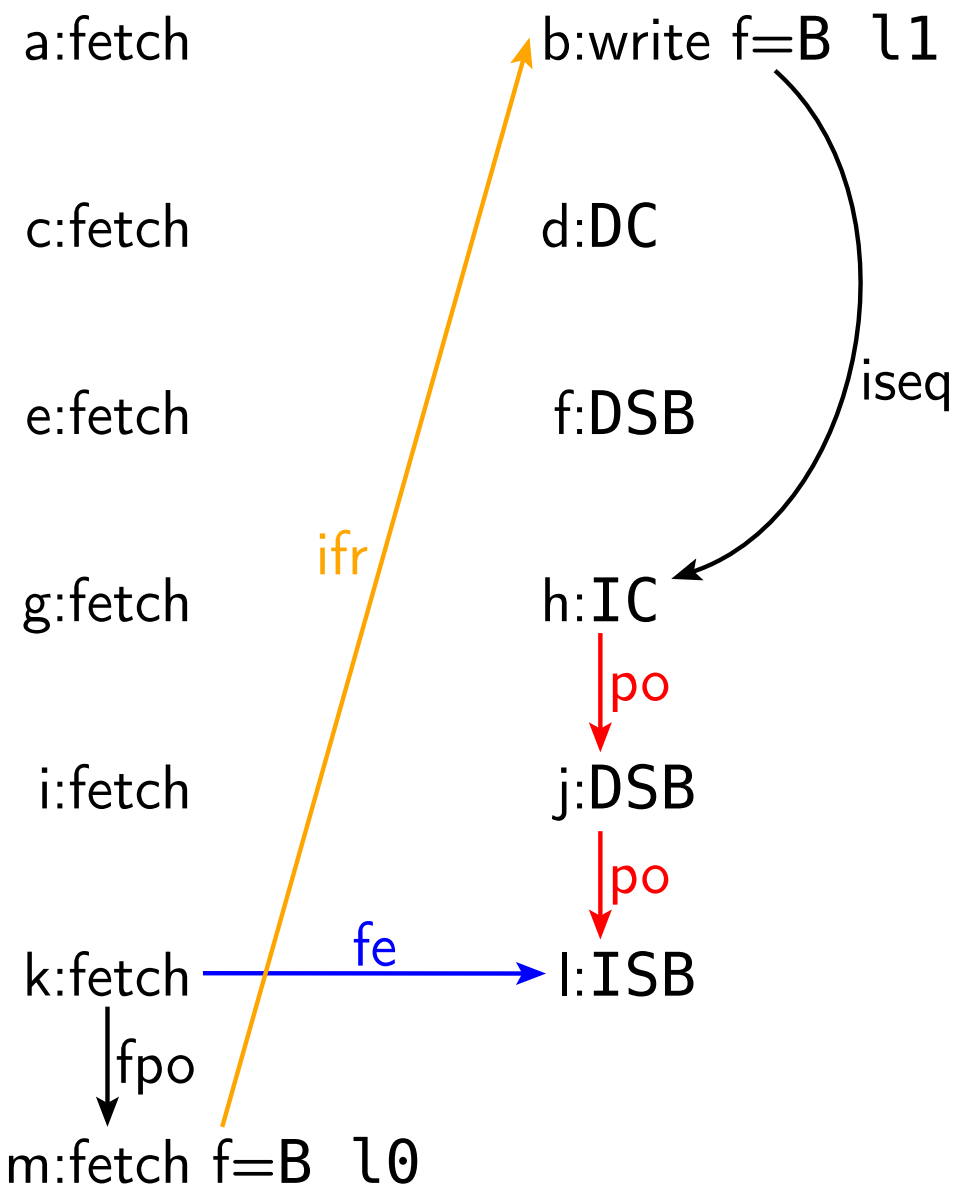

```
let iseq = [W];(wco&scl);[DC];
                (wco&scl);[IC]

(* Observed-by *)
let obs = irf | (ifr;iseq)

(* Fetch-ordered-before *)
let fob = [IF]; fpo; [IF]
    | [IF]; fe
    | [ISB]; fe^-1; fpo

(* Barrier-ordered-before *)
let bob = ...
    | [R|W|F|DC|IC]; po; [dsb.ish]
    | [dsb.ish]; po; [R|W|F|DC|IC]

(* Ordered-before *)
let ob = obs | fob | bob

(* External visibility requirement *)
acyclic ob
```

