peeling the layers under programming languages

Jean Pichon-Pharabod

Motivation

What does

x

do?

Easy: look up the value of x!

Or maybe a bit more underneath ...

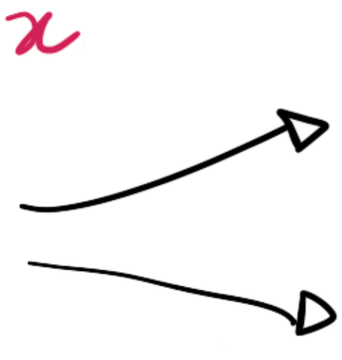
Why look?

- because it's interesting
- because it's observable
  - timing/performance behaviour !!!
  - leaky abstraction

Interpreters and compilers

↳ reify the abstract name x (it's not just a string (scoping, ...))

into something concrete



	$x = x+1$
a register X_i	ADD $X_3, X_3, \#1$

a memory location	LDR $X_5, [X_7]$
	ADD $X_5, X_5, \#1$
	STR $X_5, [X_7]$

or a mixture

$x = x+1; x = x+2$

LDR $X_5, [X_7]$
ADD $X_5, X_5, \#1$
ADD $X_5, X_5, \#2$
STR $X_5, [X_7]$

conflate RWRW into RW

Optimisation

```
int x = 0;
int y = 1;
while (x < n-1) {
    s += a[x] * a[y];
    x++;
    y++;
}
```

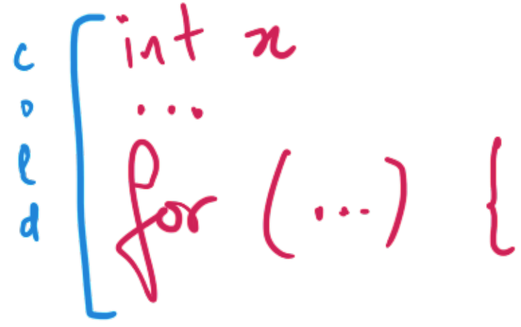
⇝

```
...
LDR X7, [X3]      ← a[x]
LDR X8, [X3, #4]  ← a[y]
MUL X7, X7, X8
...
ADD X3, X3, #4    ← a++
...
```

x and y don't exist!

silly for simplicity

JIT



→ interpreted ⇝ x in memory

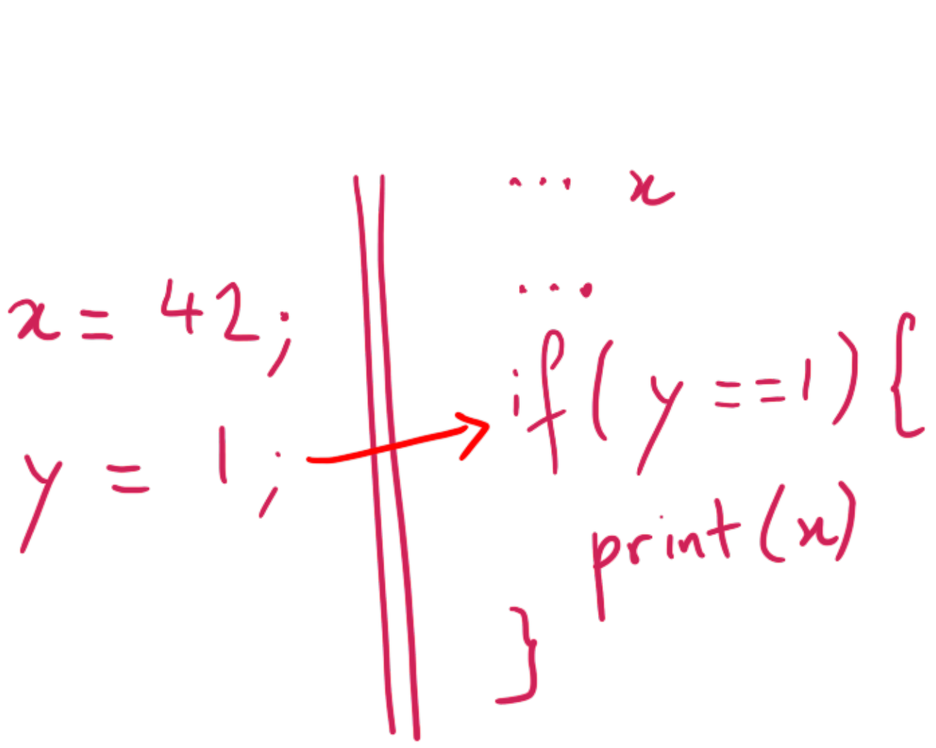
→ compiled ⇝ x in mix of register(s) and memory

→ interpreted ⇝ x in memory

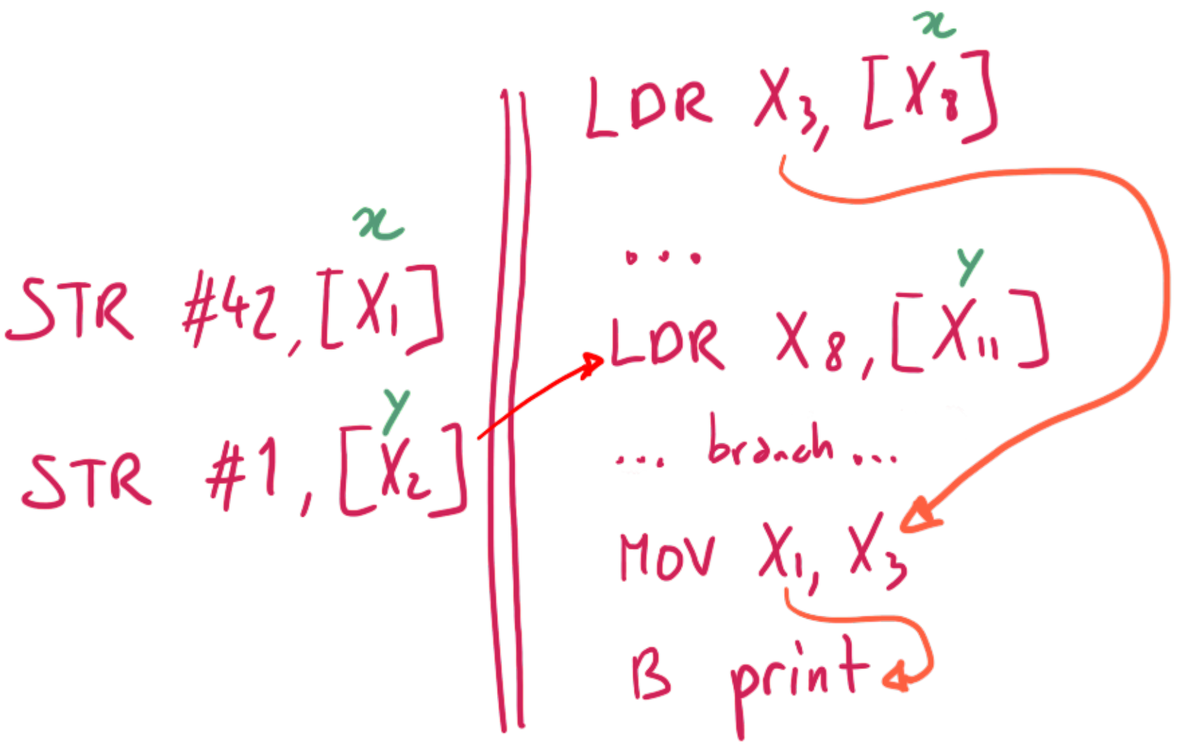
in practice, many tiers of compilers

multiple versions of x can exist simultaneously

But does it matter?



