Computer Organization Notes

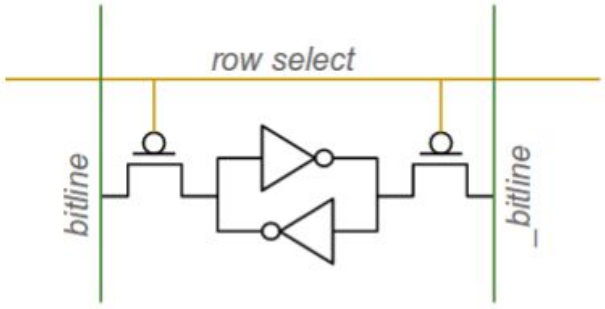
[6] The Memory Hierarchy

By Sandesh Paudel

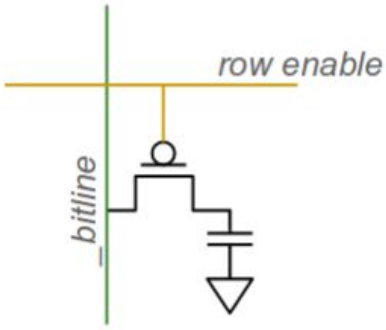
- **memory system != linear array of bytes** [that was the simplistic model]
- **memory system** = hierarchy of storage devices with different capacities, cost and access times.
- **Register:** hold the most frequently used data.
- **Cache memories:** small, fast memory nearby the CPU that act as staging areas for a subset of the data and instructions stored in the relatively slow main memory. THey hold blocks of most recently referenced instructions and data items.
- **Problem with storage:**
  - Bigger storage is slower. It takes longer to determine the location.
  - Faster storage is more expensive. In order of most to least expensive: (Flip-Flop (27 transistors!), SRAM, Dram, Disk, Tape)

[6.1] Storage Technologies

- Ram comes in two varieties: **Static RAM (SRAM) and Dynamic RAM (DRAM)**
- **Static RAM (SRAM):**
  - > faster and significantly more expensive
  - > used for cache memories
  - > **bistable:** will retain its value (2 possible values) as long as its powered.
  - > 6 transistors/bit
  - > Densest logic-only memory



- **Dynamic RAM (DRAM):**
  - > stores each bit as a charge on a capacitor.
  - > very sensitive to any disturbance (for eg, exposure to light rays). Once disturbed, will never recover.
  - > **leaks!** Will lose its charge within 10 to 100 ms.
  - > memory system must periodically **refresh** every bit of memory by reading and rewriting it. This is a major source of power consumption for DRAM.



	Transistors per bit	Relative access time	Persistent?	Sensitive?	Relative cost	Applications
SRAM	6	1×	Yes	No	1,000×	Cache memory
DRAM	1	10×	No	Yes	1×	Main memory, frame buffers

❖ **Volatile vs. Nonvolatile Memories**

- DFF, DRAM and SRAM are volatile memories
  - Lose information if powered off.
- Nonvolatile memories retain value even if powered off
  - Flash (~ 5 years)
  - Hard Disk (~ 5 years)
  - Tape (~ 15-30 years)
  - DNA (centuries)
- Uses for Nonvolatile Memories
  - Firmware (BIOS, controllers for disks, network cards, graphics accelerators, security subsystems,…)
  - Files in Smartphones, mp3 players, tablets, laptops
  - Backup

[6.2] Locality

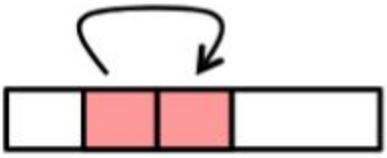
❖ **Principle of Locality:** Programm access the same set of data items over and over again, or access sets of nearby data items for a faster performance.

› Consider this for-loop:

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

> **Temporal Locality:** recently referenced items are likely to be referenced again in the near future. For eg, the reference variable sum in each iteration.

> **Spatial Locality:** Items with nearby addresses tend to be referenced close together in time. For eg, the reference instructions in sequence like the array elements in succession.



> **both hardware and software exploit locality.**

> **At the hardware level,** cache memories hold blocks of most recently referenced instructions and data items.

> **At the operating system level,** the system uses **main memory** as a cache for the most recently referenced chunks of **virtual address space**. The OS also uses main memory to cache most recently used disk blocks in the disk file system.

> **Stide-1 reference pattern [good spatial locality]:** visits each element sequentially in the memory. **Stride-k** pattern is when you visit every kth element of a contiguous vector.

> **Example of bad spatial locality:**

```
1   int sumarraycols(int a[M][N])
2   {
3       int i, j, sum = 0;
4
5       for (j = 0; j < N; j++)
6           for (i = 0; i < M; i++)
7               sum += a[i][j];
8       return sum;
9   }
```

(a)

