

Hyungjoon Kwon

Systems Software &
Architecture Lab.
Seoul National University

2024.05.10

Project #4: KSM

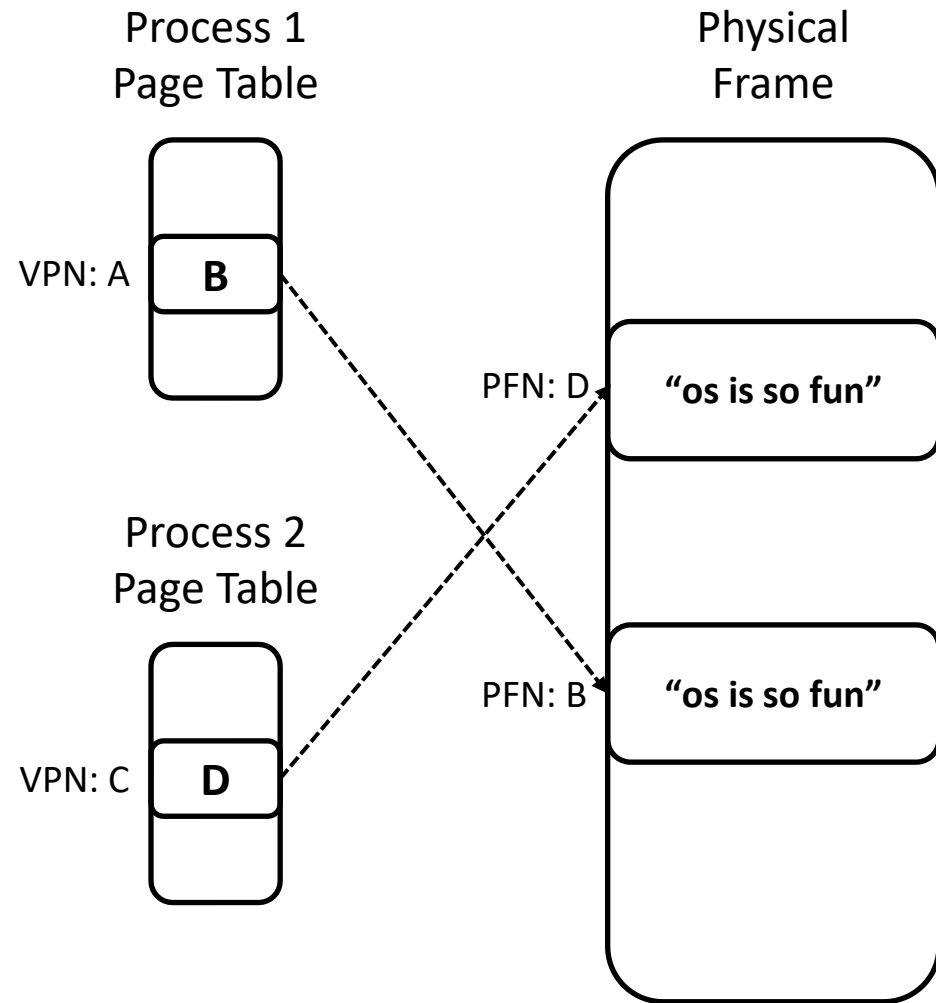
(Kernel Samepage Merging)



KSM (Kernel Samepage Merging)

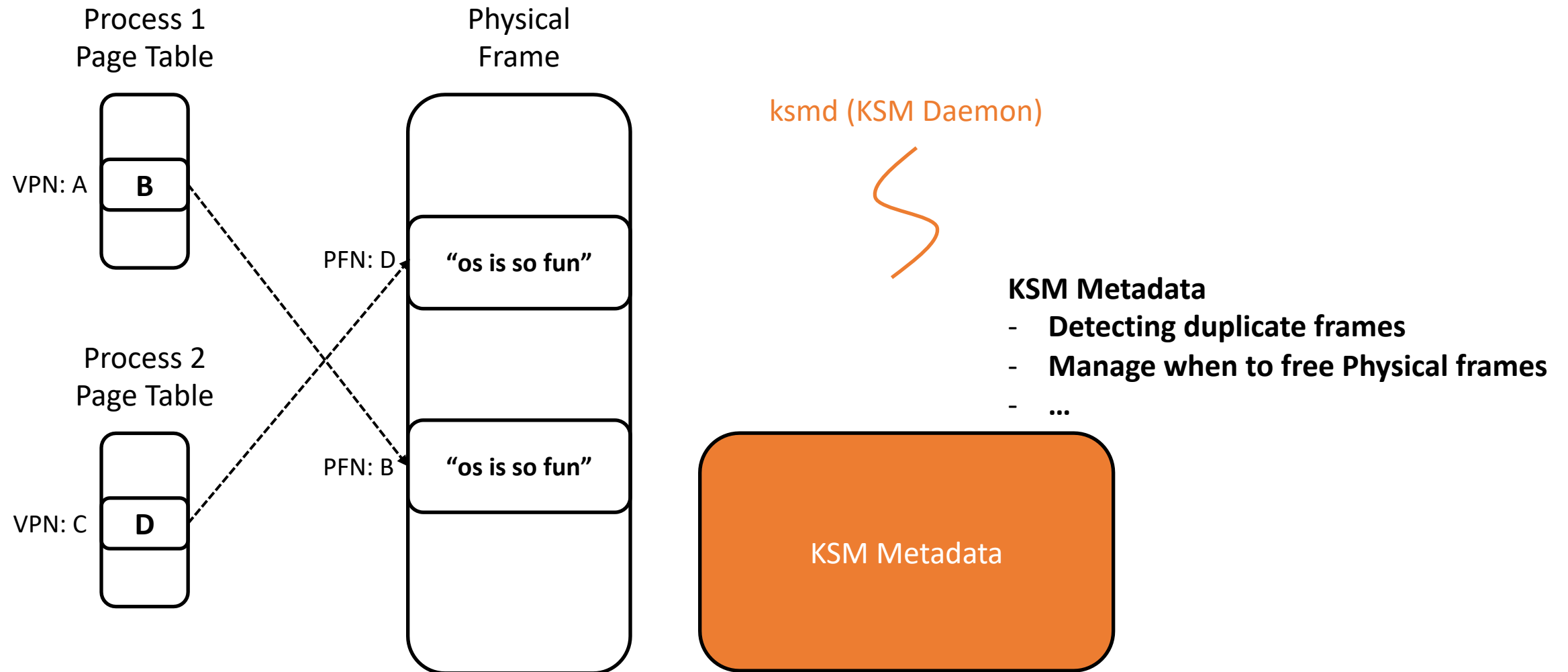
- KSM (Kernel Same page Merging) is a memory de-duplication feature that enables the kernel to **consolidate identical memory pages** into a single shared page across multiple processes.
- Merged pages are write-protected, therefore attempts to modify the merged page would cause a page fault.
- A modification to a merged page results in a **copy-on-write** action, thus preserving the integrity of the original shared page.

KSM (Kernel Samepage Merging)

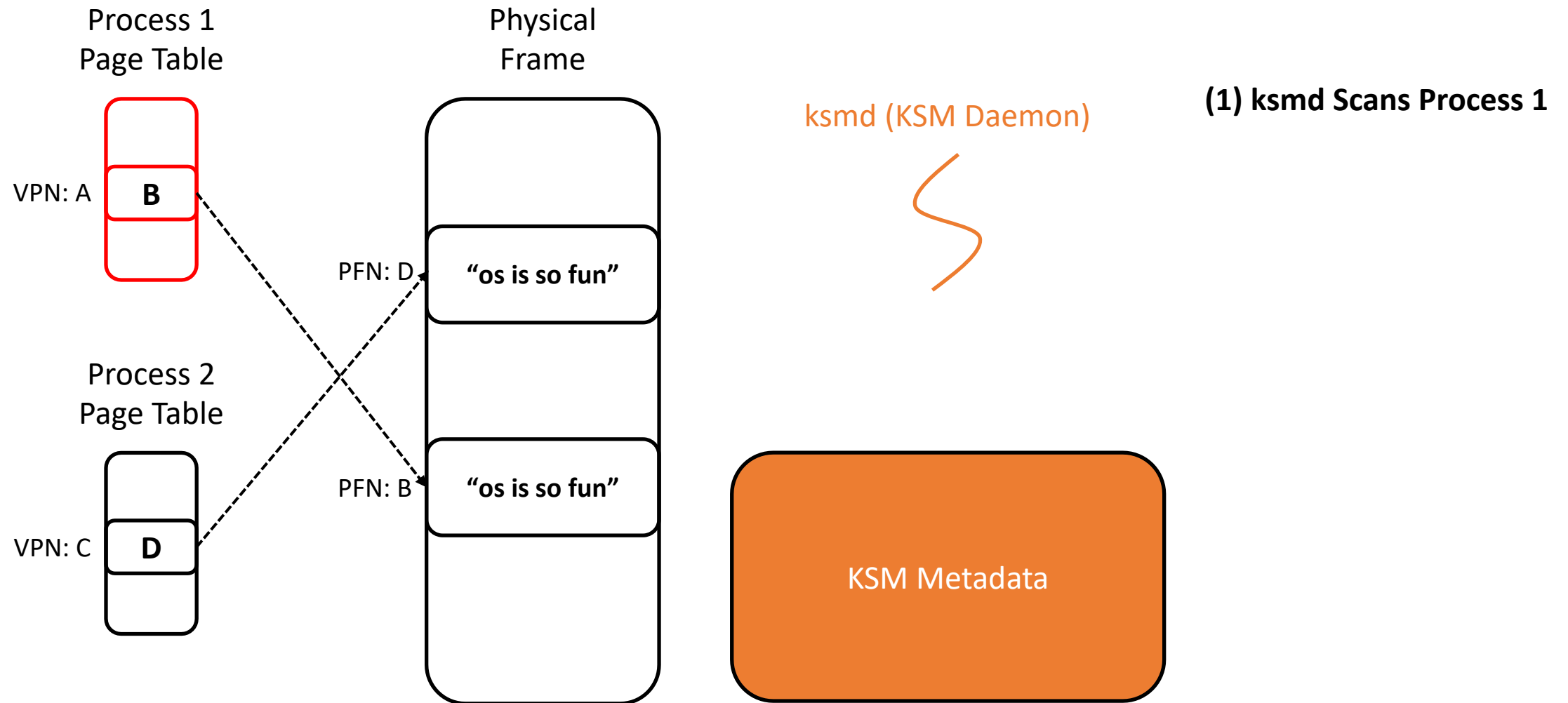


- **Process 1**
 - VPN(A) → PFN (B)
- **Process 2**
 - VPN(C) → PFN (D)
- **PFN (B) and PFN (D) has same contents**

