FOSDEM‘24
February 3, 2024

Linux Binary Compatible Unikernels

How your Application runs on Unikraft

Simon Kuenzer

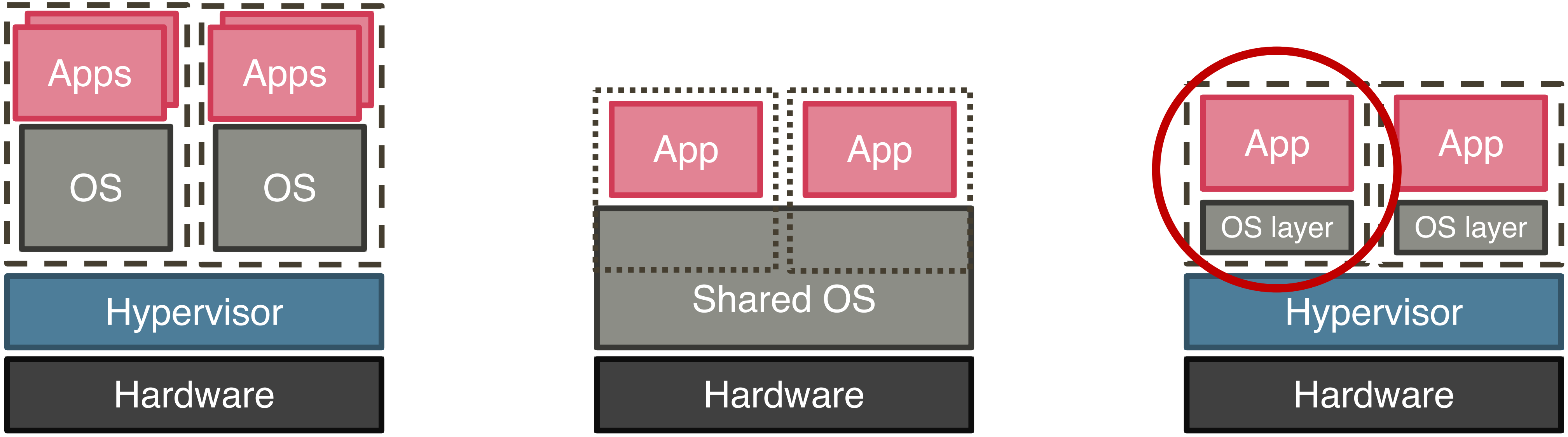
Project Founder & Lead Maintainer

CTO & Co-Founder
Unikraft GmbH
simon@unikraft.io

1

Unikraft: The Unikernel SDK

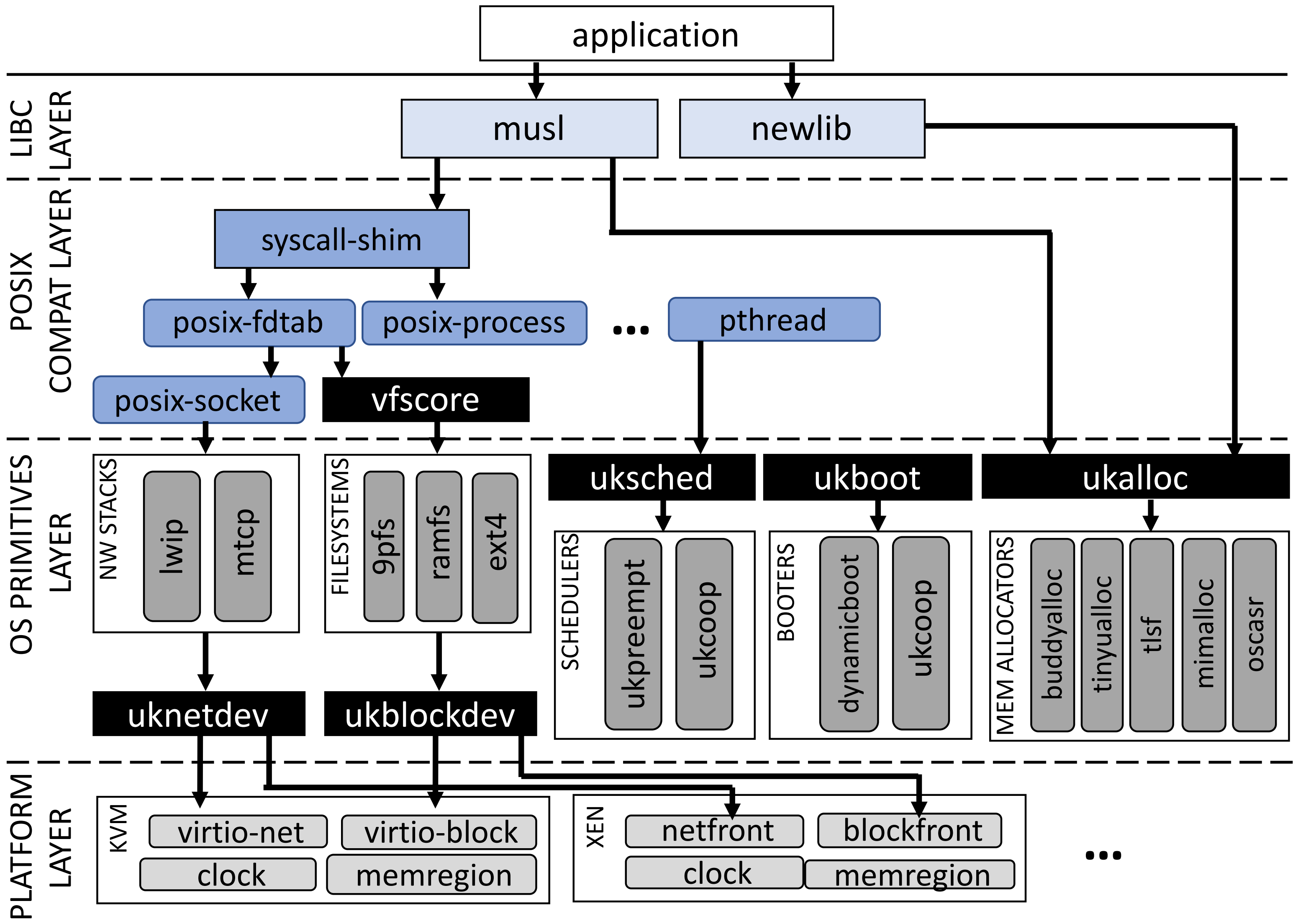
Unikernel Primer



Traditional VMs　　Containers　　Unikernel VMs

- Single purpose: One application & one target platform
  - Flat and single address spac
  - Only necessary kernel components
  - Small TCB and memory footprint

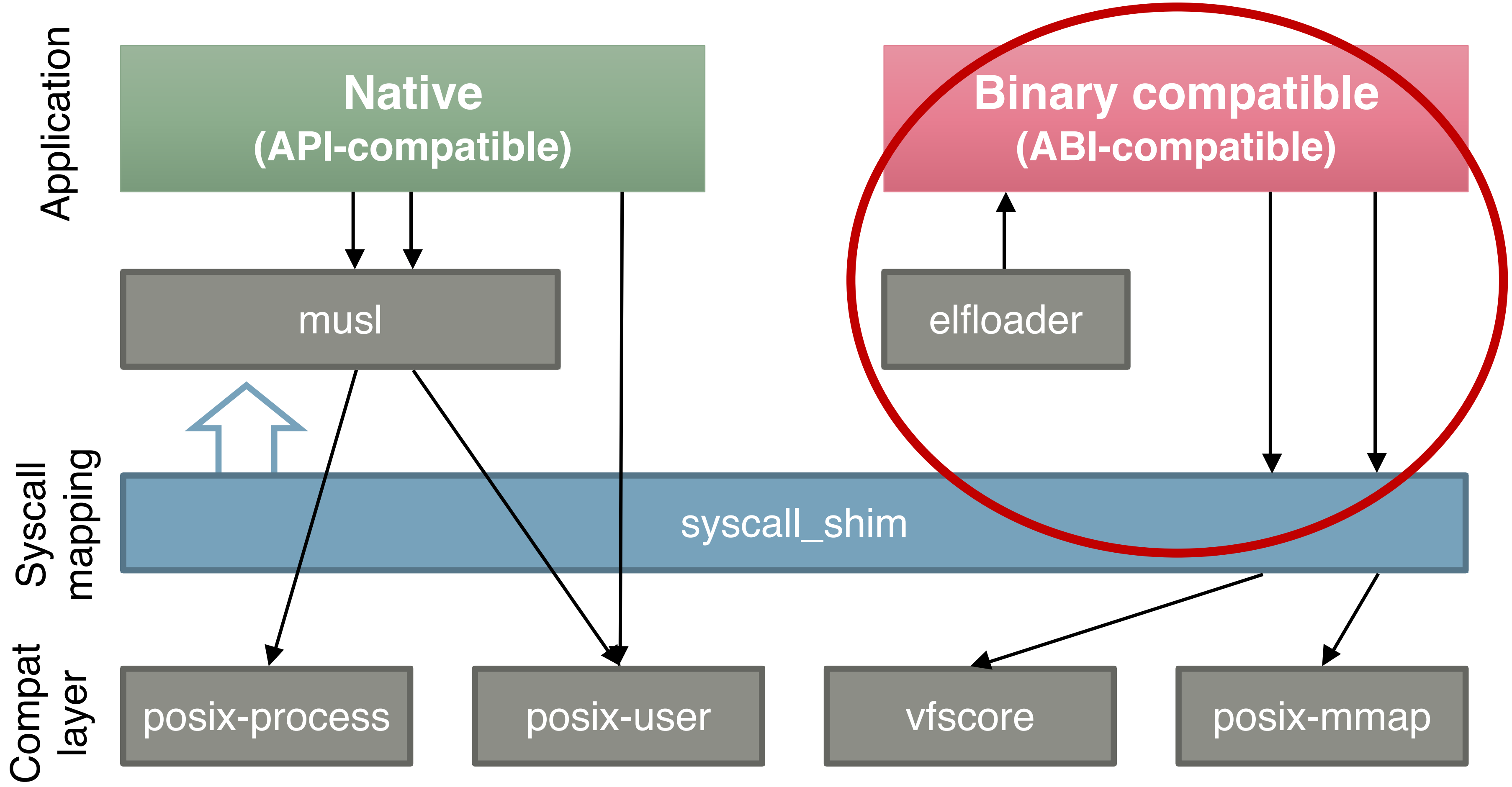
Unikraft's (Micro)-Library Stack



Current project focus: Linux Compatibility

- Our vision: Seamless application support
  - → Most software is developed for Linux
  - → Remove obstacles for running them on Unikraft

