Memory Management

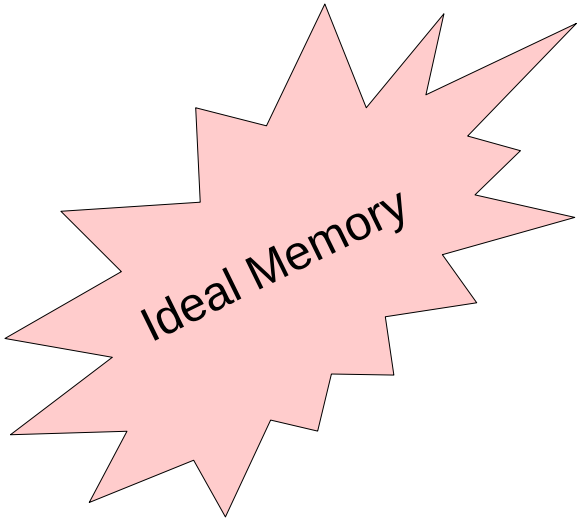
Andre M. Maier, DHBW Ravensburg
dhbw@andre-maier.com

DHBW
Duale Hochschule
Baden-Württemberg

Andre M. Maier
Dept. of Computer Science

Memory Requirements

- We want computer memory to be
  - fast
  - dense
  - cheap
  - energy-saving



- **In the physical world, these requirements are mutually exclusive!**

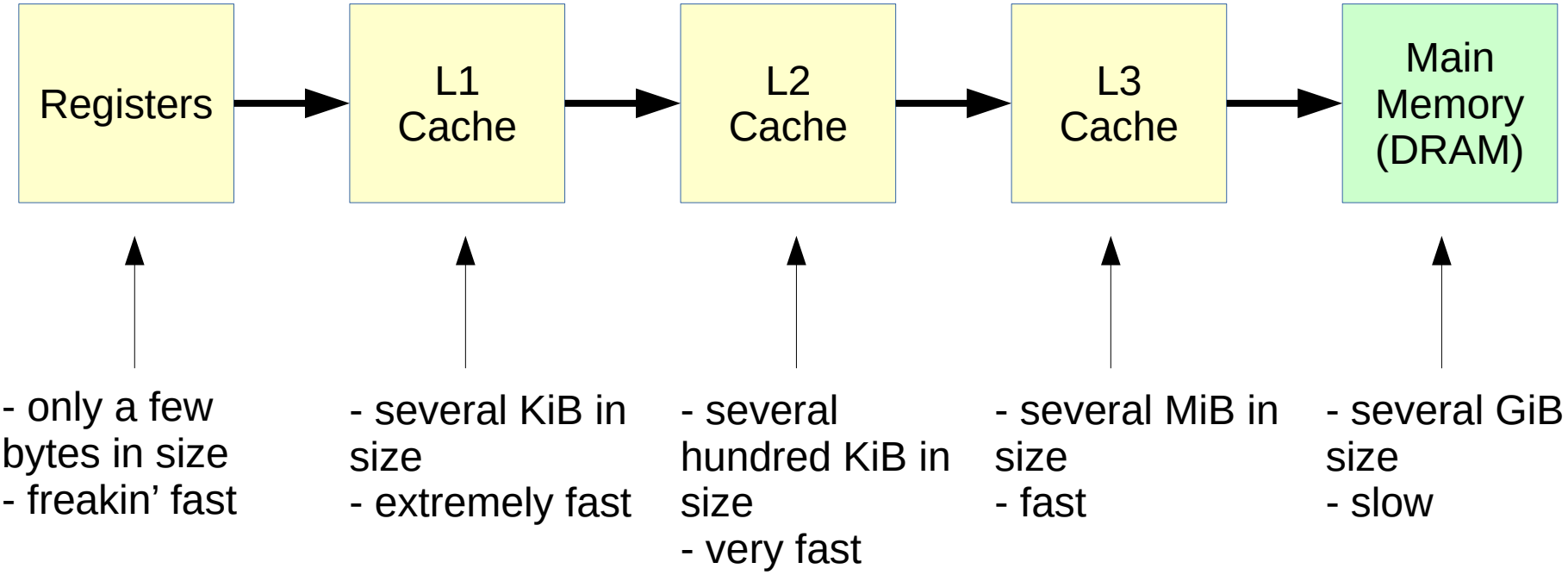
Andre M. Maier
Dept. of Computer Science

Solution

- We need to make the processor “believe” that it has access to ideal memory.
- Hierarchy of different types of memory, from
  - fast to slow
  - small to large
  - expensive to cheap
  - energy-consuming to energy-saving
- Logic decides where particular data will be stored

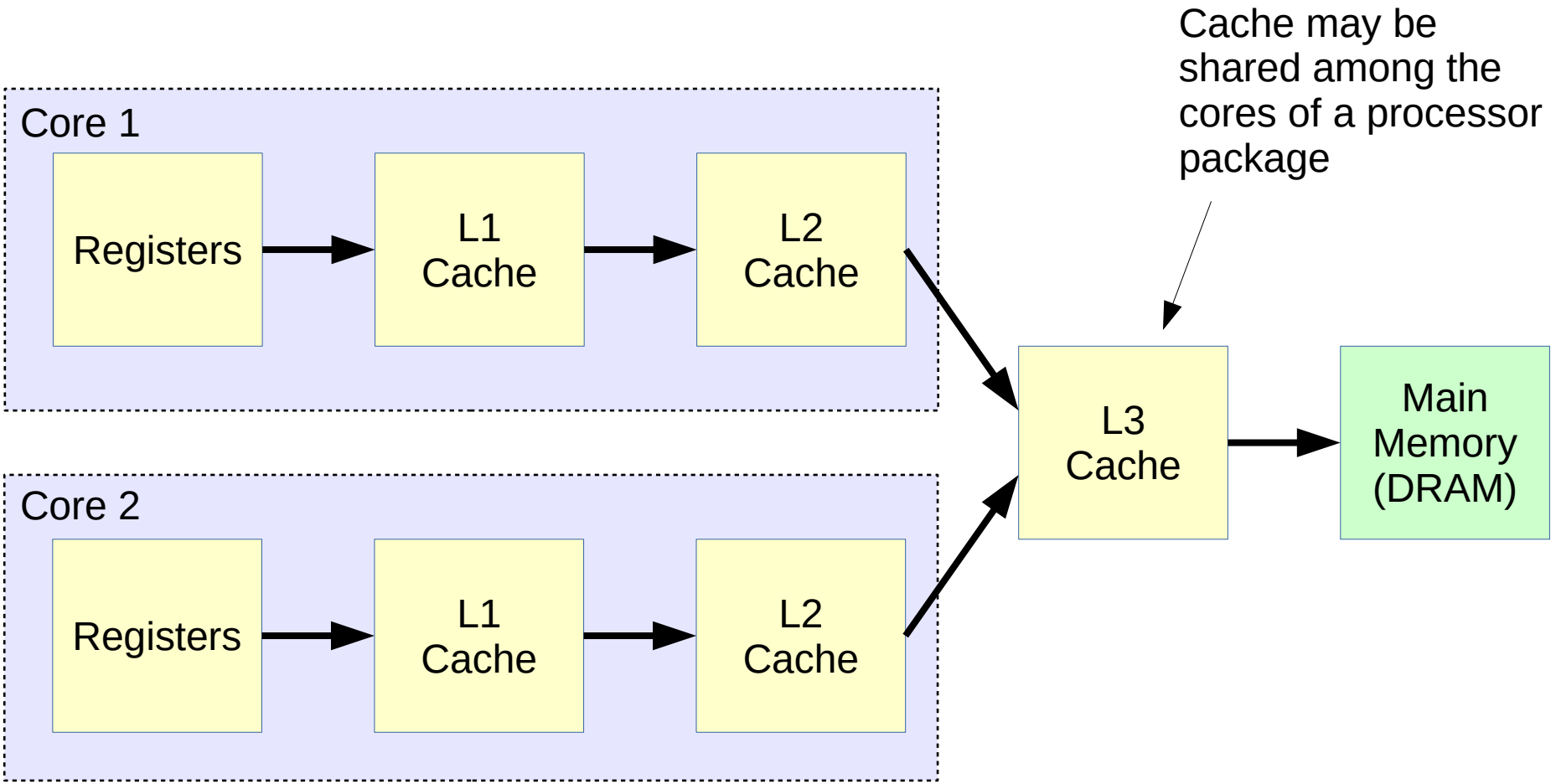
Andre M. Maier
Dept. of Computer Science

Hierarchy



Andre M. Maier
Dept. of Computer Science

Cache in Multicore-Systems (Example)



Andre M. Maier
Dept. of Computer Science

Lab Exercises

- Identify the type of CPU in your computer.
- What is the size of its L1, L2, L3 caches?
- Are there any shared cache levels?

