Engineering and Physical Sciences Research Council

Digital Security by Design

SFork: Supporting Complex Multi-Process Applications in a Single Address Space OS

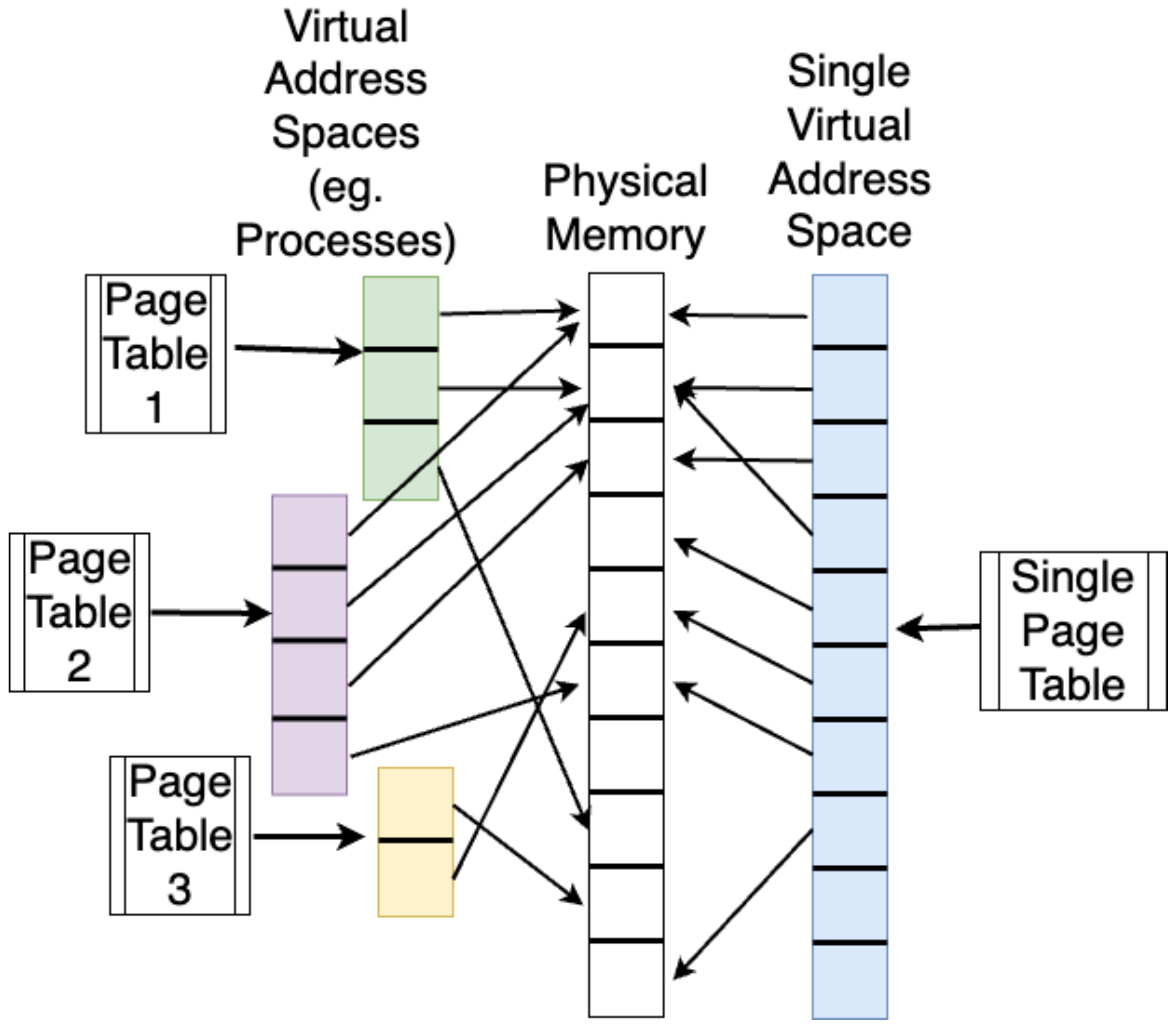
Innovate UK

MANCHESTER 1824
The University of Manchester

John Alistair Kressel, Hugo Lefeuvre, Pierre Olivier

The University of Manchester

Single Address Space OSes (SASOS)