No interleaving prints 0



prints 0 !?!

* terms and conditions apply

⟹ It does not just affect performance

it also affects (concurrent) behavior!

↑

the compiler leads to behaviour that cannot be accounted for by interleaving

"relaxed behavior"

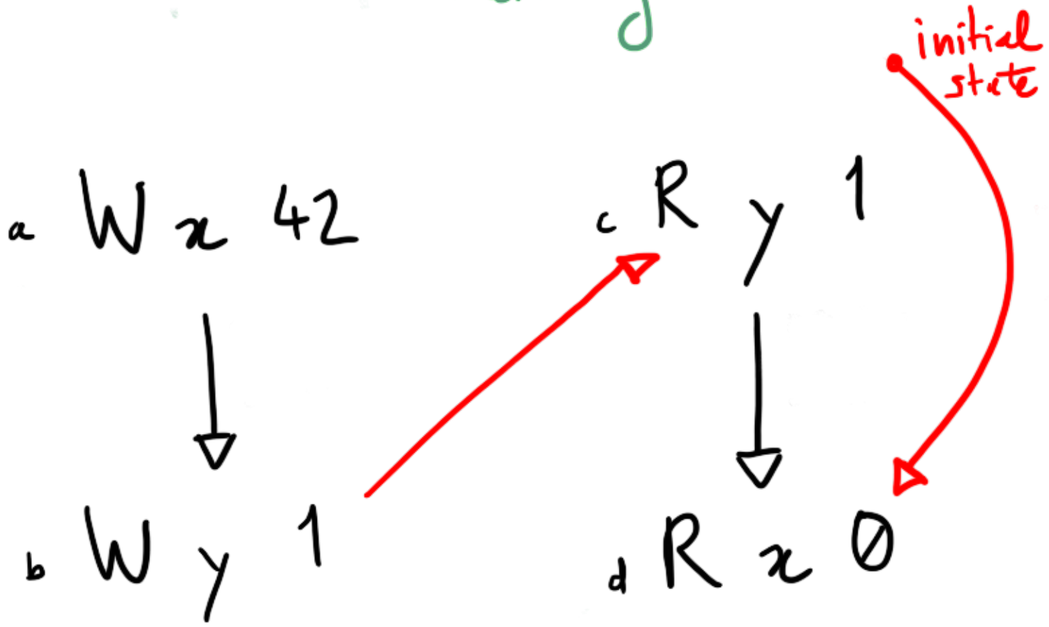
... what about the hardware?

x and y initially 0

STR #42, [x]	LDR X0, [y]
STR #1, [y]	LDR X1, [x]

Q: is $X_0 = 1, X_1 = 0$ possible? yes!