Andre M. Maier
Dept. of Computer Science

Memory Allocation for Processes

- **Static** Memory Allocation
  - Low layers of the operating system can be statically assigned to a fixed location in memory.
- **Dynamic** Memory Allocation
  - Higher layers of the operating system and applications have a limited life span. Thus, they require dynamic allocation of memory.

Andre M. Maier
Dept. of Computer Science

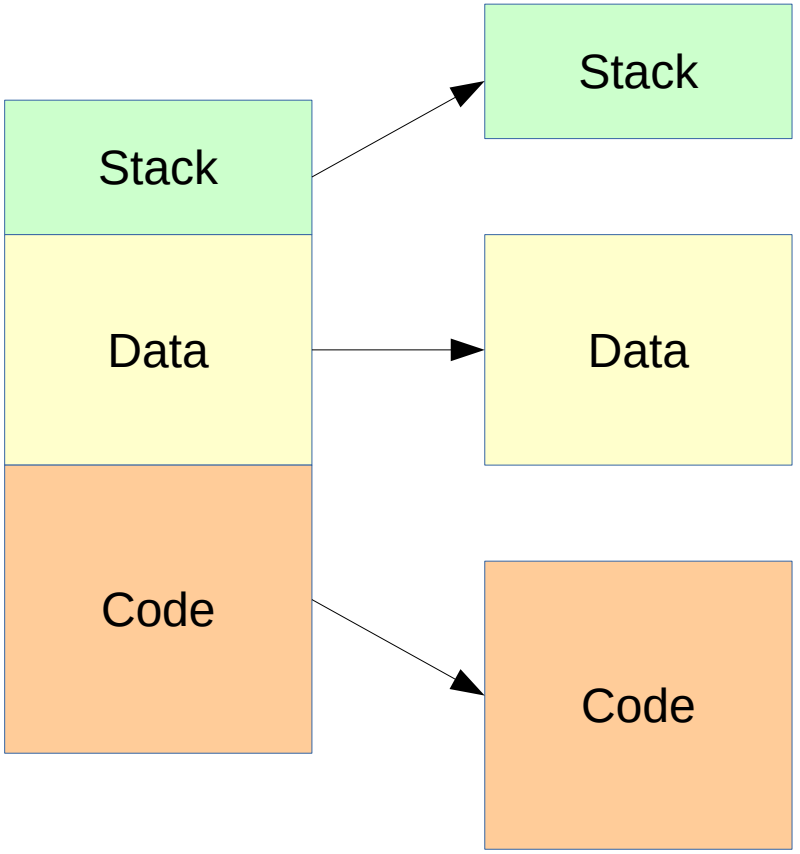
Dynamic Memory Allocation

- Dynamic memory allocation requires memory management by the operating system.
  - Implemented in *allocate()* and *free()*
- Two types of dynamic memory management
  - Segmentation without paging
  - Segmentation with paging

Andre M. Maier
Dept. of Computer Science

Simple Memory Segmentation (1)

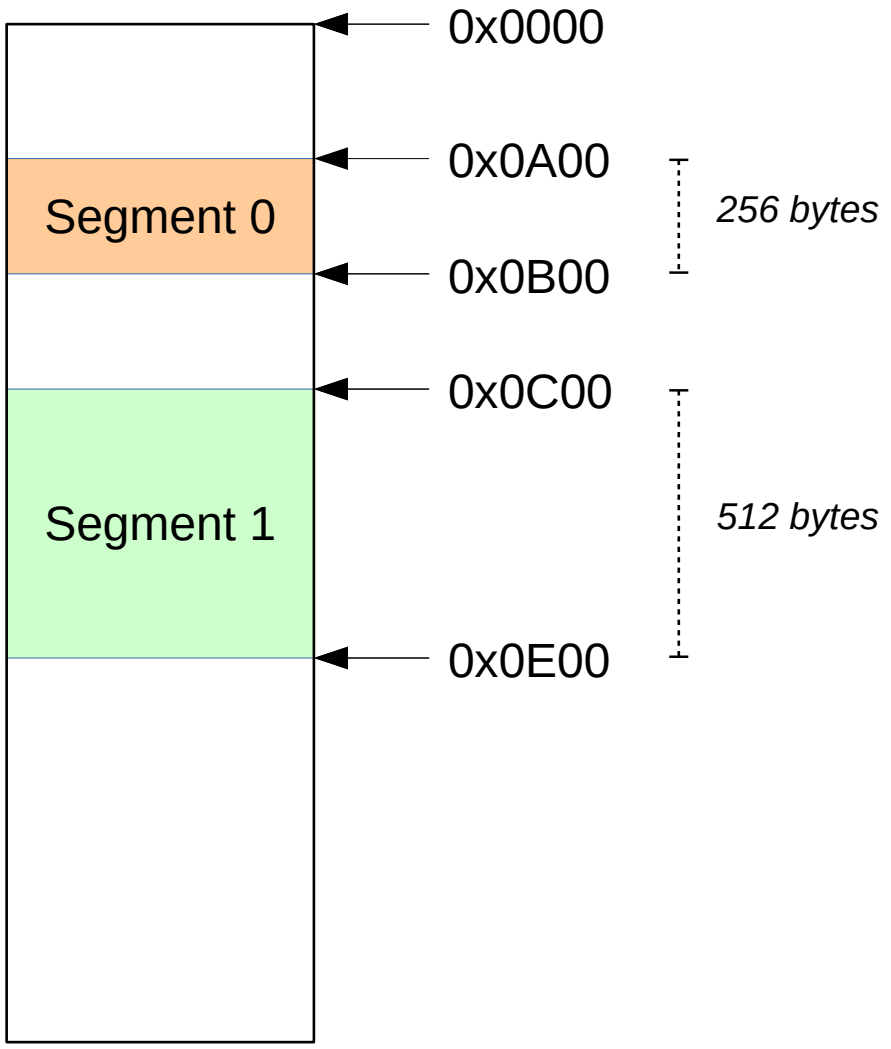
- Single process is split into at least three segments
- Process no longer needs to be contiguous
- Segmentation is visible to the programmer (x86 assembly: *.stack*)



