ELEC 377 – Operating Systems

Week 6 – Class 2

Today

- Memory Management <<<<<<<<
  - ◊ Physical vs Logical Management
  - ◊ Paging Structure
  - ◊ Shared Pages
- Virtual Memory
  - ◊ Concept
  - ◊ Demand Paging

Logical vs Physical Address Space

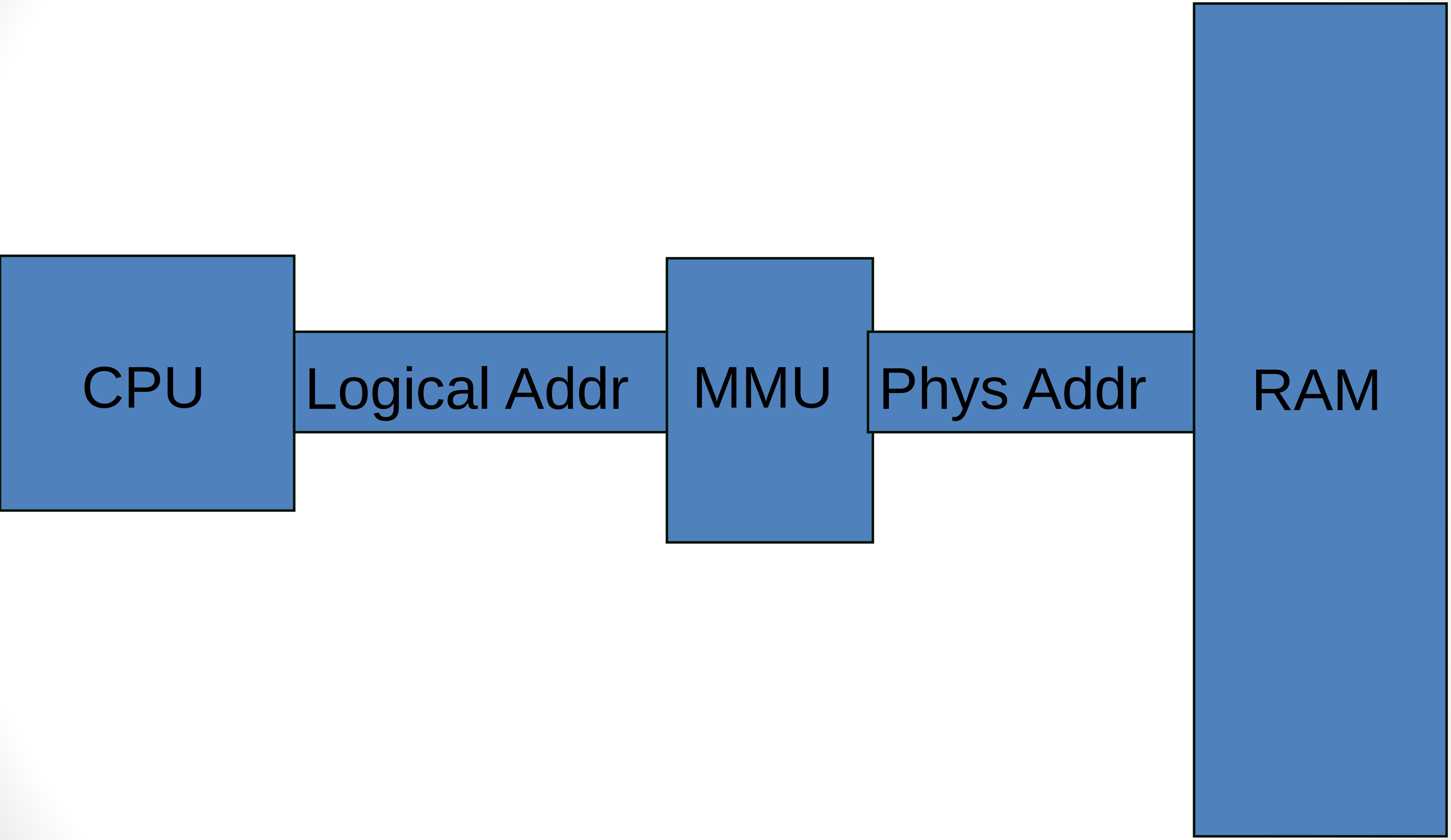
Central Concept to Memory Management

- Logical Address
  - ◊ address generated by CPU
  - ◊ also known as virtual address
- Physical Address
  - ◊ location in physical memory
- Logical and Physical address are the same in compile and load time address binding. They differ in execution time binding
- User program only deals with logical addresses. It never sees the physical address

