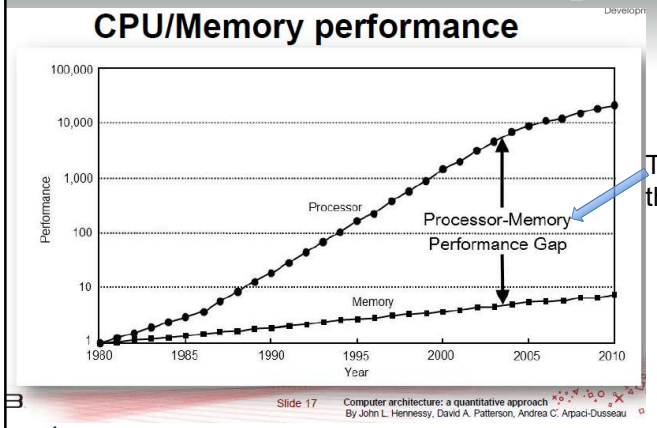
Cache in the modern computer

cache for cash

what's the problem



This is the problem

SRAM vs DRAM

Static Ram (transistor based) is low power, no refresh, fast, expensive

Dynamic Ram (capacitor based) is high power, refresh, slow, cheap

Can't afford to use too much SRAM, so we have the gap.

The modern problem

Need to keep the CPU busy, need to get data to it and keep the pipes flowing.

We need to find a way to speed up slow memory.

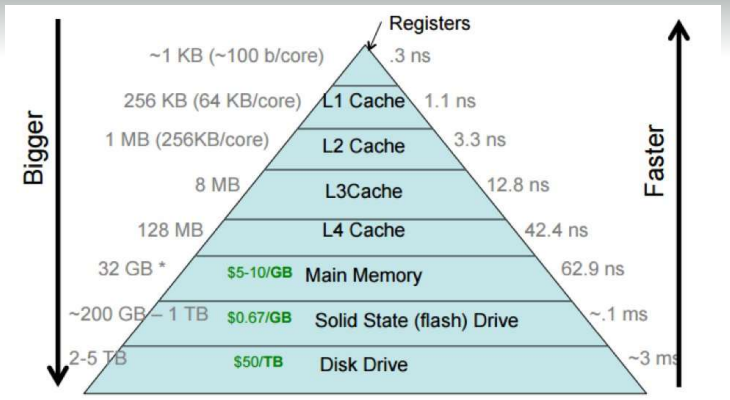
Solution is Cache

One of the predominant themes in modern architectures is the use of cache:

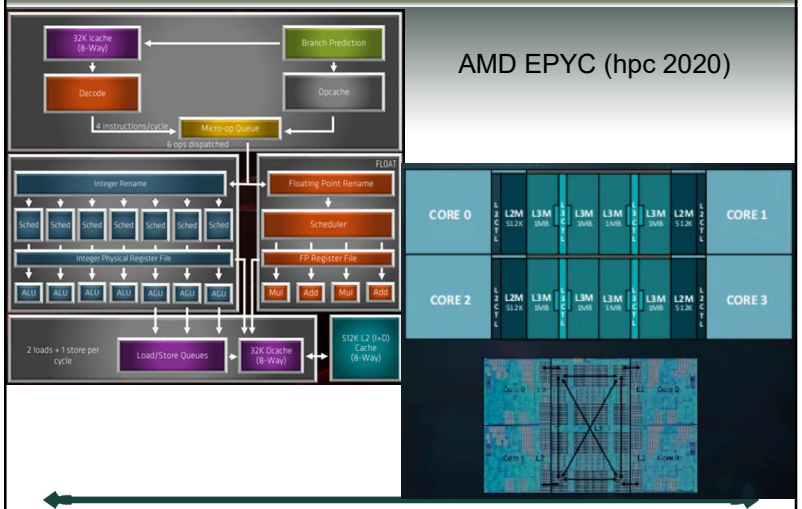
- small, high-speed, storage area that acts as a kind of buffer for a bigger, slower storage area

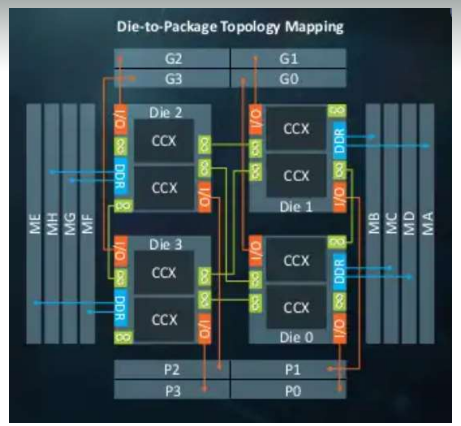
Cache is used a lot!

