Blindfold: Confidential Memory Management by Untrusted Operating System

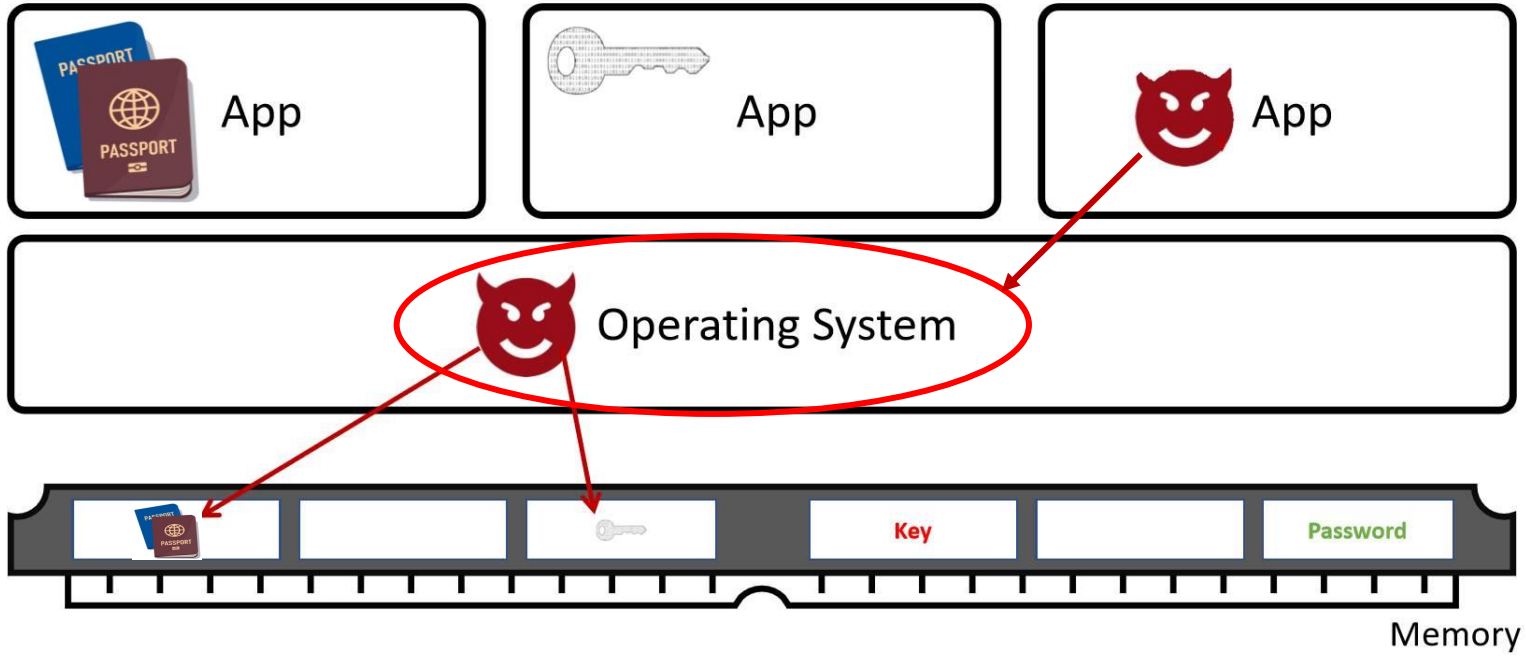
Caihua Li
Yale University
caihua.li@yale.edu

Seung-seob Lee
Yale University
seung-seob.lee@yale.edu

Lin Zhong
Yale University
lin.zhong@yale.edu

OS Is Not Trustworthy

- OS is complex & has large attack surface
  - Written in unsafe language & not certified
- Once compromised, attackers can steal app data



OS Has More Access Capabilities than It Needs

*A Librarian who **also reads** the books*

- **Management**: virtual memory & virtual CPU
- **Access**: physical frames & CPU registers

