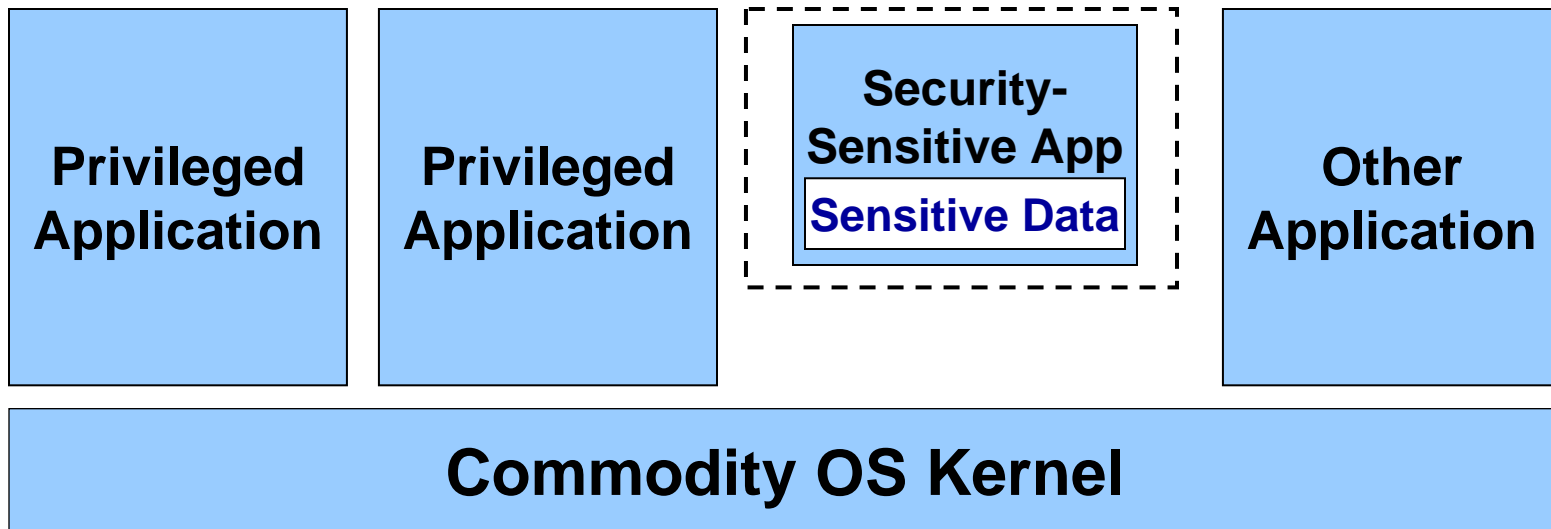


# Splitting Interfaces: Making Trust Between Applications and Operating Systems Configurable

Richard Ta-Min, Lionel Litty and **David Lie**  
Department of Electrical and Computer Engineering  
University of Toronto

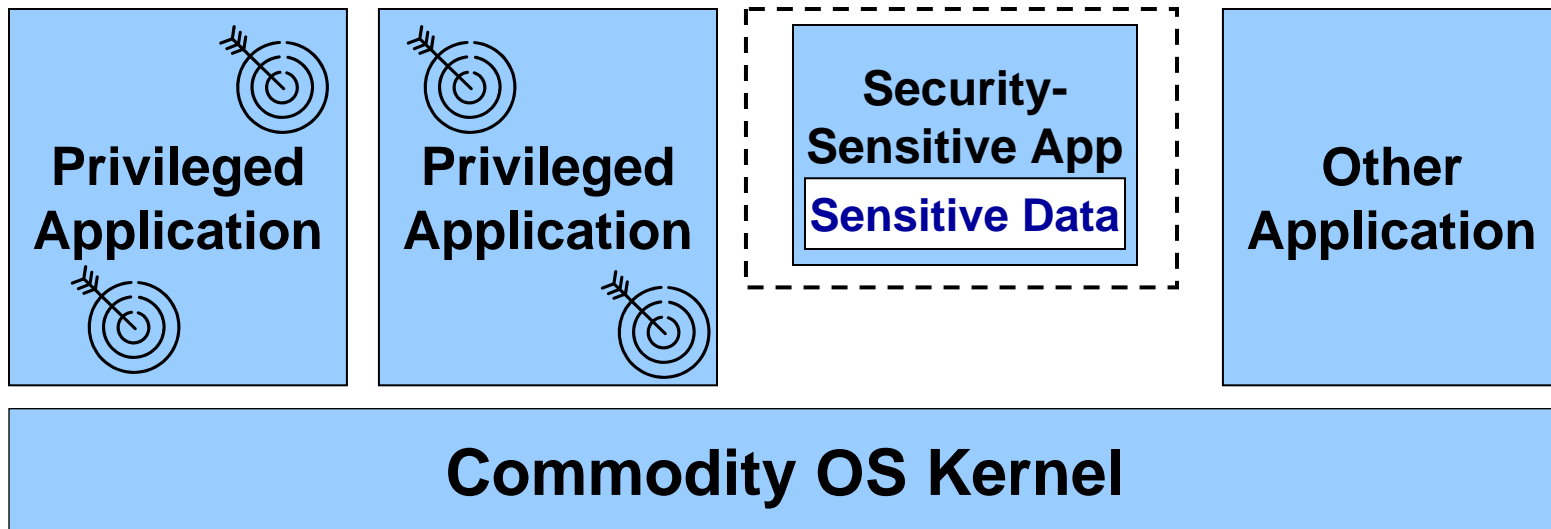
# Operating System Insecurity



- Commodity OS contain a lot of privileged code:
  - Gives attackers many opportunities to take control of OS



