Announcements

- Reading 8 (8.1-8.2, 8.5-8.6)
- Project #3 was handed out in section
  - proc2.c is now available
  - will need to produce a short paper writeup for this assignment

Priority Algorithms

- Fixed Queues
  - processes are statically assigned to a queue
  - sample queues: system, foreground, background
- Multilevel Feedback
  - processes are dynamically assigned to queues
  - penalize jobs that have been running longer
  - preemptive, with dynamic priority
  - have ***N*** ready queues (RQ0-RQ***N***),
    - start process in RQ0
    - if quantum expires, moved to i + 1 queue

Feedback scheduling (cont.)

- problem: turnaround time for longer processes
  - can increase greatly, even starve them, if new short jobs regularly enter system
  - solution1: vary preemption times according to queue
    - processes in lower priority queues have longer time slices
  - solution2: promote a process to higher priority queue
    - after it spends a certain amount of time waiting for service in its current queue, it moves up

UNIX System V

- Multilevel feedback, with
  - RR within each priority queue
  - 10ms second preemption
  - priority based on process type and execution history, lower value is higher priority
- priority recomputed once per second, and scheduler selects new process to run
- For process j, P(i) = Base + CPU(i-1)/2 + nice
  - P(*i*) is priority of process *j* at interval *i*
  - Base is base priority of process *j*
  - CPU(*i*) = U(*i*)/2 + CPU(*i*-1)/2
    - U(*i*) is CPU use of process *j* in interval *i*
    - exponentially weighted average CPU use of process *j* through interval *i*
  - nice is user-controllable adjustment factor

UNIX (cont.)

- Base priority divides all processes into (non-overlapping) fixed bands of decreasing priority levels
  - swapper, block I/O device control, file manipulation, character I/O device control, user processes
- bands optimize access to block devices (disk), allow OS to respond quickly to system calls
- penalizes CPU-bound processes w.r.t. I/O bound
- targets general-purpose time sharing environment

Windows NT

- Target:
  - single user, in highly interactive environment
  - a server
- preemptive scheduler with multiple priority levels
- flexible system of priorities, RR within each, plus dynamic variation on basis of current thread activity for ***some*** levels
- 2 priority bands, real-time and variable, each with 16 levels
  - real-time ones have higher priority, since require immediate attention(e.g. communication, real-time task)

Windows NT (cont.)

- In real-time class, all threads have fixed priority that never changes
- In variable class, priority begins at an initial value, and can change, up or down
  - FIFO queue at each level, but thread can switch queues
- Dynamic priority for a thread can be from 2 to 15
  - if thread interrupted because time slice is up, priority lowered
  - if interrupted to wait on I/O event, priority raised
  - favors I/O-bound over CPU-bound threads
  - for I/O bound threads, priority raised more for interactive waits (e.g. keyboard, display) than for other I/O (e.g. disk)

Managing Memory

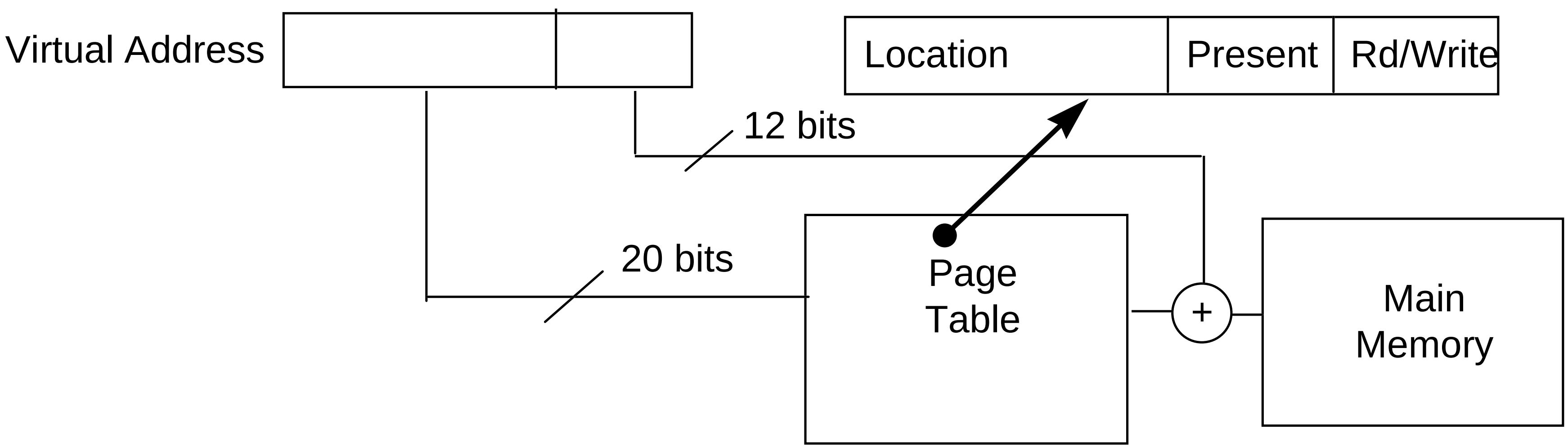
- Main memory is big, but what if we run out
  - use virtual memory
  - keep part of memory on disk
    - bigger than main memory
    - slower than main memory
- Want to have several program in memory at once
  - keeps processor busy while one process waits for I/O
  - need to protect processes from each other
  - have several tasks running at once
    - compiler, editor, debugger
    - word processing, spreadsheet, drawing program
- Use *virtual addresses*
  - look like normal addresses
  - hardware translates them to *physical addresses*

