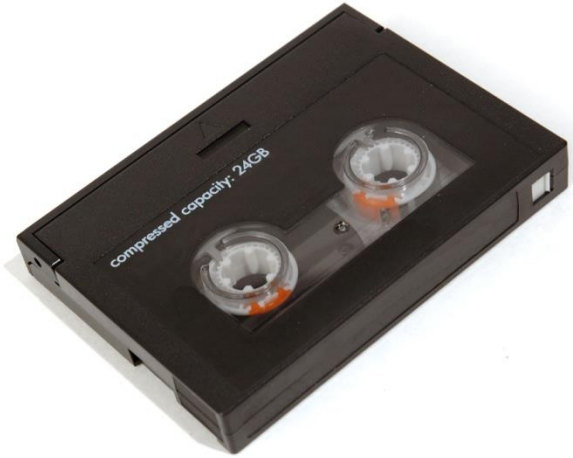
cse5441 - parallel computing

cache management

http://frostburg.collegestoreonline.com/webitemimages/326/Images/zzzzzCash%20For%20Books.jpg

is all memory the same?

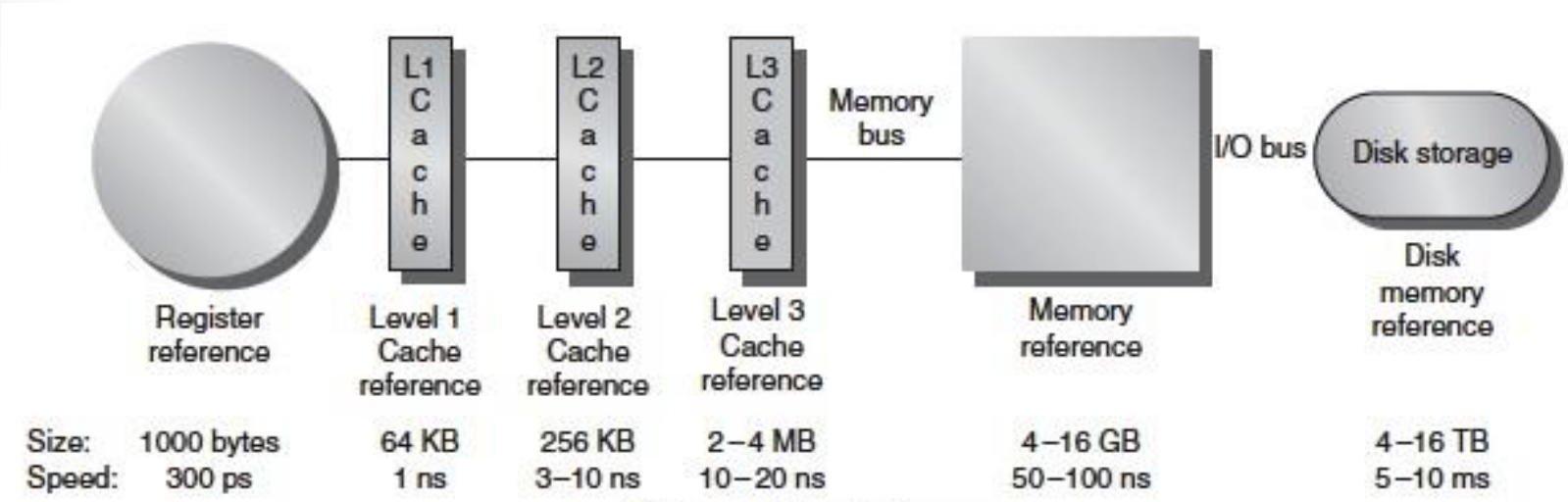


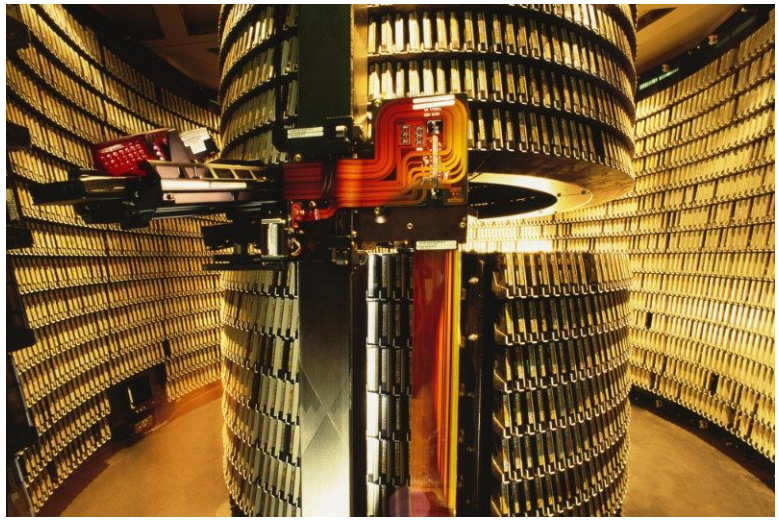
http://s.hswstatic.com/gif/adding-a-hard-disk-1-1.jpg
http://ebay.18004memory.com/ebaypics/FPM/SIMM/029A-30X2.gif
http://www.backupassist.com/blog/wp-content/uploads/2013/04/iStock_000000393598_L3.jpg
http://images2.wikia.nocookie.net/__cb20091129162961/uncyclopedia/images/0/08/5.25_inch_MFM_hard_disk_drive.JPG

what is cache?

- something precious preserved or concealed in a convenient but private place
- a computer memory with very short access time used for storage of frequently or recently used instructions or data
- a small, fast subset of a larger collection, intended to provide faster average access to the larger whole.

cache hierarchy (logical)



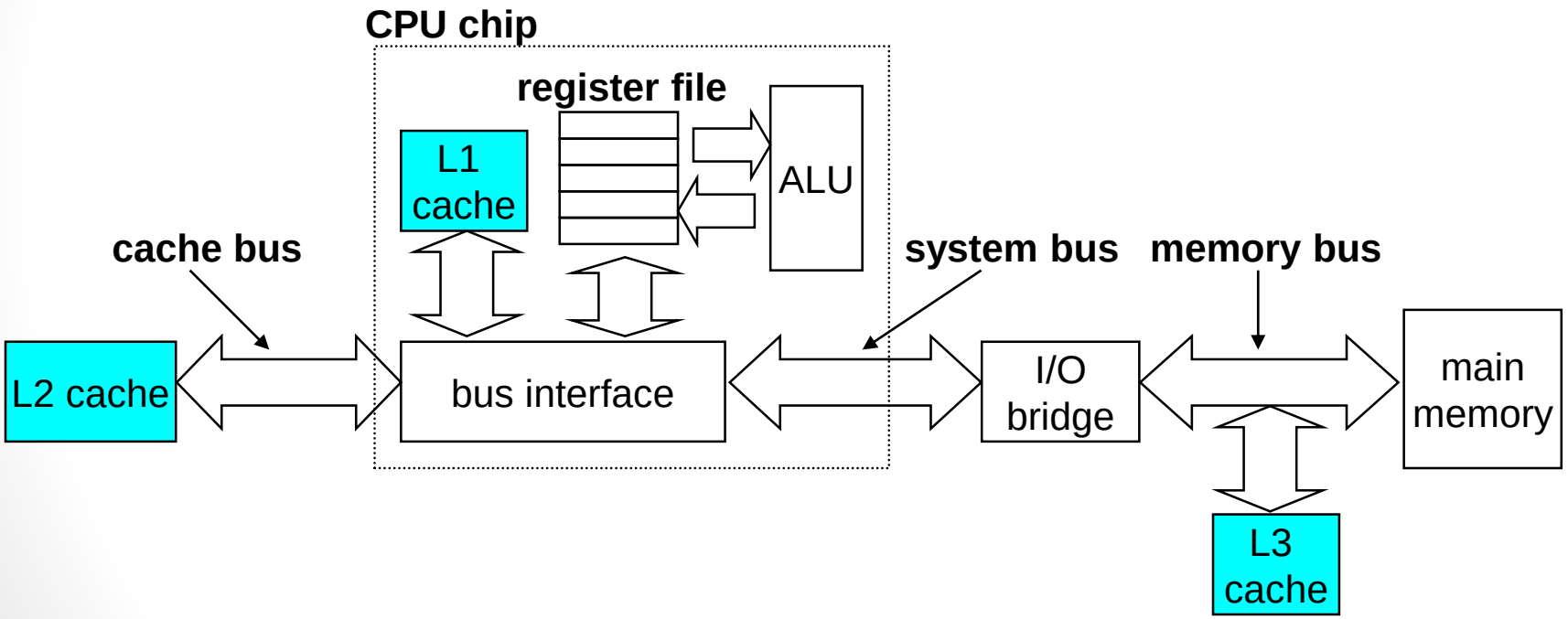


tape archival
multi-PB
seconds to minutes

http://m.eet.com/media/1171199/fig2-1.jpg
http://s7.computerhistory.org/is/image/CHM/500004392-03-01?$re-medium$

