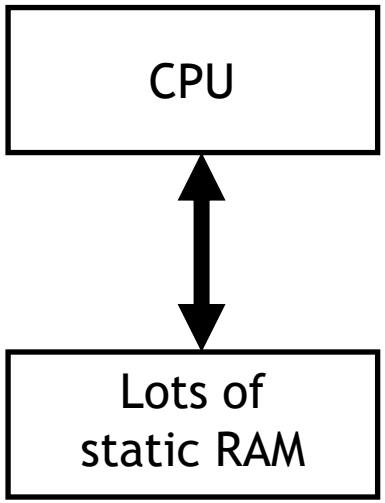
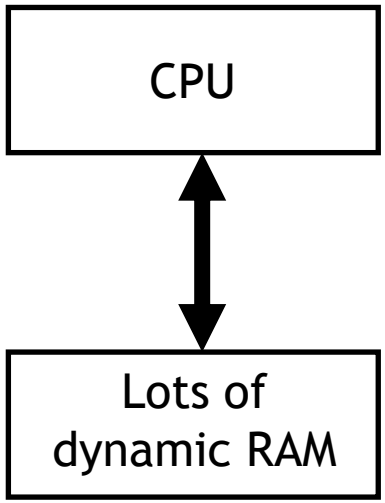
Memory and I/O

- So far we've been talking mostly about processor design, but there's a lot more to computers than just the CPU.
  - Memory and peripheral devices give us somewhere to store data.
  - Buses connect processors, memory and peripherals together.
- These affect overall system speed just as much as the processor. A 3 GHz CPU can't go very fast if it always has to wait for a 56 kbps modem!

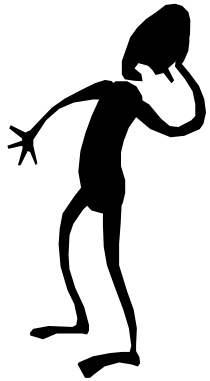
Memory

- Recall the memory tradeoff we mentioned several weeks ago.
  - Static memory is very fast, but also very expensive.
  - Dynamic memory is relatively slow, but much cheaper.

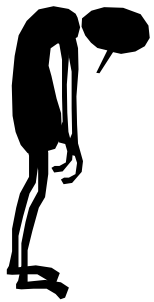




Expensive!

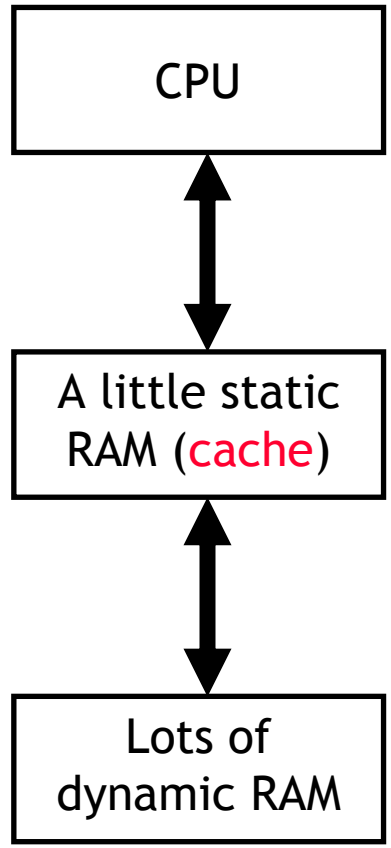


Slow!



Introducing caches

- Wouldn't it be nice if we could find a balance between fast and cheap memory?
- We do this by introducing a cache, which is a small amount of fast, expensive memory.
  - The cache goes between the processor and the slower, dynamic main memory.
  - It keeps a copy of the most frequently used data from the main memory.
- Memory access speed increases overall, because we've made the common case faster.
  - Reads and writes to the most frequently used addresses will be serviced by the cache.
  - We only need to access the slower main memory for less frequently used data.



The principle of locality

- It's usually difficult or impossible to figure out what data will be "most frequently accessed" before a program actually runs, which makes it hard to know what to store into the small, precious cache memory.
- But in practice, most programs exhibit *locality*, which the cache can take advantage of.
  - The principle of temporal locality says that if a program accesses one memory address, there is a good chance that it will access the same address again.
  - The principle of spatial locality says that if a program accesses one memory address, there is a good chance that it will also access other nearby addresses.

Temporal locality in programs

- The principle of temporal locality says that if a program accesses one memory address, there is a good chance that it will access the same address again.
- Loops are excellent examples of temporal locality in programs.
  - The loop body, in blue, will be executed many times.
  - The computer will need to access those same few locations of the instruction memory over and over.

```
        LD   R1, #0          // R1 = 0
        LD   R2, #1          // R2 = 1
FOR     BGT  R2, #5, L       // Stop when R2 > 5
        ADD  R1, R1, R2      // R1 = R1 + R2
        ADD  R2, R2, #1      // R2++
        JMP  FOR             // Go back to the loop test
L       ADD  R3, R1, R1      // R3 = R1 + R1
```

Temporal locality in data

- Programs often access the same variables over and over, especially within loops. Below, sum and i are repeatedly read and written.

```
sum = 0;
for (i = 0; i < MAX; i++)
    sum = sum + f(i);
```

- Commonly-accessed variables can sometimes be kept in registers, but this is not always possible.
  - There are a limited number of registers.
  - There are situations where the data must be kept in memory, as is the case with shared or dynamically-allocated memory.

