Formal verification of low-level execution platforms

Roberto Guanciale

Estonian Winter School
Day 03

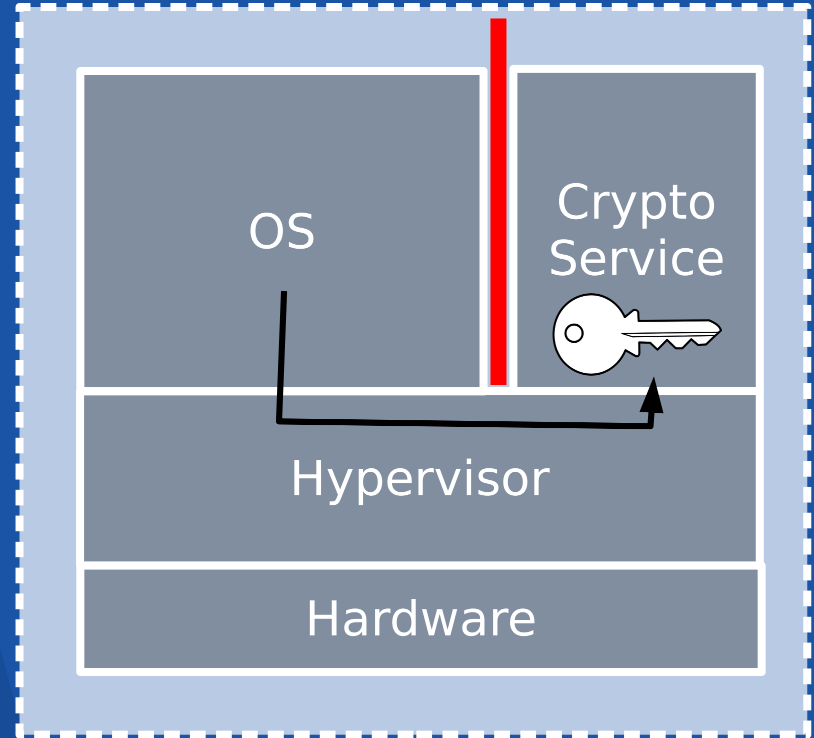
Cooperative scheduling

Static memory allocation

Message passing

Paravirtualization

No preemption



Cooperative scheduling

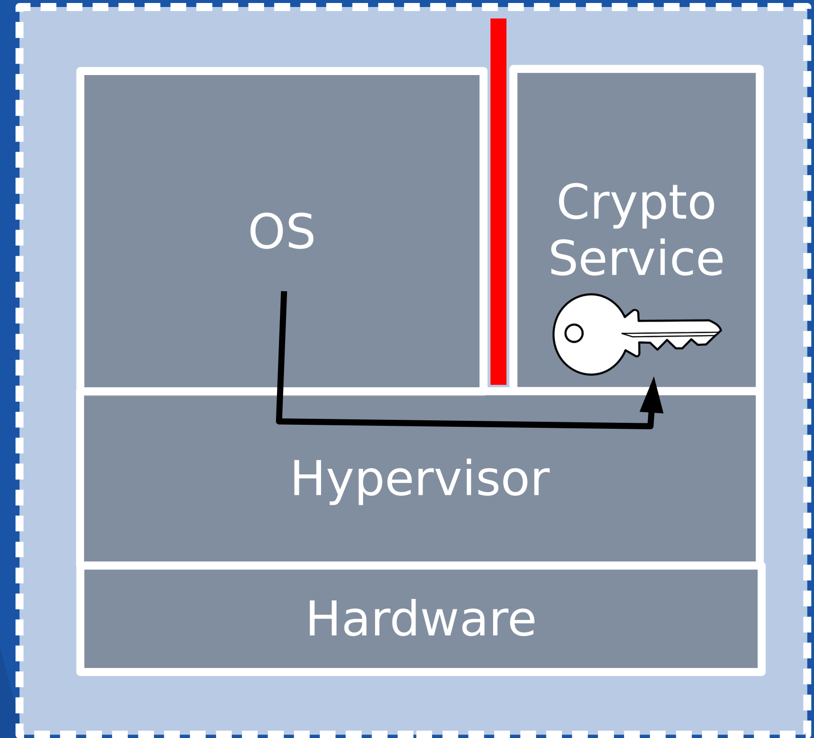
Static memory allocation

Message passing

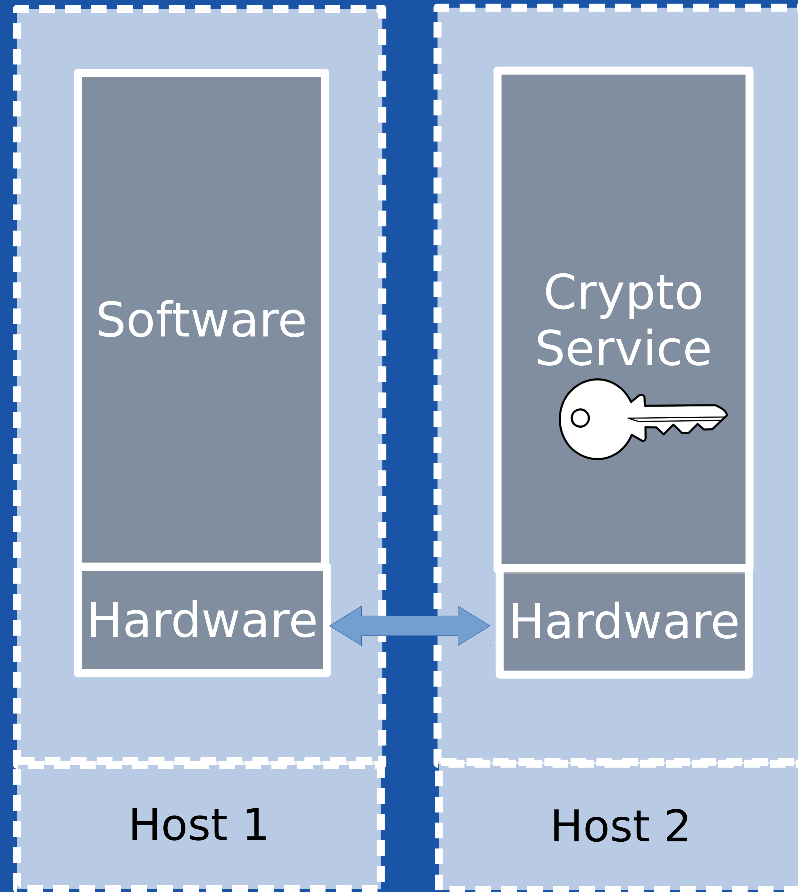
Paravirtualization

No preemption

- executed in PL1
- invoked via SW interrupt
- supervises page tables



Top Level Specification



Top level specification (ideal world)

- Two physically machines (one active)

  and hypervisor data

$$\delta = \langle \sigma_1, \sigma_2, h \rangle$$

Top level specification (ideal world)

- Two physically machines (one active)

  and hypervisor data

$$\delta = \langle \sigma_1, \sigma_2, h \rangle$$

- Standard computations

  have standard effects

$$\frac{h.a = 1 \wedge Mode(\sigma_1) = 0 \wedge \sigma_1 \longrightarrow \sigma_1'}{\langle \sigma_1, \sigma_2, h \rangle \longrightarrow_0 \langle \sigma_1', \sigma_2, h \rangle}$$

Top level specification (ideal world)

- Two physically machines (one active)

  and hypervisor data

$$\delta = \langle \sigma_1, \sigma_2, h \rangle$$

- Standard computations

  have standard effects

$$\frac{h.a = 1 \wedge Mode(\sigma_1) = 0 \wedge \sigma_1 \longrightarrow \sigma_1'}{\langle \sigma_1, \sigma_2, h \rangle \longrightarrow_0 \langle \sigma_1', \sigma_2, h \rangle}$$

- Exceptions activate the

  ideal functionalities

$$\frac{h.a = 1 \wedge Mode(\sigma_1) = 1}{\langle \sigma_1, \sigma_2, h, \rangle \longrightarrow_1 H(\sigma_1, \sigma_2, h)}$$

What H does?

- Context switch

$$\langle \sigma_1, \sigma_2, h \rangle \longrightarrow \langle \sigma_1, \sigma_2, h \text{ with } a = \neg h.a \rangle$$

What H does?

- Context switch

- Communication

$$\langle \sigma_1, \sigma_2, h \rangle \longrightarrow \langle \sigma_1, \sigma_2, h \text{ with } m_2 = \sigma_1.mem(out) \rangle$$

