MH-QEMU: MEMORY-STATE-AWARE FAULT INJECTION PLATFORM

Hideyuki JITSUMOTO*1, Yuya KOBAYASHI*3,
Akihiro NOMURA*1, Satoshi MATSUOKA*1*2

*1 Tokyo Institute of Technology, *2 RIKEN R-CCS,
*3 Degital Media Professionals Inc.

BACKGROUND AND MOTIVATIONS

OPTIMIZATION OF SDC TOLERANCE

- Detecting Silent Data Corruption(SDC)
  - Replication and comparison
    - Additional computing resource
  - Application Based Verification
    - Additional verification calculations depend on each application

Optimization of cost and performance of SDC tolerance

- Fault Injector
  - Verify OS's and applications' resiliency toward SDC such as where should be protect on software
  - Verify performance of new SDC tolerance algorithms

CAUSE OF DATA CORRUPTION

- Mainly random single bit error happened by radiations

- Error depends on memory module implementation
  - Disturbance Error on DRAM
    - Data corruption on cell which is the neighbor from manipulated cell.
  - Deterioration on NVRAM
    - The Cell become unreliable after a limited number of erase cycles.
- Error mechanism is not clear on next-gen. memory module and new memory usage algorithm
  - Hierarchical usage of different memory architecture
  - 3D structured memory

Fault injector needs to consider about hardware specific error

Current fault injector supported random single bit error
  - Other error corrected and detected by ECC -> new ECC, new device

MH-QEMU: Memory-State-Aware(MSA) Fault Injection Platform

- Hardware specific fault injection platform which can make error depends on memory access pattern and memory state:
  - emulates various hardware specific error by using memory access pattern and memory state information
  - intercepts memory access and execute user defined code:
    - logging memory access pattern
    - injecting error when error occasion condition is satisfied
  - supporting tools for making flexible fault injection scenario and analysis of the effects of fault

SDC Injection Method

- Injection to physical hardware (neutron/heavy ion beam)
  - Difficult to pinpoint SDC bits and timings
  - Damage to hardware
- Injection by program modification (code snippet, LLVM)
  - Difficult to emulate hardware specific fault
- Injection by Virtual Machine
  - treats memory modules on guest machine as the process memory on host machine
    → our work also use this method

RELATED WORK

- F-SEFI [Guen et al.]
  - VM-based fault injector
  - can intercept machine language instruction
  - Inject error to CPU logical circuit, registers, memory modules

Difference: MH-QEMU has supporting tool for injecting fault to memory module flexibly

- Real-time mapping an address used by a process on VM to physical address and the reverse
- Inject error only to memory module (currently)

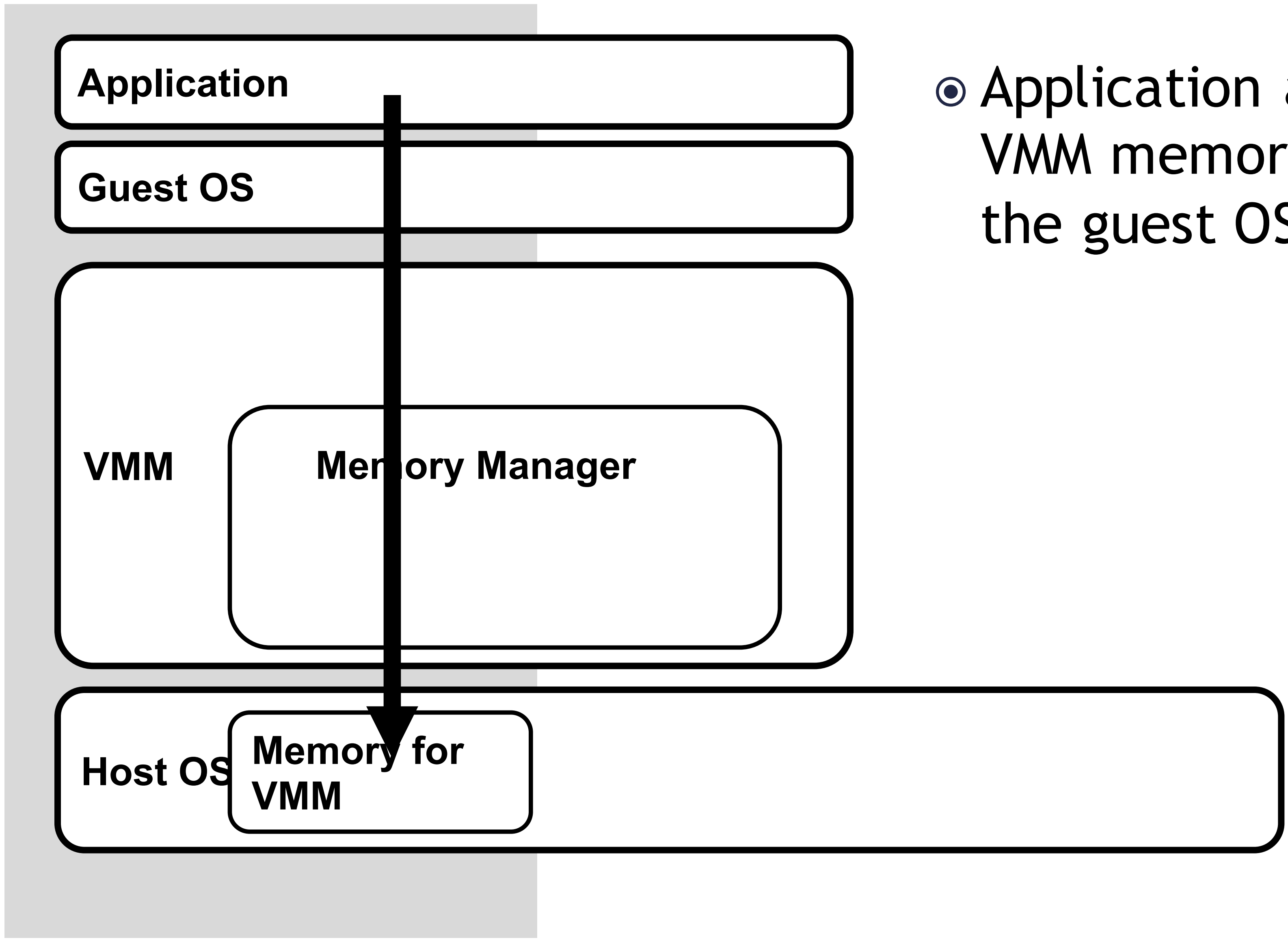
(MH-QEMU may implements on F-SEFI…)

