EPK: Scalable and Efficient Memory Protection Keys

Jinyu Gu, Hao Li, Wentai Li, Yubin Xia, Haibo Chen

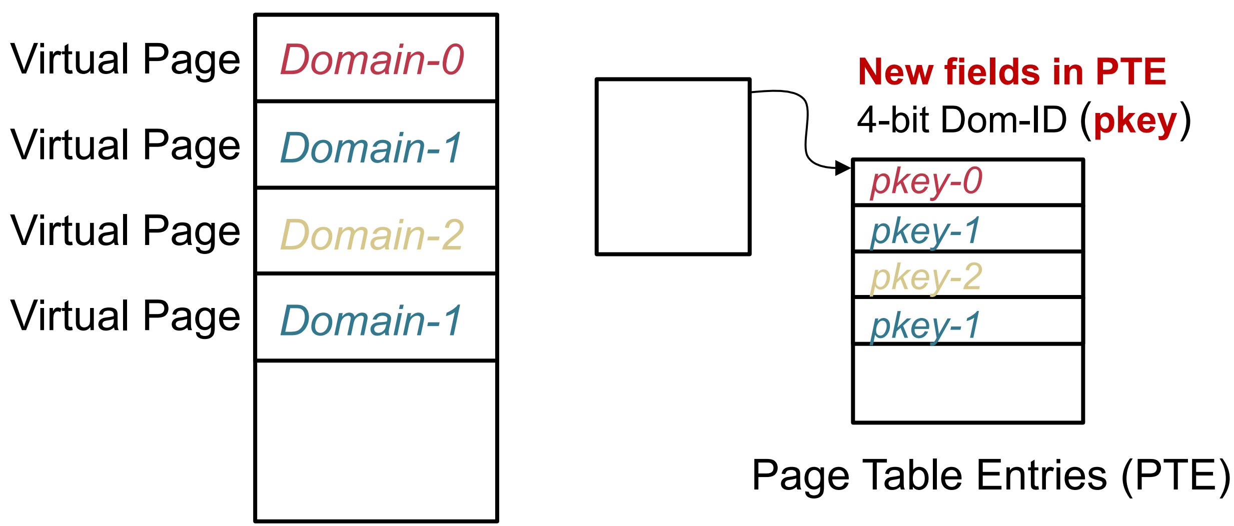
Institute of Parallel and Distributed Systems (IPADS), Shanghai Jiao Tong University

Memory Protection Key (MPK)

- **A hardware feature for intra-process memory isolation**
  - It has been introduced into Intel CPUs since 2019

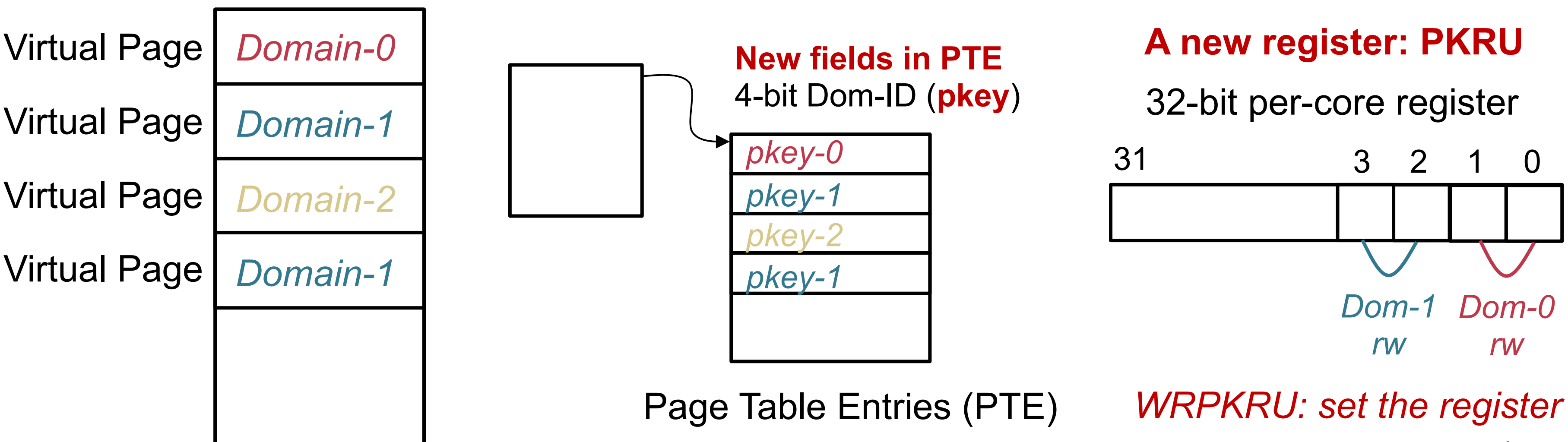
Memory Protection Key (MPK)

