Caching and the Memory Hierarchy

Professors Hugh C. Lauer and Thérèse Smith

CS-4515, Computer Architecture

Homework #2

- **Do one of the following exercises on Pipelining:–**
  - Exercise C.2:– Branch frequencies
  - Exercise C.5:– Register-memory architecture (e.g., VAX 8700)
  - Exercise C.9:– The DAXPY loop

Reading — Memory Hierarchy

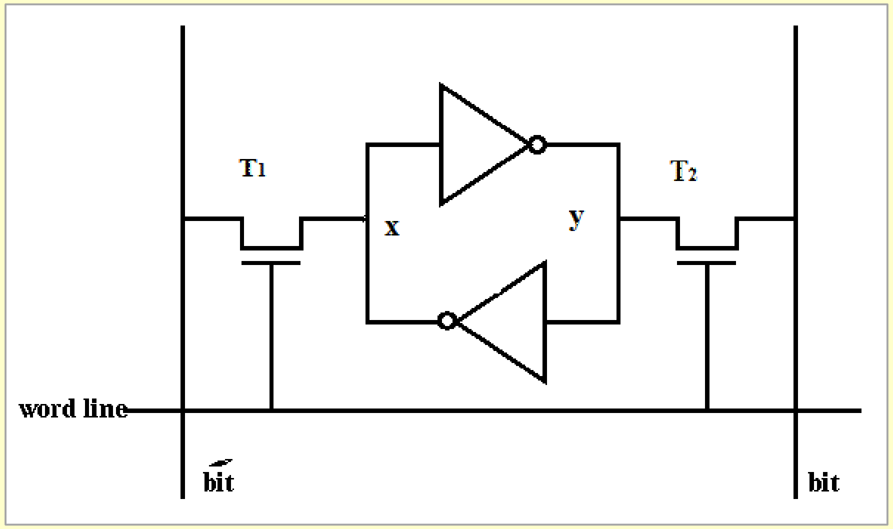
- **All of Appendix B (first)**
- **All of Chapter 2 (next)**
  - Especially 2.1, 2.4, 2.6

Memory Technology

- S-RAM — Static Random Access Memory
- D-RAM — Dynamic Random Access Memory
- Flash Memory
- Disks and other electro-mechanical devices

Static RAM

- **Array of cells**
- **Power flows constantly**
- **Data retained till …**
  - Power off
  - Overwritten
- **Very fast**
  - Read
  - Write

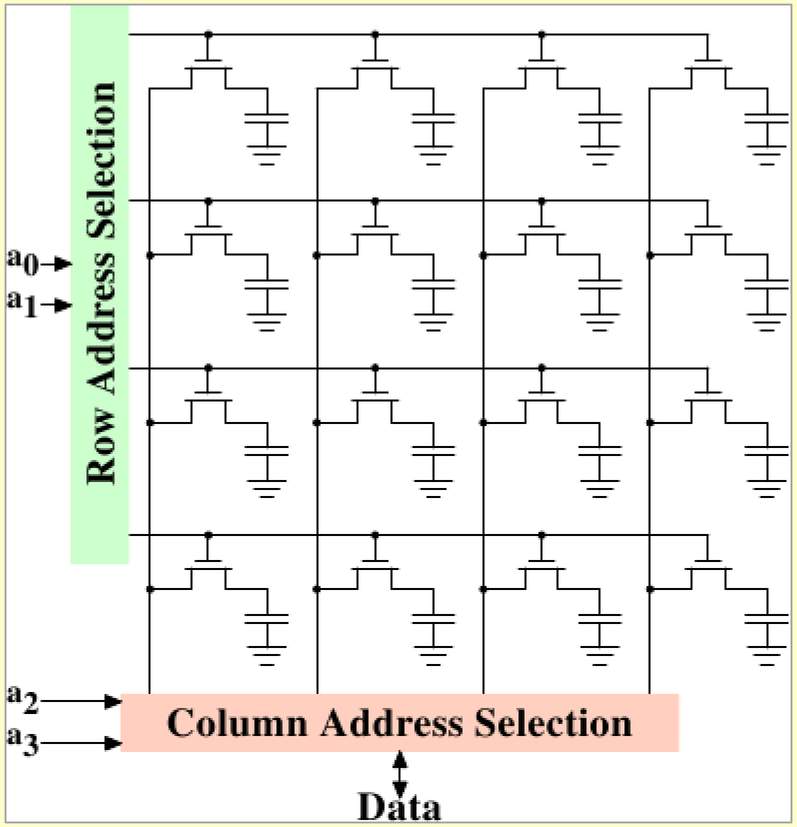


- **Used for**
  - Cache memories
  - D-RAM buffers
  - Other high-performance needs