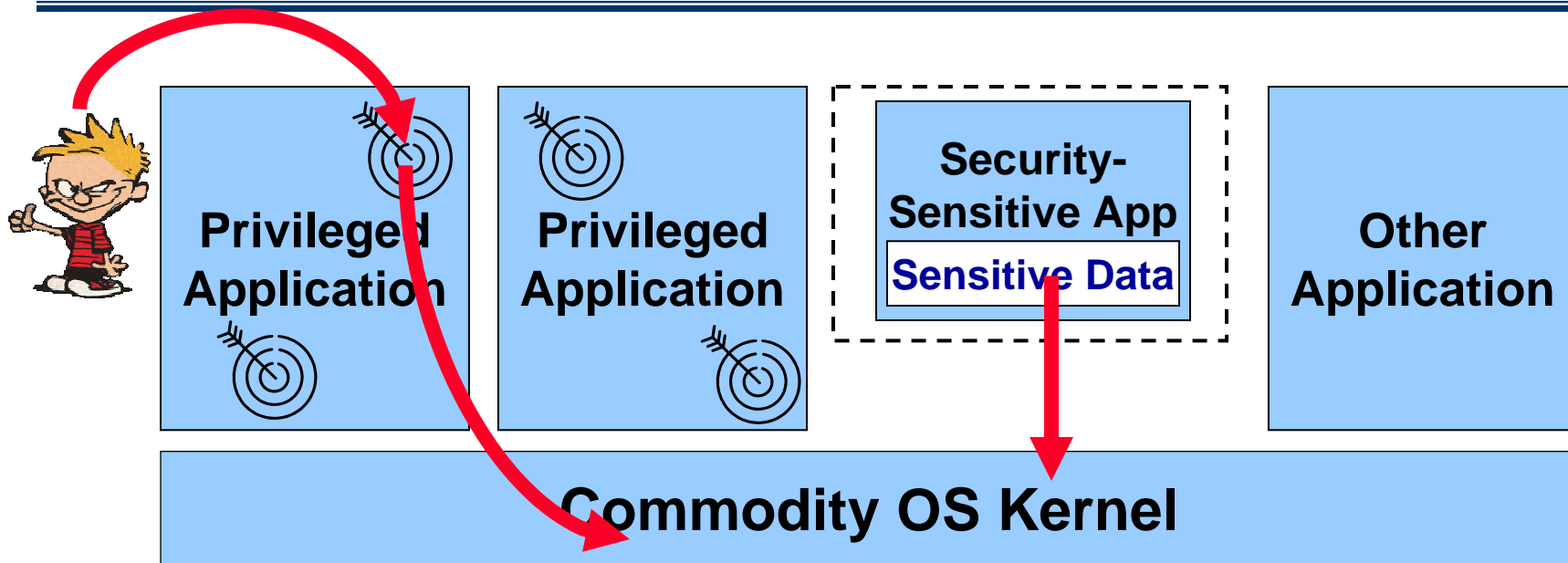
# Operating System Insecurity



- Commodity OS contain a lot of privileged code:
  - Gives attackers many opportunities to take control of OS



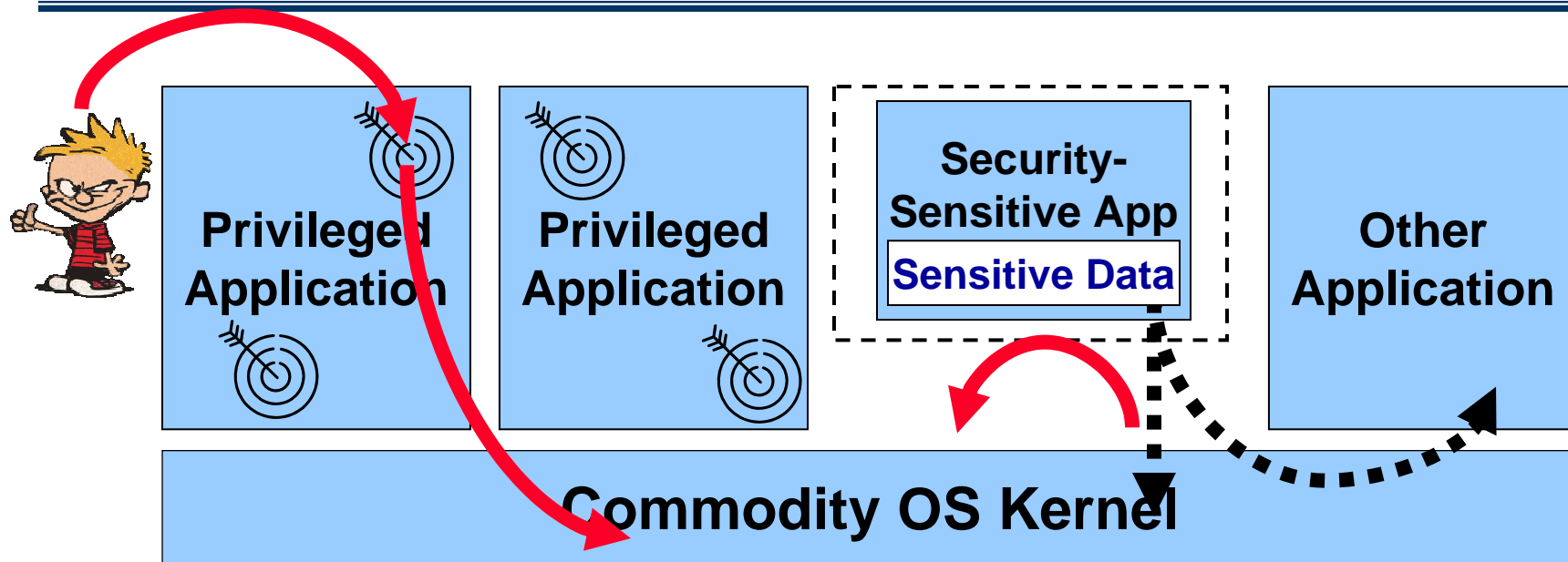
# Operating System Insecurity



- Commodity OS contain a lot of privileged code:
  - Gives attackers many opportunities to take control of OS



# Operating System Insecurity



- Commodity OS contain a lot of privileged code:
  - Gives attackers many opportunities to take control of OS

