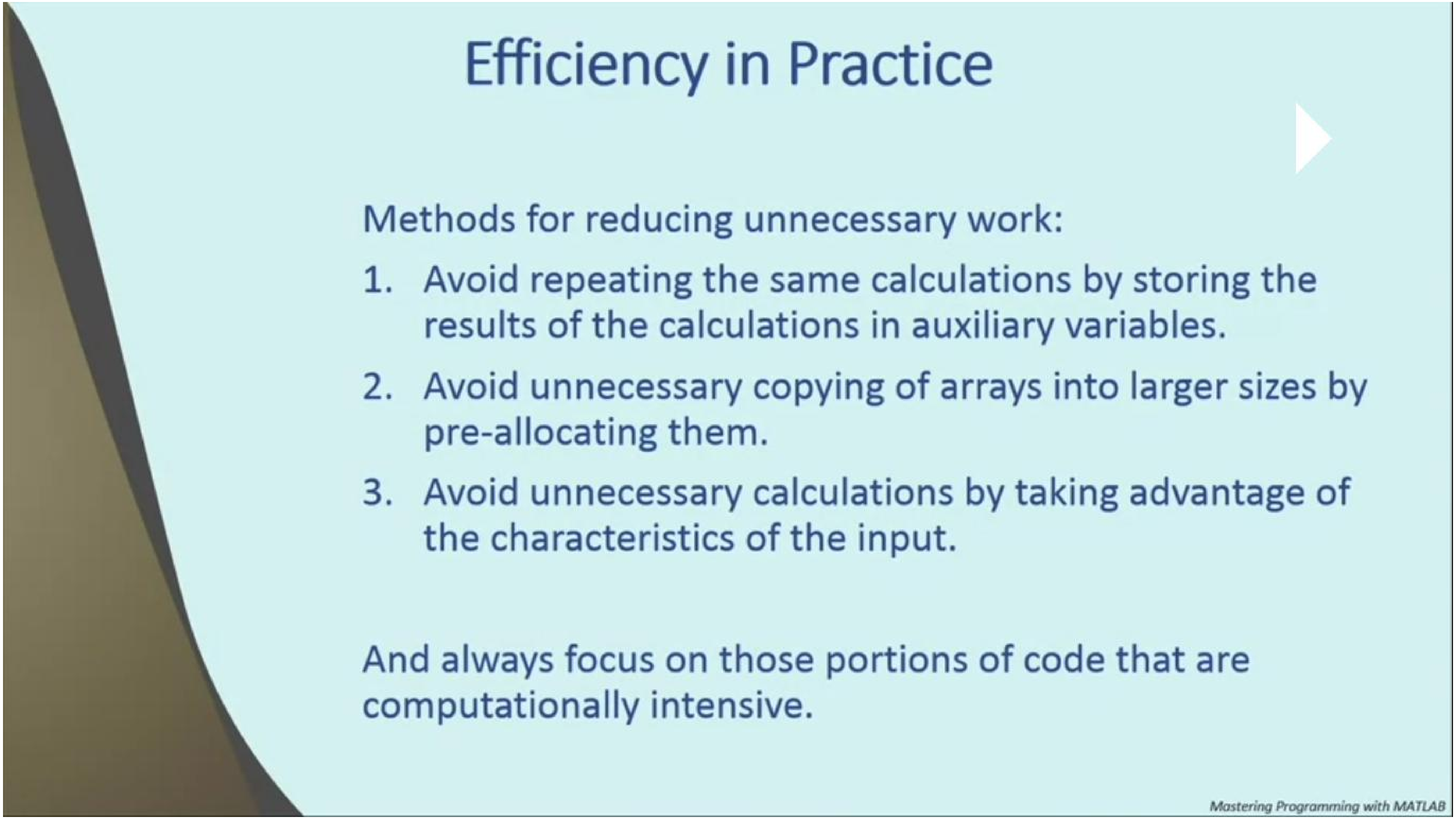
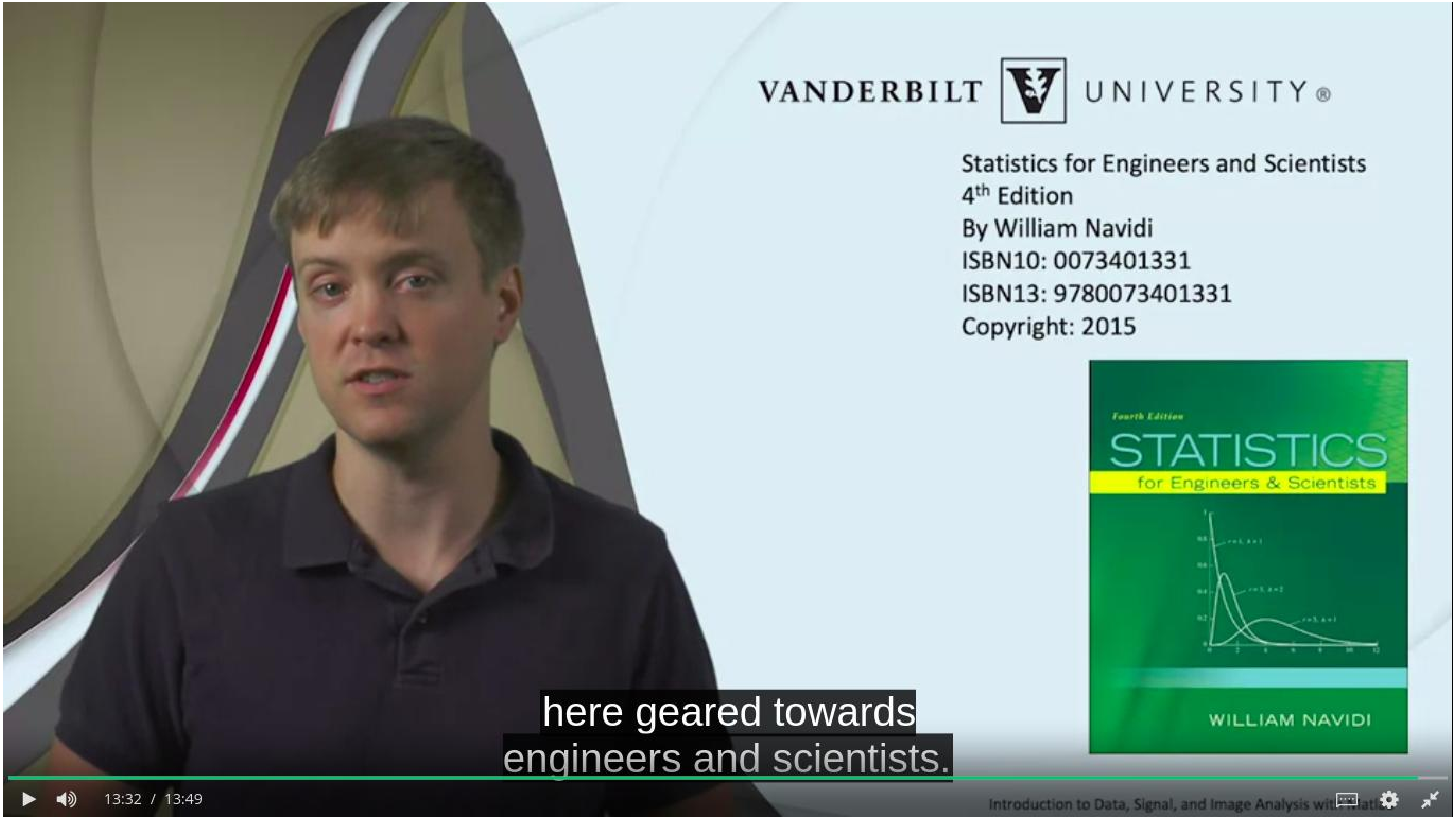