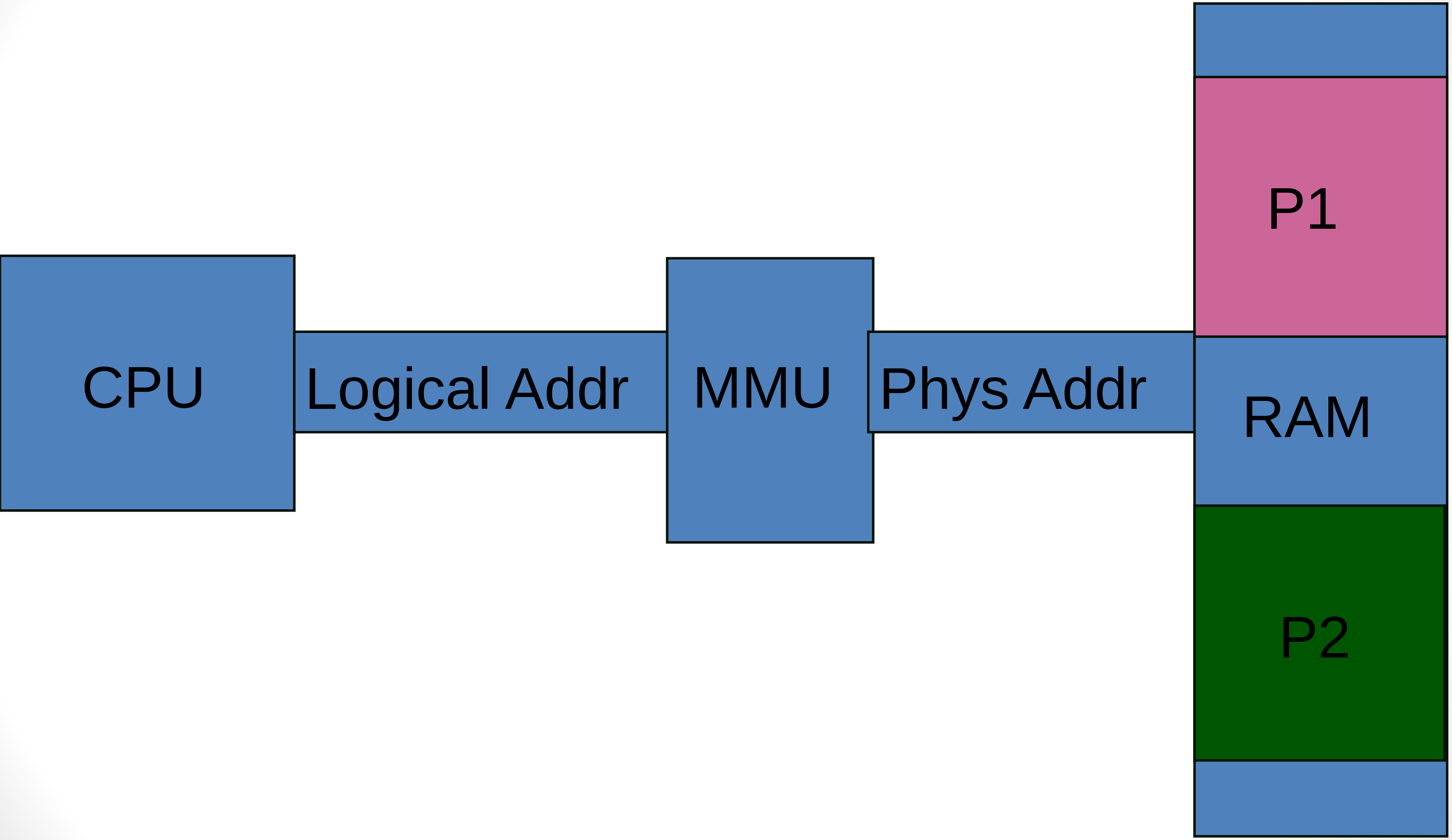
Memory Management Unit (MMU)

- Hardware that maps virtual to physical address
  - ◊ many different approaches
- One simple approach is to have a single register that is added to every virtual address
  - ◊ Similar to the original memory protection scheme talked about in Week 1.
  - ◊ Limit register is now size of memory space
  - ◊ base register is called the ***relocation*** register
  - ◊ Used by MSDOS on 386, PDP-11
- Logical addresses (0…*max*)
- Physical address (R…R+max)
  - ◊ R is the value of the relocation register

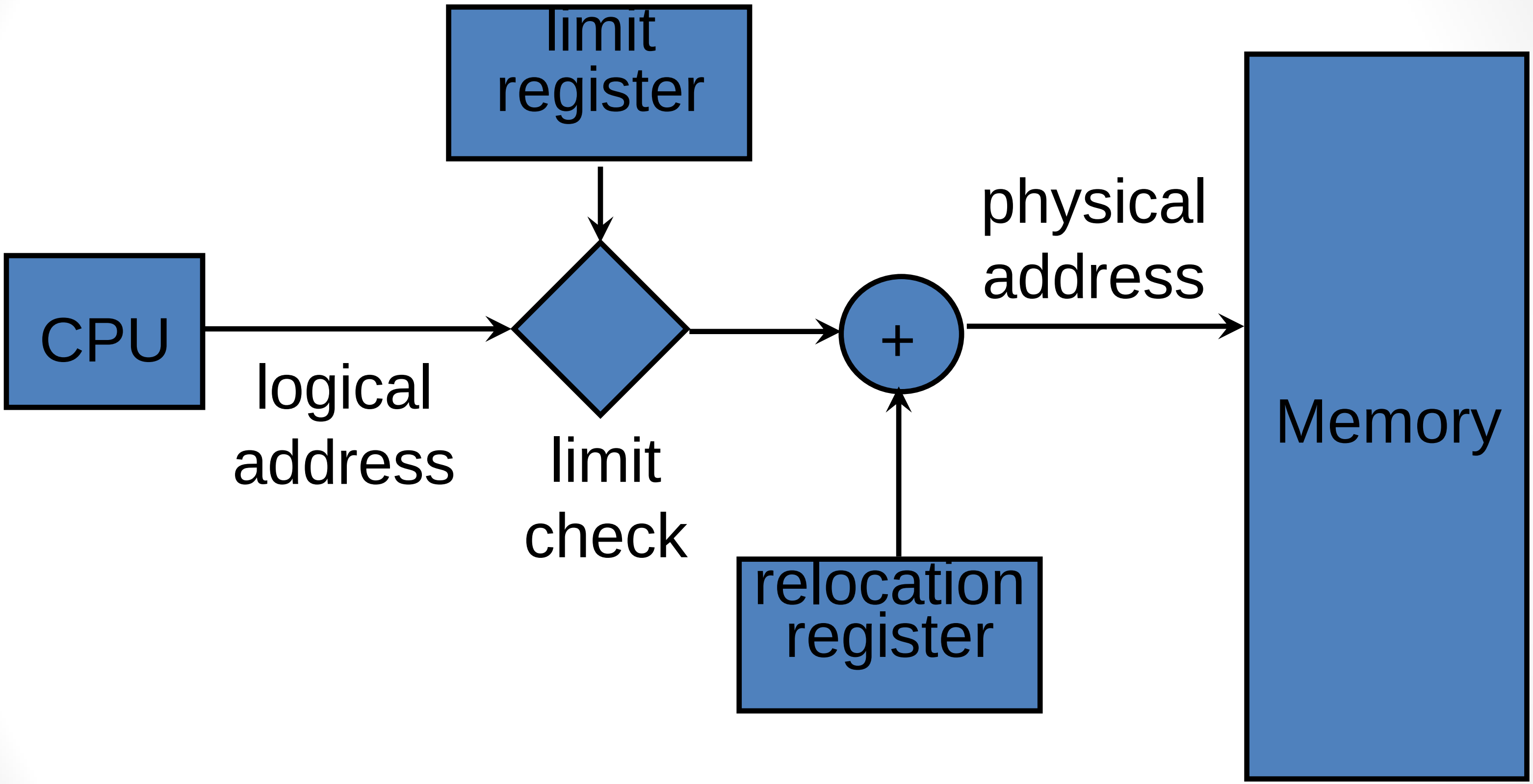
Memory Management Unit (MMU)



