AUTOREV

A framework for automatic firmware reverse engineering

AUTOREV

A framework for automatic firmware reverse enginering

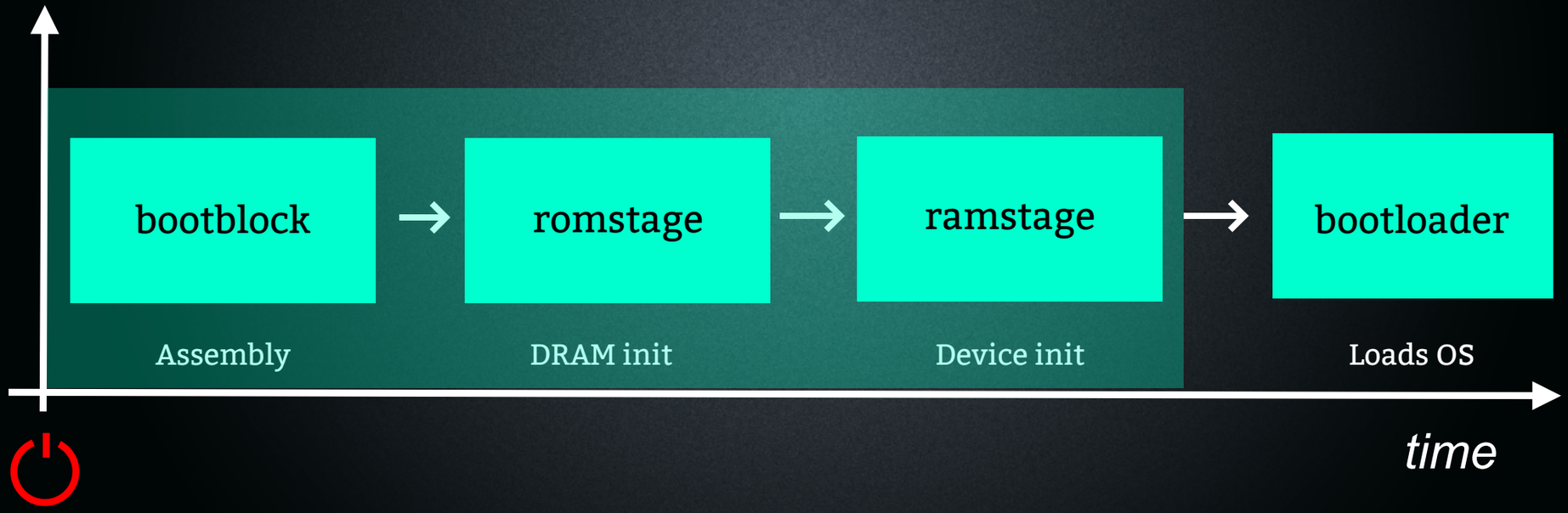
- github.com/9elements/autoref.git
- Written in golang
- Firmware Blackbox testing
- Abstract Syntax Tree generation
- C Code generation
- Backed by MYSQL DB

What is boot firmware?

- Most privilged code running on the machine
- Executed on power on and reset
- Initializes hardware
- Initializes *interfaces* to the hardware for the OS
- Jumps to the bootloader
- Boots into your OS