cache hierarchy (physical)

example



R. Bryant, CMU

how cache works

an example

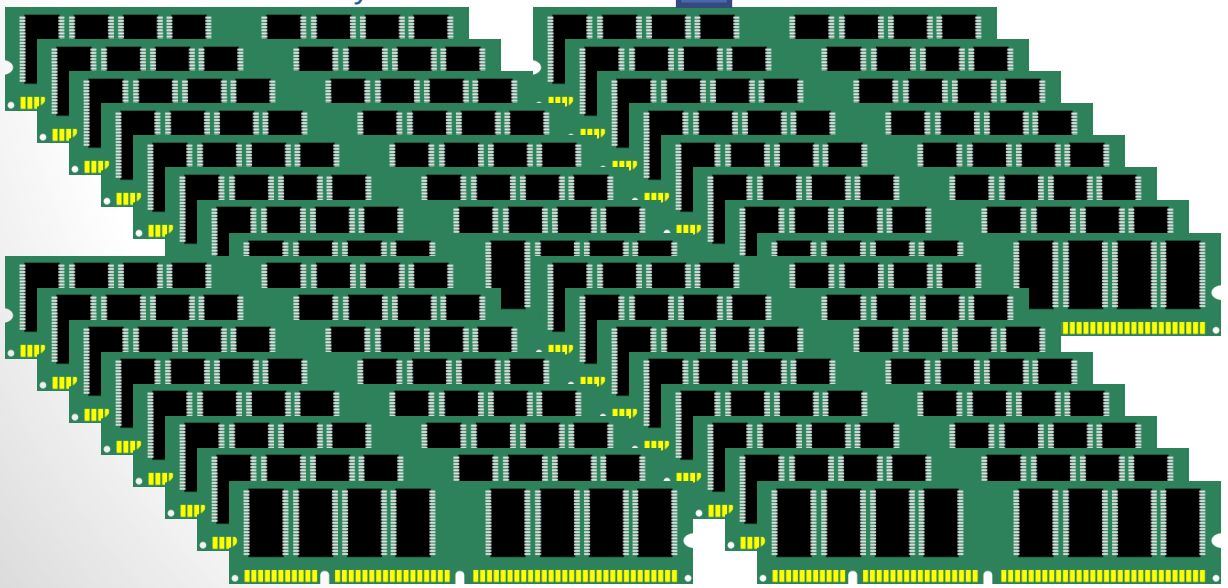
Data stored in registers are immediately available to the CPU.

This tiny, very fast CPU **register file** has room for 16 4-byte words.

Transfers from cache to registers take 1 clock cycle.

This small fast **L1 cache** has room for 512 64-byte blocks (32 Kb).

Transfers from main memory to registers (and cache) take 100 clock cycles..



This big slow **main memory** has room for 64 Gbytes.

http://g02.a.alicdn.com/kf/HTB1nb3sIFXXXXXPXXXXq6xXFXXXD/New-for-Intel-8G-8GB-Half-Turbo-Cache-Memory-Card-Moudle-for-IBM-Lenovo-W510-T510.jpg

locality of reference

```
sum = 0;
for (int i = 0; i < max; i++)
{
    sum += array[ i ];
}
```

- spatial locality of reference
  can we anticipate using a memory address based on where it is?

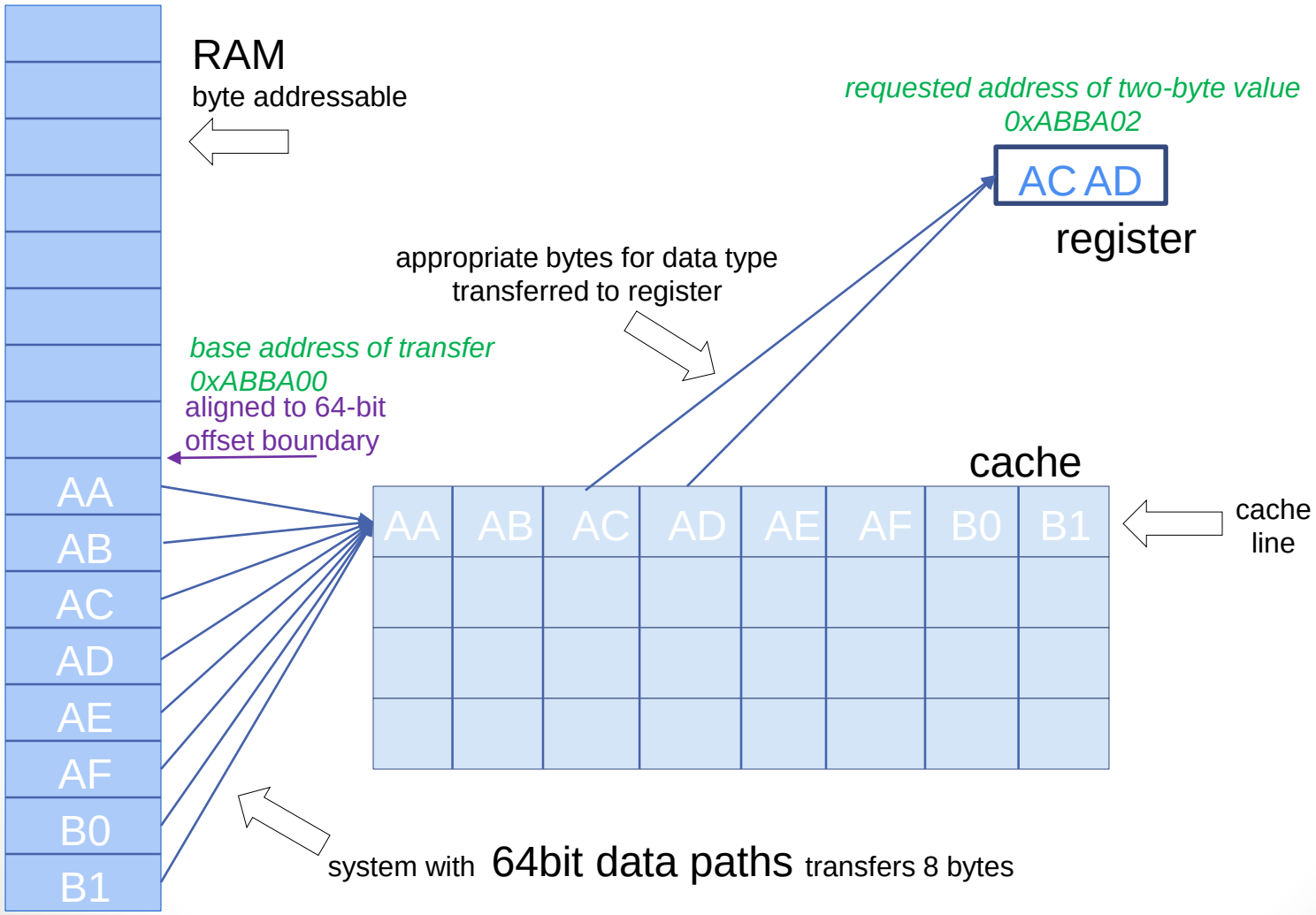
- temporal locality of reference
  can we anticipate using a memory address based on when it was last used?

an aside … register variables

```
sum = 0;
for (int i = 0; i < max; i++)
{
    sum += array[ i ];
}
```

```
R1 = 0;
for (int i = 0; i < max; i++)
{
    R1 += array[ i ];
}
sum = R1;
```

cache lines and pre-fetching



array locality (C, C++)

```
int   * array;
array = new int [max];

sum = 0;
for (int i = 0; i < max; i++)
{
    sum += array[ i ];
}
```

matrix locality (C, C++)

```
int    ** my_matrix;
my_matrix  =  new  int * [ num_rows ];

for (int i = 0; i < num_rows; i++)
{
    my_matrix[ i ]  =  new int[num_cols];
}

// set values to something interesting ...


sum = 0;
for (int i = 0; i < num_rows; i++)
{
    for (int j = 0; j < num_cols; j++)
    {
        sum += my_matrix[ i ][ j ];
    }
}
```

```
int    ** my_matrix;
my_matrix   =  new int * [ num_rows ];

my_matrix[0]    = new int [ num_rows * num_cols ]
for (int i = 1; i < num_rows; i++)
{
 my_matrix[ i ]  =  my_matrix[ i-1 ]
                              + ( num_cols * sizeof(int)] );
}

// set values to something interesting ...

sum = 0;
for (int i = 0; i < num_rows; i++)
{
    for (int j = 0; j < num_cols; j++)
    {
        sum += my_matrix[ i ][ j ];
    }
}
```