The 2 Approaches for Compatibility

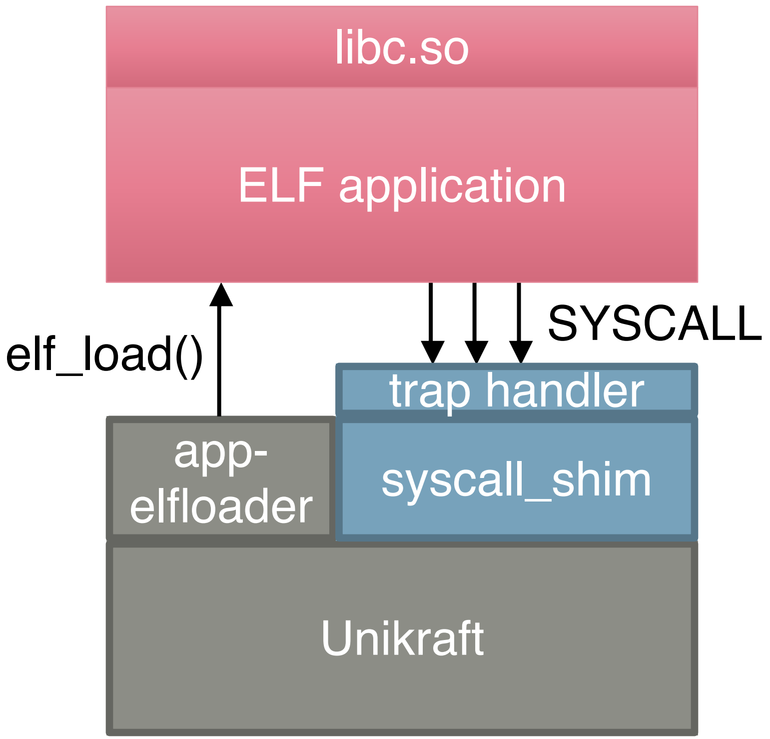


2

Loading ELF Binaries

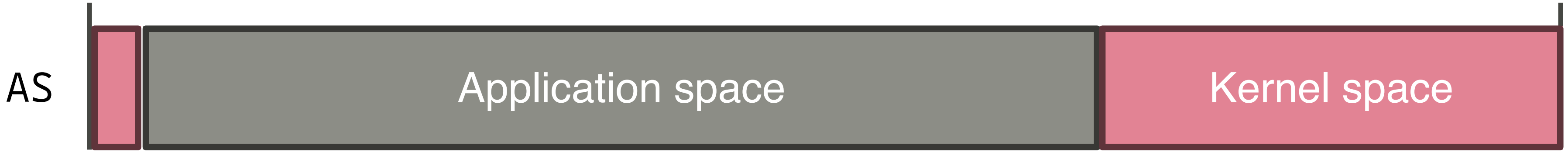
Loading ELF Binaries

- Straight-forward process:
  1) Parse & load executable/loader
  2) Prepare entrance stack, jump to entrance
  3) Interact with system calls



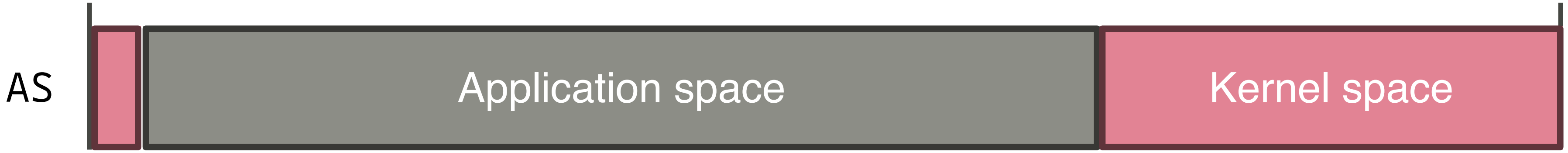
Challenge PIE vs. Non-PIE Executables

- Non-PIE dictates AS-layout
  - Single AS → only <u>one</u> non-PIE app
  - Limits area where (uni-)kernel relies



Challenge PIE vs. Non-PIE Executables

- Non-PIE dictates AS-layout
  - Single AS → only one non-PIE app
  - Limits area where (uni-)kernel relies

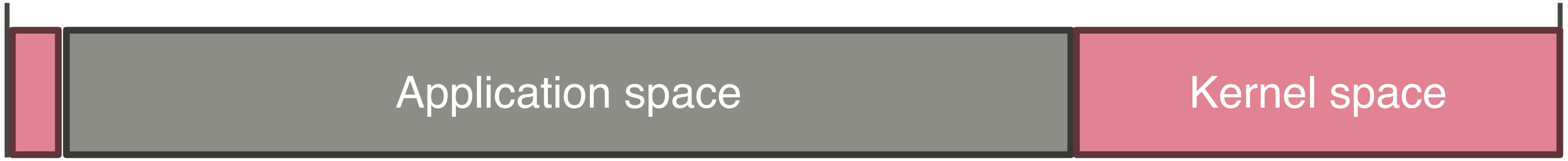


- PIE provides AS-layout flexibility
  - Multiple apps in single AS possible
  - No AS-switch on context switches

Challenge PIE vs. Non-PIE Executables

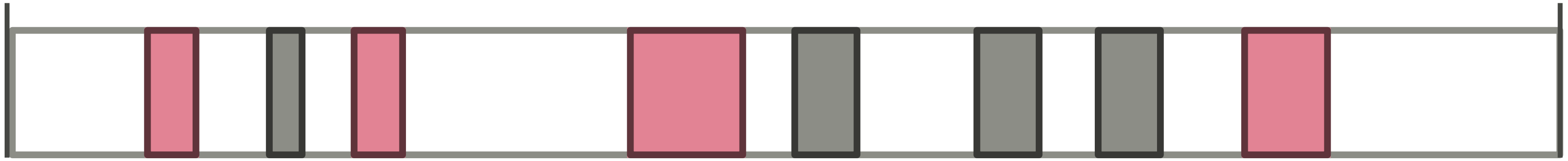
- Non-PIE dictates AS-layout
  - Single AS → only <u>one</u> non-PIE app
  - Limits area where (uni-)kernel relies

AS



- PIE provides AS-layout flexibility
  - Multiple apps in single AS possible
  - No AS-switch on context switches
  - *Opportunity:*
    Full-stack ASLR with max. entropy

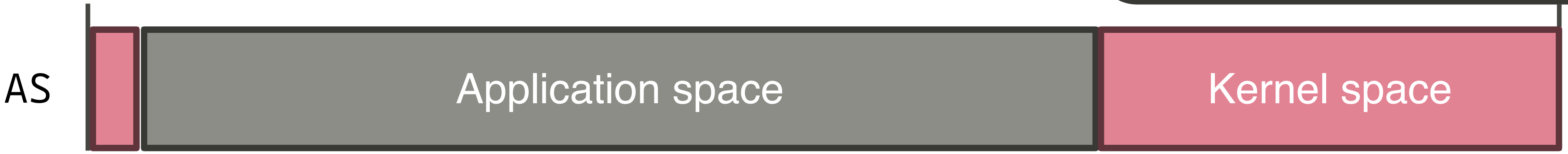
AS



Challenge PIE vs. Non-PIE Executables

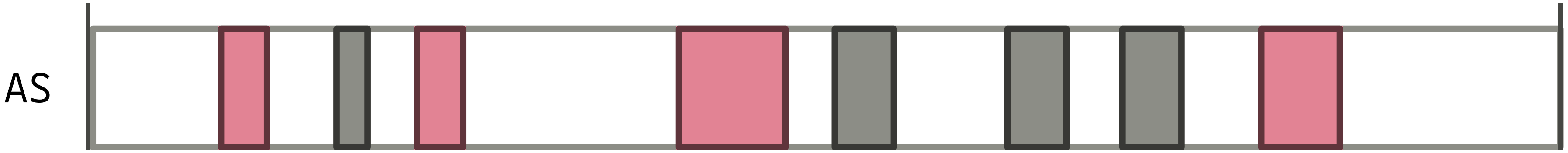
- Non-PIE dictates AS-layout
  - Single AS → only <u>one</u> non-PIE app
  - Limits area where (uni-)kernel relies

