Greetings

- Who is this...
  - David Thomas
  - *Currently*: Dept. of Electrical Engineering
  - *Previously*: MEng in Soft. Eng. in DoC
  - PhD student of Wayne Luk (*a long time ago*...)
- My interests
  - High-Performance Computing
  - FPGA and GPU acceleration
  - High-level Synthesis (C to gates)
  - Languages and parallelism

Course administrivia

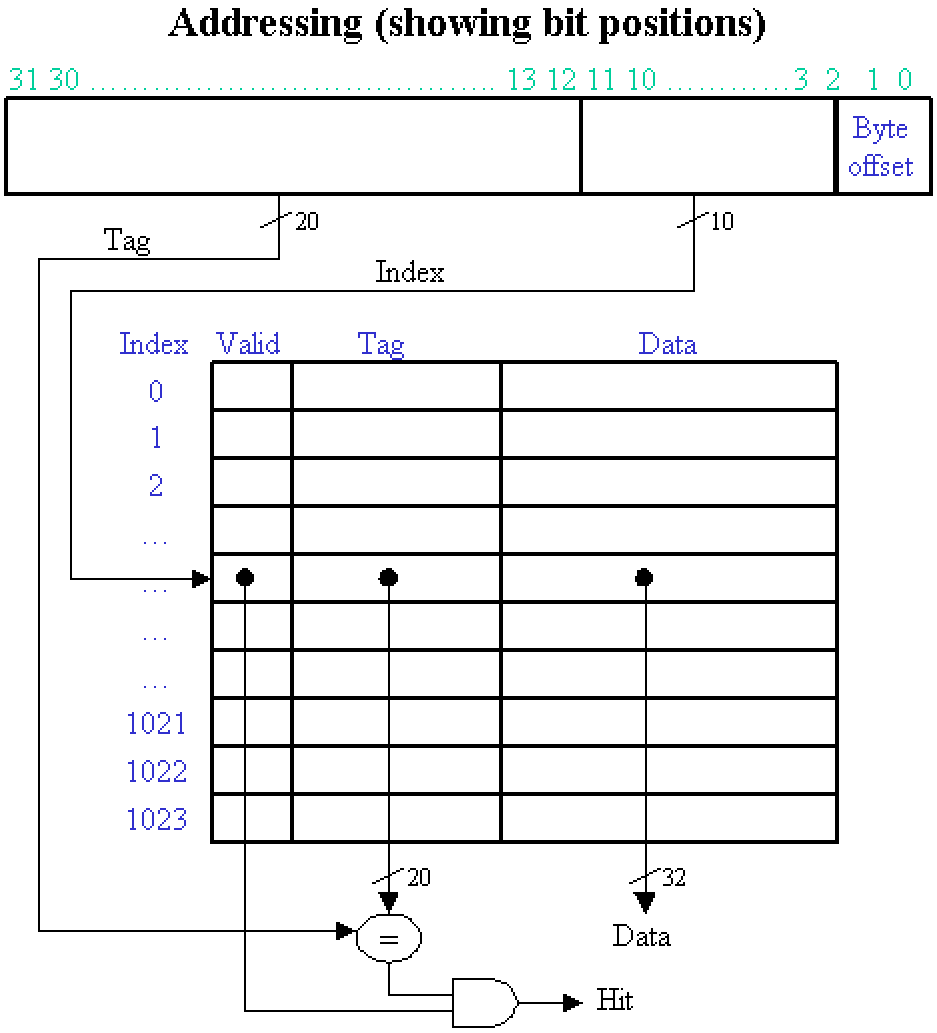
- Panopto will continue
  - Assuming the hardware and servers work
- If you post on piazza I'll eventually see it
  - Traditionally it is a ghost-town till exams
- I tend to do tutorials differently to Prof. Luk
  - Standard exercises + solutions still available
- Wed 1st of March: *No lecture*
  - Course is one lecture less than no. of lecture slots
  - Tutorial available as normal

Cache features and performance

(3rd Ed: p.492-496,505-511; 4th Ed: 475-479,487-492)

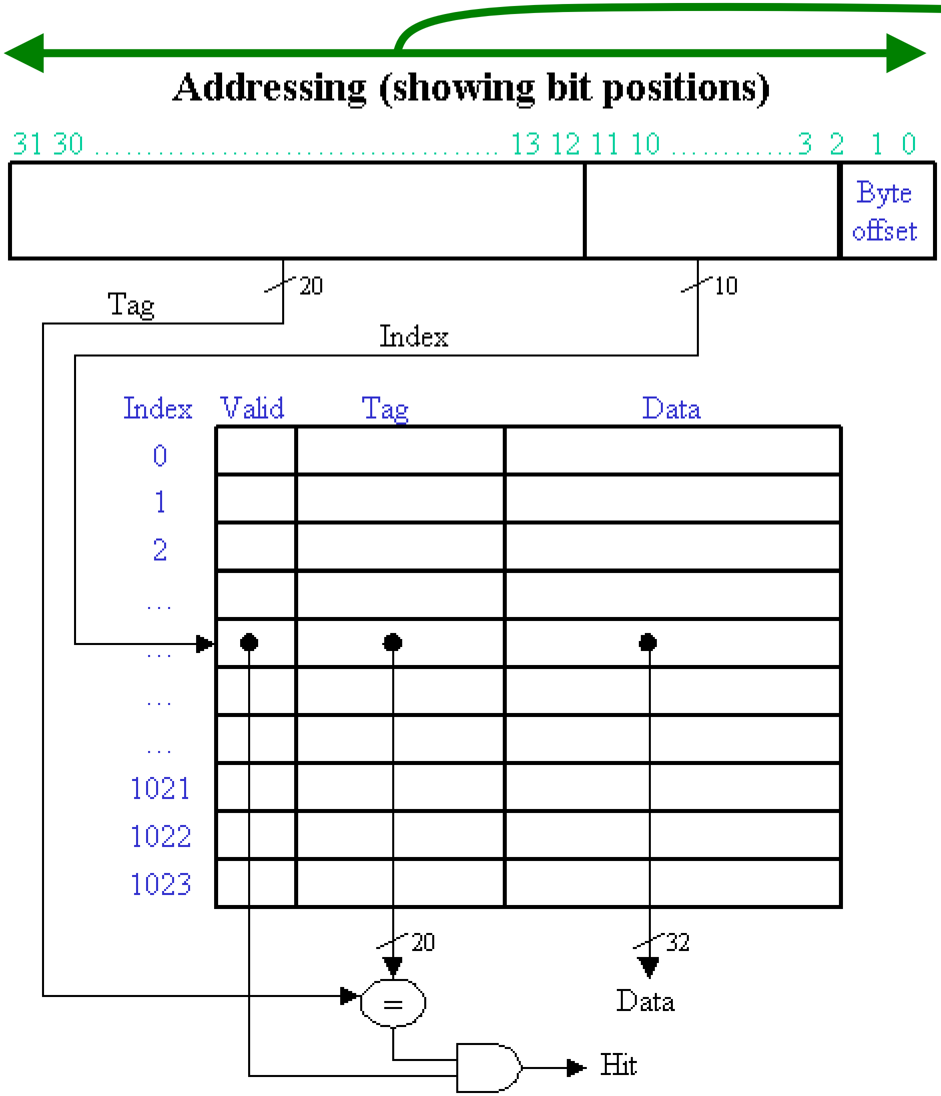
- so far:
  - locality principles
  - levels, blocks, hit, miss, miss penalty
  - direct-mapped cache, handling misses
- today:
  - review direct-mapped single- and multi-word cache
  - cache performance, read/write stall cycles
  - multi-level cache hierarchy

Direct-mapped single-word cache



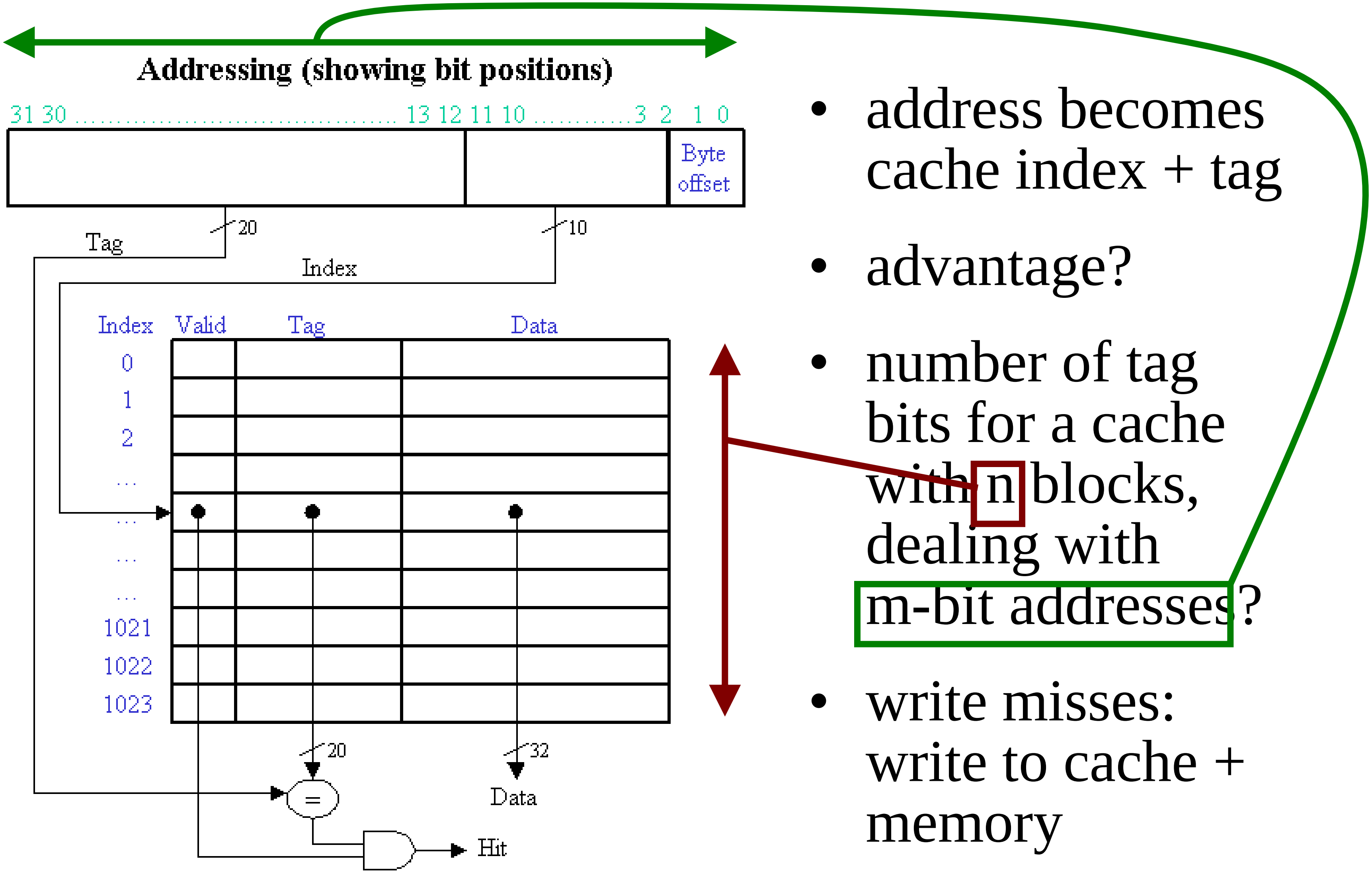
- address becomes cache index + tag
- advantage?
- number of tag bits for a cache with n blocks, dealing with m-bit addresses?
- write misses: write to cache + memory

Direct-mapped single-word cache



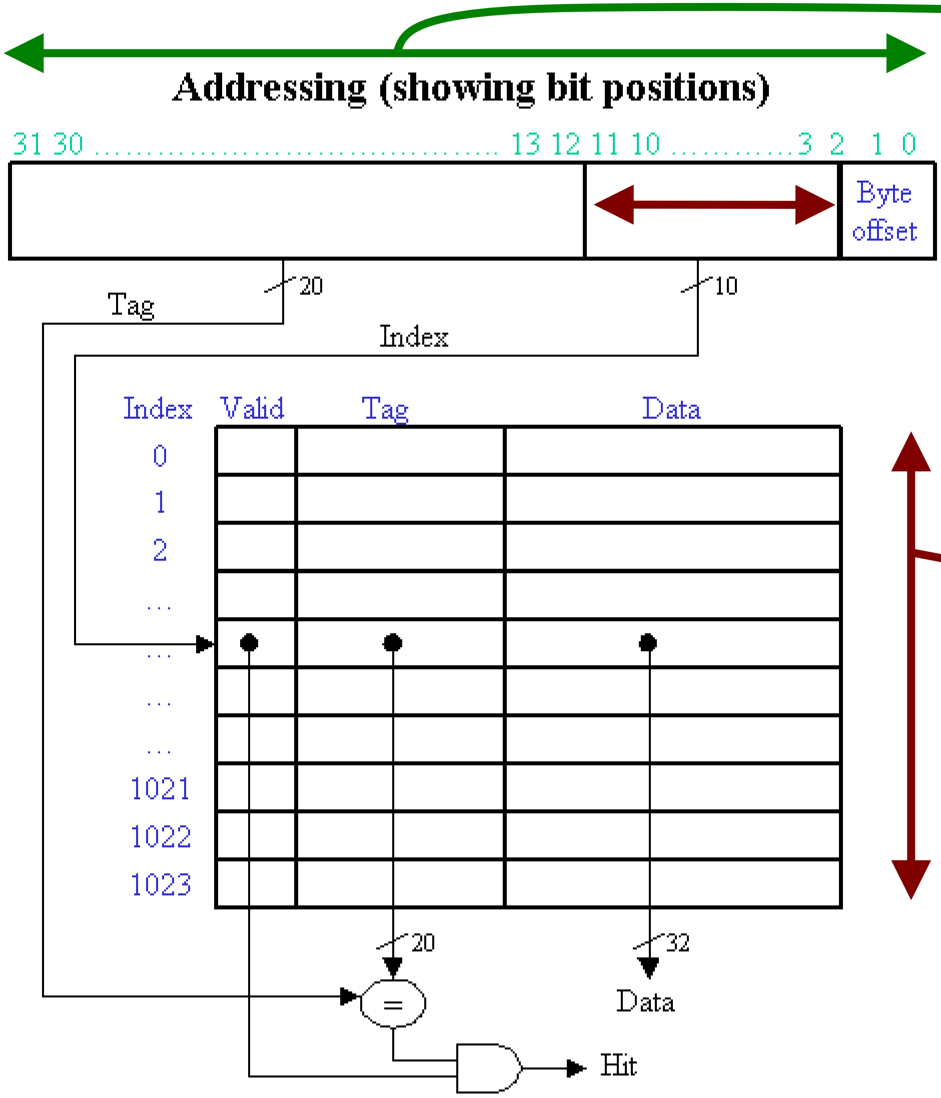
- address becomes cache index + tag
- advantage?
- number of tag bits for a cache with n blocks, dealing with m-bit addresses?
- write misses: write to cache + memory

Direct-mapped single-word cache



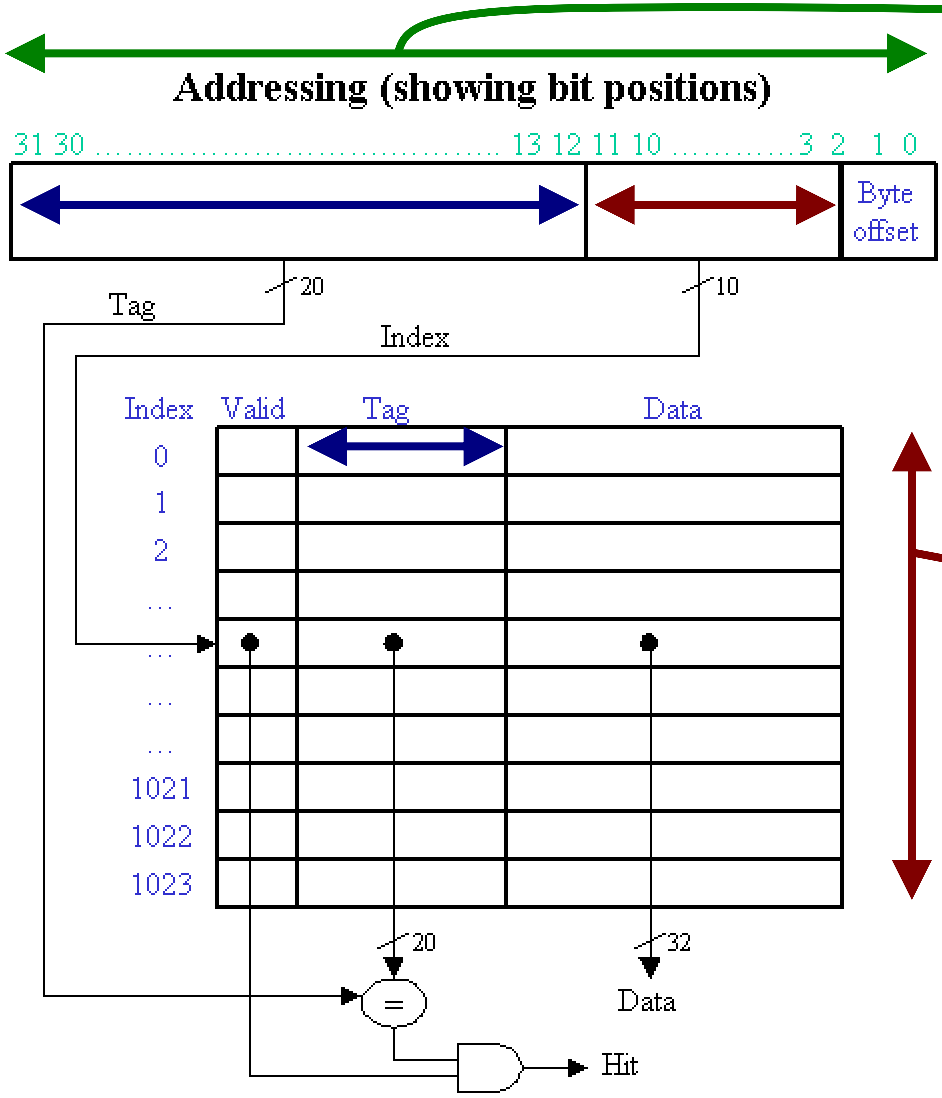
- address becomes cache index + tag
- advantage?
- number of tag bits for a cache with n blocks, dealing with m-bit addresses?
- write misses: write to cache + memory

