Operating Systems

Timo Hönig
Bochum Operating Systems and System Software (BOSS)
Ruhr University Bochum (RUB)

VII. Virtual Memory
May 24, 2023 (Summer Term 2023)

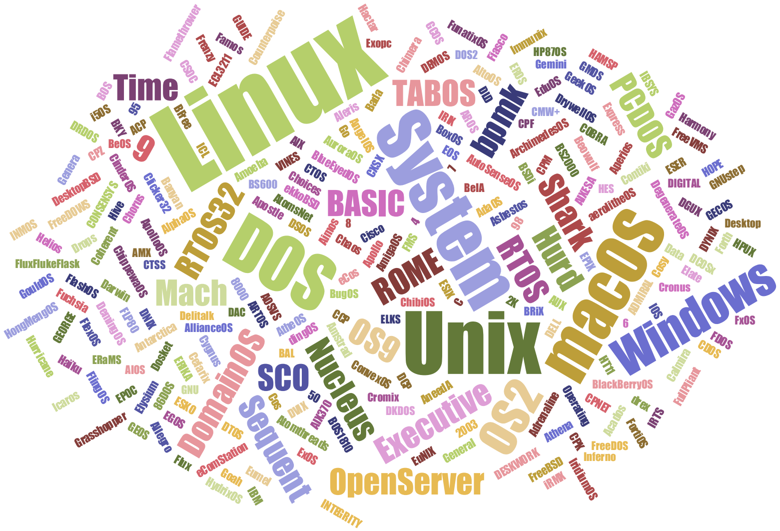
RUB Faculty of Computer Science

RUHR UNIVERSITÄT BOCHUM RUB

www.informatik.rub.de
Chair of Operating Systems and System Software

Agenda

- Recap
- Organizational Matters
- Motivation
  - Locality of Reference
  - Virtual Memory
- Demand Paging
- Page Replacement Algorithms
  - First In, First Out (FIFO),
  - Least Recently Used (LRU)
  - Second Chance (SC)
- Page Frame Allocation
  - Thrashing
  - Working Set Model
- Summary and Outlook

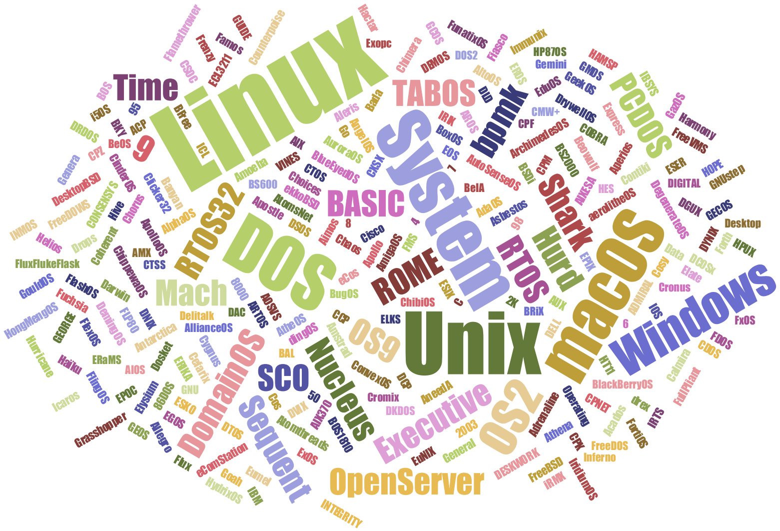


Literature References

Silberschatz, Chapter 10

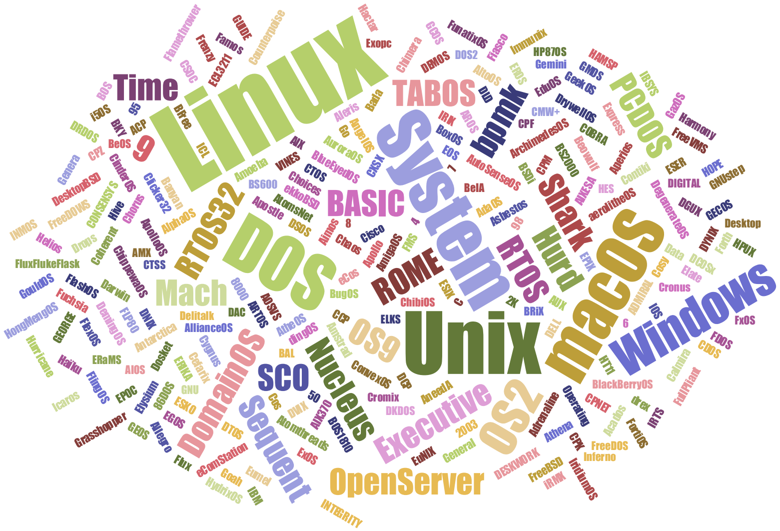
Tanenbaum, Chapter 3.3

⏮ Recap



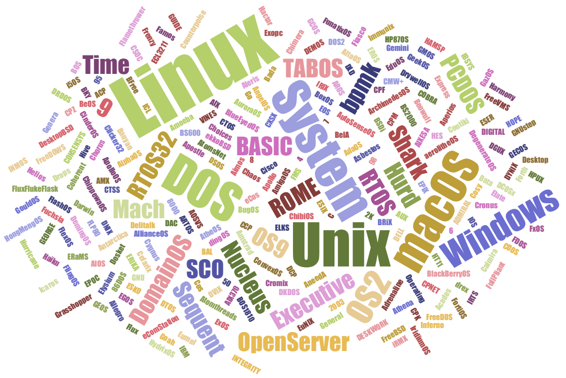
- for memory management, the operating system works very closely with the hardware
  - **segmentation** or/and **paging**
  - implicit indirection in memory access allows segments and pages of programs to be moved arbitrarily during operation
- several strategic decisions have to be made
  - placement strategies (First Fit, Best Fit, Buddy, ...)
  - different characteristics with regards to fragmentation, time overhead
  - choice of strategy depends on the expected memory access patterns

Agenda



- Recap
- Organizational Matters
- Motivation
  - Locality of Reference
  - Virtual Memory
- Demand Paging
- Page Replacement Algorithms
  - First In, First Out (FIFO)
  - Least Recently Used (LRU)
  - Second Chance (SC)
- Page Frame Allocation
  - Thrashing
  - Working Set Model
- Summary and Outlook

Organizational Matters



- lecture
  - Wednesday, 10:15 – 11:45
  - format: synchronous, **hybrid**
    - → in presence (Room HID, Building ID)
    - → online lecture (Zoom)
  - **exam:** August 7, 2023 (first appointment)
    September 25, 2023 (retest appointment)
- exercises: **group allocation ~~almost~~ complete**
  - make use of group work - for your own benefit!
- manage course material, asynchronous communication: Moodle
- `https://moodle.ruhr-uni-bochum.de/course/view.php?id=50698`

