PFF-UINTR

Page Fault Forwarding via User-Interrupts

Anton Ge
Advisors: Dr. David Schall, Ilya Meignan--Mason
Systems Research Group
https://dse.in.tum.de/

28.04.2025 – 28.08.2025

Caching for Data-Intensive Applications

Caching: keep some data in memory to avoid repeating IO operations

Key challenge: Who manages the cache ?

OS-level (mmap):

- Simple and general-purpose
- System loads data on access
- Lacks flexibility

Relies on the HW (with page faults)

User-space caching :

- Faster and more flexible
- Complex to engineer
- Specific to system/app

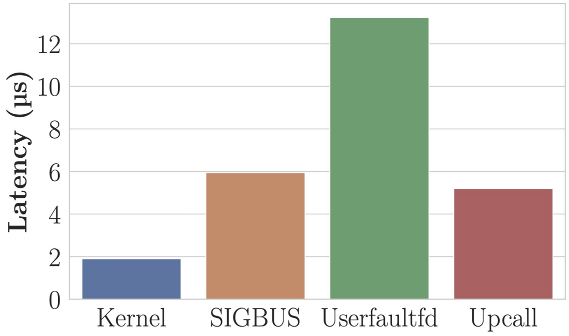
Avoids page faults with pin()/unpin()

User-Space Page Fault Handling

Third approach: Forward faults to user-space (signals[1], userfaultfd[2], upcalls[3])

- mmap's simplicity with app-level control

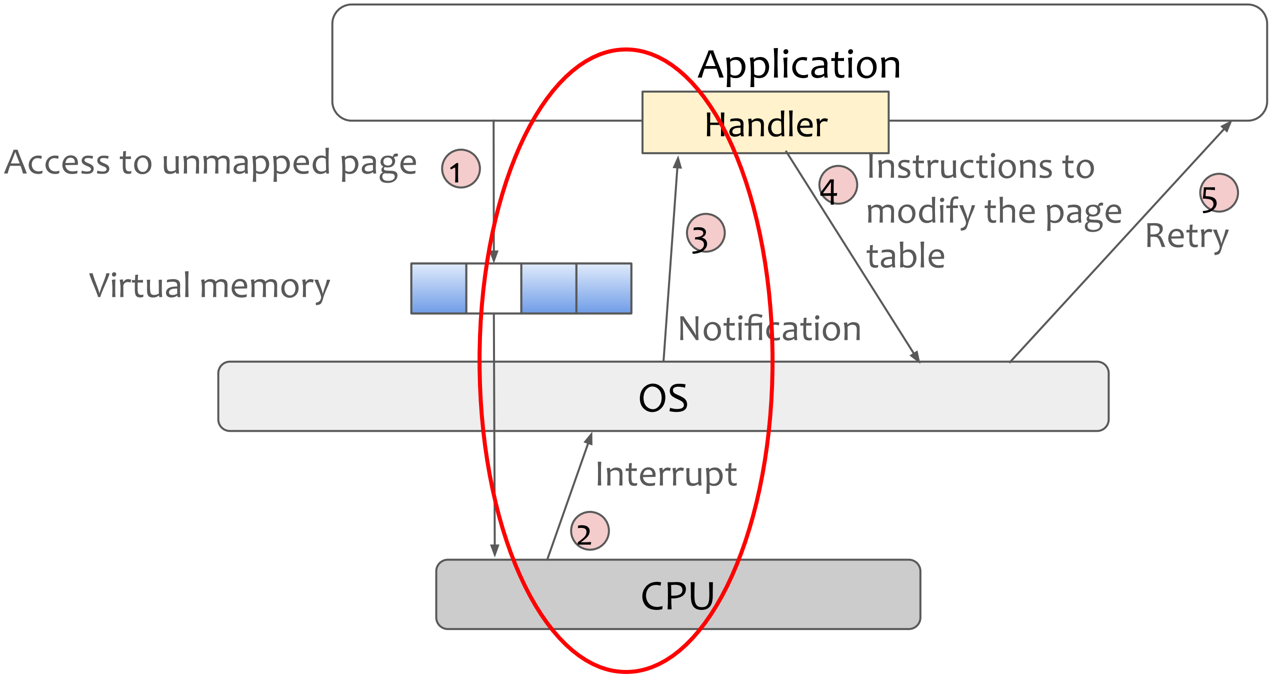
But: delays and context-switch overhead from communications



Source: Jalalian et al., 2024

[1] uMMAP-IO: User-Level Memory-Mapped I/O for HPC [HiPC'19]
[2] Enabling Scalable and Extensible Memory-Mapped Datastores in Userspace [TPDS'22]
[3] ExtMem: Enabling Application-Aware Virtual Memory Management for Data-Intensive Applications [ATC'24]