Spatial locality in programs

- The principle of spatial locality says that if a program accesses one memory address, there is a good chance that it will also access other nearby addresses.

```
LD   R1, A
LD   R2, B
ADD  R3, R1, R2
LD   R1, C
LD   R2, D
ADD  R1, R1, R2
MUL  R1, R1, R3
ST   X, R1
```

- Nearly every program exhibits spatial locality, because instructions are usually executed in sequence—if we execute an instruction at memory location *i*, then we will probably also execute the next instruction, at memory location *i+1*.
- Code fragments such as loops exhibit *both* temporal and spatial locality.

Spatial locality in data

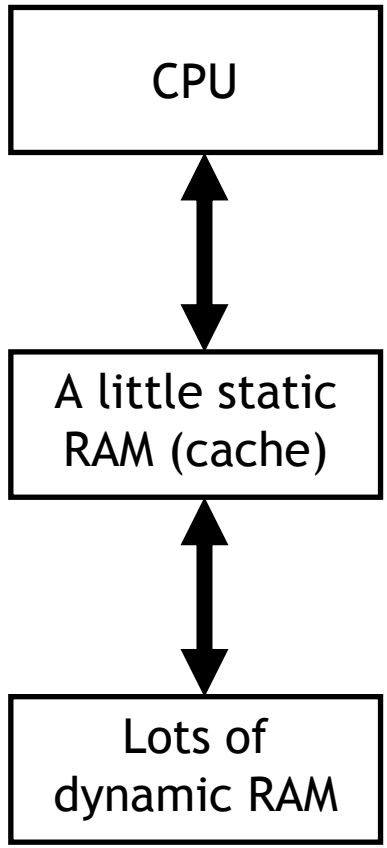
- Programs often access data that is stored contiguously.
  - Arrays, like a in the code on the top, are stored in memory contiguously.
  - The individual fields of a record or object like employee are also kept contiguously in memory.
- Data accesses may also exhibit both temporal and spatial locality, as with the array a in the first example here.

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

```
employee.name = "Homer Simpson";
employee.boss = "Mr. Burns";
employee.age = 45;
```

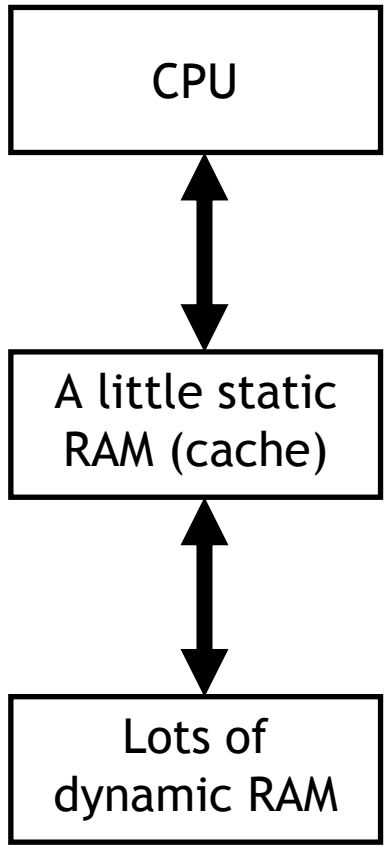
How caches take advantage of temporal locality

- The first time the processor reads from an address in main memory, a copy of that data is also stored in the cache.
  - The next time that same address is read, we can use the copy of the data in the cache *instead* of accessing the slower dynamic memory.
  - So the first read is a little slower than before since it goes through both main memory and the cache, but subsequent reads are much faster.
- This takes advantage of temporal locality—commonly accessed data is stored in the faster cache memory.



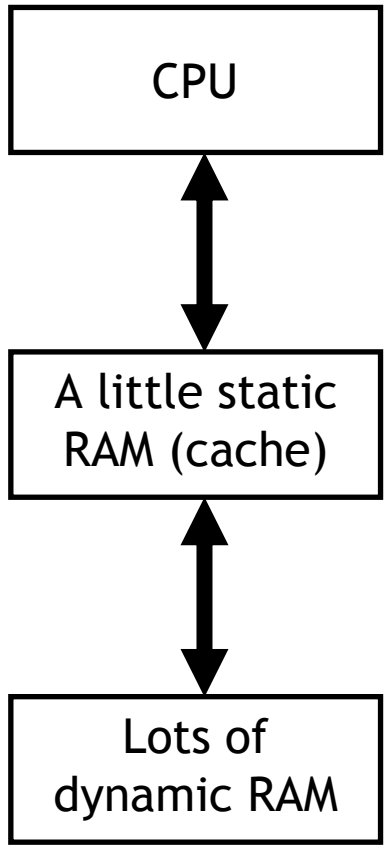
How caches take advantage of spatial locality

- When the CPU reads location *i* from main memory, a copy of that data is placed in the cache.
- But instead of just copying the contents of location *i*, we can copy *several* values into the cache at once, such as the four bytes from locations *i* through *i* + 3.
  - If the CPU later does need to read from locations *i* + 1, *i* + 2 or *i* + 3, it can access that data from the cache and not the slower main memory.
  - For example, instead of reading just one array element at a time, the cache might actually be loading four array elements at once.
- Again, the initial load incurs a performance penalty, but we're gambling on spatial locality and the chance that the CPU will need the extra data.



