

Custom Page Fault Handling with eBPF

Tal Zussman*, Teng Jiang*, Asaf Cidon
Columbia University

Overview

- Why custom page fault handling?
- Existing solution
- eBPF solution
- Applications

Why Custom Page Fault Handling?

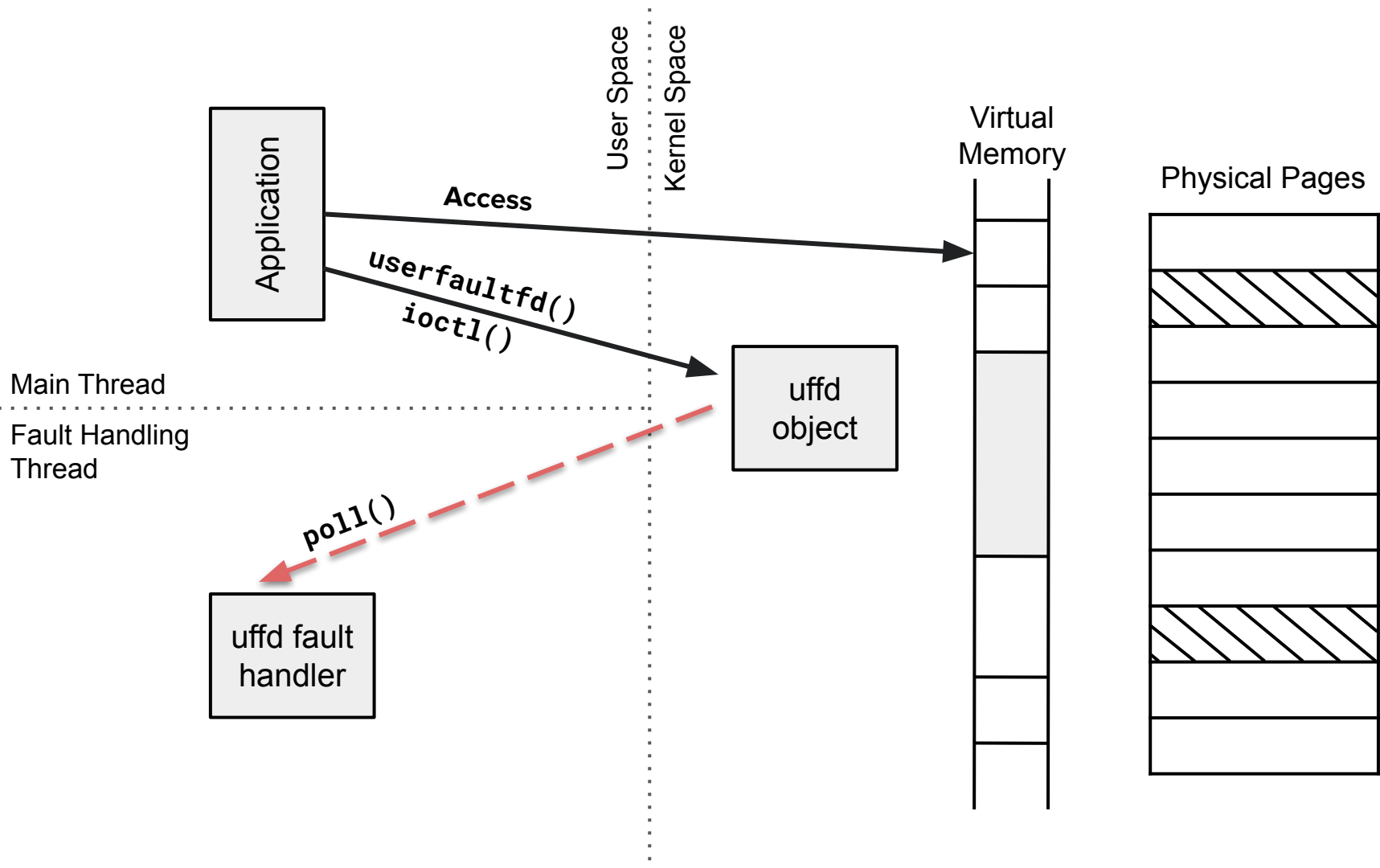
- Page access notifications
- Set faulting page contents on page-in
- **Greater application control of memory management**

