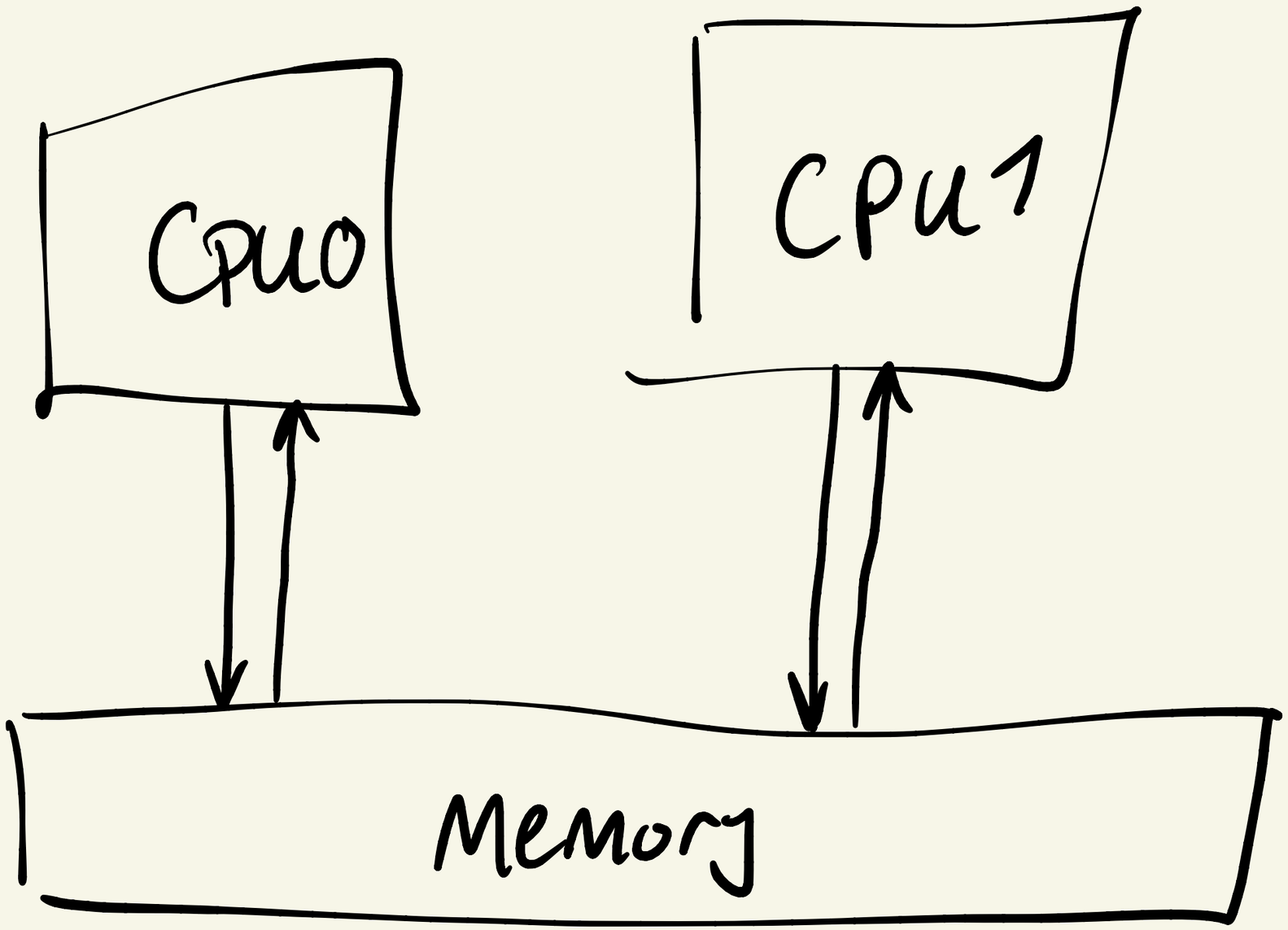
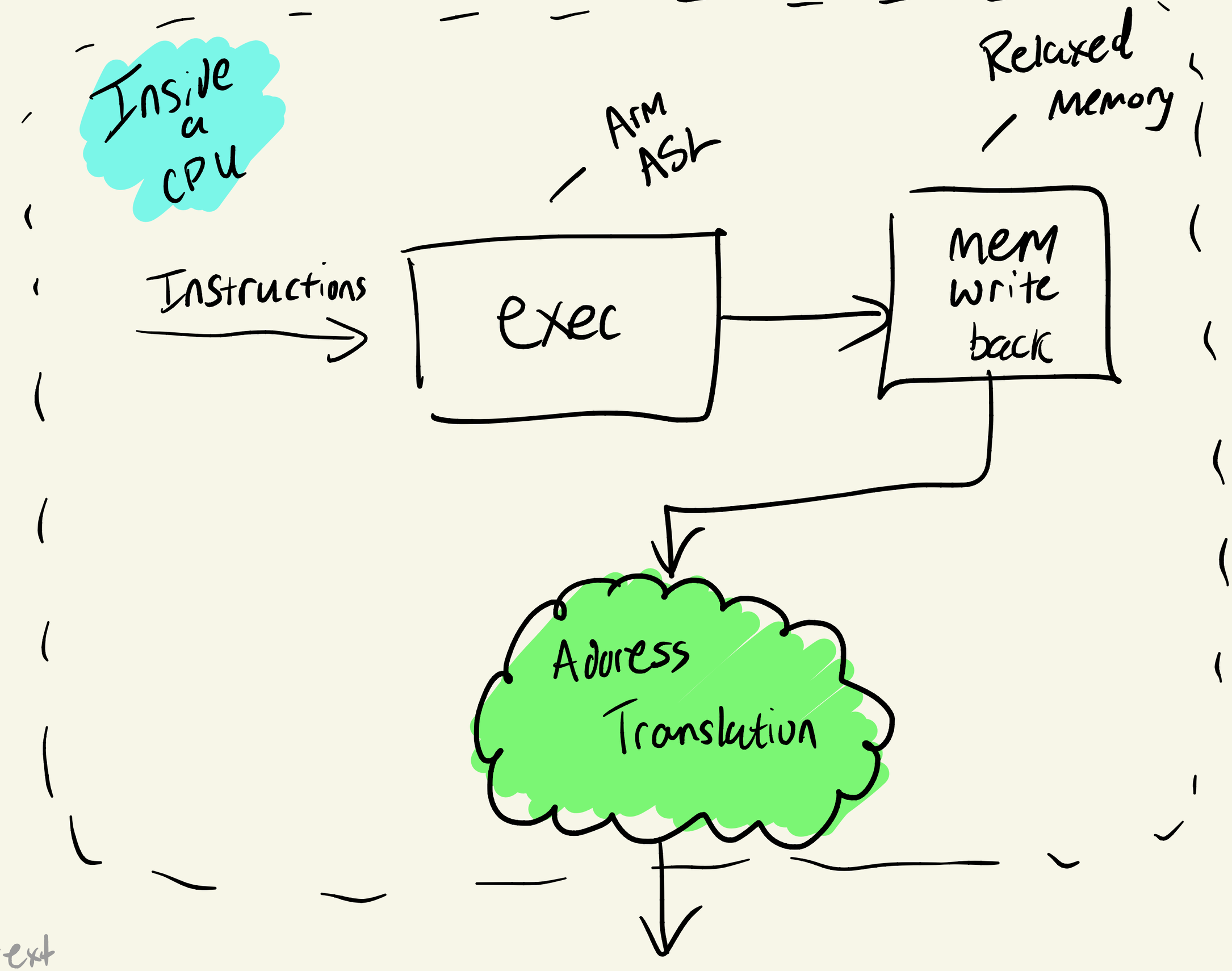
Relaxed Virtual Memory — Armv8 edition —

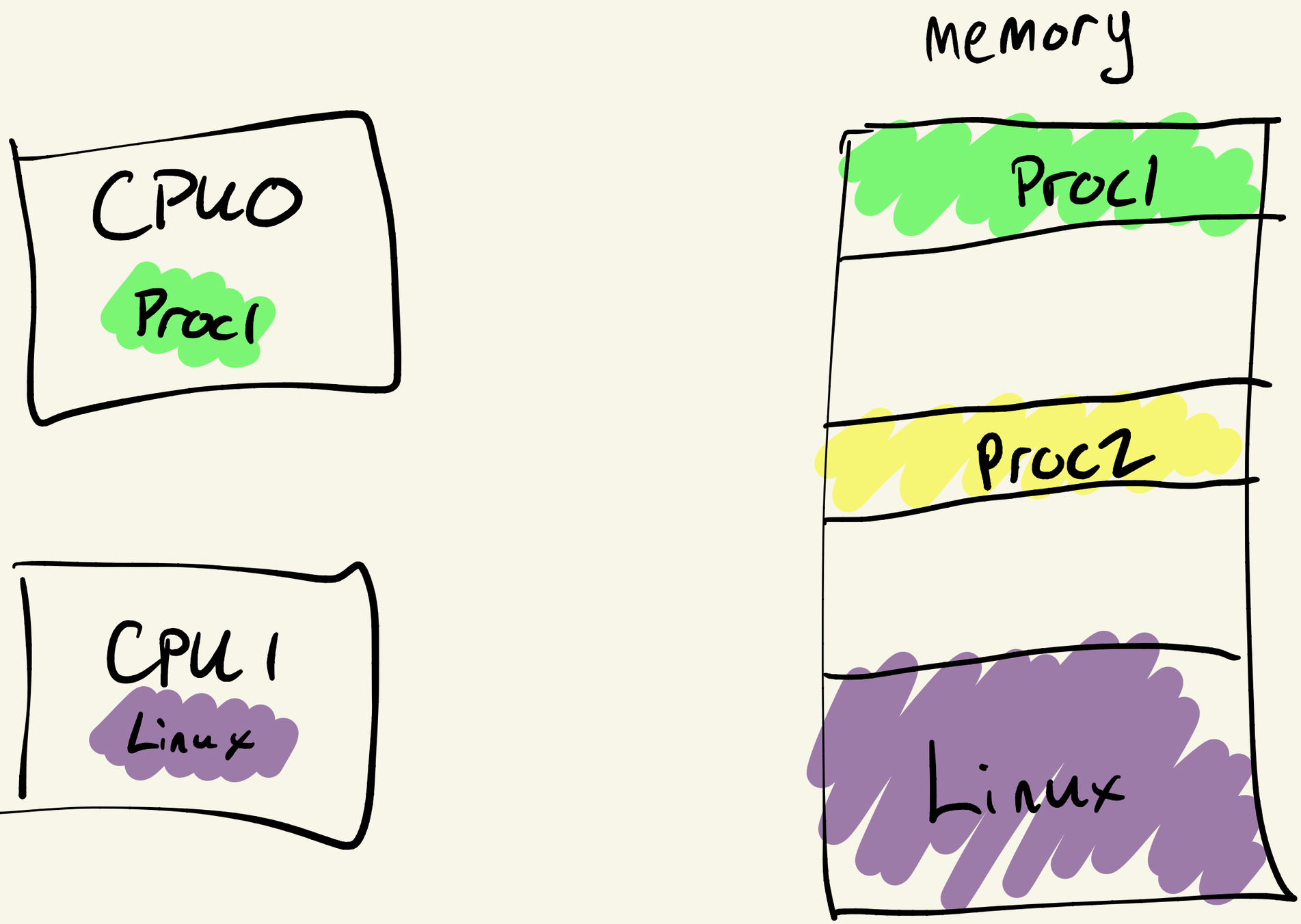
Ben Simner[1], Alasdair Armstrong[1], Jean Pichon-Pharabod[2],
Christopher Pulte[1], Richard Grisenthwaite[3], Peter Sewell[1]

