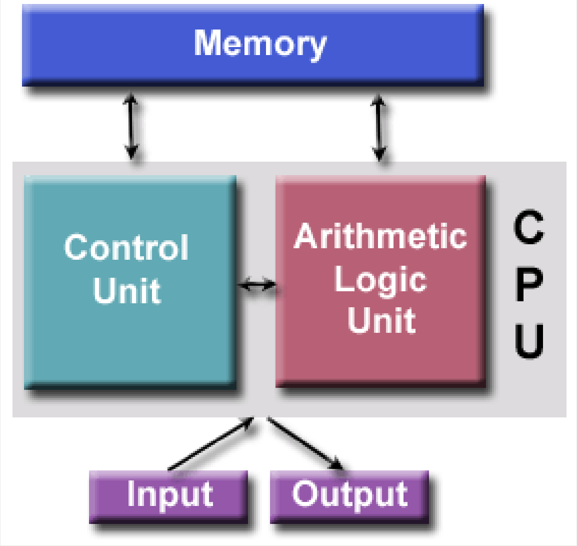
Memory Performance and Optimization

- Latency
- Bandwidth
- Capacity
- Energy

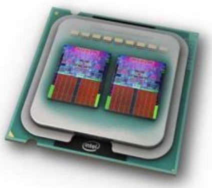
Von Neumann Architecture

- John von Neumann first authored the general requirements for an electronic computer in 1945
- Aka “stored-program computer”
  - Both program inst. and data are kept in electronic memory
- Since then, all computers have followed this basic design
- Four main components: memory, control unit, ALU, I/O

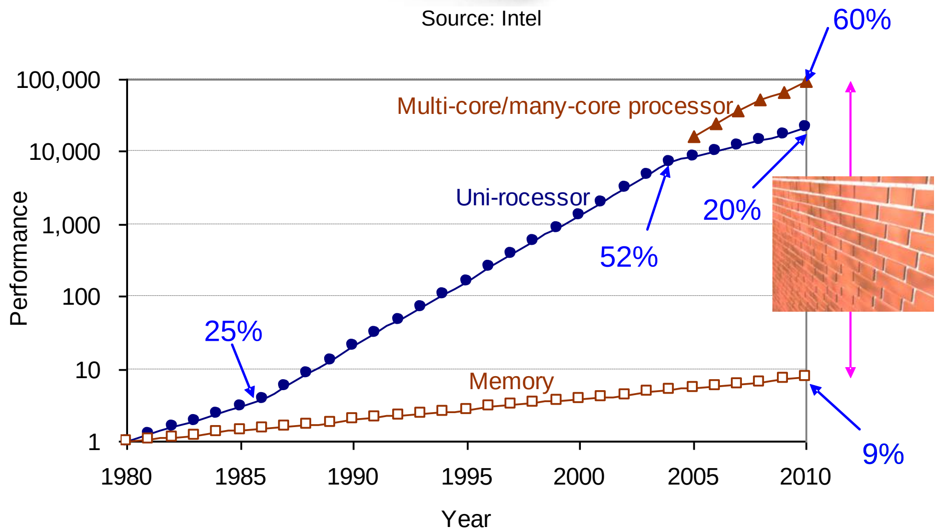


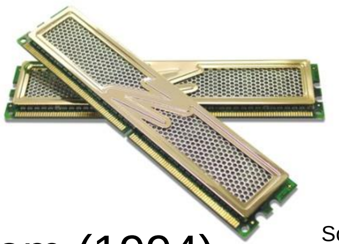
The Memory-wall Problem

- Processor performance increases rapidly
  - Uni-processor: ~52% until 2004
  - Aggregate multi-core/many-core processor performance even higher since 2004
- Memory: ~9% per year
- Processor-memory speed gap keeps increasing



Source: Intel





Source: OCZ

Memory-bounded speedup (1990), Memory wall problem (1994)