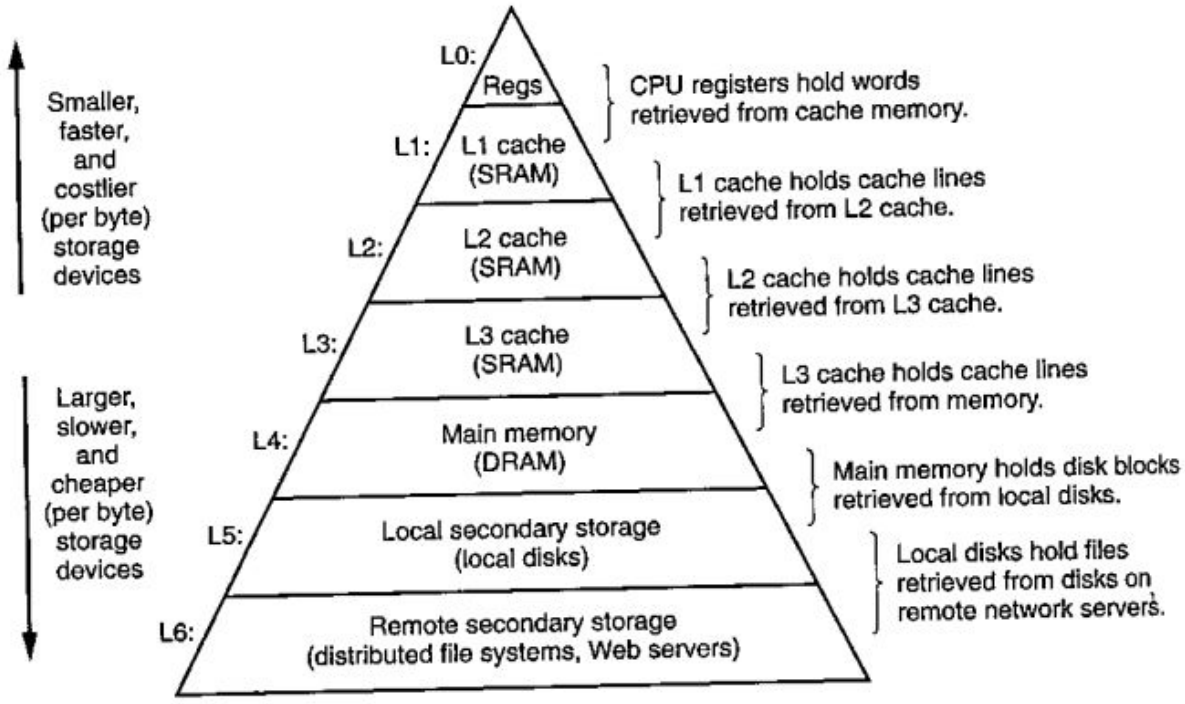
- Programs that repeatedly reference the same variables enjoy good temporal locality.
- For programs with stride-*k* reference patterns, the smaller the stride, the better the spatial locality. Programs with stride-1 reference patterns have good spatial locality. Programs that hop around memory with large strides have poor spatial locality.
- Loops have good temporal and spatial locality with respect to instruction fetches. The smaller the loop body and the greater the number of loop iterations, the better the locality.

[6.3] Memory Hierarchy

❖ **Main Idea:** We want Memory that is both **fast** and **large.** But this is not possible through a single level of Memory.

> **Memory Hierarchy:** have multiple levels of storage (**fastest near the processor**, and **bigger and slower on further levels**.

> **ensures that most of the data the processor needs in the near future is kept in the faster levels.**



