NYU

Lecture 8

Multidimensional Arrays

Motivations

- Thus far, you have used one-dimensional arrays to model linear collections of elements.

- You can use a two-dimensional array to represent a matrix or a table.

Motivations

- Example, table that describes the distances between the cities

Distance Table (in miles)

	Chicago	Boston	New York	Atlanta	Miami	Dallas	Houston
Chicago	0	983	787	714	1375	967	1087
Boston	983	0	214	1102	1763	1723	1842
New York	787	214	0	888	1549	1548	1627
Atlanta	714	1102	888	0	661	781	810
Miami	1375	1763	1549	661	0	1426	1187
Dallas	967	1723	1548	781	1426	0	239
Houston	1087	1842	1627	810	1187	239	0

Motivations

```
double[][] distances = {
   {0, 983, 787, 714, 1375, 967, 1087},
   {983, 0, 214, 1102, 1763, 1723, 1842},
   {787, 214, 0, 888, 1549, 1548, 1627},
   {1375, 1763, 1549, 661, 0, 1426, 1187},
   {967, 1723, 1548, 781, 1426, 0, 239},
   {1087, 1842, 1627, 810, 1187, 239, 0}
};
```

Declare/Create 2D Arrays

- Declare array ref var
- `dataType[][] refVar;`

- Create array and assign its reference to variable
- `refVar = new dataType[10][10];`

- Combine declaration and creation in one statement
- `dataType[][] refVar = new dataType[10][10];`

- Alternative syntax
- `dataType refVar[][] = new dataType[10][10];`

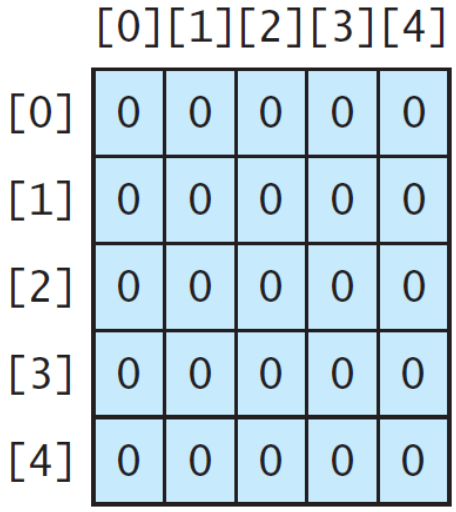
Example

```
int[][] matrix = new int[10][10];
```

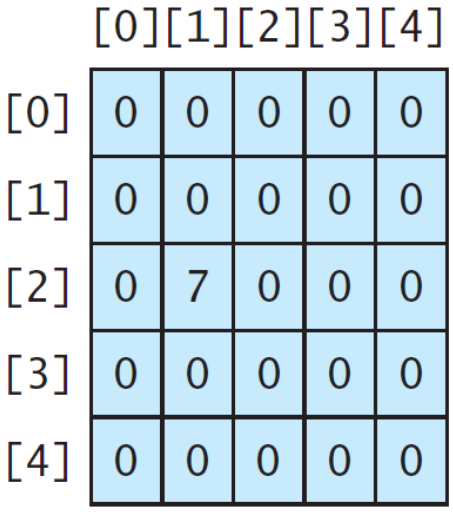
or

```
int matrix[][] = new int[10][10];

matrix[0][0] = 3;

for (int i = 0; i < matrix.length; i++)
  for (int j = 0; j < matrix[i].length; j++)
    matrix[i][j] = (int)(Math.random() * 1000);

double[][] x;
```

