Quiz 4 Review Session

•Calling conventions
•Callee saved/caller saved registers
•Stack alignment

Compiler Toolchains

•Assemblers
•Linkers
•Loaders

Control Flow Graphs

•Basic Blocks
•Edge Types
•Dominators
•Static Single Assignment (SSA) Form
•Φ-nodes

Dataflow

•Dead Code Elimination
•Constant Folding
•Copy/Constant Propagation

Final Code Optimization

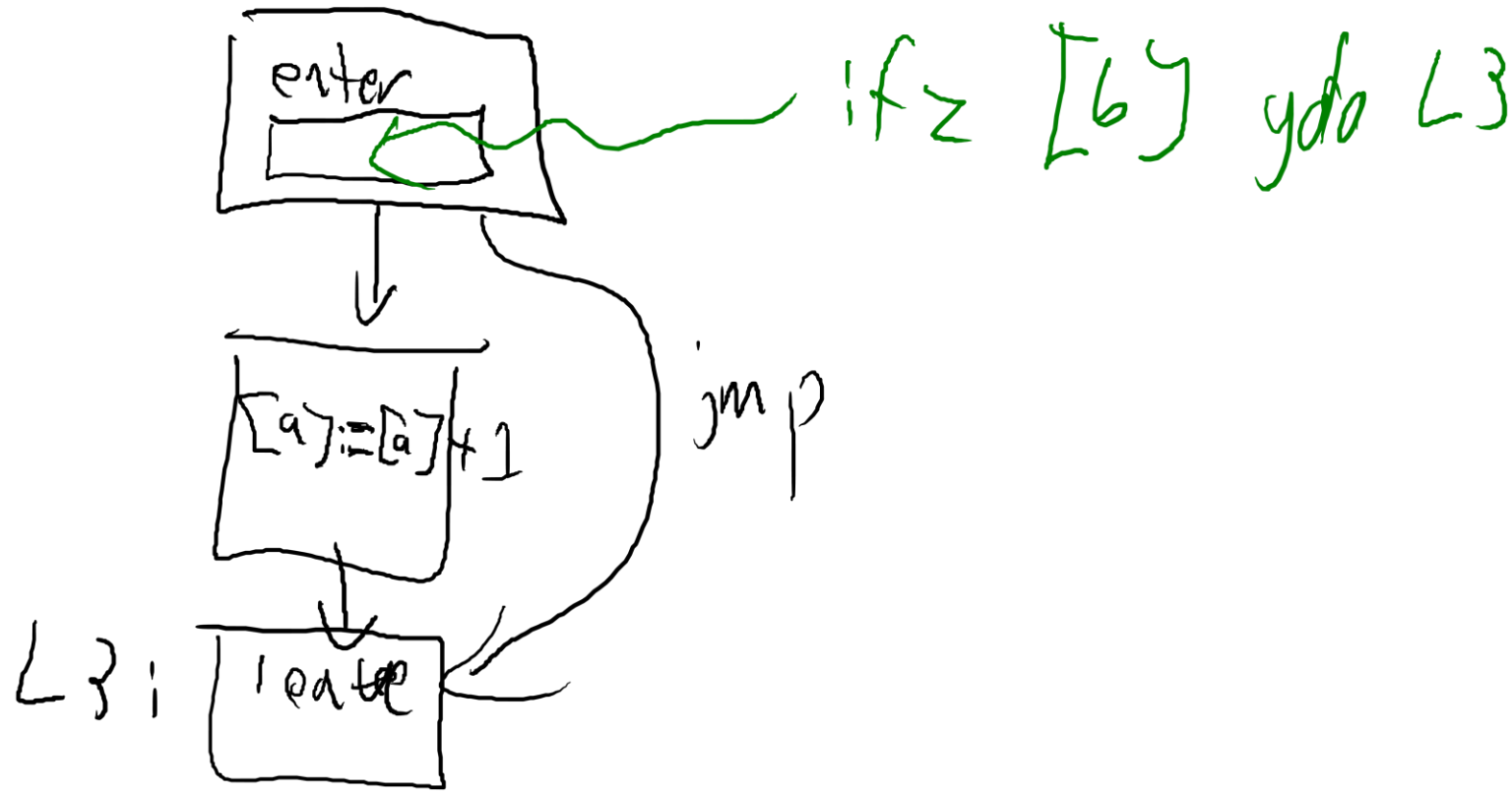
•Peephole optimization
•Interference graphs
•Register allocation
•Instruction selection
•Instruction strength reduction

