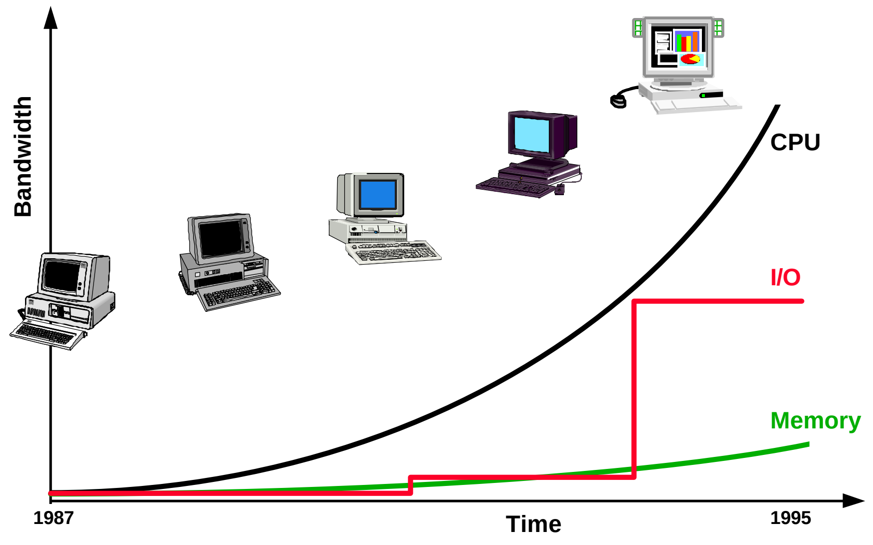
The PC Heritage



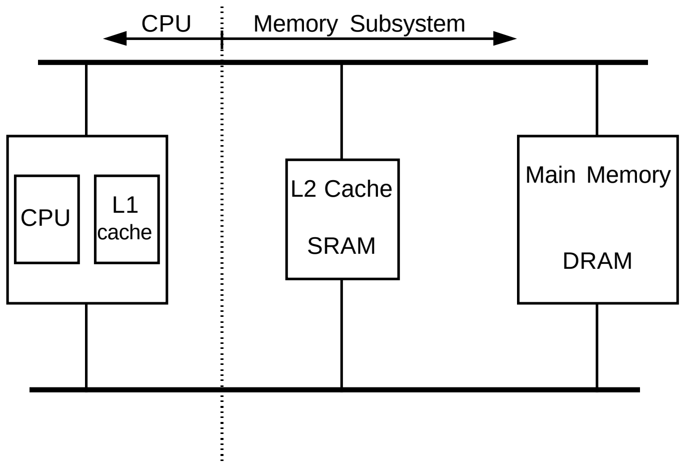
The basic desktop PC has looked pretty constant on the outside for the past 10 to 15 years. But inside, the PC technology has been moving fast and this pace is accelerating.

Microprocessor technology has fueled the growth and CPU performance has, over the last ten years, increased 400 fold. Of course, we want to deliver this increased performance directly to the user's eyes, hands and more recently ears. We want CPU performance to translate directly into system performance. Just increasing the performance of the CPU creates bottlenecks in the system design. These bottlenecks need to be removed so that we can unleash the latest capability within the CPU.

Over the past 1-2 years, the PC industry adopted the PCI IO subsystem to relieve the IO bottleneck.

Increased Pentium processor performance is revealing the next system bottleneck -- the memory subsystem.

The next system bottleneck?



Long Memory Latency Slows System Performance

Imagine the Pentium® processor is executing a program and needs to load a variable that is not in it’s L1 cache. It needs to access the memory subsystem. Execution of this load instruction will stall while the Pentium processor is waiting for this data. If the data is not in the L2 cache (or the L2 cache does not exist) the Pentium processor must wait until the data is loaded from main memory. When measured in CPU clocks, this is a long time.