OS Has More Access Capabilities than It Needs

*A Librarian who **also reads** the books*

- **Management**: virtual memory & virtual CPU
- **Access**: physical frames & CPU registers

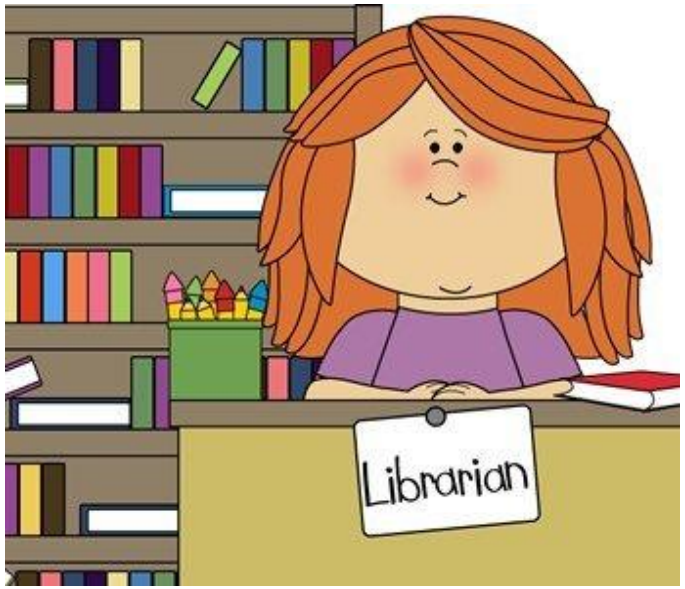
- OS ***does not care the value*** of data for management, e.g.,
  - Moving pages on memory paging
  - Moving contexts on interrupts / exceptions

Can OS Manage Memory without Access?

Answering it requires a deeper understanding of OS

*A Librarian who **also reads** the books*

?

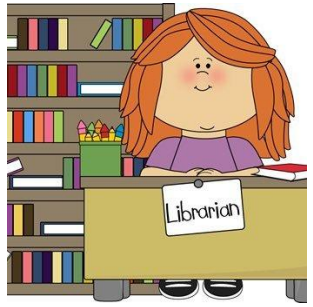


*A Librarian who **does not** read the books*

Linux Requires Only *Non-Semantic Access* to User Space for *Memory Management*

- Non-semantic access
  - OS ***does not care*** the value of data for management
  - E.g., paging, page migration, read / write system call, …

ssize_t write(int *fd*, const void **buf*, size_t *count*);



E.g., Move all the *books* from shelf #1 to #2

Linux Still Requires *Semantic Access* to User Space *beyond Memory Management*

- Non-semantic access
  - OS ***does not care*** the value of data for management
  - E.g., paging, page migration, read / write system call, …

ssize_t write(int *fd*, const void **buf*, size_t *count*);

