



Memory Paging and Replacement

Lecture 31

Klara Nahrstedt

CS241 Administrative

- Read Stallings Chapter 8.1 and 8.2 about VM
- LMP2 (Part II due on Monday, April 16)
- Quiz 10 on Friday, 6/13 about Memory Paging – see Chapter 8.1 (address translation, protection, paging hardware, inverted page table), and Chapter 8.2 (Fetch, Replacement and Placement Policies)
 - See also lecture notes, self-assessment quiz, discussion section

Vinton Cerf Lecture, April 11, 11am, 1404 SC

Distinguished Lecture Series in Computer Science

Wednesday, April 11

11:00am, 1404 Siebel

Dr. Vinton Cerf, Chief Internet Evangelist and VP at Google

Dr. Cerf is widely known as one of the founders of the Internet and as the co-designer of the TCP/IP protocol, the communications protocol that gave birth to the Internet and that is commonly used today.

This talk will be more technical in nature than the **Arnold O. Beckman Lecture** that Dr. Cerf will deliver on **Tuesday, April 10th, at 4pm in Foellinger**. Computer Science faculty and students are encouraged to attend both talks.

Concepts this Lecture

Two-Level Paging Example

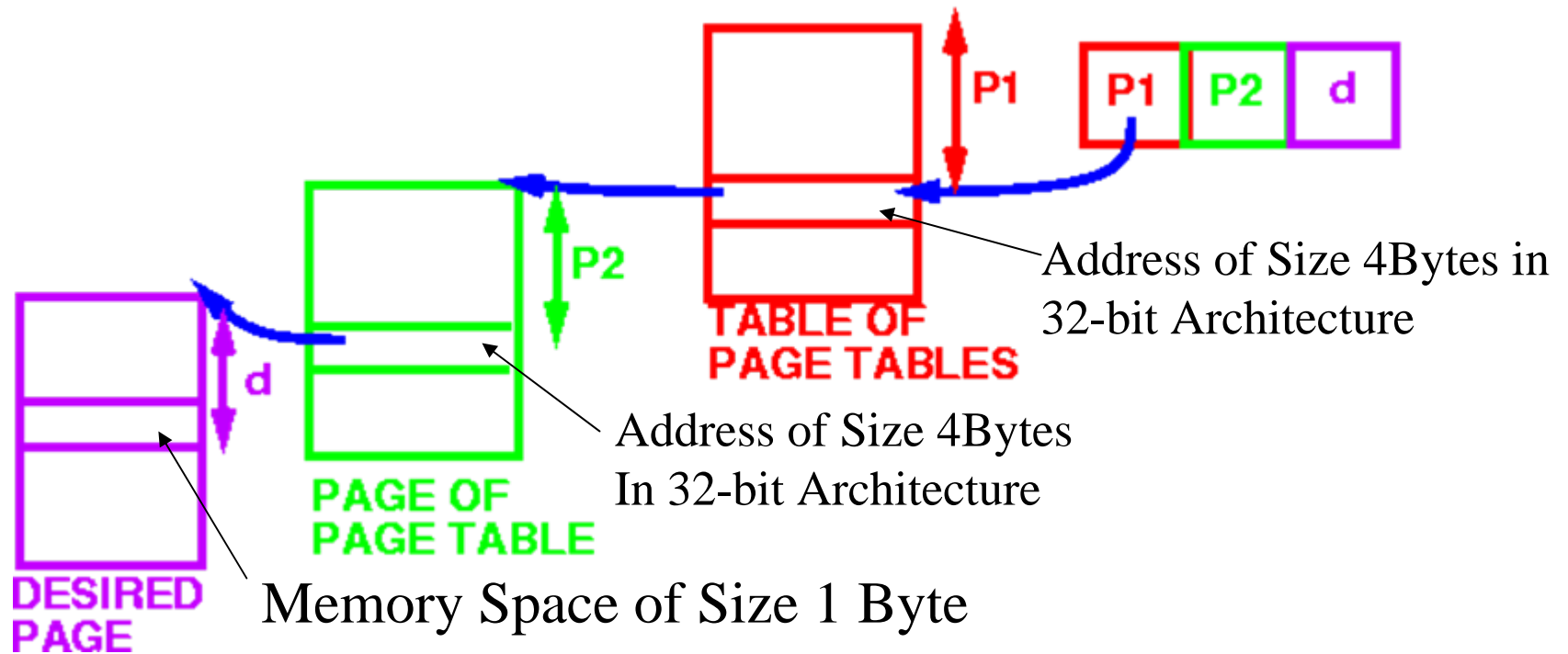
Inverted Page Table

Sharing and Protection

Introduction to Demand Paging

Multilevel Paging

LOGICAL ADDRESS



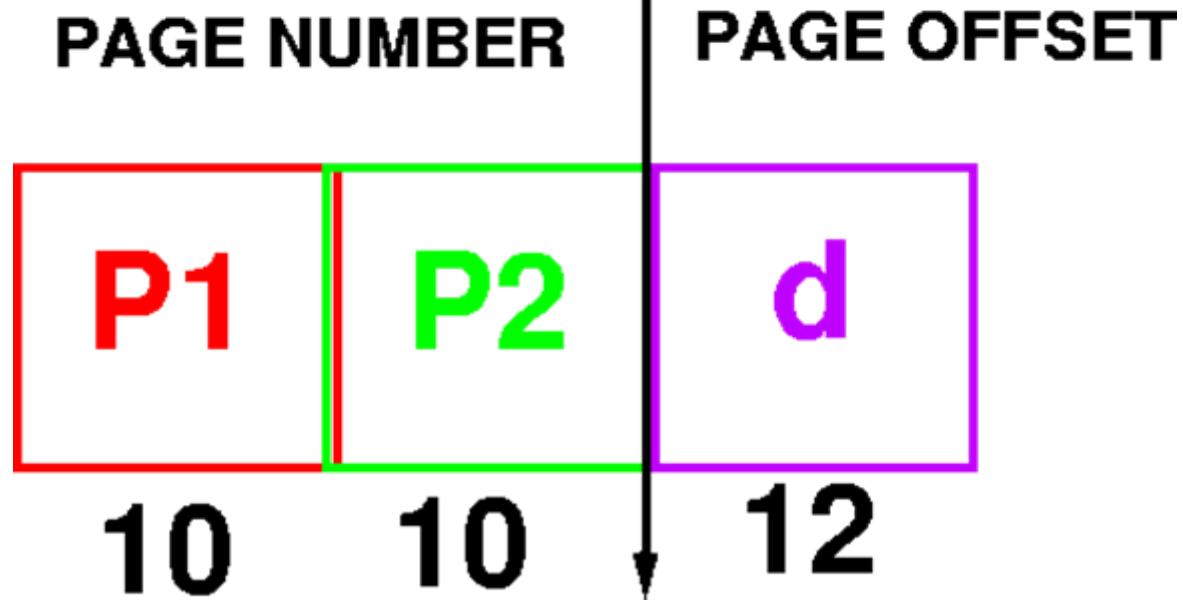
Addressing on Two-Level Page Table

32-bit Architecture, $4096 = 2^{12}$ Bytes Page Size

4K Page of Logical Memory has 4096 addressable bytes

Page the Page Table with 4K pages as well

4K Page of Page Table has 1024 addressable 4byte addresses



Multilevel Paging and Performance

Since each level is stored as a separate table in memory, converting a logical address to a physical one in a two level paging system may take up to four memory accesses.

Two Level Page Table Caching

Check cache for base address of 1st level cache.
Fetch as necessary (1 ref).

Check cache for address of 2nd level page base
address in 1st level page table. Fetch as
necessary (1 ref).

