2024年度（令和6年）版

Ver. 2024-10-18a

Course number: CSC.T363

# コンピュータアーキテクチャ
# Computer Architecture

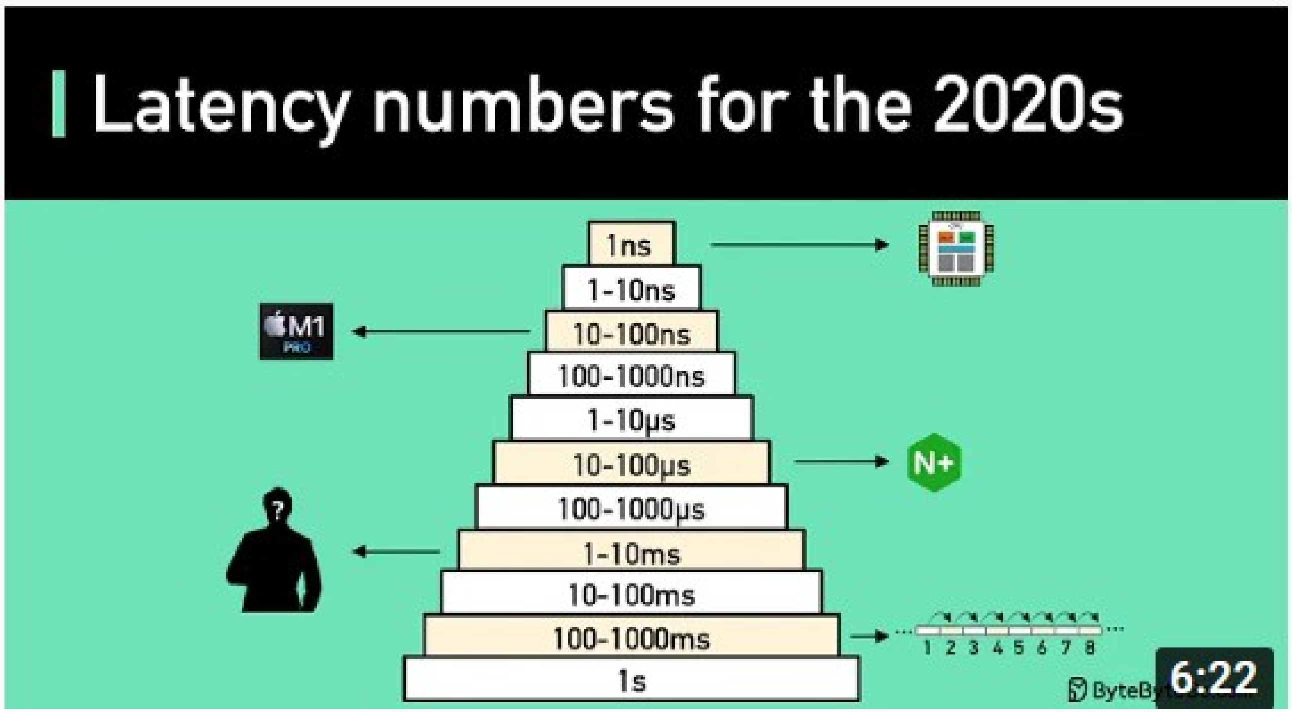
## 5. キャッシュ：セットアソシアティブ方式
## Caches: Set-Associative

www.arch.cs.titech.ac.jp/lecture/CA/
Tue 13:30-15:10, 15:25-17:05
Fri 13:30-15:10

吉瀬 謙二　情報工学系
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

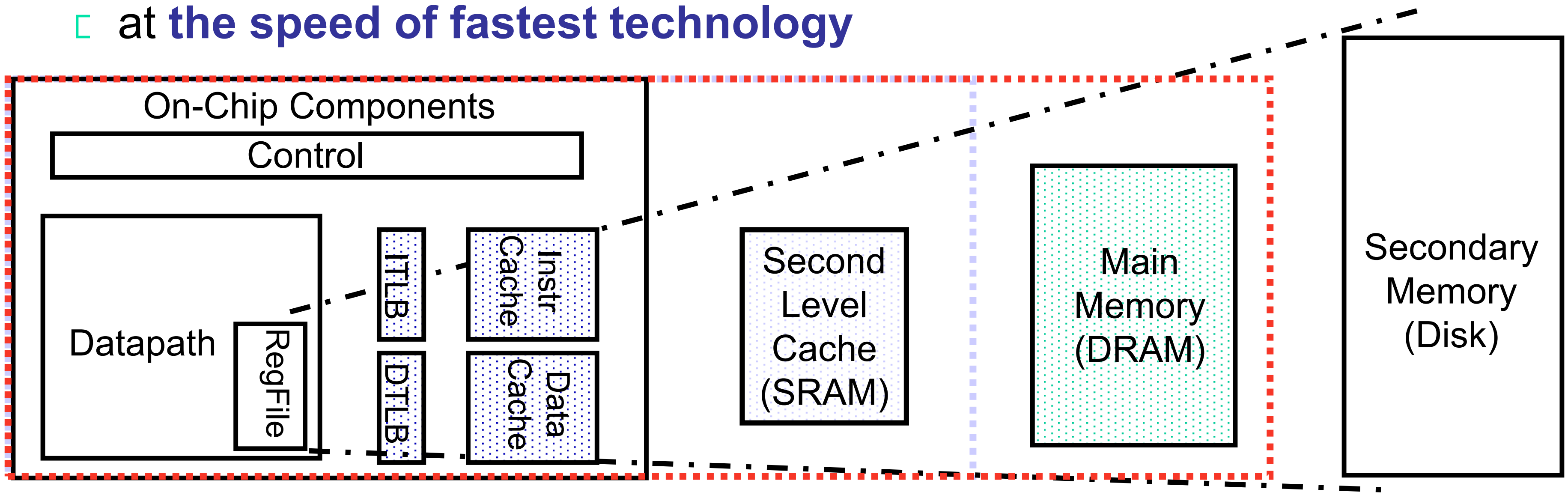
# 参考

- Latency Numbers Programmer Should Know
  - https://www.youtube.com/watch?v=FqR5vESuKe0



