Virtual Memory and Linux

Konsulko Group

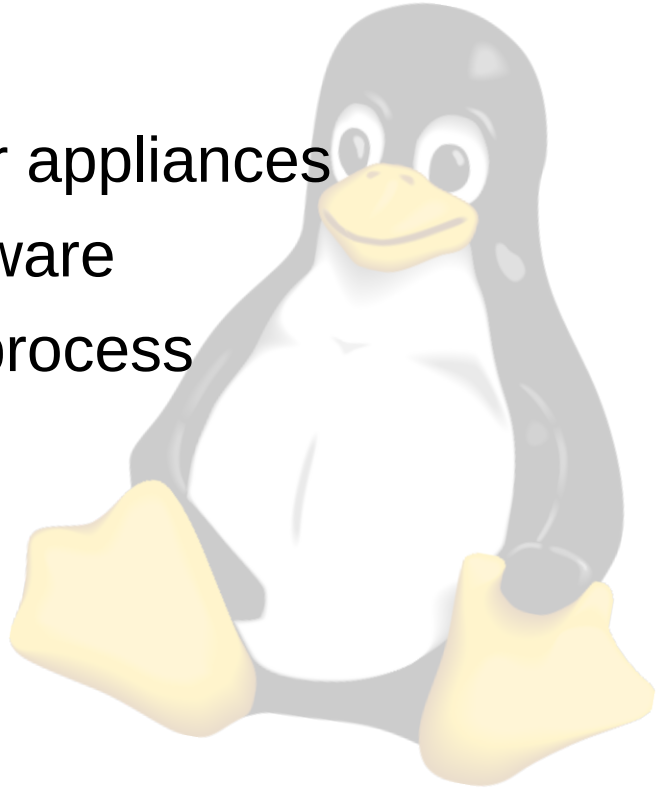
Matt Porter

Embedded Linux Conference Europe

October 13, 2016

About the original author, Alan Ott

- Unfortunately, he is unable to be here at ELCE 2016.
- Veteran embedded systems and Linux developer
- Linux Architect at **SoftIron**
  - 64-bit ARM servers and data center appliances
  - Hardware company, strong on software
  - Overdrive 3000, more products in process



Konsulko Group

Physical Memory

Konsulko Group

Single Address Space

- Simple systems have a single address space
  - Memory and peripherals share
    - Memory is mapped to one part
    - Peripherals are mapped to another
  - All processes and OS share the same memory space
    - No memory protection!
    - Processes can stomp one another
    - User space can stomp kernel mem!

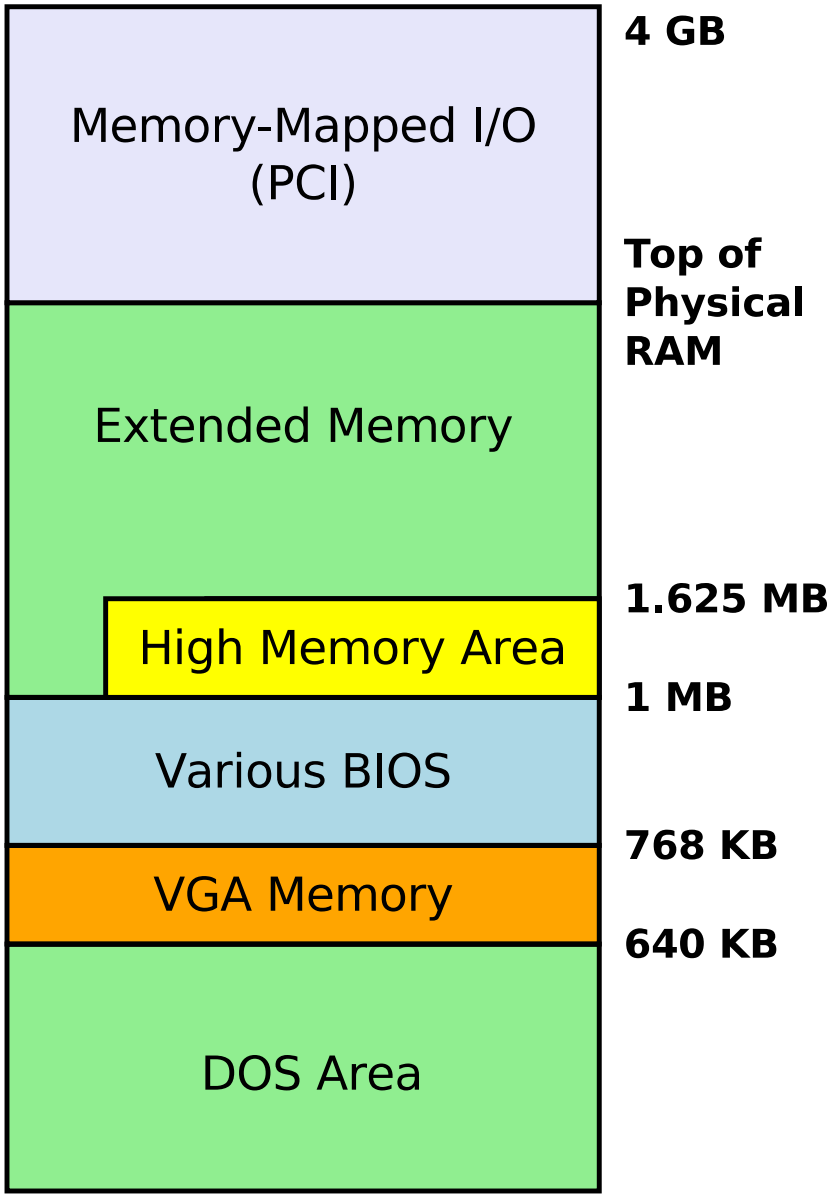
Konsulko Group

Single Address Space

- CPUs with single address space
  - 8086-80206
  - ARM Cortex-M
  - 8- and 16-bit PIC
  - AVR
  - SH-1, SH-2
  - Most 8- and 16-bit systems

Konsulko Group

x86 Physical Memory Map



- Lots of Legacy
- RAM is split (DOS Area and Extended)
- Hardware mapped between RAM areas.
- High and Extended accessed differently

Limitations

- Portable C programs expect flat memory
  - Multiple memory access methods limit portability
- Management is tricky
  - Need to know or detect total RAM
  - Need to keep processes separated
- No protection
  - Rogue programs can corrupt the entire system

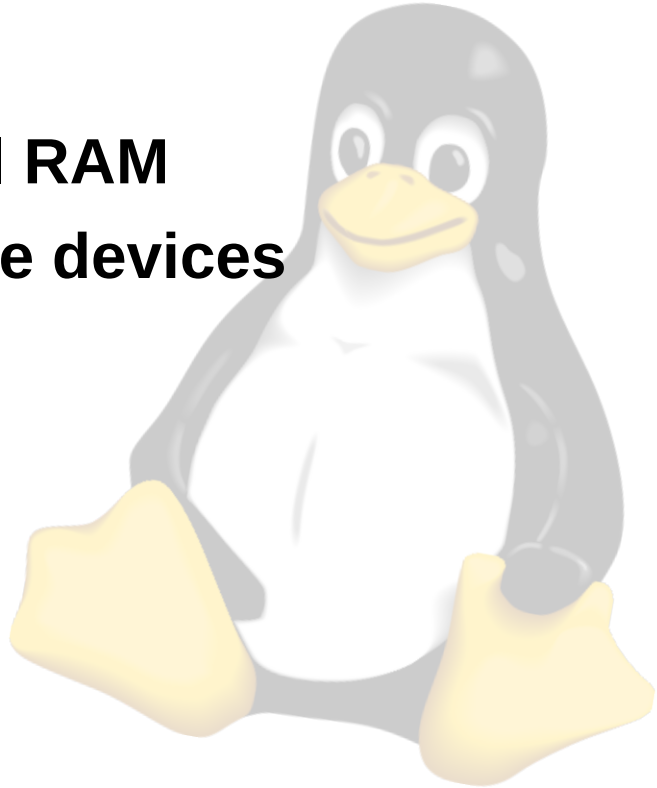
Konsulko Group

Virtual Memory

Konsulko Group

What is Virtual Memory?

- **Virtual Memory** is a system that uses an address mapping
  - Maps **virtual** address space to **physical** address space
    - Maps virtual addresses to **physical RAM**
    - Maps virtual addresses to **hardware devices**
      - PCI devices
      - GPU RAM
      - On-SoC IP blocks



Konsulko Group

What is Virtual Memory?

- Advantages
  - Each processes can have a different memory mapping
    - One process's RAM is inaccessible (and invisible) to other processes.
      - Built-in memory protection
    - Kernel RAM is invisible to user space processes
  - Memory can be moved
  - Memory can be swapped to disk

Konsulko Group

What is Virtual Memory?

- Advantages (cont)
  - Hardware device memory can be mapped into a process's address space
    - Requires the kernel to perform the mapping
  - Physical RAM can be mapped into multiple processes at once
    - Shared memory
  - Memory regions can have access permissions
    - Read, write, execute

Konsulko Group

Virtual Memory Details

- Two address spaces
  - Physical addresses
    - Addresses as used by the hardware
      - DMA, peripherals
  - Virtual addresses
    - Addresses as used by software
      - Load/Store instructions (RISC)
      - Any instruction accessing memory (CISC)

Konsulko Group

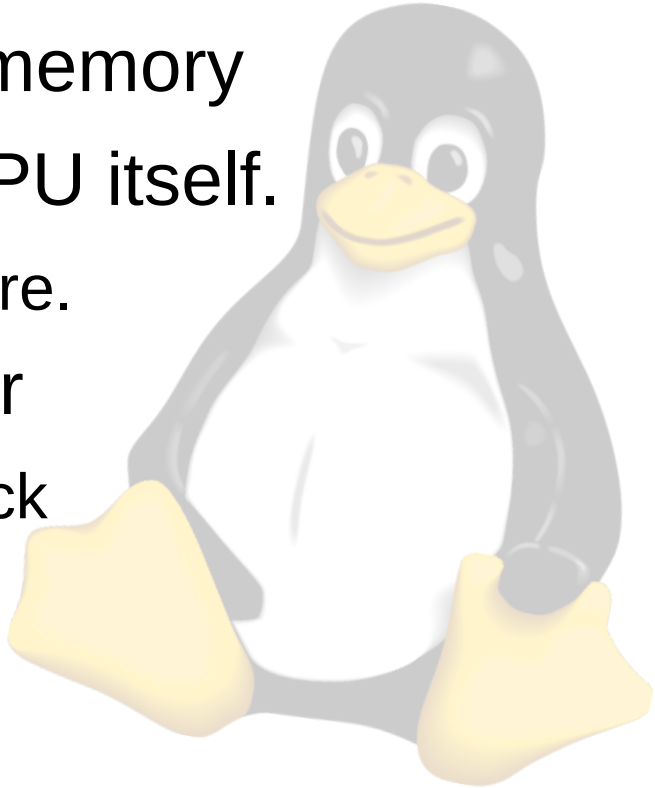
Virtual Memory Details

- Mapping is performed in hardware
  - No performance penalty for accessing already-mapped RAM regions
  - Permissions are handled without penalty
  - The same CPU instructions are used for accessing RAM and mapped hardware
  - Software, during its normal operation, will only use virtual addresses.
    - Includes kernel and user space

Konsulko Group

Memory-Management Unit

- The memory-management unit (MMU) is the hardware responsible for implementing virtual memory.
  - Sits between the CPU core and memory
  - Most often part of the physical CPU itself.
    - On ARM, it's part of the licensed core.
  - Separate from the RAM controller
    - DDR controller is a separate IP block



Konsulko Group

Memory-Management Unit

- MMU (cont)
  - Transparently handles all memory accesses from Load/Store instructions
    - Maps memory accesses using virtual addresses to **system RAM**
    - Maps accesses using virtual addresses to memory-mapped **peripheral hardware**
    - Handles **permissions**
    - Generates an exception (**page fault**) on an **invalid access**
      - Unmapped address or insufficient permissions

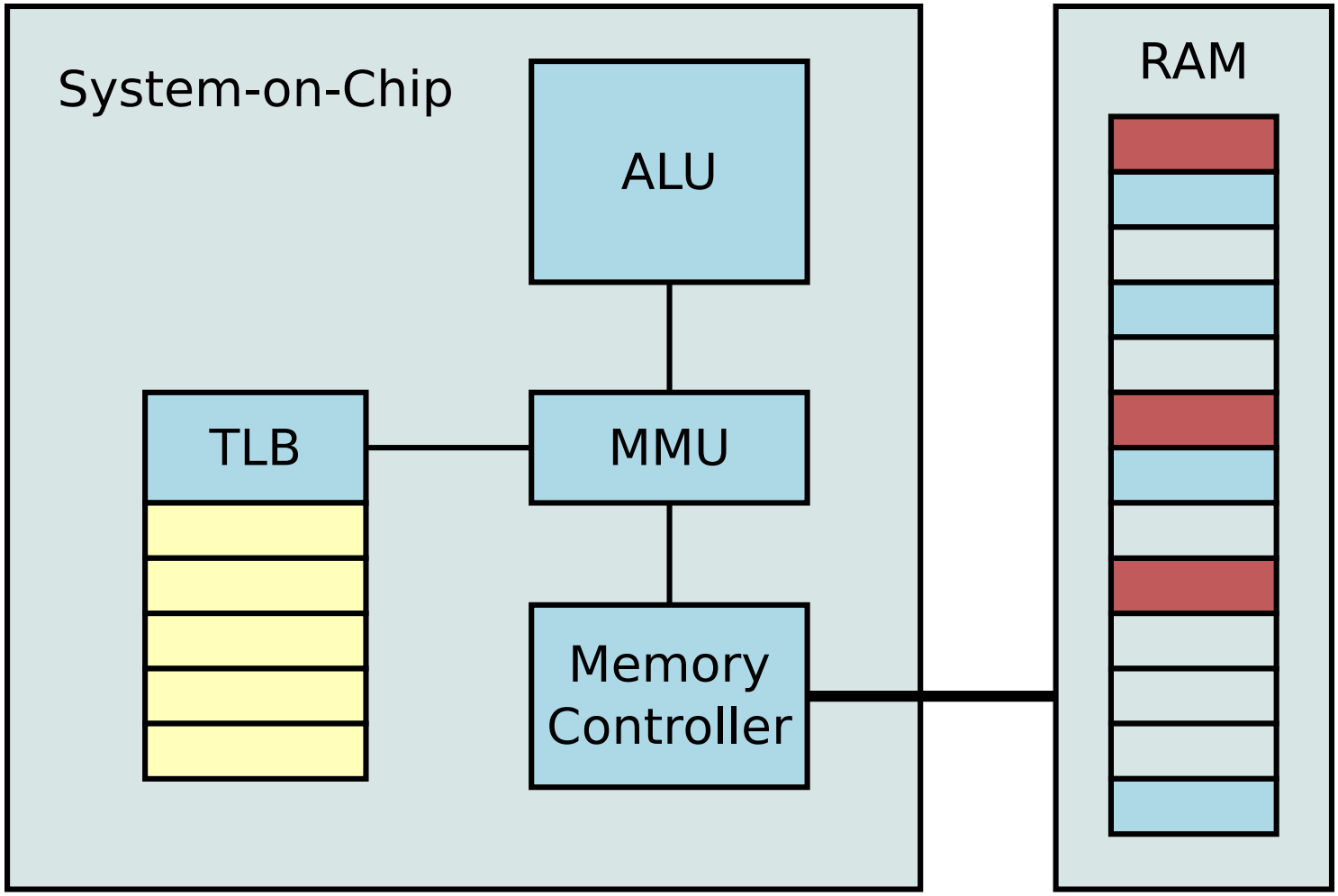
Konsulko Group

Translation Lookaside Buffer

- The TLB is a list of mappings from virtual to physical address space in hardware
  - Also holds permission bits
- There are a fixed number of entires in the TLB, which varies by CPU.
- The TLB is part of the MMU system.

Konsulko Group

Virtual Memory System (hardware)



Translation Lookaside Buffer

- TLB is consulted by the MMU when the CPU accesses a virtual address
  - If the virtual address is in the TLB, the MMU can look up the physical resource (RAM or hardware).
  - If the virtual address **is not** in the TLB, the MMU will generate a **page fault** exception and interrupt the CPU.
    - If the address is in the TLB, but the **permissions** are insufficient, the MMU will generate a page fault.