- **A hardware feature for intra-process memory isolation**
  - Partition different memory domains within one address space



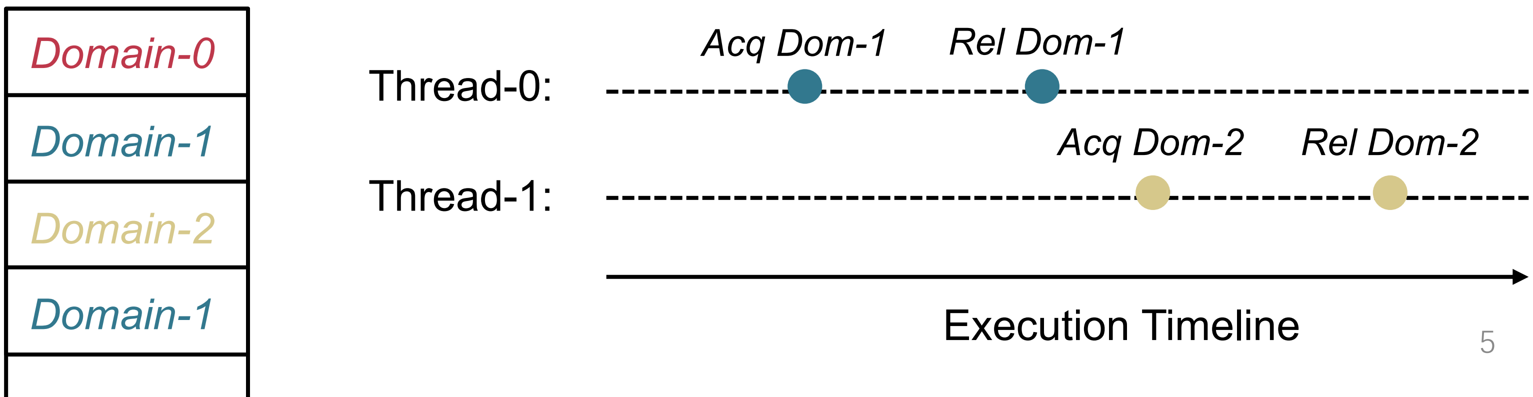
Memory Protection Key (MPK)

- **A hardware feature for intra-process memory isolation**
  - Partition different memory domains within one address space
  - Allow different threads have different memory views



Common Usage Model

- Create memory domains for separating the memory data
- A thread acquires/releases the access permission of one specific domain before/after accessing the data in it
- Acquire/release the domain access permission is efficiently achieved in the user mode (domain switching)

