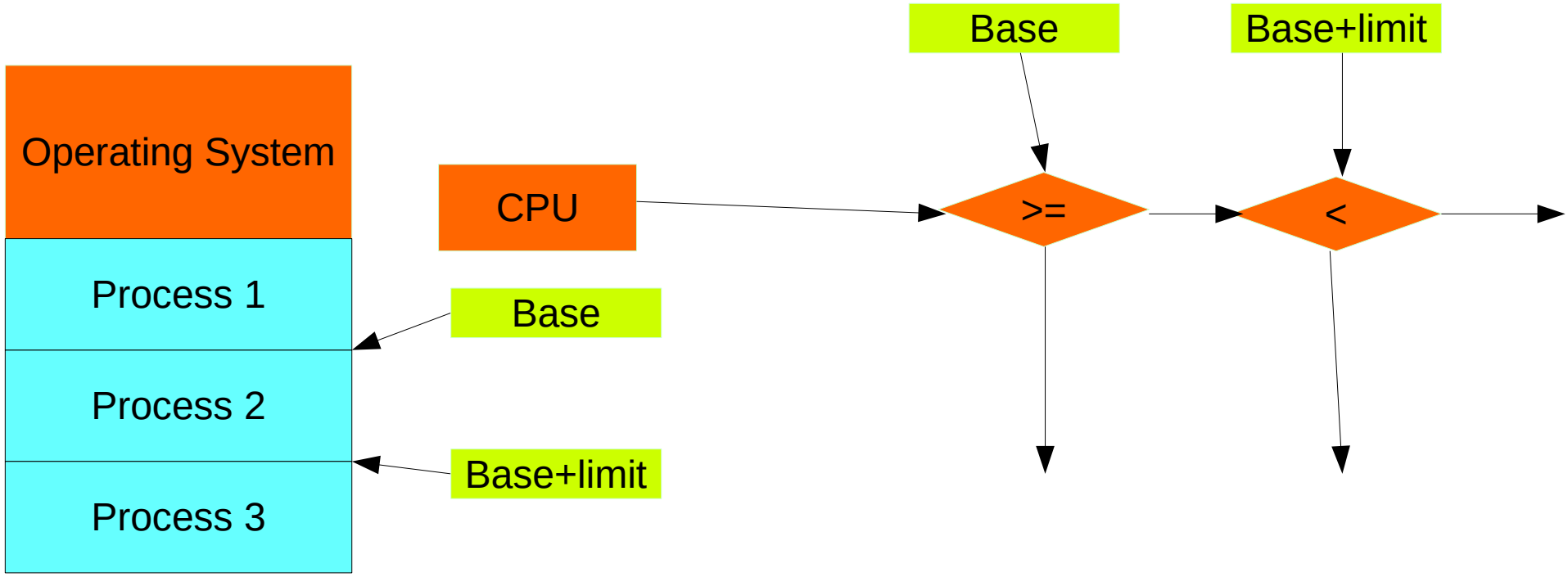
Operating Systems

Main Memory
Based on Ch. 9 of
OS Concepts by SGG

Basic Hardware

- Memory holds the instructions (text), data, etc of a process.
- There can be many processes in the system all with their memory.
- A simple way to think of them is that they occupy consecutive areas in the memory
- We should also have some minimal protection.

Basic Hardware



Address Binding

- The addresses inside an executable program are fixed in several steps
  - Source code: symbolic (like variable `i`)
  - Object code: relative to module
  - Executable file: most of them are logical (or virtual) addresses, some of them to be determined
    - Virtual address is what you get if you print a pointer as integer
  - Running code: all of them logical (or virtual)
- Throughout the execution of a program on a modern computer the addresses are understood to be logical addresses.

