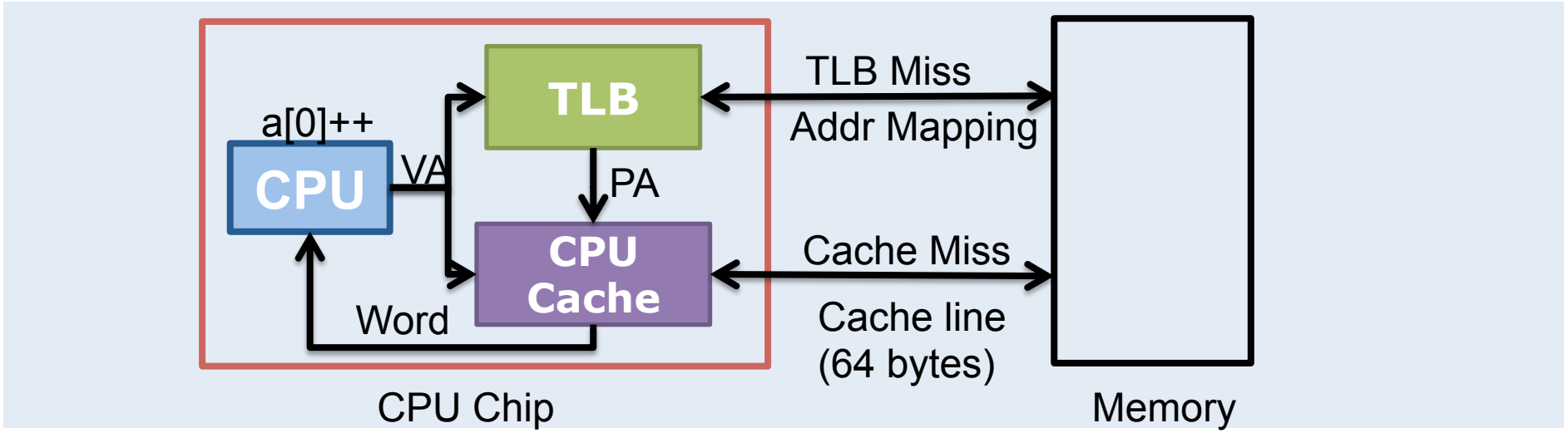
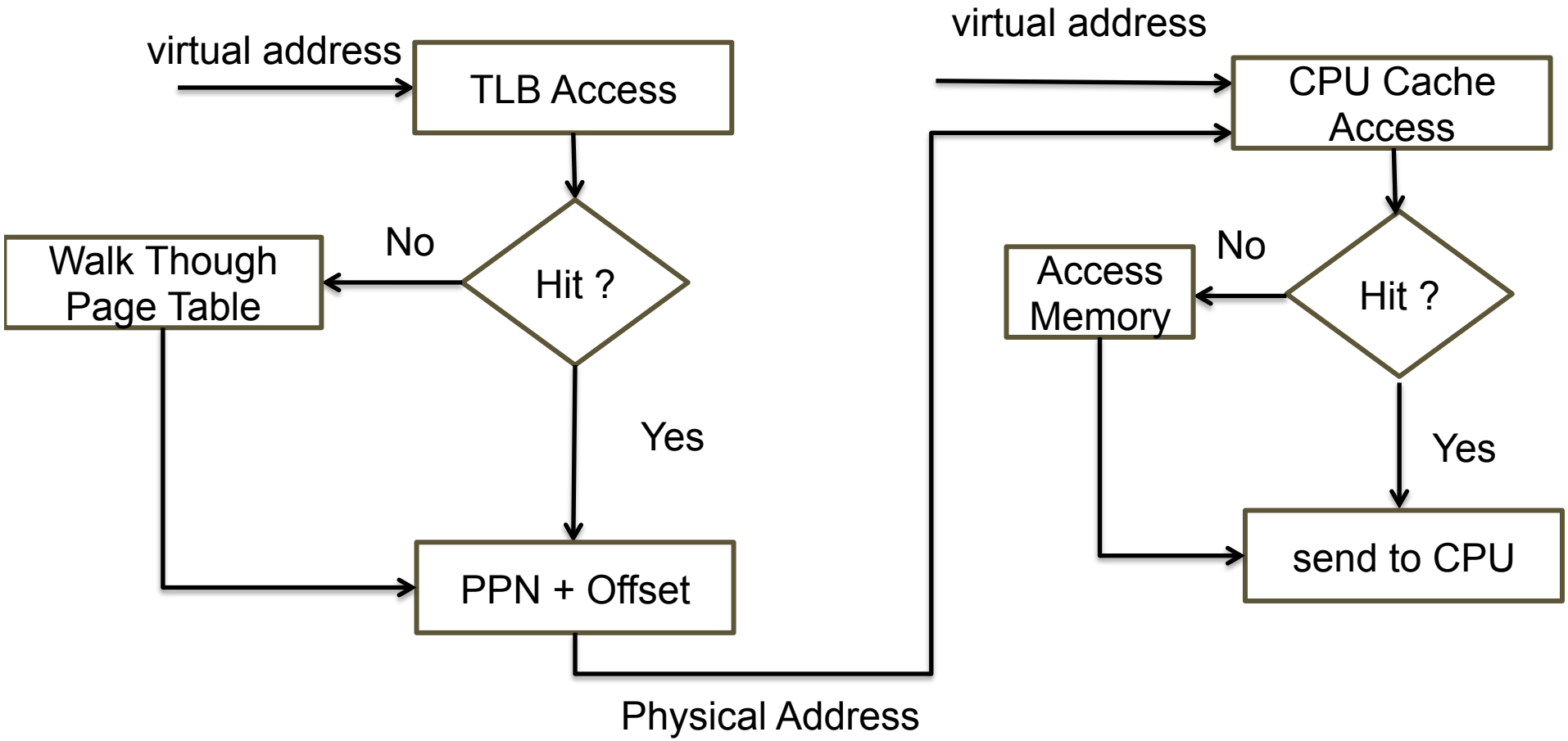
Cache-Friendly Code

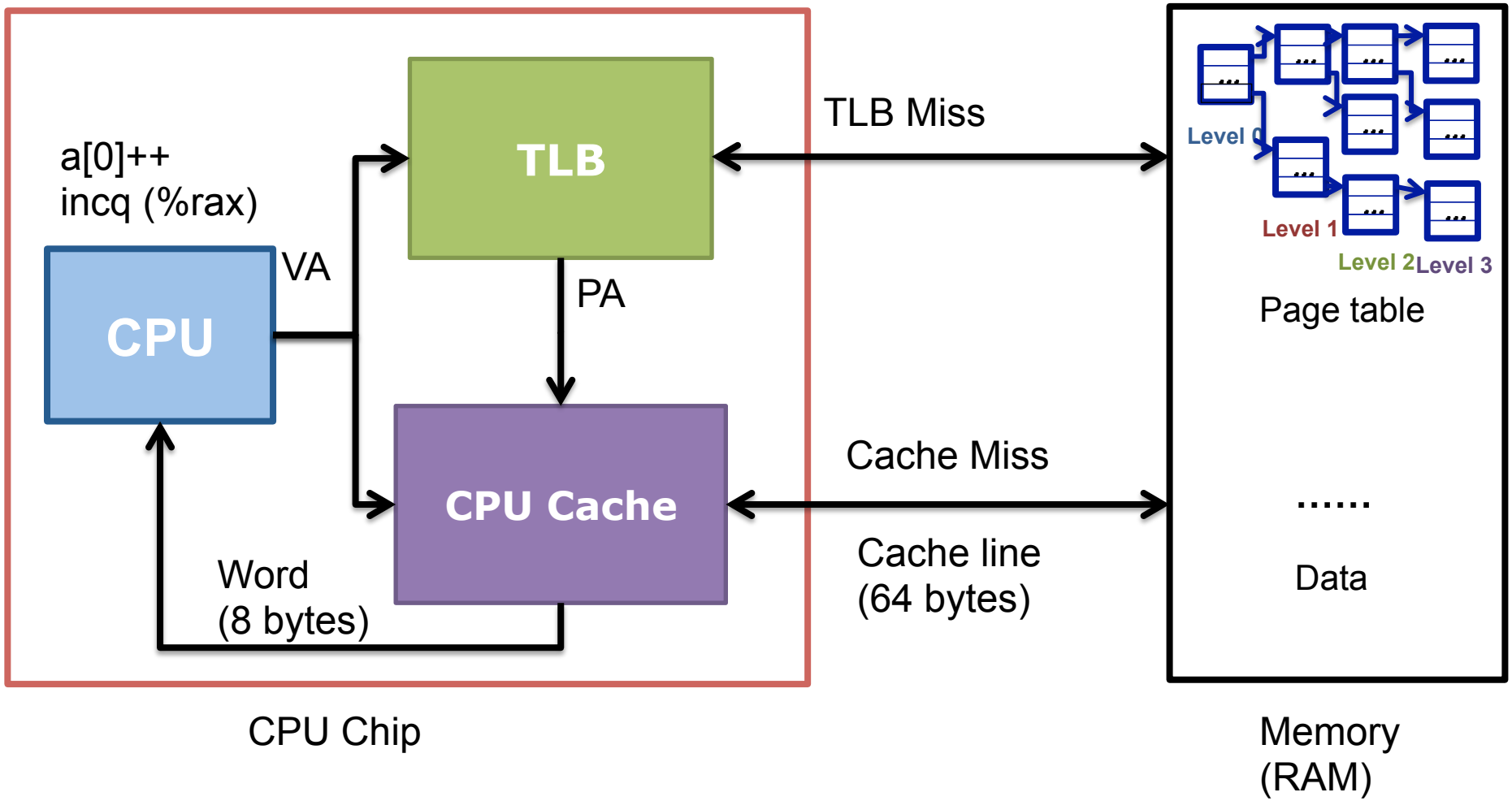
Jinyang Li

based on the slides of Tiger Wang

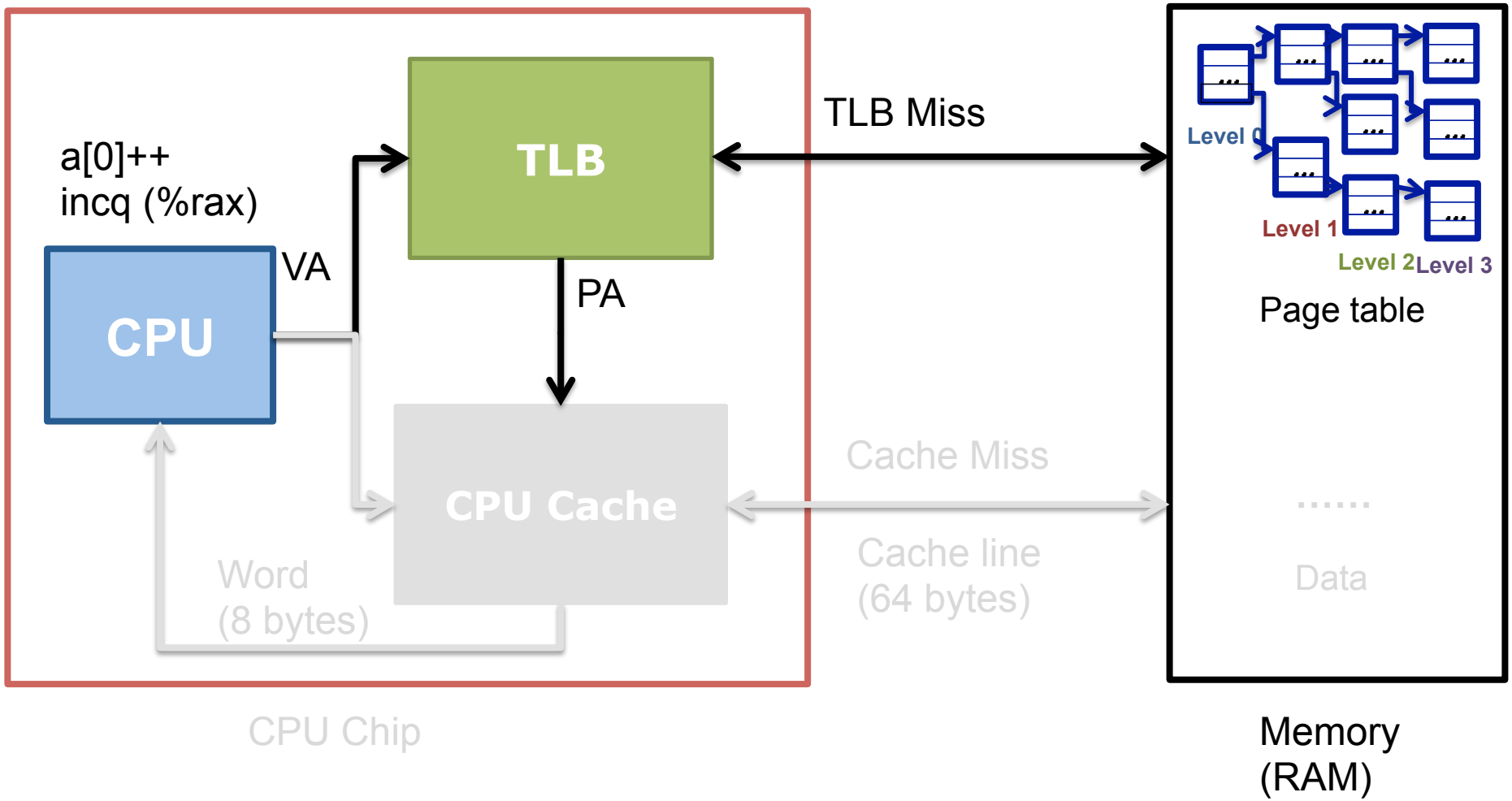

TLB
a[0]++
CPU
VA
PA
CPU
Cache
Word
TLB Miss
Addr Mapping
Cache Miss
Cache line
(64 bytes)
CPU Chip
Memory


virtual address
TLB Access
No
Hit ?
Walk Though
Page Table
Yes
PPN + Offset
Physical Address
virtual address
CPU Cache
Access
No
Hit ?
Access
Memory
Yes
send to CPU

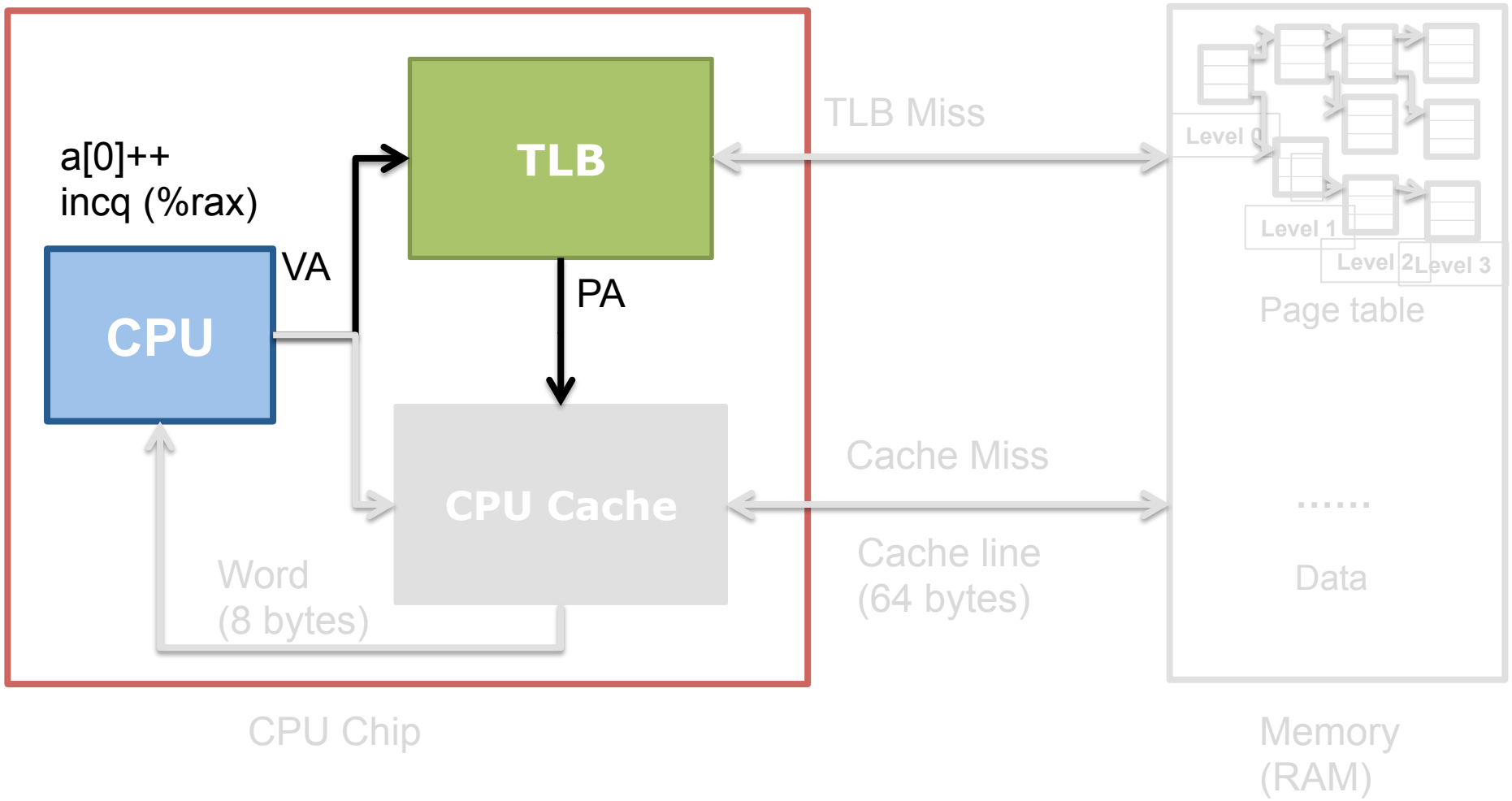
Step 1. Address Translation



Step 1. Address Translation

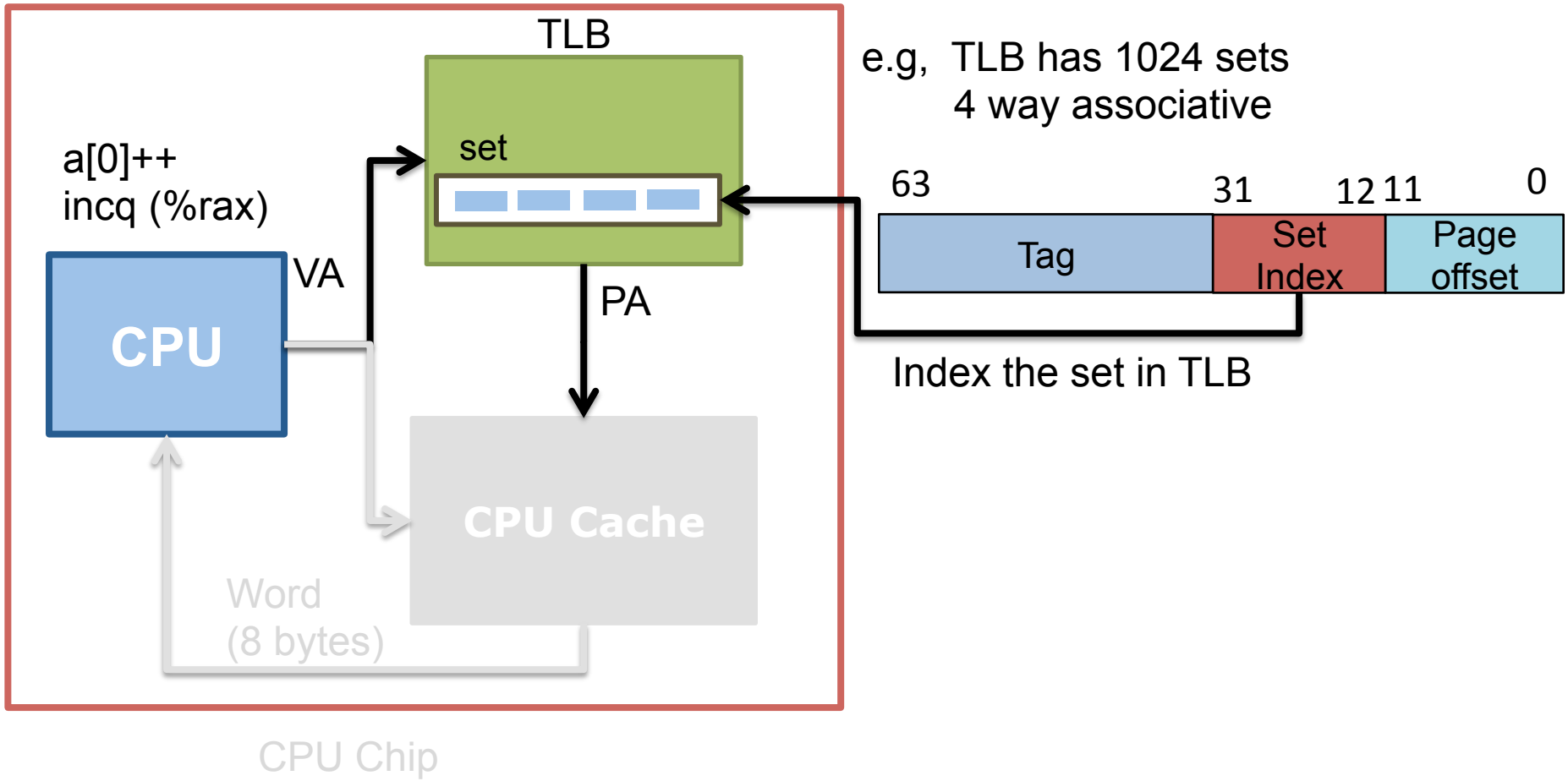


Step 1.1 Check TLB

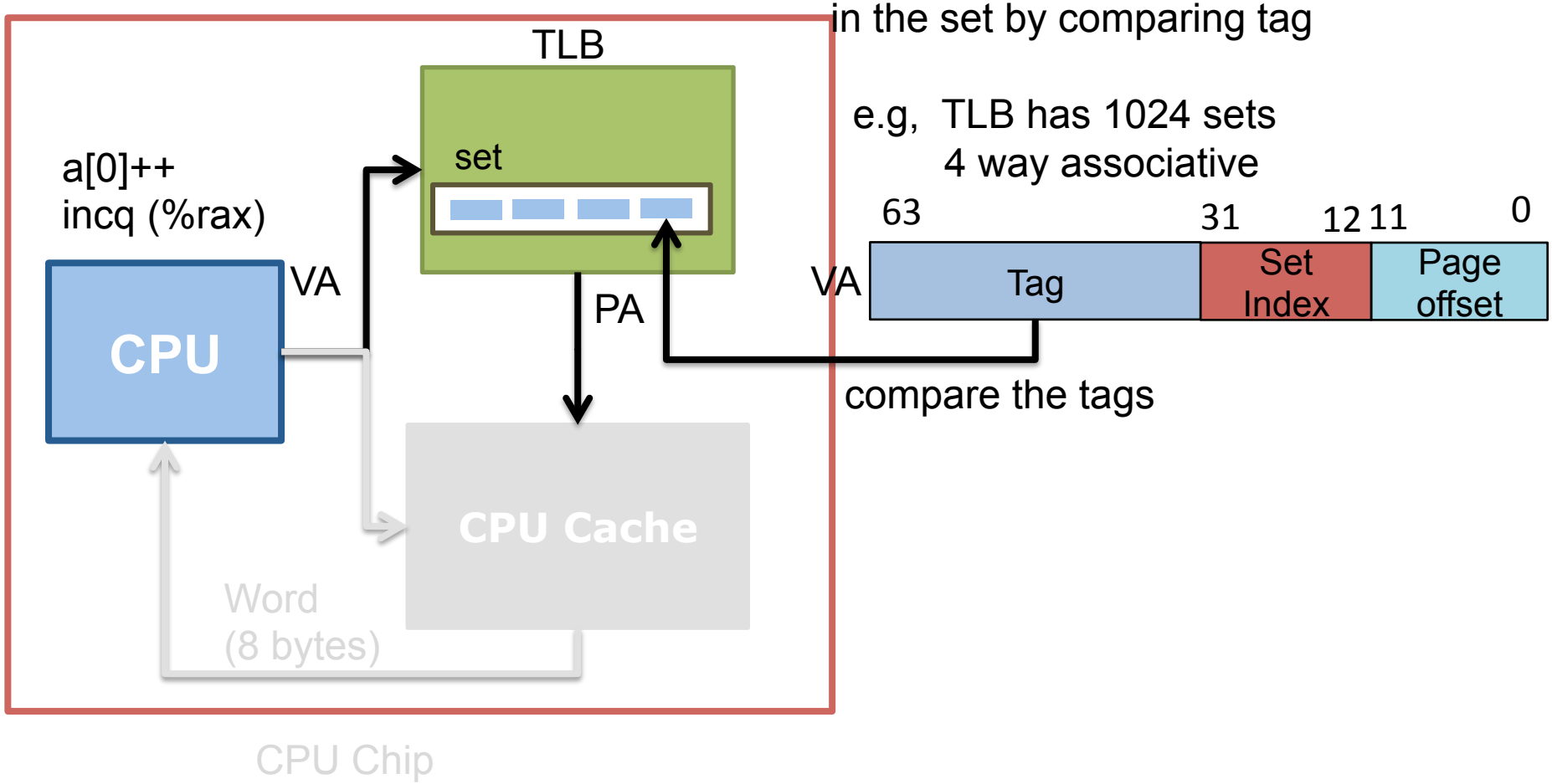


Step 1.1 Check TLB

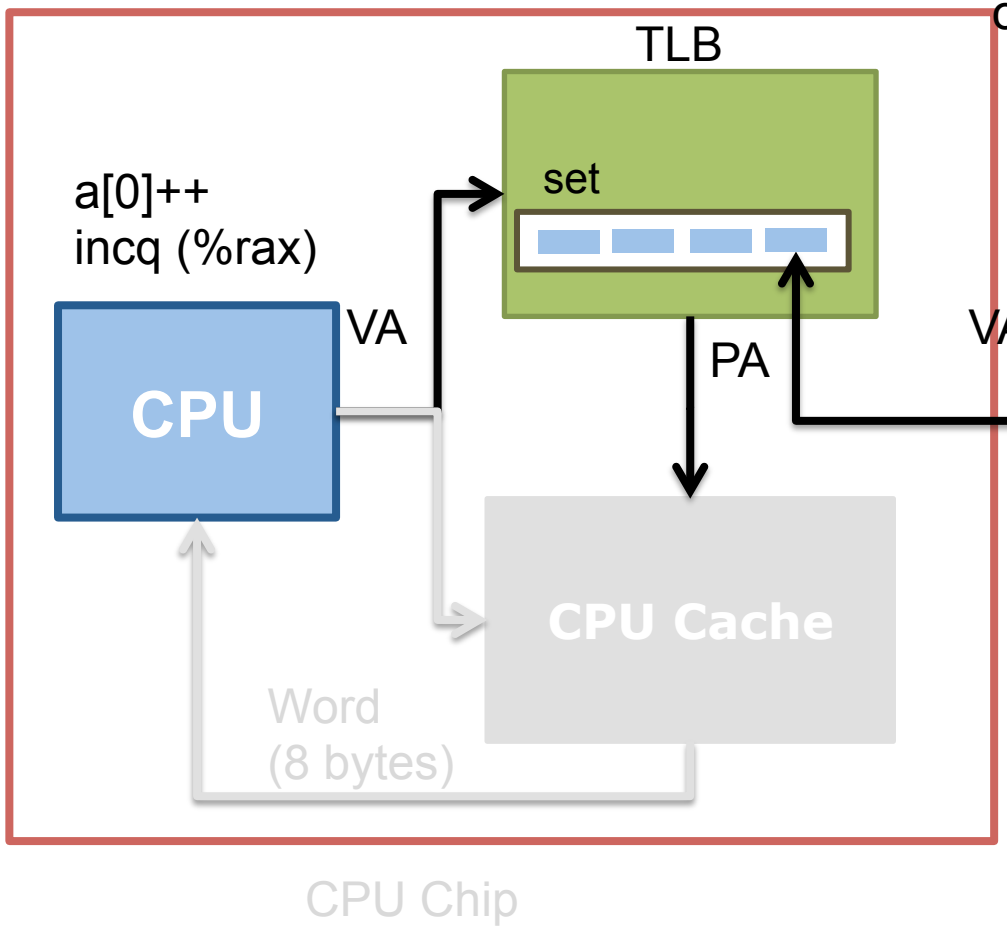
Step 1.1.1 calculate the set index in TLB



