High Performance Computing
Lecture 26

Matthew Jacob

Indian Institute of Science

Agenda

1. Program execution: Compilation, Object files, Function call and return, Address space, Data & its representation (4)
2. Computer organization: Memory, Registers, Instruction set architecture, Instruction processing (6)
3. Virtual memory: Address translation, Paging (4)
4. Operating system: Processes, System calls, Process management (6)
5. Pipelined processors: Structural, data and control hazards, impact on programming (4)
6. Cache memory: Organization, impact on programming (5)
7. Program profiling (2)
8. File systems: Disk management, Name management, Protection (4)
9. Parallel programming: Inter-process communication, Synchronization, Mutual exclusion, Parallel architecture, Programming with message passing using MPI (5)

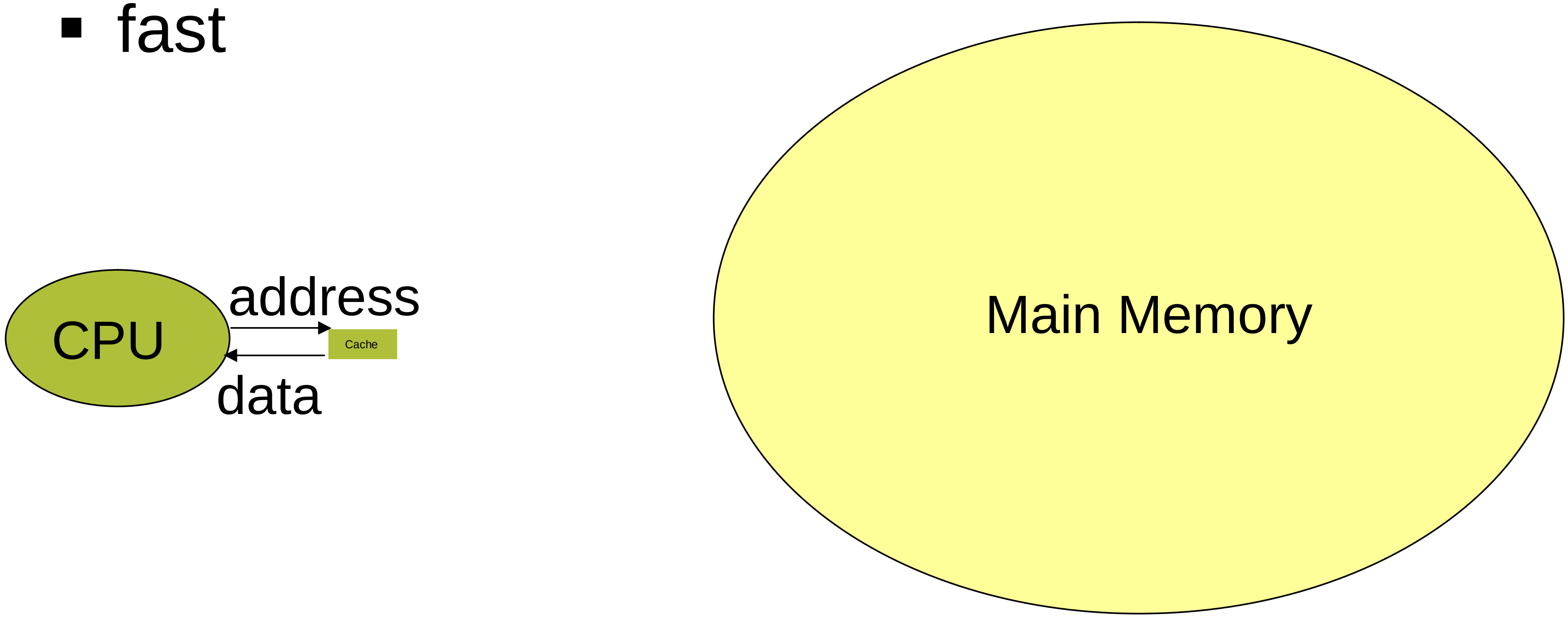
Cache Memory; Memory Hierarchy

- Recall: In discussing pipeline, we assumed that memory latency will be hidden so that it appears to operate at processor speed
- Cache Memory: HW that makes this happen
  - Design principle: Locality of Reference
  - Temporal locality: least recently used objects are least likely to be referenced in the near future
  - Spatial locality: neighbours of recently referenced locations are likely to be referenced in the near future