A closer look at coreboot

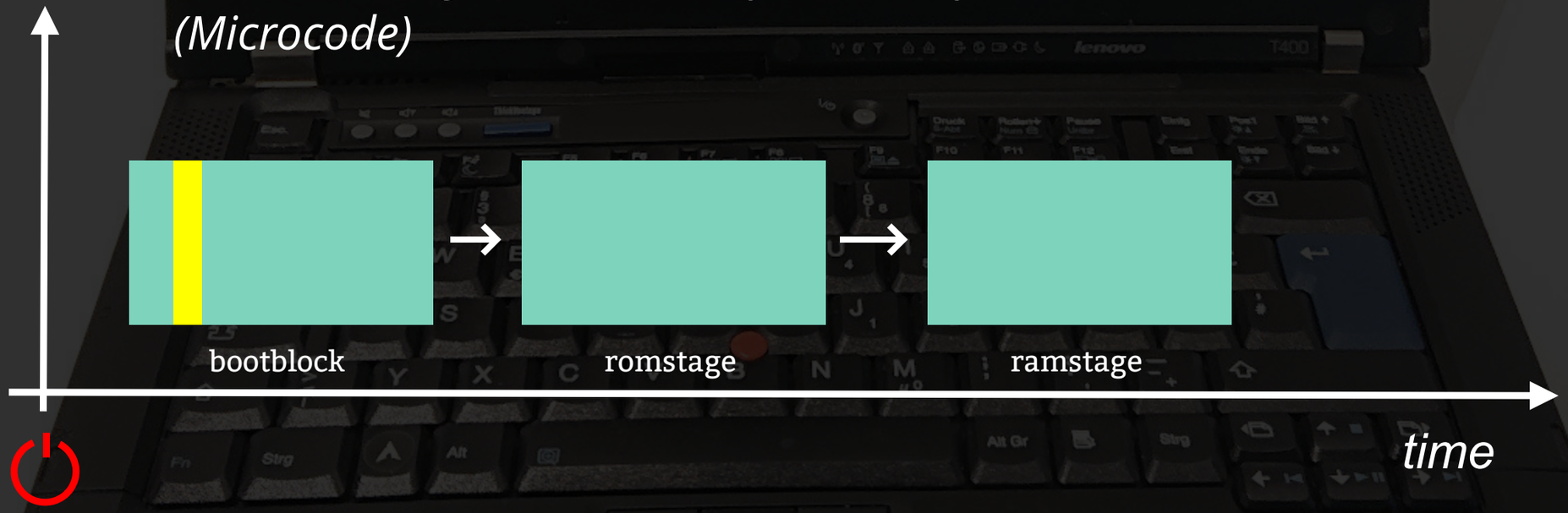
- Written in Assembly/C
- Divided into stages
- Each stages loads the next stage
- Stages initialize hardware required by the next stage



In an ideal world...

coreboot on Lenovo Thinkpad X230

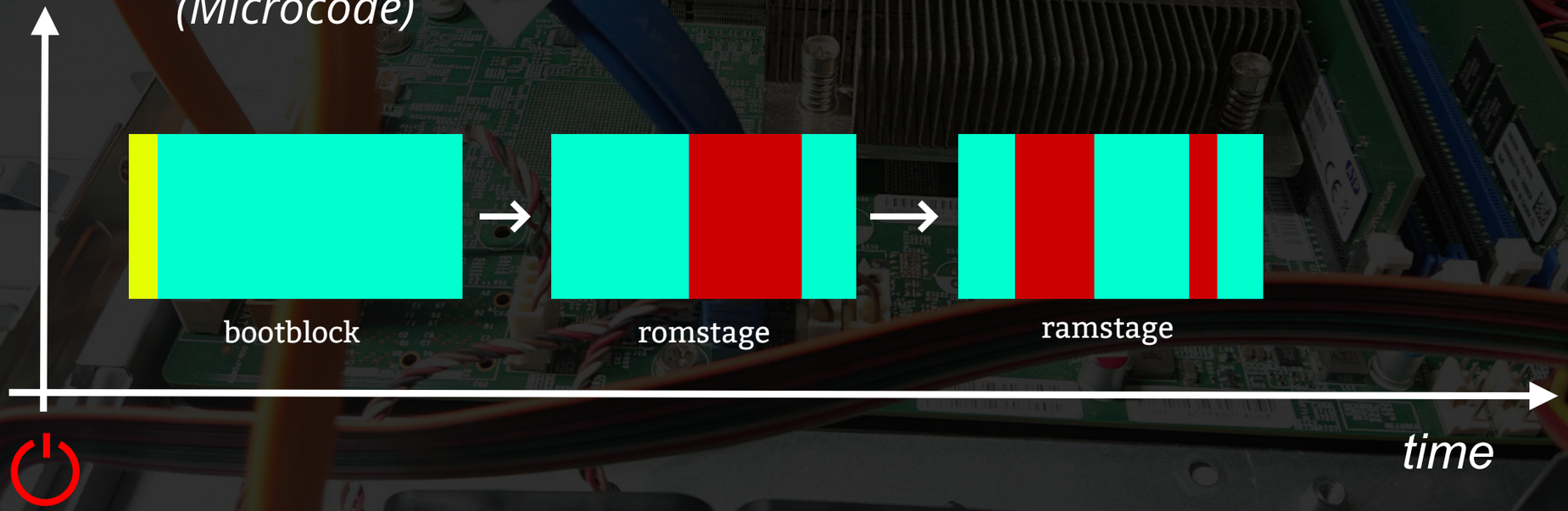
- Reverse engineered and (almost) bug free
- No BLOBs / 100% Open Source
- No free register documentation (lot's of magic values)
- Optained I/O traces from OEM firmware
- *Loads Intel signed CPU ISA updates required to boot (Microcode)*



