TODAY ⇒ Memory (Chapter 6).

- Locality
- Hierarchy
- Cache

So far ⇒ simplistic memory

- Assumption
  - All memory accesses happen in a constant amount of time.

movl 4(-1-ebp),-eax
Physically - separate

Faster!

Drawback
- small ✓
- expensive ✓

We have different levels of memory
We don't want extremities!
Access times

Register \(\approx 0\) cycles.

Cache \(\approx 1\) to 30 cycles.

Main Memory \(\approx 20\) to 200 cycles.

Disk \(\approx 10's\) of millions of cycles.

The closer your data gets to the processor, the faster your program is going to execute.
Locality.

Programs with good locality:

→ access the same data item repeatedly.

→ access nearby data items repeatedly.

2 forms

1. Temporal Locality
   → a memory location that is referenced once is likely to be referenced again in the near future.

2. Spatial Locality
   → a program is likely to reference a memory location that is near a previously referenced location.
good temporal locality,

Real world example: \rightarrow Browsers!

Twitter, \rightarrow Twitter logo.

Store this locally on some "cache" (Disk)
Instead of fetching from the web every time.
Program with good spatial locality

```c
int a[5];

for (i = 0; i < 5; i++)
    sum += a[i];
```

```
Address: 0 4 8 12 16
Contents: a0 a1 a2 a3 a4
Access order: 1 2 3 4 5
```

```
i = 0
i = 1
i = 2
```

```c
int a[2][3];

for (i = 0; i < 2; i++)
    for (j = 0; j < 3; j++)
        sum += a[i][j];
```

```
i = 0
j = 0
i = 0
j = 1
i = 0
j = 2
```
stride - 1 - reference pattern

stride - 2 - reference pattern

stride - k - reference pattern

lower k is better for spatial locality

Bad spatial locality

for (j = 0; j < 3; j++)
for (i = 0; i < 2; i++)
sum += a[i][j]

3
j = 0 i = 0
j = 0 i = 1
j = 1 i = 0
j = 1 i = 1

Address: 0 4 8 12 16 20
content: a_00 a_01 a_02 a_10 a_11 a_12
Access: 1 3 5 2 4 6
Order: 1 3 5 2 4 6
Sum up

→ repeatedly access the same variables ⇒ good temporal locality

→ sequential reference pattern ⇒ good spatial locality
    (stride-1-pattern)

CACHING

Generally a cache is a smaller, faster storage device that acts as a staging area for data stored in a larger, slower device.

The process of using a cache ⇒ caching.
Cache Hit

Say a program needs data object d from level k+1.

First → look in level k. //

If d happens to be cached at level k itself, we have a cache hit.
and we get d from k.

The more cache hits, the merrier!

Why? Accessing data from level k is faster than accessing from level k+1.

Cache miss.

What if d is not in level k?

We have a cache miss!
level $k$

Data transfer in block-sized units

level $k+1$

Block size is fixed (but only between two levels).
It can be different for other level pairs.
Then → level k fetches d from level k+1 and it will store it.

What if k is full? Remove some old block and put d there.

Cache Replacement Policy
- Random
- Least Recently Used Policy

Block to be replaced by victim block.

Arrays vs Linked List
Suppose → linear search:

good spatial locality?