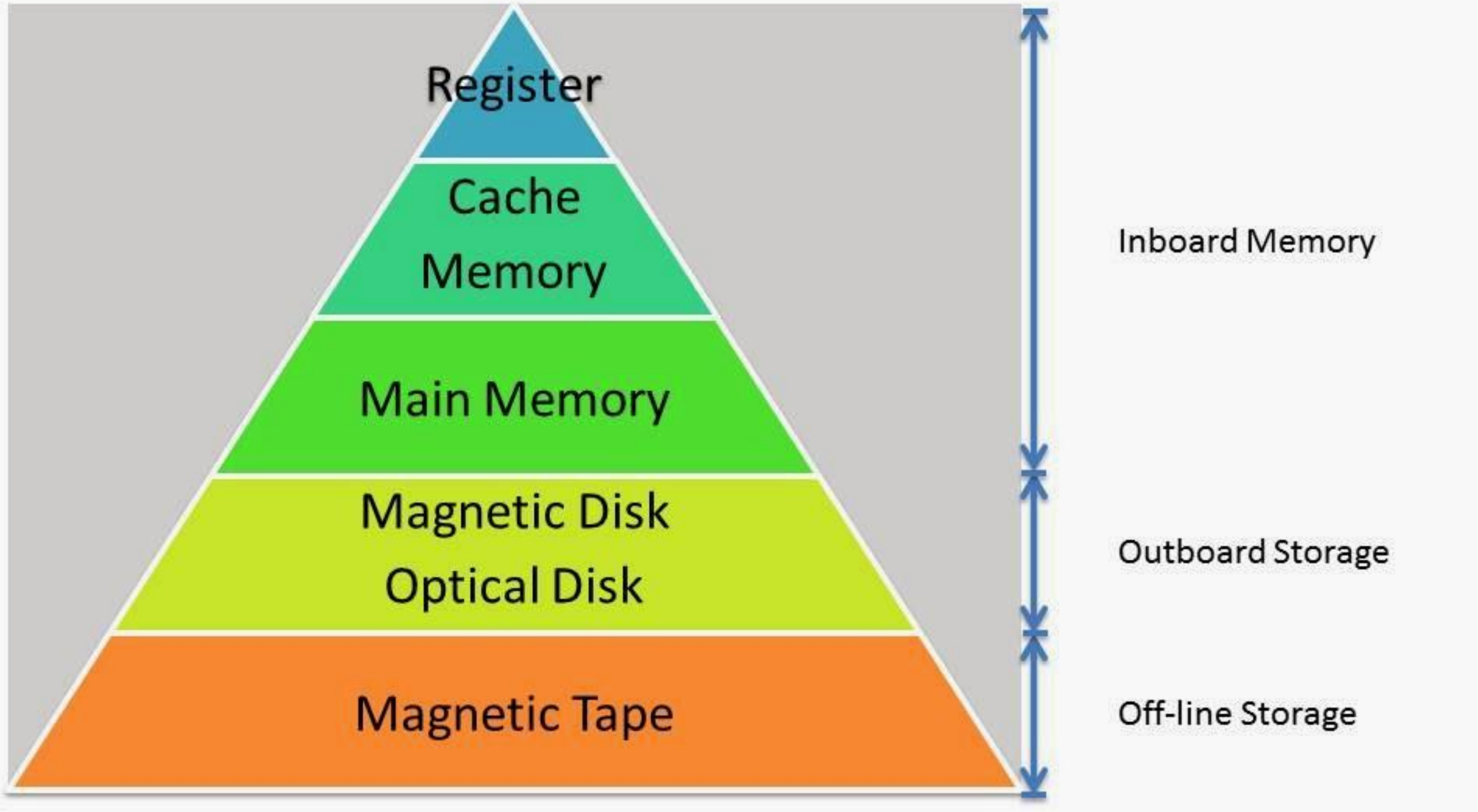
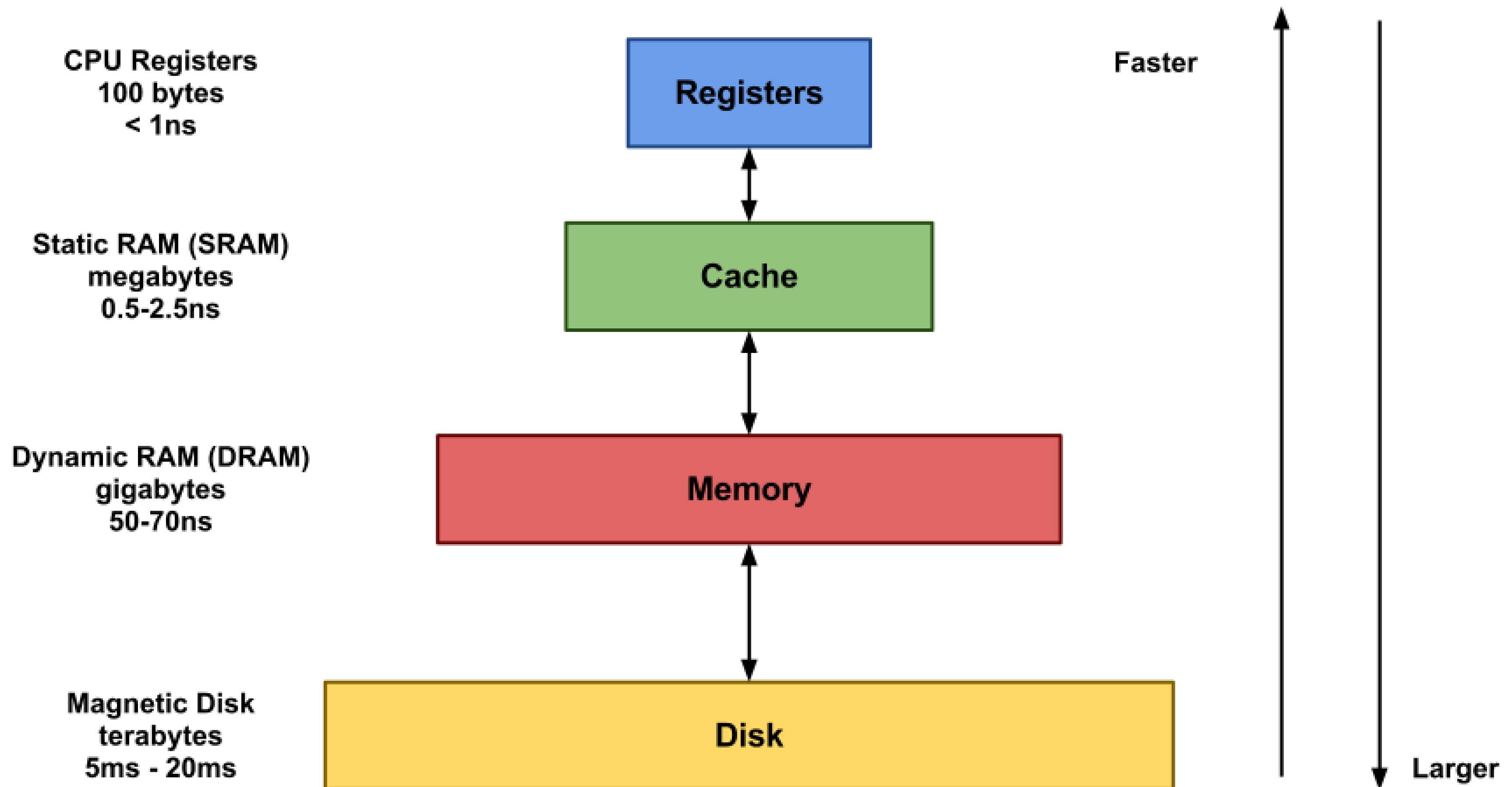
Memory Hierarchy Design

