Flash Memory Summit

Design Register Accurate SSD Software Simulator

haocheng.huang@starblaze-tech.com

IP Design Manager

Beijing Starblaze Technology

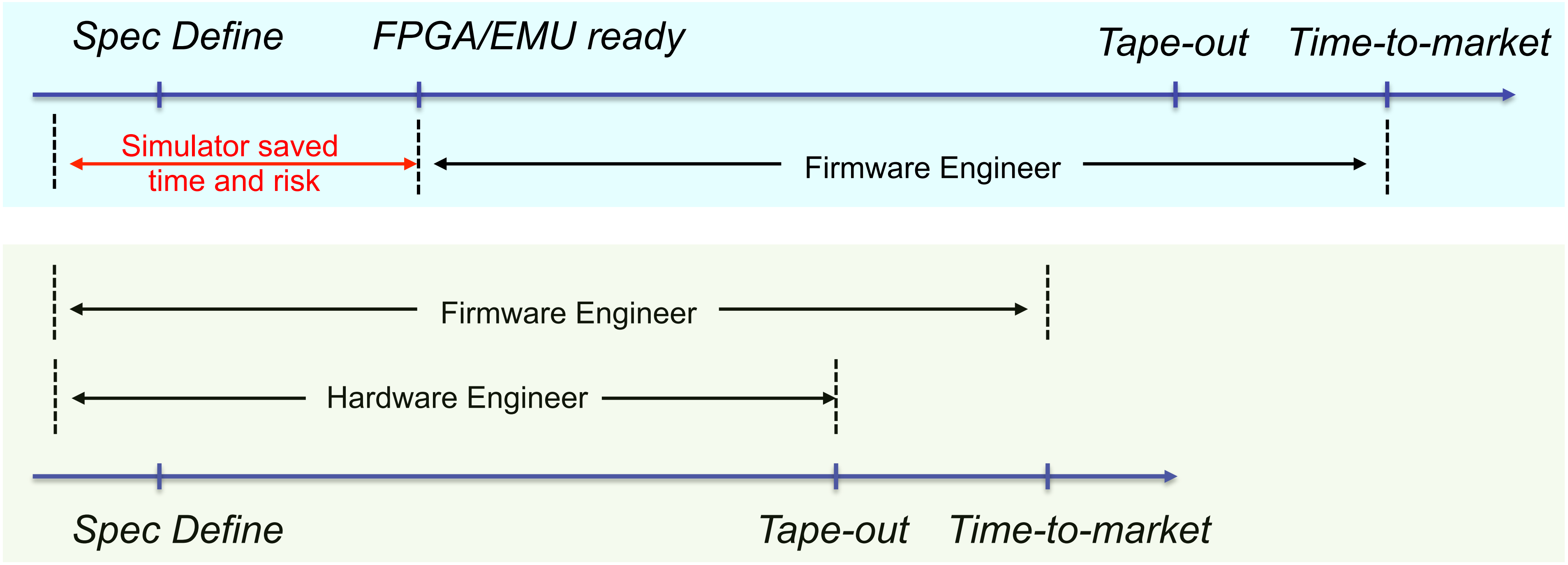
SSD Controller Design Challenge

- SSD Controller chip is hardware + firmware
  - Firmware determines the major features of SSD Controller.
  - To get best performance and power consumption, firmware needs to be fine tuned on well-optimized hardware.

- Both hardware and firmware are customized
  - Most hardware components are designed from scratch and need to be carefully optimized according to firmware usage.
  - SSD firmware is very customized and optimized to fit the hardware.

SSD Controller chip design needs very close firmware and hardware co-design!

