Week 8

Memory Management: Demand Paging

Oana Balmau

February 21, 2023

Announcements

Week 8 **Memory Management**	feb 20 C Review: Working with pthreads II	feb 21 **Demand Paging (3/3)** Optional reading: OSTEP Chapters 19 – 22 Practice Exercises Sheet: Memory Management	feb 22	feb 23 Mid-semester Q&A – not recorded • Graded Exercises Sheet Released • Grades released for OS Shell Assignment	feb 24
Week 9 **Reading week**	feb 27 **No class**	feb 28 **No class**	mar 1 **No class**	mar 2 **No class**	mar 3 **No class**
Week 10 **File Systems**	mar 6 No lab. Work on Assignment 2 **Scheduling Assignment Due**	mar 7 **Intro to File Systems (1/2)** **Recorded lecture. Do not come to class.** Optional reading: OSTEP Chapters 36, 37, 39	mar 8 **Memory Management Assignment Released**	mar 9 **Intro to File Systems (2/2)** Memory Management Assignment Overview -– with Jiaxuan	mar 10
Week 11 **File Systems**	mar 13 **Graded Exercises Due** C Review: Complex structs	mar 14 **Basic File System Implementation (1/2)** Optional reading: OSTEP Chapters 40, 41, 45	mar 15	mar 16 **Basic File System Implementation (2/2)** • Grades released for Scheduling Assignment	mar 17
Week 12 **File Systems**	mar 20 C Review: Pointers & Memory Allocation II	mar 21 **Advanced File System Implementation (1/2)**	mar 22	mar 23 **Advanced File System Implementation (2/2)**	mar 24
Week 13 **File Systems**	mar 27 C Review: Advanced debugging	mar 28 **Handling Crashes & Performance (1/2)** Optional reading: OSTEP Chapters 38, 43	mar 29	mar 30 **Handling Crashes & Performance (2/2)** • Grades released for Exercises Sheet • Practice Exercises Sheet: File Systems	mar 31
Week 14 **Advanced Topics**	apr 3 No lab. Work on Assignment 3 **Memory Management Assignment Due**	apr 4 **Advanced topics: Virtualization**	apr 5	apr 6 **Advanced topics: Operating Systems Research** (Invited Speaker: TBD)	apr 7
Week 15 **Wrap-up**	apr 10 No Lab. Prepare for end-of-semester.	apr 11 **End-of-semester Q&A**– not recorded	apr 12	apr 13 **End-of-semester Q&A** -– not recorded. **Last class!** • Grades released for Memory Management Assignment	apr 14

Announcements

Week 8 **Memory Management**	feb 20 C Review: Working with pthreads II	feb 21 **Demand Paging (3/3)** Optional reading: OSTEP Chapters 19 – 22 Practice Exercises Sheet: Memory Management	feb 22	feb 23 Mid-semester Q&A – not recorded • Graded Exercises Sheet Released • Grades released for OS Shell Assignment	feb 24
Week 9 **Reading week**	feb 27 **No class**	feb 28 **No class**	mar 1 **No class**	mar 2 **No class**	mar 3 **No class**
Week 10 **File Systems**	mar 6 No lab. Work on Assignment 2 **Scheduling Assignment Due**	mar 7 **Intro to File Systems (1/2)** **Recorded lecture. Do not come to class.** Optional reading: OSTEP Chapters 36, 37, 39	mar 8 **Memory Management Assignment Released**	mar 9 **Intro to File Systems (2/2)** Memory Management Assignment Overview -– with Jiaxuan	mar 10
Week 11 **File Systems**	mar 13 **Graded Exercises Due** C Review: Complex structs	mar 14 **Basic File System Implementation (1/2)** Optional reading: OSTEP Chapters 40, 41, 45	mar 15	mar 16 **Basic File System Implementation (2/2)** • Grades released for Scheduling Assignment	mar 17
Week 12 **File Systems**	mar 20 C Review: Pointers & Memory Allocation II	mar 21 **Advanced File System Implementation (1/2)**	mar 22	mar 23 **Advanced File System Implementation (2/2)**	mar 24
Week 13 **File Systems**	mar 27 C Review: Advanced debugging	mar 28 **Handling Crashes & Performance (1/2)** Optional reading: OSTEP Chapters 38, 43	mar 29	mar 30 **Handling Crashes & Performance (2/2)** • Grades released for Exercises Sheet • Practice Exercises Sheet: File Systems	mar 31
Week 14 **Advanced Topics**	apr 3 No lab. Work on Assignment 3 **Memory Management Assignment Due**	apr 4 **Advanced topics: Virtualization**	apr 5	apr 6 **Advanced topics: Operating Systems Research** (Invited Speaker: TBD)	apr 7
Week 15 **Wrap-up**	apr 10 No Lab. Prepare for end-of-semester.	apr 11 **End-of-semester Q&A– not recorded**	apr 12	apr 13 **End-of-semester Q&A -– not recorded. Last class!** • Grades released for Memory Management Assignment	apr 14

Announcements

Week 8 **Memory Management**	feb 20 C Review: Working with pthreads II	feb 21 **Demand Paging (3/3)** Optional reading: OSTEP Chapters 19 – 22 Practice Exercises Sheet: Memory Management	feb 22	feb 23 Mid-semester Q&A – not recorded • Graded Exercises Sheet Released • Grades released for OS Shell Assignment	feb 24
Week 9 **Reading week**	feb 27 **No class**	feb 28 **No class**	mar 1 **No class**	mar 2 **No class**	mar 3 **No class**
Week 10 **File Systems**	mar 6 No lab. Work on Assignment 2 **Scheduling Assignment Due**	mar 7 **Intro to File Systems (1/2)** **Recorded lecture. Do not come to class.** Optional reading: OSTEP Chapters 36, 37, 39	mar 8 **Memory Management Assignment Released**	mar 9 **Intro to File Systems (2/2)** Memory Management Assignment Overview – with Jiaxuan	mar 10
Week 11 **File Systems**	mar 13 **Graded Exercises Due** C Review: Complex structs	mar 14 **Basic File System Implementation (1/2)** Optional reading: OSTEP Chapters 40, 41, 45	mar 15	mar 16 **Basic File System Implementation (2/2)** • Grades released for Scheduling Assignment	mar 17
Week 12 **File Systems**	mar 20 C Review: Pointers & Memory Allocation II	mar 21 **Advanced File System Implementation (1/2)**	mar 22	mar 23 **Advanced File System Implementation (2/2)**	mar 24
Week 13 **File Systems**	mar 27 C Review: Advanced debugging	mar 28 **Handling Crashes & Performance (1/2)** Optional reading: OSTEP Chapters 38, 43	mar 29	mar 30 **Handling Crashes & Performance (2/2)** • Grades released for Exercises Sheet • Practice Exercises Sheet: File Systems	mar 31
Week 14 **Advanced Topics**	apr 3 No lab. Work on Assignment 3 **Memory Management Assignment Due**	apr 4 **Advanced topics: Virtualization**	apr 5	apr 6 **Advanced topics: Operating Systems Research** (Invited Speaker: TBD)	apr 7
Week 15 **Wrap-up**	apr 10 No Lab. Prepare for end-of-semester.	apr 11 **End-of-semester Q&A**– not recorded	apr 12	apr 13 **End-of-semester Q&A** – not recorded. **Last class!** • Grades released for Memory Management Assignment	apr 14

Demand Paging

Remember from Last Week
Simplifying Assumption

"For this week's lecture only

All of a program must be in memory

Will revisit assumption next week"

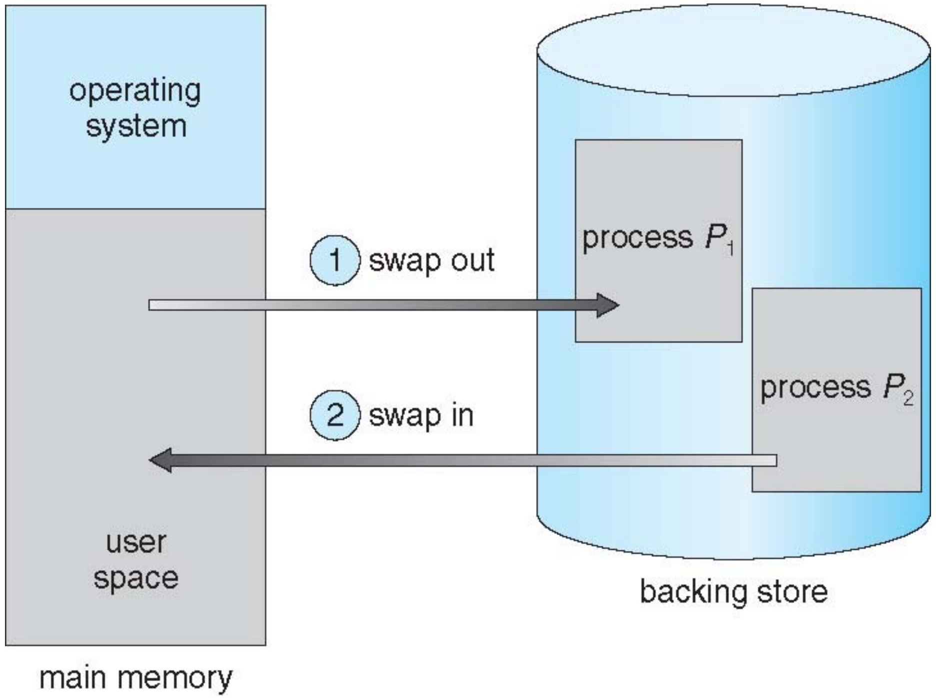
We are now going to drop this assumption

What if “out of memory”?

Need to get rid of one or more processes

Store them temporarily on disk

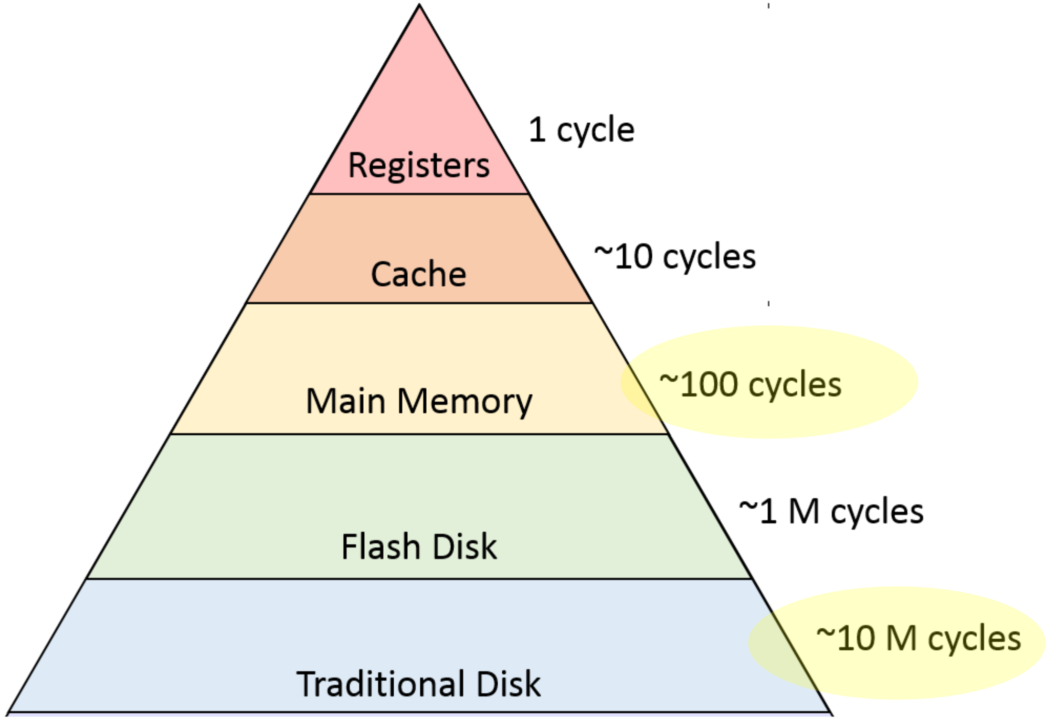
This is called **Swapping**



Process Switch to a Swapped Process?