# A Typical Memory Hierarchy

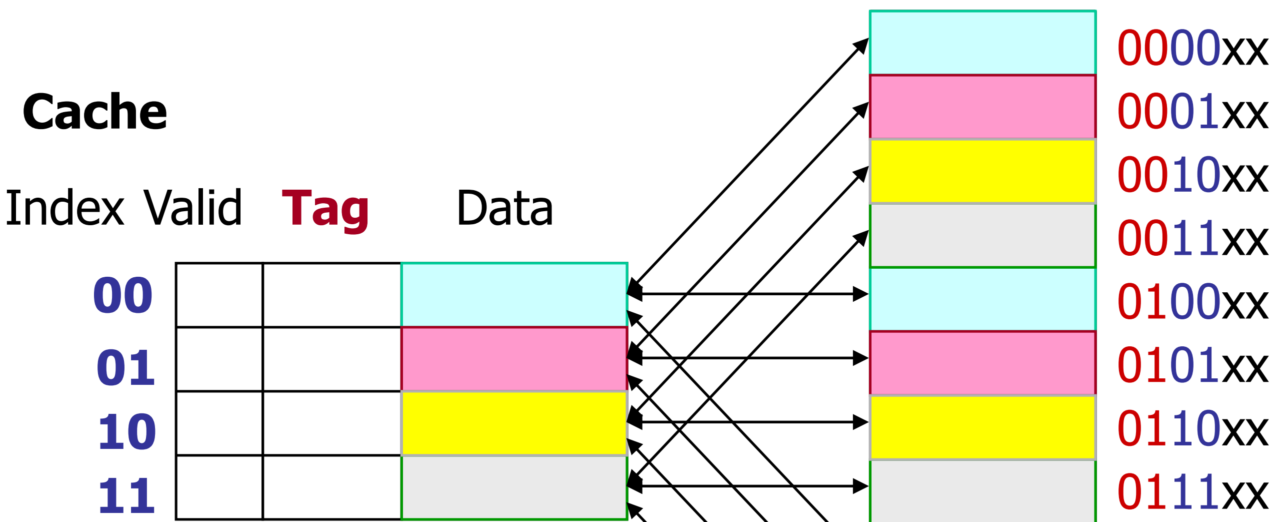
- By taking advantage of **the principle of locality** (局所性)
  - Present **much memory** in **the cheapest technology**
  - at **the speed of fastest technology**




| | | | | | |
|---|---|---|---|---|---|
| **Speed (%cycles):** | ½'s | 1's | 10's | 100's | 1,000's |
| **Size (bytes):** | 100's | K's | 10K's | M's | G's to T's |
| **Cost:** | highest | | | | lowest |

TLB: Translation Lookaside Buffer

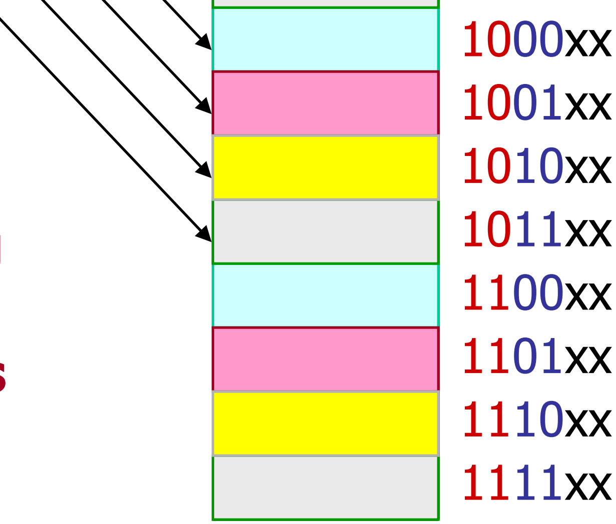
# Caching: Direct mapped (First Example)




Two low order bits define the byte in the word (32-b words)

Q1: Is it there?

Compare the cache **tag** to the **high order 2 memory address bits** to tell if the memory block is in the cache




Q2: How do we find it?

Use **next 2 low order memory address bits** – the **index** – to determine which cache block

(block address) modulo (# of blocks in the cache)

# Direct Mapped Cache Example

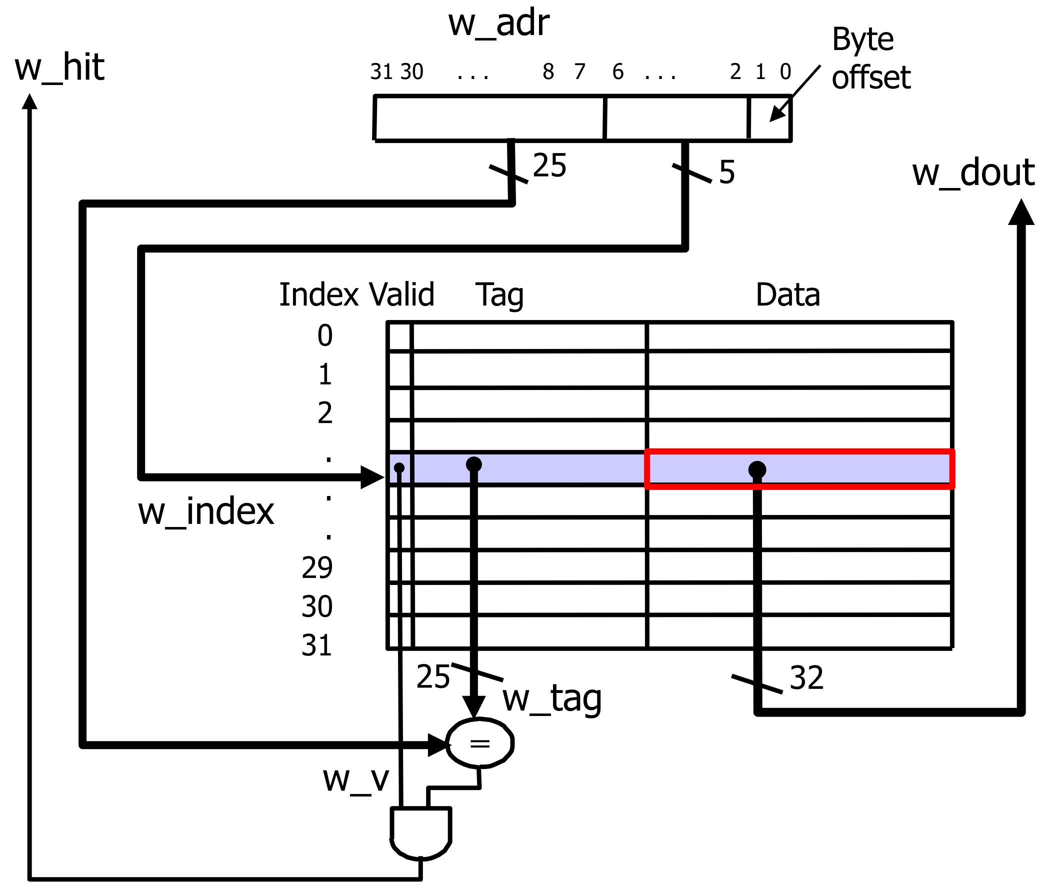
- One word/block, cache size = 1K words

```
module m_cache_direct_mapped_32 (
  input  wire        w_clk,
  input  wire        w_we,
  input  wire [31:0] w_adr,
  input  wire [4:0]  w_wadr,
  input  wire [57:0] w_wd,
  output wire        w_hit,
  output wire [31:0] w_dout
);

  reg [57:0] mem [0:31];
  integer i; initial for (i=0; i<32; i=i+1) mem[i] = 0;

  wire [4:0]  w_index = w_adr[6:2];
  wire        w_v;
  wire [24:0] w_tag;
  assign {w_v, w_tag, w_dout} = mem[w_index];
  assign w_hit = w_v & (w_adr[31:7]==w_tag);

  always @(posedge w_clk) if (w_we) mem[w_wadr] <= w_wd;
endmodule
```