Should allow applications to control how they trust the OS



# Our Solution: Proxos

---

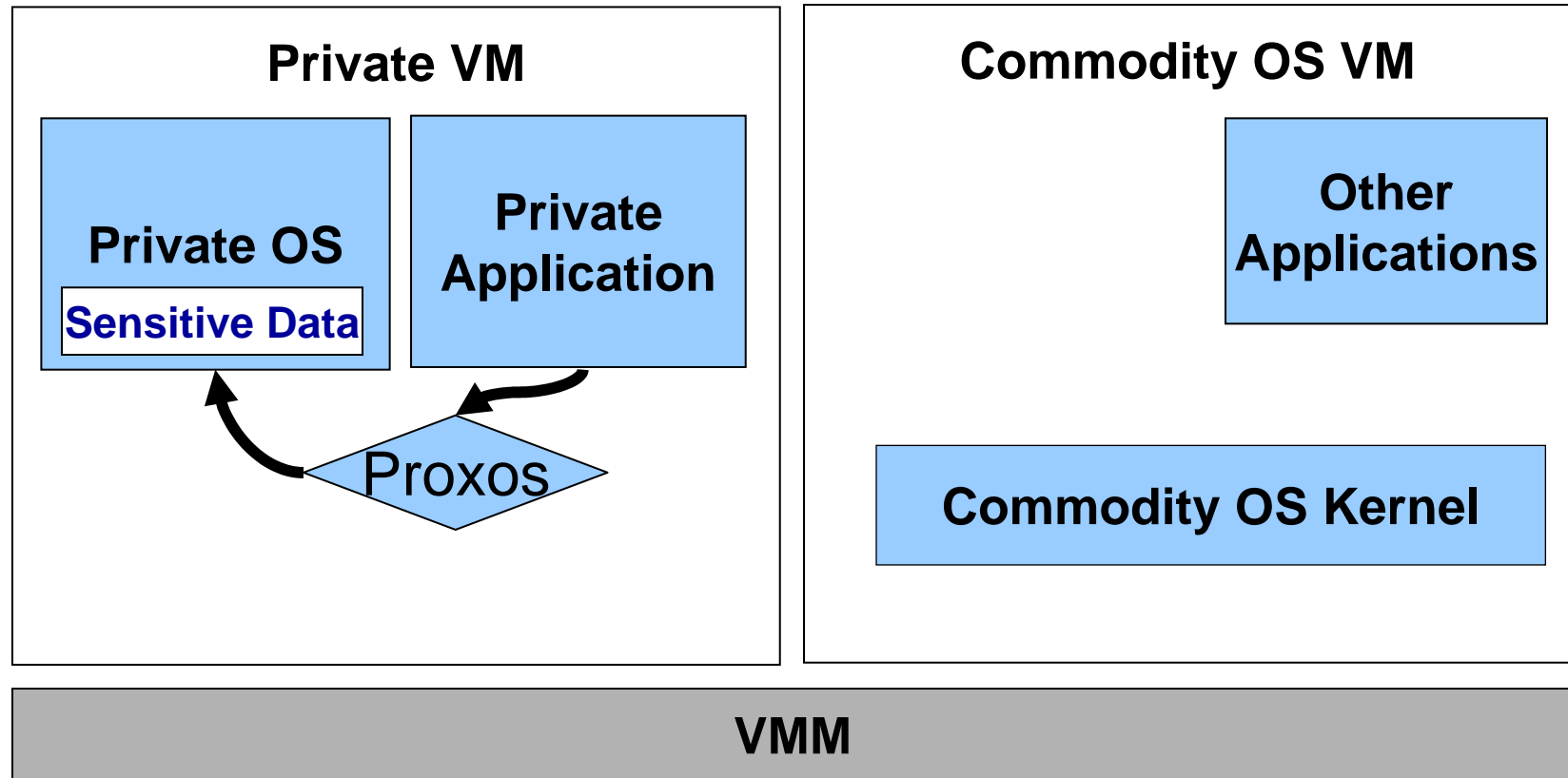
**Security-  
Sensitive  
Application**

**Other  
Applications**

**Commodity OS Kernel**

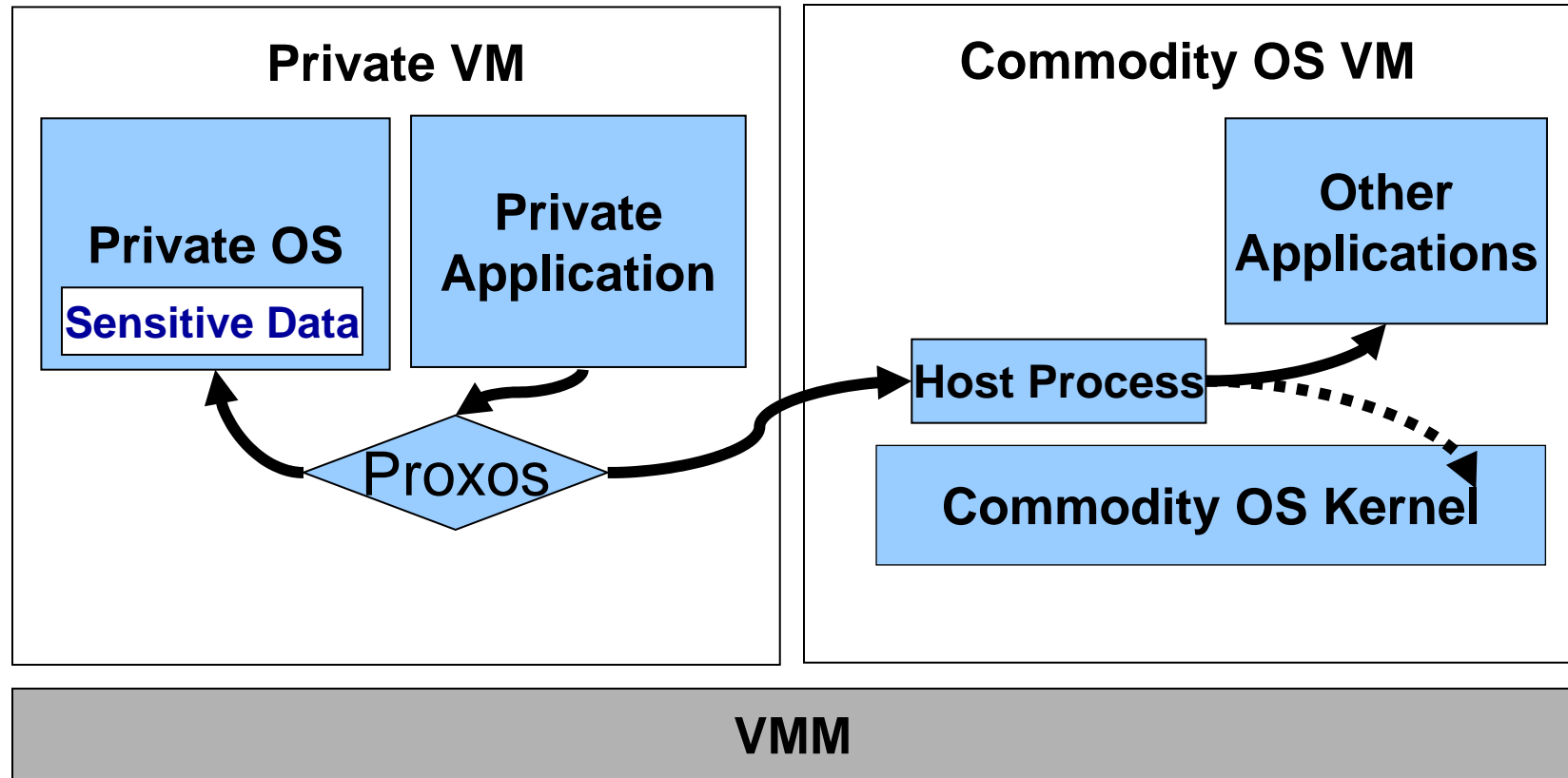


# Our Solution: Proxos



**Proxy Operating System (Proxos) routes system calls based on rules set by developer**

# Our Solution: Proxos



**Proxy Operating System (Proxos) routes system calls based on rules set by developer**

# Specifying System Call Routing Rules