- The solution for need of unlimited amounts of fast memory, is memory hierarchy
- It takes advantage of locality and cost-performance of memory technologies.
- The principle of locality, says that most programs do not access all code or data uniformly
- Locality occurs in time (temporal locality) and in space (spatial locality)
- This principle guidelines that smaller hardware can be made faster, led to hierarchies based on memories of different speeds and sizes

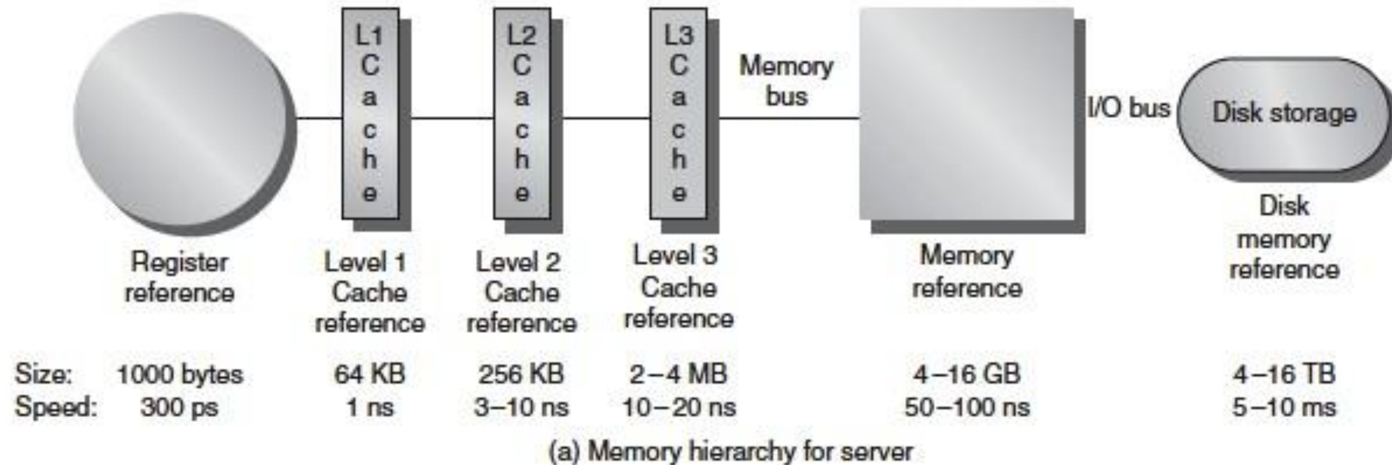
Levels of Memory



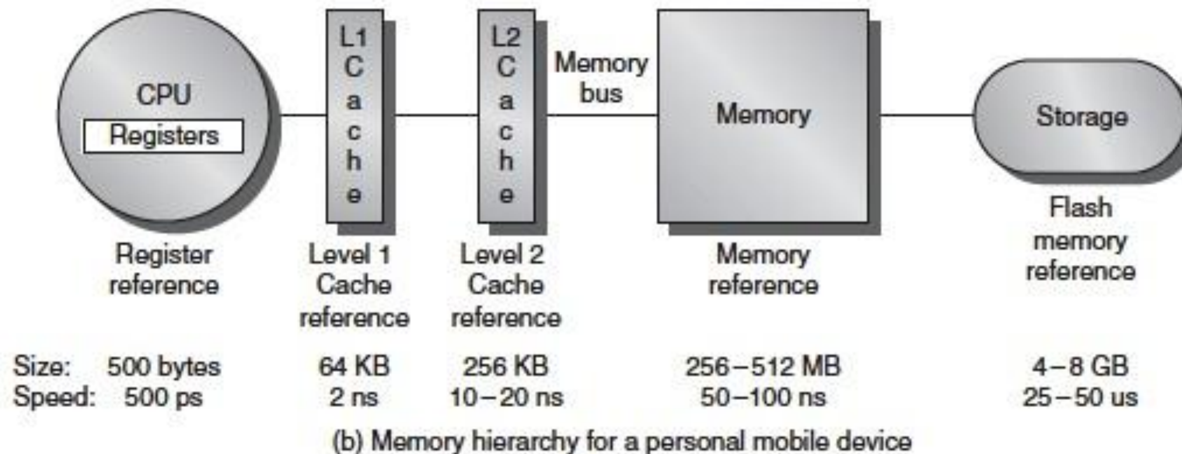
Levels of Memory



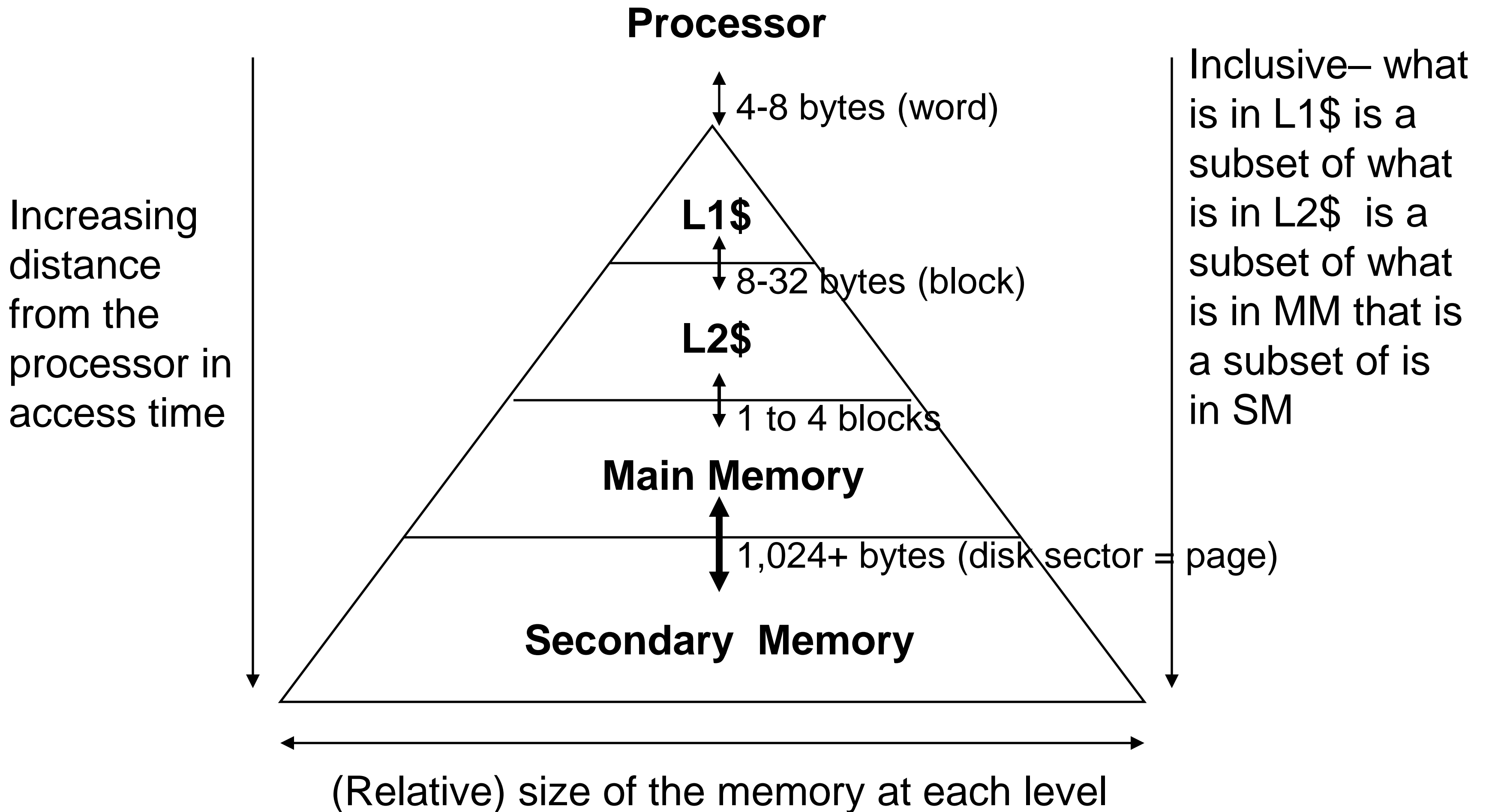
Levels of Memory



(a) Memory hierarchy for server



(b) Memory hierarchy for a personal mobile device


Processor
4-8 bytes (word)
L1$
8-32 bytes (block)
L2$
1 to 4 blocks
Main Memory
1,024+ bytes (disk sector = page)
Secondary Memory
Increasing distance from the processor in access time
Inclusive– what is in L1$ is a subset of what is in L2$ is a subset of what is in MM that is a subset of is in SM
(Relative) size of the memory at each level