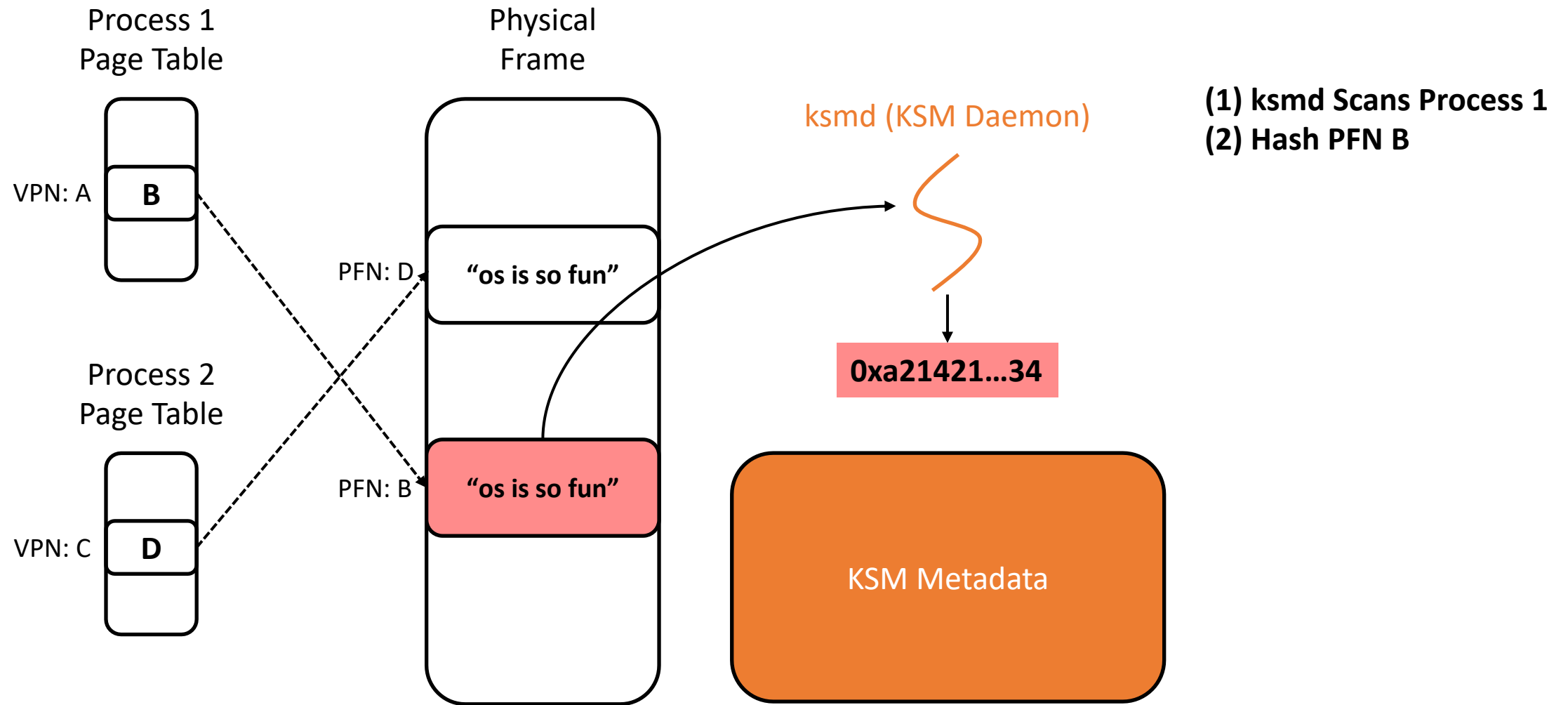
KSM (Kernel Samepage Merging)



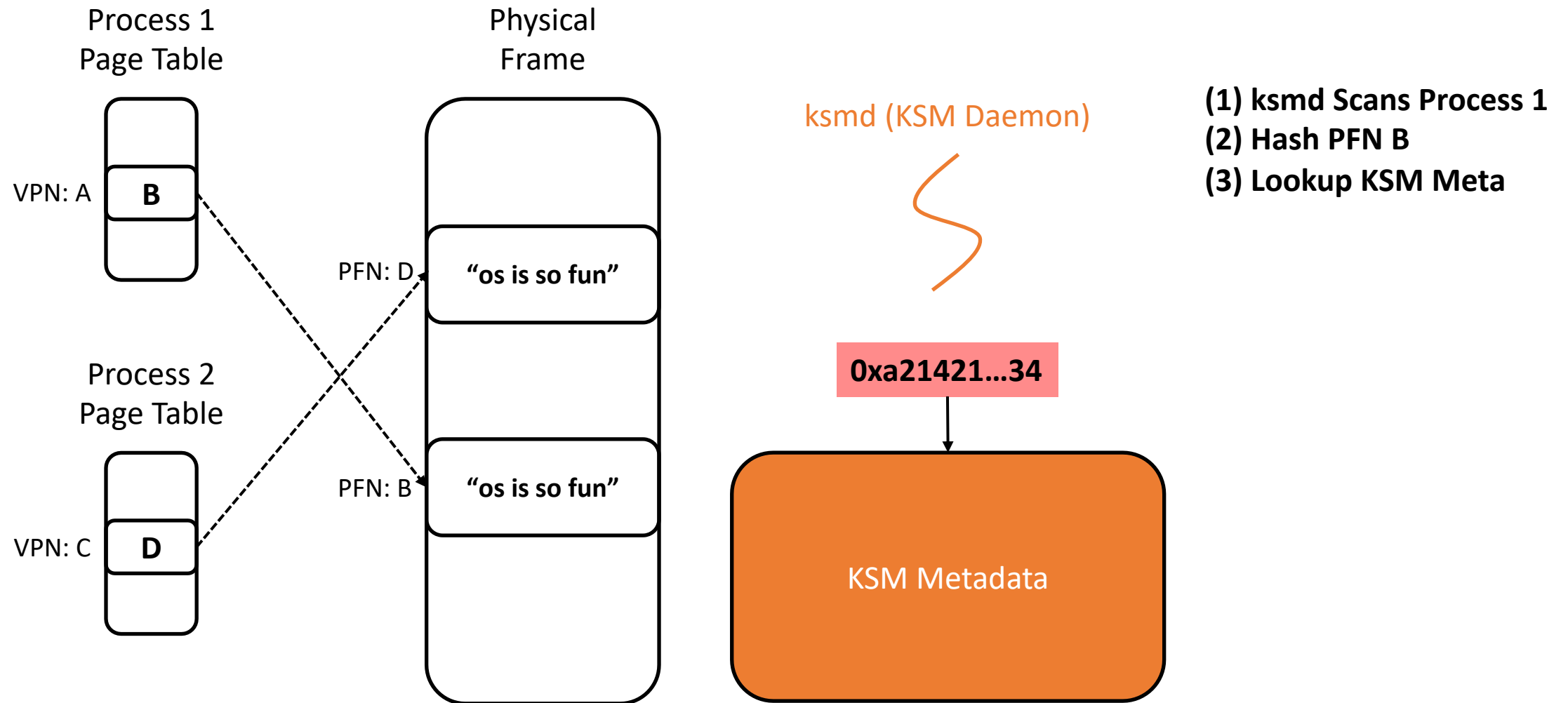
KSM (Kernel Samepage Merging)



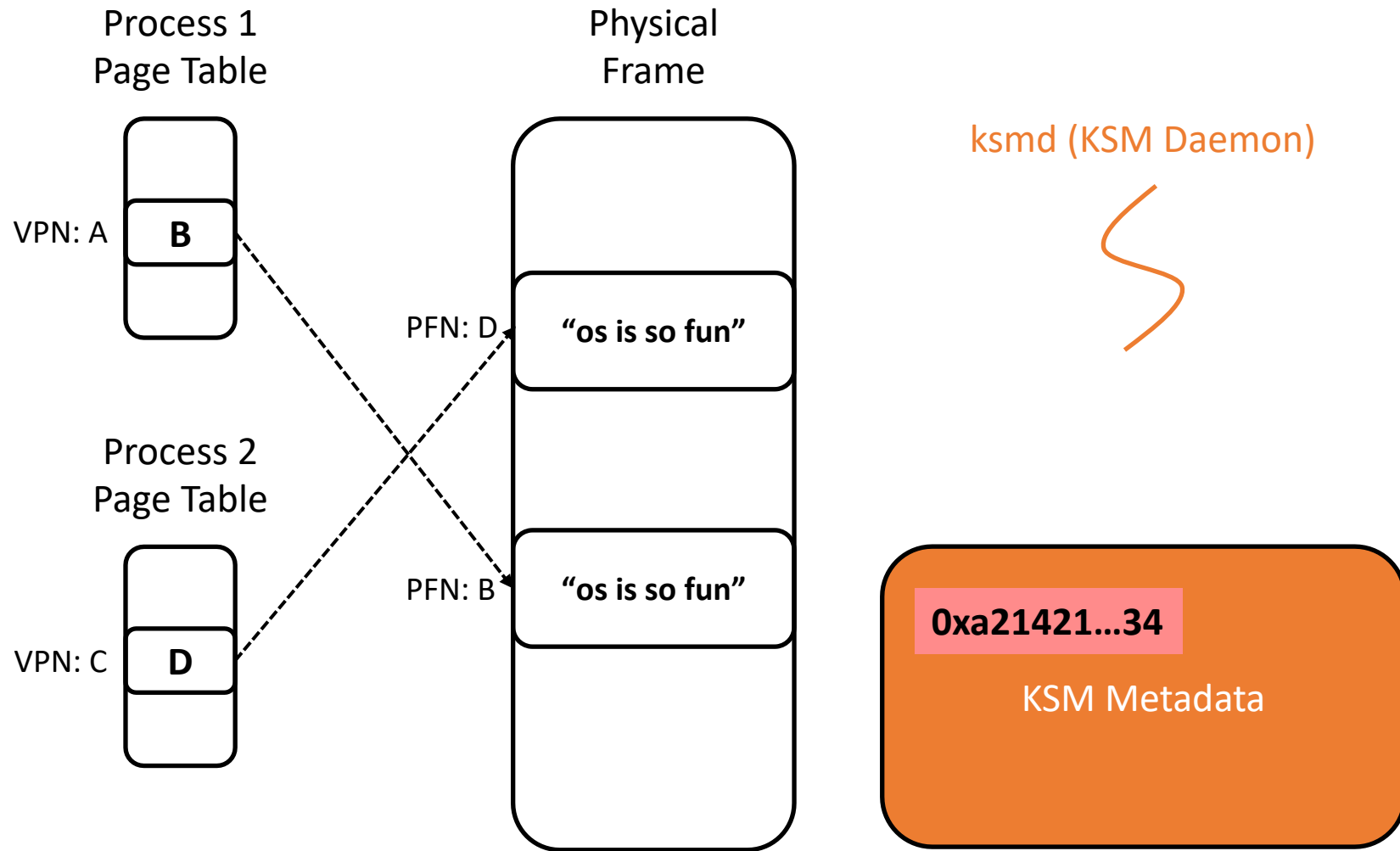
KSM (Kernel Samepage Merging)



KSM (Kernel Samepage Merging)