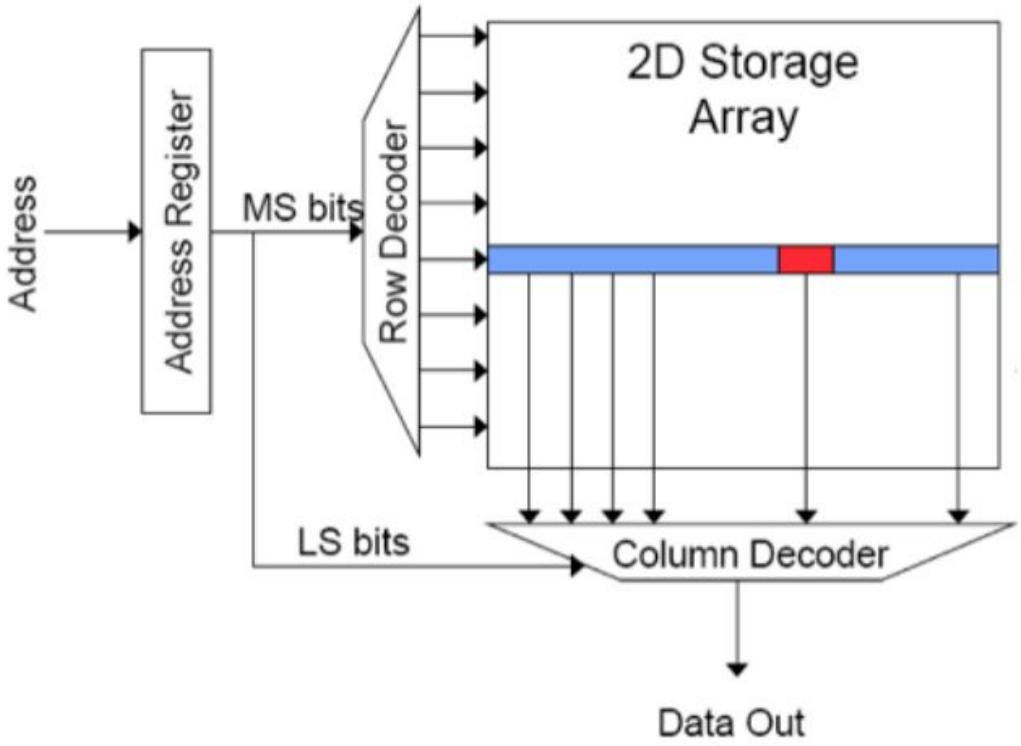
[6.32] Basics of SRAM/DRAM Technologies

❖ **RAM Terminology:**

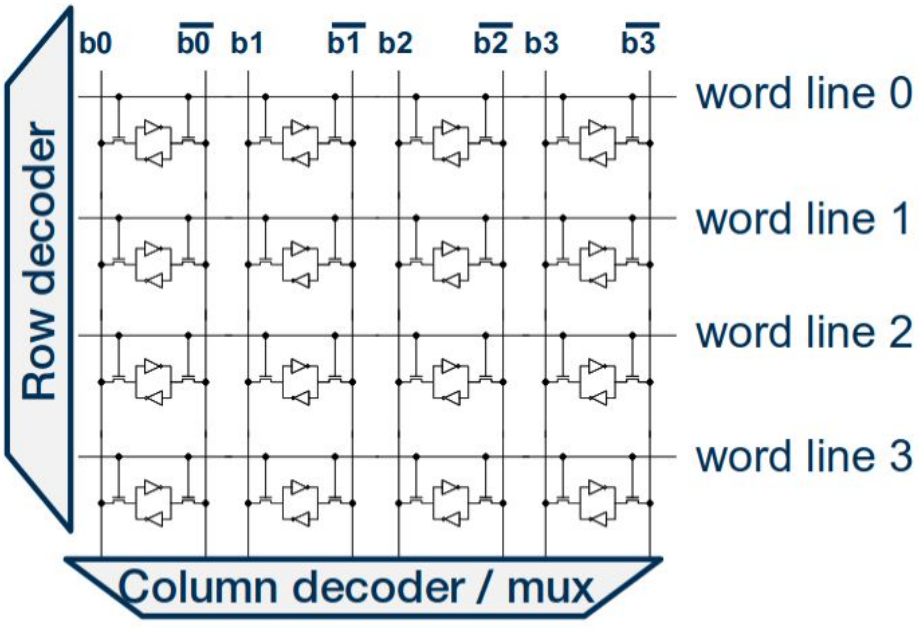
> **cell:** basic storage unit. **Stores one bit.**

- **supercell:** collection of cells. (**8 bits, 16 bits..**)
- **RAM:** a chip that is a collection of **supercells.**
- **Memory Module:** multiple RAM chips.
- **SRAM:** implements cache.
- **DRAM:** implemens main memory.

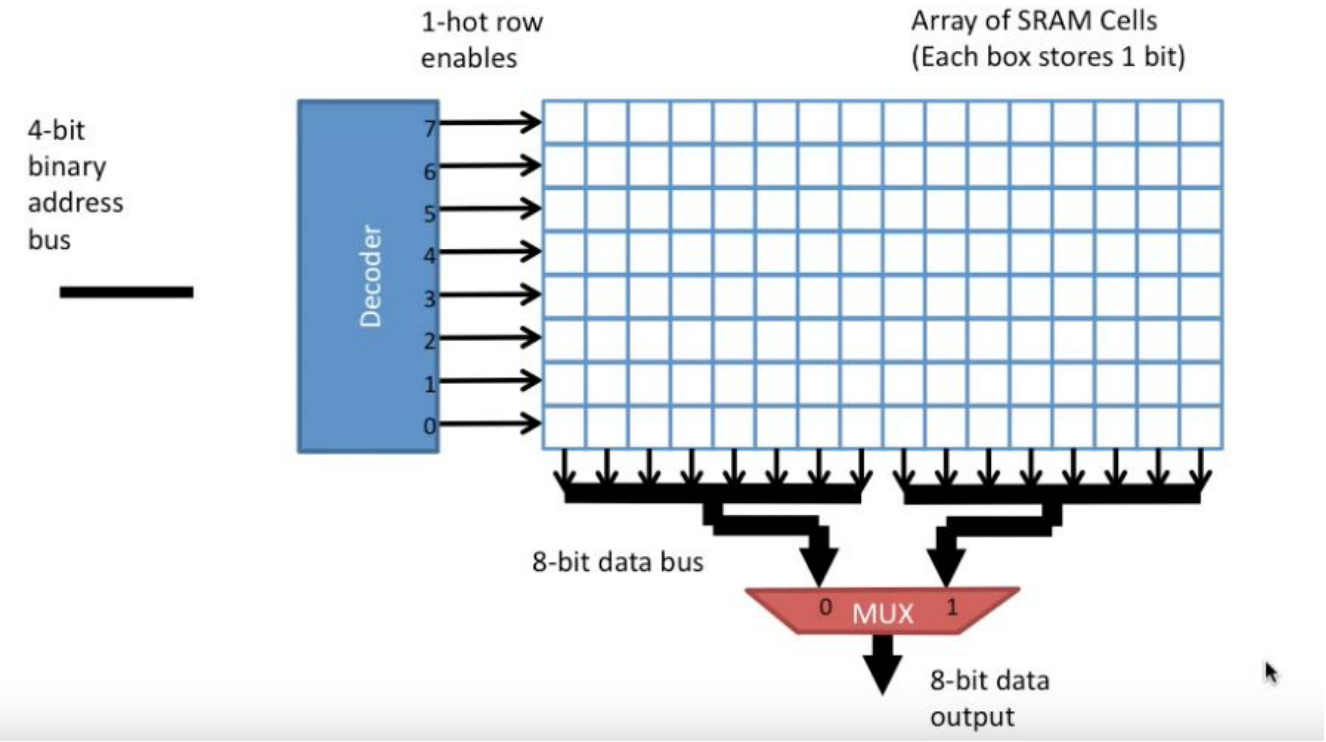
❖ **RAM Hardware:**
- **organized as 2D Array.**
- **address is spit into row address and column address.**