A Forbidden Instuction Fetch



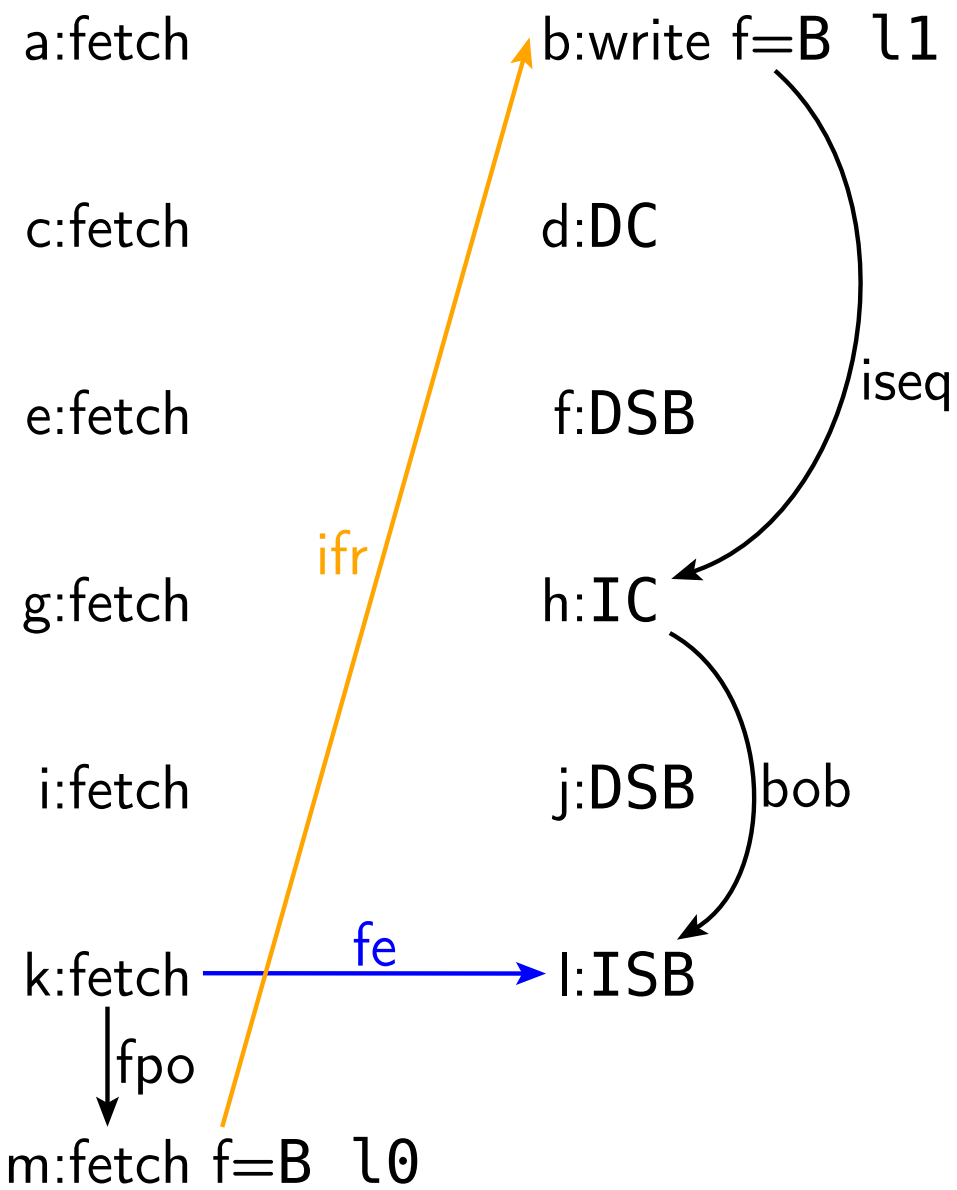

```
let iseq = [W];(wco&scl);[DC];
                (wco&scl);[IC]

(* Observed-by *)
let obs = irf | (ifr;iseq)

(* Fetch-ordered-before *)
let fob = [IF]; fpo; [IF]
    | [IF]; fe
    | [ISB]; fe^-1; fpo

(* Barrier-ordered-before *)
let bob = ...
    | [R|W|F|DC|IC]; po; [dsb.ish]
    | [dsb.ish]; po; [R|W|F|DC|IC]

(* Ordered-before *)
let ob = obs | fob | bob

(* External visibility requirement *)
acyclic ob
```

A Forbidden Instuction Fetch



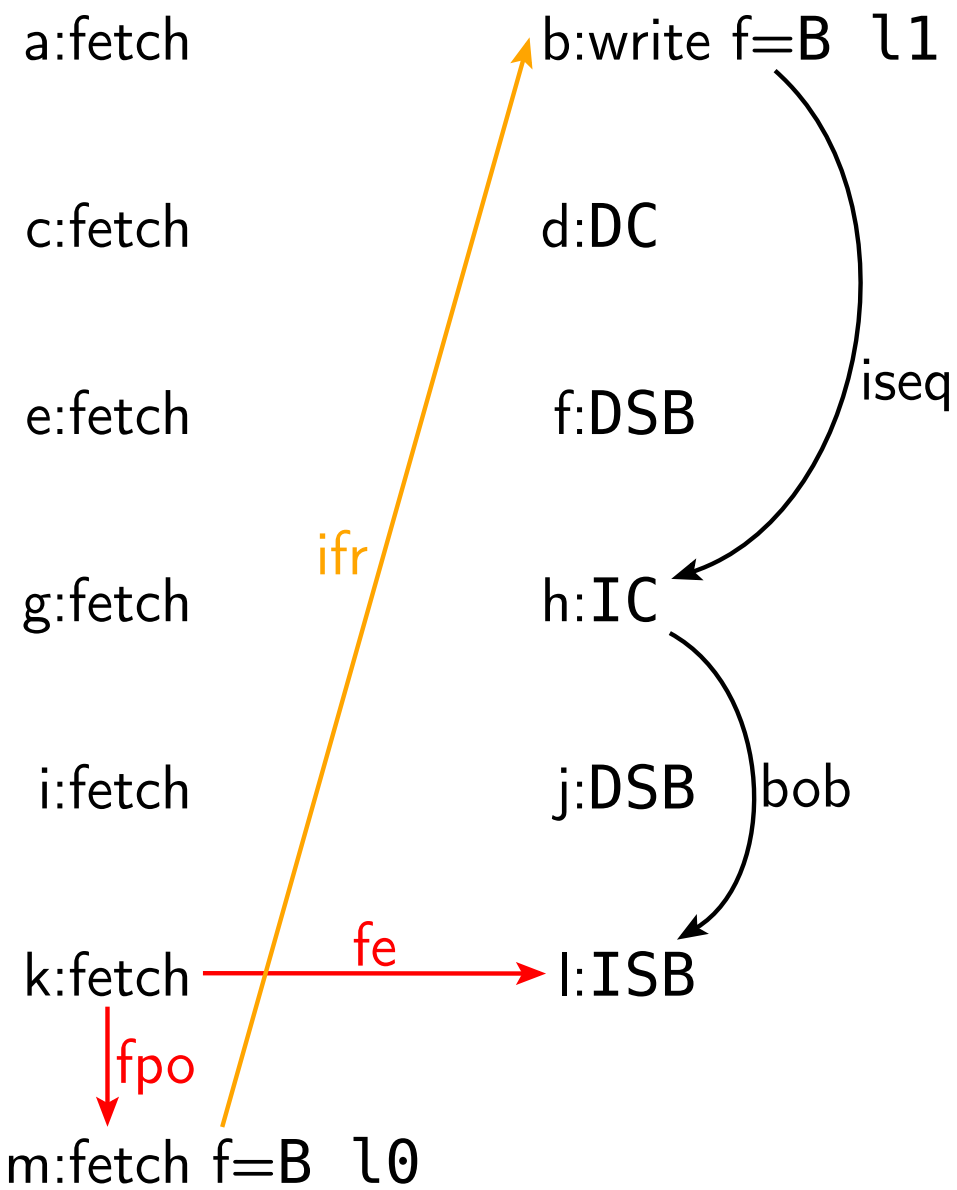

```
let iseq = [W];(wco&scl);[DC];
                (wco&scl);[IC]

(* Observed-by *)
let obs = irf | (ifr;iseq)

(* Fetch-ordered-before *)
let fob = [IF]; fpo; [IF]
    | [IF]; fe
    | [ISB]; fe^-1; fpo

(* Barrier-ordered-before *)
let bob = ...
    | [R|W|F|DC|IC]; po; [dsb.ish]
    | [dsb.ish]; po; [R|W|F|DC|IC]

(* Ordered-before *)
let ob = obs | fob | bob

(* External visibility requirement *)
acyclic ob
```