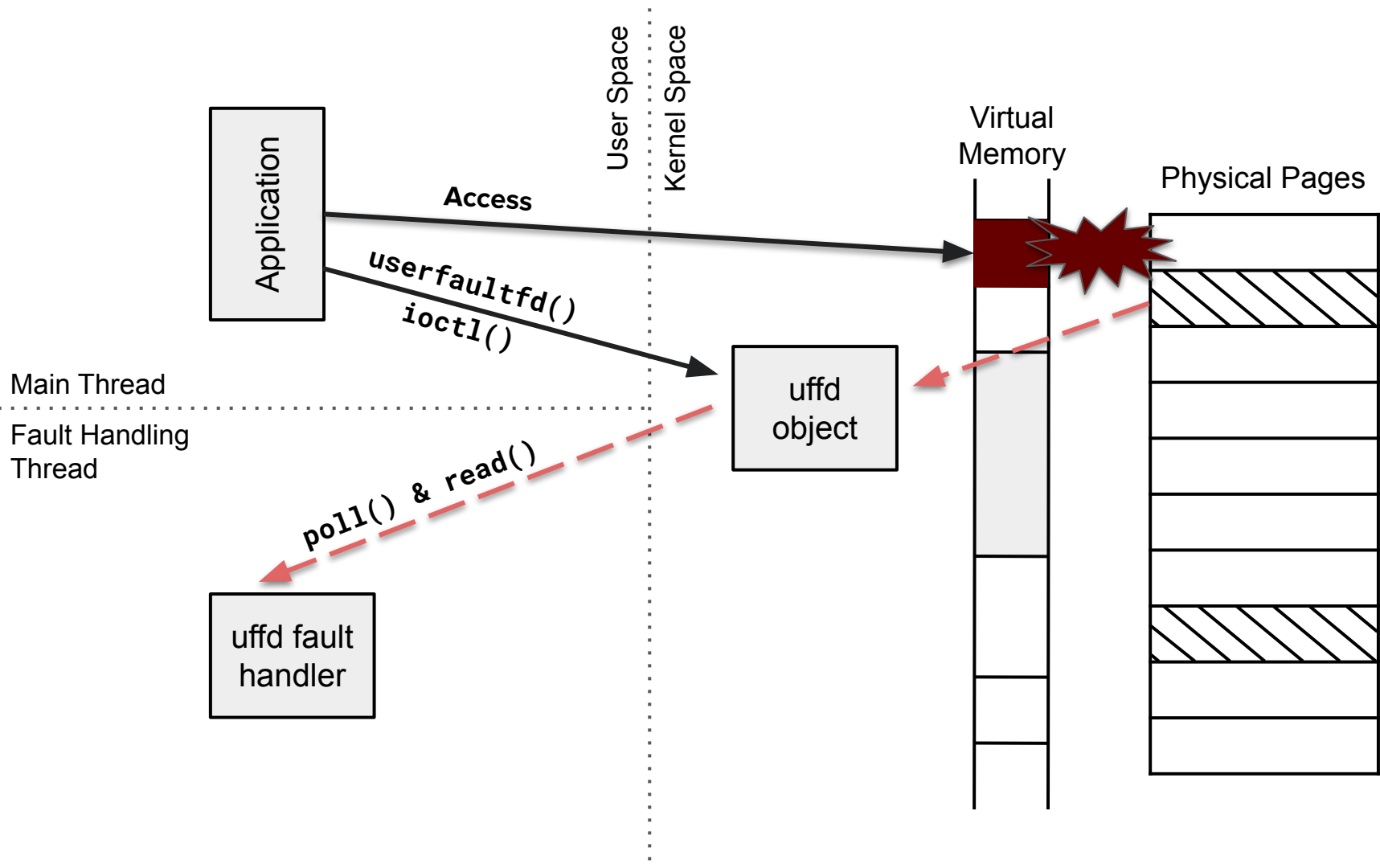
Why Custom Page Fault Handling: VM Migration