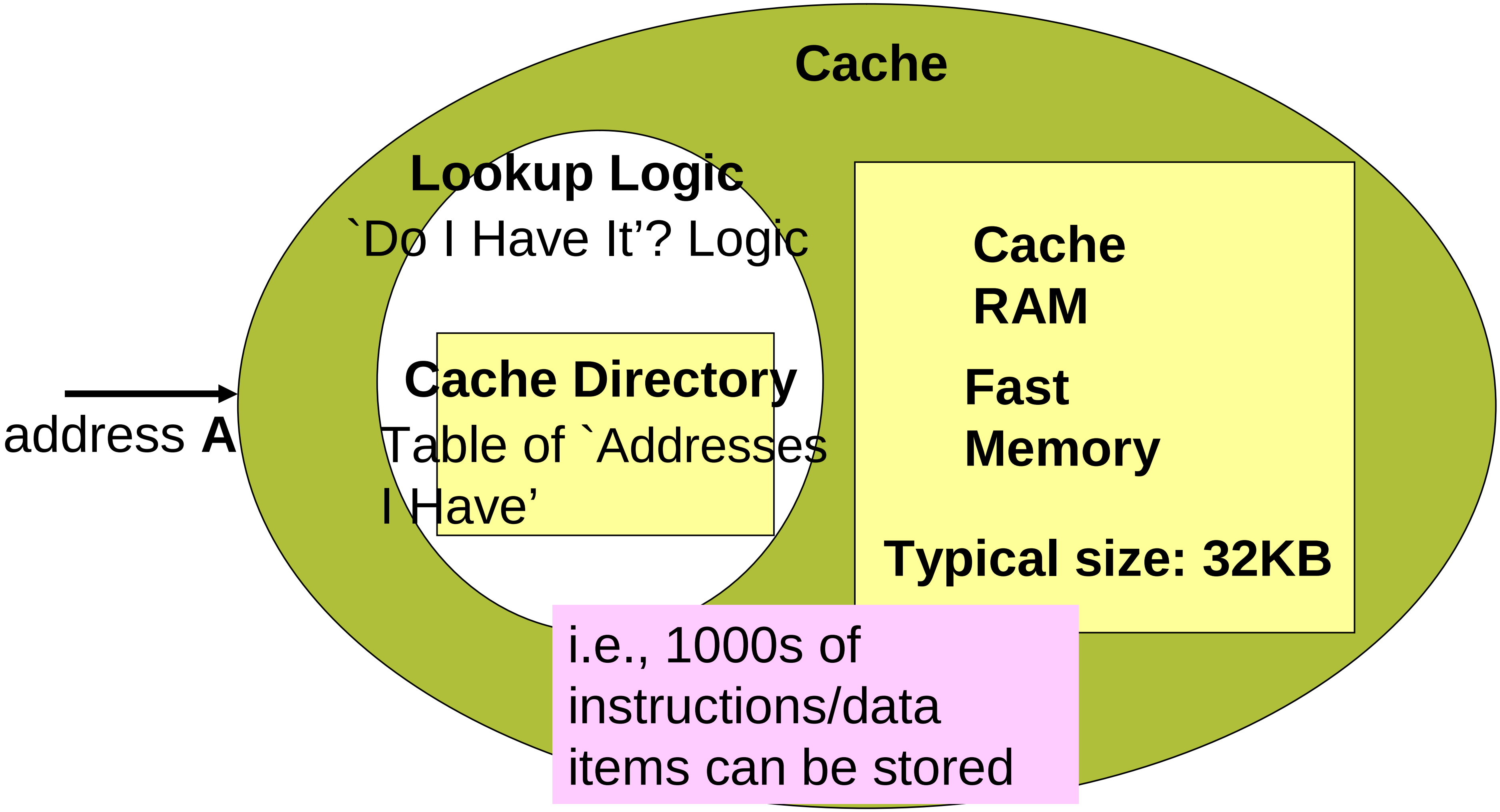
Cache Memory Exploits This

Cache: Hardware structure that provides memory objects that the processor references

- directly (most of the time)
- fast



Cache Design



address **A**

Cache

Lookup Logic

`Do I Have It'? Logic

Cache Directory

Table of `Addresses I Have'

Cache
RAM

Fast
Memory

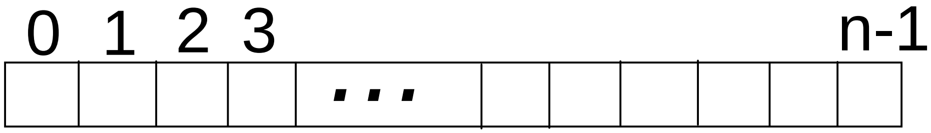
Typical size: 32KB

i.e., 1000s of instructions/data items can be stored

How to do fast Cache Lookup?