Andre M. Maier
Dept. of Computer Science

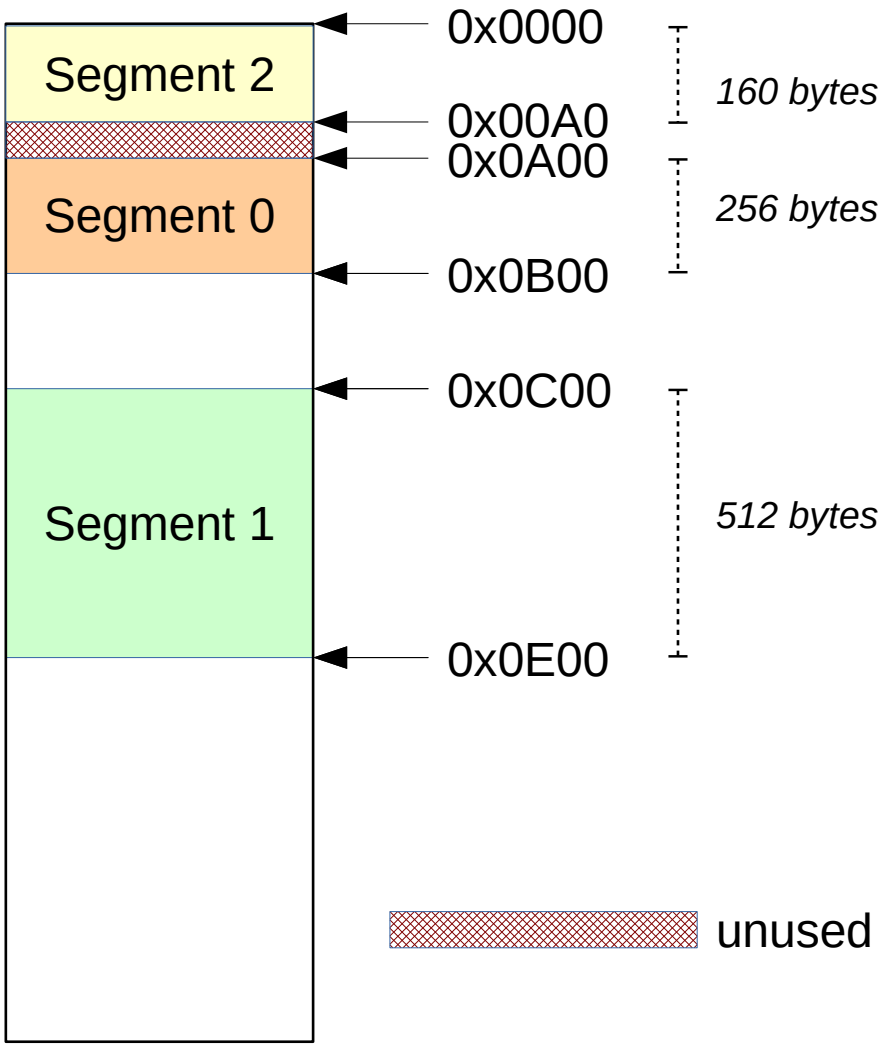
Simple Memory Segmentation (2)

- Segments can be of arbitrary size.
- The location where a segment is stored in physical memory is determined by a *placement algorithm*.



Andre M. Maier
Dept. of Computer Science

Simple Memory Segmentation (3)



- Size of a segment matches the size of the memory request.
- A requested segment of arbitrary size may not fit precisely into any region of empty memory.

Andre M. Maier
Dept. of Computer Science

Simple Segmentation (4)

- There will be **no** *internal fragmentation* ...
  - because the exact size of the segment is requested
- But *external fragmentation* may occur ...
  - because areas of unused memory too small to hold new segments are inevitable

Andre M. Maier
Dept. of Computer Science

Simple Segmentation (5)

- The OS maintains a segment table to keep track of where the segments are stored.

Segment No.	Base Address	Length (bytes)
0	0x0A00	160
1	0x0C00	256
2	0x0000	512

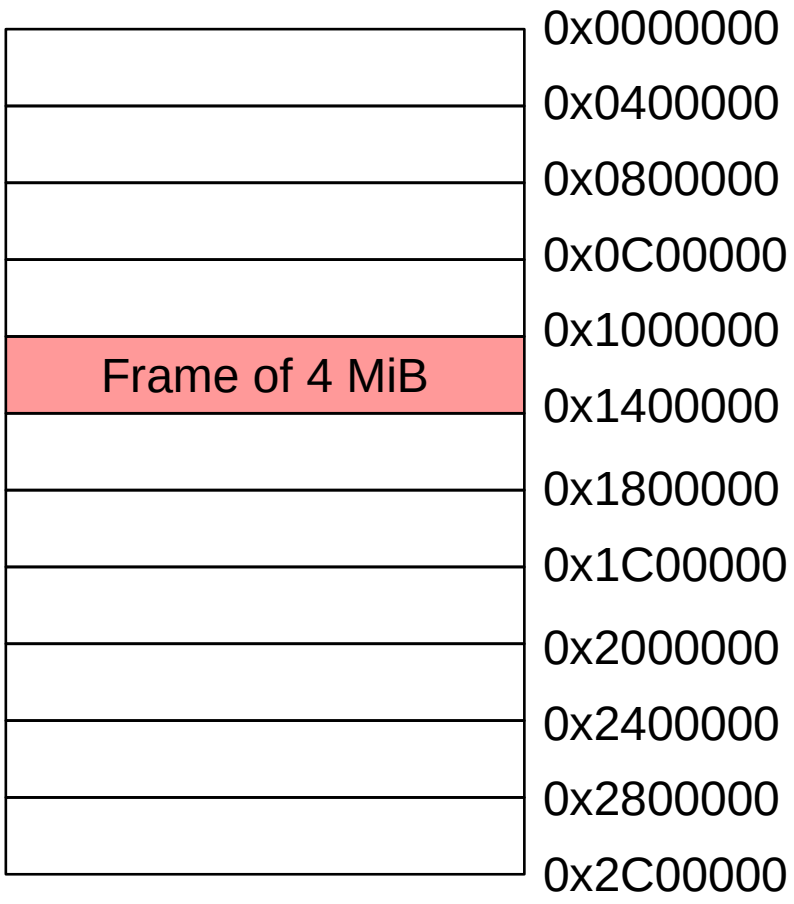
Andre M. Maier
Dept. of Computer Science

Simple Segmentation (6)

- Logical addresses consist of two components:
  - Segment number
  - Offset
- Physical address = base address + offset
- OS needs to verify the offset is less than the length of the segment!

Andre M. Maier
Dept. of Computer Science

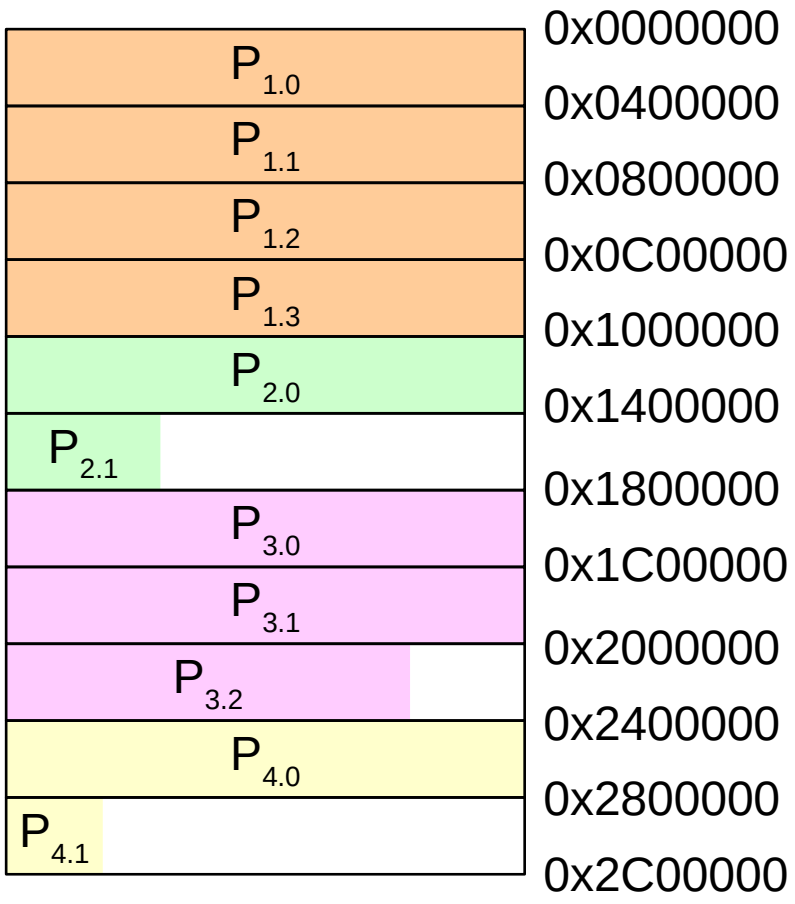
Simple Paging (1)



- Physical Memory is divided into fixed-sized chunks called **frames**.
- Frame size is a power of 2, usually between 512 bytes and 16 MiB