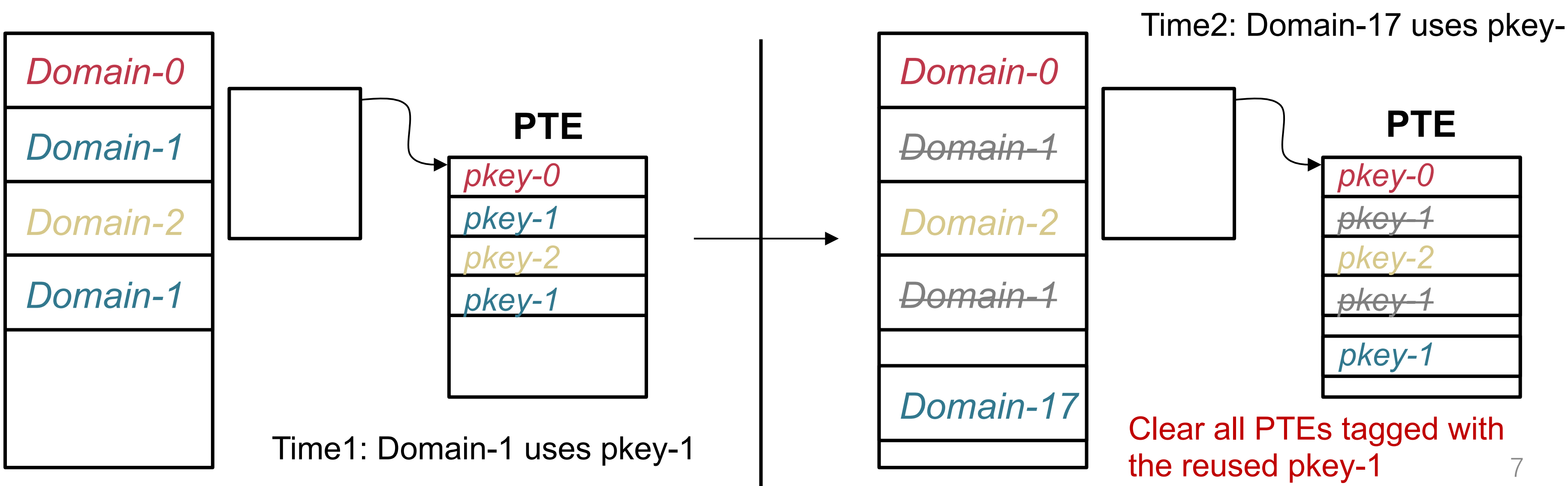


The Limitation of MPK

- **Contradiction**
  - Small (16) MPK domain number vs. Scalable domain requirements
- 😒 **Introduce isolation among different clients**
  - The client number is usually far more than 16
- 😒 **Introduce isolation to persistent memory data**
  - More domains can benefit stray access protection
- 😒 **Introduce isolation to more mutual-distrust modules**
  - Both applications and system software may contain more than 16 components that need to be isolated

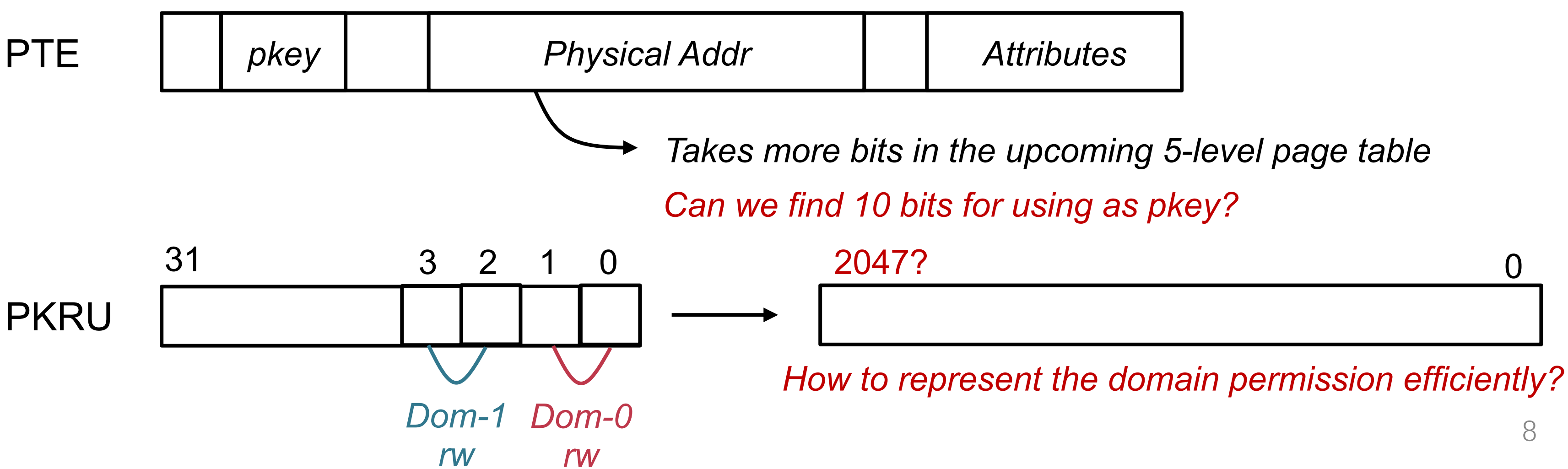
Candidate Solution 1

- **Solution-1: Resource time-division multiplexing**
  - Letting different domains use the same pkey at different times
  - **Performance issue**: Domain switching requires modifying page table entries and TLB flushing



Candidate Solution 2

- **Solution-2: Using 10 bits in the PTE as the pkey field**
  - 10 bits pkey can offer 1,024 memory domains
  - **Compatibility issue**: Non-achievable in existing CPUs; significantly reduce the usable bits in the PTE; PKRU register



