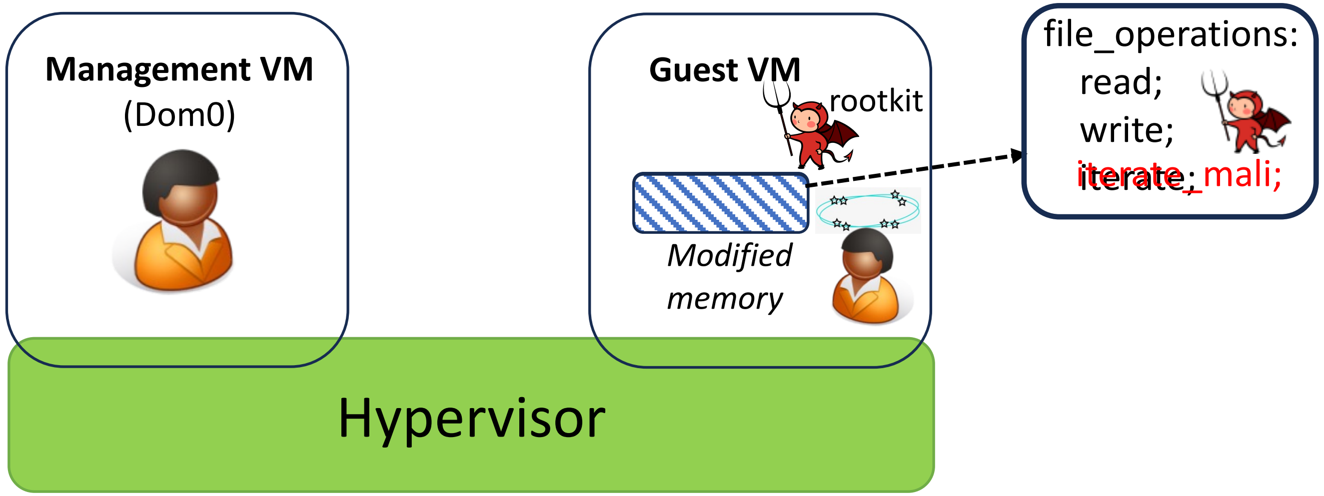
Remote Direct Memory Introspection

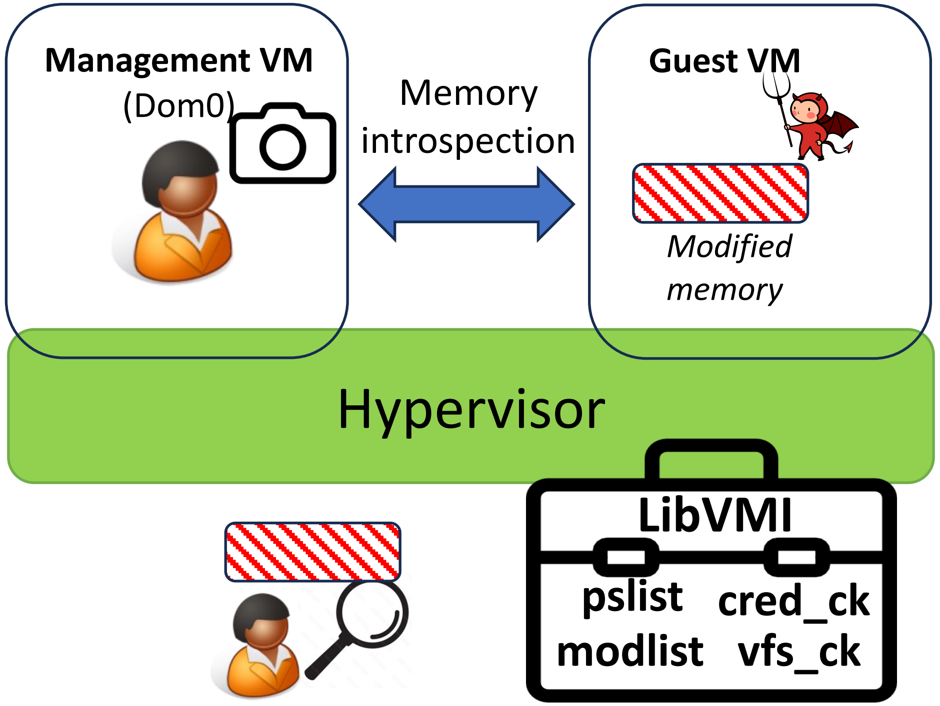
Hongyi Liu, Jiarong Xing, Yibo Huang, Danyang Zhuo, Srinivas Devadas, Ang Chen

Problem: Memory introspection



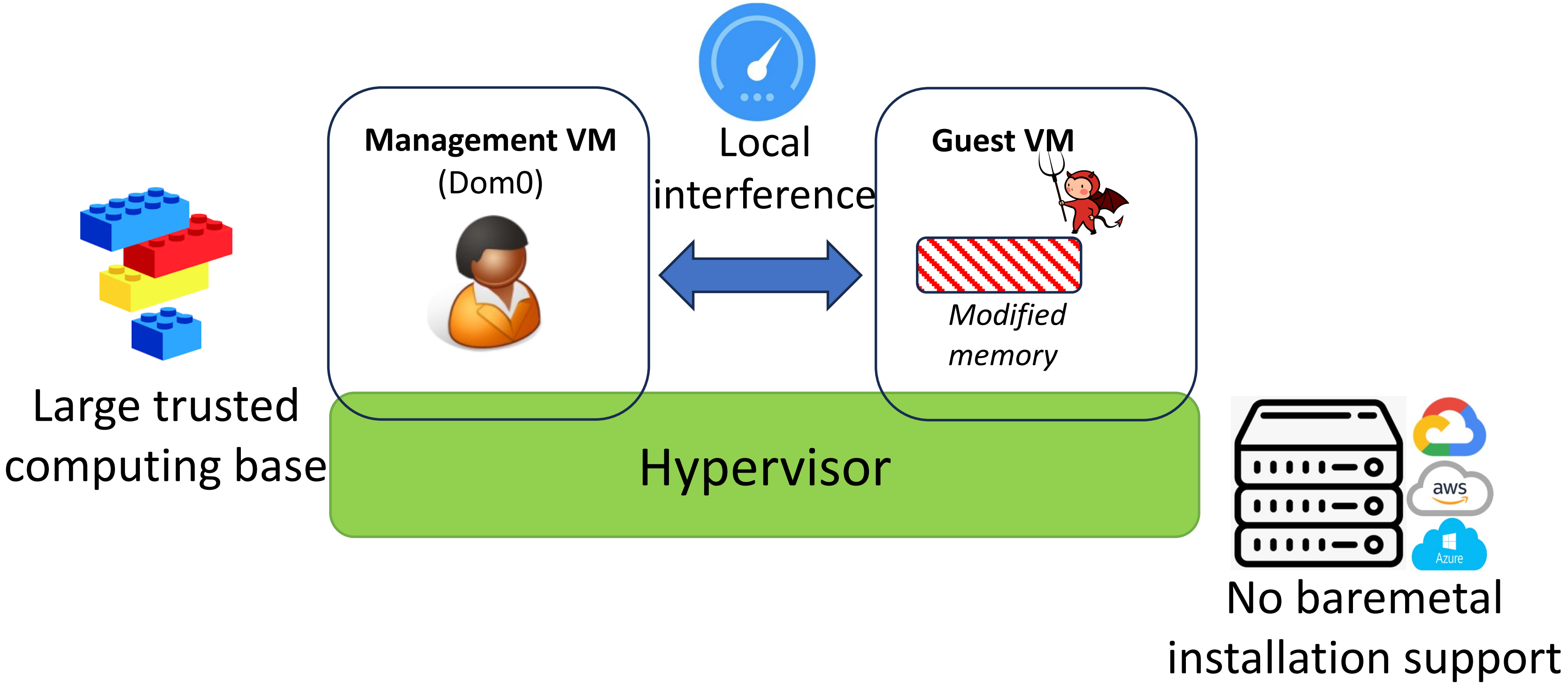
- Memory introspection is a critical security task

