

Discussion 5

Jinlang Wang

Logistics

To be covered today:

- 1) Virtual Memory
- 2) Page Replacement Algorithms
- 3) Paging & Page Tables
- 4) Memory Allocation Algorithms

Memory Partitions

Remember from Last Time...

The ideal world has memory that is...

- 1) Copious in Supply
- 2) Very Fast
- 3) Non-volatile

But the reality is that we can only pick two of the following:

- 1) Copious Memory
- 2) Fast Memory
- 3) Affordable Memory

Goal: Get the **real world** as close as possible to the **ideal world**!

Fixed Partitions

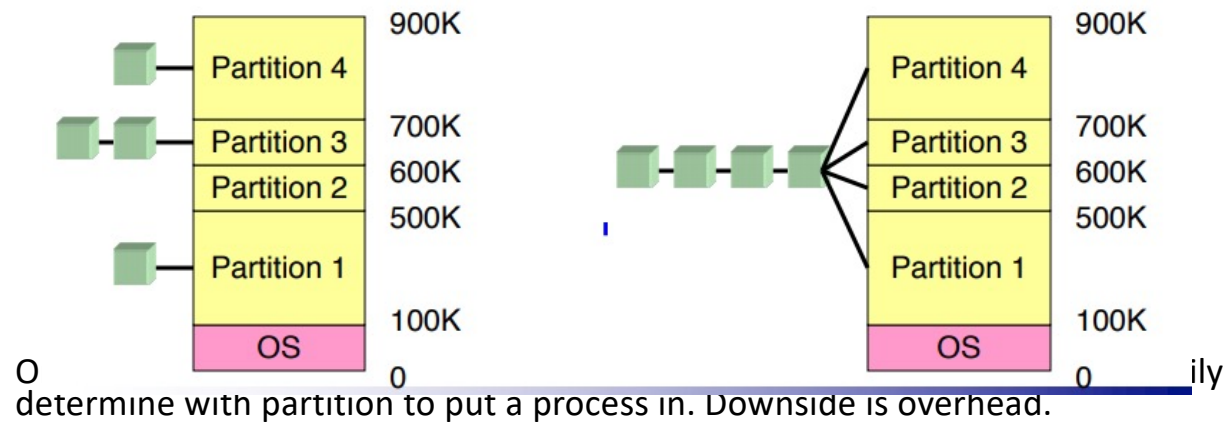
Idea:

1. Divide memory into fixed spaces
2. Each process will be assigned a fixed space when it's free

Mechanisms:

1. Separate input queues for each partition
2. Single input queue: has better ability to optimize CPU usage

What's the Difference Here?



On the left side, if we have a single input queue, we will use less memory, but it will take longer to identify which partition to assign a process to.

Memory Allocation

Base and Limit Registers

Special CPU registers: base & limit

1. Access to the registers limited to system mode

2. Registers contain:

- Base: start of the process's memory partition
- Limit: length of the process's memory partition

3. Address generation

- Physical address: location in actual memory
- Logical address: location from the process's point of view
- Physical address = base + logical address
- Logical address larger than limit => error

An Example

First, what does this diagram represent?

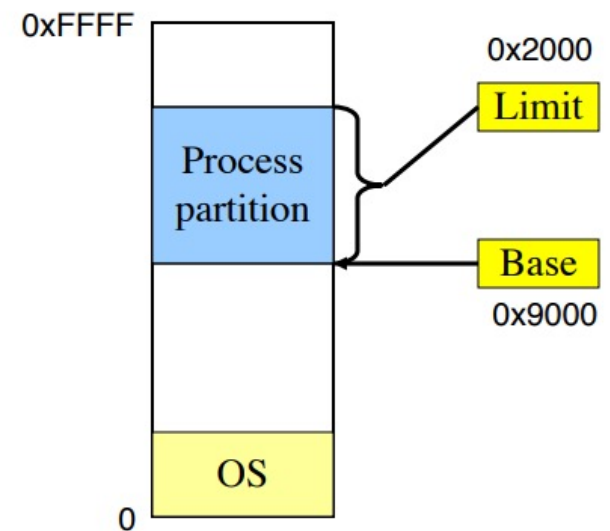
1. The processes that are allocated on the OS!

If we have a logical address of 0x1204, we can

Calculate the Physical Address like so:

Physical address = base + logical address

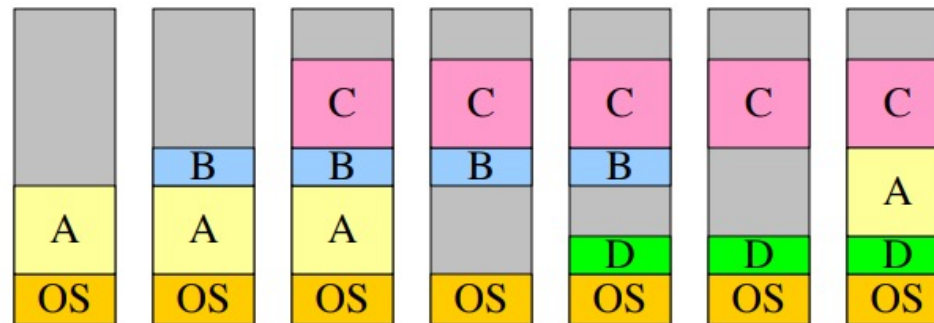
- $0x1204 + 0x9000 = 0xa204$



Swapping

Memory allocation changes as:

1. Processes come into memory
2. Processes leave memory



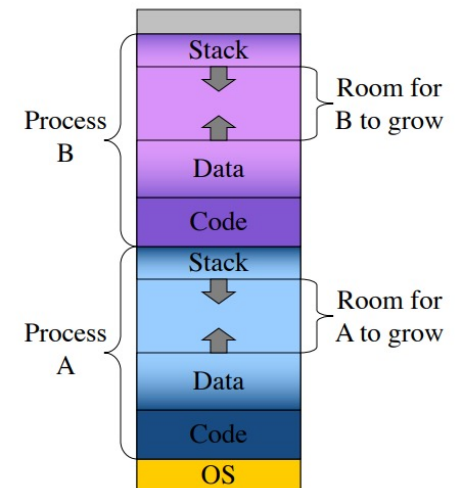
Swapping

We need to allow for programs to grow.

- Allocate more memory for data
- Meaning... a larger stack

This is handled by allocating more space than necessary at the start...

- But this is **inefficient**: it will waste memory that's not currently in use.