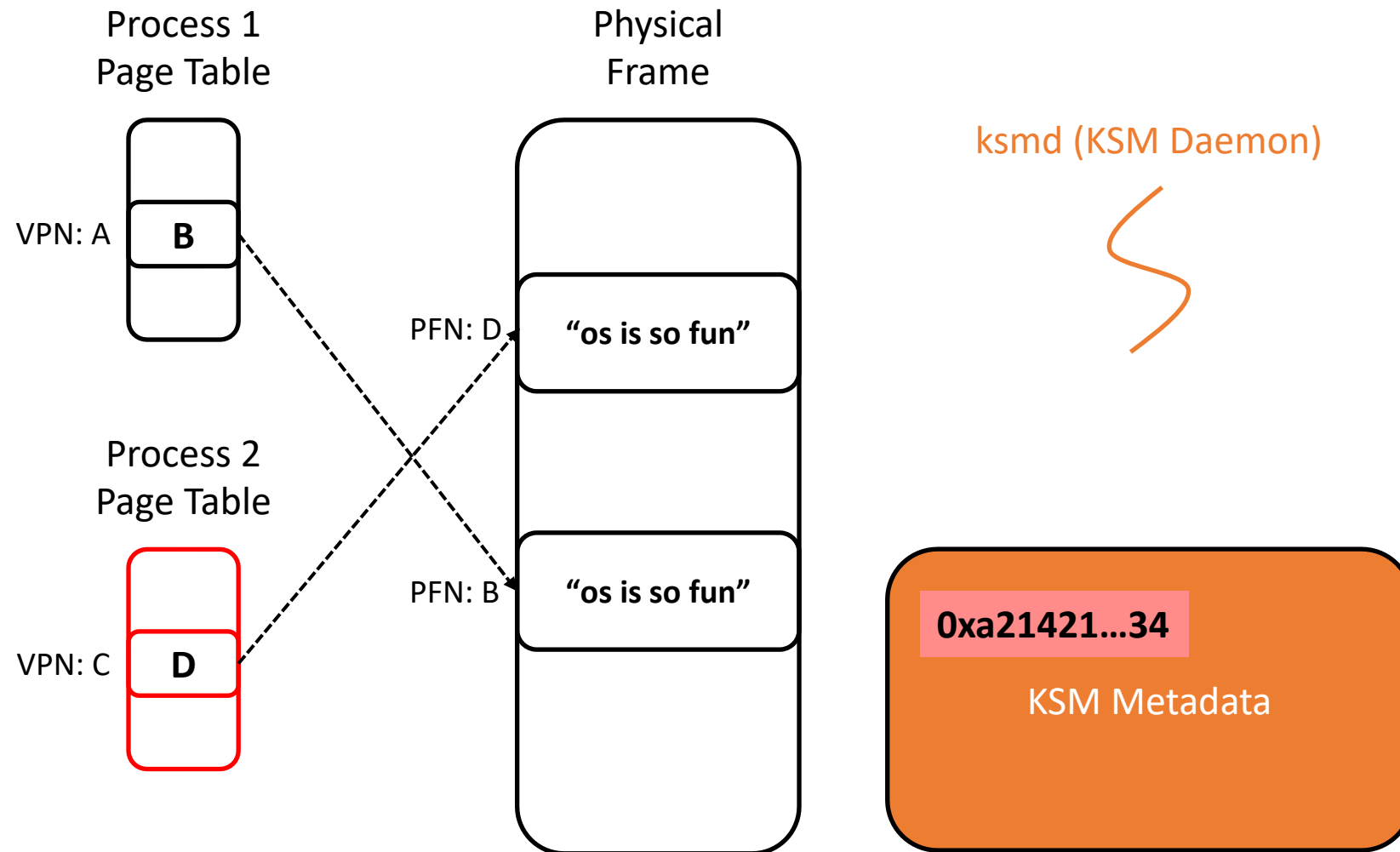


KSM (Kernel Samepage Merging)



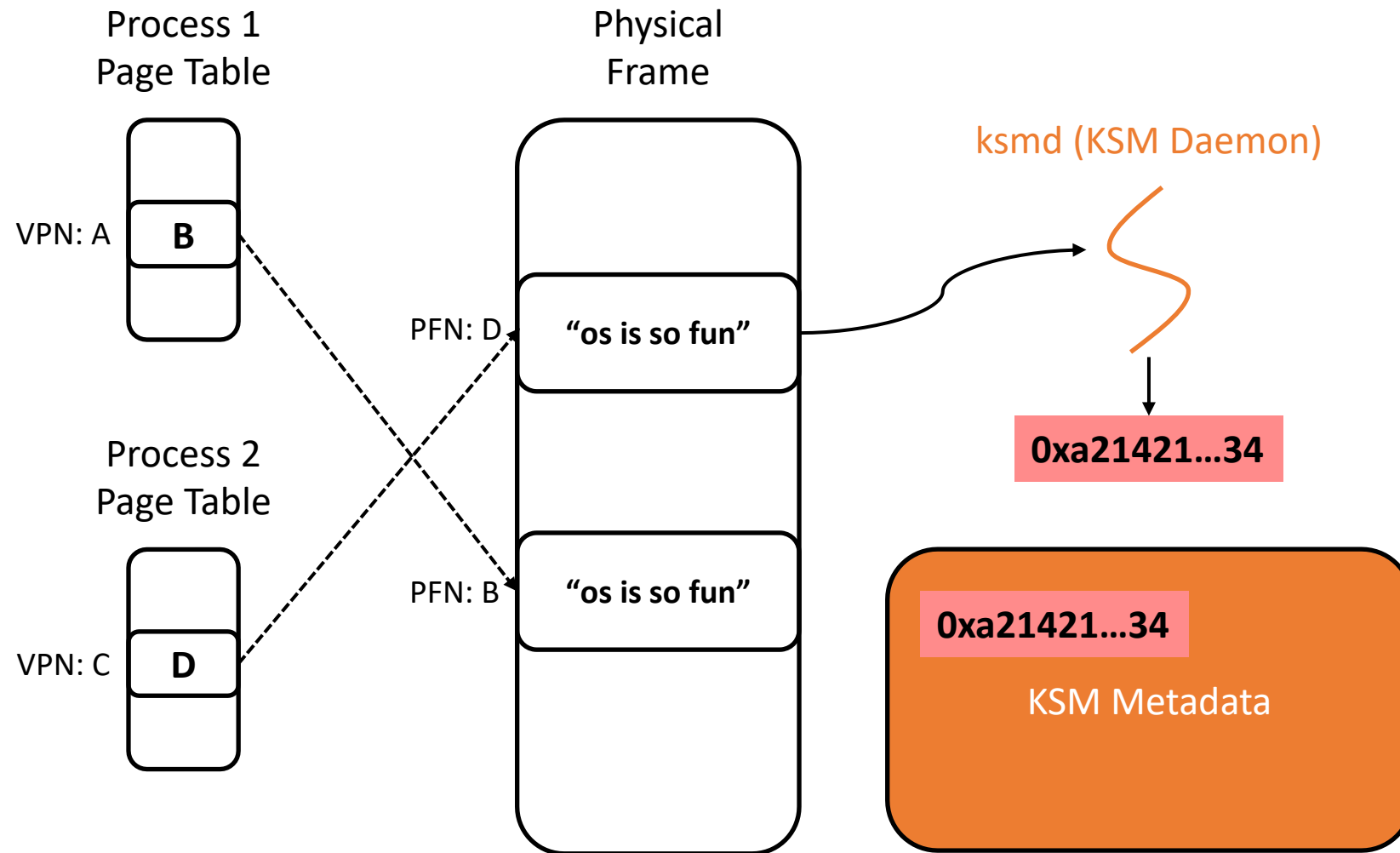
- (1) ksmd Scans Process 1
- (2) Hash PFN B
- (3) Lookup KSM Meta
- (4) Not found, update KSM Meta

KSM (Kernel Samepage Merging)



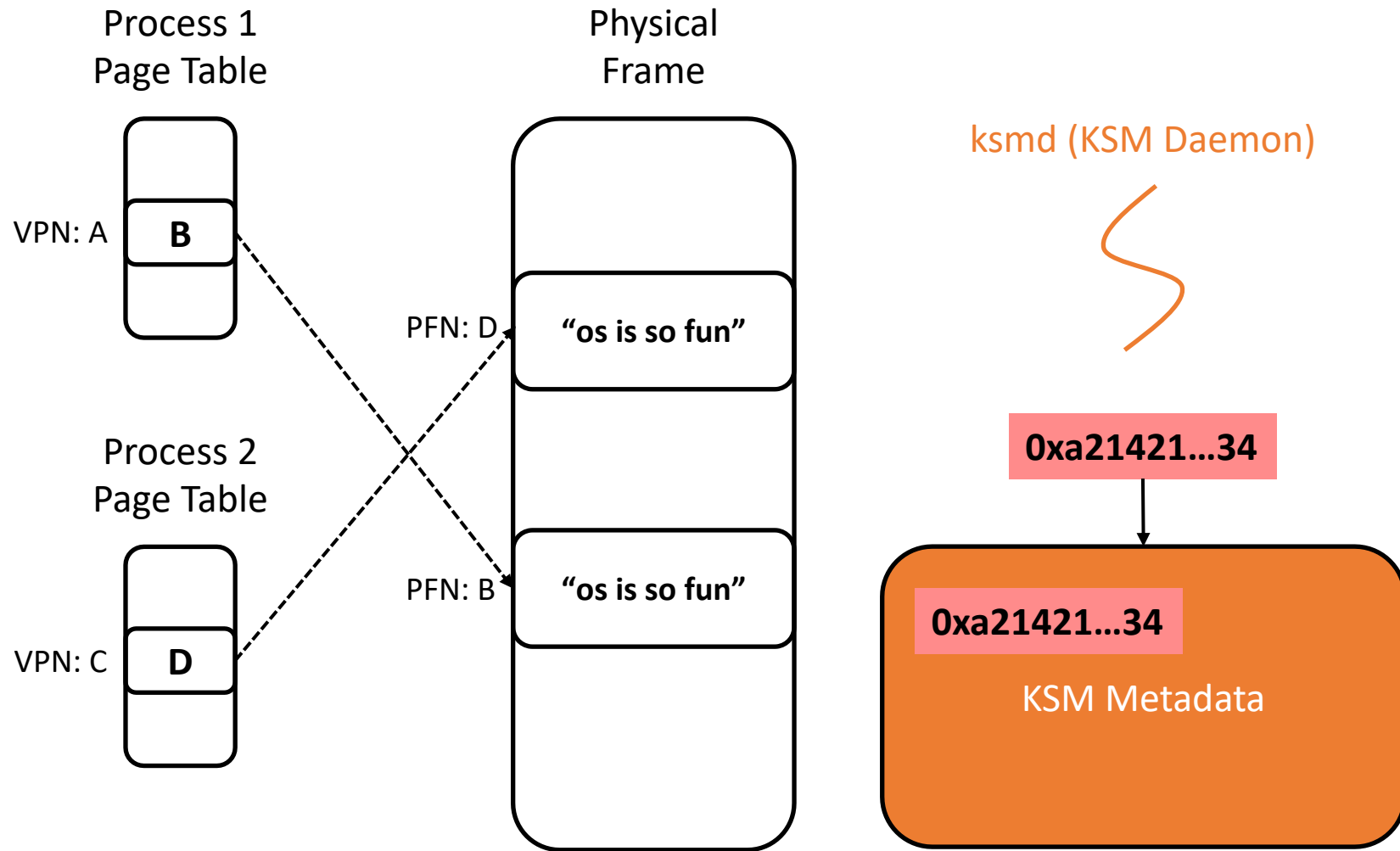
- (1) ksmd Scans Process 1
- (2) Hash PFN B
- (3) Lookup KSM Meta
- (4) Not found, update KSM Meta
- (5) ksmd Scans Process 2

KSM (Kernel Samepage Merging)