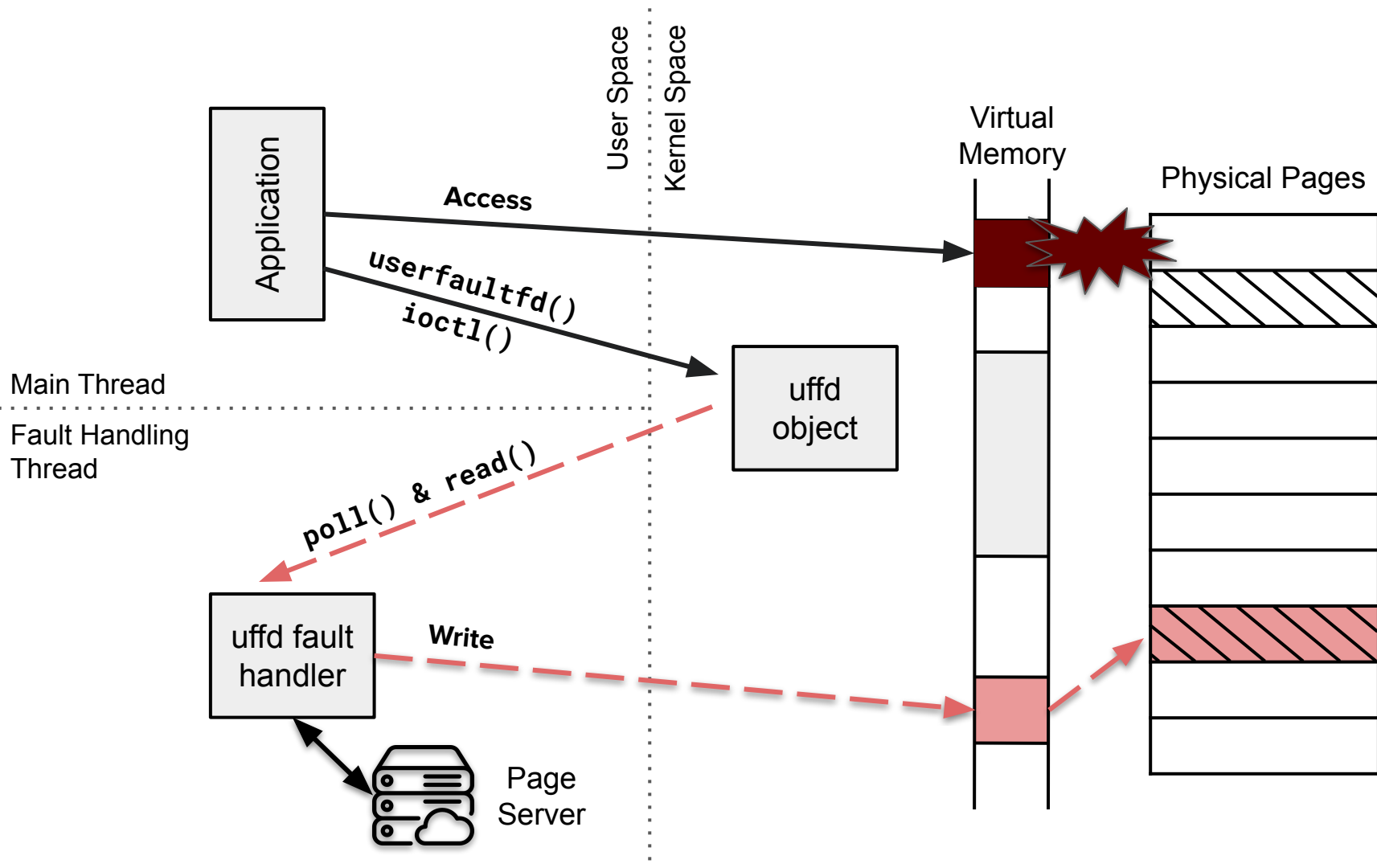
- Move VM across servers – **minimize downtime**
- Copy minimal required memory first
- Lazily copy remaining pages on access