Andre M. Maier
Dept. of Computer Science

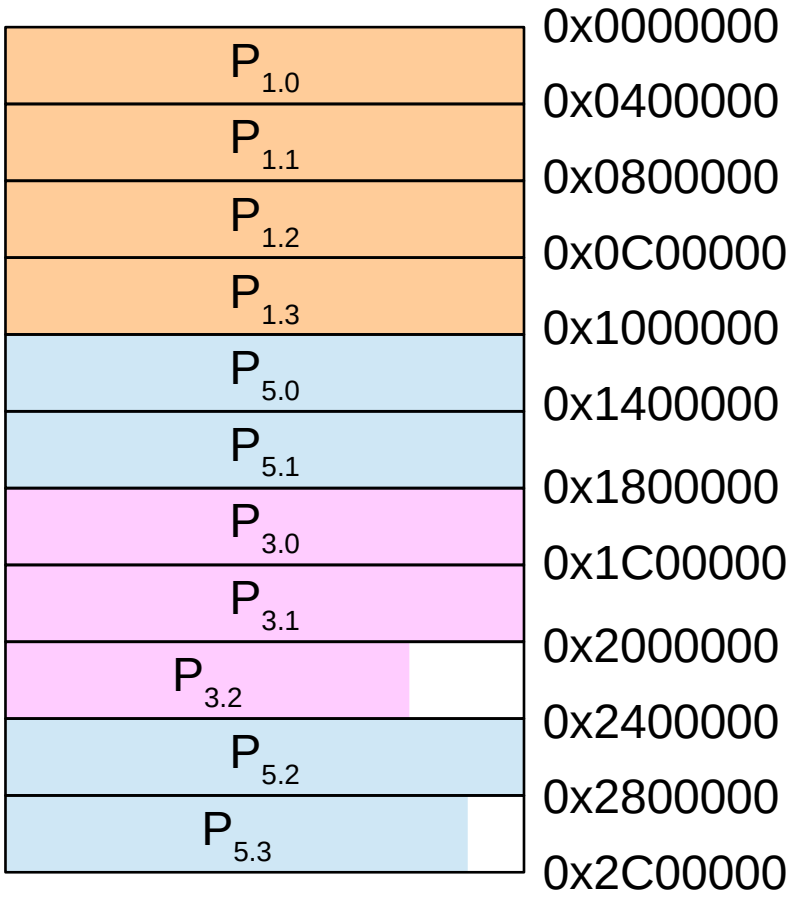
Simple Paging (2)



- Logical memory is divided into blocks of the same size called **pages**.
- When a process is executed, its pages are loaded into any available memory frames.
- Only whole frames can be assigned to a process.
  → Internal fragmentation

Andre M. Maier
Dept. of Computer Science

Simple Paging (3)



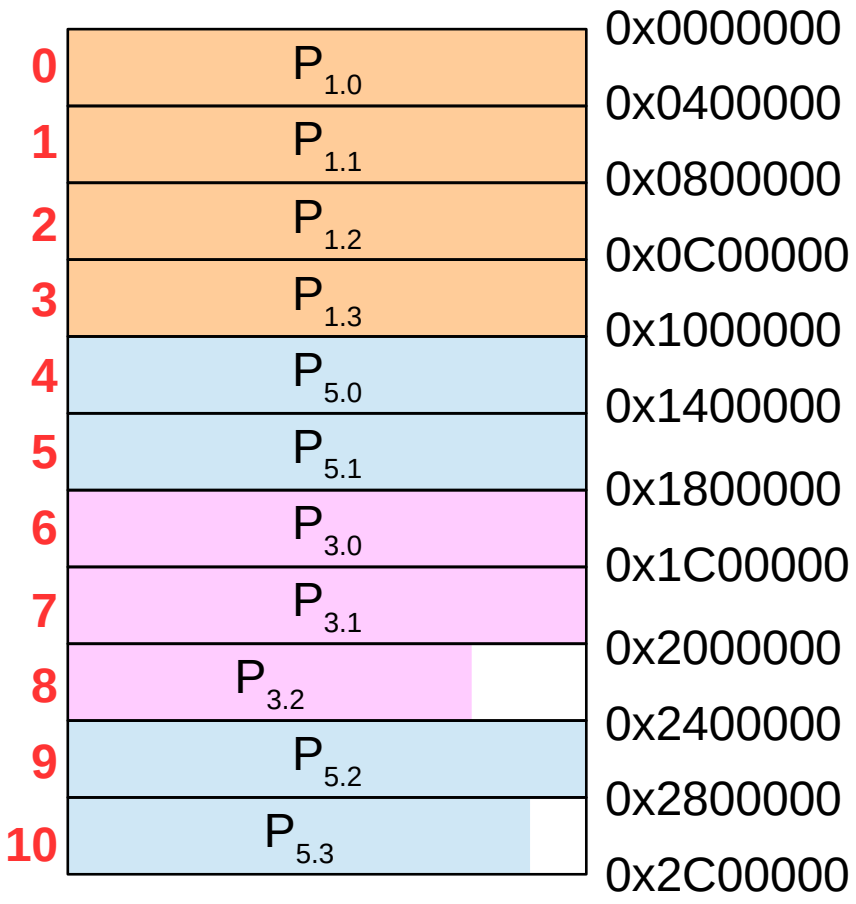
- When processes are released, non-contiguous frames of free memory will occur
- Thus, the address space is non-contiguous

Simple Paging (4)

- A **page table** is used to keep track of where the pages of a specific process are located.
- There is one page table for each process.
- The page table is also stored in memory.

Andre M. Maier
Dept. of Computer Science

Simple Paging (5)



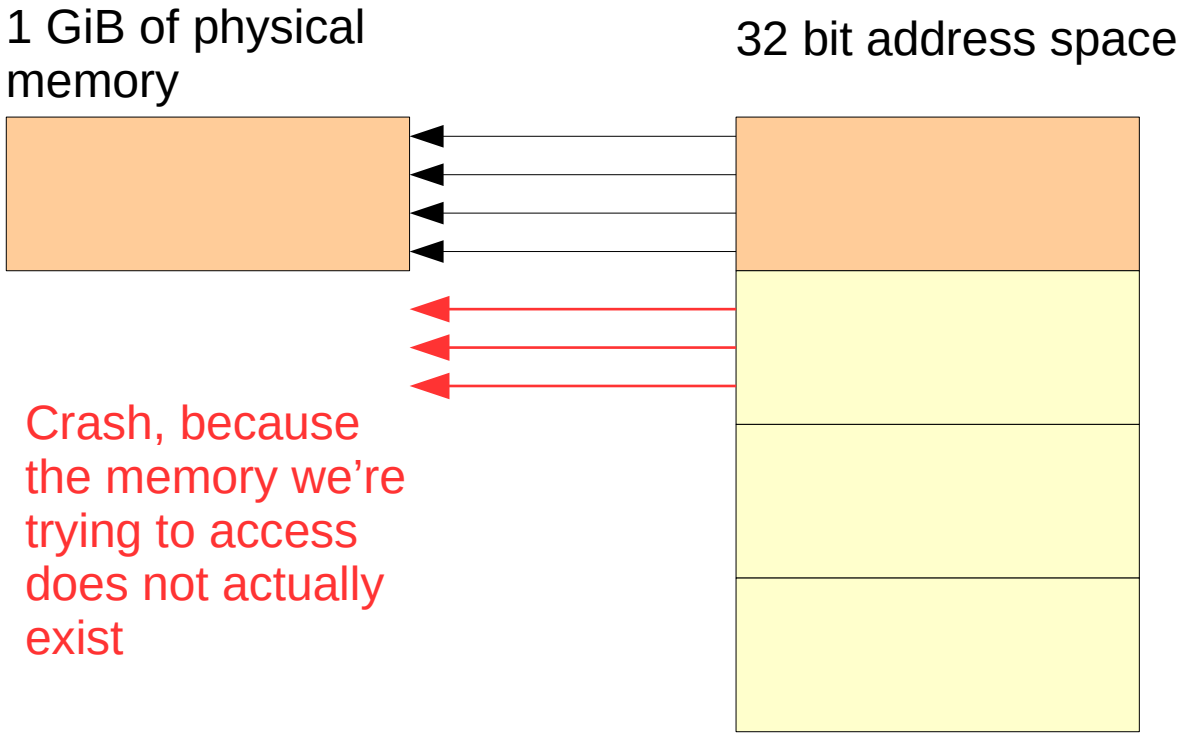
Page table of process P_5 :

Page No.	Frame No.
0	4
1	5
2	9
3	10

Andre M. Maier
Dept. of Computer Science

Why Virtual Memory? (1)