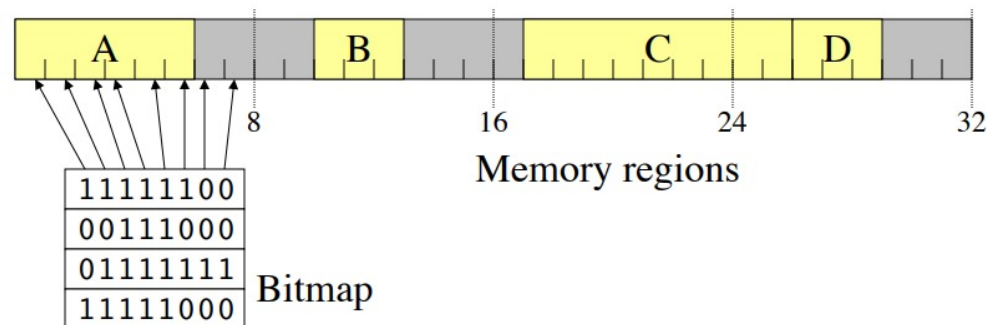
Tracking memory Usage: BitMaps

Keep track of free / allocated memory regions with a bitmap

- One bit in map corresponds to a fixed-size region of memory
- Bitmap is a constant size for a given amount of memory regardless of how much is allocated at a particular time

Chunk size determines efficiency

1. At 1 bit per 4KB chunk, we need just 256 bits (32 bytes) per MB of memory
2. For smaller chunks, we need more memory for the bitmap
3. Can be difficult to find large contiguous free areas in bitmap



Tracking memory Usage: Linked Lists

Keep track of free / allocated memory regions with a linked list

1. Each entry in the list corresponds to a contiguous region of memory
2. Entry can indicate either allocated or free (and, optionally, owning process)
3. May have separate lists for free and allocated areas

Efficient if chunks are large

1. Fixed-size representation for each region
2. More regions => more space needed for free lists

Allocating Memory

Search through region list to find a large enough space

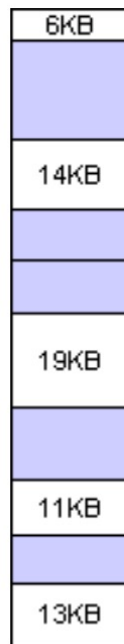
Suppose there are several choices: which one to use?

1. First fit: the first suitable hole on the list
2. Next fit: the first suitable after the previously allocated hole
3. Best fit: the smallest hole that is larger than the desired region
(wastes least space?)
4. Worst fit: the largest available hole (leaves largest fragment)

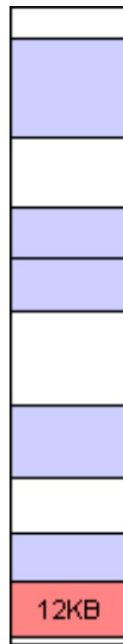
Option: maintain separate queues for different-size holes

An Example

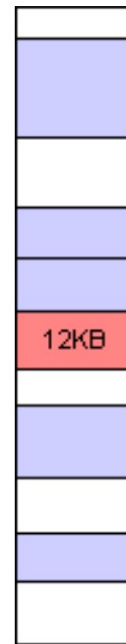
An Example



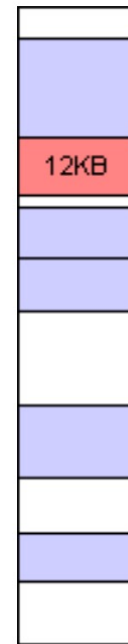
Primary
Memory



Best fit



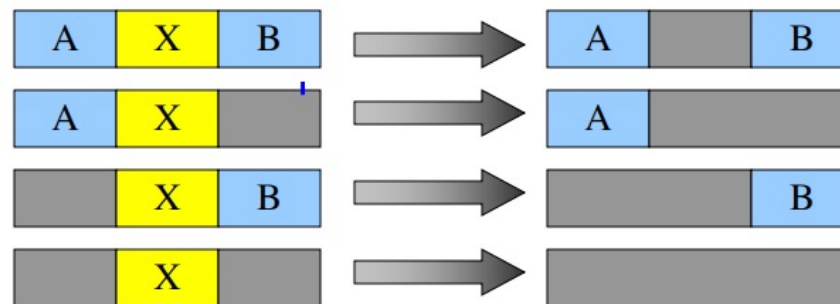
Worst fit



First fit

Freeing memory

1. Allocation structures must be updated when memory is freed
2. Easy with bitmaps: just set the appropriate bits in the bitmap
3. Linked lists: modify adjacent elements as needed
 - Merge adjacent free regions into a single region
 - May involve merging two regions with the just-freed area



Problems with Swapping

1. Process must fit into physical memory (impossible to run larger processes)
2. Memory becomes fragmented
 - External fragmentation: lots of small free areas
 - Compaction needed to reassemble larger free areas
3. Processes are either in memory or on disk: half and half doesn't do any good

Virtual Memory

The motivation

Physical memory is a **scarce resource**!

Basic Idea: We want to allow the OS to hand out more memory than exists on the system!

- Keep recently used stuff in **Physical Memory**
- Move least recently used stuff to **Disk**
- But... keep this information hidden from processes...
 - Processes will still see an address space from 0 – **max address**
 - The addresses each process accesses is **not the real address sent to memory**
 - Movement of information to and from disk is handled by the **OS**

Virtual and Physical Addresses

Programs use **virtual addresses**

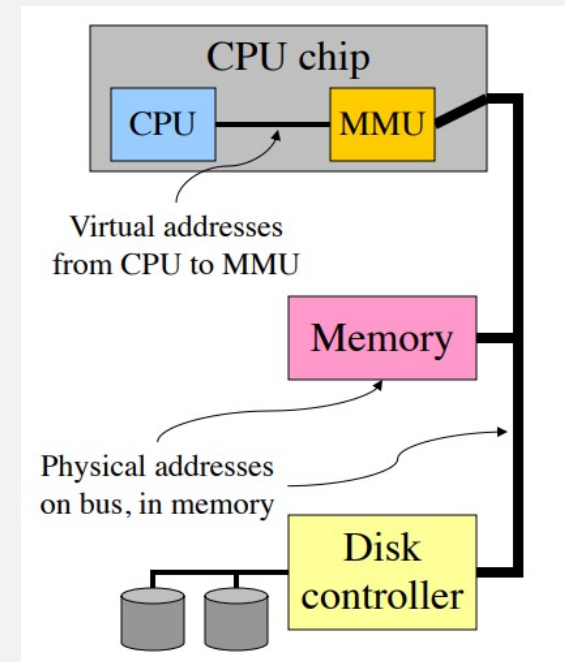
- Addresses local to the process
- Hardware translates **virtual addresses** to **physical addresses**

Enter the Memory Management Unit (MMU)

The translation from **virtual address** to **physical address** is done by the **Memory Management Unit**

- It is typically on the same chip as the CPU
- Only physical addresses leave the CPU/MMU chip

Physical memory is indexed by physical addresses



Paging and Page Tables

Paging and Page Tables

Virtual addresses are mapped to physical addresses

- A unit of mapping is called a **page** (Typically 4KiB)
- All addresses in the same virtual page are in the same physical page
- A **Page Table Entry** (PTE) contains translation for a single page.