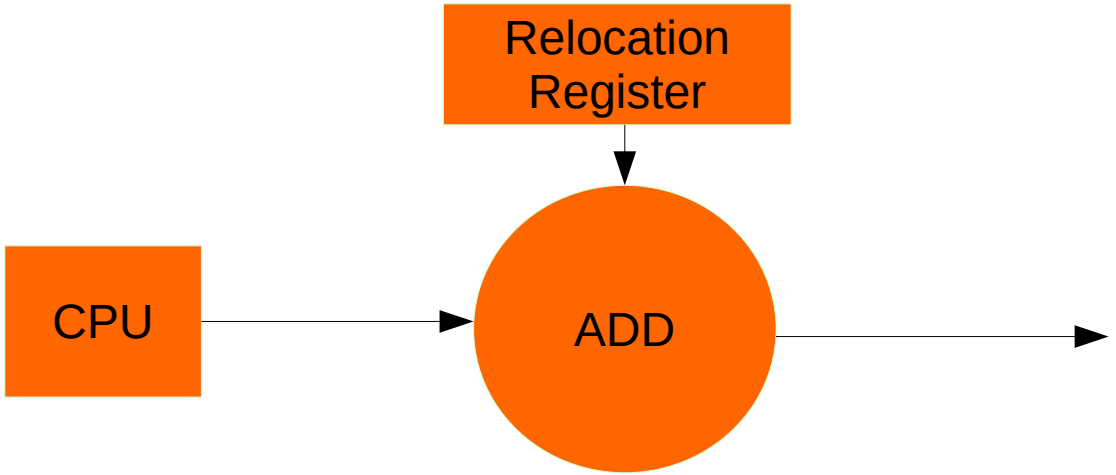
Logical vs Physical

- All programmers write code as if their code is alone in the system
  - Their address space starts at 0
  - They are safe from interference from other programs
  - They have no means to interfere with other programs' address space
- This is not what really happens, this is what it *appears* to happen

Logical vs Physical

- The program emits reads and writes from and to the memory with logical addresses
- The MMU (Memory Management Unit) translates these to physical addresses
- The MMU can be a simple relocation register and an adder or a monster set of page tables

Relocation Register



Dynamic Loading

- Programs can load (copy from disk into the memory) modules of a program as needed
  - Old systems did it to save RAM
  - Modern systems do it to speed up initialization
- They can unload them as well
- The linking has been done in advance

Dynamic linking and Shared Libraries

- The object files in a program are usually statically linked
- Most of the libraries are linked dynamically
  - With a notable exception: the dynamic linker itself!
- Addresses are finally resolved during linking
- This allows the same library to be linked twice but loaded once
  - If, for example, two programs make use of the same library.

Swapping

- When a system has many processes running, it may run out of memory
- Then a few processes are copied to the disk
  - Tricky: how do you handle pending I/O
    - Either wait until I/O is done
    - Or let the I/O happen only between the device and the kernel memory
- Mobile systems usually do not swap
  - But swap out all processes instead of terminating them to make restarting easier.

Memory Allocation: Contiguous

- The simplest way to allocate memory
- Easy to do memory protection
- When we deallocate memory, we can re-use it
  - If we are lucky the next request will exactly fill the hole left by the freed memory
- After many rounds of allocation and de-allocation we are left with lots of little holes in between allocated memory.