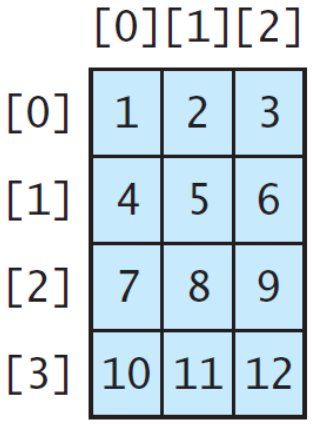
Two-dimensional Array Illustration




```
matrix = new int[5][5];
```




```
matrix[2][1] = 7;
```




```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

`matrix.length?  5`

`matrix[0].length? 5`

`array.length?  4`

`array[0].length? 3`

Shorthand Notations

- You can also use an array initializer to declare, create and initialize a two-dimensional array.

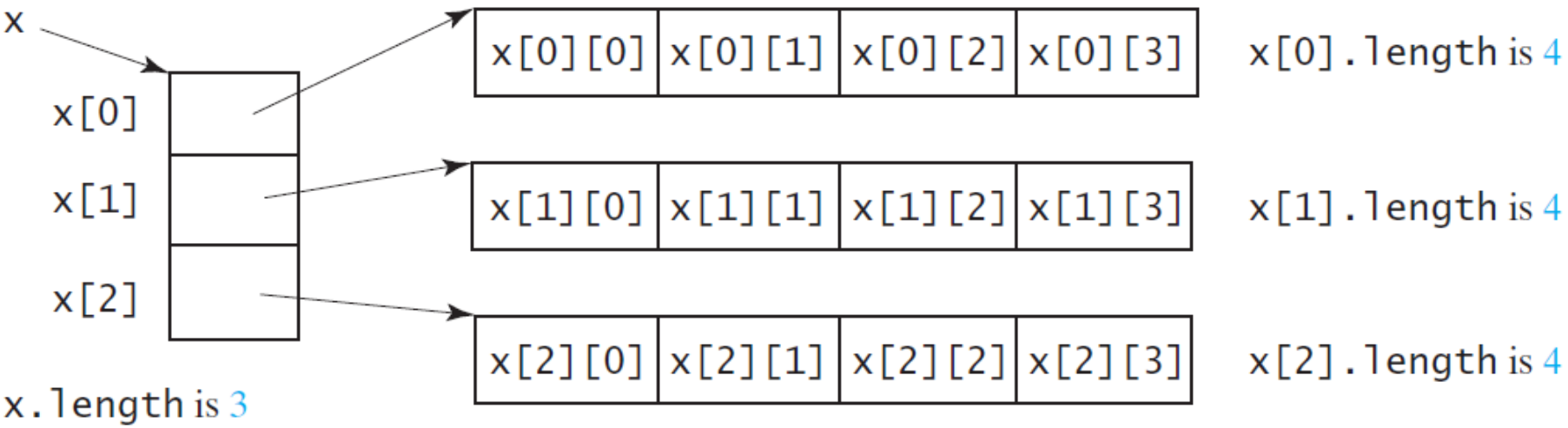
- Example:

```
int[][] array
   = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

```
int[][] array = new int[4][3];

array[0][0] = 1; array[0][1] = 2; array[0][2] = 3;
array[1][0] = 4; array[1][1] = 5; array[1][2] = 6;
array[2][0] = 7; array[2][1] = 8; array[2][2] = 9;
array[3][0] = 10; array[3][1] = 11; array[3][2] = 12;
```

Lengths of Two-dimensional Arrays

- `int[][] x = new int[3][4];`



Lengths of Two-dimensional Arrays

```
int[][] array = {
   {1, 2, 3},
   {4, 5, 6},
   {7, 8, 9},
   {10, 11, 12}
};

array.length

array[0].length
array[1].length
array[2].length
array[3].length
array[4].length ArrayIndexOutOfBoundsException
```

