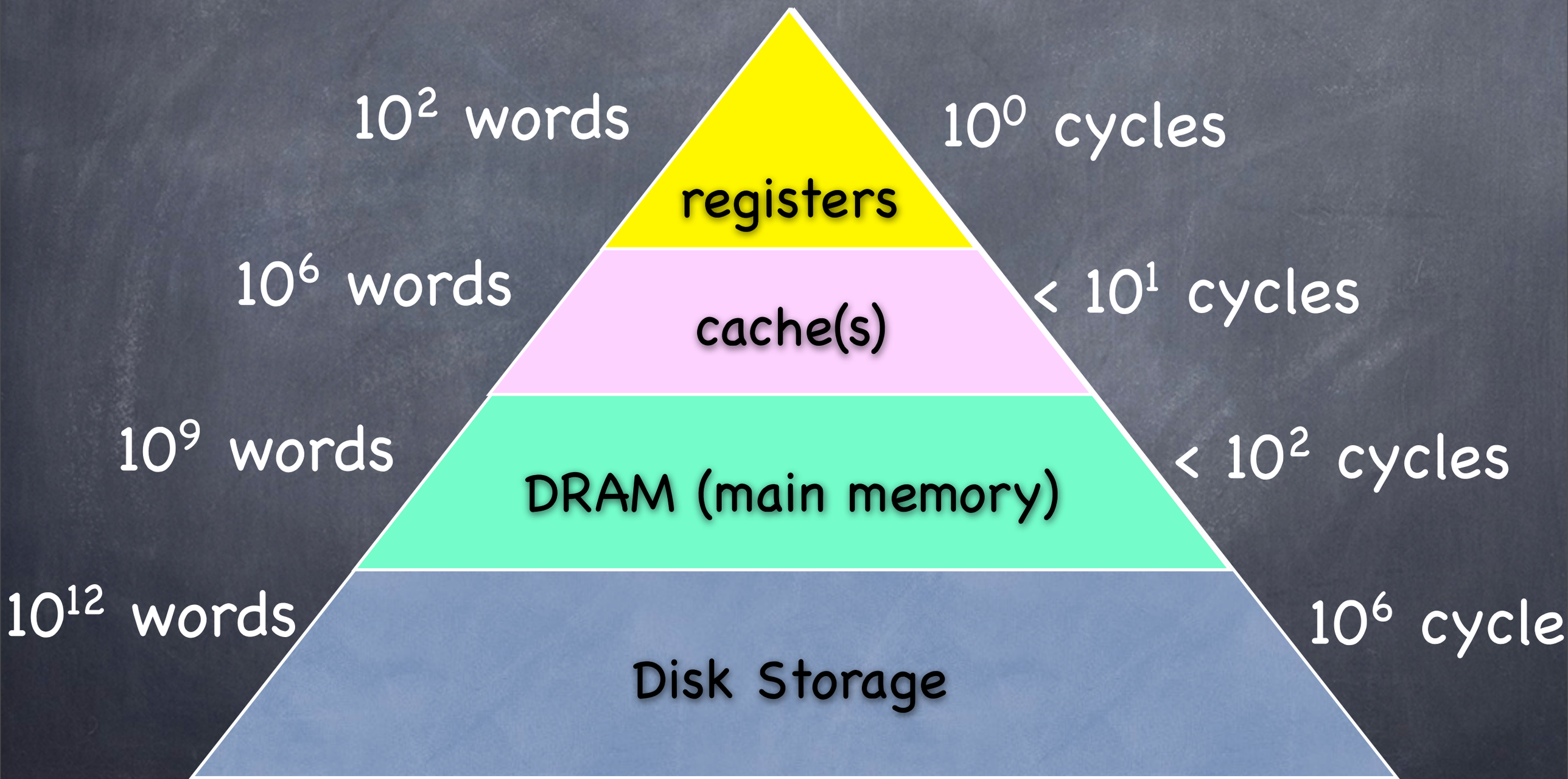
Announcements

TA Application Time - Due by Monday

Cash, Cache, or Cachet?



Units and Measures

	inch	foot	yard	fathom
Length in inches	1	12	36	72

inches away	in inches	in feet	in yards	in fathoms
0	0	0	0	0
12	4	1	?	?
72	72	6	2	1
360	360	30	10	5

Memory Units and Addresses

	byte	word	cache block	page
Size in bytes	1	4	16	1024

byte number	byte address	word address	cache block number	page number
0	0	0	0	0
4	4	1	?	?
16	16	4	1	?
1024	1024	256	63	1
8196	8196	2048	512	8

Memory Units and Addresses

	byte	word	cache block	page
Size in bytes	1	4	16	1024

byte number	byte address	word address	cache block number	page number
0	0	0	0	0
4	4	1	0	0
16	16	4	1	0
1024	1024	256	63	1
8196	8196	2048	512	8

Memory Units and Addresses

	byte	word	cache block	page
Size in bytes	1	4	4	4096

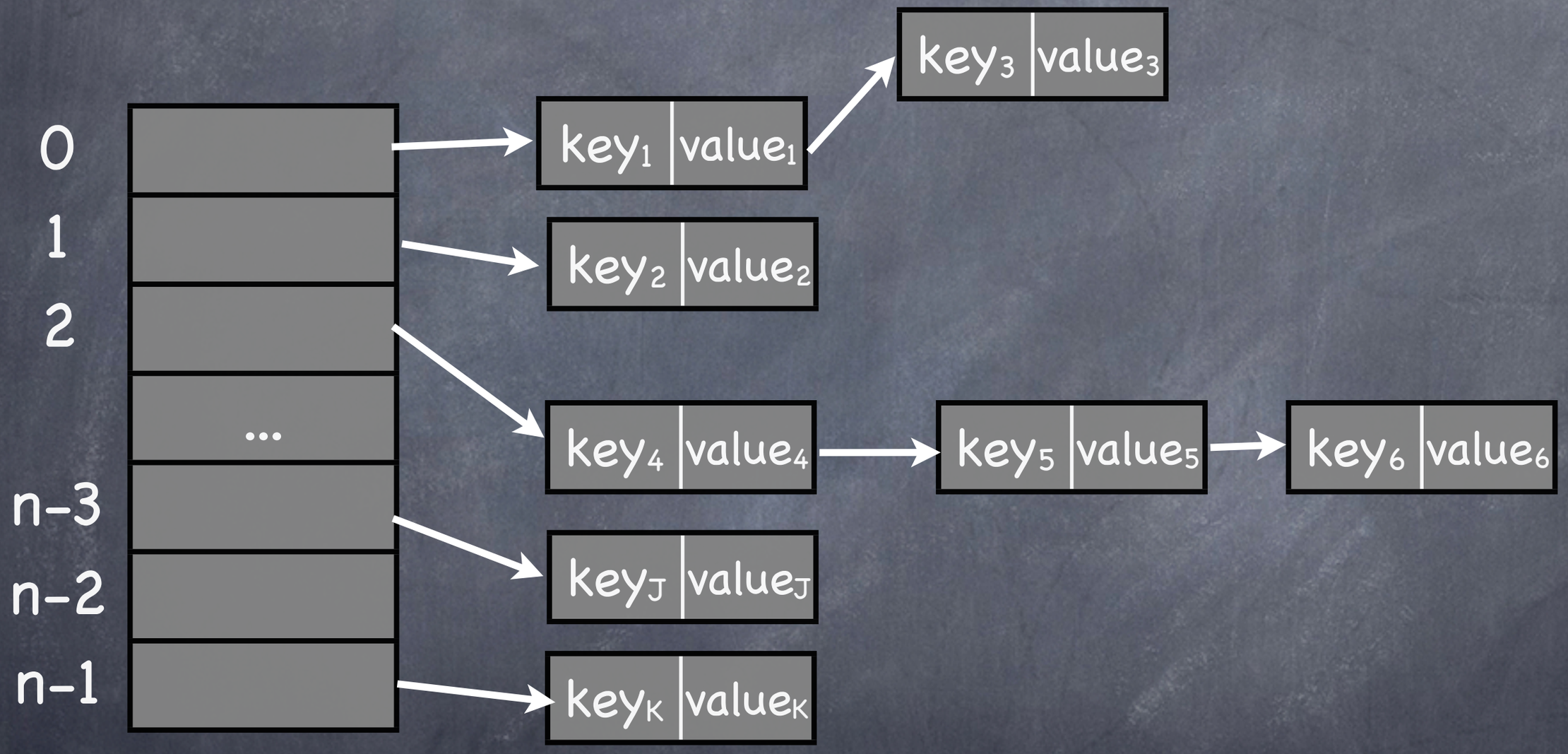
byte number	byte address	word address	cache block number	page number
0	0	0	0	0
4	4	1	1	0
16	16	4	4	0
1024	1024	256	256	0
8196	8196	2048	2048	2

