Fully Abstract Trace Semantics of Low-level Protection Mechanisms

Marco Patrignani Dave Clarke

iMinds-DistriNet, Dept. Computer Science, Katholieke Universiteit Leuven

1 November 2012

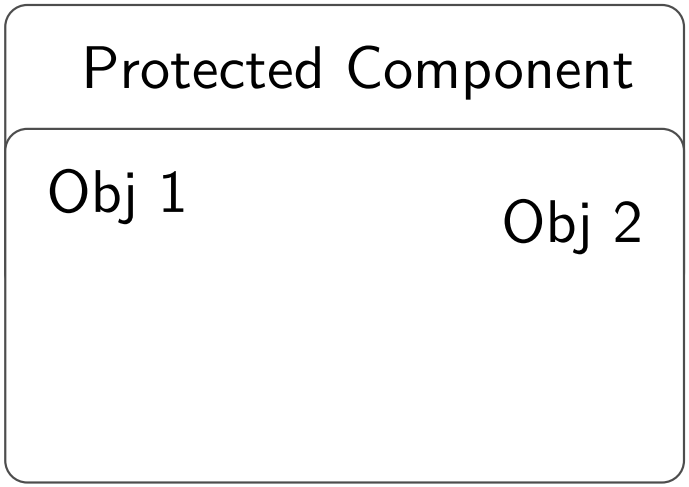
Outline

Outline

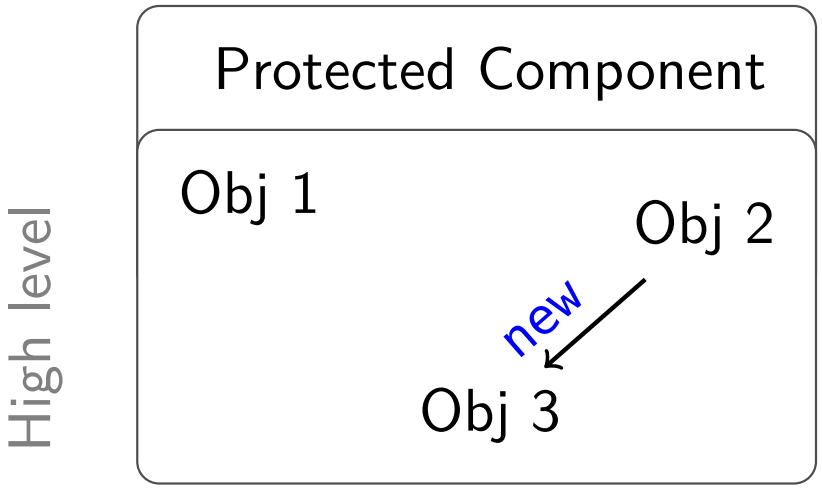
Outline

Secure Compilation, Informally

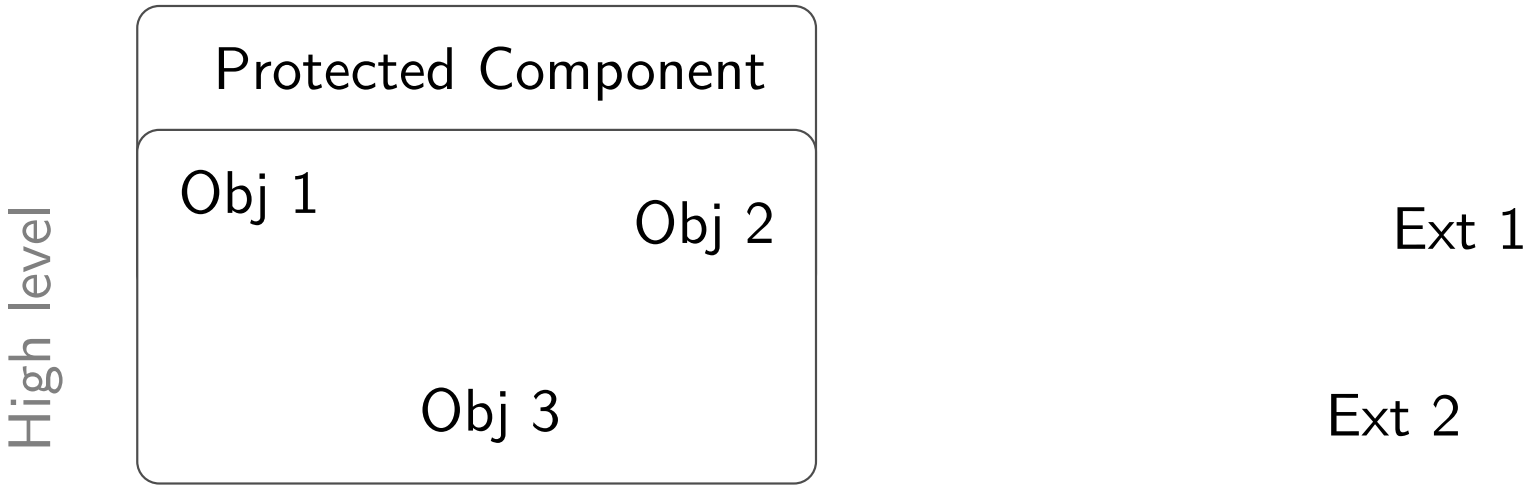
High level



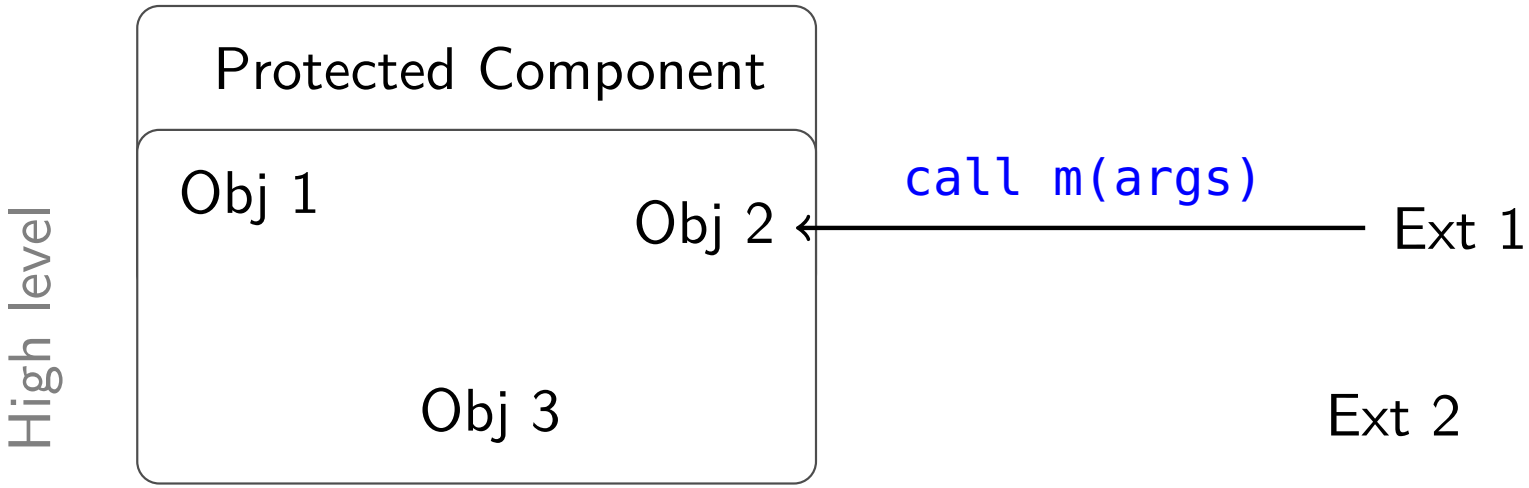
Secure Compilation, Informally



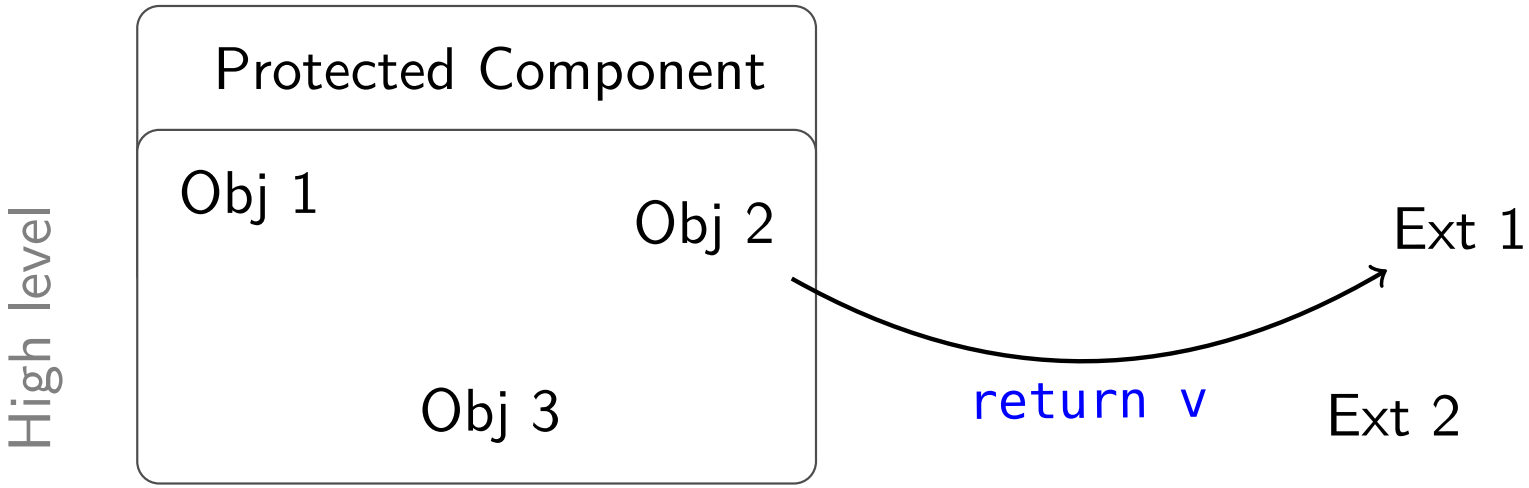
Secure Compilation, Informally



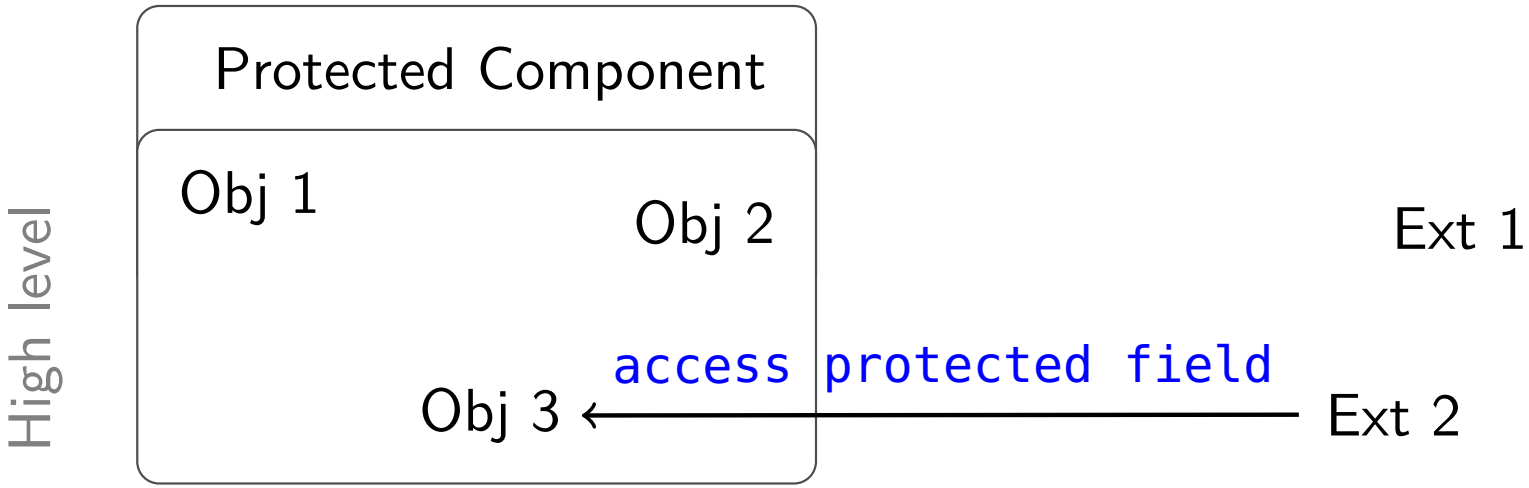
Secure Compilation, Informally