Problem: Memory introspection



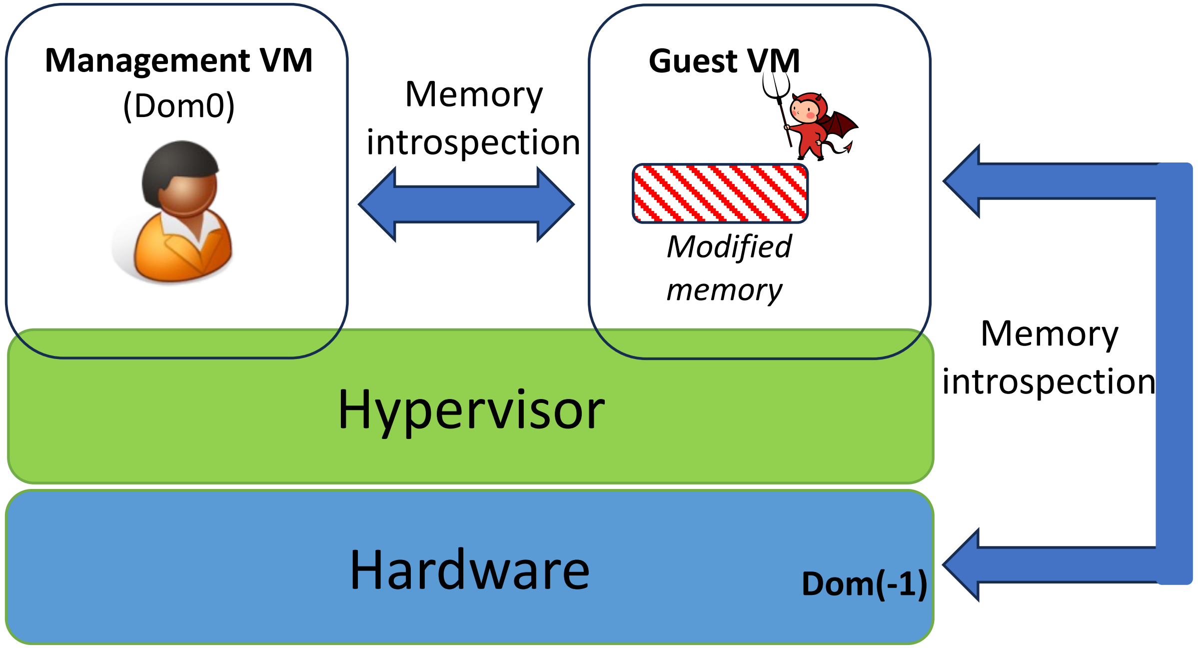
- Memory introspection is a critical security task
  - It can detect kernel-level attackers (i.e., rootkits)
  - Agent snapshots raw memory for further forensics
  - Hypervisor-based introspection is widely used
    - E.g., Livewire (NDSS'03), ImEE (SEC'17), LibVMI

Limitations of hypervisor-based introspection



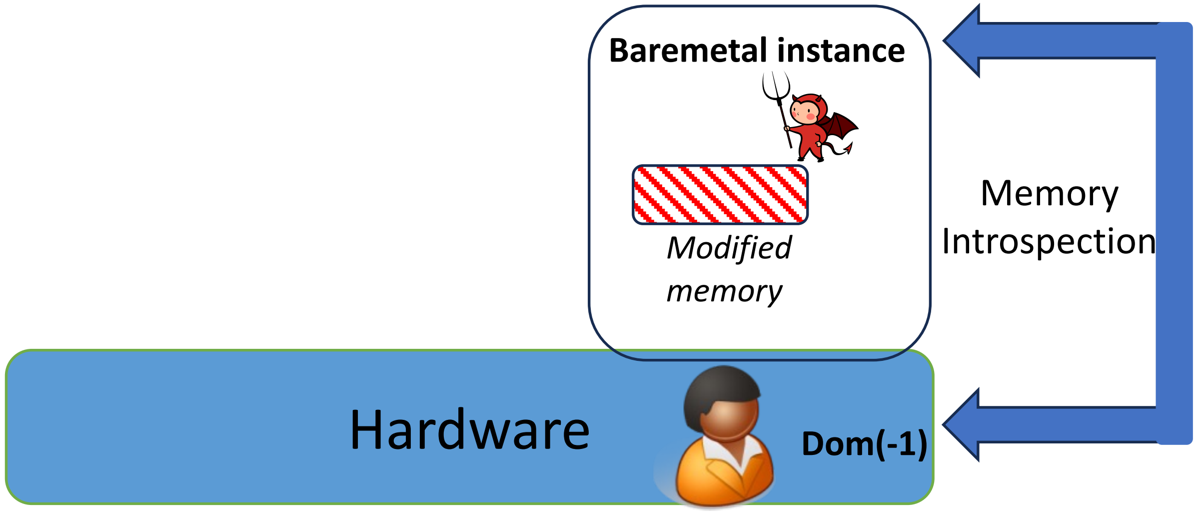
- Hypervisor-based introspection has inherent limitations
  - It causes performance interference with local workloads
  - It contains a large trusted computing base inducing vulnerabilities
  - It is not capable to support baremetal installations