Latency can be **very high**

Need to read image from disk



Process Switch to a Swapped Process?

A better solution:

- **Demand paging**
- **Not all of a process needs to be in memory**

Main Reason for Demand Paging

Virtual address spaces >> physical address space

- No machine has 2^64 bytes (16 exabytes) of memory (yet).

Why such large virtual address space?

- Convenient for programmer
- Don’t have to worry about running out

Additional Benefits of Demand Paging

+ Shorter process startup latency
    - Can start process without all of it in memory
    - Even 1 page suffices
+ Better use of main memory
    - Program often does not use certain parts
        - E.g., error handling routines
    - Program often goes through different parts
        - E.g., initialization, computation, termination

If the program is not in memory, then where is it?

Part of it is in memory

(Typically) all of it is on disk

- In a special partition called the **Backing Store**

If the program is not in memory, then where is it?

Part of it is in memory

(Typically) all of it is on disk

- In a special partition called the **Backing Store**

Note the difference with swapping:

Swapping = ***all*** of program is **in memory OR *all*** of program is **on disk**

Demand paging = **part** of program is **in memory**

If the program is not in memory, then where is it?

Remember:

- CPU can only directly access memory
- **CPU can only access data on disk through OS**

Part of it is in memory

(Typically) all of it is on disk

- In a special partition called the **Backing Store**

Note the difference with swapping:

Swapping = *all* of program is **in memory OR *all*** of program is **on disk**

Demand paging = part of program is **in memory**

Demand Paging Mechanism High Level

- What if program needs to accesses part only on disk?

Demand Paging Mechanism High Level

- What if program needs to accesses part only on disk?

- Program is suspended
- OS runs, gets page from disk
- Program is restarted

Demand Paging Mechanism High Level

- What if program needs to accesses part only on disk?

  This is called a **page fault**

- Program is suspended
- OS runs, gets page from disk
- Program is restarted

  This is called **page fault handling**

Demand Paging Issues

1. How to discover a page fault?

2. How to suspend process?

3. How to get a page from disk?

    3’. How to find a free frame in memory?

4. How to restart process?

1. Discover Page Fault

Idea: Use the valid bit in page table

- Without demand paging:
  - Valid bit = 0: page is invalid
  - Valid bit = 1: page is valid

- **With demand paging**
  - Valid bit = 0: page is invalid **OR page is on disk**
  - Valid bit = 1: page is valid **AND page is in memory**
  - OS needs additional table: invalid / on-disk?

2. Suspending the Faulting Process

Idea: Trap into the OS

- Invalid bit access **generates trap**
- Save process information in PCB when trapping into the OS

3. Getting the Page from Disk

Idea: OS handles fetch from Disk

- Assume (for now) there is at least one free frame in memory
- Allocate a free frame to process
- Find page on disk
  - OS needs to remember the swap space in page-sized units.
  - Need an extra table for that in OS.
- Get disk to transfer page from disk to frame

3. Getting the Page from Disk

Idea: OS handles fetch from Disk

- Assume (for now) there is at least one free frame in memory
- Allocate a free frame to process
- Find page on disk
  - OS needs to remember the swap space in page-sized units.
  - Need an extra table for that in OS.
- Get disk to transfer page from disk to frame **slow…**

3. While the Disk is Busy

- Shouldn’t waste CPU cycles
- Invoke scheduler to run another process
- When **disk interrupt** arrives
  - Suspend running process
  - Get **back to page fault handling**

3. While the Disk is Busy

Why OK to go through OS (and not in hardware)?

- Disk is so slow that it masks the latency of handling this in OS.

3. Completing Page Fault Handling

- Pagetable[pageno].frameno = new frameno
- **Pagetable[pageno].valid = 1**

- Set process state to ready
- Invoke scheduler

4. When Process Runs Again

Idea: Restarts the previously faulting instruction

Process now finds

- Valid bit to be set to 1
- Page in corresponding frame in memory
- Note: different from context switch, which continues with the next instruction

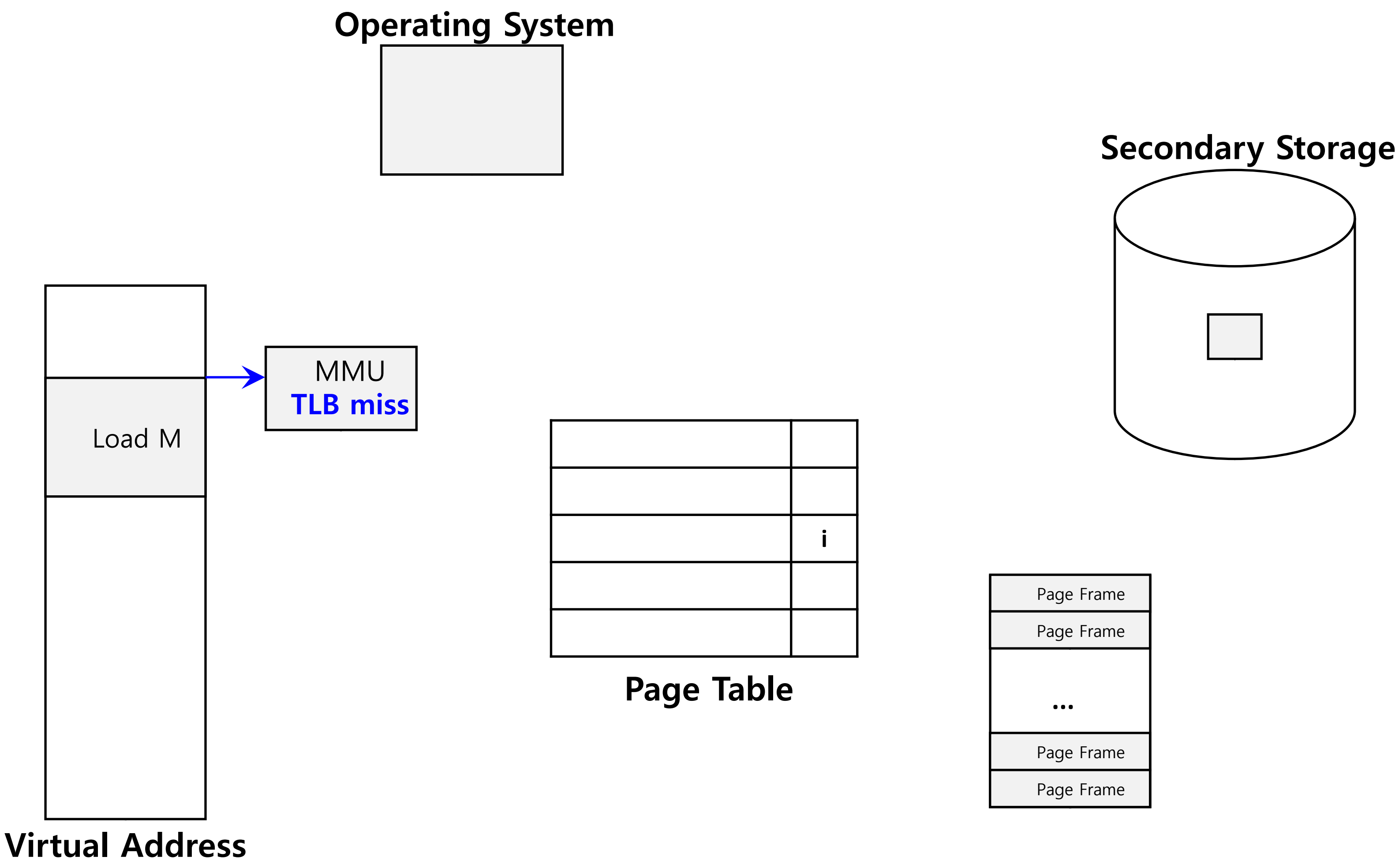
Demand Paging Summary

1. How to discover a page fault? ← **Use valid bit in page table**
2. How to suspend process? ← **Invalid bit traps into OS**
3. How to get a page from disk? ← **OS handles fetch from disk**

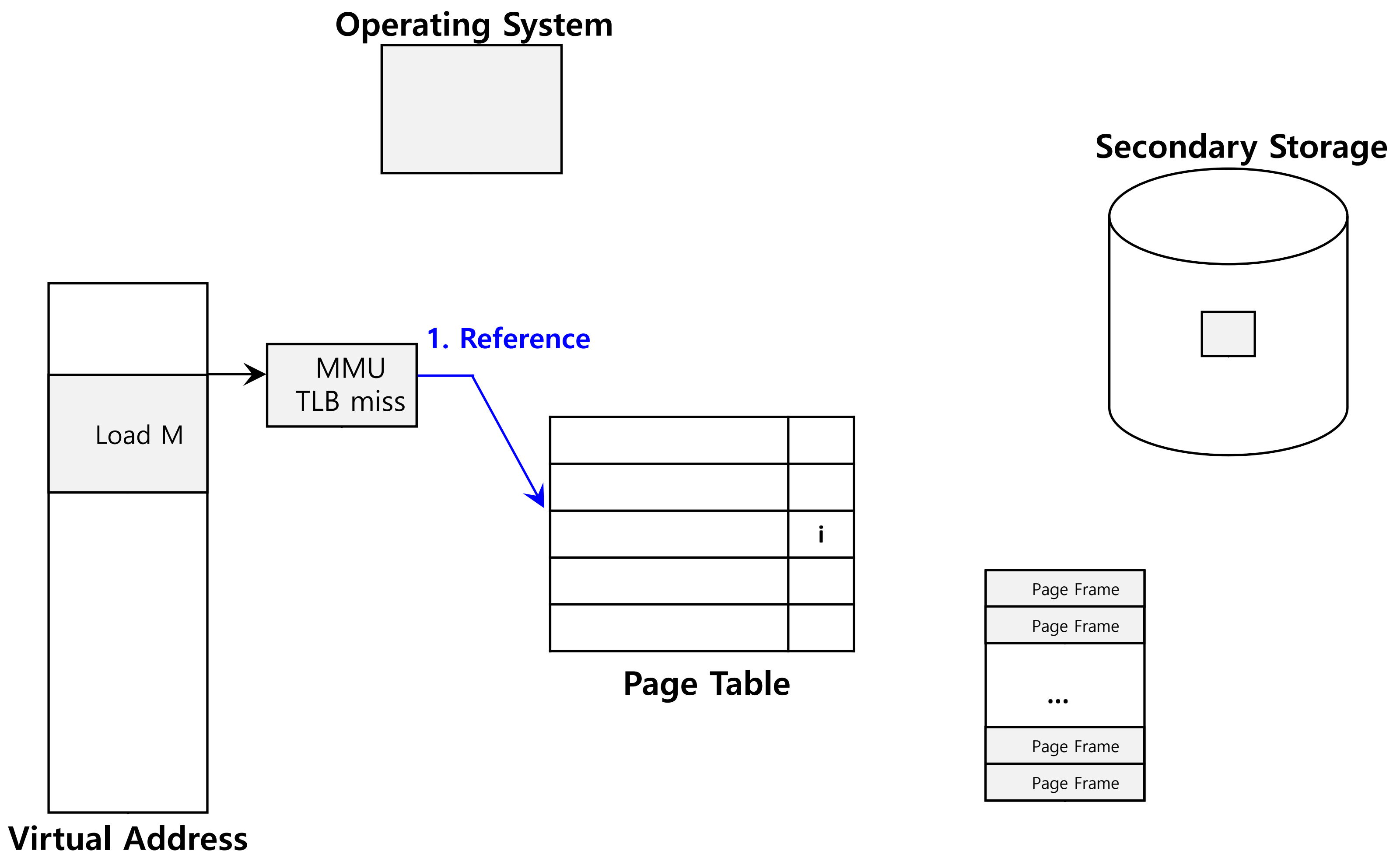
   3'. How to find a free frame in memory?