Advantages of Virtual Addressing

- Can assign non-contiguous regions of physical memory to programs
- A program can only gain access to its mapped pages
- Can have more virtual pages than the size of physical memory
  - pages that are not in memory can be stored on disk
- Every program can start at (virtual) address 0

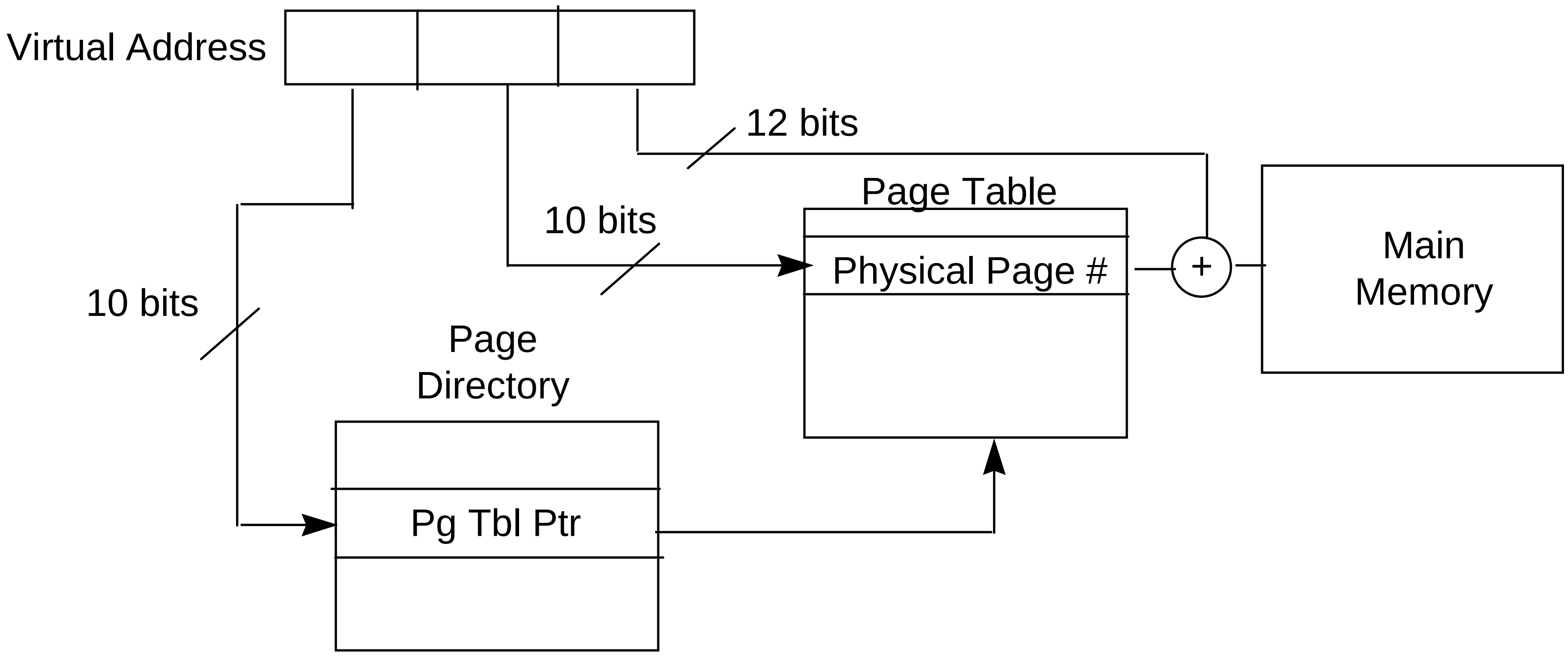
Paging

- Divide physical memory into fixed sized chunks called *pages*
  - typical pages are 512 bytes to 64k bytes
  - When a process is to be executed, load the pages that *are actually used* into memory
- Have a table to map virtual pages to physical pages
- Consider a 32 bit addresses
  - 4096 byte pages (12 bits for the page)
  - 20 bits for the page number



Problems with Page Tables

- One page table can get very big
  - 2^{20} entries (for most programs, most items are empty)
- solution1: use a hierarchy of page tables



Inverted Page Tables

- Solution to the page table size problem
- One entry per page frame of physical memory

  <process-id, page-number>

  – each entry lists process associated with the page and the page number
  – when a memory reference:
    - **<process-id,page-number,offset>**occurs, the inverted page table is searched (usually with the help of a hashing mechanism)
    - if a match is found in entry ***i*** in the inverted page table, the physical address **<i,offset>** is generated
  – The inverted page table does not store information about pages that are not in memory
    - page tables are used to maintain this information
    - page table need only be consulted when a page is brought in from disk