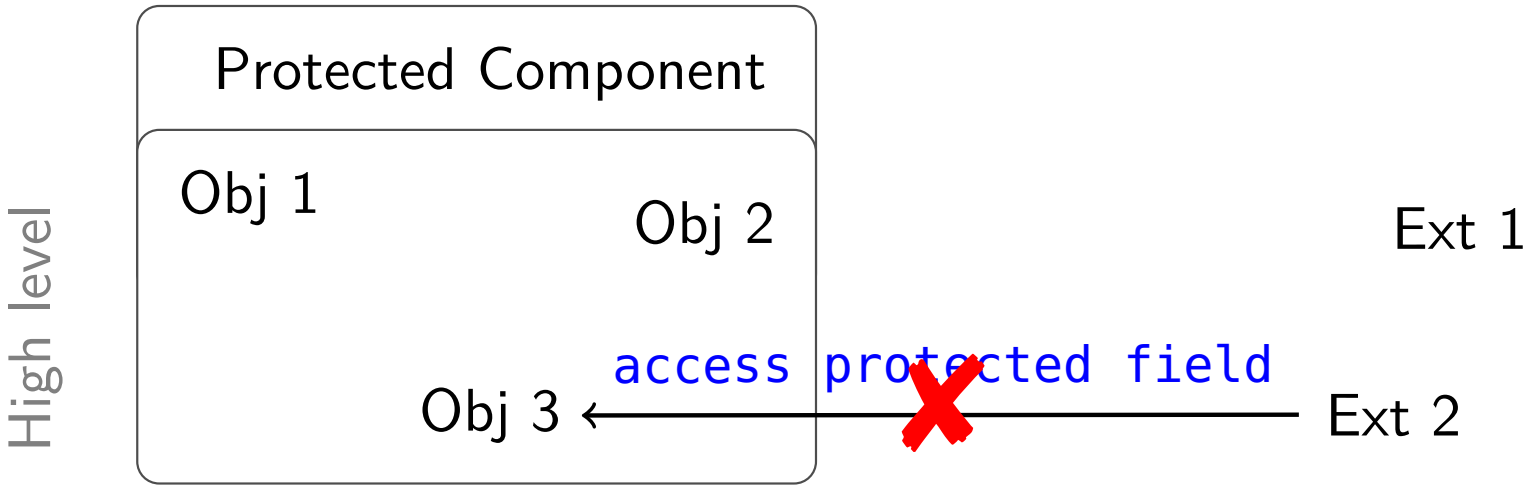
Secure Compilation, Informally



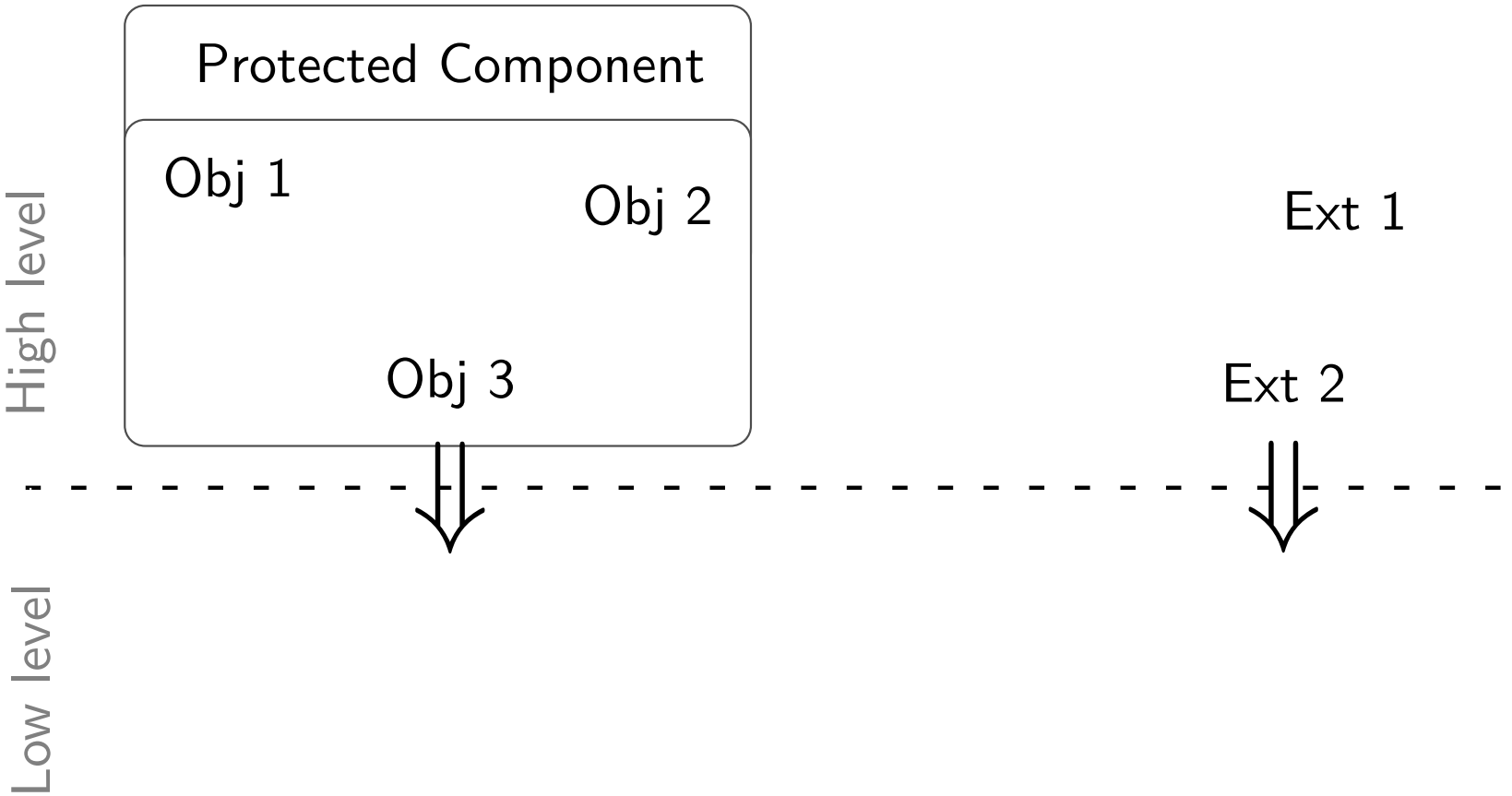
Secure Compilation, Informally



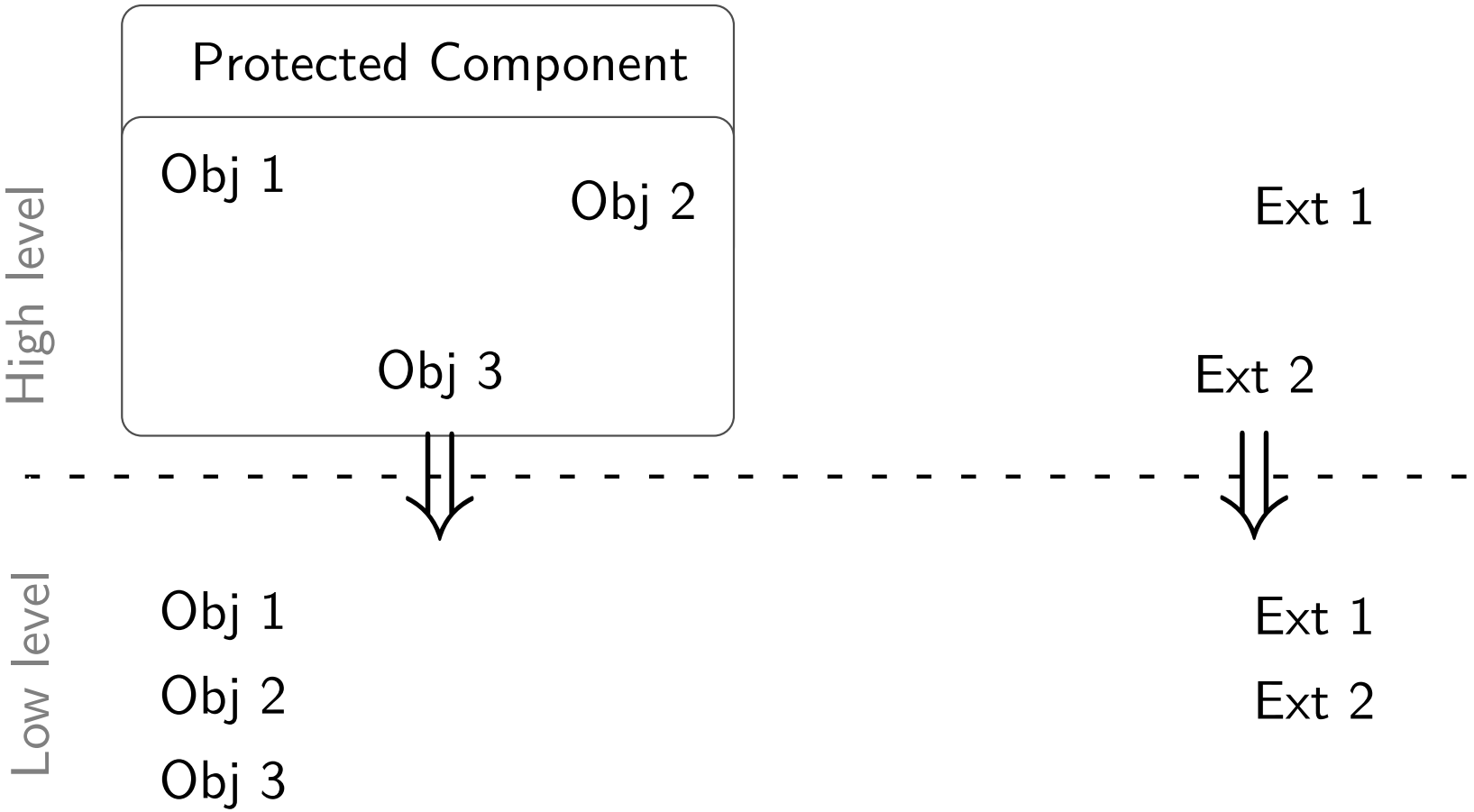
Secure Compilation, Informally



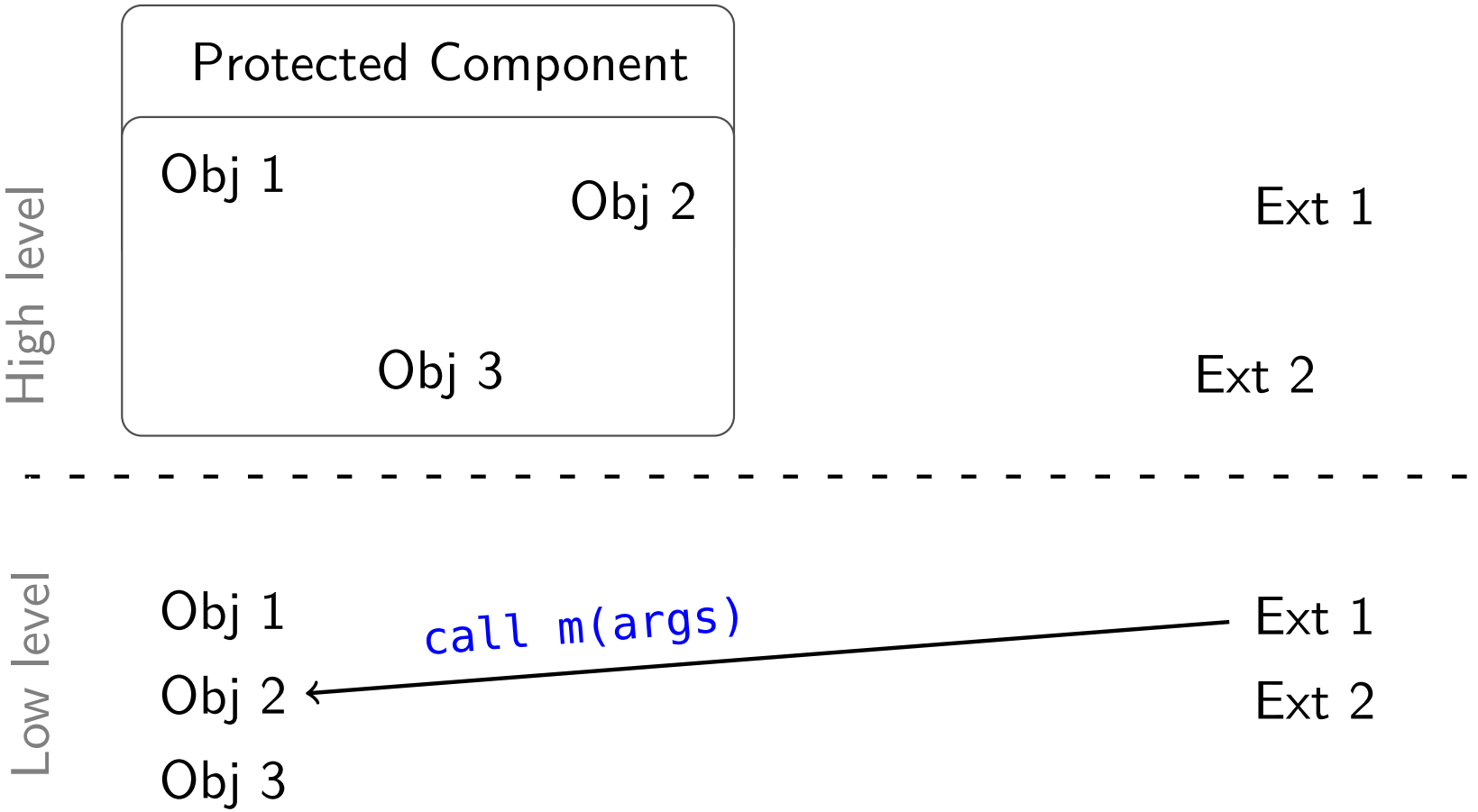
Secure Compilation, Informally



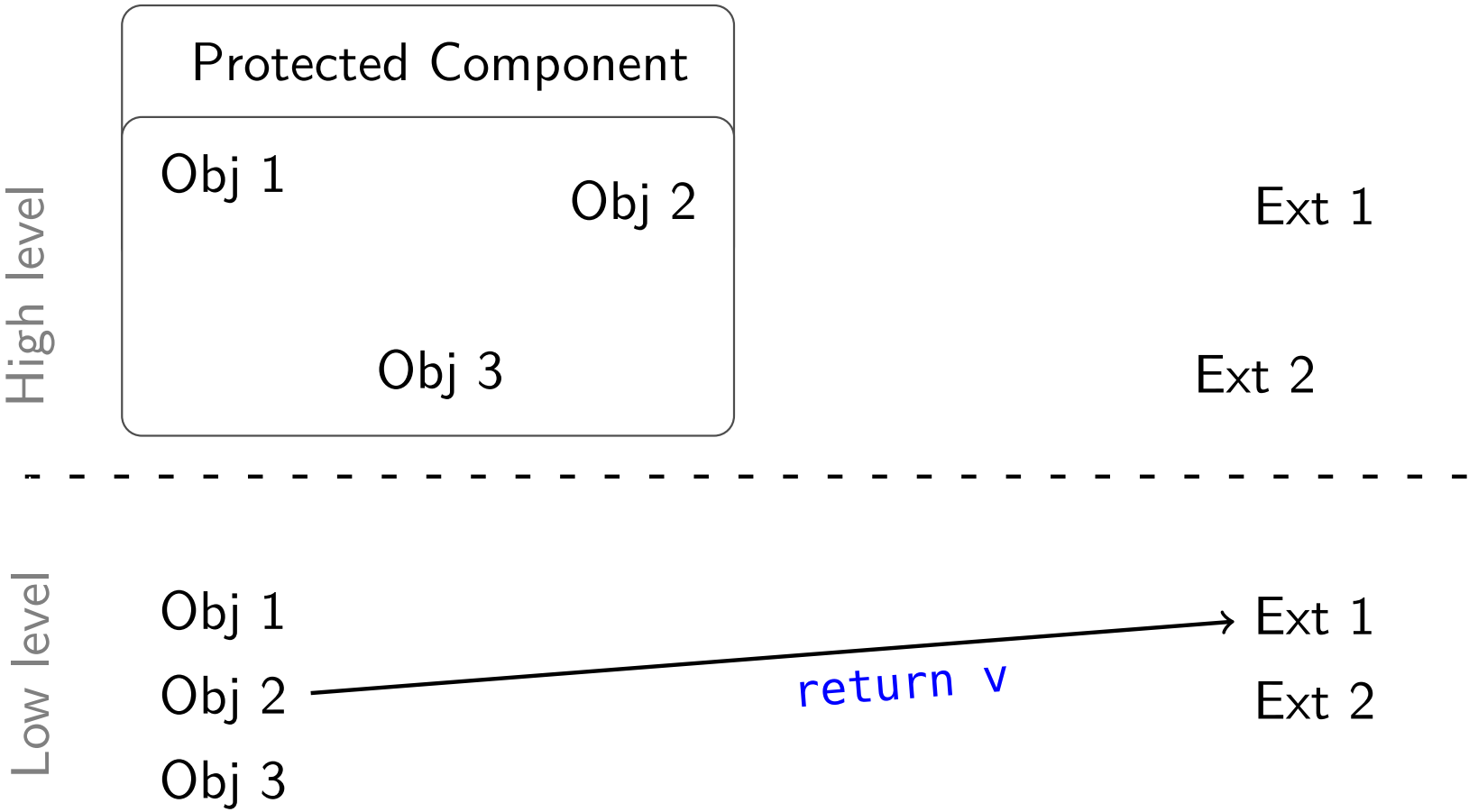
Secure Compilation, Informally



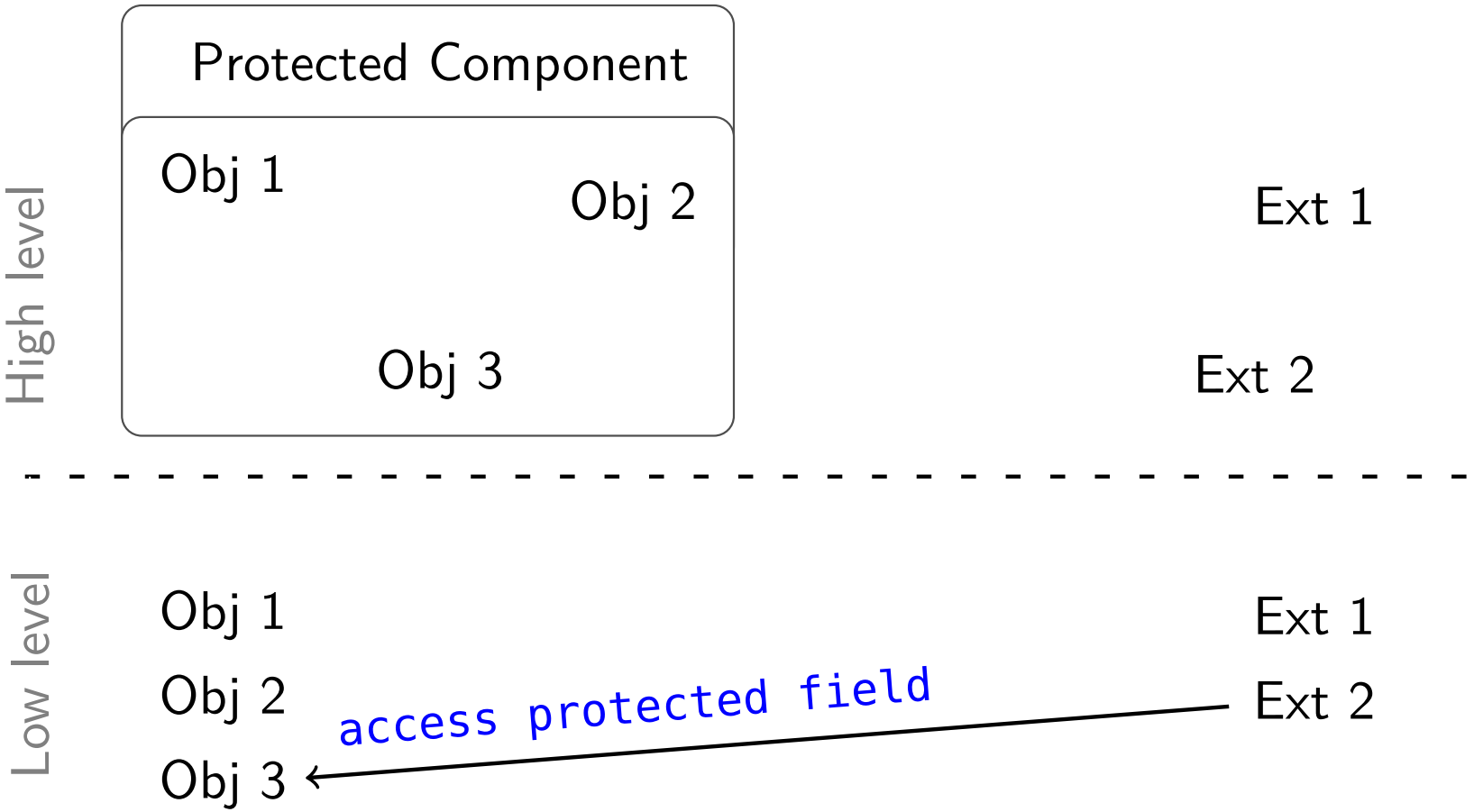
Secure Compilation, Informally



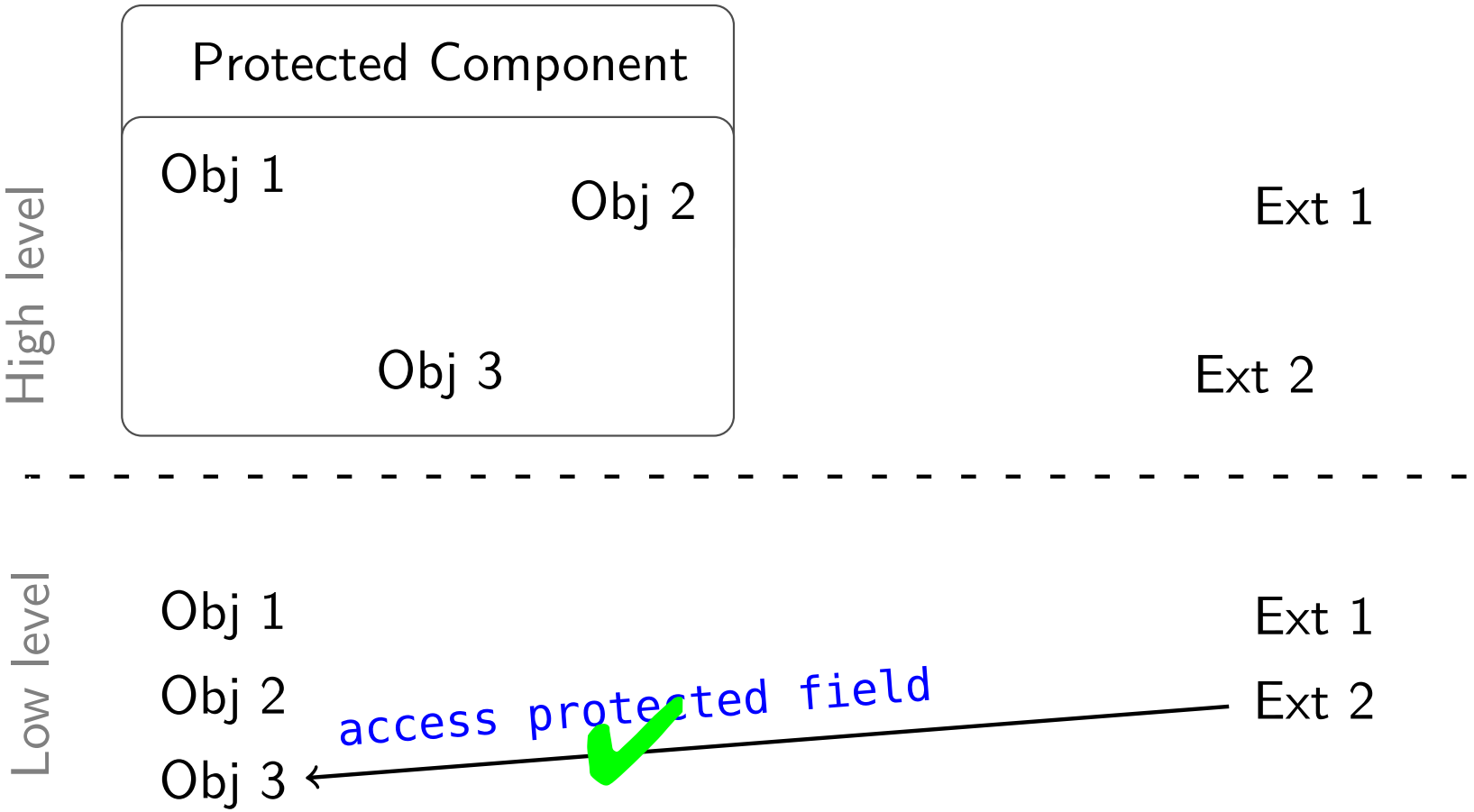
Secure Compilation, Informally



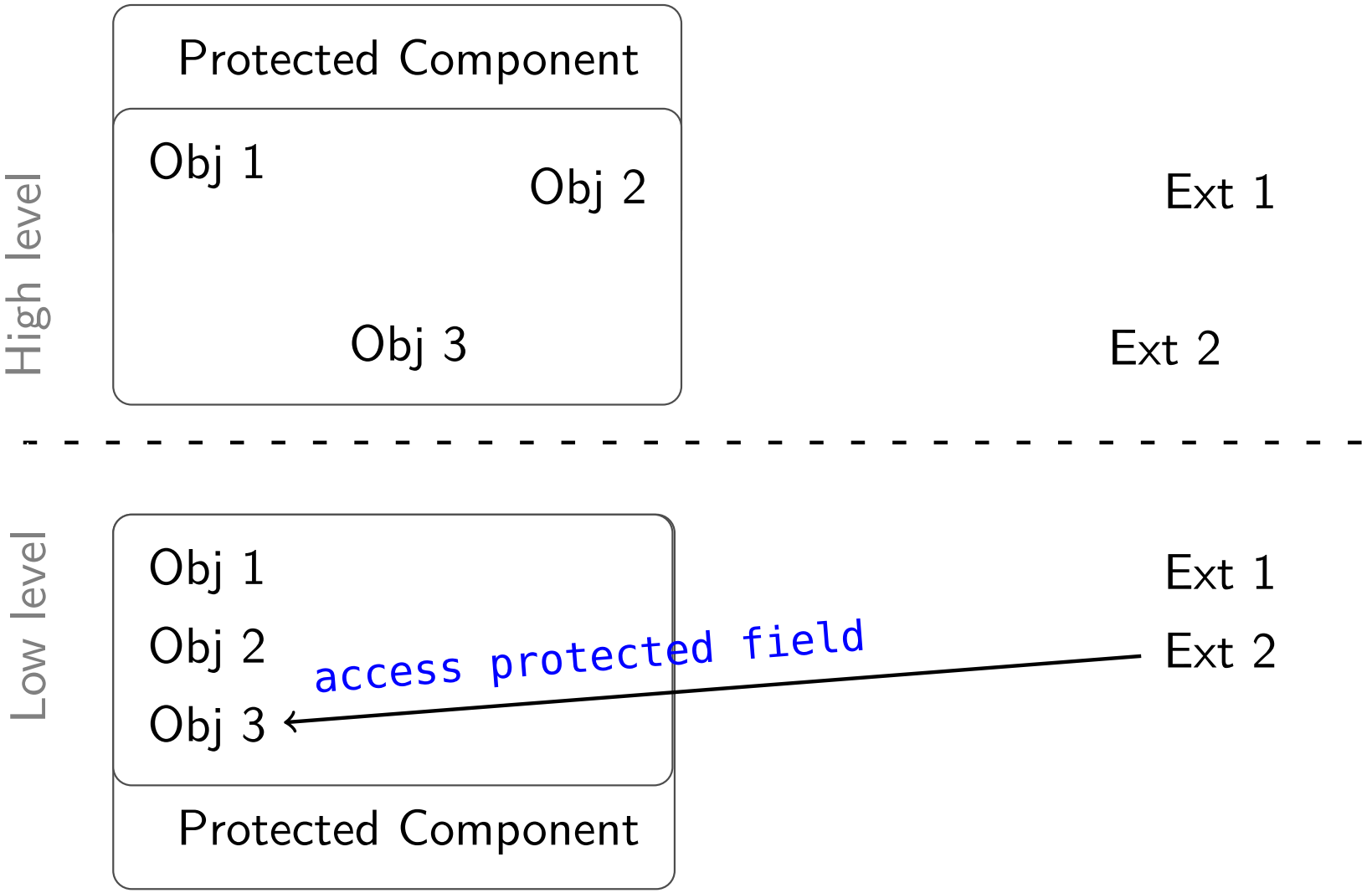
Secure Compilation, Informally