Memory Units and Addresses

	byte	word	cache block	page
Size in bytes	1	100 = 4	10000 = 16	1000000000 = 1024

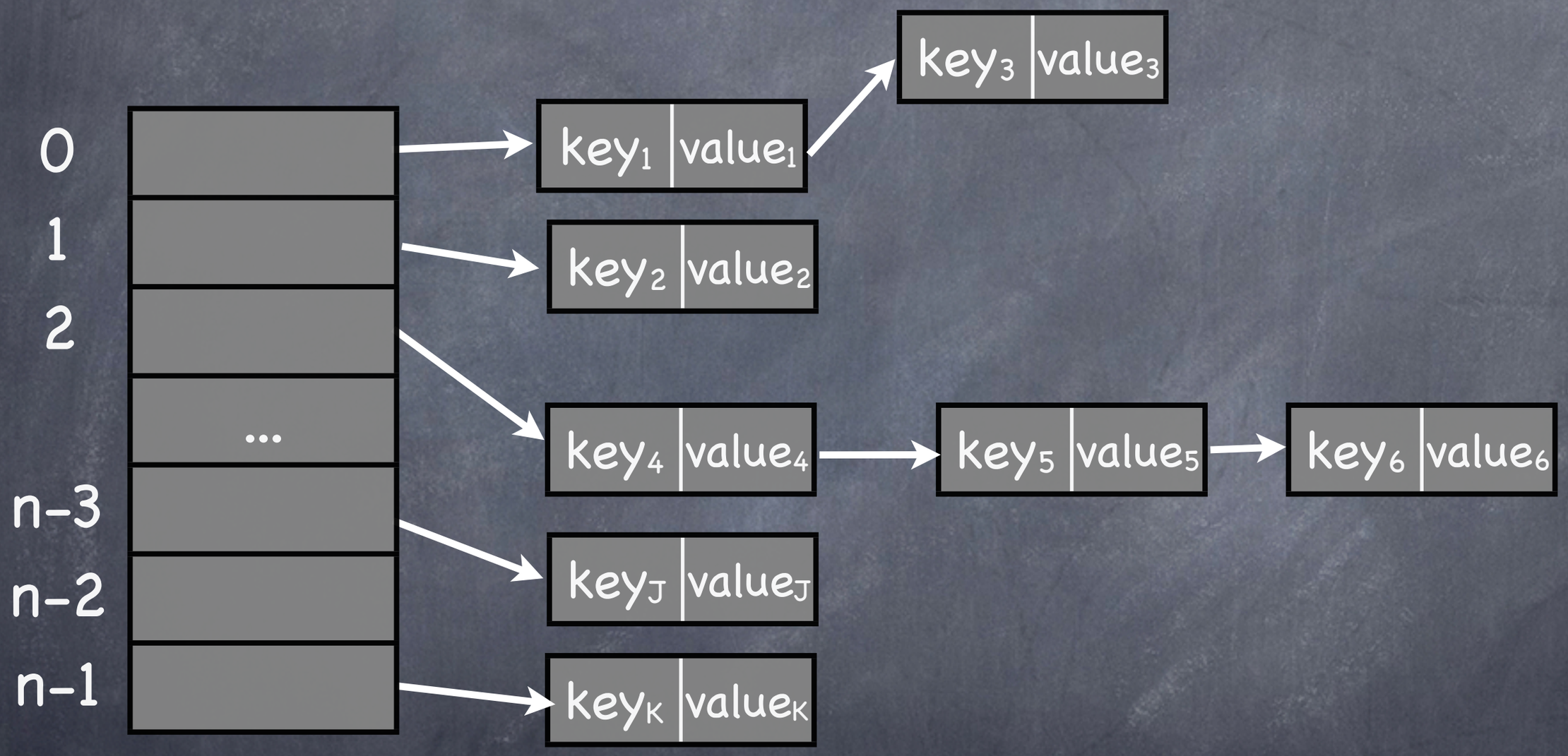
byte number	byte address	word address	cache block number	page number
0000000100	0000000100	00000001	000000	0
0000010000	0000010000	00000100	000001	0
1000000000	1000000000	10000000	100000	1

Hash vs. Cache



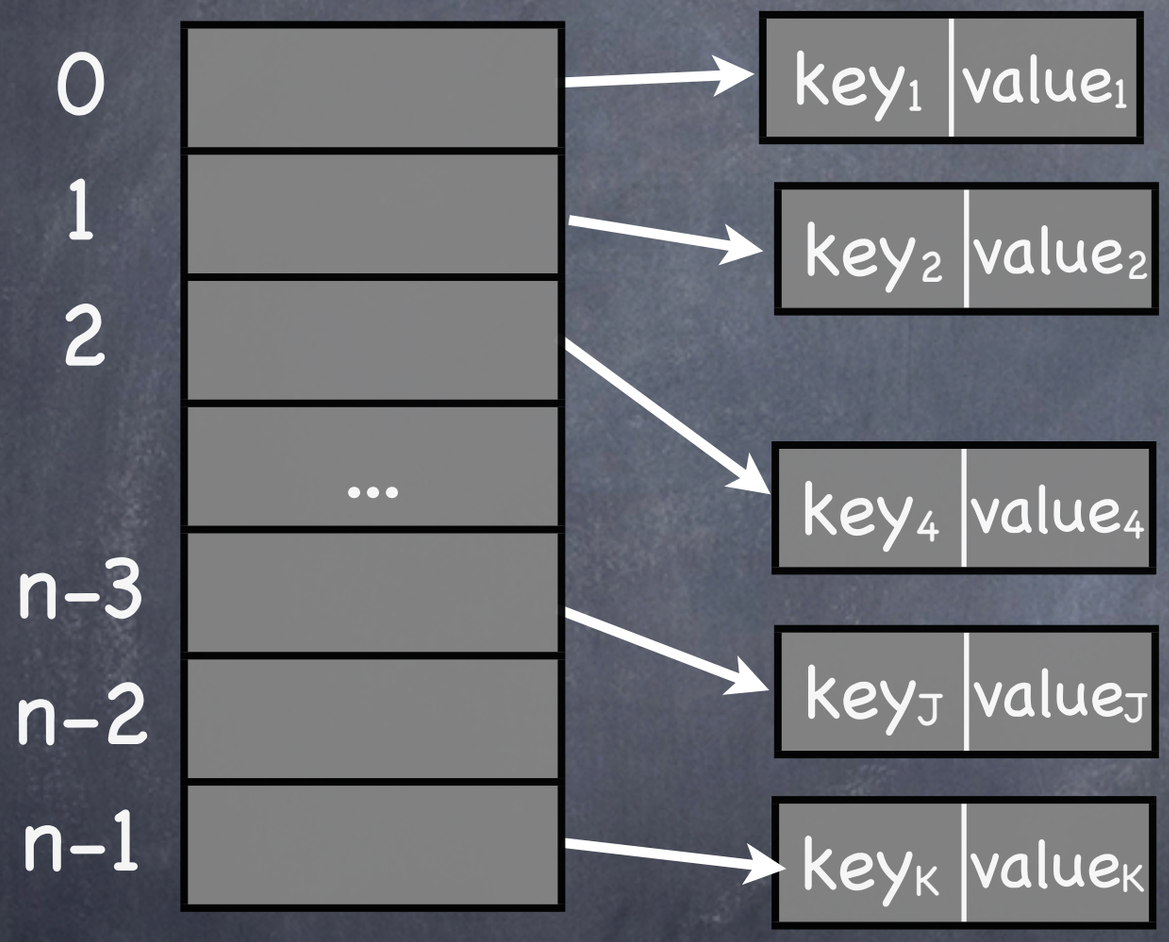
- Hash function maps keys to ``random'' bins
- Lookup within bins by linked list traversal

Forgetful Hash vs. Cache



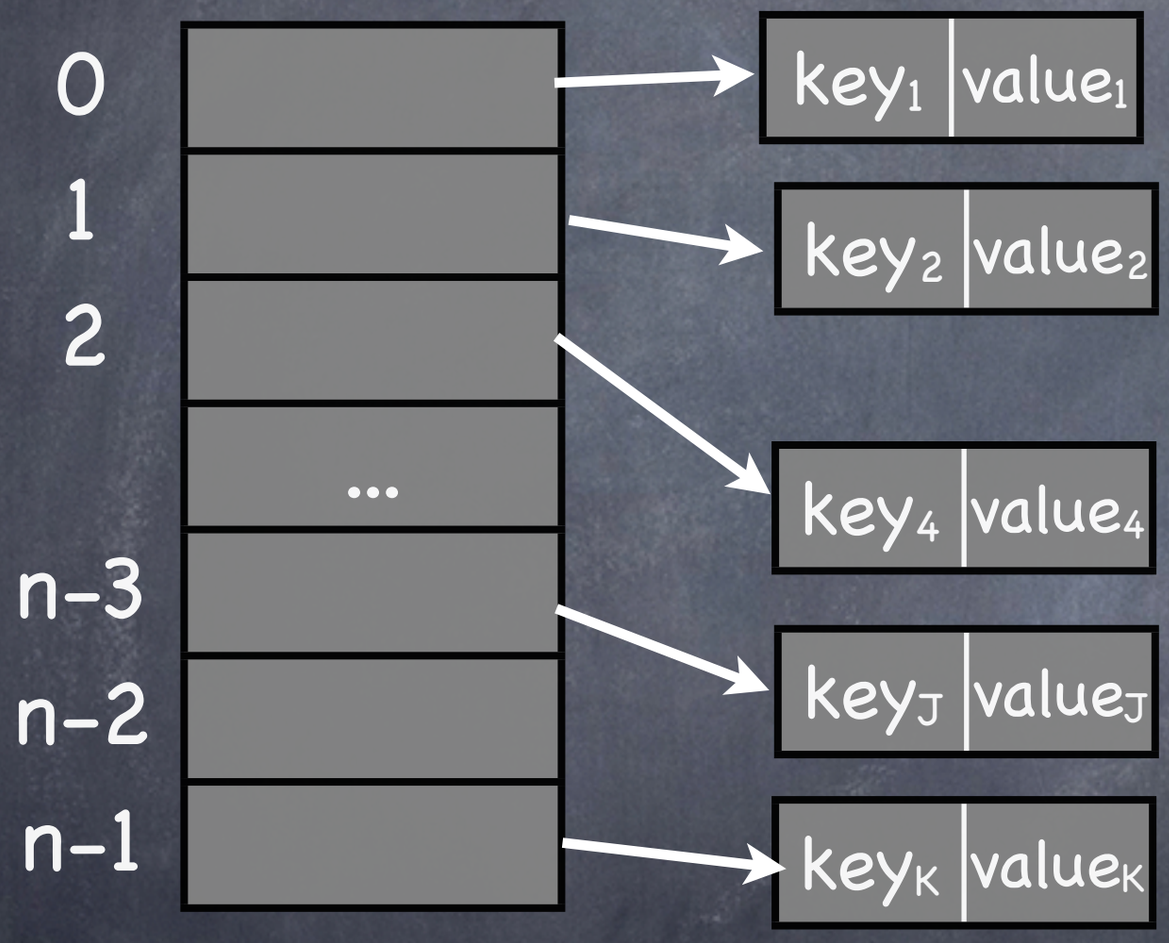
- Only keep first item in each hash bin list