Page 97 of 5th edition

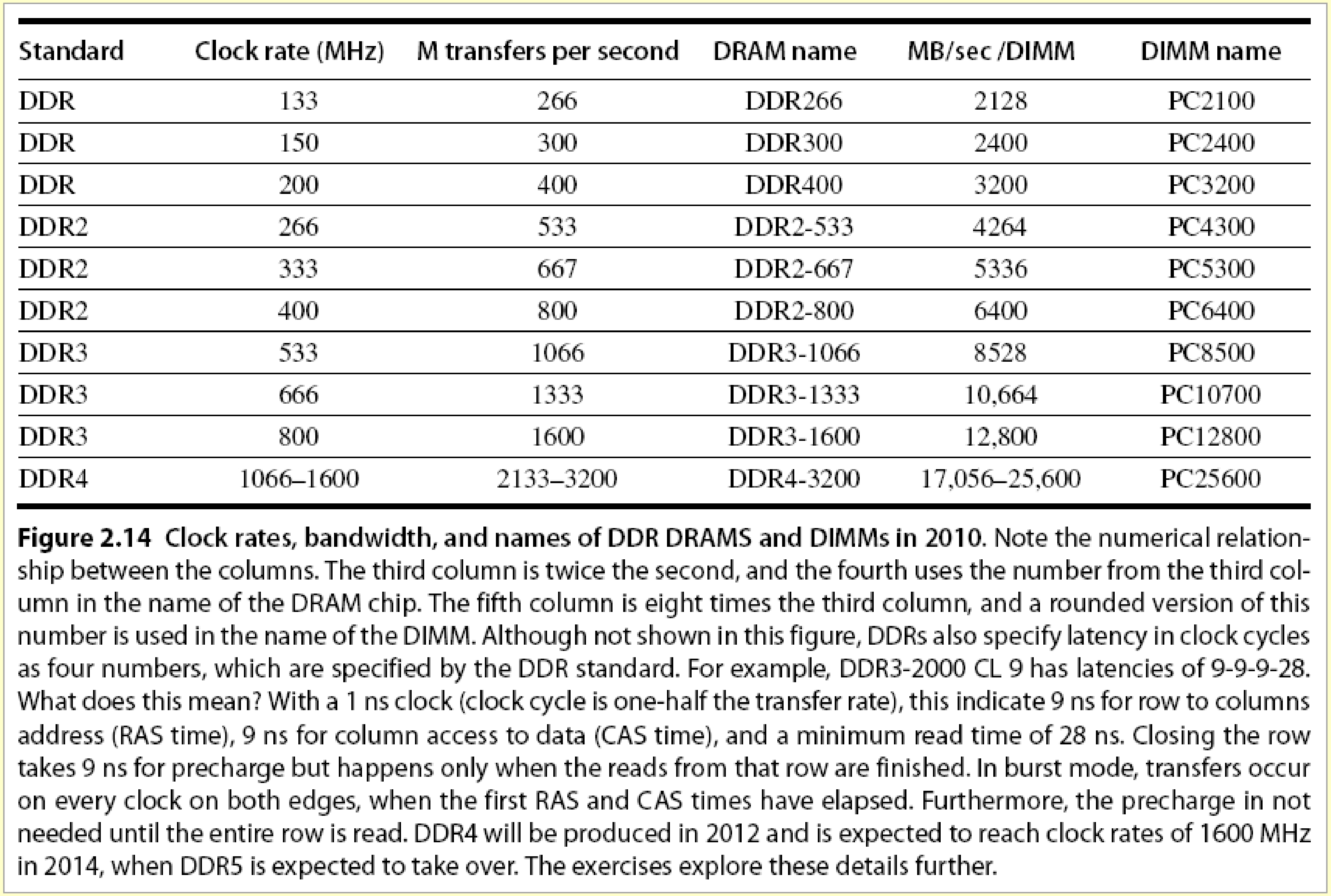
Dynamic RAM

- 1 transistor & 1 capacitor per cell
- Data stored as charge on capacitor
  - Much slower than S-RAM
  - Much denser than S-RAM
- Capacitors discharged on read
  - Must be rewritten after each read
- Capacitors "leak" over time
  - Must be recharged periodically



- Used for
  - Main memories
  - Graphics memories
  - High density, low power

Page 98-102 of 5th edition



Standard	Clock rate (MHz)	M transfers per second	DRAM name	MB/sec /DIMM	DIMM name
DDR	133	266	DDR266	2128	PC2100
DDR	150	300	DDR300	2400	PC2400
DDR	200	400	DDR400	3200	PC3200
DDR2	266	533	DDR2-533	4264	PC4300
DDR2	333	667	DDR2-667	5336	PC5300
DDR2	400	800	DDR2-800	6400	PC6400
DDR3	533	1066	DDR3-1066	8528	PC8500
DDR3	666	1333	DDR3-1333	10,664	PC10700
DDR3	800	1600	DDR3-1600	12,800	PC12800
DDR4	1066–1600	2133–3200	DDR4-3200	17,056–25,600	PC25600

Figure 2.14 Clock rates, bandwidth, and names of DDR DRAMS and DIMMs in 2010. Note the numerical relationship between the columns. The third column is twice the second, and the fourth uses the number from the third column in the name of the DRAM chip. The fifth column is eight times the third column, and a rounded version of this number is used in the name of the DIMM. Although not shown in this figure, DDRs also specify latency in clock cycles as four numbers, which are specified by the DDR standard. For example, DDR3-2000 CL 9 has latencies of 9-9-9-28. What does this mean? With a 1 ns clock (clock cycle is one-half the transfer rate), this indicate 9 ns for row to columns address (RAS time), 9 ns for column access to data (CAS time), and a minimum read time of 28 ns. Closing the row takes 9 ns for precharge but happens only when the reads from that row are finished. In burst mode, transfers occur on every clock on both edges, when the first RAS and CAS times have elapsed. Furthermore, the precharge in not needed until the entire row is read. DDR4 will be produced in 2012 and is expected to reach clock rates of 1600 MHz in 2014, when DDR5 is expected to take over. The exercises explore these details further.

Figure 2.5 of 6th edition

Recent DDR SD-Rams

Production Year	Chip Size	DRAM Type	RAS time (ns)	CAS Time (ns	Total (ns)	Total w/ precharge
2000	256 Mbit	DDR1	21	21	42	63
2002	512 Mbit	DDR1	15	15	30	45
2003	1 Gbit	DDR2	15	15	30	45
2006	2 Gbit	DDR2	10	10	20	30
2010	4 Gbit	DDR3	7	7	14	21
2016	8 Gbit	DDR4	14	14	28	42

Figure 2.4

Flash RAM

- **Non-volatile**
  - Holds data when power off
- **Written in large blocks**
  - Entire block at one time
  - Must be "erased" before re-writing
- **Wears out after ~10^5 erase/write cycles**
- **Requires support circuitry to**
  - Manage blocks
  - Map logical to physical blocks
  - Cache data for writing, reading

- **Used for**
  - Solid state "hard drive"
  - Personal mobile devices
  - Vibration environments

Page 92-93

Challenges

- **Fast memories are not large enough (S-RAMs)**
- **Large memories are not fast enough (D-RAMs)**
- **Non-volatile memories are**
  - Mechanical (Disks, etc.) — many read-write cycles
  - Solid state (Flash drives) — limited read-write duty

All have places in the Memory Hierarchy

From CS-2011 and CS-3013

- What is a *cache?*
- Why do we have them?
- Can you give some examples of caches in systems and programs?
- Why do they work?
- How do caches affect program design?

Memory Hierarchy in Processors

- **Organized into multiple levels**
- **Each level is smaller faster and more expensive per byte than the next lower level**
- **Lower level is farther from the action of a processor**
  - Longer latency to access

What is difference between speed and latency?

Basic Memory Hierarchy (continued)