- Proxos routes system calls based on the name of the resource, and the type of resource being accessed:

```
DISK:("/etc/shadow", priv_fs)
...
priv_fs = {
    .open = priv_open,
    .close = priv_close,
    .read = priv_read,
    .write = priv_write
}
```

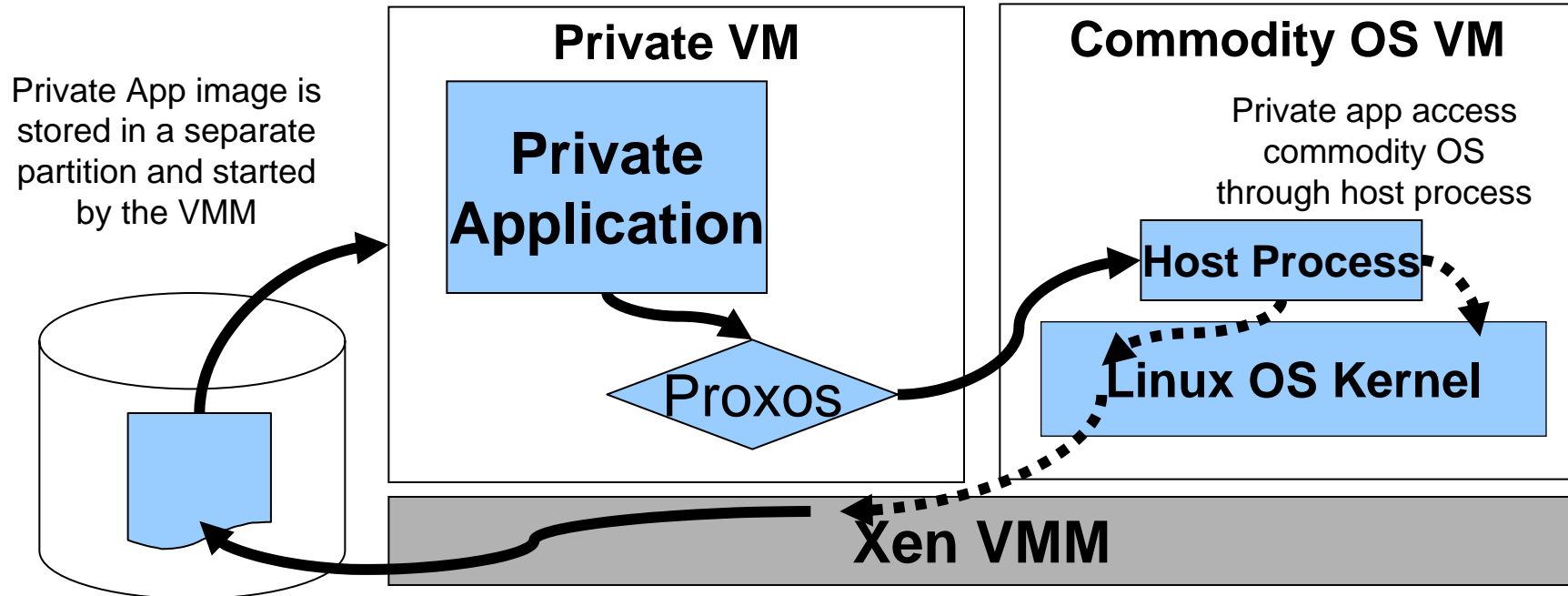
Example Routing Rules

- Rules link a method table (`priv_fs`) with name of resource (`/etc/shadow`)
- Method table has pointers to system call handlers in the private OS
- Resources not named in the rules are routed to commodity OS by default