Efficiency in Practice

Methods for reducing unnecessary work:

1. Avoid repeating the same calculations by storing the results of the calculations in auxiliary variables.
2. Avoid unnecessary copying of arrays into larger sizes by pre-allocating them.
3. Avoid unnecessary calculations by taking advantage of the characteristics of the input.

And always focus on those portions of code that are computationally intensive.

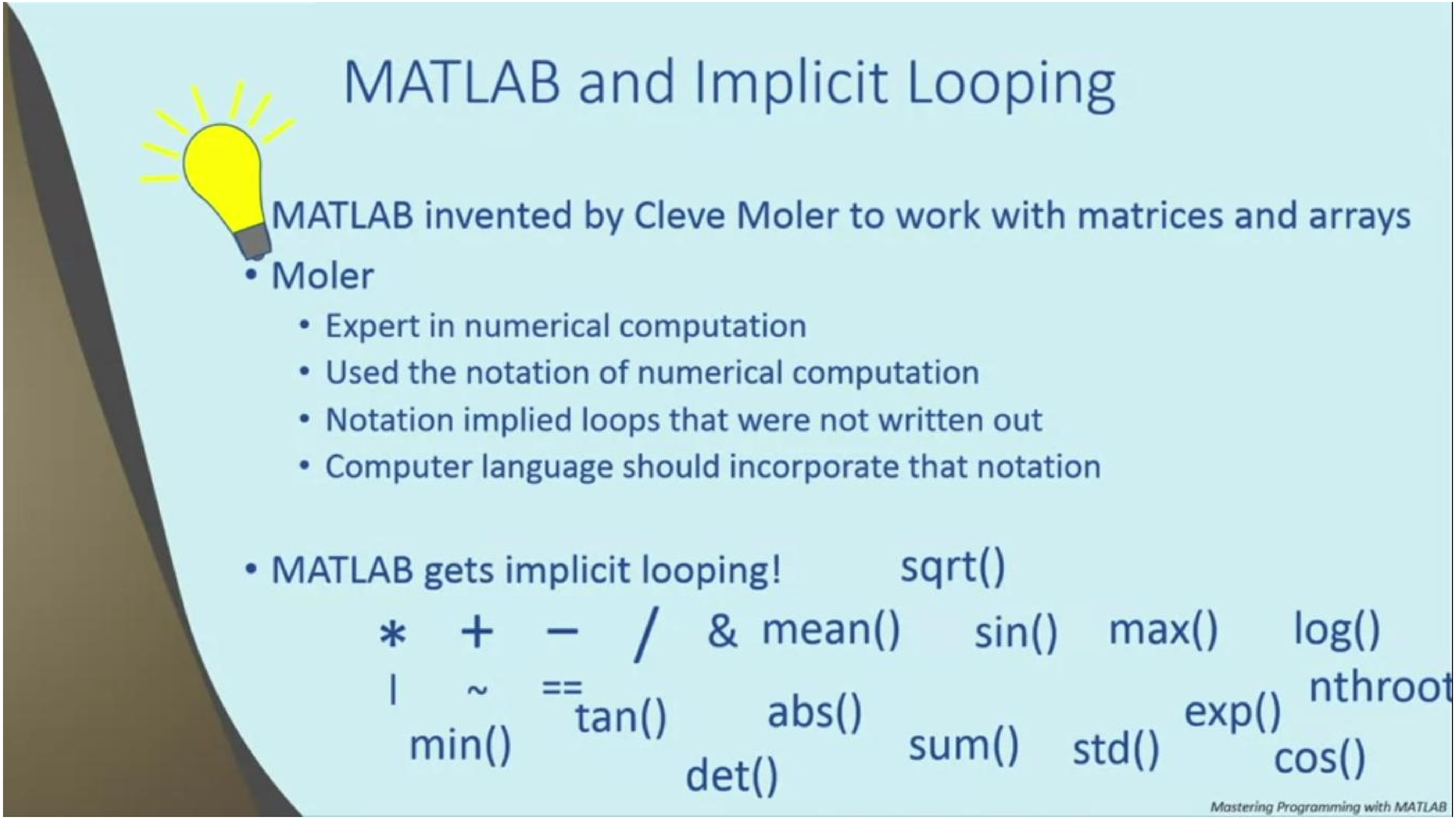


Statistics for Engineers and Scientists
4th Edition
By William Navidi
ISBN10: 0073401331