Direct-mapped single-word cache



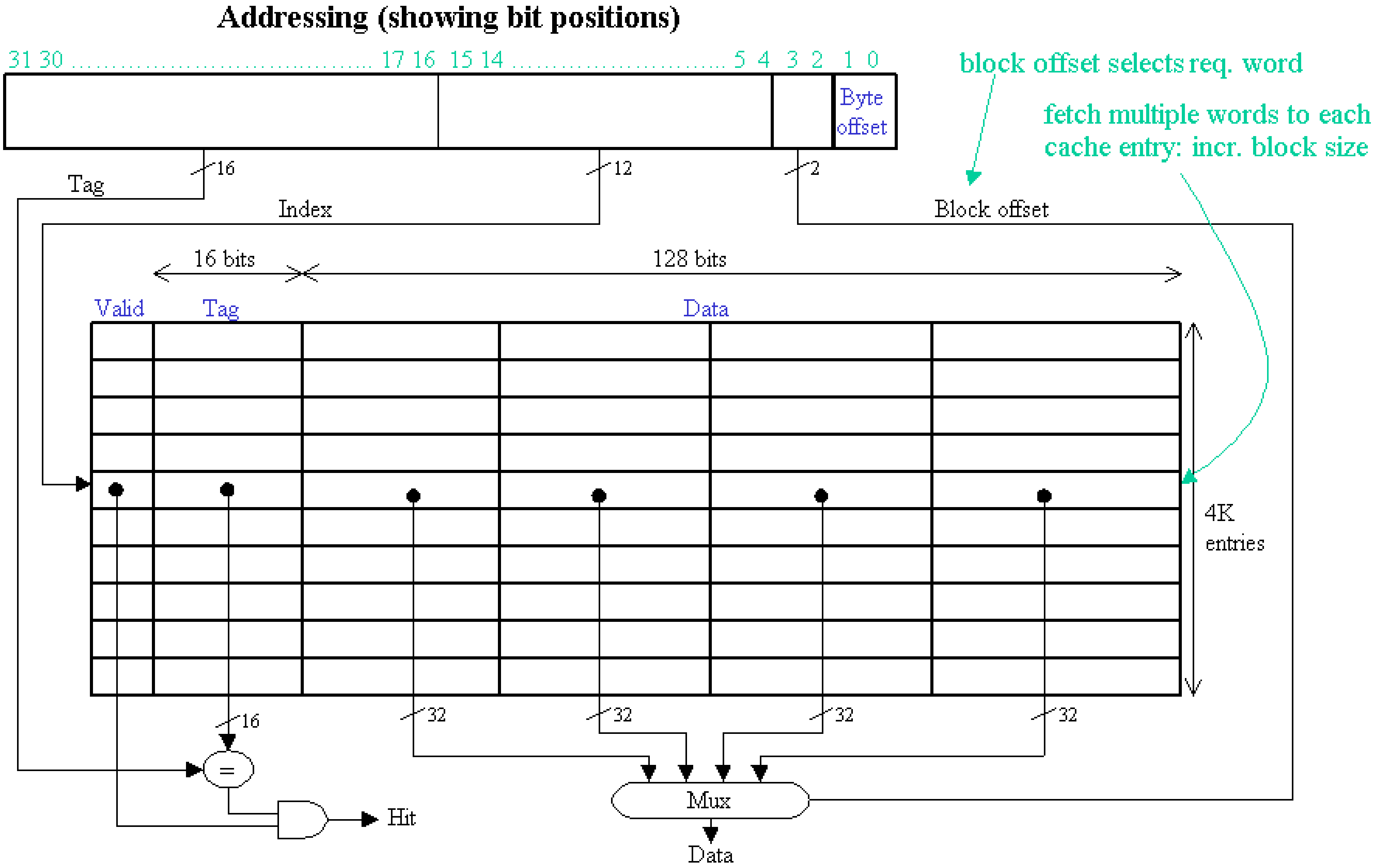
- address becomes cache index + tag
- advantage?
- number of tag bits for a cache with n blocks, dealing with m-bit addresses?
- write misses: write to cache + memory

Direct-mapped single-word cache



- address becomes cache index + tag
- advantage?
- number of tag bits for a cache with n blocks, dealing with m-bit addresses?
- write misses: write to cache + memory

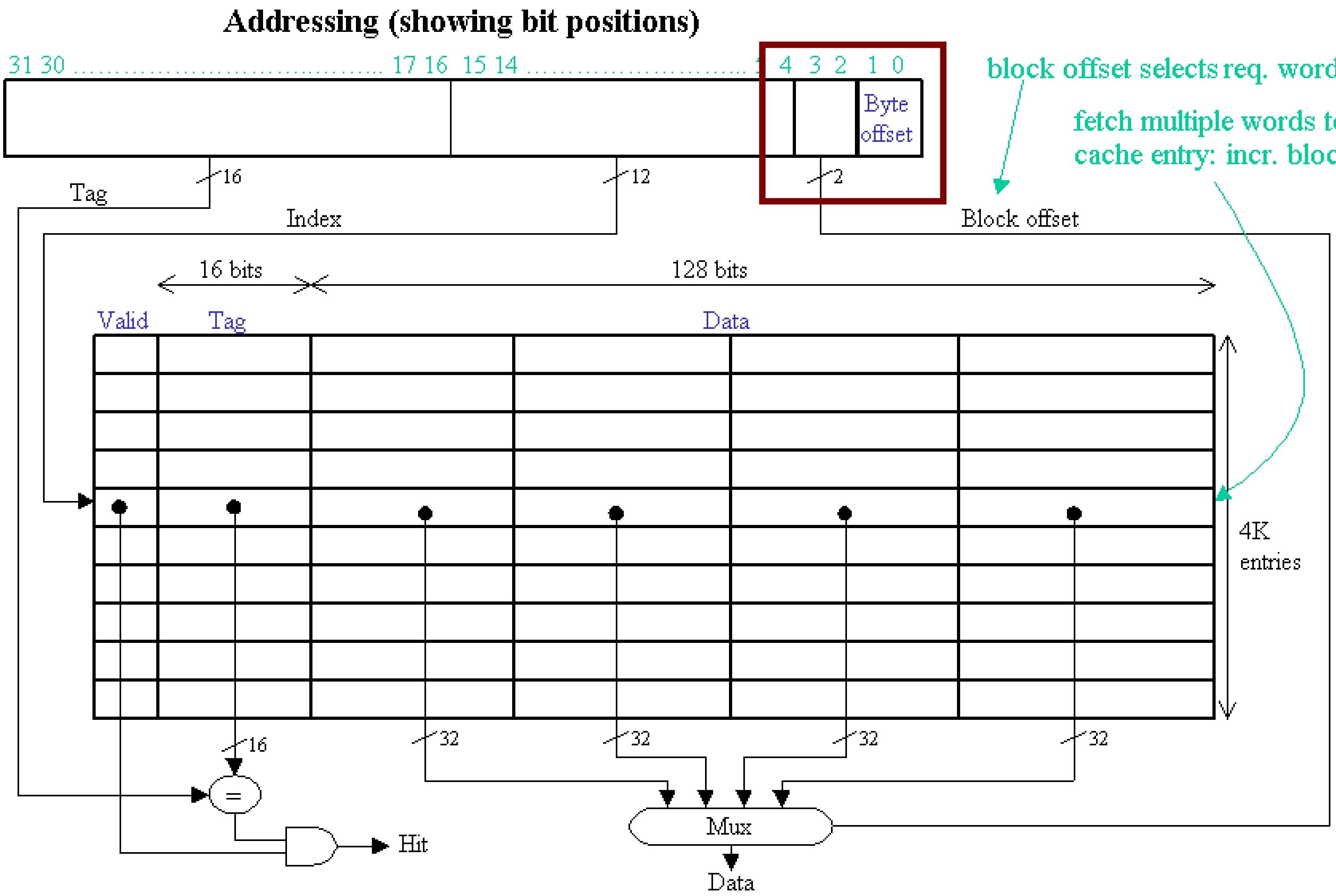
Multi-word cache



block offset selects req. word

fetch multiple words to each cache entry: incr. block size

Multi-word cache



block offset selects req. word

fetch multiple words to each cache entry: incr. block size

Direct-mapped caches

- byte: smallest addressable unit of memory
- word size : *w* >= 1, *w* in bytes
  - natural granularity of CPU; size of registers
- block size : *k* >= 1, *k* in words
  - granularity of cache to memory transfers
- cache size : *c* >= *k*, *c* in words
  - total size of cache storage; number of blocks is (`c` **`div`** `k`)

Direct-mapped caches

- byte: smallest addressable unit of memory
- word size : w >= 1, w in bytes
  - natural granularity of CPU; size of registers
- block size : k >= 1, k in words
  - granularity of cache to memory transfers
- cache size : c >= k, c in words
  - total size of cache storage; number of blocks is (`c div k`)
- Lookup `byte_address` in a direct-mapped cache

```
1. word_address  = (byte_address div w);
2. block_address = (word_address div k);
3. block_index   = (block_address mod (c div k) );
```

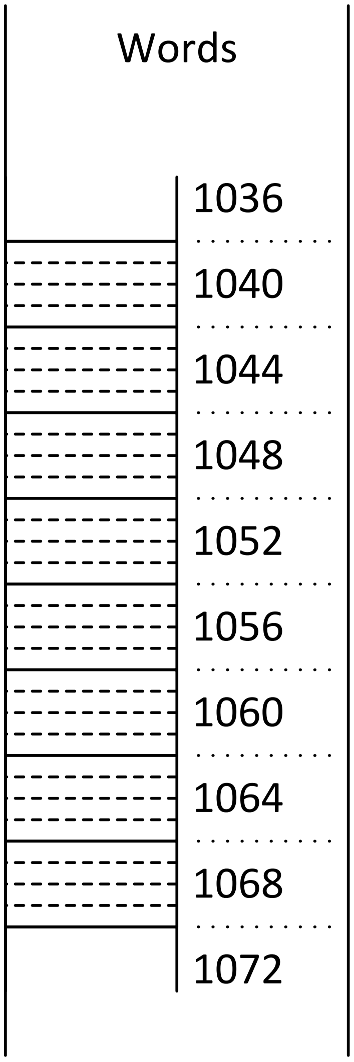
Direct-mapped caches

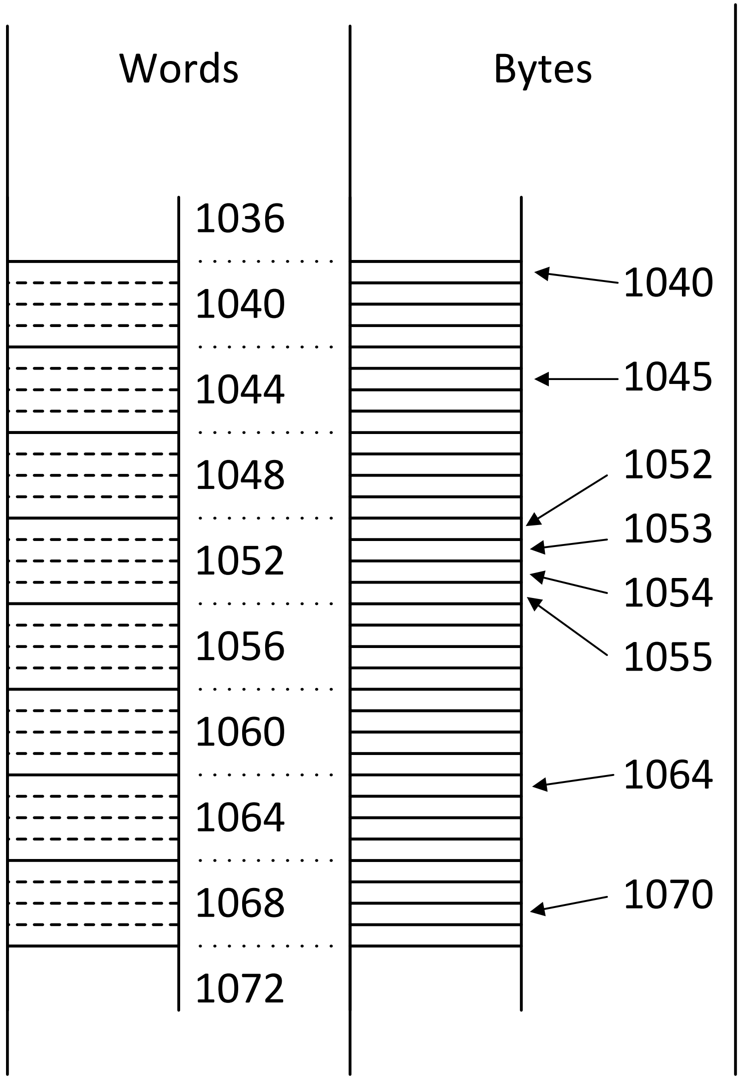
Invariant : 0 <= `block_index` < (`c` **`div`** `k`)

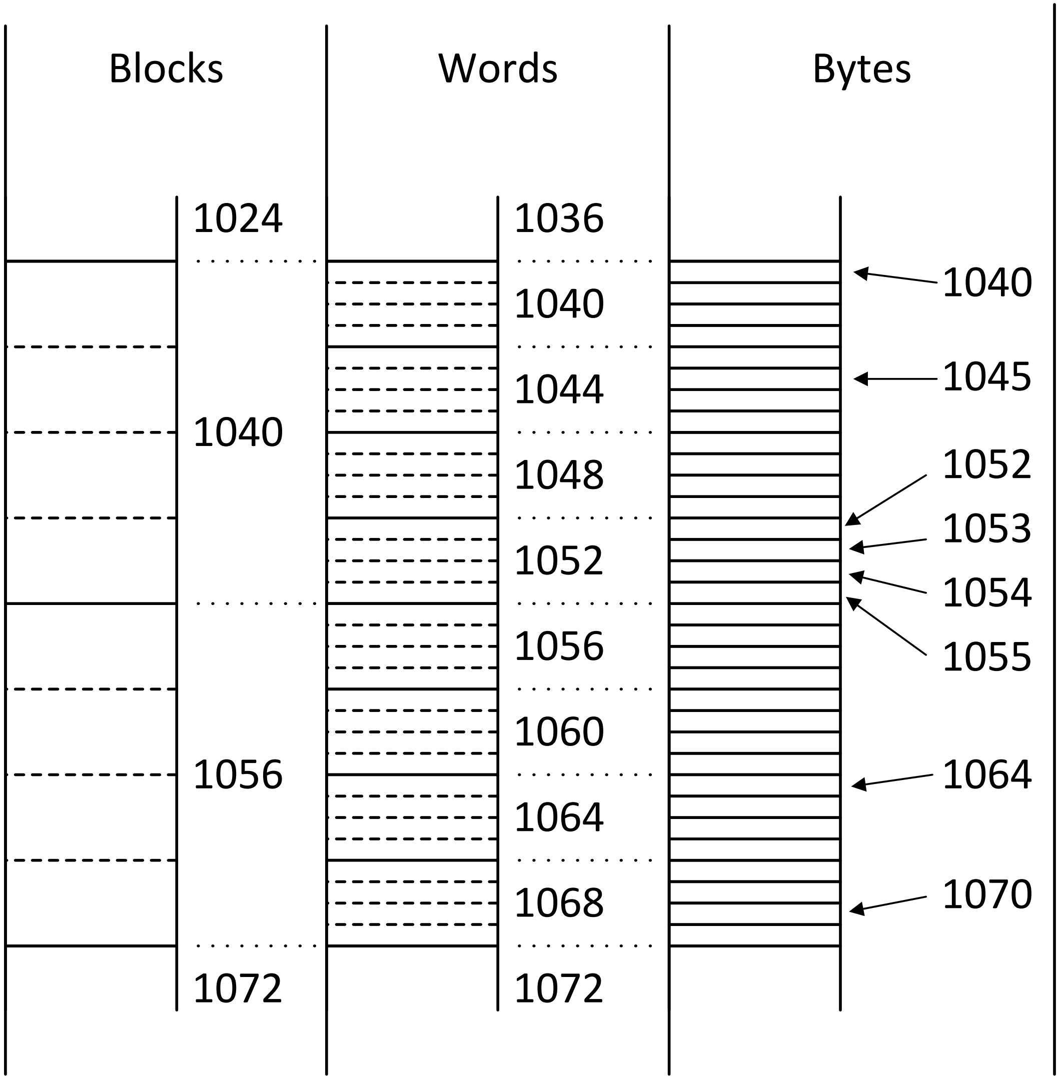
`block_index` specifies the ***only*** cache-block where `byte_address` or `word_address` can be found

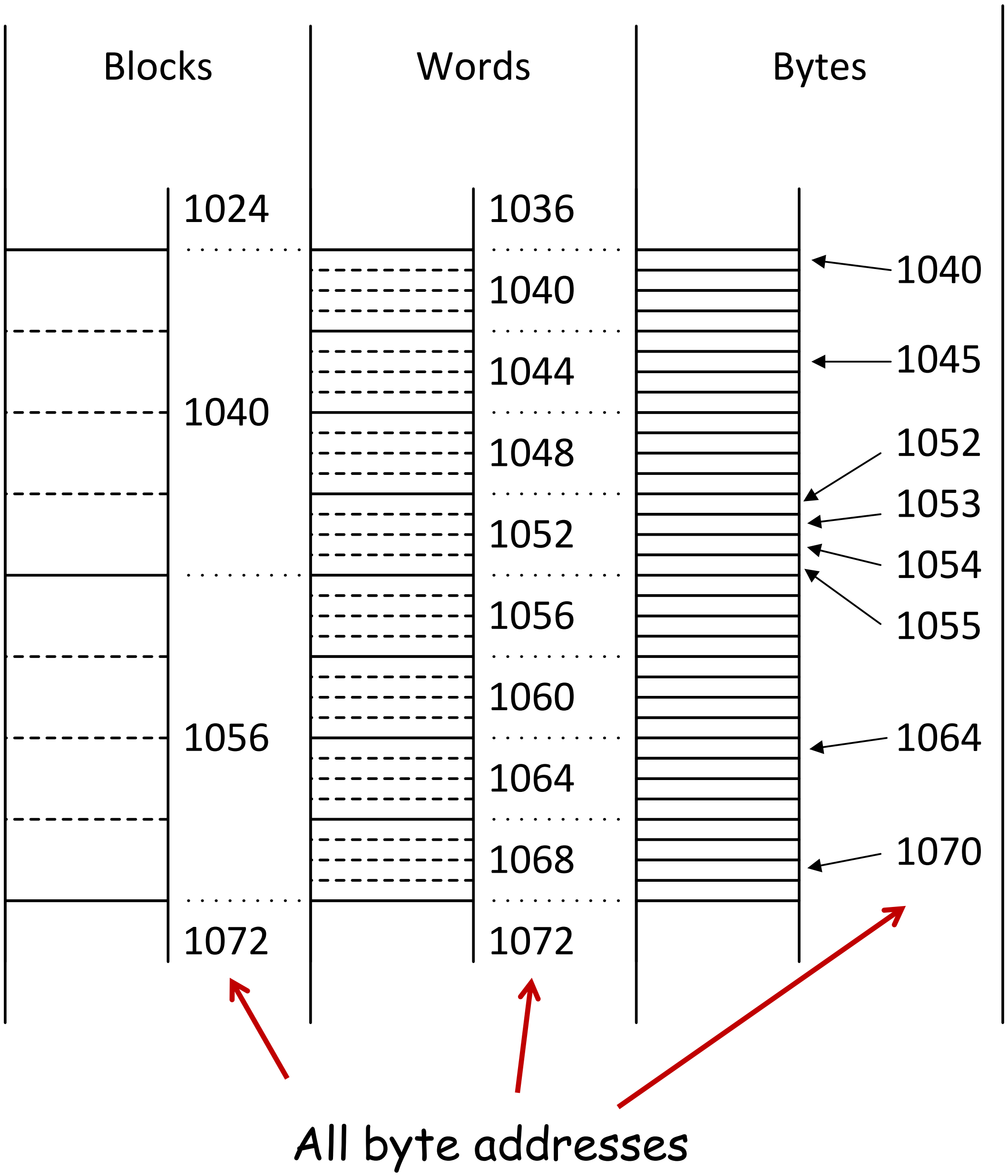
- cache size : *c* >= *k*, *c* in words
  - total size of cache storage; number of blocks is (`c` **`div`** `k`)
- Lookup `byte_address` in a direct-mapped cache

```
1.  word_address  = (byte_address div w);
2.  block_address = (word_address div k);
3.  block_index   = (block_address mod (c div k) );
```

