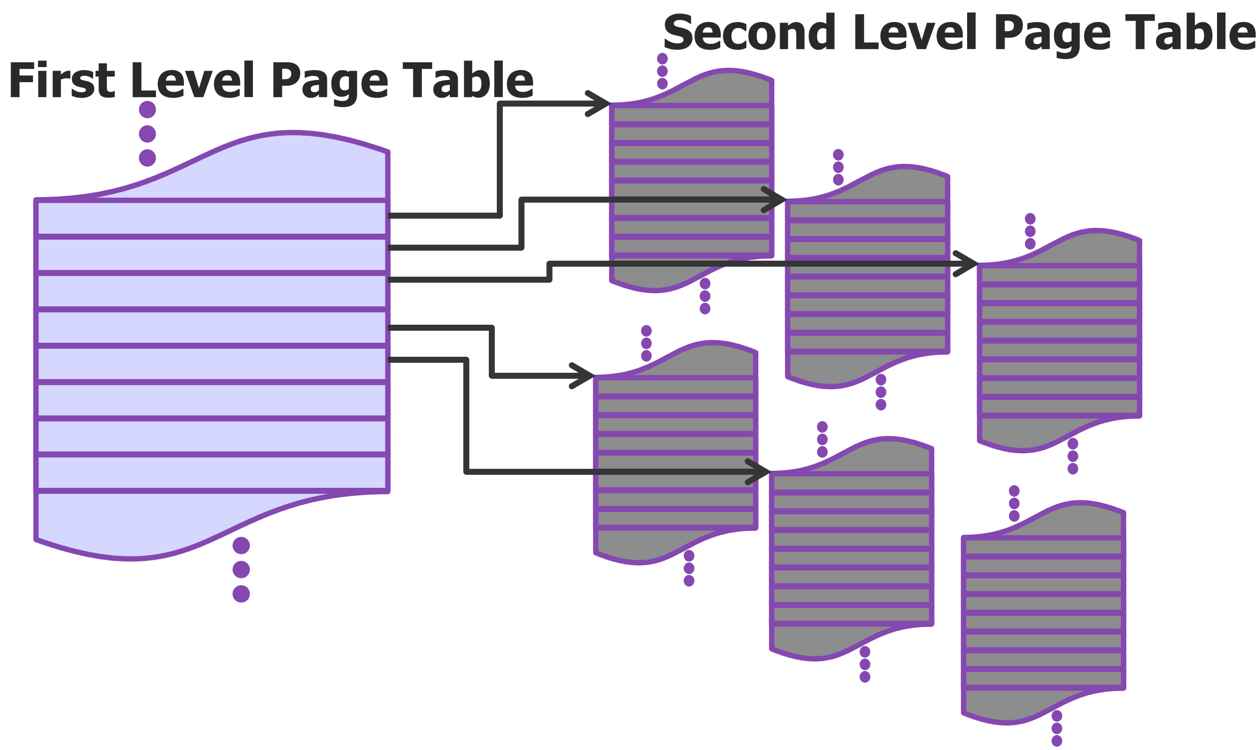
Virtual Memory Wrap Up

Multi-Level Page Tables

- Multiple levels of tables are used to look up a physical memory address.



Multi-Level Page Tables

- Each virtual address can now be divided into (n+1) different pieces for an (n) level page table.
  - **Example:** Two Level Page Table:
    - First Level Page Number (directory)
    - Second Level Page Number (page)
    - Page Offset (offset)

Multi-Level Page Tables: class exercise

- Given
  - 32-bit Virtual Addresses
  - 4 KB Pages
  - 12-bit First Level Page Number (directory)

- What are the components of the address:

  `0x48503423`

Multi-Level Page Tables: class exercise

- Given
  - 32-bit Virtual Addresses
  - 4 KB Pages
  - 12-bit First Level Page Number (directory)

- What are the components of the address:

  `0x48503423`

- **`0x485(directory), 0x03(page), 0x423(offset)`**

Multi-Level Page Tables: class exercise

- Given
  - 32-bit Virtual Addresses
  - 64 KB Pages
  - 8-bit First Level Page Table Number (directory)

- What are the components of the address:

  `0x48503423`

Multi-Level Page Tables: class exercise

- Given
  - 32-bit Virtual Addresses
  - 64 KB Pages
  - 8-bit First Level Page Table Number (directory)

- What are the components of the address:
  `0x48503423`

- **`0x48(directory), 0x50(page), 0x3423(offset)`**

Multi-Level Page Tables: class exercise

- Given
  - 32-bit Virtual Addresses
  - 4 KB Pages
  - 4 B page table entries
- If **every-level** page table fits into a single page:
  - How many levels are in the page table?
  - How many bits is the index of each level?