ISBN13: 9780073401331
Copyright: 2015

Fourth Edition
STATISTICS
for Engineers & Scientists
WILLIAM NAVIDI

here geared towards
engineers and scientists.

13:32 / 13:49
Introduction to Data, Signal, and Image Analysis wit



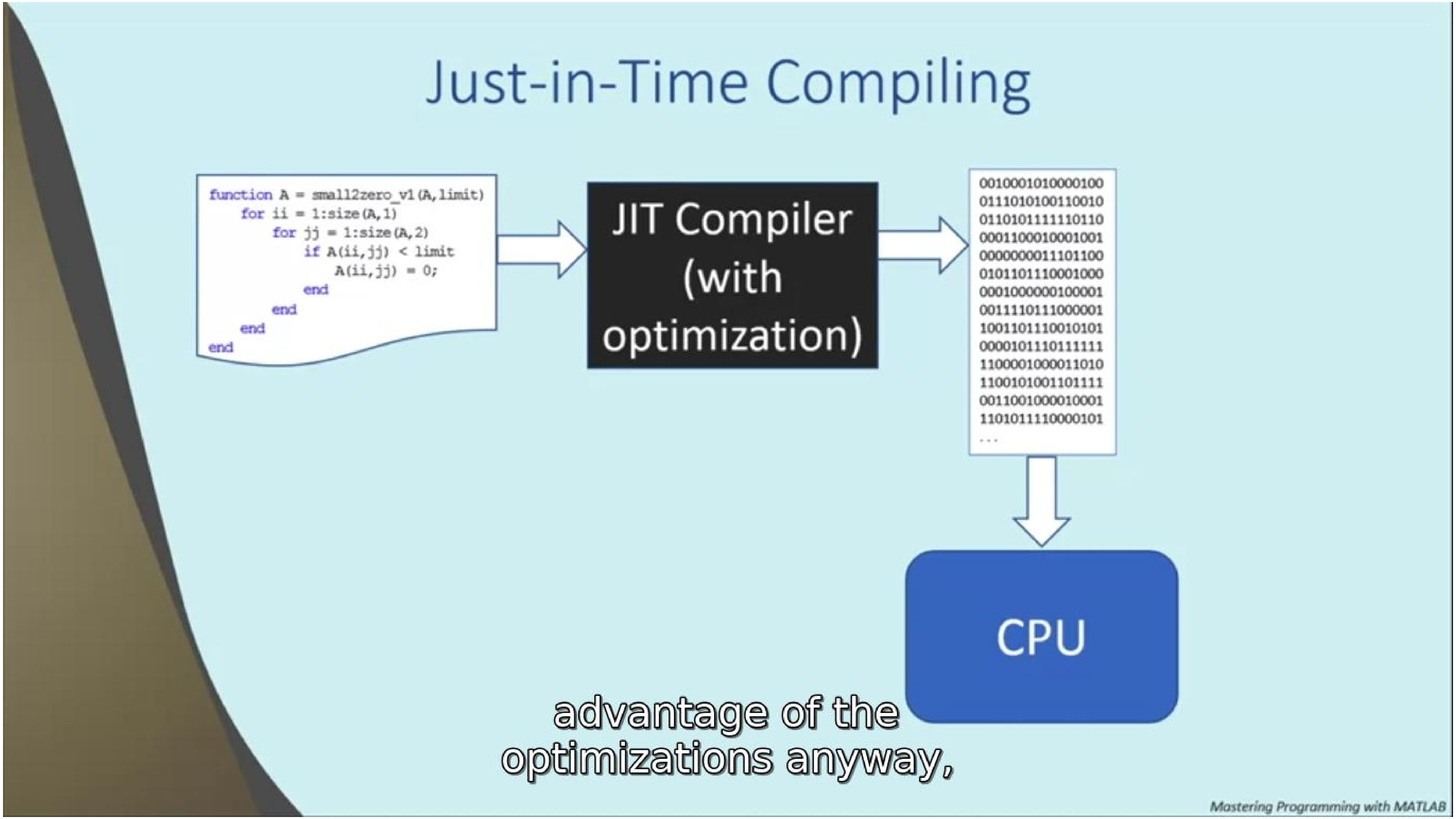
MATLAB and Implicit Looping

- MATLAB invented by Cleve Moler to work with matrices and arrays
- Moler
  - Expert in numerical computation
  - Used the notation of numerical computation
  - Notation implied loops that were not written out
  - Computer language should incorporate that notation