4. How to restart process? ← **Restart faulting instruction**

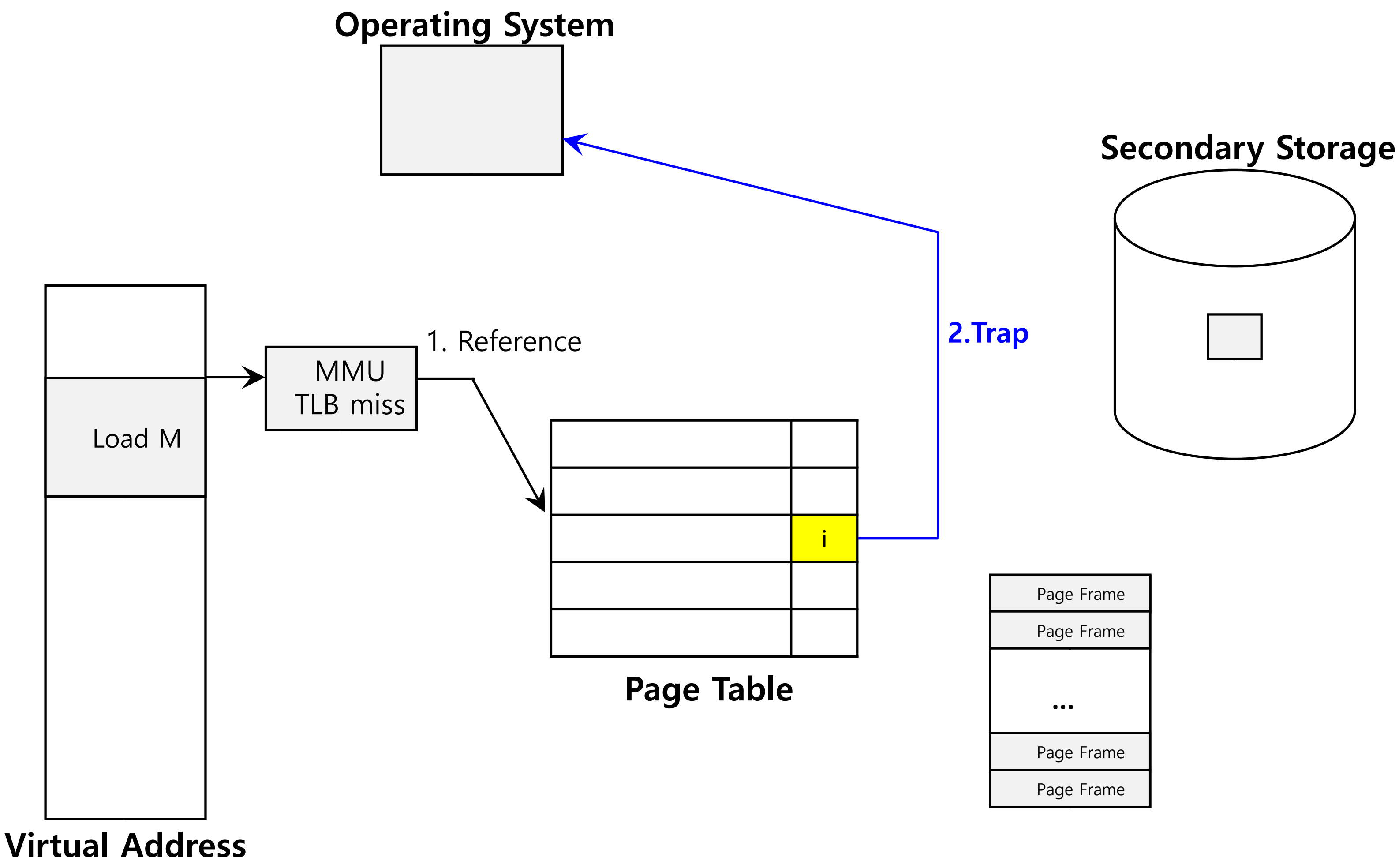
Demand Paging Summary



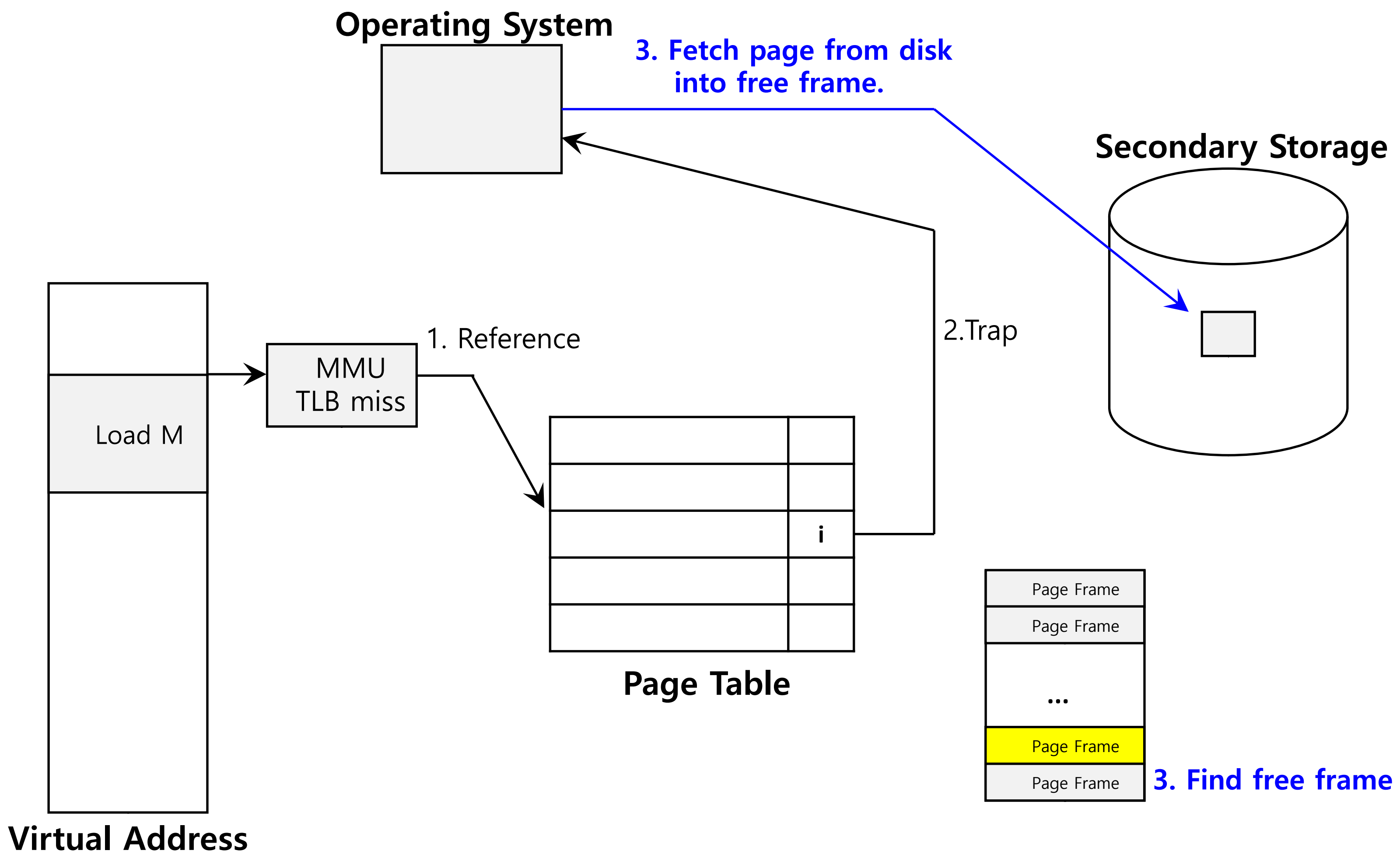
Demand Paging Summary



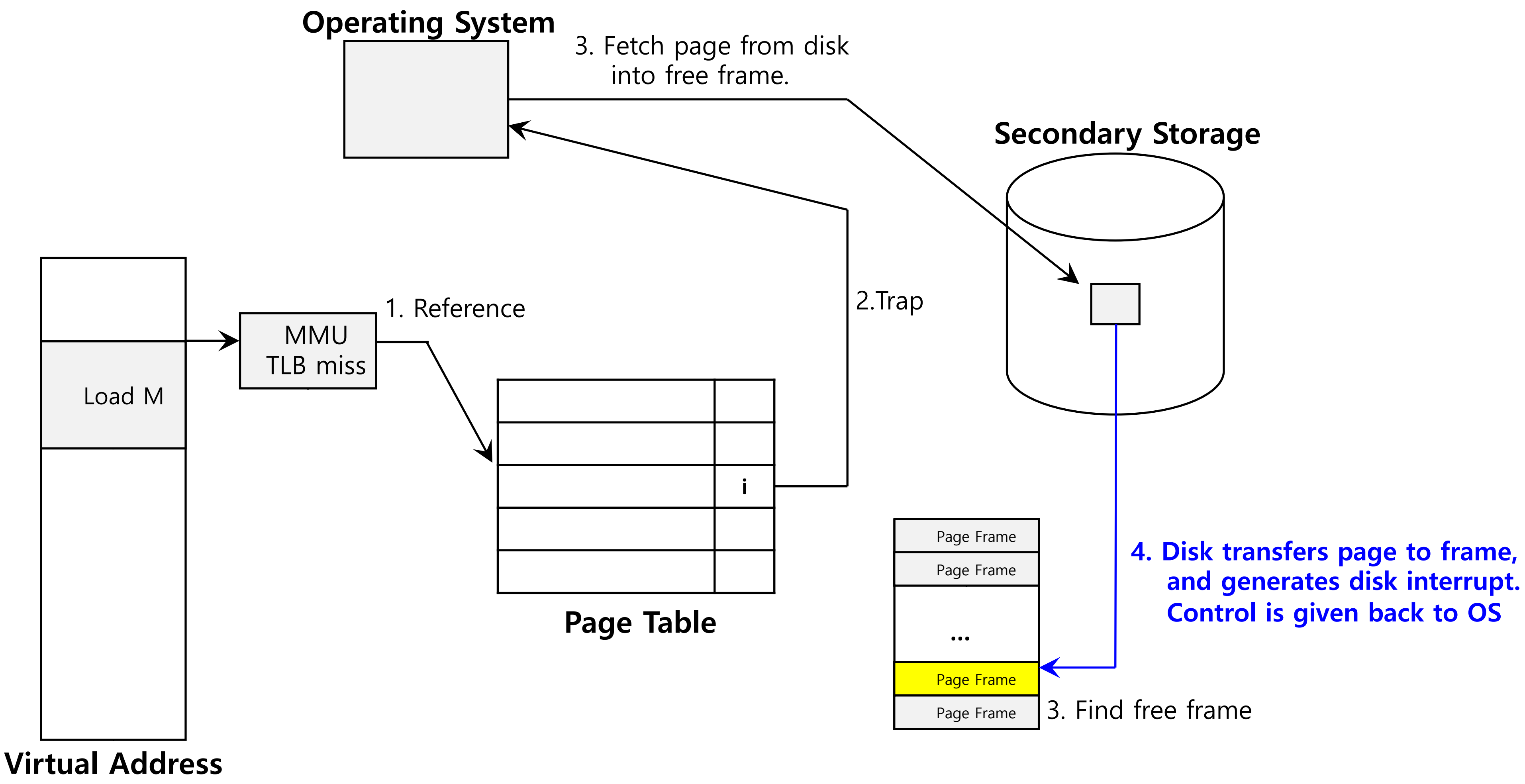
Demand Paging Summary



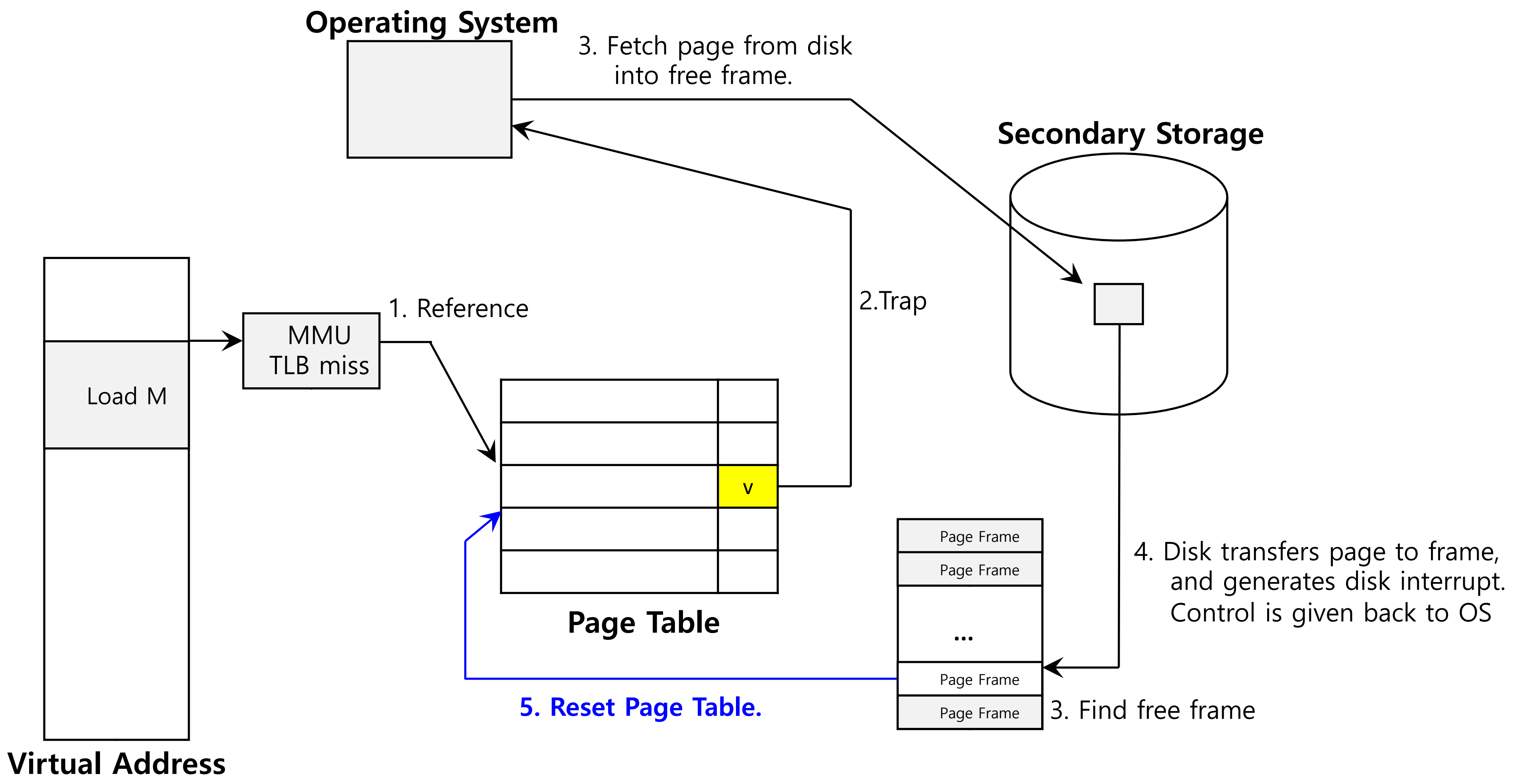
Demand Paging Summary



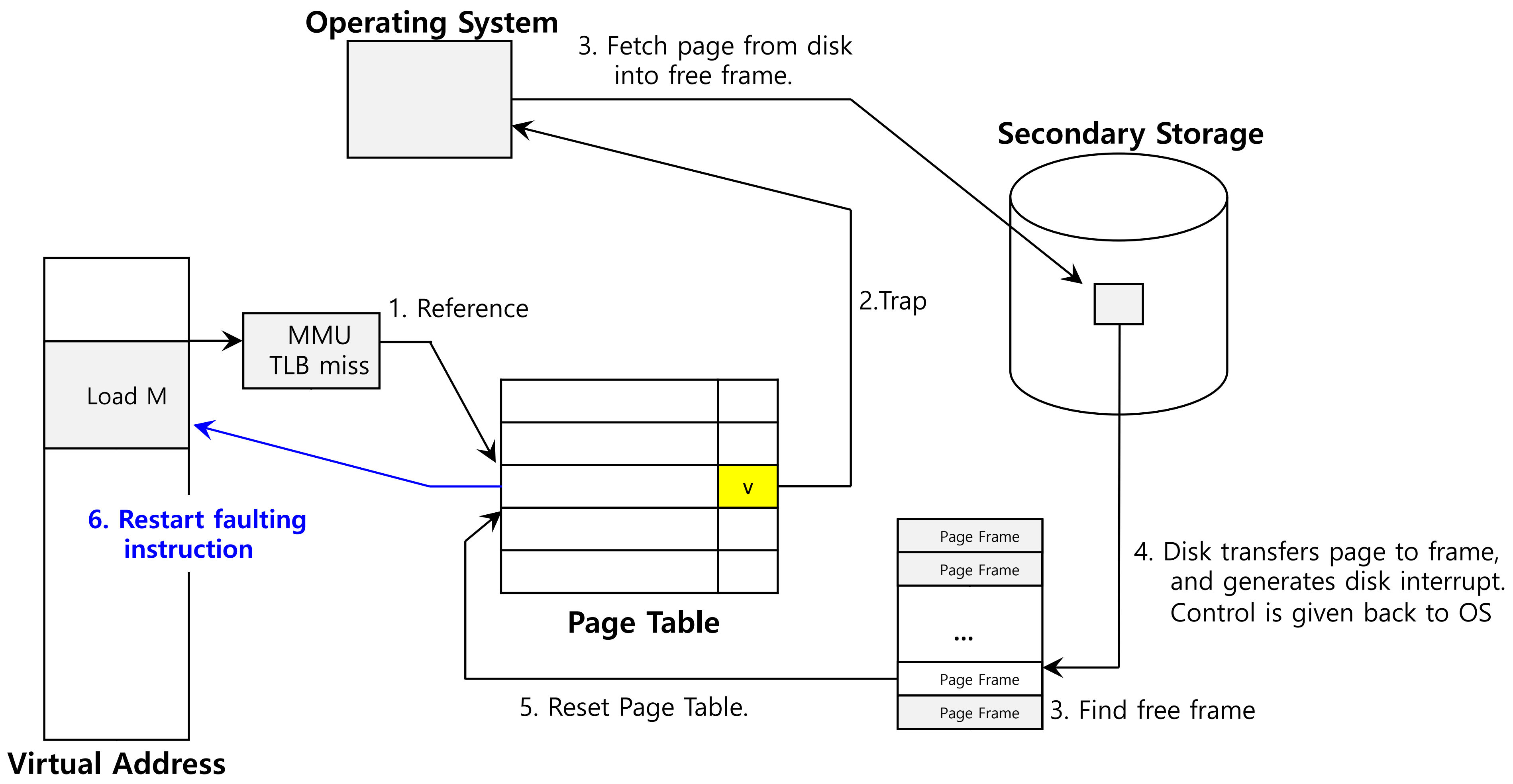
Demand Paging Summary



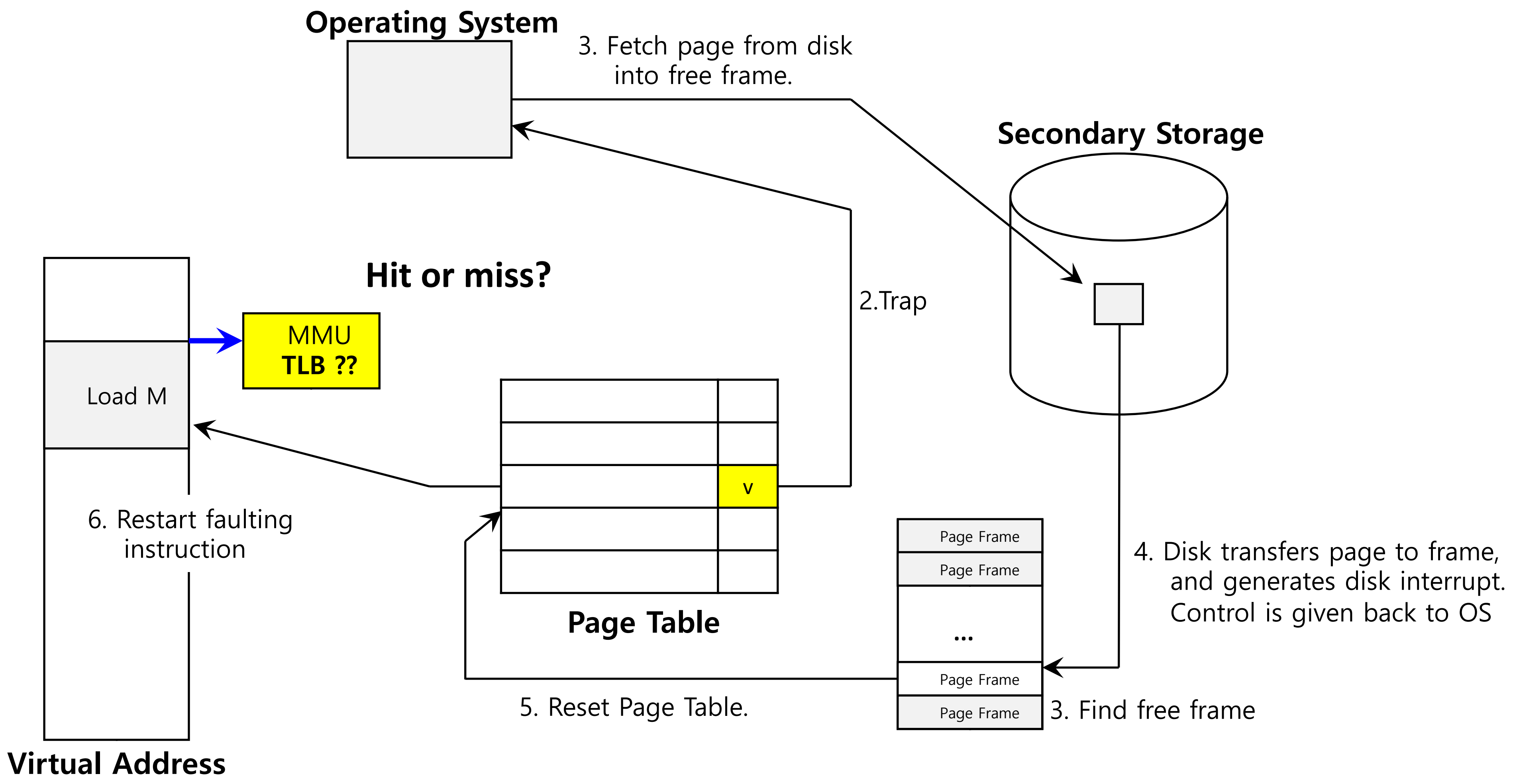
Demand Paging Summary



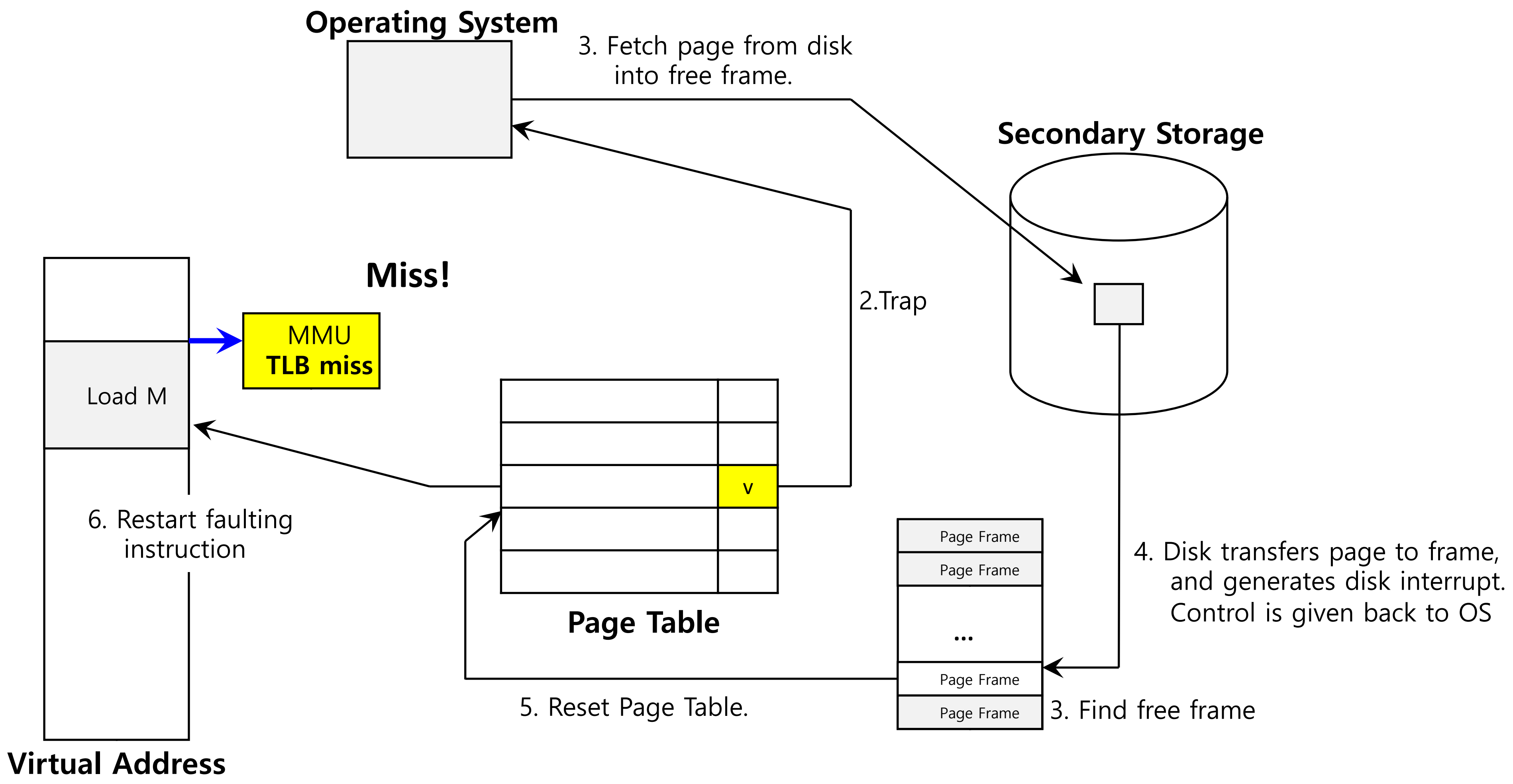
Demand Paging Summary



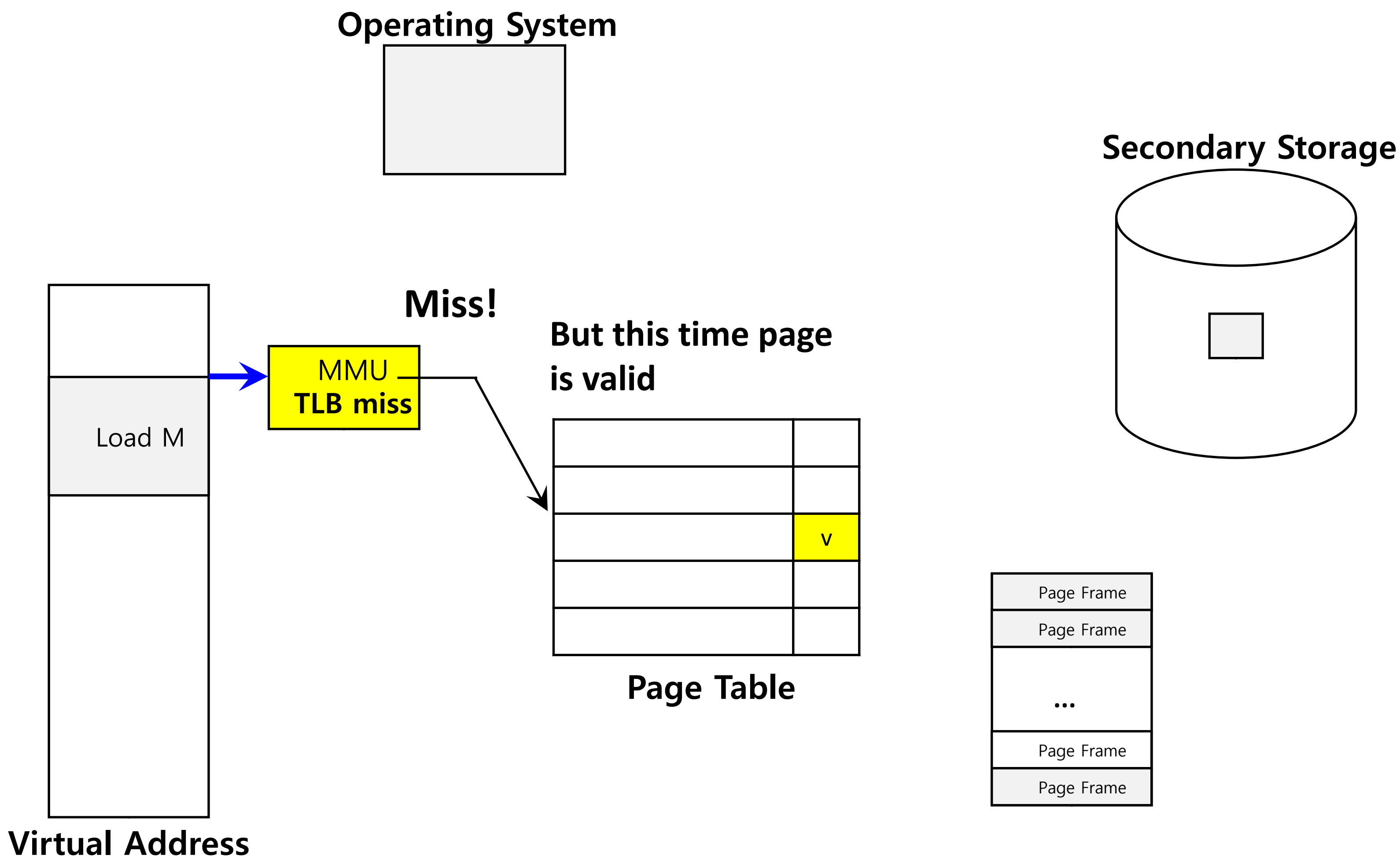
Demand Paging Summary



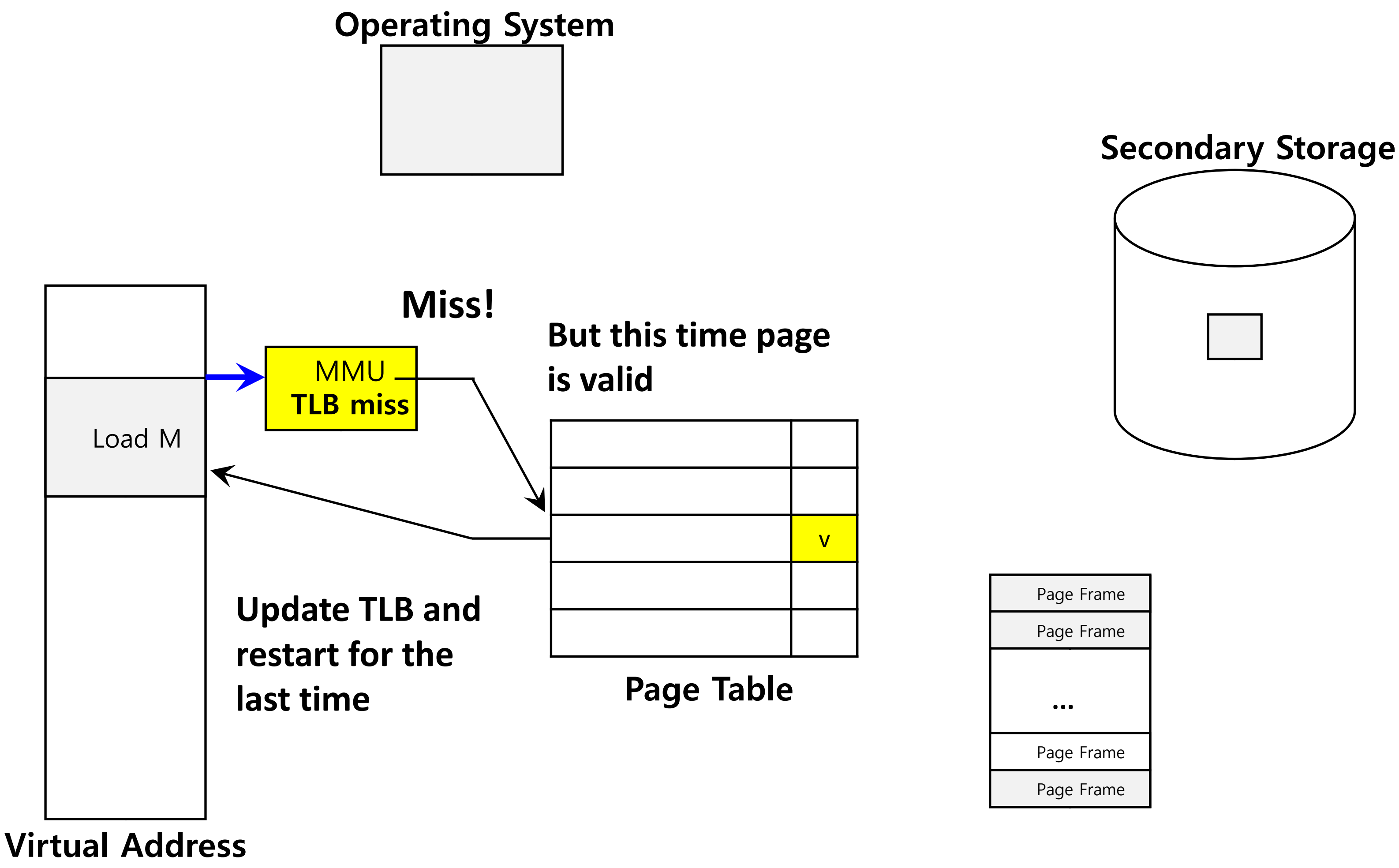
Demand Paging Summary



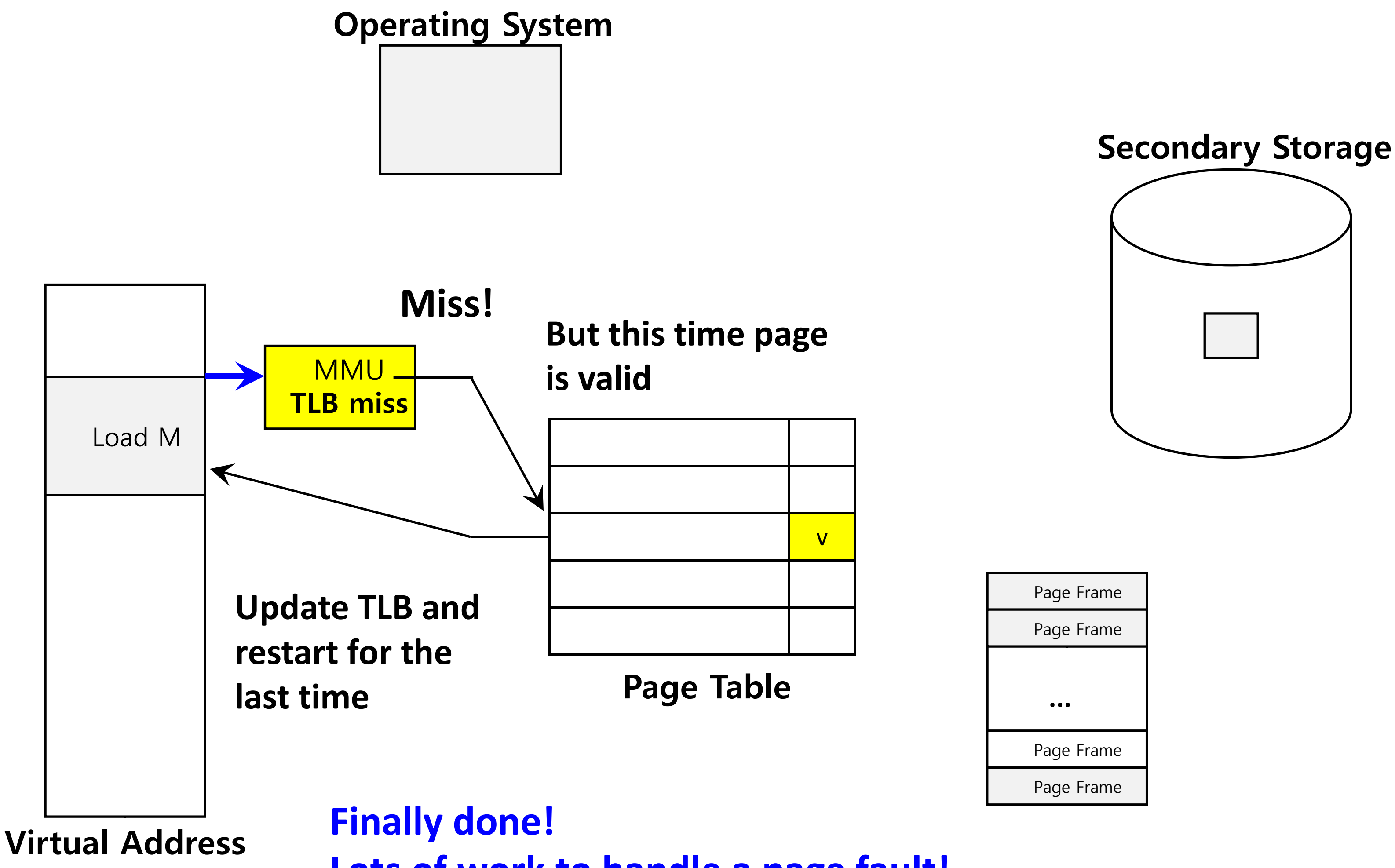
Demand Paging Summary



Demand Paging Summary



Demand Paging Summary



Finally done!
Lots of work to handle a page fault!