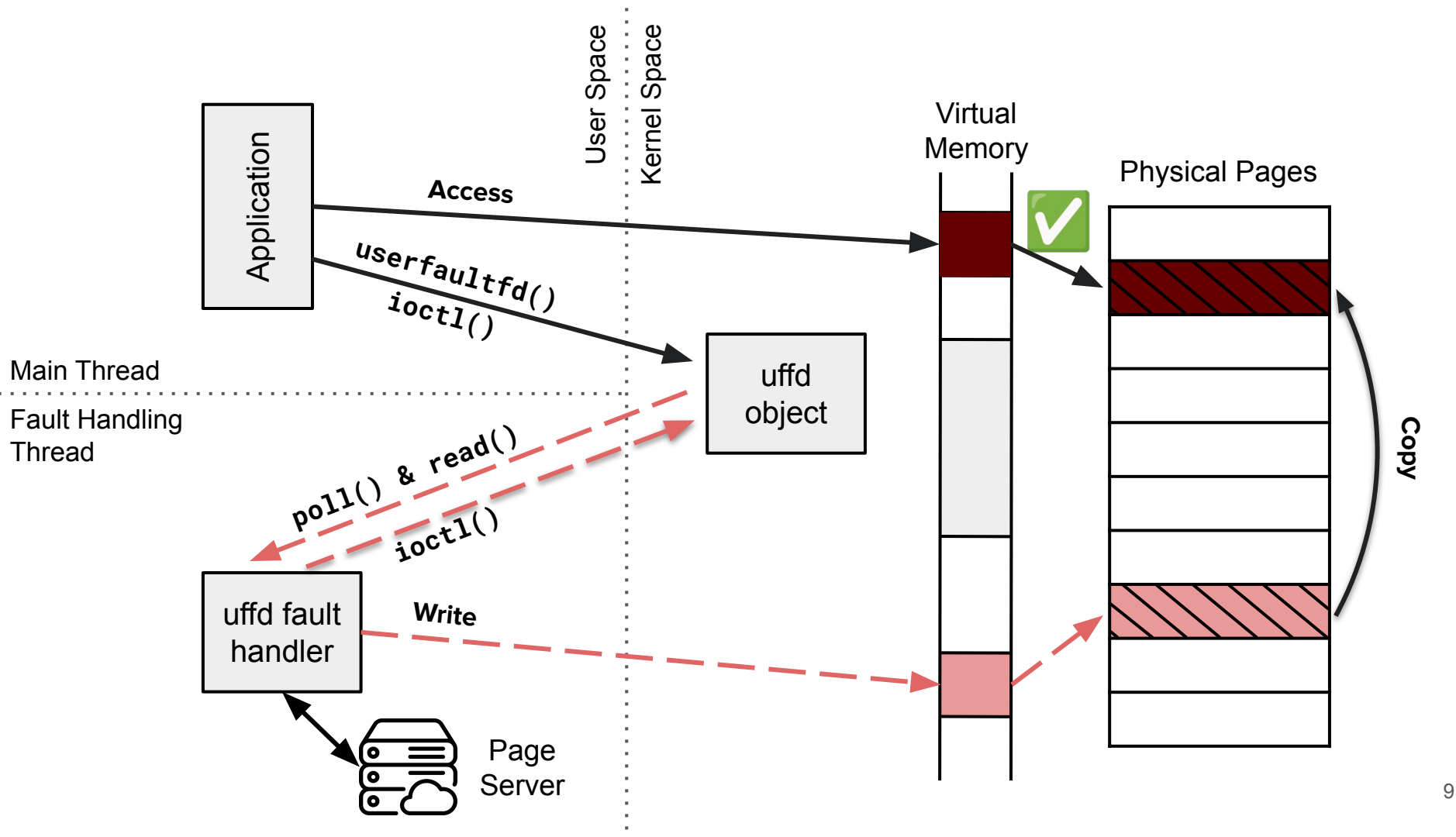
User Space Page Fault Handling

- `userfaultfd()`
- Added in Linux 4.3 (2015)
- Original motivation: post-copy VM migration
- Separate thread runs fault routine





