

# Sharing Physical Memory

S. Brobst

Advanced Computer Systems Architecture

Lecture #10

October 8, 1986

## Addressing Mechanisms

Storage: A collection of <name, value> pairs.

### Basic Operations:

**Read**(name) - Return the value associated with a given name.

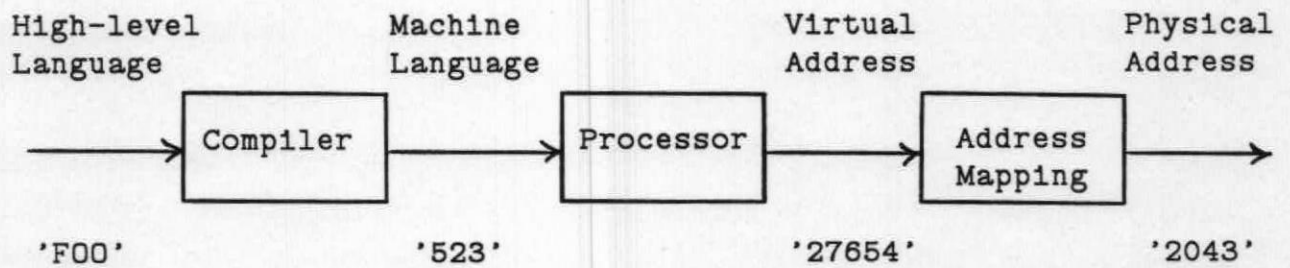
**Write**(name, value) - Associate a value with the specified name.

What is the set of names?

What is the set of values?




## Addressing Mechanisms

- Programmer given names.
- Compiler generated names.
- Processor generated names.
- Physical names (addresses).



## Addressing Mechanisms

Names and values have different specifications depending on the level of interpretation.

<u>Level</u>	<u>Name</u>	<u>Value</u>
<i>High-Level Language</i>	Identifier (typed/untyped)	<ul style="list-style-type: none"> <li>- Fixed size objects</li> <li>- Variable size objects</li> <li>- May contain other objects</li> <li>- Type or untyped</li> </ul>
<div style="display: flex; justify-content: center; align-items: center;"> <div style="text-align: center; margin-right: 10px;">  </div> <div style="text-align: center;">COMPILER</div> </div> <hr style="border: 1px solid black;"/>		
<i>Machine Language</i>	<ul style="list-style-type: none"> <li>- Linear address space</li> <li>- One address space for each "process"</li> <li>- Zero-based</li> </ul>	Fixed size bit strings
<div style="display: flex; justify-content: center; align-items: center;"> <div style="text-align: center; margin-right: 10px;">  </div> <div style="text-align: center;">PROCESSOR</div> </div> <hr style="border: 1px solid black;"/>		
<i>Virtual Address</i>	<ul style="list-style-type: none"> <li>- Two dimensional address</li> <li>- &lt;Page, Offset&gt;</li> </ul>	Fixed size bit strings
<div style="display: flex; justify-content: center; align-items: center;"> <div style="text-align: center; margin-right: 10px;">  </div> <div style="text-align: center;">ADDRESS MAPPING</div> </div> <hr style="border: 1px solid black;"/>		
<i>Physical Address</i>	<ul style="list-style-type: none"> <li>- Primary memory address, or</li> <li>- Disc address</li> </ul>	Fixed size bit strings

## Programming with Absolute Addresses

On early computers...

Programmer Names  $\equiv$  Processor Generated Names  $\equiv$  Physical Addresses

## Static Relocation

- Programs were moved to a static offset in memory to accommodate zero-based "system code."
- On EDSAC, program was relocated while being input.
- Later programs were relocated after being read into primary memory.

## Static Relocation

Single, contiguous allocation:

- No change in processor.
- Simple loading program.

## Static Relocation

Memory and Processor utilization:

- Some memory is always unused.
- CPU idles while I/O is performed.



## Partitioned Allocation

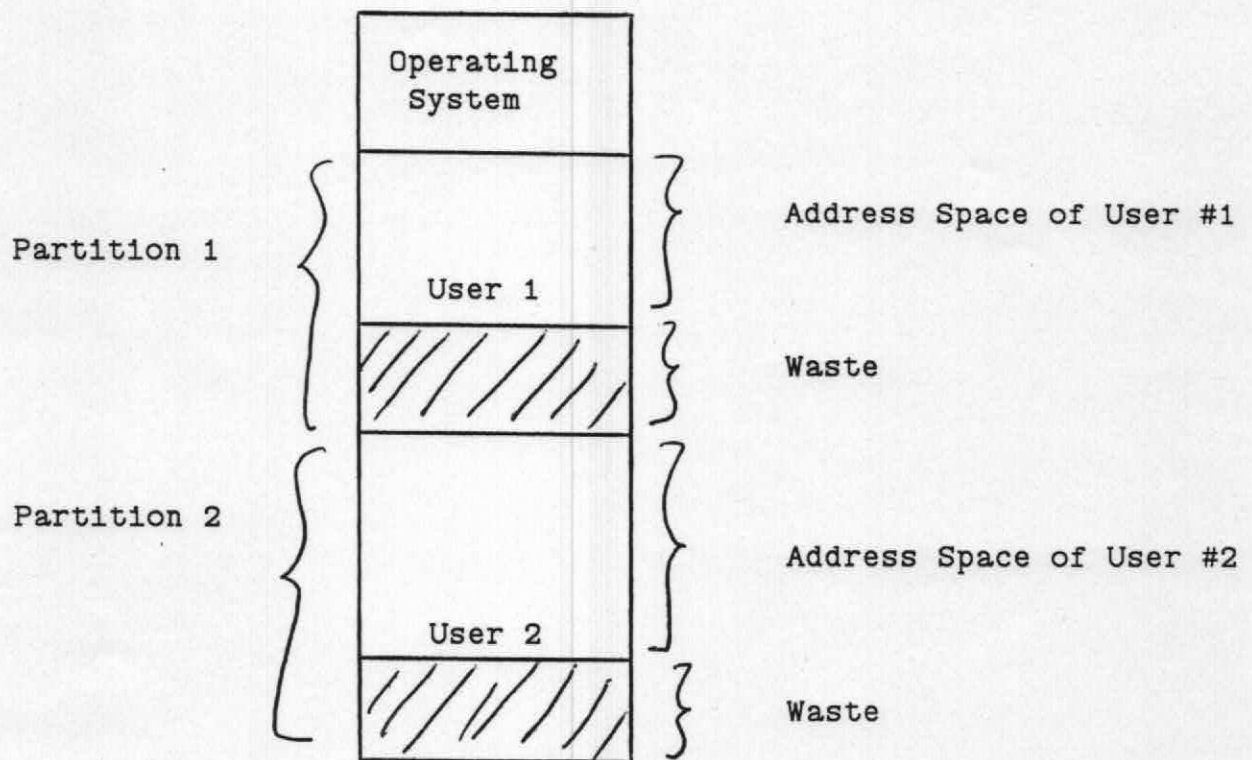
Better utilization of hardware resources is achieved if we allow CPU and I/O operations for different jobs to overlap...