1. Cambridge, UK 2. Aarhus, DK 3. Arm Ltd., UK

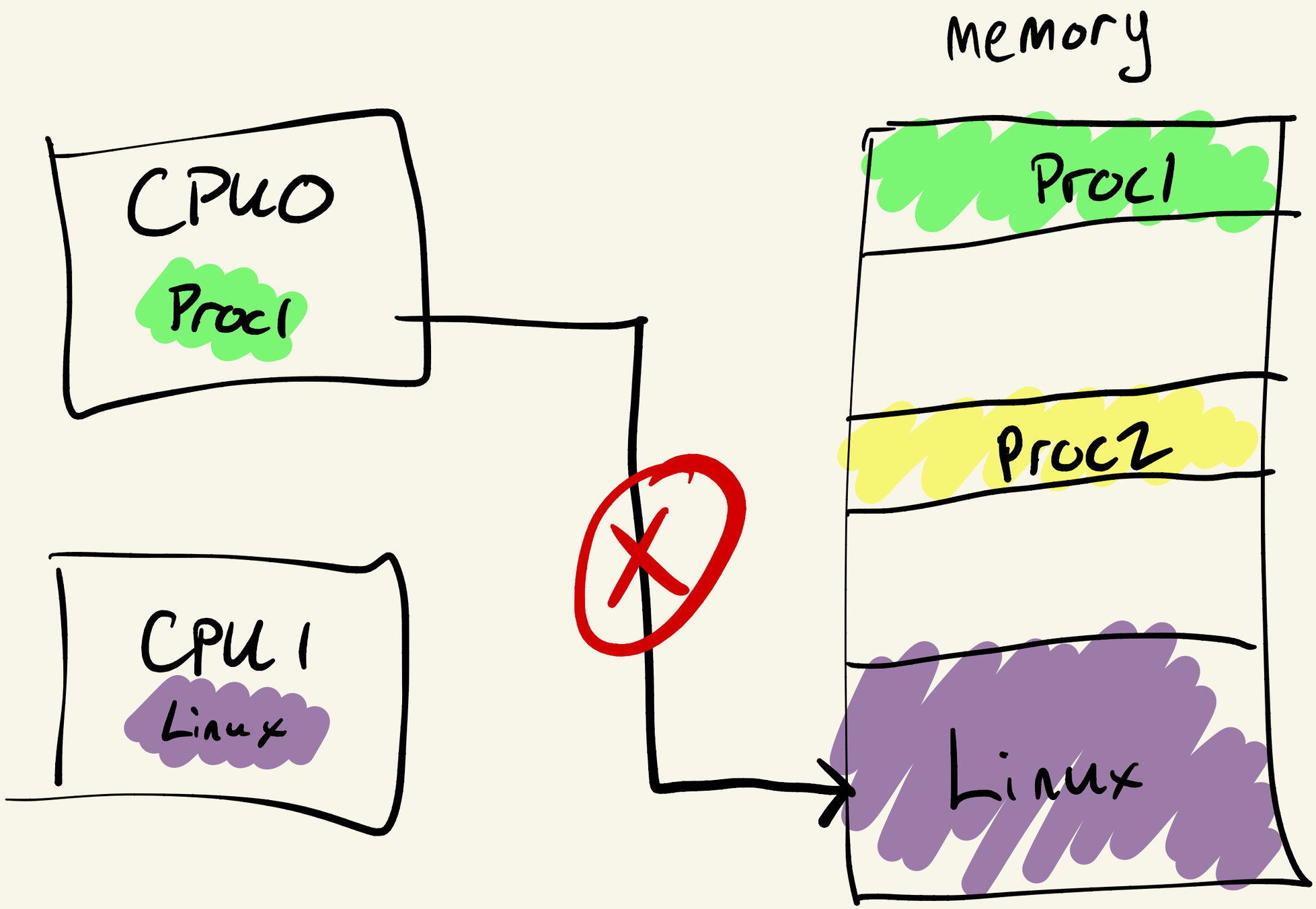

CPU0
CPU1
Memory


Inside a CPU
Arm ASL
Relaxed Memory
Instructions
exec
mem write back
Address Translation

