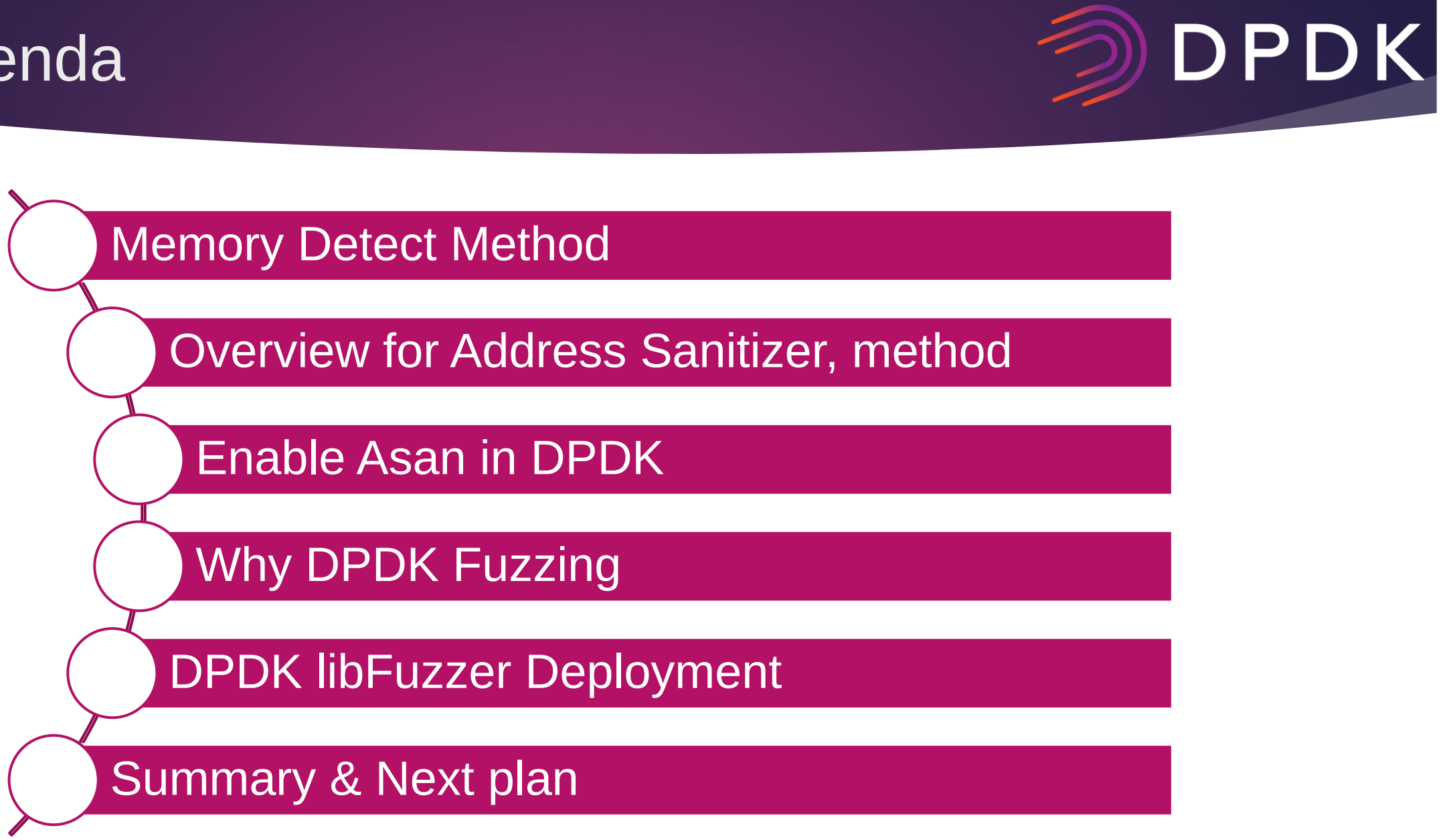
# Memory Error Detect Enhancement for DPDK Fuzzing