Some Basics

- Cache is the highest or first level of the memory hierarchy
- When the processor find a requested data item in the cache, it is called cache hit and otherwise cache miss
- A fixed size collection of data containing the requested word called a block
- The time required for the cache miss depends on both the latency and BW of the memory
- Latency determines the time to retrieve the first word of the block
- BW determines the time to retrieve the rest of this block
- Not all objects referenced by a program need to reside in main memory
- Virtual memory means some objects may reside on disk

Locality of References

- This important fundamental observation comes from properties of programs.
- The most important program property that we regularly exploit is locality of references:
- Programs tend to reuse data and instructions they have used recently.
- **90/10 rule** comes from empirical observation: *"A program spends 90% of its time in 10% of its code"*
- An implication of locality is that we can predict with reasonable accuracy what *instructions* and *data* a program will use in the near future based on its accesses in the recent past.
- **Two different types** of locality have been observed:
  - **Temporal** locality
  - **Spatial** locality

- Temporal locality items accessed recently are likely to be accessed again soon, so it is useful to place it in the cache where it can be accessed quickly
- Keep most recently accessed data closer to processor.
- Example: Instructions in a loop

```
int sum = 0;
for (int i = 0; i < n; i++)
{
sum += i;
}
```

- Spatial locality, items near those accessed recently are likely to be accessed soon
- Move blocks of contiguous words to the upper levels.
- Example: Array data

```
int sum = 0;
for (int i = 0; i < n; i++)
{
sum += a[i];
}
```

Cache Performance

- Number of cycles during which the processor is stalled waiting for a memory access – memory stall cycle
- Number of memory stall cycles depends on both the number of misses and the cost per miss – miss penalty

- CPU execution time = (CPU clock cycles + Memory stall cycles) ×Clock cycle time
- Memory stall cycles = Number of misses × Miss penalty

  = IC × Misses / Instruction × Miss penalty

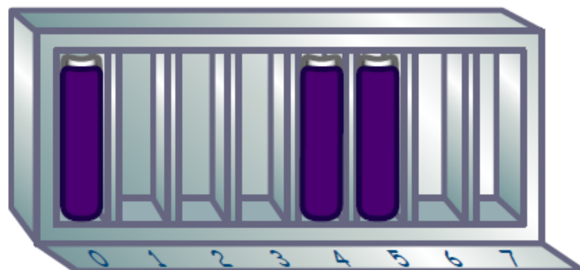
  = IC × Memory Accesses/ Instruction × Miss rate ×Miss penalty

Cache design

- **Block Placement**:
  - Where can a block be placed in the upper level?
- **Block Identification**:
  - How is a block found if it is in the upper level?
- **Block Replacement**:
  - Which block should be replaced on a miss?
- **Write Strategy**:
  - What happens on a write?

Cache design - Block Placement

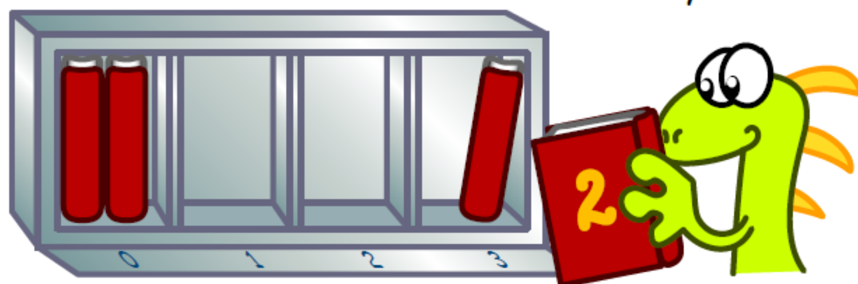
Direct Mapped



Tag	Index	Offset

A cache block can only go in one spot in the cache. It makes a cache block very easy to find, but it's not very flexible about where to put the blocks.

2-Way Set Associative



Tag	Index	Offset

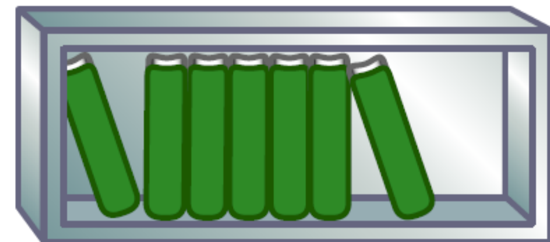
This cache is made up of sets that can fit two blocks each. The index is now used to find the set, and the tag helps find the block within the set.

4-Way Set Associative

Tag	Index	Offset

Each set here fits four blocks, so there are fewer sets. As such, fewer index bits are needed.

Fully Associative



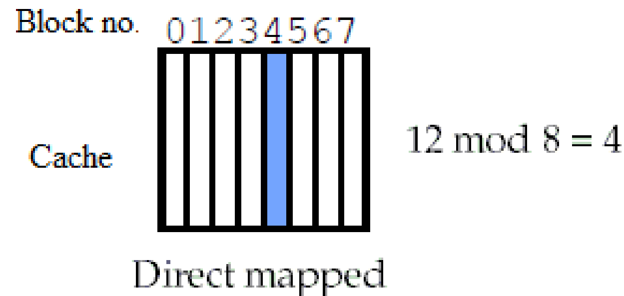
Tag	Offset

No index is needed, since a cache block can go anywhere in the cache. Every tag must be compared when finding a block in the cache, but block placement is very flexible!

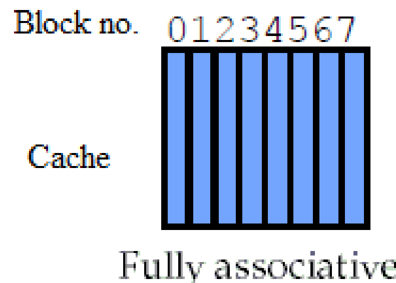
Cache design - Block Placement

- Three categories of cache organization
  - Direct Mapped
    - If each block has only one place it can appear in the cache
    - Mapping is usually

      (Block address) MOD (Number of blocks in cache)