Konsulko Group

Page Faults

- A **page fault** is a CPU exception, generated when software attempts to use an **invalid virtual address**. There are three cases:
  - The virtual address is **not mapped** for the process requesting it.
  - The processes has **insufficient permissions** for the address.
  - The virtual address is valid, but **swapped out.**
    - This is a software condition

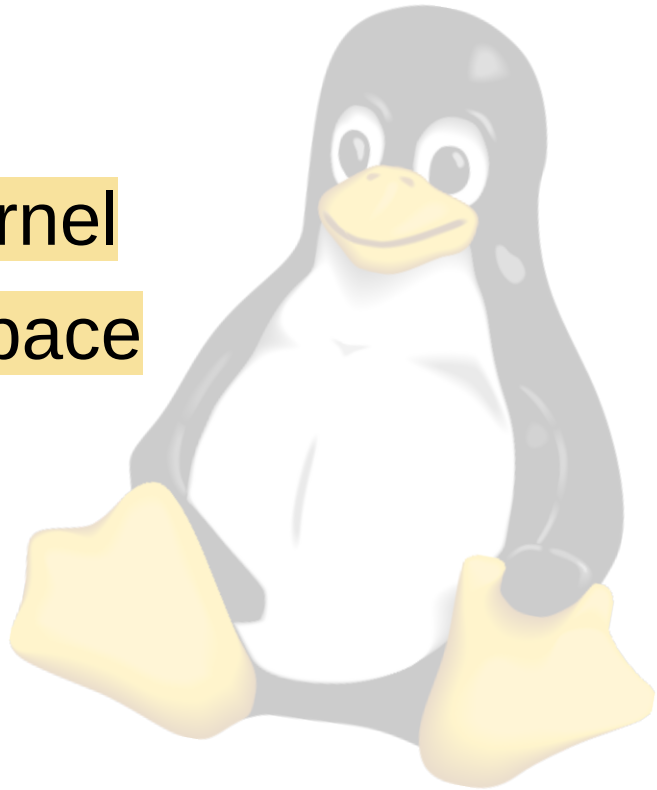
Konsulko Group

Kernel Virtual Memory

Konsulko Group

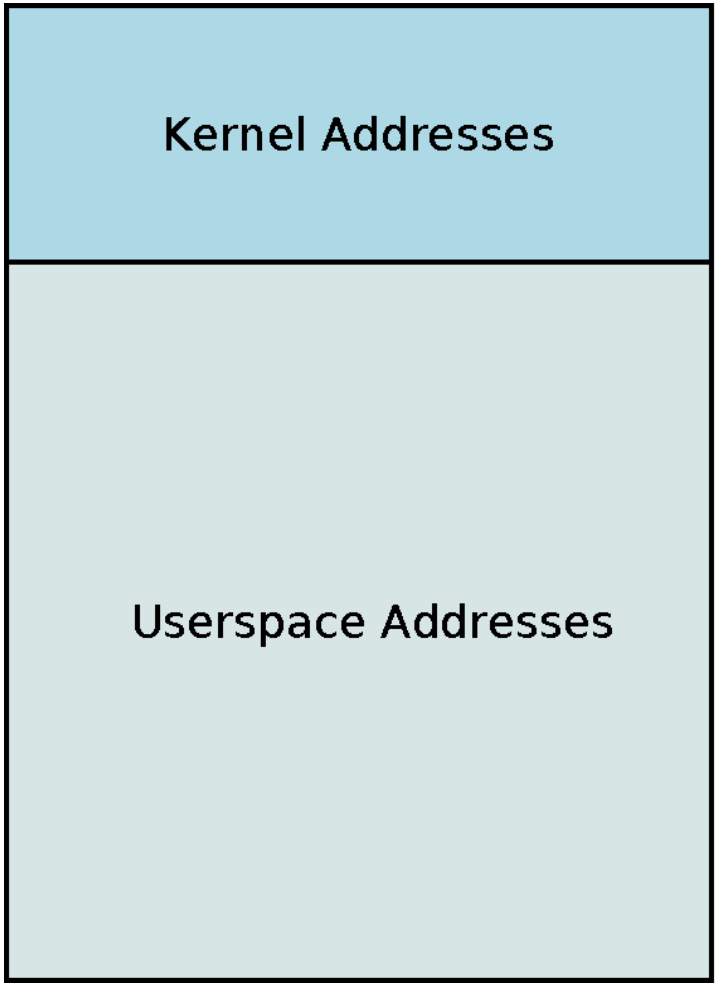
Kernel Virtual Memory

- In Linux, the kernel uses virtual addresses, as user space processes do.
  - This is not true of all OS's
- Virtual address space is split.
  - The upper part is used for the kernel
  - The lower part is used for user space
- On 32-bit, the split is at `0xC0000000`



Konsulko Group

Virtual Addresses – Linux



- By default, the **kernel** uses the **top 1GB** of virtual address space.
- Each **user space process** gets the **lower 3GB** of virtual address space.

Virtual Addresses – Linux

- Kernel address space is the area above CONFIG_PAGE_OFFSET.
  - For 32-bit, this is configurable at kernel build time.
    - The kernel can be given a different amount of address space as desired
      - See CONFIG_VMSPLIT_1G, CONFIG_VMSPLIT_2G, etc.
  - For 64-bit, the split varies by architecture, but it's high enough:
    - `0x8000000000000000` – ARM64
    - `0xffff880000000000` – x86_64

Konsulko Group

Virtual Addresses – Linux

- There are three kinds of virtual addresses in Linux.
  - The terminology varies, even in the kernel source, but the definitions in *Linux Device Drivers, 3rd Edition,* chapter 15, are somewhat standard.
    - LDD 3 can be downloaded for free at: https://lwn.net/Kernel/LDD3/

Konsulko Group

Virtual Addresses – Linux

- Three kinds of Virtual Addresses
  - Kernel:
    - Kernel Logical Address
    - Kernel Virtual Address
  - User space:
    - User Virtual Address

➢ *LDD3 also talks about Bus Addresses, which are architecture specific. We ignore them here.*

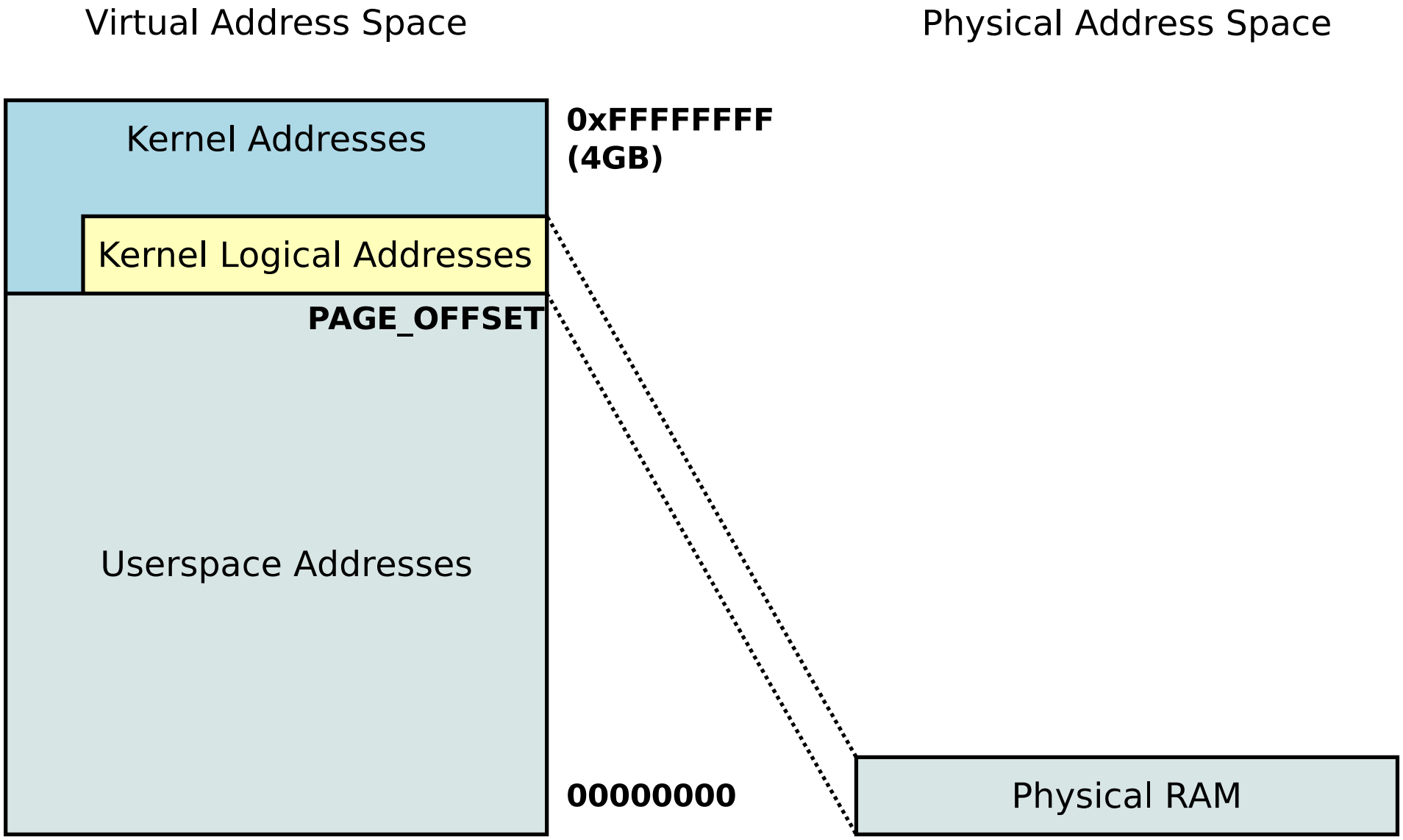
Konsulko Group

Kernel Logical Addresses

- Kernel Logical Addresses
  - Normal address space of the kernel
    - `kmalloc()`
  - Virtual addresses are a **fixed offset** from their physical addresses.
    - Virt: `0xc0000000` → Phys: `0x00000000`
  - This makes converting between physical and virtual addresses easy.

Konsulko Group

Kernel Logical Addresses



Kernel Logical Addresses

- Kernel logical addresses can be **converted** to and from physical addresses using the macros:

```
__pa(x)
__va(x)
```

- For **small-memory** systems (below ~1G of RAM) Kernel Logical address space starts at PAGE_OFFSET and goes through the **end of *physical* memory**.

Konsulko Group

Kernel Logical Addresses

- Kernel logical address space includes:
  - Memory allocated with `kmalloc()` and most other allocation methods
  - Kernel stacks (per process)
- Kernel logical memory can never be swapped out!
  - *Note that just because all physical addresses could have a kernel logical address, it doesn't mean the kernel is actually using every byte of memory on the system.*

Konsulko Group

Kernel Logical Addresses

- Kernel Logical Addresses use a **fixed mapping** between physical and virtual address space.
- This means virtually-contiguous regions are by nature also **physically contiguous**.
  - This, combined with the inability to be swapped out, makes them suitable for **DMA** transfers.

Konsulko Group

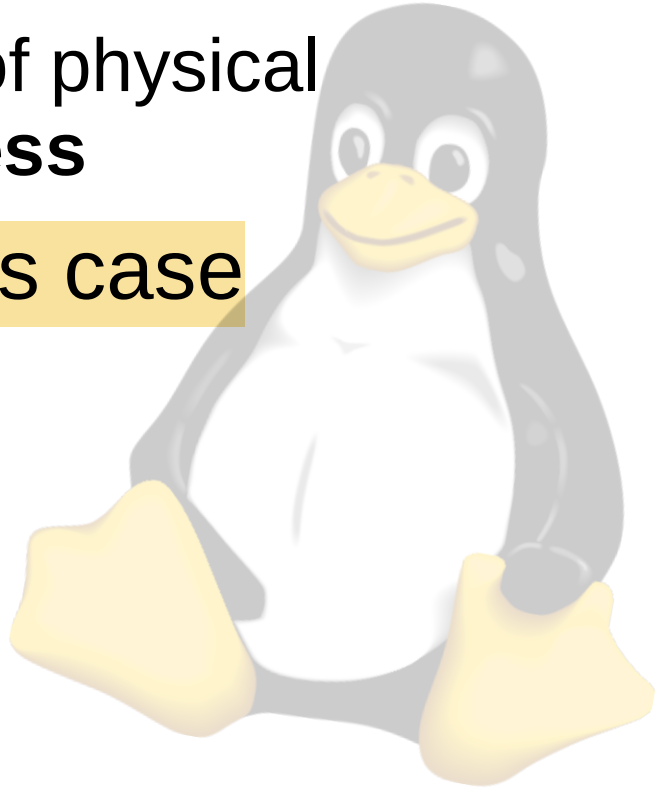
Kernel Logical Addresses

- For 32-bit **large-memory** systems (more than ~1GB RAM), not all of the physical RAM can be mapped into the kernel's address space.
  - Kernel address space is the top 1GB of virtual address space, by default.
  - Further, ~104 MB is reserved at the top of the kernel's memory space for non-contiguous allocations
    - See `vmalloc()` described later

Konsulko Group

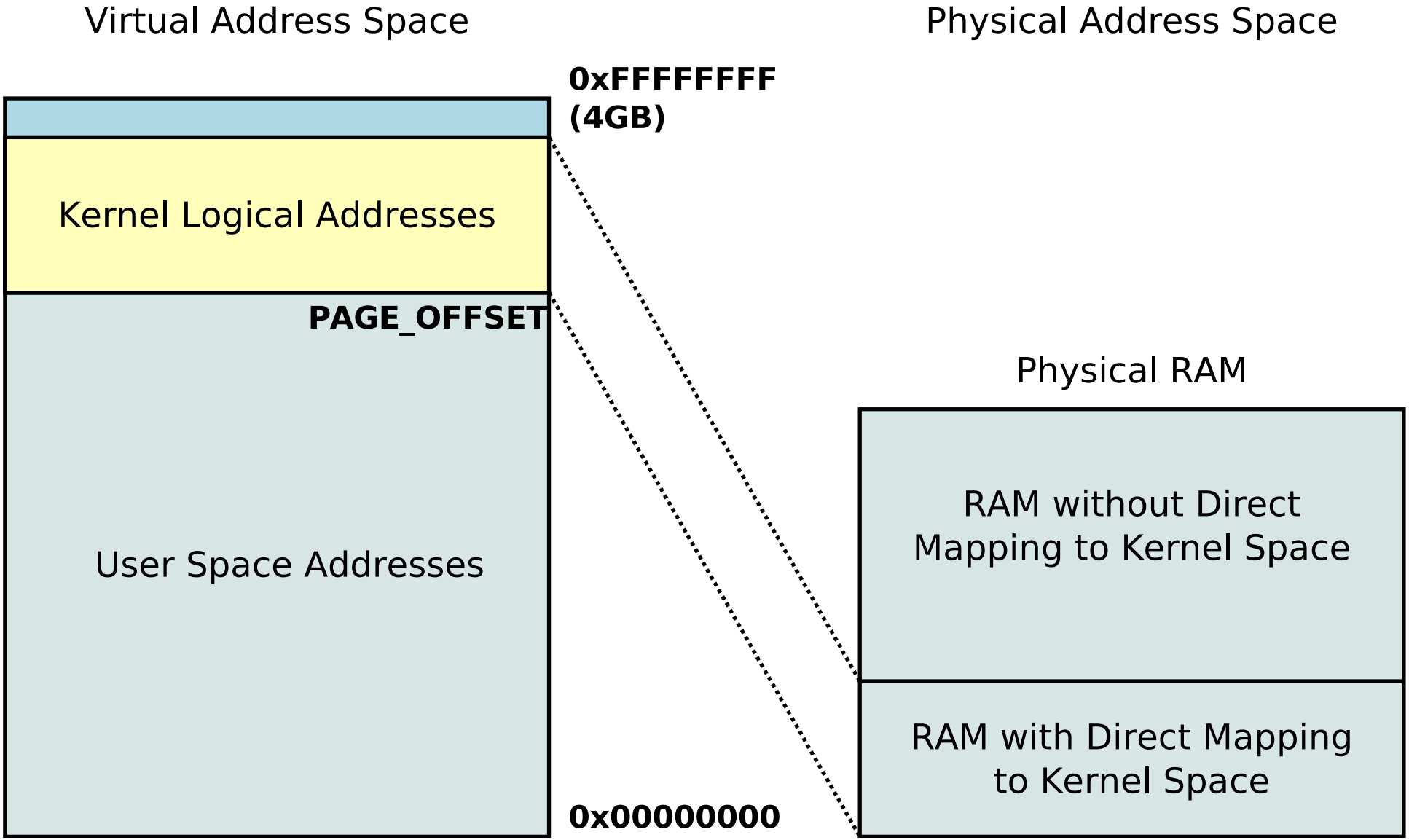
Kernel Logical Addresses

- Thus, in a large memory situation, only the **bottom part** of physical RAM is **mapped directly** into kernel logical address space.
  - Or rather, only the **bottom part** of physical RAM has a **kernel logical address**
- Note that on 64-bit systems, this case never happens.
  - There is always enough kernel address space to accommodate all the RAM.

