Laboratoire Exploration et recherche en Détection

LED

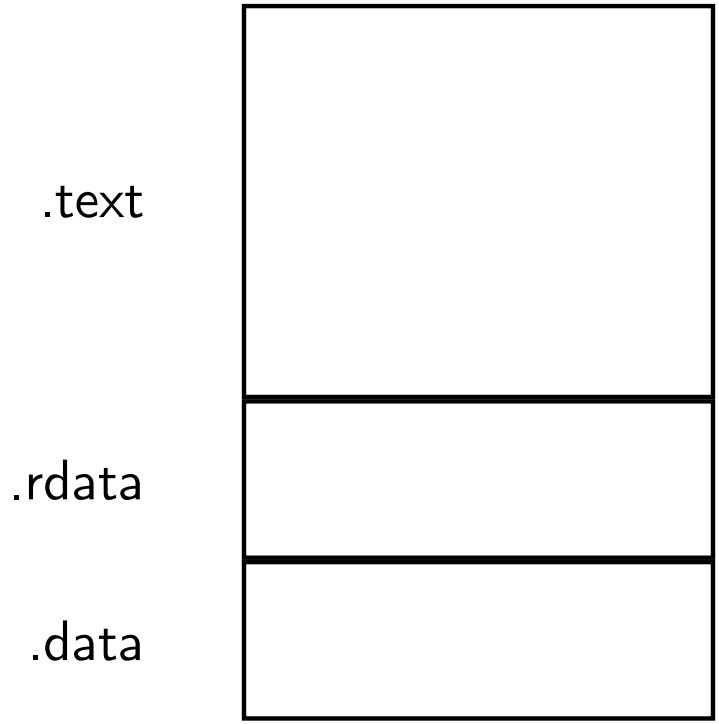
Agence Nationale de la Sécurité des Systèmes d'Information