Workflow of user-space page fault handling



Software-based solution is slow because the OS interposes on the path

Problem statement

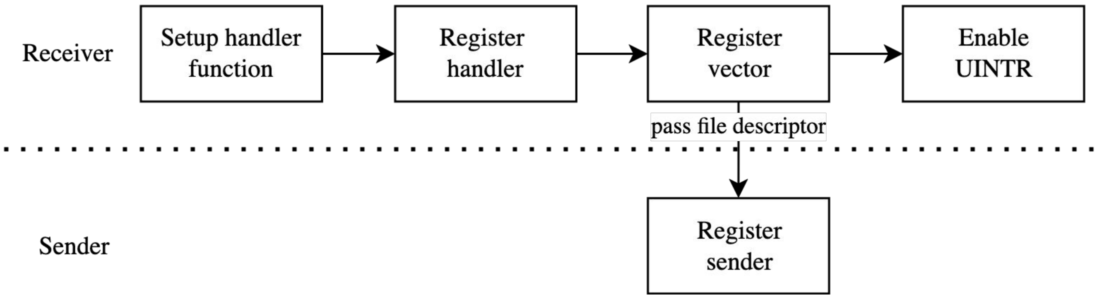
Can we efficiently send page fault information to user-space?

Overview

Key Idea: Design a hardware mechanism to deliver the page fault to user-space

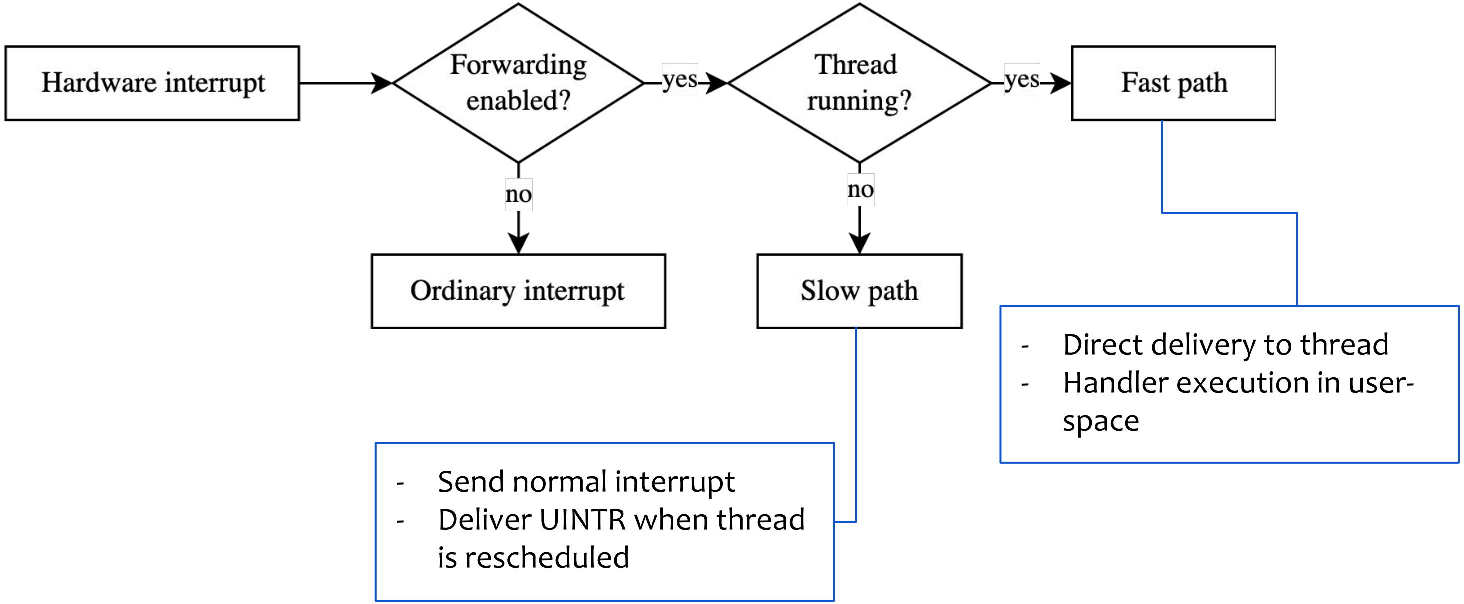
Leverage User Interrupts

- New hardware feature on Intel CPUs (from Sapphire Rapids)
- Interrupt delivered to processes in user-space without kernel involvement
- Hardware implementation: limited to Inter-Processor Interrupts (IPIs)



