Return-Oriented Programming without Returns

Seguridad Informática

Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Universidad Nacional de Rosario

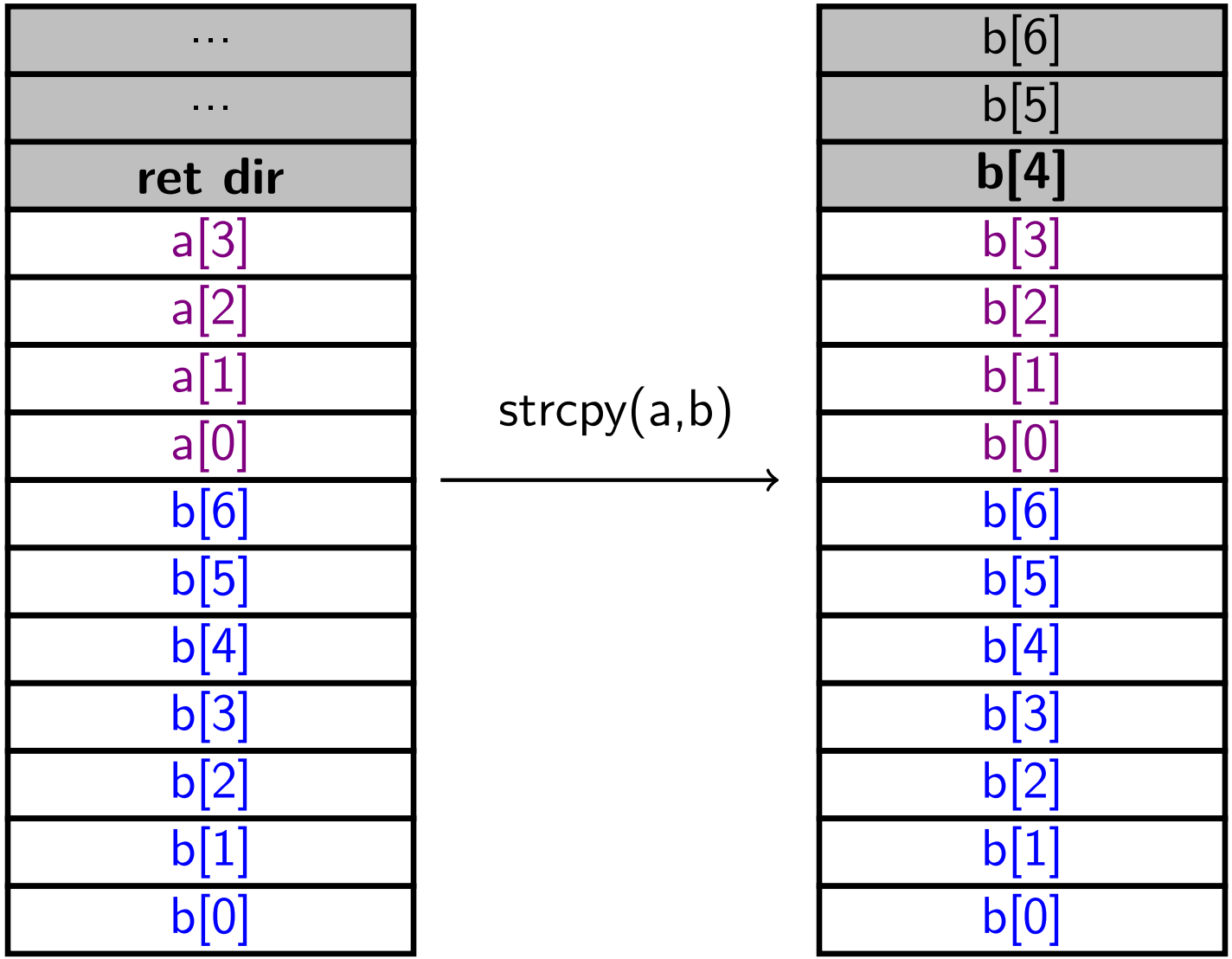
Agosto 2025

Inyección de Código

Una forma de tomar el control de un proceso es: aprovechar un desbordamiento de arreglo en la pila.

Pisar la dirección de retorno y saltar a una parte del código o ejecutar código en la misma pila.

Error de desbordamiento de arreglo en la pila



Contramedidas

- Programación fuertemente tipada y validar longitudes.
- No utilizar funciones de biblioteca y diseñar de forma segura.
- Minimizar los puntos de error.
- Verificar los retornos.
- Colocar la pila como no ejecutable y agregar canarios.

Programación orientada a los retornos[1]

Permite explotar errores de memoria sin inyectar código usando el código del mismo proceso.

Aprovechando la instrucción `return` pues realizan un **salto** y **cambia el estado**.

[1] *Return-Oriented Programing*

Conjunto de *gadgets*

Definición

Uso de secuencia de instrucciones que ya están presentes en el proceso, en particular tienen permisos de ejecución. El stack es usado como lugar para almacenar el "código".

Las secuencias deben terminar en la instrucción `return` así permiten definir una máquina virtual.

Se puede buscar posibles *Gadget* de manera automatizada dendentro del código compilado.

Conjunto de *gadgets*

Definición

Uso de secuencia de instrucciones que ya están presentes en el proceso, en particular tienen permisos de ejecución. El stack es usado como lugar para almacenar el "código".

Las secuencias deben terminar en la instrucción `return` así permiten definir una máquina virtual.

Se puede buscar posibles *Gadget* de manera automatizada dendentro del código compilado.