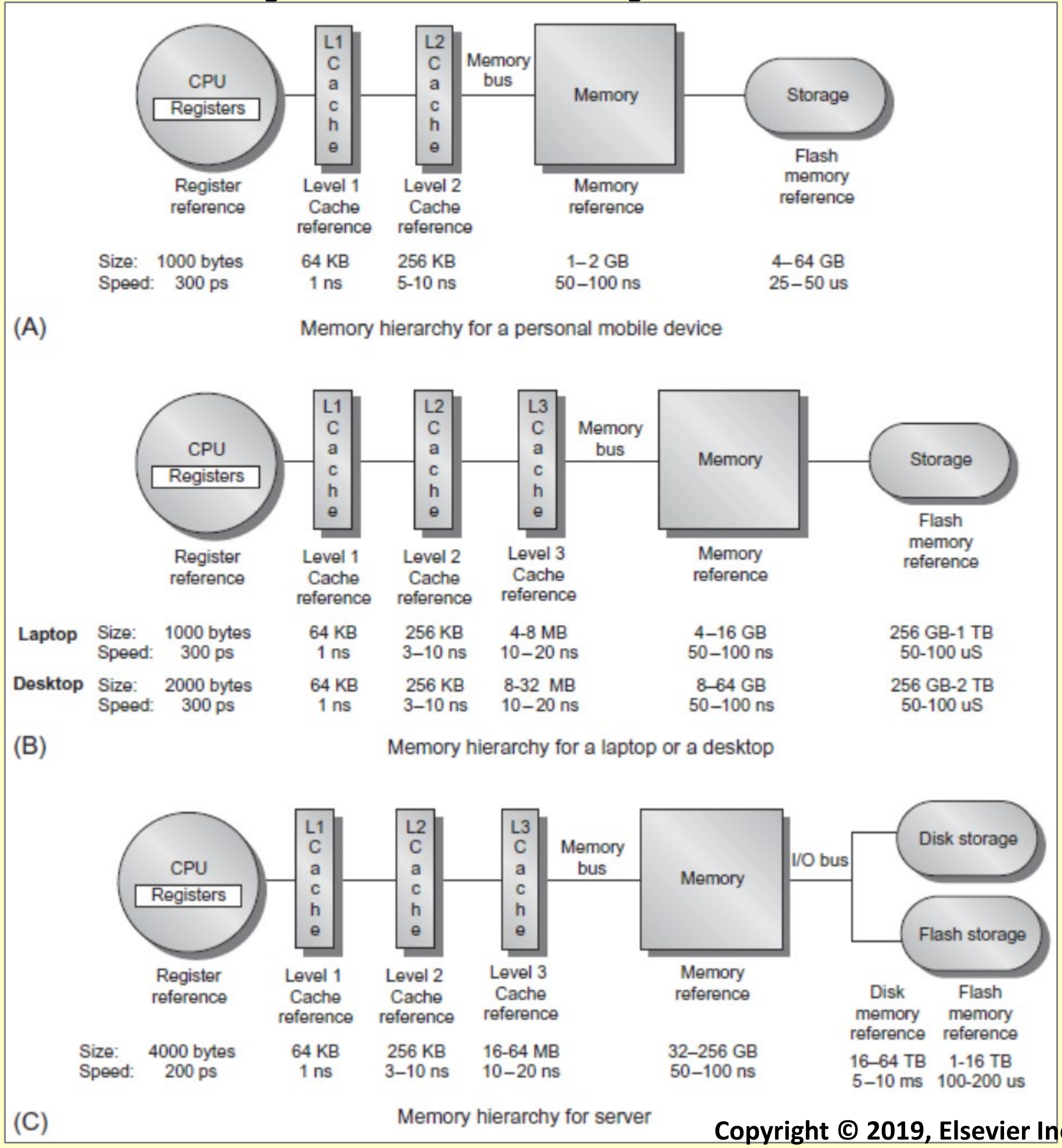


Figure 2.1

Performance Reality

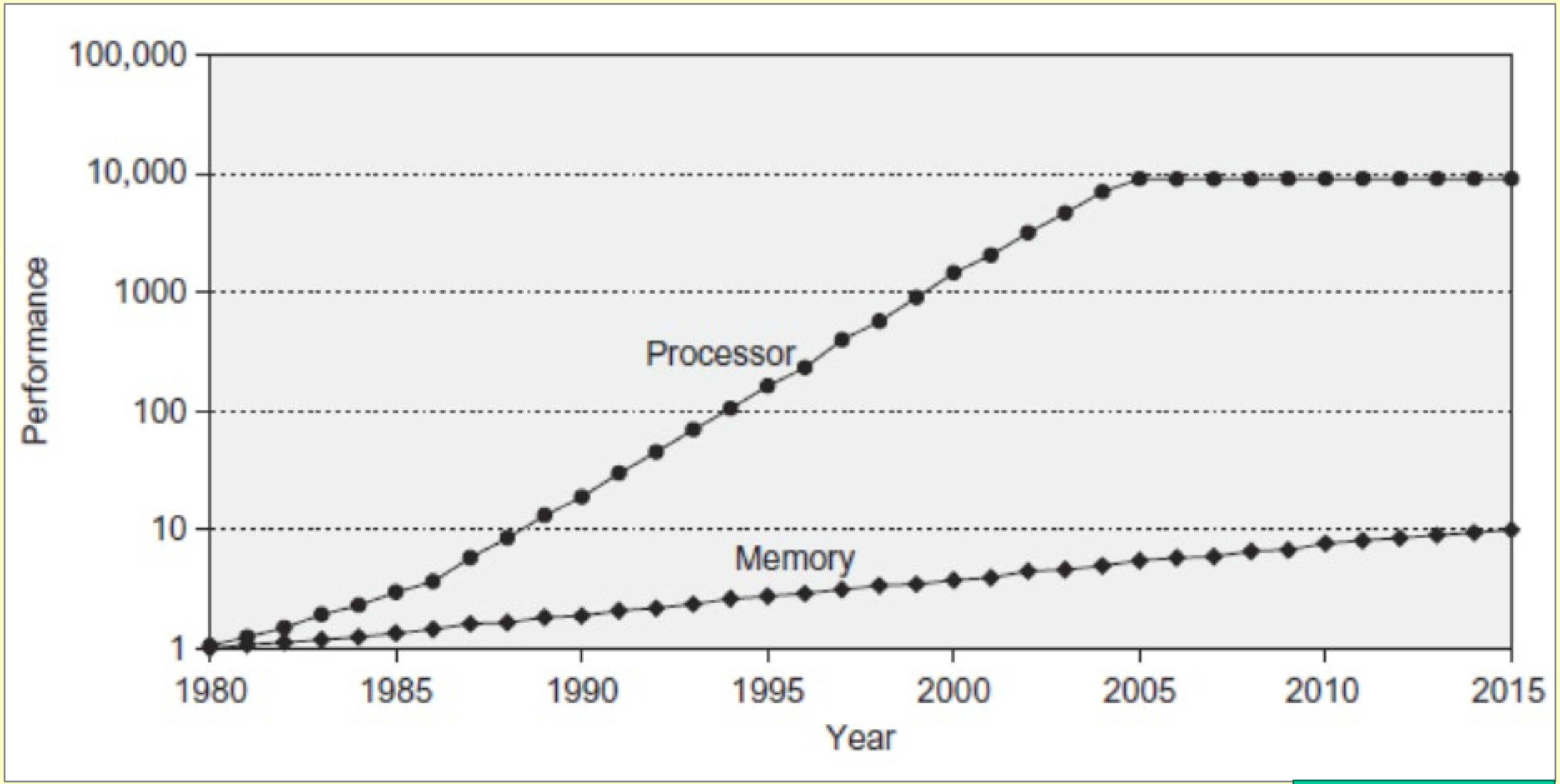


Figure 2.2

Goal

- To provide a memory system with cost per byte of the cheapest level and the speed approaching the fastest level …
- … by keeping the most frequently used information in fastest memory
- … i.e., in caches!
- … and by moving data between levels of hierarchy as needed

Cache Performance

- **Miss Rate**
  - Fraction of memory references not found in cache (misses / accesses) = 1 – hit rate
  - Typical numbers (in percentages):
    - 3-10% for L1
    - can be quite small (e.g., < 1%) for L2, depending on size and other characteristics
- **Hit Time**
  - Time to deliver a data item from cache to processor
    - includes time to determine *whether* item is in the cache
  - Typical numbers:
    - 1-2 clock cycle for L1
    - 5-20 clock cycles for L2
- **Miss Penalty**
  - Additional time required because of a miss
    - typically 50-200 cycles for main memory (Trend: increasing!)