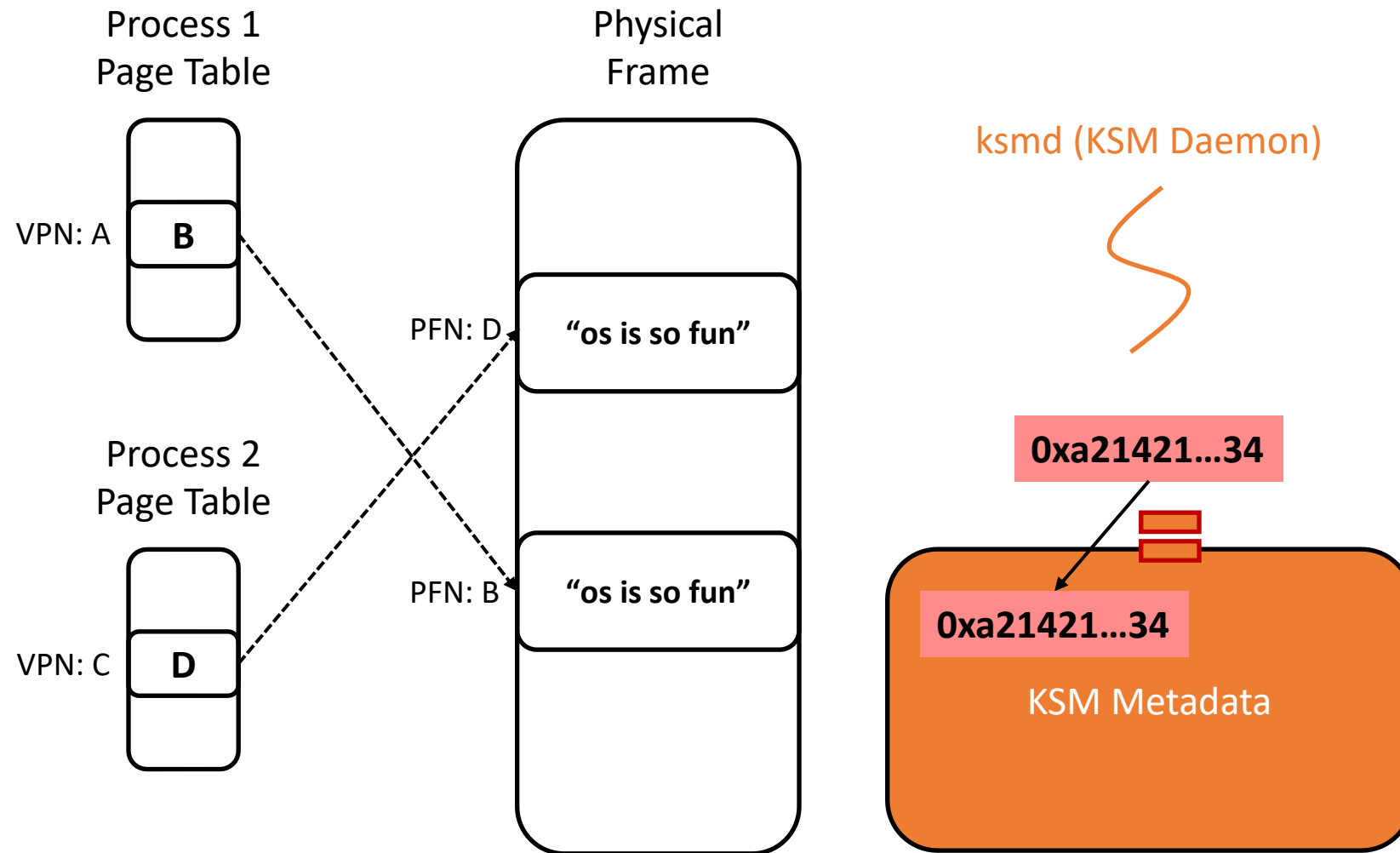
- (1) ksmd Scans Process 1
- (2) Hash PFN B
- (3) Lookup KSM Meta
- (4) Not found, update KSM Meta
- (5) ksmd Scans Process 2
- (6) Hash PFN D

KSM (Kernel Samepage Merging)



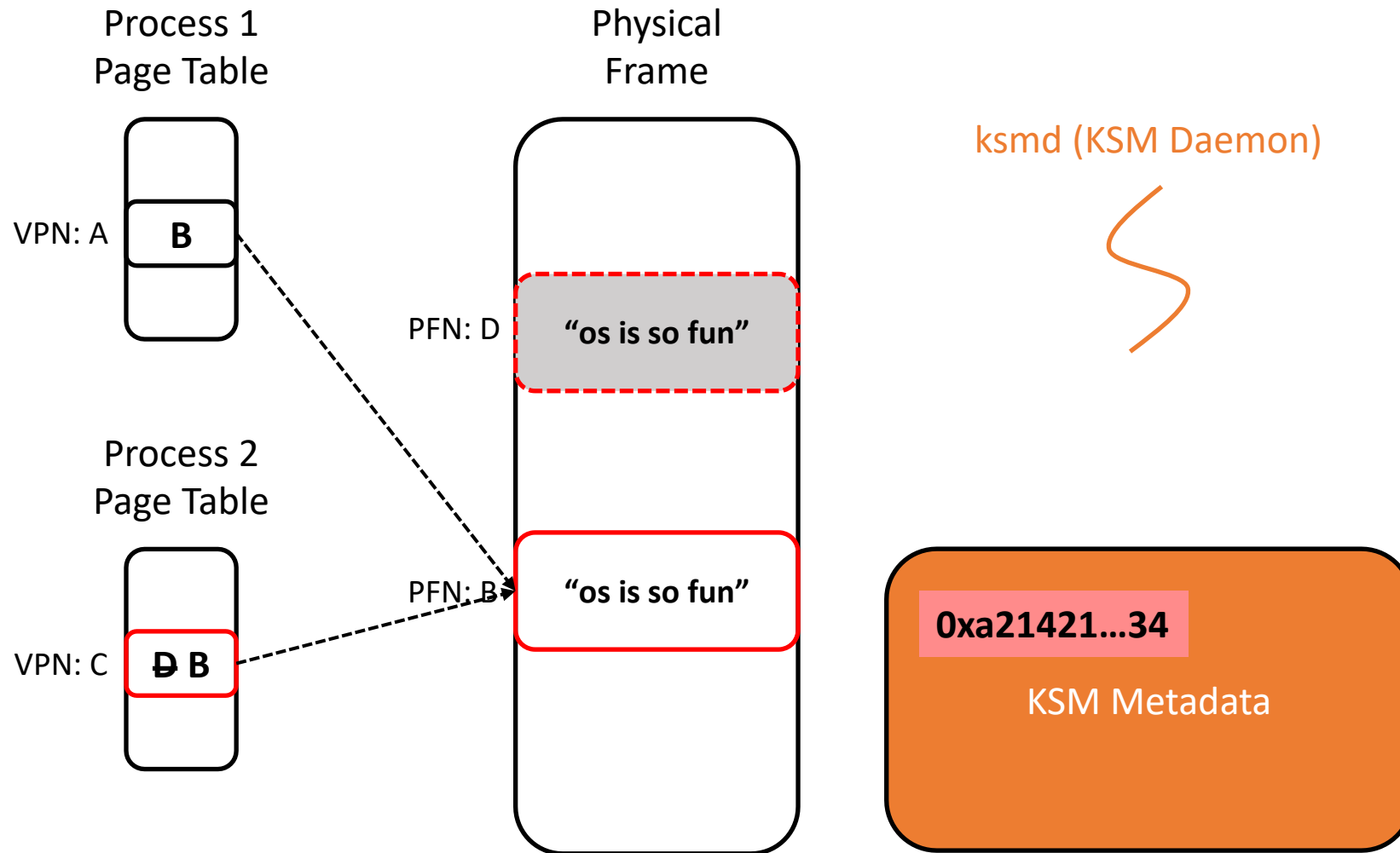
- (1) `ksmd` Scans Process 1
- (2) Hash PFN B
- (3) Lookup KSM Meta
- (4) Not found, update KSM Meta
- (5) `ksmd` Scans Process 2
- (6) Hash PFN D
- (7) Lookup KSM Meta

KSM (Kernel Samepage Merging)



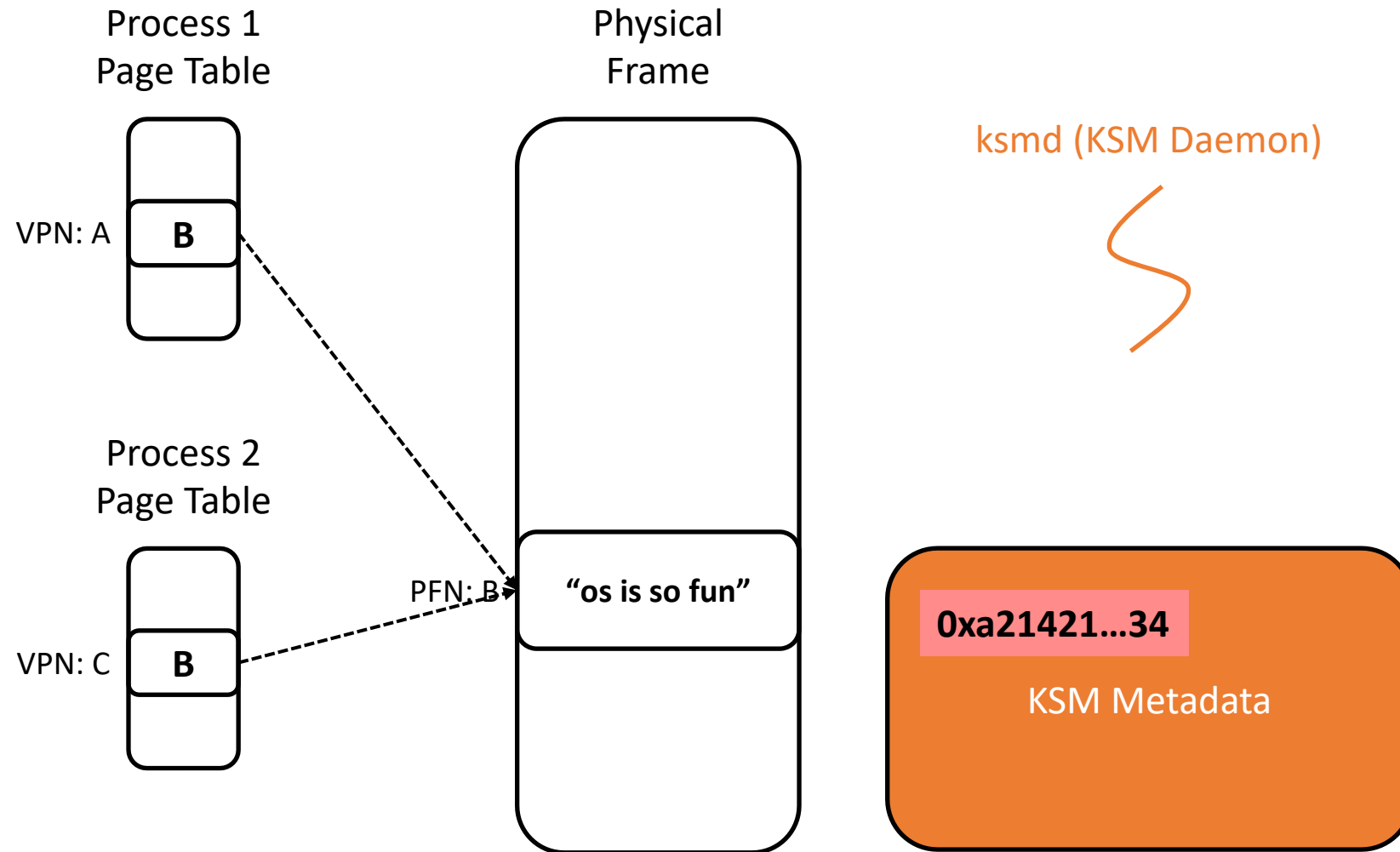
- (1) ksmd Scans Process 1
- (2) Hash PFN B
- (3) Lookup KSM Meta
- (4) Not found, update KSM Meta
- (5) ksmd Scans Process 2
- (6) Hash PFN D
- (7) Lookup KSM Meta
- (8) Found!, update KSM Meta