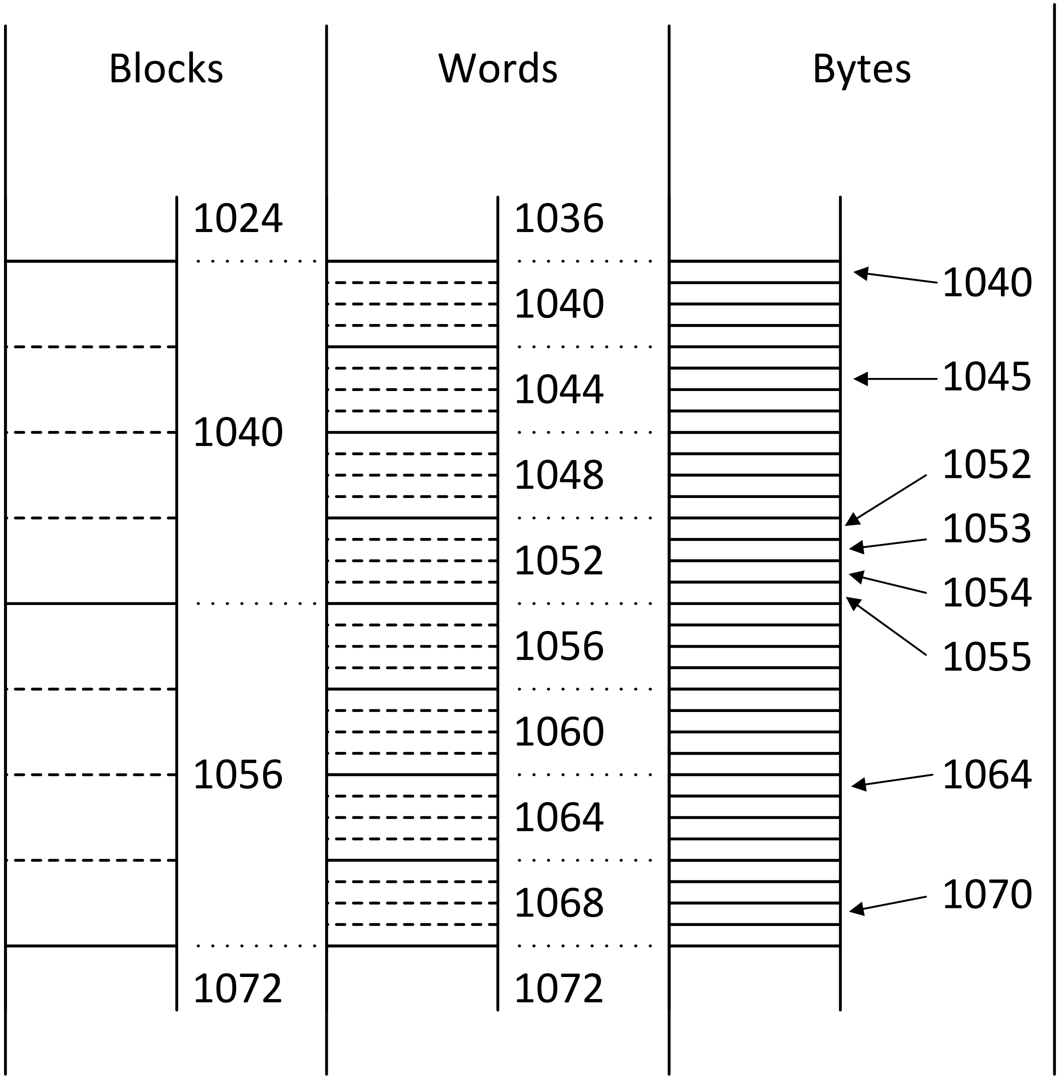

Words
1036
1040
1044
1048
1052
1056
1060
1064
1068
1072


Words
Bytes
1036
1040
1044
1048
1052
1056
1060
1064
1068
1072
1040
1045
1052
1053
1054
1055
1064
1070

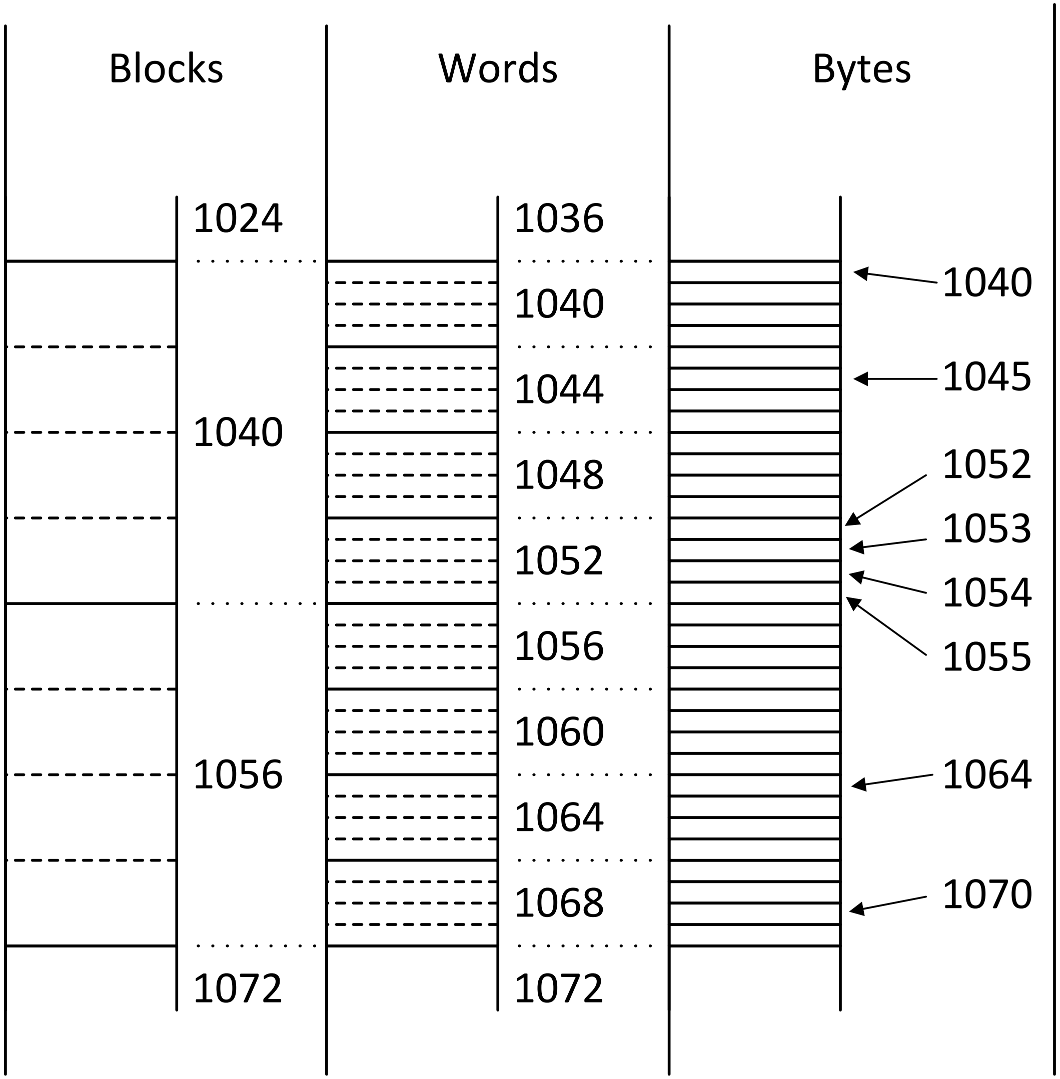

Blocks
Words
Bytes
1024
1040
1056
1072
1036
1040
1044
1048
1052
1056
1060
1064
1068
1072
1040
1045
1052
1053
1054
1055
1064
1070


Blocks
Words
Bytes
1024
1040
1056
1072
1036
1040
1044
1048
1052
1056
1060
1064
1068
1072
1040
1045
1052
1053
1054
1055
1064
1070
All byte addresses

```
addi $3, $0, 1044

lw   $4, 4($3)

addi $3, $3, 11

lb   $6, -4($3)

lw   $6, 0($3)
```



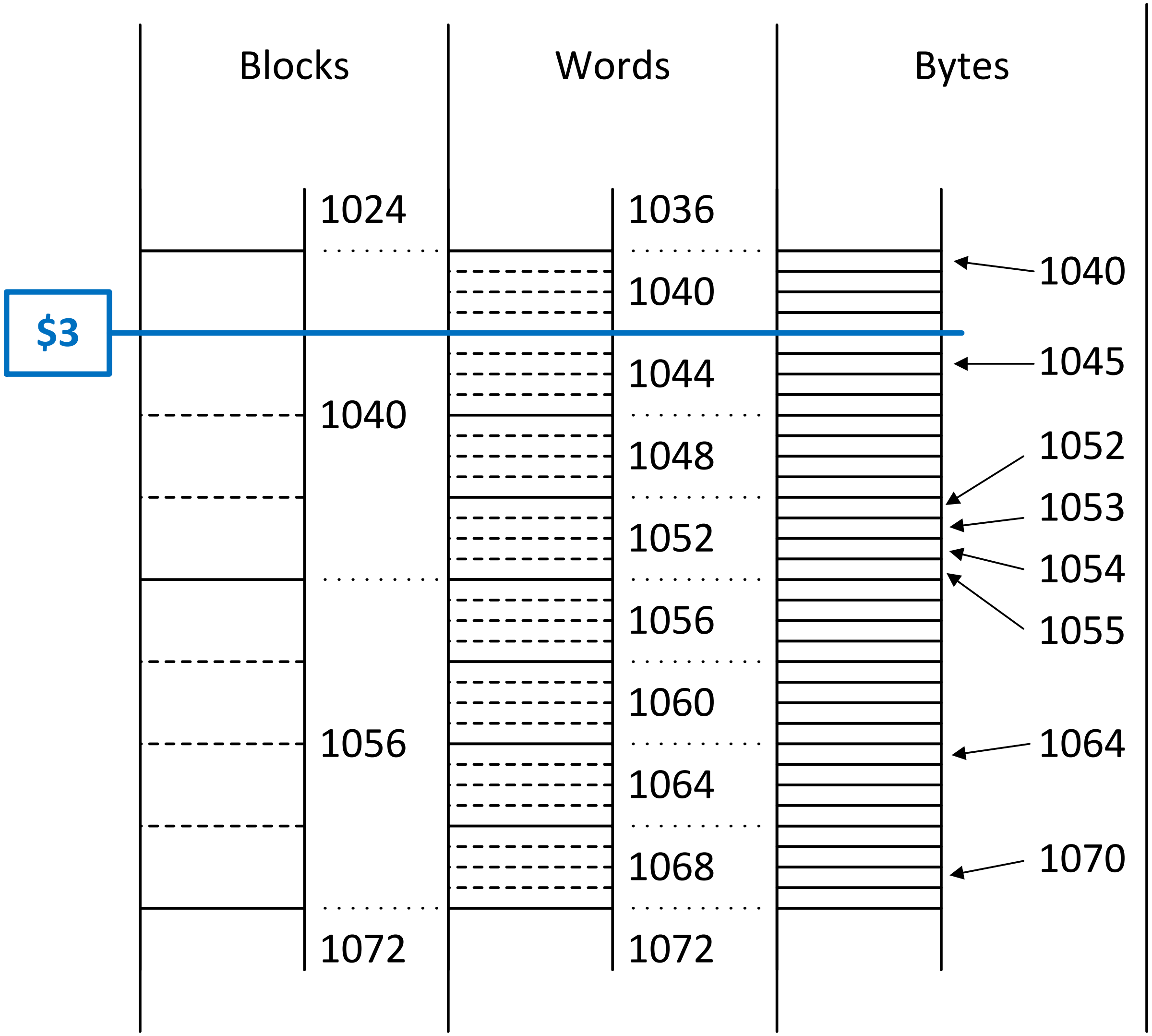
```
addi $3, $0, 1044

lw   $4, 4($3)

addi $3, $3, 11

lb   $6, -4($3)

lw   $6, 0($3)
```



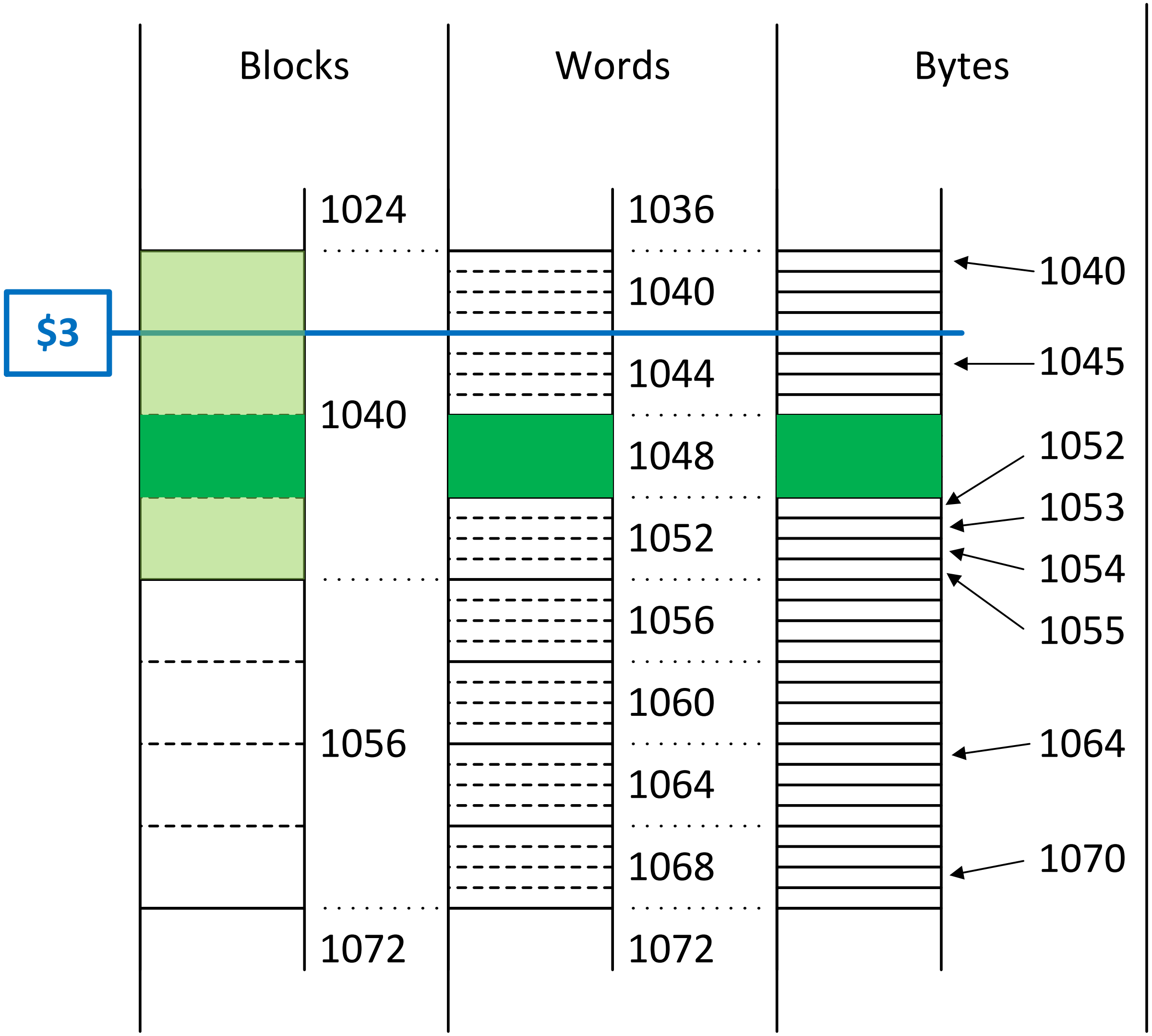
lw : load word
lb : load byte
addi : add immediate

lw $dest offset($base)

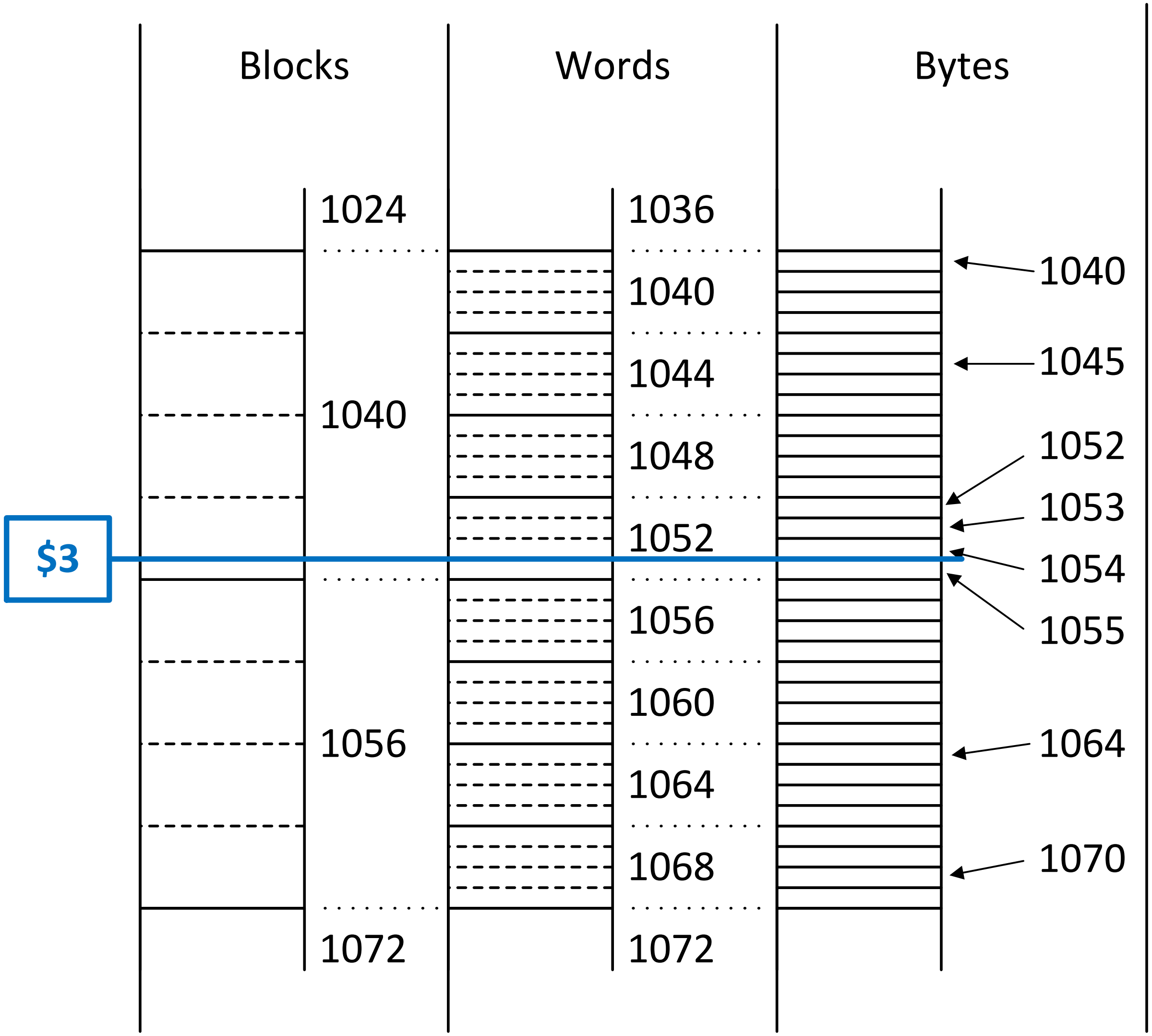
```
addi $3, $0, 1044

lw   $4, 4($3)

addi $3, $3, 11

lb   $6, -4($3)

lw   $6, 0($3)
```