Memory Management Unit (MMU)



Simple MMU

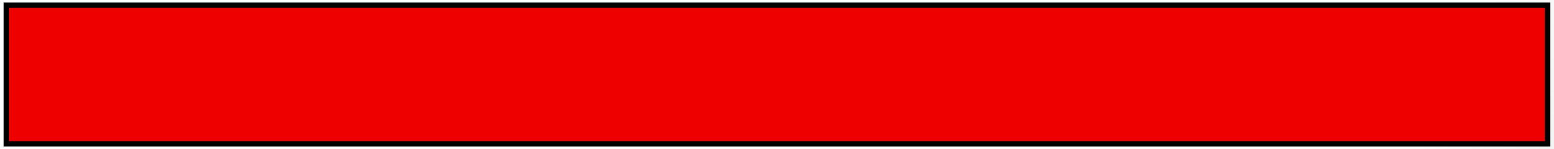

limit
register
CPU
logical
address
limit
check
+
physical
address
relocation
register
Memory

Contiguous Allocation

- main memory is divided into two parts
  - ◊ operating system (usually in same part as interrupt vector)
  - ◊ User memory (divided among processes)
- single partition allocation
  - ◊ simple allocation, each process gets a single chunk of main memory to live in
  - ◊ hardware relocation register and limit register provides relocation and memory protection.

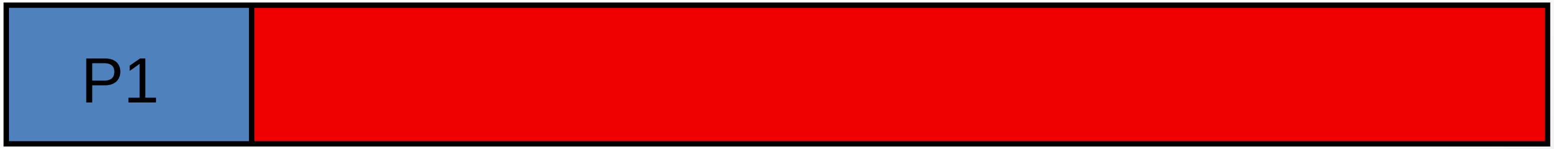
Contiguous Allocation

- When system first starts, allocation is simple
- ◊ One block of memory, and as each process starts, allocate the memory to the process



Contiguous Allocation

- When system first starts, allocation is simple
  - ◊ One block of memory, and as each process starts, allocate the memory to the process



Contiguous Allocation

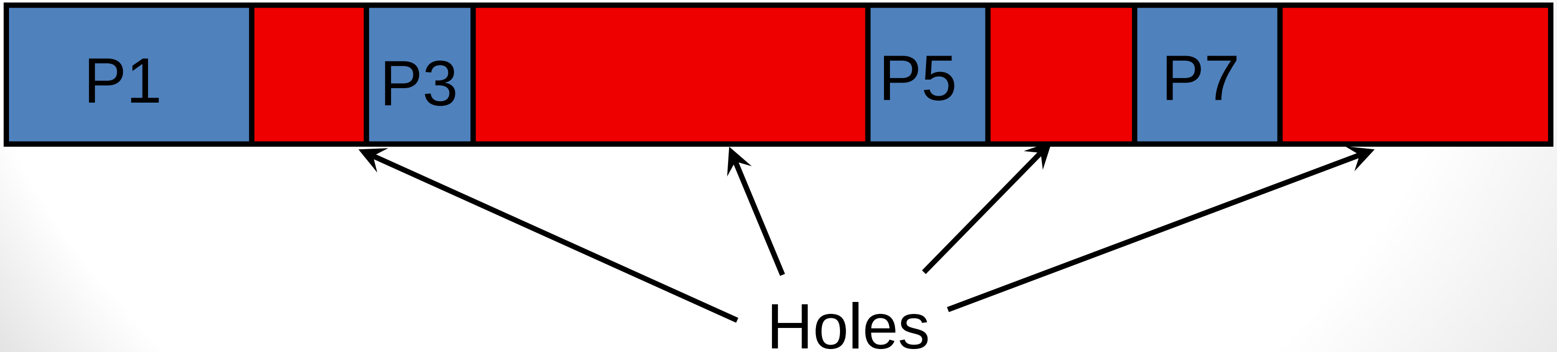
- When system first starts, allocation is simple
- ◊ One block of memory, and as each process starts, allocate the memory to the process

Contiguous Allocation

- When system first starts, allocation is simple
  - ◊ One block of memory, and as each process starts, allocate the memory to the process

Contiguous Allocation

- When system first starts, allocation is simple
  - ◊ One block of memory, and as each process starts, allocate the memory to the process
- Some processes run long, some quit soon after they start
  - ◊ Not related to size. A large program can run short or long...