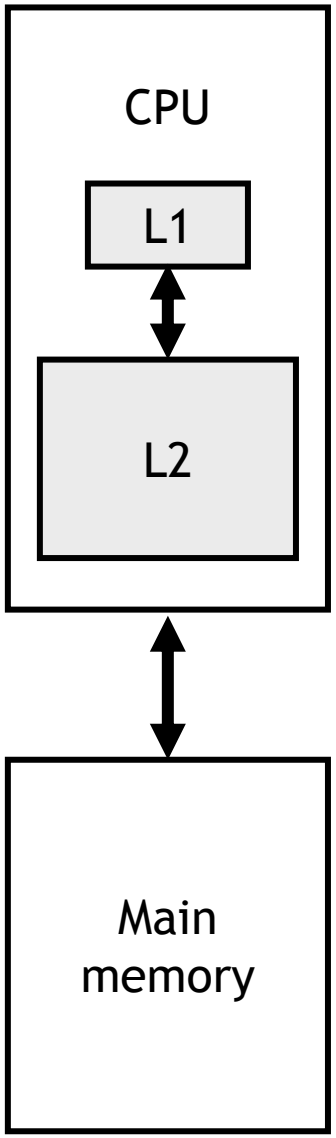
The devil is in the details

- Actually implementing a cache is tricky.
  - When we copy data from main memory, exactly where should we put it within the cache?
  - The cache is much smaller than main memory, so what happens if the cache fills up?
  - Now we could have two copies of the same data, one in the cache and one in RAM. If we modify the data, we have to modify it in both places!
- It's difficult to accurately predict the performance benefits of caching, since all programs have different memory access patterns.
- Many cache comparisons are done using benchmarks. To find out how the speed of two systems compares, just run some programs and see how long they take.



Modern caches

- Modern processors take the concept of caching one step further, and include at least two levels of caches.
  - The primary or Level 1 cache is smallest but fastest.
  - The secondary or Level 2 cache is larger and stores more data, but it's slower.
  - The main memory is the largest and slowest of all.
- These caches are actually included on the same chip as the CPU itself, to decrease the cache access delays.
- New CPUs try to provide as much cache as possible, but they are still only a fraction of the size of typical 256MB to 512MB main memories.

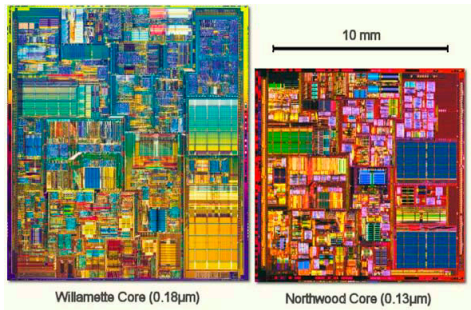


Processor	L1 cache	L2 cache
Intel Pentium 4	8+ KB	512 KB
AMD Athlon XP	128 KB	512 KB
Motorola PowerPC G4	64 KB	256 KB

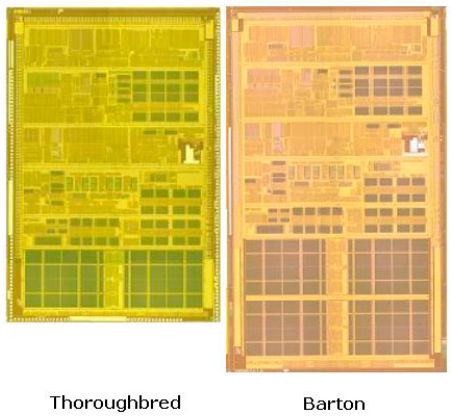
Caches and dies

- As manufacturing technology improves, designers can squeeze more and more cache memory onto a processor die.
- In the old Pentium III days the L2 cache wasn't even on the same chip as the processor itself. Companies sold processor "modules" that were composed of several chips internally.
- The second picture illustrates the size difference between an older Pentium 4 and a newer one. The newer one has an area of just 131 mm^2—the size a fingernail!
- The last picture shows the dies of an older Athlon with only 256KB of L2 cache, and a newer version with 512KB of L2 cache. You can literally see the doubling of the cache memory.

Tech Report



Tom's Hardware



Hardware.fr

CPU families

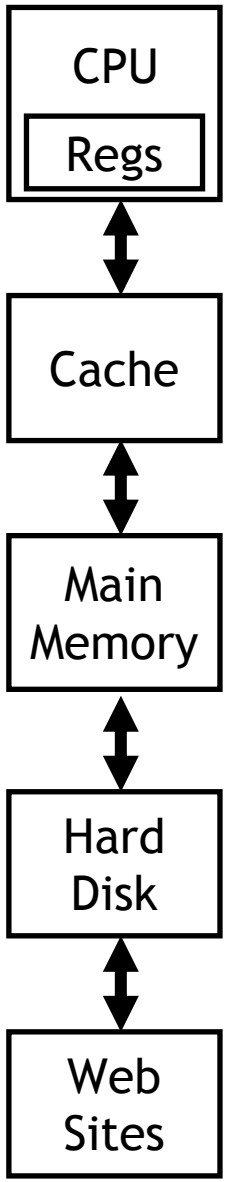
- Manufacturers often sell a range of processors based on the same design.