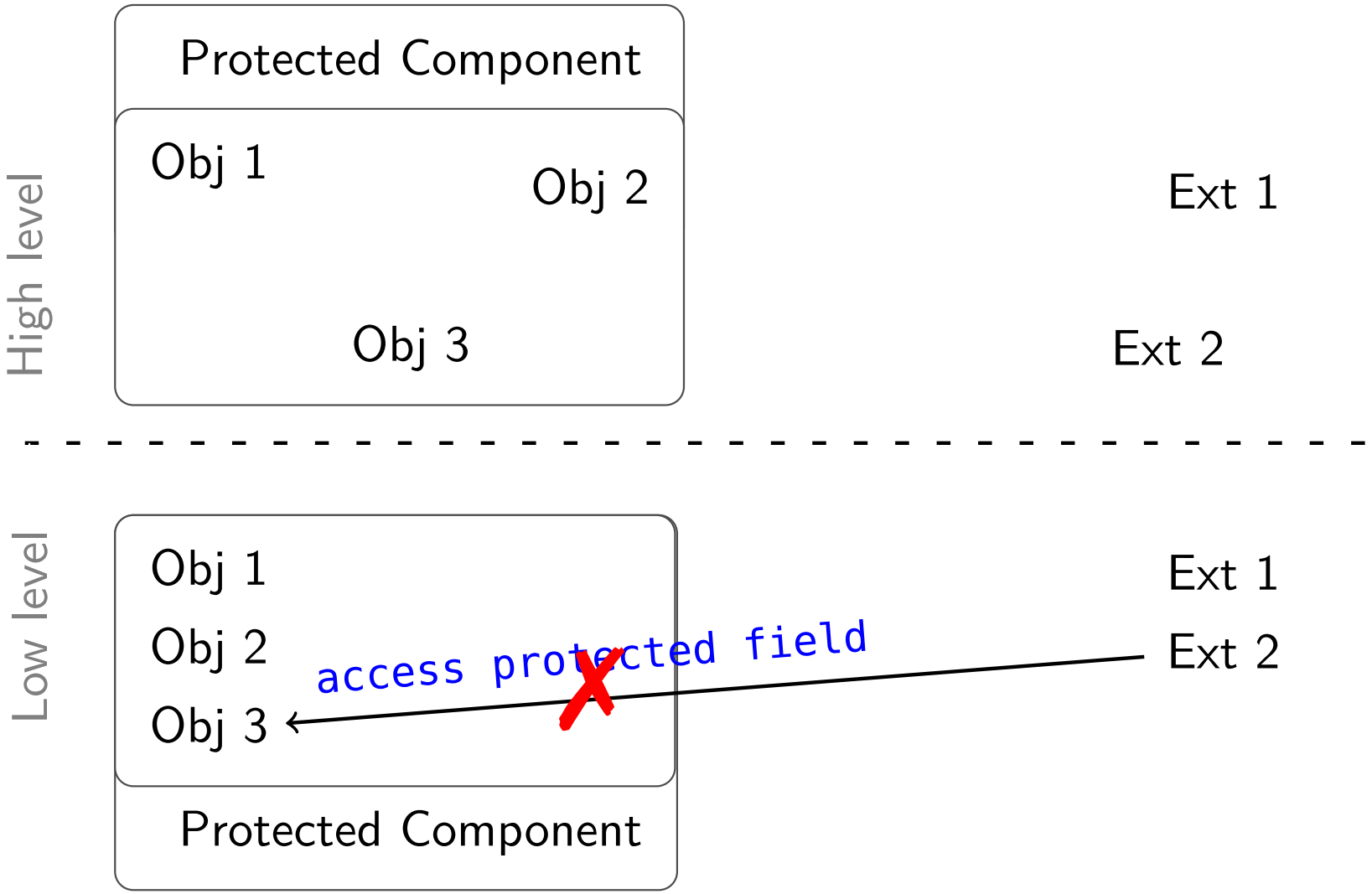
Secure Compilation, Informally



Secure Compilation, Informally

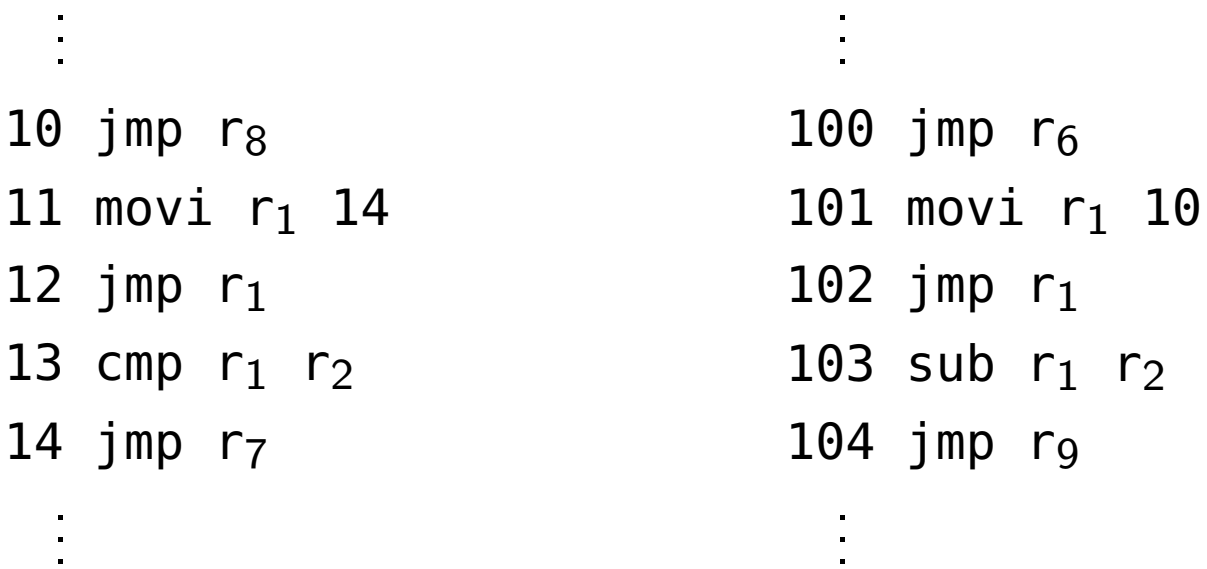


Secure Compilation, Informally

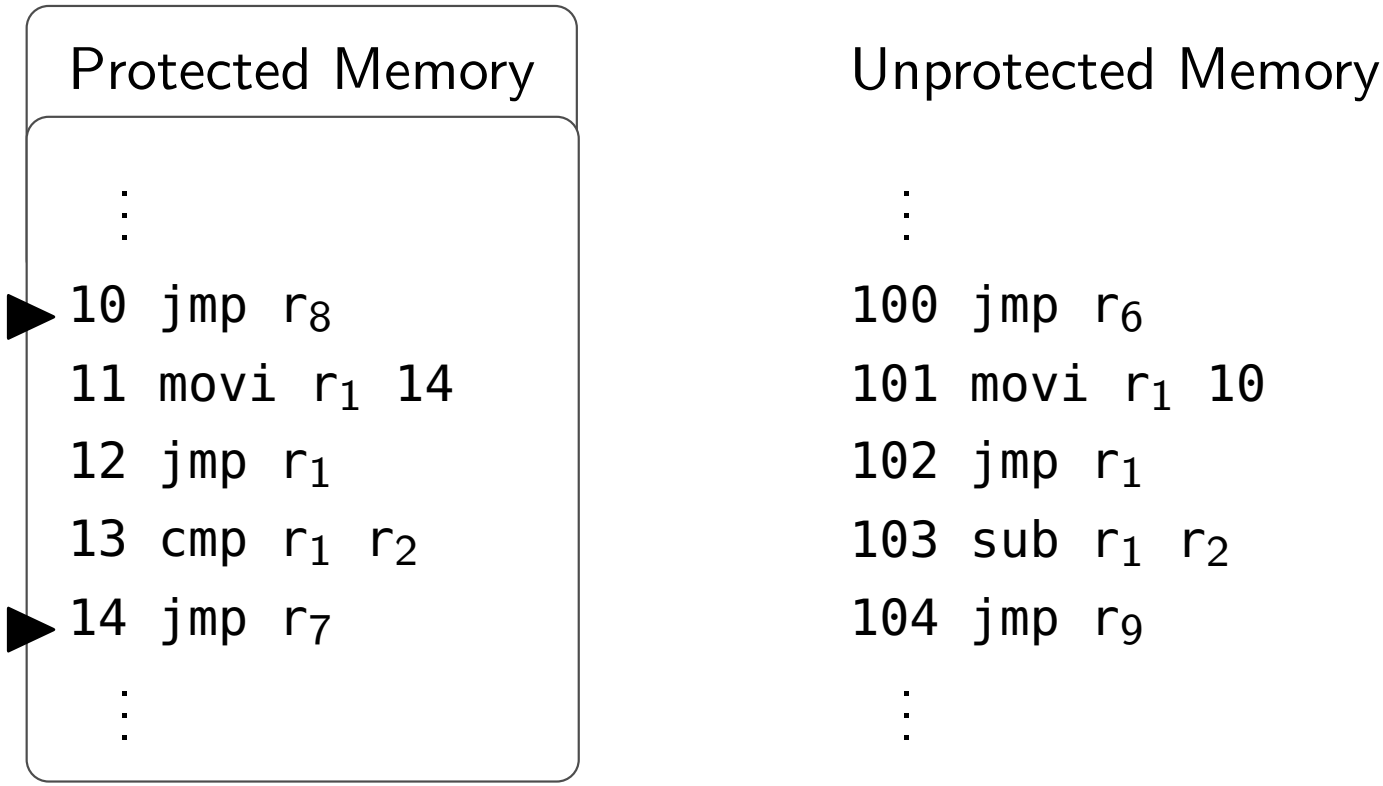


Outline

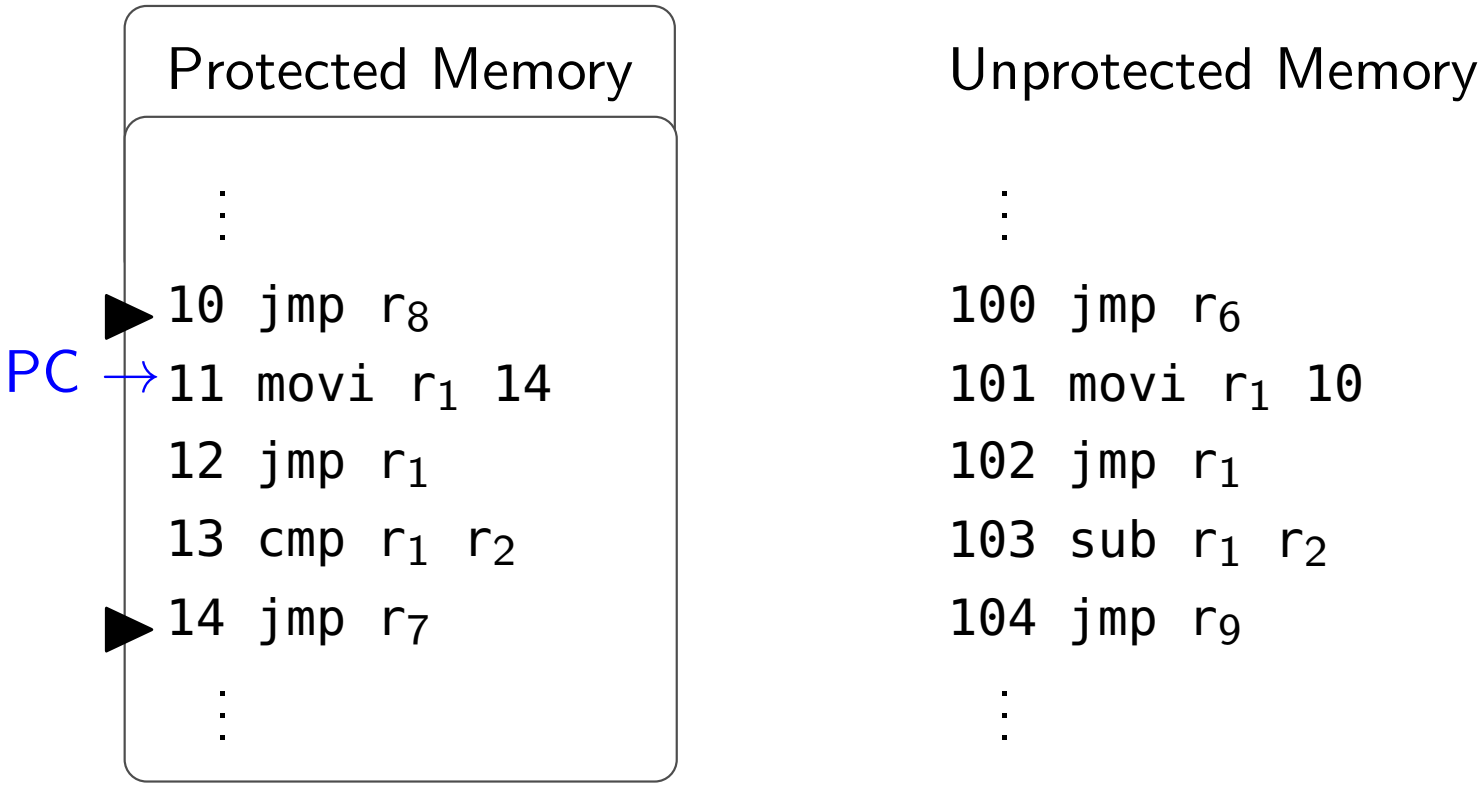
FPMAC


10 jmp r8
11 movi r1 14
12 jmp r1
13 cmp r1 r2
14 jmp r7
100 jmp r6
101 movi r1 10
102 jmp r1
103 sub r1 r2
104 jmp r9

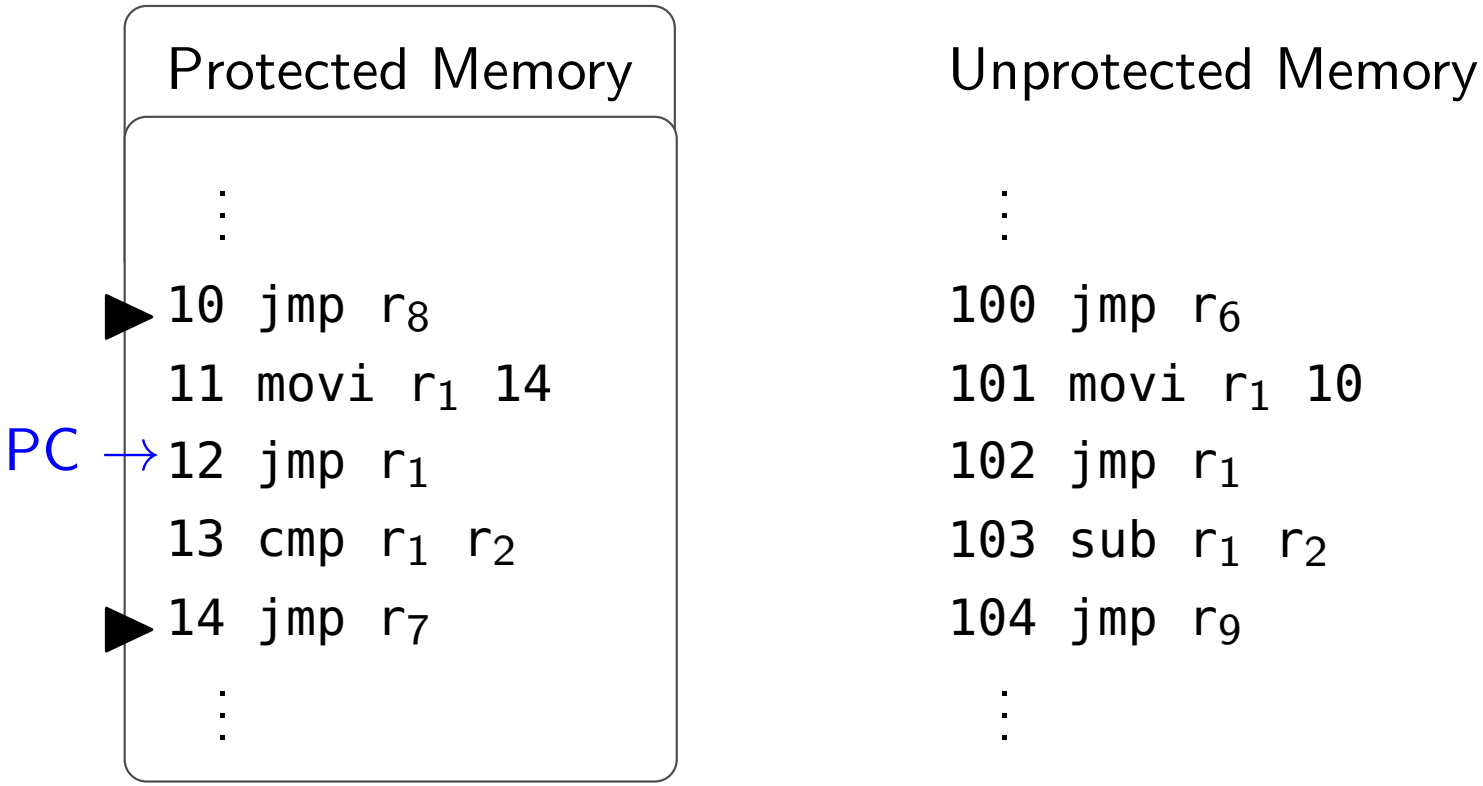
FPMAC


Protected Memory
10 jmp r8
11 movi r1 14
12 jmp r1
13 cmp r1 r2
14 jmp r7
Unprotected Memory
100 jmp r6
101 movi r1 10
102 jmp r1
103 sub r1 r2
104 jmp r9