Step 1.1 Check TLB

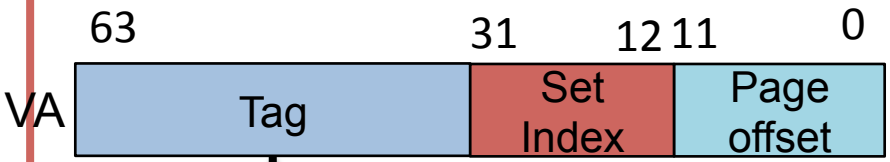


Step 1.1 Check TLB



Step 1.1.3 caculate the physical address on TLB hit

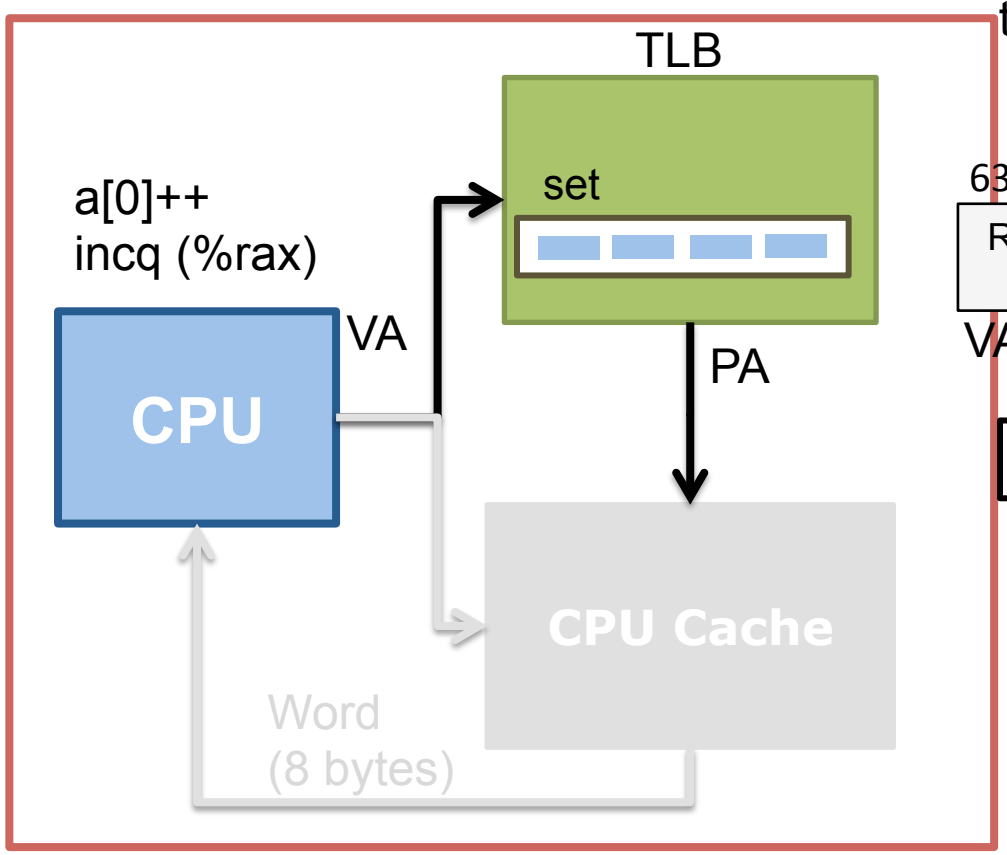
e.g, TLB has 1024 sets
4 way associative



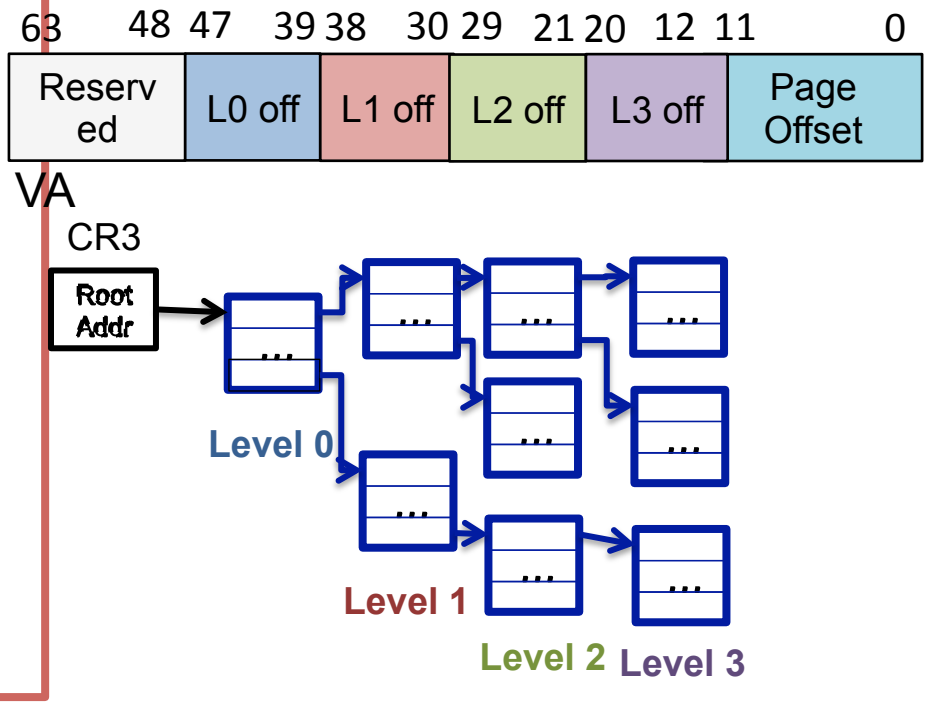
compare the tags

if entry e is present at index va[12:31] an
e.tag==va[32:63], then PA is equal to
e.PPN + va[0:11]

Step 1.2 Walk Page Table on TLB Miss

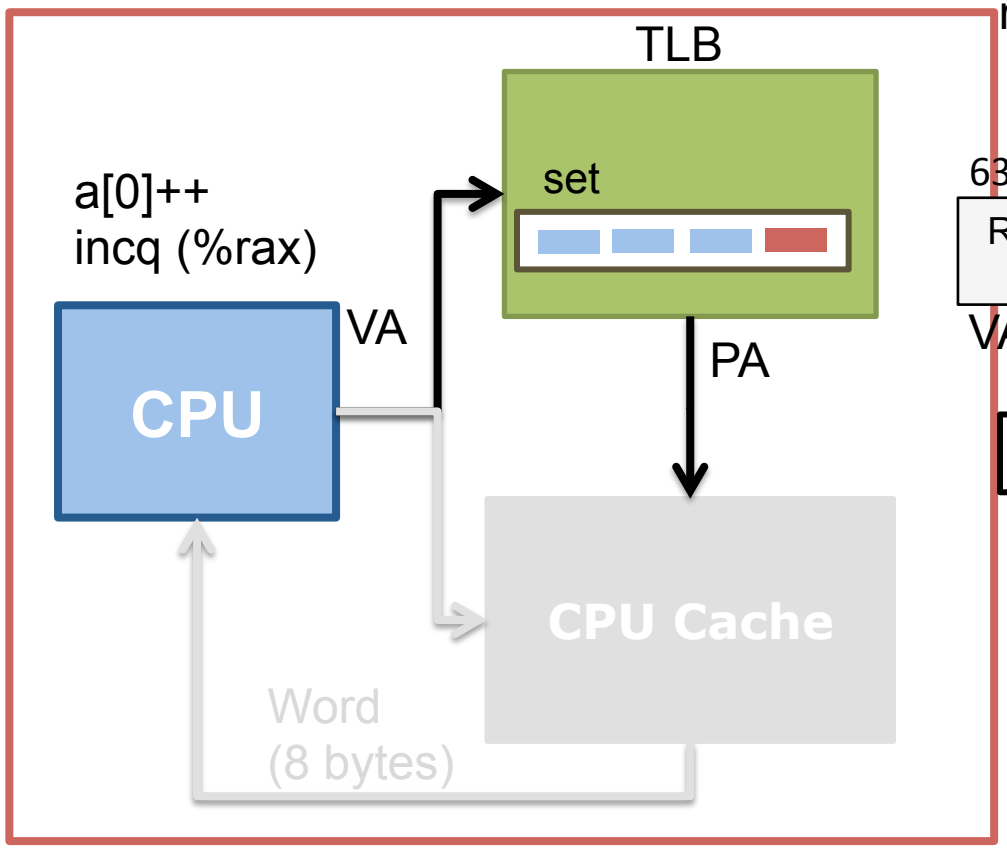


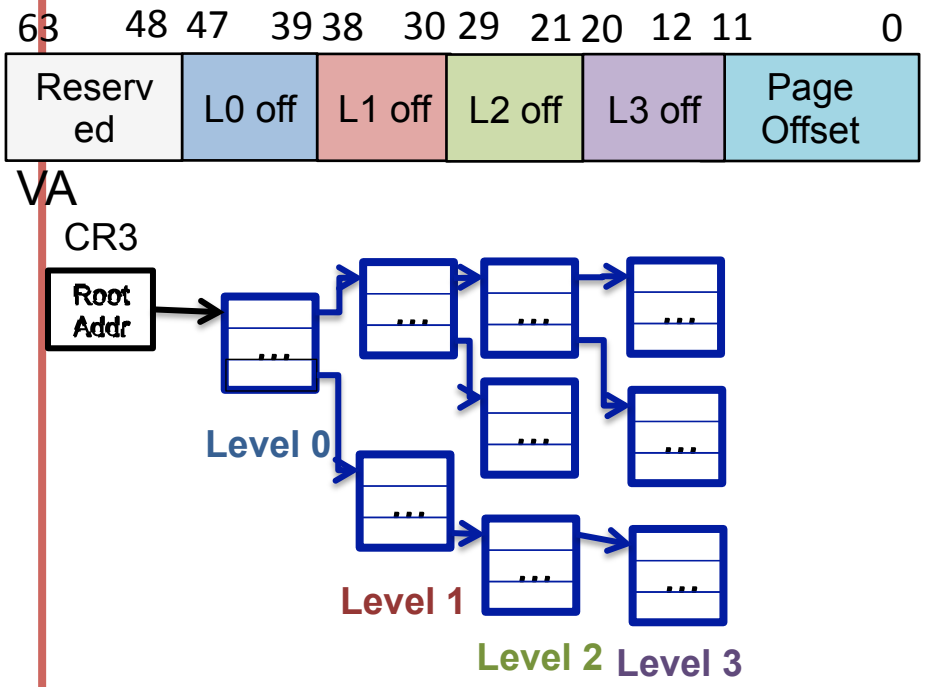
Step 1.2.1 find PPN by walking the page table with VA
e.g, 4 level page table



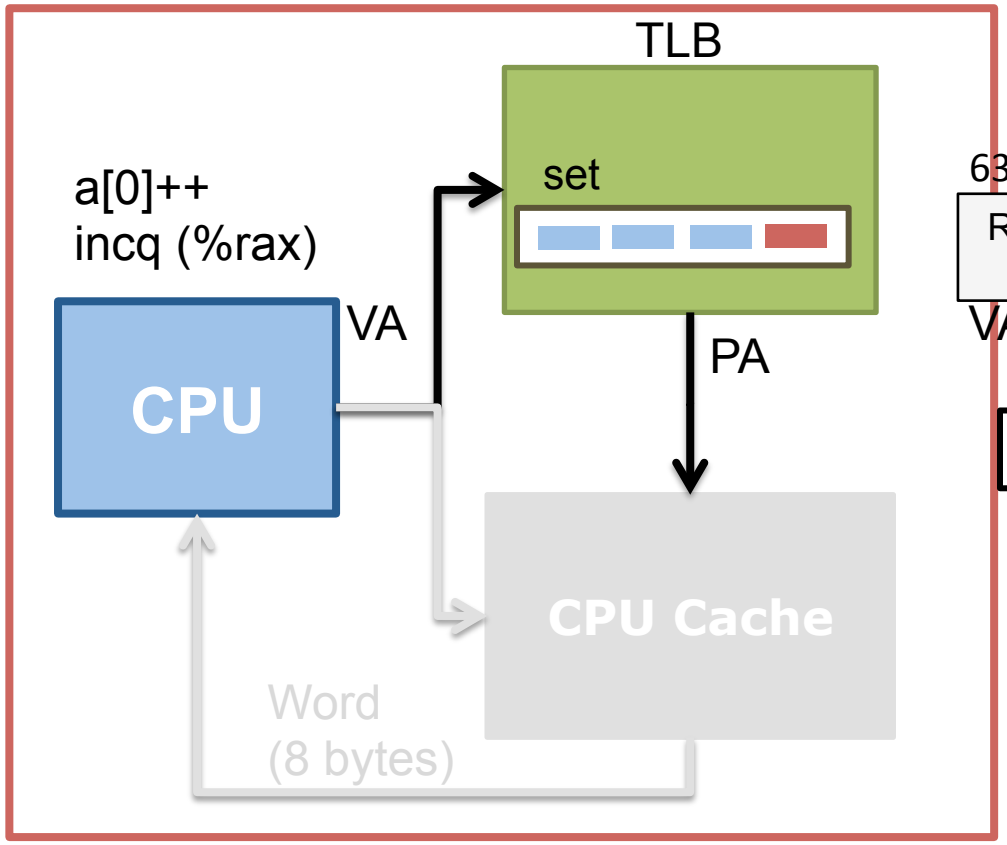
Step 1.2 Walk Through Page Table on TLB Miss

Step 1.2.2 Buffer the VPN->PPN mapping in TLB

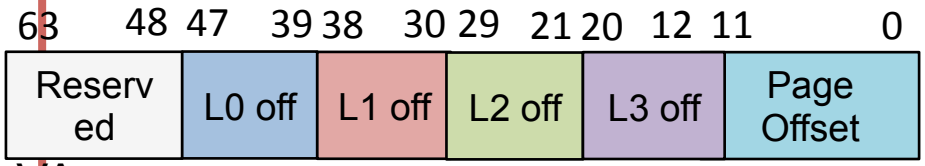


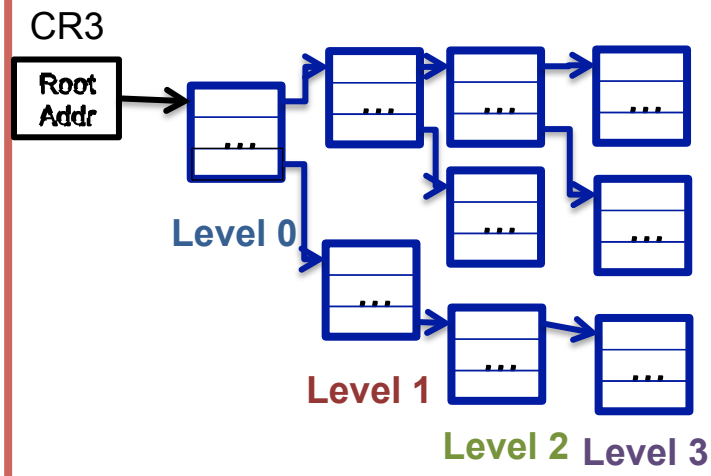


Step 1.2 Walk Through Page Table on TLB Miss



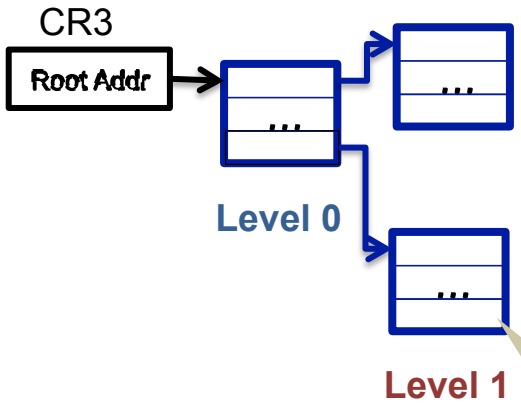
Step 1.2.3 Caculate the physical address
PA = PPN + page_offset

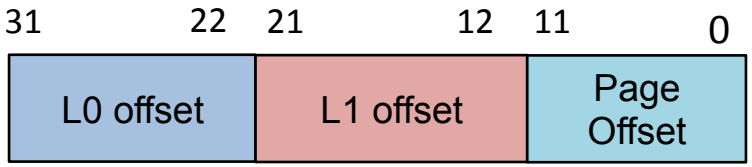




Exercise

32 bit address, 2 level page table, 4K page size

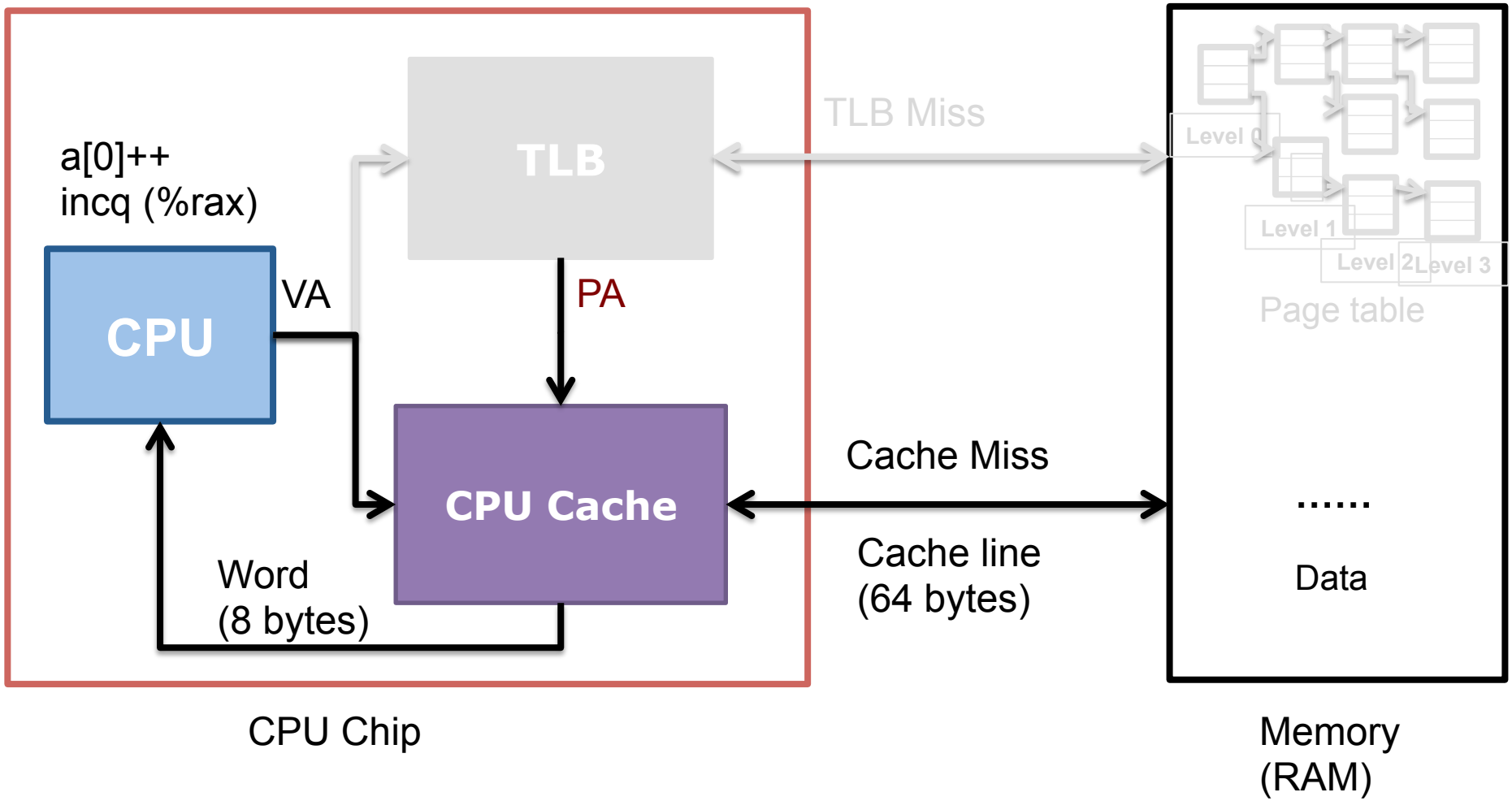




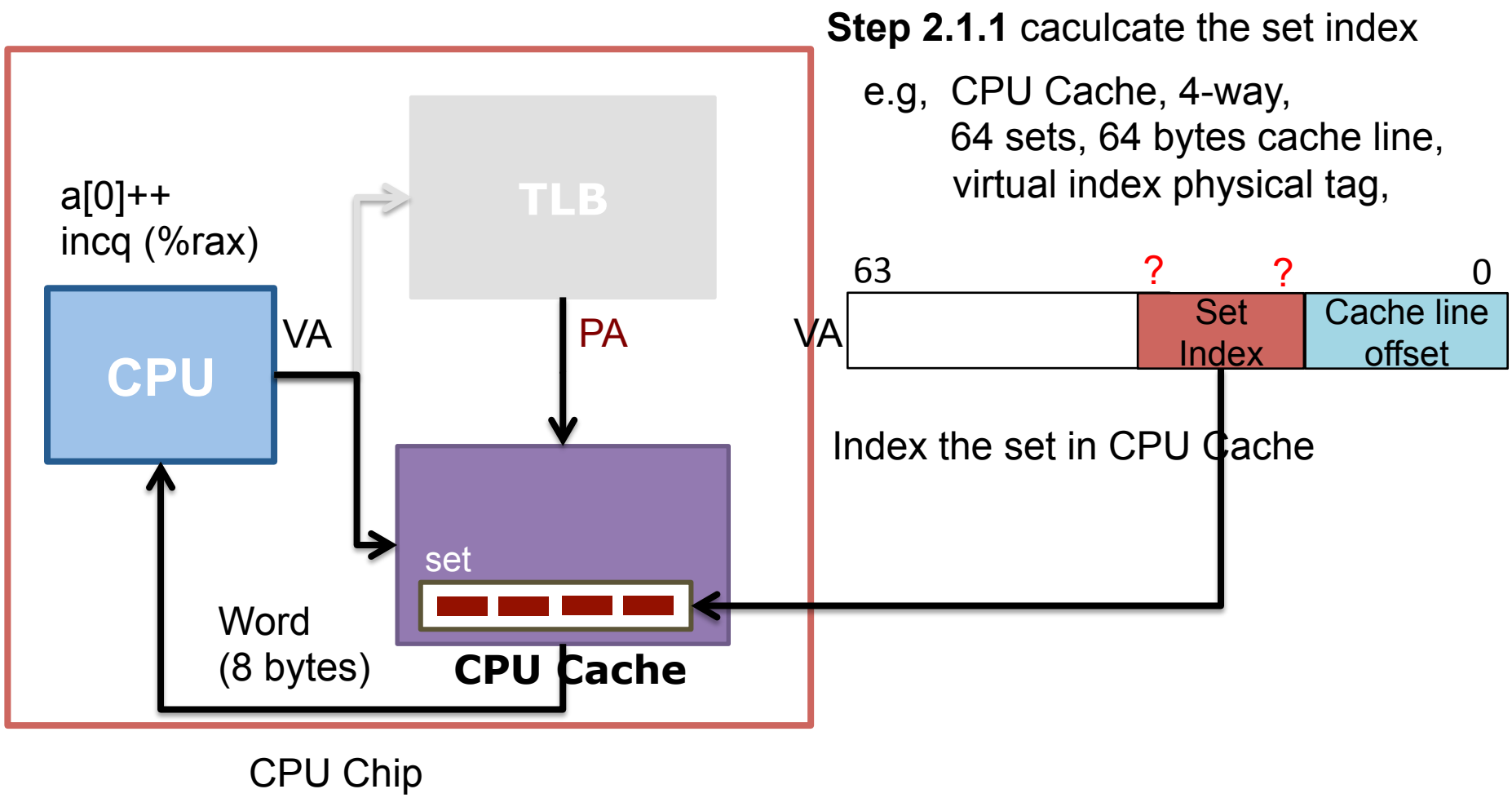
Given each page entry is 4-byte in size, how many entries per 4KB page?