- MATLAB gets implicit looping!

sqrt()
*
+
−
/
&
mean()
sin()
max()
log()
|
~
==
tan()
abs()
exp()
nthroot
min()
sum()
std()
cos()
det()

Mastering Programming with MATLAB



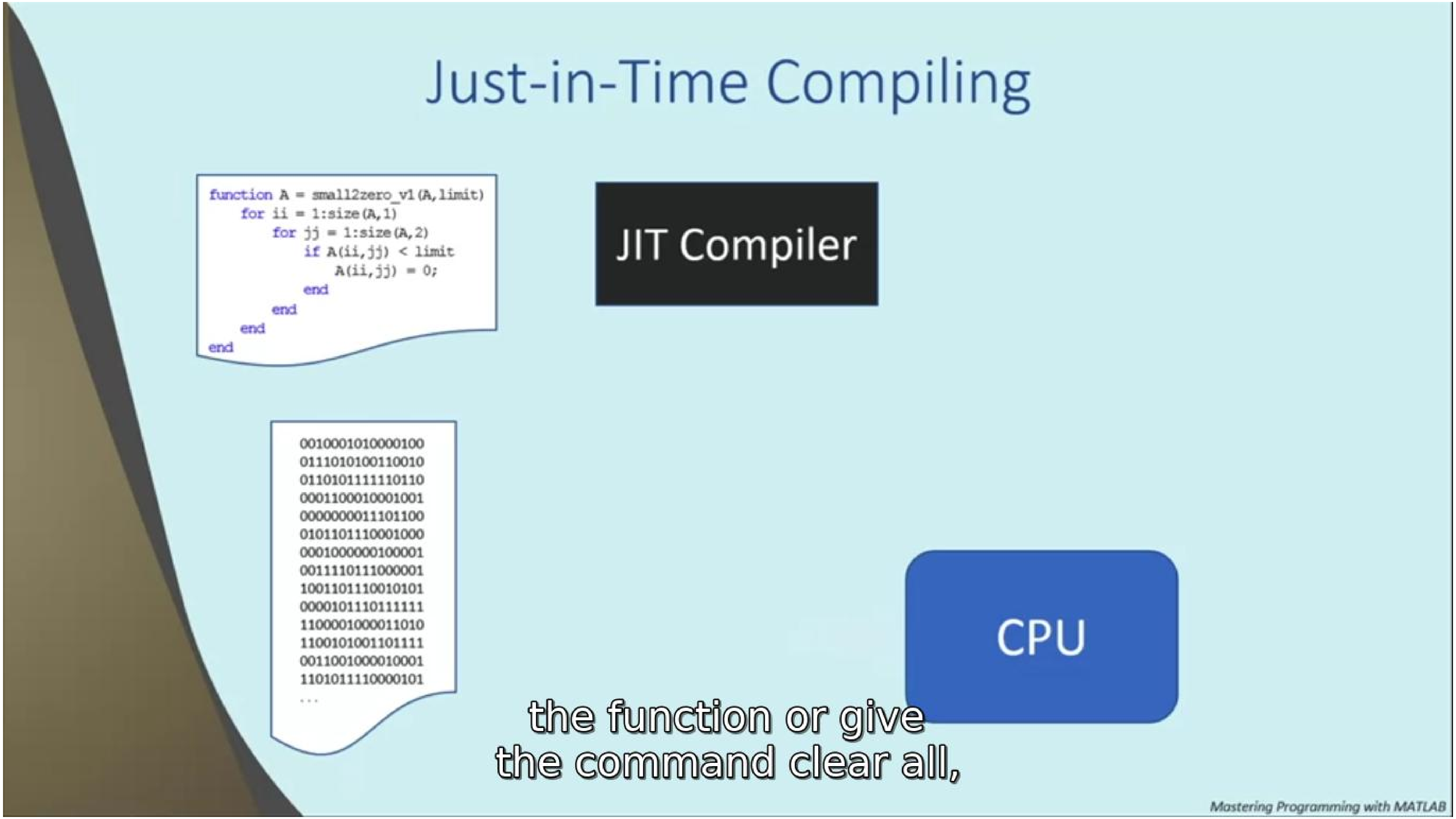
Just-in-Time Compiling

```
function A = small2zero_v1(A,limit)
    for ii = 1:size(A,1)
        for jj = 1:size(A,2)
            if A(ii,jj) < limit
                A(ii,jj) = 0;
            end
        end
    end
end
```

```
0010001010000100
0111010100110010
0110101111110110
0001100010001001
0000000011101100
0101101110001000
0001000000100001
0011110111000001
1001101110010101
0000101110111111
1100001000011010
1100101001101111
0011001000010001
1101011110000101
...
```

advantage of the optimizations anyway,

Mastering Programming with MATLAB



Just-in-Time Compiling

```
function A = small2zero_v1(A,limit)
    for ii = 1:size(A,1)
        for jj = 1:size(A,2)
            if A(ii,jj) < limit
                A(ii,jj) = 0;
            end
        end
    end
end
```