Agenda

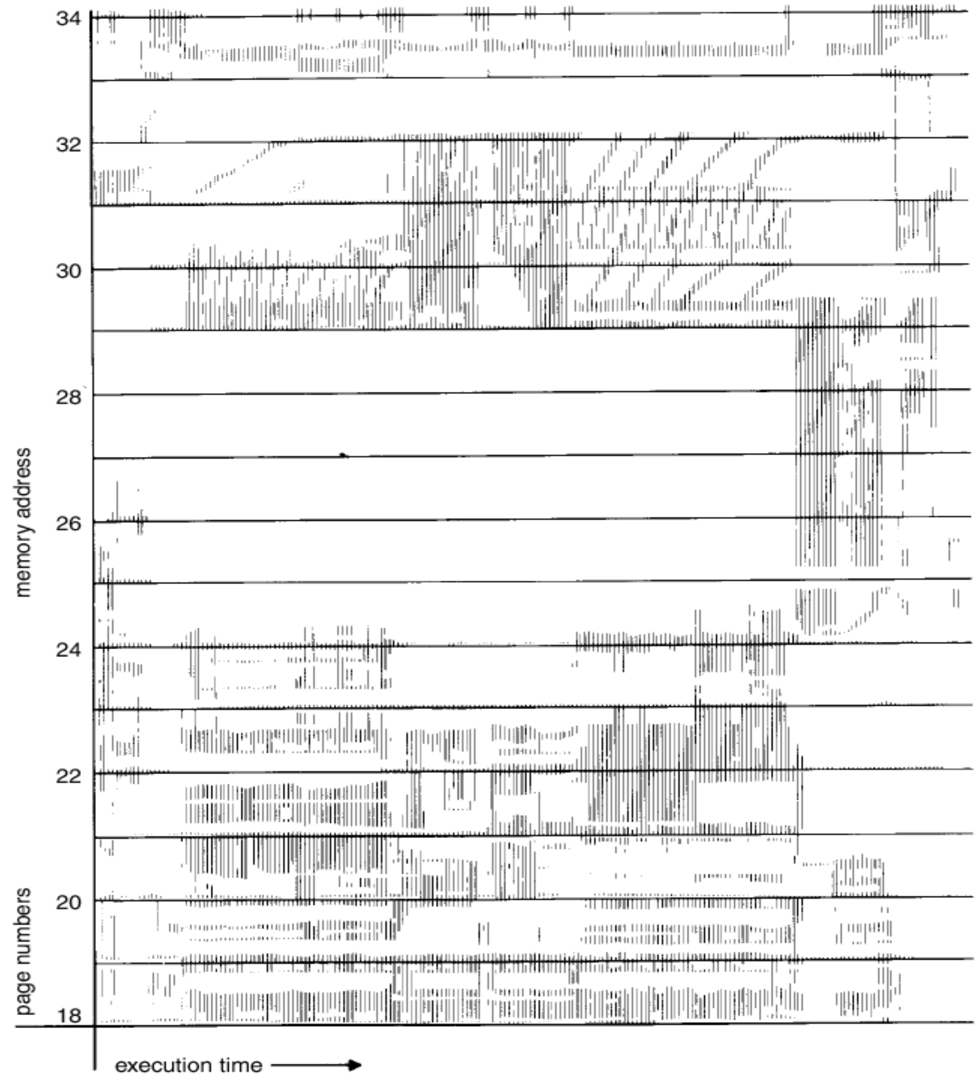
- Recap
- Organizational Matters
- **Motivation**
  - Locality of Reference
  - Virtual Memory
- Demand Paging
- Page Replacement Algorithms
  - First In, First Out (FIFO)
  - Least Recently Used (LRU)
  - Second Chance (SC)
- Page Frame Allocation
  - Thrashing
  - Working Set Model
- Summary and Outlook

Locality of Reference – Memory Access

- single instructions require only a few memory pages
- even over longer periods of time: strong locality of reference
  - instructions are executed, for example, one after the other

➔ locality can be exploited if primary memory is not large enough to keep pages of all processes at the same time

  - for example, overlay technique



Source: Silberschatz, „Operating System Concepts“

Virtual Memory – Main Idea

- add a system layer of indirection: decouple the memory requirement of processes from the system's main memory
  - processes do not need all memory locations equally often
    - → certain functions are rarely or not at all used (e.g., error handling)
    - → certain data structures are not fully populated
  - processes may require more memory than main memory is available

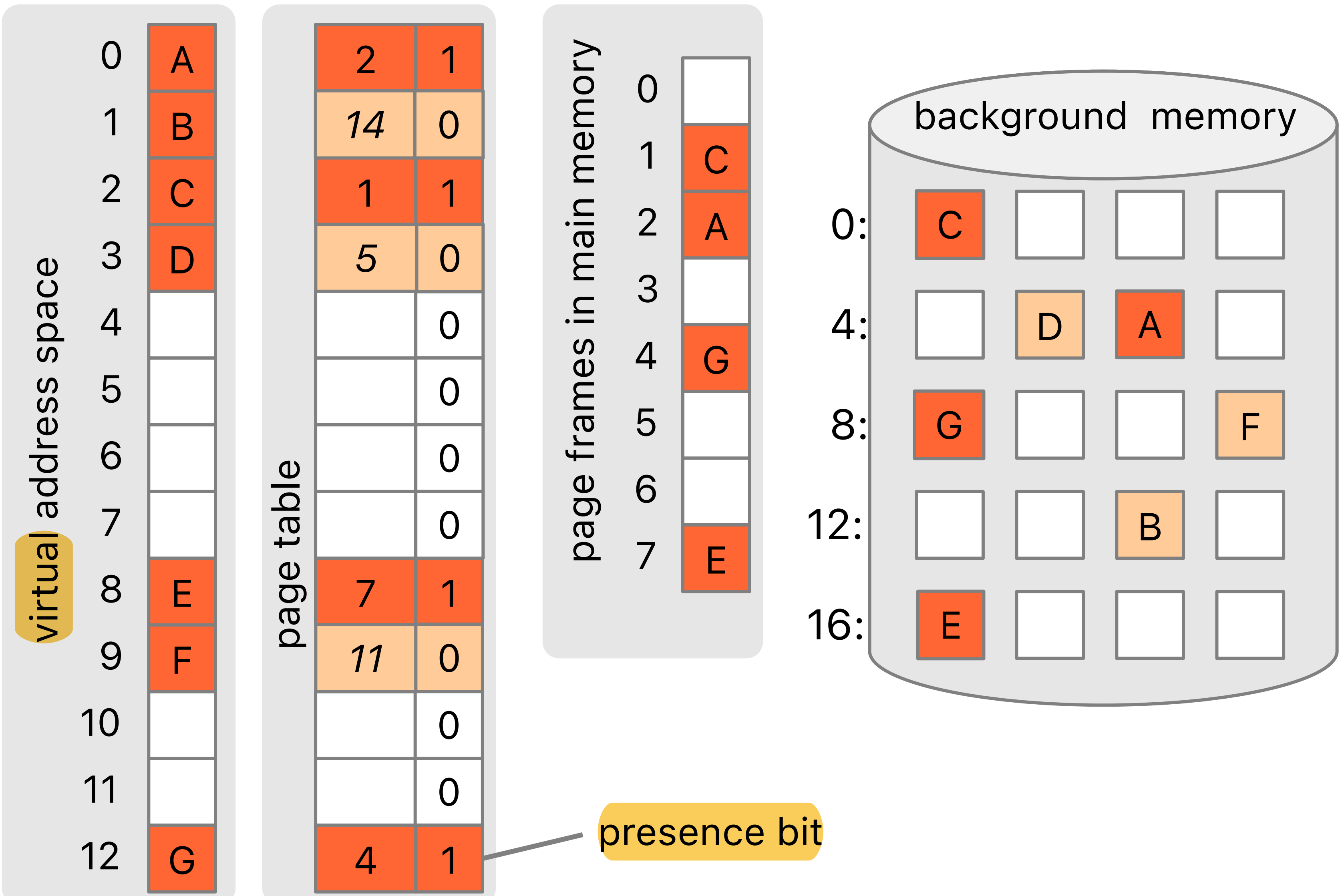
- main idea
  - delude the existence of a (unlimited) large main memory
  - swap out unused memory areas from primary to secondary storage

    byte-addressable main memory (e.g., RAM) →
    block-addressable background storage/memory (e.g., SSD, HDD)
  - provide the required memory areas on demand → demand paging