Convención de llamada

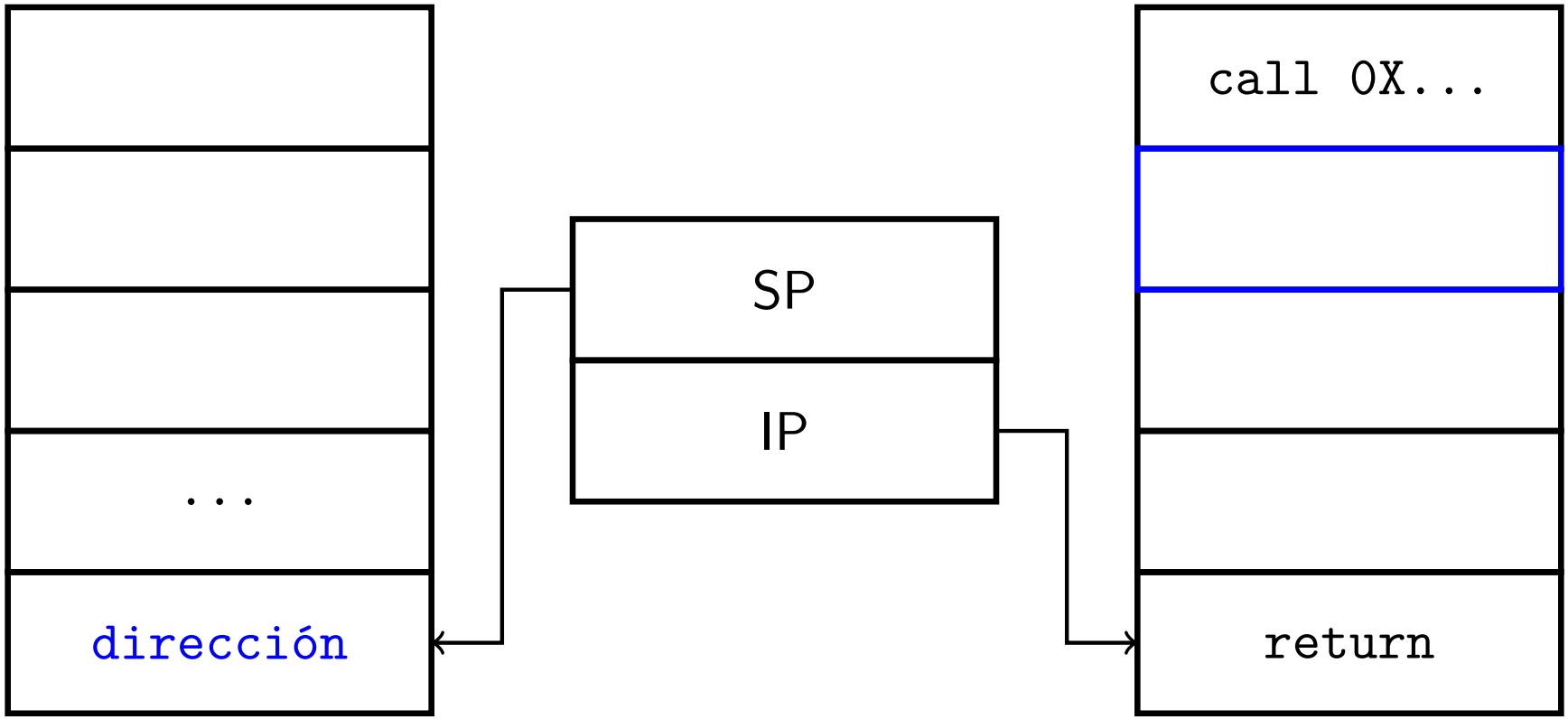


Figure: Estado de los registro para la pila e instrucciones antes de un `return`.

Convención de llamada

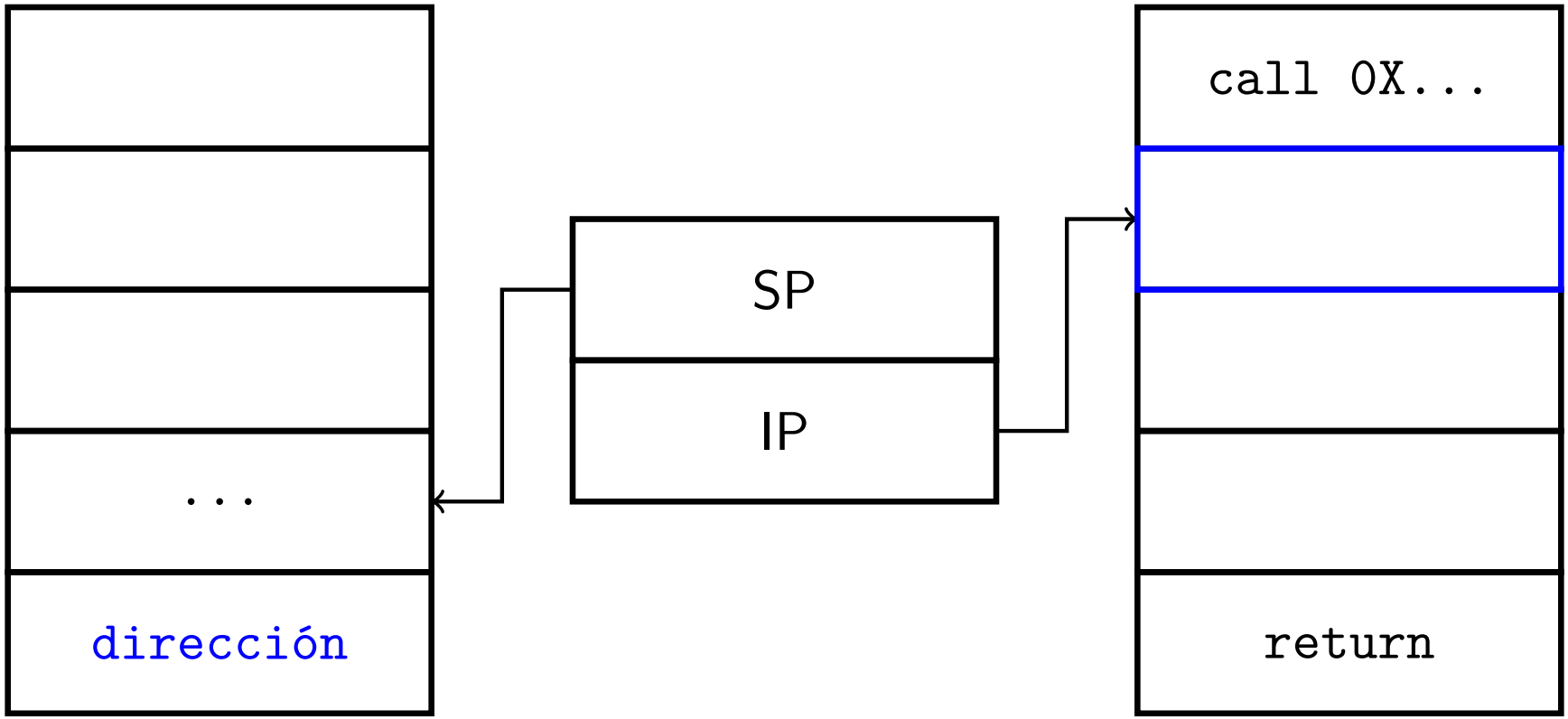


Figure: Estado de los registro para la pila e instrucciones después de un `return`.

Contramedidas

- Comparar la cantidad de `call` y `ret`.
- Validar el invariante de *last-in*, *first-out*.
- No usar la instrucción `return`.

Programación orientada a los retornos sin retornos[2]

Se asume:

- Modelo de lectura o ejecución exclusivos ($W \oplus X$).
- Contramedidas para la programación orientada a los retornos.