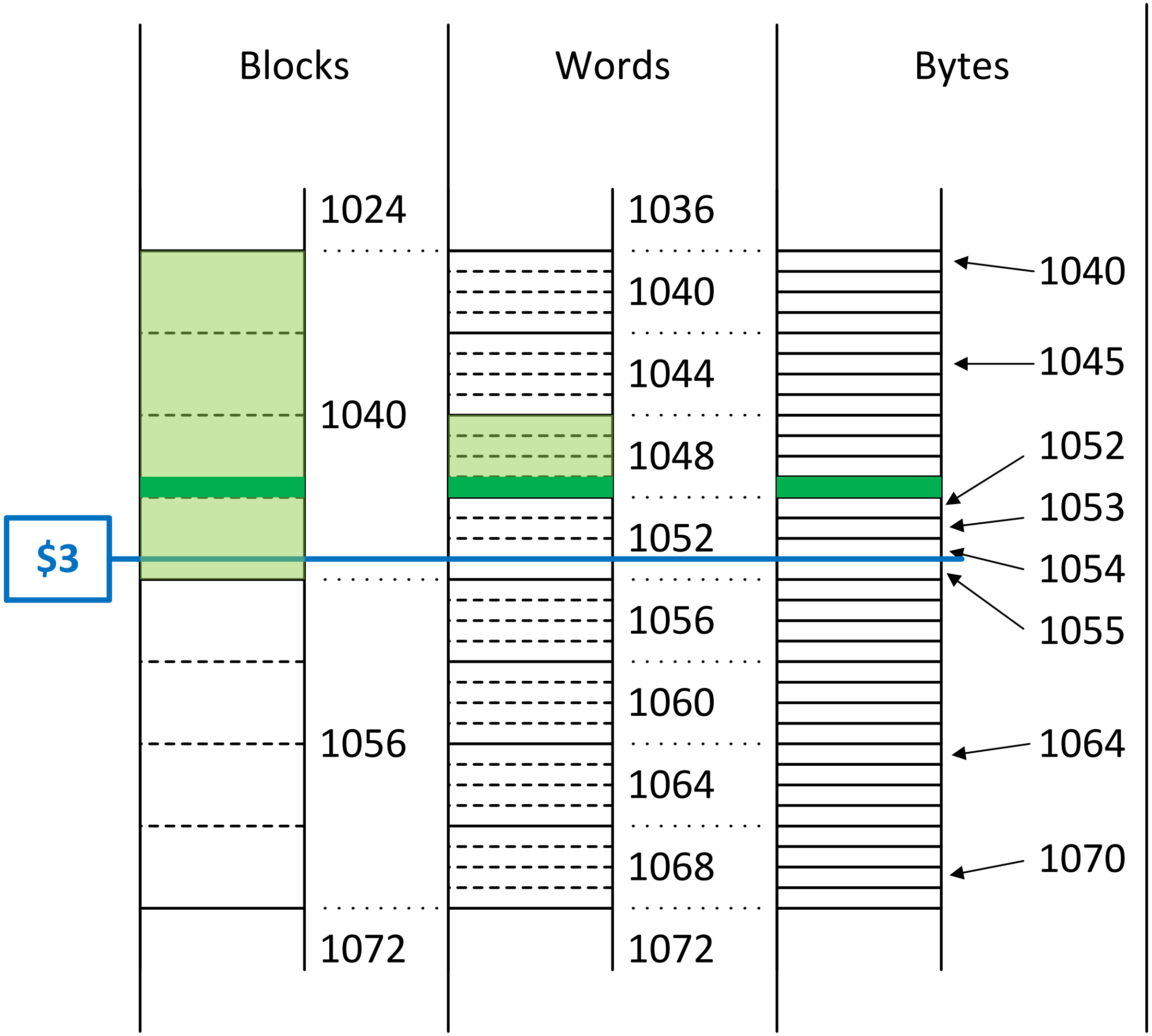
```
addi $3, $0, 1044

lw   $4, 4($3)

addi $3, $3, 11

lb   $6, -4($3)

lw   $6, 0($3)
```



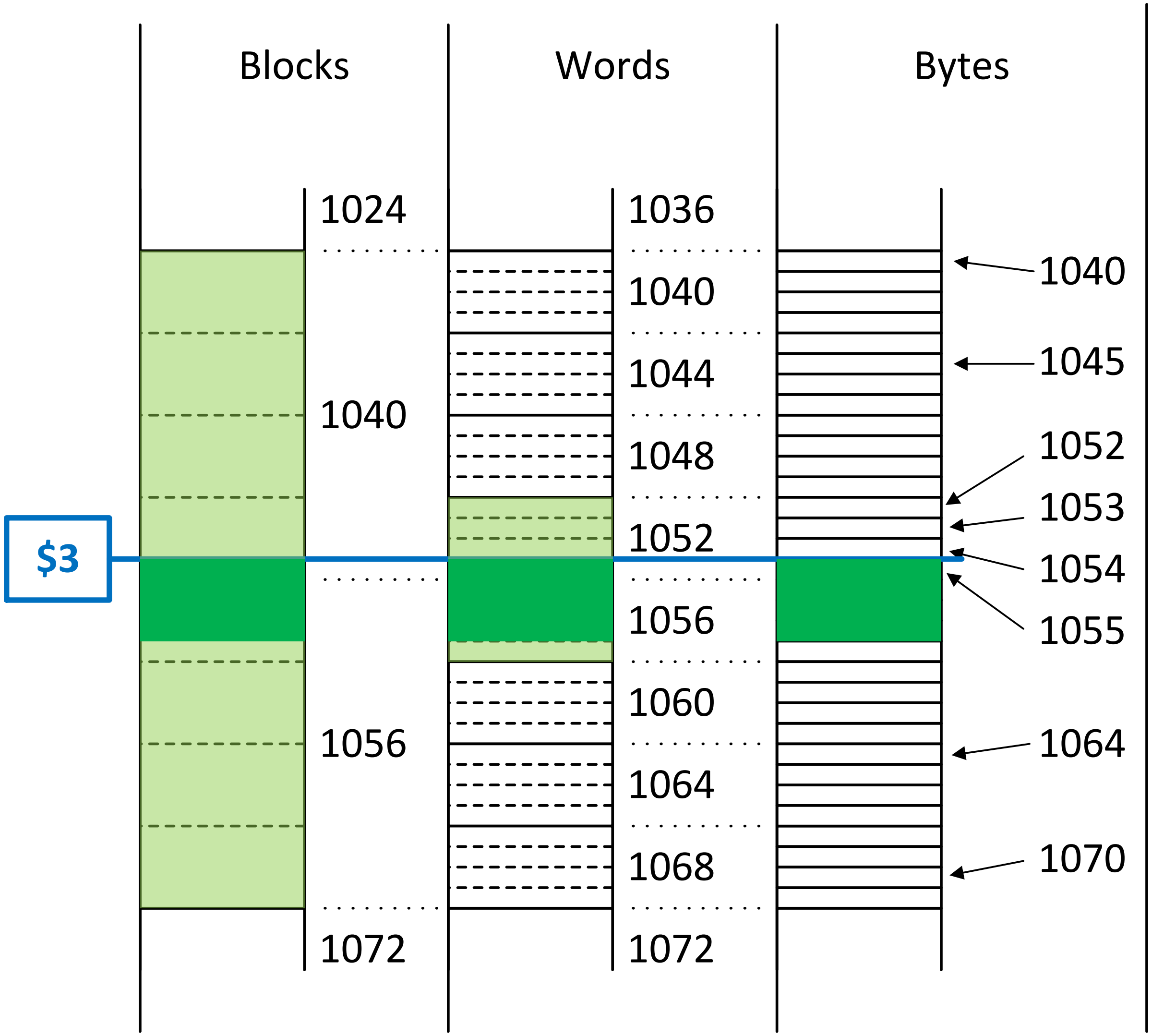
```
addi $3, $0, 1044

lw   $4, 4($3)

addi $3, $3, 11

lb   $6, -4($3)

lw   $6, 0($3)
```



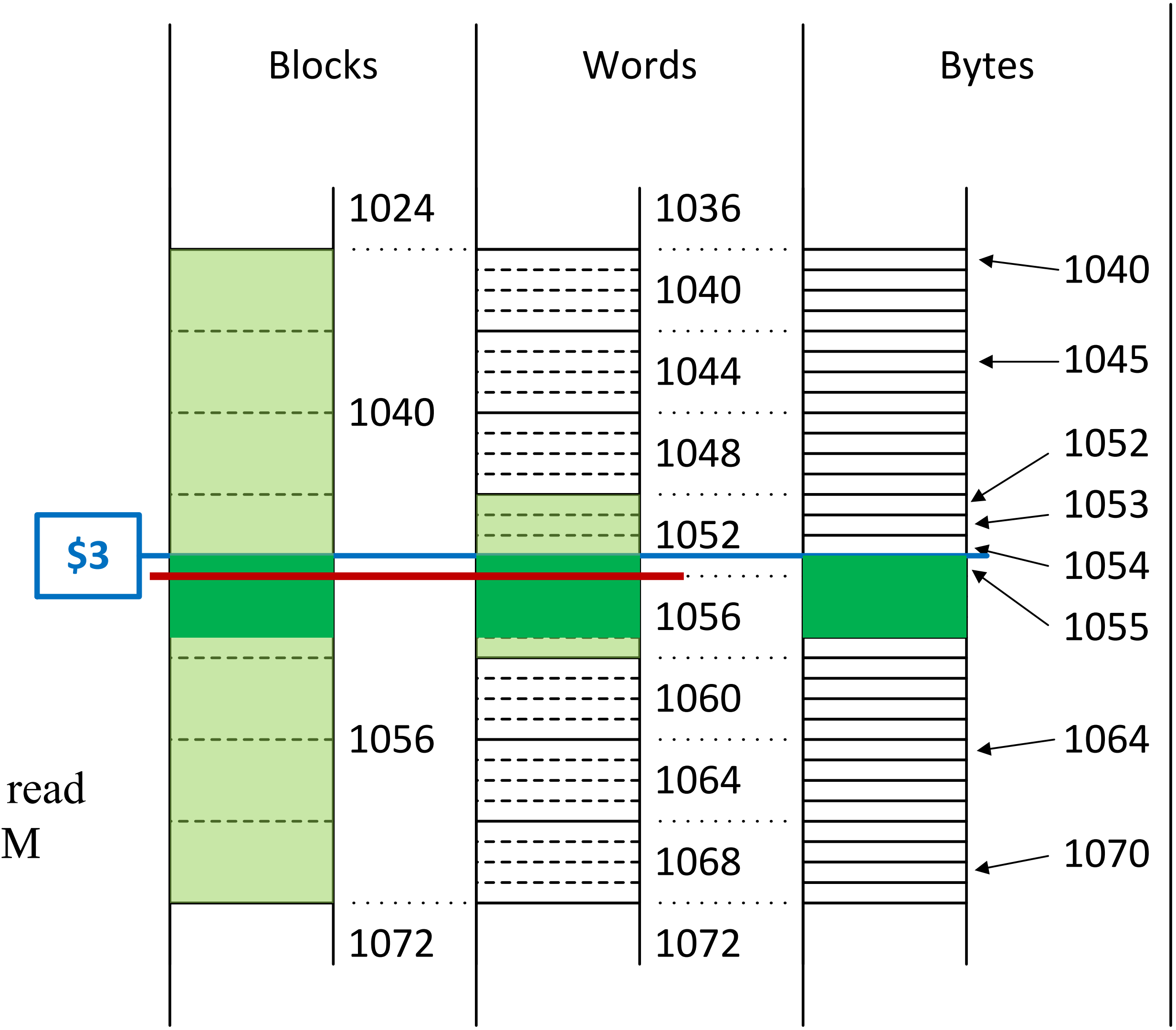
```
addi $3, $0, 1044

lw   $4, 4($3)

addi $3, $3, 1

lb   $6, -4($3)

lw   $6, 0($3)
```



```
addi $3, $0, 1044

lw   $4, 4($3)

addi $3, $3, 1

lb   $6, -4($3)

lw   $6, 0($3)
```



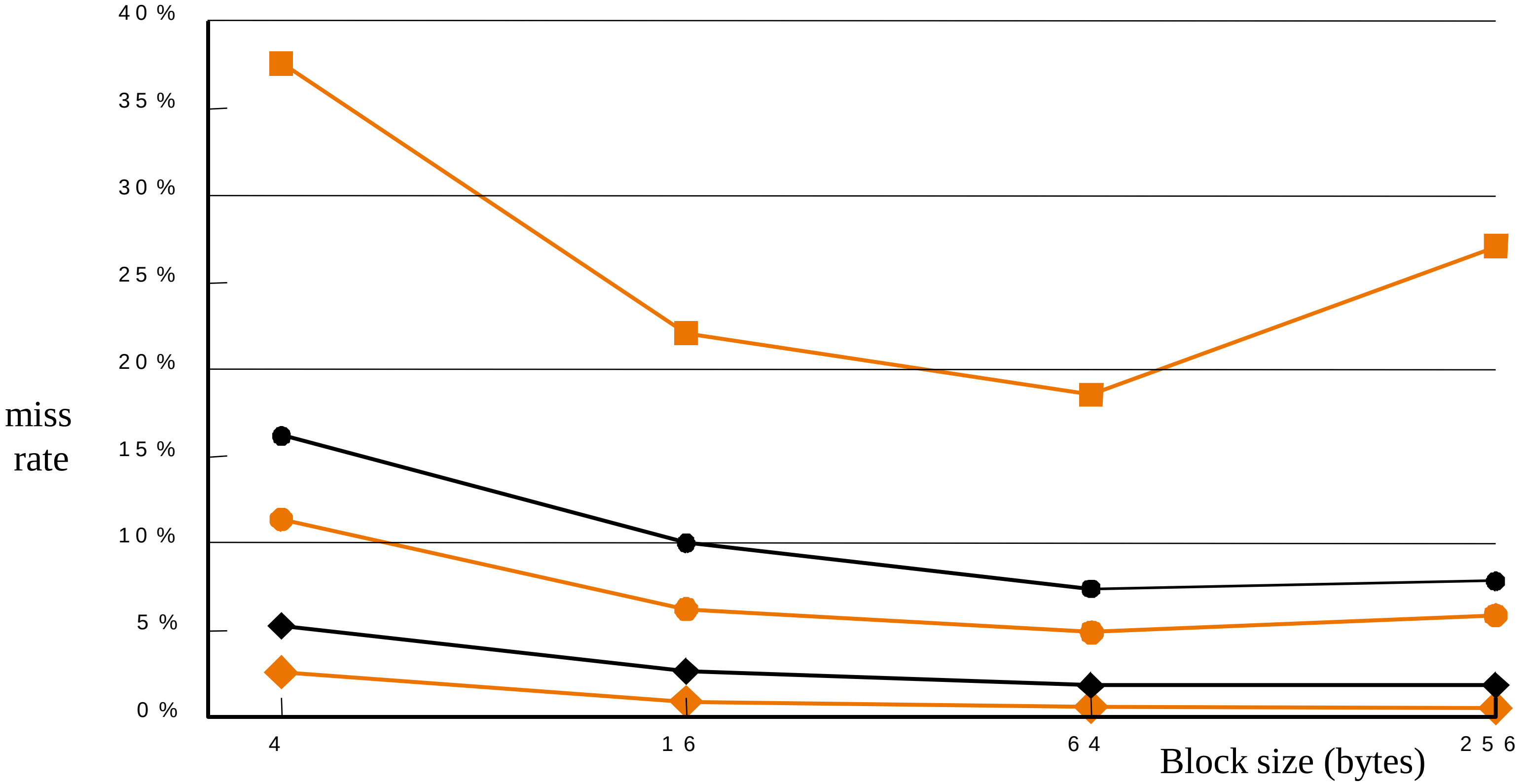
```
addi $3, $0, 1044

lw   $4, 4($3)

addi $3, $3, 1

lb   $6, -4($3)

lw   $6, 0($3)
```



Can’t straddle words in one read

- Needs two accesses to RAM
- Would need ***two*** cycles

Impact of block size on miss rate



- ↑ block size, ↓ miss rate generally
- large block for small cache: ↑ miss rate - too few blocks
- ↑ block size: ↑ transfer time between cache and main memory