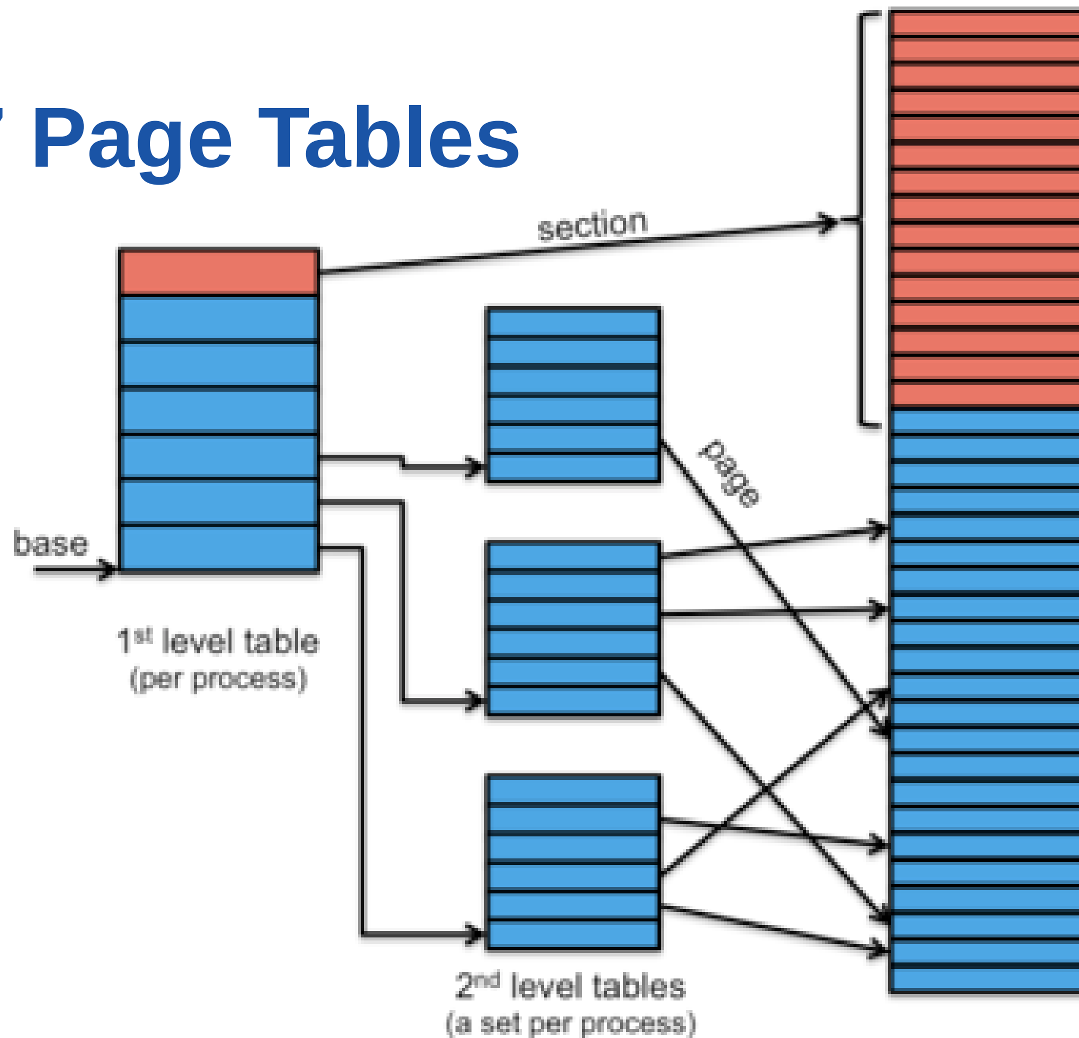
$$\langle \sigma_1, \sigma_2, h \rangle \longrightarrow \left\langle \begin{array}{l} \sigma_1 \text{ with } mem(in) = h.m_1, \\ \sigma_2, \\ h \text{ with } m_1 = \emptyset \end{array} \right\rangle$$

What H does?

- Context switch
- Communication
- Manages page tables

$$\frac{\sigma_1', d_1' = DMMU(\sigma_1, h.d_1)}{\langle \sigma_1, \sigma_2, h \rangle \longrightarrow \langle \sigma_1', \sigma_2, h \text{ with } d_1 = d_1' \rangle}$$

ARMv7 Page Tables



ARMv7 page tables

- Usually 1 L1 page table per process
- L1
  - Splits 4GB into pages of 1 MB
  - 4096 descriptors per table (16 KB)
  - 1 MB of virtual memory
    - mapped to 1 MB of physical memory (section)
    - partitioned by a L2 page table
    - unmapped

ARMv7 page tables

- Usually 1 L1 page table per process
- L1
  - Splits 4GB into pages of 1 MB
  - 4096 descriptors per table (16 KB)
  - 1 MB of virtual memory
    - mapped to 1 MB of physical memory (section)
    - partitioned by a L2 page table
    - unmapped
- L2
  - Splits 1 MB into pages of 4 KB
  - 256 descriptors per table (1 KB)
  - 4 KB of virtual memory mapped to 4 KB of physical one
  - or unmapped

Option 1: we ignore the page tables

- We will fail to guarantee Proof obligations 1 and 2

Option 1: we ignore the page tables

- We will fail to guarantee Proof obligations 1 and 2

$$if I_2(\delta) \quad \text{then}$$

$$\forall pa$$

- $AC(\delta.\sigma_1, pa, 0, acc) \rightarrow pa \in MEM_1$
- $AC(\delta.\sigma_2, pa, 0, acc) \rightarrow pa \in MEM_2$

Option 1: we ignore the page tables

- We will fail to guarantee Proof obligations 1 and 2

$$if\, I_2(\delta) \quad \text{then}$$
$$\forall pa$$
$$\bullet\ AC(\delta.\sigma_1, pa, 0, acc) \rightarrow pa \in MEM_1$$
$$\bullet\ AC(\delta.\sigma_2, pa, 0, acc) \rightarrow pa \in MEM_2$$

$$if\, I_2(\delta)\ \text{then}$$
$$I_2 \text{ does not depend on any } pa \text{ that satisfies}$$
$$\bullet\ AC(\delta.\sigma_1, pa, 0, wt)$$

Option 1: we ignore the page tables

- We will fail to guarantee Proof obligations 1 and 2
- Circumventing the MMU setup allows to
  - gain illicit access to protected resources
- Mandatory security property:
  - complete mediation of the MMU settings

Option 2: we use static page tables

- OK for simple separation kernels
- Can not support a real OS like Linux
  - No process creation
  - No memory allocation