Our Idea: EPK

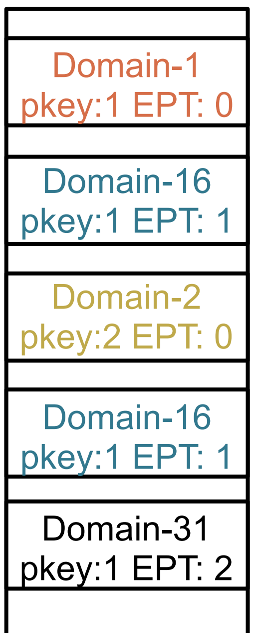
- **Design goals**
  - 😊 Scalable number of memory domains
  - 😊 No hardware modifications
  - 😊 Low performance overhead (support scalable memory domain size)
- **Observation**
  - MPK and VMFUNC share similarities in decoupling privilege-mode management and non-privilege-mode fast switching
- **Idea**
  - Reuse the same pkey in different extended page tables (EPT)
  - EPK: Extended Protection Keys

A Quick Intro of VMFUNC

- **EPT: Extended Page Table**
  - Map guest physical addresses (GPA) to host physical addresses (HPA)
  - Managed by the hypervisor
- **VMFUNC: A hardware virtualization extension**
  - VM functions: EPT switching in the VM
  - Load one EPT from a list of 512 EPTs configured by the hypervisor
  - No TLB flushing is needed when executing VMFUNC

Basic Design of EPK

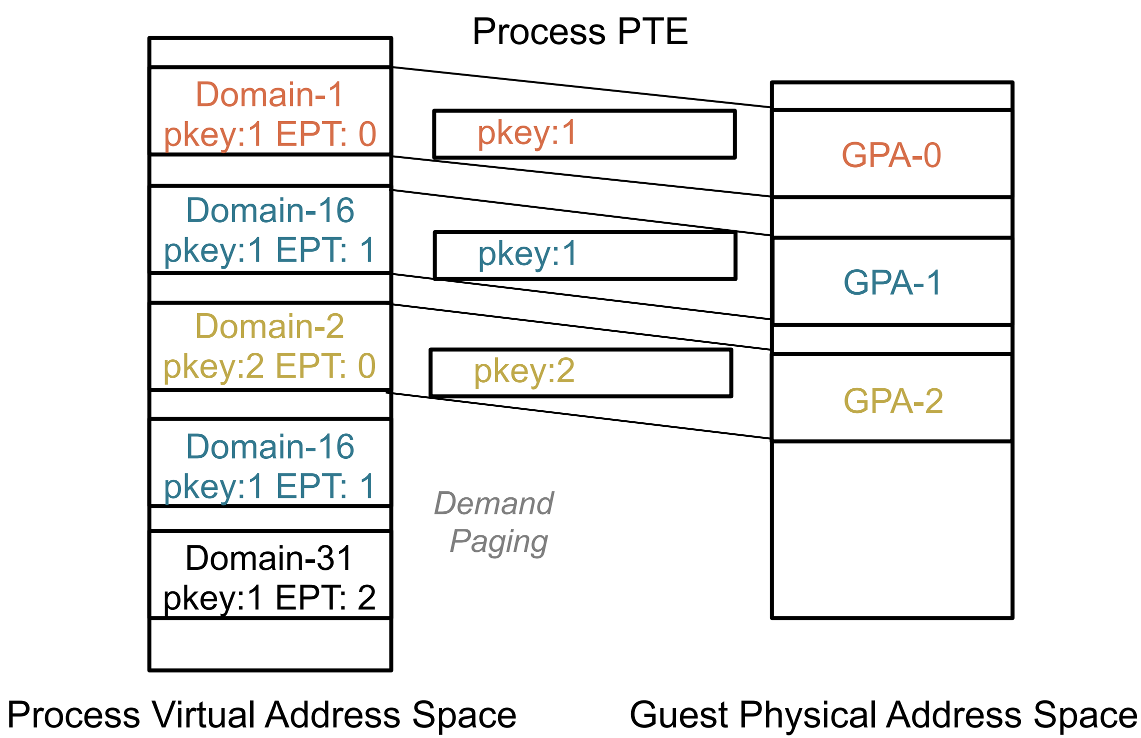
- **Use the extended protection key as the domain ID**
  - Domain ID: EPT-index (0-511) + pkey (1-15)