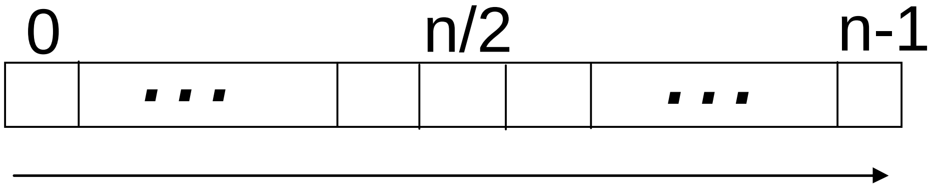
- Searching
  - Techniques to search for a specific value from a large collection of data
    - Searching for the word “phase” in a large text file
    - Searching for the number 10 in a large integer array
- Our specific search problem: looking for the address A among the 1000s of addresses in the cache directory
  - Requirement: The search must be FAST

Search Algorithms



1. Linear Search
   - Compare A with the first address in the cache directory
     - If they match, the search is successful
     - Else compare A with the second address in the directory
   - If you reach the last address in the directory without finding a match, the search was unsuccessful
   - Problem: Could take 1000s of comparisons

Search Algorithms



1. Linear Search
2. Binary Search
   - Sort the array of data items, say in increasing order
   - Compare A with the middle value
     - If they match, the search is successful
     - Else repeat for the appropriate half of the data
   - Much faster than linear search
   - Problem: Could take 10s of comparisons

Search Algorithms

1. Linear Search
2. Binary Search
3. Hash Search
   - May typically take just 1 comparison
   - The number of comparisons required doesn’t depend on the number of data values that we are searching among

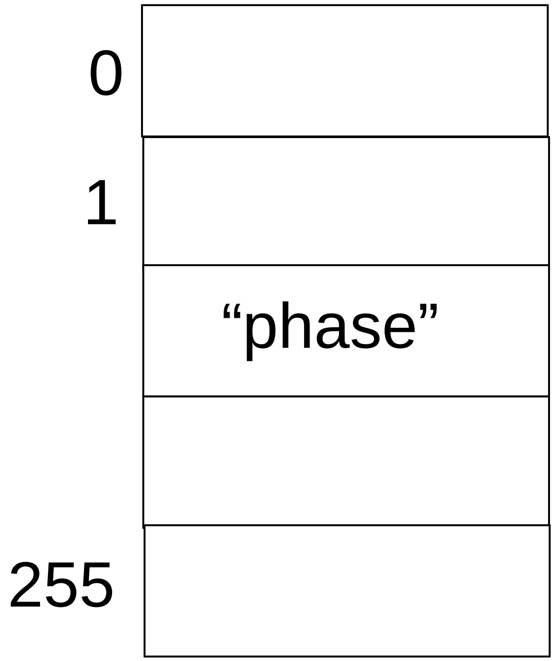
Hash Search

Hashing: A search technique that uses a hash table indexed into using a hash function

- Hash function
  - A function computed on the search string

Hash Table Example

- Example: Searching for the word “phase”
- Searching for a string of characters, $s_0 s_1 s_2 \ldots s_{len\text{-}1}$
- Hash function: $\sum_{i=0}^{len-1} s_i \ \text{div} \ \ len$



0

1

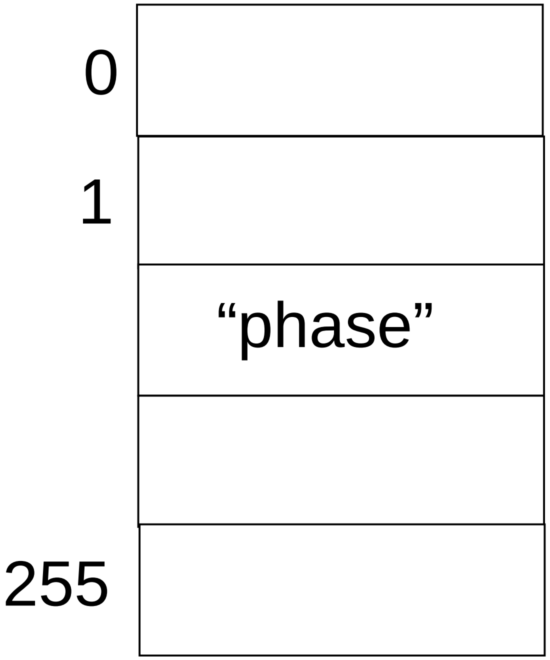
“phase”

255

- But, “phase” and “shape” will hash to the same index value
- This is called a hash collision

Hash Table Example

- Example: Searching for the word “phase”
- Searching for a string of characters, $s_0 s_1 s_2 \ldots s_{len-1}$
- Hash function: $\sum_{i=0}^{len-1} s_i \ \text{div} \ len$