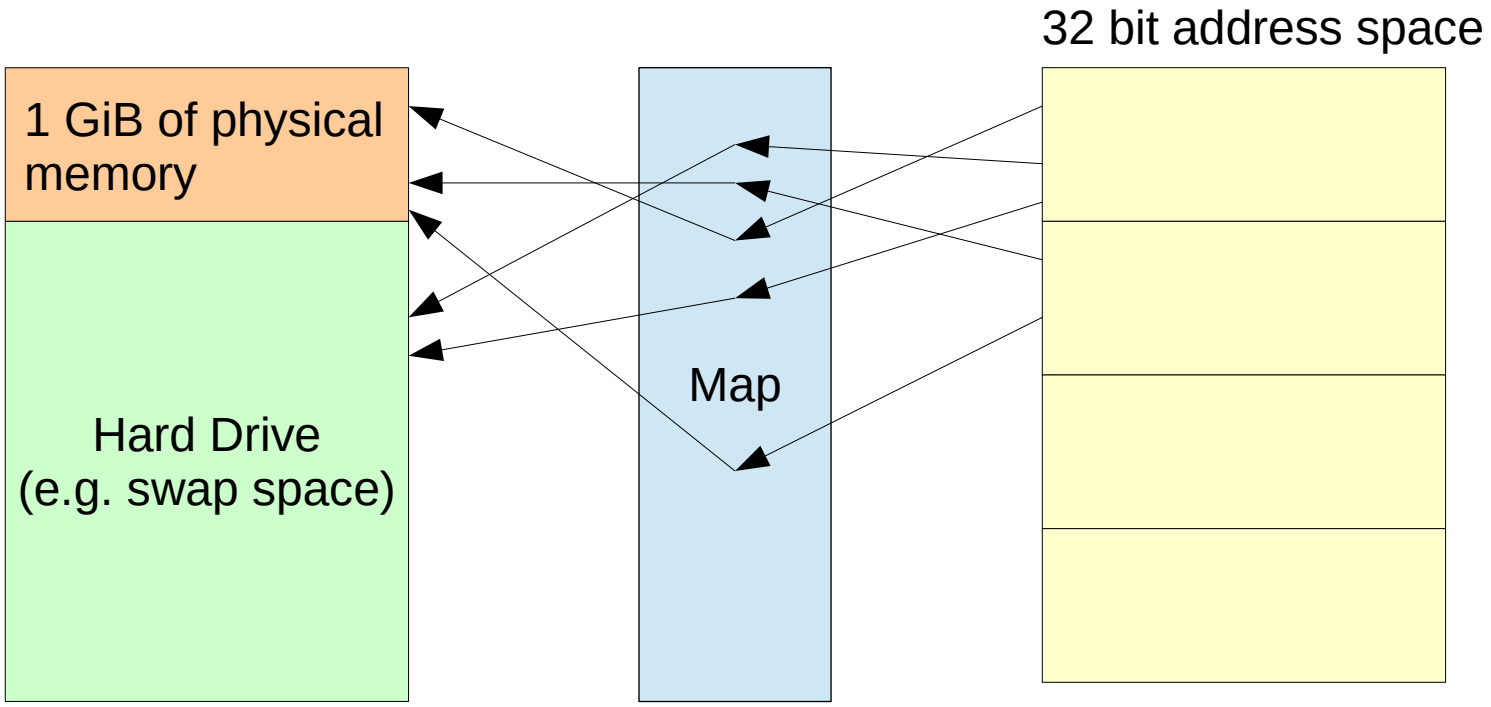
- Address space may not match the available physical memory, e.g.



Andre M. Maier
Dept. of Computer Science

Why Virtual Memory (2)

- Virtual Memory provides more flexibility by abstraction / indirection



Andre M. Maier
Dept. of Computer Science

Why Virtual Memory (3)

- Parts of the address space can be mapped to disk space.
- Memory contents that are not currently accessed can be written to disk and retrieved back to RAM when necessary.
- “Unlimited Memory”

Andre M. Maier
Dept. of Computer Science

Why Virtual Memory (4)

- Further advantages
  - Security
    - Each program has its own page table.
    - Processes cannot access directly another process' physical RAM.
  - Contiguous address space per program/process
    - Program's address space can be contiguous while actual data in physical memory may be fragmented.
    - The operating system, not the programmer, is in charge of memory management.

Andre M. Maier
Dept. of Computer Science

Downsides of Virtual Memory

- Program performance
  - Hard drives are significantly slower than RAM
  - Buying more (physical) RAM can mitigate this effect
- Memory isolation
  - More difficult to share data between programs
  - Mapping mechanism and OS have to provide additional functionality

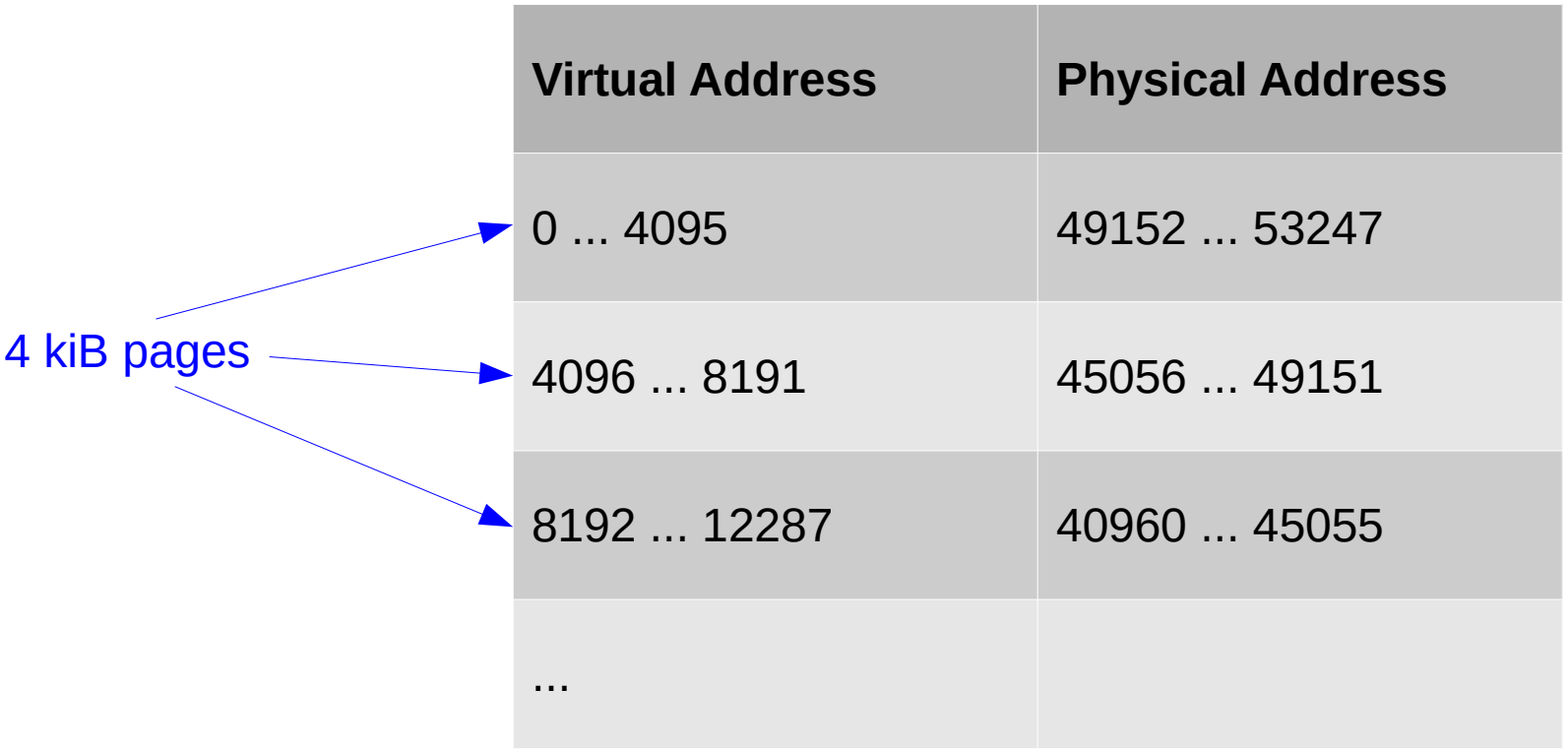
Andre M. Maier
Dept. of Computer Science

Page Table (1)

- Problem: The map that is used to convert virtual addresses into physical addresses would need to have one entry for each virtual address.
  - With an virtual address space of 32 bit this would be $2^{32} \approx 4$ billion entries!
- Solution: Pages!
- The map is therefore called a **page table**.