1 clock ~ 0.3ns



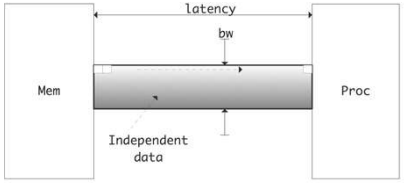
AMD EPYC (hpc 2020)





Memory hierarchy

Need to feed the beast (er, CPU)



Little's Law: Concurrency = Bandwidth x Latency

Locality, Locality, Locality

Like real estate, what makes cache useful is locality:

- temporal locality: use again what you just used (L1 instruction)
- spatial locality: use something "near" what you just used (L1 data)

To the extent that locality holds, cache solves the memory gap.

hit or miss

The process is:

- look for data in the closest place in order (register, L1, L2, L3, memory).
- If there, that is a ***cache hit***
- If not there, that is a ***cache miss*** and we move up the hierarchy looking
- Once we find it, we move it close (probably the L1) in hopes of locality.

What do we move

When we move data to the L1, what do we move?

- ***cache line***, a set of contiguous data, probably 64 bytes at a time.
  - why contiguous???

Kinds of misses

- Compulsory cache miss: first time memory is referenced
- Capacity cache miss: cache not big enough to fit problem
- Conflict cache miss: data mapped to same cache location as another
- Invalidation cache miss: another core changed value at memory address

Writes are a problem

What to do when the CPU writes:

- to the cache? OK as long as the single CPU is the only user.
  - if multiple CPUs are addressing the same memory, then one write ***invalidates*** everybody's cache line.
  - ***Cache coherency***
- to memory? That sucks, way too slow

False sharing

When multiple PE (processing elements) access different parts of the same line:

- line is needlesslv invalidated

```
static struct {
    int x;
    int y;
} f;

/* The two following functions run concurrently: */

int read_x(void) {
    return f.x;
}

void write_y(void) {
    ++f.y;
}
```

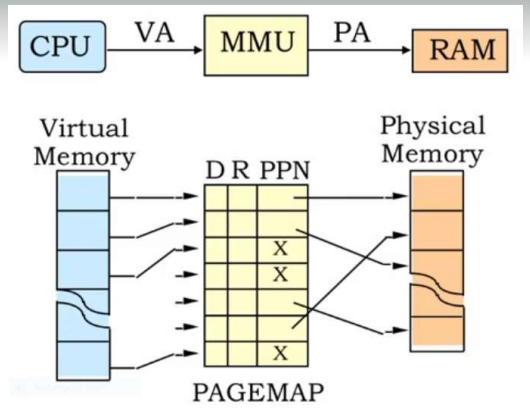

Virtual Memory

The high level details are:

- every process views memory as a flat, contiguous space.
- however, where that memory space is in actually, physical located in memory is controlled by OS/hardware

Multiple processes have their "own" memory, dispersed on physical memory

VA to PA



MMU

The memory management unit (MMU) does the VA → PA translation.

- allocated in "pages"
  - heaven help you, but page could be on disk
- translation can involve a lot of overhead

Hey, we need a cache!

Translation Lookaside Buffer

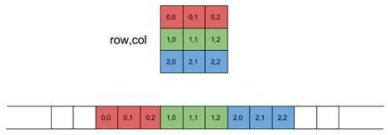
The TLB (what a great name) is a cache for translating VA → PA

- just like any other cache it has a limited size
- is much faster

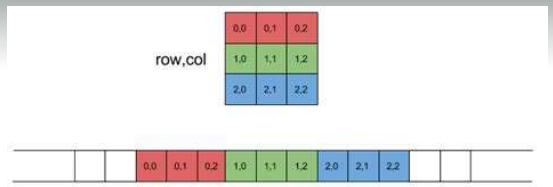
arrays in memory

This is kind of an aside, but how is an array laid out in memory (even mutli-dimenstional)

- C is "row major" order



The index calculation



A[row][col] → A[row*width + col]