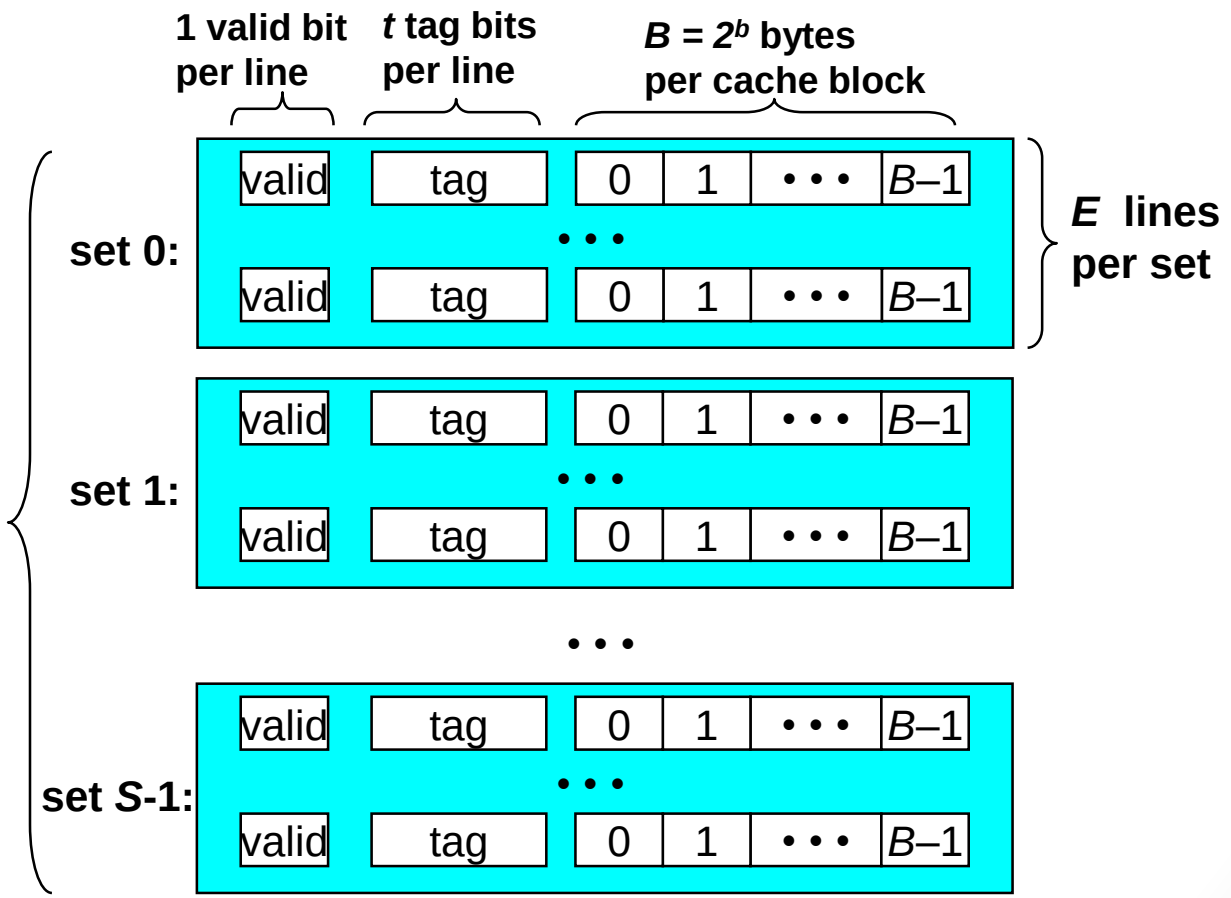
the way we declare our dynamic arrays can impact locality of reference

cache structure parameters

Cache is an array of sets.

Each set contains one or more lines.

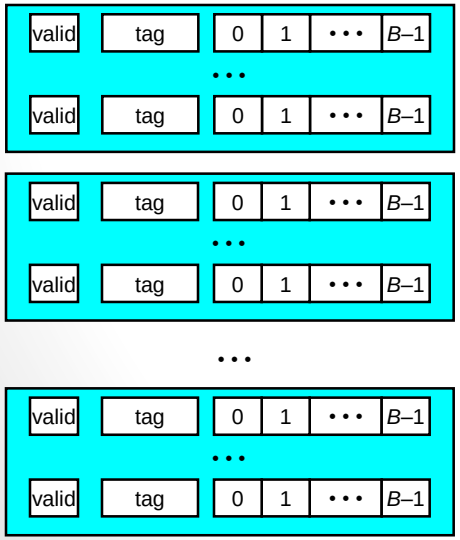
Each line holds a block of data.



usable **Cache size: $C = B \times E \times S$ data bytes**

cache structure parameters

summary



let: M = size of addressable memory
m = memory address size (bits)
$(M = 2^m)$

C = cache size (bytes) = B x E x S
B = cache block size = 2^b
E = # lines per set
S = # sets in the cache = 2^s

s = # bits for set offset
t = # tag bits
b = # bits for block offset

m = t + s + b

cache size examples

example	m	C	B	E	S	t	s	b
1	32	1024	4	1				
2	32	1024	8	4				
3	32	1024	32	32				
4	64	2048	64	16				

cache size examples

example	m	C	B	E	S	t	s	b
1	32	1024	4	1	256	22	8	2
2	32	1024	8	4				
3	32	1024	32	32				
4	64	2048	64	16				

cache size examples

example	m	C	B	E	S	t	s	b
1	32	1024	4	1	256	22	8	2
2	32	1024	8	4	32	24	5	3
3	32	1024	32	32				
4	64	2048	64	16				

cache size examples

example	m	C	B	E	S	t	s	b
1	32	1024	4	1	256	22	8	2
2	32	1024	8	4	32	24	5	3
3	32	1024	32	32	1	27	0	5
4	64	2048	64	16				

cache size examples

example	m	C	B	E	S	t	s	b
1	32	1024	4	1	256	22	8	2
2	32	1024	8	4	32	24	5	3
3	32	1024	32	32	1	27	0	5
4	64	2048	64	16	2	57	1	6

it's your turn

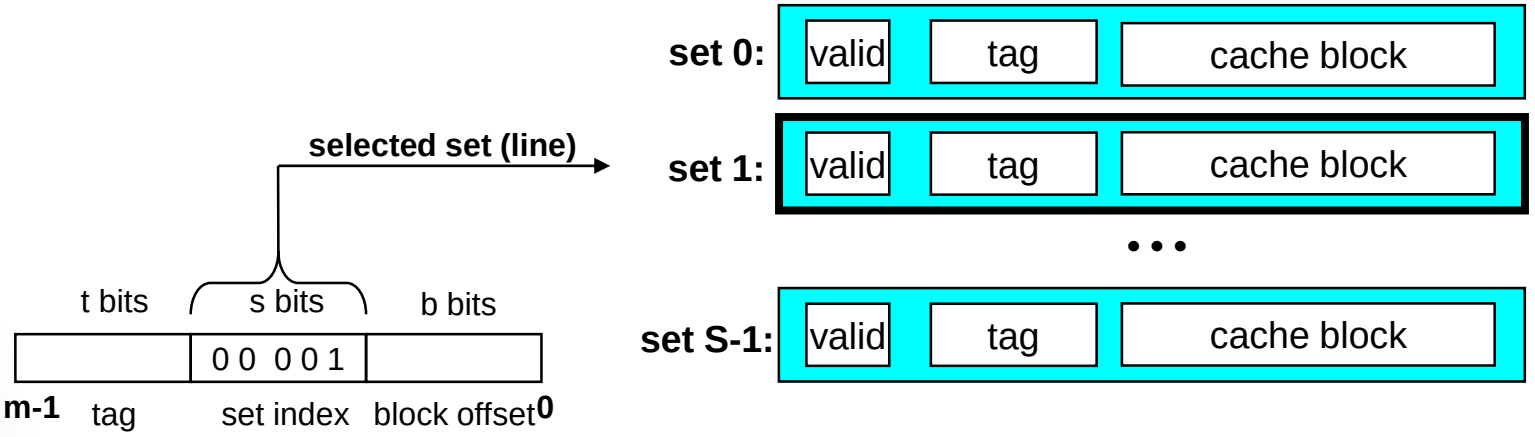
problem	m	C	B	E	S	t	s	b
1	32	4096	8	1				
2	64	1024	8		8			
3	32			32			2	5
4	64		64		16			

direct-mapped cache

problem	m	C	B	E	S	t	s	b
1	32	4096	8	1	512	20	9	3
2	64	1024	8		8			
3	32			32			2	5
4	64		64		16			