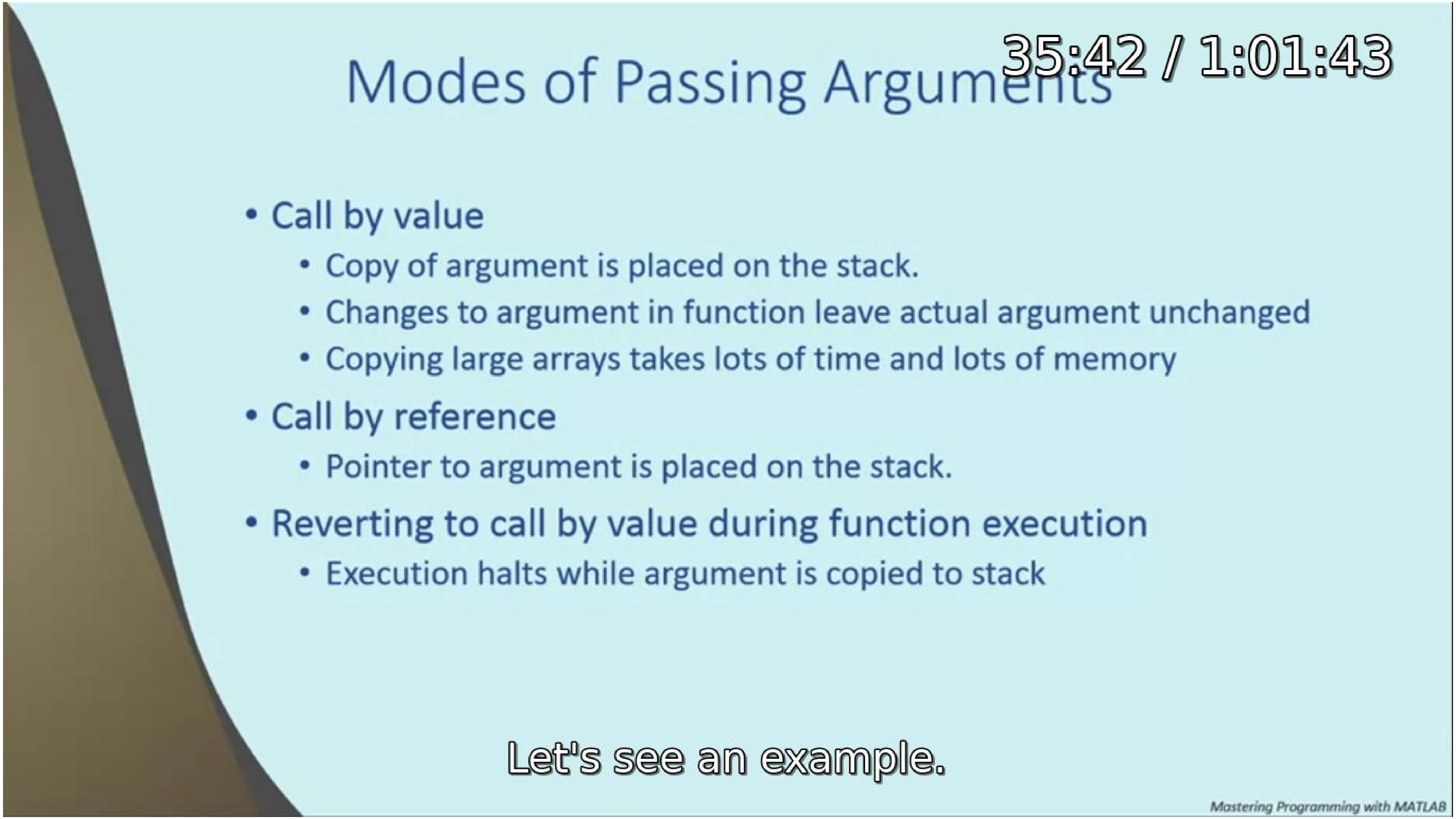
JIT Compiler

```
0010001010000100
0111010100110010
0110101111110110
0001100010001001
0000000011101100
0101101110001000
0001000000100001
0011110111000001
1001101110010101
0000101110111111
1100001000011010
1100101001101111
0011001000010001
1101011110000101
...
```