Multi-Level Page Tables: class exercise

- Given
  - 32-bit Virtual Addresses
  - 4 KB Pages
  - 4 B page table entries

- If **every-level** page table fits into a single page:
  - How many levels are in the page table? **2**
  - How many bits is the index of each level? **10**

Class exercise

- **Given:**
  - Each PTE is 16 B
  - The pointer to top-level of the page table is **0x1000**.
  - *: "PTE Content" shows the contents of the memory if it was read as a PTE, and only shows the address field of the PTE.
- **Q: On system with a single-level page table and 256 B pages:**
  - What is the physical address of the virtual address **0x0241**?

Ph. Mem Address	PTE Content*
0x1000	0x2000
0x1010	0x2100
0x1020	0x2200
0x1030	0x2300
0x1040	0x2400
0x1050	0x2500
0x2000	0x1000
0x2010	0x2000
0x2020	0x3000
0x2100	0x4000
0x2110	0x5000
0x2120	0x6000
0x2200	0x7000
0x2210	0x8000
0x2220	0x9000
0x2300	0xa000
0x2310	0xb000

Class exercise

- **Given:**
  - Each PTE is 16 B
  - The pointer to top-level of the page table is **0x1000**.
  - *: "PTE Content" shows the contents of the memory if it was read as a PTE, and only shows the address field of the PTE.

- **Q: On system with a single-level page table and 256 B pages:**
  - What is the physical address of the virtual address **0x0241**?

➔ physical address: **0x2241**

Ph. Mem Address	PTE Content*
0x1000	0x2000
0x1010	0x2100
0x1020	0x2200
0x1030	0x2300
0x1040	0x2400
0x1050	0x2500
0x2000	0x1000
0x2010	0x2000
0x2020	0x3000
0x2100	0x4000
0x2110	0x5000
0x2120	0x6000
0x2200	0x7000
0x2210	0x8000
0x2220	0x9000
0x2300	0xa000
0x2310	0xb000

Class exercise

- **Given:**
  - Each PTE is 16 B
  - The pointer to top-level of the page table is **0x1000**.
  - *: "PTE Content" shows the contents of the memory if it was read as a PTE, and only shows the address field of the PTE.

- **Q: On system with a two-level page table where the index of each level is 4-bits:**
  - What is the physical address of the virtual address **0x1234**?

Ph. Mem Address	PTE Content*
0x1000	0x2000
0x1010	0x2100
0x1020	0x2200
0x1030	0x2300
0x1040	0x2400
0x1050	0x2500
0x2000	0x1000
0x2010	0x2000
0x2020	0x3000
0x2100	0x4000
0x2110	0x5000
0x2120	0x6000
0x2200	0x7000
0x2210	0x8000
0x2220	0x9000
0x2300	0xa000
0x2310	0xb000

Class exercise

- **Given:**
  - Each PTE is 16 B
  - The pointer to top-level of the page table is **0x1000**.
  - *: "PTE Content" shows the contents of the memory if it was read as a PTE, and only shows the address field of the PTE.

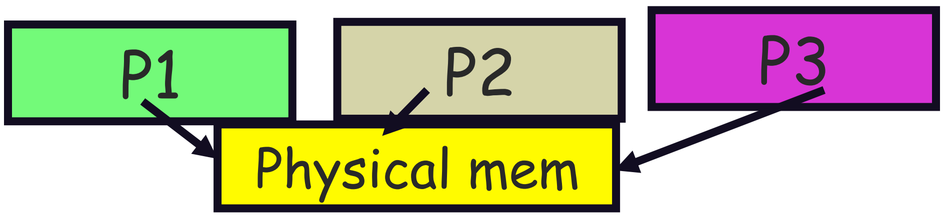
- **Q: On system with a two-level page table where the index of each level is 4-bits:**
  - What is the physical address of the virtual address **0x1234**?

➔ physical address: **0x6034**

Ph. Mem Address	PTE Content*
0x1000	0x2000
0x1010	0x2100
0x1020	0x2200
0x1030	0x2300
0x1040	0x2400
0x1050	0x2500
0x2000	0x1000
0x2010	0x2000
0x2020	0x3000
0x2100	0x4000
0x2110	0x5000
0x2120	0x6000
0x2200	0x7000
0x2210	0x8000
0x2220	0x9000
0x2300	0xa000
0x2310	0xb000

What is memory trashing?