4KB/4B = 2^10 entries

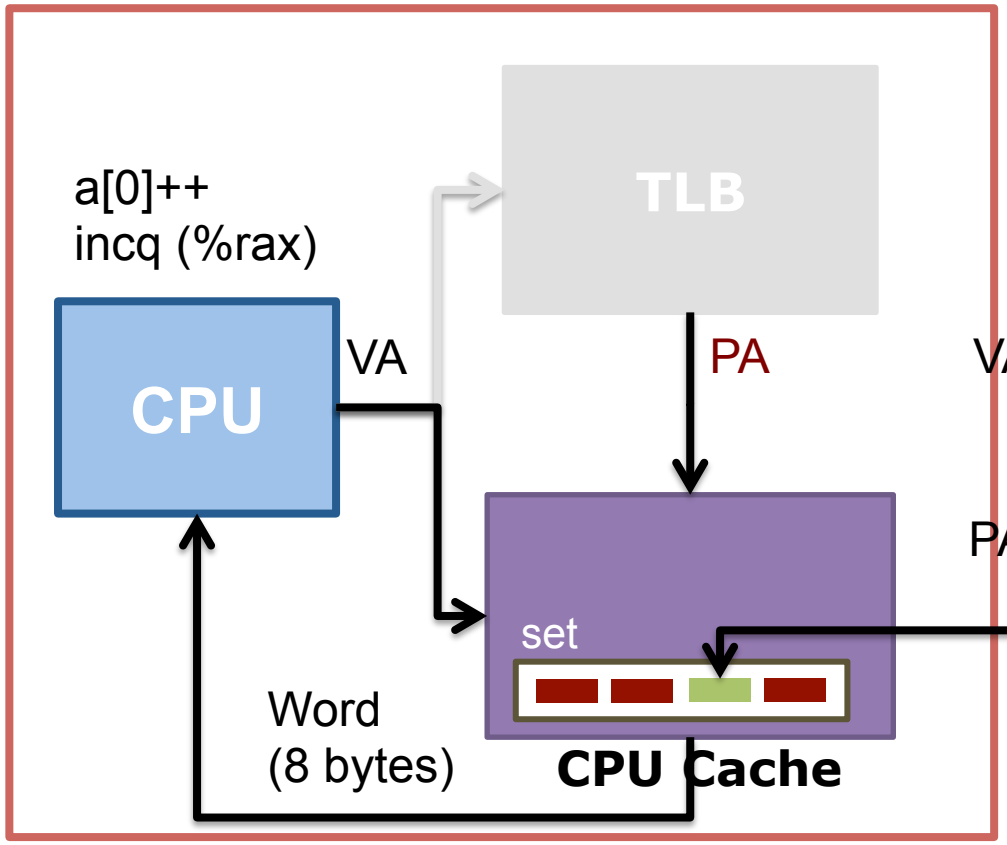
Step 2. Fetch Data



Step 2.1 Fetch Data from CPU Cache

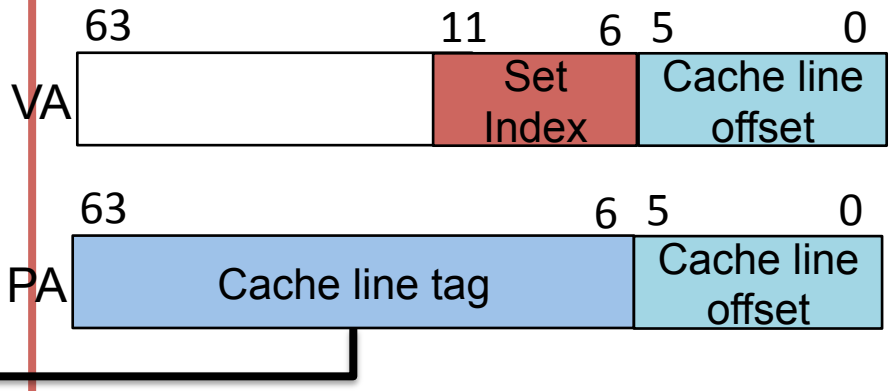


Step 2.1 Fetch Data from CPU Cache

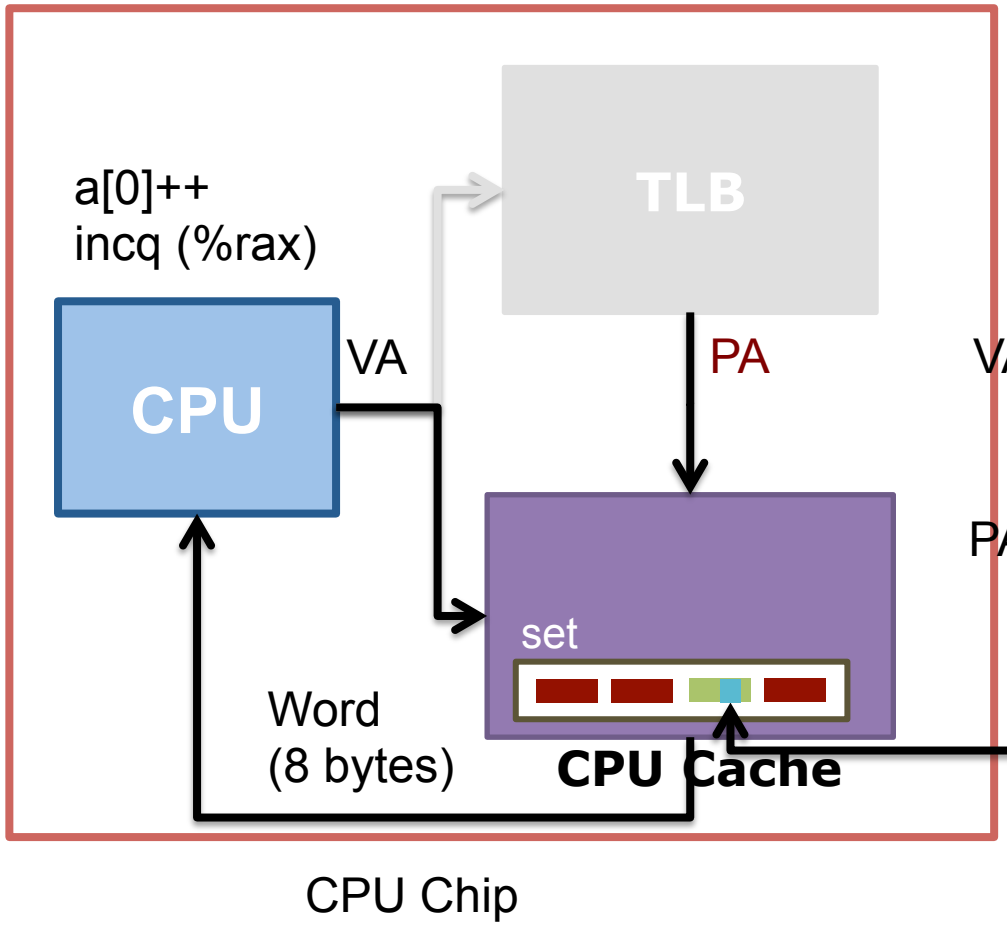


Step 2.1.2 find the buffered cache line by comparing the tag in PA

e.g, CPU Cache, 4 ways,
64 sets, 64 bytes cache line,
virtual index physical tag,

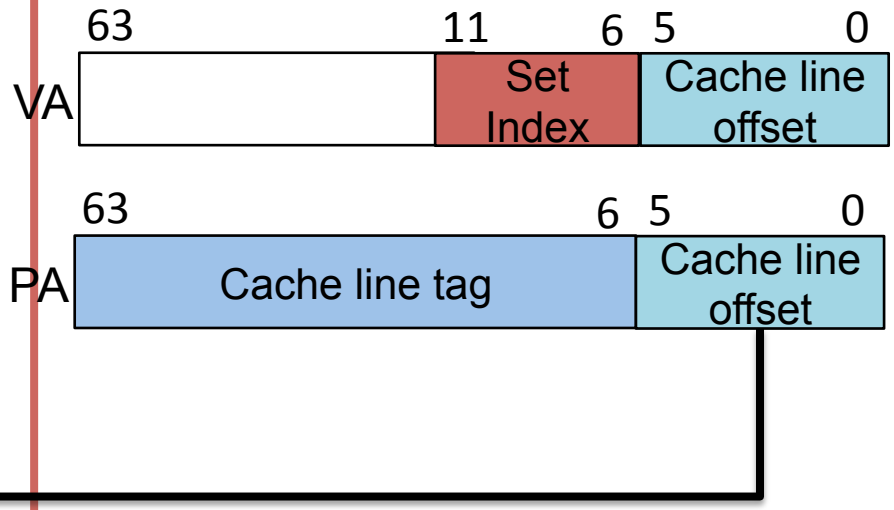


Step 2.1 Fetch Data from CPU Cache



Step 2.1.3 on cache hit, find the data in the cache line using last 6 bits

e.g, CPU Cache, 4 ways,
64 sets, 64 bytes cache line,
virtual index physical tag,



Step 2.2 Fetch Data from Memory on Cache Miss

Step 2.2.1 on cache miss, load 64 bytes data from (PA&(~0x3f))

e.g, CPU Cache, 4 ways,
64 sets, 64 bytes cache line,
virtual index physical tag,

Writing Cache-Friendly Code

Why?

- Programs with lower cache miss rates typically run faster
  - Miss rate: fraction of memory references not found in cache (misses/references)
  - Typical numbers: 3-10% for L1, can be quite small (<1%) for L2, depending on size

How to write cache friendly code?

Memory access pattern

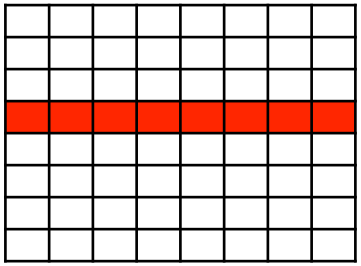
Memory layout

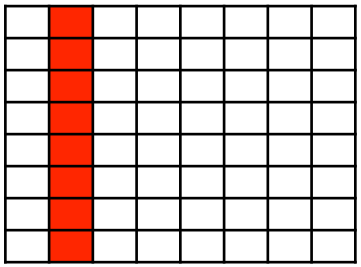
Simple example: sum of 2D array

```
int64 sumarrayrows(int64** a, int r, int c)
{
    int i, j = 0;
    int64 sum = 0

    for (int i = 0 ; i < r; i++)
      for (int j = 0 ; j < c; j++)
          sum += a[i][j];
    return sum ;
}
```

```
int64 sumarraycols(int64** a, int r, int c)
{
    int i, j = 0;
    int64 sum = 0;

    for (int j = 0 ; j < c; j++)
      for (int i = 0 ; i < r; i++)
          sum += a[i][j];
    return sum ;
}
```

Which implementation is more cache friendly?





Simple Example

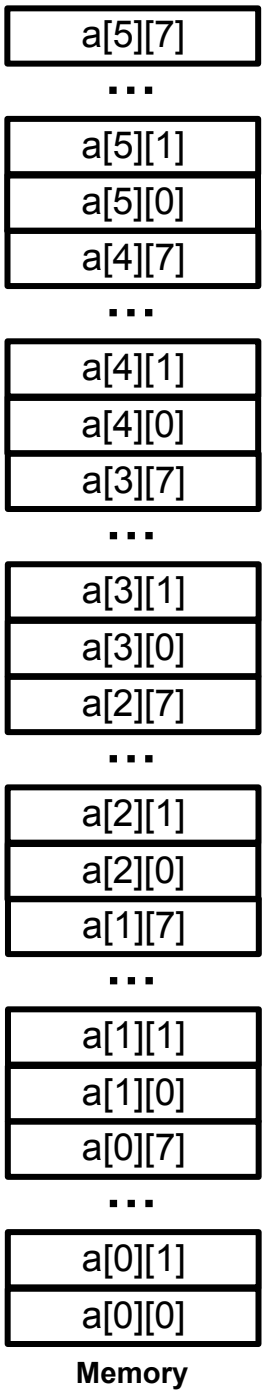
```
int64 sumarrayrows(int64** a, int r, int c)
{
    int i, j = 0;
    int64 sum = 0

    for (int i = 0 ; i < r; i++)
      for (int j = 0 ; j < c; j++)
          sum += a[i][j];
    return sum ;
}
```

```
int64 sumarraycols(int64** a, int r, int c)
{
    int i, j = 0;
    int64 sum = 0;

    for (int j = 0 ; j < c; j++)
      for (int i = 0 ; i < r; i++)
          sum += a[i][j];
    return sum ;
}
```

How many cache misses?

Example:

- CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
- Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
- local variables: i, j, sum are stored in the registers

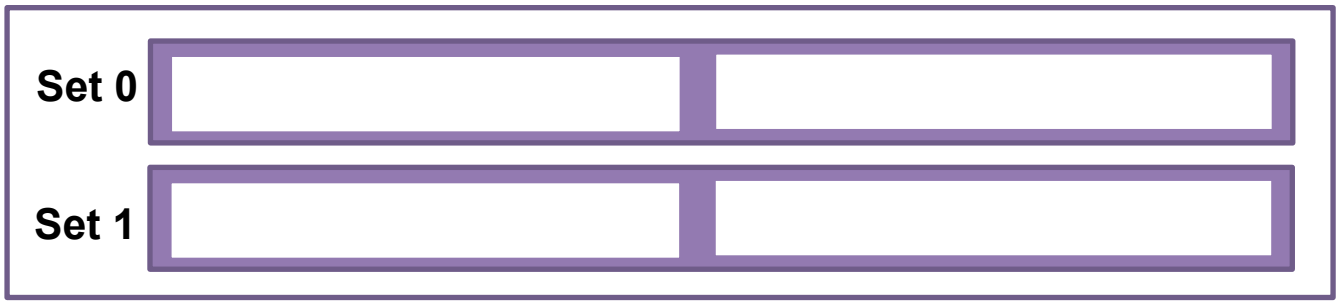


Simple Example

Example:

- CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
- Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
- local variables: i, j, sum are stored in the registers

```
for (int j = 0 ; j < c; j++)
    for (int i = 0 ; i < r; i++)
        sum += a[i][j];
```



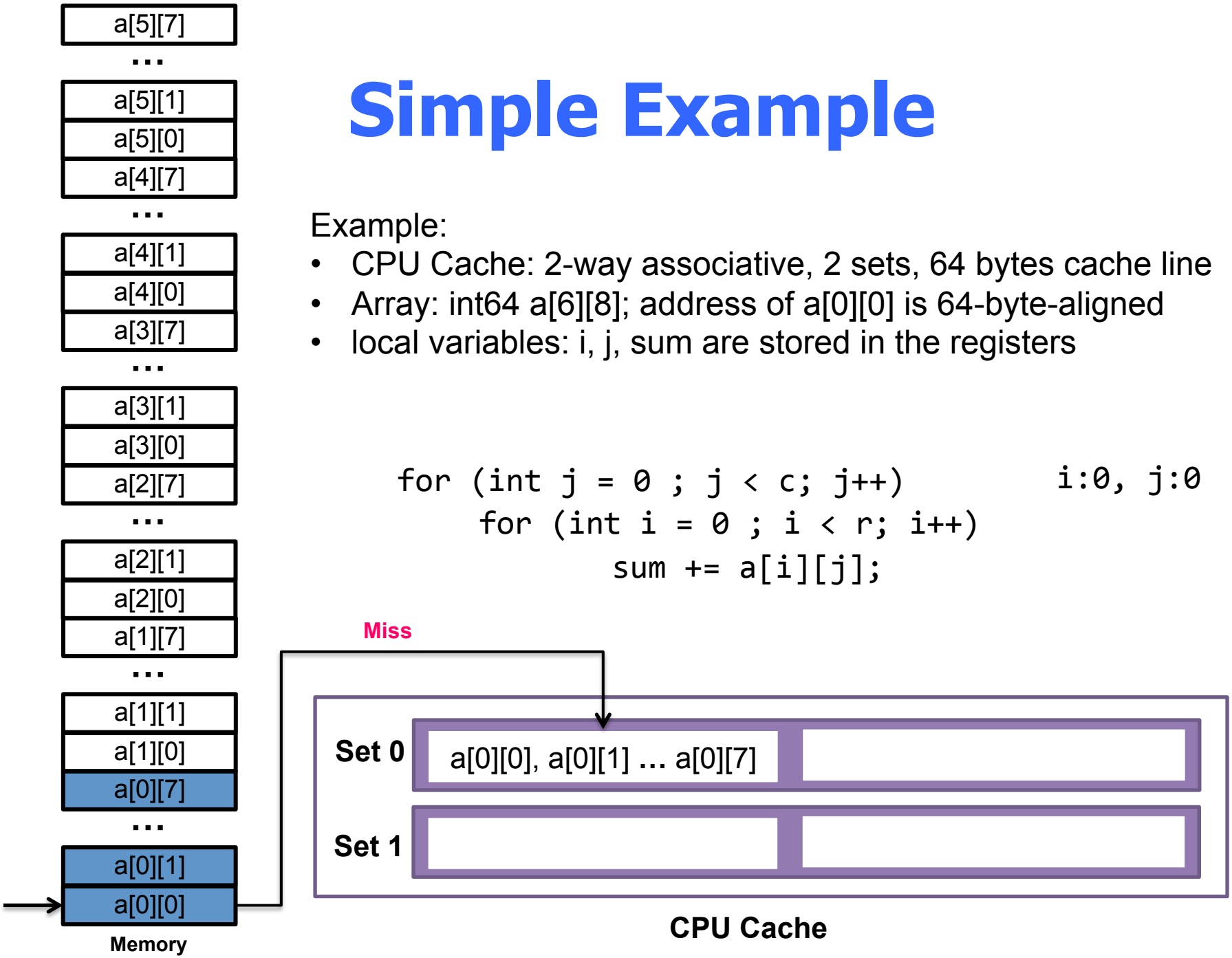
CPU Cache

Simple Example

Example:

- CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
- Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
- local variables: i, j, sum are stored in the registers

```
for (int j = 0 ; j < c; j++)          i:0, j:0
    for (int i = 0 ; i < r; i++)
        sum += a[i][j];
```

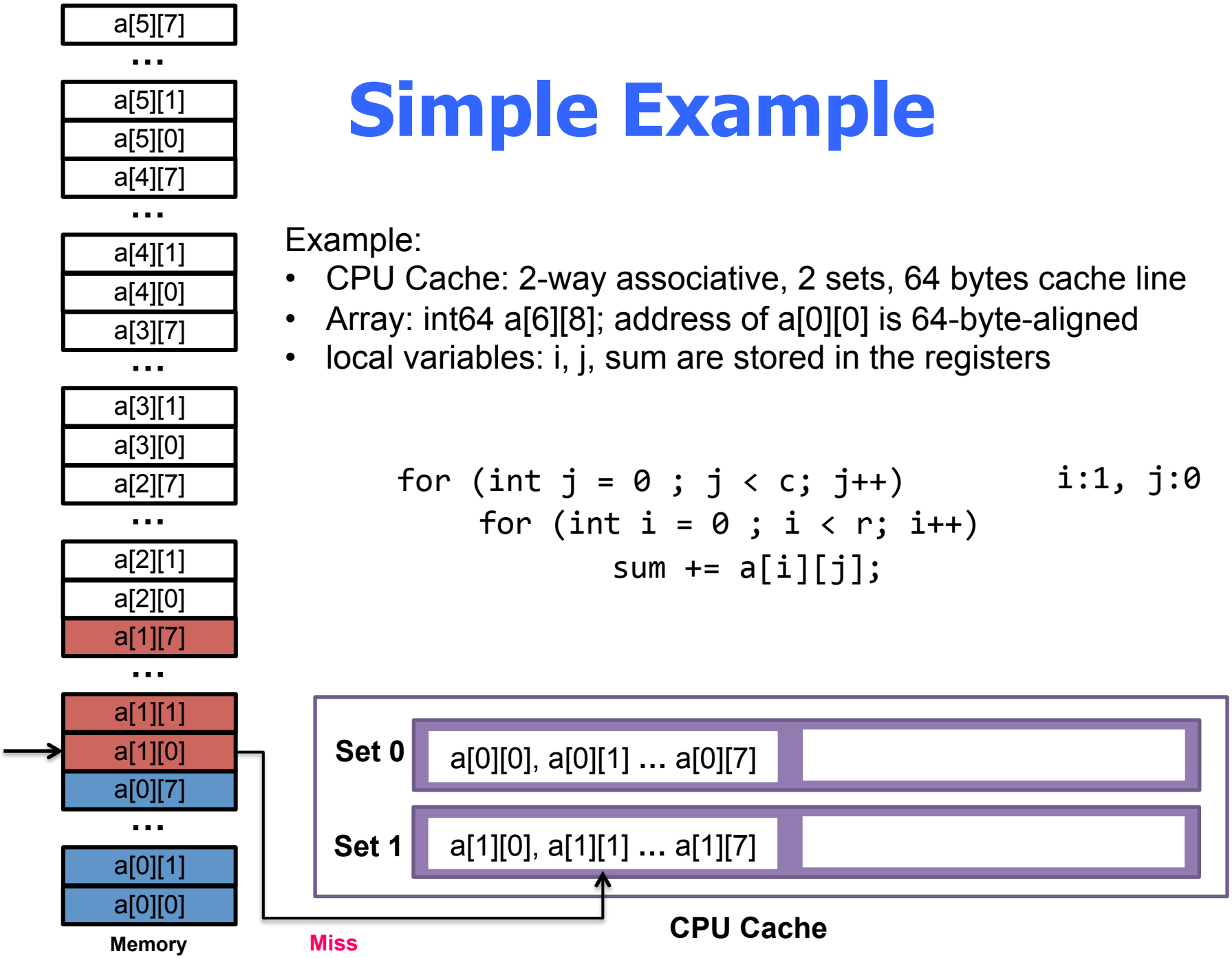


Simple Example

Example:

- CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
- Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
- local variables: i, j, sum are stored in the registers

```
for (int j = 0 ; j < c; j++)          i:1, j:0
    for (int i = 0 ; i < r; i++)
        sum += a[i][j];
```



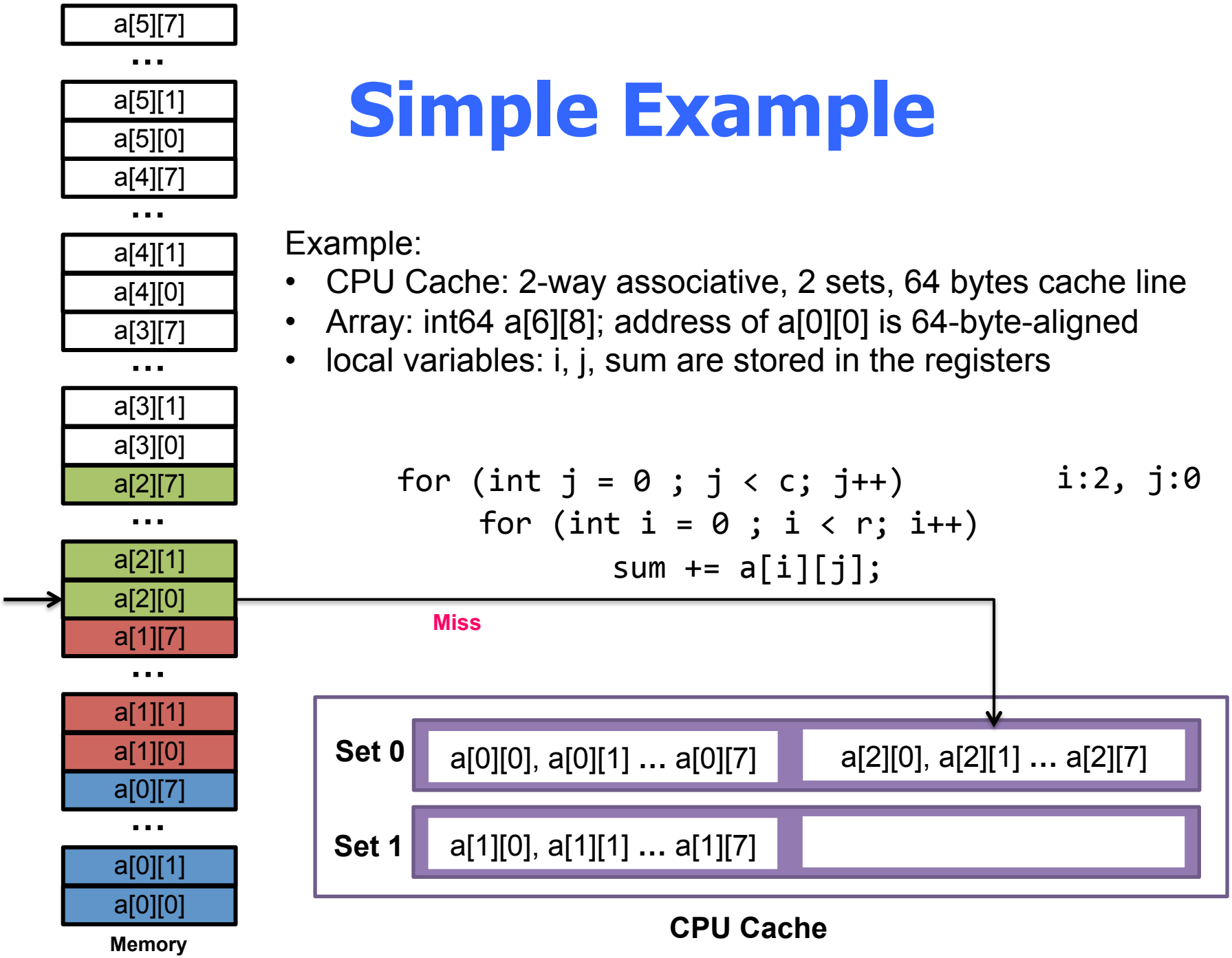
Simple Example

Example:

- CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
- Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
- local variables: i, j, sum are stored in the registers

```
for (int j = 0 ; j < c; j++)
    for (int i = 0 ; i < r; i++)
        sum += a[i][j];
```