Forgetful Hash vs. Cache



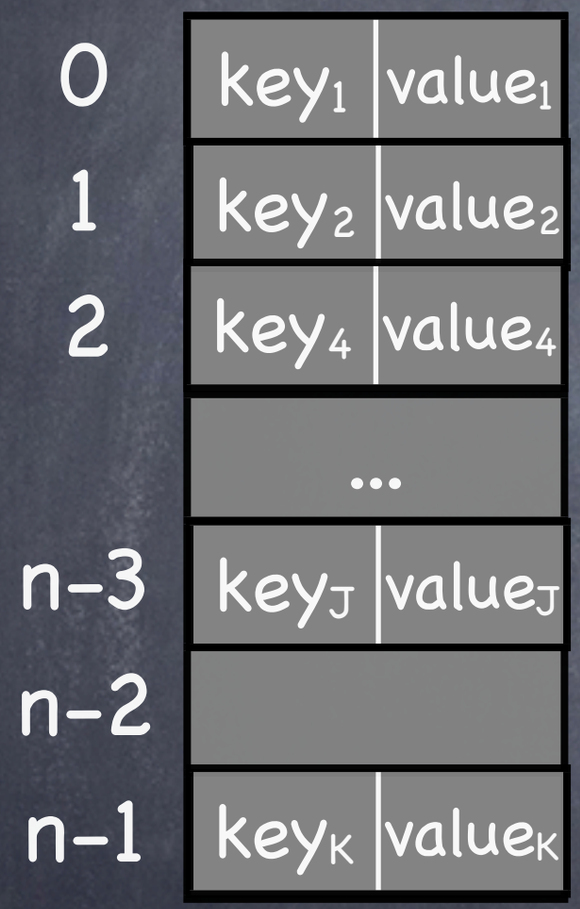
- Only keep first item in each hash bin list

Forgetful Hash vs. Cache



- No need for linked lists of size one!

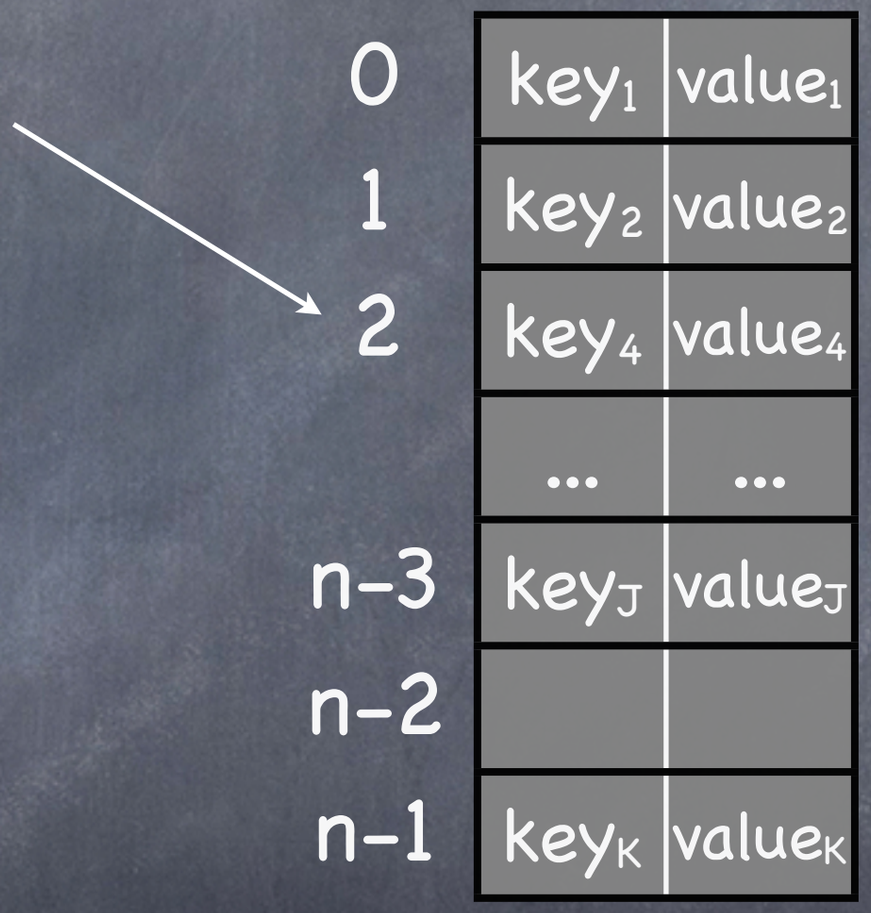
Forgetful Hash vs. Cache



- No need for linked lists of size one!

Forgetful Hash vs. Cache

bin = hash(key_{search})

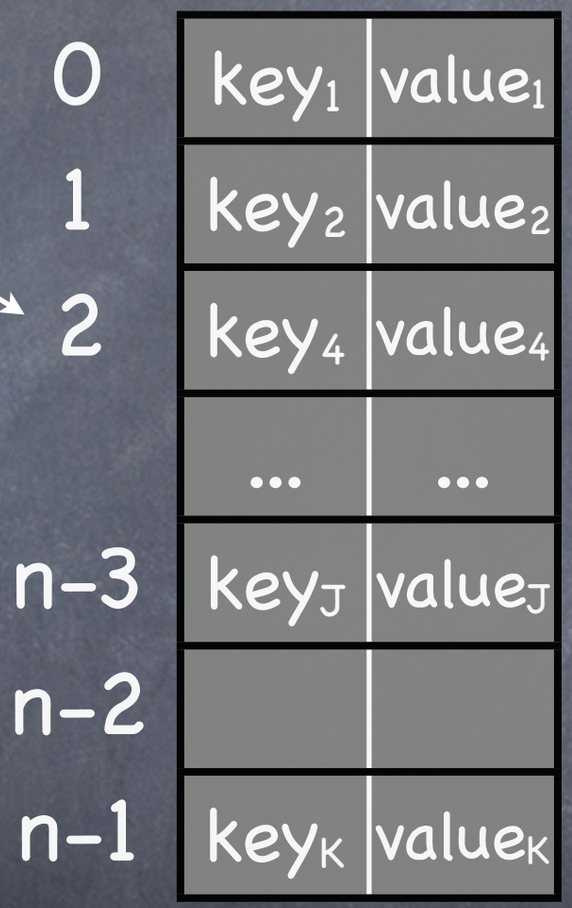


- To find a particular key, first hash the key

Forgetful Hash vs. Cache

```
bin = hash( key_search )

if ( key_bin == key_search ) {
  hit = true;
  value = value_bin;
} else {
  hit = false;
}
```

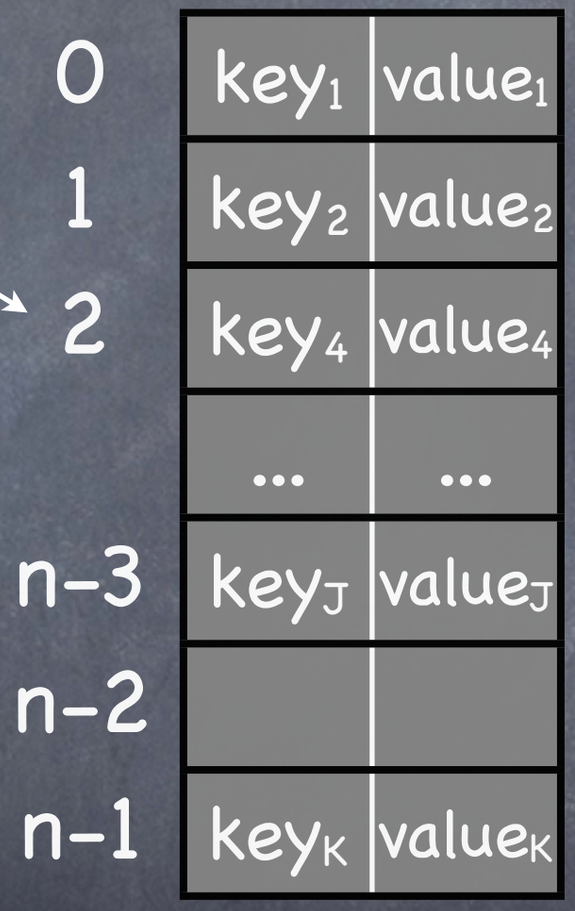


- Check key in bin to determine whe

Forgetful Hash vs. Cache

bin = hash(key_{search})

hash(v) = v % n

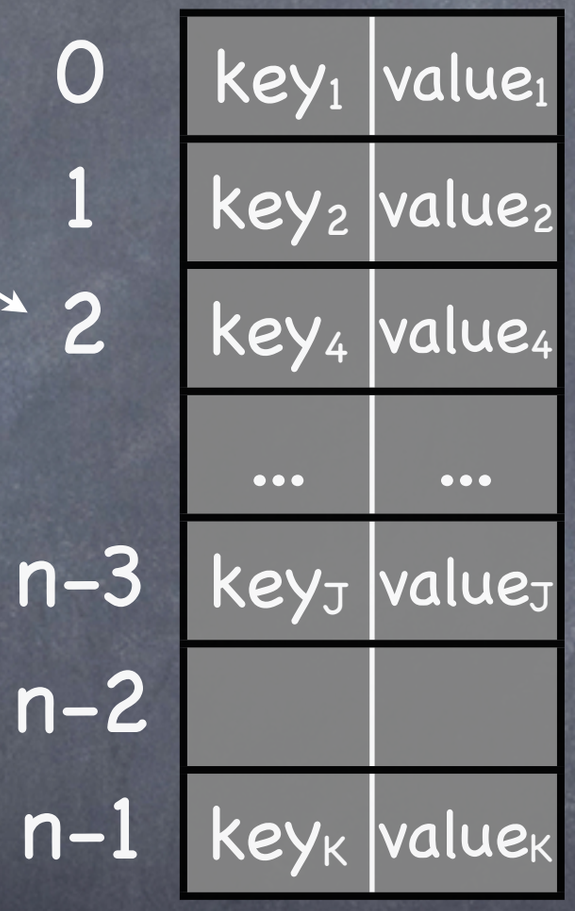


- Everyone's favorite hash function?

