Conception d'un compilateur obfuscateur

Frédéric Perriot

BeerRump 2016

Cadre du projet

- Dernier niveau pour le challenge du SSTIC (épreuve ring)
- "Whitebox" asymétrique (signature de Rabin)
- Source compilé par un compilateur maison
- cf. Fabien Perigaud pour l'algo spécifique et la solution

Bitslice

- Si je faisais une VM 32 bits...

Bitslice

- Si je faisais une VM 32 bits...
- 16 bits, ça serait mieux

Bitslice

- Si je faisais une VM 32 bits...
- 16 bits, ça serait mieux
- 8 bits, ça serait encore mieux

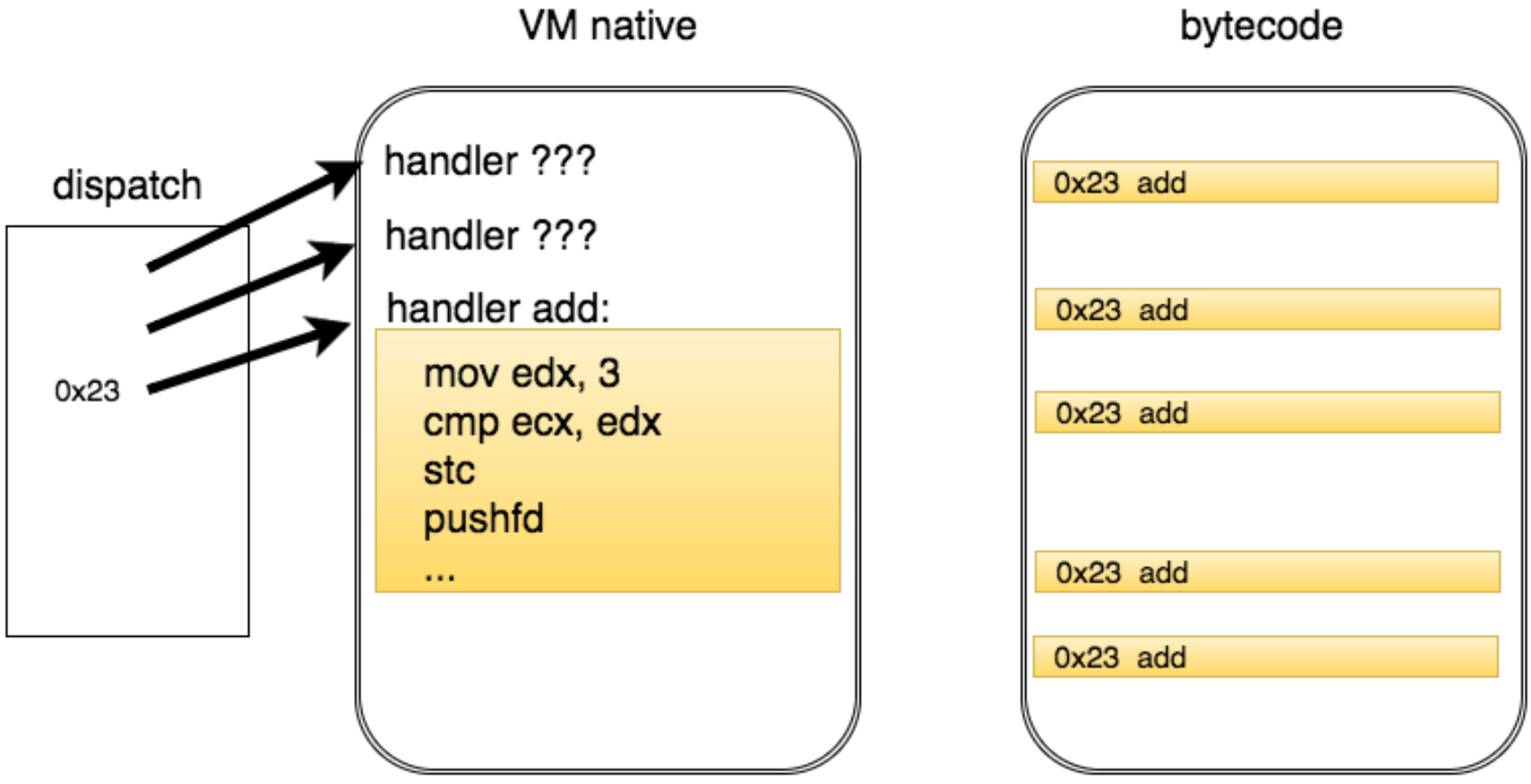
Bitslice

- Si je faisais une VM 32 bits...
- 16 bits, ça serait mieux
- 8 bits, ça serait encore mieux
- 1 bit, ça serait top

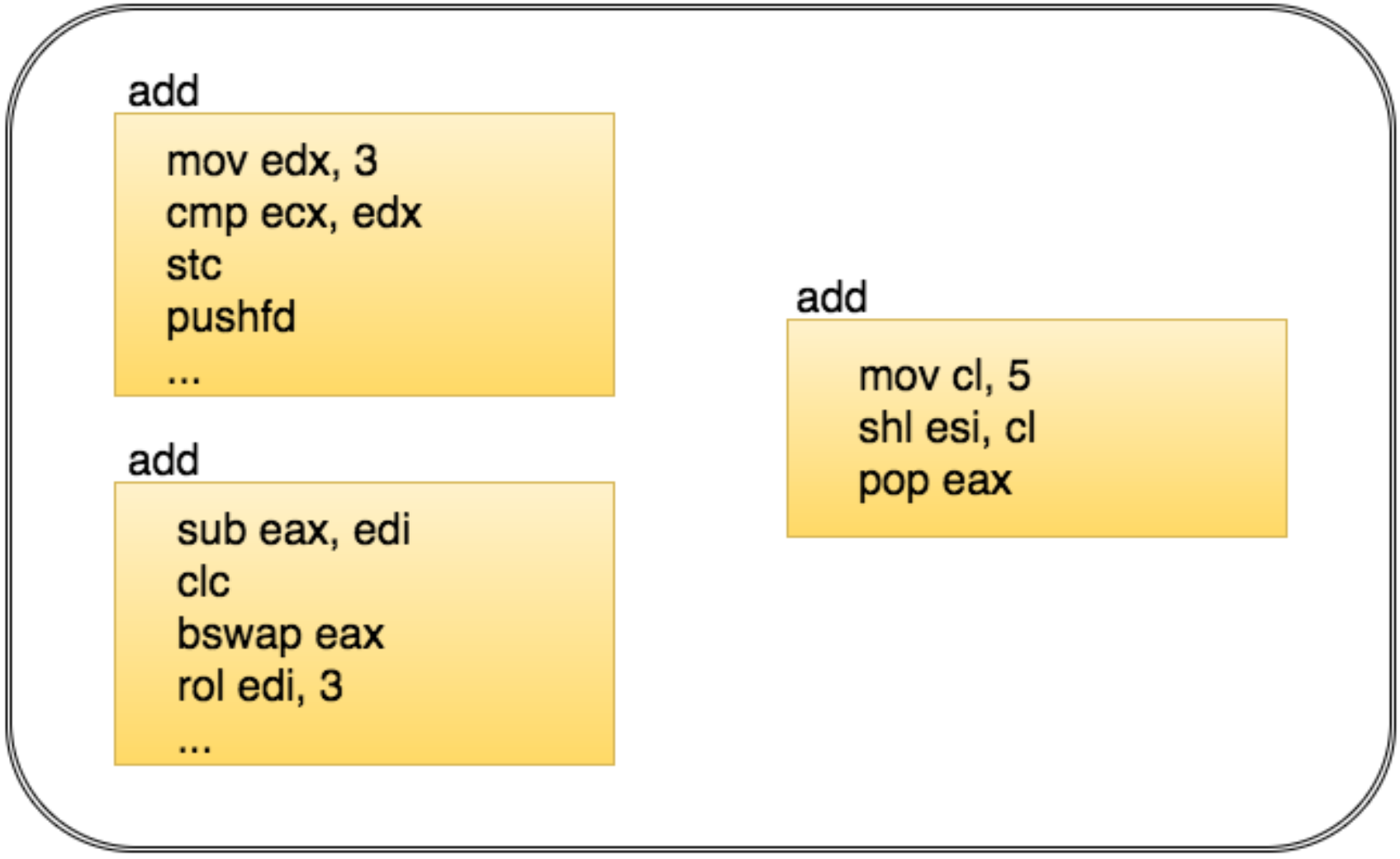
VM en "bitslice", mais dans un but d'obfuscation, pas de performance