In an ideal world...

coreboot on Supermicro X11SSH-TF

- Multiple BLOBs, the Firmware Support Package (FSP)
- FSP occupy ~500KiB
- coreboot occupy ~200KiB
- FSP has lots of features that can't be opt out
- coreboot needs to "undo" some settings made by FSP
- *Loads Intel signed CPU ISA updates required to boot (Microcode)*



Problems with BLOBs

- **B**inary **L**arge **Ob**jects
- No source code available
- Usually no documentation what is does
  - Difficult to integrate
  - Security issues
- No linker symbols
  - No unused function garbage collections
  - Size cannot be reduced
- No log output or return codes
  - Difficult to debug

What are we allowed to do?

German UrgG § 69d (3)

[...] is allowed to [...] gain basic knowledge of the used ideas and concepts, by loading, displaying, runnning, transmitting and saving the programm, even without permission granted by the copyright owner.

blackbox testing

German UrgG § 69e (1)

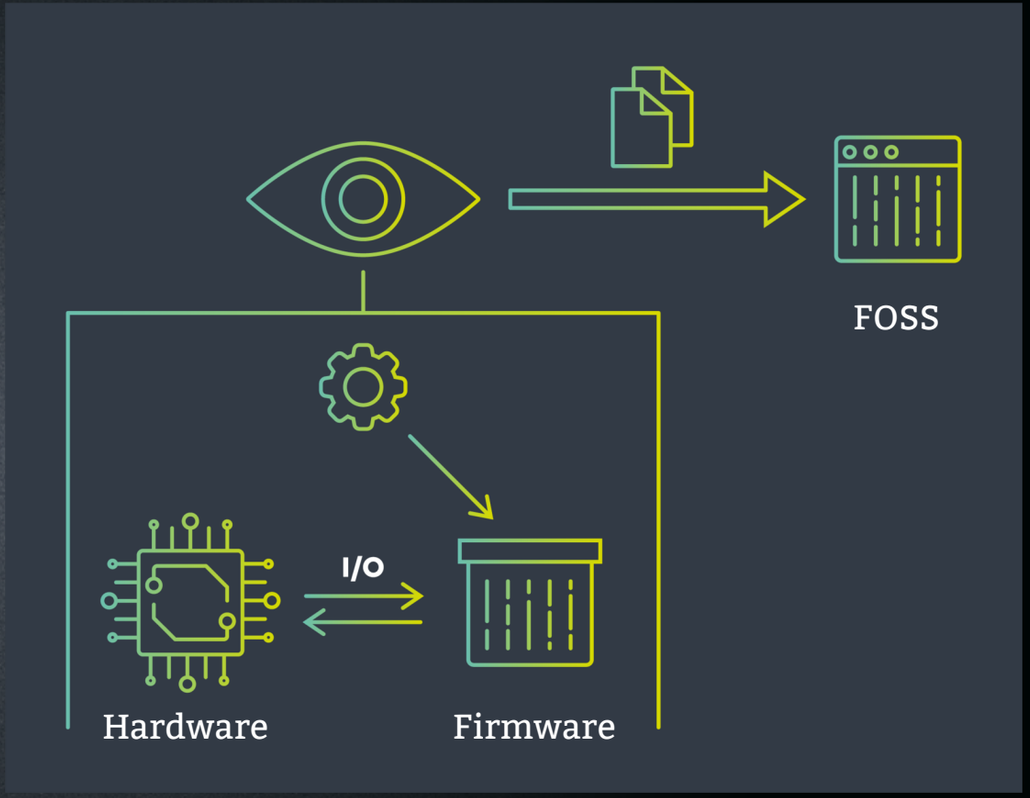
Decompilation is allowed [...] to keep interoperabiltiy [...]