> Go binaries still commonly built without PIE for Linux
> Interesting read:
> https://rain-1.github.io/golang-aslr.html



- PIE provides AS-layout flexibility
  - Multiple apps in single AS possible
  - No AS-switch on context switches
  - *Opportunity:*
    Full-stack ASLR with max. entropy

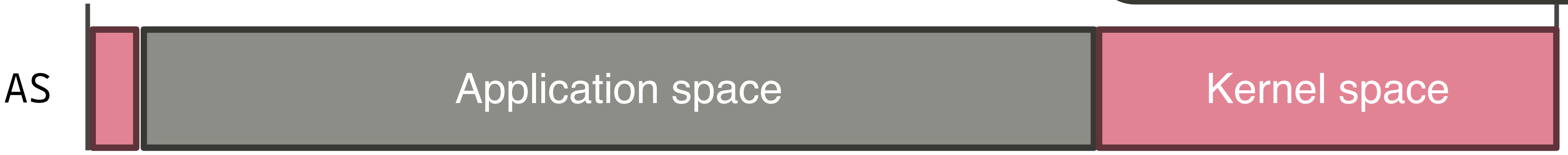
> Major distros moved to PIE for security hardening with ASLR ~5-20 years ago
> https://isopenbsdsecu.re/mitigations/pie/
> https://wiki.debian.org/Hardening/PIEByDefaultTransition



Challenge PIE vs. Non-PIE Executables

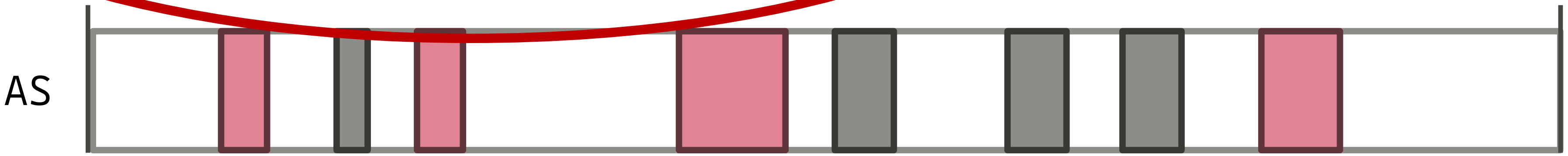
- Non-PIE dictates AS-layout
  - Single AS → only <u>one</u> non-PIE app
  - Limits area where (uni-)kernel relies

> Go binaries still commonly built without PIE for Linux
> Interesting read:
> https://rain-1.github.io/golang-aslr.html



- PIE provides AS-layout flexibility
  - Multiple apps in single AS possible
  - No AS-switch on context switches
  - *Opportunity:*
    Full-stack ASLR with max. entropy

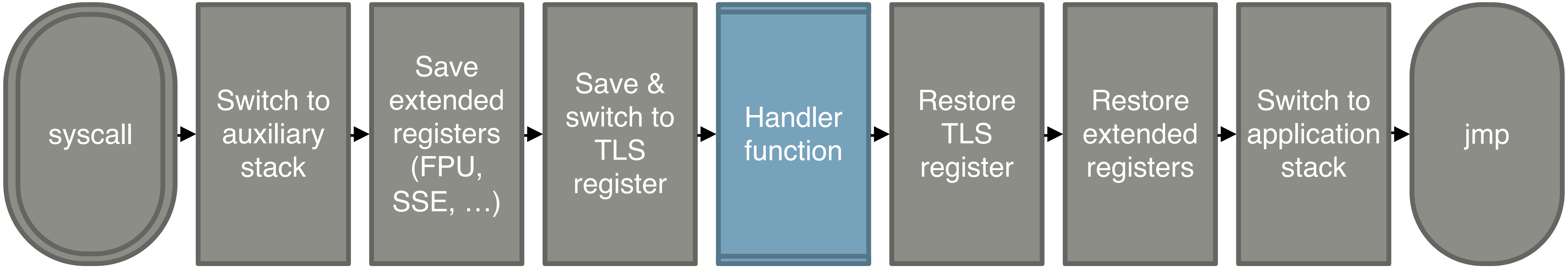
> Major distros moved to PIE for security hardening with ASLR ~5-20 years ago
> https://isopenbsdsecu.re/mitigations/pie/
> https://wiki.debian.org/Hardening/PIEByDefaultTransition



3

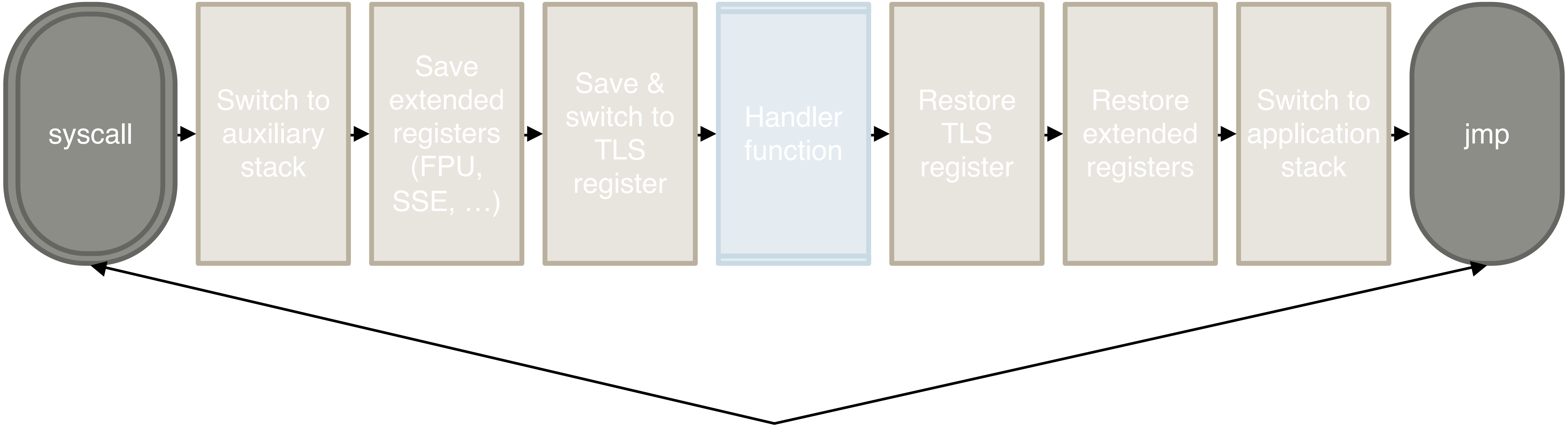
System Calls

System Call Trap Handler



**here: x86_64*

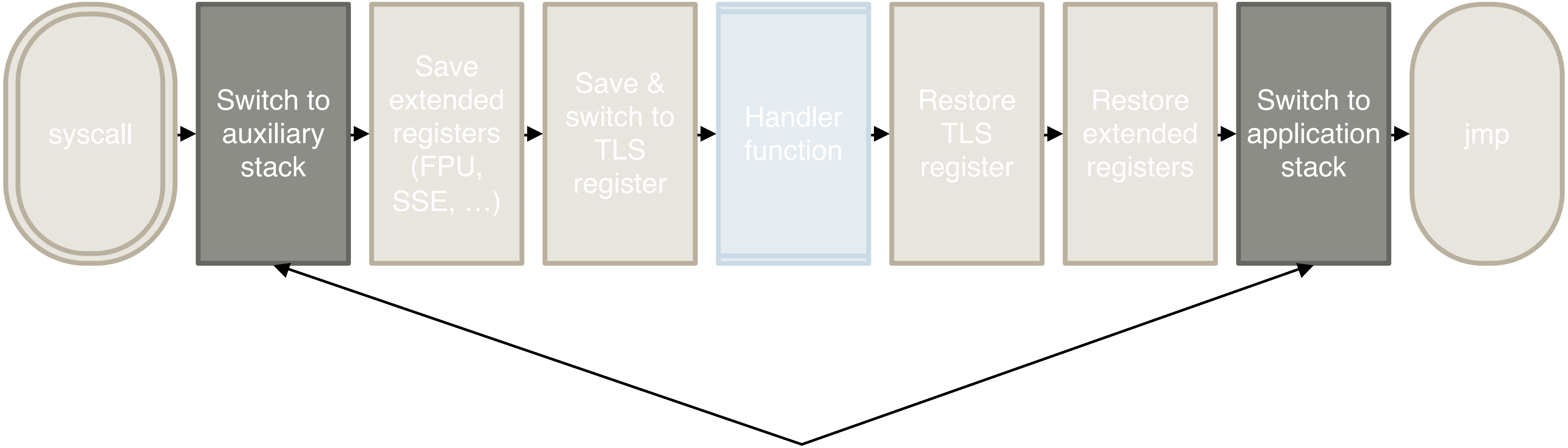
System Call Trap Handler



- Special instruction
  - Takes care of protection domain switch (that we do not need)
- *x86_64:* `jmp` *instead of* `sysret` *because of implicit privilege mode change to ring 3 [1]*