XUEQIN LIN, YINAN WANG， ZHAOYAN CHEN

INTEL

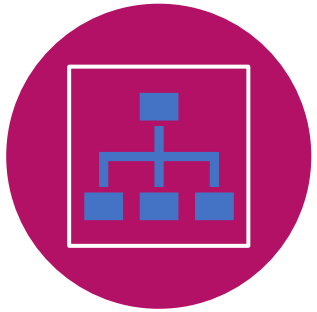
# Agenda


Memory Detect Method
Overview for Address Sanitizer, method
Enable Asan in DPDK
Why DPDK Fuzzing
DPDK libFuzzer Deployment
Summary & Next plan

# Background

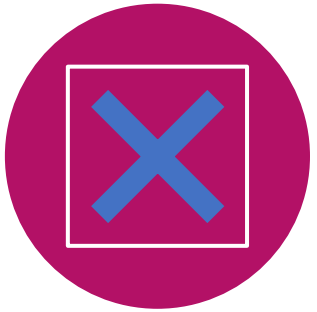
Hard debug

Critical impact



Insufficient

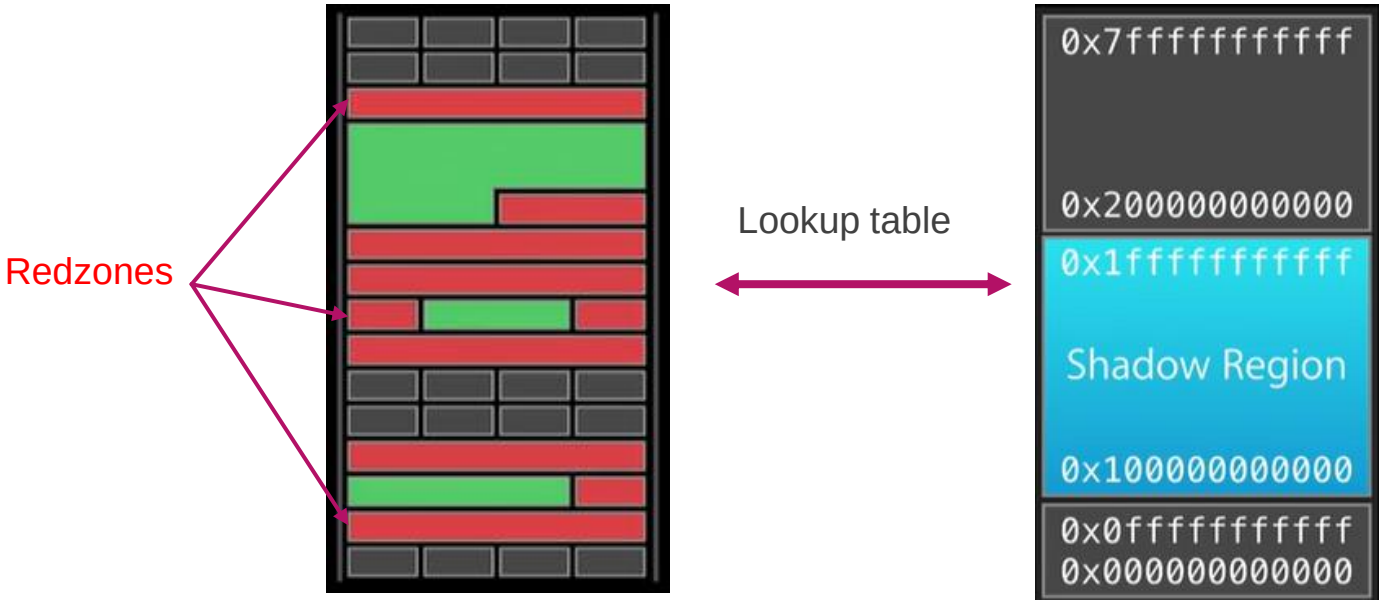