- Thrashing: as number of page frames per process decreases, the page fault rate increases.



  - Each time one page is brought in, another page, whose contents will soon be referenced, is thrown out.
  - Processes will spend all of their time blocked, waiting for pages to be fetched from disk
  - I/O utilization at 100% but the system is not getting much useful work done
  - CPU is mostly idle

Why Trashing

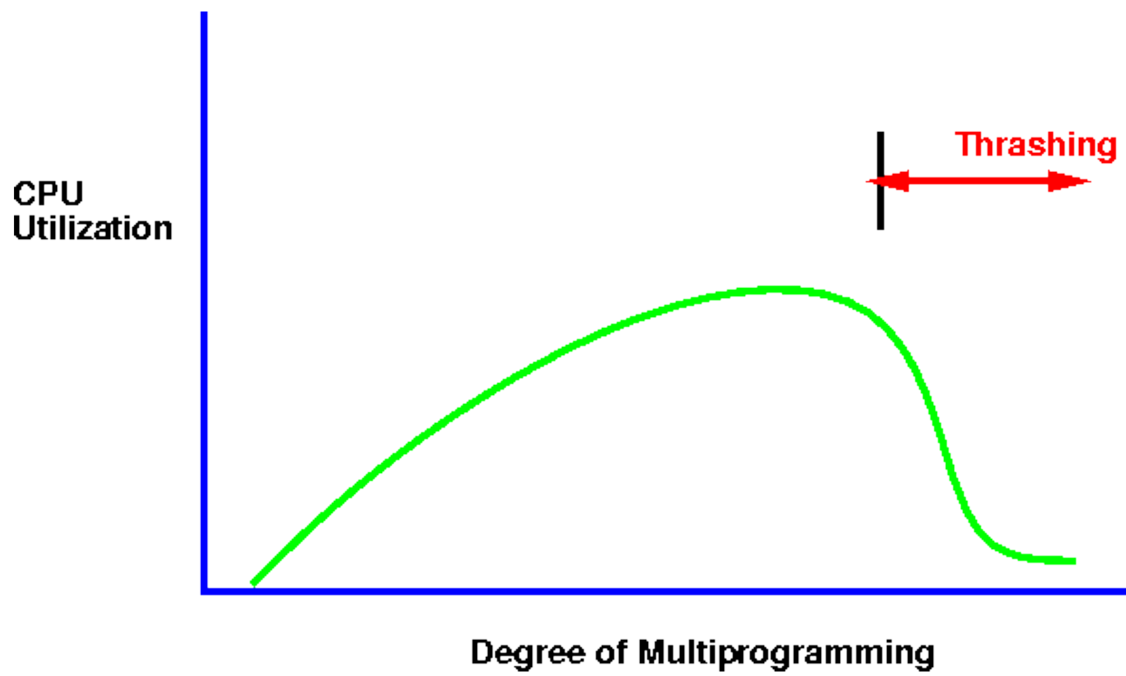
- Computation has locality
- As number of page frames allocated to a process decreases, the page frames available are not enough to contain the locality of the process.
- The processes experience heavy page faulting
  - Pages that are paged in, are used and immediately paged out.

Level of multiprogramming

- Load control has the important function of deciding how many processes will be resident in main memory
- What are the trade-offs involved?

Level of multiprogramming

- What are the trade-offs involved?
  - If too few processes are resident in memory, it can happen that all processes resident in memory are blocked so swapping is necessary and CPU is left idle
  - If too many processes are resident, then the average size of the resident set of each process will be insufficient triggering frequent page faults



Working set (1968, Denning)

- Main idea
  - figure out how much memory a process needs to keep most of its recent computation in memory with very few page faults

- How?
  - The working set model assumes temporal locality
  - Recently accessed pages are more likely to be accessed again

- Thus, as the number of page frames increases above some threshold, the page fault rate will drop dramatically