Une VM mais pas une VM

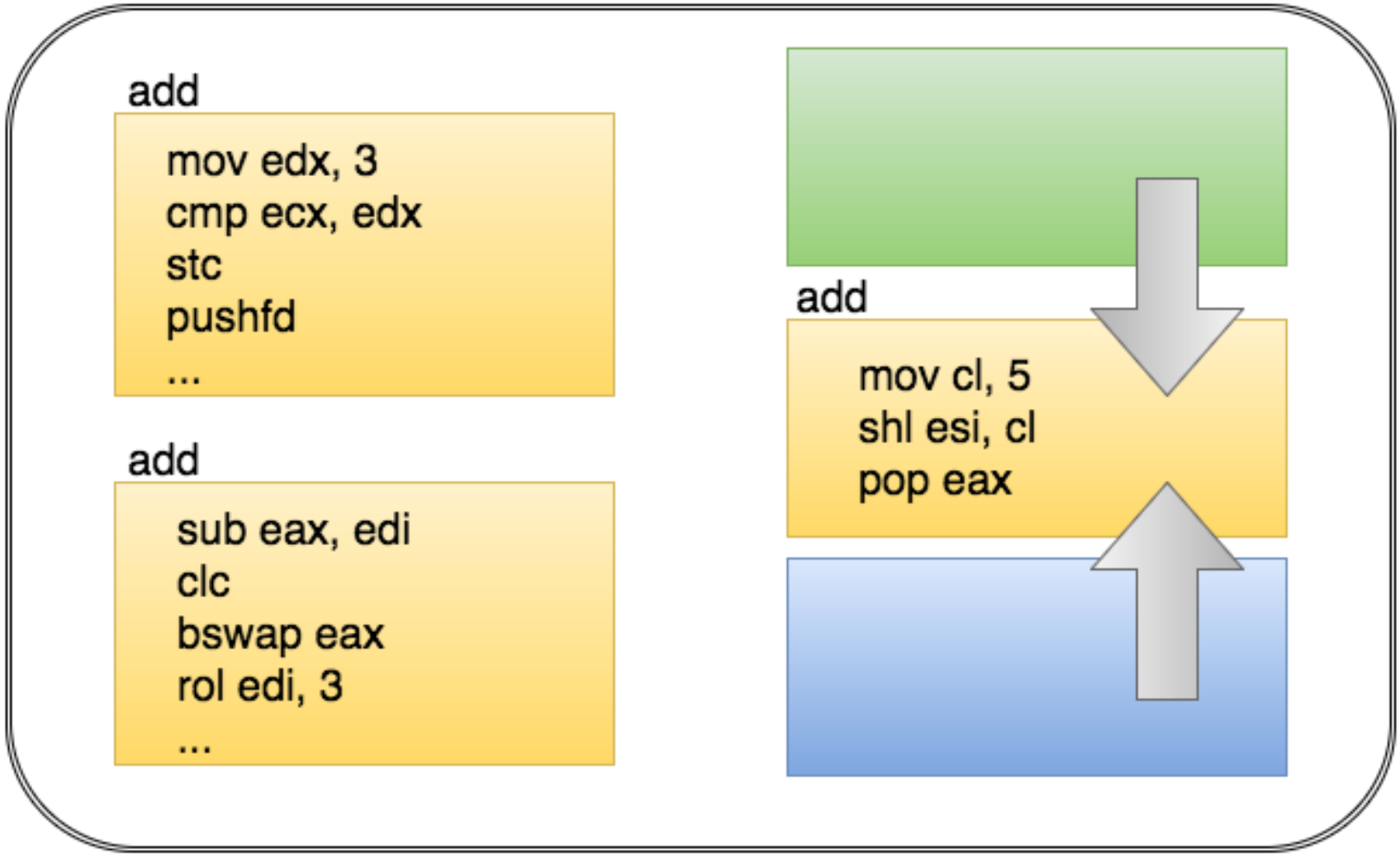
Compilateur > VM interprétée, les handlers d'opcodes sont inlinés



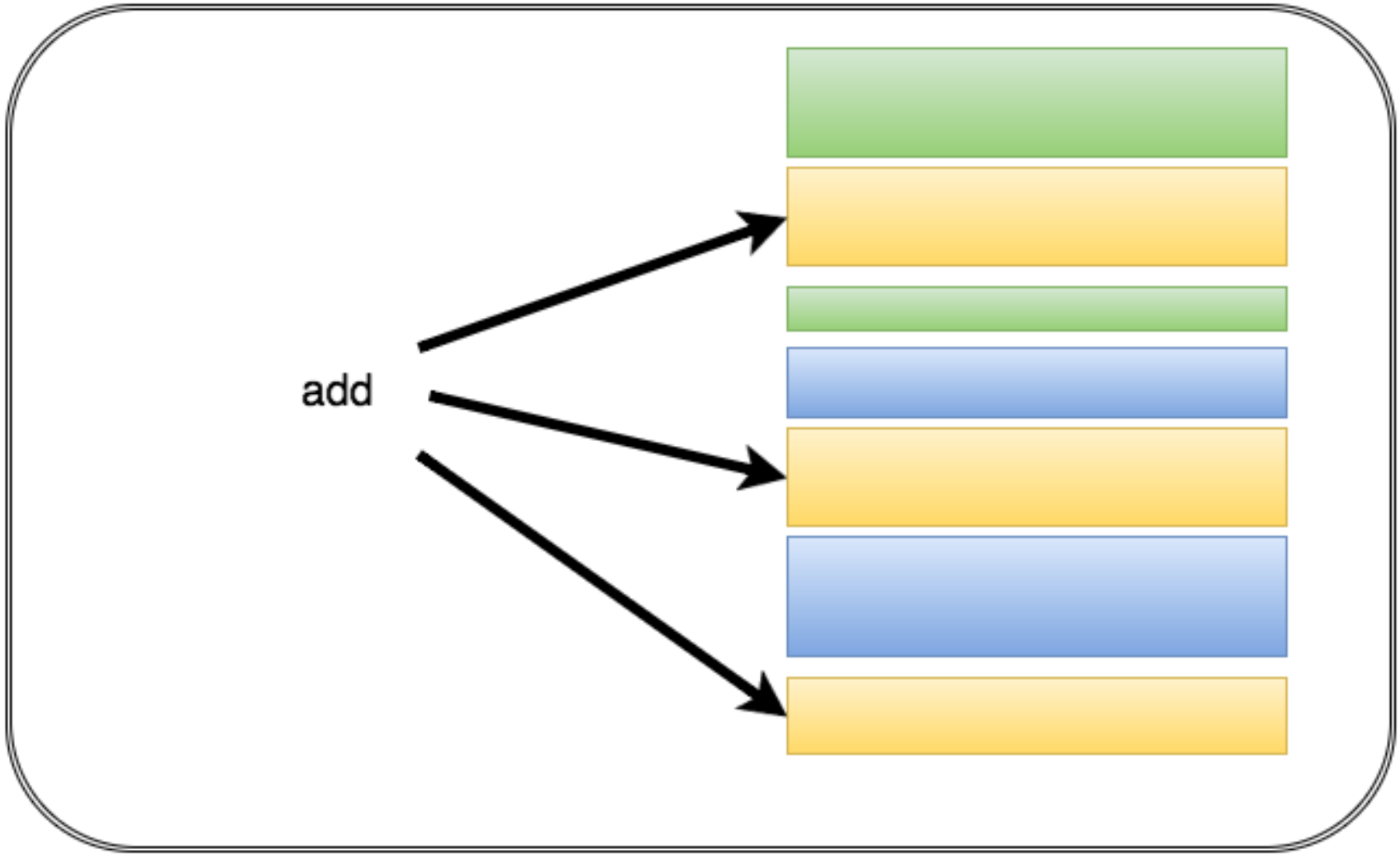
VM compilée


add
mov edx, 3
cmp ecx, edx
stc
pushfd
...
add
sub eax, edi
clc
bswap eax
rol edi, 3
...
add
mov cl, 5
shl esi, cl
pop eax

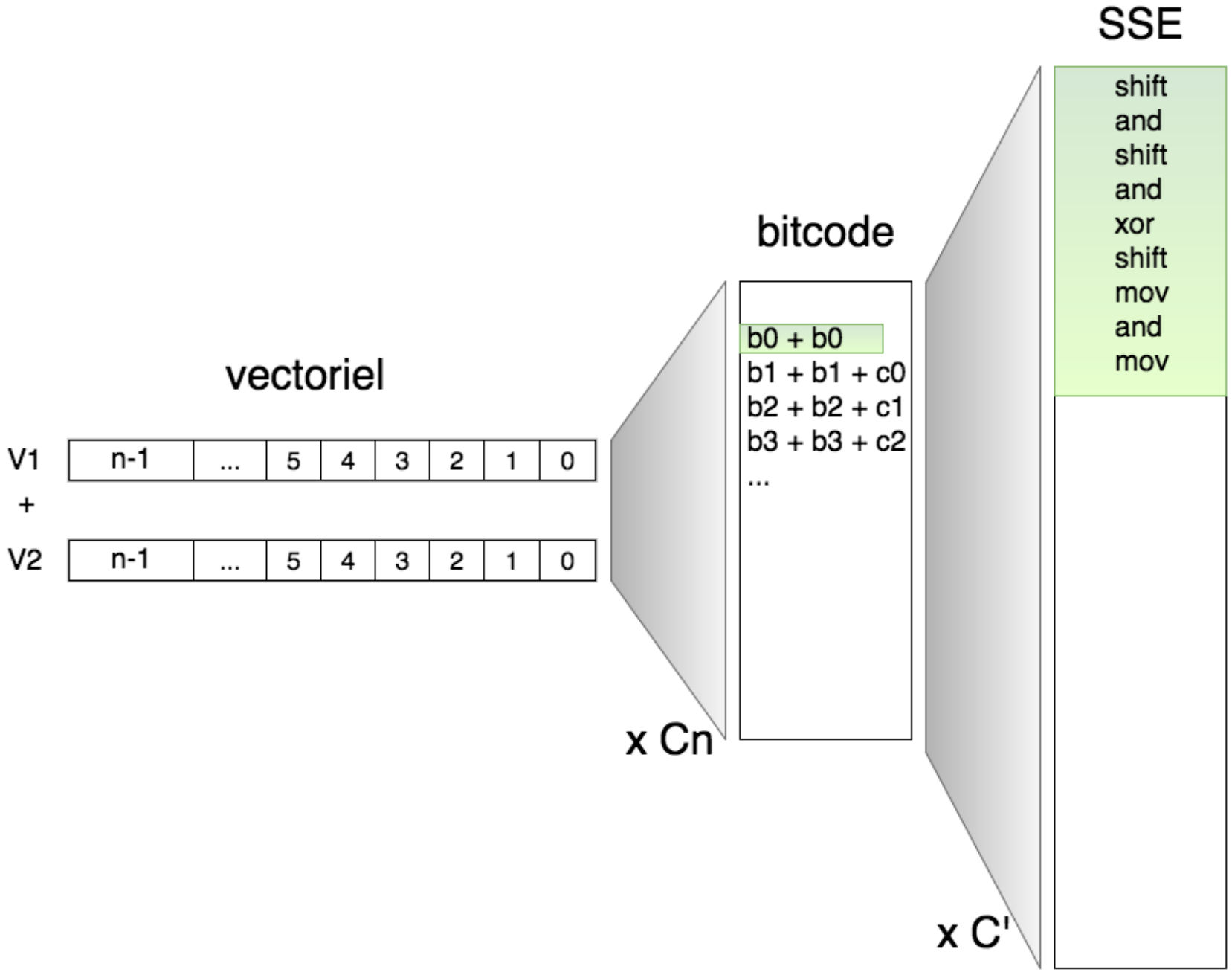
Shuffling


add
mov edx, 3
cmp ecx, edx
stc
pushfd
...
add
sub eax, edi
clc
bswap eax
rol edi, 3
...
add
mov cl, 5
shl esi, cl
pop eax