management for DPDK
RTE_MALLOC_DEBUG



No memory tool could use directly on DPDK

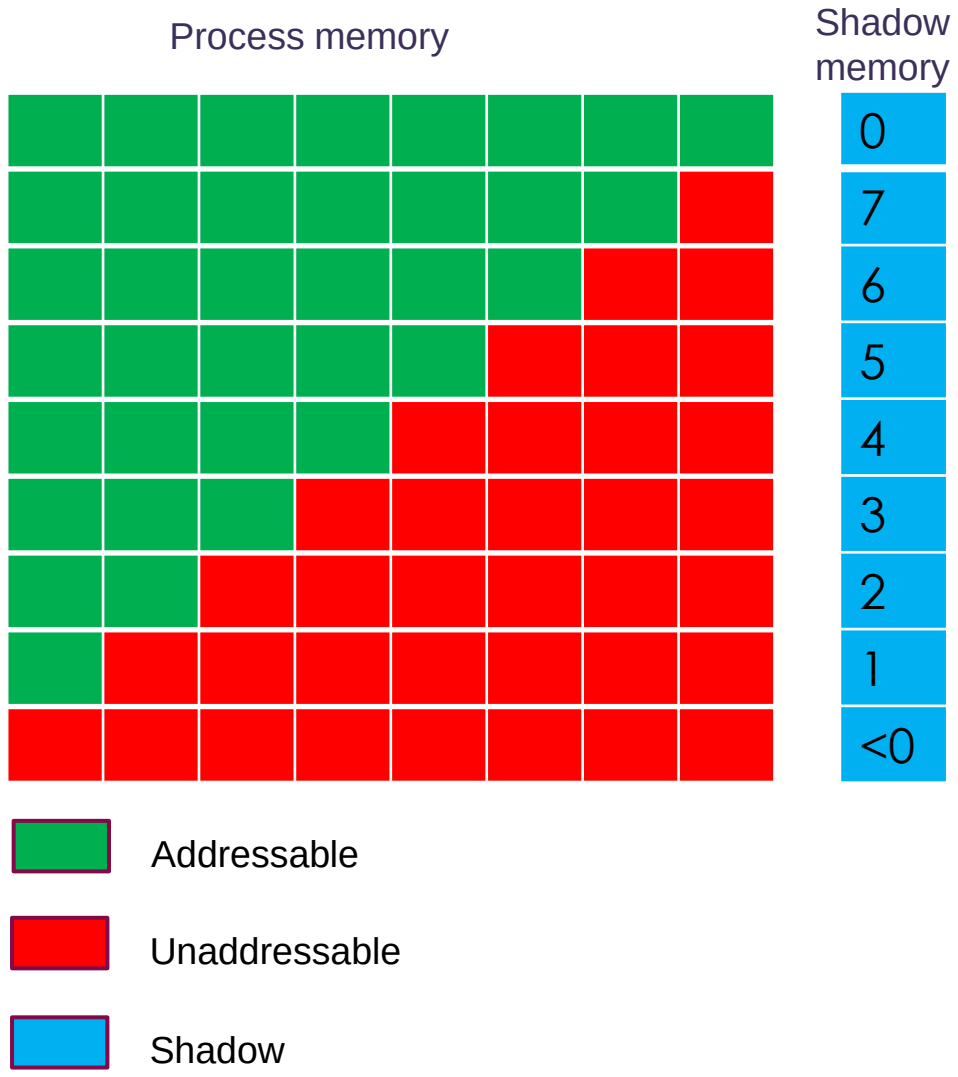
# Memory detect method

- Create poisoned **redzones** around memory allocation
- Instrument all loads/stores to check **shadow state** for each memory access
  - Build **lookup table**
- **Run-time** detect and **bookkeeping** for error messages