Why Virtual Memory

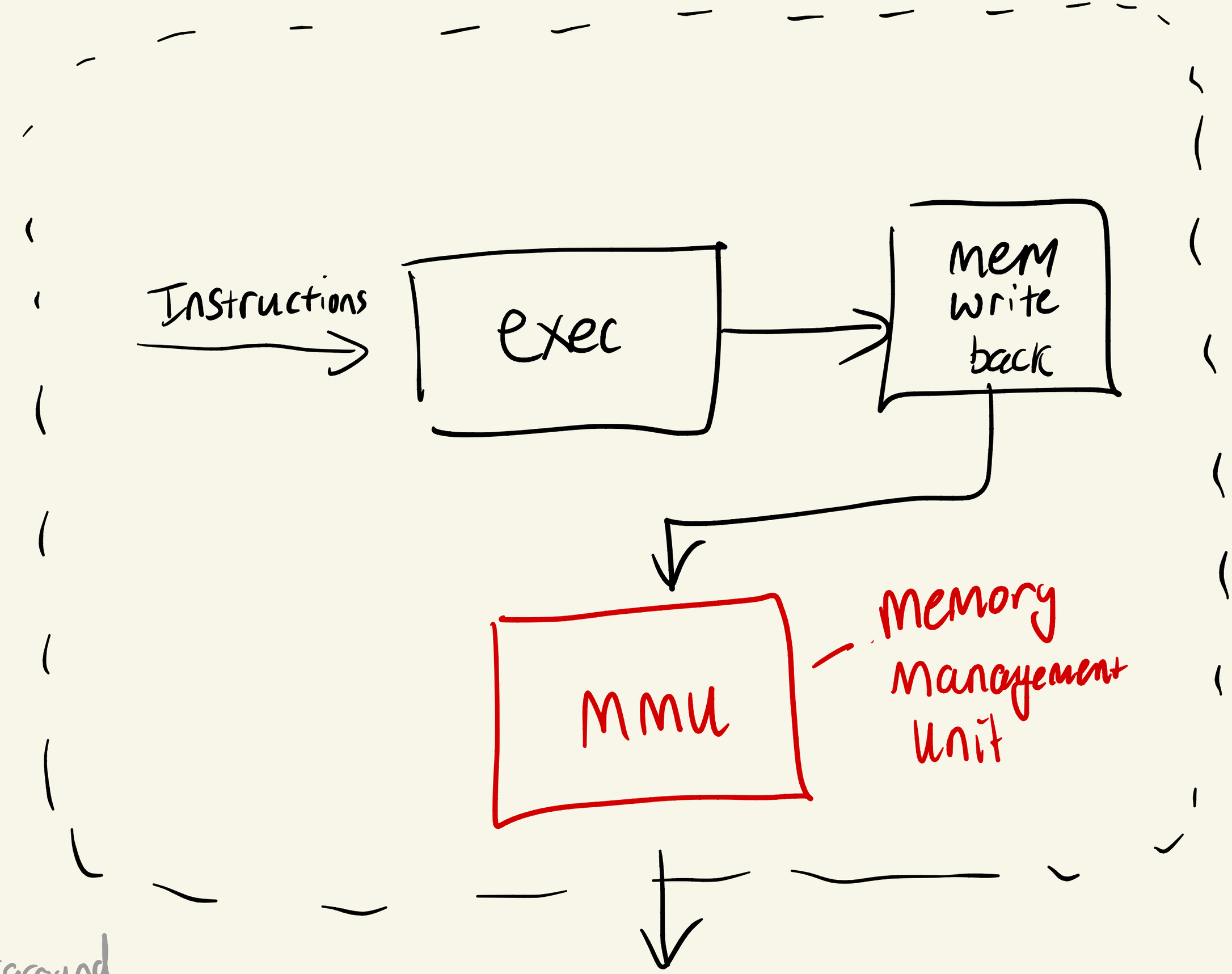


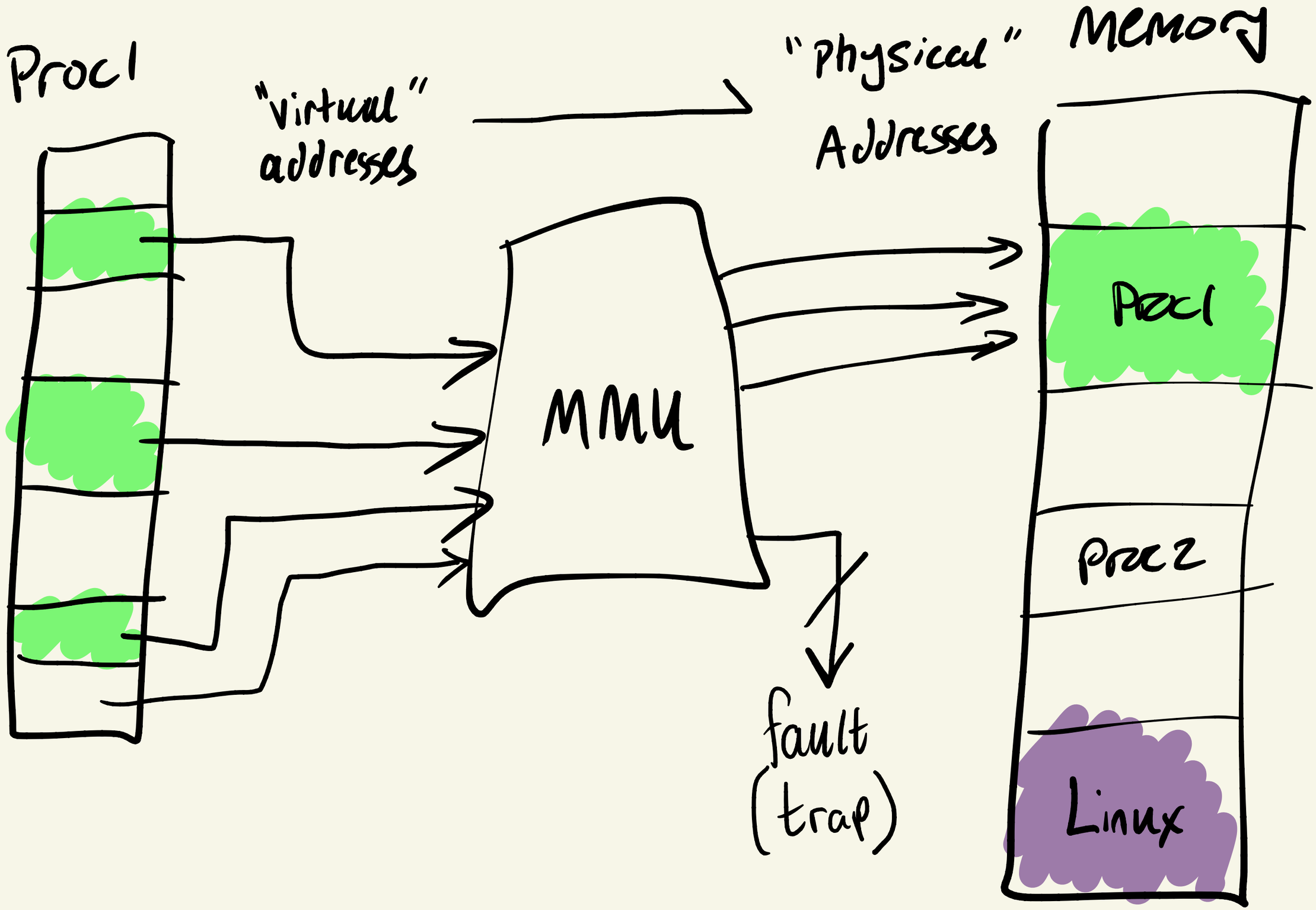
Why Virtual Memory

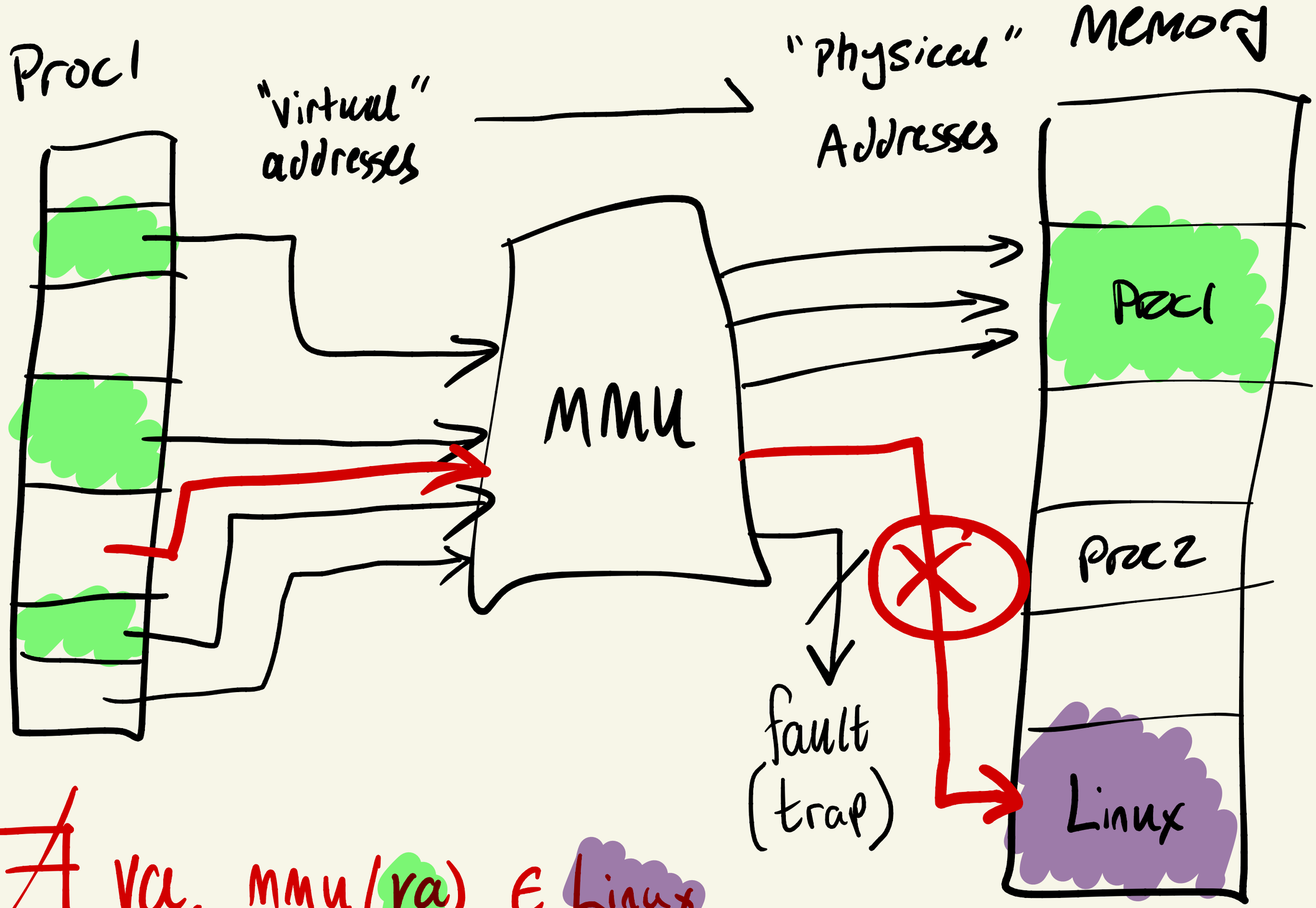


Aims

- Formal Semantics
- Build Support tooling
- reason about real Systems code implementing isolation.

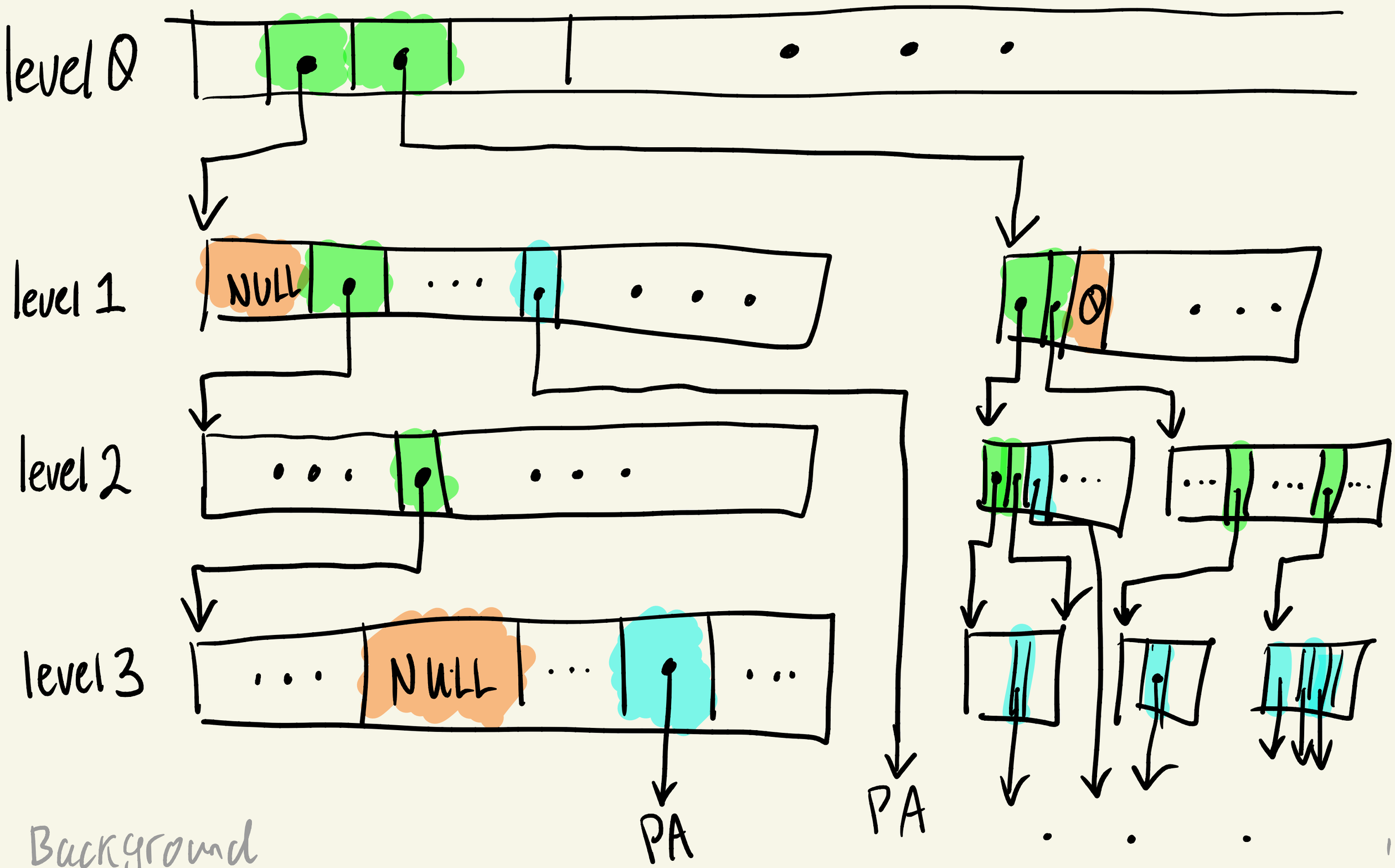

Instructions
exec
mem
write
back
MMU
memory
Management
Unit


Proc1
"virtual" addresses
"physical" Addresses
Memory
MMU
fault (trap)
Proc1
Proc2
Linux

