1 Memory Hierarchy

1.1 Storage Technologies

RAM Random Access Memory. SRAM is faster (10×) but more expensive (1000×) than DRAM.

SRAM Static Random Access Memory. Each bit is stored in a bistable memory cell. It can stay indefinitely in one of the two voltage configurations. Insensitive to disturbance.

DRAM Dynamic Random Access Memory. Each bit is stored as charge on a capacitor. Sensitive to disturbance.

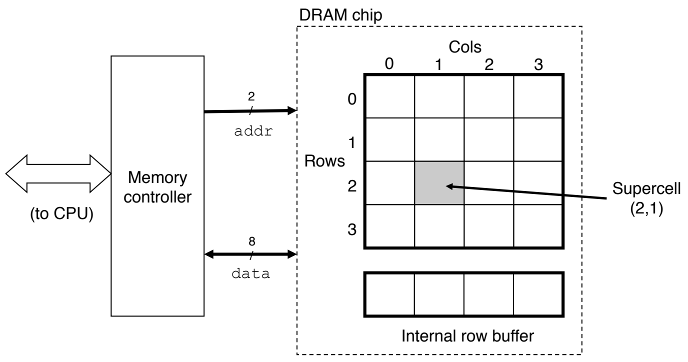


Figure 1: A 128 bit (16×8) DRAM chip

FPM DRAM Fast Page Mode DRAM.

EDO DRAM Extended Data Out DRAM.

SDRAM Synchronous DRAM.

DDR SDRAM Double Data-Rate SDRAM.

VRAM Video RAM.

RAS/CAS Row/Column Access Strobe. Row/Column address sent from the memory controller to the DRAM chip.

ROM Read-Only Memory. (Nonvolatile Memory).

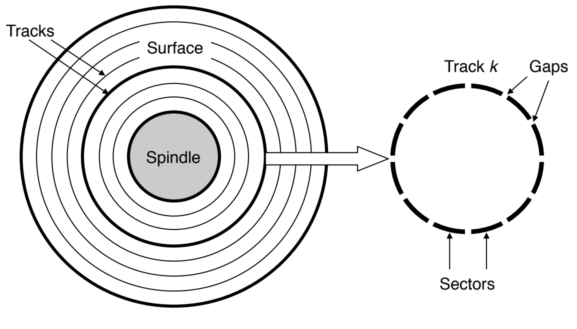
PROM Programmable ROM. Programmable only once.

EPROM Erasable PROM. Programmable ˜1000 times.

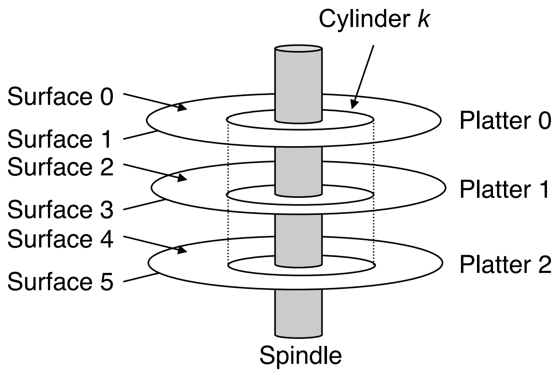
EEPROM Electrically Erasable PROM. Programmable ˜10^5 times. **Flash memory** is based on EEPROM. **SSD**(Solid State Disk) is based on flash memory.

Firmware Programs stored in ROM.

Bus Data flows between CPU and DRAM main memory through the buses. Each data transfer is called a **bus transaction**, either **read transaction** or **write transaction**.



(a) Single platter



(b) Multiple platters

Figure 2: Disk geometry

Disk Platter, surface, spindle, cylinder, track, sector, gap. Rotation rate, RPM(Revolution Per Minute).

$$Capacity = \frac{Bytes}{Sector} \times \frac{Sectors}{Track} \times \frac{Tracks}{Surface} \times \frac{Surfaces}{Platter} \times \frac{Platters}{Disk}$$

Access time:

- Seek time: $T_{\text{avg seek}} = 3 \sim 9\,ms$.
- Rotational latency: $T_{\text{avg rotation}} = \frac{1}{2} \times \frac{1}{RPM} \times \frac{60s}{1min} \sim T_{\text{avg seek}}$
- Transfer time: $T_{\text{avg transfer}} = \frac{1}{RPM} \times \frac{1}{Sectors/Track} \frac{60s}{1min} \ll T_{\text{avg seek}}$.

Disk controller maintains the mapping between logical blocks and physical disk sectors.

I/O Bus Unrelated to CPU. Intel: PCI(Peripheral Component Interconnect) bus. Connects to:

- USB(Universal Serial Bus) controller: connects to USB devices.
- Graphics card/adapter.
- Host bus adapter: connects to one or more disk drives.

Memory mapped I/O CPU uses memory-mapped I/O to send instructions to I/O devices via I/O ports, i.e. a series of special addresses in the memory space.

DMA Direct Memory Access. I/O devices carry out read/write transactions without interference of CPU.