2019

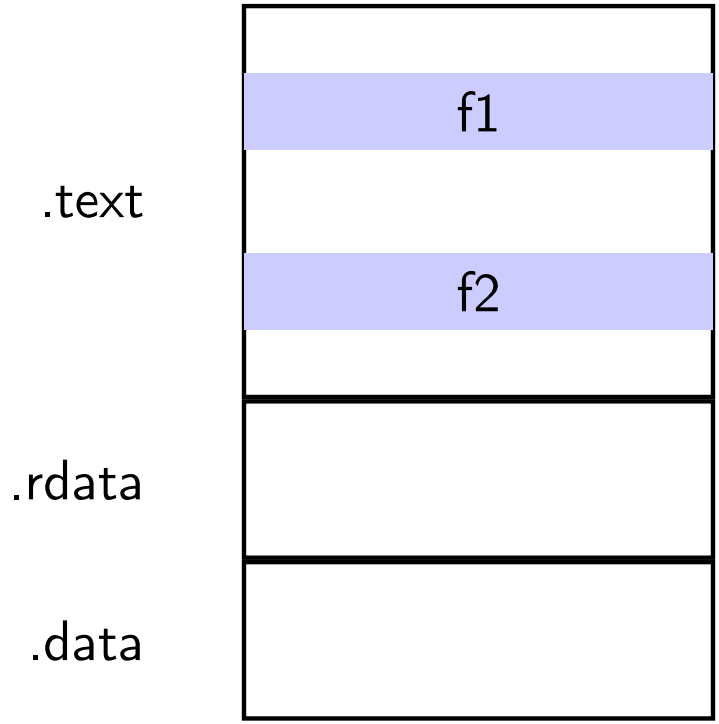
VMProtect

- Compression
- Imports protection
- Code mutation
- ...
- VIRTUALIZATION

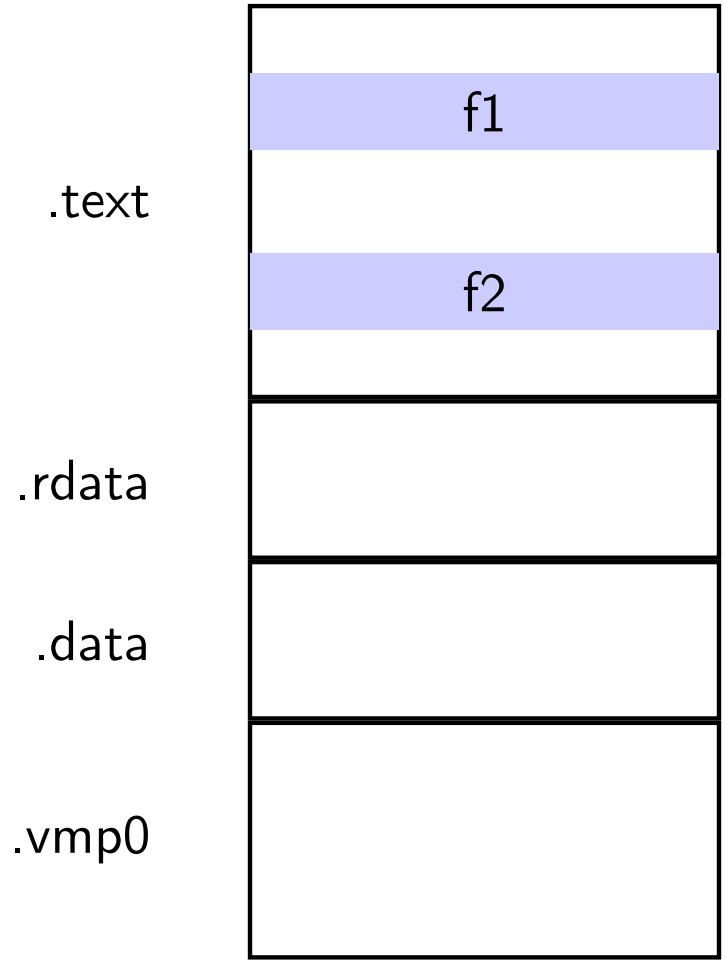
File geometry



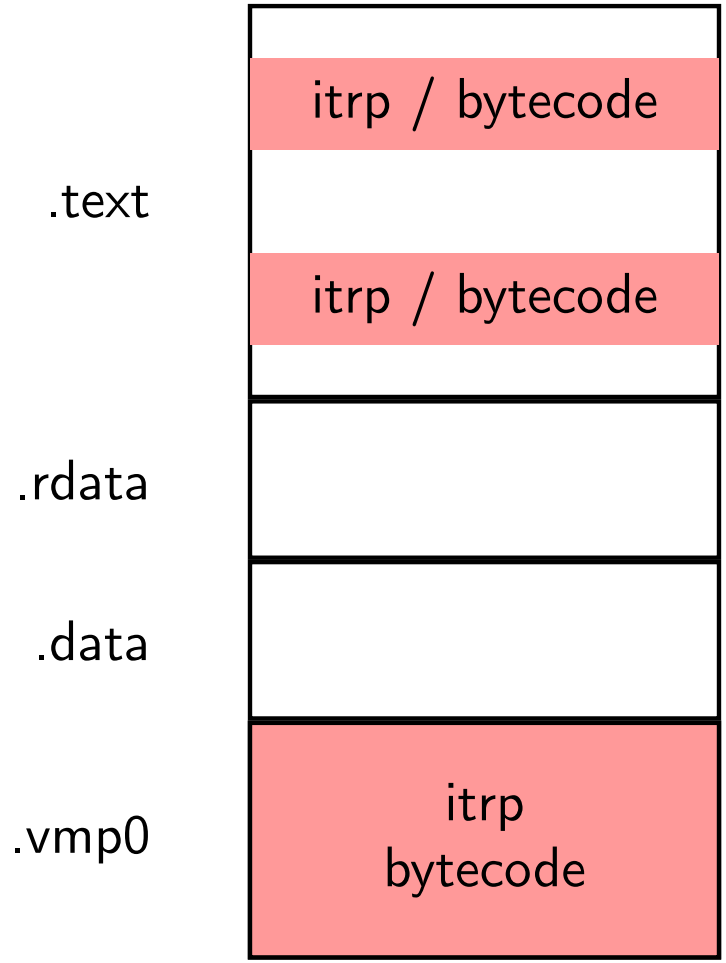
File geometry



File geometry



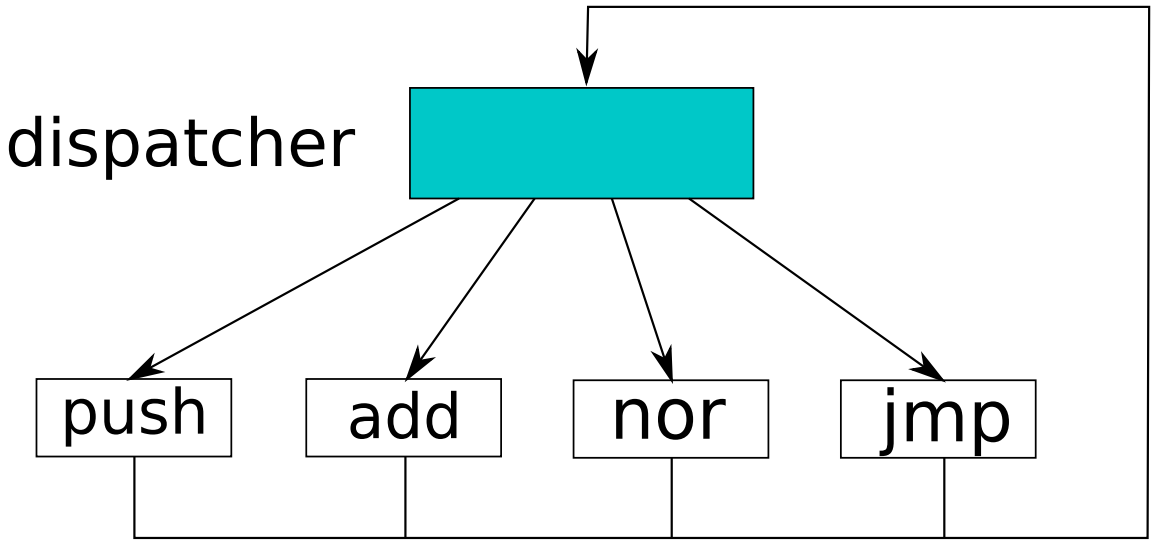
File geometry



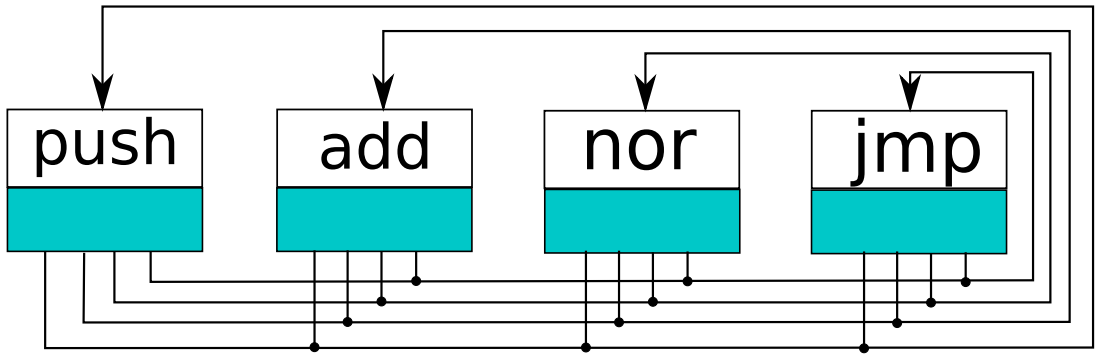
Interpreter code is obfuscated

```
.vmp0 :004047D3        and     ah, dl              (junk)
.vmp0 :004047D5        mov     eax, [ebp+0]
.vmp0 :004047D8        sal     dl, 3               (junk)
.vmp0 :004047DB        clc                         (junk)
.vmp0 :004047DC        mov     dh, 0E3h            (junk)
.vmp0 :004047DF        test    esp, eax            (junk)
.vmp0 :004047E1        mov     edx, [ebp+4]
.vmp0 :004047E4        test    eax, 0A4B7FFF0h     (junk)
.vmp0 :004047E9        test    bh, bh              (junk)
.vmp0 :004047EB        pushf                       (junk)
.vmp0 :004047EC        not     eax
.vmp0 :004047EE        test    cl, 5Ch             (junk)
.vmp0 :004047F1        clc                         (junk)
.vmp0 :004047F2        stc                         (junk)
.vmp0 :004047F3        not     edx
.vmp0 :004047F5        pushf                       (junk)
.vmp0 :004047F6        jmp     loc 4053BF
```

V2 Interpreter



V3 Interpreter



Native registers roles

	x86	x64
vip	esi	rsi
vsp	ebp	rbp
virtual registers	edi (x 16)	rdi (x 24)
decryption mask	ebx	rbx
opcode (@handler entry)	eax	rax

VM instruction set

Most important invariant across samples

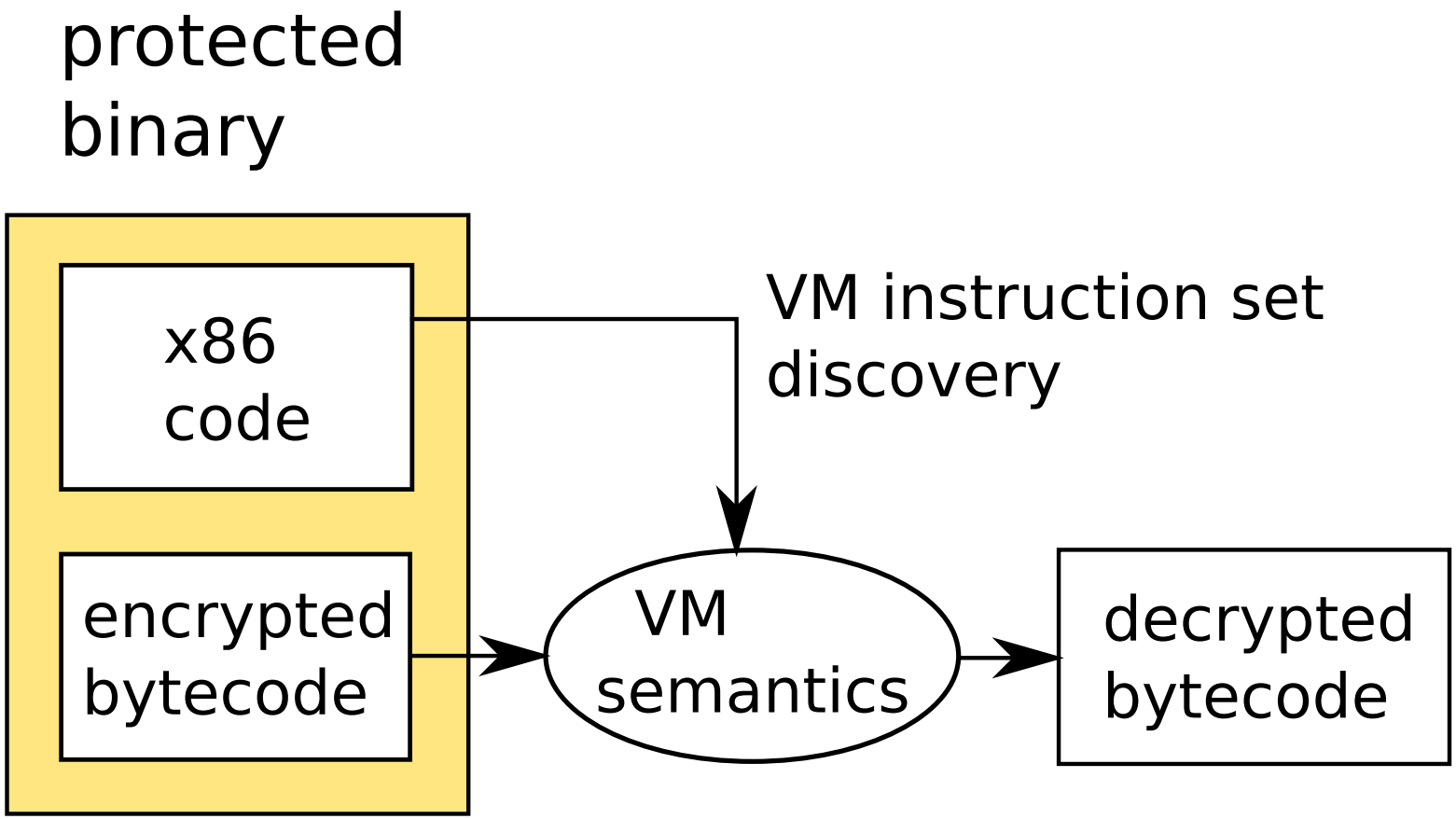
- Based on the same template
- Opcodes permutated
- Bytecode encryption varies

Who does what ?