Process Virtual Address Space

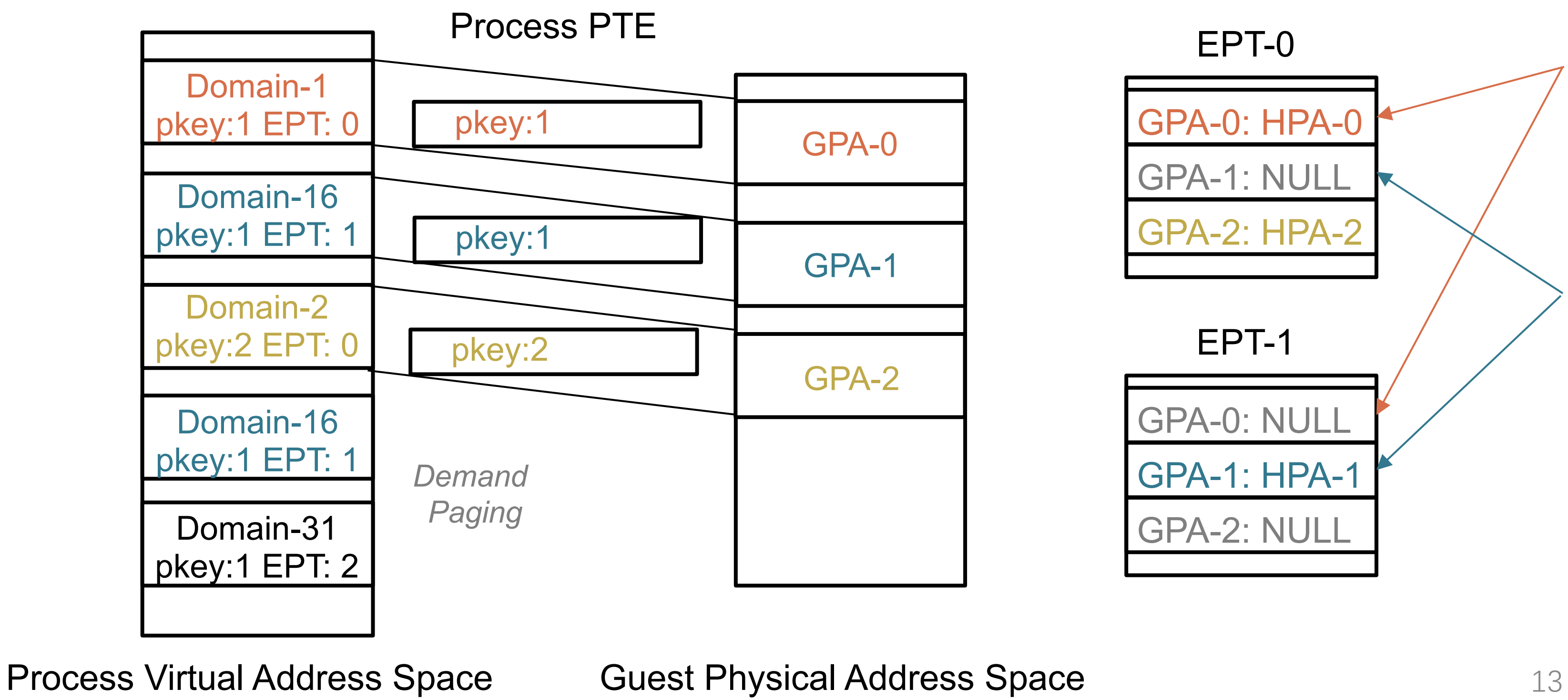
Basic Design of EPK

- **Use the extended protection key as the domain ID**
  - Domain ID: EPT-index (0-511) + pkey (1-15)