### "MULTIPROGRAMMING"

- Requires memory to be partitioned among currently "active" jobs.
- Need to protect users from each other.

## Partitioned Allocation

Assign user jobs to separate portions in primary memory.



- Base and bound registers are used for protection.
- Processor updates these registers when switching jobs.
- Protection is not easy to extend to I/O operations.

## Partitioned Allocation

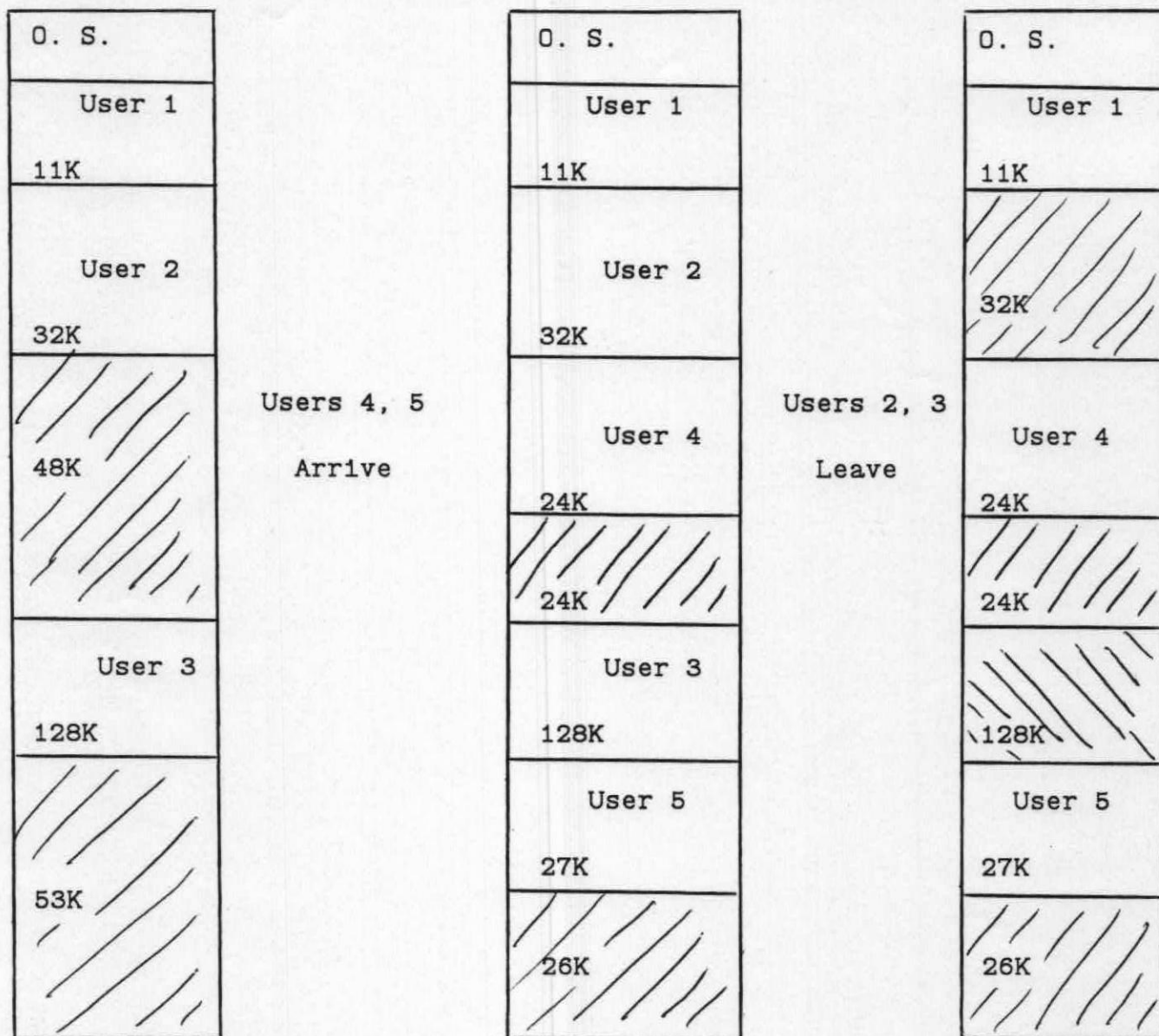
Partitions were managed in software:

Size	Location	Status
4K	68K	In use
16K	84K	Free
220K	304K	In use
220K	524K	Free

- Static partitions.
- Tremendous scope for wastage.
- User may not be able to run if job requires more memory than maximum partition provides.

## Partitioned Allocation Using Dynamic Partitions

Dynamic partitions facilitate memory allocation according to the needs of a user.

