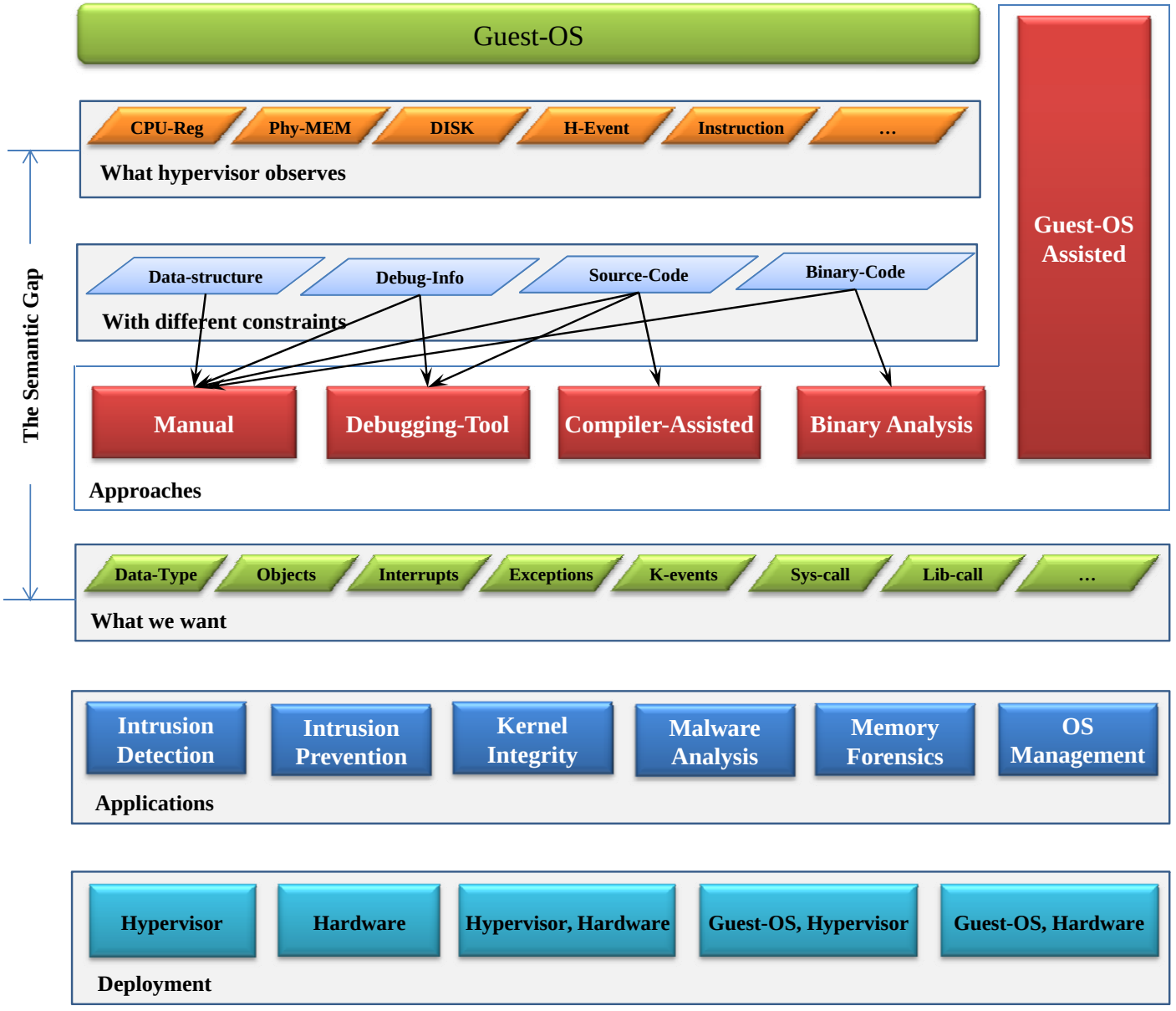
All You Ever Wanted to Know About Virtual Machine Introspection: Approaches to Bridge the Semantic Gap

Zhiqiang Lin

Department of Computer Sciences
The University of Texas at Dallas

August 24^{th}, 2015

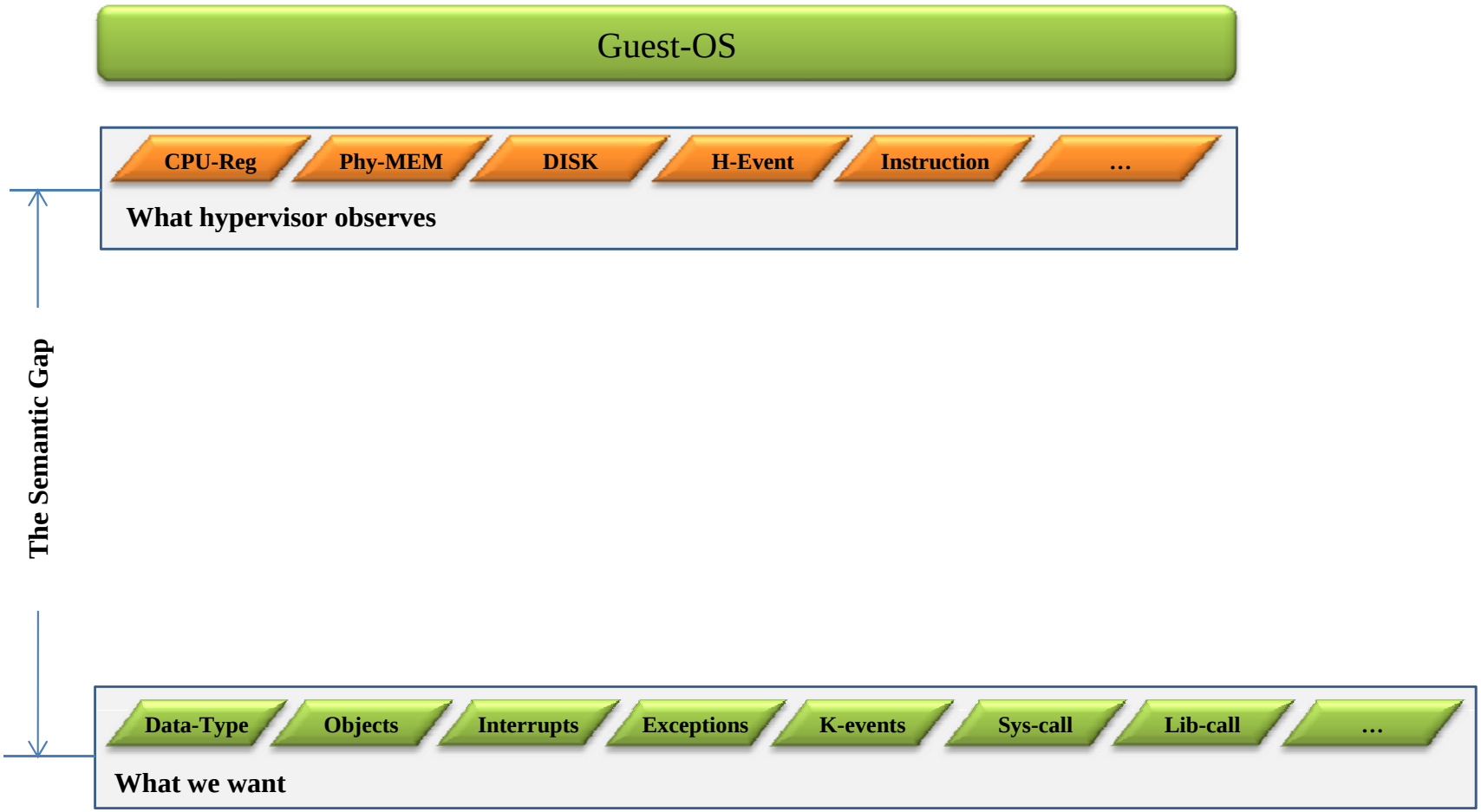
The Road Map


Guest-OS
CPU-Reg
Phy-MEM
DISK
H-Event
Instruction
…
What hypervisor observes
Guest-OS Assisted
Data-structure
Debug-Info
Source-Code
Binary-Code
With different constraints
The Semantic Gap
Manual
Debugging-Tool
Compiler-Assisted
Binary Analysis
Approaches
Data-Type
Objects
Interrupts
Exceptions
K-events
Sys-call
Lib-call
…
What we want
Intrusion Detection
Intrusion Prevention
Kernel Integrity
Malware Analysis
Memory Forensics
OS Management
Applications
Hypervisor
Hardware
Hypervisor, Hardware
Guest-OS, Hypervisor
Guest-OS, Hardware
Deployment

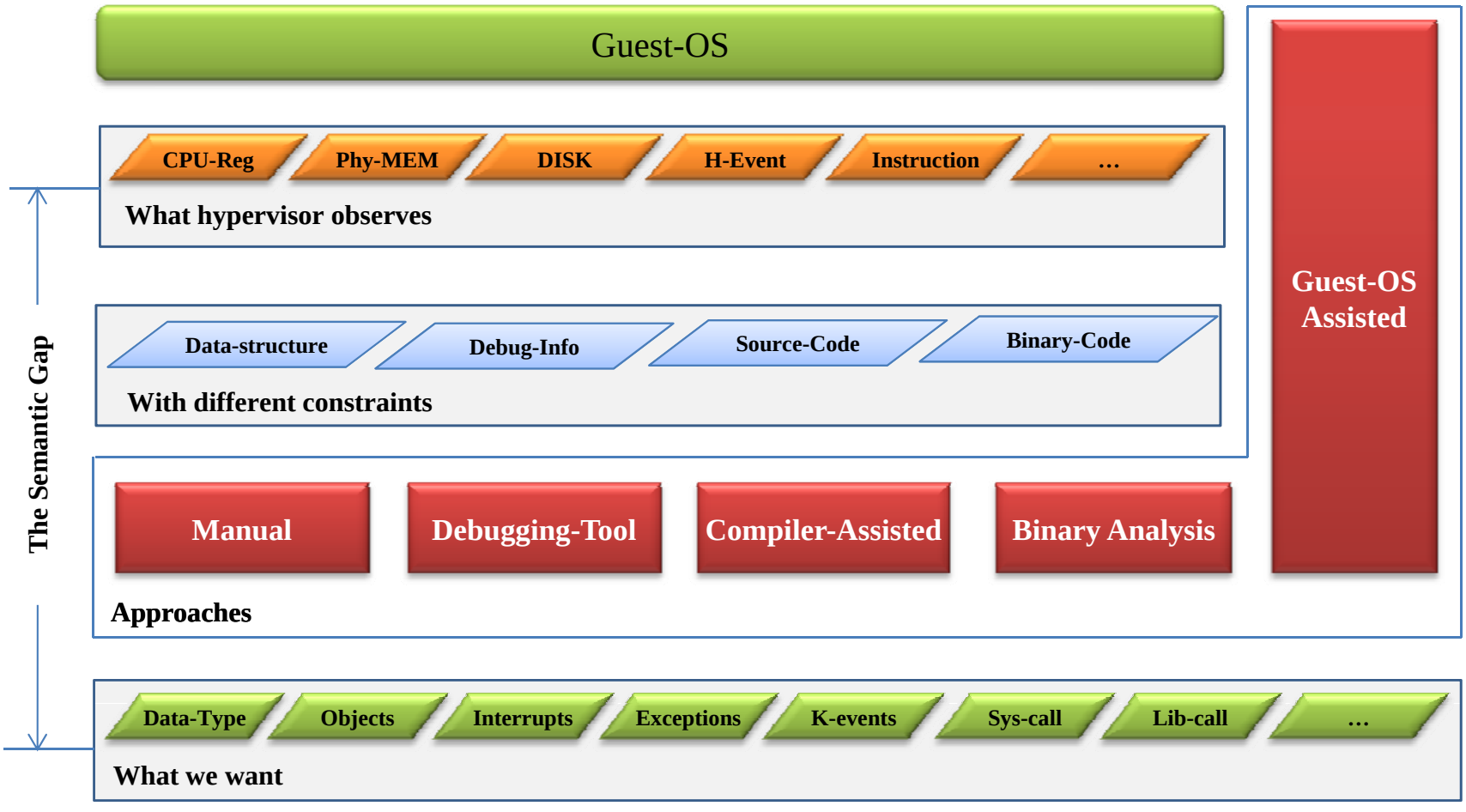
Outline

Outline

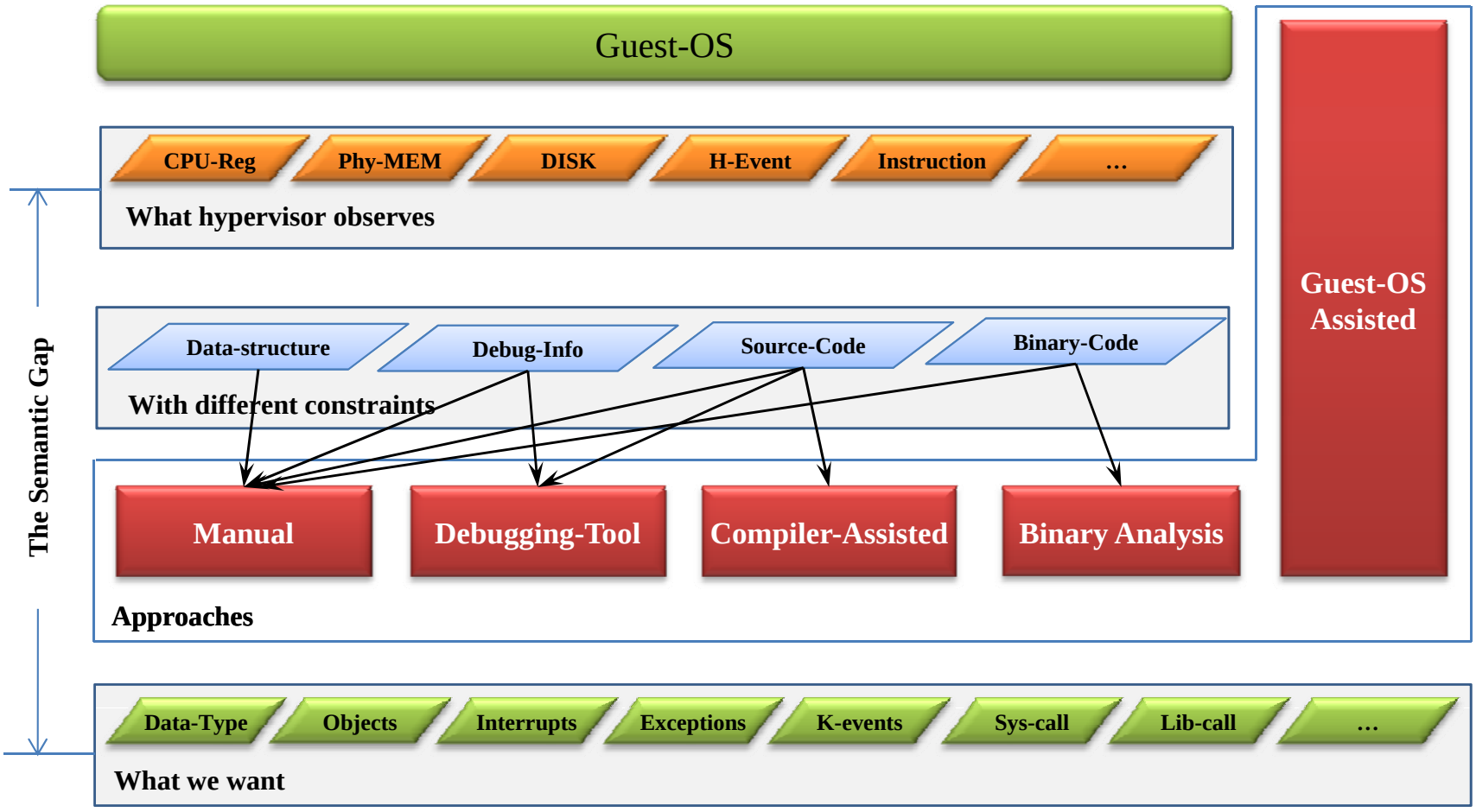
Approach Overview



Approach Overview



Approach Overview



Outline

State-of-the-art in bridging the Semantic Gap

State-of-the-art in bridging the Semantic Gap

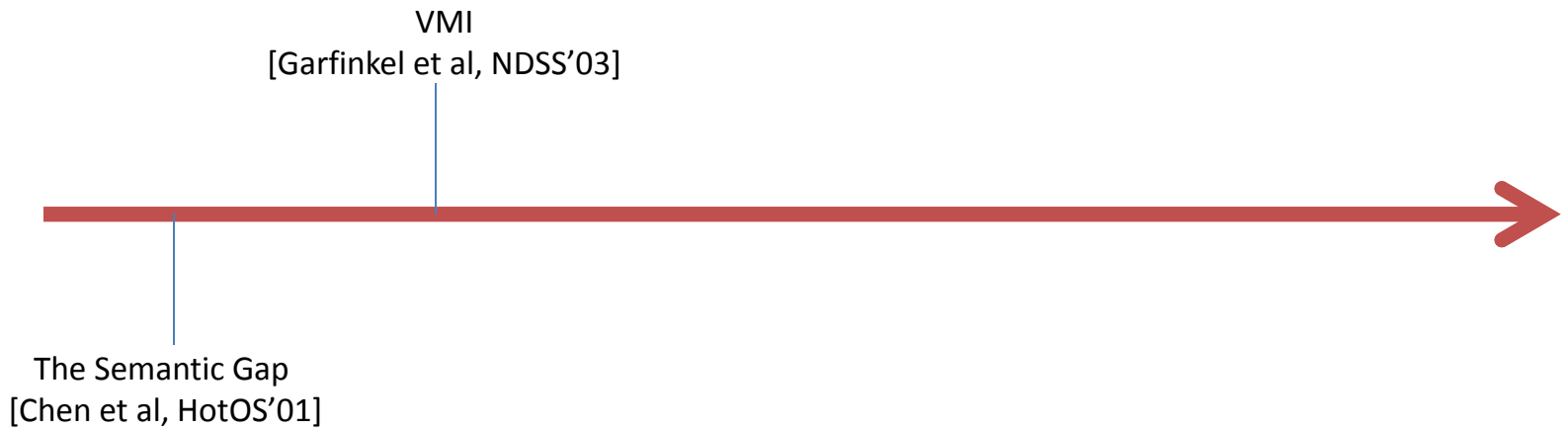
- In HotOS'01, Chen and Noble first raised the semantic gap problem in virtualization

State-of-the-art in bridging the Semantic Gap

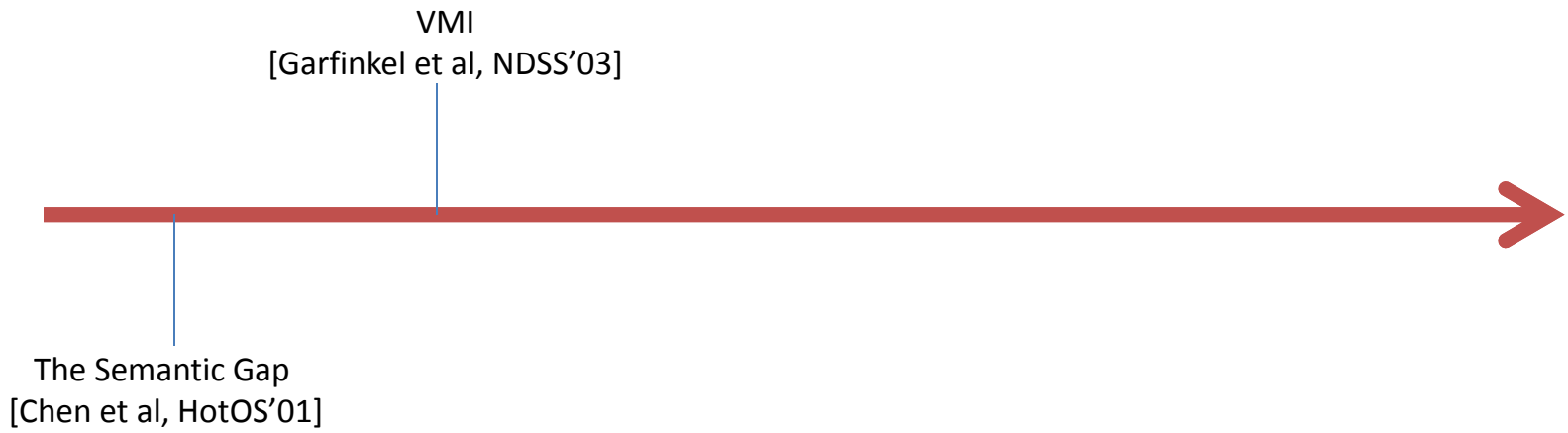
- In HotOS'01, Chen and Noble first raised the semantic gap problem in virtualization

"Services in the VM operate below the abstractions provided by the guest OS ... This can make it difficult to provide services."

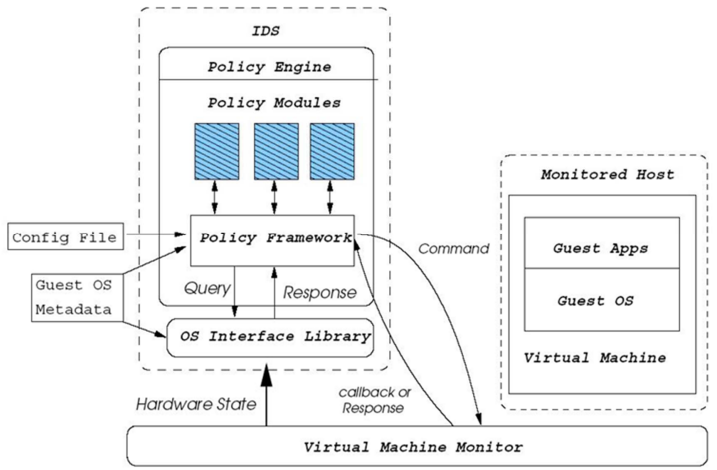
State-of-the-art in bridging the Semantic Gap



State-of-the-art in bridging the Semantic Gap



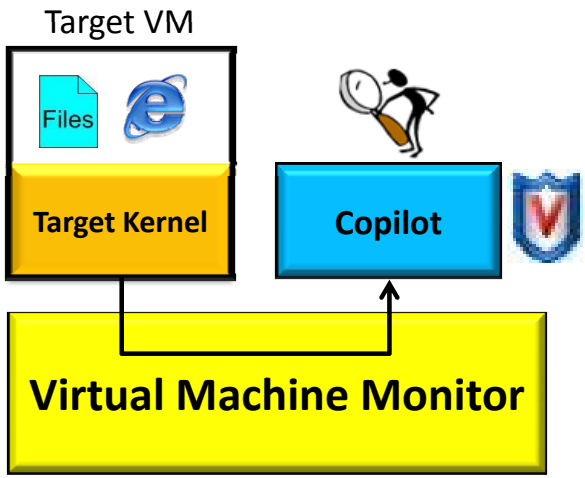
- In NDSS'03, Garfinkel et al. first proposed VMI, demonstrated for IDS
- Introspection routine is based on `crash` utility



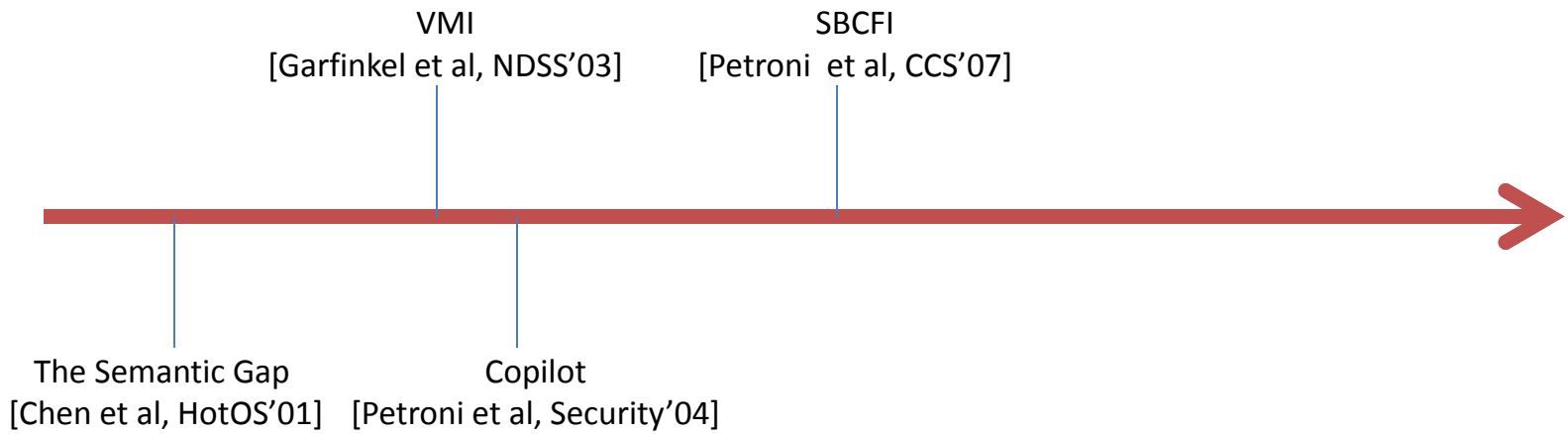
State-of-the-art in bridging the Semantic Gap

State-of-the-art in bridging the Semantic Gap

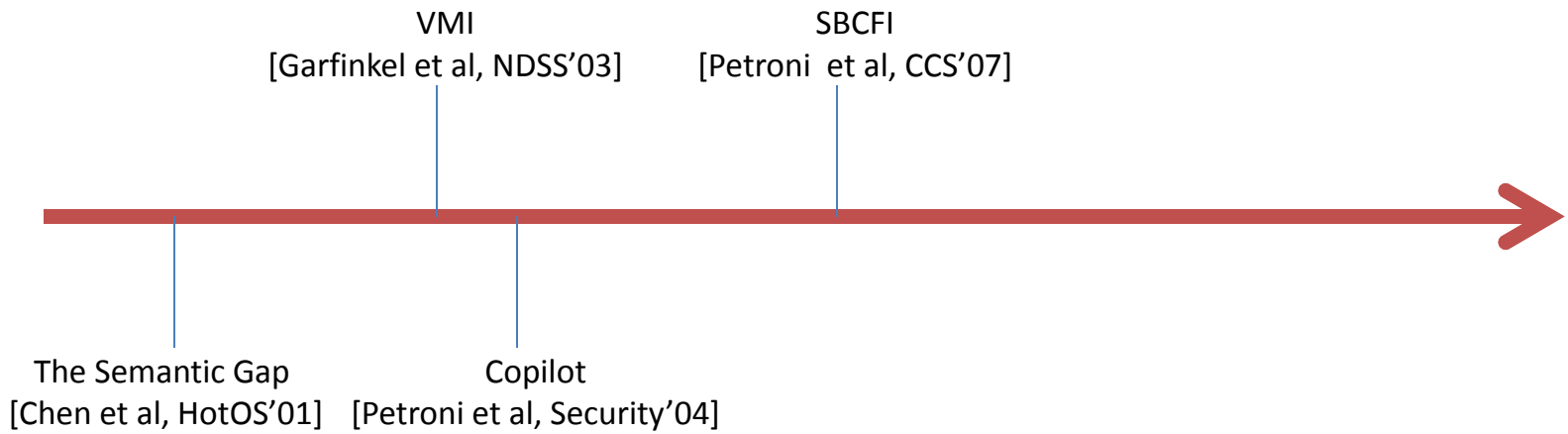
- In USENIX Security'04, Petroni et al. proposed Copilot
- Introspection routine is based on manually created code



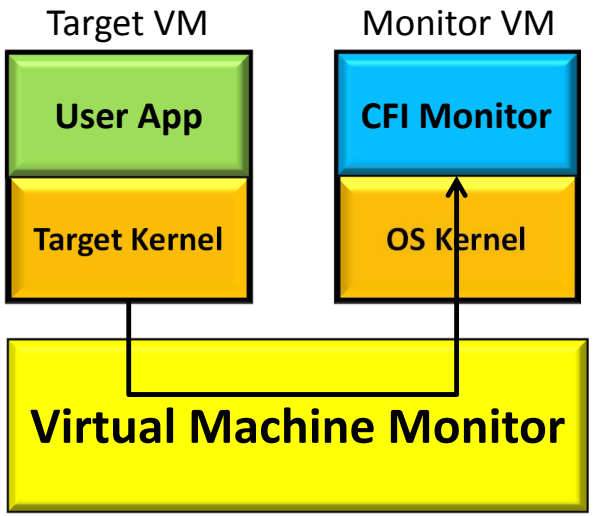
State-of-the-art in bridging the Semantic Gap



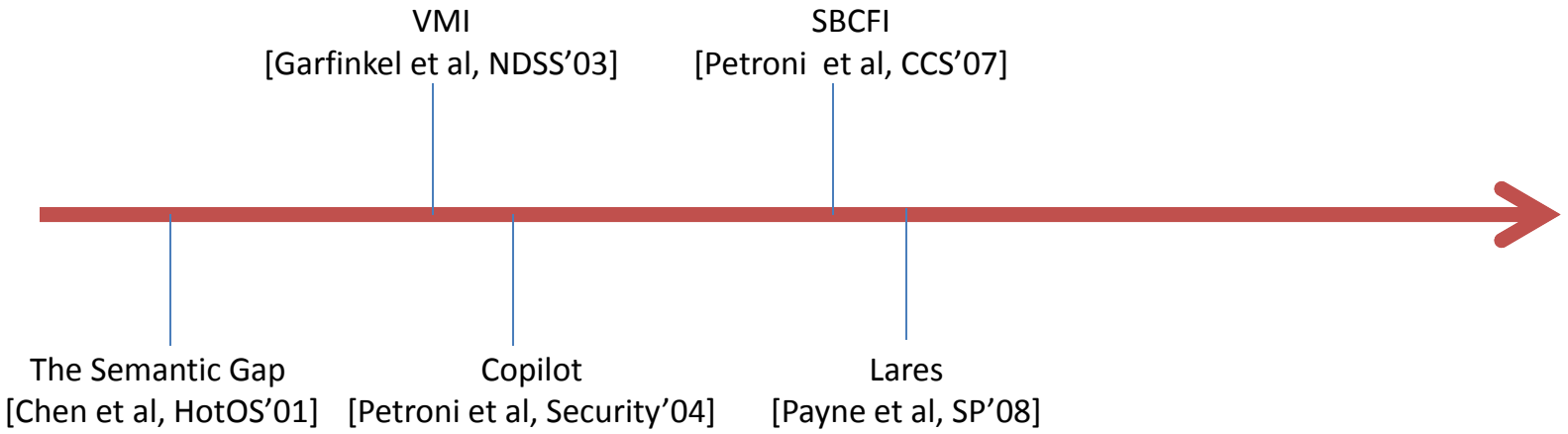
State-of-the-art in bridging the Semantic Gap



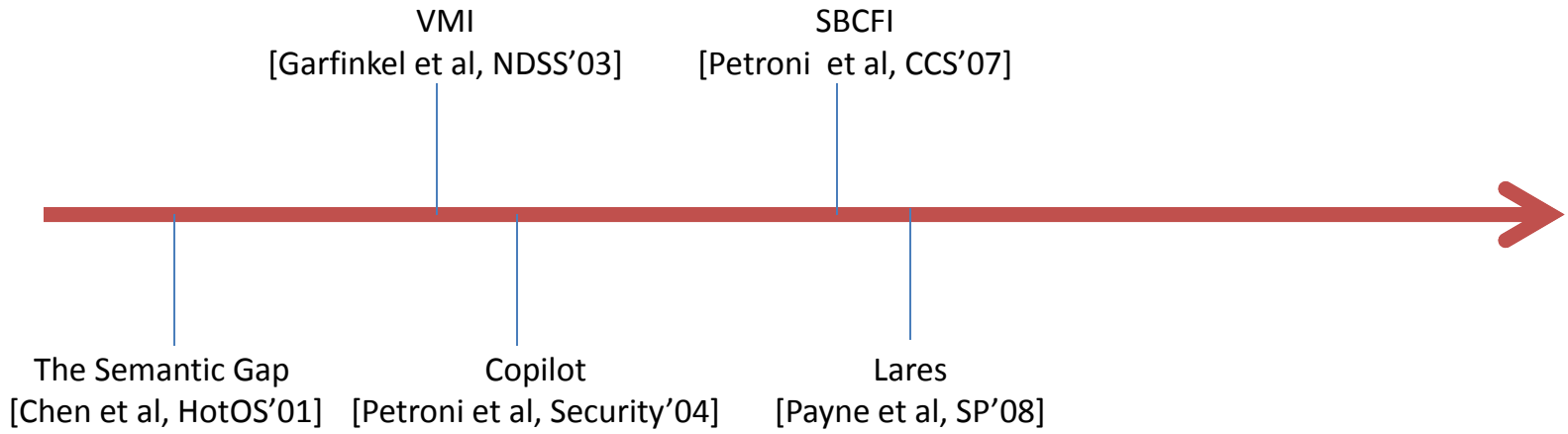
- In CCS'07, Petroni et al. proposed SBCFI
- Introspection routine is based on customized kernel source code