Cache Performance (continued)

- **Average access time =**
  - Hit time + miss_rate × miss penalty
- **Example**
  - Hit time for L1 cache = 1 cycle
  - Miss penalty for L1 cache = 10 cycles
  - Miss rate = 10%
  - $\Rightarrow$ Average access time = 1 + 0.1 * 10 = 2 cycles
- **Example 2**
  - Miss rate = 1%
  - $\Rightarrow$ Average access time = 1 + 0.01 * 10 = 1.1 cycles

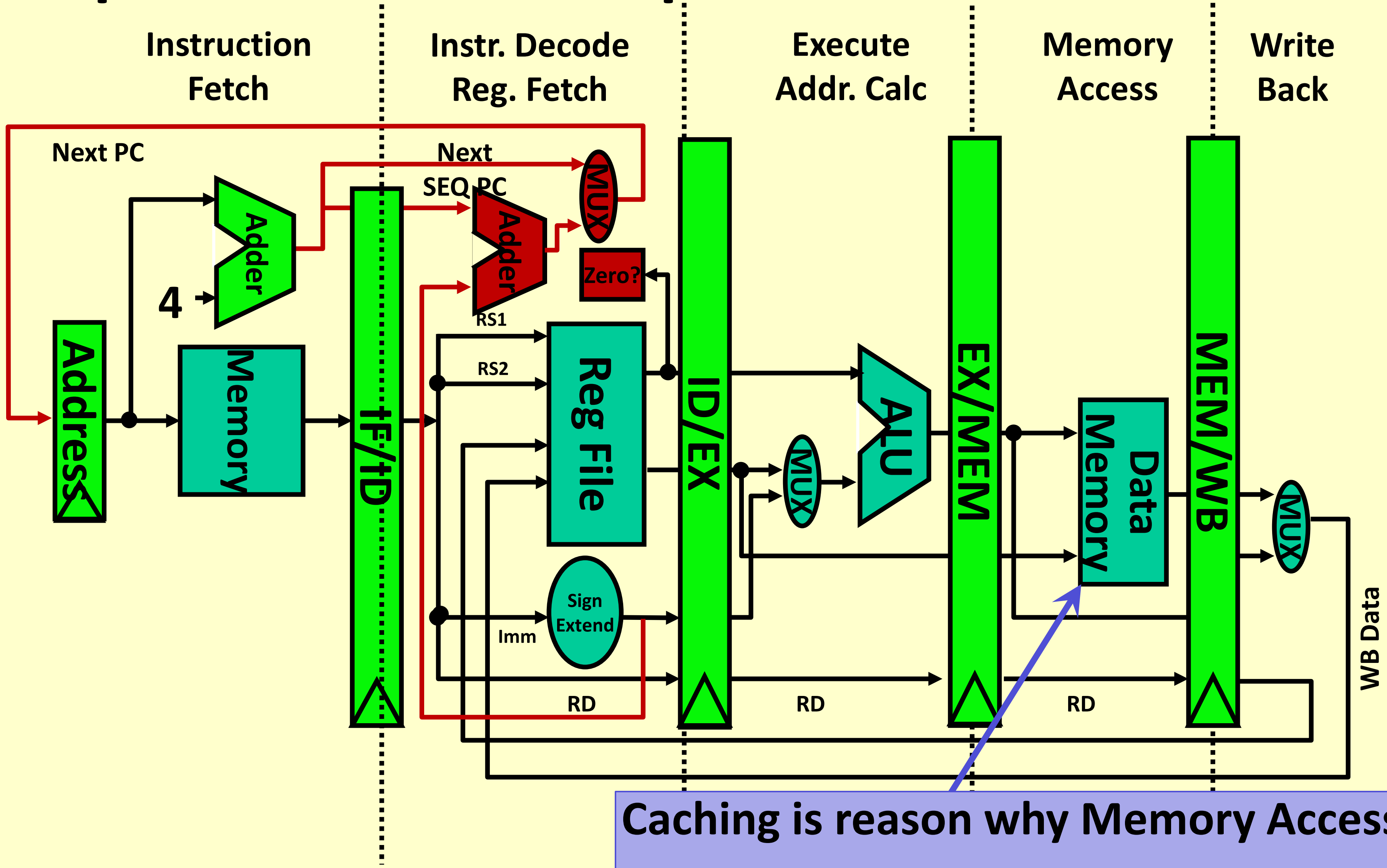
Think about those numbers

- **Huge difference between a hit and a miss**
  - Could be 100x, if just L1 and main memory
- **Would you believe 99% hits is twice as good as 97%?**
  - Consider:
    cache hit time of 1 cycle
    miss penalty of 100 cycles
  - Average access time:

    97% hits: 1 cycle + 0.03 * 100 cycles = **4 cycles**

    99% hits: 1 cycle + 0.01 * 100 cycles = **2 cycles**
- **This is why "miss rate" is used instead of "hit rate"**

Pipelined MIPS Datapath (Figure C.25, page C-38)



Caching is reason why Memory Access stage can be done in one cycle!

From CS-2011

Bryant & O'Hallaron, Ch 6

- **Programmers need to understand about caches**
  - Impact on program performance

- **E.g., matrix multiplication:–**

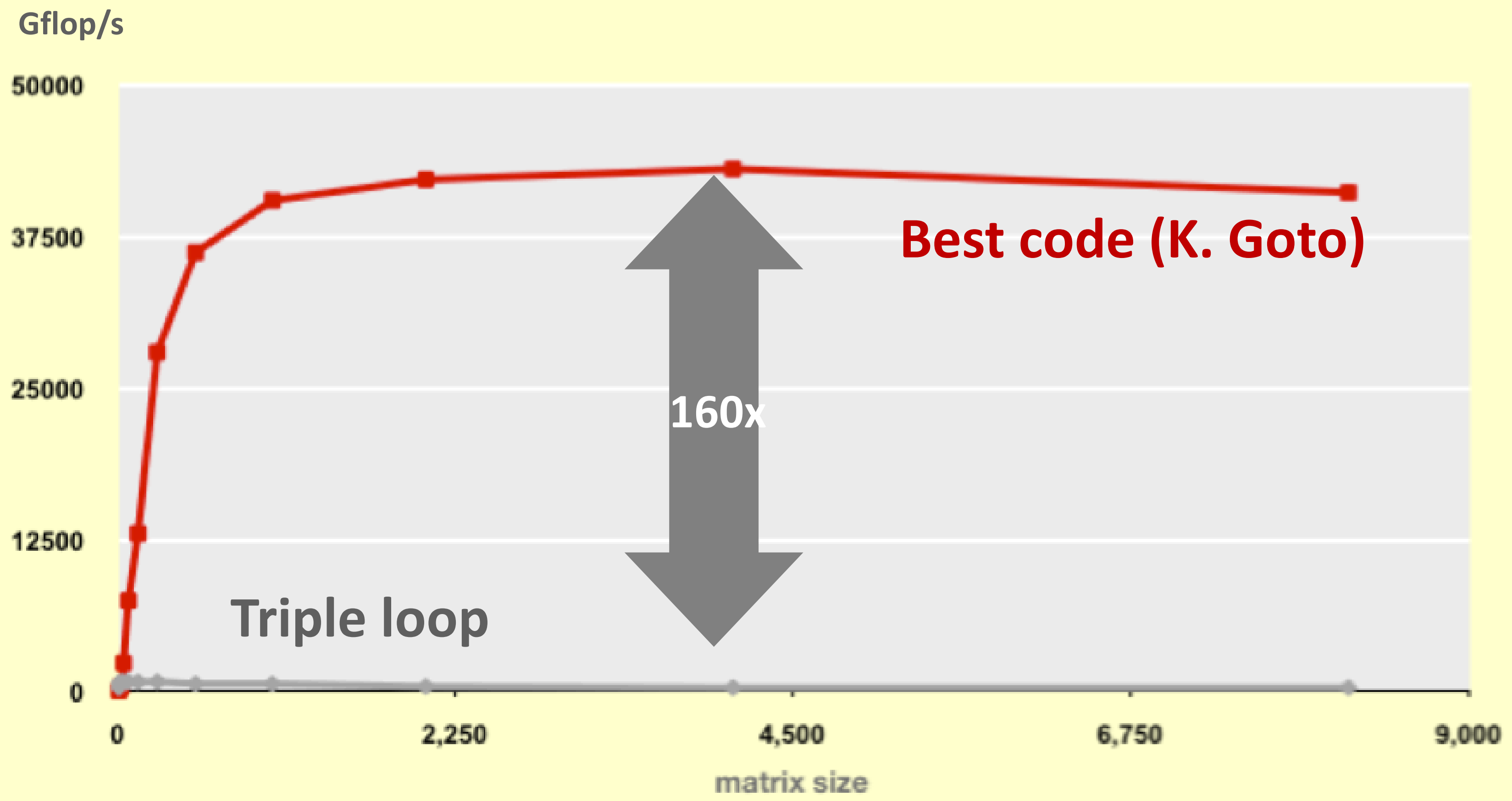
```
for (i=0; i<n; i++) {
  for (j=0; j<n; j++) {
   sum = 0.0;
   for (k=0; k<n; k++)
     sum += a[i][k] * b[k][j];
   c[i][j] = sum;
  }
}
```

```
for (k=0; k<n; k++) {
 for (i=0; i<n; i++) {
  r = a[i][k];
  for (j=0; j<n; j++)
   c[i][j] += r * b[k][j];
 }
}
```