Bytecode extraction

- Locate dispatcher / handlers
- Symbolic execution
- Extract decryption formulas
- Recognize opcodes by their effect on the virtual stack

Front-end


protected
binary
x86
code
encrypted
bytecode
VM instruction set
discovery
VM
semantics
decrypted
bytecode

Bytecode instruction set

Stack-oriented machine

- push X, pop Y, load, store

Bytecode instruction set

Stack-oriented machine

- push X, pop Y, load, store
- ALU : add, nor, shifts, mul, div

Bytecode instruction set

Stack-oriented machine

- push X, pop Y, load, store
- ALU : add, nor, shifts, mul, div
- Control flow : jmp

Bytecode instruction set

Stack-oriented machine

- push X, pop Y, load, store
- ALU : add, nor, shifts, mul, div
- Control flow : jmp
- VM exits : fixed-arity call, exitvm

Bytecode instruction set

Stack-oriented machine

- push X, pop Y, load, store
- ALU : add, nor, shifts, mul, div
- Control flow : jmp
- VM exits : fixed-arity call, exitvm
- misc : cpuid, rdtsc, FPU, checksum...

Operators expansion

operation	translates into
`not a`	`nor a a`
`and a b`	`nor (not a) (not b)`
`or a b`	`not (nor a b)`
`sub a b`	`not (add (not a) b)`
`xor a b`	`and (or a b) (or (not a) (not b))`
`flags (sub a b)`	`add (and ~0x815 (flags (and (sub a b) (sub a b)))` `(and 0x815 (flags (add (not a) b)))`

From bytecode back to CISC

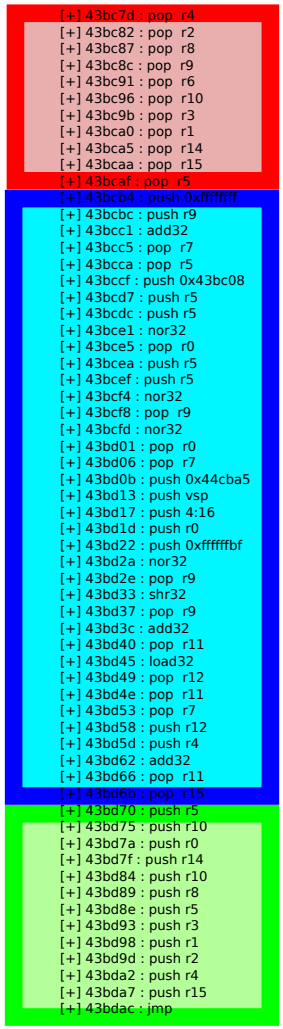
Source

```
loop:
        dec    eax
        test   eax, eax
        jnz    loop
```

Bytecode

```
[+] 43bc7d : pop  r4
[+] 43bc82 : pop  r2
[+] 43bc87 : pop  r8
[+] 43bc8c : pop  r9
[+] 43bc91 : pop  r6
[+] 43bc96 : pop  r10
[+] 43bc9b : pop  r3
[+] 43bca0 : pop  r1
[+] 43bca5 : pop  r14
[+] 43bcaa : pop  r15
[+] 43bcaf : pop  r5
[+] 43bcb4 : push 0xffffffff
[+] 43bcbc : push r9
[+] 43bcc1 : add32
[+] 43bcc5 : pop  r7
[+] 43bcca : pop  r5
[+] 43bccf : push 0x43bc08
[+] 43bcd7 : push r5
[+] 43bcdc : push r5
[+] 43bce1 : nor32
[+] 43bce5 : pop  r0
[+] 43bcea : push r5
[+] 43bcef : push r5
[+] 43bcf4 : nor32
[+] 43bcf8 : pop  r9
[+] 43bcfd : nor32
[+] 43bd01 : pop  r0
[+] 43bd06 : pop  r7
[+] 43bd0b : push 0x44cba5
[+] 43bd13 : push vsp
[+] 43bd17 : push 4:16
[+] 43bd1d : push r0
[+] 43bd22 : push 0xffffffbf
[+] 43bd2a : nor32
[+] 43bd2e : pop  r9
[+] 43bd33 : shr32
[+] 43bd37 : pop  r9
[+] 43bd3c : add32
[+] 43bd40 : pop  r11
[+] 43bd45 : load32
[+] 43bd49 : pop  r12
[+] 43bd4e : pop  r11
[+] 43bd53 : pop  r7
[+] 43bd58 : push r12
[+] 43bd5d : push r4
[+] 43bd62 : add32
[+] 43bd66 : pop  r11
[+] 43bd6b : pop  r15
[+] 43bd70 : push r5
[+] 43bd75 : push r10
[+] 43bd7a : push r0
[+] 43bd7f : push r14
[+] 43bd84 : push r10
[+] 43bd89 : push r8
[+] 43bd8e : push r5
[+] 43bd93 : push r3
[+] 43bd98 : push r1
[+] 43bd9d : push r2
[+] 43bda2 : push r4
[+] 43bda7 : push r15
[+] 43bdac : jmp
```

Bytecode


[+] 43bc7d : pop r4
[+] 43bc82 : pop r2
[+] 43bc87 : pop r8
[+] 43bc8c : pop r9
[+] 43bc91 : pop r6
[+] 43bc96 : pop r10
[+] 43bc9b : pop r3
[+] 43bca0 : pop r1
[+] 43bca5 : pop r14
[+] 43bcaa : pop r15
[+] 43bcaf : pop r5
[+] 43bcb4 : push 0xffffffff
[+] 43bcbc : push r9
[+] 43bcc1 : add32
[+] 43bcc5 : pop r7
[+] 43bcca : pop r5
[+] 43bccf : push 0x43bc08
[+] 43bcd7 : push r5
[+] 43bcdc : push r5
[+] 43bce1 : nor32
[+] 43bce5 : pop r0
[+] 43bcea : push r5
[+] 43bcef : push r5
[+] 43bcf4 : nor32
[+] 43bcf8 : pop r9
[+] 43bcfd : nor32
[+] 43bd01 : pop r0
[+] 43bd06 : pop r7
[+] 43bd0b : push 0x44cba5
[+] 43bd13 : push vsp
[+] 43bd17 : push 4:16
[+] 43bd1d : push r0
[+] 43bd22 : push 0xffffffbf
[+] 43bd2a : nor32
[+] 43bd2e : pop r9
[+] 43bd33 : shr32
[+] 43bd37 : pop r9
[+] 43bd3c : add32
[+] 43bd40 : pop r11
[+] 43bd45 : load32
[+] 43bd49 : pop r12
[+] 43bd4e : pop r11
[+] 43bd53 : pop r7
[+] 43bd58 : push r12
[+] 43bd5d : push r4
[+] 43bd62 : add32
[+] 43bd66 : pop r11
[+] 43bd6b : pop r15
[+] 43bd70 : push r5
[+] 43bd75 : push r10
[+] 43bd7a : push r0
[+] 43bd7f : push r14
[+] 43bd84 : push r10
[+] 43bd89 : push r8
[+] 43bd8e : push r5
[+] 43bd93 : push r3
[+] 43bd98 : push r1
[+] 43bd9d : push r2
[+] 43bda2 : push r4
[+] 43bda7 : push r15
[+] 43bdac : jmp

