Lecture 4: Protected Modules Architectures

Secure Compilation Seminar

Marco Patrignani

CISPA
HELMHOLTZ-ZENTRUM i. G.

Protected Modules Architecture (PMA)

- low-level isolation mechanism

Protected Modules Architecture (PMA)

- low-level isolation mechanism
- several research prototypes: Fides [SP], Sancus [Noo+13], Flicker [McC+08], TrustVisor [McC+10], Smart [**smart**]

Protected Modules Architecture (PMA)

- low-level isolation mechanism
- several research prototypes: Fides [SP], Sancus [Noo+13], Flicker [McC+08], TrustVisor [McC+10], Smart [**smart**]
- industrial prototype too: Intel SGX [McK+13]

Protected Modules Architecture (PMA)

- low-level isolation mechanism
- several research prototypes: Fides [SP], Sancus [Noo+13], Flicker [McC+08], TrustVisor [McC+10], Smart [**smart**]
- industrial prototype too: Intel SGX [McK+13]
- implemented via Hypervisor, Hardware, Software

Protected Modules Architecture (PMA)

- low-level isolation mechanism
- several research prototypes: Fides [SP], Sancus [Noo+13], Flicker [McC+08], TrustVisor [McC+10], Smart [**smart**]
- industrial prototype too: Intel SGX [McK+13]
- implemented via Hypervisor, Hardware, Software

What does PMA Provide?

Isolation:

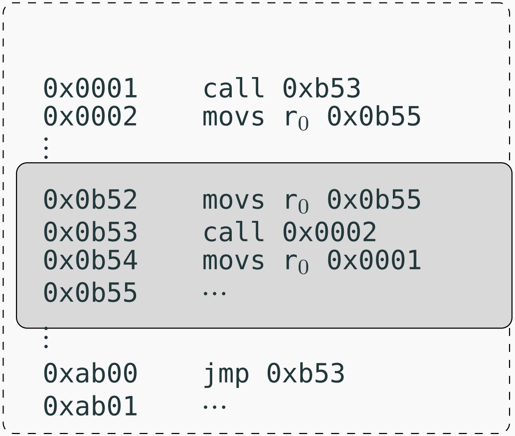
- Code encapsulation
- Data encapsulation
- program counter-based access control
- precise API exposure via Entry Points

PMA in Action (1 Module)

- memory space

```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
⋮
0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0x0001
0x0b55    ⋯
⋮
0xab00    jmp 0xb53
0xab01    ⋯
```

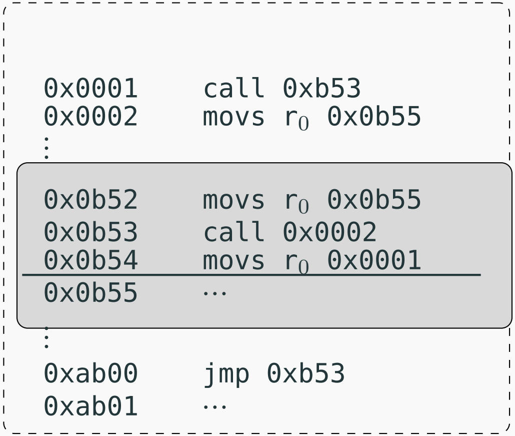
PMA in Action (1 Module)



```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
⋮
0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0x0001
0x0b55    …
⋮
0xab00    jmp 0xb53
0xab01    …
```

- memory space
- protected module = protected memory

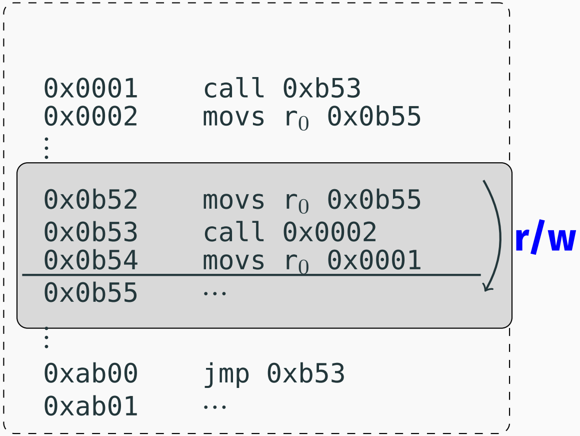
PMA in Action (1 Module)



```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
⋮
0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0x0001
0x0b55    …
⋮
0xab00    jmp 0xb53
0xab01    …
```

- memory space
- protected module = protected memory
- split in code and data

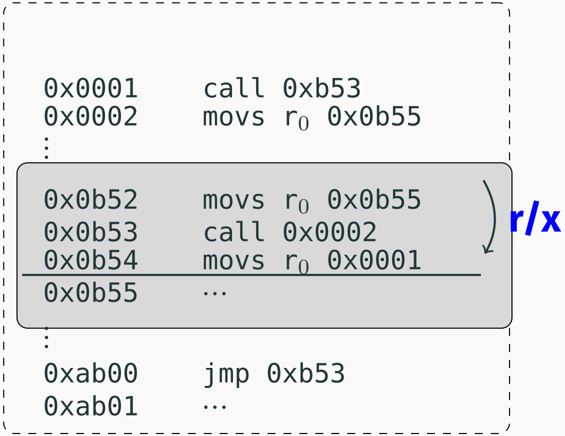
PMA in Action (1 Module)



```
0x0001    call 0xb53
0x0002    movs r₀ 0x0b55
⋮
0x0b52    movs r₀ 0x0b55
0x0b53    call 0x0002
0x0b54    movs r₀ 0x0001
0x0b55    ...
⋮
0xab00    jmp 0xb53
0xab01    ...
```

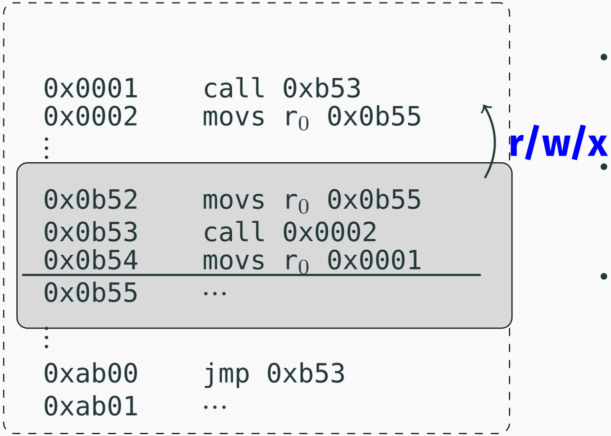
- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted

PMA in Action (1 Module)



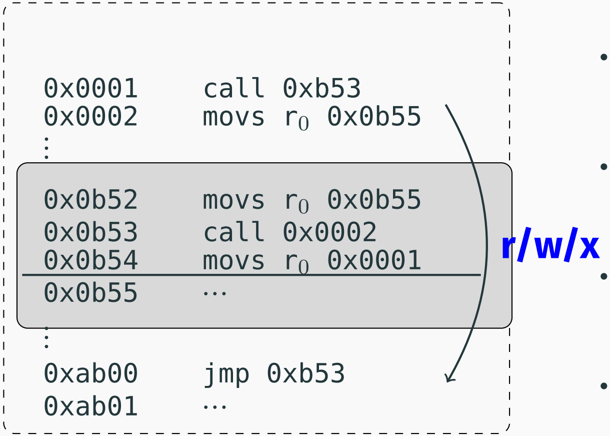
- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted

PMA in Action (1 Module)


```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
⋮
0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0x0001
0x0b55    ...
⋮
0xab00    jmp 0xb53
0xab01    ...
```

- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted

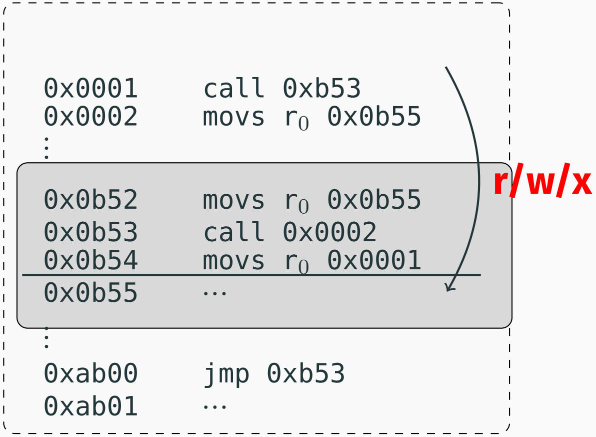
PMA in Action (1 Module)



```
0x0001     call 0xb53
0x0002     movs r0 0x0b55
⋮
0x0b52     movs r0 0x0b55
0x0b53     call 0x0002
0x0b54     movs r0 0x0001
0x0b55     …
⋮
0xab00     jmp 0xb53
0xab01     …
```

- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted
- unprotected code is restricted

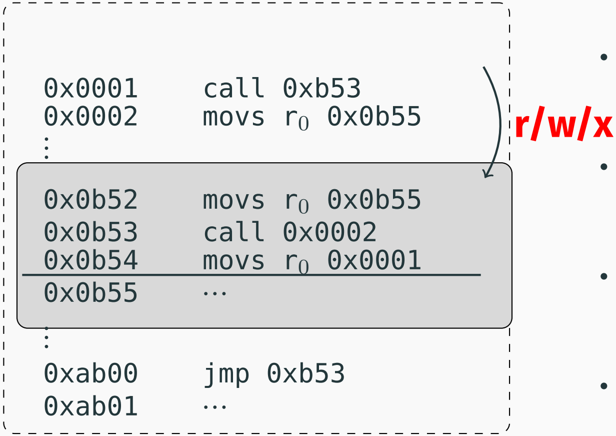
PMA in Action (1 Module)



```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
⋮
0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0x0001
0x0b55    …
⋮
0xab00    jmp 0xb53
0xab01    …
```

- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted
- unprotected code is restricted

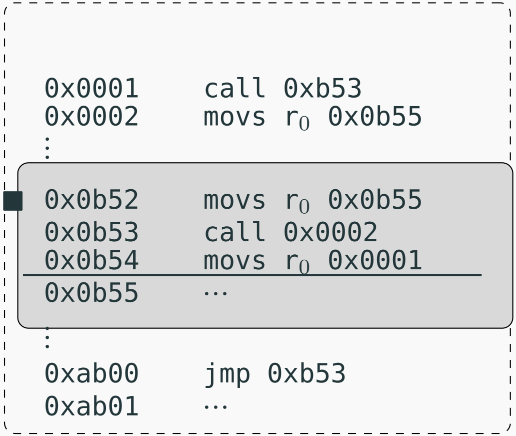
PMA in Action (1 Module)



```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
⋮
0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0x0001
0x0b55    …
⋮
0xab00    jmp 0xb53
0xab01    …
```

- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted
- unprotected code is restricted

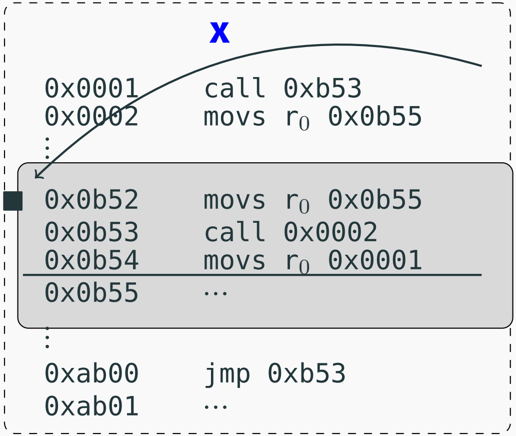
PMA in Action (1 Module)



```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
⋮
0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0x0001
0x0b55    ...
⋮
0xab00    jmp 0xb53
0xab01    ...
```

- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted
- unprotected code is restricted
- entry points for communication (■)

PMA in Action (1 Module)



```
0x0001     call 0xb53
0x0002     movs r0 0x0b55
⋮
0x0b52     movs r0 0x0b55
0x0b53     call 0x0002
0x0b54     movs r0 0x0001
0x0b55     …
⋮
0xab00     jmp 0xb53
0xab01     …
```

- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted
- unprotected code is restricted
- entry points for communication (■)

PMA Access Control Summary

From \ To	Protected			Unprotected
	Entry Point	Code	Data	
Protected	r x	r x	r w	r w x
Unprotected	x			r w x

Access Control Policy enforced based on the PC location

PMA & Assembly Execution

```
 ⋮
10 jmp r_4
11 movi r_1 14
12 jmp r_1
13 cmp r_1 r_2
14 jmp r_3
 ⋮
```

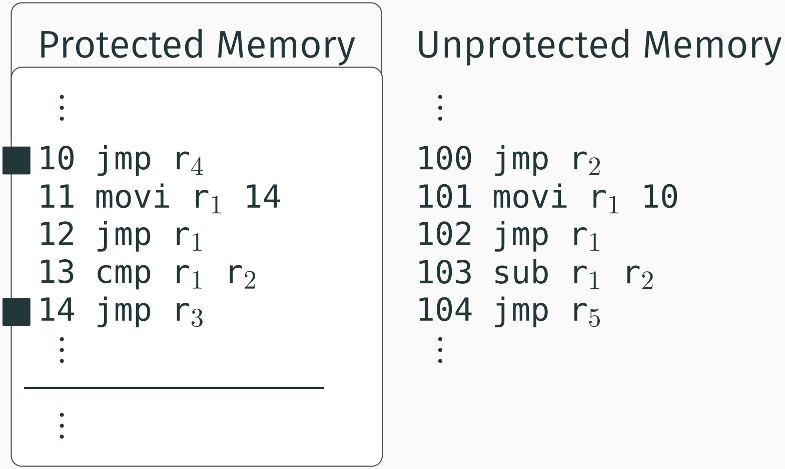
```
 ⋮
100 jmp r_2
101 movi r_1 10
102 jmp r_1
103 sub r_1 r_2
104 jmp r_5
 ⋮
```

```
 ⋮
```

r_1 = 0 ; r_2 = 101 ; r_3 = 100 ;

r_4 = 104 ; r_5 = 11

PMA & Assembly Execution



Protected Memory

⋮

10 jmp r_4

11 movi r_1 14

12 jmp r_1

13 cmp r_1 r_2

14 jmp r_3

⋮

⋮

Unprotected Memory

⋮

100 jmp r_2

101 movi r_1 10

102 jmp r_1

103 sub r_1 r_2

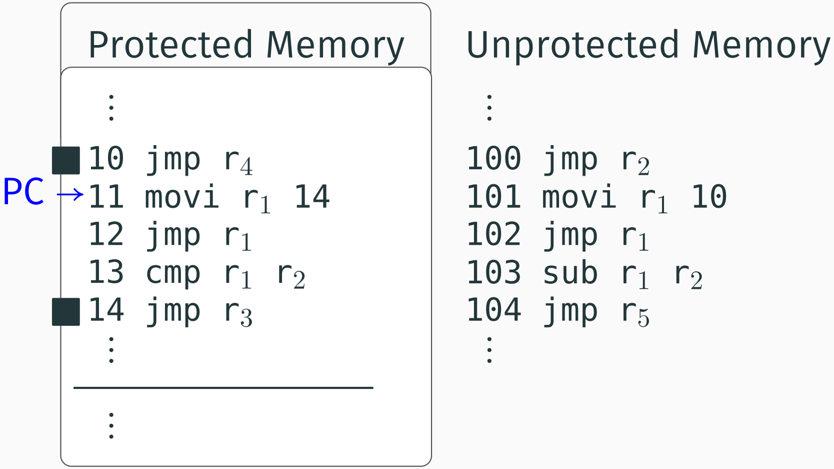
104 jmp r_5

⋮

r_1 = 0 ; r_2 = 101 ; r_3 = 100 ;

r_4 = 104 ; r_5 = 11

PMA & Assembly Execution



Protected Memory

```
 ⋮
10 jmp r4
11 movi r1 14
12 jmp r1
13 cmp r1 r2
14 jmp r3
 ⋮
──────────
 ⋮
```