KSM (Kernel Samepage Merging)



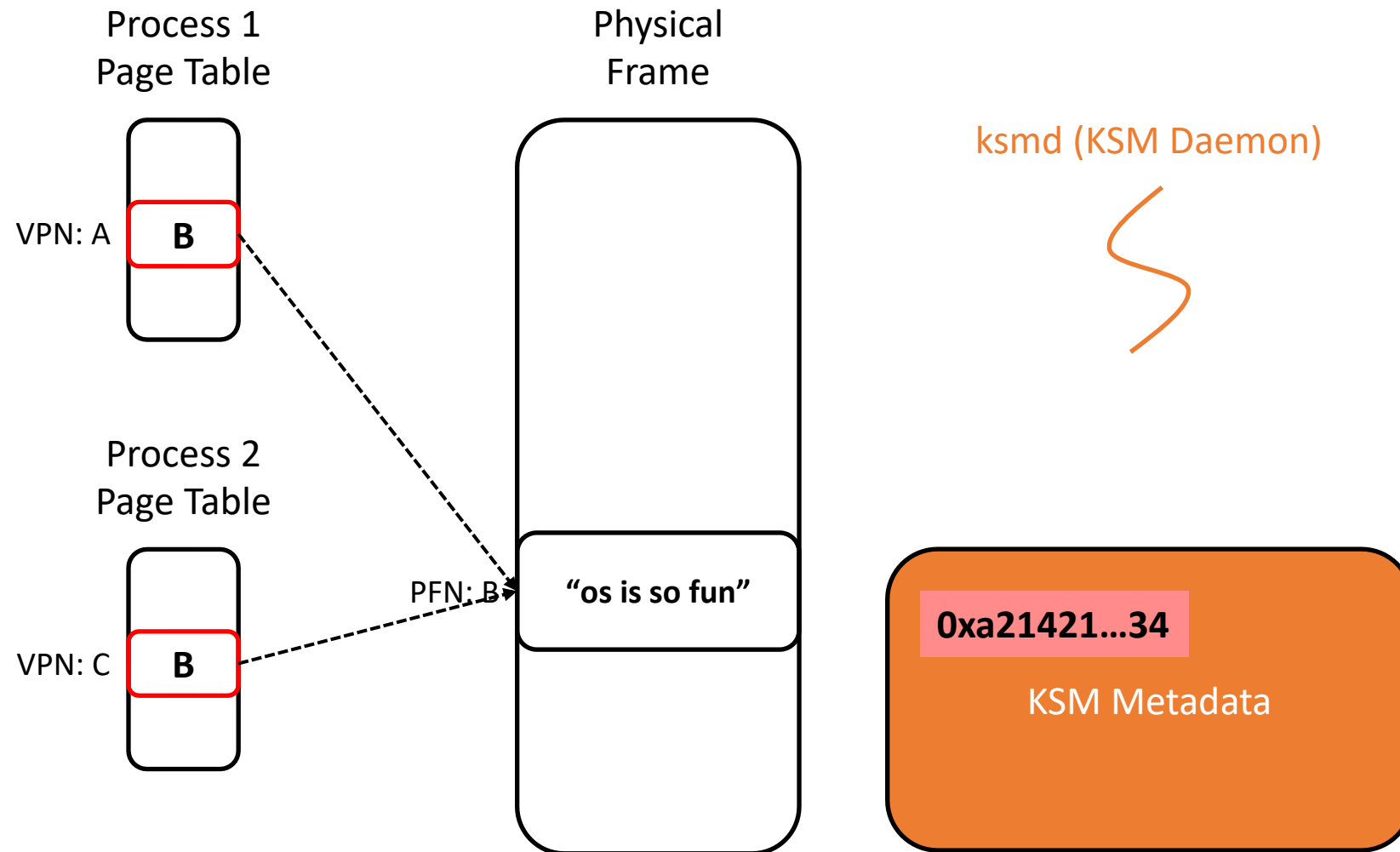
- (1) ksmd Scans Process 1
- (2) Hash PFN B
- (3) Lookup KSM Meta
- (4) Not found, update KSM Meta
- (5) ksmd Scans Process 2
- (6) Hash PFN D
- (7) Lookup KSM Meta
- (8) Found!, update KSM Meta
- (9) Merge D and B

KSM (Kernel Samepage Merging)



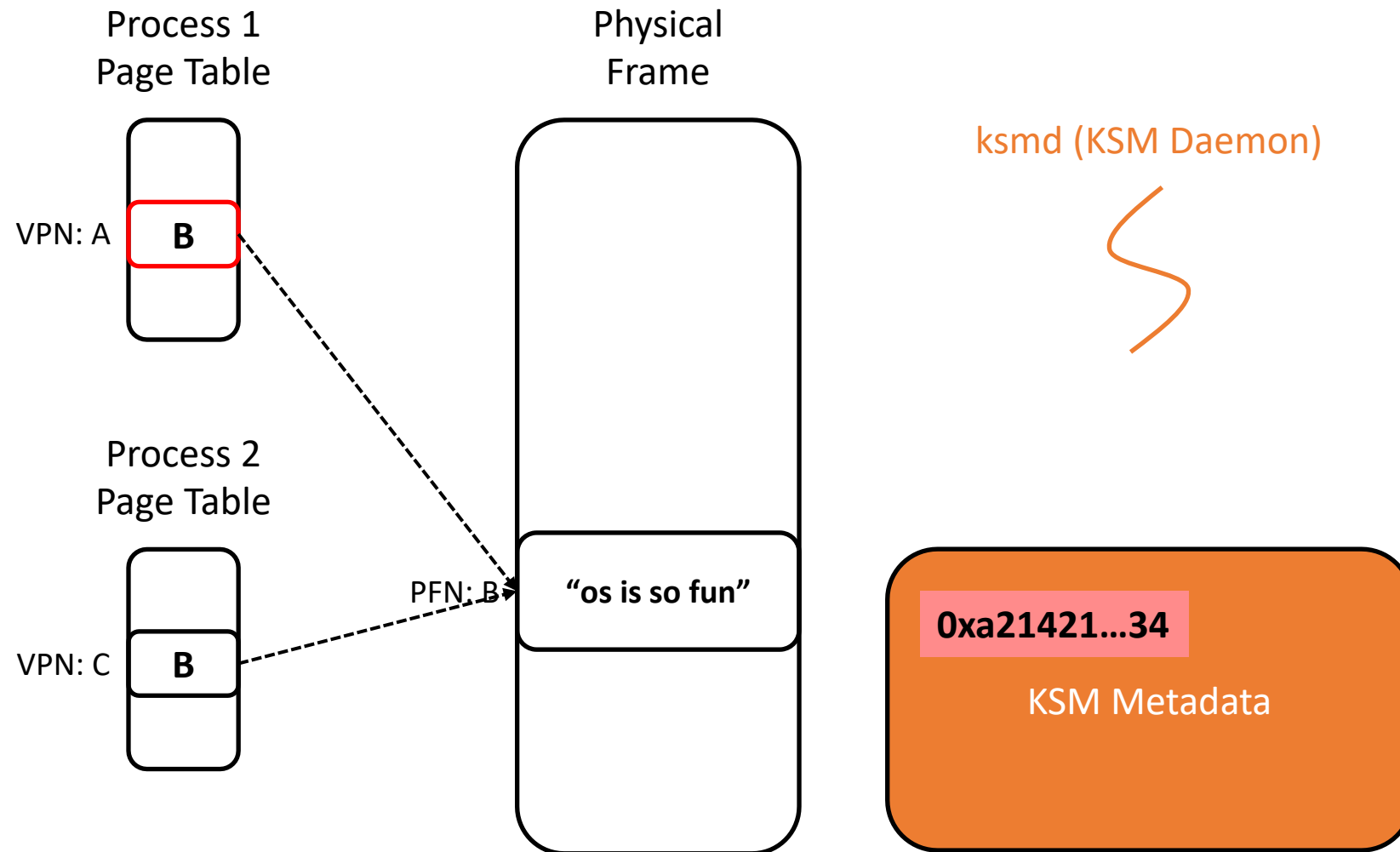
- (1) ksmd Scans Process 1
- (2) Hash PFN B
- (3) Lookup KSM Meta
- (4) Not found, update KSM Meta
- (5) ksmd Scans Process 2
- (6) Hash PFN D
- (7) Lookup KSM Meta
- (8) Found!, update KSM Meta
- (9) Merge D and B
- (10) Reclaim Frame D

KSM (Kernel Samepage Merging)