Shuffling



Facteurs d'expansion du code source



Le langage source vectoriel

Vecteurs de bits vus comme des entiers non signés de taille arbitraire

```
function kara38(u38 x, u38 y) -> u76;
{
   ...
   return (ac[0:38] # z38) +
          (qsx # q # z19)  +
          (z36 # bd);
}
```

Le langage intermédiaire ("bitcode")

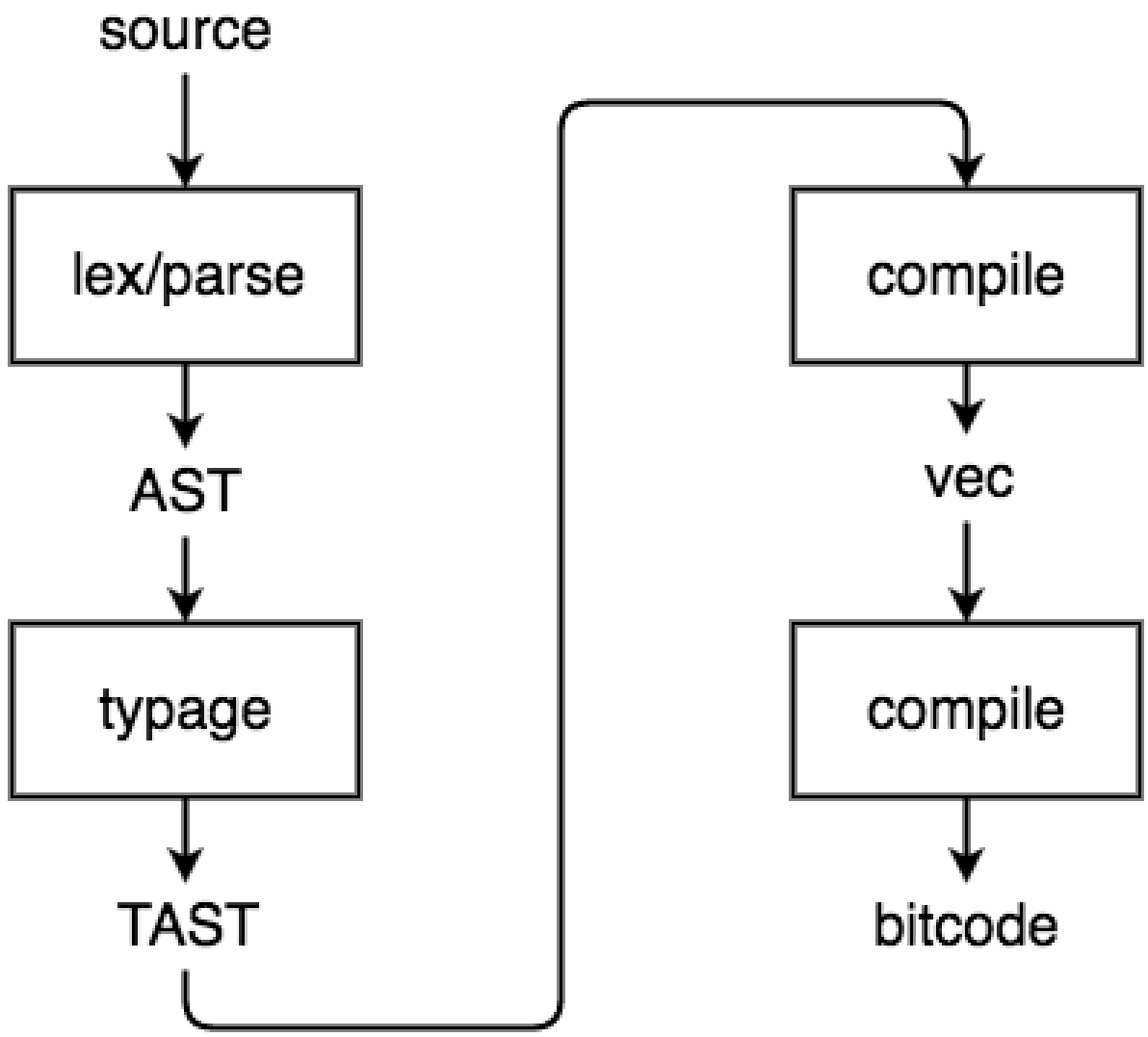
```
type insn =
  | Nop
  | Mov of dst * src
  | Xor of dst * src * src
  | Or of dst * src * src
  | And of dst * src * src
  | Not of dst * src
  | Ld0 of dst
  | Ld1 of dst
  | Ldmem of dst * ptr
  | Stmem of ptr * src
  | Jmp of block
  | Jz of src * block
  | Jnz of src * block
  | Call of funcname * dst list * src list
  | Ldarg of dst * arg
  | Return of src list
```

Le langage de sortie

- x64 (˜30 instructions) + SSE2 (˜20 instructions)

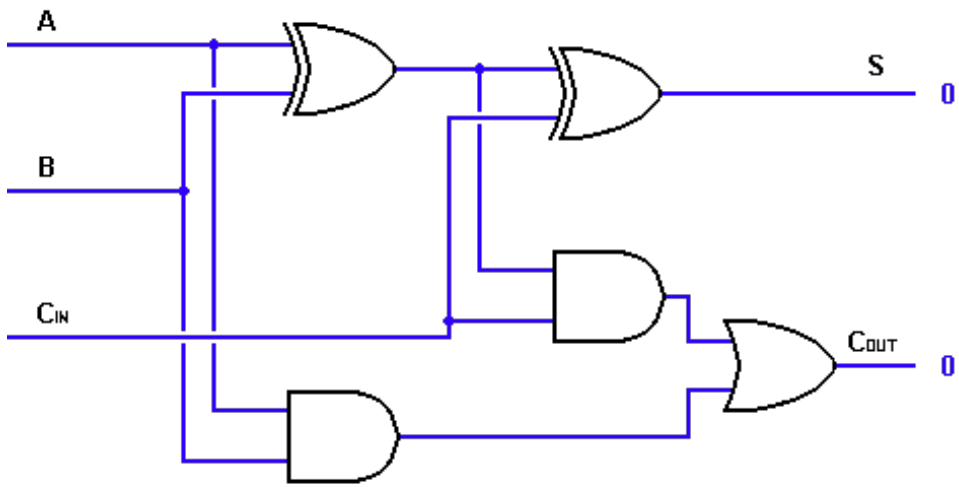
```
pshufb   xmm5, xmm1
movdqa   xmm2, xmm3
psrlq    xmm5, 6
pandn    xmm2, xmm8
pand     xmm3, xmm5
mov      di, 0x3151
por      xmm2, xmm3
mov      rdx, 0x4849686088451088
mov      rax, [rbp+24]
ror      rax, 59
movdqa   [rsp], xmm8
movdqa   xmm8, xmm2
pinsrw   xmm2, di, 4
```