E.g., Move all the *books* from shelf #1 to #2

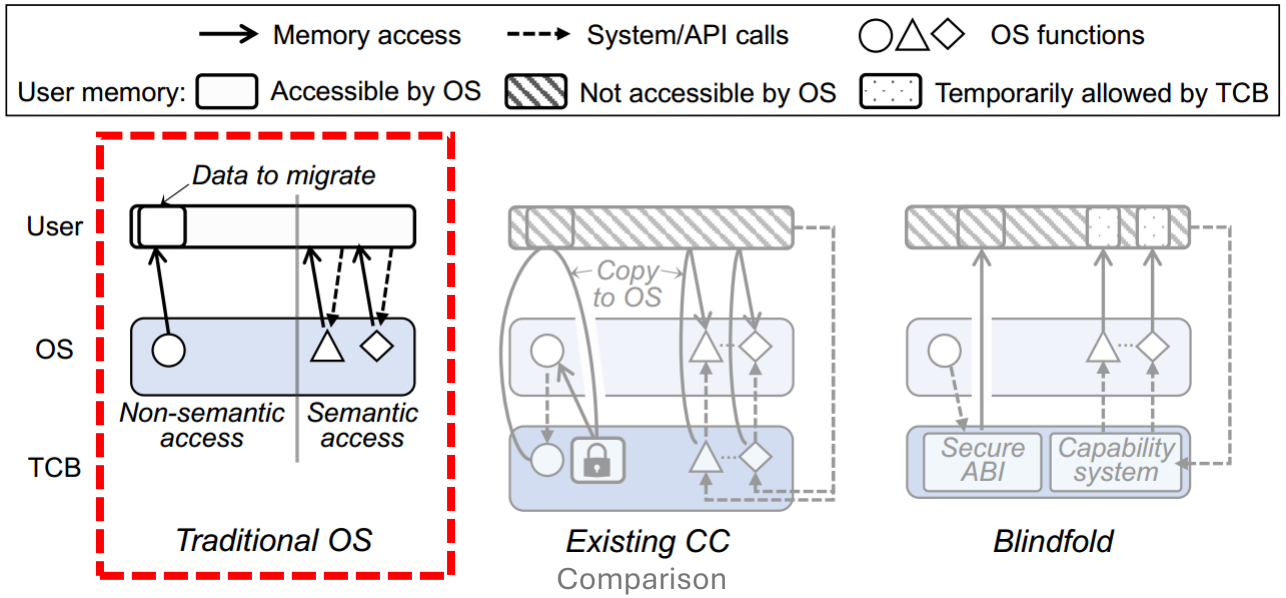
- Semantic access
  - OS ***needs the value*** of data to fulfill its job
  - E.g., syscall arguments, futex, signal handling, …

int open(const char **pathname*, int *flags*);

E.g., Locate and open the chapter about memory paging in an OS textbook

User-space Access in Traditional OS

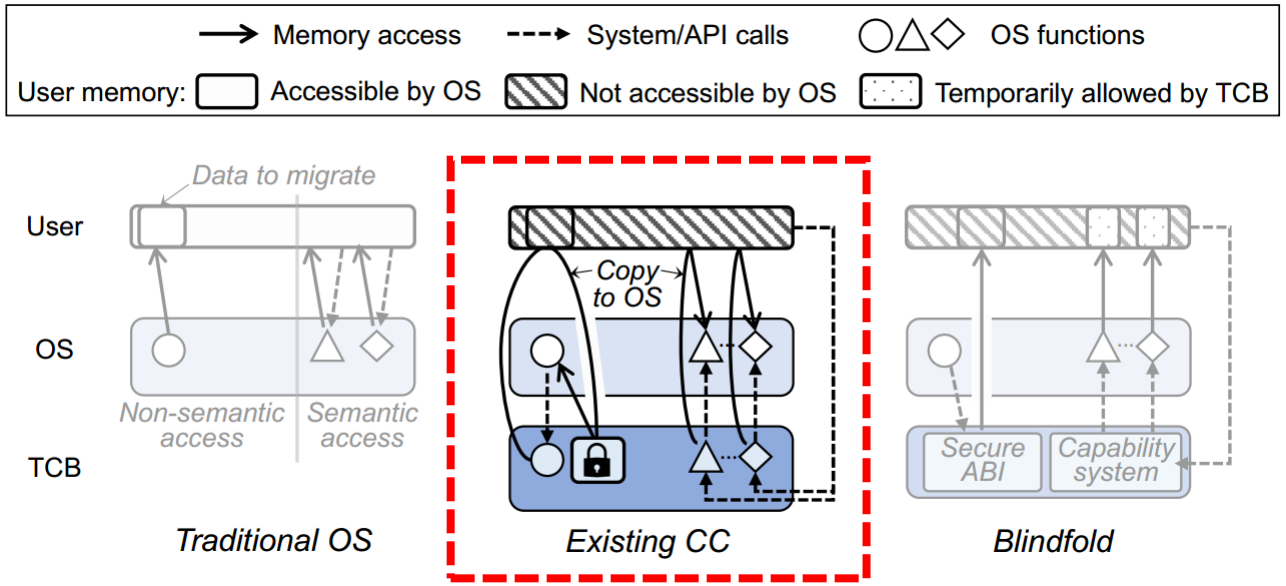
- Non-semantic: clear / copy pages
- Semantic: system call parameters & signal handling
- Direct access to user-space: ***efficient but insecure***



