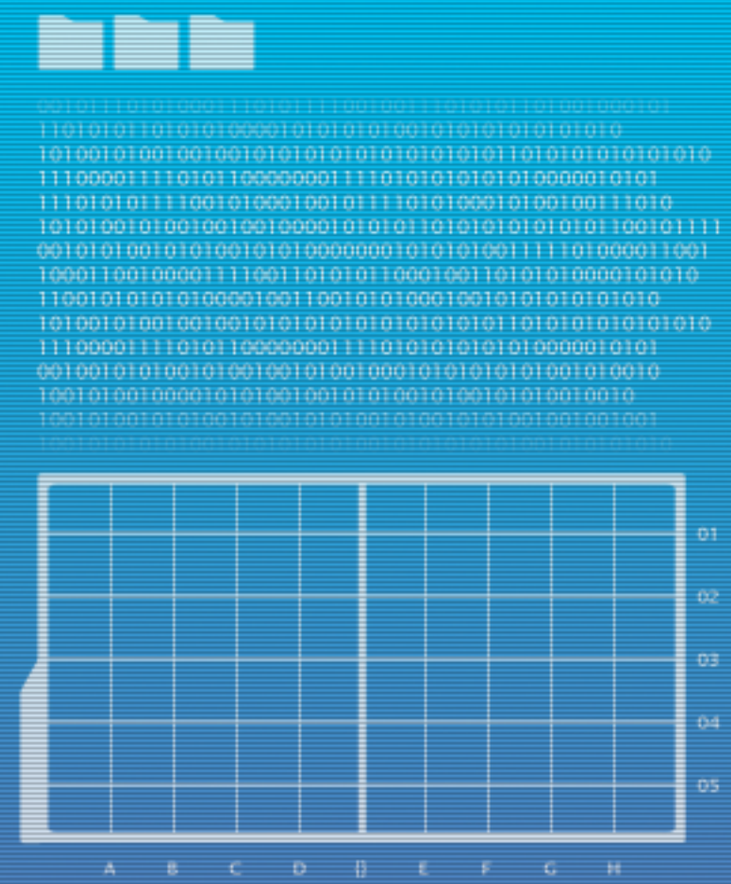
DEPARTMENT OF
INFORMATION SYSTEMS
AND COMPUTER SCIENCE



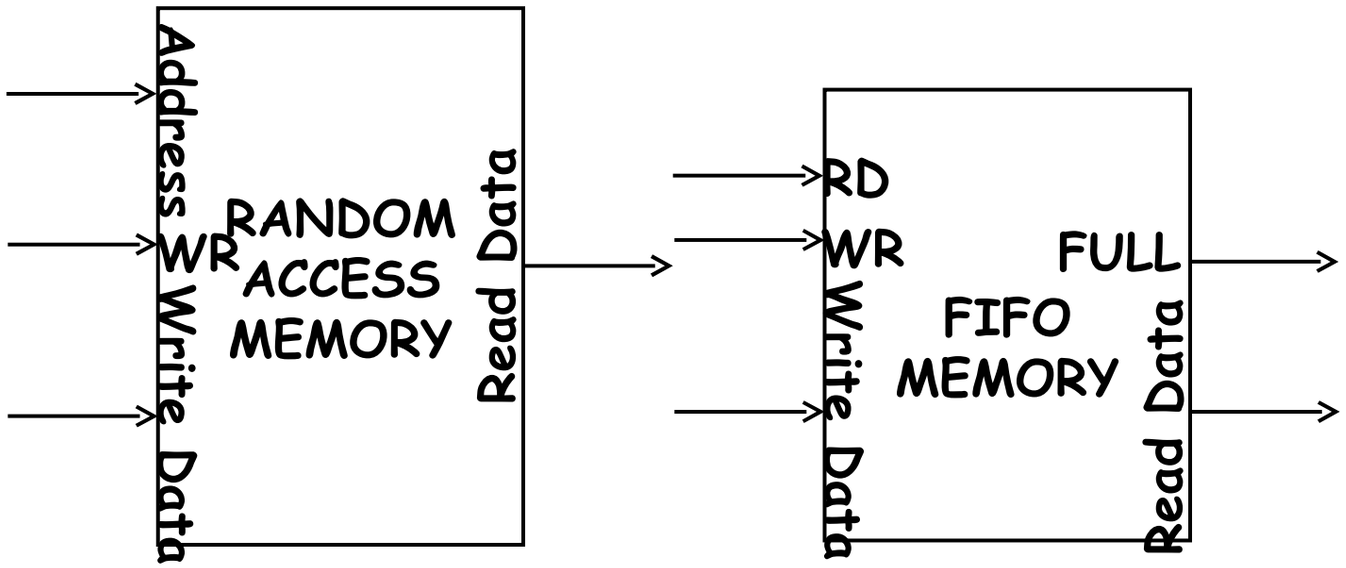
Caching

Pronounced KASHing, not ka-ching.

Semiconductor Memories

- **The majority of transistors found in a modern system are devoted to memory.**
- Memories are used for data storage.
- The need for increased memory densities and lower memory prices has been the driving force in the development of VLSI technology.
- Random-access memories (RAMs) are the most common form.

Just what exactly do you mean by "random-access"?



The term "RAM" is a catch-all phrase for a memory whose contents can both be read and written.

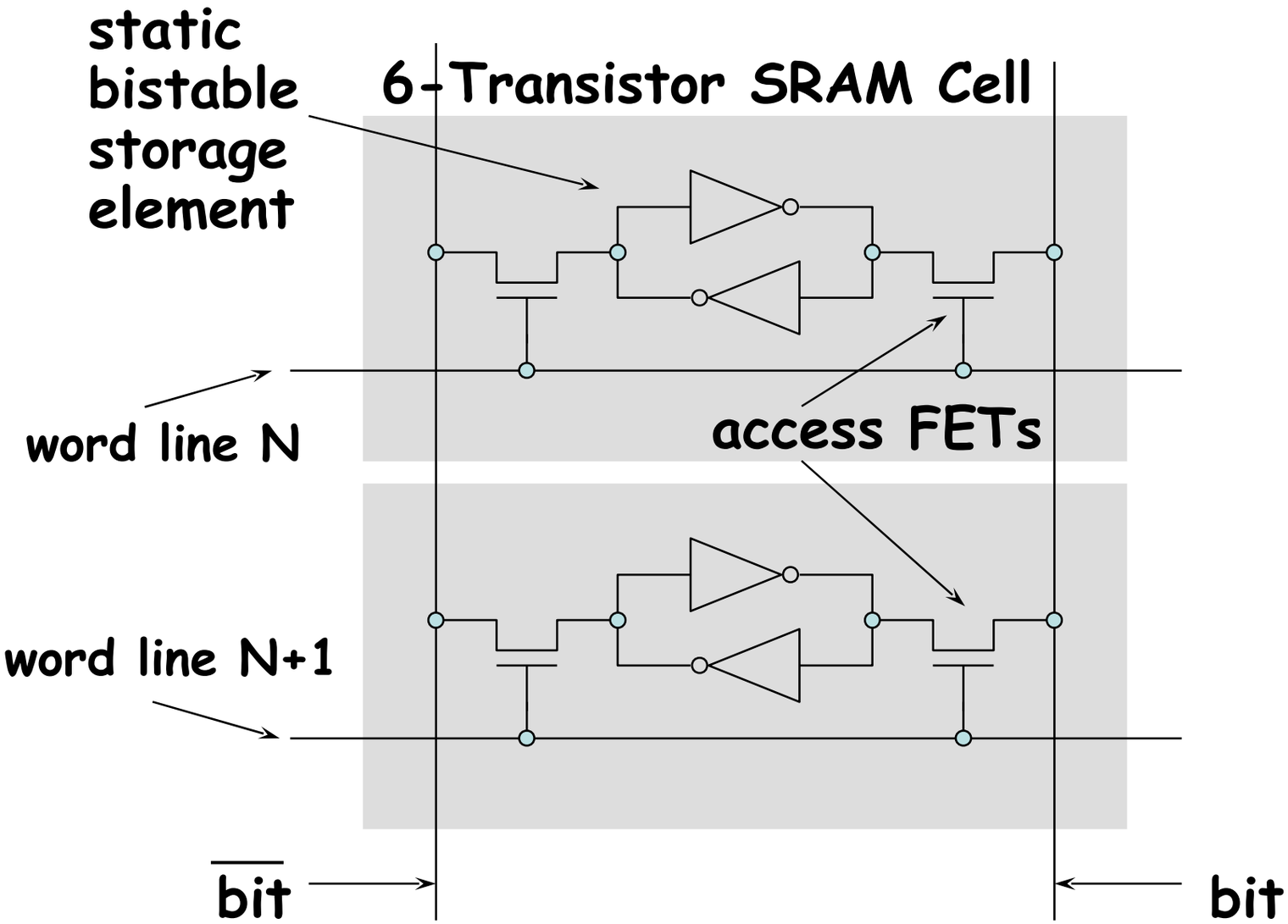
Memories by any other name

- Uses of memory:
  - "main" memory ⇒ high capacity, low cost
  - cache memories, TLB's ⇒ fast access
  - programming info (e.g. BIOS, FPGA initialization) ⇒ non-volatile
- Read-only memories: ROM (non-volatile)
  - Mask programmed / Programmable ROM (PROM)
  - Erasable PROM (EPROM) / Electrically Erasable PROM (EEPROM)
  - Block-programmable (Flash) PROM
- Read/Write or Random Access memories: RAM
  - Static RAM (SRAM) - Fast, Persistent, and Expensive
    - Multiport SRAM (Register Files, FIFOs)
    - Synchronous SRAM (Caches)
  - Dynamic RAM (DRAM) - Slow, Forgetful, and Cheap
  - Serial-access video memories (VRAM)
  - Synchronous DRAM (SDRAM, DDR SDRAM)
  - Rambus DRAM (RDRAM)

Wall of Text, Explained

- TLB – Translation Lookaside Buffer
- BIOS – Basic I/O System
- FPGA – Field-Programmable Gate Array
- DDR – Double Data Rate (usually has a number afterwards, like DDR2, to indicate how many times the word read count was doubled --- DDR3 reads twice as many as DDR2)

