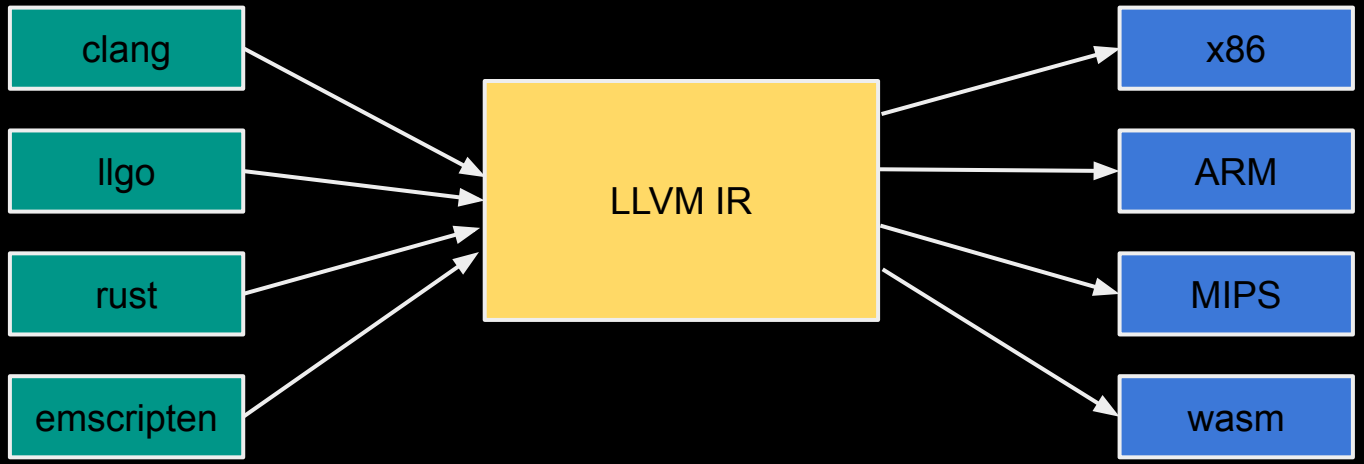

SPQR

LLVM

- Modular Compiler Infrastructure with reusable components
  - static compilers
  - JITs
  - optimization passes



LLVM IR

```c
int foo(int a, int b) {
    return a+b;
}
```

foo.c

```
; ModuleID = 'foo.bc'
source_filename = "foo.c"
target triple = "x86_64-pc-linux-gnu"

; Function Attrs: norecurse readnone willreturn
define i32 @foo(i32 %0, i32 %1) {
  %3 = add nsw i32 %1, %0
  ret i32 %3
}
```

foo.ll

LLVM passes

- Analysis or transformation on the LLVM IR bitcode
  - **ModulePass**: general interprocedural pass
  - **FunctionPass**: process a function at a time
  - **LoopPass**: process a natural loop at a time
  - **BasicBlockPass**: process a basic block at a time

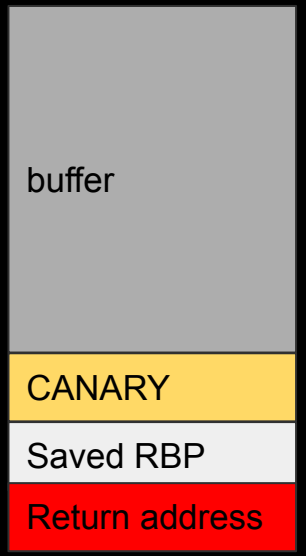
Each pass can only modify what is processing.

RAP by grsecurity

- Mitigation against control flow hijacking
  - Check RET address
  - Check indirect branches

RAP by grsecurity

- Mitigation against control flow hijacking
  - Check RET address
  - ~~Check indirect branches~~
- The stack canary already protects the return address
  - but vulnerable to information leakage



RAP by grsecurity

- Mitigation against control flow hijacking
  - Check RET address
  - ~~Check indirect branches~~