# Direct Mapped Cache Example

- One word/block, cache size = 1K words

```
module m_cache_direct_mapped_32 (
  input  wire        w_clk,
  input  wire        w_we,
  input  wire [31:0] w_adr,
  input  wire [4:0]  w_wadr,
  input  wire [57:0] w_wd,
  output wire        w_hit,
  output wire [31:0] w_dout
);

  reg [57:0] mem [0:31];
  integer i; initial for (i=0; i<32; i=i+1) mem[i] = 0;

  wire [4:0]  w_index = w_adr[6:2];
  wire        w_v;
  wire [24:0] w_tag;
  assign {w_v, w_tag, w_dout} = mem[w_index];
  assign w_hit = w_v & (w_adr[31:7]==w_tag);

  always @(posedge w_clk) if (w_we) mem[w_wadr] <= w_wd;
endmodule
```

1R/1W memory

```
module m_cache_direct_mapped_32_v2 (
  input  wire        w_clk,
  input  wire        w_we,
  input  wire [31:0] w_adr,
  input  wire [57:0] w_wd,
  output wire        w_hit,
  output wire [31:0] w_dout
);

  reg [57:0] mem [0:31];
  integer i; initial for (i=0; i<32; i=i+1) mem[i] = 0;

  wire [4:0]  w_index = w_adr[6:2];
  wire        w_v;
  wire [24:0] w_tag;
  assign {w_v, w_tag, w_dout} = mem[w_index];
  assign w_hit = w_v & (w_adr[31:7]==w_tag);

  always @(posedge w_clk) if (w_we) mem[w_index] <= w_wd;
endmodule
```

1RW memory

# Sources of Cache Misses

- Compulsory (初期参照ミス, cold start or process migration, first reference):
  - First access to a block, "cold" fact of life, not a whole lot you can do about it
  - If you are going to run "millions" of instruction, compulsory misses are insignificant
- Conflict (競合性ミス, collision):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase **associativity**
- Capacity (容量性ミス):
  - Cache cannot contain all blocks accessed by the program
  - Solution: increase cache size

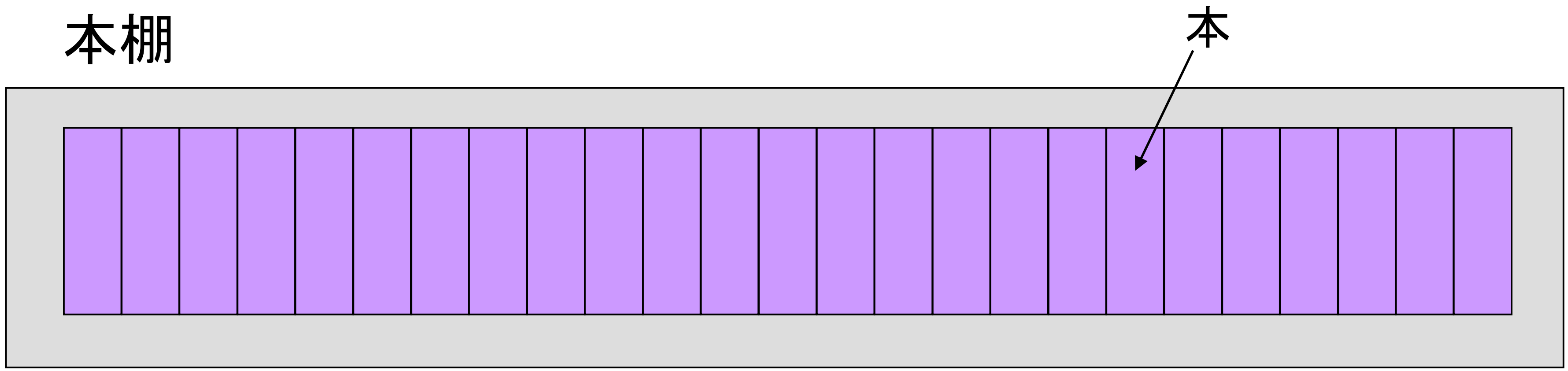
# Reducing Cache Miss Rates, Associativity

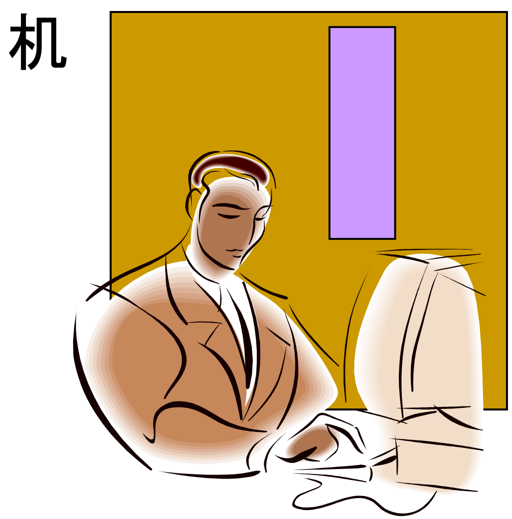
**Allow more flexible block placement**

- In a **direct mapped cache** a memory block maps to exactly **one cache block**
- At the other extreme, could allow a memory block to be mapped to any cache block – **fully associative cache**

- A compromise is to divide the cache into **sets** each of which consists of **n** "**ways**" (**n-way set associative**). A memory block maps to a unique set and can be placed in any way of that set (so there are **n choices**)