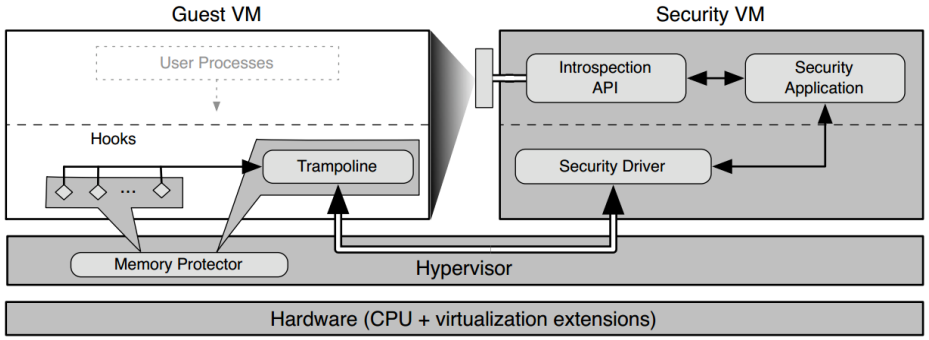
State-of-the-art in bridging the Semantic Gap



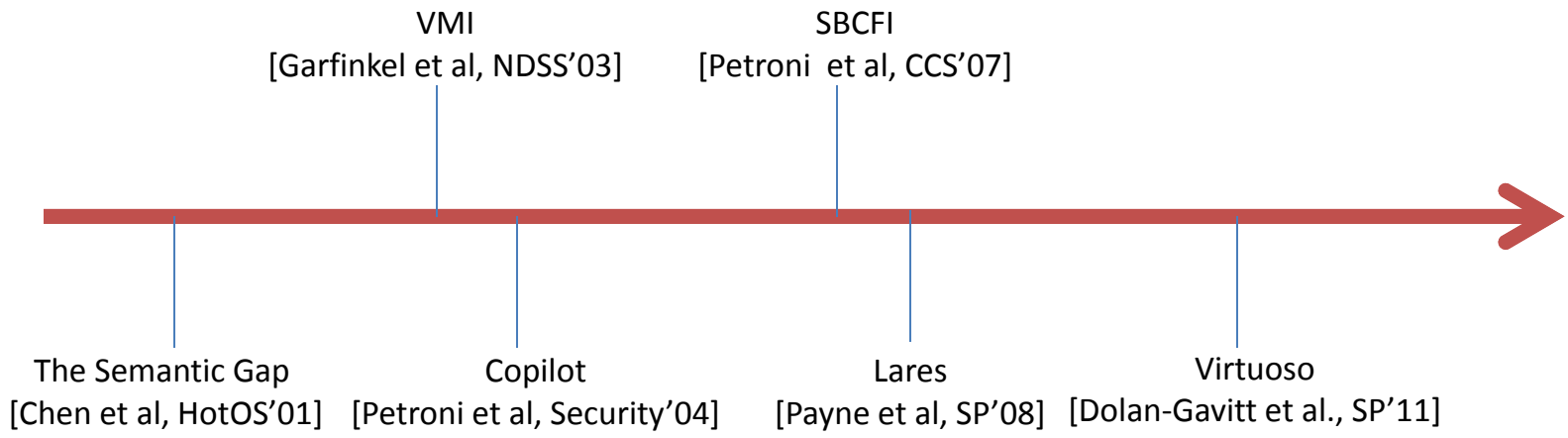
State-of-the-art in bridging the Semantic Gap



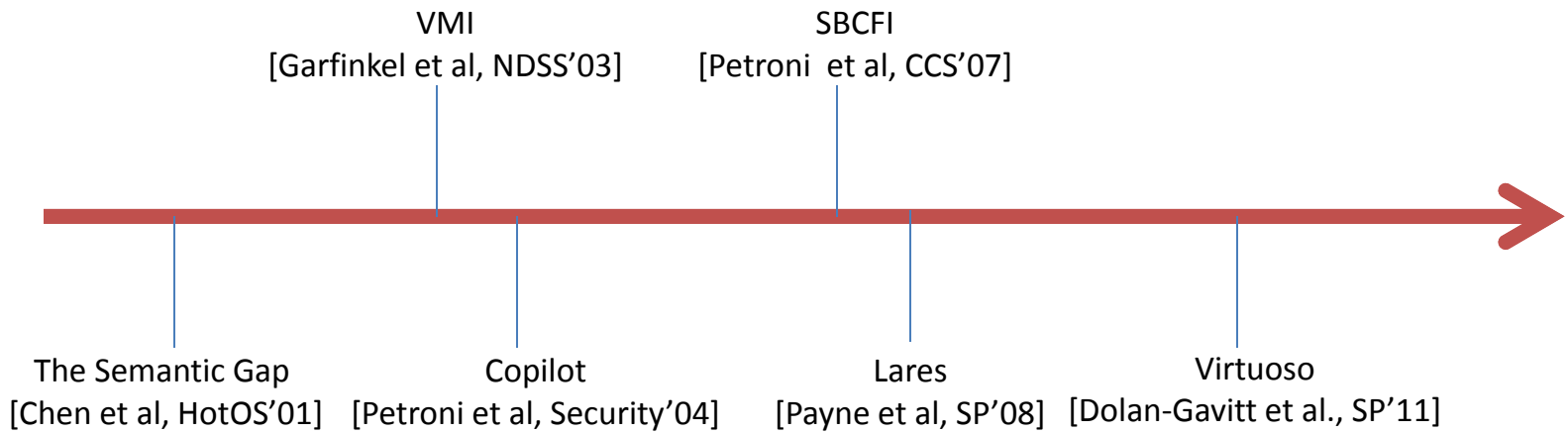
- In SP'08, Payne et al. proposed Lares
- Introspection routine is placed inside the guest OS with special help



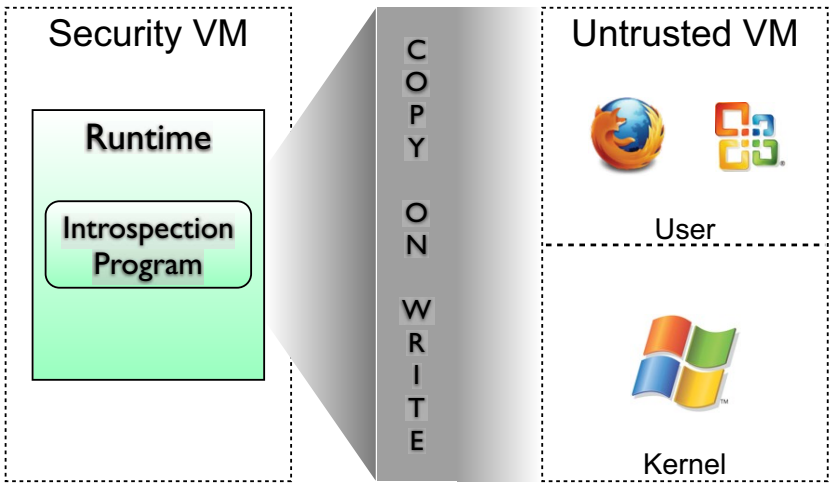
State-of-the-art in bridging the Semantic Gap



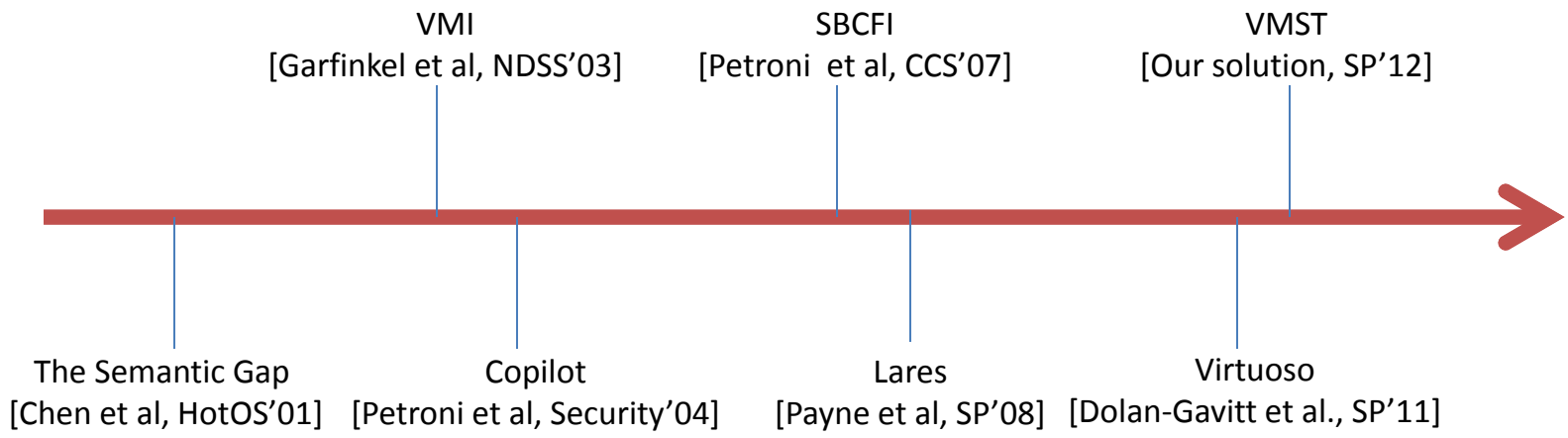
State-of-the-art in bridging the Semantic Gap



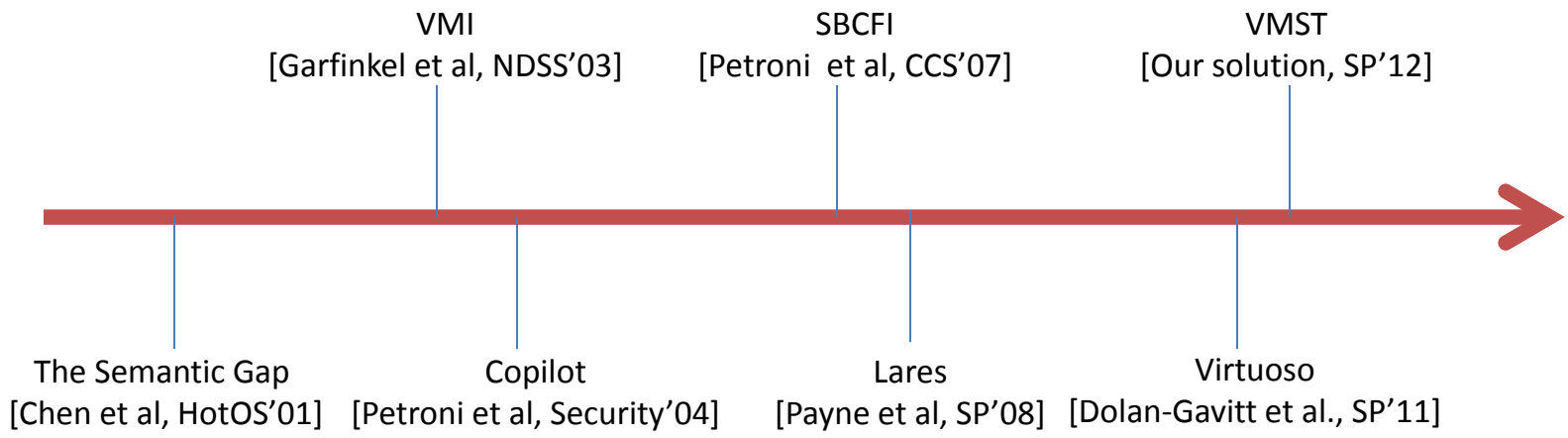
- In SP'11, Dolan-Gavitt et al. proposed Virtuoso
- Introspection routine is based on the trained user level and kernel level code



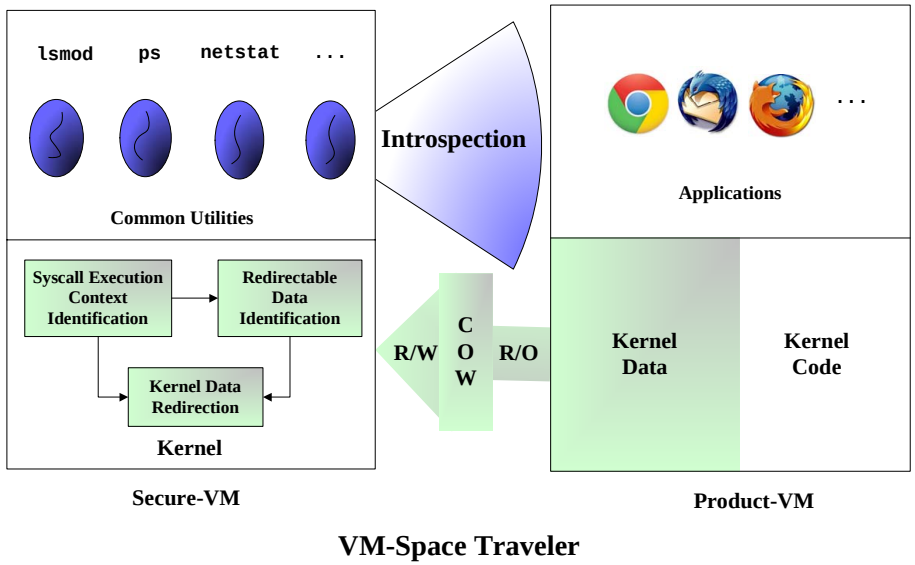
State-of-the-art in bridging the Semantic Gap



State-of-the-art in bridging the Semantic Gap



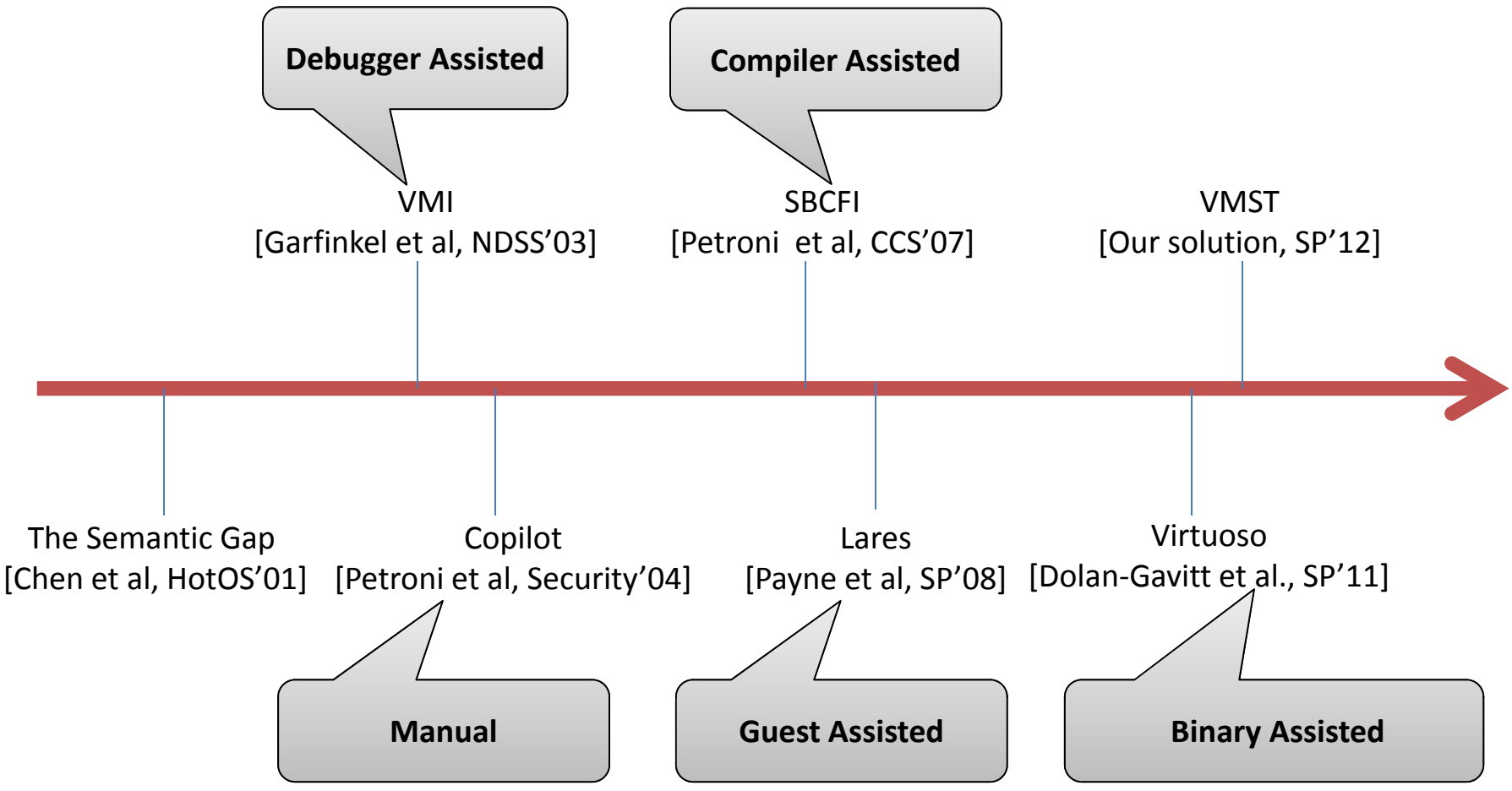
- In SP'12, we propose VM Space Traveler (VMST).
- Introspection routine is automatically generated from the native user level and kernel level code



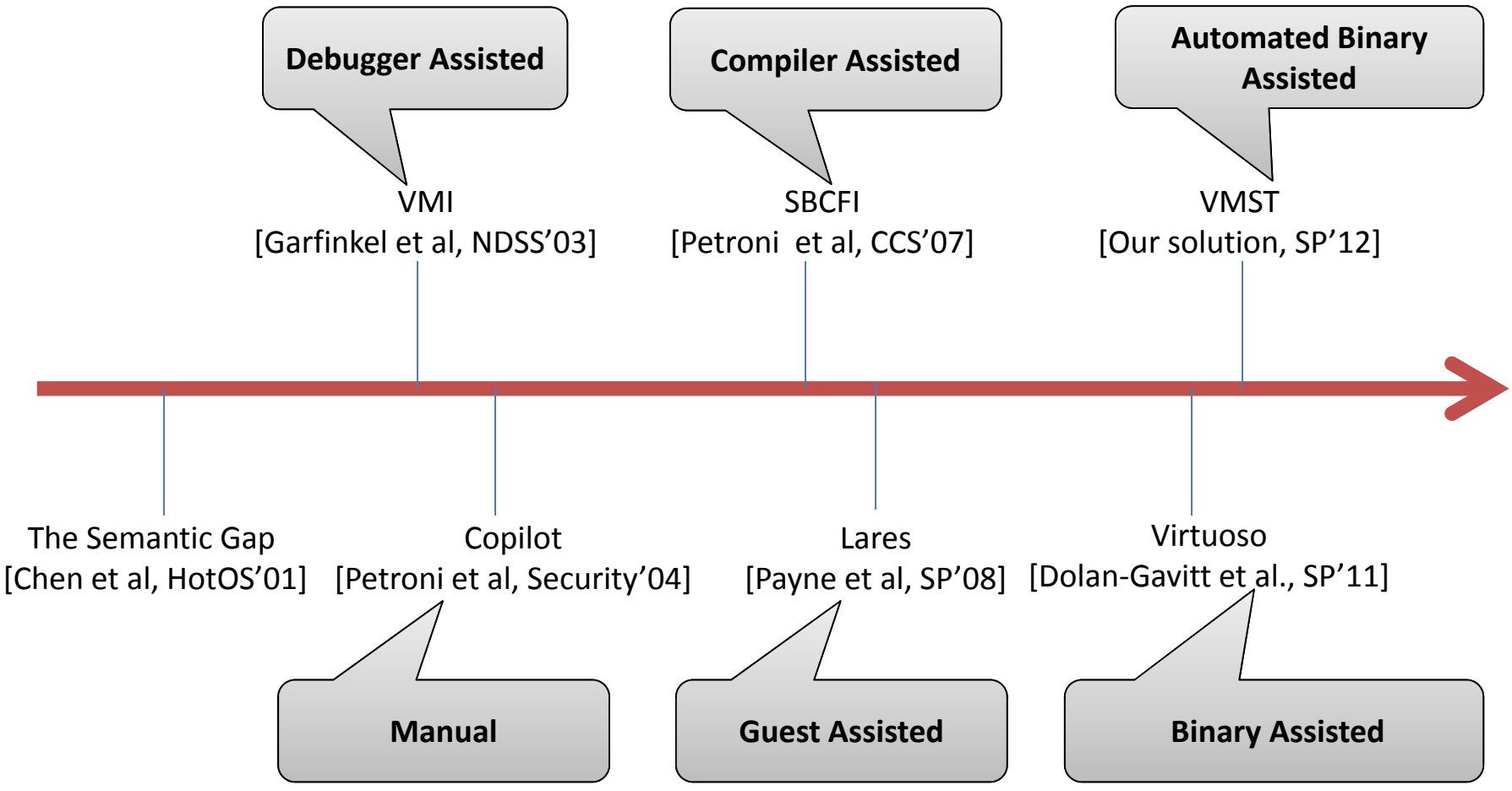
VM-Space Traveler

Outline

Short Summary



Short Summary



Approaches Comparison

Metric	Definition
Flexibility	How many constraints the approach relies on (e.g. if the approach relies on access to OS source code, it is less flexible)
Coverage	How many abstractions can be derived; the scope of the approach
Easiness	How difficult the approach is to implement
Practicality	How useful and adoptable the approach is for real-world applications
Automation	How much can be done automatically instead of by hand

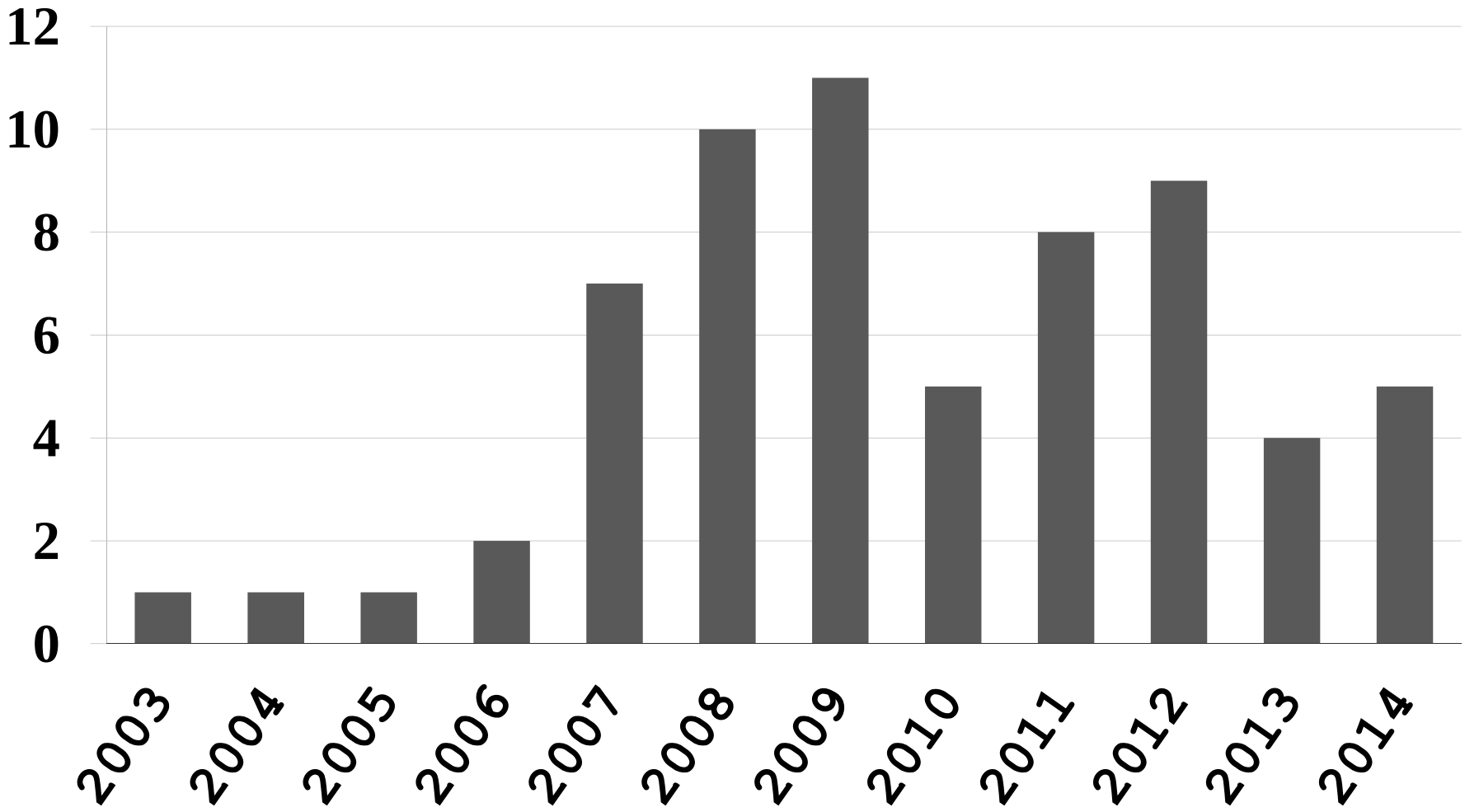
Table : Definitions of the metrics used to compare out-of-VM approaches.

Approaches Comparison

Approaches	Year Started	Flexibility	Coverage	Easiness	Practicality	Capability	Automation
Debugger Assisted	2003	◐	●	●	○	◐	●
Manual	2004	●	○	●	◐	○	○
Compiler Assisted	2007	○	●	◐	○	●	●
Guest Assisted	2008	◐	●	●	◐	●	◐
Binary Analysis Assisted	2011	●	●	○	●	●	●

Table : Comparison between different out-of-VM approaches. Note that symbol ○ denotes a low degree for that comparison item, ◐ denotes a medium degree, and ● denotes a high degree.

Academic Papers



Academic Papers

System		Approaches					Application			Deployment			
Name	**Venue**	**Manual**	**Debugger Assisted**	**Compiler Assisted**	**Binary Analysis**	**Guest Assisted**	**Detection**	**Prevention**	**Recovery**	**Bare Metal**	**Hosted Native**	**Hosted Emulation**	**Extra Hardware**
LIVEWIRE [Garfinkel and Rosenblum 2003]	NDSS '03		✓				✓					✓	
COPILOT [Petroni et al. 2004]	USENIX-SEC '04	✓					✓						✓
INTROVIRT [Joshi et al. 2005]	SOSP '05		✓				✓				✓		
ANTFARM [Jones et al. 2006]	USENIX-ATC '06	✓					✓			✓			
PFWA [Petroni et al. 2006]	USENIX-SEC '06	✓					✓						✓
EKKYS [Egele et al. 2007]	USENIX-ATC '07	✓					✓					✓	
VMSCOPE [Jiang and Wang 2007]	RAID '07	✓					✓					✓	
VMWATCHER [Jiang et al. 2007]	CCS '07	✓					✓					✓	
PANORAMA [Yin et al. 2007]	CCS '07	✓					✓					✓	
SBCFI [Petroni and Hicks 2007]	CCS '07			✓			✓			✓	✓	✓	
SECVISOR [Seshadri et al. 2007]	SOSP '07	✓						✓		✓			
XENACCESS [Payne et al. 2007]	ACSAC '07	✓					✓			✓			
HOOKFINDER [Yin et al. 2008]	NDSS '08	✓					✓					✓	
LYCOSID [Jones et al. 2008]	VEE '08	✓					✓			✓			
OVERSHADOW [Chen et al. 2008]	ASPLOS '08	✓						✓			✓		

Academic Papers

Lares [Payne et al. 2008]	SP '08			✓		✓		✓			
Patagonix [Litty et al. 2008]	USENIX-SEC '08	✓			✓	✓		✓			
HookMap [Wang et al. 2008]	RAID '08		✓		✓					✓	
NICKLE [Riley et al. 2008]	RAID '08	✓				✓				✓	
Ether [Dinaburg et al. 2008]	CCS '08	✓			✓			✓			
VICI [Fraser et al. 2008]	ACSAC '08	✓					✓	✓			
Gibraltar [Baliga et al. 2008]	ACSAC '08			✓	✓			✓	✓	✓	
Anubis [Bayer et al. 2009]	NDSS '09	✓			✓					✓	
Ktracer [Lanzi et al. 2009]	NDSS '09	✓			✓					✓	
PoKeR [Riley et al. 2009]	EuroSys '09		✓		✓					✓	
Rkprofiler [Xuan et al. 2009]	RAID '09	✓			✓					✓	
Kop [Carbone et al. 2009]	CCS '09			✓	✓			✓	✓	✓	
HookSafe [Wang et al. 2009]	CCS '09			✓		✓		✓			
Slcl [Sharif et al. 2009]	CCS '09			✓	✓	✓			✓		
Dgstg [Dolan-Gavitt et al. 2009]	CCS '09	✓			✓			✓	✓		
OSVM [Oliveira and Wu 2009]	ACSAC '09			✓		✓				✓	
MAVMM [Nguyen et al. 2009]	ACSAC '09	✓			✓			✓			
HIMA [Azab et al. 2009]	ACSAC '09	✓			✓	✓		✓			

Academic Papers

PMPDFGJ [Paleari et al. 2010]	USENIX-SEC '10	✓					✓			✓	
HYPERCHECK [Wang et al. 2010]	RAID '10	✓			✓						✓
LIVEDM [Rhee et al. 2010]	RAID '10		✓		✓					✓	
TRAILOFBYTES [Krishnan et al. 2010]	CCS '10	✓			✓			✓			
PEDA [Zhang et al. 2010]	ACSAC '10	✓			✓			✓		✓	
HUKO [Xiong et al. 2011]	NDSS '11			✓		✓		✓			
SIGGRAPH [Lin et al. 2011]	NDSS '11		✓		✓			✓	✓	✓	
GATEWAY [Srivastava andGiffin 2011]	NDSS '11		✓		✓	✓		✓			
OSCK [Hofmann et al. 2011]	ASPLOS '11		✓		✓				✓		
VIRTUOSO [Dolan-Gavitt et al. 2011a]	SP '11			✓	✓			✓	✓	✓	
SHELLOS [Snow et al. 2011]	USENIX-SEC '11	✓			✓				✓		
SWJX [Srinivasan et al.2011]	CCS '11			✓	✓				✓		
GDXJ [Gu et al. 2011]	SRDS '11			✓	✓		✓		✓		
KRUISER [Tian et al. 2012]	NDSS '12			✓	✓			✓			
V2E [Yan et al. 2012]	VEE '12	✓			✓				✓	✓	
VMST [Fu and Lin 2012]	SP '12			✓	✓					✓	
MAS [Cui et al. 2012]	USENIX-SEC '12		✓		✓			✓	✓	✓	
DROIDSCOPE [Yan and Yin 2012]	USENIX-SEC '12	✓			✓					✓	

(Continued)

Academic Papers

System		Approaches					Application			Deployment			
Name	**Venue**	**Manual**	**Debugger Assisted**	**Compiler Assisted**	**Binary Analysis**	**Guest Assisted**	**Detection**	**Prevention**	**Recovery**	**Bare Metal**	**Hosted Native**	**Hosted Emulation**	**Extra Hardware**
SYRINGE [Carbone et al. 2012]	RAID '12					✓	✓			✓			
VIGILARE [Moon et al. 2012]	CCS '12	✓					✓						✓
BLACKSHEEP [Bianchi et al. 2012]	CCS '12				✓		✓					✓	
SENTRY [Srivastava and Giffin 2012]	ACSAC '12					✓	✓	✓		✓			
EXTERIOR [Fu and Lin 2013b]	VEE '13				✓		✓		✓			✓	
KI-MON [Lee et al. 2013]	USENIX-SEC '13	✓					✓						✓
MOSS [Prakash et al. 2013]	DSN '13		✓				✓			✓	✓	✓	
TZB [Dolan-Gavitt et al. 2013]	CCS '13				✓		✓					✓	
HYBRIDBRIDGE [Saberi et al. 2014]	NDSS '14				✓		✓				✓	✓	
RTKDSM [Hizver and Chiueh 2014]	VEE '14	✓					✓			✓			
HYPERSHELL [Fu et al. 2014]	USENIX-ATC '14					✓	✓		✓		✓		
WCLM [Wu et al. 2014]	DSN '14					✓	✓		✓		✓		
TZ-RKP [Azab et al. 2014]	CCS '14	✓					✓						✓
Total	64	33	5	8	6	12	55	12	6	27	19	30	6

Outline

Our Recent Publications

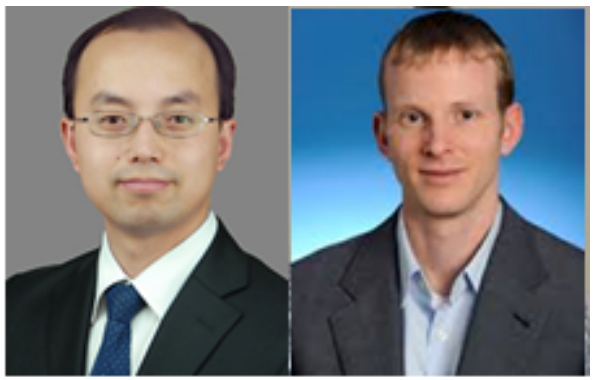
Using **Binary Code Analysis** to bridge the semantic gap and push new applications

1. VM Space Traveling: Automatically Bridging the Semantic Gap in **Virtual Machine Introspection** via Online Kernel Data Redirection. Yangchun Fu, Zhiqiang Lin (*IEEE Symposium on Security and Privacy* [Oakland'12])
2. EXTERIOR: Using A **Dual-VM** Enabled External Shell for Guest-OS Introspection, Configuration, and Recovery Yangchun Fu, Zhiqiang Lin (*ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* [VEE'13])
3. Hybrid-Bridge: Efficiently Bridging the Semantic-Gap in **Virtual Machine Introspection** via Decoupled Execution and Training Memoization. Alireza Saberi, Yangchun Fu, Zhiqiang Lin (*Network and Distributed System Symposium* [NDSS'14])
4. HyperShell: A Practical Hypervisor Layer Guest OS Shell for Automated In-**VM Management**. Yangchun Fu, Junyuan Zeng, Zhiqiang Lin (*USENIX Annual Technical Conference* [USENIX-ATC'14])
5. Automatically Deriving Pointer Reference Expressions From **Binary Code** For Memory Dump Analysis. Yangchun Fu, Zhiqiang Lin, David Brumley. (*ACM SIGSOFT Symposium on the Foundations of Software Engineering* [ESEC/FSE'15]).