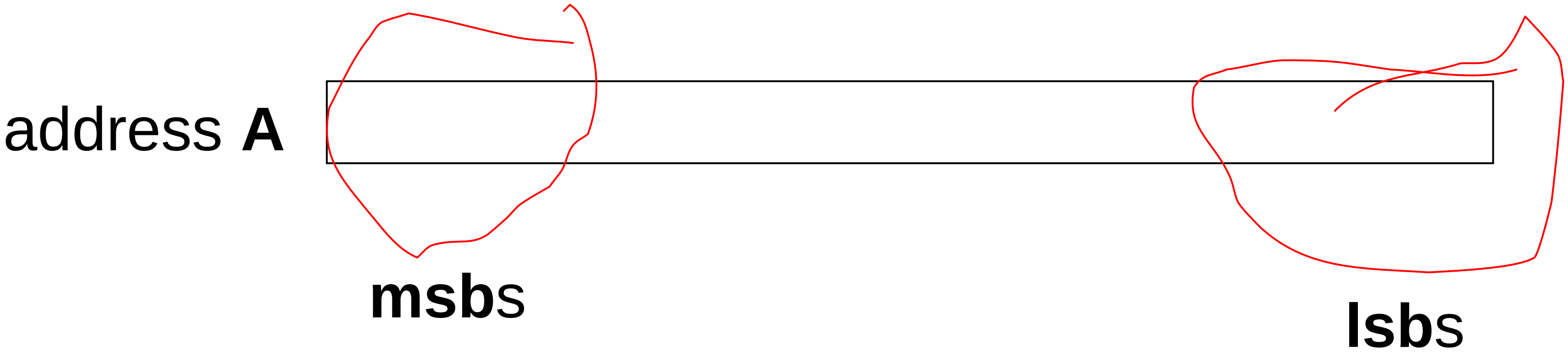


- But, “phase” and “shape” will hash to the same index value
- This is called a hash collision

How to do fast Cache Lookup?

- In the cache situation: cache lookup hardware is doing a search for an address A
- Simple hash function: select some of the bits of the address A
  - Which bits of address A?

How to do fast Cache Lookup?