DESIGN

REQUIREMENTS OF MSA FAULT INJECTOR (AND MODULES)

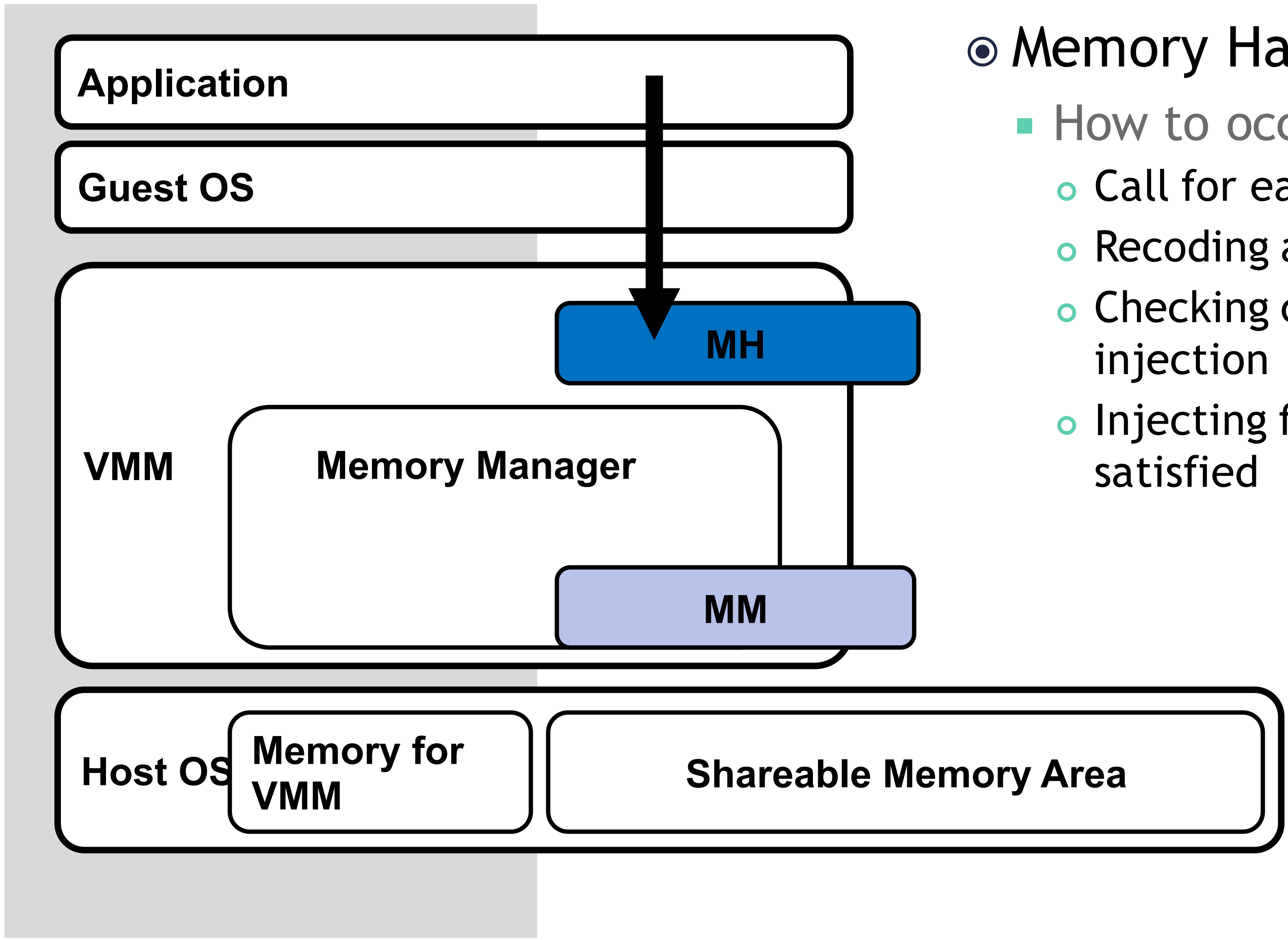
- no damage to physical hardware
  - VM-based
- emulating MSA faults flexibly without any effects into VM
  - Injection of faults from host OS to VM (MM:memory mapper)
  - Definition of memory access handler for each instruction for memory state modification (MH: memory handler)
  - Scheduling fault injection (FS: fault injection scheduler)
- supporting tools for injecting fault flexibly and analysis of the effects of fault
  - APIs for getting memory usage information of VM from host OS (ADM: application-data mapper)