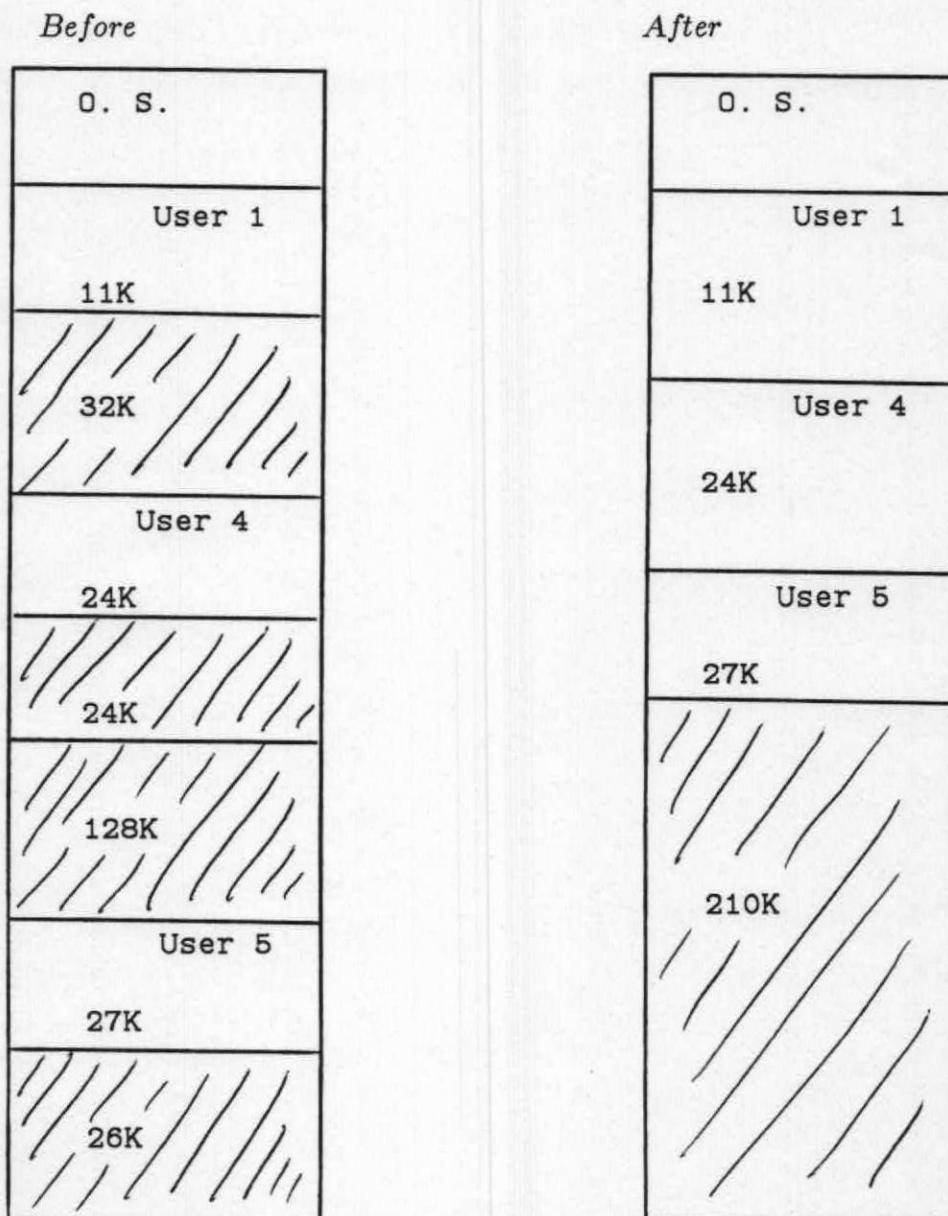


Life gets considerably more difficult:

- Scheduling
- Memory management (first fit, best fit, coalescing, fragmentation, ...)
- Operating System becomes quite sophisticated - several 100K bytes of code.

## Dynamically Relocatable Partitions

Compaction ("burping" the memory) is required to get rid of fragmented memory.



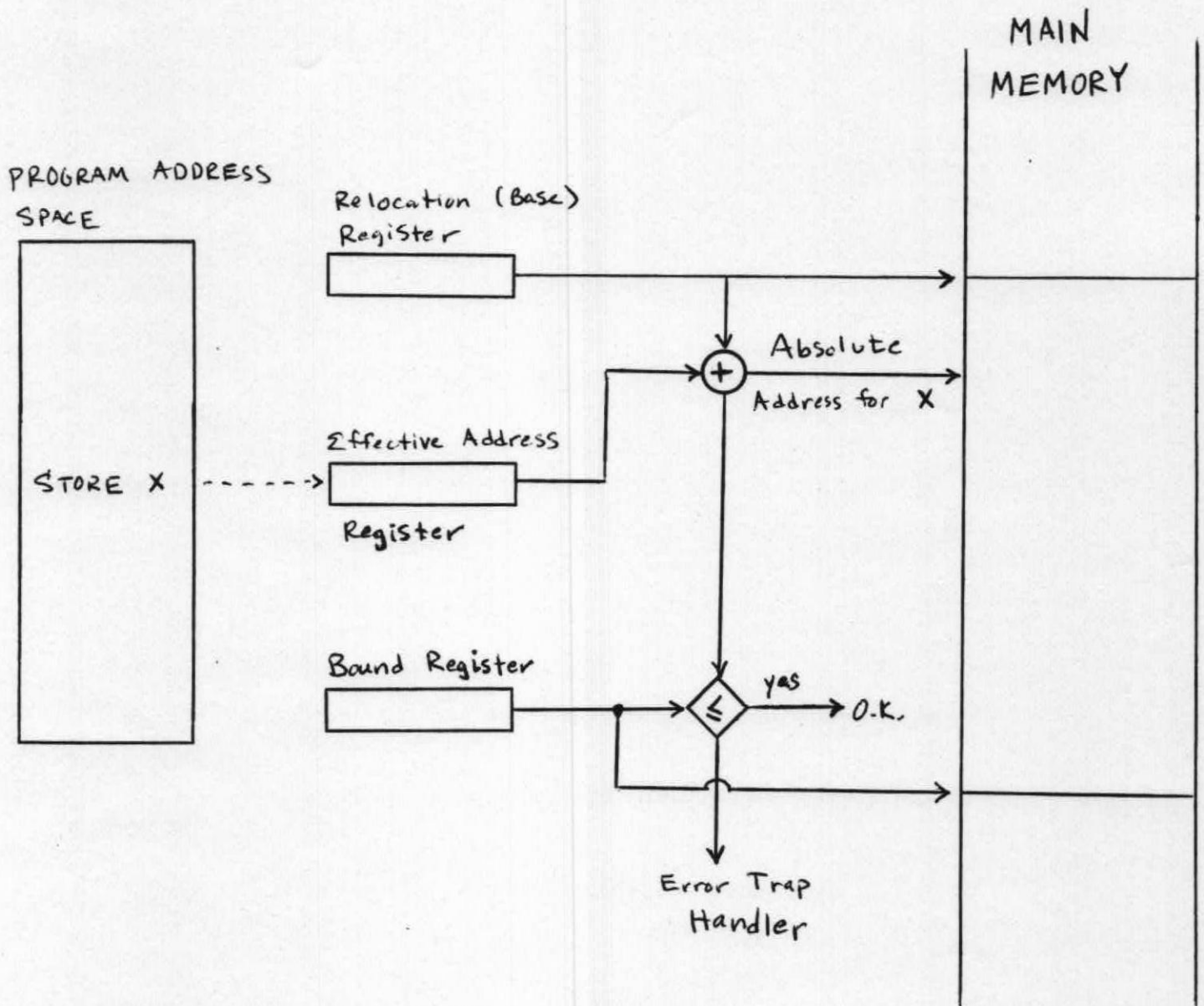
## Dynamically Relocatable Partitions

Difficulties in moving a program:

1. Base and bound registers must be updated.
2. Absolute addresses must be changed.
  - How do we know which words contain an address?
  - This is very difficult.

Consensus is to disallow the use of absolute addresses within program. all address spaces are given the illusion of starting at zero with a base register added in to yield the actual physical address.

## Dynamically Relocatable Partitions



## Dynamically Relocatable Partitions

Compact the memory (using relocation techniques) when a big enough slot is not available for an incoming job.

Unfortunately, compaction time is usually quite substantial.

Can compaction be avoided?