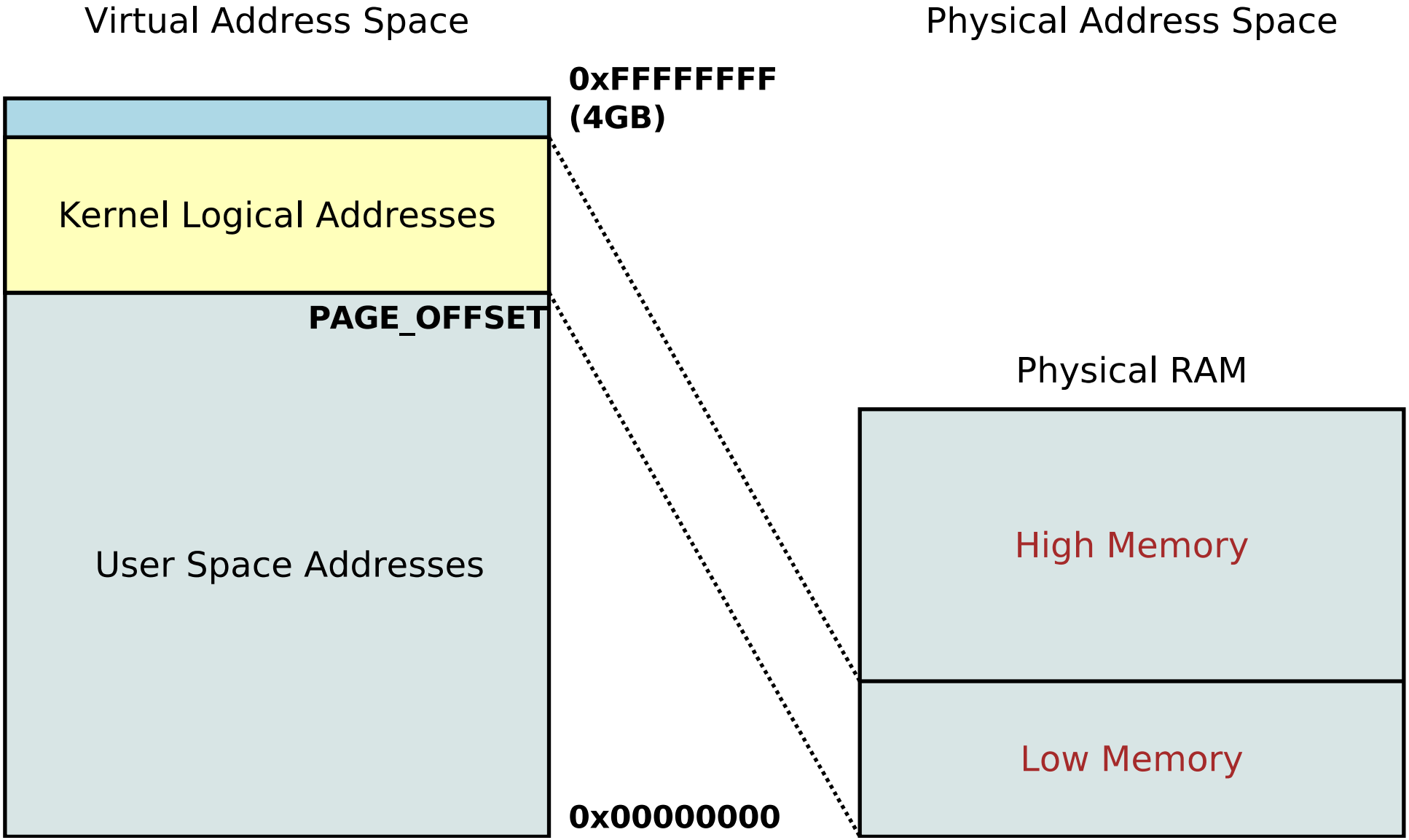


Konsulko Group

Kernel Logical Addresses (Large Mem)

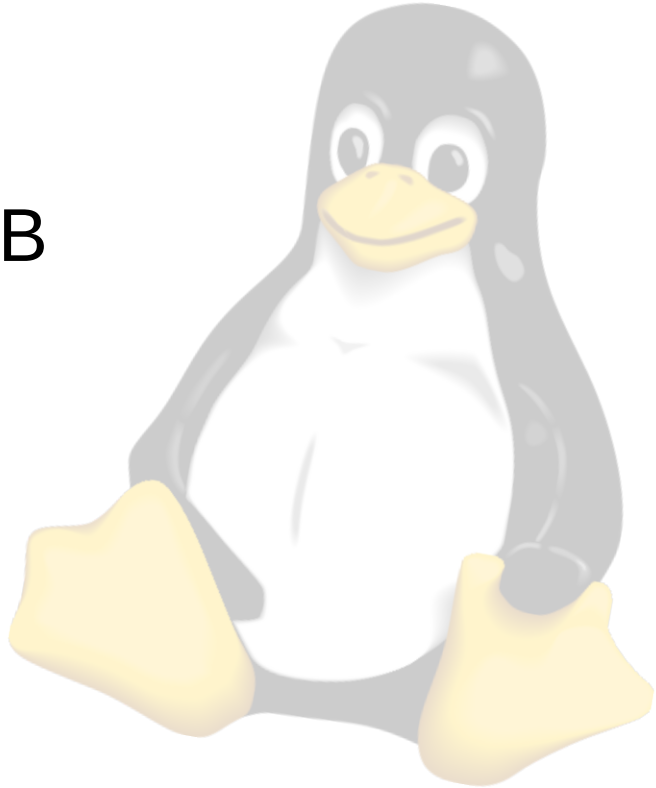


Kernel Logical Addresses (Large Mem)



Low and High Memory

- Low memory
  - Physical memory which has a kernel logical address
  - Physically contiguous
- High memory
  - Physical memory beyond ~896MB
  - Has no logical address
  - Not physically contiguous when used in the kernel
  - Only on 32-bit



Konsulko Group

Kernel Virtual Addresses

- **Kernel Virtual Addresses** are addresses in the region **above** the kernel logical address mapping.
- This is also called the `vmalloc()` area.

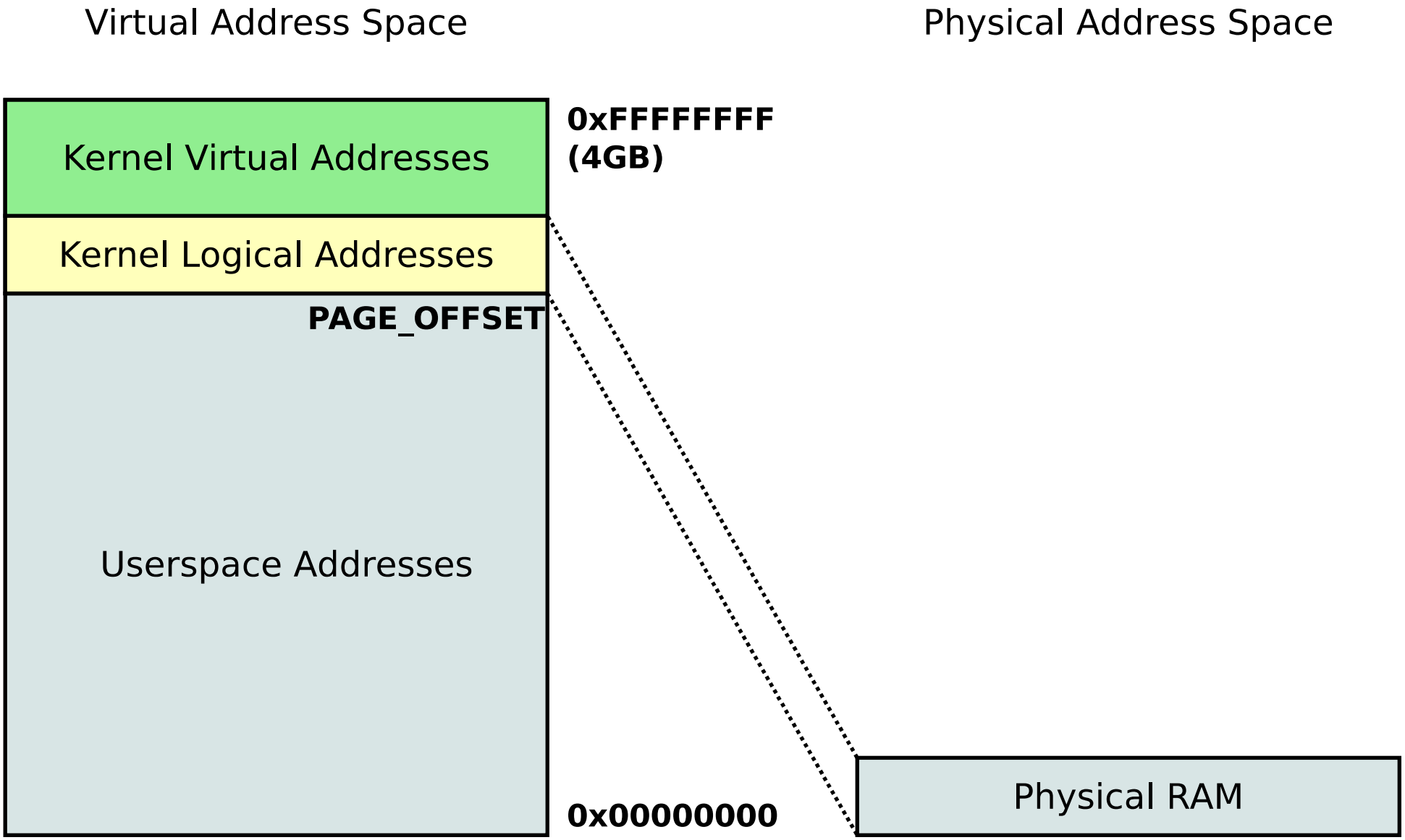
Konsulko Group

Kernel Virtual Addresses

- Kernel Virtual Addresses are used for:
  - **Non-contiguous** memory mappings
    - Often for **large buffers** which could potentially be too large to find contiguous memory
    - `vmalloc()`
  - **Memory-mapped I/O**
    - Map peripheral devices into kernel
      - PCI, SoC IP blocks
    - `ioremap()`, `kmap()`

Konsulko Group

Kernel Virtual Addresses (Small Mem)

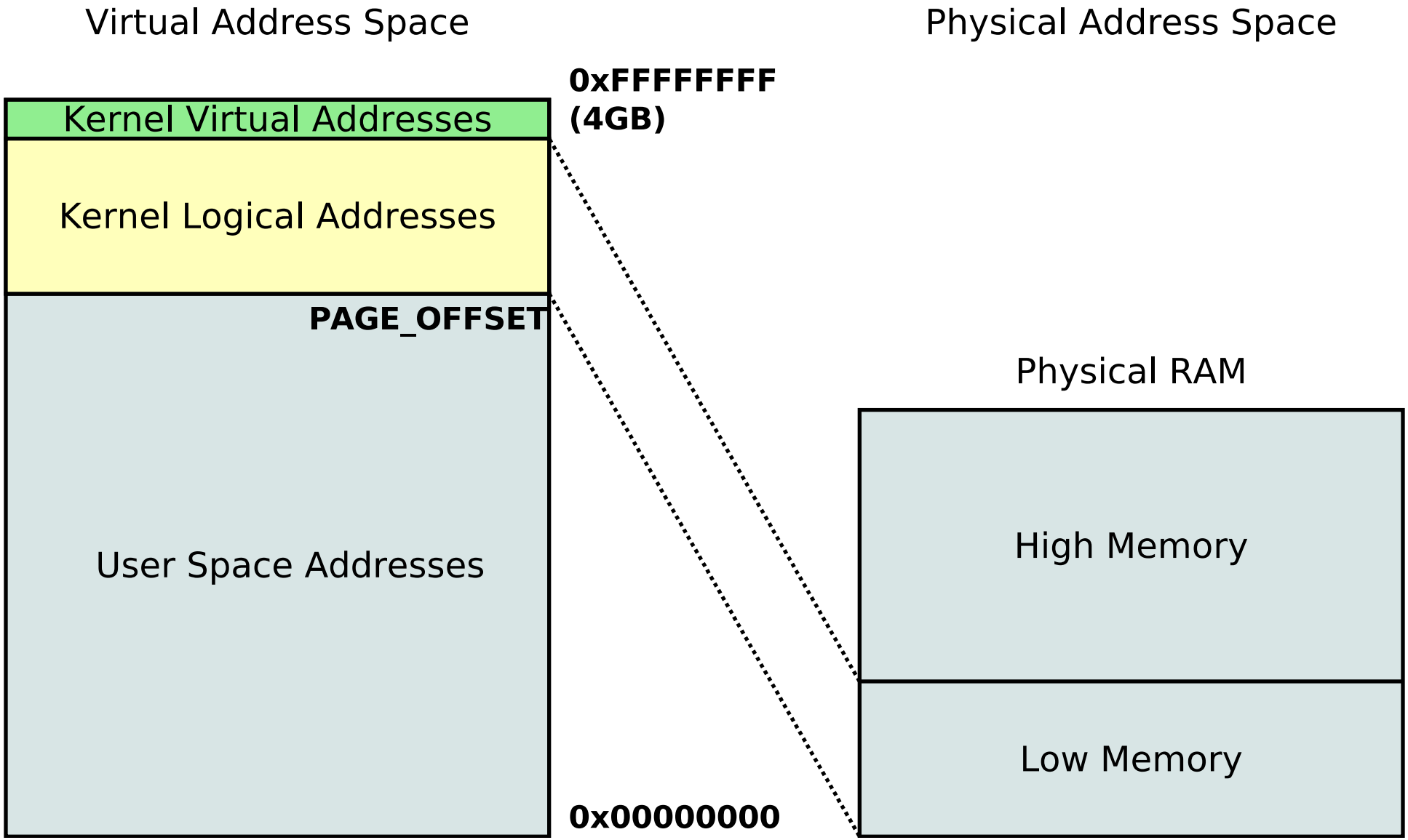


Kernel Virtual Addresses

- The important difference is that memory in the kernel virtual address area (or `vmalloc()` area) is **non-contiguous** physically.
  - This makes it easier to allocate, especially for large buffers on small-memory systems.
  - This makes it unsuitable for DMA

Konsulko Group

Kernel Virtual Addresses (Large Mem)



Kernel Virtual Addresses

- In a large memory situation, the kernel virtual address area is **smaller**, because there is more physical memory.
  - An interesting case, where **more memory** means **less** space for kernel virtual addresses.
  - In 64-bit, of course, this doesn't happen, as PAGE_OFFSET is large, and there is much more virtual address space.