Improve via Memory Hierarchy

Capacity
Access Time, Bandwidth

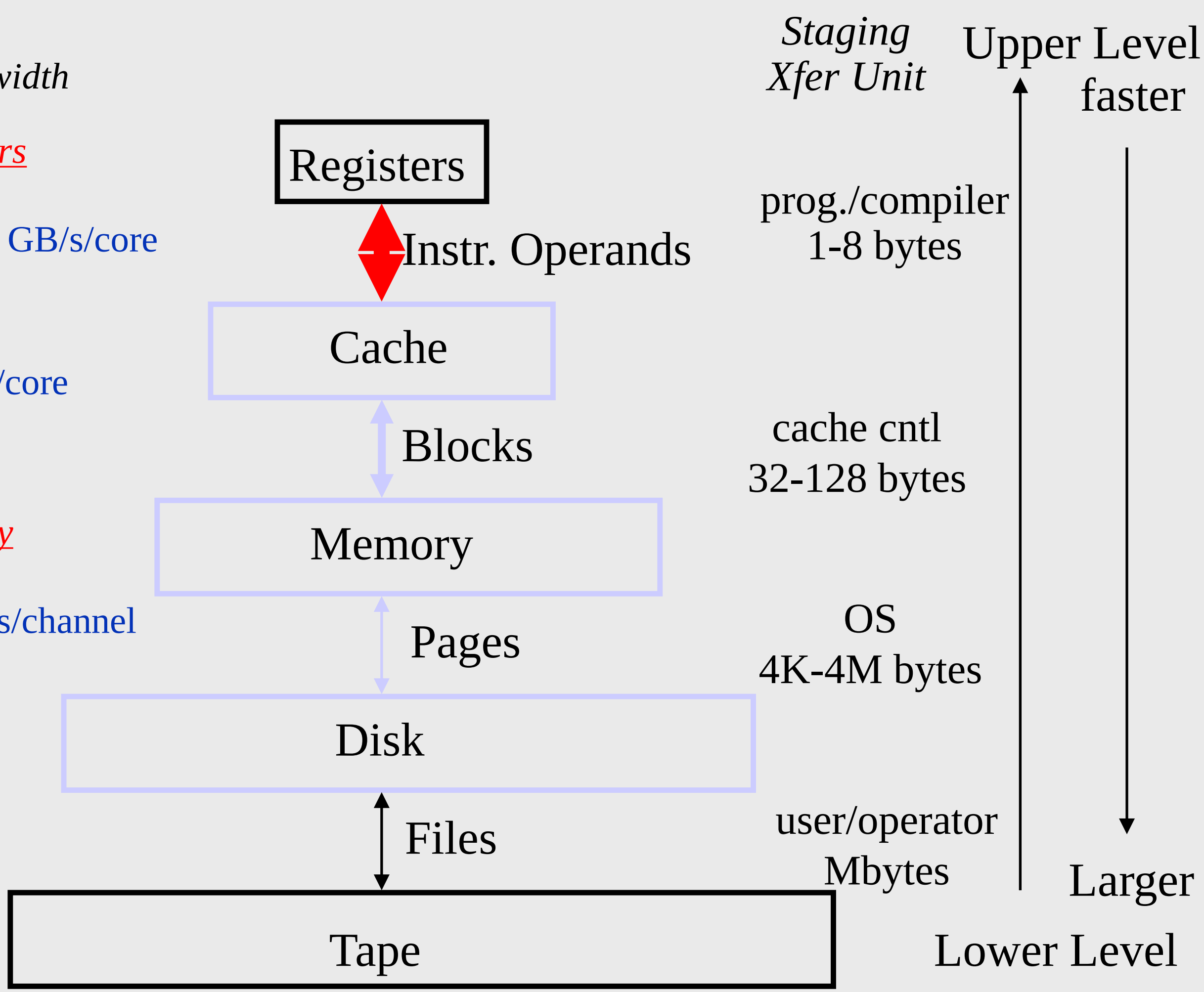
CPU Registers
<8KB
<0.2~0.5 ns, 500~800 GB/s/core

Cache
<50MB
1-10 ns, 50~150GB/s/core

Main Memory
Giga Bytes
50ns-100ns 5~10GB/s/channel

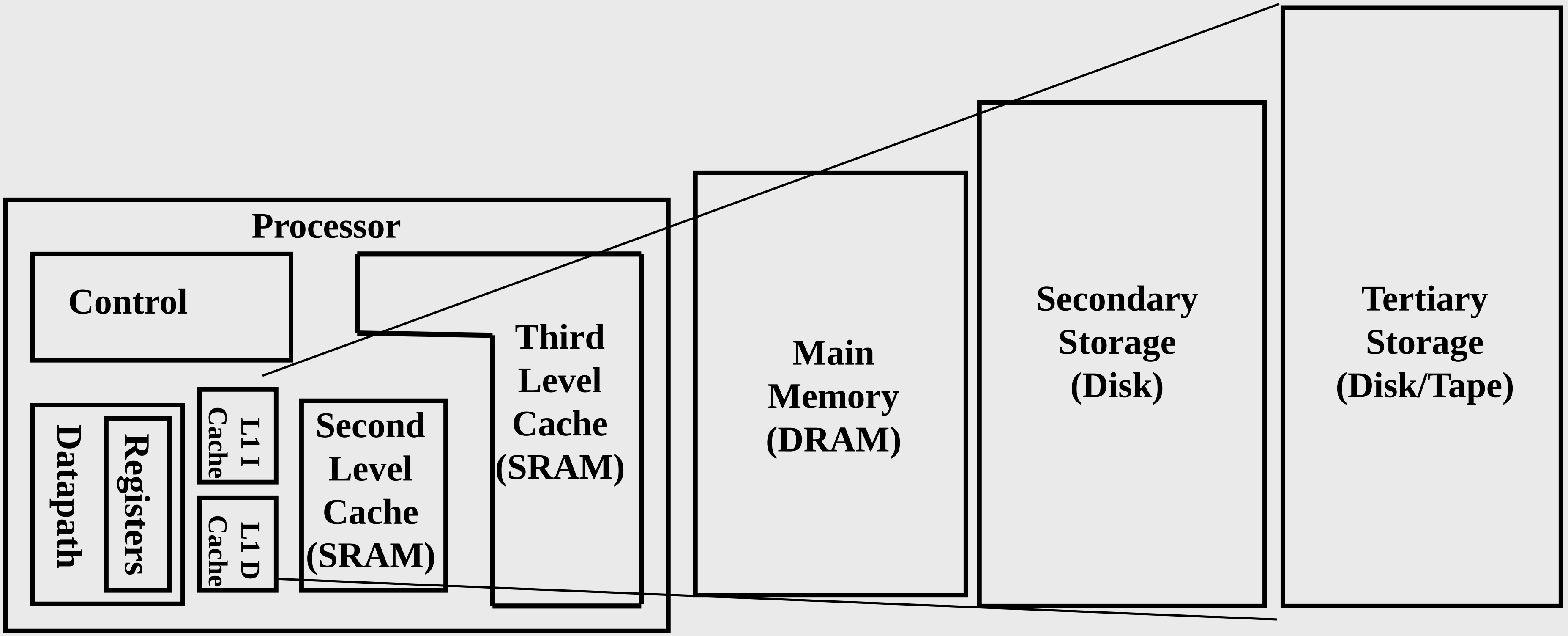
Disk
Tera Bytes, 5 ms
100~300MB/s

Tape
Peta Bytes or
infinite
sec-min



Modern Memory Hierarchy

- By taking advantage of the principle of locality:
  - Present the user with as much memory as is available in the cheapest technology.
  - Provide access at the speed offered by the fastest technology.