Contiguous Allocation

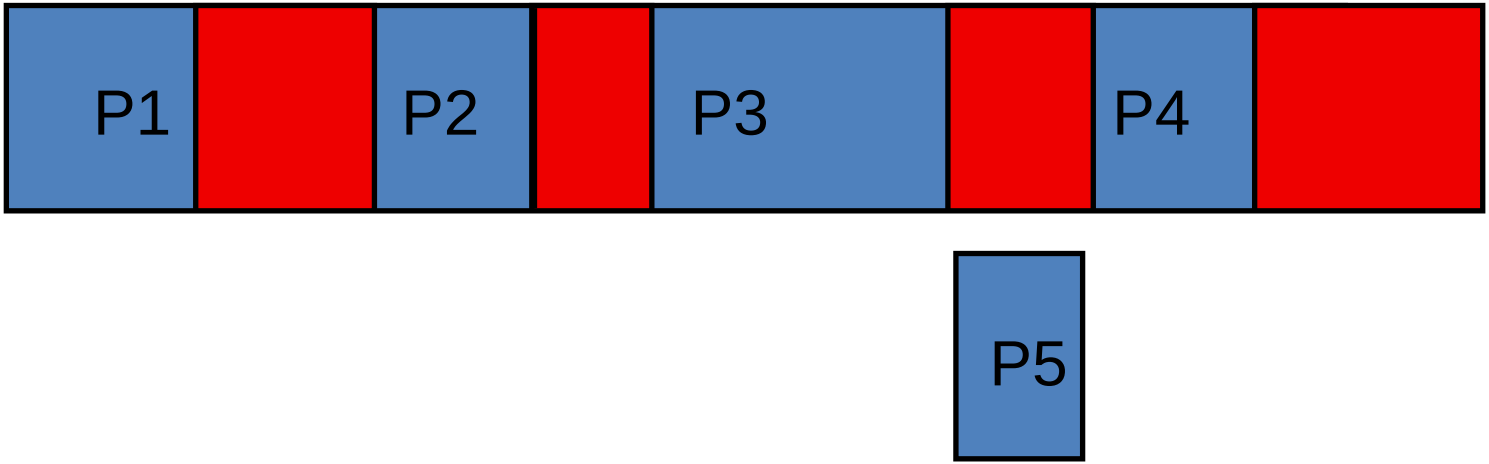
- User memory must be allocated to processes
  - ◊ fixed size segments – IBM MFT – obsolete
  - ◊ variable size segments
- OS keeps list of *holes*
  - ◊ memory not allocated to a process
  - ◊ when a process is started, find a hole big enough to hold it
  - ◊ when a process ends, add the memory to the free list
- General memory allocation problem
  - ◊ merge adjacent holes
    - on allocation or on free?

Storage Allocation

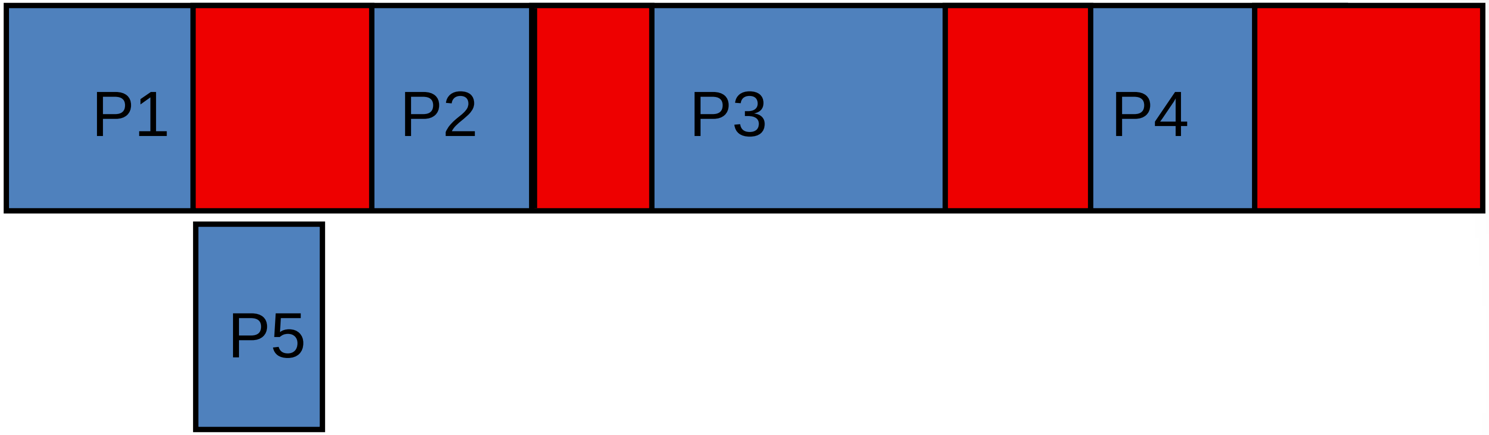
Three general approaches

- First Fit
  - ◊ use the first hole on the free list that is big enough
  - ◊ always search from beginning?
  - ◊ search from previous location
  - ◊ only look at part of list
- Best Fit
  - ◊ smallest block that is large enough
  - ◊ search entire list
- Worst Fit
  - ◊ largest block (largest remainder)
  - ◊ worst algorithm