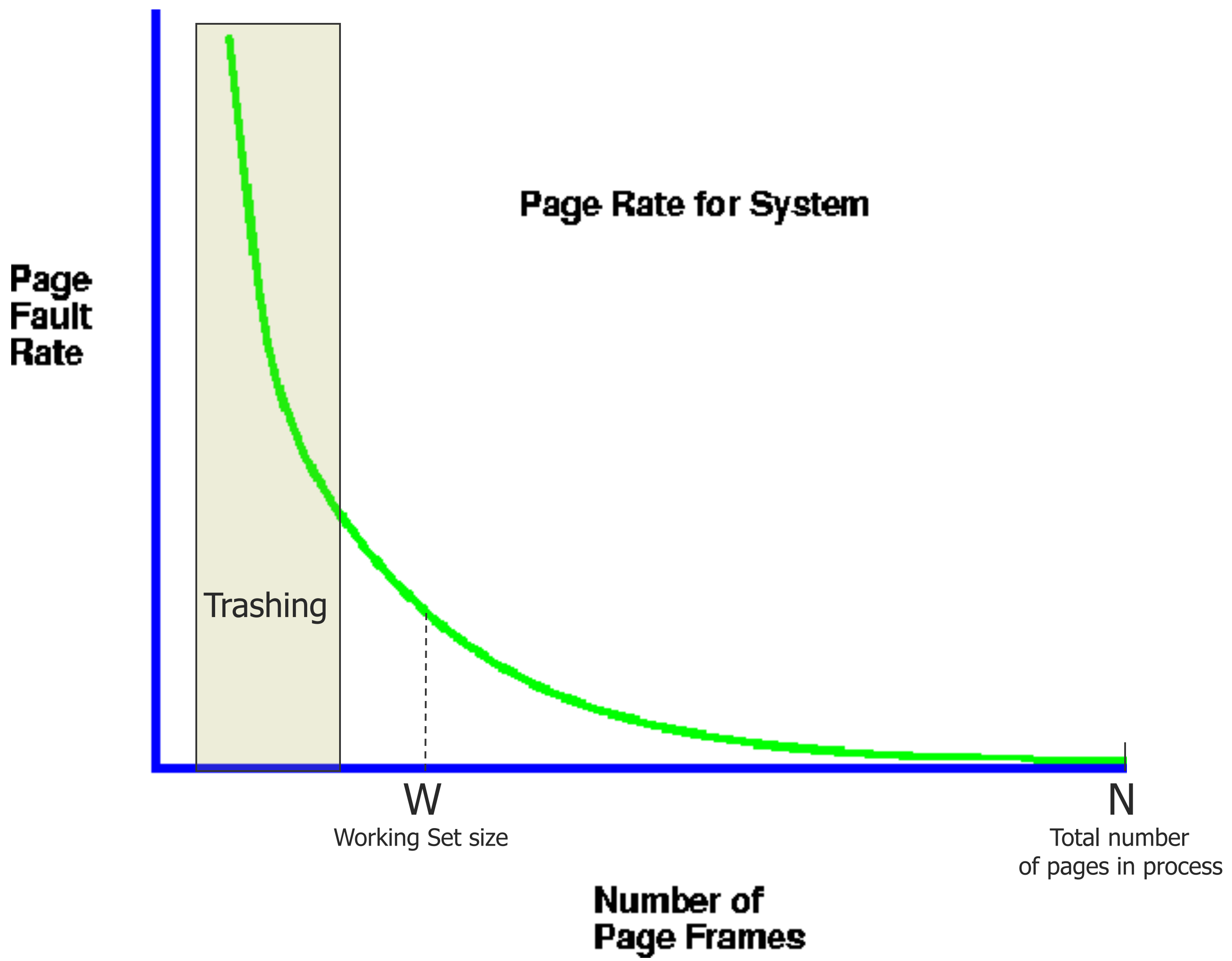
Working set (1968, Denning)

- What we want to know: collection of pages process must have in order to avoid thrashing
  - This requires knowing the future. And our trick is?

- Intuition of Working Set:
  - Pages referenced by process during last interval of execution are considered to comprise its working set
  - Δ: the working set window

- Usages of working set?
  - Cache partitioning: give each application enough space for WS
  - Page replacement: preferably discard non-WS pages
  - Scheduling: a process is not executed unless its WS is in memory