Basic Scenario of MH-QEMU: Normal Memory Access



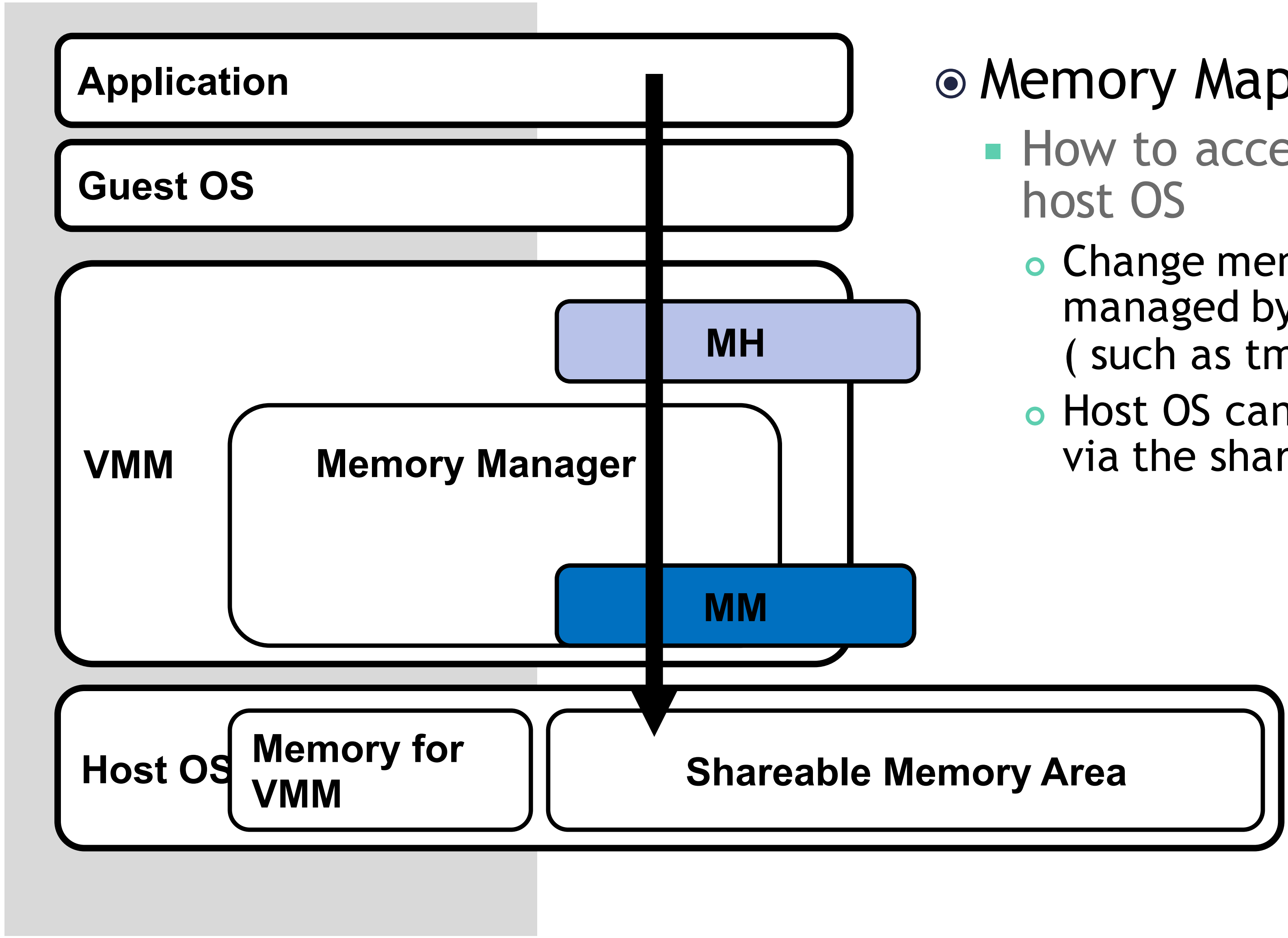
- Application accesses to the VMM memory on host OS via the guest OS

BASIC SCENARIO OF MH-QEMU: ACCESS PATTERN AND STATE INFORMATION



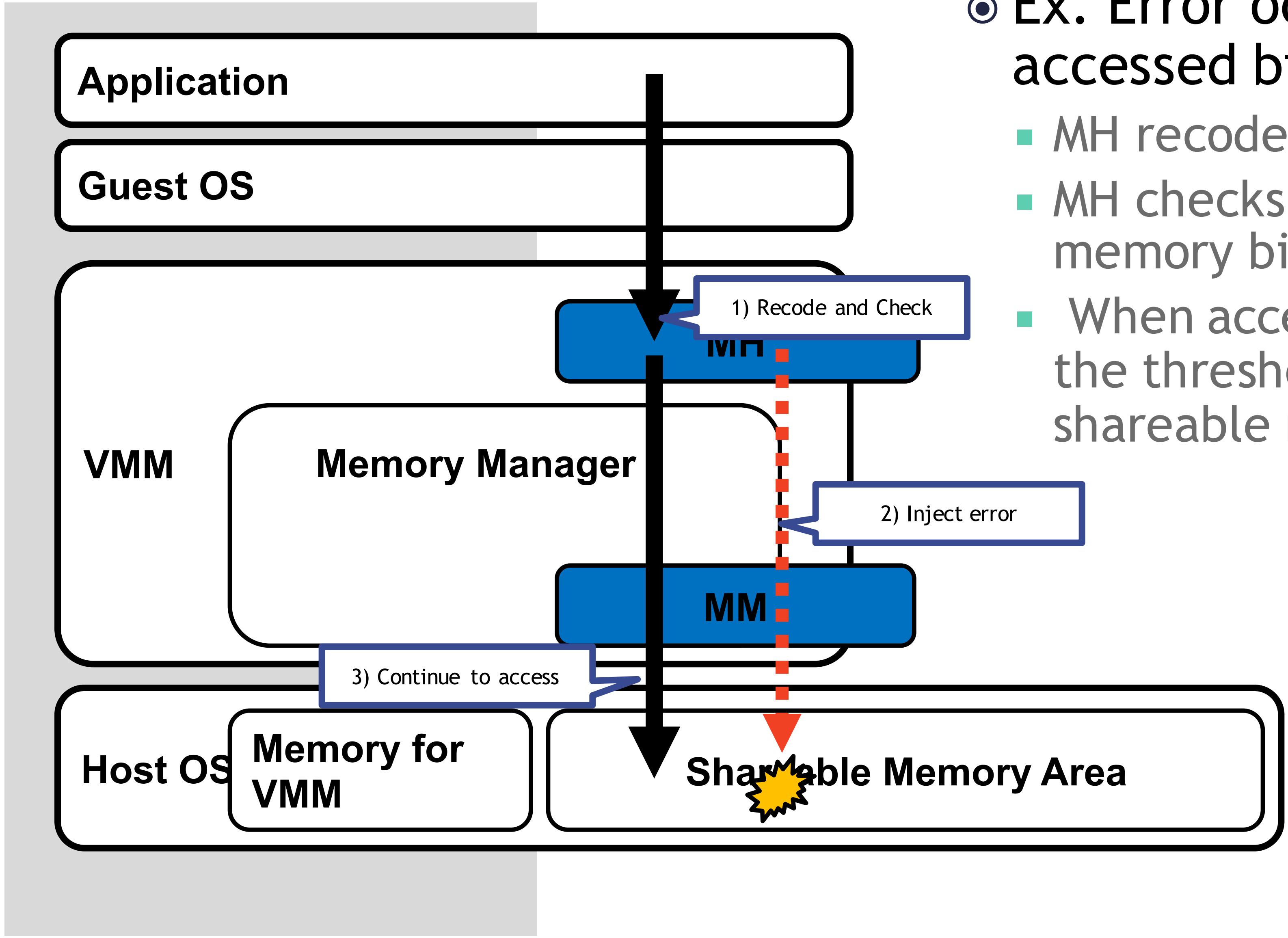
- Memory Handler (MH)
  - How to occur errors
    - Call for each memory access
    - Recoding access pattern
    - Checking condition for fault injection
    - Injecting fault if the condition is satisfied