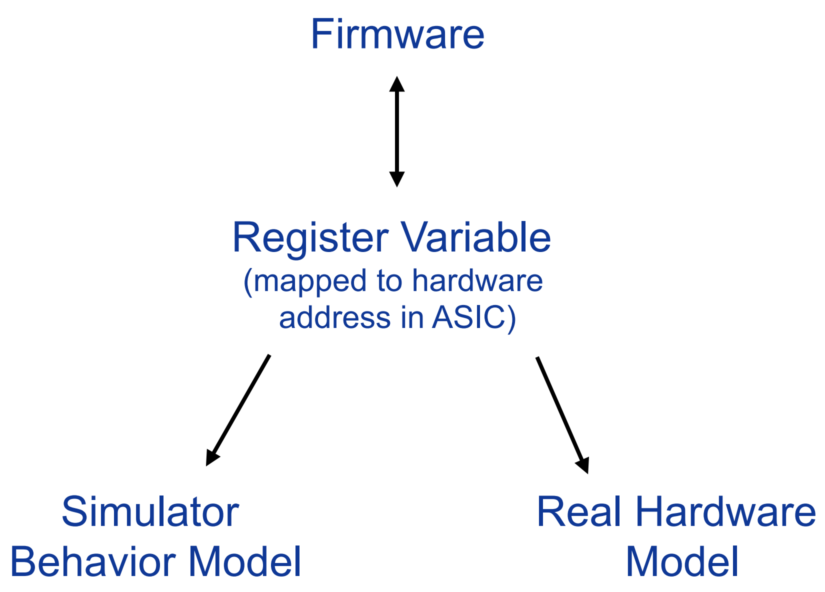
SSD Controller Design Schedule



Agenda

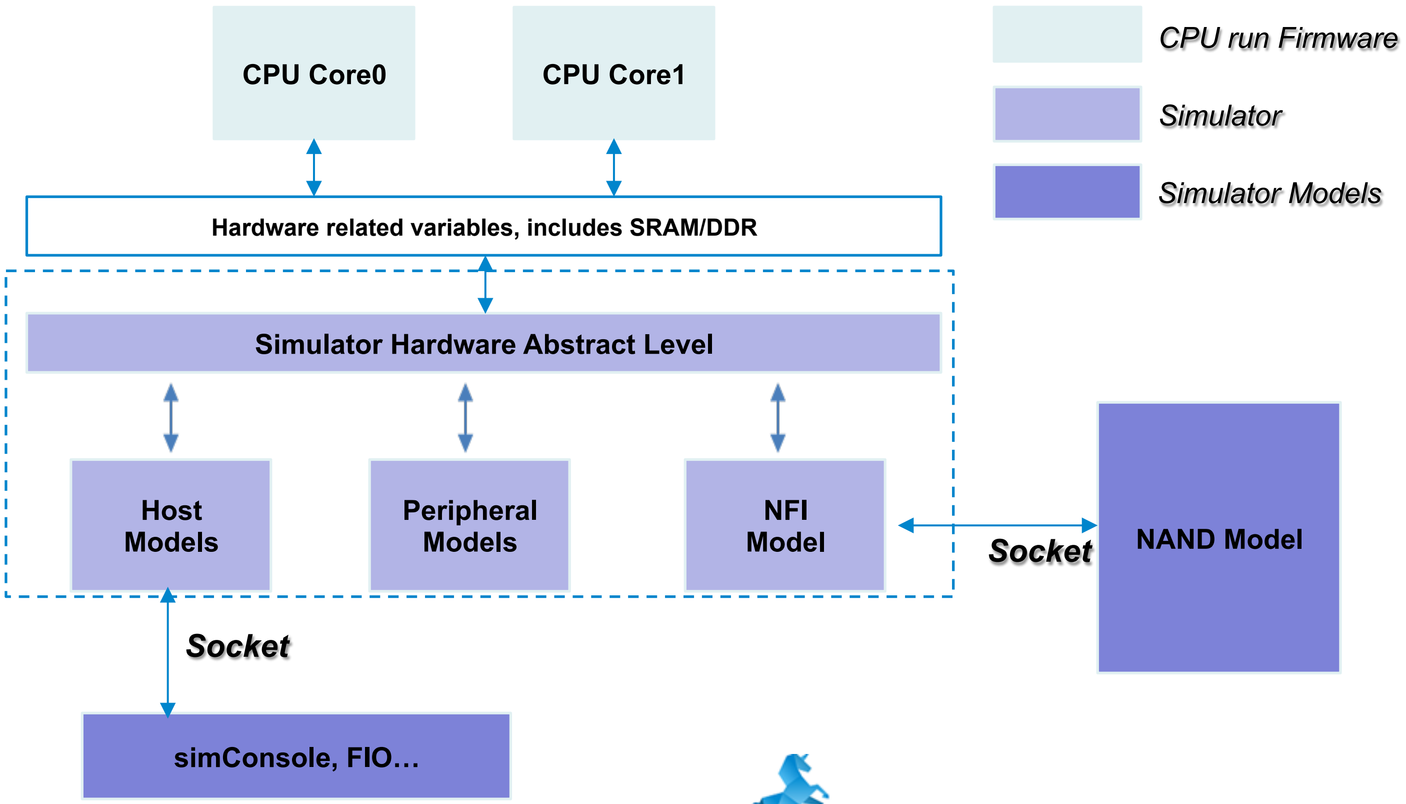
1. **Simulator Overview**
2. SSD Key Models：
    1. Peripheral Model
    2. NAND Model
    3. Host Model
3. Software-driven Design flow

Simulator Overview

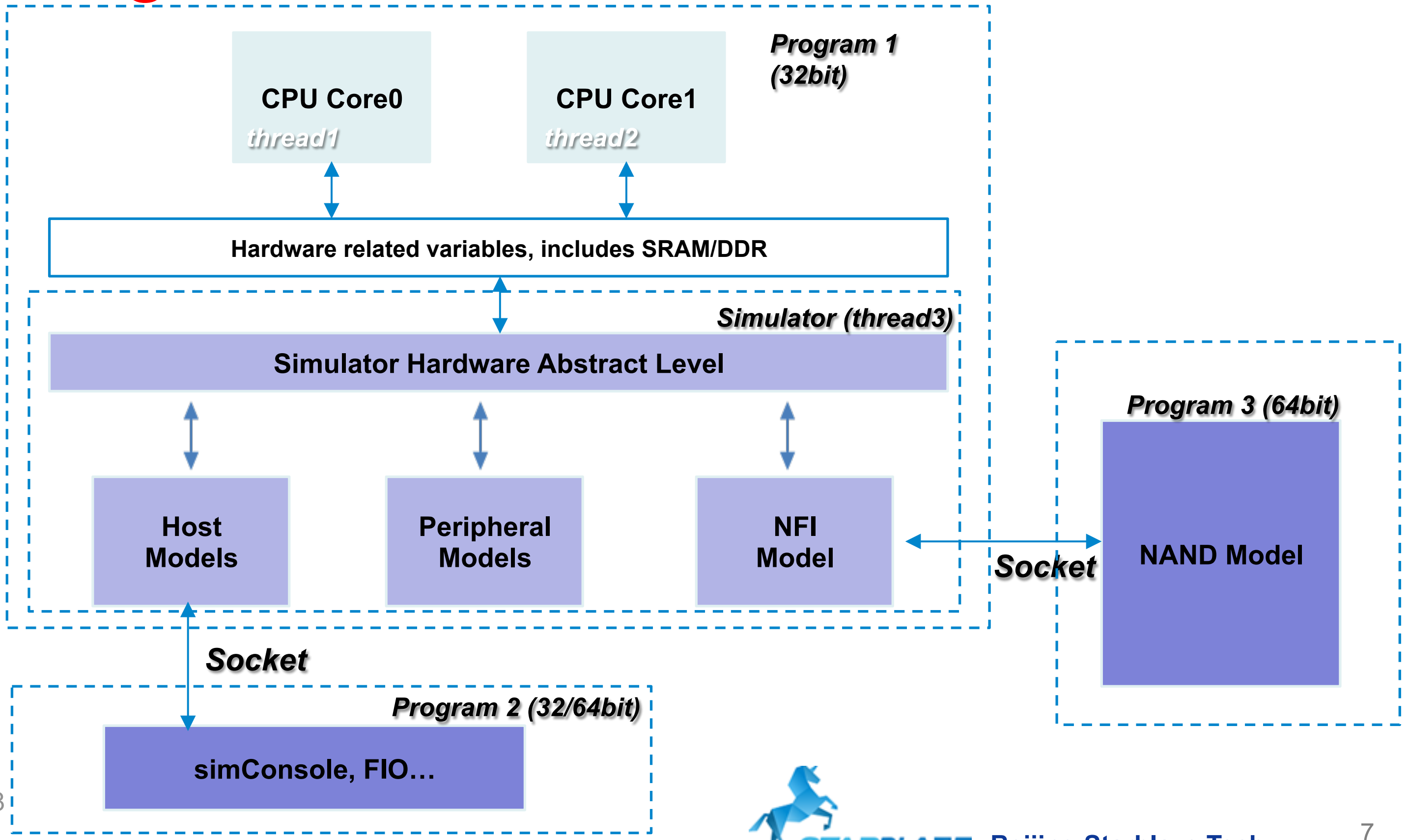


- In most SOC system, embedded firmware runs on embedded CPUs, and access the hardware resource by writing/reading the registers that mapped to the bus.
- To simulate the SSD SOC system, we run firmware on x86 system (compiled by GCC or Visual Studio). The hardware modules are written in C/C++ as behavior model, and are accessible by firmware through the register variables.