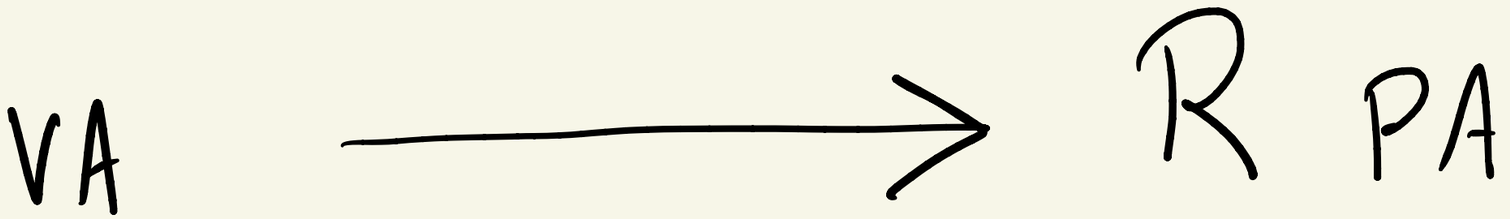


$\nexists$ va. mmu(va) $\in$ Linux

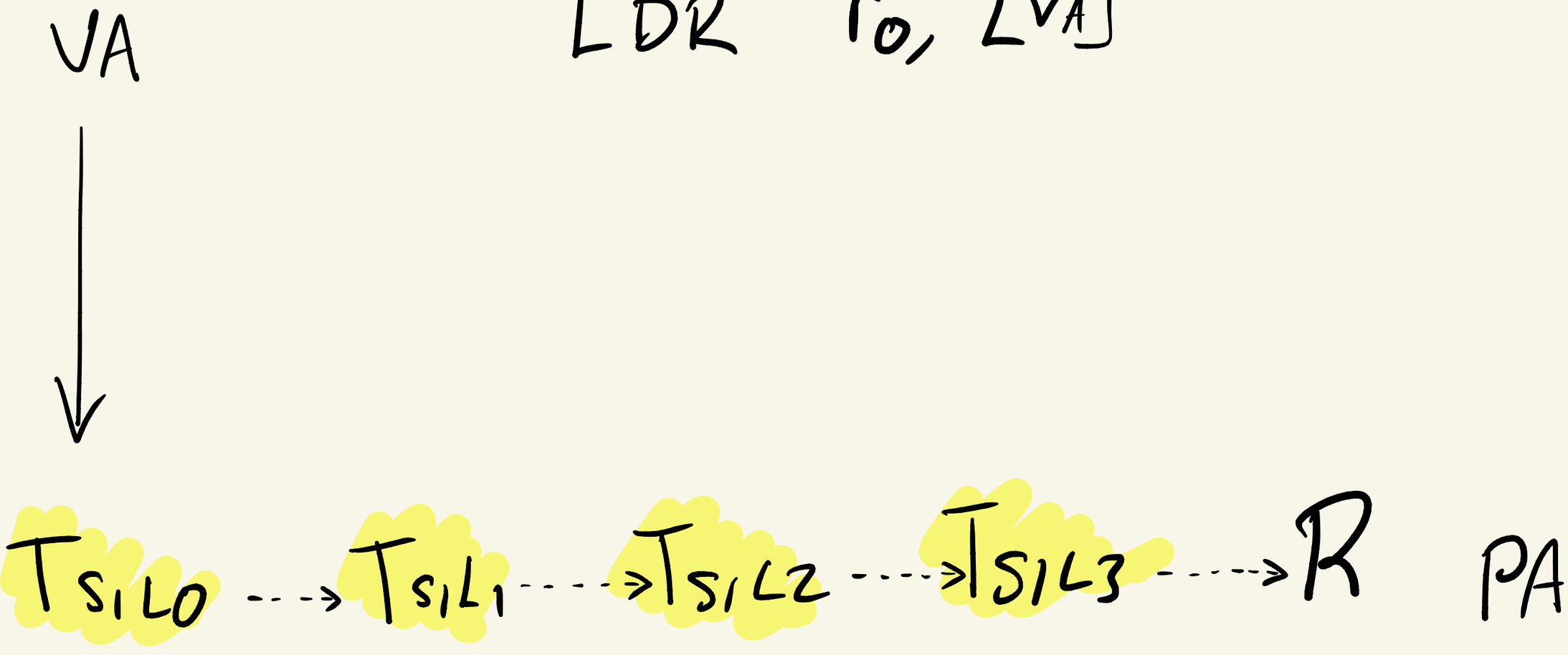
In-memory Data Structure

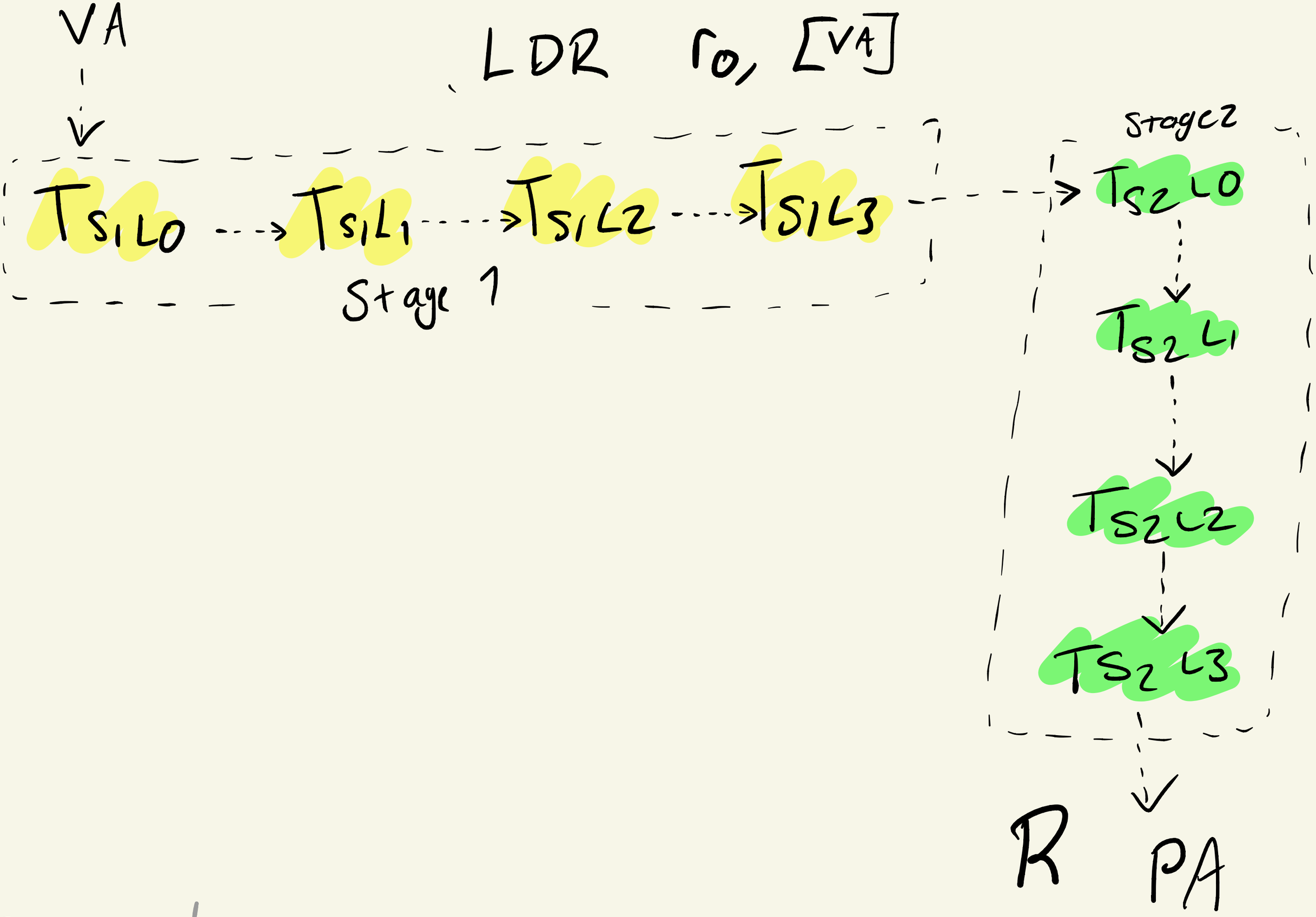


LDR r0, [VA]

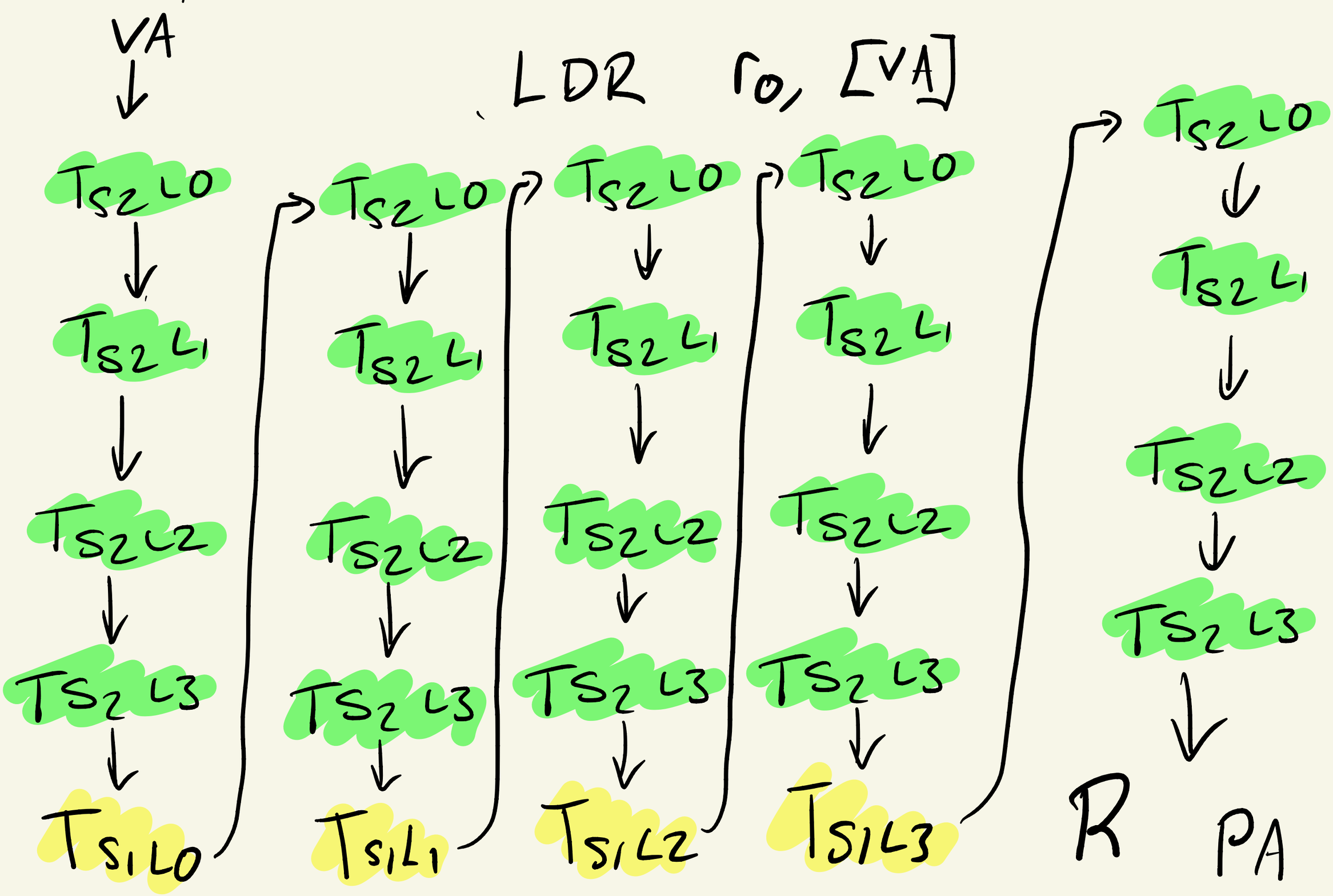


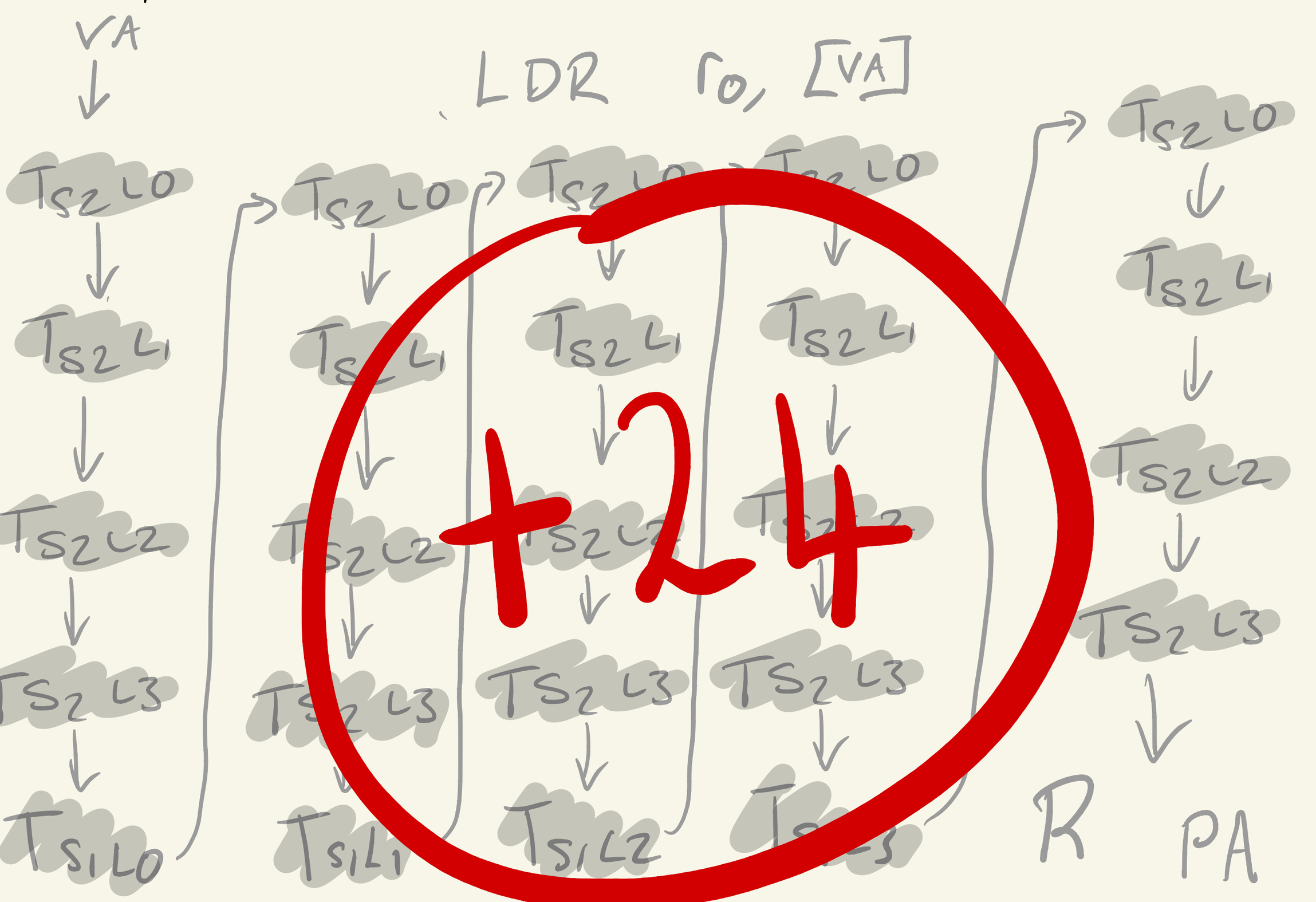
LDR r0, [VA]