BASIC SCENARIO OF MH-QEMU: REMAPPING VMM MEMORY TO SHARABLE AREA



- Memory Mapper (MM)
  - How to access VM memory from host OS
    - Change memory area from the area managed by VMM to shareable area (such as tmpfs)
    - Host OS can operate VM’s memory via the shareable area

BASIC SCENARIO OF MH-QEMU: FAULT INJECTION



- Ex. Error occurs frequent accessed bit
  - MH recodes access count
  - MH checks access count for each memory bit
  - When access count become over the threshold, MH modifies shareable memory area via MM

SUPPORTING TOOLS

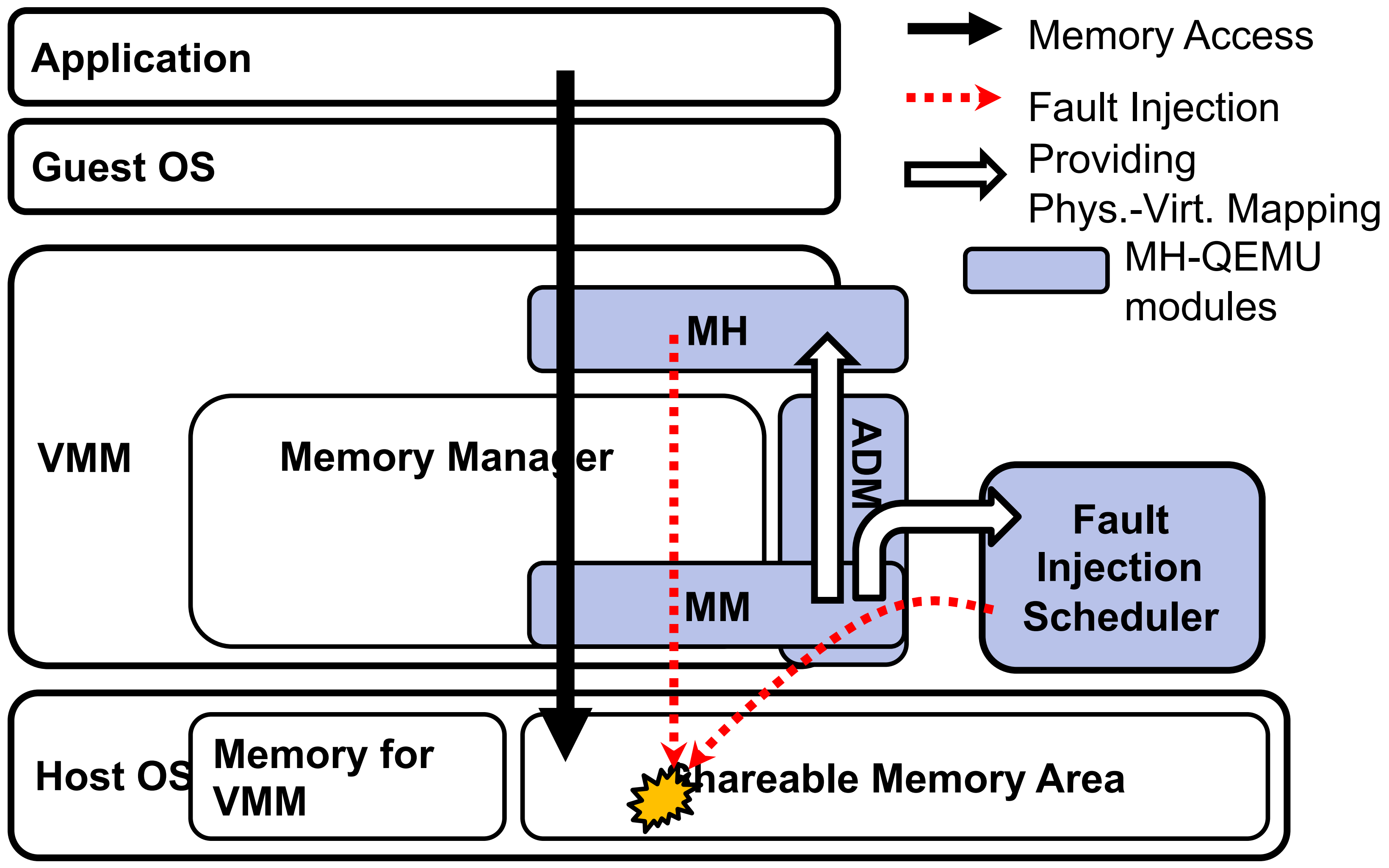
- Application-Data Mapper (ADM)

  converts application name to physical address and the reverse
  - MH-QEMU can inject error to specified application on VM
  - MH-QEMU can get application name from error-injected address
  - ADM also supply VMM's mapping information (physical-virtual)