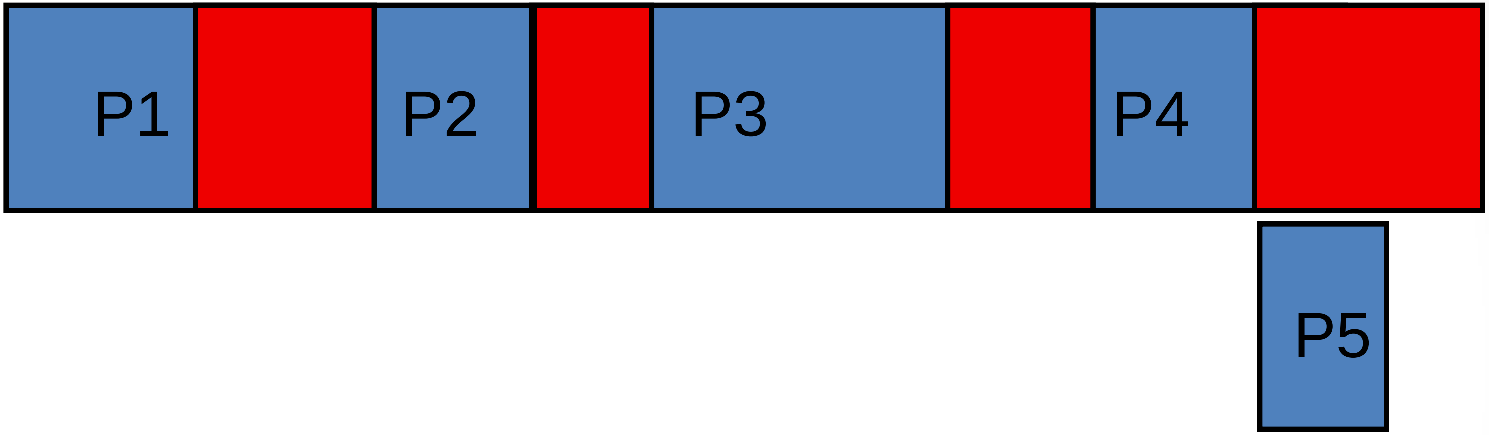
Best Fit



First Fit



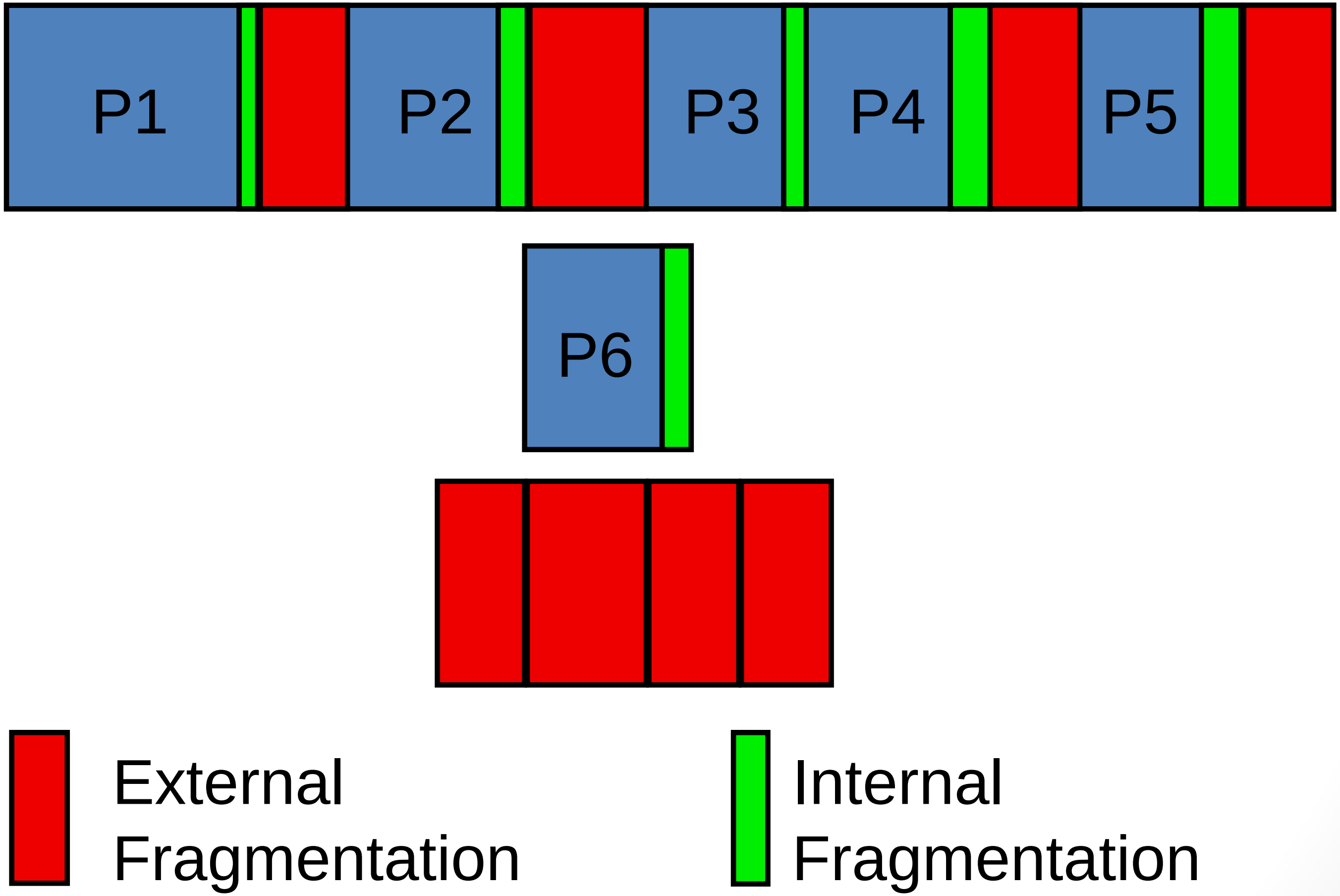
Worst Fit



Fragmentation

- Internal Fragmentation
  - ◊ if we allocate memory in units larger than a single byte (say 1K)
  - ◊ last block is only partially used
- External Fragmentation
  - ◊ lots of small holes spread throughout memory, none big enough to satisfy a request
  - ◊ worst fit tries to reduce this
  - ◊ compaction - move blocks (requires execution-time binding)
  - **50 percent rule** - N allocated blocks, 0.5 N lost to fragmentation (1/3 of memory unusable)

