Program Verification in the Presence of Cached Address Translation

Hira Taqdees Syeda | Gerwin Klein

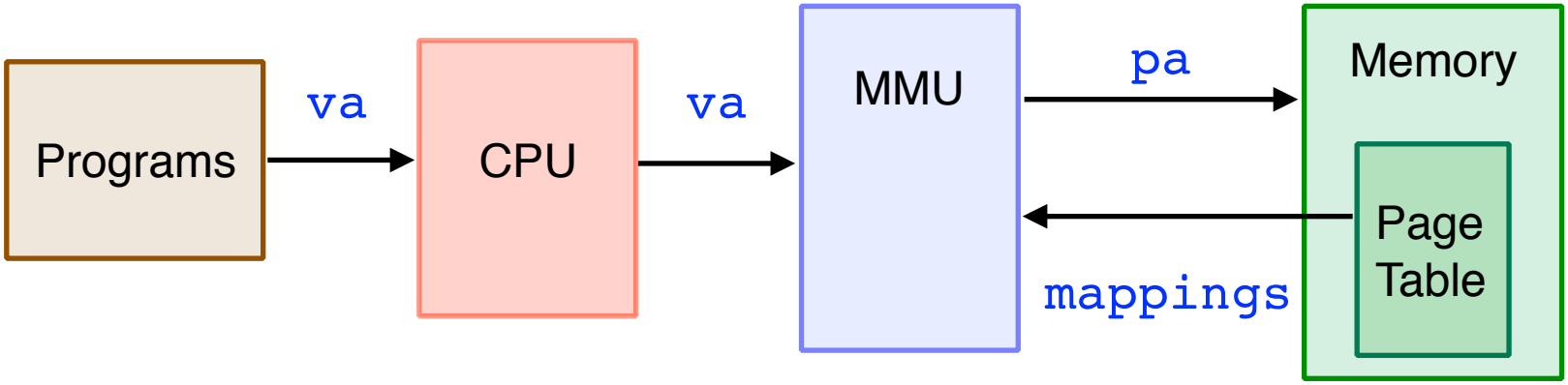
July 2018

www.ts.data61.csiro.au

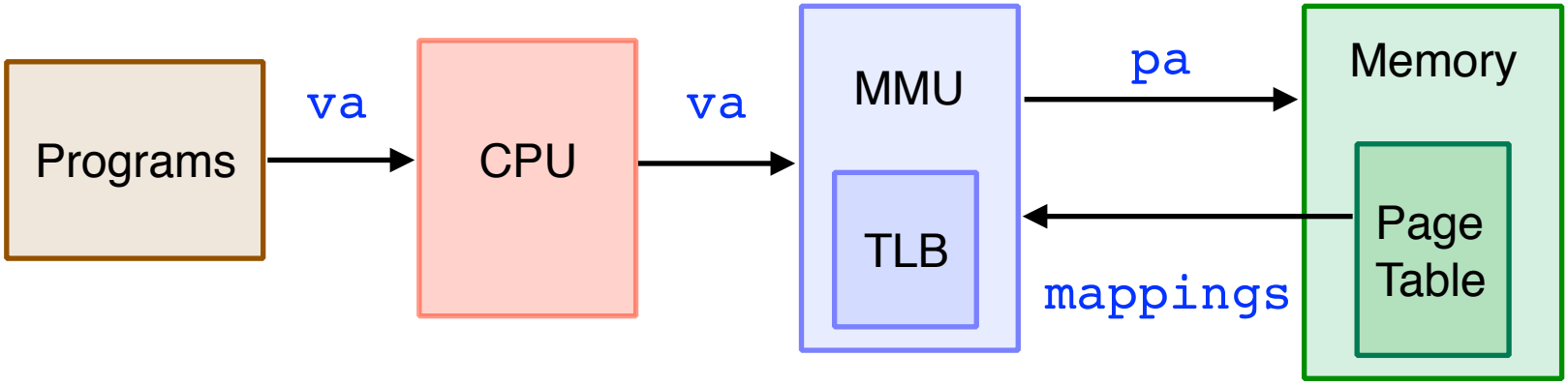
UNSW SYDNEY

CSIRO

What is
Cached Address Translation



What is Cached Address Translation



- Translation Lookaside Buffer (TLB) is
  - a dedicated cache for page table walks
  - architecture specific
  - managed by hardware and operating system together

TLB Effects on Program Execution

TLB Effects on Program Execution

- TLB being cache
  - has *no* functional effects
  - only makes execution faster, *if* maintained correctly
  - is an assumption in formally verified kernels such as seL4

TLB Effects on Program Execution

- TLB being cache
  - has *no* functional effects
  - only makes execution faster, *if* maintained correctly
  - is an assumption in formally verified kernels such as seL4

- Poorly managed TLB leads to
  - memory operations on the wrong addresses
  - inconsistent translation ⇒ system crash

TLB Effects on Program Execution

- TLB being cache
  - has *no* functional effects
  - only makes execution faster, *if* maintained correctly
  - is an assumption in formally verified kernels such as seL4

- Poorly managed TLB leads to
  - memory operations on the wrong addresses
  - inconsistent translation ⇒ system crash

- TLB-aware logic for program reasoning
  - abstract model for ARMv7-style MMU

Contributions

- TLB-aware program logic in Isabelle/HOL
  - sound abstraction of ARMv7-style MMU
  - language with TLB management primitives
  - TLB-aware Hoare logic rules

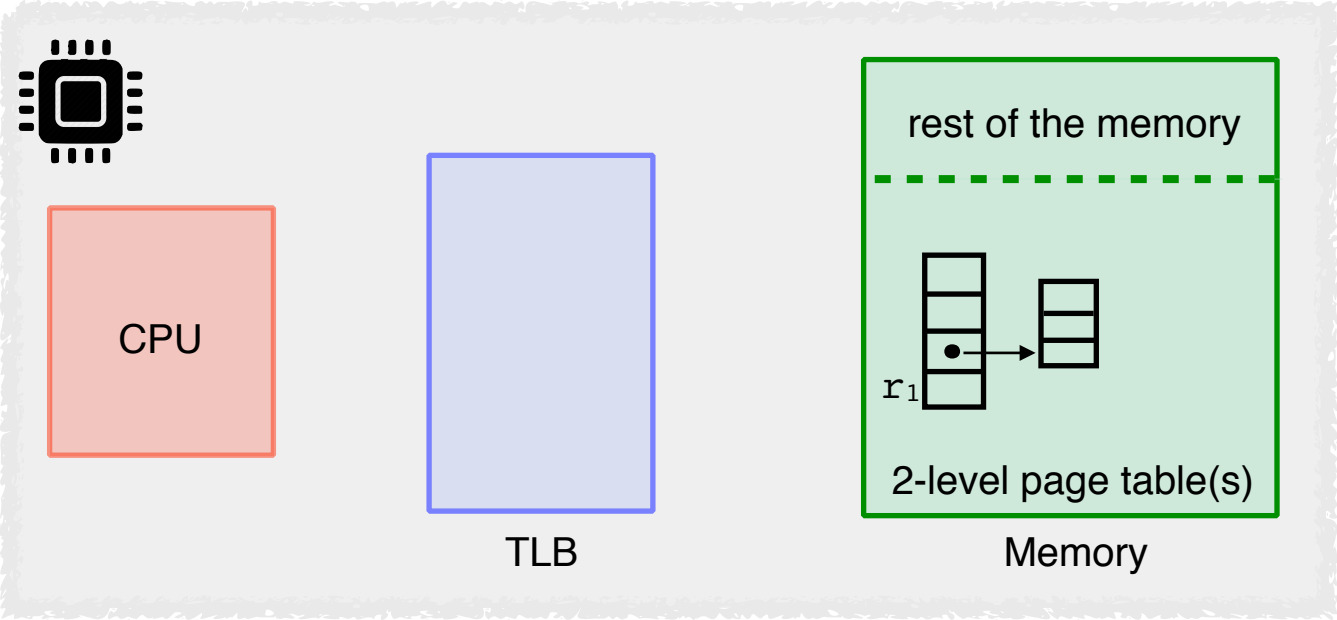
Contributions

- TLB-aware program logic in Isabelle/HOL
  - sound abstraction of ARMv7-style MMU
  - language with TLB management primitives
  - TLB-aware Hoare logic rules
- Reduction theorems for program verification at
  - user- and kernel-level execution
  - context switch

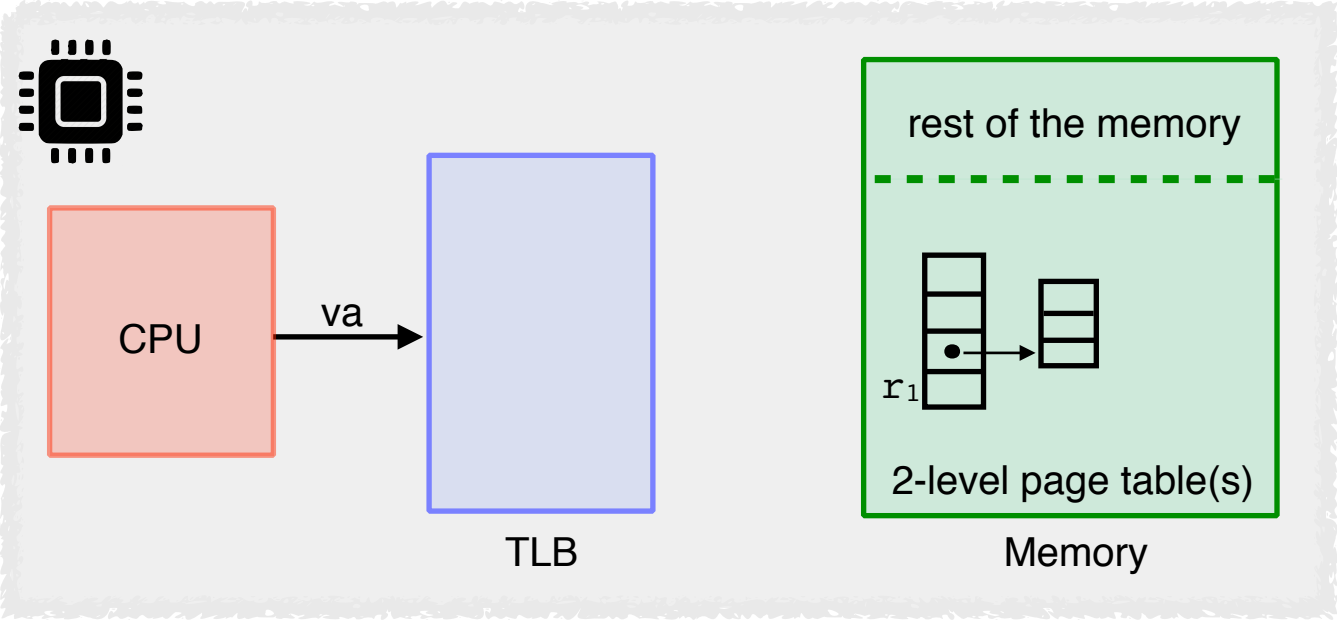
Contributions

- TLB-aware program logic in Isabelle/HOL
  - sound abstraction of ARMv7-style MMU
  - language with TLB management primitives
  - TLB-aware Hoare logic rules
- Reduction theorems for program verification at
  - user- and kernel-level execution
  - context switch