Company	Budget	Standard	High-end
Intel	Celeron	Pentium 4	Xeon
AMD	Duron	Athlon XP	Opteron

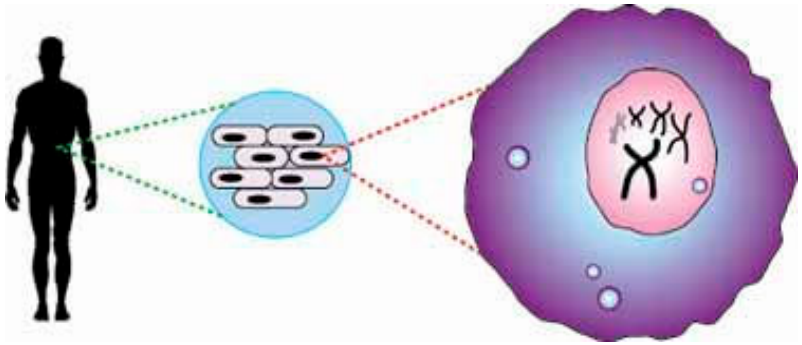
- In all cases, one of the main differences is the size of the caches.
  - The mid-level Athlon XP and Pentium 4 have 512KB of L2 cache.
  - Low-end processors have only half as much or less L2 cache memory.
  - In contrast, the high-end chips feature up to 1 MB of cache.
- This highlights the importance of caches and memory systems for overall system performance.

Extending the hierarchy even further

- This caching hierarchy can be extended quite a ways.
- CPU registers can be considered as small one-word caches which hold the most frequently used data.
- Main memory is a cache for slower hard drives. Programs and data are both loaded into main memory before being executed or edited.
- Hard drives in turn act as caches for network data.
  - Networks have relatively high delays and low transfer rates, so minimizing the number and size of transfers is desirable.
  - For example, web browsers store your most recently accessed web pages on your hard disk.
  - Administrators can install network-wide web caches like CCSO's CacheFlow, and companies like Akamai also provide caching services.



I/O is important!



- Many tasks involve reading and processing enormous quantities of data.
  - CCSO has two machines and 144GB of storage for a local web cache.
  - Institutions like banks and airlines have huge databases that must be constantly accessed and updated.
  - Celera Genomics is a company that sequences genomes, with the help of computers and 100 trillion bytes of storage!
- I/O is important for us small people too!
  - People use home computers to edit movies and music.
  - Large software packages may come on multiple compact discs.
  - Everybody and their grandparents surf the web!

I/O is slow!

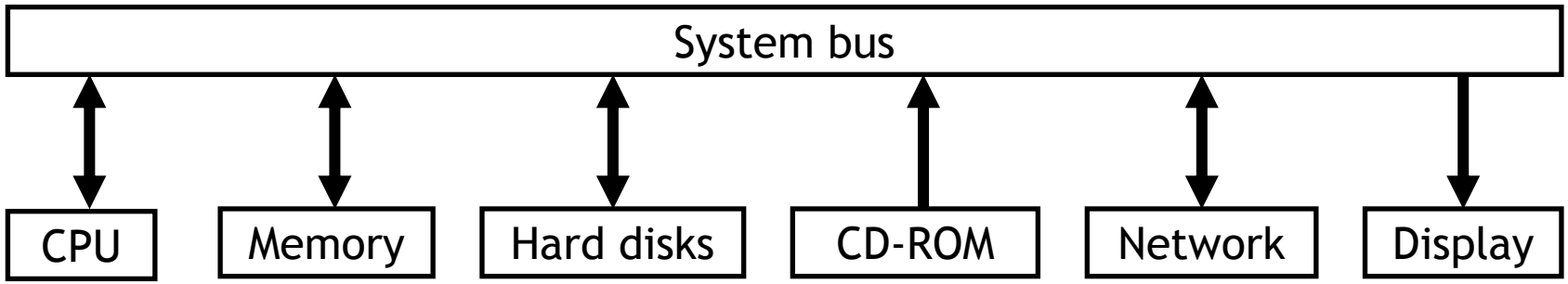
- How fast can a typical I/O device supply data to a computer?
  - A fast typist can enter 9-10 characters a second on a keyboard.
  - Common local-area network speeds go up to 100 Mbit/s, or 12.5 MB/s.
  - Today's hard disks provide transfer speeds around 40 MB per second.
- Unfortunately, this is excruciatingly slow compared to modern processors that can execute over a billion instructions per second!
  - I/O performance has not increased as quickly as CPU speeds, partially due to neglect and partially to physical limitations.
  - This is slowly changing, with faster networks, better I/O buses, RAID drive arrays, and other new technologies.