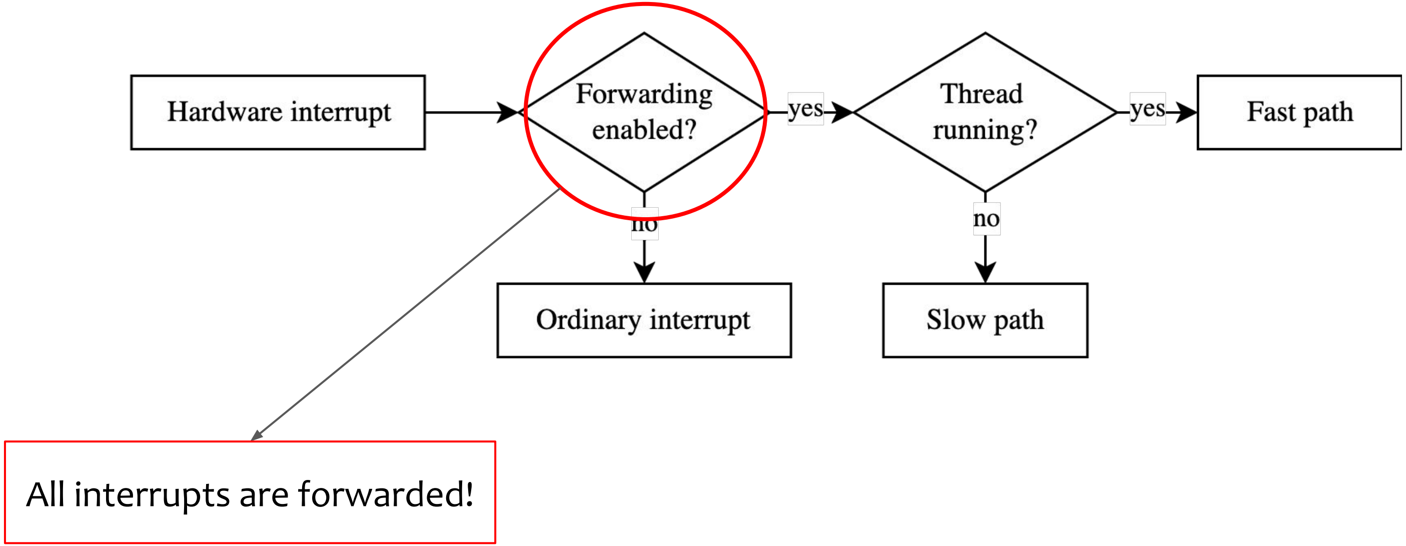
General-purpose User Interrupts

- xUI [ASPLOS'25]: extends the existing UINTR delivery architecture



Challenge

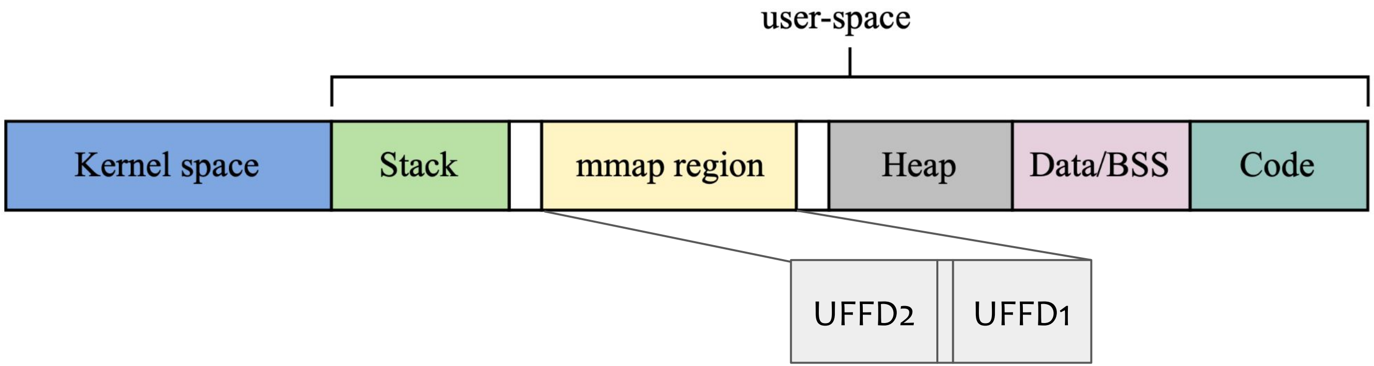
- xUI forwards all interrupts to user-space
- Application only wants to control the memory region of the cache



How can we selectively forward page faults ?

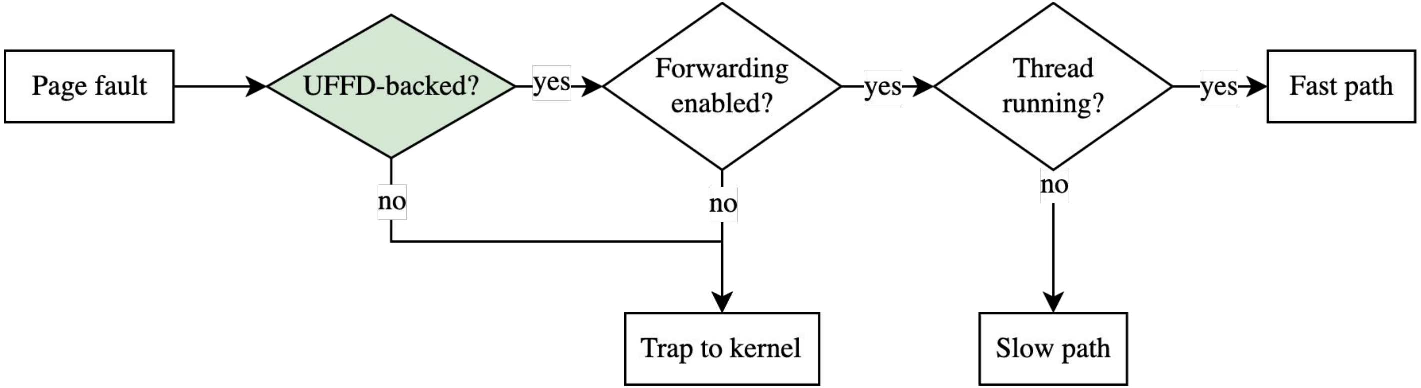
Userfaultfd (UFFD)