Starblaze Simulator Architecture



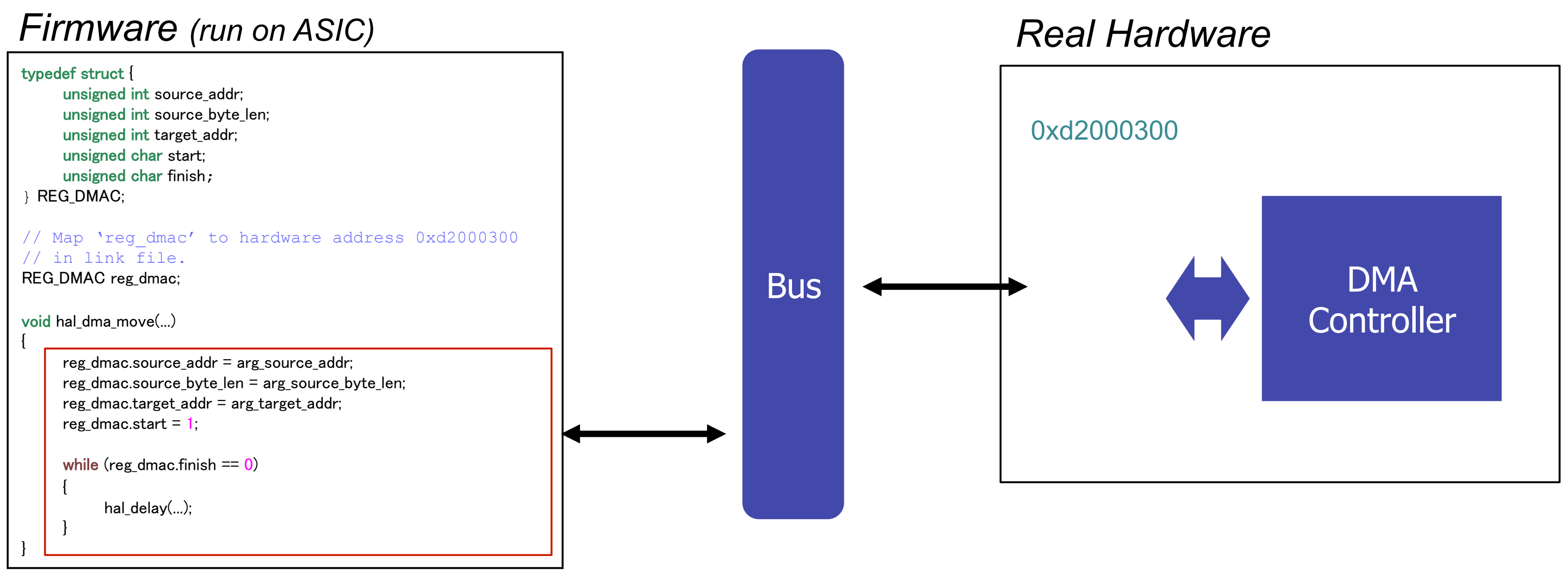
Program/Thread Partition



Agenda

1. Simulator Overview
2. **SSD Key Models：**
   1. **Peripheral Model**
   2. NAND Model
   3. Host Model
3. Software-driven Design flow

Peripheral Model



Firmware *(run on ASIC)*

```
typedef struct {
        unsigned int source_addr;
        unsigned int source_byte_len;
        unsigned int target_addr;
        unsigned char start;
        unsigned char finish;
} REG_DMAC;

// Map 'reg_dmac' to hardware address 0xd2000300
// in link file.
REG_DMAC reg_dmac;

void hal_dma_move(...)
{
        reg_dmac.source_addr = arg_source_addr;
        reg_dmac.source_byte_len = arg_source_byte_len;
        reg_dmac.target_addr = arg_target_addr;
        reg_dmac.start = 1;

        while (reg_dmac.finish == 0)
        {
                hal_delay(...);
        }
}
```

Peripheral Model

Firmware *(run on x86)*

```
typedef struct {
        unsigned int source_addr;
        unsigned int source_byte_len;
        unsigned int target_addr;
        unsigned char start;
        unsigned char finish;
} REG_DMAC;

// Simulator model will share this global variable
REG_DMAC reg_dmac;

void hal_dma_move(...)
{
        reg_dmac.source_addr = arg_source_addr;
        reg_dmac.source_byte_len = arg_source_byte_len;
        reg_dmac.target_addr = arg_target_addr;
        reg_dmac.start = 1;

        while (reg_dmac.finish == 0)
        {
                hal_delay(...);
        }
}
```

reg_dmac

Shared variable between firmware and simulator model code.

Simulator Behavior Model

```
// Share the same 'reg_dmac' variable
extern REG_DMAC reg_dmac;

void model_dmac()
{
    if (reg_dmac.start) {
        // Start DMA using configuration
        // information from variable
        // 'reg_dmac'.
        reg_dmac.finish = 1;
    }
}
```