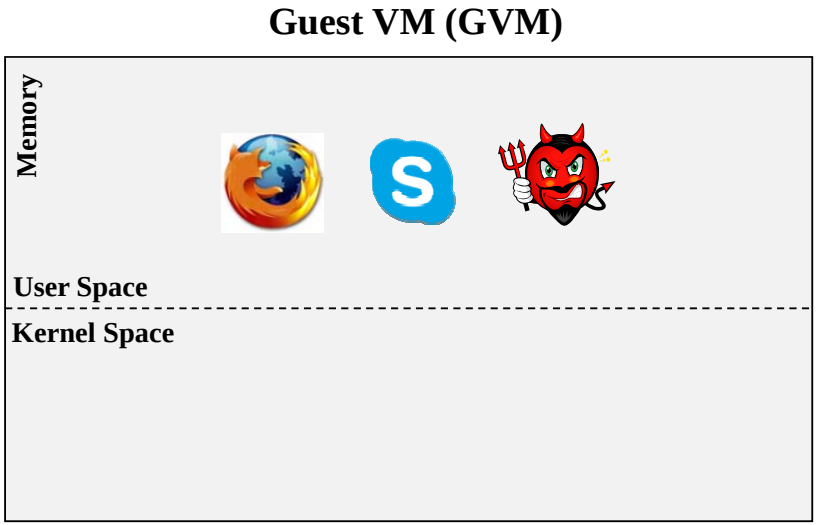
Our Recent Publications

Using **Binary Code Analysis** to bridge the semantic gap and push new applications

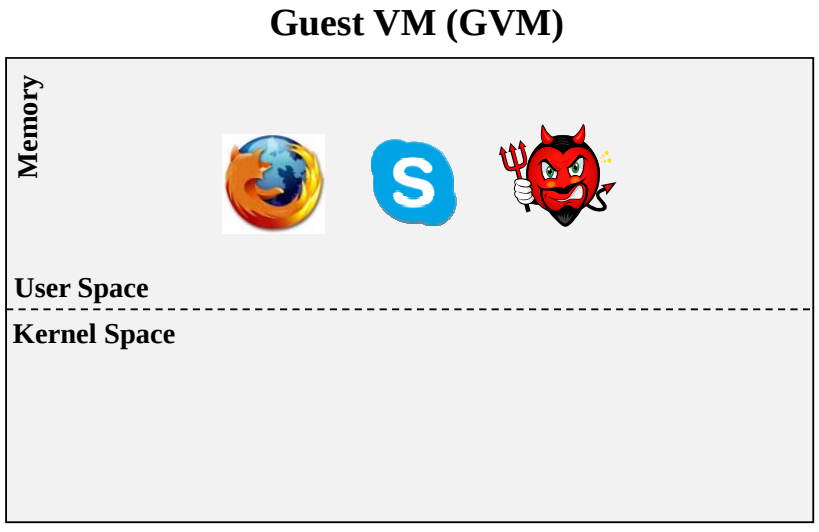
1. VM Space Traveling: Automatically Bridging the Semantic Gap in **Virtual Machine Introspection** via Online Kernel Data Redirection. Yangchun Fu, Zhiqiang Lin (*IEEE Symposium on Security and Privacy* [Oakland'12])
2. EXTERIOR: Using A **Dual-VM** Enabled External Shell for Guest-OS Introspection, Configuration, and Recovery Yangchun Fu, Zhiqiang Lin (*ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* [VEE'13])
3. Hybrid-Bridge: Efficiently Bridging the Semantic-Gap in **Virtual Machine Introspection** via Decoupled Execution and Training Memoization. Alireza Saberi, Yangchun Fu, Zhiqiang Lin (*Network and Distributed System Symposium* [NDSS'14])
4. HyperShell: A Practical Hypervisor Layer Guest OS Shell for Automated In-**VM Management**. Yangchun Fu, Junyuan Zeng, Zhiqiang Lin (*USENIX Annual Technical Conference* [USENIX-ATC'14])
5. Automatically Deriving Pointer Reference Expressions From **Binary Code** For Memory Dump Analysis. Yangchun Fu, Zhiqiang Lin, David Brumley. (*ACM SIGSOFT Symposium on the Foundations of Software Engineering* [ESEC/FSE'15]).



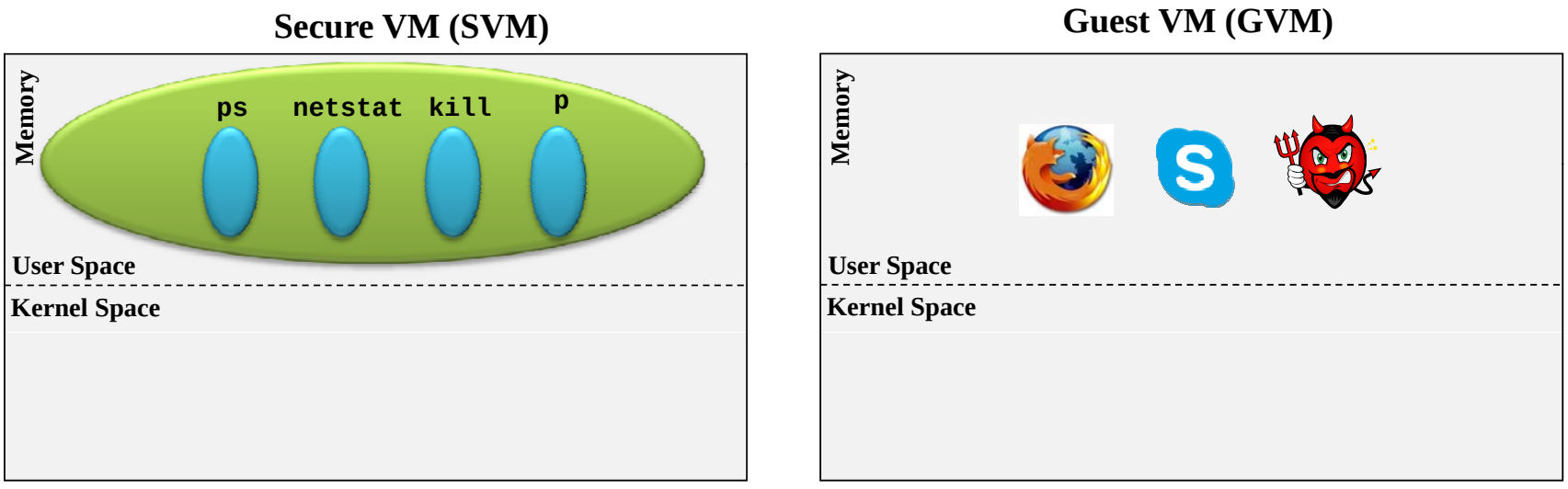
Two **Binary Code Reuse** Based Approaches



Two **Binary Code Reuse** Based Approaches



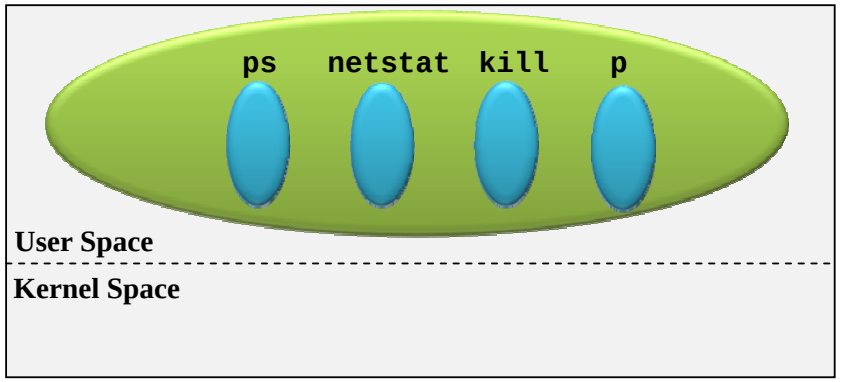
Two **Binary Code Reuse** Based Approaches



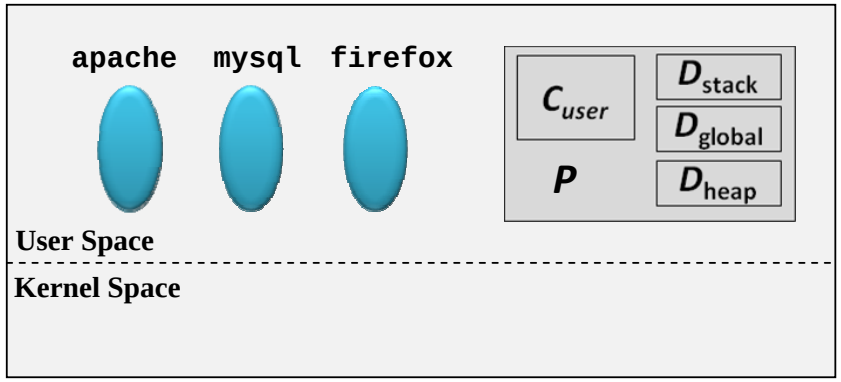
Using a trusted sibling VM to intropsect the running guest VM.

1. Redirect kernel data [SP'12, VEE'13, NDSS'14]
2. Redirect system call execution [USENIX ATC'14]

Advantages



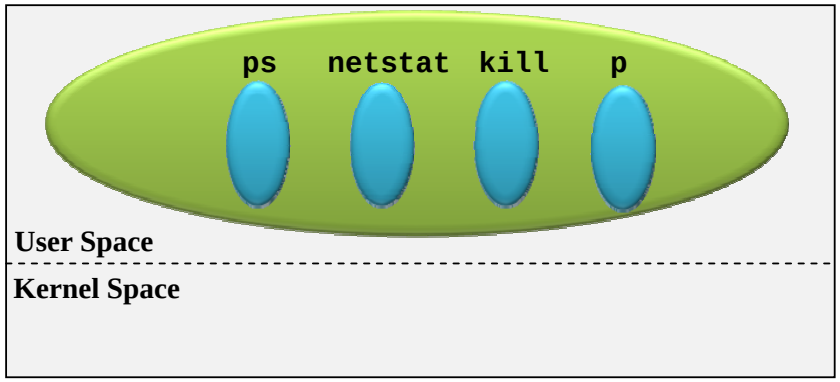
Secure VM (SVM)



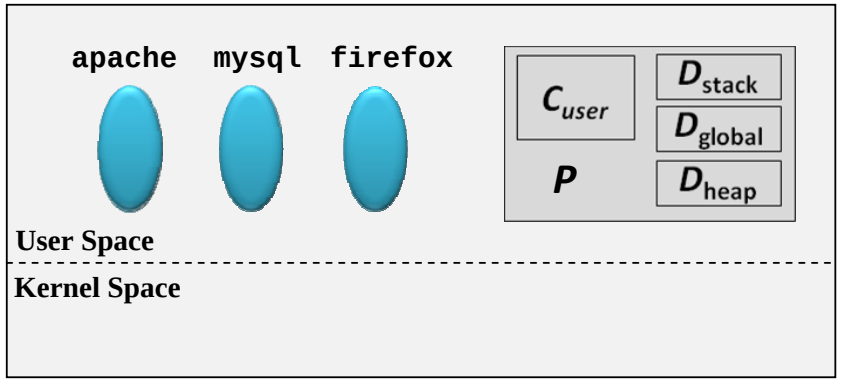
Guest VM (GVM)

- **Isolation** (SVM and GVM are isolated)
- **Trustworthiness** (trust code is running in secure VM)

Advantages



Secure VM (SVM)

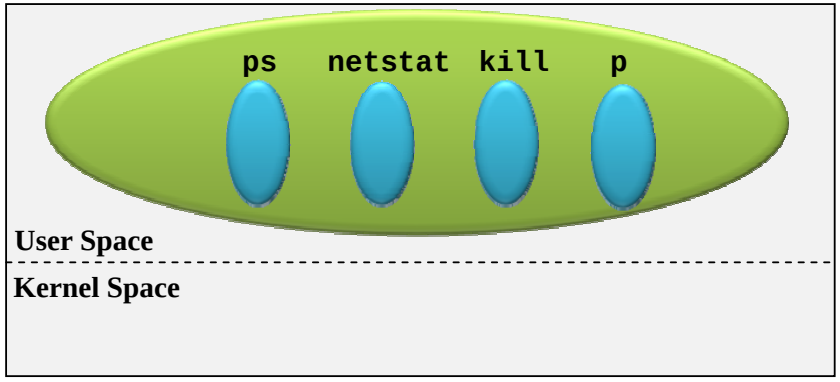


Guest VM (GVM)

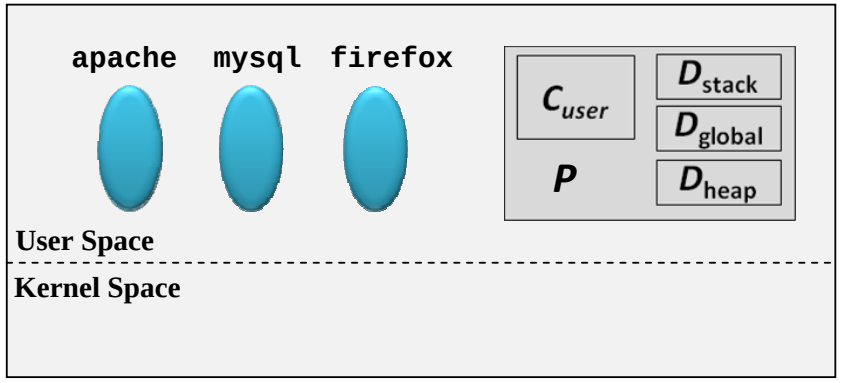
- **Isolation** (SVM and GVM are isolated)
- **Trustworthiness** (trust code is running in secure VM)

- **Automation** (no need to develop introspection utilities)
- **Transparency** (programmers write native program in SVM)

Approach-I: Redirect Kernel Data of Interest



Secure VM (SVM)

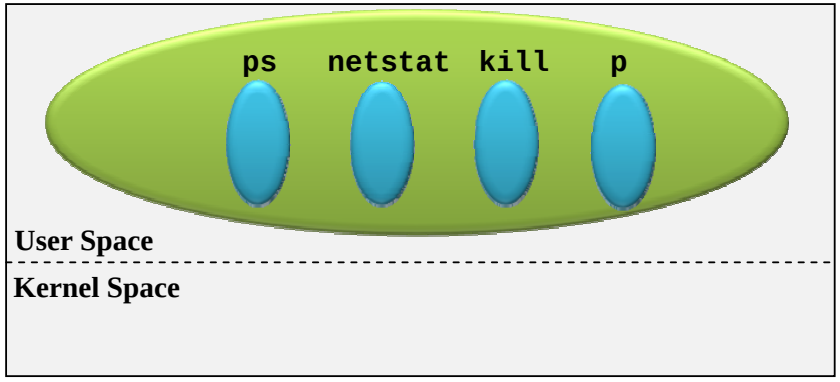


Guest VM (GVM)

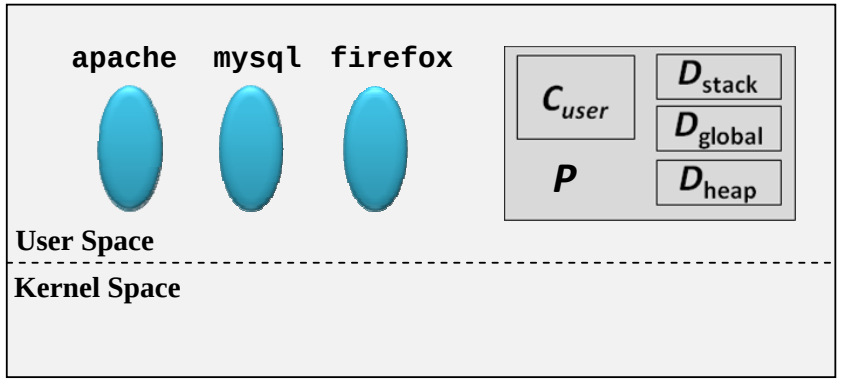
Goal

- Using a native inspection software *P* (e.g., `ps`) from SVM to transparently monitor/manage the state of GVM

Approach-I: Redirect Kernel Data of Interest



Secure VM (SVM)



Guest VM (GVM)

Goal

- Using a native inspection software *P* (e.g., `ps`) from SVM to transparently monitor/manage the state of GVM

Key Ideas

- Reusing the process execution context of *P* in SVM
- Redirecting the data-of-interest from GVM to SVM, such that *P* still feels it inspect its own OS.

Key-Idea I: Execution Context Reusing

In-VM `getpid` Program

```
1 #include <stdio.h>
2 #include <unistd.h>
3
4 int main()
5 {
6     printf("pid=%d\n",getpid());
7     return 0;
8 }
```

```
 1 execve("./getpid",..)           = 0
 2 brk(0)                    = 0x83b8000
 3 access("/etc/ld.so.nohwcap",.) = -1
...
23 getpid()                        = 13849
...
26 write(1, "pid=13849\n", 10)     = 10
27 exit_group(0)                   = ?
```

Key-Idea I: Execution Context Reusing

In-VM `getpid` Program

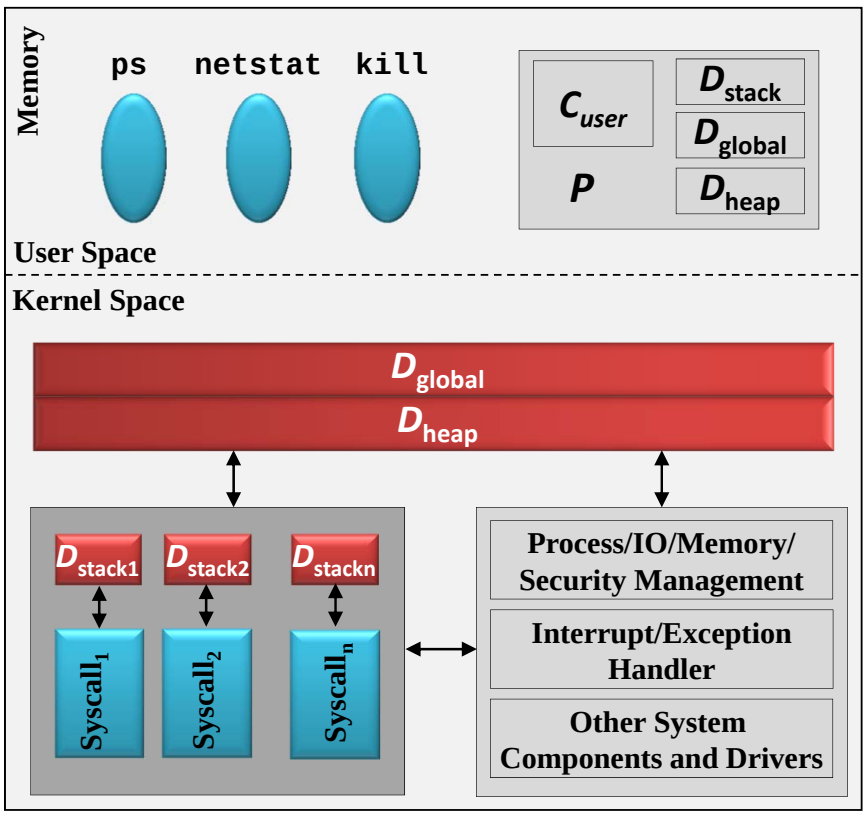
```
1 #include <stdio.h>
2 #include <unistd.h>
3
4 int main()
5 {
6     printf("pid=%d\n",getpid());
7     return 0;
8 }
```

```
 1 execve("./getpid",..)           = 0
 2 brk(0)                    = 0x83b8000
 3 access("/etc/ld.so.nohwcap",.) = -1
...
23 getpid()                        = 13849
...
26 write(1, "pid=13849\n", 10)     = 10
27 exit_group(0)                   = ?
```

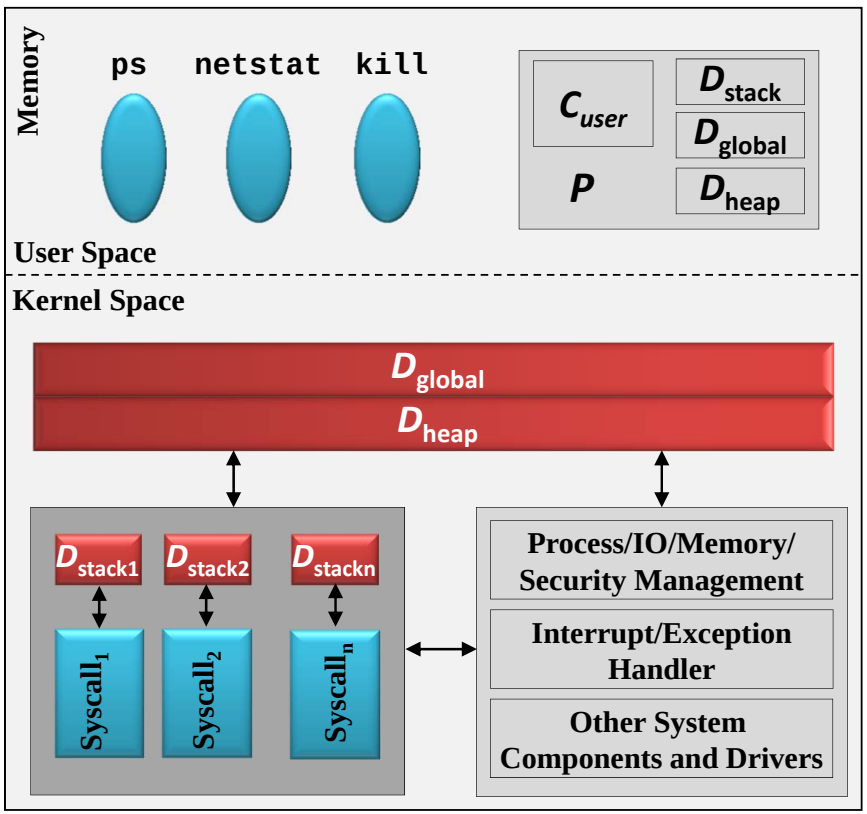
Insight

- Reuse the execution context of a native process in SVM.
- When syscall `getpid` executed, redirects the data from other VM.

Key Idea II: Data Space Travel (Data Forwarding)

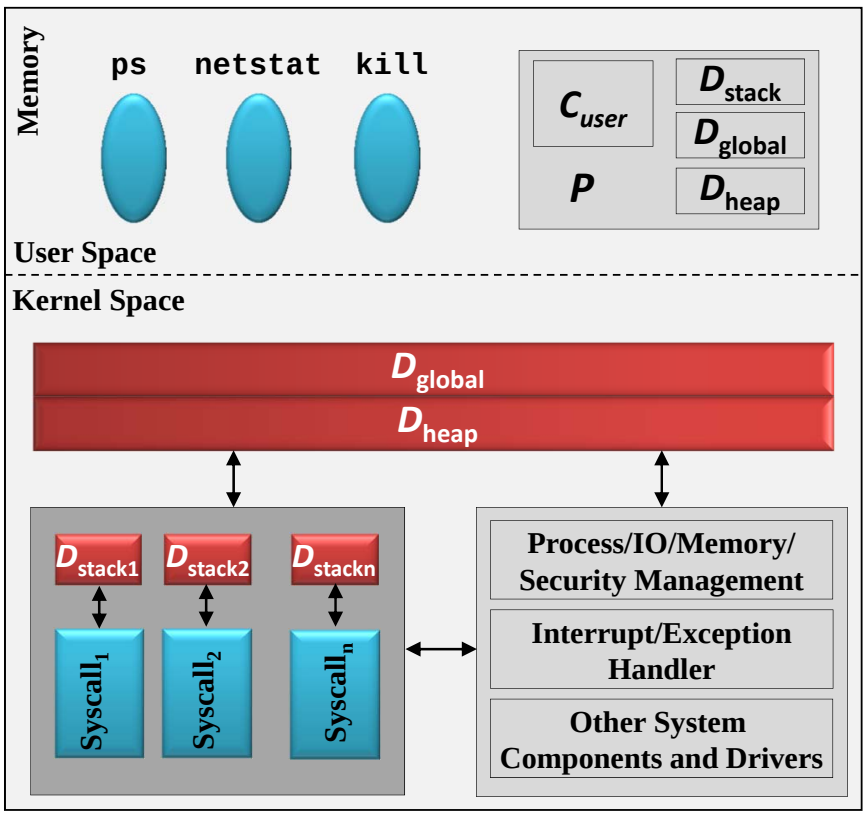


Key Idea II: Data Space Travel (Data Forwarding)



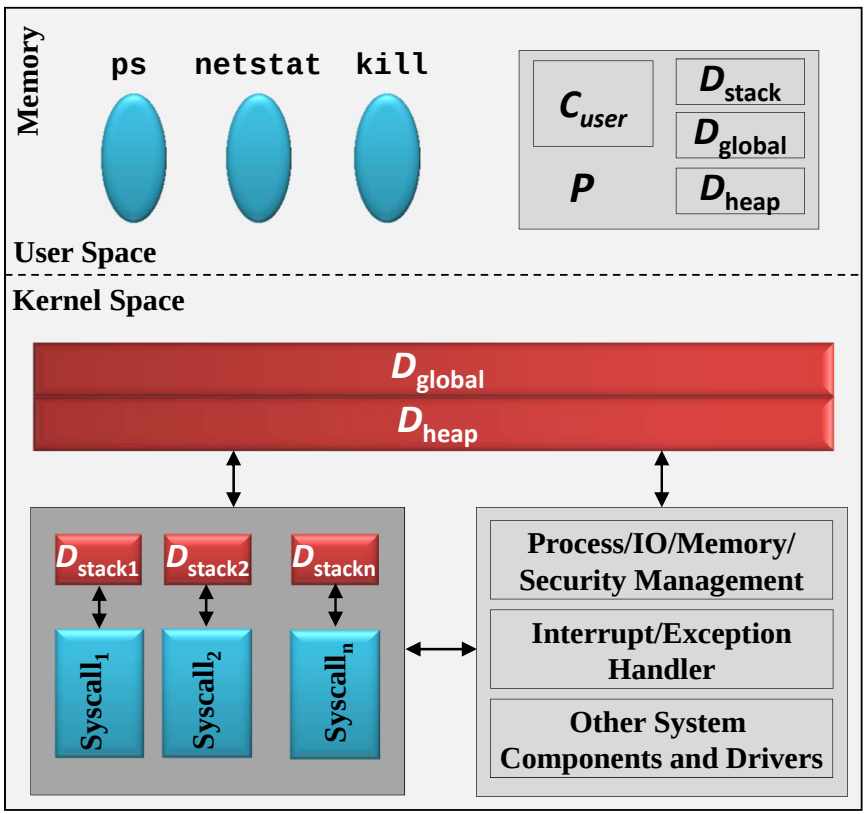
$$
\begin{aligned}
\mathcal{P} &= \mathcal{C}(\mathcal{D}) \\
&= \mathcal{C}(\mathcal{D}_{user}, \mathcal{D}_{kernel}) \\
&= \mathcal{C}(\{\mathcal{D}^{stack}_{user}, \mathcal{D}^{heap}_{user}, \mathcal{D}^{global}_{user}\}, \\
&\quad \{\mathcal{D}^{stack}_{kernel}, \mathcal{D}^{heap}_{kernel}, \mathcal{D}^{global}_{kernel}\})
\end{aligned}
$$

Key Idea II: Data Space Travel (Data Forwarding)



$$
\begin{aligned}
\mathcal{P} &= \mathcal{C}(\mathcal{D}) \\
&= \mathcal{C}(\mathcal{D}_{user}, \mathcal{D}_{kernel}) \\
&= \mathcal{C}(\{\mathcal{D}^{stack}_{user}, \mathcal{D}^{heap}_{user}, \mathcal{D}^{global}_{user}\}, \\
&\quad \{\mathcal{D}^{stack}_{kernel}, \mathcal{D}^{heap}_{kernel}, \mathcal{D}^{global}_{kernel}\}) \\
\mathcal{P}_{out} &= \mathcal{C}_{out}(\mathcal{D}_{user}, \mathcal{D}_{kernel}) \\
&= \mathcal{C}_{in}(\mathcal{D}_{user}, \mathcal{D}'_{kernel}) \\
&= \mathcal{C}_{in}(\mathcal{D}_{user}, \\
&\quad \{\mathcal{D}^{stack}_{kernel}, \mathcal{D}'^{heap}_{kernel}, \mathcal{D}'^{global}_{kernel}\})
\end{aligned} \tag{1}
$$

Key Idea II: Data Space Travel (Data Forwarding)



$$
\begin{aligned}
\mathcal{P} &= \mathcal{C}(\mathcal{D}) \\
&= \mathcal{C}(\mathcal{D}_{user}, \mathcal{D}_{kernel}) \\
&= \mathcal{C}(\{\mathcal{D}^{stack}_{user}, \mathcal{D}^{heap}_{user}, \mathcal{D}^{global}_{user}\}, \\
&\quad \{\mathcal{D}^{stack}_{kernel}, \mathcal{D}^{heap}_{kernel}, \mathcal{D}^{global}_{kernel}\}) \\
\mathcal{P}_{out} &= \mathcal{C}_{out}(\mathcal{D}_{user}, \mathcal{D}_{kernel}) \\
&= \mathcal{C}_{in}(\mathcal{D}_{user}, \mathcal{D}'_{kernel}) \\
&= \mathcal{C}_{in}(\mathcal{D}_{user}, \\
&\quad \{\mathcal{D}^{stack}_{kernel}, \mathcal{D}'^{heap}_{kernel}, \mathcal{D}'^{global}_{kernel}\})
\end{aligned} \tag{1}
$$

e.g., `mov eax, [0x1c0eff08]`

Why it works? (e.g., sys_getpid)

```
<sys_getpid>:
<task_tgid_vnr>:
1:  c10583e0: push   %ebp
2:  c10583e1: mov    %esp,%ebp
3:  c10583e3: push   %ebx
4:  c10583e4: sub    $0x14,%esp

// Accessing Global Variable: struct task_strut current_task
5:  c10583e7: mov    %fs:0xc17f34cc,%ebx
        c10583ea: R_386_32   current_task
```