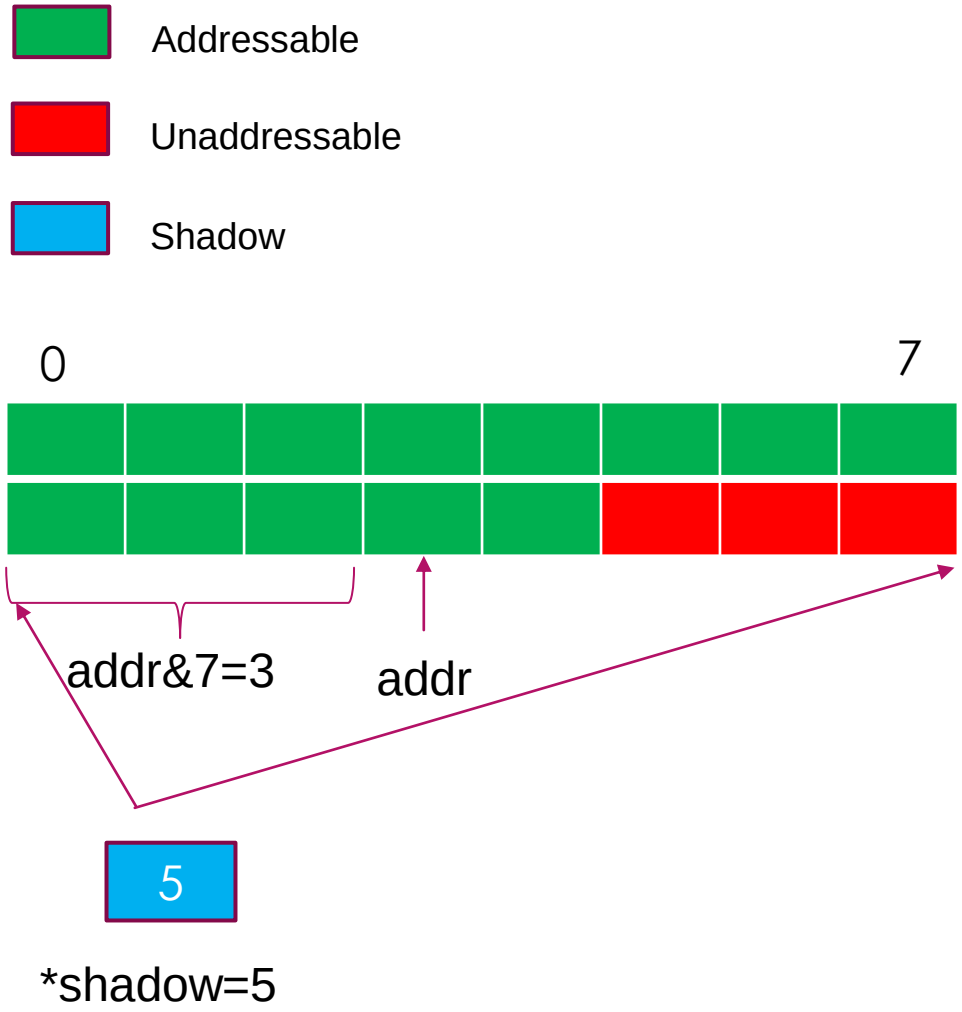


# Shadow byte

- Every aligned 8-byte word of memory has 9 states
- All of 8 bytes are addressable, shadow state is 0
- All of 8 bytes are unaddressable, shadow state is negative(<0), start with “ f ”, use “ fa ” for heap, “ f1 ” for stack…
- First N bytes are addressable, the rest 8-N bytes are not(1<=N<=7)
- ShadowAddr = (Addr >> 3) + Offset



# Instrumentation




❑ **8 bytes access**

```
long *addr = (long *)0xffff800012345678;

char *shadow = (char *)(((unsigned long)addr >> 3) +
SHADOW_OFFSET);
if (*shadow)
    report_bug();

*addr = 0;
```

❑ **1, 2, 4 bytes access**

```
//replace above if() as below, N=1,2,4
if (*shadow && *shadow < ((unsigned long)addr & 7) + N)
```

# Address Sanitizer overview

## Support

- LLVM 3.1 (GCC, Clang)
- Support Linux i386/x86_64, MacOS, Android, Windows…

## Detect

- Buffer overflows(heap, stack, global variable)
- Stack-use-after-return, heap-use-after-free
- more

# Run-time library


## Initialize shadow memory at startup

- Shadow mapping on Linux/x86_64 with SHADOW_OFFSET = 0x00007fff8000

## Provide full malloc/free replacement

- Insert poisoned redzones around allocated memory
- Quarantine for free-ed memory

## Collect stack traces for every malloc/free

## Bookkeeping for error messages

# Enable Asan in DPDK


## Why DPDK can’t use Asan directly?

- DPDK use different memory management and API from glibc
- Run-time library libasan.so can’t hook DPDK memory API
- Can’t set redzone and add to shadow mapping

# Quick enable Asan in DPDK

- **Provide malloc/free replacement**
  - Enable RTE_MALLOC_DEBUG, add trailer cookie(redzone) in heap_alloc() to help identify buffer overflows
  - Calculate redzone and shadow address, map redzone and shadow memory address

```
#define MALLOC_ELEM_TRAILER_ADDR(elem) (((void *)RTE_PTR_ADD(elem,  elem->size - MALLOC_ELEM_TRAILER_LEN)))
#define MEM_TO_SHADOW(mem) (((mem) >> 3) + 0x00007fff8000)
…
int64_t ptr = (uint64_t)MALLOC_ELEM_TRAILER_ADDR(elem);
char *shadow = (char *)MEM_TO_SHADOW(ptr);
```