Remember Assumption: Getting the Page from Disk

"Assume (for now) there is at least one free frame in memory"

If no free frame available:

- Pick a frame to be replaced
- Invalidate its page table entry (and TLB entry)
- You may have to write that frame to disk

Remember Assumption: Getting the Page from Disk

"Assume (for now) there is at least one free frame in memory"

If no free frame available:

- Pick a frame to be replaced
- Invalidate its page table entry (and TLB entry)
- You may have to write that frame to disk
  - Page table has a modified bit
  - If set, write out page to disk
  - If not, proceed with page fault handling

Remember Assumption: Getting the Page from Disk

"Assume (for now) there is at least one free frame in memory"

If no free frame available:

- Pick a frame to be replaced
  - **How?**
- Invalidate its page table entry (and TLB entry)
- You may have to write that frame to disk
  - Page table has a modified bit
  - If set, write out page to disk
  - If not, proceed with page fault handling

How to pick with page/frame to replace?

Different page replacement policies:

- Random
- FIFO (First In, First Out)
- OPT
- LRU

Page Faults and Performance

- Normal memory access
  - ~ nanoseconds
- Faulting memory access
  - Disk i/o ~ 10 milliseconds

- Too many page faults **-> program very slow**

- **Hence, importance of good page replacement policy**

Page Replacement Policies

- Random
- FIFO (First In, First Out)
- OPT
- LRU

Page Replacement Policies

- Random
- FIFO (First In, First Out)
- OPT
- LRU

Plus, in general, prefer replacing clean over dirty
- 1 disk i/o instead of 2

Random