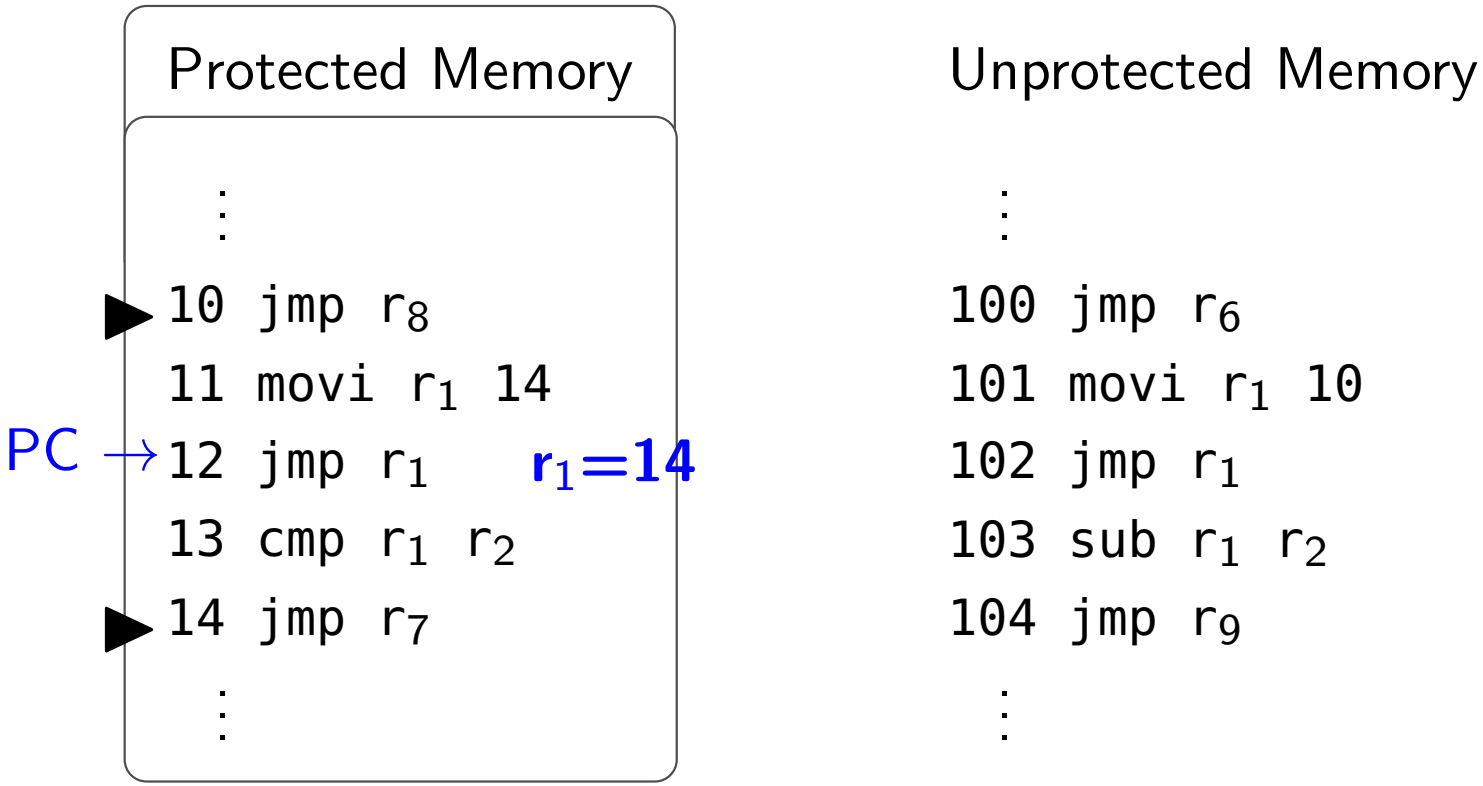
FPMAC


Protected Memory
10 jmp r8
PC →
11 movi r1 14
12 jmp r1
13 cmp r1 r2
14 jmp r7
Unprotected Memory
100 jmp r6
101 movi r1 10
102 jmp r1
103 sub r1 r2
104 jmp r9

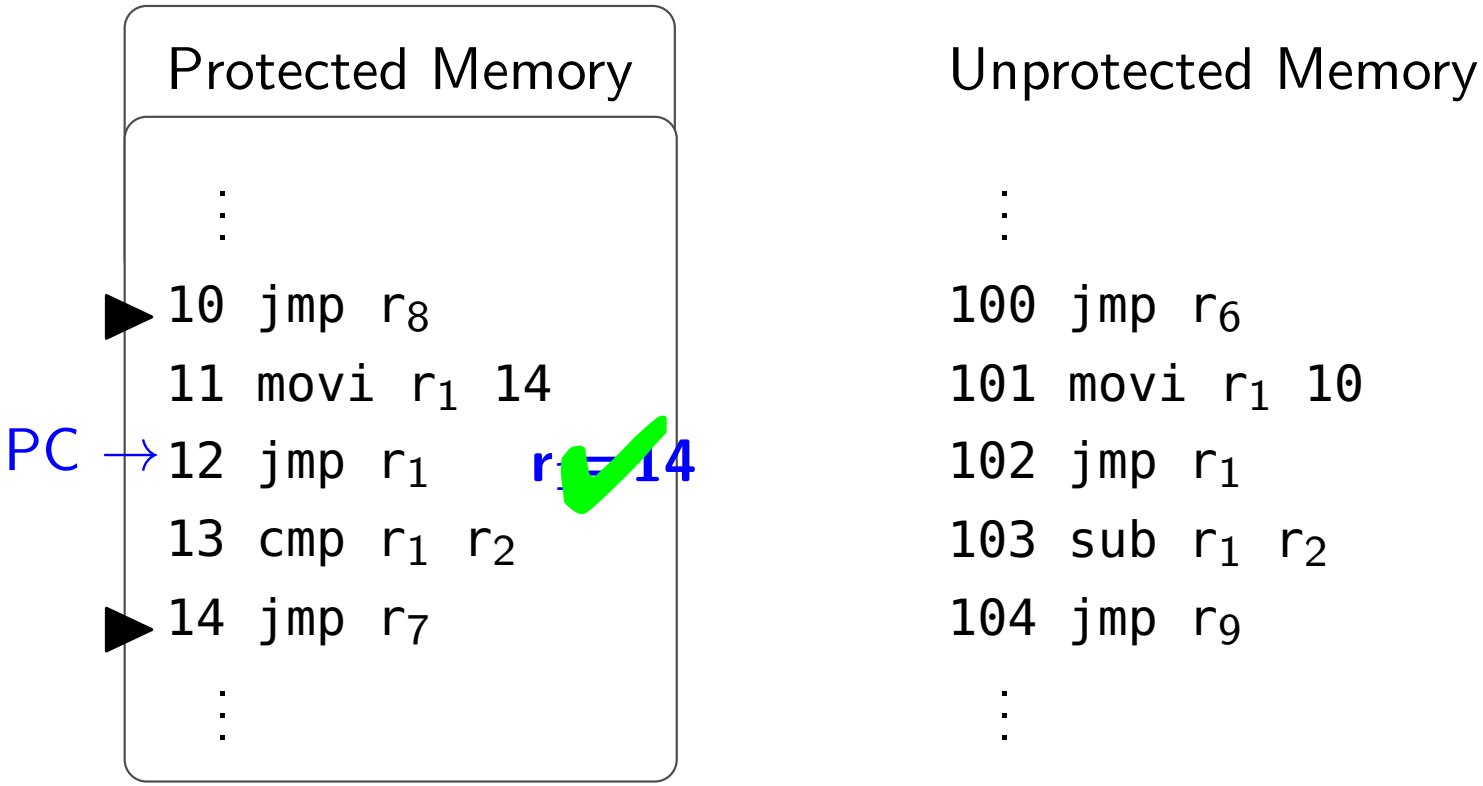
FPMAC


Protected Memory
10 jmp r8
11 movi r1 14
PC →
12 jmp r1
13 cmp r1 r2
14 jmp r7
Unprotected Memory
100 jmp r6
101 movi r1 10
102 jmp r1
103 sub r1 r2
104 jmp r9

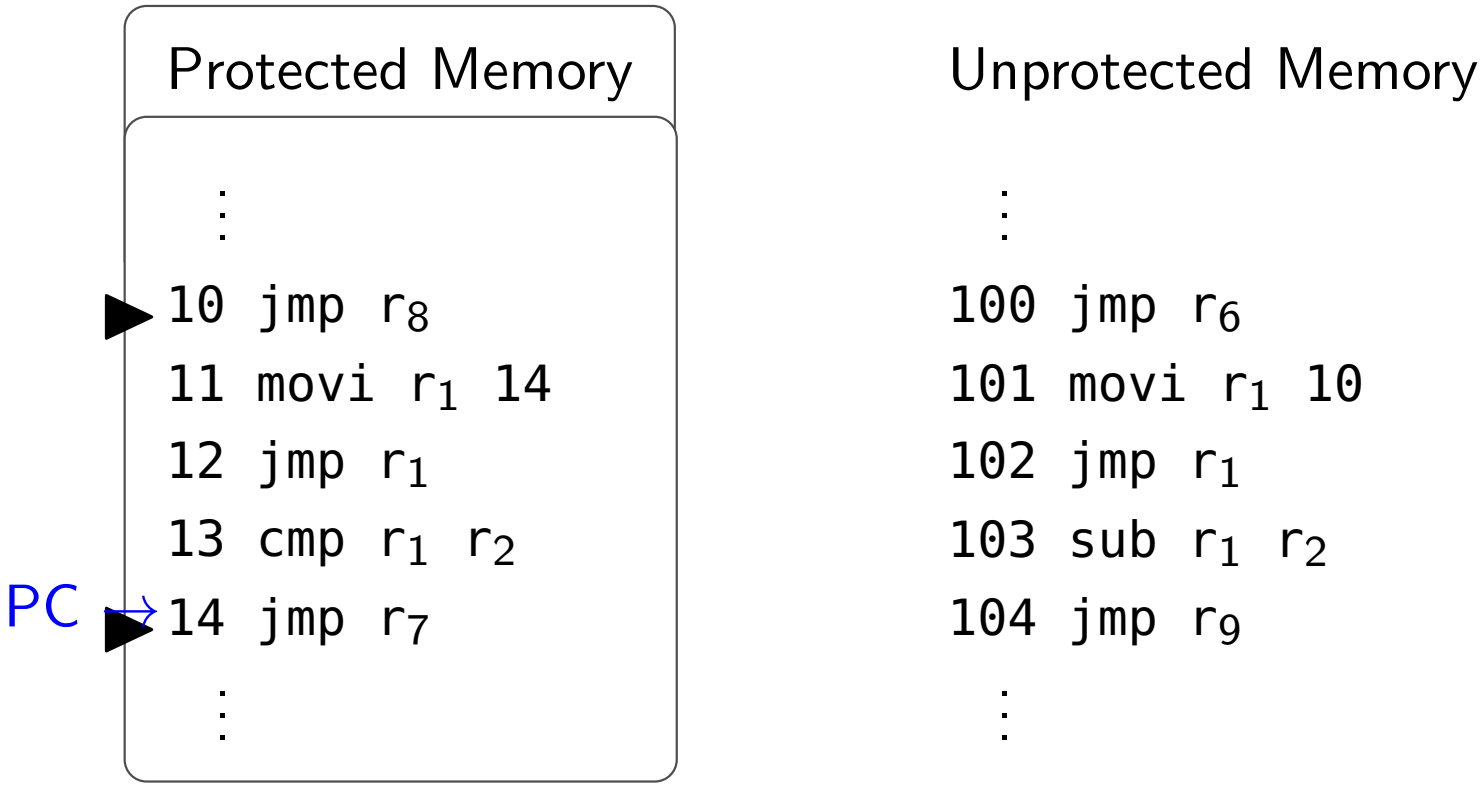
FPMAC


Protected Memory
10 jmp r8
11 movi r1 14
PC →12 jmp r1
r1=14
13 cmp r1 r2
14 jmp r7
Unprotected Memory
100 jmp r6
101 movi r1 10
102 jmp r1
103 sub r1 r2
104 jmp r9

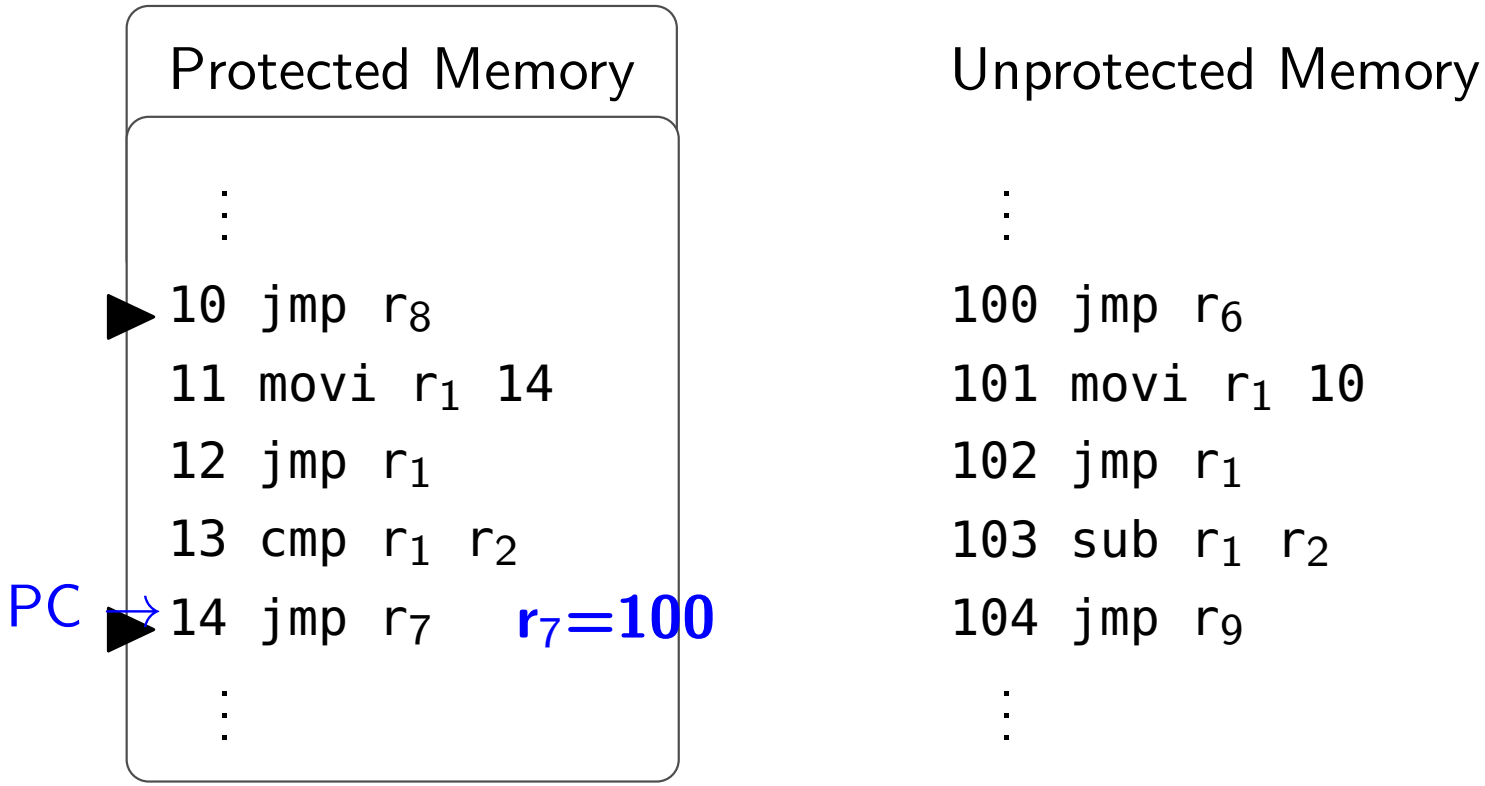
FPMAC


Protected Memory
10 jmp r8
11 movi r1 14
PC →
12 jmp r1
13 cmp r1 r2
14 jmp r7
Unprotected Memory
100 jmp r6
101 movi r1 10
102 jmp r1
103 sub r1 r2
104 jmp r9

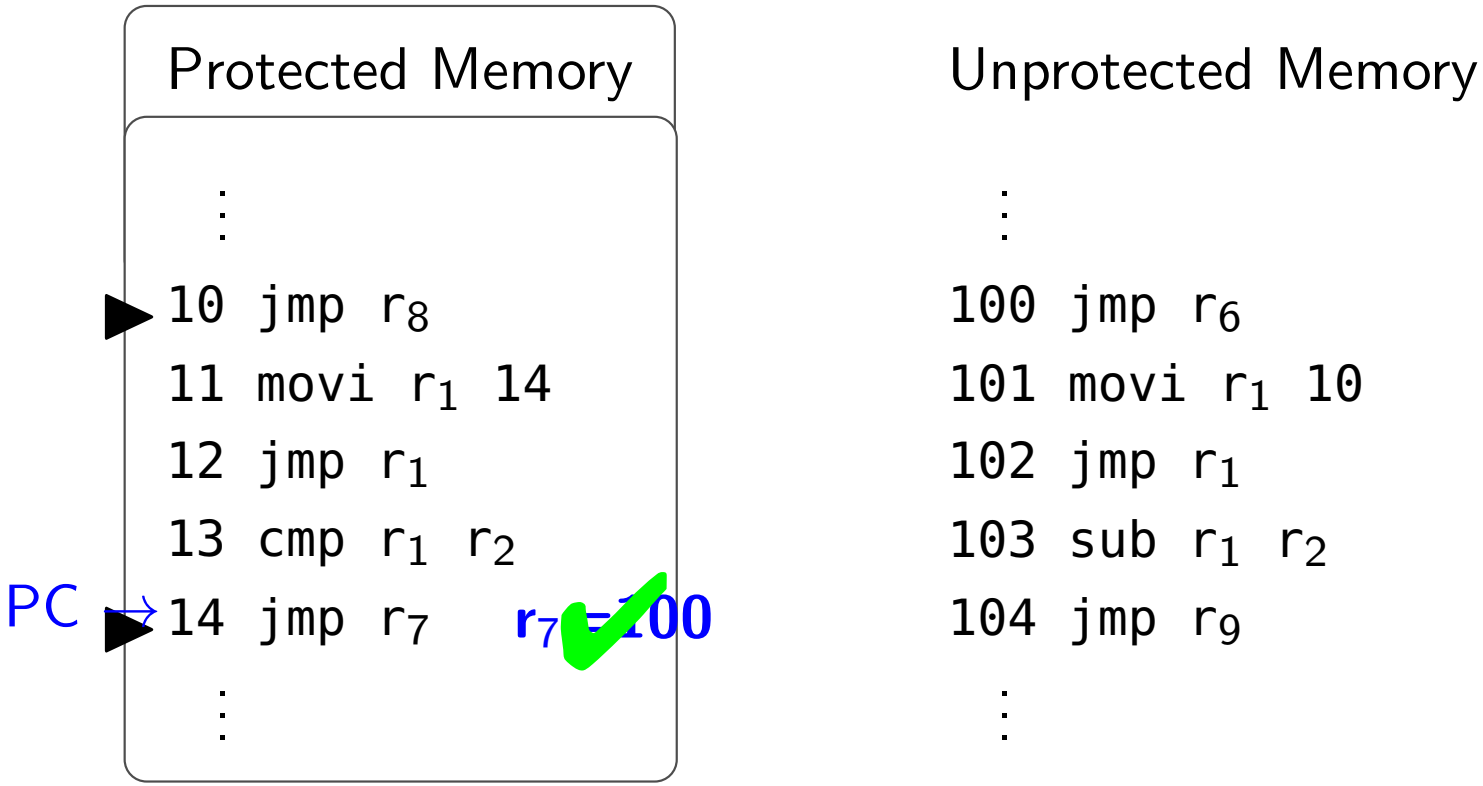
FPMAC


Protected Memory
10 jmp r8
11 movi r1 14
12 jmp r1
13 cmp r1 r2
PC
14 jmp r7
Unprotected Memory
100 jmp r6
101 movi r1 10
102 jmp r1
103 sub r1 r2
104 jmp r9

