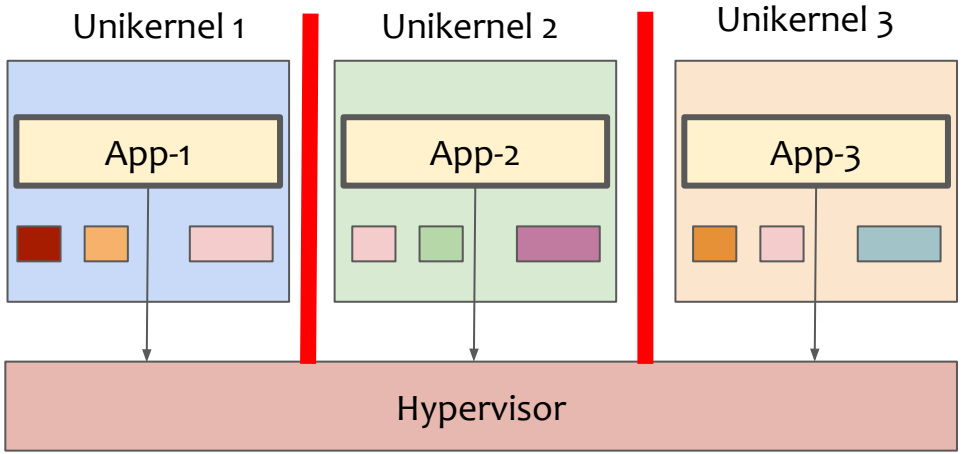
uIO: Lightweight and Extensible Unikernels

Masanori Misono, Peter Okelmann, Charalampos Mainas, Pramod Bhatotia

ACM SoCC 2024

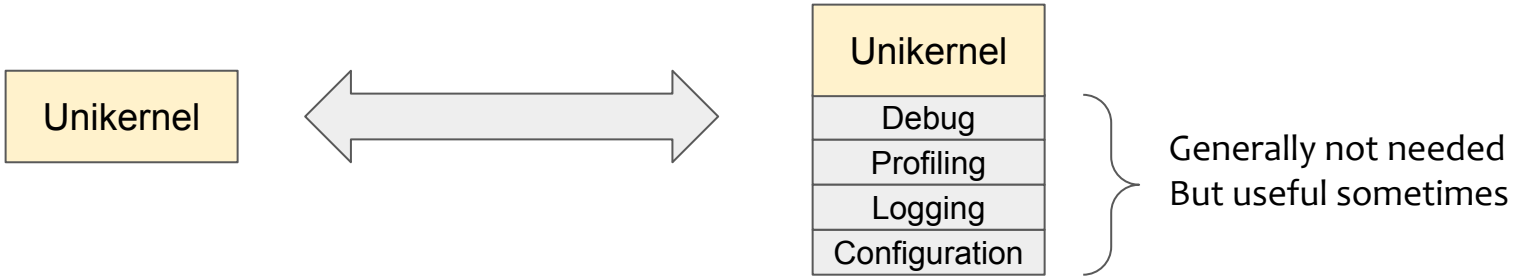
Unikernels

- *Specialized OS* for an application
  - 👍 **Better performance** by optimization
  - 👍 **Short boot time** thanks to small vm image size
  - 👍 **Strong isolation** by hypervisor



Problem of unikernels

- **Tension** between image size and available functionality/tools of unikernels
- **No common Interface** for the management (no ssh)



Extensibility is crucial for real-world adaptation

State-of-the-art

- Hypervisor-specific debugging tool
  - Uniprof (Xen), dumpcore (solo5), gdb stub provided by a hypervisor
  - ❌ Specific to the use-case
- Running unikernel as process on Linux
  - Unikraft linux mode, solo5-spt
  - ❌ Not usable when deploying
- Debugging and extending general VM/containers at runtime
  - VMSH (EuroSys'22), CNTR (ATC'18), HyperShell (ATC'14)
  - ❌ Targeting Linux environment