Forgetful Hash vs. Cache

bin = hash(key_{search})

hash(v) = v % 2^n



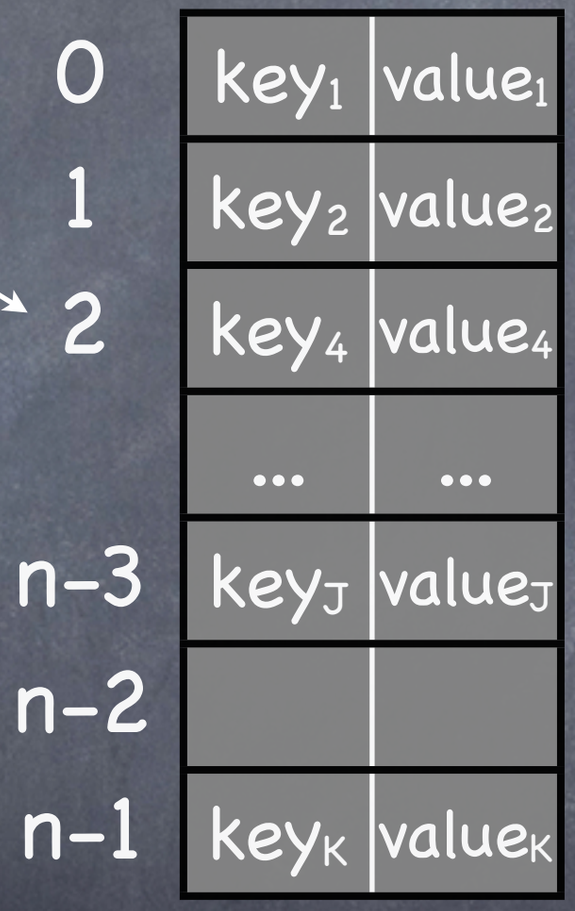
- Everyone's favorite hash function!

Forgetful Hash vs. Cache

bin = hash(key_{search})

hash(v) = v % 2^n

= last n bits of v



- Everyone's favorite hash function!

Cache

bin = last n bits of
cache block number

hash(v) = v % 2^n
= last n bits of v

0	cache blk num_1	blk $contents_1$
1	cache blk num_2	blk $contents_2$
2	cache blk num_4	blk $contents_4$
	...	...
n-3	cache blk num_J	blk $contents_J$
n-2		
n-1	cache blk num_K	blk $contents_K$

- In cache, key = block number, value = contents

Saving a few bits

bin = blk num % 2^n

0	cache blk num_1 /2^n	blk $contents_1$
1	cache blk num_2 /2^n	blk $contents_2$
2	cache blk num_4 /2^n	blk $contents_4$
	...	...
n-3	cache blk num_J /2^n	blk $contents_J$
n-2		
n-1	cache blk num_K /2^n	blk $contents_K$

- bin number = last n bits of blk num

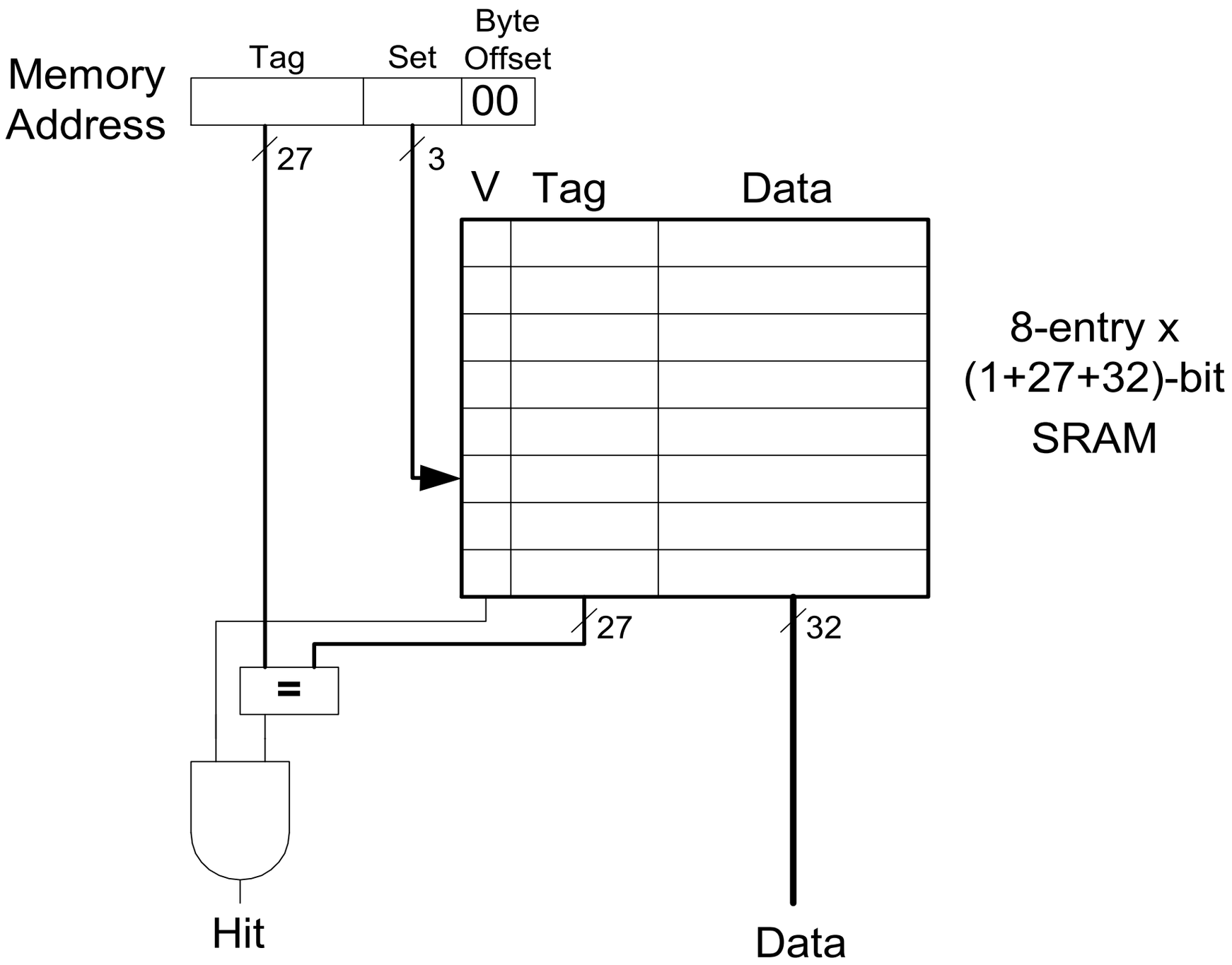
Valid bits

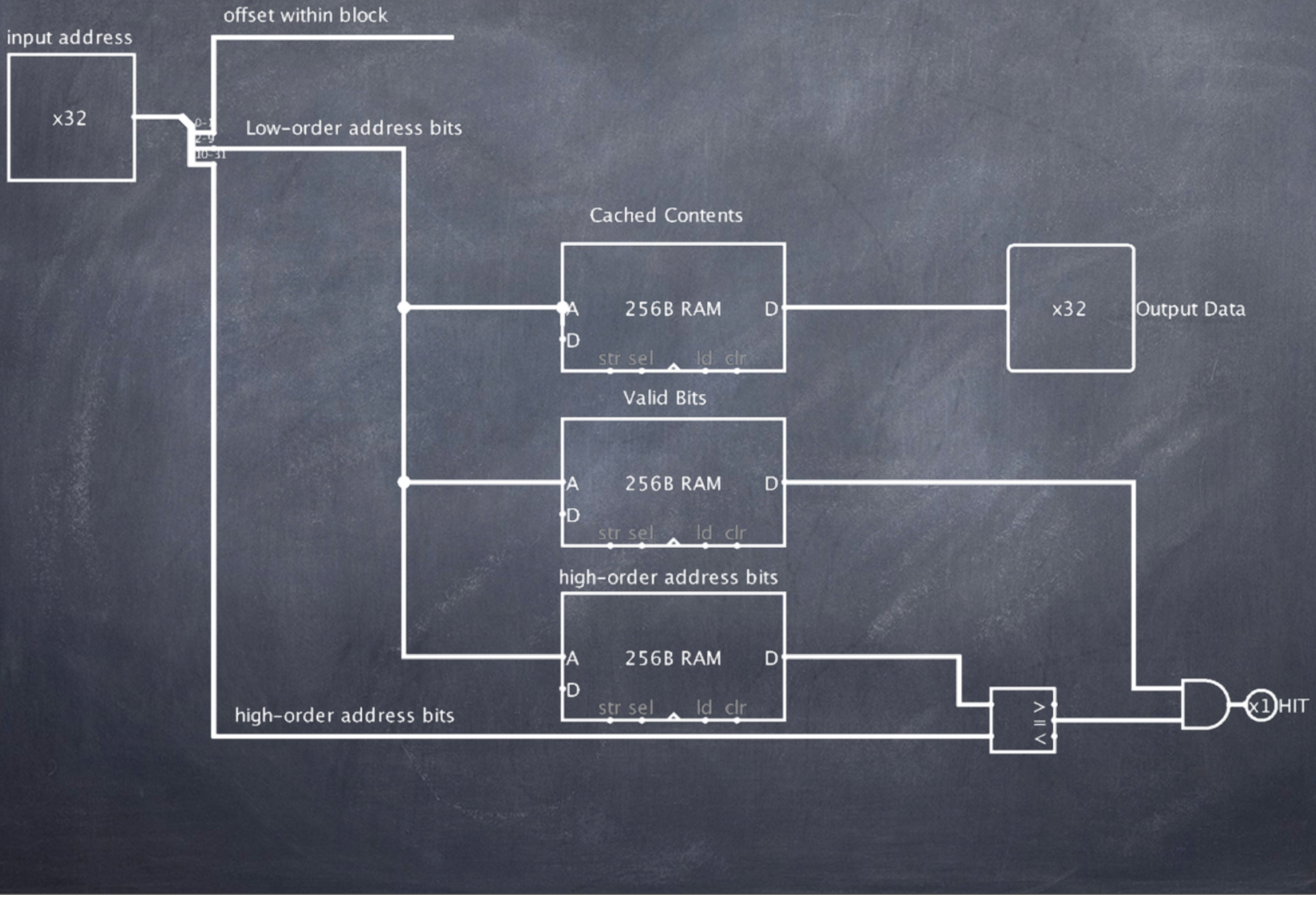
bin = blk num % 2^n

0	x	cache blk num_1 /2^n	blk $contents_1$
1	x	cache blk num_2 /2^n	blk $contents_2$
2	x	cache blk num_4 /2^n	blk $contents_4$
	–	…	…
n-3	x	cache blk num_J /2^n	blk $contents_J$
n-2	–		
n-1	x	cache blk num_K /2^n	blk $contents_K$