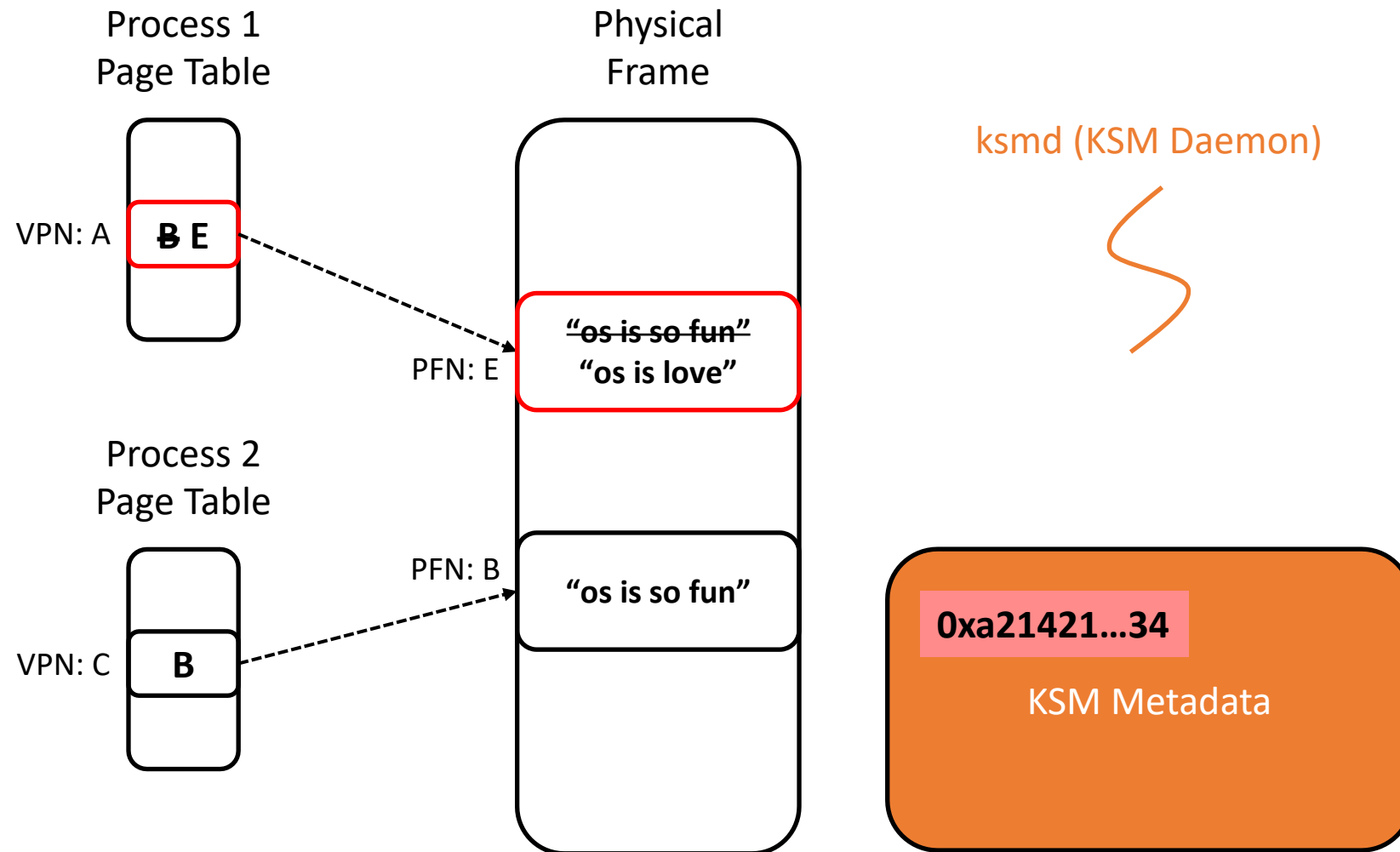
- (1) ksmd Scans Process 1
- (2) Hash PFN B
- (3) Lookup KSM Meta
- (4) Not found, update KSM Meta
- (5) ksmd Scans Process 2
- (6) Hash PFN D
- (7) Lookup KSM Meta
- (8) Found!, update KSM Meta
- (9) Merge D and B
- (10) Reclaim Frame D
- (11) Clear Write Bit of PTEs

KSM (Kernel Samepage Merging)



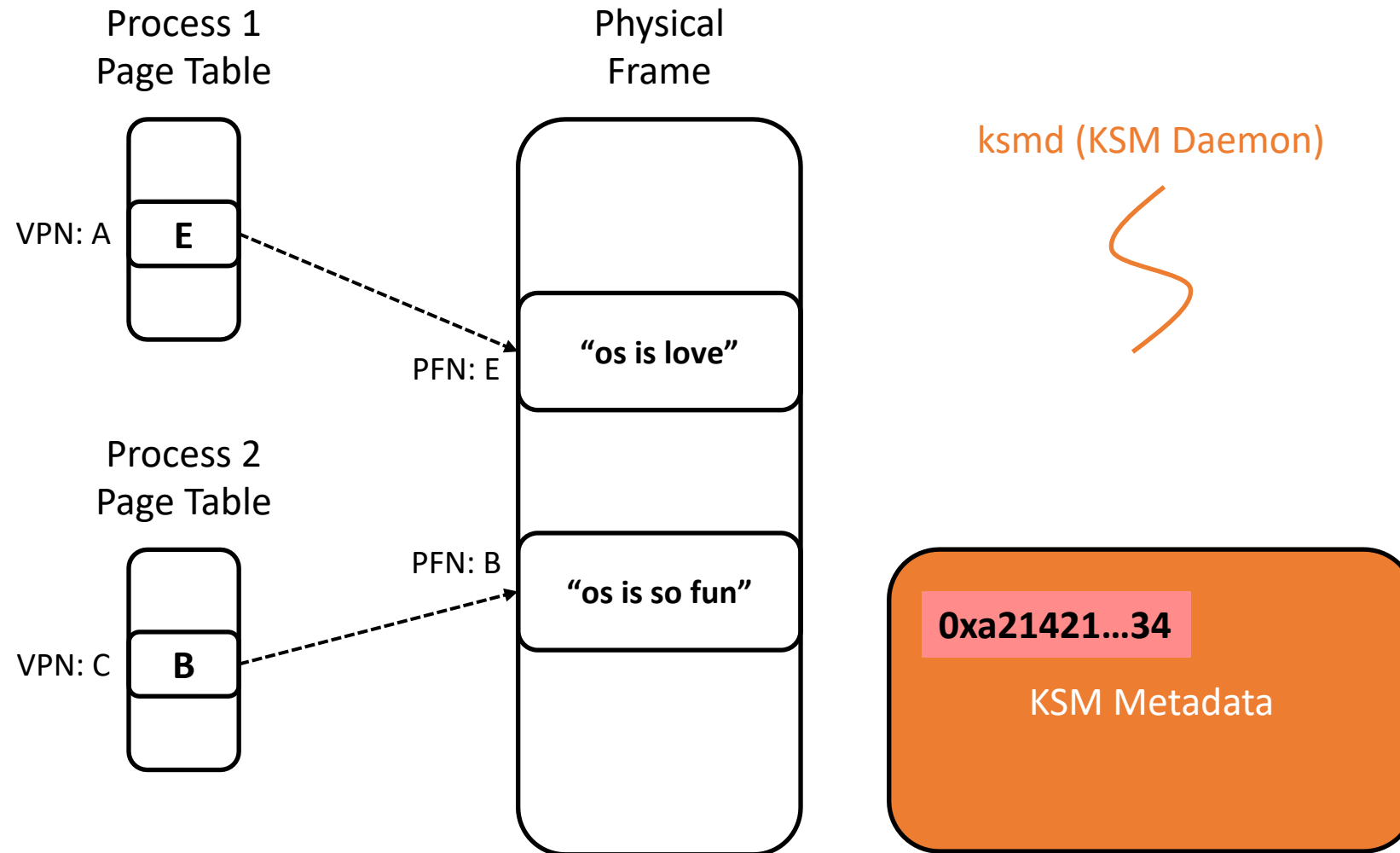
- (1) ksmd Scans Process 1
- (2) Hash PFN B
- (3) Lookup KSM Meta
- (4) Not found, update KSM Meta
- (5) ksmd Scans Process 2
- (6) Hash PFN D
- (7) Lookup KSM Meta
- (8) Found!, update KSM Meta
- (9) Merge D and B
- (10) Reclaim Frame D
- (11) Clear Write Bit of PTEs
- (12) Process 1 writes to VPN: A

KSM (Kernel Samepage Merging)



- (1) ksmd Scans Process 1
- (2) Hash PFN B
- (3) Lookup KSM Meta
- (4) Not found, update KSM Meta
- (5) ksmd Scans Process 2
- (6) Hash PFN D
- (7) Lookup KSM Meta
- (8) Found!, update KSM Meta
- (9) Merge D and B
- (10) Reclaim Frame D
- (11) Clear Write Bit of PTEs
- (12) Process 1 writes to VPN: A
- (13) Page Fault + Copy on Write

KSM (Kernel Samepage Merging)



- (1) ksmd Scans Process 1
- (2) Hash PFN B
- (3) Lookup KSM Meta
- (4) Not found, update KSM Meta
- (5) ksmd Scans Process 2
- (6) Hash PFN D
- (7) Lookup KSM Meta
- (8) Found!, update KSM Meta
- (9) Merge D and B
- (10) Reclaim Frame D
- (11) Clear Write Bit of PTEs
- (12) Process 1 writes to VPN: A
- (13) Page Fault + Copy on Write
- (14) Update KSM Metadata

I. Implement the *ksm()* (70 points)

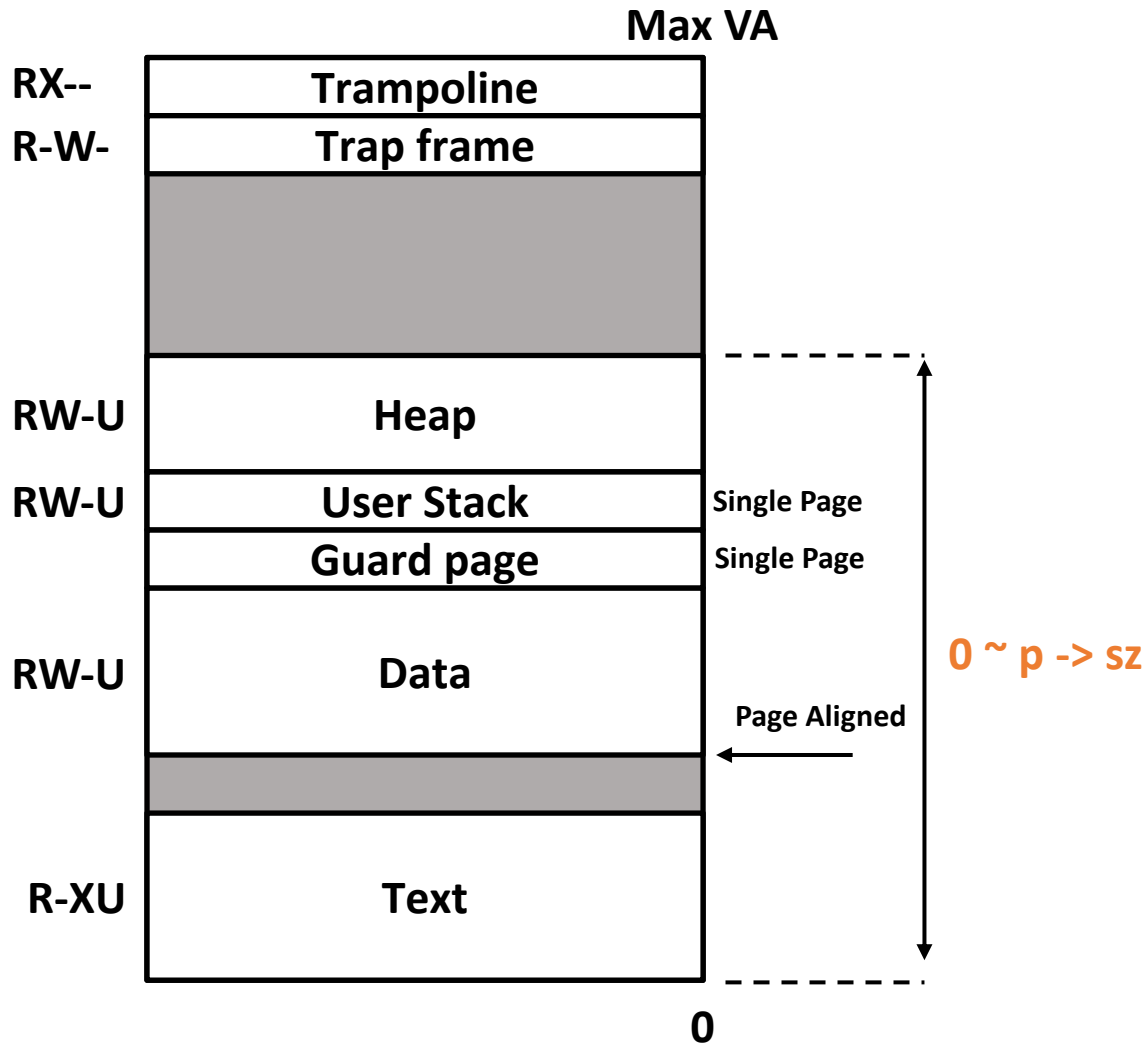
- Your task is to implement a new system call named *ksm()*
- ```
int ksm(int *scanned, int *merged);
```

  - scanned: Total number of scanned frames
  - merged: Total number of merged frames
  - return value: Total number of free frames ('freemem')
- System call number is assigned to 24 (already done in branch pa4)
- The initial free memory could differ due to implementation details.