Multi-core Microprocessor Component Elements

- Multiple processor cores
  - One or more processors
- L1 caches
  - Instruction cache
  - Data cache
- L2 cache
  - Joint instruction cache
  - Dedicated to individual core processor
- L3 cache
  - Not all systems
  - Shared among multiple cores
  - Often off die but in same package
- Memory interface
  - Address translation and management (sometimes)
  - North bridge
- I/O interface
  - South bridge

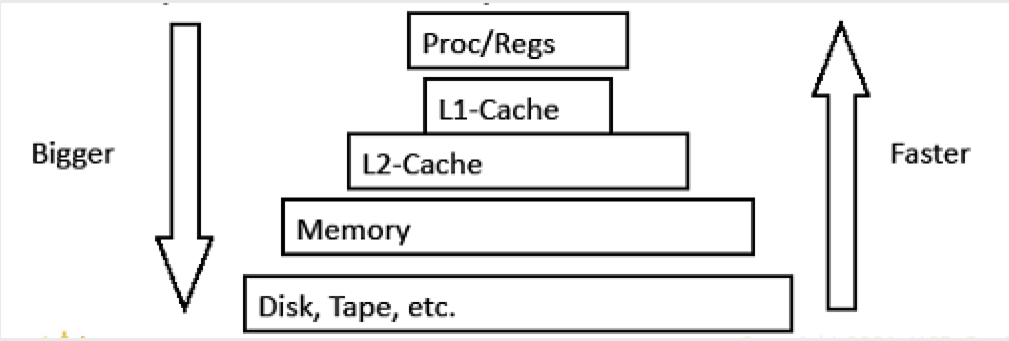
Memory

- Serves as storage of computational state and instructions to transform that state
- Organized in a hierarchy, ordered by increasing access latency and decreasing bandwidth:
  - Primary storage
    - Directly accessible by the CPU
    - Volatile (loses contents after power-down) in most cases today
    - Includes: CPU registers, CPU caches, main memory
  - Secondary storage
    - Not directly accessible by the CPU
    - Data transfers accomplished through an intermediate area in primary storage and dedicated I/O channels
    - Non-volatile
    - Includes: hard disks, removable optical disks (CD and DVD drives), USB sticks, tape drivers, etc.
  - Tertiary storage
    - Involves automated (robotic) mechanisms mounting removable media on demand
    - Includes: tape libraries, optical jukeboxes
  - Off-line storage
    - involves human interaction to handle the storage media
    - Used for physical information transfer, e.g., to ensure information security

What is a cache

- Small, fast storage used to improve average access time to slow memory.
- Exploits spacial and temporal locality
- In computer architecture, almost everything is a cache!
  - Registers a cahe on variables
  - First-level cache a cache on seconde-level cache
  - Second-level cache a cache on memory
  - Memory a cache on disk (virtual memory)
  - TLB a cache on page table
  - Branch-prediction a cache on prediction information?

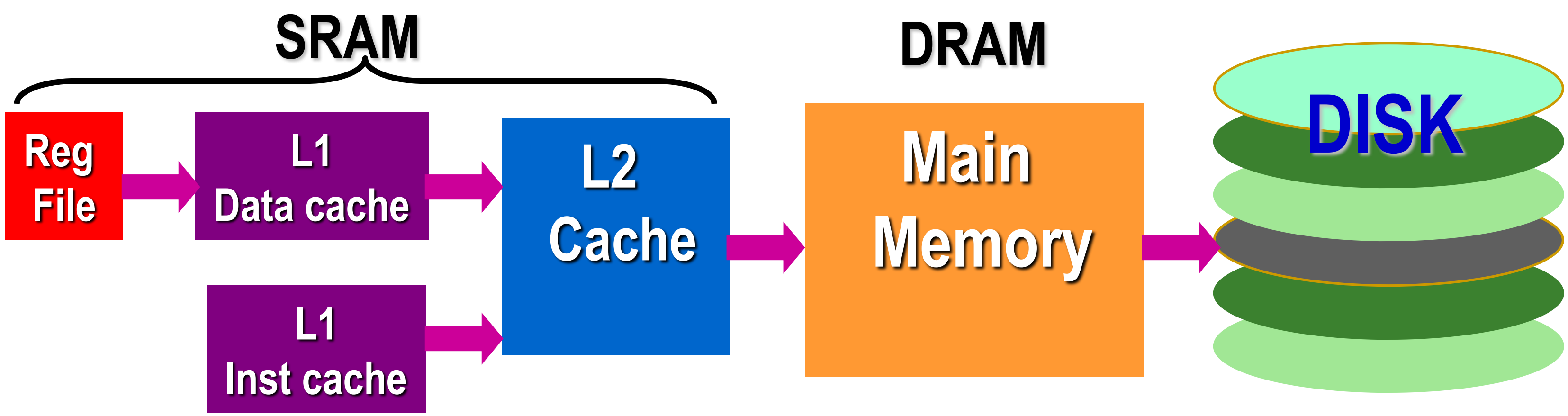
Fig:



Any Questions

Question: Can we build a big cache?