CS-2011, First Lecture

Example Matrix Multiplication

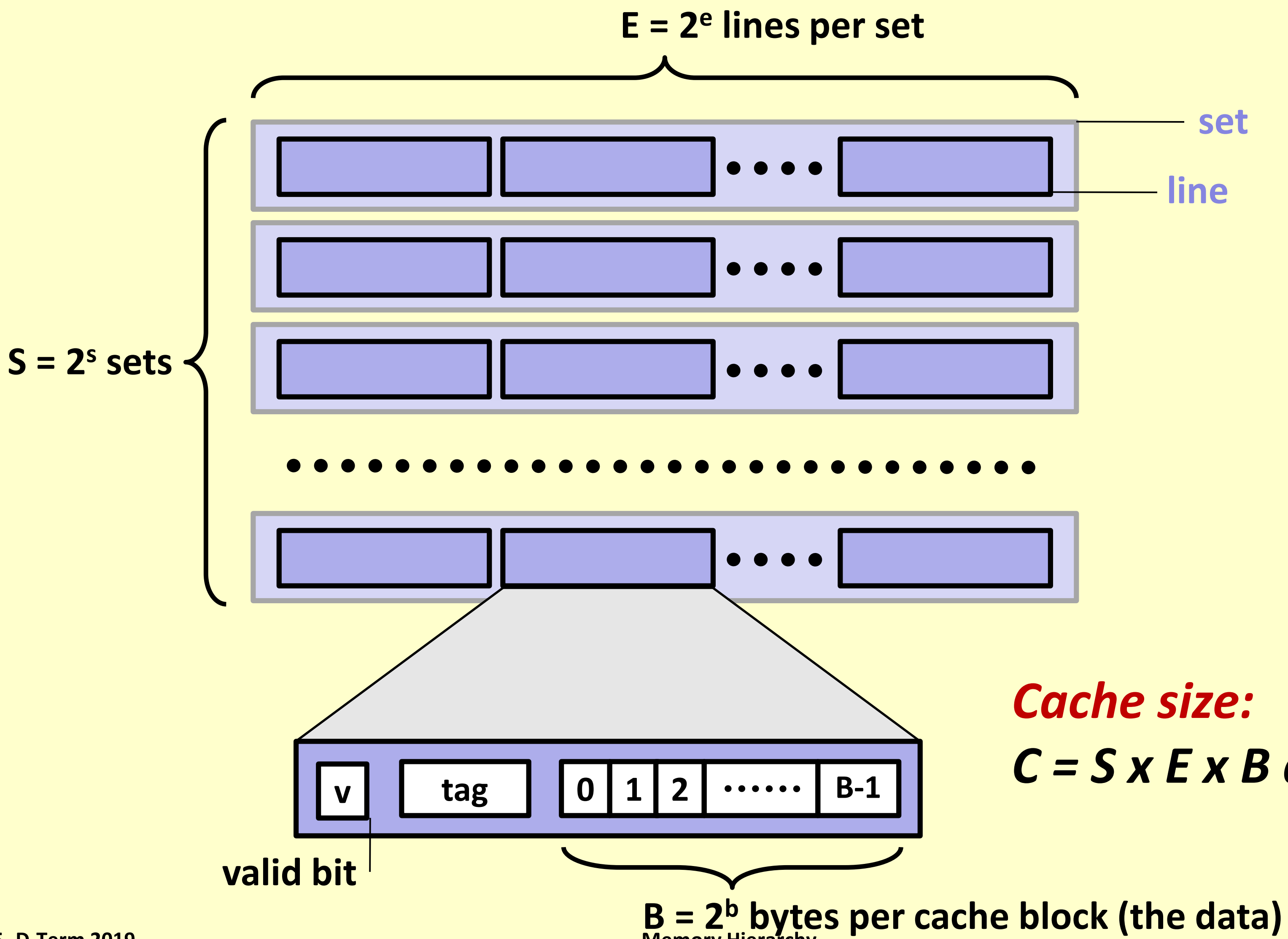
Matrix-Matrix Multiplication (MMM) on 2 x Core 2 Duo 3 GHz (double precision)



- Standard desktop computer, vendor compiler, using optimization flags
- Both implementations have exactly the same operations count ($2n^3$)
- What is going on?

Questions?

Cache Organization (S, E, B)