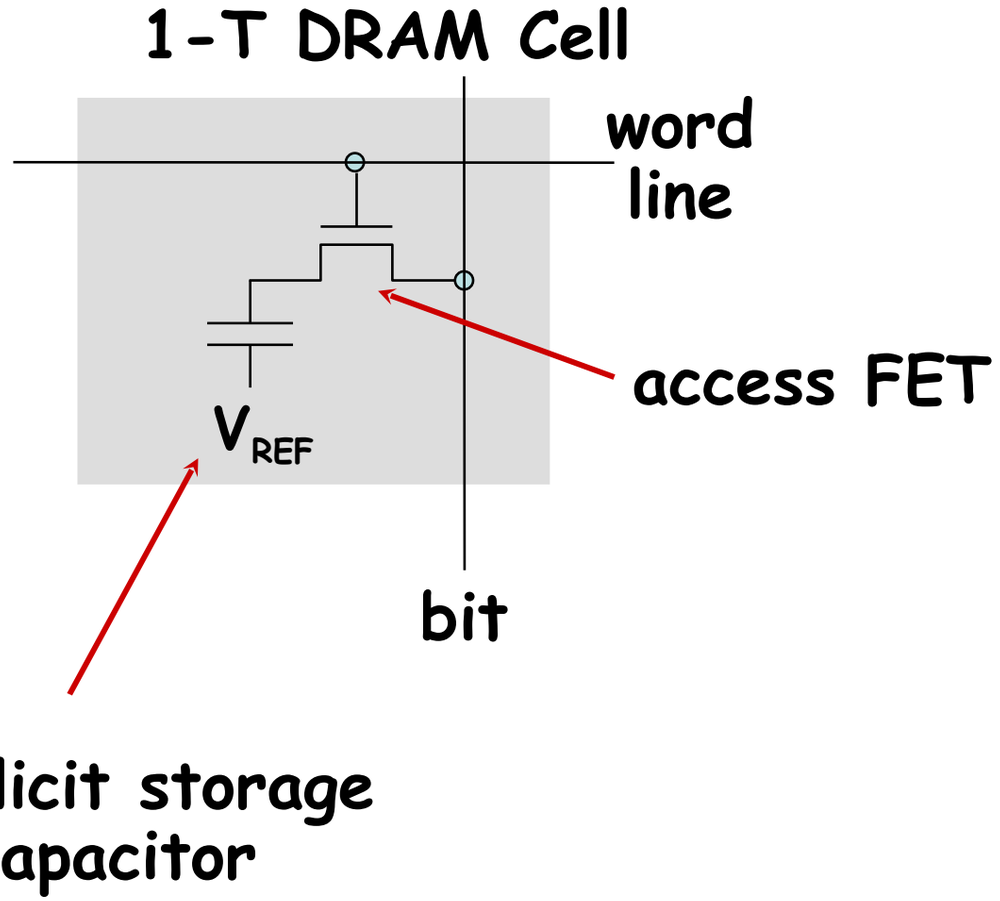
SRAM Memory Cell



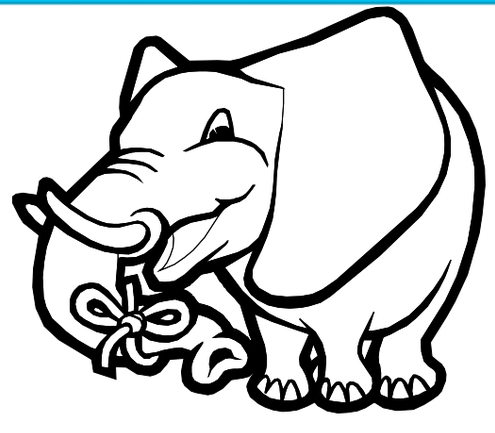
- Based on inverter loop, similar idea to RS latch.
- There are TWO "bit-lines" per bit of data: one for the bit, the other for its complement
- To Read:
  - A single word line is activated (driven to "1"), and the access transistors enable the selected cells and their complements onto the bit lines.
- To Write:
  - Similar to read, except the bit-lines are driven with the desired value of the cell.
  - The writing has to "overpower" the original contents of the memory cell.

When DENSITY is the name of the game

- Six transistors/cell may not sound like much, but they can add up quickly.
- Fewest number of transistors that can be used to store a bit?
- One transistor and one capacitor:
  - Capacitor stores data in the form of charge (1) or no-charge (0).
  - Transistor lets it be read or written.



The Problem with Capacitors…

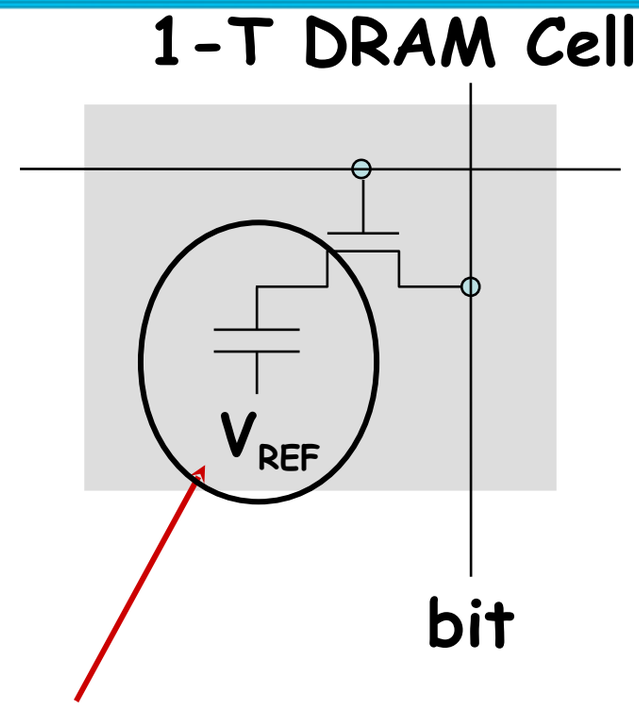


… is that they **forget!**

- Charge is lost when you read, so you need to *rewrite* it while reading.
- Worse, even when you don't read, charge *leaks* out.
- So periodically, every memory location must be "refreshed":
  - Memory controller circuit cycles through different addresses to read and rewrite their contents.
  - Refreshing can slow things down because the CPU can't use the memory during a refresh cycle.

Another Problem

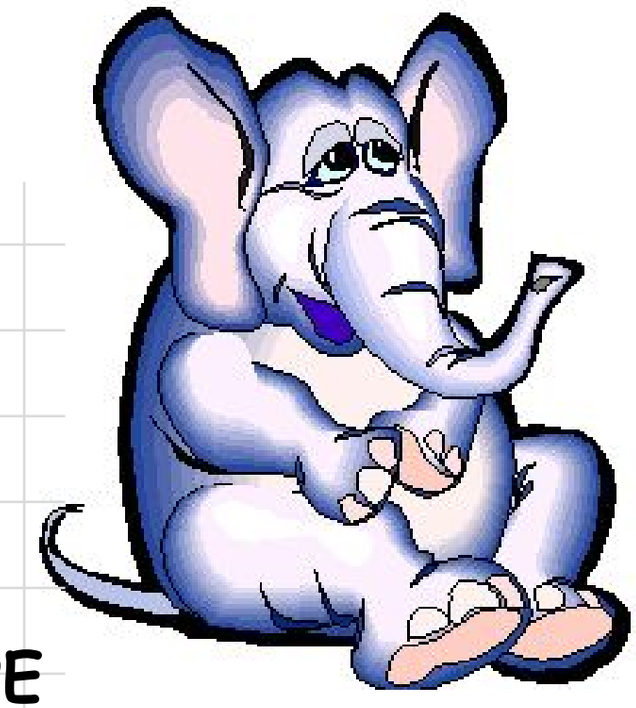
- Capacitors are SLOW.
- We want big C so it doesn't leak out too fast.
- But big C means big RC constant, which means long delays.
- Bottom Line:
  - DRAMs can be BIG.
    - Lots of bits in the same area, and a single DRAM is TINY (think microns).
  - But they are SLOW!

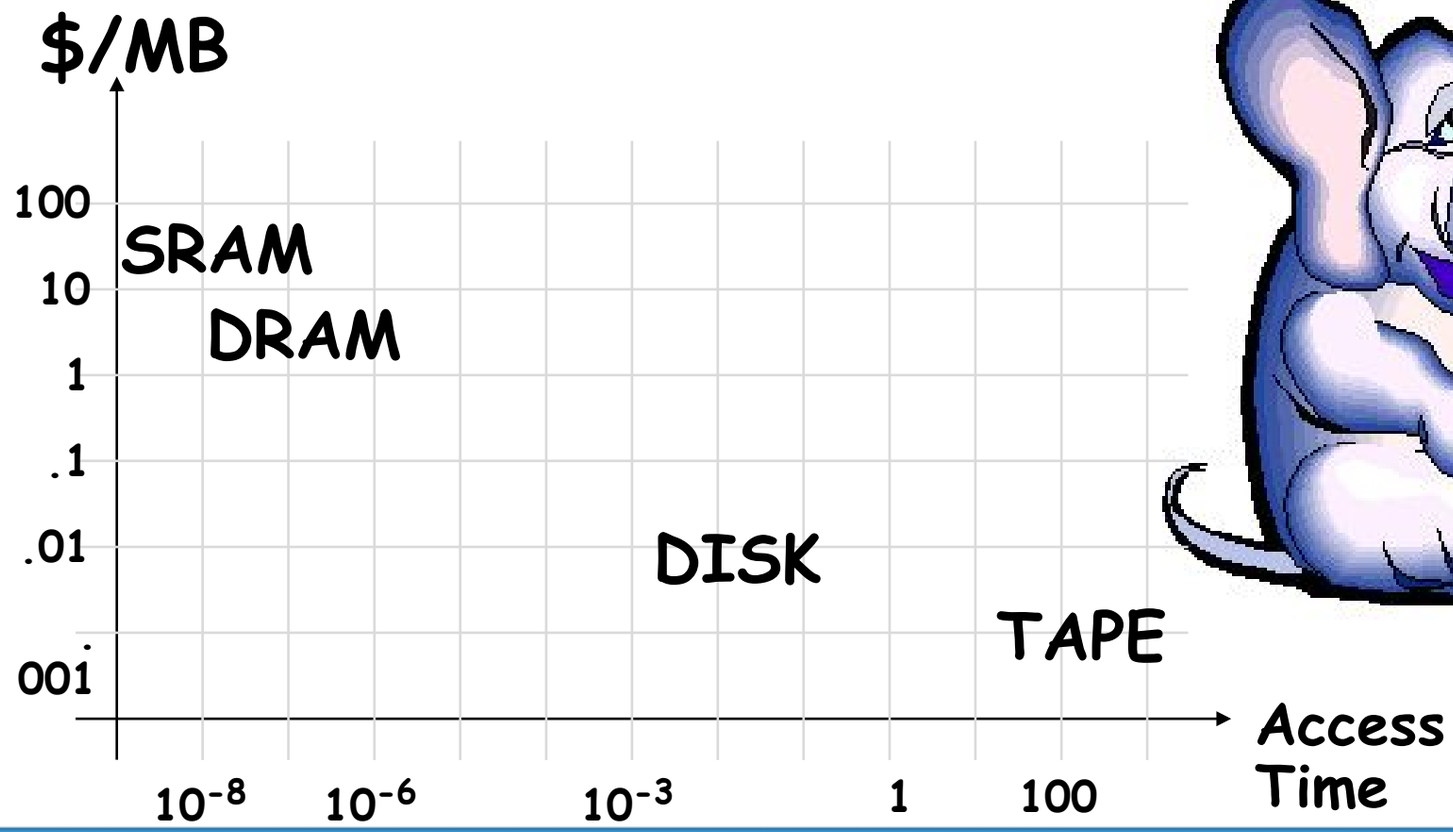


Quantity vs Quality…

- The trade-off:
  - BIG and SLOW
  - or, SMALL and FAST
  - but not BIG and FAST

Is there a way out?