A Forbidden Instuction Fetch



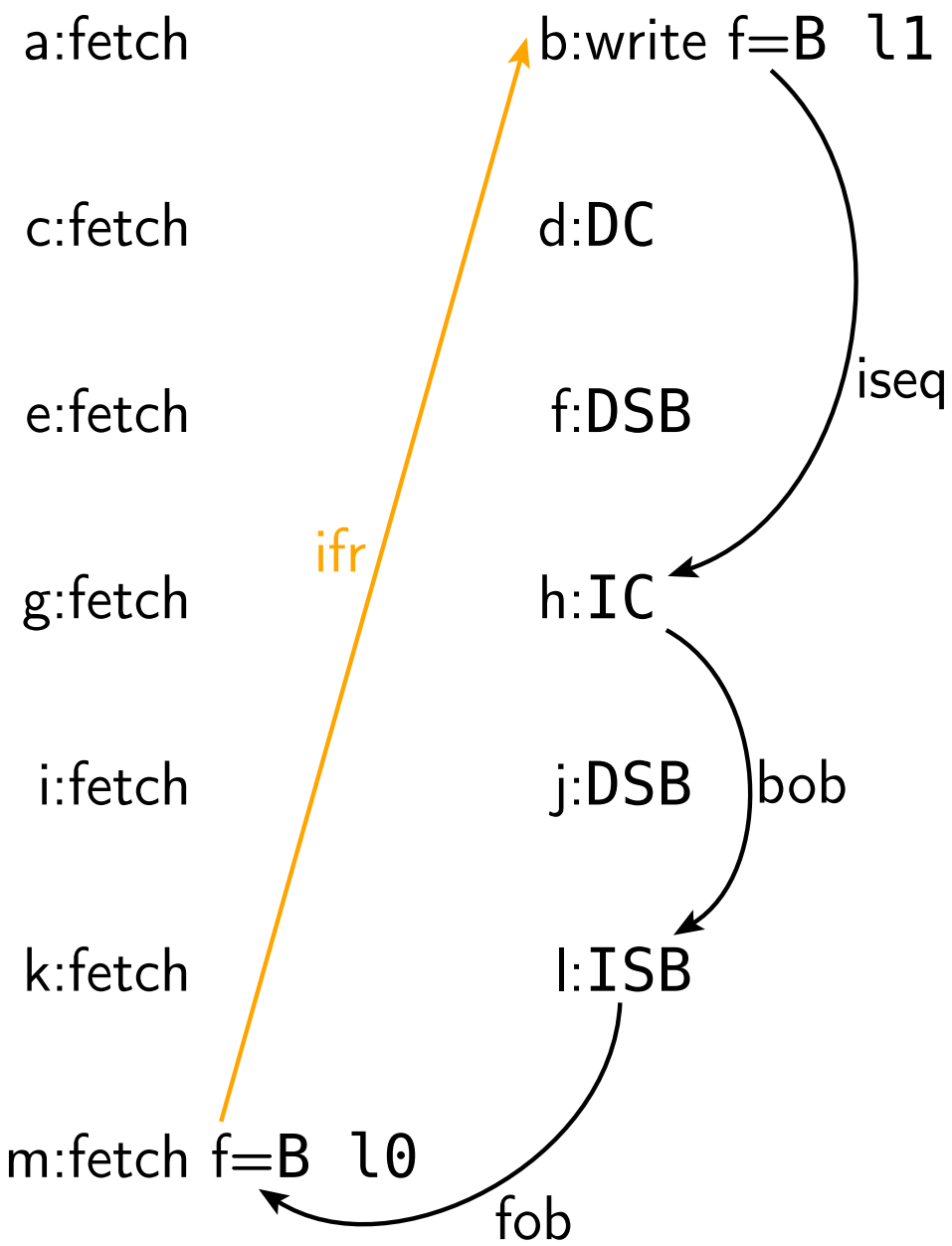

```
let iseq = [W];(wco&scl);[DC];
                (wco&scl);[IC]

(* Observed-by *)
let obs = irf | (ifr;iseq)

(* Fetch-ordered-before *)
let fob = [IF]; fpo; [IF]
    | [IF]; fe
    | [ISB]; fe^-1; fpo

(* Barrier-ordered-before *)
let bob = ...
    | [R|W|F|DC|IC]; po; [dsb.ish]
    | [dsb.ish]; po; [R|W|F|DC|IC]

(* Ordered-before *)
let ob = obs | fob | bob

(* External visibility requirement *)
acyclic ob
```