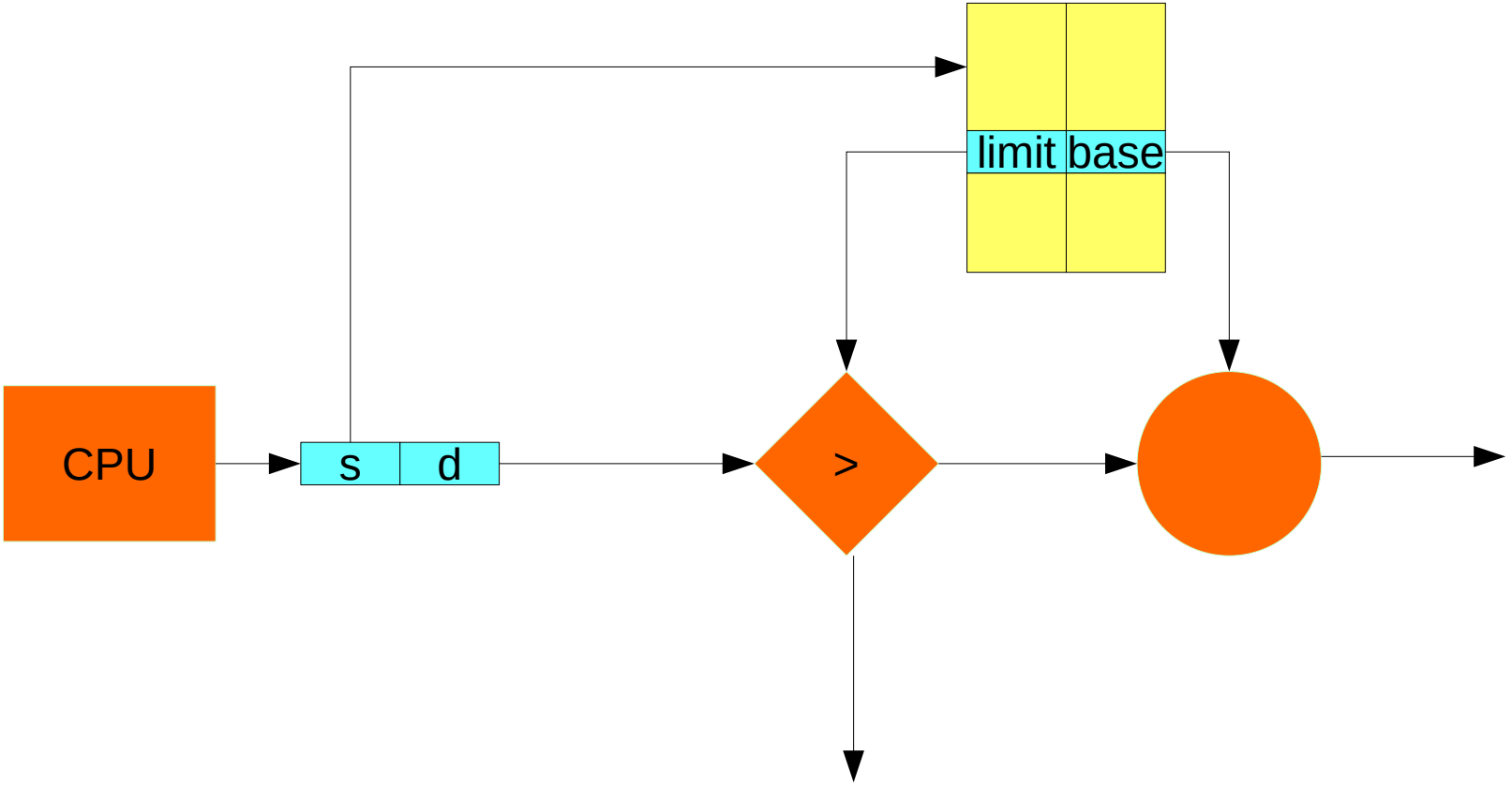
Memory Allocation: Contiguous

- If we use variable size partitions we end up with up to 1/3 of memory lost to little holes
  - This is called external fragmentation
    - Because the fragments are outside the allocated space
- We allocate memory space using some rule
  - Best fit: Find the free chunk of memory closest in size
  - First fit: find the first that fits
  - Worst fit: fit it into the bigest to leave a usable leftover
- Best fit and first fit work the best. First fit is the cheaper of the two.

Segmentation (old fashioned)

- Programmers think of memory as text, data, heap, stack, libraries, etc
- It is natural to design a system that preserves this view
- We use variable size segments that can increase and shrink in size as needed