Measuring I/O performance

- There are three general performance measurements for I/O systems.
  - An application that accesses large quantities of data will demand high bandwidth or transfer speed.
  - Programs that access small amounts of data in frequent intervals may care more about the latency or delay.
  - Throughput, which is the number of transactions performed per unit time, accounts for latency, bandwidth and overhead times.
- Home network users can be affected by both bandwidth and latency.
  - If you download large files over Kazaa, bandwidth will be the limiting factor; a 4Mbit/s DSL line will outperform a 56Kbit/s modem.
  - If you send short instant messages, the latency becomes the limiting factor—it takes a long time for data to reach Swaziland.

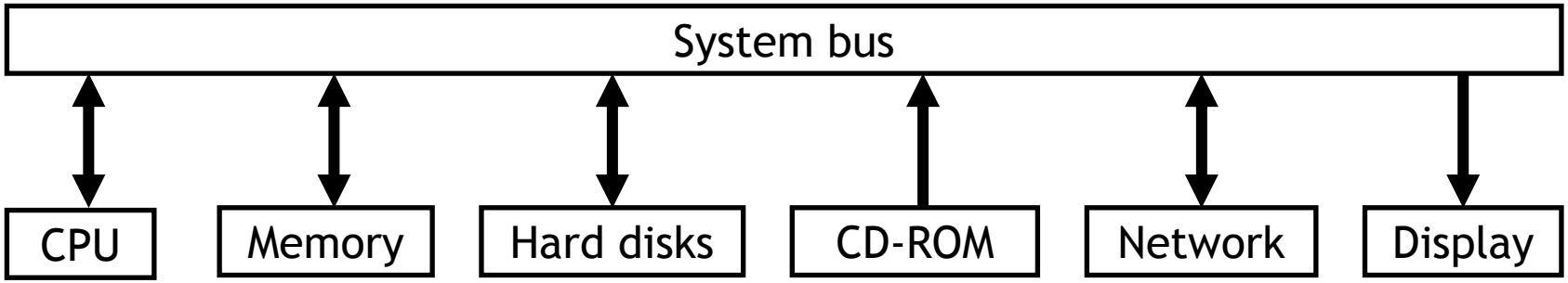
Computer buses

- Each computer has several shared pathways, called buses, that connect processors, memory, and I/O devices.
- The simplest kind of bus is linear, as shown below.
  - All devices share the same bus.
  - Only one device at a time may transfer data on the bus.



- You've already seen lower-level buses that connect register files, RAMs and ALUs together inside a processor.
- Buses are also similar to networks that connect entire systems together.