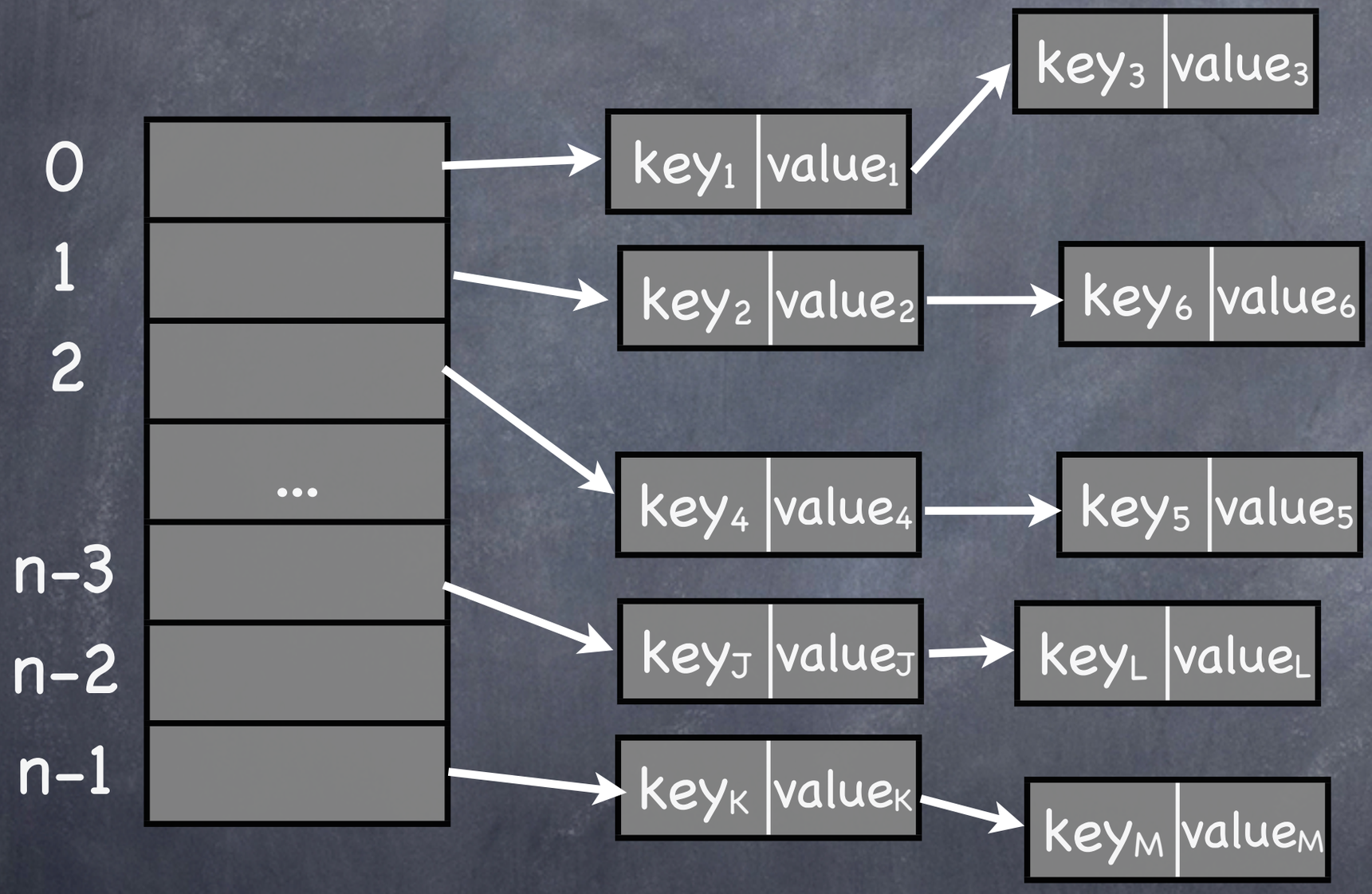
- Valid bit marks empty cache entries

Direct Mapped Cache Hardware



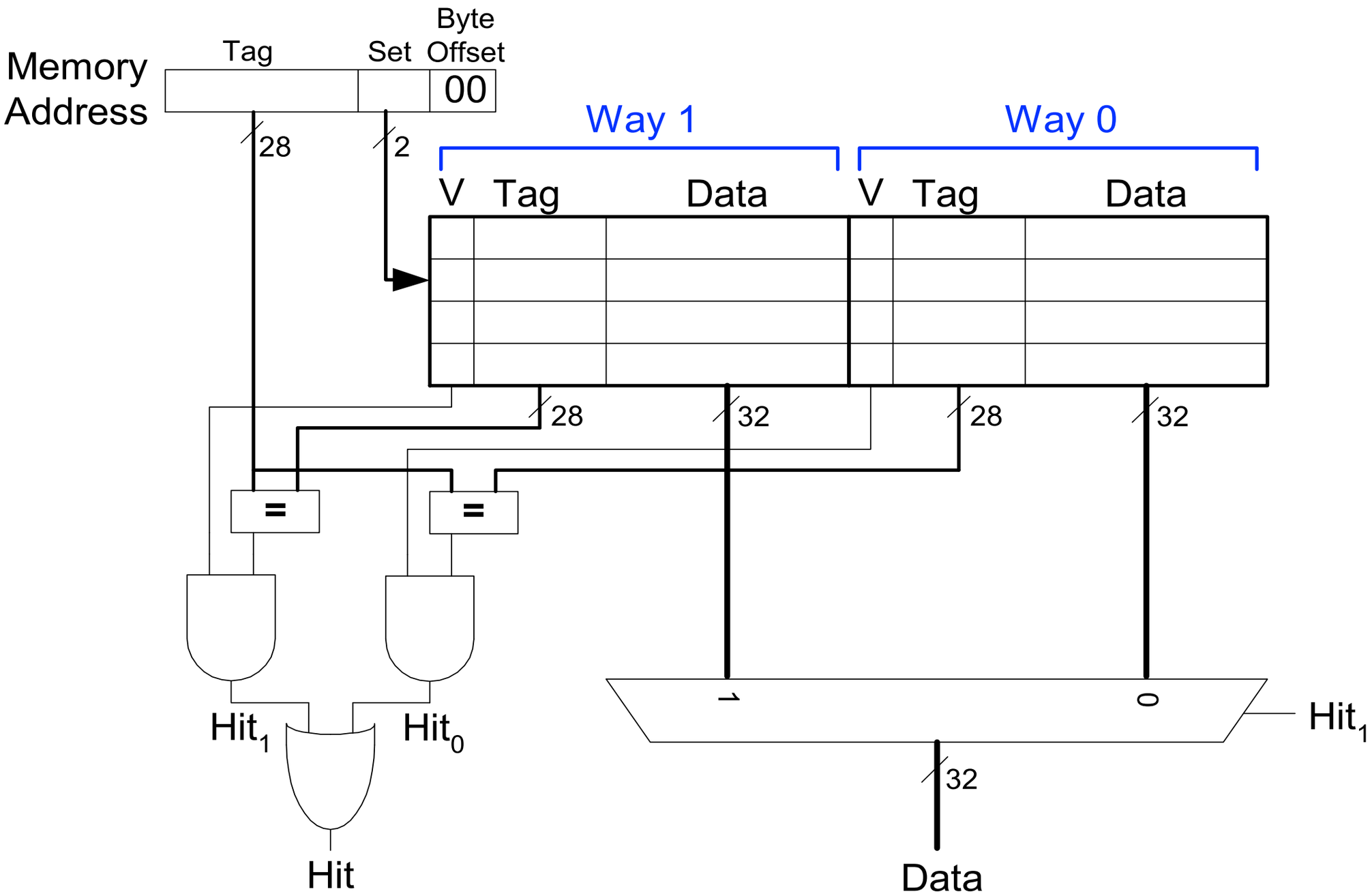

input address
x32
offset within block
0-1
2-9
10-31
Low-order address bits
Cached Contents
A
D
256B RAM
D
str sel
ld clr
x32
Output Data
Valid Bits
A
D
256B RAM
D
str sel
ld clr
high-order address bits
A
D
256B RAM
D
str sel
ld clr
high-order address bits
>
=
<
x1
HIT

Hash vs. Cache



- Suppose each hash chain is limited to the same, small size (2?)