**Interface is partitioned into accesses to sensitive and non-sensitive resources**



# Starting a Private Application

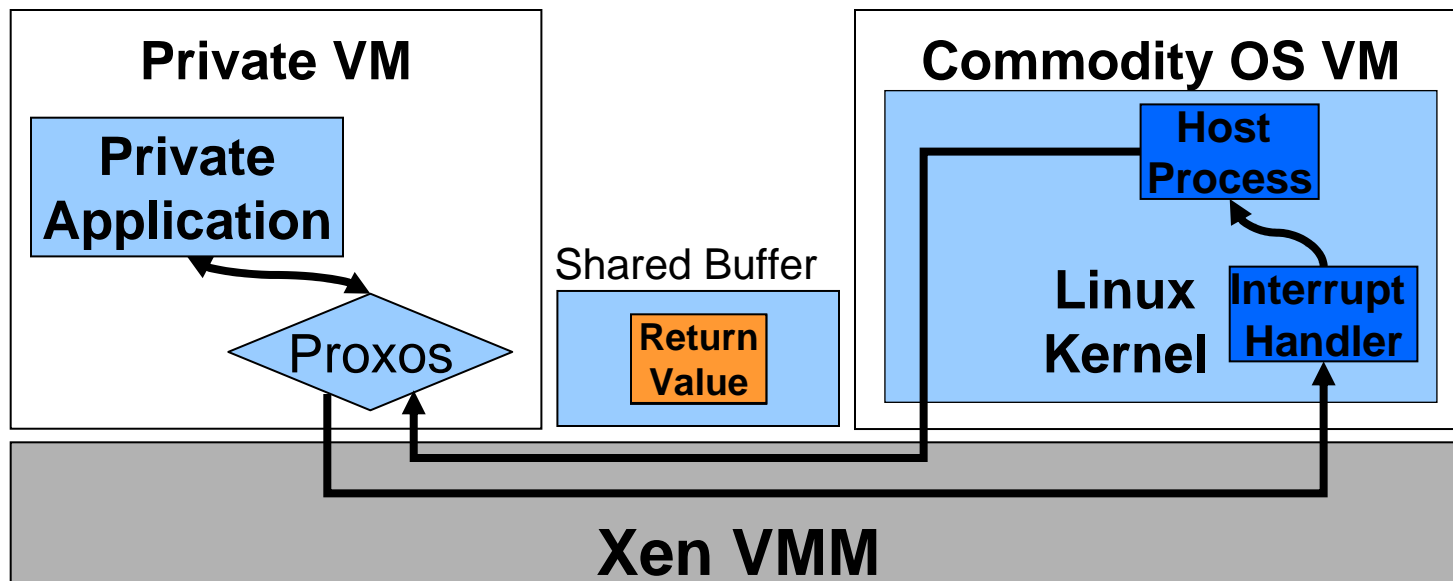


- Hypercall added to VMM for host process to start private app:
  - Private VM system calls executed same host that started it

**Commodity OS doesn't have to trust private app any more than it trusts a regular app**



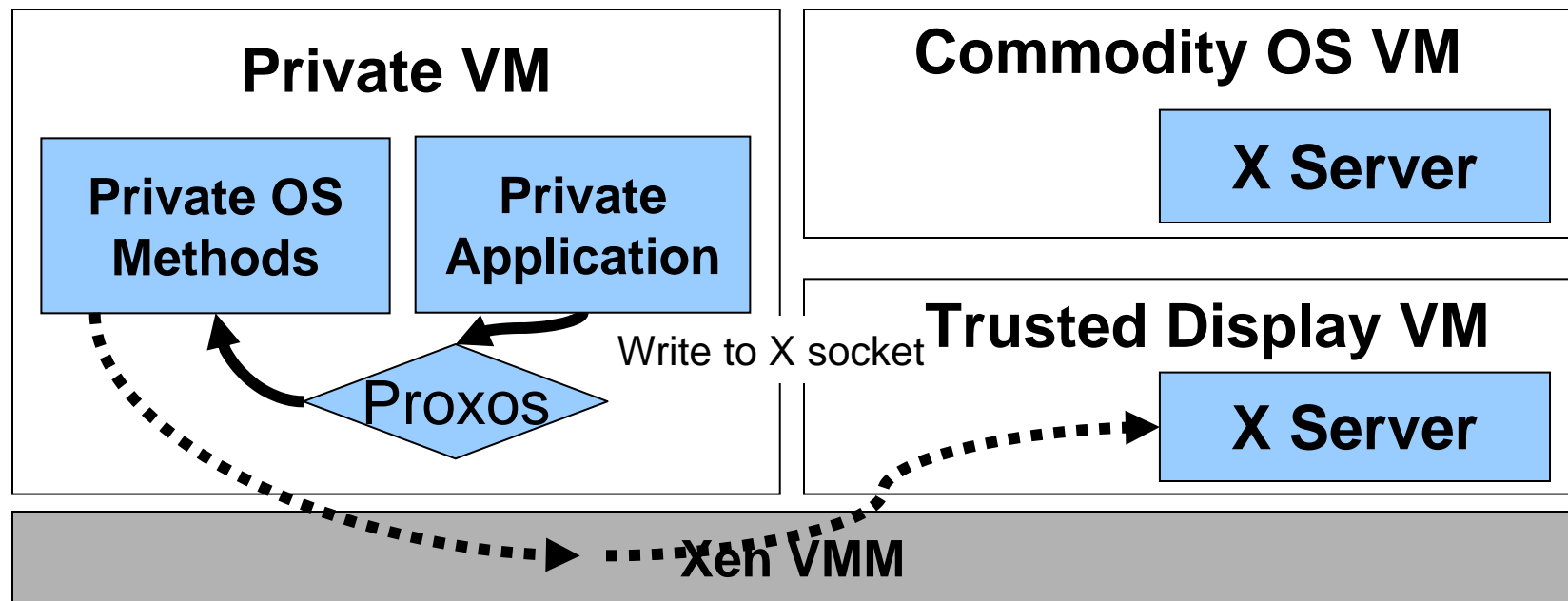
# Routing System Calls



- System calls routed to commodity OS using RPC's:
  - During startup, a shared memory region between the commodity OS and Proxos is created