Konsulko Group

User Virtual Memory

Konsulko Group

User Virtual Addresses

- **User Virtual Addresses** represent memory used by user space programs.
  - This is most of the memory on most systems
  - This is where most of the complication is
- User virtual addresses are all addresses **below** PAGE_OFFSET.
- Each process has its own mapping
  - Threads share a mapping
  - Complex behavior with `clone(2)`

Konsulko Group

User Virtual Addresses

- Unlike kernel logical addresses, which use a fixed mapping between virtual and physical addresses, user space processes make full use of the MMU.
  - Only the used portions of RAM are mapped
  - Memory is not contiguous
  - Memory may be swapped out
  - Memory can be moved

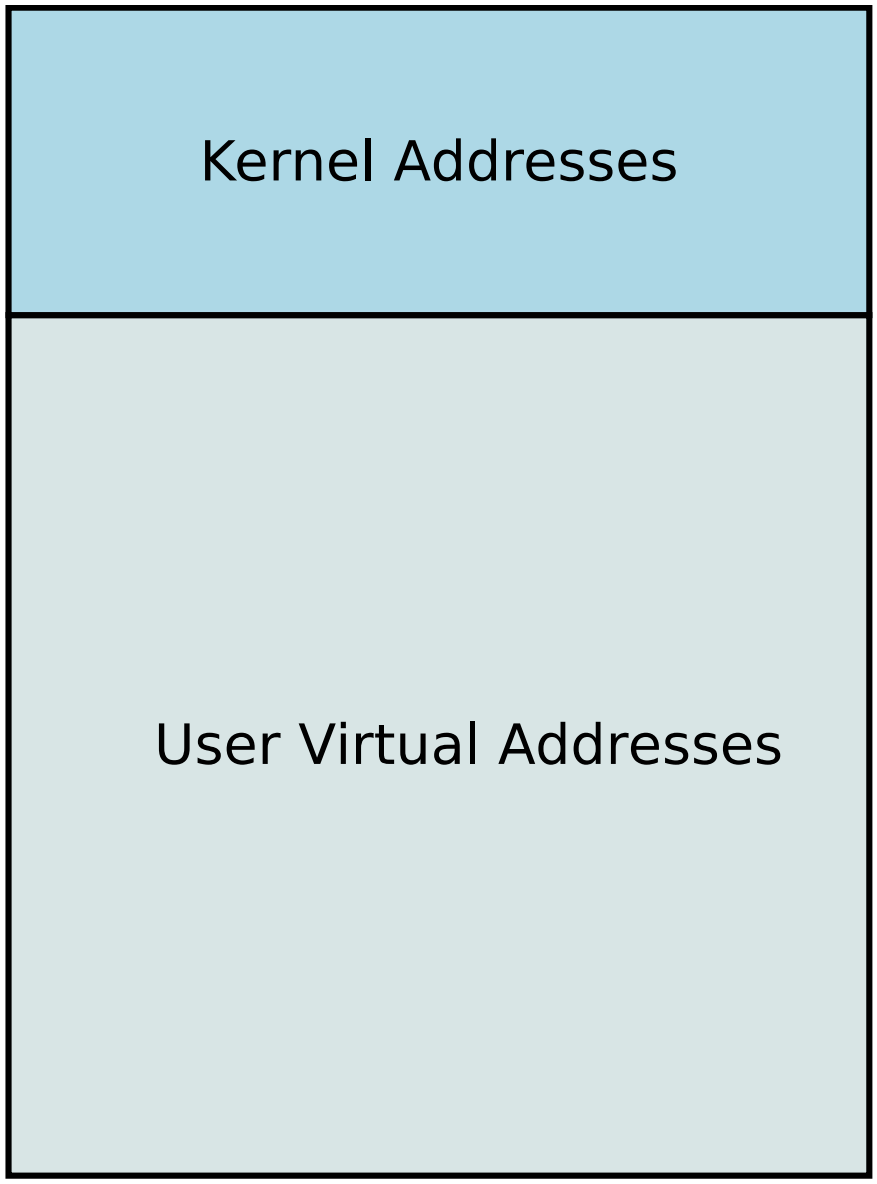
Konsulko Group

User Virtual Addresses

- Since user virtual addresses are not guaranteed to be swapped in, or even allocated at all, user buffers are not suitable for use by the kernel (or for DMA), by default.
- Each process has its own memory map
  - `struct mm`, pointers in `task_struct`
- At context switch time, the memory map is changed.
  - This is part of the overhead

Konsulko Group

User Virtual Addresses



- Each process will have its own mapping for user virtual addresses
- The mapping is changed during context switch

The Memory Management Unit

Konsulko Group

The MMU

- The Memory Management Unit (MMU) is a hardware component which manages virtual address mappings
  - Maps virtual addresses to physical addresses
- The MMU operates on basic units of memory called **pages**.
  - Page size varies by architecture
  - Some architectures have configurable page sizes

Konsulko Group

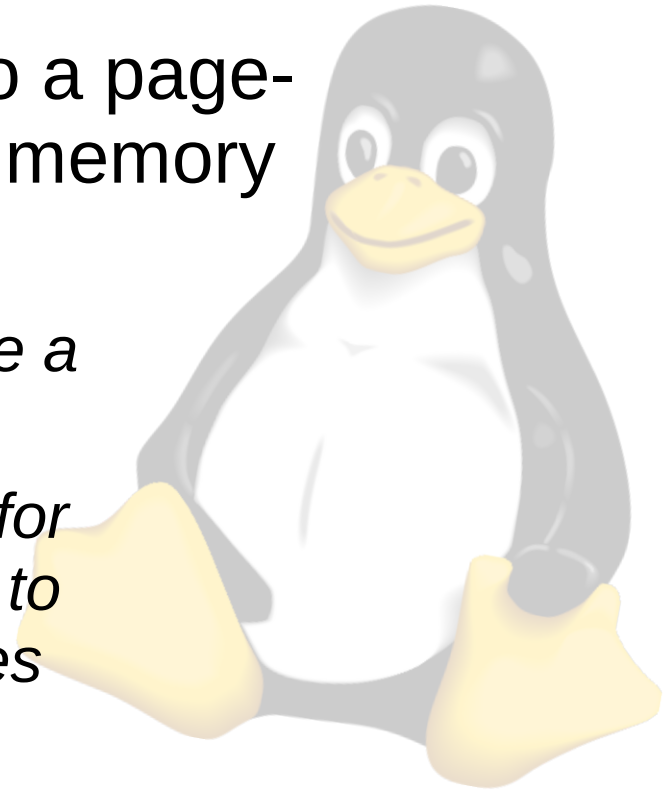
The MMU

- Common page sizes:
  - ARM – 4k
  - ARM64 – 4k or 64k
  - MIPS – widely configurable
  - x86 – 4k
    - *Architectures which are configurable are configured at kernel build time.*

Konsulko Group

The MMU

- Terminology
  - A **page** is a unit of memory sized and aligned at the page size.
  - A **page frame**, or frame, refers to a page-sized and page-aligned physical memory block.
    - *A page is somewhat abstract, where a frame is concrete*
    - *In the kernel, the abbreviation **pfn**, for **page frame number**, is often used to refer to refer to physical page frames*



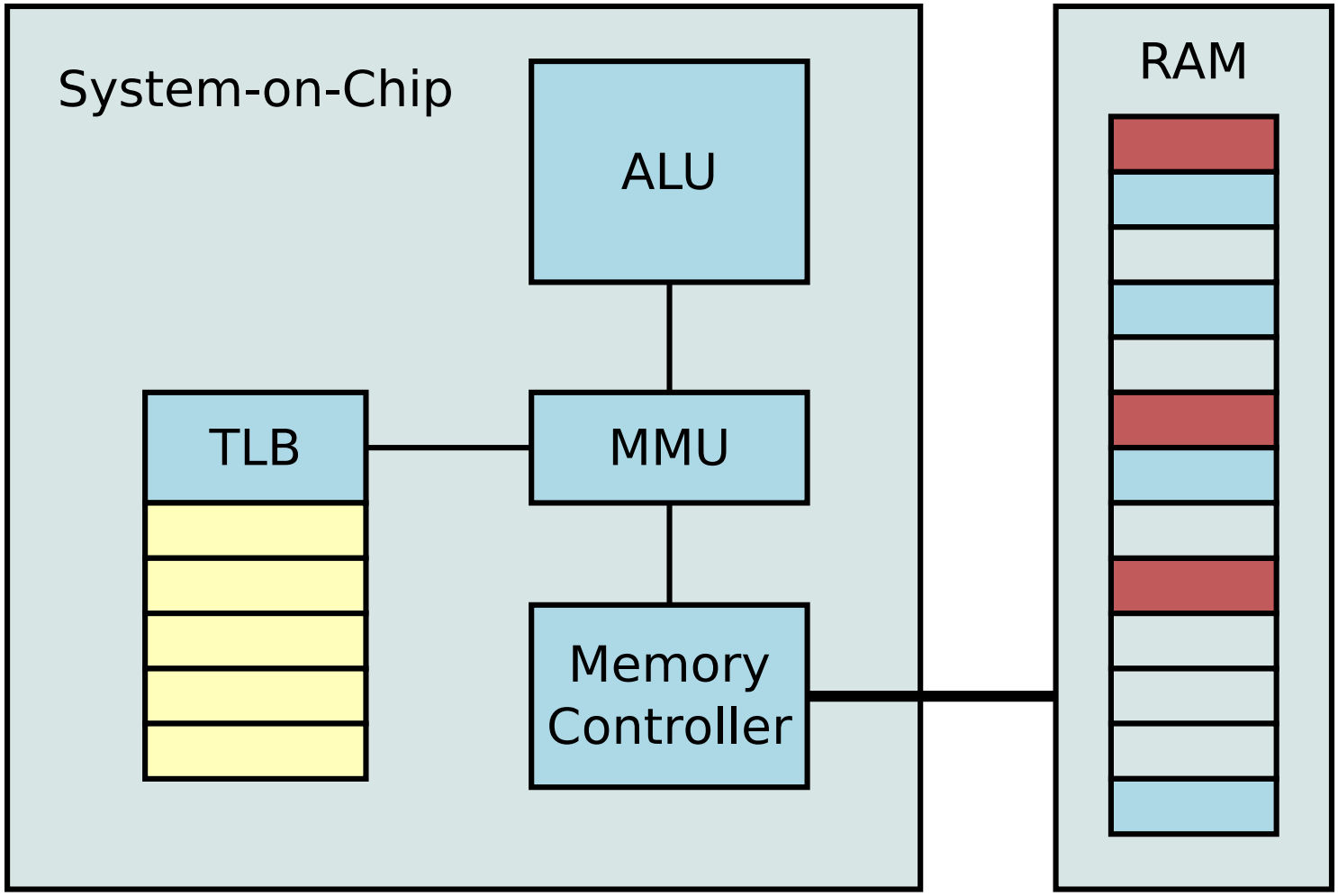
Konsulko Group

The MMU

- The MMU operates on pages
  - The MMU maps physical frames to virtual addresses.
  - A memory map for a process will contain **many mappings**
  - A mapping often covers **multiple pages**
  - The TLB holds each mapping
    - Virtual address
    - Physical address
    - Permissions

Konsulko Group

Virtual Memory System (hardware)



Page Faults

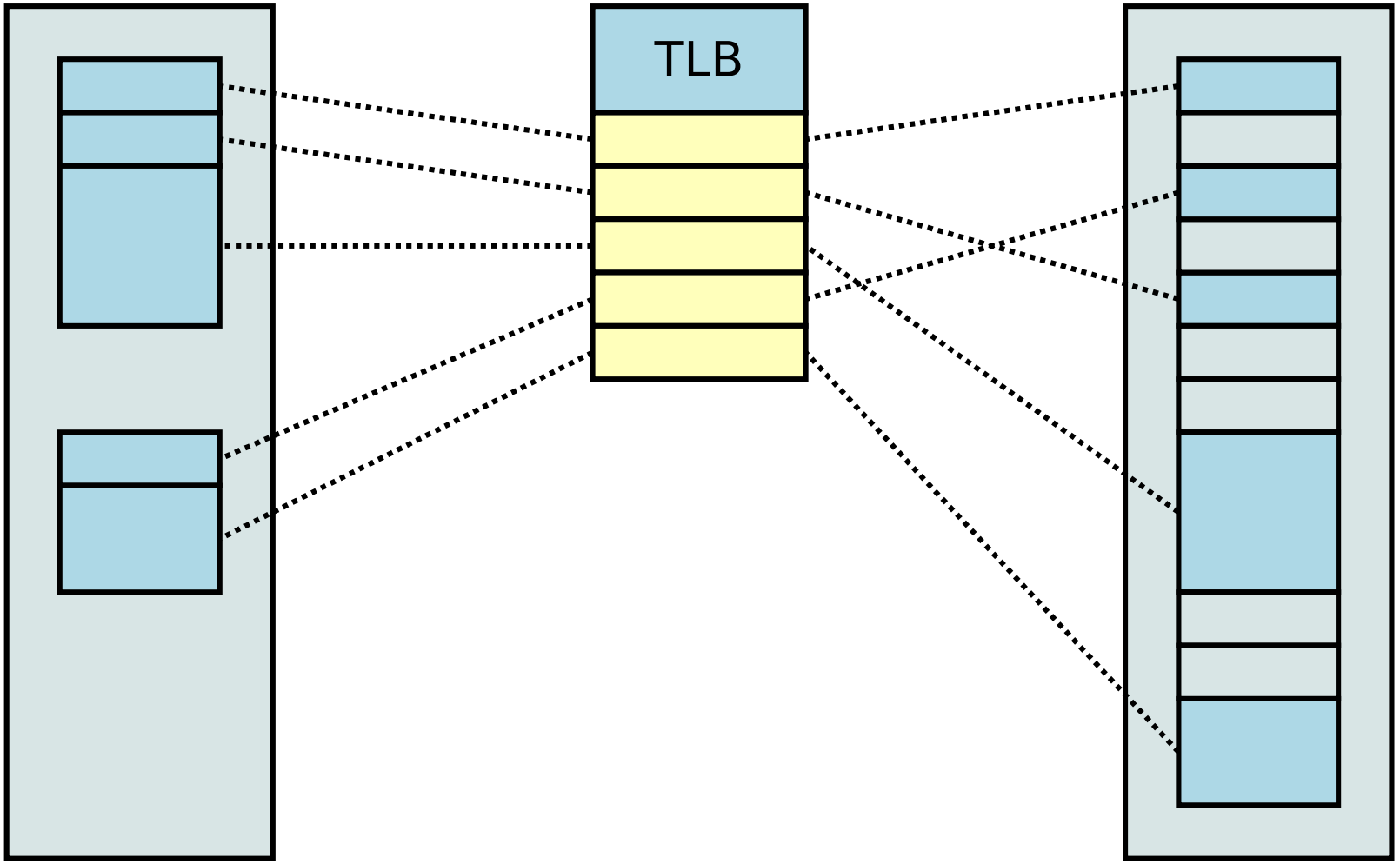
- When a process accesses a region of memory that is not mapped, the MMU will generate a page fault exception.
  - The kernel handles page fault exceptions regularly as part of its memory management design.
    - TLB is often smaller than the total number of maps for a process.
    - Page faults at context switch time
    - Lazy allocation