N-Way Set Associative Cache



Cache Terminology

Capacity (C):

the number of data bytes a cache stores

Block/Line size (b):

bytes of data brought into cache at once

Number of blocks/lines ($B = C/b$):

number of blocks in cache: $B = C/b$

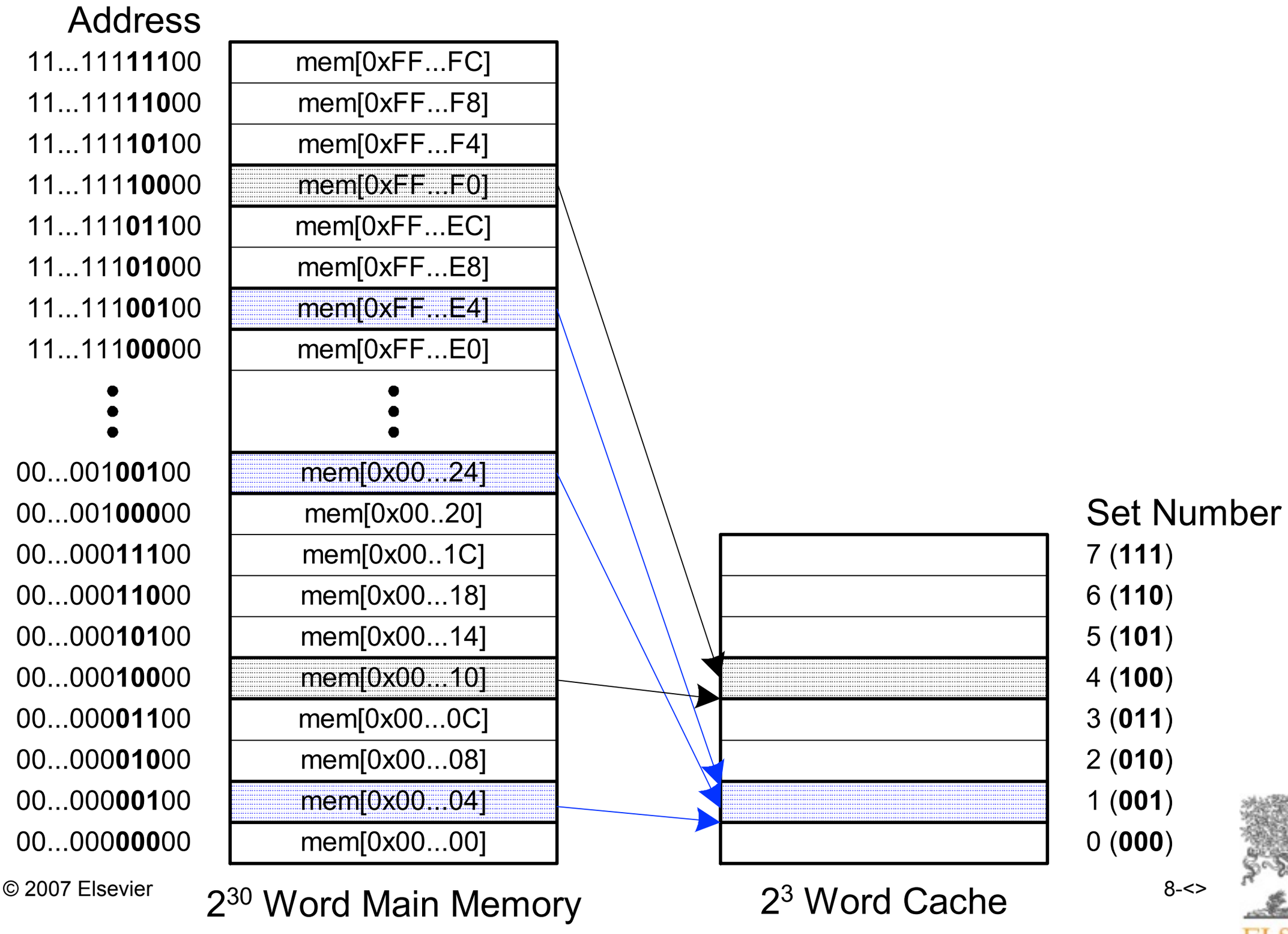
Degree of associativity (N):

number of blocks/lines in a set

Number of sets ($S = B/N$):

each memory address maps to exactly one cache set

Direct Mapped Cache



(Partial) Loop Unrolling

```
for ( i = 0; i < 1000; i++ ) {
      a[i] = 0;
}
```

7 instruction loop

```
      la  $t0,a
#     for(int i=0;i<1000;i++)
      li  $t3,0
      li  $t4,1000
zeroloop:
      slt $t5,$t3,$t4
      beq   $t5,$0,zeroloopEnd

#     a[i] = 0;
      add   $t6,$t3,$t0
      sw $0,0($t6)



      addi  $t3,$t3,1
      b  zeroloop:
zeroloopEnd:
```

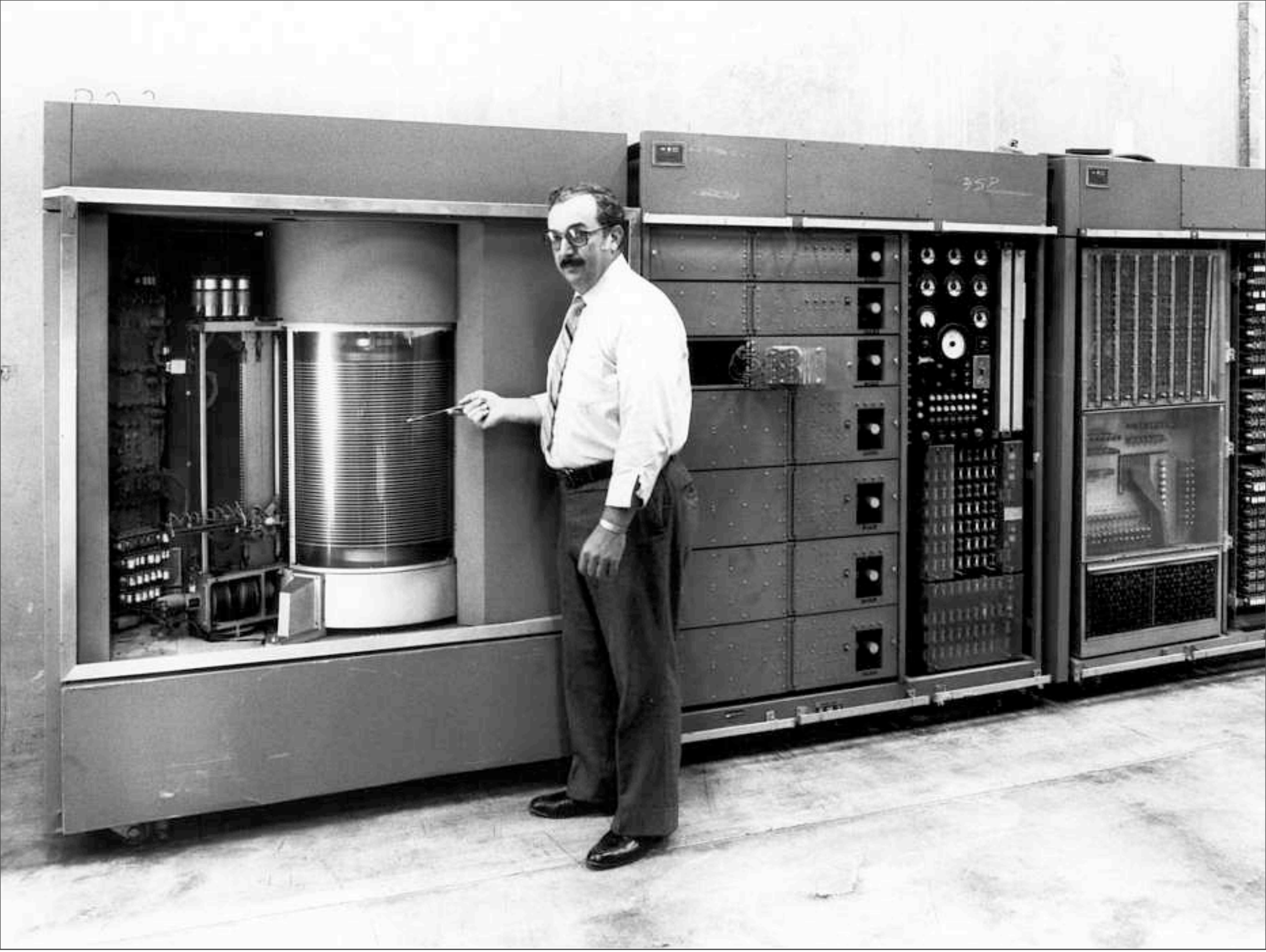
(Partial) Loop Unrolling

```
for ( i = 0; i < 1000; i = i + 4 ) {
    a[i] = 0;
    a[i+1] = 0;
    a[i+2] = 0;
    a[i+3] = 0;
}
```

9 instruction loop

```
    la  $t0,a
#   for(int i=0;i<1000;i=i+2)
    li  $t3,0
    li  $t4,1000
zeroloop2:
    slt $t5,$t3,$t4
    beq  $t5,$0,zeroloop2End

#   a[i] = 0;
    add  $t6,$t3,$t0
    sw $0,0($t6)
    sw $0,4($t6)
    sw $0,8($t6)
   sw $0,12($t6)
    addi  $t3,$t3,4
    b  zeroloop2:
zeroloopEnd2:
```