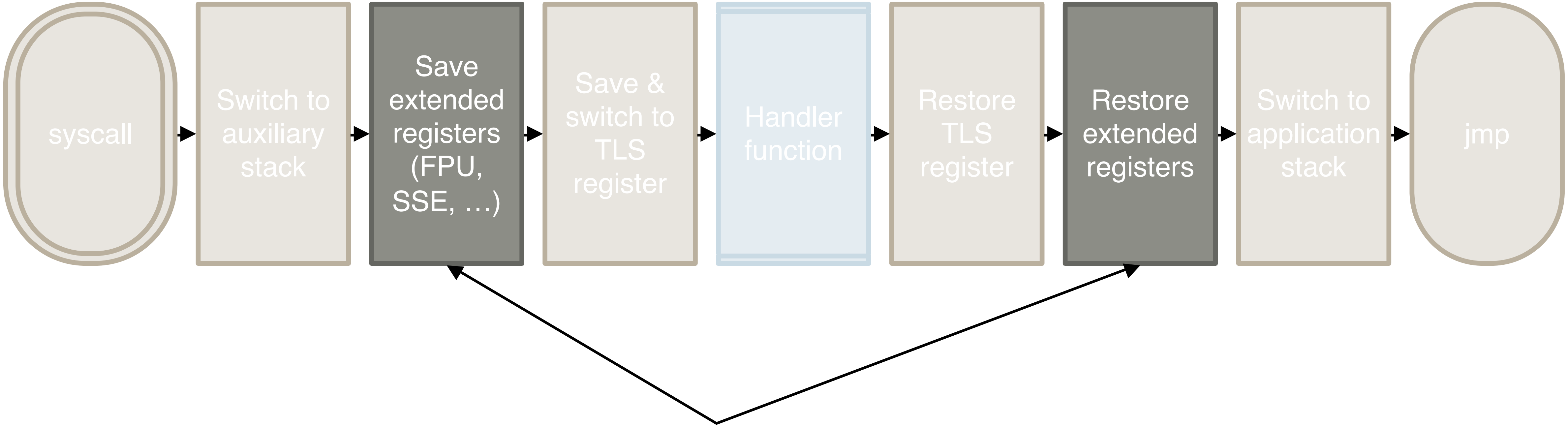
[1] P. Olivier, et al., A binary-compatible unikernel, VEE 2019, https://dl.acm.org/doi/10.1145/3313808.3313817

System Call Trap Handler



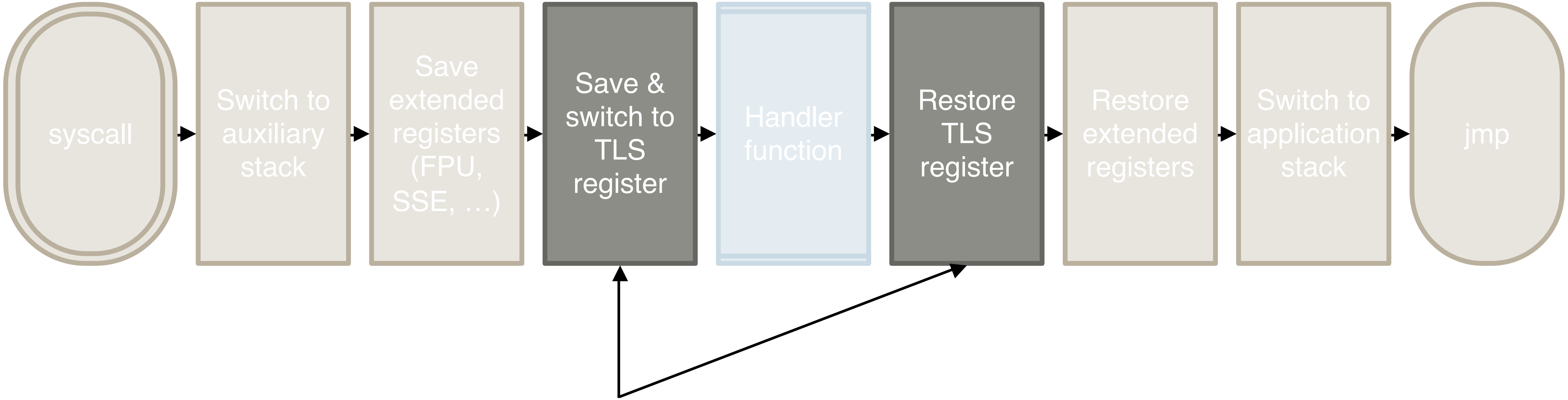
- Needed to be compliant with Linux ABI:
  The system call handler must not require a userland stack
- In reality: Only needed for apps where userland stack is too small (e.g., go)

System Call Trap Handler



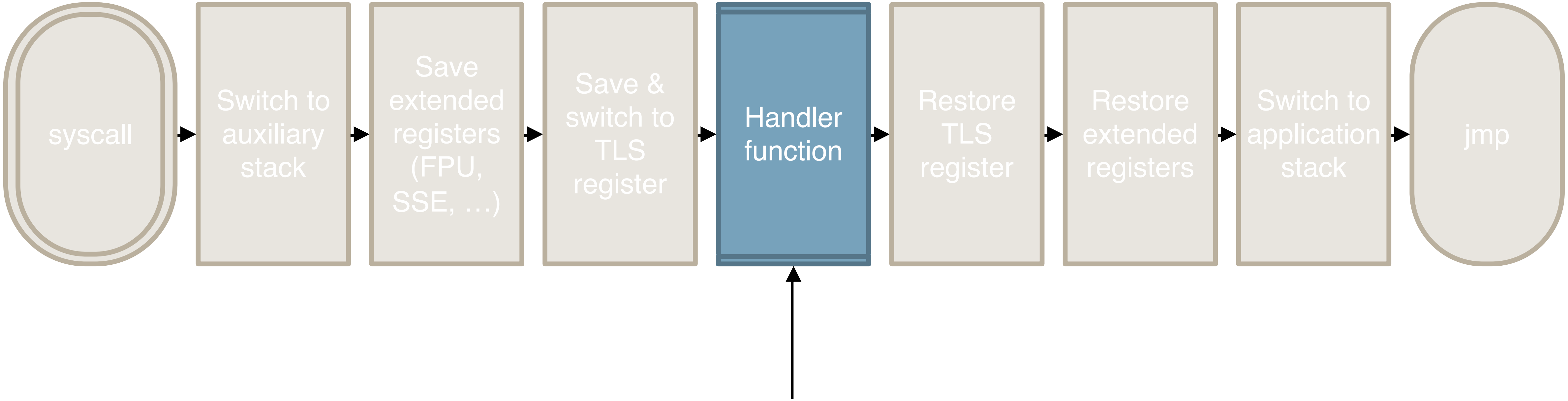
- Needed if we compile Unikraft with full CPU features utilization

System Call Trap Handler



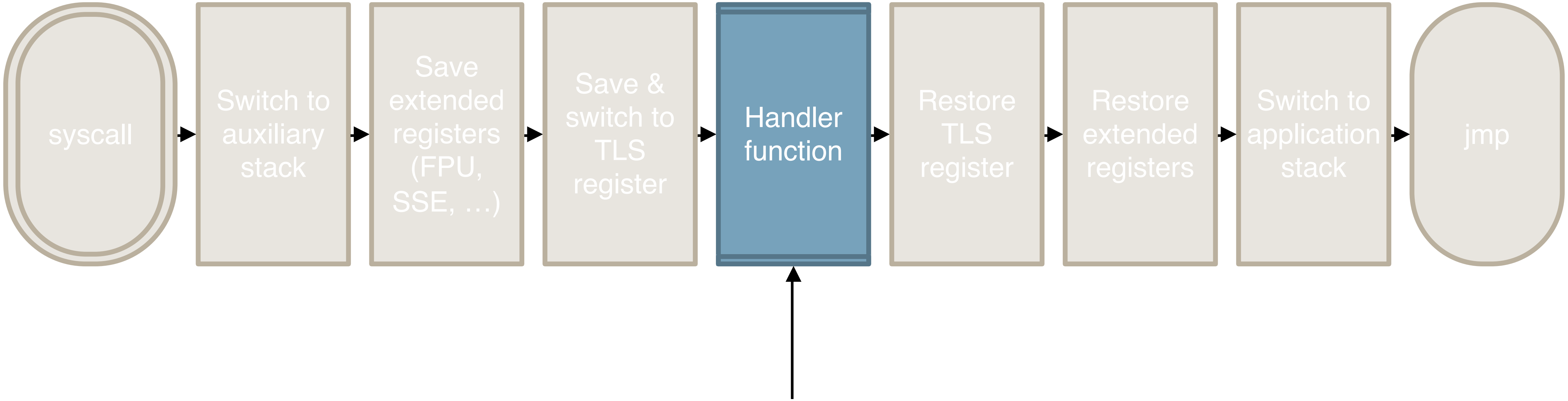
- TLS used as TCB in Unikraft
  - Compartmentalization of library implementations (no central TCB structure definiton needed)