Comparison

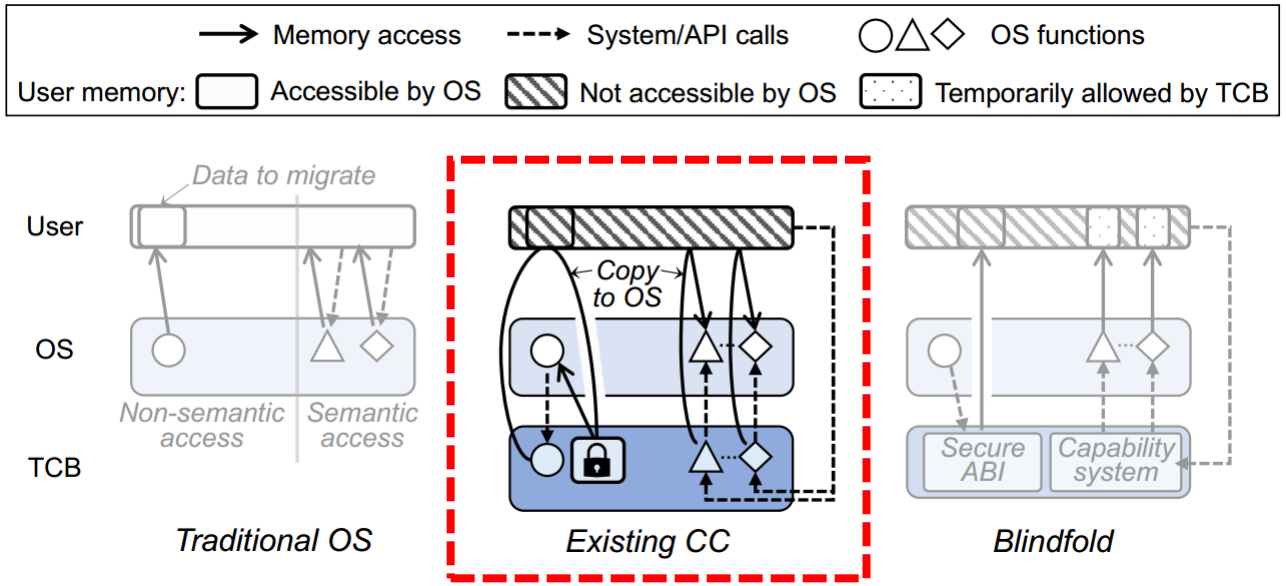
User-space Access in Prior Work

- **Non-semantic**: clear / copy pages
  - Always provide an encrypted view, or hide user's private data from OS
  - *Encryption is **expensive**, or the OS's **optimizations stop functioning***
- Semantic: system call parameters & signal handling



