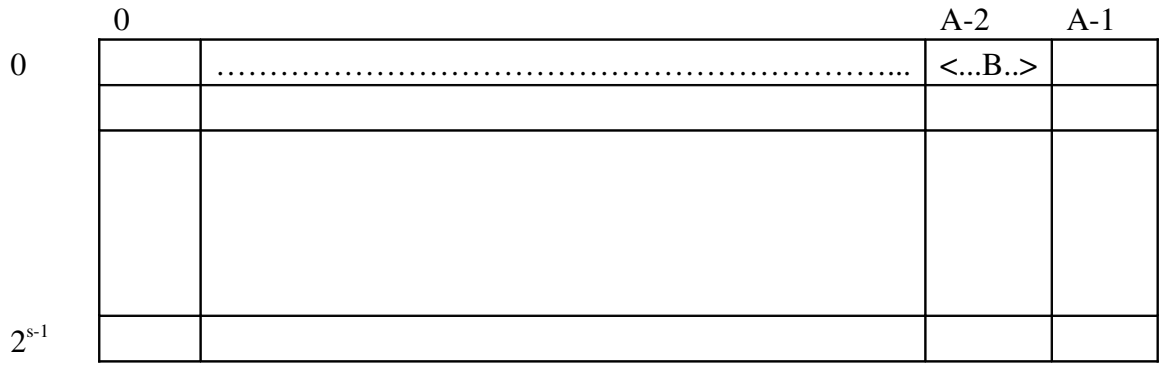
Cache Fundamentals

Cache Memory is characterized by three parameters

1. Associativity: Decides the cache location where a block be placed. There are three type of block placement policies
    1. Direct Mapped
    2. Fully Associative
    3. Set Associative
2. Block Size: Block size is size of data read/written in each cache operation. It is also known as cache line size
3. Capacity: Size of the cache is known as its capacity.

How is a Block found in the Cache?

Apartment Analogy of cache:



Total cache capacity: C = S.A.B

Where 2^{S-1} = #of sets

A = Associativity of each set

B = Block Size

Block Address		Block Offset
Tag	Index	

- Each memory address referenced by a CPU instruction is treated as consisting of two parts by cache management unit – Block Address and Block Offset.
- Block address component is further divided into two parts:
    - Index identifies the cache set which generated address maps to.
    - After identifying cache set, the Tag field is used to distinguish between all possible memory addresses which get mapped to same cache set.
- After identifying cached block, the block offset part identifies the offset within the cached block where the referenced data may be found.

3-C model: Cache misses

- Cold misses: Compulsory miss when first time you're accessing a memory block.
- Capacity misses: Occurs when the amount of data accessed is greater than the actual cache capacity -- essentially it occurs even when fully associative cache is in use.
- Conflict misses: Occurs when the set to which reference address resolves to is full.

4th C: Coherency miss, occurs in parallel architectures

Example: MVM:

y<-A(mxn).x(n)

code:

```
do i = 0 to m - 1,
        y[i] := 0
        do j = 0 to n -1,
                y[i] := y[i] + A[i,j] * x[j];
        od
od
```

Cold misses: Occur when A, x as well as y is accessed.

# of Cold Misses = e/B{m+n+m.n}

Where e = size of each element of arrays A, x and y.
B = block size.

Capacity misses:
None for small problem size.

Cold misses are always constant; conflict misses increase (Why?).

Conflict Misses (Pathological Case):
Consider a language like FORTRAN, which stores matrices in column-major order. If the code depicted earlier is executed as FORTRAN program, each access to an element of A will bring in a cache block sized data into cache. Since elements of A are stored in column major order, that results in some portion of A's column being brought into cache. Now consider a pathological case, where all elements in a row of matrix A map to a single cache set. In this case, if the elements of A are accessed in row major order, as done by the code above, each periodic read on A will result in eviction of cache entry from a particular set.

In such a case, we can use loop interchange transformation to avoid conflict misses.

```
{Assume all y[i]'s are 0}
do j = 0 to n - 1,
        do i = 0 to m - 1,
                y[i] := y[i] + A[i,j] * x[j];
        od
od
```

What kind of locality is exploited by above code?
A - spatial, x - temporal, y - spatial

Can we do better than this?
- Blocked implementation of algorithm.

Trade-offs:
Principle: When any data block is brought into the cache, it is used as much as possible before it gets evicted.

New code (Strip-mining technique, assume row-major order of storage):

```
do i = 0, m-1, B
        do j = 0 to n-1
                do k = i, min(i+B-1, m-1)
                        y[k] = y[k] + A[k, j] * x[j];
                od
        od
od
```

In the new code, the core compute kernel does MVM for a problem size = B. B is chosen so that all three sub-arrays of A, x and y fit into the cache. The access to x can be changed to blocked in similar fashion. (Not shown here). Such computation, regardless of the problem size suffers only from cold misses. (Also known as blocked matrix-vector multiply)

Address Translation in Virtual Memory:

Virtual memory is a technique which gives an application program the impression that it has contiguous working memory, while in fact it is physically fragmented and may even overflow on to disk storage.

Page:
A page is a block of contiguous virtual memory addresses.

Page Tables:
Almost all implementations use page tables to translate the virtual addresses seen by the application

program into physical addresses (also referred to as "real addresses") used by the hardware to process instructions. Each entry in a page table contains: the starting virtual address of the page; either the real memory address at which the page is actually stored or an indicator that the page is currently held in a disk file (if the system uses disk files to let applications use amounts of virtual memory which exceed real memory).

All addresses in program are virtual addresses.

Translation Lookaside Buffer (TLB):
A Translation Lookaside Buffer (TLB) is a CPU cache that is used by memory management hardware to improve the speed of virtual address translation. The recent memory references are stored in this cache, so that future access to the same address can be resolved without going through page table lookup, which is expensive.

Why is TLB important?
- miss on TLB is very expensive (~100-~2000 cycles). 1% miss rate on TLB really hurts.
Remember: not only data locality but also locality of referenced addresses is very important.