



# CSCI [4 | 6] 730 Operating Systems

## Virtual Memory



Maria Hybinette, UGA

# Chapter 10: Virtual Memory Questions?

- What is virtual memory and when is it useful?
- What is demand paging?
- When should pages in memory be replaced?
- When should pages be put on disk?
- What is the working set model?

Maria Hybinette, UGA

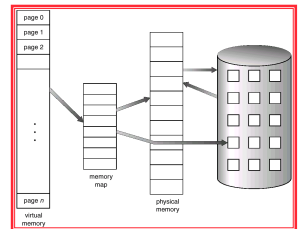
## Operating System's Goals

- Support processes when there is not enough physical memory
  - » Single process with very large address space
  - » Multiple processes with combined address spaces
- User code should be independent of amount of physical memory
  - » Correctness, if not performance

Maria Hybinette, UGA

## The Illusion: "Virtual" Memory

- OS provides an *illusion* of more memory than is physically available:
  - » Large logical space but really
  - » small physical memory
- Why does this work?
  - » Only part of the program needs to be in memory (at a particular time) for execution
  - » Relies on key properties of user processes (workload) and machine architecture (hardware)



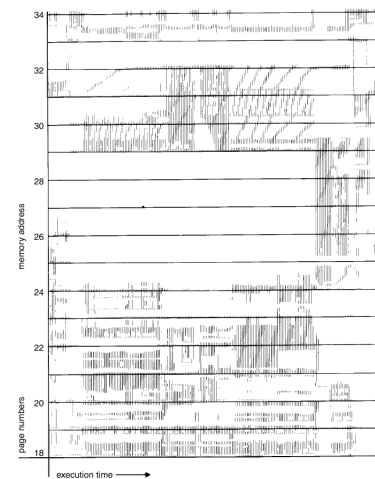
Maria Hybinette, UGA

## The Million Dollar Question?

- How do the OS decide what is in "main" memory and what is on disk?
- How can we decide?
  - » Memory Access Patterns?



Maria Hybinette, UGA



Maria Hybinette, UGA

## Observations: Memory Access Patterns

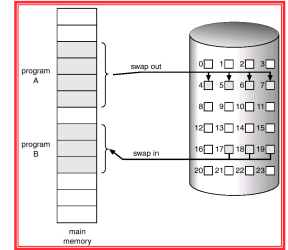
- Sequential memory accesses of a process are predictable and tend to have **locality of reference**:
  - » **Spatial**: reference memory addresses *near* previously referenced addresses (in memory)
  - » **Temporal**: reference memory addresses that have been referenced in the past
- Processes spend majority of time in small portion of code
  - » **Estimate**: 90% of time in 10% of code
- **Implication**:
  - » Process only uses **small** amount of address space at any moment
  - » Only small amount of address space must be resident in physical memory

Maria Hyönnelie, UGA

7

## Approach: Demand Paging

- Bring in pages into memory **only** when needed
  - » Less memory
  - » Less I/O
  - » Faster response time?
- Process viewed as a sequence of pages rather than contiguous address space
  - » **Pager** (whole process)
  - » **Swapper** (portion of a process)



Maria Hyönnelie, UGA

8

## Virtual Memory Approach: Intuition

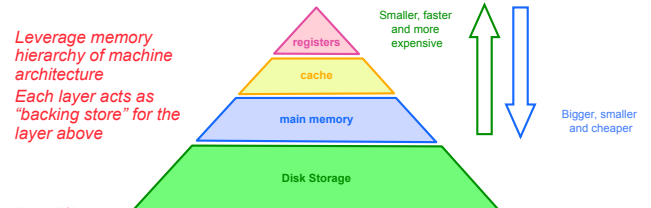
- **Idea**: OS keeps **unreferenced** pages on disk
  - » Slower, cheaper backing store than memory
- Process can run when not all pages are loaded into main memory
- OS and hardware cooperate to provide illusion of large disk as fast as main memory
  - » Same behavior as if all of address space in main memory
  - » Hopefully have similar performance
- **Requirements**:
  - » OS must have mechanism to identify location of each page in address space in **memory** or on **disk**
  - » OS must have policy for determining which pages live in memory and which on disk

Maria Hyönnelie, UGA

9

## Virtual Address Space Mechanisms

- Each page in virtual address space maps to one of three locations:
  - » **Physical main memory**: Small, fast, expensive
  - » **Disk (backing store)**: Large, slow, cheap
  - » **Nothing (error)**: Free