System Call Trap Handler

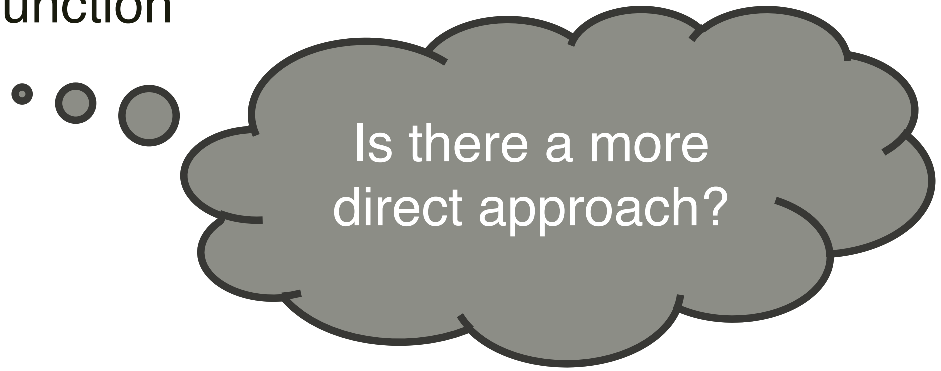


- Actual system call handler function

System Call Trap Handler

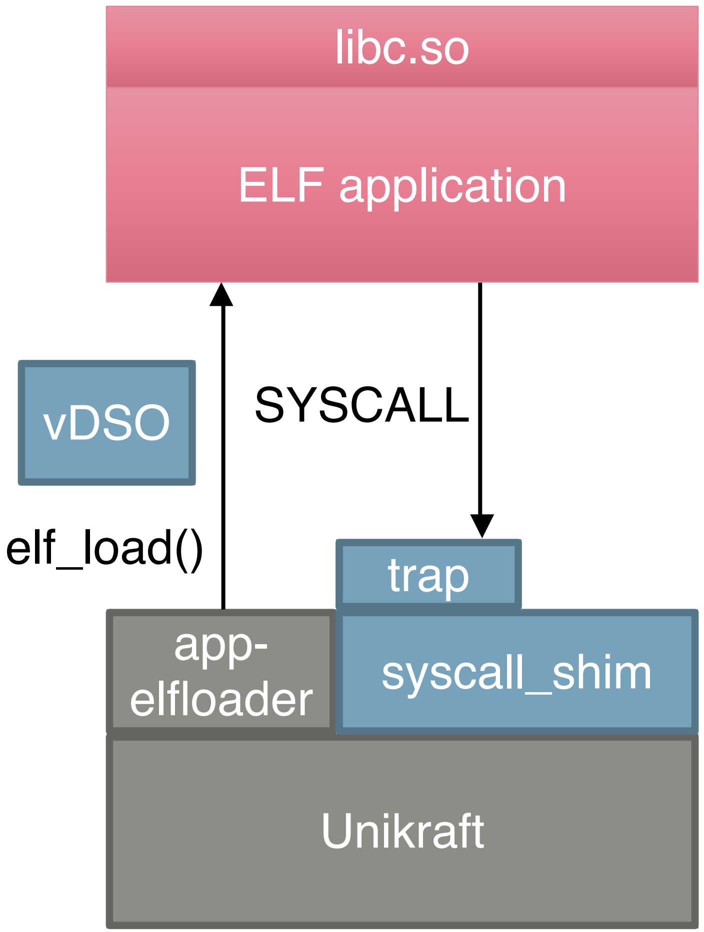


- Actual system call handler function



vDSO and __kernel_vsyscall()

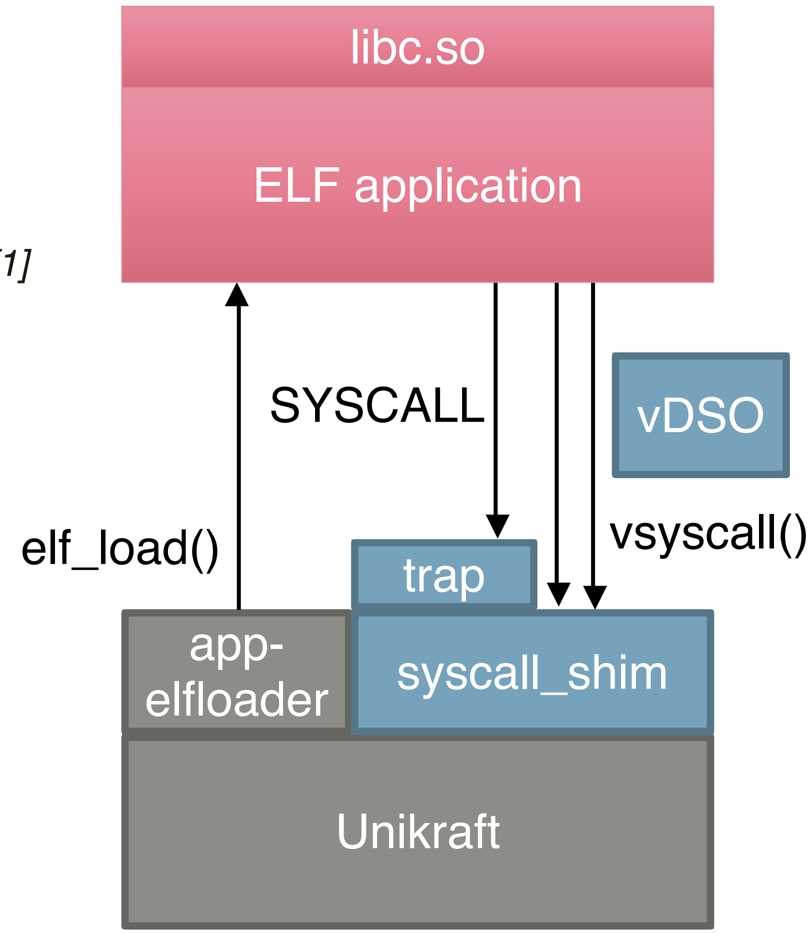
- vDSO[1] in Unikraft is a symbol lookup table only
  - Within single-AS/single-protection domain we can directly execute kernel functions



[1] https://man7.org/linux/man-pages/man7/vdso.7.html
[2] System V Application Binary Interface, 3.2.1 Registers, https://gitlab.com/x86-psABIs/x86-64-ABI

vDSO and __kernel_vsyscall()

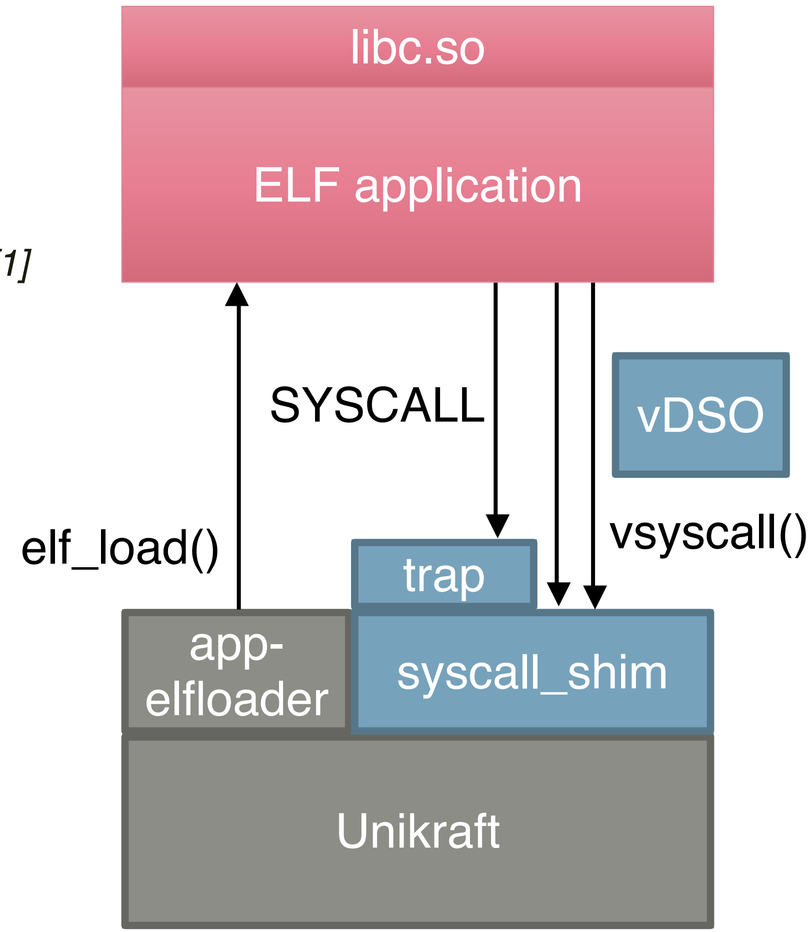
- vDSO[1] in Unikraft is a symbol lookup table only
  - Within single-AS/single-protection domain we can directly execute kernel functions
- Resurrect __**kernel_vsyscall()**
  - *Origin i386: Switch between int_0x80/sysenter/syscall depending on CPU [1]*
  - Idea: Use this mechanism to enter Unikraft
    - Normal function call
    - <u>No</u> trap, interrupt or privilege domain change
    - <u>No need</u> to save & restore extended context [2]



[1] https://man7.org/linux/man-pages/man7/vdso.7.html
[2] System V Application Binary Interface, 3.2.1 Registers, https://gitlab.com/x86-psABIs/x86-64-ABI

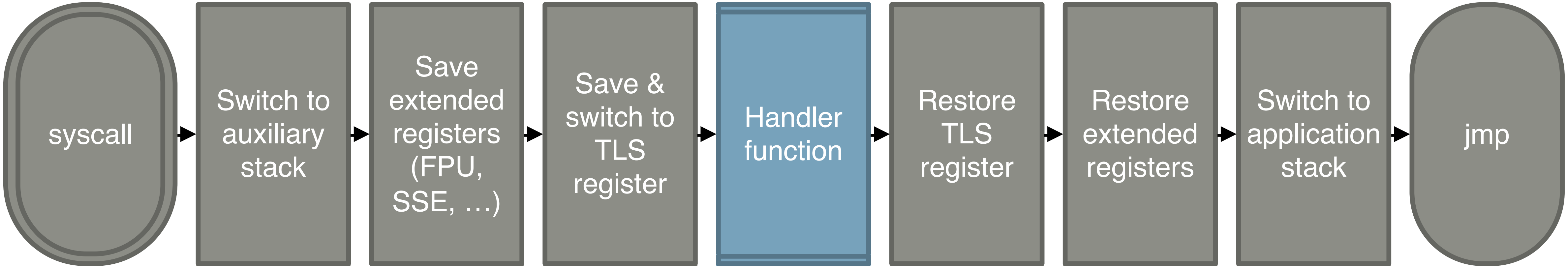
vDSO and __kernel_vsyscall()