A Forbidden Instuction Fetch



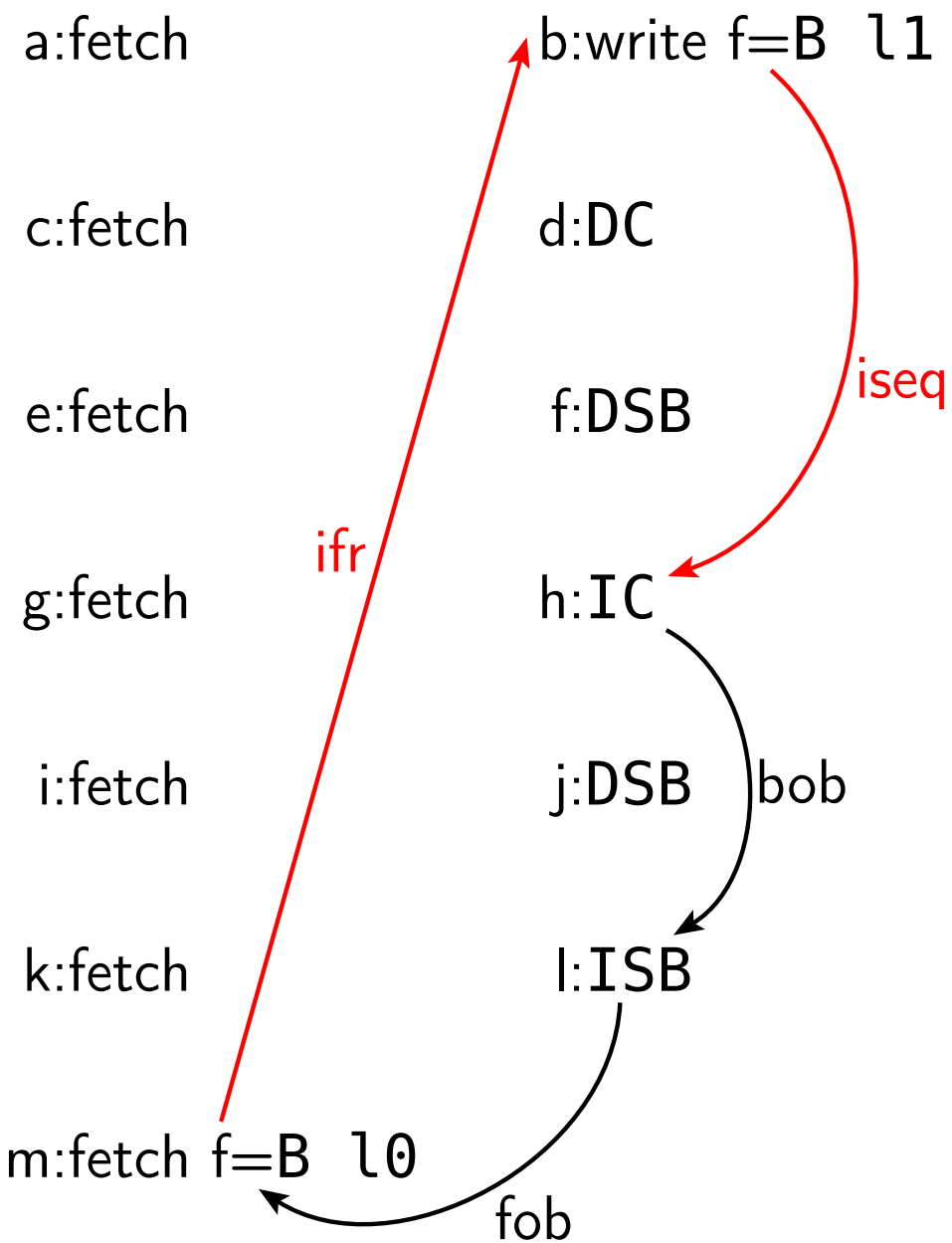

```
let iseq = [W];(wco&scl);[DC];
                (wco&scl);[IC]

(* Observed-by *)
let obs = irf | (ifr;iseq)

(* Fetch-ordered-before *)
let fob = [IF]; fpo; [IF]
    | [IF]; fe
    | [ISB]; fe^-1; fpo

(* Barrier-ordered-before *)
let bob = ...
    | [R|W|F|DC|IC]; po; [dsb.ish]
    | [dsb.ish]; po; [R|W|F|DC|IC]

(* Ordered-before *)
let ob = obs | fob | bob

(* External visibility requirement *)
acyclic ob
```

A Forbidden Instuction Fetch



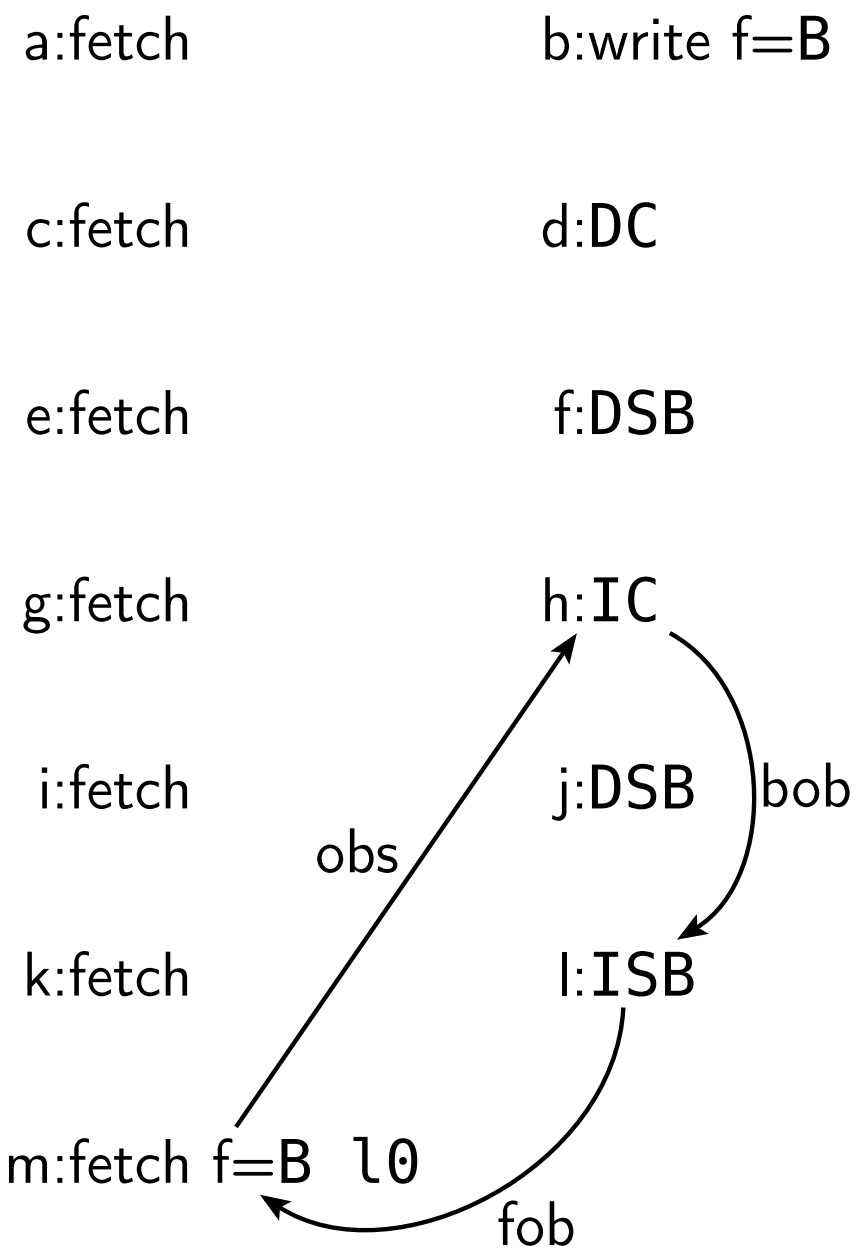

```
let iseq = [W];(wco&scl);[DC];
                (wco&scl);[IC]

(* Observed-by *)
let obs = irf | (ifr;iseq)

(* Fetch-ordered-before *)
let fob = [IF]; fpo; [IF]
    | [IF]; fe
    | [ISB]; fe^-1; fpo

(* Barrier-ordered-before *)
let bob = ...
    | [R|W|F|DC|IC]; po; [dsb.ish]
    | [dsb.ish]; po; [R|W|F|DC|IC]

(* Ordered-before *)
let ob = obs | fob | bob

(* External visibility requirement *)
acyclic ob
```