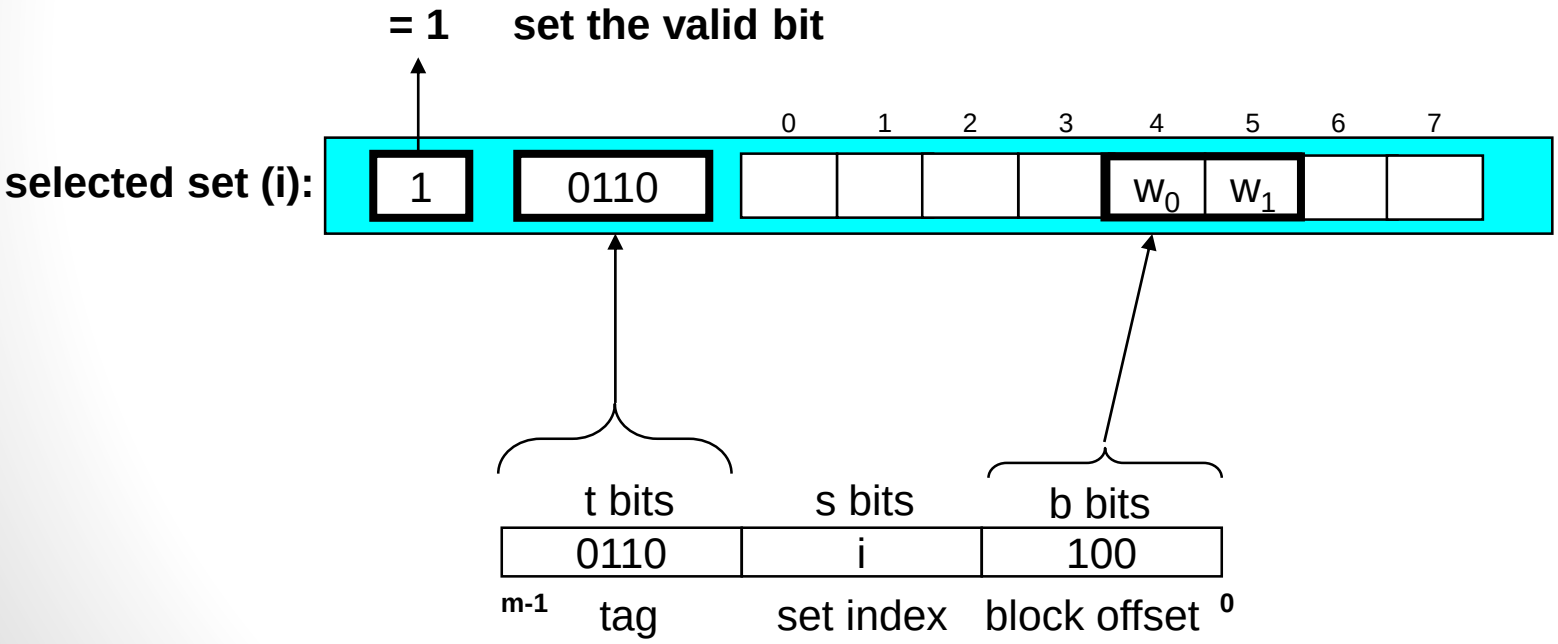
direct-mapped cache – load a 2-byte value set selection

- Use the set (= line) index bits to determine the set of interest.



direct-mapped cache – load a 2-byte value tag and block update, validate

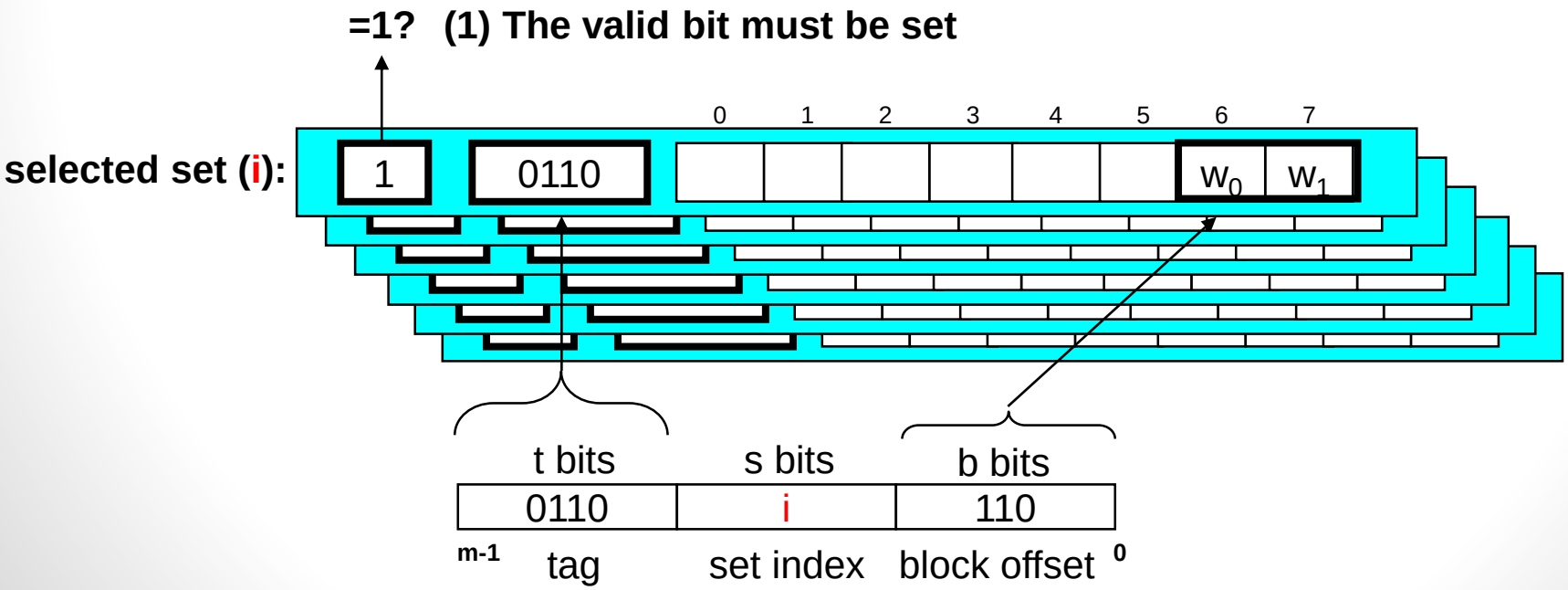
- update the tag bits
- load the cache block
- set valid bit



direct-mapped cache
cache hit

starting with the complete address for the next memory access:

- decode s bits to find set
- check valid bit
- match tag
- use block offset to locate data



it's your turn

problem	m	C	B	E	S	t	s	b
1	32	4096	8	1	512	20	9	3

sizeof(int) = 4
data path width = B
int value_1 = 30
@value_1 = 0x8800200C

Assuming a cache miss,
what is loaded into cache, and where, when value_1 is accessed?

it's your turn

problem	m	C	B	E	S	t	s	b
1	32	4096	8	1	512	20	9	3

sizeof(int) = 4
data path = B
int valueA[256] = 30
@valueA[256] = 0x88002008

```
for ( i = 0; i < 256; i++ )
{
    sum += valueA[i];
]
```

assume sum is a register variable.
What is loaded into cache, and where, as the elements of valueA[] are accessed?

it's your turn

problem	m	C	B	E	S	t	s	b
1	32	4096	16	1	256	20	8	4

sizeof(int) = 4
data path = B
int valueA[2048] = *whatever*
@valueA[2048] = 0x88002008

```
for ( i = 0; i < 1024; i++ )
{
    sum += valueA[i];
]
```

assume sum is a register variable.
What is loaded into cache, and where, as the elements of valueA[] are accessed?
how many main memory accesses are performed?