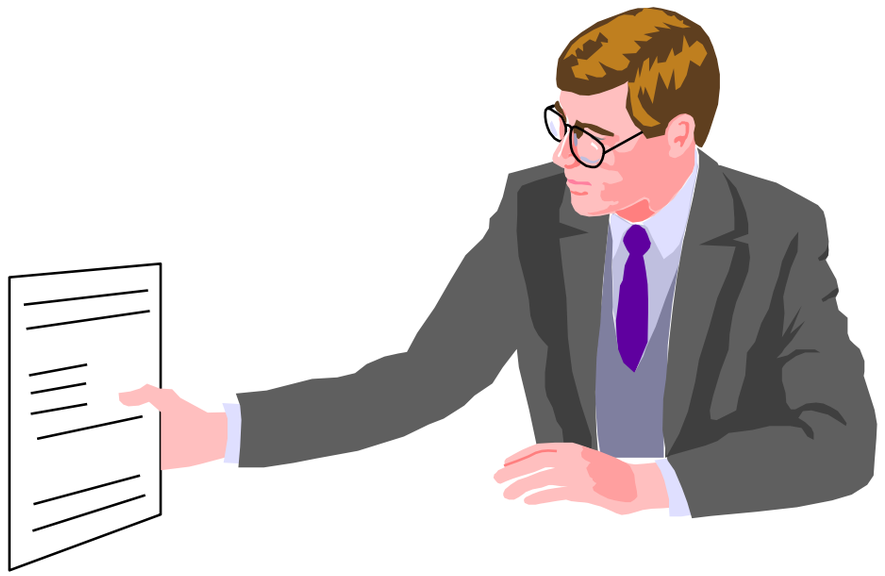


A day in the office: Which is faster?

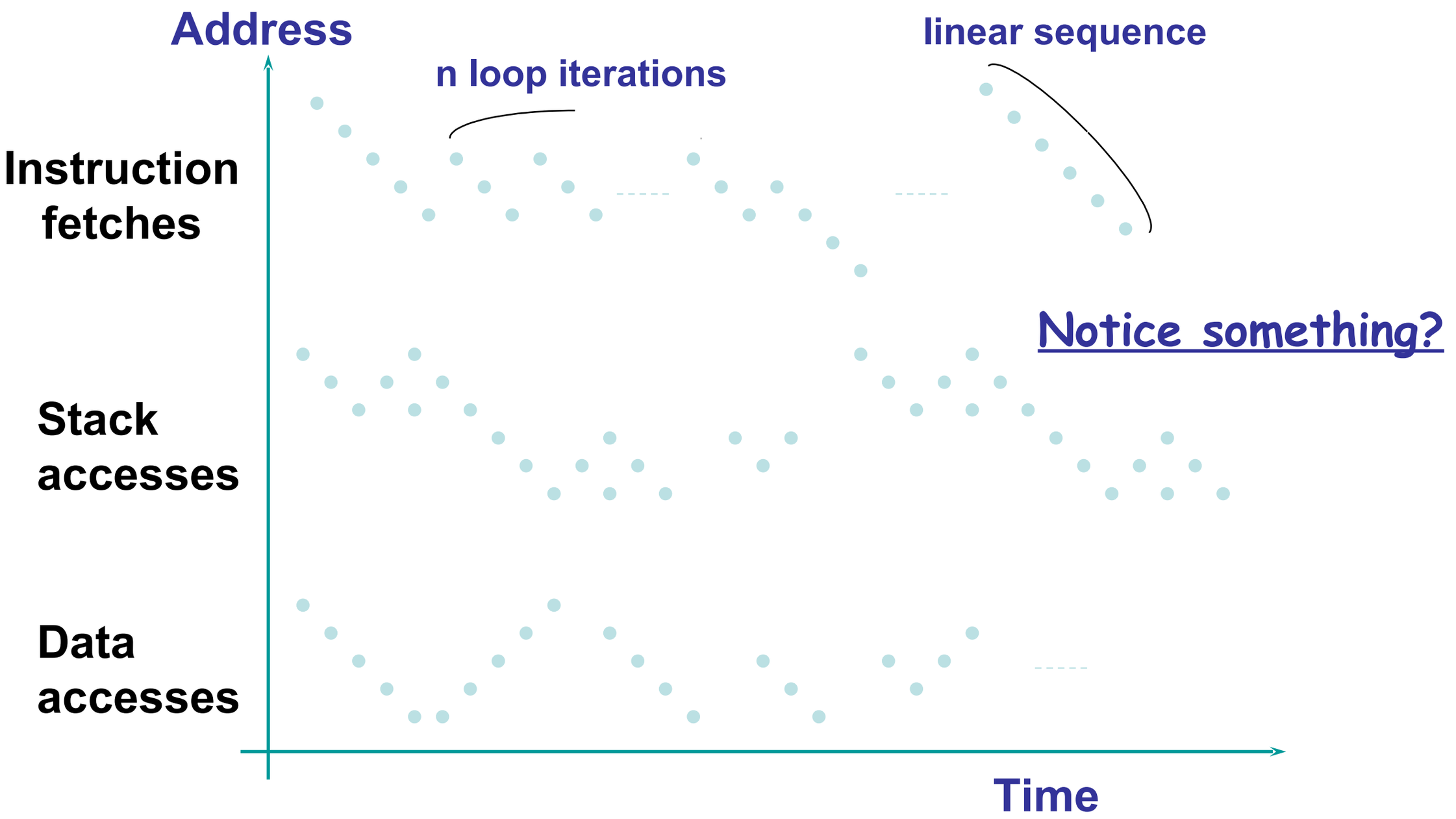
Find "Bitdiddle, Ben"

5-Minute Access Time:

5-Second Access Time:



Typical Memory Reference Patterns



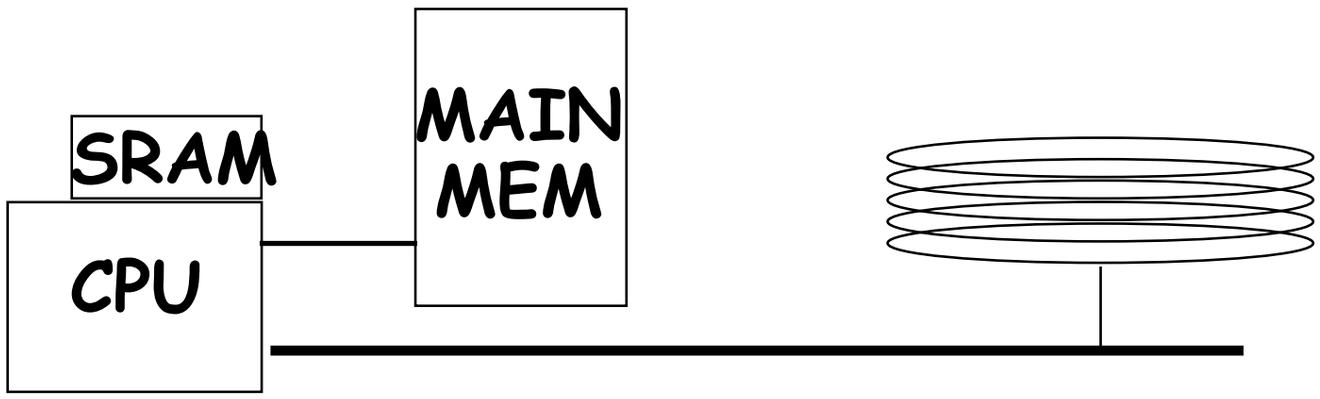
Notice something?

Locality

- Key observations:
  - If a location is referenced it is likely to be referenced again in the near future.
    This is called temporal locality.
  - If a location is referenced it is likely that locations near it will be referenced in the near future.
    This is called spatial locality.

Memory Hierarchy

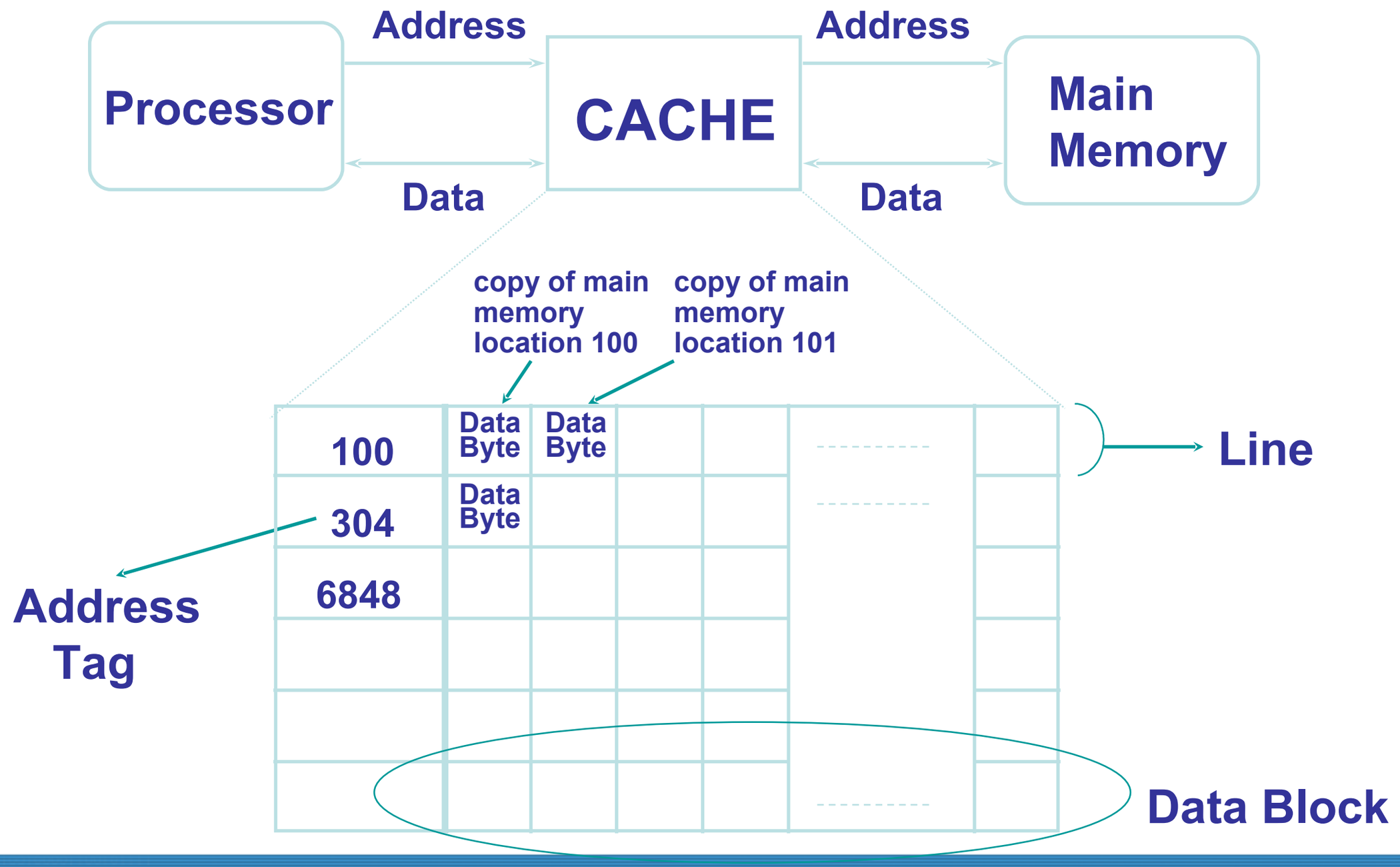
- Idea: Use small but fast memory for "more needed" data, and store the unneeded data in larger slower memory.
  - Data you need now or very soon should be in registers or in SRAM cache.
  - Data you will use soon should be in main memory.
  - Data you don't need yet can be on disk (virtual memory).
- So, if we use the data in the SRAM a lot, then we don't feel the delay of main memory and disk.
  - As a whole, it will look like a BIG and FAST memory!



