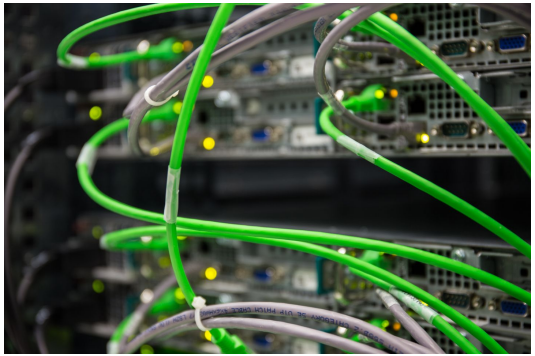
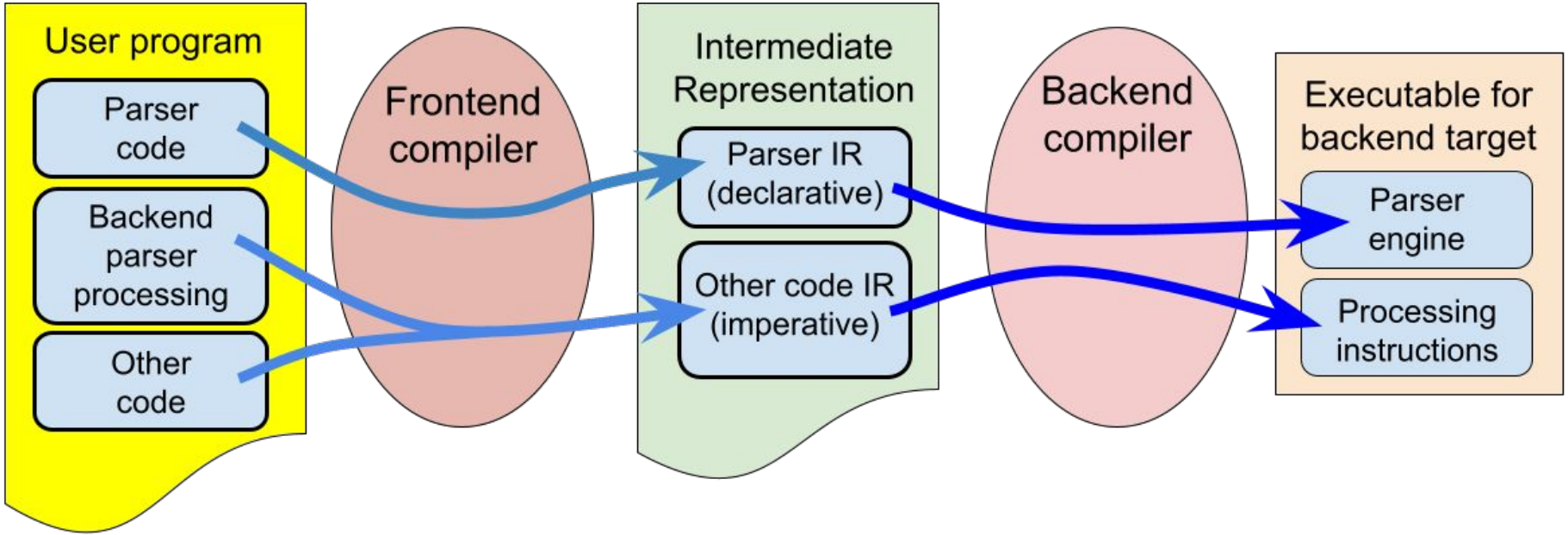
Compilers and Linkers: Network offload

Network offload model



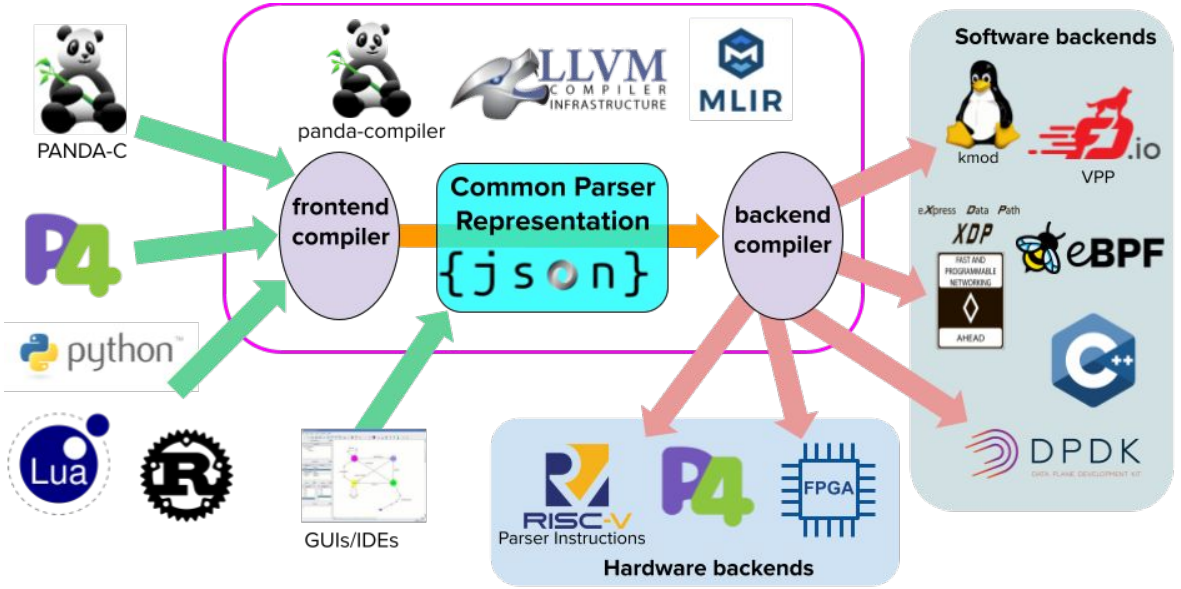
- Model for network offloading
- Write parser in preferred language (P4, C, JSON, etc)
- Compile DSL to Parser Intermediate Language
- Compile Parser Intermedia Language to specific hardware or software (XDP/eBPF)
- Load parser with devlink or load program