  - Set redzone as unaddressable, “fa” as heap negative state.

```
set_redzone(shadow, 0xfa);
```

# Quick enable Asan in DPDK


- Add redzone in malloc_heap_free() for free

```
uint64_t ptr = (uint64_t)MALLOC_ELEM_TRAILER_ADDR(elem);
char *shadow = (char *)MEM_TO_SHADOW(ptr);
```

- **Support run-time library libdpdk_asan.so build in meson.build**

```
add_project_link_arguments('-ldpdk_asan', language: 'c')
dpdk_extra_ldflags += '-ldpdk_asan’
```

- **Compile code**
  - Compile with -Db_sanitize=address flag to enable ASan tool
  - Compile with -Dbuildtype=debug to enable gdb for more debug information

# Hello-world sample: heap-buffer-overflow


char *p = rte_zmalloc(NULL, 64 , 64);
p[65] = 'a';

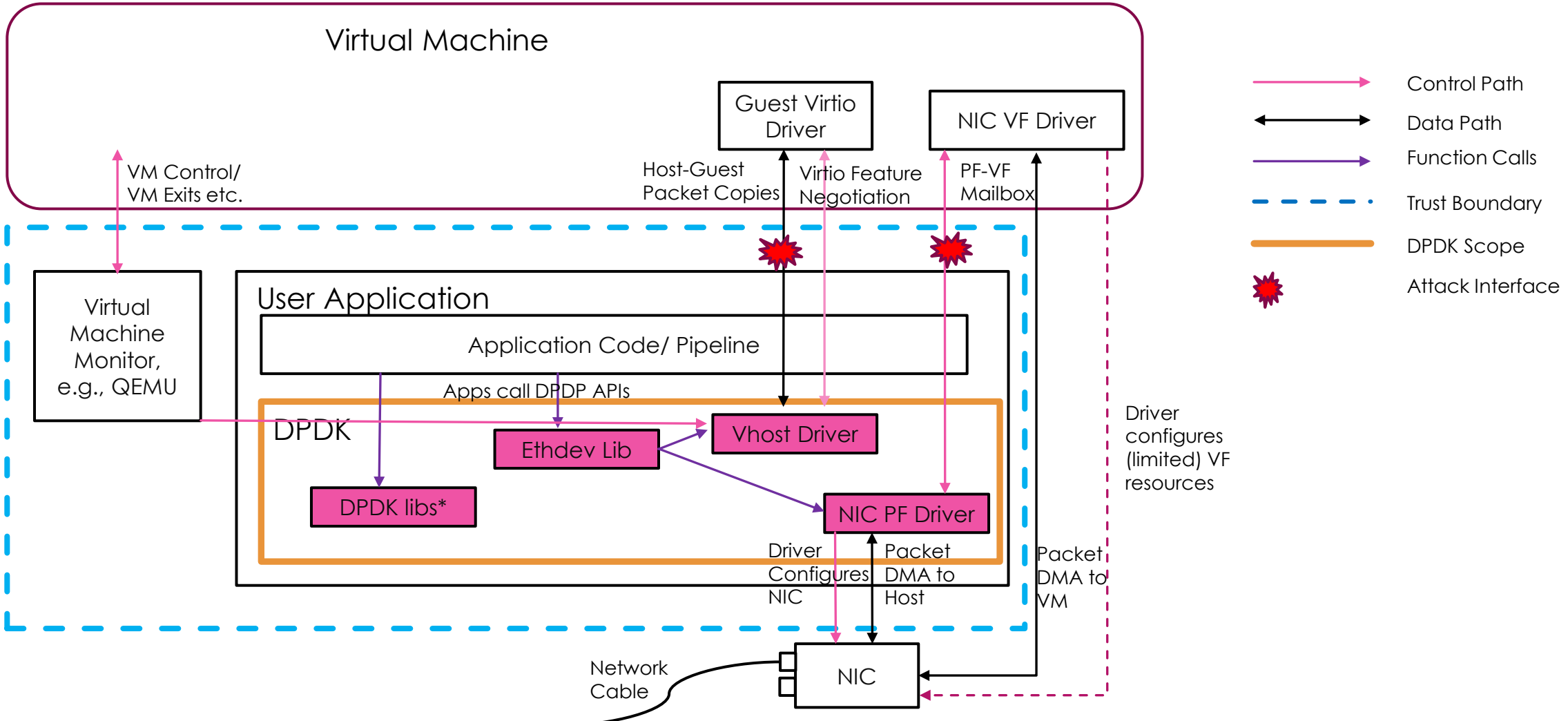
```
=================================================================
hello from core 3
==104064==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x7f6b3f9589c1 at pc 0x562e1c9d6578 bp 0x7ffc611f1b40 sp 0x7
ffc611f1b30
WRITE of size 1 at 0x7f6b3f9589c1 thread T0
    #0 0x562e1c9d6577 in main ../examples/helloworld/main.c:44
    #1 0x7f731d8abb96 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x21b96)
    #2 0x562e1c9d6369 in _start (/root/memory_debug/dpdk_asan311/x86_64-native-linuxapp-gcc/examples/dpdk-helloworld+0x6fa369)


SUMMARY: AddressSanitizer: heap-buffer-overflow ../examples/helloworld/main.c:44 in main
Shadow bytes around the buggy address:
  0x0fede7f230e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fede7f230f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fede7f23100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fede7f23110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fede7f23120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0fede7f23130: 00 00 00 00 00 00 00 00[fa]00 00 00 00 00 00 00
  0x0fede7f23140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fede7f23150: 00 00 00 00 00 00 00 00 fa 00 00 00 00 00 00 00
  0x0fede7f23160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fede7f23170: 00 00 00 00 00 00 00 00 fa 00 00 00 00 00 00 00
  0x0fede7f23180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
```