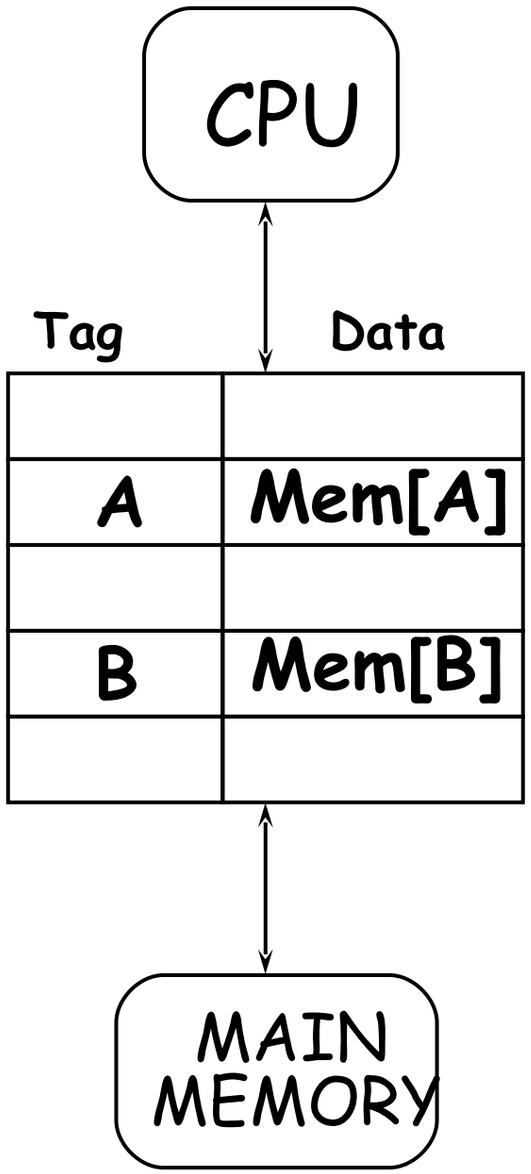
Caches

- Caches exploit both properties of memory reference patterns:
  - Exploit temporal locality by remembering the contents of recently accessed locations.
  - Exploit spatial locality by remembering blocks of contents of recently accessed locations.

Inside a Cache



Basic Cache Algorithm



- ON REFERENCE TO Mem[X], look for X among cache tags:
- If *X = TAG(i) , for some cache line i:* ***HIT!***
  - Read the data from the Cache and return to CPU.
  - No need to go to memory!
- If *X not found in TAG of any cache line:* ***MISS!***
  - Read Mem[X].
  - Select line k in cache to put data.
  - Write Mem[X] into line k of cache.
  - Set Tag(k) to X.
  - Return data to CPU.

Average Access Time

- α is **HIT RATIO**: Fraction of references that hit in cache
- $1 - \alpha$ is **MISS RATIO**: Fraction of references that miss
- Average access time for a single-cache system:

Prob. of hit — cost of hit — Prob. of miss — cost of miss

$$\alpha\, t_c + (1 - \alpha)(t_m + t_c)$$

$$= t_c + (1 - \alpha)\, t_m$$

Goal: make MISS RATIO very SMALL!

Example

$$\text{Ave. Mem. Access Time} = t_c + (1 - \alpha)\, t_m$$

- What's AMAT without Cache? $AMAT = t_m = 40$ ns
- What's AMAT if α is 75%?

$$AMAT = 4 + (1 - .75)\, 40 = 14 \text{ ns}$$

- How high must α be to get an AMAT of 5 ns?

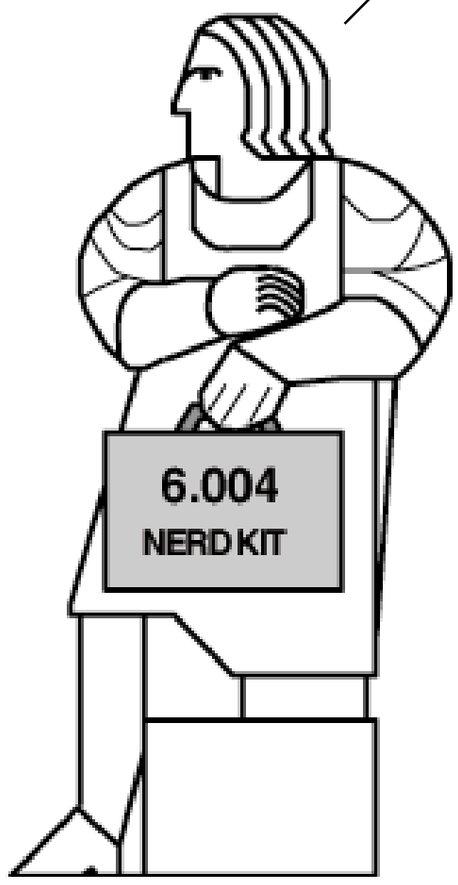
$$5 \text{ ns} = 4 + (1 - \alpha)\, 40 \quad \text{(solve for } \alpha\text{)}$$
$$\alpha = 1 - (5-4) / 40 = 97.5\%$$

Note: a must be really high

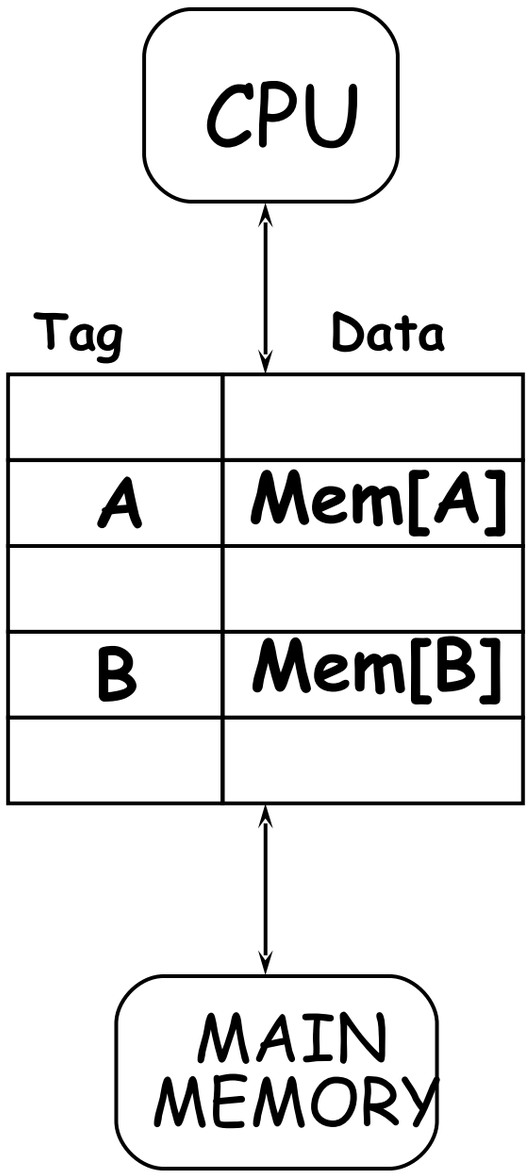
An MIT joke

Why study?
The quiz is open book.

To reduce average access time.



Recap: Basic Cache Algorithm



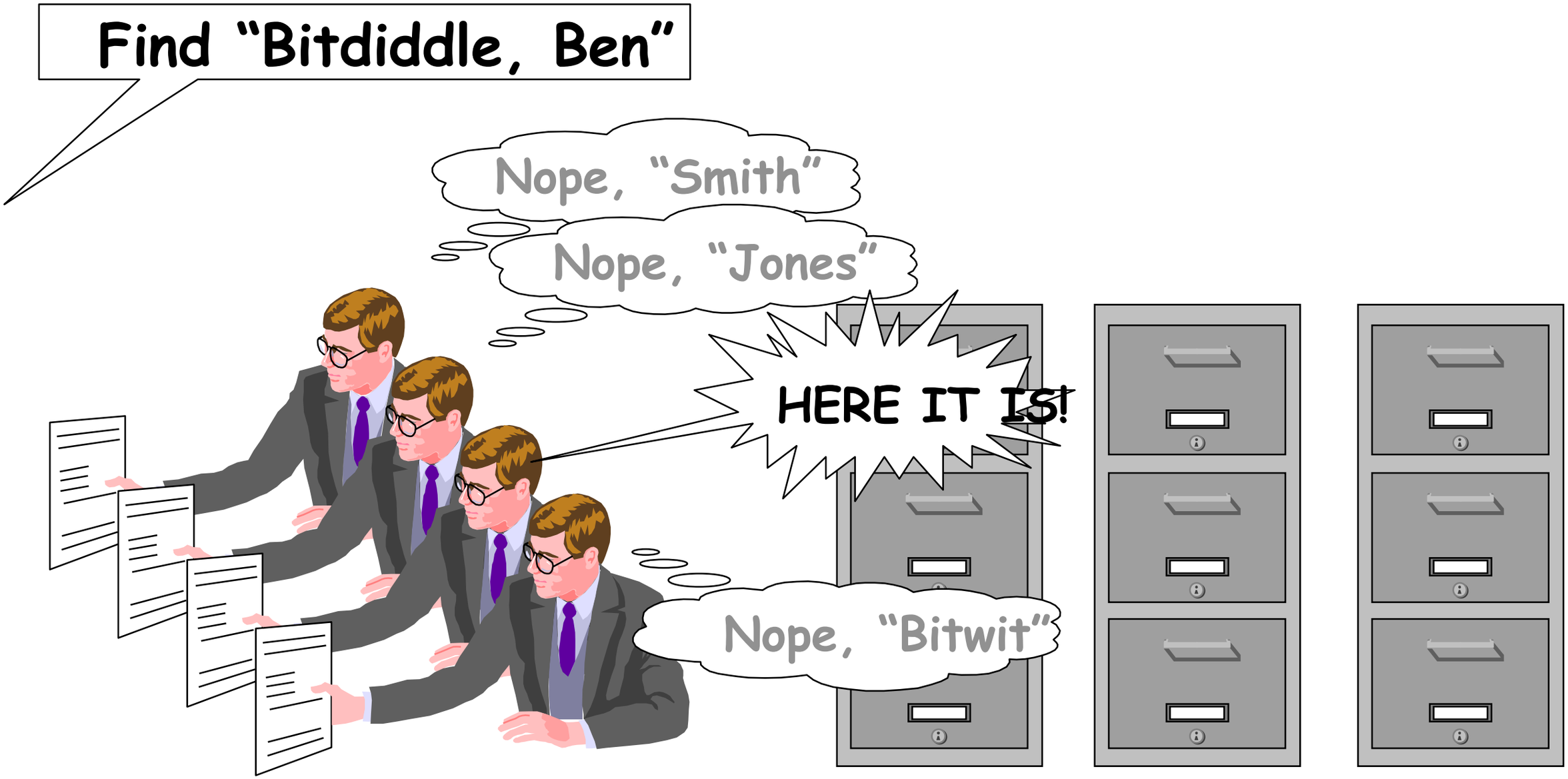
- ON REFERENCE TO Mem[X], look for X among cache tags:

HOW?

