




# 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 (1) Caches: Set-Associative (1)



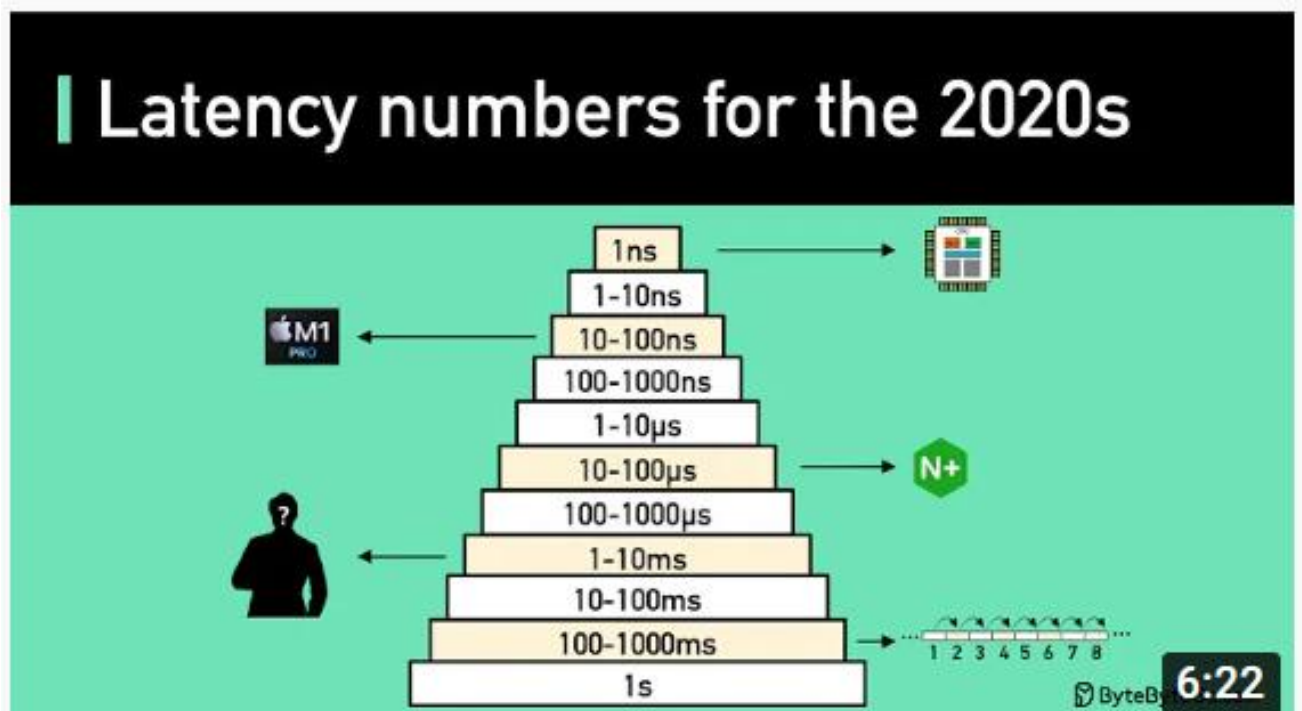
[www.arch.cs.titech.ac.jp/lecture/CA/](http://www.arch.cs.titech.ac.jp/lecture/CA/)  
Tue 13:30-15:10, 15:25-17:05  
Fri 13:30-15:10



吉瀬 謙二 情報工学系  
Kenji Kise, Department of Computer Science  
kise\_at\_c.titech.ac.jp

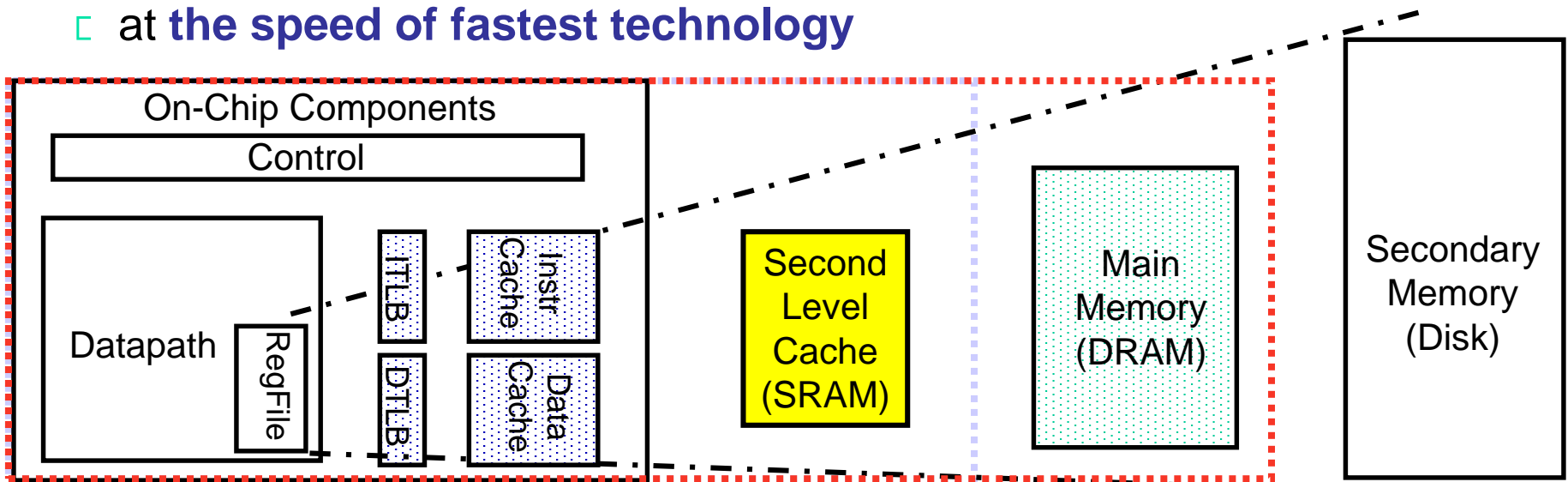
# 参考

- 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**



<b>Speed (%cycles):</b>	1/2's	1's	10's	100's	1,000's
<b>Size (bytes):</b>	100's	K's	10K's	M's	G's to T's
<b>Cost:</b>	highest				lowest

