

# Lec 10 Calling Conventions

## # IR for calling convention

Goal: args are pure, args evaled left to right

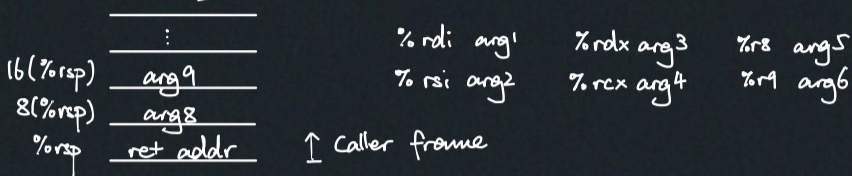
higher level  $d \leftarrow f(s_1, \dots, s_n)$

lower level CALL f

## # x86-64 calling convention

first 6 args\* go on certain regs, others go stack  
 \* except floating point args go on %xmm0 - %xmm7

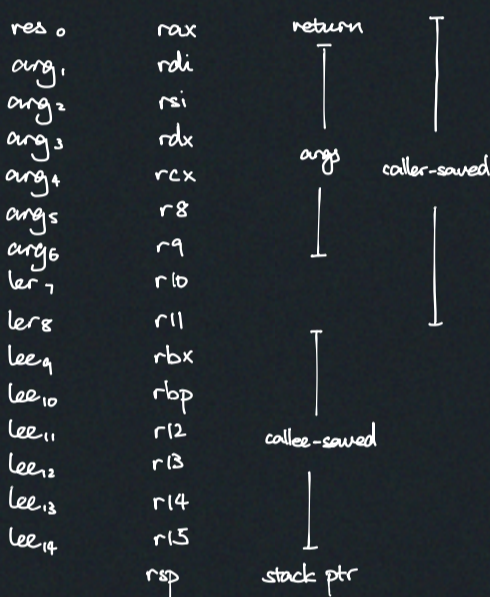
stack when calling func



Good practices

- 8 byte per arg on stack
- Statically precompute frame size
- %rbp aligned 0 mod 16 before call, 8 mod 16 on entry

## Renaming



## # Reg alloc

Caller-saved & arg regs can get overwritten

$\frac{l: \text{call } f \quad \text{caller-save}(r)}{\text{def}(l, r)} \quad \frac{l: \text{ret } s \quad \text{callee-save}(r)}{\text{use}(l, r)}$

Suppose  $d \leftarrow f(s_1, s_2, s_3)$

↓  
 $\text{arg}_3 \leftarrow s_3$   
 $\text{arg}_2 \leftarrow s_2$   
 $\text{arg}_1 \leftarrow s_1$   
 call f  
 $d \leftarrow \text{res}_0$

← if  $\dagger$  used here then it interferes with all the caller-saved regs

At start of func, try not hold precoloured regs for too long

```
f:
  x ← arg_1
  y ← arg_2
  z ← arg_3
```

## # Saving stuff

Standard: save all lee\* onto stack & restore them at end  
 ↳ but then they live throughout the func

Spill techniques:

1. reg alloc on caller-saved first, spill to caller-saved if necessary, then spill to stack
2. move them to temp and move them back, let reg alloc decide
3. copy propagation...?

e.g. (2)

```
t1 ← lee_9
t2 ← lee_10
⋮
lee_10 ← t2
lee_9 ← t1
```