Fragmentation



Dynamic Loading

- memory is always in short supply
- not all routines are loaded when the program is loaded
  - ◊ only loaded when needed
  - ◊ some routines are rarely if ever used
  - ◊ does not require any special support from operating system
- some execution environments support dynamic loading (IBM mainframe, Java VM)
  - ◊ external programs are called by name, OS provides binding

Dynamic Linking

- *Static* linking is when the all of the modules including system libraries are linked together at compile time.
- *Dynamic* linking provides stubs for each routine.
  - ◊ when the routine is called the first time, the routine is loaded
  - ◊ primarily used for shared libraries
    – libraries commonly used by many programs
    e.g. strcpy, fopen, fclose.
    – allows updates and bug fixes without relinking
  - ◊ if libraries are to be shared between processes, then operating system must provide support (memory protection changes)

Overlays

- common on older systems (MS-DOS)
- no OS support required (although OS can get in the way)
- program is broken into multiple parts
- one common part of program always in memory
- other parts of program are replaced as needed
- common in early games for MS-DOS
  - different levels of the game might have different code parts, as each level is loaded, the code overlays the previous code
- also common in tools like compilers and assemblers
- complex details in overlays, not common today

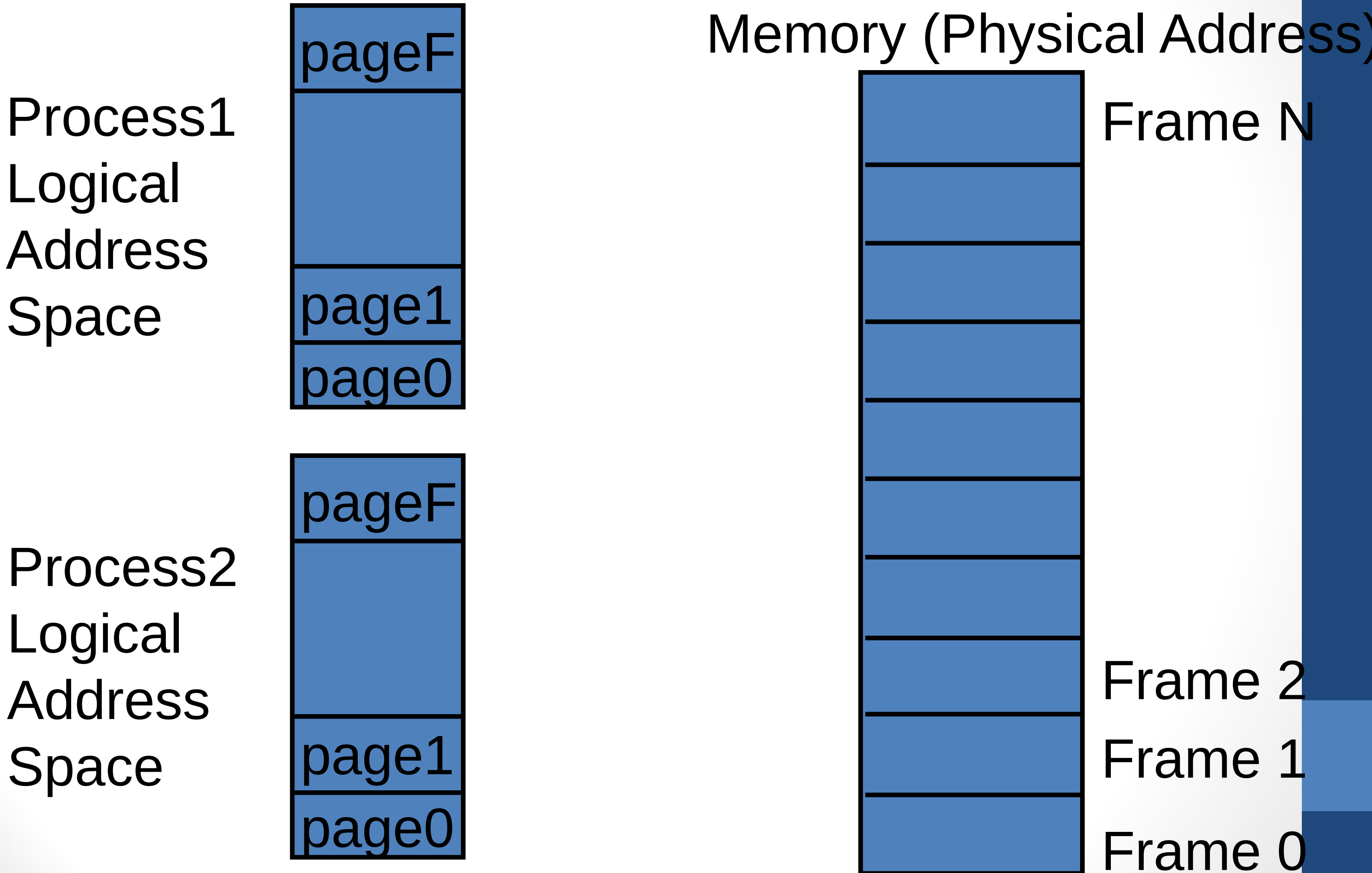
Swapping

- processes can be temporarily stored (*swapped*) from memory to a *backing store*
  ◊ very fast hard drive - continuous store
- If memory binding is not execution time, then process must be swapped back into same place in memory
  ◊ PDP-11 Unix used swapping to relocate and resize processes
- make room for higher priority processes
- major time is transfer time - amount of memory swapped.
- Used with some modifications on many systems