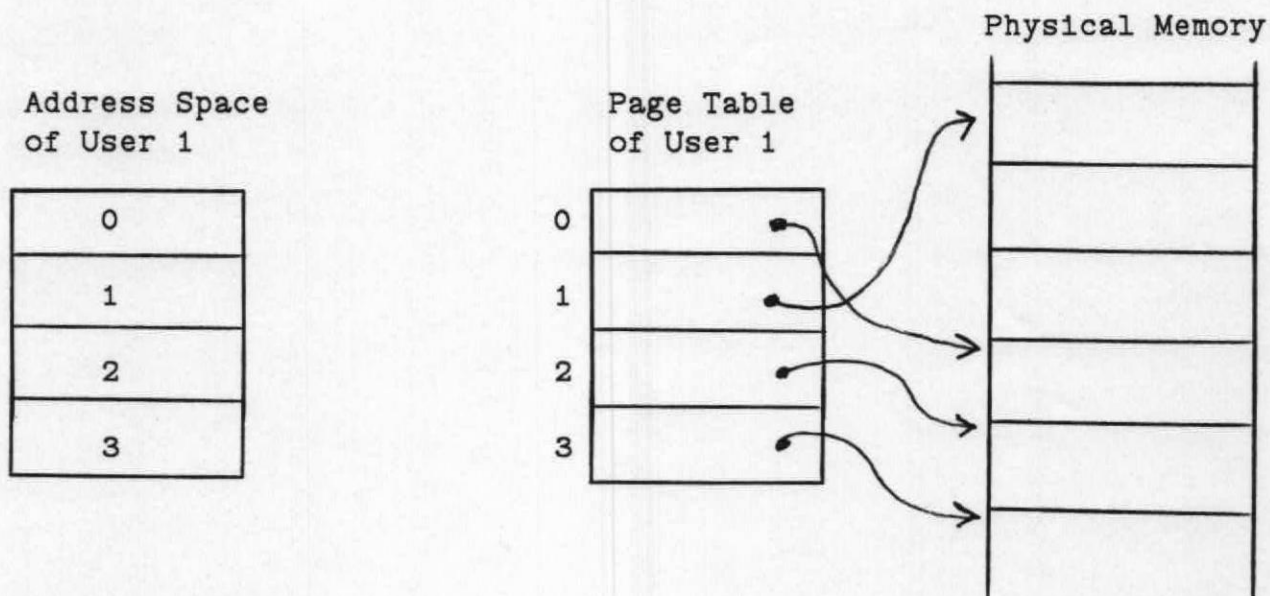


## Paged Memory Systems

In a paged memory system we relax the contiguous allocation requirement.

- Physical memory is treated as a collection of *fixed size* chunks called "pages."
- The user address space is envisioned as a set of contiguous pages, but may not be stored as such.
- Page table gives a mapping from pages in user address space to pages in physical memory.

## Paged Memory Systems



Processor generated address is interpreted as a  $\langle \text{page, offset} \rangle$  pair:

Page #	Offset
--------	--------

Hardware looks up origin of page in page table by using page number as an index into table.

## Paged Memory Systems

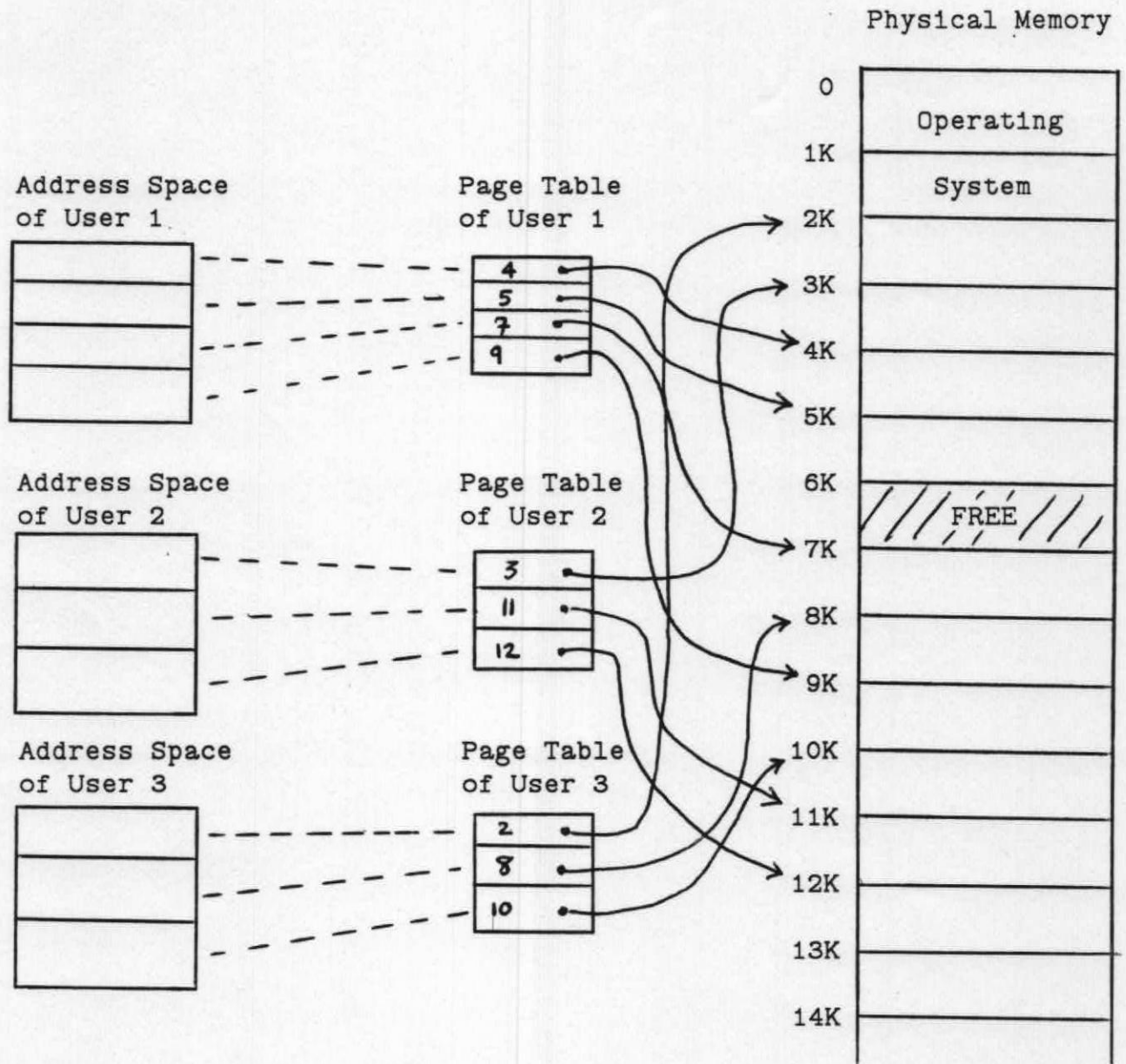
Paged systems get rid of *external* fragmentation, but introduce *internal* fragmentation.

A larger page size results in higher memory wastage due to internal fragmentation.