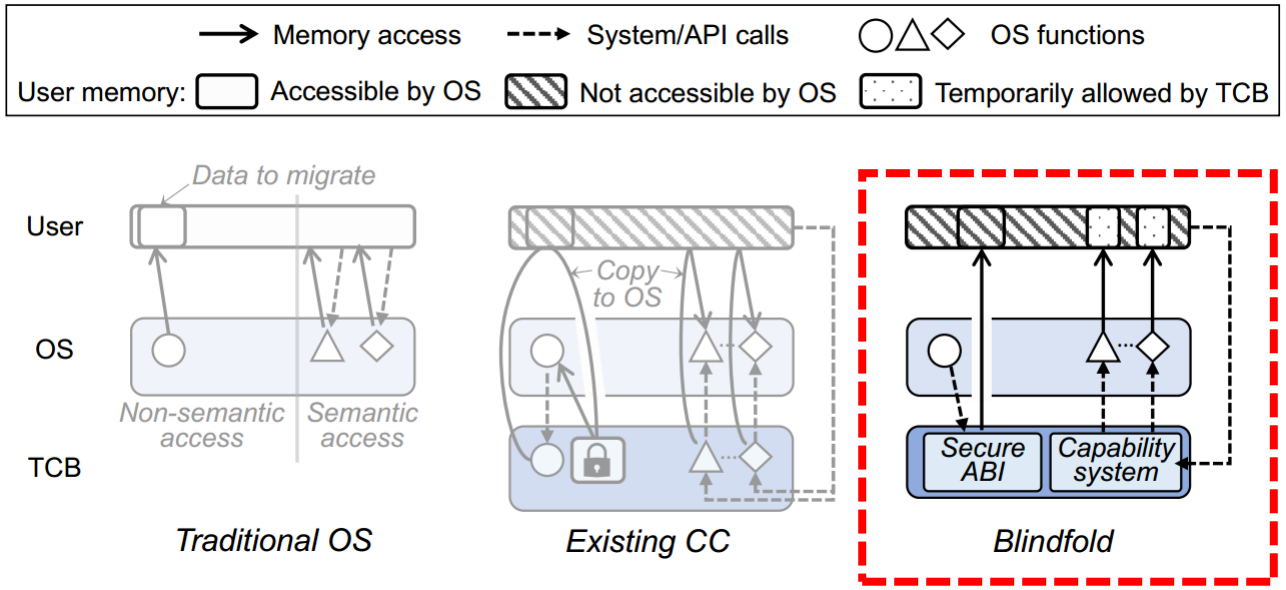
User-space Access in Prior Work

- Non-semantic: clear / copy pages
- **Semantic**: system call parameters & signal handling
  - Copy to buffer & need ***case-by-case handlings*** for signal / syscalls like futex
  - *Extra data copy &* ***complex TCB***



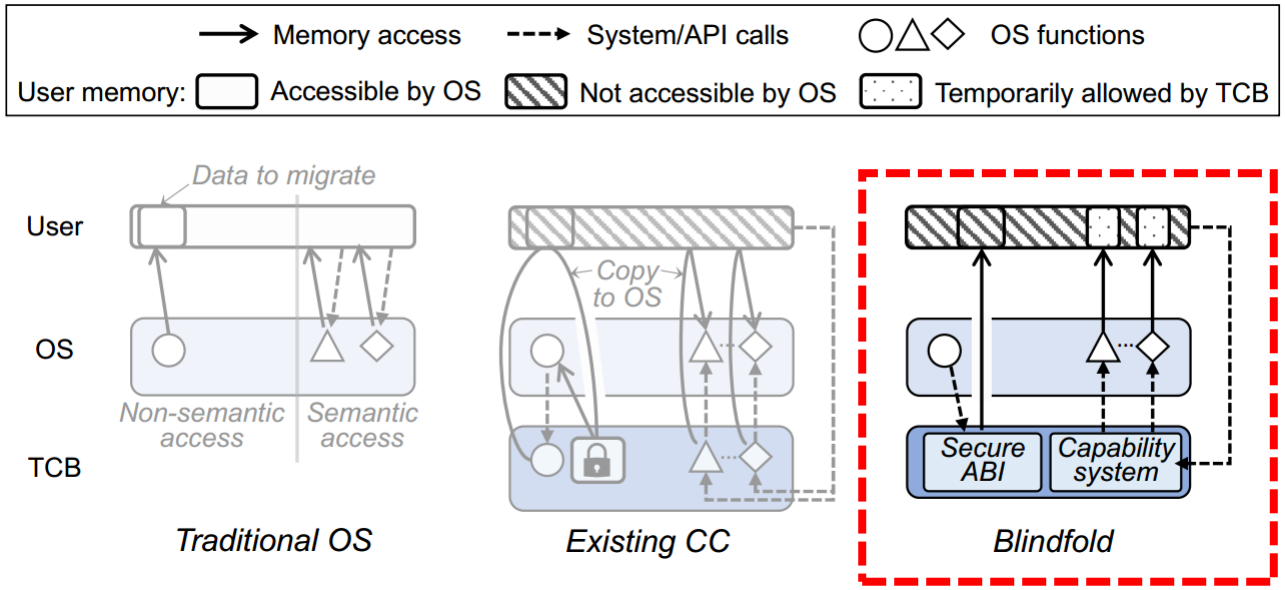
Key Design I: User-space Access in Blindfold