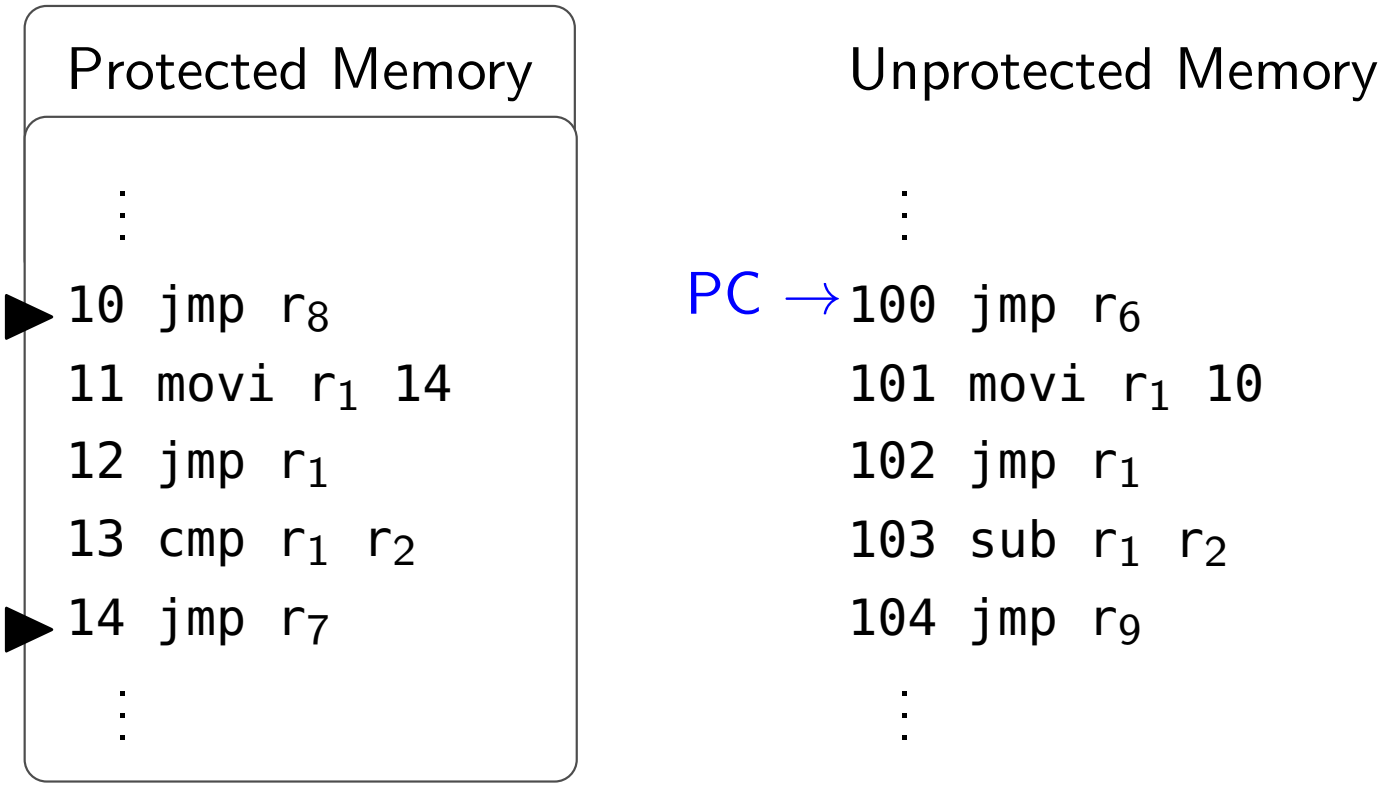
FPMAC


Protected Memory
10 jmp r8
11 movi r1 14
12 jmp r1
13 cmp r1 r2
PC
14 jmp r7
r7=100
Unprotected Memory
100 jmp r6
101 movi r1 10
102 jmp r1
103 sub r1 r2
104 jmp r9

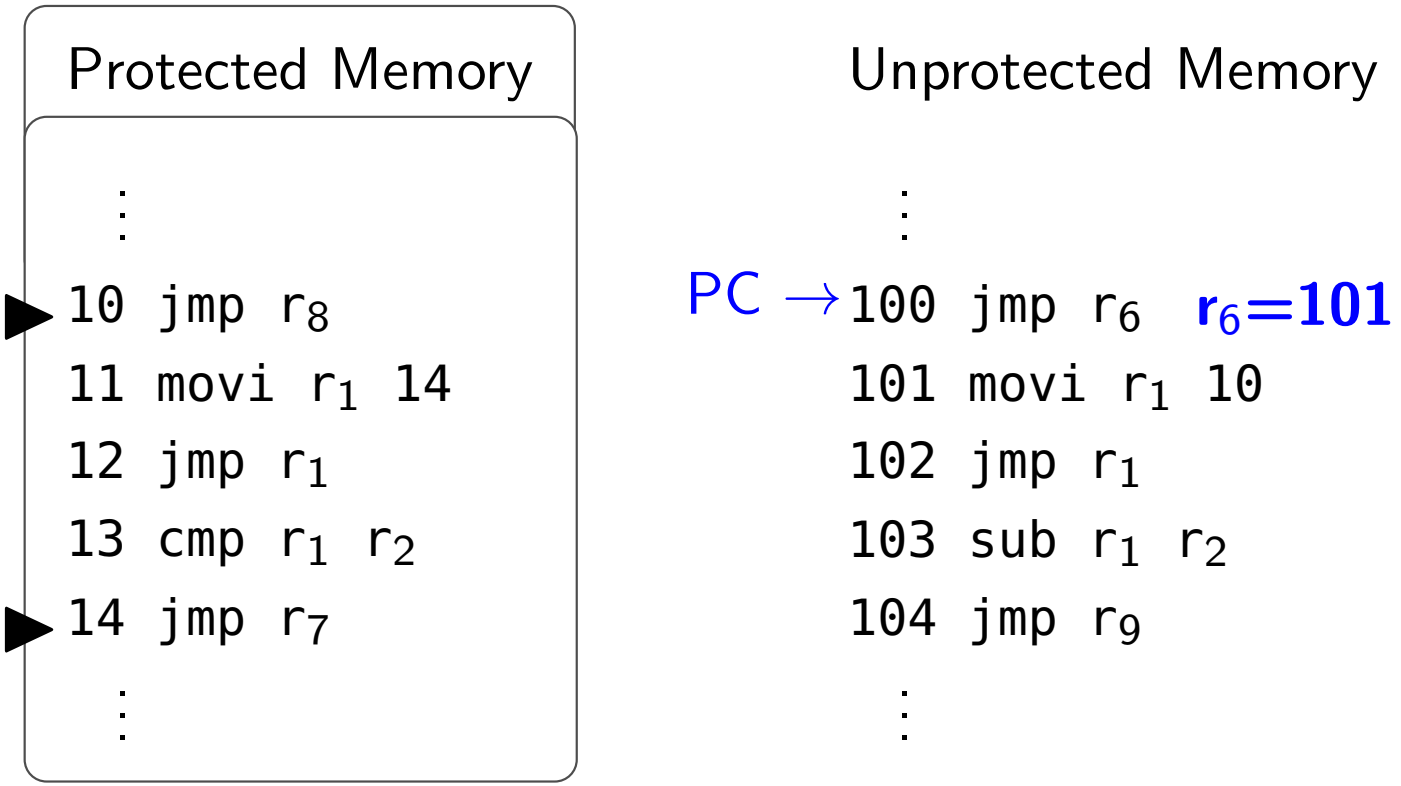
FPMAC


Protected Memory
10 jmp r8
11 movi r1 14
12 jmp r1
13 cmp r1 r2
PC
14 jmp r7
r7 = 100
Unprotected Memory
100 jmp r6
101 movi r1 10
102 jmp r1
103 sub r1 r2
104 jmp r9

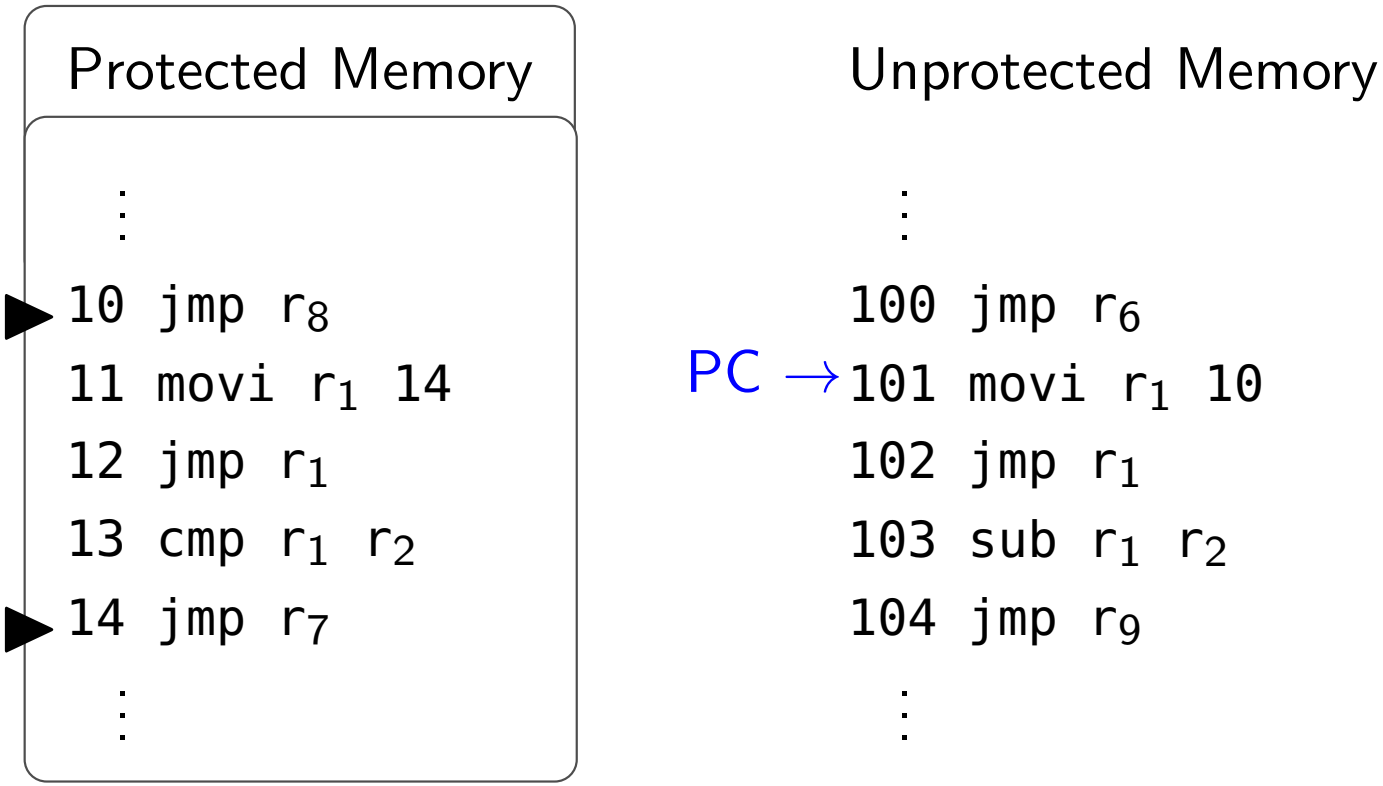
FPMAC


Protected Memory
10 jmp r8
11 movi r1 14
12 jmp r1
13 cmp r1 r2
14 jmp r7
Unprotected Memory
PC →
100 jmp r6
101 movi r1 10
102 jmp r1
103 sub r1 r2
104 jmp r9

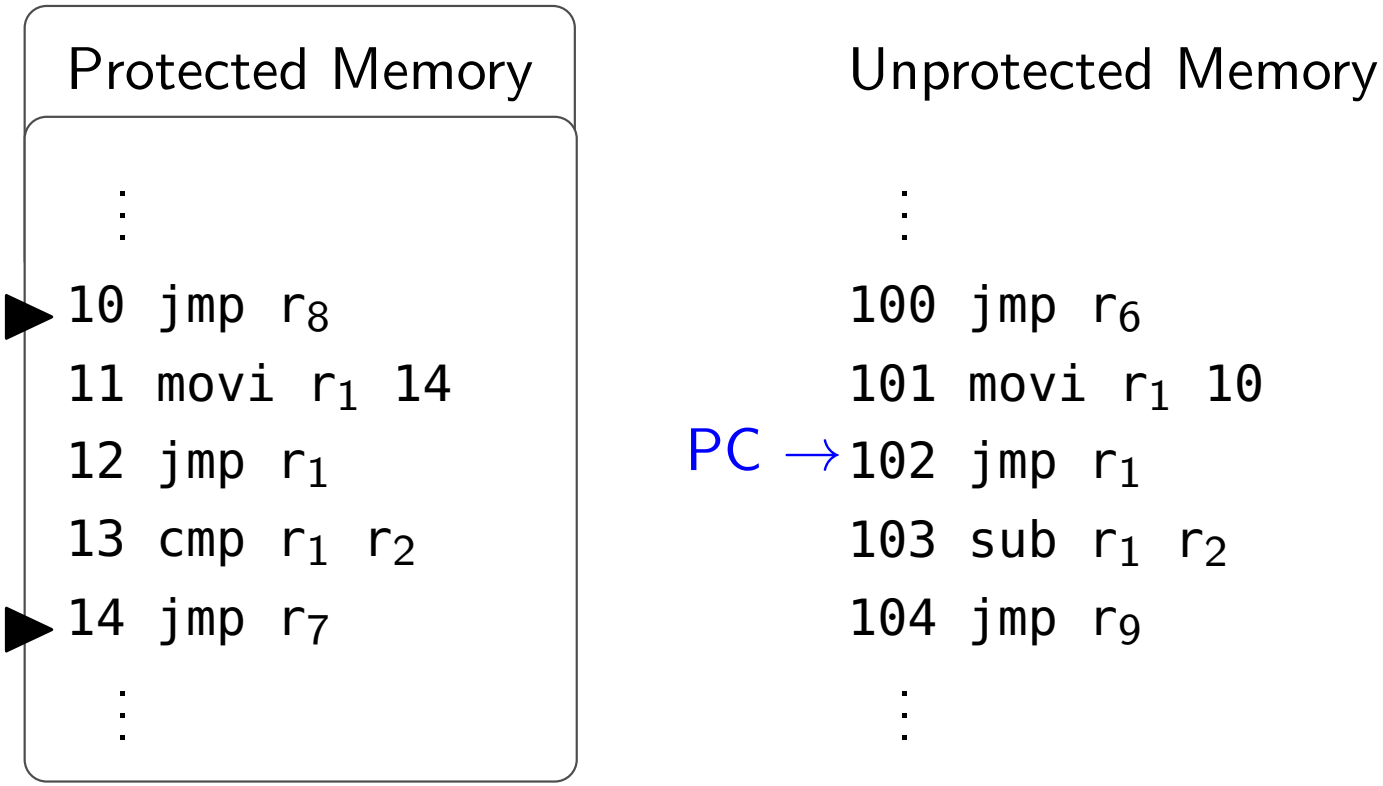
FPMAC


Protected Memory
10 jmp r8
11 movi r1 14
12 jmp r1
13 cmp r1 r2
14 jmp r7
Unprotected Memory
PC →
100 jmp r6 r6=101
101 movi r1 10
102 jmp r1
103 sub r1 r2
104 jmp r9

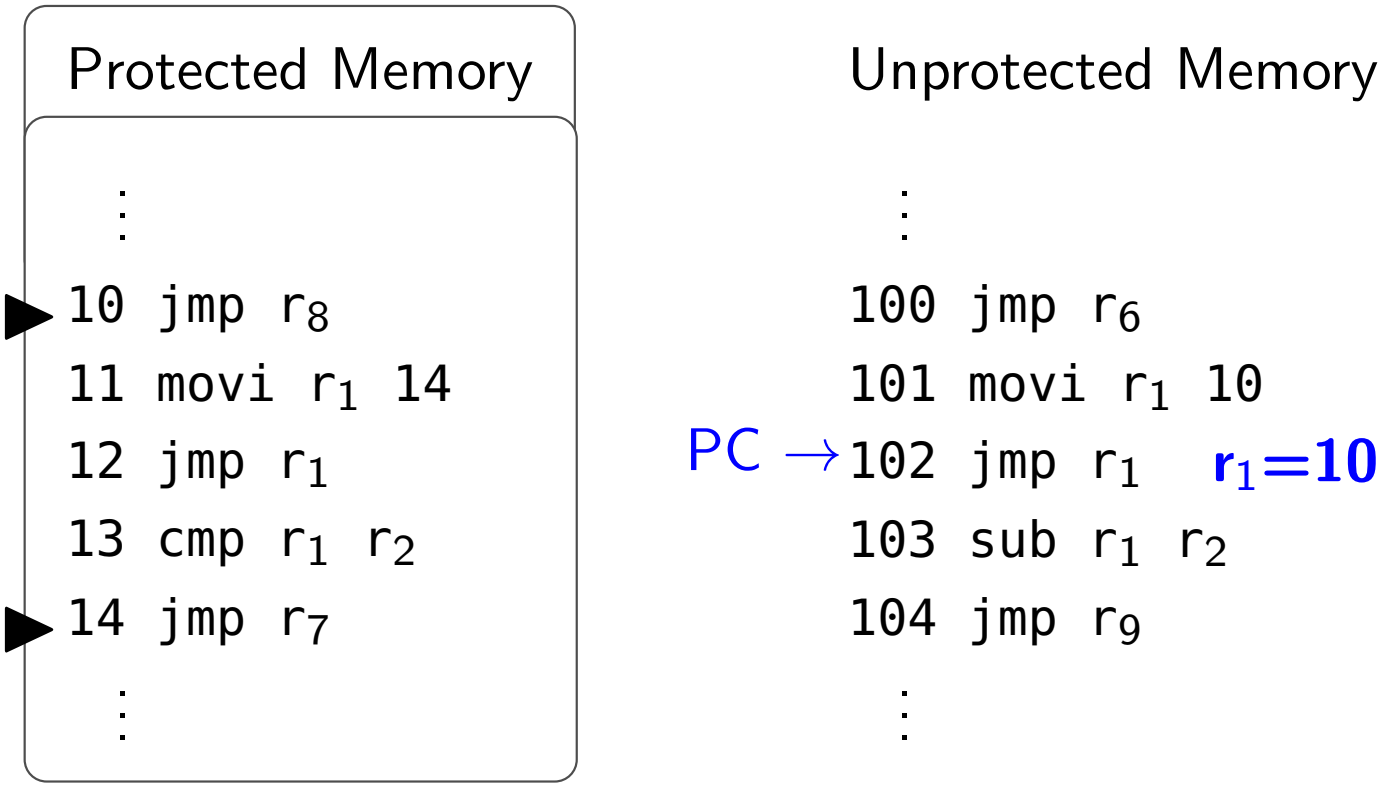
FPMAC


Protected Memory
10 jmp r8
11 movi r1 14
12 jmp r1
13 cmp r1 r2
14 jmp r7
Unprotected Memory
100 jmp r6
PC →101 movi r1 10
102 jmp r1
103 sub r1 r2
104 jmp r9