German UrgG § 69e (2)

Information gained by decompilation must not be [...] publicly published [...] or used to create a program doing basically the same.

BlackBox testing

1. Observe/change BIOS options
2. Observe Inputs/Outputs
3. Generate model

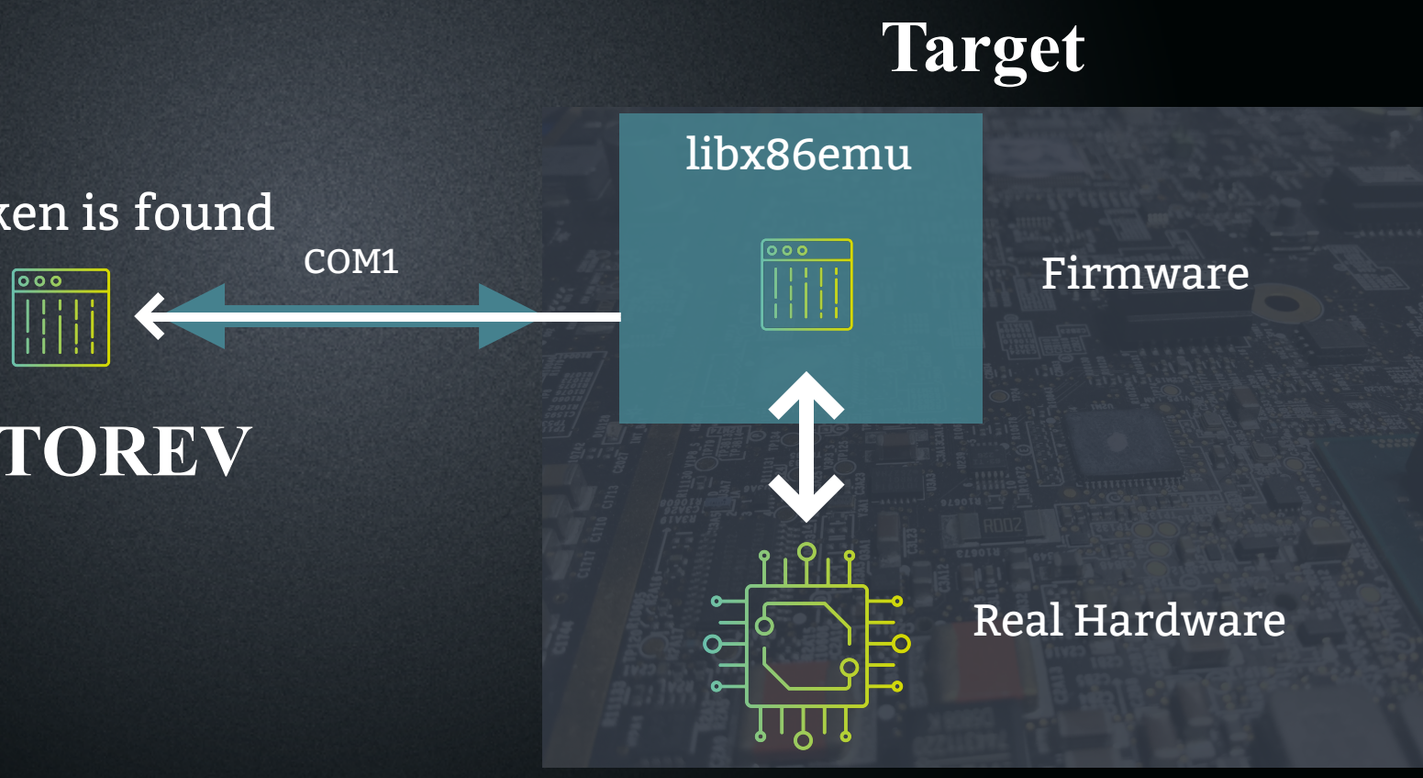


Issues:

- Only captures current hardware
- Corner cases difficult to catch
- Model might be incomplete
- Lot's of data to analyse
- No emulator for low level hardware

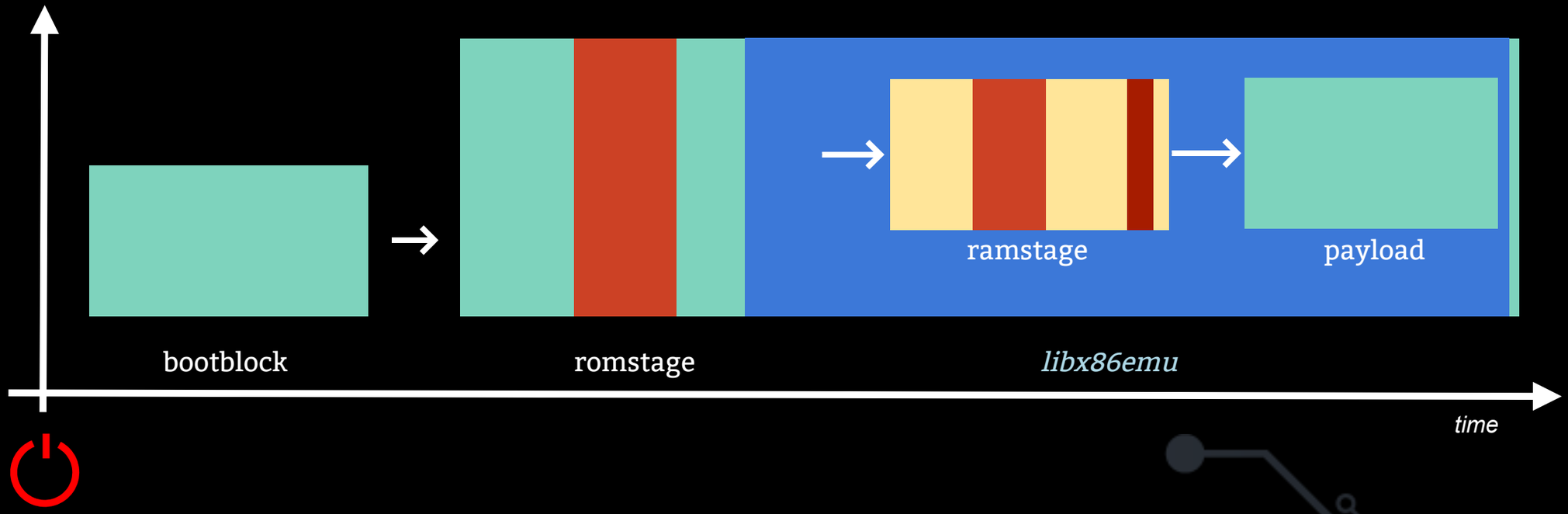
AUTOREV

- Needs real hardware
- SerialICE/Avatar2 run firmware in patched qemu
- Moves virtualisation into firmware
- Traces I/O over serial
- Build in shell allows
  - BIOS option upload
  - Faking I/O
  - Skipping I/O until token is found
- Client/Server model



libx86emu

- https://github.com/wfeldt/libx86emu
- Emulates i386 CPUs only
- Hooks for machines specific instructions
- Patched to output I/O to serial port
- Romstage is never left, but ramstage doesn't know



Firmware inputs/outputs

What needs to be oberserved:

- Memory accesses
  - read32()/write32()
- PCI (MMCONF)
  - pci_read_config32()/pci_write_config32()
- Port accesses
  - inb()/outb()
- MSR (Machine specific register)
  - wrmsr()/rdmsr()
- CPUID
  - cpuid()