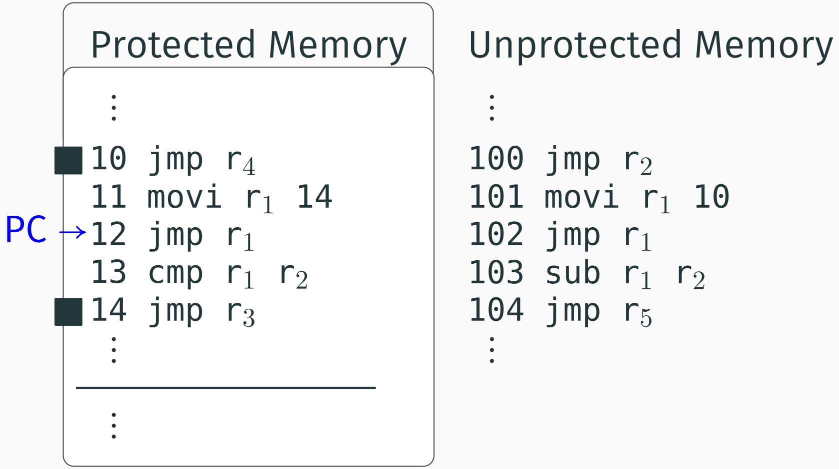
Unprotected Memory

```
 ⋮
100 jmp r2
101 movi r1 10
102 jmp r1
103 sub r1 r2
104 jmp r5
 ⋮
```

$r_1 = 0$; $r_2 = 101$; $r_3 = 100$;

$r_4 = 104$; $r_5 = 11$

PMA & Assembly Execution



Protected Memory

```
⋮
10 jmp r4
11 movi r1 14
12 jmp r1
13 cmp r1 r2
14 jmp r3
⋮
⋮
```

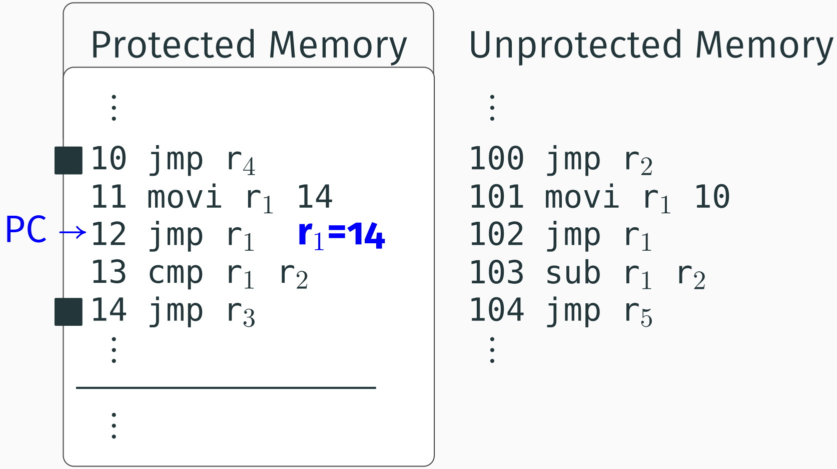
Unprotected Memory

```
⋮
100 jmp r2
101 movi r1 10
102 jmp r1
103 sub r1 r2
104 jmp r5
⋮
```

r_1 = 14 ; r_2 = 101 ; r_3 = 100 ;

r_4 = 104 ; r_5 = 11

PMA & Assembly Execution



Protected Memory

```
 ⋮
■10 jmp r4
 11 movi r1 14
PC →12 jmp r1    r1=14
 13 cmp r1 r2
■14 jmp r3
 ⋮
─────────────
 ⋮
```

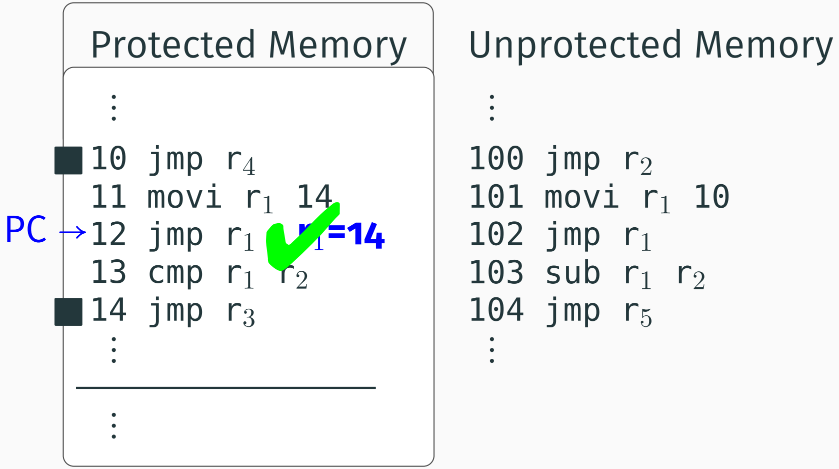
Unprotected Memory

```
 ⋮
100 jmp r2
101 movi r1 10
102 jmp r1
103 sub r1 r2
104 jmp r5
 ⋮
```

r_1 = 14 ; r_2 = 101 ; r_3 = 100 ;

r_4 = 104 ; r_5 = 11

PMA & Assembly Execution



Protected Memory:

```
  ⋮
10 jmp r4
11 movi r1 14
12 jmp r1
13 cmp r1 r2
14 jmp r3
  ⋮
```

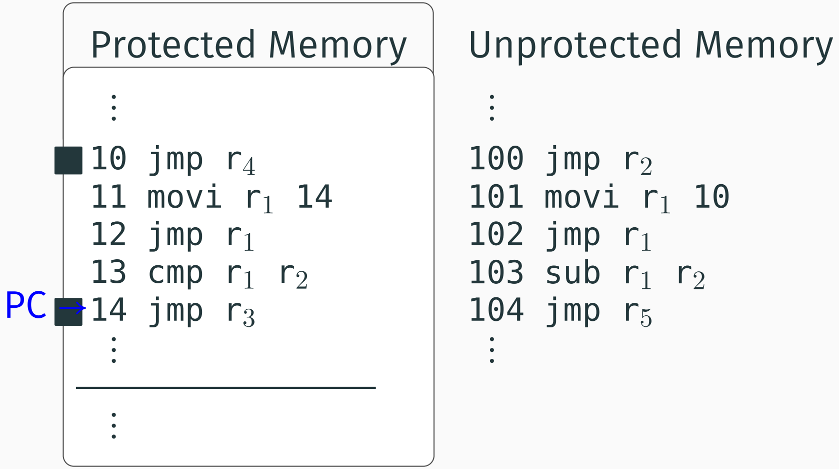
Unprotected Memory:

```
  ⋮
100 jmp r2
101 movi r1 10
102 jmp r1
103 sub r1 r2
104 jmp r5
  ⋮
```

r_1 = 14 ; r_2 = 101 ; r_3 = 100 ;

r_4 = 104 ; r_5 = 11

PMA & Assembly Execution

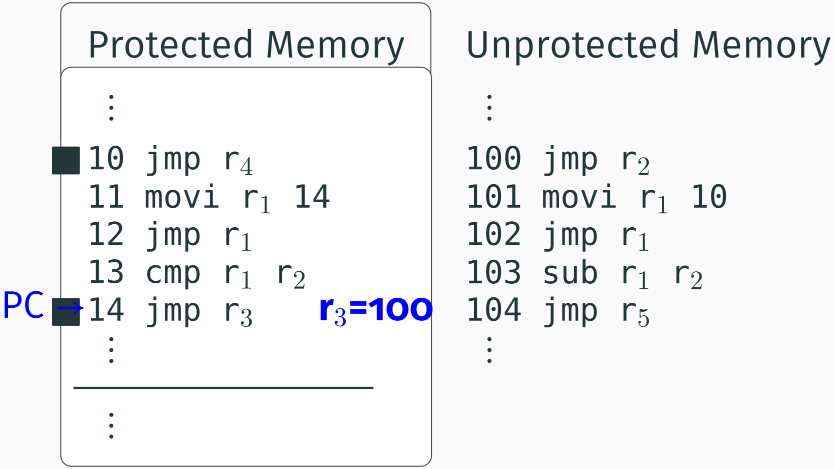


```
Protected Memory        Unprotected Memory
  ⋮                       ⋮
10 jmp r4               100 jmp r2
11 movi r1 14           101 movi r1 10
12 jmp r1               102 jmp r1
13 cmp r1 r2            103 sub r1 r2
14 jmp r3               104 jmp r5
  ⋮                       ⋮
  ⋮
```