- Fault Injection Scheduler (FS)

  executes time-based events
  enables/disables other MH-QEMU modules
  - MH is high cost -> needs to apply on appropriate timing
  - Time-based error injection

OVERVIEW OF MH-QEMU

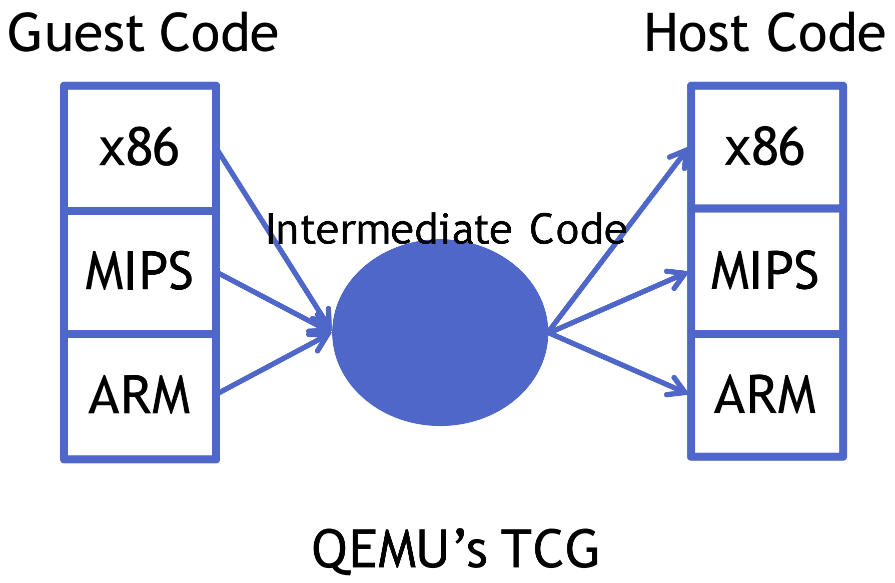


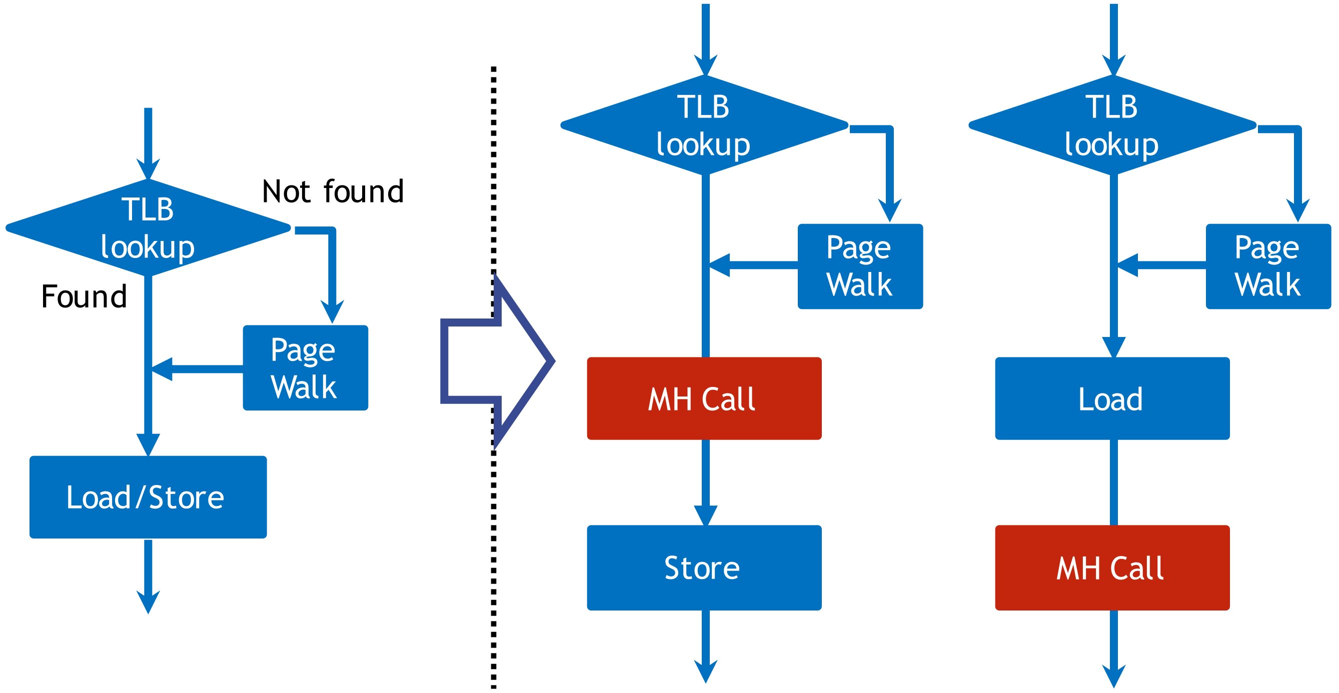
MH-QEMU-based Access

IMPLEMENTATION (MH & ADM)

MH: Interception of Memory Access

- Modified Tiny Code Generator on QEMU
  - TCG = CPU virtualization module
  - Insert MH calling code to load and store instruction