On the other hand, a small page size requires a larger page table.

### Paged memory Systems

We maintain one page table *per user*; each containing one entry for each user page.



In addition, a single page *frame* table is kept for the whole system to keep track of free page frames, etc.

## Paged Memory Systems

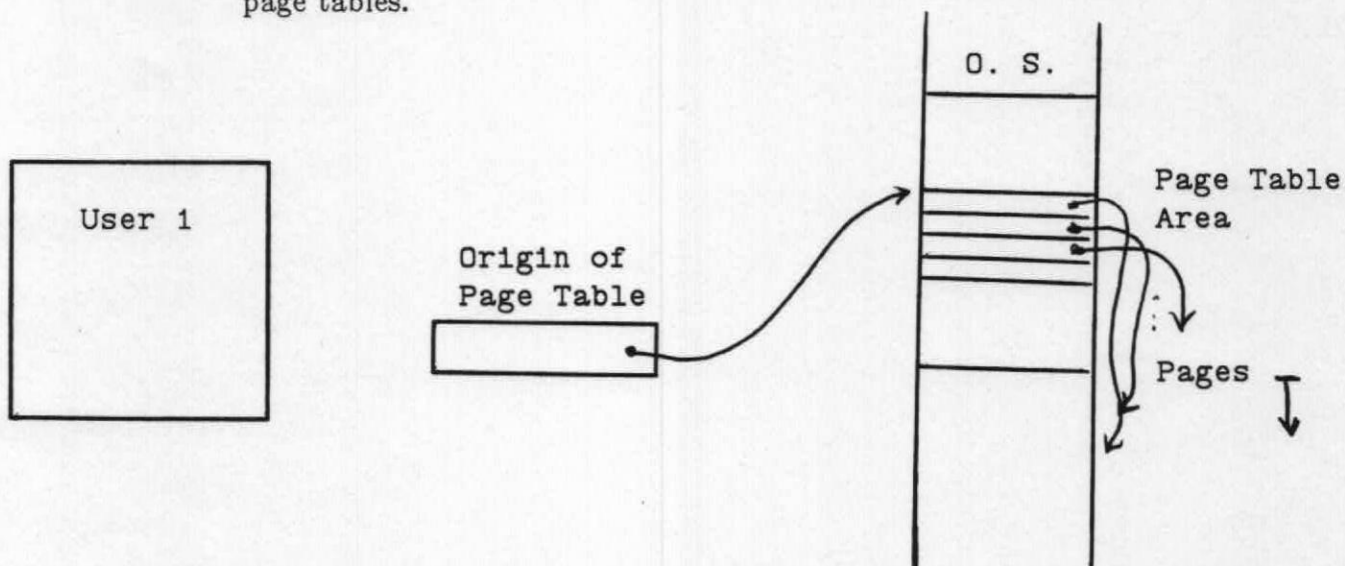
Where should page tables reside?

1. In a special set of registers.

- Expensive to switch users because page table registers must be saved and loaded.
- Expensive in hardware.
- Used in XDS 940.

2. In main memory.

- Memory reference overhead is 100%.
- Should page tables reside in a special area or in just another "page".
- Need two memory management schemes: one for pages and one for page tables.



## Demand Paging

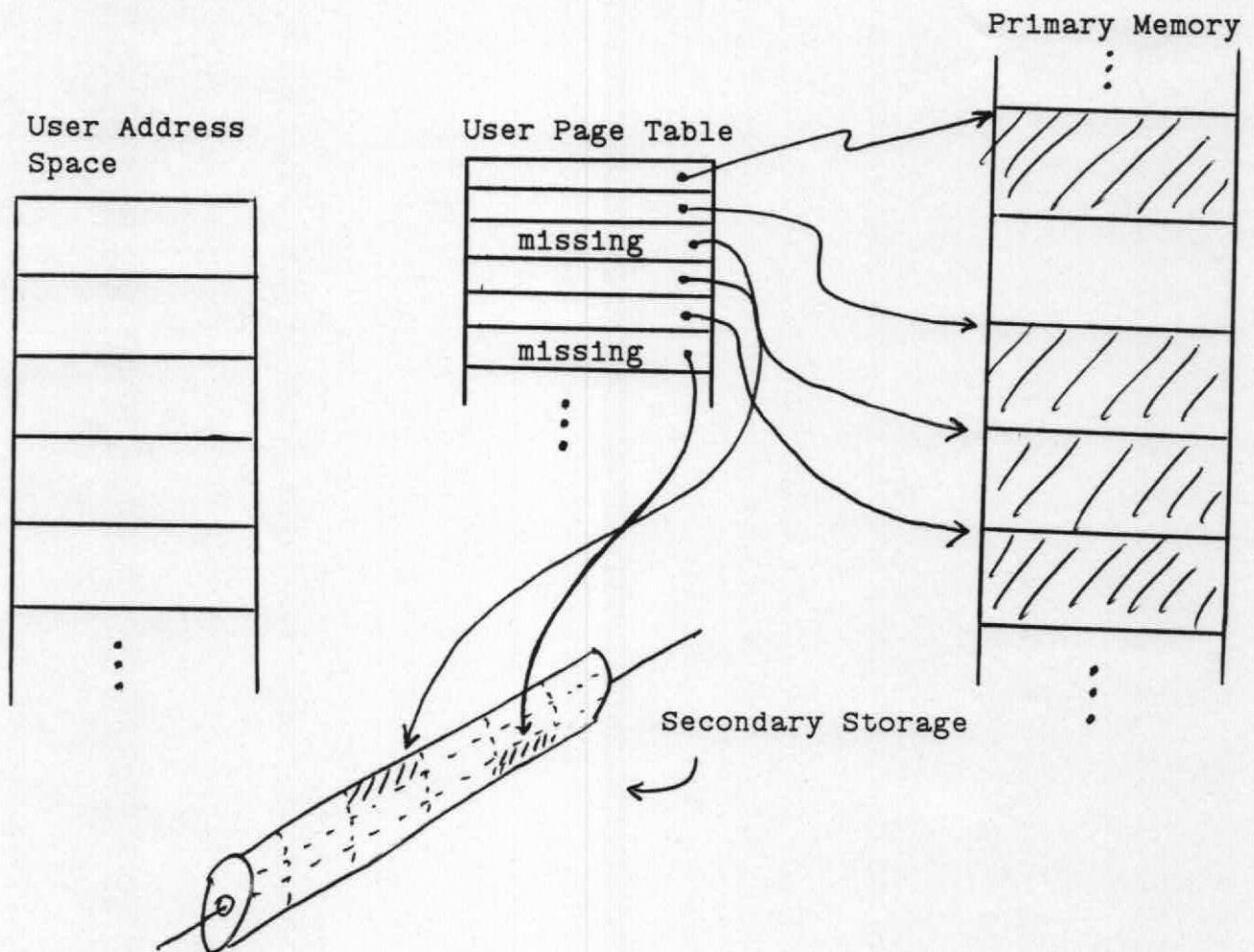
Idea is to bring in a page from secondary memory only when it is demanded by the processor. Why?