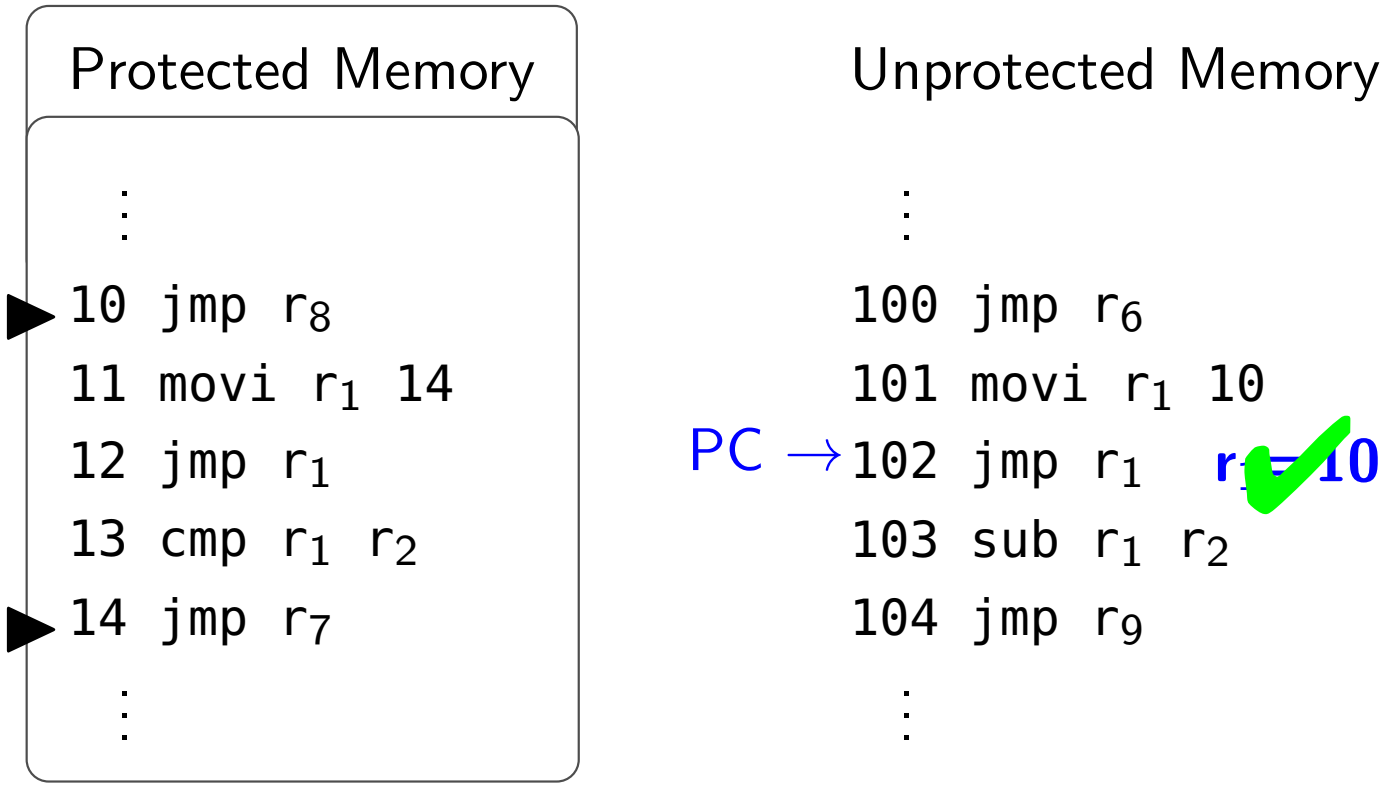
FPMAC


Protected Memory
10 jmp r8
11 movi r1 14
12 jmp r1
13 cmp r1 r2
14 jmp r7
Unprotected Memory
100 jmp r6
101 movi r1 10
PC →102 jmp r1
103 sub r1 r2
104 jmp r9

FPMAC

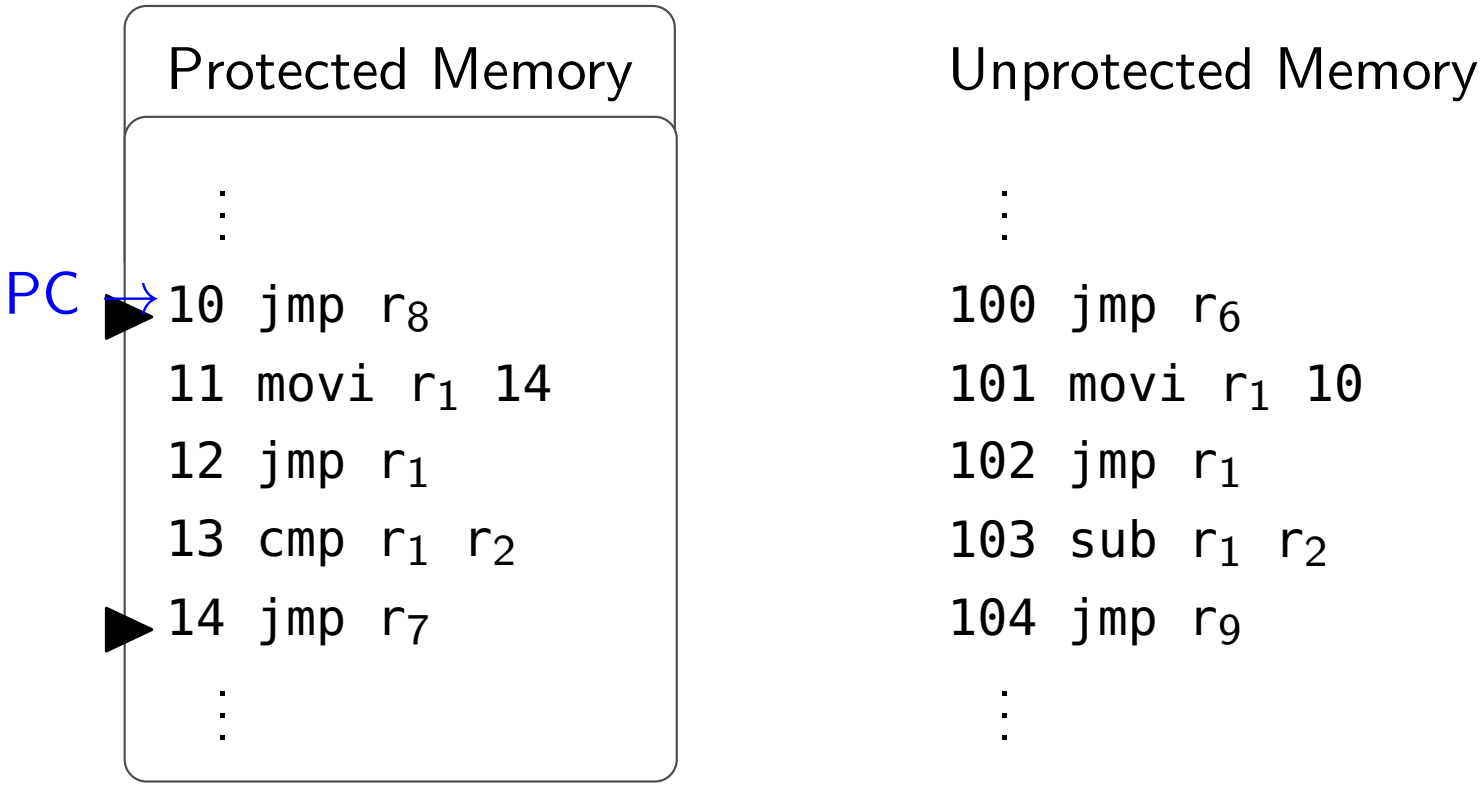

Protected Memory
10 jmp r8
11 movi r1 14
12 jmp r1
13 cmp r1 r2
14 jmp r7
Unprotected Memory
100 jmp r6
101 movi r1 10
PC →102 jmp r1
r1=10
103 sub r1 r2
104 jmp r9

FPMAC

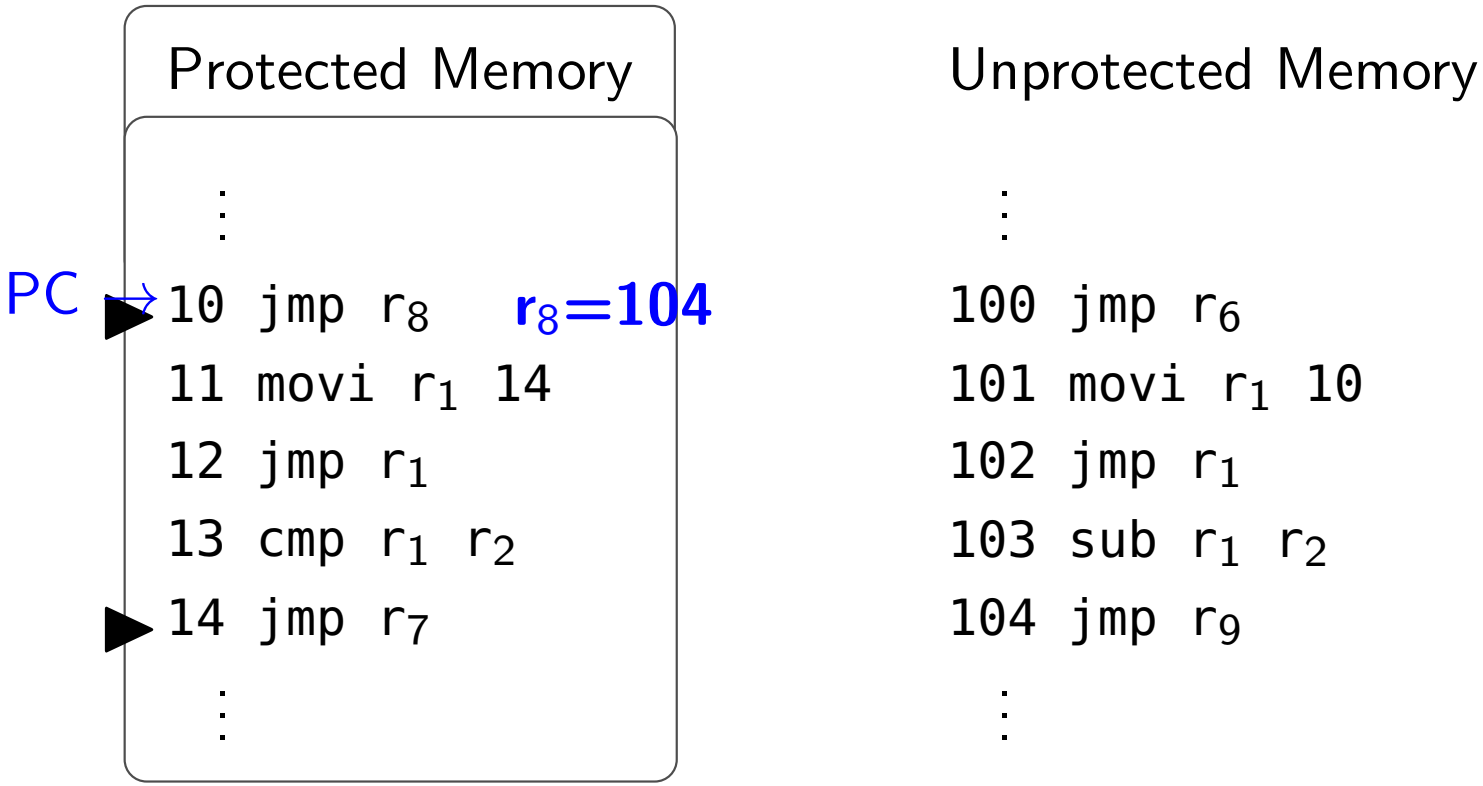


10 is an entry point

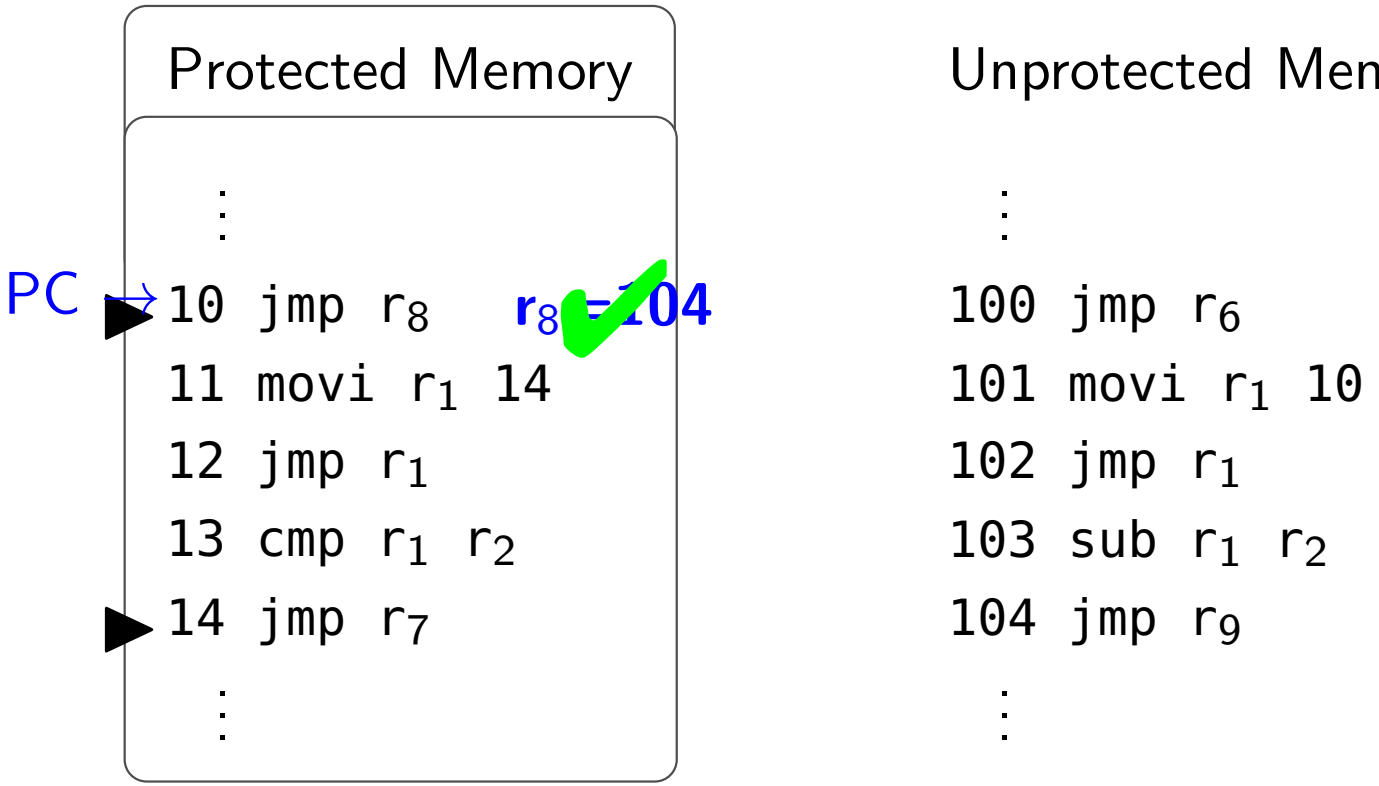
FPMAC


Protected Memory
PC
10 jmp r8
11 movi r1 14
12 jmp r1
13 cmp r1 r2
14 jmp r7
Unprotected Memory
100 jmp r6
101 movi r1 10
102 jmp r1
103 sub r1 r2
104 jmp r9

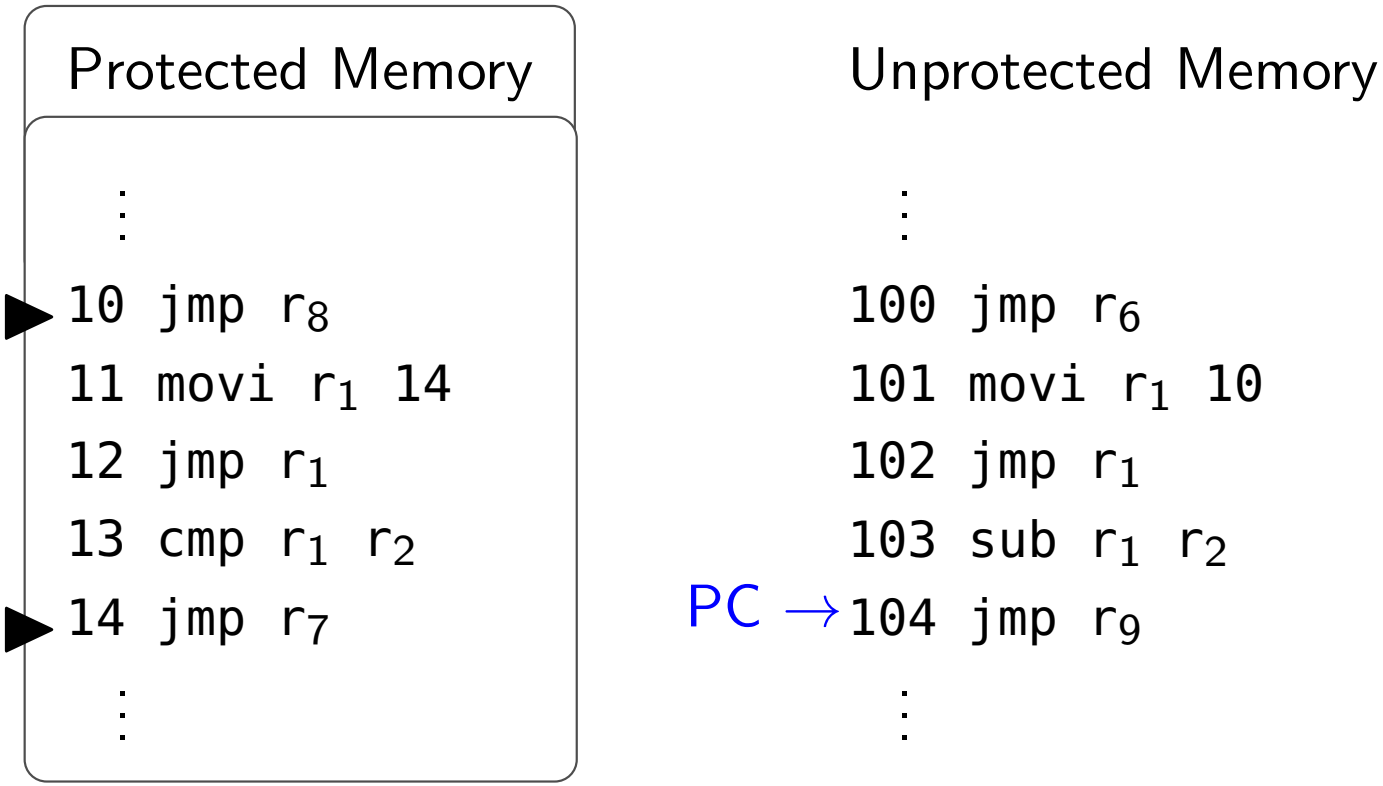
FPMAC


Protected Memory
PC
10 jmp r8
r8=104
11 movi r1 14
12 jmp r1
13 cmp r1 r2
14 jmp r7
Unprotected Memory
100 jmp r6
101 movi r1 10
102 jmp r1
103 sub r1 r2
104 jmp r9

