Last week

- Grading labs & test 2
- Do have lab; if you are finished, let me know
- Office hours
- "Final" goes out on 13th
- Extended topics

Register Allocation

Registers are faster than cache

Registers are ^ faster than memory

~2-3 orders of magnitude

```
mov [rsp+16], rax
mov rcx, rax
```

Algorithms

- ☆ Greedy
- ☆ Linear scaning
- ☆ Graph coloring

Greedy

algorithmic registers

rax, rsp, rbp,
r10, r11, rcx

storage registers

r8, r9, rdx,
r12, r13, r14, r15

local stack

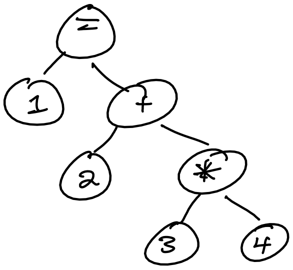
next free offset — name mapping

(-8 , {....})

[[rbp-8]; [rbp-16] ; ....]

[r8, r9, rdx, r12, r13, r14, r15, [rbp-8], [rbp-16], ...]

Example : 1 = 2 + 3*4



Suppose : Storage regs : only r8, r9, rdx

[r8, r9, rdx, [rbp-8], ...]

```
mov rax, 2
mov r8, rax
mov rax, 4
mov r9, rax
mov rax, 6
mov rdx, rax
mov rax, 8
mov [rbp-8], rax
mov rax, rdx
imul rax, [rbp-8]
mov rdx, rax
   ⋮
```

Problem case

let x = 1 in
let y = 2 in
let z = 3 in
let ~~answer =~~

in
x + y + z + answer

?

Option :

- Pre-analyze code to determine what should go where
- Recognize that we need more space while generating

Linear scanning

```
mov rax, 2
mov r8, rax
mov rax, 4
mov r9, rax
mov rax, 6
mov rdx, rax
mov rax, 8
mov [rbp - 8], r8
mov r8, rax
   ⋮
mov r8, [rbp-8]
   ⋮
```

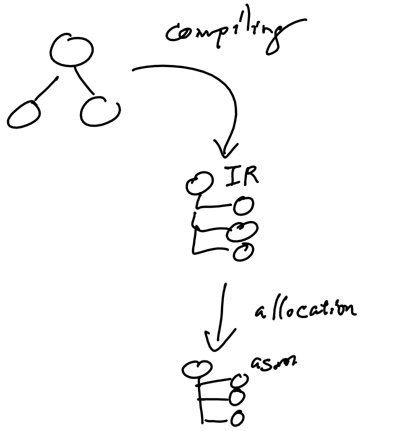
tricky

1 = 2+3*4

Most compilers do not directly generate assembly.

Most compilers use an intermediate representation (IR).

```
mov  ans, 2
mov  loc(1), ans
mov  ans, 4
mov  loc(2), ans
mov  ans, 6
mov  loc(3), ans
mov  ans, 8
mov  loc(4), ans
mov  ans, loc(3)
imul ans, loc(4)
  ⋮
```

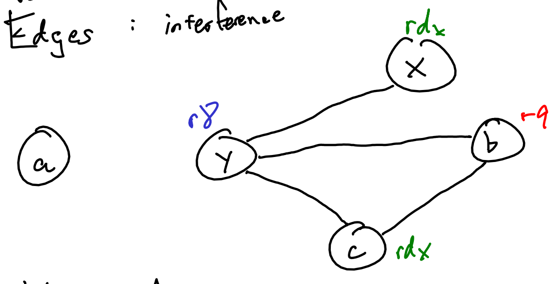


Graph Coloring

"interference"

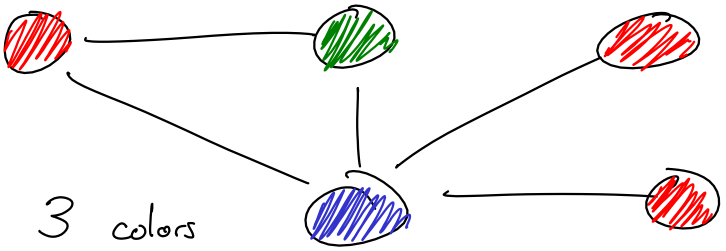
```
let x = 1 in     ———— {x}
let y = 2+x in   ———— {x,y}
let a = x in     ———— {y}
let b = y*2 in   ———— {y,b}
let c = 3 in     ———— {y,b,c}
y+b+c
```

Vertices: abstract storage locations
Edges : interference



Color graph

Color represents a concrete storage location



Goal: color vertices s.t. no two neighbors have same color, minimizing colors

Generally: NP-complete
colors ≥ 3

Good polynomial approximations

gen code ~120% faster than linear scanning