Existing works focus on specific use-cases or relies on Linux environment

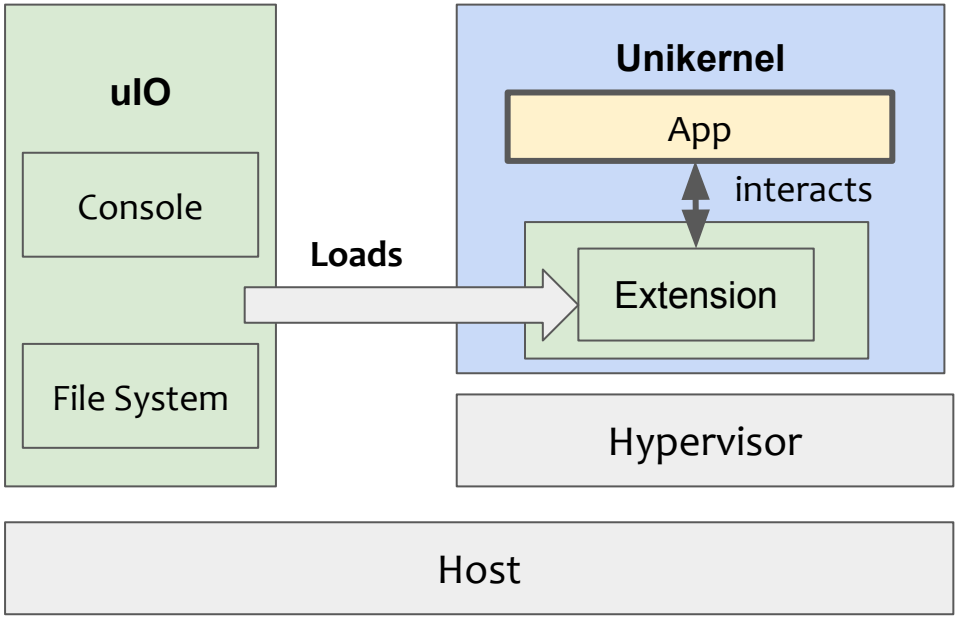
Problem statement

How do we achieve on-demand extensibility in unikernels?

- Design goals:
  - **Lightweightness:** Keep unikernels advantages
  - **Generality:** Generic interface for extensibility for unikernels
  - **Safety:** Prevent loaded program from accidentally compromising the app