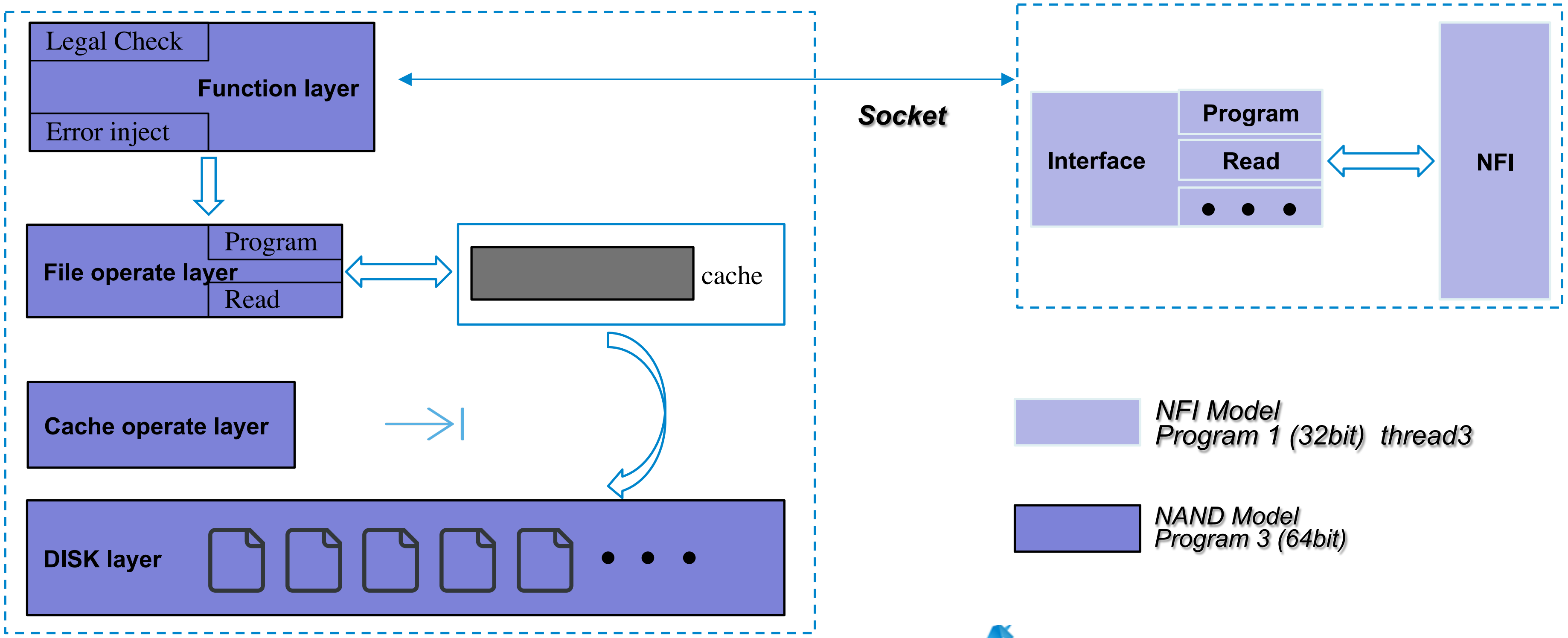
Beijing Starblaze Tech

Agenda

1. Simulator Overview
2. **SSD Key Models：**
    1. Peripheral Model
    2. **NAND Model**
    3. Host Model
3. Software-driven Design flow