TLB: Translation Lookaside Buffer

# 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**

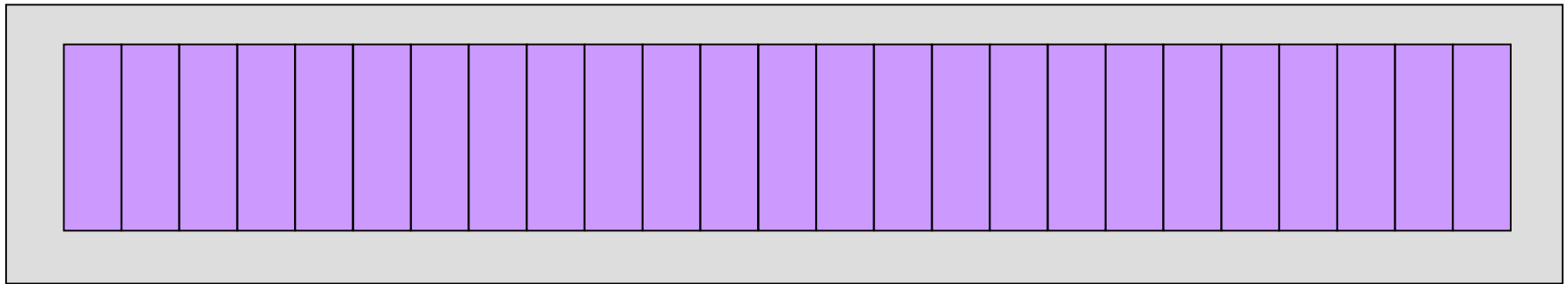
At the other extreme, could allow a memory block to be mapped to any cache block – **fully associative cache**

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



本棚

机



# Caching: Direct mapped (First Example)

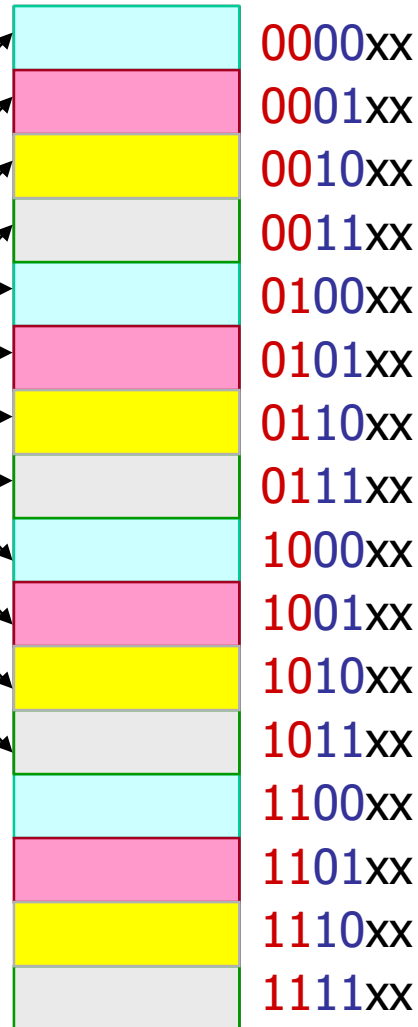


## Cache

Index	Valid	Tag	Data
00			Cyan
01			Pink
10			Yellow
11			Grey

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



## Main Memory

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

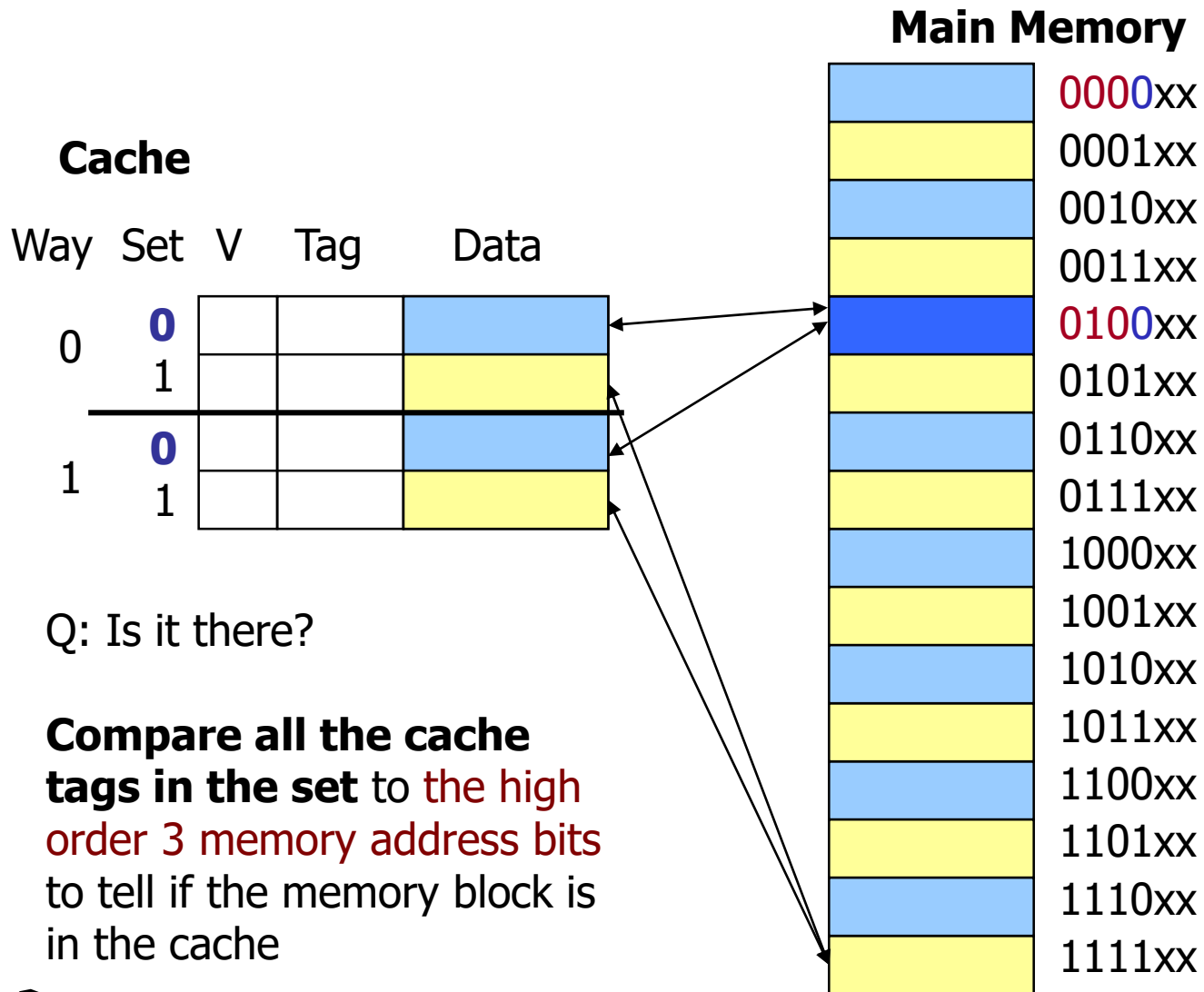
Q2: How do we find it?

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

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



# Set Associative Cache Example



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

Q: How do we find it?