Solution: Memory Hierarchy



Memory Technology Metrics

- Memory access *latencies* impact how quickly the information is available to the processor
  - 100s of ps (CPU registers) to 10s… 100s of ns (main memory) for primary storage
  - Around 100us for FLASH memory, several ms for hard drives, hundreds of ms for optical disks
  - Single to tens of seconds for robotic storage libraries
- Memory bandwidth defines the maximum rate at which information is available for processing, preventing processor *starvation*
  - 10s of GB/s for registers and L1 cache, single GB/s for main memory
  - Ranges from 10s MB/s (optical media and FLASH) to 100s MB/s (high performance hard disks)
  - Low 10s to over 100 MB/s for single tape driver; aggregate throughput may be much higher depending on aggregate number of devices
- Capacity: determines maximum problem size possible to compute (otherwise memory *starvation* and potential *contention* between competing processes/threads may result)
  - Capacities range from few bytes per register, few of KB to tens of MB for CPU caches, 100s of GB to 1 TB for hard disks, and up to tens of PB for large tertiary storage systems
- Capacity density: determines additional costs associated with assembling storage of required capacity, such as volume, or device footprint

Technology Classes and Related Metrics

- Microprocessors
  - Clock rate
  - Instructions per Cycles (CPI)
  - Power
- Memory
  - Access Times
  - Bandwidth
  - Capacity, Size
- Networking
  - Bandwidth
  - Latency

Main Memory Implementations

- Static Random Access Memory (SRAM)
  - Fastest access time
  - Modest power consumption per bit
  - Relatively expensive to manufacture
  - Discrete SRAM is slowly displaced in favor of PSRAM, which is based on a DRAM equipped with an SRAM-like interface
- Dynamic Random Access Memory (DRAM)
  - Cheapest production cost per bit
  - Highest capacity per die area
- FLASH Memory
  - Non-volatile
  - Capacities approach those of DRAM
  - NvRAM

The Principle of Locality

- The Principle of Locality:
  - Programs access a relatively small portion of the address space at any instant of time.
- Two Different Types of Locality:
  - **Temporal Locality** (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - **Spatial Locality** (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon
    (e.g., straight line code, array access)
    - Cache Block or Cache Line
- Last 30 years, HW relied on locality for speed

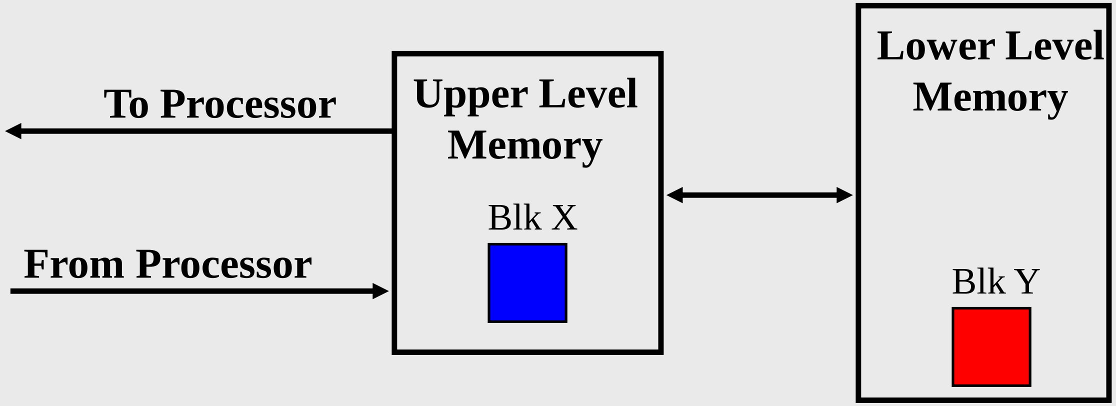
Terminology

- Cache
  - Originally referring to the first level of the memory hierarchy
  - The principle of locality applies to many levels
  - File caches, name caches, web caches
- Hit: data appears in some block in the upper level
  - Hit Rate: the fraction of memory accesses found in the upper level
  - Hit Time: Time to access the upper level which consists of

    RAM access time + Time to determine hit/miss

Terminology

- Miss: data needs to be retrieve from a block in the lower level (Block Y)
  - Miss Rate = 1 - (Hit Rate)
  - Miss Penalty: Time to replace a block in the upper level + Time to deliver the block the processor
- Hit Time << Miss Penalty



Memory Hierarchy Basics

- Miss rate
  - Fraction of cache access that result in a miss
- Causes of misses
  - Compulsory
    - First reference to a block
  - Capacity
    - Blocks discarded and later retrieved
  - Conflict
    - Program makes repeated references to multiple addresses from different blocks that map to the same location in the cache

Three (or Four) Cs

- Compulsory Misses:
  - The first access to a block is not in the cache, so the block must be brought into the cache
- Capacity Misses:
  - If the cache cannot contain all the blocks needed during execution of a program, capacity misses will occur due to blocks being discarded and later retrieved.
- Conflict Misses:
  - If block-placement strategy is set associative or direct mapped, conflict misses will occur because a block can be discarded and later retrieved if too many blocks map to its set. Also called collision misses or interference misses

Fourth kind

- Coherence Misses:
  - in multiprocessor systems and multi-core processors
  - Sometimes are false sharing

Memory Hierarchy Basics

$$\frac{\text{Misses}}{\text{Instruction}} = \frac{\text{Miss rate} \times \text{Memory accesses}}{\text{Instruction count}} = \text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}}$$

$$\text{Average memory access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