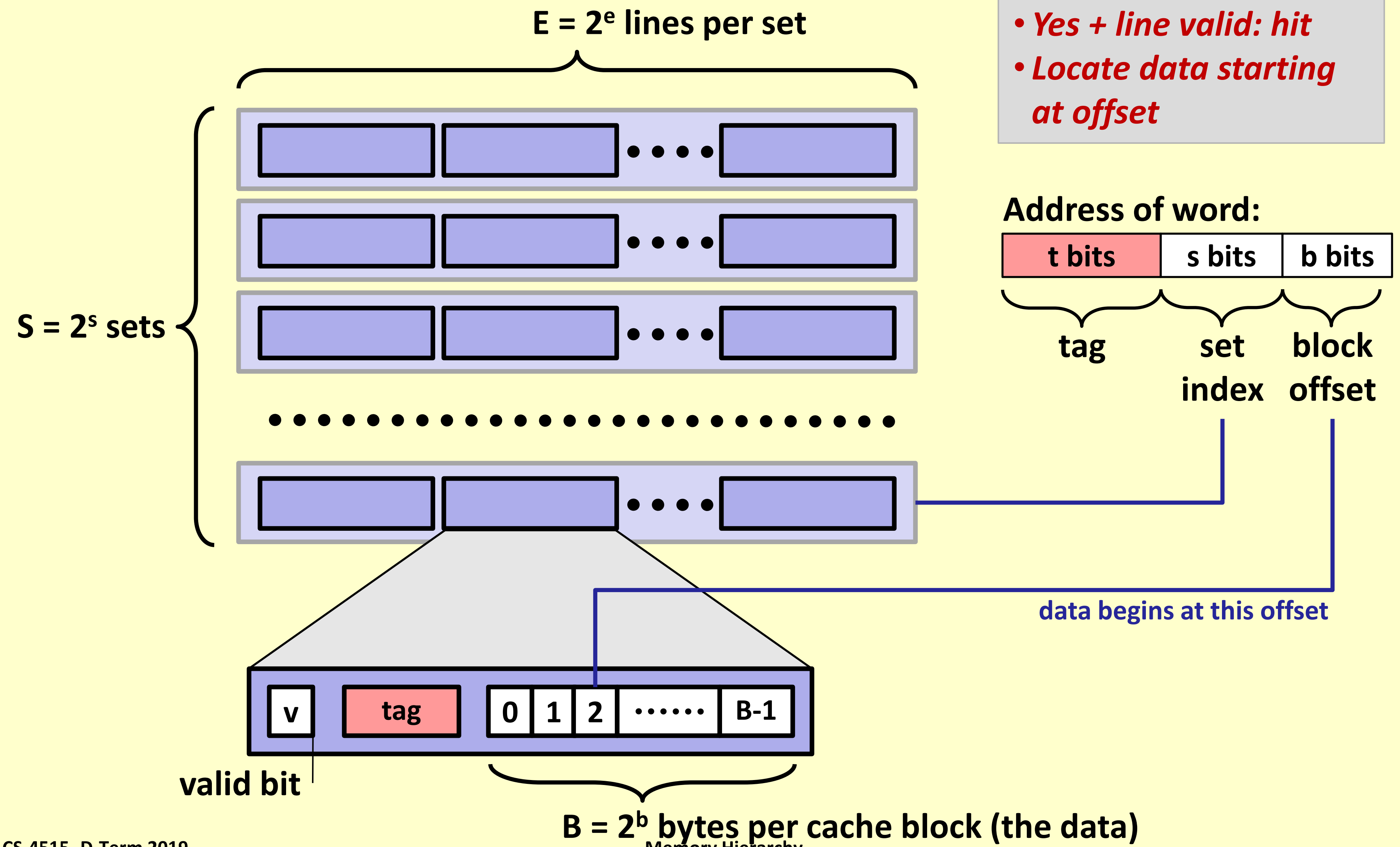


Cache size:

C = S x E x B data bytes

Cache Read

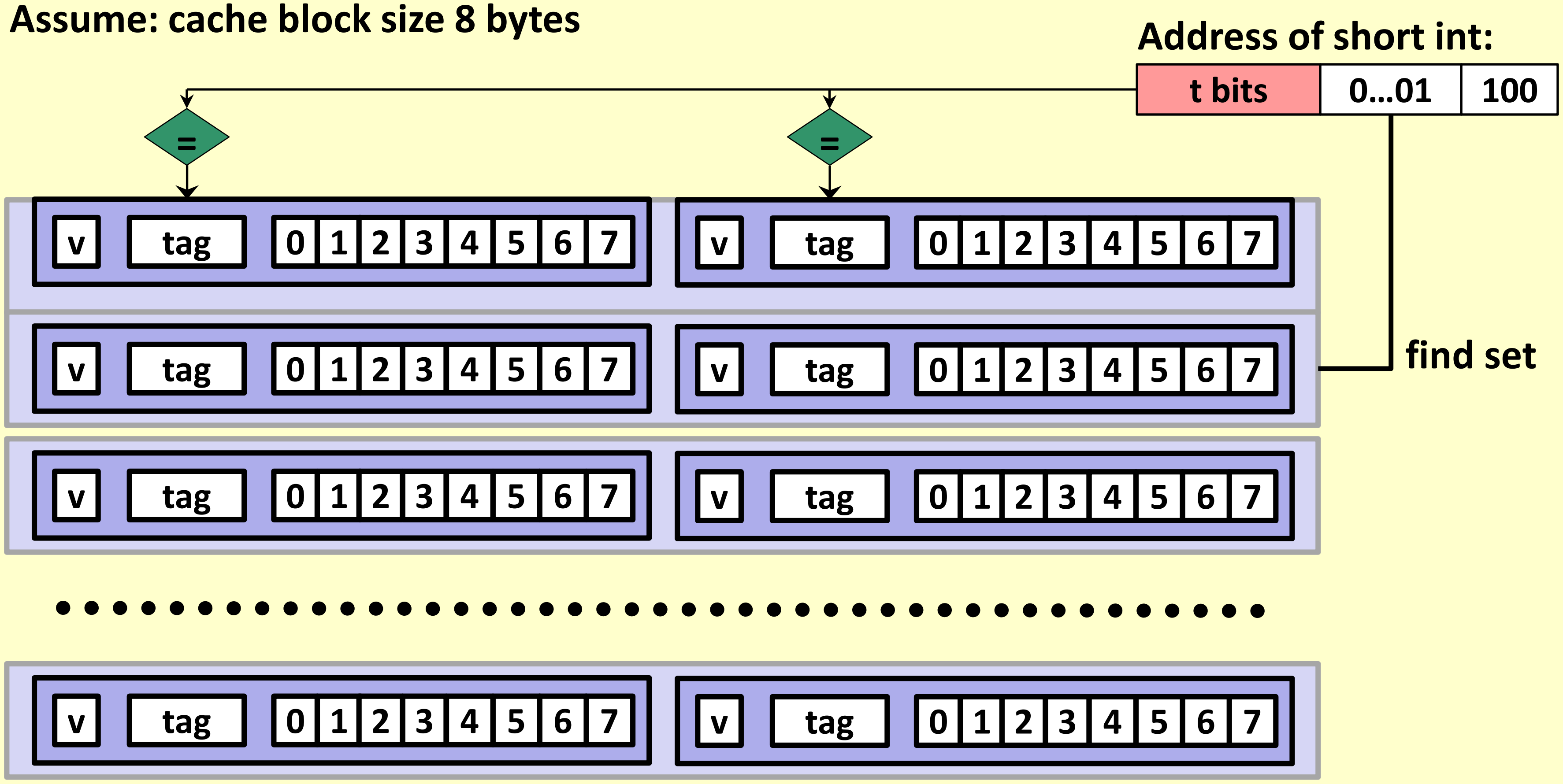
- *Locate set*
- *Check if any line in set has matching tag*
- *Yes + line valid: hit*
- *Locate data starting at offset*



E-way Set Associative Cache (Here: E = 2)

E = 2: Two lines per set

Assume: cache block size 8 bytes



Enough circuitry to check all tags in set at same time!