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





# Another Reference String Mapping

- Consider the main memory word reference string

0 4 0 4 0 4 0 4

0 miss

00	Mem(0)

4 miss

<del>00</del>	<del>Mem(0)</del>

0 miss

<del>01</del>	<del>Mem(4)</del>

4 miss

<del>00</del>	<del>Mem(0)</del>

0 miss

<del>01</del>	<del>Mem(4)</del>

4 miss

<del>00</del>	<del>Mem(0)</del>

0 miss

<del>01</del>	<del>Mem(4)</del>

4 miss

<del>00</del>	<del>Mem(0)</del>

- 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

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)

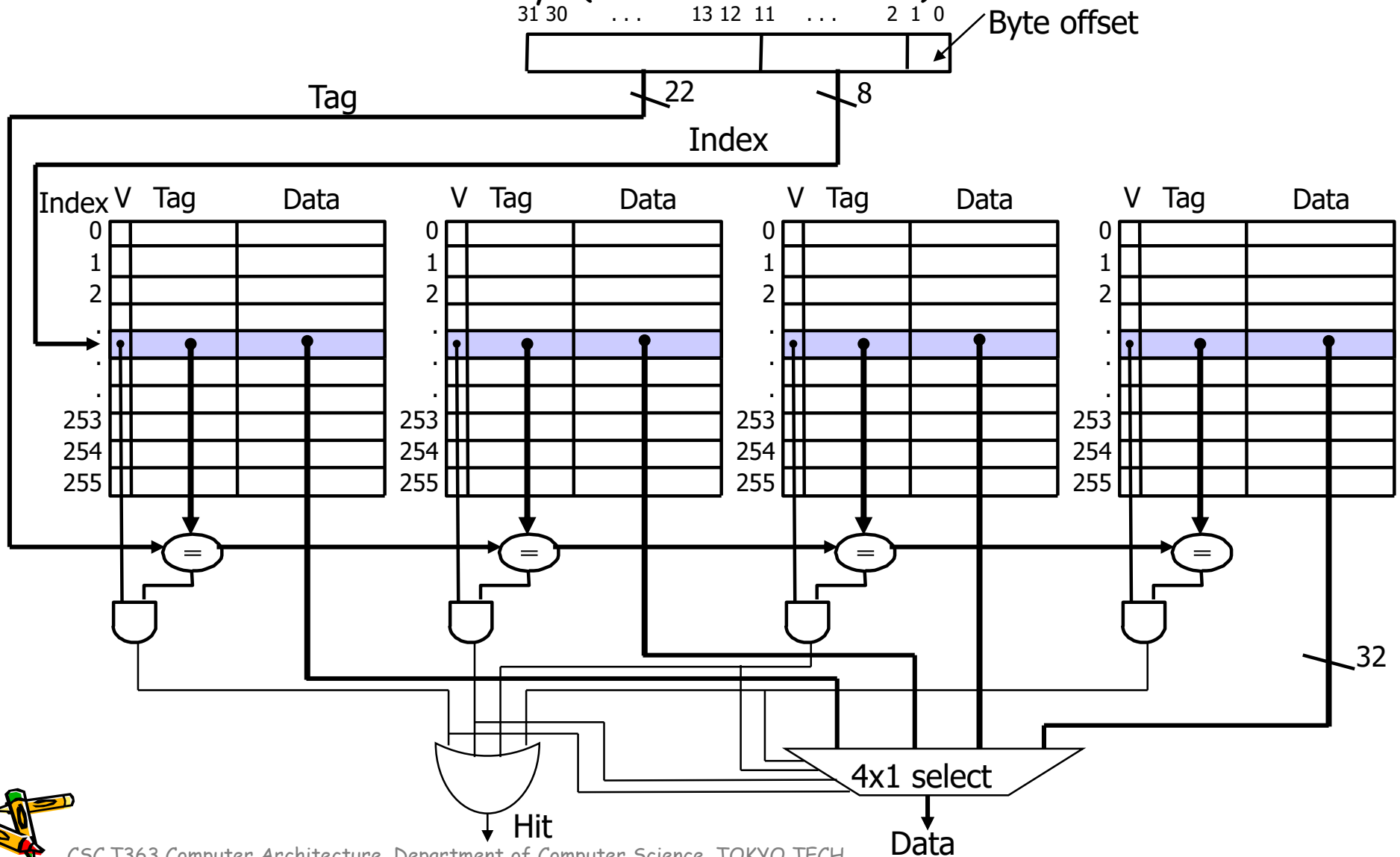
- 8 requests, 2 misses

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



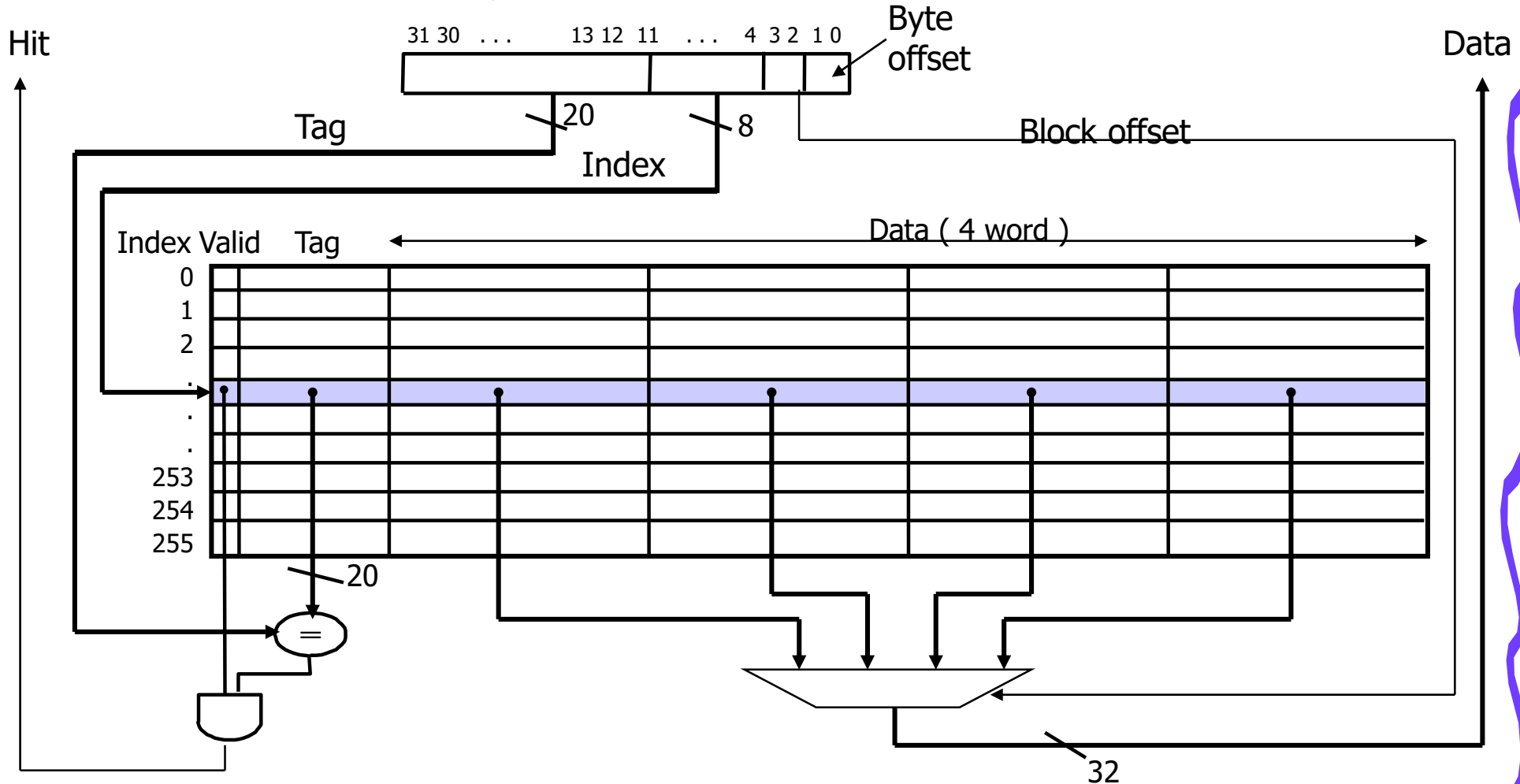
# Four-Way Set Associative Cache

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



# Multiword Block Direct Mapped Cache

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

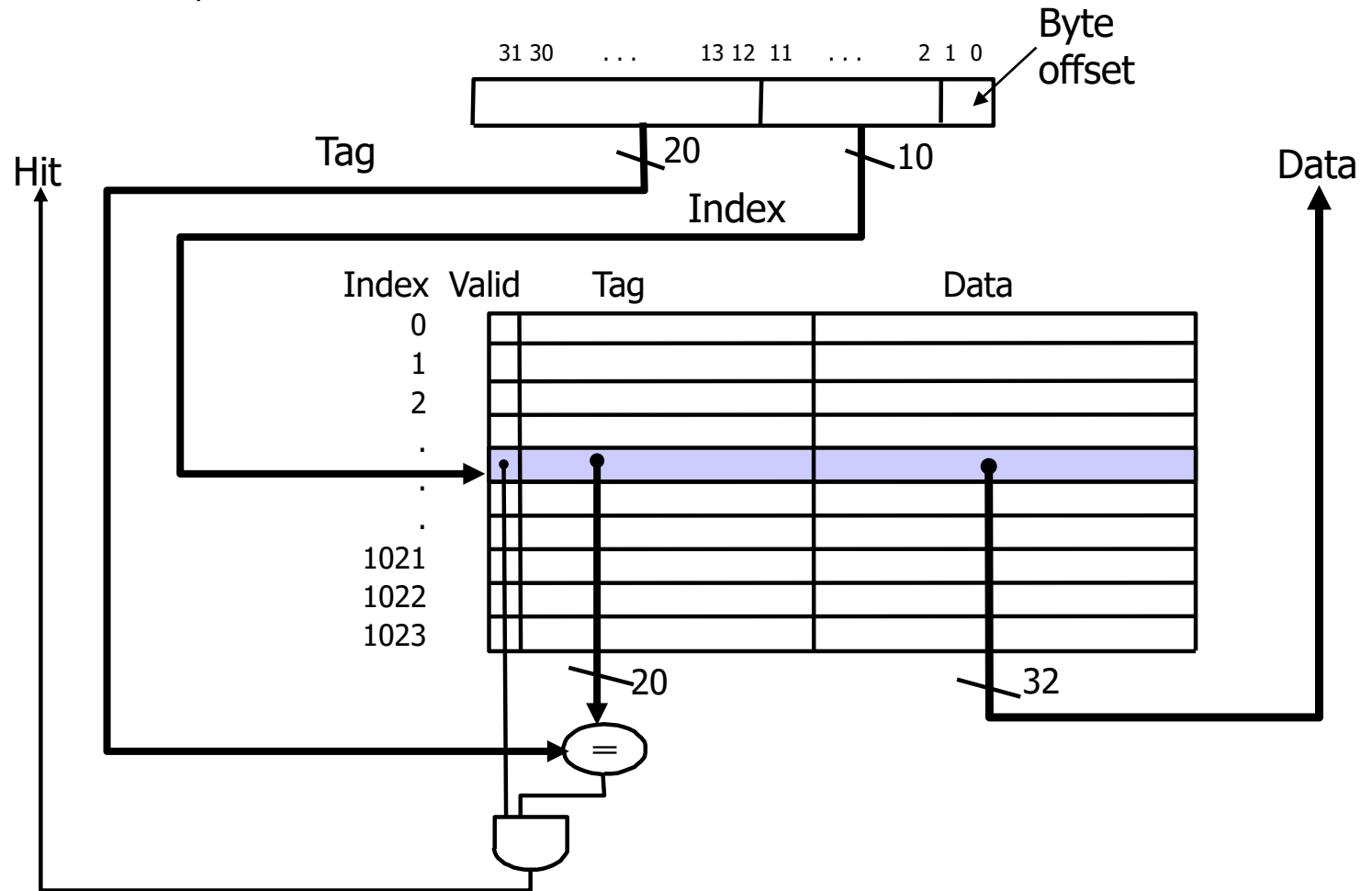


Data



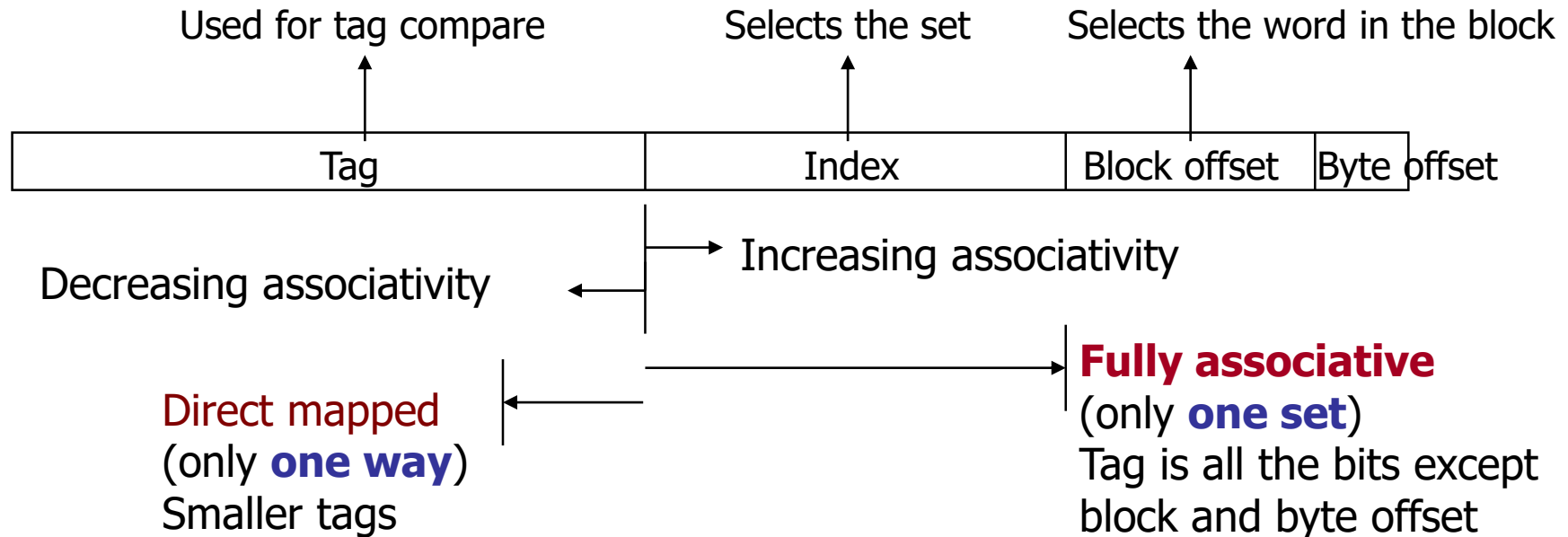
# MIPS Direct Mapped Cache Example

- One word/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**