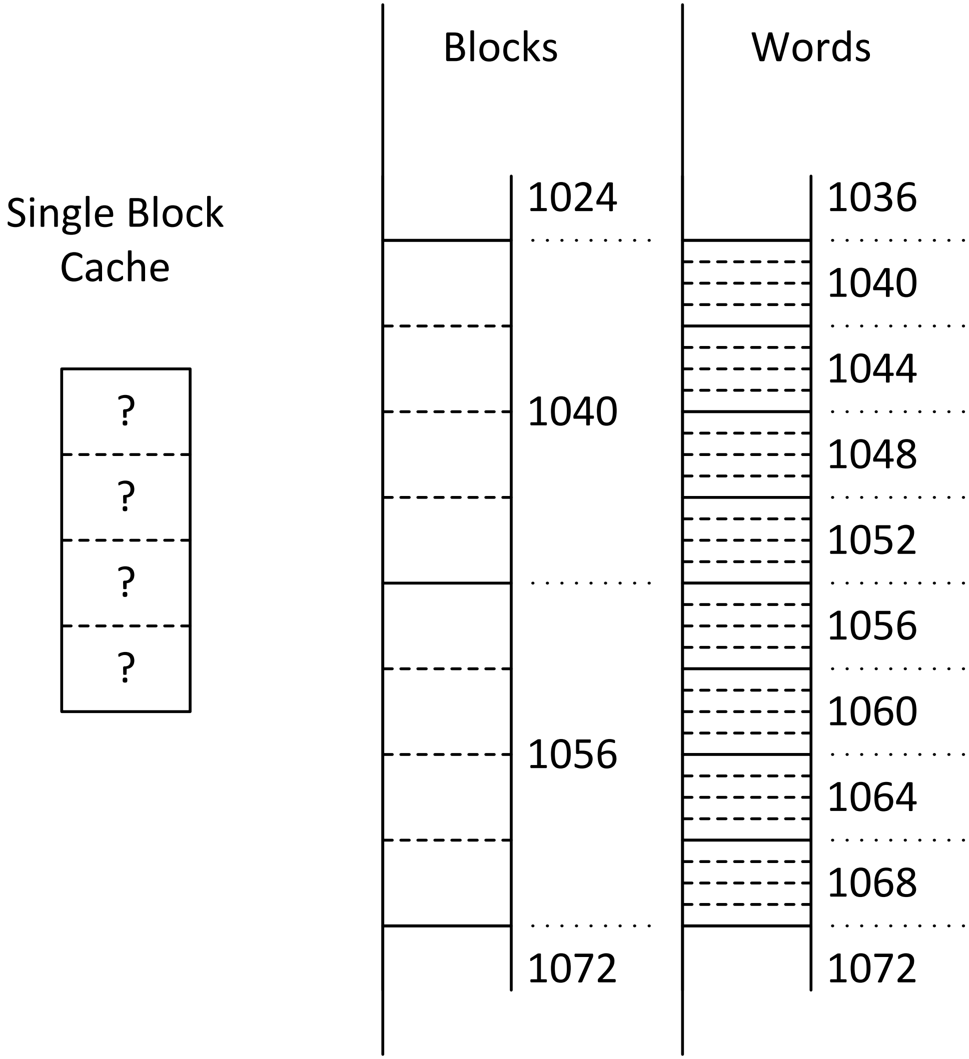
Multi-word cache: handling misses

- cache miss on read:
  - same way as single-word block
  - bring back the entire multi-word block from memory

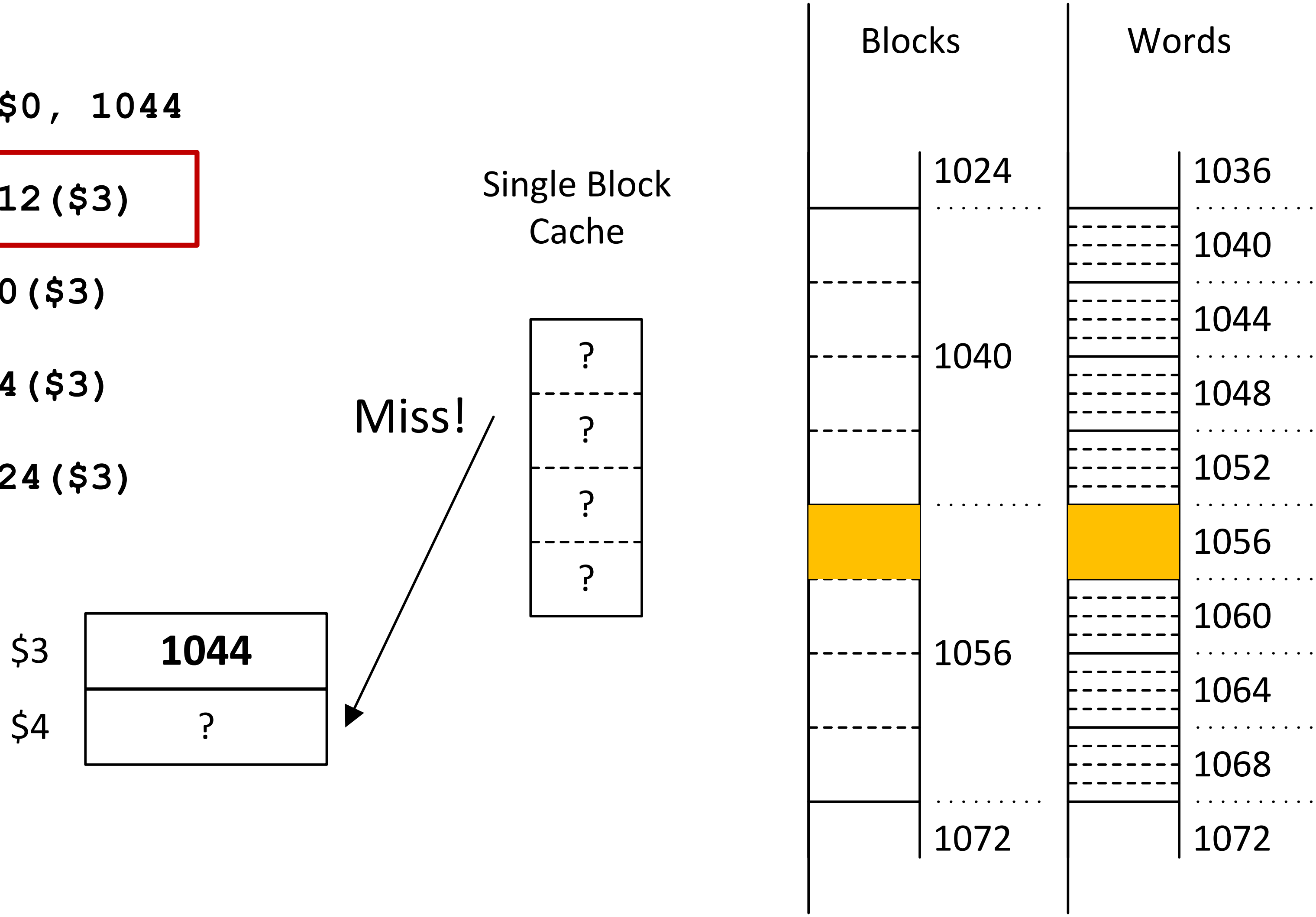
- cache miss on write, given write-through cache:
  - single-word block: disregard hit or miss, just write to cache and write buffer / memory
  - do the same for multi-word block?

```
addi $3, $0, 1044
lw   $4, 12($3)
sw   $4, 0($3)
lw   $4, 4($3)
lw   $4, 24($3)
```

$3	1044
$4	?

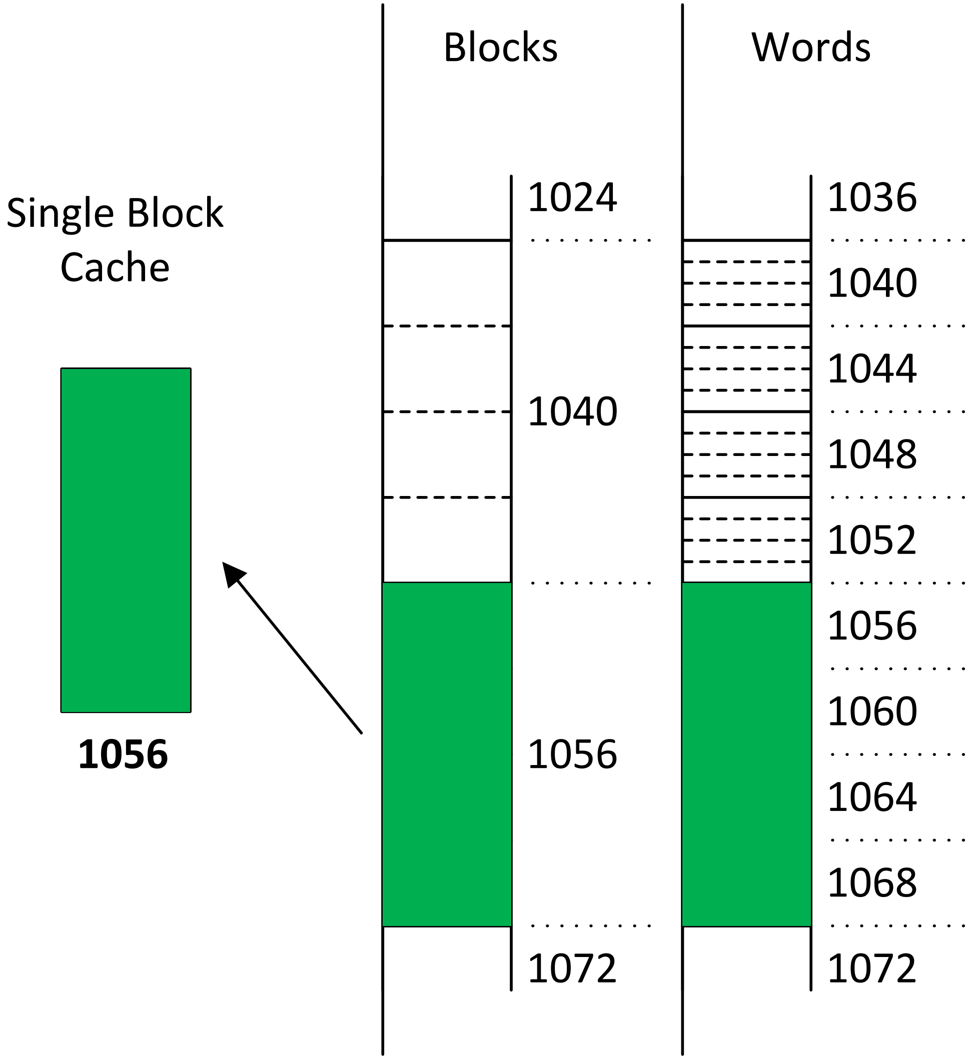


```
addi $3, $0, 1044
lw   $4, 12($3)
sw   $4, 0($3)
lw   $4, 4($3)
lw   $4, 24($3)
```

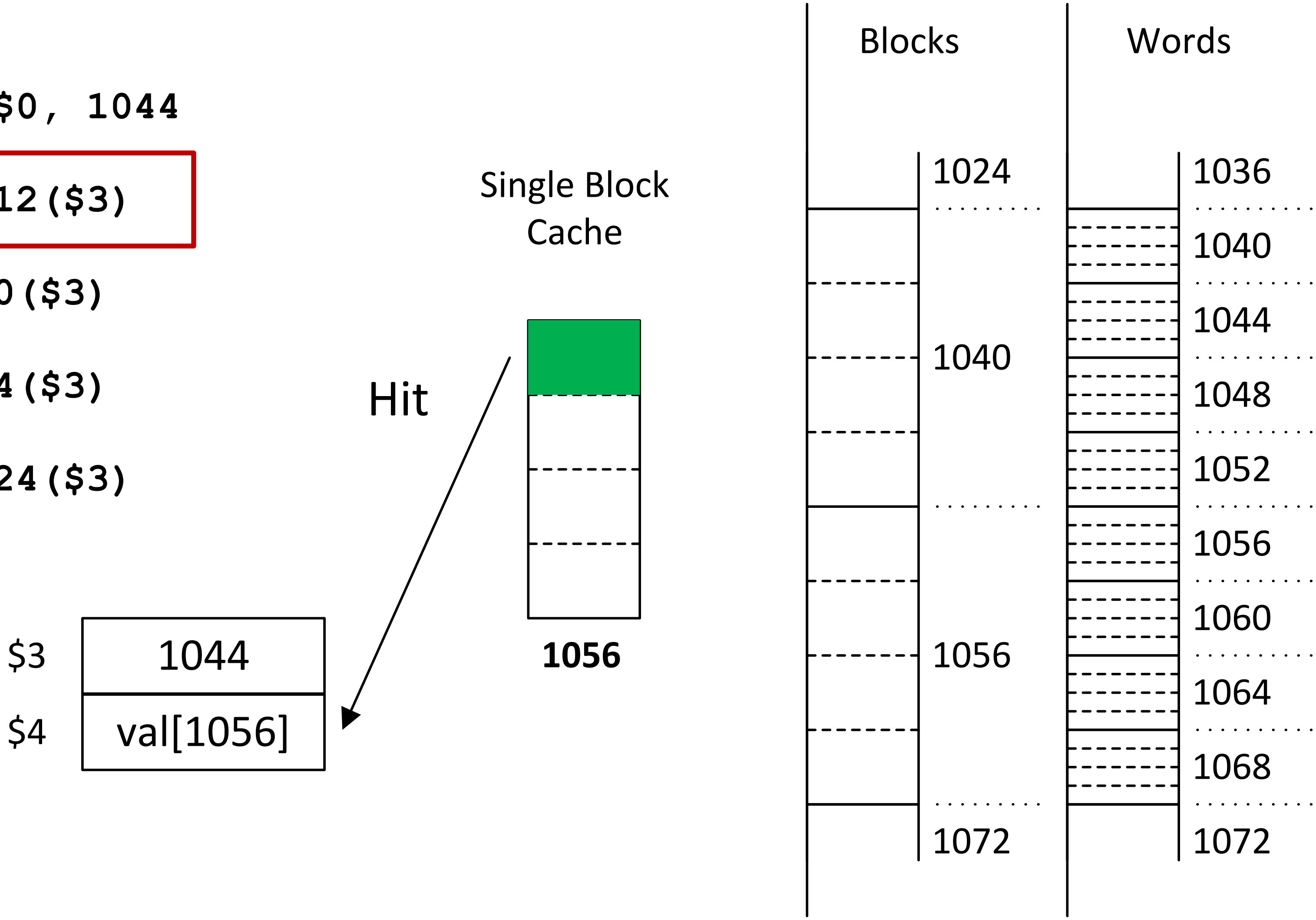


```
addi $3, $0, 1044
lw   $4, 12($3)
sw   $4, 0($3)
lw   $4, 4($3)
lw   $4, 24($3)
```

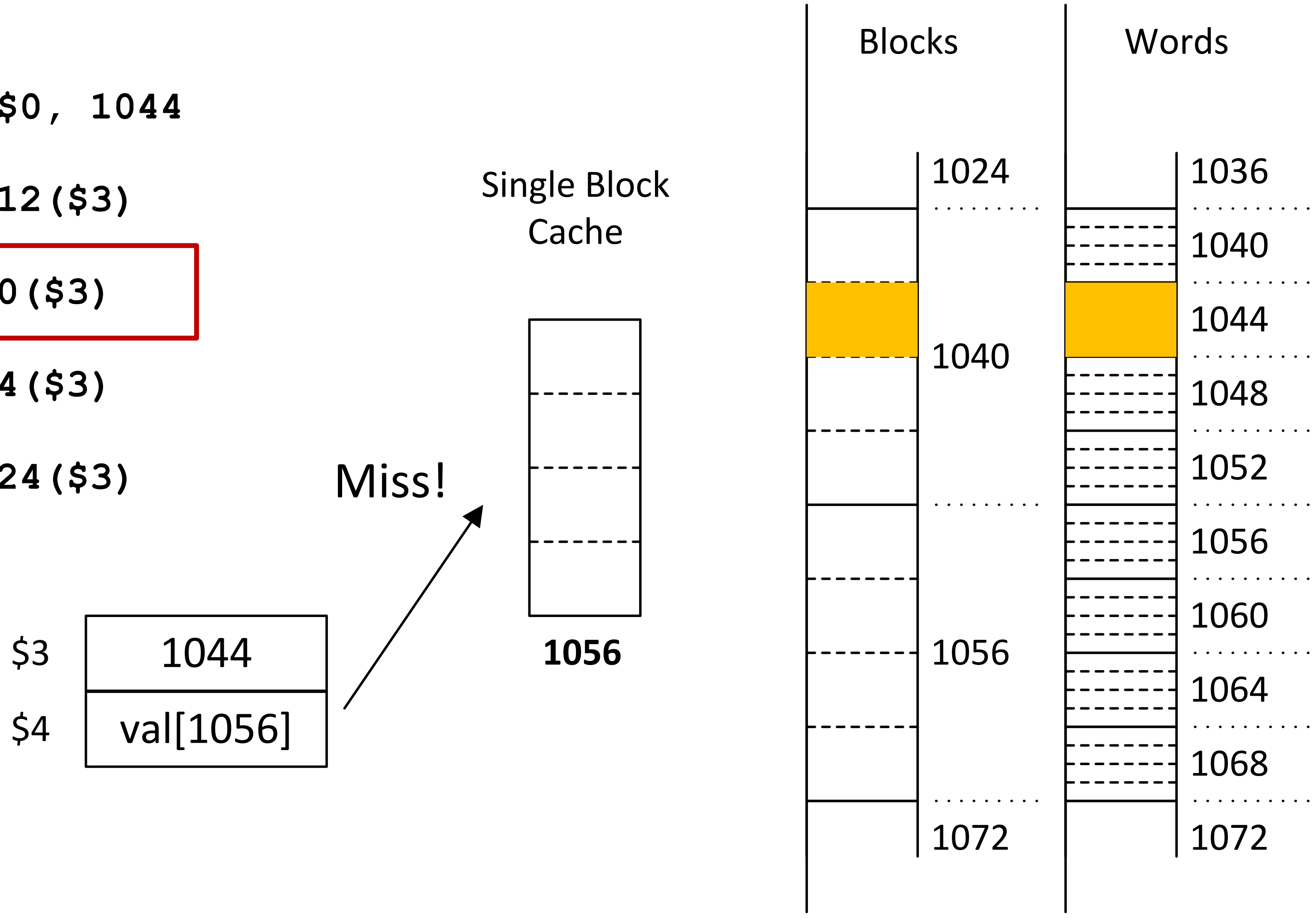
$3	1044
$4	?



```
addi $3, $0, 1044
lw   $4, 12($3)
sw   $4, 0($3)
lw   $4, 4($3)
lw   $4, 24($3)
```