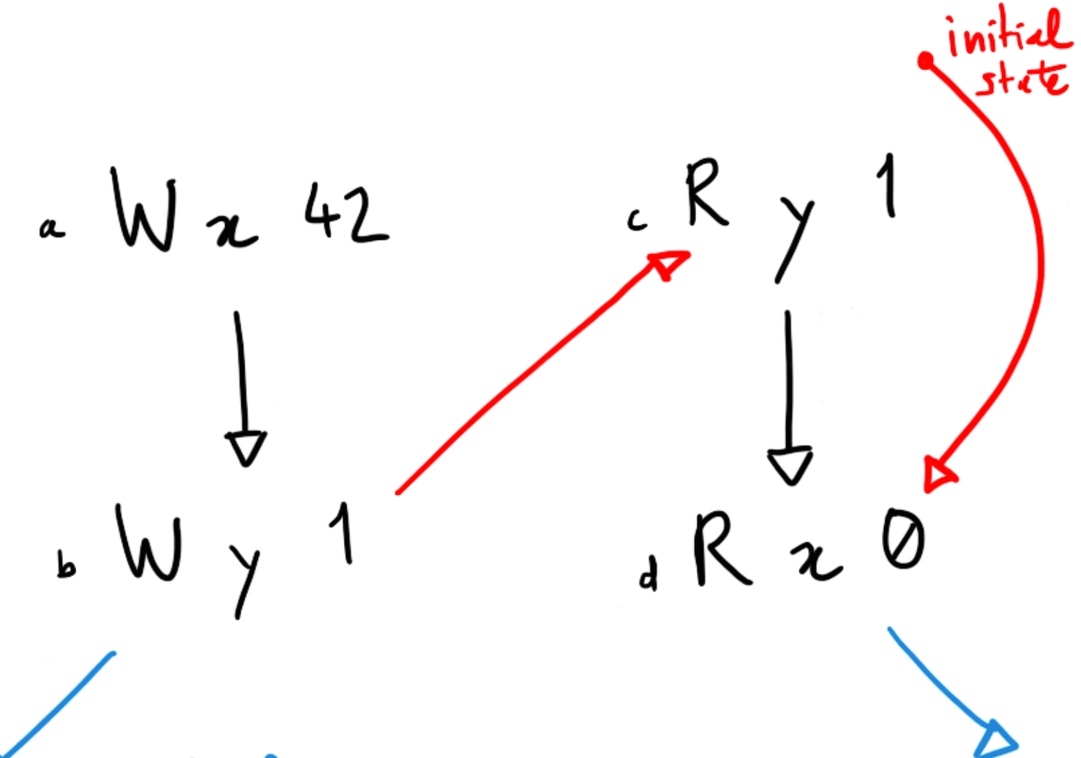
non-interleaving



Key

→ program order

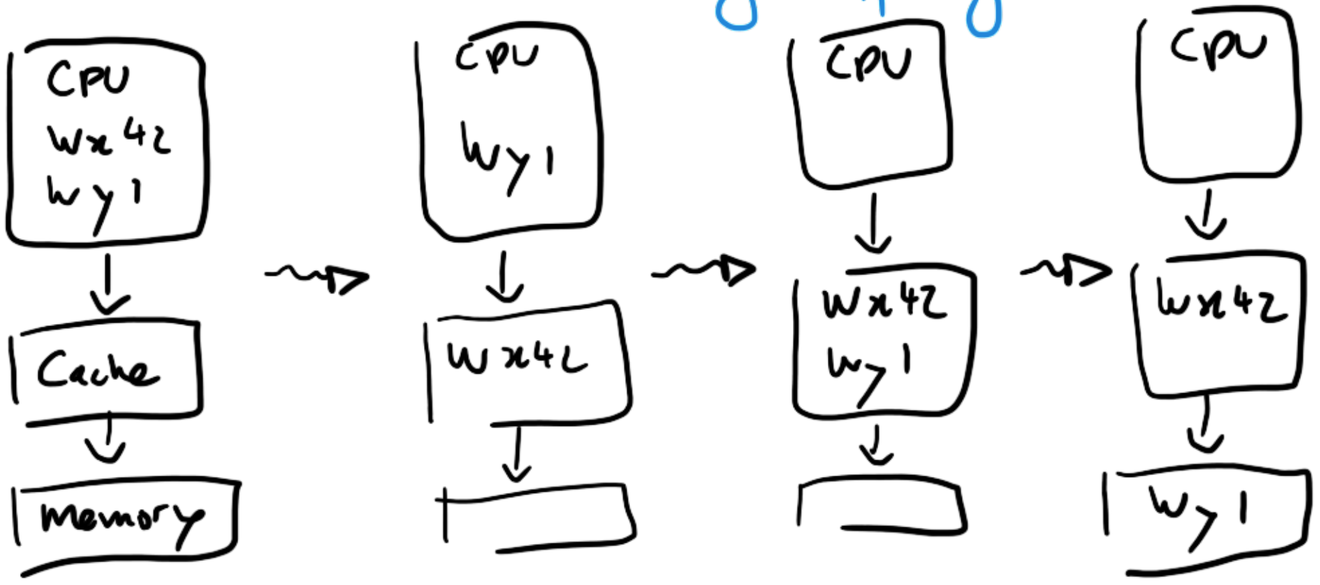
→ reads from (backwards)



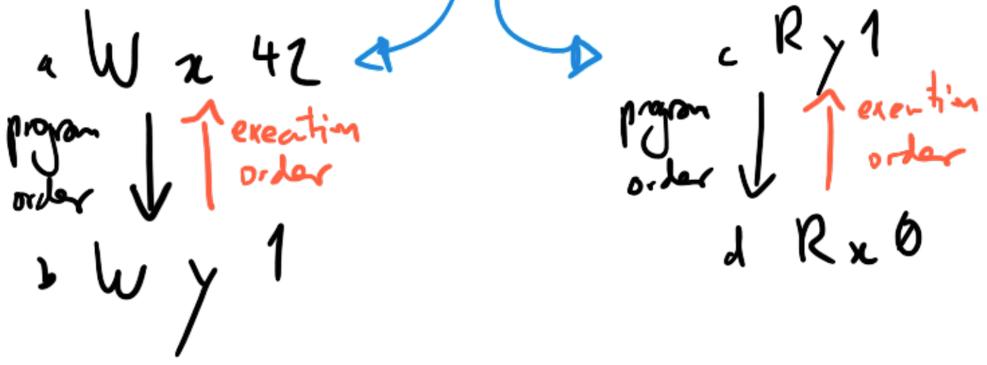
Allowed and observed on many architectures (Arm, Power, RISC-V)

(other types of relaxed behavior on x86)

Reason 1: caching / buffering



Reason 2: out-of-order execution & speculation



Q What is one to do?

Do we need to know the details of each individual machine?

~10 years ago, yes :-(

Not anymore! :-) Flur et al '16, Pulte et al '18

and others

Precise, authoritative model for Arm-A

Can prove that some behaviour is allowed/forbidden

machine-checked

If the hardware disagrees, it is wrong!

↝ are we done?

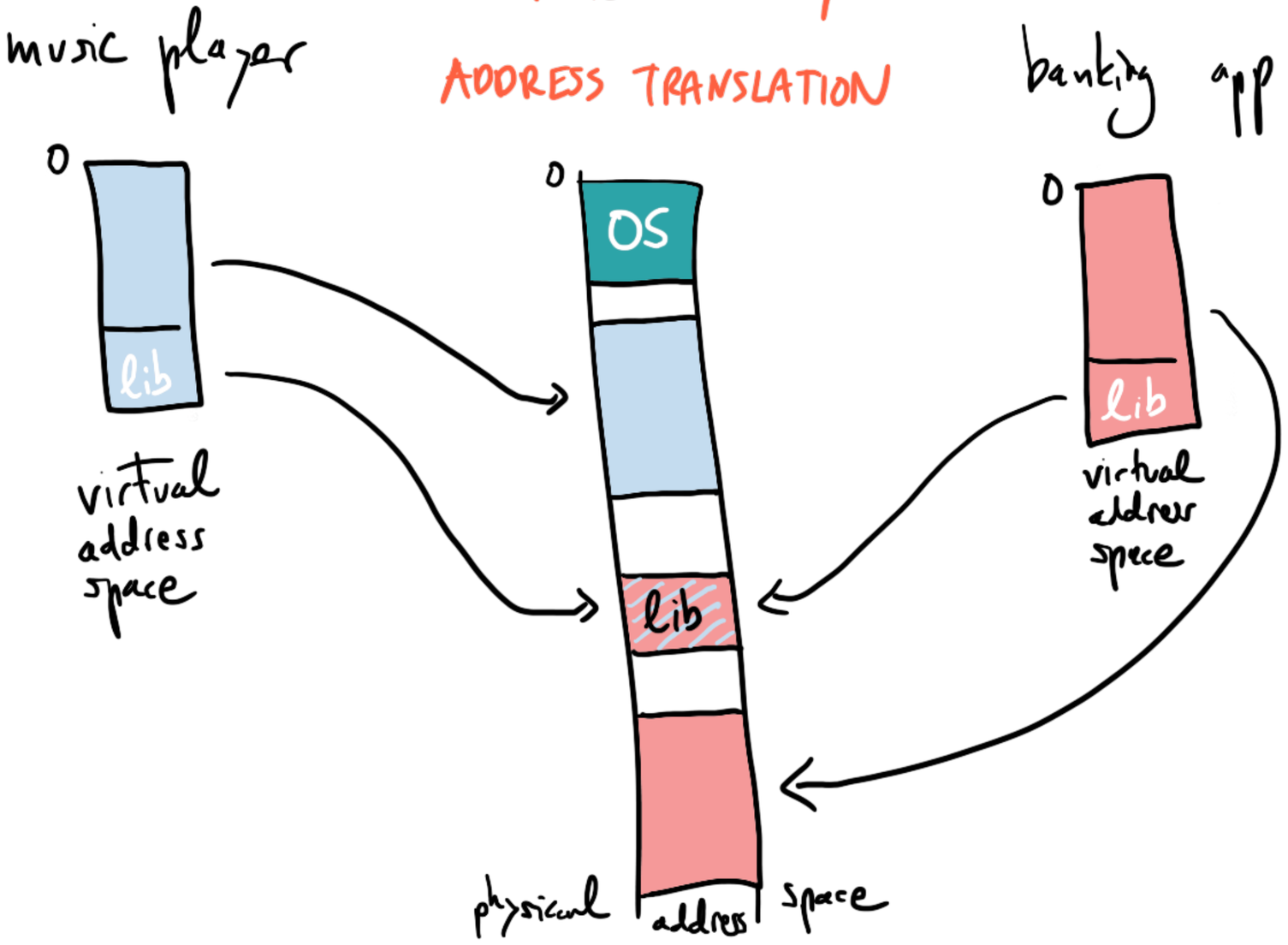
```
let ca = fr | co                                (* Coherence-after *)
let obs = rfe | fre | coe                       (* Observed-by *)
let dob = addr | data                           (* Dependency-ordered-before *)
      | ctrl; [W]
      | (ctrl | (addr; po)); [ISB]; po; [R]
      | addr; po; [W]
      | (ctrl | data); coi
      | (addr | data); rfi
let aob = rmw                                   (* Atomic-ordered-before *)
      | [range(rmw)]; rfi; [A | Q]
let bob = po; [dmb.full]; po                    (* Barrier-ordered-before *)
      | [L]; po; [A]
      | [R]; po; [dmb.ld]; po
      | [A | Q]; po
      | [W]; po; [dmb.st]; po; [W]
      | po; [L]
      | po; [L]; coi
let ob = (obs | dob | aob | bob)+               (* Ordered-before *)

acyclic po-loc | ca | rf    as internal
```