A Forbidden Instuction Fetch




```
let iseq = [W];(wco&scl);[DC];
                (wco&scl);[IC]

(* Observed-by *)
let obs = irf | (ifr;iseq)

(* Fetch-ordered-before *)
let fob = [IF]; fpo; [IF]
    | [IF]; fe
    | [ISB]; fe^-1; fpo

(* Barrier-ordered-before *)
let bob = ...
    | [R|W|F|DC|IC]; po; [dsb.ish]
    | [dsb.ish]; po; [R|W|F|DC|IC]

(* Ordered-before *)
let ob = obs | fob | bob

(* External visibility requirement *)
acyclic ob
```

A Forbidden Instuction Fetch




```
let iseq = [W];(wco&scl);[DC];
                (wco&scl);[IC]

(* Observed-by *)
let obs = irf | (ifr;iseq)

(* Fetch-ordered-before *)
let fob = [IF]; fpo; [IF]
    | [IF]; fe
    | [ISB]; fe^-1; fpo

(* Barrier-ordered-before *)
let bob = ...
    | [R|W|F|DC|IC]; po; [dsb.ish]
    | [dsb.ish]; po; [R|W|F|DC|IC]

(* Ordered-before *)
let ob = obs | fob | bob

(* External visibility requirement *)
acyclic ob
```