Insight: Dom(-1) security offloading



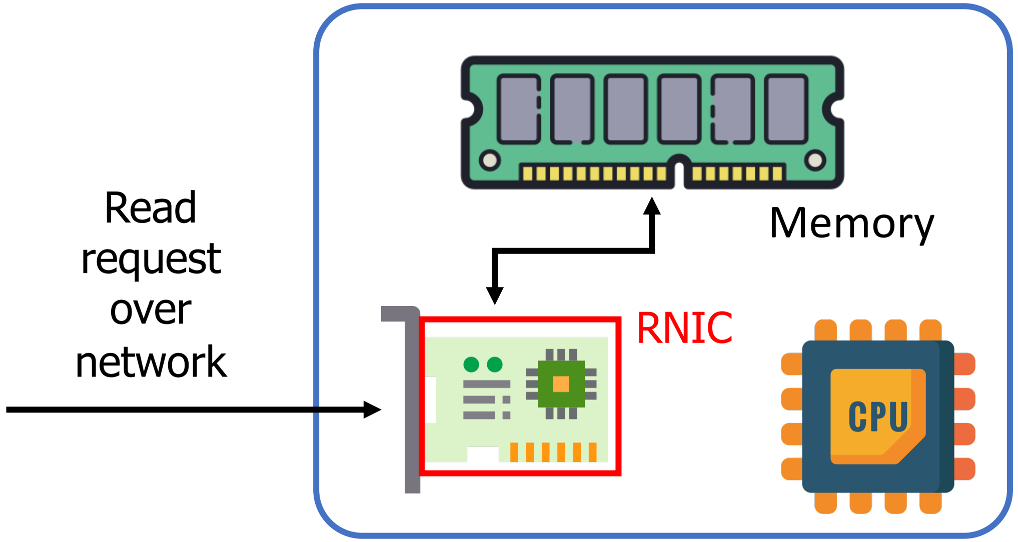
- Can we solve the problem by moving one layer below?
  - Dom(-1) security is enforced in widely-deployed hardware
  - Dom(-1) substrate enforces efficient security execution

Insight: Dom(-1) security offloading



- Can we solve the problem by moving one layer below?
  - Dom(-1) security enforced in widely-deployed hardware
  - Dom(-1) substrate enforces efficient execution
  - Dom(-1) security supports baremetal installation

Opportunities: Remote Direct Memory Access



RDMA

- RDMA enables reading/writing remote memory with CPU bypassed
  - RNIC (RDMA NIC) can perform DMA to remote memory over network
  - RDMA has been widely deployed in cloud datacenters

★ RDMA can serve as memory datapath for Dom(-1) introspection

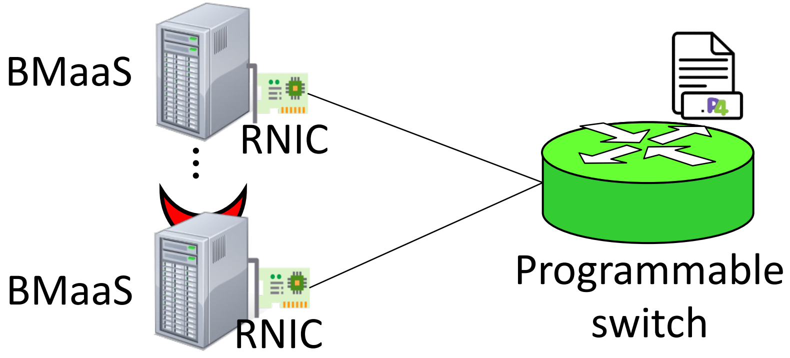
Opportunities: Programmable switches


```
table NxtPC_tab {
	key = {
		rdma.qpn:	exact;
		meta.pred:	exact;
	}
	actions = {compute_NxtPC;}
}
```

- Programmed with high-level language, e.g., P4.
  - Parse RDMA headers, enforce match-action tables and stateful operations
- Run at line speed (Tbps) and are commercially available
- Have been widely used for network security
  - E.g., PortCatcher-CCS'22, IMAP-NSDI'22, Bedrock-Security'22
  - This work is the first to use programmable switches for kernel security

Programmable switches can efficiently enforce control logics

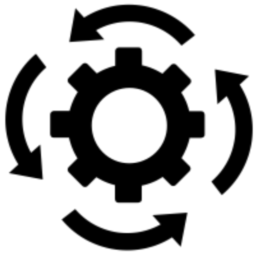
RDMI: Remote Direct Memory Introspection



RDMI execution model

DSL queries

AIM instructions



RDMI runtime

RDMI components

- RDMI: A new paradigm for memory introspection
  - DSL: Introspection abstractions hiding low-level programming details
  - AIM: Instruction set for better resource sharing and live deployment
  - Runtime: Reconfigurable engines to instantiate AIM instructions