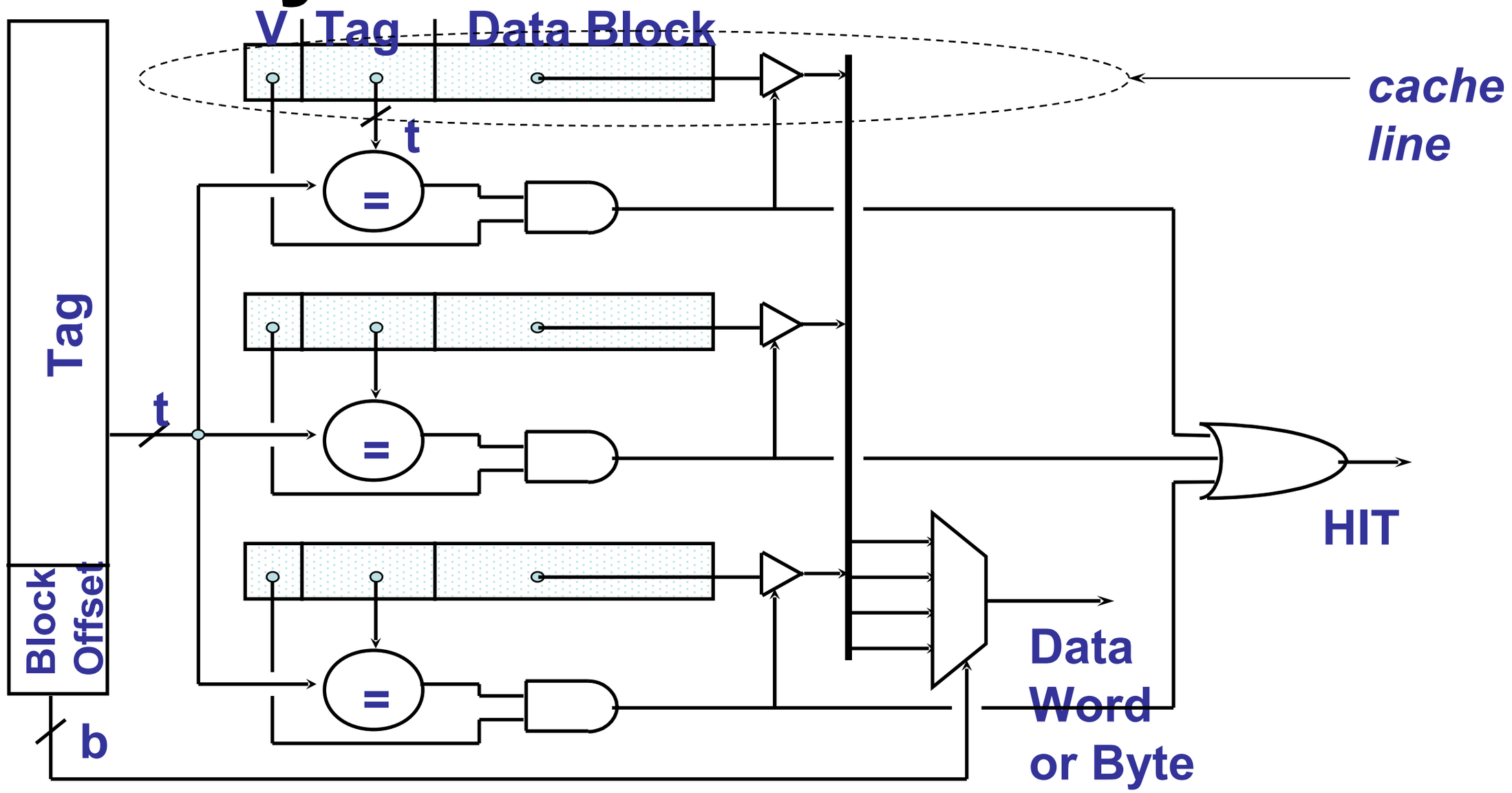
Searching Tags

- There are different ways of searching the tags for a match, depending on the cache:
  - Fully associative cache: Address can be anywhere in cache.
    - Search all tags in parallel.
  - Direct-mapped cache: Address can be in only one place in the cache.
    - Check the tag of that place only.
  - Set-Associative cache: Address can be in a few (usually 2 to 8) places in the cache.
    - Check the tags of the set of places.
- A set is a collection of cache lines in which a given memory block may be placed.
  - 4-way set associative means 4 lines per set.

Fully Associative Cache: Parallel Lookup

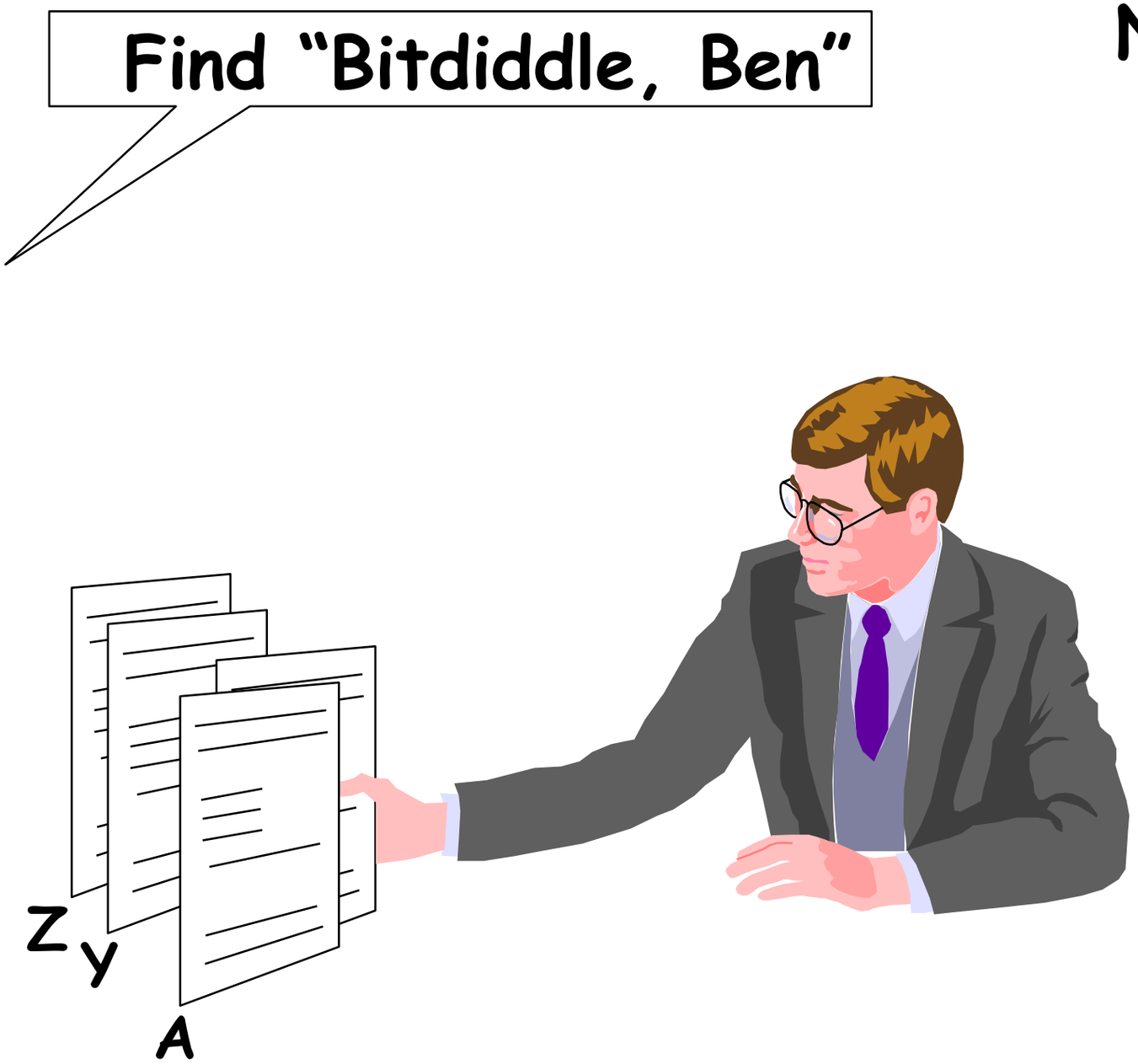


Fully Associative Cache



- Compare Tags in parallel.
- 1 Comparator for each line of cache.
- Very expensive and somewhat slower because there is more hardware (remember fan-in?)

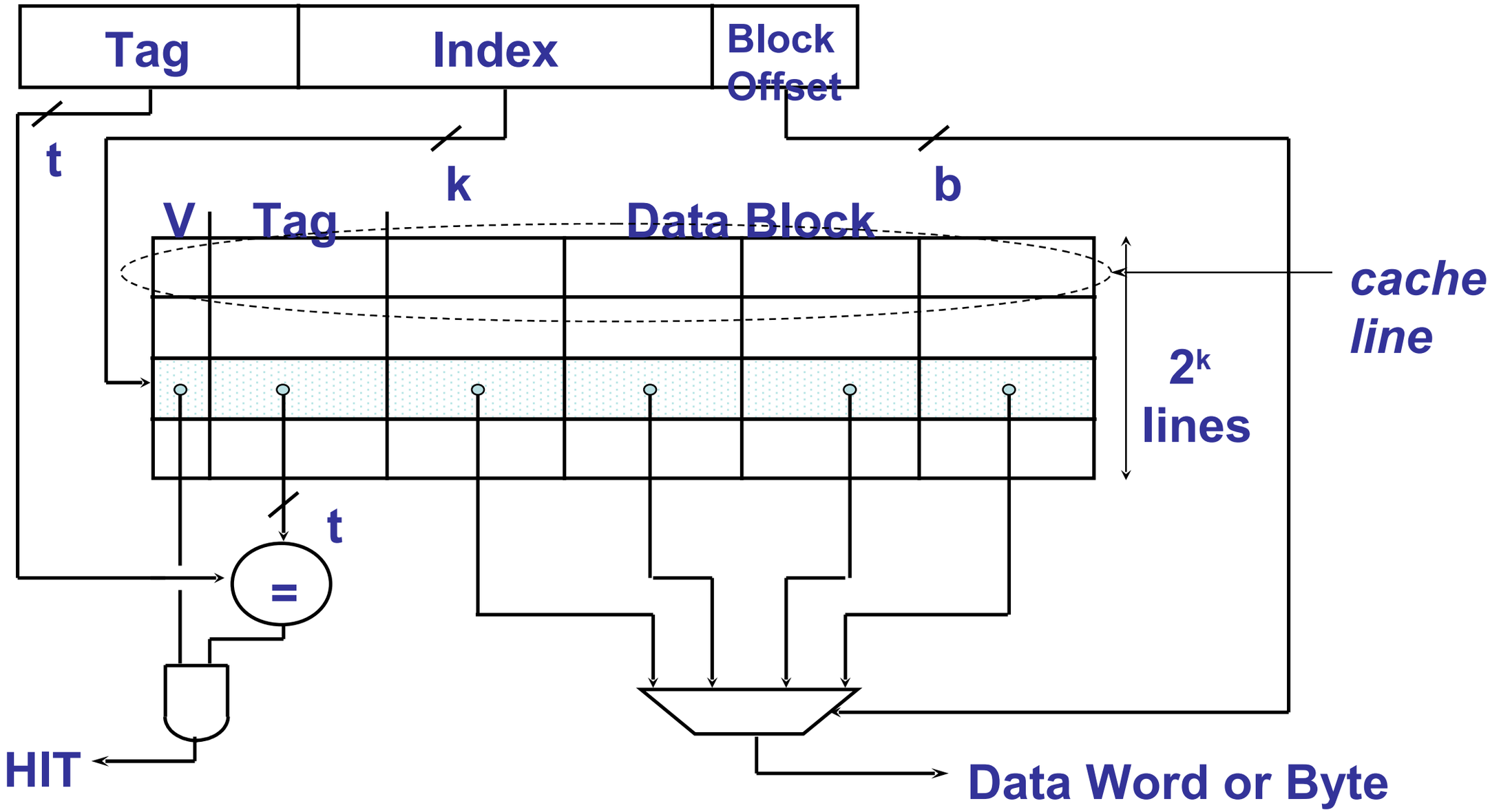
Direct-Mapped Cache



NO Parallel Lookup:

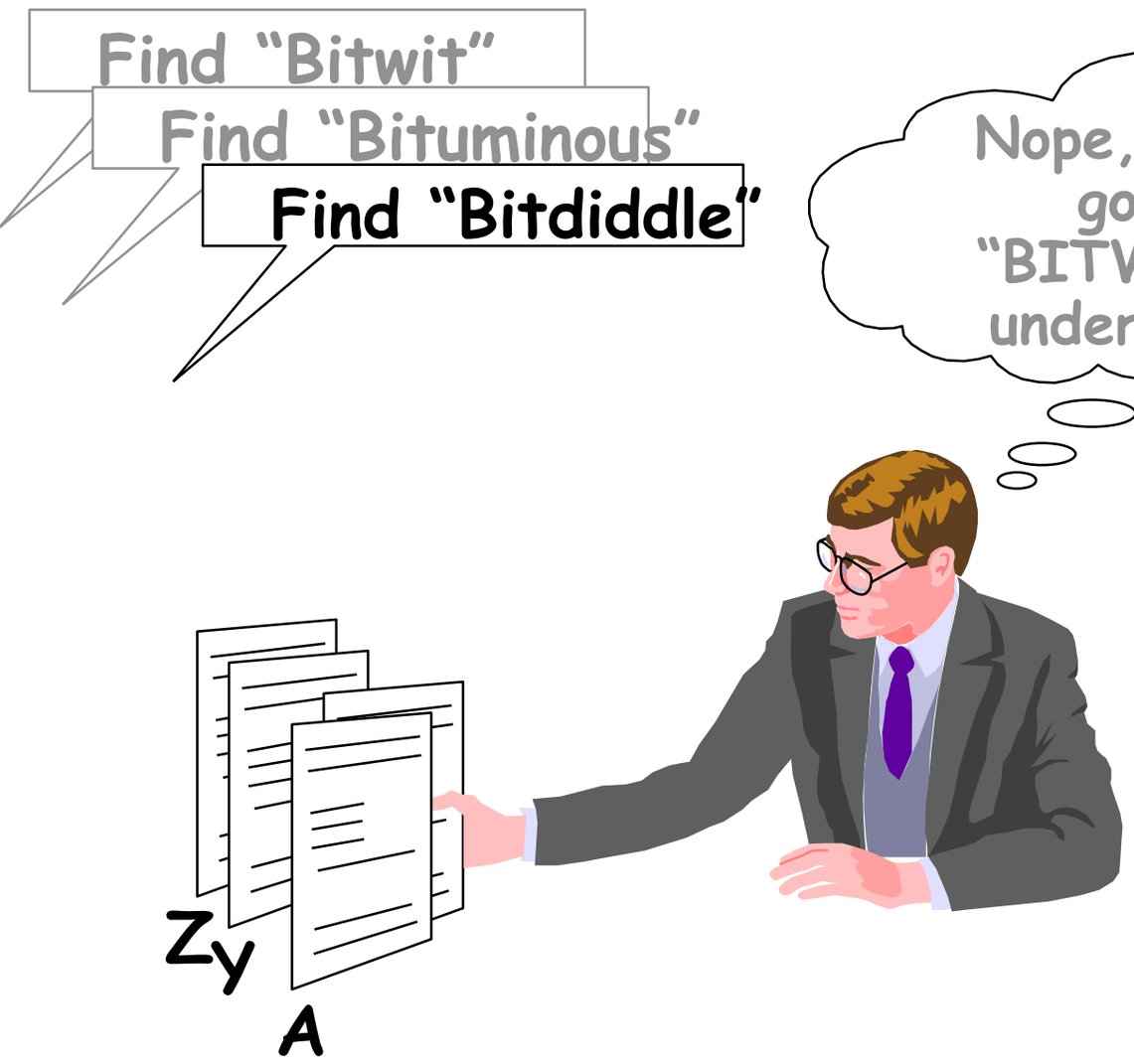
Look in JUST ONE place, determined by parameters of incoming request (address bits).

Direct-Mapped Cache



- Index selects a line first, then check tag.
- Cheaper: only 1 comparator for entire cache.
- For the same silicon space, more data bytes.

The Problem with Collisions



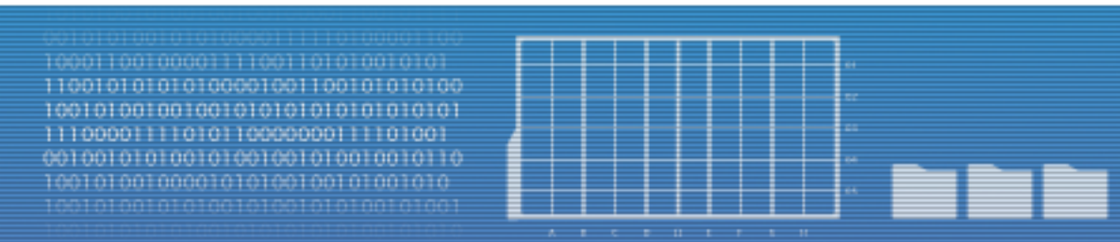
PROBLEM:

Contention among B's....

- **CAN'T cache both "Bitdiddle" & "Bitwit" at the same time because they use the same line.**

Problem with Collisions

- A partial solution: Use the *lower* bits of address as index.
- Spatial locality says the upper bits are likely to remain the same since we access data near each other.
- In our example, names starting in B would tend to come together.
- Solution: Cache it by *last* letter of name.
- This *reduces* collisions.
- But this doesn't solve everything.

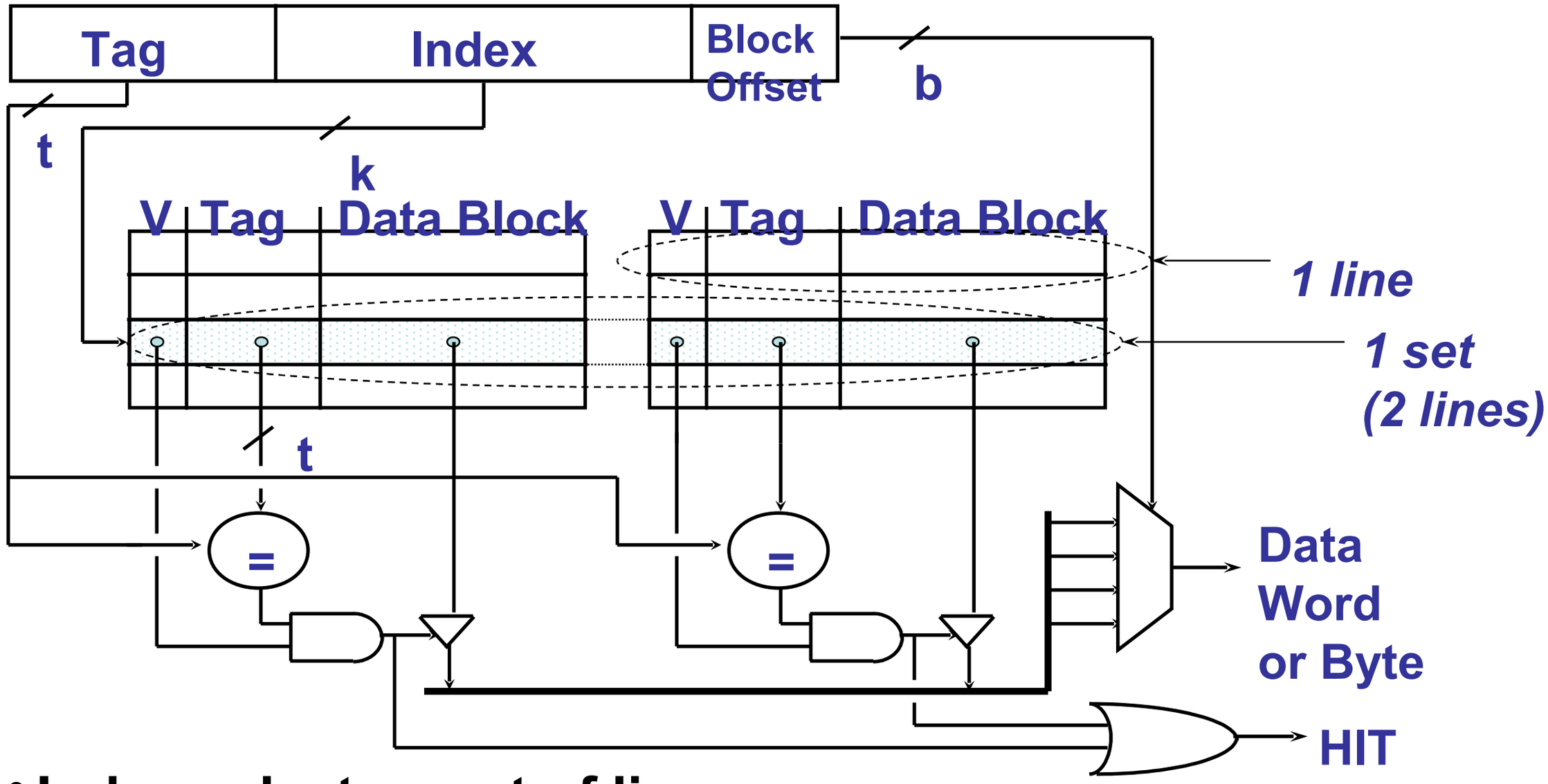


DISCS

Set Associative Caches

- Idea: Have 2 or more lines per index.
- Lines with same index form a <u>set</u>.
- Given index, we find the set and search tags in parallel within set.
- Like a phone book with a page for “A”, “B”, etc.
  - Direct mapped would be a phone book which only allows one name per letter.
  - N-way set associative would allow N names per letter.