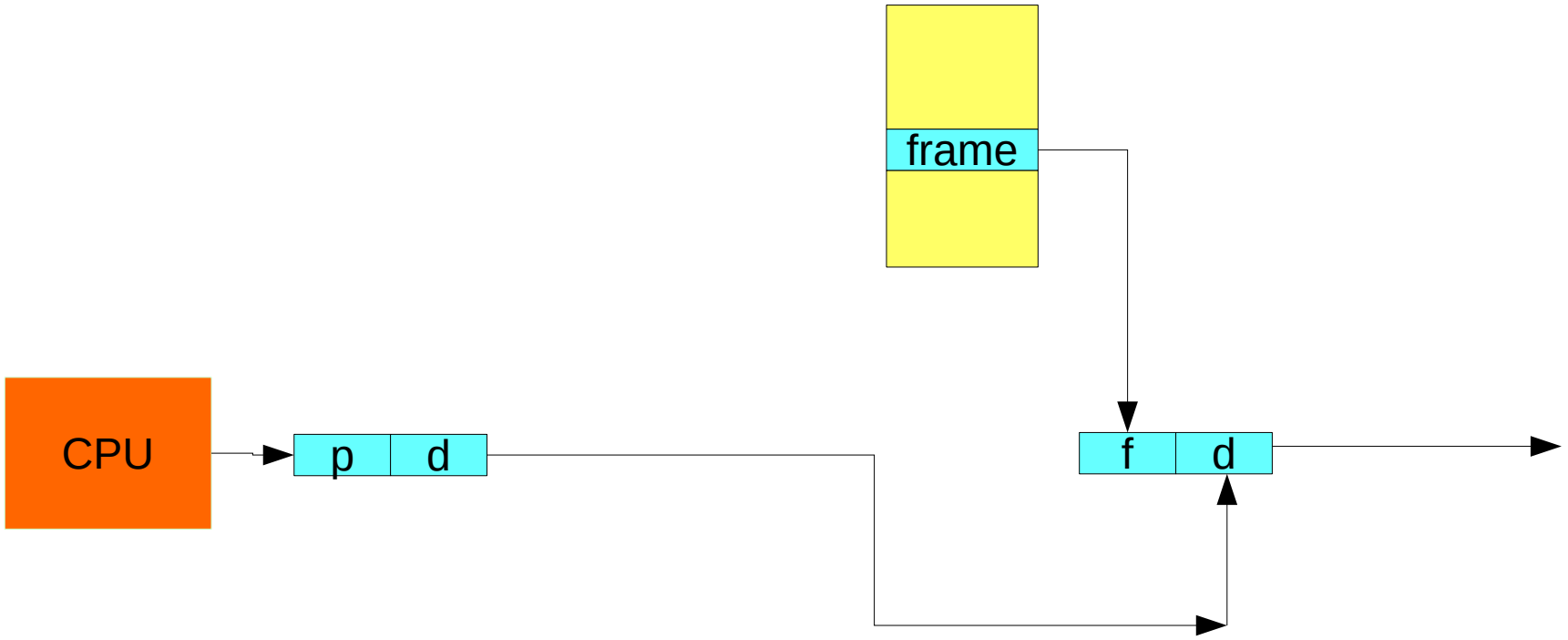
Segmentation Hardware



Paging

- Like segmentation allows bits and pieces of the address space to be everywhere
- Pages are fixed size (kind-of)
- Avoids external fragmentation
  - But has internal fragmentation (if page size is small, it is not a problem)
- The address space is cut into chunks called pages
- The main memory is cut into chunks called frames
- We can find the page size with
  - getconf PAGESIZE

Paging



Hardware Support

- Looking up the page tables creates extra memory references
- Page tables become bigger, memory falling behind the CPU in performance, the delay becomes unacceptable.
- Time for some hardware acceleration

Hardware support

- Most processes read and write the same few pages for extended time
- We cache the page table entries of these pages into a fast associative memory
- It is called TLB. Looking up the TLB is part of the pipeline
- If the entry is not there we call it **TLB Miss**
- If it is there the extra cost of paging is (almost) zero.

TLB

- When we have a TLB miss we go to the real page table. This takes several cycles.
- When we get the needed page table entry we put it in the TLB
  - This means we have to kick something out, like any cache
- At every context switch the TLB is fluched out to avoid confusion
- We can avoid this by using ASID (Address Space ID)

Miss Rate, Miss Penalty

- TLB misses are expensive
- Example: If the cost of a regular memory access is 1ns, and the cost of a miss is 20ns and the miss rate is 1%, what is the overall cost of accesssing the memory
  - .99*1+.01*20 = 1.19
- The calculations are more complex because we have many other factors affecting the performance

Protection

- What happens if a user process tries to access an invalid part of the page table
  - It will contain garbage...
- To avoid havoc we introduce one extra bit on each page table entry: the valid bit (or valid invalid)
- In smaller address spaces we could use a page-table-length-register

Shared pages

- Paged memory allows page sharing
- Very useful if several processes make use of the same library
  - They do not load a copy each. They all share a single copy
  - The code has to be re-entrant and position independent.
- Can be used as an interprocess communication mechanism