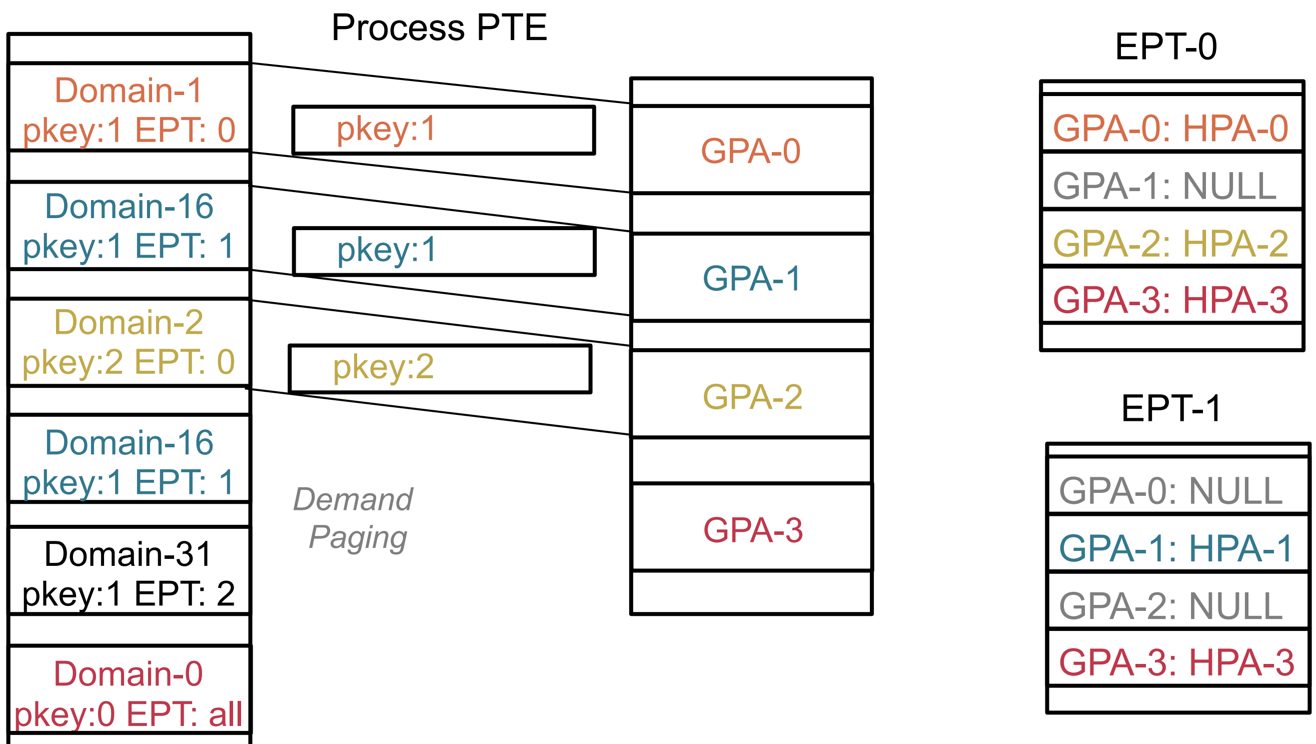
Basic Design of EPK

- **Use the extended protection key as the domain ID**
  - Domain ID: EPT-index (0-511) + pkey (1-15)



Basic Design of EPK

- **Use the extended protection key as the domain ID**
  - Domain ID: EPT-index (0-511) + pkey (1-15)



APIs

```
/* Allocate domain IDs with affinity */
int alloc_domains(int num, int dom_ids[]);

/* Free domain IDs */
int free_domains(int num, int dom_ids[]);

/* Allocate a virutal memory range for a domain */
void *domain_mmap(int dom_id, void *addr, size_t len,
                 int prot, int flags);

/* Remove some mappings */
void domain_munmap(void *addr, size_t len);

/* Retrieve the access permission of a domain */
void domain_begin(int id, int prot);

/* Release the domain permission */
void domain_end(int id);
```

Challenges

- **Challenge-1: EPT management**
  - How to make a VM seamlessly run with different EPTs
  - How to bridge the semantic gap for filling EPT mapping
- **Challenge-2: Multi-domain access**
  - How to transparently access multiple domains across different EPTs

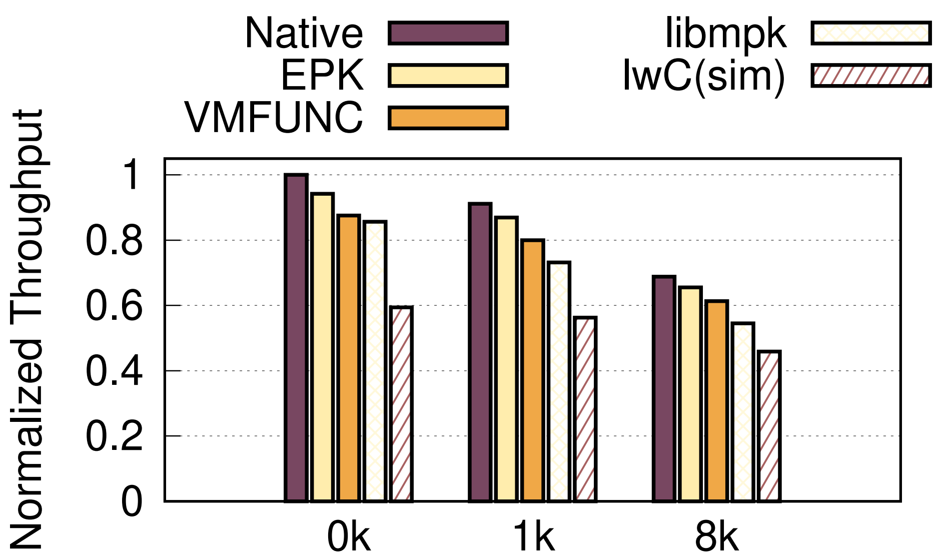
Please refer to the paper if interested

Evaluation Setup

- **Platform**
  - Dell PowerEdge R640 Server
  - Intel Xeon Gold 6138 CPU (HT disabled, 2.0GHz fixed)
  - EPK implemented on Linux/KVM-4.19.88
- **Comparison**
  - Vanilla VM process (with no isolation)
  - A VMFUNC solution (Use one EPT as one domain)
  - libmpk (time-division MPK pkey multiplexing) *[ATC 19]*