uIO overview

- uIO provides *unikernels overlay* for on-demand extensibility



Outline

- ~~Overview~~
- Design
- Evaluation

Design challenges

#1 Generic overlay interface in unikernels

Virtio-based overlay interface

#2 Dynamic extension loading and execution

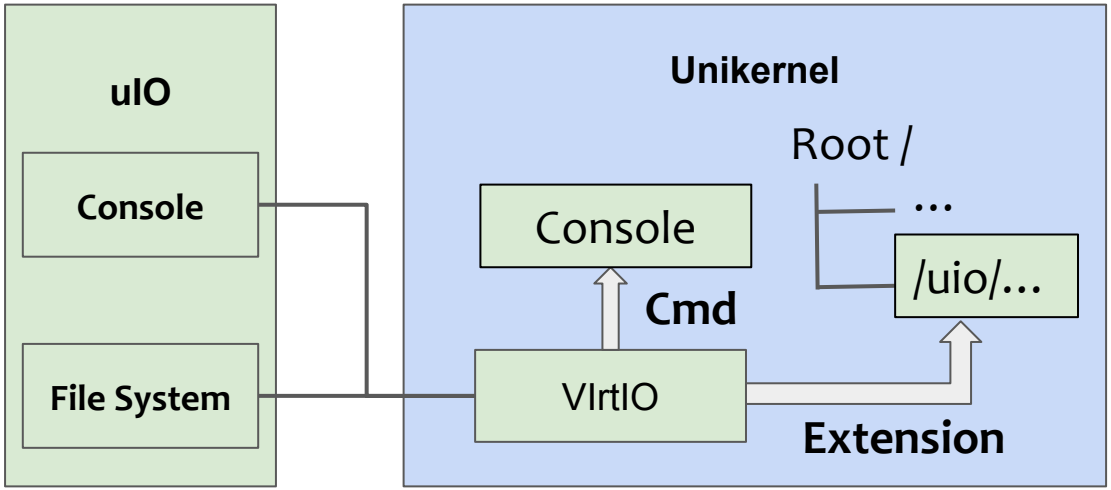
Load and link to the unikernel context

#3 Lightweight safety

HW-assisted and language-based isolation