Andre M. Maier
Dept. of Computer Science

Page Table (2)

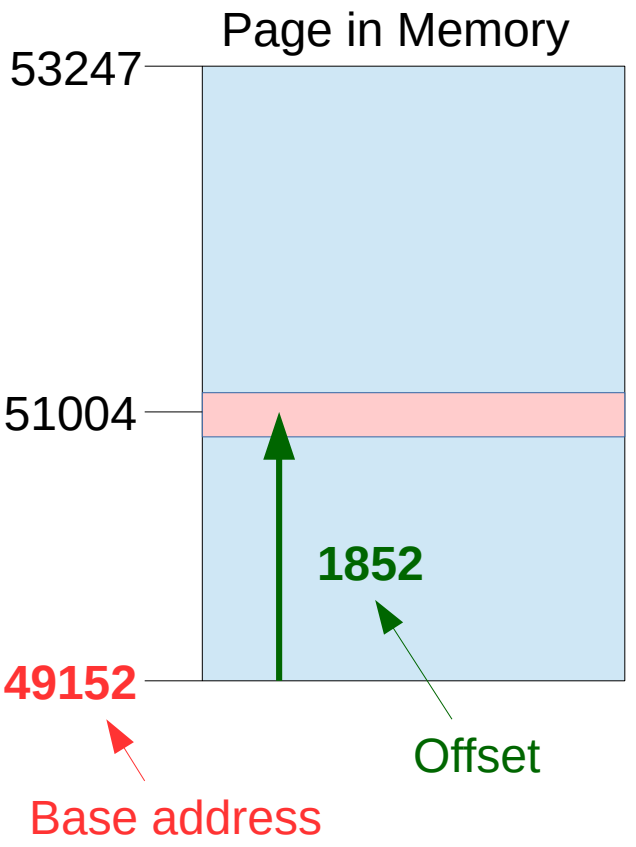


Virtual Address	Physical Address
0 ... 4095	49152 ... 53247
4096 ... 8191	45056 ... 49151
8192 ... 12287	40960 ... 45055
...	

Andre M. Maier
Dept. of Computer Science

Address Translation (1)

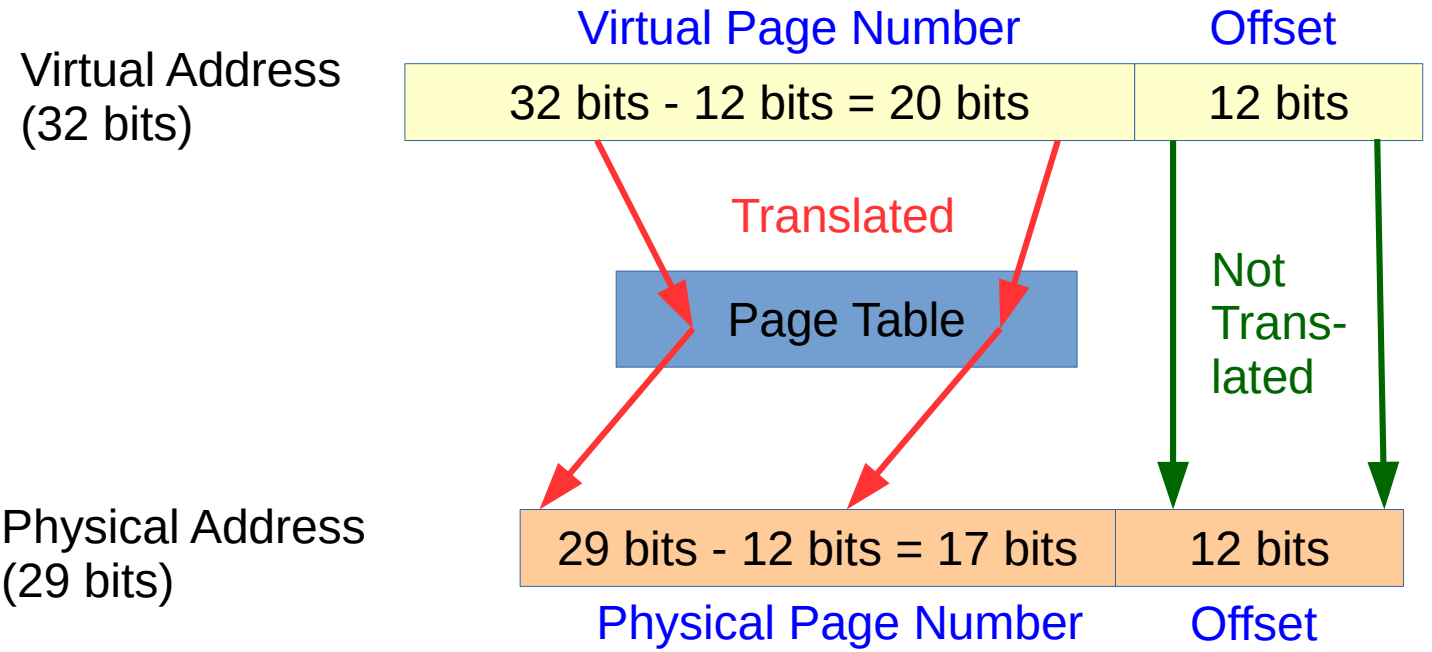
- A page table entry cannot (and does not) contain address ranges.
- In order to access memory within a page, we need to know the page's base address and an offset.
- We can use the same offset value in both virtual and physical memory.



Andre M. Maier
Dept. of Computer Science

Address Translation (2)

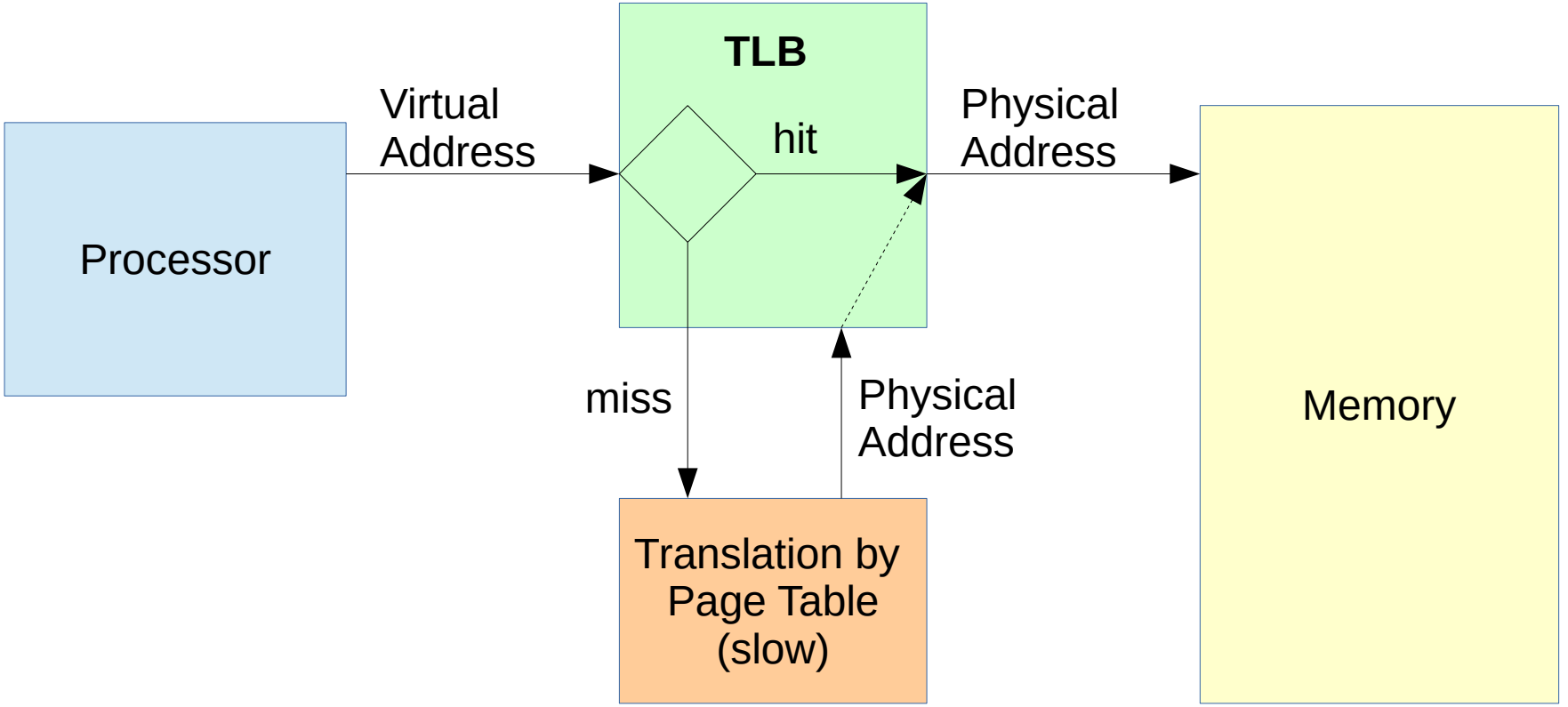
Example: 32 bits VM address space, 4kiB page size, 512 MiB physical memory