Necesitamos otros conjunto de instrucciones con las propiedades del `return`:

1. Trasferir el control de ejecución.
2. Actualizar el estado del proceso.

[2] *Return-Oriented Programing without Returns*

Si no usamos `return`

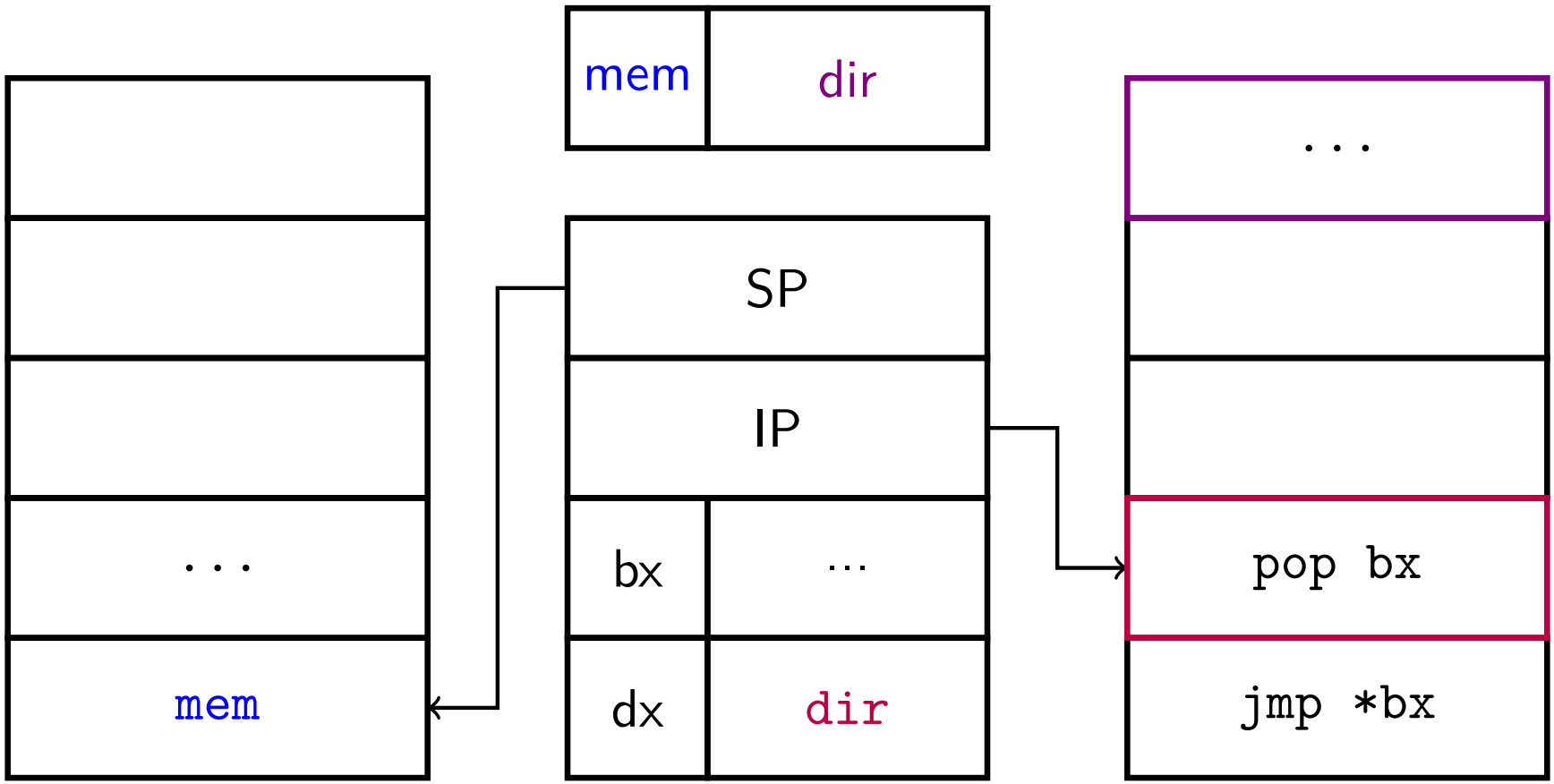


Figure: Ejecución del "trampolín": `pop` saca de la pila y coloca en **bx**