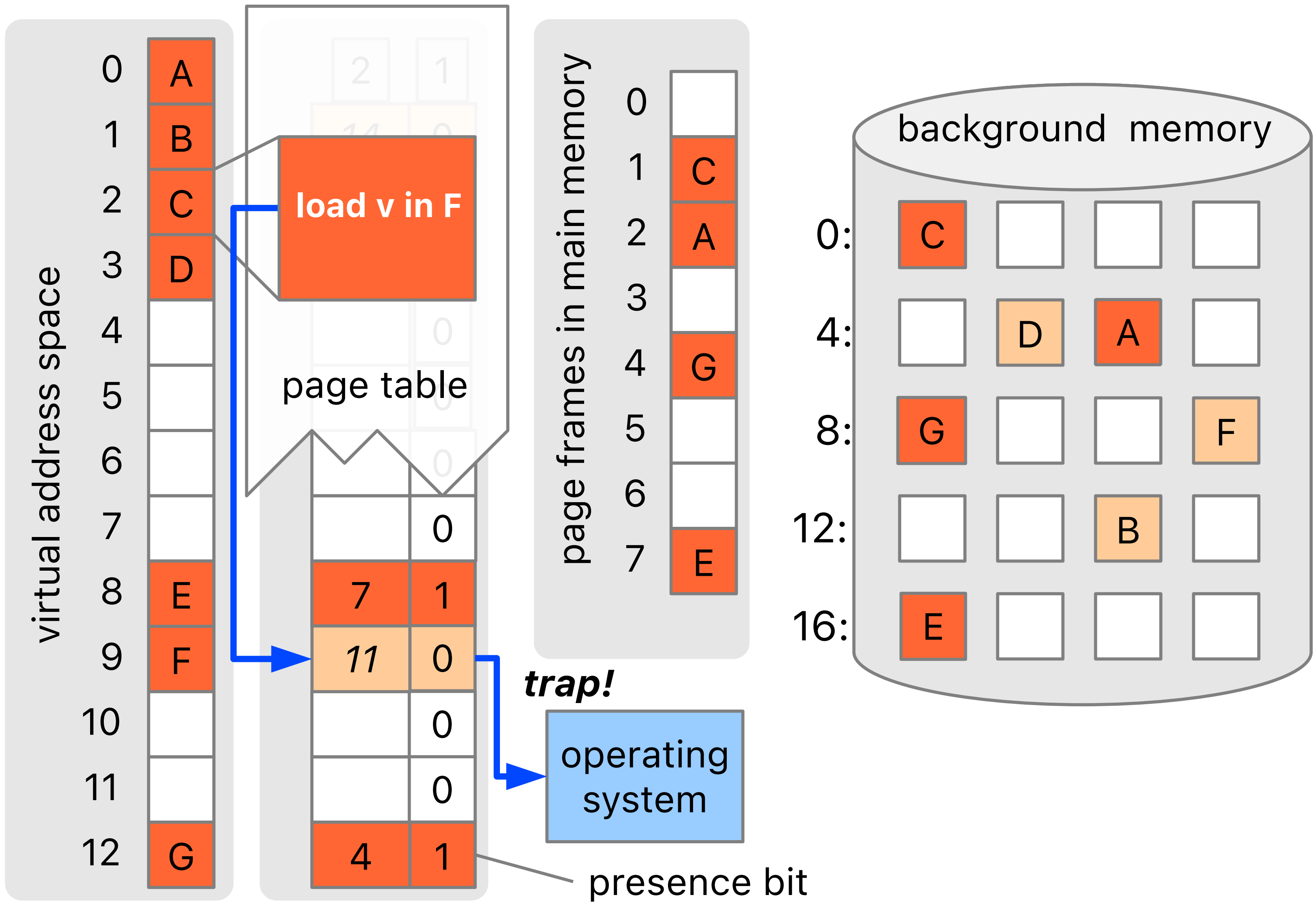
Agenda

- Recap
- Organizational Matters
- Motivation
  - Locality of Reference
  - Virtual Memory
- **Demand Paging**
- Page Replacement Algorithms
  - First In, First Out (FIFO)
  - Least Recently Used (LRU)
  - Second Chance (SC)
- Page Frame Allocation
  - Thrashing
  - Working Set Model
- Summary and Outlook