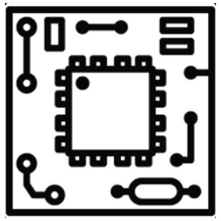
RDMI benefits

Baremetal support

Remote execution

Efficient introspection

COTS deployment



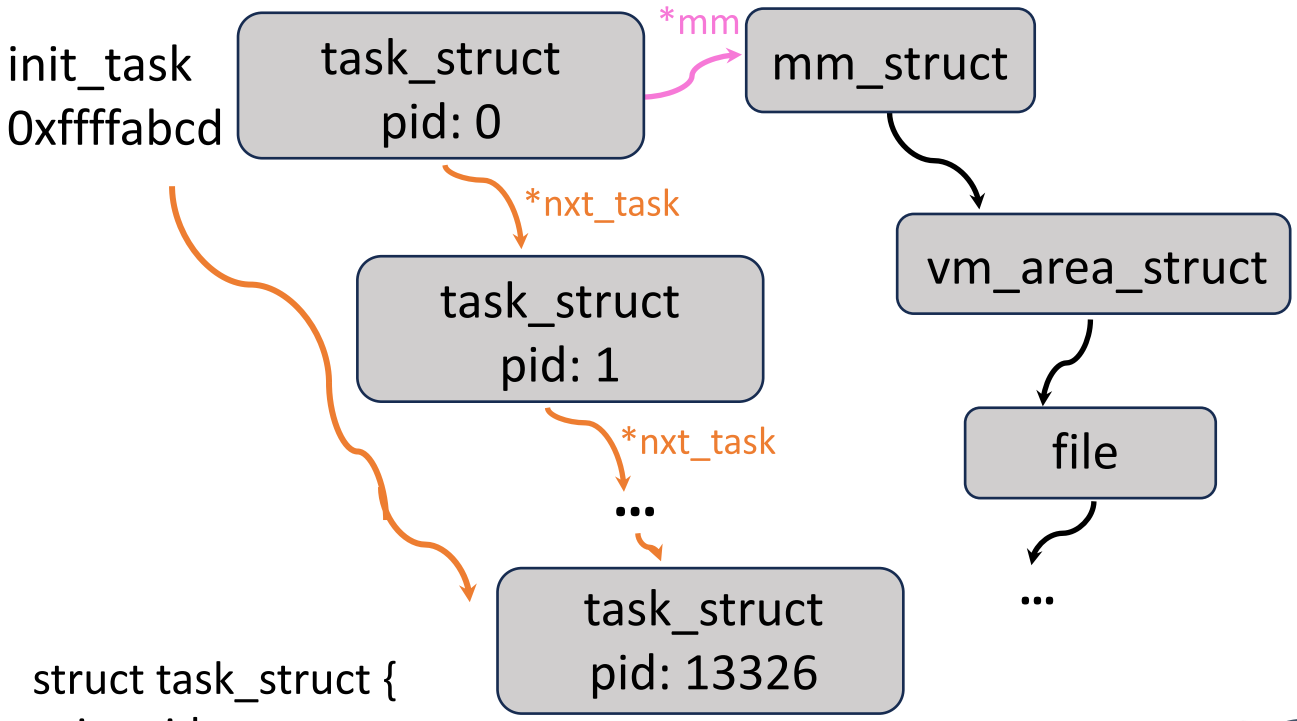
Programmable DSL policies

- RDMI offers protections with new benefits

Outline

- Motivation: Better memory introspection
- Opportunity: Dom(-1) security execution
- Approach: Remote Direct Memory Introspection
- RDMI design:
  - Design #1: Introspection DSL design
  - Design #2: Abstract introspection machine
  - Design #3: Reconfigurable introspection engines
- Evaluation
- Conclusion

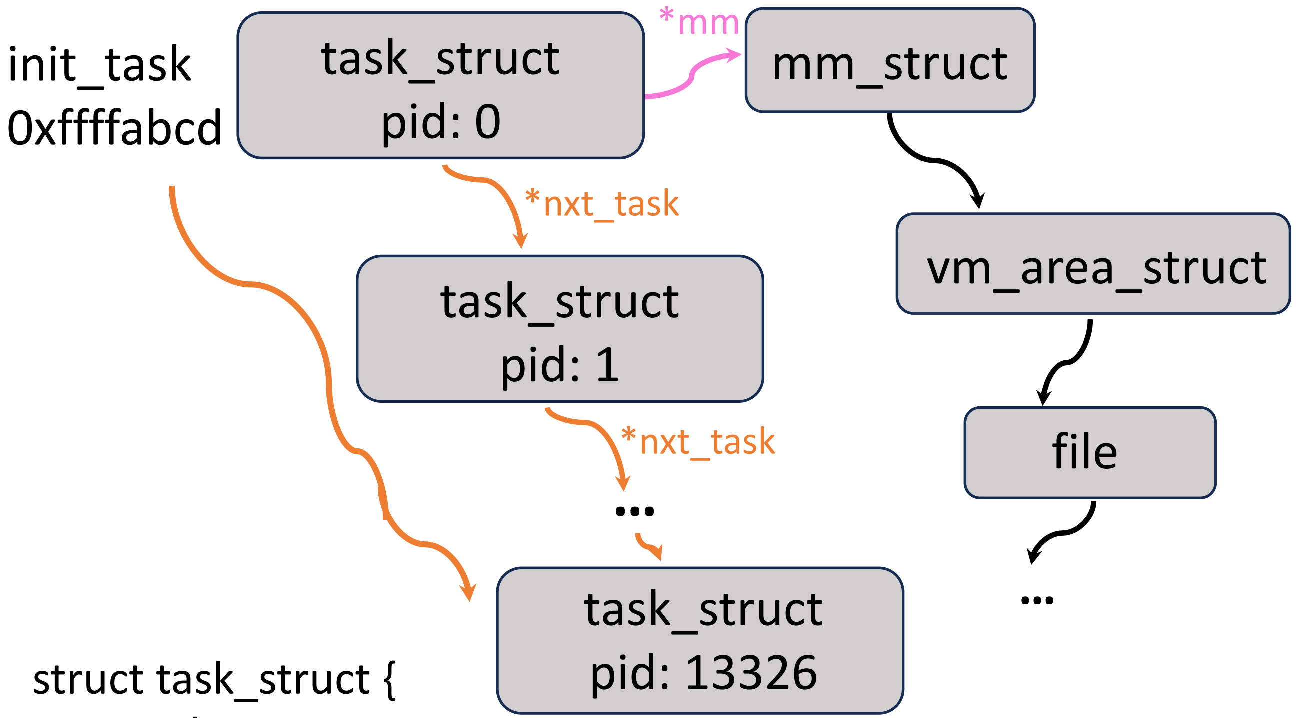
Introspection is a "graph processing" task




```
struct task_struct {
    int pid;
    struct task_struct *nxt_task;
    struct mm_struct *mm;
};
```

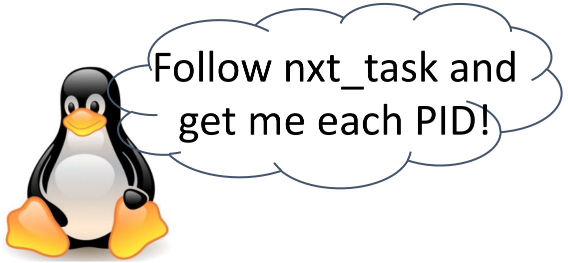
- Memory introspection shares similar execution model with graph processing.

Introspection is a “graph processing” task




```
struct task_struct {
    int pid;
    struct task_struct *nxt_task;
    struct mm_struct *mm;
};
```

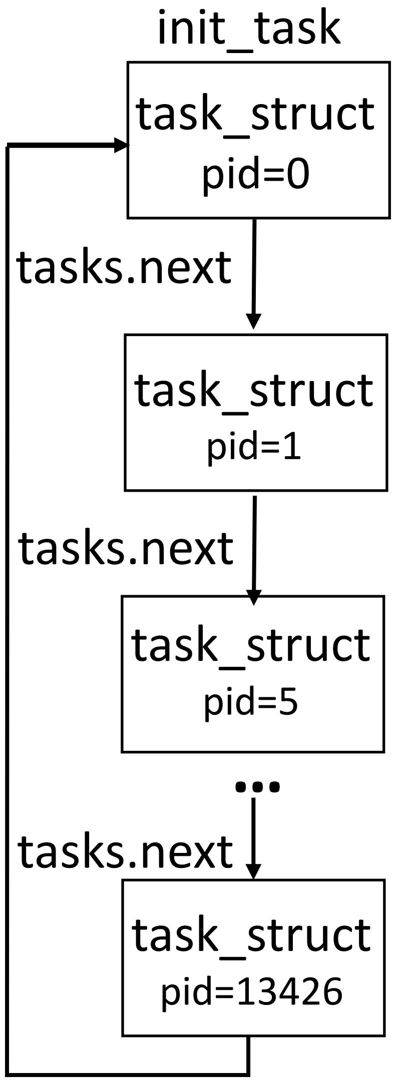
Operators	Descriptions
kgraph(addr)	Initialize traversal at addr
in(ptr)	Dereference ptr into another DS
traverse(ptr_nxt, ptr_end, type)	Traverse ptr_nxt until ptr_end with type
values(f1, …, fn)	Acquire values from current address
…	…



Gremlin Lang

- Memory introspection shares similar execution model with graph processing.