(a)

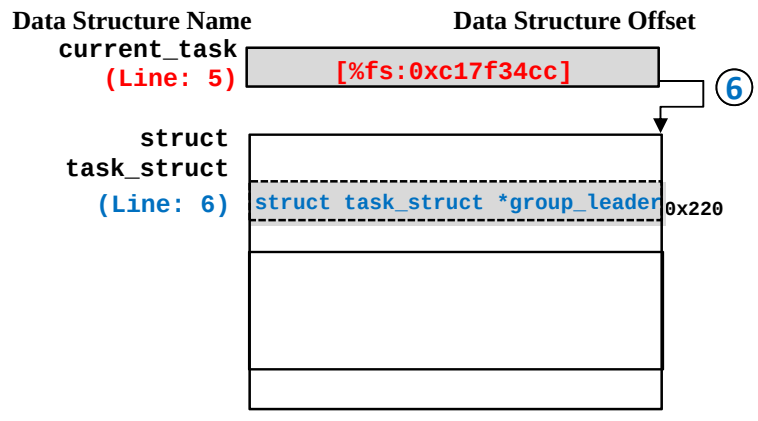
Data Structure Name	Data Structure Offset
current_task (Line: 5)	[%fs:0xc17f34cc]

(b)

Why it works? (e.g., sys_getpid)

```
<sys_getpid>:
<task_tgid_vnr>:
1:  c10583e0: push   %ebp
2:  c10583e1: mov    %esp,%ebp
3:  c10583e3: push   %ebx
4:  c10583e4: sub    $0x14,%esp

// Accessing Global Variable: struct task_strut current_task
5:  c10583e7: mov    %fs:0xc17f34cc,%ebx
        c10583ea: R_386_32   current_task

// Accessing struct task_struct: current_task->group_leader
6:  c10583fe: mov    0x220(%ebx),%eax
```

(a)

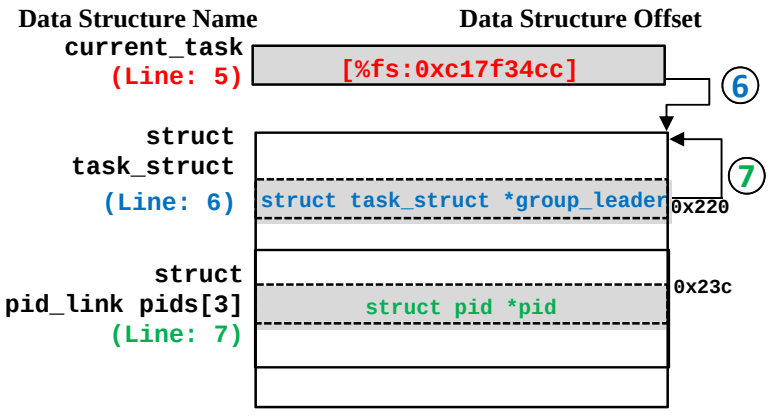


(b)

Why it works? (e.g., sys_getpid)

```
<sys_getpid>:
<task_tgid_vnr>:
1:  c10583e0: push   %ebp
2:  c10583e1: mov    %esp,%ebp
3:  c10583e3: push   %ebx
4:  c10583e4: sub    $0x14,%esp

// Accessing Global Variable: struct task_strut current_task
5:  c10583e7: mov    %fs:0xc17f34cc,%ebx
        c10583ea: R_386_32   current_task

// Accessing struct task_struct: current_task->group_leader
6:  c10583fe: mov    0x220(%ebx),%eax

// Accessing struct pid: current_task->group_leader->pids[0]->pid
7:  c1058404: mov    0x23c(%eax),%eax
```

(a)

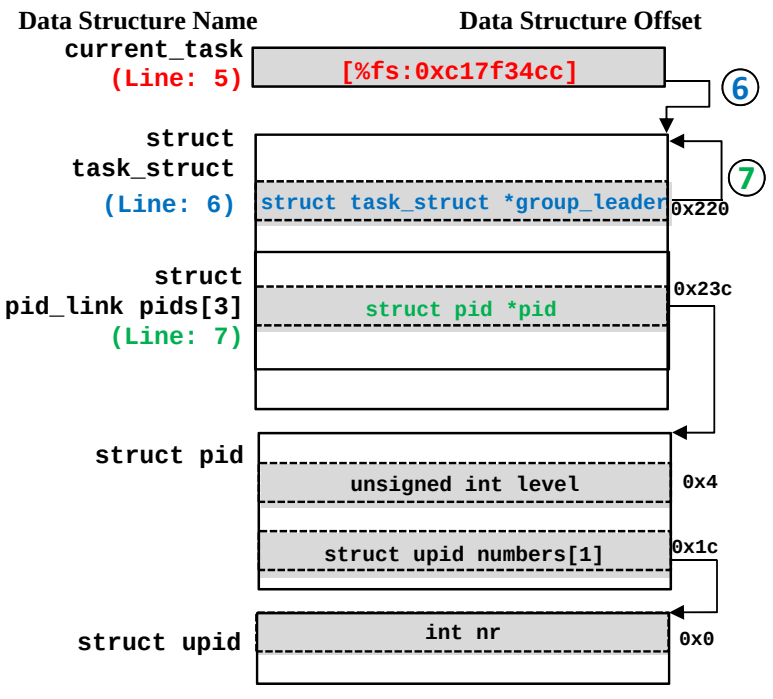


(b)

Why it works? (e.g., sys_getpid)

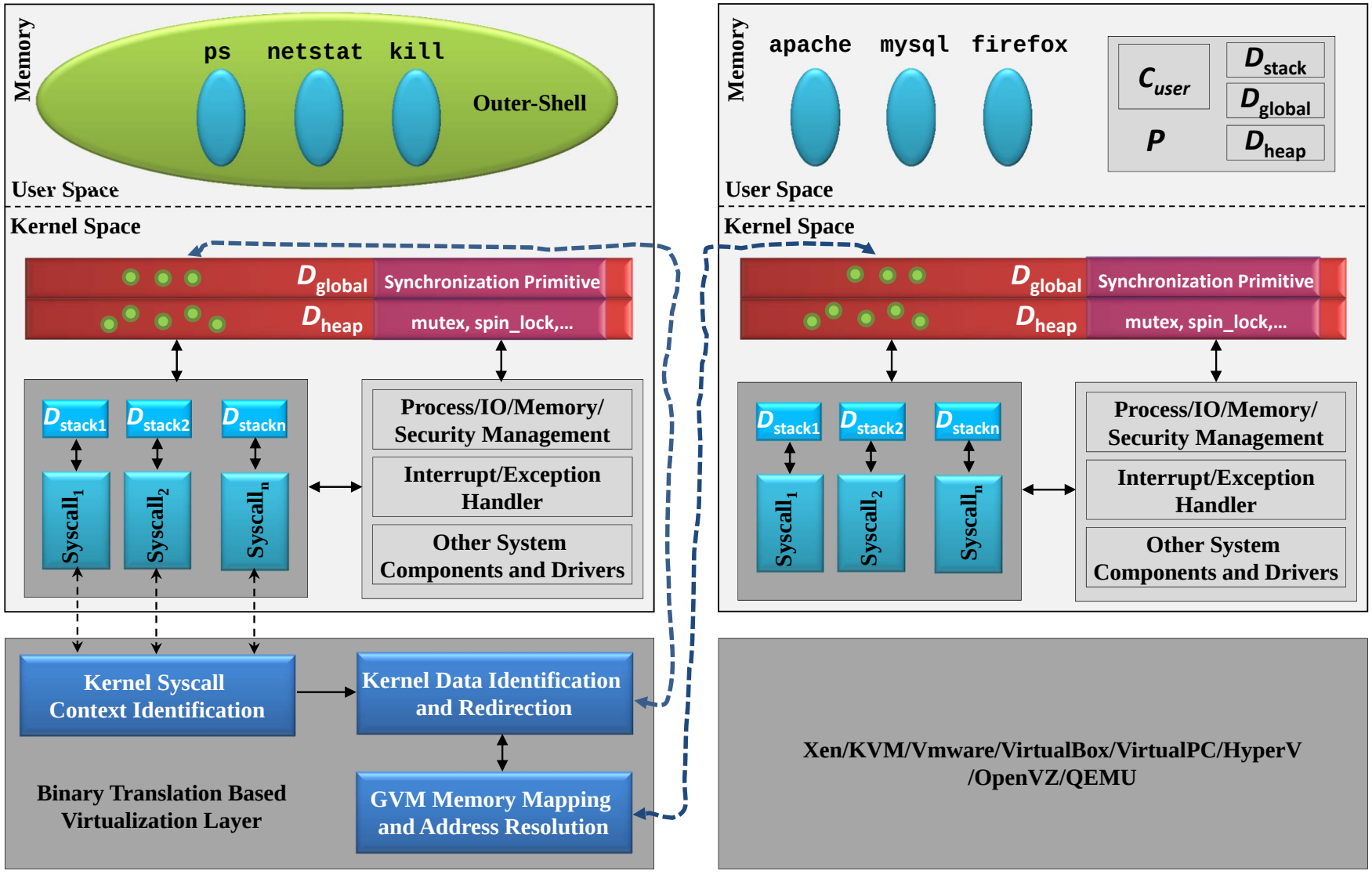
```
<sys_getpid>:
<task_tgid_vnr>:
1:  c10583e0: push   %ebp
2:  c10583e1: mov    %esp,%ebp
3:  c10583e3: push   %ebx
4:  c10583e4: sub    $0x14,%esp

// Accessing Global Variable: struct task_strut current_task
5:  c10583e7: mov    %fs:0xc17f34cc,%ebx
        c10583ea: R_386_32   current_task

// Accessing struct task_struct: current_task->group_leader
6:  c10583fe: mov    0x220(%ebx),%eax

// Accessing struct pid: current_task->group_leader->pids[0]->pid
7:  c1058404: mov    0x23c(%eax),%eax

8:  c105840a: call   c1065660 <pid_vnr>
9:  c105840f: add    $0x14,%esp
```

(a)

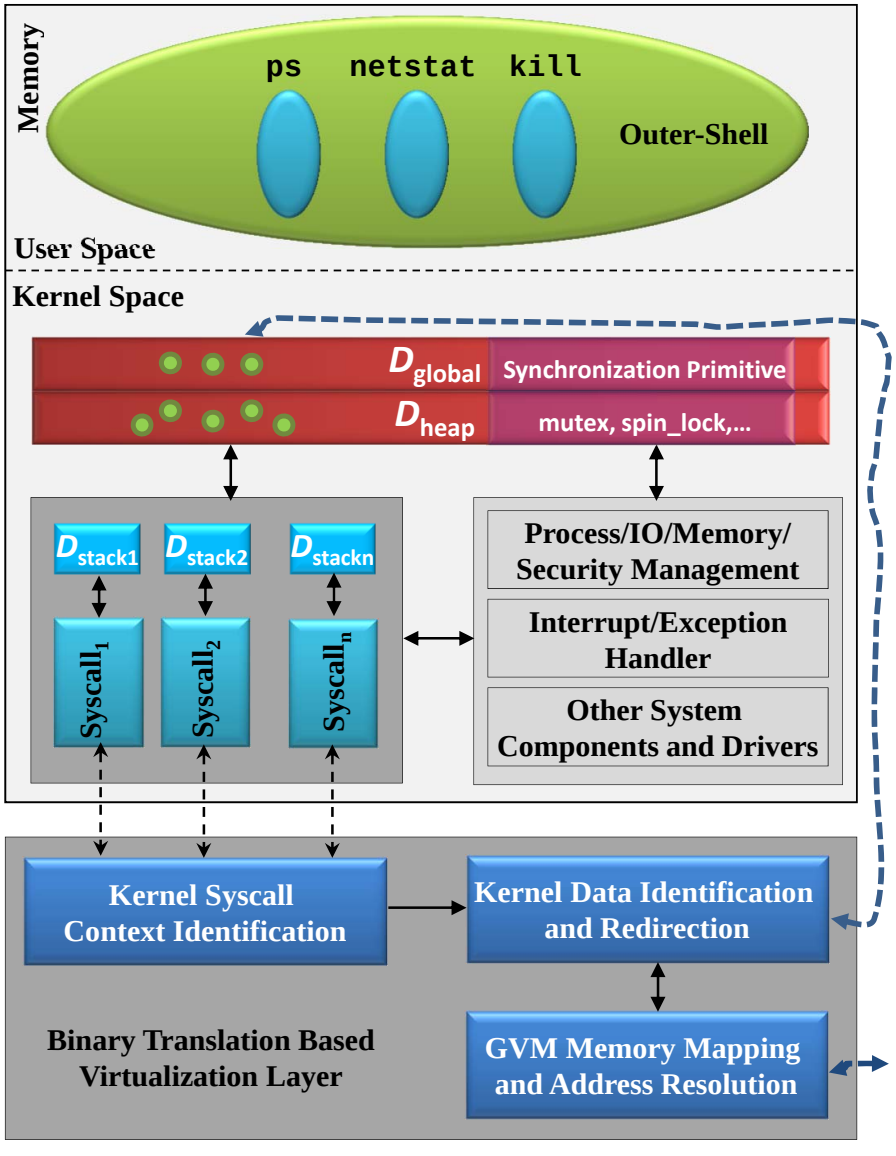


(b)

Our Architecture ([Oakland'12], [VEE'13])



Kernel Syscall Context Identification

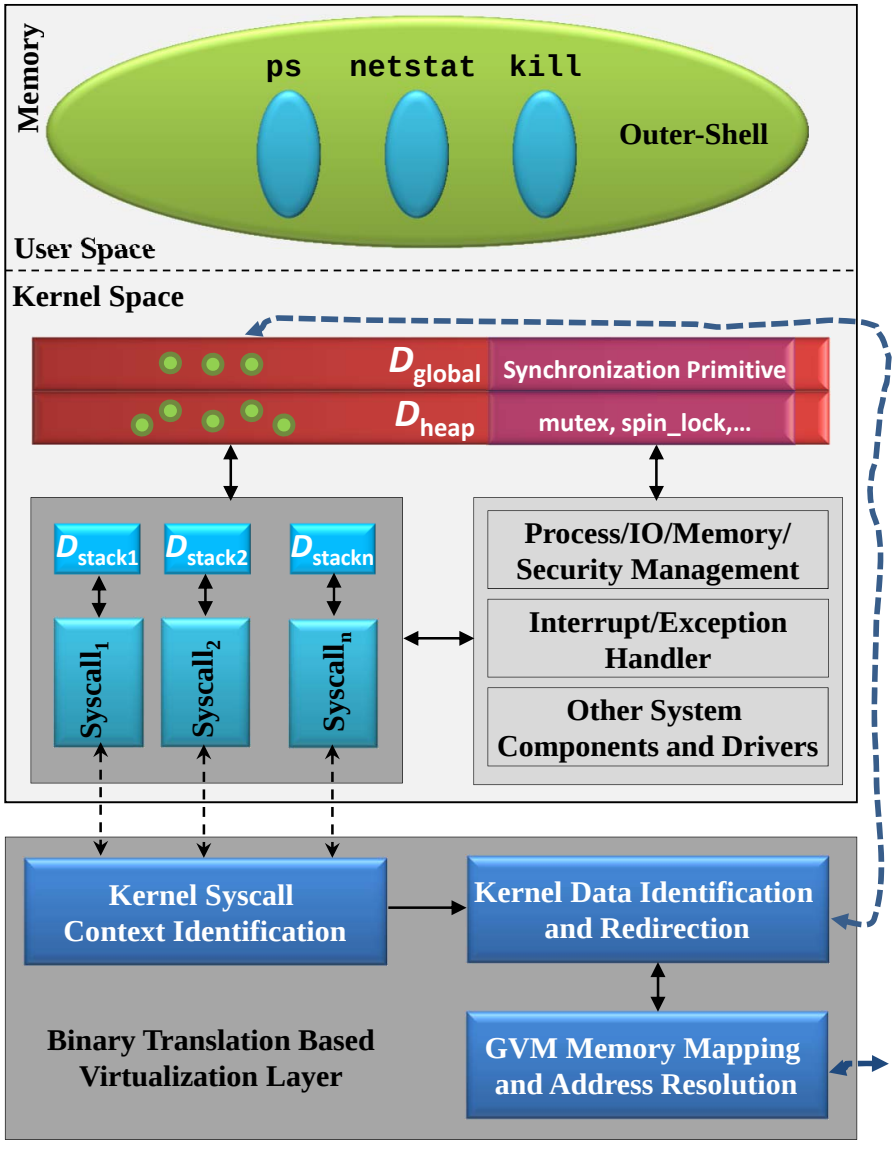


Secure VM (SVM)

Syscall of `getpid` process

```
1  execve("./getpid", ...)= 0
2  brk(0)          = 0x83b8000
3  access("/etc/ld.so.nohwcap", ...)
...
23 getpid()= 13849
24 fstat(3, st_mode=S_IFREG ...)= 0
25 mmap2(NULL, ...)= 0x4001f000
26 write(1, "pid=13849",10) = 10
27 exit_group(0)
```

Kernel Data Identification and Redirection



Secure VM (SVM)

The Algorithm

```
1: DynamicBinaryInstrumentation(i):
2:   if SysCallExecContext(s):
3:     if SysCallRedirectable(s):
4:       RedirectableDataTracking(i);
5:       for α in MemoryAddress(i):
6:         if DataRead(α):
7:           PA(α) ← V2P(α)
8:           Load(PA(α))
9:         else:
10:          if Configuration:
11:            Store(PA(α))
12:          else: //Introspection
13:            COW-Store(PA(α))
```

Taint Analysis (Data Flow Tracking): An Example

- movl $0x8048118,%eax

mov %eax, 0x4(%esp)

movl $0x8049128,(%esp)

call 0x80480e0 <strcpy>

mov $0x14, %eax

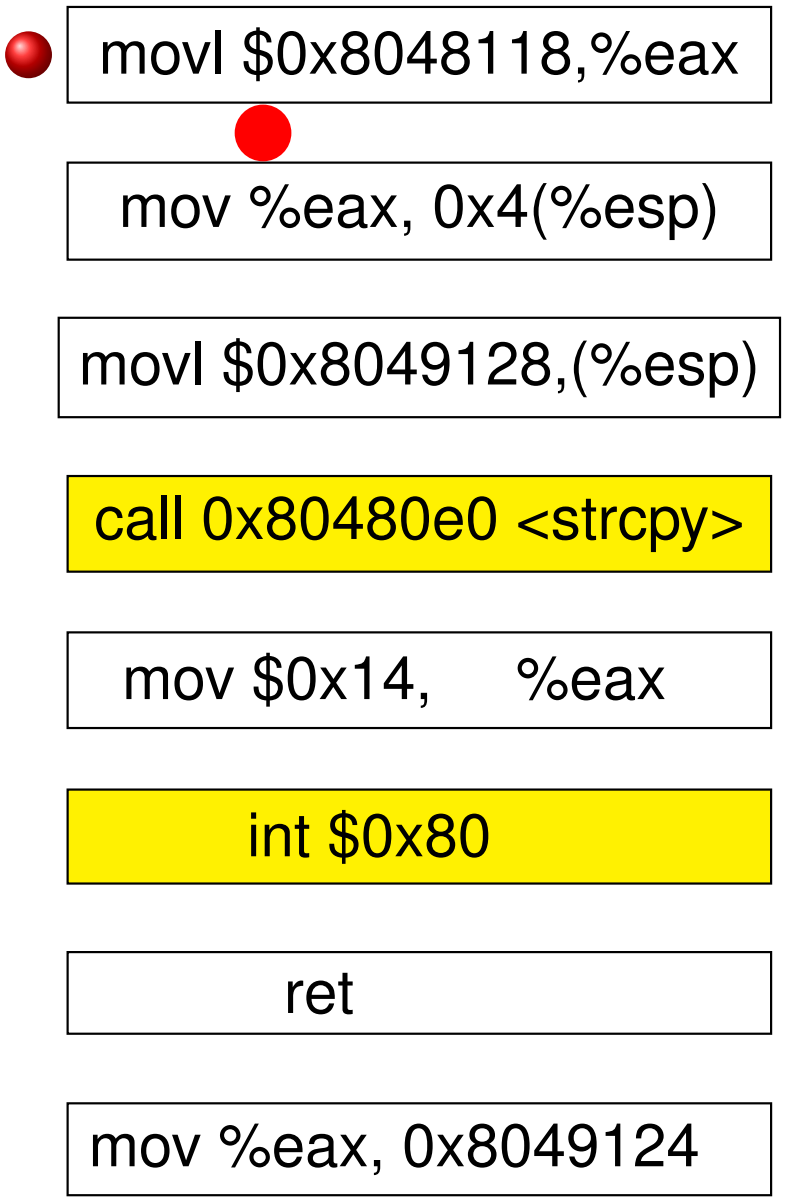
int $0x80

ret

mov %eax, 0x8049124

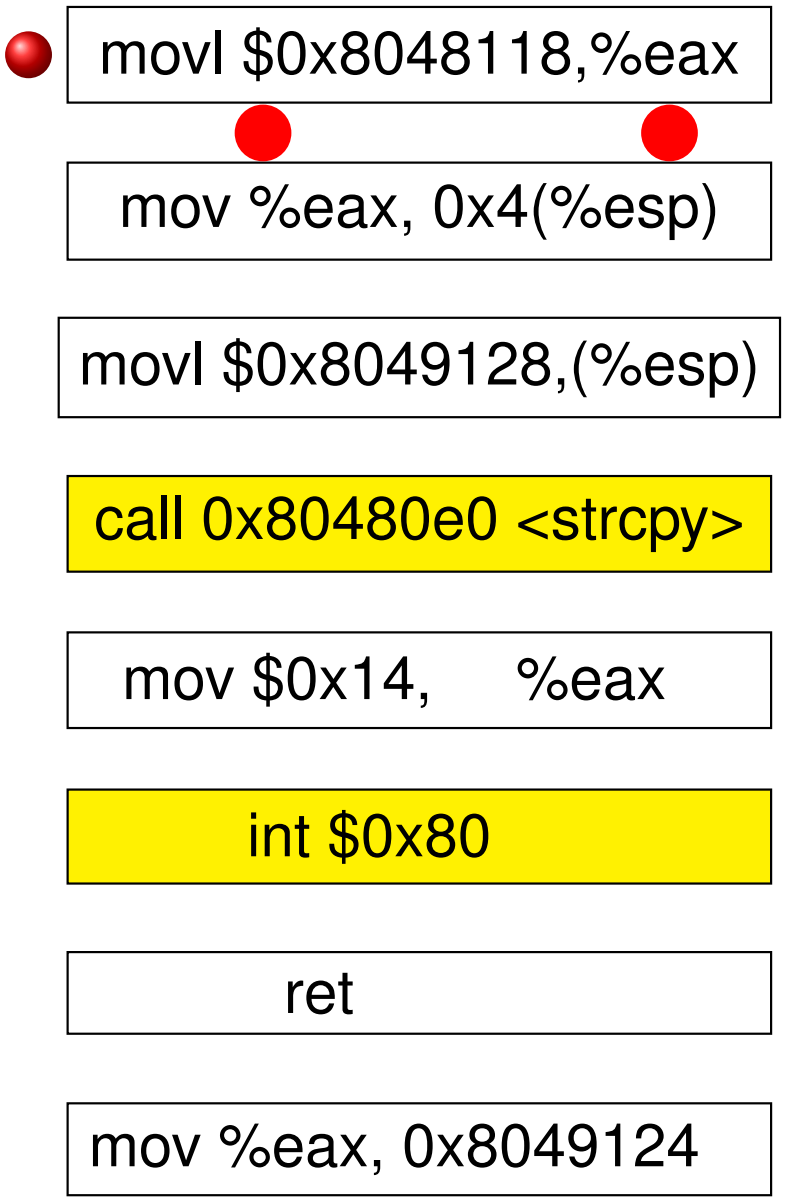
Mem,Reg	Tag	Type

Taint Analysis (Data Flow Tracking): An Example



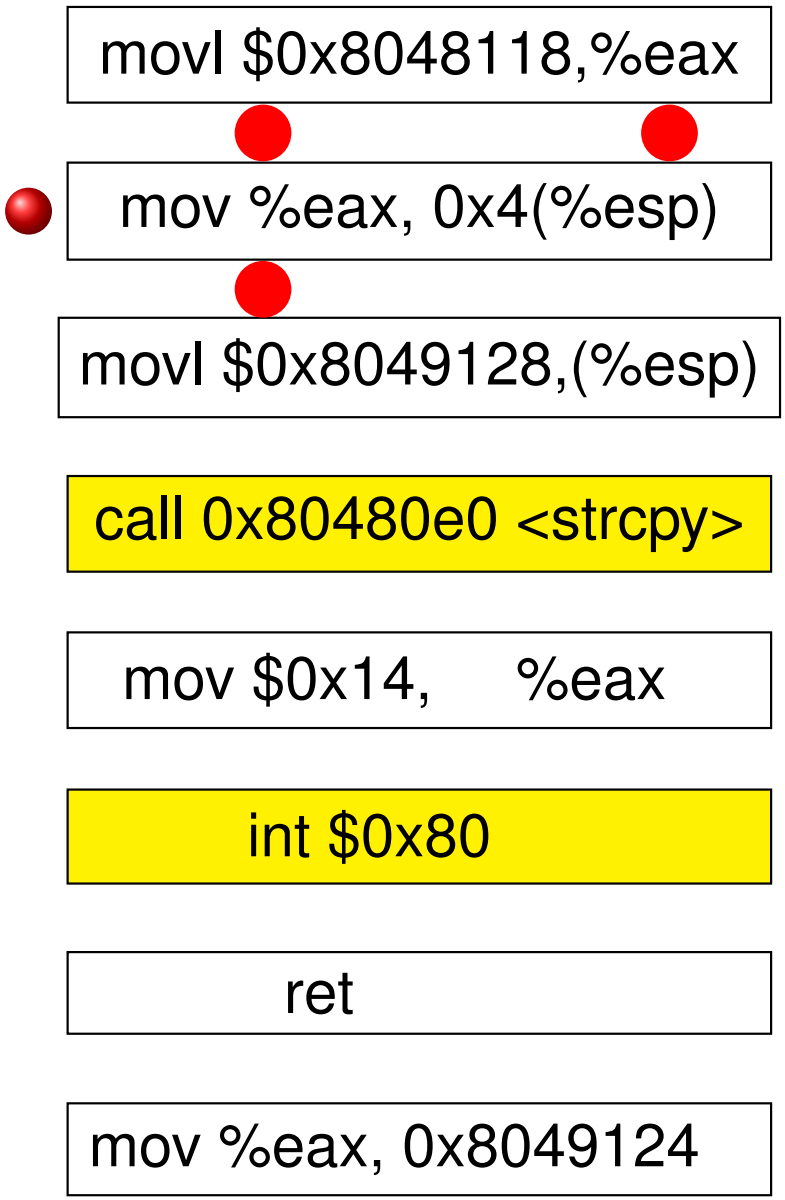
Mem,Reg	Tag	Type
0x8048118	●	N/A

Taint Analysis (Data Flow Tracking): An Example



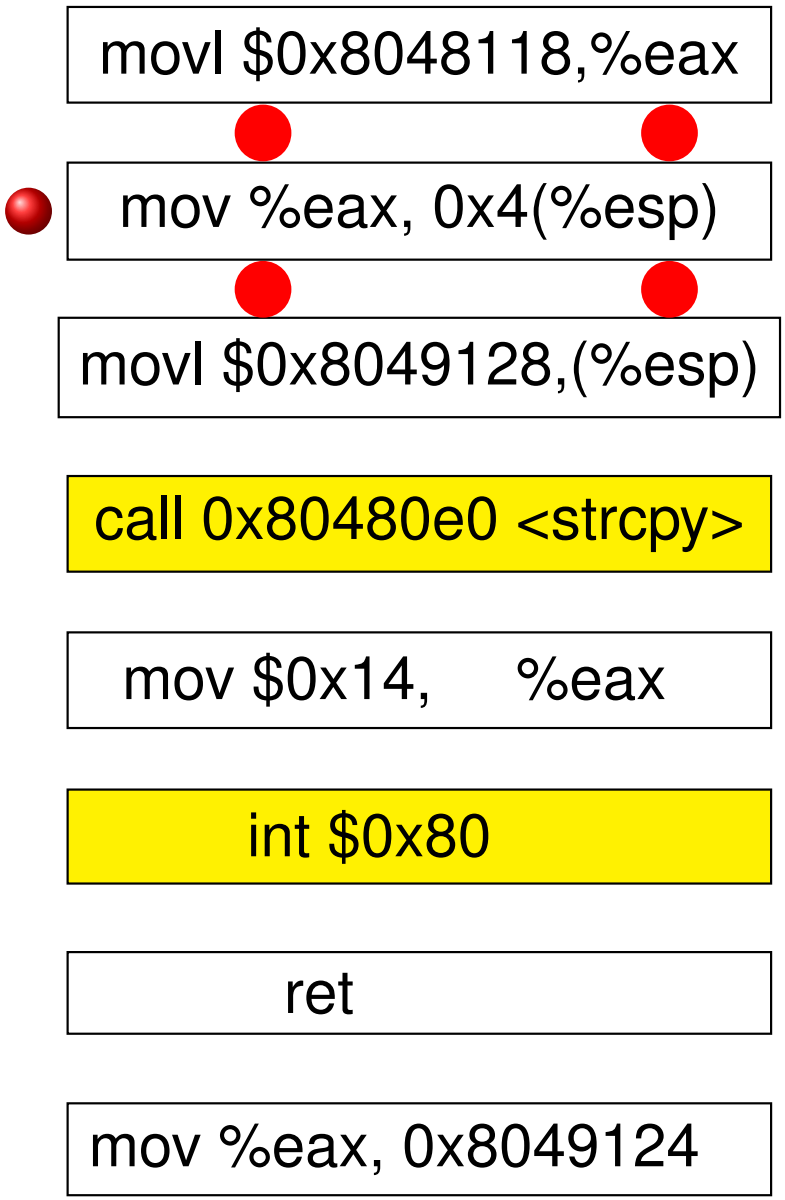
Mem,Reg	Tag	Type
0x8048118	●	N/A
eax	●	

Taint Analysis (Data Flow Tracking): An Example



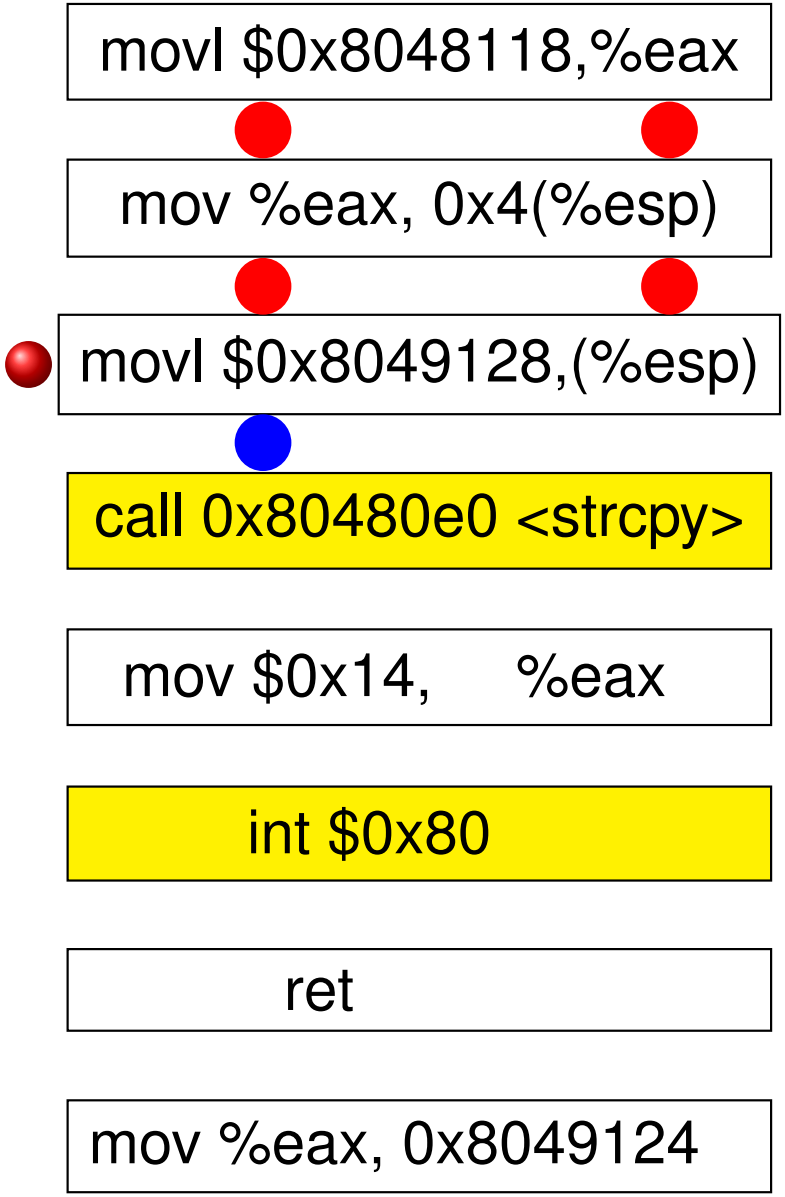
Mem,Reg	Tag	Type
0x8048118	●	N/A
eax	●	

Taint Analysis (Data Flow Tracking): An Example



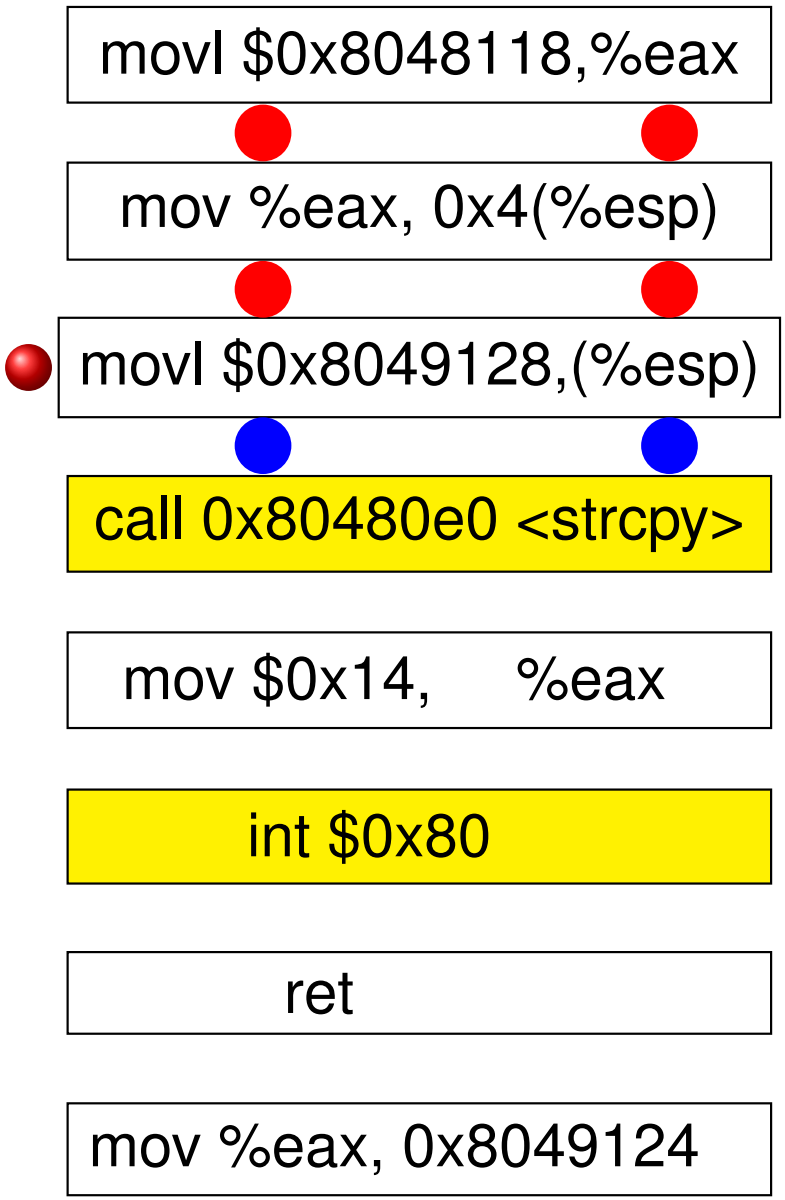
Mem,Reg	Tag	Type
0x8048118	●	N/A
eax	●	
0x4(%esp)	●	