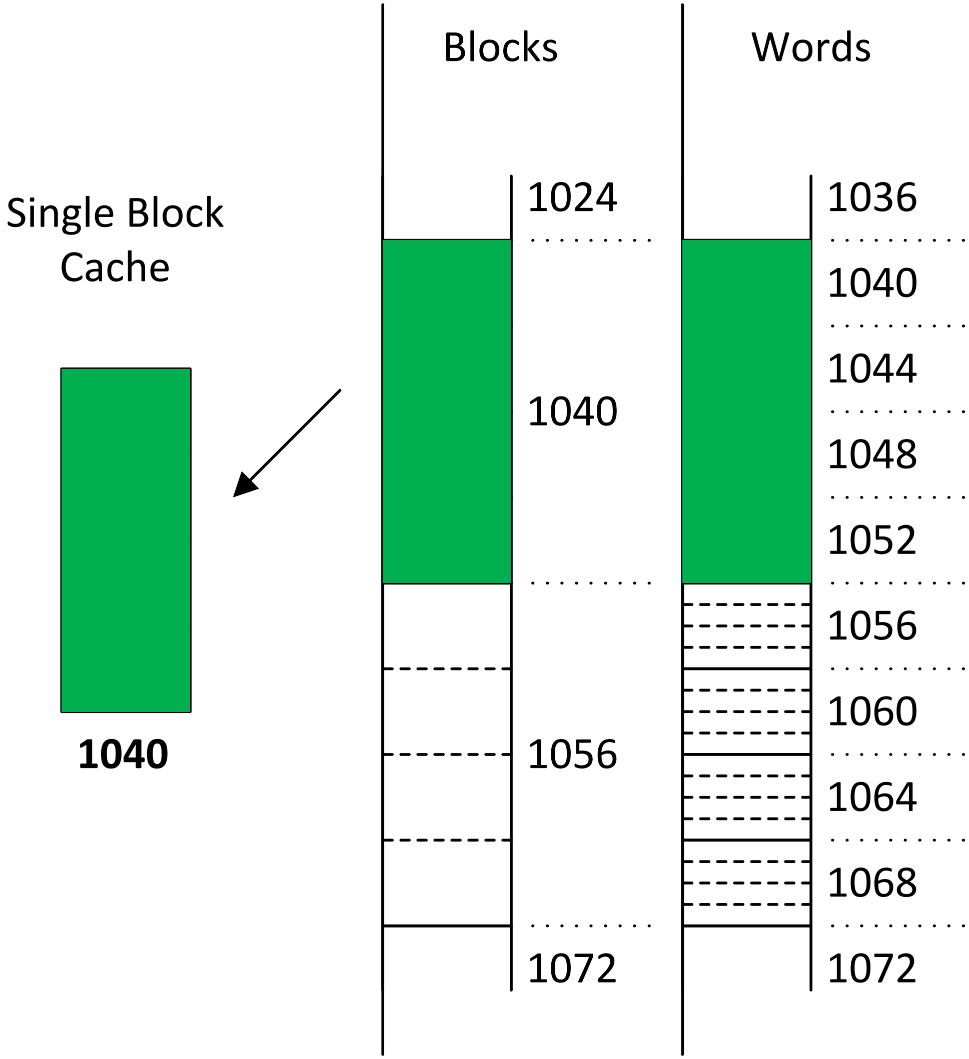


```
addi $3, $0, 1044

lw   $4, 12($3)

sw   $4, 0($3)

lw   $4, 4($3)

lw   $4, 24($3)
```

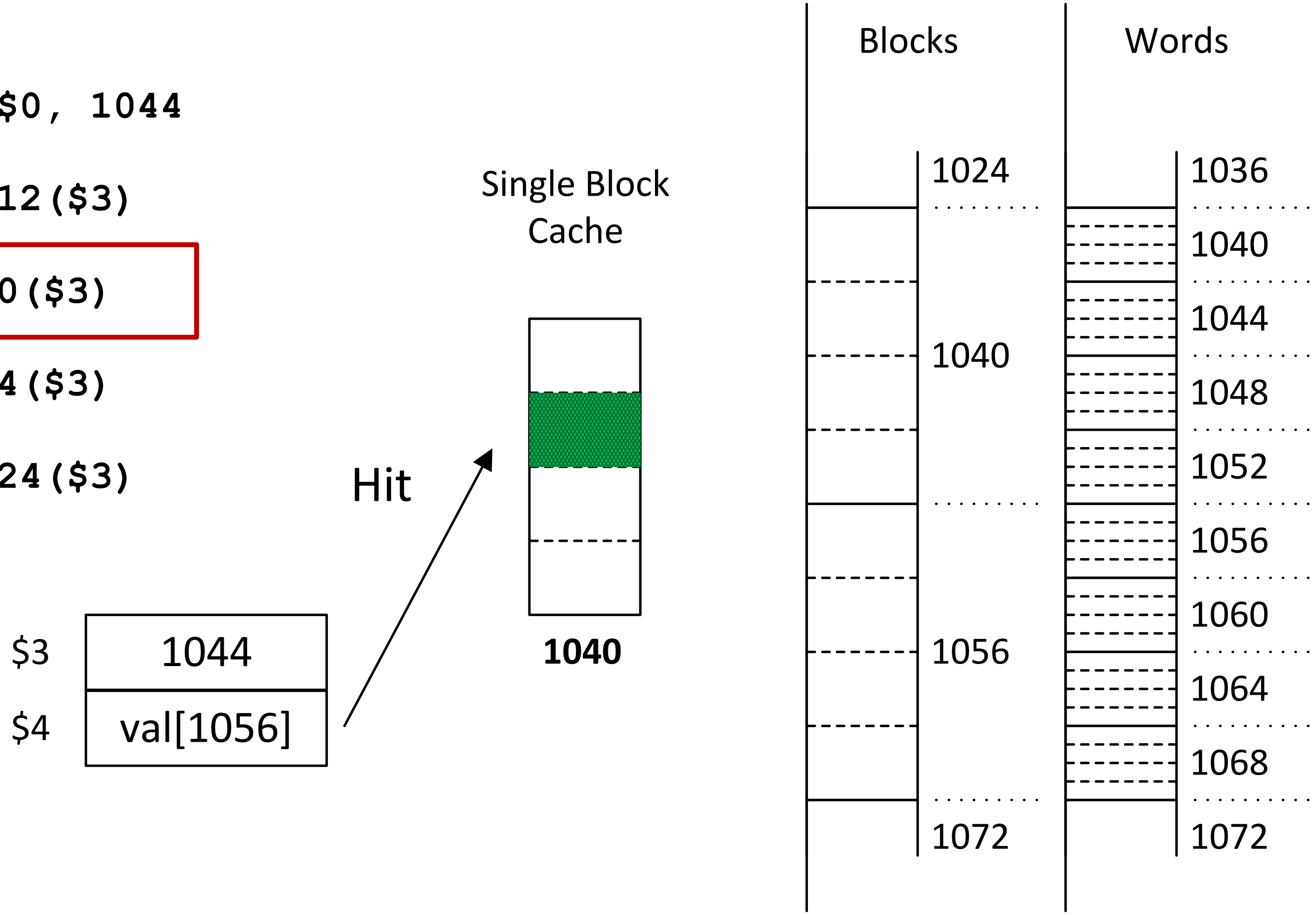


```
addi $3, $0, 1044

lw   $4, 12($3)

sw   $4, 0($3)

lw   $4, 4($3)

lw   $4, 24($3)
```

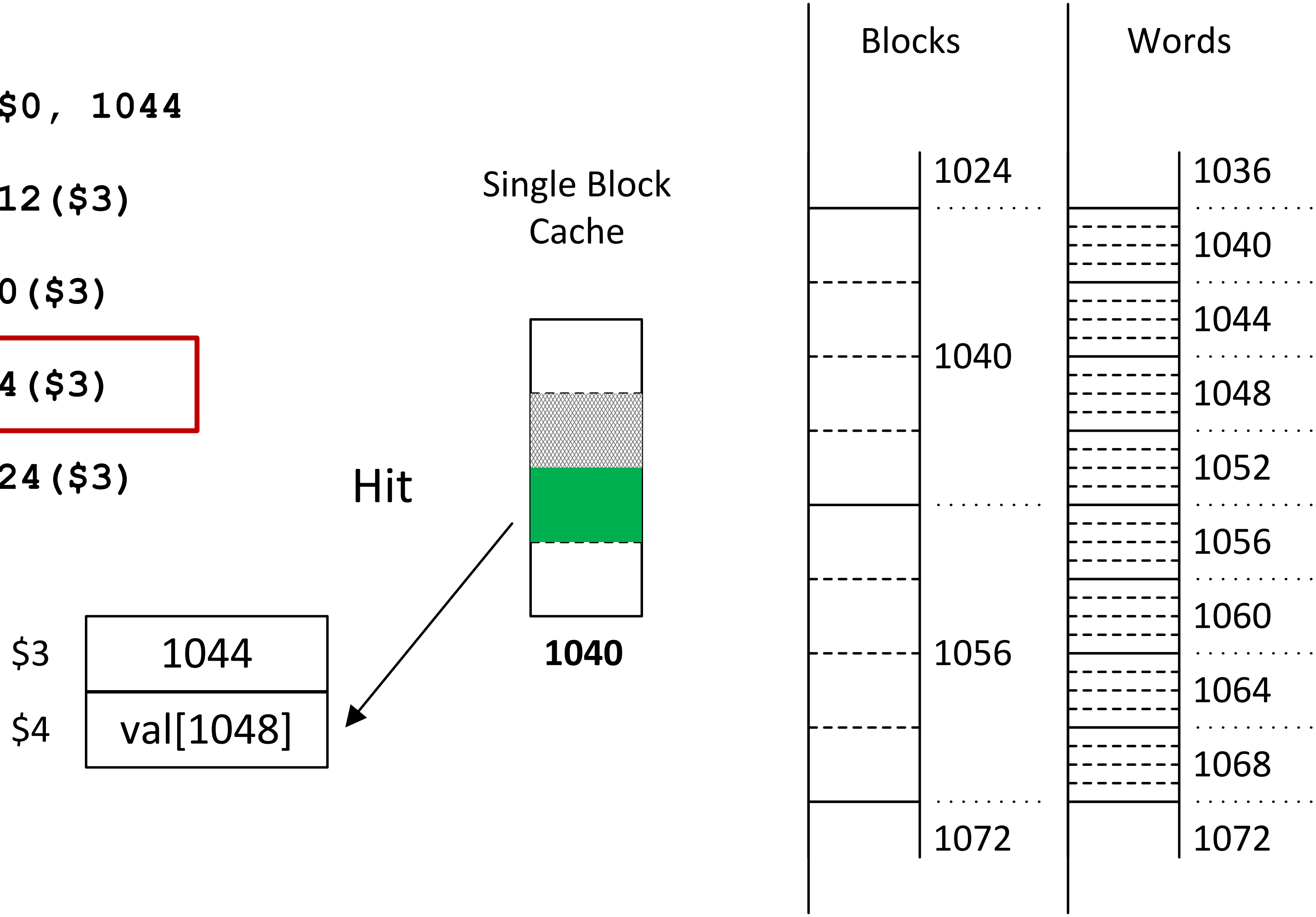
$3	1044
$4	val[1056]



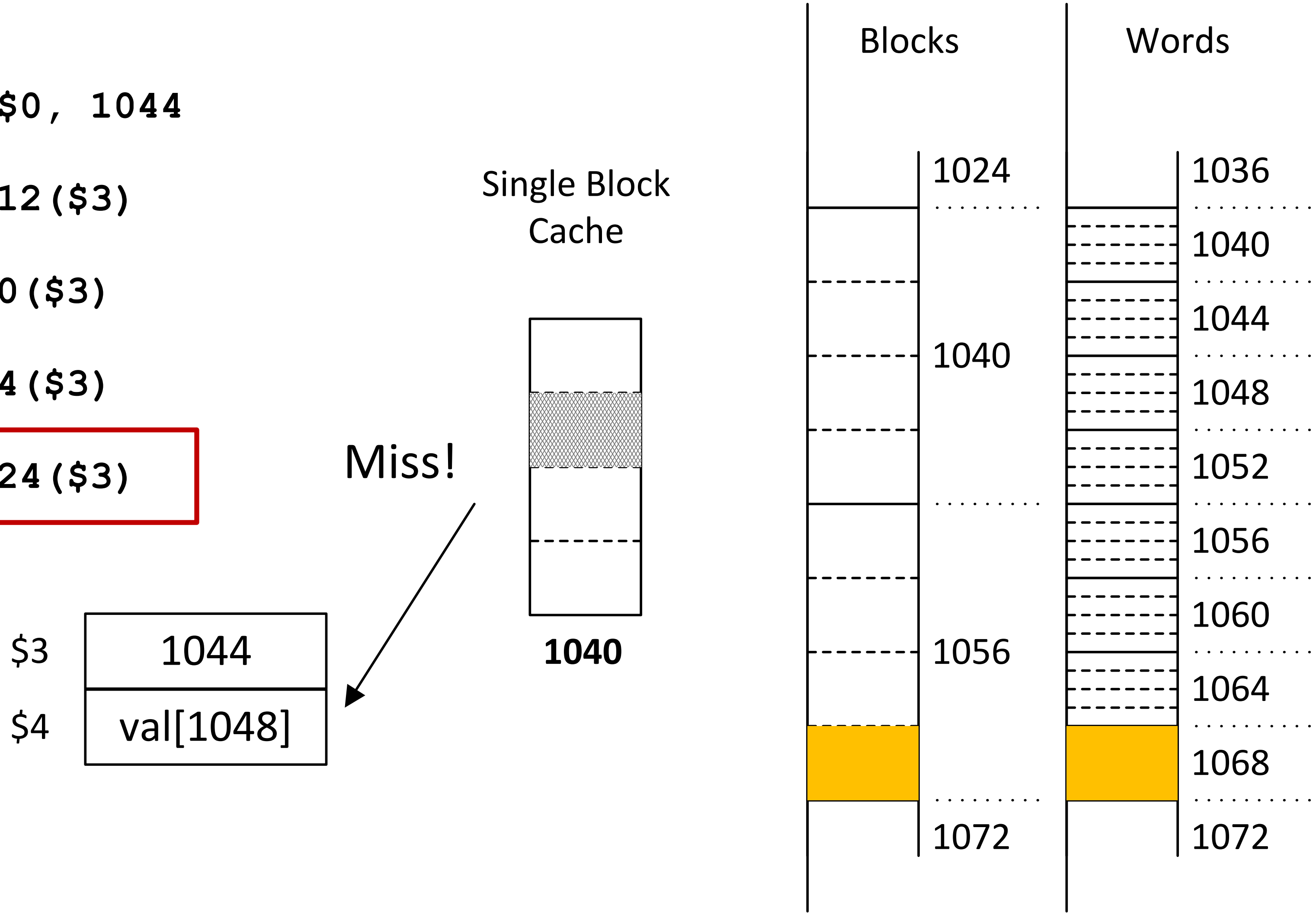
```
addi $3, $0, 1044

lw   $4, 12($3)

sw   $4, 0($3)

lw   $4, 4($3)

lw   $4, 24($3)
```



```
addi $3, $0, 1044

lw   $4, 12($3)

sw   $4, 0($3)

lw   $4, 4($3)

lw   $4, 24($3)
```

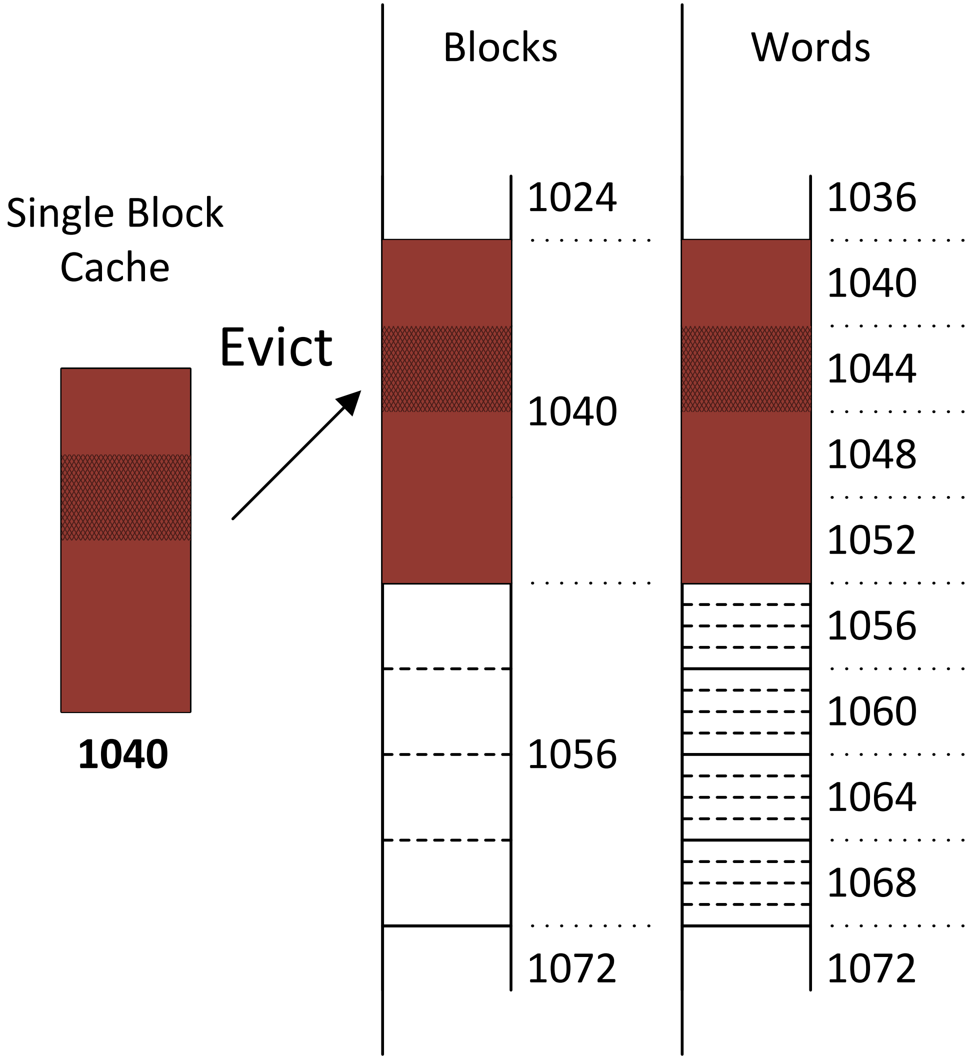


```
addi $3, $0, 1044
lw   $4, 12($3)
sw   $4, 0($3)
lw   $4, 4($3)
lw   $4, 24($3)
```