  - Fully associative
    - Block can be placed anywhere in the cache

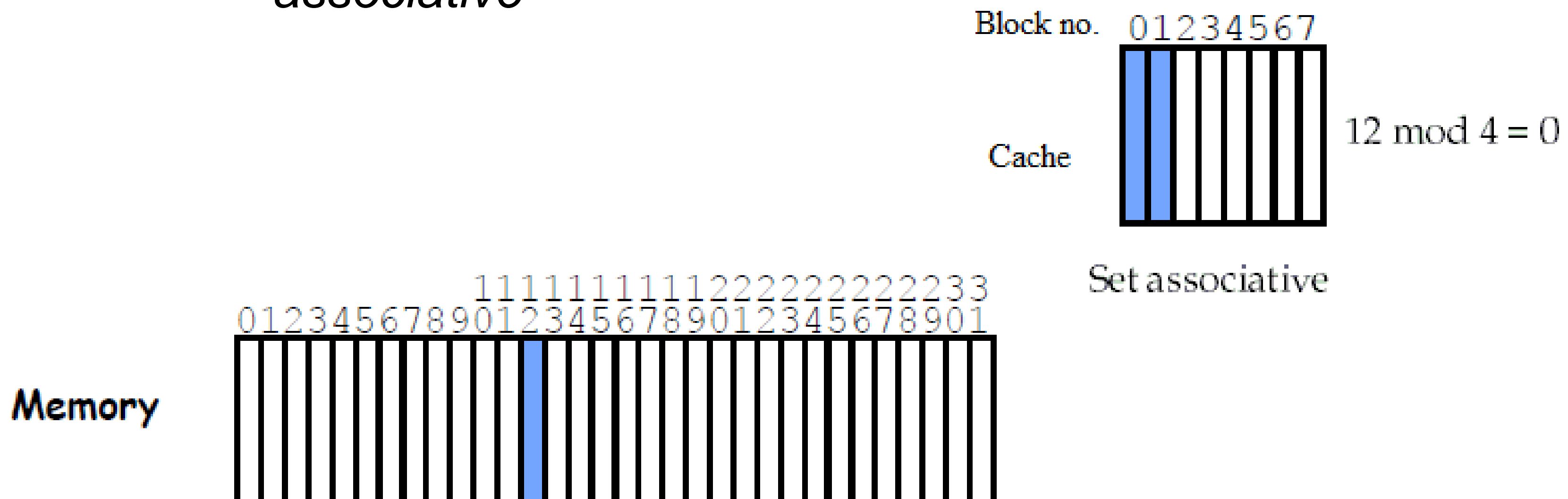


Cache design - Block Placement

– Set associative
  - If a block can be placed in a restricted set of places in the cache
  - A set is a group of blocks in the cache
  - A block is first mapped onto a set, and then the block can be placed anywhere within that set.
  - Set is chosen by bit selection

    (Block address) MOD (Number of sets in cache)

  - *If there are n blocks in a set, cache placement is n-way set associative*



Cache design - Block Identification

- Caches have address tag on each block frame that gives the block address
- The tag is checked to see if it matches the block address from the processor
- All possible tags are searched in parallel as speed is critical
- A valid bit is added to the tag to say whether or not this entry contains a valid address. If the bit is not set, there cannot be a match on this address

Cache design - Block Replacement

- When a miss occurs, the cache controller must select a block to be replaced with the desired data
- In a direct-mapped placement, only one block frame is checked for a hit, and only that block can be replaced
- With fully-associative or set-associative placement, there are many blocks to choose from on a miss
- Three primary strategies employed are:
  - Random – to spread allocation uniformly, candidate blocks are randomly selected.
  - Least-recently used (LRU) – to reduce the chance of throwing out information that will be needed soon, accesses to blocks are recorded. Relying on the past to predict the future, the block replaced is the one that has been unused for the longest time.
  - First-in, first-out (FIFO) – because LRU can be complicated to calculate, this approximates LRU by determining the oldest block rather than the LRU

Assoc:	2-way		4-way		8-way	
Size	LRU	Ran	LRU	Ran	LRU	Ran
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Cache design - Write Strategy

- Reads dominate processor cache accesses. Of the data cache traffic, writes are 28%.
- The block can be read from the cache at the same time that the tag is read and compared, so the block read begins as soon as the block address is available.
- If the read is a hit, the requested block is passed on the processor immediately. If it is a miss, there is no benefit – just ignore the value read

- Modifying a block can not begin until the tag is checked to see if the address is a hit
- Because tag checking cannot occur in parallel, writes normally take longer than reads
- Another complexity is the processor also specifies the size of the write, only that portion of the block can be changed

Cache design - Write Strategy

- There are two basic options when writing to the cache: *write-through* and *write-back*.
  - Write-through – the information is written to both the block in the cache and to the block in the lower-level memory
  - Write-back – the information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.

- With write-back writes occur at the speed of the cache memory, and multiple writes within block require only one write to the lower level. Uses less memory BW making it suitable for multiprocessors

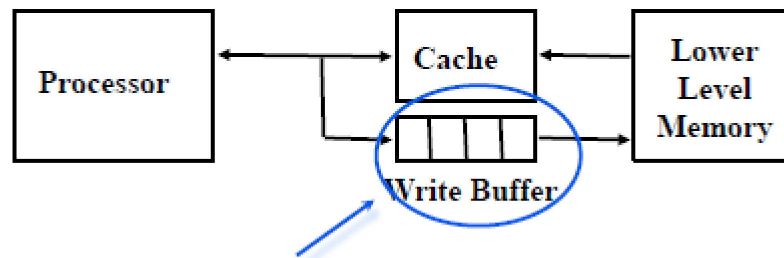
- Write-through easy to implement; cache is always clean. Data coherency

Cache design - Write Strategy

- Use write-back for processor caches to reduce the memory traffic and write-through to keep the cache consistent with the lower levels of memory hierarchy

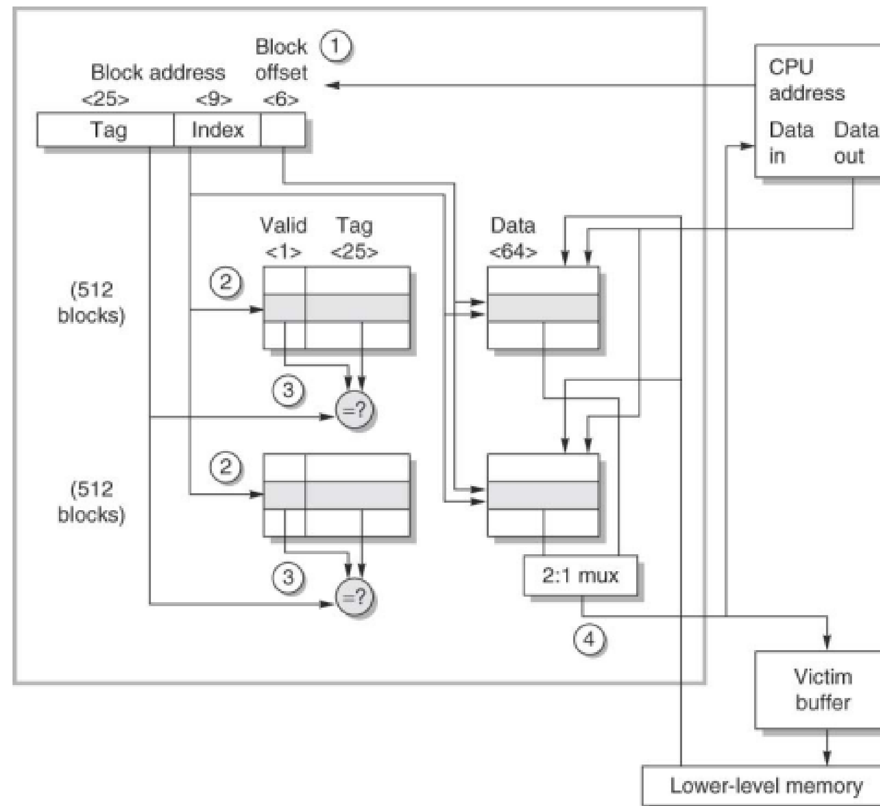
- When the processor must wait for writes to complete during write-through, the processor is said to *write stall*.

- Write buffer allows the processor to continue as soon as the data are written to the buffer, thereby overlapping processor execution with memory updating



Example – Opteron Data Cache

- AMD Opteron microprocessor
  – Cache contains 64KB of data in 64-byte blocks with two-way set associative placement, least-recently used replacement.



Cache Performance

- A better measure is the average memory access time.