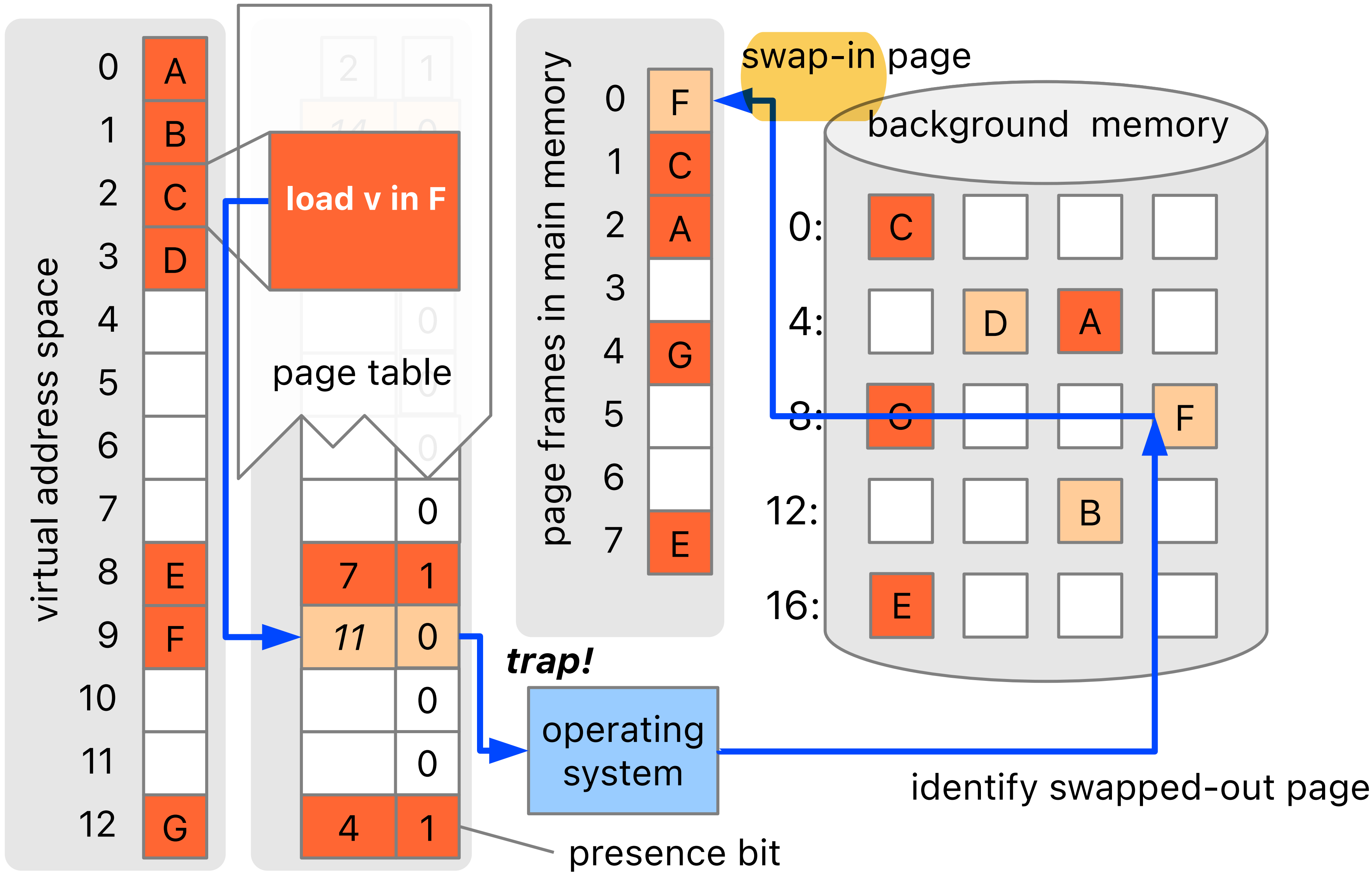
Demand Paging – Provide Pages on Demand



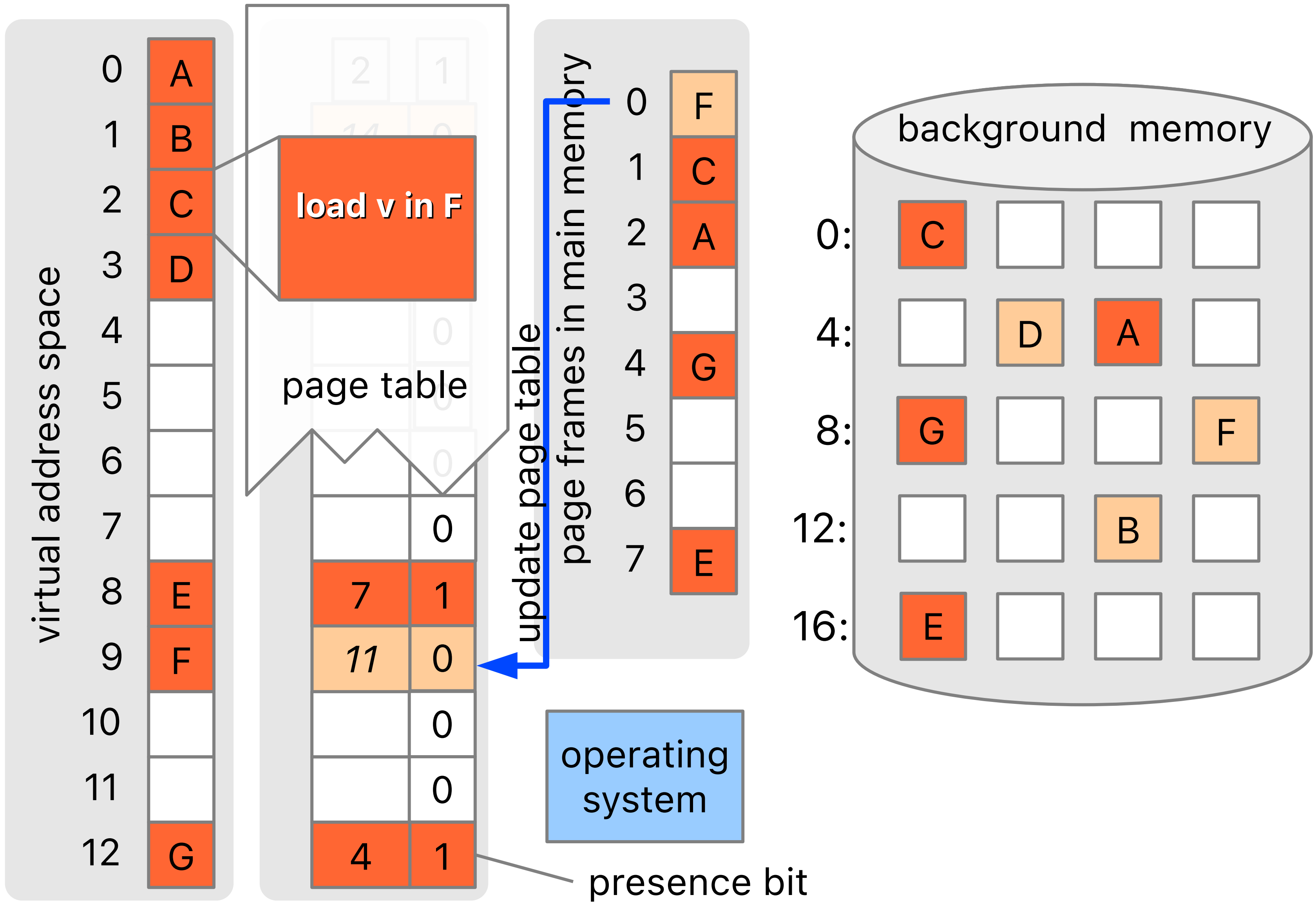
Demand Paging – Page Fault Handling



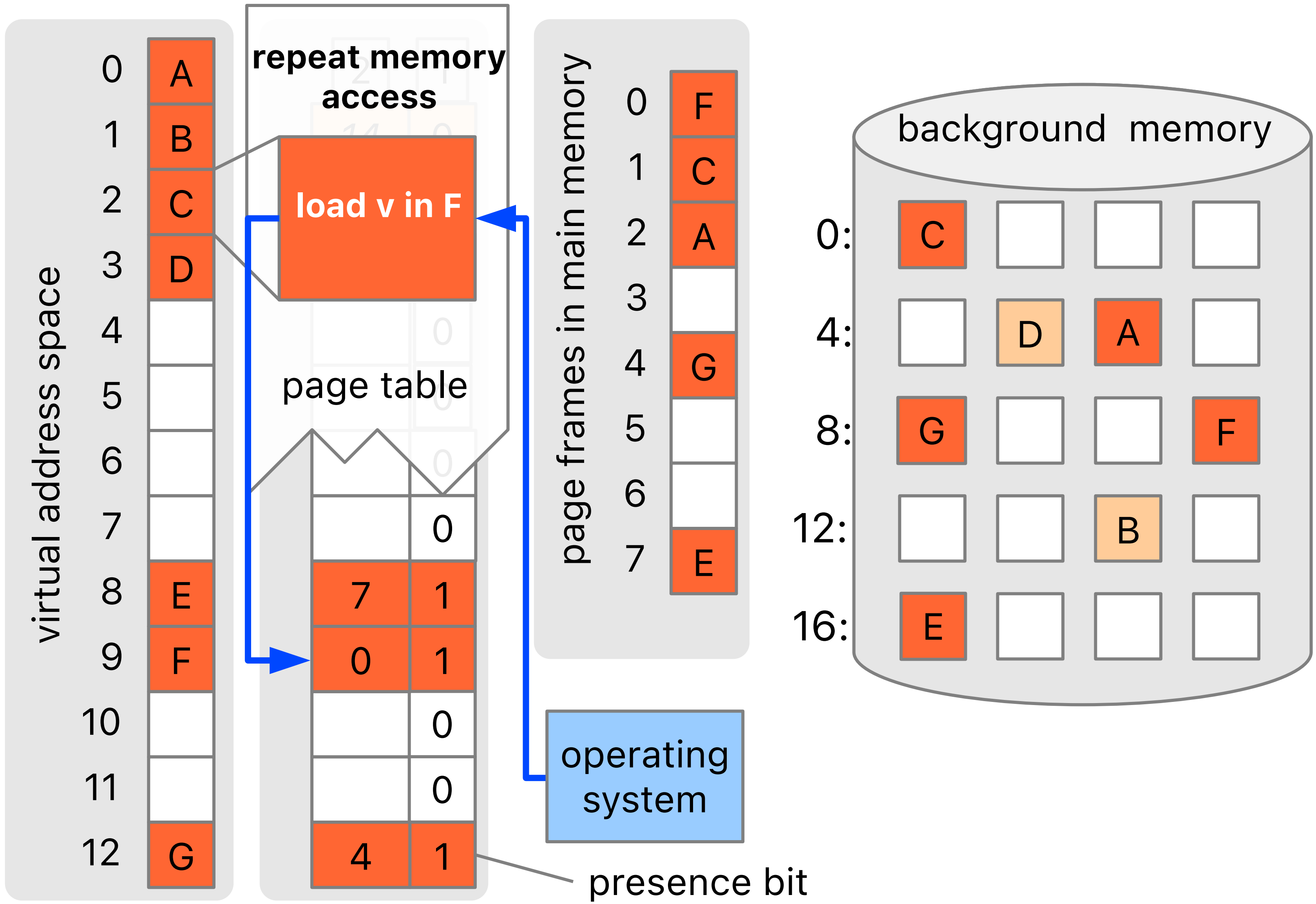
Demand Paging – Page Fault Handling



Demand Paging – Page Fault Handling



Demand Paging – Page Fault Handling



Demand Paging – Discussion

- operational costs of demand paging (i.e., performance impact)
  - without page fault
    - ➔ effective access time between 10 and 200 nanoseconds
  - with page fault
    - ➔ *p*: probability of a page fault
    - ➔ first assumption: time to swap-in a page from background memory is 25 milliseconds (8 ms latency, 15 ms positioning time, 1 ms transmission time)
    - ➔ second assumption: main memory access time is 100 ns
    - ➔ effective access time:
      $(1 - p) \cdot 100 + p \cdot 25000000 = 100 + 24999900 \cdot p$