Virtual Address Spaces (eg. Processes)

Physical Memory

Single Virtual Address Space

Page Table 1

Page Table 2

Page Table 3

Single Page Table

Pros

- ✓ No overhead from switching page tables
- ✓ Fast IPC

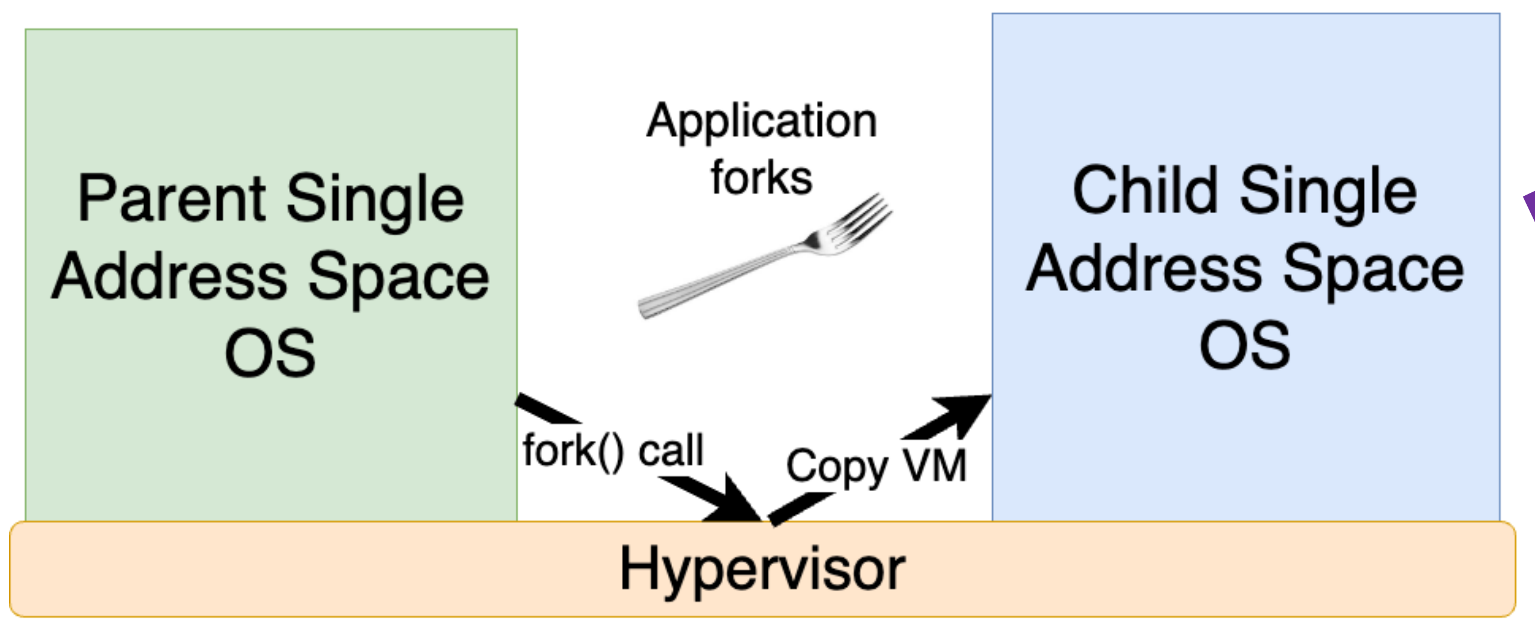
Cons

- ✕ No support for multiple processes through POSIX `fork()`

Problem: Lack of `fork()` Support

SASOSes are incompatible with forking processes which creates a **copy of their address space**.