Option 3: we allocate them in the hypervisor memory

- We mediate every update

Option 3: we allocate them in the hypervisor memory

- We mediate every update
- To create a new L1
  - Linux prepares 16 KB of data in its own memory

    and invokes the hypervisor
  - The hypervisor checks the content

    and copy the data into its own memory

Option 3: we allocate them in the hypervisor memory

- We mediate every update
- To create a new L1
  - Linux prepares 16 KB of data in its own memory

    and invokes the hypervisor
  - The hypervisor checks the content

    and copy the data into its own memory
- Simple policy

Option 3: we allocate them in the hypervisor memory

- We mediate every update
- To create a new L1
  - Linux prepares 16 KB of data in its own memory

    and invokes the hypervisor
  - The hypervisor checks the content

    and copy the data into its own memory
- Simple policy
- Inefficient (due to memory copy)
- Requires to statically allocate a pool of page tables

Option 4: Direct paging (e.g. Xen 86)

- Page tables reside in the guest memory
- Guest can manipulate them when they are not in use
- Hypervisor mediates accesses to the active ones
- Guest is in charge of explicitly requesting
  - allocation
  - deallocation
  - update

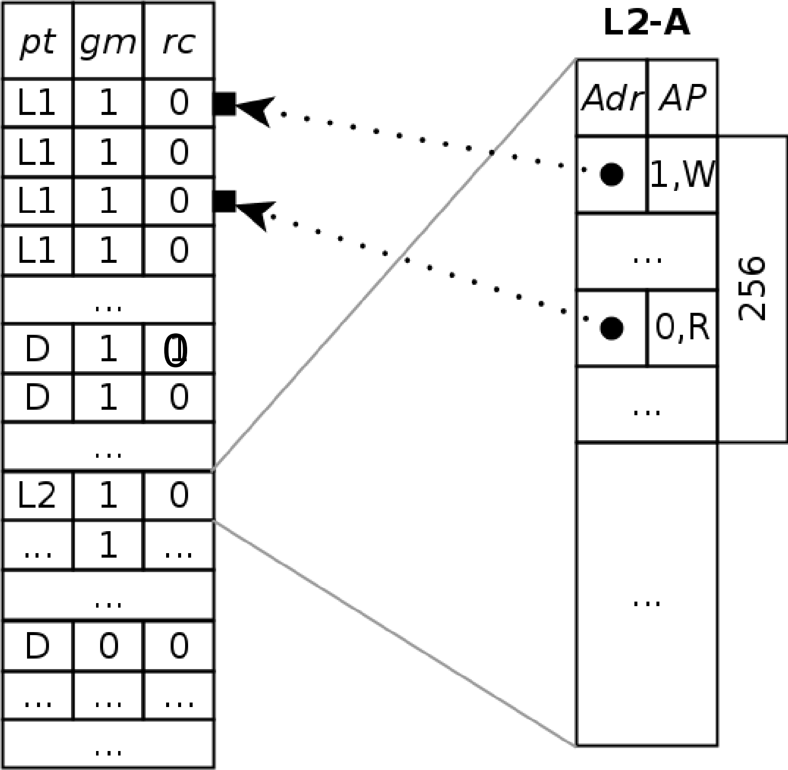
Direct paging

- Physical memory partitioned into blocks of 4KB, typed as
  - L1 page table / L2 page table / Writable
- Hypervisor prevents un-sound requests:
  - no access outside the guest memory
  - no writable access to a block L1/L2
  - correct type for pointed L2s

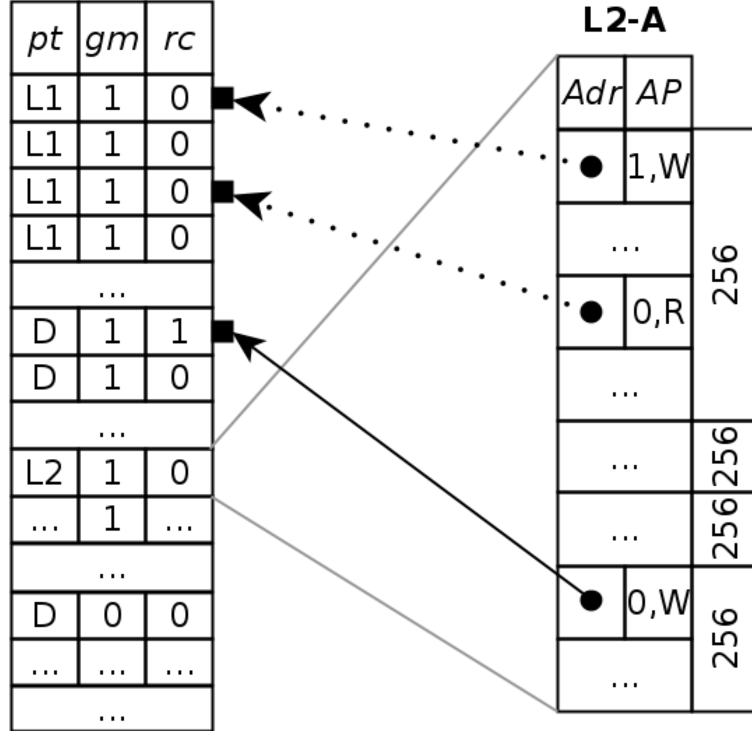
Direct paging

- Physical memory partitioned into blocks of 4KB, typed as
  - L1 page table / L2 page table / Writable
- Hypervisor prevents un-sound requests:
  - no access outside the guest memory
  - no writable access to a block L1/L2
- Reference counter:
  - L2: number of L1s using this L2?
  - Writable: number of writable virtual aliases