r_1 = 14 ; r_2 = 101 ; r_3 = 100 ;

r_4 = 104 ; r_5 = 11

PMA & Assembly Execution

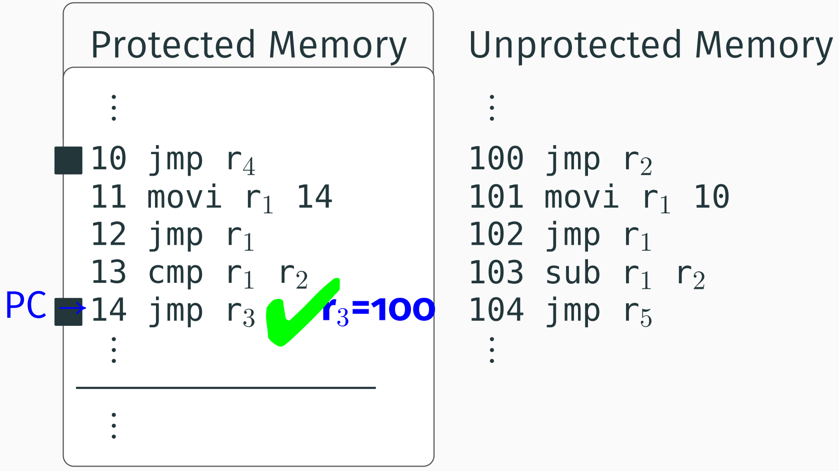


```
Protected Memory        Unprotected Memory
   ⋮                       ⋮
10 jmp r4               100 jmp r2
11 movi r1 14           101 movi r1 10
12 jmp r1               102 jmp r1
13 cmp r1 r2            103 sub r1 r2
14 jmp r3               104 jmp r5
   ⋮                       ⋮
   ⋮
```

r_1 = 14 ; r_2 = 101 ; r_3 = 100 ;

r_4 = 104 ; r_5 = 11

PMA & Assembly Execution



Protected Memory

```
  ⋮
10 jmp r4
11 movi r1 14
12 jmp r1
13 cmp r1 r2
14 jmp r3
  ⋮
```

```
  ⋮
```

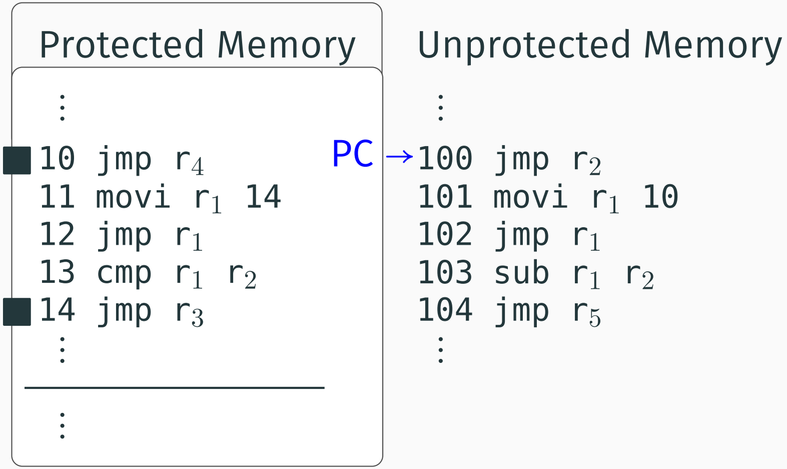
Unprotected Memory

```
  ⋮
100 jmp r2
101 movi r1 10
102 jmp r1
103 sub r1 r2
104 jmp r5
  ⋮
```

r_1 = 14 ; r_2 = 101 ; r_3 = 100 ;

r_4 = 104 ; r_5 = 11

PMA & Assembly Execution



Protected Memory:

```
 ⋮
10 jmp r4
11 movi r1 14
12 jmp r1
13 cmp r1 r2
14 jmp r3
 ⋮
─────────────
 ⋮
```

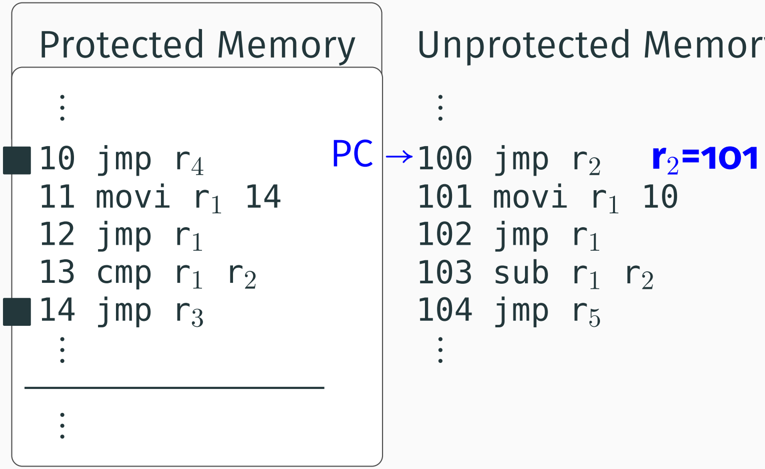
Unprotected Memory:

```
 ⋮
100 jmp r2
101 movi r1 10
102 jmp r1
103 sub r1 r2
104 jmp r5
 ⋮
```

$r_1 = 14$; $r_2 = 101$; $r_3 = 100$;

$r_4 = 104$; $r_5 = 11$

PMA & Assembly Execution



Protected Memory

⋮

10 jmp r_4

11 movi r_1 14

12 jmp r_1

13 cmp r_1 r_2

14 jmp r_3

⋮

⋮

Unprotected Memory

⋮

PC → 100 jmp r_2 **r_2=101**

101 movi r_1 10

102 jmp r_1

103 sub r_1 r_2

104 jmp r_5

⋮

$r_1 = 10$; $r_2 = 101$; $r_3 = 100$;

$r_4 = 104$; $r_5 = 11$

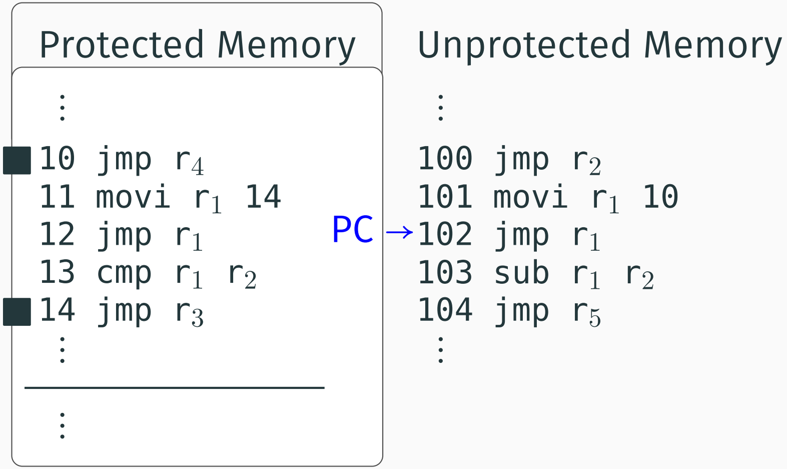
PMA & Assembly Execution

Protected Memory		Unprotected Memory
⋮		⋮
■ 10 jmp r_4		100 jmp r_2
11 movi r_1 14	PC →	101 movi r_1 10
12 jmp r_1		102 jmp r_1
13 cmp r_1 r_2		103 sub r_1 r_2
■ 14 jmp r_3		104 jmp r_5
⋮		⋮
———————		
⋮		

r_1 = 10 ; r_2 = 101 ; r_3 = 100 ;

r_4 = 104 ; r_5 = 11

PMA & Assembly Execution



Protected Memory

```
⋮
■ 10 jmp r4
  11 movi r1 14
  12 jmp r1
  13 cmp r1 r2
■ 14 jmp r3
⋮
────────────────
⋮
```