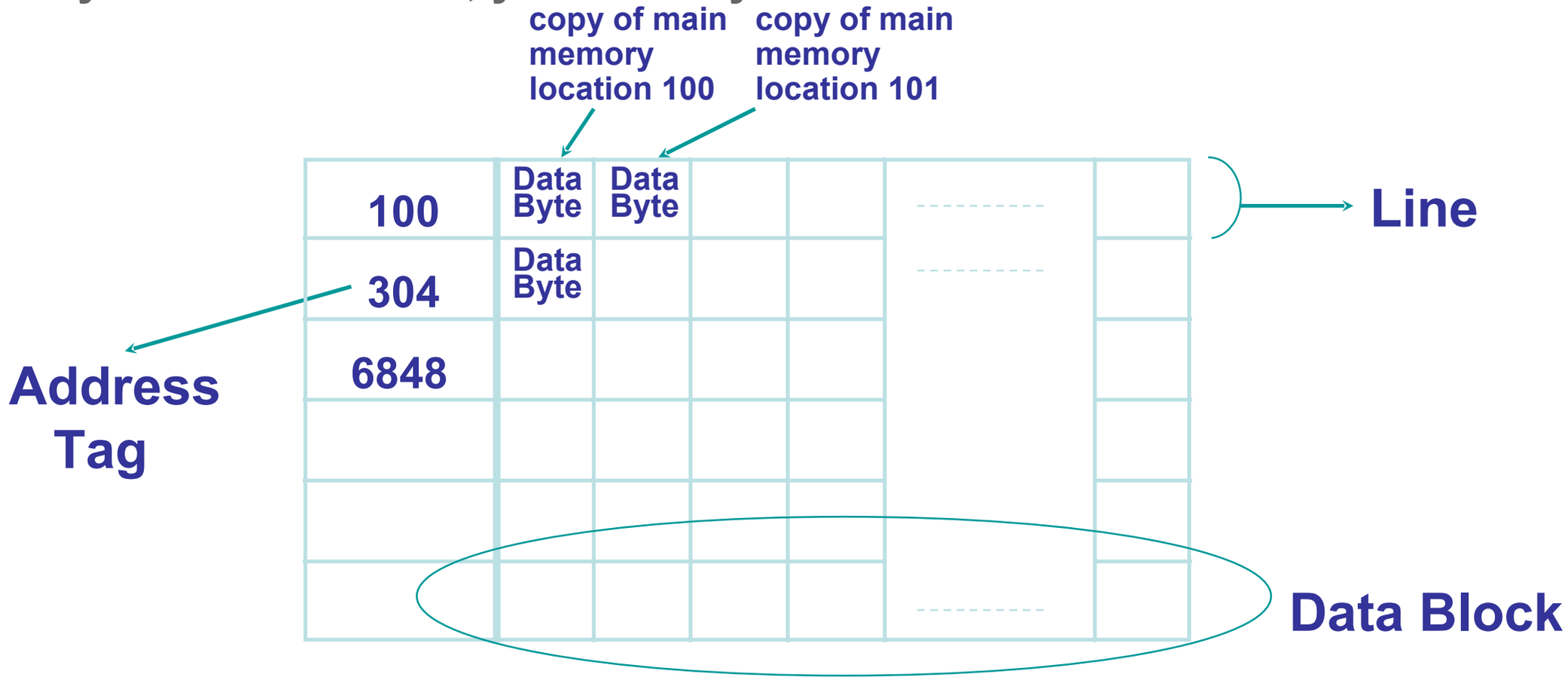
2-Way Set-Associative Cache



- Index selects a <u>set</u> of lines.
- N comparators (for N-way set associative).
- More expensive/slower than direct-mapped, but better hit rates because less conflicts.

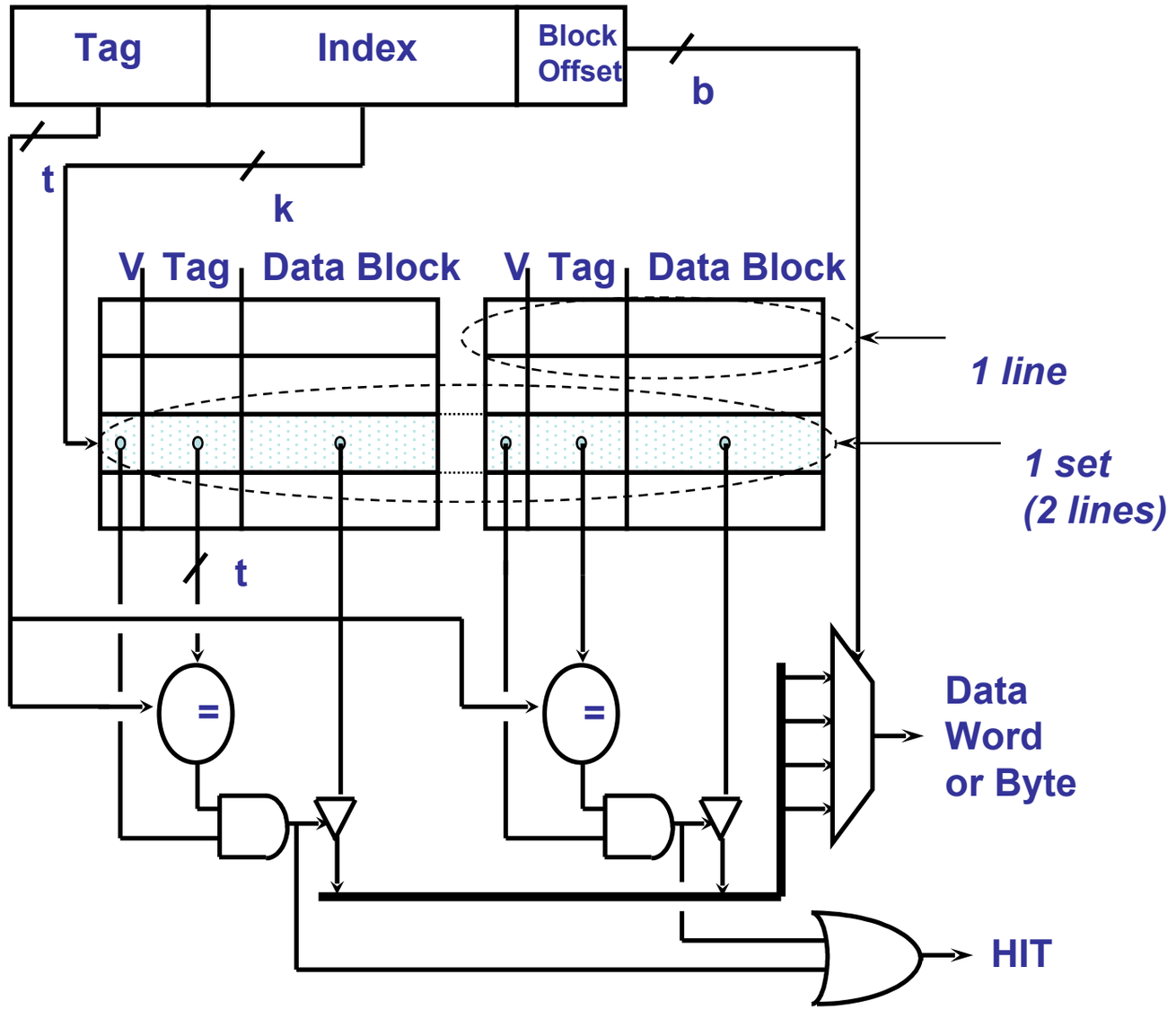
Cache Blocks

- Note: Each “line” in cache contains a “block” of data.
  - The entry for address 100 contains the data for 100, 101, 102, 103, 104, 105, etc.
- Why? Spatial Locality.
  - If you load from 100, you’re likely to load from 104 in the near future.



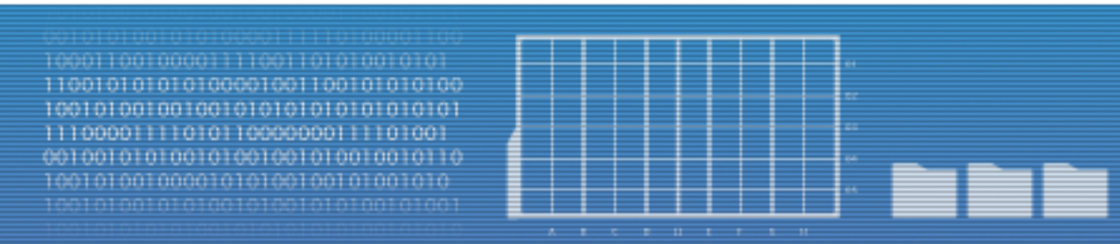
Blocks, Tags, and Indices

- In reality, tag and index are taken from the "block number".
  - If block size = 8 words (32 bytes), then:
    - address 0 to 31 is block #0, 32 to 63 is block #1, etc.
  - The block offset determines *where* in block a byte is.
    - address 48 would be block #1, offset 16, or byte 16 (17th byte) of block #1.
- Index is $\log_2$ [# of sets] bits long (how much info do you need to know what set to look at?)
- Tag is the rest of the address.
  - Technically, tag can be the whole block number (and therefore include the index).
  - No need to do this (except for a fully-associative cache) since all addresses in the same set will have same index. It would be a waste of bits to repeat the index in the tag.



Cache Terms I (parts of cache)

- Data Word: Unit of data in processor's ISA (Instruction-Set Architecture).
  - In the Beta (simulated CPU that you will be using in BSim and making in JSim): 32 bits = 4 bytes.
- Data Block: Groups of adjacent data words read into cache at the same time.
  - Typically a small power-of-2, e.g., 16 to 128 words.
    - We will assume 1-byte words for now so that we can give the block size in bytes, which will allow some of our calculations to actually make sense.
- Cache Line: Place in cache where a data block is stored.
  - Includes data block, tag, valid bit, dirty bit, etc.
- Set: A group of blocks where a particular main memory address *A* can be placed in the cache.
  - N-way set-associative: N lines/set, thus # sets == #lines/N
  - Fully-associative: 1 set for the whole cache
  - Direct-mapped: # of sets == # of lines (1 line / set)



Cache Terms II (how address is divided)

- Block offset: Chooses 1 byte within a block.
  - Last $\log_2$B bits, where B is the block size in bytes
  - In Beta, we read 4 bytes at a time, so block offset must be multiple of 4 (otherwise, the code will act weird).
- Index: Chooses which set to check.
  - Next $\log_2$(# of sets) bits of address, after block offset.
  - In direct-mapped cache, index chooses 1 line then comparators check the line's TAG for a match.
  - In N-way set-associative cache, index chooses 1 set of N lines then comparators check all N lines in set for a matching TAG.
  - For fully-associative, no index since only 1 big set.
- Tag: Checks if line contains the desired address.
  - The rest of the bits, after block offset and index.

Exercises

- Given:
  - Total memory address space (M)
  - Total cache size (C)
  - block size (B)
  - set associativity (N)
- Draw the cache:
  - How many lines per set?
  - How many sets?
  - How many bits is the block offset?
  - How many bits is the index?
  - How many bits is the tag?

in bytes

# of lines in cache

N

(C/B) / N

$log_2 B$

$log_2((C/B)/N)$

$log_2 M - log_2((C/B)/N) - log_2 B$

Remember, tag does not include index and block offset bits.