Page Table Structure

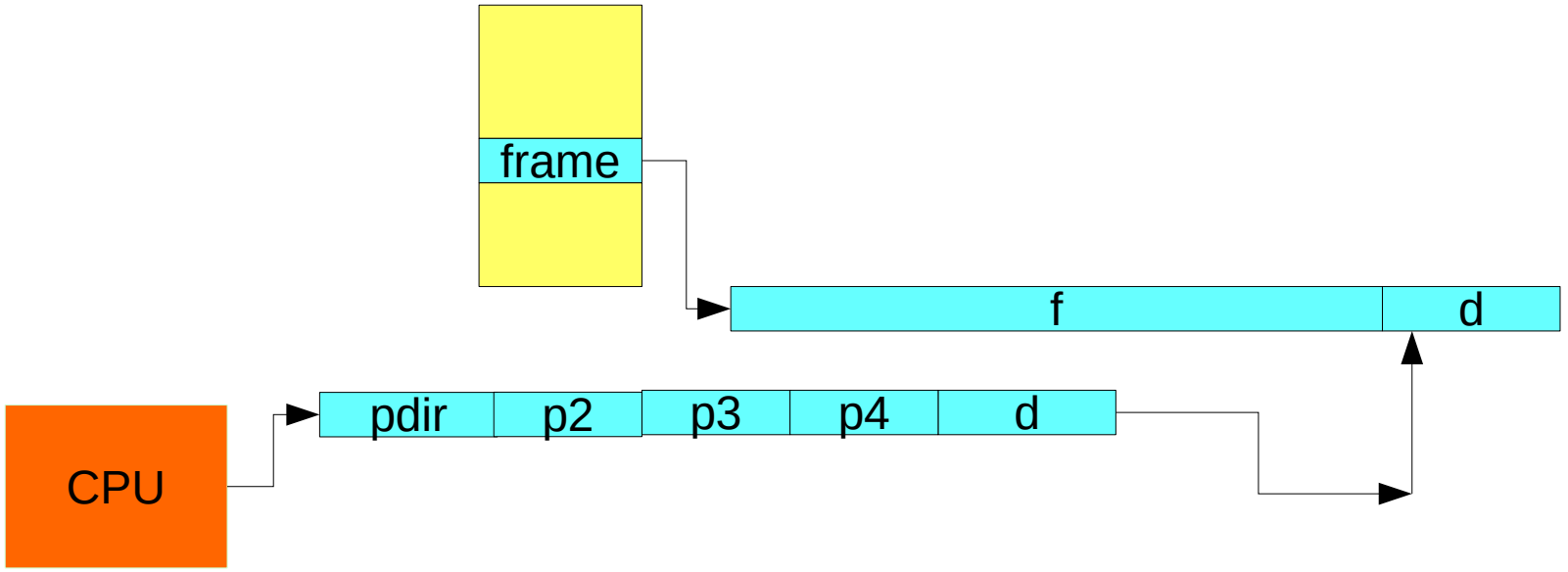
- In a 32 bit system of old
  - Page size: 1K (10 bits)
  - Page table size: 8-16MB
  - Back then few computers had that much memory altogether!
- In a 64 bit system
  - Page size: 4K (12 bits)
  - Page table size: 2^52 (too big to say in decimal)

Page Table Structure

- Hierarchical paging
  - For a 32 bit system
    - 12 bit offset (4K page size)
    - 1024 entry (10 bit) outer page table
    - Several 1024 entry page tables
    - Most of the page tables are not allocated
  - For a 64 bit system
    - We need about 7 layers of tables to keep the sizes under control.
    - AMD-64 has 4 levels of page hierarchy but only 48-bit virtual address space