Two Level Page Table Caching

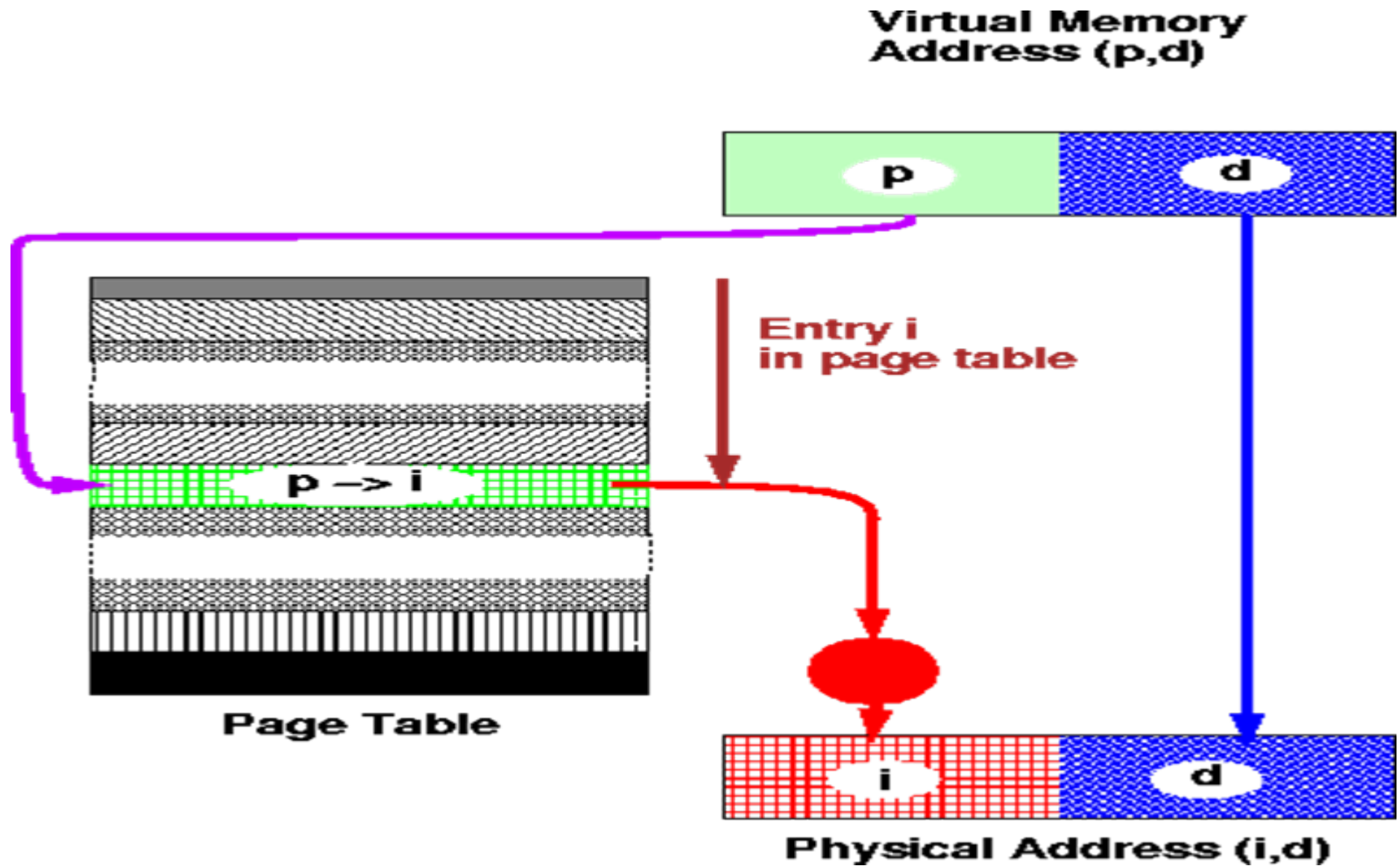
Check cache for address of 2nd level page table entry. Fetch as necessary (1 ref).

Check cache for address of page stored in 2nd level page table. Fetch as necessary (1 ref).