- Creates special region in the memory where page faults are handled in user-space

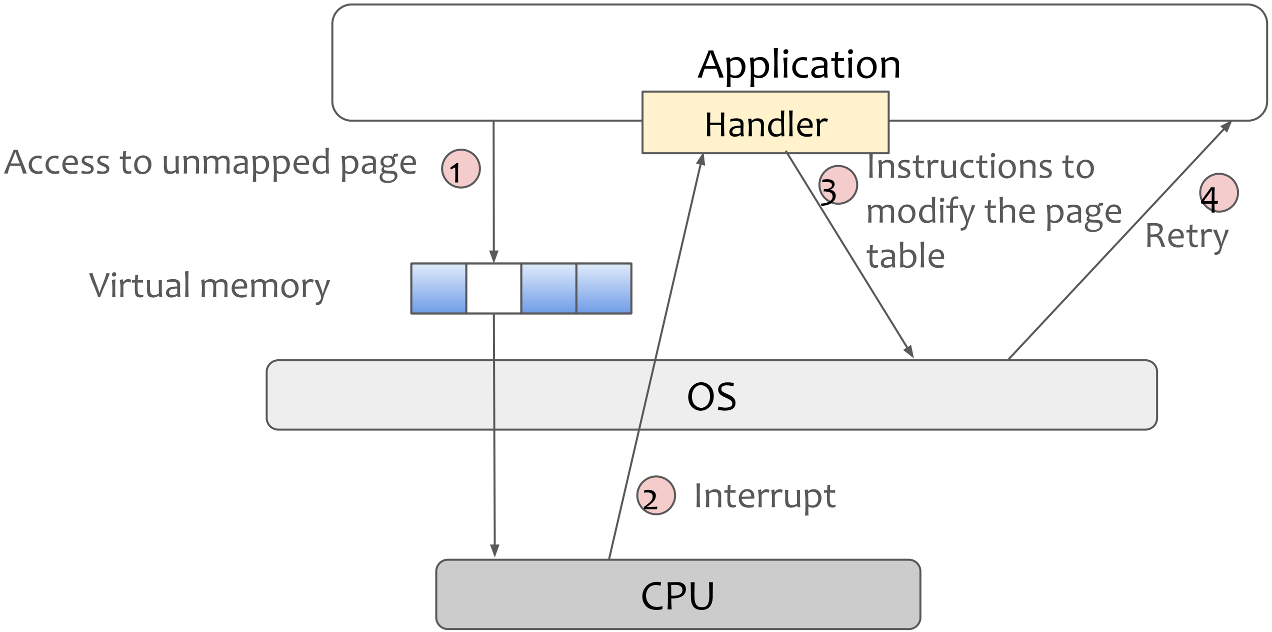


Design Overview

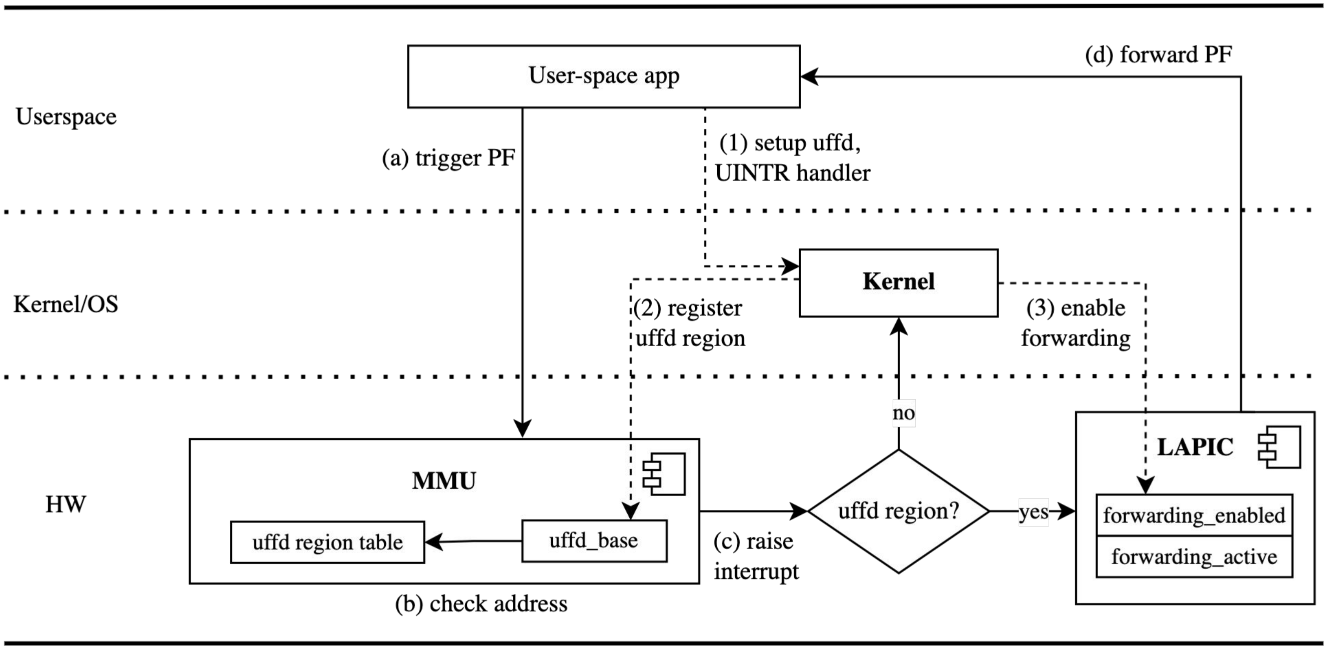
Make the MMU aware of UFFD regions



Workflow

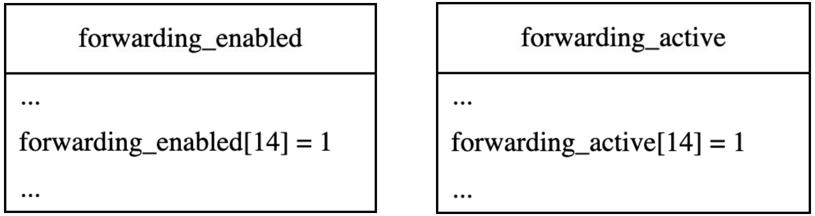

Application
Handler
Access to unmapped page
1
Virtual memory
3
Instructions to modify the page table
4
Retry
OS
2
Interrupt
CPU

Hardware Workflow

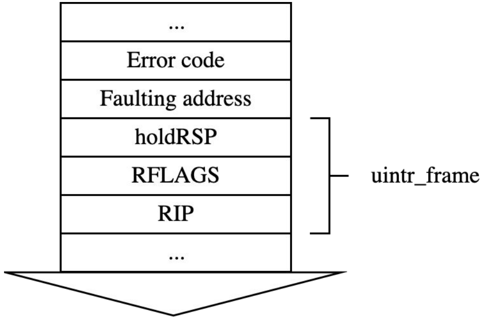


Design Details

- Key components:
  - MMU: stores and checks uffd-backed address ranges
  - Extended local APIC: registers for interrupt forwarding



- Faulting address and error code:
  - passed through stack/extended uintr_frame struct



Implementation

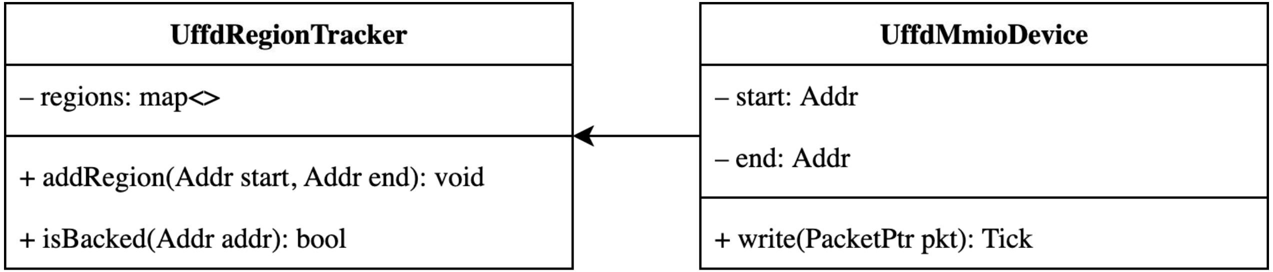
- gem5 X86-system based on xUI [ASPLOS '25]
  - Interrupt forwarding by injecting into the X86ISA interrupt logic
  - Custom fault classes and microcode routines

⇒ Simulate behavior of local APIC

Implementation

- gem5 X86-system based on xUI [ASPLOS ‘25]

- Address region tracking



Implementation

- gem5 X86-system based on xUI [ASPLOS '25]

- Address region tracking

- Linux device and driver
  - Map address for MMIO → enable access to UffdMmioDevice in gem5
  - Expose pointer to MMIO region
  - Pass uffd-region to gem5 device during userfaultfd setup