Advantage

- **Can find and validate cache entry without searching**
  - Usually 1 cycle (for L1 caches)

- **Detect cache-miss in one cycle**

Multi-level Caches

- **If item is in L1, use it**
  - At cost of L1 cache hit
- **If not, fetch from L2**
  - At cost of L1 cache miss, L2 cache hit
- **If not in L2, fetch from L3 (or next layer in hierarchy)**
  - At cost of L2 cache miss, L3 cache hit
- **…**

Cache Performance (again)

- **Average access time at L_i**
  - Hit time L_i + miss_rate$_i$ × miss penalty$_i$
- **Miss penalty$_i$**
  - Hit time L_{i+1} + miss_rate$_{i+1}$ × miss penalty$_{i+1}$
- **Miss penalty$_{i+1}$**
  - Hit time L_{i+2} + miss_rate$_{i+2}$ × miss penalty$_{i+2}$
- **Miss penalty$_{i+2}$**
  - Hit time L_{i+3} + miss_rate$_{i+3}$ × miss penalty$_{i+3}$

Cache Performance (again)

- **Average access time at L_i**
  - Hit time L_i + miss_rate$_i$ × miss penalty$_i$
- **Miss penalty$_i$**
  - Hit time L_{i+1} + miss_rate$_{i+1}$ × miss penalty$_{i+1}$
- **Miss penalty$_{i+1}$**
  - Hit time L_{i+2} + miss_rate$_{i+2}$ × miss penalty$_{i+2}$
- **Miss penalty$_{i+2}$**
  - Hit time L_{i+3} + miss_rate$_{i+3}$ × miss penalty$_{i+3}$

- L1 cache
  - Hit time – 1 cycle

- L2 cache
  - Hit time – 50 cycles

- DRAM
  - Hit time – 200 cycles

- Page fault
  - Hit time – 10 msec!

What about writing?

- **If item is already in cache**
  - Do you "write through" immediately to next level …

    OR

  - … "write back" to cache and update next level later?

- **If item is *not* in cache when you try to write**
  - Do you take a "write-allocate cache miss" for item,

    OR

  - … write directly to next level with no cache miss?

Typically

- **“Write through” is *not* accompanied by cache miss**
  - I.e., no need to fetch a cache line if all you are doing is updating it!

- **“Write back” requires**
  - “Write allocate miss” to fill out rest of cache line
  - “Dirty bit” to indicate cache line must be written before being reclaimed

Write buffer

- **Most processors support “write through” with a “write buffer”**
  - I.e., a queue of data items to be written to memory
- **Searchable on cache misses**
  - To bring recently “written” items into cache
- **Merge related writes in buffer**
  - To improve sequential writes

The Three Cs Model

- **Three categories of Cache Misses**
  - Compulsory
  - Capacity
  - Conflict

The Three Cs Model (continued)

- **Compulsory Misses**
  - The very first access to a block cannot be in the cache.
  - Need to bring the data to the cache first
  - These will happen no matter how large the cache is.
- **Capacity Misses**
  - If a cache is not large enough to hold all needed blocks
- **Conflict Misses**
  - If “set” is not large enough to hold all needed blocks with same index

The 4th C

- **Coherency misses**
  - Keep two caches in synch when sharing same memory
  - E.g., 2-4 cores, each with its own L1 and L2 caches
- **(Formerly) esoteric topic relegated to Chapter 5**
- **(Currently) front-page issue with multi-core processors**
  - See especially §5.2, §5.5

Questions?

Cache Optimizations

1. Larger block size to reduce miss rate
2. Bigger Caches to reduce miss rate
3. Higher associativity to reduce miss rate
4. Multilevel caches to reduce miss penalty
5. Giving priority to read misses over writes to reduce miss penalty
6. Avoiding address translation during indexing of the cache to reduce hit time

Section §B.3 quantifies these — important reading

1. Larger Block Size to Reduce Miss Rate

- **Easiest way to reduce miss rate**
  - Uses spatial locality and increased block size
- **Reduces compulsory misses but increases miss penalty**
- **Larger blocks**
  - Fewer tags which may slightly reduce static power
  - May increase capacity and conflict misses in smaller caches
- **Block size depends on cache size and miss penalty**

2. Bigger Caches to Reduce Miss Rate

- Reduces capacity misses
- May increase hit times of larger cache memory, power, and cost
- Larger caches increase static and dynamic power

3. Higher Associativity to Reduce Miss Rate

- Reduces conflict misses
- Increased associativity may increase hit time and power consumption

4. Multilevel Caches to Reduce Miss Penalty

- **Tough choice:–making hit time fast *vs.* making larger caches**
  - Adding levels of cache between original cache and main memory simplify the decision
  - 1st level of cache can match clock speed
  - 2nd (or 3rd) can capture accesses that would go to the main memory
- **Focus on misses in the 2nd level leads to bigger blocks and capacity, and higher associativity**
- **Multilevel Caches are more power efficient than a single cache**

5. Priority to Read Misses over Writes

- **Write buffer is a good place to implement this**
  - They create hazards because they hold updated location needed on read misses
    - Called read-after-write hazard through memory
- **Solution: check write buffer on read miss**
  - If there are no conflicts and the memory system is available
    - Sending read before write reduces miss penalty
- **Most processors give priority to read over write**
- **This has little effect on power consumption**

6. Avoid Address Translation on Cache Miss

- **Programs "speak" virtual addresses**
  - Caches deal in physical addresses
- **Avoid TLB lookup in critical path of cache access**

- **Common optimization:– use page offset to index the cache**
  - Identical in both addresses
- **Impacts L1 size**

Cache Optimizations

1. **Larger block size to reduce miss rate**
2. **Bigger Caches to reduce miss rate**
3. **Higher associativity to reduce miss rate**
4. Multilevel caches to reduce miss penalty
5. Giving priority to read misses over writes to reduce miss penalty
6. Avoiding address translation during indexing of the cache to reduce hit time

TANSTAAFL

Cache Optimizations

1. Larger block size to reduce miss rate
2. Bigger Caches to reduce miss rate
3. Higher associativity to reduce miss rate
4. **Multilevel caches to reduce miss penalty**
5. **Giving priority to read misses over writes to reduce miss penalty**
6. Avoiding address translation during indexing of the cache to reduce hit time

Routinely part of most modern processors

Cache Optimizations

1. Larger block size to reduce miss rate
2. Bigger Caches to reduce miss rate
3. Higher associativity to reduce miss rate
4. Multilevel caches to reduce miss penalty
5. Giving priority to read misses over writes to reduce miss penalty
6. **Avoiding address translation during indexing of the cache to reduce hit time** **<— next slide**