- Dynamic relocation, even with paged allocation, requires the *whole* program to be loaded in the memory to be executed.
- For effective multiprogramming (without demand paging) each user should use only a part of their processor address space (counter-productive for software development).
- Not all parts of a program are equally useful (*e.g.*, exception handling routines).

## Demand Paging

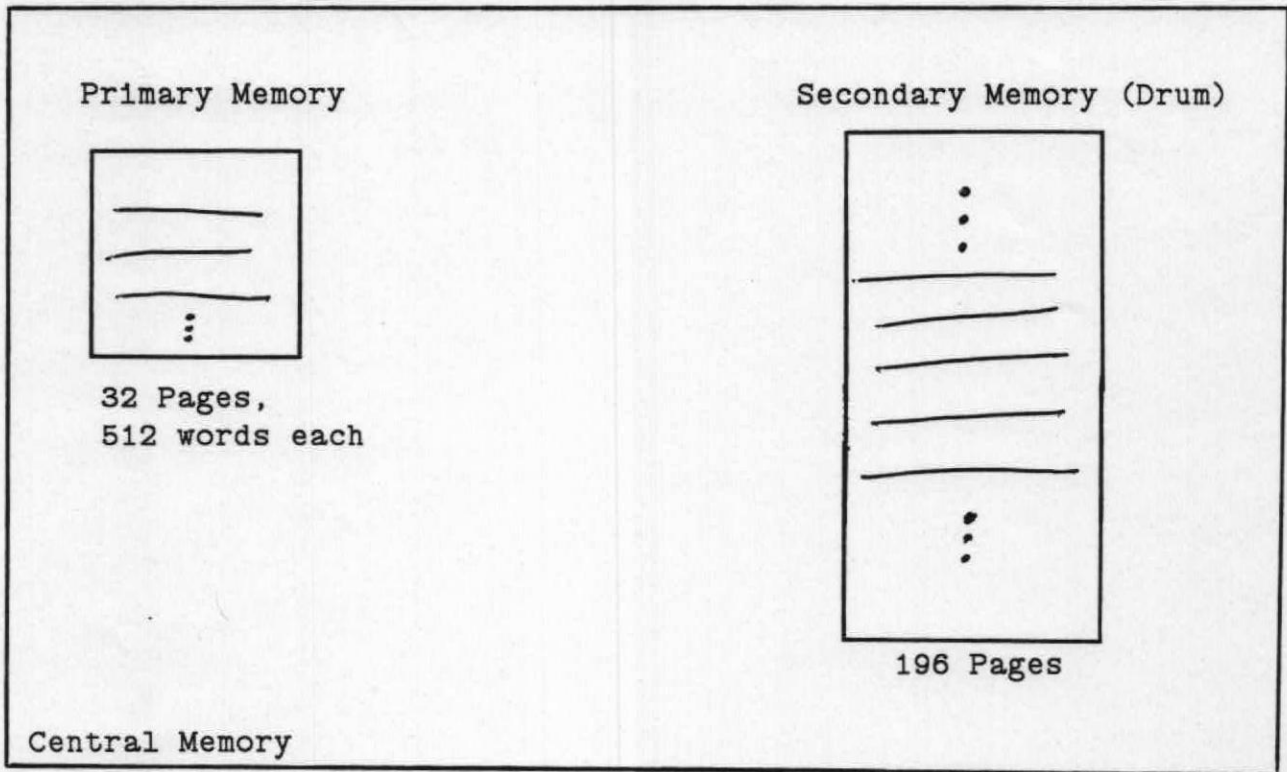
Bring a page into primary memory only when it is (implicitly) demanded by the processor.

- User is given a "virtual" address space - the illusion of a much larger address space than he really has.
- Primary memory is like a cache for the drum.



## Demand Paging

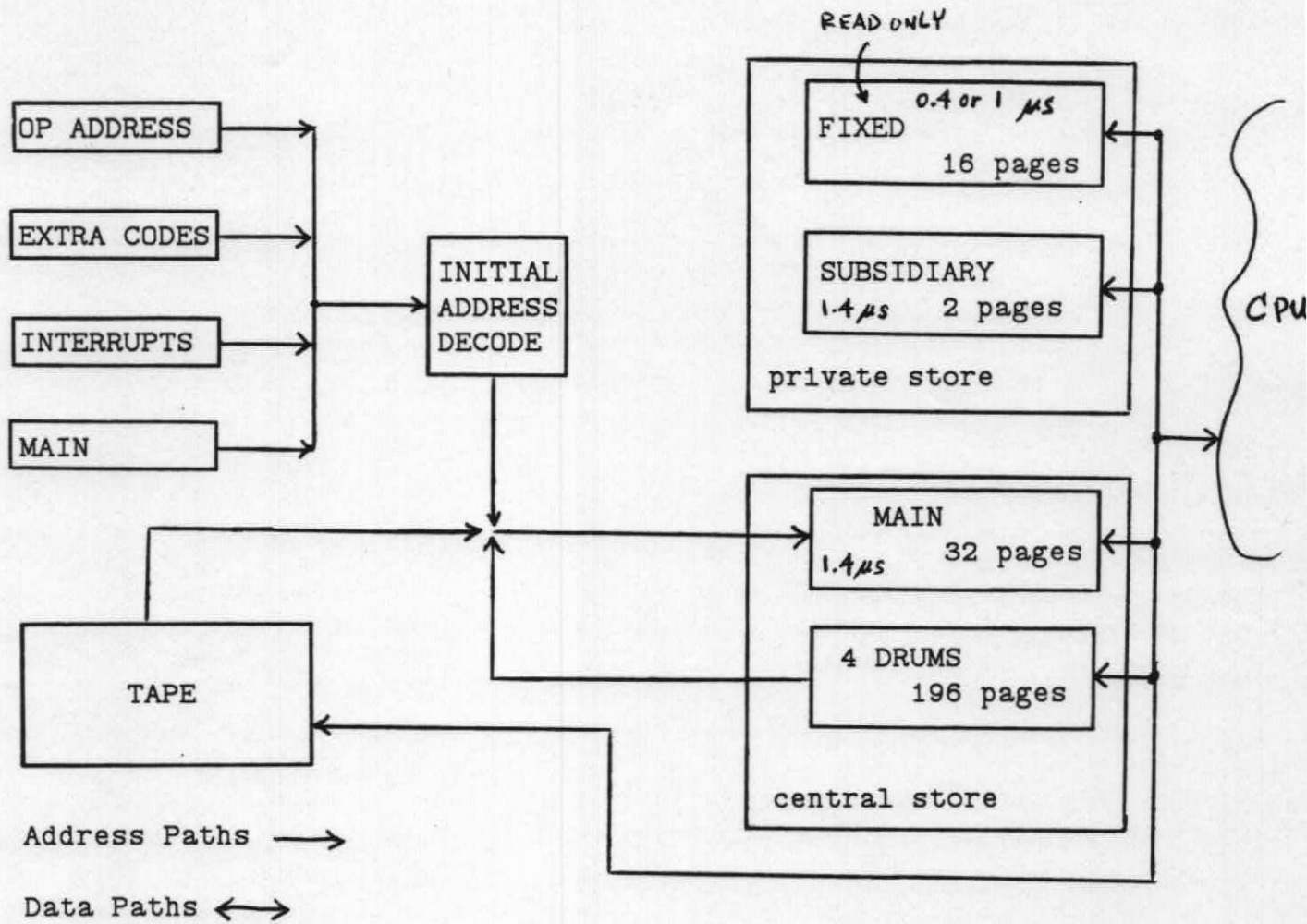
The ATLAS computer developed by Kilburn, et. al. in the early 1960s was one of the first machines to make use of demand paging.