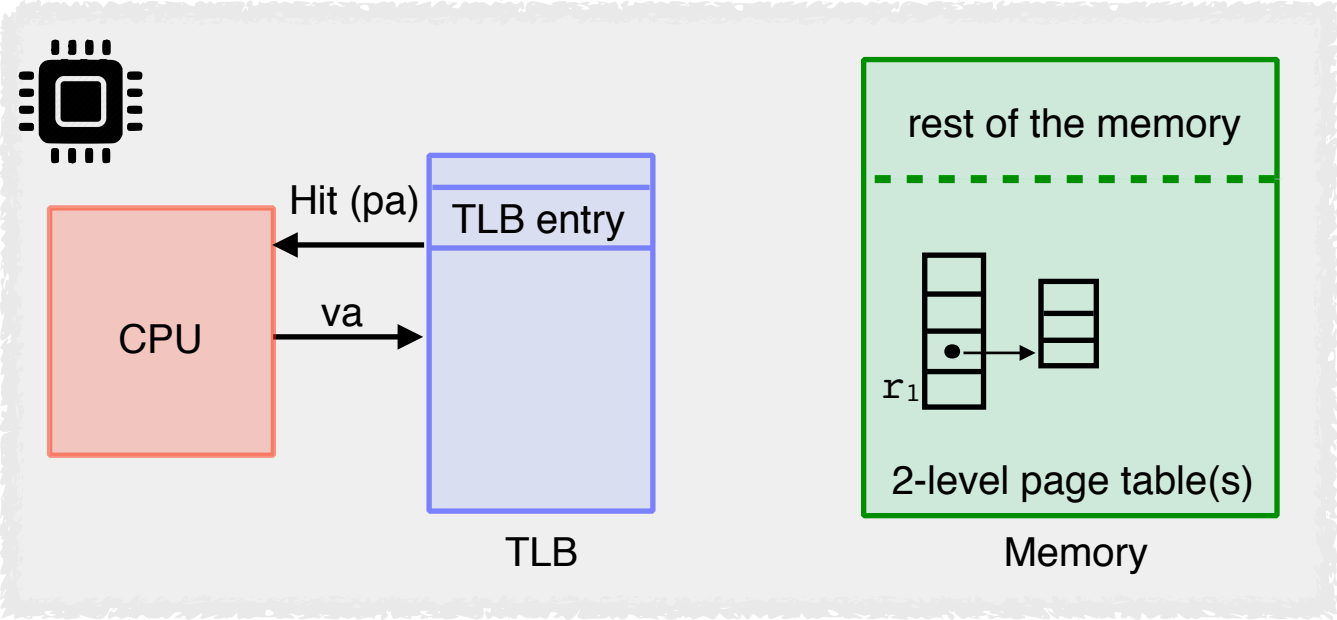
ARMv7-style MMU



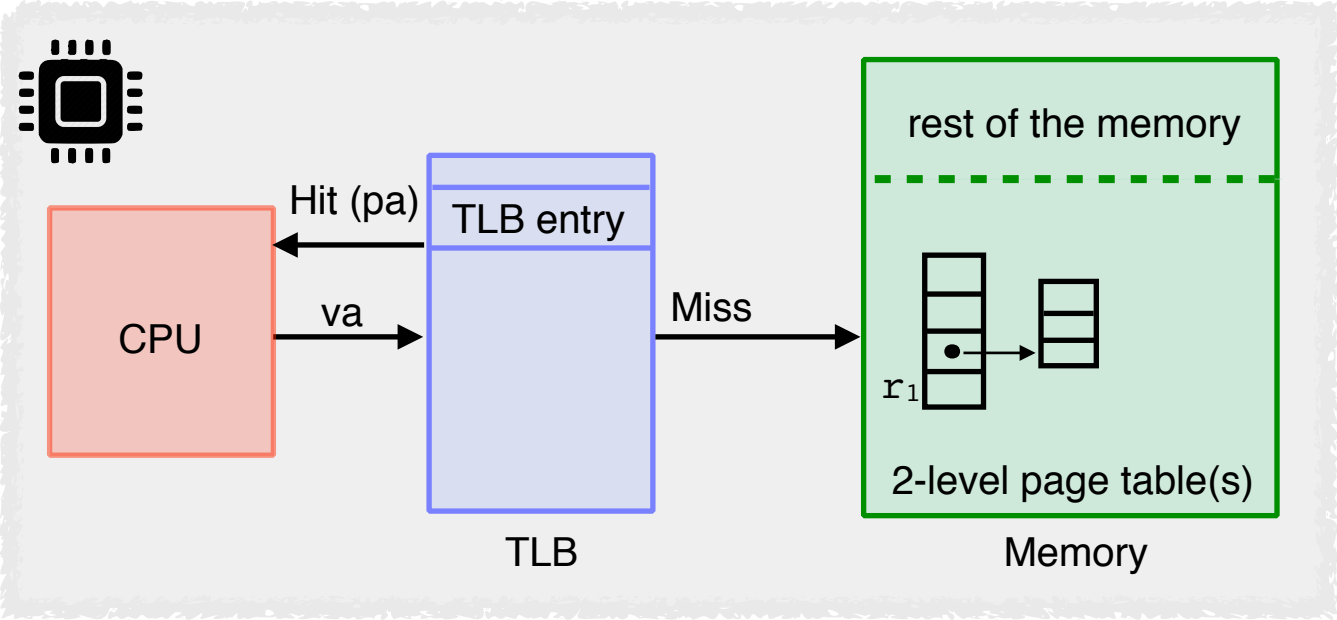
ARMv7-style MMU



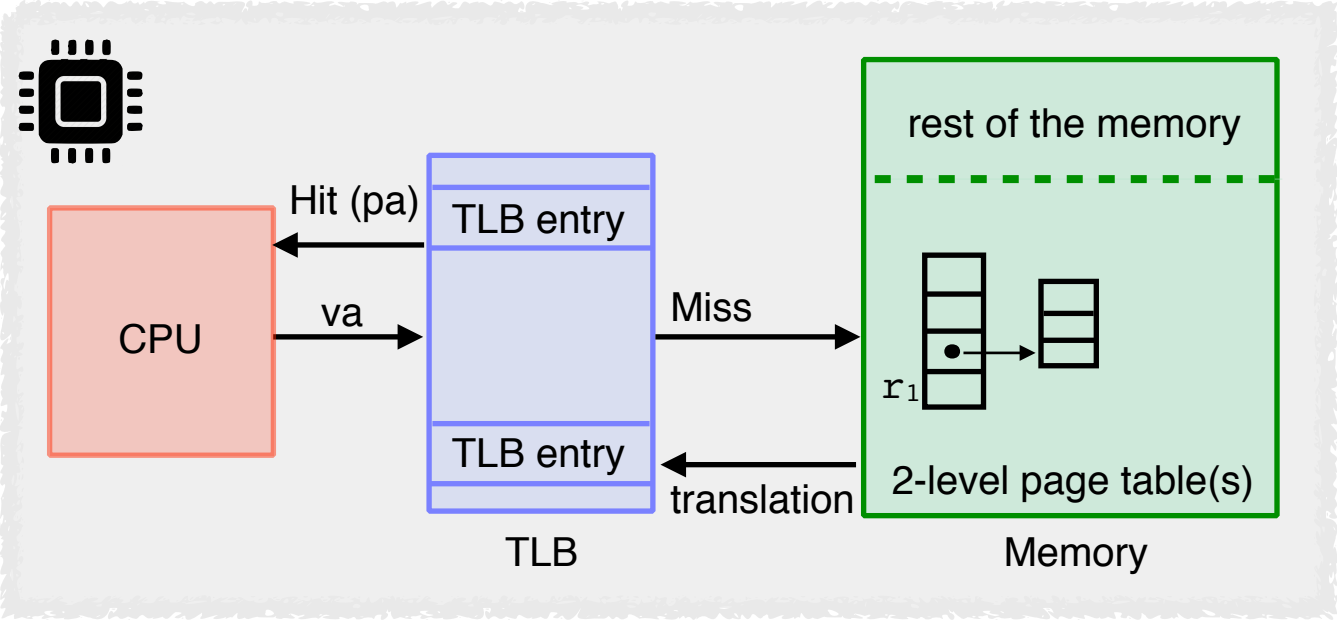
ARMv7-style MMU



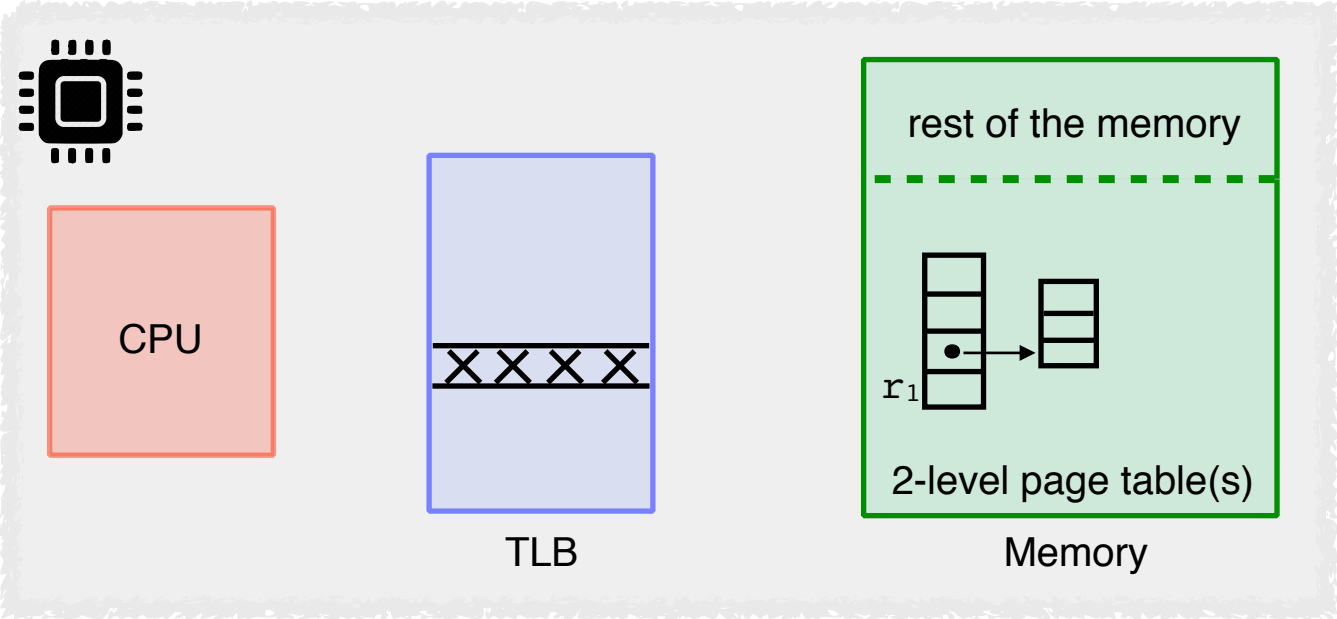
ARMv7-style MMU



ARMv7-style MMU

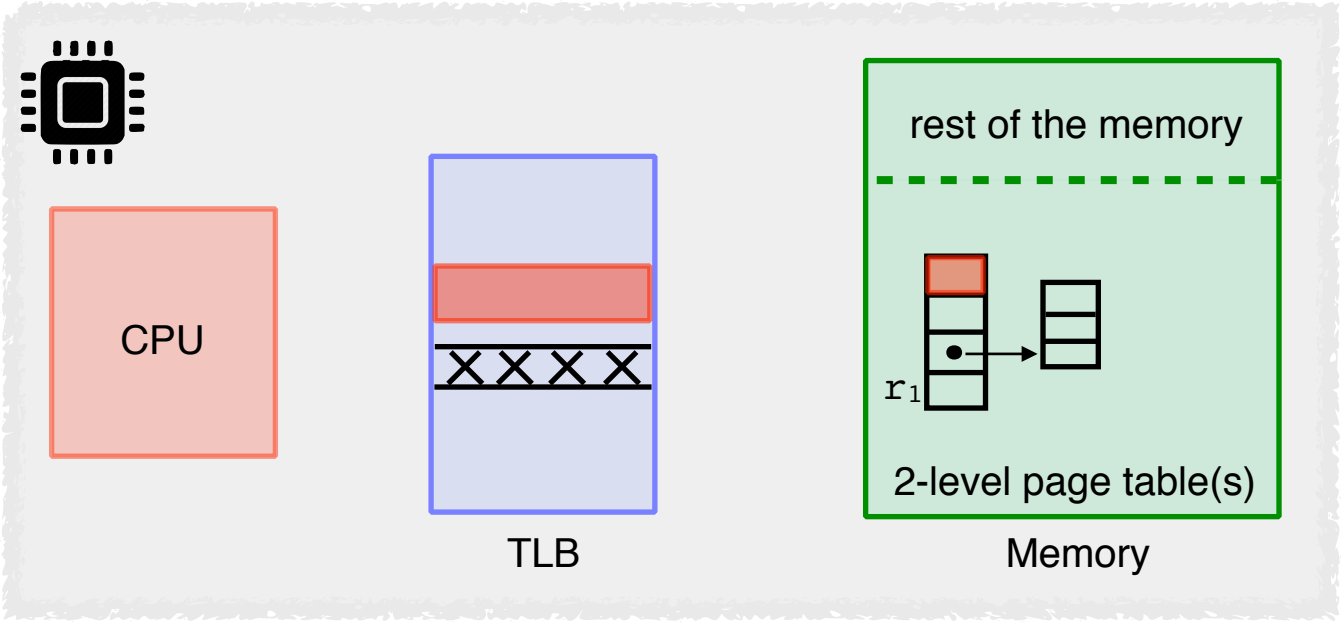


ARMv7-style MMU



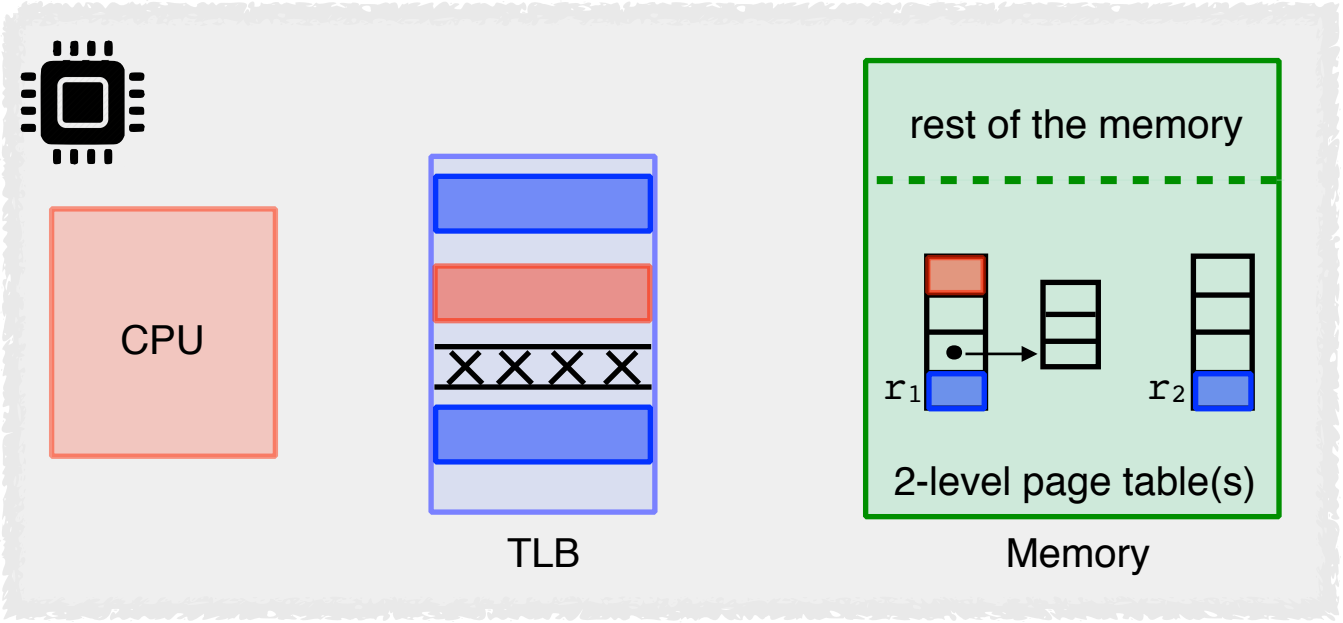
- TLB eviction (××)