Not quite... I don't want my music player to affect my banking app, but I want them to both use the same os library efficiently

VIRTUAL MEMORY

ADDRESS TRANSLATION



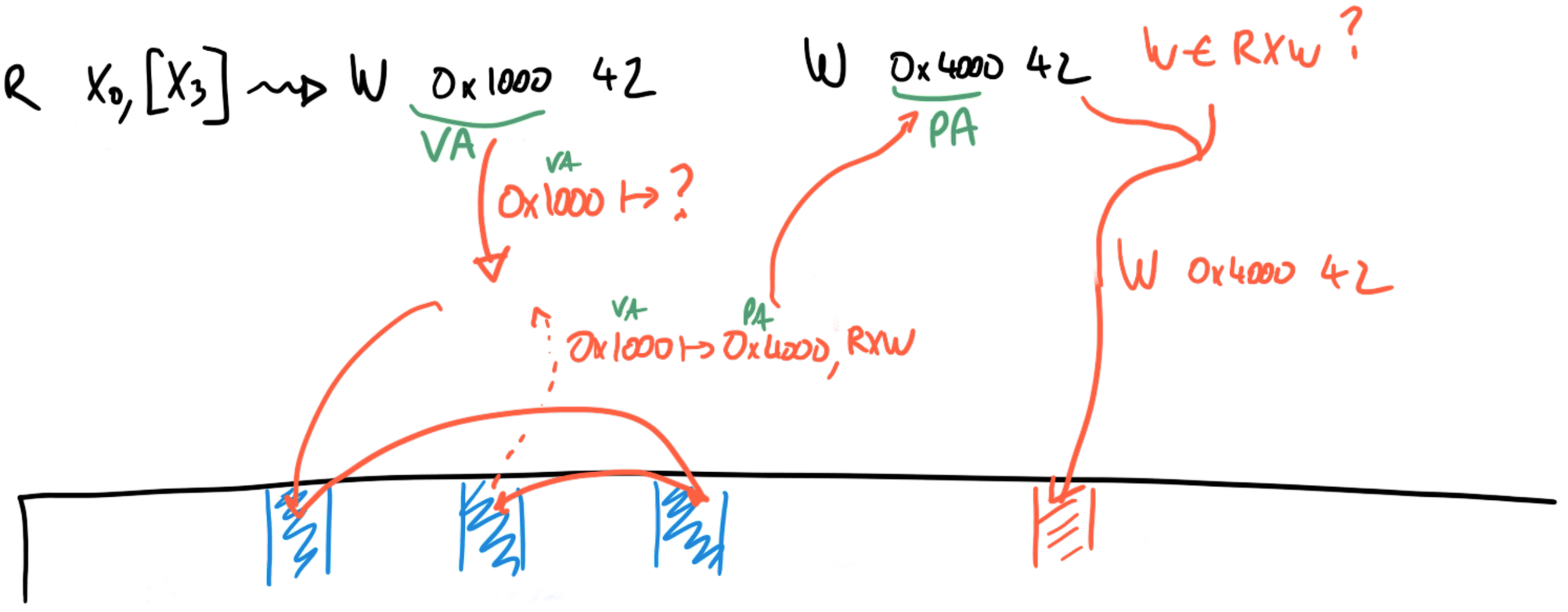
How to implement such address translation? ← configurable, adaptable, ...

↳ hardware-software collaboration, using an in-memory data structure: page tables