Si no usamos return

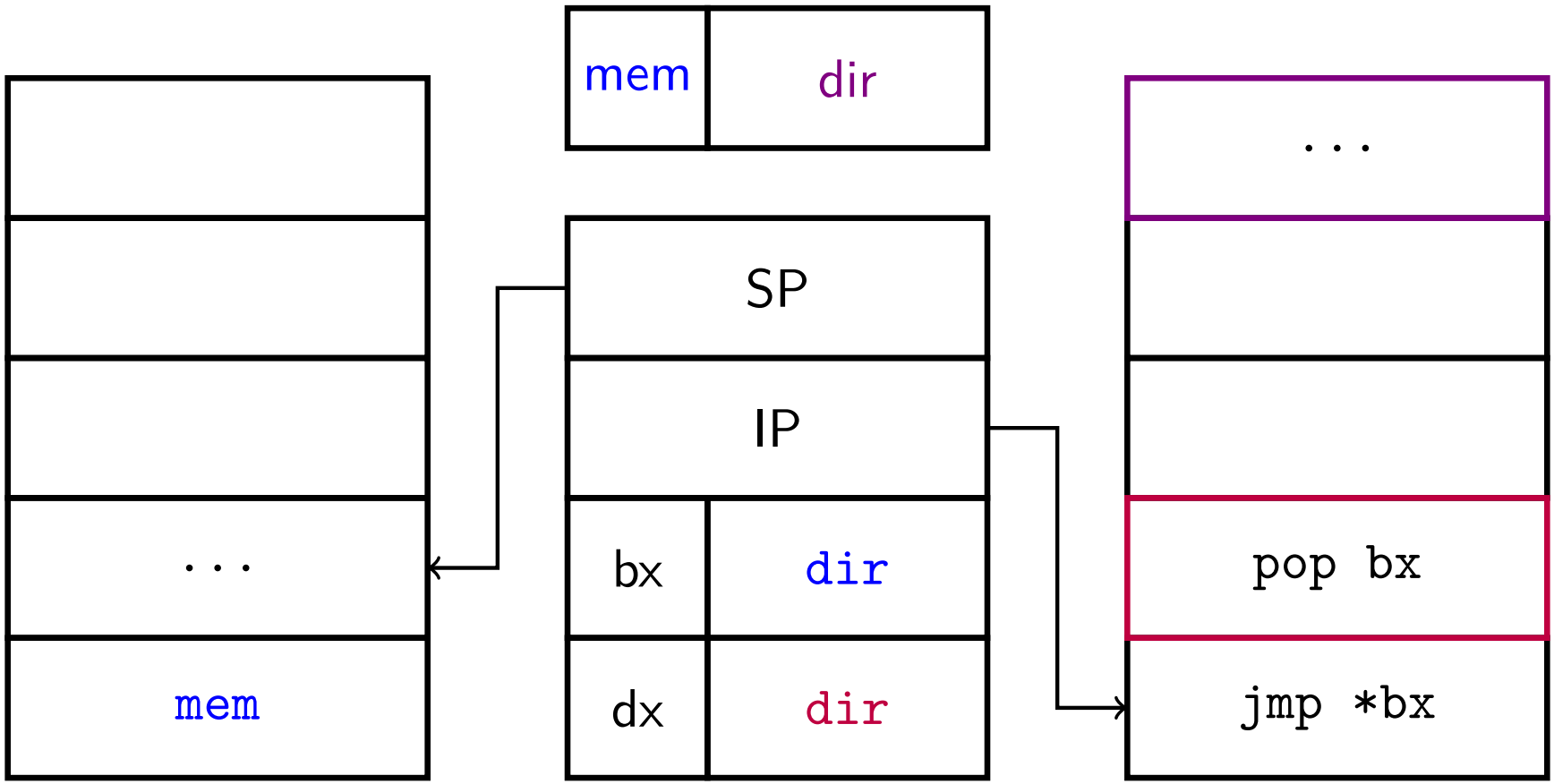


Figure: Ejecución del "trampolín": `jmp` salta a la dirección de **bx**

Si no usamos return

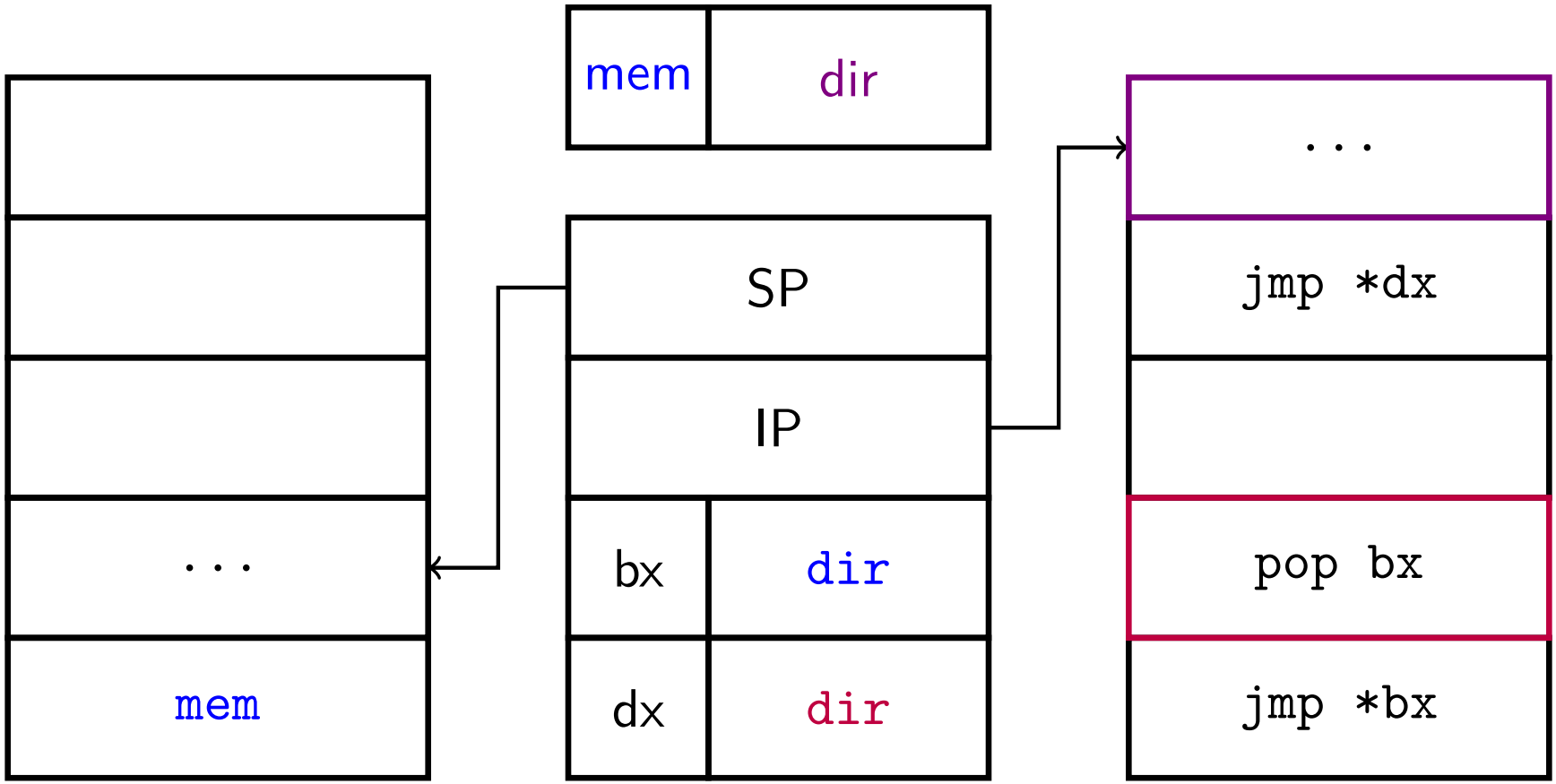


Figure: Ejecución del código del *gadget*. Debe terminar con un salto al trampolín.