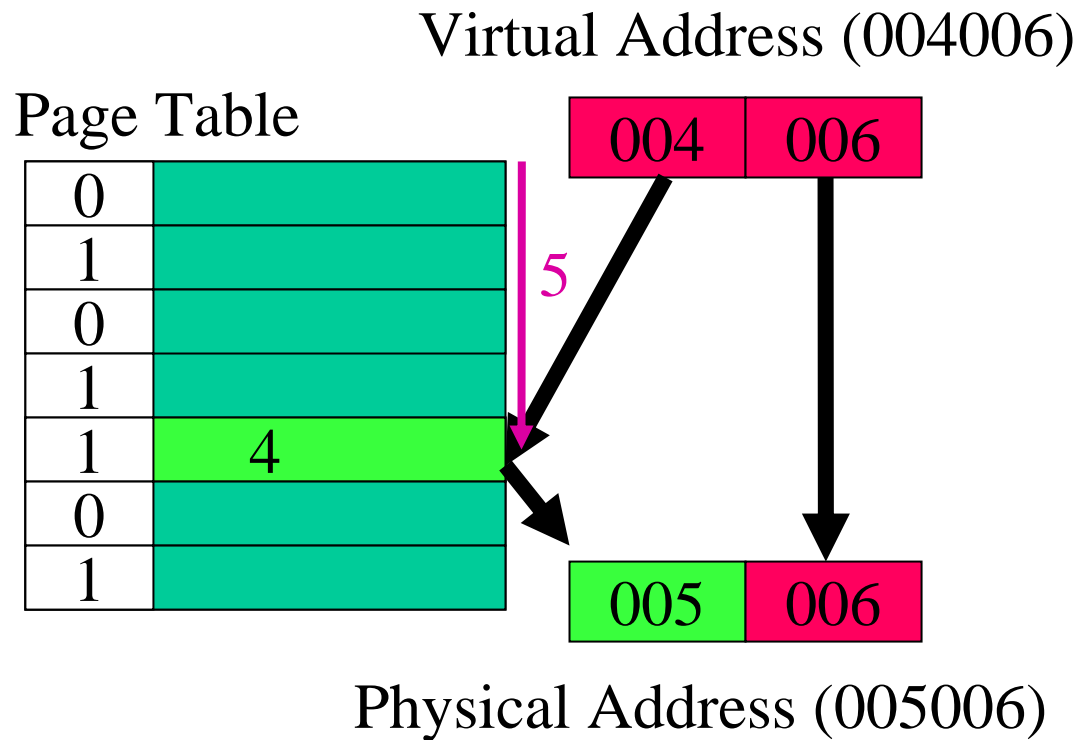
Multilevel Paging and Performance

Since each level is stored as a separate table in memory, converting a logical address to a physical one with a three-level page table may take four memory accesses. Why?