ARMv7-style MMU



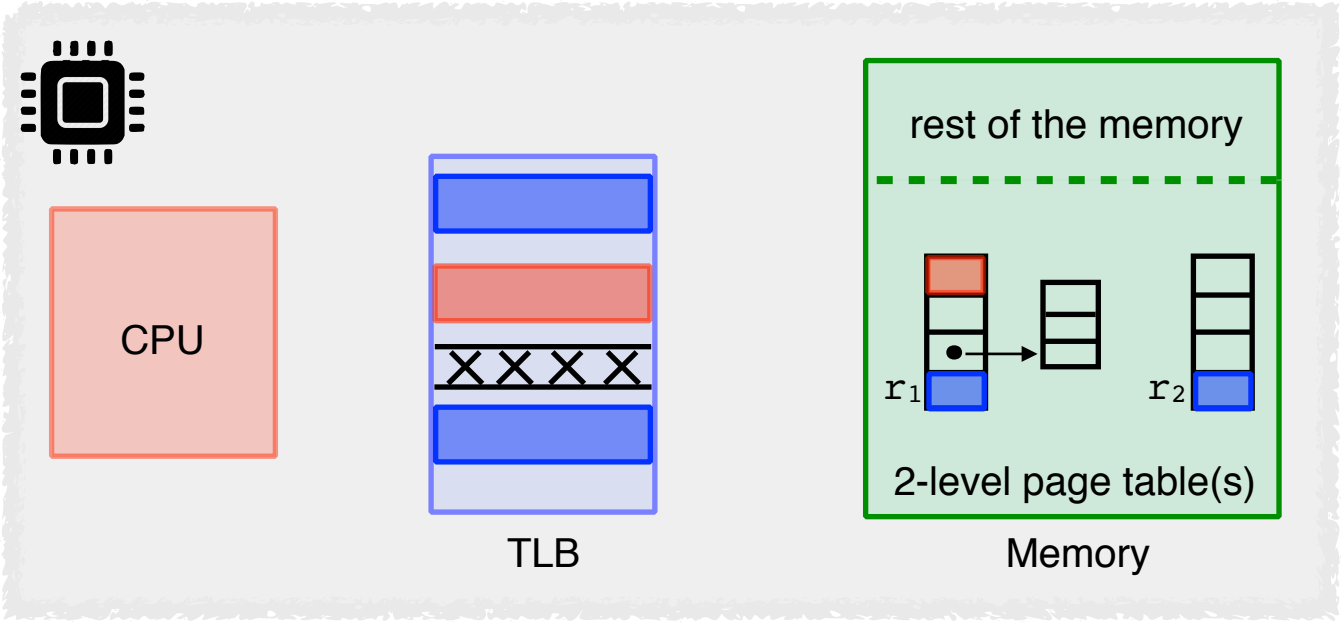
- TLB eviction (✕✕)
- TLB incoherency (▭)
  - stale translation entries w.r.t. page table(s)

ARMv7-style MMU



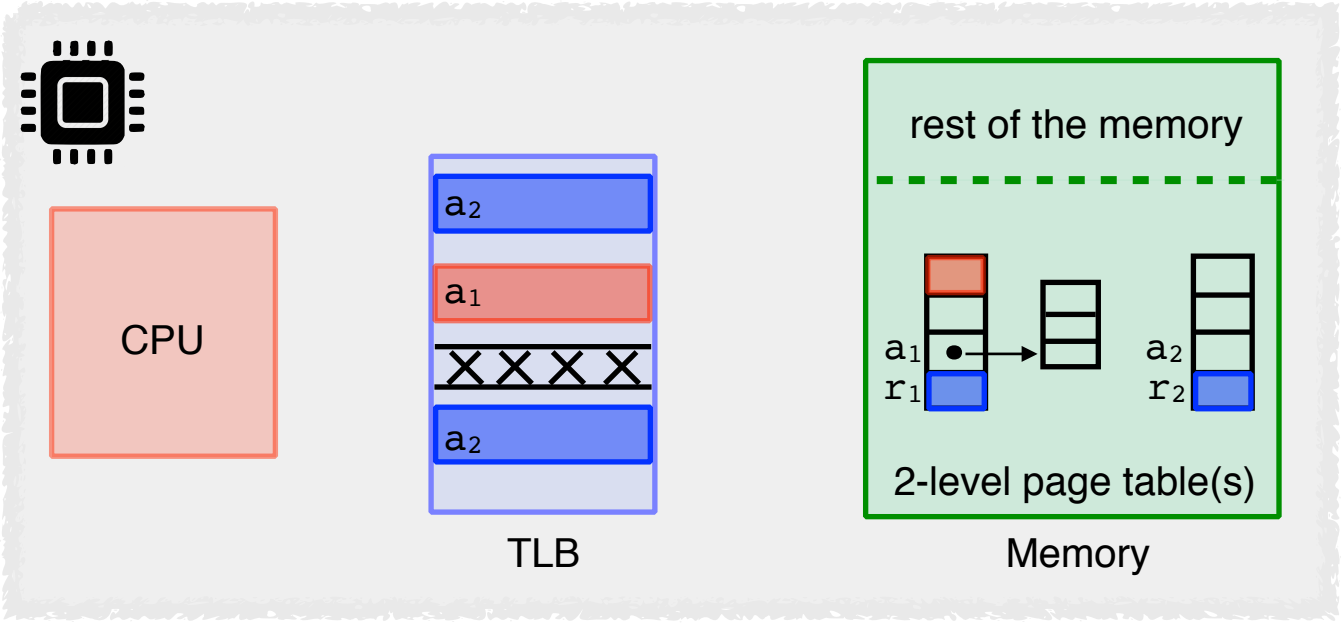
- TLB eviction (✕✕)
- TLB incoherency (▭)
  - stale translation entries w.r.t. page table(s)
- TLB inconsistency (▭)
  - more than one translation entries

ARMv7-style MMU



- TLB maintenance operations after updating
  - page table(s)
  - root register

ARMv7-style MMU



- TLB maintenance operations after updating
  - page table(s)
  - root register
- Selective invalidation using Address Space IDentifier - ASID
  - ASID register

Sound Abstraction of ARMv7-style MMU

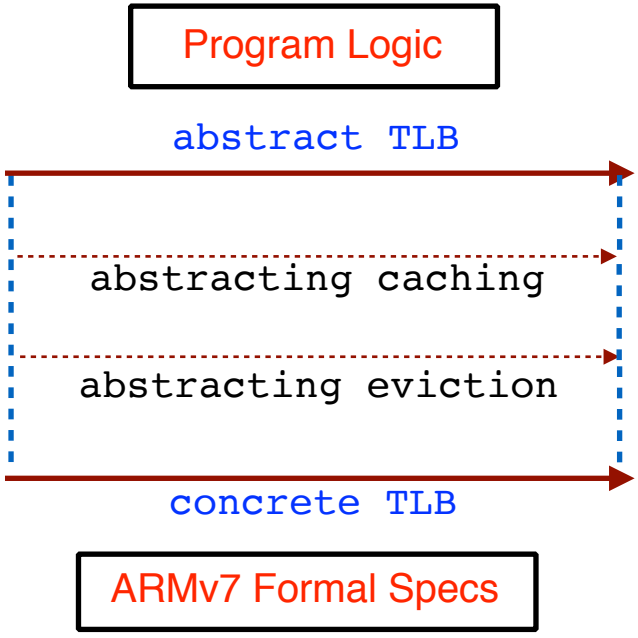
Sound Abstraction of ARMv7-style MMU

- Formalised TLB model
  - hardware details
  - instructions affecting the TLB state

Sound Abstraction of ARMv7-style MMU

- Formalised TLB model
  - hardware details
  - instructions affecting the TLB state

Stepwise data refinement
to achieve functional abstraction

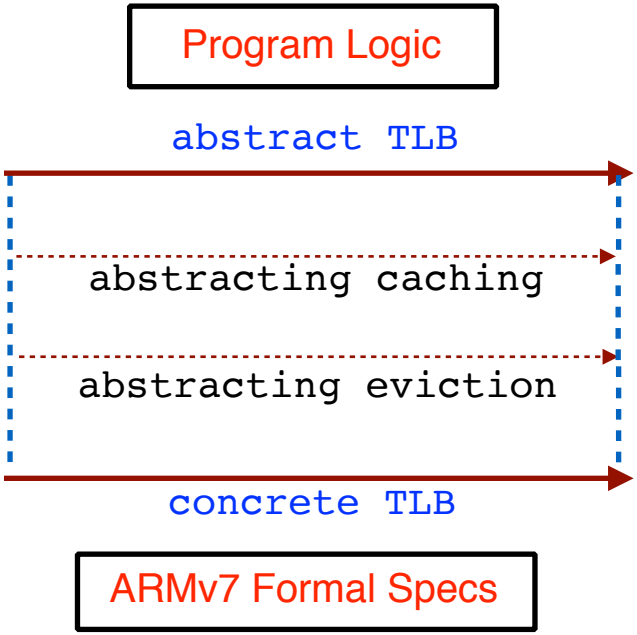


Sound Abstraction of ARMv7-style MMU

- Formalised TLB model
  - hardware details
  - instructions affecting the TLB state

Stepwise data refinement
to achieve functional abstraction

Functionality of a TLB
is captured by the record of
inconsistent virtual addresses

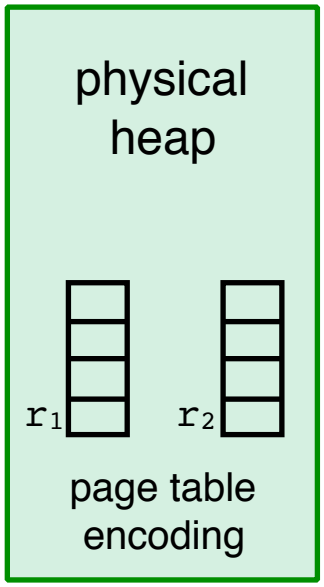


Sound Abstraction of ARMv7-style MMU

processor mode

active ASID

active root



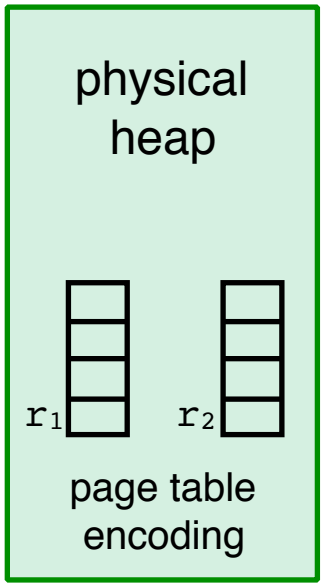
abstract TLB

Sound Abstraction of ARMv7-style MMU

processor mode

active ASID

active root



abstract TLB

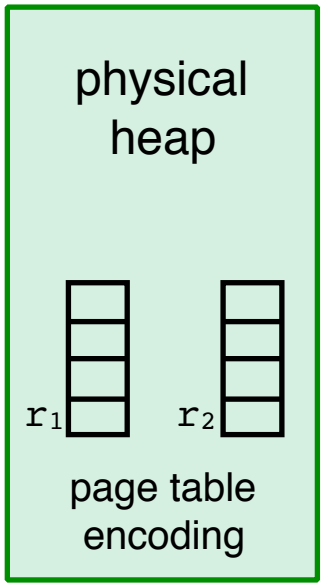
incon_set	pt_snapshot
set of inconsistent vaddrs for active ASID	page table state for all inactive ASIDs