Prologue

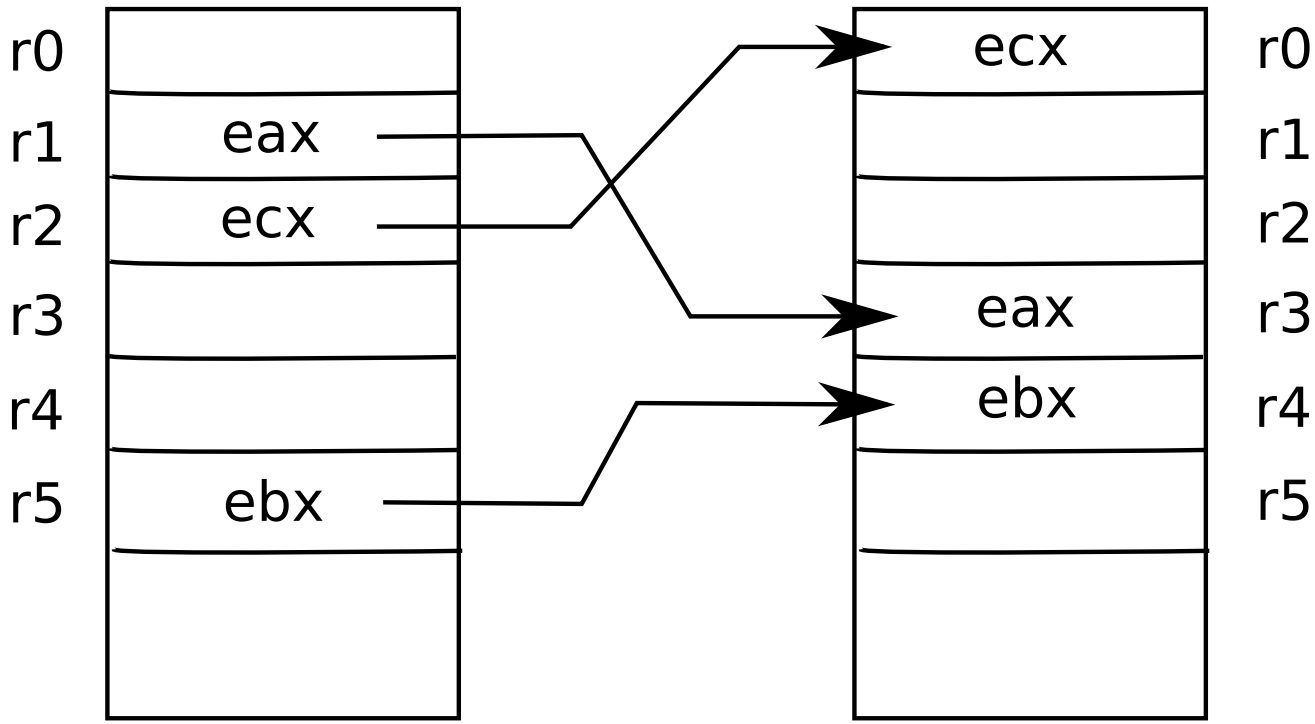
```
[+] 43bc7d : pop  r4
[+] 43bc82 : pop  r2
[+] 43bc87 : pop  r8
[+] 43bc8c : pop  r9
[+] 43bc91 : pop  r6
[+] 43bc96 : pop  r10
[+] 43bc9b : pop  r3
[+] 43bca0 : pop  r1
[+] 43bca5 : pop  r14
[+] 43bcaa : pop  r15
[+] 43bcaf : pop  r5
```

Epilogue

```
[+] 43bd70 : push r5
[+] 43bd75 : push r10
[+] 43bd7a : push r0
[+] 43bd7f : push r14
[+] 43bd84 : push r10
[+] 43bd89 : push r8
[+] 43bd8e : push r5
[+] 43bd93 : push r3
[+] 43bd98 : push r1
[+] 43bd9d : push r2
[+] 43bda2 : push r4

[+] 43bda7 : push r15
[+] 43bdac : jmp
```

Virtual registers permutation



Body

```
[+] 43bcb4 : push 0xffffffff
[+] 43bcbc : push r9
[+] 43bcc1 : add32
[+] 43bcc5 : pop  r7
[+] 43bcca : pop  r5
[+] 43bccf : ...
[+] 43bcd7 : ...
[+] 43bcdc : ...
[+] 43bce1 : ...
```

How do we simplify the body ?

Mutable registers

```
pop  r4
pop  r2
...
pop  r5

push 0xffffffff
push r9
add32
pop  r7
pop  r5
...
```

SSA form

```
pop  r4.1
pop  r2.1
...
pop  r5.1

push 0xffffffff
push r9.1
add32
pop  r7.1
pop  r5.2
...
```

Explicit operands IR

add32 ⟶

```
pop Y
pop X
R, F = add X, Y
push R
push F
```

Mostly stackless

```
r5.2, r7.1 = r9.1 + 0xffffffff
push 0x43bc08
r0.1 = flags (nor r5.2 r5.2)
r9.2 = flags (nor r5.2 r5.2)
r7.2, r0.2 = nor (nor r5.2 r5.2) (nor r5.2 r5.2)
push 0x44cba5
r9.3 = flags ...
r9.4 = flags ...
r11.1 = flags ...
r12.1 = load32 (vsp + (shr (nor 0xffffffbf r0.2) 4))
pop  r11.2
pop  r7.3
r11.1 = flags ...
r15.2 = r4.1 + r12.1
...
jmp  r15.2
```

Mostly stackless

```
r5.2, r7.1 = r9.1 + 0xffffffff
push 0x43bc08
r0.1 = flags (nor r5.2 r5.2)
r9.2 = flags (nor r5.2 r5.2)
r7.2, r0.2 = nor (nor r5.2 r5.2) (nor r5.2 r5.2)
push 0x44cba5
r9.3 = flags ...
r9.4 = flags ...
r11.1 = flags ...
r12.1 = load32 (vsp + (shr (nor 0xffffffbf r0.2) 4))
pop  r11.2
pop  r7.3
r11.1 = flags ...
r15.2 = r4.1 + r12.1
...
jmp  r15.2
```

Liveness

```
r5.2, r7.1 = r9.1 + 0xffffffff
push 0x43bc08
r0.1 = flags (nor r5.2 r5.2)
r9.2 = flags (nor r5.2 r5.2)
r7.2, r0.2 = nor (nor r5.2 r5.2) (nor r5.2 r5.2)
push 0x44cba5
r9.3 = flags ...
r9.4 = flags ...
r11.1 = flags ...
r12.1 = load32 (vsp + (shr (nor 0xffffffbf r0.2) 4))
pop  r11.2
pop  r7.3
r11.1 = flags ...
r15.2 = r4.1 + r12.1
...
jmp  r15.2
```

Dead Store Elimination

```
r5.2, _ = r9.1 + 0xffffffff
push 0x43bc08
r0.2 = flags (nor (nor r5.2 r5.2) (nor r5.2 r5.2))
push 0x44cba5
r12.1 = load32 (vsp + (shr (nor 0xffffffbf r0.2) 4))
pop  _
pop  _
r15.2 = r4.1 + r12.1
...
jmp  r15.2
```

How to restore original operators ?

```
r5.2, _ = r9.1 + 0xffffffff
push 0x43bc08
r0.2 = flags (nor (nor r5.2 r5.2) (nor r5.2 r5.2))
push 0x44cba5
r12.1 = load32 (vsp + (shr (nor 0xffffffbf r0.2) 4))
pop  _
pop  _
r15.2 = r4.1 + r12.1
...
jmp  r15.2
```

Algebraic transform

```
r5.2, _ = r9.1 + 0xffffffff
push 0x43bc08
r0.2 = flags (and r5.2 r5.2)
push 0x44cba5
r12.1 = load32 (vsp + (shr (and 0x40 (not r0.2)) 4))
pop  _
pop  _
r15.2 = r4.1 + r12.1
...
jmp  r15.2
```