- vDSO[1] in Unikraft is a symbol lookup table only
  - Within single-AS/single-protection domain we can directly execute kernel functions
- Resurrect __**kernel_vsyscall()**
  - *Origin i386: Switch between int_0x80/sysenter/syscall depending on CPU [1]*
  - Idea: Use this mechanism to enter Unikraft
    - Normal function call
    - No trap, interrupt or privilege domain change
    - No need to save & restore extended context [2]
  - Patch application's libc.so
    - Most syscalls done via libc wrappers

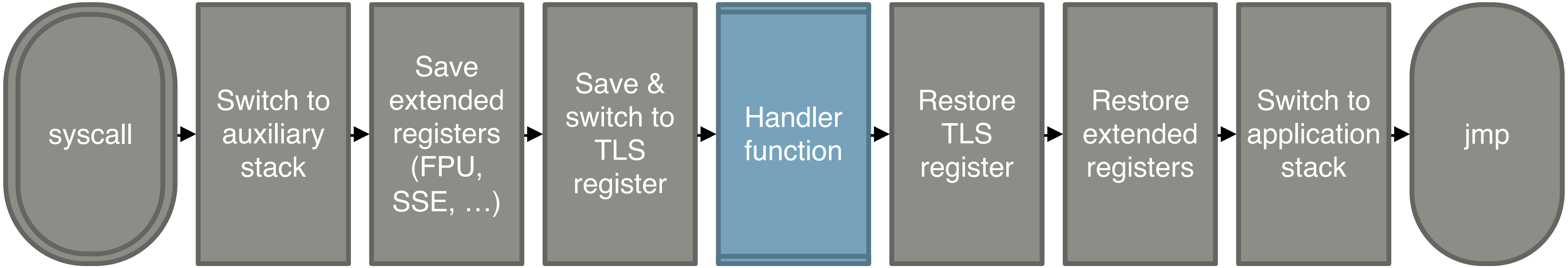


[1] https://man7.org/linux/man-pages/man7/vdso.7.html
[2] System V Application Binary Interface, 3.2.1 Registers, https://gitlab.com/x86-psABIs/x86-64-ABI

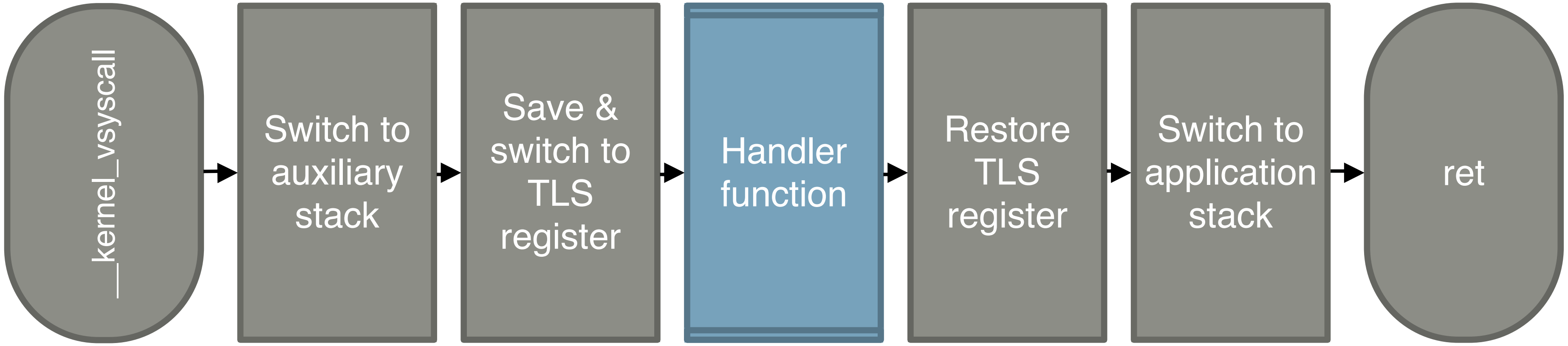
System Call Trap Handler



System Call Trap Handler



Function call __kernel_vsyscall()