# Agenda


- Background
- Memory Detect Method
- Address Sanitizer Overview
- Enable Asan in DPDK
- **Why DPDK Fuzzing**
- **DPDK libFuzzer Deployment**
- **Summary & Next plan**

# Why DPDK Fuzzing

## DPDK Threat Modeling



# Why DPDK Fuzzing


- Memory related uncovered issues are typically vulnerabilities that can be exploited, e.g. **CVE-2020-14374***
- Identify **attack interface** memory related issues is vital important
- Fuzzing can help to finds bugs that are overlooked by regular scripted testing
- Fuzzing can utilize **detect tools** provided by compiler , e.g., Address Sanitizer

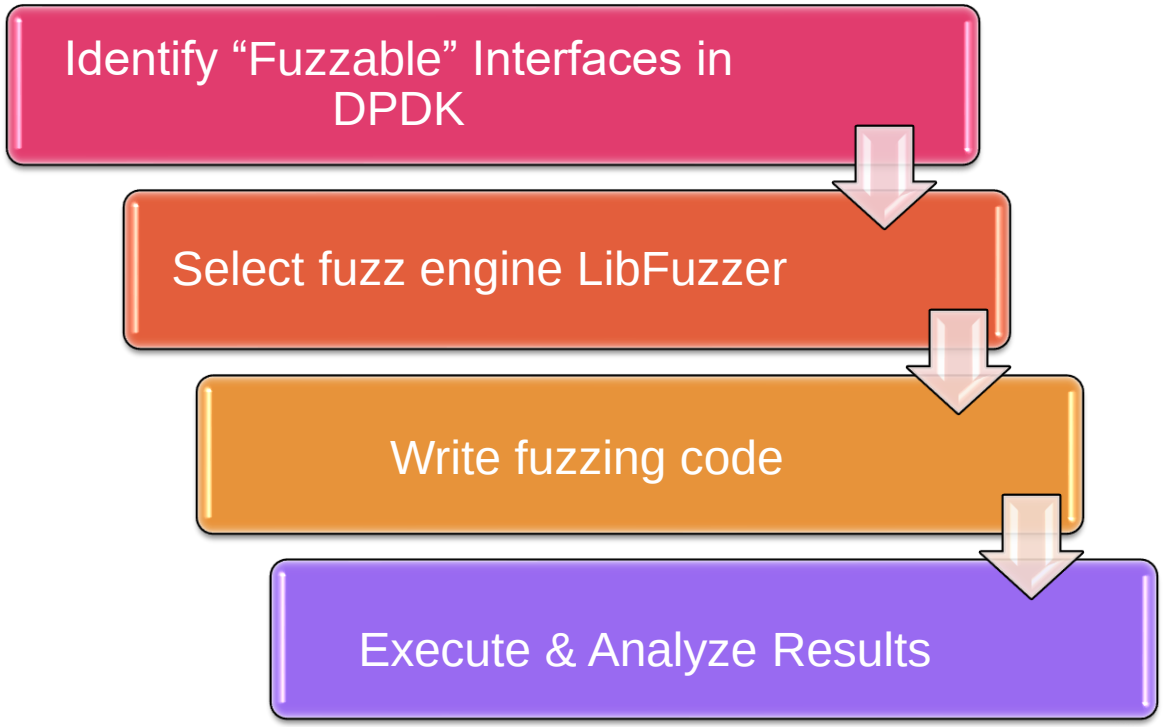
* Note: DPDK vhost多个安全漏洞通告-启明星辰 (venustech.com.cn)