Recognize conditional load from stack

```
r5.2,    = r9.1 + 0xffffffff
push 0x43bc08
r0.2 = flags (and r5.2 r5.2)
push 0x44cba5
r12.1 = load32 (vsp + (shr (and 0x40 (not r0.2)) 4))
pop  _
pop  _
r15.2 = r4.1 + r12.1
...
jmp  r15.2
```

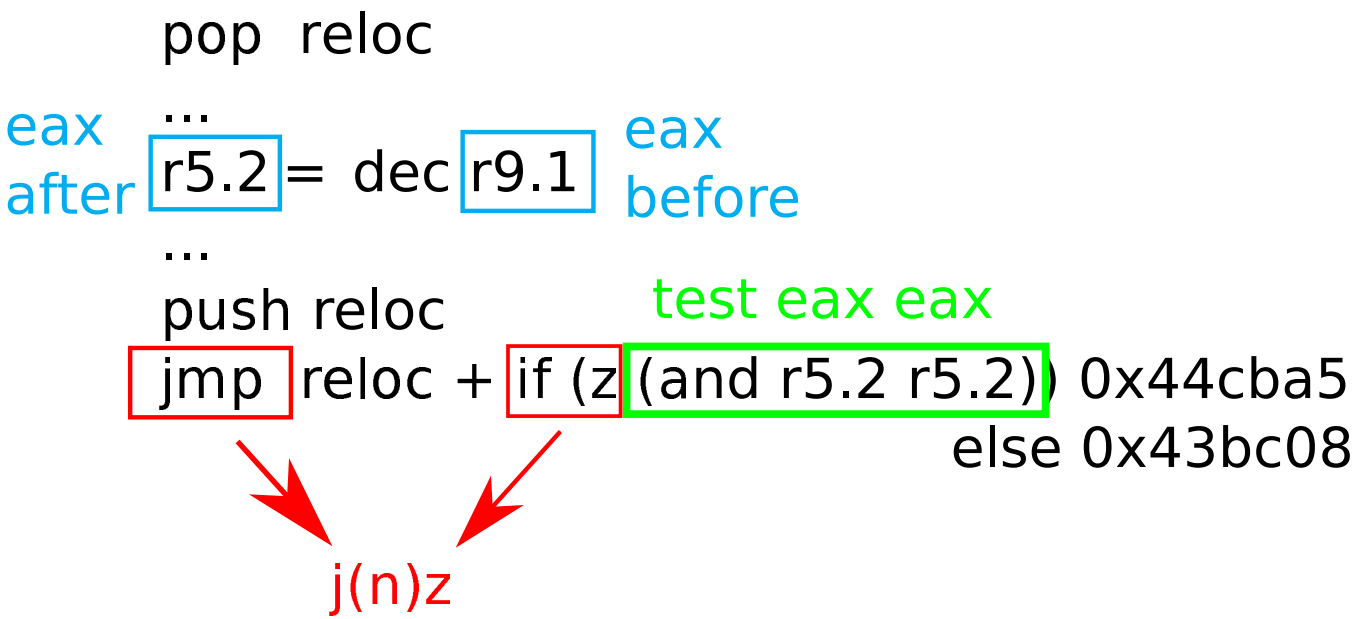
Introduce "if"

```
r5.2, _ = r9.1 + 0xffffffff
push 0x43bc08
r0.2 = flags (and r5.2 r5.2)
push 0x44cba5
r12.1 = if (bit #6 r0.2) 0x44cba5 else 0x43bc08
pop  _
pop  _
r15.2 = r4.1 + r12.1
...
jmp  r15.2
```

Branch destination

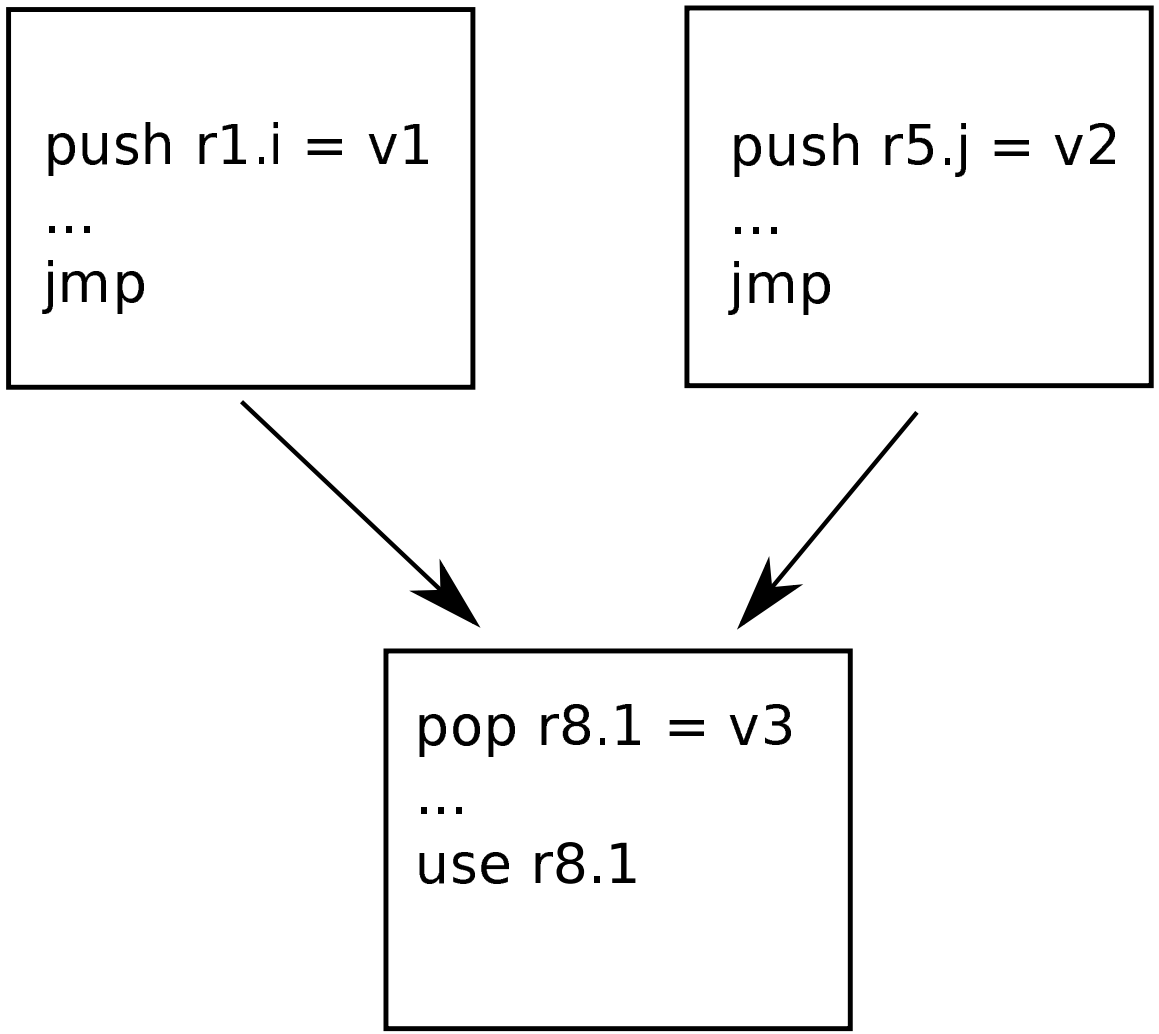
```
pop r4.1
...
r5.2, _ = r9.1 + 0xffffffff
...
push r4.1
jmp r4.1 + if (bit #6 flags (and r5.2 r5.2)) 0x44cba5
                                    else 0x43bc08
```