- **Non-semantic**: clear / copy pages
  - For IO and swapping: provide OS an encrypted view
  - Moving within memory: trigger TCB to operate pages on behalf of OS
  - Allow OS to ***manage sensitive user pages*** & encrypt data ***only when necessary***
- Semantic: system call parameters & signal handling

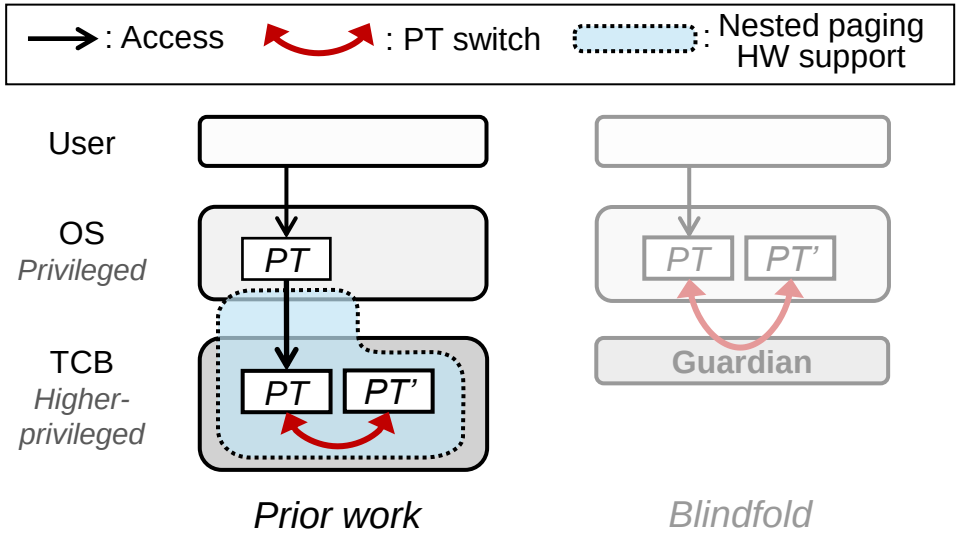


Key Design I: User-space Access in Blindfold

- Non-semantic: clear / copy pages
- **Semantic**: system call parameters & signal handling
  - Capabilities are created / destroyed before / after syscalls & exceptions
  - The TCB copies objects on behalf of OS after capability checks
  - Provide a ***universal mechanism*** & no extra data copy

