Relaxed Virtual Memory

— Armv8 edition —

Ben Simner[1], Alasdair Armstrong[1], Thibaut Pérami[1]
Jean Pichon-Pharabod[2], Christopher Pulte[1]
Richard Grisenthwaite[3], Peter Sewell[1]

[1] Cambridge
[2] Aarhus
[3] Arm

relaxed Memory?

2005
Java memory model

2009:
ARMv7
+ POWER

2016
Armv8

2019
Armv8
Persistent memory

2022
Armv8
VMSA

... ... ?

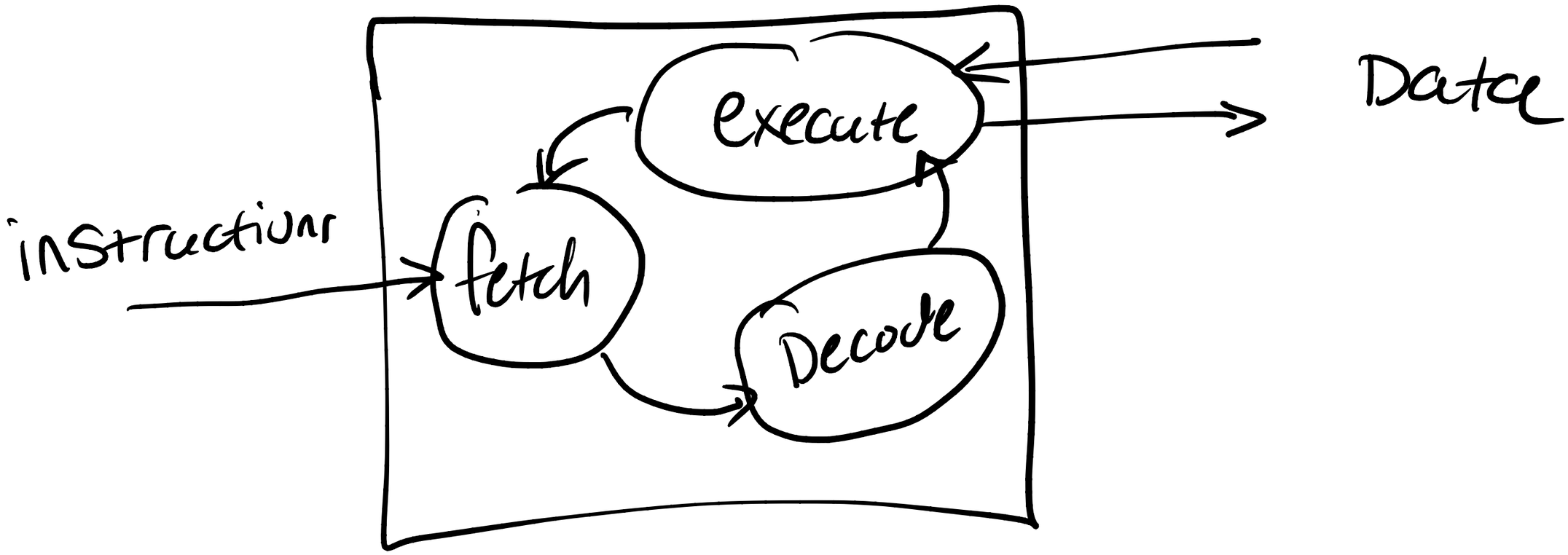
2009
X86-TSO

2011
C/C++

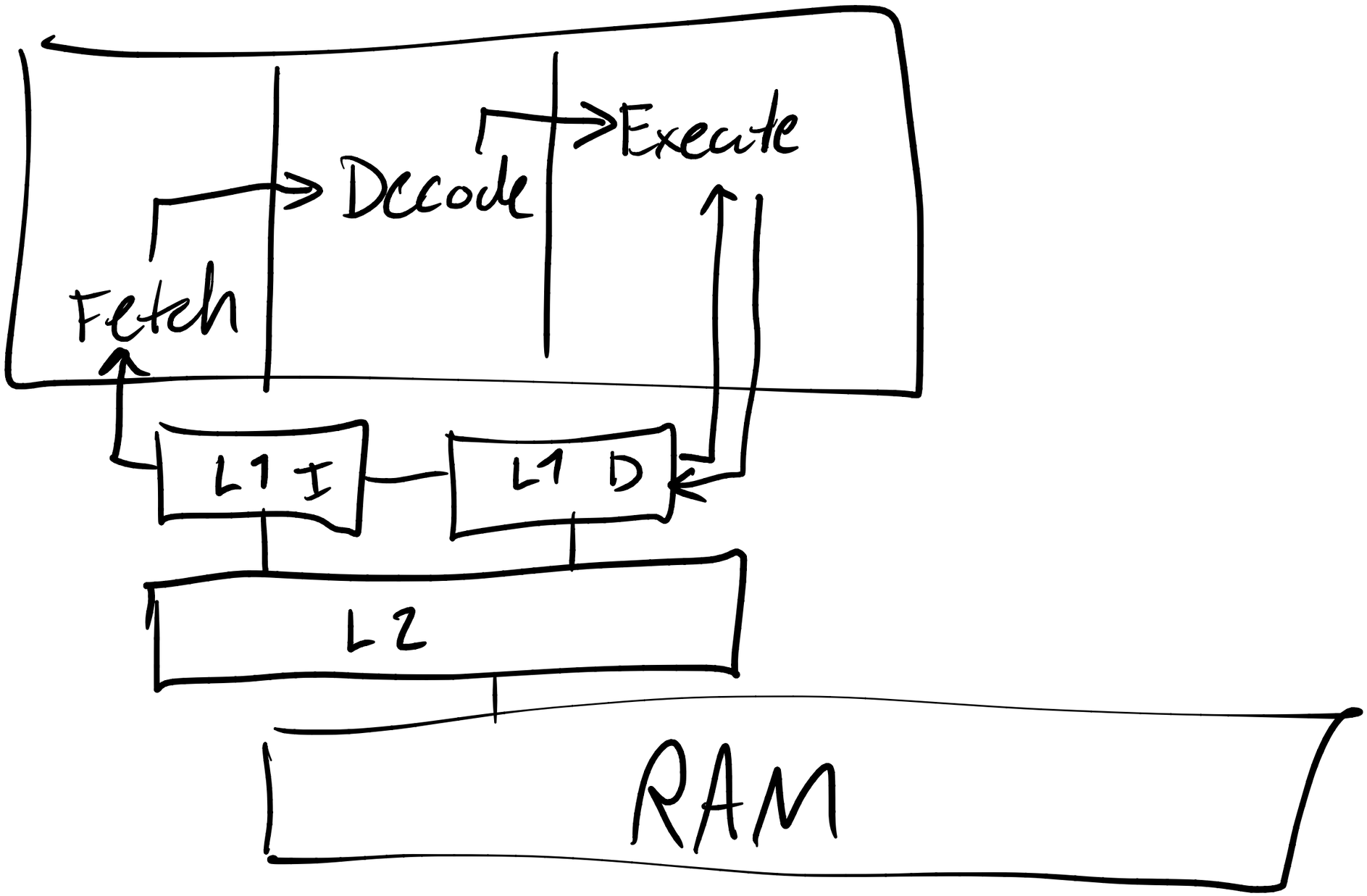
2017
Mixed Size
ARMv8

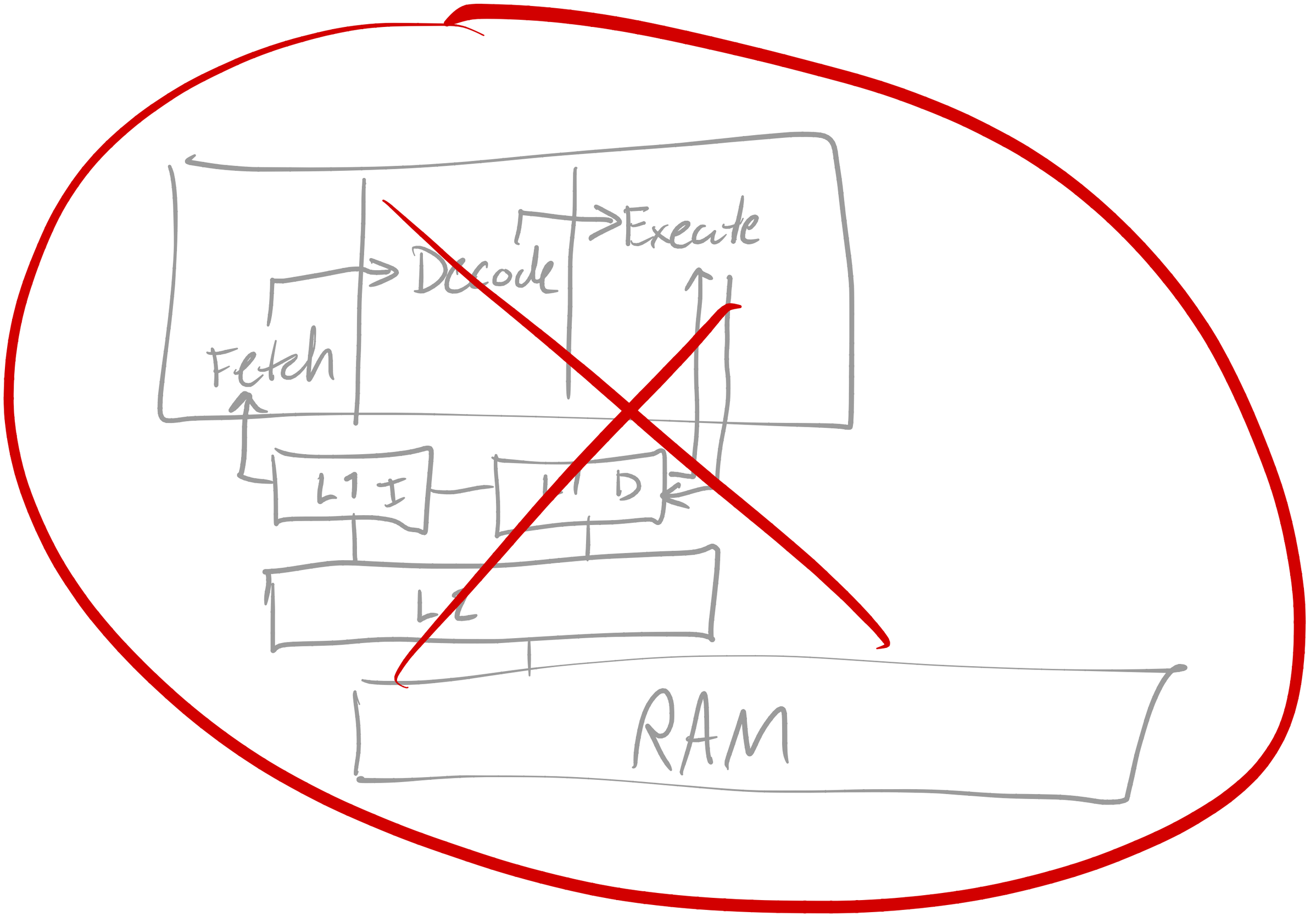
2020
Armv8 ifetch

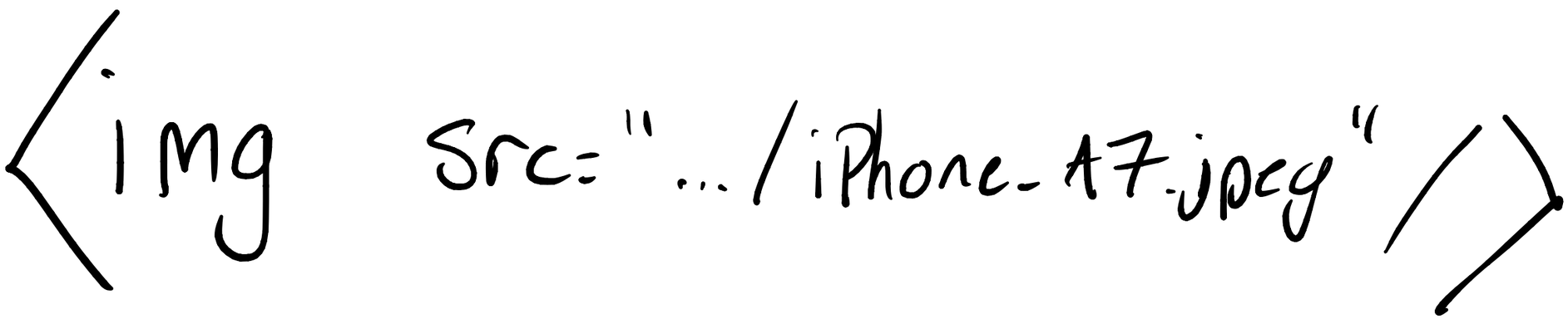
Hardware

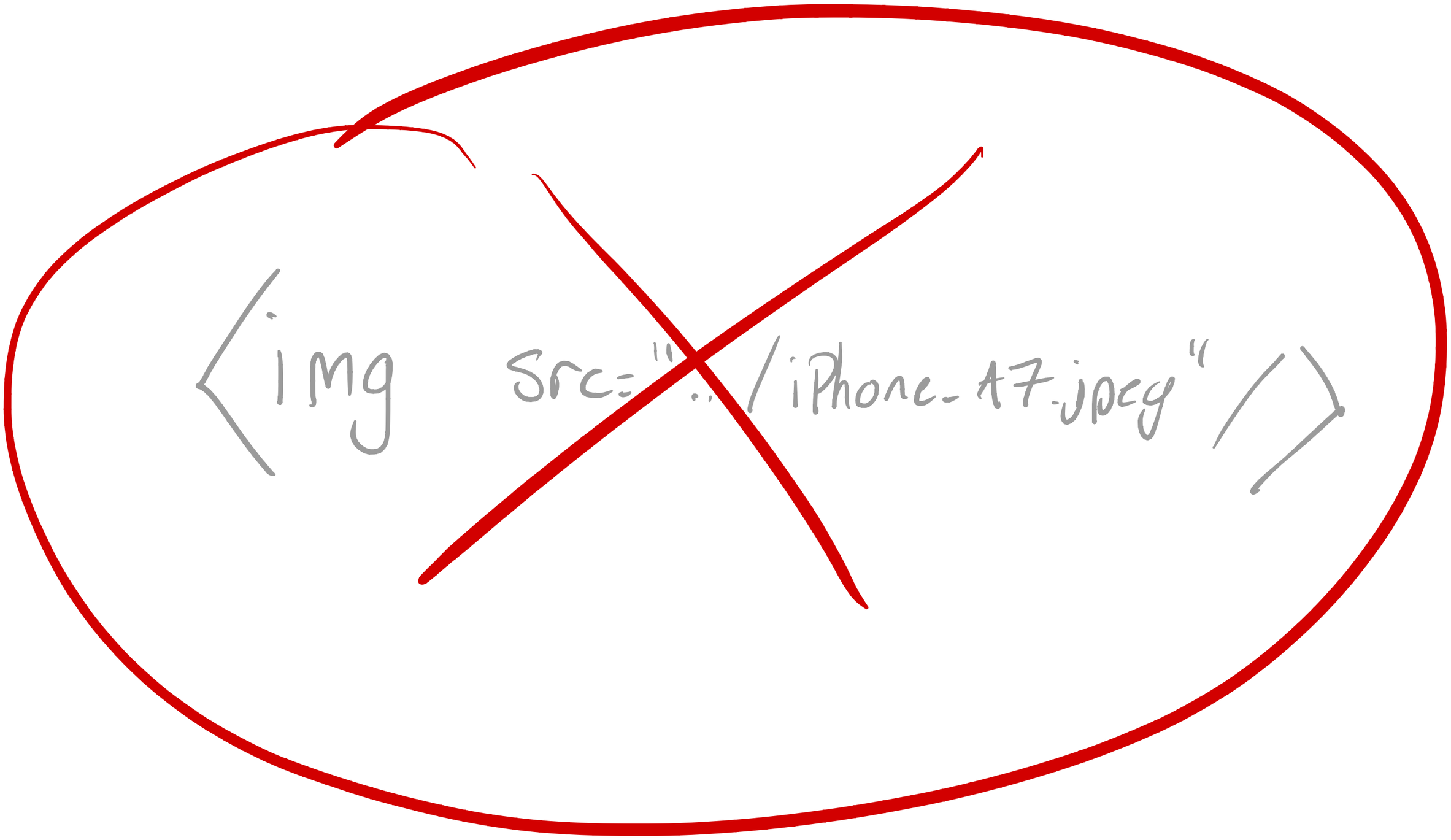


A Better Cartoon?