CISC

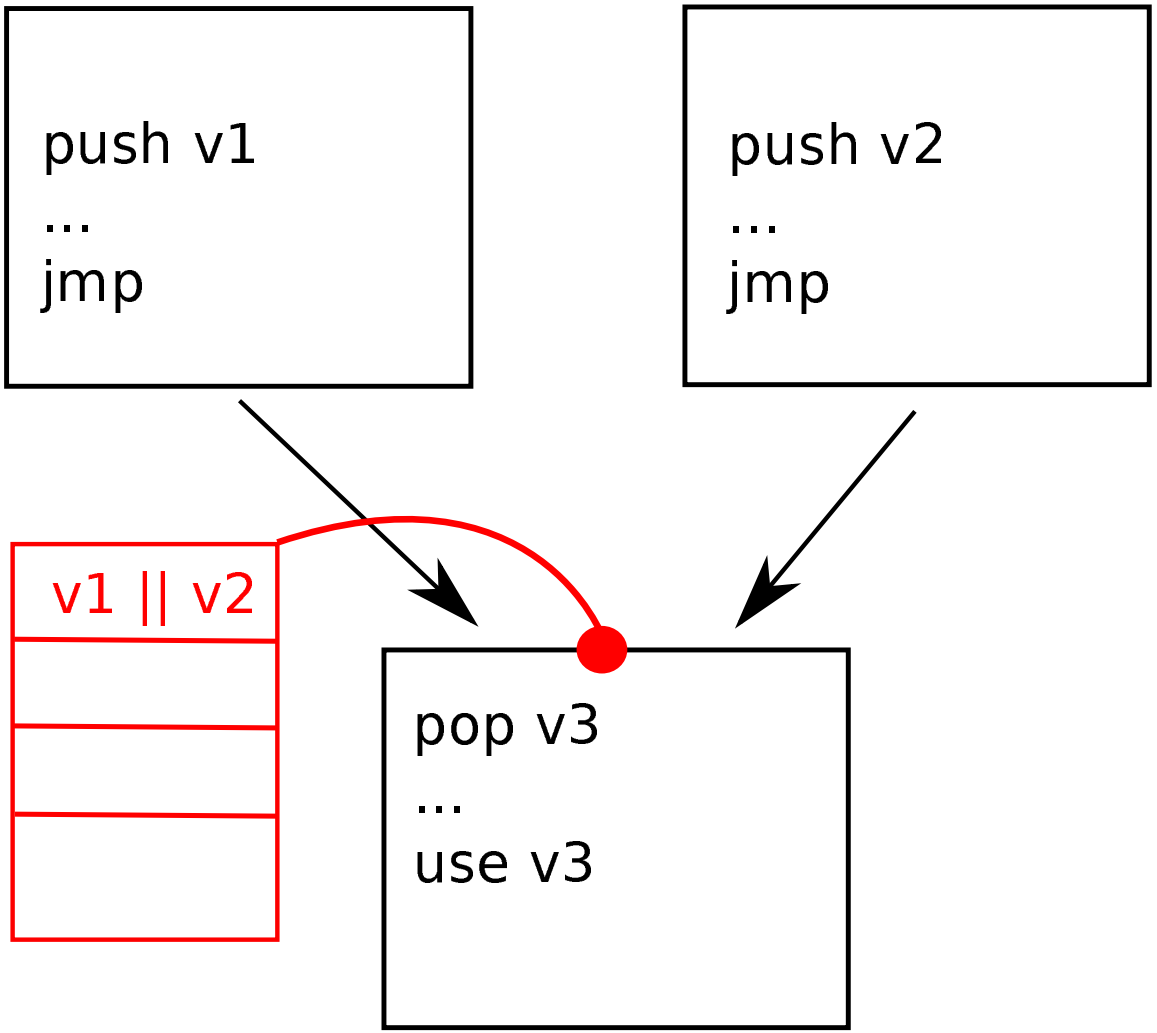

pop reloc
...
eax after
r5.2 = dec r9.1
eax before
...
push reloc
test eax eax
jmp reloc + if (z (and r5.2 r5.2)) 0x44cba5
else 0x43bc08
j(n)z

How do we undo the virtual registers permutation ?