Inverted Page Table



Inverted Page Table



Inverted Page Table Implementation

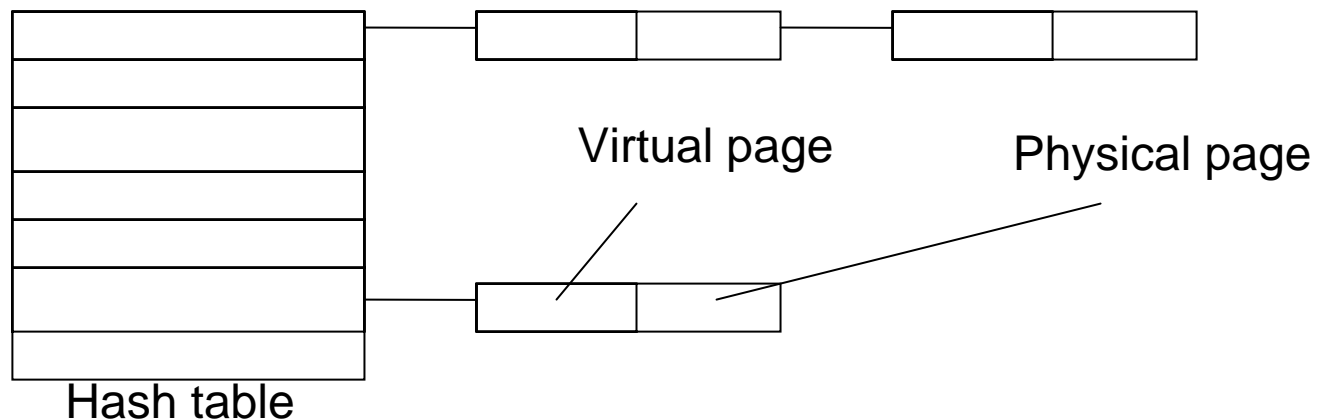
TLB is same as before

TLB miss is handled by software

In-memory page table is managed using a hash table

Number of entries \geq number of physical frames

Not found: page fault



Inverted Page Table

one entry for each real page of memory.

entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page.

Inverted Page Table

Decreases memory needed to store each page table, but increases time needed to search table when a page reference occurs.

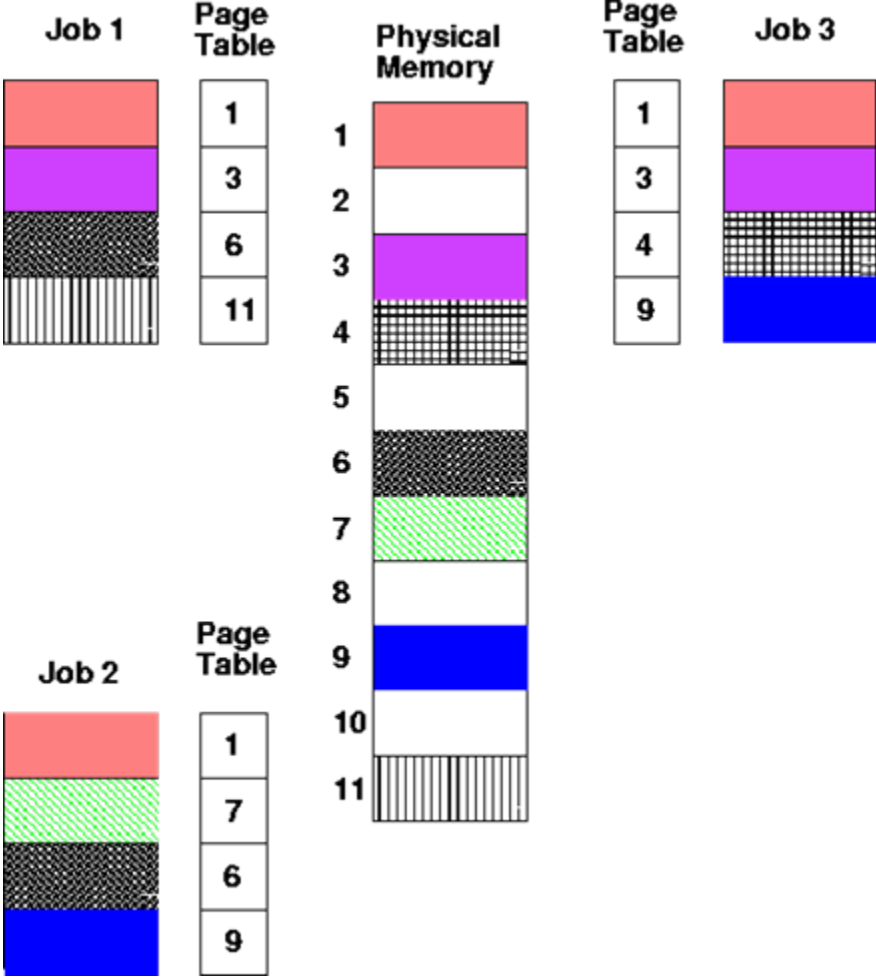
Use hash table to limit the search to one -- or at most a few page-table entries.

Sharing Pages

Code and data can be shared by mapping them into pages with common page frame mappings.

Code and data must be position independent if VM mappings for the shared data are different.