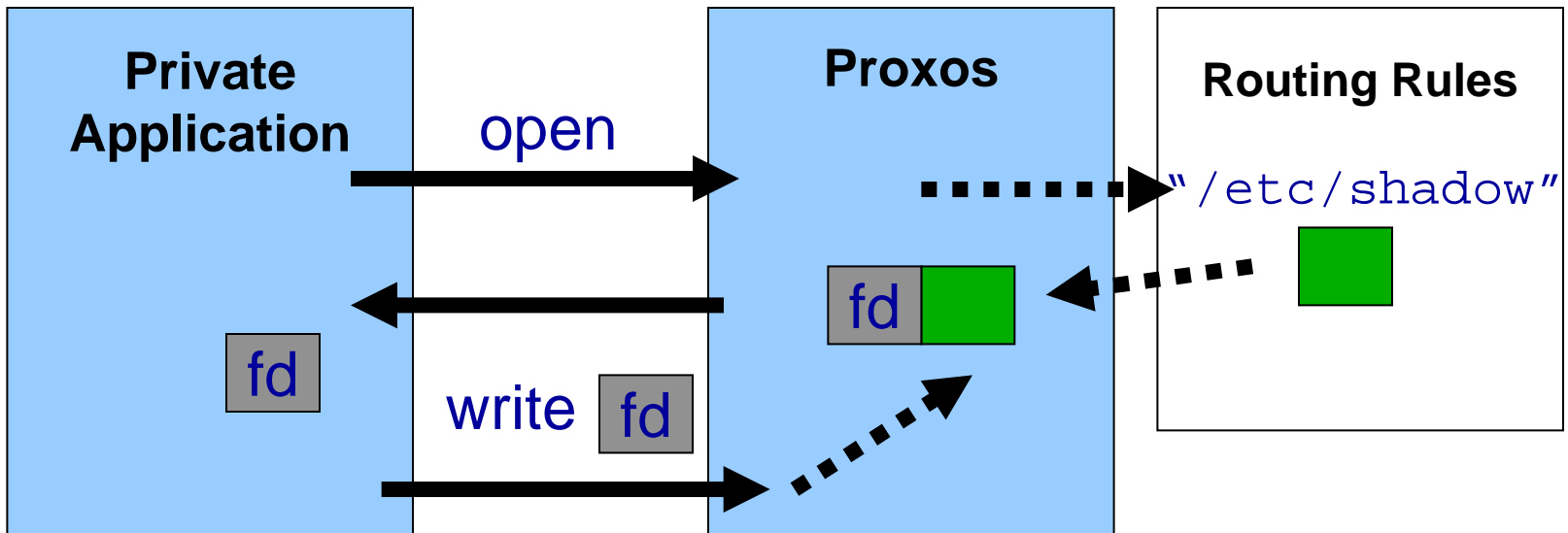
# User Interface



- To implement a trusted X interface:
  - System calls to X server socket are routed to the private OS
  - Private OS translates calls to trusted X server in another VM



# Proxos Prototype



- Proxos associates method tables with file descriptors:
  - When application open a resource, Proxos compares name with rules and binds methods to the descriptor
  - Methods are used in subsequent calls on the descriptor



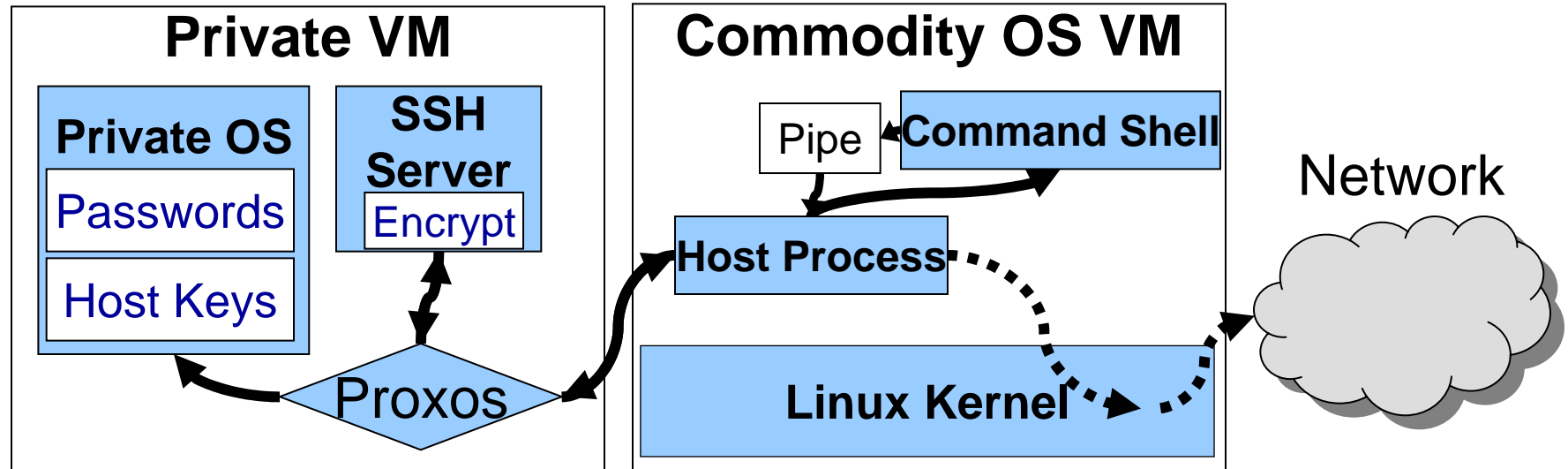
# Prototype Artefacts

---

- Proxos currently has artefacts that require minor porting for applications:
  - Single address space so forked code must be serialized
  - Not safe to dynamically load libraries from commodity OS, all code must be statically linked
  - No multi-threading support
- We ported three applications to Proxos:
  1. SSH server
  2. Web browser
  3. SSL web server