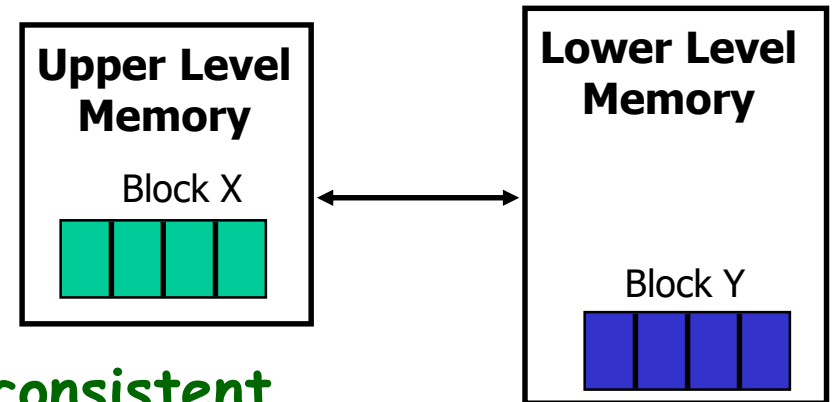
# Handling Cache Hits (Miss is the next issue)

- **Read hits (I\$ and D\$)**

- this is what we want!

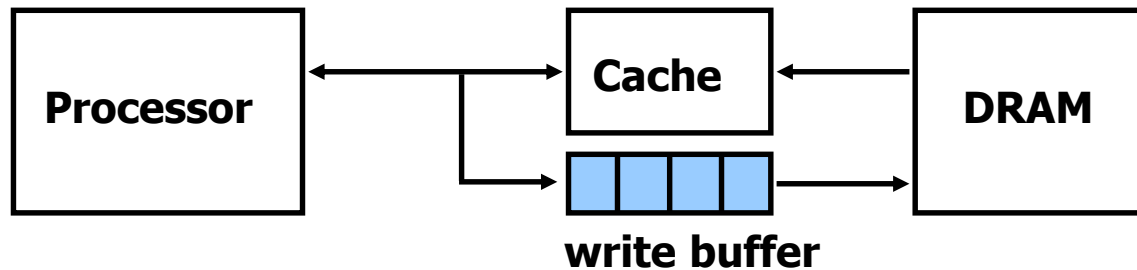
- **Write hits (D\$ only)**

- allow cache and memory to be **inconsistent**
  - write the data only into the cache block (**write-back**)
  - need a **dirty** bit for each data cache block to tell if it needs to be written back to memory when it is evicted
- require the cache and memory to be **consistent**
  - always write the data into both the cache block and the next level in the memory hierarchy (**write-through**) so don't need a dirty bit
  - writes run at the speed of the next level in the memory hierarchy – **so slow!** – or can use a **write buffer**, so only have to stall if the write buffer is full





# Write Buffer for Write-Through Caching



- **Write buffer** between the cache and main memory
  - Processor: writes data into the cache and the write buffer
  - **Memory controller**: writes contents of the write buffer to memory
- The write buffer is just a **FIFO**
  - Typical number of entries: 4