↳ edits page tables

↳ looks up page tables

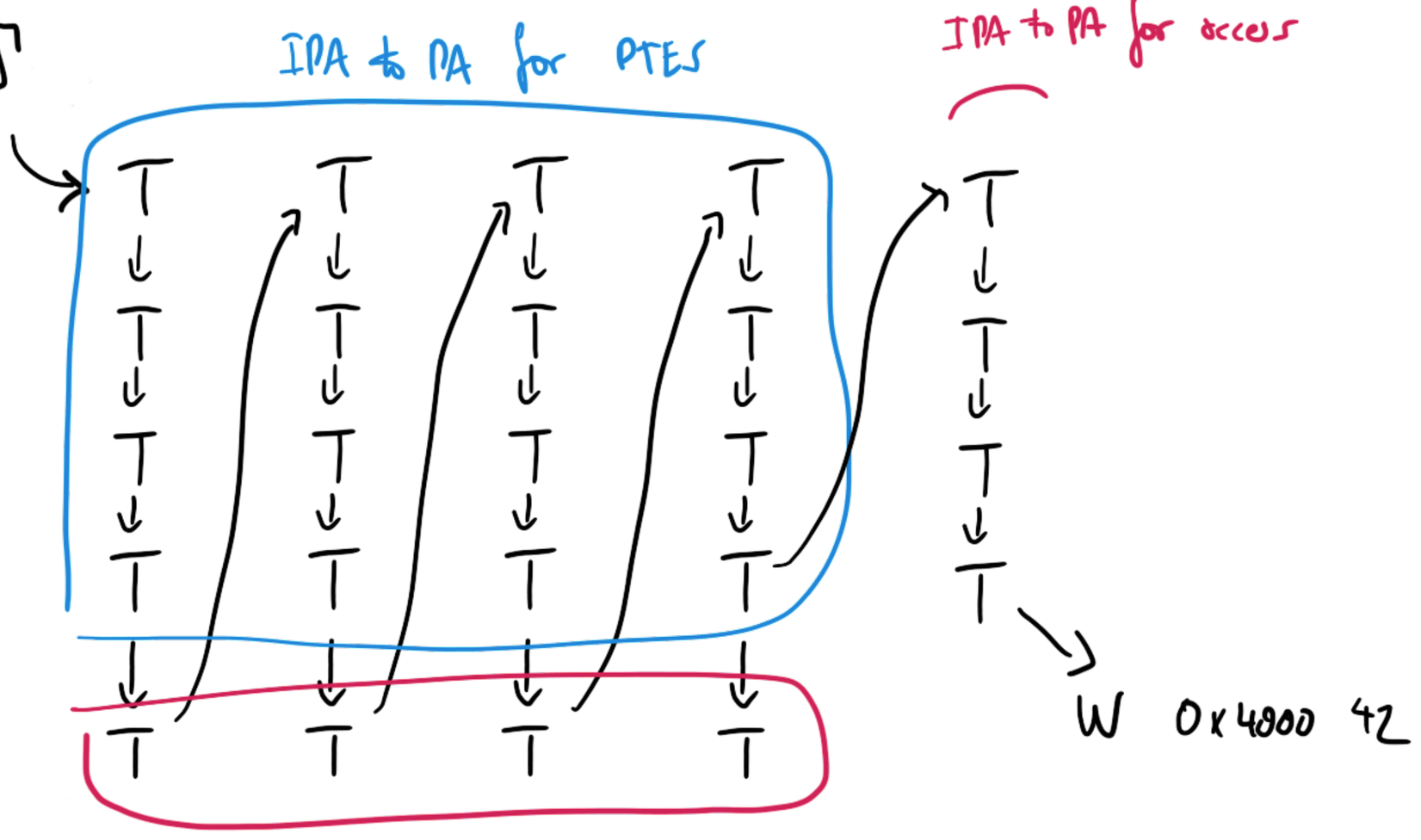
STR $X_0, [X_3]$ ⇝ W 0x1000 42



Accessing memory requires memory lookups! up to 4 levels in a page table

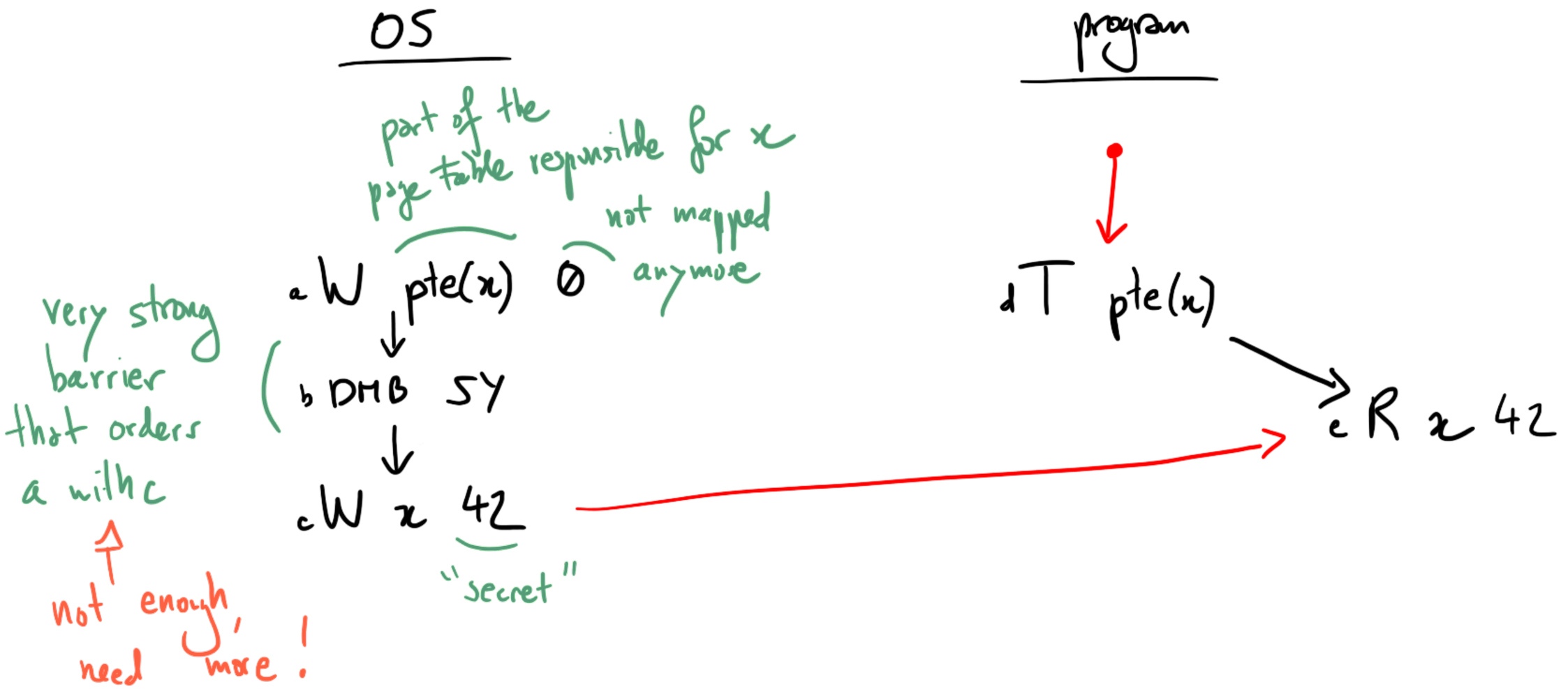
... and there are two stages of translation!

IF pc 「STR X0,[X1]」



How is this tractable?

↳ aggressive caching ... with strange effects



```
let tlb-affects =
  (* see extended version *)

let TLB_barrier =
  ([TLBI] ; tlb-affects ; [T] ; tfr ; [W])^-1
  & wco

let maybe_TLB_cached =
  ([T] ; trf^-1 ; wco ; [TLBI-S1]) & tlb-
     affects^-1

let tcache1 = [T & Stage1] ; tfr ; TLB_barrier
let tcache2 = [T & Stage2] ; tfr ; TLB_barrier

let speculative =
    ctrl
  | addr; po
  | [T] ; instruction-order
(* translation-ordered-before *)
let tob =
    [T_f] ; tfre
  | ([T_f] ; tfri)
    & (po ; [DSB.SY] ; instruction-order)^-1
  | [T] ; iio ; [R|W] ; po ; [W]
  | speculative ; trfi
(* observed by *)
let obs = rfe | fr | wco
  | trfe
(* ordered-before TLBI and translate *)
let obtlbi_translate =
    tcache1
  | tcache2
    & (iio^-1 ; [T & Stage1] ; trf^-1 ; wco^-1)
  | (tcache2 ; wco? ; [TLBI-S1])
    & (iio^-1 ; [T & Stage1] ; maybe_TLB_cached
     )

(* ordered-before TLBI *)
let obtlbi =
    obtlbi_translate
  | [R|W|Fault] ; iio^-1 ; (obtlbi_translate &
     ext) ; [TLBI]

(* context-change ordered-before *)
let ctxob =
    speculative ; [MSR]
  | [CSE] ; instruction-order
  | [ContextChange] ; po ; [CSE]
  | speculative ; [CSE]
  | po ; [ERET] ; instruction-order ; [T]
```

```
(* ordered-before a translation fault *)
let obfault =
    data ; [Fault & IsFromW]
  | speculative ; [Fault & IsFromW]
  | [dmbst] ; po ; [Fault & IsFromW]
  | [dmbld] ; po ; [Fault & (IsFromW|IsFromR)]
  | [A|Q] ; po ; [Fault & (IsFromW | IsFromR)]
  | [R|W] ; po ; [Fault & IsFromW & IsReleaseW]

(* ETS-ordered-before *)
let obETS =
    (obfault ; [Fault]) ; iio^-1 ; [T_f]
  | ([TLBI] ; po ; [dsb] ; instruction-order ;
     [T]) & tlb-affects

(* dependency-ordered-before *)
let dob =
    addr | data
  | speculative ; [W]
  | addr; po; [W]
  | (addr | data); rfi
  | (addr | data); trfi

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

(* barrier-ordered-before *)
let bob = [R] ; po ; [dmbld]
  | [W] ; po ; [dmbst]
  | [dmbst]; po; [W]
  | [dmbld]; po; [R|W]
  | [L]; po; [A]
  | [A | Q]; po; [R | W]
  | [R | W]; po; [L]
  | [F | C]; po; [dsbsy]
  | [dsb] ; po

(* Ordered-before *)
let ob = (obs | dob | aob | bob
  | iio | tob | obtlbi | ctxob | obfault |
     obETS)^+

(* Internal visibility requirement *)
acyclic po-loc | fr | co | rf as internal
(* External visibility requirement *)
irreflexive ob as external
(* Atomic requirement *)
empty rmw & (fre; coe) as atomic
(* Writes cannot forward to po-future
     translates *)
acyclic (po-pa | trfi) as translation-internal
```