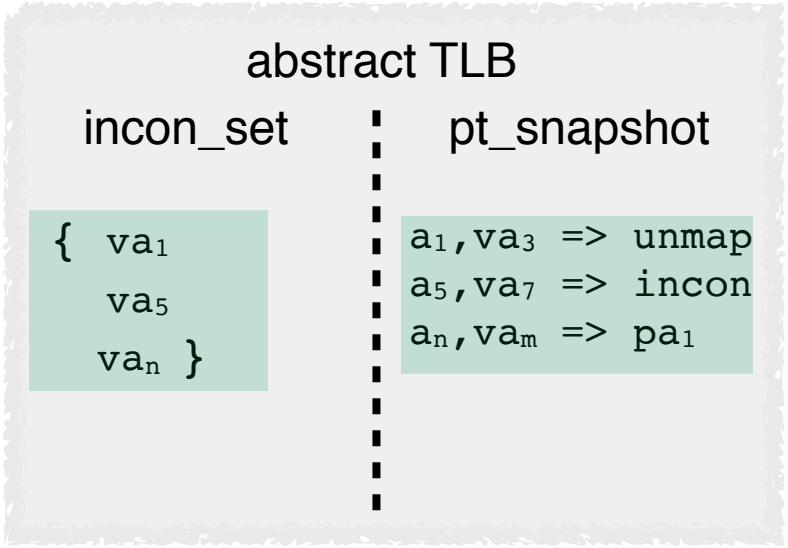
Sound Abstraction of ARMv7-style MMU

processor mode

active ASID

active root





Contributions

- TLB-aware program logic in Isabelle/HOL
  - sound abstraction of ARMv7-style MMU ✓
  - language with TLB management primitives
  - TLB-aware Hoare logic rules
- Reduction theorems for program verification at
  - user- and kernel-level execution
  - context switch

Language — Syntax

- Heap language with TLB management primitives
  - arithmetic expressions **`aexp`**
    - constant, unary and binary operations
    - HeapLookup

Language — Syntax

- Heap language with TLB management primitives
  - arithmetic expressions **`aexp`**
    - constant, unary and binary operations
    - HeapLookup
  - boolean expressions **`bexp`**
    - negation, comparison and binary operations

Language — Syntax

- Heap language with TLB management primitives
  - commands
    - skip and sequence
    - if-then-else and while

Language — Syntax

- Heap language with TLB management primitives
  - commands
    - skip and sequence
    - if-then-else and while
    - assignment `aexp := aexp`

Language — Syntax

- Heap language with TLB management primitives
  - commands
    - skip and sequence
    - if-then-else and while
    - assignment `aexp := aexp`
    - flush
      - `flushALL`, `flushASID`, `flushVA` and `flushASIDVA`

Language — Syntax

- Heap language with TLB management primitives
  - commands
    - skip and sequence
    - if-then-else and while
    - assignment `aexp := aexp`
    - flush
      - `flushALL`, `flushASID`, `flushVA` and `flushASIDVA`
    - updateRoot
    - updateASID
    - updateMode
      - `kernel` or `user`

Contributions

- TLB-aware program logic in Isabelle/HOL
  - sound abstraction of ARMv7-style TLB ✓
  - language with TLB management primitives ✓
  - TLB-aware Hoare logic rules
- Reduction theorems for program verification at
  - user- and kernel-level execution
  - context switch

Program Logic

- Operational semantics
  - **`aexp`** — partial function from **`state`** to 32-bit **`value`**
  - **`bexp`** — partial function from **`state`** to **`bool`**
  - **`command`** — relation between **`state`** and **`state option`**

Program Logic

- Operational semantics
  - **`aexp`** — partial function from **`state`** to 32-bit **`value`**
  - **`bexp`** — partial function from **`state`** to **`bool`**
  - **`command`** — relation between **`state`** and **`state option`**

- Hoare triple — $\{\!|P|\!\}\ c\ \{\!|Q|\!\}$
  - soundness is derived directly from the operational semantics
  - logic rules are in weakest-precondition form

Program Logic — Rules

- assignment
- updateRoot
- updateASID

☆ other rules are in standard Hoare logic form

Program Logic — Rules

- assignment

$$\models \{\!|\lambda \mathtt{s}.\ [\![\mathtt{l}]\!]\ \mathtt{s} = \lfloor \mathtt{vp} \rfloor \wedge [\![\mathtt{r}]\!]\ \mathtt{s} = \lfloor \mathtt{v} \rfloor \wedge \mathtt{vp} \notin \mathcal{IC}\ \mathtt{s} \wedge \mathtt{Addr\ vp} \hookrightarrow_s \mathtt{pp} \wedge \mathtt{P\ (heap\_iset\_update}_s\ \mathtt{(pp} \mapsto \mathtt{v))}|\!\}$$
$$\mathtt{l ::= r}\ \{\!|\mathtt{P}|\!\}$$

Program Logic — Rules

- assignment

$$\models \{\!|\lambda \mathtt{s}.\ [\![\mathtt{l}]\!]\ \mathtt{s} = \lfloor \mathtt{vp} \rfloor \wedge [\![\mathtt{r}]\!]\ \mathtt{s} = \lfloor \mathtt{v} \rfloor \wedge \mathtt{vp} \notin \mathcal{IC}\ \mathtt{s} \wedge \mathtt{Addr\ vp} \hookrightarrow_s \mathtt{pp} \wedge \mathtt{P\ (heap\_iset\_update}_s\ \mathtt{(pp} \mapsto \mathtt{v))}|\!\}$$
$$\mathtt{l ::= r}\ \{\!|\mathtt{P}|\!\}$$

- ✓ successful evaluation of l to a vp
- ✓ consistency of vp
- ✓ valid address translation of vp to pp
- ✓ reasoning about `heap` and `incon_set` update

Program Logic — Rules

- assignment

$$\vdash \{\!|\lambda \mathtt{s}.\ [\![\mathtt{l}]\!]\ \mathtt{s} = \lfloor \mathtt{vp} \rfloor \wedge [\![\mathtt{r}]\!]\ \mathtt{s} = \lfloor \mathtt{v} \rfloor \wedge \mathtt{vp} \notin \mathcal{IC}\ \mathtt{s} \wedge \mathtt{Addr\ vp} \hookrightarrow_s \mathtt{pp} \wedge \mathtt{P\ (heap\_iset\_update}_s\ \mathtt{(pp} \mapsto \mathtt{v))} |\!\}$$
$$\mathtt{l ::= r}\ \{\!|\mathtt{P}|\!\}$$

- ✓ successful evaluation of l to a vp
- ✓ consistency of vp
- ✓ valid address translation of vp to pp
- ✓ reasoning about `heap` and `incon_set` update

Program Logic — Rules

- assignment

$$\models \{\!|\lambda s.\ [\![\mathtt{l}]\!]\ s = \lfloor \mathtt{vp} \rfloor \wedge [\![\mathtt{r}]\!]\ s = \lfloor \mathtt{v} \rfloor \wedge \mathtt{vp} \notin \mathcal{IC}\ s \wedge \mathtt{Addr\ vp} \hookrightarrow_s \mathtt{pp} \wedge \mathtt{P\ (heap\_iset\_update}_s\ \mathtt{(pp} \mapsto \mathtt{v))}|\!\}$$
$$\mathtt{l ::= r}\ \{\!|\mathtt{P}|\!\}$$

- ✓ successful evaluation of l to a vp
- ✓ consistency of vp
- ✓ valid address translation of vp to pp
- ✓ reasoning about `heap` and `incon_set` update

Program Logic — Rules

- assignment

$$\models \{\!|\lambda s.\ [\![\mathtt{l}]\!]\ s = \lfloor \mathtt{vp} \rfloor \wedge [\![\mathtt{r}]\!]\ s = \lfloor \mathtt{v} \rfloor \wedge \mathtt{vp} \notin \mathcal{IC}\ s \wedge \mathtt{Addr\ vp} \hookrightarrow_s \mathtt{pp} \wedge \mathtt{P\ (heap\_iset\_update}_s\ \mathtt{(pp} \mapsto \mathtt{v))}|\!\}\ \mathtt{l ::= r}\ \{\!|\mathtt{P}|\!\}$$

- ✓ successful evaluation of l to a vp
- ✓ consistency of vp
- ✓ valid address translation of vp to pp
- ✓ reasoning about `heap` and `incon_set` update

Program Logic — Rules

- assignment

$$\models \{\!|\lambda \mathtt{s}.\ [\![\mathtt{l}]\!]\ \mathtt{s} = \lfloor \mathtt{vp} \rfloor \wedge [\![\mathtt{r}]\!]\ \mathtt{s} = \lfloor \mathtt{v} \rfloor \wedge \mathtt{vp} \notin \mathcal{IC}\ \mathtt{s} \wedge \mathtt{Addr\ vp} \hookrightarrow_s \mathtt{pp} \wedge \mathtt{P\ (heap\_iset\_update}_s\ \mathtt{(pp} \mapsto \mathtt{v))}|\!\}$$
$$\mathtt{l ::= r}\ \{\!|\mathtt{P}|\!\}$$

- ✓ successful evaluation of l to a vp
- ✓ consistency of vp
- ✓ valid address translation of vp to pp
- ✓ reasoning about `heap` and `incon_set` update

Program Logic — Rules

- assignment

$$\models \{\!|\lambda s.\ [\![\mathtt{l}]\!]\ s = \lfloor \mathtt{vp} \rfloor \wedge [\![\mathtt{r}]\!]\ s = \lfloor \mathtt{v} \rfloor \wedge \mathtt{vp} \notin \mathcal{IC}\ s \wedge \mathtt{Addr\ vp} \hookrightarrow_s \mathtt{pp} \wedge \mathtt{P\ (heap\_iset\_update}_s\ \mathtt{(pp} \mapsto \mathtt{v))}|\!\}$$
$$\mathtt{l ::= r}\ \{\!|\mathtt{P}|\!\}$$

✓ `incon_set` update

comparison of the active page table
before and after the assignment for
all remapped and unmapped addresses

Program Logic — Rules

- assignment

$$\models \{\!|\lambda s.\ [\![l]\!]\ s = \lfloor vp \rfloor \wedge [\![r]\!]\ s = \lfloor v \rfloor \wedge vp \notin \mathcal{IC}\ s \wedge \mathtt{Addr}\ vp \hookrightarrow_s pp \wedge P\ (\mathtt{heap\_iset\_update}_s\ (pp \mapsto v))|\!\}$$
$$l ::= r \quad \{\!|P|\!\}$$

✓ `incon_set` update

before assignment	after assignment
`incon_set : { va1 , va2 }`	
`va3 is mapped to pa3`	
`va4 is mapped to pa4`	

Program Logic — Rules

- assignment

$$\models \{\!|\lambda s.\ [\![l]\!]\ s = \lfloor vp \rfloor \wedge [\![r]\!]\ s = \lfloor v \rfloor \wedge vp \notin \mathcal{IC}\ s \wedge \mathtt{Addr}\ vp \hookrightarrow_s pp \wedge P\ (\mathtt{heap\_iset\_update}_s\ (pp \mapsto v))|\!\}$$
$$l ::= r \ \{\!|P|\!\}$$

✓ `incon_set` update

before assignment	after assignment
`incon_set : { va1 , va2 }`	
`va3 is mapped to pa3`	
`va4 is mapped to pa4`	

Program Logic — Rules

- assignment

$$\models \{\!|\lambda s.\ [\![l]\!]\ s = \lfloor vp \rfloor \wedge [\![r]\!]\ s = \lfloor v \rfloor \wedge vp \notin \mathcal{IC}\ s \wedge \mathtt{Addr}\ vp \hookrightarrow_s pp \wedge \mathtt{P}\ (\mathtt{heap\_iset\_update}_s\ (pp \mapsto v))|\!\}$$
$$l ::= r \ \{\!|P|\!\}$$

✓ `incon_set` update

before assignment	after assignment
`incon_set : { va1 , va2 }`	
`va3 is mapped to pa3`	
`va4 is mapped to pa4`	

Program Logic — Rules

- assignment

$$\models \{\!|\lambda s.\ [\![l]\!]\ s = \lfloor vp \rfloor \wedge [\![r]\!]\ s = \lfloor v \rfloor \wedge vp \notin \mathcal{IC}\ s \wedge \mathtt{Addr}\ vp \hookrightarrow_s pp \wedge \mathtt{P}\ (\mathtt{heap\_iset\_update}_s\ (pp \mapsto v))|\!\}$$
$$l ::= r \ \{\!|P|\!\}$$

✓ `incon_set` update

before assignment	after assignment
`incon_set : { va1 , va2 }`	
`va3 is mapped to pa3`	`va3 is remapped to pa5`
`va4 is mapped to pa4`	`va4 is unmapped`

Program Logic — Rules

- assignment

$$\vdash \{\!|\lambda s.\ [\![l]\!]\ s = \lfloor vp \rfloor \wedge [\![r]\!]\ s = \lfloor v \rfloor \wedge vp \notin \mathcal{IC}\ s \wedge \mathtt{Addr}\ vp \hookrightarrow_s pp \wedge \mathtt{P}\ (\mathtt{heap\_iset\_update}_s\ (pp \mapsto v))|\!\}$$
$$l ::= r \ \{\!|P|\!\}$$

✓ `incon_set` update

before assignment	after assignment
`incon_set : { va1 , va2 }`	
`va3 is mapped to pa3`	`va3 is remapped to pa5`
`va4 is mapped to pa4`	`va4 is unmapped`

Program Logic — Rules

- assignment

$$\models \{\!|\lambda s.\ [\![l]\!]\ s = \lfloor vp \rfloor \wedge [\![r]\!]\ s = \lfloor v \rfloor \wedge vp \notin \mathcal{IC}\ s \wedge \mathtt{Addr}\ vp \hookrightarrow_s pp \wedge \mathtt{P}\ (\mathtt{heap\_iset\_update}_s\ (pp \mapsto v))|\!\}$$
$$l ::= r \ \{\!|P|\!\}$$