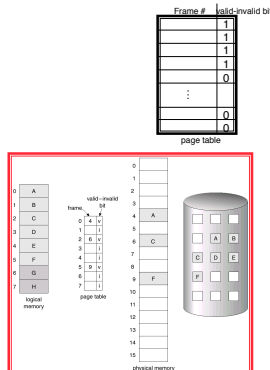
Maria Hyönnelie, UGA

10

## Virtual Address Space Mechanisms

Extend page tables with an extra bit to indicate whether it is in memory or on disk (a resident bit):

- **valid (or invalid)**
- **Page in memory**: valid bit set in page table entry (PTE)
- **Page out to disk**: valid bit cleared (**invalid**)
  - PTE points to **block** on disk
  - Causes trap into OS when page is referenced
  - **Trap: page fault**



Maria Hyönnelie, UGA

11

## Virtual Memory Mechanisms (cont)

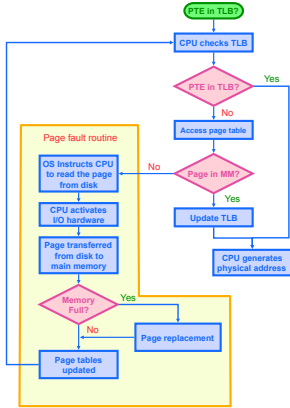
The TLB factor: Hardware and OS cooperate to translate addresses

- **First**, hardware checks **TLB** for virtual address
  - » **TLB hit**: Address translation is done; page in physical memory
  - » **TLB miss**:
    - Hardware or OS **walk** page tables
    - If PTE designates page is **valid**, then page in physical memory
- **Main Memory Miss**: Not in main memory: **Page fault** (i.e., **invalid**)
  - » Trap into OS (not handled by hardware)
  - » OS selects victim page in memory to replace
    - Write victim page out to disk if **modified** (add **dirty** bit to PTE)
  - » OS reads referenced page from disk into memory
  - » Page table is updated, **valid** bit is set
  - » Process continues execution

Maria Hyönnelie, UGA

12

# Flow of "Paging" Operations



# Virtual Memory Policies

- OS needs to decide on **policies** on page faults concerning:
  - » Page selection (When to bring in)
    - When should a page (or pages) on disk be brought into memory?
    - Two cases
      - When process starts, code pages begin on disk
      - As process runs, code and data pages may be moved to disk
  - » Page replacement (What to replace)
    - Which resident page (or pages) in memory should be thrown out to disk?
- Goal: Minimize number of page faults
  - » Page faults require milliseconds to handle (reading from disk)
  - » Implication: Plenty of time for OS to make good decision

Maria Hyönnelie, UGA

14

# The When: Page Selection

- When should a page be brought from disk into memory?
- Request paging: User specifies which pages are needed for process
  - » Problems:
    - Manage memory by hand
    - Users do not always know future references
    - Users are not impartial
- Demand paging: Load page **only** when page fault occurs
  - » Intuition: Wait until page must absolutely be in memory
  - » When process starts: No pages are loaded in memory
  - » Advantage: Less work for user
  - » Disadvantage: Pay cost of page fault for every newly accessed page



Maria Hyönnelie, UGA

15

# Page Selection Continued

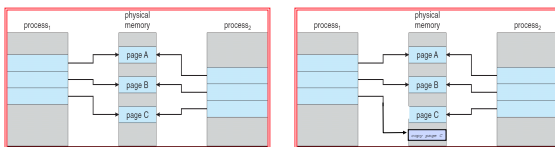
- Prepaging (anticipatory, prefetching): OS loads page into memory **before** page is referenced
  - » OS predicts future accesses (**oracle**) and brings pages into memory ahead of time
    - How?
    - Works well for some access patterns (e.g., sequential)
  - » Advantages: May avoid page faults
  - » Problems? :
- Hints: Combine **demand** or **prepaging** with **user-supplied hints** about page references
  - » User specifies: may need page in future, don't need this page anymore, or sequential access pattern, ...
  - » Example: `madvise()` in Unix (1994 4.4 BSD UNIX)

Maria Hyönnelie, UGA

16

# Virtual Page Optimizations

- Copy-on-Write: on process creation allow parent and child to share the same page in memory until one modifies the page.