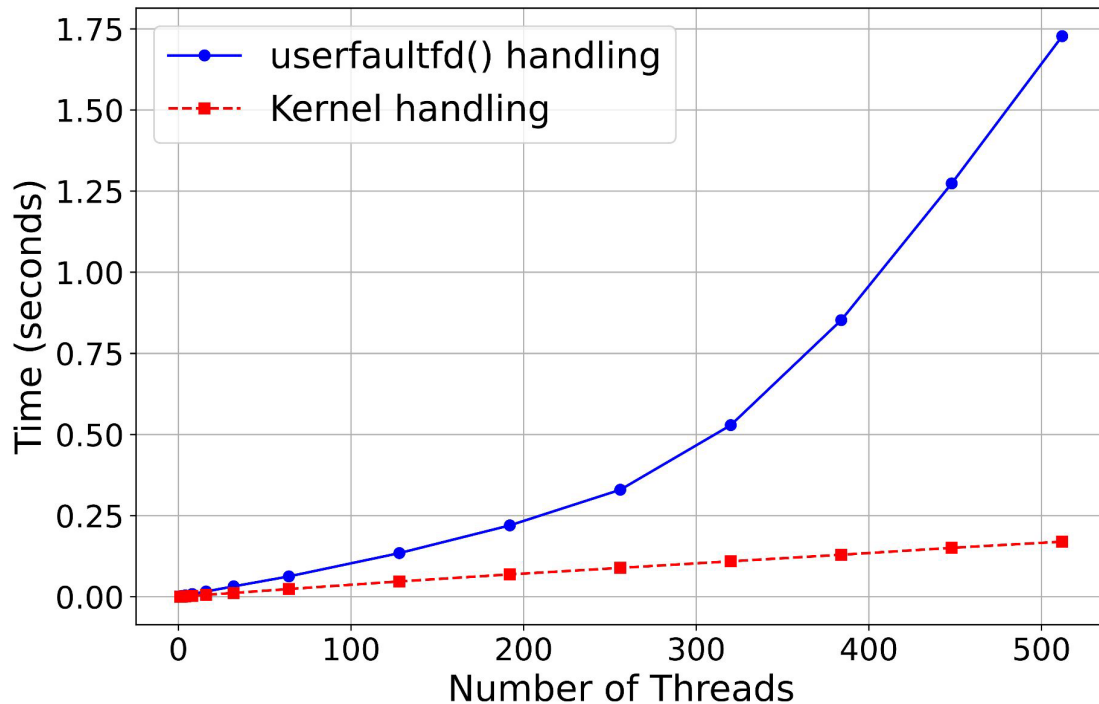




Performance Limitations of `userfaultfd()`

- Fault-handling thread leads to context switch
- At least three kernel-user space crossings
- Excess data copying

Scalability Limitations of userfaultfd()