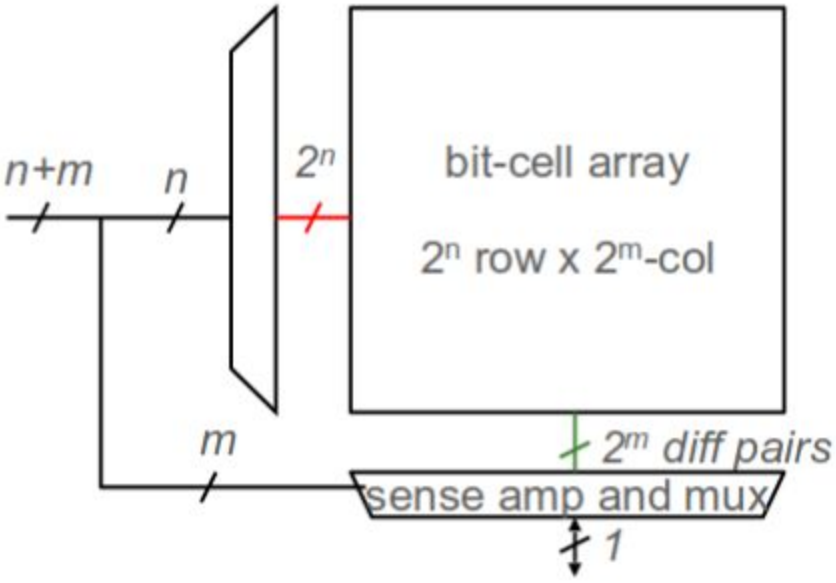


❖ **SRAM ARRAY HARDWARE: (word lines are rows, bit lines are columns)**



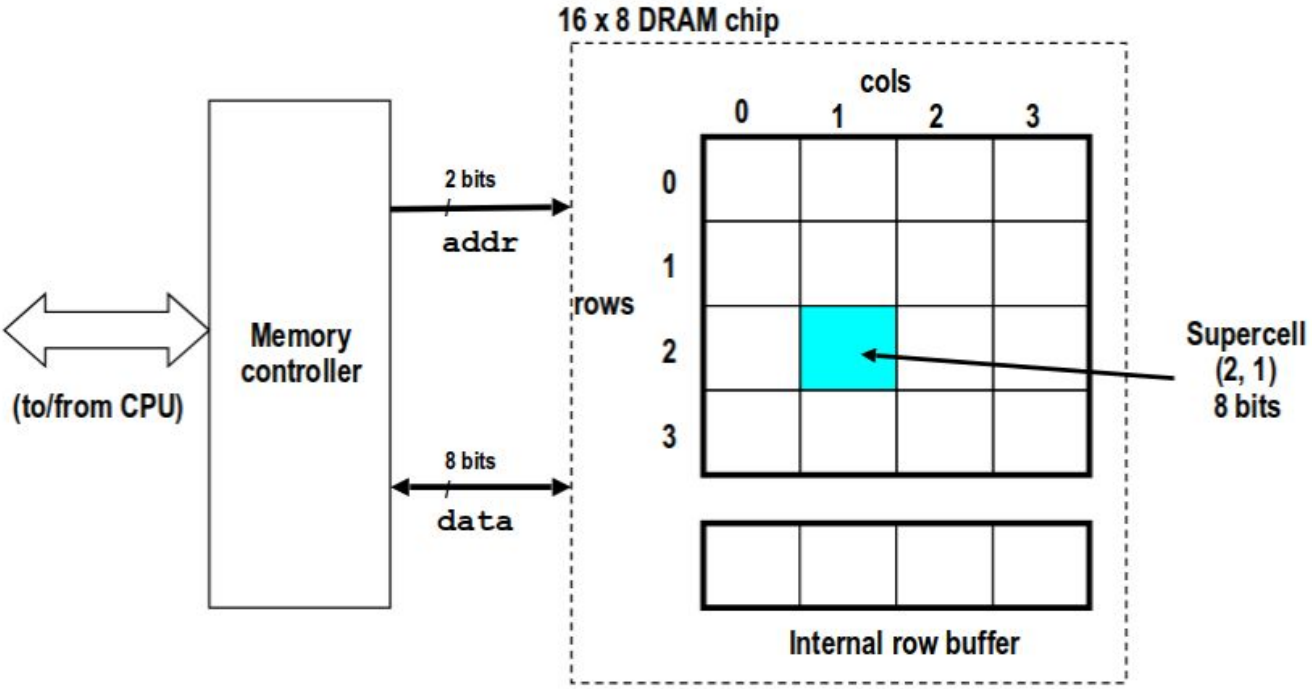
***Example of hardware implementation of SRAM CELLS**