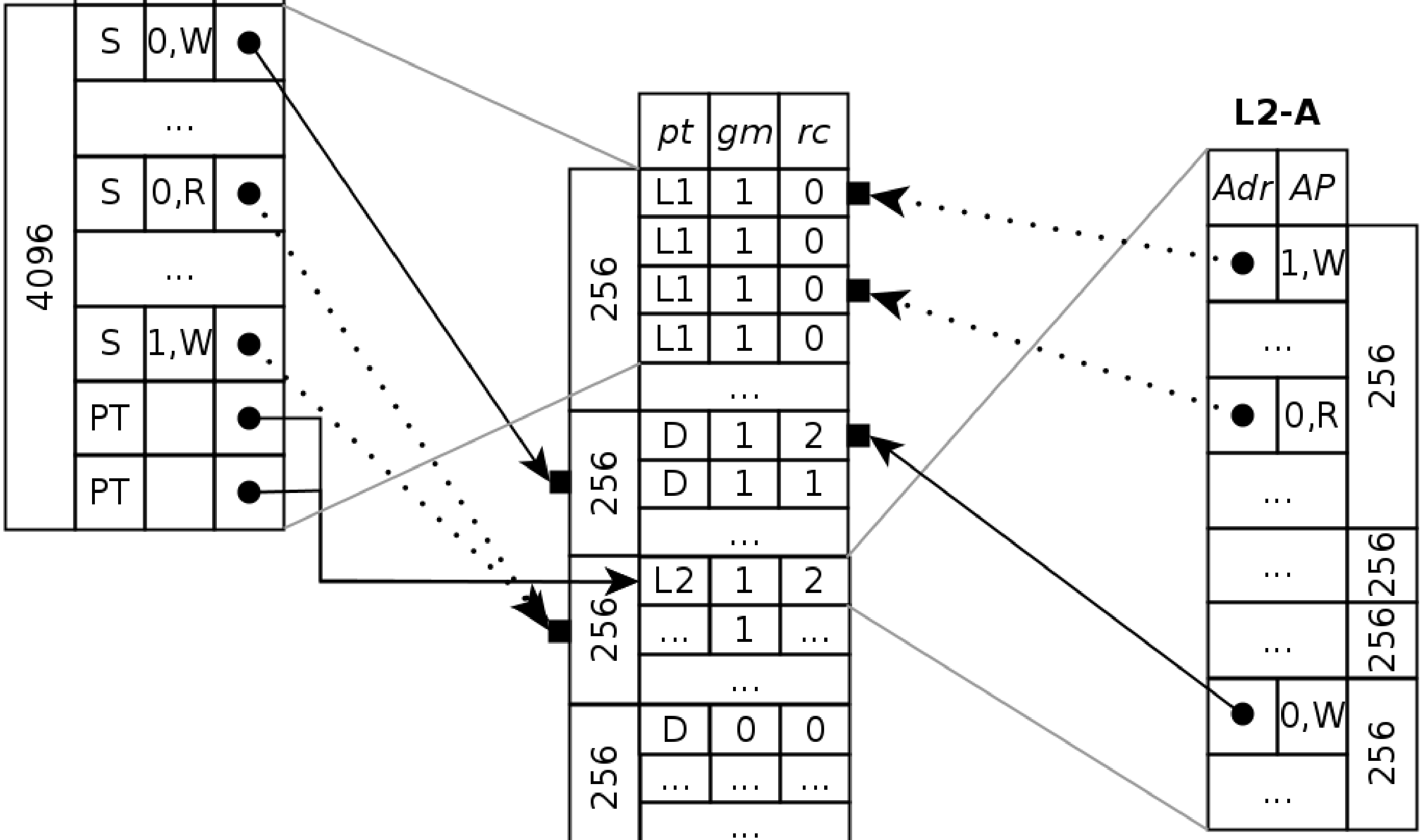
pt	*gm*	*rc*
L1	1	0
L1	1	0
L1	1	0
L1	1	0
	...	
D	1	0
D	1	0
	...	
L2	1	0
...	1	...
	...	
D	0	0
...	...	...
	...	