Taint Analysis (Data Flow Tracking): An Example



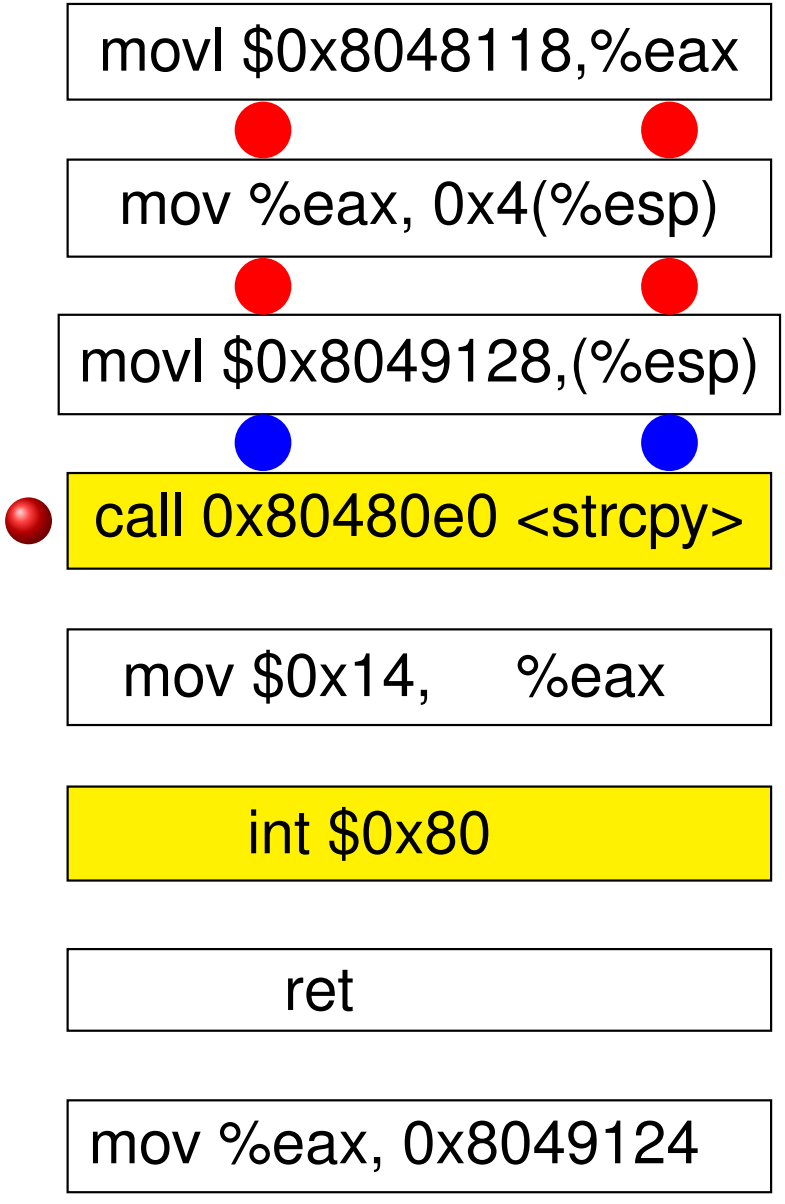
Mem,Reg	Tag	Type
0x8048118	●	N/A
eax	●	
0x4(%esp)	●	
0x8049128	●	N/A

Taint Analysis (Data Flow Tracking): An Example



Mem,Reg	Tag	Type
0x8048118	●	N/A
eax	●	
0x4(%esp)	●	
0x8049128	●	N/A
(%esp)	●	

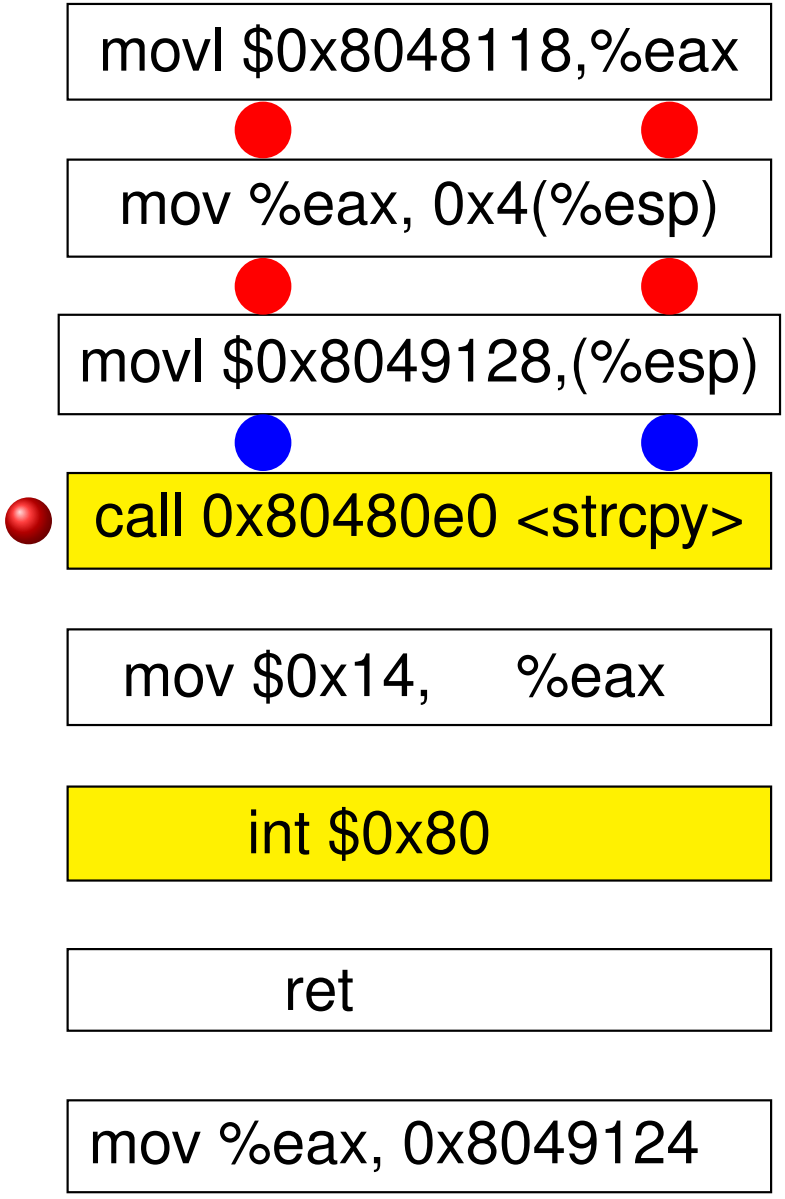
Taint Analysis (Data Flow Tracking): An Example



Mem,Reg	Tag	Type
0x8048118	●	N/A
eax	●	
0x4(%esp)	●	
0x8049128	●	N/A
(%esp)	●	

(esp+4) → char*
(esp) → char*
strcpy(char*, char*)

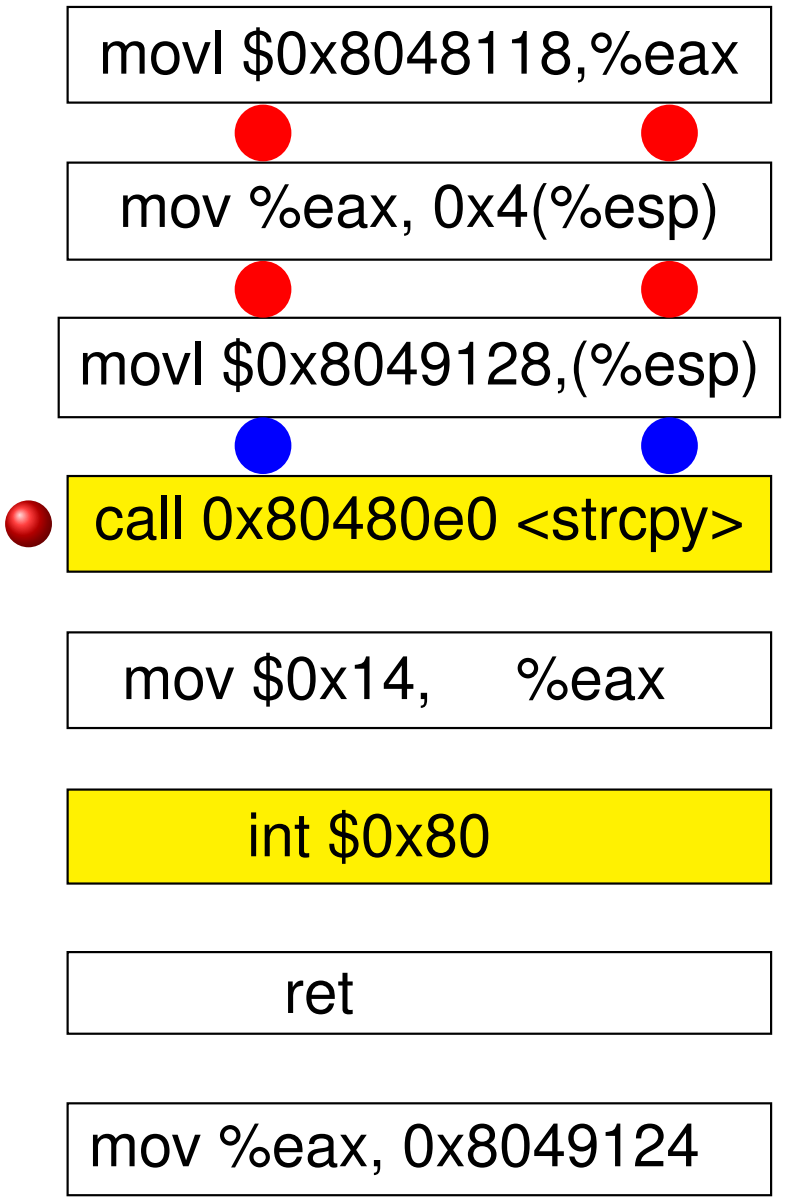
Taint Analysis (Data Flow Tracking): An Example



Mem,Reg	Tag	Type
0x8048118	●	N/A
eax	●	
0x4(%esp)	●	char*
0x8049128	●	N/A
(%esp)	●	char*

(esp+4) → char*
(esp) → char*
strcpy(char*, char*)

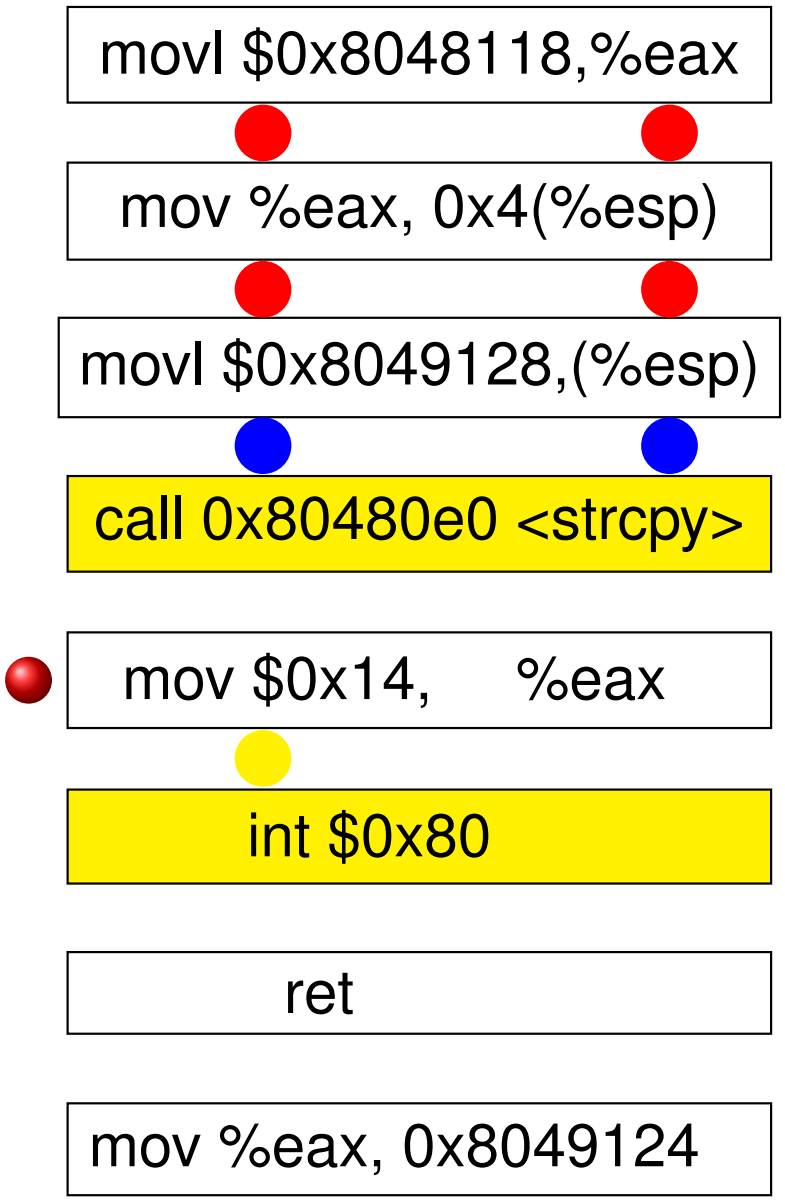
Taint Analysis (Data Flow Tracking): An Example



Mem,Reg	Tag	Type
0x8048118	●	char*
eax	●	char*
0x4(%esp)	●	char*
0x8049128	●	char*
(%esp)	●	char*

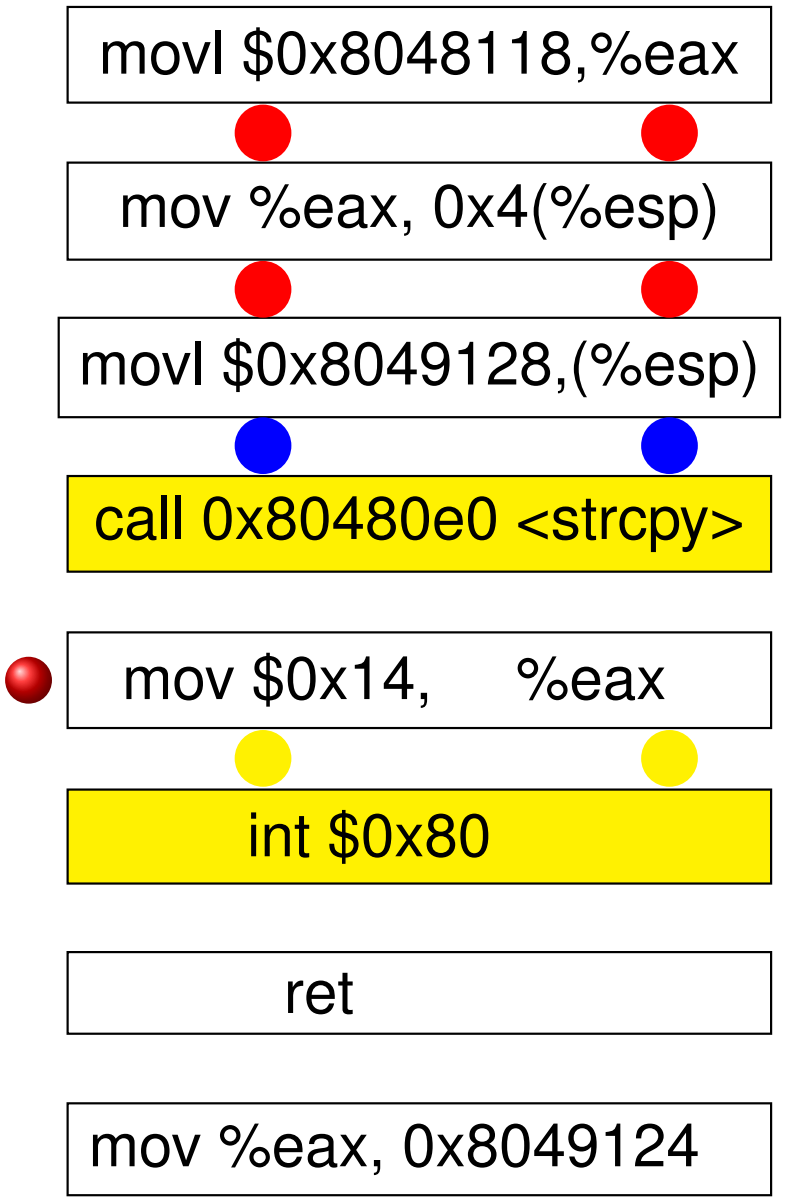
(esp+4) $\rightarrow$ char*
(esp) $\rightarrow$ char*
strcpy(char*, char*)

Taint Analysis (Data Flow Tracking): An Example



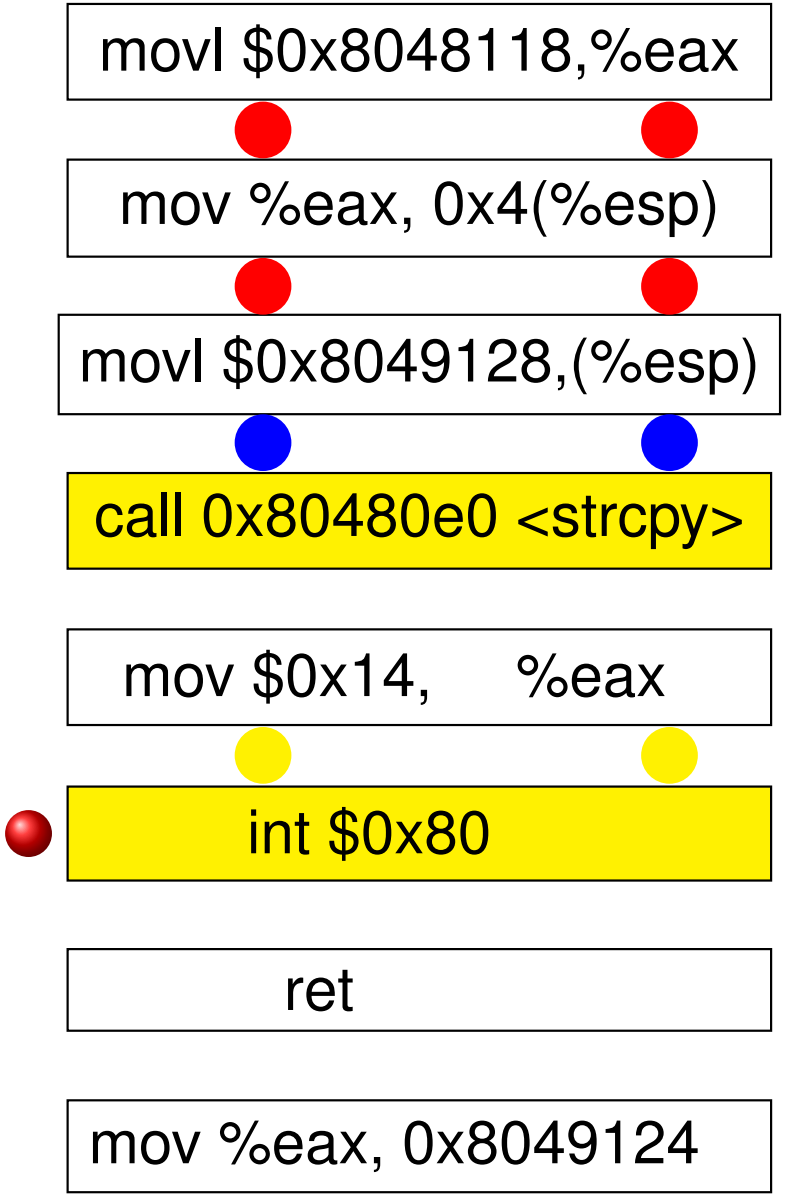
Mem,Reg	Tag	Type
0x8048118		char*
eax		char*
0x4(%esp)		char*
0x8049128		char*
(%esp)		char*

Taint Analysis (Data Flow Tracking): An Example



Mem,Reg	Tag	Type
0x8048118	●	char*
eax	●	imm_t
0x4(%esp)	●	char*
0x8049128	●	char*
(%esp)	●	char*

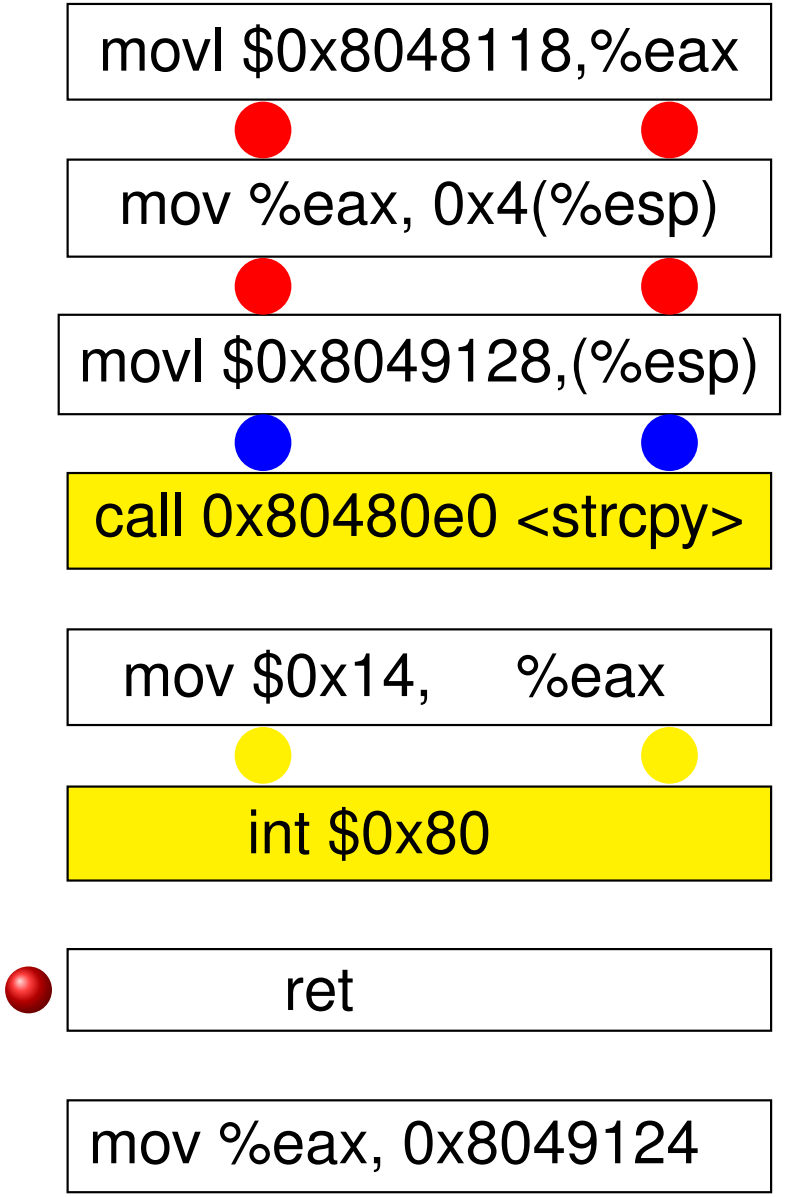
Taint Analysis (Data Flow Tracking): An Example



Mem,Reg	Tag	Type
0x8048118	●	char*
eax	●	imm_t
0x4(%esp)	●	char*
0x8049128	●	char*
(%esp)	●	char*

getpid eax → pid_t

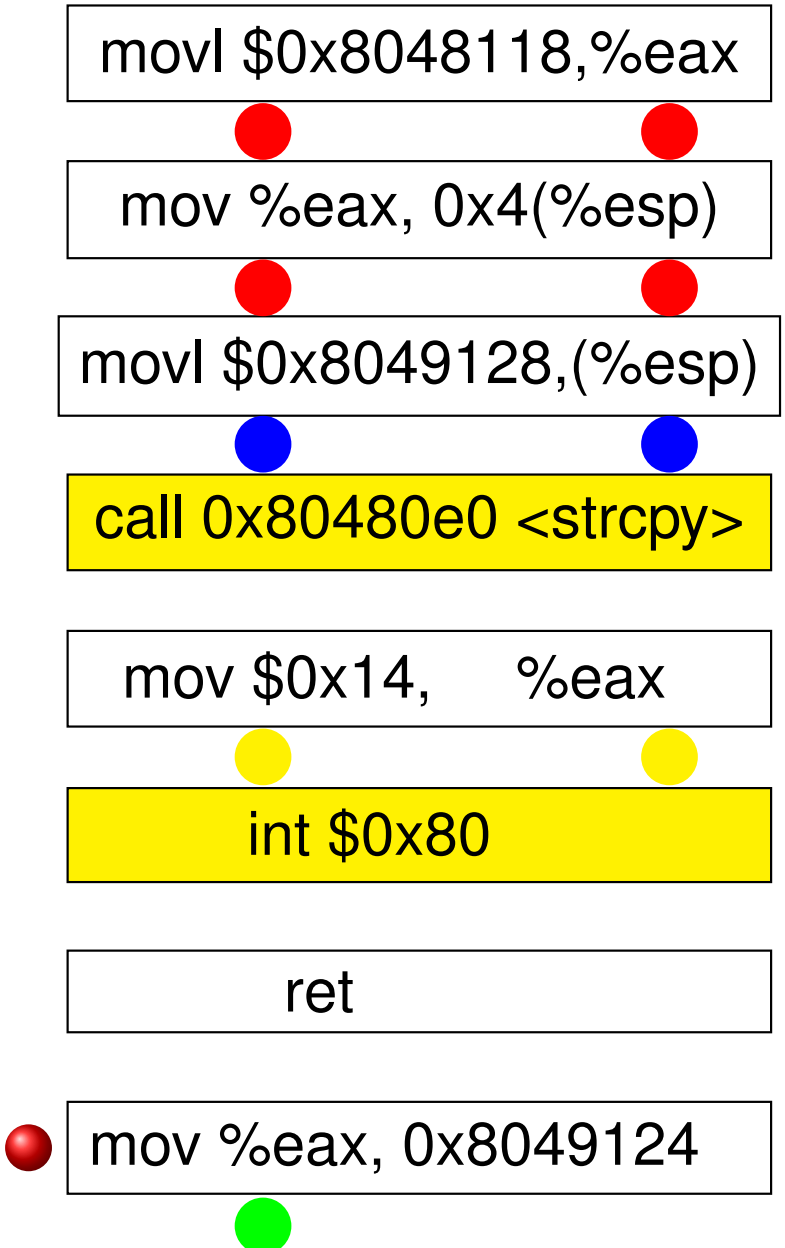
Taint Analysis (Data Flow Tracking): An Example



Mem,Reg	Tag	Type
0x8048118	●	char*
eax	●	pid_t
0x4(%esp)	●	char*
0x8049128	●	char*
(%esp)	●	char*