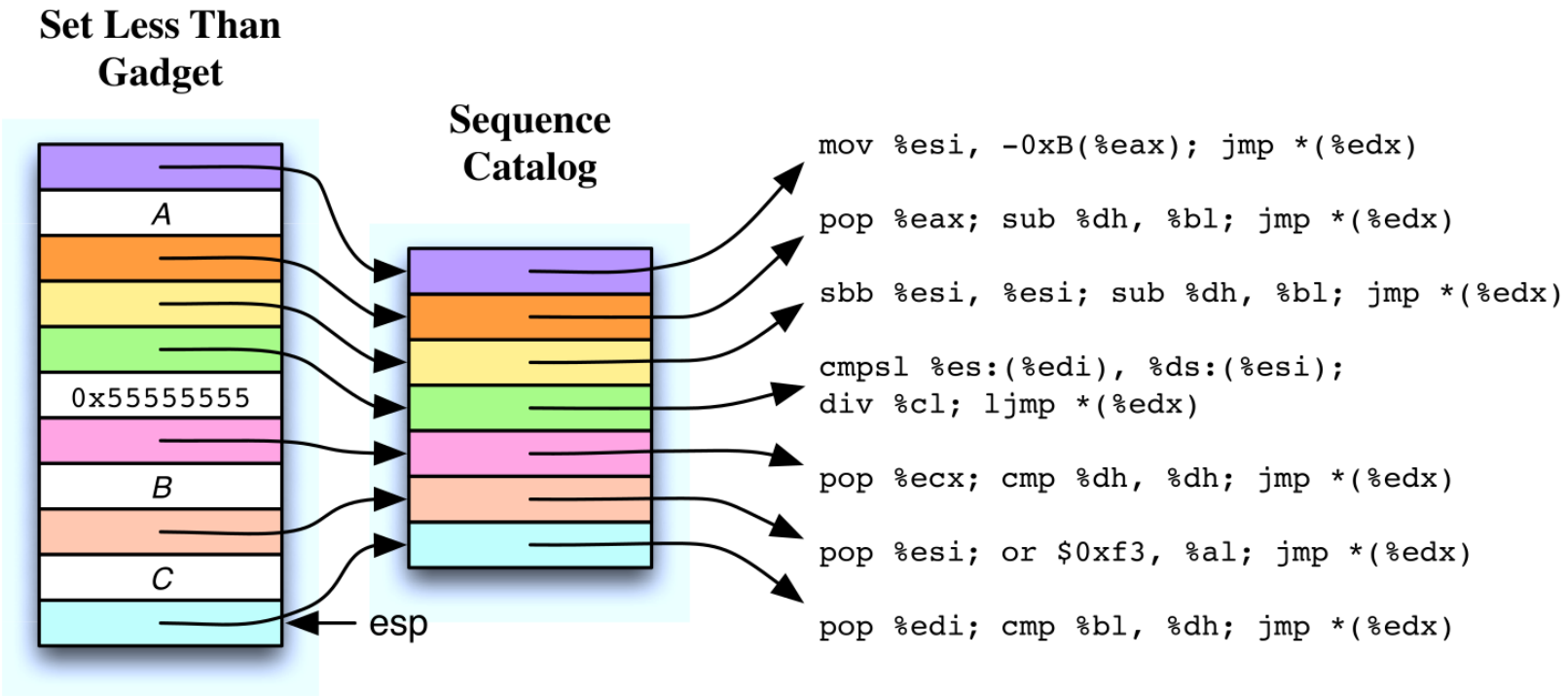
INTEL x86

En este caso el conjunto de *gadgets* proviene de `libc` y `libul`.
Los *gadgets* son 19: `load` (`immediate`), `move`, `store`, `add` (`immediate`), `subtract`, `negate`, `and` (`immediate`), `or` (`immediate`), `xor` (`immediate`), *complement*, `branch unconditional`, `branch conditional`, `set less than` y `call`.

INTEL x86: Observaciones

- Se busca que las operaciones actualicen valores en memoria.
- Las ramas condicionales actualizan valores de memoria y no el registro de instrucciones.
- La convención para hacer llamadas a funciones utilizada en el paper no es la actual en Linux.

INTEL x86: *gadget* ($\leq$)



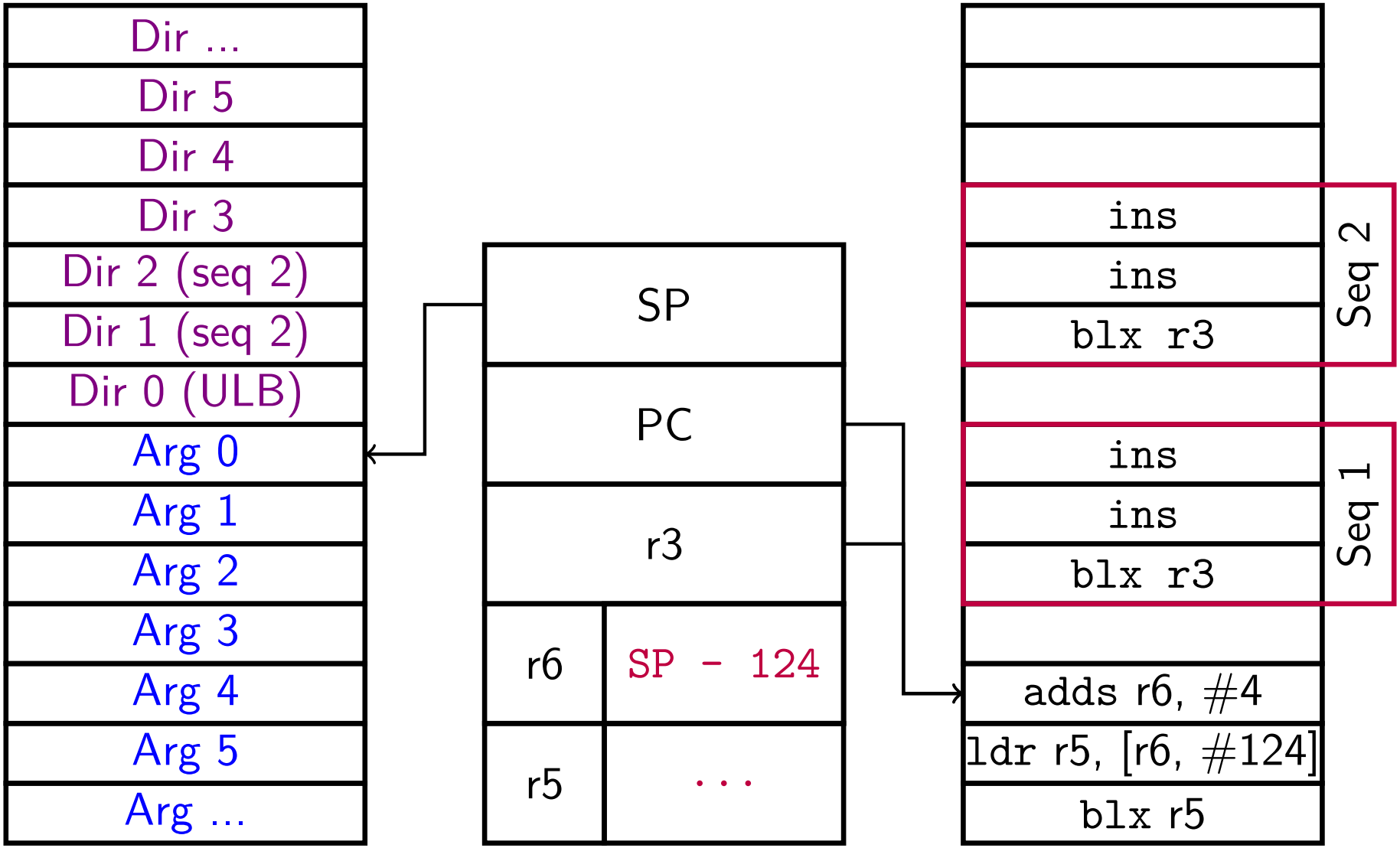
ARM

En ARM no hay una convención de llamada mediante instrucciones.