Andre M. Maier
Dept. of Computer Science

Translation Lookaside Buffer (1)

- The Translation Lookaside Buffer (TLB) is a cache for page table entries.



Andre M. Maier
Dept. of Computer Science

Translation Lookaside Buffer (2)

- The TLB speeds up address translation.
- The smaller a cache, the faster it will be.
  - Separate TLBs for data (DTLB) and instructions (ITLB)
  - Subdivided into Translation Registers (DTR,ITR) and Translation Caches (DTC,ITC) in modern operating systems
- Typically, a TLB has 64 entries (4kiB page size) or 32 entries (2MiB page size)

Andre M. Maier
Dept. of Computer Science

Swapping (1)

- If the page to be accessed does not reside in physical RAM but on hard disk ...
  - the page table contains an entry that indicates the page is to be found on disk
  - a **page fault** exception is triggered
  - the operating system's page fault handler takes over to handle the problem
  - whatever nifty algorithm may be in place, it is now going to take a painfully long time to access the page

Swapping (2)

- The operating system's page fault handler first needs to free some RAM for the page to be retrieved from disk
- A **page replacement algorithm** determines which of the pages that currently reside in RAM will be swapped out. If the content of the page has been altered since it was last loaded into RAM, it is considered **dirty**. Dirty pages need to be written back to disk.
- The page to be retrieved is then loaded into RAM, and its page table entry is updated accordingly.

Andre M. Maier
Dept. of Computer Science

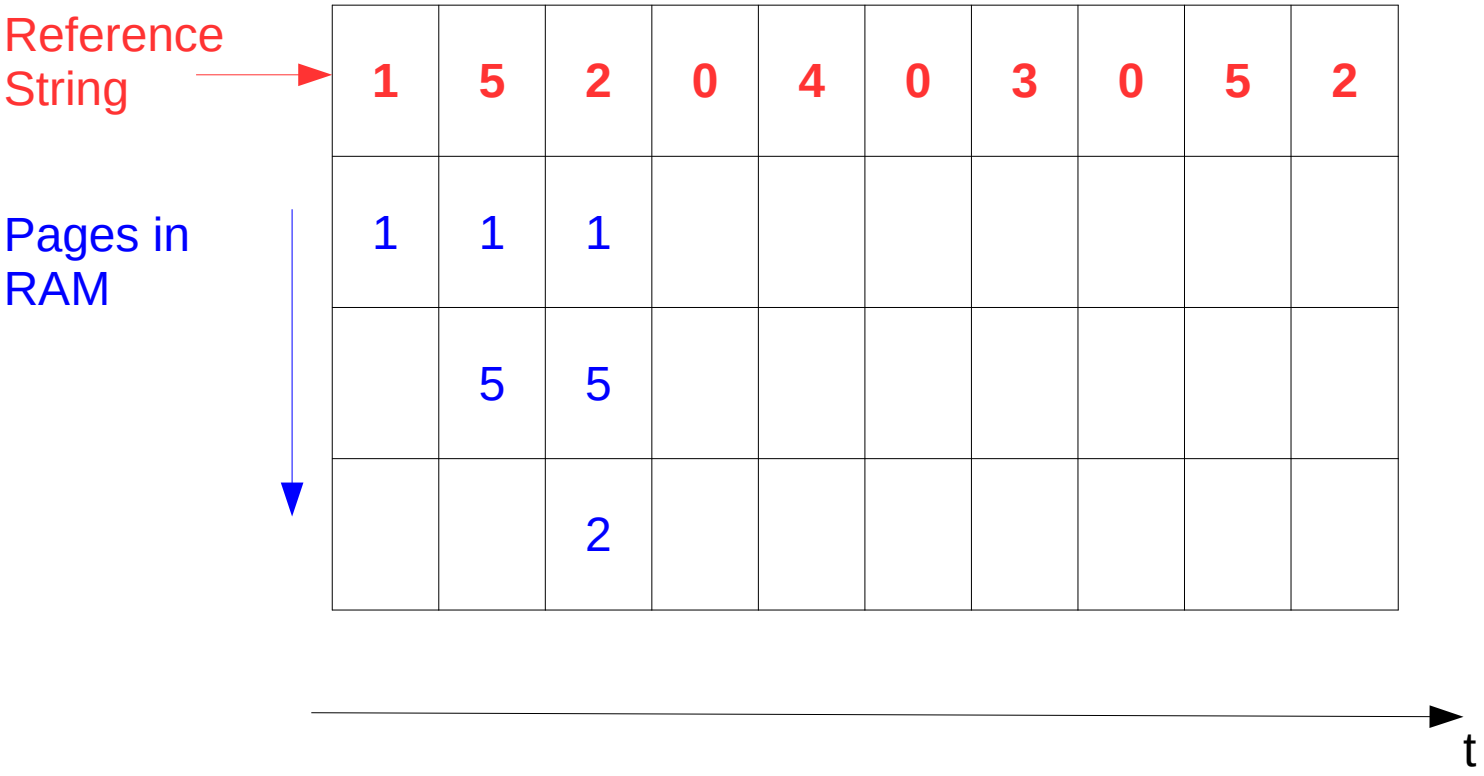
Page Replacement Algorithms (1)

- Optimal algorithm
  - determines “... the page whose next use will occur farthest in the future.” [1]
  - is the one that leads to the least page faults.
  - is impossible to implement in general purpose operating systems, because future page demands cannot be foreseen.

[1] https://en.wikipedia.org/wiki/Page_replacement_algorithm (Nov 19, 2018)

Page Replacement Algorithms (2)