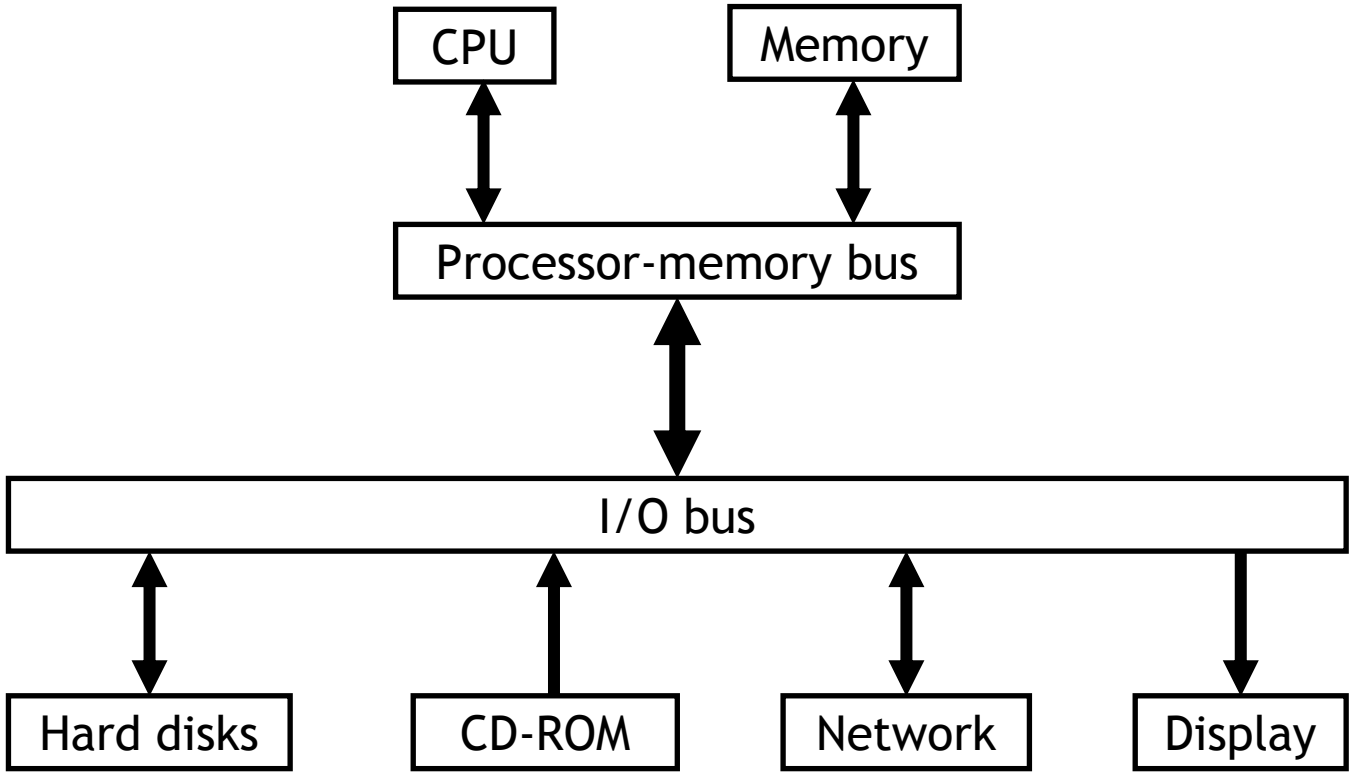
Scalability



- Unfortunately, this bus is not very expandable or scalable.
- The more devices there are, the more contention there will be. If several devices try to send data on the bus simultaneously, only one will succeed while the others must wait.
- Also, the more devices you connect, the longer the bus will have to be.
  - It will take more time for signals to propagate.
  - Keeping signals synchronized across long distances is hard.
  - You also need to keep the signals from degrading.

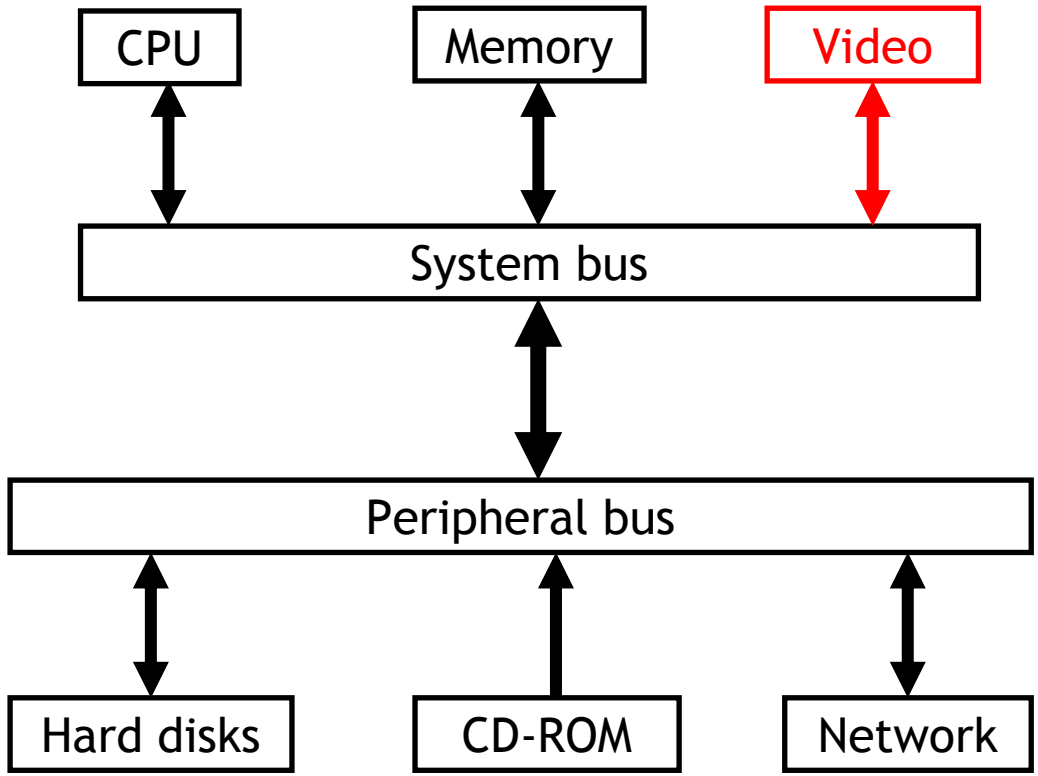
Hierarchical buses

- Splitting the bus into different segments can help.
  - Since the CPU and memory need to communicate so often, a shorter and faster processor-memory bus can be dedicated to them.
  - A separate I/O bus would connect the slower devices to each other, and eventually to the processor.