Fetch
Decode
Execute
L1 I
L1 D
L2
RAM

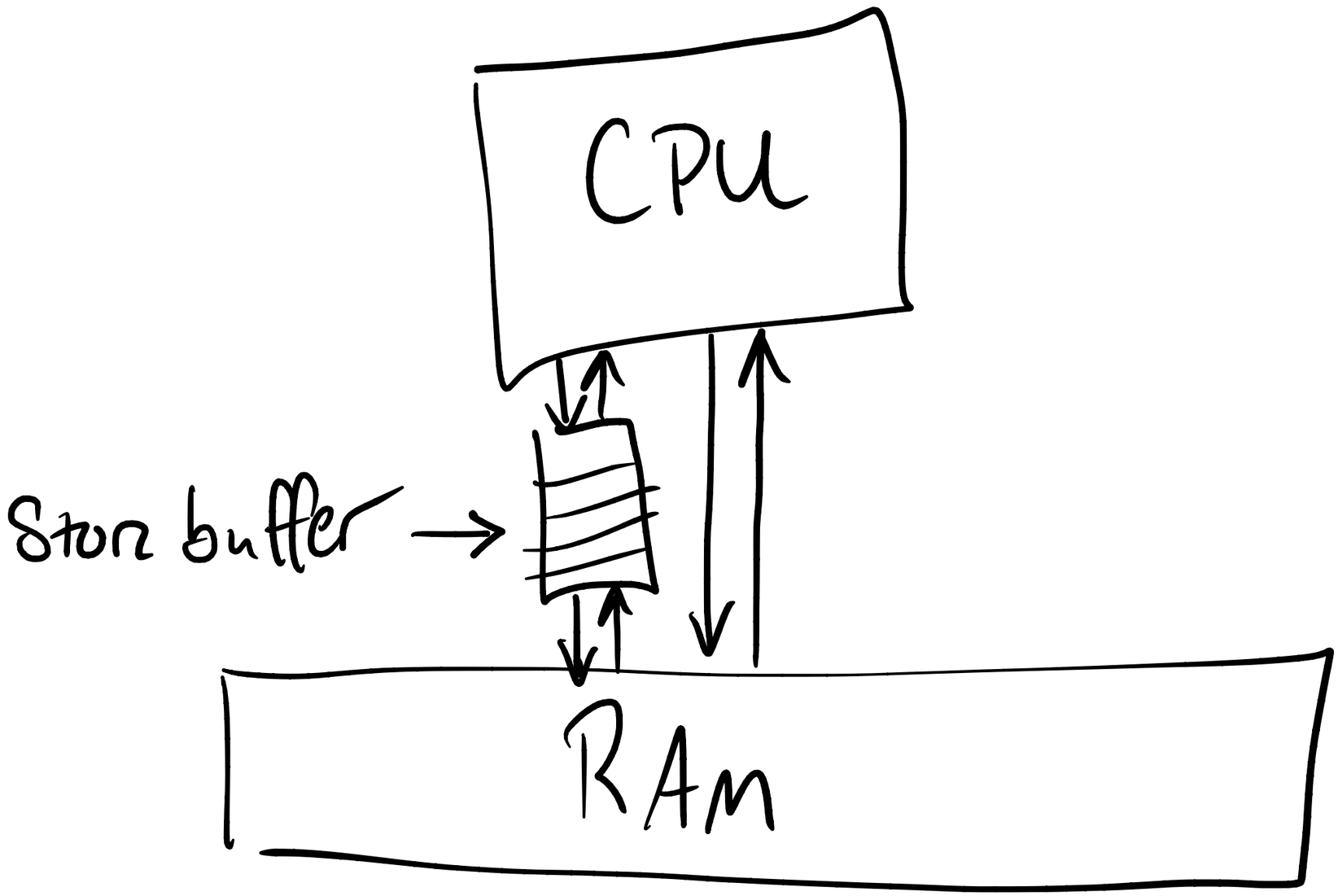


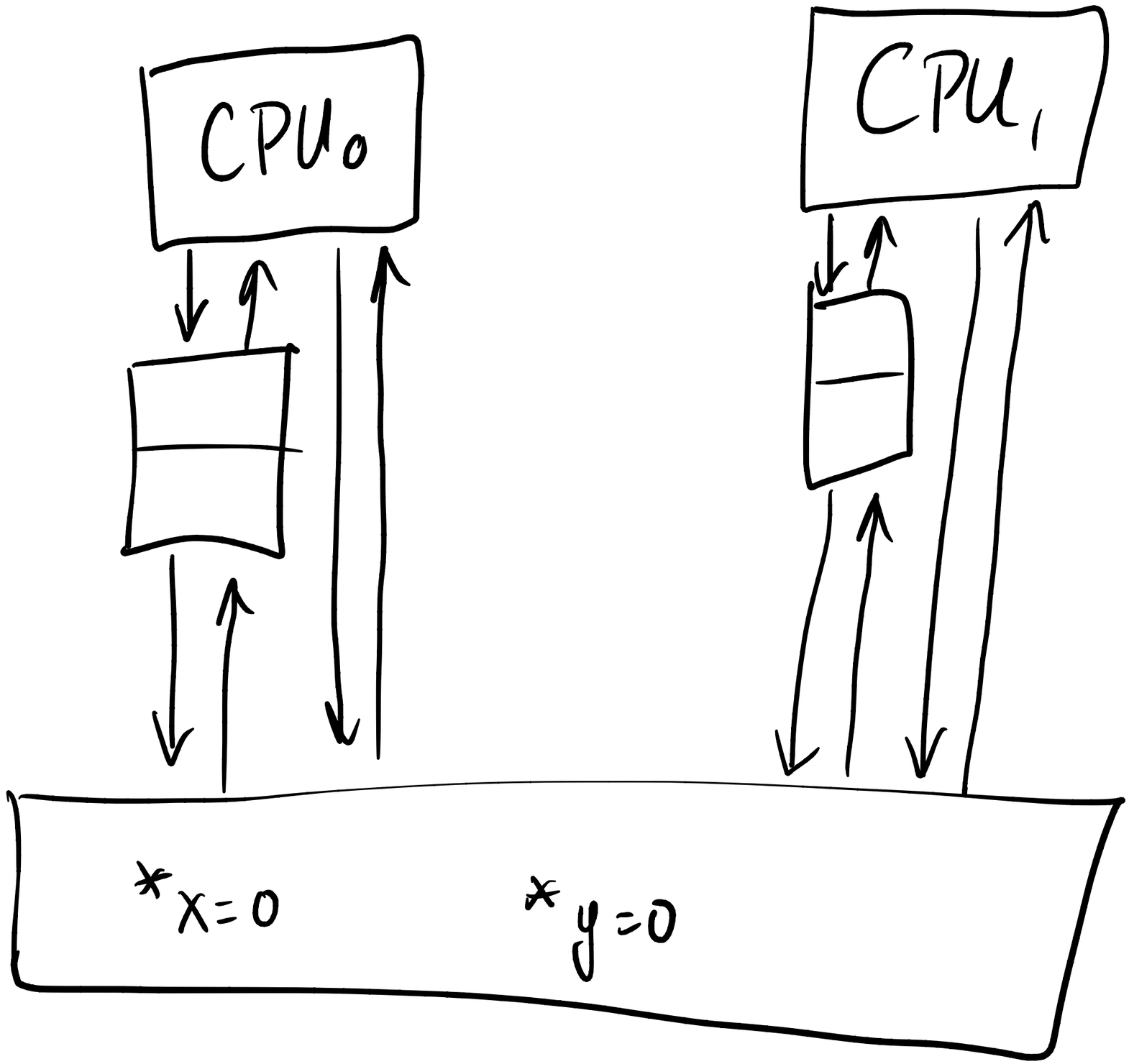


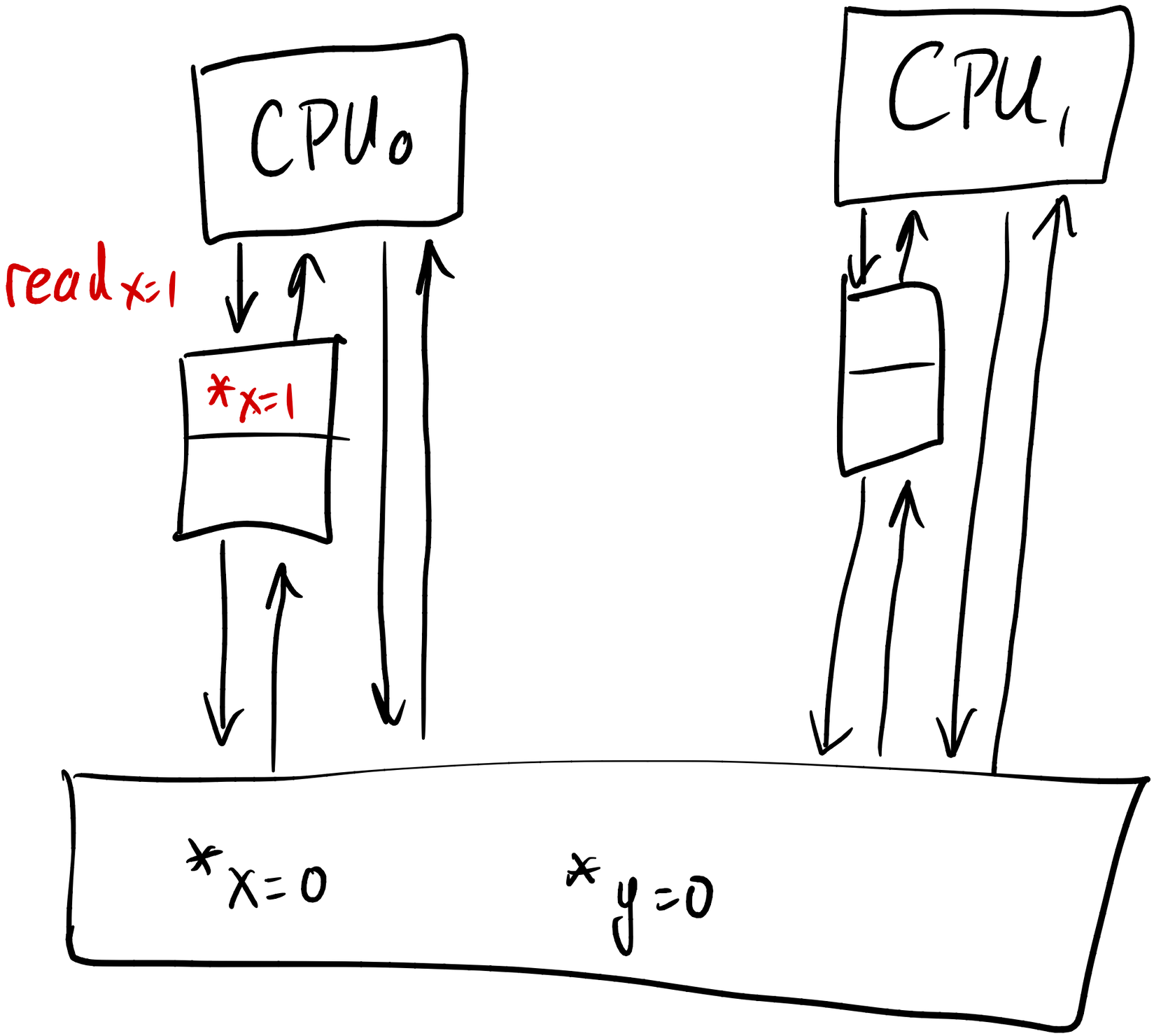
Programmer-visible behaviour

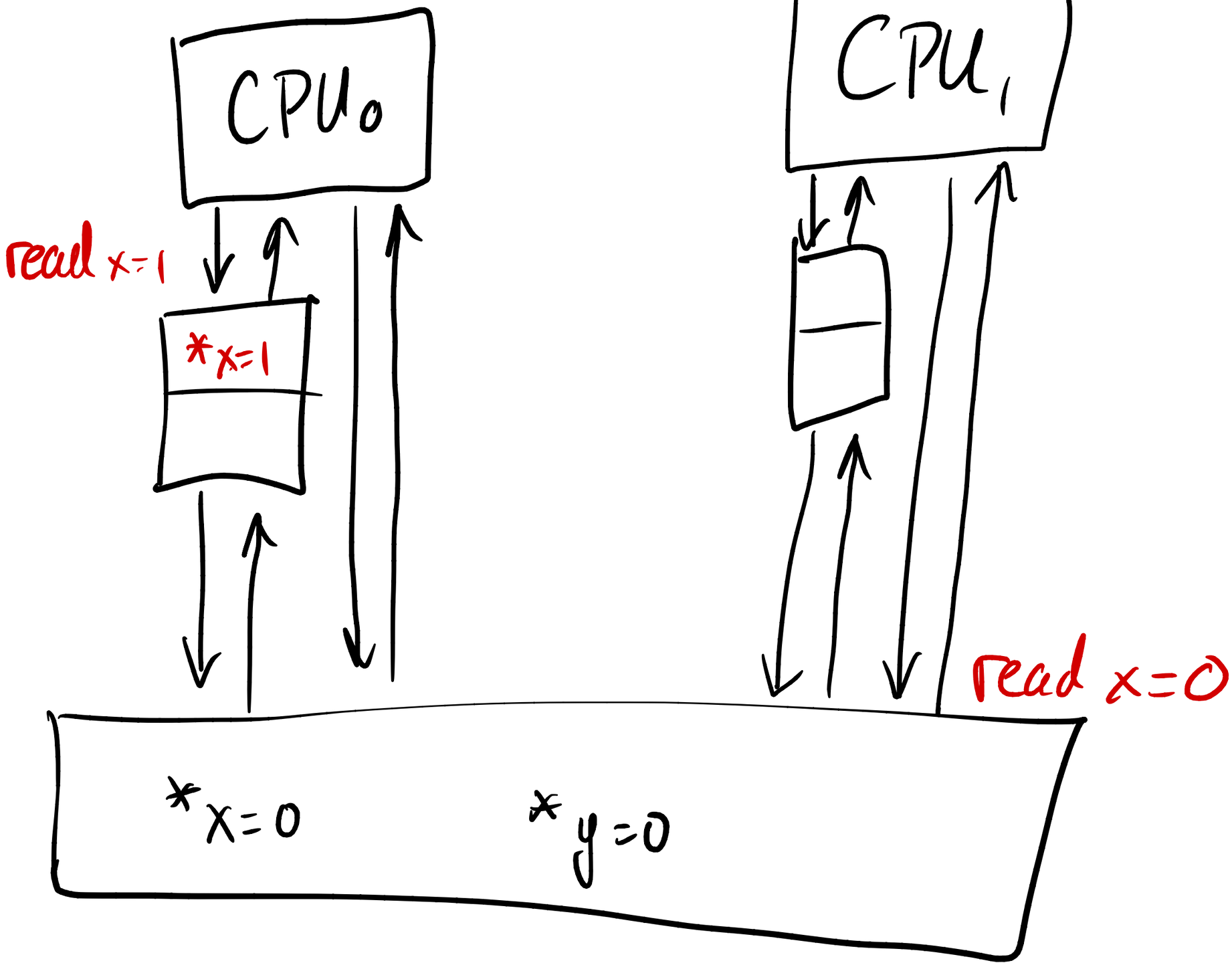
```
for (int i=0; i < N; i++) {
    buf[i] = ...;
}
```