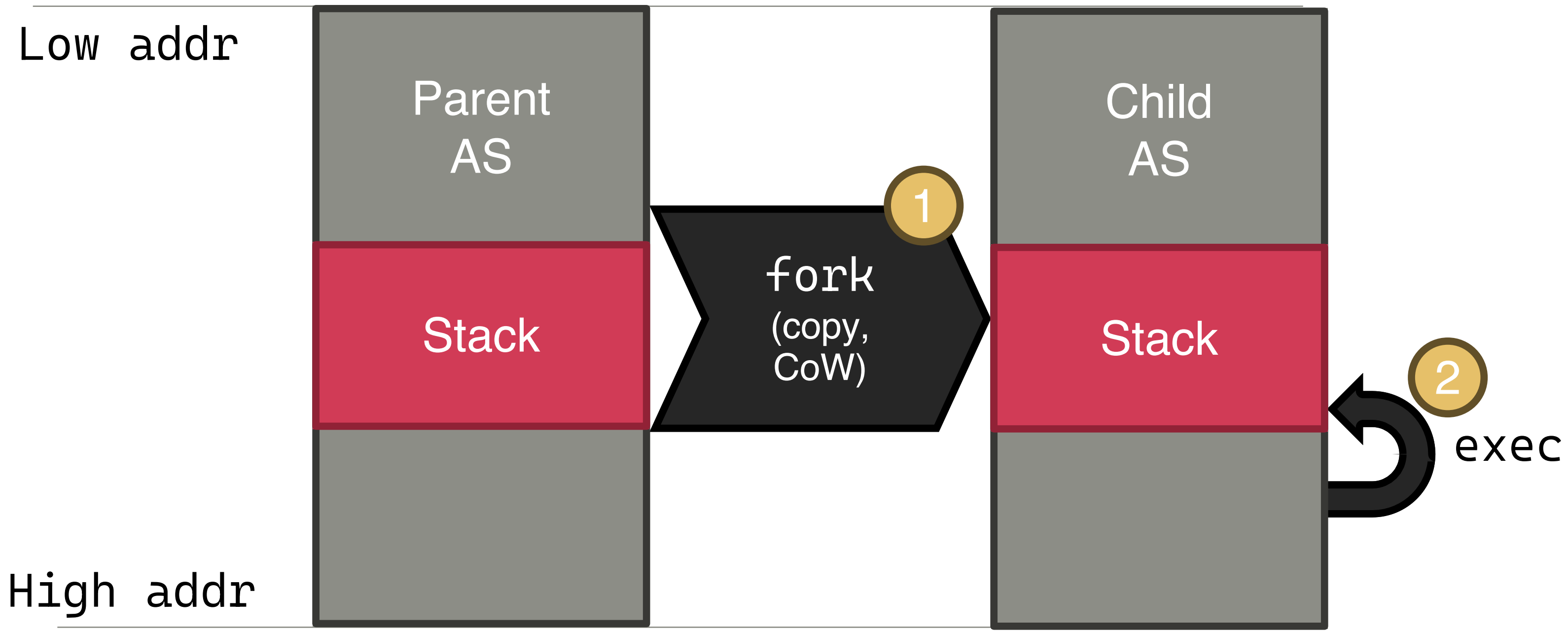


4

The **fork** Dilemma

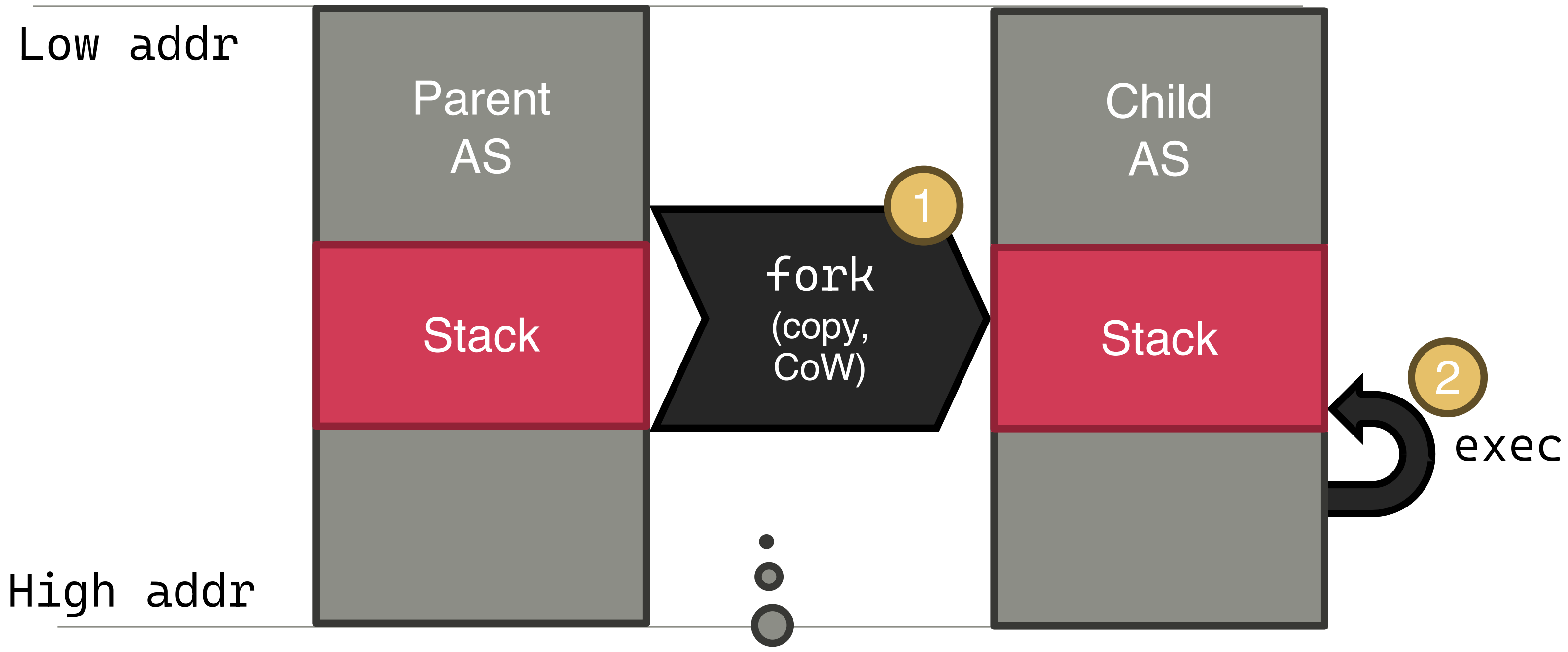
The `fork` Dilemma

- `fork` traditionally used for
  a) Creating worker processes
  b) Instantiating new applications with `fork` + `exec`



The `fork` Dilemma

- `fork` traditionally used for
  a) Creating worker processes
  b) Instantiating new applications with `fork` + `exec`

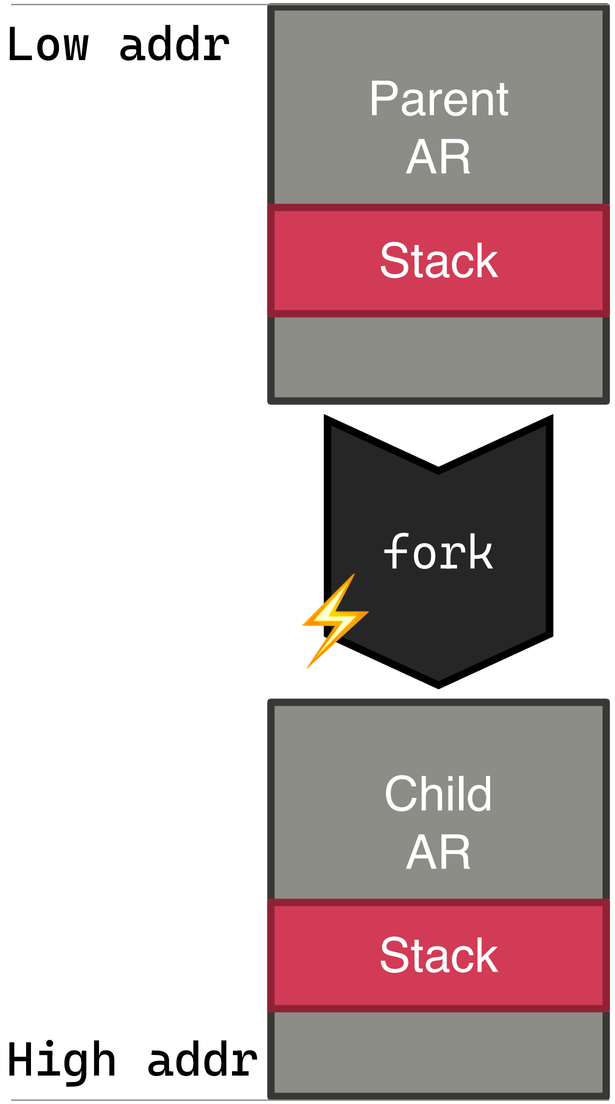


→ Issue: Mechanism relies on per-process ASes

fork in a Unikernel

- Single AS: Child must be located at different address range as the parent
  - Copy&Patching hardly possible without compiler support, e.g.,
    - return addresses on the stack
    - absolute pointers

  → Worker processes cannot be created this way 😢
  luckily, recent software prefer multi-thread model instead



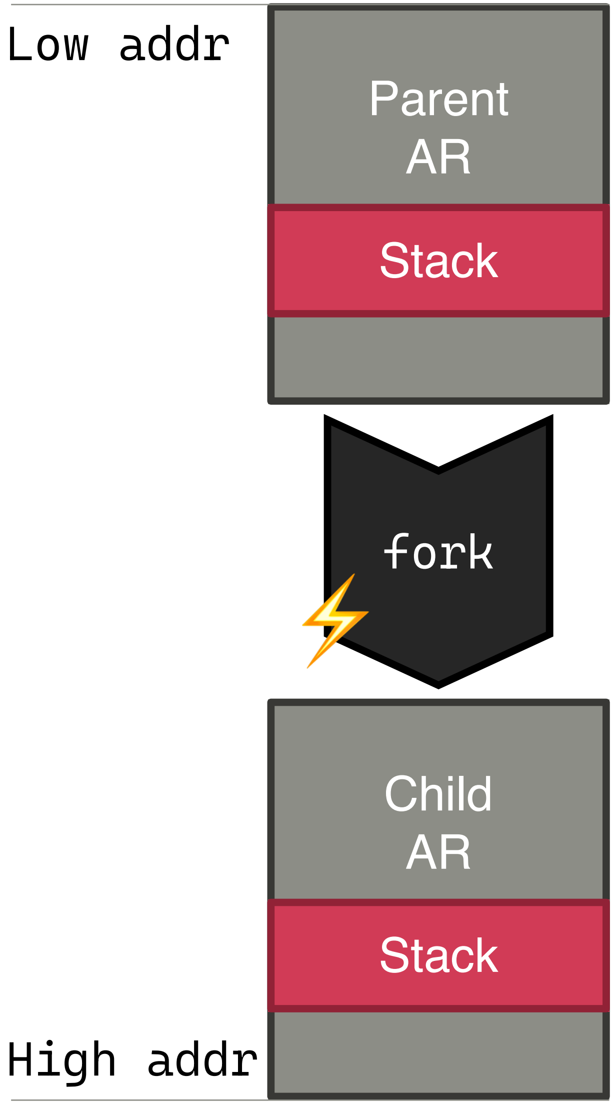
[1] A. Baumann, et al., A fork() in the road, ACM HotOS'19,
https://www.microsoft.com/en-us/research/uploads/prod/2019/04/fork-hotos19.pdf

`fork` in a Unikernel

- Single AS: Child must be located at different address range as the parent
  - Copy&Patching hardly possible without compiler support, e.g.,
    - return addresses on the stack
    - absolute pointers

  → Worker processes cannot be created this way 😢
  luckily, recent software prefer multi-thread model instead

→ Instantiating new application (`fork+exec`)
  - A PIE application can be loaded to any address
  - In principle multi-process with single-AS should work



[1] A. Baumann, et al., A fork() in the road, ACM HotOS'19, https://www.microsoft.com/en-us/research/uploads/prod/2019/04/fork-hotos19.pdf

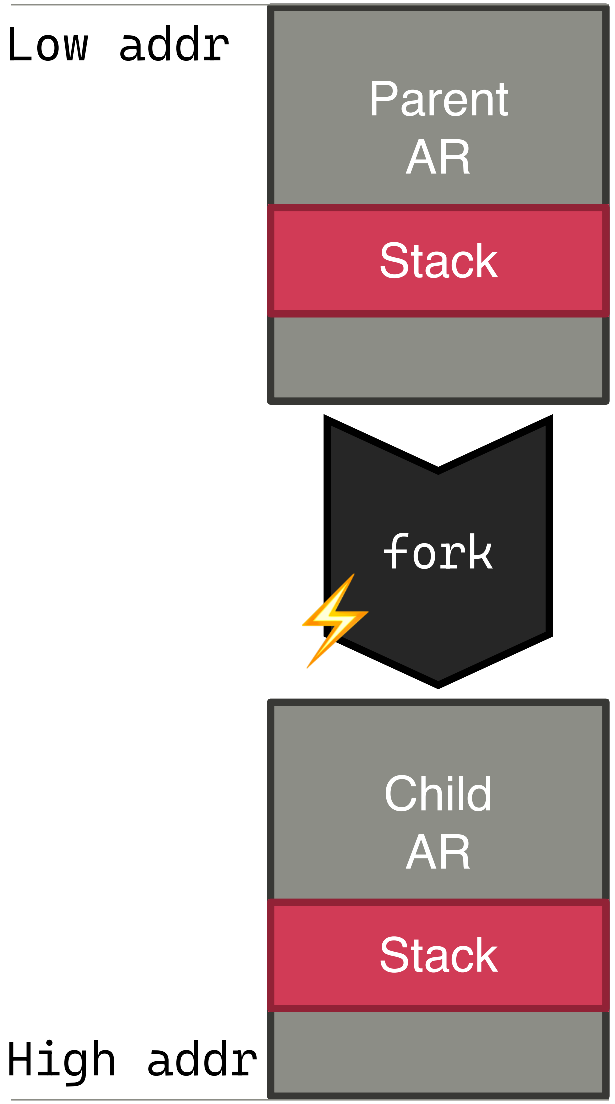
`fork` in a Unikernel

- Single AS: Child must be located at different address range as the parent
  - Copy&Patching hardly possible without compiler support, e.g.,
    - return addresses on the stack
    - absolute pointers