# How DPDK Fuzzing

## Fuzzing Definition

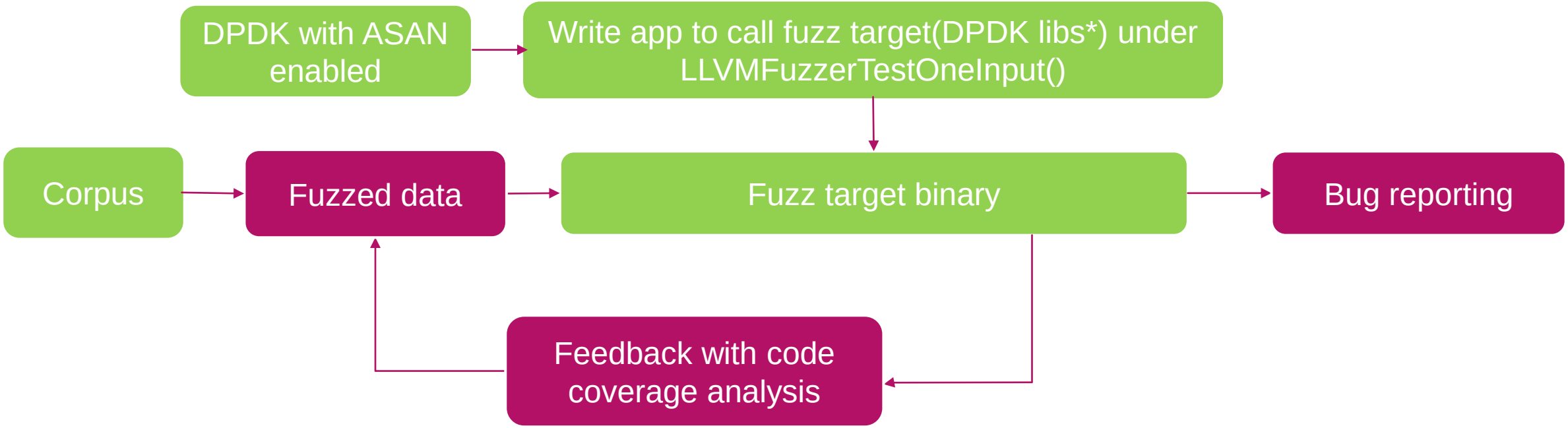
"Fuzzing is an automated **software testing technique** that involves providing invalid, unexpected, or **random** data as **inputs** to a computer program. The program is then **monitored** for exceptions such as **crashes, failing** built-in code **assertions**, or potential **memory leaks**"*

## DPDK Fuzzing Process




*https://en.wikipedia.org/wiki/Fuzzing

# How DPDK Fuzzing




Note: Need develop Libfuzzer framework

* Note: DPDK Libs can include any number of libraries included in DPDK. Generally, would include e.g., distributor, fragmentation/reassembly, GRO/GSO,ETC.