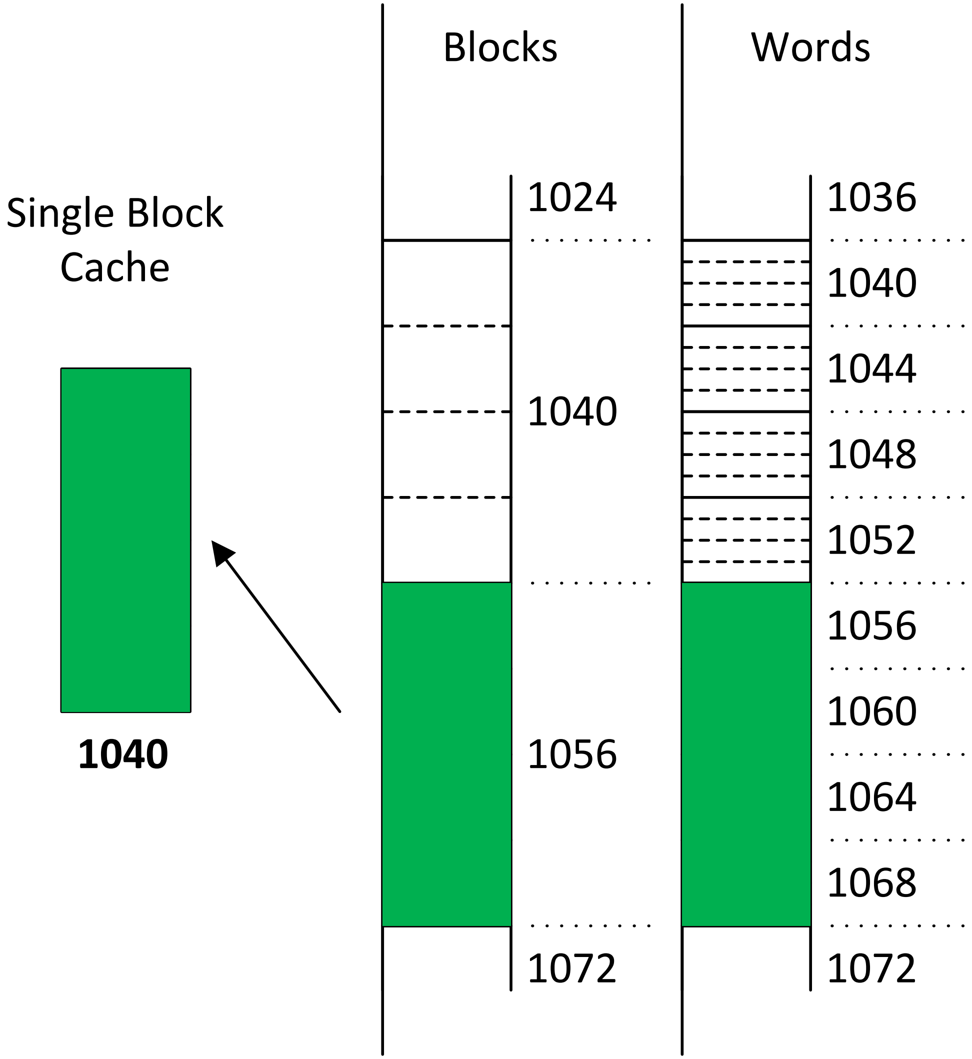
```
addi $3, $0, 1044

lw   $4, 12($3)

sw   $4, 0($3)

lw   $4, 4($3)

lw   $4, 24($3)
```

$3	1044
$4	val[1048]

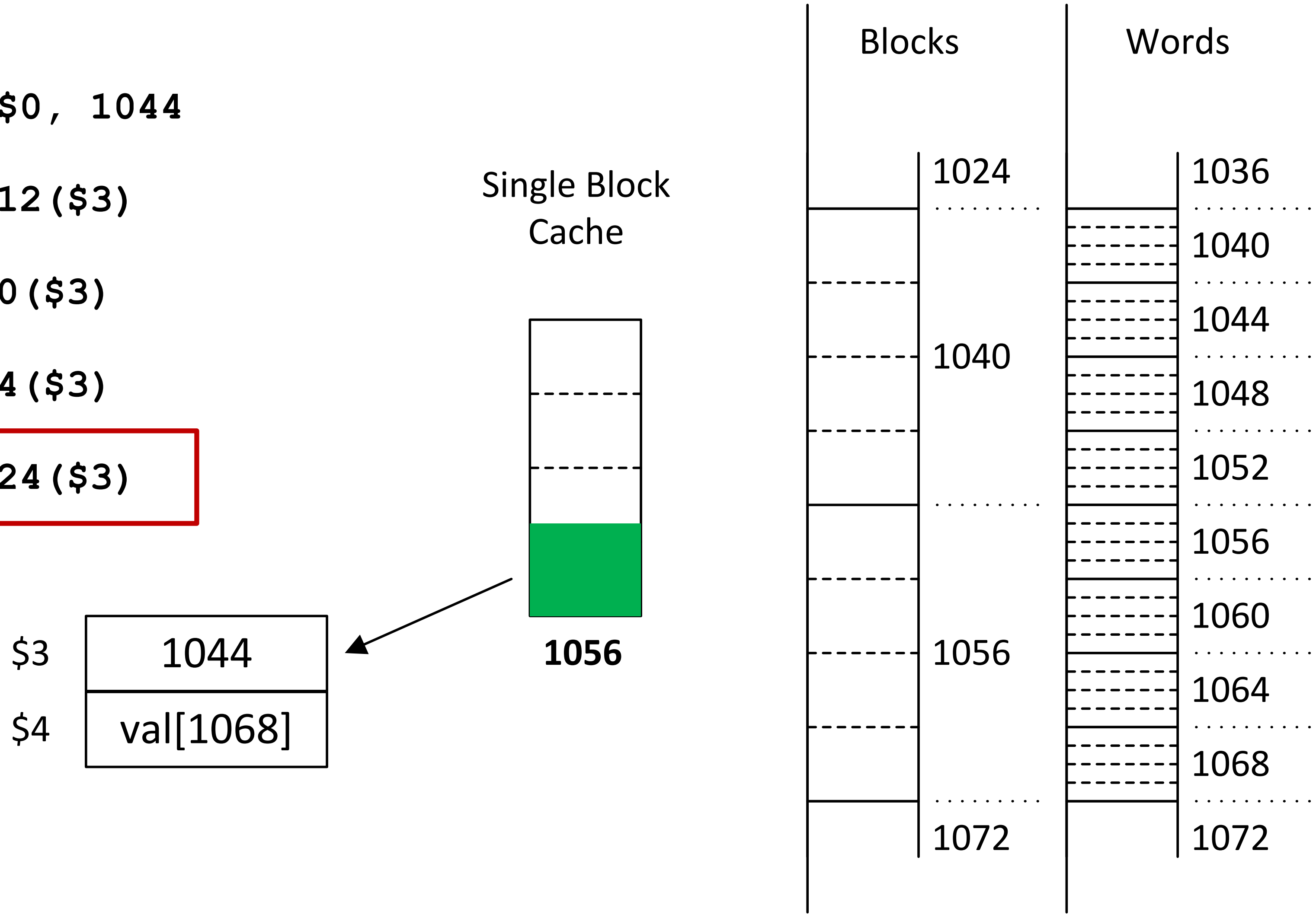


```
addi $3, $0, 1044

lw   $4, 12($3)

sw   $4, 0($3)

lw   $4, 4($3)

lw   $4, 24($3)
```

$3	1044
$4	val[1048]



```
addi $3, $0, 1044

lw   $4, 12($3)

sw   $4, 0($3)

lw   $4, 4($3)

lw   $4, 24($3)
```



Cache performance

- CPU time = (execution cycles + memory stall cycles)
  × cycle time

Cache performance

- CPU time = (execution cycles + memory stall cycles) × cycle time
- memory stall cycles = read stall cycles + write stall cycles

Cache performance

- CPU time = (execution cycles + memory stall cycles) × cycle time
- memory stall cycles = read stall cycles + write stall cycles
- read stall cycles =

Cache performance

- CPU time = (execution cycles + memory stall cycles) × cycle time
- memory stall cycles = read stall cycles + write stall cycles
- read stall cycles = $\underbrace{\frac{\text{reads}}{\text{program}} \times \begin{matrix}\text{read miss}\\ \text{rate (\%)}\end{matrix}}_{\text{read miss per program}} \times \underbrace{\begin{matrix}\text{read miss}\\ \text{penalty (cycle)}\end{matrix}}_{\begin{matrix}\text{number of cycles}\\ \text{per read miss}\end{matrix}}$

Cache performance

- CPU time = (execution cycles + memory stall cycles) × cycle time

- memory stall cycles = read stall cycles + write stall cycles

- read stall cycles = $\underbrace{\frac{\text{reads}}{\text{program}} \times \text{read miss rate (\%)}}_{\text{read miss per program}} \times \underbrace{\text{read miss penalty (cycle)}}_{\text{number of cycles per read miss}}$

- write stall cycles =

Cache performance

- CPU time = (execution cycles + memory stall cycles) × cycle time
- memory stall cycles = read stall cycles + write stall cycles
- read stall cycles = $\underbrace{\frac{\text{reads}}{\text{program}} \times \begin{array}{l}\text{read miss}\\ \text{rate (\%)}\end{array}}_{\text{read miss per program}} \times \underbrace{\begin{array}{l}\text{read miss}\\ \text{penalty (cycle)}\end{array}}_{\substack{\text{number of cycles}\\ \text{per read miss}}}$
- write stall cycles = $\frac{\text{writes}}{\text{program}} \times \begin{array}{l}\text{write miss}\\ \text{rate (\%)}\end{array} \times \begin{array}{l}\text{write miss}\\ \text{penalty (cycle)}\end{array}$

  + write buffer stalls (when full)

Effect on CPU

- assume hit time insignificant (data transfer time dominated)
- let c: CPI-no stall, i: instruction miss rate
  p: miss penalty, d: data miss rate
  n: instruction count, f: load/store frequency
- total memory stall cycles: nip + nfdp

Effect on CPU

- assume hit time insignificant (data transfer time dominated)
- let c: CPI-no stall, i: instruction miss rate
  p: miss penalty, d: data miss rate
  n: instruction count, f: load/store frequency
- total memory stall cycles: $nip + nfdp$
- total CPU cycles without stall: nc
  total CPU cycles with stall: $n(c + ip + fdp)$

Effect on CPU

- assume hit time insignificant (data transfer time dominated)
- let c: CPI-no stall, i: instruction miss rate
  p: miss penalty, d: data miss rate
  n: instruction count, f: load/store frequency
- total memory stall cycles: $nip + nfdp$
- total CPU cycles without stall: nc
  total CPU cycles with stall: $n(c + ip + fdp)$
- % time on stall:

$$\frac{ip + fdp}{c + ip + fdp}$$

Faster CPU

- same memory speed, halved CPI
  - % time on stall = $\dfrac{\text{ip} + \text{fdp}}{(½)\text{c} + \text{ip} + \text{fdp}} > \dfrac{\text{ip} + \text{fdp}}{\text{c} + \text{ip} + \text{fdp}}$
  - lower CPI results in greater impact of stall cycles

Faster CPU

- same memory speed, halved CPI
  - % time on stall = $\dfrac{ip + fdp}{(½)c + ip + fdp} > \dfrac{ip + fdp}{c + ip + fdp}$
  - lower CPI results in greater impact of stall cycles
- same memory speed, halved clock cycle time t
  - miss penalty: 2p
  - total CPU time with new clock: $n(c + 2ip + 2fdp)$

Faster CPU

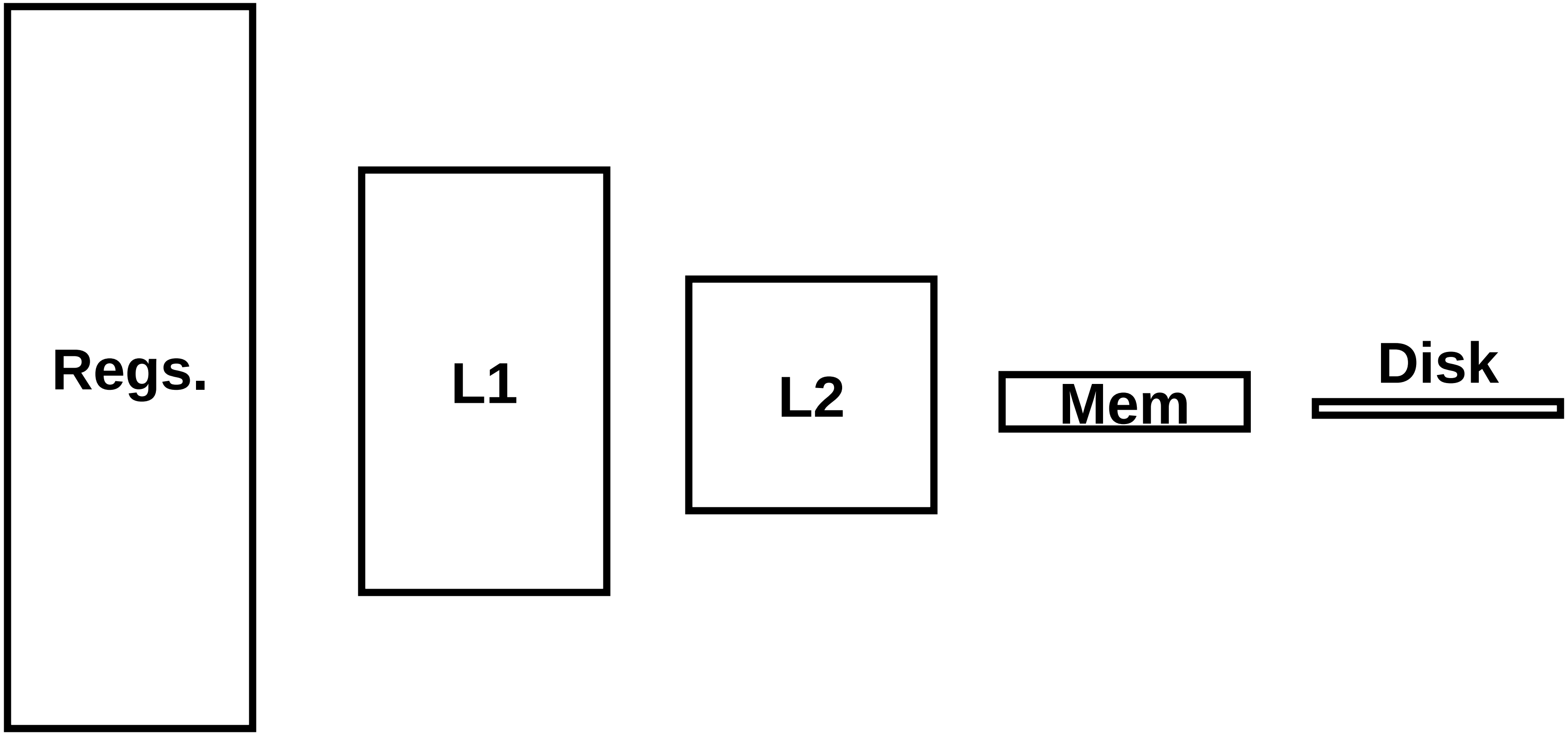
- same memory speed, halved CPI
  - % time on stall = $\dfrac{ip + fdp}{(½)c + ip + fdp} > \dfrac{ip + fdp}{c + ip + fdp}$
  - lower CPI results in greater impact of stall cycles
- same memory speed, halved clock cycle time t
  - miss penalty: $2p$
  - total CPU time with new clock: $n(c + 2ip + 2fdp)$
  - performance improvement = $\dfrac{\text{exec. time with old clock}}{\text{exec. time with new clock}}$

$$= \frac{n(c + ip + fdp) \times t}{n(c + 2ip + 2fdp) \times t/2}$$

$$= \frac{c + ip + fdp}{c + 2ip + 2fdp} \times 2 < 2$$

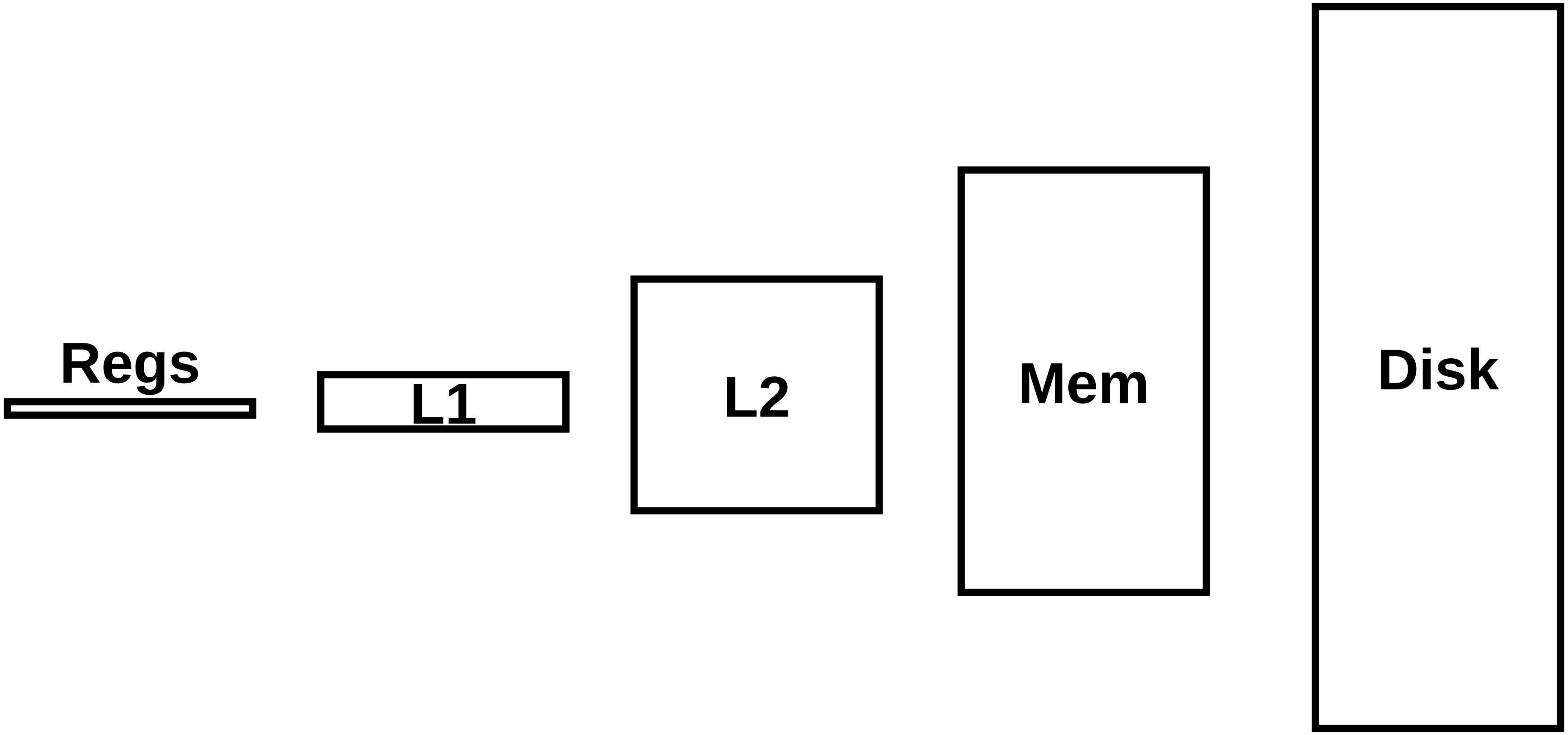
Multi-level cache hierarchy

- how can we look at the cache hierarchy?
  - – performance view



Multi-level cache hierarchy

- how can we look at the cache hierarchy?
  - performance view
  - capacity view

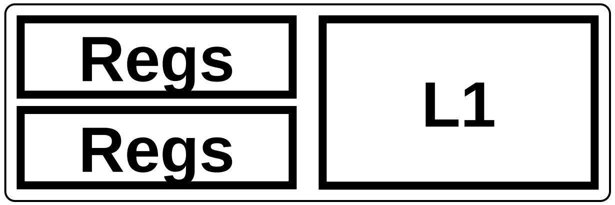


Multi-level cache hierarchy

- how can we look at the cache hierarchy?
  - performance view
  - capacity view
  - physical hierarchy