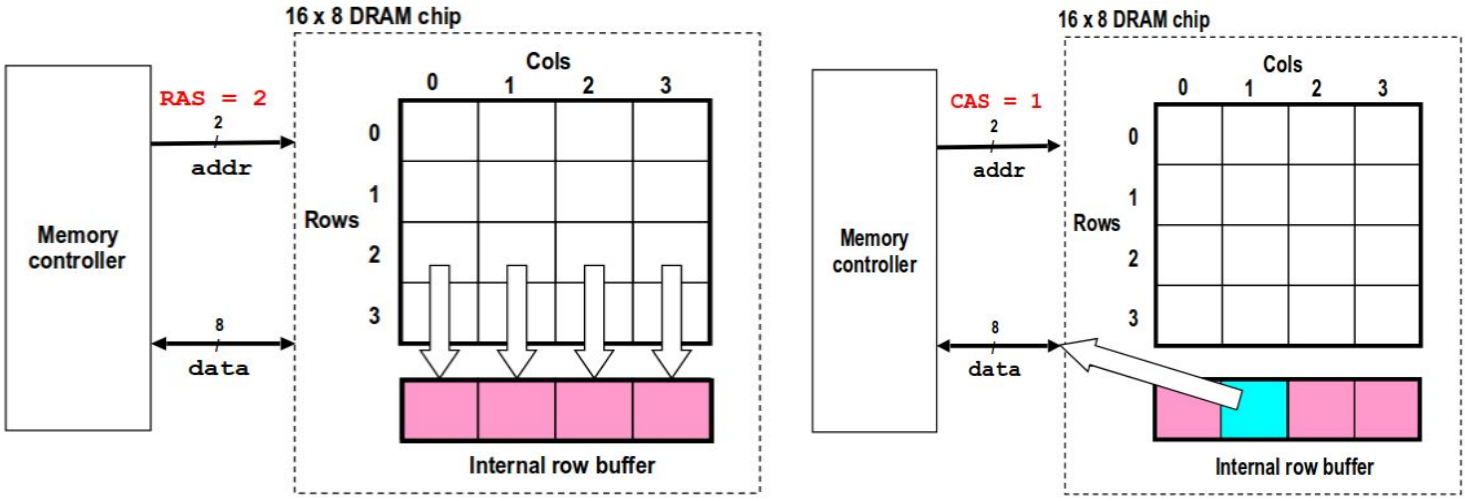


> Some notes on SRAM:
- SRAMS are laid out very carefully. Producing them over many bits reduces cost.
- In large arrays, the latency of access is dominated by the latency of the wires. The larger the array, the slower it is.

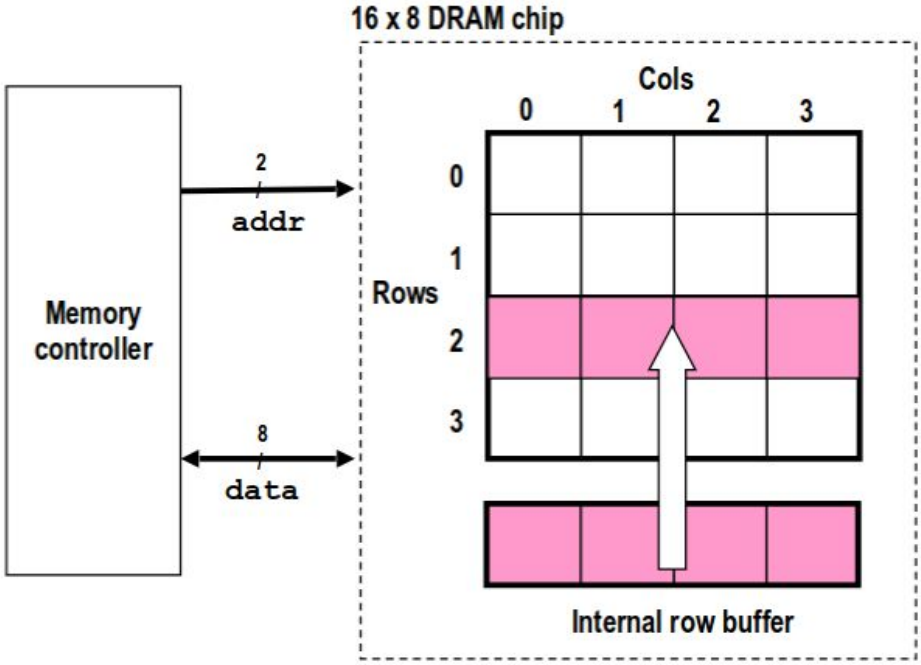
❖ DRAM Chip Hardware:



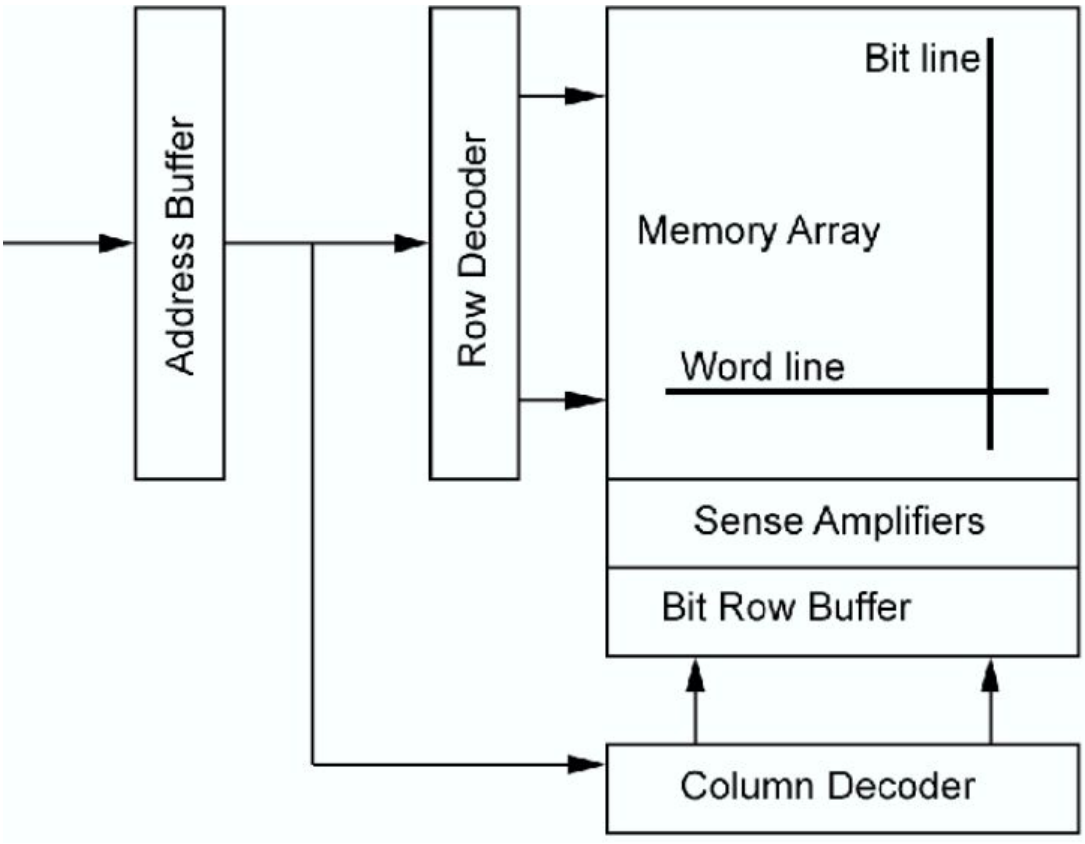
> DRAM Read Operations:
- The RAS (row access strobe) selects row 2.
- All of Row 2 is copied to the internal row buffer.
- The Column access strobe selects column 1.
- Supercell (2,1) is copied from buffer to data lines and back to CPU.



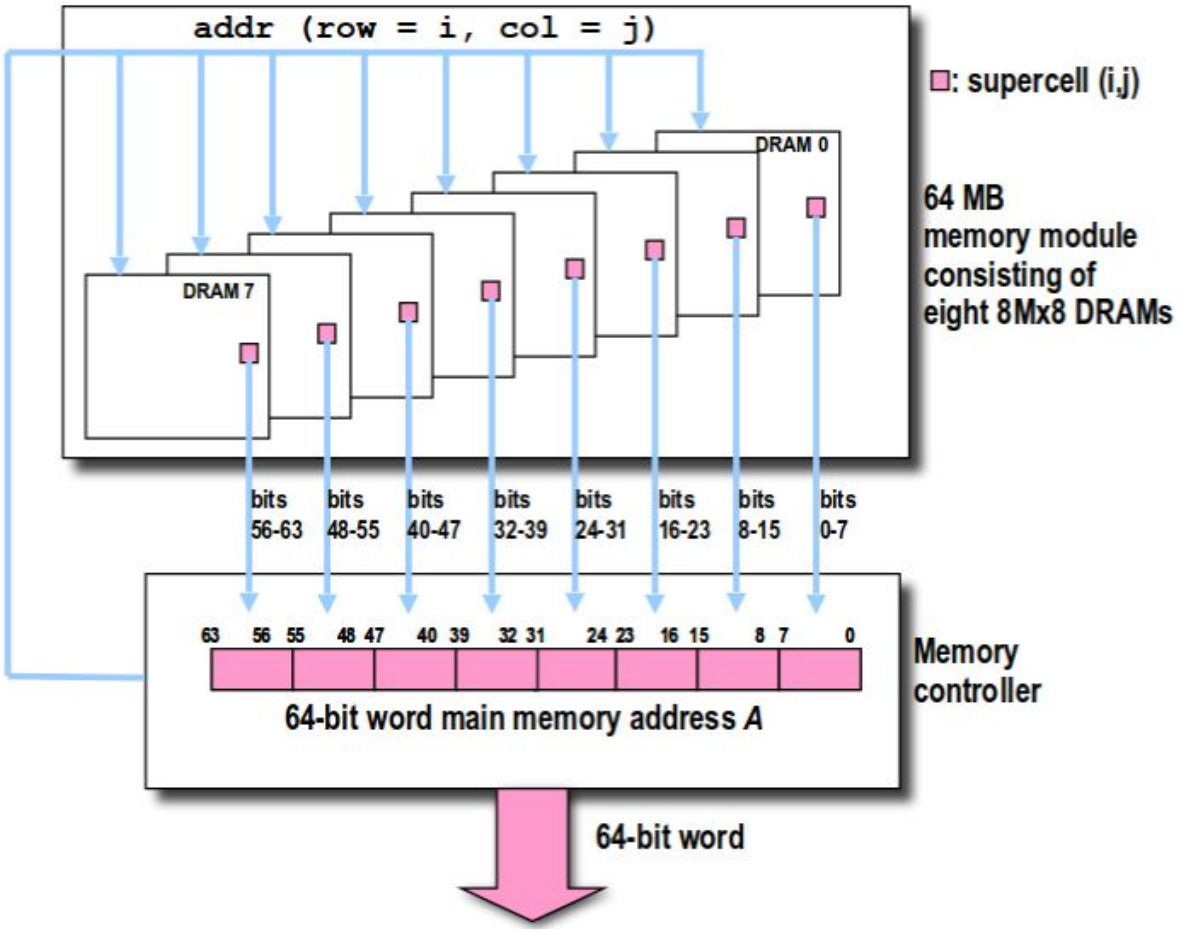
- **And finally, a Sense amplifier amplifies and regenerates the bitline(column) to refresh the cells. A DRAM controller must periodically read each row within the refresh time (10ms) to restore charge.**