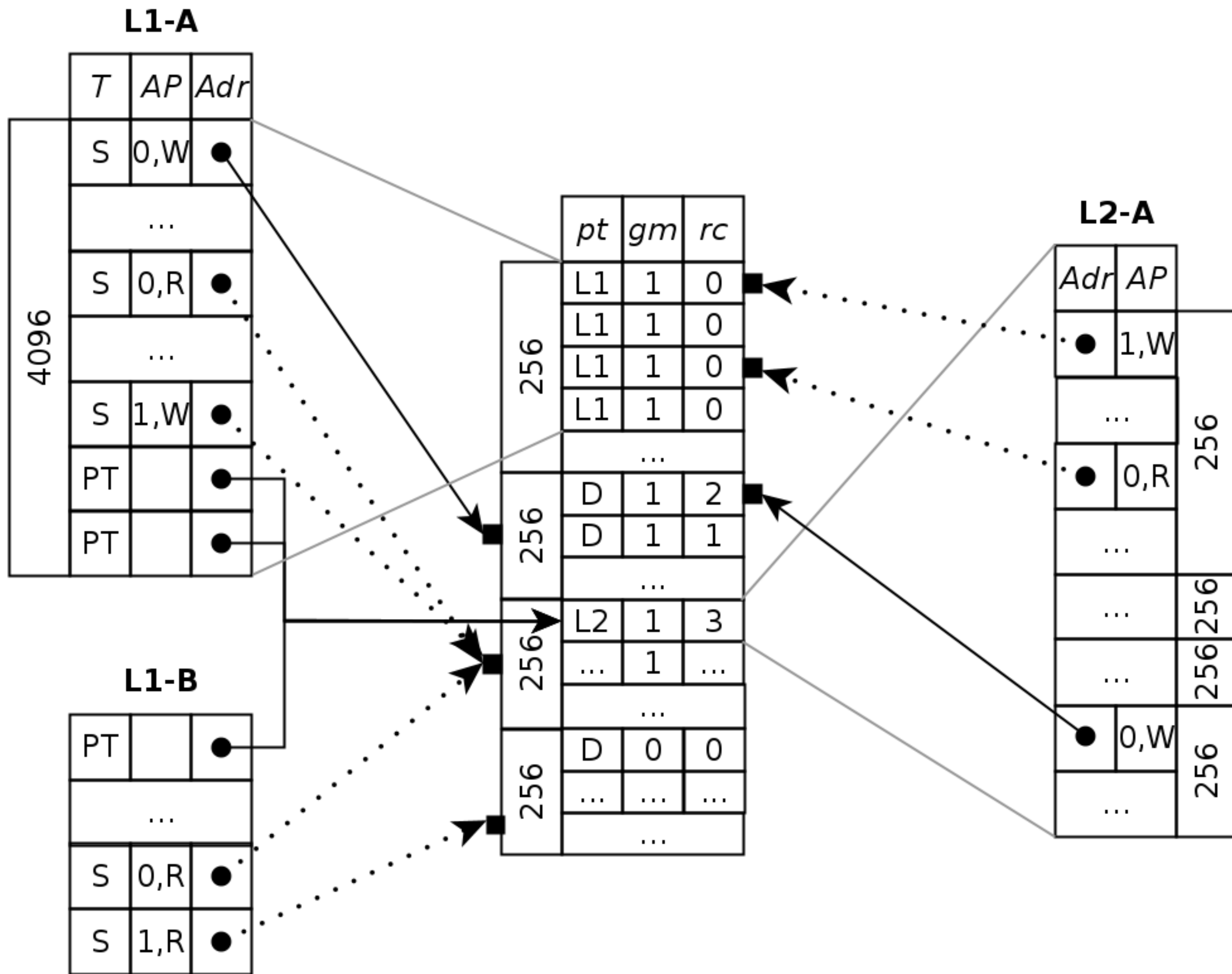
pt
gm
rc
L1 1 0
L1 1 0
L1 1 0
L1 1 0
...
D 1 0
D 1 0
...
L2 1 0
... 1 ...
...
D 0 0
... ...
...
L2-A
Adr
AP
1,W
...
0,R
...
...
256