➔ rate of page faults must be *very* low
  - *p* close to zero
  - realistic only due to data locality

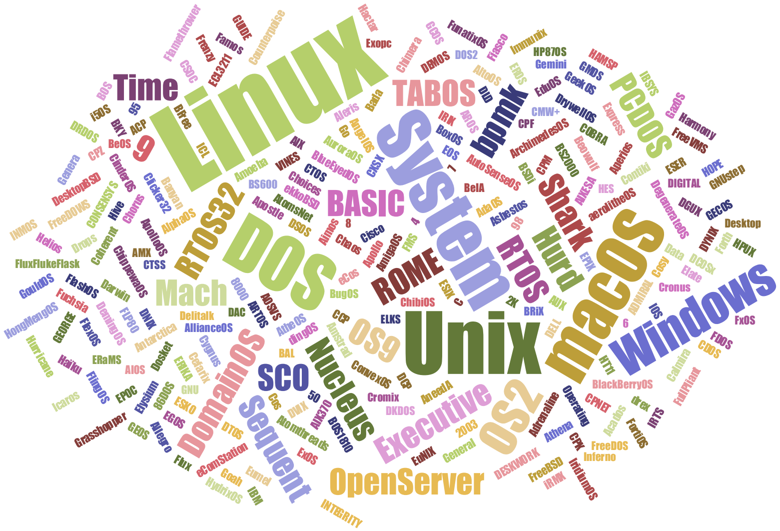
Demand Paging – Characteristics

- operational concerns on process generation (e.g., fork)
  - **copy-on-write**
    - → also easy to implement with paging MMU
    - → finer granularity than with segmentation
  - program execution and memory accesses are interleaved
    - → required pages are loaded only as they are needed

- page locking
  - lock pages that are used by DMA during trap (i.e., page fault handling)
  - necessary for input/output operations

Agenda

- Recap
- Organizational Matters
- Motivation
  - Locality of Reference
  - Virtual Memory
- Demand Paging
- Page Replacement Algorithms
  - First In, First Out (FIFO)
  - Least Recently Used (LRU)
  - Second Chance (SC)
- Page Frame Allocation
  - Thrashing
  - Working Set Model
- Summary and Outlook



Page Replacement Algorithms

- demand paging: need for free page frame(s)
- modus operandi:
  - page fault: exception handling via trap into the operating system
  - if no free page frame is available: swap-out a page
  - swap-in the required page
  - repeat memory access
- what to do if there is no free page frame?
  - a page must be swapped-out to the background memory to free up space for a new page in main memory
  - pages that have not been changed (dirty bit in page table)
  - if page was changed: process preemption requires swapping
- problem
  - which page ("victim") should be swapped-out?

Page Replacement Algorithms

- consideration of replacement strategies and their effect on reference sequences
- reference sequence
  - sequence of page numbers that model the memory access behavior of a process
  - determination of reference sequences, for example, by recording the accessed addresses
    - → reduce the recorded sequence to page numbers
    - → group accesses to the same page in immediate succession
  - reference sequence: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

First-In, First-Out (FIFO)

- oldest page is replaced
- necessary states:
  - age (time of swap-in) for each page frame
- sequence of replacement

reference sequence		1	2	3	4	1	2	5	1	2	3	4	5
main memory	page frame 1	**1**											
	page frame 2												
	page frame 3												
control states (age per page frame)	page frame 1	0											
	page frame 2	>											
	page frame 3	>											

First-In, First-Out (FIFO)

- oldest page is replaced
- necessary states:
  - age (time of swap-in) for each page frame
- sequence of replacement

reference sequence		1	2	3	4	1	2	5	1	2	3	4	5
main memory	page frame 1	**1**	1										
	page frame 2		**2**										
	page frame 3												
control states (age per page frame)	page frame 1	0	1										
	page frame 2	>	0										
	page frame 3	>	>										

First-In, First-Out (FIFO)

- oldest page is replaced
- necessary states:
  - age (time of swap-in) for each page frame
- sequence of replacement

reference sequence		1	2	3	4	1	2	5	1	2	3	4	5
main memory	page frame 1	**1**	1	1									
	page frame 2		**2**	2									
	page frame 3			**3**									
control states (age per page frame)	page frame 1	0	1	2									
	page frame 2	>	0	1									
	page frame 3	>	>	0									

First-In, First-Out (FIFO)

- oldest page is replaced
- necessary states:
  - age (time of swap-in) for each page frame
- sequence of replacement

reference sequence		1	2	3	4	1	2	5	1	2	3	4	5
main memory	page frame 1	**1**	1	1	**4**								
	page frame 2		**2**	2	2								
	page frame 3			**3**	3								
control states (age per page frame)	page frame 1	0	1	2	0								
	page frame 2	>	0	1	2								
	page frame 3	>	>	0	1								

First-In, First-Out (FIFO)

- oldest page is replaced
- necessary states:
  - age (time of swap-in) for each page frame
- sequence of replacement

reference sequence		1	2	3	4	1	2	5	1	2	3	4	5
main memory	page frame 1	**1**	1	1	**4**	4							
	page frame 2		**2**	2	2	**1**							
	page frame 3			**3**	3	3							
control states (age per page frame)	page frame 1	0	1	2	0	1							
	page frame 2	>	0	1	2	0							
	page frame 3	>	>	0	1	2							

First-In, First-Out (FIFO)

- oldest page is replaced
- necessary states:
  - age (time of swap-in) for each page frame
- sequence of replacement

reference sequence		1	2	3	4	1	2	5	1	2	3	4	5
main memory	page frame 1	**1**	1	1	**4**	4	4						
	page frame 2		**2**	2	2	**1**	1						
	page frame 3			**3**	3	3	**2**						
control states (age per page frame)	page frame 1	0	1	2	0	1	2						
	page frame 2	>	0	1	2	0	1						
	page frame 3	>	>	0	1	2	0						

First-In, First-Out (FIFO)

- oldest page is replaced
- necessary states:
  - age (time of swap-in) for each page frame
- sequence of replacement

reference sequence		1	2	3	4	1	2	5	1	2	3	4	5
main memory	page frame 1	**1**	1	1	**4**	4	4	**5**					
	page frame 2		**2**	2	2	**1**	1	1					
	page frame 3			**3**	3	3	**2**	2					
control states (age per page frame)	page frame 1	0	1	2	0	1	2	0					
	page frame 2	>	0	1	2	0	1	2					
	page frame 3	>	>	0	1	2	0	1					

First-In, First-Out (FIFO)

- oldest page is replaced
- necessary states:
  - age (time of swap-in) for each page frame
- sequence of replacement

reference sequence		1	2	3	4	1	2	5	1	2	3	4	5
main memory	page frame 1	**1**	1	1	**4**	4	4	**5**	5				
	page frame 2		**2**	2	2	**1**	1	1	1				
	page frame 3			**3**	3	3	**2**	2	2				
control states (age per page frame)	page frame 1	0	1	2	0	1	2	0	1				
	page frame 2	>	0	1	2	0	1	2	3				
	page frame 3	>	>	0	1	2	0	1	2				

First-In, First-Out (FIFO)

- oldest page is replaced
- necessary states:
  - age (time of swap-in) for each page frame
- sequence of replacement

reference sequence		1	2	3	4	1	2	5	1	2	3	4	5
main memory	page frame 1	**1**	1	1	**4**	4	4	**5**	5	5	5	5	5
	page frame 2		**2**	2	2	**1**	1	1	1	1	**3**	3	3
	page frame 3			**3**	3	3	**2**	2	2	2	2	**4**	4
control states (age per page frame)	page frame 1	0	1	2	0	1	2	0	1	2	3	4	5
	page frame 2	>	0	1	2	0	1	2	3	4	0	1	2
	page frame 3	>	>	0	1	2	0	1	2	3	4	0	1

- 9 swap-in operations

First-In, First-Out (FIFO)