  - Works fine if **store frequency is low**
- Memory system designer's nightmare, Write buffer **saturation**
  - One solution is to use a write-back cache; another is to use an L2 cache

# Handling Cache Misses

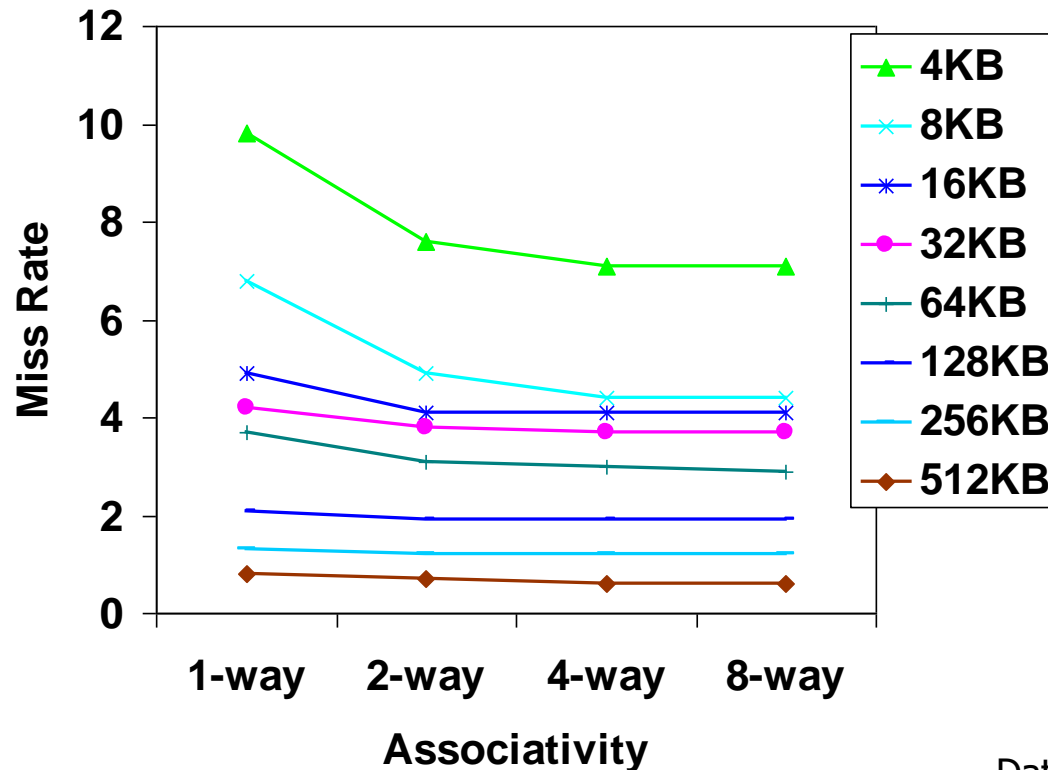


- **Read misses (I\$ and D\$)**
  - **stall** the entire pipeline, fetch the block from the next level in the memory hierarchy, install it in the cache and send the requested word to the processor, then let the pipeline resume
- **Write misses (D\$ only)**
  - **Write allocate**
    - (a) write the word into the cache updating both the tag and data, no need to check for cache hit, no need to stall
    - (b) **stall** the pipeline, fetch the block from next level in the memory hierarchy, install it in the cache, write the word from the processor to the cache, then let the pipeline resume
  - **No-write allocate** – skip the cache write and just write the word to the write buffer (and eventually to the next memory level), no need to stall if the write buffer isn't full; must invalidate **the cache block** since it will be **inconsistent**