cse5441 - parallel computing

cache management

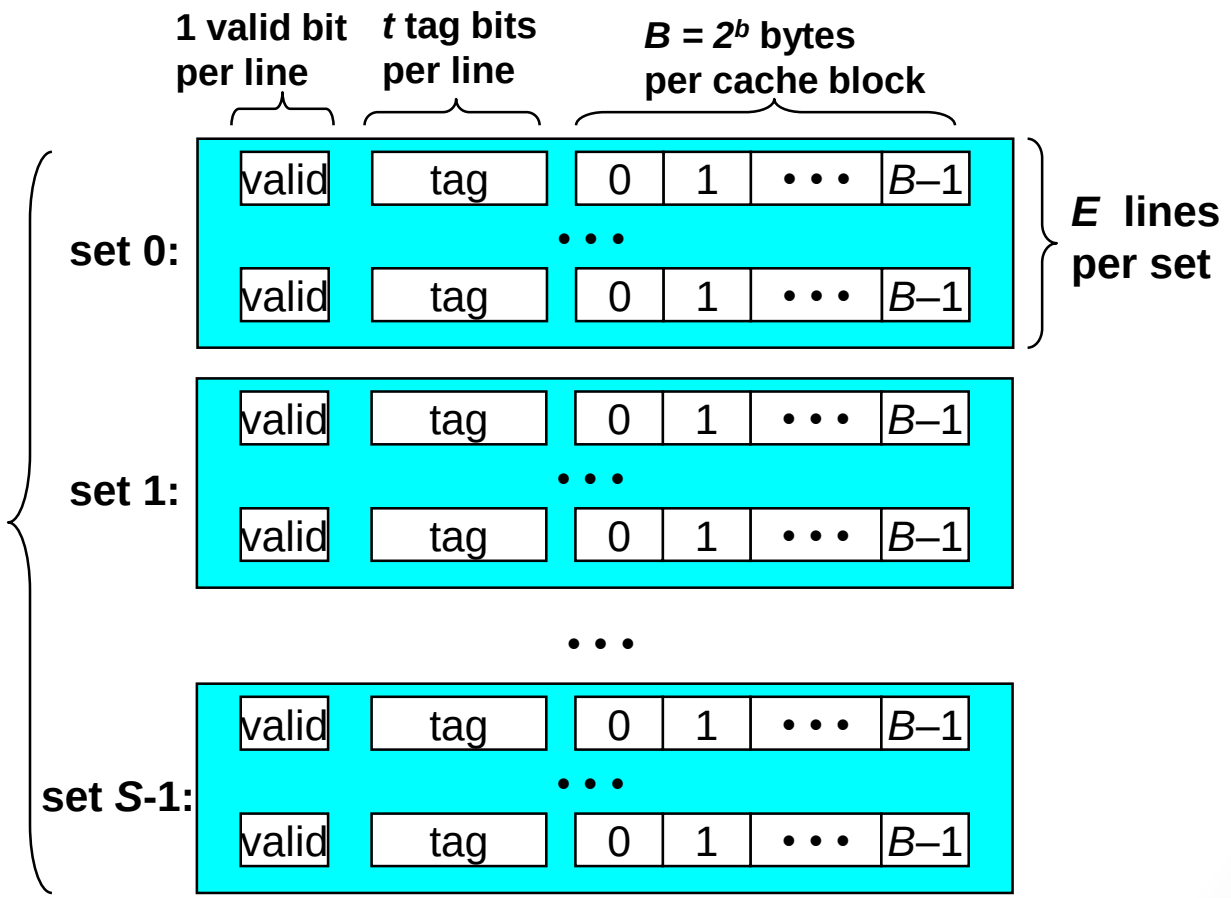
http://frostburg.collegestoreonline.com/webitemimages/326/Images/zzzzzCash%20For%20Books.jpg

cache structure - review

Cache is an array of sets.

Each set contains one or more lines.

Each line holds a block of data.



Cache size: $C = B \times E \times S$ data bytes

direct-mapped cache

example	m	C	B	E	S	t	s	b
1	32	1024	4	1	256	22	8	2

main memory address: AAAA0000

- tag: 1010 1010 1010 1010 0000 00
- set: 0000 0000
- ofst: 00

this will map to set 0 with offset 0
this will load addresses AAAA0000 – AAAA0003 into cache

cache lines

example	m	C	B	E	S	t	s	b
1	32	1024	4	1	256	22	8	2

- addresses within the block range exist on same cache line

access one byte:

main memory address: AAAA0000
- tag: 1010 1010 1010 1010 0000 00
- set: 0000 0000
- ofst: 00

this will map to set 0 with offset 0
this will load addresses
AAAA0000 – AAAA0003
into cache

access one byte:

main memory address: AAAA0001
- tag: 1010 1010 1010 1010 0000 00
- set: 0000 0000
- ofst: 01

this will map to set 0 with offset 1
this will load addresses
AAAA0000 – AAAA0003
into cache

- this also holds for addresses AAAA0002 and AAAA0003

set bit operation

example	m	C	B	E	S	t	s	b
1	32	1024	4	1	256	22	8	2

- as set bits change, cache line moves to next set

main memory address: AAAA0000

tag: 1010 1010 1010 1010 0000 00
- set: 0000 0000
- ofst: 00

set 0 with offset 0

main memory address: AAAA0001-3

tag: 1010 1010 1010 1010 0000 00
- set: 0000 0000
- ofst: 01 - 11

set 0 with offset 1 - 3

main memory address: AAAA0004

tag: 1010 1010 1010 1010 0000 00
- set: 0000 0001
- ofst: 00

set 1 with offset 0

- this is good for spatial locality

set wrap-around

example	m	C	B	E	S	t	s	b
1	32	1024	4	1	256	22	8	2

- addresses which share same set bits will over-write each other

main memory address: AAAA0000

tag: 1010 1010 1010 1010 0000 00
- set: 0000 0000
- ofst: 00

set 0 with offset 0

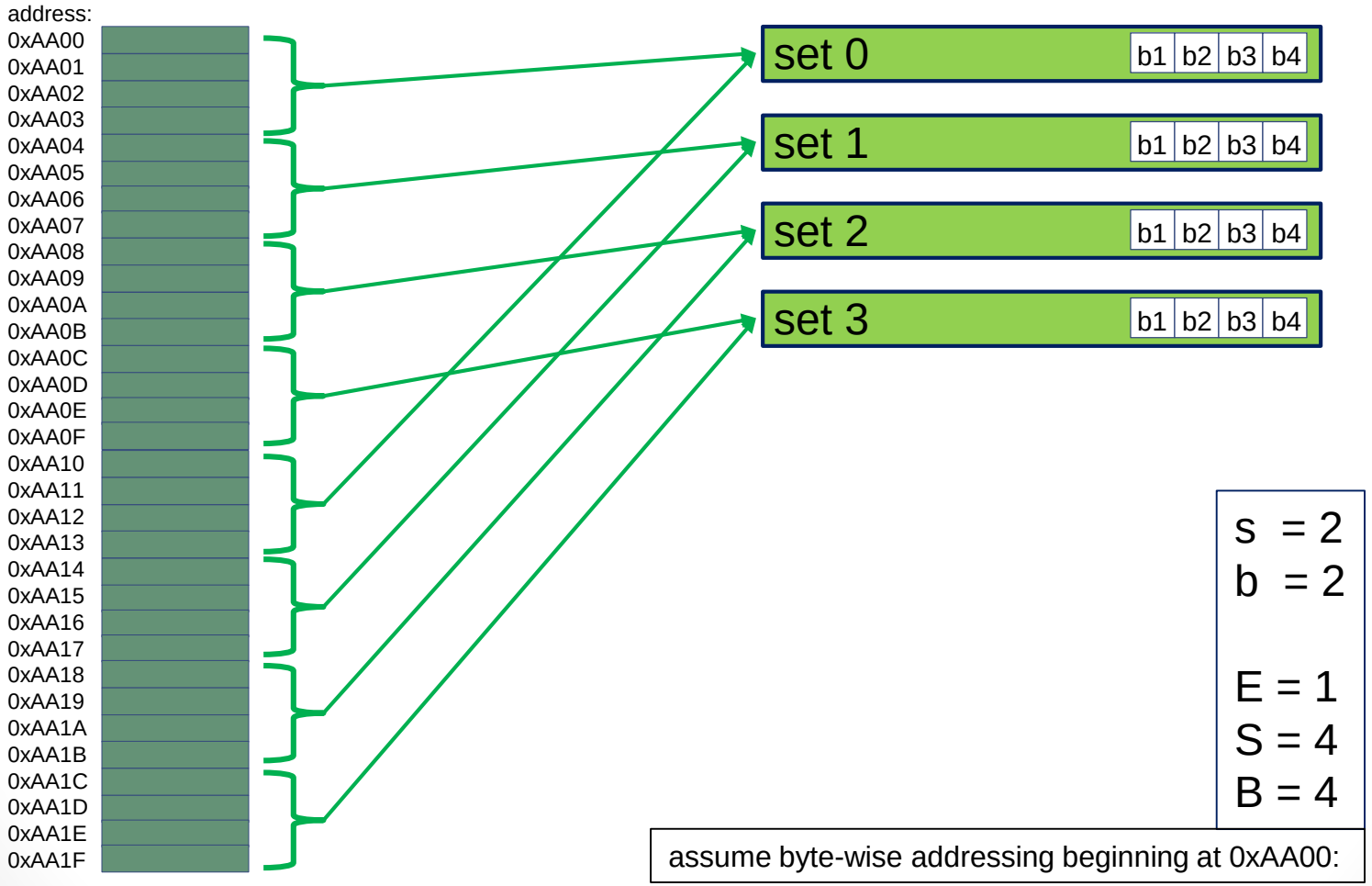
. . .