✓ `incon_set` update

before assignment	after assignment
`incon_set : { va1 , va2 }`	
`va3 is mapped to pa3`	`va3 is remapped to pa5`
`va4 is mapped to pa4`	`va4 is unmapped`

Program Logic — Rules

- assignment

$$\models \{\!|\lambda s.\ [\![l]\!]\ s = \lfloor vp \rfloor \wedge [\![r]\!]\ s = \lfloor v \rfloor \wedge vp \notin \mathcal{IC}\ s \wedge \mathtt{Addr}\ vp \hookrightarrow_s pp \wedge P\ (\mathtt{heap\_iset\_update}_s\ (pp \mapsto v))|\!\}$$
$$l ::= r \ \{\!|P|\!\}$$

✓ `incon_set` update

before assignment	after assignment
`incon_set : { va1 , va2 }`	`incon_set : { va1 , va2 , va3 , va4 }`
`va3 is mapped to pa3`	`va3 is remapped to pa5`
`va4 is mapped to pa4`	`va4 is unmapped`

Program Logic — Rules

- assignment ✓
- updateRoot
- updateASID

Program Logic — Rules

- updateRoot

$$\models \{\!|\lambda s.\ \texttt{kernel s} \wedge [\![\texttt{rte}]\!]\ \texttt{s} = \lfloor \texttt{rt} \rfloor \wedge \texttt{P (root\_iset\_update}_s\ \texttt{Addr rt)}|\!\}$$
$$\texttt{updateRoot rte}\ \{\!|\texttt{P}|\!\}$$

✓ available in kernel mode

✓ reasoning about `root` and `incon_set` update

Program Logic — Rules

- updateRoot

$$\models \{\!|\lambda s.\ \texttt{kernel s} \wedge [\![\texttt{rte}]\!]\ \texttt{s} = \lfloor \texttt{rt} \rfloor \wedge \texttt{P (root\_iset\_update}_s\ \texttt{Addr rt)}|\!\}\ \texttt{updateRoot rte}\ \{\!|\texttt{P}|\!\}$$

✓ available in kernel mode

✓ reasoning about `root` and `incon_set` update

Program Logic — Rules

- updateRoot

$$\models \{\!|\lambda s.\ \texttt{kernel s} \wedge [\![\texttt{rte}]\!]\ \texttt{s} = \lfloor \texttt{rt} \rfloor \wedge \texttt{P (root\_iset\_update}_s\ \texttt{Addr rt)}|\!\}\ \texttt{updateRoot rte}\ \{\!|\texttt{P}|\!\}$$

✓ available in kernel mode

✓ reasoning about `root` and `incon_set` update

Program Logic — Rules

- updateRoot

$$\vdash \{\!|\lambda s.\ \texttt{kernel s} \wedge [\![\texttt{rte}]\!]\ s = \lfloor \texttt{rt} \rfloor \wedge \texttt{P (root\_iset\_update}_s\ \texttt{Addr rt)}|\!\}\ \texttt{updateRoot rte}\ \{\!|\texttt{P}|\!\}$$

✓ available in kernel mode

✓ reasoning about `root` and `incon_set` update

`incon_set` update:

comparison of the two page tables
before and after updating root for
all remapped and unmapped addresses

Program Logic — Rules

- assignment ✓
- updateRoot ✓
- updateASID

Program Logic — Rules

- updateASID

$$\{\!|\lambda s.\ \texttt{kernel s} \wedge \texttt{P (asid\_pt\_snpshot\_update}_s\ \texttt{a)}|\!\}\ \texttt{updateASID a}\ \{\!|\texttt{P}|\!\}$$

✓ available in kernel mode

✓ reasoning about `asid, incon_set` and `pt_snapshot` update

Program Logic — Rules

- updateASID

$$\{\lambda s.\ \texttt{kernel s} \wedge \texttt{P (asid\_pt\_snpshot\_update}_s\ \texttt{a)}\}\ \texttt{updateASID a}\ \{\texttt{P}\}$$

✓ available in kernel mode

✓ reasoning about `asid, incon_set` and `pt_snapshot` update

Program Logic — Rules

- updateASID

$$\{\lambda s.\ \texttt{kernel s} \wedge \texttt{P (asid\_pt\_snpshot\_update}_s\ \texttt{a)}\}\ \texttt{updateASID a}\ \{\texttt{P}\}$$

✓ available in kernel mode

✓ reasoning about `asid, incon_set` and `pt_snapshot` update

Program Logic — Rules

- updateASID

$$\{\lambda s.\ \texttt{kernel s} \wedge \texttt{P (asid\_pt\_snpshot\_update}_s\ \texttt{a)}\}\ \texttt{updateASID a}\ \{\texttt{P}\}$$

Steps:

Program Logic — Rules

- updateASID

$\{\lambda s.\ \texttt{kernel s} \wedge \texttt{P (asid\_pt\_snpshot\_update}_s\ \texttt{a)}\}\ \texttt{updateASID a}\ \{\texttt{P}\}$

Steps:

1

store the `incon_set` and the page table of the active ASID to the `pt_snapshot`

Program Logic — Rules

- updateASID

$\{\lambda s.\ \texttt{kernel s} \wedge \texttt{P (asid\_pt\_snpshot\_update}_s\ \texttt{a)}\}\ \texttt{updateASID a}\ \{\texttt{P}\}$

Steps:

1 store the `incon_set` and the page table of the active ASID to the `pt_snapshot`

2 update the ASID

Program Logic — Rules

- updateASID

$\{\lambda s.\ \texttt{kernel s} \wedge \texttt{P (asid\_pt\_snpshot\_update}_s\ \texttt{a)}\}\ \texttt{updateASID a}\ \{\texttt{P}\}$

Steps:

1 store the `incon_set` and the page table of the active ASID to the `pt_snapshot`

2 update the ASID

3 compute new `incon_set` from the `pt_snapshot` and the active page table

Program Logic — Rules

- updateASID

$\{\!|\lambda s.\ \texttt{kernel s} \wedge \texttt{P (asid\_pt\_snpshot\_update}_s\ \texttt{a)}|\!\}\ \texttt{updateASID a}\ \{\!|\texttt{P}|\!\}$

Steps: switching from a_1 to a_2

before updateASID	after updateASID
active_ASID : a_1	active_ASID :
incon_set : { va_3 , va_7 }	incon_set : { , }
pt_sanpshot: a_1, va_3 => unmap a_1, va_7 => pa_1 a_1, va_6 => unmap a_2, va_1 => Incon a_2, va_6 => pa_2	pt_snapshot:
page table: va_6 => pa_7	

Program Logic — Rules

- updateASID

$\{\lambda s.\ \text{kernel } s \wedge P\ (\text{asid\_pt\_snpshot\_update}_s\ a)\}$ updateASID a $\{P\}$

Steps: switching from a_1 to a_2

before updateASID

active_ASID : a_1

incon_set : { va_3 , va_7 }

pt_sanpshot:
a_1, va_3 => unmap
a_1, va_7 => pa_1
a_1, va_6 => unmap
a_2, va_1 => Incon
a_2, va_6 => pa_2

page table: va_6 => pa_7

after updateASID

active_ASID :

incon_set : { , }

pt_snapshot:
a_1, va_3 => Incon
a_1, va_7 => Incon
a_1, va_6 => pa_7

1

Program Logic — Rules

- updateASID

$\{\lambda s.\ \texttt{kernel s} \wedge \texttt{P (asid\_pt\_snpshot\_update}_s\ \texttt{a)}\}$ `updateASID a` $\{P\}$

Steps: switching from a_1 to a_2

before updateASID

active_ASID : a_1

incon_set : { va_3 , va_7 }

pt_sanpshot:
a_1, va_3 => unmap
a_1, va_7 => pa_1
a_1, va_6 => unmap
a_2, va_1 => Incon
a_2, va_6 => pa_2

page table: va_6 => pa_7

after updateASID

active_ASID : a_2 **2**

incon_set : { , }

pt_snapshot:
a_1, va_3 => Incon
a_1, va_7 => Incon
a_1, va_6 => pa_7

Program Logic — Rules

- updateASID

$\{\lambda s.\ \text{kernel } s \wedge P\ (\text{asid\_pt\_snpshot\_update}_s\ a)\}$ updateASID a $\{P\}$

Steps: switching from a_1 to a_2

before updateASID

active_ASID : a_1

incon_set : { va_3 , va_7 }

pt_sanpshot:
a_1, va_3 => unmap
a_1, va_7 => pa_1
a_1, va_6 => unmap
a_2, va_1 => Incon
a_2, va_6 => pa_2

page table: va_6 => pa_7

after updateASID

active_ASID : a_2

incon_set : { va_1 , va_6 } **3**

pt_snapshot:
a_1, va_3 => Incon
a_1, va_7 => Incon
a_1, va_6 => pa_7

Program Logic — Rules

- assignment ✓
- updateRoot ✓
- updateASID ✓

Take-away:

TLB has been reduced to consistency check

Inconsistency is recomputed after every instruction

Contributions

- TLB-aware program logic in Isabelle/HOL
  - sound abstraction of ARMv7-style MMU
  - language with TLB management primitives
  - TLB-aware Hoare logic rules

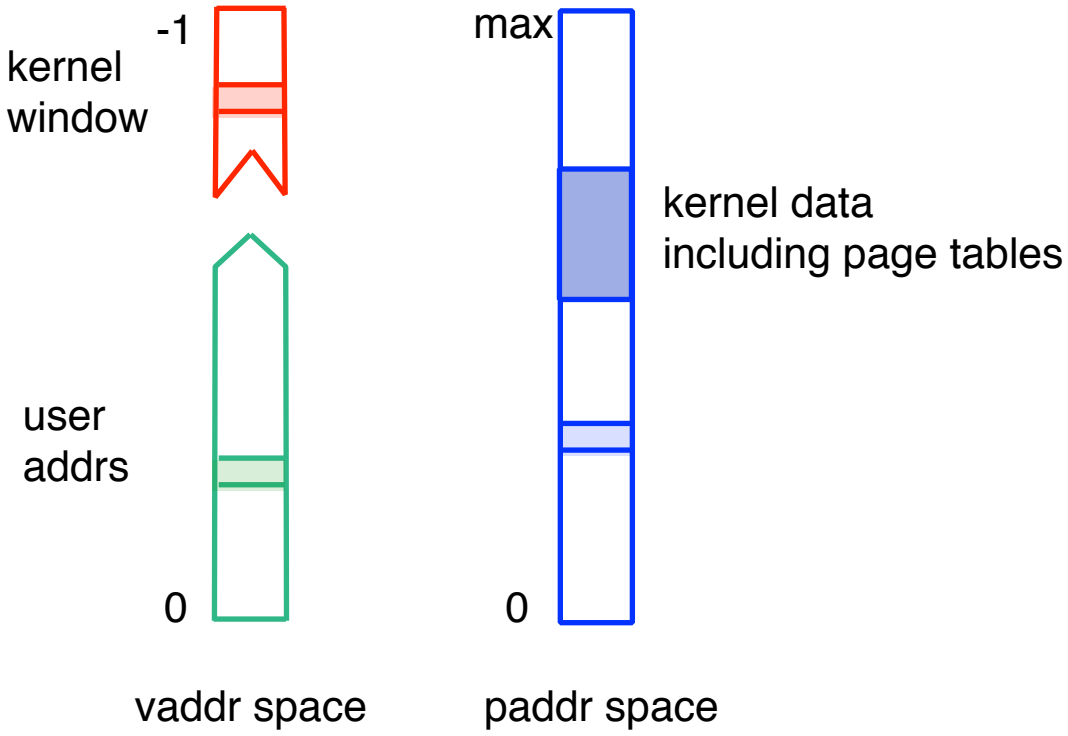
- Reduction theorems for program verification at
  - user- and kernel-level execution
  - context switch