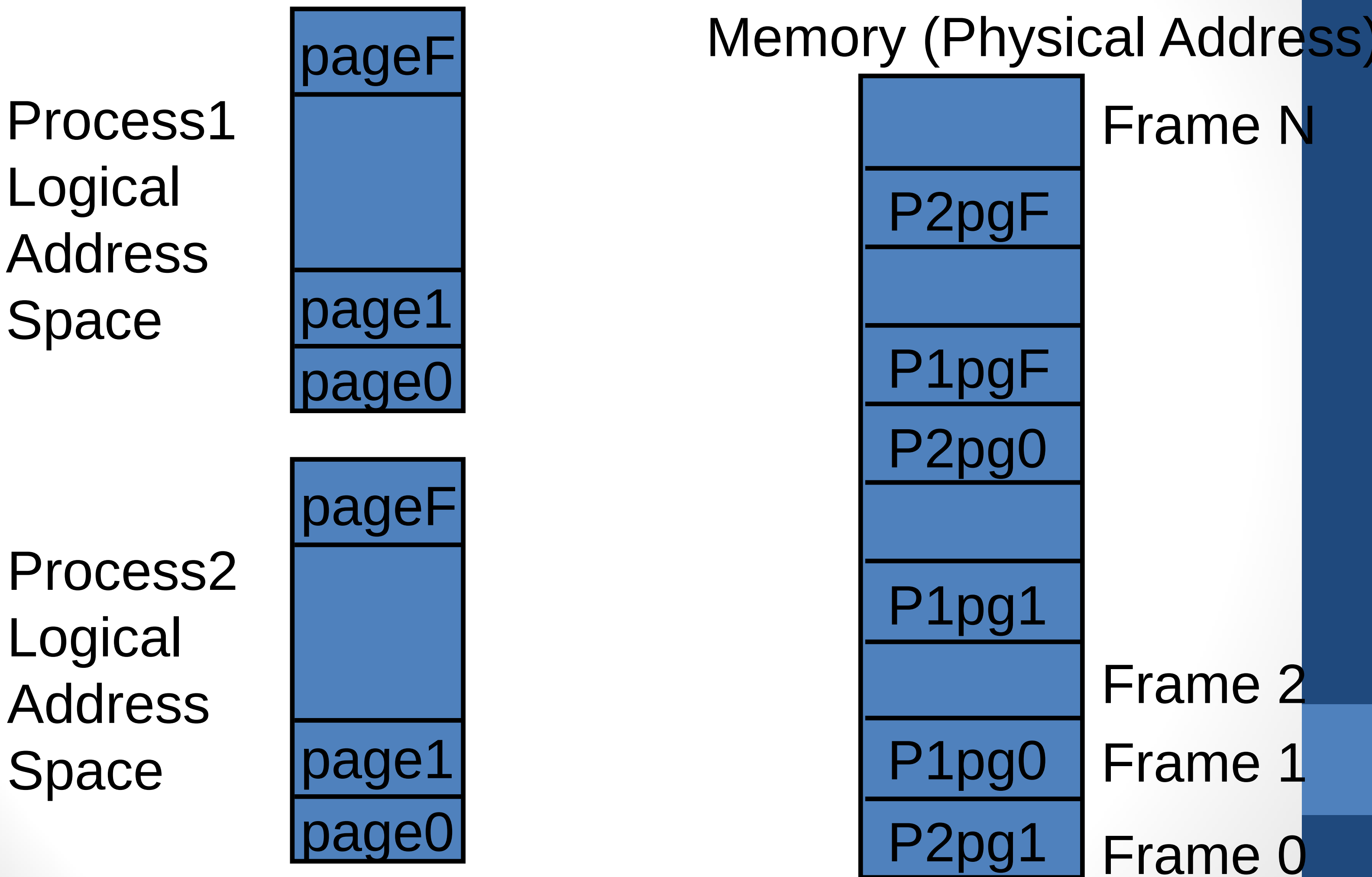
Paging

- Why should memory have to be contiguous
- Physical memory is divided into frames (512 bytes to 8K sizes typically)
- Logical memory is divided into pages (same size as frames)
- If process needs n pages, find n free frames in memory
  - ◊ no need to be contiguous
- Table translates from each page to the appropriate frame
- No external fragmentation, but still have internal fragmentation