Page Table translates virtual page number to a physical page number

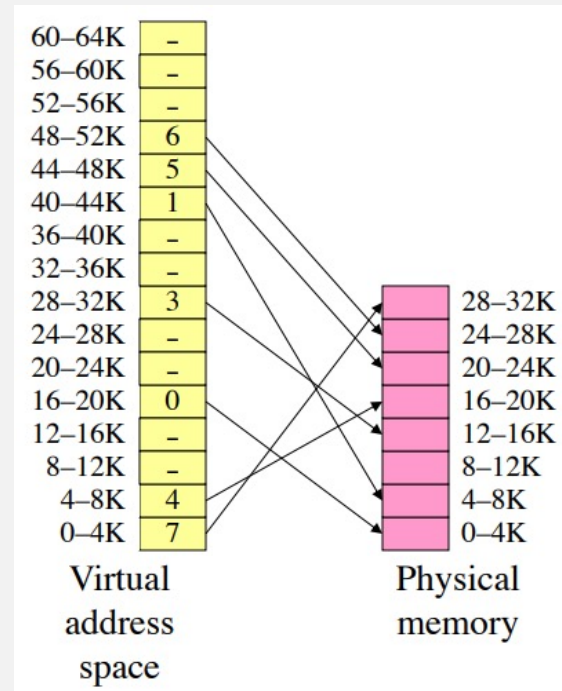
- Not all virtual memory has a physical page
- Not every physical page needs to be used

Example

Consider a 64KB virtual address space and a 32 KB physical memory:

- Where would a Page whose lives at 17 KB in the virtual address space map to in Physical memory?

This looks like a prime candidate for a certain data structure... can anyone name it?



What's in a Page Table Entry?

Each entry in the page table contains:

1) **Valid bit**: set if this logical page number has a corresponding physical frame in memory

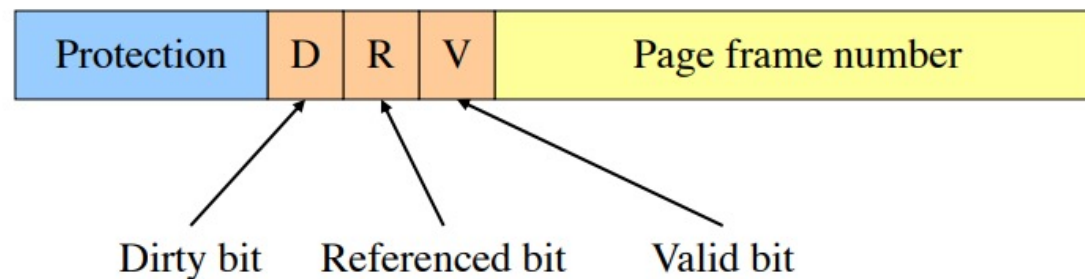
- If not valid, remainder of PTE is irrelevant

2) **Page frame number**: page in physical memory

3) **Referenced bit**: set if data on the page has been accessed

4) **Dirty (modified) bit**: set if data on the page has been modified

5) **Protection information**



Mapping Logical to Physical Addresses

Split address from CPU into two pieces:

- 1) Page number (p)
- 2) Page offset (d)

Page Number:

- 1) Index into the page table
- 2) Page table contains the base address of page in physical memory