Case-Study: Intra-Process Isolation

NGINX v1.12.1: One worker thread
Workload: Use ab generate 300 clients sending file requests
Isolation: isolate each client's session key in one domain



Overhead

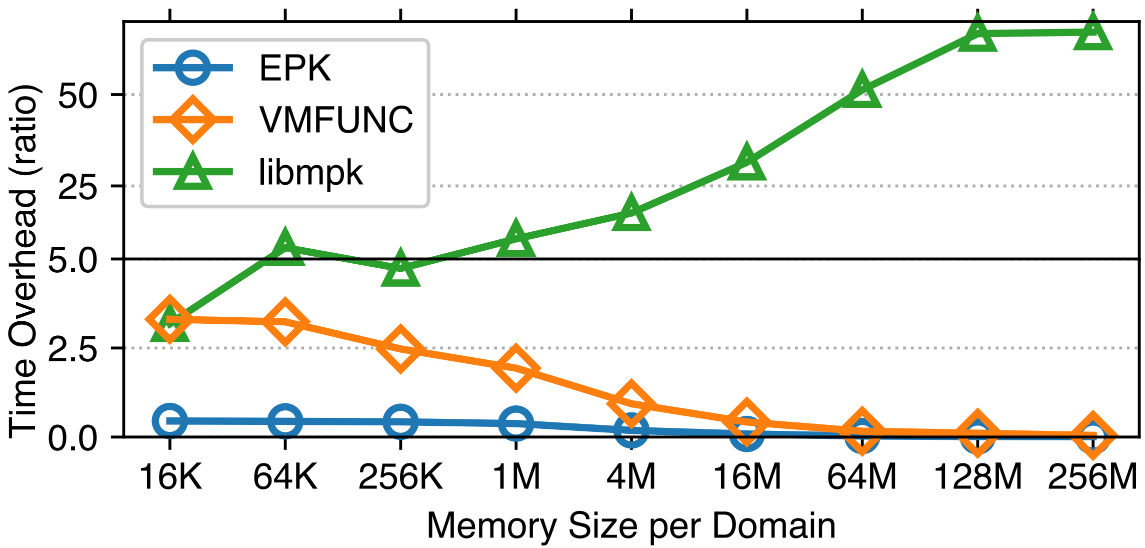
EPK: 5%

VMFUNC: 12%

libmpk: 18%

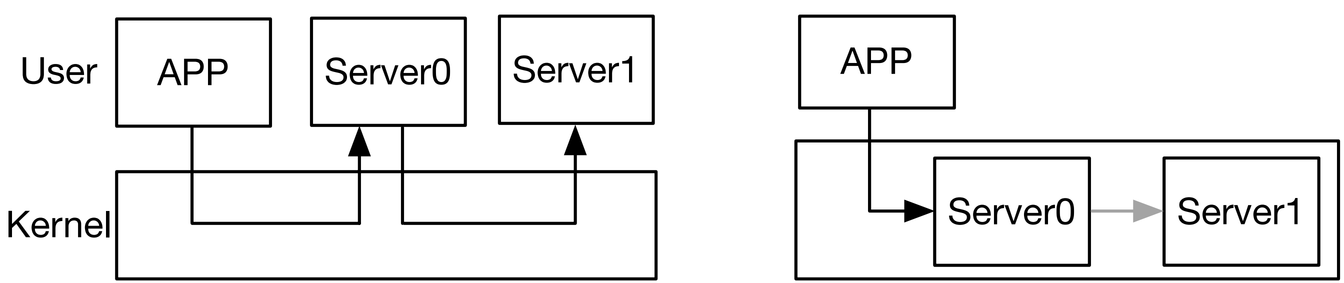
Case-Study: Memory Data Isolation

Hash-table
Workload: switch to random domain repeatedly and operate the hash-table in it
Isolation: separate each hash table in one individual memory domain