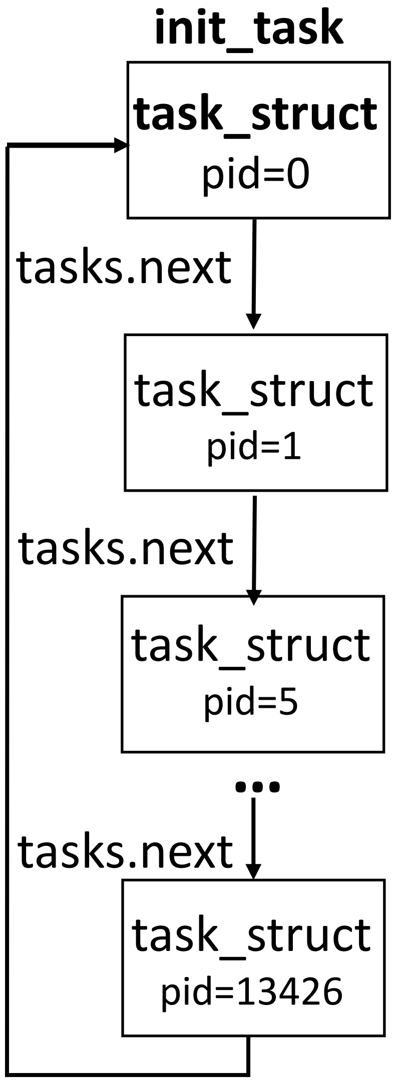
RDMI query example




```
// Go through PSlist and retrieve PID
```

Sample query: Process descriptor list traversal

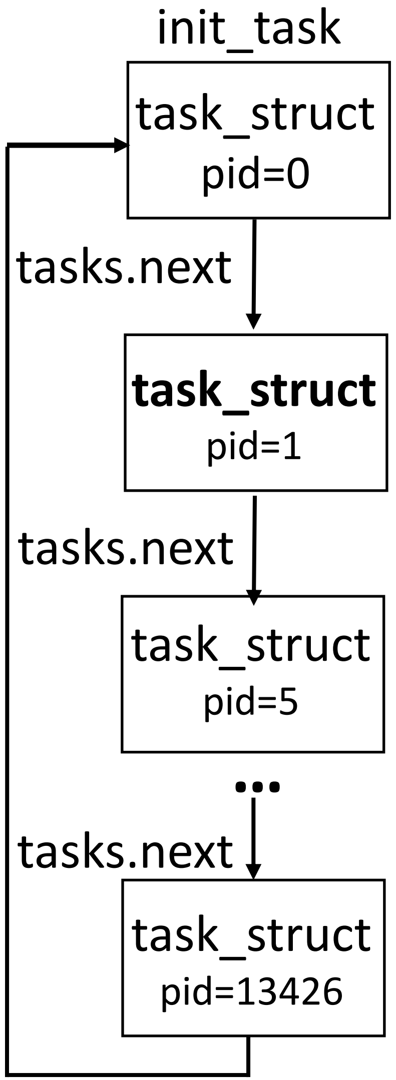
RDMI query example




```
// Go through PSlist and retrieve PID
// Initialize introspection at init_task
kgraph(init_task)
```

Sample query: Process descriptor list traversal

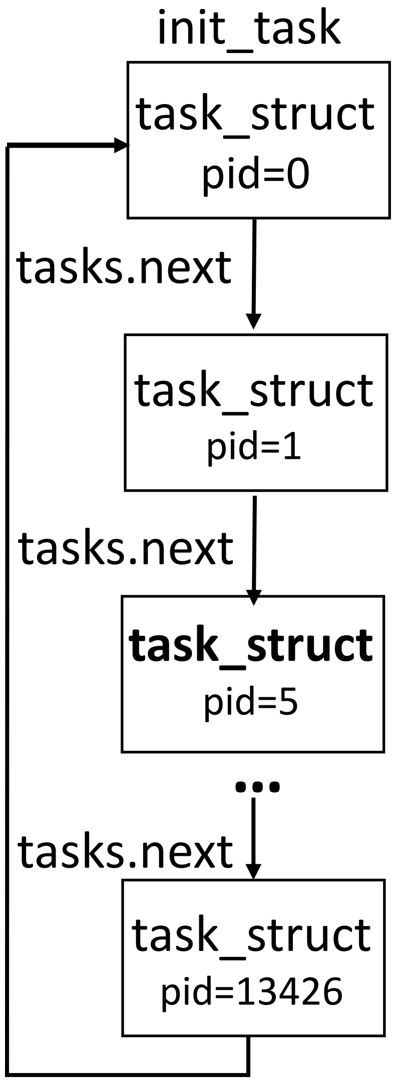
RDMI query example




```
// Go through PSlist and retrieve PID
// Initialize introspection at init_task
kgraph(init_task)
// Traverse task linked list
.traverse(tasks.next, &init_task.tasks, task_struct)
```

Sample query: Process descriptor list traversal

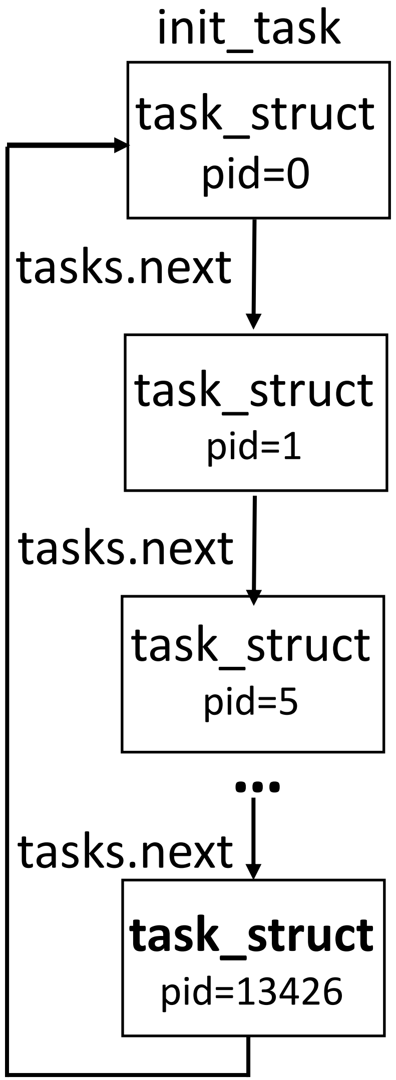
RDMI query example




```
// Go through PSlist and retrieve PID
// Initialize introspection at init_task
kgraph(init_task)
// Traverse task linked list
.traverse(tasks.next, &init_task.tasks, task_struct)
```