- larger main memory with 4 page frames
  - → 10 swap-in operations
- FIFO anomaly (Bélády's anomaly, 1969)

reference sequence		1	2	3	4	1	2	5	1	2	3	4	5
main memory	page frame 1	**1**	1	1	1	1	1	**5**	5	5	5	**4**	4
	page frame 2		**2**	2	2	2	2	2	**1**	1	1	1	**5**
	page frame 3			**3**	3	3	3	3	3	**2**	2	2	2
	page frame 4				**4**	4	4	4	4	4	**3**	3	3
control states (age per page frame)	page frame 1	0	1	2	3	4	5	0	1	2	3	0	1
	page frame 2	>	0	1	2	3	4	5	0	1	2	3	0
	page frame 3	>	>	0	1	2	3	4	5	0	1	2	3
	page frame 4	>	>	>	0	1	2	3	4	5	0	1	2

Optimal Replacement Strategy (OPT)

- longest forward distance (LFD, dt. Vorwärtsabstand)
  - time interval until next access to the corresponding page
- strategy OPT (or MIN) is optimal (with fixed number of page frames): minimum number of swap-in/replacements (here: 7)
  - always replace the page with the longest forward distance

reference sequence		1	2	3	4	1	2	5	1	2	3	4	5
main memory	page frame 1	**1**	1	1	1	1	1	1	1	1	**3**	**4**	4
	page frame 2		**2**	2	2	2	2	2	2	2	2	2	2
	page frame 3			**3**	**4**	4	4	**5**	5	5	5	5	5
control states (LFD)	page frame 1	4	3	2	1	3	2	1	>	>	>	>	>
	page frame 2	>	4	3	2	1	3	2	1	>	>	>	>
	page frame 3	>	>	7	7	6	5	5	4	3	2	1	>

Optimal Replacement Strategy (OPT)

- increase the size of the main memory (4 page frames): 6 swap-in operations
  - no anomaly

reference sequence		1	2	3	4	1	2	5	1	2	3	4	5
main memory	page frame 1	**1**	1	1	1	1	1	1	1	1	1	**4**	4
	page frame 2		**2**	2	2	2	2	2	2	2	2	2	2
	page frame 3			**3**	3	3	3	3	3	3	3	3	3
	page frame 4				**4**	4	4	**5**	5	5	5	5	5
control states (LFD)	page frame 1	4	3	2	1	3	2	1	>	>	>	>	>
	page frame 2	>	4	3	2	1	3	2	1	>	>	>	>
	page frame 3	>	>	7	6	5	4	3	2	1	>	>	>
	page frame 4	>	>	>	7	6	5	5	4	3	2	1	>

Optimal Replacement Strategy – Discussion

- implementation of OPT practically impossible
  - reference sequence would have to be known in advance (oracle)
  - OPT is only useful for comparing strategies
- research for strategies that are as close to OPT as possible
  - for example: **Least Recently Used** (LRU)

Least Recently Used (LRU)

- longest backward distance
  - time interval since the last access to the corresponding page
- LRU strategy (10 swap-in operations)
  - always replace the page that has not been used for the longest time

reference sequence		1	2	3	4	1	2	5	1	2	3	4	5
main memory	page frame 1	**1**	1	1	**4**	4	4	**5**	5	5	**3**	3	3
	page frame 2		**2**	2	2	**1**	1	1	1	1	1	**4**	4
	page frame 3			**3**	3	3	**2**	2	2	2	2	2	**5**
control states (oldest access time)	page frame 1	0	1	2	0	1	2	0	1	2	0	1	2
	page frame 2	>	0	1	2	0	1	2	0	1	2	0	1
	page frame 3	>	>	0	1	2	0	1	2	0	1	2	0

Least Recently Used (LRU)

- increase the size of the main memory (4 page frames)
- 8 swap-in operations

reference sequence		1	2	3	4	1	2	5	1	2	3	4	5
main memory	page frame 1	**1**	1	1	1	1	1	1	1	1	1	1	**5**
	page frame 2		**2**	2	2	2	2	2	2	2	2	2	2
	page frame 3			**3**	3	3	3	**5**	5	5	5	**4**	4
	page frame 4				**4**	4	4	4	4	4	**3**	3	3
control states (oldest access time)	page frame 1	0	1	2	3	0	1	2	0	1	2	3	0
	page frame 2	>	0	1	2	3	0	1	2	0	1	2	3
	page frame 3	>	>	0	1	2	3	0	1	2	3	0	1
	page frame 4	>	>	>	0	1	2	3	4	5	0	1	2

Least Recently Used (LRU) – Discussion

- no anomaly
  - in general, there is a class of algorithms (stack algorithms) for which no anomaly occurs:
    - with stack algorithms, if there are *k* page frames at any time, a subset of the pages is swapped-in that would also be swapped-in if there were *k+1* page frames at the same time
    - LRU: there are always the last *k* used pages swapped-in
    - OPT: the *k* already used pages are swapped-in, which will be accessed next
- problem
  - implementation of LRU only possible with hardware support
  - every memory access must be considered

Least Recently Used (LRU) – Discussion

- naive idea: hardware support through counter
  - CPU has a counter that is incremented with each memory access
  - with each memory access: current counter value is written into the respective page descriptor
  - evict the page with the smallest counter reading (→ search!)

- complex implementation
  - many additional memory accesses
  - high memory requirements
  - minimum search in page-fault handling

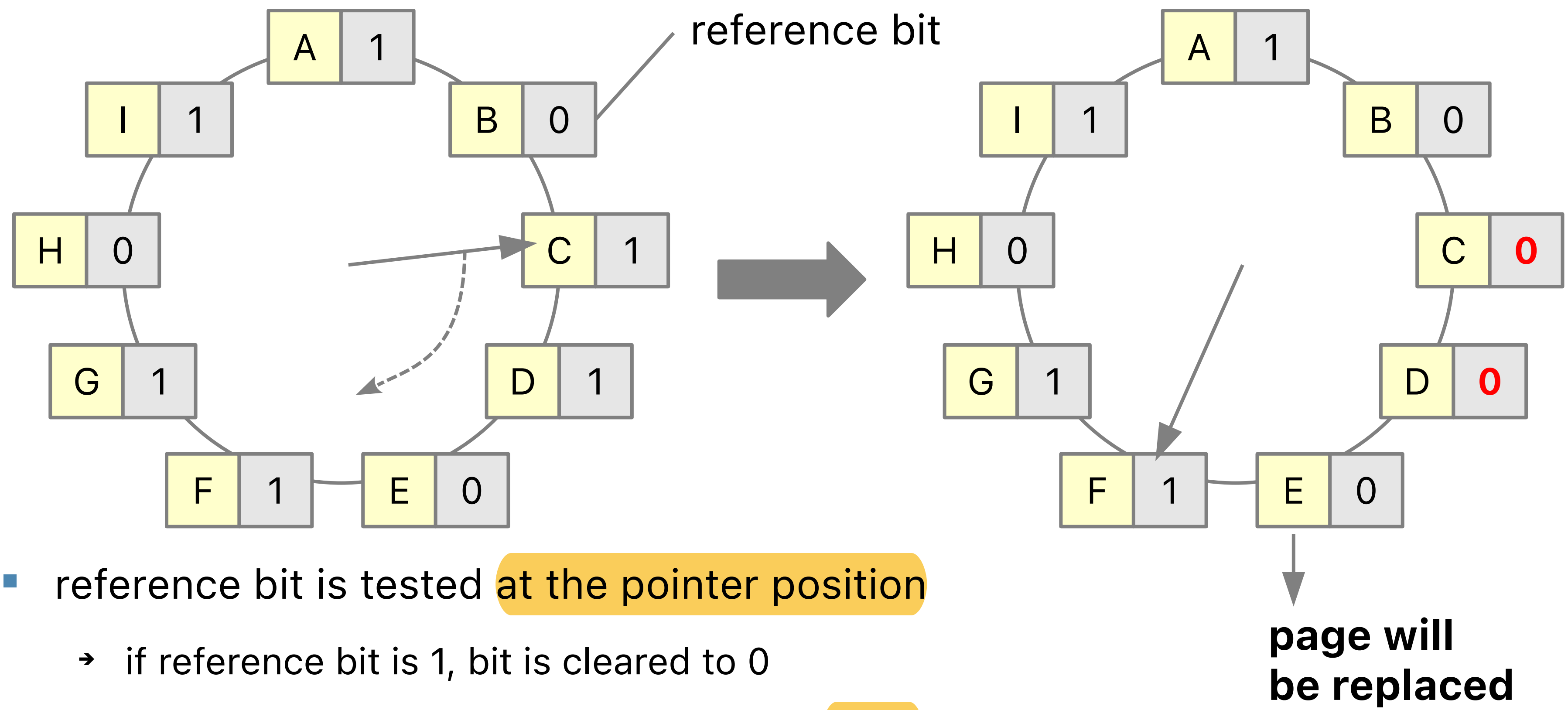
Second Chance (SC)