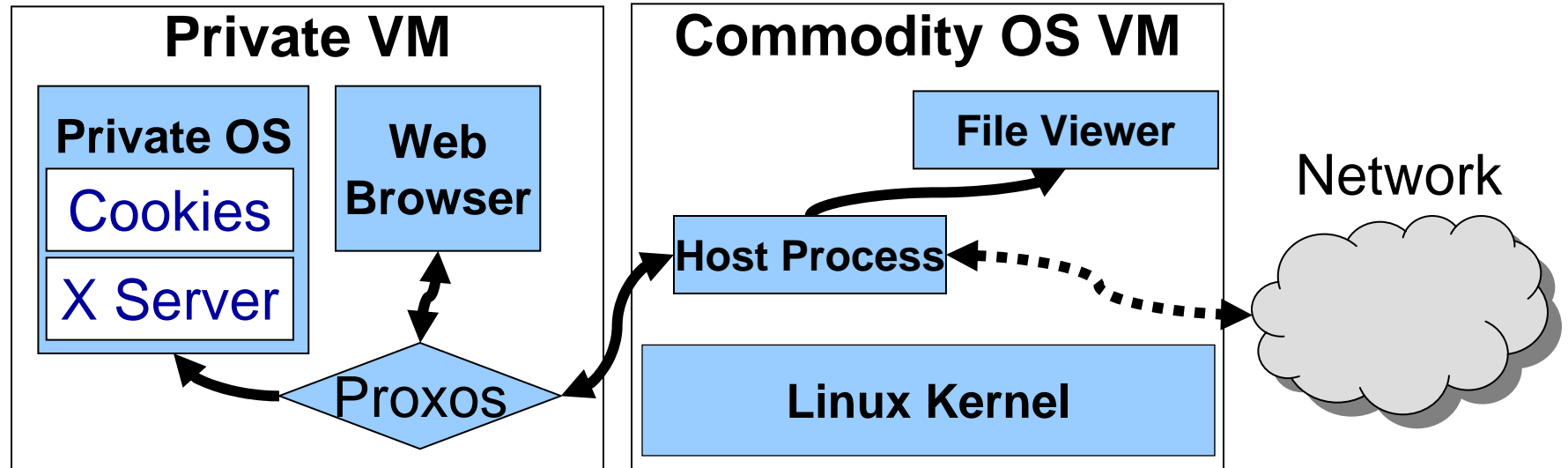
# Application: SSH Server



- Proxos allows applications to have access to commodity OS, but isolated sensitive resources at the same time. Ex SSH Server:
  - Sensitive data such as user passwords and the host key stored in private OS
  - All network packets decrypted in private app before sent to command shell



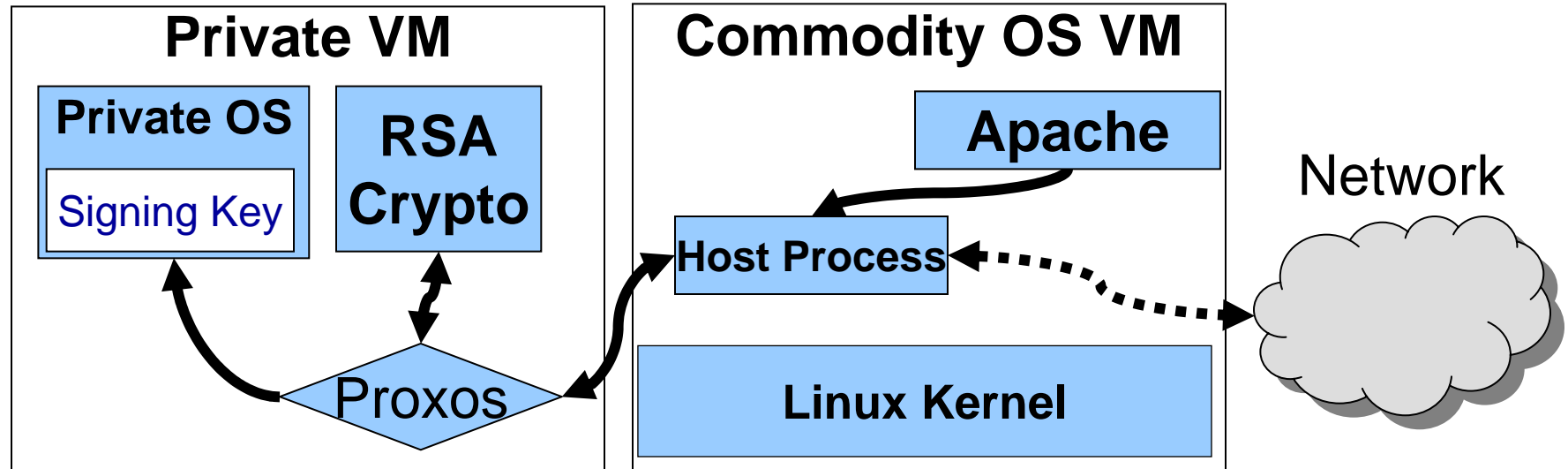
# Application: Graphical Web Browser



- Graphical Web Browser:
  - Sensitive user data and user interface is isolated from commodity OS
  - Browser can save downloaded files and invoke file viewers on the commodity OS



# Application: Apache & SSL



- Server Application: Apache & SSL extension
  - Private signing key isolated from commodity OS
  - To maintain performance, minimize private VM startup/shutdown by making host process persistent



