
Lecture 15: Page Placement

CSC 469H1F / CSC 2208H1F

Fall 2007

Angela Demke Brown



Memory Management Policies

- Recall from 369, 3 policies characterize a virtual memory management scheme:
 - Fetch Policy - *when* to fetch a page
 - Placement Policy - *where* to put the page
 - Are some physical pages preferable to others?
 - Replacement Policy - *what* page to evict to make room?

Placement Policy

- Address translation allows us to allocate any physical page for any virtual page
- We'll look at 3 reasons why choosing physical pages carefully can be better than random placement
 - *Cache conflicts*
 - NUMA multiprocessors
 - Energy savings

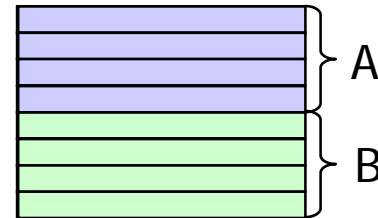
Cache Access

- Data is loaded into cache by blocks called *lines*
 - 32 - 128 byte line sizes are typical
- Restrictions on block placement create 3 categories of cache organization:
 - Each block can be stored in exactly 1 location in the cache → *direct-mapped*
 - Mapping is (block address) modulo (# blocks in cache)
 - Any block can be stored in any cache line → *fully associative*
 - Each block can be stored in a restricted set of locations in the cache → *set associative*
 - Map block address to set first using (block addr) % (# of sets), then place block within set
 - If N locations in a set, called *N-way set associative*

Direct Mapped Example

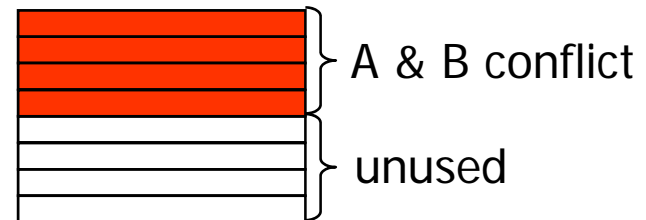
- 8 byte line size, 8 lines in cache \rightarrow 64 bytes total cache size
- 32 byte page size \rightarrow data from one page will occupy 4 lines in cache
- Case 1: access all bytes on pages 2 and 3 (A on page 2, B on page 3)

```
for (i=0; i < 32; i++)  
    A[i] = B[i];
```



- Case 2: access all bytes on pages 2 and 4

```
for (i=0; i < 32; i++)  
    A[i] = B[i];
```



What address is used?

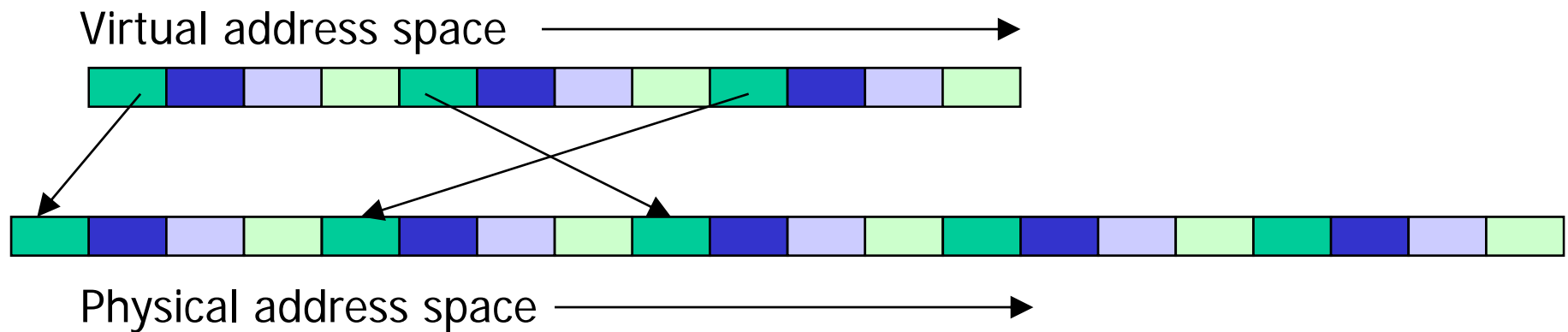
- Virtual address
 - ✓ does not need to be translated before checking cache
 - ✓ Application programmer can reason about conflicts
 - ✗ Cache needs to be flushed on context switch
- Physical address
 - ✓ Data may stay in cache across context switches
 - ✗ Vaddr must be translated before checking cache
 - ✗ conflicts depend on what physical page is allocated

Conflict-aware page placement

- OS can select physical pages on allocation to try to reduce cache conflicts
- IDEA: assign a *colour* to each page such that pages with different colours do not conflict in the cache
 - All pages with same colour map to same lines or sets in the cache
 - Number of colours = $(\text{cache size}) / (\text{pg size} * \text{associativity})$
 - In previous example, 2 colours
 - A page's colour is $(\text{page number}) \bmod (\text{num colours})$
- 2 main OS allocation strategies:
 - *Page coloring*
 - *Bin Hopping*

Page Coloring

- Assign colour to virtual and physical pages
- On page fault, allocate a physical page with the same colour as the virtual page
 - Exploits spatial locality
 - Programmer reasoning about virtual addresses still applies
 - Implemented in SGI Irix, Solaris, NT
 - OS keeps per-color free lists