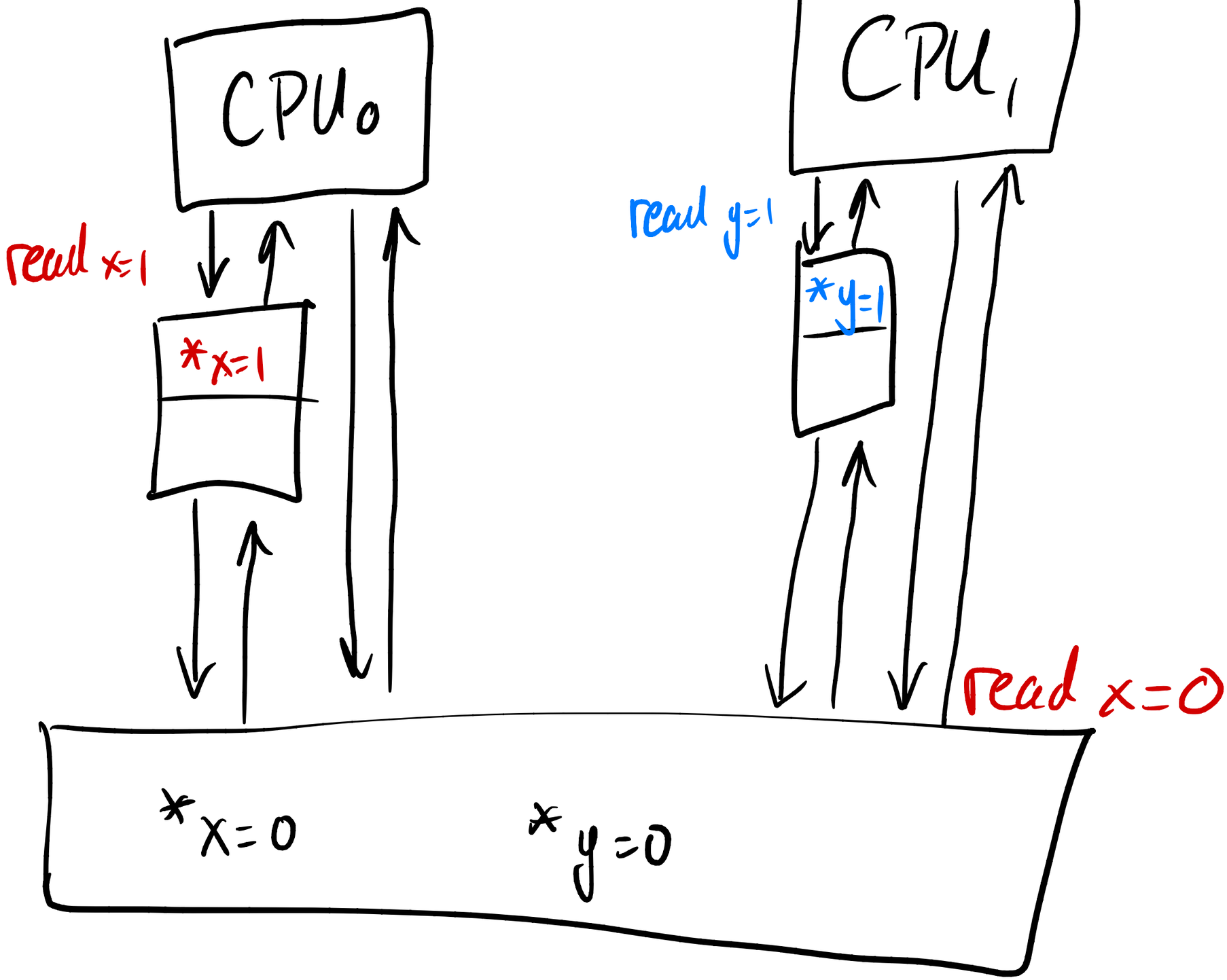
Programmer-visible behaviour

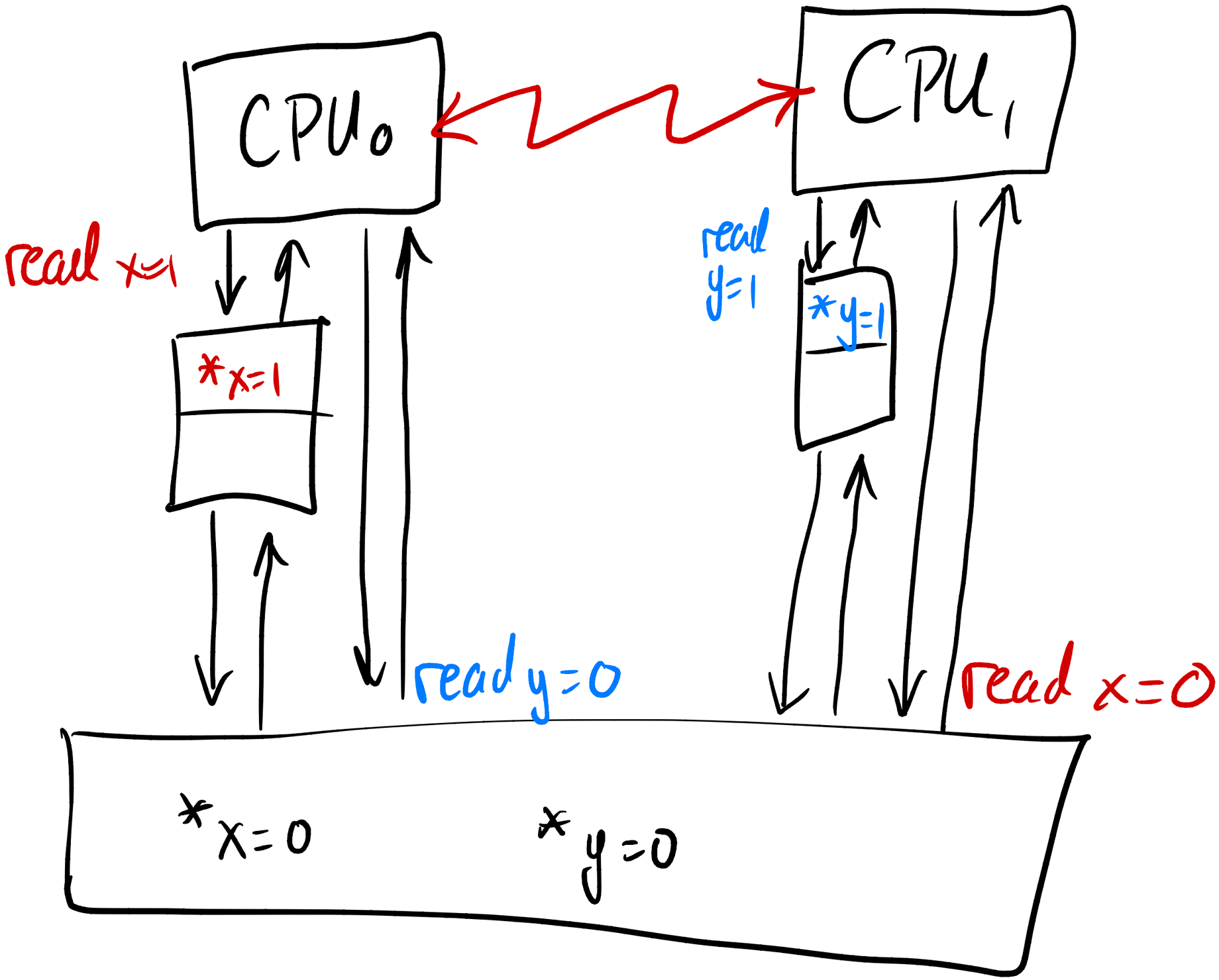



CPU0
CPU1
*x=0
*y=0


CPU0
read x=1
*x=1
CPU1
*x=0
*y=0


CPU0
CPU1
read x=1
*x=1
read x=0
*x=0
*y=0

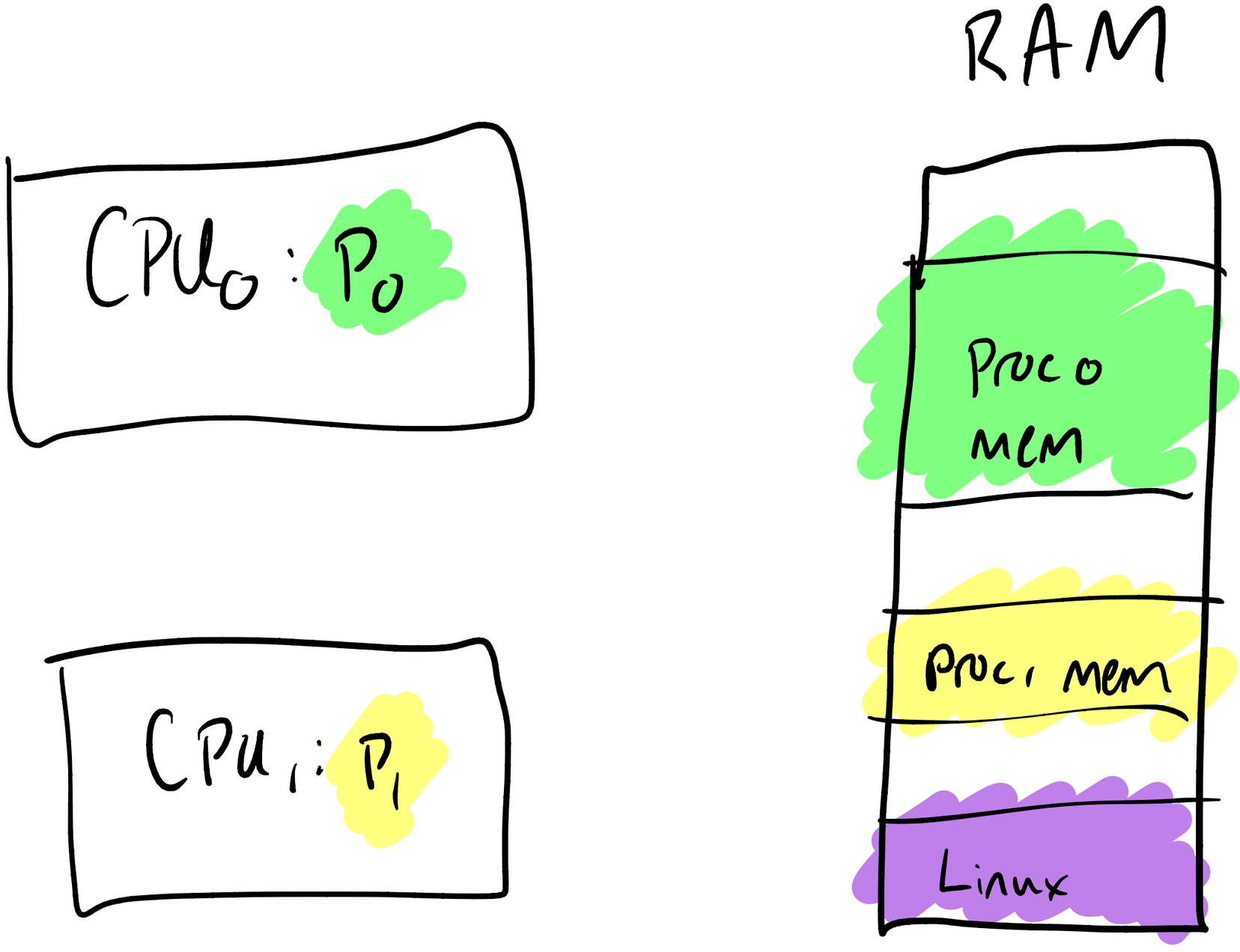

CPU0
read x=1
*x=1
CPU1
read y=1
*y=1
read x=0
*x=0
*y=0

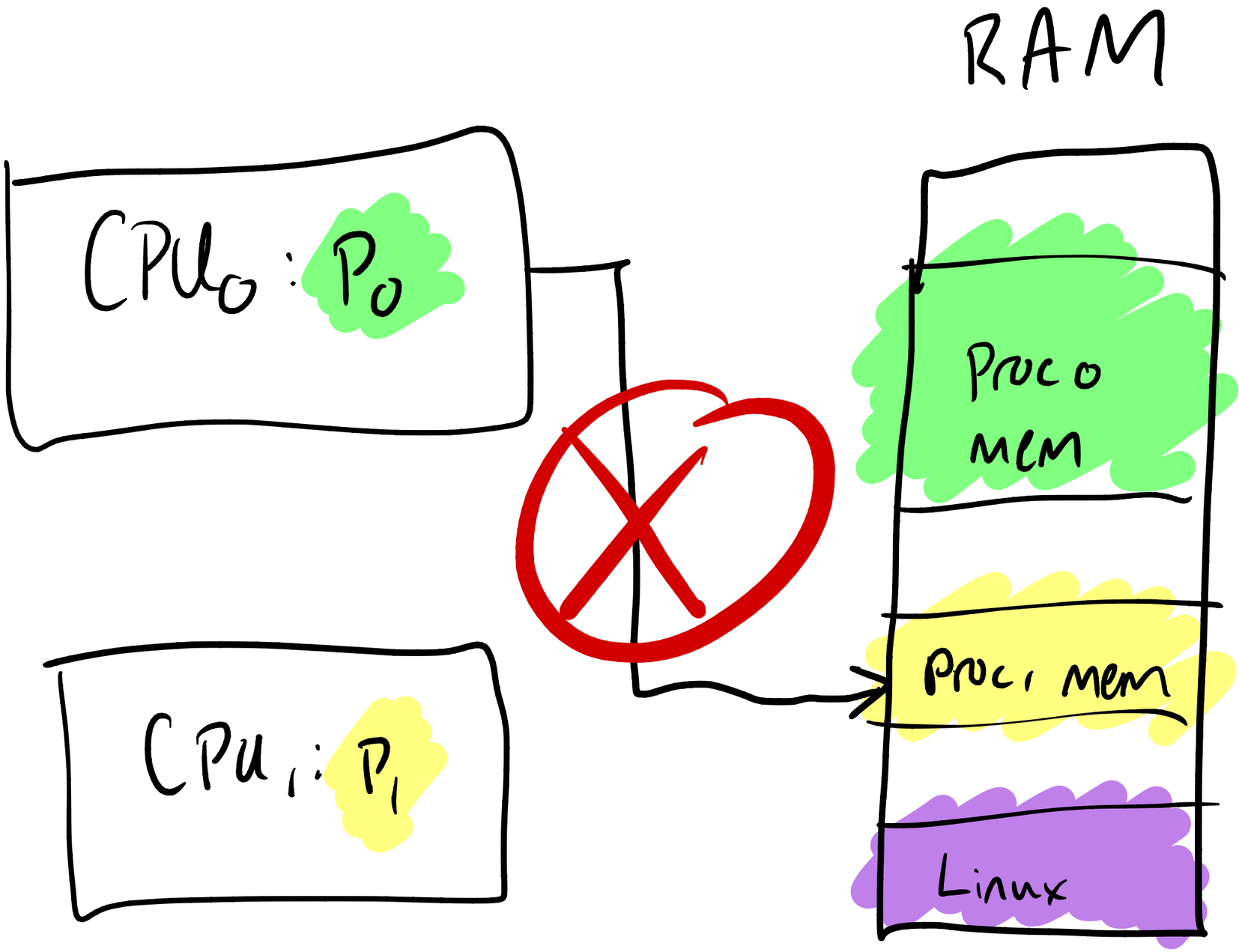

CPU0
CPU1
read x=1
read y=1
*x=1
*y=1
read y=0
read x=0
*x=0
*y=0

...2022:

Virtual

Memory


RAM
CPU0 : P0
Proc 0 mem
CPU1 : P1
Proc1 mem
Linux


RAM
CPU0 : P0
Proc 0 mem
CPU1 : P1
Proc1 mem
Linux

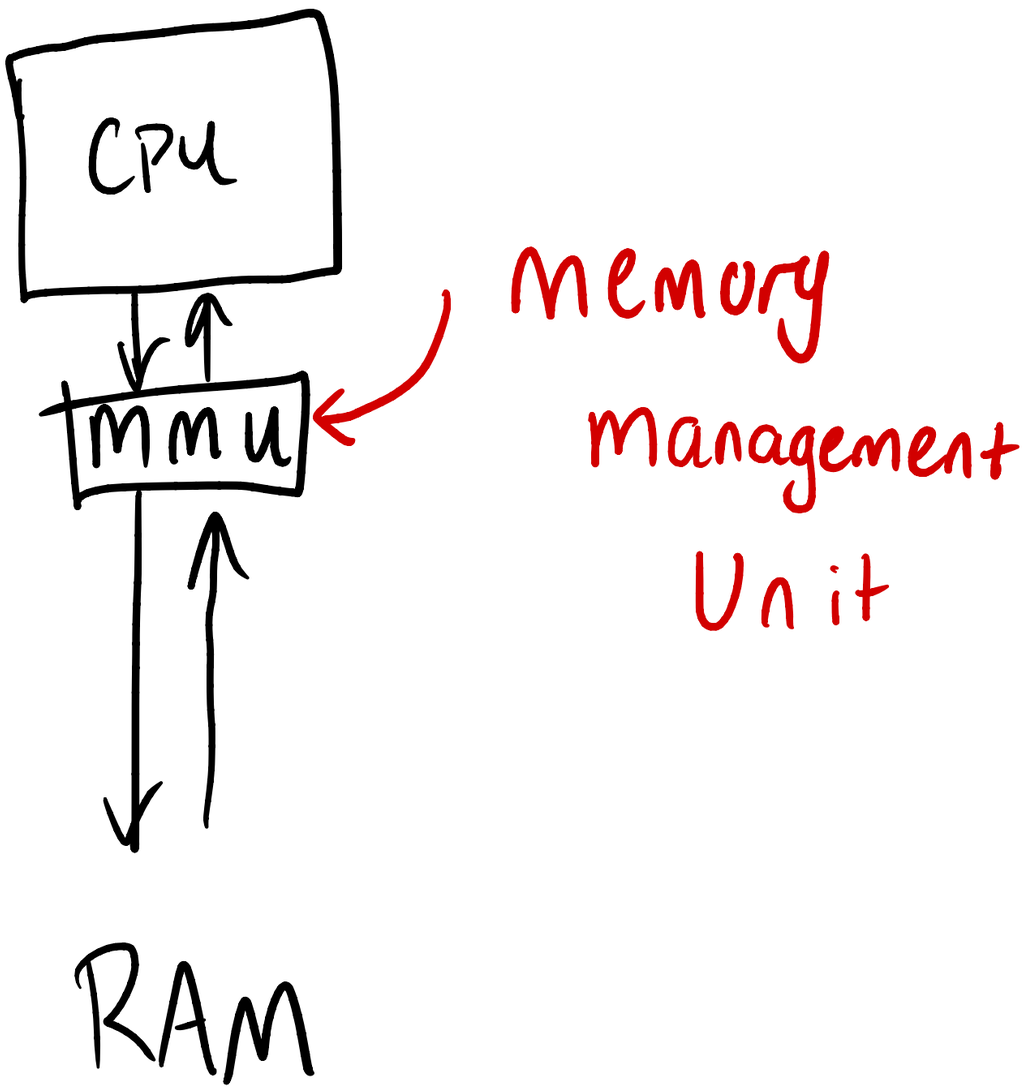
Aims

1. formal Semantics of Programmer-visible behaviours

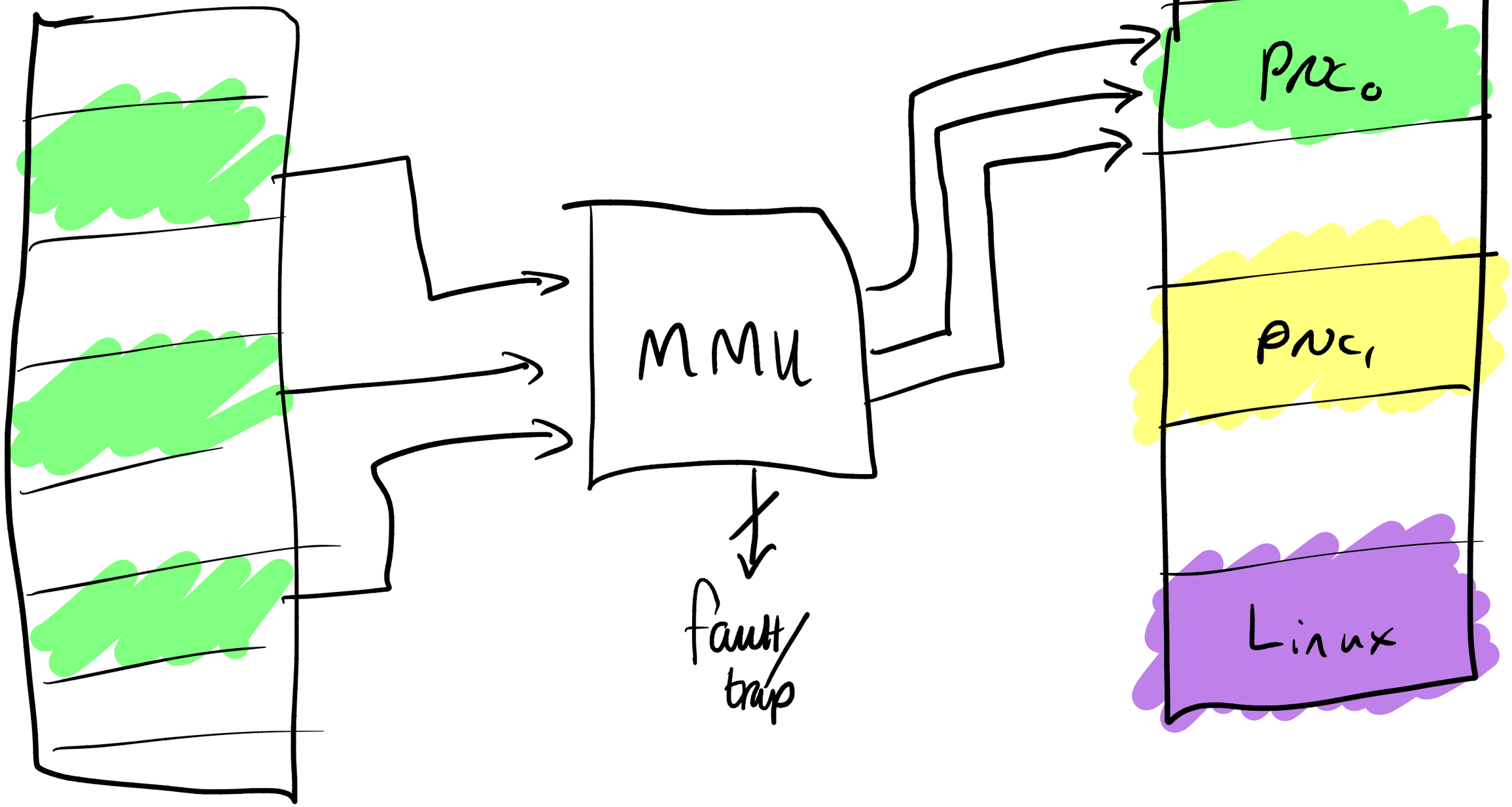
2. build Support tooling

3. use Semantics to reason about code

Memory Management



Proc view



Proc view