Page Offset

- 1) Added to base address to get actual physical memory address

Page Size = 2^d

Example

4KB (4096 B) pages

32 bit logical addresses

$$2^d = 4096$$

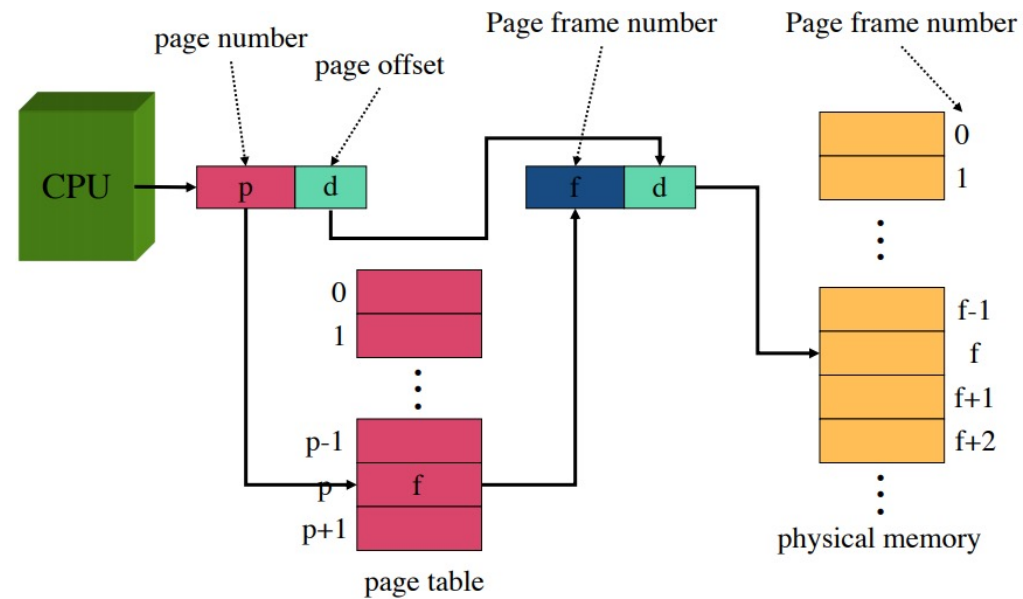
$$d = 12 \text{ bits}$$

$$32 - 12 = 20 \text{ bits}$$

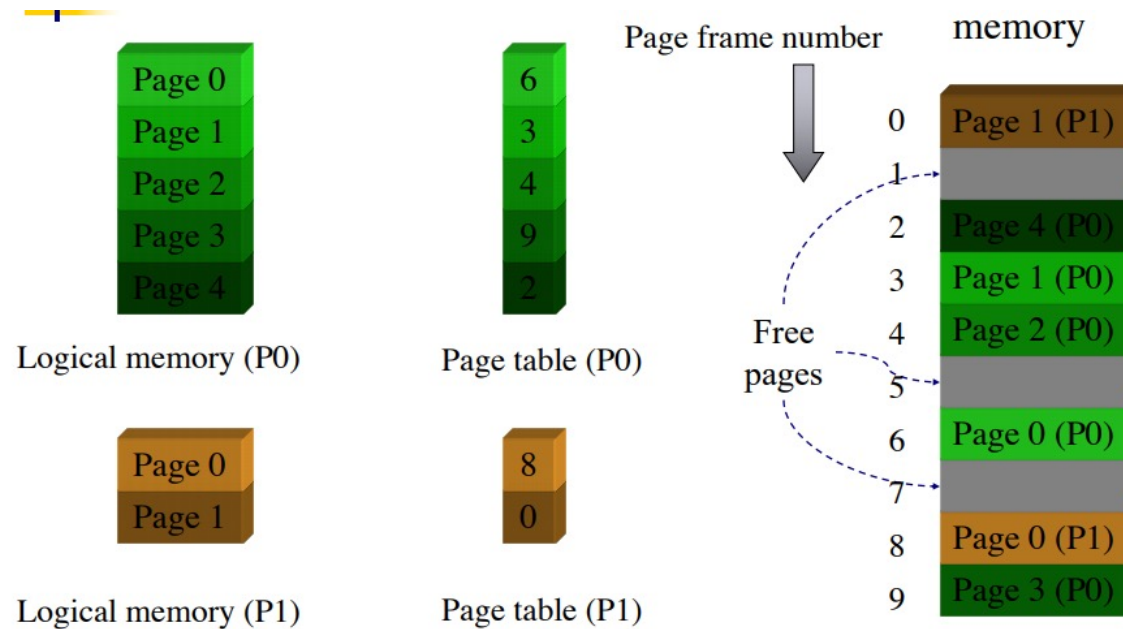


Address Translation Architectures & Paging Structures

Address Translation Architecture



Memory and Paging Structures



Two-Level Page Tables

Problem: page tables can be too large

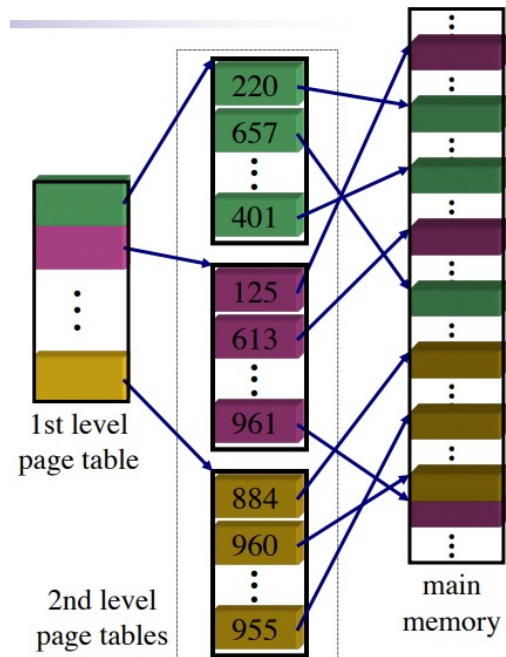
- 2^{32} bytes in 4KB pages need 1 million PTEs

Solution: use multi-level page tables

- “Page size” in first page table is large (megabytes)
 - PTE marked invalid in first page table needs no 2nd level page table
- 1st level page table has pointers to 2nd level page tables

2nd level page table has actual physical page numbers in it

The Model



More on Two-Level Page Tables

1) Tradeoffs between 1st and 2nd level page table sizes

- Total number of bits indexing 1st and 2nd level table is constant for a given page size and logical address length
- Tradeoff between number of bits indexing 1st and number indexing 2nd level tables
 - More bits in 1st level: fine granularity at 2nd level
 - Fewer bits in 1st level: maybe less wasted space?

2) All addresses in table are physical addresses

3) Protection bits kept in 2nd level table

Two-Level Paging Example

System characteristics:

- 8 KB pages
- 32-bit logical address divided into: 13 bit page offset, 19 bit page number

Page number divided into:

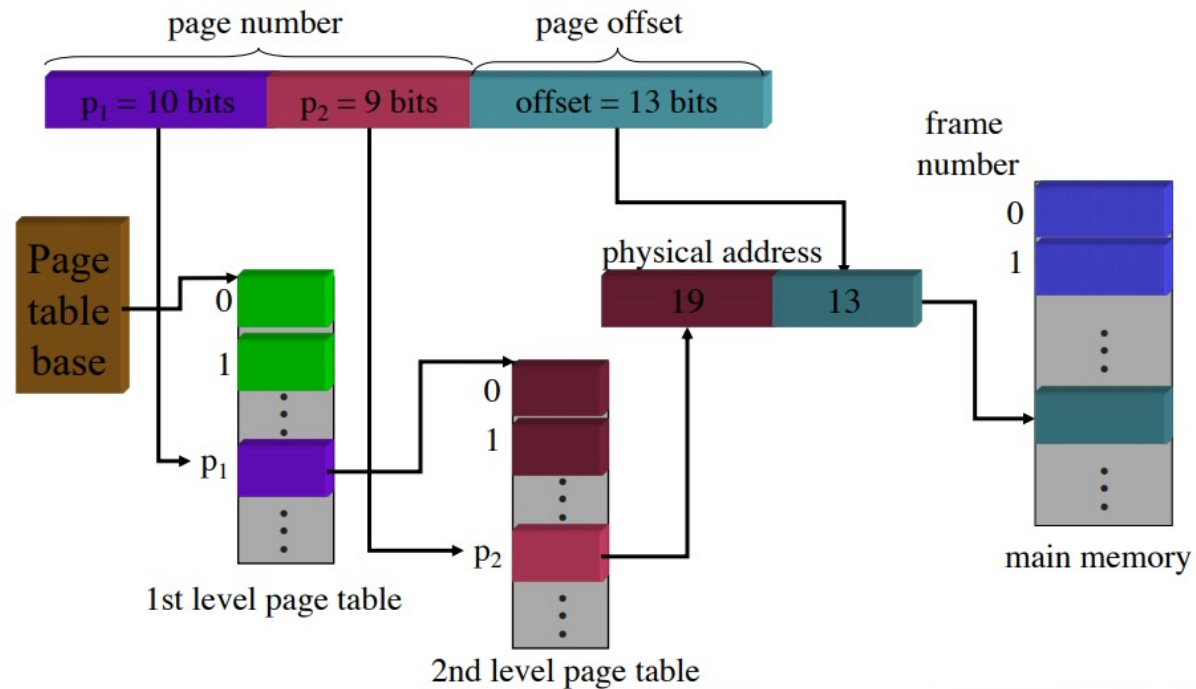
- 10 bit page number
- 9 bit page offset