Evaluation

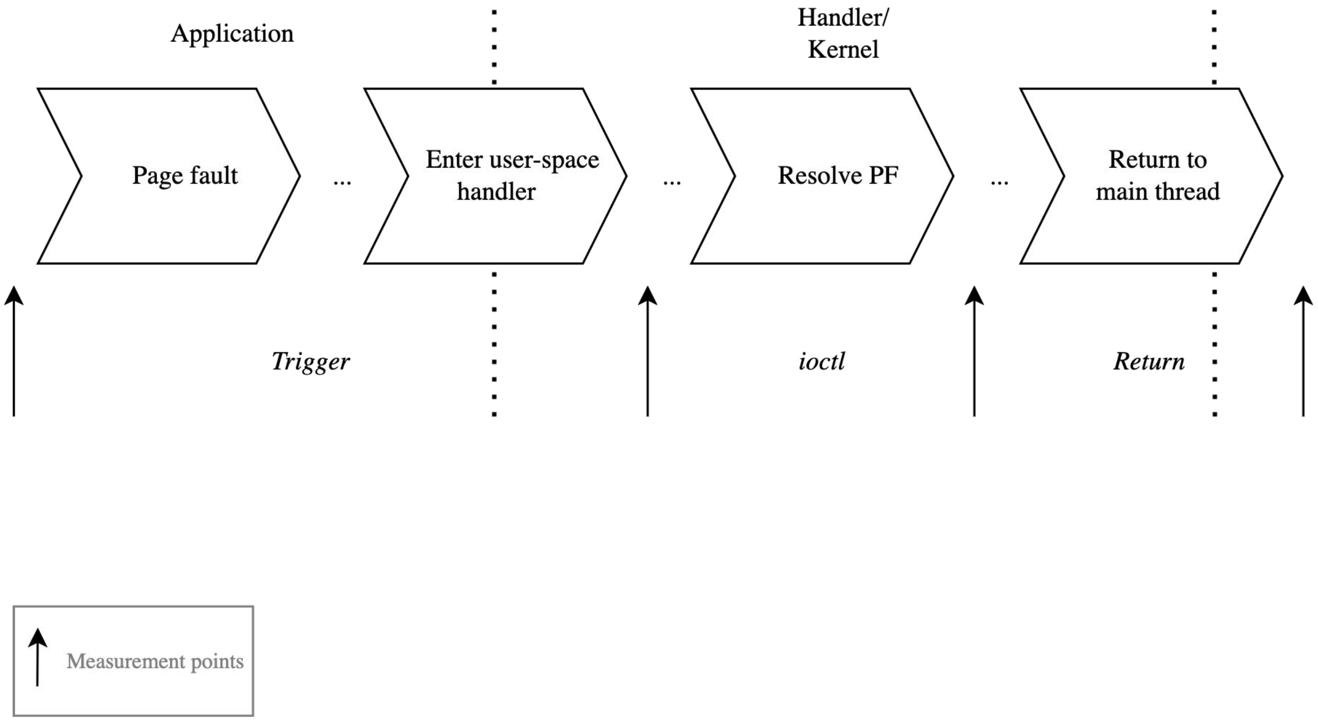
- Can UINTR-based page fault forwarding reduce the latency of user-space fault handling?
    - Synthetic page fault benchmark

- How does page fault forwarding impact the performance of more realistic workloads?
    - Graph traversal with breadth-first search

Evaluation

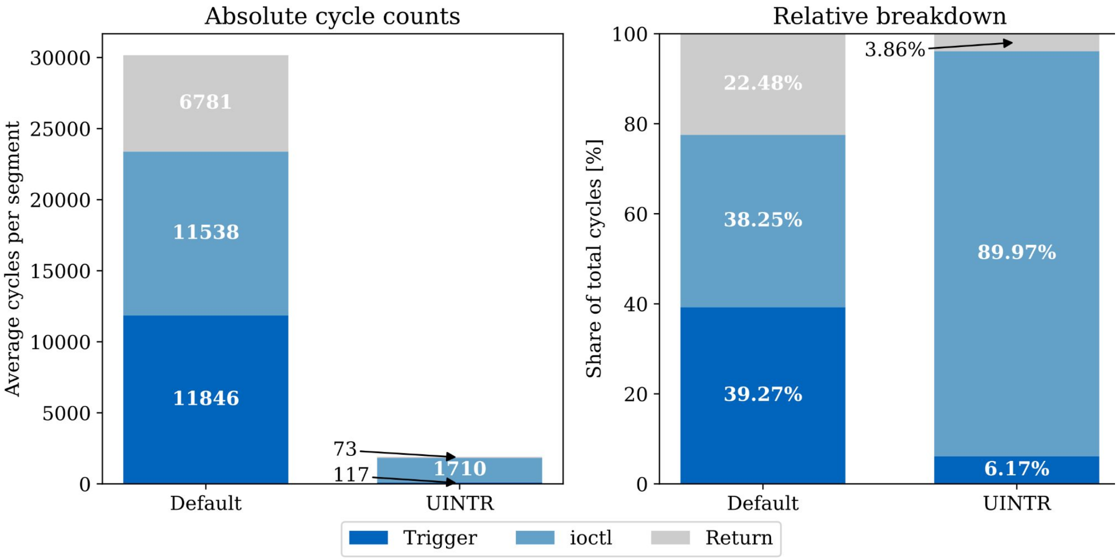
- Experimental setup: gem5 system
  - Simulation mode: fullsystem
  - CPU: out-of-order (O3) CPU
  - Memory: 16 GB DDR3
  - Linux kernel: 6.0.0 with Intel UINTR patches and UFFD modifications