# Cache Associativity

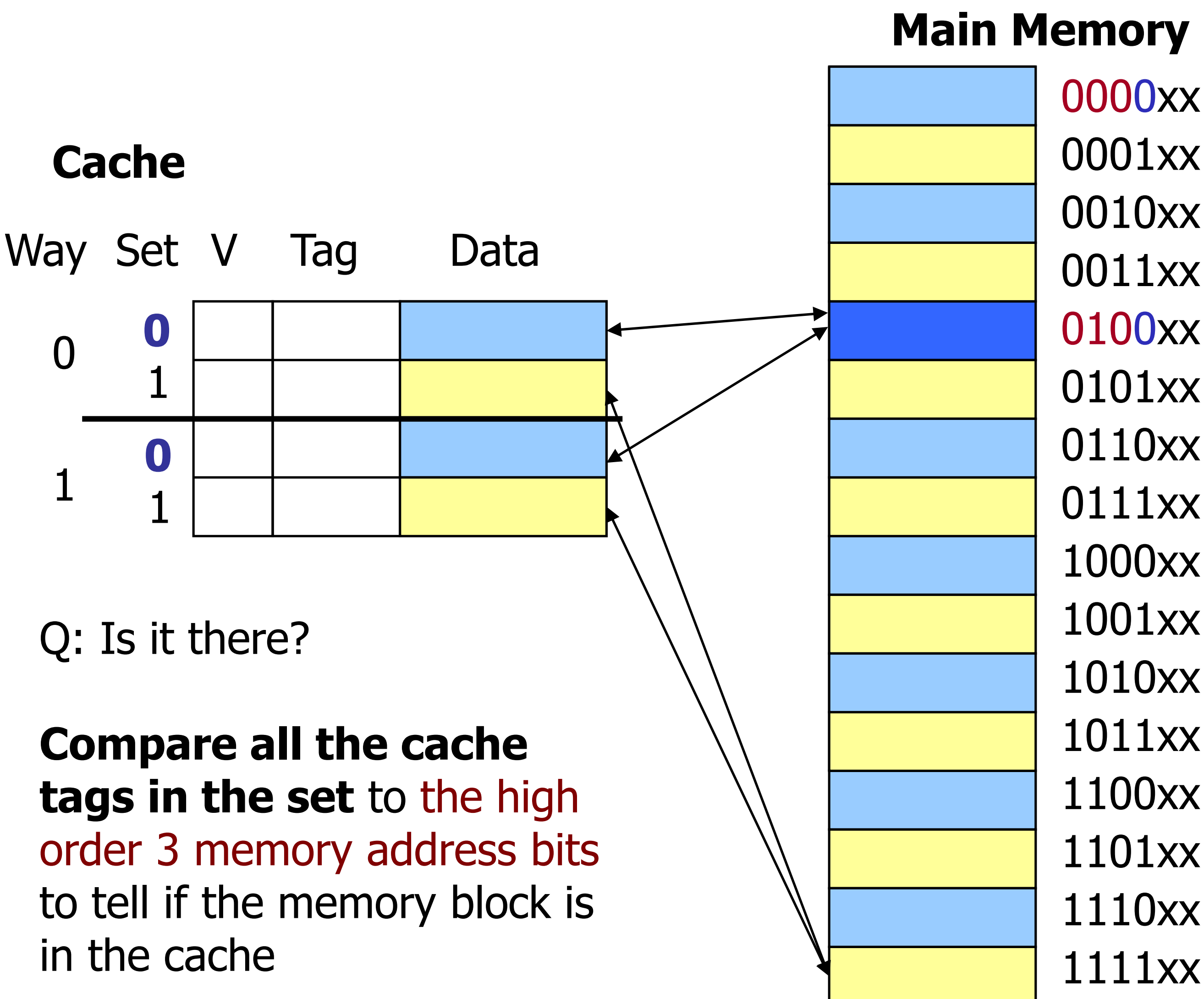



Direct Mapped


Set Associative

# Set Associative Cache Example




Two low order bits define the byte in the word (32-b words)
**One word blocks**

Q: Is it there?

**Compare all the cache tags in the set** to the high order 3 memory address bits to tell if the memory block is in the cache

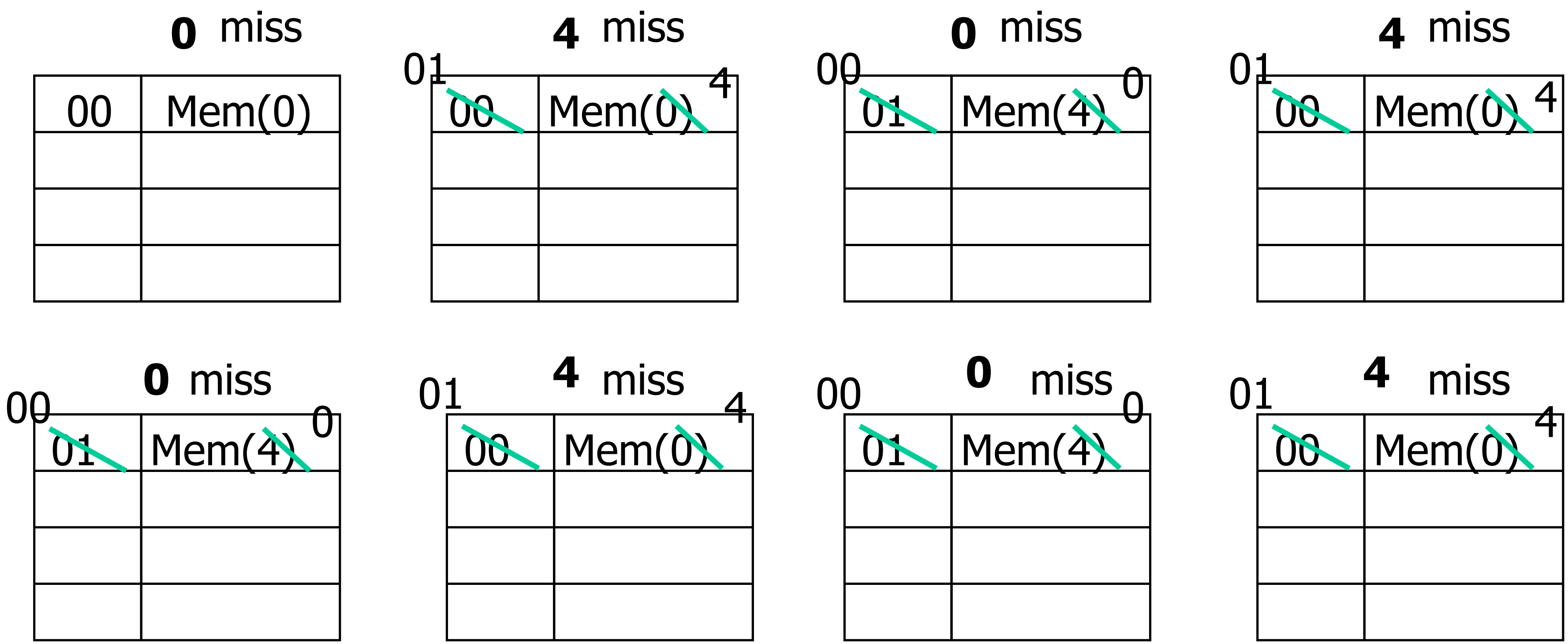
Q: How do we find it?