Ragged Arrays

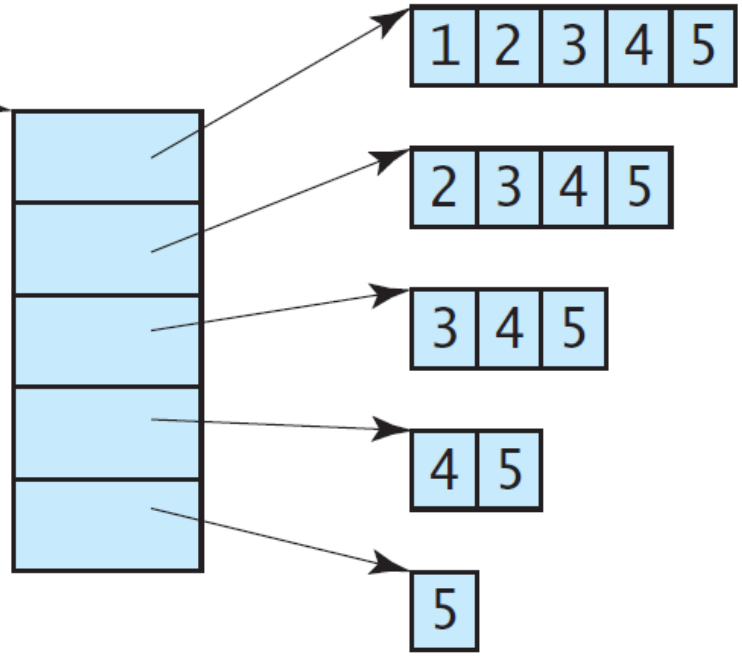
- Each row in a two-dimensional array is itself an array.
- The rows can have different lengths. Such an array is known as a **ragged array**.
- Example:

```
int[][] matrix = {
{1, 2, 3, 4, 5},
{2, 3, 4, 5},
{3, 4, 5},
{4, 5},
{5}
};
```

```
matrix.length is 5
matrix[0].length is 5
matrix[1].length is 4
matrix[2].length is 3
matrix[3].length is 2
matrix[4].length is 1
```

Ragged Arrays

```
int[][] triangleArray = {
   {1, 2, 3, 4, 5},
   {2, 3, 4, 5},
   {3, 4, 5},
   {4, 5},
   {5}
};
```



Processing Two-Dimensional Arrays

- Initializing arrays with input values
- Printing arrays
- Summing all elements
- Summing all elements by column
- Which row has the largest sum
- Finding the smallest index of the largest element
- Random shuffling

Initializing arrays with input values

```
java.util.Scanner input = new Scanner(System.in);
System.out.println("Enter " + matrix.length + " rows and
" +
  matrix[0].length + " columns: ");
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length;
column++) {
    matrix[row][column] = input.nextInt();
  }
}
```

Initializing arrays with random values

```
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length;
column++) {
    matrix[row][column] = (int)(Math.random() * 100);
  }
}
```

Printing arrays

```
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length;
column++) {
    System.out.print(matrix[row][column] + " ");
  }

  System.out.println();
}
```

Summing all elements

```
int total = 0;
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length;
column++) {
    total += matrix[row][column];
  }
}
```

Summing elements by column

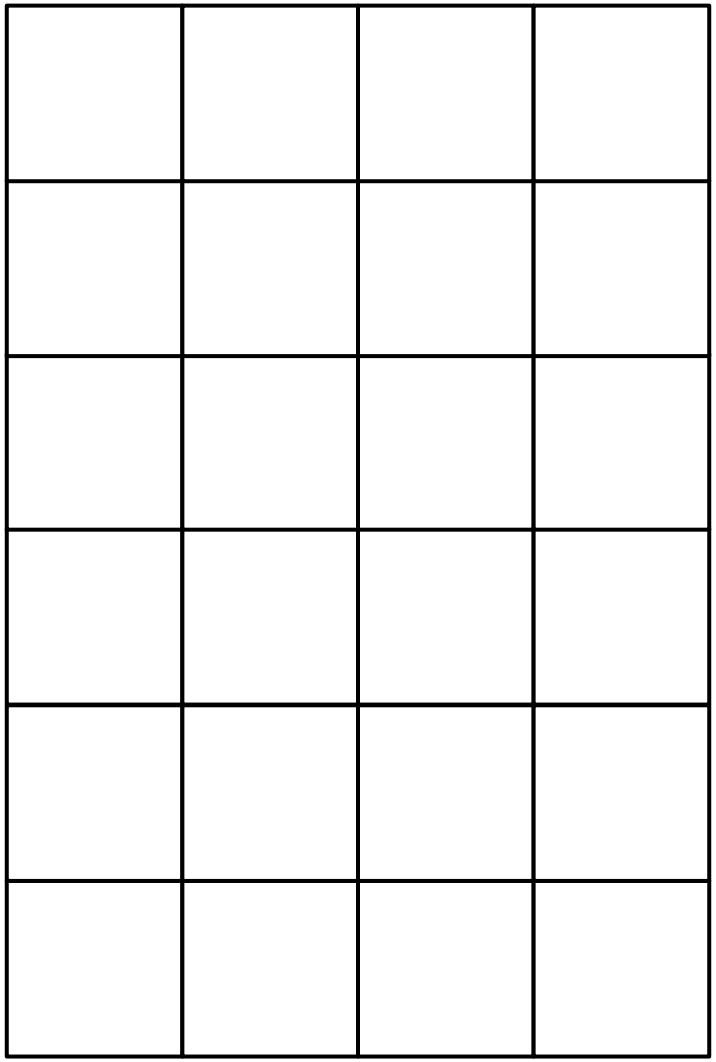
```
for (int column = 0; column < matrix[0].length; column++)
{
  int total = 0;
  for (int row = 0; row < matrix.length; row++)
    total += matrix[row][column];
  System.out.println("Sum for column " + column + " is "
    + total);
}
```