Konsulko Group

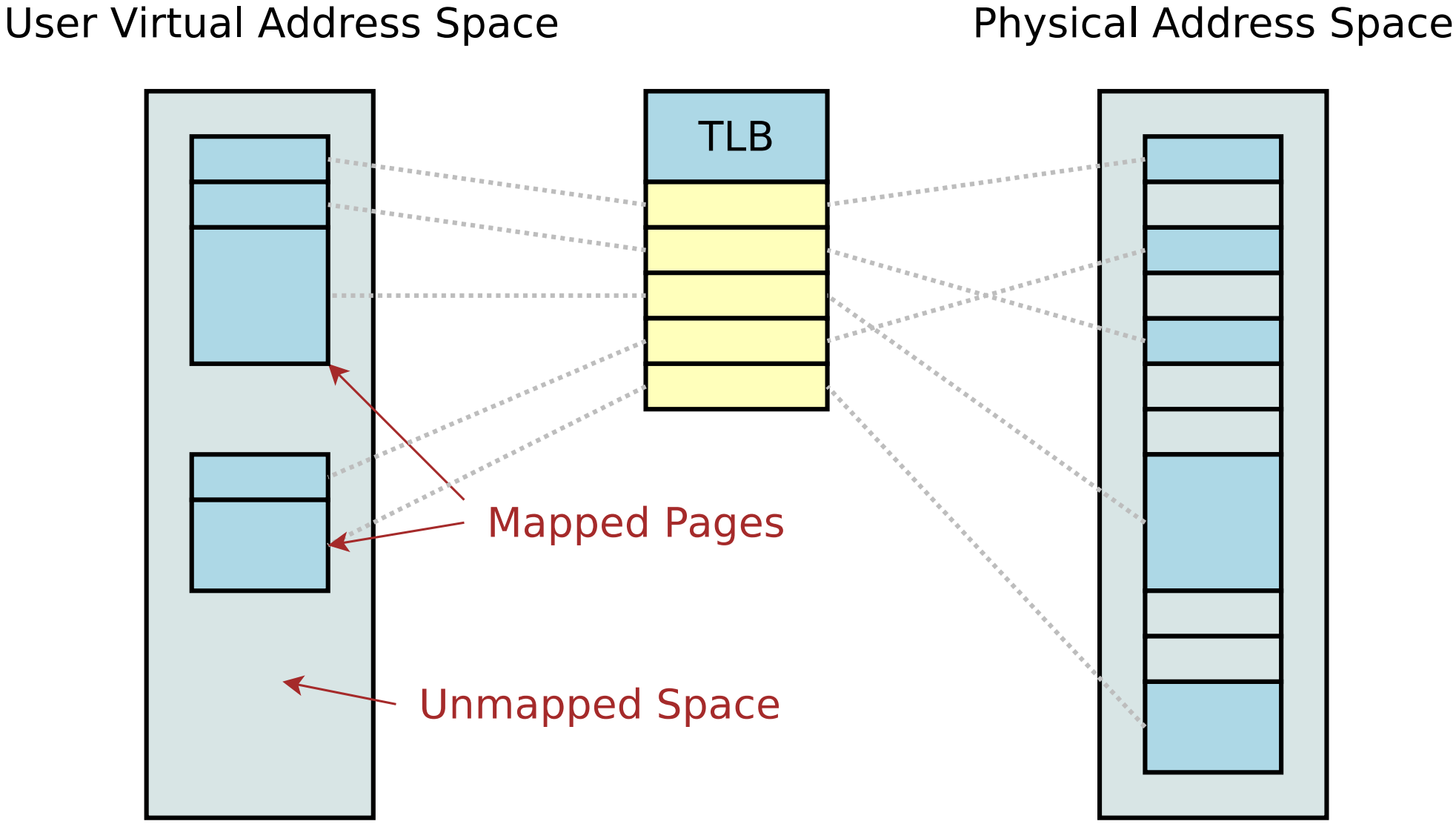
Basic TLB Mappings

User Virtual Address Space

Physical Address Space



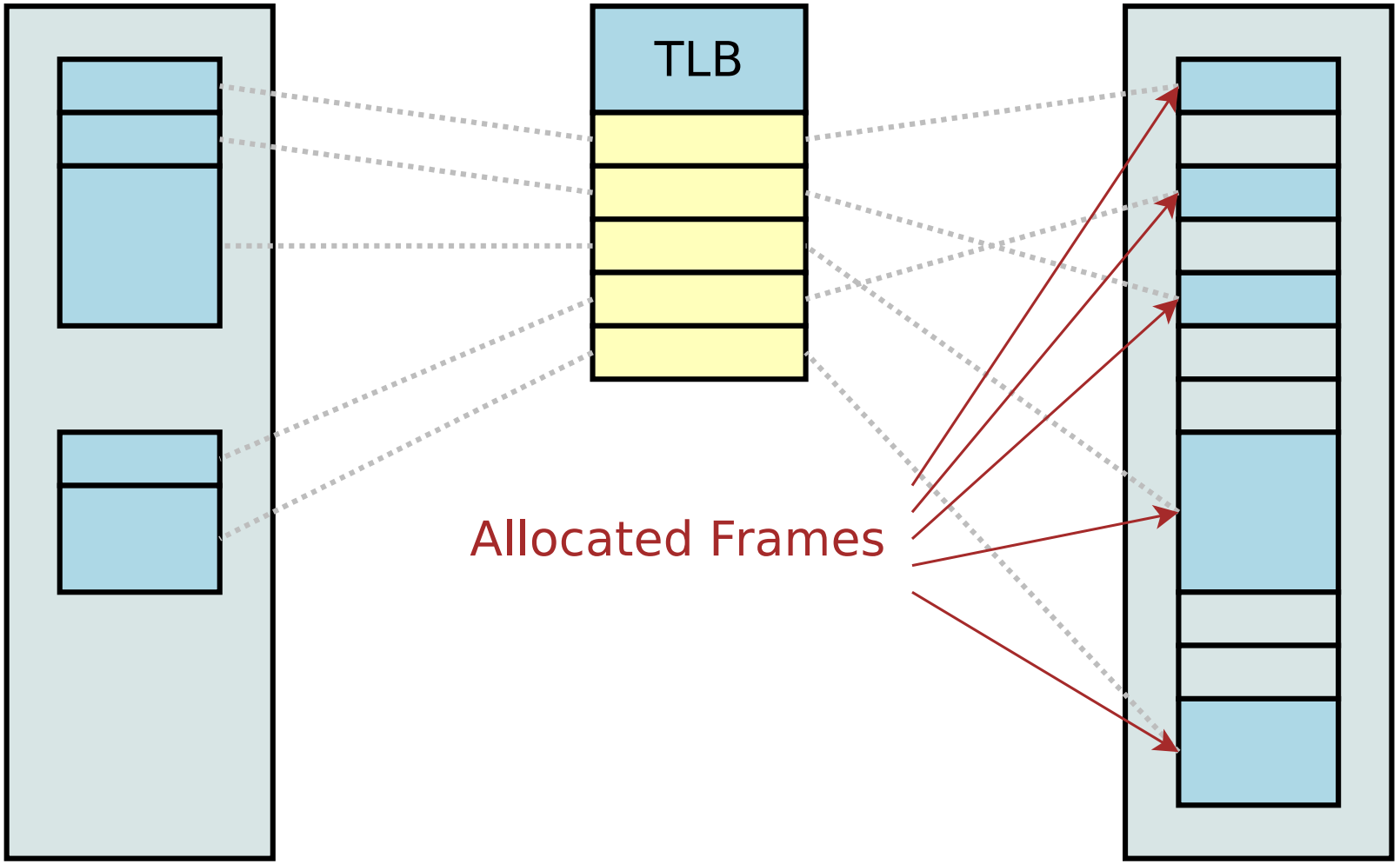
Basic TLB Mappings



Basic TLB Mappings

User Virtual Address Space

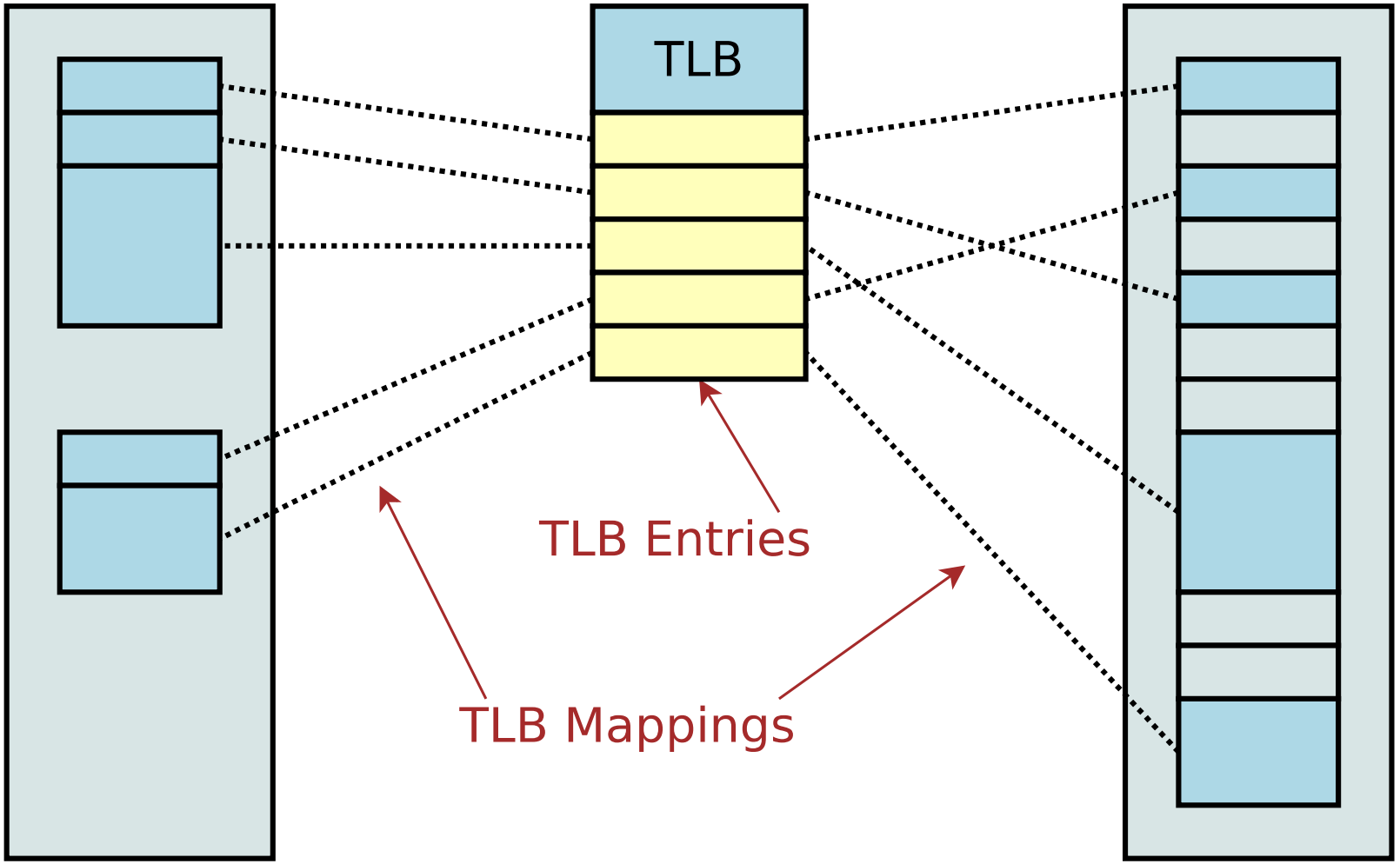
Physical Address Space



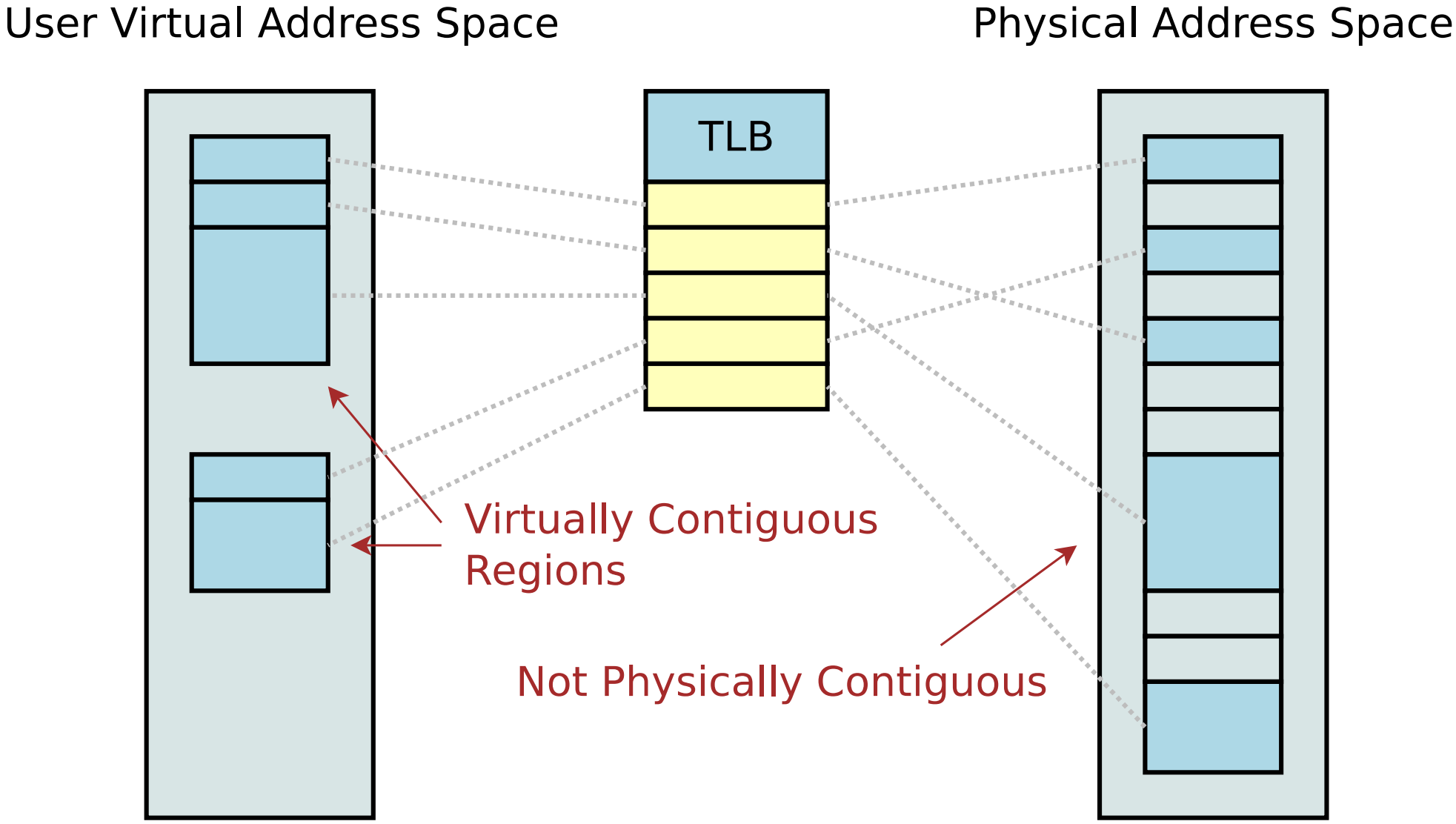
Basic TLB Mappings

User Virtual Address Space

Physical Address Space



Basic TLB Mappings



Basic TLB Mappings

- Mappings to virtually contiguous regions do not have to be physically contiguous.
  - This makes memory easier to allocate.
  - Almost all user space code does not need physically contiguous memory.

Konsulko Group

Multiple Processes

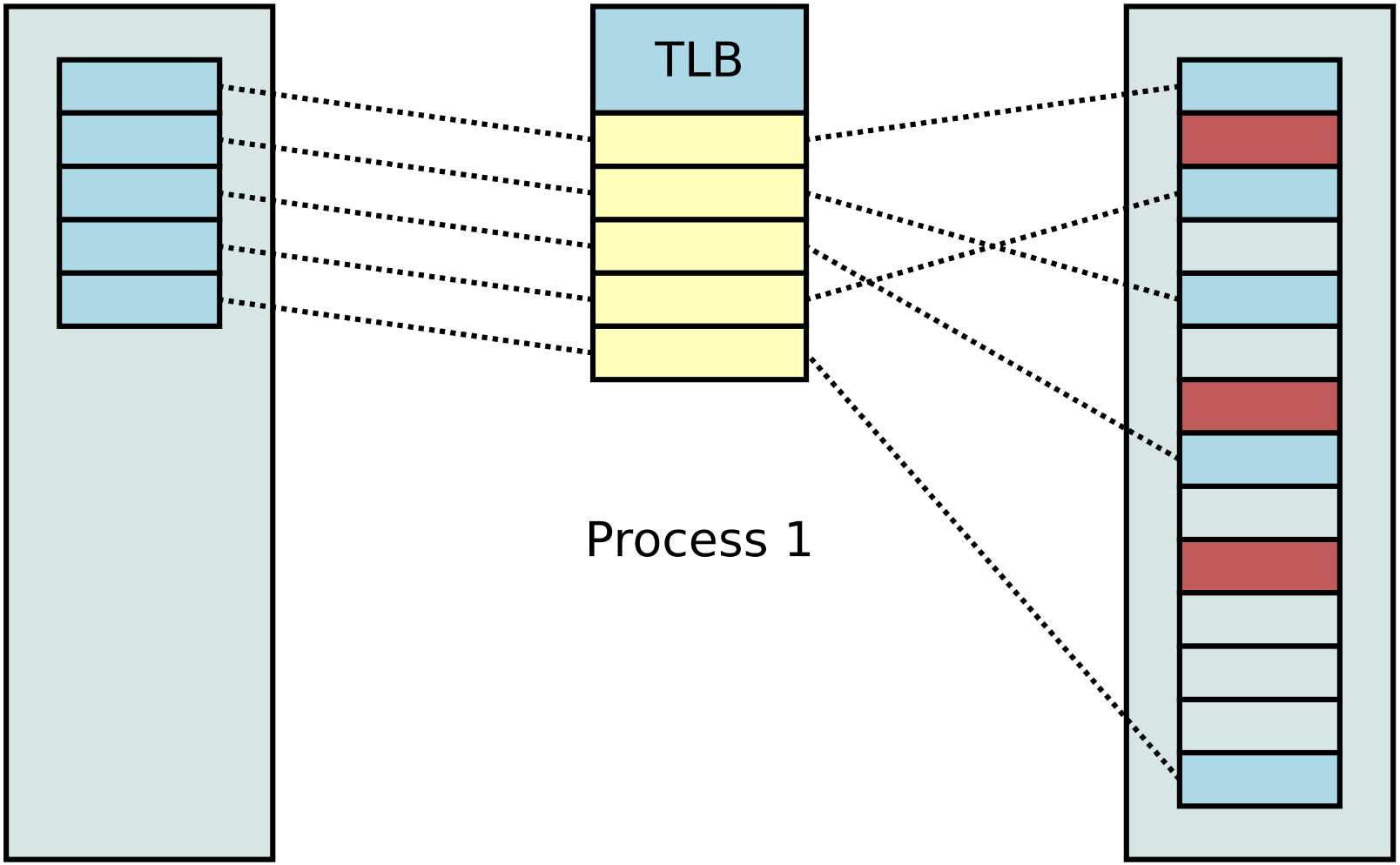
- Each process has its own set of mappings.
  - The same virtual addresses in two different processes will likely be used to map **different physical addresses**.