- Note that speculative and multithreaded processors may execute other instructions during a miss
  - Reduces performance impact of misses

Conventional AMAT

- The traditional AMAT(Average Memory Access Time) :

$$AMAT = HitCycle + MR \times AMP$$

- MR is the miss rate of cache accesses; and AMP is the average miss penalty

AMAT is **Recursive**

- $AMAT = HitCycle + MR \times AMAT_2$

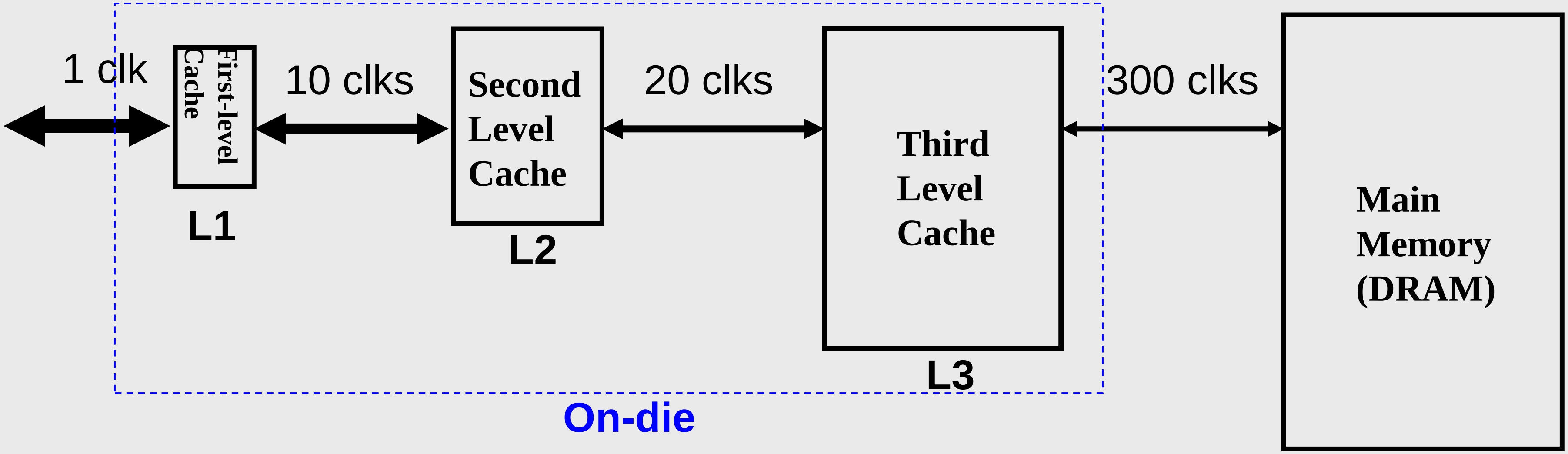
  $= HitCycle + MR \times (H_2 + MR_2 \times AMP_2)$

- $AMAT = HitCycle + MR \times (H_2 + MR_2 \times AMAT_3)$

  $= HitCycle + MR \times (H_2 + MR_2 \times (H_3 + MR_3 \times AMP_3))$

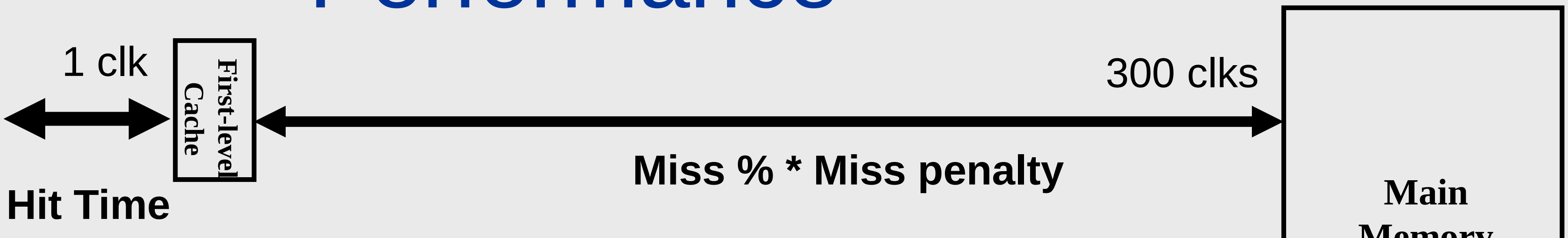
- Etc.

Reducing Penalty: Multi-Level Cache



- Average Memory Access Time (AMAT)
  $= T_{hit}(L1) +$
  $Miss\%(L1) * (T_{hit}(L2) + Miss\%(L2) * (T_{hit}(L3) + Miss\%(L3) * T(memory)))$
- Example: (Latency as shown above)
  - Miss rate: L1=10%, L2=5%, L3=1% (*Be careful miss rate definition*)
  - AMAT = 1 + 1 + 0.1 + 0.015
    = 2.115
  - 2.115 vs. 31 (if no L2, L3 memory hierarchy)

Memory Hierarchy Performance



- Average Memory Access Time (AMAT)

  = Hit Time + Miss rate * Miss Penalty

  = T_{hit}(L1) + Miss%(L1) * T (memory)
- Example:
  - Cache Hit = 1 cycle
  - Miss rate = 10%
  - Miss penalty = 300 cycles
  - AMAT = 1 + 300*10% = 31 cycles
- To further improve it?