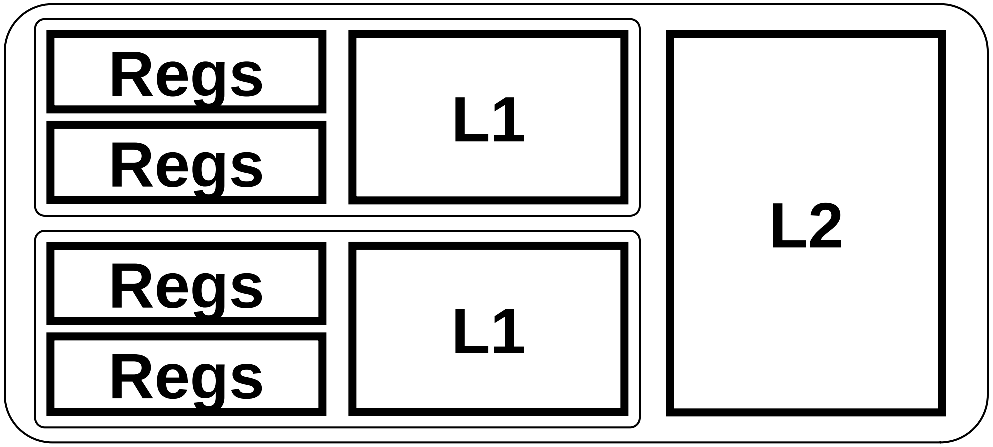
Multi-level cache hierarchy

- how can we look at the cache hierarchy?
  - performance view
  - capacity view
  - physical hierarchy



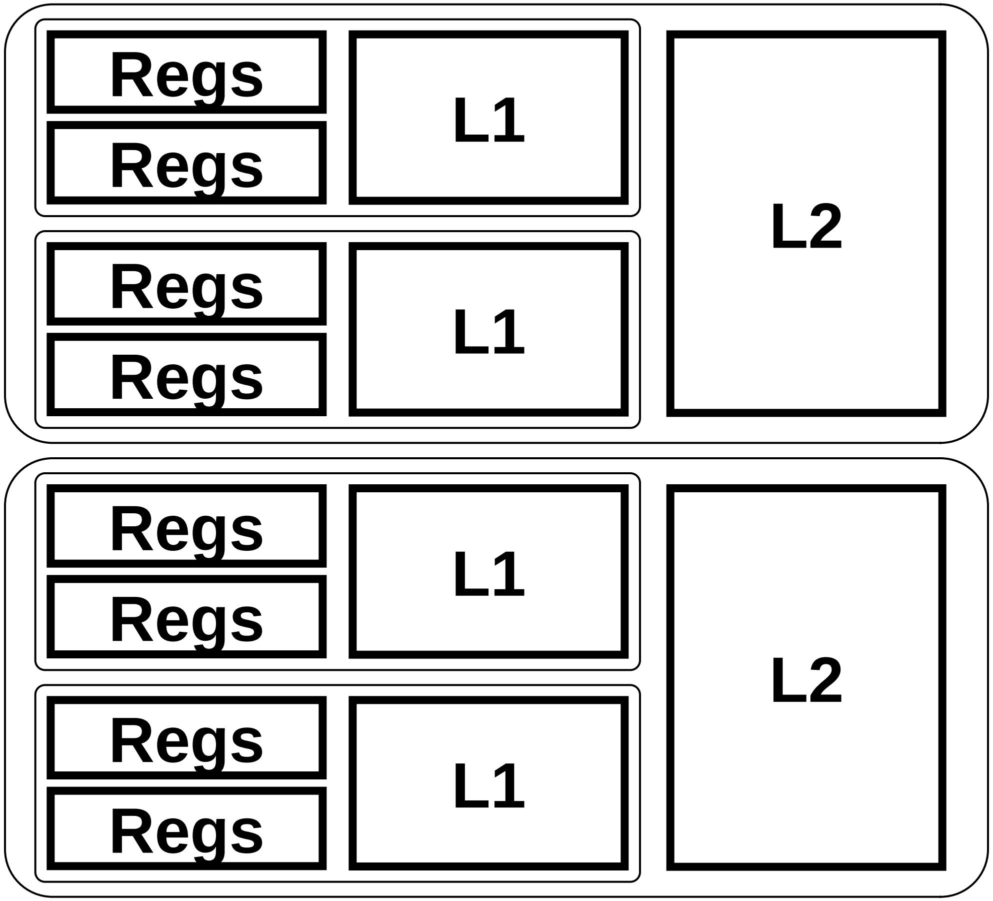
Multi-level cache hierarchy

- how can we look at the cache hierarchy?
  - performance view
  - capacity view
  - physical hierarchy



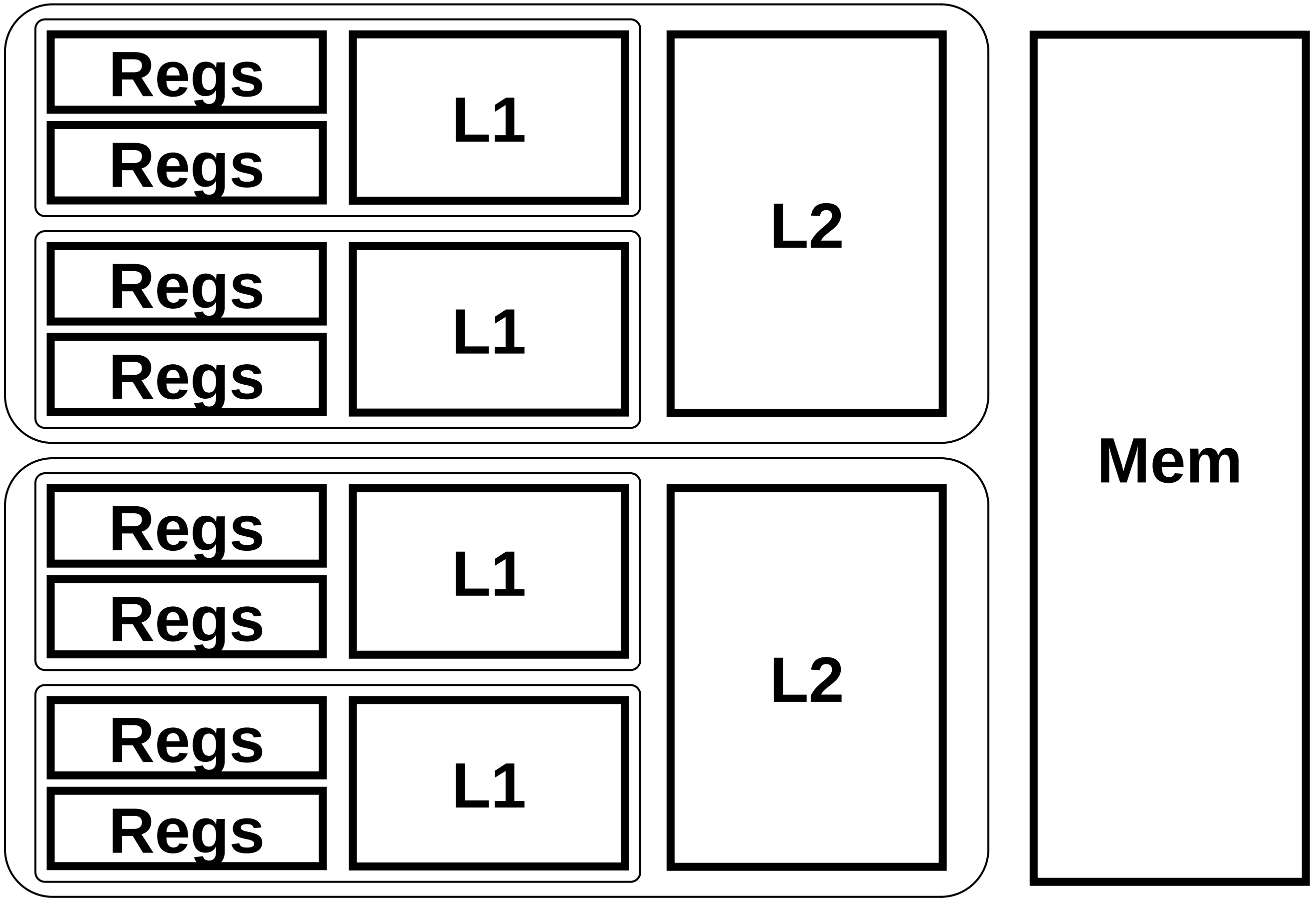
Multi-level cache hierarchy

- how can we look at the cache hierarchy?
  - performance view
  - capacity view
  - physical hierarchy



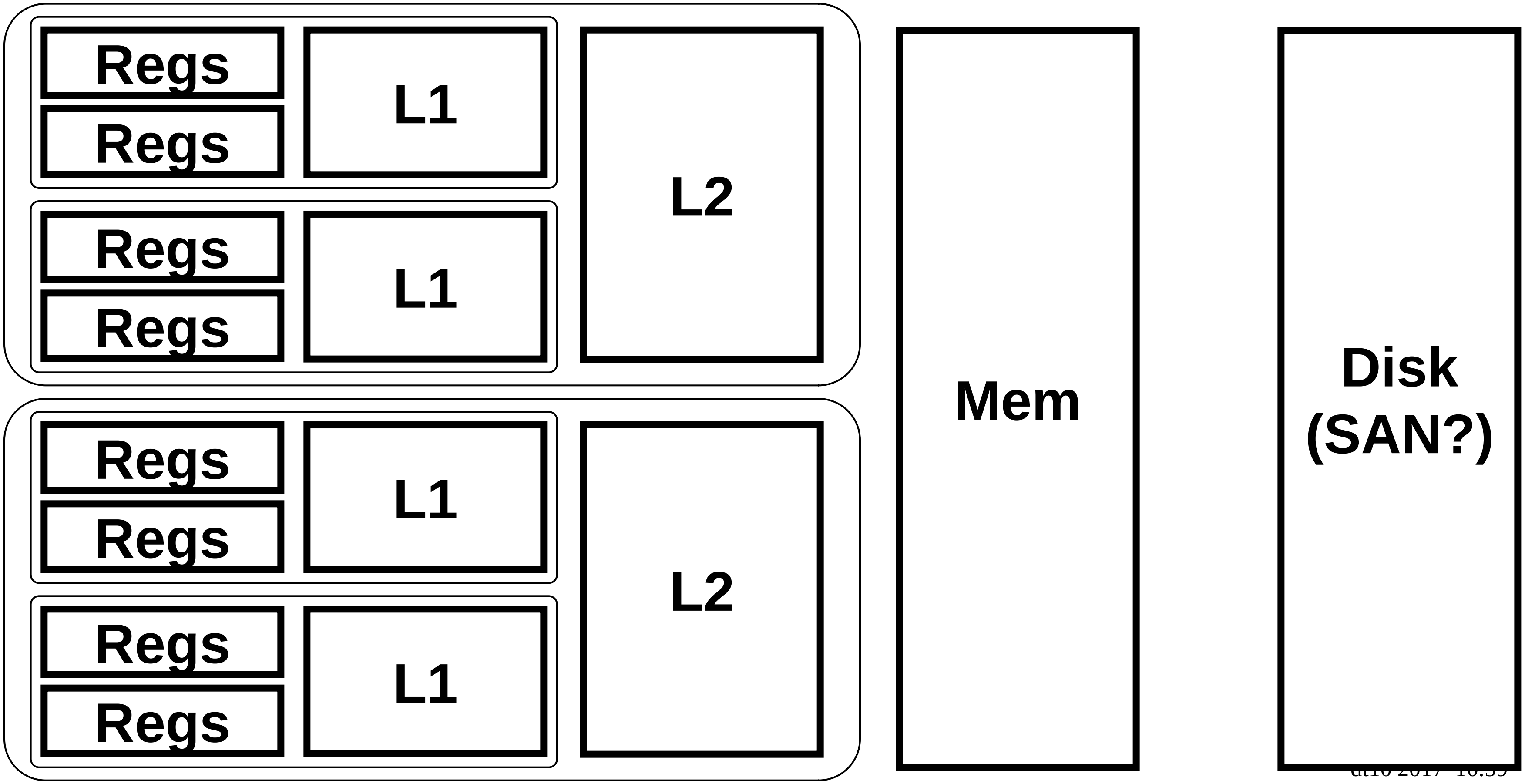
Multi-level cache hierarchy

- how can we look at the cache hierarchy?
  - performance view
  - capacity view
  - physical hierarchy



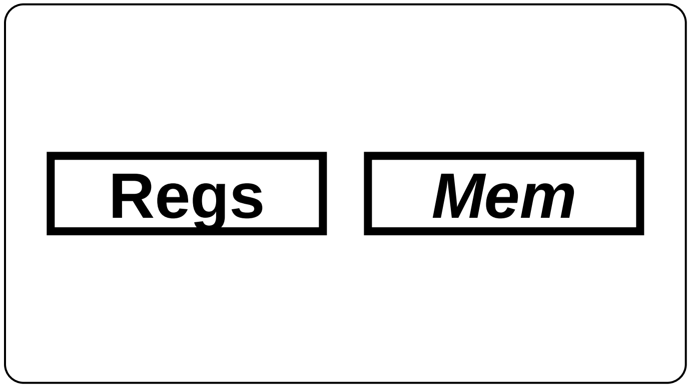
Multi-level cache hierarchy

- how can we look at the cache hierarchy?
  - performance view
  - capacity view
  - physical hierarchy



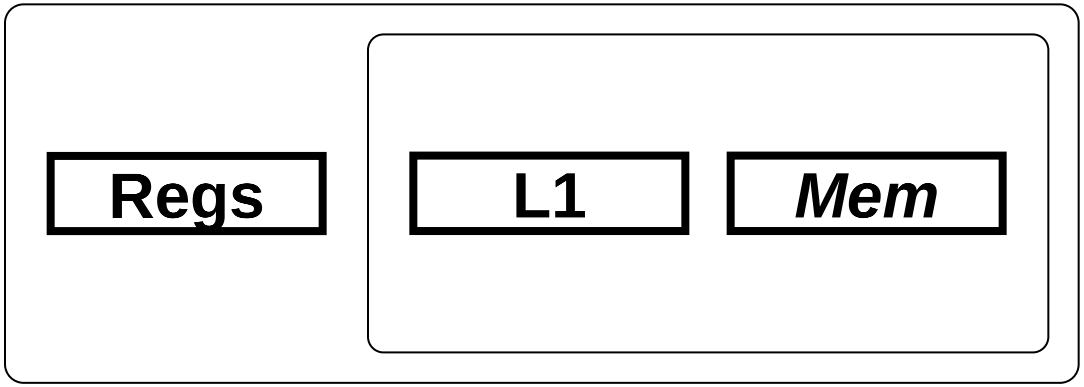
Multi-level cache hierarchy

- how can we look at the cache hierarchy?
  - performance view
  - capacity view
  - physical hierarchy
  - abstract hierarchy



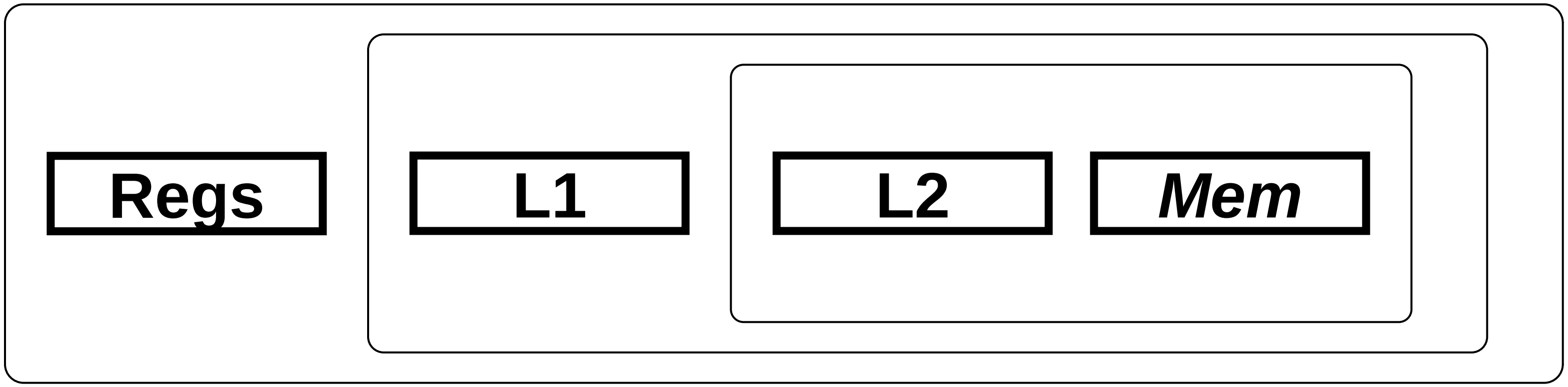
Multi-level cache hierarchy

- how can we look at the cache hierarchy?
  - performance view
  - capacity view
  - physical hierarchy
  - abstract hierarchy



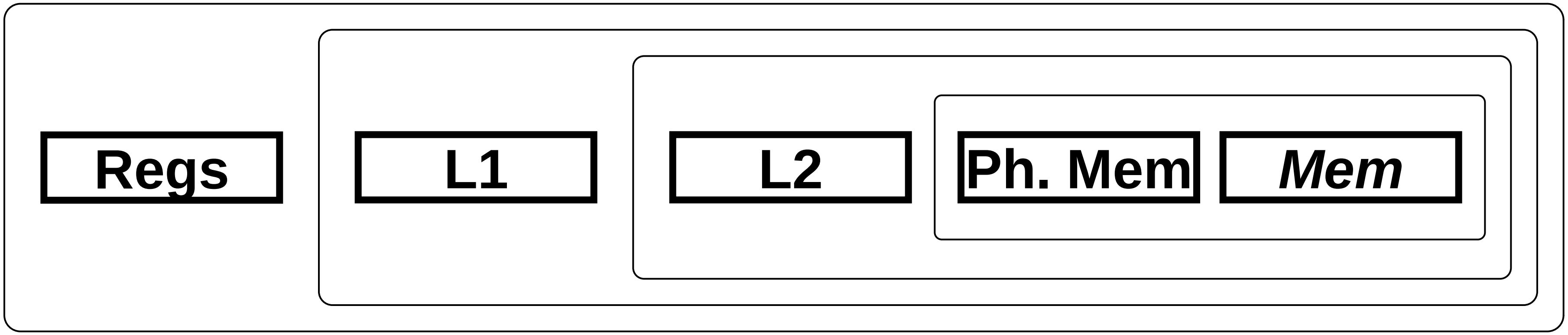
Multi-level cache hierarchy

- how can we look at the cache hierarchy?
  - performance view
  - capacity view
  - physical hierarchy
  - abstract hierarchy



Multi-level cache hierarchy

- how can we look at the cache hierarchy?
  - performance view
  - capacity view
  - physical hierarchy
  - abstract hierarchy



Typical scale

- L1
  - size: tens of KB
  - hit time: complete in one clock cycle
  - miss rates: 1-5%
- L2:
  - size: hundreds of KB
  - hit time: a few clock cycles
  - miss rates: 10-20%
- L2 miss rate: fraction of L1 misses also miss L2
  - why so high?
- complex: different block size/placement for L1, L2