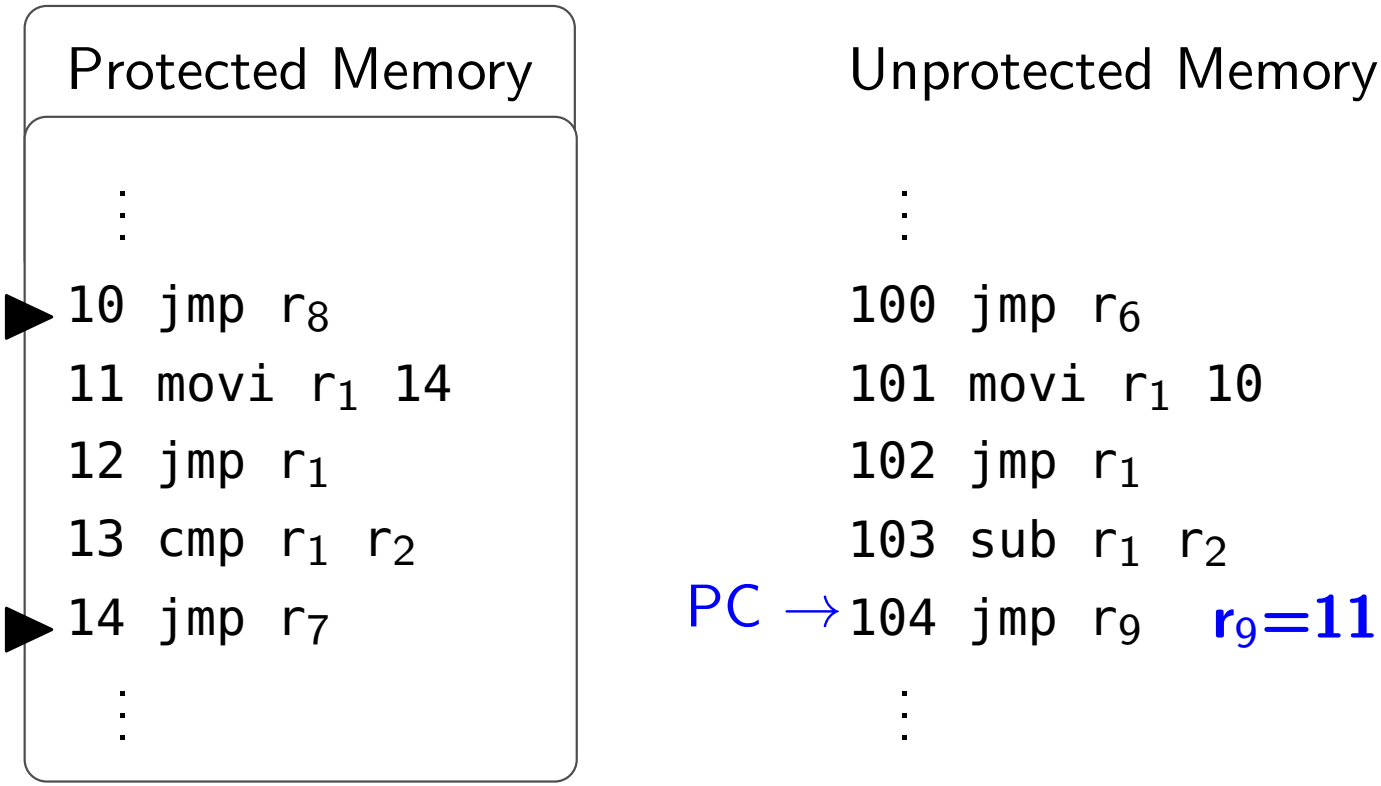
FPMAC


Protected Memory
PC
10 jmp r8
11 movi r1 14
12 jmp r1
13 cmp r1 r2
14 jmp r7
r8 = 104
Unprotected Memory
100 jmp r6
101 movi r1 10
102 jmp r1
103 sub r1 r2
104 jmp r9

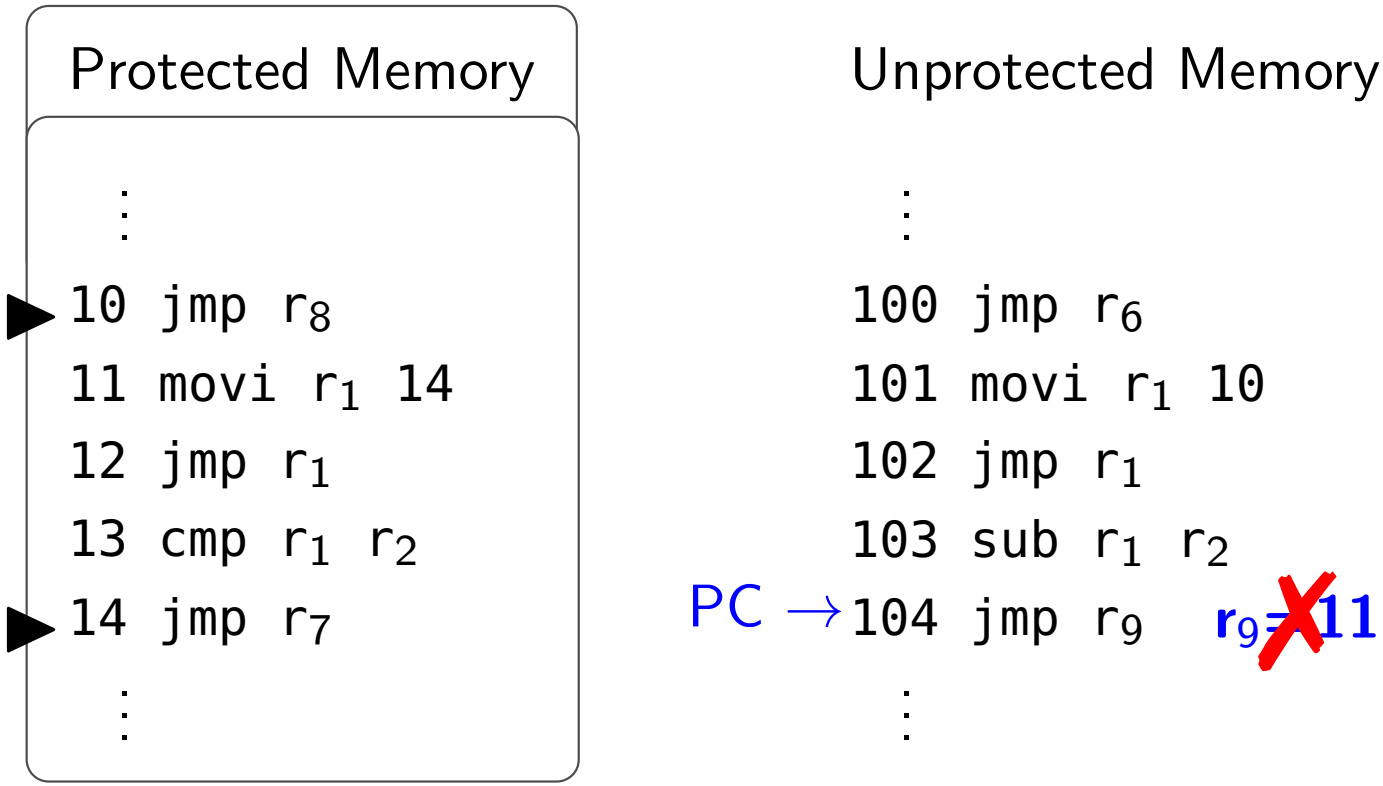
FPMAC


Protected Memory
10 jmp r8
11 movi r1 14
12 jmp r1
13 cmp r1 r2
14 jmp r7
Unprotected Memory
100 jmp r6
101 movi r1 10
102 jmp r1
103 sub r1 r2
PC →
104 jmp r9

FPMAC


Protected Memory
10 jmp r8
11 movi r1 14
12 jmp r1
13 cmp r1 r2
14 jmp r7
Unprotected Memory
100 jmp r6
101 movi r1 10
102 jmp r1
103 sub r1 r2
PC →
104 jmp r9
r9=11

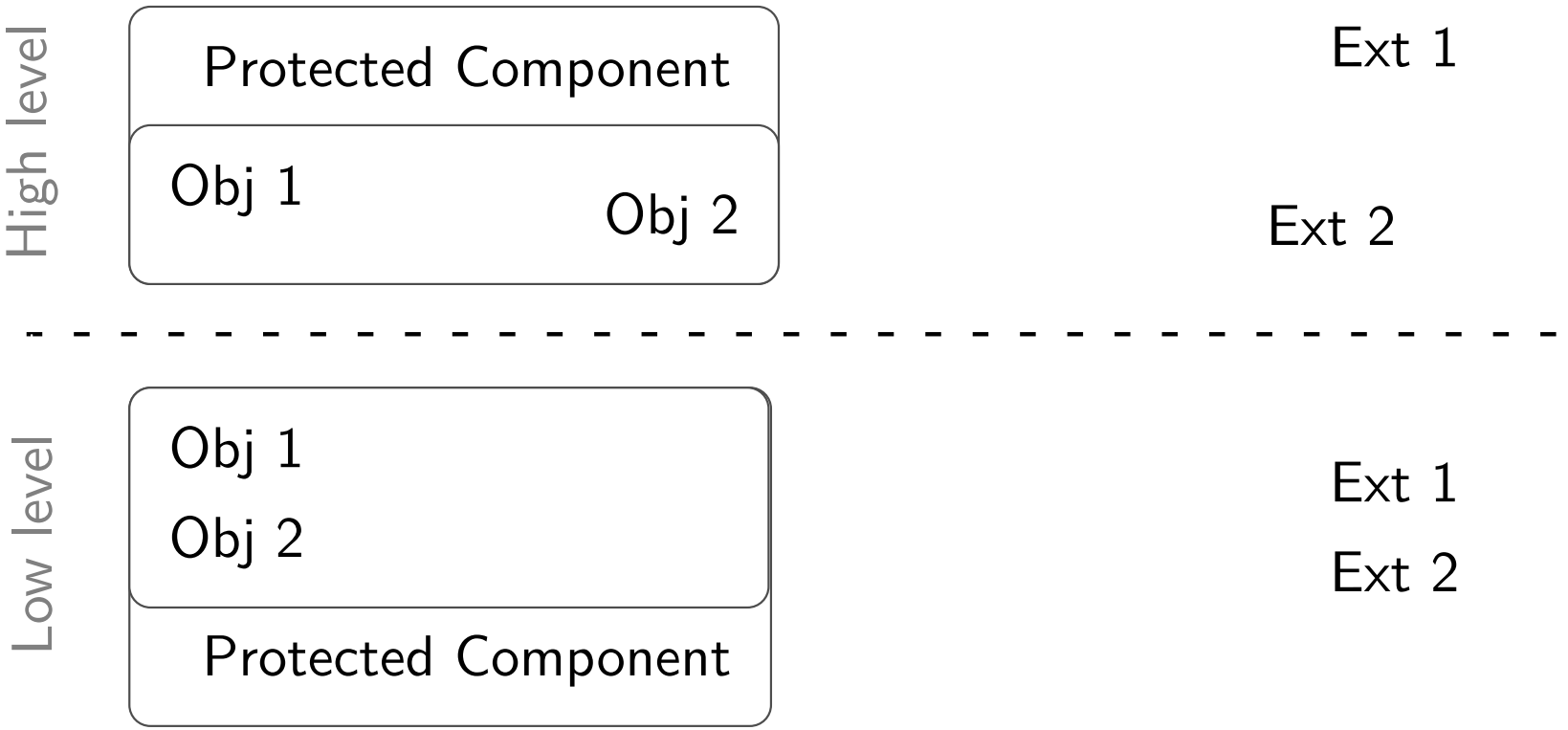
FPMAC



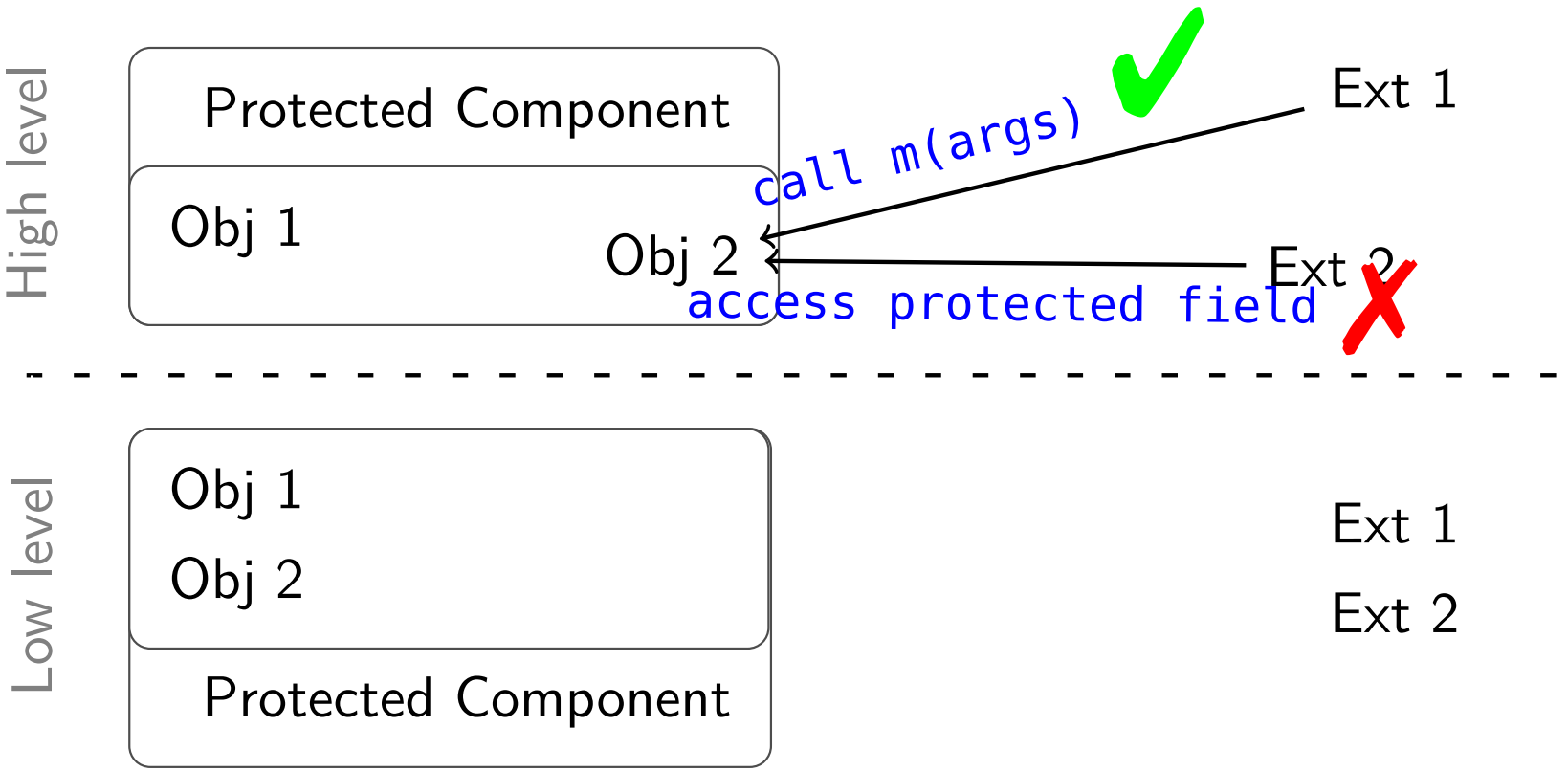
11 is not an entry point

Outline

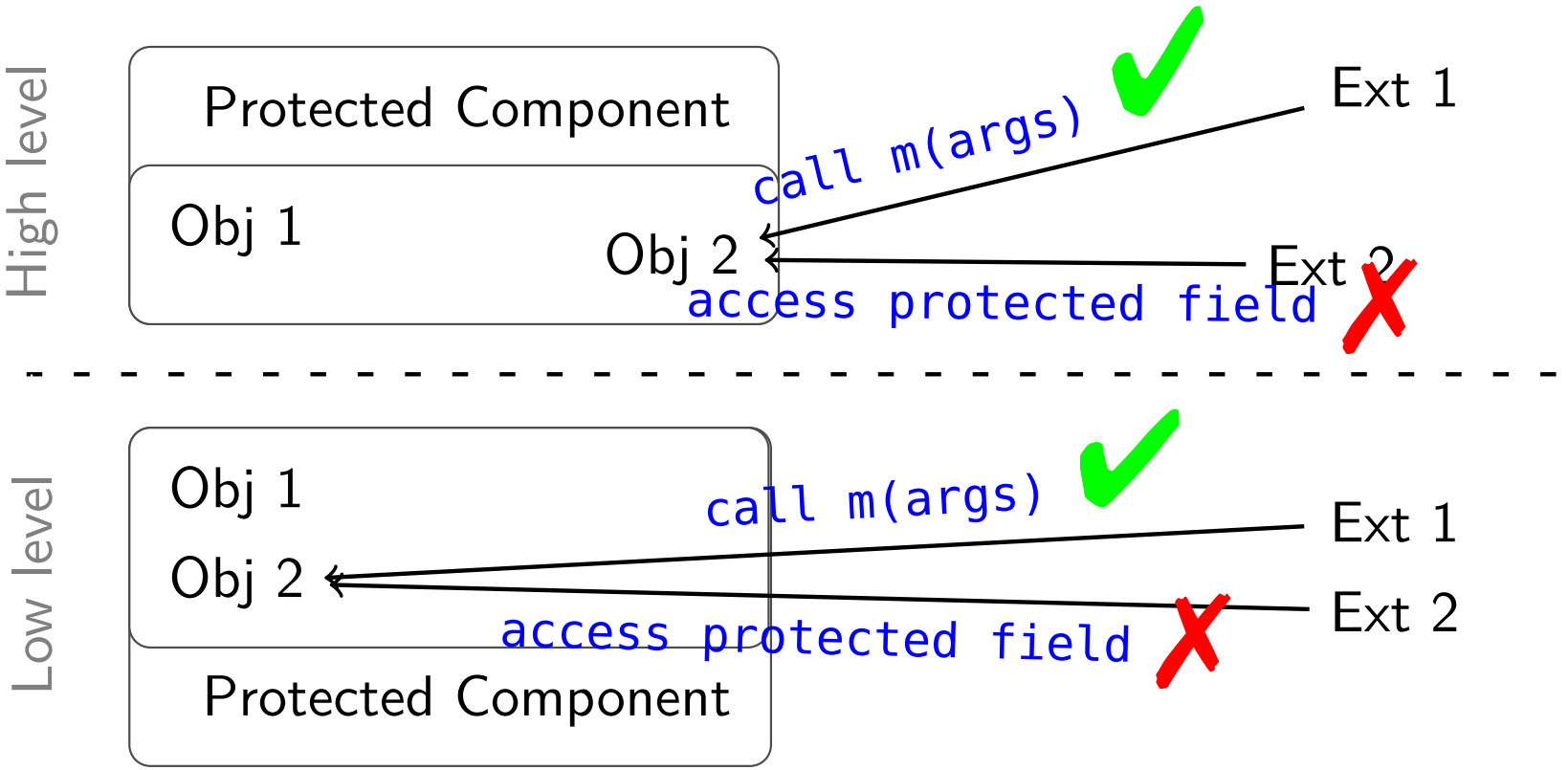
Secure Compilation, Informally



Secure Compilation, Informally



Secure Compilation, Informally



Secure Compilation, Formally

$$C_1 \simeq_H C_2 \iff C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

Secure Compilation, Formally

$$C_1 \simeq_H C_2 \iff C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

Secure Compilation, Formally

$$C_1 \simeq_H C_2 \iff C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

Contextual Equivalence

Contextual Equivalence

$$C_1 \simeq C_2 \triangleq \forall \mathbb{C}.\ \mathbb{C}[C_1]\Uparrow \iff \mathbb{C}[C_2]\Uparrow$$