User has the illusion of 196 pages in his address space even though at most 32 are in core memory at a time.



## Hardware Organization of ATLAS



- Pages in private store are not swappable.
- Subsidiary store provided the working space for programs resident in the fixed store.

## ATLAS Hardware

- 48-bit machine with 24-bit addresses.
  - (12-bit page address,  
9-bit word address,  
2-bit byte address)
- System code and mathematical subroutines are kept in fixed store (ROM) which is twice as fast as core memory.
- The private store (fixed and subsidiary) is not paged.
- Core Memory Speed =  $1.4 \mu\text{s}$ 
  - Drum Rotation Time = 14 ms
  - Page Transfer Time = 2 ms
- A **Page Address Register (PAR)** is associated with every page frame in main memory. A PAR contains:
  - Address of page occupying the frame
  - Lockout bit
  - Usage information

## Demand Paging in ATLAS

1. Page address of an operand is compared against all 32 PARS.
  - *Match*  $\Rightarrow$  Proceed with normal access.
  - *No Match*  $\Rightarrow$  Page fault; state of partially executed instruction is saved and page is retrieved from drum.
2. A free page frame is always maintained to initiate an input transfer upon demand.
3. If no more free page frames remain, a page in the main memory has to be selected and thrown out. (Usage learning program!)
4. To minimize drum latency effect, the first empty page on drum was selected.
5. A "directory" or "page table" is used to keep track of where a page resides on drum.

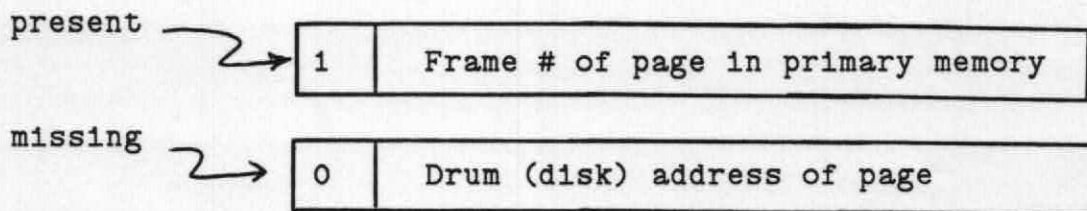
## Demand Paging in ATLAS

1. Support for a "present" bit in each page table entry to indicate whether a page is in primary or secondary memory.
2. Processor must recognize a "missing page trap" and save the state of each partially executed instruction while fault is processed.
3. Individual page usage must be recorded to assist page fault handler in implementing an effective page replacement policy.

### Software Support for Demand Paging

1. One page table is required for *each* process.

Page Table Entry:



... plus additional information in each entry.

2. A primary storage map (page frame table) for the whole system.

0:	process 1
1:	free
2:	process 2
3:	process 1
4:	O. S.
5:	process 3
	⋮



One entry for each page frame.

## Demand Paging

The most serious problem with demand paging is *thrashing* (excessive I/O).

The scenario:

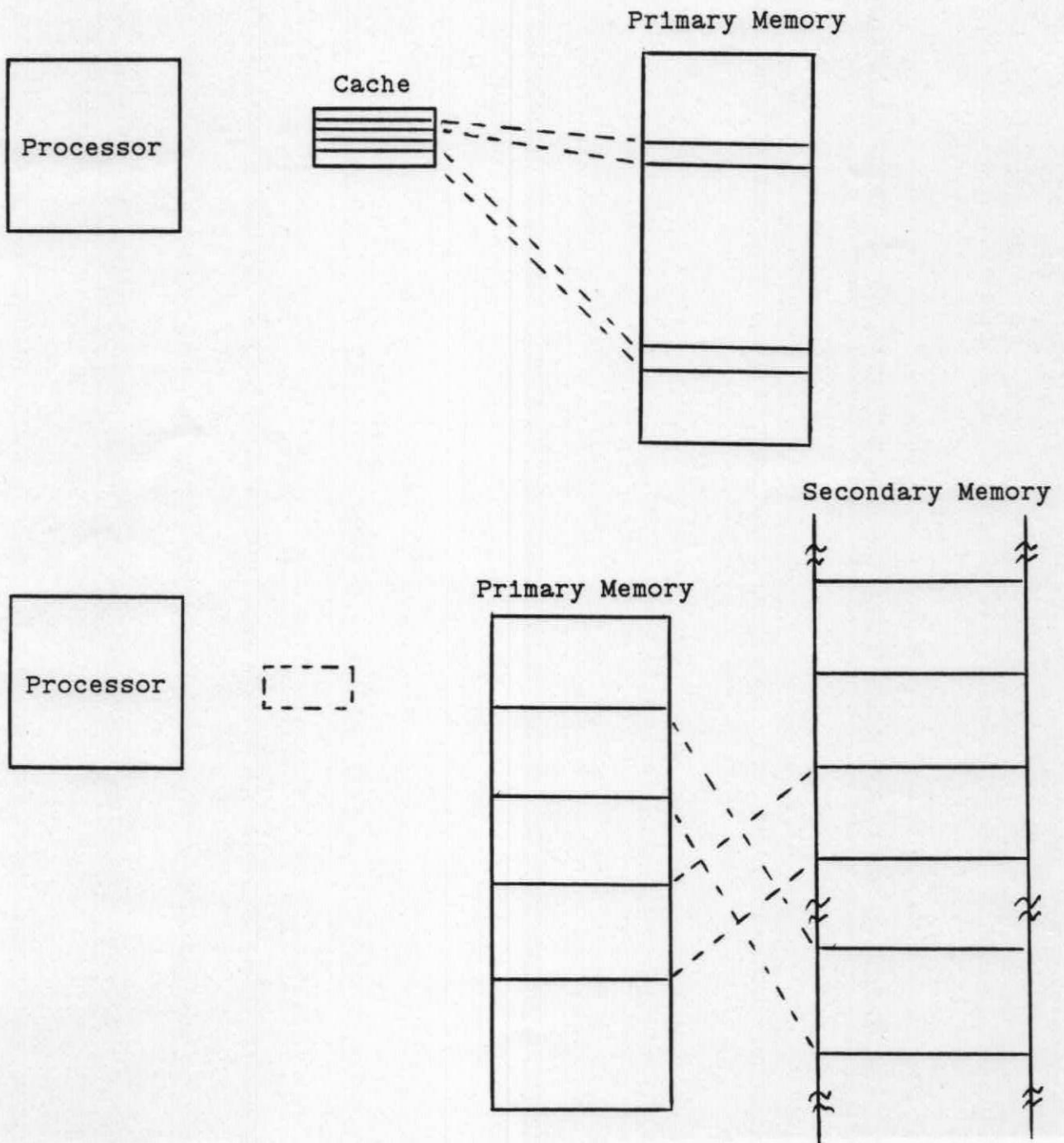
1. Page X is thrown out to make space for the demanded page Y.
2. After a few references, page X is needed again ...