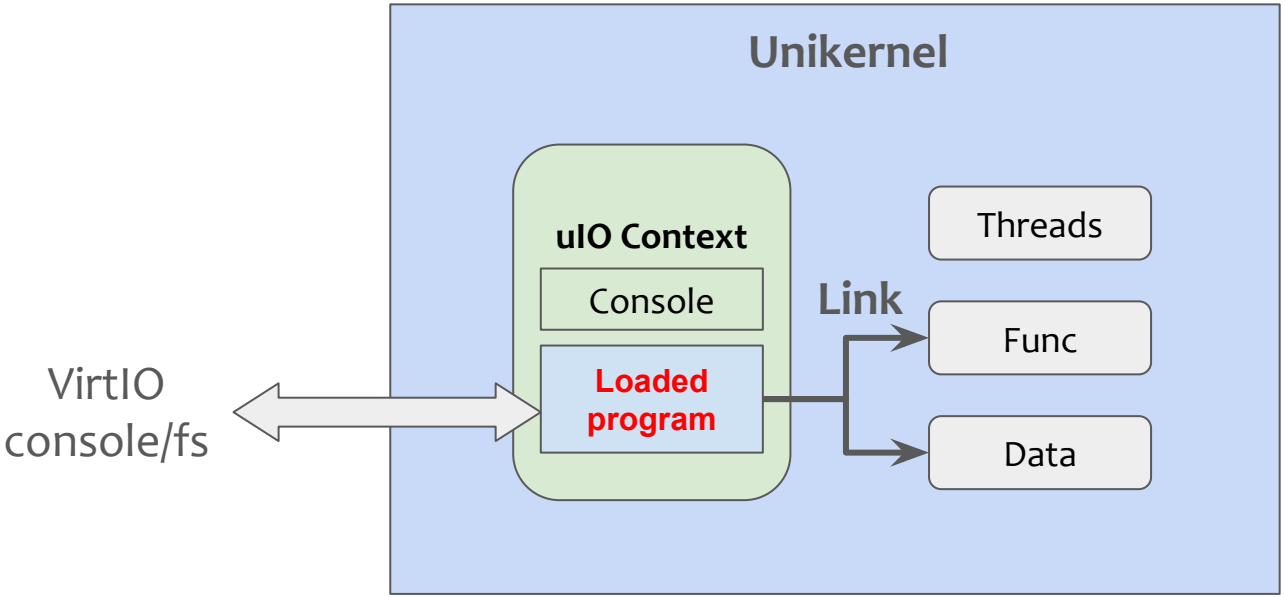
#1 Overlay interface

- Provide **virtio-based overlay** for console and file system

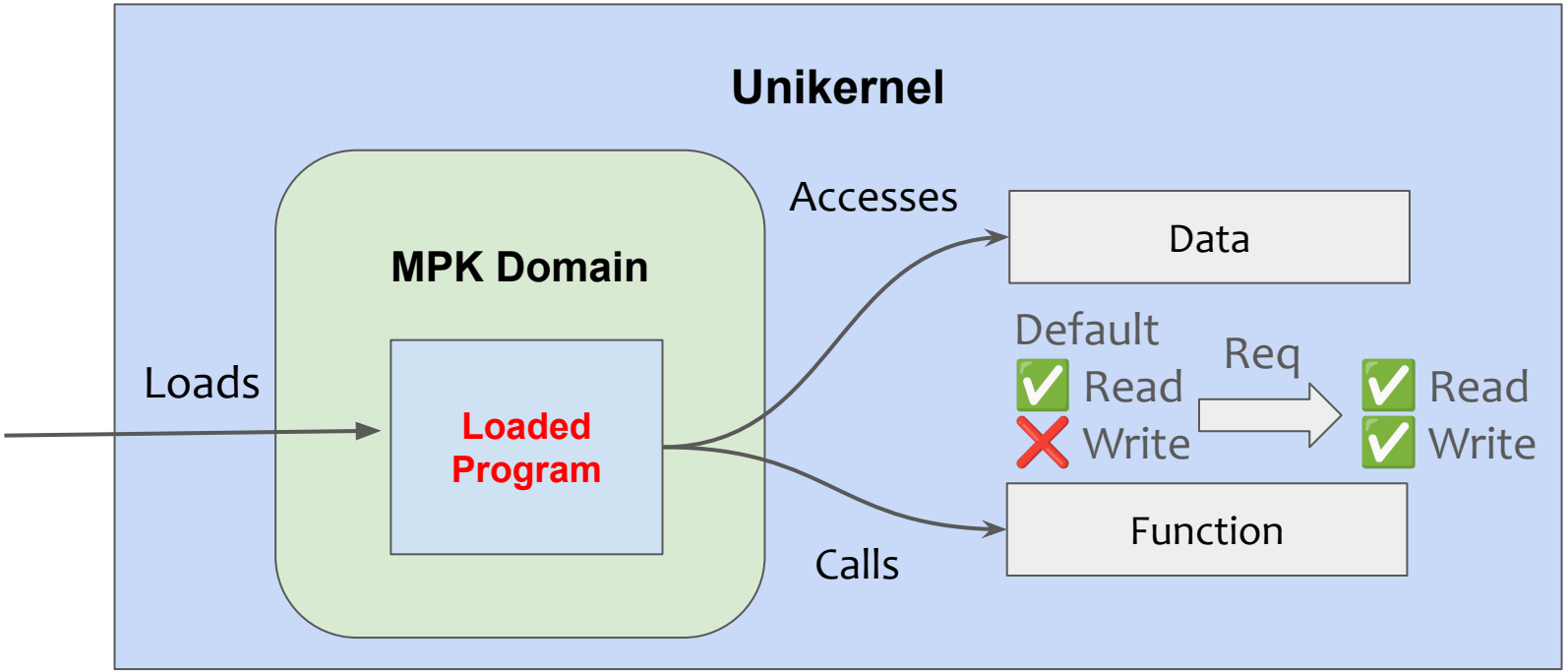


#2 uIO context for extension execution

- Schedulable entity, handling user request from the outside
- **Directly link** loaded program with the unikernel

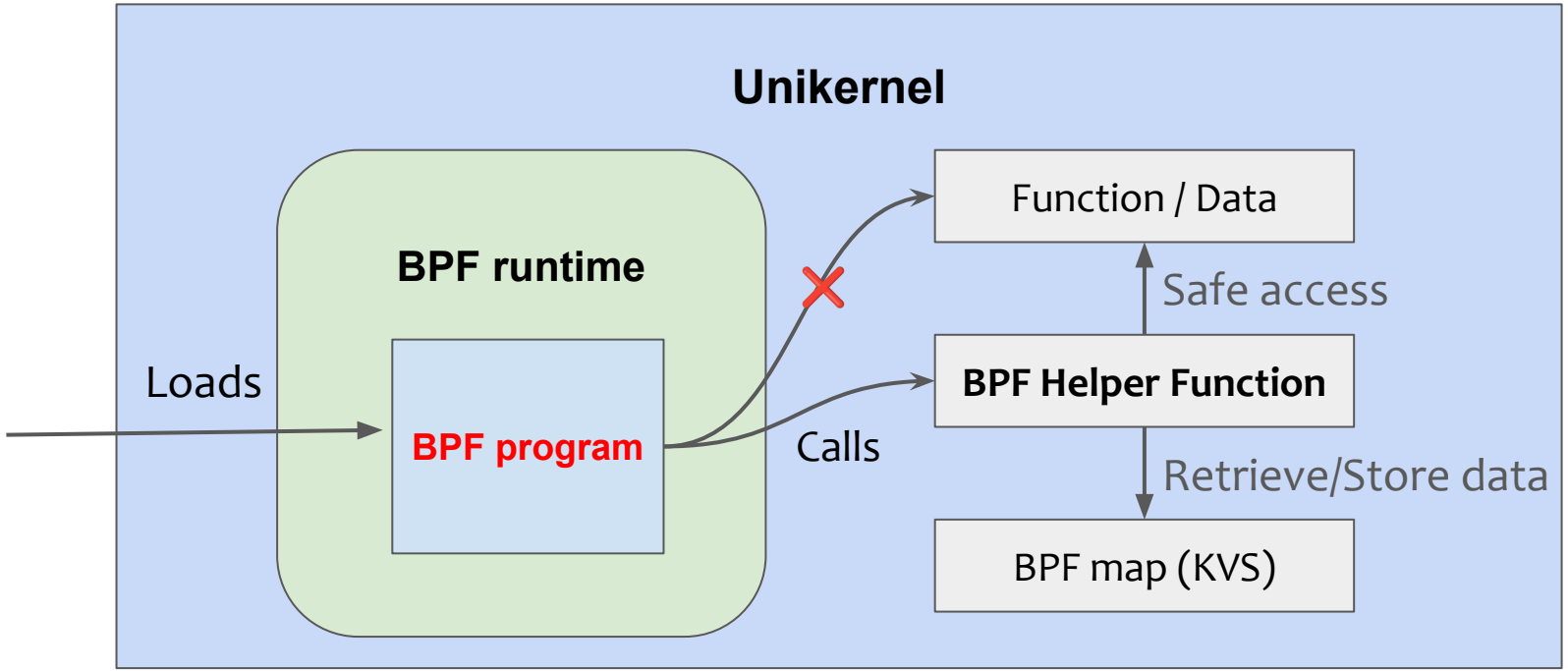


#3 Safe execution environment (1) HW-based (MPK)