# I. Implement the *ksm()* (70 points)

- Scan the page frames used by **each user process p** from the virtual address **0 to p->sz**. The region will contain code, data, heap, and stack (+ stack guard) pages. Exclude trampoline, trapframe, kernel stack, and page table pages from scanning as they are not targets for KSM.
- You can assume that a single physical frame can be shared by up to **16 different virtual pages**, except for the **zero-page**. There is no limit on the number of virtual pages that can share the zero-page.
- When performing *ksm()*, exclude the page frames used by **init** process (pid 1), the **sh** process (pid 2), and the process invoking the *ksm()* system call itself from scanning. Assume that the shell process always runs with pid 2.
- Duplicated page frames are **merged only through the *ksm()* system call**; they should not be merged at the time of page allocation.
- There should be no memory leak. The `freemem` value should remain identical before and after executing a program.
- `ksm3` user program should run with background `ksmd` running. To run a process background, run `'ksmd &'`

# User Virtual Memory



```
// Per-process state
struct proc {
 struct spinlock lock;

 // p->lock must be held when using these:
 enum procstate state; // Process state
 void *chan; // If non-zero, sleeping on chan
 int killed; // If non-zero, have been killed
 int xstate; // Exit status to be returned to parent's wait
 int pid; // Process ID

 // wait_lock must be held when using this:
 struct proc *parent; // Parent process

 // these are private to the process, so p->lock need not be held.
 uint64 kstack; // Virtual address of kernel stack
 uint64 sz; // Size of process memory (bytes)
 pagetable_t pagetable; // User page table
 struct trapframe *trapframe; // data page for trampoline.S
 struct context context; // swtch() here to run process
 struct file *ofile[NOFILE]; // Open files
 struct inode *cwd; // Current directory
 char name[16]; // Process name (debugging)
};
```

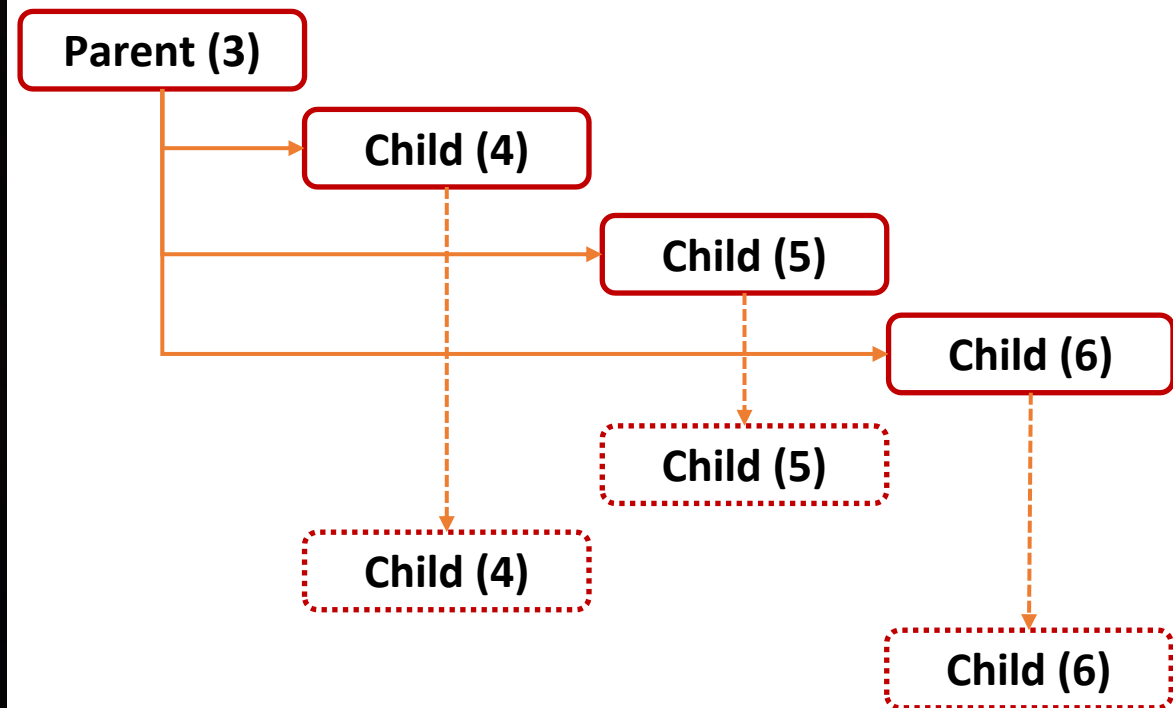
# Zero Page

- You are required to preallocate a "zero-page" in the system which is filled with zeroes. Since BSS and heap pages are initialized to zero, those pages can be mapped to the zero-page.
- There is **no limit** on the number of virtual pages that can share the zero-page.

# User program example

- ksm l

```
$ ksm1
pid 3: ksmd starts
ksm: scanned=0, merged=0, freemem=31458
KSMD: after forking child 4
ksm: scanned=7, merged=5, freemem=31450
KSMD: after forking child 5
ksm: scanned=14, merged=6, freemem=31443
KSMD: after forking child 6
ksm: scanned=21, merged=6, freemem=31436
KSMD: after terminating child 5
ksm: scanned=14, merged=0, freemem=31443
KSMD: after terminating child 4
ksm: scanned=7, merged=0, freemem=31450
KSMD: after terminating child 6
ksm: scanned=0, merged=0, freemem=31458
```



\*The scanned & freemem value could differ due to implementation detail

# User program example

- ksm I (with debug message)

```
pid 3: ksmd starts
ksm: scanned=0, merged=0, freemem=31456
KSMd: after forking child 4
[KSM] ===== Scanning Process[4] =====
[KSM] Scanning VPN(0): Unique Page (PA: 0x0000000087f49000)
[KSM] Scanning VPN(1): Mapped to zero page
[KSM] Scanning VPN(2): Mapped to zero page
[KSM] Scanning VPN(3): Mapped to zero page
[KSM] Scanning VPN(4): Mapped to zero page
[KSM] Scanning VPN(5): Mapped to zero page
[KSM] Scanning VPN(6): Unique Page (PA: 0x0000000087f51000)
ksm: scanned=7, merged=5, freemem=31448
KSMd: after forking child 5
[KSM] ===== Scanning Process[4] =====
[KSM] Scanning VPN(0):
[KSM] Scanning VPN(1):
[KSM] Scanning VPN(2):
[KSM] Scanning VPN(3):
[KSM] Scanning VPN(4):
[KSM] Scanning VPN(5):
[KSM] Scanning VPN(6):
[KSM] ===== Scanning Process[5] =====
[KSM] Scanning VPN(0): Hash match, (old pa 0x0000000087f4c000 -> new pa 0x0000000087f49000) (RC -> 2)
[KSM] Scanning VPN(1): Mapped to zero page
[KSM] Scanning VPN(2): Mapped to zero page
[KSM] Scanning VPN(3): Mapped to zero page
[KSM] Scanning VPN(4): Mapped to zero page
[KSM] Scanning VPN(5): Mapped to zero page
[KSM] Scanning VPN(6): Unique Page (PA: 0x0000000087f6e000)
ksm: scanned=14, merged=6, freemem=31441
```

Text region

Stack region



# User program example

- ksm I (with debug message)

```
pid 3: ksmd starts
ksm: scanned=0, merged=0, freemem=31456
KSMd: after forking child 4
[KSM] ===== Scanning Process[4] =====
[KSM] Scanning VPN(0): Unique Page (PA: 0x0000000087f49000)
[KSM] Scanning VPN(1): Mapped to zero page
[KSM] Scanning VPN(2): Mapped to zero page
[KSM] Scanning VPN(3): Mapped to zero page
[KSM] Scanning VPN(4): Mapped to zero page
[KSM] Scanning VPN(5): Mapped to zero page
[KSM] Scanning VPN(6): Unique Page (PA: 0x0000000087f51000)
ksm: scanned=7, merged=5, freemem=31448
KSMd: after forking child 5
[KSM] ===== Scanning Process[4] =====
[KSM] Scanning VPN(0):
[KSM] Scanning VPN(1):
[KSM] Scanning VPN(2):
[KSM] Scanning VPN(3):
[KSM] Scanning VPN(4):
[KSM] Scanning VPN(5):
[KSM] Scanning VPN(6):
[KSM] ===== Scanning Process[5] =====
[KSM] Scanning VPN(0): Hash match, (old pa 0x0000000087f4c000 -> new pa 0x0000000087f49000) (RC -> 2)
[KSM] Scanning VPN(1): Mapped to zero page
[KSM] Scanning VPN(2): Mapped to zero page
[KSM] Scanning VPN(3): Mapped to zero page
[KSM] Scanning VPN(4): Mapped to zero page
[KSM] Scanning VPN(5): Mapped to zero page
[KSM] Scanning VPN(6): Unique Page (PA: 0x0000000087f6e000)
ksm: scanned=14, merged=6, freemem=31441
```

**Does nothing because it has been merged or scanned**

# User program example

- ksm I (with debug message)

```
pid 3: ksmd starts
ksm: scanned=0, merged=0, freemem=31456
KSMd: after forking child 4
[KSM] ===== Scanning Process[4] =====
[KSM] Scanning VPN(0): Unique Page (PA: 0x0000000087f49000)
[KSM] Scanning VPN(1): Mapped to zero page
[KSM] Scanning VPN(2): Mapped to zero page
[KSM] Scanning VPN(3): Mapped to zero page
[KSM] Scanning VPN(4): Mapped to zero page
[KSM] Scanning VPN(5): Mapped to zero page
[KSM] Scanning VPN(6): Unique Page (PA: 0x0000000087f51000)
ksm: scanned=7, merged=5, freemem=31448
KSMd: after forking child 5
[KSM] ===== Scanning Process[4] =====
[KSM] Scanning VPN(0):
[KSM] Scanning VPN(1):
[KSM] Scanning VPN(2):
[KSM] Scanning VPN(3):
[KSM] Scanning VPN(4):
[KSM] Scanning VPN(5):
[KSM] Scanning VPN(6):
[KSM] ===== Scanning Process[5] =====
[KSM] Scanning VPN(0): Hash match, (old pa 0x0000000087f4c000 -> new pa 0x0000000087f49000) (RC -> 2)
[KSM] Scanning VPN(1): Mapped to zero page
[KSM] Scanning VPN(2): Mapped to zero page
[KSM] Scanning VPN(3): Mapped to zero page
[KSM] Scanning VPN(4): Mapped to zero page
[KSM] Scanning VPN(5): Mapped to zero page
[KSM] Scanning VPN(6): Unique Page (PA: 0x0000000087f6e000)
ksm: scanned=14, merged=6, freemem=31441
```