Logical address looks like this:

- p_1 is an index into the 1st level page table
- p_2 is an index into the 2nd level page table pointed to by p_1

Two-Level Address Translation Example



Implementing Page Tables in Hardware

1) Page table resides in main (physical) memory

2) CPU uses special registers for paging

- Page table base register (PTBR) points to the page table
- Page table length register (PTLR) contains length of page table: restricts maximum legal logical address

3) Translating an address requires two memory accesses

First access reads page table entry (PTE)

Second access reads the data / instruction from memory

4) Reduce number of memory accesses

- Can't avoid second access (we need the value from memory)
- Eliminate first access by keeping a hardware cache (called a **translation lookaside buffer or TLB**) of recently used page table entries

Translation Lookaside Buffer (TLB)

Enter TLB... (Sounds like DLB :P)

1) Search the TLB for the desired logical page number (Sounds cachey...)

- Search entries in parallel
- Use standard cache techniques

2) If desired logical page number is found, get frame number from TLB

3) If desired logical page number isn't found

- Get frame number from page table in memory
- Replace an entry in the TLB with the logical & physical page numbers from this reference

Logical page #	Physical frame #
8	3
unused	
2	1
3	0
12	12
29	6
22	11
7	4

Example TLB

Handling TLB Misses

1) If PTE isn't found in TLB, OS needs to do the lookup in the page table

2) Lookup can be done in hardware or software

3) Hardware TLB replacement

- CPU hardware does page table lookup
- Can be faster than software
- Less flexible than software, and more complex hardware

4) Software TLB replacement

- OS gets TLB exception
- Exception handler does page table lookup & places the result into the TLB
- Program continues after return from exception
- Larger TLB (lower miss rate) can make this feasible

Inverted Page Tables

Enter Inverted Page Tables

1) Reduce page table size further: keep one entry for each frame in memory

2) PTE contains:

- Virtual address pointing to this frame
- Information about the process that owns this page

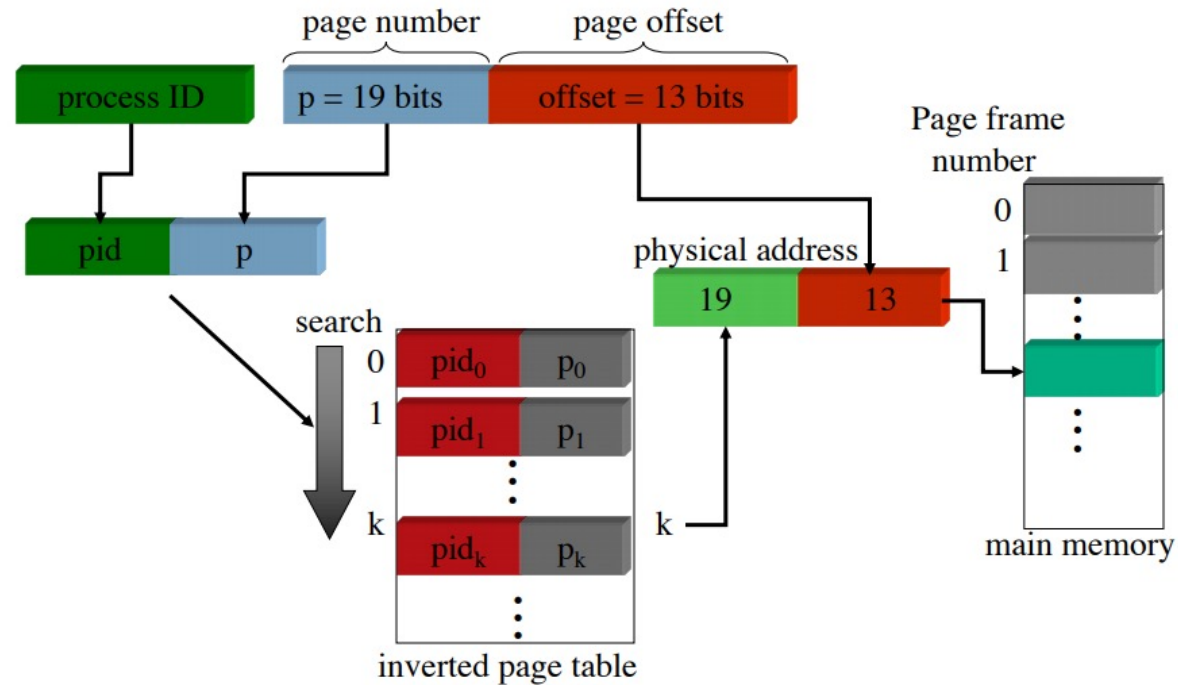
3) Search page table by:

- Hashing the virtual page number and process ID
- Starting at the entry corresponding to the hash result
- Search until either the entry is found or a limit is reached

4) Page frame number is index of PTE

5) Improve performance by using more advanced hashing algorithms

Inverted Page Table Architecture



Page Replacement Algorithms

Motivation for Page Replacement Algorithms

1) Page Fault forces a choice

- What is a Page Fault? We don't have room in our Page Table for another page!!
- We don't have room for a new page (steady state)
- Which page must be removed to make room for an incoming page?

2) How is a page removed from physical memory?

- If the page is unmodified, simply overwrite it: **a copy already exists on disk**
- If the page has been modified, it must be written back to disk: prefer unmodified pages?
- We had a certain field in our **PTEs** that reflected this, can anyone name it?
 - The Dirty Bit!

3) Better not choose an often used page

- Will likely need to be brought back in soon

Page Replacement Algorithms

We will go over the following:

- 1) Optimal Page Replacement Algorithm
- 2) Not-Recently-Used (NRU) Algorithm
- 3) First In First Out (FIFO) Algorithm
- 4) Second Chance Algorithm
- 5) Least Recently Used (LRU) Algorithm

Optimal Page Replacement

What's the best we can possibly do?