Precise model

Simner, ..., P-P, ... '22

(being incorporated in authoritative doc)

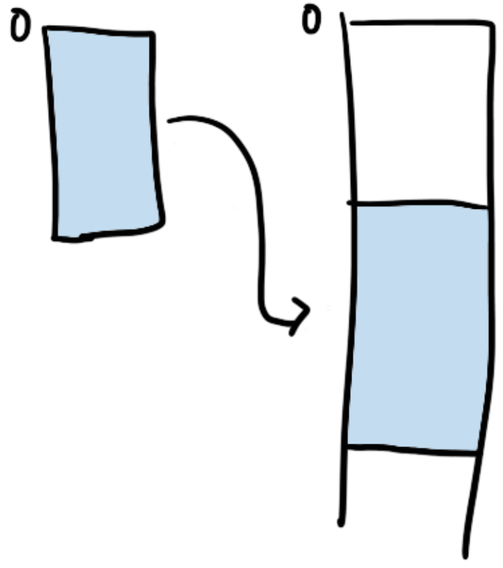
No need to know details of h/w for page table manipulation

in the OS/hypervisor ...

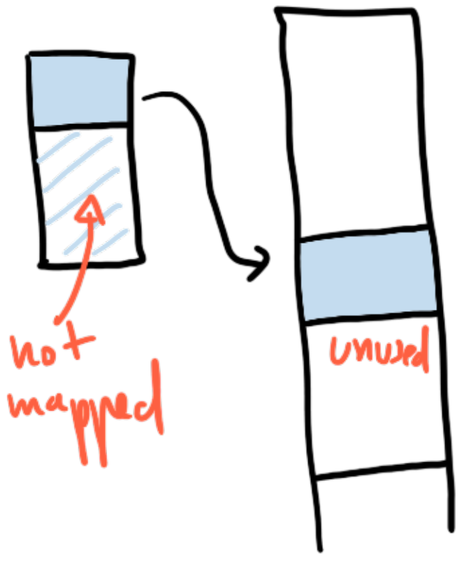
What is an OS/hypervisor? An exception handler

Paging on demand: apps are greedy → only give them physical space when they use it