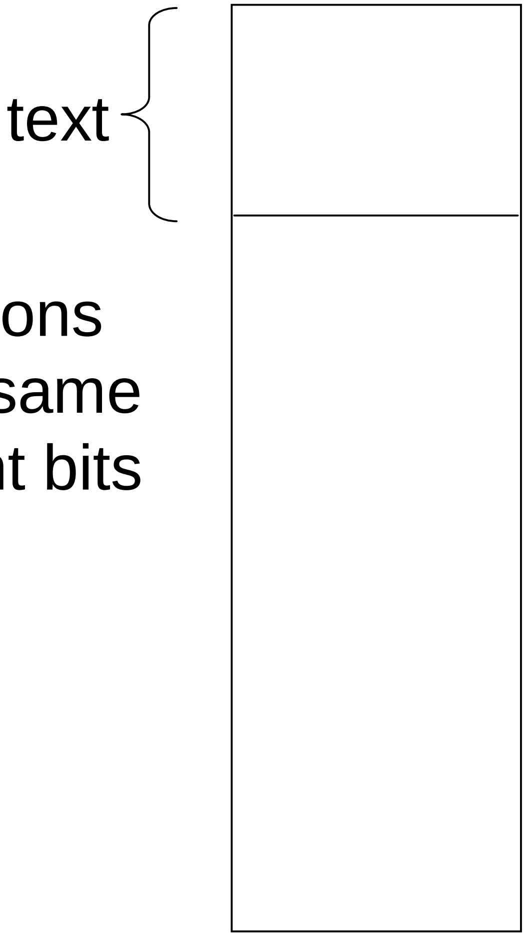
address **A**

msbs

lsbs

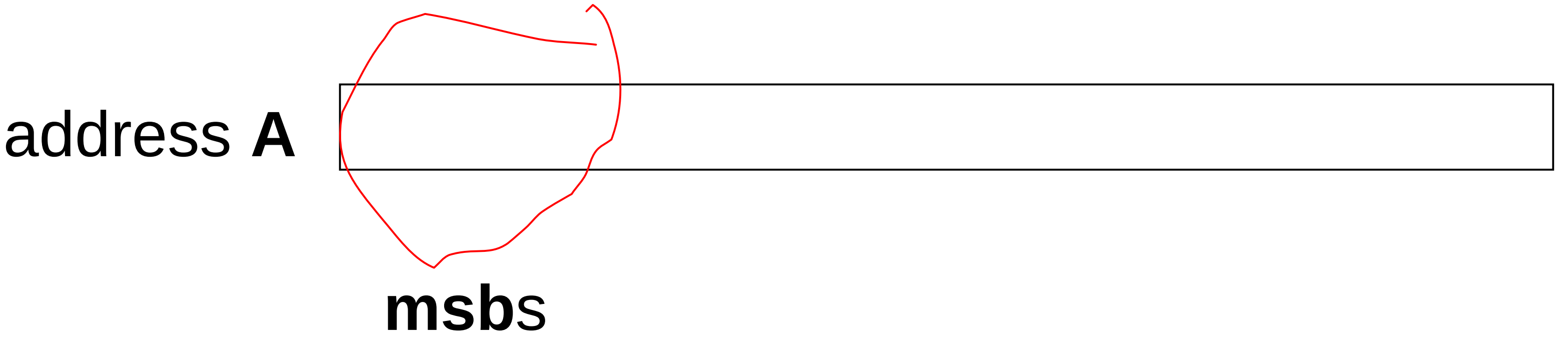
Most Significant Bits

Main memory addresses



All the instructions may have the same most significant bits

How to do fast Cache Lookup?



address **A**

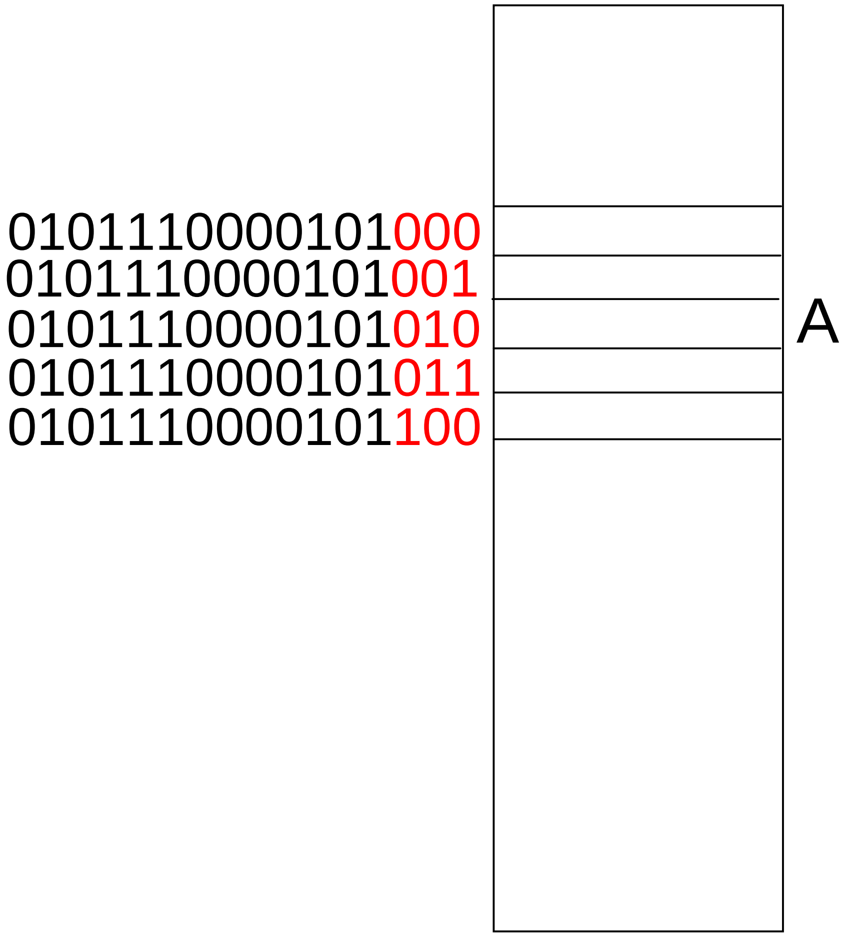
msbs

For a small program, everything would index into the same place in the hash table (collisions)