Measurement Locations



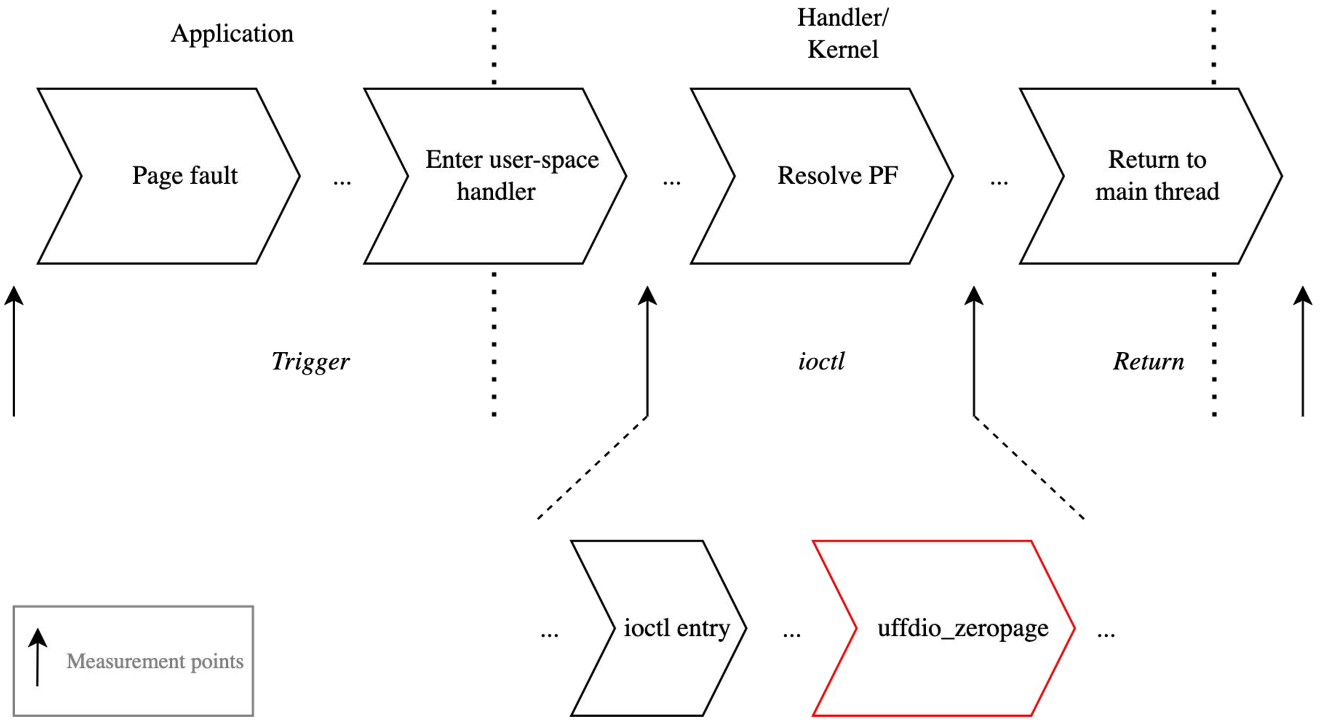
Evaluation

- Page fault benchmark: default Userfaultfd vs Page Fault Forwarding



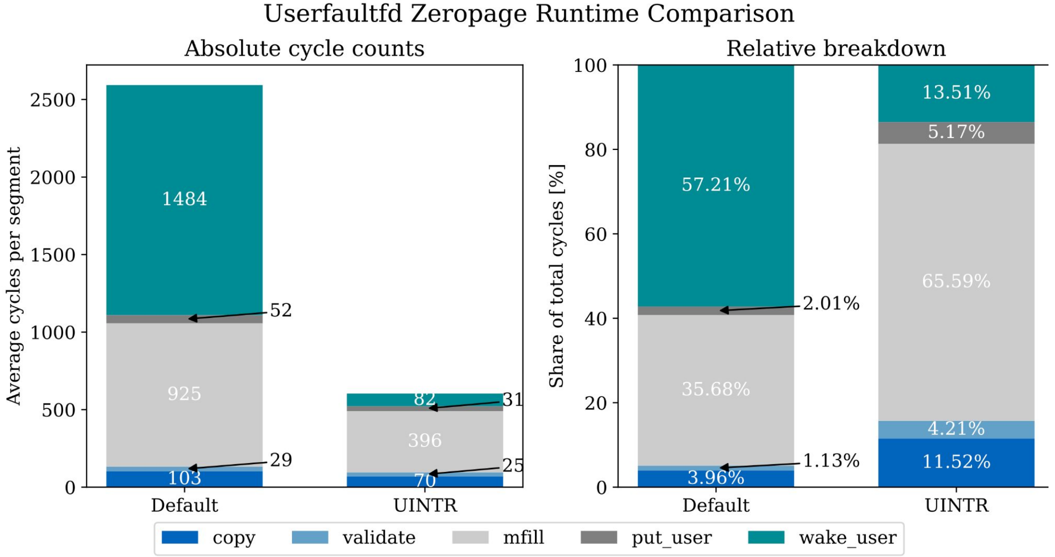
Page Fault Forwarding reduces latency from fault notification and resolution, and eliminates the need for a polling thread.