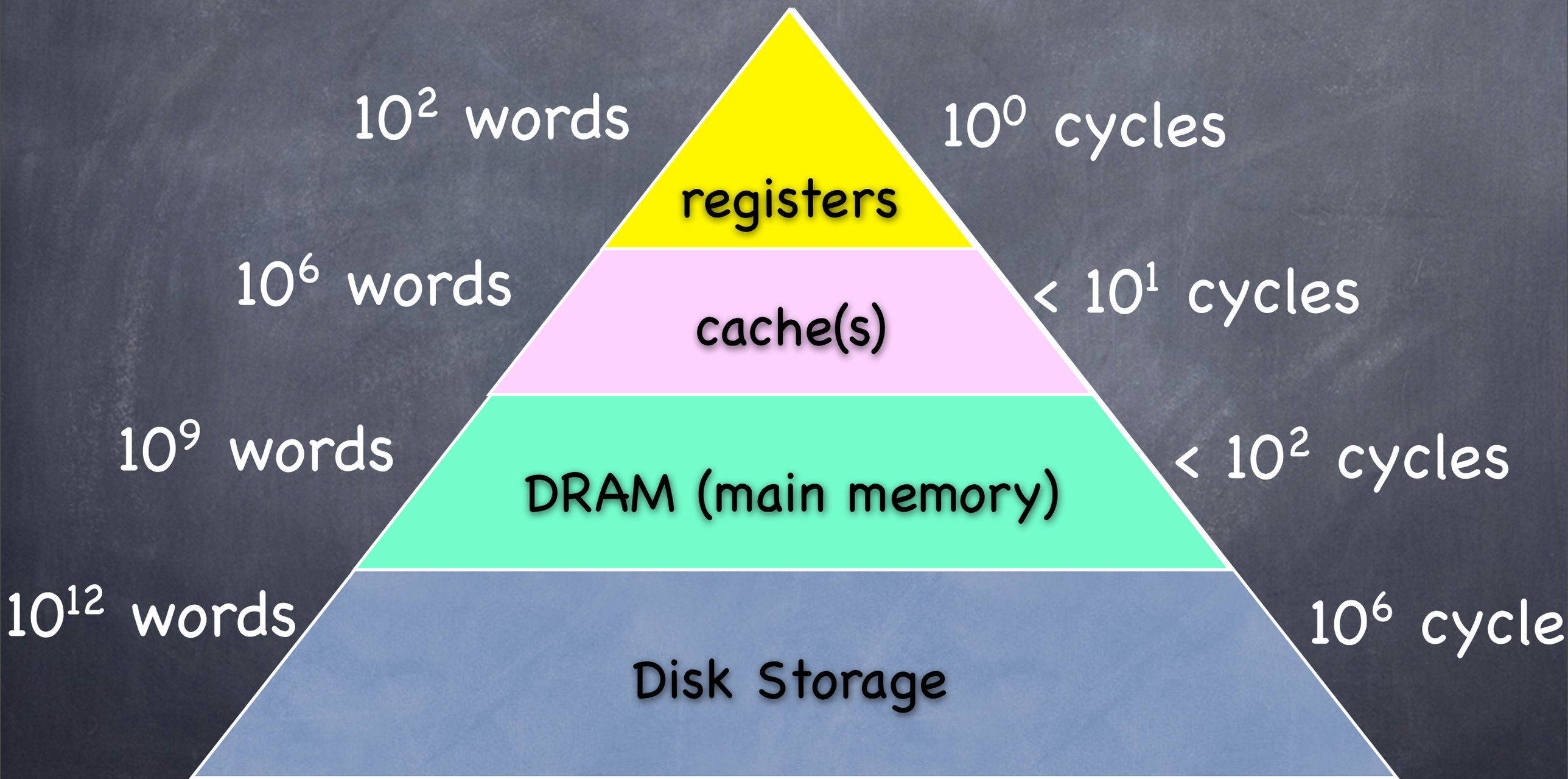


Tuesday, November 27, 12


2311 Disk
Storage Drives

Tuesday, November 27, 12

Virtual Memory?



Cache Miss Penalty

	Cache	DRAM	Disk
	5	25	1000000
0.5	3	13	500001
0.8	1.8	5.8	200001
0.9	1.4	3.4	100001
0.999	1.00	1.02	1001
0.99999	1	1	11
0.999999	1	1	2

Access time in cycles

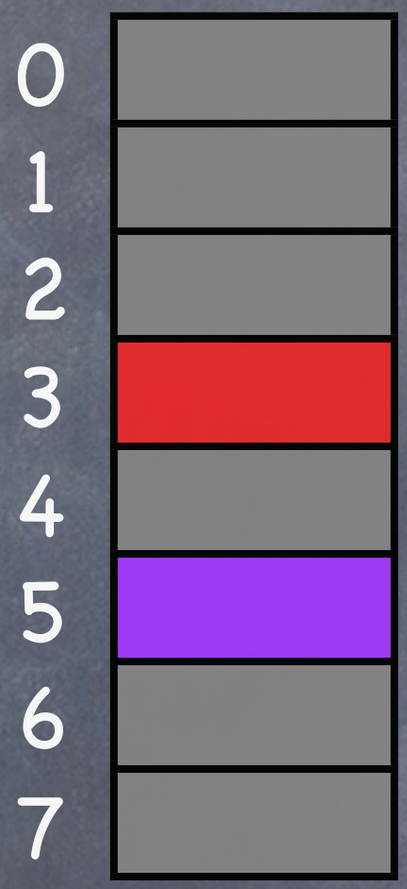
Hit Rate

Effective Memory Access Times

Effective = 1*hit-rate + access-time*(1 - hit-rate)