Cache Performance

- CPU execution time =

  (CPU clock cycles + Memory stall cycles) * Clock cycle time

  = IC $\times$ $(CPI_{exe} + f_{mem} \times MR \times AMP)$ $\times$ Clock cycle time
- Memory stall cycles

  $= f_{mem} \times MR \times AMP$

  $= f_{mem} \times AMAT$ (if cache hit is not part of the computing)

- CPI = Ideal CPI + average stall cycles per instruction

- Memory stall cycles

  =IC * Reads per instruction * Read miss rate * Read miss penalty + IC * Writes per instruction * Write miss rate * Write miss penalty

Example 1 (CPU execution)

- Assume that we have a computer where
  - Ideal CPI = 1.0
  - 50% loads and stores
  - Miss penalty = 25 cycles and
  - Miss rate = 2%, how much faster would the computer be if all instructions were cache hits?
- CPU exec time (with no cache misses)= (IC x CPI + 0) x Clock cycle = IC x 1.0 x Clock cycle
- Memory Stall cycles = IC x (1 + 0.5) x 0.02 x 25

  = IC x 0.75

  CPU executime$_{cache}$ = (IC x 1.0 + IC x 0.75) x Clock cycle

  Performance ratio = 1.75

  Computer with no cache misses is 1.75 times faster

Example 2 (CPI calculation)

Ideal CPI	1.0
Data Miss	1.5
Inst Miss	0.5

- Suppose a processor executes at
  - Clock Rate = 1000 MHz (1 ns per cycle)
  - CPI = 1.0
  - 50% arithmetic/logic, **30**% load/store, 20% control
- Suppose that 10% of memory operations get 100 cycle miss penalty

- CPI = ideal CPI + average stalls cycles per instruction
  = 1.0(cycle)
  +(0.30 (data-operations/instruction)
  x 0.10 (miss/data-op) x 100 (cycle/miss))
  = 1.0 cycle + 3.0 cycle
  = 4.0 cycle
- 75 % of the time the processor is stalled waiting for memory!
- a 1% instruction miss rate would add an additional 1.0 cycles to the CPI!

Cache Design Considerations

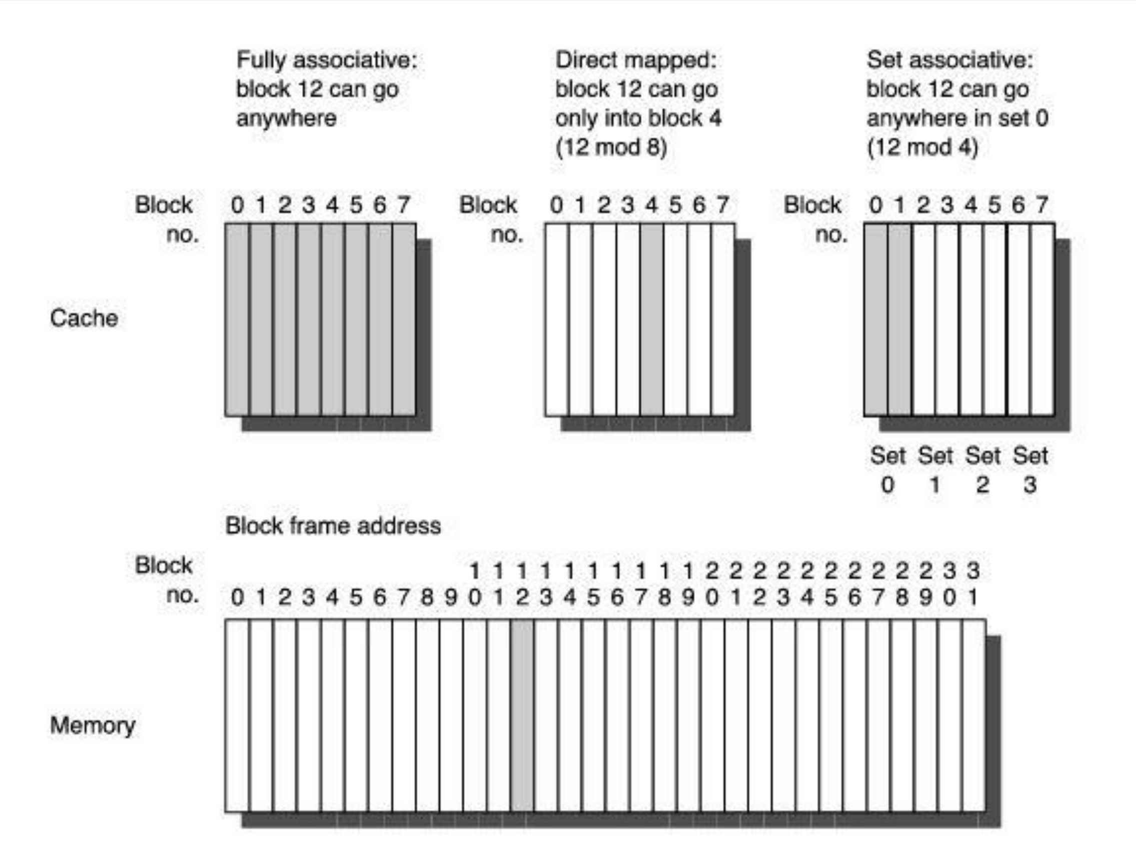
- Block size
  - What is the atomic unit of storage in the cache?
- Block placement
  - Where can a block be placed in the cache?
- Block identification
  - How is the block found in the cache?
- Block replacement
  - Which block should be replaced on a miss?
- Write strategy
  - What happens on a write?