Example trace output

```
IP: 7f780737, Type: p, Dir: in, Addr: 000f90d8, Value: 0000000000000400, Access: 16
IP: 7f780764, Type: p, Dir: out, Addr: 000f90d0, Value: 00000000e1000408, Access: 32
IP: 7f780780, Type: p, Dir: out, Addr: 000f90dc, Value: 0000000000000000, Access: 32
IP: 7f78078a, Type: p, Dir: in, Addr: 000f90d8, Value: 0000000000000400, Access: 16
IP: 7f7807ae, Type: p, Dir: out, Addr: 000f90d8, Value: 0000000000000500, Access: 16
IP: 7f7807b8, Type: p, Dir: out, Addr: 000f90da, Value: 000000000000f0e1, Access: 16
IP: 7f7807df, Type: p, Dir: out, Addr: 000f90d4, Value: 0000000008000000, Access: 32
IP: 7f7807e6, Type: p, Dir: in, Addr: 000f90d8, Value: 0000000000000500, Access: 16
IP: 7f7807f8, Type: p, Dir: out, Addr: 000f90d8, Value: 0000000000000501, Access: 16
IP: 7f7807ff, Type: p, Dir: in, Addr: 000f90d8, Value: 0000000000000500, Access: 16
IP: 7f780378, Type: m, Dir: in, Addr: fde10002, Value: 0000000000000000, Access: 8
IP: 7f7803ea, Type: m, Dir: out, Addr: fde10002, Value: 0000000000000002, Access: 8
```

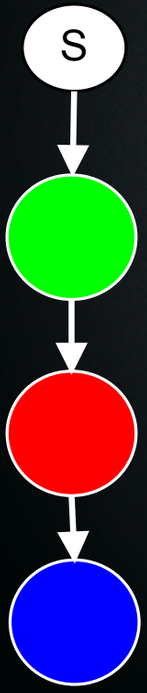
Example trace output

```
pci_write_config16(PCI_DEV(0x3, 0x1c, 0x4), 0xf90da, 0x0000f0e1);
pci_write_config32(PCI_DEV(0x3, 0x1c, 0x4), 0xf90d4, 0x08000000);
pci_read_config16(PCI_DEV(0x3, 0x1c, 0x4), 0xf90d8); // 0x00000500
pci_write_config16(PCI_DEV(0x3, 0x1c, 0x4), 0xf90d8, 0x00000501);
pci_read_config16(PCI_DEV(0x3, 0x1c, 0x4), 0xf90d8); // 0x00000500
read8((void *)0xfde10002); // 0x00000000
write8((void *)0xfde10002, 0x00000002);
read32((void *)0xfdc73418); // 0x00000000
read32((void *)0xfdba2e3c); // 0x00000100
write32((void *)0xfdba2e3c, 0x00000100);
read32((void *)0xfdc73418); // 0x00000000
pci_read_config16(PCI_DEV(0x3, 0x1d, 0x0), 0xfa000); // 0x00008086
```

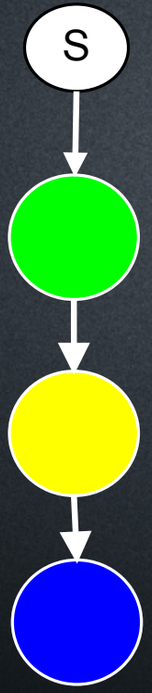
Abstract Syntax Tree (AST)

- AUTOREV support changing and iterating over BIOS Options
- Generates an AST containing nodes
- Example: color = Hash(traced I/O value)
- **LCS** algorithm to generate minimal AST