Random shuffling

```
for (int i = 0; i < matrix.length; i++) {
  for (int j = 0; j < matrix[i].length; j++) {
    int i1 = (int)(Math.random() * matrix.length);
    int j1 = (int)(Math.random() * matrix[i].length);
    // Swap matrix[i][j] with matrix[i1][j1]
    int temp = matrix[i][j];
    matrix[i][j] = matrix[i1][j1];
    matrix[i1][j1] = temp;
  }
}
```

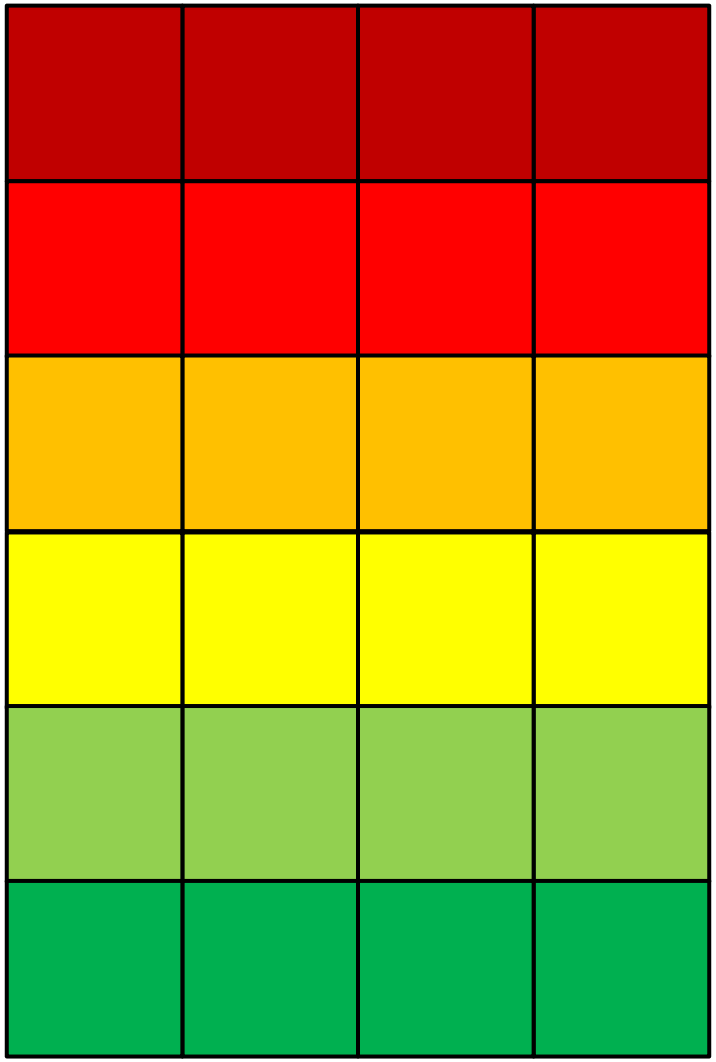
Matrices and memory

```
double[][] image = new double[6][4];
```