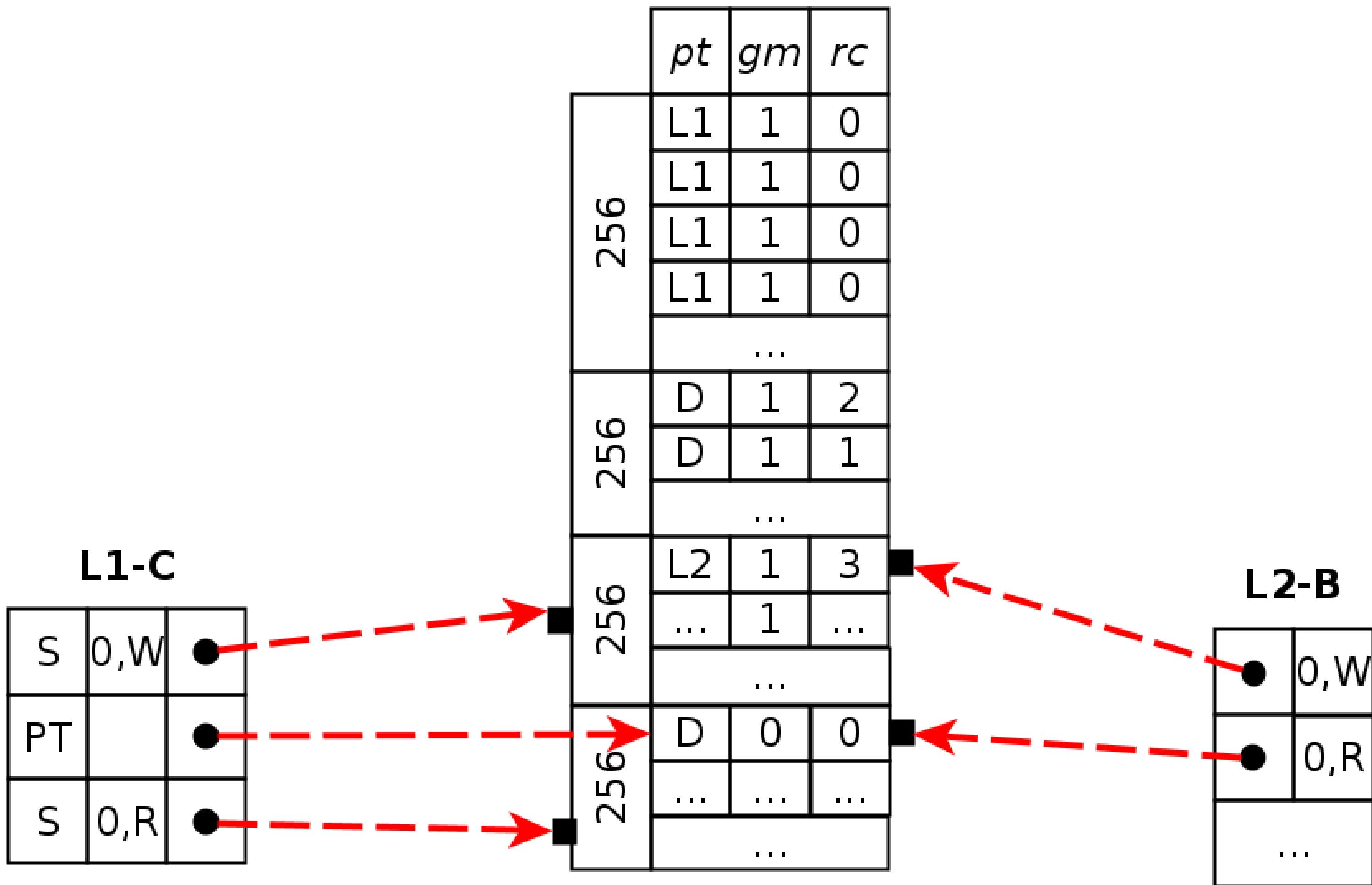
L2-A
pt
gm
rc
L1
1
0
L1
1
0
L1
1
0
L1
1
0
...
D
1
1
D
1
0
...
L2
1
0
...
1
...
...
D
0
0
...
...
...
...
Adr
AP
1,W
...
0,R
...
...
...
0,W
...
256
256
256
256



L1-A
T
AP
Adr
S
0,W
...
S
0,R
...
S
1,W
PT
PT
4096
pt
gm
rc
L1
1
0
L1
1
0
L1
1
0
L1
1
0
...
D
1
2
D
1
1
...
L2
1
2
...
1
...
...
D
0
0
...
...
...
...
256
256
256
256
L2-A
Adr
AP
1,W
...
0,R
...
...
...
0,W
...
256
256
256
256



L1-A
T
AP
Adr
S
0,W
...
S
0,R
...
S
1,W
PT
PT
4096
L1-B
PT
...
S
0,R
S
1,R
pt
gm
rc
L1
1
0
L1
1
0
L1
1
0
L1
1
0
...
D
1
2
D
1
1
...
L2
1
3
...
1
...
...
D
0
0
...
...
...
...
256
256
256
256
L2-A
Adr
AP
1,W
...
0,R
...
...
...
0,W
...
256
256
256
256



pt
gm
rc
256
L1
1
0
L1
1
0
L1
1
0
L1
1
0
...
256
D
1
2
D
1
1
...
L1-C
256
L2
1
3
...
1
...
...
L2-B
S
0,W
PT
S
0,R
256
D
0
0
...
...
...
...
0,W
0,R
...