Application explicitly request write access to the main unikernel memory

#3 Safe execution environment (2) language-based (BPF)



Application developers expose helper functions for their needs

Outline

- ~~Overview~~
- ~~Design~~
- Evaluation

Implementation

- Prototype on **Unikraft** unikernel
  - Virtio-console for console
  - Virtio-9p for filesystem
  - Integrate uBPF runtime
    - Use interpreter mode, dynamic safety checking
- Enabling **real-world use cases**
  - 🐛 Interactive debug environment
  - 📝 Nginx re-configuration
  - 📈 Performance monitoring with performance counters
  - 👓 BPF-based introspection and function tracing

Evaluation

Question: Does uIO preserve unikernels benefits?

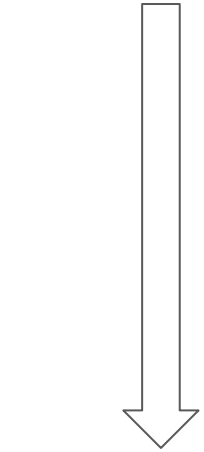
- **Image size overhead**
- **Application performance**
- Robustness
- Console responsiveness
- Program loading time
- File system performance

Refer to the paper

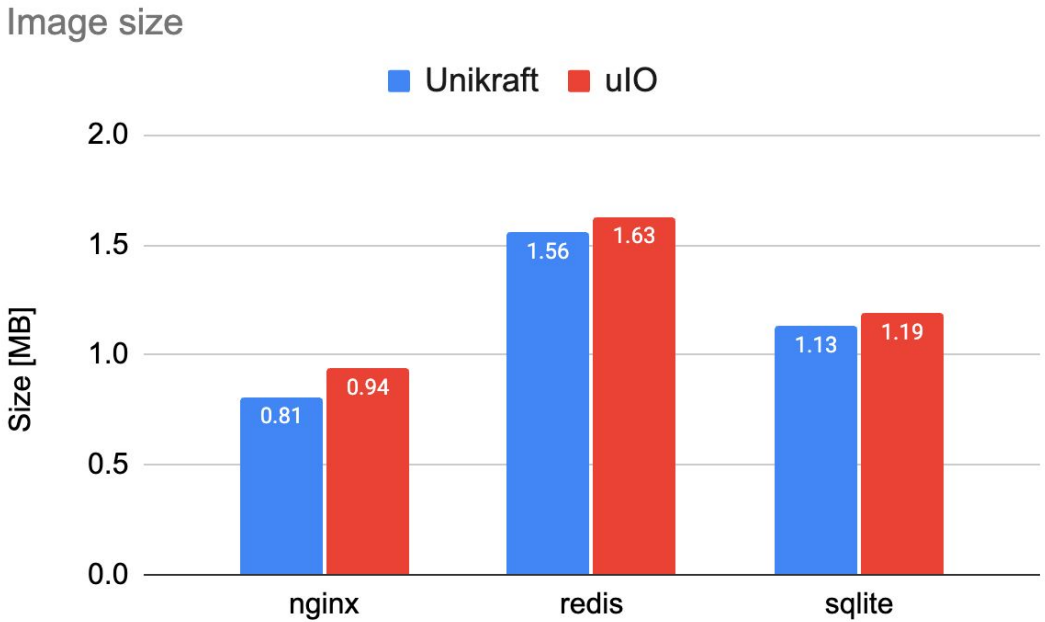
Experimental setup

- Intel Xeon Gold 5317 CPU, 256GB memory
- VM: 1 vCPU and 1GB of memory

Image size overhead



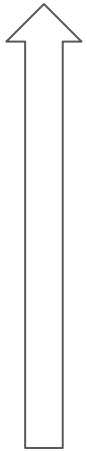
Lower is better

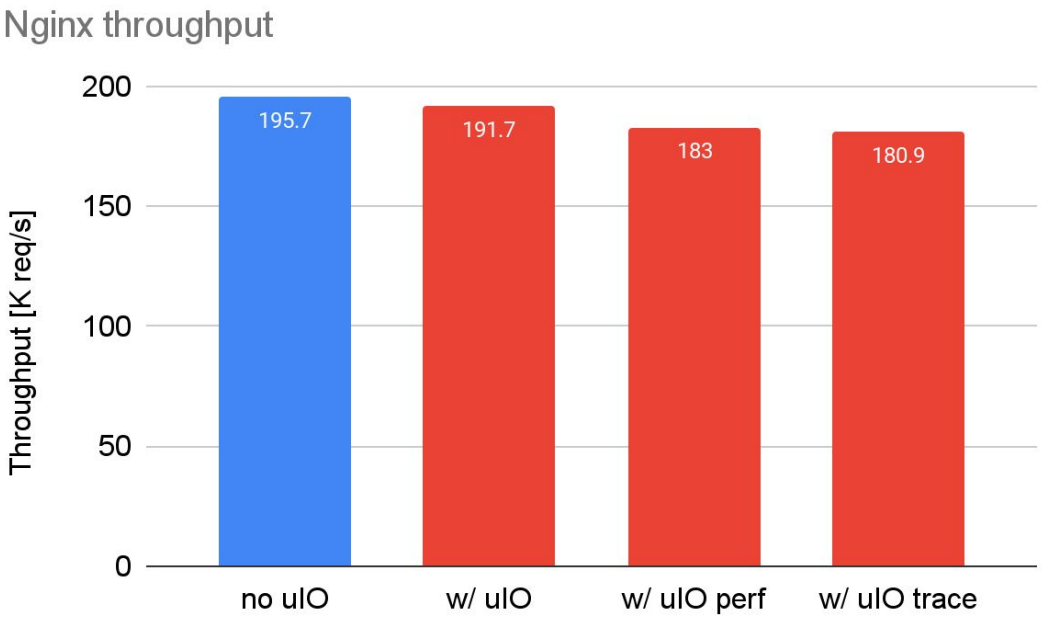


uIO increases image size only by several kilo bytes

Application performance (nginx)

Higher is better





uIO achieves extensibility with minimal overhead

Summary

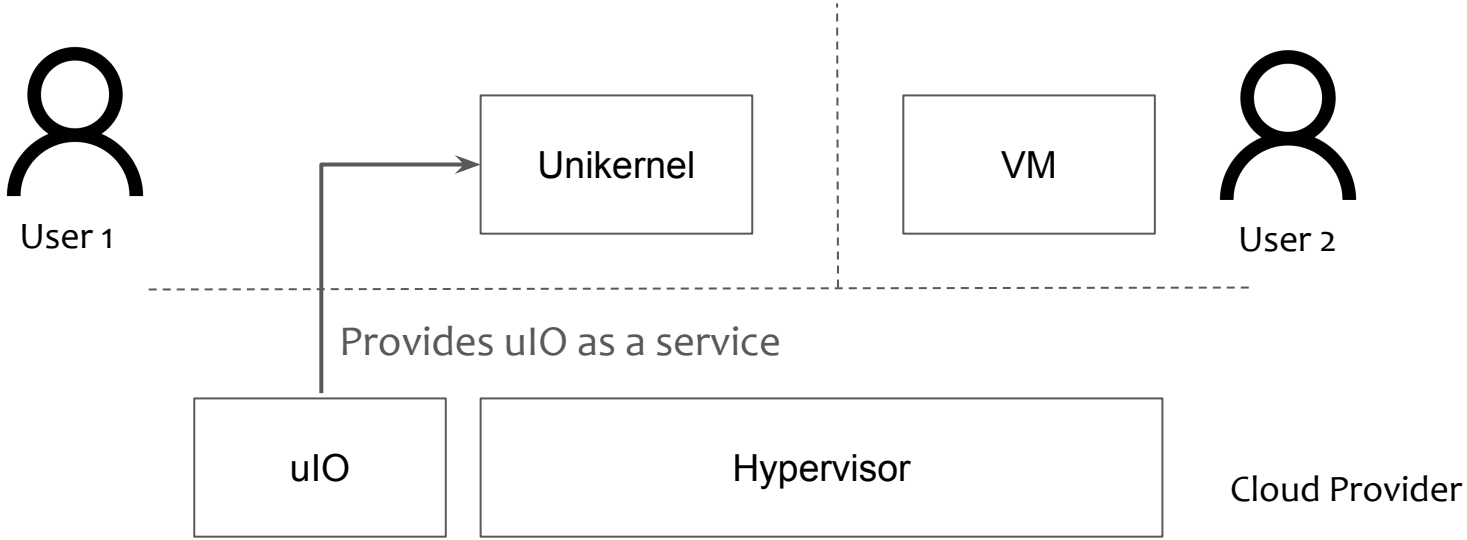
- **uIO** provides unikernels overlay to realize on-demand extensibility
- Two safe execution environment with tradeoff between performance and safety
  - Hardware-assisted memory isolation (**MPK**)
  - Language-based isolation (**BPF**)
- Prototype on Unikraft unikernel and present several use-cases