Random page is replaced

+ Easy to implement

☹ Does not take advantage of spatial/temporal locality.

FIFO

Oldest page is replaced

- Age = Time since brought into memory

\+ Easy to implement

- Keep a queue of pages
- Bring in a page: stick at the end of the queue
- Need replacement: pick head of queue

\+ Fair

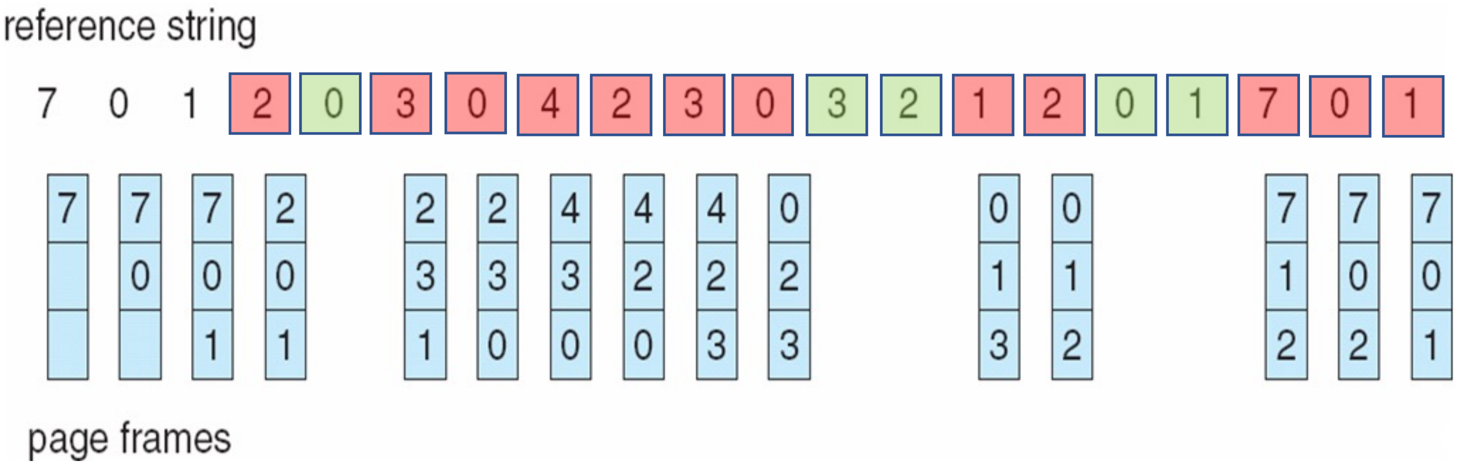
- All pages receive equal residency

☹ Does not take into account “hot” pages that may always be needed

Quiz: FIFO Page Replacement

?? page faults (not counting initial paging in)

Quiz: FIFO Page Replacement



12 page faults (not counting initial paging in)

OPT: An Optimal Algorithm

Replace the page that will be referenced the furthest in the future

+ Provably optimal

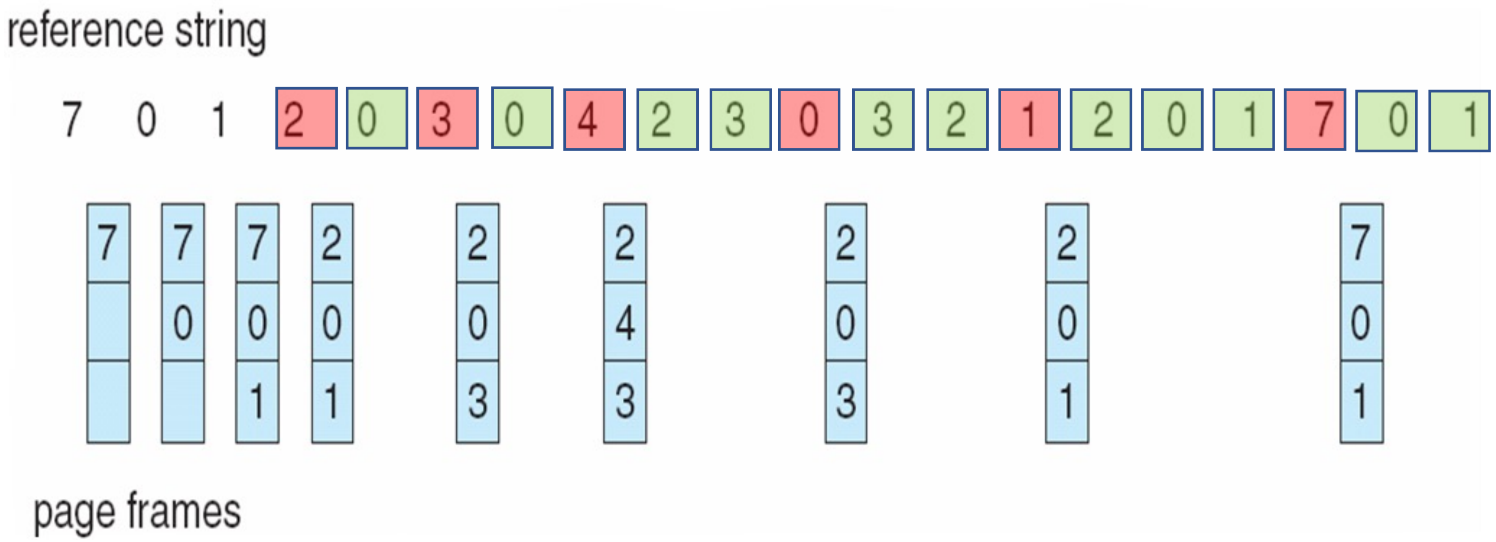
☹ Can’t implement (Can’t predict the future)