  → Worker processes cannot be created this way 😢
  luckily, recent software prefer multi-thread model instead

→ Instantiating new application (`fork+exec`)
  - A PIE application can be loaded to any address
  - In principle multi-process with single-AS should work

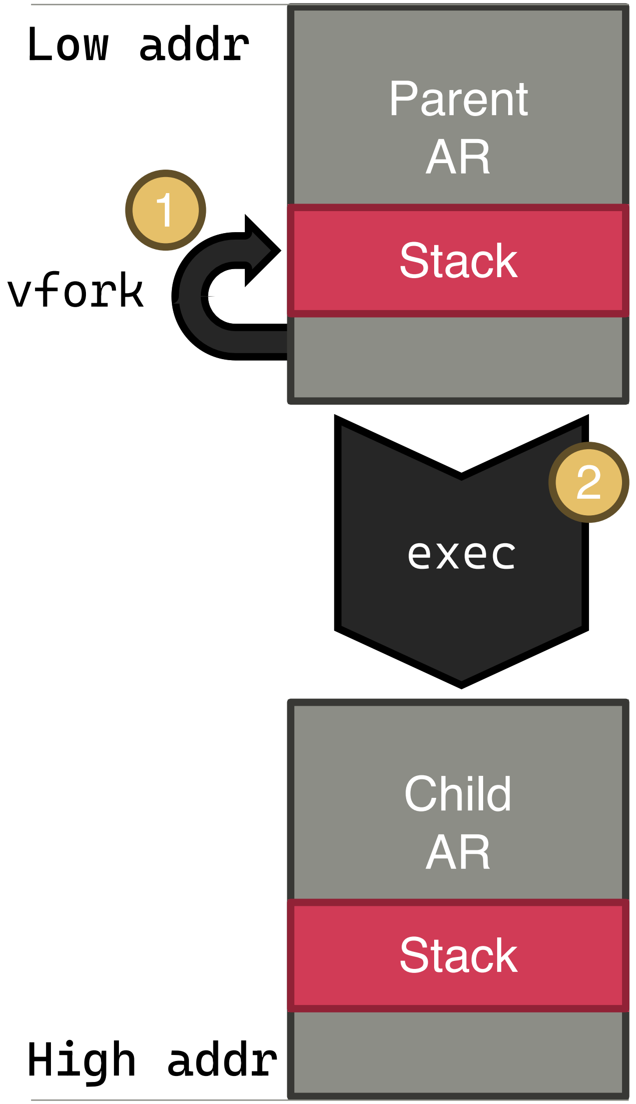
Is fork-exec-model constraining us? [1]



[1] A. Baumann, et al., A fork() in the road, ACM HotOS’19, https://www.microsoft.com/en-us/research/uploads/prod/2019/04/fork-hotos19.pdf

A Solution: `vfork+exec`

- `vfork` [1]: Shares memory and stack with parent
  - No MMU required → we can keep single AS
  - Parent is suspended until child exits or calls `exec`
- `exec`: will drop current memory image and launch a new one from executable
  - → PIE executable loaded to different base address and executed (elfloader)

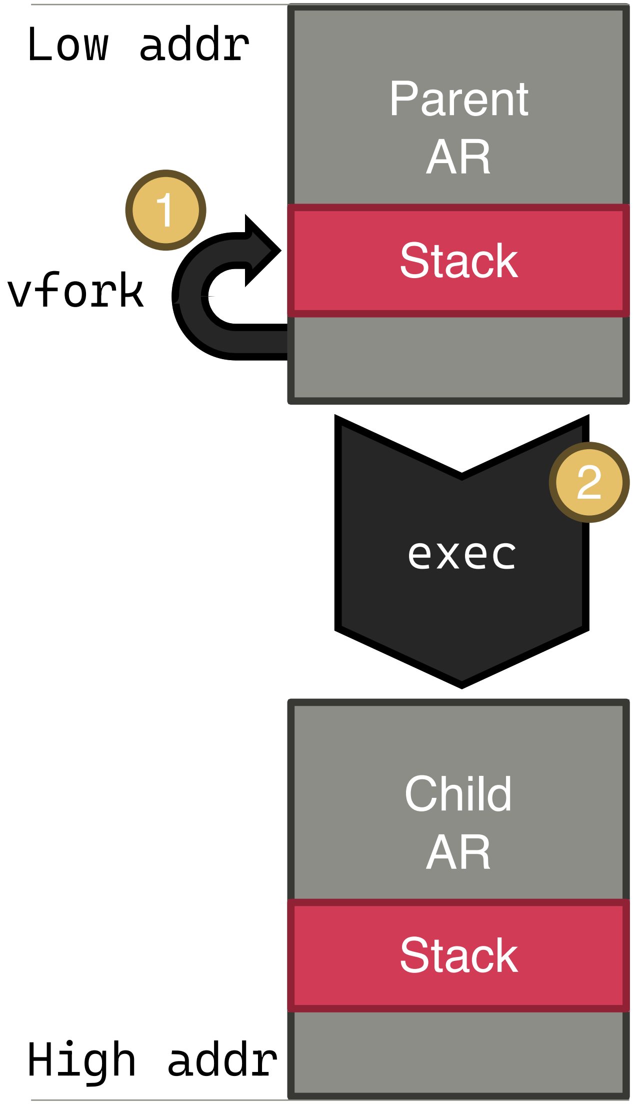


[1] https://man7.org/linux/man-pages/man2/vfork.2.html

A Solution: vfork+exec

- `vfork` [1]: Shares memory and stack with parent
  - No MMU required → we can keep single AS
  - Parent is suspended until child exits or calls `exec`
- `exec`: will drop current memory image and launch a new one from executable
  - → PIE executable loaded to different base address and executed (elfloader)

→ Outlook/Trial: Translate `fork+exec` to `vfork+exec`



[1] https://man7.org/linux/man-pages/man2/vfork.2.html

5

Risk of Bloat due to Linux Compatibility

Risk of Bloat due to Linux Compatibility

- Network interfaces and routing (getifaddr() and co.)
  - Need complex subsystem in between: netlink sockets
  - Alternative: Provide functions directly via the vDSO (trade-off: libc patching)

[1] H. Lefeuvre, et al., Loupe: Driving the Development of OS Compatibility Layers, ASPLOS'24, https://arxiv.org/pdf/2309.15996.pdf
[2] http://refspecs.linuxfoundation.org/fhs

Risk of Bloat due to Linux Compatibility

- Network interfaces and routing (getifaddr() and co.)
  - Need complex subsystem in between: netlink sockets
  - Alternative: Provide functions directly via the vDSO (trade-off: libc patching)
- Applications relying on specific Linux behaviors
  - For example: Preemptive scheduling:
    - e.g., frankenphp, mysql, initialize thread pools with busy waiting

[1] H. Lefeuvre, et al., Loupe: Driving the Development of OS Compatibility Layers, ASPLOS’24, https://arxiv.org/pdf/2309.15996.pdf
[2] http://refspecs.linuxfoundation.org/fhs

Risk of Bloat due to Linux Compatibility

- Network interfaces and routing (getifaddr() and co.)
  - Need complex subsystem in between: netlink sockets
  - Alternative: Provide functions directly via the vDSO (trade-off: libc patching)
- Applications relying on specific Linux behaviors
  - For example: Preemptive scheduling:
    - e.g., frankenphp, mysql, initialize thread pools with busy waiting
- System call stubbing [1]:
  - Not all system calls need a full implementation
    - A number of syscalls can be stubbed (fake-it) but application dependent

[1] H. Lefeuvre, et al., Loupe: Driving the Development of OS Compatibility Layers, ASPLOS'24, https://arxiv.org/pdf/2309.15996.pdf
[2] http://refspecs.linuxfoundation.org/fhs

Risk of Bloat due to Linux Compatibility

- Network interfaces and routing (getifaddr() and co.)
  - Need complex subsystem in between: netlink sockets
  - Alternative: Provide functions directly via the vDSO (trade-off: libc patching)
- Applications relying on specific Linux behaviors
  - For example: Preemptive scheduling:
    - e.g., frankenphp, mysql, initialize thread pools with busy waiting
- System call stubbing [1]:
  - Not all system calls need a full implementation
    - A number of syscalls can be stubbed (fake-it) but application dependent
- Filesystem Hierarchy Standard [2]:
  - Specific files and file systems (e.g., /proc, /etc) at expected places and behavior
    Many of them can resolved by placing files with meaningful content in the VFS

[1] H. Lefeuvre, et al., Loupe: Driving the Development of OS Compatibility Layers, ASPLOS'24, https://arxiv.org/pdf/2309.15996.pdf
[2] http://refspecs.linuxfoundation.org/fhs

Join us!

- OSS project
  unikraft.org
- Get started with kraftkit
  github.com/unikraft/kraftkit
- Code & Contributing
  github.com/unikraft
- Follow us on
  - Discord: https://bit.ly/UnikraftDiscord
  - Twitter: @UnikraftSDK
  - LinkedIn: https://linkedin.com/company/unikraft-sdk

Thank you!