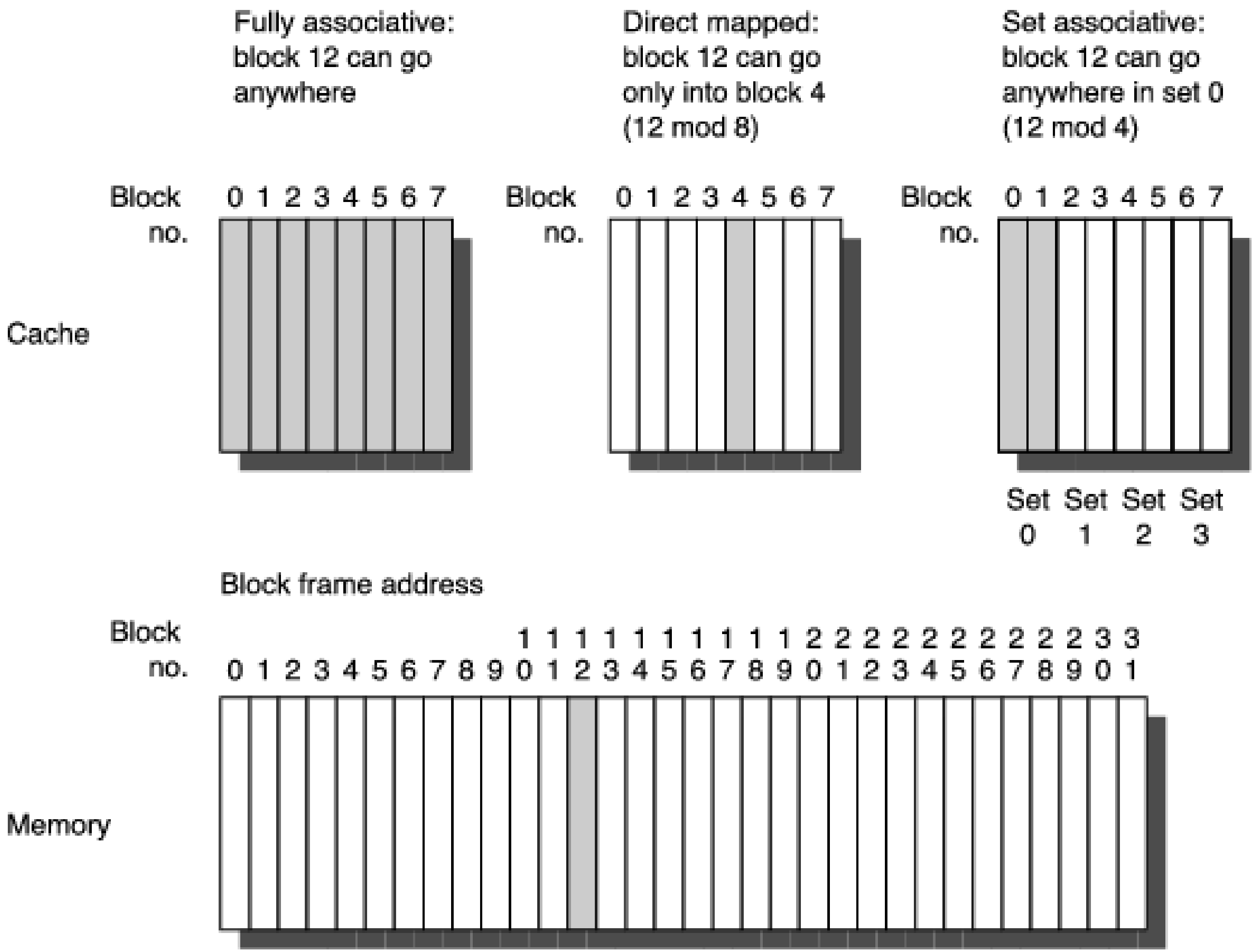
Block size

- Block size
  - Typical values for L1 cache: 32 to 128 bytes
- Exploit spatial locality:
  - Bring in larger blocks
    - Slows down time it takes to fix a miss
    - Too large and “hog” storage (“cache pollution”)

Cache Associativity

- Fully associative: Block can go anywhere in cache
- Direct Mapping: Block can only go in one place in the cache
- N-way associative: Block can go in one of a set of places in the cache
  - A set is a group of blocks in the cache