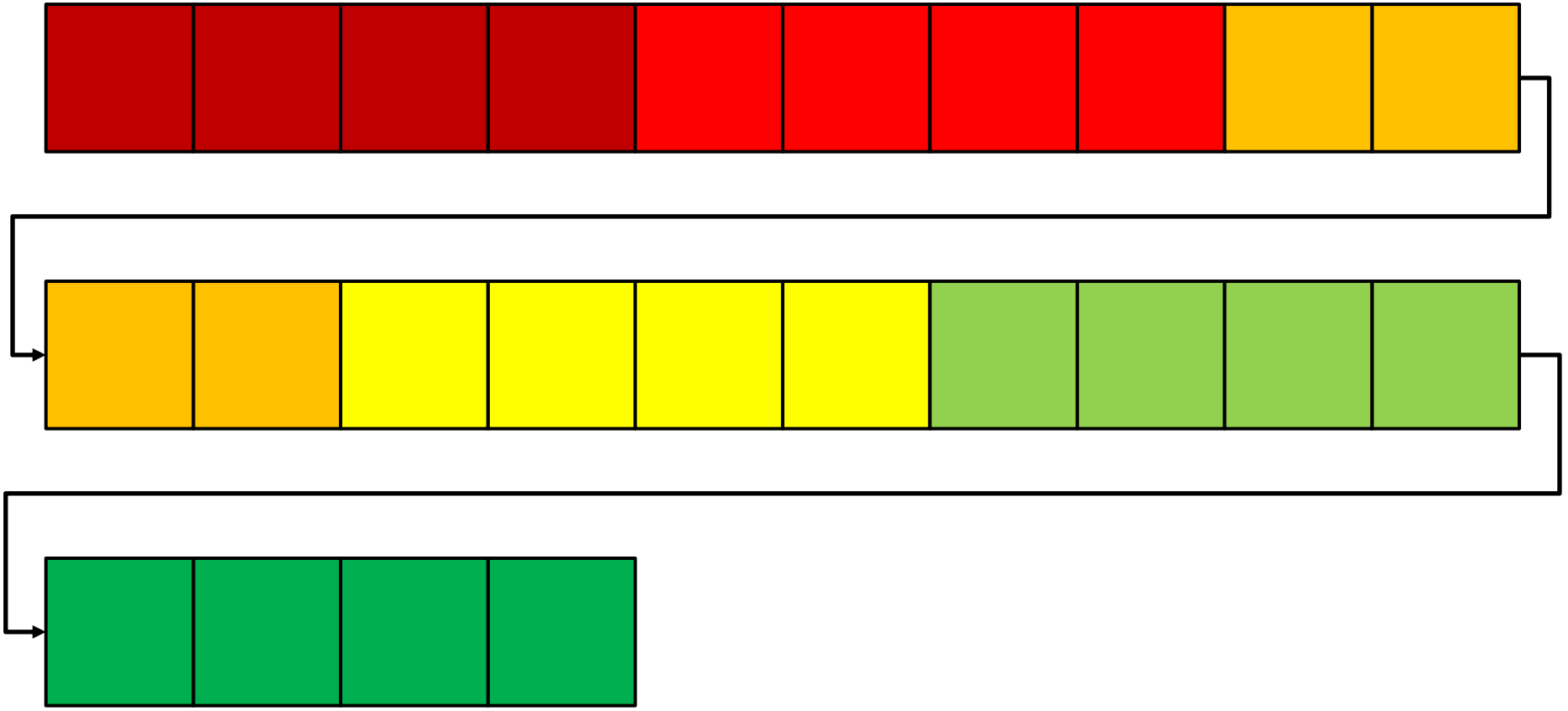
Matrices and memory

```
double[][] image = new double[6][4];
```