  Average memory access time=Hit time + Miss rate*Miss penalty.

  where Hit time – is the time to hit in the cache

  Miss penalty – is the time to replace the block from memory
- It is an indirect measure of performance, although it is a better measure than miss rate

Six Basic Cache Optimizations

- Three categories
  - Reducing Miss rate
    - Larger block size
    - Larger cache size
    - Higher associativity
  - Reducing the miss penalty
    - Multilevel caches
    - Giving reads priority over writes
  - Reducing the time to hit in the cache
    - Avoiding address translation when indexing the cache

Six Basic Cache Optimizations

- Three Cs model – all misses are sort into three categories:
  - Compulsory – It is a cache miss that occurs because the desired data was never in the cache. The very first access to a block cannot be in the cache, so the block must be brought into the cache. Compulsory misses are those that occur even if we have infinite cache
  - Capacity – if the cache cannot contain all the blocks needed during execution of a program, capacity misses will occur because of blocks being discarded and later retrieved
  - Conflict – if the block placement strategy is not fully associative, conflict misses will occur because a block may be discarded and later retrieved if conflicting blocks map to its set
  - Coherency miss

Six Basic Cache Optimizations

1. Larger block size to reduce miss rate

   – the simplest way to reduce the miss rate is to take advantage of spatial locality and increase the block size. Larger blocks also reduces compulsory misses, but they increase the miss penalty

2. Bigger caches to reduce miss rate

   - the way to reduce capacity misses is to increase cache capacity. Drawbacks include potentially longer hit time of the larger cache memory and higher cost and power

3. Higher associativity to reduce miss rate

   - increasing associativity reduces conflict misses. Greater associativity can come at the cost of increased hit time

4. Multilevel caches to reduce miss penalty

   - a conflicting decision is whether to make the cache hit timefast, to keep pace with increased clock rate of processors or to make the cache large, to reduce the gap between the original cache and main memory? The solution is – adding another level of cache.

The first-level cache can be small enough to match a fast clock cycle time, yet the second level cache can be large enough to capture many accesses that would go to main memory

Average memory access time = Hit time L1 + Miss rate L1* (Hit time L2 + Miss rate L2 * Miss penalty L2)

5. Giving priority to read misses over writes to reduce miss penalty

   - it is implemented in a write buffer. Write buffers creates hazards because they hold the updated value of a location needed on a read miss (RAW). One solution is to check the contents of the write buffer on a read miss. If there are no conflicts and the if the memory system is available, sending the read before the writes reduces the miss penalty.

6. Avoiding address translation during indexing of the cache to reduce hit time

   - caches must cope with the translation of a virtual address from the processor to a physical address to access memory. The drawback of virtually indexed, physically tagged optimization is that the size of the page limits the size of the cache. Direct-mapped cache can be no longer bigger than the page size.

- Six basic cache optimizations:
  - Larger block size
    - Reduces compulsory misses
    - Increases capacity and conflict misses, increases miss penalty
  - Larger total cache capacity to reduce miss rate
    - Increases hit time, increases power consumption
  - Higher associativity
    - Reduces conflict misses
    - Increases hit time, increases power consumption
  - Higher number of cache levels
    - Reduces overall memory access time
  - Giving priority to read misses over writes
    - Reduces miss penalty
  - Avoiding address translation in cache indexing
    - Reduces hit time

Ten advanced optimizations of cache performance

Opt.1: small and simple first-level caches to reduce hit time and power

- The time consuming portion of a cache hit is using the index portion of the address to read the tag memory and then compare it to the address
- Smaller hardware can be faster, so a small cache can help the hit time
- It is also critical to keep an L2 cache small enough to fit on the same chip as the processor to avoid the time penalty of going off chip
- Next suggestion is to keep the cache simple, such as using direct mapping.
- Advantage of direct mapped cache is that we can overlap the tag check with the transmission of the data. This effectively reduces the hit time
- Hence, the fast clock cycle encourages small and simple cache designs for the first–level caches

Opt.2: Way Prediction to reduce hit time

- Another approach reduces conflict misses and yet maintains the hit time speed of direct-mapped cache
- In way-prediction, extra bits are kept in the cache to predict the way or block within the set of the next cache access.
- This prediction means the multiplexer is set early to select the desired clock, and only a single tag comparison is performed that clock cycle in parallel with reading the cache data
- A miss results in checking the other blocks for matches in the next clock cycle
- Added to each block of a cache are block predictor bits
- The bits select which of the blocks to try on the next cache access

- If the predictor is correct, the cache access latency is the fast hit time
- If not, it tries the other block, changes the way predictor, and has a latency of one extra clock cycle
- Simulations suggested set prediction accuracy is in excess of 90% for a two-way set, so a way prediction saves pipeline stages more than 90% of the time
- Way prediction is a good match to speculative processors, since they must already undo actions when speculation is unsuccessful

Opt.3: Pipelined cache access to increase cache bandwidth

This optimization is to pipeline cache access so that the effective latency of a first-level cache hit can be multiple clock cycles, giving fast clock cycle time and high bandwidth

Opt.4: Non-blocking caches to increase cache bandwidth

- For pipelined computers that allow out-of-order completion, the processor need not stall on a data cache miss
- Eg. The processor could continue fetching instructions from the instruction cache while waiting for the data cache to return the missing data
- A non-blocking cache or lockup-free cache escalates the potential benefits of such a scheme by allowing the data cache to continue to supply cache hits during a miss
- This hit under miss optimization reduces the effective miss penalty by being helpful during a miss instead of ignoring the requests of the processor
- The cache may further lower the effective miss penalty if it can overlap multiple misses: a ‘hit under multiple miss’ or ‘miss under miss’ optimization.

Opt.5: Multibanked caches to increase cache bandwidth

- Rather than treat the cache as a single monolithic block, we can divide it into independent banks that can support simultaneous accesses
- Banks were originally used to improve performance of main memory and are now used inside modern DRAM chips as well as with caches
- Banking works best when the accesses naturally spread themselves across the banks, so the mapping of addresses to banks affects the behavior of the memory system
- A simple mapping that spread the addresses of the block sequentially across the banks are called sequential interleaving

Opt.6: Critical word first and early restart to reduce miss penalty

- This technique is based on the observation that the processor normally needs just one word of the block at a time
- This strategy is impaatience: don't wait for the full block to be loaded before sending the requested word and restarting the processor
- There are two specific strategies:
  - Critical word first – request the missed word first from the memory and send it to the processor as soon as it arrives; let the processor continue execution while filling the rest of the words in the block
  - Early restart – fetch the words in normal order, but as soon as the requested word of the block arrives, send it to the processor and let the processor continue execution
- These techniques only benefits designs with large cache blocks

- Caches normally continue to satisfy accesses to other blocks while the rest of the block is being filled
- Given spatial locality, there is a good chance that the next reference is to the rest of the block
- Just as with non-blocking caches, the miss penalty is not simple to calculate
- When there is a second request in critical word first, the effective miss penalty is the non-overlapped time from the reference until the second piece arrives
- The benefits of critical word first and early restart depends on the size of the block and the likelihood of another access to the portion of the block that has not yet been fetched

Opt.7: Merging write buffer to reduce miss penalty

- Write-through caches rely on write buffers, as all stores must be sent to the next lower level of the hierarchy
- Even write-back cache use a simple buffer when a block is replaced
- If the write buffer is empty, the data and the full address are written in the buffer and the write is finished from processor's perspective; the processor continues working while write buffer prepares to write the word to memory
- If the buffer contains other modified blocks, the addresses can be checked to see if the address of this new data matches the address of a valid write buffer entry

- If so, the new data are combined with that entry
- Write merging is the name of this optimization
- If the buffer is full and there is no address match, the cache must wait until the buffer has an empty entry.
- This optimization uses the memory more effectively since multiword writes are usually faster than writes performed one word at a time
- This optimization also reduces stalls due to the write buffer being full

Opt.8: Compiler optimizations to reduce miss rate

- So far, the techniques have required changing the hardware.
- This technique reduces miss rate without any hardware changes
- Code and data rearrangement
- Code can easily be rearranged without affecting correctness
- Reordering the procedures of a program might reduce instruction miss rates by reducing conflict misses
- Aligning basic blocks so that the entry point at the beginning of a cache block decreases the chance of a cache miss for sequential code
- If the compiler knows that a branch is likely to be taken, it can improve spatial locality by changing the sense of the branch and swapping the basic block at the branch target with the basic block sequentially after the branch
- Branch straightening is the name of this optimization
- Data have even fewer restrictions on location than code
- The goal of such transformations is to try to improve the spatial and temporal locality of the data