Using the Most Significant address bits for hashing is not a good idea

Least Significant Bits

Main memory addresses



0101110000101000

0101110000101001

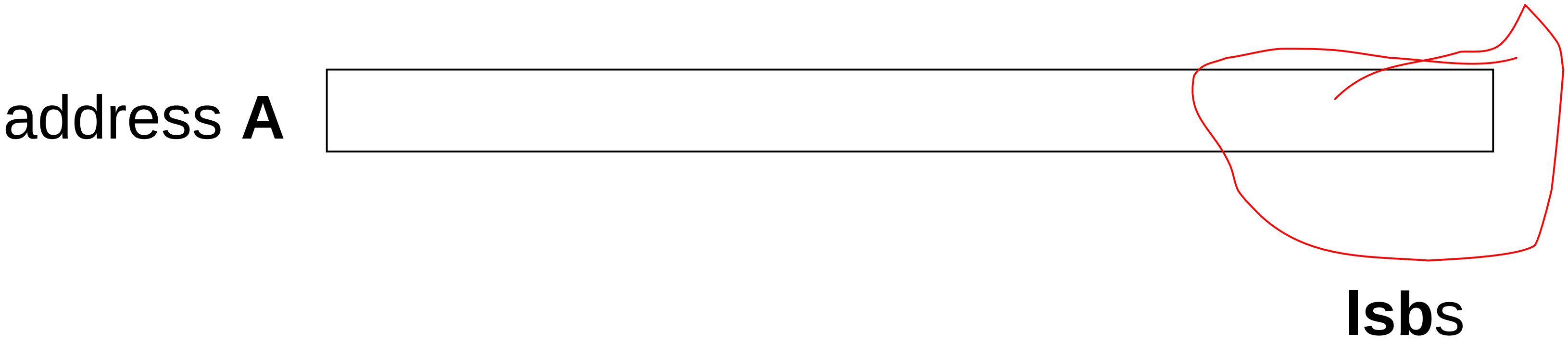
0101110000101010

0101110000101011

0101110000101100

A and its neighbours typically differ only in their least significant bits

How to do fast Cache Lookup?



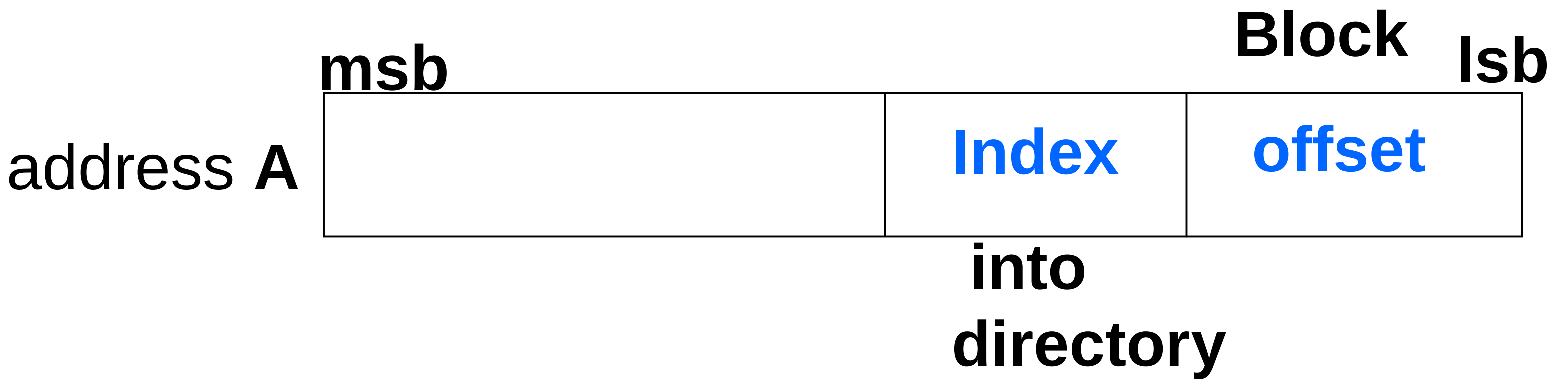
address **A**

lsbs

A and its neighbours possibly differ only in these bits; but they should be treated as one unit, not hashing into different hash table entries

Using the Least Significant address bits for hashing is not a good idea

Memory address



msb

Block

lsb

address **A**

Index

offset

into directory

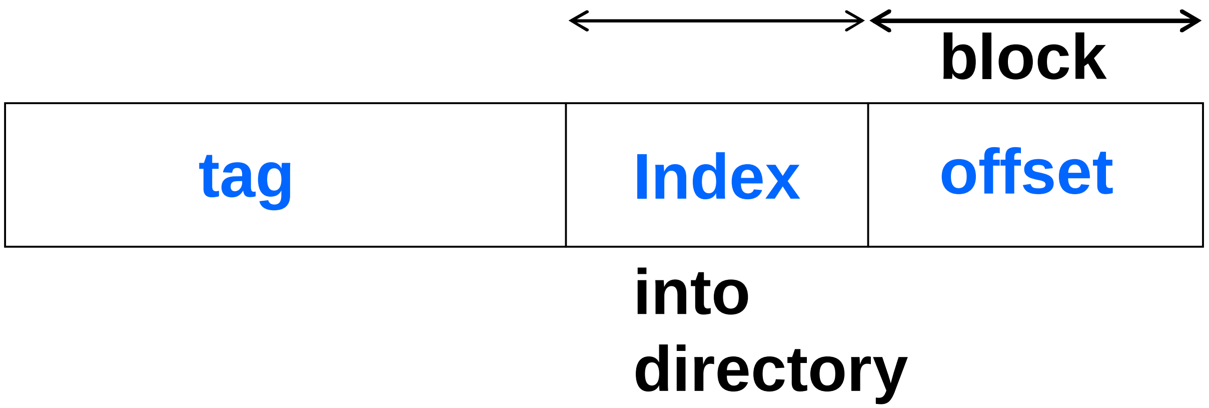
Block: Serves the same purposes in cache memory as the Page does in virtual memory