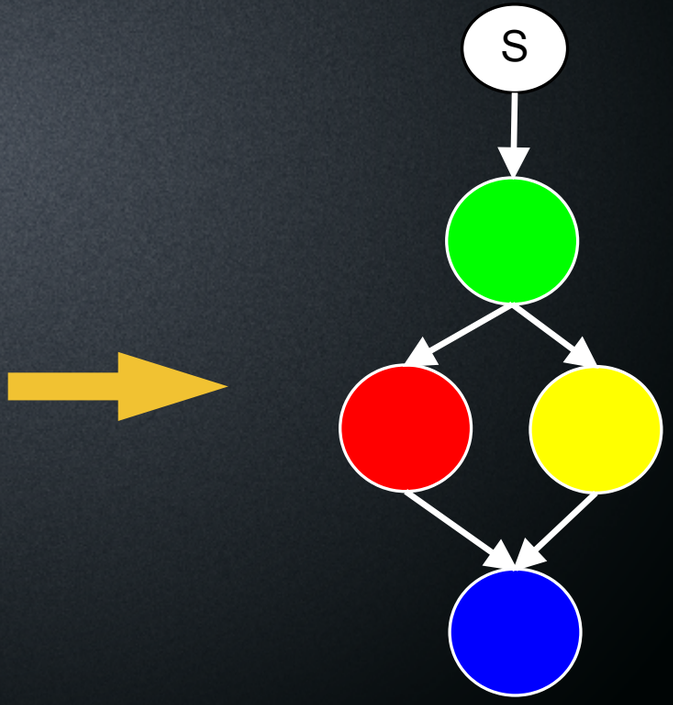
BIOS Option 1 = True



BIOS Option 1 = False

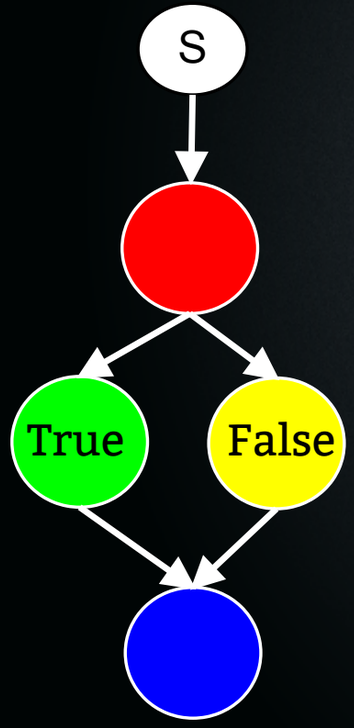


Merge into optimal mesh



AST to HLS conversion

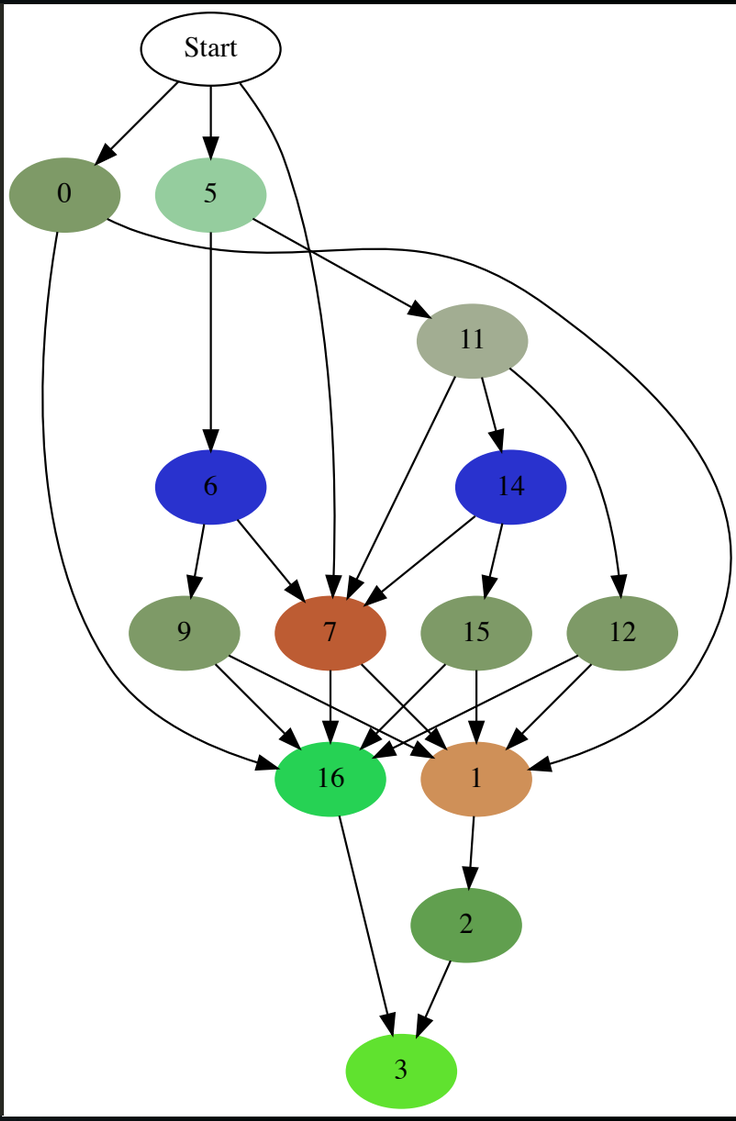
- Automatic addition of metadata to every node
- Analysing the mesh
- Finding conditions to BIOS options
- Generating C code




```
1 read32(0xdeadbeef);
2 if (option[1]) {
3         write32(0xdeadbeef, 1);
4 } else {
5         write32(0xdeadbeef, 2);
6 }
7 outb(0x1234, 0x80);
```