- Assume perfect knowledge of the future
- Not realizable in practice (usually)
- Useful for comparison: if another algorithm is within 5% of optimal, not much more can be done...

Algorithm: replace the page that will be used furthest in the future

- Only works if we know the whole sequence!
- Can be approximated by running the program twice
 - Once to generate the reference trace
 - Once (or more) to apply the optimal algorithm

Nice, but not achievable in real systems!

The Assumption

- Since it is a 32-bit address space.
 - First 20 bits is used for the address
 - The rest is used for offset

Page Address Page Offset



l 190a7c20
s 3856bbe0
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28
l 190aff38

Optimal Page Replacement Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume OPT

0		
1		
2		

```
l 190a7c20
s 3856bbe0
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28
l 190aff38
```

Optimal Page Replacement Example

- Let's suppose you have 12KB of physical memory
- Page has 4KB
- Assume OPT
- You already know memory access trace at the beginning

0		
1		
2		

l 190a7c20
s 3856bbe0
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28
l 190aff38

Optimal Page Replacement Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume OPT
 - You already know memory access trace at the beginning
 - When evicting needed

0	190a7
1	3856b
2	190af

We need to evict
someone!!



Pagefault again

l 190a7c20
s 3856bbe0
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28
l 190aff38

Optimal Page Replacement Example

- Let's suppose you have 12KB of physical memory
- Page has 4KB
- Assume OPT
- You already know memory access trace at the beginning
- When evicting needed

Pagefault again

0	190a7
1	3856b
2	190af

We need to evict someone!!



l 190a7c20
s 3856bbe0
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28
l 190aff38

Optimal Page Replacement Example

- Let's suppose you have 12KB of physical memory

- Page has 4KB
- Assume OPT
- You already know memory access trace at the beginning
- When evicting needed

Pagefault again

0	190a7
1	3856b
2	190af

**We need to evict
someone!!
But page 190a7 will
be used later!**



l 190a7c20
s 3856bbe0
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28
l 190aff38

Optimal Page Replacement Example

- Let's suppose you have 12KB of physical memory **Pagefault** again

- Page has 4KB
- Assume OPT
- You already know memory access trace at the beginning
- When evicting needed

0	190a7
1	3856b
2	190af

**We need to evict
someone!!
But page 190a7 will
be used later!**

**Go next until find
one that is no
longer needed in
the future**



l 190a7c20
s 3856bbe0
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28
l 190aff38

Optimal Page Replacement Example

- Let's suppose you have 12KB of physical memory

- Page has 4KB
- Assume OPT
- You already know memory access trace at the beginning
- When evicting needed

Pagefault again

0	190a7
1	190af
2	

We need to evict
someone!!
But page 190a7 will
be used later!

Go next until find
one that is no
longer needed in
the future



l 190a7c20
s 3856bbe0
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28
l 190aff38

Optimal Page Replacement

Example

- Let's suppose you have 12KB of physical memory **Pagefault** again

- Page has 4KB
- Assume OPT
- You already know memory access trace at the beginning
- When evicting needed

0	190a7
1	190af
2	15216

**We need to evict
someone!!
But page 190a7 will
be used later!**

**Go next until find
one that is no
longer needed in
the future**

1 190a7c20
s 3856bbe0
1 190afc20
1 15216f00
1 190a7c20
1 190a7c28
1 190a7c28
1 190aff38

Optimal Page Replacement Example

- Let's suppose you have 12KB of physical memory **Pagefault** again

- Page has 4KB
- Assume OPT
- You already know memory access trace at the beginning
- When evicting needed

0	190a7
1	3856b
2	15216

We need to evict someone!!
But page 190a7 will be used later!

Go next until find one that is no longer needed in the future

1 190a7c20
s 3856bbe0
1 190afc20
1 15216f00
1 190a7c20
1 190a7c28
1 190a7c28
1 190aff38

What if there is a tie?
Multiple pages no longer needed in the future?

Optimal Page Replacement Example

- Let's suppose you have 12KB of physical memory **Pagefault** again
 - Page has 4KB
 - Assume OPT
 - You already know memory access trace at the beginning
 - When evicting needed

0	190a7
1	3856b
2	15216

**We need to evict
someone!!
But page 190a7 will
be used later!**

**Go next until find
one that is no
longer needed in
the future**

l **190a7c20**
s **3856bbe0**
l **190afc20**
l **15216f00**
l 190a7c20
l 190a7c28
l 190a7c28
l 190aff38

What if there is a tie?
Multiple pages no longer needed in
the future?
Use **LRU** among those tie pages

Not Recently Used (NRU) Algorithm

Each page has reference bit and dirty bit

- Bits are set when page is referenced and/or modified

Pages are classified into four classes

- 0: not referenced, not dirty
- 1: not referenced, dirty
- 2: referenced, not dirty
- 3: referenced, dirty

Clear reference bit for all pages periodically

- Can't clear dirty bit: needed to indicate which pages need to be flushed to disk
- Class 1 contains dirty pages where reference bit has been cleared

Algorithm: remove a page from the lowest non-empty class

- Select a page at random from that class

Easy to understand and implement

Performance adequate (though not optimal)

First-In, First-Out (FIFO) Algorithm

Maintain a linked list of all pages

- Maintain the order in which they entered memory

Page at front of list replaced

Advantage: (really) easy to implement

Disadvantage: page in memory the longest may be often used

- This algorithm forces pages out regardless of usage
- Usage may be helpful in determining which pages to keep

Second Chance Page Replacement Algorithm

Modify FIFO to avoid throwing out heavily used pages

- If reference bit is 0, throw the page out
- If reference bit is 1
 - Reset the reference bit to 0
 - Move page to the tail of the list
 - Continue search for a free page

Still easy to implement, and better than plain FIFO

Second Chance Algorithm Example

- Let's suppose you have 12KB of physical memory
- Page has 4KB
- Assume Second Chance Algorithm

0		
1		
2		

```
l 190a7c20
s 3856bbe0
l 190a7c24
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28
```

Second Chance Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Second Chance Algorithm
 - Referenced bit (R bit): set to 1 **If page accessed again after allocated in memory**

0		
1		
2		



R bits

l 190a7c20
s 3856bbe0
l 190a7c24
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28

Second Chance Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Second Chance Algorithm
 - Referenced bit (R bit)

0		
1		
2		



R bits



1 **190a7c20**
s 3856bbe0
1 190a7c24
1 190afc20
1 15216f00
1 190a7c20
1 190a7c28
1 190a7c28

Second Chance Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Second Chance Algorithm
 - Referenced bit (R bit)