Program Verification

- Address space management
  - inspired by seL4

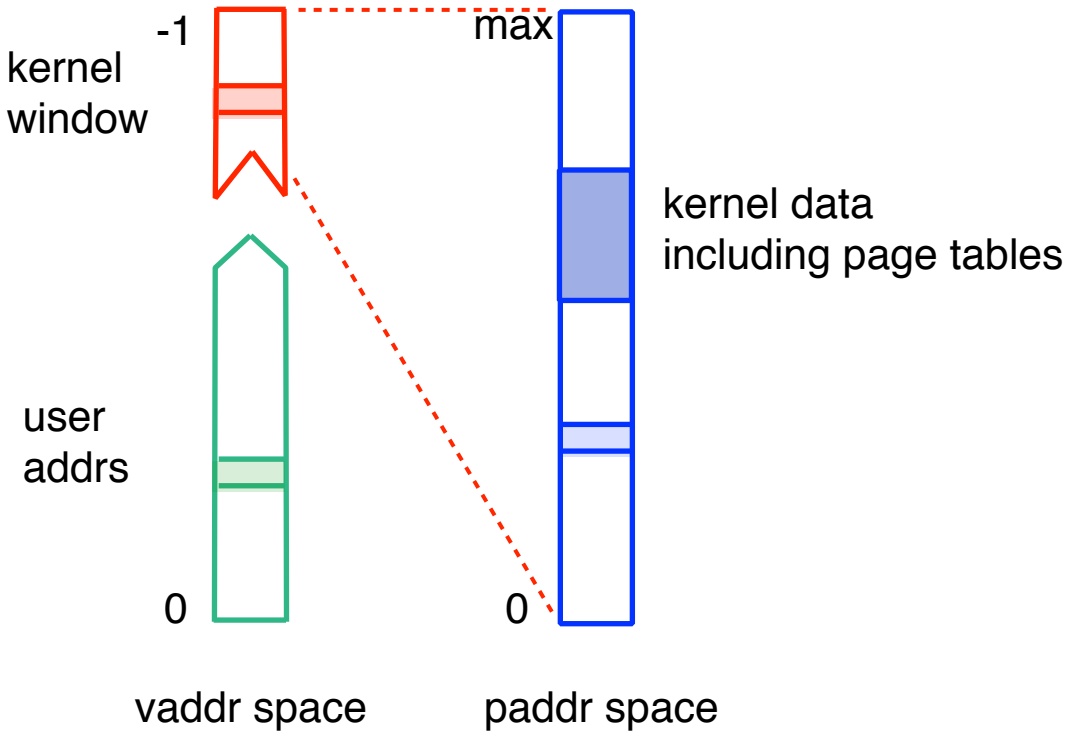
Program Verification

- Address space management
  - inspired by seL4



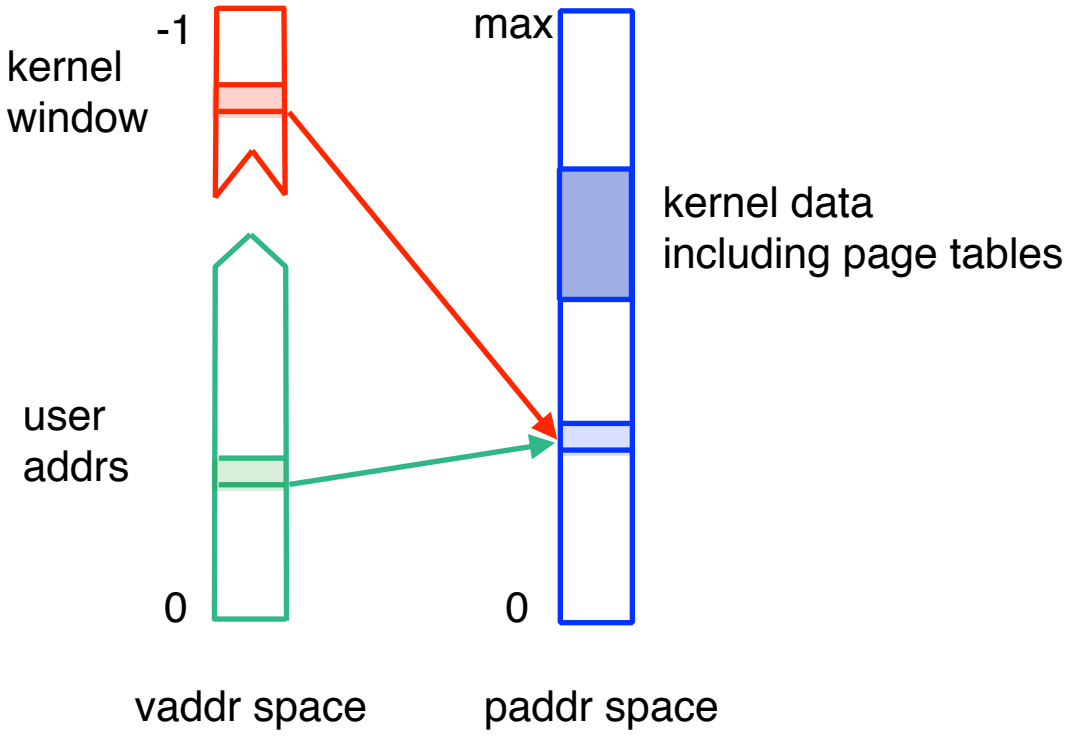
Program Verification

- Address space management
  - inspired by seL4



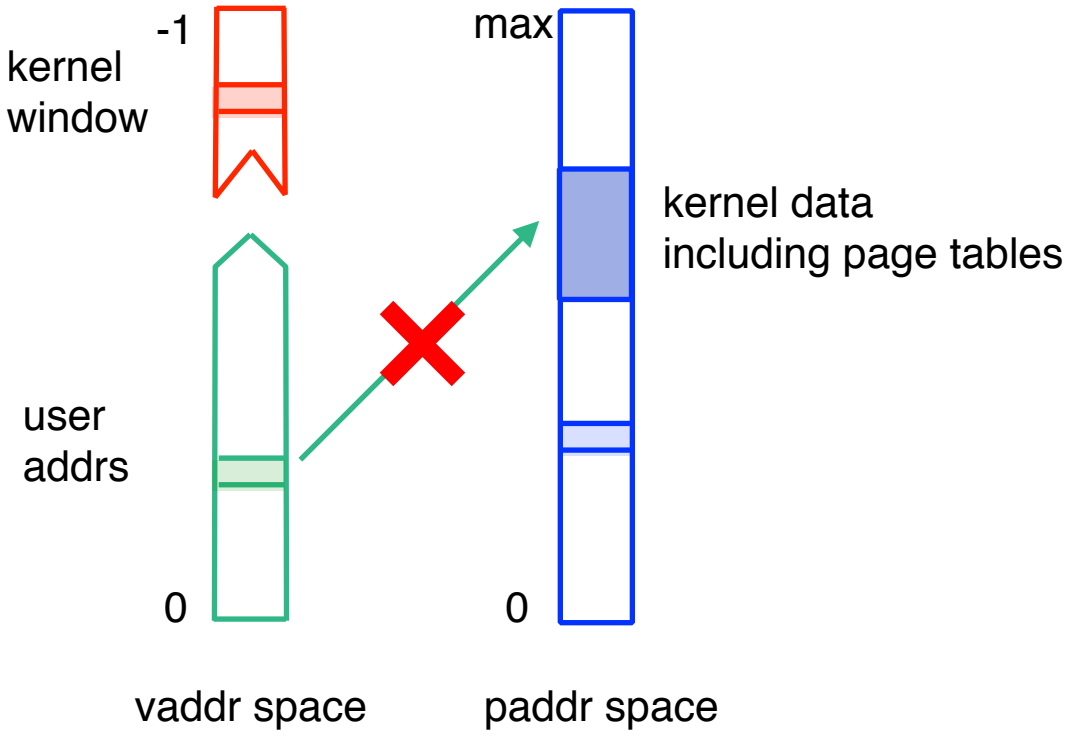
Program Verification

- Address space management
  - inspired by seL4



Program Verification

- Address space management
  - inspired by seL4



Reduction Theorems

- User-level assignment
  - user cannot update page table,
    hence cannot affect TLB consistency

$$\{\!|\lambda s.\ \texttt{mmu\_layout s} \wedge \texttt{mode s} = \texttt{User} \wedge \mathcal{IC}\ \texttt{s} = \emptyset \wedge [\![\texttt{lval}]\!]\ \texttt{s} = \lfloor \texttt{vp} \rfloor \wedge [\![\texttt{rval}]\!]\ \texttt{s} = \lfloor \texttt{v} \rfloor \wedge \texttt{Addr vp} \hookrightarrow_s \texttt{p} |\!\}$$
$$\texttt{lval} ::= \texttt{rval}$$
$$\{\!|\lambda s.\ \texttt{mmu\_layout s} \wedge \texttt{mode s} = \texttt{User} \wedge \mathcal{IC}\ \texttt{s} = \emptyset \wedge \texttt{heap s p} = \lfloor \texttt{v} \rfloor |\!\}$$

Reduction Theorems

- User-level assignment
  - user cannot update page table,
    hence cannot affect TLB consistency

$$\{\!|\lambda s.\ \texttt{mmu\_layout s} \wedge \texttt{mode s = User} \wedge \mathcal{IC}\ \texttt{s} = \emptyset \wedge [\![\texttt{lval}]\!]\ \texttt{s} = \lfloor \texttt{vp} \rfloor \wedge [\![\texttt{rval}]\!]\ \texttt{s} = \lfloor \texttt{v} \rfloor \wedge \texttt{Addr vp} \hookrightarrow_s \texttt{p} |\!\}$$
$$\texttt{lval ::= rval}$$
$$\{\!|\lambda s.\ \texttt{mmu\_layout s} \wedge \texttt{mode s = User} \wedge \mathcal{IC}\ \texttt{s} = \emptyset \wedge \texttt{heap s p} = \lfloor \texttt{v} \rfloor |\!\}$$

Reduction Theorems

- User-level assignment
  - user cannot update page table,
    hence cannot affect TLB consistency

$$\{\!|\lambda s.\ \texttt{mmu\_layout s} \wedge \texttt{mode s} = \texttt{User} \wedge \mathcal{IC}\ \texttt{s} = \emptyset \wedge [\![\texttt{lval}]\!]\ \texttt{s} = \lfloor \texttt{vp} \rfloor \wedge [\![\texttt{rval}]\!]\ \texttt{s} = \lfloor \texttt{v} \rfloor \wedge \texttt{Addr vp} \hookrightarrow_s \texttt{p} |\!\}$$

$$\texttt{lval ::= rval}$$

$$\{\!|\lambda s.\ \texttt{mmu\_layout s} \wedge \texttt{mode s} = \texttt{User} \wedge \mathcal{IC}\ \texttt{s} = \emptyset \wedge \texttt{heap s p} = \lfloor \texttt{v} \rfloor |\!\}$$

Reduction Theorems

- User-level assignment
  - user cannot update page table,
    hence cannot affect TLB consistency

$$\{\!|\lambda s.\ \texttt{mmu\_layout s} \wedge \texttt{mode s} = \texttt{User} \wedge \mathcal{IC}\ \texttt{s} = \emptyset \wedge [\![\texttt{lval}]\!]\ \texttt{s} = \lfloor \texttt{vp} \rfloor \wedge [\![\texttt{rval}]\!]\ \texttt{s} = \lfloor \texttt{v} \rfloor \wedge \texttt{Addr vp} \hookrightarrow_s \texttt{p} |\!\}$$

$$\texttt{lval ::= rval}$$

$$\{\!|\lambda s.\ \texttt{mmu\_layout s} \wedge \texttt{mode s} = \texttt{User} \wedge \mathcal{IC}\ \texttt{s} = \emptyset \wedge \texttt{heap s p} = \lfloor \texttt{v} \rfloor |\!\}$$

Reduction Theorems

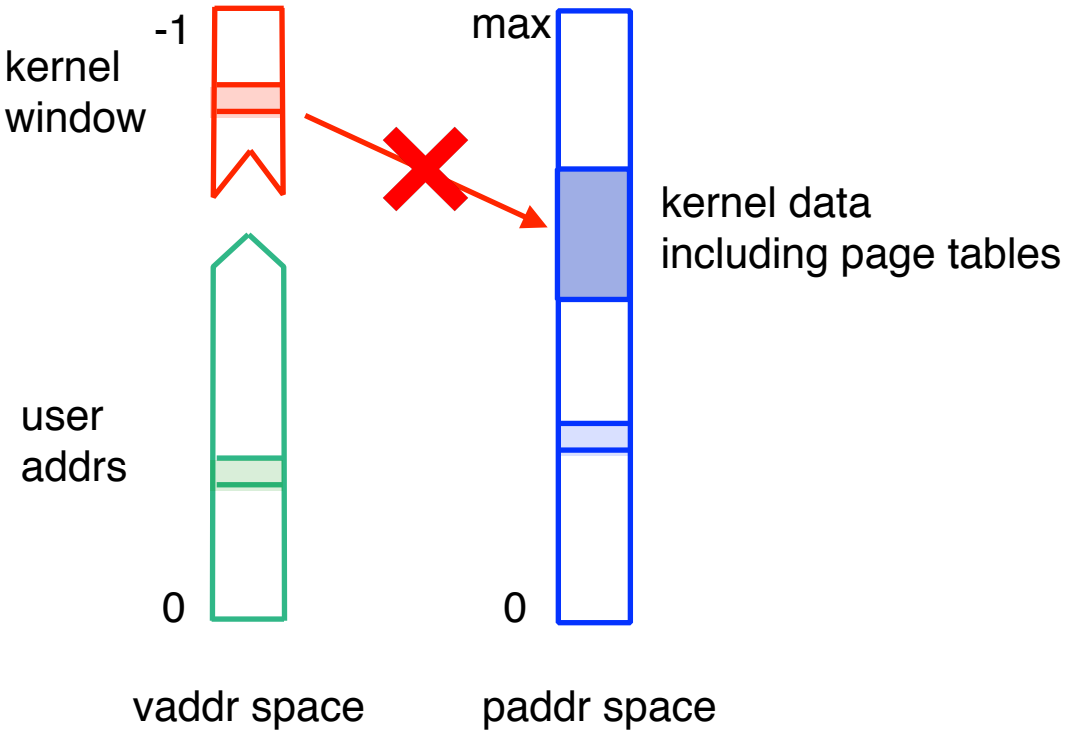
- User-level assignment
  - user cannot update page table,
    hence cannot affect TLB consistency

$$\{\!|\lambda s.\ \texttt{mmu\_layout}\ s \wedge \texttt{mode}\ s = \texttt{User} \wedge \mathcal{IC}\ s = \emptyset \wedge [\![\texttt{lval}]\!]\ s = \lfloor \texttt{vp} \rfloor \wedge [\![\texttt{rval}]\!]\ s = \lfloor \texttt{v} \rfloor \wedge \texttt{Addr vp} \hookrightarrow_s \texttt{p} |\!\}$$
$$\texttt{lval} ::= \texttt{rval}$$
$$\{\!|\lambda s.\ \texttt{mmu\_layout}\ s \wedge \texttt{mode}\ s = \texttt{User} \wedge \mathcal{IC}\ s = \emptyset \wedge \texttt{heap}\ s\ \texttt{p} = \lfloor \texttt{v} \rfloor |\!\}$$

user-level reasoning has been reduced to
standard Hoare logic rule with address translation