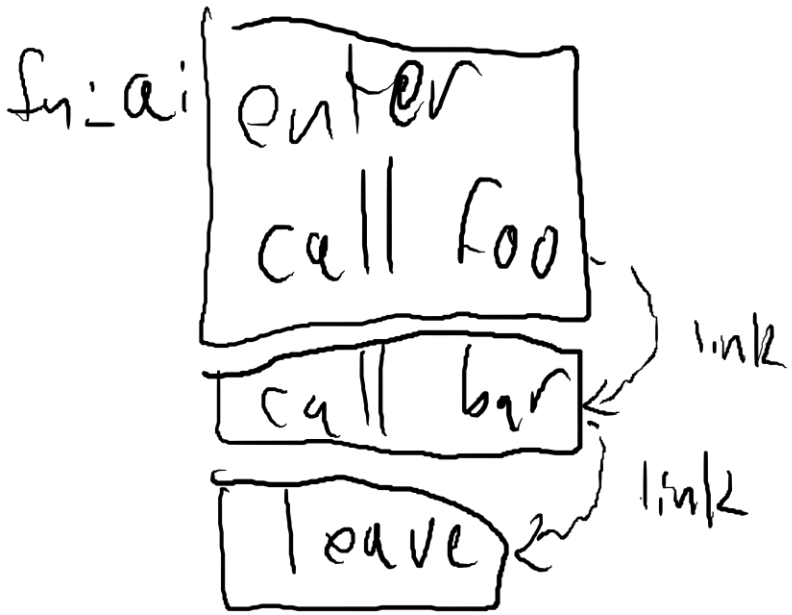
Heap Management

•Allocating Heap Memory
•Garbage Collection
  - Reference Counting
  - Mark and Sweep

CFG

Overlay on a linear code repr.






```
a : () -> void {
    foo();
    bar();
}
```

```
L1: nop
L2: nop   <- L
L3: nop
L4: goto L2
```