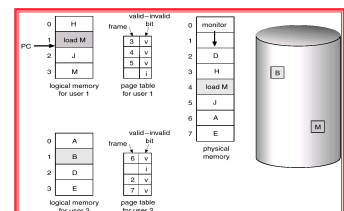


Maria Hyönnelie, UGA

17

# What happens if there is no free frame?

- Page replacement
  - » find some page in memory, but not really in use, swap it out
- Observation: Same page may be brought into memory several times



Maria Hyönnelie, UGA

18

## Page Replacement

- Which page in main memory should selected as victim?
  - » Write out victim page to disk if modified (dirty bit set)
  - » If victim page is not modified (clean), just discard
- OPT: Replace page *not used for longest time* in future
  - » **Advantage:** Guaranteed to minimize number of page faults
  - » **Disadvantage:** Requires that OS predict the future
    - Not practical, but a good for comparison.
- Random: Replace any page at random
  - » **Advantage:** Easy to implement
  - » **Surprise?:** Works okay when memory is not severely over-committed

Maria Hyönnelie, UGA

15

## Page Replacement Continued

- FIFO: Replace page that has been in memory the longest
  - » **Intuition:** First referenced long time ago, done with it now
  - » **Advantages:**
    - Fair: All pages receive equal residency
    - Easy to implement (circular buffer)
  - » **Disadvantage:** Some pages may always be needed
- LRU: Replace page *not used* for longest time in past
  - » **Intuition:** Use past to predict the future
  - » **Advantages:**
    - With locality, LRU approximates OPT
  - » **Disadvantages:**
    - Harder to implement, must track which pages have been accessed
    - Does not handle all workloads well

Maria Hyönnelie, UGA

20

## How to Evaluate Page Replacement Algorithms?

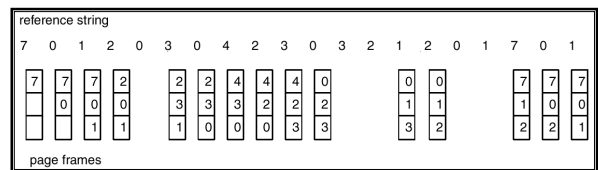
- **Want:** lowest page-fault rate.
- **Idea:** Keep track of memory references – test with particular string of memory references and count page faults.
- **Algorithm:** Convert address to page
  - » **Example:** Assume 100 bytes per page and
    - **Step 1:** Assume the address sequence:
      - 0100, 0210, 0250, 0300, 0350, 0380, 0400, 0160, 0250, 0505, 0100, 0110, 0230, 0350, 0450, 0450, 0500, 0500
    - **Step 2:** Convert address to a page reference string:
      - 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.
    - **Step 3:** Count page faults.



Maria Hyönnelie, UGA

21

## Example: Counting Faults of FIFO Page Replacement Algorithm



- 3 Frames are available
- FIFO: Replace page that has been in memory the longest
- Count page faults ?

Maria Hyönnelie, UGA

22

## Page Replacement Example

Page reference string: A B C A B D A D B C B  
Three pages of physical memory

	OPT	FIFO	LRU
ABC			
A			
B			
D			
A			
D			
B			
C			
B			

Maria Hyönnelie, UGA

23

## Page Replacement: Adding More Memory

- Add more physical memory, what happens to performance?
  - » Ideally the numbers of page faults should decrease as number of available frames increases
  - » 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.
  - » If 1 page frame : Number of page faults?
    - 12 page faults, one fault for every page
  - » If 12 frames : Number of page faults?
    - 5 page faults

Maria Hyönnelie, UGA

24

## First-In-First-Out (FIFO) Algorithm: Add Memory

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)



- 4 frames



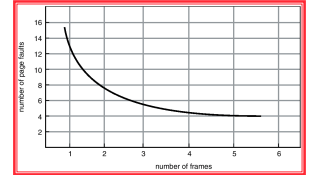
- FIFO Replacement – Belady's Anomaly
  - Violates the Principle: More frames  $\Rightarrow$  less page faults

Maria Hyönnelie, UGA

26

## Page Replacement: Add memory

- Add more physical memory, what happens to performance?
  - Ideally the numbers of page faults should decrease as number of available frames increases
  - 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.
  - If 1 page frame : Number of page faults?
    - 12 page faults, one fault for every page (11 faults)
  - If 12 frames : Number of page faults?
    - 5 page faults



Maria Hyönnelie, UGA

27

## Page Replacement Comparison

- Add more physical memory, what happens to performance?
  - LRU, OPT: Add more memory, **guaranteed** to have fewer (or same number of) page faults
    - Smaller memory sizes are guaranteed to contain a subset of larger memory sizes
  - FIFO: Add more memory, **usually** have fewer page faults
    - Belady's anomaly: May actually have **more** page faults!