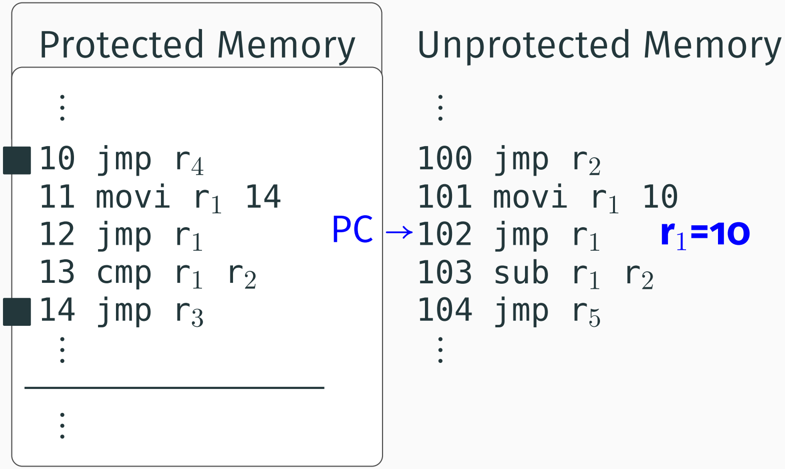
Unprotected Memory

```
       ⋮
       100 jmp r2
       101 movi r1 10
PC →   102 jmp r1
       103 sub r1 r2
       104 jmp r5
       ⋮
```

r_1 = 10 ; r_2 = 101 ; r_3 = 100 ;

r_4 = 104 ; r_5 = 11

PMA & Assembly Execution



Protected Memory

⋮

■ 10 jmp r_4
11 movi r_1 14
12 jmp r_1
13 cmp r_1 r_2
■ 14 jmp r_3

⋮

⋮

Unprotected Memory

⋮

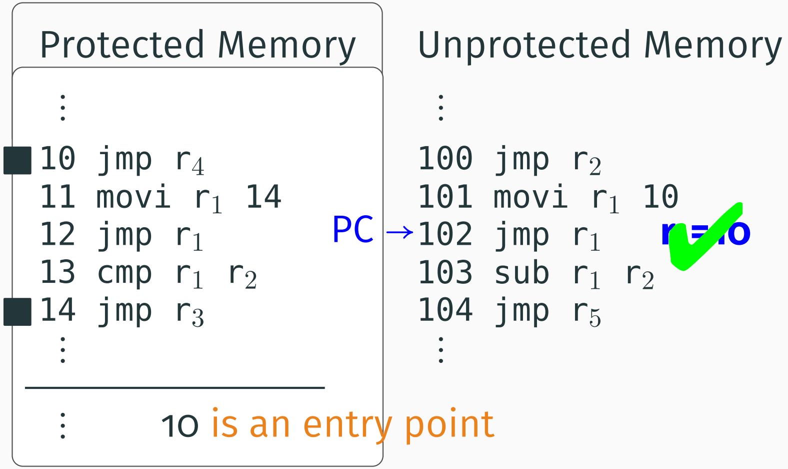
100 jmp r_2
101 movi r_1 10
102 jmp r_1 **r_1=10**
103 sub r_1 r_2
104 jmp r_5

⋮

r_1 = 10 ; r_2 = 101 ; r_3 = 100 ;

r_4 = 104 ; r_5 = 11

PMA & Assembly Execution



Protected Memory

```
 ⋮
■10 jmp r4
 11 movi r1 14
 12 jmp r1
 13 cmp r1 r2
■14 jmp r3
 ⋮
 ───────────
 ⋮
```

Unprotected Memory

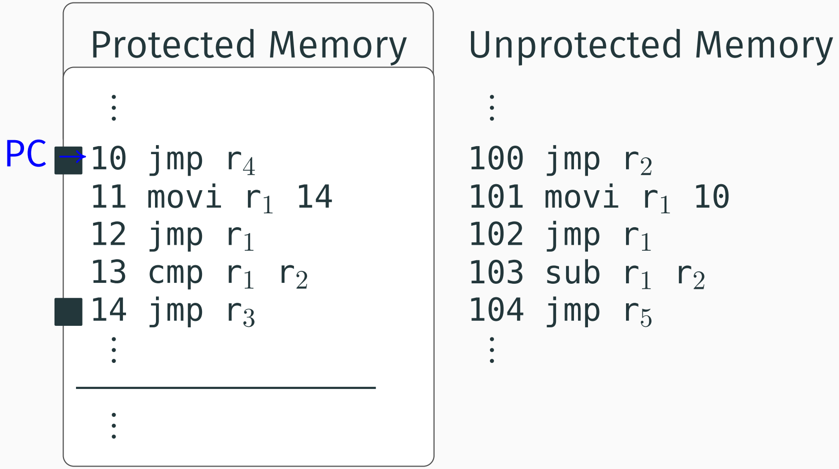
```
     ⋮
     100 jmp r2
     101 movi r1 10
PC → 102 jmp r1      r=10 ✓
     103 sub r1 r2
     104 jmp r5
     ⋮
```

10 is an entry point

r_1 = 10 ; r_2 = 101 ; r_3 = 100 ;

r_4 = 104 ; r_5 = 11

PMA & Assembly Execution



Protected Memory

⋮

10 jmp r_4

11 movi r_1 14

12 jmp r_1

13 cmp r_1 r_2

14 jmp r_3

⋮

⋮

Unprotected Memory

⋮

100 jmp r_2

101 movi r_1 10

102 jmp r_1

103 sub r_1 r_2

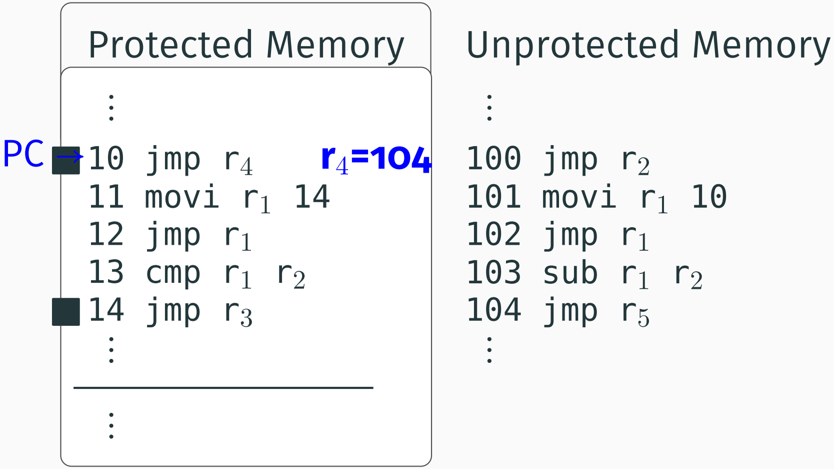
104 jmp r_5

⋮

r_1 = 10 ; r_2 = 101 ; r_3 = 100 ;

r_4 = 104 ; r_5 = 11

PMA & Assembly Execution



Protected Memory

⋮

10 jmp r_4
11 movi r_1 14
12 jmp r_1
13 cmp r_1 r_2
14 jmp r_3

⋮

⋮

Unprotected Memory

⋮

100 jmp r_2
101 movi r_1 10
102 jmp r_1
103 sub r_1 r_2
104 jmp r_5

⋮

r_1 = 10 ; r_2 = 101 ; r_3 = 100 ;

r_4 = 104 ; r_5 = 11

PMA & Assembly Execution

Protected Memory

⋮

PC → 10 jmp r_4 ✓ r_4=104

11 movi r_1 14

12 jmp r_1

13 cmp r_1 r_2

14 jmp r_3

⋮

⋮

Unprotected Memory

⋮

100 jmp r_2

101 movi r_1 10

102 jmp r_1

103 sub r_1 r_2

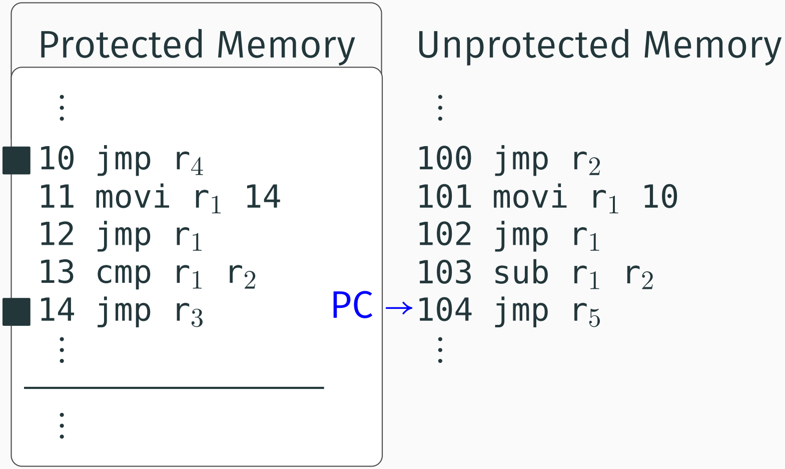
104 jmp r_5

⋮

r_1 = 10 ; r_2 = 101 ; r_3 = 100 ;