Konsulko Group

Multiple Processes – Process 1

User Virtual Address Space

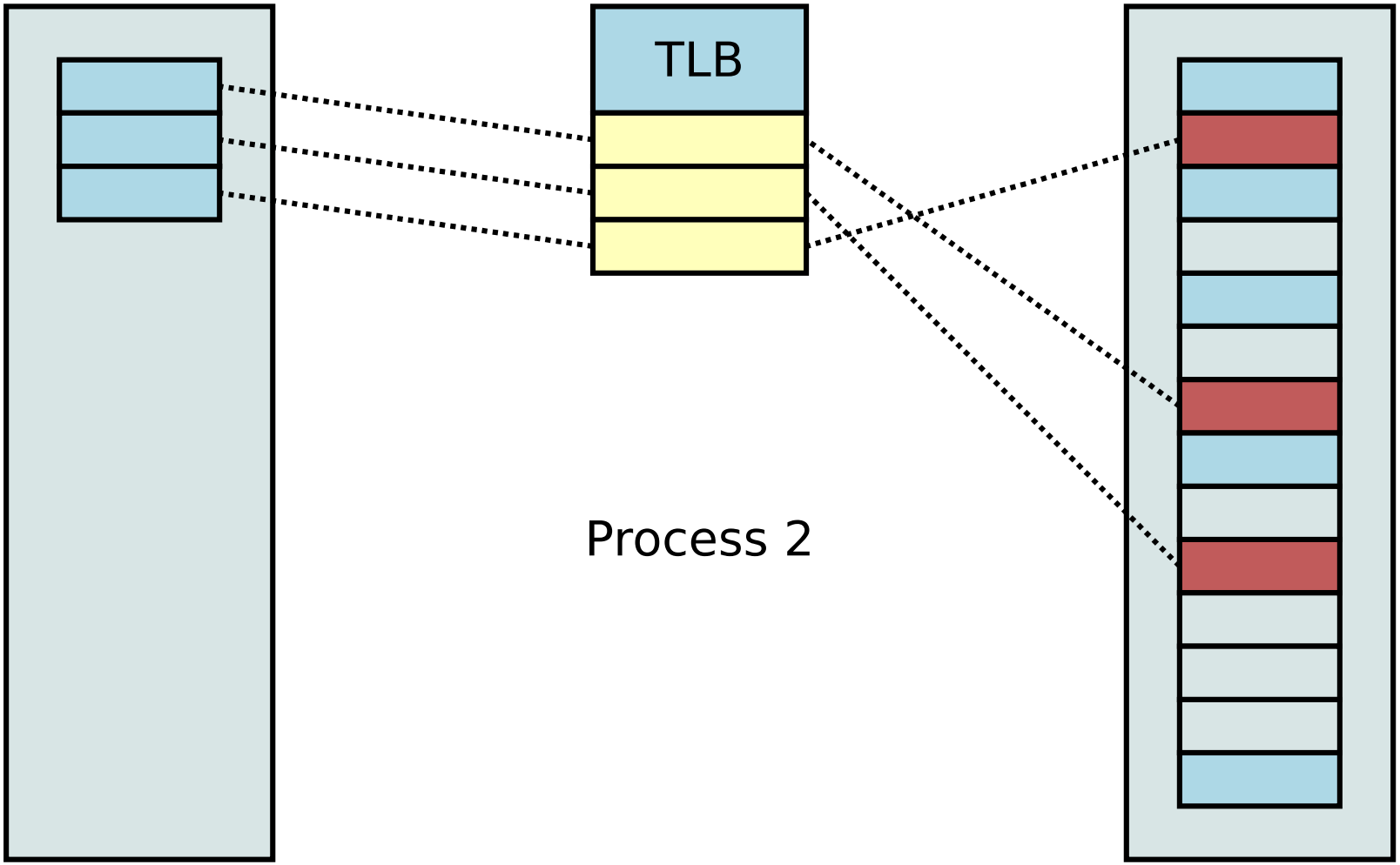
Physical Address Space



Multiple Processes – Process 2

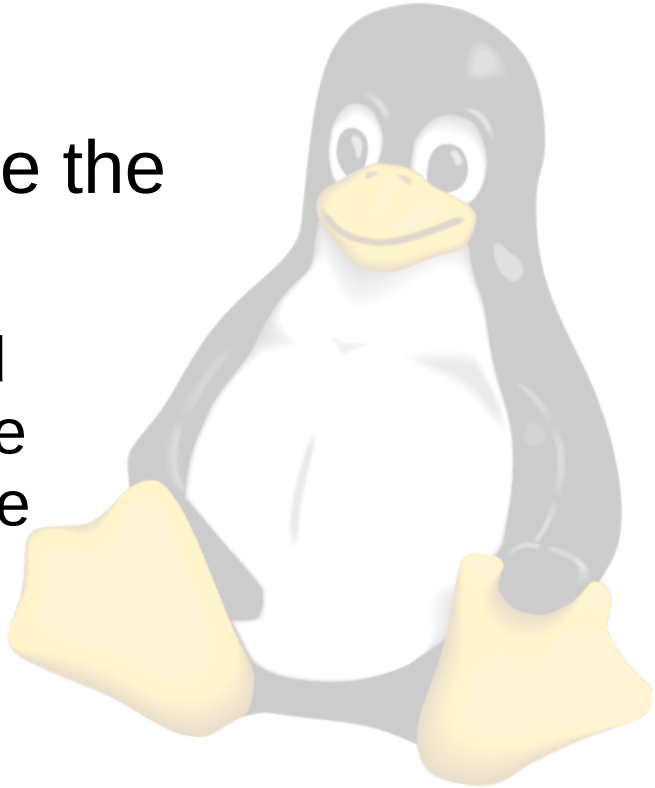
User Virtual Address Space

Physical Address Space



Shared Memory

- Shared memory is easily implemented with an MMU.
  - Simply map the same physical frame into two different processes.
  - The virtual addresses need not be the same.
    - If pointers to values inside a shared memory region are used, it might be important for them to have the same virtual addresses, though.

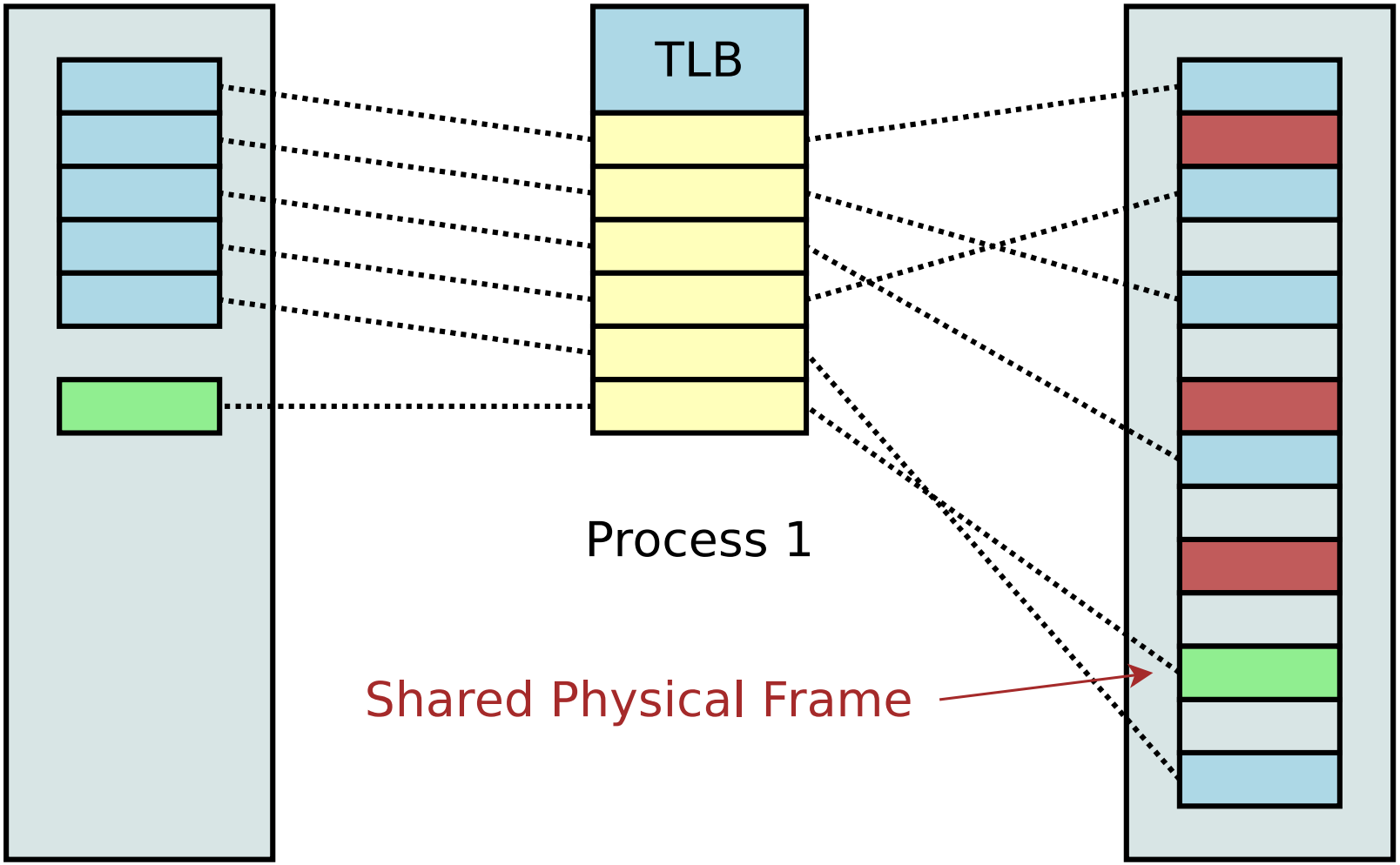


Konsulko Group

Shared Memory – Process 1

User Virtual Address Space

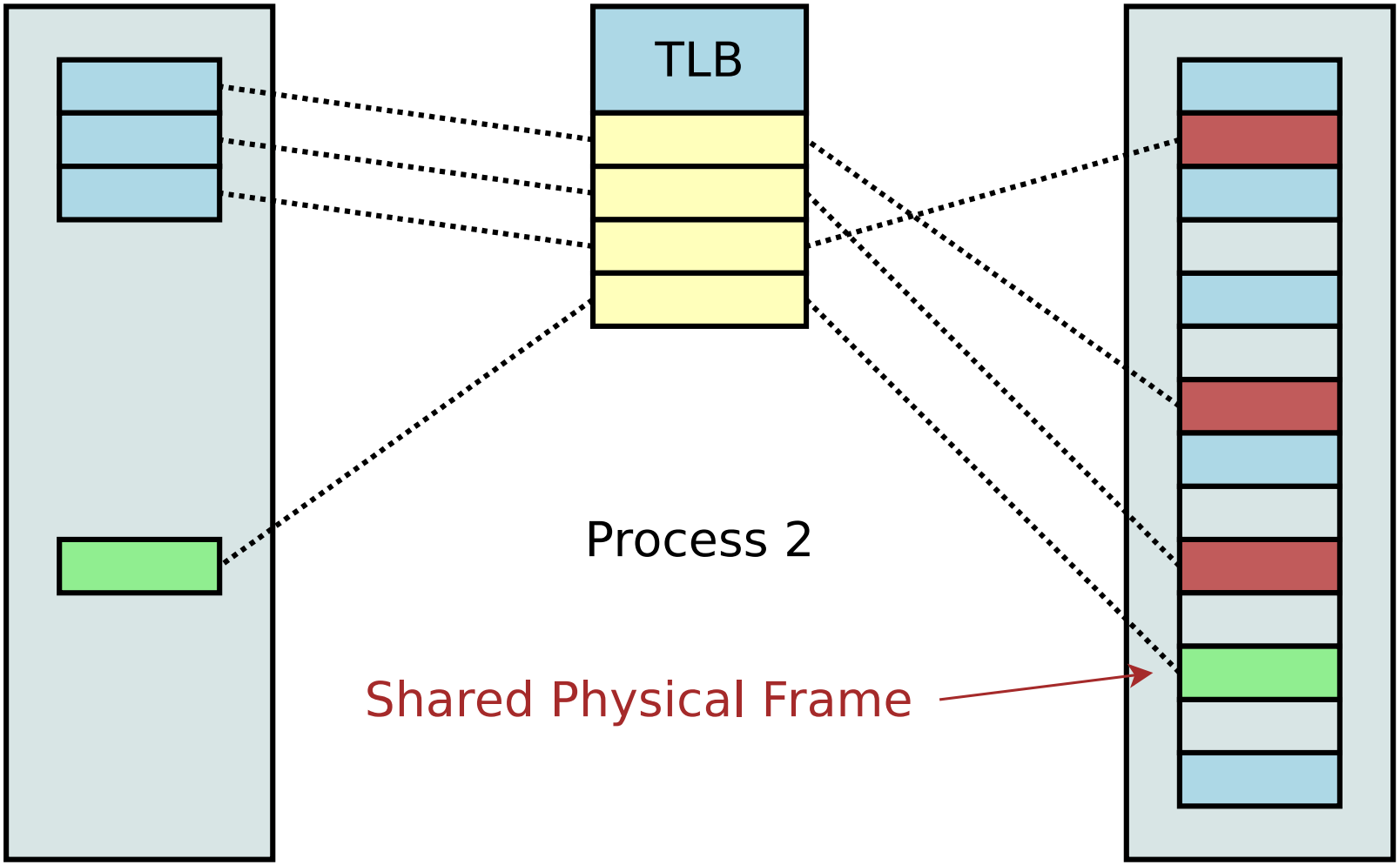
Physical Address Space



Shared Memory – Process 2

User Virtual Address Space

Physical Address Space



Shared Memory

- Note in the previous example, the shared memory region was mapped to **different virtual addresses** in each process.
- The `mmap()` system call allows the user space process to **request** a **specific virtual address** to map the shared memory region.
  - The kernel may not be able to grant a mapping at this address, causing `mmap()` to return failure.

Konsulko Group

Lazy Allocation

- The kernel will not allocate pages requested by a process immediately.
  - The kernel will wait until those pages are actually used.
  - This is called **lazy allocation** and is a performance optimization.
    - For memory that gets allocated but doesn't get used, allocation never has to happen!

Konsulko Group

Lazy Allocation

- Process
  - When memory is requested, the kernel simply creates a record of the request in its page tables and then returns (quickly) to the process, without updating the TLB.
  - When that newly-allocated memory is touched, the CPU will generate a page fault, because the CPU doesn't know about the mapping.

Konsulko Group

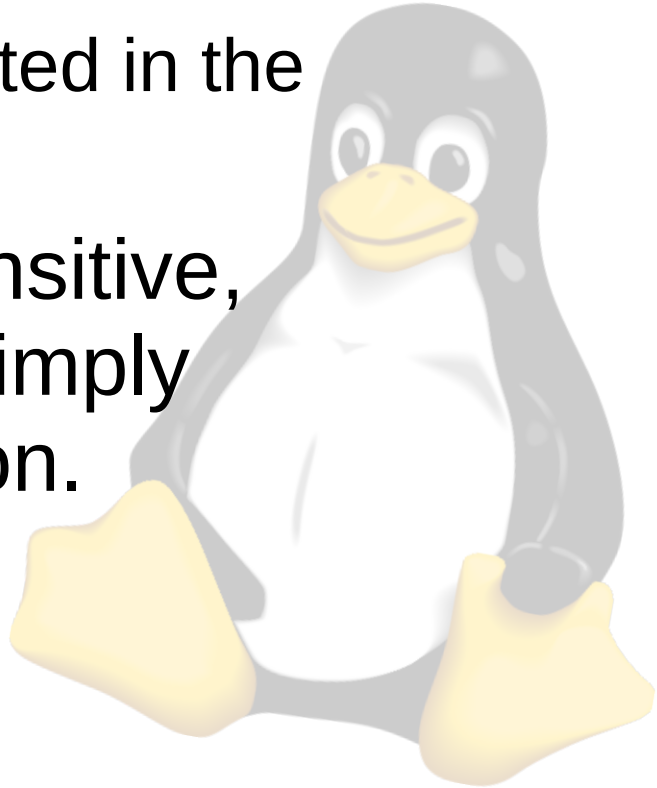
Lazy Allocation

- Process (cont)
  - In the page fault handler, the kernel uses its page tables to determine that the mapping is valid (from the kernel's point of view) yet unmapped in the TLB.
  - The kernel will allocate a physical page frame and update the TLB with the new mapping.
  - The kernel returns from the exception handler and the user space program can resume.