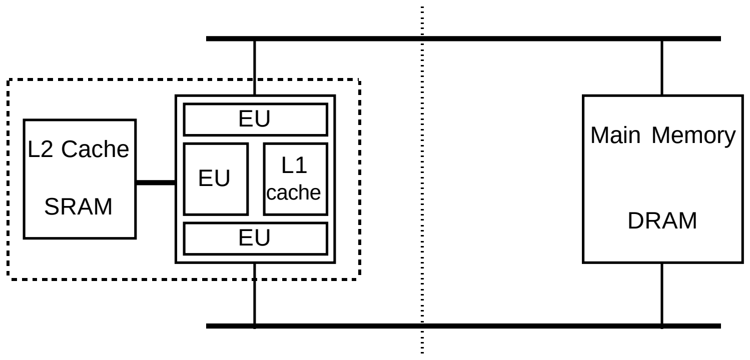
As the speed of the CPU increases relative to the main memory, the CPU’s percentage performance loss when accessing main memory is increased.

One solution is to increase the size and speed of the L2 cache. While effective, this is an expensive way to solve the problem.

Another solution is to increase the speed of main memory. The DRAM industry has been focused on increased density and not on speed. Yes, there is some work on specialist DRAMs, but these are very expensive, too.

Let’s look at the CPU and use it to creatively solve our dilemma.

The P6 Architecture Difference



- P6 designed to address key system challenges
  - Uses Dynamic Execution technology
  - Reduces impact of slow memory subsystem
  - Integrates a non-blocking L2 cache
- Results in spare CPU bus bandwidth for MP and/or IO

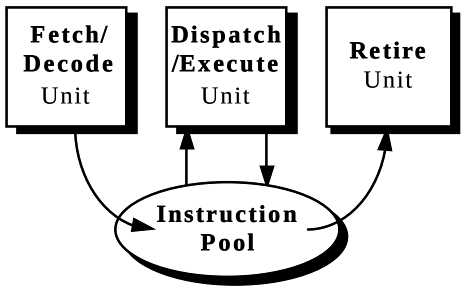
The P6's primary system challenge is to *increase system performance* and do it with *today's commodity memory technology*.

The P6 uses Dynamic Execution technology to remove the performance impacts of the relatively slow main memory subsystem. We shall see in a few slides that the P6 finds other useful work to do rather than just wait for the memory subsystem to respond. This is more complicated for the microprocessor, and we shall see that the P6 is actually implemented using three execution units. It is **much** easier, however, for the PC platform to absorb. Rather than encourage a change in the PC platform, the P6 was designed to provide even higher performance using the same PC hardware components. Any increase in complexity in the system has been absorbed by the P6 processor.

We have integrated a 256K non-blocking L2 cache into the CPU package in the first P6, and this cache is connected via its own private bus. This results in only cache-to-memory and IO traffic on the CPU bus. Even with the most stressful of benchmarks that we have simulated, the P6 uses less than 25% of its external bus bandwidth. This will give us scope to add more CPUs or more IO without dramatically affecting the PC hardware paradigm.

Dynamic Execution

- P6 implemented as three independent units



- Dynamic Execution is the unique combination of:

– Multiple Branch Prediction	- *get many instructions*
– Dataflow Analysis	- *decide best instructions to execute*
– Speculative Execution	- *execute instructions in preferred order*

The internal architecture of the P6 is very different from that of the Pentium® processor since it tackles the task of executing programs in a fundamentally different way.

The Pentium processor treats a program as a linear sequence of instructions. These instructions are presented to the execution unit two at a time (the Pentium processor has a level 2 superscalar engine) and they are executed. A data read that misses the cache will stall execution and the Pentium processor will wait.

The P6 opens up a wide instruction window using an instruction pool and it divides the task of executing the program among three independent units. The "fetch/decode" unit is responsible for putting instructions into the pool. To do this it fetches from the current instruction pointer and analyzes the instructions as they are decoded. If a conditional branch is encountered, its destination is predicted and program fetch continues from there. The P6 predicts the direction of these branches correctly >90% of the time even when these branches are multiple levels deep.