Opt.9: Hardware prefetching of instructions and data to reduce miss penalty or miss rate

- Non-blocking caches effectively reduces the miss penalty by overlapping execution with memory access
- Another approach is to prefetch items before the processor requests them
- Both instruction and data can be prefetched either directly into the caches or into an external buffer that can be more quickly accessed than main memory
- Instruction prefetch is frequently done in hardware outside of the cache
- Typically a processor fetches two blocks on a miss: the requested block and the next consecutive block

- The requested block is placed in the instruction cache when it returns, and the prefetched block is placed into the instruction stream buffer
- If the requested block is present in the instruction stream buffer, the original cache request is cancelled, and the block is read from the stream buffer and the next prefetch request is issued
- Prefetching relies on utilizing memory bandwidth that otherwise would be unused
- But it interferes with demand misses, it can actually lower performance
- Help from compilers can reduce useless prefetching

Opt.10: Compile controller prefetching to reduce miss penalty or miss rate

- An alternative to hardware prefetching is for the compiler to insert prefetch instructions to request data before the processor needs it
- There are two flavors of prefetch:
- Register prefetch – will load the value into a register
- Cache prefetch – loads data only into the cache and not the register
- Either of these can be faulting or nonfaulting: that is the address does or does not cause an exception for virtual address faults and protection violations.
- The most effective prefetch is 'semantically invisible' to a program
- It doesnot change the contents of registers and memory and can not cause virtual memory faults

- Prefetching makes sense only if the processor can proceed while prefetching the data; that is, the caches do not stall but continue to supply instructions and data; while waiting for the prefetched data to return
- Like hardware controlled prefetching, the goal is to overlap execution with the prefetching of data
- Loops are the important targets, as they lend themselves to prefetch optimizations
- If the miss penalty is small, the compiler just unrolls the loop once or twice, and it schedules the prefetches with the execution

Summary

Technique	Hit time	Band-width	Miss penalty	Miss rate	Power consumption	Hardware cost/ complexity	Comment
Small and simple caches	+			–	+	0	Trivial; widely used
Way-predicting caches	+				+	1	Used in Pentium 4
Pipelined cache access	–	+				1	Widely used
Nonblocking caches		+	+			3	Widely used
Banked caches		+			+	1	Used in L2 of both i7 and Cortex-A8
Critical word first and early restart			+			2	Widely used
Merging write buffer			+			1	Widely used with write through
Compiler techniques to reduce cache misses				+		0	Software is a challenge, but many compilers handle common linear algebra calculations
Hardware prefetching of instructions and data			+	+	–	2 instr., 3 data	Most provide prefetch instructions; modern high-end processors also automatically prefetch in hardware.
Compiler-controlled prefetching			+	+		3	Needs nonblocking cache; possible instruction overhead; in many CPUs

Figure 2.11 Summary of 10 advanced cache optimizations showing impact on cache performance, power consumption, and complexity. Although generally a technique helps only one factor, prefetching can reduce misses if done sufficiently early; if not, it can reduce miss penalty. + means that the technique improves the factor, – means it hurts that factor, and blank means it has no impact. The complexity measure is subjective, with 0 being the easiest and 3 being a challenge.

Memory Technology

- Performance metrics
  - Latency is concern of cache
  - Bandwidth is concern of multiprocessors and I/O
  - Access time
    - Time between read request and when desired word arrives
  - Cycle time
    - Minimum time between unrelated requests to memory
- DRAM used for main memory, SRAM used for cache

Memory Technology

- SRAM
  - Requires low power to retain bit
  - Requires 6 transistors/bit
- DRAM
  - Must be re-written after being read
  - Must also be periodically refreshed
    - Every ~ 8 ms
    - Each row can be refreshed simultaneously
  - One transistor/bit
  - Address lines are multiplexed:
    - Upper half of address: row access strobe (RAS)
    - Lower half of address: column access strobe (CAS)

Memory Technology

- Amdahl:
  - Memory capacity should grow linearly with processor speed
  - Unfortunately, memory capacity and speed has not kept pace with processors
- Some optimizations:
  - Multiple accesses to same row
  - Synchronous DRAM
    - Added clock to DRAM interface
    - Burst mode with critical word first
  - Wider interfaces
  - Double data rate (DDR)
  - Multiple banks on each DRAM device

Memory Optimizations

- DDR:
  - DDR2
    - Lower power (2.5 V -> 1.8 V)
    - Higher clock rates (266 MHz, 333 MHz, 400 MHz)
  - DDR3
    - 1.5 V
    - 800 MHz
  - DDR4
    - 1-1.2 V
    - 1600 MHz

- GDDR5 is graphics memory based on DDR3

Memory Optimizations

- Graphics memory:
  - Achieve 2-5 X bandwidth per DRAM vs. DDR3
    - Wider interfaces (32 vs. 16 bit)
    - Higher clock rate
      - Possible because they are attached via soldering instead of socketted DIMM modules
- Reducing power in SDRAMs:
  - Lower voltage
  - Low power mode (ignores clock, continues to refresh)

Flash Memory

- Type of EEPROM
- Must be erased (in blocks) before being overwritten
- Non volatile
- Limited number of write cycles
- Cheaper than SDRAM, more expensive than disk
- Slower than SRAM, faster than disk

Memory Dependability

- Memory is susceptible to cosmic rays
- *Soft errors*: dynamic errors
  - Detected and fixed by error correcting codes (ECC)
- *Hard errors*: permanent errors
  - Use sparse rows to replace defective rows

- Chipkill: a RAID-like error recovery technique

Virtual Memory & Virtual Machines

- Security and privacy are two of the most challenges for information technology
- Virtual memory provides an architectural support for protecting processes from each other

Protection via virtual memory

- Page based virtual memory, including a translation look aside buffer that caches page table entries, is the primary mechanism that protects processes form each other
- When computers are running multiple processes, it would be too expensive to dedicate a full address space worth of memory for each process. Hence there must be a means of sharing a smaller amount of physical memory among many processes