Maria Hyönnelie, UGA

27

## Implementing LRU

- Software Perfect LRU (Stack)
  - OS maintains ordered list of physical pages by reference time
  - When page is referenced: Move page to front of list (top)
  - When need victim: Pick page at back of list (bottom)
  - Trade-off: Slow on memory reference, fast on replacement
- Hardware Perfect LRU
  - Associate register with each page
  - When page is referenced: Store system clock in register
  - When need victim: Scan through registers to find **oldest** clock
  - Trade-off: Fast on memory reference, slow on replacement (especially as size of memory grows)
- In practice, do not implement Perfect LRU
  - LRU is an approximation anyway, so approximate more
  - Goal: Find an old page, but not necessarily the very oldest

Maria Hyönnelie, UGA

28

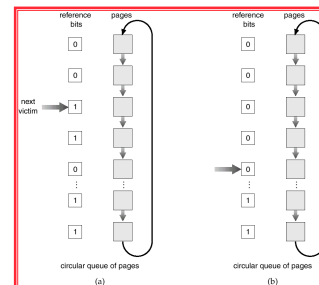
## Clock or Second Chance Algorithm

- Hardware
  - Keep **use** (or **reference**) bit for each page frame initialized to 0.
  - When page is referenced: set **use** bit (1)
- Operating System
  - Page replacement: Look for page with **use** bit cleared (0) (has not been referenced for awhile)
  - Implementation:
    - Keep pointer to last examined page frame
    - Traverse pages in circular buffer
    - Clear **use** bits while searching for replacement
    - Stop when find page with already cleared **use** bit, replace this page

Maria Hyönnelie, UGA

28

## Clock Algorithm Example



- Worst Case:
  - All bits are set  $\rightarrow$  FIFO (slow)

Maria Hyönnelie, UGA

29

## Clock Extensions

- **Replace multiple pages at once**
  - » Intuition: Expensive to run replacement algorithm and to write single block to disk
  - » Find multiple victims each time
- **Two-handed clock**
  - » Intuition:
    - If takes long time for clock hand to sweep through pages, then all use bits might be set
    - Traditional clock cannot differentiate between usage of different pages
  - » Allow smaller time between clearing use bit and testing
    - First hand: Clears use bit
    - Second hand: Looks for victim page with use bit still cleared

Maria Hyömette, UGA

31

## More Clock Extensions

- **Add software byte**
  - » Intuition: Keep track of history when last used
- **Implementation: Reference bit**
  - » With each page associate a bit, initially = 0
  - » When page is referenced bit set to 1.
  - » Keep a *history* of reference bit in an 8 bit byte:
    - Shift reference bit for each page to high order bit, and other bits right one bit.
    - 11000100 (more recently used than below)
    - 01110111

Maria Hyömette, UGA

32

## More Clock Extensions (R/W)

- Use **dirty** bit to give preference to dirty pages
  - » Intuition: More expensive to **replace** dirty pages
    - Dirty pages must be written to disk, clean pages do not
  - » Replace pages that have use bit and **dirty** bit cleared

0, 0	Not recently used, not modified	Best to replace
0, 1	Not recently used, but modified	Needs to be written out
1, 0	Recently used, not modified	Probably used again soon
1, 1	Recently used and modified	Probably used again soon and need to be written out

Maria Hyömette, UGA

33

## Problems with LRU-based Replacement

- **Locality of reference:**
  - » Same pages referred frequently (warm pages)
  - » Example: 2, 1, 3, 2, 4, 2, 4, 1, 5, 6, 2, ...
- **LRU takes advantage of this!**
- **Leading question:**
  - » Is a page that has been accessed once in the past as likely to be accessed in the future as one that has been accessed **N** times?

Maria Hyömette, UGA

34

## Problems with LRU-based Replacement

- **Example:** 2, 1, 3, 2, 4, 2, 4, 1, 5, 6, 2, ...
- **Problem:**
  - » Dislodges warm pages if a long sequence of one time page references occur.
    - In the above ex, page 2 may get dislodged by the access pattern ..., 4, 1, 5, 6,
  - » LRU does not consider *frequency* of accesses
- **Solution:** Track frequency of accesses to page
  - » Pure LFU (Least-frequently-used) replacement
- **Problem:** but LFU can never forget pages from the far past...