virtual addresses

MMU

Physical address

RAM

Proc$_0$

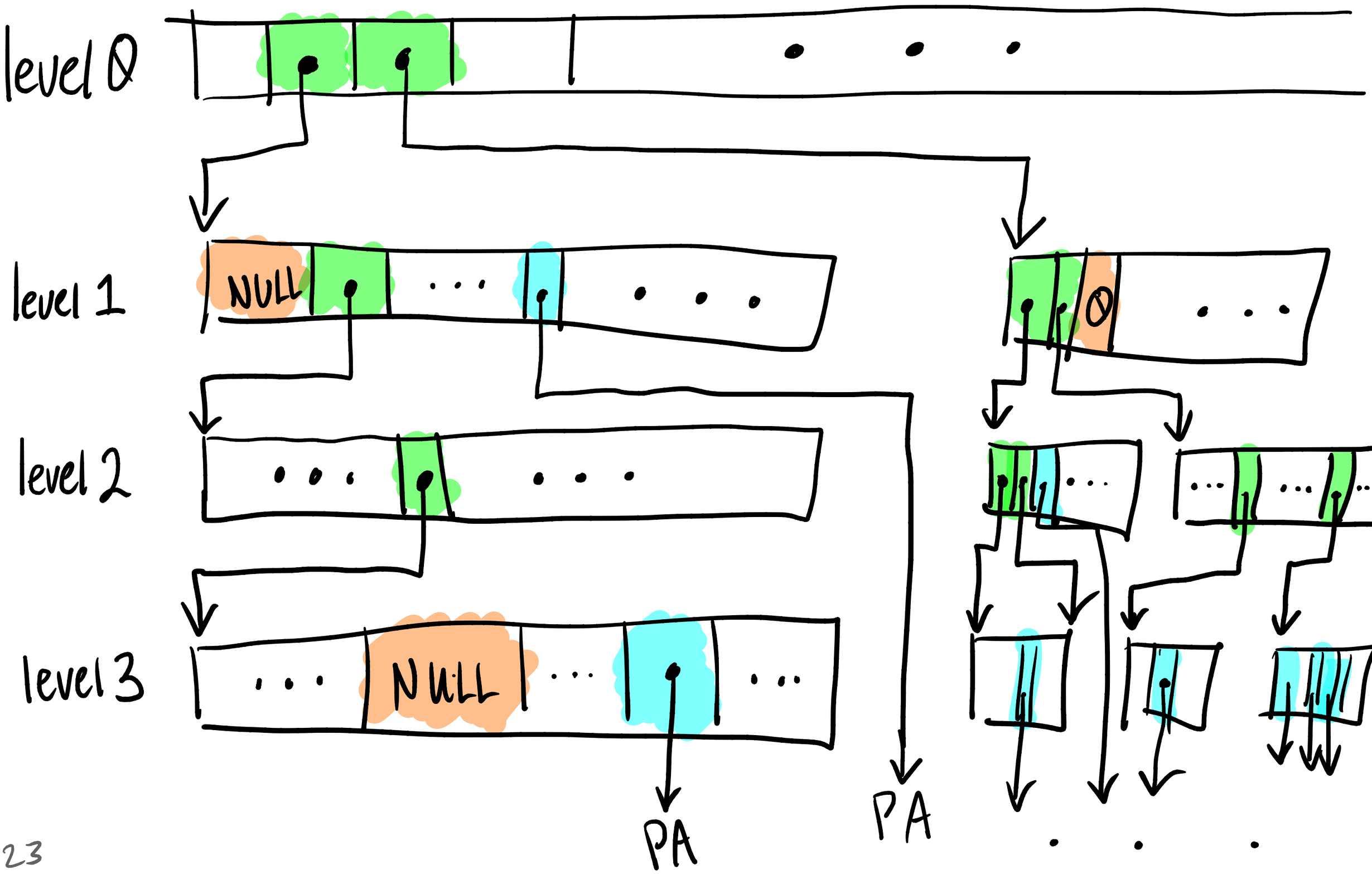
Proc$_1$

Linux

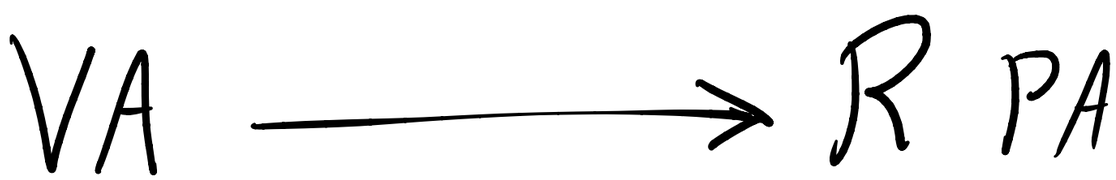
fault/trap

$\nexists va.\ mmu(va) \in Linux$

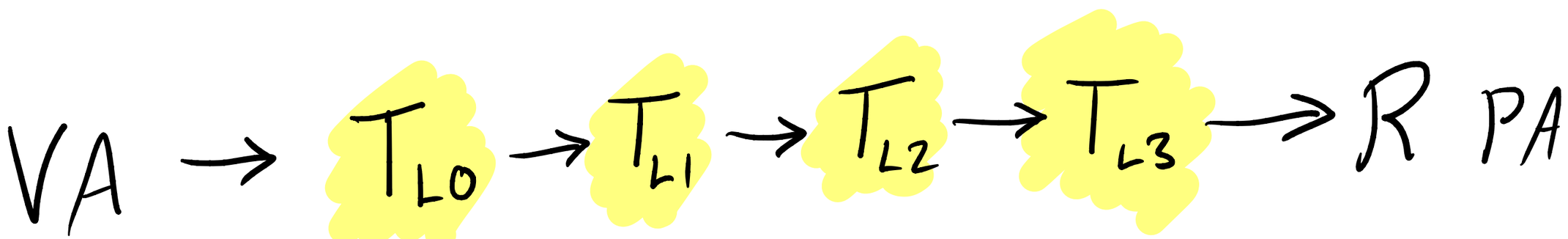
In-memory Data Structure

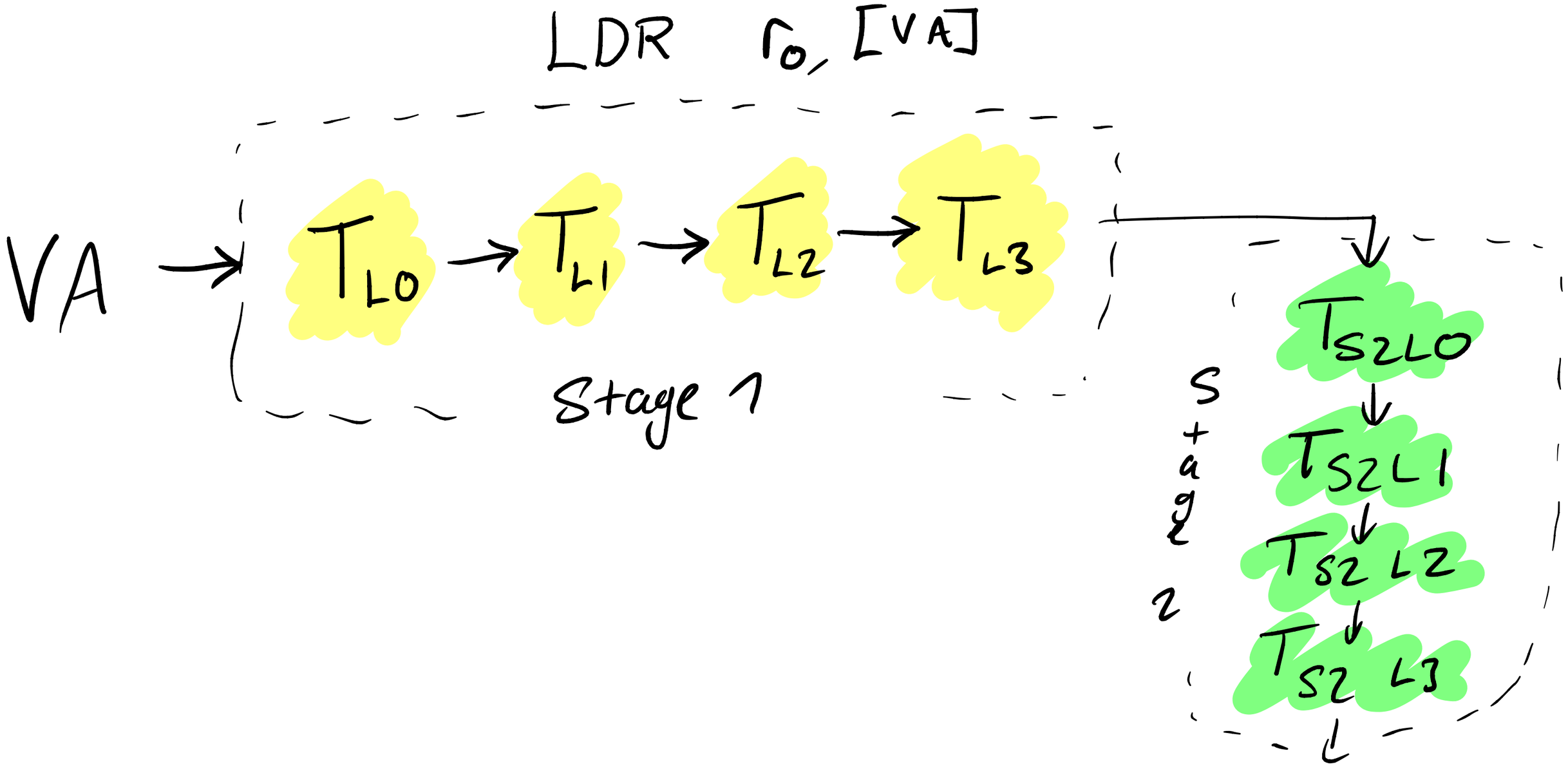


LDR r0, [VA]

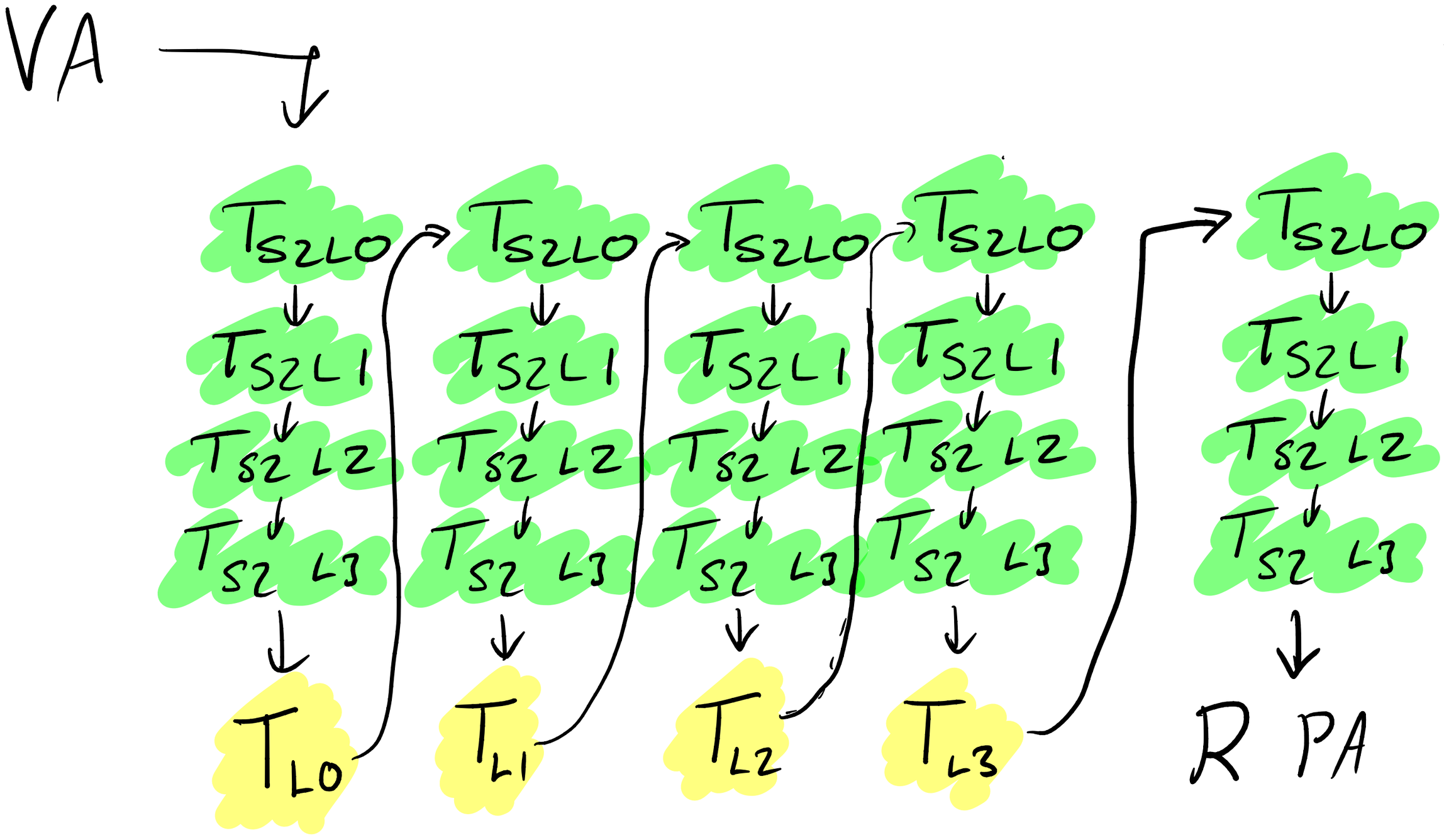


LDR r0, [VA]