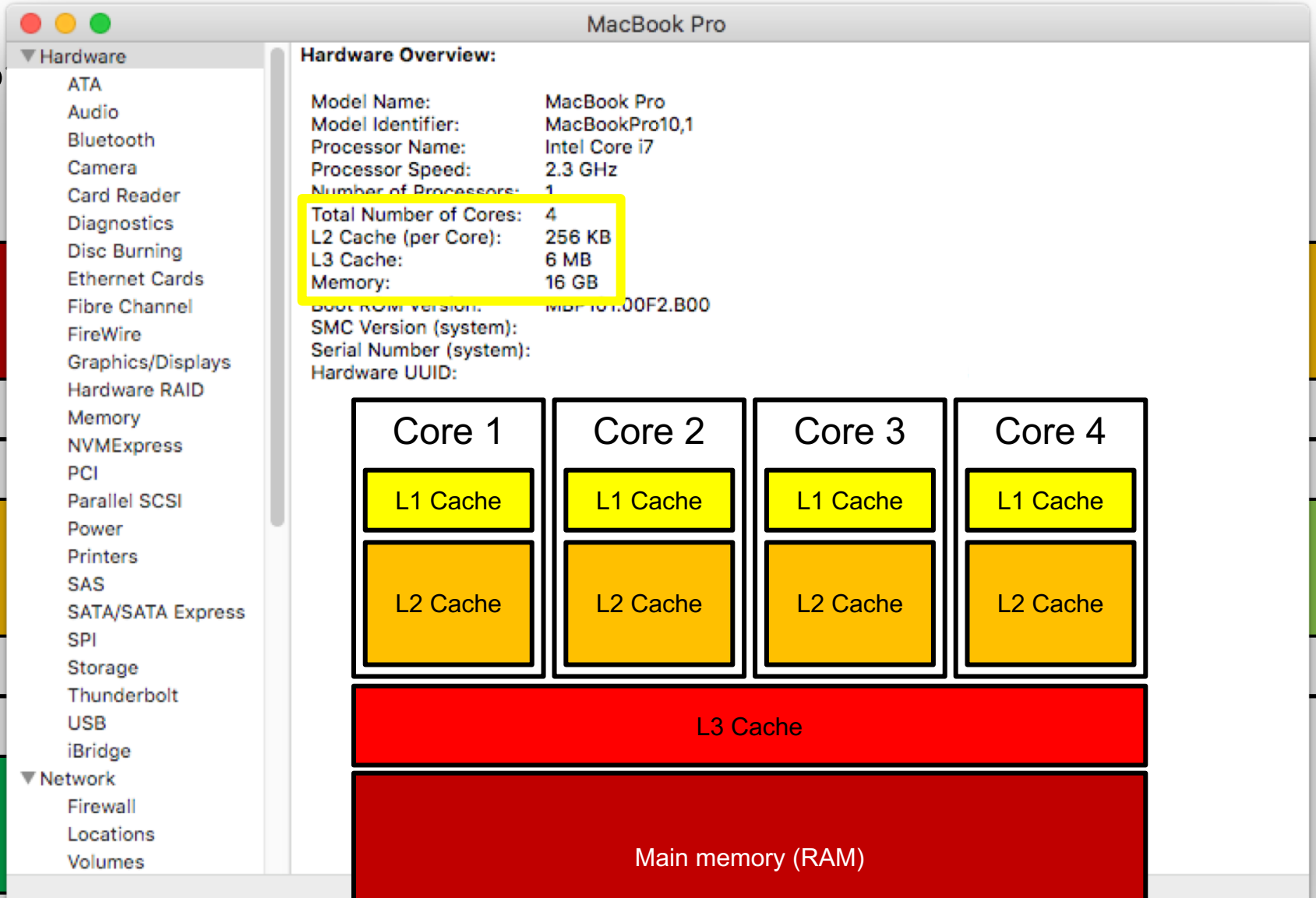


Matrices and memory

```
double[][] image = new double[6][4];
```

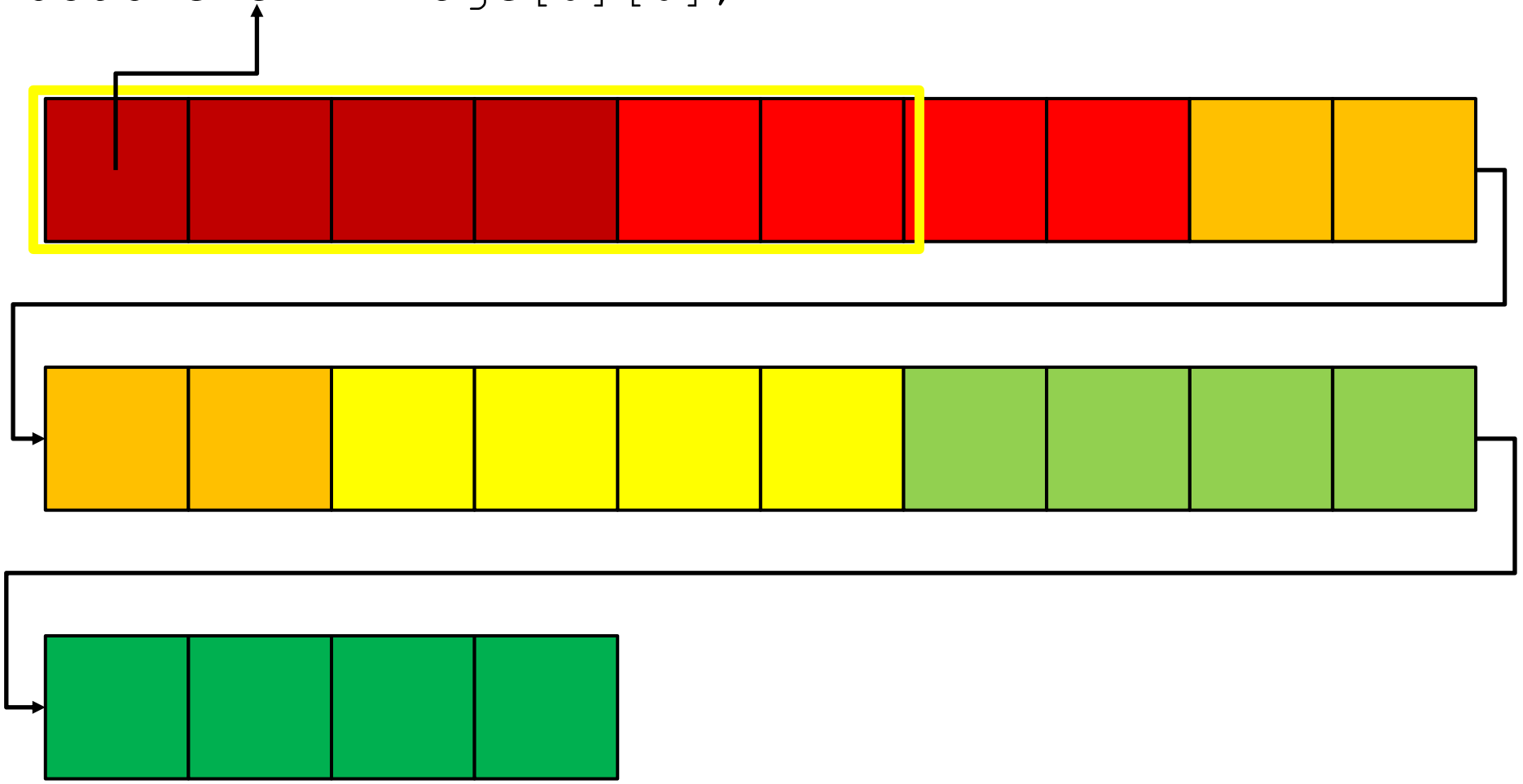


Matrices and memory



Matrices and memory

```
double[][] image = new double[4][6];
double c = image[0][0];
```



Multidimensional Arrays

- Occasionally, you will need to represent n-dimensional data structures.
- In Java, you can create n-dimensional arrays for any integer n.

- The way to declare two-dimensional array variables and create two-dimensional arrays can be generalized to declare n-dimensional array variables and create n-dimensional arrays for n >= 3.

Multidimensional Arrays

```
double[][][] scores = {
   {{7.5, 20.5}, {9.0, 22.5}, {15, 33.5}, {13, 21.5}, {15, 2.5}},
   {{4.5, 21.5}, {9.0, 22.5}, {15, 34.5}, {12, 20.5}, {14, 9.5}},
   {{6.5, 30.5}, {9.4, 10.5}, {11, 33.5}, {11, 23.5}, {10, 2.5}},
   {{6.5, 23.5}, {9.4, 32.5}, {13, 34.5}, {11, 20.5}, {16, 7.5}},
   {{8.5, 26.5}, {9.4, 52.5}, {13, 36.5}, {13, 24.5}, {16, 2.5}},
   {{9.5, 20.5}, {9.4, 42.5}, {13, 31.5}, {12, 20.5}, {16, 6.5}}
};
```