Maria Hyömette, UGA

35

## Questions

- How to allocate memory **across competing processes?**
- What is thrashing? What is a working set?
- How to ensure working set of all processes fit?

Maria Hyömette, UGA

36

## Allocating Memory across Processes

- **Problem:**
  - » 2 processes and 25 free frames how are they divided up?
- **Three General Approaches:**
  - » Global Replacement
  - » Per-Process Replacement
  - » Per-User Replacement

Maria Hyönnelie, UGA

37

## Global Replacement

- **Global replacement**
  - » Pages from all processes lumped into single replacement pool
  - » Each process competes with other processes for frames
  - » **Advantages:**
    - Flexibility of allocation
    - Minimize total number of page faults
  - » **Disadvantages:**
    - One memory-intensive process can hog memory, hurt all processes (not fair)
    - Paging behavior of one process depends on the behavior of other processes

Maria Hyönnelie, UGA

38

## Per-process replacement

- **Per-process free pool of pages:**
  - » **Equal, Fixed Allocation:** Fixed number of pages per process
    - 100 frames and 5 processes, give each 20 pages.
    - Fixed fraction of physical memory
  - » **Proportional Allocation:**
    - Proportional to size of address space of a process.
    - Adjust size allocated if a process have higher priority
- **Page fault in one process only replaces frame of that process**
- **Advantage:** Relieves interference from other processes
- **Disadvantage:** Potentially inefficient allocation of resources

Maria Hyönnelie, UGA

39

## Per-User Replacement

- **Advantages:** Users running more processes cannot hog memory
- **Disadvantage:** Inefficient allocation

Maria Hyönnelie, UGA

40

## Over Committing Memory

- **When does the Virtual Memory illusion break?**
- **Example:**
  - » Set of processes *frequently* referencing 33 important pages - more than memory available
    - Physical memory can fit 32 pages
- **What happens?**
  - » **System Repeat Cycle:**
    - Reference page not in memory
    - Replace a page in memory with newly referenced page
    - Replace another page right away again, since all its pages are in active use...

Maria Hyönnelie, UGA

41

## Thrashing

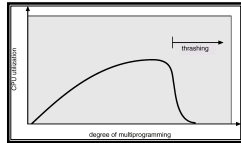
- **Thrashing:**
  - » **Definition:** Spends more time paging than **execution**, i.e. system reading and writing pages instead of executing useful instructions
  - » **Symptom:** Average memory access time equals to disk access time
    - Breaks the virtual memory illusion because memory appears as slow as disk rather than disk appearing fast as memory (system is reading/writing instead of executing)
    - Memory appears as slow as disk, instead of disk appearing as fast as memory
  - » **Processes execute less – system admits more processes**  
-> thrashing gets worse

Maria Hyönnelie, UGA

42

## System does not know it is thrashing

- If a process does not have “enough” pages, the page-fault rate is very high.
  - » low CPU utilization.
  - » operating system thinks that it needs to **increase** the degree of multiprogramming.
  - » another process added to the system
- Why the CPU utilization decreases:
  - » Suppose a process need more frames, starts faulting, removing frames from others, in turn making the other processes fault
  - » Processes queue up for the paging device, CPU decreases
  - » OS add processes that immediately need new frames further taking away pages from running processes



Maria Hyönnelie, UGA

46

## Thrashing: Solutions

- **Limit thrashing** by using a local replacement
  - » Process does not steal frames from other and cause latter to thrash
  - » Average service time for a page fault will still increase...
- **Admission Control:**
  - » Determine of much memory each process needs
  - » Long-term scheduling policy:
    - Run only processes whose memory requirement can be satisfied
- **What if memory requirement of one process is too high?**
  - » **Observation:** a process moves through different “localities” through out its lifetime
    - Locality: Set of pages that are actively used together
  - » **Solution:** Amortize page allocated so that a process get enough page for its current locality

Maria Hyönnelie, UGA

47

## Motivation for Solution

- Thrashing cannot be fixed with better replacement policies
  - » Page replacement policies do not indicate that a page must be kept in memory
  - » Only show which pages are **better** than others to replace
- Student’s analogy to thrashing: Too many courses
  - » Solution: Drop a course
- OS solution: Admission control
  - » Determine how much memory each process needs
  - » Long-term scheduling policy
    - Run only those processes whose memory requirements can be satisfied
  - » What if memory needs of one process are too large?