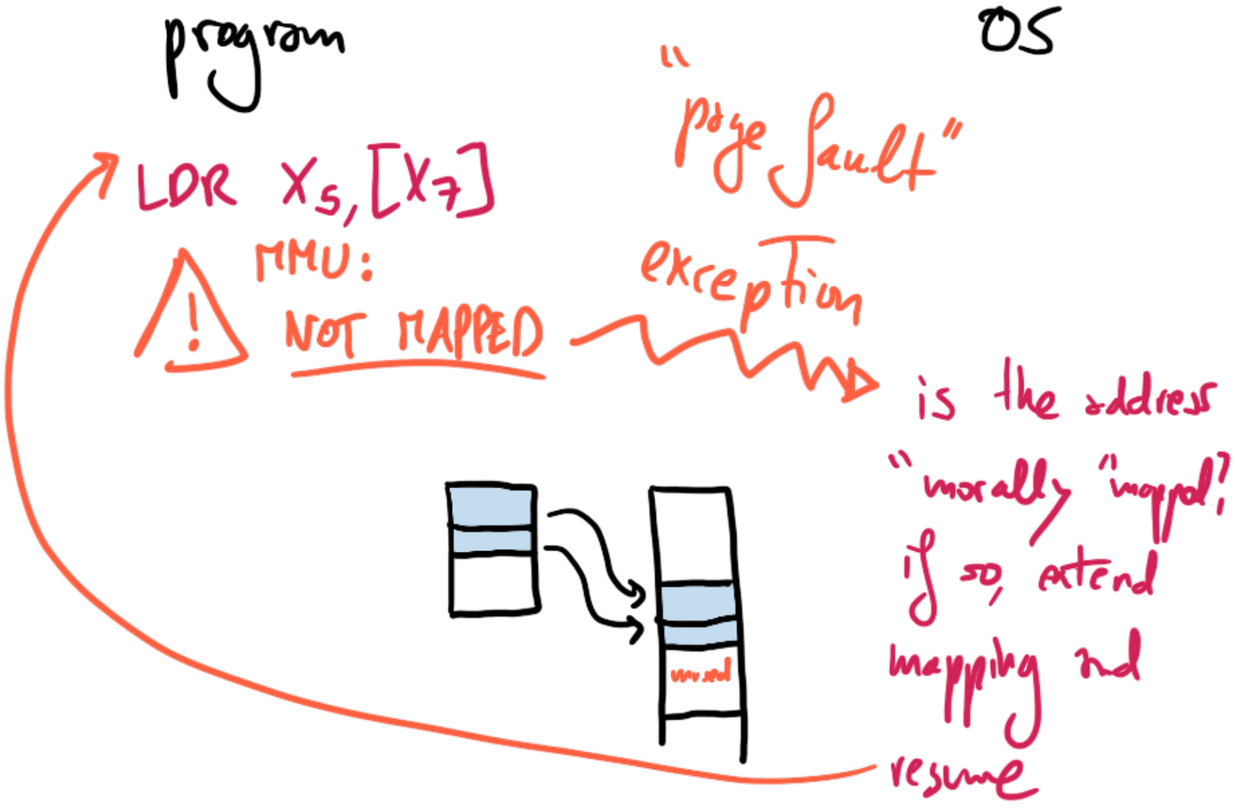
"moral" mapping



actual mapping



Also, CoW



```
"Arm-A exceptions"

include "cos.cat"
include "arm-common.cat"

(* might-be speculatively
     executed *)
let speculative =
    ctrl
  | addr; po
  | if "SEA_R" then [R]; po
      else 0
  | if "SEA_W" then [W]; po
      else 0

(* context-sync-events *)
let CSE =
    ISB
  | if "FEAT_ExS" & ~"EIS"
     then 0 else TE
  | if "FEAT_ExS" & ~"EOS"
     then 0 else ERET

let ASYNC =
  TakeInterrupt

(* observed by *)
let obs = rfe | fr | co

(* dependency-ordered-
     before *)
let dob =
    addr | data
  | speculative ; [W]
  | speculative ; [ISB]
  | (addr | data); rfi

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

(* barrier-ordered-before
     *)
let bob =
    [R] ; po ; [dmbld]
  | [W] ; po ; [dmbst]
  | [dmbst]; po; [W]
  | [dmbld]; po; [R|W]
  | [L]; po; [A]
  | [A | Q]; po; [R | W]
  | [R | W]; po; [L]
  | [dsb]; po

(* contextually-ordered-
     before *)
let ctxob =
    speculative; [MSR|CSE]
  | [MSR]; po; [CSE]
  | [CSE]; po

(* async-ordered-before *)
let asyncob =
    speculative; [ASYNC]
  | [ASYNC]; po

(* Ordered-before *)
let ob = (obs | dob | aob |
  bob | ctxob | asyncob)+

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

(* External visibility
     requirement *)
irreflexive ob as external

(* Atomic: Basic LDXR/STXR
     constraint to forbid
     intervening writes. *)
empty rmw & (fre; coe) as
     atomic
```

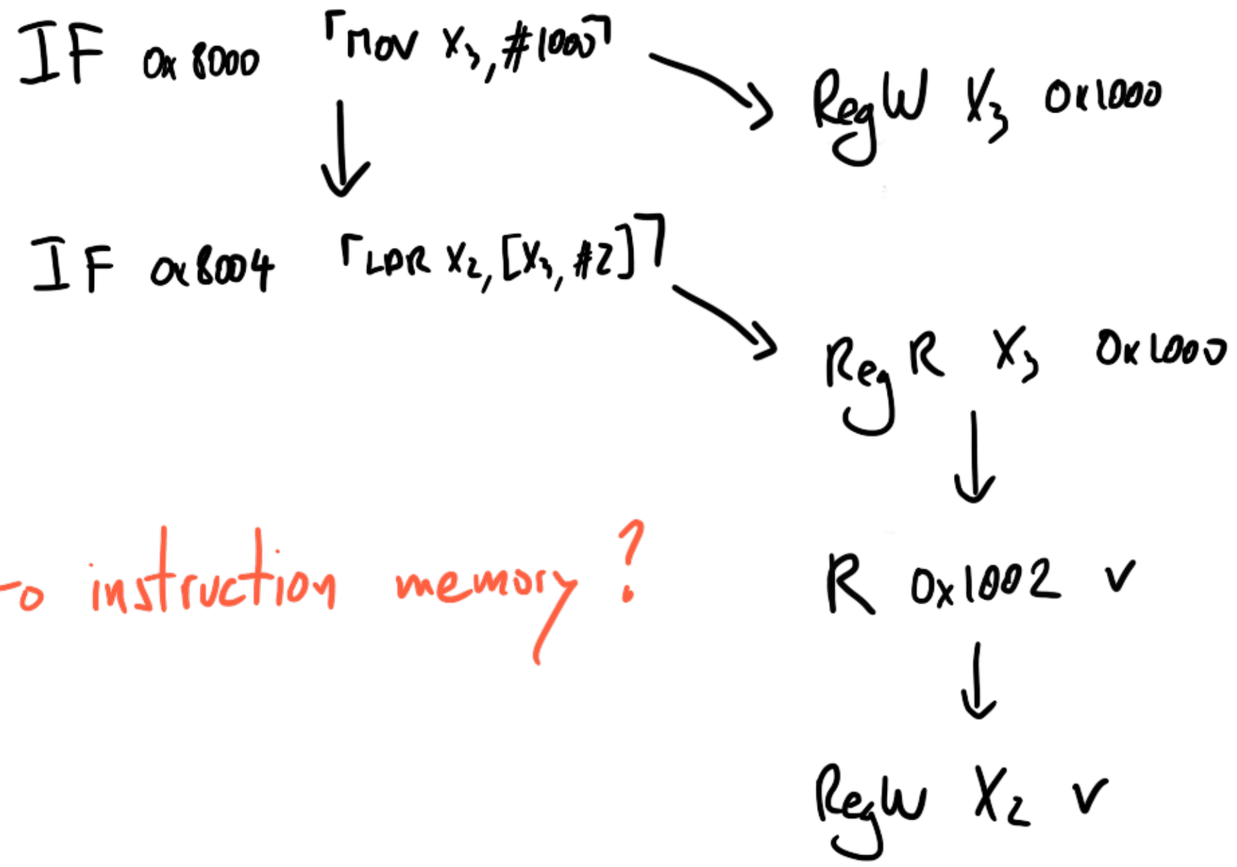
Precise model

Simner, .., P-P, .. in submission

↳ we are now pretty well equipped to consider OS/hypervisor code!

What about instruction fetches?

```
MOV X3, #1000
LDR X2, [X3, #2]
```



What if you write to instruction memory?

Simner, ..., P-P, ... '20

Precise model

(being incorporated)

```
let iseq = [W];(wco&scl);[DC];          (*1*)
               (wco&scl);[IC]

(* Observed-by *)
let obs = rfe | fr | wco                (*2*)
        | irf | (ifr;iseq)              (*3,4*)

(* Fetch-ordered-before *)
let fob = [IF]; fpo; [IF]               (*5*)
  | [IF]; fe                            (*6*)
  | [ISB]; fe^-1; fpo                   (*7*)

(* Dependency-ordered-before *)
let dob = addr | data
  | ctrl; [W]
  | (ctrl | (addr; po)); [ISB]
(*| [ISB]; po; [R] *)                   (*8*)
  | 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]        (*9*)
  | [dsb.ish]; po; [R|W|F|DC|IC]        (*10*)
  | [dmb.sy]; po; [DC]                  (*11*)

(* Cache-op-ordered-before *)
let cob = [R|W]; (po&scl); [DC]         (*12*)
  | [DC]; (po&scl); [DC]                (*13*)

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

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

(* External visibility requirement *)
irreflexive ob as external

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

(* Constrained unpredictable *)
let cff = ([W];loc;[IF]) \              (*14*)
            ob^-1 \ (co;iseq;ob)
cff_bad cff ≡ CU                        (*15*)
```