main memory address: AAAA0400

tag: 1010 1010 1010 1010 0000 01
- set: 0000 0000
- ofst: 00

set 0 with offset 0

set wrap-around



hit rate example

example	m	C	B	E	S	t	s	b
1	32	1024	4	1	256	22	8	2

read address: AAAA0000	cold miss will load AAAA0000 – AAAA0003
read address: AAAA0001	cache hit
read address: AAAA0002	cache hit
read address: AAAA0003	cache hit
read address: AAAA0004	cold miss will load AAAA0004 – AAAA0007
and so forth ...	"hit rate" will be ¾ = 75%

direct-mapped cache

example	m	C	B	E	S	t	s	b
1	32	1024	4	1	256	22	8	2

```
for (i = 0; i < 1000000; i++)
{
      sum += my_array[i];
}
```

size of data items

- let sizeof(int) = 2 (bytes)
- let @my_array[1000000] = AAAA0000
- let sum be a register variable

affected cache performance

what percentage of memory accesses are cache hits? 50%

direct-mapped cache

example	m	C	B	E	S	t	s	b
1	32	1024	4	1	256	22	8	2

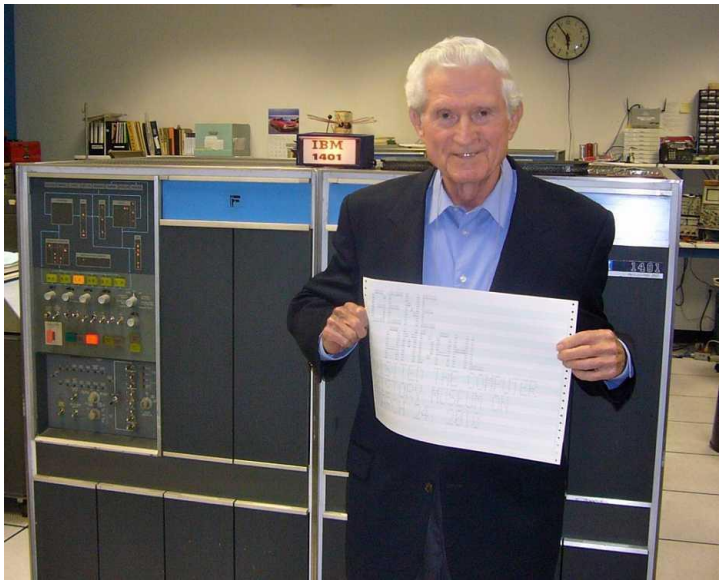
```
for (i = 0; i < 1000000; i += 2)
{
      sum += my_array[i];
}
```

- very slight change
- should be less work, right?

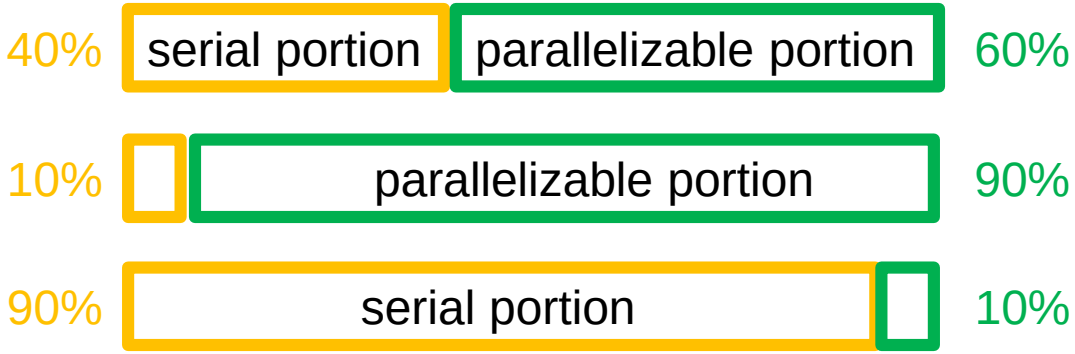
what percentage of memory accesses are cache hits? 0%

scalability ...

Amdahl's Law



The theoretical maximum parallel speed-up
is equal to the percentage of the program
being parallelized.

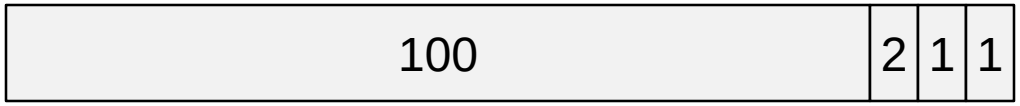


http://ibm-1401.info/GeneAmdahl_1401_Mar2010-.jpg

Amdahl's Law

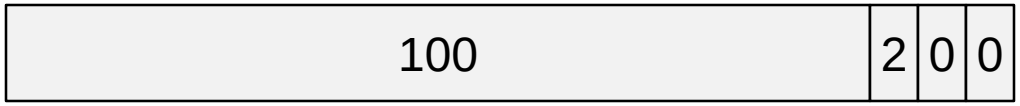
a different application

original example, 50% cache hits:




```
   500,000  *  100 cycles per cache miss memory access
+  500,000  *    2 cycles per cache miss computation
+  500,000  *    1 cycle   per cache hit memory access
+  500,000  *    1 cycle   per cache hit computation
 ========
52,000,000      total work
```

second example, half the numbers, no cache hits:




```
   500,000  *  100 cycles per cache miss memory access
+  500,000  *    2 cycles per cache miss computation
+        0  *    1 cycle   per cache hit memory access
+        0  *    1 cycle   per cache hit computation
 ========
51,000,000      total work
```

1.9% improvement

it's your turn ...

```
for (i = 999999; i >= 0; i--)
{
      sum += my_array[i];
}
```

- let @my_array[1000000] = AAAA0000
- let sum be a register variable

	m	C	B	E	S	t	s	b
P1	32	4096	16	1	?	?	?	?

what are S, t, s and b?

let sizeof(int) = 2 (bytes), what percentage of memory accesses are cache hits?

	m	C	B	E	S	t	s	b
P2	64	4096	16	1	?	?	?	?

what are S, t, s and b?

let sizeof(int) = 4 (bytes), what percentage of memory accesses are cache hits?

it's your turn ... again ...

example	m	C	B	E	S	t	s	b
1	32	1024	4	1	256	22	8	2

```
for (i = 0; i < 1024; i++)
{
      sum += my_matrix[ i ][ 0 ];
}
```

- let sizeof(short int) = 1 (byte)
- let @my_matrix[1024][1024] = AAAA0000 (assume contiguous)
- let sum be a register variable

what percentage of memory accesses are cache hits?

how could I fix this?

part one

- let sizeof(short int) = 1 (byte)
- let @sum[1024] = AAAA0000
- let @my_matrix[1024][1024] = AAAA0400 (contiguous)
- assume sum[x] are not assigned registers

same cache parameters:

- C = 1024 B = 4
- E = 1 S = 256
- t = 22 s = 8
- b = 2 m = 32

```
for (j = 0; j < 1024; j++)
{
    for (i = 0; i < 1024; i++)
    {
        sum[ j ] += my_matrix[ i ][ j ];
    }
}
```

```
for (i = 0; i < 1024; i++)
{
    for (j = 0; j < 1024; j++)
    {
        sum[ j ] += my_matrix [ i ] [ j ];
    }
}
```

how could I fix this?

part two

- let sizeof(short int) = 1 (byte)
- let @sum[1024] = AAAA0000
- let @pad[4] = AAAA0400
- let @my_matrix[1024][1024] = AAAA0404
- assume sum[x] are not assigned registers

same cache parameters:

- C = 1024 B = 4
- E = 1 S = 256
- t = 22 s = 8
- b = 2 m = 32

padding & tiling

```
for (j = 0; j < 1024; j++)
{
    for (i = 0; i < 1024; i++)
    {
        sum[ j ] += my_matrix[ i ][ j ];
    }
}
```

```
for (j = 0; j < 1024; j += 4)
{
    for (i = 0; i < 1024; i++)
    {
        sum[ j ]   += my_matrix[ i ] [ j ];
        sum[ j+1 ] += my_matrix[ i ][ j+1 ];
        sum[ j+2 ] += my_matrix[ i ][ j+2 ];
        sum[ j+3 ] += my_matrix[ i ][ j+3 ];
    }
}
```

cache terminology

summary

cache	a smaller, faster memory component, organized into sets of cache lines
cache line	a unit of cache, containing data block, tag (identification) bits and a valid bit (sometimes in context – just the cache block)
cache block	a set of elements of a cache line containing fetched data
valid bit	an indicator that the cache line contains current data
tag bits	a section of a fetched address stored in a cache line which, with the set id, fully identifies the data block
word	an architecture-specific element which is some number of bytes in size

cache terminology

summary

M	virtual address space = 2^m
m	size, in bits, of memory addresses
C	total (usable) cache size (in bytes) = B x E x S
B	cache block size (in bytes) = 2^b
b	number of address bits used as the cache block offset
E	number of cache lines per set
S	number of cache sets
s	number of address bits used to specify set
t	number of address bits used as cache tag. coupled with the set id, specifies which virtual memory address block is currently in a cache line