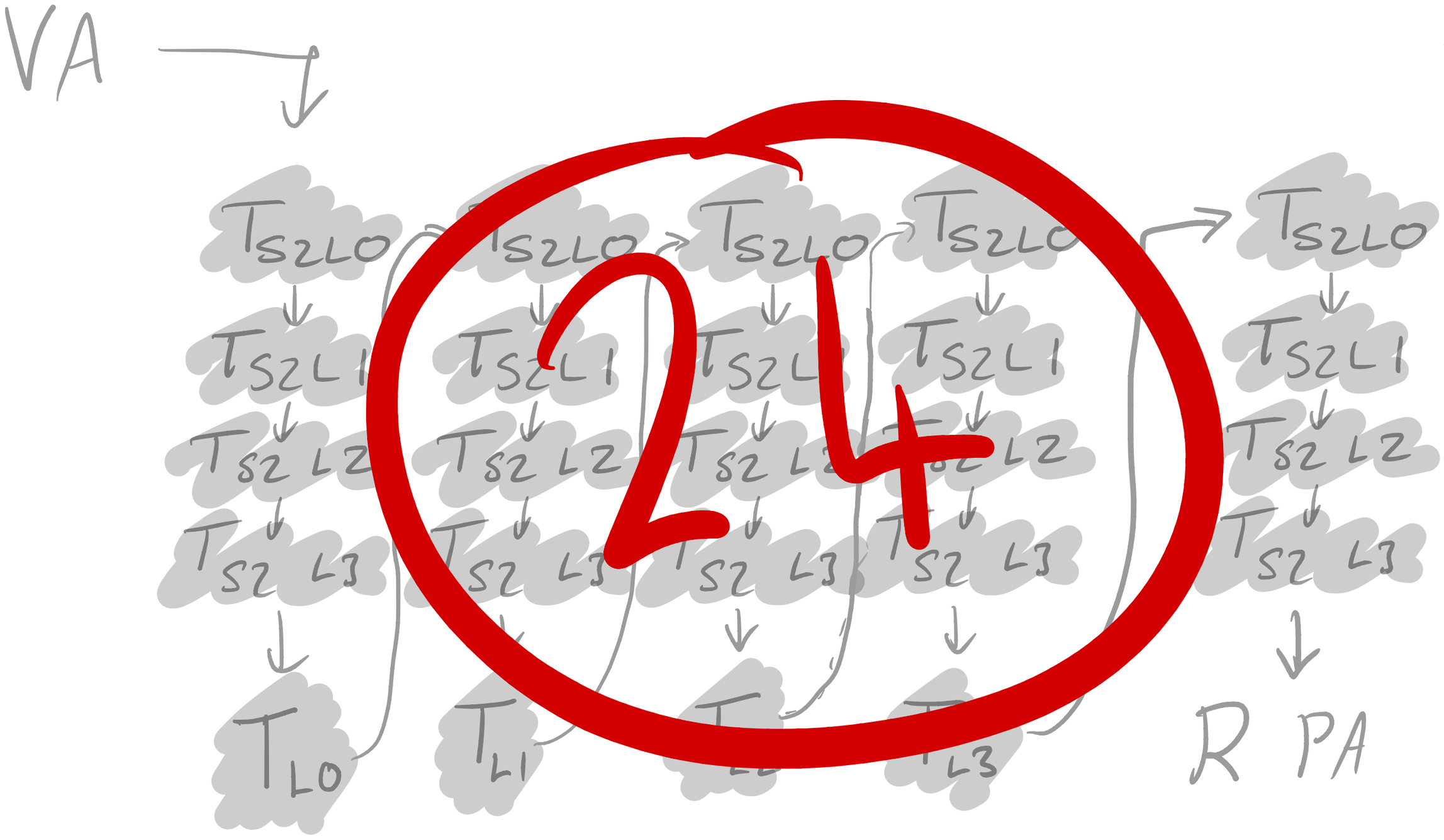


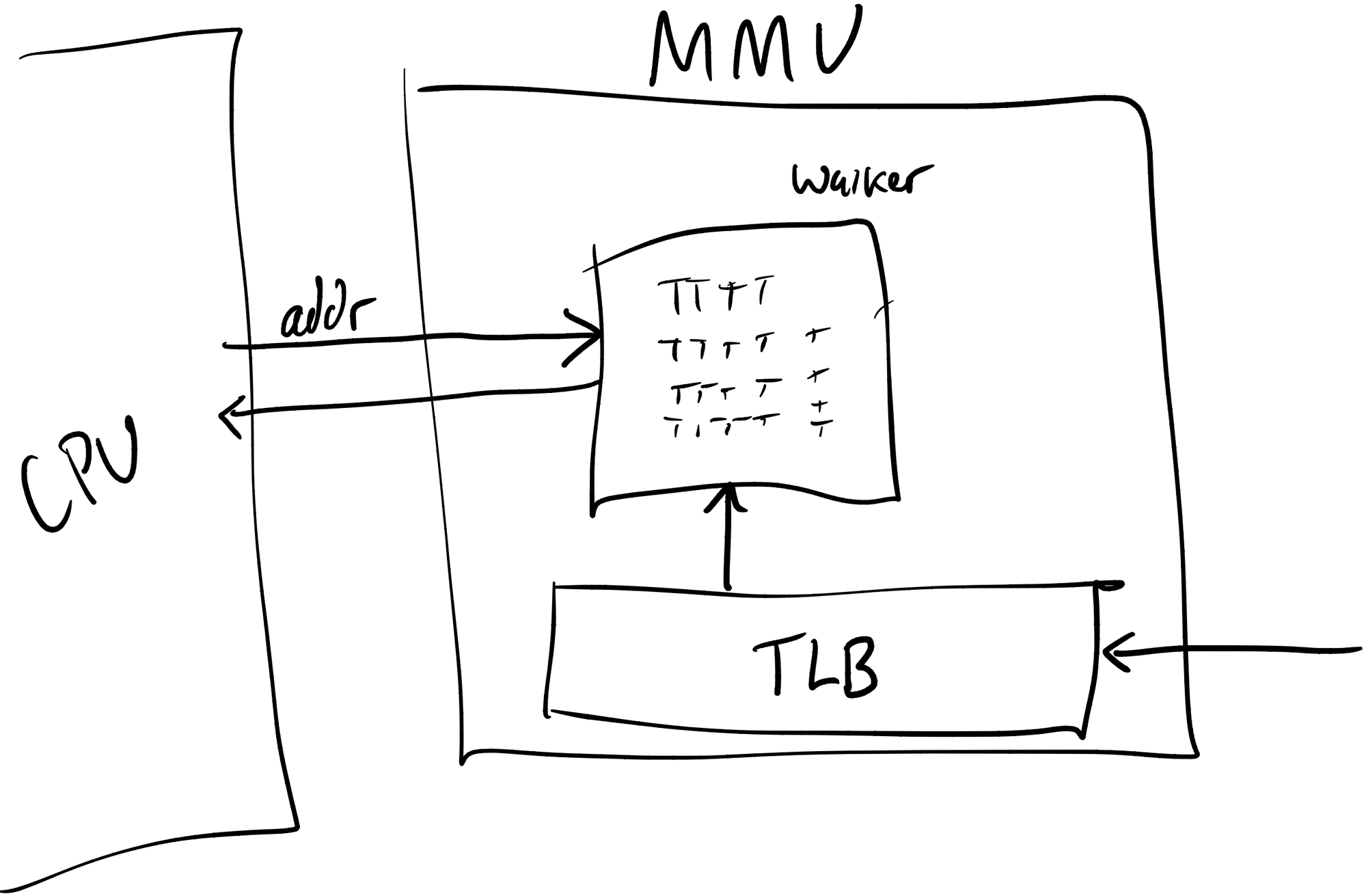

LDR r0, [VA]
VA
TL0
TL1
TL2
TL3
Stage 1
TS2L0
TS2L1
TS2L2
TS2L3
Stage 2
RPA

LDR r0, [VA]



LDR r0, [VA]




MMU
Walker
addr
CPU
to
memory
TLB

Modelling: TLBs and walkers

1. Clarify Architecture

2. Extended Arm Axiomatic Model

3. isla-axiomatic tool

Architectural Design Questions

- Physical/virtual Coherence
- Spontaneous translation table walks
- Coherent TLB fills
- Re-ordered translations
- In-order walks
- Speculative translations
- Micro TLBs
- Multi-copy atomicity
- ETS
- Break-before-make
- write forwarding
- TLB Shootdown
- ASIDs and VMIDs

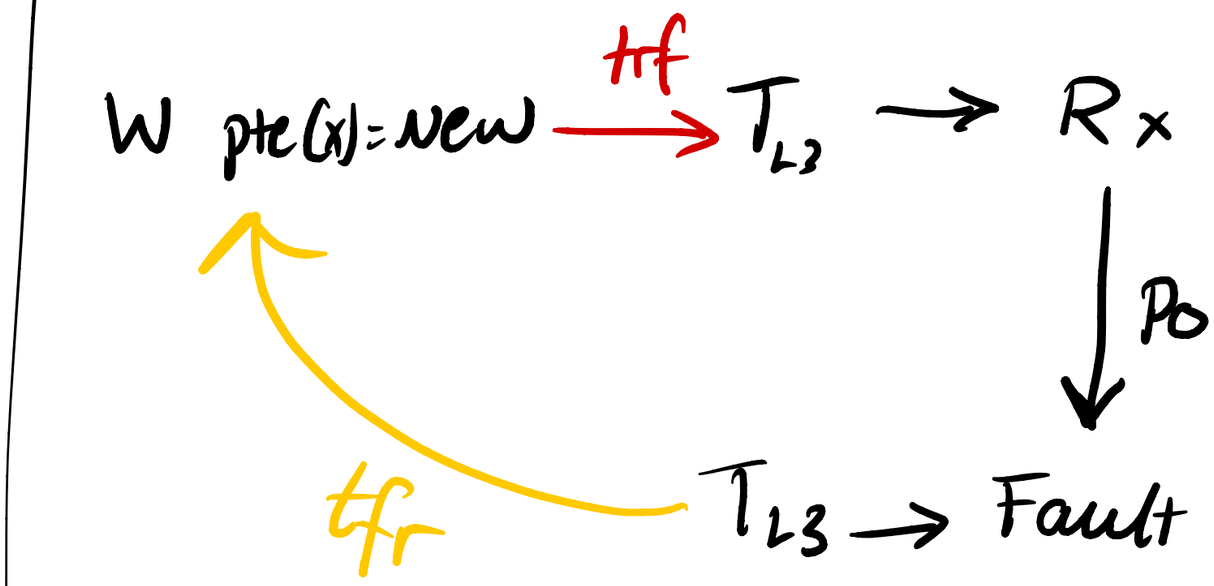
Example: Instruction re-ordering

init: $x \mapsto$ NULL

Thread 0	Thread 1
Store pte(x)=new	$r_0 \leftarrow$ load x
	$r_1 \leftarrow$ load x