QEMU Demo

```
1  outw(0xddaa, 0x80);
2
3  if (config) {
4    struct testconfig {
5    uint32_t config1;
6    uint32_t config2;
7    uint32_t config3;
8    uint32_t config4;
9    } *myconfig = (struct testconfig *)config;
10
11   if (myconfig->config1 || myconfig->config2) {
12         pci_write_config8(dev, Q35_PAM0 + 6, 0x30);
13   }
14   if (myconfig->config1) {
15         pci_read_config32(dev, Q35_PAM0);
16   }
17   if (myconfig->config2) {
18         outb(0x00, 0x80);
19   }
20   pci_write_config8(dev, Q35_PAM0 + 2, myconfig->config3);
21   if (myconfig->config4) {
22          pci_write_config8(dev, Q35_PAM0 + 1, myconfig->config4 + 1);
23   } else {
24     outb(0x12, 0x80);
25     outb(0x34, 0x80);
26   }
27 }
28 outw(0xaadd, 0x80);
```



QEMU Demo

Reconstructed program

```
if ((BiosOption1 == 0 && BiosOption2 == 0 && BiosOption3 == 1)) {
  pci_write_config8(PCI_DEV(0x0, 0x0, 0x0), 0x0092, 0x00000001);
  if ((BiosOption4 == 0)) {
    outb(0x12, 0x0080);
    outb(0x34, 0x0080);
  }
  else if ((BiosOption1 == 1 && BiosOption3 == 1 && BiosOption4 == 1)) {
    pci_write_config8(PCI_DEV(0x0, 0x0, 0x0), 0x0091, 0x00000002);
  }
}
else if ((BiosOption2 == 1) ||
    (BiosOption1 == 1 && BiosOption2 == 0)) {
  pci_write_config8(PCI_DEV(0x0, 0x0, 0x0), 0x0096, 0x00000030);
  if ((BiosOption1 == 0 && BiosOption2 == 1)) {
    outb(0x0, 0x0080);
    if ((BiosOption3 == 0)) {
      pci_write_config8(PCI_DEV(0x0, 0x0, 0x0), 0x0092, 0x00000000);
      if ((BiosOption4 == 0)) {
        outb(0x12, 0x0080);
        outb(0x34, 0x0080);
      }
      else if ((BiosOption3 == 1 && BiosOption4 == 1 && BiosOption1 == 1)) {
        pci_write_config8(PCI_DEV(0x0, 0x0, 0x0), 0x0091, 0x00000002);
      }
    }
    else if ((BiosOption2 == 1 && BiosOption3 == 1 && BiosOption1 == 0)) {
      pci_write_config8(PCI_DEV(0x0, 0x0, 0x0), 0x0092, 0x00000001);
      if ((BiosOption4 == 0)) {
        outb(0x12, 0x0080);
        outb(0x34, 0x0080);
```

TODOs

- Adding plugin support
- Loop detection
- Read-Modify-Write detection
- Dead code detection
- Segfault/reboot detection
- Pretty High Language generation
- Optimise code

Can we reverse the complete FSP?

To test:

- 4 CPU sockets
  - 4 CPUs each socket
- 4 DIMMs
  - 4 DIMMS on each socket
- 2 PCH
- 16 USB ports
- 32 PCIe lanes
  - empty slot, Gen1, Gen2, Gen3
- 8 SATA ports
- > 512 UPDs

$$2^{588} * 15 \text{ minutes} = \ldots$$

Aaaand then you have to analyse the AST...

patrick.rudolph@9elements.com

u-root.slack.com slack