cache terminology

summary

direct-mapped	a cache which has one line per set (E=1)
registers	very small on-cpu no-latency memory, access typicaly controlled by compilers
level one (L1) cache	very small on-chip minimal latency memory (~64Kb/core, 1-2cycles)
level two (L2) cache	small off-chip very low latency memory (~256Kb/core, ~10 cycles)
level three (L3) cache	off-board shared low latency memory (~2-20Mb, ~10-50 cycles)
main memory ("RAM")	slower cost effective memory which functions as a cache for "permanent" storage devices (many GB, ~300-500 cycles)

cache terminology

summary

spatial locality	the tendency to access a memory address which is close to a previous recently accessed address
temporal locality	the tendency to access a memory address which has be previously recently accessed
fetching	moving data from one class of storage into a faster class
pre-fetching	moving data to a faster storage class in anticipation of it's actual use
data path	a bus or other circuit over which data may move
data path width	the number of bits which may transfer over a data path in a single transaction

cache terminology

summary

row major	a method for organizing multi-dimensional arrays of data so that successive row elements are stored in adjacent memory locations
column major	a method for organizing multi-dimensional arrays of data so that successive column elements are stored in adjacent memory locations
padding	a technique for adding unused memory to a data structure to avoid alignment effects
tiling	formally, a set of disjoint open sets whose closure covers an entire data set. also a technique for ordered access of spatially localized subsets of data in order to improve average latency

cache terminology

summary

cold miss	misses causes by the first reference to a given memory location. Also called a *compulsory miss* as any specific memory location must be initially loaded into cache.
conflict miss	misses relating to lines that were previously in the cache, but had been evicted due to competition/conflict with other memory addresses.
capacity miss	misses which were neither cold nor conflict misses, but due solely to the finite size of the cache.

cache performance metrics

hit rate	cache hits / number of references
miss rate	cache misses / number of references
hit time	time to locate and deliver cached data to the processor (in cycles)
miss penalty	additional time to deliver data to the processor required because the data was not in a cache

cse5441 - parallel computing

cache management

http://frostburg.collegestoreonline.com/webitemimages/326/Images/zzzzzCash%20For%20Books.jpg

set bits

example	m	C	B	E	S	t	s	b
1	32	1024	4	1	256	22	8	2

- let sizeof(element) = 4 bytes
- let @ my_array[256] = AAAA0400
- assume sum, max in registers

```
for (i = 0; i < 256; i++)
{
      sum += my_array[i];
}
```

```
max = my_array[ 0 ]
for (i = 1; i < 256; i++)
{
    if ( my_array[ i ] > max)
        max = my_array[ i ];
}
```

access my_array[0]
AAAA0400 = 1010 1010 1010 1010 0000 0100 0000 0000
tag = AAAA00 = 1010 1010 1010 1010 0000 01
set = 00 = 00 0000 00

next access
AAAA0404 = 1010 1010 1010 1010 0000 0100 0000 0100
tag = AAAA00 = 1010 1010 1010 1010 0000 00
set = 01 = 00 0000 01

.
.
.

if set bits were MSB ...

example	m	C	B	E	S	t	s	b
1	32	1024	4	1	256	22	8	2

```
for (i = 0; i < 256; i++)
{
      sum += my_array[i];
}
```

```
max = my_array[ 0 ]
for (i = 1; i < 256; i++)
{
    if ( my_array[ i ] > max)
        max = my_array[ i ];
}
```

access my_array[0]
AAAA0400 = 1010 1010 1010 1010 0000 0100 0000 0000
tag = AA0000 = 1010 1010 0000 0100 0000 00
set = AA = 1010 1010

next access
AAAA0404 = 1010 1010 1010 1010 0000 0100 0000 0100
tag = AA0004 = 1010 1010 0000 0100 0000 01
set = AA = 1010 1010
.
.
.

direct-mapped cache

example	m	C	B	E	S	t	s	b
5	32	16k	32	1				

let @my_array[4096] = AAAA0000
sizeof(int) = 8 bytes

sum is in a register

```
for (i = 0; i < 4096; i++)
{
    sum += my_array[ i ];
}
```

direct-mapped cache

example	m	C	B	E	S	t	s	b
5	32	16k	32	1	512	18	9	5

let @my_array[4096] = AAAA0000
sizeof(int) = 8 bytes

sum is in a register

```
for (i = 0; i < 4096; i++)
{
    sum += my_array[ i ];
}
```

AAAA0000 = 1010 1010 1010 1010 0000 0000 0000 0000
tag =
set =
ofst =

direct-mapped cache

example	m	C	B	E	S	t	s	b
5	32	16k	32	1	512	18	9	5

let @my_array[4096] = AAAA0000
sizeof(int) = 8 bytes

sum is in a register

```
for (i = 0; i < 4096; i++)
{
    sum += my_array[ i ];
}
```

AAAA0000 = 1010 1010 1010 1010 0000 0000 0000 0000 (cold) miss
tag = AAAA00 = 1010 1010 1010 1010 00
set = 00 = 00 0000 000
ofst = 00 = 0 0000

AAAA0008 = 1010 1010 1010 1010 0000 0000 0000 1000
tag =
set =
ofst =

direct-mapped cache

example	m	C	B	E	S	t	s	b
5	32	16k	32	1	512	18	9	5

let @my_array[4096] = AAAA0000
sizeof(int) = 8 bytes

sum is in a register

```
for (i = 0; i < 4096; i++)
{
    sum += my_array[ i ];
}
```

AAAA0000 = 1010 1010 1010 1010 0000 0000 0000 0000 (cold) miss
tag = AAAA00 = 1010 1010 1010 1010 00
set = 00 = 00 0000 000
ofst = 00 = 0 0000

AAAA0008 = 1010 1010 1010 1010 0000 0000 0000 1000 hit
tag = AAAA00 = 1010 1010 1010 1010 00
set = 00 = 00 0000 000
ofst = 08 = 0 1000

AAAA0010 = 1010 1010 1010 1010 0000 0000 0001 0000

direct-mapped cache

example	m	C	B	E	S	t	s	b
5	32	16k	32	1	512	18	9	5

let @my_array[4096] = AAAA0000
sizeof(int) = 8 bytes

sum is in a register

```
for (i = 0; i < 4096; i++)
{
      sum += my_array[ i ];
}
```

. . .

AAAA0008 = 1010 1010 1010 1010 0000 0000 0000 1000 hit
tag = AAAA00 = 1010 1010 1010 1010 00
set = 00 = 00 0000 000
ofst = 08 = 0 1000

AAAA0010 = 1010 1010 1010 1010 0000 0000 0001 0000 hit
tag = AAAA00 = 1010 1010 1010 1010 00
set = 00 = 00 0000 000
ofst = 10 = 1 0000

AAAA0018 = 1010 1010 1010 1010 0000 0000 0001 1000

direct-mapped cache

example	m	C	B	E	S	t	s	b
5	32	16k	32	1	512	18	9	5

let @my_array[4096] = AAAA0000
sizeof(int) = 8 bytes

sum is in a register

```
for (i = 0; i < 4096; i++)
{
      sum += my_array[ i ];
}
```

. . .

AAAA0010 = 1010 1010 1010 1010 0000 0000 0001 0000 hit
tag = AAAA00 = 1010 1010 1010 1010 00
set = 00 = 00 0000 000
ofst = 10 = 1 0000

AAAA0018 = 1010 1010 1010 1010 0000 0000 0001 1000 hit
tag = AAAA00 = 1010 1010 1010 1010 00
set = 00 = 00 0000 000
ofst = 18 = 1 1000

AAAA0020 = 1010 1010 1010 1010 0000 0000 0010 0000

direct-mapped cache

example	m	C	B	E	S	t	s	b
5	32	16k	32	1	512	18	9	5

let @my_array[4096] = AAAA0000
sizeof(int) = 8 bytes

sum is in a register

```
for (i = 0; i < 4096; i++)
{
    sum += my_array[ i ];
}
```

. . .