Outcome: r_0 sees new
r_1 sees old

allowed



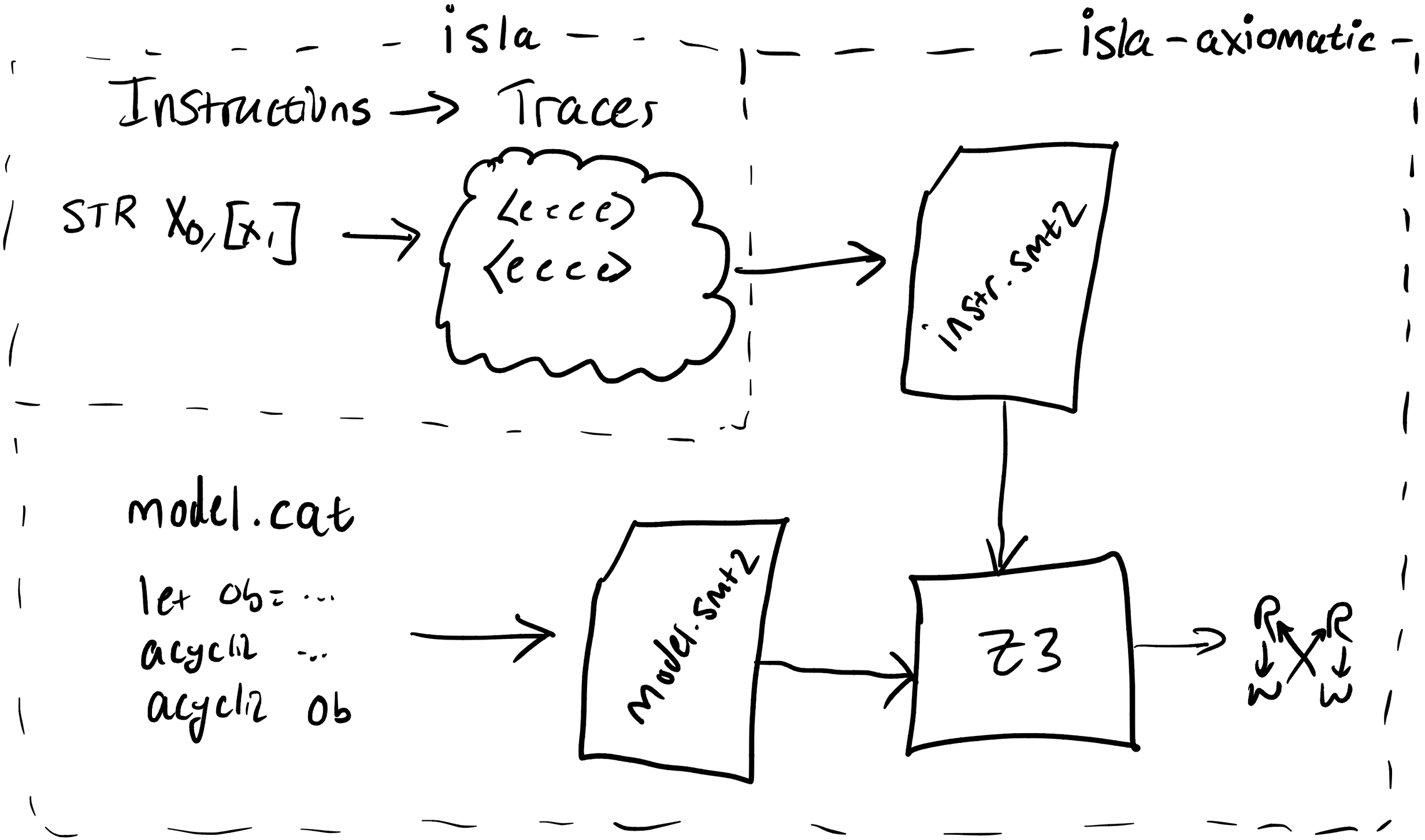
Axiomatic Model

```
let tlb-affects =
  ...

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
```

Relations

Axioms

Executable model



isla – axiomatic

isla-axiomatic.cl.cam.ac.uk/#

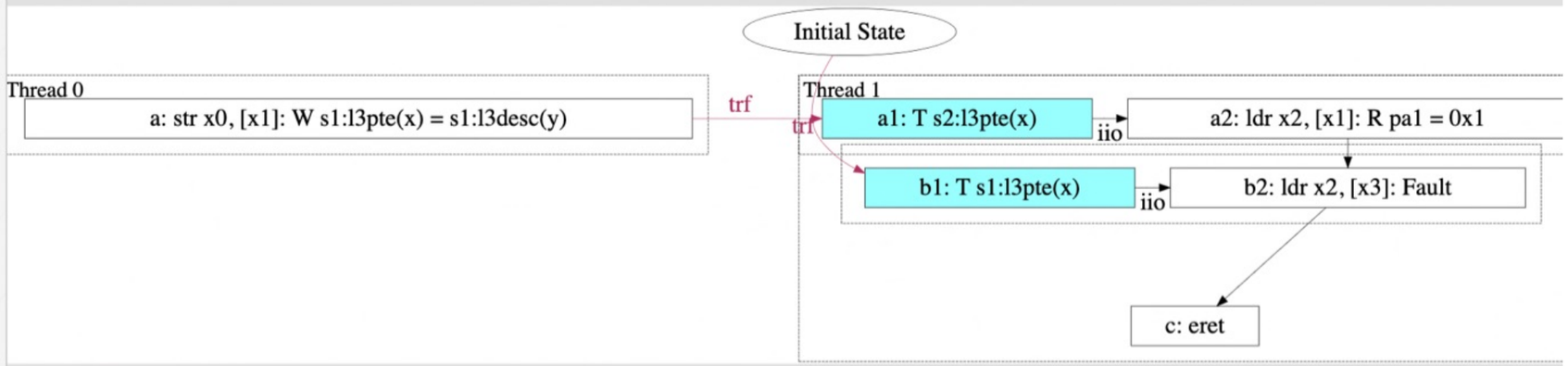
Isla | W / aarch64-vmsa-strong | Litmus file | Memory model | Sail architecture (*aarch64-vmsa*) | **Run test** | Options | Share

W.toml | Memory model

```
    identity 0x1000 with code;
"""

[thread.0]
init = {}
code = """
    STR X0,[X1]
"""

[thread.0.reset]
R0 = "desc3(y, page_table_base)"
R1 = "pte3(x, page_table_base)"

[thread.1]
init = {}
code = """
    LDR X2,[X1]
    MOV X0,X2
    LDR X2,[X3]
"""

[thread.1.reset]
R1 = "x"
R3 = "x"
VBAR_EL1 = "extz(0x1000, 64)"

"PSTATE.SP" = "0b0"
"PSTATE.EL" = "0b00"

[section.thread1_el1_handler]
address = "0x1400"
code = """
    MOV X2,#0

    MRS X13,ELR_EL1
    ADD X13,X13,#4
    MSR ELR_EL1,X13
    ERET
"""

[final]
assertion = "1:X0=1 & 1:X2=0"
```