Shared Pages



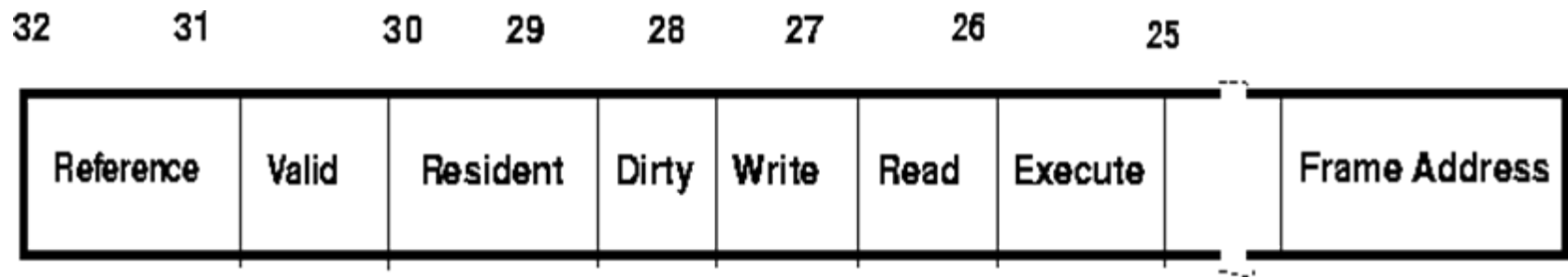
Protection

Can add read, write, execute protection bits to page table to protect memory.

Check is done by hardware during access.

Can give shared memory location different protections from different processes by having different page table protection access bits.

Page Protection



Legend:

reference - page has been accessed

valid - page exists

resident - page is cached in primary memory

dirty - page has been changed since page in

Introduction to Demand Paging - Paging Policies

Fetch Strategies

When should a page be brought into primary (main) memory from secondary (disk) storage.

Placement Strategies

When a page is brought into primary storage, where is it to be put?

Replacement Strategies

Which page now in primary storage is to be removed from primary storage when some other page or segment is to be brought in and there is not enough room.

Demand Paging

Algorithm

Never bring a page into primary memory until its needed.

Page fault

Check if a valid virtual memory address. Kill job if not.

If valid reference, check if its cached in memory already (perhaps for some other process.) If so, skip to 7).

Find a free page frame.