Example: Optimal Algorithm



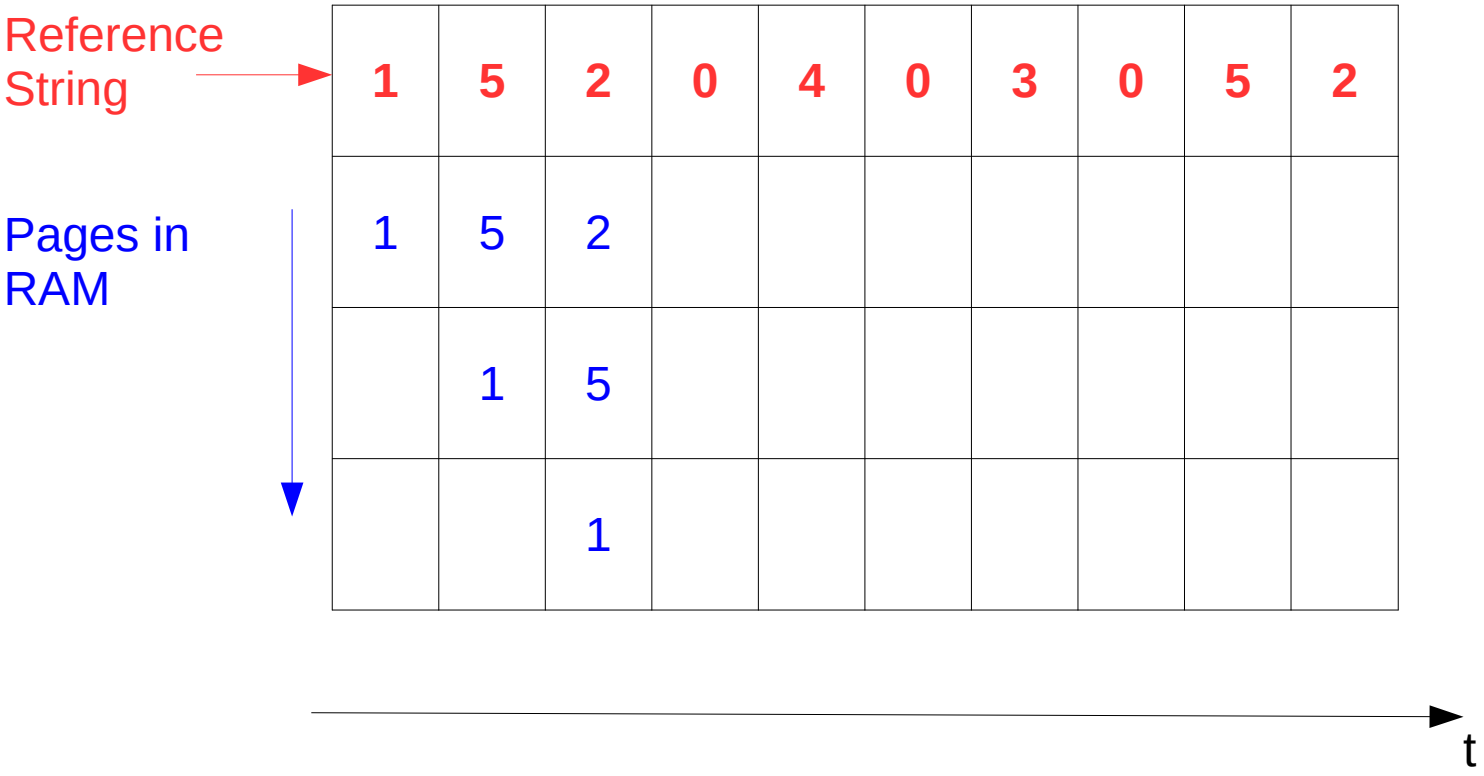
Andre M. Maier
Dept. of Computer Science

Page Replacement Algorithms (3)

- First-In-First-Out (FIFO)
  - picks always the oldest page in memory.
  - is simple to implement.
  - low administrative overhead, but performs poorly regarding our goal to minimize page faults.
  - experiences Bélády's anomaly.
    - Under certain circumstances, the number of page faults may increase as the number of page frames increases.

Page Replacement Algorithms (4)

Example: FIFO Algorithm



Andre M. Maier
Dept. of Computer Science

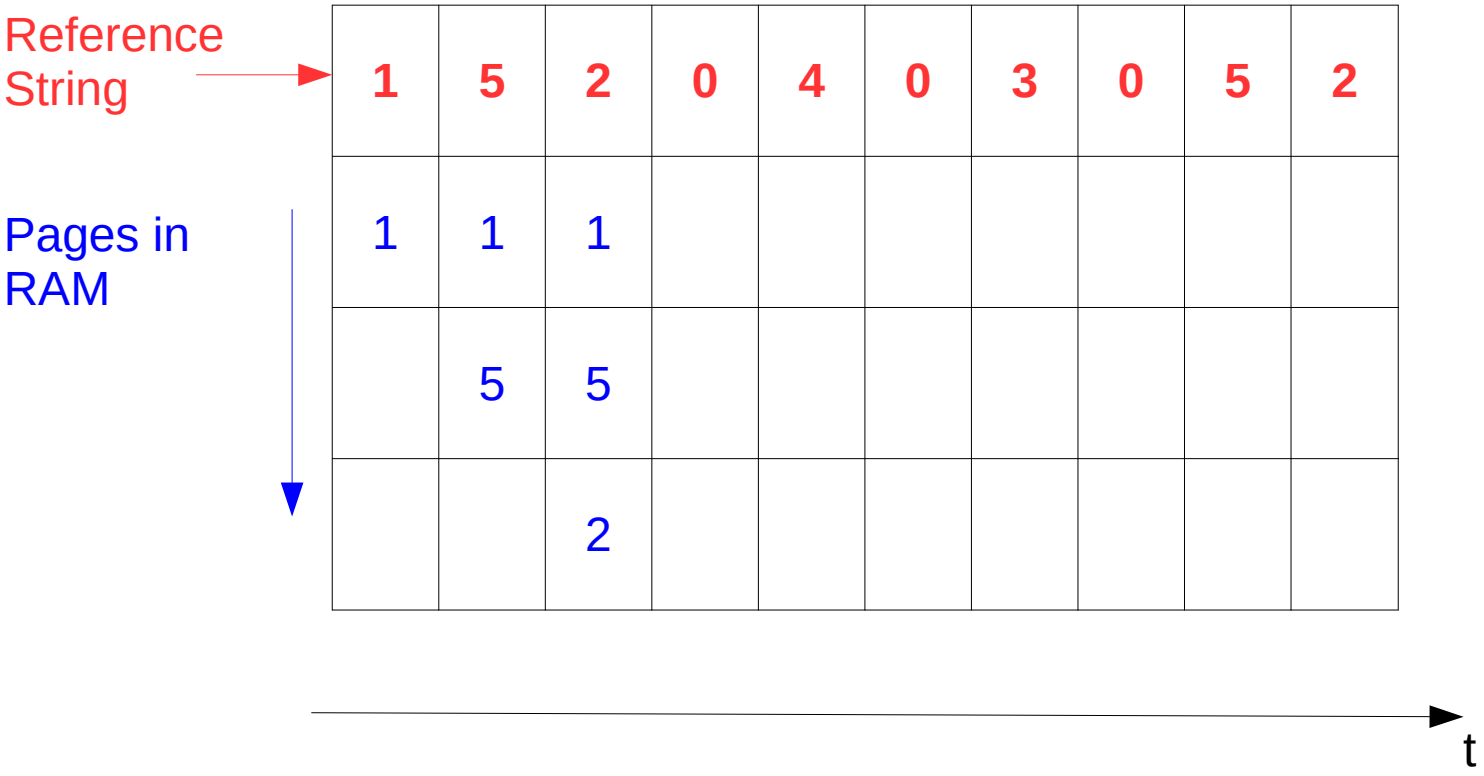
Page Replacement Algorithms (5)

- Least Recently Used (LRU)
  - picks the page that has been used least recently.
  - assumes that frequently used pages in the past will be more likely to be frequently used in the near future as well.
  - High implementation costs (either time consuming software, i.e. linked list method, or hardware)
  - Modifications and similar algorithms
    - Not Recently Used (NRU)
    - Most Recently Used (MRU)
    - Adaptive Replacement Cache (ARC)

Andre M. Maier
Dept. of Computer Science

Page Replacement Algorithms (6)

Example: LRU Algorithm



Andre M. Maier
Dept. of Computer Science

Exercises

- *Second-chance* and *Clock* are yet another two page replacement algorithms.
  - How do they work?
  - What are their advantages / disadvantages?
- Would choosing a random page for replacement be more or less efficient than other
- Implement the LRU algorithm in C to simulate the replacement for a given reference string.

Andre M. Maier
Dept. of Computer Science

Lab Exercises (1)

- Make yourself familiar with the following commands, files, and directories:
  - *free*, *vmstat*
  - */proc/meminfo*
  - */proc/sys/vm*
- Examine the */proc/<PID>* directory for memory management related files
- Write a C program that allocates most of your RAM as quickly as possible. Use a GUI system resource monitor to observe the memory allocation process.

Andre M. Maier
Dept. of Computer Science

Lab Exercises (2)

- Make yourself familiar with the following commands and files:
  - *swapon*, *swapoff*
  - */proc/sys/vm/swappiness*
- Modify your C program to allocate more memory than your computer's RAM size. Use a GUI system resource monitor to observe the memory allocation process.
- Disable swapping and re-run your program. What happens if the system runs out of memory entirely?
- How is swapping different in Microsoft Windows, and what options does it provide?