# Benefits of Set Associative Caches

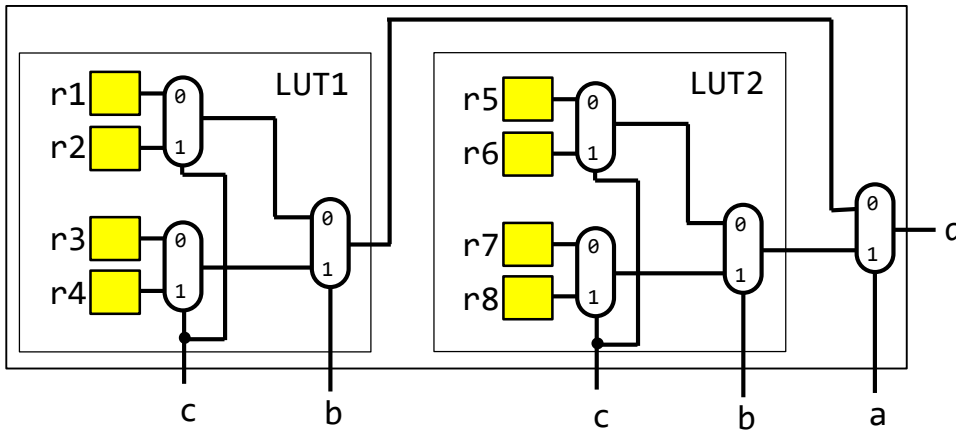
The choice of direct mapped or set associative depends on the cost of a miss versus the cost of implementation



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

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

# ルックアップテーブル (Lookup Table, LUT)



2個の2入力のLUTで3入力のLUTを構成

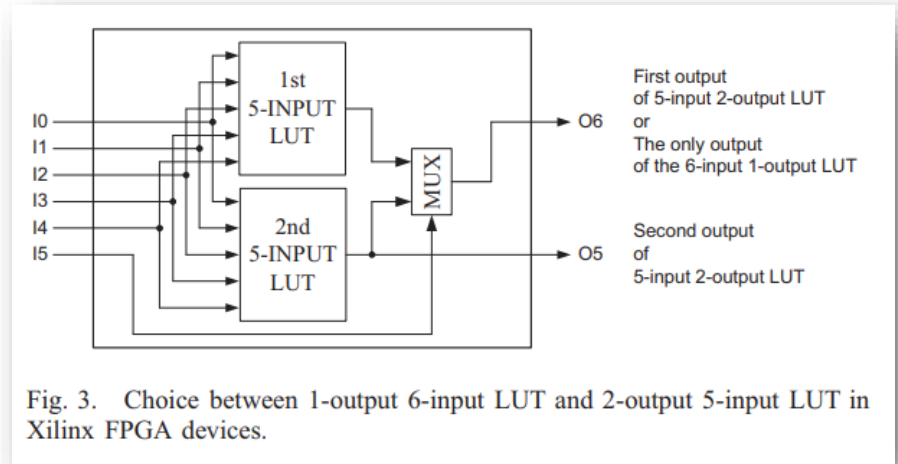


Fig. 3. Choice between 1-output 6-input LUT and 2-output 5-input LUT in Xilinx FPGA devices.



# Xilinx 7 Series FPGA Configuration Logic Block (CLB)

## 7 Series FPGAs Configurable Logic Block

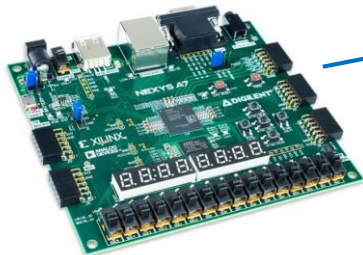
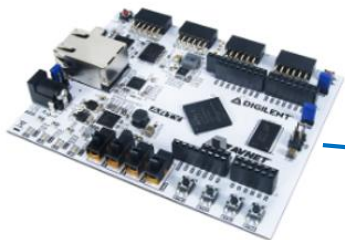
*User Guide*

Slices = SLICEL + SLICEM  
Distributed RAM (bit) = SLICEM \* 256

UG474 (v1.8) September 27, 2016

Table 1-2: Artix-7 FPGA CLB Resources

Device	Slices <sup>(1)</sup>	SLICEL	SLICEM	6-input LUTs	Distributed RAM (Kb)	Shift Register (Kb)	Flip-Flops
7A12T	2,000 <sup>(2)</sup>	1,316	684	8,000	171	86	16,000
7A15T	2,600 <sup>(2)</sup>	1,800	800	10,400	200	100	20,800
7A25T	3,650	2,400	1,250	14,600	313	156	29,200
7A35T	5,200 <sup>(2)</sup>	3,600	1,600	20,800	400	200	41,600
7A50T	8,150	5,750	2,400	32,600	600	300	65,200
7A75T	11,800 <sup>(2)</sup>	8,232	3,568	47,200	892	446	94,400
7A100T	15,850	11,100	4,750	63,400	1,188	594	126,800
7A200T	33,650	22,100	11,550	134,600	2,888	1,444	269,200