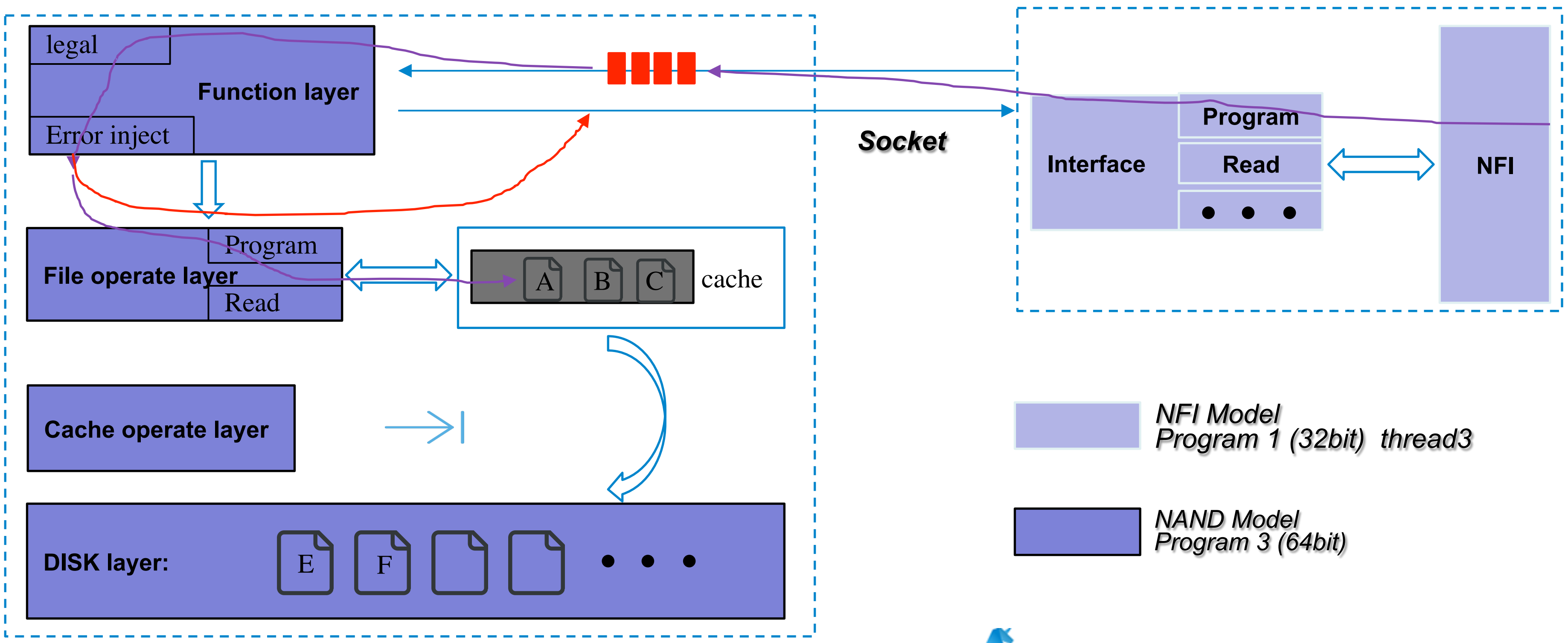
Flash Memory Summit

NAND Model



▨	*NFI Model Program 1 (32bit) thread3*
▨	*NAND Model Program 3 (64bit)*

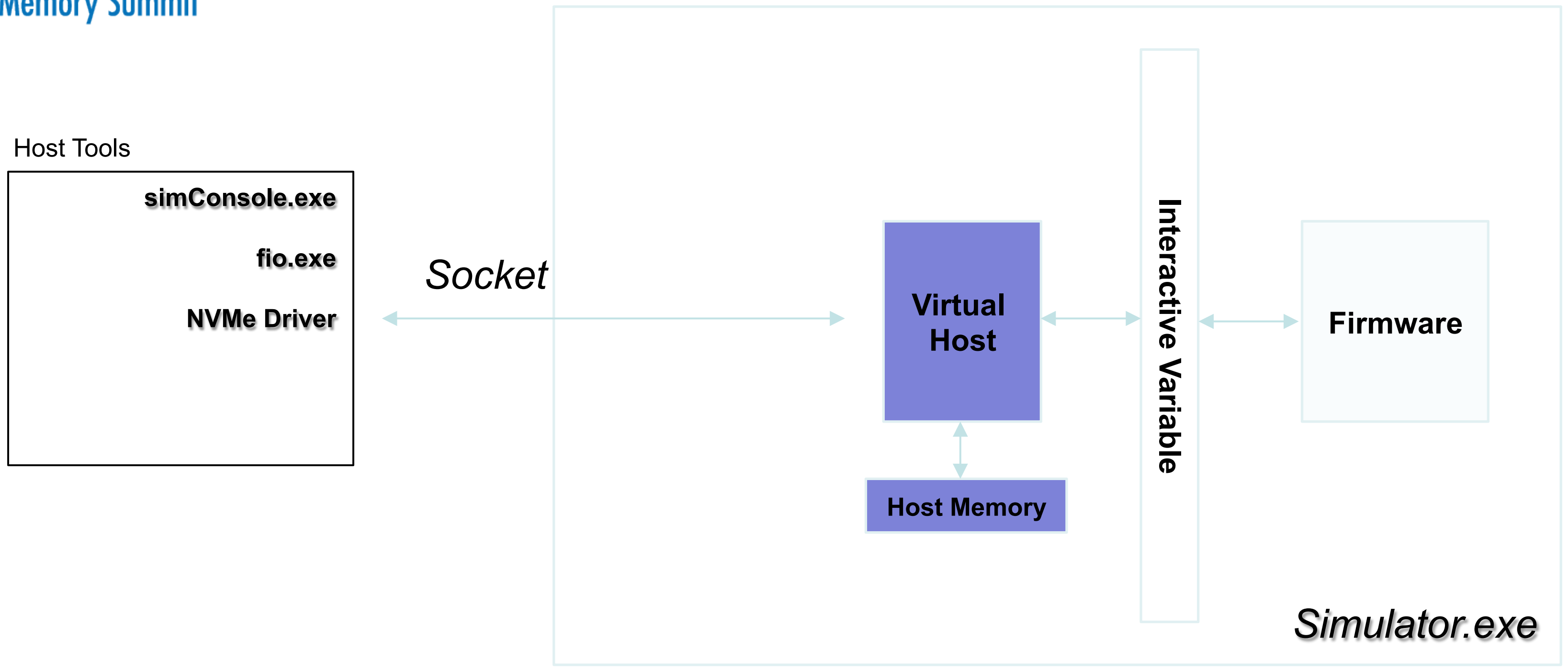
NAND Model


legal
Function layer
Error inject
Program
File operate layer
Read
A
B
C
cache
Cache operate layer
DISK layer:
E
F
Socket
Interface
Program
Read
NFI
NFI Model
Program 1 (32bit) thread3
NAND Model
Program 3 (64bit)

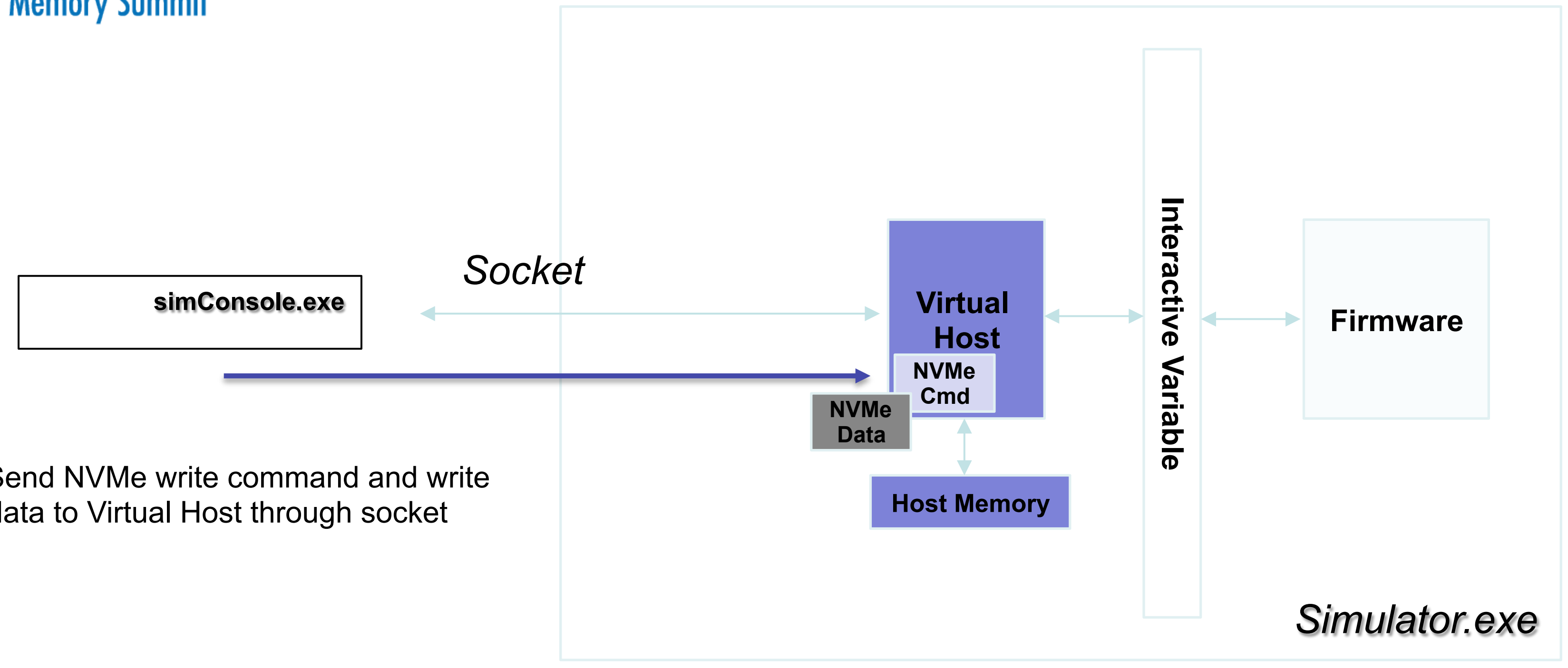
Agenda

1. Simulator Overview
2. **SSD Key Models：**
   1. Peripheral Model
   2. NAND Model
   3. **Host Model**
3. Software-driven Design flow