r_4 = 104 ; r_5 = 11

PMA & Assembly Execution



Protected Memory

⋮

■ 10 jmp r_4

11 movi r_1 14

12 jmp r_1

13 cmp r_1 r_2

■ 14 jmp r_3

⋮

⋮

Unprotected Memory

⋮

100 jmp r_2

101 movi r_1 10

102 jmp r_1

103 sub r_1 r_2

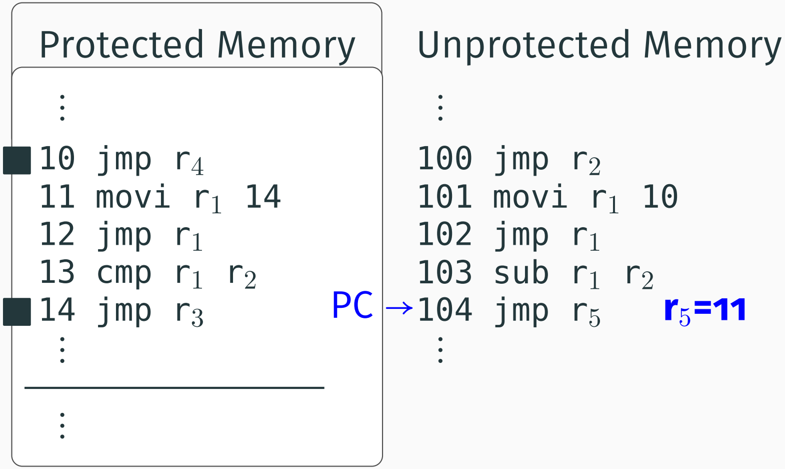
PC → 104 jmp r_5

⋮

r_1 = 10 ; r_2 = 101 ; r_3 = 100 ;

r_4 = 104 ; r_5 = 11

PMA & Assembly Execution



Protected Memory

```
⋮
10 jmp r4
11 movi r1 14
12 jmp r1
13 cmp r1 r2
14 jmp r3
⋮
────────────
⋮
```

Unprotected Memory

```
⋮
100 jmp r2
101 movi r1 10
102 jmp r1
103 sub r1 r2
104 jmp r5
⋮
```

r_1 = 10 ; r_2 = 101 ; r_3 = 100 ;

r_4 = 104 ; r_5 = 11

PMA & Assembly Execution

Protected Memory

⋮

■ 10 jmp r_4
11 movi r_1 14
12 jmp r_1
13 cmp r_1 r_2
■ 14 jmp r_3

⋮

⋮ 11 is not an entry point

Unprotected Memory

⋮

100 jmp r_2
101 movi r_1 10
102 jmp r_1
103 sub r_1 r_2
PC → 104 jmp r_5 ✗ r_5=11

⋮

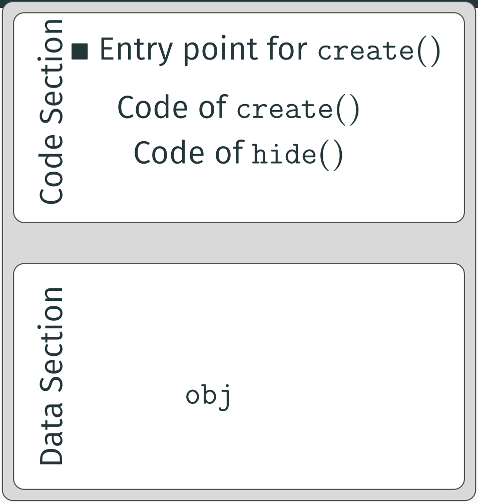
r_1 = 10 ; r_2 = 101 ; r_3 = 100 ;
r_4 = 104 ; r_5 = 11

Entry Points = API

```
class C1
  public create() : C1{
    this.hide();
    return new C1();
  }
  private hide(): Unit{
    return null;
  }
object obj : C1;
```

Entry Points = API

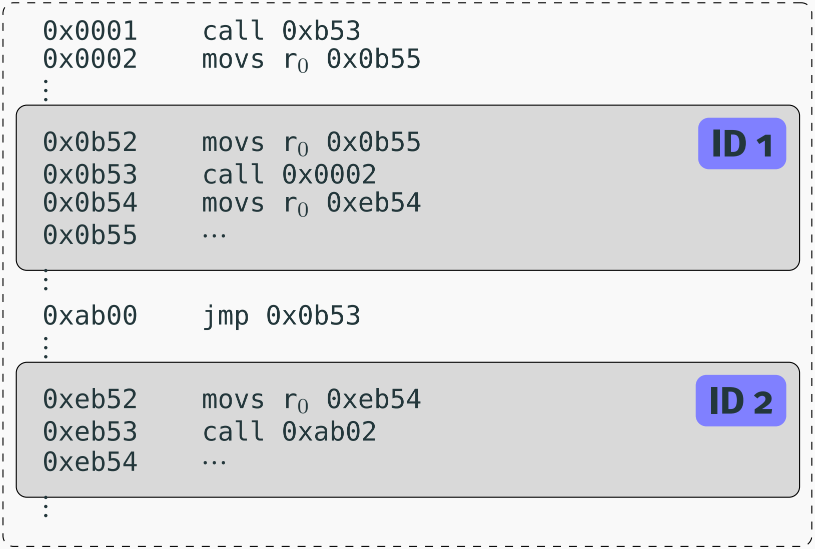
```
class C1
  public create() : C1{
    this.hide();
    return new C1();
  }
  private hide(): Unit{
    return null;
  }
object obj : C1;
```



PMA in Action (N Modules)

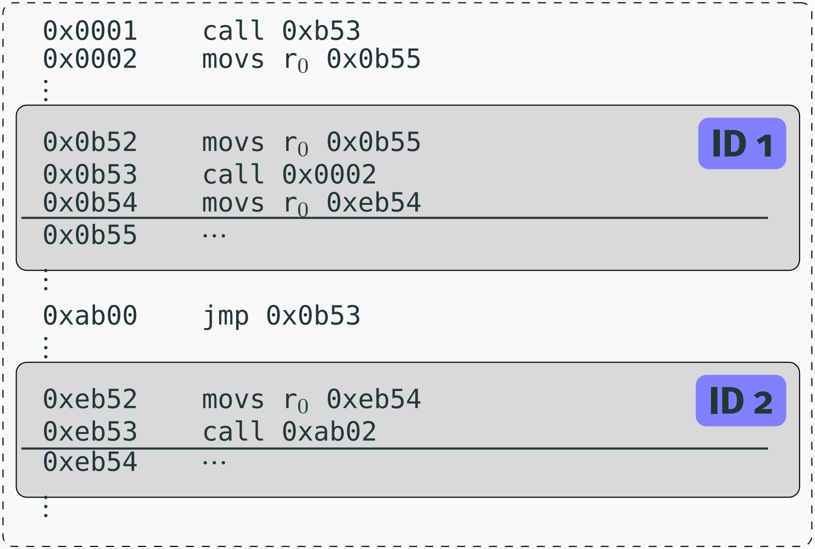
```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
⋮

0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0xeb54
0x0b55    …
⋮
0xab00    jmp 0x0b53
⋮

0xeb52    movs r0 0xeb54
0xeb53    call 0xab02
0xeb54    …
⋮
```

PMA in Action (N Modules)