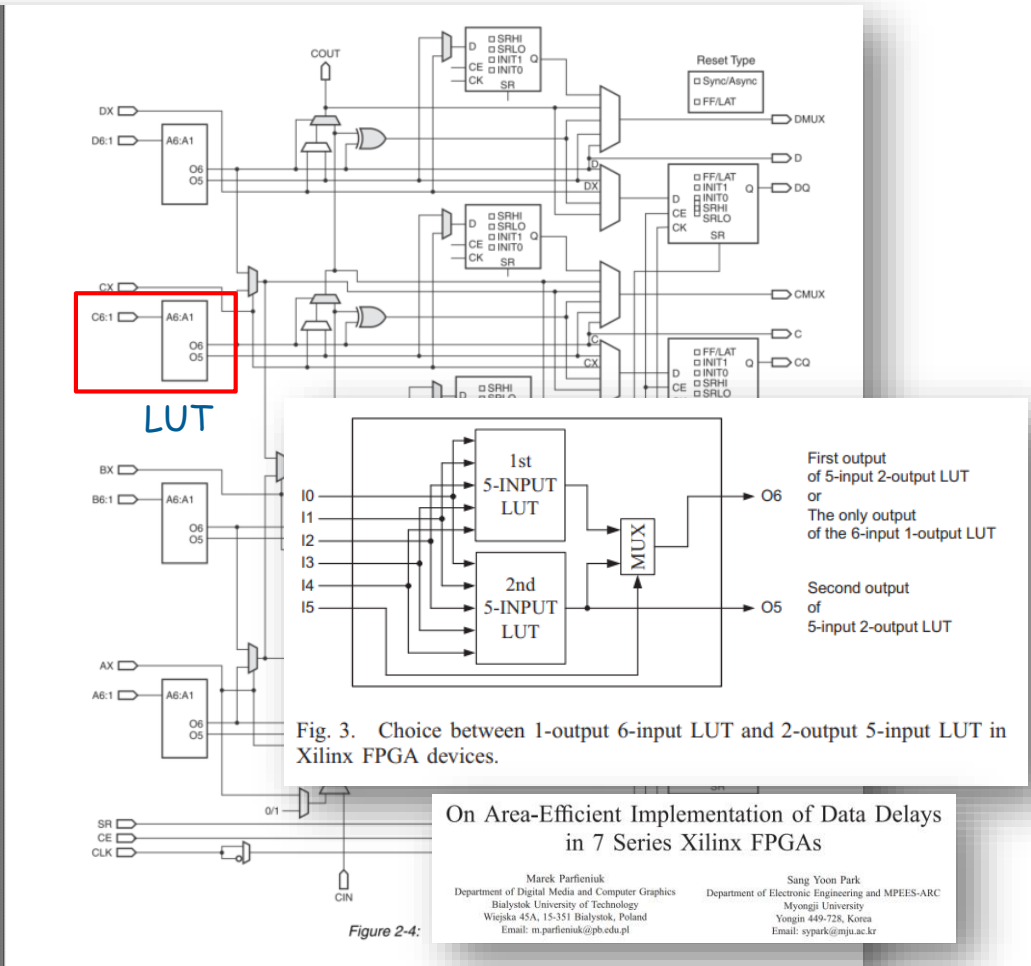
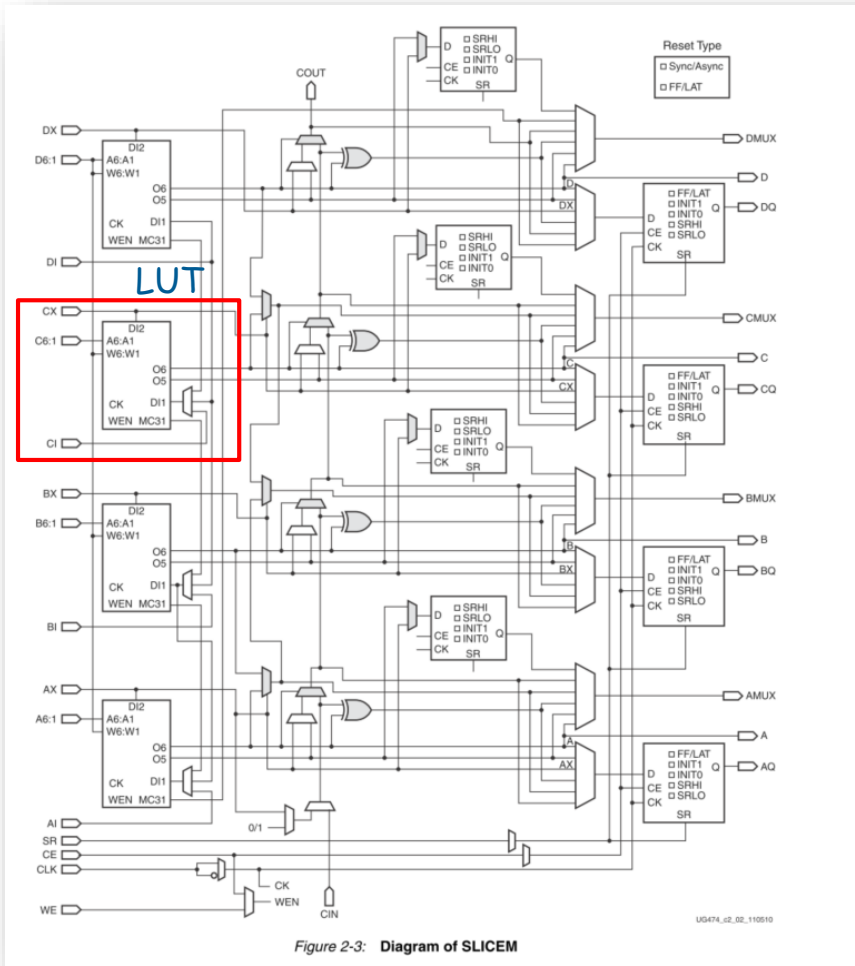


# Xilinx 7 Series Configuration Logic Block (CLB)



SLICEM

SLICEL



**On Area-Efficient Implementation of Data Delays in 7 Series Xilinx FPGAs**

Marek Parfieniuk  
Department of Digital Media and Computer Graphics  
Białystok University of Technology  
Wajtko 45A, 15-351 Białystok, Poland  
Email: m.parfieniuk@pb.edu.pl

Sang Yoon Park  
Department of Electronic Engineering and MPEES-ARC  
Myongji University  
Yongin 449-728, Korea  
Email: sypark@mju.ac.kr



# Distributed RAM

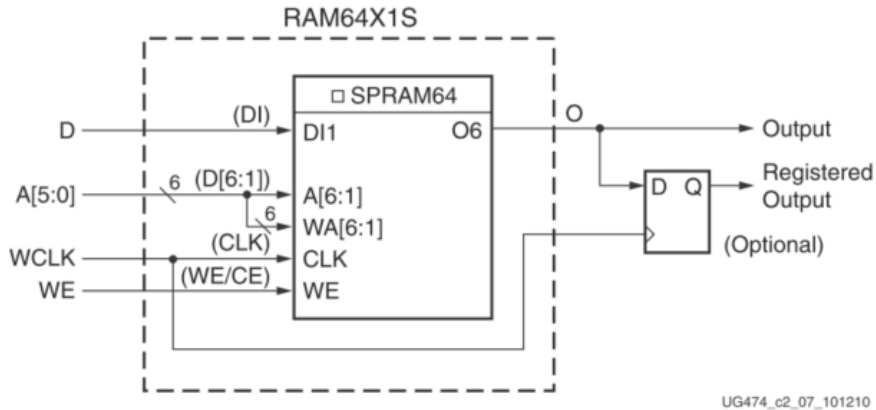


Figure 2-8: 64 X 1 Single Port Distributed RAM (RAM64X1S)

```

module m_RAM64X1S (clk, a, d, we, dout);
input wire clk;
input wire [5:0] a;
input wire d, we;
output wire dout;

reg [0:0] mem [0:63];
assign dout = mem[a];
always @(posedge clk) if(we) mem[a] <= d;
endmodule
    
```

