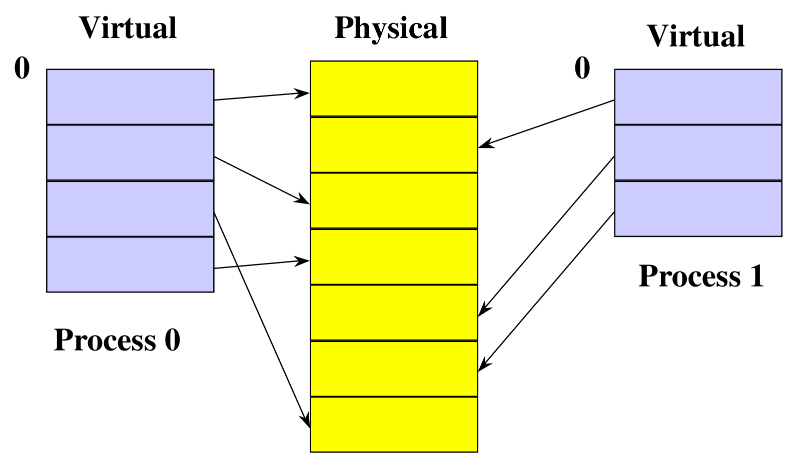
Paging: Example



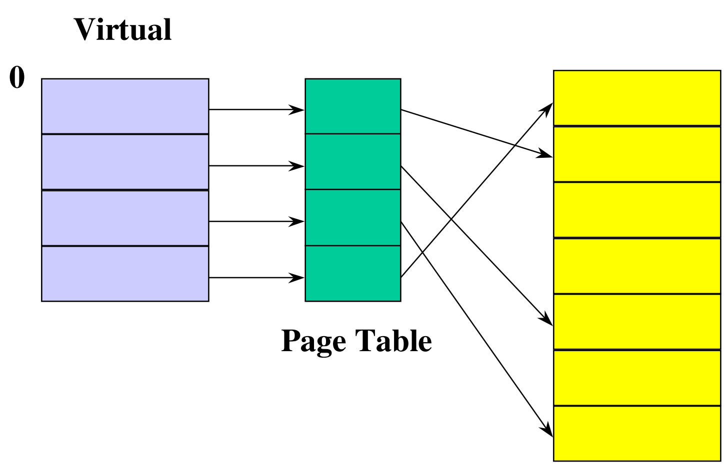
Key Facts

- Virtual address spaces of different processes are *independent*
  - Two or more may have the same address range
  - Yet the mappings differentiate between them
- A virtual page has no storage of its own
  - It must be backed by a physical frame (real page) that provides the actual storage
  - A contiguous virtual space need not be physically contiguous

Key Facts

- Physical address space is *independent* of virtual address spaces
  - They can have different sizes
  - Allows process size to be independent of available physical memory size
- Page size is always a power of 2, to simplify hardware addressing

Page Tables



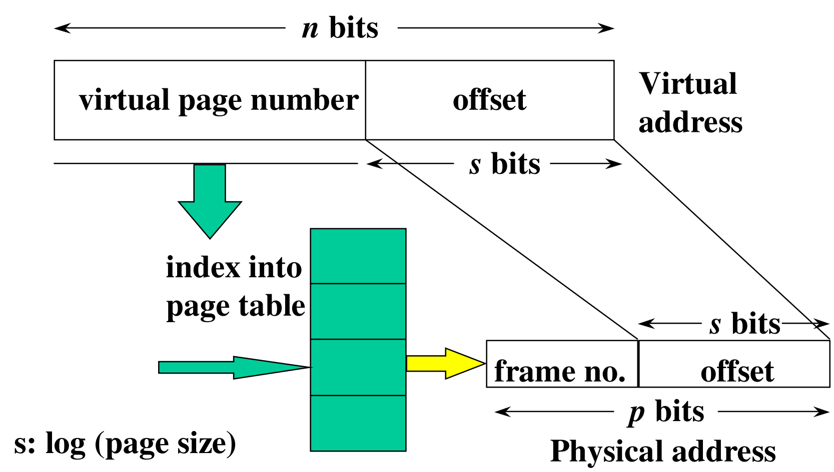
Page Table Structure

- Indexed by virtual page number
- Contains frame number (if any)
- Contains protection bits
- Contains reference bit