The dispatch/execute unit is looking into the instruction pool for work to do. When it finds an instruction that has all of its operands ready, then it dispatches it to an execution unit. The P6 uses Dataflow Analysis to preferably choose those instructions that minimize the overall execution time of the program. The results are later returned to the instruction pool. Note that instructions execute depending upon their readiness to execute and not on original program order. In this way a "slow" instruction will not block a "fast" instruction. The process is called speculative execution since we do not commit the results to permanent machine state (i.e. real registers and real memory variables) at this time -- that is, we may have predicted incorrectly during the fetch/decode stage and need to be able to "unravel" our work.

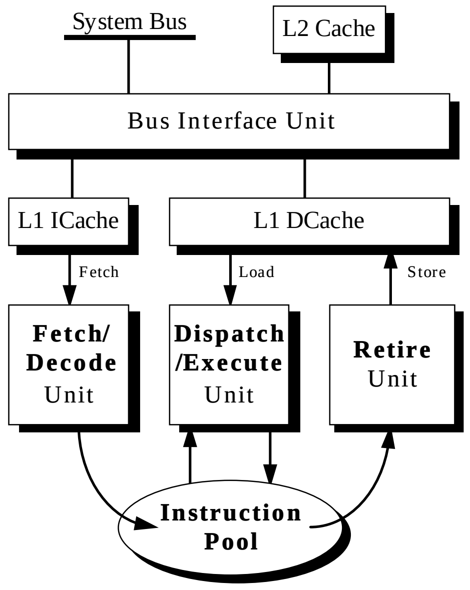
The retirement unit is looking into the instruction pool for instructions that have completed. It commits the results to permanent machine state by retiring these instructions in the original program order.

The fetch/decode unit operates in-order and can deliver up to three instructions into the pool in each clock. The dispatch/execute unit operates out-of-order and can process up to five instructions per clock although three is typical. The retirement unit can complete up to three instructions per clock. We classify the P6 as a Level 3 Superscalar engine.

Non Blocking Caches

- First P6 has integrated 256K L2 cache
- P6 has 8K/8K data/ instruction L1 caches
- All caches are non-blocking