1. Reduce translation table size
2. Exploit spatial locality of reference

The Cache Directory contains one entry for each cache block, just like the Page Table contains one entry for each virtual page

Summing up

- A cache is organized in terms of blocks, memory locations that share the same address bits other than lsbs
- Main memory is also organized in terms of blocks
- The cache hardware views an address as

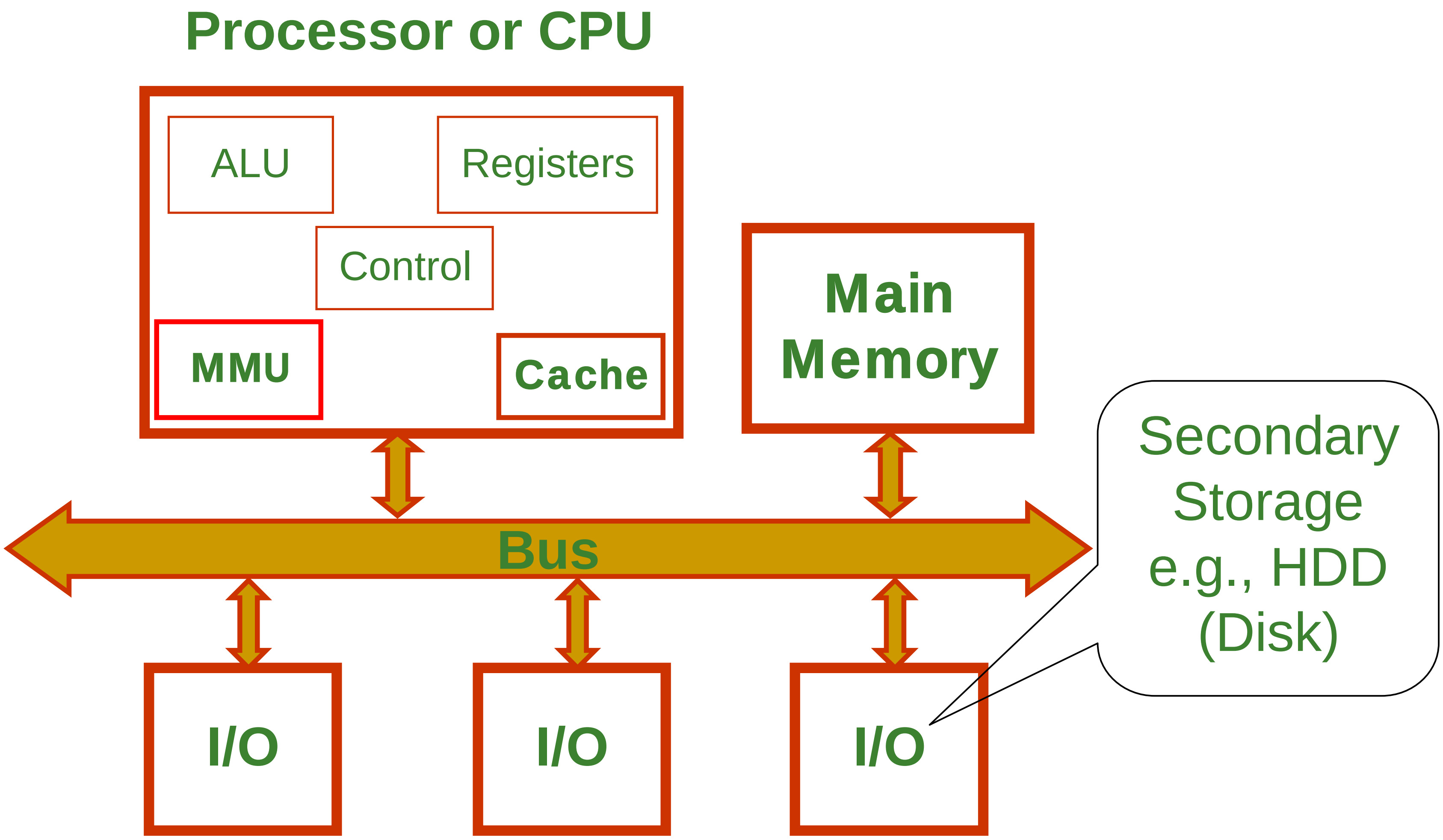


block

tag	Index	offset

into directory

Recall: Kinds of Memory



Registers, Cache, Main Memory

- Circuits that can remember things
  - Either by the state that a flip-flop is in or by the amount of charge stored
    - In both cases, the information is lost when the power source is turned off
- Uses a basic circuit for one bit of information
- This is replicated to remember a number of pieces of information that is each more than one bit in size