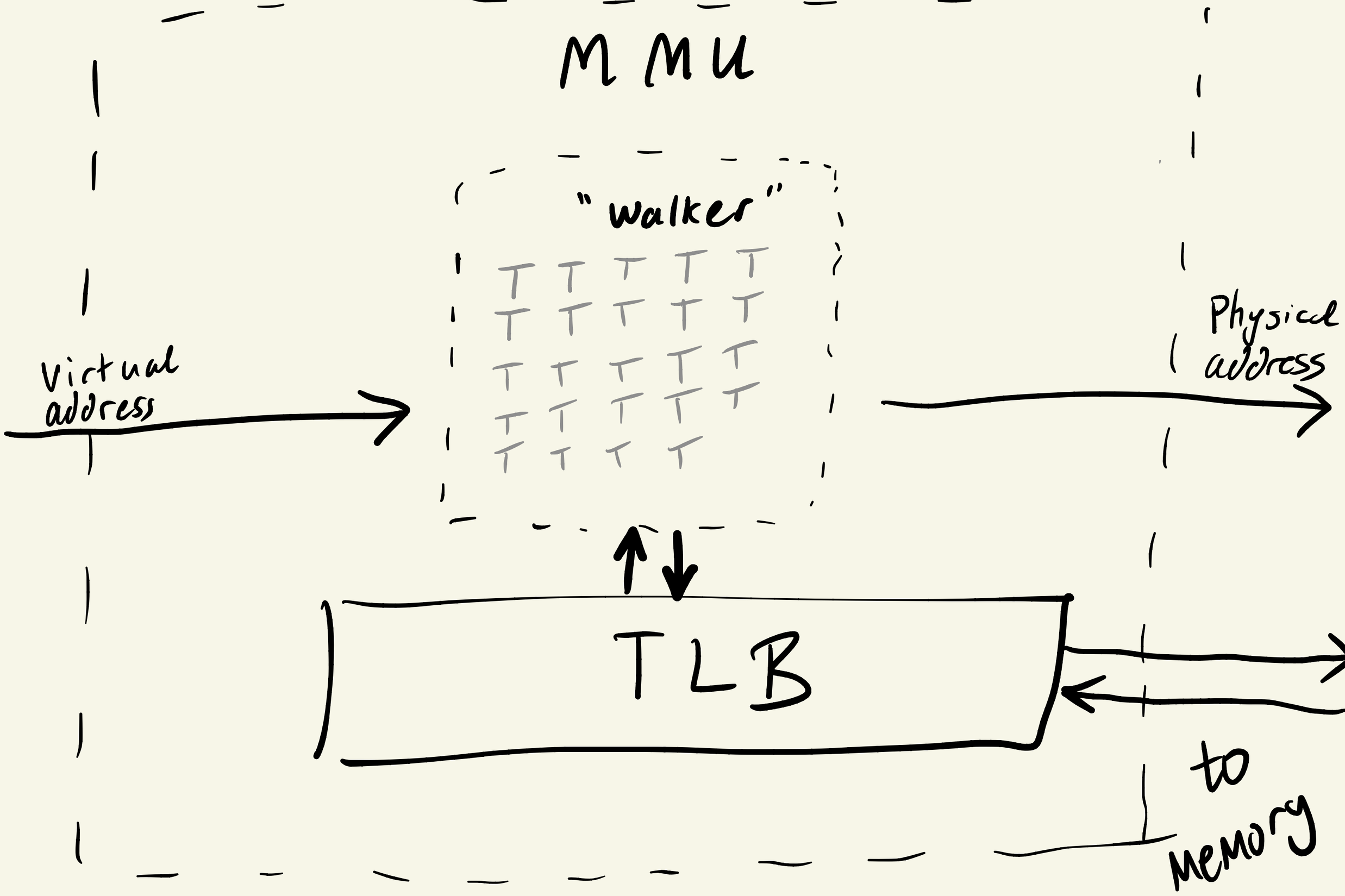



VA
LDR r0, [VA]
Stage 1
Stage 2
R
PA

LDR r0, [VA]


VA
TS2 L0
TS2 L1
TS2 L2
TS2 L3
TS1 L0
TS1 L1
TS1 L2
TS1 L3
TS2 L0
TS2 L1
TS2 L2
TS2 L3
TS1 L0
TS1 L1
TS1 L2
TS1 L3
TS2 L0
TS2 L1
TS2 L2
TS2 L3
R
PA


VA
LDR r0, [VA]
TS2 L0
TS2 L1
TS2 L2
TS2 L3
TS1 L0
TS1 L1
TS1 L2
+24
R
PA


MMU
"walker"
Virtual address
Physical address
TLB
to memory

Issues & Contributions

- TLBs
- Arm manual
  11,530 pages

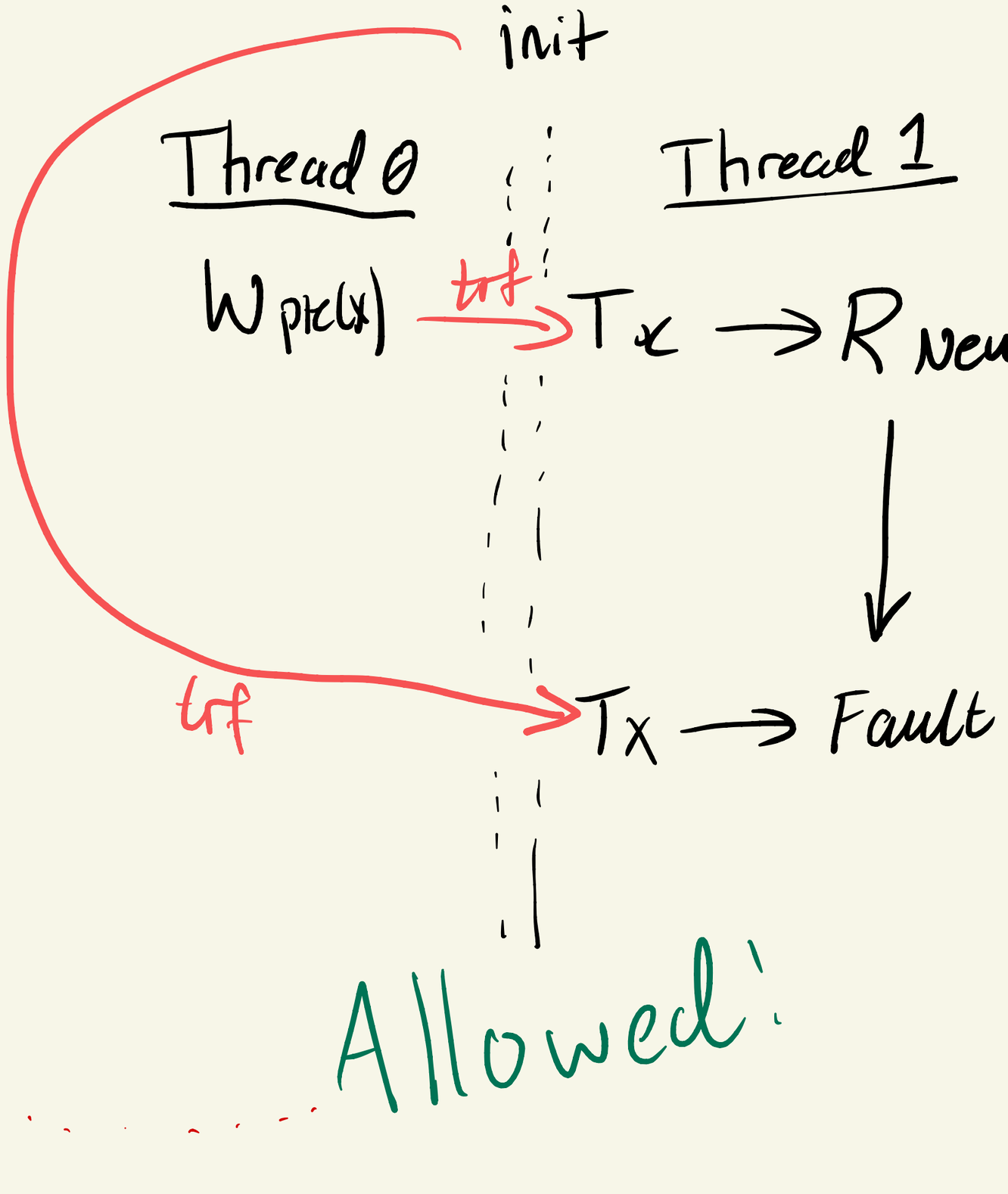
- Clarify Spec
  by discussion with Arm
- Extended Arm model
- Protected KVM
  Litmus tests

Behaviours

Init: $X \mapsto NULL$

Thread 0	Thread 1
$W_{ptr(x)} = New$	$r_1 = Rx$
	$r_2 = Rx$

Can see 'New' then later stop seeing it?



Allowed!

isla – axiomatic

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

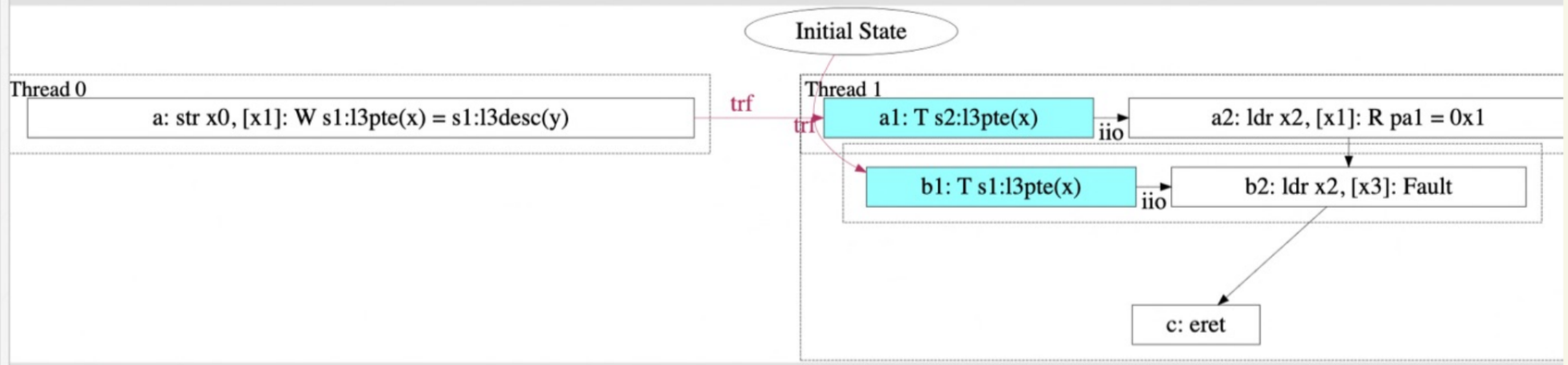
Isla | W / aarch64-vmsa-strong | Litmus file | Memory model | Sail architecture (*aarch64-vmsa*) | **Run test** | Options | Share