Sample query: Process descriptor list traversal

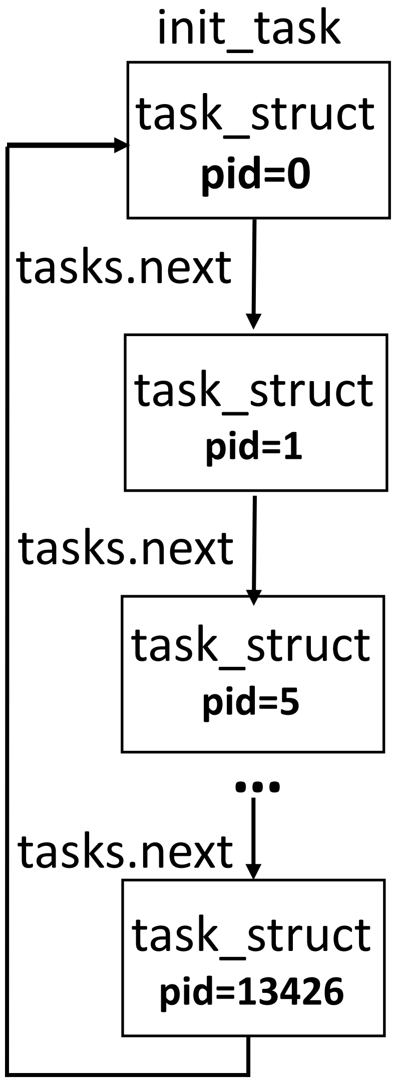
RDMI query example




```
// Go through PSlist and retrieve PID
// Initialize introspection at init_task
kgraph(init_task)
// Traverse task linked list
.traverse(tasks.next, &init_task.tasks, task_struct)
```

Sample query: Process descriptor list traversal

RDMI query example




```
// Go through PSlist and retrieve PID
// Initialize introspection at init_task
kgraph(init_task)
// Traverse task linked list
.traverse(tasks.next, &init_task.tasks, task_struct)
// get each pid value while traversing
.values(pid)
```

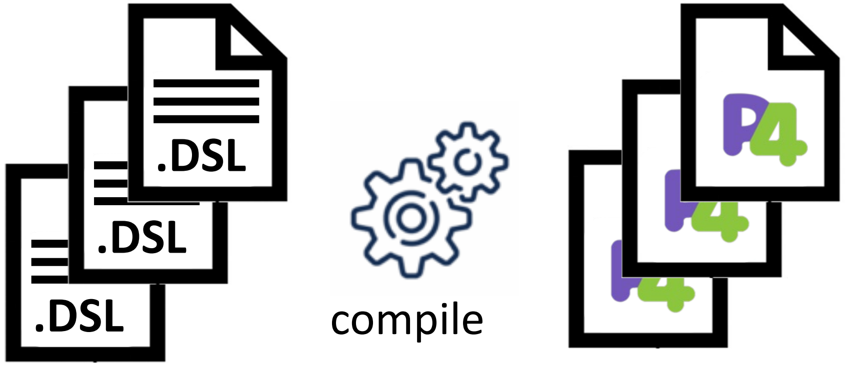
Sample query: Process descriptor list traversal

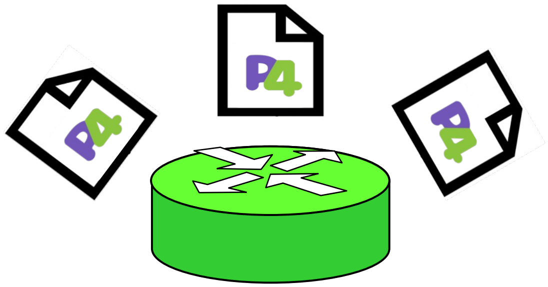
RDMI introspection queries

Policy	LoC	Policy	LoC
P1. Task list traversal	3	P7. Process memory map check	7
P2. Privilege escal. analysis	4	P8. Keyboard sniffer check	5
P3. VFS hook detection	4	P9. Module list traversal	4
P4. Netfilter hijacking detection	7	P10. Afinfo operation check	6
P5. TTY keylogger check	11	P11. Open file list	11
P6. Syscall check	4	-	-

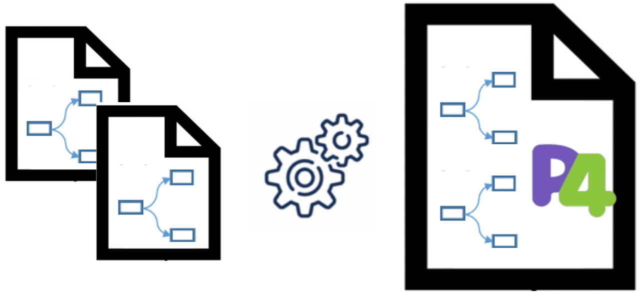
- RDMI can express a range of useful introspection queries with a few LoC