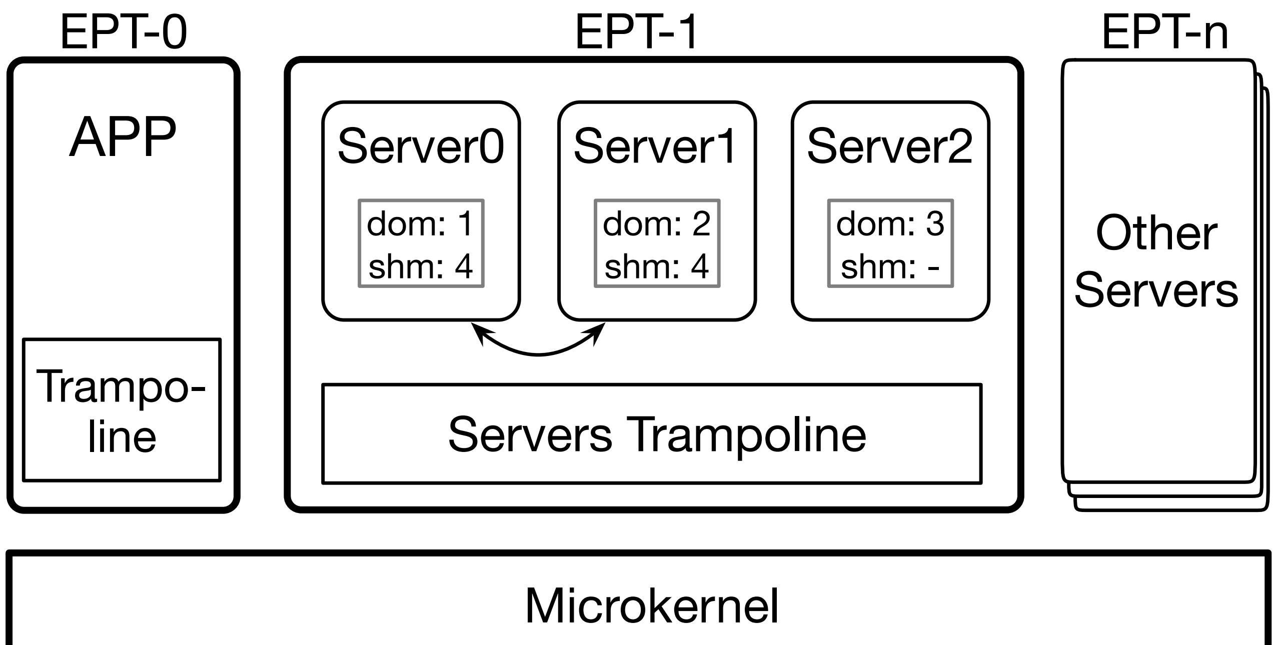
Case-Study: Boosting IPCs

- **Microkernel OS**
  - Implement OS functionalities in different user-space system servers
  - Isolation vs. Communication cost
  - UnderBridge: isolate servers with MPK-based domains *[ATC 20]*
    - <u>Limitation: small domain number due to the MPK limitation</u>



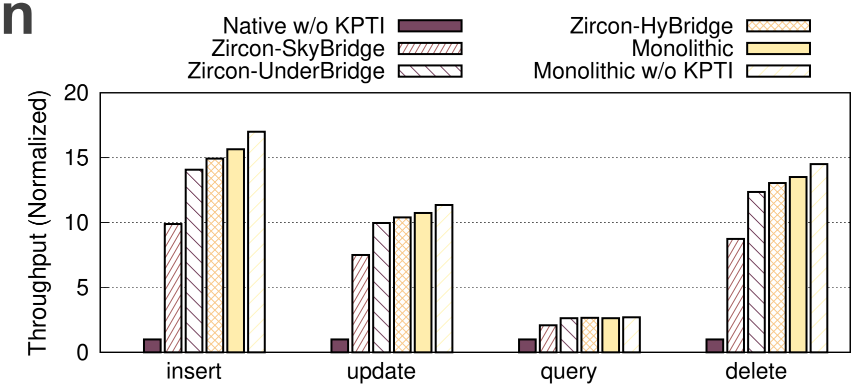
Case-Study: Boosting IPCs

- **HyBridge: A microkernel IPC design based on EPK**



Case-Study: Boosting IPCs

- **Application-to-Server IPC**
  - 527 cycles in HyBridge vs. 723 cycles in UnderBridge *[ATC 20]*
- **Cross-Server IPC**
  - 110 cycles in HyBridge vs. 437 cycles in SkyBridge *[EuroSys 19]*
- **Sqlite3 performance on Zircon**
  - 9x speedup than native
  - 66% speedup than SkyBridge



Conclusion

- **EPK**
  - First combines MPK and hardware virtualization features
  - Scalable and efficient memory domain mechanism
  - Three case studies to show the potential usages

Thanks!

Email: gujinyu@sjtu.edu.cn