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 →



Contributions

Lots of Behaviours
(Not mentioned here)

- 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

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

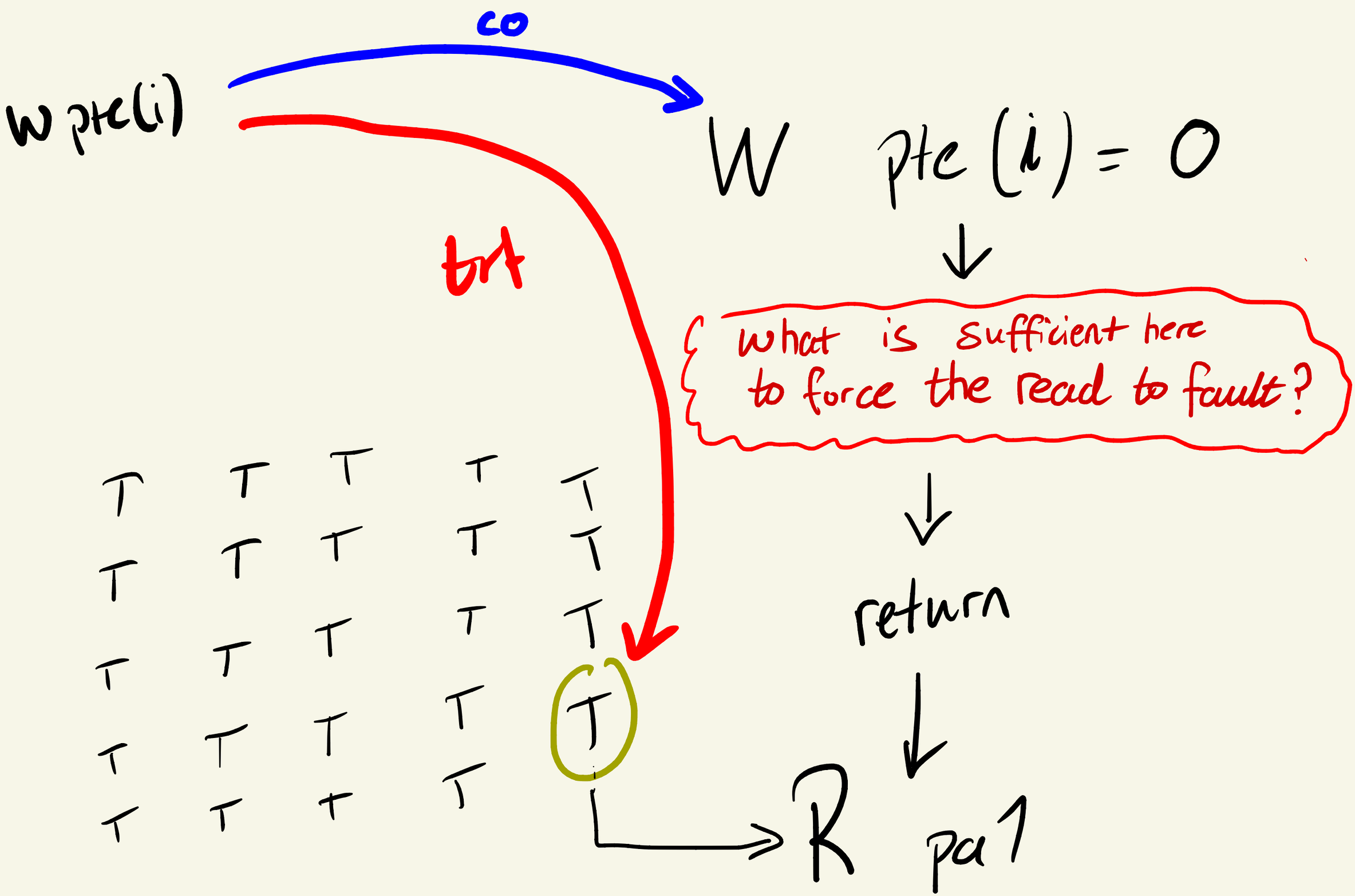
Axioms

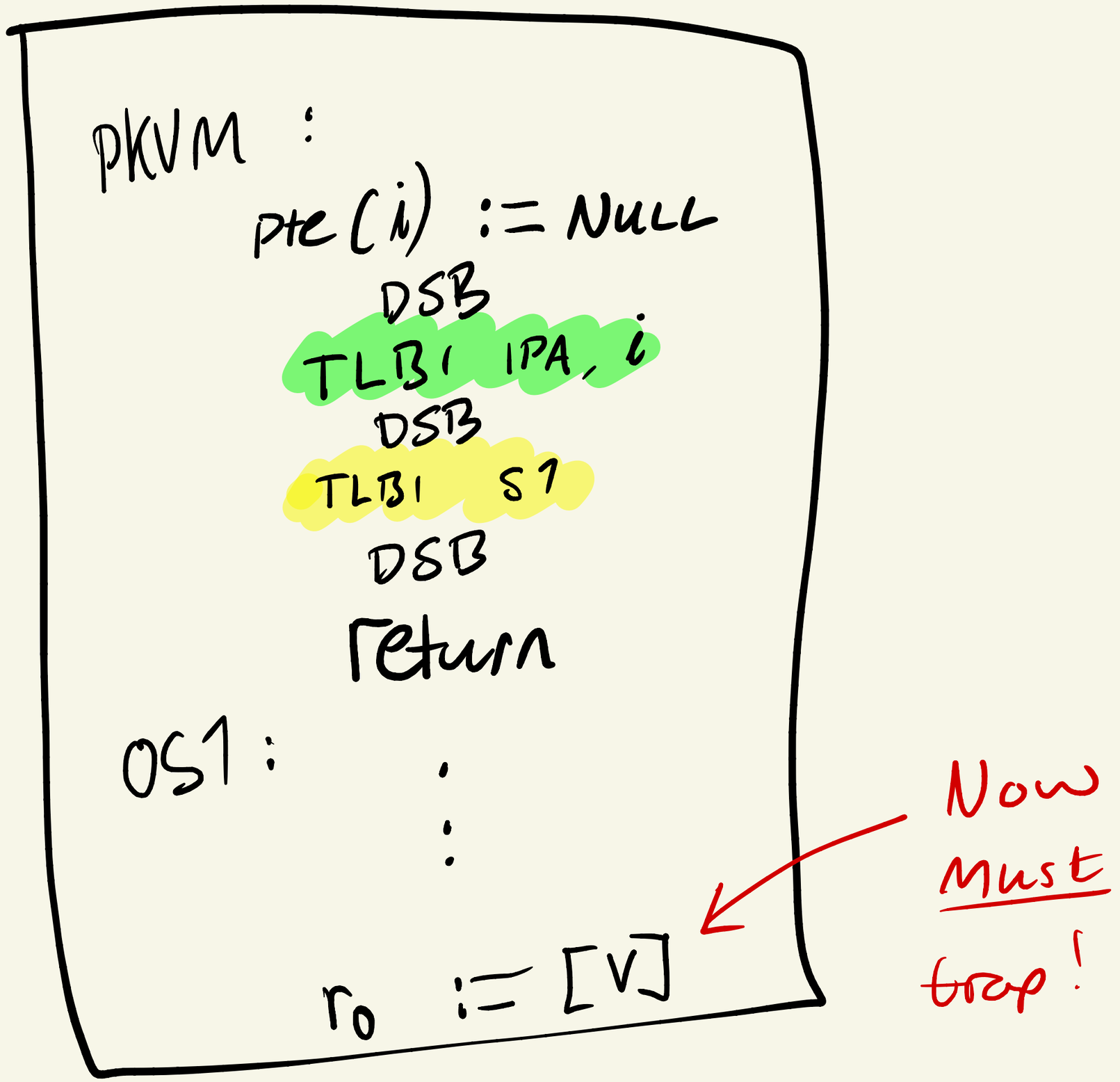
Contributions

pKVM Examples

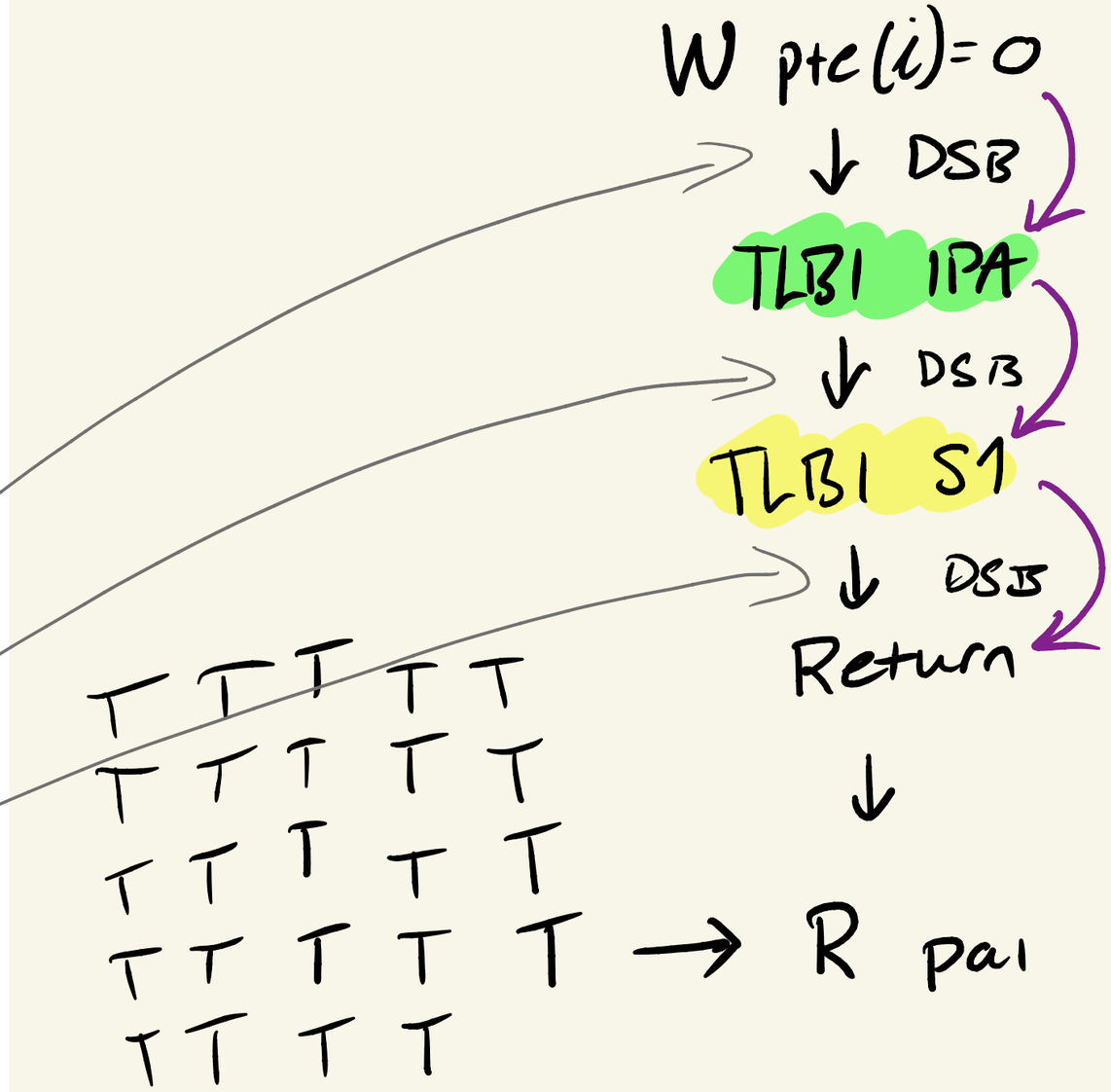
```
pKVM.unshare_page (broken):
    // unmap memory
    pte(i) := NULL
    // continue guest OS
    return

OS:
    ⋮
    r0 := [V]
```