0		
1		
2		



R bits

Pagefault since it is
not in the page
table



1 **190a7c20**
s 3856bbe0
1 190a7c24
1 190afc20
1 15216f00
1 190a7c20
1 190a7c28
1 190a7c28

Second Chance Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Second Chance Algorithm
 - Referenced bit (R bit)

0	0	190a7
1		
2		



R bits

Pagefault since it is
not in the page
table



1 190a7c20
s 3856bbe0
1 190a7c24
1 190afc20
1 15216f00
1 190a7c20
1 190a7c28
1 190a7c28

Second Chance Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Second Chance Algorithm
 - Referenced bit (R bit)

0	0	190a7
1		
2		



R bits



l **190a7**c20
s **3856b**be0
l 190a7c24
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28

Second Chance Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Second Chance Algorithm
 - Referenced bit (R bit)

0	0	190a7
1		
2		



R bits

Pagefault since it is not in the page table



l 190a7c20
s 3856bbe0
l 190a7c24
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28

Second Chance Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Second Chance Algorithm
 - Referenced bit (R bit)

0	0	190a7
1	0	3856b
2		



R bits

Pagefault since it is
not in the page
table



l 190a7c20
s 3856bbe0
l 190a7c24
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28

Second Chance Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Second Chance Algorithm
 - Referenced bit (R bit)

0	0	190a7
1	0	3856b
2		



R bits



l **190a7**c20
s **3856b**be0
l **190a7**c24
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28

Second Chance Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Second Chance Algorithm
 - Referenced bit (R bit)

0	0	190a7
1	0	3856b
2		


R bits

Page 190a7
accessed **again**



l **190a7**c20
s **3856b**be0
l **190a7**c24
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28

Second Chance Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Second Chance Algorithm
 - Referenced bit (R bit)

0	1	190a7
1	0	3856b
2		

↑
R bits

No evicting needed
set R bit of 190a7 to 1

Page 190a7
accessed again

1 190a7c20
s 3856bbe0
1 190a7c24
1 190afc20
1 15216f00
1 190a7c20
1 190a7c28
1 190a7c28

Second Chance Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Second Chance Algorithm
 - Referenced bit (R bit)

0	1	190a7
1	0	3856b
2		



R bits



l 190a7c20
s 3856bbe0
l 190a7c24
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28

Second Chance Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Second Chance Algorithm
 - Referenced bit (R bit)

0	1	190a7
1	0	3856b
2		



R bits

Pagefault since it is not in the page table



l 190a7c20
s 3856bbe0
l 190a7c24
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28

Second Chance Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Second Chance Algorithm
 - Referenced bit (R bit)

0	1	190a7
1	0	3856b
2	0	190af


R bits

Pagefault since it is
not in the page
table



l 190a7c20
s 3856bbe0
l 190a7c24
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28

Second Chance Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Second Chance Algorithm
 - Referenced bit (R bit)

0	1	190a7
1	0	3856b
2	0	190af



R bits



l 190a7c20
s 3856bbe0
l 190a7c24
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28

Second Chance Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Second Chance Algorithm
 - Referenced bit (R bit)

0	1	190a7
1	0	3856b
2	0	190af


R bits

Pagefault again

l 190a7c20
s 3856bbe0
l 190a7c24
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28



Second Chance Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Second Chance Algorithm
 - Referenced bit (R bit)

0	1	190a7
1	0	3856b
2	0	190af


R bits

We need to evict
someone!!

Pagefault again



l 190a7c20
s 3856bbe0
l 190a7c24
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28

Second Chance Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Second Chance Algorithm
 - Referenced bit (R bit)

0	1	190a7
1	0	3856b
2	0	190af

↑
R bits

We need to evict
someone!!

Pagefault again

1 190a7c20
s 3856bbe0
1 190a7c24
1 190afc20
1 15216f00
1 190a7c20
1 190a7c28
1 190a7c28

Second Chance Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Second Chance Algorithm
 - Referenced bit (R bit)

0	1	190a7
1	0	3856b
2	0	190af

↑
R bits

We need to evict
someone!!

Pagefault again

1 190a7c20
s 3856bbe0
1 190a7c24
1 190afc20
1 15216f00
1 190a7c20
1 190a7c28
1 190a7c28

Second Chance Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Second Chance Algorithm
 - Referenced bit (R bit)

0	1	190a7
1	0	3856b
2	0	190af

↑
R bits

We need to evict
someone!!
Entry 0: R bit is 1

Pagefault again

1 190a7c20
s 3856bbe0
1 190a7c24
1 190afc20
1 15216f00
1 190a7c20
1 190a7c28
1 190a7c28

Second Chance Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Second Chance Algorithm
 - Referenced bit (R bit)

0	0	190a7
1	0	3856b
2	0	190af

↑
R bits

We need to evict
someone!!
Entry 0: R bit is 1
Set it to 0 and go to
next entry

Pagefault again

1 190a7c20
s 3856bbe0
1 190a7c24
1 190afc20
1 15216f00
1 190a7c20
1 190a7c28
1 190a7c28

Second Chance Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Second Chance Algorithm
 - Referenced bit (R bit)

0	0	190a7
1	0	3856b
2	0	190af

↑
R bits

We need to evict
someone!!
Entry 1: R bit is 0

Pagefault again

1 190a7c20
s 3856bbe0
1 190a7c24
1 190afc20
1 15216f00
1 190a7c20
1 190a7c28
1 190a7c28

Second Chance Algorithm Example


- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Second Chance Algorithm
 - Referenced bit (R bit)

0	0	190a7
1	0	3856b
2	0	190af


R bits

We need to evict
someone!!
Entry 1: R bit is 0
Evict entry 1

Pagefault again


1 190a7c20
s 3856bbe0
1 190a7c24
1 190afc20
1 15216f00
1 190a7c20
1 190a7c28
1 190a7c28

Second Chance Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Second Chance Algorithm
 - Referenced bit (R bit)

0	0	190a7
1	0	190af
2	0	

↑
R bits

We need to evict
someone!!
Entry 1: R bit is 0
Evict entry 1

Pagefault again

1 190a7c20
s 3856bbe0
1 190a7c24
1 190afc20
1 15216f00
1 190a7c20
1 190a7c28
1 190a7c28

Second Chance Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Second Chance Algorithm
 - Referenced bit (R bit)

Pagefault again

0	0	190a7
1	0	190af
2	0	15216

↑
R bits

We need to evict
someone!!
Entry 1: R bit is 0
Evict entry 1

l 190a7c20
s 3856bbe0
l 190a7c24
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28

Second Chance Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Second Chance Algorithm
 - Referenced bit (R bit)

0	0	190a7
1	0	190af
2	0	15216

↑
R bits

We need to evict
someone!!
Entry 1: R bit is 0
Evict entry 1

Similar as FIFO but
pages accessed
again will get
another chance

Pagefault again

1 190a7c20
s 3856bbe0
1 190a7c24
1 190afc20
1 15216f00
1 190a7c20
1 190a7c28
1 190a7c28

Least Recently Used (LRU) Algorithm

Assume pages used recently will be used again soon

- Throw out page that has been unused for longest time

Must keep a linked list of pages

- Most recently used at front, least at rear
- Update this list every memory reference!
 - This can be somewhat slow: hardware has to update a linked list on every reference!