CPU

the function or give
the command clear all,

Mastering Programming with MATLAB



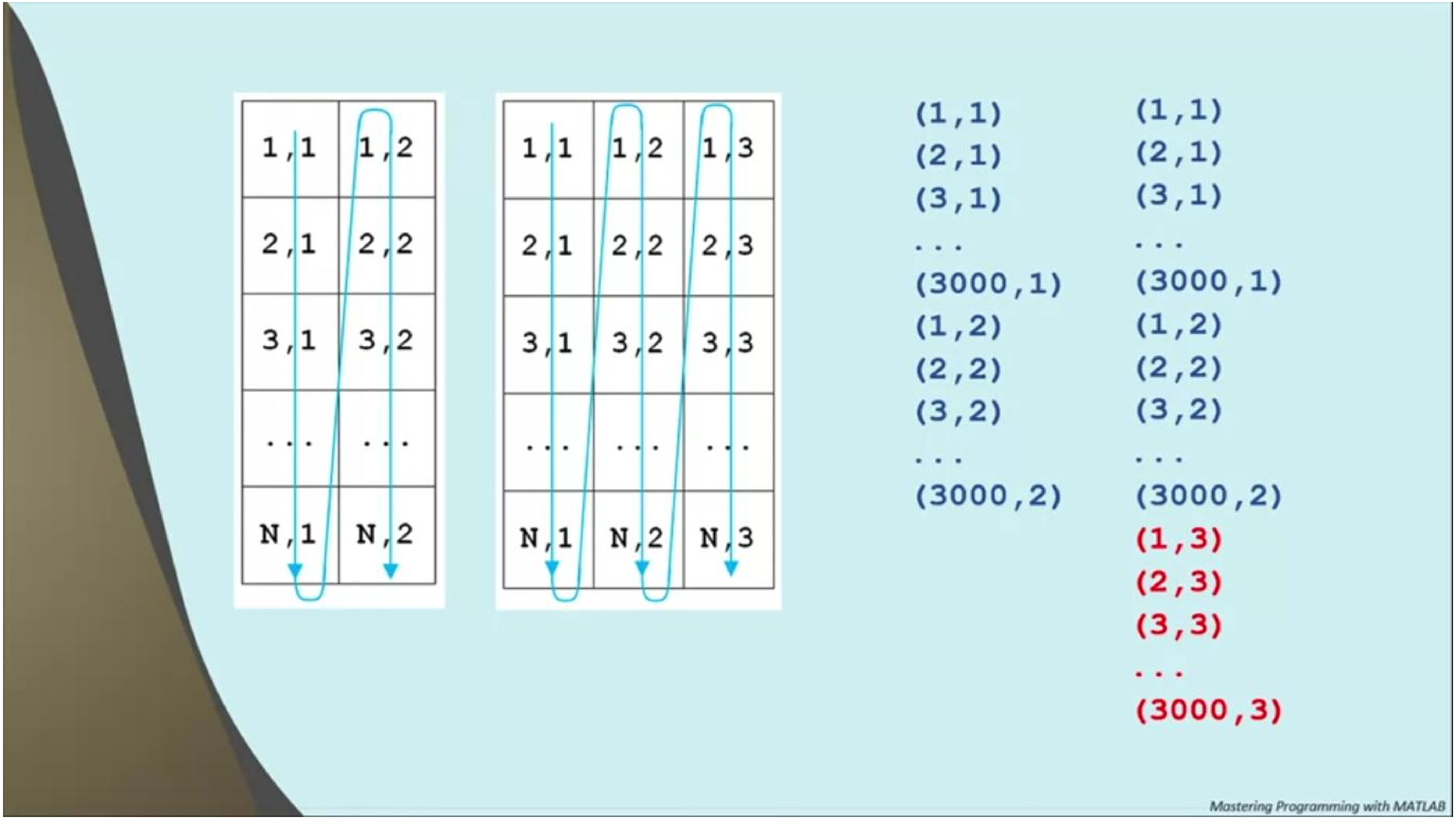
35:42 / 1:01:43

Modes of Passing Arguments

- Call by value
  - Copy of argument is placed on the stack.
  - Changes to argument in function leave actual argument unchanged
  - Copying large arrays takes lots of time and lots of memory
- Call by reference
  - Pointer to argument is placed on the stack.
- Reverting to call by value during function execution
  - Execution halts while argument is copied to stack

Let's see an example.