Konsulko Group

Lazy Allocation

- In a lazy allocation case, the user space program **never is aware** that the page fault happened.
  - The page fault can only be detected in the time that was lost to handle it.
- For processes that are time-sensitive, pages can be **pre-faulted**, or simply touched, at the start of execution.
  - Also see `mlock()` and `mlockall()` in this case.



Konsulko Group

Page Tables

- The entries in the TLB are a **limited resource**.
- Far more mappings can be made than can exist in the TLB at one time.
- The kernel must keep track of all of the mappings all of the time.
- The kernel stores all this information in the **page tables**.
  - See `struct_mm` and `vm_area_struct`

Konsulko Group

Page Tables

- Since the TLB can only hold a limited subset of the total mappings for a process, some valid mappings will not have TLB entries.
  - When these addresses are touched, the CPU will generate a page fault, because the CPU has no knowledge of the mapping; only the kernel does.

Konsulko Group

Page Tables

- When the page fault handler executes in this case, it will:
  - Find the appropriate mapping for the offending address in the kernel's page tables
  - Select and remove an existing TLB entry
  - Create a TLB entry for the page containing the address
  - Return to the user space process
    - *Observe the similarities to lazy allocation handling*

Konsulko Group

Swapping

- When memory utilization is high, the kernel may swap some frames to disk to free up RAM.
  - Having an MMU makes this possible.
- The kernel can copy a frame to disk and remove its TLB entry.
- The frame can be re-used by another process

Konsulko Group

Swapping

- When the frame is needed again, the CPU will generate a page fault (because the address is not in the TLB).
- The kernel can then, at page fault time:
  - Put the process to sleep
  - Copy the frame from the disk into an unused frame in RAM
  - Fix the page table entry
  - Wake the process

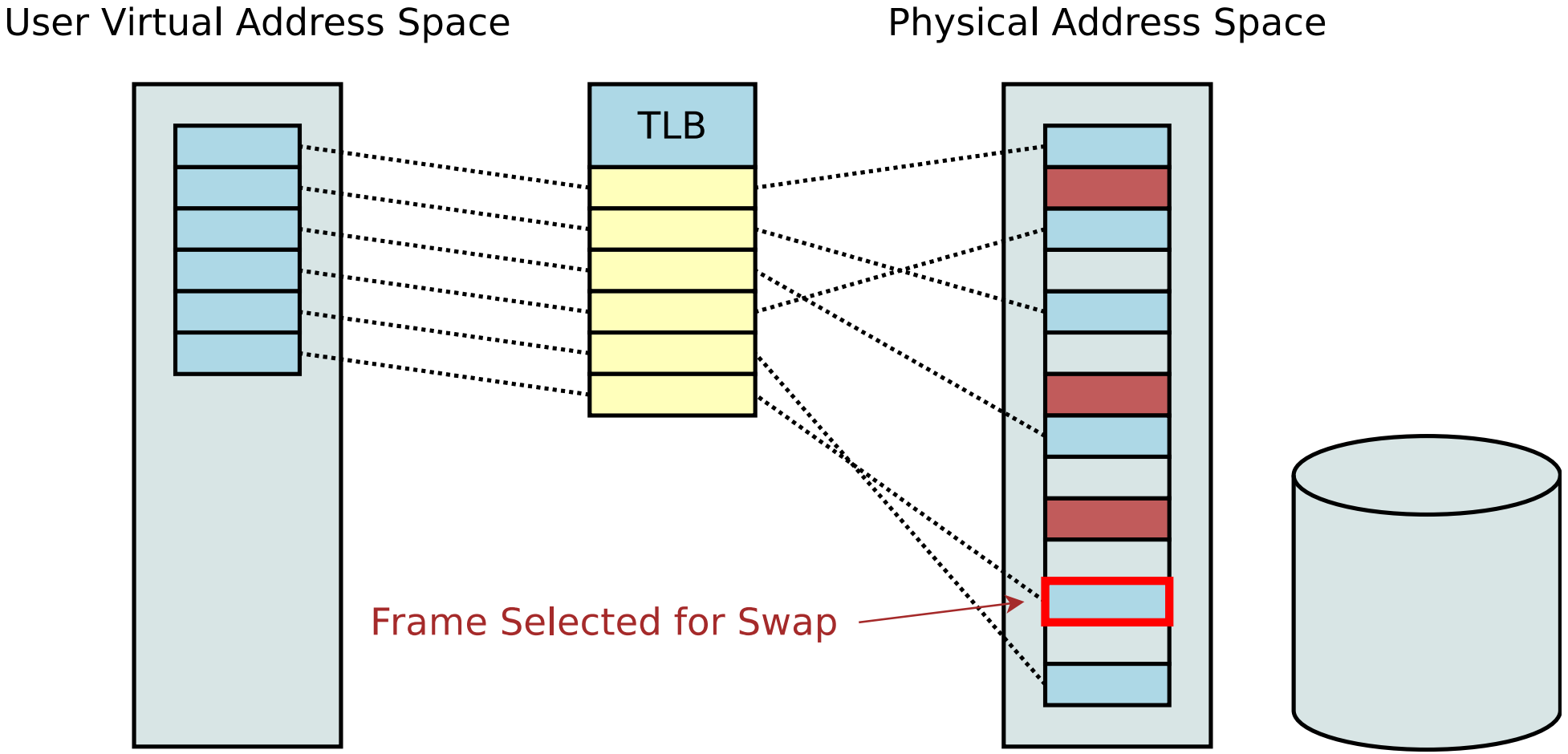
Konsulko Group

Swapping

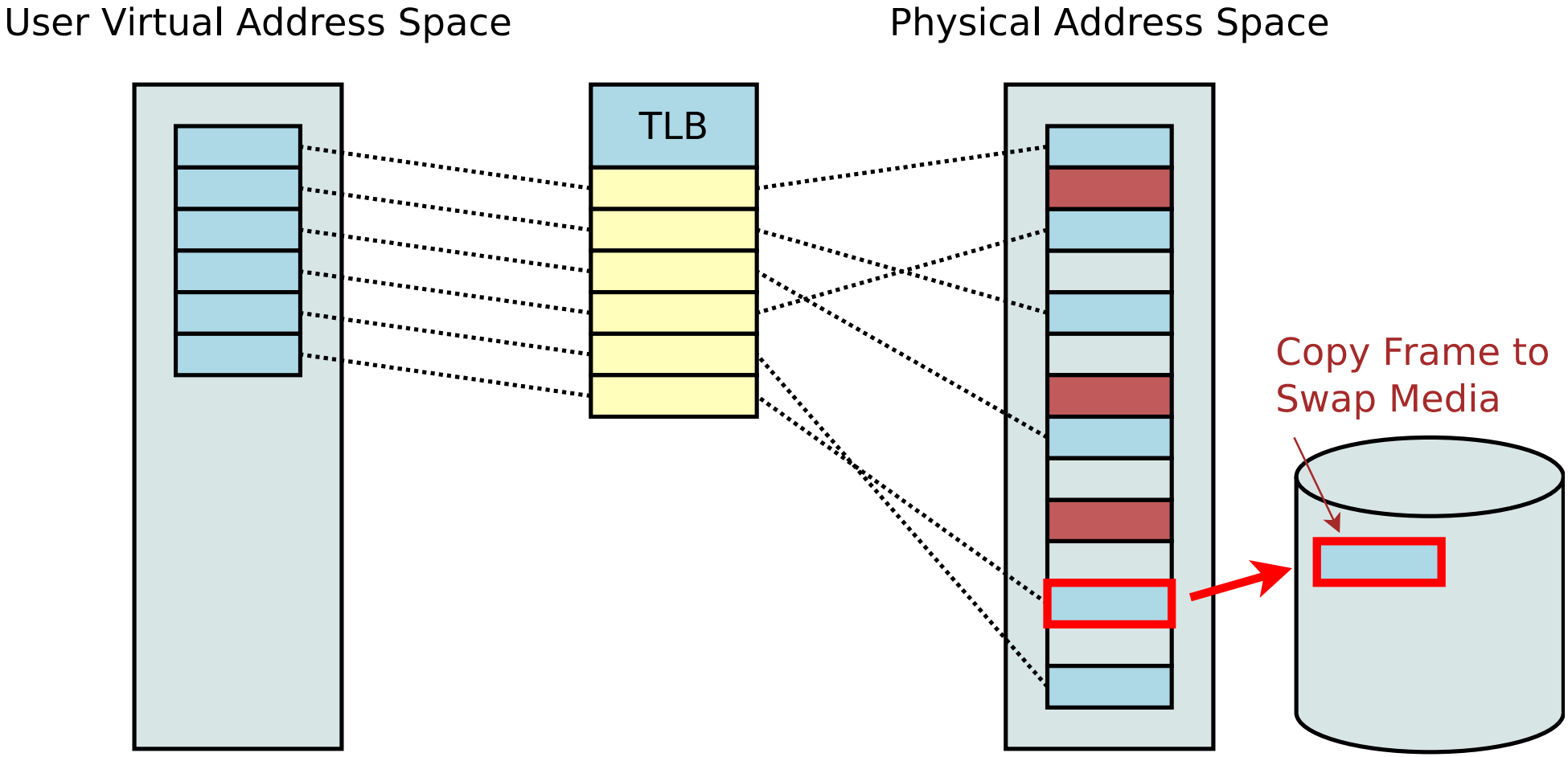
- Note that when the page is restored to RAM, it's not necessarily restored to the same physical frame where it originally was located.
- The MMU will use the same virtual address though, so the user space program will not know the difference
  - *This is why user space memory cannot typically be used for DMA.*