DSL compilation: Naïve solution



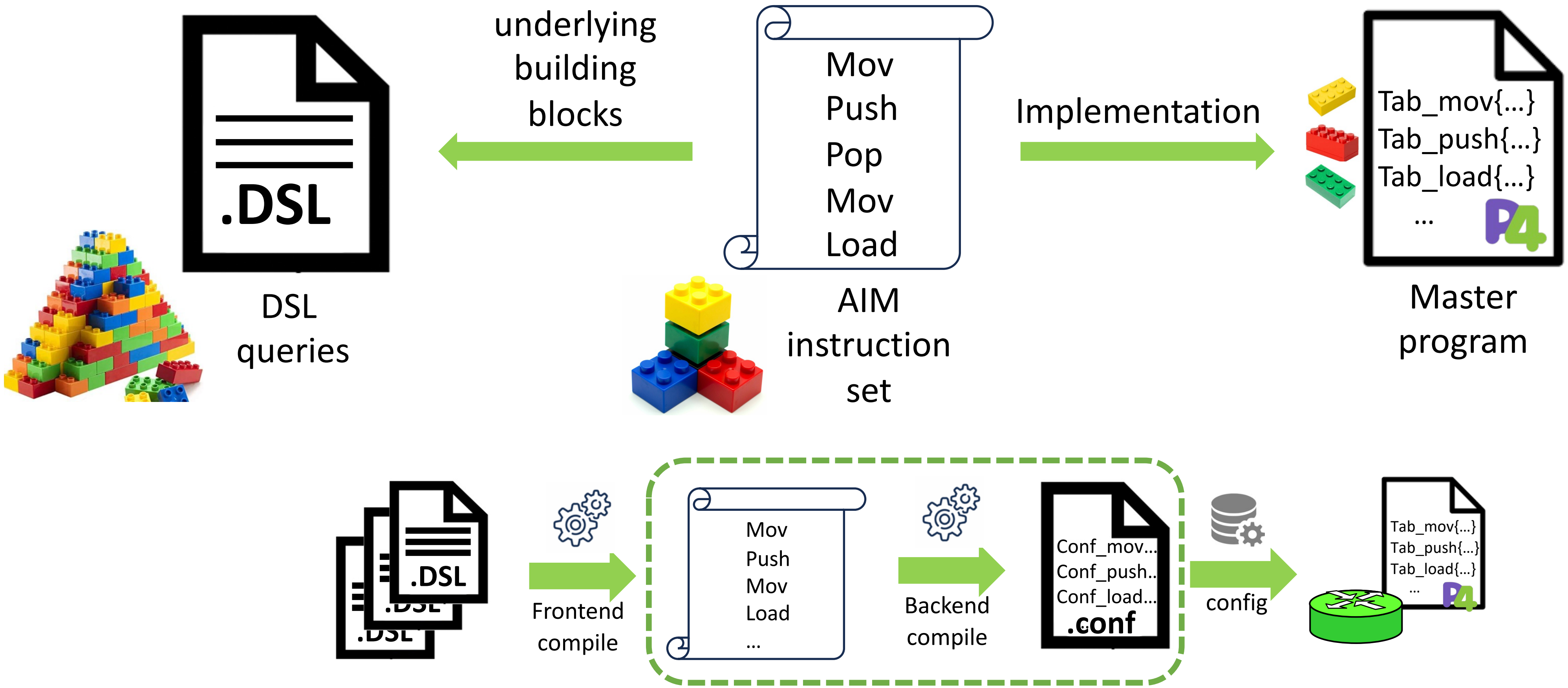


Deploying policies incurs downtime



Sharing resources across policies is hard

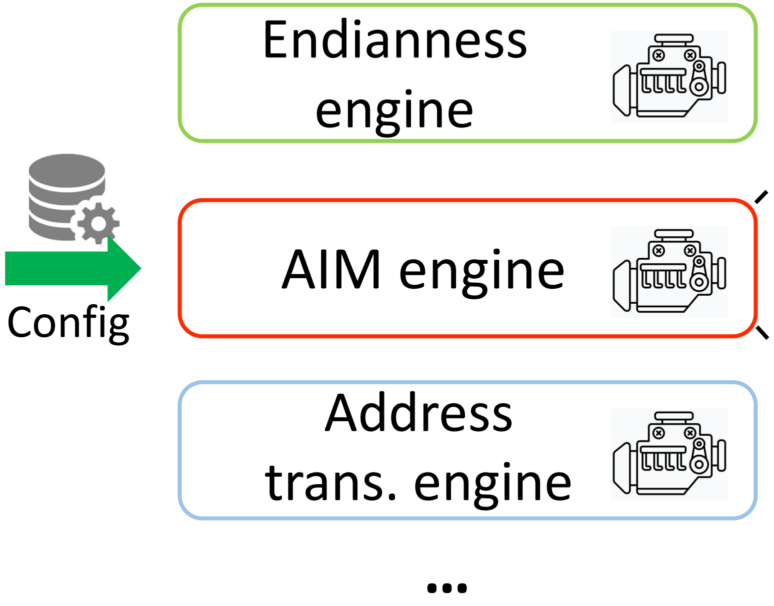
Abstract introspection machine



- AIMs are underlying building block for DSL primitives
- AIMs are implemented in a master switch program
- AIMs are further compiled to configure the match action table

Reconfigurable introspection engines

table	rdma.qpn	meta.pred	param
NxtPC_tab	7	0	8
NxtPC_tab	8	0	9
NxtPC_tab	8	1	10
…	…	…	…



Reconfigurable engine sets

```
table NxtPC_tab {
    key = {
        rdma.qpn:    exact;
        meta.pred:   exact;
    }
    actions = {compute_NxtPC;}
}
…
```

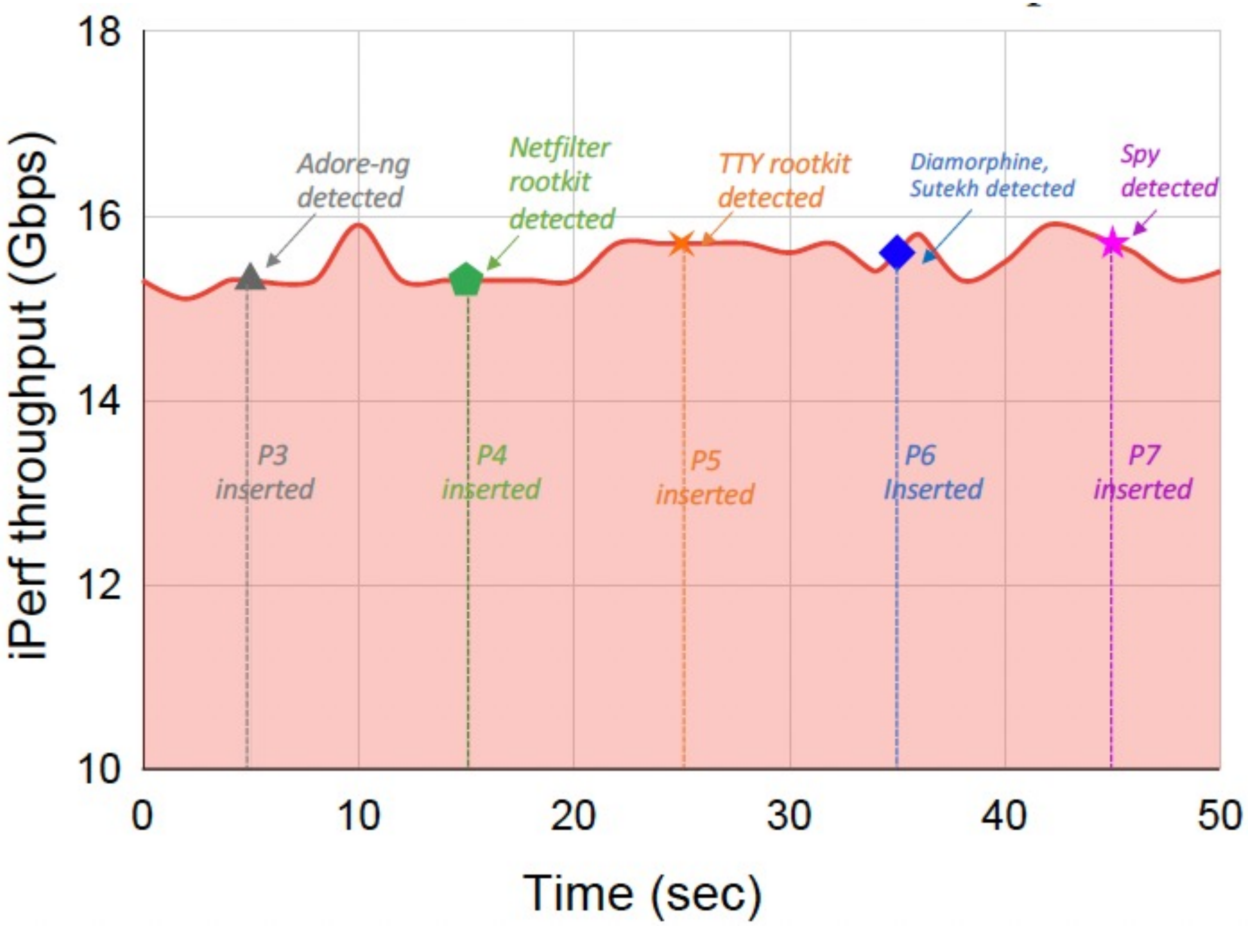
Match action tables (MAT)