✉ Masanori Misono <masanori.misono@in.tum.de>

Source: github.com/TUM-DSE/uio

Backups

Deployment model



uIO leaves the responsibility of user authentication to the cloud provider

eBPF (extended Berkeley Packet Filter)

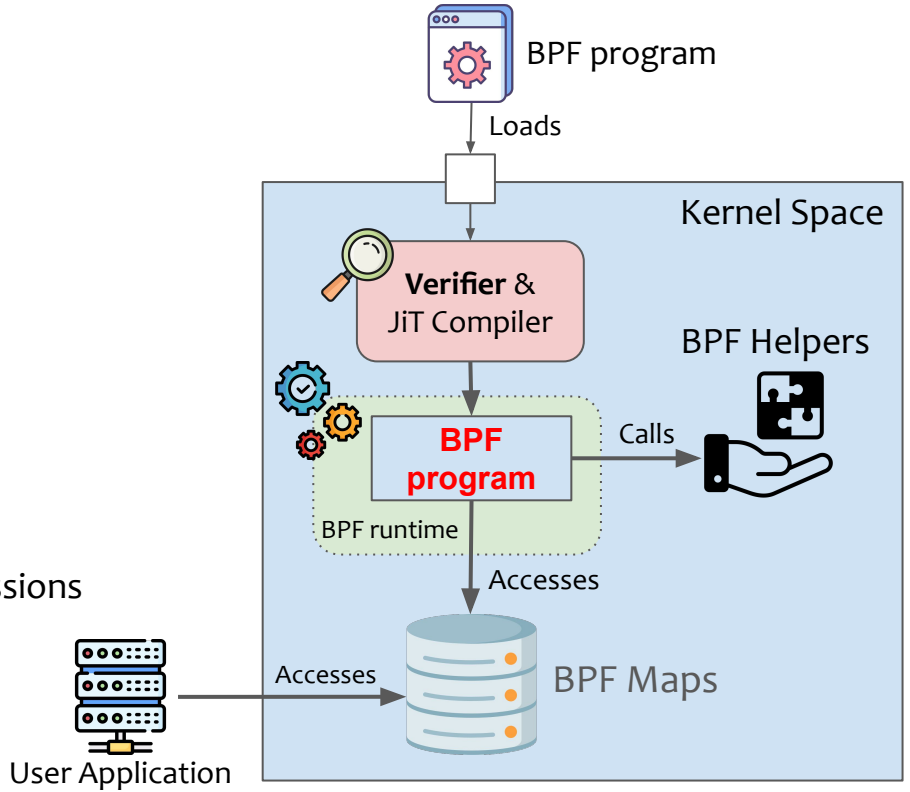
Lightweight in-kernel language VM

Sandbox property can be ensured by:

- Using **interpreters** (weaker)
- Using **verifiers** to verify in advance (stronger)
  - Detects potential sandbox escalation
  - Forbid undefined behaviors

Useful features:

- Maps (kv-store)
- Helper functions
- Program Types: Runtime context & helper permissions



Evaluation - Safety

Evaluation Program	MPK	BPF (interpreter)	BPF (verifier)
OOB*	SEGV	Terminated	Rejected
OOB* with Nullptr	SEGV	Terminated	Rejected
Infinity Loop	System freezes	Terminated	Rejected
Division by Zero	Error Ignored	Handled^	Rejected (explicit) Handled (runtime)
Instruction Type Safety	Error Ignored	Error Ignored	Rejected
Program Type Safety	Error Ignored	Error Ignored	Rejected
Helper Function Type Safety	Error Ignored	Error Ignored	Rejected