# DPDK libFuzzer Deployment

- Build DPDK with ASAN enabled (Introduced in first part)
- Write app feeds fuzzed data into fuzz target under LLVMFuzzerTestOneInput()

```
//dpdk_fuzz.c

int
LLVMFuzzerTestOneInput(uint8_t *data, size_t size)
{
        do_IP_fragment(data, size);

        return 0;
}
```

- Build fuzz target

  e.g., “cc=clang -g **-fsanitize=address,fuzzer** dpdk_fuzz.c -o dpdk_fuzz”

# DPDK libFuzzer Deployment


- DPDK initialize

```
root@dpdk-yinan-ntb1:/home/yinan/dpdk/dpdk-fuzzing-test-master/fuzzings/ip_fragmentation/build# ./ip_fragmentation /home/yinan/dpdk/corpus/
fuzzing> set arg -c/0x6/-n/4/-w/0000:03:00.0/--file-prefix=cy/----ip-type=ipv4
fuzzing> set arg -c/0x6/-n/4/-w/0000:03:00.0/--file-prefix=cy/--/--ip-type=ipv4
fuzzing> EAL: Detected 56 lcore(s)
EAL: Detected 2 NUMA nodes
EAL: WARNING! Base virtual address hint (0x100005000 != 0x7f61a6254000) not respected!
EAL:    This may cause issues with mapping memory into secondary processes
EAL: Multi-process socket /var/run/dpdk/cy/mp_socket
EAL: Selected IOVA mode 'VA'
EAL: Probing VFIO support...
EAL: VFIO support initialized
```

- Begin fuzz

```
optind: 1 optopt:63 optarg:(null) argc:2 argv[0]:./ip_fragmentation argv[1]:--ip-type=ipv4
INFO: Seed: 2243594100
INFO: Loaded 1 modules   (208 inline 8-bit counters): 208 [0x5b0e20, 0x5b0ef0),
INFO: Loaded 1 PC tables (208 PCs): 208 [0x572cf0,0x5739f0),
INFO:       99 files found in /home/yinan/dpdk/corpus/
INFO: -max_len is not provided; libFuzzer will not generate inputs larger than 4096 bytes
INFO: seed corpus: files: 99 min: 1b max: 1365b total: 12438b rss: 44Mb
#100    INITED cov: 32 ft: 37 corp: 6/2174b exec/s: 0 rss: 46Mb
#101    NEW    cov: 32 ft: 38 corp: 7/2341b lim: 1365 exec/s: 0 rss: 46Mb L: 167/1365 MS: 1 InsertRepeatedBytes-
#103    NEW    cov: 34 ft: 40 corp: 8/2420b lim: 1365 exec/s: 0 rss: 46Mb L: 79/1365 MS: 2 EraseBytes-EraseBytes-
#111    NEW    cov: 34 ft: 41 corp: 9/2612b lim: 1365 exec/s: 0 rss: 46Mb L: 192/1365 MS: 3 ShuffleBytes-InsertByte-CrossOver-
#120    NEW    cov: 36 ft: 43 corp: 10/2751b lim: 1365 exec/s: 0 rss: 46Mb L: 139/1365 MS: 4 ChangeBit-CMP-CrossOver-ChangeBinInt- DE: "\x02\x00\x00\x00\x00\x00\x00\x00"-
#122    NEW    cov: 37 ft: 44 corp: 11/2918b lim: 1365 exec/s: 0 rss: 46Mb L: 167/1365 MS: 2 ShuffleBytes-ChangeBinInt-
#130    NEW    cov: 38 ft: 45 corp: 12/3968b lim: 1365 exec/s: 0 rss: 46Mb L: 1050/1365 MS: 3 CrossOver-EraseBytes-EraseBytes-
#141    NEW    cov: 38 ft: 46 corp: 13/5253b lim: 1365 exec/s: 0 rss: 46Mb L: 1285/1365 MS: 1 CopyPart-
#294    REDUCE cov: 38 ft: 46 corp: 13/5238b lim: 1365 exec/s: 0 rss: 46Mb L: 1035/1365 MS: 3 EraseBytes-ChangeBit-InsertRepeatedBytes-
#404    REDUCE cov: 38 ft: 46 corp: 13/4268b lim: 1365 exec/s: 0 rss: 46Mb L: 315/1365 MS: 5 CopyPart-CrossOver-ChangeByte-ChangeASCIIInt-CrossOver-
#413    REDUCE cov: 38 ft: 46 corp: 13/3805b lim: 1365 exec/s: 0 rss: 46Mb L: 572/1365 MS: 4 InsertByte-InsertByte-PersAutoDict-EraseBytes- DE: "\x02\x00\x00\x00\x00\x00\x00\x00"-
```


# Thanks

xueqin.lin@intel.com &  yinan.wang@intel.com