Use next 1 low order memory address bit to determine which cache set

# Another Reference String Mapping (Direct Mapped)

- Consider the main memory word reference string

0 4 0 4 0 4 0 4




- 8 requests, 8 misses
    - Ping pong effect due to **conflict** misses - two memory locations that map into the same cache block

# Another Reference String Mapping (Set Associative)

Consider the main memory word reference string

0 4 0 4 0 4 0 4

Start with an empty cache –
all blocks initially marked as not valid

**0** miss

| | |
|---|---|
| 000 | Mem(0) |
| | |
| | |
| | |

**4** miss

| | |
|---|---|
| 000 | Mem(0) |
| | |
| 010 | Mem(4) |
| | |

**0** hit

| | |
|---|---|
| 000 | Mem(0) |
| | |
| 010 | Mem(4) |
| | |

**4** hit

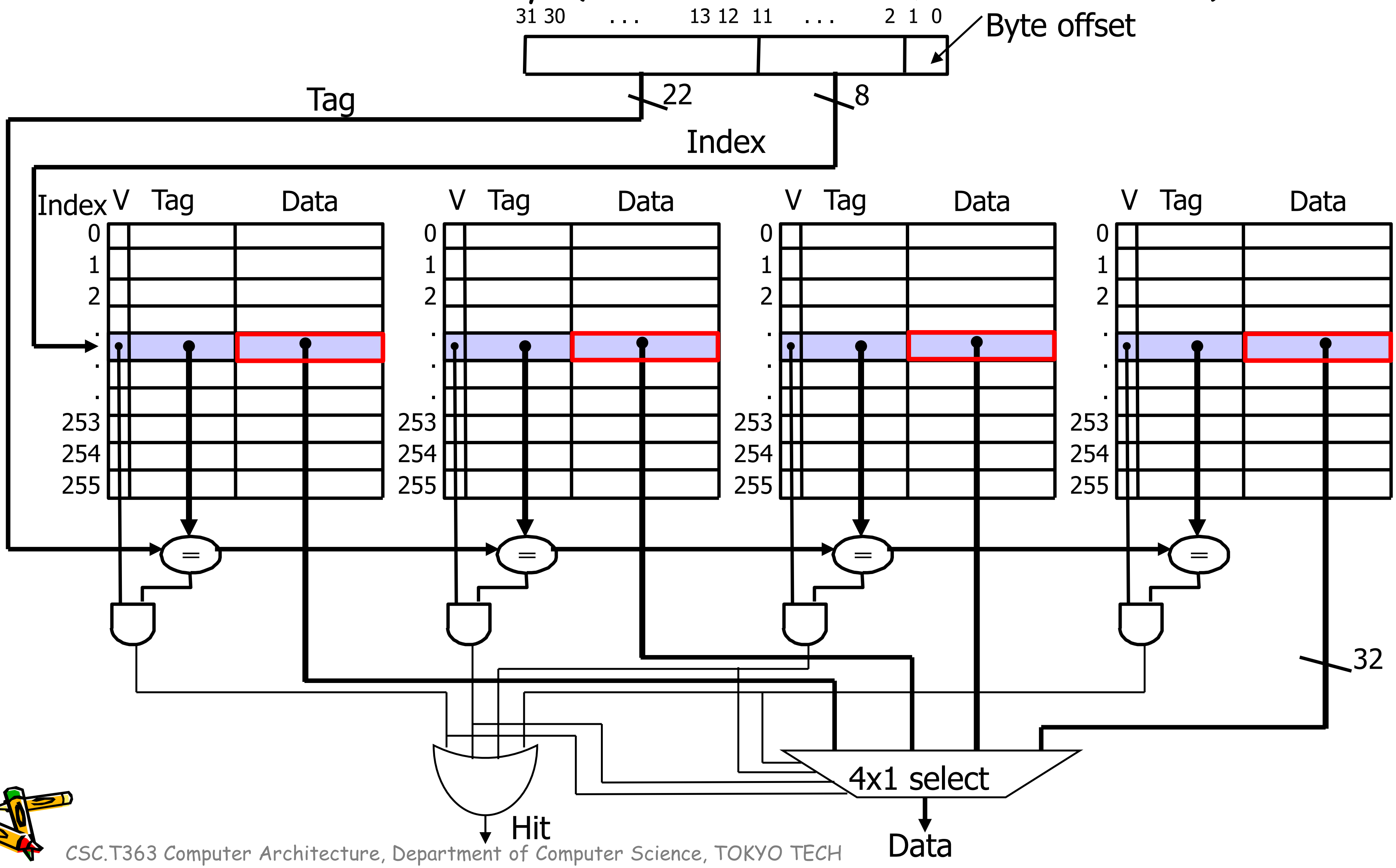
| | |
|---|---|
| 000 | Mem(0) |
| | |
| 010 | Mem(4) |
| | |

- 8 requests, 2 misses

- Solves the **ping pong effect** in a direct mapped cache due to conflict misses