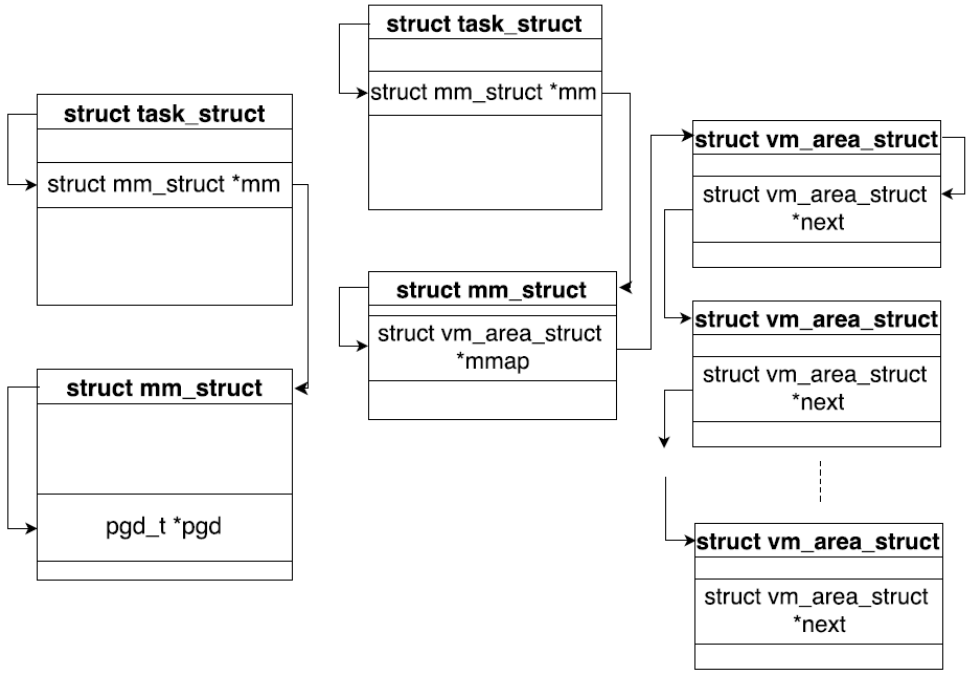




ADM: Mapping App. and Phy. Address

- OS information stored VM memory (Limitation: Linux)
  - can get via MM without interacting guest OS
- Symbol table of kernel

Required OS Information
- Page table
- Process information
  → managed by list structure



<u>All information can get from
kernel symbol by following the list connection</u>

EXAMPLE OF MH IMPLEMENTATION

```
memory_access_handler(physaddr, virtaddr){
 range ← ADM_get_heep_addr(target_name)
 if (virtaddr is in range){
  count[virtaddr]++
 }
 for(addr ← each range){
  if (count[addr] >= threshold){
   records addr
   MM_flipbit(addr)
   ADM_write_processinfo(target_name)
   FS_turnoffMe()
  }
 }
}
```

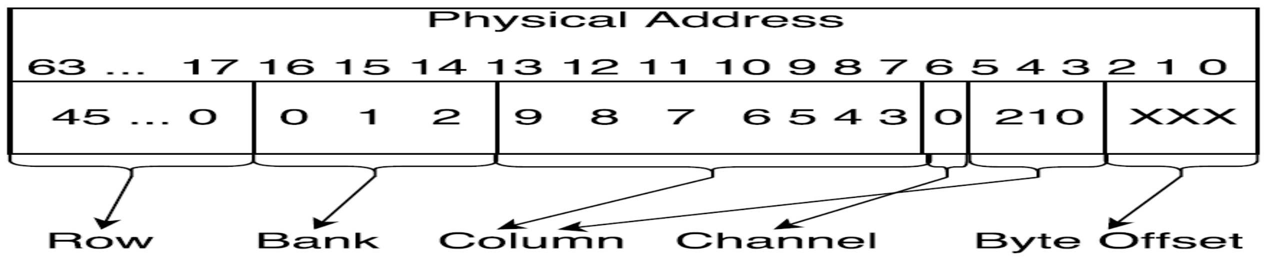
- Injecting error on frequent accessed bit on specific application
  - gets application address range
  - counts memory access if accessed bit is in the range
  - checks access count
  - recodes the overrun address for post-analysis
  - flips the bit on the overrun address
  - Recode the process information for post-analysis
  - Turn off MH for performance

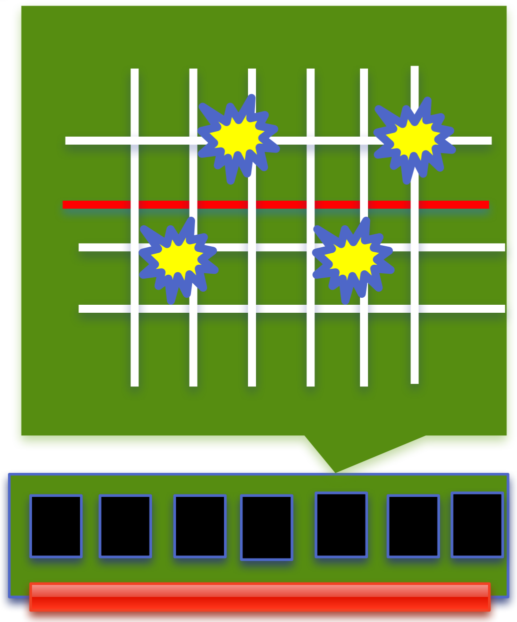
HOST SERVER	
CPU	2* Intel X5650 (2.67GHz, 6core/12thread) VT-x
Memory	ECC DDR4 SDRAM 48GB
OS	CentOS 7.1 (Kernel 3.10.0)
VM SERVER (8VM/HOST)	
CPU	x86_64 Architecture
Memory	512MB
OS	Scientific Linux 7.4 (Kernel 3.10.0)