Must trap?


co
W pte(i)
trt
W pte(i) = 0
what is sufficient here
to force the read to fault?
return
R pa1


pKVM :
ptc(i) := NULL
DSB
TLBI IPA, i
DSB
TLBI S1
DSB
return
OS1 :
r0 := [v]
Now Must trap!

```
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

(* ordered-before TLBI and translate *)
let obtlbi_translate =
      ...
    | (tcache2 ; wco? ; [TLBI-S1])
        & (iio^-1 ; [T & Stage1] ; maybe_TLB_cached)

(* ordered-before TLBI *)
let obtlbi =
      obtlbi_translate
    | ...

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

(* barrier-ordered-before *)
let bob =
      ...
    | [F | C]; po; [dsbsy]
    | [dsb] ; po

(* Ordered-before *)
let ob = (... | bob | iio | obtlbi | ctxob)^+

irreflexive ob as external
```



Contributions

```
let tlb_might_affect =
    [ TLBI-S1 & ~TLBI-S2 &  TLBI-VA  &  TLBI-ASID & TLBI-VMID] ; (same-va-page & same-asid & same-vmid) ; [T & Stage1]
  | [ TLBI-S1 & ~TLBI-S2 & ~TLBI-VA  &  TLBI-ASID & TLBI-VMID] ; (same-asid & same-vmid) ; [T & Stage1]
  | [ TLBI-S1 & ~TLBI-S2 & ~TLBI-VA  & ~TLBI-ASID & TLBI-VMID] ; same-vmid ; [T & Stage1]
  | [~TLBI-S1 &  TLBI-S2 &  TLBI-IPA & ~TLBI-ASID & TLBI-VMID] ; (same-ipa-page & same-vmid) ; [T & Stage2]
  | [~TLBI-S1 &  TLBI-S2 & ~TLBI-IPA & ~TLBI-ASID & TLBI-VMID] ; same-vmid ; [T & Stage2]
  | [ TLBI-S1 &  TLBI-S2 & ~TLBI-IPA & ~TLBI-ASID & TLBI-VMID] ; same-vmid ; [T]
  | ( TLBI-S1 &           ~TLBI-IPA & ~TLBI-ASID & ~TLBI-VMID) * (T & Stage1)
  | (           TLBI-S2 & ~TLBI-IPA & ~TLBI-ASID & ~TLBI-VMID) * (T & Stage2)

let tlb-affects =
    [TLBI-IS] ; tlb_might_affect
  | ([~TLBI-IS] ; tlb_might_affect) & int
```

W ptc(i) = 0

↓ DSB

TLBI IPA

↓ DSB

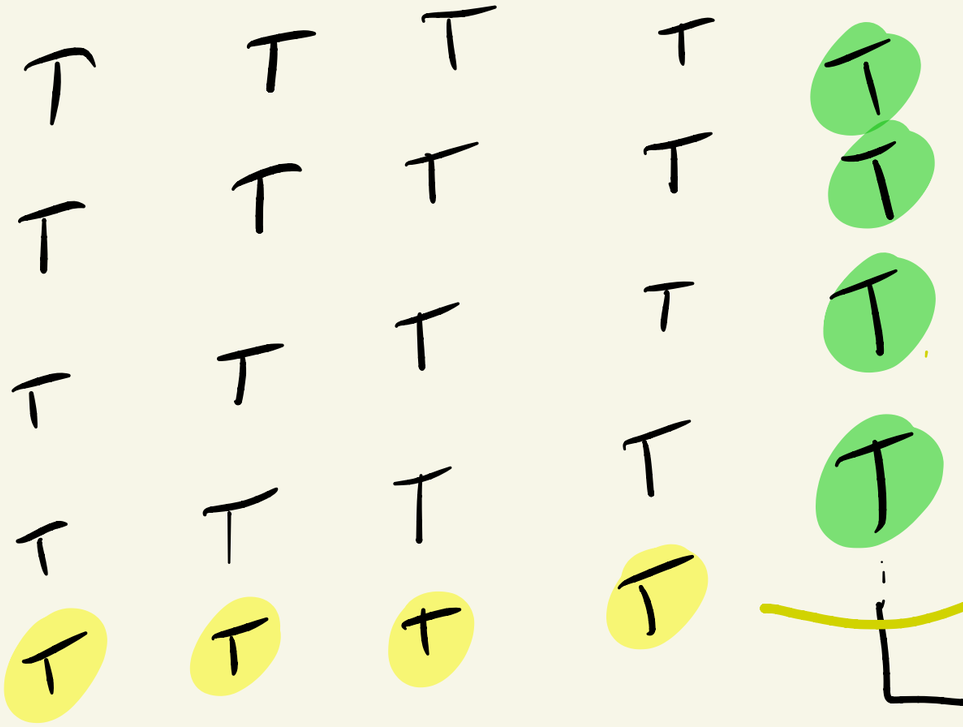
TLBI S1

↓ DSB

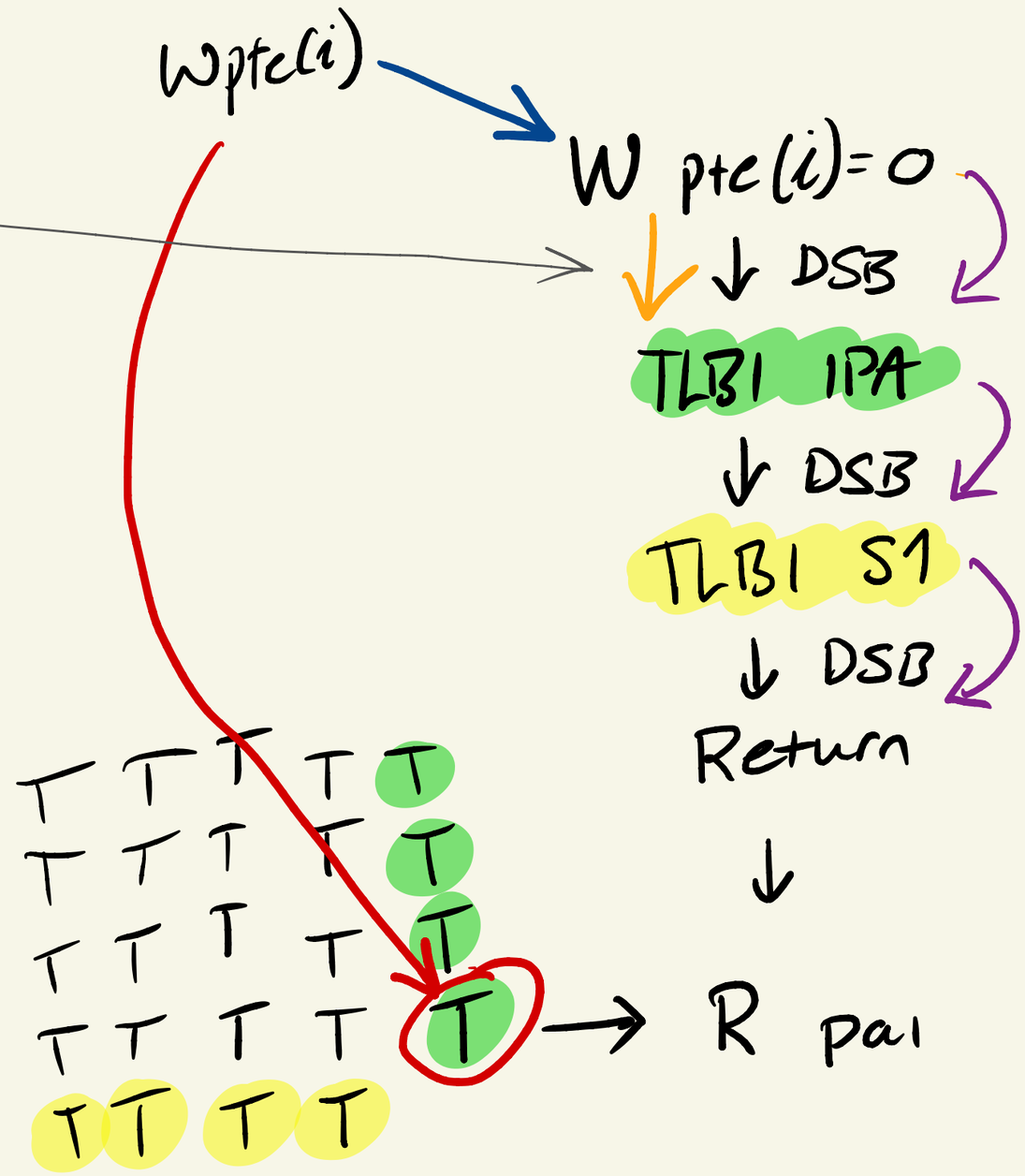
return

↓

R pa1

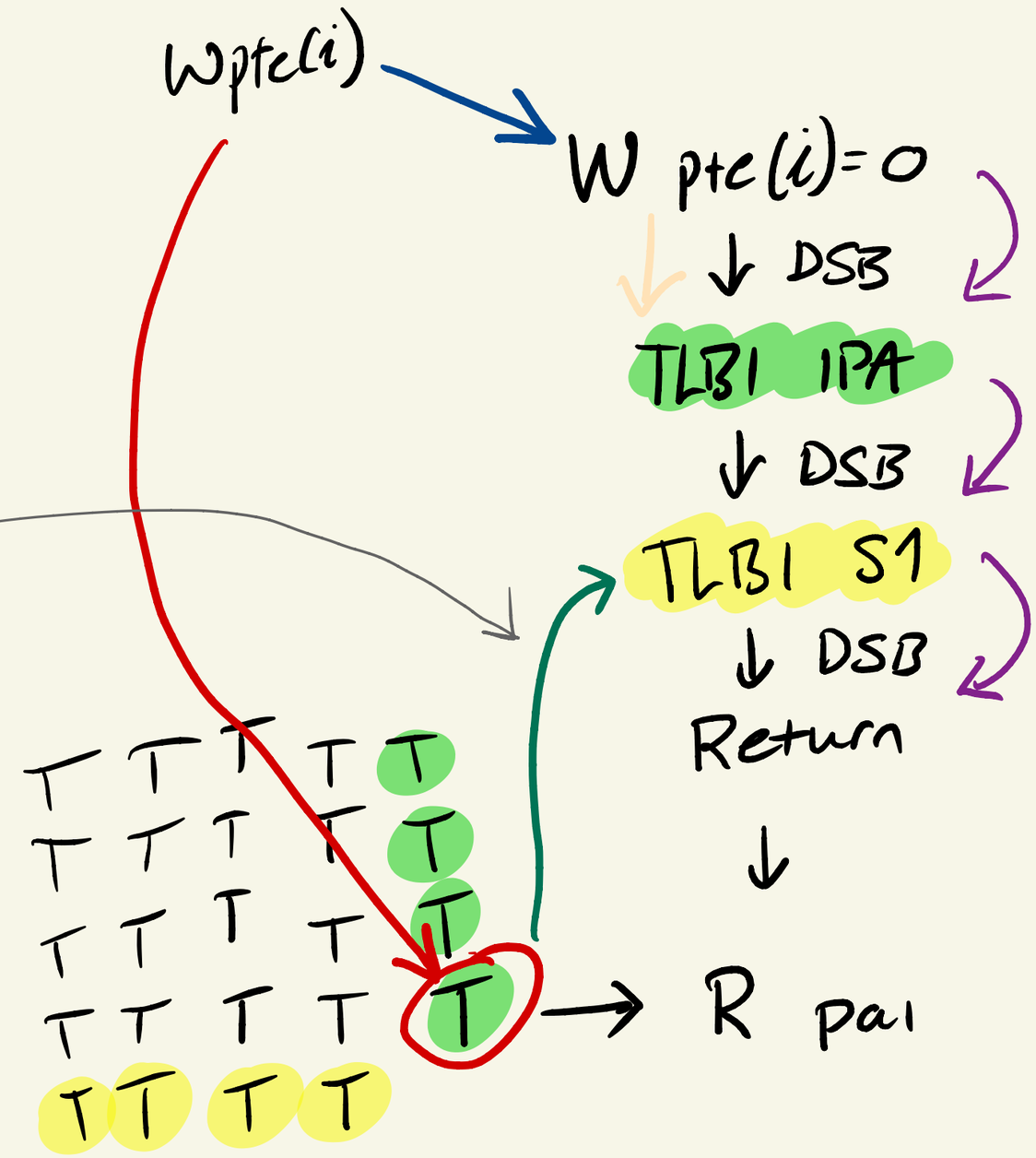


```
let tlb-affects =
    ...

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

(* ordered-before TLBI and translate *)
let obtlbi_translate =
      ...
    | (tcache2 ; wco? ; [TLBI-S1])
        & (iio^-1 ; [T & Stage1] ; maybe_TLB_cached)

(* ordered-before TLBI *)
let obtlbi =
      obtlbi_translate
    | ...

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

(* barrier-ordered-before *)
let bob =
      ...
    | [F | C]; po; [dsbsy]
    | [dsb] ; po

(* Ordered-before *)
let ob = (... | bob | iio | obtlbi | ctxob)^+

irreflexive ob as external
```