Phases du Front end

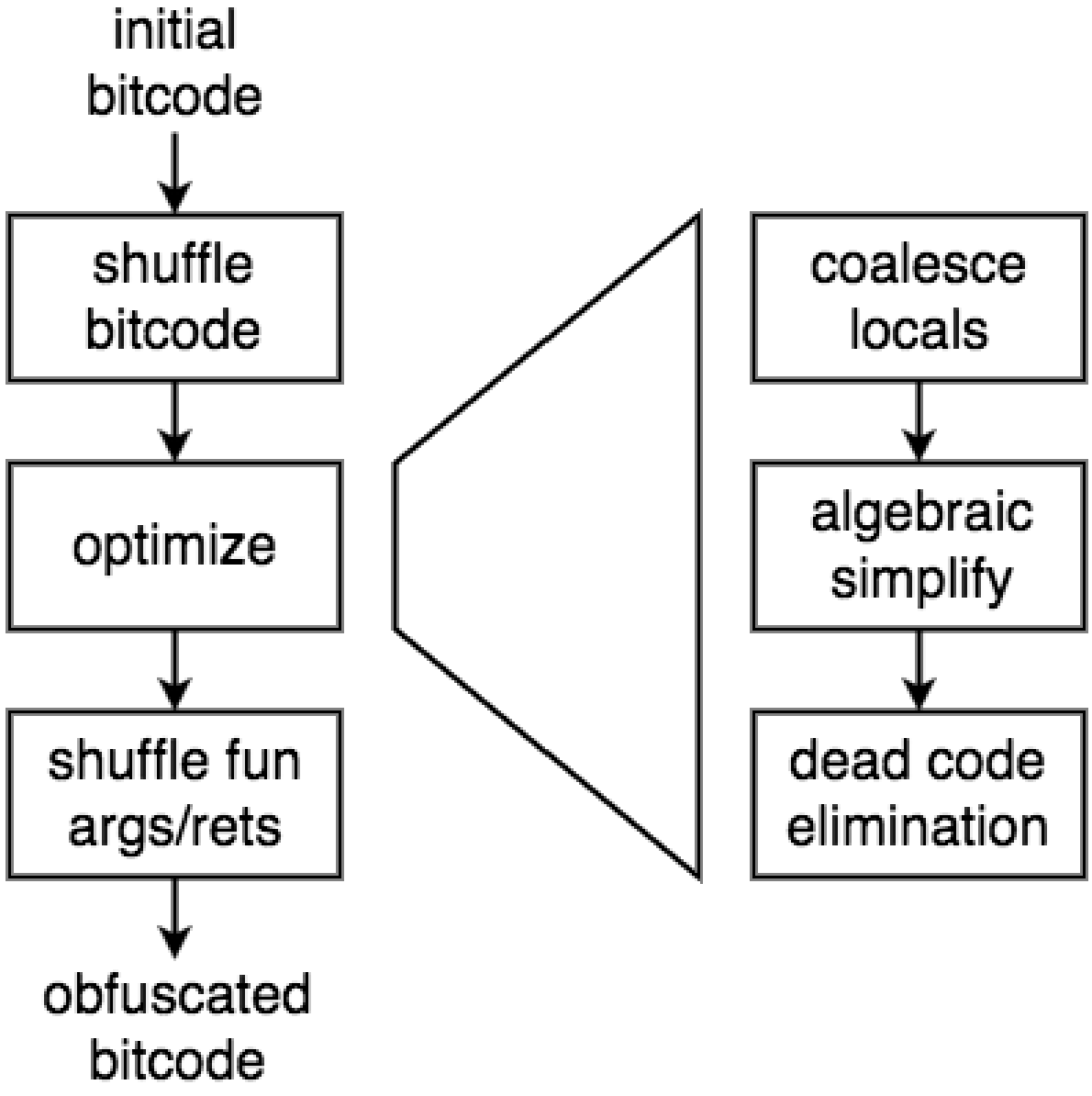


Vectoriel vers bitcode

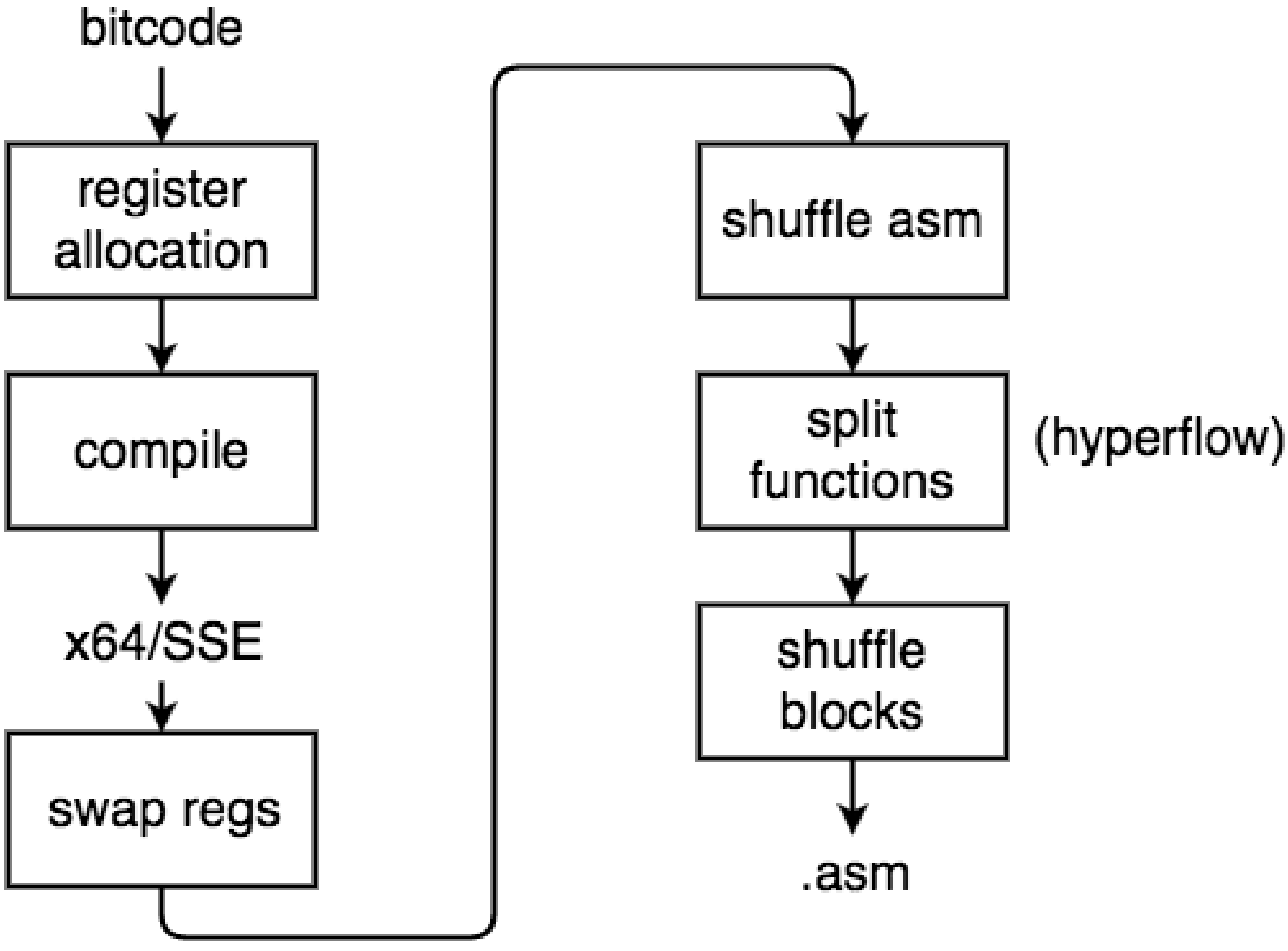
- and, or, xor, not, shift, extract, concat: parallélisme trivial
- add est le seul cas intéressant (circuit full adder)