A word about graphics

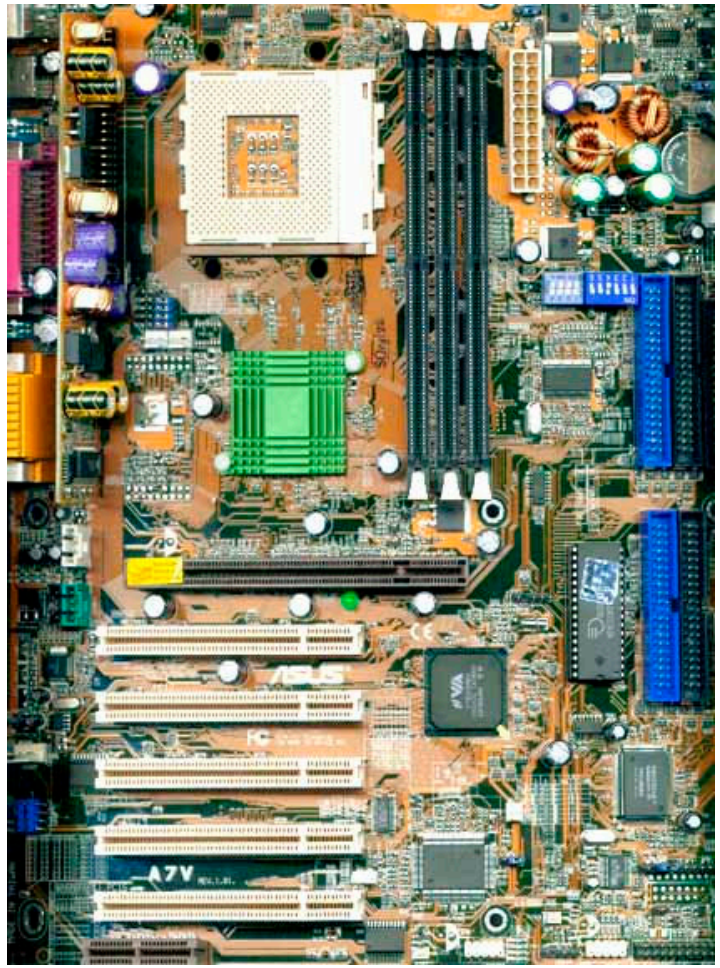
- Video displays are one of the most-frequently used devices in systems, especially with the advent of graphical user interfaces, games, movies and animations.
- Most newer machines place the video card on the main system bus for faster access to the CPU and memory.



Open up and say ahhh

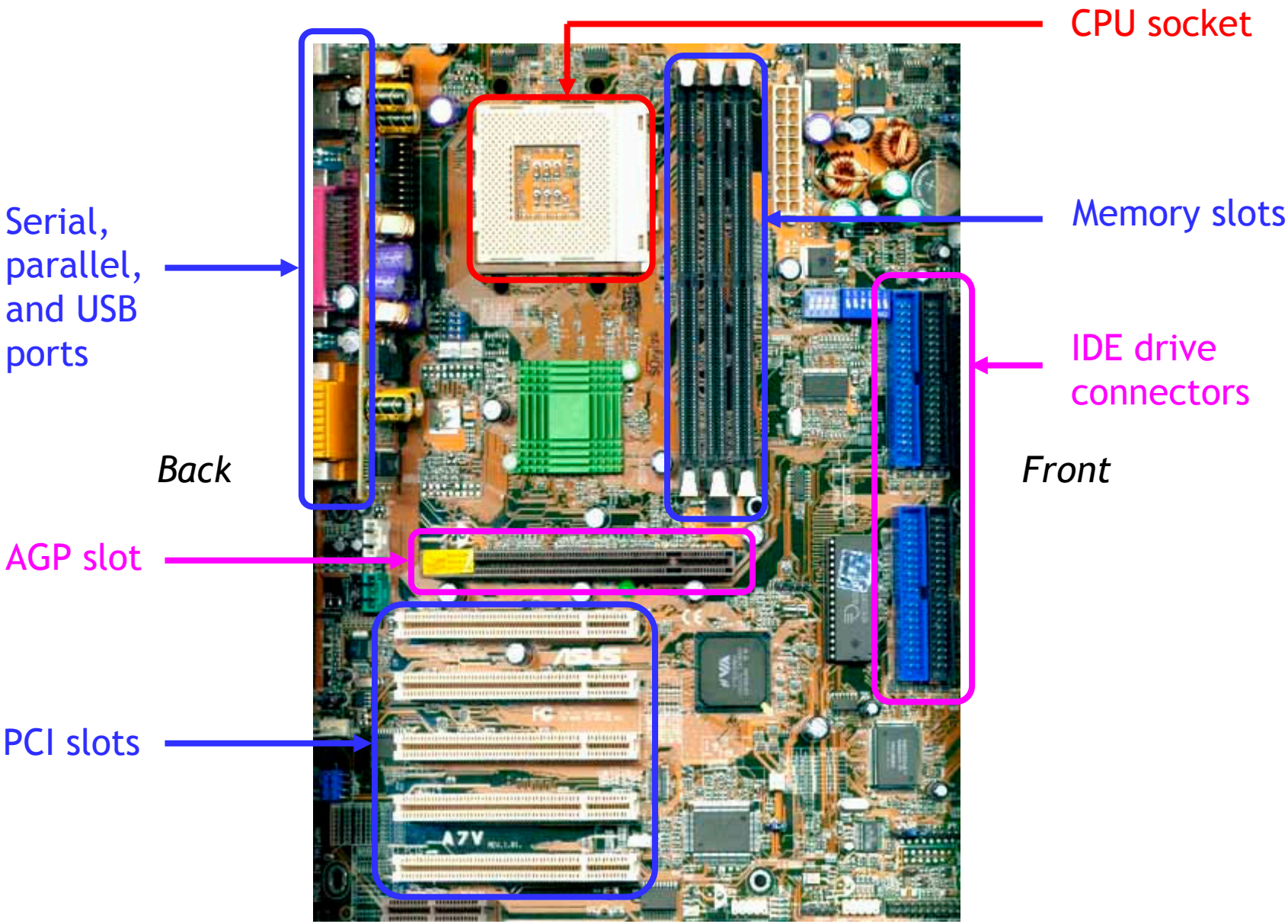
- If you open up your computer, you'll find the motherboard, which connects everything together.