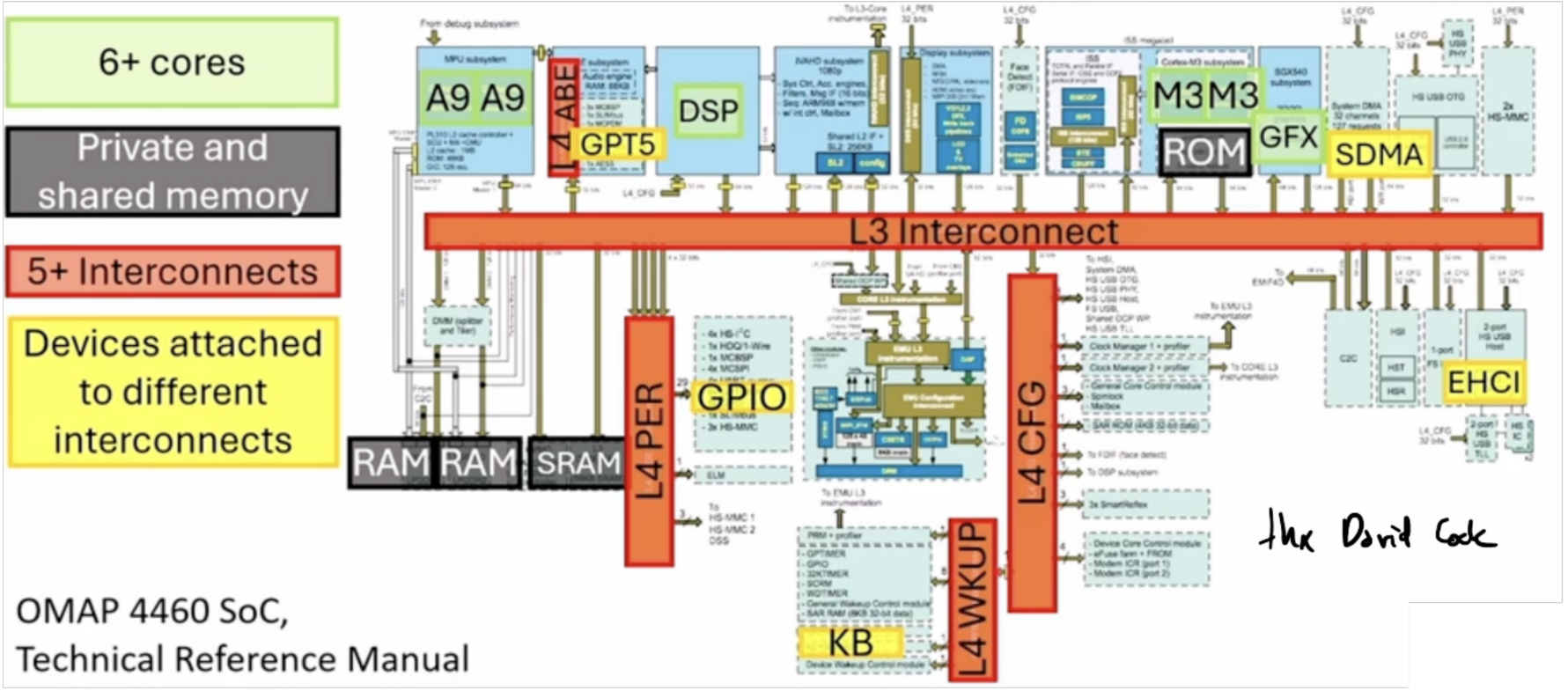
Are we done?

Nowhere near!

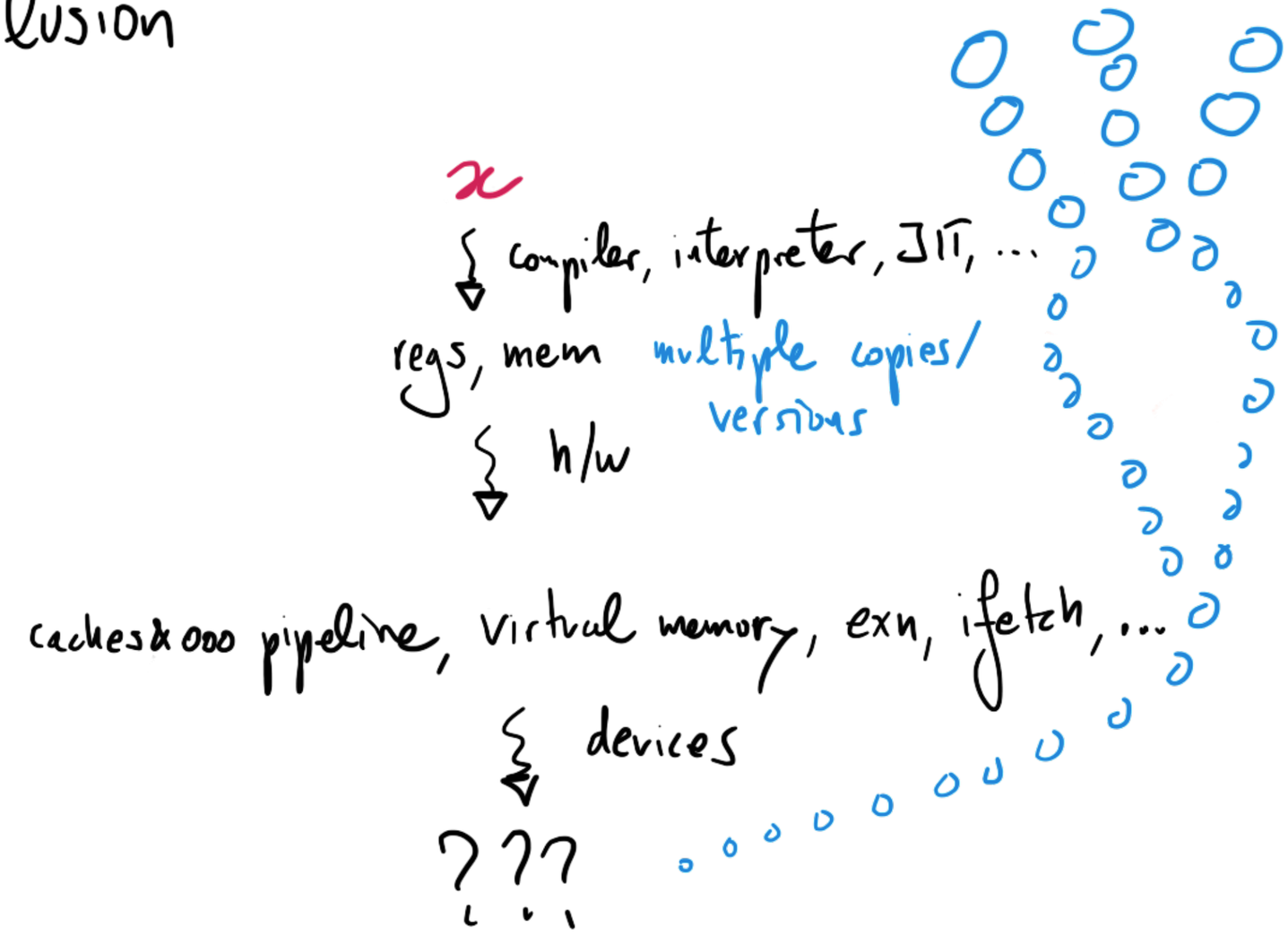
↳ IOMMU for devices

...

An old, simple machine:



Conclusion



leaky abstractions

⇒ observable behaviour that bubbles up