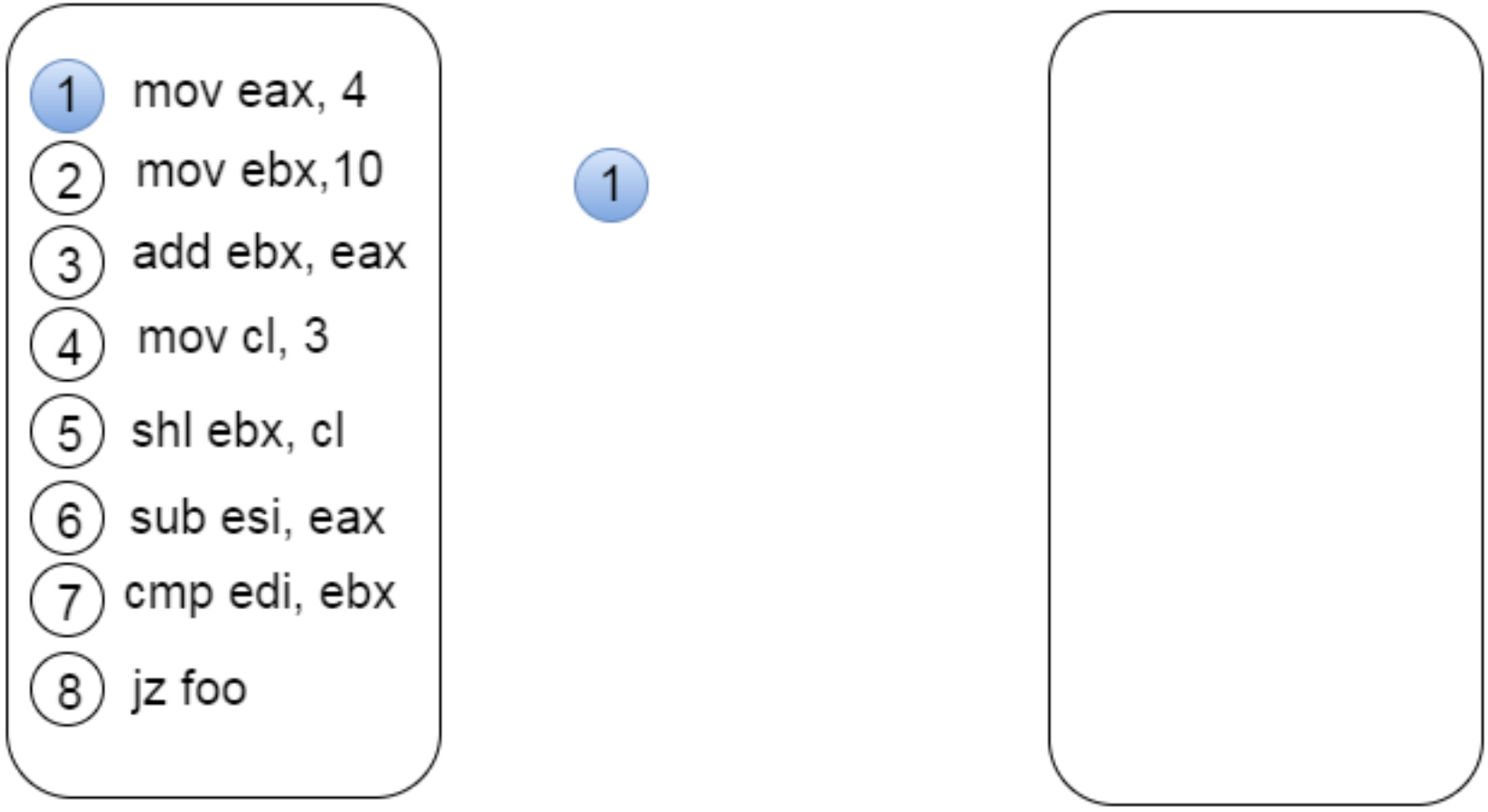
Phases du Middle end



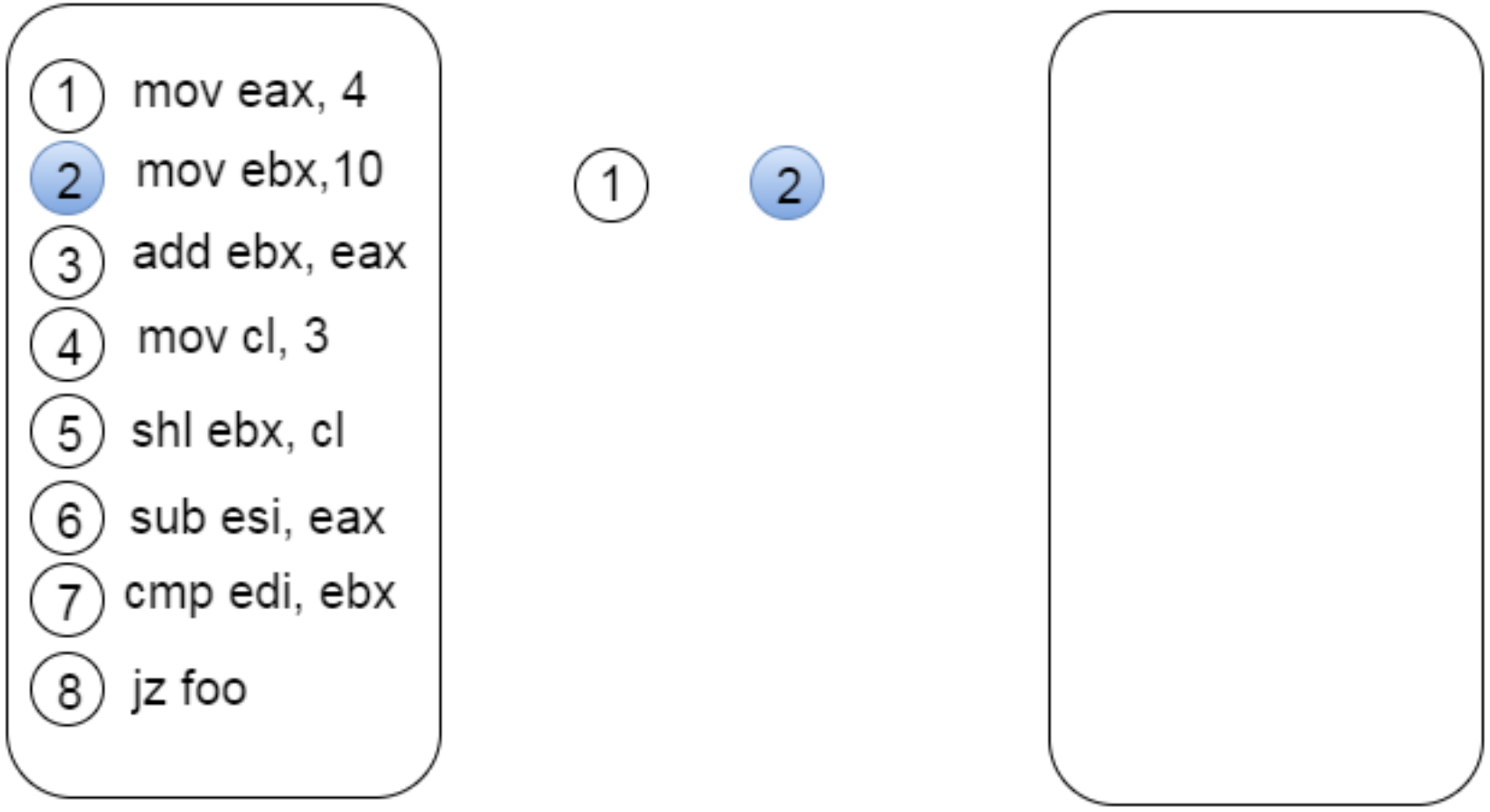
Phases du Back end



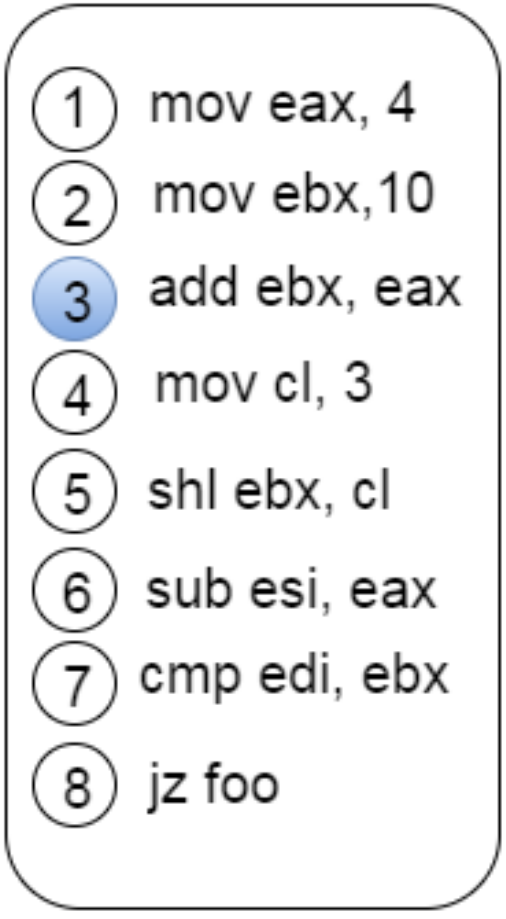
Graphe de dépendance et shuffling

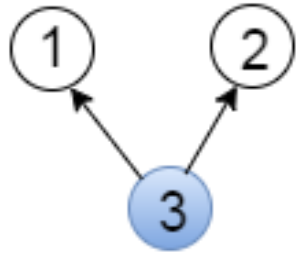


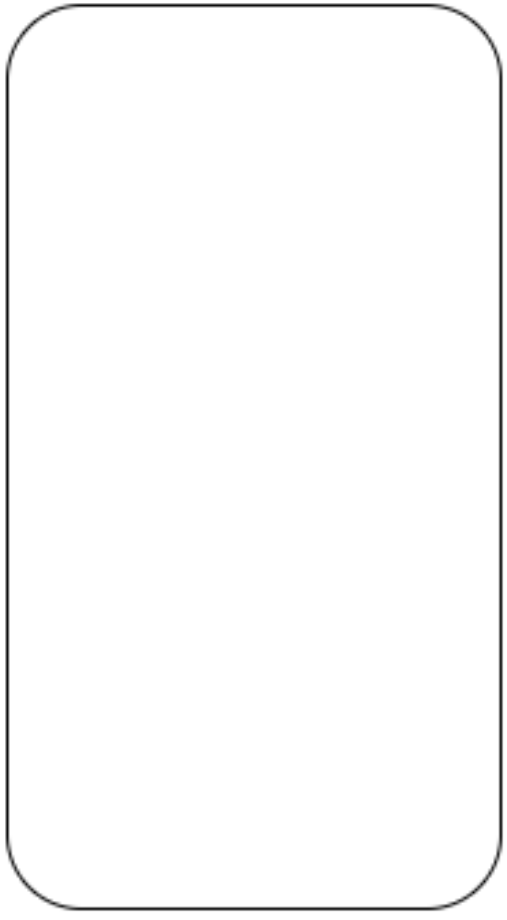
Graphe de dépendance et shuffling



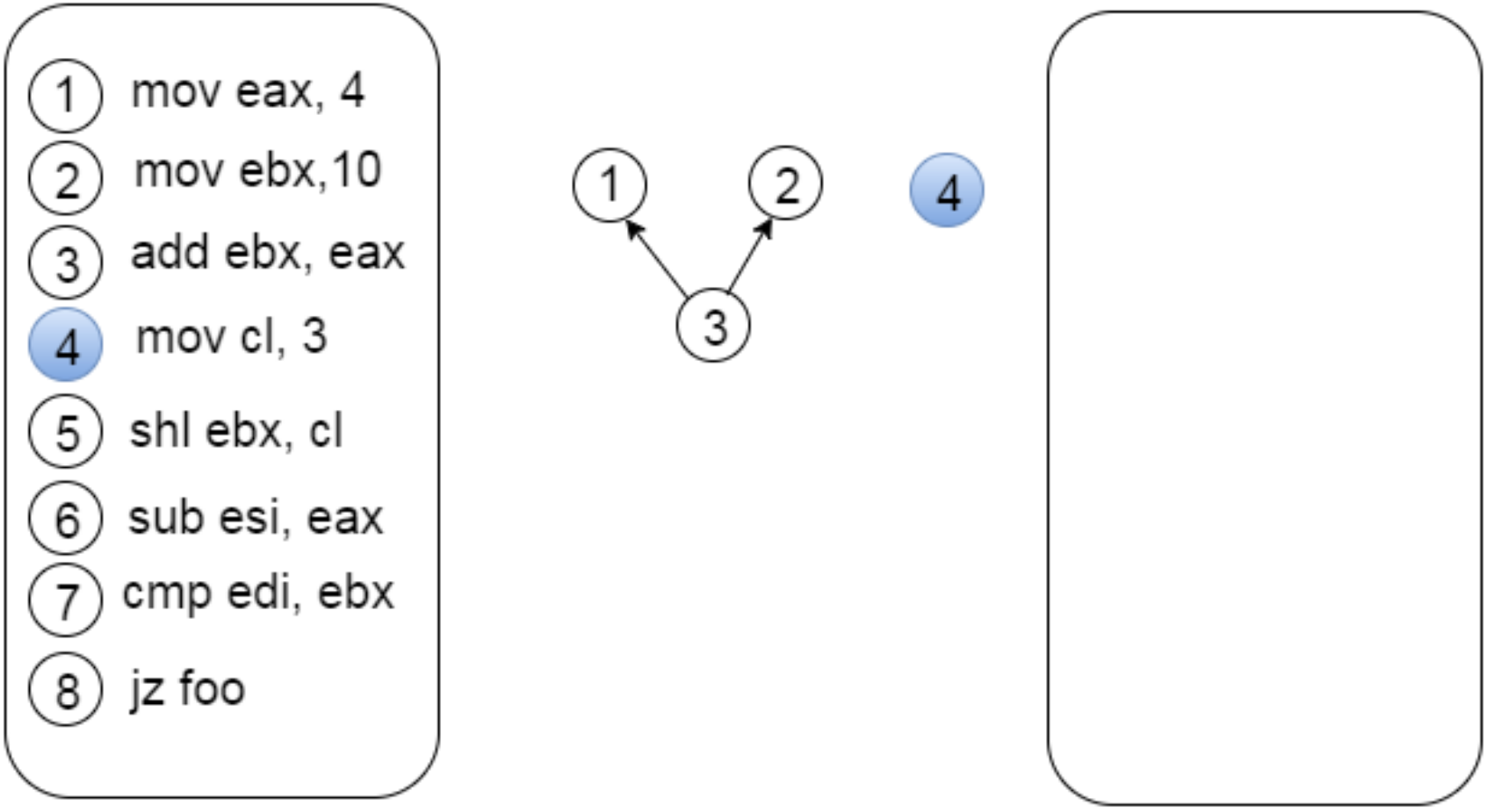
Graphe de dépendance et shuffling





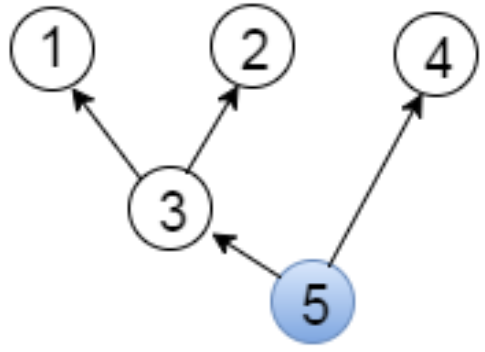


Graphe de dépendance et shuffling



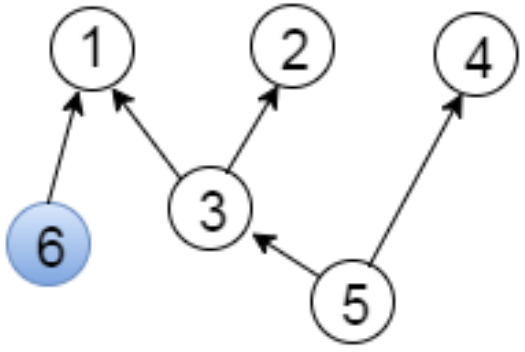
Graphe de dépendance et shuffling

1. mov eax, 4
2. mov ebx,10
3. add ebx, eax
4. mov cl, 3
5. shl ebx, cl
6. sub esi, eax
7. cmp edi, ebx
8. jz foo