AAAA0018 = 1010 1010 1010 1010 0000 0000 0001 0100 hit
tag = AAAA00 = 1010 1010 1010 1010 00
set = 00 = 00 0000 000
ofst = 10 = 1 0000

AAAA0020 = 1010 1010 1010 1010 0000 0000 0010 0000 (cold) miss
tag = AAAA00 = 1010 1010 1010 1010 00
set = 01 = 00 0000 001
ofst = 00 = 0 0000

it's your turn ...

declarations and code may not be modified

given: @my_vals1[4096] = 0xAAAA0000
@my_vals2[4096] = 0xAAAA1000
@results[4096] = 0xAAAA2000

sizeof(element) = 1

```
for (i = 0; i < 4096; i++)
{
    results[i]  + = my_vals1[ i ] – my_vals2[i];
}
```

specify a DM cache that will achieve a hit rate of ≥ 50%

(m, C, B, E, S, t, s b = ?)

it's your turn ...

declarations and code may be modified

```
given:    @my_vals1[4096] = 0xAAAA0000
          @my_vals2[4096] = 0xAAAA1000
          @results[4096]  = 0xAAAA2000

          sizeof(element) = 1
```

```
for (i = 0; i < 4096; i++)
{
     results[i]  + = my_vals1[ i ] – my_vals2[i];
}
```

specify a DM cache that will achieve a hit rate of ≥ 50%

(m, C, B, E, S, t, s b = ?)

what is the smallest DM cache that could achieve a hit rate of ≥ 50%

it's your turn ...

declarations and code may not be modified

given: @my_vals1[4096] = 0xAAAA0000
@my_vals2[4096] = 0xAAAA1000
@results[4096] = 0xAAAA2000

sizeof(element) = 1

```
for (i = 0; i < 4096; i+=2)
{
    results[i]  + = my_vals1[ i ] – my_vals2[i];
}
```

specify a DM cache that will achieve a hit rate of ≥ 50%

(m, C, B, E, S, t, s b = ?)

what is the smallest DM cache that could achieve a hit rate of ≥ 50%

it's your turn ...

declarations and code may not be modified

given: @my_vals1[4096] = 0xAAAA0000
@my_vals2[4096] = 0xAAAA4000
@results[4096] = 0xAAAA8000

sizeof(element) = 4

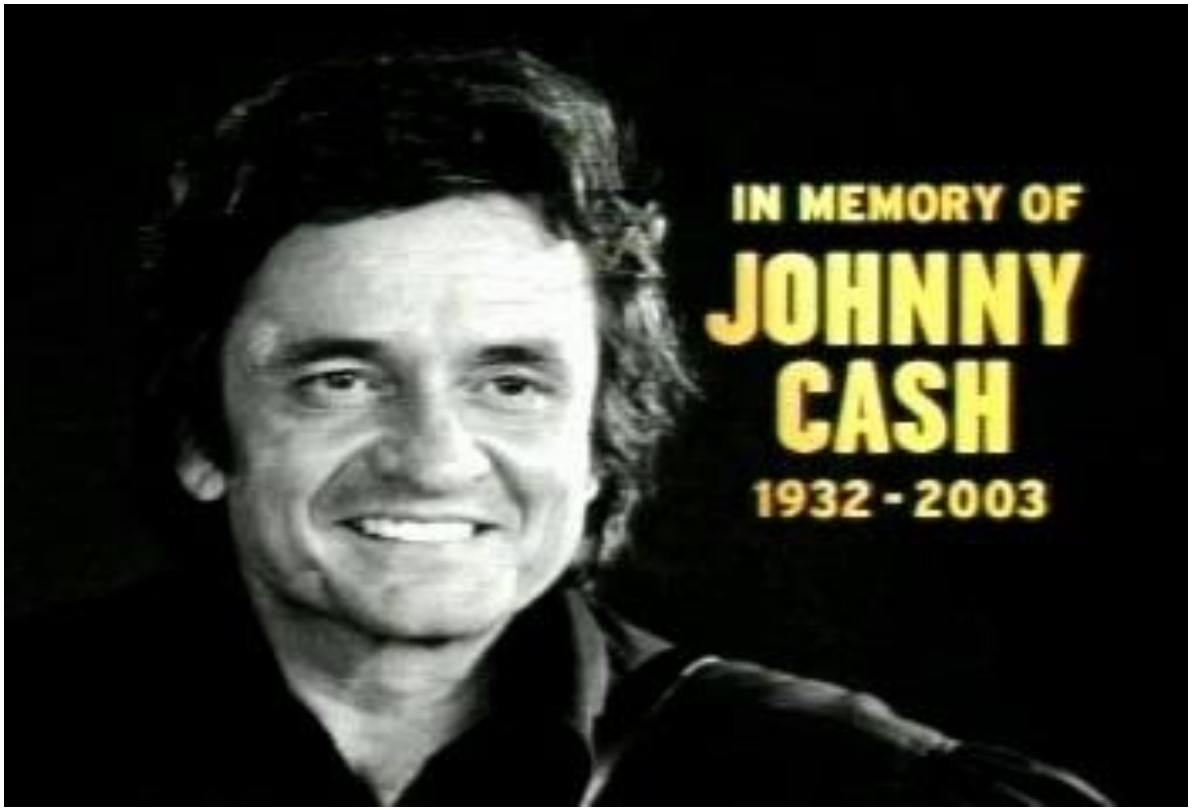
```
for (i = 0; i < 4096; i++)
{
    results[i]  + = my_vals1[ i ] – my_vals2[i];
}
for (i = 0; i < 4096; i++)
{
    results[i]++;
}
```

specify a DM cache that will achieve an overall hit rate of ≥ 50%

(m, C, B, E, S, t, s b = ?)

what is the smallest DM cache that could achieve a hit rate of 100% for the second loop?

cache memory



http://www.freetimeweb.nl/home/picture/sub/muziek/cash/cash_in-memory.jpg

cse5441 - parallel computing

cache management

http://frostburg.collegestoreonline.com/webitemimages/326/Images/zzzzzCash%20For%20Books.jpg

cache management - answers

slide 19:

problem	m	C	B	E	S	t	s	b
1	32	4096	8	1	512	20	9	3
2	64	1024	8	16	8	58	3	3
3	32	4096	32	32	4	25	2	5
4	64	2048	64	2	16	54	4	6

cache management - answers

slide 24:

* set one is selected
* valid bit is set
* tag set to 0x88002
* blocks 4-7 set to 0x00, 0x00, 0x00, 0x1E
* blocks 0-3 set to the integer value stored at 0x8802008

slide 25:

* valueA[0-1] stored in set 1,
  [2-3] stored in set 2,
  [3-4] stored in set 3, . . .
  [255-256] stored in set 128
* tag and valid bits being set as appropriate

slide 26:

* valueA[0-1] stored in set 0,
  [2-5] stored in set 1,
  [6-9] stored in set 2,
  . . .
  [1014-1017] stored in set 254,
  [1018-1023] stored in set 255,
  [1024-1025] overwrites set 0,
* tag and valid bits being set as appropriate

total of 257 main memory accesses

cache management - answers

slide 39: P1: S=256, t=20, s=8, b=4
P2: S=256, t=52, s=8, b=4

slide 40: hit rate 0%

slide 60: One answer is to make the cache large enough to hold all values with a block size of at least 2

E = 1 (DMC)
B = 2, b = 1
S ≥ 2048*3 = 6144, S = 8192
s = 13
C = 16K (16,384)
t = m-14

Another approach to same conclusion is to realize that bits 13-14 are the LSB changing with the base values of the arrays. If those bits are in s, then the array elements of same index will be in different sets. So, b+s ≥ 14.

Even better, make B larger ...

E=1
b=12, B = 4096
s = 2, S = 4
C = 16K
t = m-14
cache hit rate is now 99+%

cache management - answers

slide 61:

Now we can add padding between the arrays:

```
my_vals1[4098];          or, add B bytes
my_vals2[4098];
results[4096];
```

Now, my_vals1[0], my_vals2[0]and results[0] will map to different sets as long as LSB 1 and 2 are in s:

E = 1
B = 2, b = 1
s = 2, S = 4
C = 8 bytes
hit rate = 50%

Making B larger and adding more padding improves cache performance

```
my_vals1[4352];          or, add B bytes
my_vals2[4352];
results[4096];
```

E = 1
b = 8, B = 256
s = 2, S = 4
C = 1K
hit rate = +99%
Larger values of S will still work while providing more realistic general-purpose configurations. Larger values of B can also be used to further improve performance, at the expense of increasing padding.

cache management - answers

slide 62: We will need a block size of at least 4 to hold two referenced entries.
If s+b >= 14, then all corresponding array elements will go to different sets.

E = 1 (DMC)
B = 4, b = 2
s = 12
t = m-14
S = 4096
C = 16K (16,384)

cache management - answers

slide 63: We need block size B ≥ 8, b ≥ 3, to get two size-4 elements into a cache line.
To get base addresses of arrays into different sets, we need b+s ≥ 16, so s ≥ 13.

E = 1 (DMC)
B = 8, b = 3
s = 13
t = m-16
S = 8192
C = 64K (65,536)

my_vals1[ ] will map to sets 0 – 2047
my_vals2[ ] will map to sets 2048 – 4095
results[ ] will map to sets 4096 – 6143
every other access for each array would be a hit, so 50% hit rate for loop 1
with no contention, all of results[ ] will be in cache for loop 2, for hit rate of 100% on that loop.

The cache may be as small as 16K to achieve a 100% hit rate for loop 2.
In this case, since b ≥ 3, s can be at most 11, resulting in all arrays thrashing and a loop 1 hit rate of 0%.
However, the array access order is (RH) my_vals1[ ], my_vals2[ ], (LH) results[ ].
results[ ] will be last loaded into each cache line and will remain there for loop 2.