A Forbidden Instuction Fetch

a:fetch b:write f=B l1

c:fetch d:DC

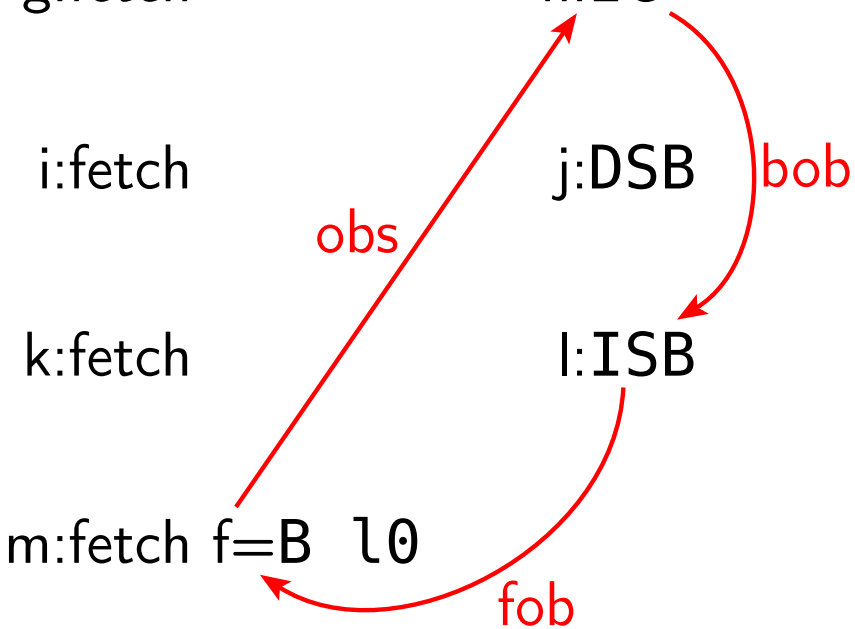
e:fetch f:DSB

g:fetch h:IC

i:fetch j:DSB

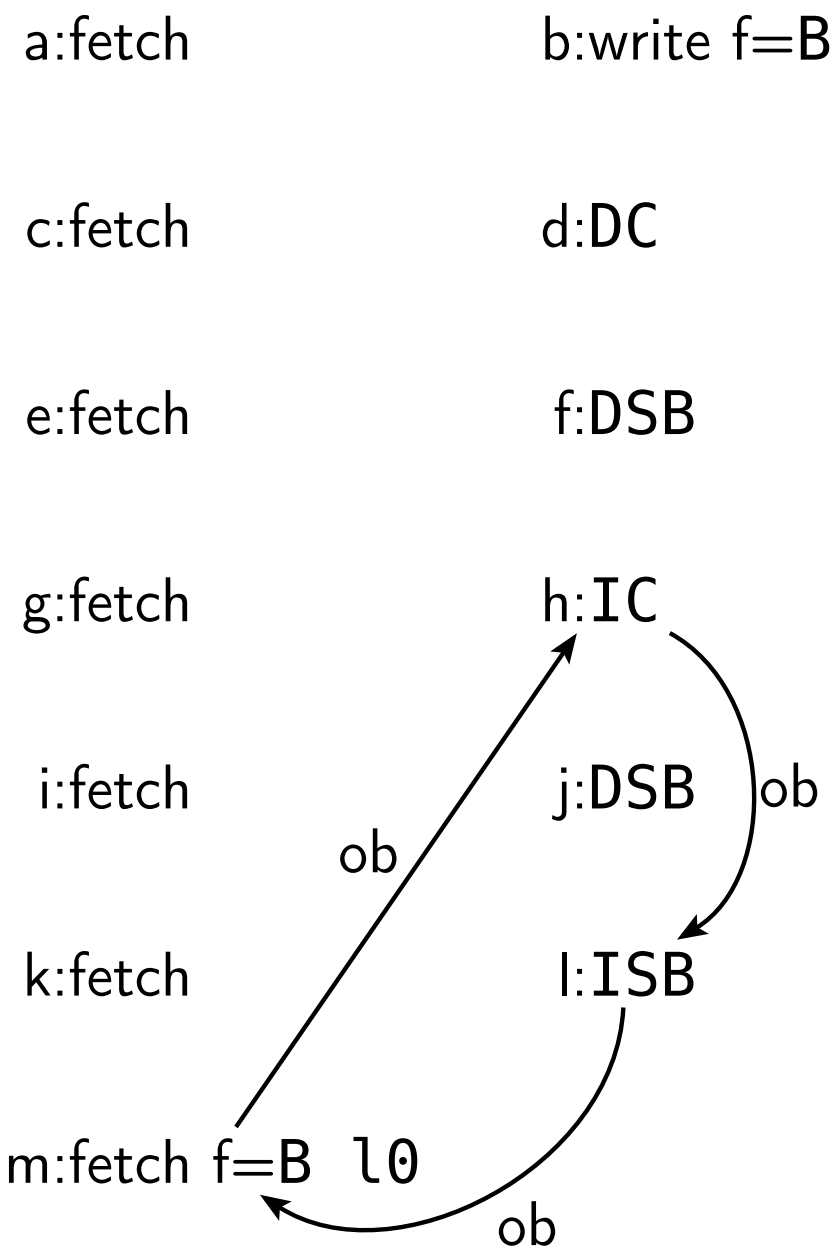
k:fetch l:ISB

m:fetch f=B l0



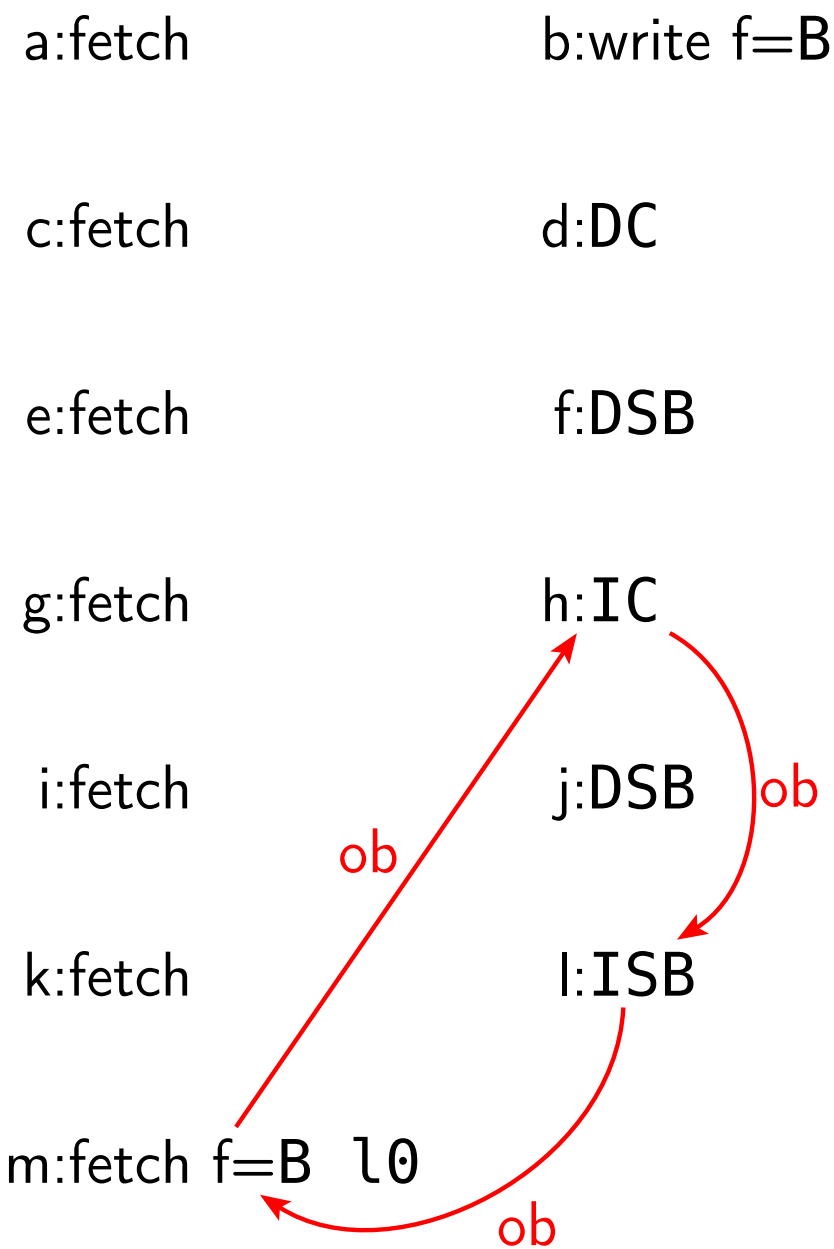

```
let iseq = [W];(wco&scl);[DC];
                (wco&scl);[IC]

(* Observed-by *)
let obs = irf | (ifr;iseq)

(* Fetch-ordered-before *)
let fob = [IF]; fpo; [IF]
    | [IF]; fe
    | [ISB]; fe^-1; fpo

(* Barrier-ordered-before *)
let bob = ...
    | [R|W|F|DC|IC]; po; [dsb.ish]
    | [dsb.ish]; po; [R|W|F|DC|IC]

(* Ordered-before *)
let ob = obs | fob | bob

(* External visibility requirement *)
acyclic ob
```