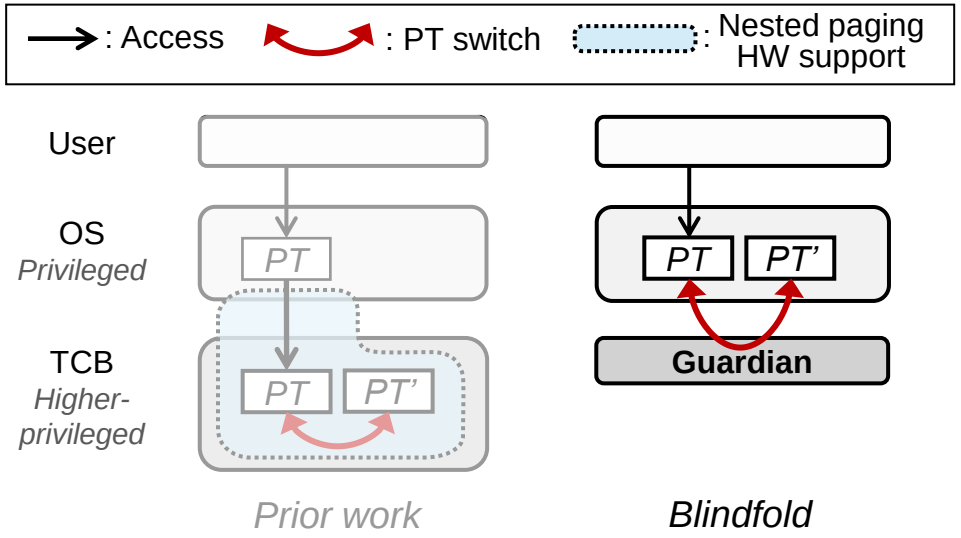


Key Design II: Switching Instead of Nesting



- Previous works leverage virtualization
  - Switch between two nested page tables to provide different views of memory
  - *Increase TCB & require nested paging HW*

Key Design II: Switching Instead of Nesting



- Previous works leverage virtualization
  - Switch between two nested page tables to provide different views of memory
  - *Increase TCB & require nested paging HW*

- Blindfold employs mediation
  - Mediate page tables & switch between them
  - Mediate control flow via switching exception vector tables
  - Small TCB (about half of that in prior systems)

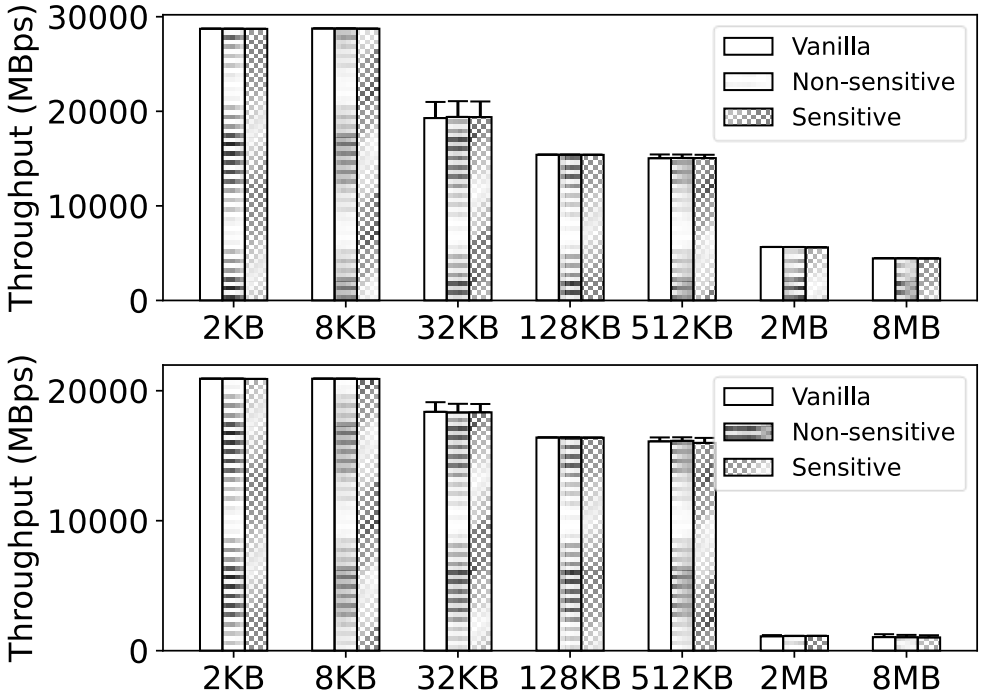
Evaluation: Questions to Answer

- Memory related overhead, e.g., page table mediation
  - Application-side – memory access overhead
  - OS-level – memory management overhead
    - Memory paging
    - Encrypting sensitive user pages
    - Optimization for reducing encryption overhead
- System call-related overhead, e.g., capability check
  - In memory-intensive and I/O-intensive applications
  - Optimization for improving system call performance