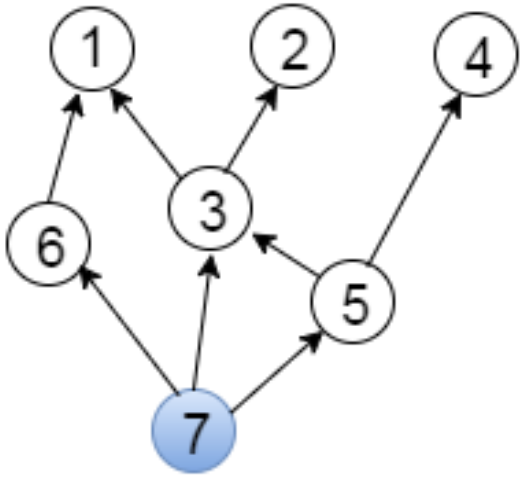
Graphe de dépendance et shuffling

1. mov eax, 4
2. mov ebx,10
3. add ebx, eax
4. mov cl, 3
5. shl ebx, cl
6. sub esi, eax
7. cmp edi, ebx
8. jz foo



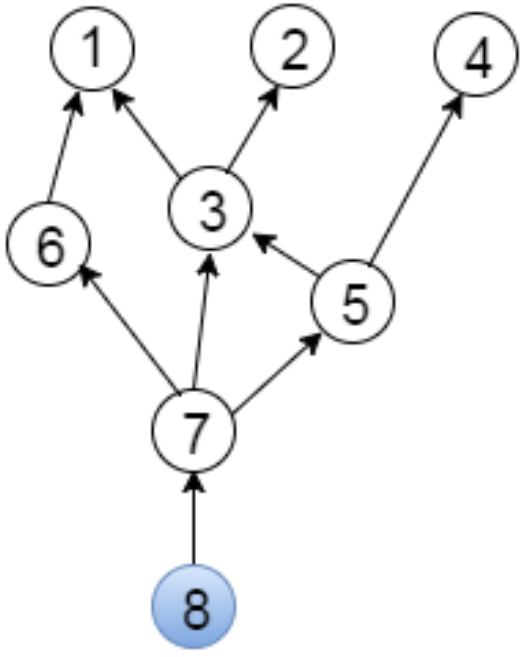
Graphe de dépendance et shuffling

1. mov eax, 4
2. mov ebx,10
3. add ebx, eax
4. mov cl, 3
5. shl ebx, cl
6. sub esi, eax
7. cmp edi, ebx
8. jz foo



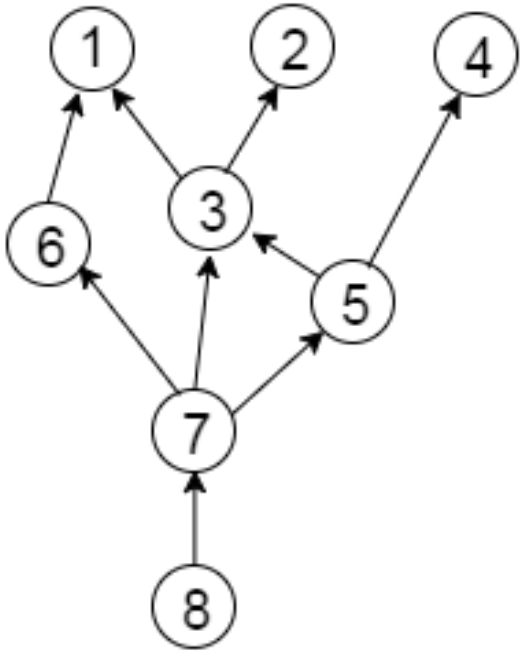
Graphe de dépendance et shuffling

1. mov eax, 4
2. mov ebx,10
3. add ebx, eax
4. mov cl, 3
5. shl ebx, cl
6. sub esi, eax
7. cmp edi, ebx
8. jz foo



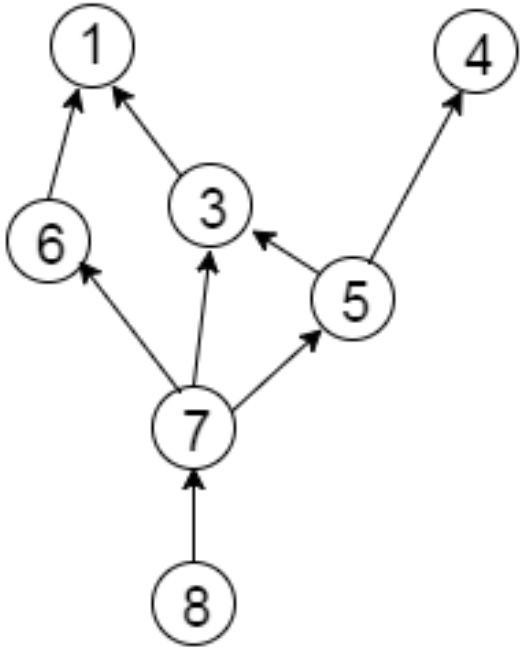
Graphe de dépendance et shuffling

1. mov eax, 4
2. mov ebx,10
3. add ebx, eax
4. mov cl, 3
5. shl ebx, cl
6. sub esi, eax
7. cmp edi, ebx
8. jz foo



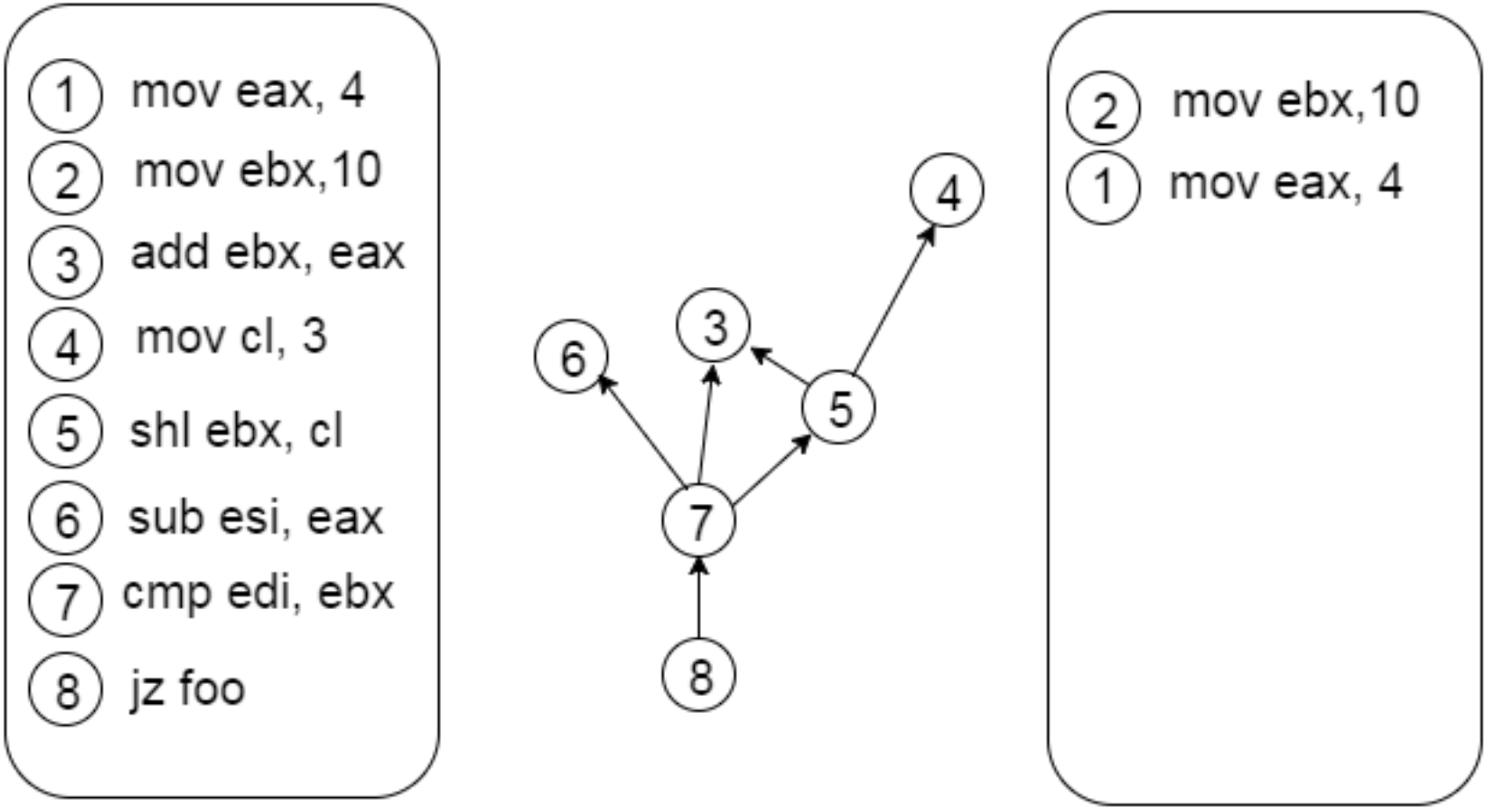
Graphe de dépendance et shuffling

1. mov eax, 4
2. mov ebx,10
3. add ebx, eax
4. mov cl, 3
5. shl ebx, cl
6. sub esi, eax
7. cmp edi, ebx
8. jz foo