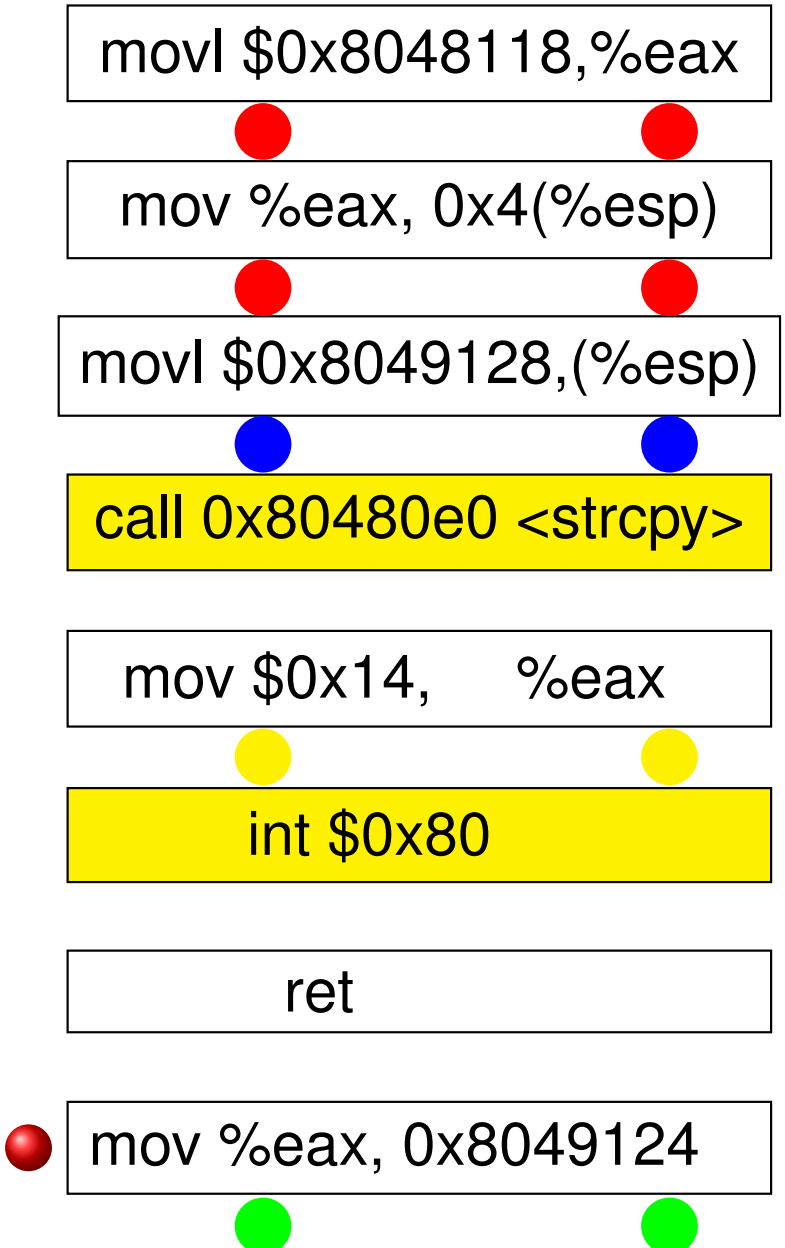
getpid eax → pid_t

Taint Analysis (Data Flow Tracking): An Example



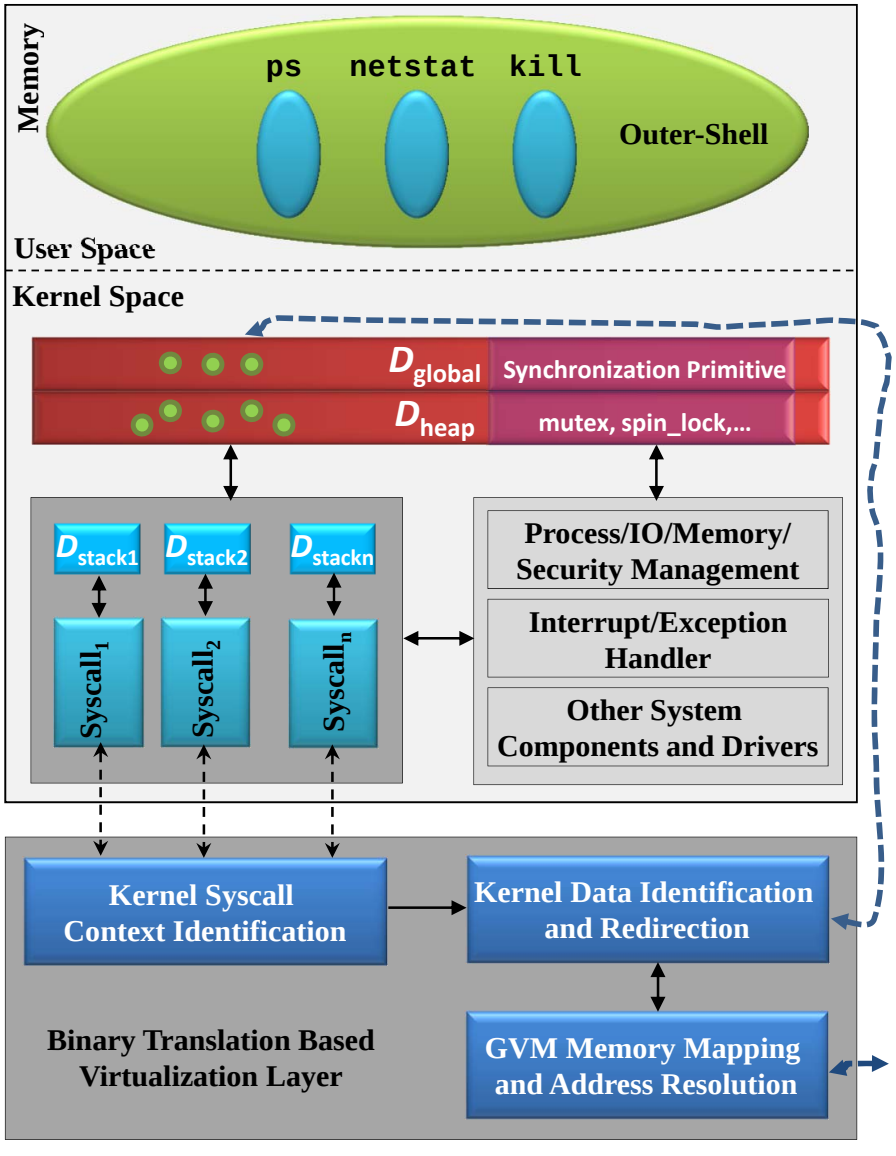
Mem,Reg	Tag	Type
0x8048118		char*
eax		pid_t
0x4(%esp)		char*
0x8049128		char*
(%esp)		char*

Taint Analysis (Data Flow Tracking): An Example

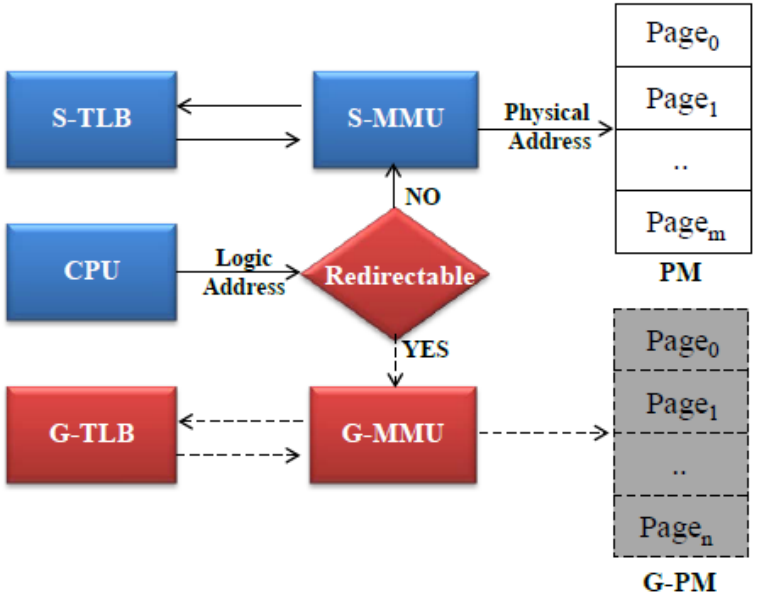


Mem,Reg	Tag	Type
0x8048118	●	char*
eax	●	pid_t
0x4(%esp)	●	char*
0x8049128	●	char*
(%esp)	●	char*
0x8049124	●	pid_t

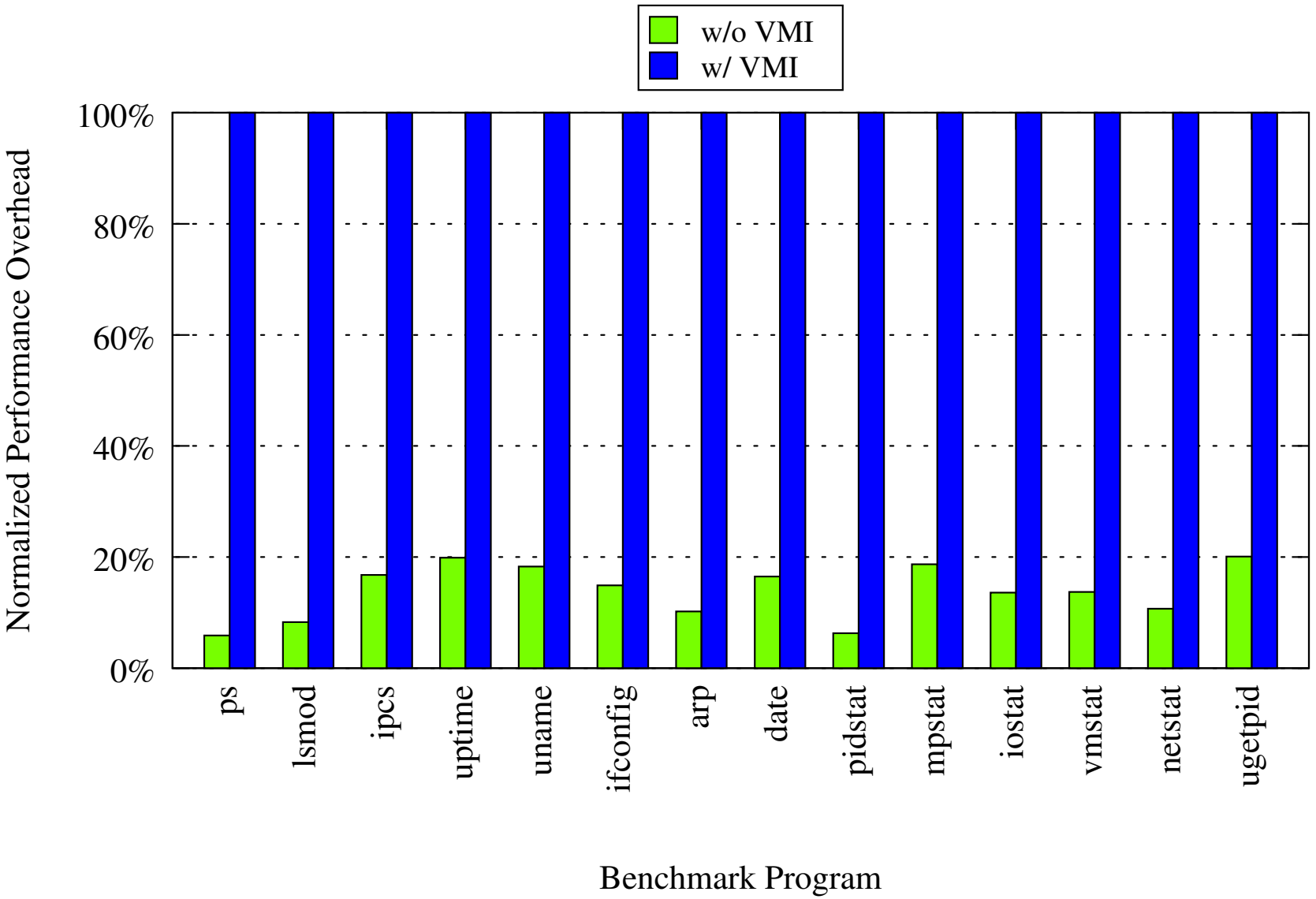
GVM Memory Mapping and Address Resolution



Secure VM (SVM)



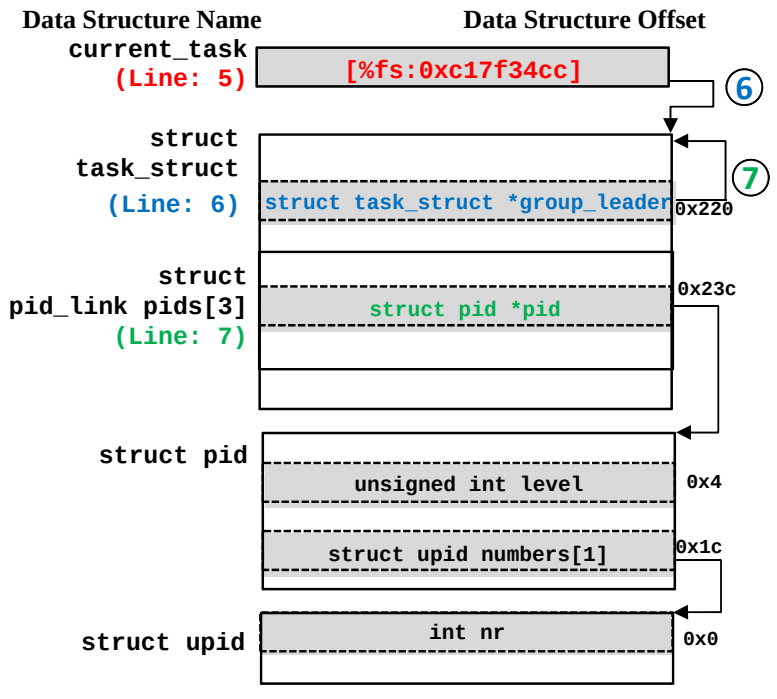
Experimental Result (Very SLOW)



Can we do it faster? Recall `sys_getpid` example

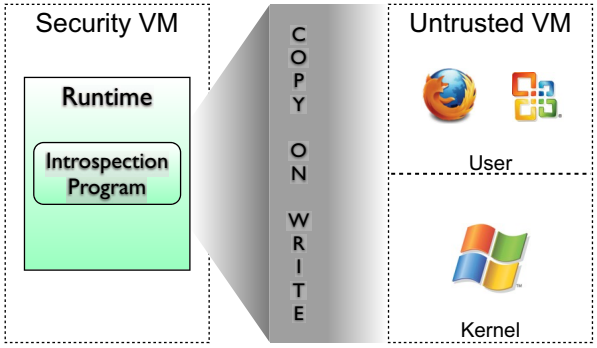
```
<sys_getpid>:
<task_tgid_vnr>:
1:  c10583e0: push   %ebp
2:  c10583e1: mov    %esp,%ebp
3:  c10583e3: push   %ebx
4:  c10583e4: sub    $0x14,%esp

// Accessing Global Variable: struct task_strut current_task
5:  c10583e7: mov    %fs:0xc17f34cc,%ebx
        c10583ea: R_386_32   current_task

// Accessing struct task_struct: current_task->group_leader
6:  c10583fe: mov    0x220(%ebx),%eax

// Accessing struct pid: current_task->group_leader->pids[0]->pid
7:  c1058404: mov    0x23c(%eax),%eax

8:  c105840a: call   c1065660 <pid_vnr>
9:  c105840f: add    $0x14,%esp
```

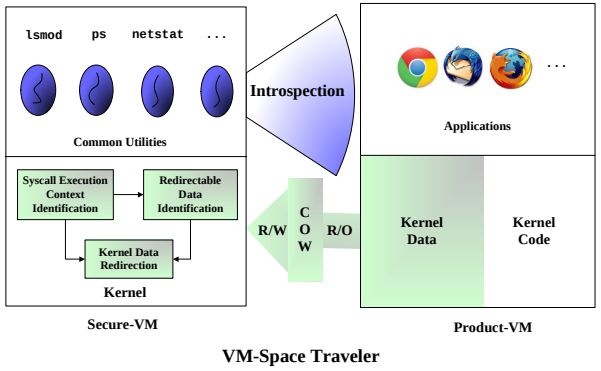
(a)



(b)

Insight: Can We Combine Offline and Online?





VM-Space Traveler

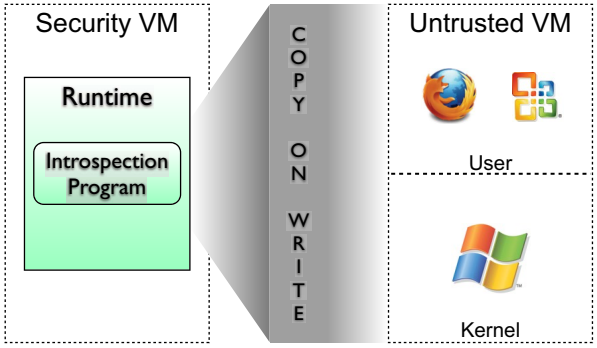
Virutoso [SP'11]

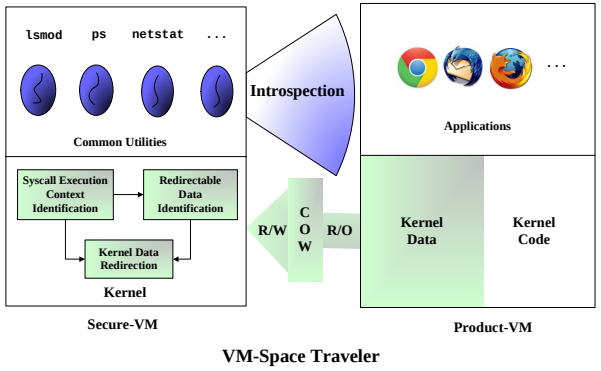
- Training → offline
- Binary code translation

VMST[SP'12]

- Taint analysis →online
- Data redirection

Insight: Can We Combine Offline and Online?





VM-Space Traveler

Virutoso [SP'11]

- Training → offline
- Binary code translation

VMST[SP'12]

- Taint analysis →online
- Data redirection

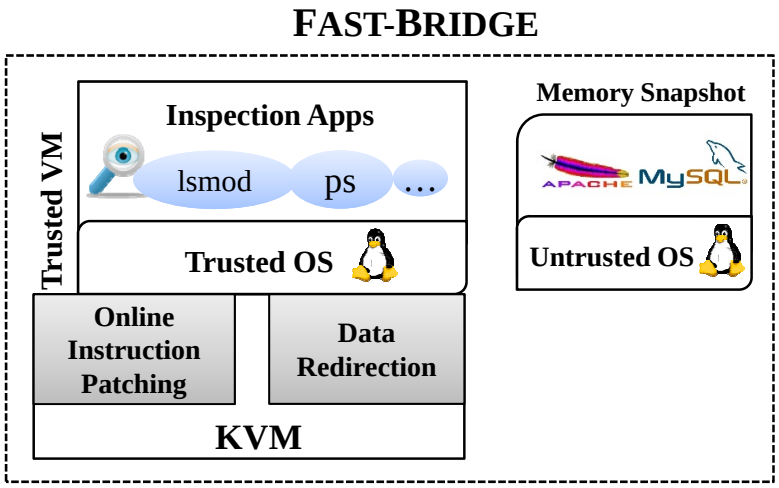
Hybrid

- Decouple the taint analysis
- Combine online and offline with a fallback (much like an OS page fault mechanism) and memoization