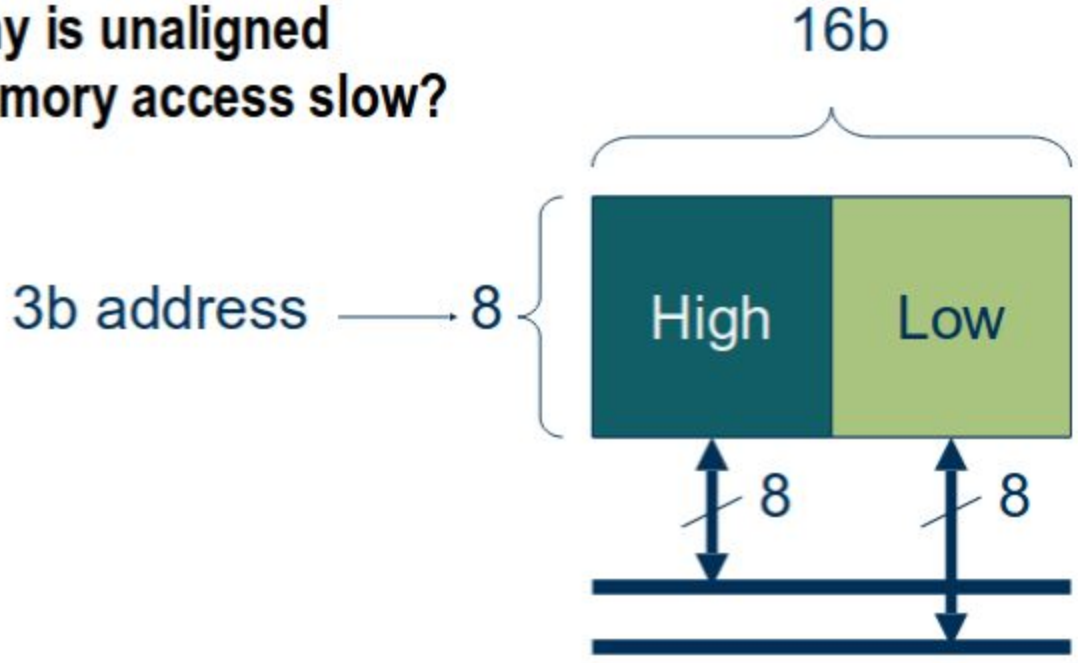
-
-



❖ **Memory Modules:**



Why is unaligned memory access slow?

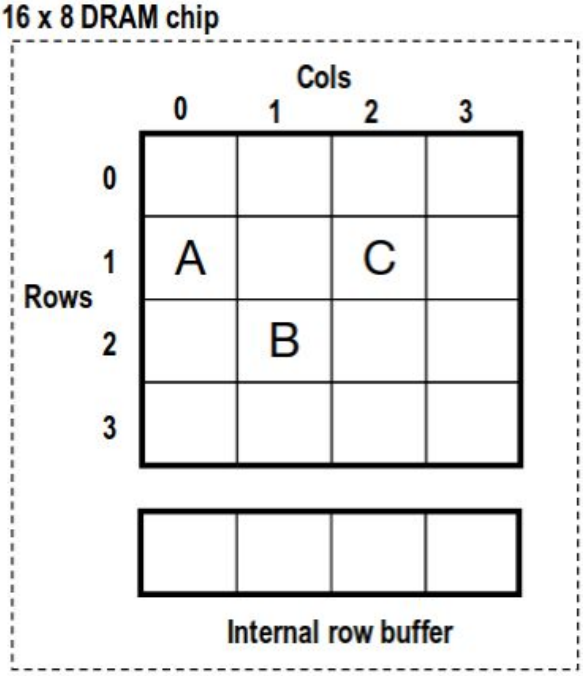


Narrower RAMs Enable Greater Capacity

- Given Constant Total Width (pins)
- Multiple smaller chips are more reliable than one big chip
  - Yield rate issue