(More details in the paper)

Evaluation: Memory Access

- For common case: Memory access has negligible overhead
  - For both non-sensitive and sensitive applications

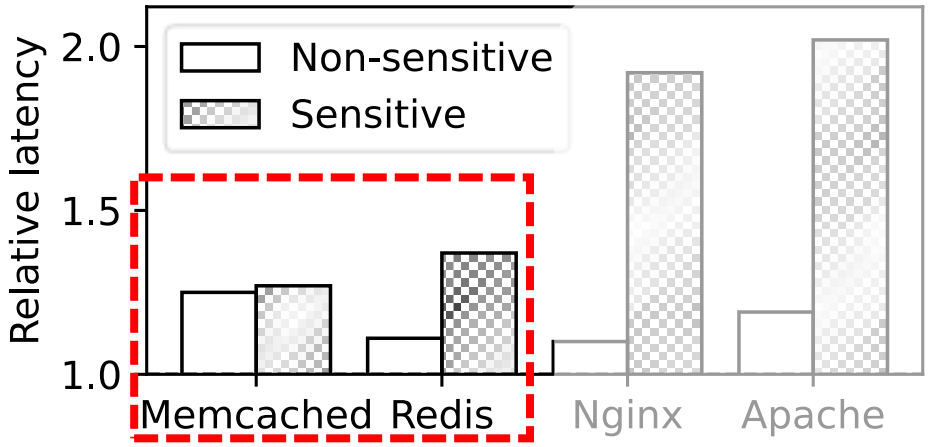


Memory Read

Memory Write

Evaluation: Memory Paging

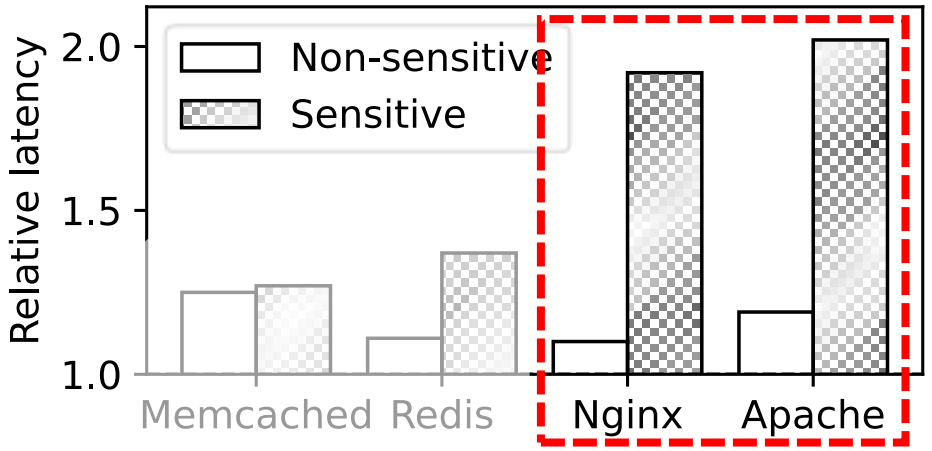
- For memory intensive applications like Memcached & Redis
  - Typically invoke system calls far less frequently than IO intensive apps
- Overhead of memory paging is about 10%~20%



Memory Paging Overhead

Evaluation: Semantic Access

- For IO intensive applications like Nginx & Apache
- Semantic access for system call parameters incurs high overhead (e.g., 80%) when involving large number of system calls



Semantic Access Overhead

Takeaways

- OS is *not trustworthy* & has *more access than necessary*
- Linux needs only *non-semantic access* for *memory management*, but requires *semantic access* for tasks like handling system calls
- Blindfold uses a general capability system to limit semantic access
- Blindfold uses page table mediation & switching *to* ***let OS manage memory securely without knowing its content***