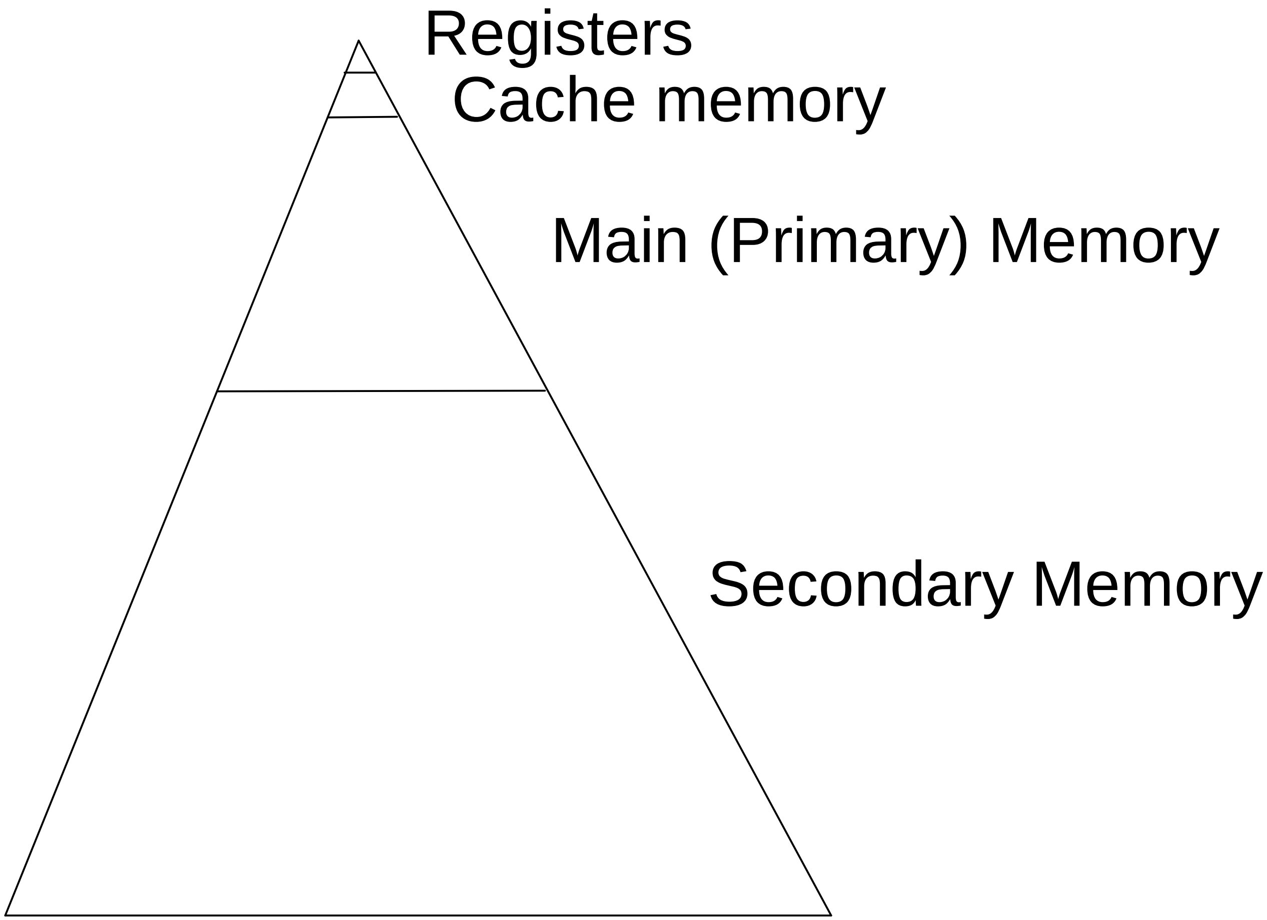
Main Memory

- In advertisements, you read of a computer with “2GB RAM”
- RAM: Random Access Memory
  - Able to handle arbitrarily ordered requests without favouring any particular request

Memory Hierarchy

- CPU registers
  - few in number (typically 16/32/128)
  - subcycle access time (nsec)
- Cache memory
  - on-chip memory
  - 10’s of KBytes (to a few MBytes)
  - access time of a few cycles
- Main memory
  - 100’s of MBytes storage (to a few GBytes)
  - access time several 10’s of cycles
- Secondary storage (like disk)
  - 100’s of GBytes storage (to a few TBytes)
  - access time of msecs

Memory Hierarchy


Registers
Cache memory
Main (Primary) Memory
Secondary Memory