Contextual Equivalence

$$C_1 \simeq C_2 \triangleq \forall \mathbb{C}. \mathbb{C}[C_1]\Uparrow \iff \mathbb{C}[C_2]\Uparrow$$

Contextual Equivalence

$$C_1 \simeq C_2 \triangleq \forall \mathbb{C}. \mathbb{C}[C_1]\Uparrow \iff \mathbb{C}[C_2]\Uparrow$$

All contexts

Secure Compilation

$$C_1 \simeq_H C_2 \iff C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

Secure Compilation

$$C_1 \simeq_H C_2 \iff C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

$$(\forall \mathbb{C}.\ \mathbb{C}[C_1]\Uparrow \iff \mathbb{C}[C_2]\Uparrow\ ) \iff (\forall \mathbb{M}.\ \mathbb{M}[C_1^{\downarrow}]\Uparrow \iff \mathbb{M}[C_2^{\downarrow}]\Uparrow\ )$$

Secure Compilation

$$C_1 \simeq_H C_2 \iff C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

NO WAY

$$(\forall \mathbb{C}.\ \mathbb{C}[C_1]\Uparrow \iff \mathbb{C}[C_2]\Uparrow) \iff (\forall \mathbb{M}.\ \mathbb{M}[C_1^{\downarrow}]\Uparrow \iff \mathbb{M}[C_2^{\downarrow}]\Uparrow)$$

Secure Compilation

$$C_1 \simeq_H C_2 \iff C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

Secure Compilation

$$C_1 \simeq_H C_2 \quad \Leftarrow \quad C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

Secure Compilation

$$C_1 \simeq_H C_2 \quad \Leftarrow \quad C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

Secure Compilation

$$C_1 \simeq_H C_2 \quad \Rightarrow \quad C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

Secure Compilation

$$C_1 \simeq_H C_2 \quad \Rightarrow \quad \boxed{\begin{array}{c} C_1^{\downarrow} \simeq_L C_2^{\downarrow} \\ \Updownarrow \\ \mathrm{Traces}(C_1^{\downarrow}) = \mathrm{Traces}(C_2^{\downarrow}) \end{array}}$$

Fully Abstract Trace Semantics

Secure Compilation

$$C_1 \not\approx_H C_2 \quad \Leftarrow \quad \text{Traces}(C_1^{\downarrow}) \neq \text{Traces}(C_2^{\downarrow})$$

Secure Compilation

$$C_1 \not\approx_H C_2 \quad \Leftarrow \quad \mathsf{Traces}(C_1^{\downarrow}) \neq \mathsf{Traces}(C_2^{\downarrow})$$

Secure Compilation

$C_1 \not\simeq_H C_2$ ⇐

$$C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$
$$\Updownarrow$$
$$\mathrm{Traces}(C_1^{\downarrow}) = \mathrm{Traces}(C_2^{\downarrow})$$

Fully Abstract Trace Semantics

Outline

Outline

Syntax

`movl` r_d r_s	`movs` r_d r_s
`movi` r_d i	`add` r_d r_s
`sub` r_d r_s	`cmp` r_1 r_2
`jmp` r_i	`je` r_i
`jl` r_i	`call` r_i
`ret`	`halt`

Outline

Operational and Trace Semantics

Operational Semantics

$$\frac{\begin{array}{c}\text{(Eval-movs)} \\ M(p) \cong (\texttt{movs}\ \texttt{r}_\texttt{d}\ \texttt{r}_\texttt{s}) \\ s \vdash \texttt{validJump}(p, p+1) \\ s \vdash \texttt{writeAllowed}(p, *r_d) \\ M' = M[*r_d \mapsto r_s]\end{array}}{(p, r, f, M, s) \twoheadrightarrow (p+1, r, f, M', s)}$$

Trace Semantics

Operational and Trace Semantics

Operational Semantics

$$\frac{\begin{array}{c}\text{(Eval-movs)} \\ M(p) \cong (\texttt{movs}\ \texttt{r}_\texttt{d}\ \texttt{r}_\texttt{s}) \\ s \vdash \texttt{validJump}(p, p+1) \\ s \vdash \texttt{writeAllowed}(p, *r_d) \\ M' = M[*r_d \mapsto r_s]\end{array}}{\underbrace{(p, r, f, M, s)} \twoheadrightarrow (p+1, r, f, M', s)}$$

$M = \text{total}$

Trace Semantics

Operational and Trace Semantics

Operational Semantics

$$\frac{\begin{array}{c}\text{(Eval-movs)}\\ M(p) \cong (\texttt{movs}\ \texttt{r}_\texttt{d}\ \texttt{r}_\texttt{s}) \\ s \vdash \texttt{validJump}(p, p+1) \\ s \vdash \texttt{writeAllowed}(p, *r_d) \\ M' = M[*r_d \mapsto r_s]\end{array}}{\underbrace{(p, r, f, M, s)}_{M = \text{total}} \twoheadrightarrow (p+1, r, f, M', s)}$$

Trace Semantics

$$\frac{\begin{array}{c}\text{(Trace-internal)}\\ (p, r, f, m, s) \twoheadrightarrow (p', r', f', m', s) \\ s \vdash \texttt{internalJump}(p, p')\end{array}}{(p, r, f, m, s) \overset{\tau_i}{\Longrightarrow} (p', r', f', m', s)}$$

Operational and Trace Semantics

Operational Semantics

$$\frac{\begin{array}{c}\text{(Eval-movs)}\\ M(p) \cong (\texttt{movs}\ \texttt{r}_\texttt{d}\ \texttt{r}_\texttt{s}) \\ s \vdash \texttt{validJump}(p, p+1) \\ s \vdash \texttt{writeAllowed}(p, *r_d) \\ M' = M[*r_d \mapsto r_s]\end{array}}{\underbrace{(p, r, f, M, s)}_{M = \text{total}} \twoheadrightarrow (p+1, r, f, M', s)}$$

Trace Semantics

$$\frac{\begin{array}{c}\text{(Trace-internal)}\\ (p, r, f, m, s) \twoheadrightarrow (p', r', f', m', s) \\ s \vdash \texttt{internalJump}(p, p')\end{array}}{\underbrace{(p, r, f, m, s)}_{m = \text{protected}} \overset{\tau_i}{\twoheadrightarrow} (p', r', f', m', s)}$$

Operational and Trace Semantics

Operational Semantics

$$\frac{\begin{array}{c}\text{(Eval-movs)}\\ M(p) \cong (\texttt{movs r}_\texttt{d}\ \texttt{r}_\texttt{s}) \\ s \vdash \texttt{validJump}(p, p+1) \\ s \vdash \texttt{writeAllowed}(p, *r_d) \\ M' = M[*r_d \mapsto r_s]\end{array}}{(p, r, f, M, s) \twoheadrightarrow (p+1, r, f, M', s)}$$

$$\frac{\begin{array}{c}\text{(Eval-jump)}\\ M(p) \cong (\texttt{jmp r}_\texttt{d}) \qquad p' = r_d \\ s \vdash \texttt{validJump}(p, p')\end{array}}{(p, r, f, M, s) \twoheadrightarrow (p', r, f, M, s)}$$

Trace Semantics

$$\frac{\begin{array}{c}\text{(Trace-internal)}\\ (p, r, f, m, s) \twoheadrightarrow (p', r', f', m', s) \\ s \vdash \texttt{internalJump}(p, p')\end{array}}{(p, r, f, m, s) \overset{\tau_i}{\twoheadrightarrow} (p', r', f', m', s)}$$

Operational and Trace Semantics

Operational Semantics

$$\frac{\begin{array}{c}\text{(Eval-movs)}\\ M(p) \cong (\texttt{movs r}_\texttt{d}\ \texttt{r}_\texttt{s}) \\ s \vdash \texttt{validJump}(p, p+1) \\ s \vdash \texttt{writeAllowed}(p, *r_d) \\ M' = M[*r_d \mapsto r_s]\end{array}}{(p, r, f, M, s) \twoheadrightarrow (p+1, r, f, M', s)}$$

$$\frac{\begin{array}{c}\text{(Eval-jump)}\\ M(p) \cong (\texttt{jmp r}_\texttt{d}) \qquad p' = r_d \\ s \vdash \texttt{validJump}(p, p')\end{array}}{(p, r, f, M, s) \twoheadrightarrow (p', r, f, M, s)}$$

Trace Semantics

$$\frac{\begin{array}{c}\text{(Trace-internal)}\\ (p, r, f, m, s) \twoheadrightarrow (p', r', f', m', s) \\ s \vdash \texttt{internalJump}(p, p')\end{array}}{(p, r, f, m, s) \xrightarrow{\tau_i} (p', r', f', m', s)}$$

$$\frac{\begin{array}{c}\text{(Trace-callback)}\\ (p, r, f, m, s) \twoheadrightarrow (p', r', f', m', s) \\ s \vdash \texttt{exitJump}(p, p') \qquad m(p) \cong (\texttt{jmp}) \\ \overline{v} = \mathrm{R}_4 :: \ldots :: \mathrm{R}_{11}\end{array}}{(p, r, f, m, s) \xrightarrow{\texttt{call } p'(\overline{v})!} (p', r', f', m', s)}$$

Operational and Trace Semantics

Operational Semantics

$$\frac{\begin{array}{c}\text{(Eval-movs)}\\ M(p) \cong (\texttt{movs}\ \texttt{r}_\texttt{d}\ \texttt{r}_\texttt{s}) \\ s \vdash \texttt{validJump}(p, p+1) \\ s \vdash \texttt{writeAllowed}(p, *r_d) \\ M' = M[*r_d \mapsto r_s]\end{array}}{(p, r, f, M, s) \twoheadrightarrow (p+1, r, f, M', s)}$$

$$\frac{\begin{array}{c}\text{(Eval-jump)}\\ M(p) \cong (\texttt{jmp}\ \texttt{r}_\texttt{d}) \qquad p' = r_d \\ s \vdash \texttt{validJump}(p, p')\end{array}}{(p, r, f, M, s) \twoheadrightarrow (p', r, f, M, s)}$$

Trace Semantics

$$\frac{\begin{array}{c}\text{(Trace-internal)}\\ (p, r, f, m, s) \twoheadrightarrow (p', r', f', m', s) \\ s \vdash \texttt{internalJump}(p, p')\end{array}}{(p, r, f, m, s) \xRightarrow{\tau_i} (p', r', f', m', s)}$$

$$\frac{\begin{array}{c}\text{(Trace-callback)}\\ (p, r, f, m, s) \twoheadrightarrow (p', r', f', m', s) \\ s \vdash \texttt{exitJump}(p, p') \qquad m(p) \cong (\texttt{jmp}) \\ \overline{v} = \mathrm{R}_4 :: \ldots :: \mathrm{R}_{11}\end{array}}{(p, r, f, m, s) \xRightarrow{\texttt{call}\ p'(\overline{v})!} (p', r', f', m', s)}$$

Full Abstraction of the Trace Semantics in General

$$\mathsf{Traces}(C_1^{\downarrow}) = \mathsf{Traces}(C_2^{\downarrow}) \iff C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

$$\alpha ::= \gamma? \mid \gamma!$$
$$\gamma ::= \texttt{call}\ p(r_4 :: \ldots :: r_{11}) \mid \texttt{ret}\ r_0$$

Full Abstraction of the Trace Semantics in General

$$\mathsf{Traces}(C_1^{\downarrow}) = \mathsf{Traces}(C_2^{\downarrow}) \iff C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

$$\alpha ::= \gamma? \mid \gamma!$$
$$\gamma ::= \texttt{call}\ p(r_4 :: \ldots :: r_{11}) \mid \texttt{ret}\ r_0$$

✗ does not hold in general

Full Abstraction of the Trace Semantics in General

$$\text{Traces}(C_1^{\downarrow}) = \text{Traces}(C_2^{\downarrow}) \iff C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

$$\alpha ::= \gamma? \mid \gamma!$$
$$\gamma ::= \texttt{call}\ p(r_4 :: \ldots :: r_{11}) \mid \texttt{ret}\ r_0$$

✗ does not hold in general

Flags, unused registers
Readouts, writeouts

Full Abstraction of the Trace Semantics in General

$$\text{Traces}(C_1^{\downarrow}) = \text{Traces}(C_2^{\downarrow}) \iff C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

$$\alpha ::= \gamma? \mid \gamma!$$
$$\gamma ::= \texttt{call}\ p(r_4 :: \ldots :: r_{11}) \mid \texttt{ret}\ r_0$$

✗ does not hold in general

Change the semantics

Flags, unused registers

Readouts, writeouts

Reset

Forbid

Full Abstraction of the Trace Semantics in General

$$\text{Traces}(C_1^{\downarrow}) = \text{Traces}(C_2^{\downarrow}) \iff C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

$$\alpha ::= \gamma? \mid \gamma!$$
$$\gamma ::= \texttt{call}\ p(r_4 :: \ldots :: r_{11}) \mid \texttt{ret}\ r_0$$

✗ does not hold in general

Change the semantics

✓ holds for compiled components

Flags, unused registers

Readouts, writeouts

Reset

Forbid

Full Abstraction of the Trace Semantics in General

$$\mathsf{Traces}(C_1^{\downarrow}) = \mathsf{Traces}(C_2^{\downarrow}) \iff C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

$$\alpha ::= \gamma? \mid \gamma!$$
$$\gamma ::= \texttt{call}\ p(r_4 :: \ldots :: r_{11}) \mid \texttt{ret}\ r_0$$

✗ does not hold in general

Full Abstraction of the Trace Semantics in General

$$\text{Traces}(C_1^{\downarrow}) = \text{Traces}(C_2^{\downarrow}) \iff C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

$$\alpha ::= \gamma? \mid \gamma!$$
$$\gamma ::= \texttt{call}\ p(r_4 :: \ldots :: r_{11}) \mid \texttt{ret}\ r_0$$

✗ does not hold in general

Change the labels

Full Abstraction of the Trace Semantics in General

$$\text{Traces}(P_1) = \text{Traces}(P_2) \iff P_1 \simeq_L P_2$$

$$\alpha ::= (r, f)\gamma? \mid (r, f)\gamma!$$
$$\gamma ::= \texttt{call}\ p(r_4 :: \ldots :: r_{11}) \mid \texttt{ret}\ r_0 \mid \texttt{read}\ r_s \mid \texttt{write}\ r_s$$

Change the labels

Full Abstraction of the Trace Semantics in General

$$\text{Traces}(P_1) = \text{Traces}(P_2) \iff P_1 \simeq_L P_2$$

$$\alpha ::= (r, f)\gamma? \mid (r, f)\gamma!$$
$$\gamma ::= \texttt{call}\ p(r_4 :: \ldots :: r_{11}) \mid \texttt{ret}\ r_0 \mid \texttt{read}\ r_s \mid \texttt{write}\ r_s$$

holds in general

Change the labels

Outline

Main Theorem

$$\text{Traces}(P_1) = \text{Traces}(P_2) \quad \Longleftrightarrow \quad P_1 \simeq P_2$$

Main Theorem

$$\mathsf{Traces}(P_1) = \mathsf{Traces}(P_2) \iff \forall \mathbb{M}.\ \mathbb{M}[P_1]\Uparrow \iff \mathbb{M}[P_2]\Uparrow$$

Main Theorem

$$\text{Traces}(P_1) = \text{Traces}(P_2) \iff \forall \mathbb{M}.\ \mathbb{M}[P_1]\Uparrow \iff \mathbb{M}[P_2]\Uparrow$$

Proof that scales to both solutions

Soundness

Soundness:

$$\mathsf{Trace}(P_1) = \mathsf{Trace}(P_2) \Leftarrow P_1 \simeq P_2$$

Soundness

Soundness:

$$\text{Trace}(P_1) \neq \text{Trace}(P_2) \Rightarrow P_1 \not\approx P_2$$

Soundness

Soundness:

$$\mathsf{Trace}(P_1) \neq \mathsf{Trace}(P_2) \Rightarrow \exists \mathbb{M}.\ \mathbb{M}[P_1]\Uparrow \quad \not\Longleftrightarrow \quad \mathbb{M}[P_2]\Uparrow$$

Soundness

Soundness:

$$\mathsf{Trace}(P_1) \neq \mathsf{Trace}(P_2) \Rightarrow \exists \mathbb{M}.\ \mathbb{M}[P_1]\Uparrow \iff \mathbb{M}[P_2]\Uparrow$$

- Two traces have a different action at index i

Soundness

Soundness:

$$\text{Trace}(P_1) \neq \text{Trace}(P_2) \Rightarrow \exists \mathbb{M}.\ \mathbb{M}[P_1]\Uparrow \iff \mathbb{M}[P_2]\Uparrow$$

- Two traces have a different action at index i
- Create $\mathbb{M}$ such that it replicates the traces until action i

Soundness

Soundness:

$$\text{Trace}(P_1) \neq \text{Trace}(P_2) \Rightarrow \exists \mathbb{M}.\ \mathbb{M}[P_1]\Uparrow \iff \mathbb{M}[P_2]\Uparrow$$

- Two traces have a different action at index i
- Create $\mathbb{M}$ such that it replicates the traces until action i
- At action i, $\mathbb{M}$ makes P_1 diverge and P_2 terminate

Completeness

Completeness:

$$\text{Trace}(P_1) = \text{Trace}(P_2) \Rightarrow P_1 \simeq P_2$$

Completeness

Completeness:

$$\mathsf{Trace}(P_1) = \mathsf{Trace}(P_2) \Rightarrow \forall \mathbb{M}.\ \mathbb{M}[P_1]\Uparrow \iff \mathbb{M}[P_2]\Uparrow$$

Completeness

Completeness:

$$\mathsf{Trace}(P_1) = \mathsf{Trace}(P_2) \Rightarrow \forall \mathbb{M}.\ \mathbb{M}[P_1]\Uparrow \iff \mathbb{M}[P_2]\Uparrow$$

Proof strategy: coinduction

Completeness

Completeness:

$$\mathrm{Trace}(P_1) = \mathrm{Trace}(P_2) \Rightarrow \forall \mathbb{M}.\ \mathbb{M}[P_1]\Uparrow \iff \mathbb{M}[P_2]\Uparrow$$

Proof strategy: coinduction

$$\left.\begin{array}{l} P_1 \text{ stuck} \wedge P_2 \text{ stuck} \\ P_1 \text{ terminated} \wedge P_2 \text{ terminated} \\ P_1\Uparrow \text{ locally} \wedge P_2\Uparrow \text{ locally} \\ P_1' \simeq_L P_2' \wedge P_1 \stackrel{\alpha}{\Longrightarrow} P_1' \wedge P_2 \stackrel{\alpha}{\Longrightarrow} P_2' \end{array}\right\} \Rightarrow P_1 \simeq_L P_2$$

Outline

Conclusion

- A fully abstract trace semantics for FPMACs eliminates the need to use contextual equivalence

Conclusion

- A fully abstract trace semantics for FPMACs eliminates the need to use contextual equivalence
- This can be used to reason more easily about these systems (e.g. prove secure compilation to FPMACs)

Questions

?