Host Model Overview

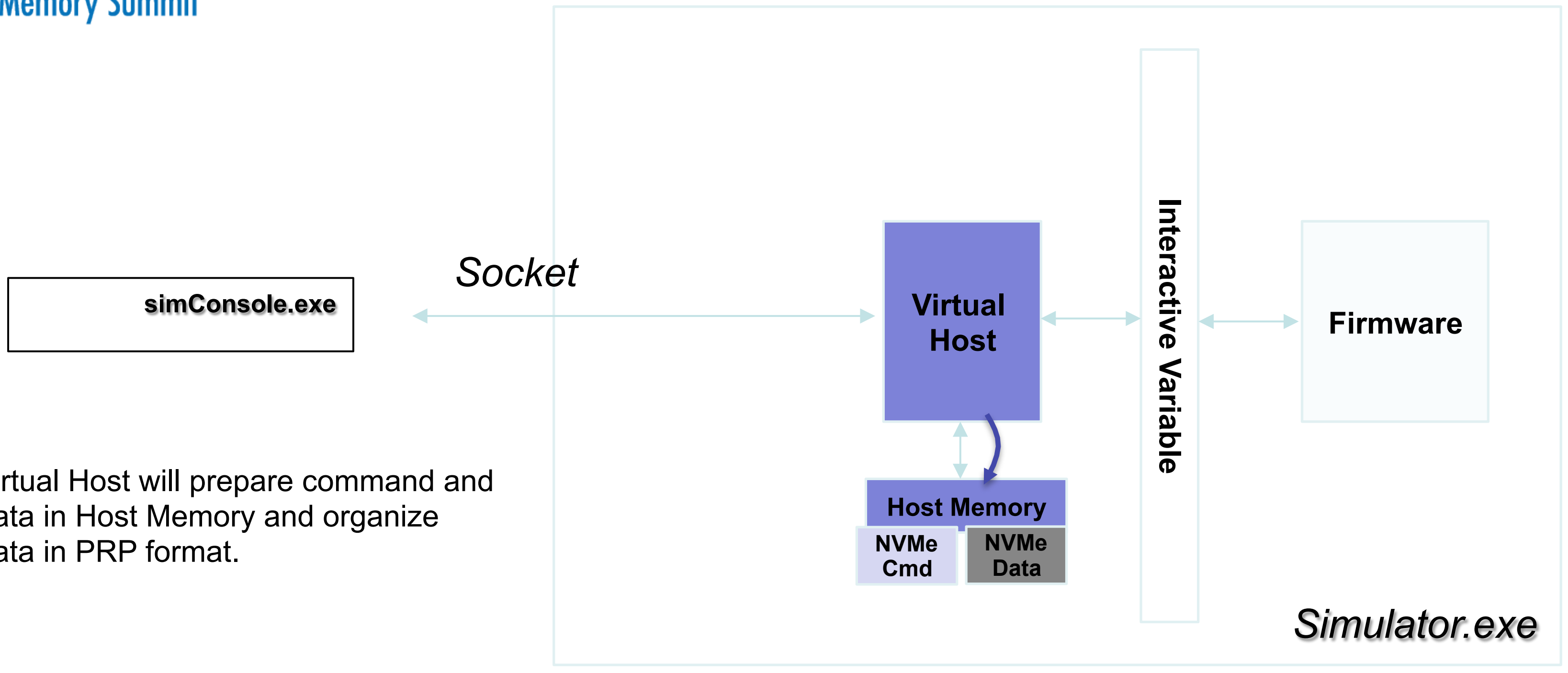


NVMe Write



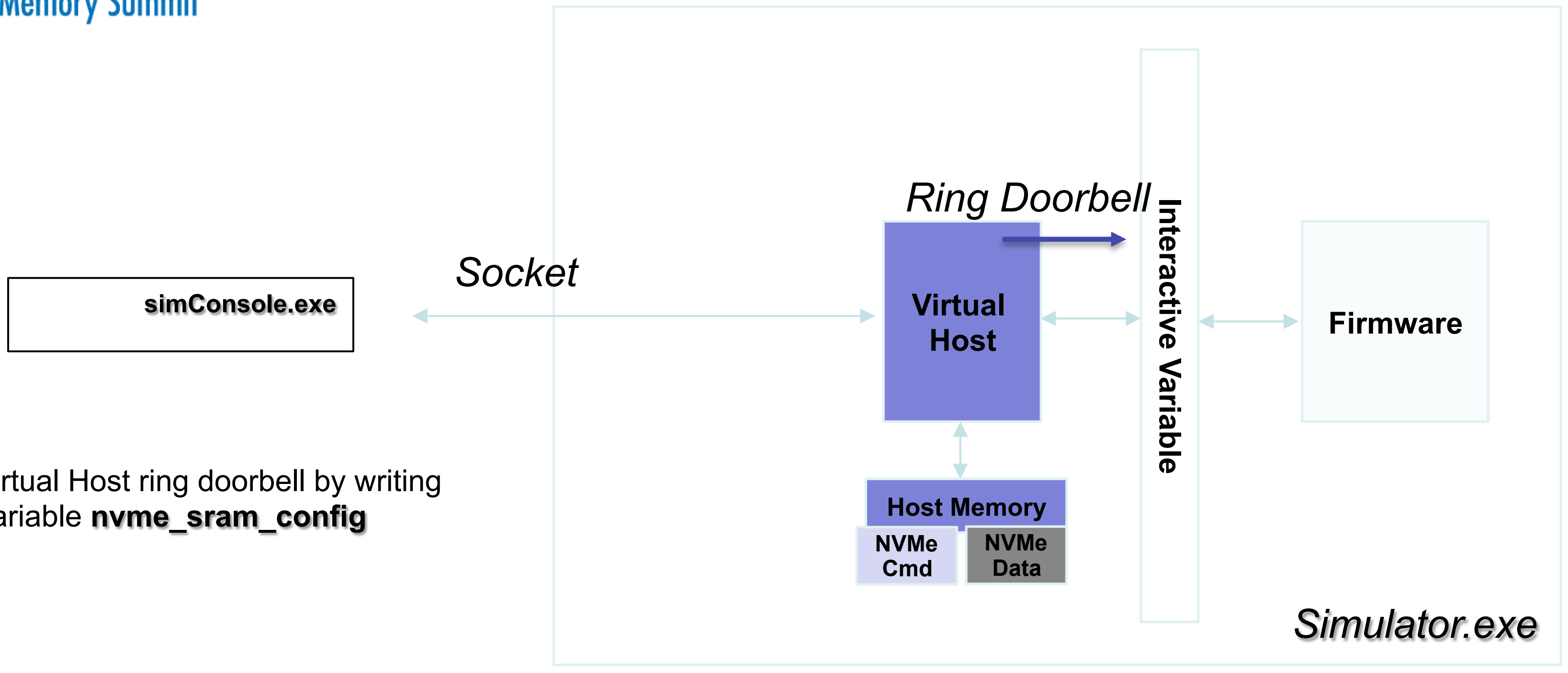
1. Send NVMe write command and write data to Virtual Host through socket

NVMe Write



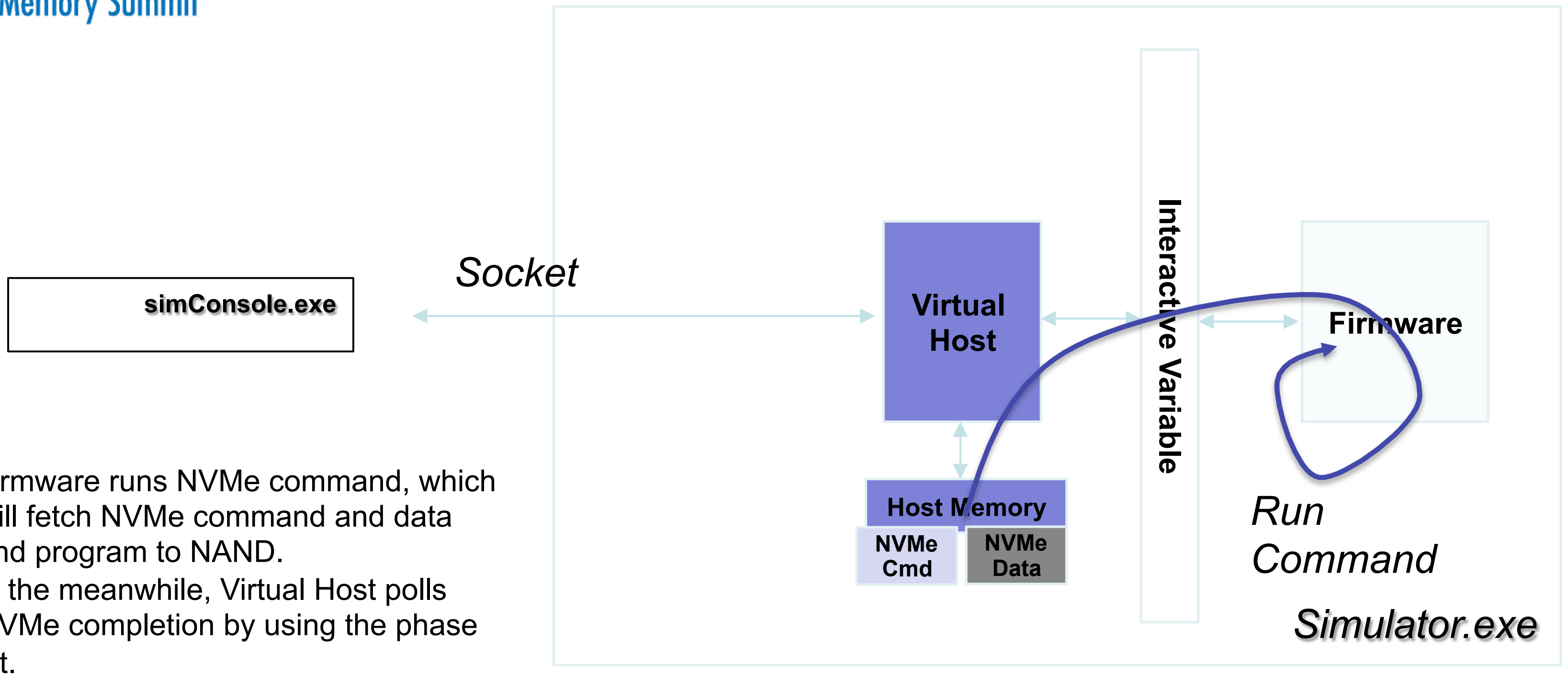
2. Virtual Host will prepare command and data in Host Memory and organize data in PRP format.