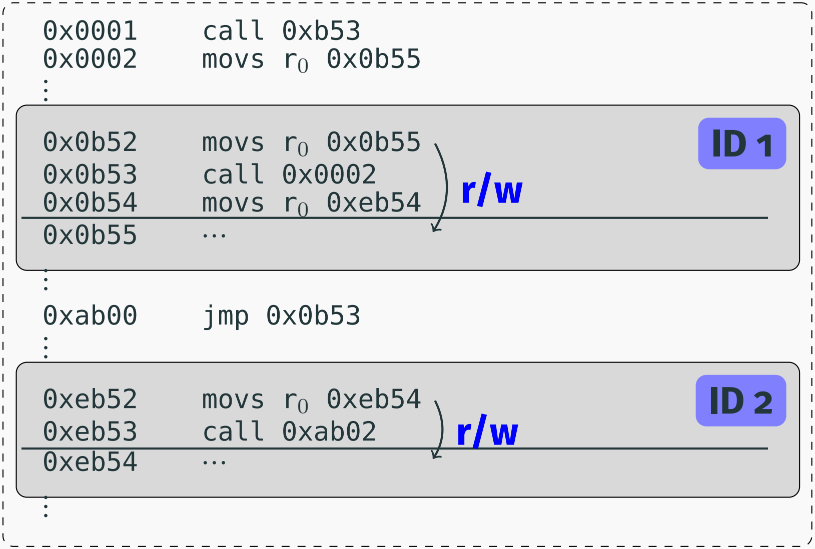
```
0x0001    call 0xb53
0x0002    movs r_0 0x0b55
⋮
0x0b52    movs r_0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r_0 0xeb54
0x0b55    ⋯
⋮
0xab00    jmp 0x0b53
⋮
0xeb52    movs r_0 0xeb54
0xeb53    call 0xab02
0xeb54    ⋯
⋮
```

PMA in Action (N Modules)



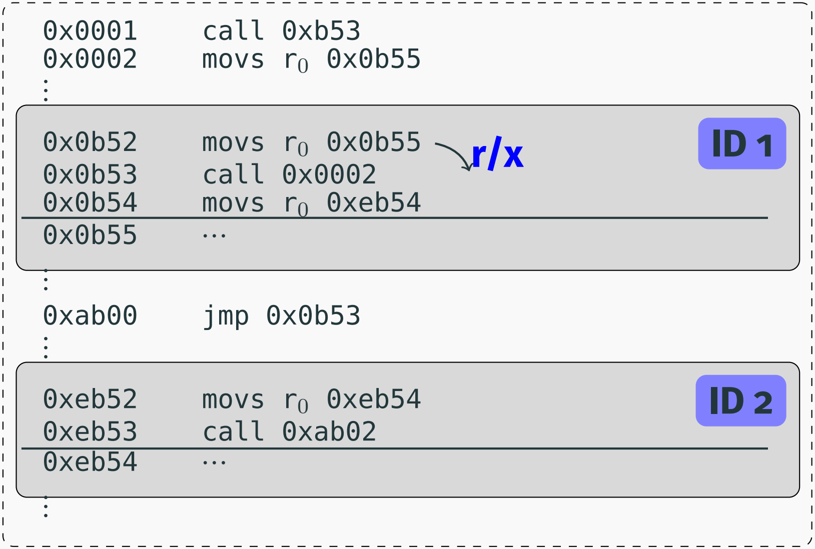
```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
⋮
0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0xeb54
0x0b55    ...
⋮
0xab00    jmp 0x0b53
⋮
0xeb52    movs r0 0xeb54
0xeb53    call 0xab02
0xeb54    ...
⋮
```

PMA in Action (N Modules)



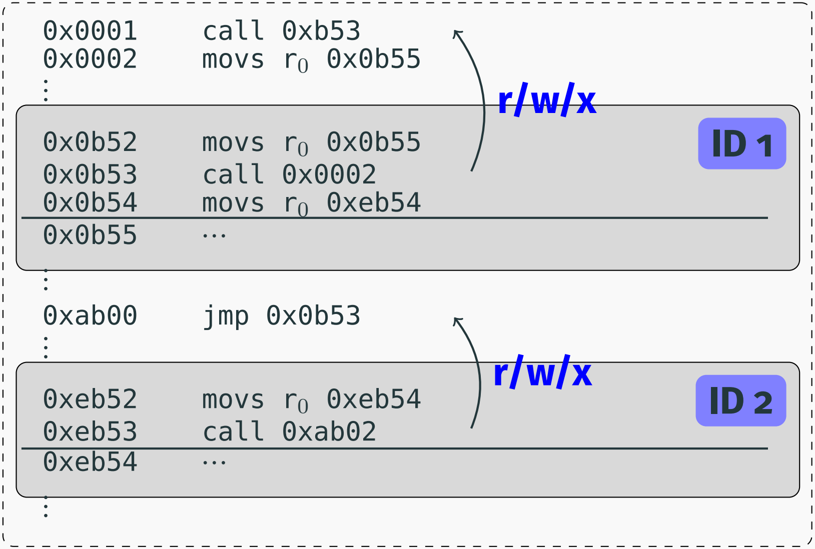
```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
⋮
0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0xeb54
0x0b55    ...
⋮
0xab00    jmp 0x0b53
⋮
0xeb52    movs r0 0xeb54
0xeb53    call 0xab02
0xeb54    ...
⋮
```

PMA in Action (N Modules)



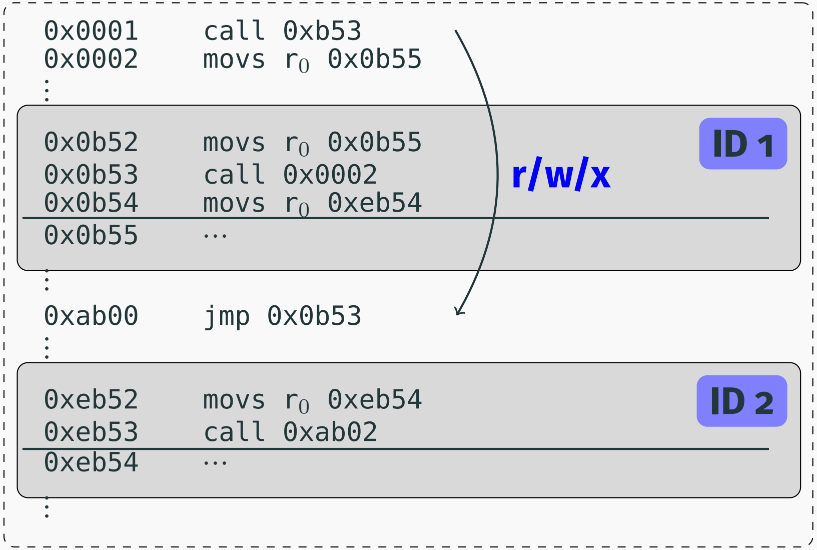
```
0x0001    call 0xb53
0x0002    movs r_0 0x0b55
:

0x0b52    movs r_0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r_0 0xeb54
-------------------------------------------
0x0b55    ...
:

0xab00    jmp 0x0b53
:

0xeb52    movs r_0 0xeb54
0xeb53    call 0xab02
-------------------------------------------
0xeb54    ...
:
```

PMA in Action (N Modules)



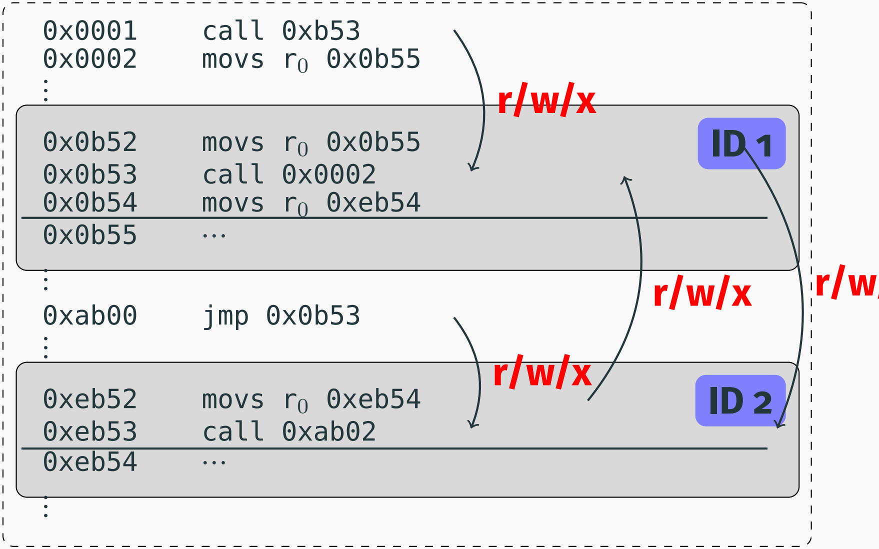
```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
:
0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0xeb54
0x0b55    ...
:
0xab00    jmp 0x0b53
:
0xeb52    movs r0 0xeb54
0xeb53    call 0xab02
0xeb54    ...
:
```