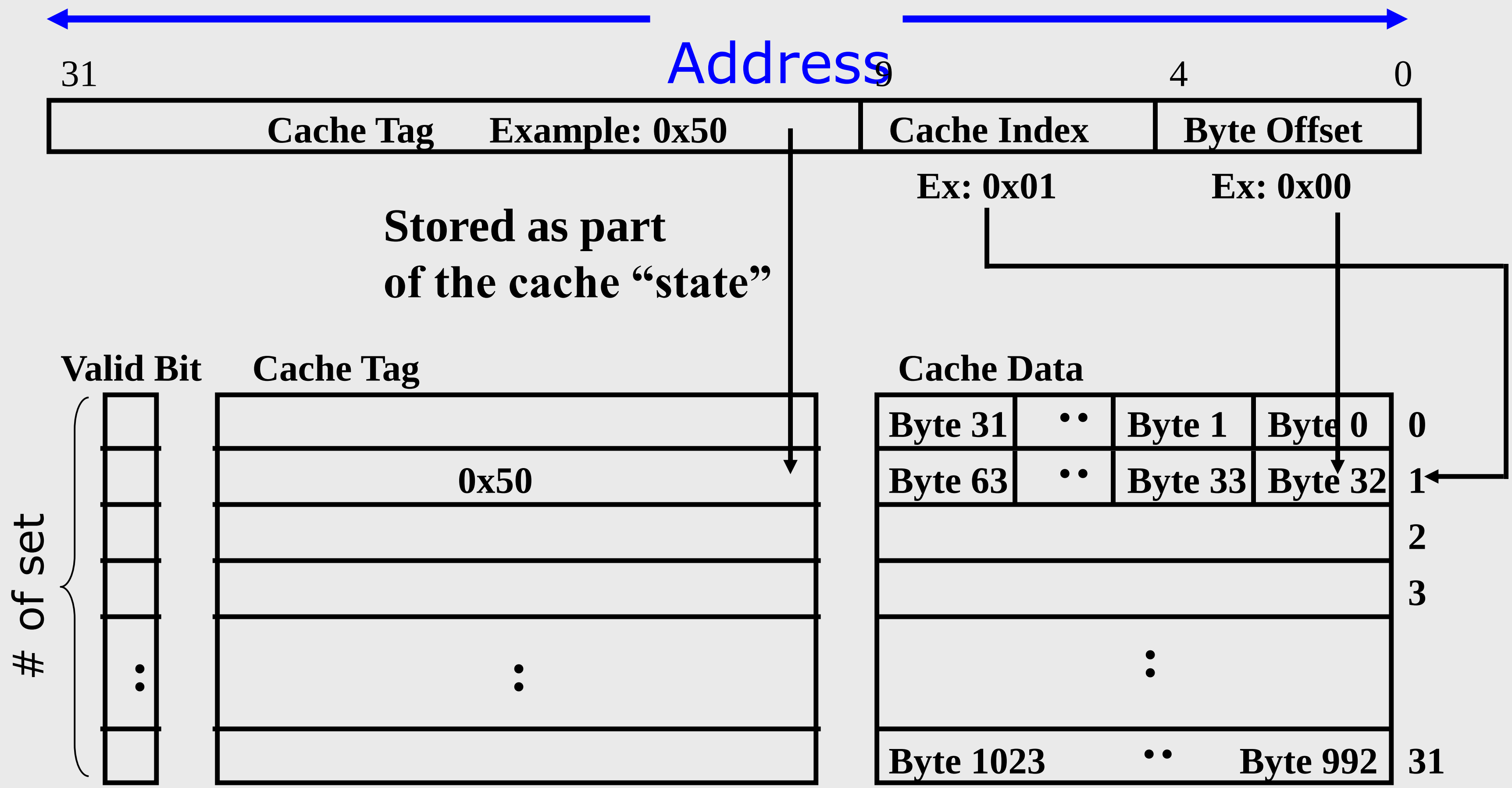

Fully associative:
block 12 can go
anywhere
Direct mapped:
block 12 can go
only into block 4
(12 mod 8)
Set associative:
block 12 can go
anywhere in set 0
(12 mod 4)
Block no.
0 1 2 3 4 5 6 7
Cache
Set 0 Set 1 Set 2 Set 3
Block frame address
Memory

How is a block found if it is in the cache?

- Tag to check all blocks in the set, index to select the set, and offset to select desired data with in the block
- Tag on each block
  - Valid bit – If not set, there cannot be a match on this address
  - No need to check index or block offset

Block Address		Block Offset
Tag	Index	

Example: 1KB Direct mapped Cache, 32-byte Lines



Replacement Policy

- FIFO
- Random
- LRU (Least Recently Used)

	Associativity					
	Two-way			Four-way		
Size	LRU	Random	FIFO	LRU	Random	FIFO
16 KB	114.1	117.3	115.5	111.7	115.1	113.3
64 KB	103.4	104.3	103.9	102.4	102.3	103.1
256 KB	92.2	92.1	92.5	92.1	92.1	92.5

Data cache misses per 1000 instructions (SPEC2000 benchmarks)

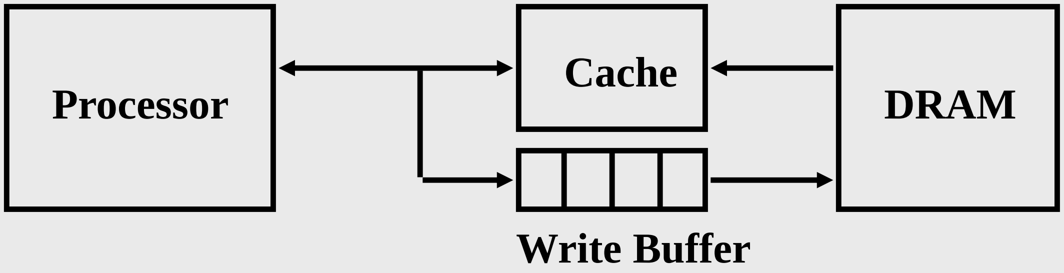
Write Policy

- 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.
  - is line clean or dirty?
  - Write back buffer is typically employed to perform the operation in the background

Write back – Write through

- Pros and Cons of each?
  - WT: Easier to implement
  - WT: Lower level memory has the most recent copy of data
  - WB: writes occur at the speed of cache memory
  - WB: no repeated writes to same location
  - WB: saves power
- CPU must wait for writes to complete during WT
  - Write stall
- WT always combined with write buffers so that no need to wait for lower level memory

Write Buffer for Write Through



- A Write Buffer is needed between the Cache and Memory
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory
- Write buffer is just a FIFO:
  - Typical number of entries: 4 to 8
  - Works fine if: Store frequency (w.r.t. time) << 1 / DRAM write cycle
- Memory system designer's nightmare:
  - Store frequency (w.r.t. time) -> 1 / DRAM write cycle
  - Write buffer saturation