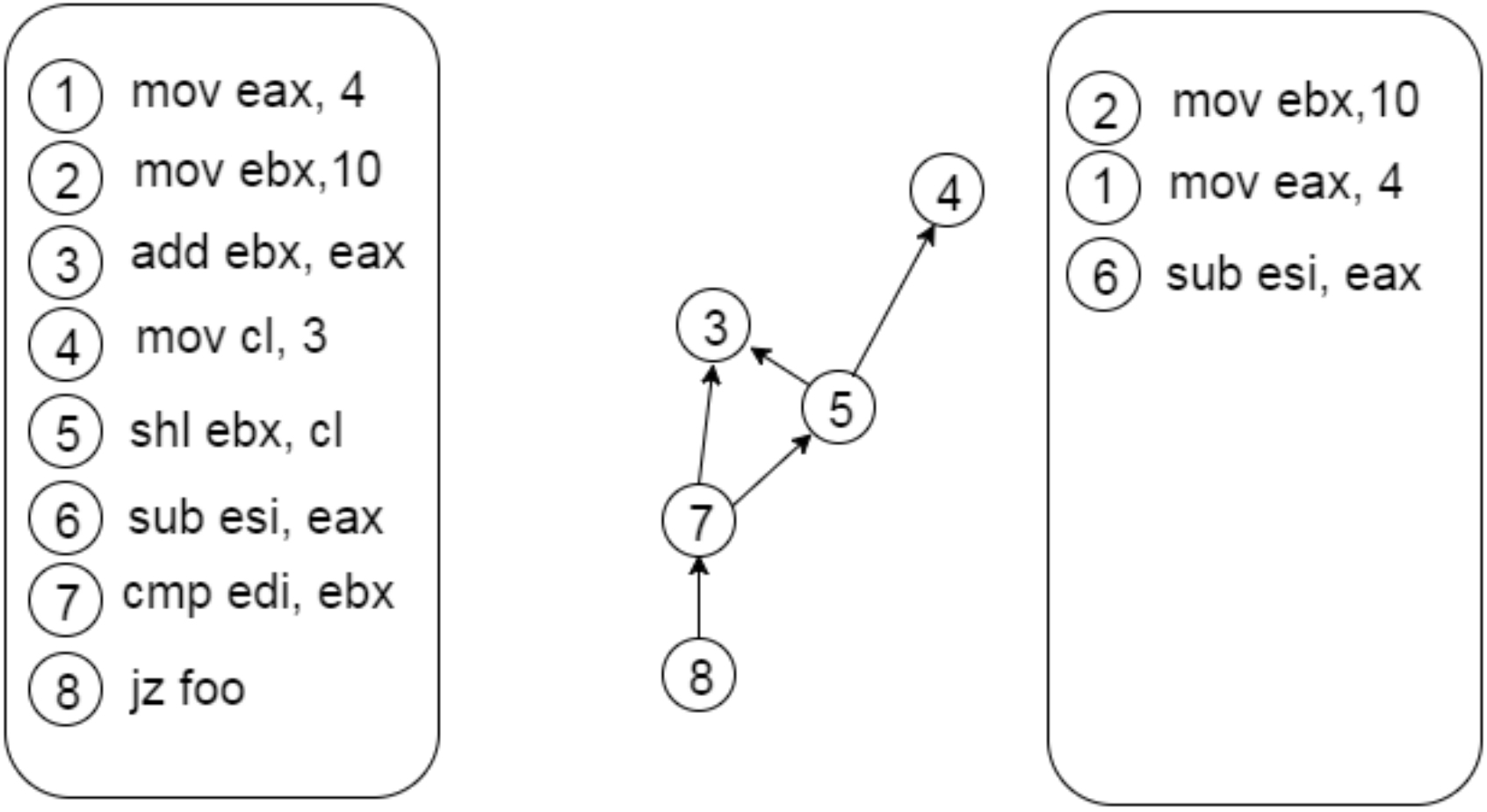


2. mov ebx,10

Graphe de dépendance et shuffling

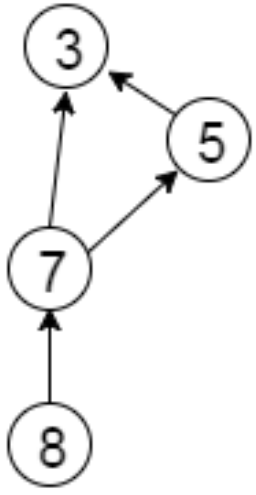


Graphe de dépendance et shuffling



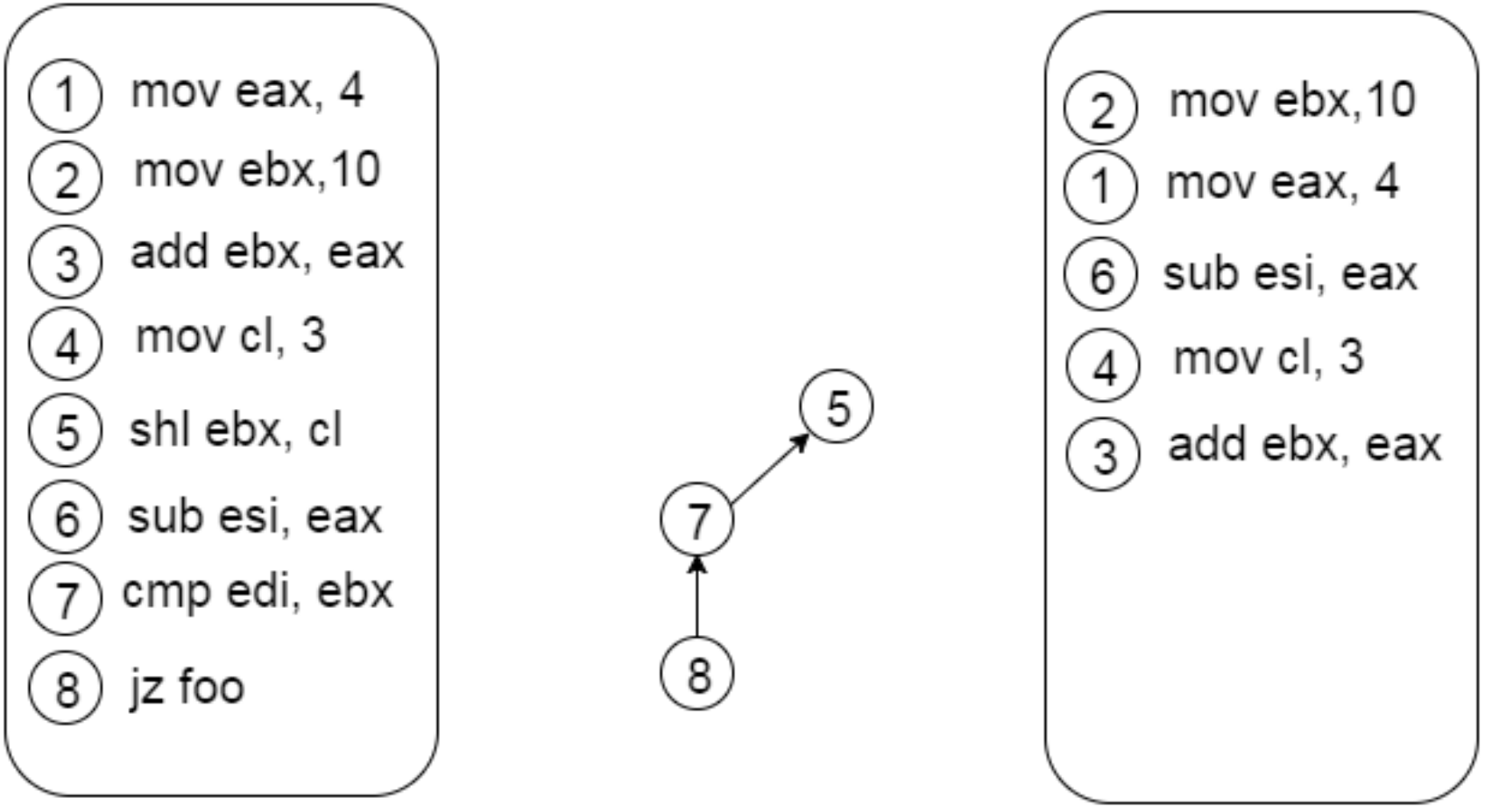
Graphe de dépendance et shuffling

1. mov eax, 4
2. mov ebx,10
3. add ebx, eax
4. mov cl, 3
5. shl ebx, cl
6. sub esi, eax
7. cmp edi, ebx
8. jz foo