Mastering Programming with MATLAB

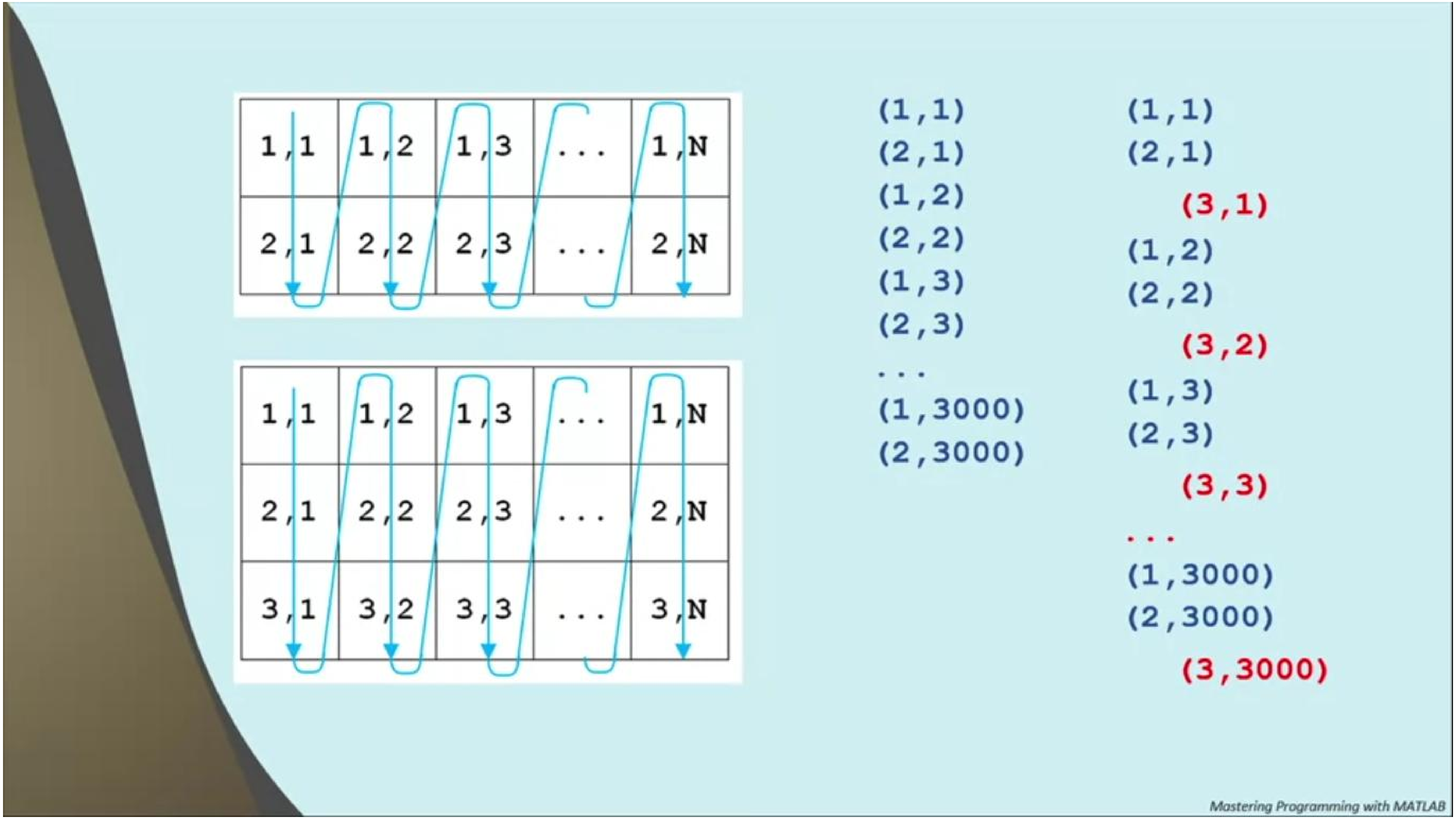


1,1	1,2
2,1	2,2
3,1	3,2
...	...
N,1	N,2

1,1	1,2	1,3
2,1	2,2	2,3
3,1	3,2	3,3
...	...	...
N,1	N,2	N,3

```
(1,1)
(2,1)
(3,1)
...
(3000,1)
(1,2)
(2,2)
(3,2)
...
(3000,2)
```

```
(1,1)
(2,1)
(3,1)
...
(3000,1)
(1,2)
(2,2)
(3,2)
...
(3000,2)
(1,3)
(2,3)
(3,3)
...
(3000,3)
```