**The solution:** Allocate more memory to the process.

- The "working set" of a process must be kept in the primary memory at all times.
- May have to reduce the number of concurrently resident processes to get enough pages for working sets.

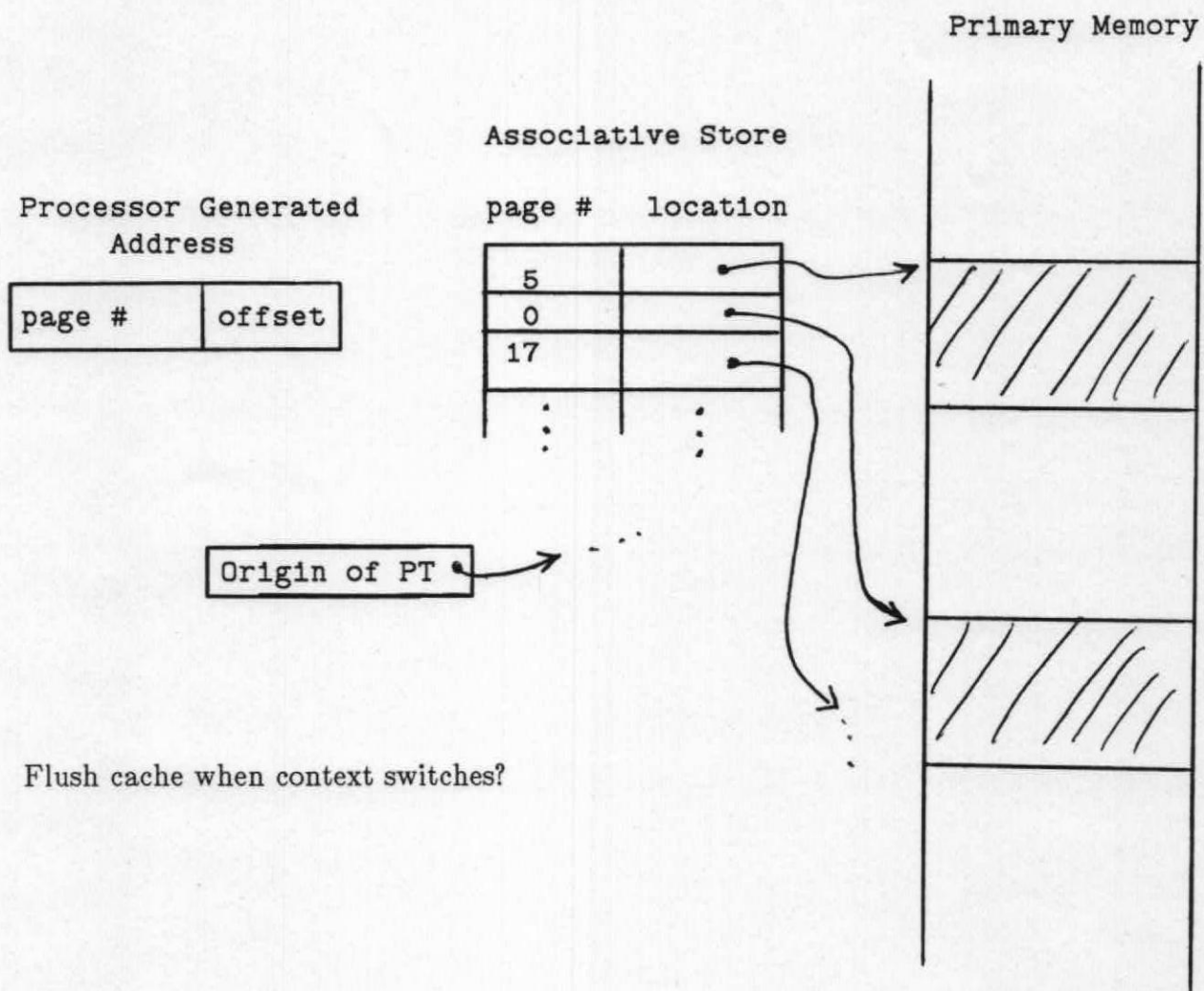
### Demand Paging

Primary memory acts like a cache for the secondary memory.



### Demand Paging

The use of a cache or associative store (TLB) to assist in address translation is essential to reduce the extra memory references required to access page table entries.



Flush cache when context switches?

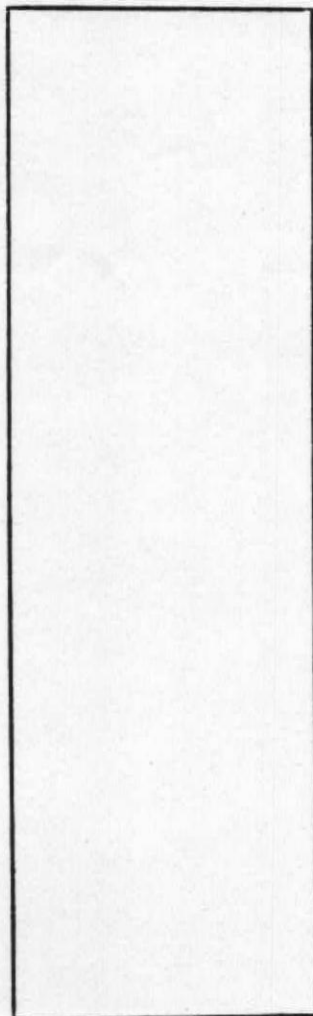


## Demand Paging

Demand paging has eliminated the concern about the amount of physical (primary) memory available to the processor.

- Makes software development much easier.
- Allow multiple users to share primary memory more effectively.

Virtual Address  
Space



Mapping



Physical Address  
Space