A basis of comparison for other algorithms

Quiz: Optimal Page Replacement

?? page faults (not counting initial paging in)

Quiz: Optimal Page Replacement



6 page faults (not counting initial paging in)

LRU: Least Recently Used

Cannot look into the future, but can try to predict future using past

Replace least recently accessed page

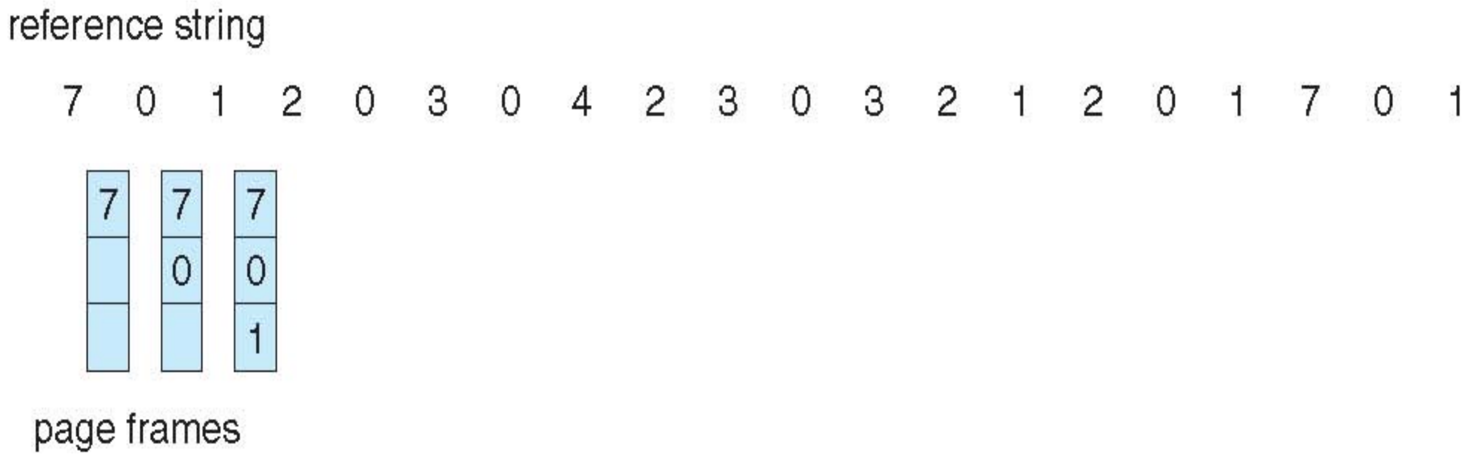
+ With locality, LRU approximates OPT

☹ Harder to implement, must track which pages have been accessed

☹ Does not handle all workloads well

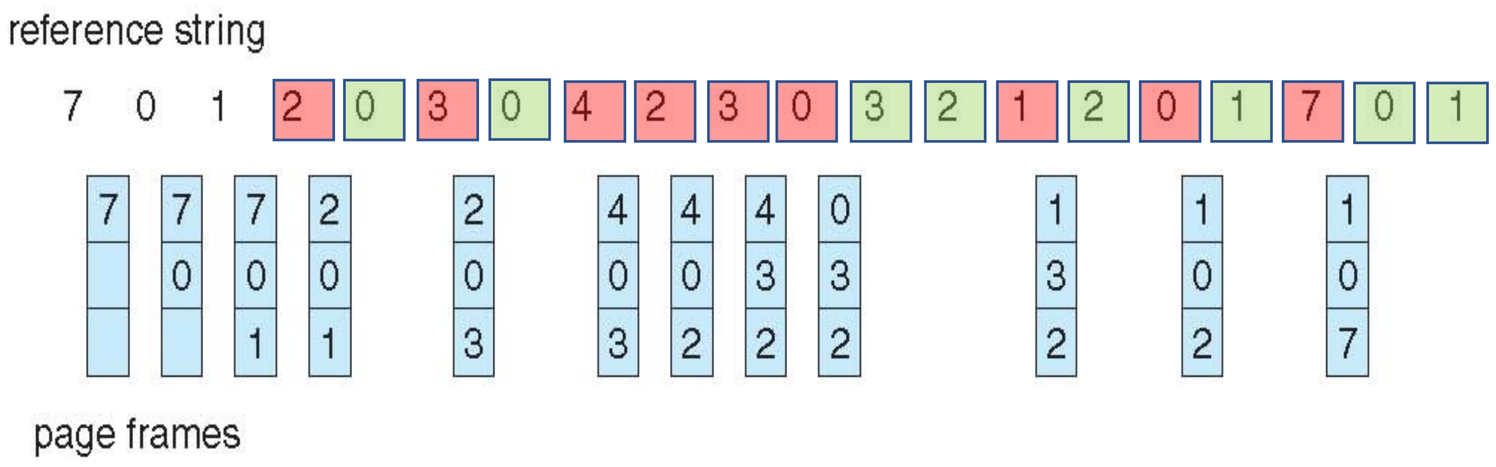
- Example: Large array scans that repeat. Popular in DBMS.

LRU Page Replacement



?? page faults (not counting initial paging in)

LRU Page Replacement



9 page faults (not counting initial paging in)

LRU Implementation

- Too expensive to implement exactly
  - Need to timestamp every memory reference

- But can be (well) approximated

LRU Approximation with Hardware Support

Use a **reference bit**

- Bit in page table
- Hardware sets bit when page is referenced

Periodically

- Read out and store all reference bits
- Reset all reference bits to zero

Keep all reference bits for some time

- The more bits kept, the better approximation

Replacement

- **Page with smallest value of reference bit history**

Let’s practice!

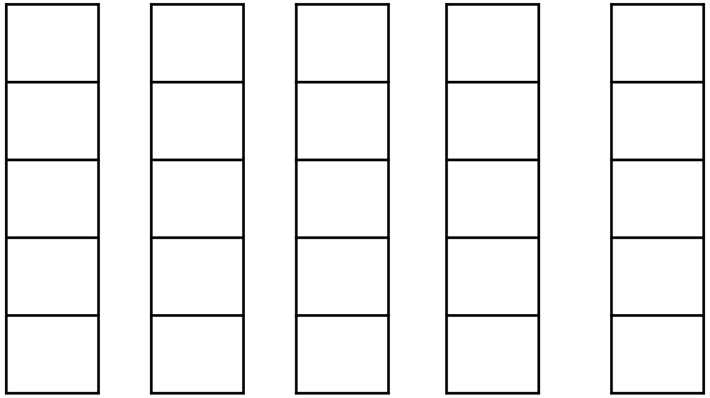
Page Replacement Policies

Consider a **cache of size 5.**

1. Generate worst-case address reference streams for FIFO and LRU. Worst-case reference streams cause the most misses possible.
2. Compare with OPT in all scenarios from point 1.
3. For the worst case reference streams, how much bigger of a cache is needed to improve performance dramatically and approach OPT?

Page Replacement Policies

Worst case FIFO:



Page Replacement Policies

Worst case FIFO:

1 2 3 4 5 6 1 2 3 4 5 6

1	**1**	**1**	**1**	**1**
	2	**2**	**2**	**2**
		3	**3**	**3**
			4	**4**
				5

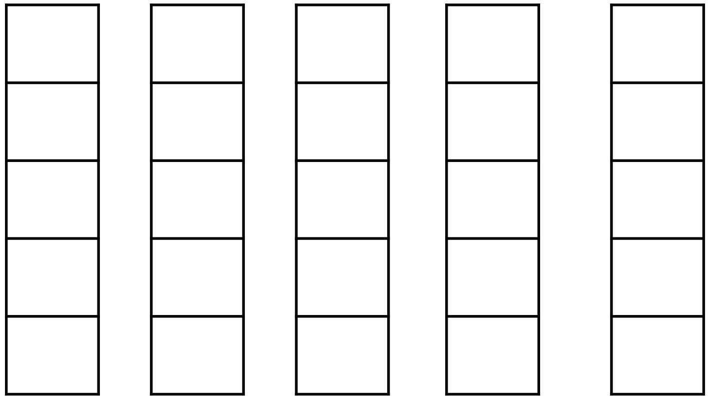
Page Replacement Policies

Worst case FIFO:

1	2	3	4	5	6	1	2	3	4	5	6
1	**1**	**1**	**1**	**1**	**6**	**6**	**6**	**6**	**6**	**5**	**5**
	2	**2**	**2**	**2**	**2**	**1**	**1**	**1**	**1**	**1**	**6**
		3	**3**	**3**	**3**	**3**	**2**	**2**	**2**	**2**	**2**
			4	**4**	**4**	**4**	**4**	**3**	**3**	**3**	**3**
				5	**5**	**5**	**5**	**5**	**4**	**4**	**4**

Page Replacement Policies

Worst case LRU:



Page Replacement Policies

Worst case LRU:

1 2 3 4 5 6 1 2 3 4 5 6

1	1	1	1	1
	2	2	2	2
		3	3	3
			4	4
				5

Page Replacement Policies

Worst case LRU:

1	2	3	4	5	6	1	2	3	4	5	6
1	1	1	1	1	6	6	6	6	6	5	5
	2	2	2	2	2	1	1	1	1	1	6
		3	3	3	3	3	2	2	2	2	2
			4	4	4	4	4	3	3	3	3
				5	5	5	5	5	4	4	4