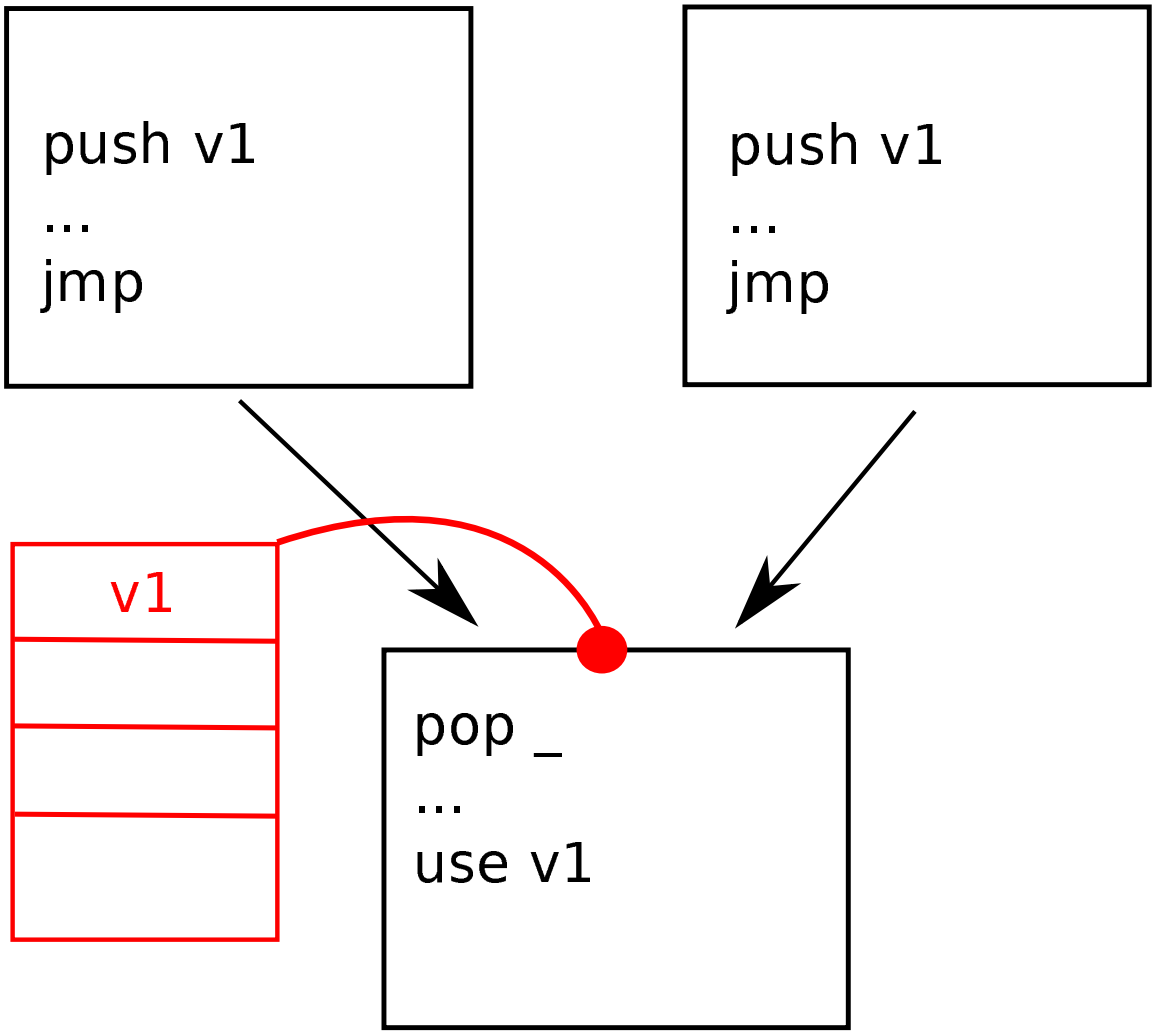
Global SSA variables



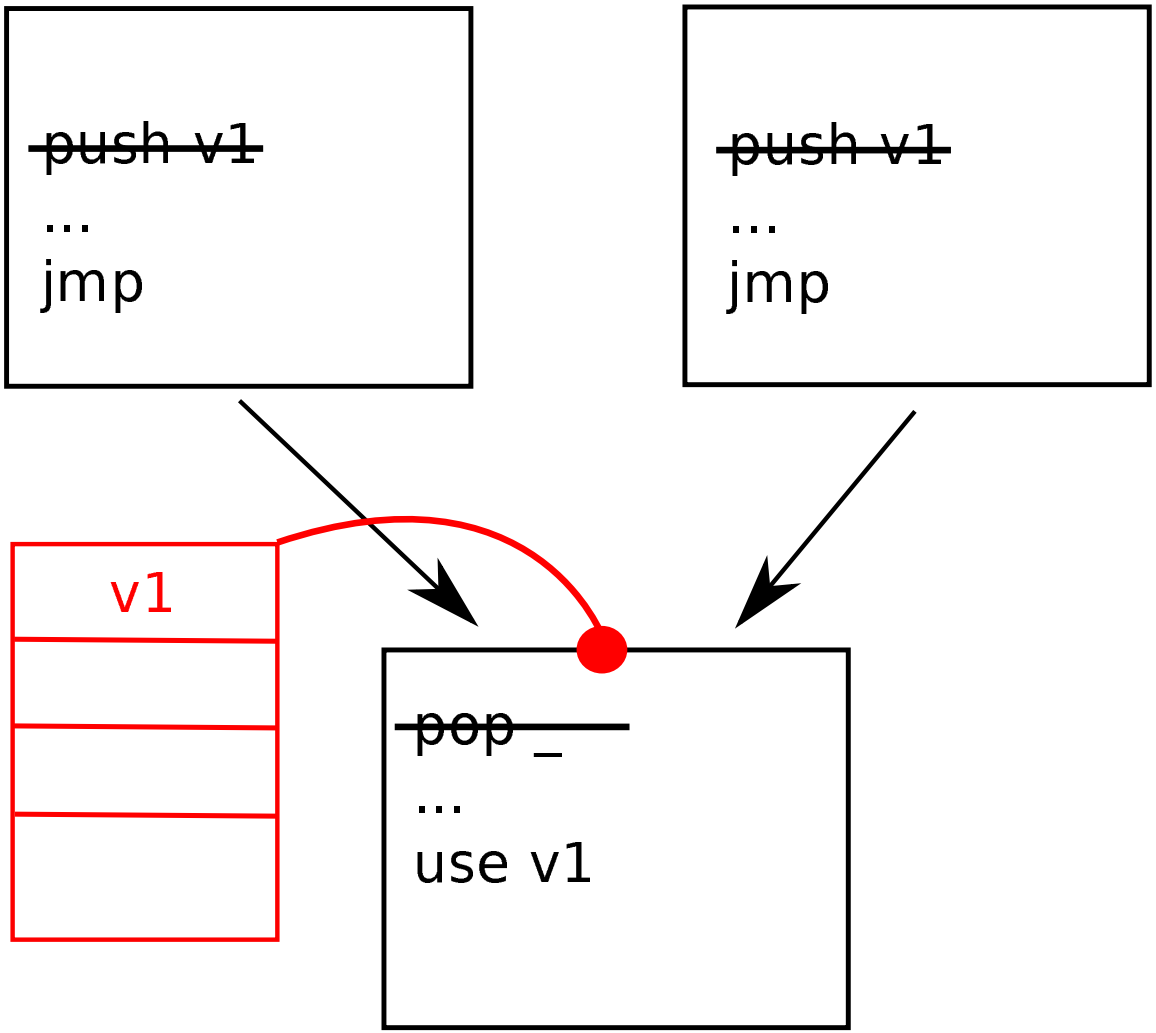
Data-flow analysis on stack state



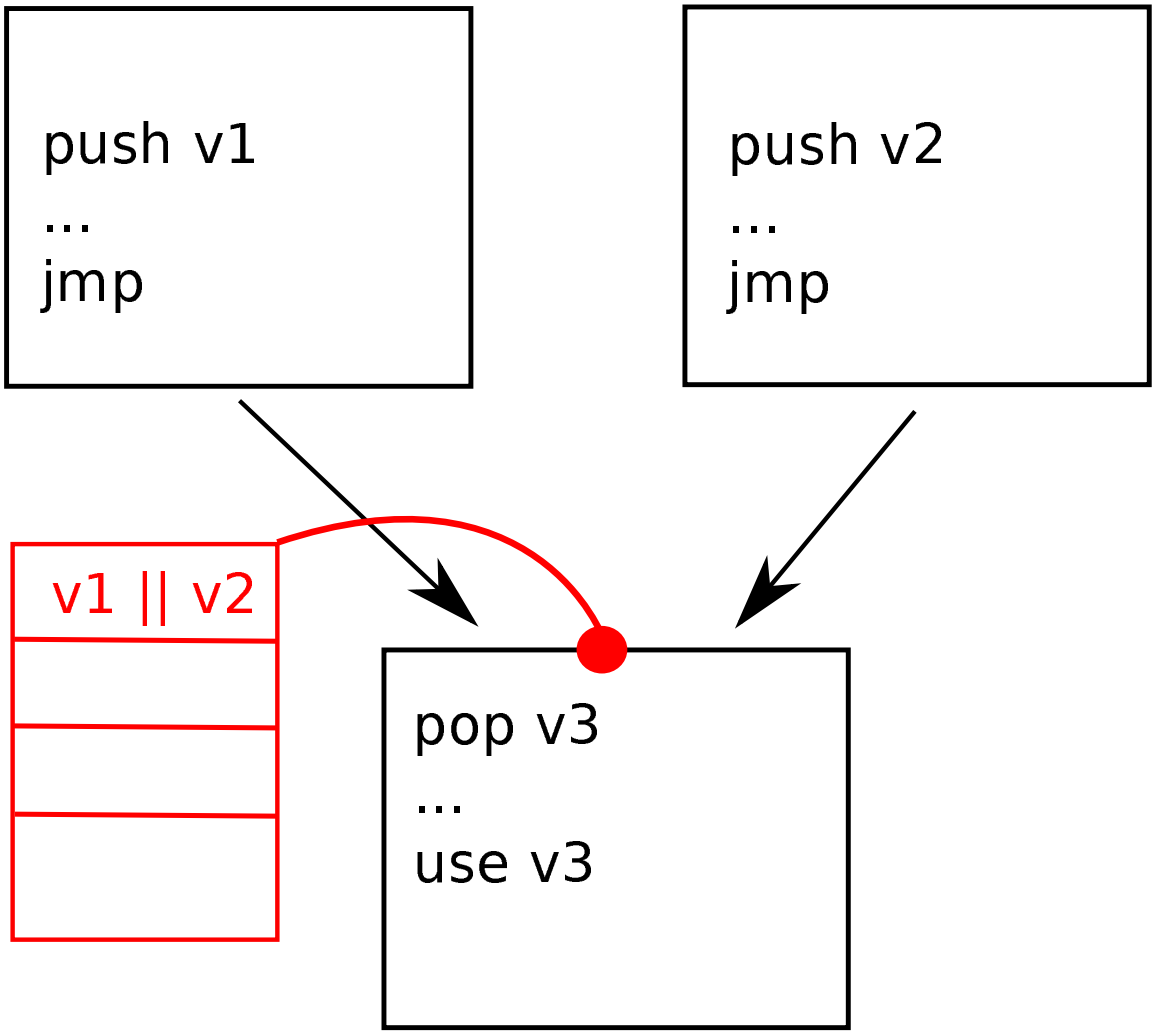
Simple case : v1 == v2



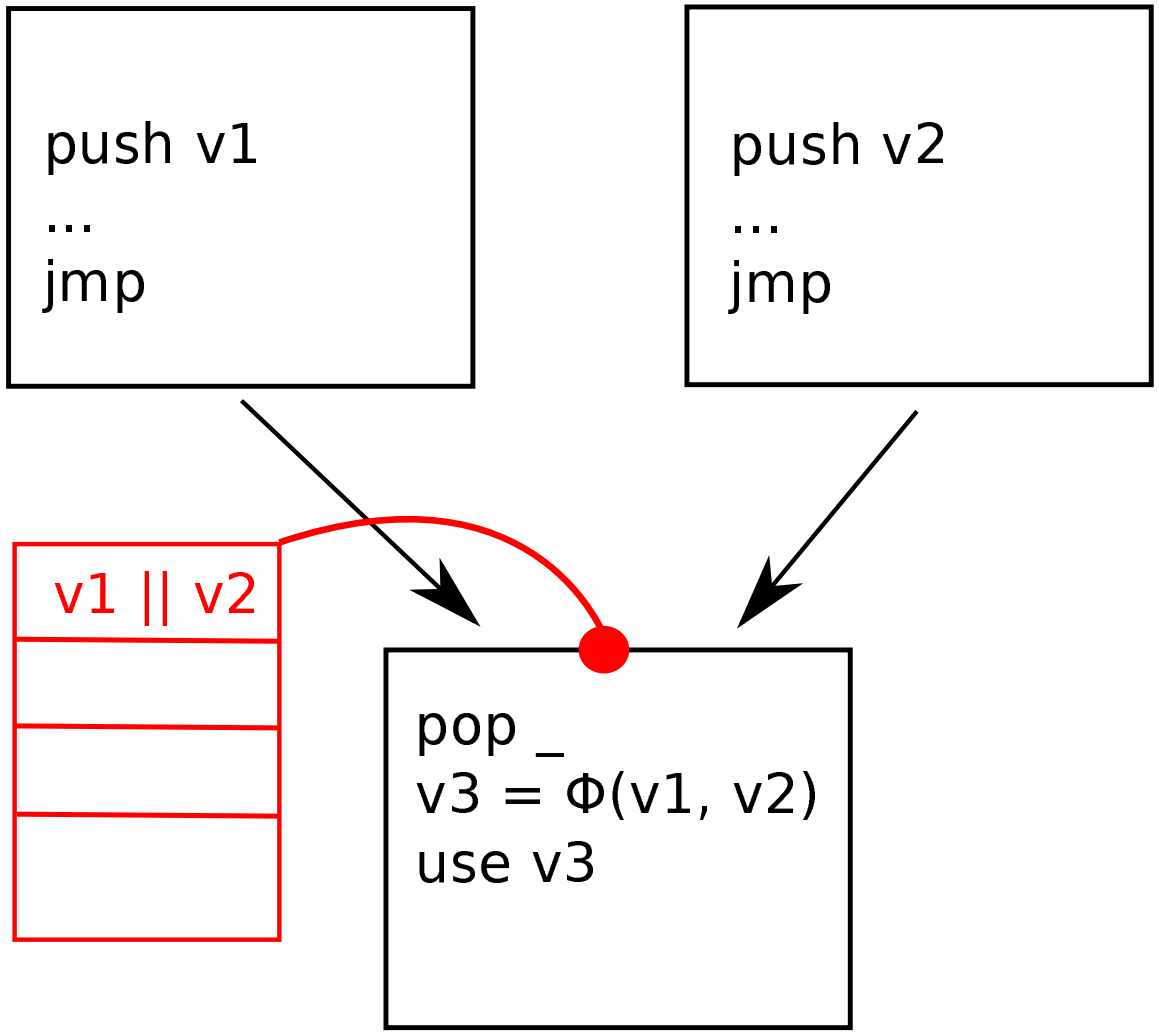
Cancel pair



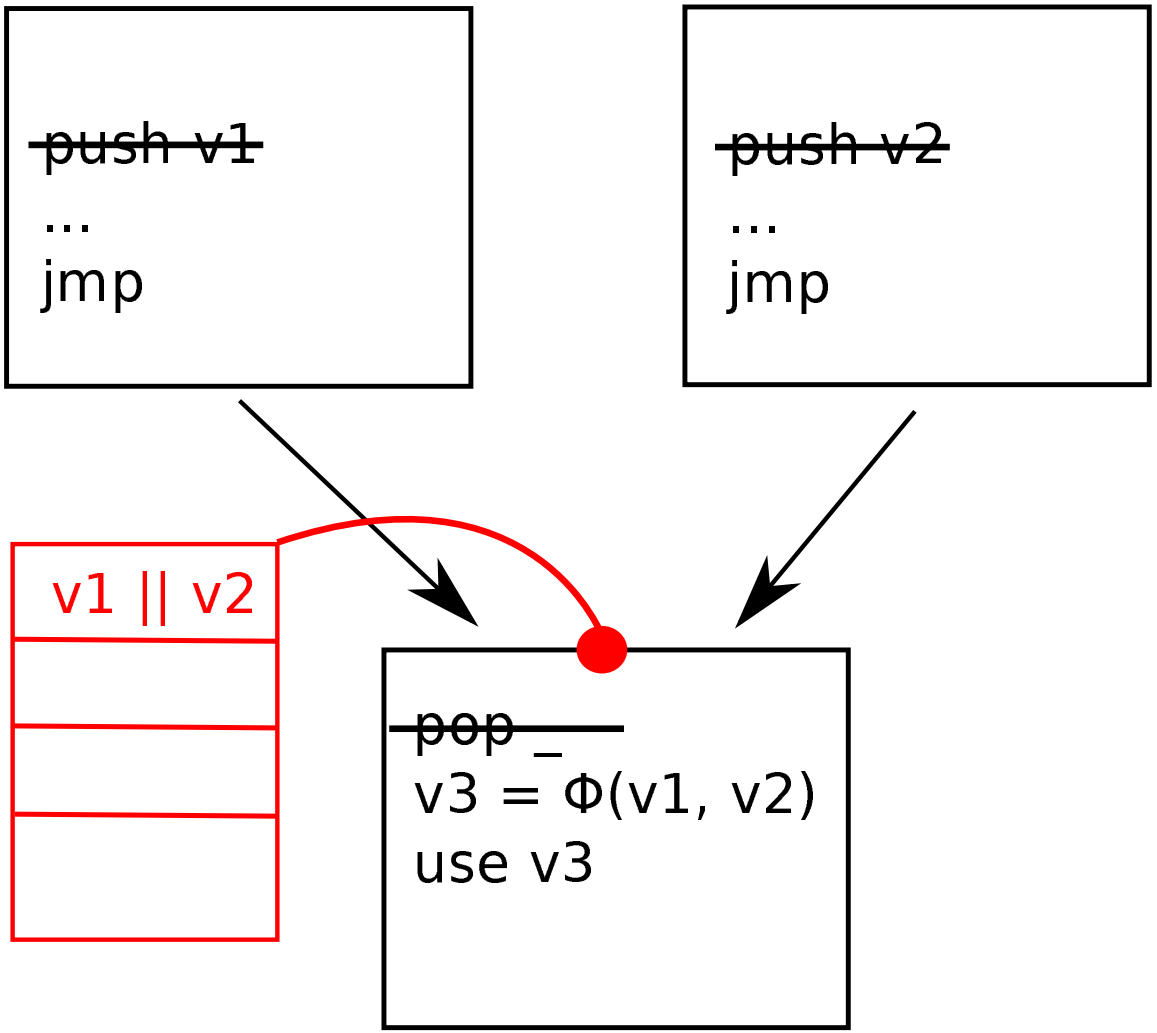
General case



Introduce phi function



Cancel pair



Getting rid of phi functions

- Global liveness analysis
- Global dead store elimination
- Most phi functions disappear
- Why ?

Entry stacks from two predecessors

reloc
ebp
temp
esi
eax
...

reloc
ebp
constant
esi
eax
...

Recovering native registers ?

- Yes, partially
- Use VM entries and exits
- Register allocation still necessary

Recovering call graph ?

- Yes, with heuristics
- Recognize call/ret patterns
- Recognize functions

That's all folks !

"la comparaison entre l'assembleur et le bytecode est parlante même si on ne comprend pas l'assembleur"
–SSTIC relecteur 4