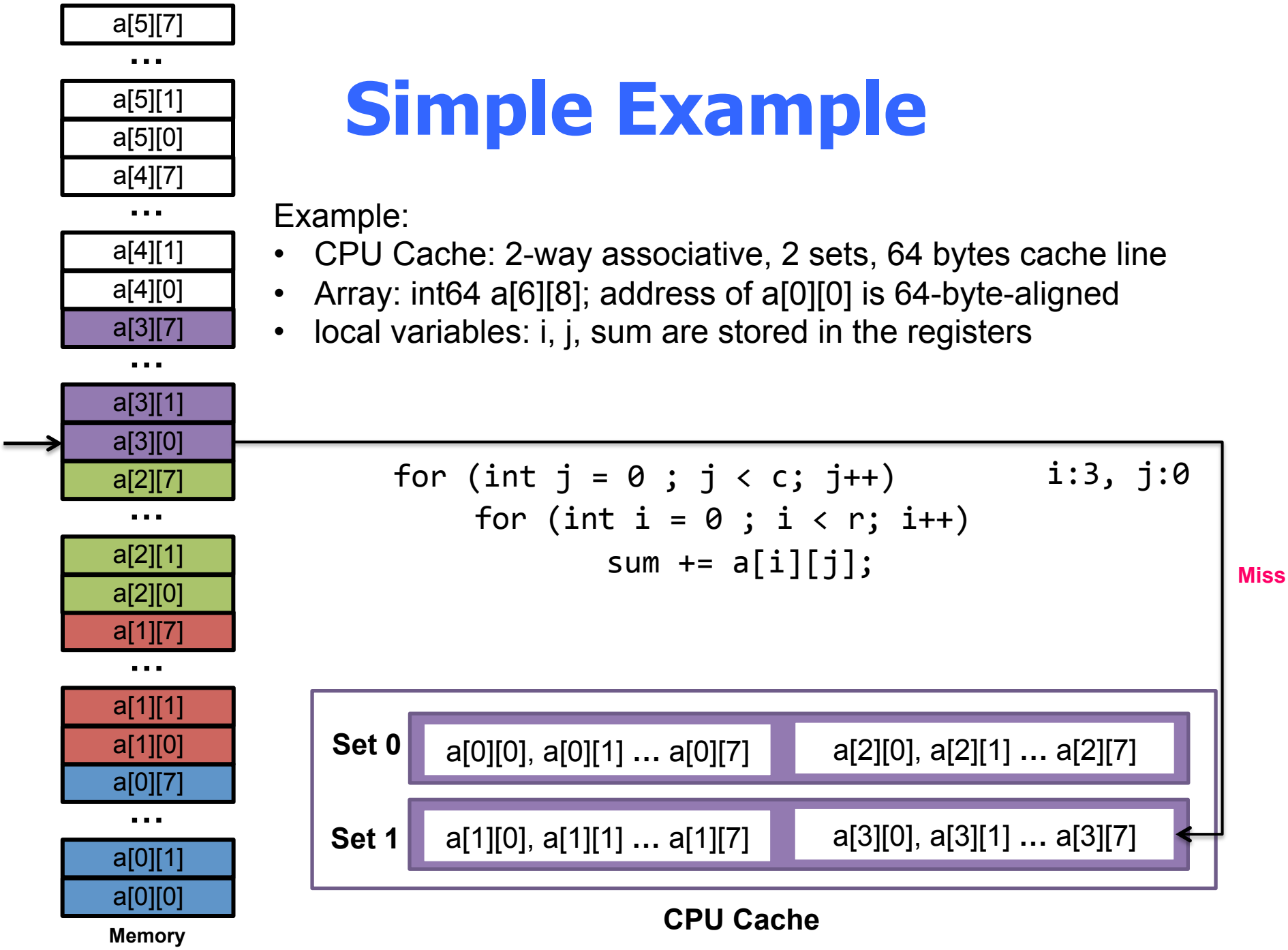
i:2, j:0



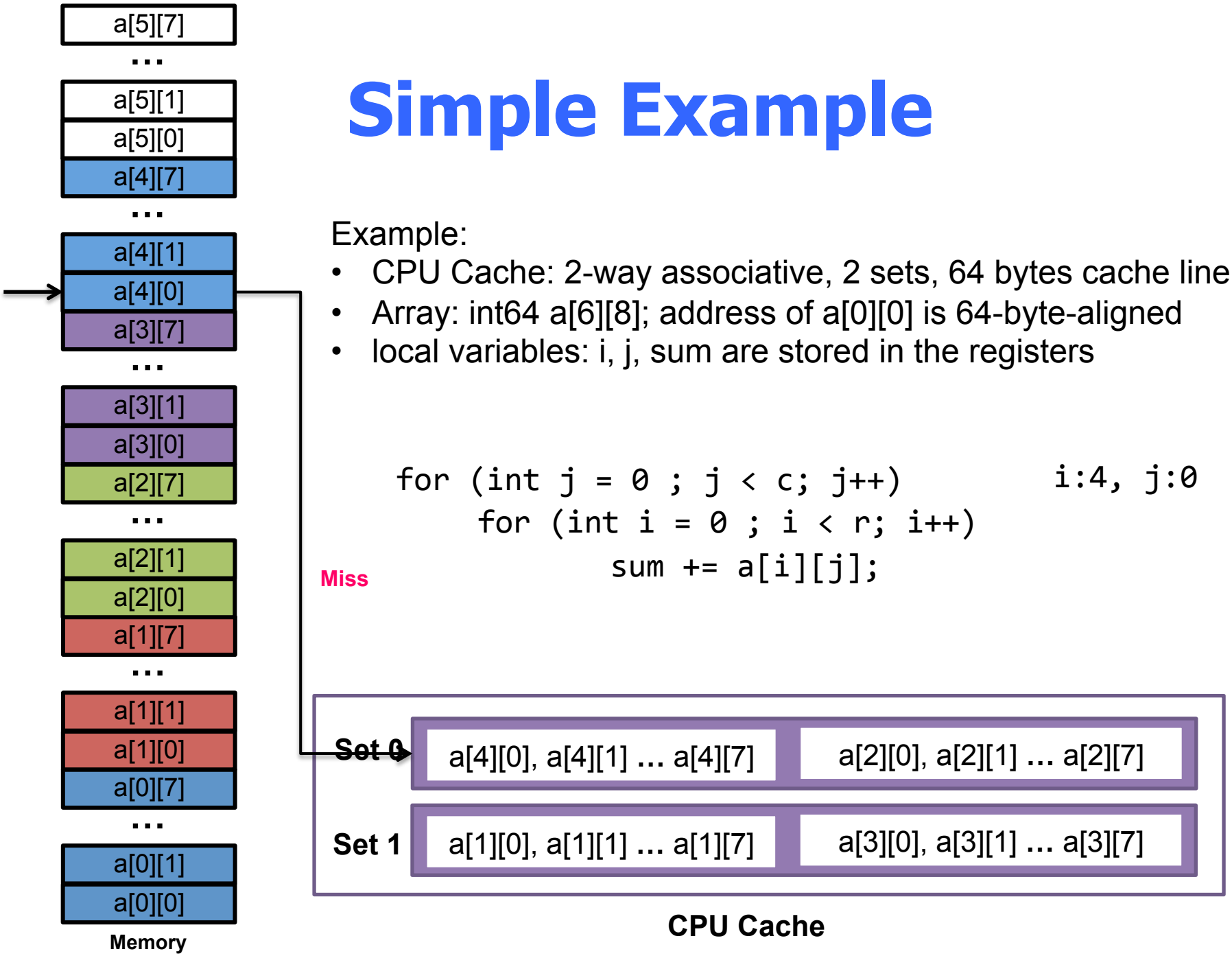
Simple Example

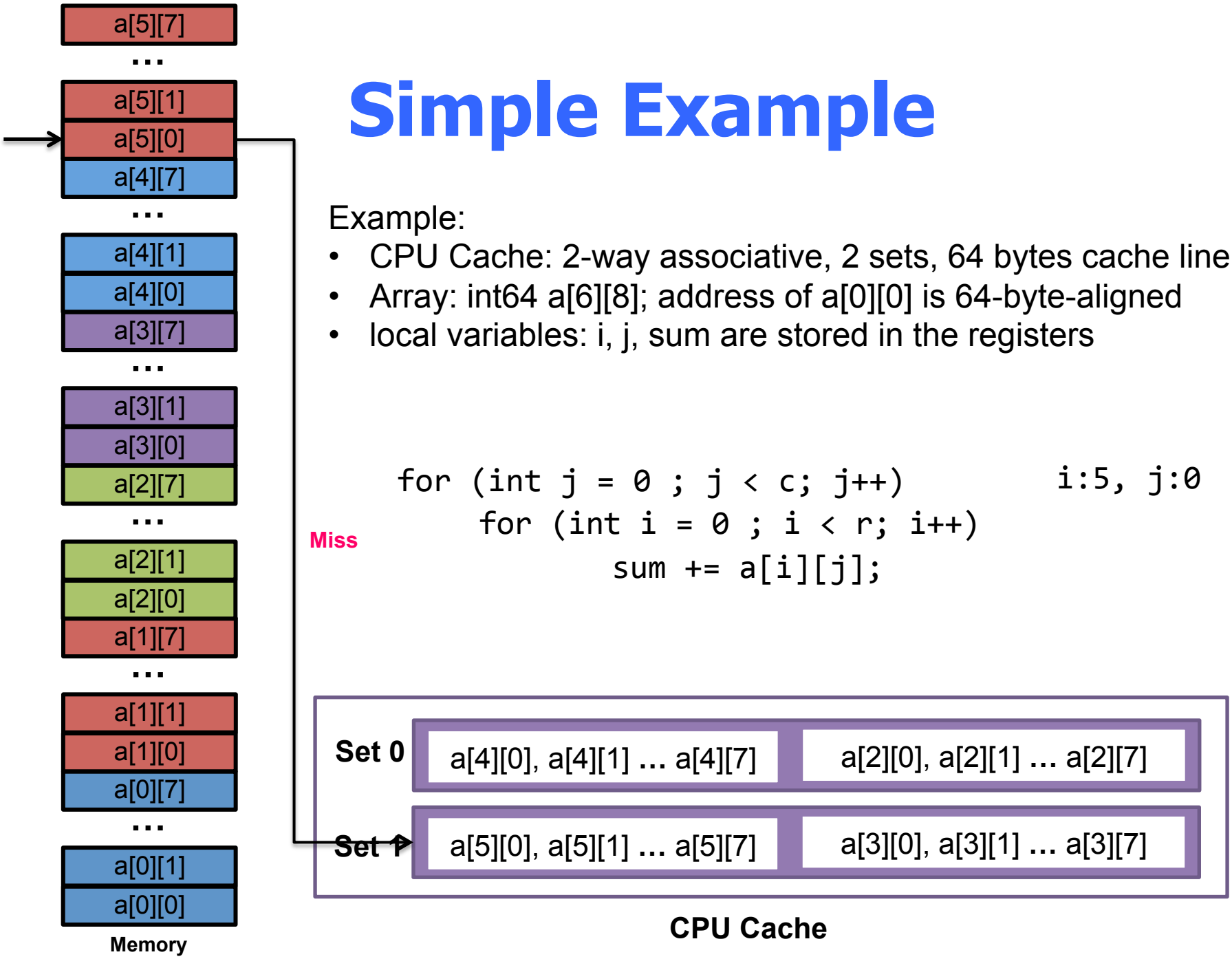
Example:

- CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
- Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
- local variables: i, j, sum are stored in the registers



Simple Example



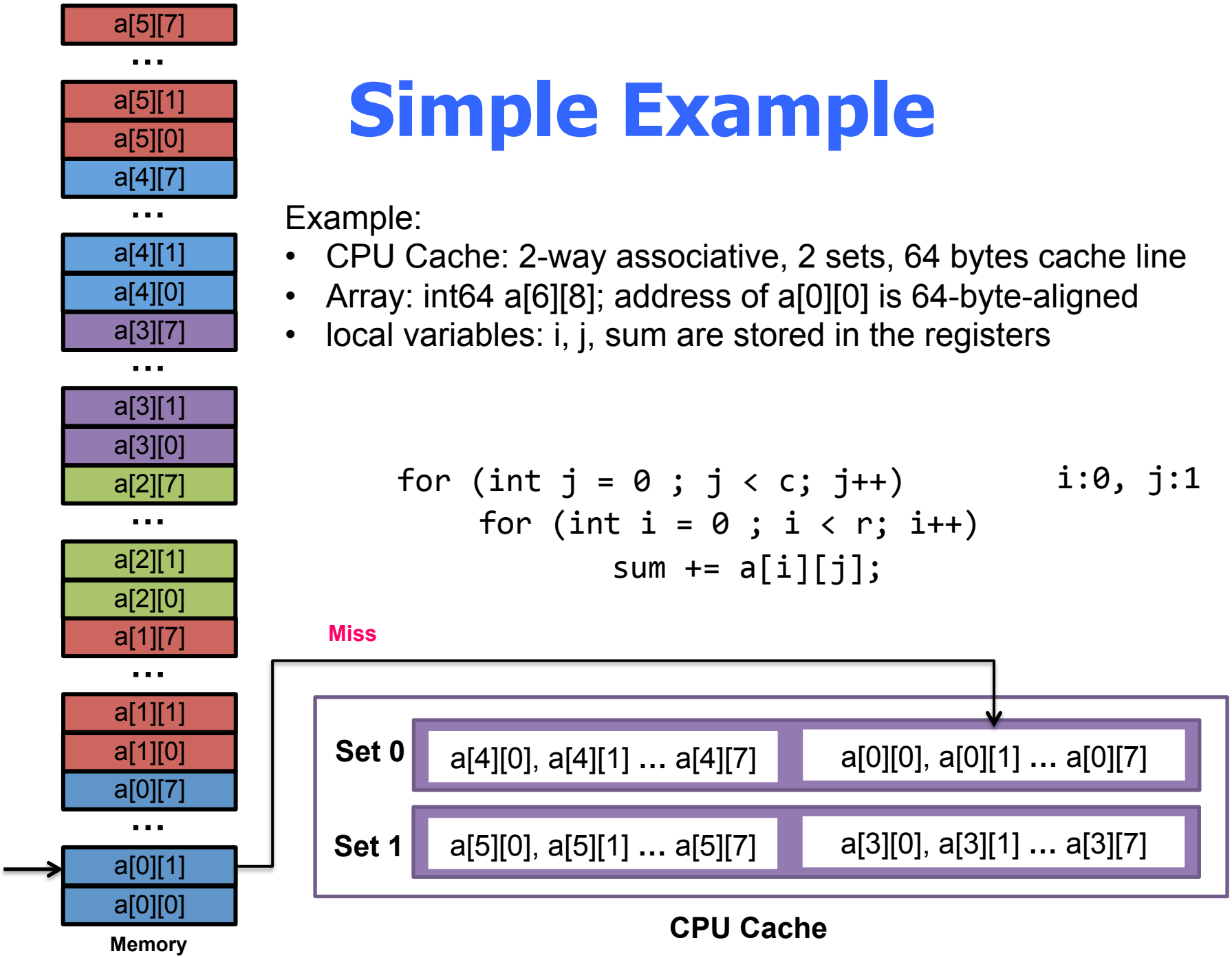

a[5][7]
...
a[5][1]
a[5][0]
a[4][7]
...
a[4][1]
a[4][0]
a[3][7]
...
a[3][1]
a[3][0]
a[2][7]
...
a[2][1]
a[2][0]
a[1][7]
...
a[1][1]
a[1][0]
a[0][7]
...
a[0][1]
a[0][0]
Memory
Simple Example
Example:
• CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
• Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
• local variables: i, j, sum are stored in the registers
for (int j = 0 ; j < c; j++)
for (int i = 0 ; i < r; i++)
sum += a[i][j];
i:5, j:0
Miss
Set 0
a[4][0], a[4][1] ... a[4][7]
a[2][0], a[2][1] ... a[2][7]
Set 1
a[5][0], a[5][1] ... a[5][7]
a[3][0], a[3][1] ... a[3][7]
CPU Cache

Simple Example

Example:

- CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
- Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
- local variables: i, j, sum are stored in the registers

```
for (int j = 0 ; j < c; j++)          i:0, j:1
    for (int i = 0 ; i < r; i++)
        sum += a[i][j];
```

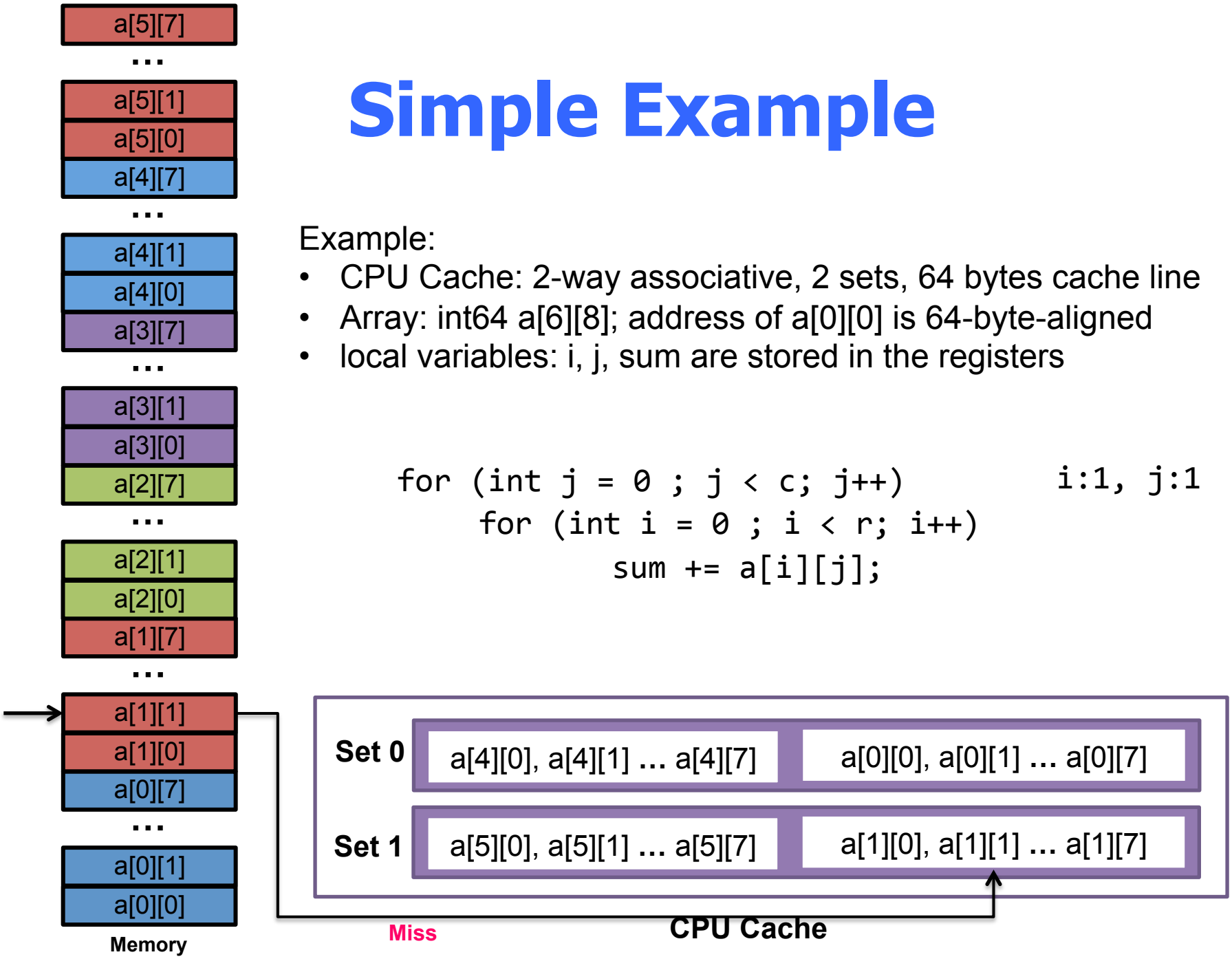


Simple Example

Example:

- CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
- Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
- local variables: i, j, sum are stored in the registers

```
for (int j = 0 ; j < c; j++)          i:1, j:1
    for (int i = 0 ; i < r; i++)
        sum += a[i][j];
```

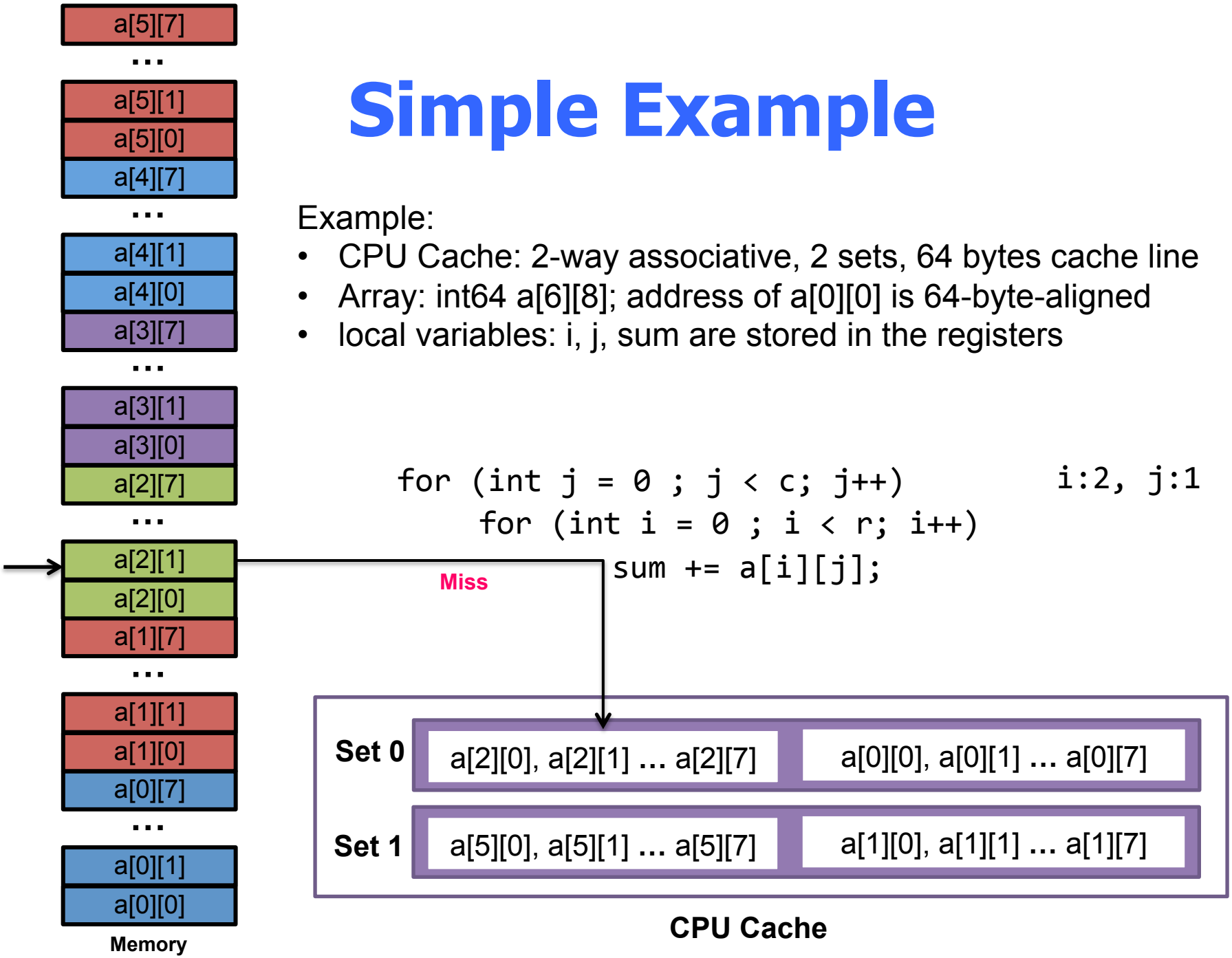


Simple Example

Example:

- CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
- Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
- local variables: i, j, sum are stored in the registers

```
for (int j = 0 ; j < c; j++)          i:2, j:1
    for (int i = 0 ; i < r; i++)
        sum += a[i][j];
```



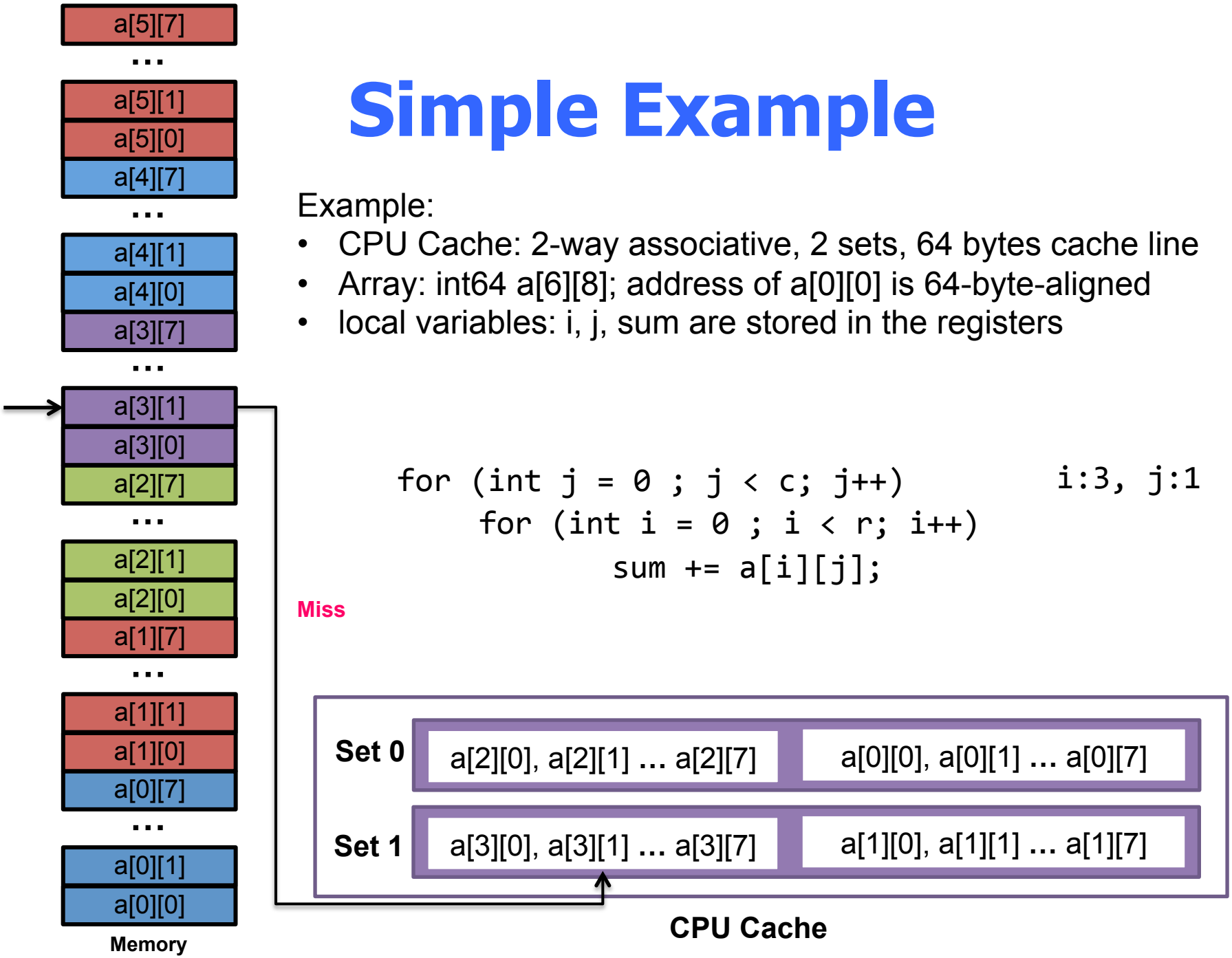
Simple Example

Example:

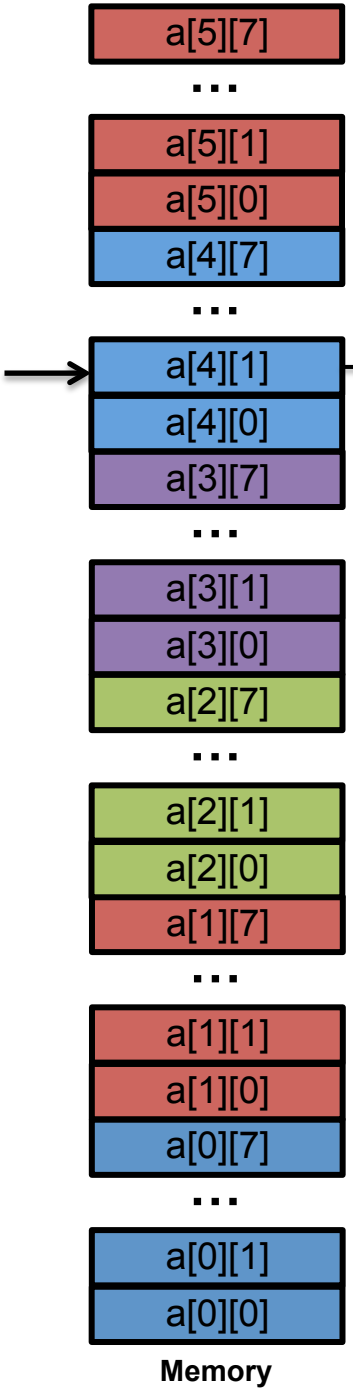
- CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
- Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
- local variables: i, j, sum are stored in the registers

```
for (int j = 0 ; j < c; j++)          i:3, j:1
    for (int i = 0 ; i < r; i++)
        sum += a[i][j];
```