NVMe Write



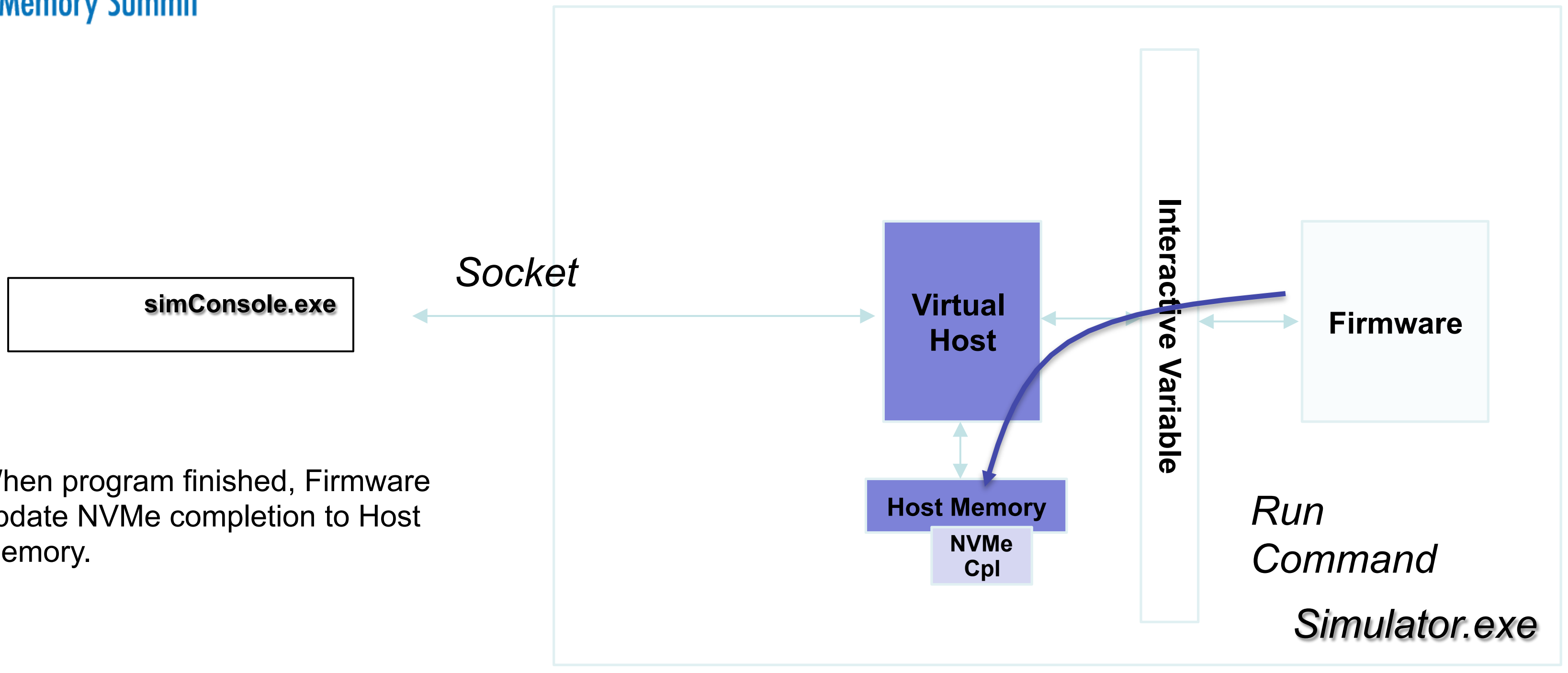
3. Virtual Host ring doorbell by writing variable **nvme_sram_config**