Hybrid-Bridge: Architecture Overview

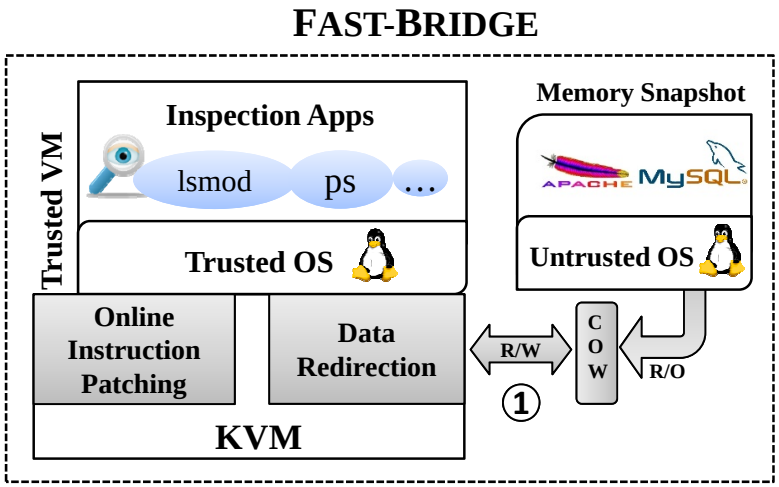
HYBRID-BRIDGE

Hybrid-Bridge: Architecture Overview



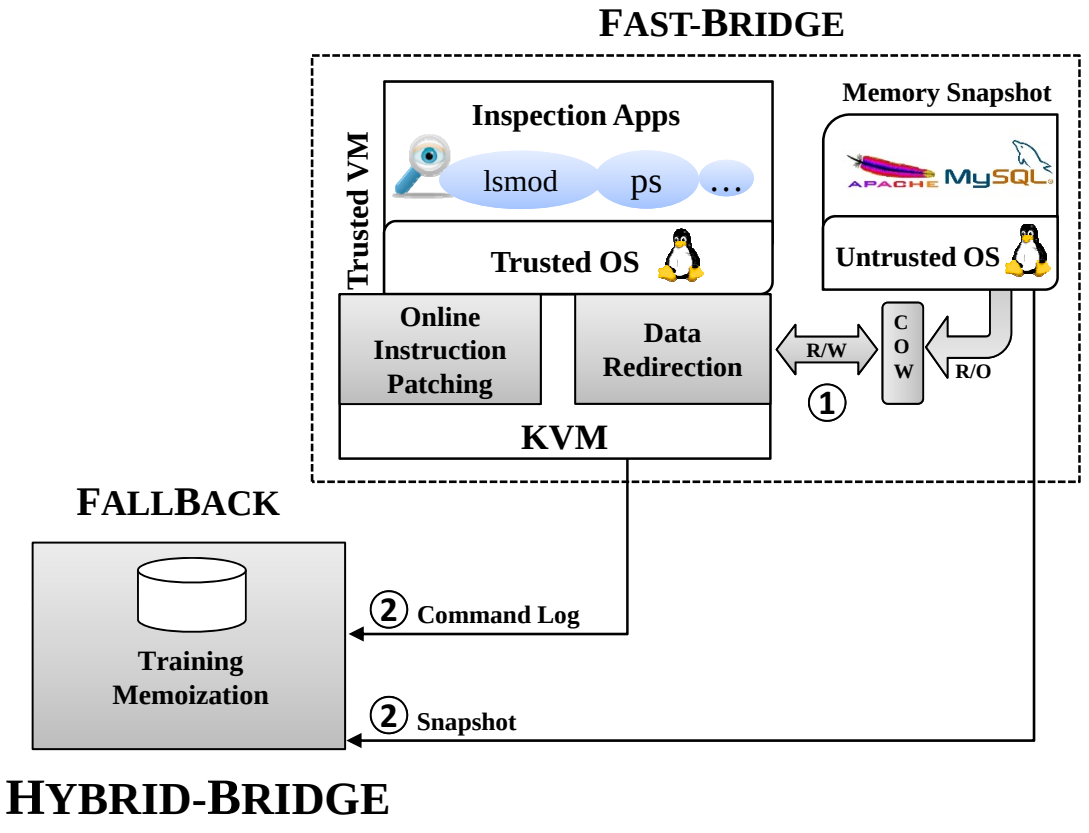
HYBRID-BRIDGE

Hybrid-Bridge: Architecture Overview

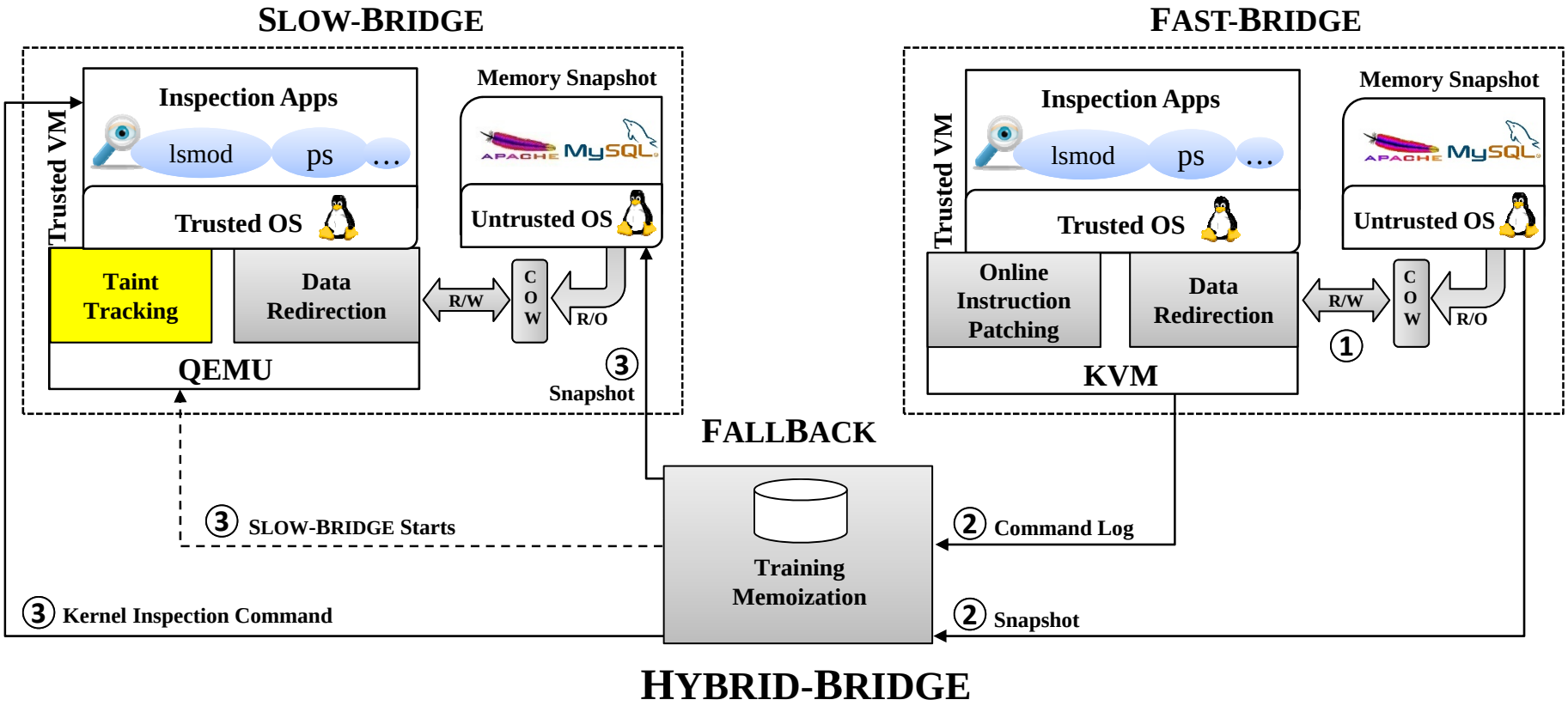


HYBRID-BRIDGE

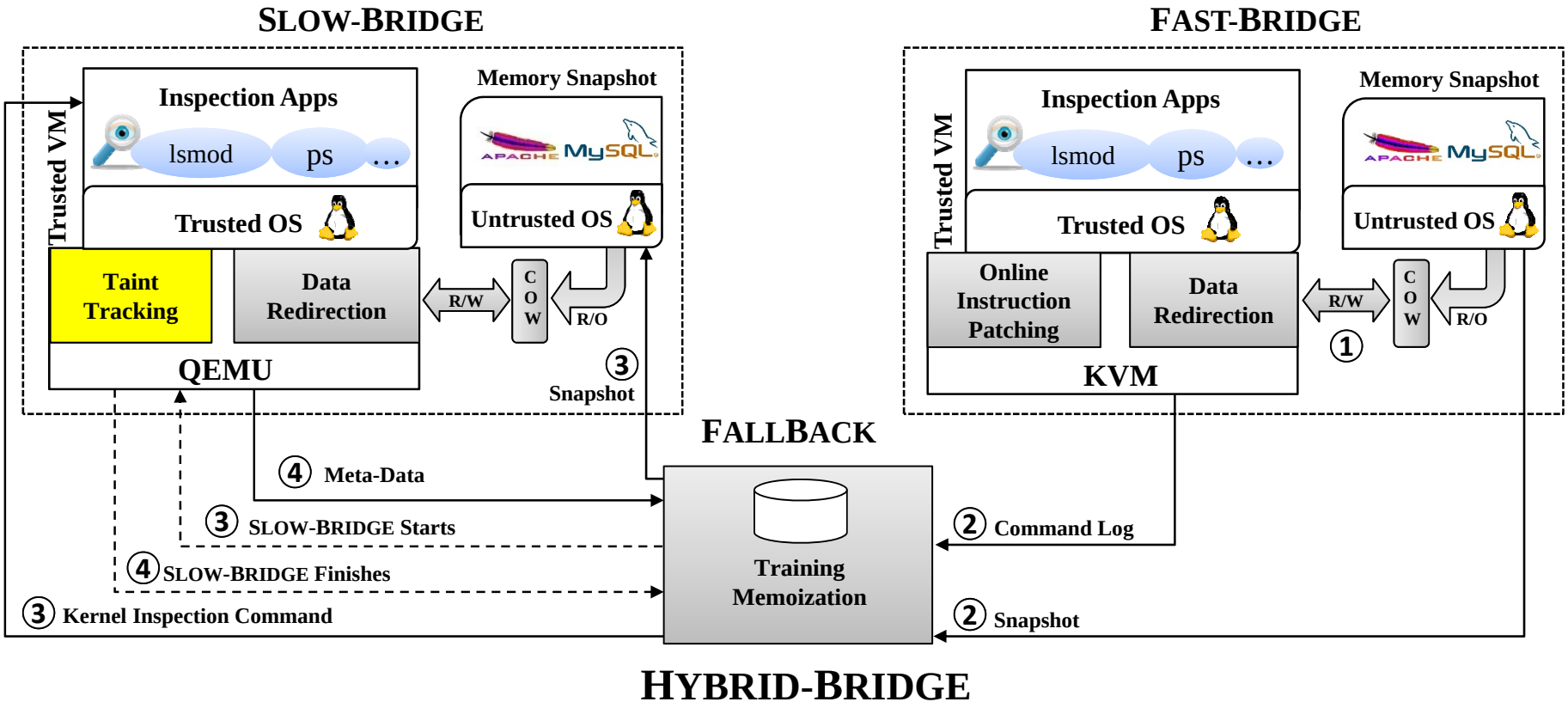
Hybrid-Bridge: Architecture Overview



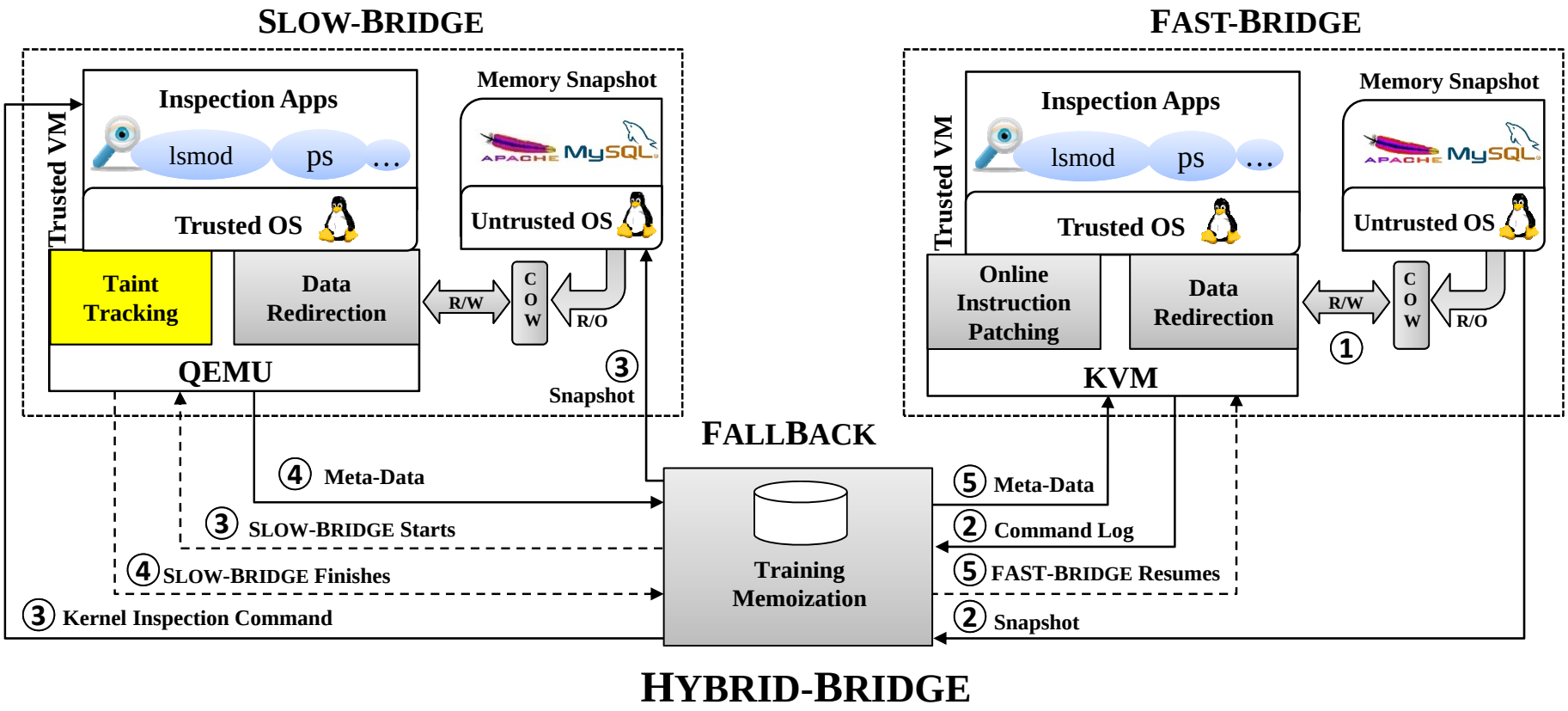
Hybrid-Bridge: Architecture Overview



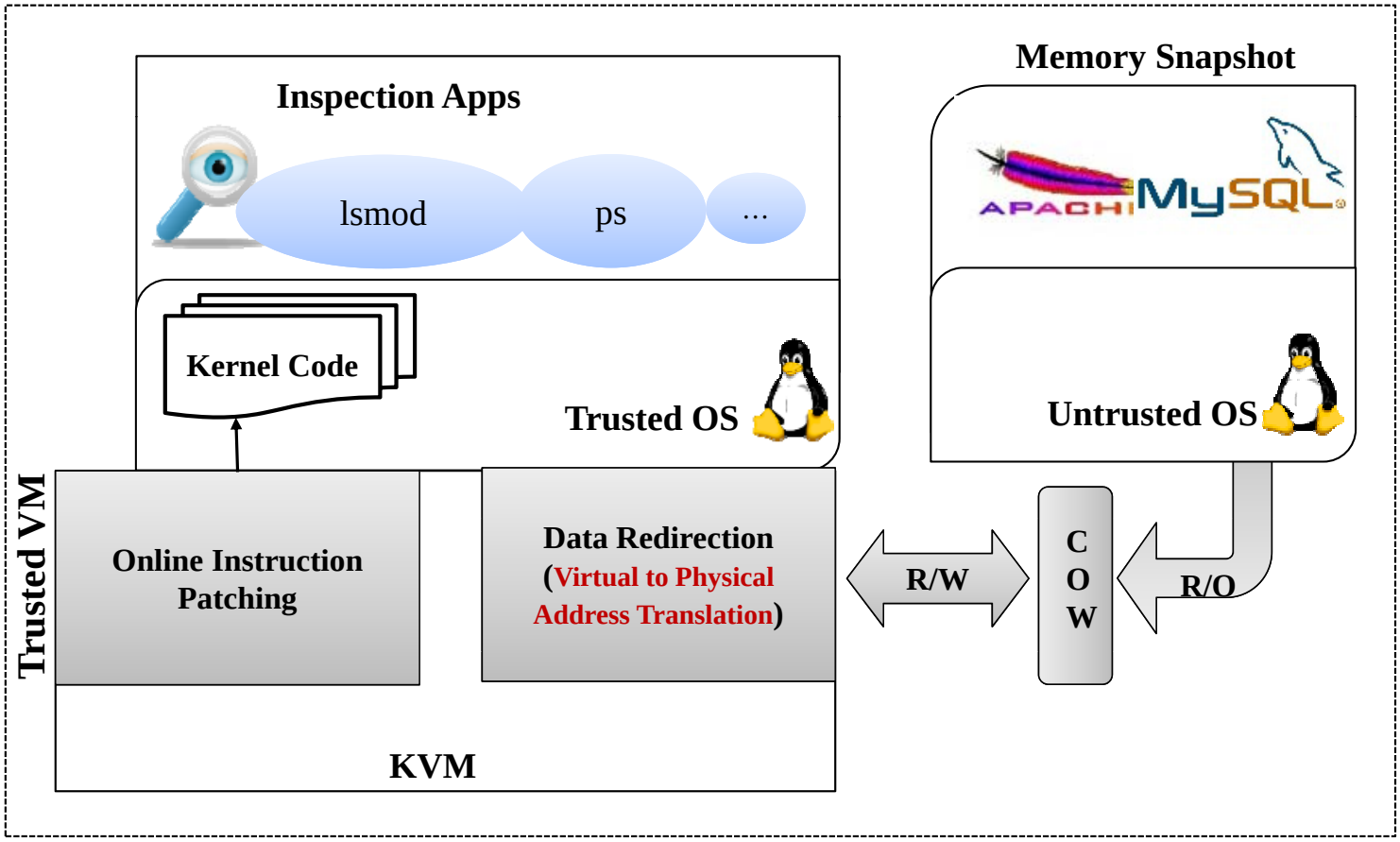
Hybrid-Bridge: Architecture Overview



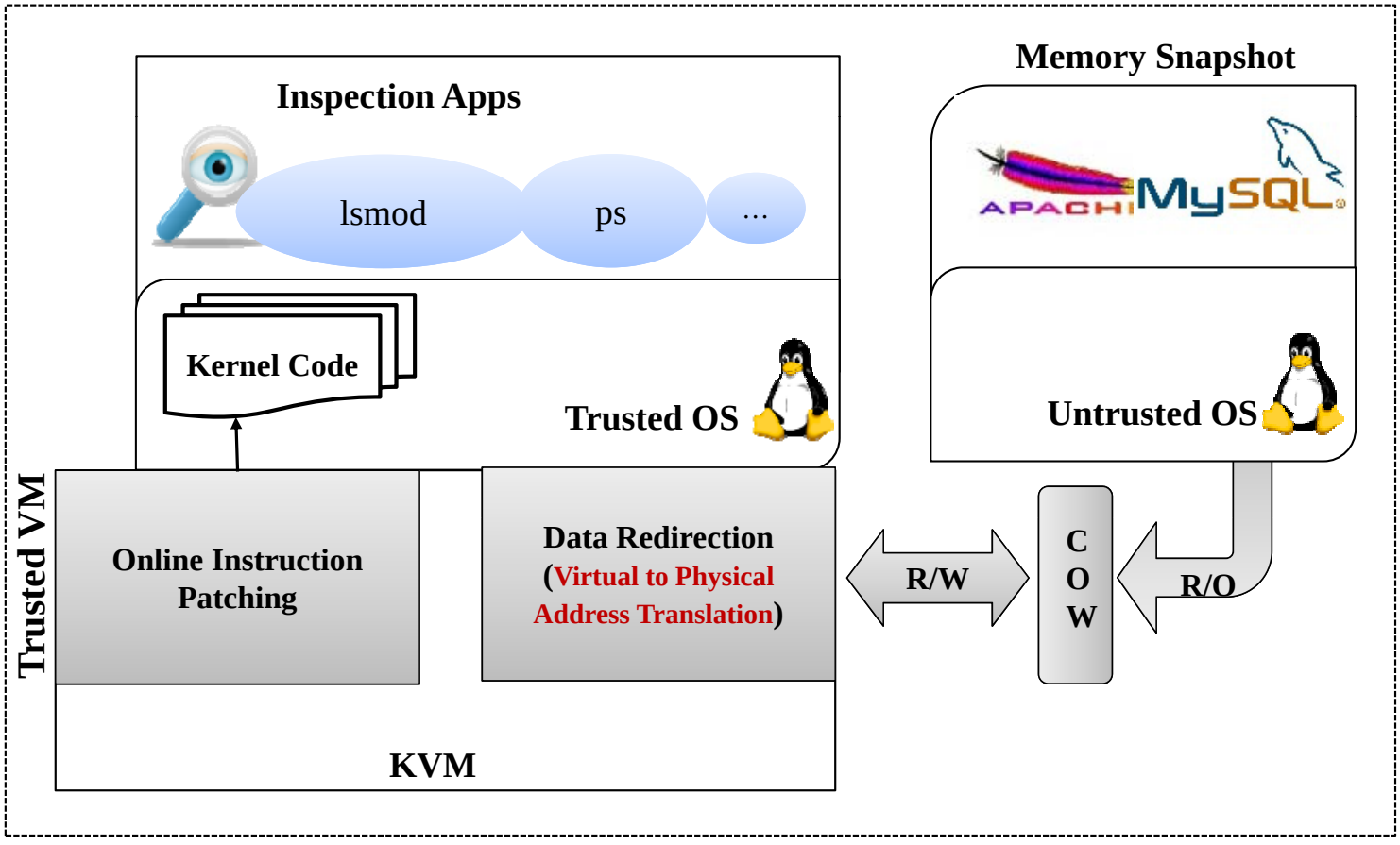
Hybrid-Bridge: Architecture Overview



Fast-Bridge


Inspection Apps
lsmod
ps
...
Memory Snapshot
APACHI MySQL
Kernel Code
Trusted OS
Untrusted OS
Trusted VM
Online Instruction Patching
Data Redirection (Virtual to Physical Address Translation)
R/W
C O W
R/O
KVM

Fast-Bridge



Kernel Data Redirection

- Static Kernel Binary Rewriting (hard)
- Dynamic Kernel Binary Instrumentation (slow)

Data Redirection in KVM via Instruction Patching

Original Code Page

`<sys_getpid>:`
`<task_tgid_vnr>:`
`c10583e0: push %ebp`

`c10583e1: mov %esp,%ebp`
`c10583e3: push %ebx`
`c10583e4: sub $0x14,%esp`
`c10583e7: mov %fs:0xc17f34cc,%ebx` `c10583ea: R_386_32 current_task`
`c10583fe: mov 0x220(%ebx),%eax`
`c1058404: mov 0x23c(%eax),%eax`
`c105840a: call c1065660` `<pid_vnr>`
`c105840f: add $0x14,%esp`

Data Redirection in KVM via Instruction Patching

Original Code Page	Non-Redirectable Code Page	Redirectable Code Page
<sys_getpid>: <task_tgid_vnr>: c10583e0: push %ebp	push %ebp	int 3
c10583e1: mov %esp,%ebp	mov %esp,%ebp	mov %esp,%ebp
c10583e3: push %ebx	push %ebx	int 3
c10583e4: sub $0x14,%esp	sub $0x14,%esp	sub $0x14,%esp
c10583e7: mov %fs:0xc17f34cc,%ebx c10583ea: R_386_32 current_task	int 3	mov %fs:0xc17f34cc,%ebx c10583ea: R_386_32 current_task
c10583fe: mov 0x220(%ebx),%eax	int 3	mov 0x220(%ebx),%eax
c1058404: mov 0x23c(%eax),%eax	int 3	mov 0x23c(%eax),%eax
c105840a: call c1065660 <pid_vnr>	call c1065660 <pid_vnr>	int 3
c105840f: add $0x14,%esp	add $0x14,%esp	add $0x14,%esp

Data Redirection in KVM via Instruction Patching

Original Code Page	Non-Redirectable Code Page	Redirectable Code Page
`<sys_getpid>:`		
`<task_tgid_vnr>:`		
`c10583e0: push %ebp`	`push %ebp`	`int 3`
`c10583e1: mov %esp,%ebp`	`mov %esp,%ebp`	`mov %esp,%ebp`
`c10583e3: push %ebx`	`push %ebx`	`int 3`
`c10583e4: sub $0x14,%esp`	`sub $0x14,%esp`	`sub $0x14,%esp`
`c10583e7: mov %fs:0xc17f34cc,%ebx` `c10583ea: R_386_32 current_task`	`int 3`	`mov %fs:0xc17f34cc,%ebx` `c10583ea: R_386_32 current_task`
`c10583fe: mov 0x220(%ebx),%eax`	`int 3`	`mov 0x220(%ebx),%eax`
`c1058404: mov 0x23c(%eax),%eax`	`int 3`	`mov 0x23c(%eax),%eax`
`c105840a: call c1065660 <pid_vnr>`	`call c1065660 <pid_vnr>`	`int 3`
`c105840f: add $0x14,%esp`	`add $0x14,%esp`	`add $0x14,%esp`

Data Redirection in KVM via Instruction Patching

Original Code Page	Non-Redirectable Code Page	Redirectable Code Page
<sys_getpid>: <task_tgid_vnr>: c10583e0: push %ebp	push %ebp	int 3
c10583e1: mov %esp,%ebp	mov %esp,%ebp	mov %esp,%ebp
c10583e3: push %ebx	push %ebx	int 3
c10583e4: sub $0x14,%esp	sub $0x14,%esp	sub $0x14,%esp
c10583e7: mov %fs:0xc17f34cc,%ebx *c10583ea: R_386_32 current_task*	int 3	mov %fs:0xc17f34cc,%ebx *c10583ea: R_386_32 current_task*
c10583fe: mov 0x220(%ebx),%eax	int 3	mov 0x220(%ebx),%eax
c1058404: mov 0x23c(%eax),%eax	int 3	mov 0x23c(%eax),%eax
c105840a: call c1065660 <pid_vnr>	call c1065660 <pid_vnr>	int 3
c105840f: add $0x14,%esp	add $0x14,%esp	add $0x14,%esp

VMexit

Data Redirection in KVM via Instruction Patching

Original Code Page	Non-Redirectable Code Page	Redirectable Code Page
<sys_getpid>: <task_tgid_vnr>: c10583e0: push %ebp	push %ebp	int 3
c10583e1: mov %esp,%ebp	mov %esp,%ebp	mov %esp,%ebp
c10583e3: push %ebx	push %ebx	int 3
c10583e4: sub $0x14,%esp	sub $0x14,%esp	sub $0x14,%esp
c10583e7: mov %fs:0xc17f34cc,%ebx *c10583ea: R_386_32 current_task*	int 3	mov %fs:0xc17f34cc,%ebx *c10583ea: R_386_32 current_task*
c10583fe: mov 0x220(%ebx),%eax	int 3	mov 0x220(%ebx),%eax
c1058404: mov 0x23c(%eax),%eax	int 3	mov 0x23c(%eax),%eax
c105840a: call c1065660 <pid_vnr>	call c1065660 <pid_vnr>	int 3
c105840f: add $0x14,%esp	add $0x14,%esp	add $0x14,%esp

VMexit

Data Redirection in KVM via Instruction Patching

Original Code Page	Non-Redirectable Code Page	Redirectable Code Page
<sys_getpid>: <task_tgid_vnr>: c10583e0: push %ebp	push %ebp	int 3
c10583e1: mov %esp,%ebp	mov %esp,%ebp	mov %esp,%ebp
c10583e3: push %ebx	push %ebx	int 3
c10583e4: sub $0x14,%esp	sub $0x14,%esp	sub $0x14,%esp
c10583e7: mov %fs:0xc17f34cc,%ebx *c10583ea: R_386_32 current_task*	int 3	mov %fs:0xc17f34cc,%ebx *c10583ea: R_386_32 current_task*
c10583fe: mov 0x220(%ebx),%eax	int 3	mov 0x220(%ebx),%eax
c1058404: mov 0x23c(%eax),%eax	int 3	mov 0x23c(%eax),%eax
c105840a: call c1065660 <pid_vnr>	call c1065660 <pid_vnr>	int 3
c105840f: add $0x14,%esp	add $0x14,%esp	add $0x14,%esp

VMexit

VMexit

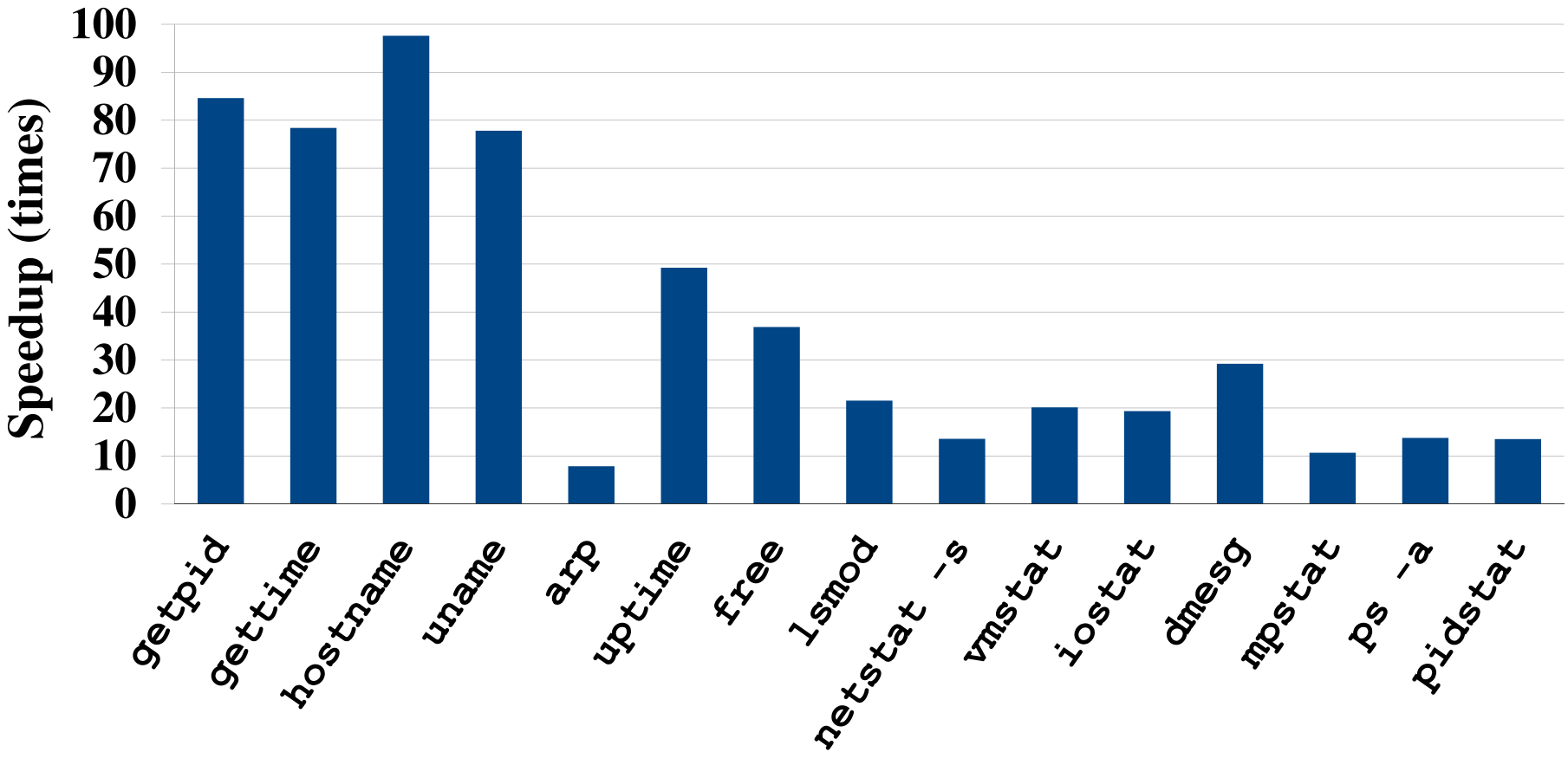
Data Redirection in KVM via Instruction Patching

Original Code Page	Non-Redirectable Code Page	Redirectable Code Page
<sys_getpid>: <task_tgid_vnr>: c10583e0: push %ebp	push %ebp	int 3
c10583e1: mov %esp,%ebp	mov %esp,%ebp	mov %esp,%ebp
c10583e3: push %ebx	push %ebx	int 3
c10583e4: sub $0x14,%esp	sub $0x14,%esp	sub $0x14,%esp
c10583e7: mov %fs:0xc17f34cc,%ebx *c10583ea: R_386_32 current_task*	int 3	mov %fs:0xc17f34cc,%ebx *c10583ea: R_386_32 current_task*
c10583fe: mov 0x220(%ebx),%eax	int 3	mov 0x220(%ebx),%eax
c1058404: mov 0x23c(%eax),%eax	int 3	mov 0x23c(%eax),%eax
c105840a: call c1065660 <pid_vnr>	call c1065660 <pid_vnr>	int 3
c105840f: add $0x14,%esp	add $0x14,%esp	add $0x14,%esp

VMexit

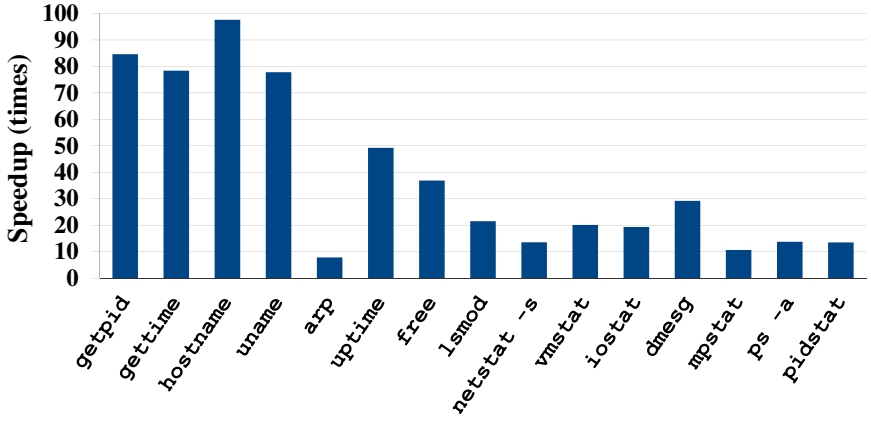
VMexit

Fast-Bridge Speedup Compared to VMST



Fast-Bridge Speedup Compared to VMST

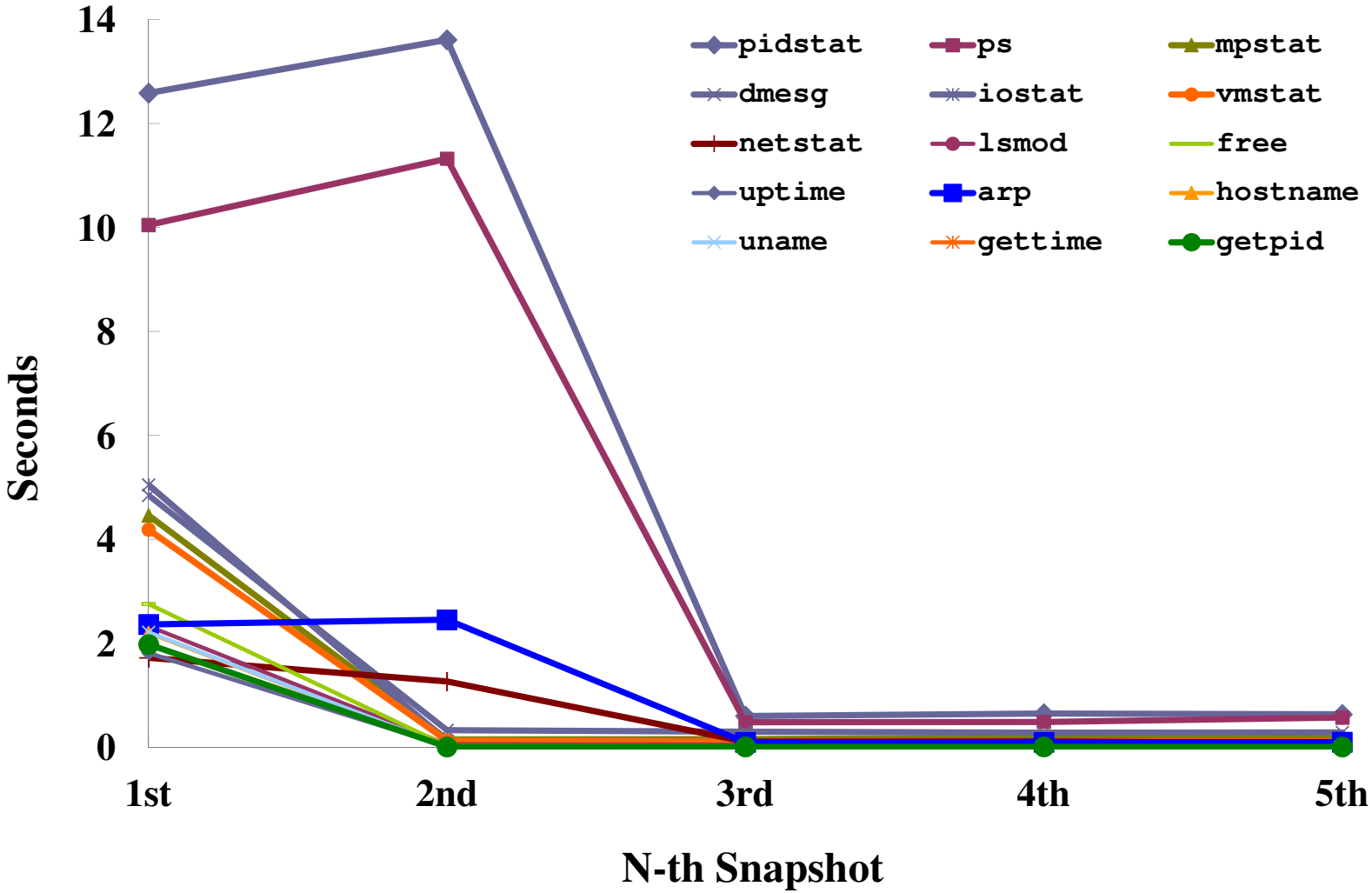
App. Name	HYBRID-BRIDGE *#VMExit*	Speedup FAST-BRIDGE vs. VMST
getpid	2	84.60X
gettime	4	78.40X
hostname	10	97.60X
uname	10	77.80X
arp	1852	7.86X
uptime	1892	49.25X
free	3927	36.88X
lsmod	11875	21.54X
netstat	23165	13.59X
vmstat	86578	20.13X
iostat	97390	19.35X
dmesg	11663	29.22X
mpstat	124525	10.68X
ps	418124	13.76X
pidstat	490713	13.53X



FAST-BRIDGE vs. VIRTUOSO

App. Name	Description	#X86 Inst. in VIRTUOSO	FAST-BRIDGE (sec.)	FAST-BRIDGE vs. VIRTUOSO
gettime	Tells current time of system	482	0.005	4.60X
getpid	Shows pid of current process	516	0.005	4.80X
tinyps	A compact version of PS	140843	0.064	23.45X
getprocname	Displays current Process Name	294797	0.132	20.57X

How often does it fall back to SLOW-BRIDGE



Approach-II: Redirect System call Execution [ATC'14]

In-VM `getpid` Program

```
1 #include <stdio.h>
2 #include <unistd.h>
3
4 int main()
5 {
6     printf("pid=%d\n",getpid());
7     return 0;
8 }
```

```
 1 execve("./getpid",..)           = 0
 2 brk(0)                    = 0x83b8000
 3 access("/etc/ld.so.nohwcap",.) = -1
...
23 getpid()                        = 13849
...
26 write(1, "pid=13849\n", 10)     = 10
27 exit_group(0)                   = ?
```

Approach-II: Redirect System call Execution [ATC'14]

In-VM `getpid` Program

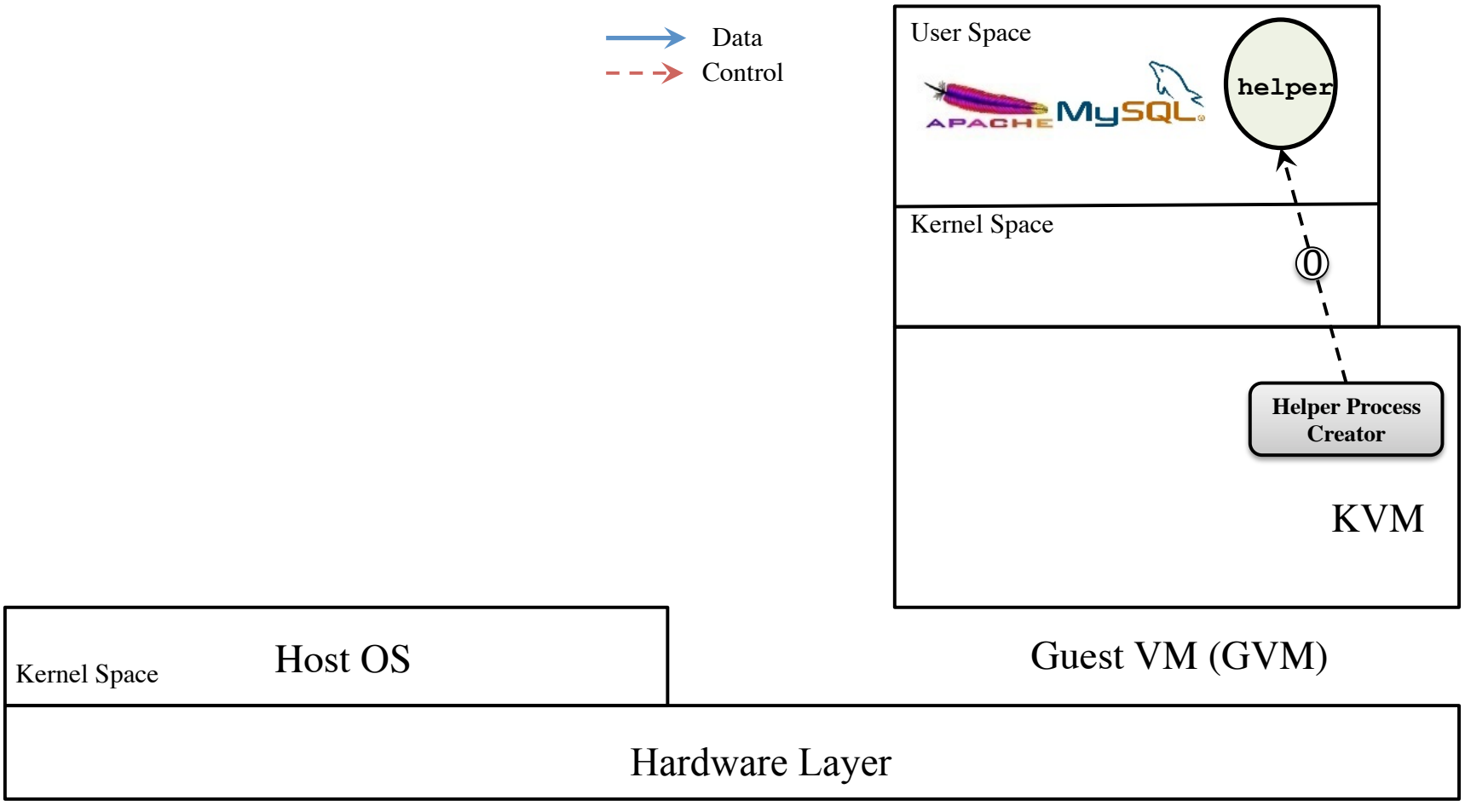
```
1 #include <stdio.h>
2 #include <unistd.h>
3
4 int main()
5 {
6     printf("pid=%d\n",getpid());
7     return 0;
8 }
```

```
 1 execve("./getpid",..)           = 0
 2 brk(0)                    = 0x83b8000
 3 access("/etc/ld.so.nohwcap",.) = -1
...
23 getpid()                        = 13849
...
26 write(1, "pid=13849\n", 10)     = 10
27 exit_group(0)                   = ?
```

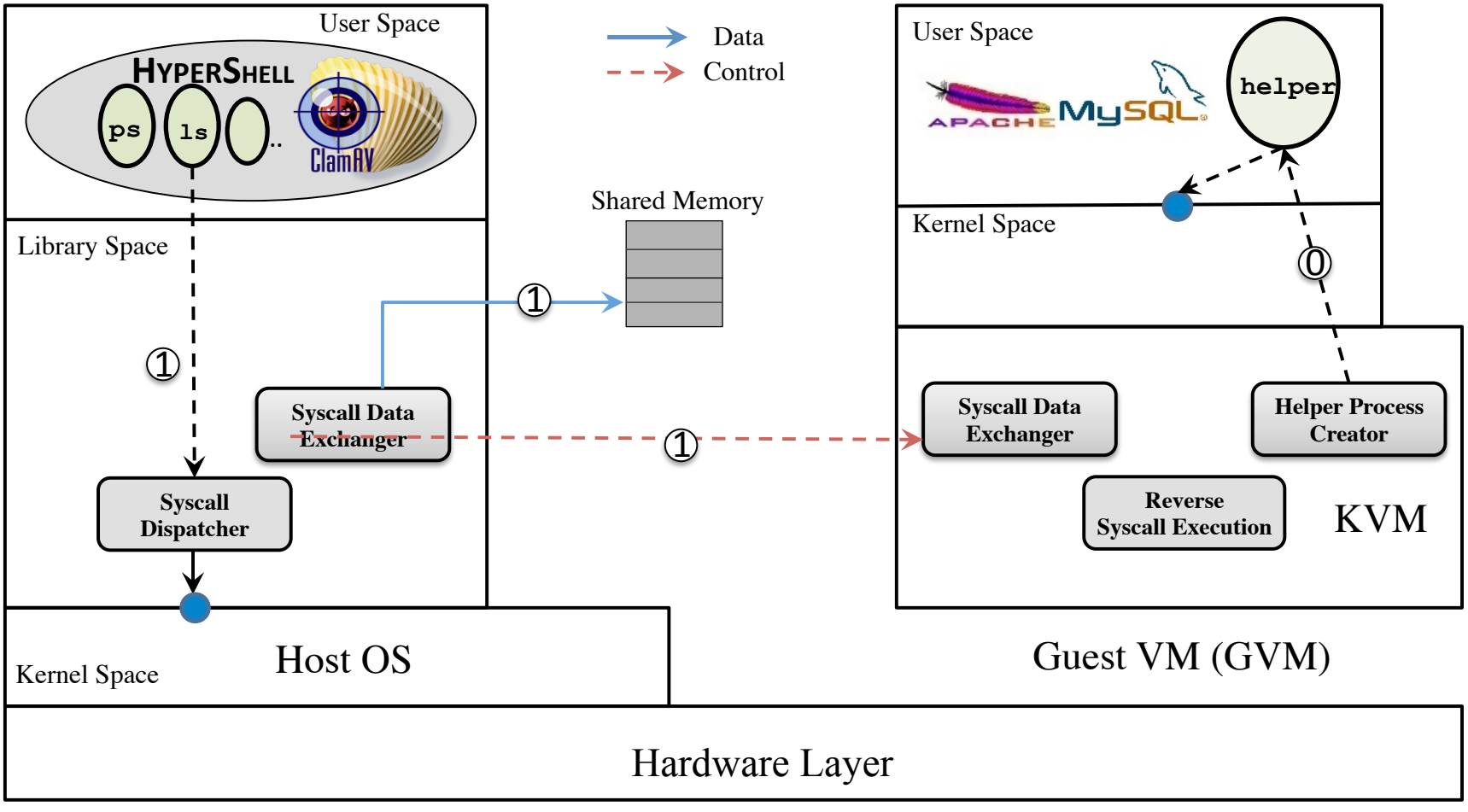
Key Insight

- System call is the only interface to request OS service.
- Pushing the execution of `getpid` system call from SVM to GVM.