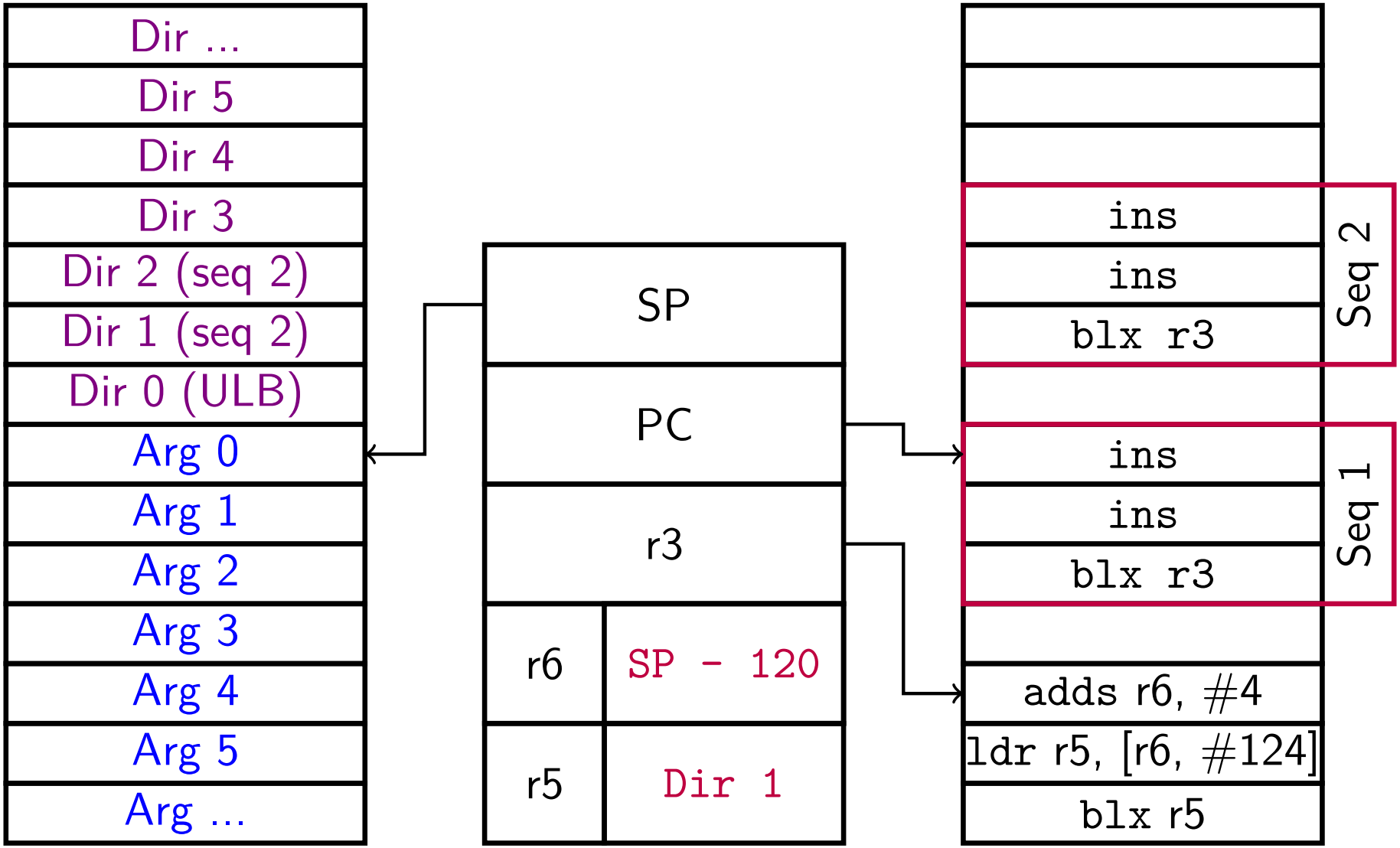
Hay menos registros pero todos son de propósito general y pueden ser modificados.

No hay instrucciones `call` y `ret`. En su lugar se tienen `bl` y `blx`.

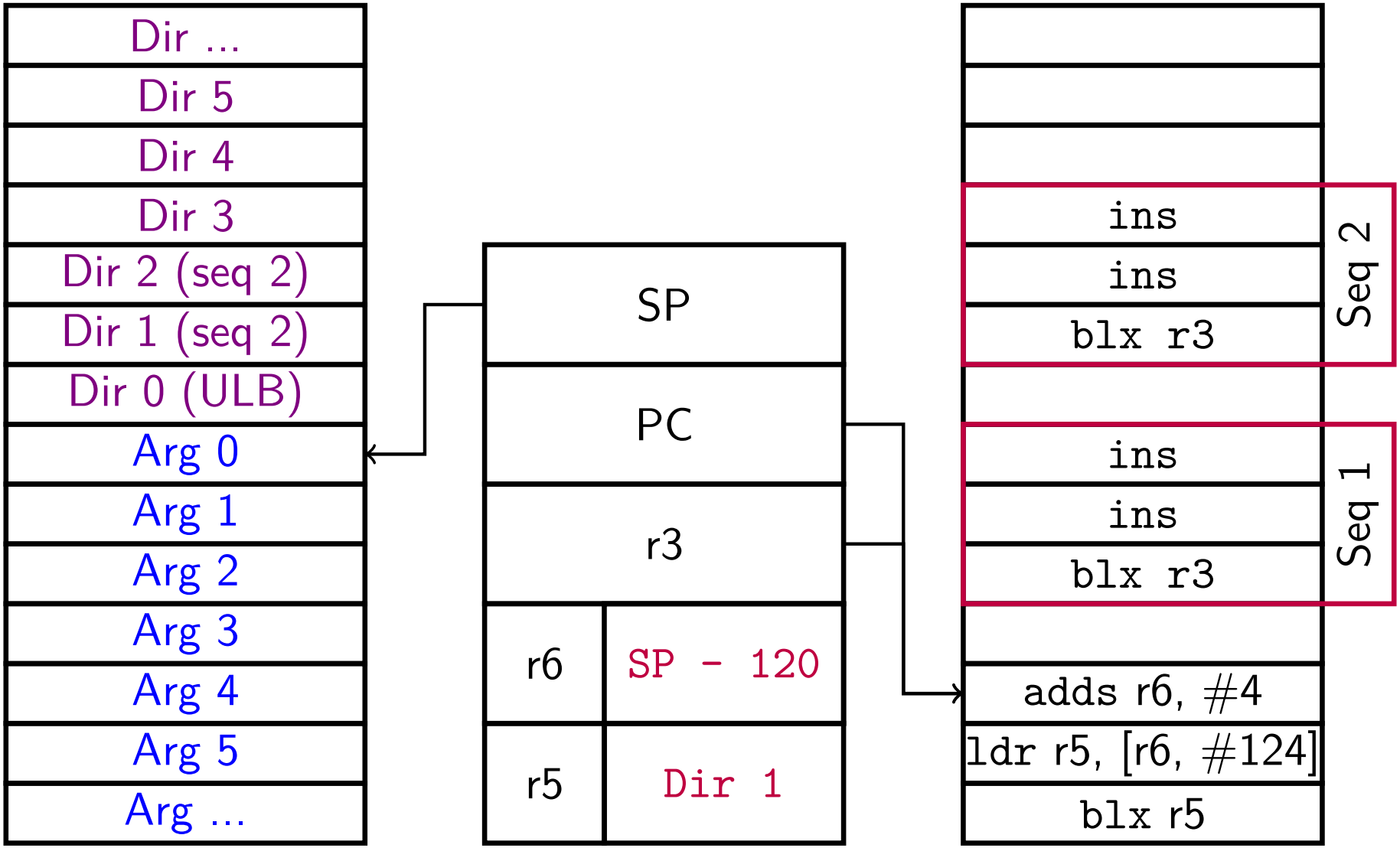
ARM


Dir ...
Dir 5
Dir 4
Dir 3
Dir 2 (seq 2)
Dir 1 (seq 2)
Dir 0 (ULB)
Arg 0
Arg 1
Arg 2
Arg 3
Arg 4
Arg 5
Arg ...
SP
PC
r3
r6
SP - 124
r5
...
ins
ins
blx r3
Seq 2
ins
ins
blx r3
Seq 1
adds r6, #4
ldr r5, [r6, #124]
blx r5