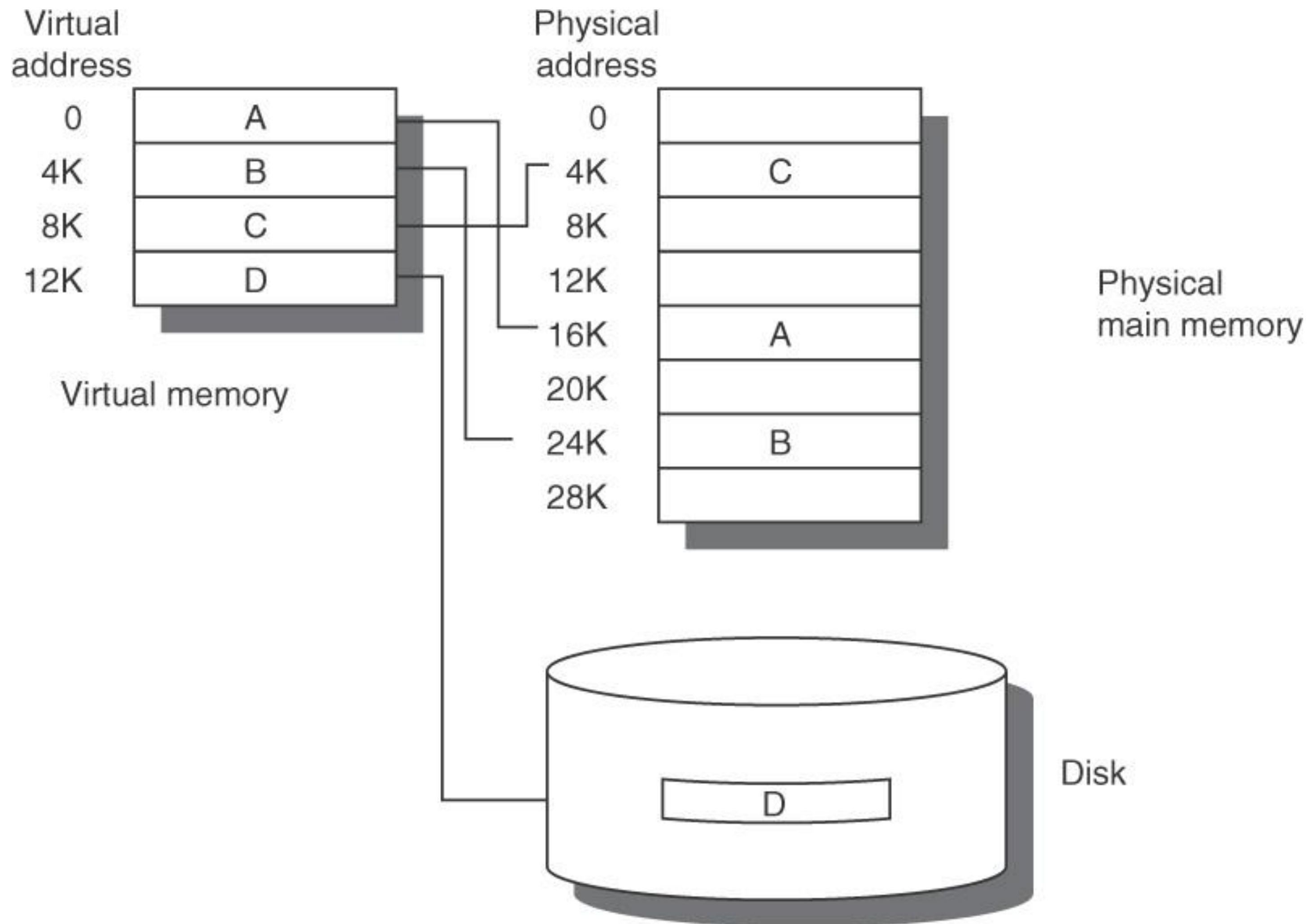
- One way to do this is by virtual memory (VM)
- It divides physical memory into blocks and allocates them to different processes
- Inherent in such an approach must be a protection scheme that restricts a process to the blocks belonging only to that process
- Most forms of virtual memory also reduce the time to start a program, since not all the code and data need be in physical memory before a program can begin
- Although VM provides protection, sharing is not the reason that VM was invented

- VM was invented to relieve programmers of their burden to fit in the code in the available physical memory of the computer appropriately
- It automatically manages the two levels of the memory hierarchy represented by main memory and secondary storage
- In addition to sharing protected memory space and automatically managing the memory hierarchy, VM also simplifies loading the program for execution
- Called relocation, this mechanism allows the same program to run in any location in physical memory

Virtual memory

- Virtual memory divides physical memory into blocks and allocates them to different processes.
- With virtual memory, each *process* has its own **address space**.
- In most modern microprocessors intended for general-purpose use, a **memory management unit**, or **MMU**, is built into the hardware.
- The MMU's job is to translate **virtual addresses into physical addresses**.
- a **program** runs in a **process**, and each process has its own **virtual memory space**.
- The unit exchanged between memory and disc is called a **page**

Virtual address space



© 2007 Elsevier, Inc. All rights reserved.

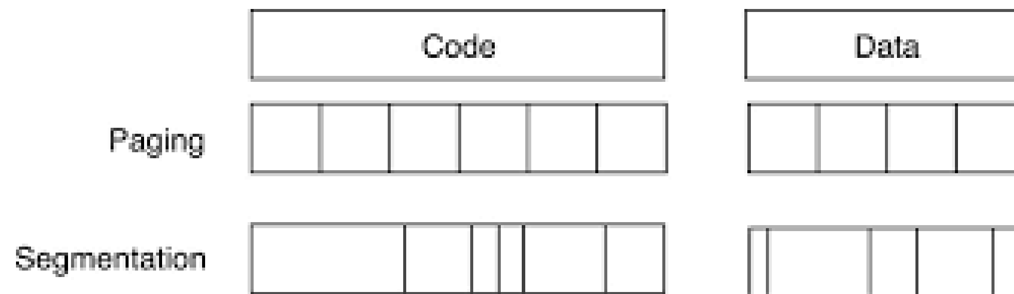
- The address produced by the processor is called a virtual address
- This gets translated by a MMU via a page table into a physical address (PT hit) or page fault (PT miss)
- This mapping process is called address translation or memory mapping
- Two classes
  - Fixed size blocks called pages
  - Variable size blocks called segments

- Several general memory hierarchy ideas about caches are analogous to VM
- Page or segment is used for a block
- Page fault or address fault is used for a miss
- With VM, the processor produces virtual addresses that are translated by a combination of hardware and software to physical addresses, which access main memory
- This process is called memory mapping or address translation
- Replacement on cache miss is primarily controlled by hardware, while VM replacement is controlled by OS

- The size of the processor address determines the size of the VM, but the cache size is independent of the processor address size
- VM systems can be categorized into two classes:
- Those with fixed size blocks called – pages (fixed at 4096 to 8192 bytes)
- Those with variable size blocks – segments (size varies – 1byte to 2^{32} bytes)

Paged Vs Segmented Virtual Memory

- Paged Virtual Memory – memory divided into fixed sized pages
  - each page has a base physical address
- Segmented Virtual Memory – memory is divided into variable length segments
  - each segment has a base pysical address + length



© 2003 Elsevier Science (USA). All rights reserved.