Measurement Locations – Within IOCTL



Evaluation

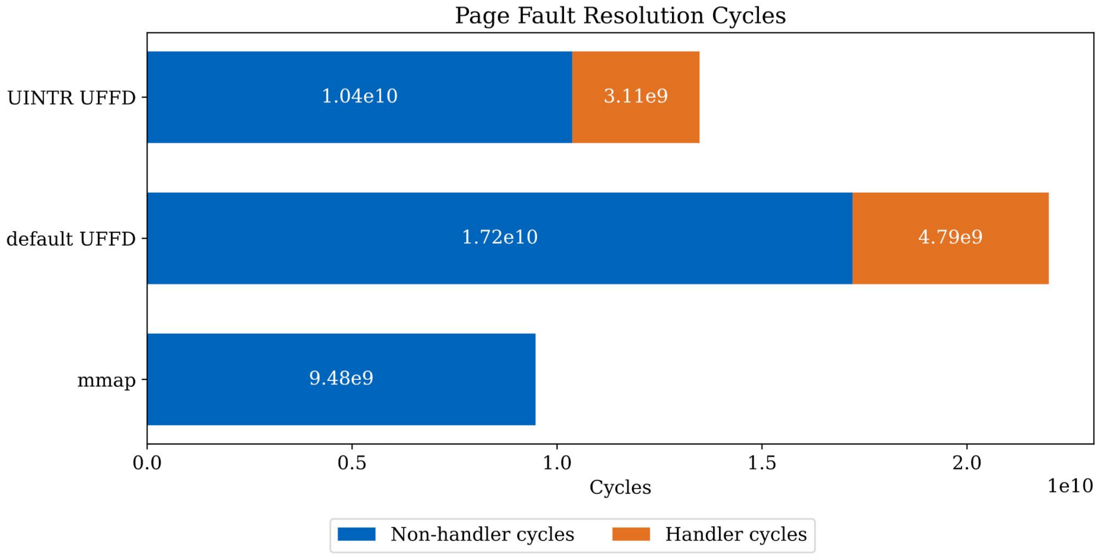
- Page Fault Forwarding: detailed breakdown of the ioctl (userfaultfd_zeropage)



Page Fault Forwarding avoids waitqueue management and locking overhead.

Evaluation

- BFS on CSR graph with 4.19×10^{6} vertices, 1.34×10^{8} edges (31 MB)



Lower page fault handling latency translates into better end-to-end performance

Summary

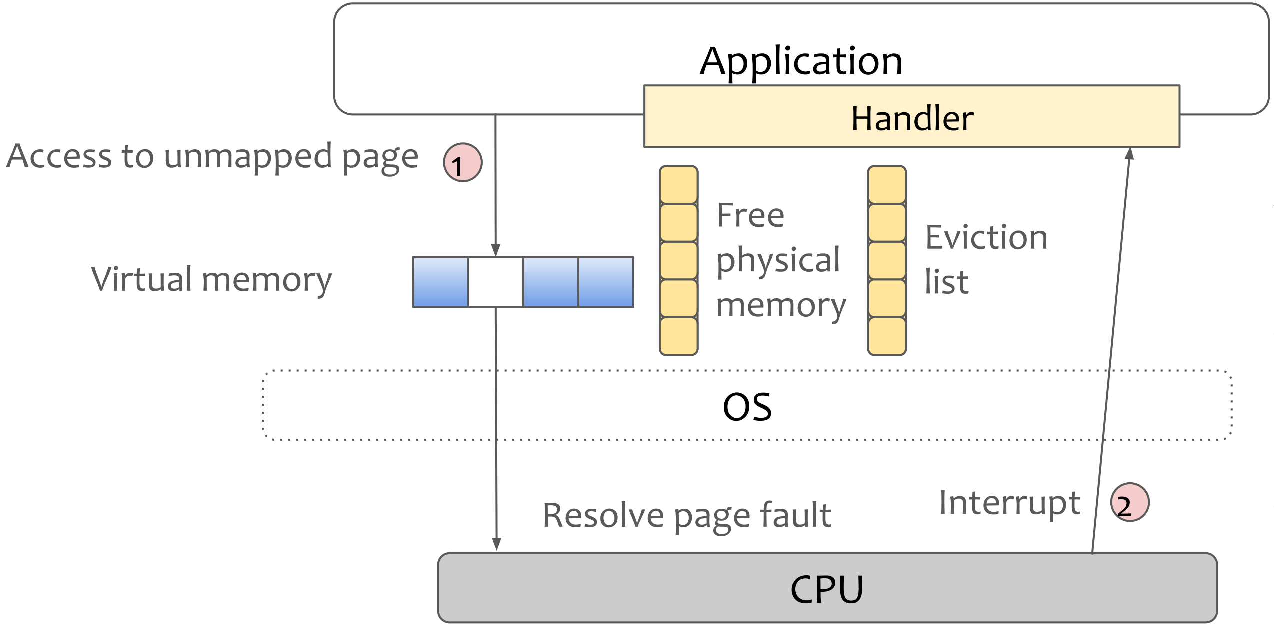
Data-intensive apps: high page fault latency in user-space handling
- Software-only: kernel involvement
- Hardware forwarding: no page fault support

Forwarding page faults at UFFD-backed addresses via UINTR
- xUI with page fault metadata and forwarding logic
- Lower notification and fault resolution latency → measurable speedups in realistic workloads
- No separate handler thread

Repository link: https://github.com/TUM-DSE/uintr

Follow-up?

- Current design still requires the OS to modify the page table
- Can we completely bypass the kernel for memory management?
- Key Idea: Resolve the page fault in hardware using application directions



On return:

1. Anonymous page: update lists and retry access
2. File-backed page: read from storage and update lists