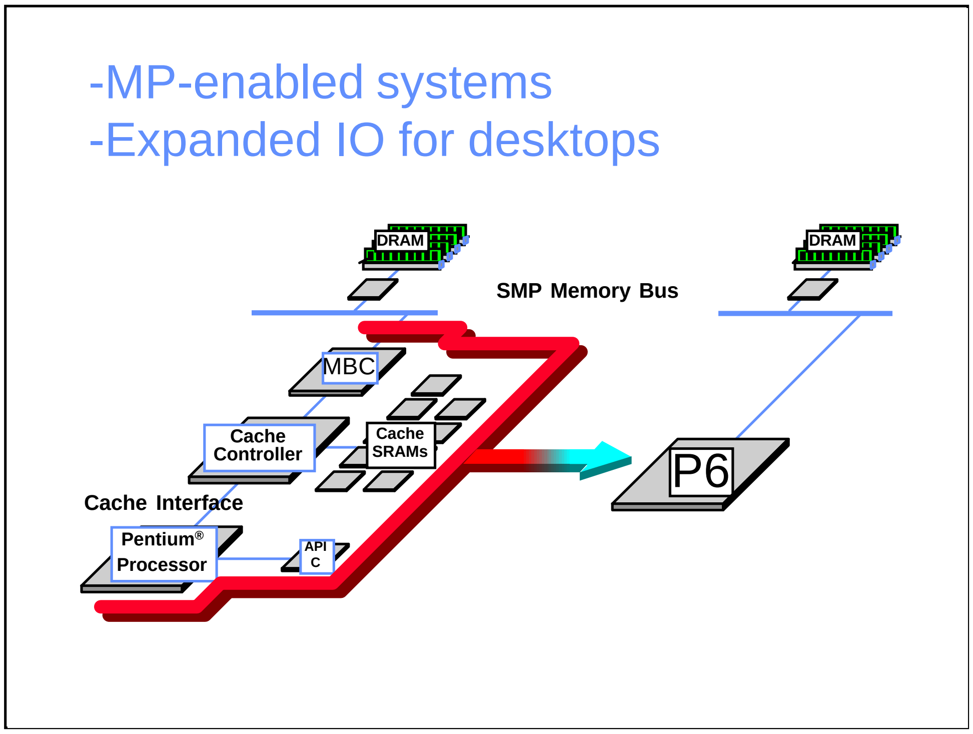
Cache miss does NOT stall the P6 engine



The L1 caches were designed to be non-blocking and since suitable, commercial SRAM was not available we designed and implemented our own non-blocking L2 cache too. The L2 cache controller is on the P6 CPU die and the tags and data array are implemented on a companion die. The caches are dual ported and can support a load and store on each clock. These operations are pipelined so a load and store can be started on every clock.

The internal CPU engine does not stall on a cache miss so the caches are called non-blocking. The L2 cache has a throughput of 4 clocks so it can support 4 concurrent accesses (one in each stage of its pipeline). If the CPU core issues a fifth load then this is held in a load buffer waiting for the cache. Note that, once again, the CPU is not stalled on this fifth load. In fact, the load buffer can hold up to 12 entries queued for the cache/memory subsystem before the CPU core is requested to stall.

Accesses that miss on the L2 cache are passed to the memory subsystem. The P6 external bus is a transaction bus with separate address request and data response phases. The P6 bus interface unit can support up to 4 outstanding memory subsystem requests for a single P6 and up to 8 outstanding for any P6 bus. The external bus protocol supports retry and defer mechanisms, so in a multiprocessor design with extensive IO, data responses could be returned out-of-order also.



-MP-enabled systems
-Expanded IO for desktops

The P6 core communicates with its L2 cache on a dedicated, private bus. This results in all CPU-to-cache traffic being contained within the P6 package. The external bus is a cache-to-memory bus and is only used to service L2 cache misses and IO requests. The P6 external bus is a standardized, SMP system bus (standardized since it is implemented in silicon). Remember that this external bus is also a transaction bus so there are no wasted cycles between the request and response phases.

The P6 bus interface unit snoops all memory transactions on the P6 external bus using the MESI protocol to keep its internal caches coherent in a multi-processing (MP) environment. The P6 supports cache-based semaphores which will dramatically reduce bus traffic when synchronizing multiple processors.

Even with the most stressful of benchmarks that we have simulated, the P6 external bus is less than 25% utilized. This will allow 4 P6's to be simply connected together to form an MP system. No external glue logic is required, the 4 CPUs are all connected in parallel and they operate in parallel using a standard SMP paradigm.

So the P6 integrates all the system-level components into a "single unit of MP". By integrating the cache controller, cache interface, cache SRAM, and APIC, we have a much simpler Multiprocessing design. A simpler, lower cost design will ensure that more MP machines are introduced to the market at lower prices, driving up the demand for MP applications.

What does this mean for **desktops**? If the system doesn't use 4 CPUs, then the 'spare' bandwidth can be used for IO devices. With 2 CPUs, for example, we still have 50% = 250MB/s of bandwidth which is two maxed out PCI buses. That's four 30 frame/sec full screen videos.

In summary, the internal microarchitecture of the P6 was optimized to provide exceptional performance while using today's memory and IO subsystems. On the way, we preserved enough system bus bandwidth to enable multiple high bandwidth peripherals that are now becoming popular. The P6 gives more performance and will allow the standard PC to support our increased needs on the desktop through the mid and late 1990's.