- core idea: use of reference bits
  - reference bit in page descriptor is automatically set by hardware when page is accessed
    - ➔ easy to implement
    - ➔ reduced number of additional memory accesses
    - ➔ supported by modern processors and MMUs (e.g., x86: access bit)

- goal: approximation of LRU
  - reference bit is is initially set to 1 for a freshly swapped-in page
  - when a victim page is searched, reference bits in the page table are inspected in turn
    - ➔ if inspected reference bit is 1, it is set to 0 (second chance)
    - ➔ else: if inspected reference bit is 0, the page is chosen as victim and swapped-out

Second Chance (SC)

- implementation with a circulating pointer (clock)



- reference bit is tested at the pointer position
  - if reference bit is 1, bit is cleared to 0
  - if reference bit is 0, replaceable page is found
  - pointer is incremented; if no victim page is found: repeat
- if all reference bits are set to 1: Second Chance degrades to FIFO

Second Chance (SC)

- modus operandi with 3 page frames
- 9 swap-in operations

reference sequence		1	2	3	4	1	2	5	1	2	3	4	5
main memory	page frame 1	1	1	1	4	4	4	5	5	5	5	5	5
	page frame 2		2	2	2	1	1	1	1	1	3	3	3
	page frame 3			3	3	3	2	2	2	2	2	4	4
control states (reference bits)	page frame 1	1	1	1	1	1	1	1	1	1	0	0	1
	page frame 2	0	1	1	0	1	1	0	1	1	1	1	1
	page frame 3	0	0	1	0	0	1	0	0	1	0	1	1
	circulating pointer	2	3	1	2	3	1	2	2	2	3	1	1

不用替换时，指针不变，
但是更新ref bit

Second Chance (SC)

- increase the size of the main memory (4 page frames)
- 10 swap-in operations

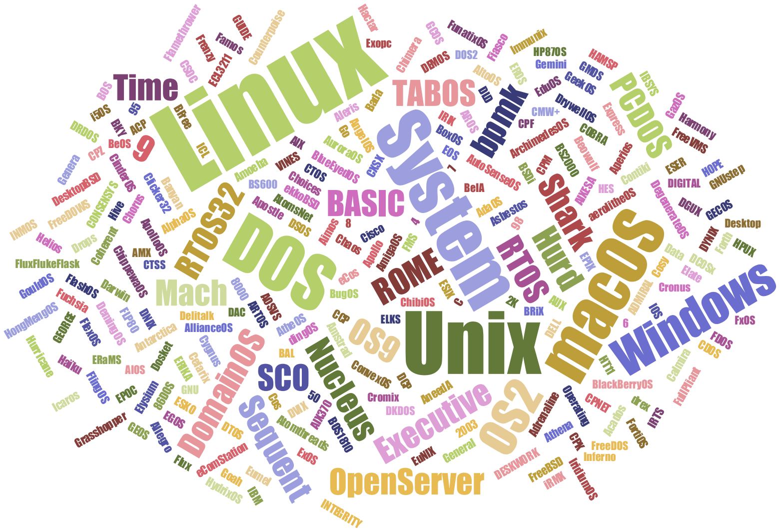
reference sequence		1	2	3	4	1	2	5	1	2	3	4	5
main memory	page frame 1	**1**	1	1	1	1	1	**5**	5	5	5	**4**	4
	page frame 2		**2**	2	2	2	2	2	**1**	1	1	1	**5**
	page frame 3			**3**	3	3	3	3	3	**2**	2	2	2
	page frame 4				**4**	4	4	4	4	4	**3**	3	3
control states (reference bits)	page frame 1	1	1	1	1	1	1	1	1	1	1	1	1
	page frame 2	0	1	1	1	1	1	0	1	1	1	0	1
	page frame 3	0	0	1	1	1	1	0	0	1	1	0	0
	page frame 4	0	0	0	1	1	1	0	0	0	1	0	0
	circulating pointer	**2**	**3**	**4**	**1**	**1**	**1**	**2**	**3**	**4**	**1**	**2**	**3**

Second Chance (SC)

- second chance may lead to FIFO anomaly
  - if all reference bits are set, the decision is made according to FIFO
- however, normally Second Chance comes close to LRU

- enhanced Second Chance variant
  - modification bit can be considered additionally (dirty bit)
  - four classes (reference bit, modification bit):
    - → (0,0): neither recently used nor modified
    - → (0,1): not recently used but modified
    - → (1,0): recently used and unmodified
    - → (1,1): recently used and modified
  - search for the lowest class

Agenda



- Recap
- Organizational Matters
- Motivation
  - Locality of Reference
  - Virtual Memory
- Demand Paging
- Page Replacement Algorithms
  - First In, First Out (FIFO)
  - Least Recently Used (LRU)
  - Second Chance (SC)
- Page Frame Allocation
  - Thrashing
  - Working Set Model
- Summary and Outlook

Page Frame Allocation

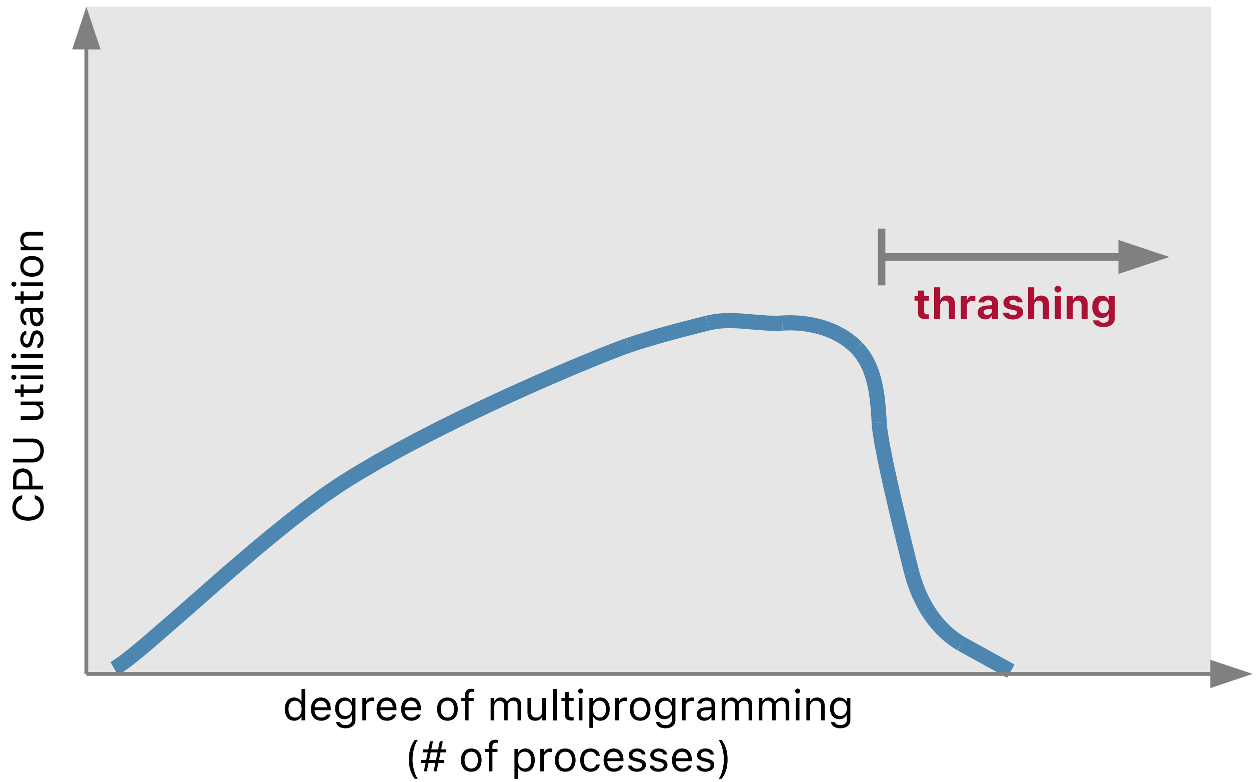
- problem: distribution of available page frames to processes
  - how many swapped-in pages should a process possess?
    - → maximum: limited by size of available memory (number of page frames)
    - → minimum: depends on the processor architecture
      - – at least the number of pages necessary which in theory are required for executing a single instruction
      - – for example: 2 pages for instructions, 4 pages for the data
- homogenous allocation
  - number of processes determines how many page frames each process can possess
- size-based allocation
  - process size influences page frame allocation