NVMe Write



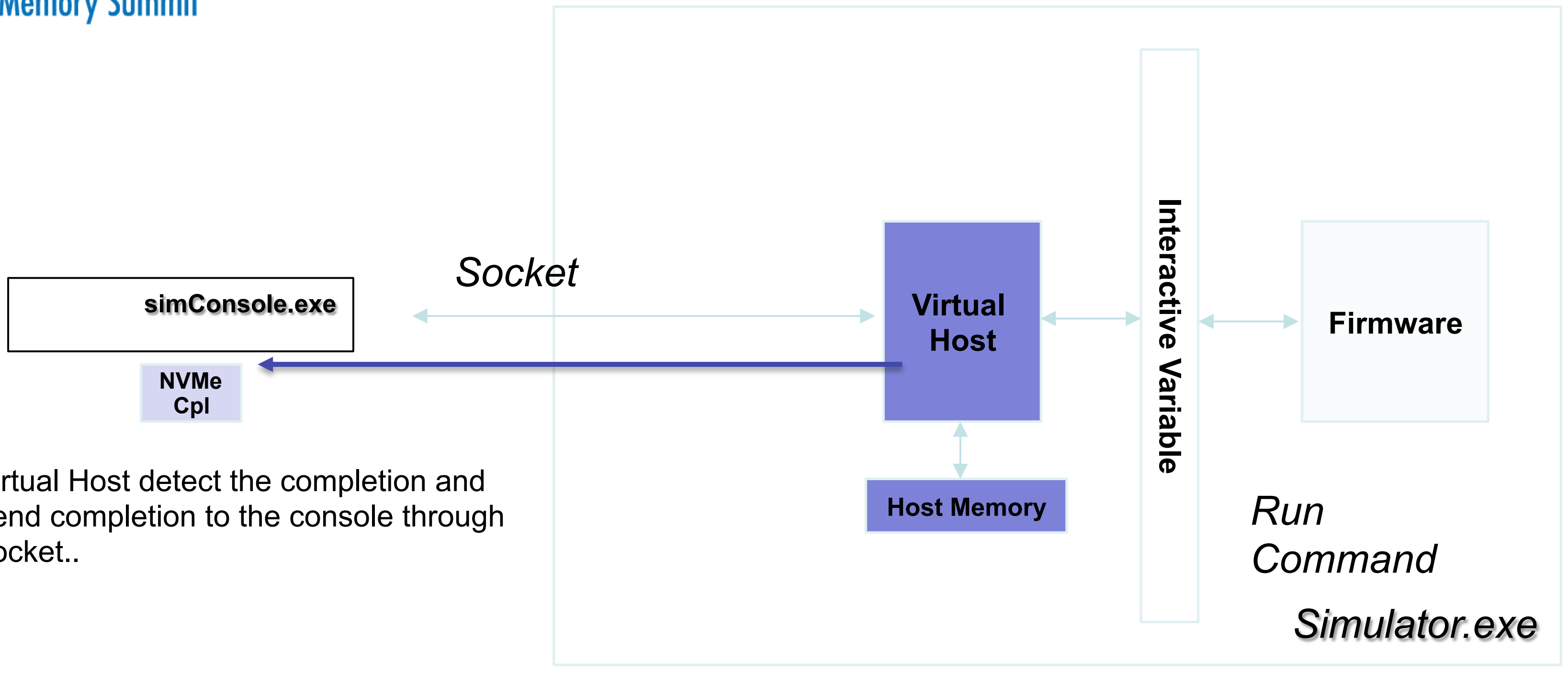
4. Firmware runs NVMe command, which will fetch NVMe command and data and program to NAND.
   In the meanwhile, Virtual Host polls NVMe completion by using the phase bit.

NVMe Write



5. When program finished, Firmware update NVMe completion to Host Memory.

NVMe Write

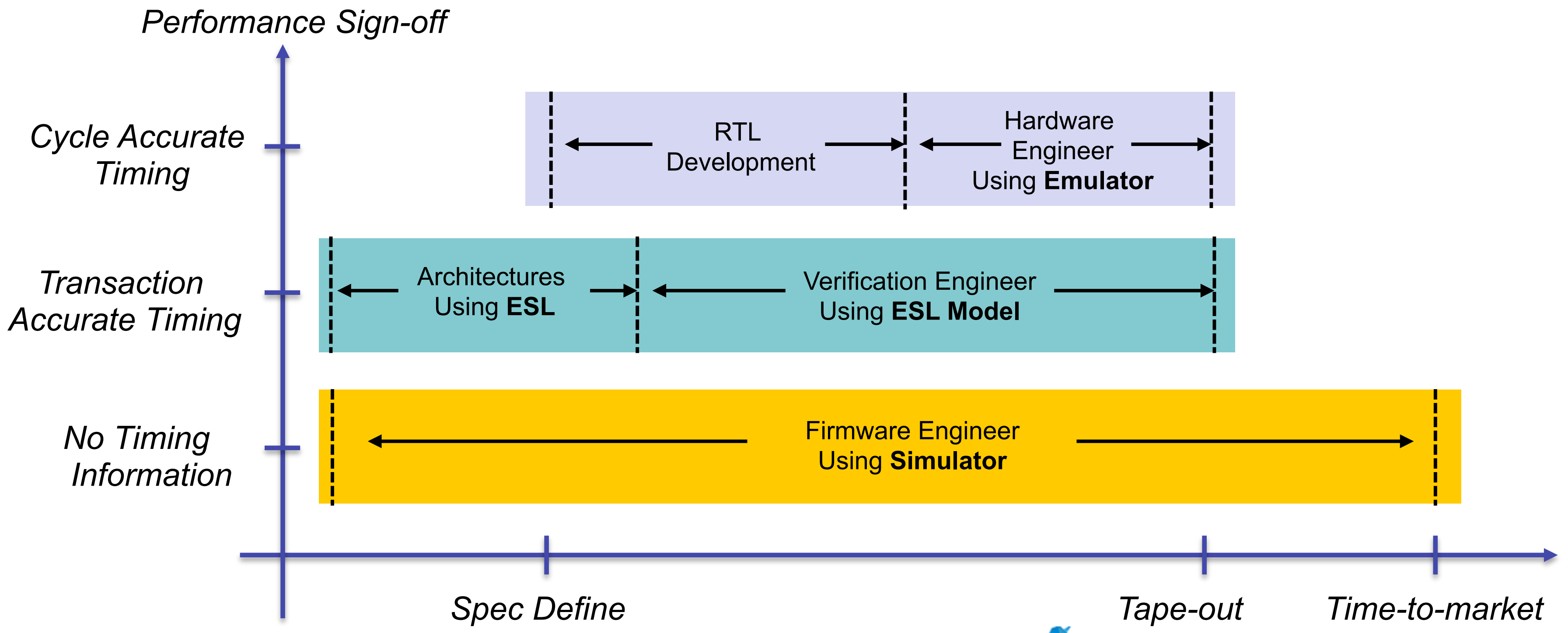


6. Virtual Host detect the completion and send completion to the console through socket..

Agenda

1. Simulator Overview
2. SSD Key Models：
    1. Peripheral Model
    2. NAND Model
    3. Host Model
3. **Software-driven Design flow**

Starblaze Software-driven Design Flow



Performance Sign-off
Cycle Accurate Timing
Transaction Accurate Timing
No Timing Information
RTL Development
Hardware Engineer Using **Emulator**
Architectures Using **ESL**
Verification Engineer Using **ESL Model**
Firmware Engineer Using **Simulator**
Spec Define
Tape-out
Time-to-market

Q & A

- Starblaze Booth: 649

Invitation	Topic	Presenter
SSDS-102-1 Tuesday August 7th Ballroom F,5:05pm to 5:22pm	Low-Power Design of SSDs	Daniel Sun
SOFT-201-1 Wednesday August 8th GAMR 3, 8:30am to 10:50am	Key-Value Store Friendly SSD Interface Design and Optimization	Teng Yang
CTRL-302A-1 Thursday August 9th Ballroom A, 2:10pm to 3:25pm	Take Full Advantage of LDPC Soft Bit Decoding	Feng Tang
EMBD-302B-1 Thursday, August 9th, Ballroom A,3:40-4:10 pm	Computing Storage in the AI IoT Era	Daniel Sun