- Reconfigurable introspection engines instantiate the AIM instructions
- Engines are implemented as MATs reconfigurable for different AIM streams

Experiment setup

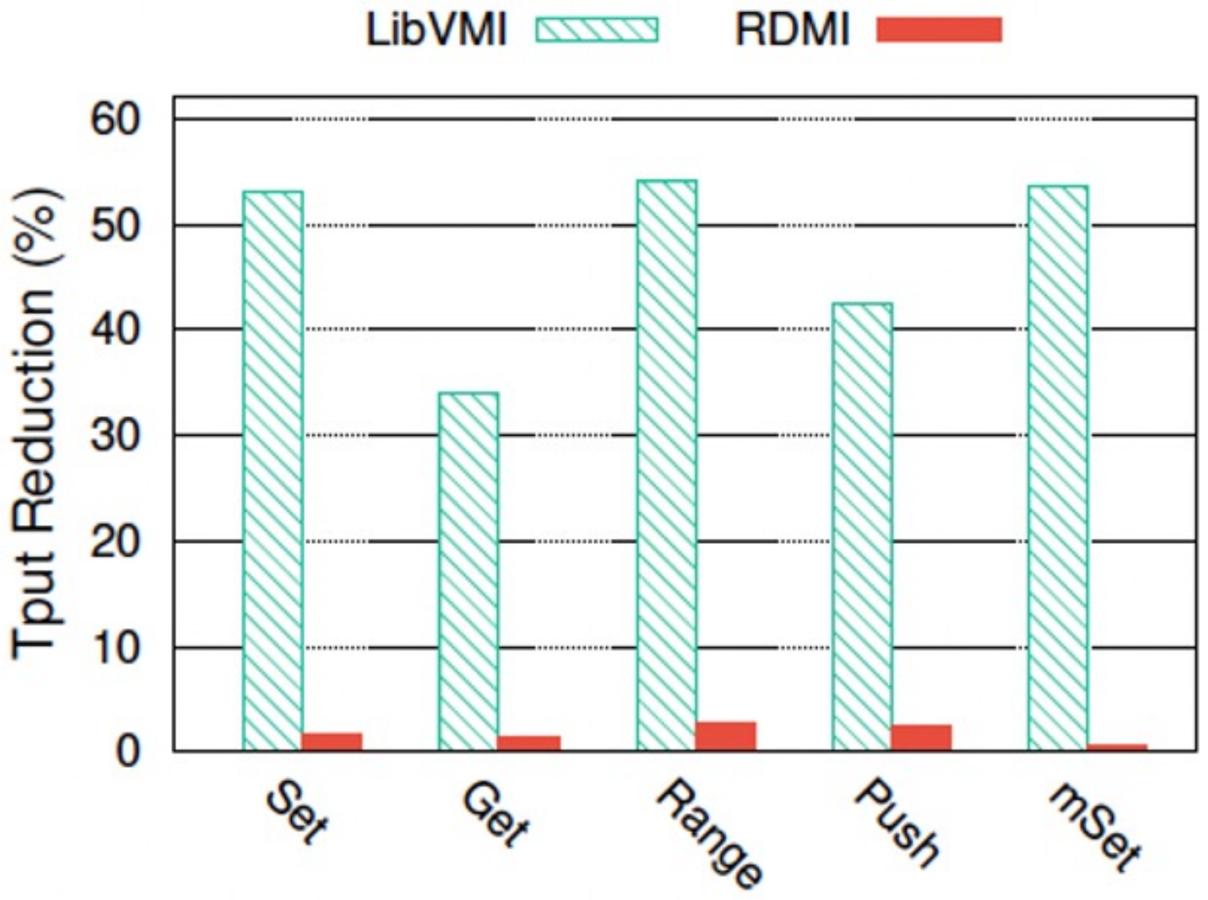
- RDMI prototype:
  - Runs with Tofino Wedge 100BF-32X switch and Mellanox CX-4 NIC
  - 2500 LoC of P4 + 2700 LoC of C++
- Baseline:
  - LibVMI based introspection
- Real world threats:
  - Adore-ng and 5 other rootkits
- Real world applications:
  - Redis and Nginx workloads

Evaluation: Introspection effectiveness



- RDMI detects real-world rootkits in baremetal machine
- RDMI's policy deployment won't affect normal traffic

Evaluation: Workload interference



- RDMI’s interference to guest workload is negligible

Summary

- Motivation: Better memory introspection
- Insight: Dom(-1) security offloading
    - Supported by widely-deployed hardware
- **RDMI: Remote direct memory introspection**
    - DSL support for introspection queries
    - AIM for resource sharing and live deployment
    - Runtime for supporting executions
- RDMI improves cloud security on several aspects
    - E.g., Baremetal support, higher attack detection rates
- Source code: https://github.com/aladinggit/RDMI/

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

hl87 at rice dot edu