Network offload model



Common Parser Representation

- Interoperability
- Promote a standard for IR
- Compiler technology improved drastically
- JSON
- MLIR



Common Parser Representation

- JSON
- Declarative
- Network Parser concepts
- TLVs
- JSON -> MLIR

```
"name" : "parse_ipv4",
"min-hdr-length" : 20,
"next-proto" : {
  "field-off" : 9,
  "field-len" : 1,
  "table" : "parse_ipv4_table"
},
"metadata" : {
  "ents" : [
  {
    "name" : "headers.ipv4",
    "type" : "extract",
    "md-off" : 68,
    "hdr-src-off" : 0,
    "length" : 20
...
```

MLIR

- MLIR is an **hybrid** IR
- Several dialects working together
- Multi-level abstractions with different dialects
- Pattern Matching and multi-pass lowering
- A dialect for parsers !

MLIR - netparser dialect

```
"netparser.parseNode"(%arg0, %arg1, %arg2, %arg3, %arg4, %arg5) <{name = @ether_node}> ({
  %0 = arith.subi %arg0, %arg1 : i64
  %c14_i64 = arith.constant 14 : i64
  %1 = arith.cmpi slt, %0, %c14_i64 : i64
  scf.if %1 {  "netparser.stop"() <{fail = 5 : i64}> : () -> ()  }
  %c12_i64 = arith.constant 12 : i64
  %2 = "netparser.load"(%arg2, %c12_i64) <{size = 2 : i64}> : (!raw_buffer, i64) -> i16
  %3 = arith.index_cast %2 : i16 to index
  scf.index_switch %3
  case 2048 {
    "netparser.callParseNode"() <{callee = @ip_overlay_node}> : () -> ()
    scf.yield
  } ...
```

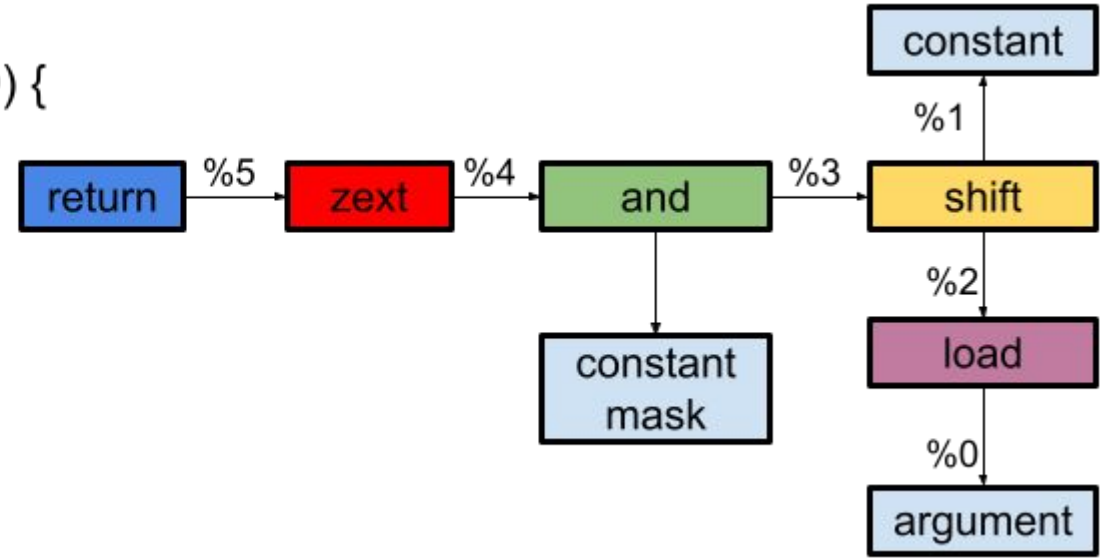
length check

Protocol extraction

Protocol Table

Converting to IR from general purpose languages

```
define i64 @ipv4_length(ptr %0) {
  %2 = load i8, ptr %0, align 4
  %3 = shl i8 %2, 2
  %4 = and i8 %3, 60
  %5 = zext i8 %4 to i64
  ret i64 %5
}
```



Converting to IR from P4

```
state parse_ipv4 {
  packet.extract(headers.ipv4);
  transition select(headers.ipv4.protocol) {
    PROTO_TCP: parse_tcp;
    PROTO_UDP: parse_udp;
  }
}
```

```
"name" : "parse_ipv4",
"min-hdr-length" : 20,
"next-proto" : {
  "field-off" : 9,
  "field-len" : 1,
  "table" : "parse_ipv4_table"
},
"metadata" : {
  "ents" : [
  {
    "name" : "headers.ipv4",
    "type" : "extract",
    "md-off" : 68,
    "hdr-src-off" : 0,
    "length" : 20
...
```

Thanks!