Konsulko Group

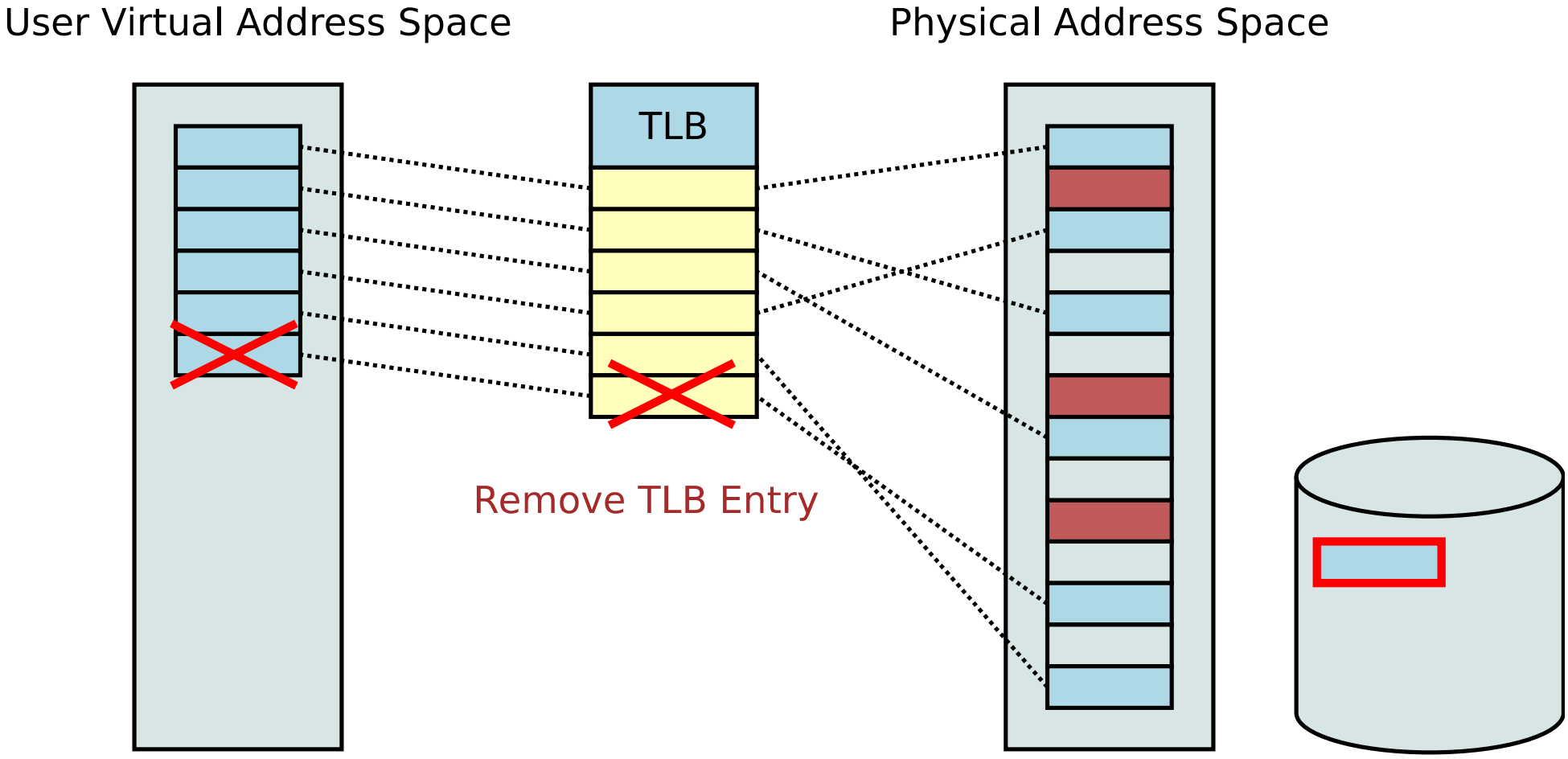
Swapping Out



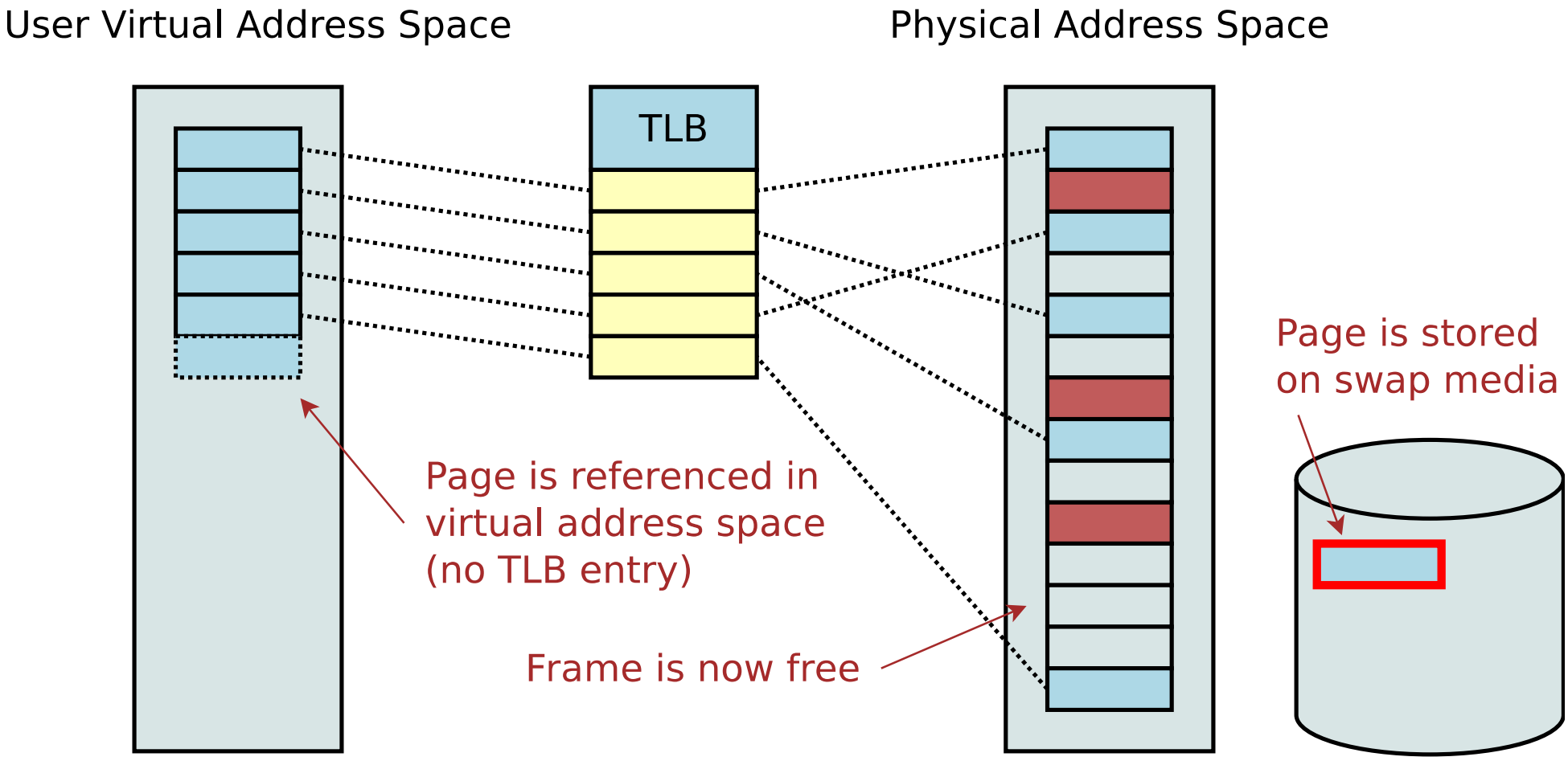
Swapping Out



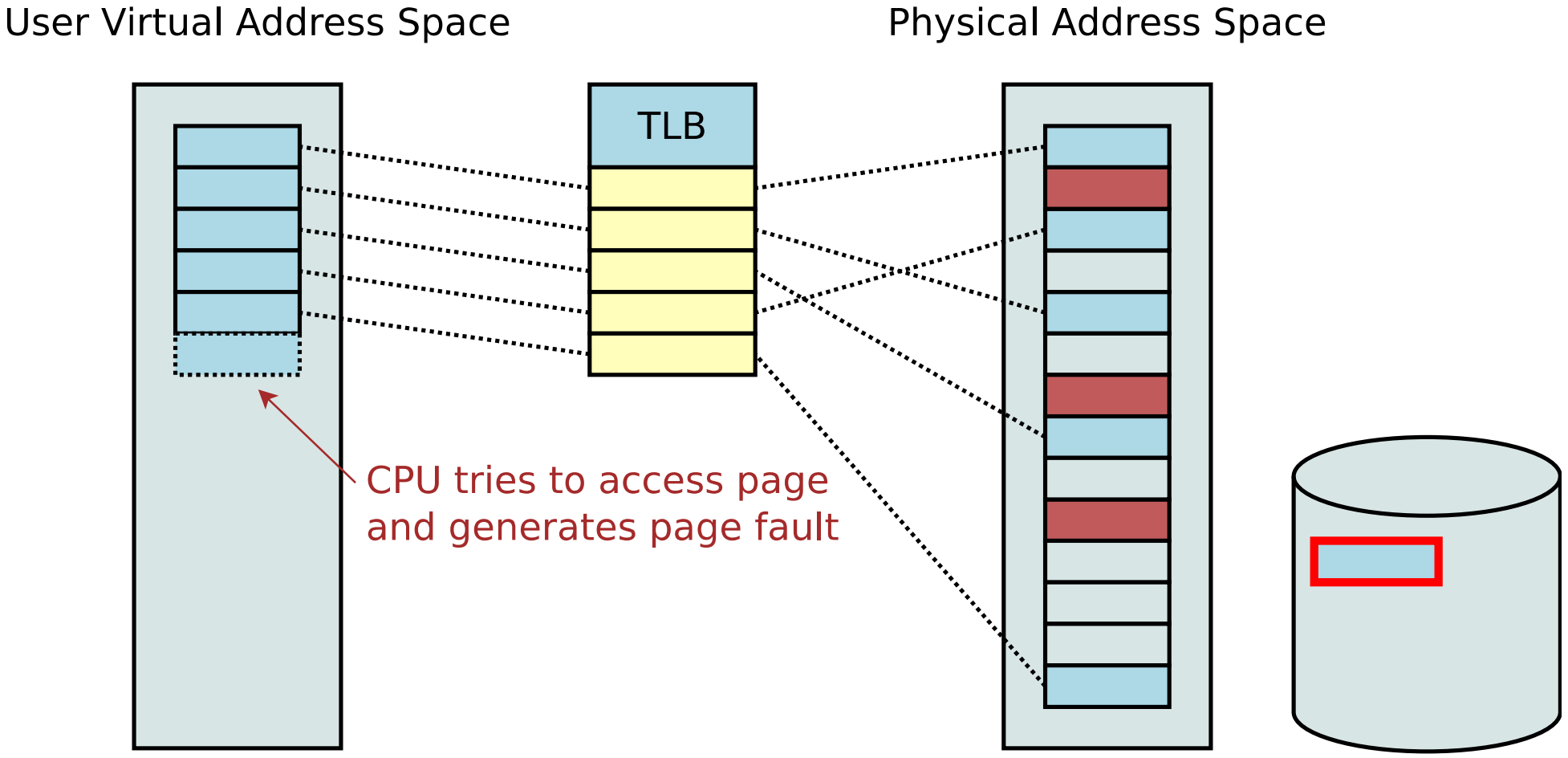
Swapping Out



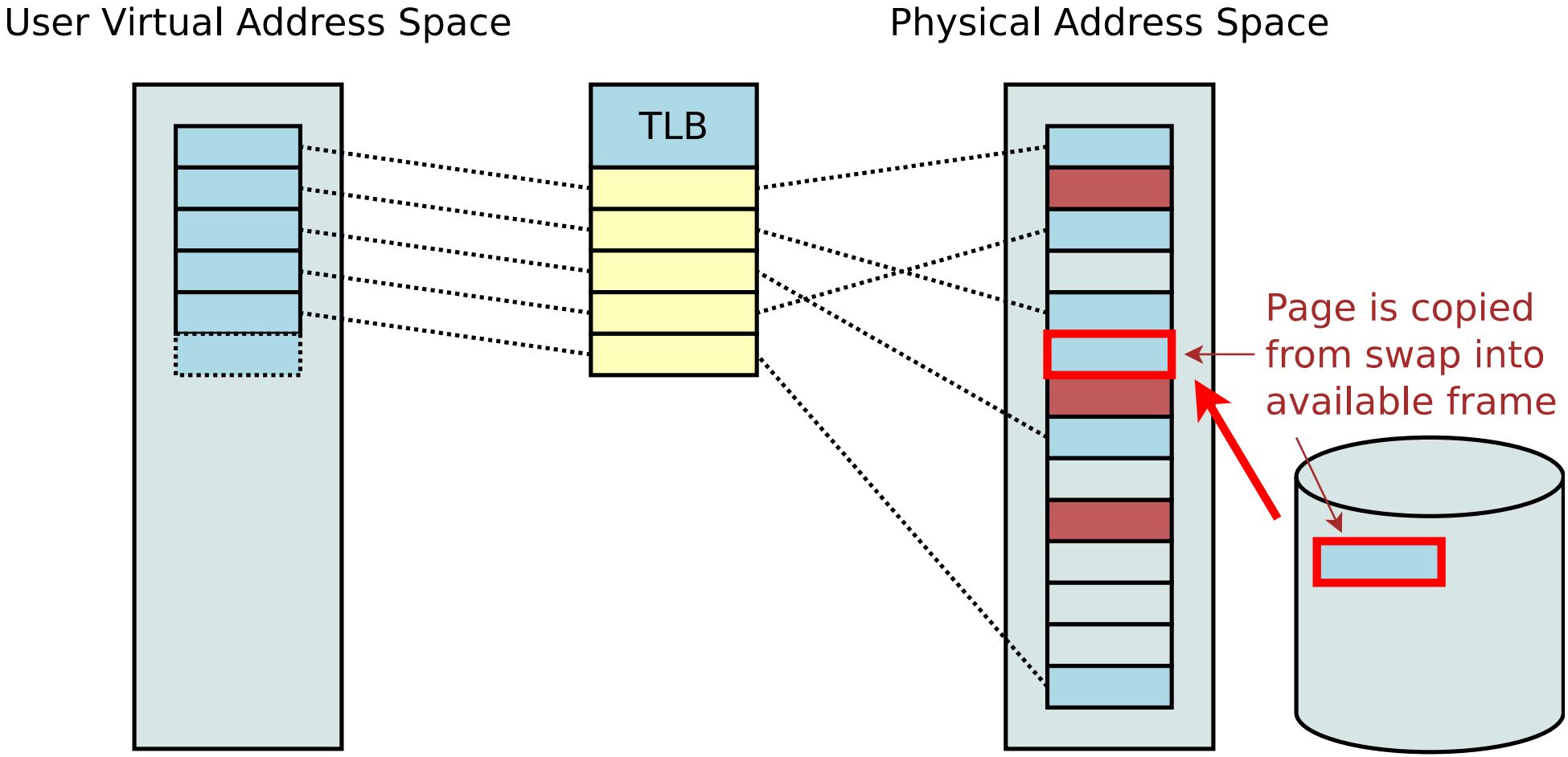
Swapping Out



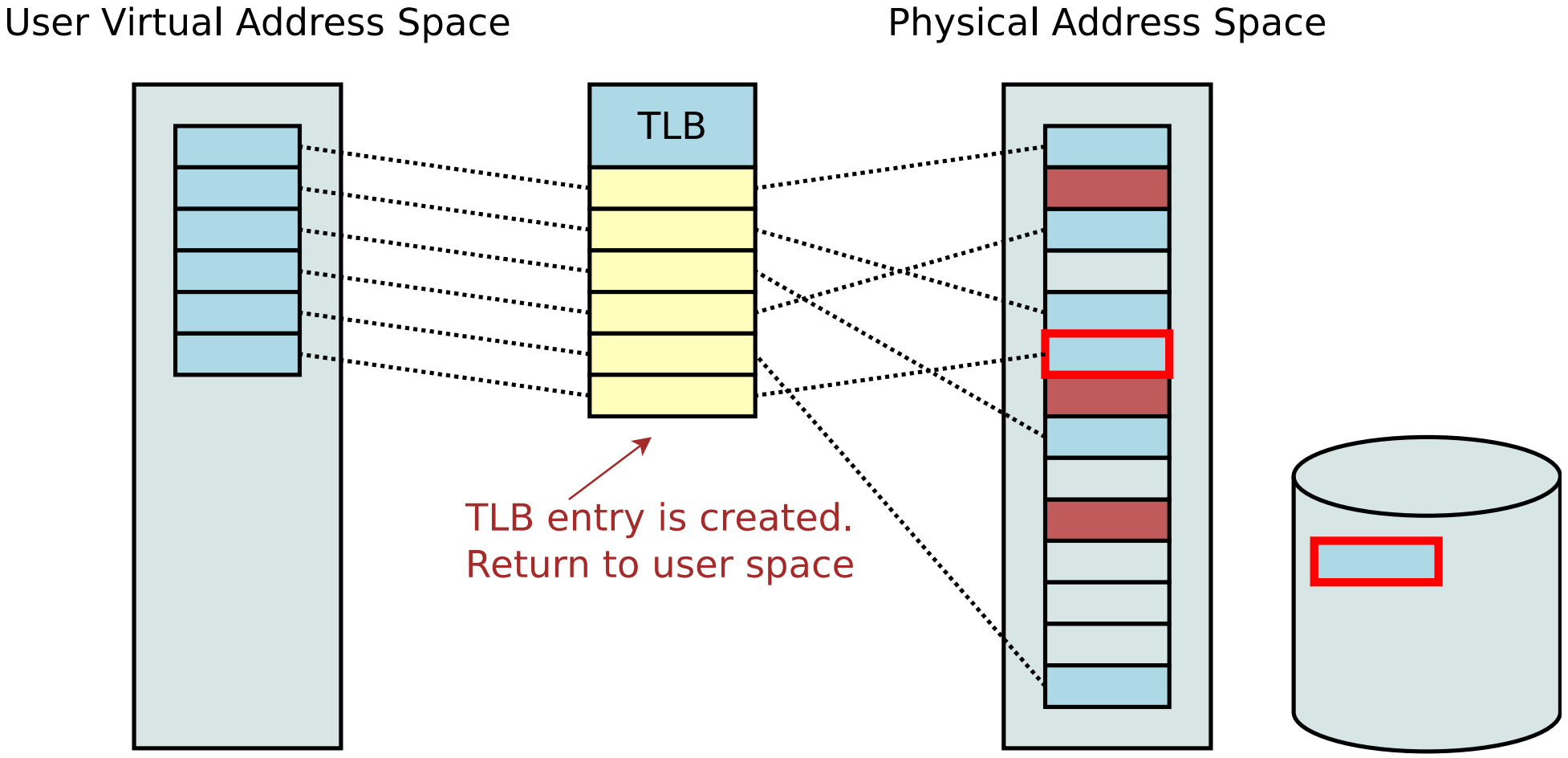
Swapping In


User Virtual Address Space
Physical Address Space
TLB
CPU tries to access page
and generates page fault

Swapping In


User Virtual Address Space
Physical Address Space
TLB
Page is copied
from swap into
available frame

Swapping In


User Virtual Address Space
Physical Address Space
TLB
TLB entry is created.
Return to user space

User Space

Konsulko Group

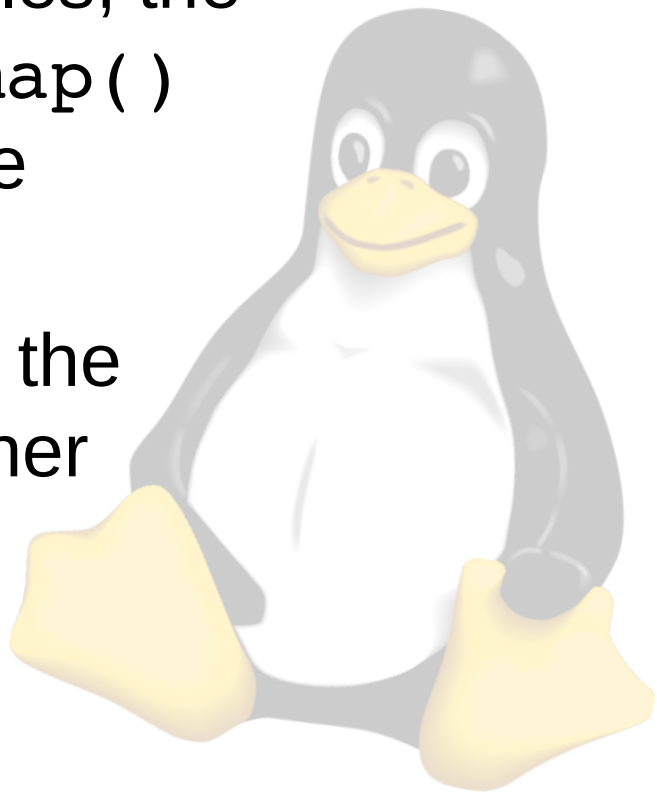
User Space

- There are several ways to allocate memory from user space
  - Ignoring the familiar *alloc() functions, which sit on top of platform methods.
- `mmap()` can be used directly to allocate and map pages.
- `brk()`/`sbrk()` can be used to increase the heap size.

Konsulko Group

mmap()

- mmap() is the standard way to allocate large amounts of memory from user space
  - While mmap() is often used for files, the MAP_ANONYMOUS flag causes mmap() to allocate normal memory for the process.
  - The MAP_SHARED flag can make the allocated pages sharable with other processes.



Konsulko Group

`brk()`/`sbrk()`

- `brk()` sets the top of the **program break**.
  - The man page says this is the top of the data segment, but inspection of `kernel/sys.c` shows it separate from the data segment.
  - This in effect increases the size of the heap.
- `sbrk()` increases the program break (rather than setting it directly).

Konsulko Group

`brk()/sbrk()`

- Lazy Allocation
  - See `mm/mmap.c` for `do_brk()`
  - `do_brk()` is implemented similar to `mmap()`.
    - Modify the page tables for the new area
    - Wait for the page fault
    - *Optionally, `do_brk()` can pre-fault the new area and allocate it. See `mlock(2)` to control this behavior.*

Konsulko Group

High-Level Implementation

- The familiar C allocators `malloc()` and `calloc()` will use either `brk()` or `mmap()` depending on the requested allocation size.
  - Small allocations use `brk()`
  - Large allocations use `mmap()`
  - See `mallopt(3)` and the `M_MMAP_THRESHOLD` parameter to control this behavior.

Konsulko Group

Stack

- Stack Expansion
  - If a process accesses memory beyond its stack, the CPU will trigger a page fault.
  - The page fault handler detects the address is just beyond the stack, and allocates a new page to extend the stack.
    - The new page will not be physically contiguous with the rest of the stack.
  - See `__do_page_fault()` in `arch/arm/mm/fault.c`

Konsulko Group

Summary

- Physical Memory
- Virtual Memory
- Kernel addressing
- User space addressing
- Swapping
- User space allocation

Konsulko Group