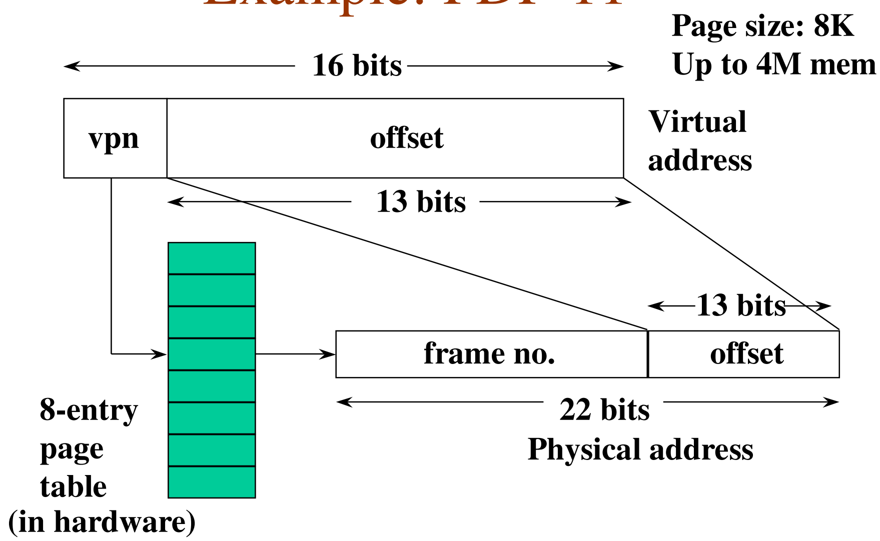
Frame No.	**v**	**w**	**r**	**x**	**f**	**m**
Frame No.	**v**	**w**	**r**	**x**	**f**	**m**
Frame No.	**v**	**w**	**r**	**x**	**f**	**m**

v: valid bit
w: write bit
r: read bit
x: execute bit (rare)
f: reference bit
m: modified bit