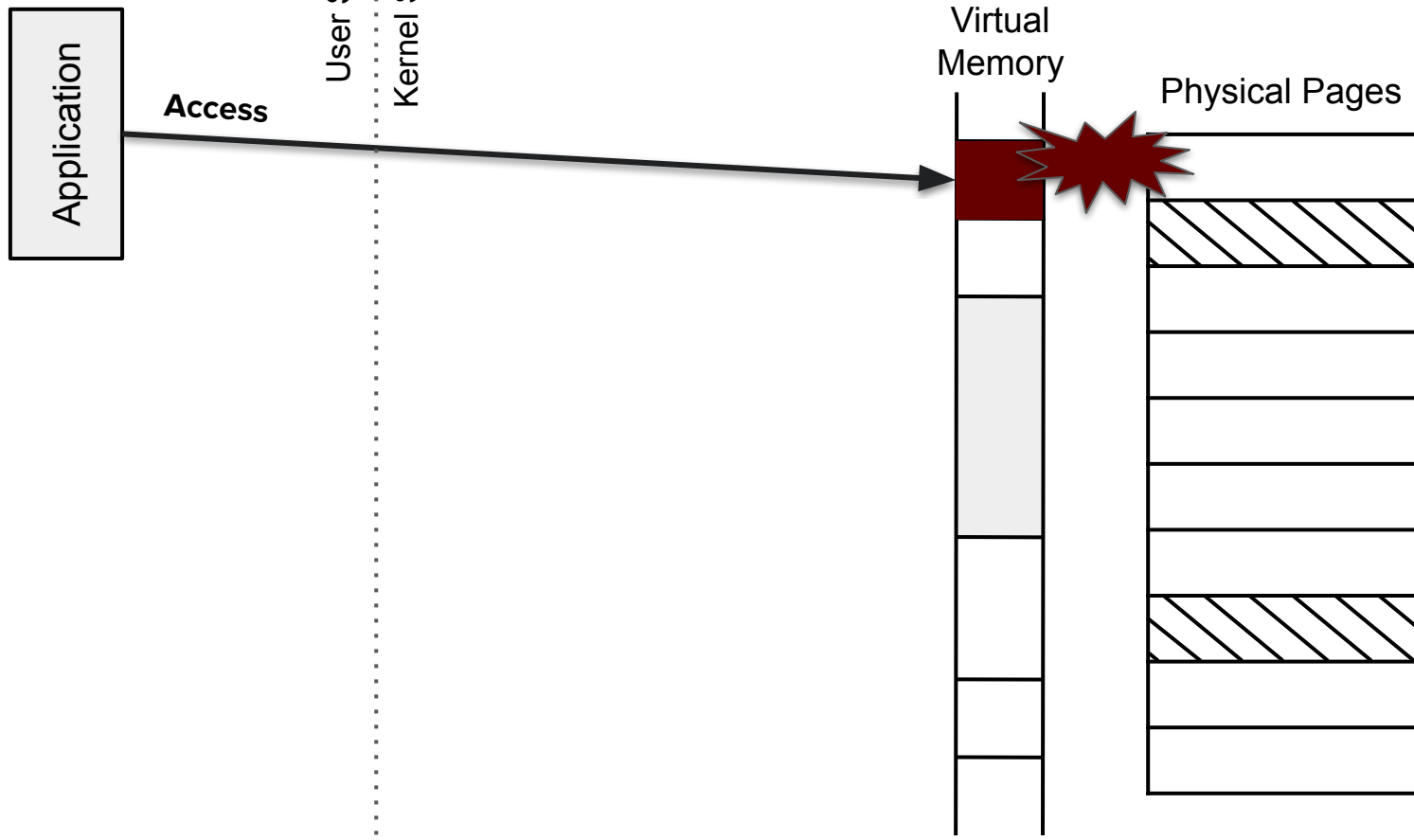


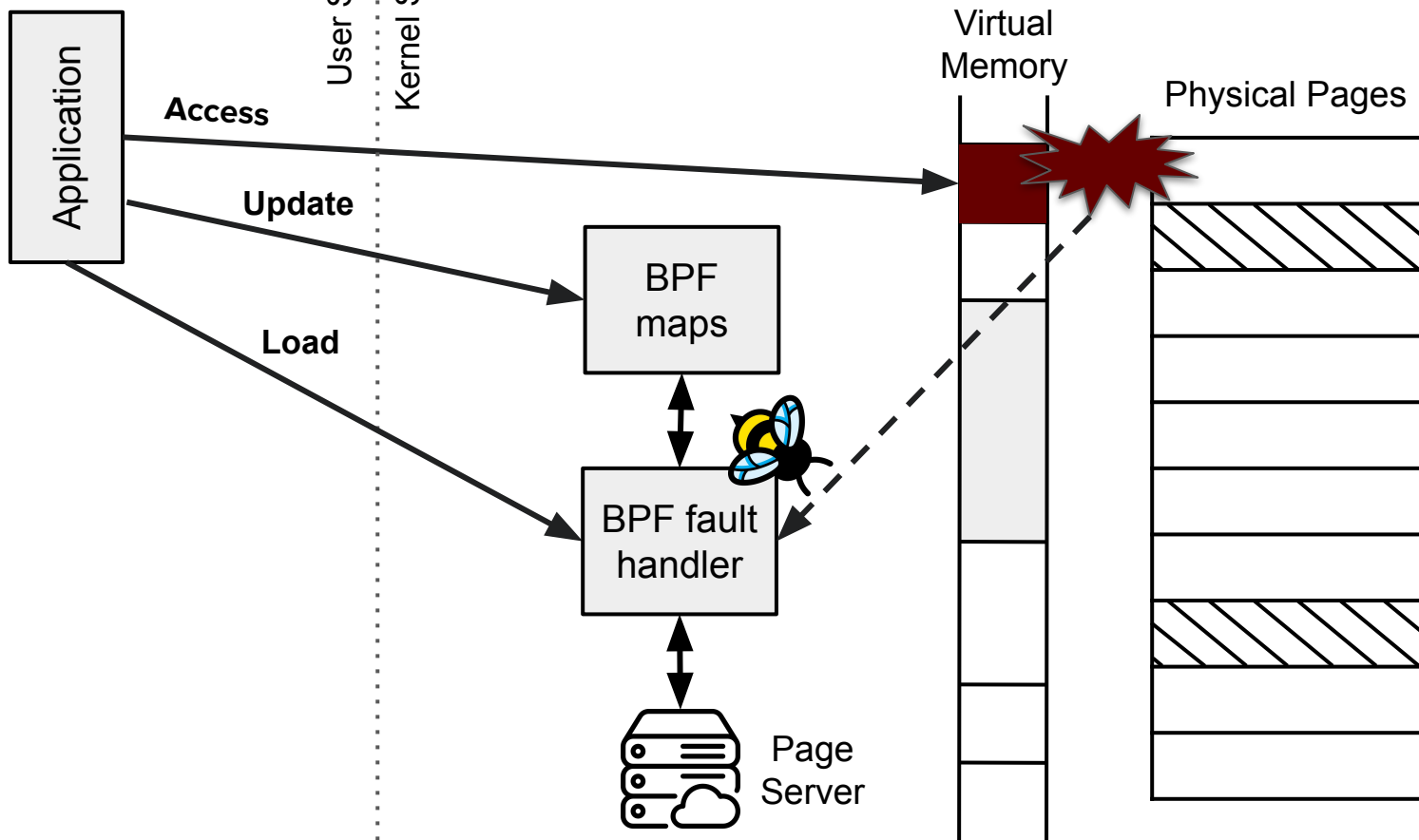
Security Limitations of `userfaultfd()`

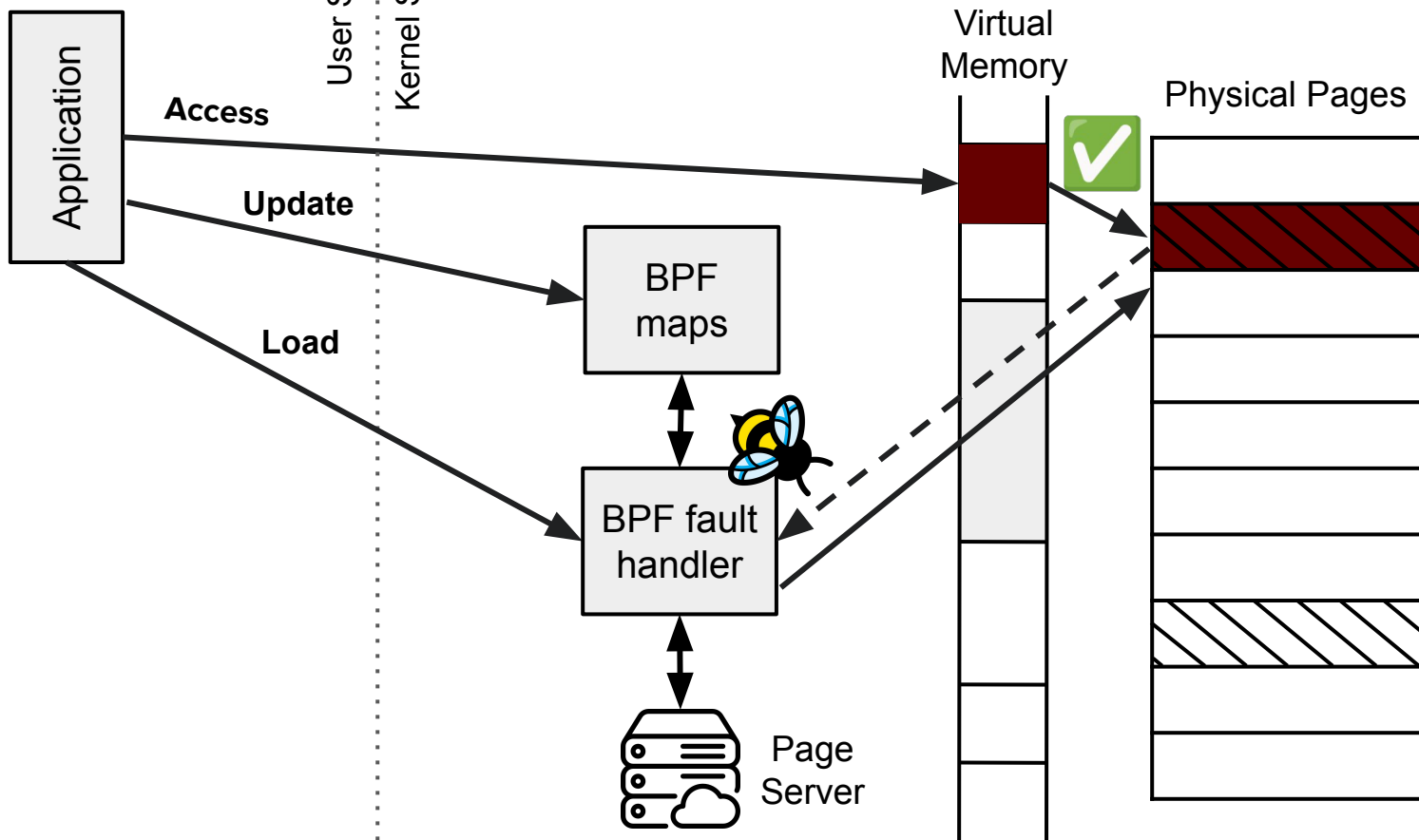
- Exploited in a number of vulnerabilities
 - Indefinitely block kernel execution at a specific point
- Disabled in some container runtimes by default

eBPF Page Fault Handling

- Push fault-handling routine into kernel
- Per-VMA eBPF programs







Benefits of eBPF Fault Handling

- No fault-handling thread – each thread handles own page faults
- At least one kernel-user space crossing
- Potential for zero-copy fault handling

Applications

- Post-copy VM migration
- Faster Java garbage collection
- Accelerating start-up of serverless functions
- Performant memory debugging
- more...

Conclusion

- Applications desire custom page fault handling
- `userfaultfd()` has real limitations
- We propose eBPF-based page fault handling

Thank you!

Contact:

Tal Zussman
tz2294@columbia.edu

Teng Jiang
tj2488@columbia.edu