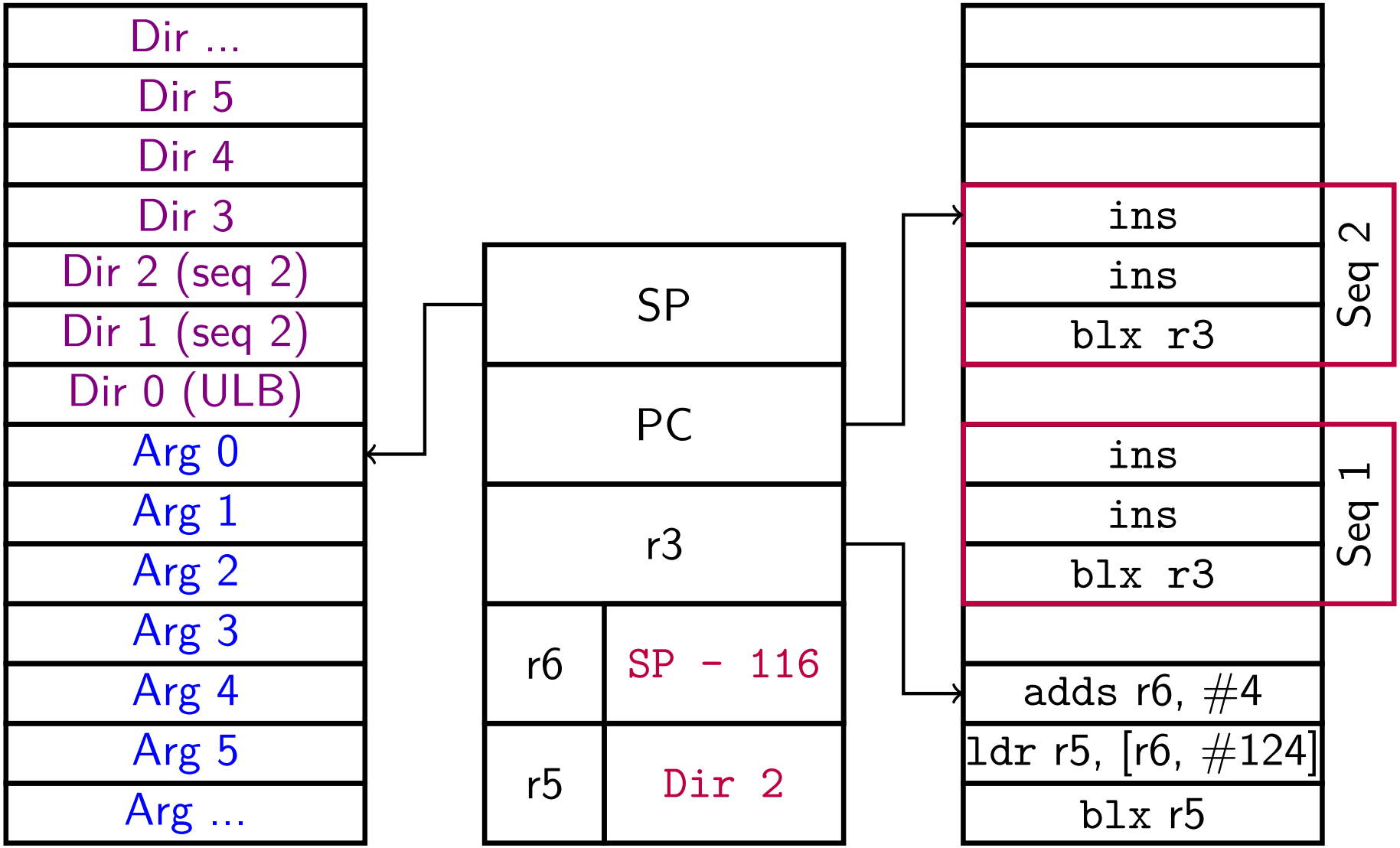
ARM


Dir ...
Dir 5
Dir 4
Dir 3
Dir 2 (seq 2)
Dir 1 (seq 2)
Dir 0 (ULB)
Arg 0
Arg 1
Arg 2
Arg 3
Arg 4
Arg 5
Arg ...
SP
PC
r3
r6
SP - 120
r5
Dir 1
ins
ins
blx r3
Seq 2
ins
ins
blx r3
Seq 1
adds r6, #4
ldr r5, [r6, #124]
blx r5

ARM


Dir ...
Dir 5
Dir 4
Dir 3
Dir 2 (seq 2)
Dir 1 (seq 2)
Dir 0 (ULB)
Arg 0
Arg 1
Arg 2
Arg 3
Arg 4
Arg 5
Arg ...
SP
PC
r3
r6
SP - 120
r5
Dir 1
ins
ins
blx r3
Seq 2
ins
ins
blx r3
Seq 1
adds r6, #4
ldr r5, [r6, #124]
blx r5