After fork, child 5 and 4 has the same text area

Stack may differ

# Hints

- There are three major flows that you should modify
  - KSM flow
  - Page fault flow (copy-on-write)
  - Physical page release flow (process exit, exec, ...)
- Try to print as much debug message as possible
  - PFN
  - VPN
  - Which page is merged to which
  - Which page was released
  - Which page performed a CoW

## 2. Design Document (30 points)

- There are limitations in simply comparing the output of the user programs to genuinely check if the KSM was properly implemented.
- You should explain what you have considered, and what you have done.
- Requirements
  - Data structures
  - Overall flowchart
  - Algorithm design
  - Implementation details
  - Testing and validation

# Tips

- Read Chap. 3, Chap. 4 of the [xv6 book](#) to understand RISC-V's virtual memory subsystem and page-fault exceptions in xv6.
- For your reference, the following roughly shows the amount of changes you need to make for this project assignment.
- Each “+” symbol indicates 1~10 lines of code that should be added, deleted, or altered.

|               |  |       |
|---------------|--|-------|
| kernel/defs.h |  | +     |
| kernel/exec.c |  | +     |
| kernel/proc.c |  | +     |
| kernel/trap.c |  | +     |
| kernel/vm.c   |  | +     |
| kernel/ksm.h  |  | +++++ |
| kernel/ksm.c  |  | +++++ |

# Restrictions

- For this project assignment, you can assume a uniprocessor RISC-V system (**CPUS = 1**) with a physical memory size of **128 MiB**.
- Please use the qemu version 8.2.0 or later. To determine the qemu version, use the command: `$ qemu-system-riscv64 --version`
- We will run `qemu-system-riscv64` with the `-icount shift=0` option, which enables aligning the host and virtual clocks. This setting is already included in the Makefile for the `pa4` branch.
- You only need to change the files in the `./kernel` directory (mostly to `ksm.h` and `ksm.c` files provided in the skeleton code). Any other changes outside the `./kernel` directory will be ignored during grading.

# Skeleton Code

## ■ Skeleton Code

- You should work on the pa4 branch of the xv6-riscv-snu repository as follows:

```
$ git clone https://github.com/snu-csl/xv6-riscv-snu
$ git checkout pa4
```

- The pa4 branch has a user-level utility program named **ksmd**, **ksm1**, **ksm2**, **ksm3** which can be built from the **user/ksmd.c**, **user/kms1.c**, **user/ksm2.c**, **user/ksm3.c** file

# Notification

## ■ Due

- 11:59 PM, May 26 (Sunday)

## ■ Submission

- Run the `make submit` command to generate a tarball named `xv6-pa4- $\{STUDENTID\}$ .tar.gz` in the `xv6-riscv-snu` directory
- Upload the compressed file to the submission server
- The total number of submissions for this project will be limited to **50**
- Only the version marked `FINAL` will be considered for the project score



# Using GDB with QEMU

# GDB with QEMU

- In the xv6-riscv-snu directory,
- Run `make qemu-gdb` to run QEMU
- In another shell, run `gdb-multiarch ./kernel/kernel`
- `gdb-multiarch` automatically sets the target architecture to “riscv:rv64”

```
telomere37 — ssh tony_snu — tony_snu — ssh tony_snu — zsh — ttys000
[tony@snu-tony: ~/pa4-skel(pa4)]$ ls
fs.img kernel LICENSE Makefile mkfs README user
[tony@snu-tony: ~/pa4-skel(pa4)]$ make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 1 -nographic -icount shift=0 -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 -S -gdb tcp::26000
```

```
telomere37 — ssh tony_snu — tony_snu — ssh tony_snu — zsh — ttys000
[tony@snu-tony: ~/pa4-skel/kernel(pa4)]$ gdb-multiarch kernel
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
 <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from kernel...
(gdb)
```

# GDB with QEMU

- In GDB, enter target remote :<port>
- You can find TCP port in the QEMU log

```
csl@sys.snu.ac.kr x + v
csl@sys ~/injae/xv6-riscv-snu % make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 4 -nog
raphic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id
=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 -S -gdb tcp::26000
```

```
csl@sys.snu.ac.kr x + v
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from kernel/kernel...
warning: File "/home/csl/injae/xv6-riscv-snu/.gdbinit" auto-loading has been declined
by your `auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
 add-auto-load-safe-path /home/csl/injae/xv6-riscv-snu/.gdbinit
line to your configuration file "/home/csl/.gdbinit".
To completely disable this security protection add
 set auto-load safe-path /
line to your configuration file "/home/csl/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual. E.g., run from the shell:
--Type <RET> for more, q to quit, c to continue without paging--
info "(gdb)Auto-loading safe path"
(gdb) target remote :26000
Remote debugging using :26000
0x00000000000001000 in ?? ()
(gdb)
```

# GDB with QEMU

- The xv6 virtual machine has stopped at 0x1000 (the very beginning of the text section)
- To continue, enter c in GDB

```
csl@sys.snu.ac.kr x + v
csl@sys ~/injae/xv6-riscv-snu % make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 4 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 -S -gdb tcp::26000

xv6 kernel is booting

hart 3 starting
hart 2 starting
hart 1 starting
init: starting sh
$ █
```

(Running)

```
csl@sys.snu.ac.kr x + v
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
 <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from kernel/kernel...
warning: File "/home/csl/injae/xv6-riscv-snu/.gdbinit" auto-loading has been declined
by your `auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
 add-auto-load-safe-path /home/csl/injae/xv6-riscv-snu/.gdbinit
line to your configuration file "/home/csl/.gdbinit".
To completely disable this security protection add
 set auto-load safe-path /
line to your configuration file "/home/csl/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual. E.g., run from the shell:
--Type <RET> for more, q to quit, c to continue without paging--
info "(gdb)Auto-loading safe path"
(gdb) target remote :26000
Remote debugging using :26000
0x0000000000000100 in ?? ()
(gdb) c
Continuing.
█
```

# GDB with QEMU

- To stop again, enter Ctrl-C in GDB
- Then the xv6 virtual machine stops immediately

```
csl@sys.snu.ac.kr x + v
csl@sys ~/injae/xv6-riscv-snu % make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 4 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 -S -gdb tcp::26000

xv6 kernel is booting

hart 3 starting
hart 2 starting
hart 1 starting
init: starting sh
$ █

(Stopped)
```

```
csl@sys.snu.ac.kr x + v
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from kernel/kernel...
warning: File "/home/csl/injae/xv6-riscv-snu/.gdbinit" auto-loading has been declined by your `auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
 add-auto-load-safe-path /home/csl/injae/xv6-riscv-snu/.gdbinit
line to your configuration file "/home/csl/.gdbinit".
To completely disable this security protection add
 set auto-load safe-path /
line to your configuration file "/home/csl/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual. E.g., run from the shell:
--Type <RET> for more, q to quit, c to continue without paging--
info "(gdb)Auto-loading safe path"
(gdb) target remote :26000
Remote debugging using :26000
0x0000000000001000 in ?? ()
(gdb) c
Continuing.
^C
Thread 1 received signal SIGINT, Interrupt.
mycpu () at kernel/proc.c:79
79 {
(gdb) █
```

# GDB with QEMU

- Let's set a breakpoint at exec()
- Enter b exec in GDB

```
csl@sys.snu.ac.kr x + v
csl@sys ~/injae/xv6-riscv-snu % make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 4 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 -S -gdb tcp::26000

xv6 kernel is booting

hart 3 starting
hart 2 starting
hart 1 starting
init: starting sh
$ █
```

(Stopped)

```
csl@sys.snu.ac.kr x + v
Reading symbols from kernel/kernel...
warning: File "/home/csl/injae/xv6-riscv-snu/.gdbinit" auto-loading has been declined by your `auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
 add-auto-load-safe-path /home/csl/injae/xv6-riscv-snu/.gdbinit
line to your configuration file "/home/csl/.gdbinit".
To completely disable this security protection add
 set auto-load safe-path /
line to your configuration file "/home/csl/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual. E.g., run from the shell:
--Type <RET> for more, q to quit, c to continue without paging--
info "(gdb)Auto-loading safe path"
(gdb) target remote :26000
Remote debugging using :26000
0x0000000000001000 in ?? ()
(gdb) c
Continuing.
^C
Thread 1 received signal SIGINT, Interrupt.
mycpu () at kernel/proc.c:79
79 {
(gdb) b exec
Breakpoint 1 at 0x80004ec0: file kernel/exec.c, line 24.
(gdb) █
```

# GDB with QEMU

- Enter c in GDB to resume the xv6 machine

```
csl@sys.snu.ac.kr x + v
csl@sys ~/injae/xv6-riscv-snu % make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 4 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 -S -gdb tcp::26000

xv6 kernel is booting

hart 3 starting
hart 2 starting
hart 1 starting
init: starting sh
$ █
```

(Running)

```
csl@sys.snu.ac.kr x + v
by your `auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
 add-auto-load-safe-path /home/csl/injae/xv6-riscv-snu/.gdbinit
line to your configuration file "/home/csl/.gdbinit".
To completely disable this security protection add
 set auto-load safe-path /
line to your configuration file "/home/csl/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual. E.g., run from the shell:
--Type <RET> for more, q to quit, c to continue without paging--
 info "(gdb)Auto-loading safe path"
(gdb) target remote :26000
Remote debugging using :26000
0x0000000000001000 in ?? ()
(gdb) c
Continuing.
^C
Thread 1 received signal SIGINT, Interrupt.
mycpu () at kernel/proc.c:79
79 {
(gdb) b exec
Breakpoint 1 at 0x80004ec0: file kernel/exec.c, line 24.
(gdb) c
Continuing.
█
```

# GDB with QEMU

- Run ls command in the xv6 machine
- Then the xv6 machine hits the breakpoint and stops right before starting exec() function

```
csl@sys.snu.ac.kr x + v
csl@sys ~/injae/xv6-riscv-snu % make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 4 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 -S -gdb tcp::26000

xv6 kernel is booting

hart 3 starting
hart 2 starting
hart 1 starting
init: starting sh
$ ls
█
```

(Stopped)

```
csl@sys.snu.ac.kr x + v
set auto-load safe-path /
line to your configuration file "/home/csl/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual. E.g., run from the shell:
--Type <RET> for more, q to quit, c to continue without paging--
info "(gdb)Auto-loading safe path"
(gdb) target remote :26000
Remote debugging using :26000
0x0000000000001000 in ?? ()
(gdb) c
Continuing.
^C
Thread 1 received signal SIGINT, Interrupt.
mycpu () at kernel/proc.c:79
79 {
(gdb) b exec
Breakpoint 1 at 0x80004ec0: file kernel/exec.c, line 24.
(gdb) c
Continuing.
[Switching to Thread 1.2]

Thread 2 hit Breakpoint 1, exec (path=path@entry=0x3fffff9f00 "ls",
 argv=argv@entry=0x3fffff9e00) at kernel/exec.c:24
24 {
(gdb) █
```



# More about GDB

- To learn GDB in detail, search for GDB on Google
- There are many useful videos about GDB in YouTube
- [\[JTJ\]의 리눅스탐험\] GDB 활용하기](#)

Thank you!