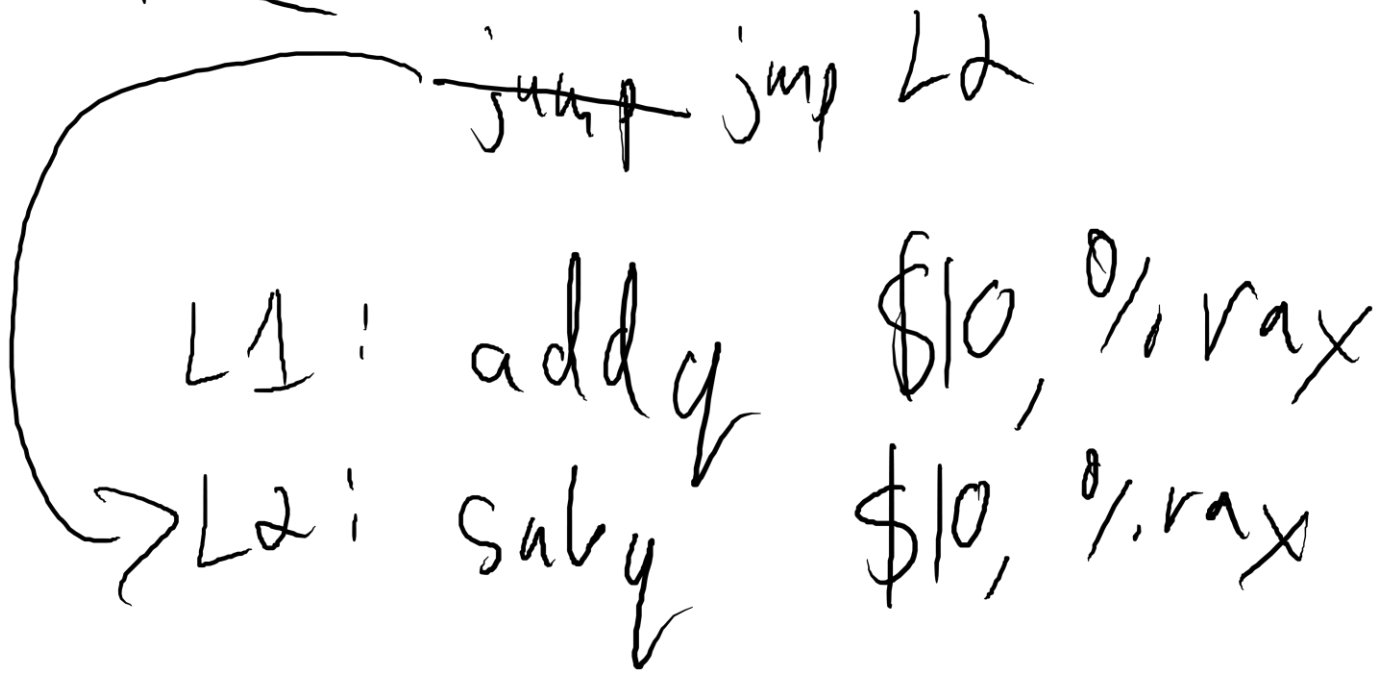
```
a: () -> void {
    foo()
}
```

jmp edge

Fallthrough edge

link edge

Peephole



Caller-Saved

```
movq $7, %rax
call bar
```

pushq %rax

popq %rax

→ do not expect %rax to be $7

Callee-Saved

```
movq $11, %r14
call foo
```

→ expect %r14 to have $11 in it

%rbp

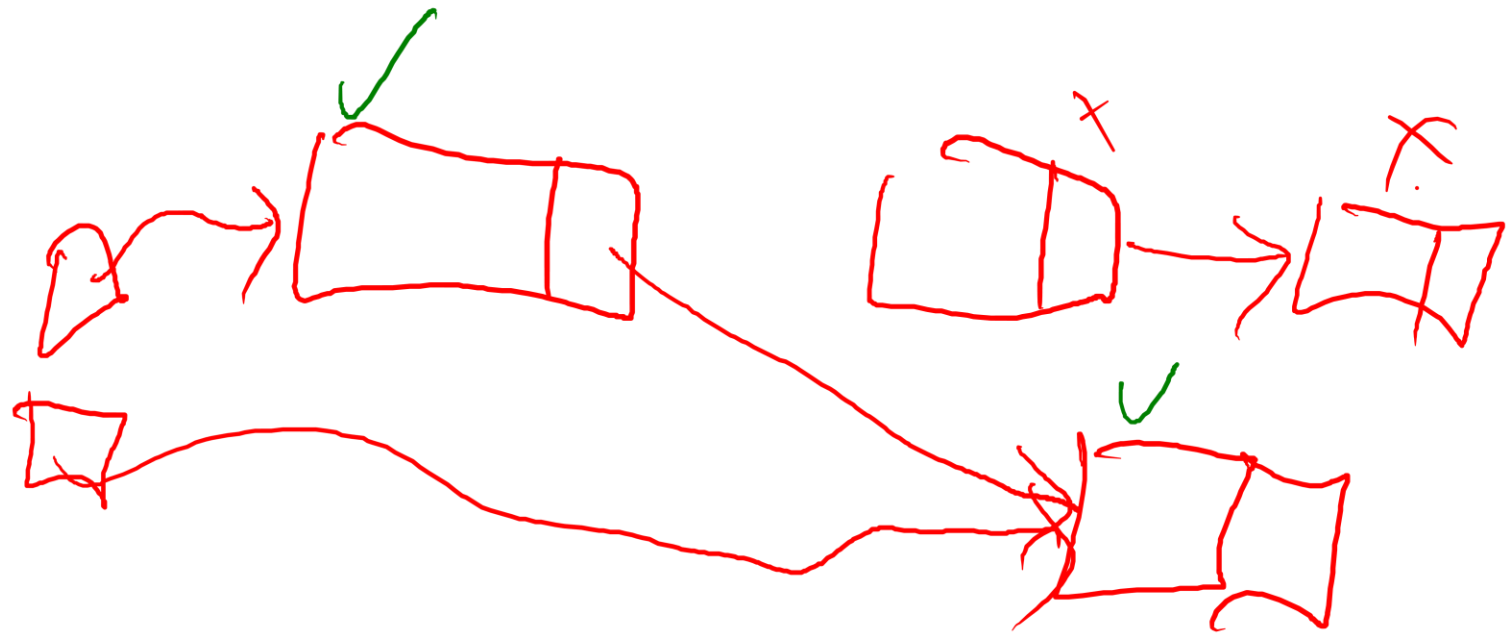
Garbage Collection

Automatic release of allocated memory at runtime

Naive reference Counting

Mark and Sweep

Mark and Sweep



```
fn () {
    <var> passwd = ...;
    for (i = passwd; i < strlen(passwd); ++) {
        passwd[i] = 0;
    }
}
```

Interference Graph

Nodes: variables

Edges: simultaneous live ners

a = 1
output a
b = 2
output b
output a

a = 1
b = 2
output a
output b

a = 1
b = 2
return