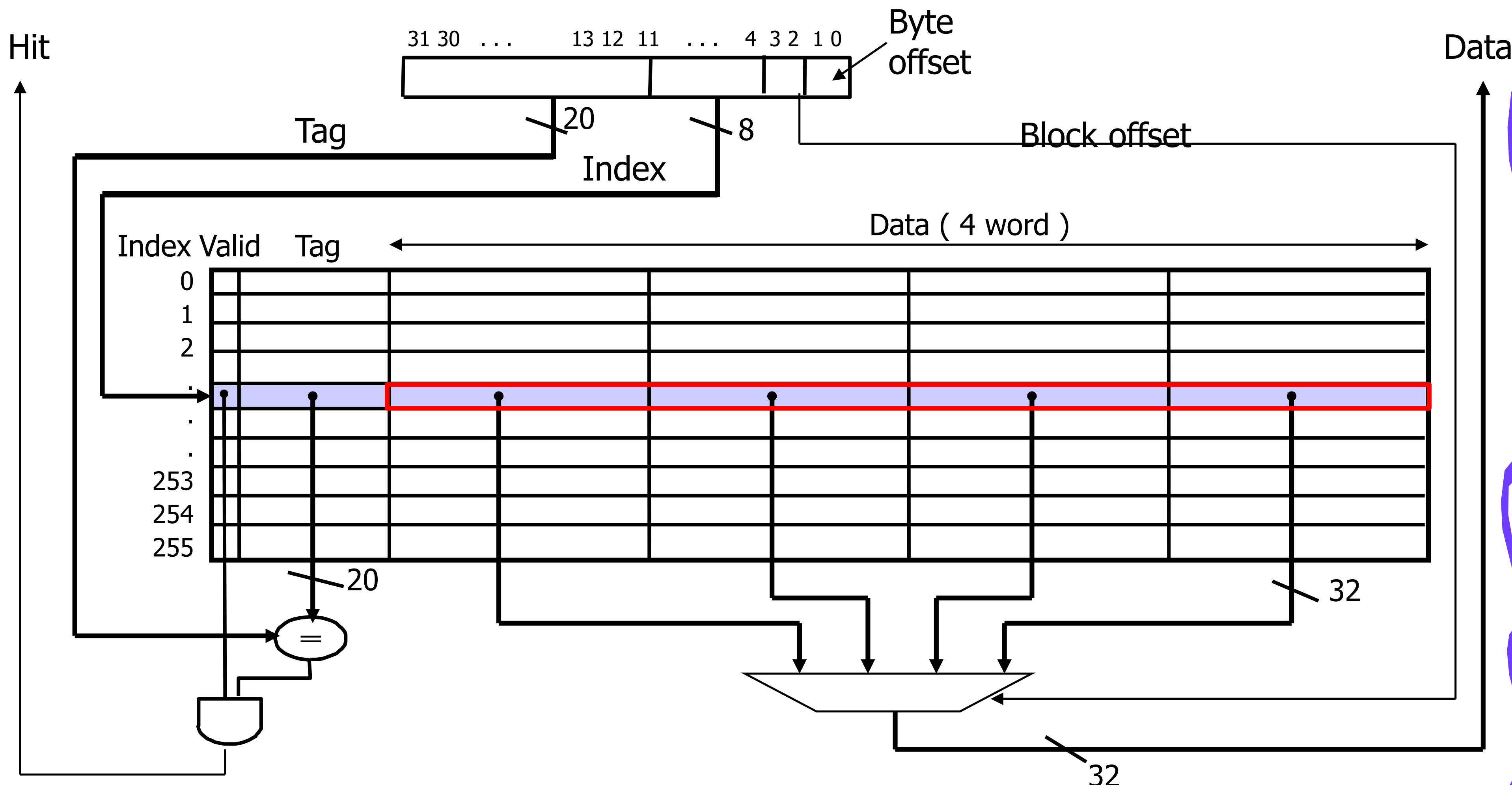
# Four-Way Set Associative Cache

$2^8$ = 256 sets each with four ways (each with one block, one word / block)



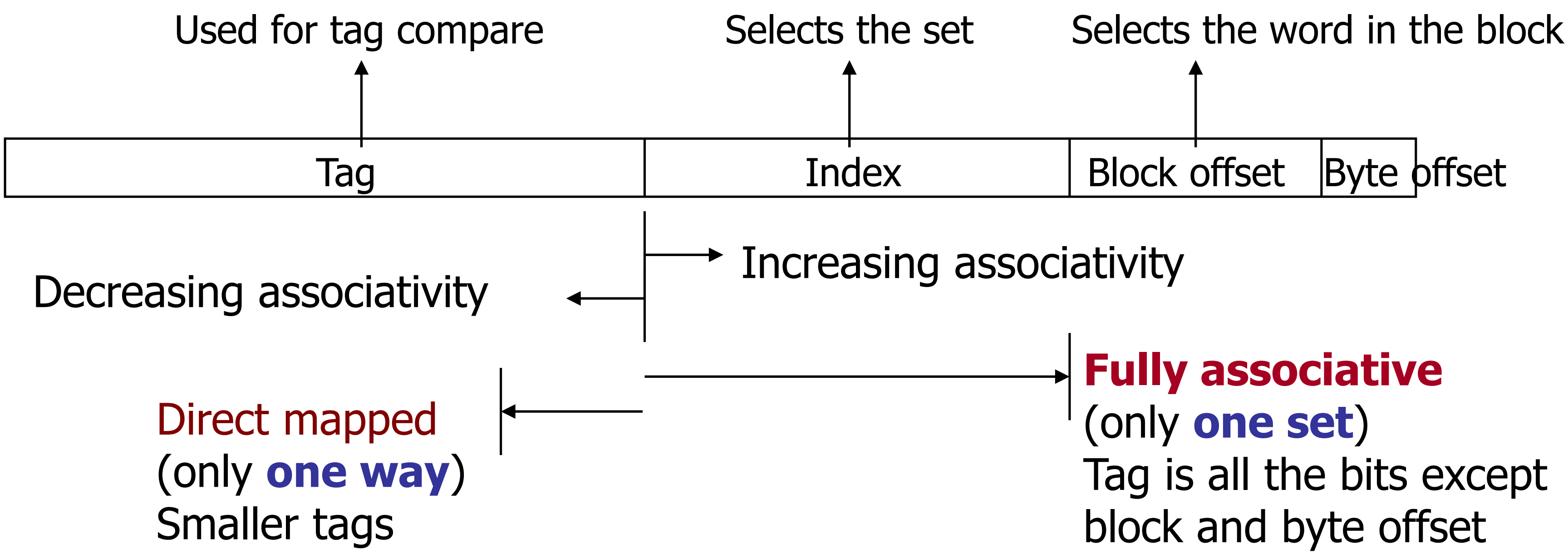
# Multiword Block Direct Mapped Cache

- Four words/block, cache size = 1K words



# Range of Set Associative Caches

For a fixed size cache



# Costs of Set Associative Caches

**N-way set associative** cache costs

- N comparators (delay and area)
- MUX delay (set selection) before data is available
- Data available **after** set selection and Hit/Miss decision.