LUTRAM = 1

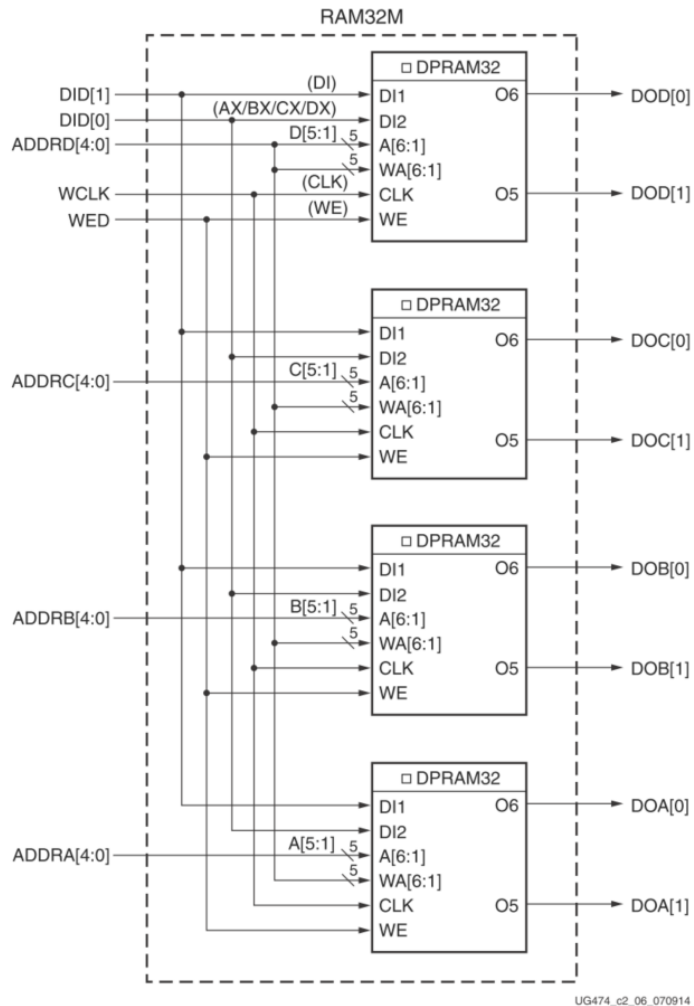
Table 2-3: Distributed RAM Configuration

RAM	Description	Primitive	Number of LUTs
32 x 1S	Single port	RAM32X1S	1
32 x 1D	Dual port	RAM32X1D	2
32 x 2Q	Quad port	RAM32M	4
32 x 6SDP	Simple dual port	RAM32M	4
64 x 1S	Single port	RAM64X1S	1
64 x 1D	Dual port	RAM64X1D	2
64 x 1Q	Quad port	RAM64M	4
64 x 3SDP	Simple dual port	RAM64M	4
128 x 1S	Single port	RAM128X1S	2
128 x 1D	Dual port	RAM128X1D	4
256 x 1S	Single port	RAM256X1S	4

- Single port
  - Common address port for synchronous writes and asynchronous reads
    - Read and write addresses share the same address bus



# Distributed RAM



UG474\_c2\_06\_070914

Figure 2-6: 32 X 2 Quad Port Distributed RAM (RAM32M)

Table 2-3: Distributed RAM Configuration

RAM	Description	Primitive	Number of LUTs
32 x 1S	Single port	RAM32X1S	1
32 x 1D	Dual port	RAM32X1D	2
32 x 2Q	Quad port	RAM32M	4
32 x 6SDP	Simple dual port	RAM32M	4
64 x 1S	Single port	RAM64X1S	1
64 x 1D	Dual port	RAM64X1D	2
64 x 1Q	Quad port	RAM64M	4
64 x 3SDP	Simple dual port	RAM64M	4
128 x 1S	Single port	RAM128X1S	2
128 x 1D	Dual port	RAM128X1D	4
256 x 1S	Single port	RAM256X1S	4

- Quad port
  - One port for synchronous writes and asynchronous reads
  - Three ports for asynchronous reads





# Distributed RAM

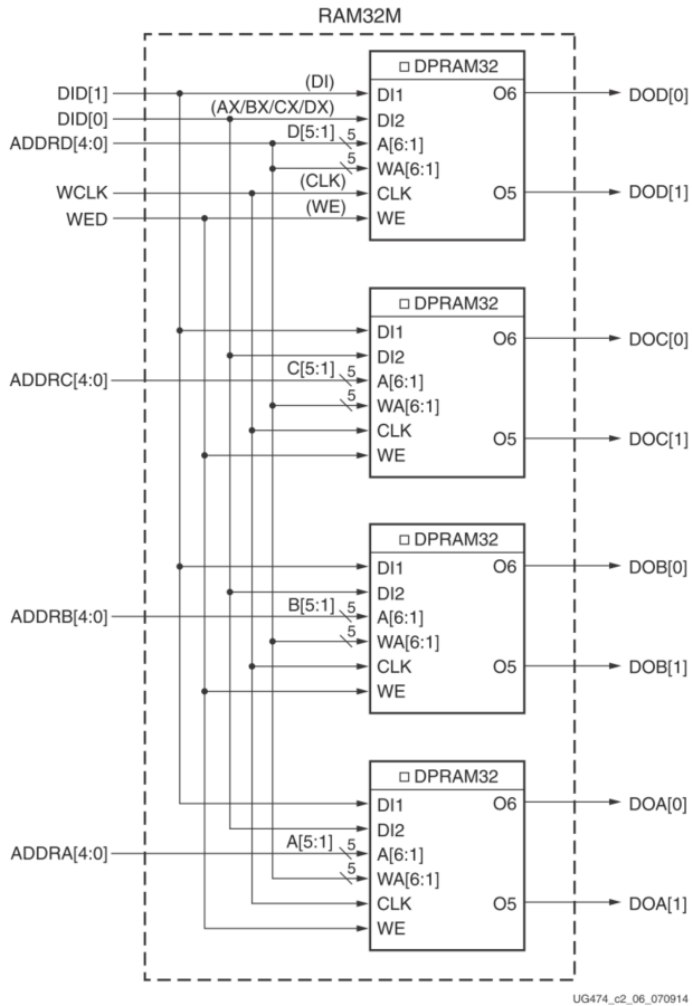


Figure 2-6: 32 X 2 Quad Port Distributed RAM (RAM32M)

```

module m_RAM32M_Q (clk, a1, a2, a3, a4, d, we, dout1, dout2, dout3, dout4);
  input wire clk;
  input wire [4:0] a1, a2, a3, a4;
  input wire [1:0] d;
  input wire we;
  output wire [1:0] dout1, dout2, dout3, dout4;

  reg [1:0] mem [0:31];
  assign dout1 = mem[a1];
  assign dout2 = mem[a2];
  assign dout3 = mem[a3];
  assign dout4 = mem[a4];
  always @(posedge clk) if(we) mem[a1] <= d;
endmodule
    
```

Failed Routes	LUT	FF	BRAM	URAM	DSP
	4	0	0.0	0	0
	0	4	0	0.0	0

LUTRAM = 4