Most existing solutions treat OS as process and copy entire OS [2,3]



Parent Single Address Space OS

Application forks

fork() call

Copy VM

Child Single Address Space OS

Hypervisor

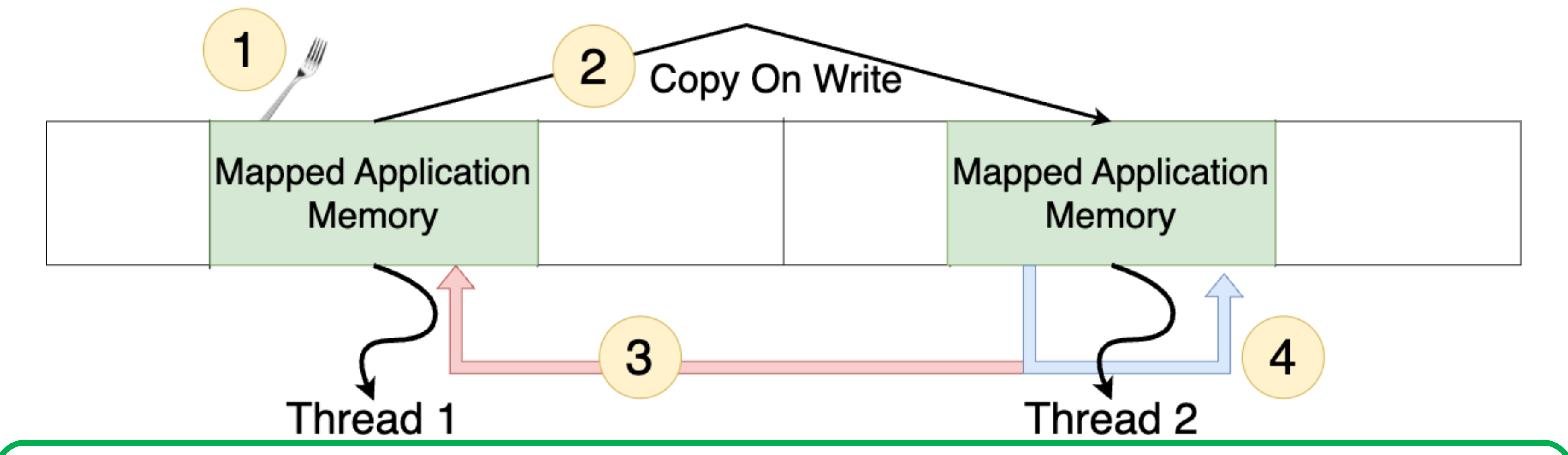
Multiple address spaces!

- ✕ Loses single address space performance advantages by reintroducing multiple address spaces
- ✕ Cloning entire OS is costlier and more resource intensive

Previous attempt at true single address space `fork()` suffers from coarse granularity of memory sharing and no isolation between processes [4]

How can we transparently and securely support POSIX `fork()` in a SASOS without losing the single address space?

SFork: Emulate Processes with Threads in a Single Address Space OS



1

2 Copy On Write

Mapped Application Memory

Mapped Application Memory

3

4

Thread 1

Thread 2

Challenges & Solutions

- Providing isolation between processes
- Ensuring pointers to parent memory are identified and updated in the child

We will solve these problems using CHERI [1]

- In pure capability mode (purecap) all pointers are bounded - processes are restricted to their portion of the address space
- Pointers to parent memory can be identified because capabilities are tagged

Advantages

- ✓ Lower resource consumption than multiple VMs
- ✓ Faster IPCs by using capabilities in the same address space
- ✓ Faster context switches (same page table)
- ✓ Isolation between processes

How SFork Works:

1. Application transparently calls `fork()`
   - New process mapped to another part of the address space
2. Copy On Write
   - Parent process memory copied when modified
3. Identify pointers to parent process
   - Copied memory will contain pointers to parent memory
4. Update pointers to point to child memory

Current Progress (April 2024)

- Unikraft, a popular unikernel, ported to purecap on Morello
- Purecap Unikraft running bare-metal and under bhyve on CHERIBSD host OS
- Paravirtualised I/O (VirtIO) support for purecap Unikraft running under bhyve
- Applications such as SQLite, Redis and a http server running on purecap Unikraft
- Basic `fork()` building blocks implemented with work ongoing

References

[1] *Woodruff, Jonathan, et al. "The CHERI capability model: Revisiting RISC in an age of risk." ISCA 2014*

[2] *Lupu, Costin, et al. "Nephele: Extending Virtualization Environments for Cloning Unikernel-Based VMs." EuroSys 2023*

[3] *Zhang, Yiming, et al. "KylinX: A Dynamic Library Operating System for Simplified and Efficient Cloud Virtualization." USENIX ATC 2018*

[4] *Wilkinson, Tim, et al. Compiling for a 64-bit Single Address Space Architecture Technical Report 1993*