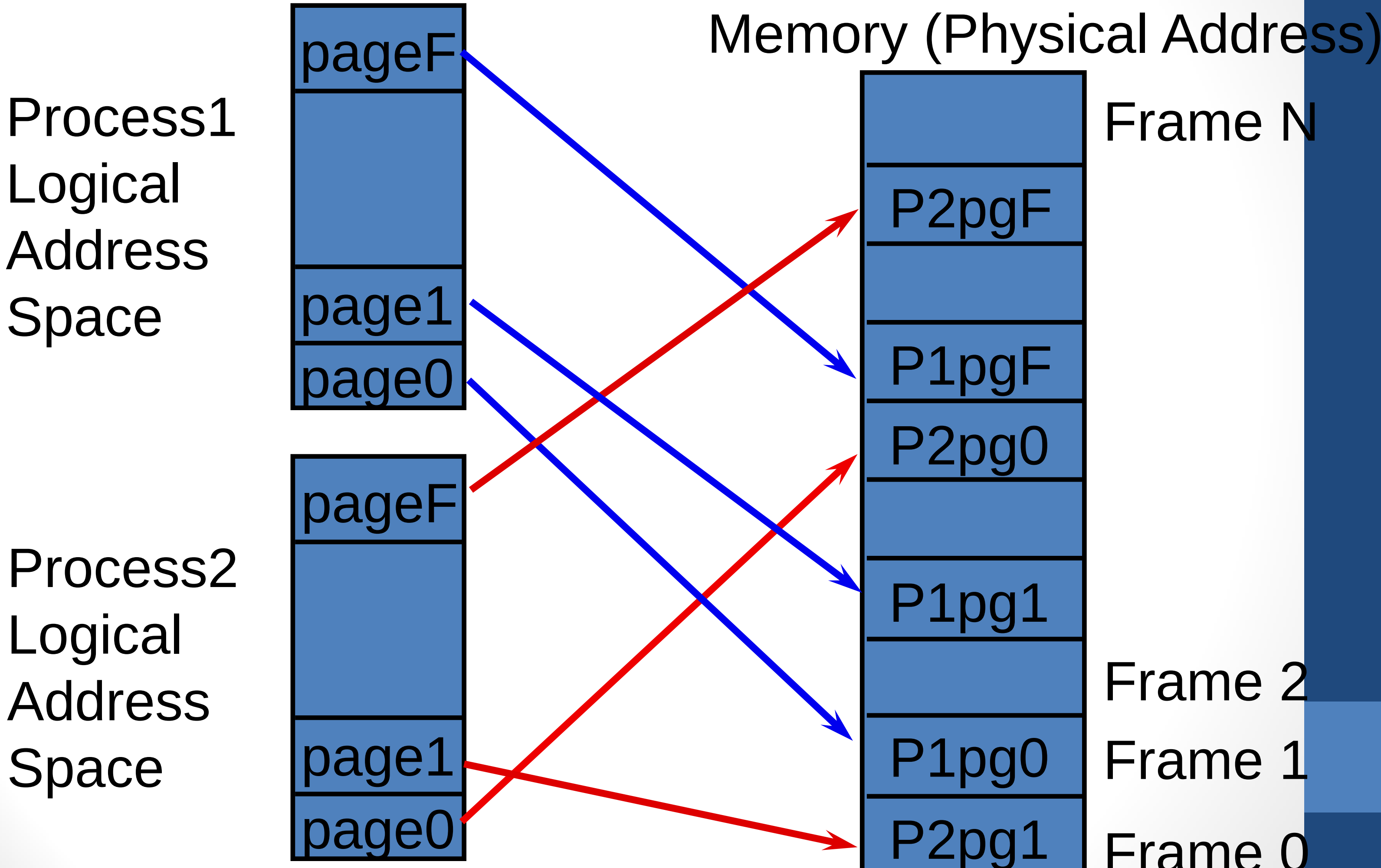
Memory is mapped by Page Tables



Memory is mapped by Page Tables



Memory is mapped by Page Tables



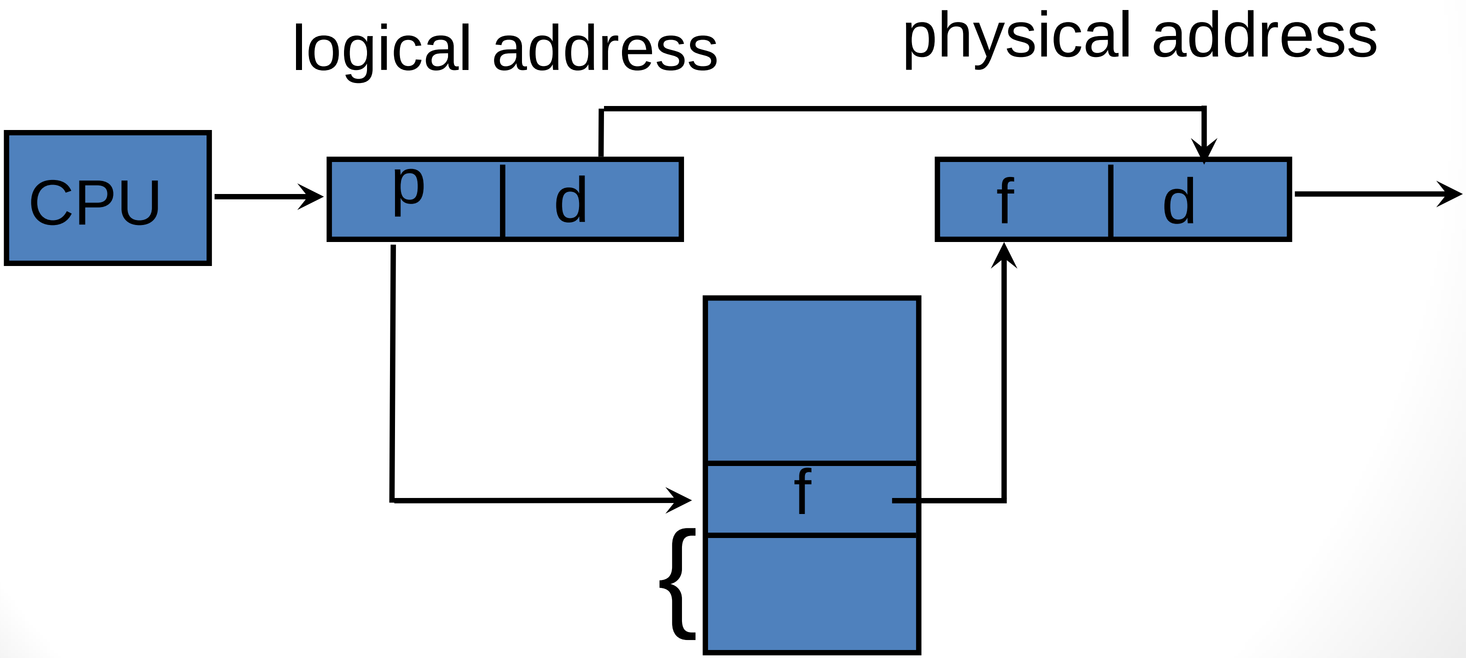
Paging

- Logical Address Space and Physical Address space may not be the same size!!
  ◊ physical address may be larger
  (e.g. 32 bit logical, 40 bit physical)
  ◊ physical address may be smaller
  (64 bit logical = 1.8e19 bytes)

- Frame and page always the same size
  – always power of 2

Page Table

- address generated by CPU is divided into two parts
  - ◊ page number(p) - index into page table
  - ◊ page offset(d) – location within the page



Memory is mapped by Page Tables