ARM

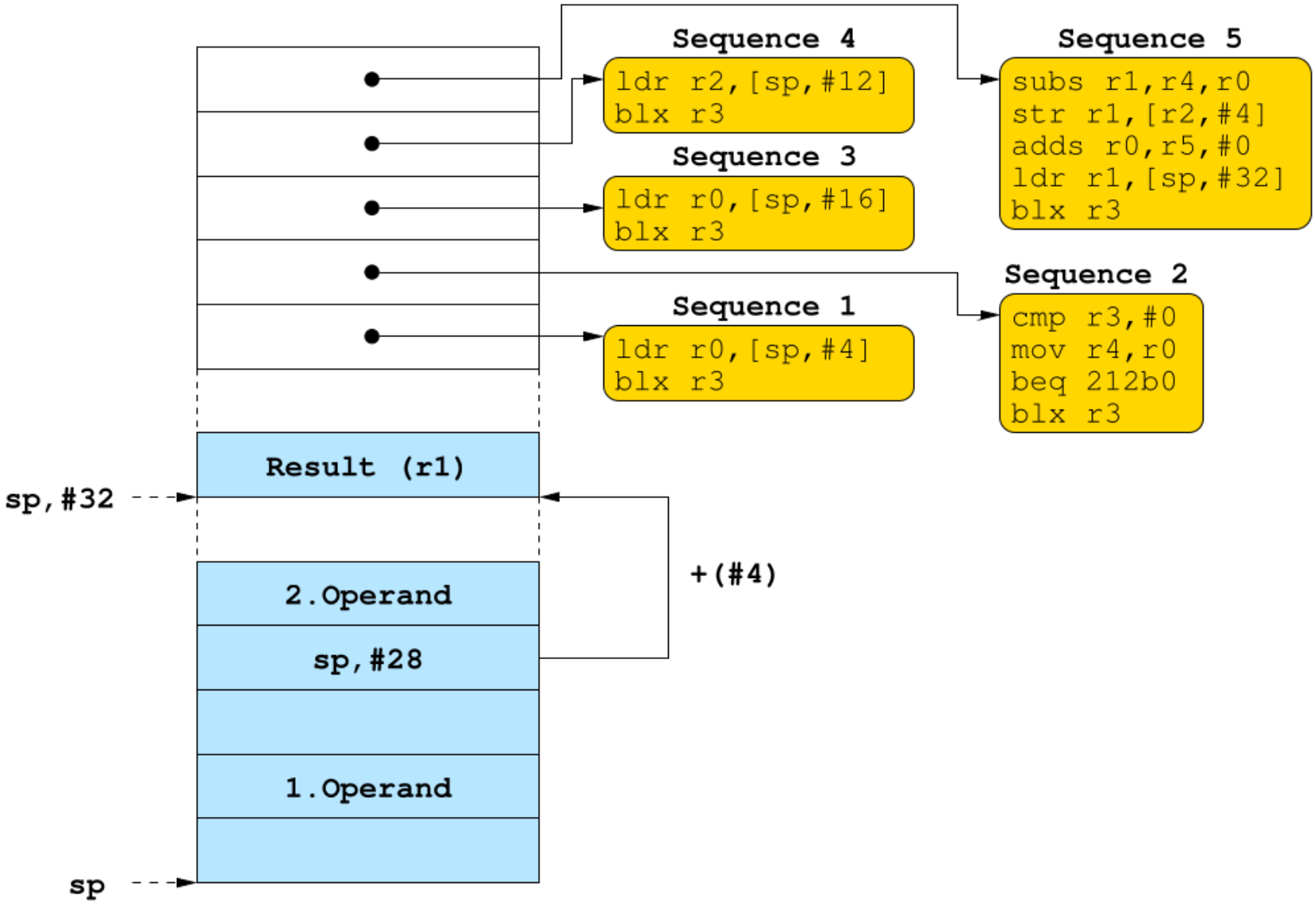

Dir ...
Dir 5
Dir 4
Dir 3
Dir 2 (seq 2)
Dir 1 (seq 2)
Dir 0 (ULB)
Arg 0
Arg 1
Arg 2
Arg 3
Arg 4
Arg 5
Arg ...
SP
PC
r3
r6
SP - 116
r5
Dir 2
ins
ins
blx r3
Seq 2
ins
ins
blx r3
Seq 1
adds r6, #4
ldr r5, [r6, #124]
blx r5

ARM (observaciones)

- Las instrucciones deben terminar con `blx`.
- Se construyeron los *gadgets* con **libc** y **libwebcore**
- Se requieren 3 registros en lugar de 1.

Este ataque no puede detectarse por los métodos para detectar retornos.

ARM *gadget* ($\leq$)

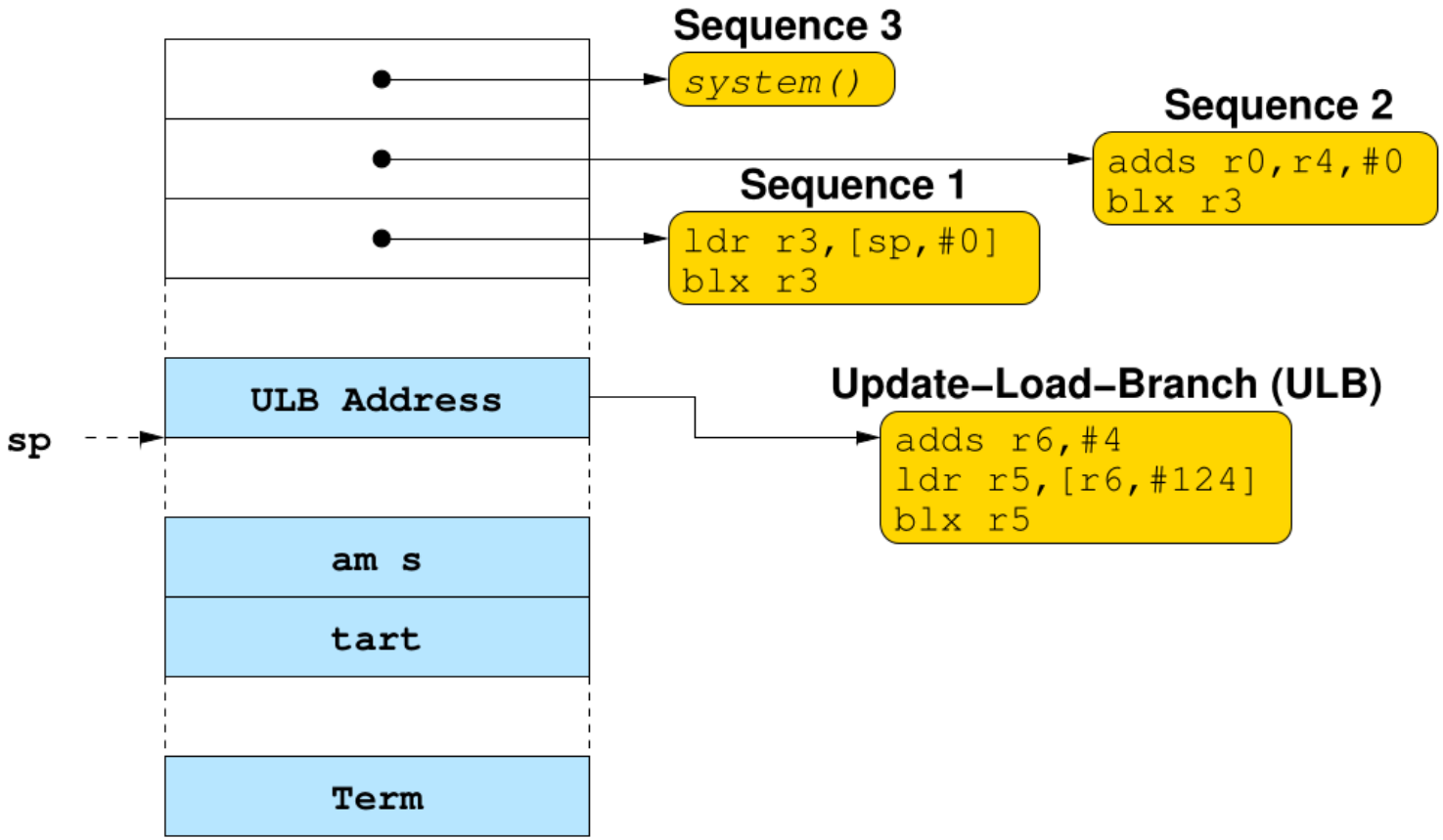


Ataques Concretos: Linux Intel x86

```
struct foo {
    char buffer[160];
    jmp_buf jb;
};

int main( int argc, char **argv ) {
    struct foo *f = malloc( sizeof *f );
    if( setjmp(f->jb) )
        return 0;
    strcpy( f->buffer, argv[1] );
    longjmp( f->jb, 1 );
}
```

Ataques Concretos: Google Android ARM



Obtener una shell en el emulador para el desarrollador de android.
Se utiliza el mismo código a traves de una interfaz de java para C.

Conclusiones

- Encontrar un trampolín basta para construir a su alrededor un conjunto de *gadgets*.
- Esta nueva forma de programar es un problema grave de seguridad.
- Las posibles mitigaciones pueden tener una gran penalidad.