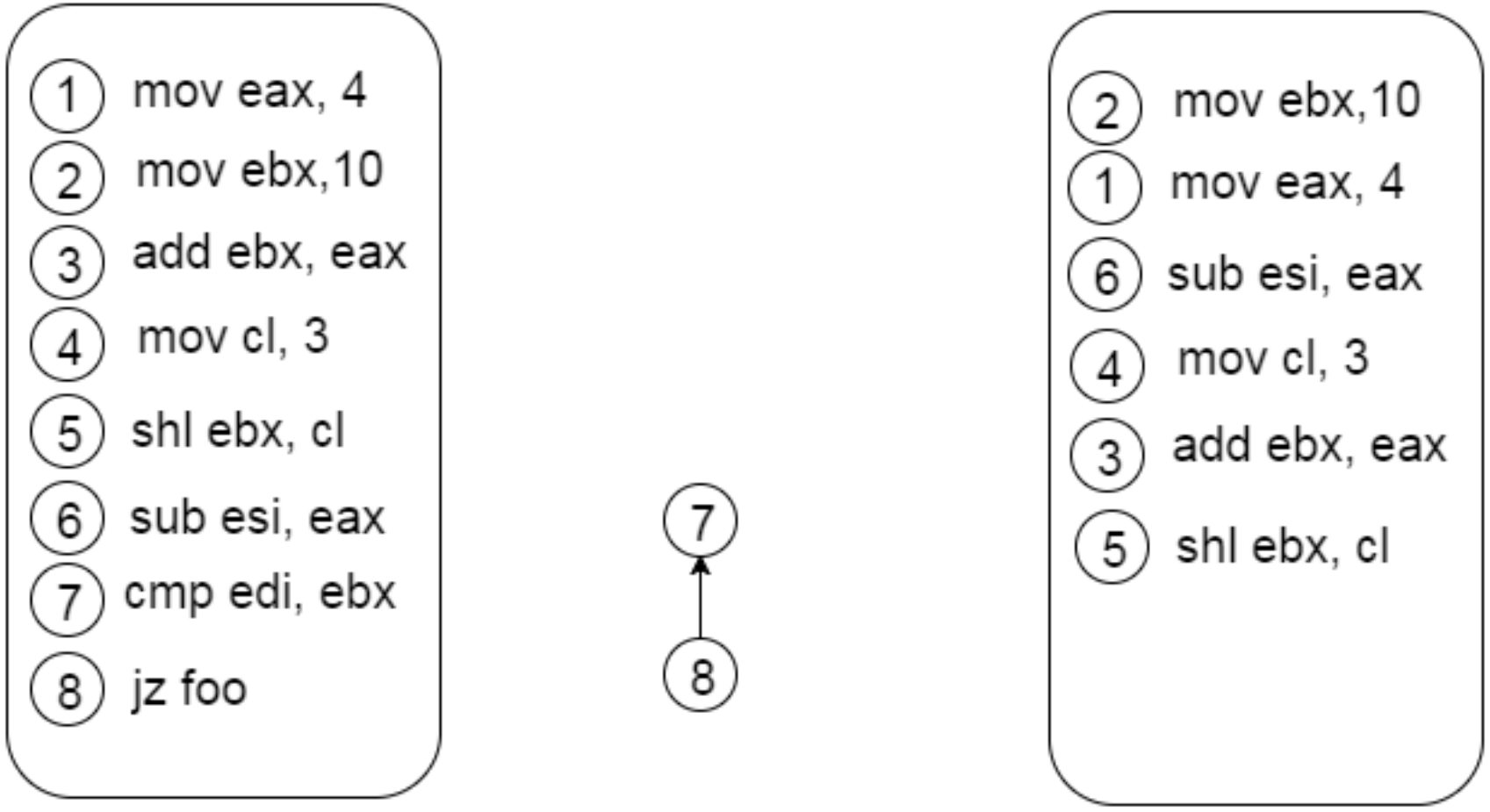


2. mov ebx,10
1. mov eax, 4
6. sub esi, eax
4. mov cl, 3

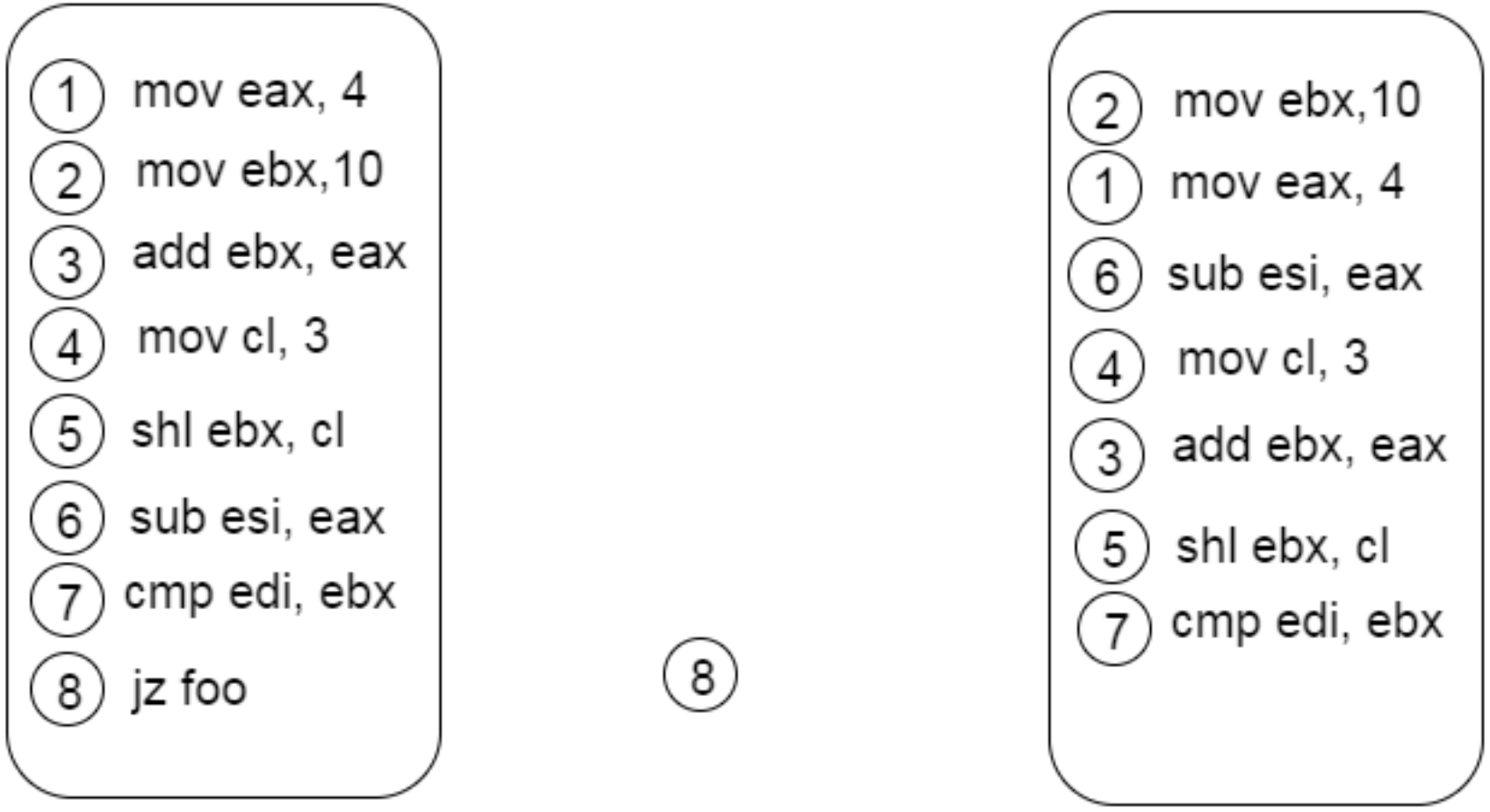
Graphe de dépendance et shuffling



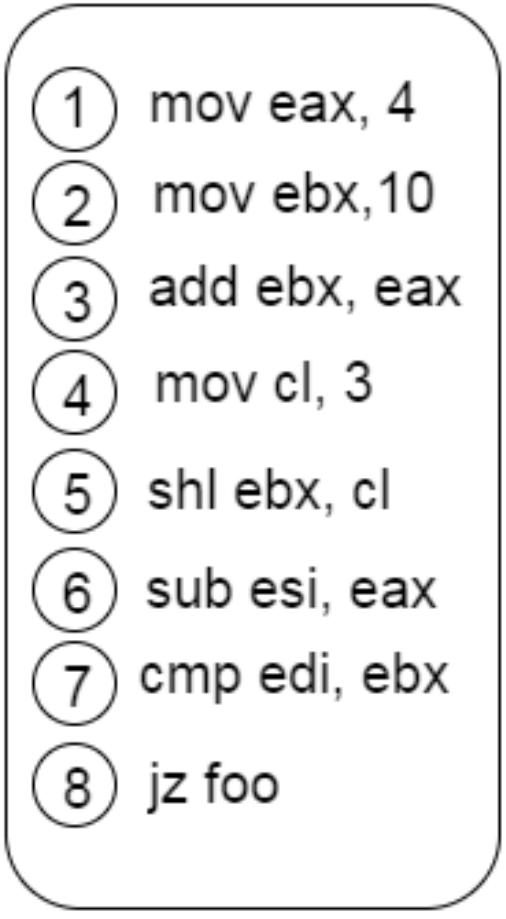
Graphe de dépendance et shuffling

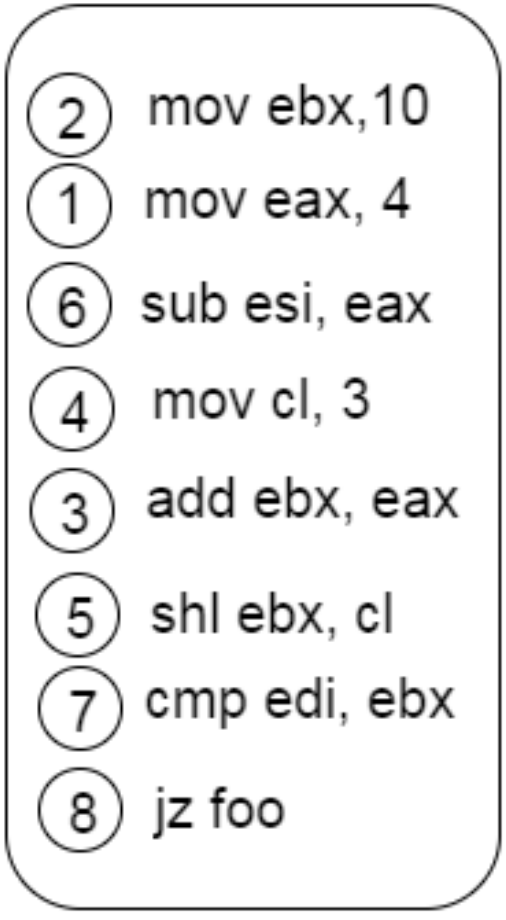


Graphe de dépendance et shuffling



Graphe de dépendance et shuffling





Graphe de dépendance et shuffling

Deux problèmes:

- Taille du graphe quadratique en la taille du bloc
- Equiprobabilité des sorties valides?

Options du compilateur (1)

```
Usage: hlvc [options] <file.hlv> <file.asm>
Options:
 -r mask: randomization options mask
        1 = randomize register masks
        2 = randomize scratch registers
        4 = shuffle instructions in assembly blocks
        8 = randomize opcode choice
       16 = shuffle instructions in bitcode blocks
       32 = shuffle function arguments/returns
       64 = randomize locals registers
      128 = dynamically swap registers
      256 = shuffle code blocks
```

Options du compilateur (2)

```
-o mask: optimization options mask
        1 = coalesce non-interfering locals
        2 = algebraic simplifications
        4 = dead code elimination
-c mask: code generation options mask
        1 = hyperflow
        2 = produce trace
-bc <file.bc>: save bitcode to file
```

Merci!