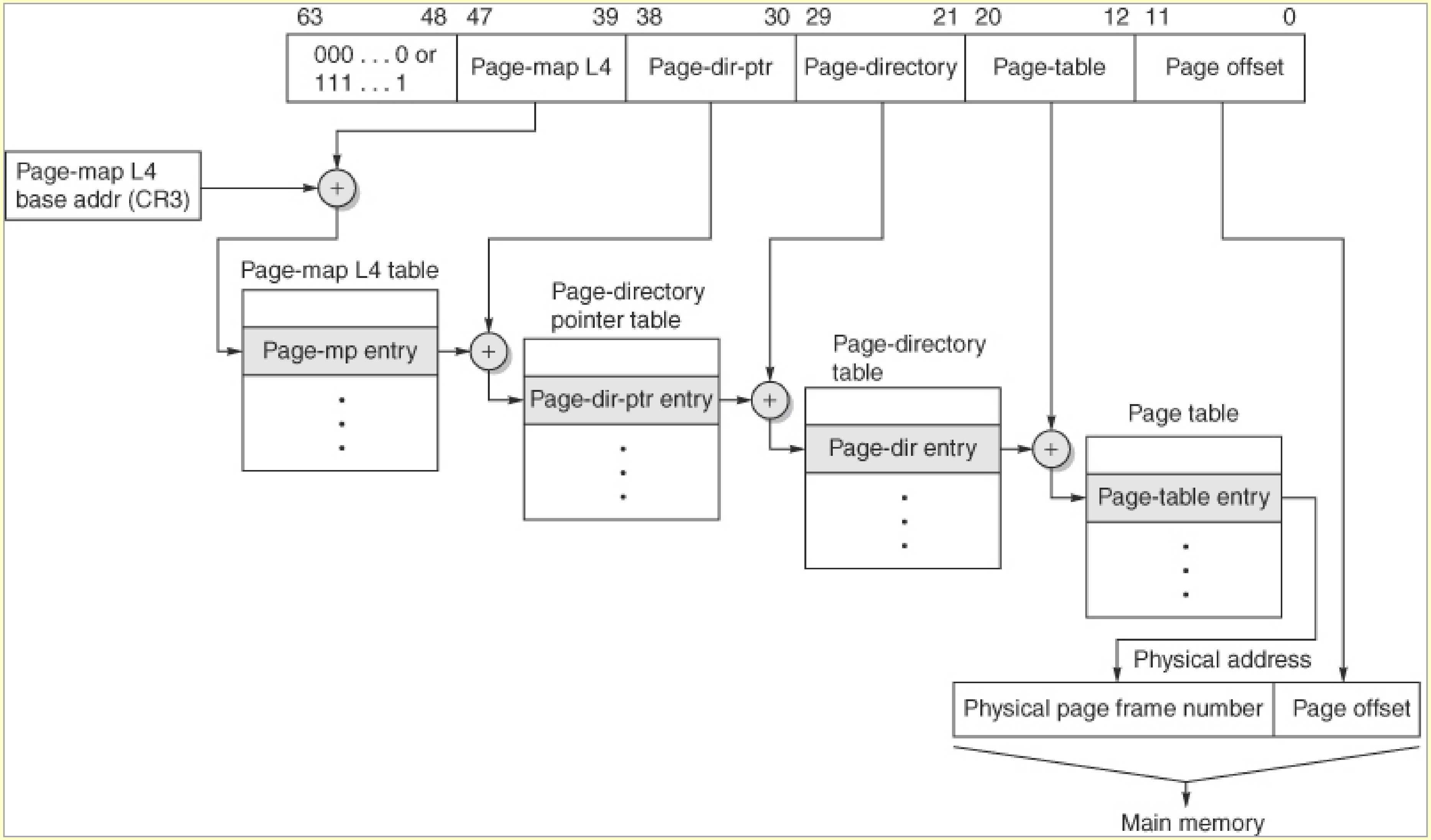
Virtual Memory

- **Virtual memory is where the “action” is**
  - All programs run in virtual memory
  - All programs “speak” virtual addresses
    - Even the kernel!
- **Physical memory is a *cache* of virtual memory**
  - Page Fault <—> Cache miss
  - Page replacement <—> Flush cache entries

  - Address translation via hierarchical *page tables*

Fig B.27

Page table structure — 64-bit Opteron



TLB — Translation Lookaside Buffer

- **TLB is a *cache* of Page Table Entries**
  - Indexed by low-order bits of page address
  - Tags = high-order bits

Optimize for Common Case

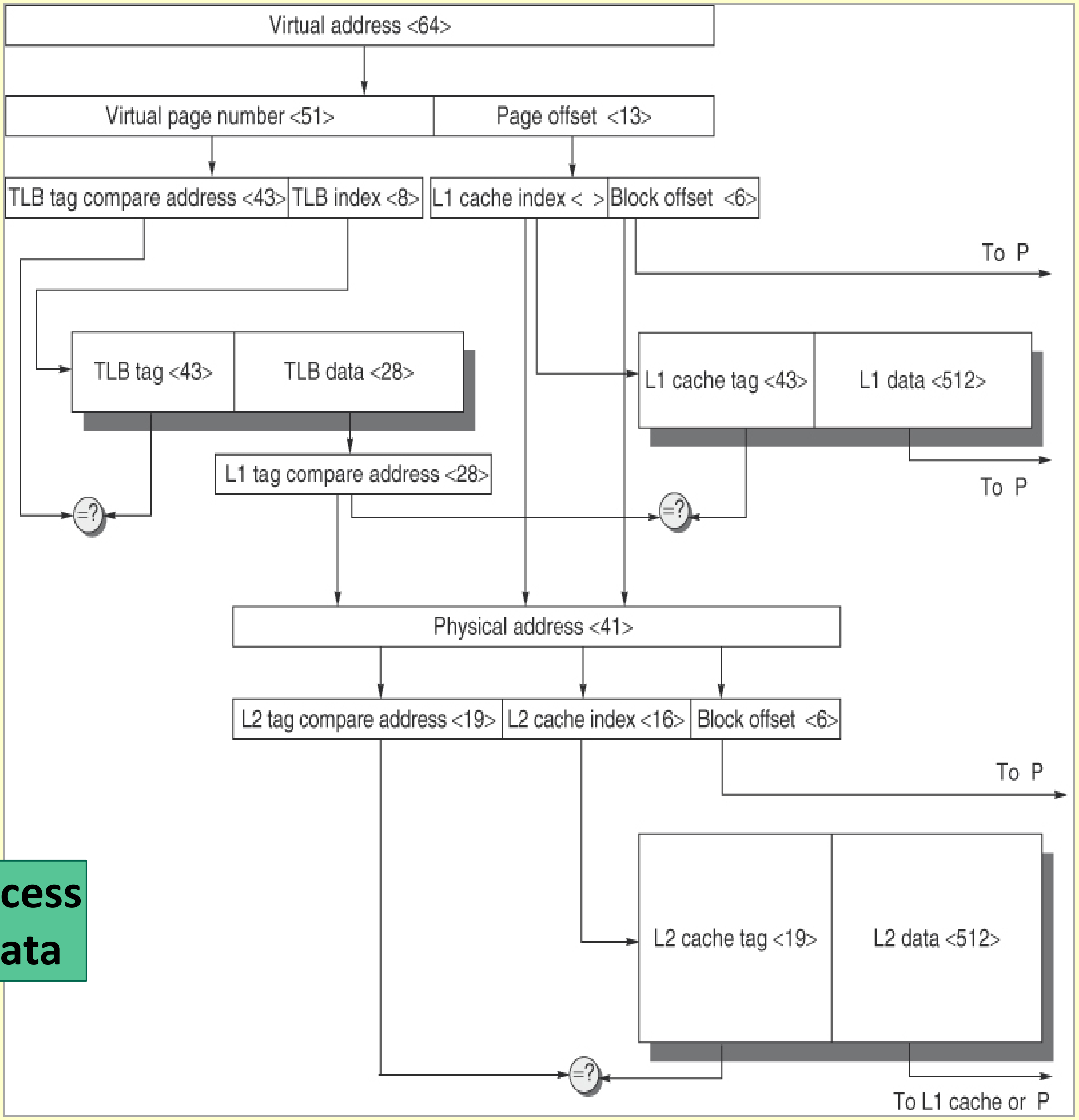
- **Read from/write to simple memory location**
  - Already “paged in”
  - Recently accessed
  - Avoid overhead of page table lookup on every access

- **Solution:–**
  - Virtually indexed, physically tagged L1 cache

Hypothetical architecture

Check L1 cache tag by *physical address*

Why?



Index L1 cache by *virtual address*

Result: 1-cycle access to most active data

Fig. B.25

Questions?

CS-4515
Computer Architecture

Professor Hugh C. Lauer
Professor Thérèse Smith

CS-4515, Computer Architecture