Page Frame Allocation

- global and local page frame allocation
  - **local**: processes always replace their own pages
    - ➔ page fault handling: responsibility of the individual process, only
  - **global**: processes can also displace pages of other processes
    - ➔ better efficiency, as unused pages can be used by other processes

Thrashing

- thrashing (dt. Seitenflattern) happens when swapped-out pages are immediately addressed again
  - process spends more time waiting for page faults handled than actually executing its own code



Thrashing

- causes
  - process operates with a number of page frames close to the necessary minimum
  - too many processes in the system at the same time
  - poor replacement strategy

➔ local page allocation resolves thrashing between processes

➔ allocation of a sufficiently large number of page frames resolves thrashing within the pages of a process

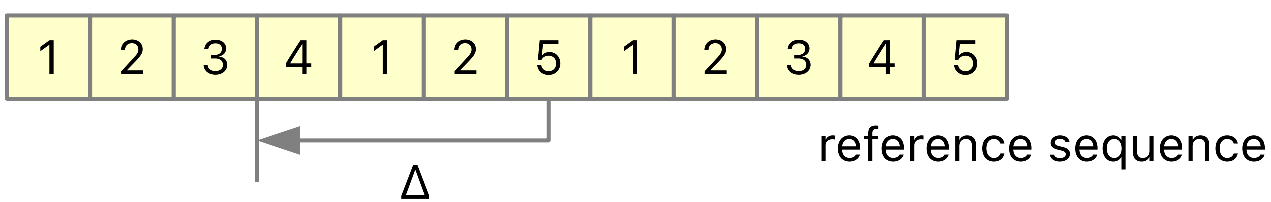
- limitation of the number of processes

Thrashing – Solution 1

- solution 1: swap-out processes
- inactive processes do not need page frames
  - page frames are distributed to a smaller number of processes
  - needs to be interlinked with scheduling
    - ➔ prevent starvation of processes
    - ➔ achieve good response times (low latency)

Thrashing – Solution 2

- solution 2: working set model
- working set: amount of pages that a process really needs
  - can only be approximated, as usually not predictable
- approach by looking at the last Δ pages that were referenced
  - appropriate choice of Δ
    - → too large: overlapping of local access patterns
    - → too small: work set does not contain all necessary pages



Working Set Model

- example: working set model for different Δ

reference sequence		1	2	3	4	1	2	5	1	2	3	4	5
Δ = 3	page 1	X	X	X		X	X	X	X	X	X		
	page 2		X	X	X		X	X	X	X	X	X	
	page 3			X	X	X					X	X	X
	page 4				X	X	X					X	X
	page 5							X	X	X			X
Δ = 4	page 1	X	X	X	X	X	X	X	X	X	X	X	
	page 2		X	X	X	X	X	X	X	X	X	X	X
	page 3			X	X	X	X				X	X	X
	page 4				X	X	X	X				X	X
	page 5							X	X	X	X		X

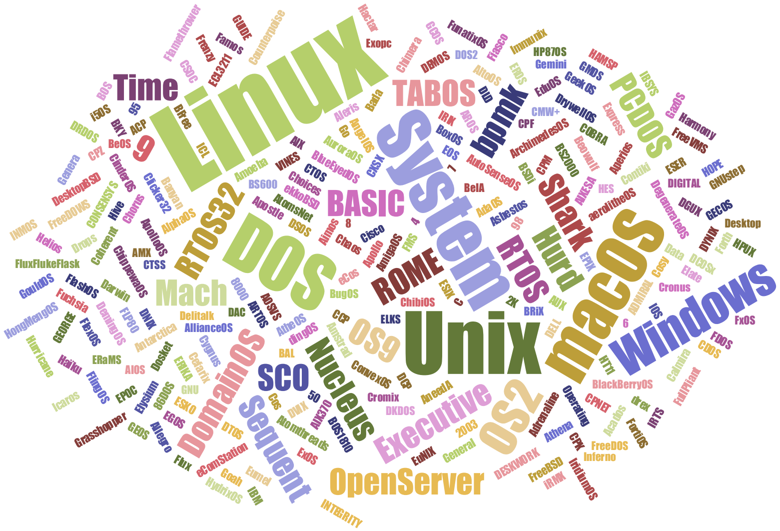
Working Set Model with Timer

- implementation challenges similar to LRU: track how long pages have not been referenced
- algorithm with timer interrupt
  - regular interrupts update the page age information by means of the reference bit:
    - ➔ if reference bit is set (page was used), age is set to zero;
    - ➔ otherwise: age is increased
    - ➔ only the pages of the currently running process are "aged"
  - pages with age > Δ are no longer part of the working set of the respective process
- <u>impractical</u> as to run-time overhead

Thrashing – Solution 3

- solution 3: react on rate of page faults
- thrashing can be more easily avoided by directly controlling the page fault rate
  - page-fault logging per process
  - page-fault rate < threshold: decrease number of page frames
  - page-fault rate > threshold: increase number of page frames

Agenda



- Recap
- Organizational Matters
- Motivation
  - Locality of Reference
  - Virtual Memory
- Demand Paging
- Page Replacement Algorithms
  - First In, First Out (FIFO)
  - Least Recently Used (LRU)
  - Second Chance (SC)
- Page Frame Allocation
  - Thrashing
  - Working Set Model
- **Summary and Outlook**

⏭ Summary and Outlook

- **summary**
  - virtual memory enables the use of large logical address spaces despite limitations of the main memory
  - but: comfort comes at a price
    - ➔ increased complexity and effort at the hardware level
    - ➔ complex algorithms in the operating system → runtime overhead
    - ➔ strategy selection depends on the usage pattern
  - more simple (special-purpose) operating systems that do not necessarily need this "luxury" are better off without it
- **outlook: filesystems**
  - abstraction methods to represent data blocks as files and directories
  - data management and storage, error handling and recovery

References and Acknowledgments

Lecture

- Systemnahe Programmierung in C (SPiC), Betriebssysteme (Jürgen Kleinöder, Wolfgang Schröder-Preikschat)
- Betriebssysteme und Rechnernetze (Olaf Spinczyk, Embedded Software Systems Group, Universität Osnabrück)

Teaching Books and Reference Book

[1] Avi Silberschatz, Peter Baer Galvin, Greg Gagne: *Operating System Concepts*, John Wiley & Sons, 2018.

[2] Andrew Tanenbaum, Herbert Bos: *Modern Operating Systems*, Pearson, 2015.

[3] Wolfgang Schröder-Preikschat: *Grundlage von Betriebssystemen – Sachwortverzeichnis*, 2023.
`https://www4.cs.fau.de/~wosch/glossar.pdf`