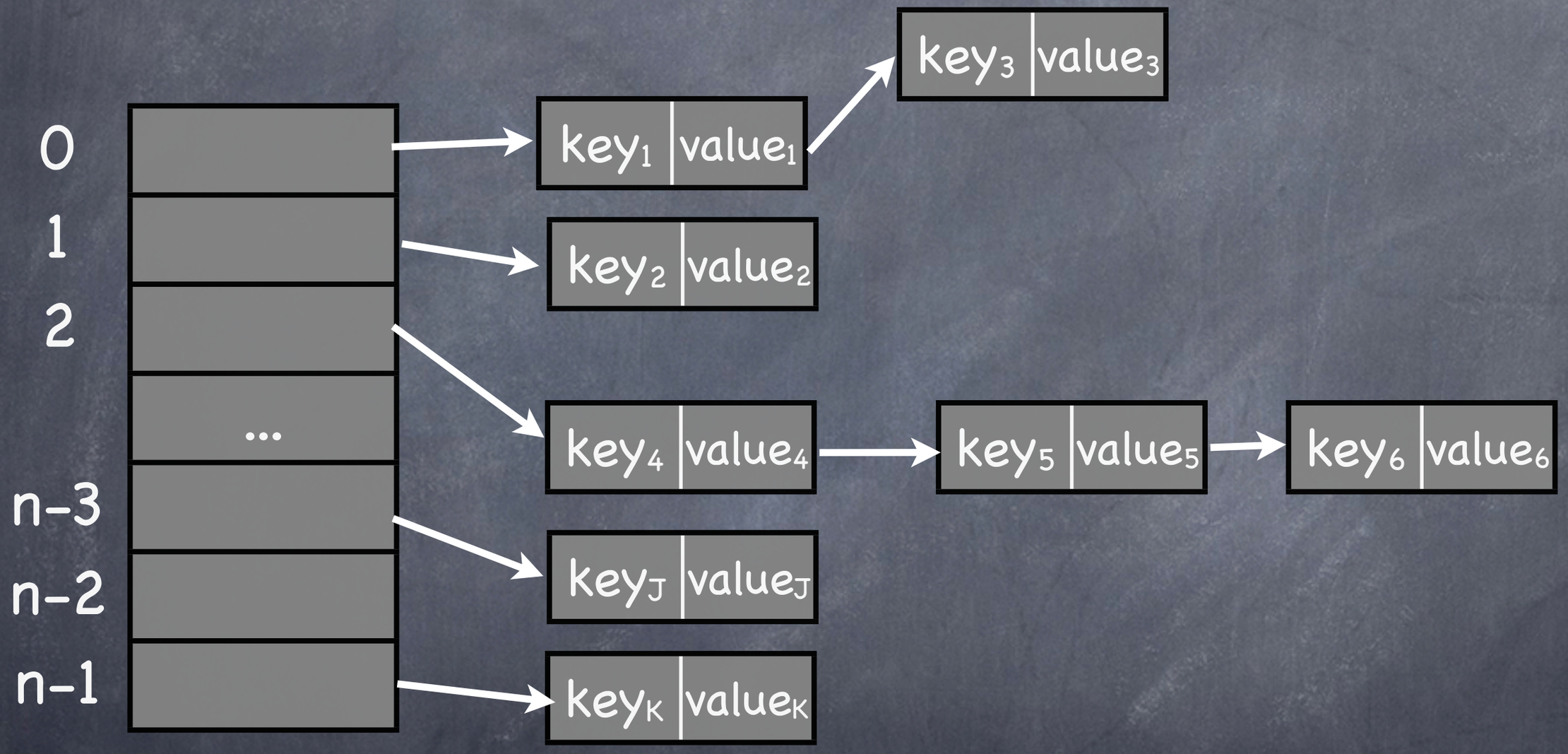
Program 1's Virtual Addr Space

Program 2's Virtual Addr Space



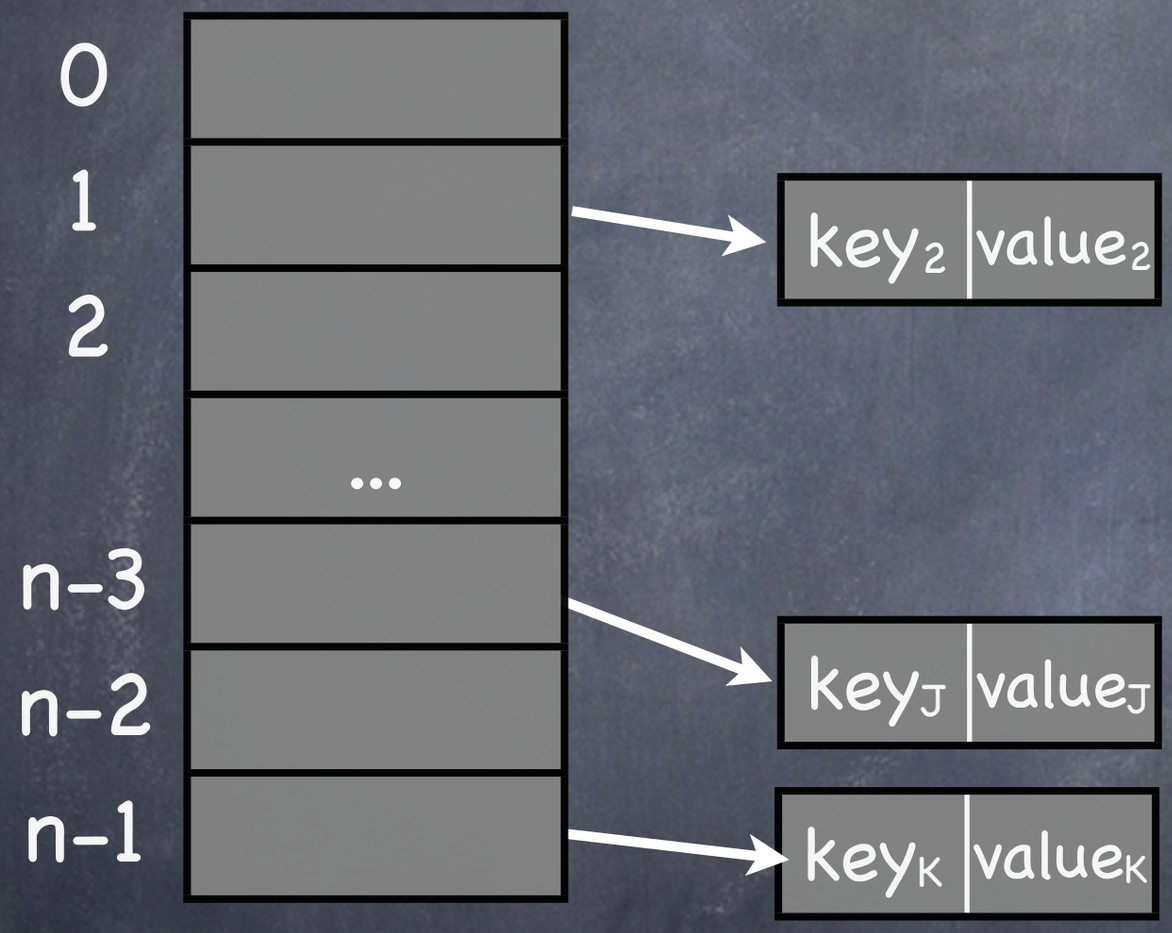
Real/Physical Memory

Hash vs. Page Table



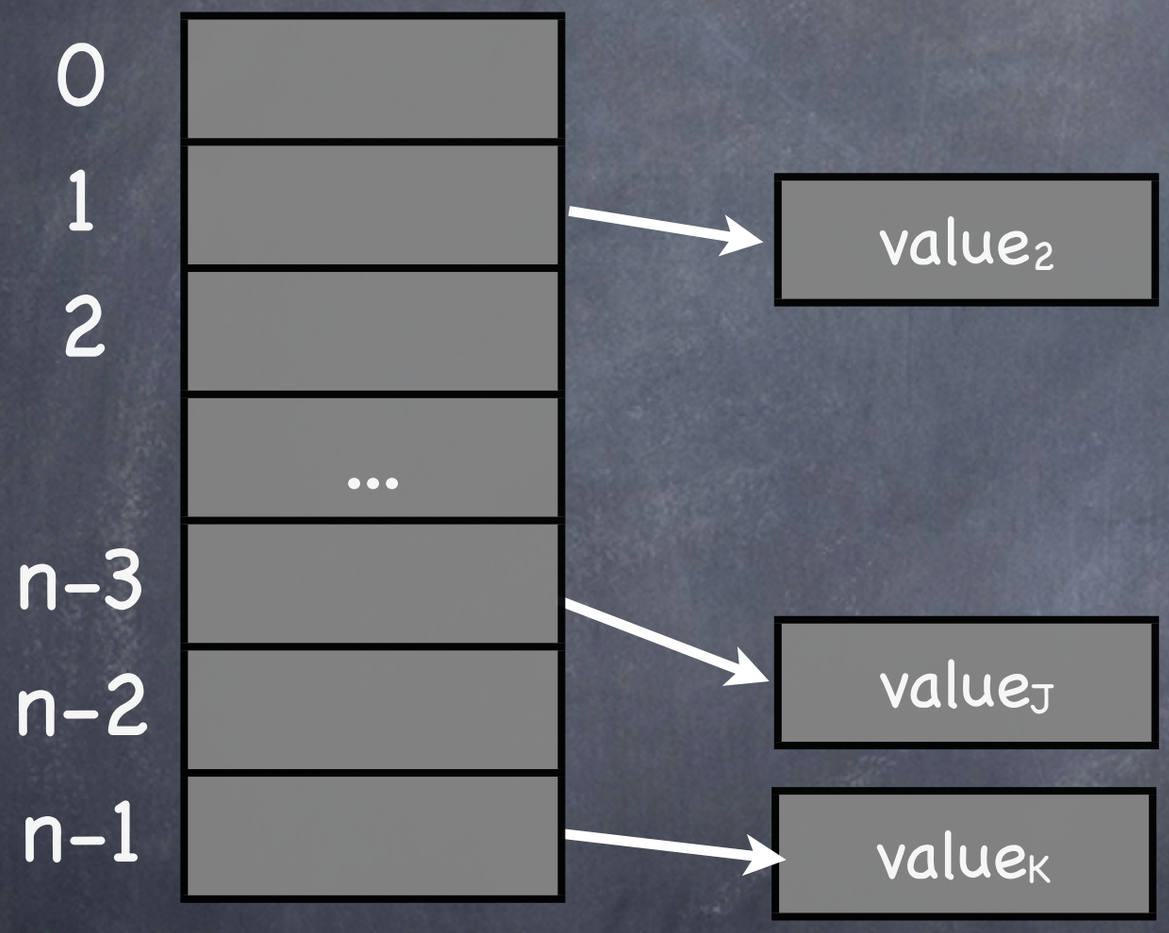
- Can't afford to traverse chains
- Can't afford to discard entries!

Sparse Hash = Page Table



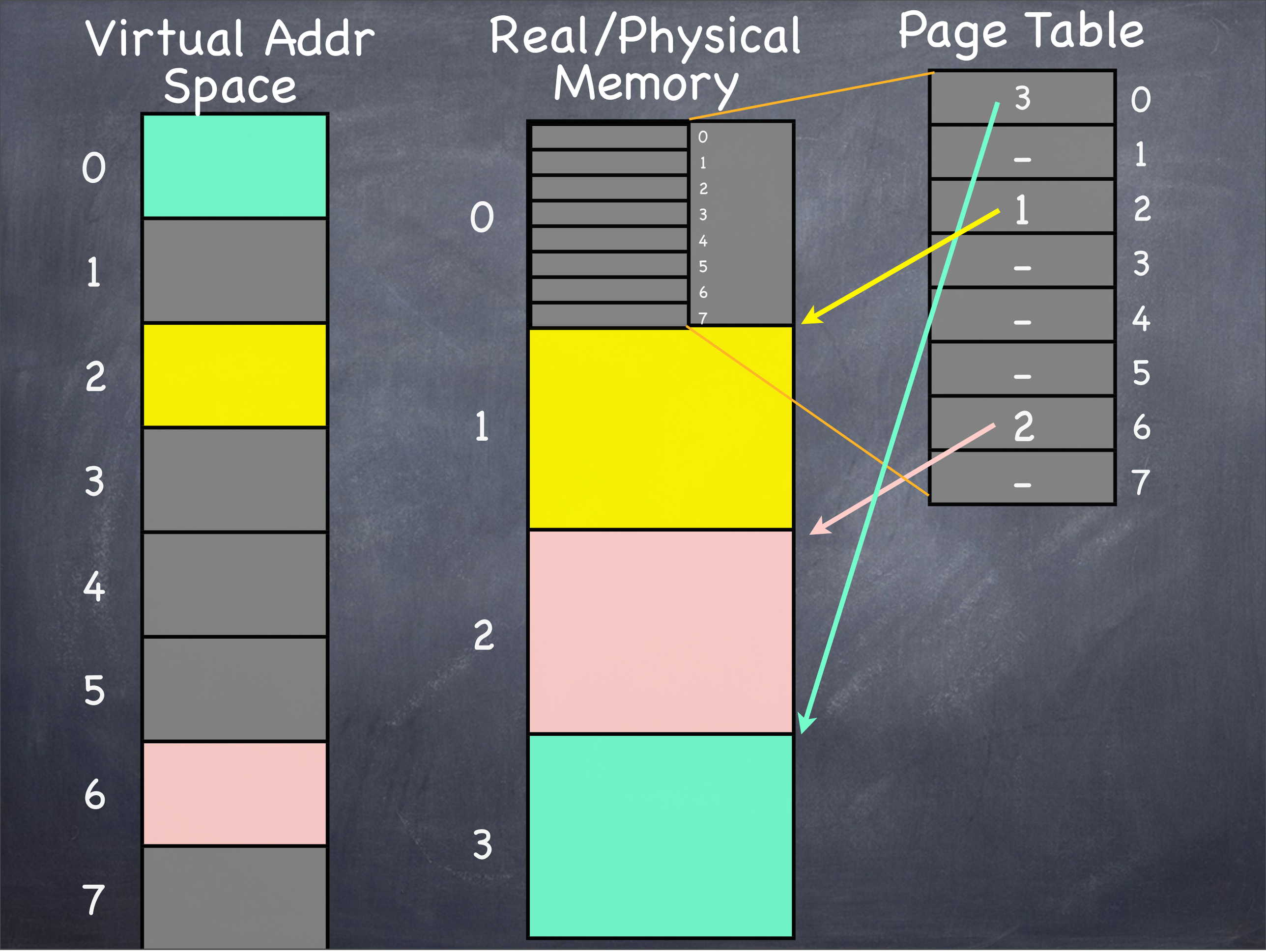
- Let n = number of pages in address space

Sparse Hash = Page Table



hash(x) = x !

- "Values" in key/value pairs of page table are huge. Can't afford to put them inline.


Virtual Addr Space
0
1
2
3
4
5
6
7
Real/Physical Memory
0
1
2
3
0
1
2
3
4
5
6
7
Page Table
3
–
1
–
–
–
2
–
0
1
2
3
4
5
6
7