Miss



Simple Example



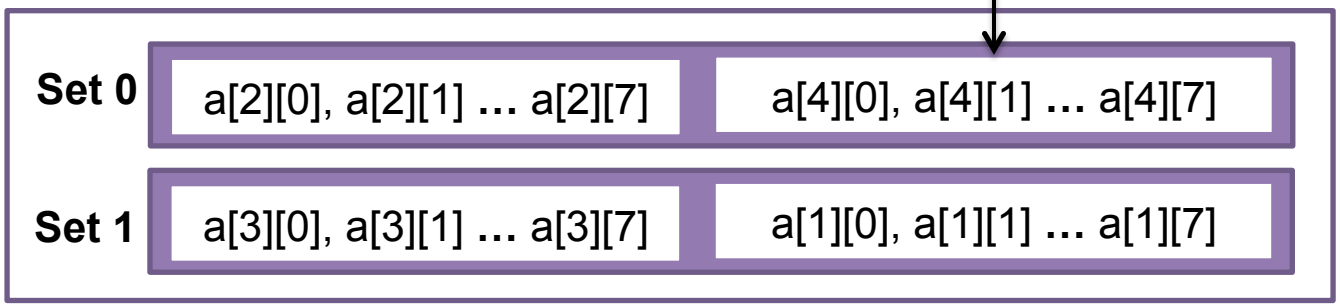
Example:

- CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
- Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
- local variables: i, j, sum are stored in the registers

```
for (int j = 0 ; j < c; j++)
    for (int i = 0 ; i < r; i++)
        sum += a[i][j];
```

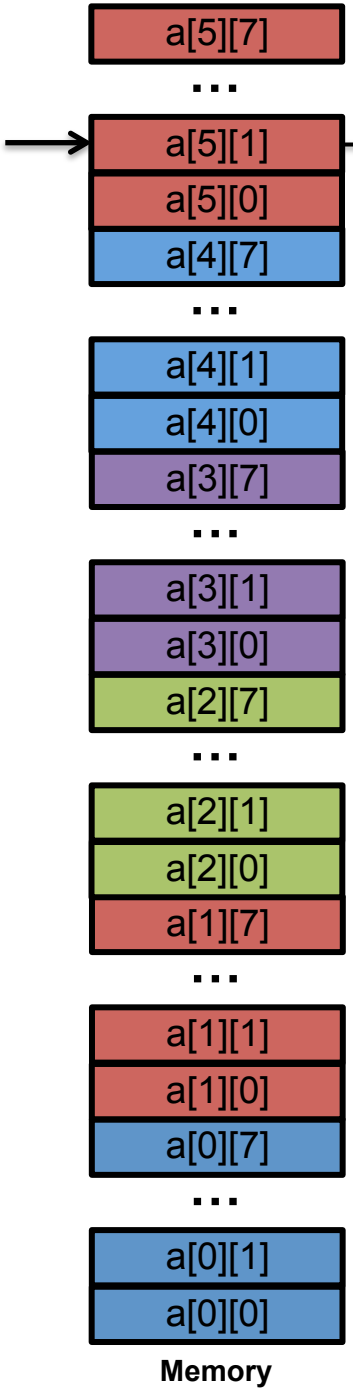
i:4, j:1

Miss



CPU Cache

Simple Example

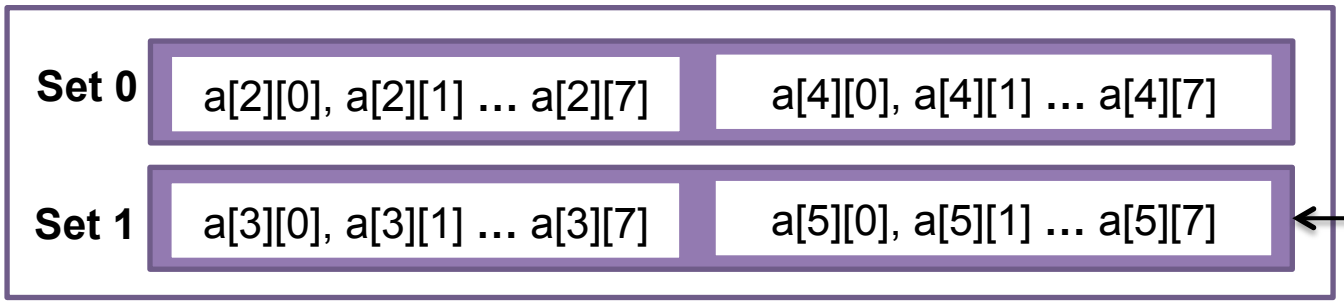


Example:

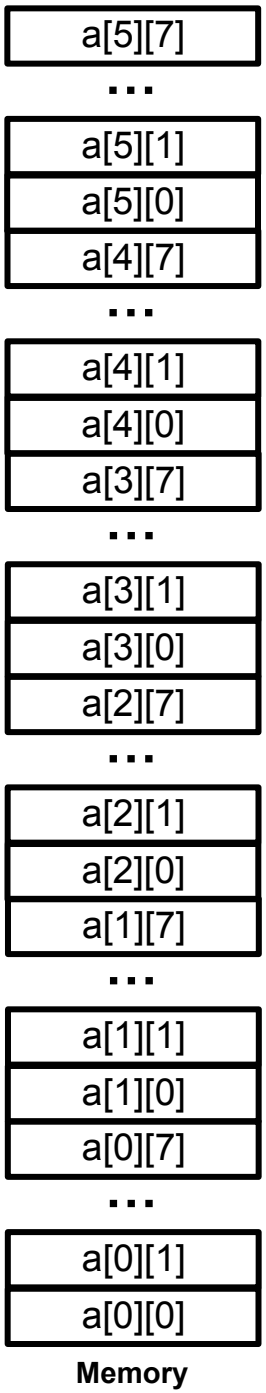
- CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
- Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
- local variables: i, j, sum are stored in the registers

```
for (int j = 0 ; j < c; j++)          i:5, j:1
    for (int i = 0 ; i < r; i++)
        sum += a[i][j];
```

Miss



CPU Cache

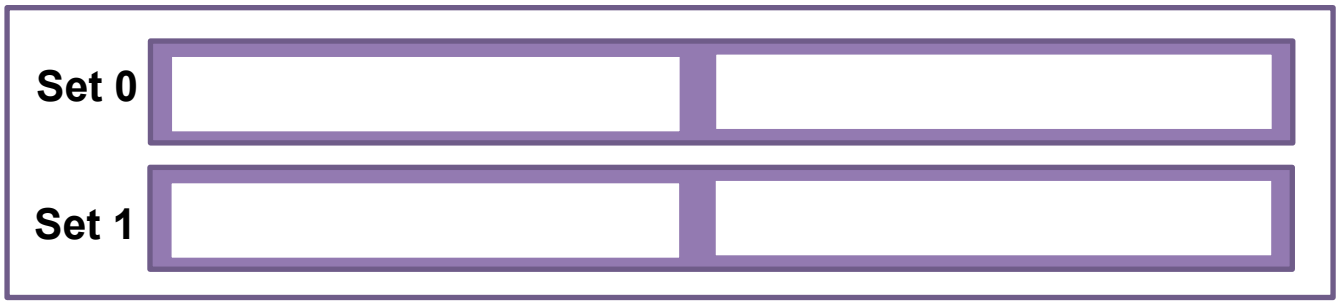


Simple Example

Example:

- CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
- Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
- local variables: i, j, sum are stored in the registers

```
for (int i = 0 ; i < r; i++)
    for (int j = 0 ; j < c; j++)
        sum += a[i][j];
```

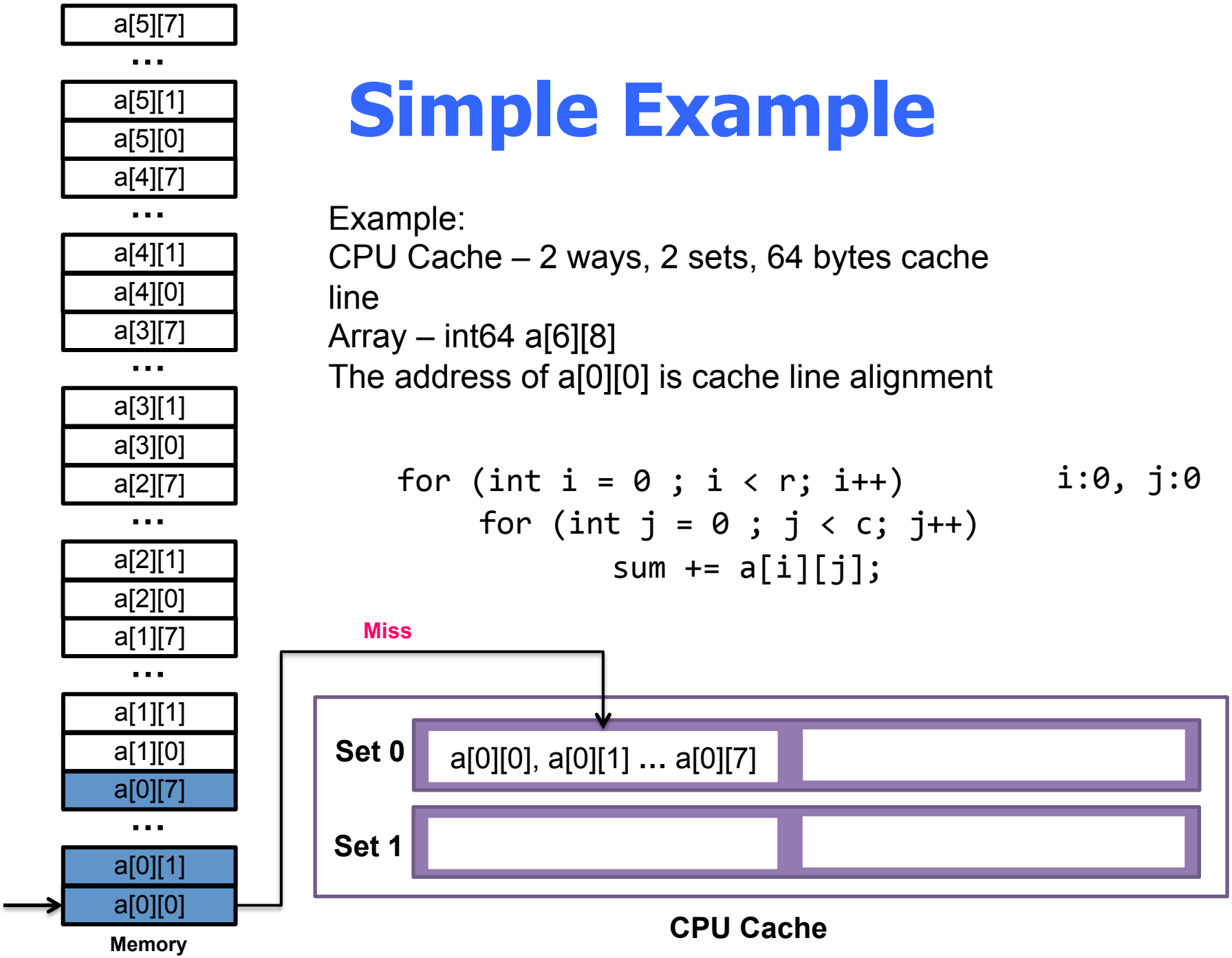


CPU Cache

Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```
for (int i = 0 ; i < r; i++)          i:0, j:0
    for (int j = 0 ; j < c; j++)
        sum += a[i][j];
```

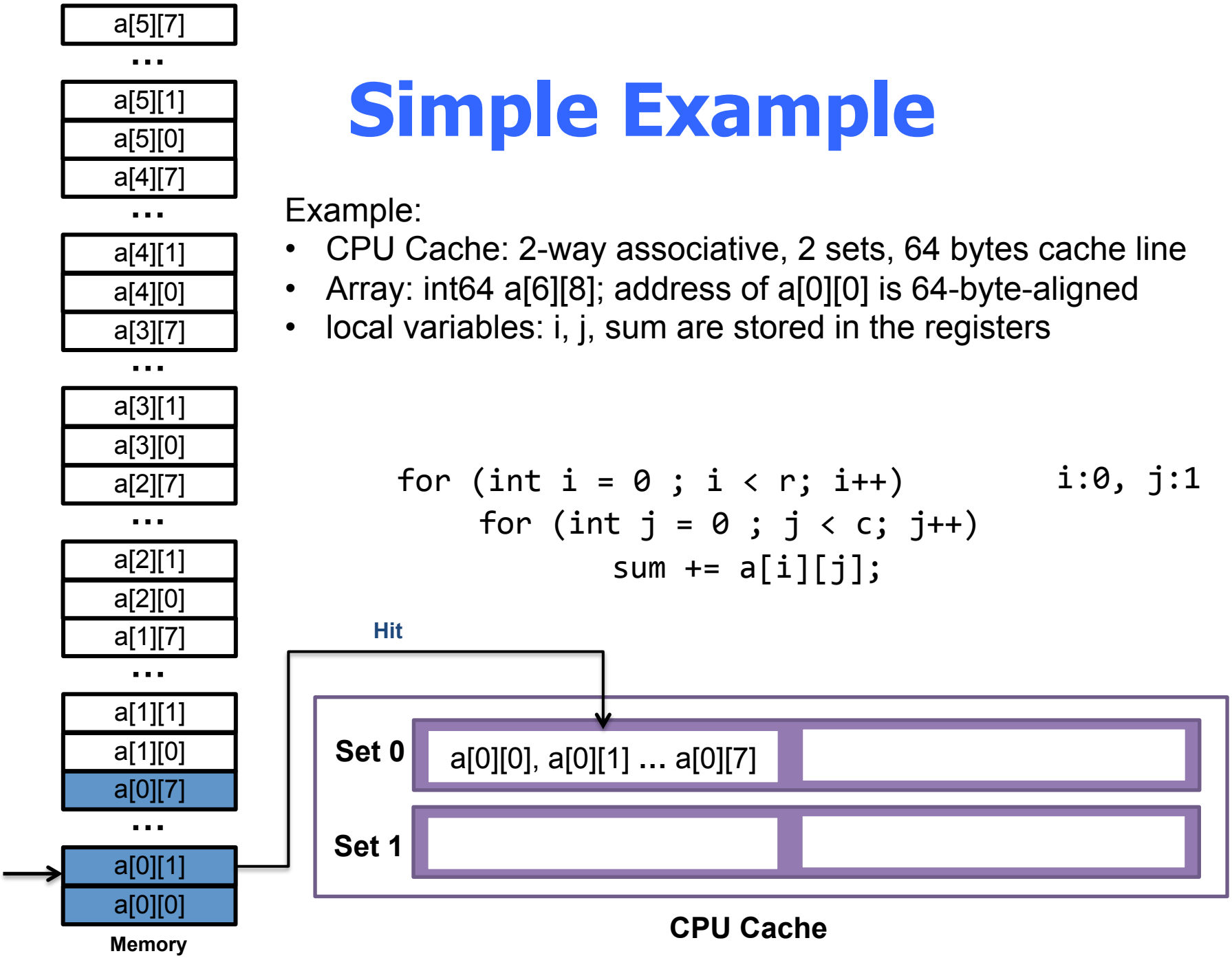


Simple Example

Example:

- CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
- Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
- local variables: i, j, sum are stored in the registers

```
for (int i = 0 ; i < r; i++)          i:0, j:1
    for (int j = 0 ; j < c; j++)
        sum += a[i][j];
```

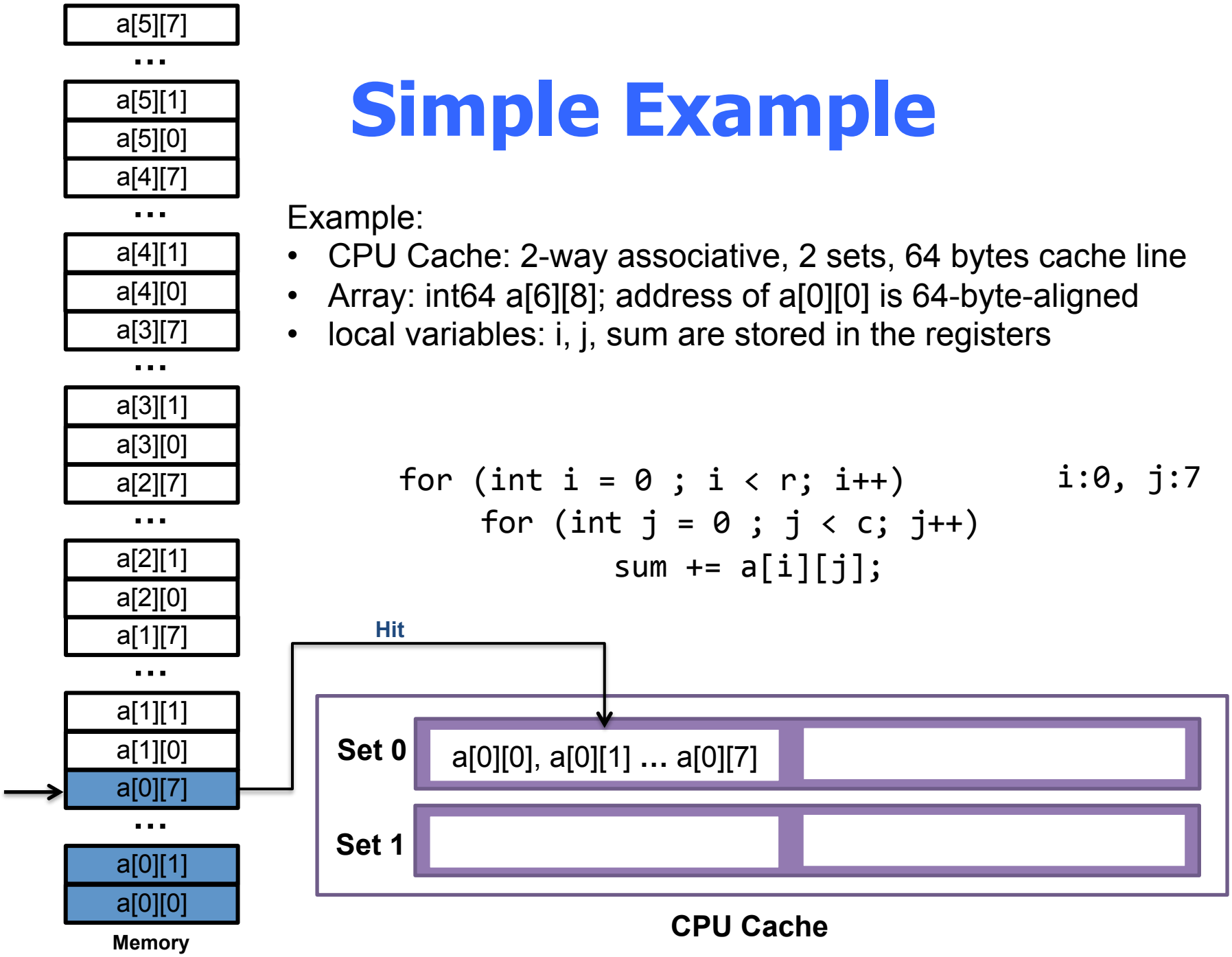


Simple Example

Example:

- CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
- Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
- local variables: i, j, sum are stored in the registers

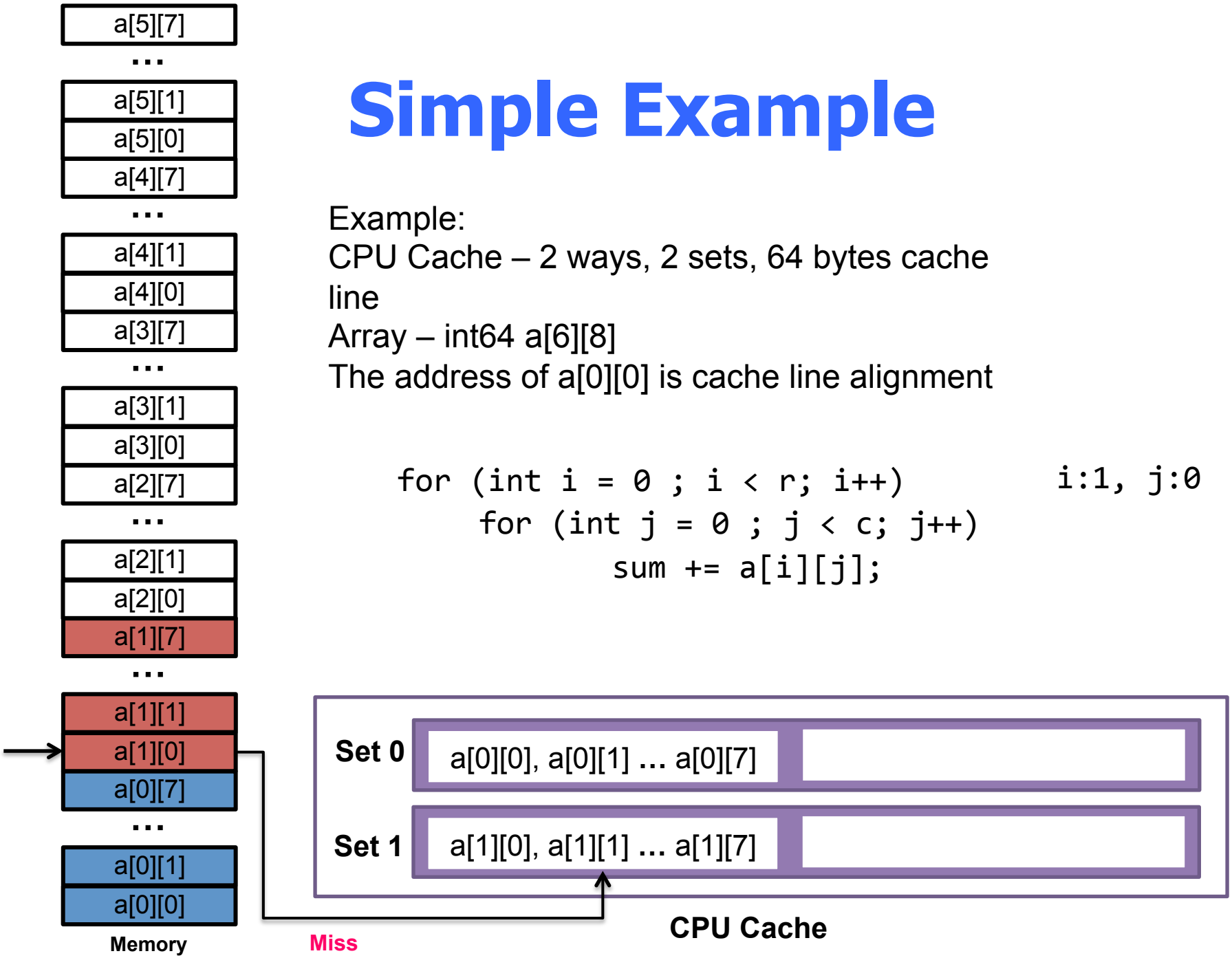
```
for (int i = 0 ; i < r; i++)          i:0, j:7
    for (int j = 0 ; j < c; j++)
        sum += a[i][j];
```



Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```
for (int i = 0 ; i < r; i++)          i:1, j:0
    for (int j = 0 ; j < c; j++)
        sum += a[i][j];
```

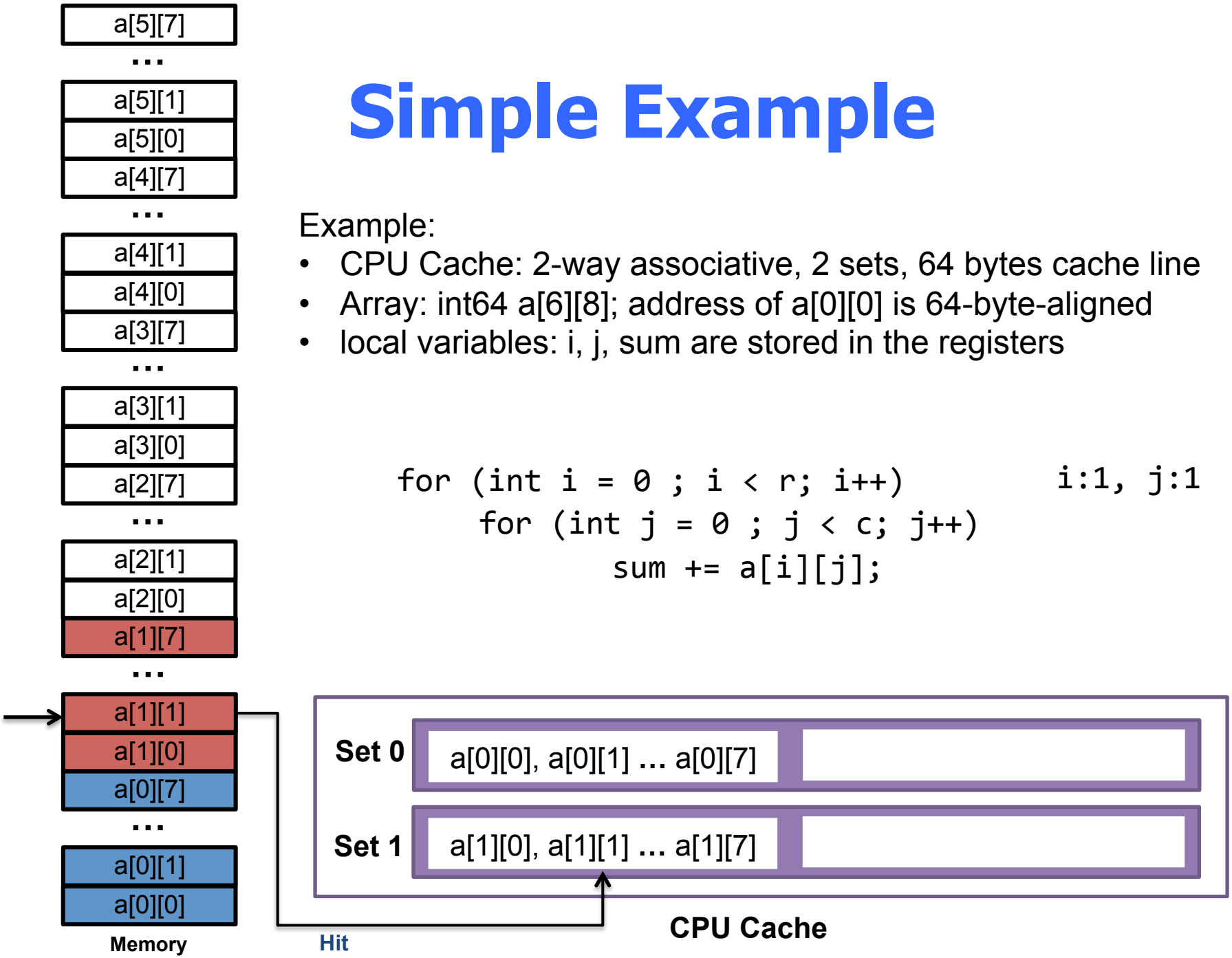


Simple Example

Example:

- CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
- Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
- local variables: i, j, sum are stored in the registers

```
for (int i = 0 ; i < r; i++)          i:1, j:1
    for (int j = 0 ; j < c; j++)
        sum += a[i][j];
```

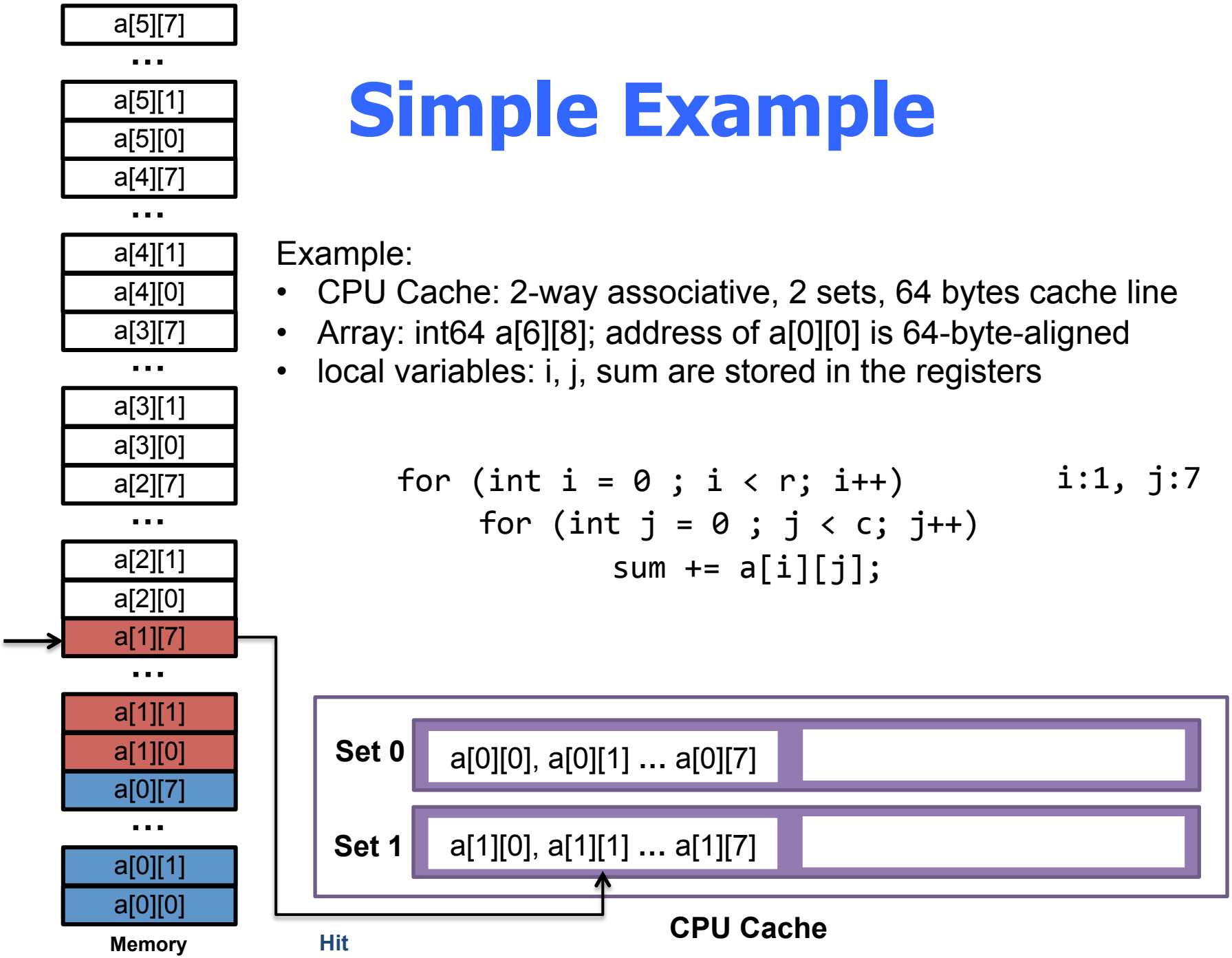


Simple Example

Example:

- CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
- Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
- local variables: i, j, sum are stored in the registers

```
for (int i = 0 ; i < r; i++)          i:1, j:7
    for (int j = 0 ; j < c; j++)
        sum += a[i][j];
```



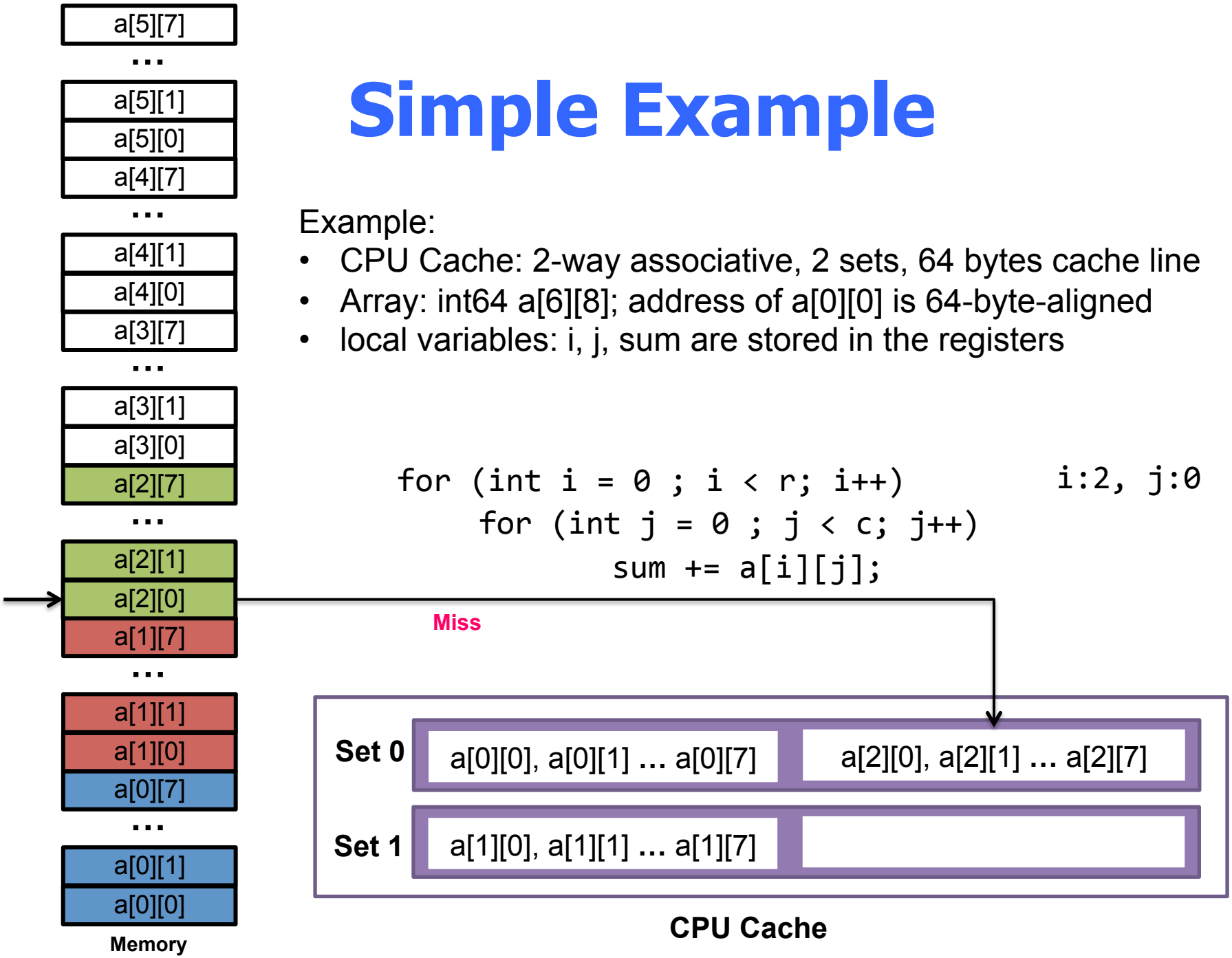
Simple Example

Example:

- CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
- Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
- local variables: i, j, sum are stored in the registers

```
for (int i = 0 ; i < r; i++)
    for (int j = 0 ; j < c; j++)
        sum += a[i][j];
```

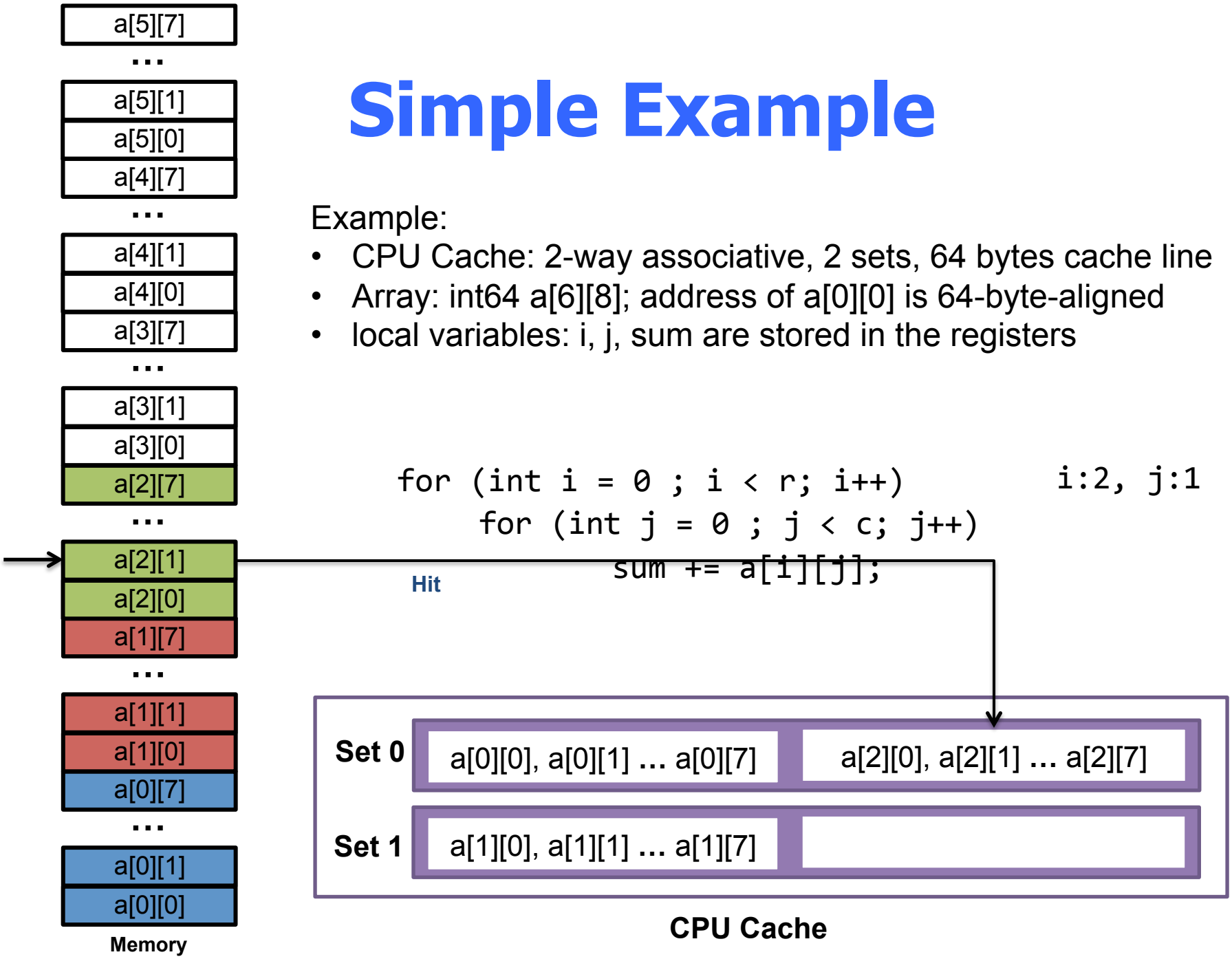
i:2, j:0



Simple Example

Example:

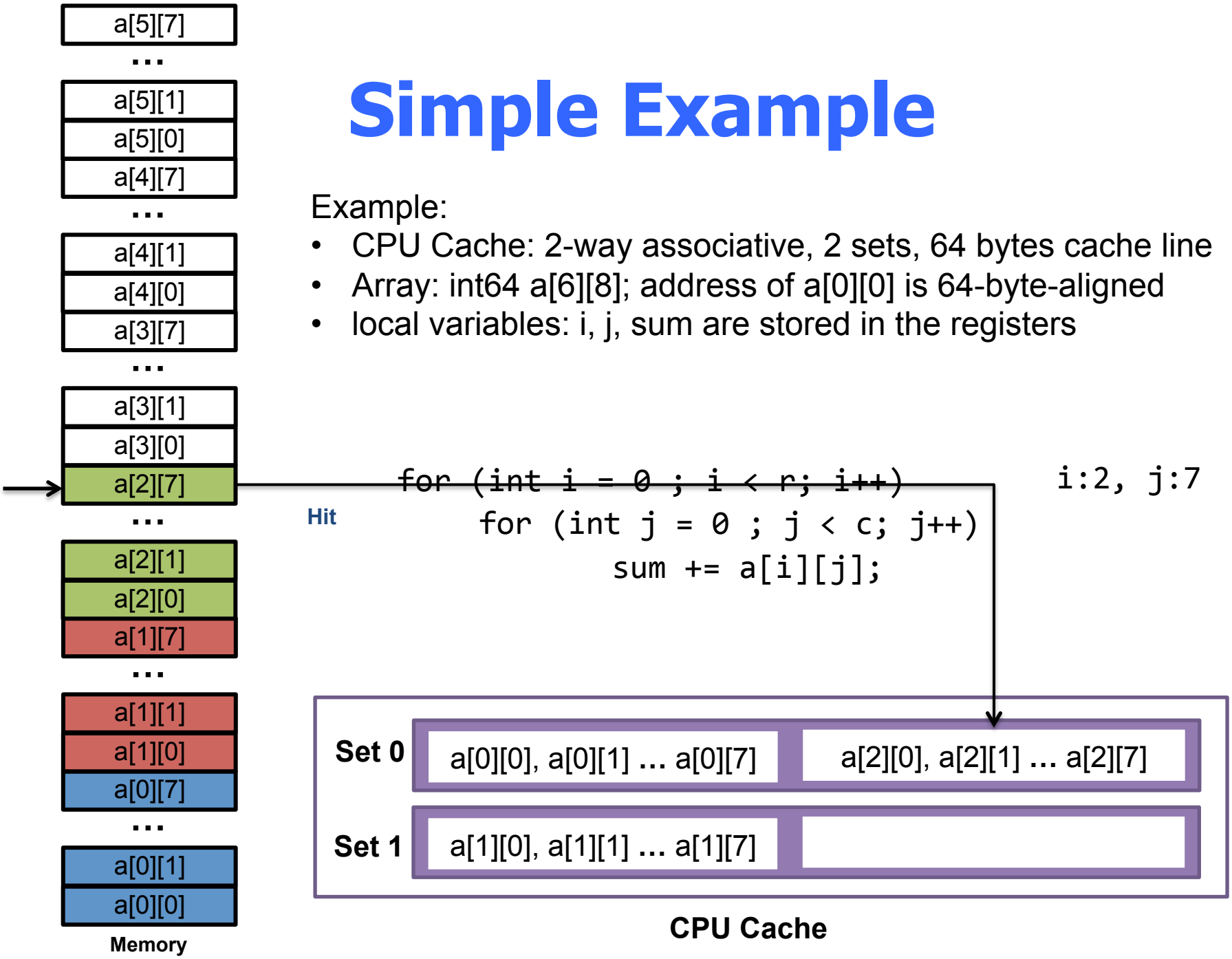
- CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
- Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
- local variables: i, j, sum are stored in the registers



Simple Example

Example:

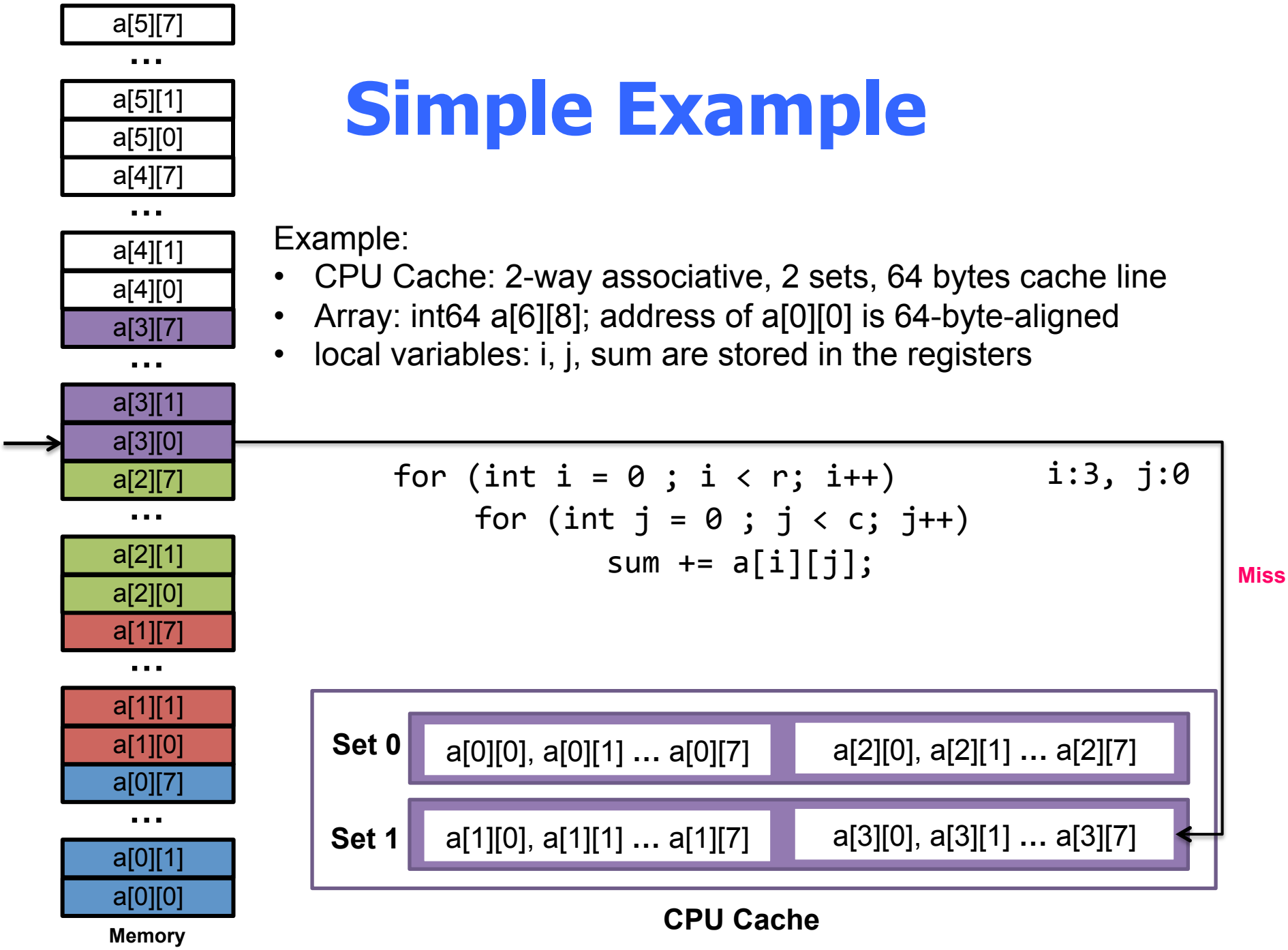
- CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
- Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
- local variables: i, j, sum are stored in the registers



Simple Example

Example:

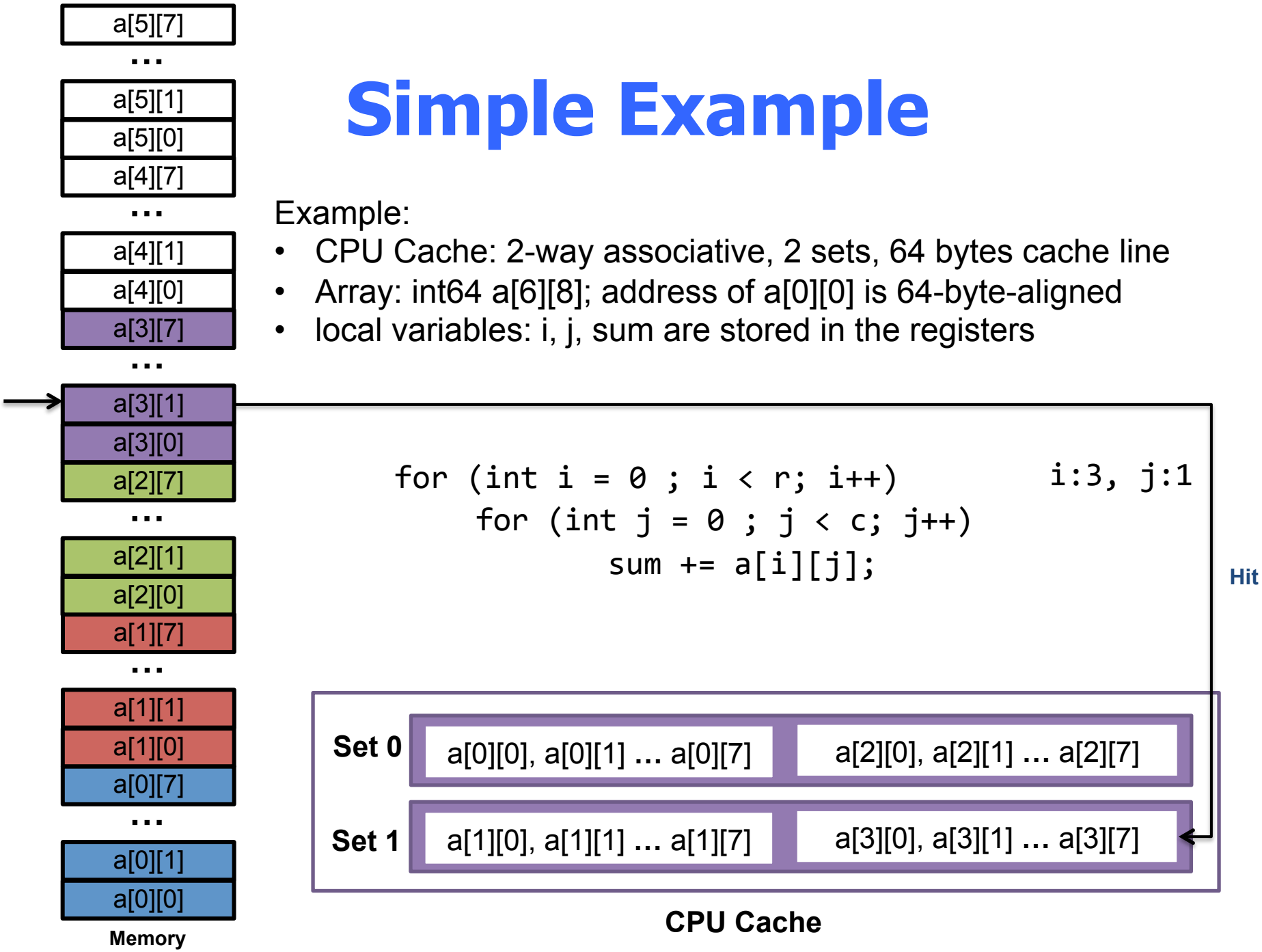
- CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
- Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
- local variables: i, j, sum are stored in the registers



Simple Example

Example:

- CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
- Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
- local variables: i, j, sum are stored in the registers



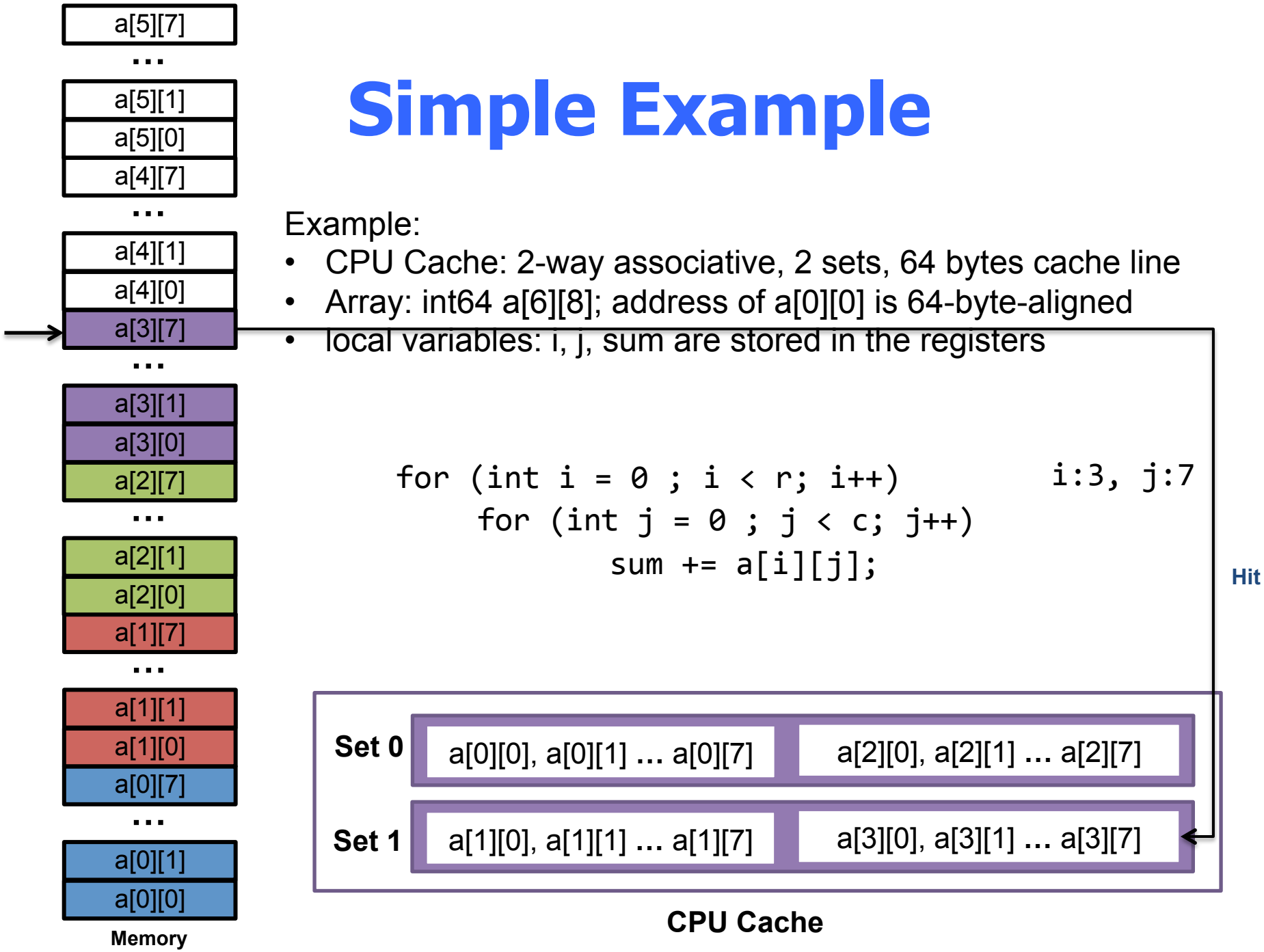
Simple Example

Example:

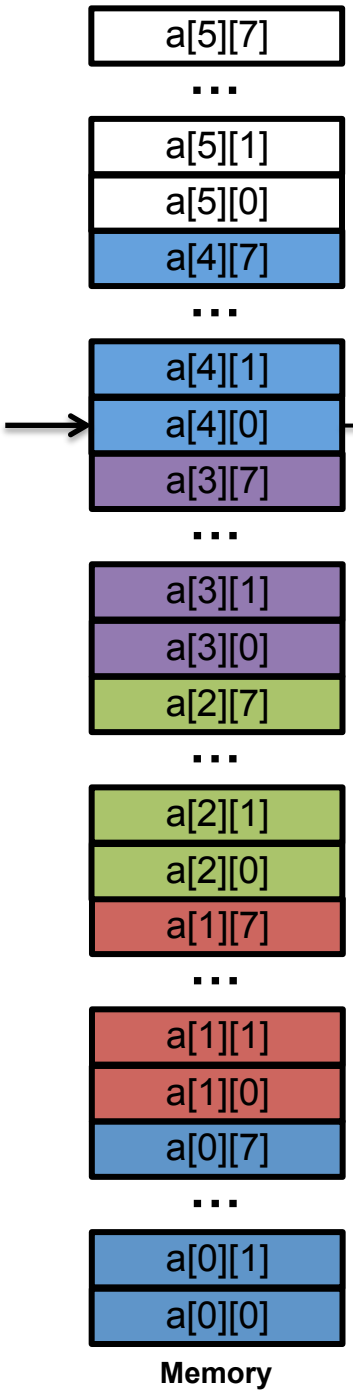
- CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
- Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
- local variables: i, j, sum are stored in the registers

```
for (int i = 0 ; i < r; i++)
    for (int j = 0 ; j < c; j++)
        sum += a[i][j];
```

i:3, j:7



Simple Example

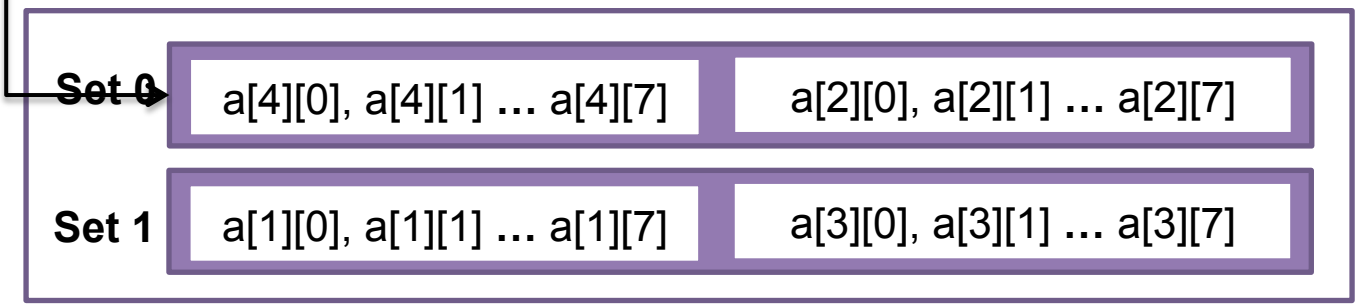


Example:

- CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
- Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
- local variables: i, j, sum are stored in the registers

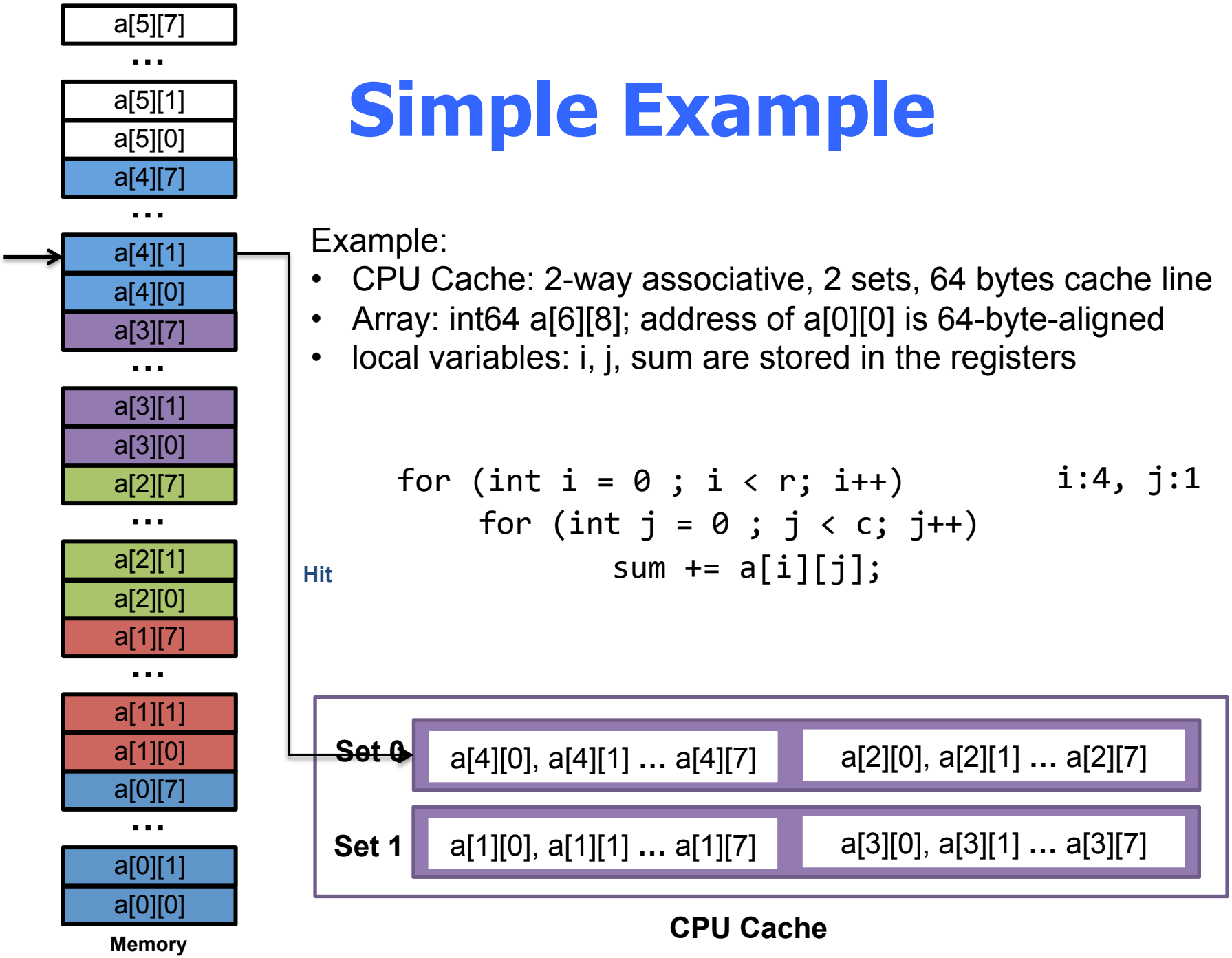
```
for (int i = 0 ; i < r; i++)          i:4, j:0
    for (int j = 0 ; j < c; j++)
        sum += a[i][j];
```

Miss



CPU Cache

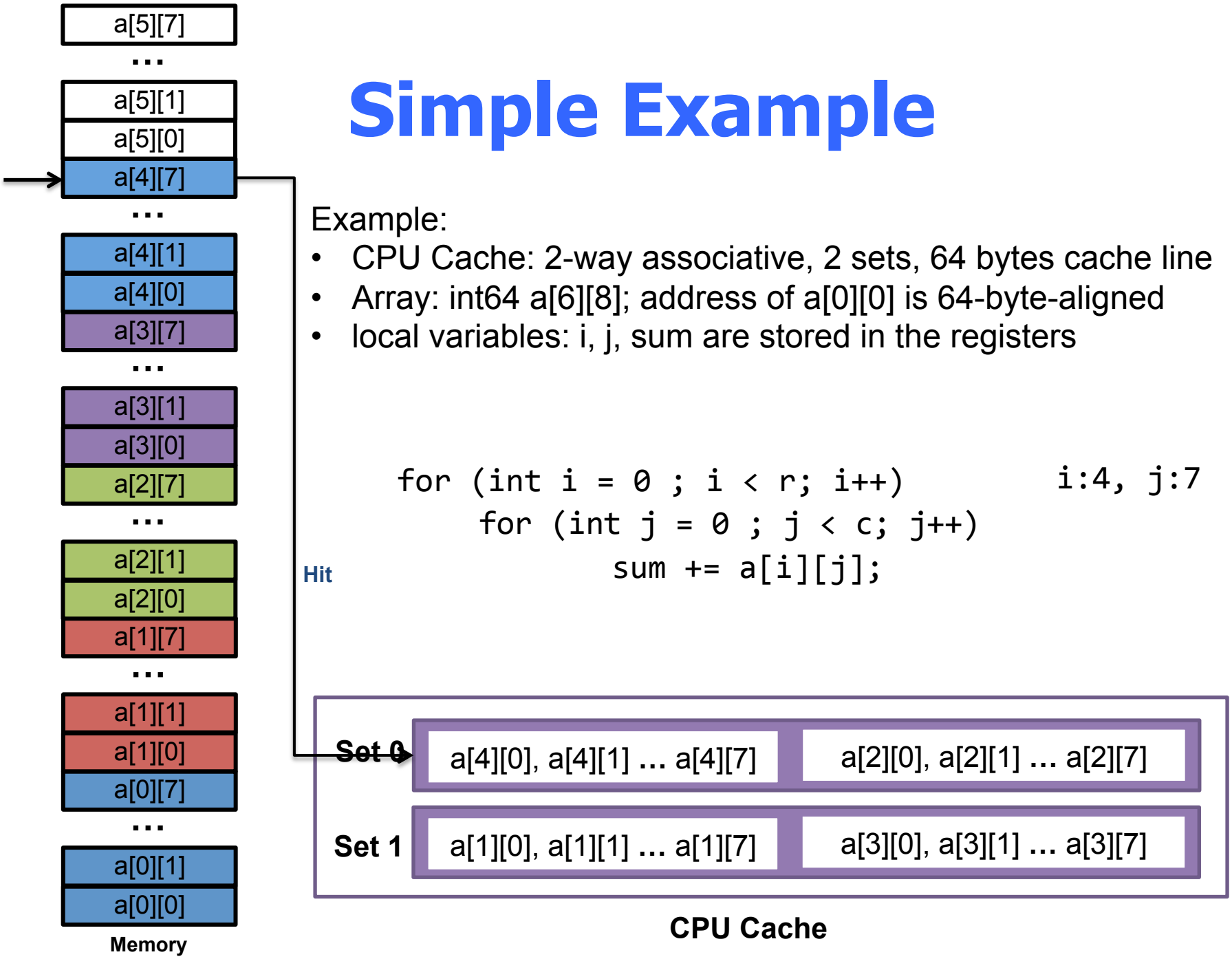
Simple Example

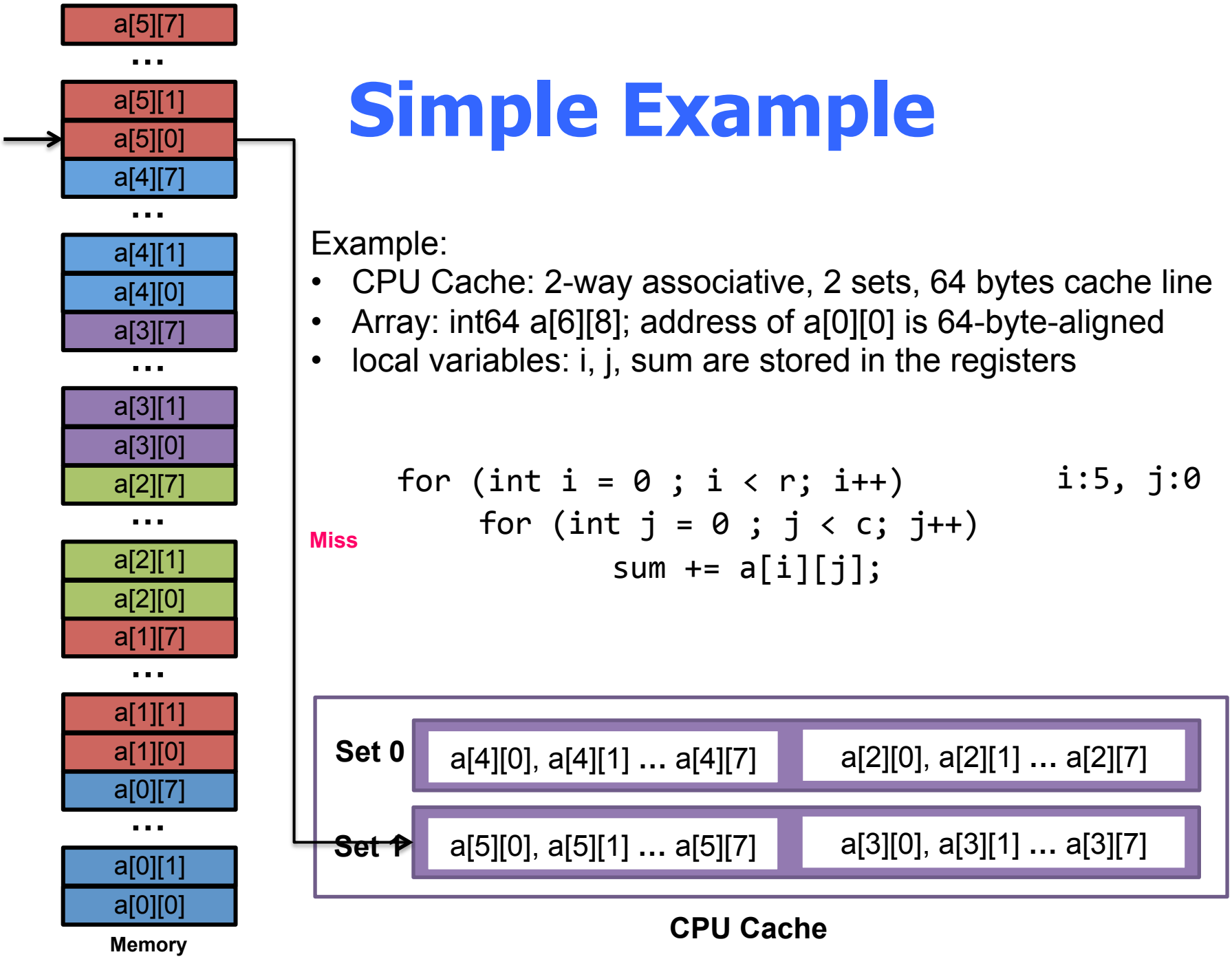


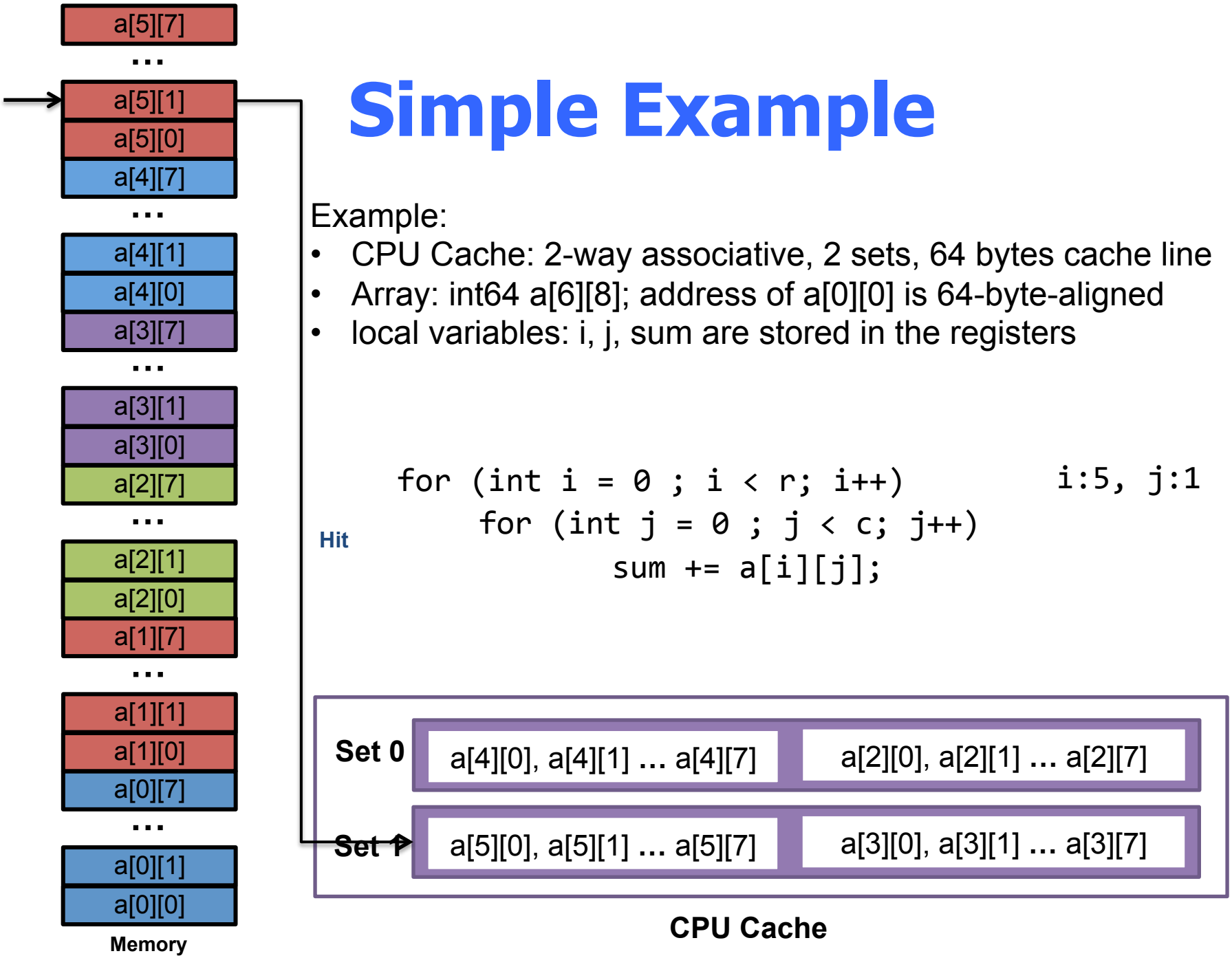
Example:

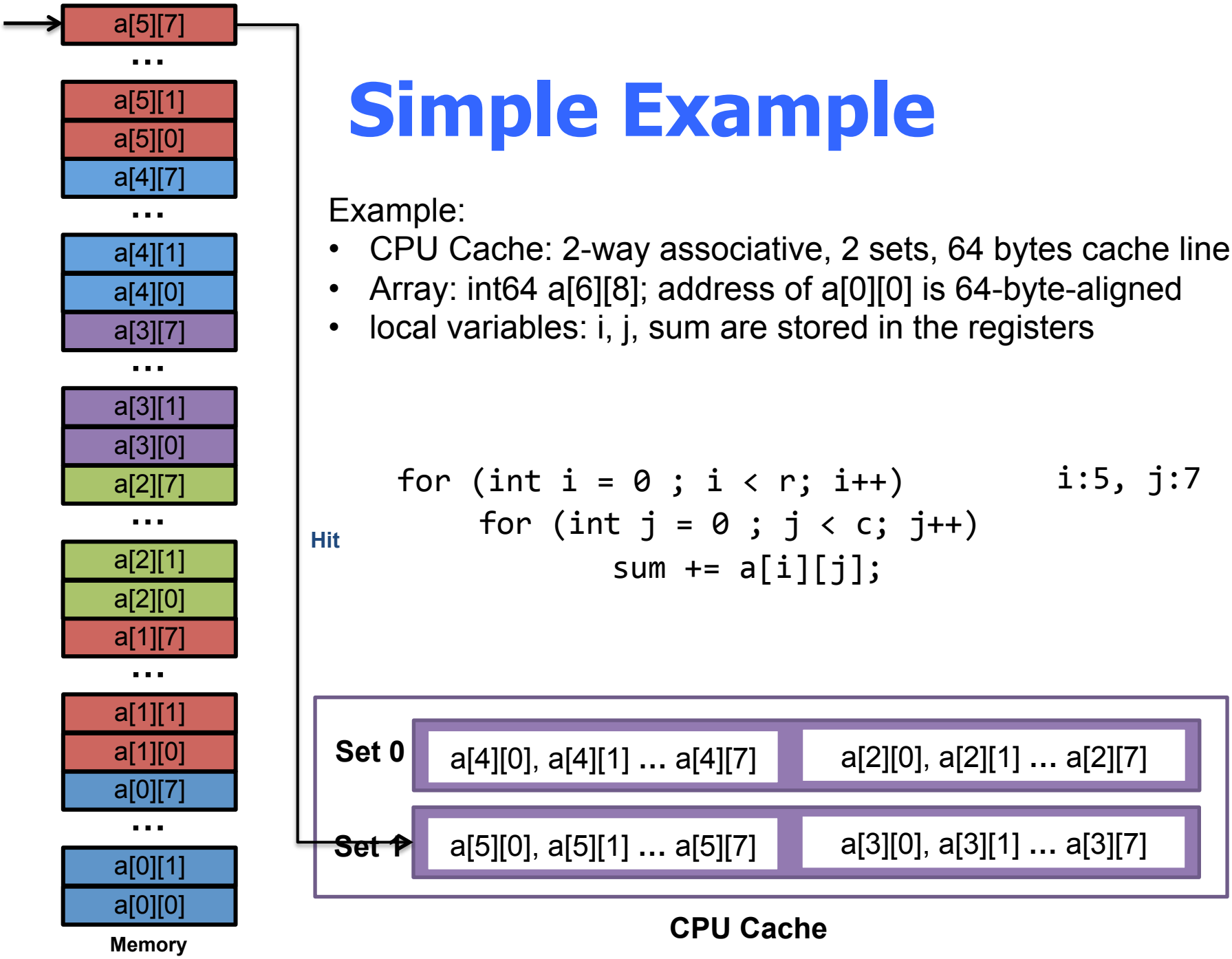
- CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
- Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
- local variables: i, j, sum are stored in the registers

```
for (int i = 0 ; i < r; i++)          i:4, j:1
    for (int j = 0 ; j < c; j++)
        sum += a[i][j];
```


a[5][7]
...
a[5][1]
a[5][0]
a[4][7]
...
a[4][1]
a[4][0]
a[3][7]
...
a[3][1]
a[3][0]
a[2][7]
...
a[2][1]
a[2][0]
a[1][7]
...
a[1][1]
a[1][0]
a[0][7]
...
a[0][1]
a[0][0]
Memory
Simple Example
Example:
• CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
• Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
• local variables: i, j, sum are stored in the registers
for (int i = 0 ; i < r; i++)
for (int j = 0 ; j < c; j++)
sum += a[i][j];
i:4, j:7
Hit
Set 0
a[4][0], a[4][1] ... a[4][7]
a[2][0], a[2][1] ... a[2][7]
Set 1
a[1][0], a[1][1] ... a[1][7]
a[3][0], a[3][1] ... a[3][7]
CPU Cache


a[5][7]
...
a[5][1]
a[5][0]
a[4][7]
...
a[4][1]
a[4][0]
a[3][7]
...
a[3][1]
a[3][0]
a[2][7]
...
a[2][1]
a[2][0]
a[1][7]
...
a[1][1]
a[1][0]
a[0][7]
...
a[0][1]
a[0][0]
Memory
Simple Example
Example:
• CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
• Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
• local variables: i, j, sum are stored in the registers
for (int i = 0 ; i < r; i++)
for (int j = 0 ; j < c; j++)
sum += a[i][j];
i:5, j:0
Miss
Set 0
a[4][0], a[4][1] ... a[4][7]
a[2][0], a[2][1] ... a[2][7]
Set 1
a[5][0], a[5][1] ... a[5][7]
a[3][0], a[3][1] ... a[3][7]
CPU Cache


a[5][7]
...
a[5][1]
a[5][0]
a[4][7]
...
a[4][1]
a[4][0]
a[3][7]
...
a[3][1]
a[3][0]
a[2][7]
...
a[2][1]
a[2][0]
a[1][7]
...
a[1][1]
a[1][0]
a[0][7]
...
a[0][1]
a[0][0]
Memory
Simple Example
Example:
• CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
• Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
• local variables: i, j, sum are stored in the registers
for (int i = 0 ; i < r; i++)
for (int j = 0 ; j < c; j++)
sum += a[i][j];
i:5, j:1
Hit
Set 0
a[4][0], a[4][1] ... a[4][7]
a[2][0], a[2][1] ... a[2][7]
Set 1
a[5][0], a[5][1] ... a[5][7]
a[3][0], a[3][1] ... a[3][7]
CPU Cache

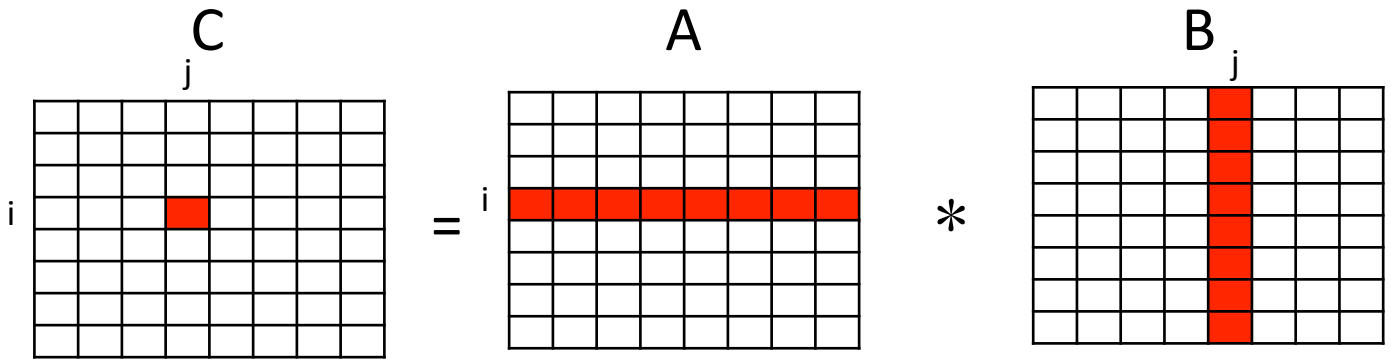

a[5][7]
...
a[5][1]
a[5][0]
a[4][7]
...
a[4][1]
a[4][0]
a[3][7]
...
a[3][1]
a[3][0]
a[2][7]
...
a[2][1]
a[2][0]
a[1][7]
...
a[1][1]
a[1][0]
a[0][7]
...
a[0][1]
a[0][0]
Memory
Hit
Simple Example
Example:
• CPU Cache: 2-way associative, 2 sets, 64 bytes cache line
• Array: int64 a[6][8]; address of a[0][0] is 64-byte-aligned
• local variables: i, j, sum are stored in the registers
for (int i = 0 ; i < r; i++)
for (int j = 0 ; j < c; j++)
sum += a[i][j];
i:5, j:7
Set 0
a[4][0], a[4][1] ... a[4][7]
a[2][0], a[2][1] ... a[2][7]
Set 1
a[5][0], a[5][1] ... a[5][7]
a[3][0], a[3][1] ... a[3][7]
CPU Cache

Matrix Multiplication (ijk)

Cache: 2-way, 2 sets, 64B cacheline
Matrix A, B, C: double[8][8]
1st elements of A,B,C, are 64B-aligned

C = A * B → C[i,j] = A[i:] • B[:j]

```
for (int i=0; i < N; i++)  {
  for (int j=0; j < N; j++) {
   for (int k=0; k < N; k++)
      C[i][j] += A[i][k] * B[k][j];
  }
}
```



Matrix Multiplication

Cache: 2-way, 2 sets, 64B cacheline
Matrix A, B, C: double[8][8]
1st elements of A,B,C, are 64B-aligned

```
for (int i=0; i < N; i++)  {
  for (int j=0; j < N; j++) {
   for (int k=0; k < N; k++)
      C[i][j] += A[i][k] * B[k][j];
  }
}
```

MM—ijk

```
for (int i=0; i < N; i++)  {
  for (int k=0; k < N; k++) {
     for (int j=0; j < N; j++)
      C[i][j] += A[i][k] * B[k][j];
  }
}
```

MM—ikj

```
for (int k=0; k < N; k++)  {
  for (int j=0; j < N; j++) {
     for (int i=0; i < N; i++)
      C[i][j] += A[i][k] * B[k][j];
  }
}
```

MM—kji

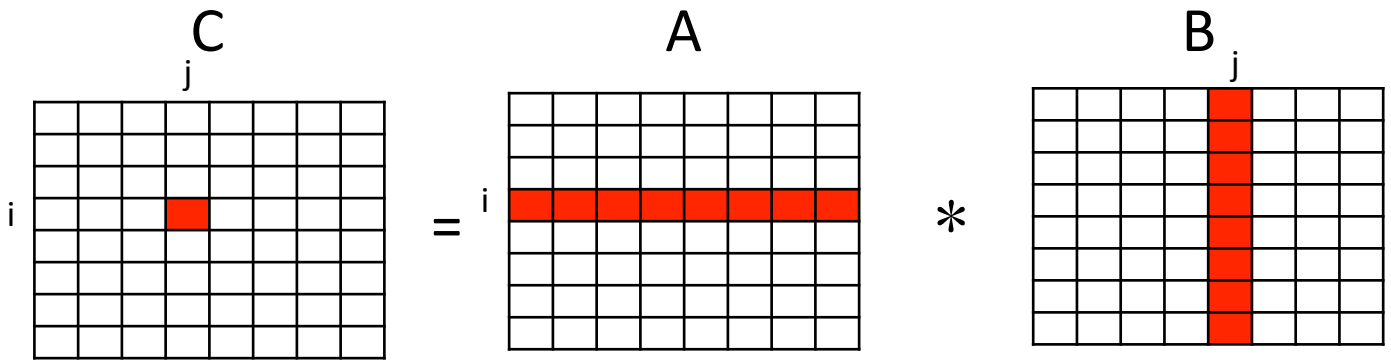
Which one is cache friendly?
Which one is worst?