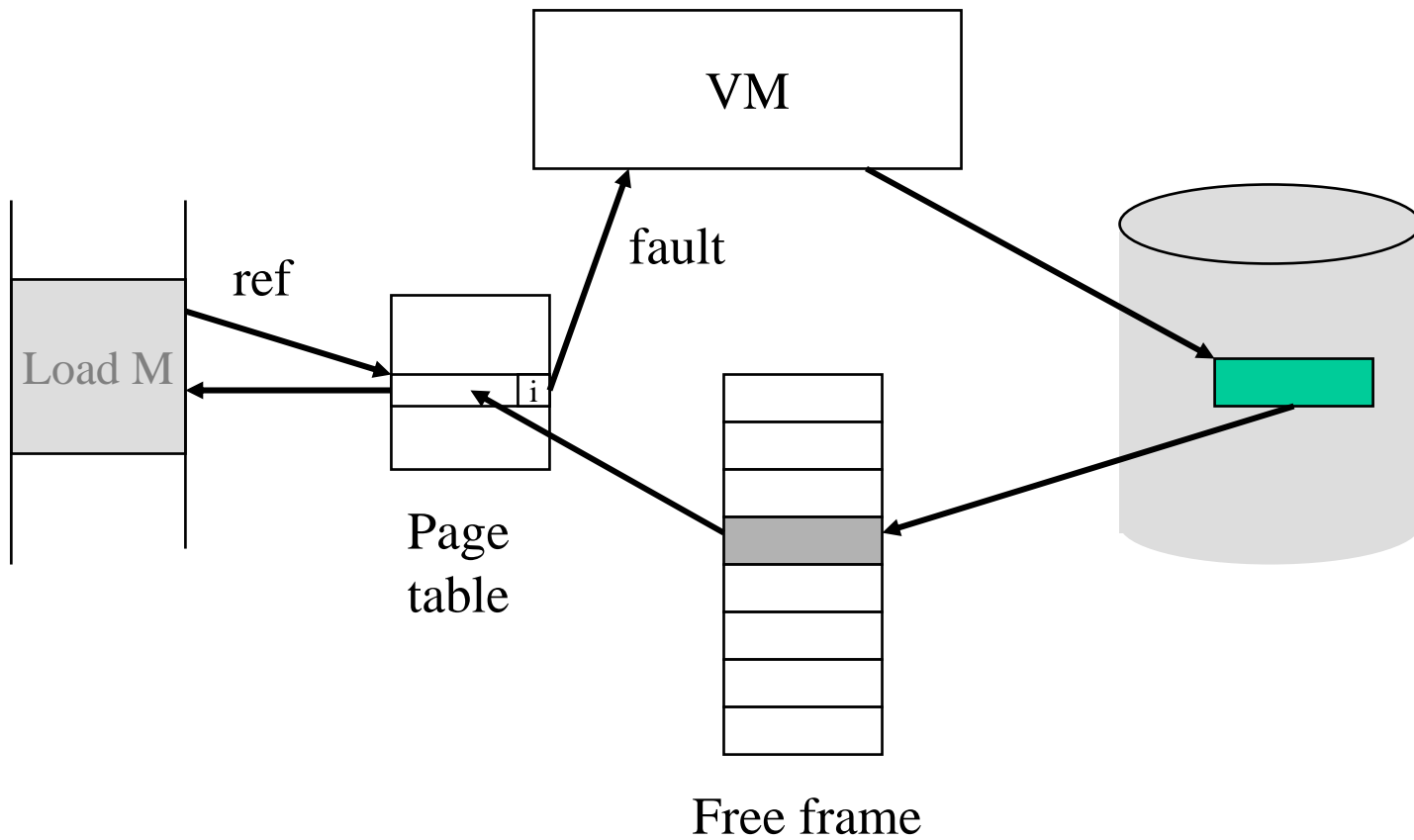
Map address into disk block and fetch disk block into page frame.

Suspend user process.

When disk read finished, add vm mapping for page frame.

If necessary, restart process.

Demand Paging Example



Page Replacement

Find location of page on disk

Find a free page frame

- If free page frame use it

- Otherwise, select a page frame using the page replacement algorithm

- Write the selected page to the disk and update any necessary tables

Read the requested page from the disk.

Restart the user process.

It is necessary to be careful of synchronization problems.

- For example, page faults may occur for pages being paged out.

Issue: Eviction

Hopefully, kick out a less-useful page

Dirty pages require writing, clean pages don't

Hardware has a dirty bit for each page frame indicating this page has been updated or not

Where do you write? To "swap space"

Goal: kick out the page that's least useful

Problem: how do you determine utility?

Heuristic: temporal locality exists

Kick out pages that aren't likely to be used again

Terminology

Reference string: the memory reference sequence generated by a program.

Paging – moving pages to (from) disk

Optimal – the best (theoretical) strategy

Eviction – throwing something out

Pollution – bringing in useless pages/lines

Page Replacement Strategies

The Principle of Optimality

Replace the page that will not be used again the farthest time in the future.

Random page replacement

Choose a page randomly

FIFO - First in First Out

Replace the page that has been in primary memory the longest

LRU - Least Recently Used

Replace the page that has not been used for the longest time

LFU - Least Frequently Used

Replace the page that is used least often

NRU - Not Recently Used

An approximation to LRU.

Working Set

Keep in memory those pages that the process is actively using.

Principal of Optimality

Description:

Assume that each page can be labeled with the number of instructions that will be executed before that page is first references, i.e., we would know the future reference string for a program.

Then the optimal page algorithm would choose the page with the highest label to be removed from the memory.

This algorithm provides a basis for comparison with other schemes.

Impractical because it needs future references

If future references are known

should not use demand paging

should use pre paging to allow paging to be overlapped with computation.

Summary

Two Level Paging

Inverted Page Table

Sharing/Protection

Introduction to Demand Paging

Fetch Policies

Replacement Policies

Principle of Optimality