Keep the most often-used data in a small, fast SRAM (often local to CPU chip). The reason this strategy works: **LOCALITY**.

- **Temporal locality**: If a location has been accessed recently, it is likely to be accessed (reused) soon
- **Spatial locality**: If a location has been accessed recently, it is likely that nearby locations will be accessed soon

**AMAT** (Average Memory Access Time) = HitTime + MissRatio * MissPenalty
Example: Direct-Mapped Caches

64-line direct-mapped cache → 64 indexes → 6 index bits

Read Mem[0x400C]

0100 0000 0000 1100

TAG: 0x40
INDEX: 0x3
OFFSET: 0x0

HIT, DATA 0x42424242

Would 0x4008 hit?
INDEX: 0x2 → tag mismatch
→ MISS

<table>
<thead>
<tr>
<th>Valid bit</th>
<th>Tag (24 bits)</th>
<th>Data (32 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x000058</td>
<td>0xDEADBEEF</td>
</tr>
<tr>
<td>1</td>
<td>0x000058</td>
<td>0x00000000</td>
</tr>
<tr>
<td>2</td>
<td>0x000058</td>
<td>0x00000007</td>
</tr>
<tr>
<td>3</td>
<td>0x000040</td>
<td>0x42424242</td>
</tr>
<tr>
<td>4</td>
<td>0x000007</td>
<td>0x6FBA2381</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>63</td>
<td>0x000058</td>
<td>0xF7324A32</td>
</tr>
</tbody>
</table>

Part of the address (index bits) is encoded in the location
Tag + Index bits unambiguously identify the data’s address

Block Size

- Take advantage of spatial locality: Store multiple words per data block
  - Another advantage: Reduces size of tag memory!
  - Potential disadvantage: Fewer blocks in the cache
- Example: 4-block, 16-word direct-mapped cache

Valid bit | Tag (26 bits) | Data (4 words, 16 bytes)

32-bit BYTE address
Tag bits: 26 (=32-4-2)
Index bits: 2 (4 indexes)

Block offset bits: 4 (16 bytes/block)
Problem 1. ★

(A) The timing for a particular cache is as follows: checking the cache takes 1 cycle. If there’s a hit the data is returned to the CPU at the end of the first cycle. If there’s a miss, it takes 10 additional cycles to retrieve the word from main memory, store it in the cache, and return it to the CPU. If we want an average memory access time of 1.4 cycles, what is the minimum possible value for the cache’s hit ratio?

Minimum possible value of hit ratio: \(0.96\)

\[
1.4 = 1 + (1 - \text{HR}) 
\times 10 \quad \Rightarrow \quad \text{HR} = 0.96
\]

(B) If the cache block size, i.e., words/cache line, is doubled but the total number of data words in the cache is unchanged, how will the following cache parameters change? Please circle the best answer.

- # of offset bits: UNCHANGED … +1 … -1 … 2x … 0.5x … CAN’T TELL
- # of tag bits: UNCHANGED … +1 … -1 … 2x … 0.5x … CAN’T TELL
- # of cache lines: UNCHANGED … +1 … -1 … 2x … 0.5x … CAN’T TELL

Consider a direct-mapped cache with 64 total data words with 1 word/cache line. This cache architecture is used for parts (C) through (F).

(C) If cache line number 5 is valid and its tag field has the value 0x1234, what is the address in main memory of the data word currently residing in cache line 5?

Main memory address of data word in cache line 5: \(0x123414\)

Tag: 24 bits, Index: 6 bits (000101), block offset: 2 bits (00)

The program shown on the right repeatedly executes an inner loop that sums the 16 elements of an array that is stored starting in location 0x310.

The program is executed for many iterations, then a measurement of the cache statistics is made during one iteration through all the code, i.e., starting with the execution of the instruction labeled outer_loop: until just before the next time that instruction is executed.

```
. = 0 // tell assembler to start at
   // address 0
outer_loop:
  addi x4, x0, 16 // initialize loop index J
  mv x1, x0 // x1 holds sum, initially 0
loop:
  subi x4, x4, 1 // decrement index
  slli x2, x4, 2 // convert to byte offset
  lw x3, 0x310(x2) // load value from A[J]
  add x1, x1, x3 // add to sum
  bne x4, x0, loop // loop until all words are summed
  j outer_loop // perform test again!
```

6.004 Worksheet - 4 of 6 - L15 – Memory Hierarchy
(D) In total, how many instruction fetches occur during one complete iteration of the outer loop? How many data reads?

**Number of instruction fetches:** __83__

Instruction fetch = $2 + 5 \times 16 + 1 = 83$

**Number of data reads:** __16__

(E) How many instruction fetch misses occur during one complete iteration of the outer loop? How many data read misses? Hint: remember that the array starts at address 0x310.

**Number of instruction fetch misses:** __4__

**Number of data read misses:** __4__

Data: 0x310, 0x314, 0x318, 0x31C, 0x320, 0x324, 0x328 ……
Instruction: 0x0 addi, 0x4 mv, 0x8 subi, 0xC slli, 0x10 lw, 0x14 add, 0x18 bne, 0x1C j ……

**Bold Blue addresses** are cache misses / conflicts.

(F) What is the hit ratio measured after one complete iteration of the outer loop?

**Hit ratio:** $\frac{91}{99}$, where $99 = 83 + 16$  $91 = (83-4) + (16-4)$
Problem 2.

The RISC-V Engineering Team is working on the design of a cache. They’ve decided that the cache will have a total of \(2^{10} = 1024\) data words, but are still thinking about the other aspects of the cache architecture.

First assume the team chooses to build a direct-mapped cache with a block size of 4 words.

(A) Please answer the following questions:

- Number of lines in the cache: 256
- Tag: 20bit, Index: 8bit, Offset: 4bit
- Number of bits in the tag field for each cache entry: 20

(B) This cache takes 2 clock cycles to determine if a memory access is a hit or a miss and, if it’s a hit, return data to the processor. If the access is a miss, the cache takes 20 additional clock cycles to fill the cache line and return the requested word to the processor. If the hit rate is 90%, what is the processor’s average memory access time in clock cycles?

\[
\text{Average memory access time assuming 90\% hit rate (clock cycles): } 4 \\
2 + (1-90\%) \times 20 = 4
\]