Maria Hyönnelie, UGA

48

## Working Set

- **Informal definition**
  - » Collection of pages the process is referencing frequently
  - » Collection of pages that must be resident to avoid thrashing
- **Formal definition**
  - » Assume locality; use **recent past** to predict future
  - » Pages referenced by process in last T seconds of execution
  - » Working set changes slowly over time
- **Example:**



Maria Hyönnelie, UGA

49

## - Balance Set -

- **Motivation:** Process should not be scheduled unless working set is resident in main memory
- Divide runnable processes into two groups:
  - » Active: Working set is loaded
  - » Inactive: Working set is swapped to disk
- **Balance set:** Sum of working sets of all active processes
- **Interaction with scheduler**
  - » If balance set exceeds size of memory, move some process to inactive set
    - Which process???
  - » If balance set is less than size of memory, move some process to active set
    - Which process?
  - » Any other decisions?

Maria Hyönnelie, UGA

47

## Working Set Implementation

- Leverage **use** bits (as in the clock algorithm)
- OS maintains **idle time** for each page
  - » Amount of CPU received by process since last access to page
  - » Periodically scan all resident pages of a process
    - If use bit is set, clear page’s idle time
    - If use bit is clear, add process CPU time (since last scan) to idle time
  - » If idle time < T, page is in working set

Maria Hyönnelie, UGA

48



## Thought Questions

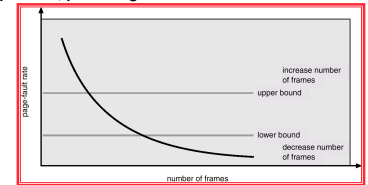
- How should value of T be configured?
  - » What if T is too large?
- How should working set be defined when pages are shared?
  - » Put jobs sharing pages in same balance set
- What processes should compose balance set?
- How much memory needed for a “balanced system”?
  - » Balanced system: Each resource (e.g., CPU, memory, disk) becomes bottleneck at nearly same time
  - » How much memory is needed to keep the CPU busy?
  - » With working set approach, CPU may be idle even with runnable processes

Maria Hyönnelä, UGA

48

## Page-Fault Frequency Scheme

- **Observation:** Thrashing has a high page-fault rate
- **Idea:** Control page fault-rate by controlling # frames that are allocated to a process
  - » Too high page fault rate : process need more frames
  - » Too low : process have to many frames
- **Approach:** Establish “acceptable” page-fault rate (upper and lower bound)
  - » If actual rate falls below lower limit, process loses frame.
  - » If actual rate exceeds upper limit, process gains frame.



Maria Hyönnelä, UGA

50

## Current Trends

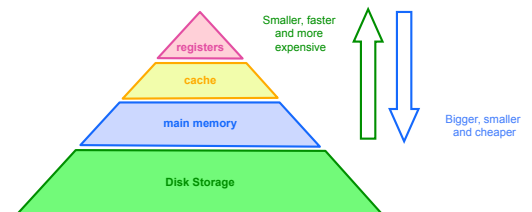
- **VM code is not as critical**
  - » Reason #1: Personal vs. time-shared machine
    - Why does this matter?
  - » Reason #2: Memory is more affordable, more memory
- **Less hardware support for replacement policies**
  - » Software emulation of use and dirty bits
- **Larger page sizes**
  - » Better TLB coverage
  - » Smaller page tables
  - » Disadvantage: More internal fragmentation
    - Multiple page sizes

Maria Hyönnelä, UGA

51

## Memory Hierarchy

Leverage memory hierarchy of machine architecture  
Each layer acts as “backing store” for the layer above



Maria Hyönnelä, UGA

52

## Example: Average Access Time calculation

- **Average access time calculation:**
  - » **H:** Percentage of references that hit page in physical memory
  - » **C<sub>AccessMemory</sub>:** Cost of referencing page in memory (e.g., 100 ns)
  - » **C<sub>PageFault</sub>:** Cost of page fault (e.g., 20 ms or 20,000,000ns)
  - »  $H * C_{AccessMemory} + (1-H) * C_{PageFault}$
- **Example: 1 out of every 33 references misses, H = 97%**
  - »  $0.97 * (100 \text{ ns}) + (0.03) * (20000000 \text{ ns}) = 750000 \text{ ns} = 750 \text{ us}$
  - » More than 1000 times slower than physical memory access
- **Need very high hit rate for acceptable performance**

Maria Hyönnelä, UGA

53