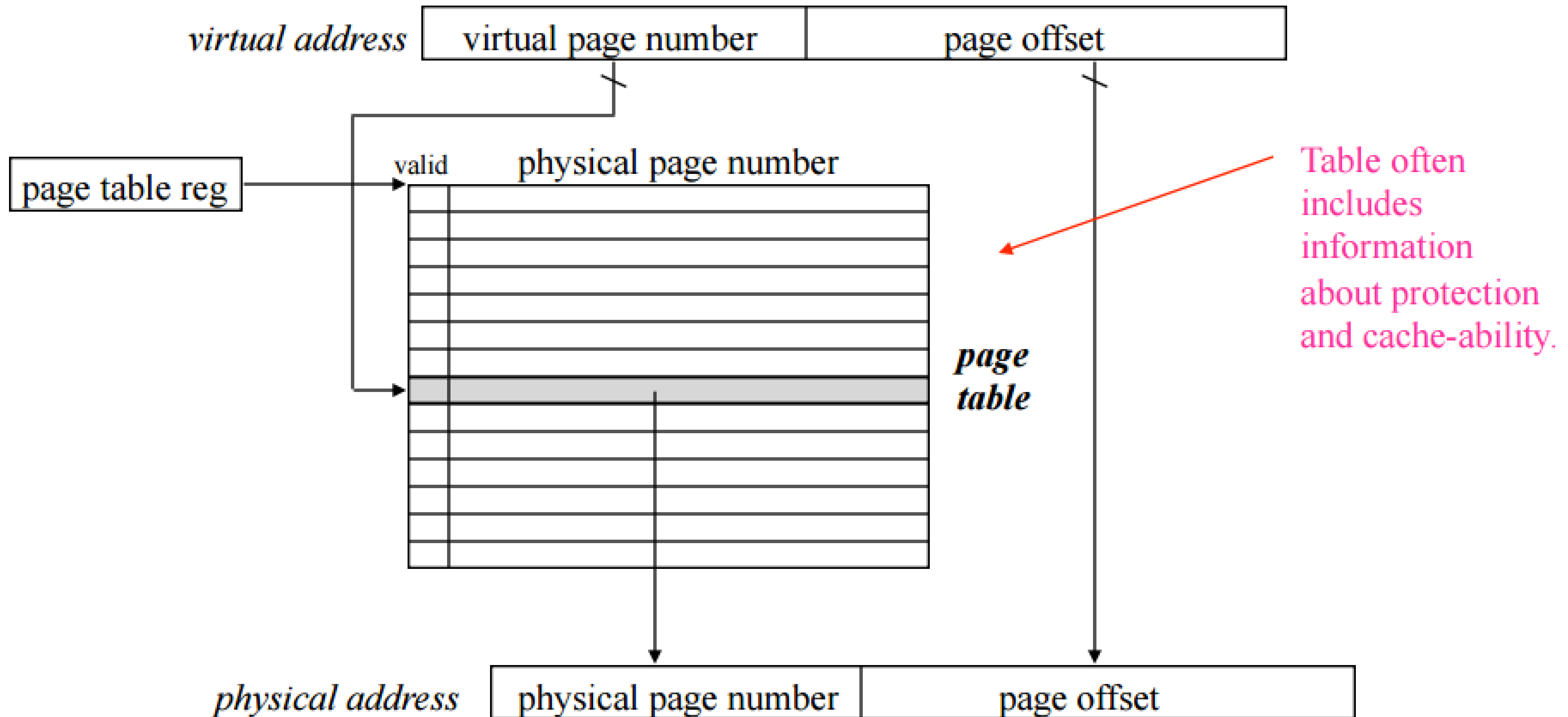
Parameter	First-level cache	Virtual memory
Block (page) size	16–128 bytes	4096–65,536 bytes
Hit time	1–3 clock cycles	100–200 clock cycles
Miss penalty	8–200 clock cycles	1,000,000–10,000,000 clock cycles
(access time)	(6–160 clock cycles)	(800,000–8,000,000 clock cycles)
(transfer time)	(2–40 clock cycles)	(200,000–2,000,000 clock cycles)
Miss rate	0.1–10%	0.00001–0.001%
Address mapping	25–45-bit physical address to 14–20-bit cache address	32–64-bit virtual address to 25–45-bit physical address

Figure B.20 Typical ranges of parameters for caches and virtual memory. Virtual memory parameters represent increases of 10 to 1,000,000 times over cache parameters. Normally, first-level caches contain at most 1 MB of data, whereas physical memory contains 256 MB to 1 TB.

Page Table

- Viritual memory is usually done by dividing memory up into **pages**
- **Data structure** indexed by the page or segment number
- For segmentation – offset added to segment's physical address to obtain final physical address
- For paging – offset concatenated to physical page address
- data structure that holds the mapping from virtual to physical addresses – page table
- The **page frame** is the actual physical storage in memory.

Address translation via Paging



- all page mappings are in the page table, so hit/miss is determined solely by the valid bit (i.e., no tag)

Techniques for Fast Address Translation

- Page tables are stored in the main memory, each memory access of a program requires at least one memory accesses to translate virtual into physical address.
- On the cache miss, there will be two memory accesses.
- When a translation for a virtual page is used, it will probably be needed again in the near future because the references to the words on that page have both temporal and spatial locality.
- Each virtual memory reference can cause two physical memory accesses :

  -One to fetch the page table.

  -One to fetch the data.
- To overcome this problem a high-speed cache is set up for page table entries called a Translation Lookaside Buffer (TLB).
- TLB is a special cache used to keep track of recently used transactions.

- When a virtual memory address is referenced by a program, the search starts in the CPU.
- First, instruction caches are checked.
- If the required memory is not in these very fast caches, the system has to look up the memory's physical address.
- At this point, TLB is checked for a quick reference to the location in physical memory.

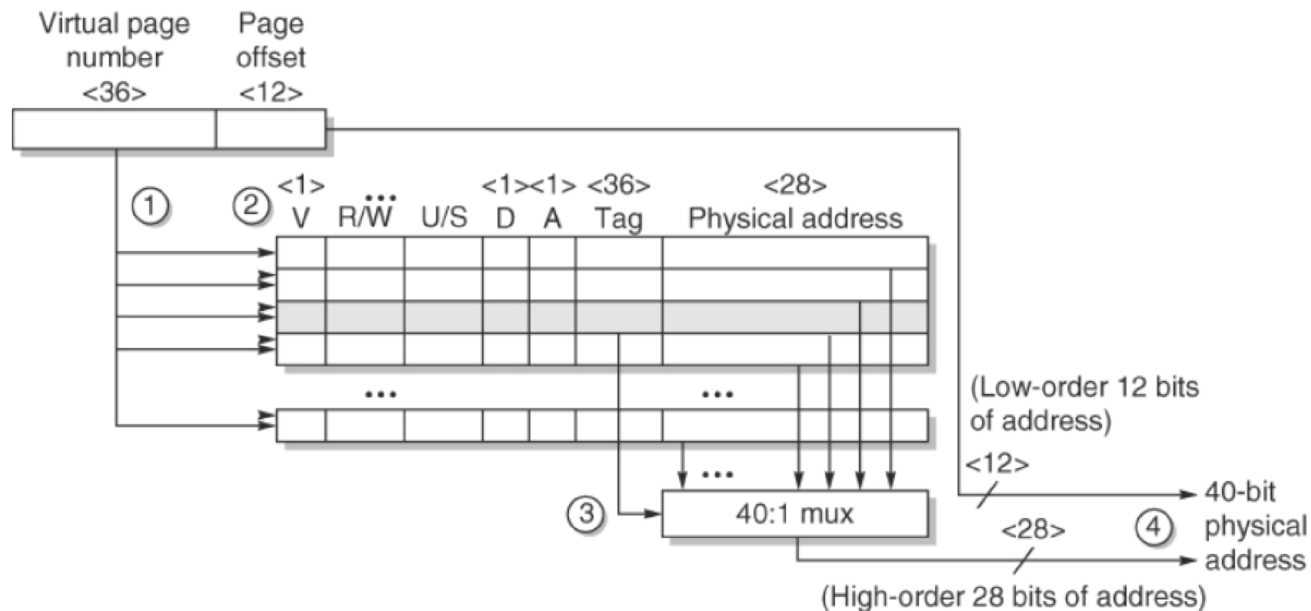


Figure B.24 Operation of the Opteron data TLB during address translation. The four steps of a TLB hit are shown as circled numbers. This TLB has 40 entries. Section B.5 describes the various protection and access fields of an Opteron page table entry.

- Paged virtual memory access logically takes at least twice as long, with one memory access to obtain the physical address and second access to get the data
- The solution is to rely on the principle of locality
- If accesses have locality, then the address translations also must have locality
- By keeping these address translations in a spatial cache, a memory access rarely requires a second access to translate the data
- This special address translation cache is referred to as TLB
- A TLB entry is like a cache entry where the tag holds portion of the virtual address and the data portion holds a physical page address, protection field, valid bit and usually a use bit and a dirty bit

- The OS changes these bits by changing the value in the page table and then validating the corresponding TLB entry
- When the entry is reloaded from the page table, the TLB gets an accurate copy of bits
- The computer then maps virtual addresses to physical addresses

Protection via virtual machines

- The broadest definition of VM includes basically all emulation methods that provide a standard software interface, such as the Java VM
- A single computer runs multiple VMs and can support a no. of different OSes.

- On a conventional platform, a single OS ‘owns’ all the hardware resources, but with a VM, multiple Oses all share the hardware resources
- The software that supports VM is called a virtual machine monitor (VMM)
- The VMM is the heart of the VM technology
- The underlying hardware platform is called the host, and its resources are shared among the guest VMs
- The VMM determines how to map virtual resources to physical resources
- A physical resource may be time shared, partitioned..
- The VMM is much smaller than the traditional OS
- The VM provides two other benefits: Managing software and Managing hardware