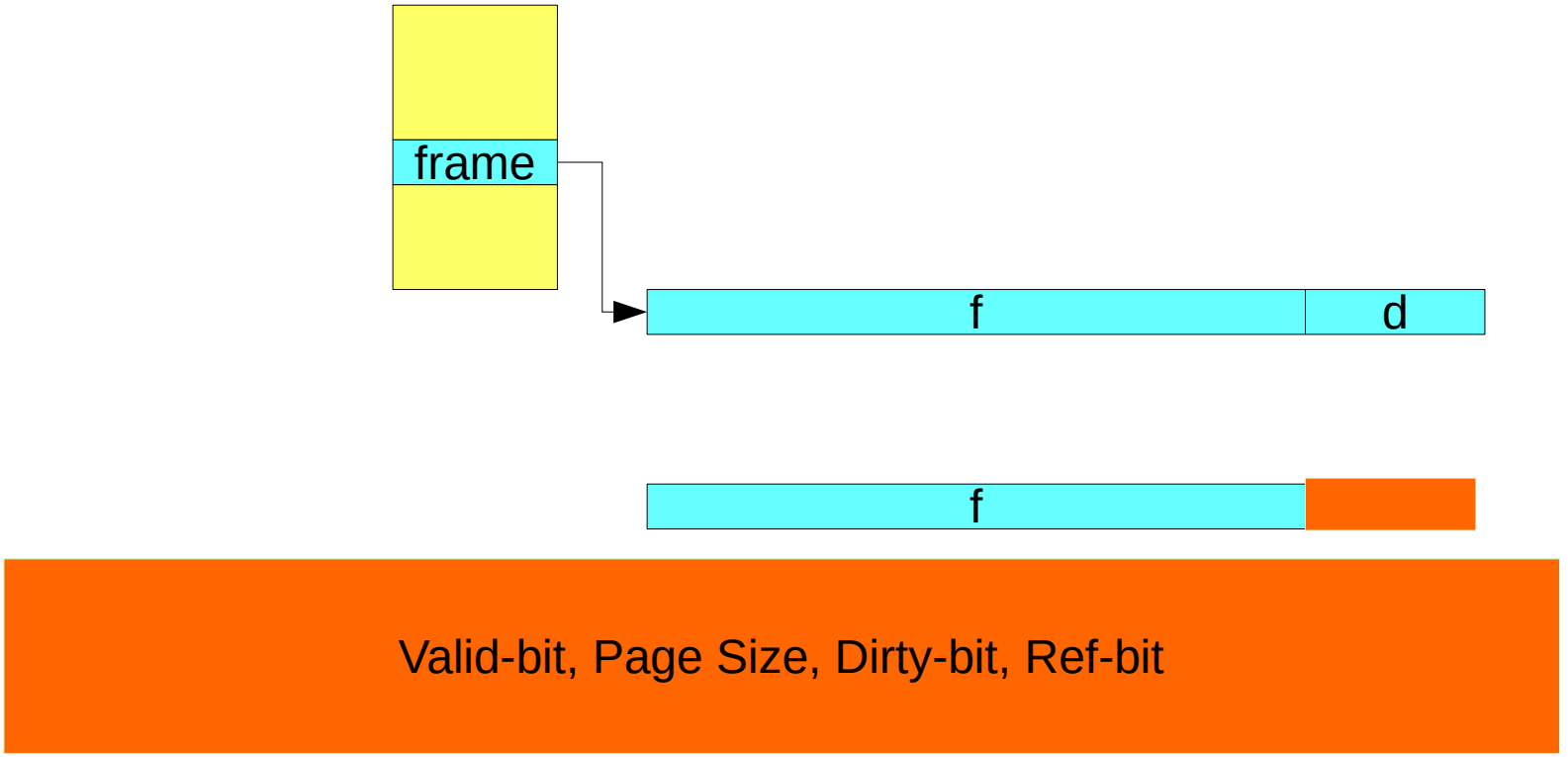
Page Table Structure

- Hierarchical paging (x86-64)



Page Table Structure

- Hierarchical paging (x86-64)



Hierarchical page tables

- The x86-64 architecture supports 4KB (12 bit), 2MB (21 bit) and 1GB (30 bit) page sizes.
- Every page table entry has a few extra bits (right next to the valid bit) that indicates the size of the page.
  - 1GB pages need two extra memory accesses
  - 2MB page need three extra memory accesses
  - 4KB page need four extra memory accesses

Hashed Page Tables

- Instead of having a huge page table (or multi level tree) we use a hashing function and map the addresses to a smaller table.
- When there is a conflict we chain
- Every entry has three fields:
  - Page, frame, next
- There are varieties that work better for clustered keys (most pages are clustered)
  - A group of clustered pages can be represented by a single entry in the hash table.

Inverted Page Tables

- Every process is supposed to have a page table
- Most processes use a tiny part of their virtual address space
  - Big waste
- The physical memory is of fixed size
- One idea is to index the table by the frame number
- When we get a request to find a frame, we scan the whole table
  - That’s STUPID

Inverted Page Tables

- Instead we use a technique for data storage and retrieval
- The best seems to be hashing. We have one hash table for the whole system.
- The difference now is that we have to keep an ASID to distinguish various address spaces
- We can reduce the size of the hash table by having one entry per group of contiguous pages.
- Sharing pages is tricky.

Inverted Page Tables

- Example: SPARC Solaris (by Oracle)
  - Has two hash tables (user and kernel)
  - Has a TSB (Translation Storage Buffer), besides the good old TLB.
    - It is a bigger but slower TLB that resides in main memory.
  - On a TLB miss, the TSB is searched.
  - On a TSB miss the hash table is searched.

Swapping

- If we run out of memory we can swap processes out to the disk and give its pages to other processes.
- This is rarely needed anymore. A version of this is used for suspending the CPU (swap out everything)
- Most often we page out to the disk the pages that are not used as often
  - More on this in the next chapter.
  - Not used on SSDs.
  - Mobile devices use a form of swapping.

IA-32 Paging

- Uses two level hierarchical page tables
  - 10-10-12 bits
- If the valid bit is zero this may mean that the page table (or the page) has been paged out to the disk.
- Some OSes can use Page Address Extension
  - 32 bit virtual address but much bigger physical address
- Similar to IA-32 but with 3 different page sizes, and four levels of hierarchical page tables.

ARM Paging system

- Has many versions. One of them is very similar to X86-64
- They love to use different names
  - Granule size for the smallest page size
  - Region size for other page sizes
- It uses two levels of TLB