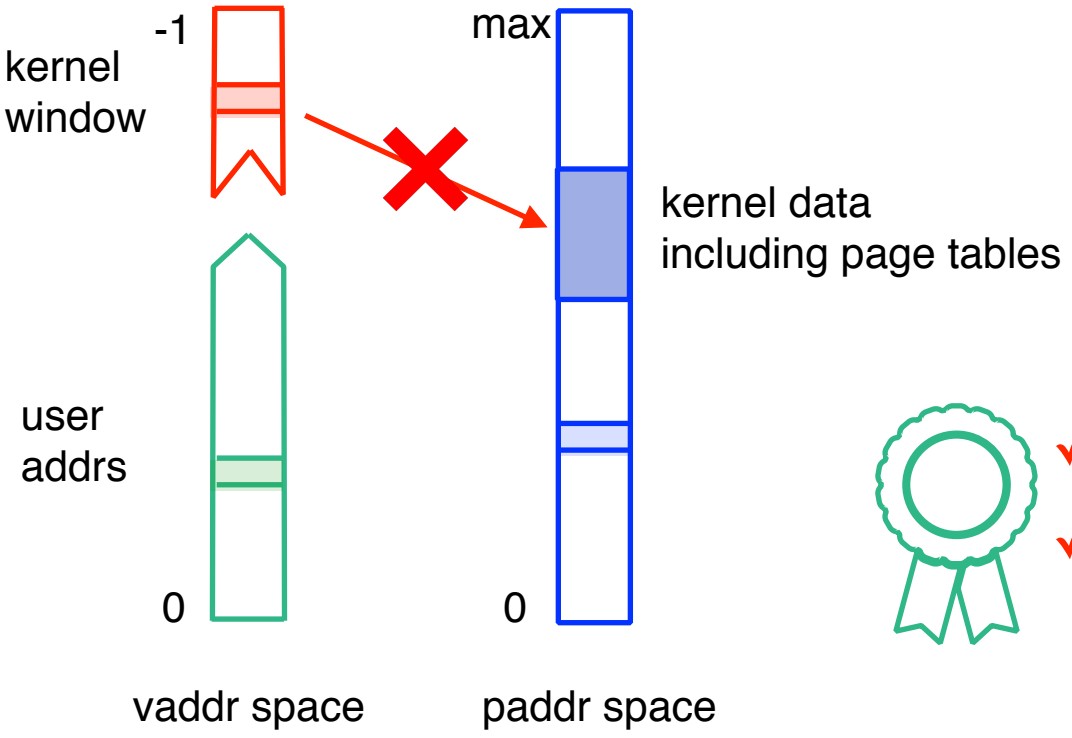
Reduction Theorems

- Kernel-level assignment
  - that doesn't modify page table



Reduction Theorems

- Kernel-level assignment
  - that doesn't modify page table

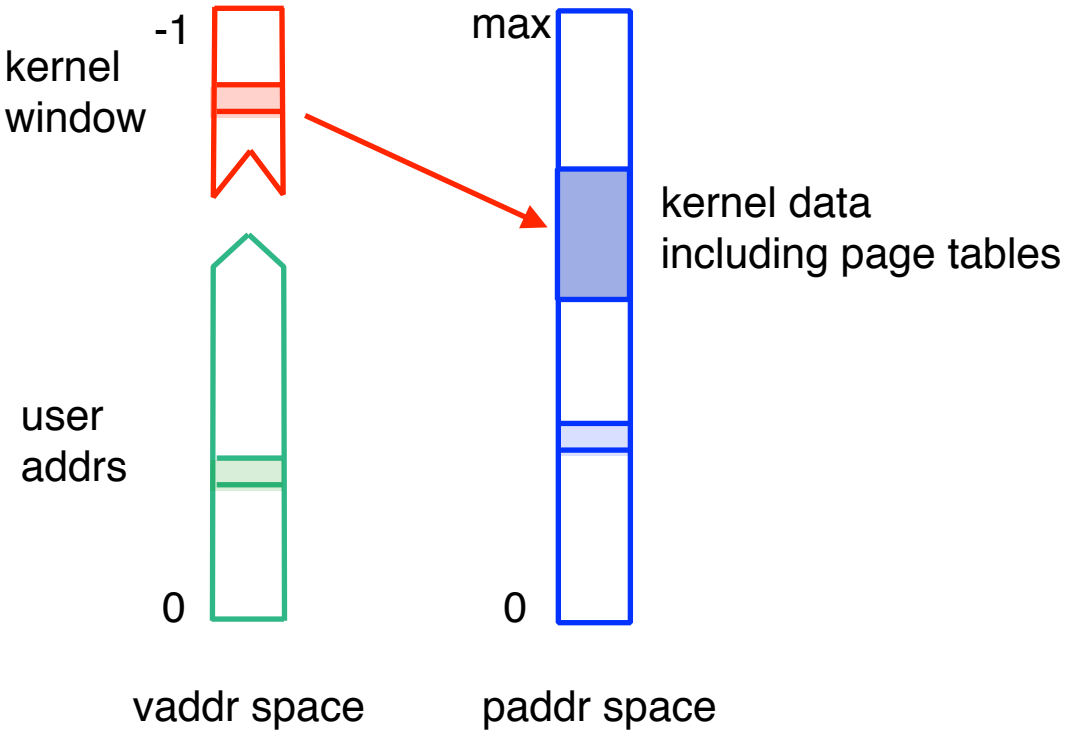


✓TLB remains consistent

✓reduction to
standard Hoare logic

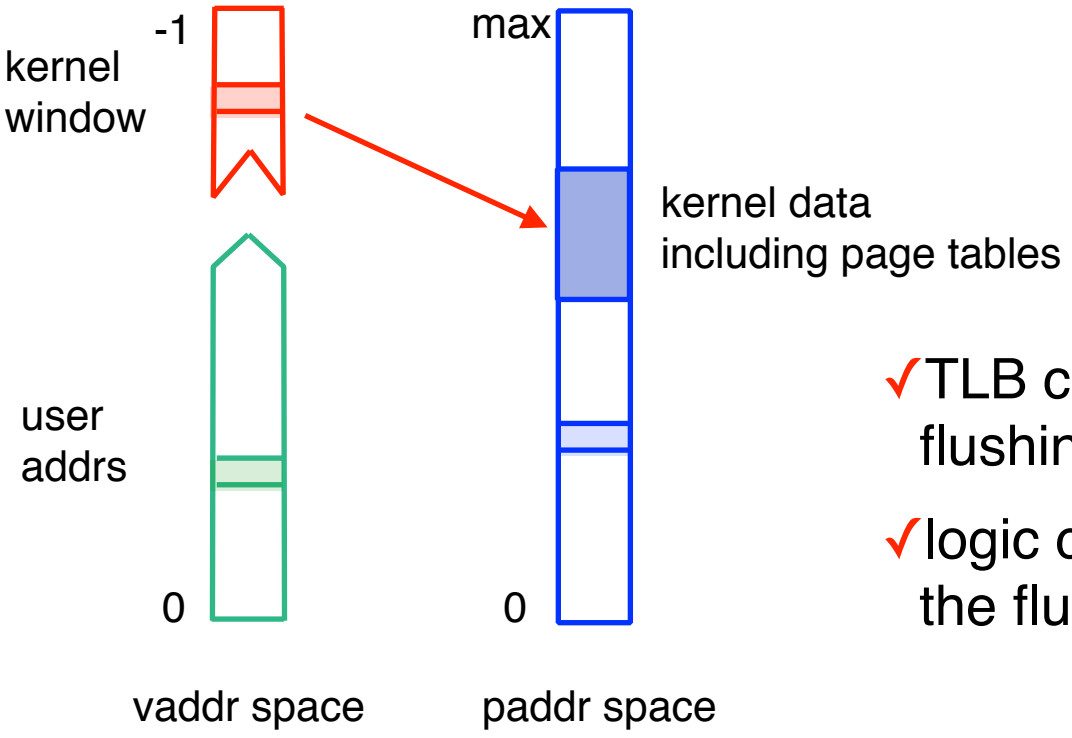
Reduction Theorems

- Kernel-level assignment
  - that does modify page table



Reduction Theorems

- Kernel-level assignment
  - that does modify page table



✓TLB consistency is regained by flushing the entries

✓logic correctly identifies when the flush is due

Contributions

- TLB-aware program logic in Isabelle/HOL
  - sound abstraction of ARMv7-style MMU
  - language with TLB management primitives
  - TLB-aware Hoare logic rules

- Reduction theorems for program verification at
  - user- and kernel-level execution ✓
  - context switch

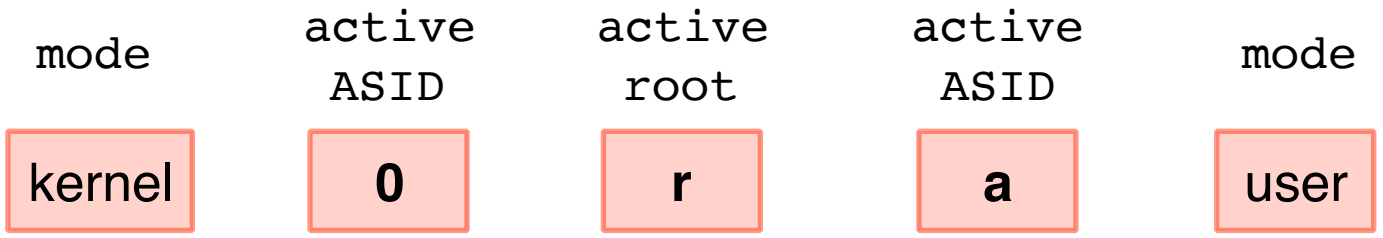
Context Switch

- Operations — updating the
  - root register, then updating the
  - ASID register

Context Switch

- Operations — updating the
  - root register, then updating the
  - ASID register

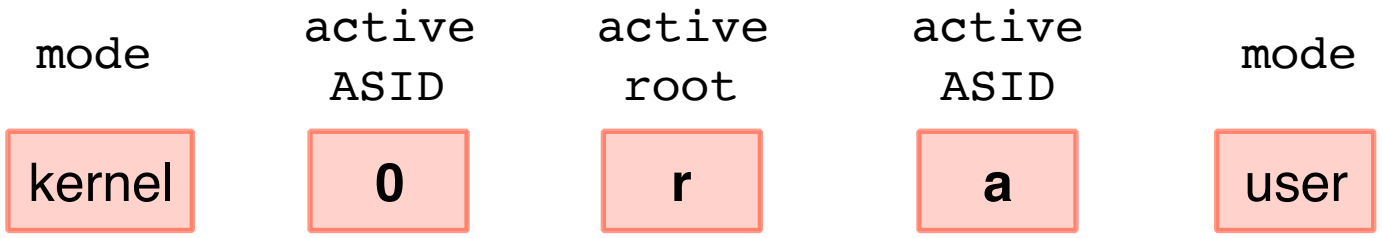
- ARM's recommended sequence to avoid TLB flush



Context Switch

- Operations — updating the
  - root register, then updating the
  - ASID register

- ARM's recommended sequence to avoid TLB flush



logic can prove that
assigned ASIDs remain consistent

Taken Together

Taken Together

- simplicity of the logic and memory model
- reduction to Hoare logic for most use-cases

Taken Together

- simplicity of the logic and memory model
- reduction to Hoare logic for most use-cases

more in the paper:

details of the reduction theorems

Taken Together

- simplicity of the logic and memory model
- reduction to Hoare logic for most use-cases

more in the paper:

details of the reduction theorems

theories available on github:

`SEL4PROJ/tlb`

Taken Together

- simplicity of the logic and memory model
- reduction to Hoare logic for most use-cases

more in the paper:

details of the reduction theorems

theories available on github:

`SEL4PROJ/tlb`

Questions

Reduction Theorems

- Kernel-level assignment
  - that doesn't modify page table

Reduction Theorems

- Kernel-level assignment
  - that doesn't modify page table

$$\{\!|\lambda s.\ \texttt{mmu\_layout s} \wedge \texttt{mode s} = \texttt{Kernel} \wedge \texttt{asids\_consistent s} \wedge$$
$$[\![\texttt{lval}]\!]\ \texttt{s} = \lfloor \texttt{vp} \rfloor \wedge [\![\texttt{rval}]\!]\ \texttt{s} = \lfloor \texttt{v} \rfloor \wedge$$
$$\texttt{Addr vp} \in \texttt{kernel\_safe s} \wedge \texttt{k\_phy\_ad vp} \notin \texttt{kernel\_data\_area s} \wedge$$
$$\texttt{safe\_set (kernel\_safe s) s}\ |\!\}$$

```
lval ::= rval
```