: Memory Safety

: Termination

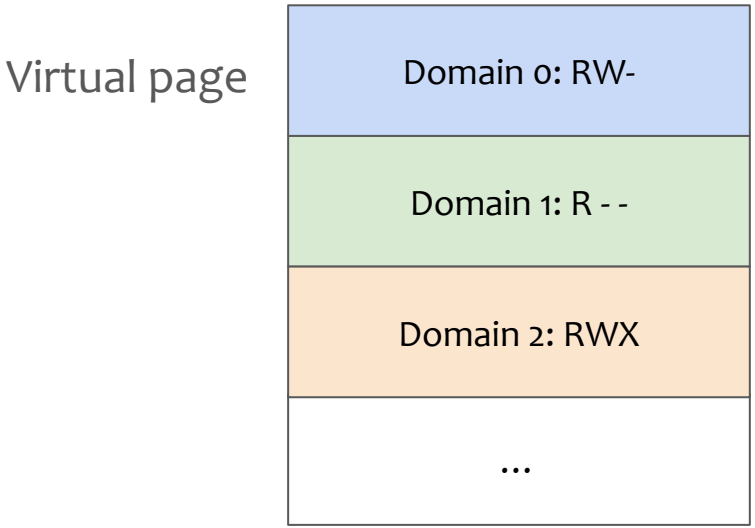
: Runtime Errors

: Type Safety

*) OOB: Out of bound memory access, ^) the interpreter returns 0

MPK

- Domain-based memory isolation



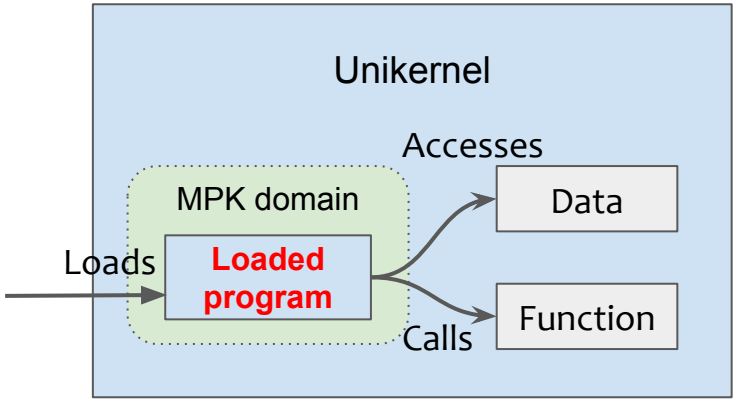
- Use upper bits of page table entry to specify domain
- Update permission using *wrpkru* instruction

MPK in the kernel mode

- MPK enforces permission checks on **any user-accessible page** (=U/S bit = 1)
- We modify Unikraft memory management so that it allocates a page as user page to use MPK
- Note
  - This imply that SMAP and SMEP needs to be disabled
    - Otherwise cannot access user pages in ring-0
    - This does not raise any security concern for unikernels
  - The latest Intel processors support PKS (Supervisor Protection Keys)
    - This provides MPK functionality for kernel pages as well

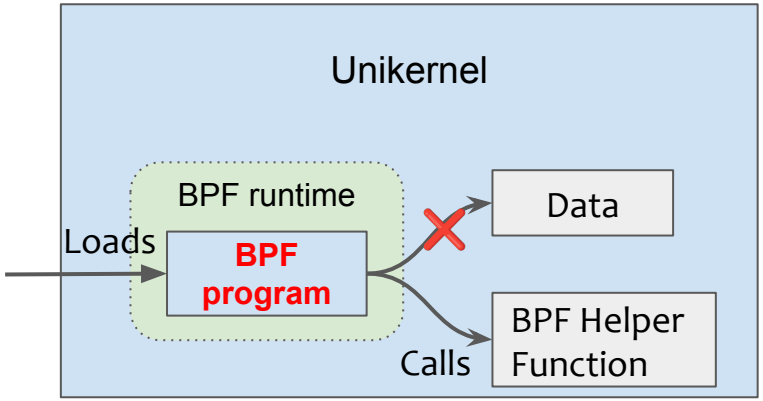
Safe execution environment

Hardware-based isolation (**MPK**)



✅ Maximum flexibility (access arbitrary data)
❌ Limited safety guarantee

Language-based isolation (**eBPF**)



✅ Storonger safety guarantee
❌ Limited functionality (compensated by helpers)

Users can choose the execution environment depending on the needs