1.2 Locality & memory hierarchy

- Temporal locality & spacial locality
- Stride-k reference pattern: stride-1 reference pattern has the best spacial locality.

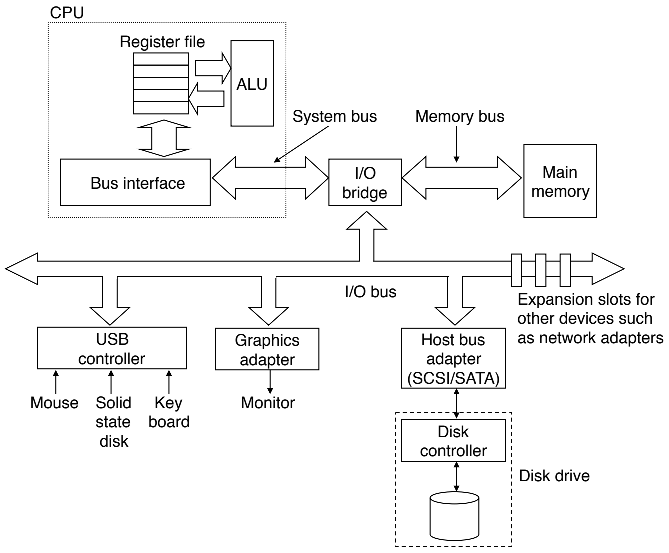


Figure 3: System bus, memory bus, IO bus

- Memory hierarchy & typical visit time: Register (0) - L1 SRAM cache (4) - L2 SRAM cache (10) - L3 SRAM cache (50) - Main DRAM memory (200) - Hard disk (10000000)

1.3 Cache memories

- m-bit address: t-bit tags + s-bit set index + b-bit block offset.
- Direct-mapped cache: $E = 1$. E-way set associative cache: $1 < E < C/B$. Full associative cache: $S = 1$.
- 3 steps: set selection; line matching; word extraction. In case of cache miss: line replacement.
- Cache thrash: cache keeps loading & evicting the same blocks.
- Handle write: write-through (directly write to lower layer) v.s. write-back (put off write until line gets replaced).
- Handle write cache miss: write-allocate (load from lower layer then update) v.s. not-write-allocate (directly write to lower layer).
- Usually: write-through + not-write-allocate; write-back + write-allocate.
- Write cache-friendly code: improve cache hit rate. For example, to write code for matrix multiplication $C = A \times B$, instead of straightforwardly writing:

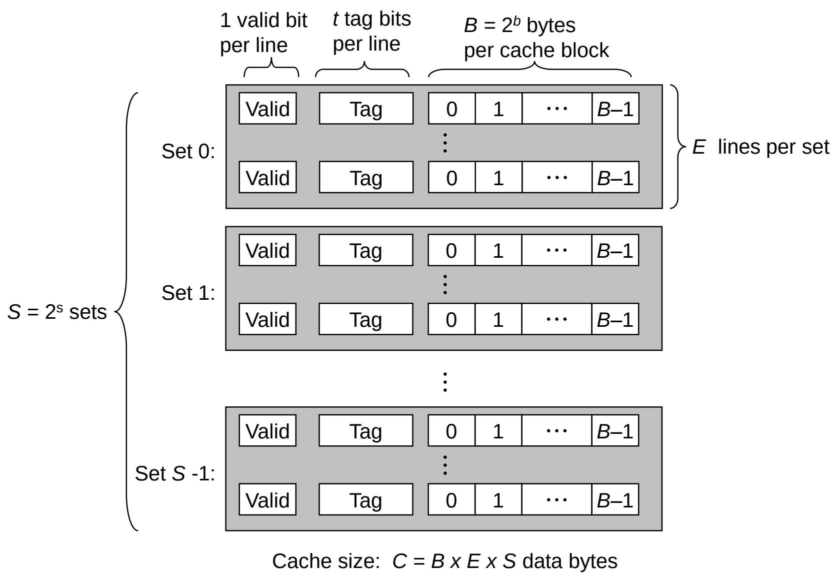


Figure 4: Organization of cache memory(S,E,B,m)

```
//ijk
for(i = 0; i < n; ++i)
  for(j = 0; j < n; ++j) {
    sum = 0.0;
    for(k = 0; k < n; ++k)
      sum += A[i][k] * B[k][j];
    C[i][j] = sum;
  }
```

```
//kji
for(k = 0; k < n; ++k)
  for(j = 0; j < n; ++j) {
    r = B[k][j];
    for(i = 0; i < n; ++i)
      C[i][j] += A[i][k] * r;
  }
```

we should write:

```
//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];
  }
```

Table 1: Analysis of different inner loops

version	load	store	A miss	B miss	C miss	total miss
ijk & jik	2(AB)	0	0.25	1.00	0.00	1.25
kji & jki	2(AC)	1(C)	1.00	0.00	1.00	2.00
ikj & kij	2(AC)	1(C)	0.00	0.25	0.25	0.50