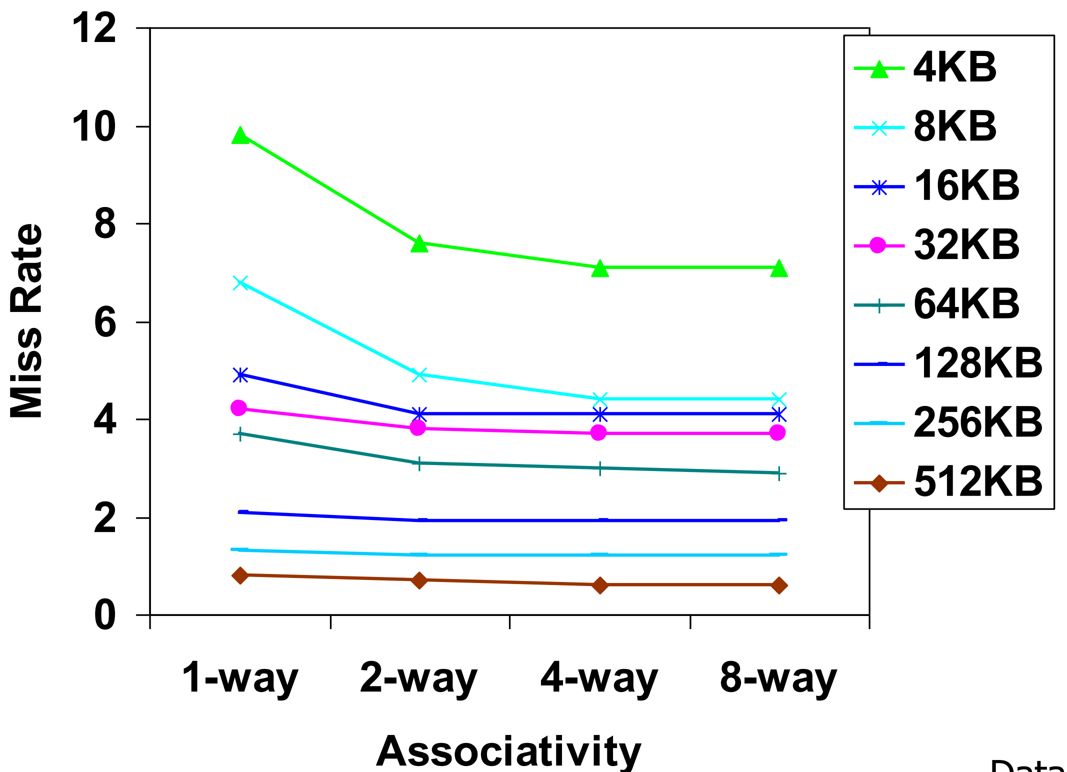
**When a miss occurs**,

which way's block do we pick **for replacement** ?

- **Least Recently Used (LRU):**
  - the block replaced is the one that has been unused for the longest time
    - Must have hardware to keep track of when each way's block was used
    - For **2-way set associative**, takes **one bit per set** → set the bit when a block is referenced (and reset the other way's bit)
- **Random**

# Benefits of Set Associative Caches

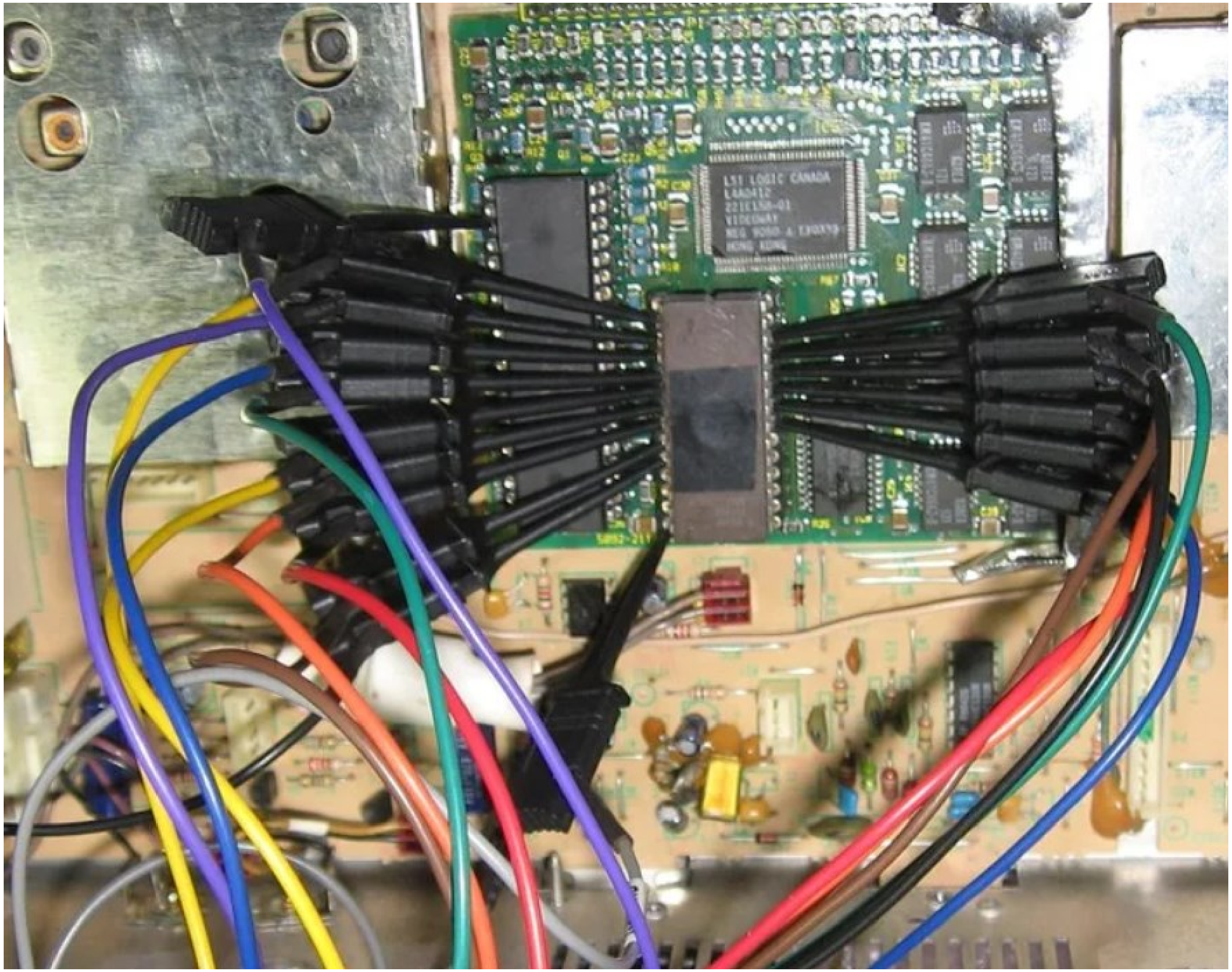
The choice of direct mapped or set associative depends on **the cost of a miss** versus **the cost of implementation**




Data from Hennessy & Patterson, *Computer Architecture*, 2003

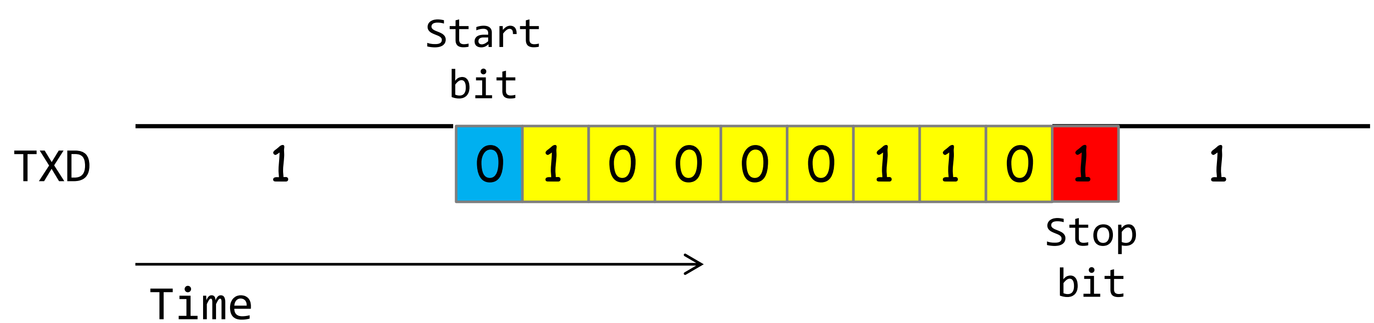
- Largest gains are in going from **direct mapped** to **2-way**

