Registers

used	not used
rax	rbx
rsp	rcx
r10	rdx
r11	r8
rbp	r9
rdi	r12
	r13
	r14
	r15
	rsi

2+3

register access	~1 cycle
L1 cache	~4 cycles
L2 cache	~10 cycles
RAM	~100 cycles

```
mov rax, 4          <- 1 cycle
mov [rbp-8], rax    <- ~100 cycles
mov rax, 6          <- 1 cycle
mov [rbp-16], rax   <- ~4 cycles
mov rax, [rbp-8]    <- ~4 cycles
add rax, [rbp-16]   <- ~4 cycles
```

```
let a = 5 in
let b = after (2) in
let c = a + b in
let d = 6 in
let e = b + 2 in
let f = d + e in
let g = f + 5 in
c + g
```

(-8, {})

([[rbp-8], [rbp-16], [rbp-24], ...], {})

(-8, [r12, r13, r14, r15], {})

```
mov rax, 10
mov [rbp-8], rax          ; store a
mov rax, 4
add rax, 2
mov [rbp-16], rax         ; store b
mov rax, [rbp-8]
add rax, [rbp-16]
mov [rbp-24], rax         ; store c
mov rax, 12
mov [rbp-32], rax         ; store d
mov rax, [rbp-16]
add rax, 4
mov [rbp-40], rax         ; store e
mov rax, [rbp-32]
add rax, [rbp-40]
mov [rbp-48], rax         ; store f
mov rax, [rbp-48]
add rax, 10
mov [rbp-56], rax         ; store g
mov [rbp-24], rax
add rax, [rbp-56]
```

r12 r13 r14 r15 [rbp-8]

Use registers as needed: linear scanning

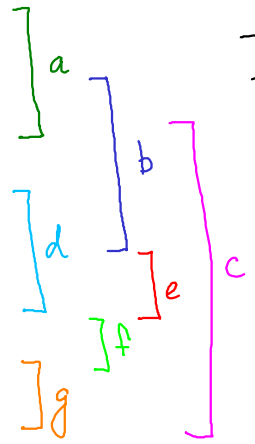
Swapping registers out to memory = spilling

Good for JIT

produce code ~30% slower than best techniques

very fast to compile

```
let a = 5 in
let b = after (2) in
let c = a + b in
let d = 6 in
let e = b + 2 in
let f = d + e in
let g = f + 5 in
c + g
```

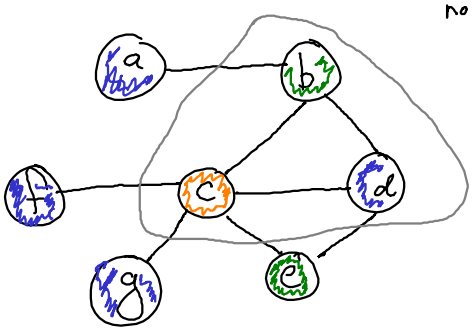


Interference Graph (coexistence)

Vertices: variables

Edges: coexistence

Coloring: mapping from $V \mapsto$ Color s.t. no adjacent vertices share a color



R12

R13

R15

optimal Graph coloring : NP-complete

Approximately: polynomial time: within constant factor

expected polynomial time: optimal

```
mov rax, 10
mov LOC1, rax          ; store a
mov rax, 4
add rax, 2
mov LOC2, rax          ; store b
mov rax, LOC1
add rax, LOC2
mov LOC3, rax          ; store c
mov rax, 12
mov LOC4, rax          ; store d
mov rax, LOC2
add rax, 4
mov LOC5, rax          ; store e
mov rax, LOC4
add rax, LOC5
mov LOC6, rax          ; store f
mov rax, LOC6
add rax, 10
mov LOC7, rax          ; store g
mov LOC3, rax
add rax, LOC7
```