Matrix Multiplication (ijk)

Cache: 2-way, 2 sets, 64B cacheline
Matrix A, B, C: double[8][8]
1st elements of A,B,C, are 64B-aligned

C = A * B → C[i,j] = A[i:] • B[:j]

```
for (int i=0; i < N; i++)  {
  for (int j=0; j < N; j++) {
   for (int k=0; k < N; k++)
      C[i][j] += A[i][k] * B[k][j];
  }
}
```

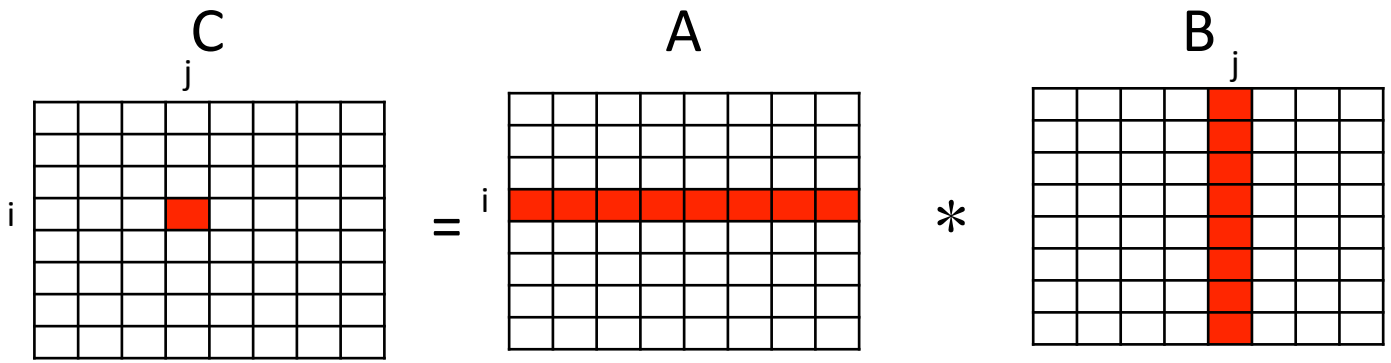


Matrix Multiplication (ijk)

Cache: 2-way, 2 sets, 64B cacheline
Matrix A, B, C: double[8][8]
1st elements of A,B,C, are 64B-aligned

C = A * B → C[i,j] = A[i:] • B[:j]

```
for (int i=0; i < N; i++)  {
  for (int j=0; j < N; j++) {
    for (int k=0; k < N; k++)
      C[i][j] += A[i][k] * B[k][j];
  }
}
```

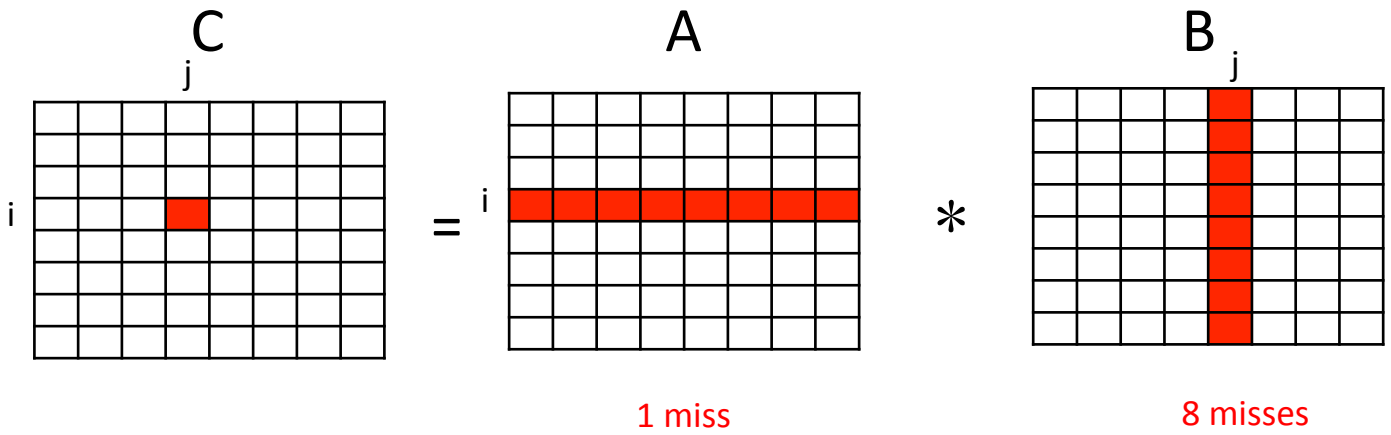


Matrix Multiplication (ijk)

Cache: 2-way, 2 sets, 64B cacheline
Matrix A, B, C: double[8][8]
1st elements of A,B,C, are 64B-aligned

C = A * B → C[i,j] = A[i:] • B[:j]

```
for (int i=0; i < N; i++)  {
  for (int j=0; j < N; j++) {
    for (int k=0; k < N; k++)
      C[i][j] += A[i][k] * B[k][j];
  }
}
```

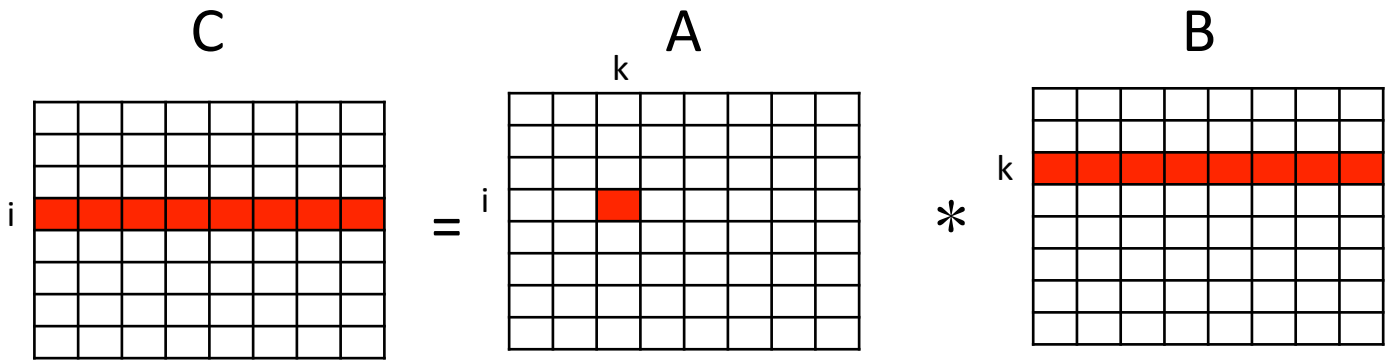


Matrix Multiplication (ikj)

Cache: 2-way, 2 sets, 64B cacheline
Matrix A, B, C: double[8][8]
1st elements of A,B,C, are 64B-aligned

```
C = A * B → C[i:] +=  A[i,k] * B[k:]
                      (k in [0, N))
```

```
for (int i=0; i < N; i++)  {
  for (int k=0; k < N; k++) {
    for (int j=0; j < N; j++)
      C[i][j] += A[i][k] * B[k][j];
  }
}
```

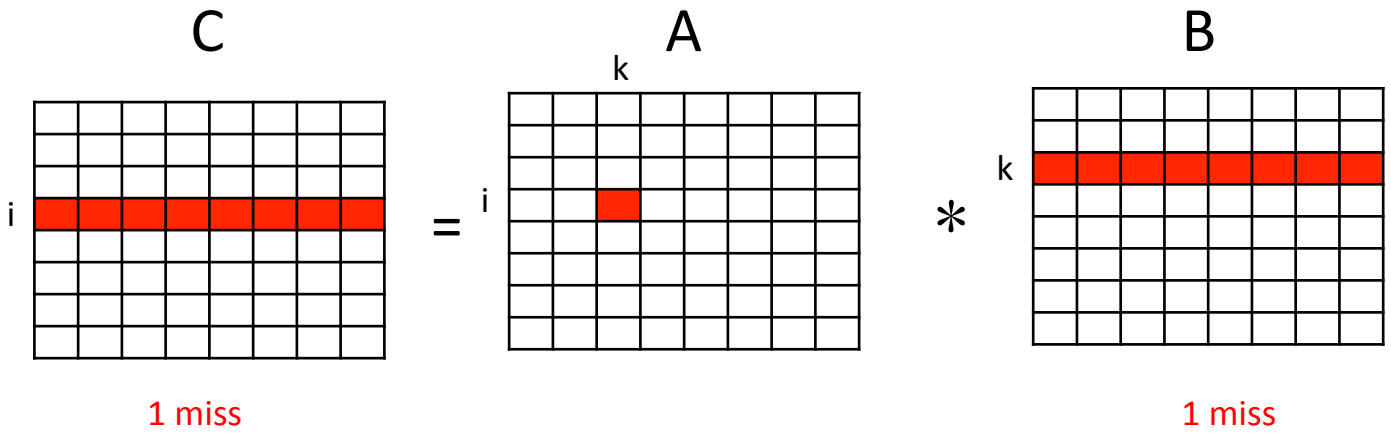


Matrix Multiplication (ikj)

Cache: 2-way, 2 sets, 64B cacheline
Matrix A, B, C: double[8][8]
1st elements of A,B,C, are 64B-aligned

```
C = A * B → C[i:] +=  A[i,k] * B[k:]
                      (k in [0, N))
```

```
for (int i=0; i < N; i++)  {
  for (int k=0; k < N; k++) {
    for (int j=0; j < N; j++)
      C[i][j] += A[i][k] * B[k][j];
  }
}
```

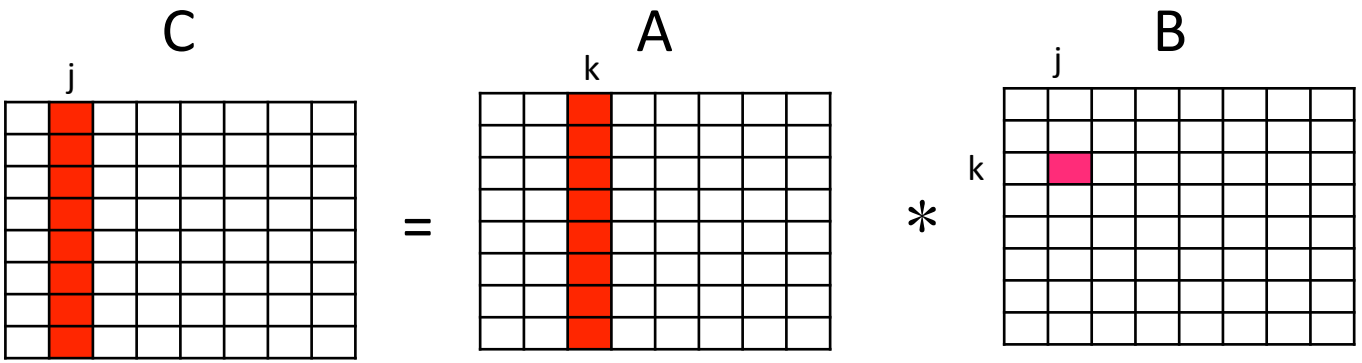


Matrix Multiplication (kji)

Cache: 2-way, 2 sets, 64B cacheline
Matrix A, B, C: double[8][8]
1st elements of A,B,C, are 64B-aligned

```
C = A * B → C[i:] += A[i,k] * B[k:]
                     (k in [0, N))
```

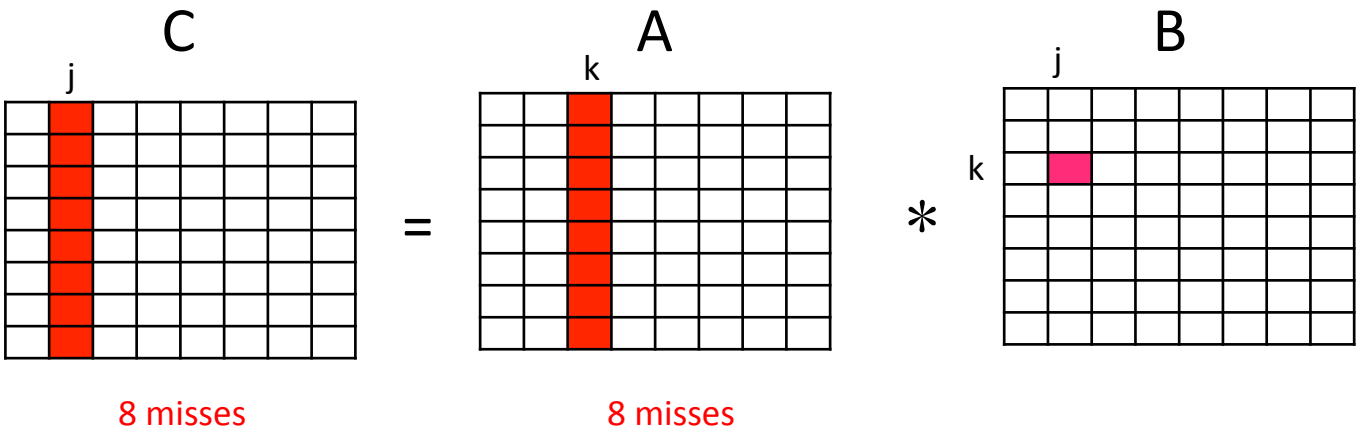
```
for (int k=0; k < N; k++)  {
  for (int j=0; j < N; j++) {
    for (int i=0; i < N; i++)
      C[i][j] += A[i][k] * B[k][j];
  }
}
```



Matrix Multiplication (kji)

Cache: 2-way, 2 sets, 64B cacheline
Matrix A, B, C: double[8][8]
1st elements of A,B,C, are 64B-aligned

```
C = A * B → C[:j] +=  A[:k] * B[k:j]
                      k in [k, N)
for (int k=0; k < N; k++)  {
  for (int j=0; j < N; j++) {
    for (int i=0; i < N; i++)
      C[i][j] += A[i][k] * B[k][j];
  }
}
```



Memory Layout

```
typedef struct {

  char g;
  int64_t id;
  char g1;
  int64_t id1;
  char g2;
  int64_t id2;

} info;
```

```
typedef struct {

  int64_t id;
  int64_t id1;
  int64_t id2;
  char g;
  char g1;
  char g2;

} info;
```

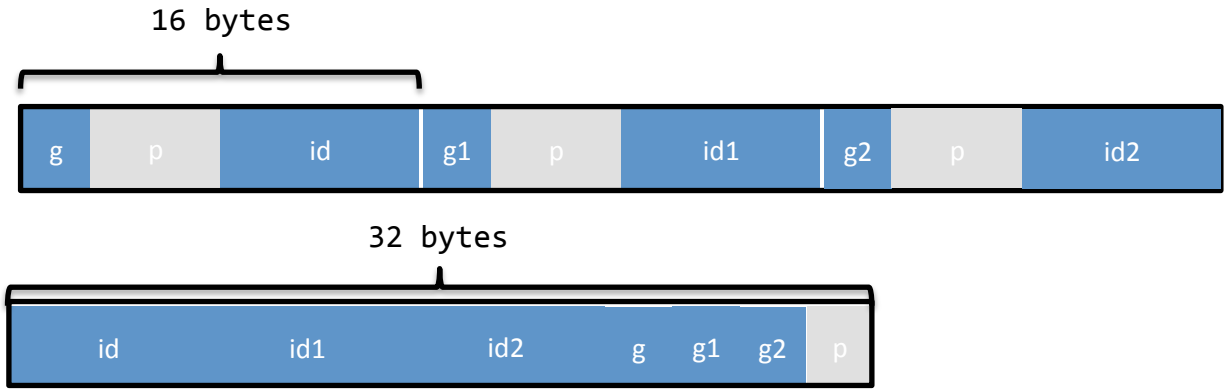
```
for(int i = 0 ; i < N; i++) {
        a[i].id = i;
        a[i].id1 = i+1;
        a[i].id2 = i+2;
        a[i].g2 = 'y';
        a[i].g1 = 'e';
        a[i].g = 's';
 }
```

Memory Layout

```
typedef struct {

  char g;
  int64_t id;
  char g1;
  int64_t id1;
  char g2;
  int64_t id2;

} info;
```

```
typedef struct {

  int64_t id;
  int64_t id1;
  int64_t id2;
  char g;
  char g1;
  char g2;

} info;
```

```
for(int i = 0 ; i < N; i++) {
        a[i].id = i;
        a[i].id1 = i+1;
        a[i].id2 = i+2;
        a[i].g2 = 'y';
        a[i].g1 = 'e';
        a[i].g = 's';
 }
```



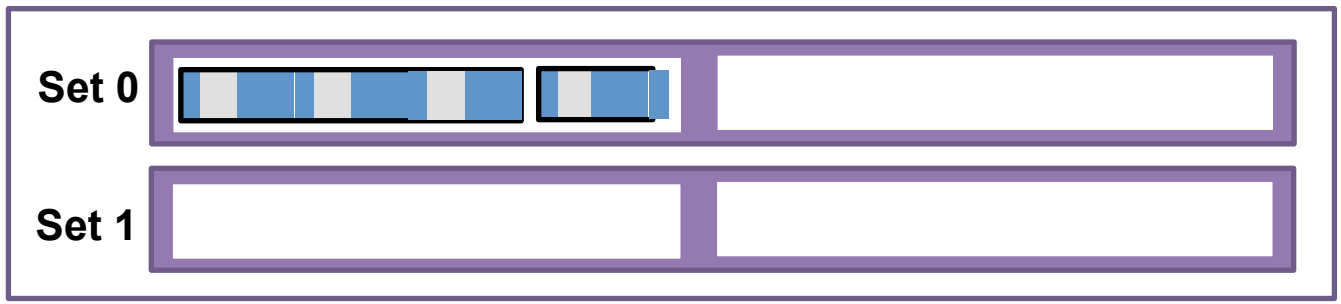
Memory Layout

```
typedef struct {

  char g;
  int64_t id;
  char g1;
  int64_t id1;
  char g2;
  int64_t id2;

} info;
```

```
for(int i = 0 ; i < N; i++) {
        a[i].id = i;
        a[i].id1 = i+1;
        a[i].id2 = i+2;
        a[i].g2 = 'y';
        a[i].g1 = 'e';
        a[i].g = 's';
 }
```

CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – info a[2]
The address of a[0] is cache line alignment

access a[0]



CPU Cache

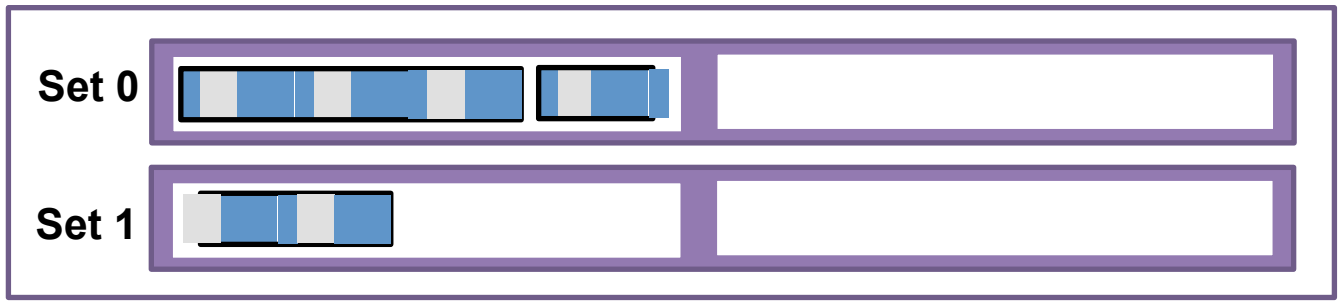
Memory Layout

```
typedef struct {

  char g;
  int64_t id;
  char g1;
  int64_t id1;
  char g2;
  int64_t id2;

} info;
```

```
for(int i = 0 ; i < N; i++) {
        a[i].id = i;
        a[i].id1 = i+1;
        a[i].id2 = i+2;
        a[i].g2 = 'y';
        a[i].g1 = 'e';
        a[i].g = 's';
 }
```

CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – info a[2]
The address of a[0] is cache line alignment

access a[1]



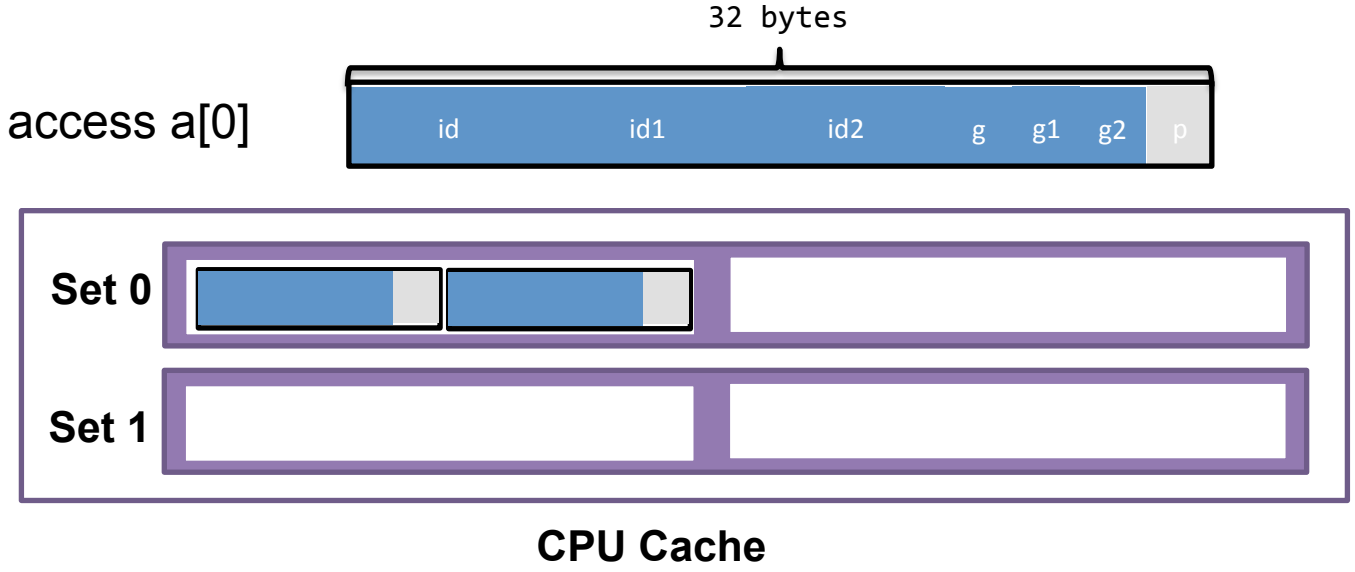
CPU Cache

Memory Layout

```
typedef struct {

  int64_t id;
  int64_t id1;
  int64_t id2;
  char g;
  char g1;
  char g2;

} info;
```

```
for(int i = 0 ; i < N; i++) {
        a[i].id = i;
        a[i].id1 = i+1;
        a[i].id2 = i+2;
        a[i].g2 = 'y';
        a[i].g1 = 'e';
        a[i].g = 's';
 }
```

CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – info a[2]
The address of a[0] is cache line alignment



Memory Layout

```
typedef struct {

  int64_t id;
  int64_t id1;
  int64_t id2;
  char g;
  char g1;
  char g2;

} info;
```

```
for(int i = 0 ; i < N; i++) {
        a[i].id = i;
        a[i].id1 = i+1;
        a[i].id2 = i+2;
        a[i].g2 = 'y';
        a[i].g1 = 'e';
        a[i].g = 's';
 }
```

CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – info a[2]
The address of a[0] is cache line alignment