# Hardware debug



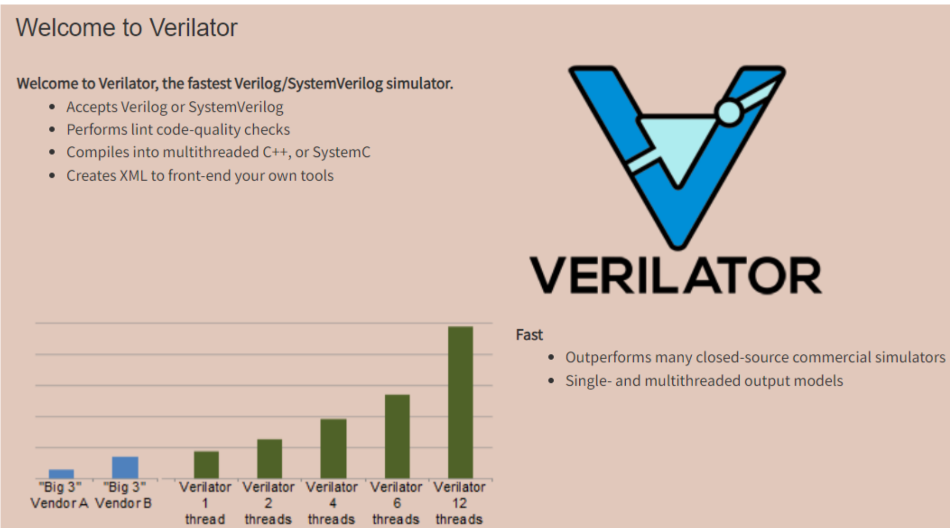
# UART (Universal Asynchronous Receiver/Transmitter)

- 調歩同期方式によるシリアル信号をパラレル信号に変換したり，その逆方向の変換をおこなう集積回路をUARTと呼ぶ．8ビット(1バイト)単位でデータを送信・受信する．
- UARTを用いることで，FPGAとコンピュータの間でのお手軽なデータ通信が可能．
- 例えば，'a' という文字を送信する場合，'a' は 8'h61, 8'b01100001（次スライドのASCII Tableを参照）なので，下図のタイミングで送信線TXDを制御する．
  - データが送信されるまで送信線TXD を1とする．
  - まず，青色で示した0（これをスタートビットと呼ぶ）を送信することで，データ送信の開始を明示．
  - 次に，黄色で示した様に送信したいデータ 8'b01100001 の最下位ビットから順番に送信する．
  - 最後に，赤色で示した1(これをストップビットと呼ぶ)を送信する．
- 1ビットを送受信するための時間間隔は送信側と受信側で同じレートを用いる．これをボー・レート (baud) と呼ぶ．例えば，1000 baud であれば，1ビット送信の間隔は 1msec となる．



# Verilator, the fastest Verilog/SystemVerilog simulator

In the ACRi room, verilator is installed in **/tools/cad/bin/**.




https://www.veripool.org/verilator/

```
$ /tools/cad/bin/verilator --binary --trace main3_v2.v
$ obj_dir/Vmain3_v2
   131 send 33
   226 recv 33 :00000000
   262 send 86
   357 recv 86 :00000000
   393 send 20
   488 recv 20 :00000000
   524 send 00
   619 recv 00 :00000000
   655 send 33
   750 recv 33 :00000000
   786 send 86
   881 recv 86 :00000000
   917 send 20
  1012 recv 20 :00000000
  1048 send 00
  1143 recv 00 :00000000
  1179 send 00
  1274 recv 00 :00000000
  1310 send 00
- main3_v2.v:47: Verilog $finish
```

# main3_v2.v

```
/**************************************************************************/
module m_main (
    input  wire w_clk       , // 100 MHz clock signal
    input  wire w_rxd
);

    wire        w_en      ;
    wire [7:0] w_dout     ;
    m_uart_rx m_uart_rx0 (w_clk, w_rxd, w_dout, w_en);

    reg [31:0] r_data = 0;
    always @(posedge w_clk) if (w_en) r_data <= {w_dout, r_data[31:8]};

    reg [31:0] r_sum = 0;

    // Complete here to calculate the checksum r_sum
    // Connect VIO to r_sum to verify the checksum value

    vio_0 vio_00 (w_clk, r_sum);

endmodule
```

```
/**************************************************************************/
`ifndef SYNTHESIS
module top;

    reg r_clk = 0; always #5 r_clk <= !r_clk;

    reg  [31:0] r_tc        = 0            ;
    reg  [15:0] r_cnt       = 1            ;
    reg   [3:0] r_bit       = 0            ;
    reg         r_en        = 0            ;
    reg  [63:0] r_tx_data   = {32'h00208633 , 32'h00208633} ; // add x12, x1, x2
    always @(posedge r_clk) begin
        r_tc <= r_tc + 1;
        r_cnt   <= (r_cnt>=`UART_CNT*13) ? 0 : r_cnt+1;
        if (r_cnt==0) begin
            r_bit   <= r_bit+1   ;
            r_en    <= 1         ;
        end else begin
            r_en    <= 0         ;
        end
        if (r_en) r_tx_data <= {8'h0, r_tx_data[63:8]};
    end

    wire w_txd;
    m_uart_tx m_uart_tx0 (r_clk, r_en, r_tx_data[7:0], w_txd);
    m_main    m          (r_clk, w_txd);

    initial $dumpfile("dump.vcd");
    initial $dumpvars(0, m);

    reg [2:0] r_rx_bit = 0;
    always @(posedge r_clk) begin
        if (r_en) begin
            $write("%6d send %x ¥n", r_tc, r_tx_data[7:0]);
        end
        if (m.w_en) begin
            $write("%6d recv %x :%08x¥n", r_tc, m.w_dout, m.r_sum);
            r_rx_bit <= r_rx_bit+1;
        end
        if (r_bit==10) $finish();
    end
endmodule

module  vio_0 (
    input wire w_clk,
    input wire [31:0] r_data
);
endmodule
`endif // !SYNTHESIS
```

# Clock rate is mainly determined by

- Switching speed of gates (transistors)
- The number of levels of gates
  - The maximum number of gates cascaded in series in any combinational logics.
  - In this example, the number of levels of gates is 3.
- Wiring delay and fanout
- The slowest of all paths is called the critical path