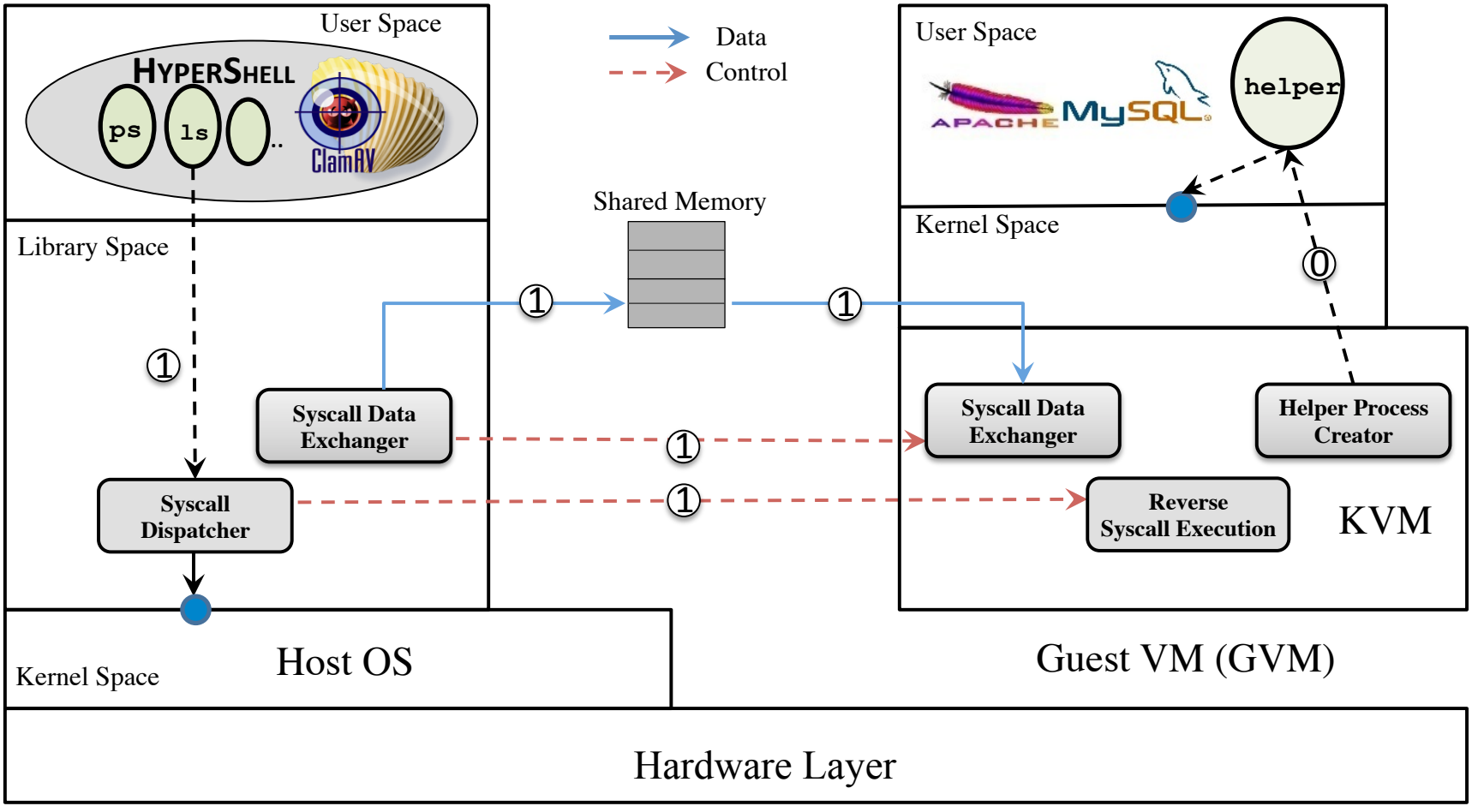
Approach-II: Design & Implementation [ATC'14]



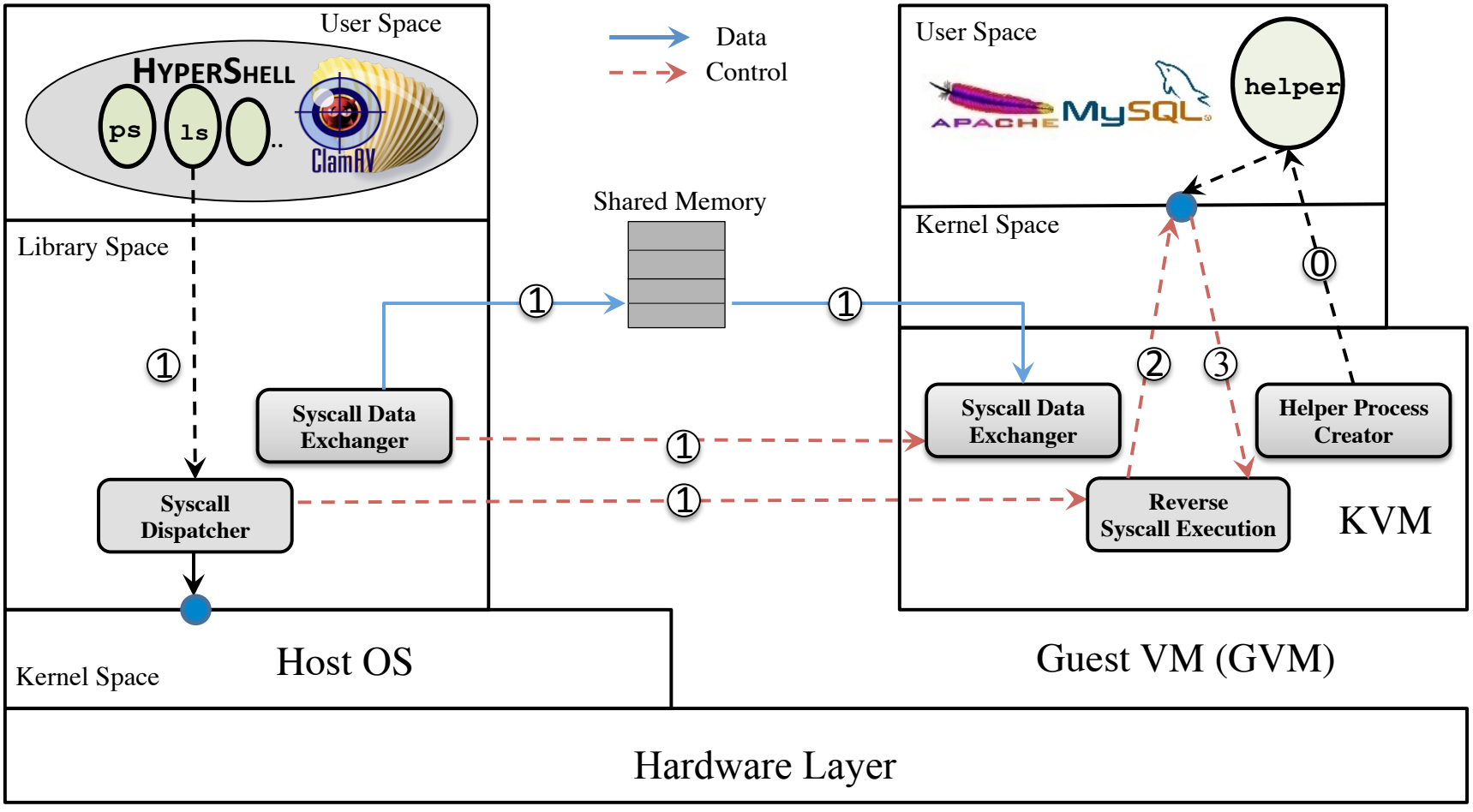
Approach-II: Design & Implementation [ATC'14]



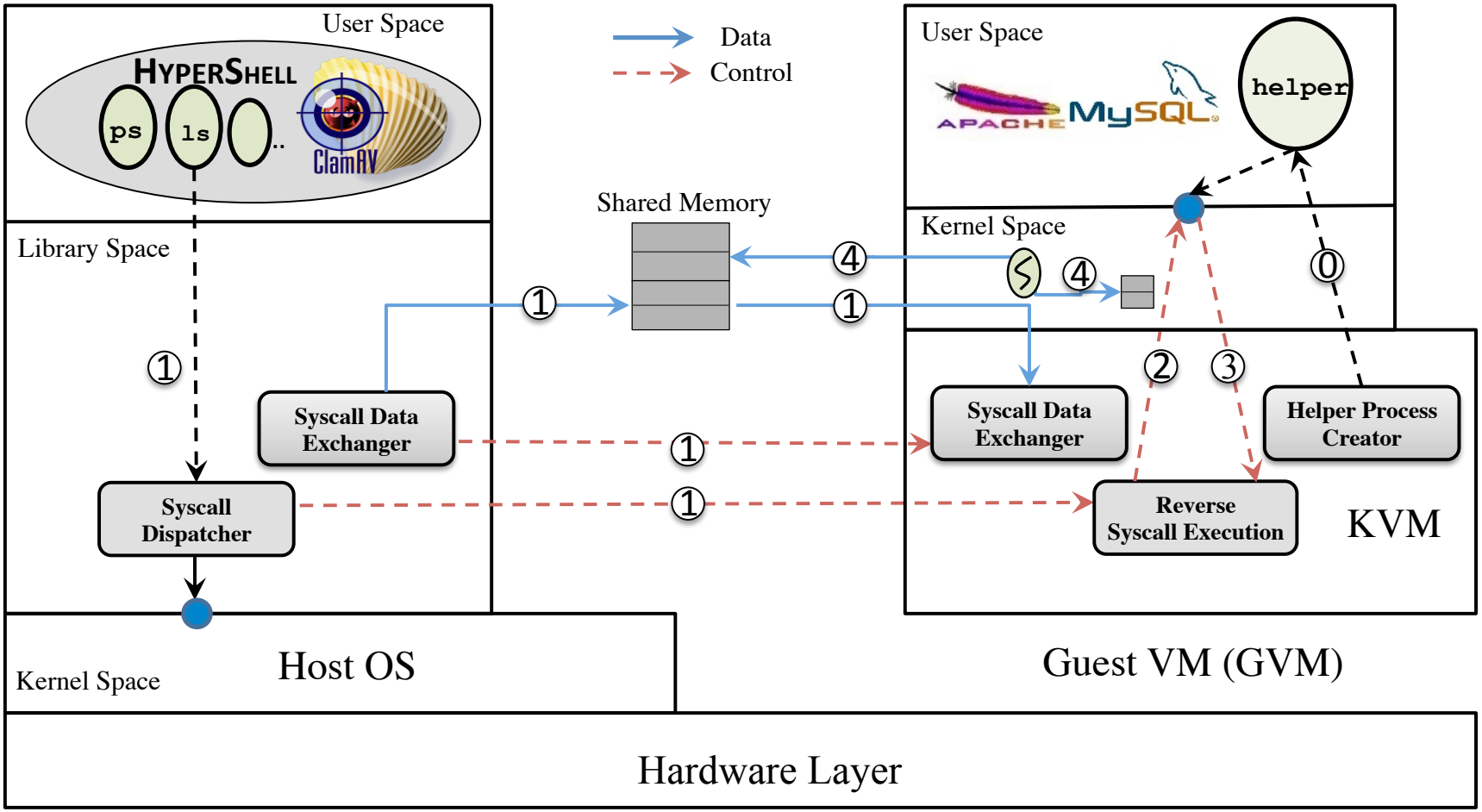
Approach-II: Design & Implementation [ATC'14]



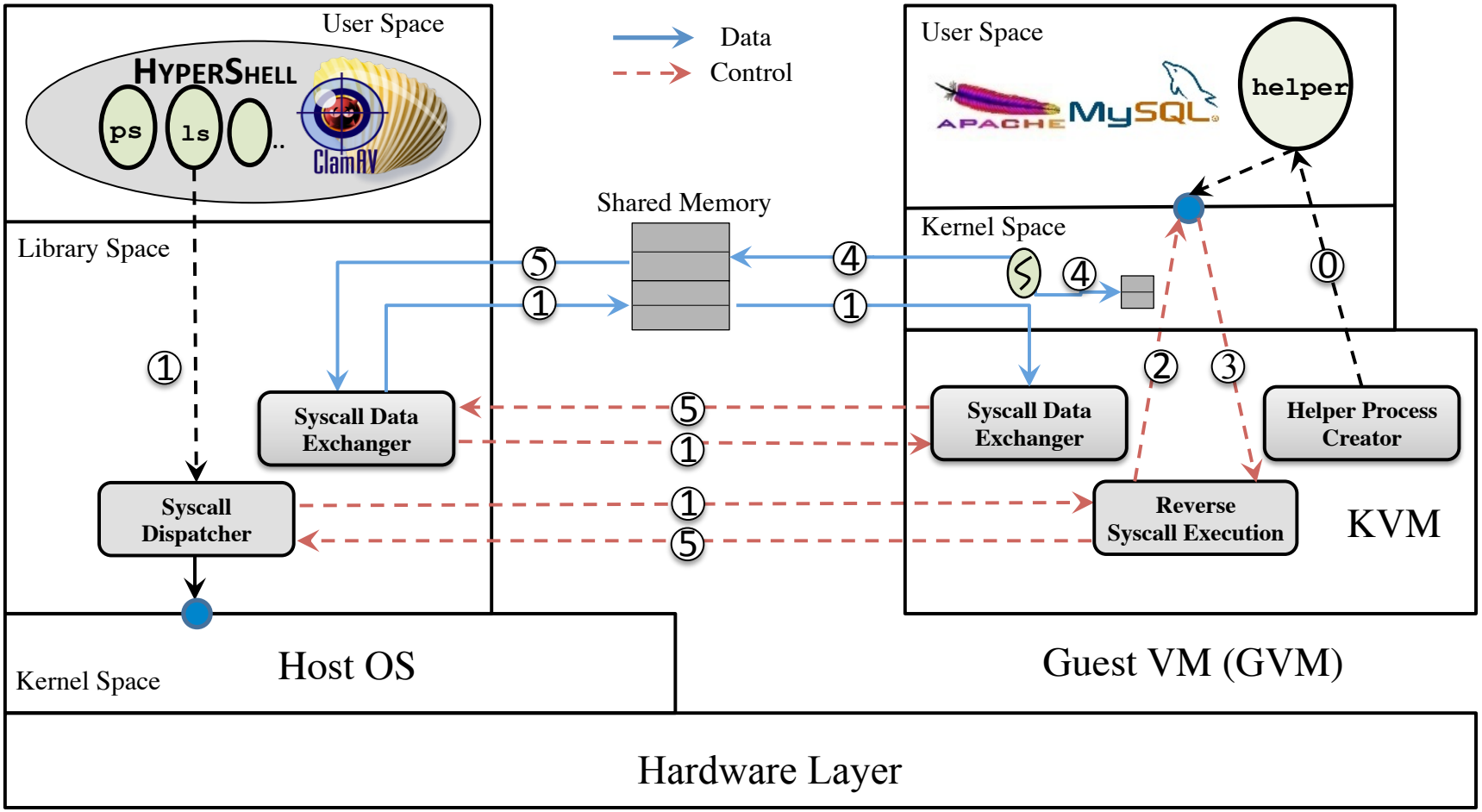
Approach-II: Design & Implementation [ATC'14]



Approach-II: Design & Implementation [ATC'14]



Approach-II: Design & Implementation [ATC'14]



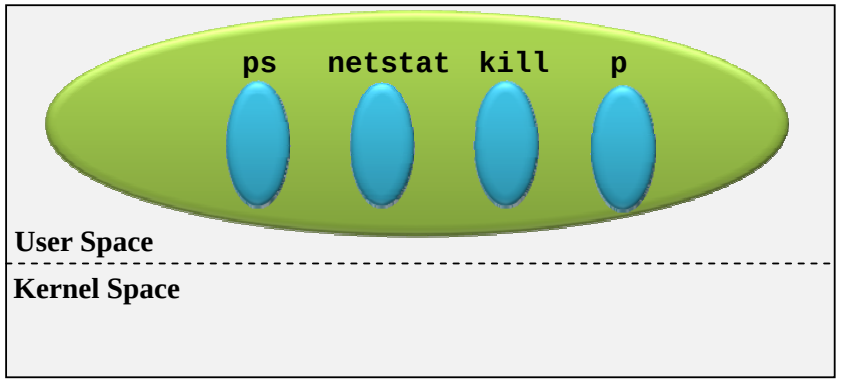
Performance Impact: HyperShell [ATC'14]

Process	**S**	$B(ms)$	$D(ms)$	$T(X)$
ps	✗	1.33	5.42	4.08
pidstat	✗	1.95	7.56	3.88
nice	✓	0.07	0.11	1.57
getpid	✓	0.01	0.02	2.00
mpstat	✗	0.29	0.66	2.28
pstree	✗	0.69	6.03	8.74
chrt	✓	0.11	0.16	1.45
renice	✓	0.11	0.18	1.64
top	✗	504.92	510.85	1.01
nproc	✓	0.07	0.26	3.71
sleep	✓	1.27	1.28	1.01
pgrep	✓	0.89	4.72	5.30
pkill	✓	0.87	4.33	4.98
snice	✓	0.17	0.65	3.82
echo	✓	0.07	0.09	1.29
pwdx	✓	0.05	0.07	1.40
pmap	✓	0.16	0.36	2.25
kill	✓	0.01	0.04	4.00
killall	✓	0.62	3.03	4.89
Memory	**S**	$B(ms)$	$D(ms)$	$T(X)$
free	✗	0.04	0.08	2.00
vmstat	✗	0.19	0.33	1.74
slabtop	✗	0.22	0.36	1.64
Modules	**S**	$B(ms)$	$D(ms)$	$T(X)$
rmmod	✓	0.51	3.14	6.16
modinfo	✓	0.48	1.54	3.21
lsmod	✓	0.10	0.17	1.70
Environment	**S**	$B(ms)$	$D(ms)$	$T(X)$
who	✓	0.14	0.72	5.14
env	✓	0.07	0.11	1.57
printenv	✓	0.07	0.1	1.43
whoami	✓	0.19	0.45	2.37
stty	✓	0.11	0.46	4.18
users	✓	0.09	0.53	5.89
uname	✓	0.09	0.11	1.22
id	✓	0.26	0.85	3.27
date	✗	0.11	0.12	1.09
w	✗	0.95	6.62	6.97
hostname	✓	0.04	0.06	1.50
groups	✓	0.21	0.62	2.95
hostid	✓	0.16	0.56	3.50
locale	✓	0.09	0.17	1.89
getconf	✓	0.09	0.34	3.78
System Utils	**S**	$B(ms)$	$D(ms)$	$T(X)$
uptime	✗	0.07	0.47	6.71
sysctl	✓	8.5	42.72	5.03
arch	✓	0.07	0.11	1.57
dmesg	✓	0.38	0.51	1.34
lscpu	✓	0.26	1.21	4.65
mcookie	✗	0.29	0.49	1.69
Disk/Devices	**S**	$B(ms)$	$D(ms)$	$T(X)$
blkid	✓	0.14	0.61	4.36
badblocks	✓	0.35	0.44	1.26
lspci	✓	31.40	36.52	1.16
iostat	✓	0.45	1.04	2.31
du	✓	0.11	0.53	4.82
df	✓	0.16	0.35	2.19
Filesystem	**S**	$B(ms)$	$D(ms)$	$T(X)$
sync	✓	8.07	6.53	0.81
getcap	✓	0.04	0.08	2.00
lsof	✓	3.31	6.12	1.85
pwd	✓	0.07	0.11	1.57
Files	**S**	$B(ms)$	$D(ms)$	$T(X)$
chgrp	✓	0.19	0.47	2.47
chmod	✓	0.07	0.14	2.00
chown	✓	0.19	0.47	2.47
cp	✓	0.11	0.27	2.45
uniq	✓	0.09	0.35	3.89
file	✓	0.87	1.72	1.98
find	✓	0.20	0.58	2.90
grep	✓	0.35	2.14	6.11
ln	✓	0.08	0.14	1.75
ls	✓	0.14	0.27	1.93
mkdir	✓	0.10	0.19	1.90
mkfifo	✓	0.10	0.19	1.90
mknod	✓	0.10	0.19	1.90
mv	✓	0.15	0.31	2.07
rm	✓	0.08	0.15	1.88
od	✓	0.12	0.35	2.92
cat	✓	0.07	0.18	2.57
link	✓	0.07	0.13	1.86
comm	✓	0.08	0.22	2.75
shred	✗	0.72	0.92	1.28
truncate	✓	0.07	0.26	3.71
head	✓	0.07	0.15	2.14
vdir	✓	0.63	3.95	6.27
nl	✓	0.08	0.17	2.13
tail	✓	0.08	0.20	2.50
namei	✓	0.07	0.13	1.86
whereis	✓	2.05	4.86	2.37
stat	✓	0.27	0.78	2.89
readlink	✓	0.07	0.12	1.71
unlink	✓	0.07	0.13	1.86
cut	✓	0.08	0.17	2.13
dir	✓	0.07	0.20	2.86
mktemp	✓	0.09	0.18	2.00
rmdir	✓	0.07	0.13	1.86
ptx	✓	0.12	0.45	3.75
chcon	✓	0.06	0.12	2.00
Network	**S**	$B(ms)$	$D(ms)$	$T(X)$
ifconfig	✗	0.32	1.15	3.59
ip	✓	0.10	0.20	2.00
route	✓	138.65	150.32	1.08
ipmaddr	✓	0.13	0.34	2.62
iptunnel	✓	0.09	0.29	3.22
nameif	✓	0.10	0.21	2.10
netstat	✗	0.25	0.37	1.48
arp	✓	0.14	0.24	1.71
ping	✗	15.02	18.2	1.21
Avg.	-	7.27	8.45	2.73

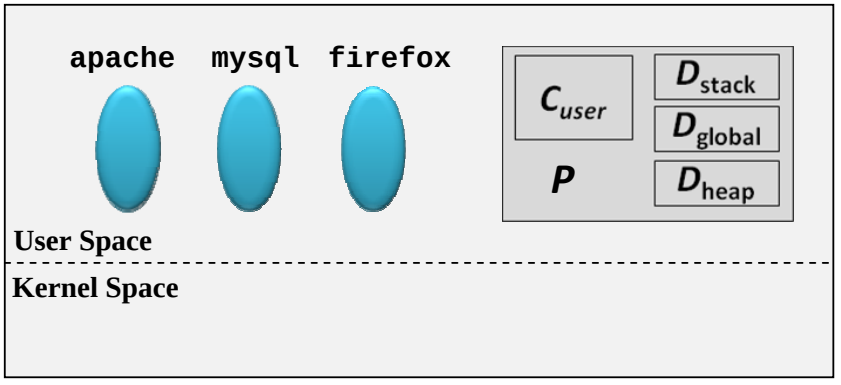
Comparison with the most related work

Systems	Execution Context Reuse	wo/ Dual-VM Architecture	wo/ Identical Kernel	wo/ Trust to Guest Kernel	High Code Coverage	Fully Automated	Memory Introspection	Disk Introspection	Guest Management	Process Monitoring
VIRTUOSO	✗	✓	✗	✓	✗	✗	✓	✗	✗	✗
VMST	✓	✗	✗	✓	✓	✓	✓	✗	✗	✗
EXTERIOR	✓	✗	✗	✓	✓	✓	✓	✗	✓	✗
PROCESSIMPLANTING	✓	✓	✓	✗	✓	✓	✗	✗	✓	✗
PROCESSOUTGRAFTING	✓	✗	✓	✓	✓	✓	✗	✗	✗	✓
GEARS	✓	✓	✓	✗	✓	✗	✓	✓	✓	✓
HYPERSHELL	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓

Two Approaches to bridging the semantic gap

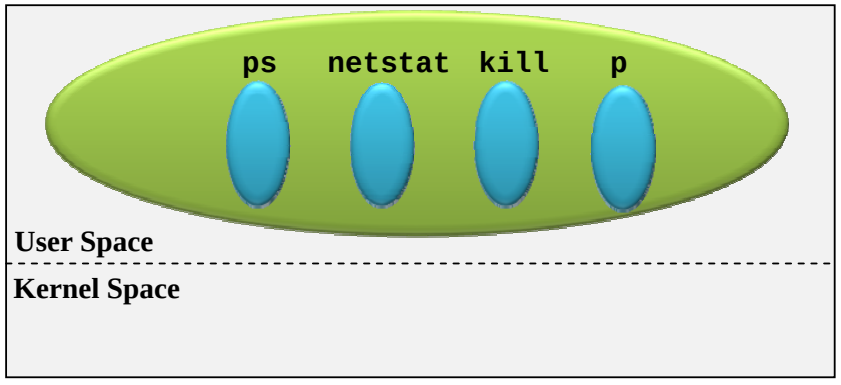


Secure VM (SVM)

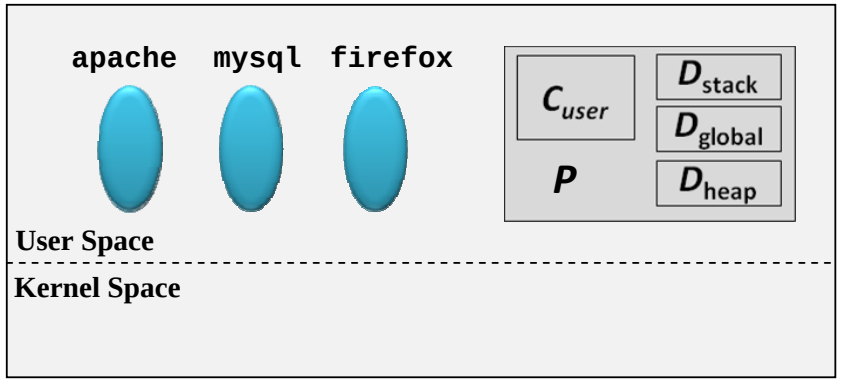


Guest VM (GVM)

Two Approaches to bridging the semantic gap



Secure VM (SVM)

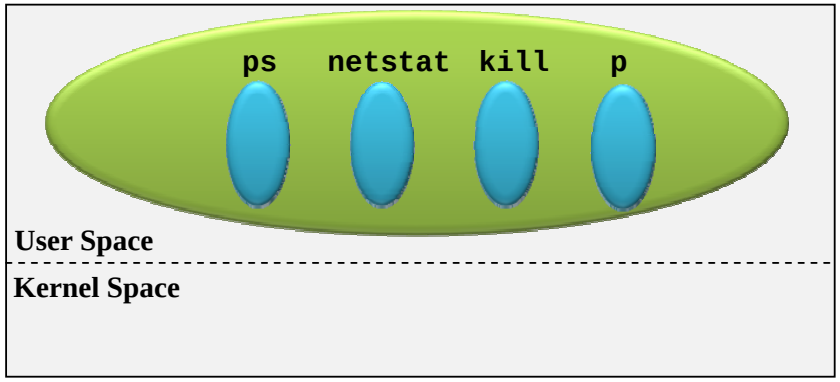


Guest VM (GVM)

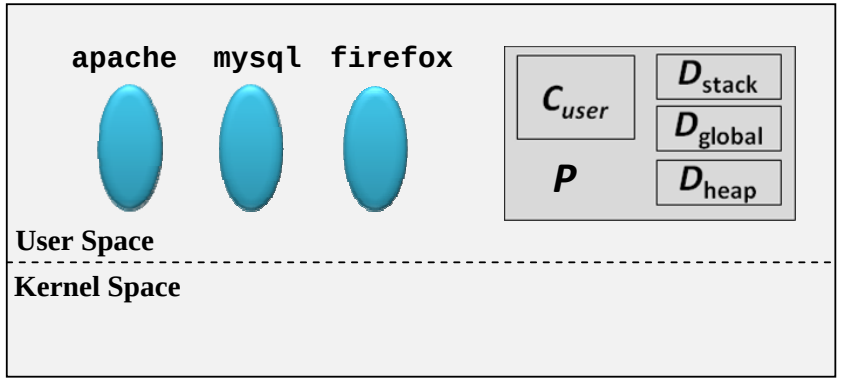
Reusing **(legacy) binary code** with a trusted secure VM to intropsect the running guest VM.

1. Redirect kernel data [SP'12, VEE'13, NDSS'14] → Fine-grained, slower performance
2. Redirect system call execution [USENIX ATC'14] → More practical, fast performance

For Cloud Developers: No Gap, Everything is Native



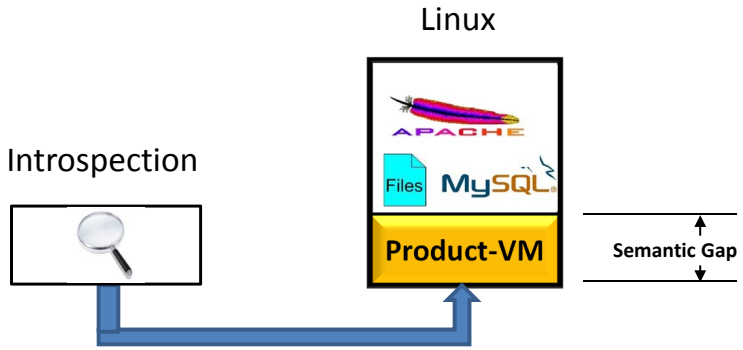
Secure VM (SVM)



Guest VM (GVM)

In-VM `getpid` Program

```
1 #include <stdio.h>
2 #include <unistd.h>
3
4 int main()
5 {
6     printf("pid=%d\n",getpid());
7     return 0;
8 }
```



References

1. VM Space Traveling: Automatically Bridging the Semantic Gap in **Virtual Machine Introspection** via Online Kernel Data Redirection. Yangchun Fu, Zhiqiang Lin (*IEEE Symposium on Security and Privacy* [Oakland'12])
2. EXTERIOR: Using A **Dual-VM** Enabled External Shell for Guest-OS Introspection, Configuration, and Recovery Yangchun Fu, Zhiqiang Lin (*ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* [VEE'13])
3. Hybrid-Bridge: Efficiently Bridging the Semantic-Gap in **Virtual Machine Introspection** via Decoupled Execution and Training Memoization. Alireza Saberi, Yangchun Fu, Zhiqiang Lin (*Network and Distributed System Symposium* [NDSS'14])
4. HyperShell: A Practical Hypervisor Layer Guest OS Shell for Automated In-**VM Management**. Yangchun Fu, Junyuan Zeng, Zhiqiang Lin (*USENIX Annual Technical Conference* [USENIX-ATC'14])
5. Automatically Deriving Pointer Reference Expressions From **Binary Code** For Memory Dump Analysis. Yangchun Fu, Zhiqiang Lin, David Brumley. (*ACM SIGSOFT Symposium on the Foundations of Software Engineering* [ESEC/FSE'15]).

`http://www.utdallas.edu/~zhiqiang.lin`