PMA in Action (N Modules)



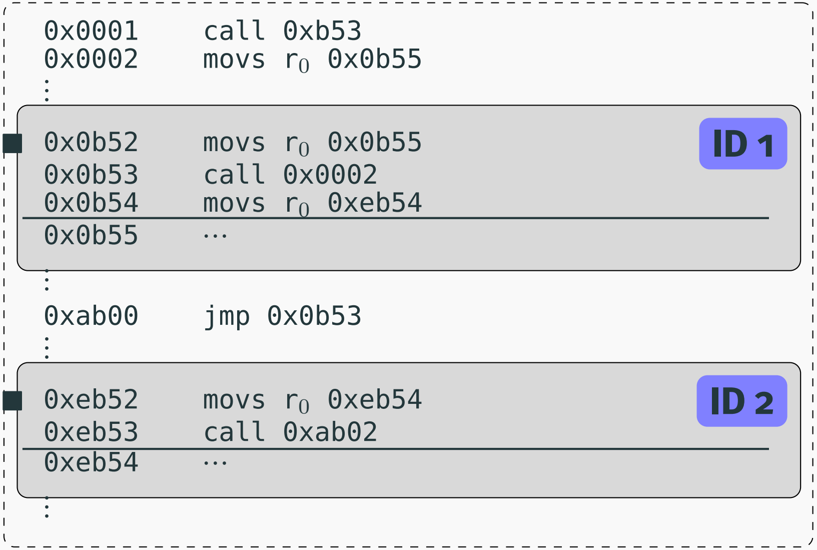
```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
⋮
0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0xeb54
0x0b55    ...
⋮
0xab00    jmp 0x0b53
⋮
0xeb52    movs r0 0xeb54
0xeb53    call 0xab02
0xeb54    ...
⋮
```

PMA in Action (N Modules)



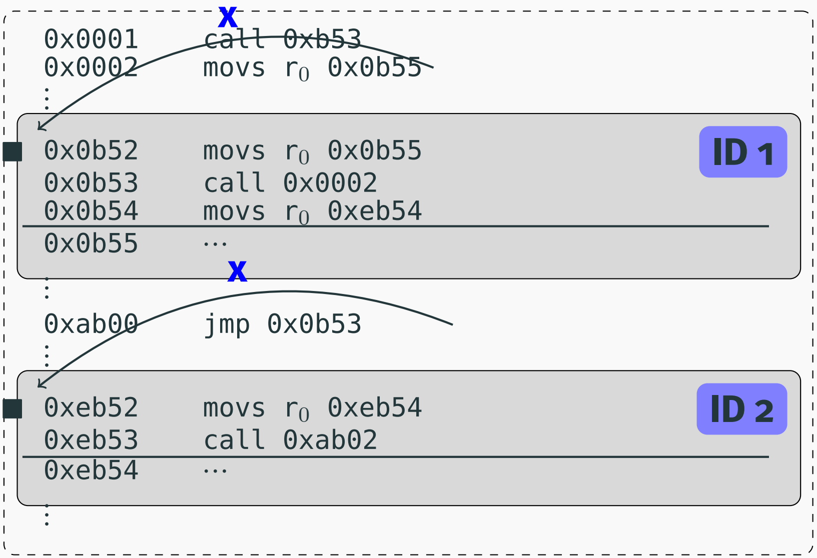
```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
⋮
0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0xeb54
0x0b55    ...
⋮
0xab00    jmp 0x0b53
⋮
0xeb52    movs r0 0xeb54
0xeb53    call 0xab02
0xeb54    ...
⋮
```

PMA in Action (N Modules)

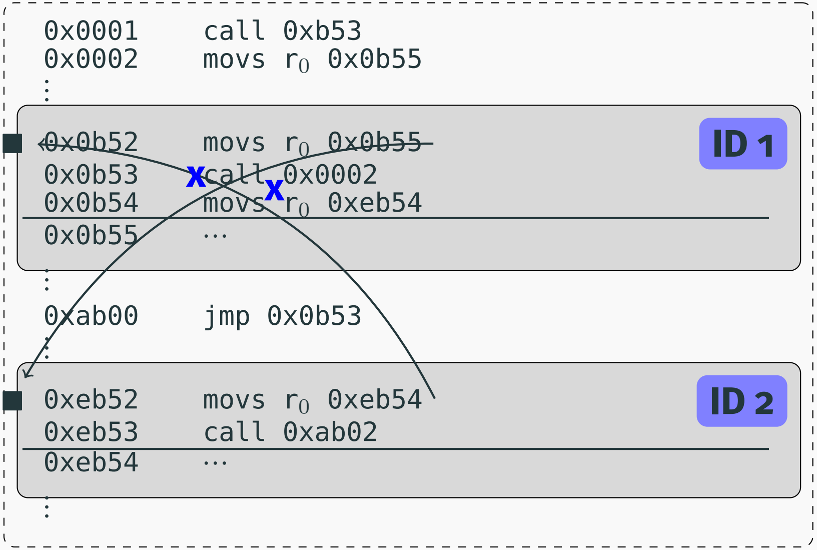


```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
⋮
0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0xeb54
0x0b55    ...
⋮
0xab00    jmp 0x0b53
⋮
0xeb52    movs r0 0xeb54
0xeb53    call 0xab02
0xeb54    ...
⋮
```

PMA in Action (N Modules)



PMA in Action (N Modules)



```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
⋮
0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0xeb54
0x0b55    ...
⋮
0xab00    jmp 0x0b53
⋮
0xeb52    movs r0 0xeb54
0xeb53    call 0xab02
0xeb54    ...
⋮
```

PMA & Trust

- Coarse-grained trust domains

PMA & Trust

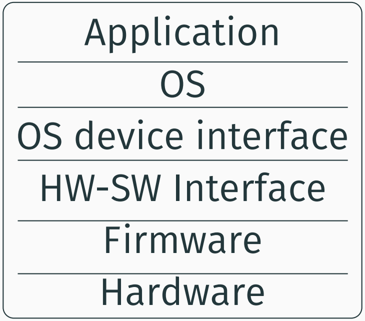
- Coarse-grained trust domains
- The same trust domain fits the same Module

PMA & Trust

- Coarse-grained trust domains
- The same trust domain fits the same Module
- May require libraries and code to be split among different Modules

Threat Models

Different implementations address different attacks:



Formalising PMA

Language: assembly

Formalising PMA

Language: assembly

- Instruction list

Formalising PMA

Language: assembly

- Instruction list
- Memory

Formalising PMA

Language: assembly

- Instruction list
- Memory
- Registers file

Formalising PMA

Language: assembly

- Instruction list
- Memory
- Registers file
- Module sizes

Helper functions for the ACP

References

[McC+08] Jonathan M. McCune et al. “Flicker: an execution infrastructure for TCB minimization”. In: *SIGOPS Oper. Syst. Rev.* 42 (Apr. 2008), pp. 315–328. ISSN: 0163-5980.

[McC+10] Jonathan M. McCune et al. “TrustVisor: Efficient TCB Reduction and Attestation”. In: *SP ’10*. IEEE, 2010, pp. 143–158. ISBN: 978-0-7695-4035-1.

[McK+13] Frank McKeen et al. “Innovative instructions and software model for isolated execution”. In: *HASP ’13*. Tel-Aviv, Israel, 2013. ISBN: 978-1-4503-2118-1.

[Noo+13] Job Noorman et al. “Sancus: Low-cost trustworthy extensible networked devices with a zero-software Trusted Computing Base”. In: *Proceedings of the 22nd USENIX conference on Security symposium*. 2013.

[SP] Raoul Strackx and Frank Piessens. “Fides: selectively hardening software application components against kernel-level or process-level malware”. In: *Proceedings CCS ’12*.