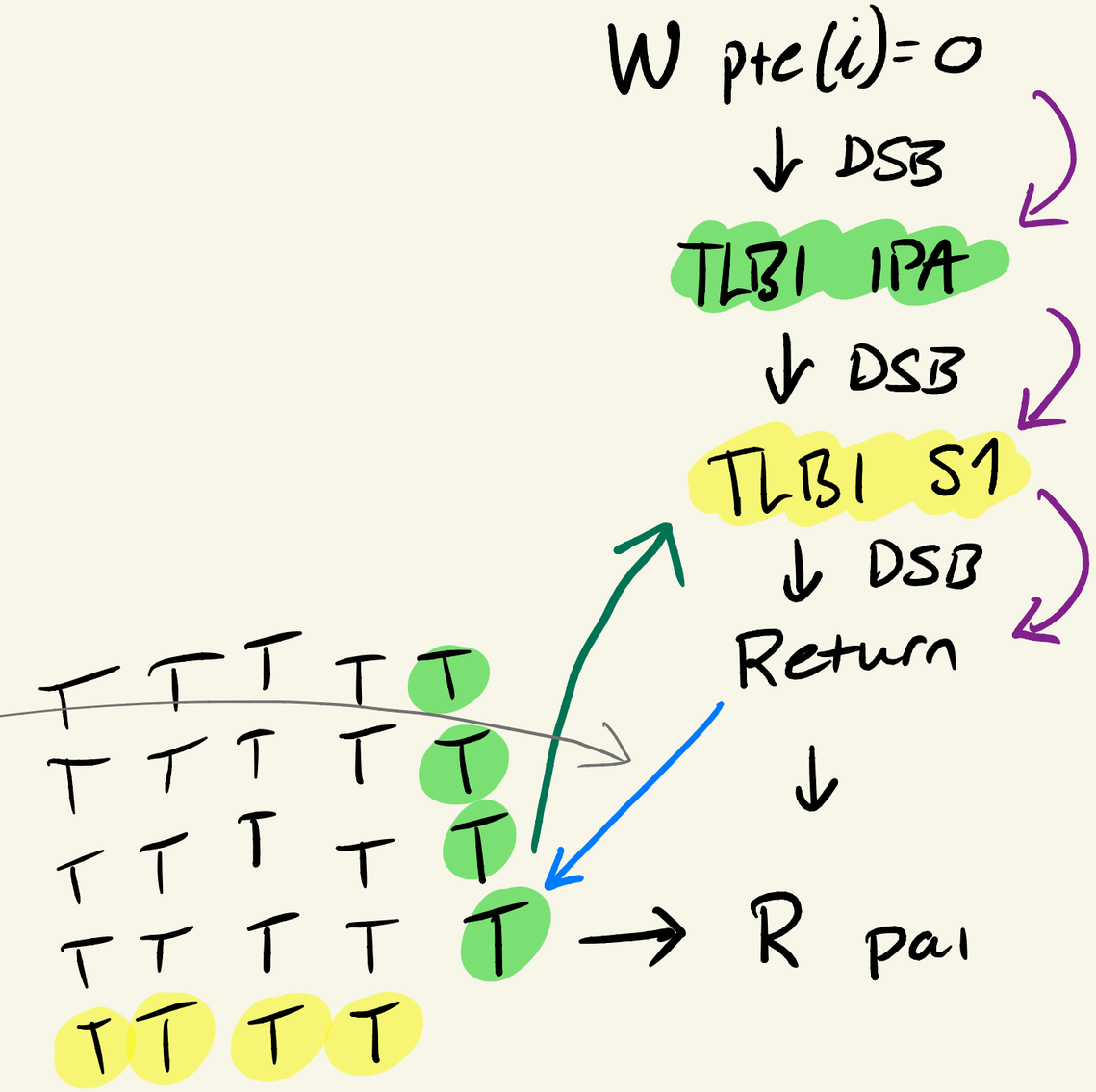
Contributions

```
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

(* ordered-before TLBI and translate *)
let obtlbi_translate =
    ...
  | (tcache2 ; wco? ; [TLBI-S1])
      & (iio^-1 ; [T & Stage1] ; maybe_TLB_cached)

(* ordered-before TLBI *)
let obtlbi =
      obtlbi_translate
    | ...

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

(* barrier-ordered-before *)
let bob =
      ...
    | [F | C]; po; [dsbsy]
    | [dsb] ; po

(* Ordered-before *)
let ob = (... | bob | iio | obtlbi | ctxob)^+

irreflexive ob as external
```



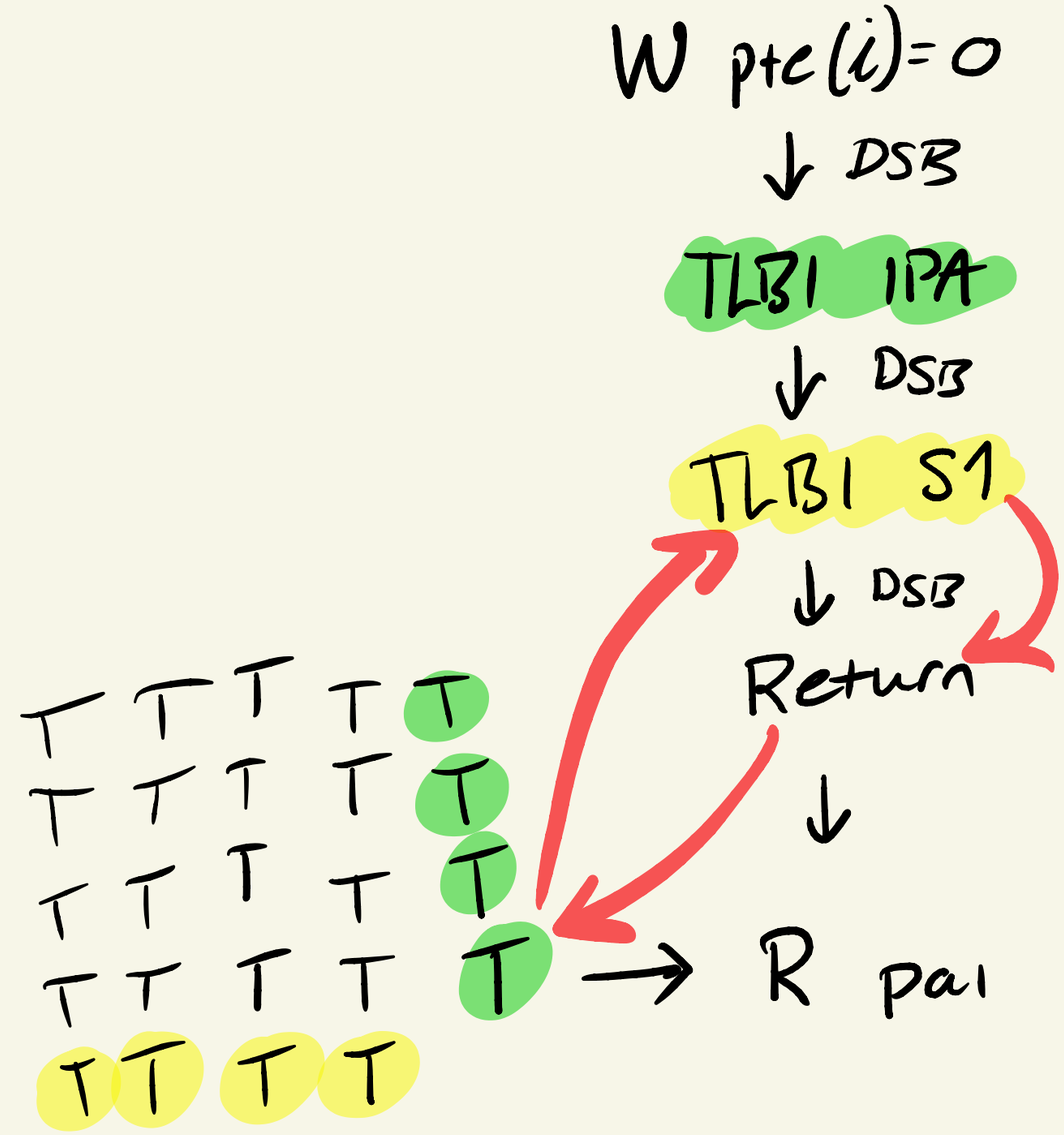
Contributions

```
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

(* ordered-before TLBI and translate *)
let obtlbi_translate =
      ...
    | (tcache2 ; wco? ; [TLBI-S1])
        & (iio^-1 ; [T & Stage1] ; maybe_TLB_cached)

(* ordered-before TLBI *)
let obtlbi =
      obtlbi_translate
    | ...

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

(* barrier-ordered-before *)
let bob =
      ...
    | [F | C]; po; [dsbsy]
    | [dsb] ; po

(* Ordered-before *)
let ob = (... | bob | iio | obtlbi | ctxob)^+

irreflexive ob as external
```



Contributions

```
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

(* ordered-before TLBI and translate *)
let obtlbi_translate =
      ...
    | (tcache2 ; wco? ; [TLBI-S1])
        & (iio^-1 ; [T & Stage1] ; maybe_TLB_cached)

(* ordered-before TLBI *)
let obtlbi =
      obtlbi_translate
    | ...

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

(* barrier-ordered-before *)
let bob =
      ...
    | [F | C]; po; [dsbsy]
    | [dsb] ; po

(* Ordered-before *)
let ob = (... | bob | iio | obtlbi | ctxob)^+

irreflexive ob as external
```



Contributions

Recap

① Clarified Architecture

② Axiomatic model
+ tooling

③ pKVM tests

Future

use the model!

more Arm features

... more architectures?