EventGraph

Relations ← 1 of 1 →



More examples: TLBs

Init: $x \mapsto old$

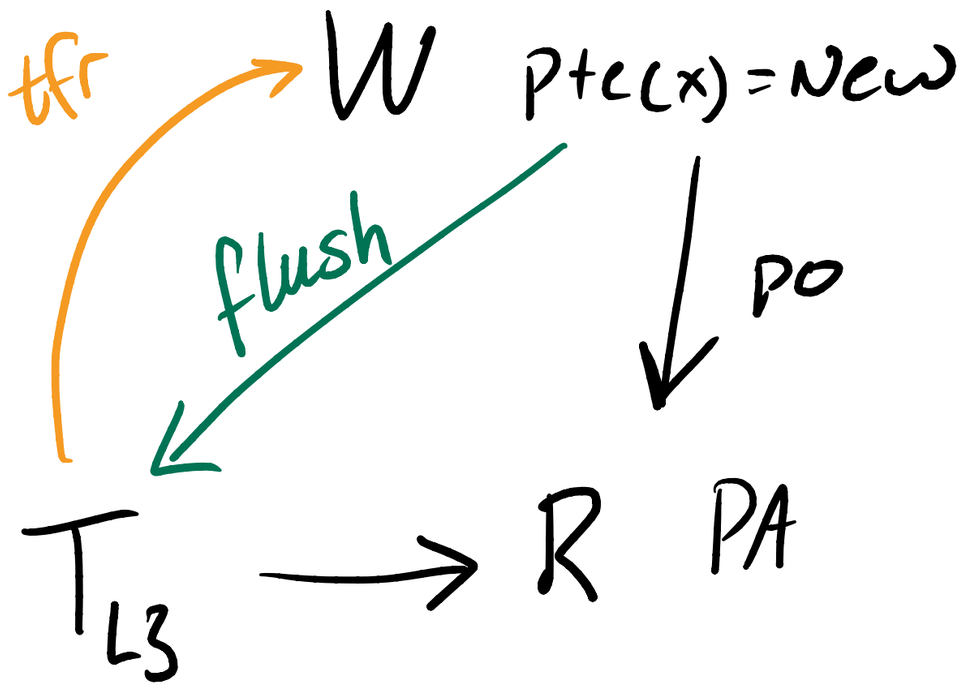
Store pte(x) = New

DSB.SY

ISB

$r_0 \leftarrow$ Load x

Outcome r_0 sees old allowed



More examples: TLB maintenance

Init: $x \mapsto old$

Store pte(x) = New

DSB.SY

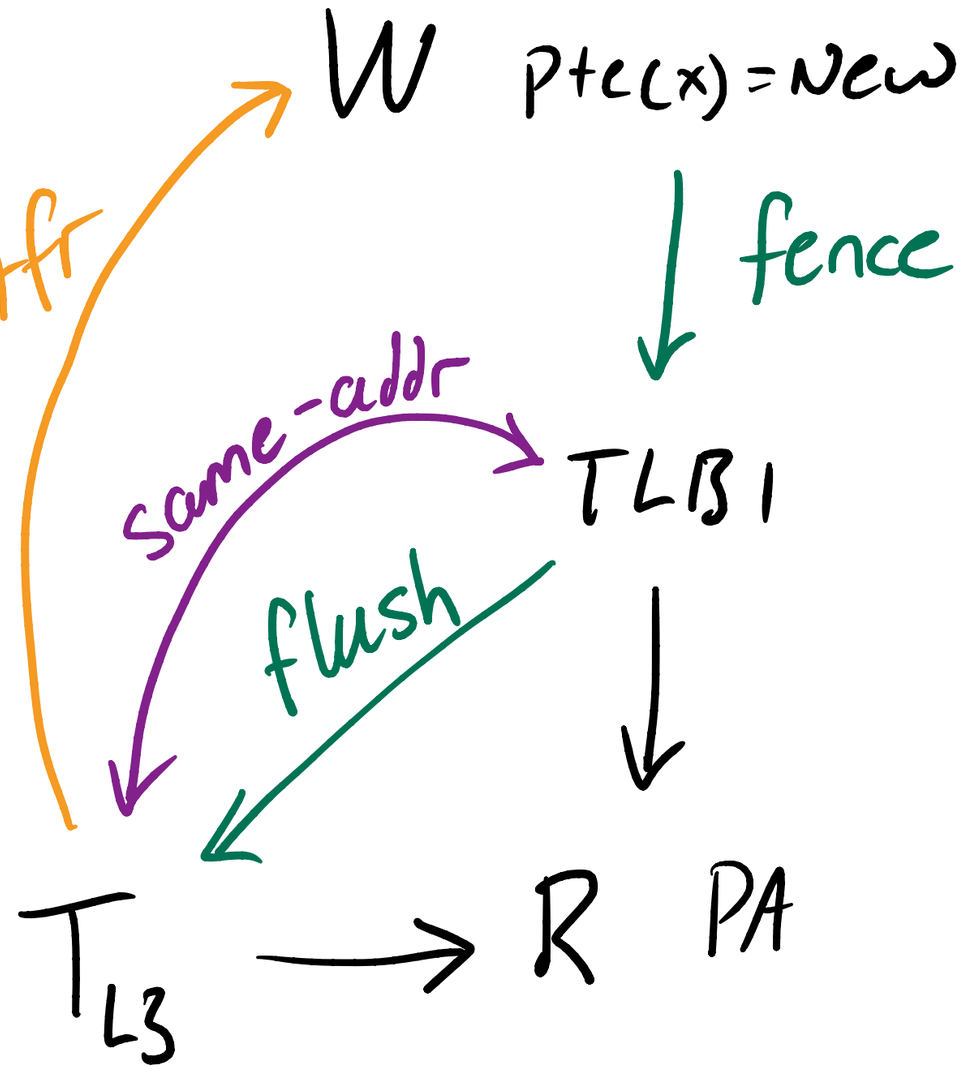
TLBI x

DSB.SY

ISB

$r_0 \leftarrow$ Load x

Outcome: r_0 sees old — Forbidden

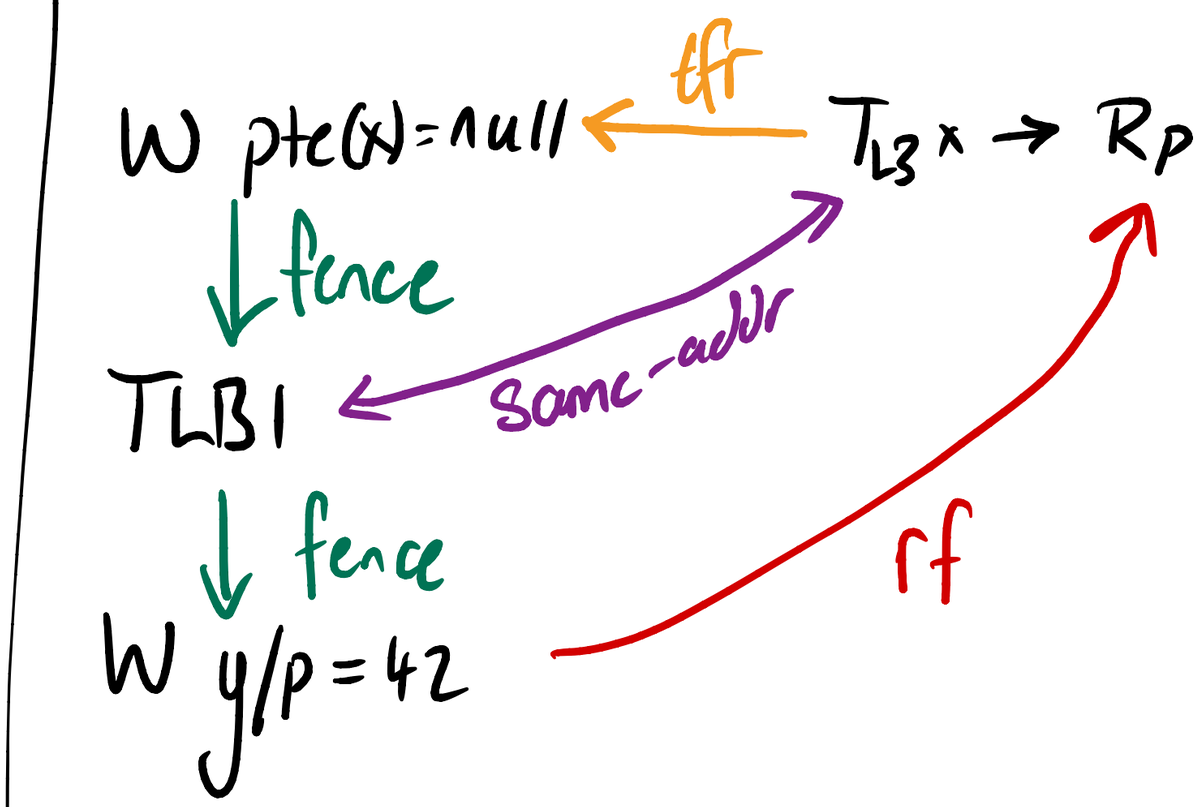


More examples: Breaking

init $x \mapsto p$
$\wedge$ $y \mapsto p$

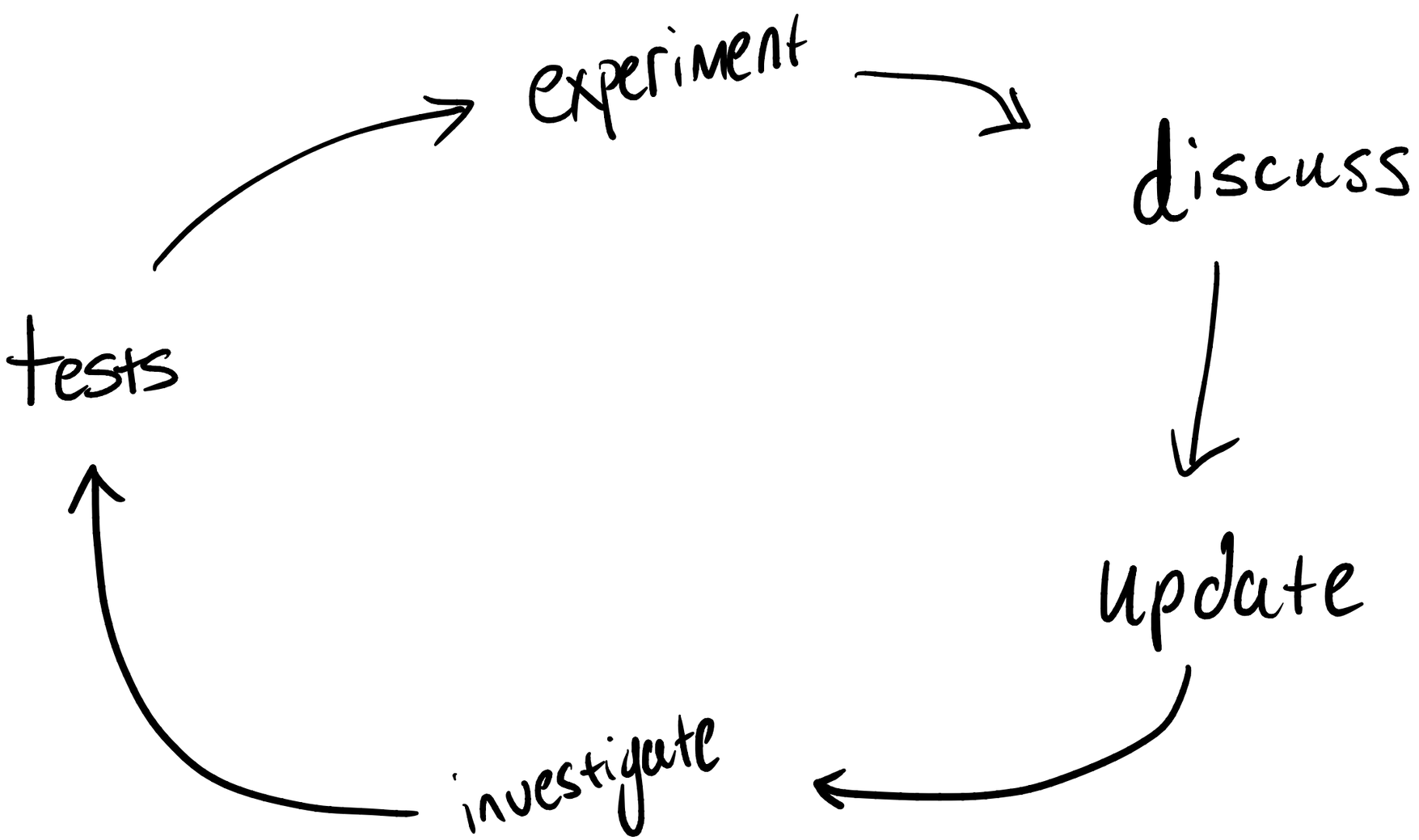
Store pte(x) := NULL	$r_0 \leftarrow$ load x
DSB.SY	
TLBI x	
DSB.SY	
Store y = 42	

Outcome $r_0 = 42$ forbidden

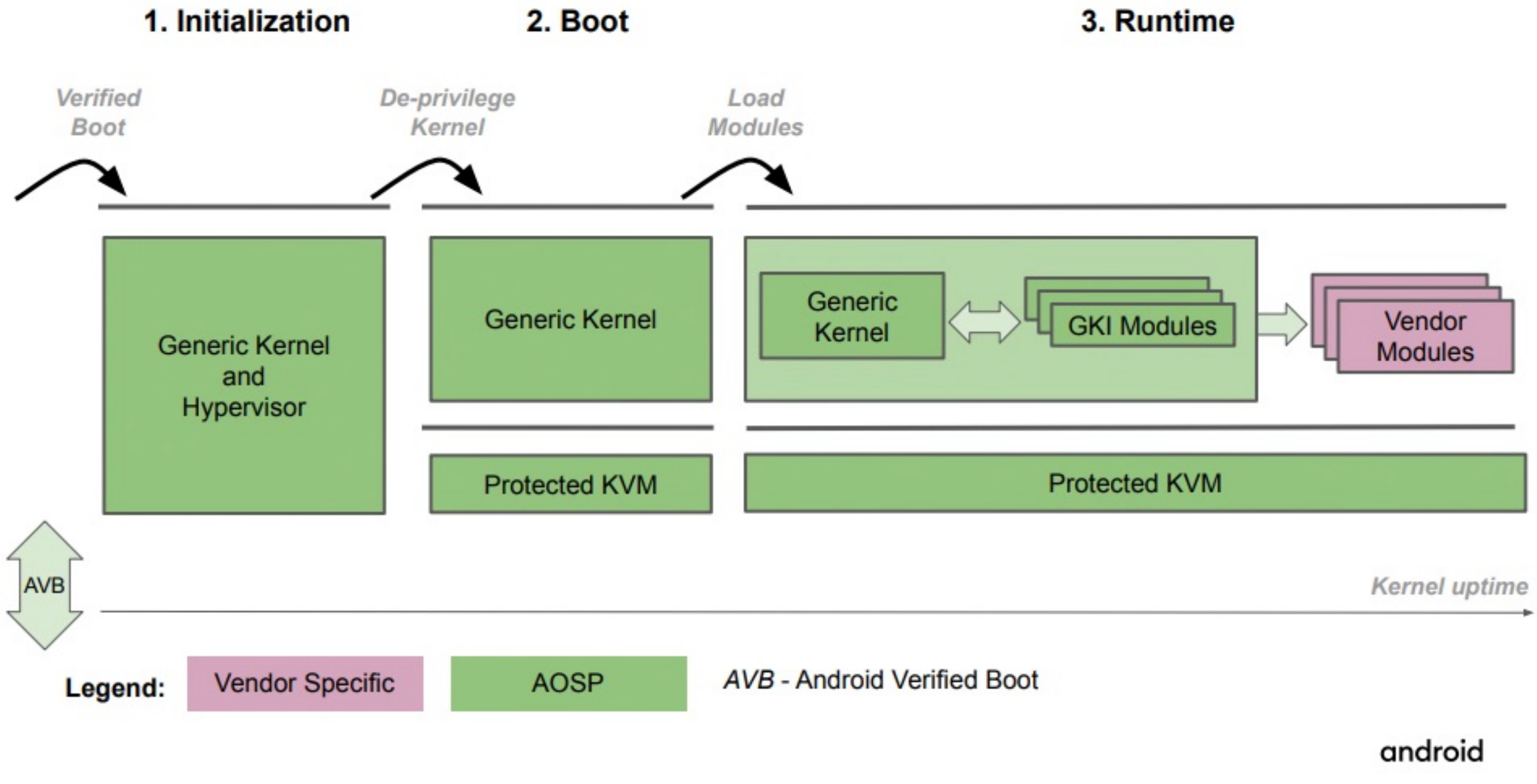


. . . and much more . . .

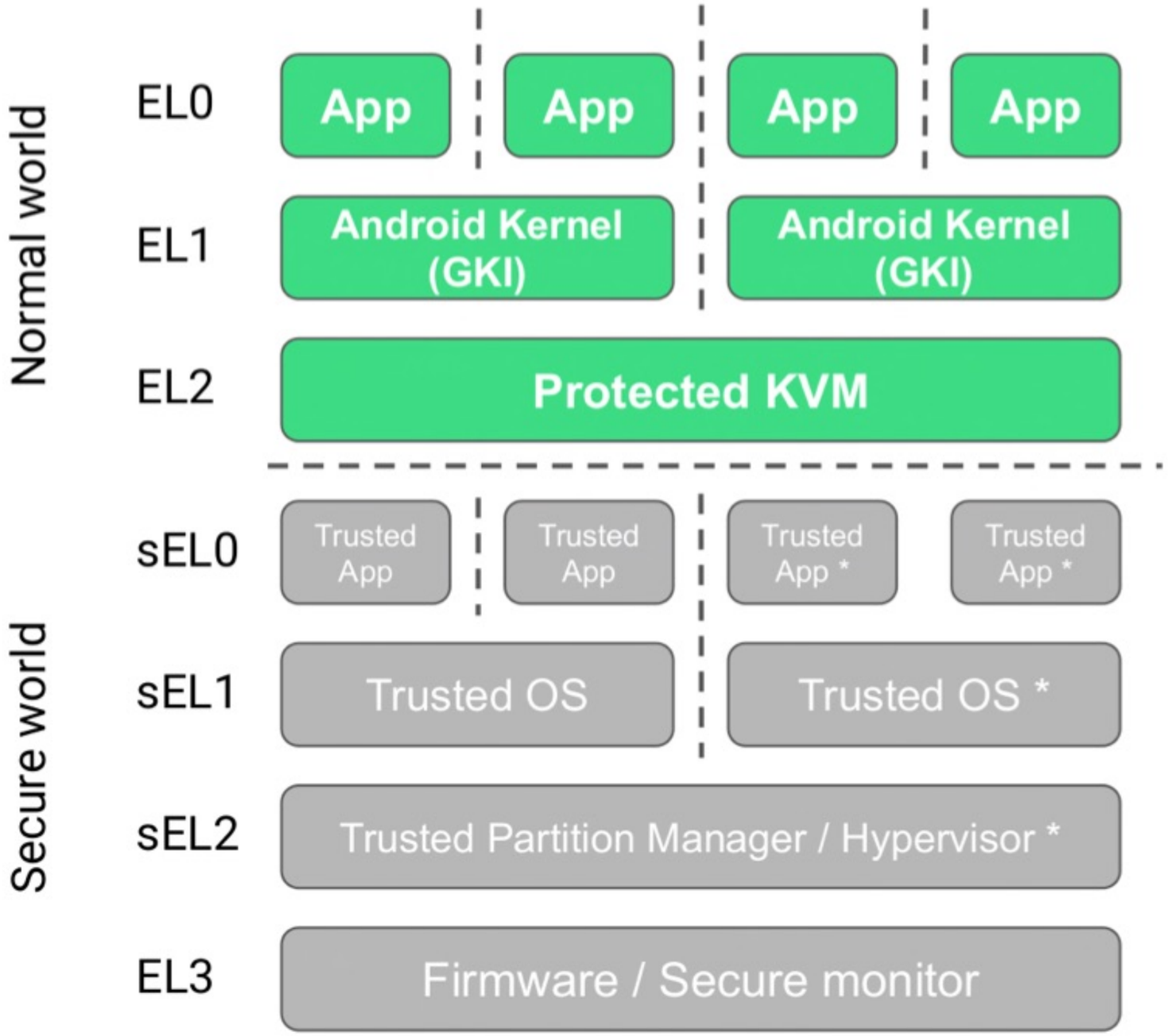
An aside: Clarifying the architecture



pKVM



Credit: Will Deacon, Google 2020 @ KVM Forum



Credit: Will Deacon KVM Forum 2020

Recap

clarify architecture

Formal semantics

Tooling