Alternatively, keep counter in each page table entry

- Global counter increments with each CPU cycle
- Copy global counter to PTE counter on a reference to the page
- For replacement, evict page with lowest counter value

LRU Algorithm Example

- Let's suppose you have 12KB of physical memory
- Page has 4KB
- Assume Least Recently Used(LRU)

0	
1	
2	



1 190a7c20
s 3856bbe0
1 190afc20
1 15216f00
1 190a7c20
1 190a7c28
1 190a7c28
1 190aff38

LRU Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Least Recently Used(LRU)

0	
1	
2	

Pagefault since it is not in the page table



1 190a7c20
s 3856bbe0
1 190afc20
1 15216f00
1 190a7c20
1 190a7c28
1 190a7c28
1 190aff38

LRU Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Least Recently Used(LRU)

0	190a7
1	
2	

Pagefault since it is not in the page table



1 190a7c20
s 3856bbe0
1 190afc20
1 15216f00
1 190a7c20
1 190a7c28
1 190a7c28
1 190aff38

LRU Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Least Recently Used(LRU)

0	190a7
1	
2	



l 190a7c20
s 3856bbe0
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28
l 190aff38

LRU Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Least Recently Used(LRU)

0	190a7
1	
2	

Pagefault since it is not in the page table



l 190a7c20
s 3856bbe0
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28
l 190aff38

LRU Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Least Recently Used(LRU)

0	190a7
1	3856b
2	

Pagefault since it is not in the page table




l 190a7c20
s 3856bbe0
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28
l 190aff38

LRU Algorithm Example

- Let's suppose you have 12KB of physical memory
- Page has 4KB
- Assume Least Recently Used(LRU)

0	190a7
1	3856b
2	



l 190a7c20
s 3856bbe0
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28
l 190aff38

LRU Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Least Recently Used(LRU)

0	190a7
1	3856b
2	190af

Pagefault since it is not in the page table



l 190a7c20
s 3856bbe0
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28
l 190aff38

LRU Algorithm Example

- Let's suppose you have 12KB of physical memory
- Page has 4KB
- Assume Least Recently Used(LRU)

0	190a7
1	3856b
2	190af



l **190a7**c20
s **3856b**be0
l **190af**c20
l **15216**f00
l 190a7c20
l 190a7c28
l 190a7c28
l 190aff38

LRU Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume Least Recently Used(LRU)

0	190a7
1	3856b
2	190af

We need to evict
someone!!

Pagefault again



l 190a7c20
s 3856bbe0
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28
l 190aff38

LRU Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume **Least Recently Used(LRU)**

0	190a7
1	3856b
2	190af

We need to evict
someone!!

Pagefault again



l 190a7c20
s 3856bbe0
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28
l 190aff38

LRU Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume **Least Recently Used(LRU)**

0	190a7
1	3856b
2	190af

We need to evict
someone!!

Pagefault again



l 190a7c20
s 3856bbe0
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28
l 190aff38

LRU Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume **Least Recently Used(LRU)**

0	3856b
1	190af
2	

We need to evict
someone!!

Pagefault again



l 190a7c20
s 3856bbe0
l 190afc20
l 15216f00
l 190a7c20
l 190a7c28
l 190a7c28
l 190aff38

LRU Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Assume **Least Recently Used(LRU)**

Pagefault again

0	3856b
1	190af
2	15216

**We need to evict
someone!!**



l **190a7c20**
s **3856bbe0**
l **190afc20**
l **15216f00**
l 190a7c20
l 190a7c28
l 190a7c28
l 190aff38

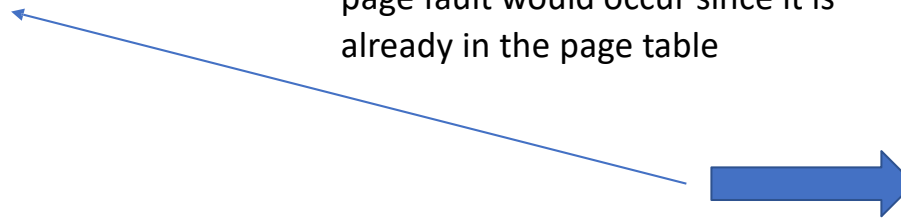
LRU Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB

0	3856b
1	190af
2	15216

Assume we skip to page **190af** no
page fault would occur since it is
already in the page table

l **190a7c20**
s **3856bbe0**
l **190afc20**
l **15216f00**
l 190a7c20
l 190a7c28
l 190a7c28
l **190aff38**



LRU Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB

0	3856b
1	190af
2	15216

However we need to change its position since it was **recently used**

l **190a7c20**
s **3856bbe0**
l **190afc20**
l **15216f00**
l 190a7c20
l 190a7c28
l 190a7c28
l **190aff38**



LRU Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB

0	3856b
1	15216
2	190af

However we need to change its position since it was **recently used**

l **190a7c20**
s **3856bbe0**
l **190afc20**
l **15216f00**
l 190a7c20
l 190a7c28
l 190a7c28
l **190aff38**



LRU Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Set dirty bit to true if a store

0	3856b	1
1	190af	0
2	15216	0

Set dirty bit to true for a store

1 190a7c20
S 3856bbe0
1 190afc20
1 15216f00
1 190a7c20
1 190a7c28
1 190a7c28
1 190aff38



LRU Algorithm Example

- Let's suppose you have 12KB of physical memory
 - Page has 4KB
 - Set dirty bit to true if a store

0	3856b	1
1	190af	0
2	15216	0

Set dirty bit to true for a store



Dirty bit identifies a memory address that will need to be "written to" since it was modified.

1 190a7c20
S 3856bbe0
1 190afc20
1 15216f00
1 190a7c20
1 190a7c28
1 190a7c28
1 190aff38