Memory Scheduling

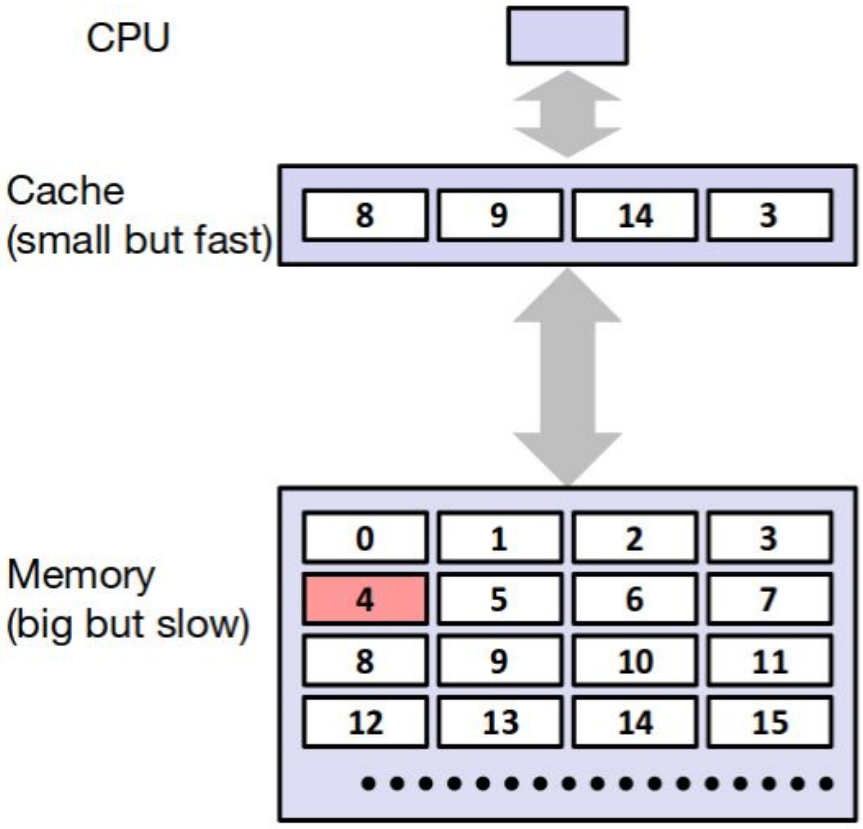
- Assume the following memory accesses: A, B, C
- Which one is faster?
  - A —> B —> C
  - A —> C —> B
- Most common memory scheduling policy: FR-FCFS
  - First-ready, first-come-first-serve
  - Prioritize addresses to data that is already in the row buffer; otherwise first-come-first-serve



[6.4] Cache Memories

> early computers had a hierarchy of three levels {CPU registers, main memory and disk storage}

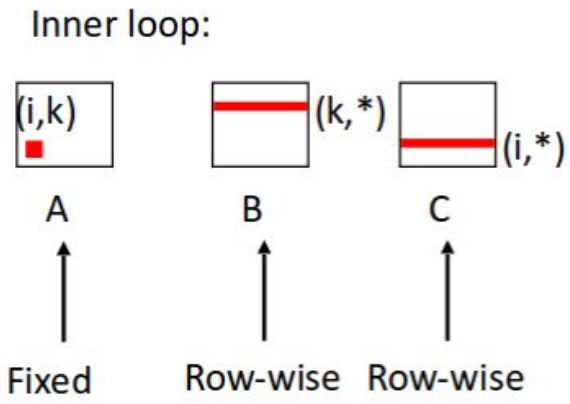
> inserted a small SRAM cache memory called L1 cache between cpu register and memory to bridge the increasing gap between CPU and main memory. They would later add L2, L3.



Matrix Multiplication (ikj)

```
/* ikj */
for (i=0; i<n; i++) {
  for (k=0; k<n; k++) {
    r = a[i][k];
    for (j=0; j<n; j++)
      c[i][j] += r * b[k][j];
  }
}
```

matmult/mm.c



Misses per inner loop iteration:

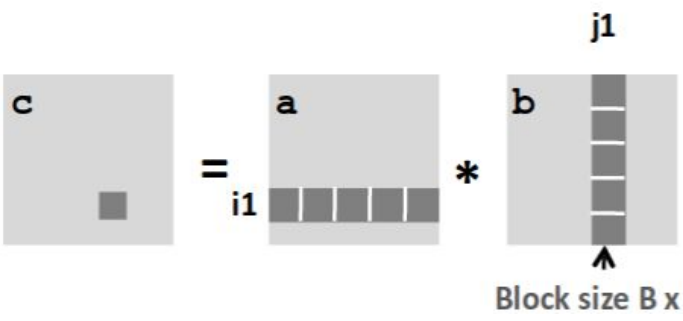
A	B	C
0.0	0.25	0.25

Blocked Matrix Multiplication

```
c = (double *) calloc(sizeof(double), n*n);

/* Multiply n x n matrices a and b  */
void mmm(double *a, double *b, double *c, int n) {
    int i, j, k;
    for (i = 0; i < n; i+=B)
        for (j = 0; j < n; j+=B)
            for (k = 0; k < n; k+=B)
               /* B x B mini matrix multiplications */
                 for (i1 = i; i1 < i+B; i++)
                     for (j1 = j; j1 < j+B; j++)
                         for (k1 = k; k1 < k+B; k++)
                            c[i1*n+j1] += a[i1*n + k1]*b[k1*n + j1];
}
```

matmult/bmm.c



- No blocking: (9/8) * n^3
- Blocking: 1/(4B) * n^3

- Suggest largest possible block size B, but limit $3B^2 < C$!

- Reason for dramatic difference:
  - Matrix multiplication has inherent temporal locality:
    - Input data: $3n^2$, computation $2n^3$
    - Every array elements used O(n) times!
  - But program has to be written properly

Cache Summary

- Cache memories can have significant performance impact

- You can write your programs to exploit this!
  - Focus on the inner loops, where bulk of computations and memory accesses occur.
  - Try to maximize spatial locality by reading data objects with sequentially with stride 1.
  - Try to maximize temporal locality by using a data object as often as possible once it's read from memory.