A Forbidden Instuction Fetch



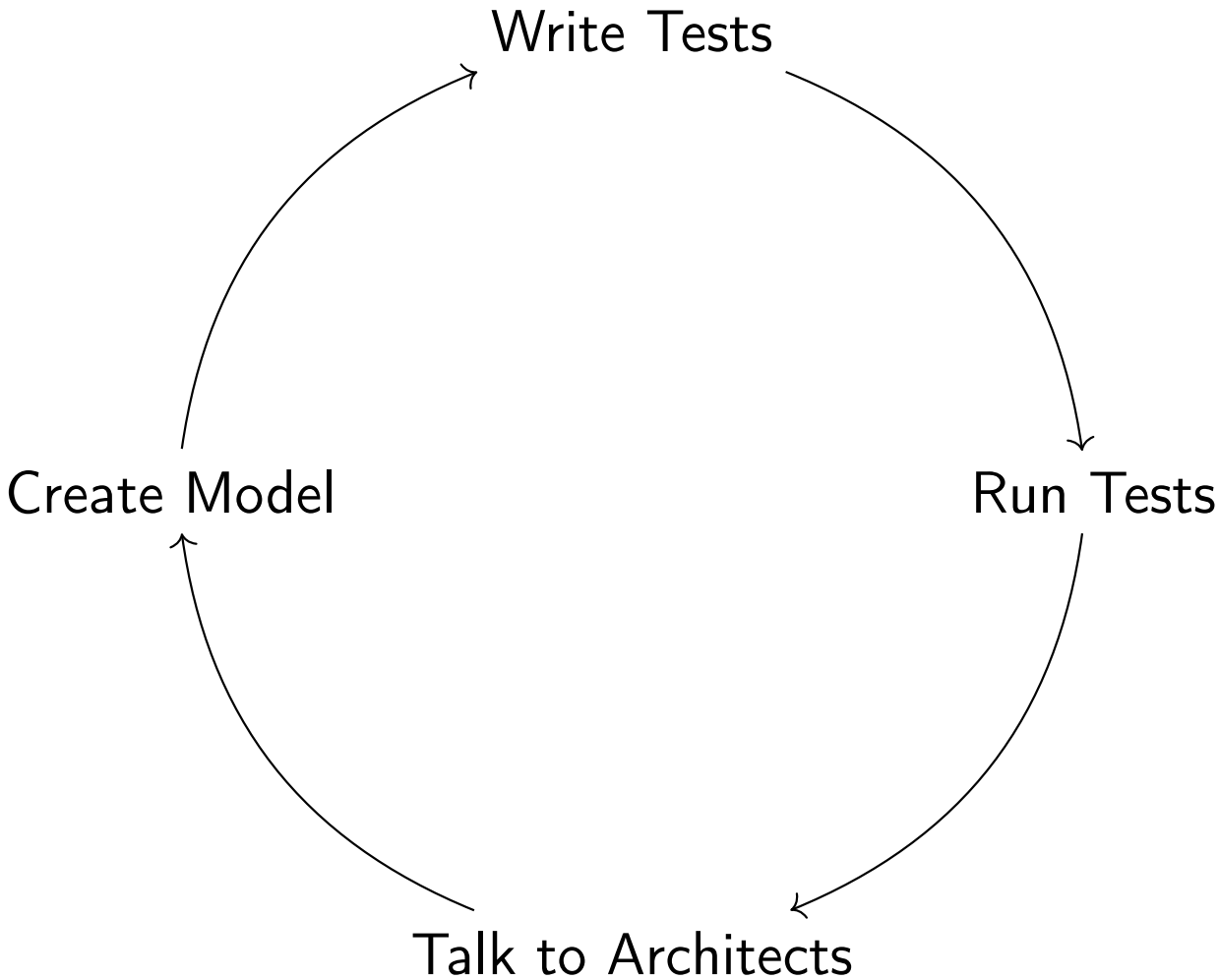

```
let iseq = [W];(wco&scl);[DC];
                (wco&scl);[IC]

(* Observed-by *)
let obs = irf | (ifr;iseq)

(* Fetch-ordered-before *)
let fob = [IF]; fpo; [IF]
    | [IF]; fe
    | [ISB]; fe^-1; fpo

(* Barrier-ordered-before *)
let bob = ...
    | [R|W|F|DC|IC]; po; [dsb.ish]
    | [dsb.ish]; po; [R|W|F|DC|IC]

(* Ordered-before *)
let ob = obs | fob | bob

(* External visibility requirement *)
acyclic ob
```

A Forbidden Instuction Fetch




```
let iseq = [W];(wco&scl);[DC];
                (wco&scl);[IC]

(* Observed-by *)
let obs = irf | (ifr;iseq)

(* Fetch-ordered-before *)
let fob = [IF]; fpo; [IF]
    | [IF]; fe
    | [ISB]; fe^-1; fpo

(* Barrier-ordered-before *)
let bob = ...
    | [R|W|F|DC|IC]; po; [dsb.ish]
    | [dsb.ish]; po; [R|W|F|DC|IC]

(* Ordered-before *)
let ob = obs | fob | bob

(* External visibility requirement *)
acyclic ob
```

Modelling Process



Validation

Validating the model:

- approx. 35 hand-written tests.
- approx. 1500 auto-generated tests.

Ran on multiple devices and compared results to our models:

Found some hardware bugs;

Many places hardware not as relaxed as architecture allows!