Hypersonic PC



Tom's Hardware

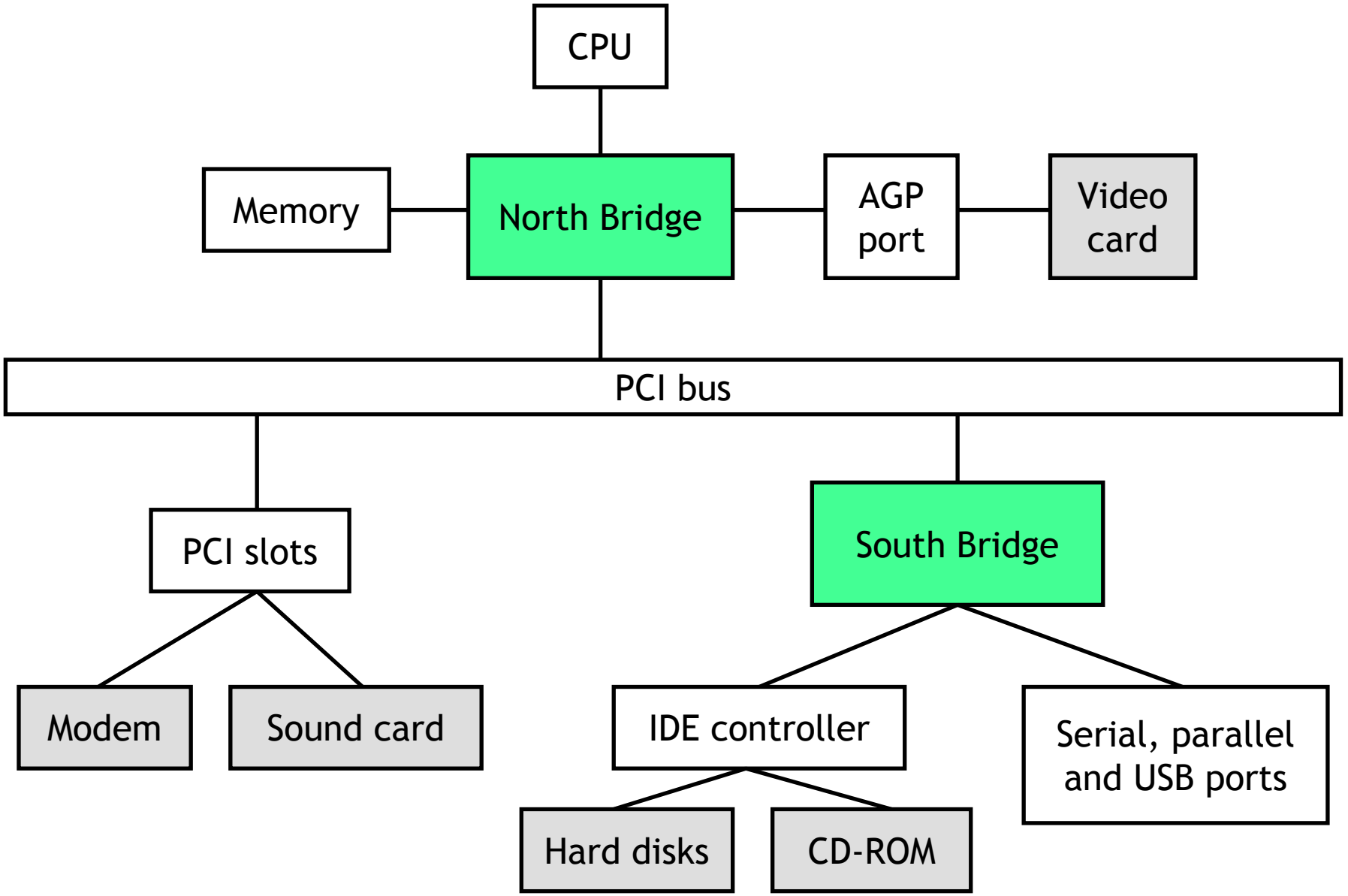
The mothership



What is all that stuff?

- Different motherboards support different CPUs, types of memories, and expansion options.
- This picture is an old Asus A7V motherboard (all I can afford).
  - The CPU socket supports AMD Duron and Athlon processors.
  - There are three DIMM slots for standard PC100 memory. Using 512MB DIMM modules, you can get up to 1.5GB of main memory.
  - The AGP slot is just for video cards.
  - IDE ports connect internal storage devices like hard disks or CD-ROMs.
  - PCI slots hold other internal devices such as network and sound cards.
  - Serial, parallel and USB ports are used to attach external devices such as scanners and printers.

How is it all connected?



Summary

- Memory and peripheral bus speed are as important as processor speed in determining overall system performance.
- Caches use a little static RAM to dramatically speed up memory accesses.
  - Most programs exhibit locality, meaning they are likely to access the same or nearby memory locations in the near future.
  - By keeping a copy of recently accessed data and nearby locations in the faster cache RAM, we can typically make future accesses faster.
- Buses connect the processor, memory and peripheral devices together.
  - Buses and peripherals are usually much slower than the CPU itself.
  - Contention between devices is a big problem that can sometimes be solved with hierarchical buses.