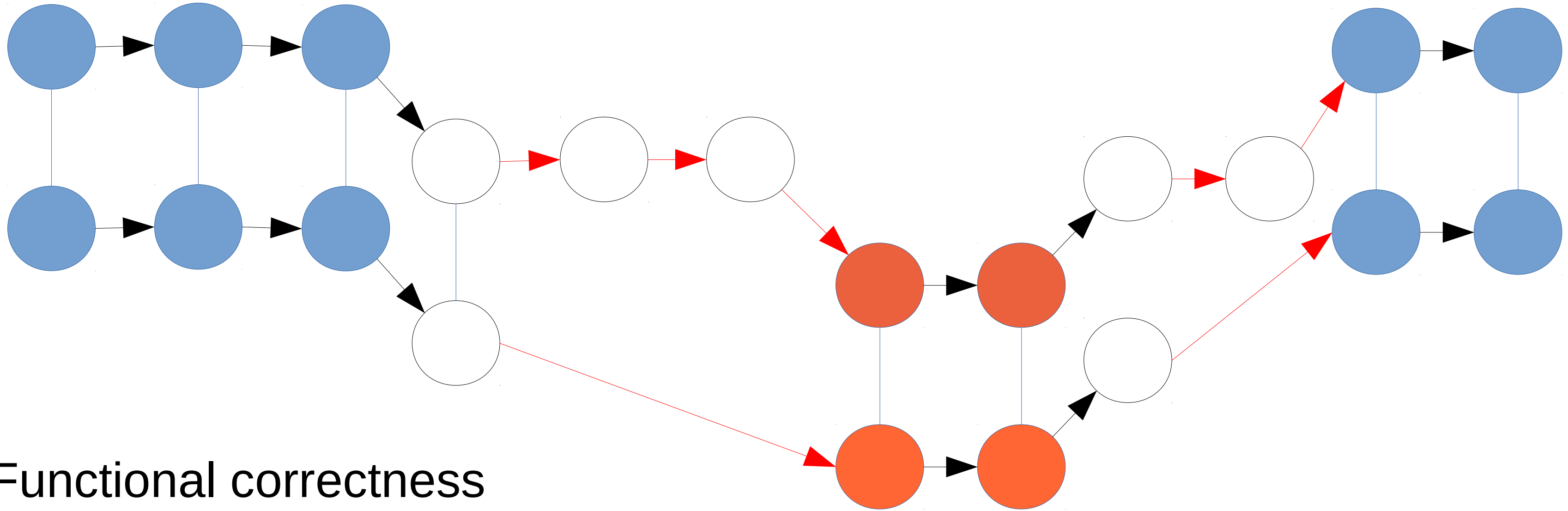
Direct paging

- 9 API to activate, create and free page tables and to map or unmap memory blocks
- Intuitive (and proved) security condition:
  - the block type can change only if its reference counter is zero

Direct paging: e.g. allocation of a L1

1) Guest has at least one writable virtual address
2) The guest writes the content of the PT in the physical block
3) The guest removes (by invoking hypercalls) all mapping that enable itself to write into the physical block
   - The reference counter is now 0
4) The guest can invoke the hypercall “allocate L1”, which
   - validates the entries
   - updates the reference counters
5) the block can now be used by the guest as parameter of “switch page table”, to activate the fresh allocated PT

Proof Decomposition: PL1 transitions



- Functional correctness
- The handlers code respects the specification

PL1 proof

- Verify that the invariant guarantees the proof obligations of the proof

  for PL0

- Prove that the invariant is preserved by the TLS
- Verify that the memory equivalence is preserved by TLS and real transitions:
  - Prove a refinement between the specification and implementation
  - Verify functional correctness of the binary code

PL1 proof: Obligation O1

- Page tables correctly typed
- no access outside the guest memory
- no writable access to a block L1/L2

$$if\, I_1(\sigma) \text{ and } I_2(\delta) \text{ and } \pi_2(\sigma) = \delta.h \text{ then}$$

$$\forall pa$$

- $AC(\delta.\sigma_1, pa, 0, acc) \to pa \in MEM_1$
- $AC(\delta.\sigma_2, pa, 0, acc) \to pa \in MEM_2$
- $AC(\sigma, pa, 0, acc) \to pa \in MEM_{\delta.h.a}$

PL1 proof: Obligation O2

- Page tables correctly typed
- no access outside the guest memory
- no writable access to a block L1/L2
- Invariant depends only on hypervisor data and page tables

$$if I_1(\sigma) \text{ and } I_2(\delta) \text{ and } \pi_2(\sigma) = \delta.h \text{ then}$$

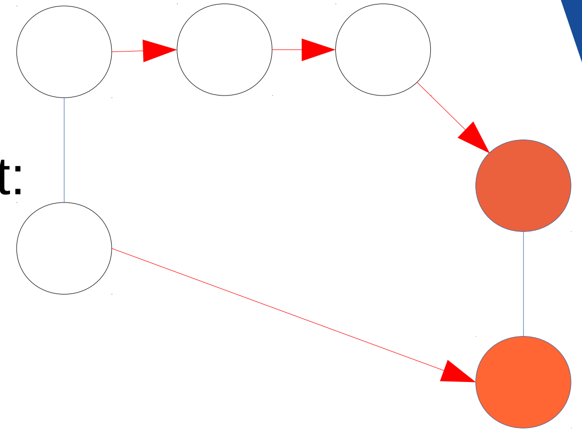
$$I_1, I_2, \pi_1, \pi_2 \text{ do not depend on any } pa \text{ that satisfies}$$

- $AC(\delta.\sigma_1, pa, 0, wt)$
- $AC(\sigma, pa, 0, wt)$

PL1 proof: Invariant preserved by TLS

- Hypervisor TLS decodes the page tables correctly
- Hypervisor data structures are updated correctly
  - e.g. reference counter
- TLS contains all necessary checks
  - Checking reference counter = 0 is a sufficient condition to ensure sound-type change

PL1 proof: refinement



- For every hypervisor handler define a contract:
  - precondition P and postcondition Q
- Show that

$$\text{if } \sigma \approx \delta \text{ then } P(\sigma)$$

- Show that

$$\text{if } \sigma \approx \delta \text{ and } P(\sigma) \text{ and } Q(\sigma, \sigma') \text{ then } \sigma' \approx H(\delta)$$

- Show that the hypervisor respect the contract

PL1 proof: binary code verification

- we use an external tool: Binary Analysis Platform
- we compute the weakest precondition $WP(\sigma)$
- we use an SMT solver to check that $WP(\sigma) \Rightarrow P(\sigma)$

Summary

- ARMv7 MMU
- Direct Paging
- Sketch of proof for PL1
- Binary code verification
- Upcoming
  - Exercises on toy CPU model
  - When the models are unsound

THANKS!

Any questions?

You can find me at robertog@kth.se
http://prosper.sics.se/

References