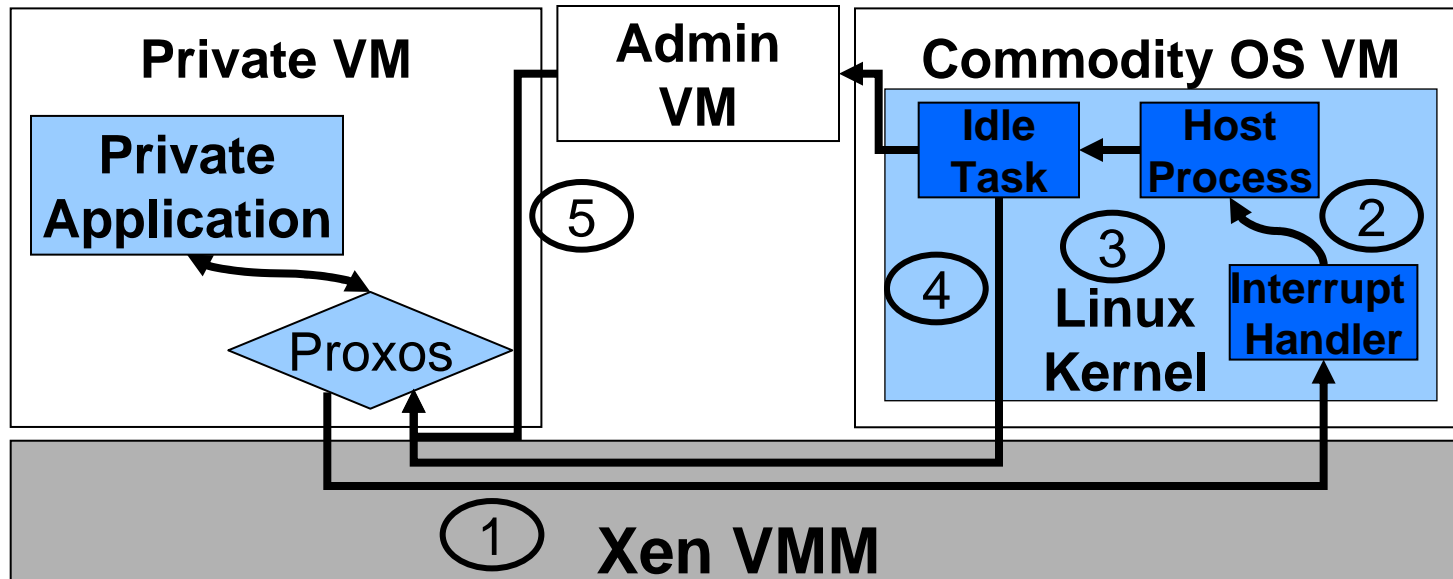
# Evaluation: Effort to Port Applications

Applications	Routing Rules (LOC)	App Modifications (LOC)
Web Browser	53	22
SSH Server	35	108
Apache & OpenSSL	28	556
Glibc	--	218

- Two tasks required to port applications:
  1. Identify sensitive resources and write routing rules
  2. Modify applications to account for differences between Proxos and commodity OS
- Routing rules are local to each private application:
  - Adding a new application does not affect rules of other apps



# Evaluation: Performance Overhead



- Overheads are dominated by forwarded system calls:
  - System calls experience 4 context switches
  - An additional context switch is caused by an artefact of our VMM configuration



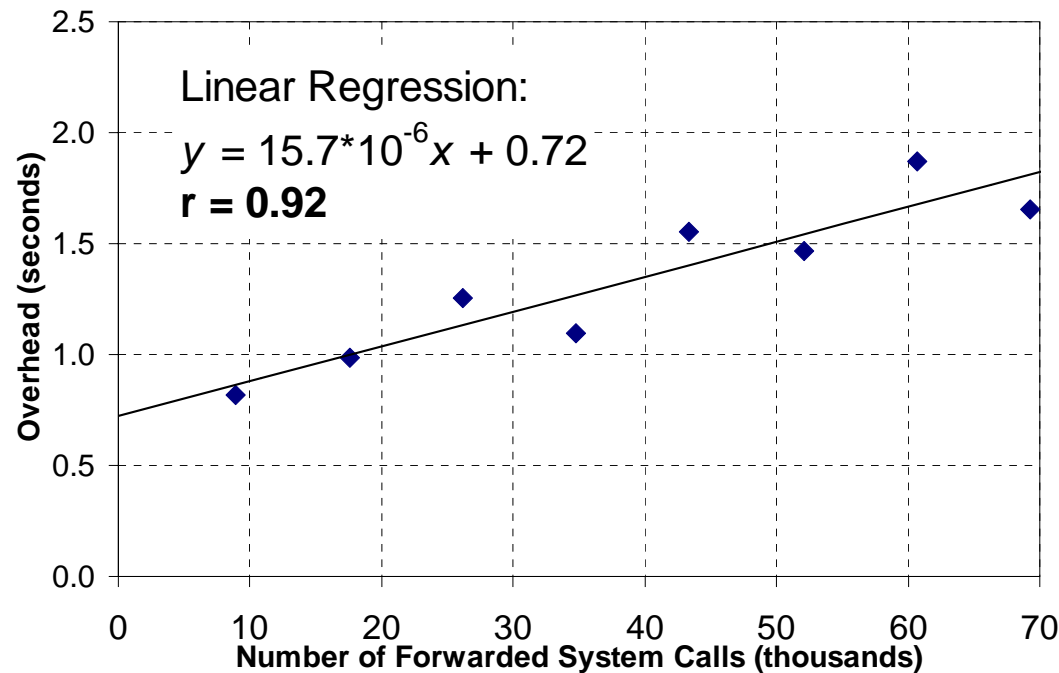
# Microbenchmarks

- Ran LMBench suite:
  - Minimal context switch time on our machine is 2.88 $\mu$ s
    - Expected 5 context switches gives 14.4  $\mu$ s
  - TLB misses resulting from context switches increase cost

Benchmark	Linux ( $\mu$ s)	Proxos ( $\mu$ s)	Overhead ( $\mu$ s)
NULL System Call	0.37	12.88	<b>12.51</b>
fstat	0.57	14.28	<b>13.71</b>
read	0.45	13.51	<b>13.06</b>
write	0.42	13.24	<b>12.82</b>
stat	8.76	25.98	<b>17.22</b>
open & close	14.57	47.18	<b>32.61</b>



# Application Benchmark



- Linear relationship: forwarded system calls is the dominant source of overhead
- Overheads overall are modest:
  - SSH experiences 6% slowdown and Apache even has a slight speedup



# Conclusions

---

- By partitioning system call interface and routing system calls:
  - Can isolate sensitive resources from commodity OS
  - Still allow application to use commodity OS resources
  - Applications do not lose functionality
- Porting effort is reasonable:
  - Routing rules are short and on the order of 10's of LOC
  - Rules are local to each application
- Performance:
  - Performance overheads are reasonable
  - Largely due to context switches and VM startup/shutdown
  - Some overheads can be removed by modifying application