USECASE

USECASE: EVALUATION OF NPB CG RESILIENCE

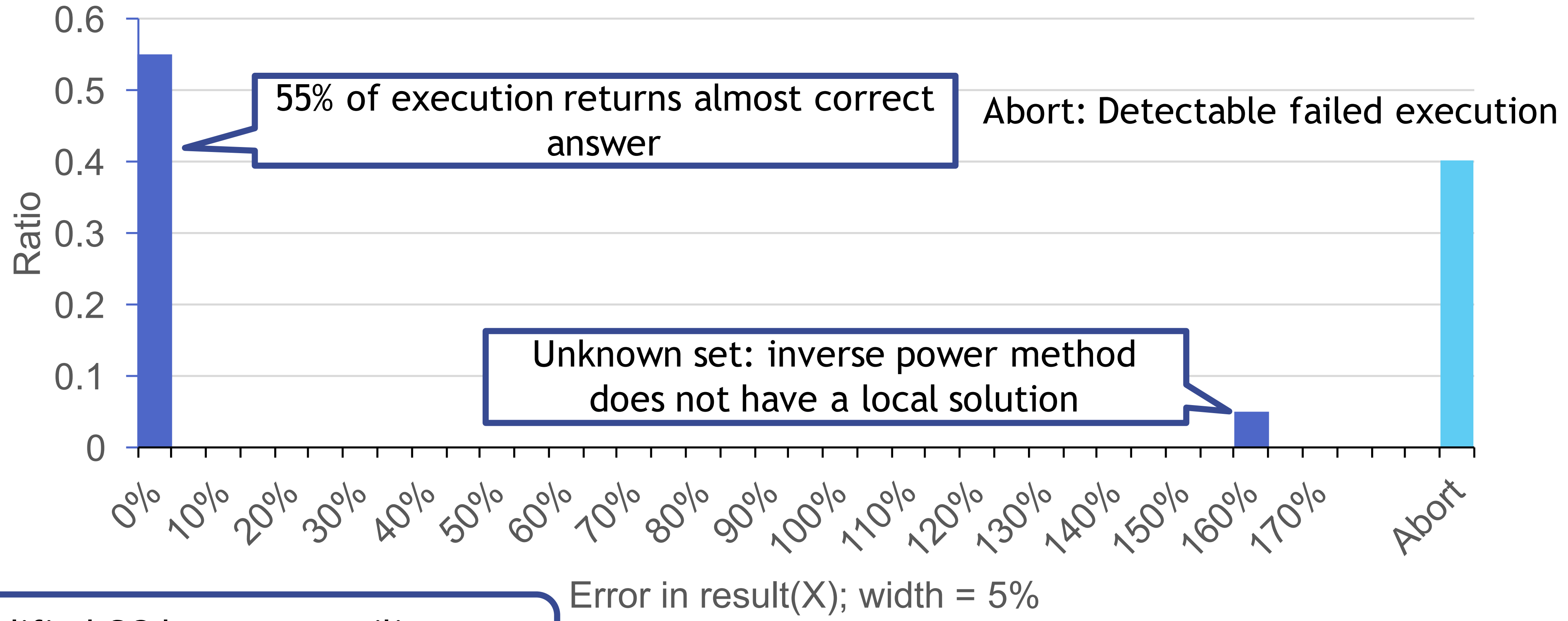
- Injecting Row-hammer fault to Modified NPB CG
  - Row-hammer fault
    - By frequent access to a specific memory row, surrounding rows get data corruption
    - Access count threshold = 1000, Error occasion rate = 5 x 10^{-10}
  - Modified NPB CG
    - CG may have tolerance toward DC by iterative method
    - Original NPB CG has constant loop -> continue the loop until it converges
  - Address Mapping rule: Intel 82955X-MCH





DISTRIBUTION OF COMPUTATION ERROR

- 2443 runs



55% of execution returns almost correct answer

Abort: Detectable failed execution

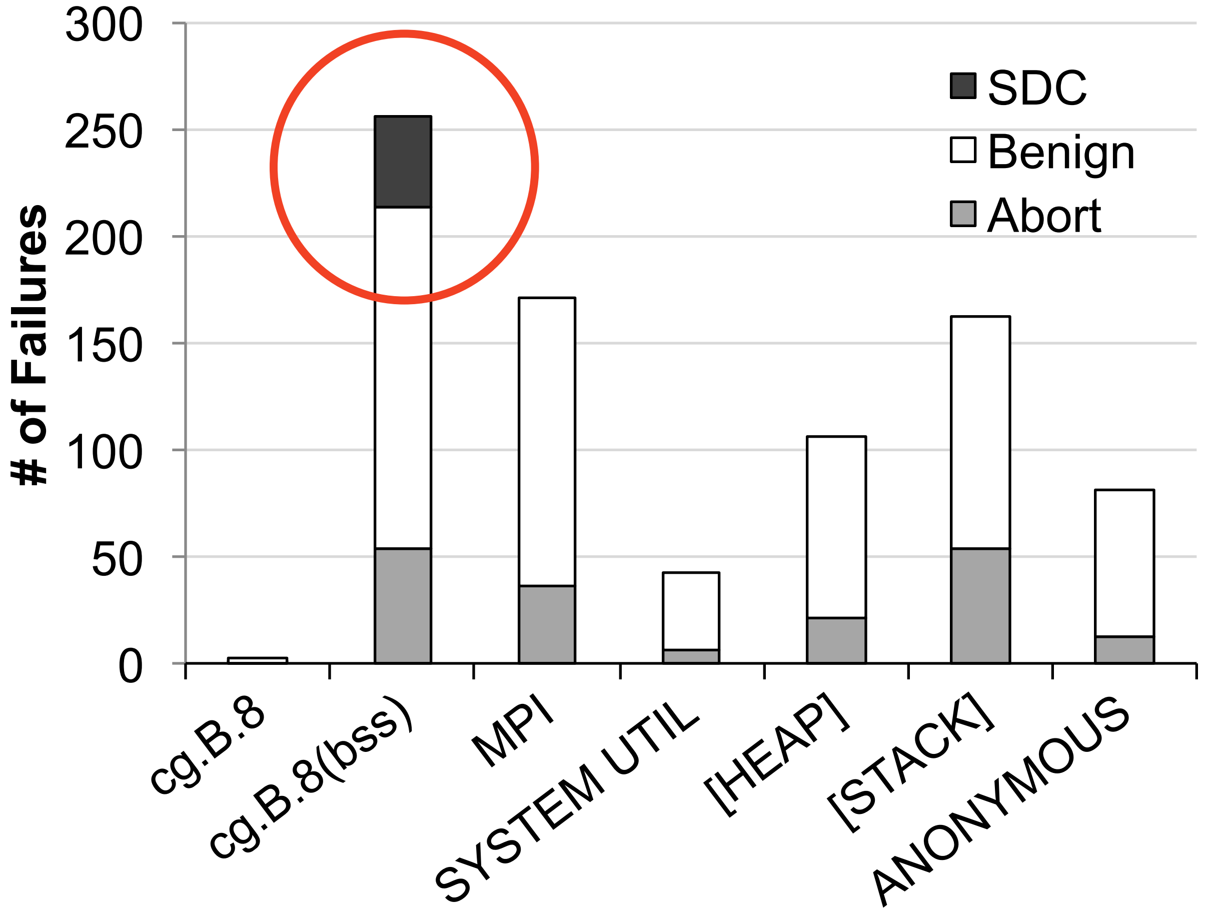
Unknown set: inverse power method does not have a local solution