Reduction Theorems

- Kernel-level assignment
  - that doesn't modify page table

$$\{\!|\lambda s.\ \texttt{mmu\_layout s} \wedge \texttt{mode s} = \texttt{Kernel} \wedge \texttt{asids\_consistent s} \wedge$$
$$[\![\texttt{lval}]\!]\ \texttt{s} = \lfloor \texttt{vp} \rfloor \wedge [\![\texttt{rval}]\!]\ \texttt{s} = \lfloor \texttt{v} \rfloor \wedge$$
$$\texttt{Addr vp} \in \texttt{kernel\_safe s} \wedge \texttt{k\_phy\_ad vp} \notin \texttt{kernel\_data\_area s} \wedge$$
$$\texttt{safe\_set (kernel\_safe s) s}\ |\!\}$$

$$\texttt{lval} ::= \texttt{rval}$$

$$\{\!|\lambda s.\ \texttt{mmu\_layout s} \wedge \texttt{mode s} = \texttt{Kernel} \wedge \texttt{safe\_set (kernel\_safe s) s} \wedge$$
$$\texttt{asids\_consistent s} \wedge \texttt{heap s (k\_phy\_ad vp)} = \lfloor \texttt{v} \rfloor |\!\}$$

Reduction Theorems

- Kernel-level assignment
  - that doesn't modify page table

$$\{\lambda s.\ \texttt{mmu\_layout s} \wedge \texttt{mode s} = \texttt{Kernel} \wedge \texttt{asids\_consistent s} \wedge$$
$$[\![\texttt{lval}]\!]\ \texttt{s} = \lfloor \texttt{vp} \rfloor \wedge [\![\texttt{rval}]\!]\ \texttt{s} = \lfloor \texttt{v} \rfloor \wedge$$
$$\texttt{Addr vp} \in \texttt{kernel\_safe s} \wedge \texttt{k\_phy\_ad vp} \notin \texttt{kernel\_data\_area s} \wedge$$
$$\texttt{safe\_set (kernel\_safe s) s}\}$$

$$\texttt{lval} ::= \texttt{rval}$$

$$\{\lambda s.\ \texttt{mmu\_layout s} \wedge \texttt{mode s} = \texttt{Kernel} \wedge \texttt{safe\_set (kernel\_safe s) s} \wedge$$
$$\texttt{asids\_consistent s} \wedge \texttt{heap s (k\_phy\_ad vp)} = \lfloor \texttt{v} \rfloor\}$$

Reduction Theorems

- Kernel-level assignment
  - that doesn't modify page table

$$\{\lambda s.\ \texttt{mmu\_layout s} \wedge \texttt{mode s} = \texttt{Kernel} \wedge \texttt{asids\_consistent s} \wedge [\![\texttt{lval}]\!]\ \texttt{s} = \lfloor \texttt{vp} \rfloor \wedge [\![\texttt{rval}]\!]\ \texttt{s} = \lfloor \texttt{v} \rfloor \wedge \texttt{Addr vp} \in \texttt{kernel\_safe s} \wedge \texttt{k\_phy\_ad vp} \notin \texttt{kernel\_data\_area s} \wedge \texttt{safe\_set (kernel\_safe s) s}\}$$

$$\texttt{lval} ::= \texttt{rval}$$

$$\{\lambda s.\ \texttt{mmu\_layout s} \wedge \texttt{mode s} = \texttt{Kernel} \wedge \texttt{safe\_set (kernel\_safe s) s} \wedge \texttt{asids\_consistent s} \wedge \texttt{heap s (k\_phy\_ad vp)} = \lfloor \texttt{v} \rfloor\}$$

Reduction Theorems

- Kernel-level assignment
  - that doesn't modify page table

$$\{\lambda s.\ \texttt{mmu\_layout s} \wedge \texttt{mode s} = \texttt{Kernel} \wedge \texttt{asids\_consistent s} \wedge [\![\texttt{lval}]\!]\ \texttt{s} = \lfloor \texttt{vp} \rfloor \wedge [\![\texttt{rval}]\!]\ \texttt{s} = \lfloor \texttt{v} \rfloor \wedge \texttt{Addr vp} \in \texttt{kernel\_safe s} \wedge \texttt{k\_phy\_ad vp} \notin \texttt{kernel\_data\_area s} \wedge \texttt{safe\_set (kernel\_safe s) s}\}$$

$$\texttt{lval} ::= \texttt{rval}$$

$$\{\lambda s.\ \texttt{mmu\_layout s} \wedge \texttt{mode s} = \texttt{Kernel} \wedge \texttt{safe\_set (kernel\_safe s) s} \wedge \texttt{asids\_consistent s} \wedge \texttt{heap s (k\_phy\_ad vp)} = \lfloor \texttt{v} \rfloor\}$$

Context Switch

- Operations —
  - update root register to `root r`, then
  - update ASID register to `ASID a`

Context Switch

- Operations —
  - update root register to `root r`, then
  - update ASID register to `ASID a`

- ARMv7-A manual's recommended sequence

$$\{\!|\lambda s.\ \texttt{mmu\_layout s} \wedge \texttt{asids\_consistent s} \wedge \texttt{mode s} = \texttt{Kernel} \wedge \mathcal{IC}\ \texttt{s} = \emptyset \wedge \texttt{0} \notin \texttt{ran (root\_map s)} \wedge \texttt{root\_map s (Addr r)} = \lfloor \texttt{a} \rfloor |\!\}$$
$$\texttt{UpdateASID 0;; updateRoot(Const r);; UpdateASID a;; SetMode User}$$
$$\{\!|\lambda s.\ \texttt{mmu\_layout s} \wedge \mathcal{IC}\ \texttt{s} = \emptyset \wedge \texttt{mode s} = \texttt{User} \wedge \texttt{asids\_consistent s} |\!\}$$

Context Switch

- Operations —
  - update root register to `root r`, then
  - update ASID register to `ASID a`

- ARMv7-A manual's recommended sequence

$$\{\!|\lambda s.\ \texttt{mmu\_layout s} \wedge \texttt{asids\_consistent s} \wedge \texttt{mode s = Kernel} \wedge$$
$$\mathcal{IC}\ \texttt{s} = \emptyset \wedge \texttt{0} \notin \texttt{ran (root\_map s)} \wedge \texttt{root\_map s (Addr r)} = \lfloor \texttt{a} \rfloor |\!\}$$
$$\texttt{UpdateASID 0;; updateRoot(Const r);; UpdateASID a;; SetMode User}$$
$$\{\!|\lambda s.\ \texttt{mmu\_layout s} \wedge \mathcal{IC}\ \texttt{s} = \emptyset \wedge \texttt{mode s = User} \wedge \texttt{asids\_consistent s} |\!\}$$

Context Switch

- Operations —
  - update root register to `root r`, then
  - update ASID register to `ASID a`

- ARMv7-A manual's recommended sequence

$$\{\!|\lambda s.\ \texttt{mmu\_layout s} \wedge \texttt{asids\_consistent s} \wedge \texttt{mode s = Kernel} \wedge$$
$$\mathcal{IC}\ \texttt{s} = \emptyset \wedge \texttt{0} \notin \texttt{ran (root\_map s)} \wedge \texttt{root\_map s (Addr r)} = \lfloor \texttt{a} \rfloor |\!\}$$
$$\texttt{UpdateASID 0;; updateRoot(Const r);; UpdateASID a;; SetMode User}$$
$$\{\!|\lambda s.\ \texttt{mmu\_layout s} \wedge \mathcal{IC}\ \texttt{s} = \emptyset \wedge \texttt{mode s = User} \wedge \texttt{asids\_consistent s} |\!\}$$

Context Switch

- Operations —
  - update root register to `root r`, then
  - update ASID register to `ASID a`

- ARMv7-A manual's recommended sequence

$$\{\!|\lambda s.\ \texttt{mmu\_layout s} \wedge \texttt{asids\_consistent s} \wedge \texttt{mode s = Kernel} \wedge \mathcal{IC}\ \texttt{s} = \emptyset \wedge \texttt{0} \notin \texttt{ran (root\_map s)} \wedge \texttt{root\_map s (Addr r)} = \lfloor \texttt{a} \rfloor |\!\}$$
$$\texttt{UpdateASID 0;; updateRoot(Const r);; UpdateASID a;; SetMode User}$$
$$\{\!|\lambda s.\ \texttt{mmu\_layout s} \wedge \mathcal{IC}\ \texttt{s} = \emptyset \wedge \texttt{mode s = User} \wedge \texttt{asids\_consistent s} |\!\}$$

Context Switch

- Operations —
  - update root register to `root r`, then
  - update ASID register to `ASID a`

- ARMv7-A manual's recommended sequence

$$\{\!|\lambda s.\ \texttt{mmu\_layout s} \wedge \texttt{asids\_consistent s} \wedge \texttt{mode s = Kernel} \wedge \mathcal{IC}\ \texttt{s} = \emptyset \wedge \texttt{0} \notin \texttt{ran (root\_map s)} \wedge \texttt{root\_map s (Addr r)} = \lfloor \texttt{a} \rfloor |\!\}$$
$$\texttt{UpdateASID 0;; updateRoot(Const r);; UpdateASID a;; SetMode User}$$
$$\{\!|\lambda s.\ \texttt{mmu\_layout s} \wedge \mathcal{IC}\ \texttt{s} = \emptyset \wedge \texttt{mode s = User} \wedge \texttt{asids\_consistent s} |\!\}$$

Context Switch

- Operations —
  - update root register to `root r`, then
  - update ASID register to `ASID a`

- ARMv7-A manual's recommended sequence

$$\{\!|\lambda s.\ \texttt{mmu\_layout s} \wedge \texttt{asids\_consistent s} \wedge \texttt{mode s = Kernel} \wedge$$
$$\mathcal{IC}\ \texttt{s} = \emptyset \wedge \texttt{0} \notin \texttt{ran (root\_map s)} \wedge \texttt{root\_map s (Addr r)} = \lfloor \texttt{a} \rfloor |\!\}$$
$$\texttt{UpdateASID 0;; updateRoot(Const r);; UpdateASID a;; SetMode User}$$
$$\{\!|\lambda s.\ \texttt{mmu\_layout s} \wedge \mathcal{IC}\ \texttt{s} = \emptyset \wedge \texttt{mode s = User} \wedge \texttt{asids\_consistent s} |\!\}$$

Context Switch

- Operations —
  - update root register to `root r`, then
  - update ASID register to `ASID a`

- ARMv7-A manual's recommended sequence

$$\{\!|\lambda s.\ \texttt{mmu\_layout s} \wedge \texttt{asids\_consistent s} \wedge \texttt{mode s = Kernel} \wedge \mathcal{IC}\ \texttt{s} = \emptyset \wedge \texttt{0} \notin \texttt{ran (root\_map s)} \wedge \texttt{root\_map s (Addr r)} = \lfloor \texttt{a} \rfloor |\!\}$$
$$\texttt{UpdateASID 0;; updateRoot(Const r);; UpdateASID a;; SetMode User}$$
$$\{\!|\lambda s.\ \texttt{mmu\_layout s} \wedge \mathcal{IC}\ \texttt{s} = \emptyset \wedge \texttt{mode s = User} \wedge \texttt{asids\_consistent s} |\!\}$$

Context Switch

- Operations —
  - update root register to `root r`, then
  - update ASID register to `ASID a`

- ARMv7-A manual's recommended sequence

$$\{\lambda s.\ \texttt{mmu\_layout s} \wedge \texttt{asids\_consistent s} \wedge \texttt{mode s = Kernel} \wedge \mathcal{IC}\ \texttt{s} = \emptyset \wedge \texttt{0} \notin \texttt{ran (root\_map s)} \wedge \texttt{root\_map s (Addr r)} = \lfloor \texttt{a} \rfloor\}$$
$$\texttt{UpdateASID 0;; updateRoot(Const r);; UpdateASID a;; SetMode User}$$
$$\{\lambda s.\ \texttt{mmu\_layout s} \wedge \mathcal{IC}\ \texttt{s} = \emptyset \wedge \texttt{mode s = User} \wedge \texttt{asids\_consistent s}\}$$

Context Switch

- Operations —
  - update root register to `root r`, then
  - update ASID register to `ASID a`

- ARMv7-A manual's recommended sequence

$$\{\lambda s.\ \texttt{mmu\_layout s} \wedge \texttt{asids\_consistent s} \wedge \texttt{mode s = Kernel} \wedge \mathcal{IC}\ \texttt{s} = \emptyset \wedge \texttt{0} \notin \texttt{ran (root\_map s)} \wedge \texttt{root\_map s (Addr r)} = \lfloor \texttt{a} \rfloor\}$$
$$\texttt{UpdateASID 0;; updateRoot(Const r);; UpdateASID a;; SetMode User}$$
$$\{\lambda s.\ \texttt{mmu\_layout s} \wedge \mathcal{IC}\ \texttt{s} = \emptyset \wedge \texttt{mode s = User} \wedge \texttt{asids\_consistent s}\}$$