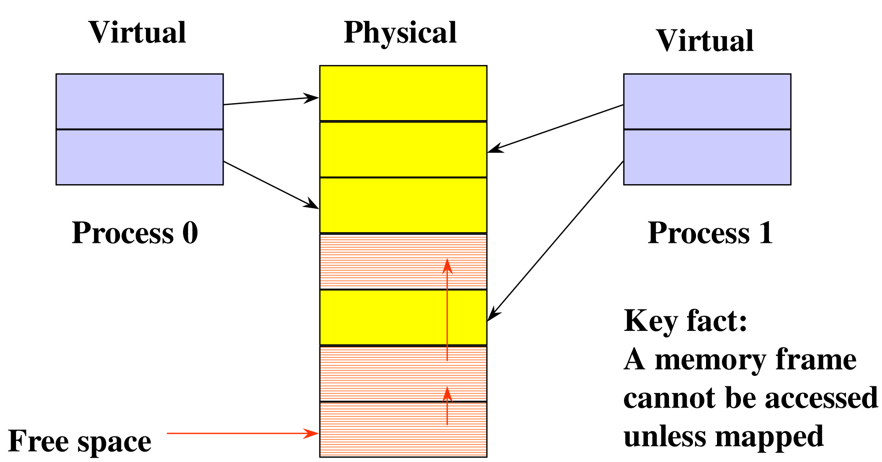
Mapping Virtual to Real Addresses



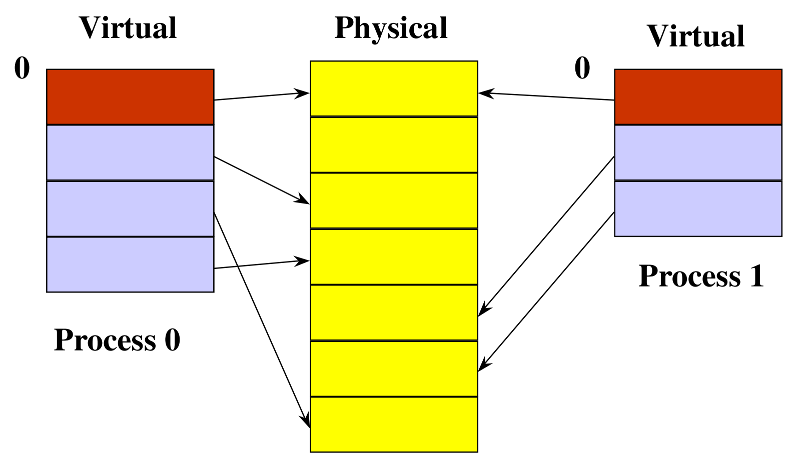
Example: PDP-11



Weird Stuff: Free Page Management



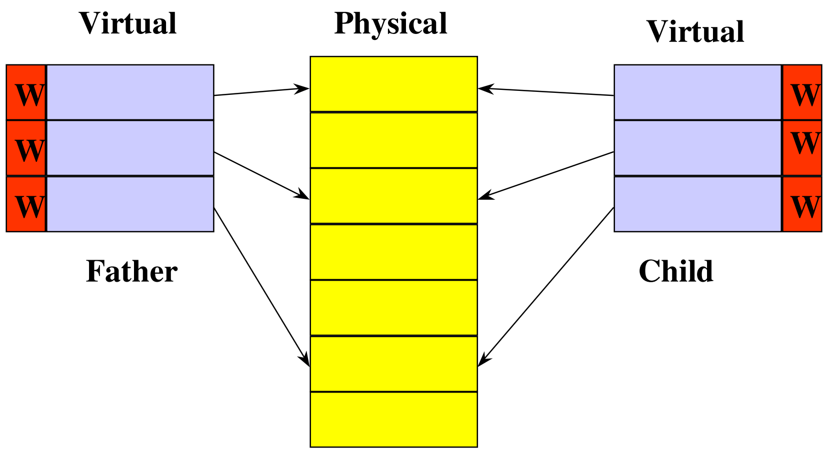
Fun Stuff: Sharing



Sharing

- Processes can share pages by mapping their virtual pages to the same memory frame
- In UNIX and Windows, code segments of processes running the same program to share the pages containing executables
  - saves a lot of memory
  - saves loading time for frequently run programs
- Fine tuning using protection bits (rwx)

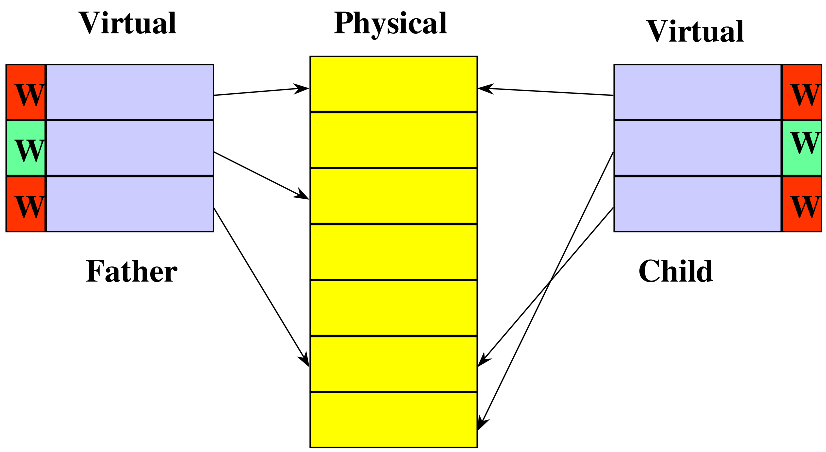
Fork Revisited: Copy-on-Write



Copy-on-Write Fork

- Initially no memory copying
  - Efficient
  - If an exec follows, no much harm
- Page tables set the protection to disallow writes
- If either father or child attempts to write, a page fault occurs

Copy-on-Write



Requirements for Sharing

- Page frames must have a reference count
  - Cannot be deallocated unless unmapped
  - Adds complexity to memory manager
- Protection bits must be set properly

Page Table Implementation

Page Table Implementations

Key issues:

- Each instruction requires one or more memory accesses:
  - Mapping must be done very quickly
- Where do we store the page table?
  - It is now part of the context, with implications on context switching
- Hardware must support auxiliary bits

PDP-11 Example Revisited

- Page table is small (8 entries)
  - Can be implemented in hardware
  - Moderate effect on context switching time
- Each process will need an 8-entry array in its PCB to store page table when not running
- Protections works fine

But: what if address space is 32-bit

Page Table Size Problems

Assume 16K page size and 32-bit address space

Then:

- For each process, there are 2^{19} virtual pages
- Page table size: 2^{19} * 4 bytes/entry
  - Page table requires 2 Mbytes
  - Cannot be stored in hardware, slowing down the mapping from virtual to physical addressing