Modified CG has some resiliency toward row-hammer error

RELATIONSHIP OF FAULT AND PROCESS MEMORY REGION

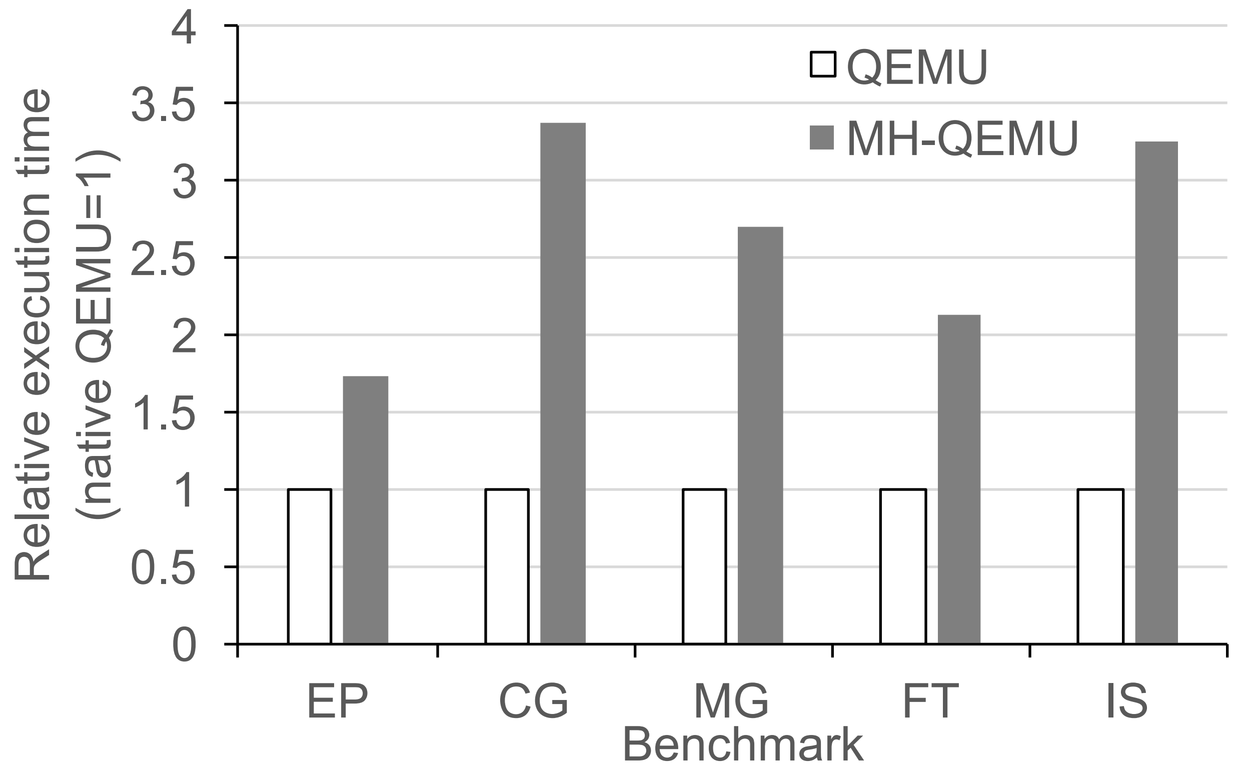
- Pickup 825 runs of the previous evaluation
- Only protect BSS region
  - Almost of data set to BSS…

Specify protection area without knowledge of CG algorithm



OVERHEAD OF MH-QEMU (REFERENCE)

- NPB 3.3.1 class B with 8 processes
- Exec. time comparison between QEMU and MH-QEMU
  - MM overhead is negligible
  - Showing MH overhead
- Blank MH (calling only)



CONCLUSION

- Develop MH-QEMU: Memory-state-aware fault injection platform
  - For evaluate resiliency toward hardware specific error and effects of error on next-gen. hardware
  - Easily implement flexible fault injection scenario on memory module
  - Demonstrate resiliency evaluation by modified NPB CG that has iterative calculation

FUTURE WORK

- Use appropriate application for demonstration
  - NPB CG is not suit -> HPCG ?
- Reducing execution cost
  - KVM and instruction level code insertion ? (dyinst, Intel PIN)
- Supporting other hardware
  - CPU circuit/register
    - Implement on F-SEFI ?
- Evaluating many applications about resiliency
  - optimizing SDC detection cost