			Snapdragon 810				Exynos 8895	
	Status	Total	A53	A57	Tegra K1	Snapdragon 820	A53	M2
CoFF	Allow	294K/2.8G	291K/620M	$^{U}0/430M$	$^{U}0/30M$	$^{U}0/520M$	3.0K/580M	$^{U}0/600M$
CoFF-isb	Allow	211K/2.8G	200K/620M	$^{U}0/430M$	$^{U}0/30M$	$^{U}0/520M$	11K/580M	$^{U}0/600M$
CoFF.branches	Allow	$^{U}0/550M$	?	?	$^{U}0/30M$	$^{U}0/520M$	?	?
CoFR	Forbid	0/2.8G	0/620M	0/430M	0/30M	0/520M	0/580M	0/600M
CoRF	Allow	979M/2.8G	231M/620M	179M/430M	6.2K/30M	124M/520M	238M/580M	207M/580M
CoRF+ctrl-isb	Allow	2.2G/2.8G	422M/620M	411M/430M	$^{U}0/30M$	506M/520M	343M/580M	540M/580M
FOW	Allow	$^{U}0/1.1G$	$^{U}0/310M$	?	?	$^{U}0/250M$	$^{U}0/290M$	$^{U}0/290M$
ISA2+dmb+cachesync+ctrl-isb	Allow	$^{U}0/1.1G$	$^{U}0/310M$	?	?	$^{U}0/250M$	$^{U}0/290M$	$^{U}0/290M$
ISA2+dmb+dsb-cachesync+ctrl-isb	Forbid	0/1.1G	0/310M	?	?	0/250M	0/290M	0/290M
ISA2.F+dc-dmb+dsb-ic-dsb+ctrl-isb	Allow	$^{U}0/2.9G$	$^{U}0/910M$	?	?	$^{U}0/250M$	$^{U}0/850M$	$^{U}0/850M$
ISA2.F+dc-dsb+dsb-ic-dsb+ctrl-isb	Forbid	0/2.9G	0/910M	?	?	0/250M	0/850M	0/850M
MP.FF	Allow	972K/2.7G	548K/620M	39K/430M	$^{U}0/30M$	26K/500M	353K/560M	5.5K/580M
MP.FF+cachesync+po	Forbid	0/2.7G	0/620M	0/430M	0/30M	0/500M	0/560M	0/580M
MP.FF+dmb+po	Allow	453K/2.7G	287K/620M	13K/430M	$^{U}0/30M$	3.9K/500M	142K/560M	6.0K/580M
MP.FR+dmb+po	Forbid	0/2.7G	0/620M	0/430M	0/30M	0/500M	0/560M	0/580M
MP.R.RF+addr-cachesync+dmb+ctrl-isb	Forbid	0/1.1G	0/300M	?	?	0/250M	0/280M	0/290M
MP.RF+cachesync+ctrl	Allow	3.4K/2.7G	1.3K/600M	556/420M	$^{U}0/30M$	1.5K/500M	30/560M	$^{U}0/580M$
MP.RF+cachesync+ctrl-isb	Forbid	$^{F}7/2.7G$	0/600M	0/430M	0/30M	$^{F}7/500M$	0/560M	0/580M
MP.RF+dc+ctrl-isb	Allow	5.6G/6.8G	1.4G/1.8G	797M/1.0G	$^{U}0/30M$	491M/500M	1.4G/1.7G	1.6G/1.7G
MP.RF+dc-dmb+ctrl-isb	Allow	5.7G/6.8G	1.4G/1.8G	832M/1.0G	$^{U}0/30M$	490M/500M	1.3G/1.7G	1.6G/1.7G
MP.RF+dc-dsb+ctrl-isb	Allow	5.6G/6.7G	1.4G/1.8G	795M/1.0G	$^{U}0/30M$	492M/500M	1.3G/1.7G	1.6G/1.7G
MP.RF+dc-dsb-ic-dsbnsh+ctrl-isb	Allow	216M/6.7G	$^{U}0/1.8G$	$^{U}0/1.0G$	$^{U}0/30M$	216M/500M	$^{U}0/1.7G$	$^{U}0/1.7G$
MP.RF+ic+ctrl-isb	Allow	985M/2.7G	$^{U}0/600M$	417M/420M	$^{U}0/20M$	13/500M	$^{U}0/560M$	568M/580M
MP.RFF+dc-dsb+ctrl-isb	Allow	1/6.7G	1/1.8G	$^{U}0/1.0G$	$^{U}0/30M$	$^{U}0/500M$	$^{U}0/1.7G$	$^{U}0/1.7G$
MP.RFF+dc-dsb+ctrl-isb-isb	Allow	$^{U}0/6.7G$	$^{U}0/1.8G$	$^{U}0/1.0G$	$^{U}0/30M$	$^{U}0/500M$	$^{U}0/1.7G$	$^{U}0/1.7G$
MP.RRF+dmb+addr-cachesync-isb	Allow	$^{U}0/2.7G$	$^{U}0/600M$	$^{U}0/420M$	$^{U}0/20M$	$^{U}0/500M$	$^{U}0/560M$	$^{U}0/580M$
SM	Allow	5.4G/5.4G	1.2G/1.2G	840M/840M	38M/40M	1.0G/1.0G	1.1G/1.1G	1.2G/1.2G
SM+RW	Allow	$^{U}0/2.7G$	$^{U}0/600M$	$^{U}0/420M$	$^{U}0/20M$	$^{U}0/500M$	$^{U}0/560M$	$^{U}0/580M$
SM+RW+RW	Forbid	0/1.1G	0/300M	?	?	0/250M	0/280M	0/290M
SM+cachesync	Allow	4.2G/5.4G	1.2G/1.2G	840M/840M	$^{U}0/40M$	1.0G/1.0G	1.1G/1.1G	$^{U}0/1.2G$

Future

- Exceptions and Interrupts
- Pagetables and TLBs
- Devices, DMA, Non-Volatile Memory

End

So far:

- Re-cap “relaxed-memory” / x86-TSO.
  - Operational & “Axiomatic” models
- JIT usage
- Arm self-modifying code
- ARMv8 Architectural Operational Model
- ARMv8 Axiomatic Model
- Modelling and validation