Page Fault Rate vs. Allocated Frames



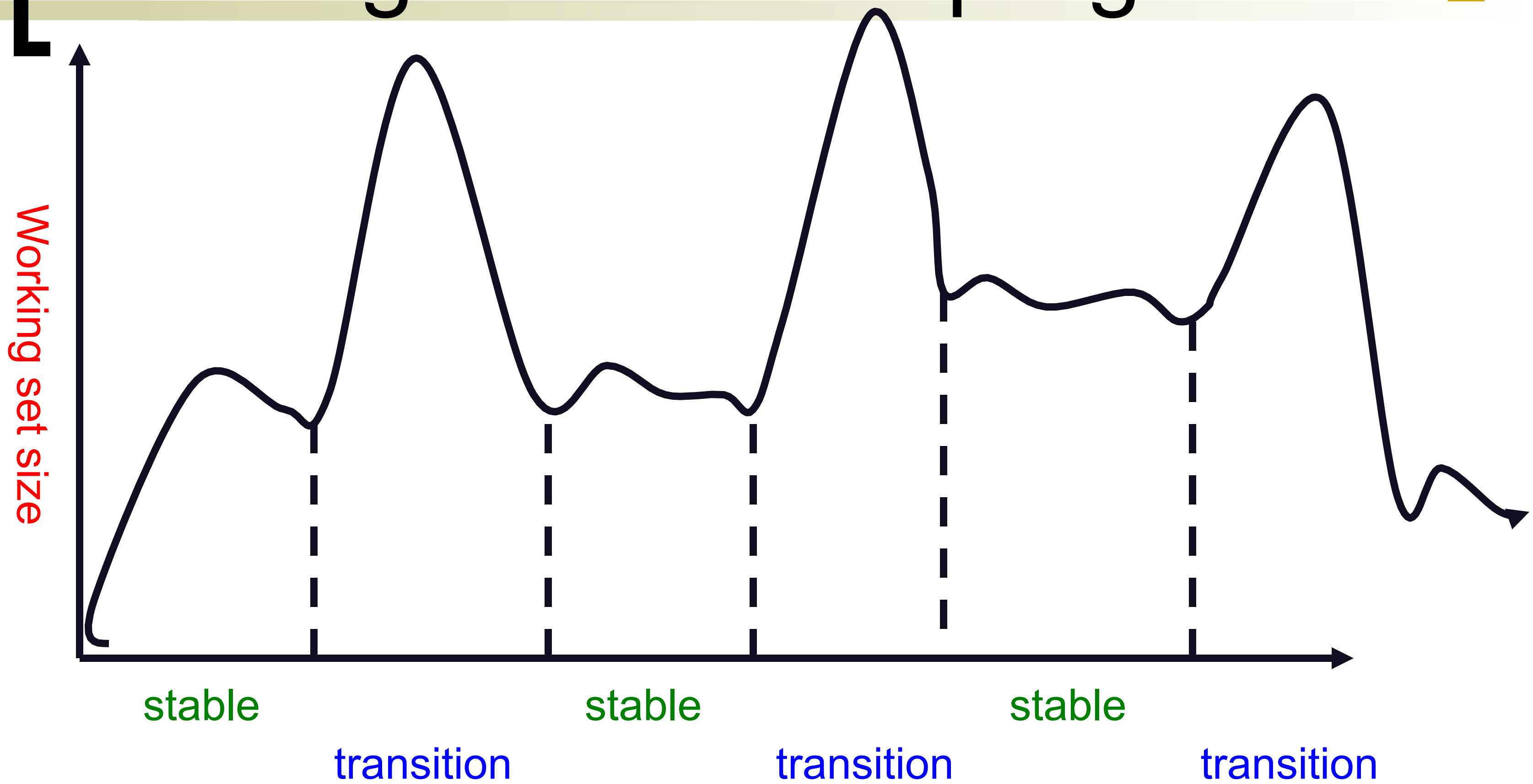
Working Set (1968, Denning)

- Strategy for sizing the resident set of a process based on Working set
  - Keep track of working set of each process
  - Periodically remove from the resident set the pages that don’t belong to working set anymore
  - A process is scheduled for execution only if its working set is in main memory

Working Set Size

- Choosing
  - Δ too small
    - Will not encompass entire locality
  - Δ too large
    - Will encompass several localities
  - $\Delta = \infty$
    - Will encompass entire program

Working sets of real programs



- Typical programs have phases

Page Size Considerations

- Page size is a crucial parameter for performance of virtual memory
  - Small pages
    - Large page tables
    - Minimizes internal fragmentation
    - Good for locality of reference
  - Large pages
    - Small page tables
    - Significant amounts of a page may not be referenced
    - locality is not well exploited anymore and page fault rate increases
  - Real systems (can be reconfigured)
    - Windows: default 8KB
    - Linux: default 4 KB