DISCS

Exercises

- Given:
  - Total memory address space (M)
  - Total cache size (C)
  - block size (B)
  - set associativity (N)

  } in bytes

- Draw the cache:
- How many lines per set? N
- How many sets? $(C/B) / N$
- How many bits is the block offset? $\log_2 B$
- How many bits is the index? $\log_2 C - \log_2 B - \log_2 N$
- How many bits is the tag? $\log_2 M - \log_2 C + \log_2 N$

DISCS

Exercises

1) M = 2^{27} bytes, C = 256KB, B = 16 bytes

a) Direct-mapped

b) 4-way set-associative

c) Fully-associative

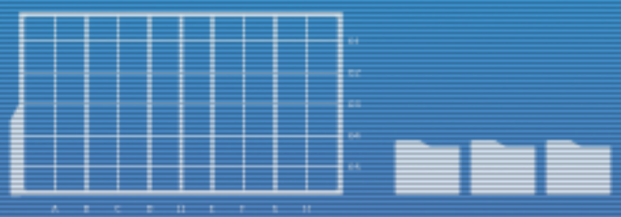
2) M = 2^{32} bytes, C = 64KB, B = 32 bytes

a) Direct-mapped

b) 2-way set-associative

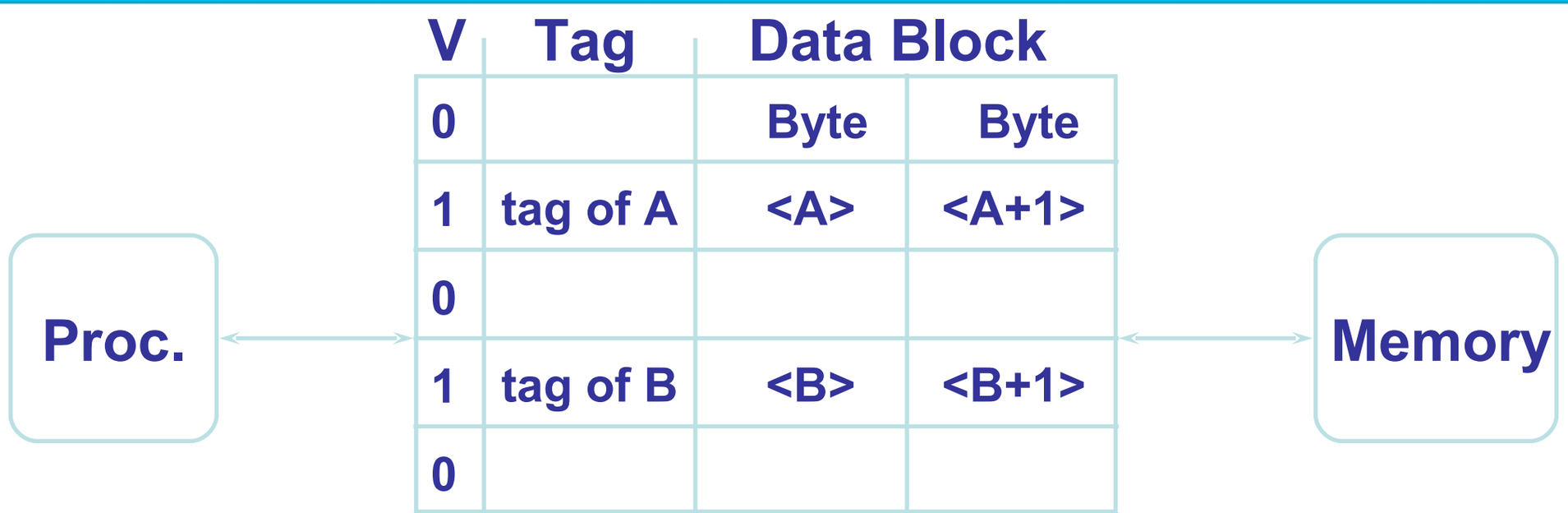
c) Fully-associative

- Draw the cache:
  - How many lines per set?
  - How many sets?
  - How many bits is the block offset?
  - How many bits is the index?
  - How many bits is the tag?



DISCS

Valid Bits



- Valid bit for each line in cache.
- Says “the data block of this line has valid data”.
- Valid bit must be 1 for cache line to HIT.
- At power-up or reset, we set all valid bits to 0.
- Set valid bit to 1 when cache line is first replaced.
- Flush cache by setting all valid bits to 0, under external program control.

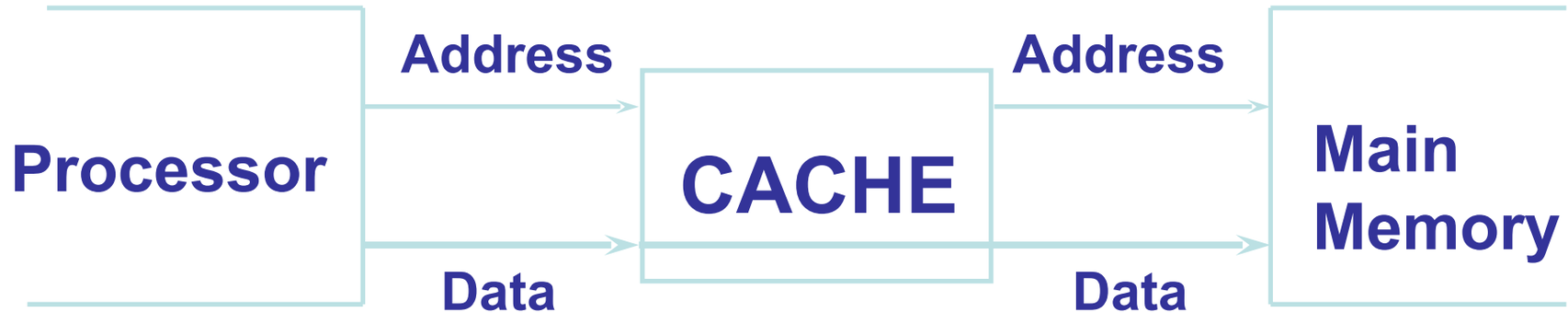
Block Replacement: Selecting the Victim

- After a read miss, what cache block should we replace with the data block read from main memory?

- Direct-mapped cache: Unique line.
- Associative caches:
  - Least recently used (LRU): Replace block in set corresponding to address which has not been read or written for the longest time.
  - Random: Select block in set randomly.
  - First-In First-Out (FIFO): Select block which has been in the set for the longest time.

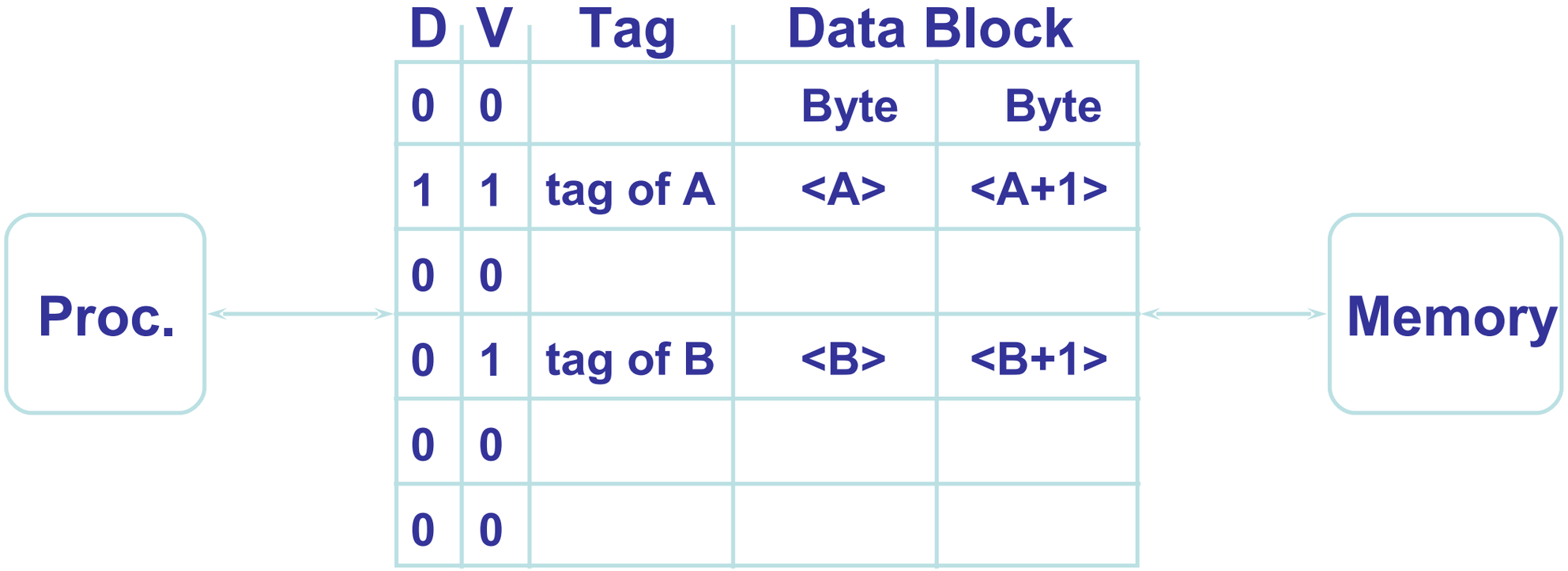
Write Policy

- What happens when we want to write data into a particular address?



- Possibilities are:
  - Write-Through: Writes go to main memory and cache (auto-save?)
  - Write-Back: Write cache, write main memory only when block is replaced.

Dirty Bits for Write-Back Caches



- Means “the data in this line has been changed but not written back to main memory yet”.
- In other words, cache data is more recent while memory data is “stale” for specific address/es.
- OK to keep data dirty, until line is chosen for replacement.
- When the line corresponding to A is replaced, its data block has to be written to main memory if its dirty bit is set.

The 3 C's: Causes of Cache Misses

- Compulsory
  - Because data is not yet in cache (e.g. program just started).
- Capacity
  - Because data needed doesn't fit in cache; was replaced since cache was full.
- Conflict (Collision)
  - Due to lack of associativity; non-related lines replace each other because their common set was full.

Fixing the 3 C's

- Compulsory
  - Increase block size (to pre-fetch more data).
  - This solution works well for instruction cache.
  - Disadvantages:
    - Bigger miss-penalty, more data is being written in cache.
    - Possibly more unnecessarily pre-fetched data.
    - If block size is too big without increasing the cache size, we start getting more capacity and conflict misses.
      - Cache gets full too quickly, fewer sets available.

Fixing the 3 C's

- Capacity
  - Increase cache size (obviously).
  - Only happens in fully-associative caches.
  - Disadvantages:
    - Bigger cache means longer delays.

Fixing the 3 C's

- Conflict
  - Increase associativity (more data slots per set).
  - Does not happen in a fully-associative cache (no specific line assignments).
  - Disadvantages:
    - More associativity means:
      - More hardware
      - Longer delays

The Three C's

For a <u>fixed block size</u> ...