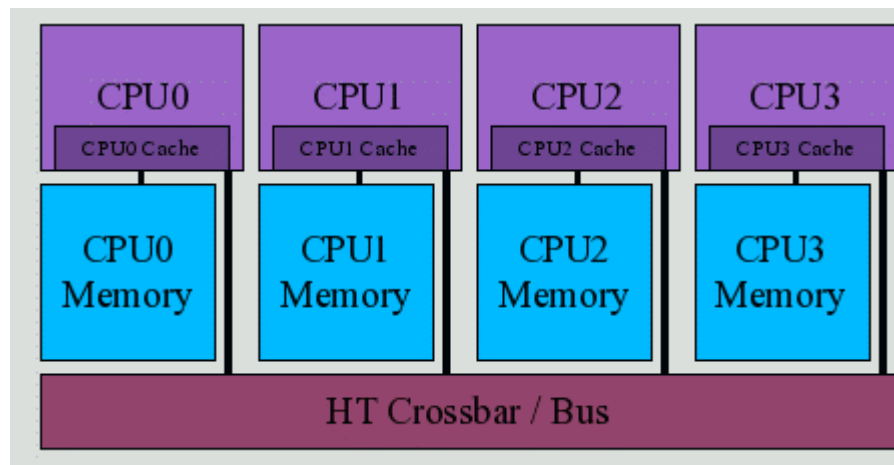


Bin Hopping

- Assign colors to physical pages and keep per-colour free lists as before
- On page fault, allocate physical page of next colour from last one previously allocated
 - Exploits temporal locality
 - Implemented in Digital Unix

NUMA Multiprocessors

- NUMA == Non Uniform Memory Access
- Multiprocessor design where each processor (or small set of processors) have a bank of local memory, but can also access remote memory
 - Local memory is faster to access than remote

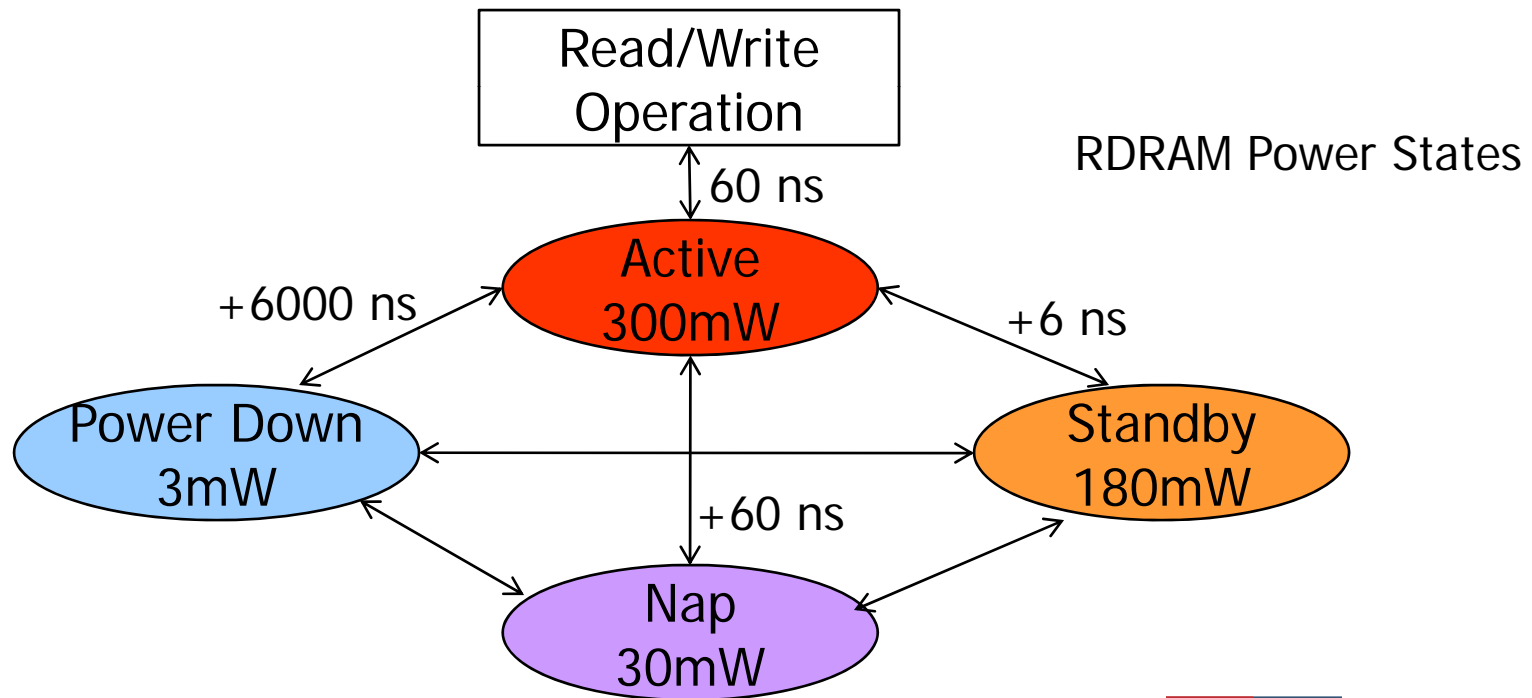


NUMA Page Placement

- Want to allocate “local” memory as much as possible
 - Local at allocation time may not be local at access time
 - May want to migrate pages
- Keep per-memory bank free lists
 - Possibly in addition to per-color lists
- SGI Irix made NUMA placement policy user-selectable
 - Round-robin
 - Random
 - First touch
 - Migratable / non-migratable

Power Aware Page Placement

- Keeping memory contents valid consumes power
 - Repeatedly refresh memory cells
- Memory chips can have multiple power states



Exploiting multiple power states

- Physical address determines which chip is accessed
 - Addresses 0 to $N-1$ to chip 0, N to $2N-1$ to chip 1
- Virtual memory page allocation influences chip-level locality
- Try to cluster page allocation to improve chances to power-down some memory chips
 - Exploit temporal locality
 - Need to be aware of both added delay and savings in power usage