Page Replacement Policies

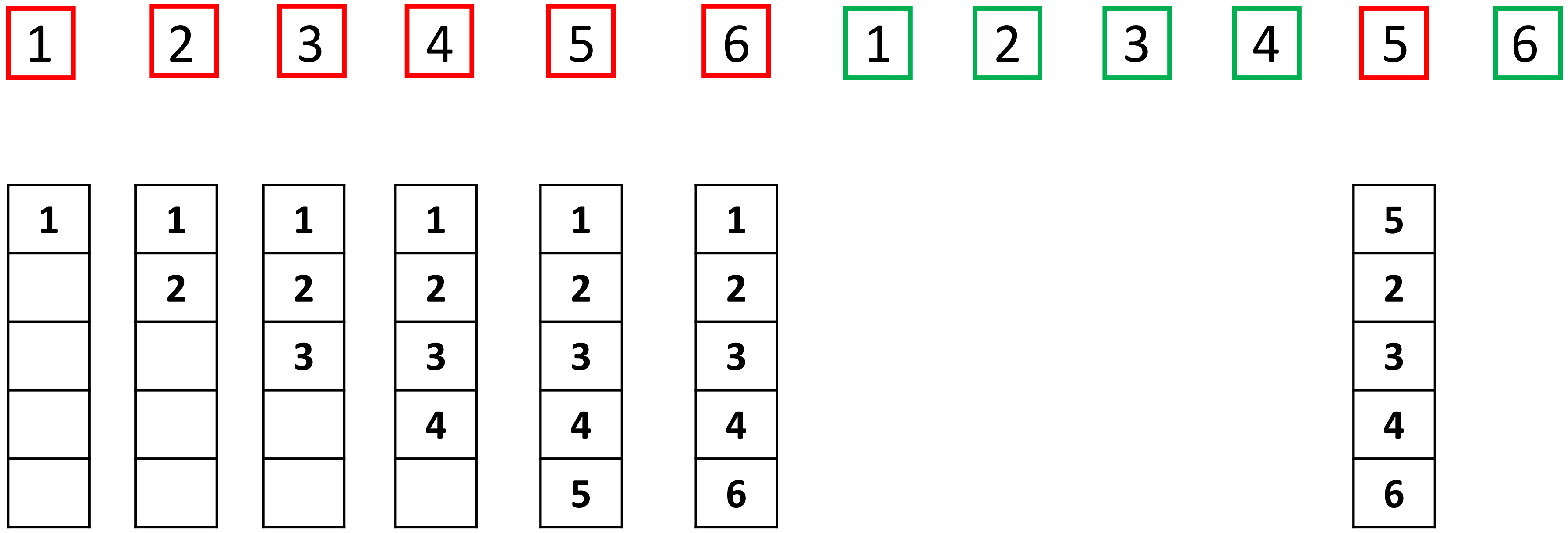
OPT:

$\boxed{1}$ $\boxed{2}$ $\boxed{3}$ $\boxed{4}$ $\boxed{5}$ 6 1 2 3 4 5 6

1	**1**	**1**	**1**	**1**
	2	**2**	**2**	**2**
		3	**3**	**3**
			4	**4**
				5

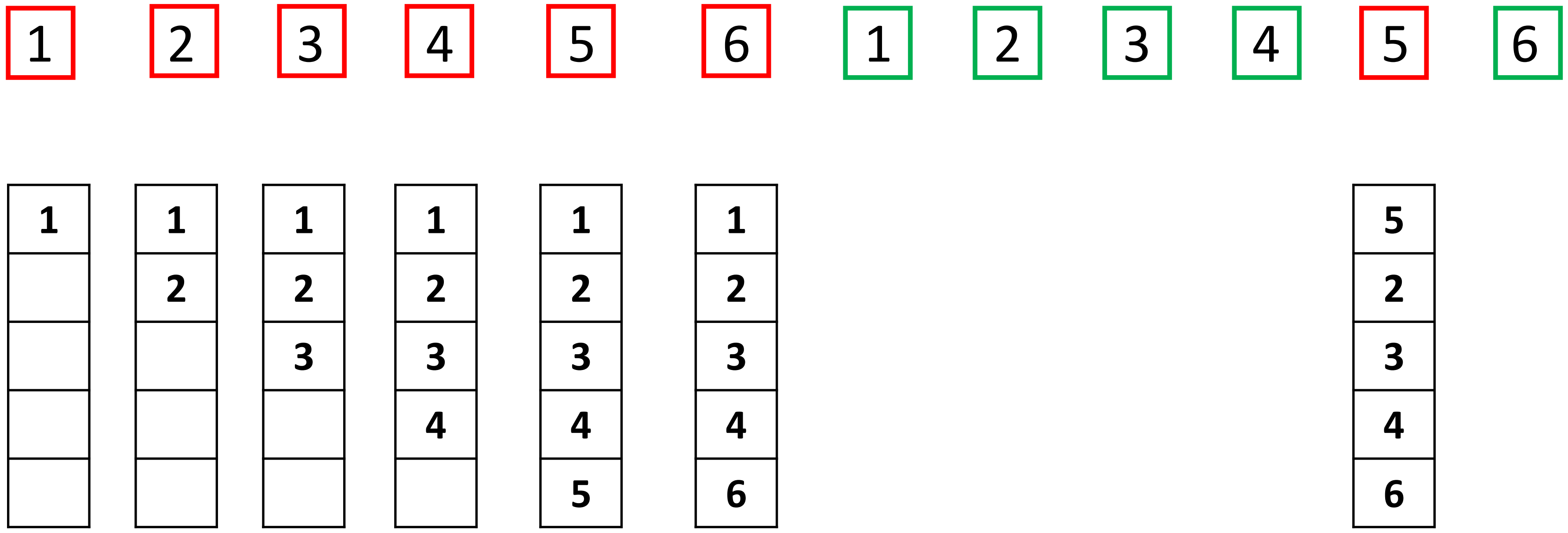
Page Replacement Policies

OPT:



Page Replacement Policies

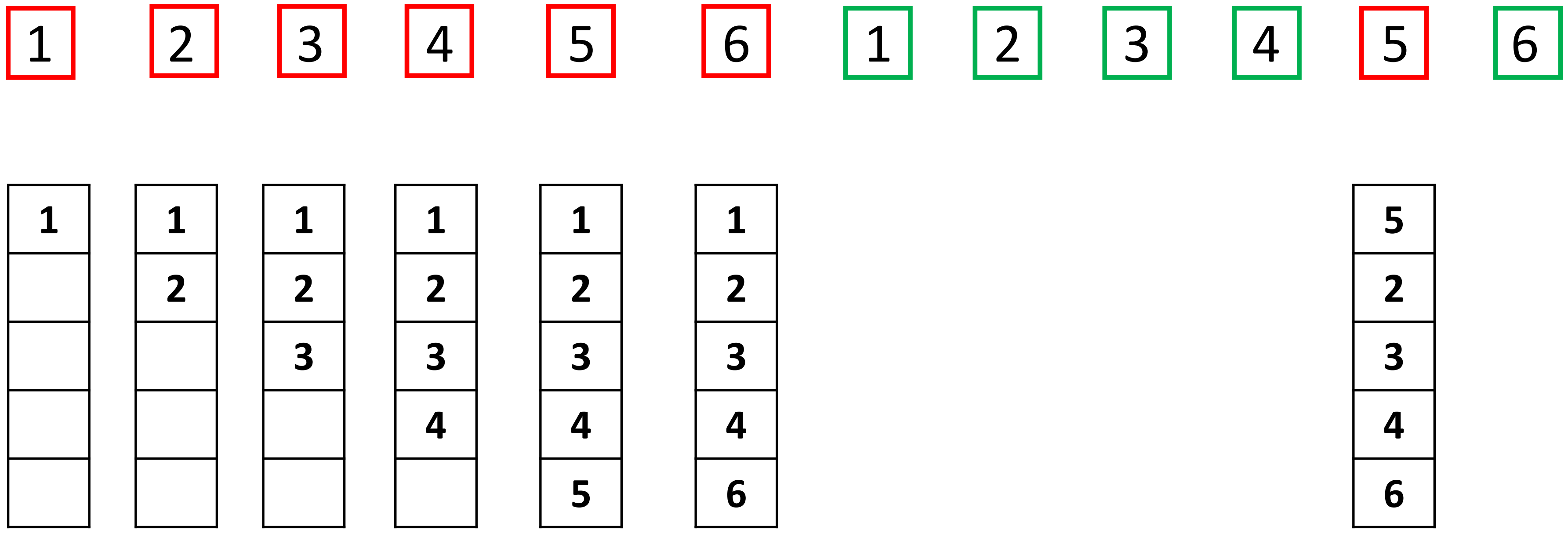
OPT:



For the worst case reference streams, how much bigger of a cache is needed to improve performance dramatically and approach OPT?

Page Replacement Policies

OPT:

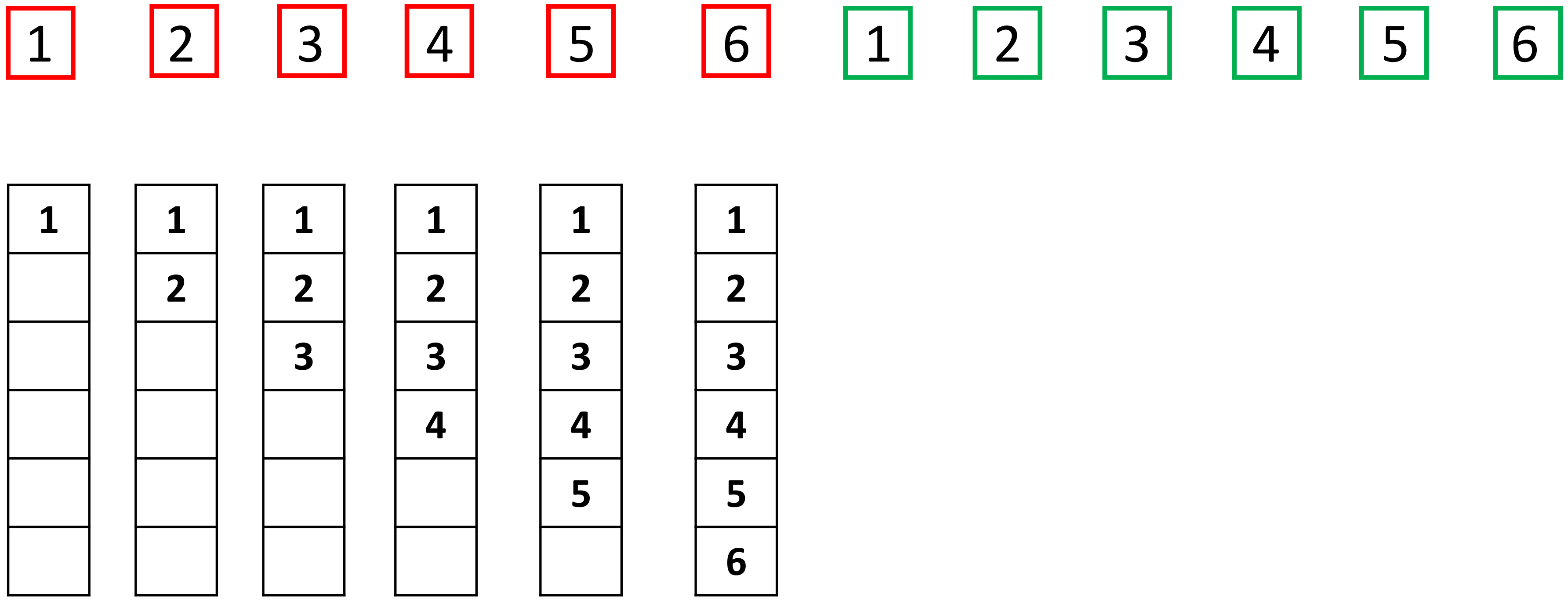


For the worst case reference streams, how much bigger of a cache is needed to improve performance dramatically and approach OPT?

→ Need a cache of 6

Page Replacement Policies

FIFO/LRU, cache of 6:



Summary – Key Concepts

- TLB
- Page Table for very large address spaces
- Demand paging
  - Page fault
- Page replacement policies:
  - Random
  - FIFO (First In, First Out)
  - OPT
  - LRU (Least Recently Used)

Further Reading

Operating Systems: Three Easy Pieces by R. & A. Arpaci-Dusseau

Chapters 19–22

https://pages.cs.wisc.edu/~remzi/OSTEP/

Credits:

Some slides adapted from the OS courses of Profs. Remzi and Andrea Arpaci-Dusseau (University of Wisconsin-Madison), Prof. Willy Zwaenepoel (University of Sydney), and Prof. Youjip Won (Hanyang University).