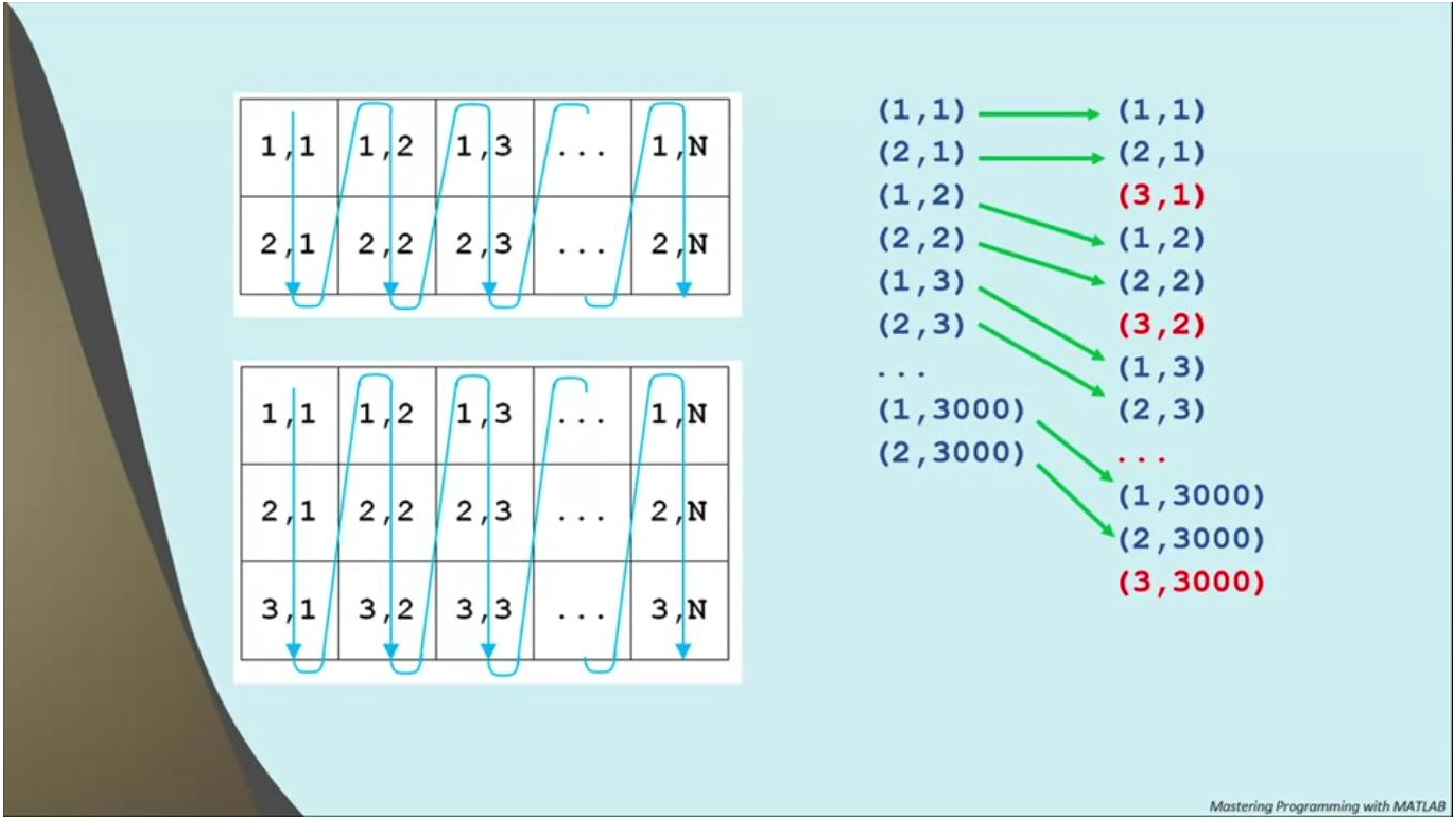


1,1	1,2	1,3	...	1,N
2,1	2,2	2,3	...	2,N

1,1	1,2	1,3	...	1,N
2,1	2,2	2,3	...	2,N
3,1	3,2	3,3	...	3,N

```
(1,1)
(2,1)
(1,2)
(2,2)
(1,3)
(2,3)
...
(1,3000)
(2,3000)
```

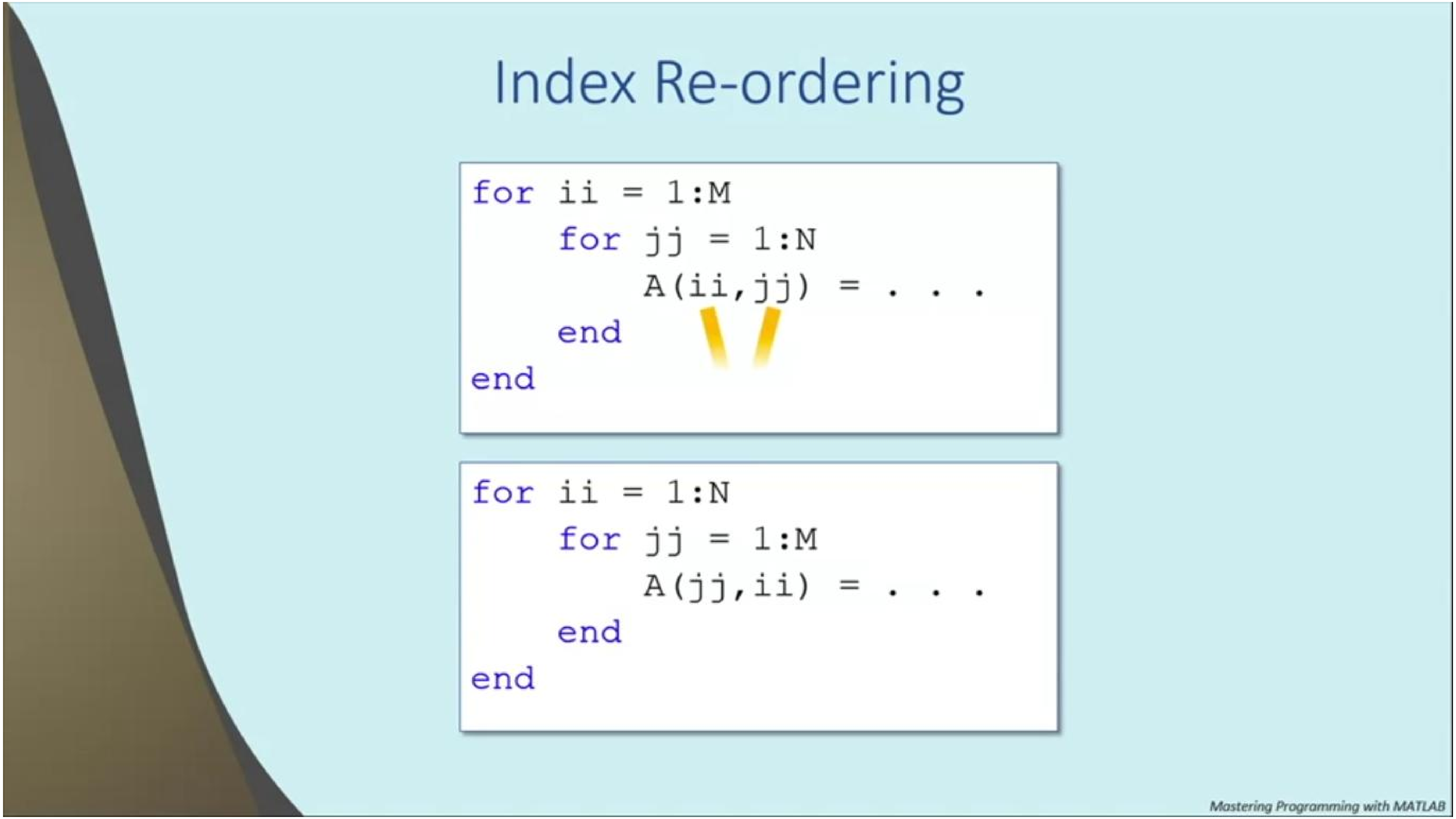
```
(1,1)
(2,1)
    (3,1)
(1,2)
(2,2)
    (3,2)
(1,3)
(2,3)
    (3,3)
...
(1,3000)
(2,3000)
    (3,3000)
```



1,1	1,2	1,3	...	1,N
2,1	2,2	2,3	...	2,N

1,1	1,2	1,3	...	1,N
2,1	2,2	2,3	...	2,N
3,1	3,2	3,3	...	3,N

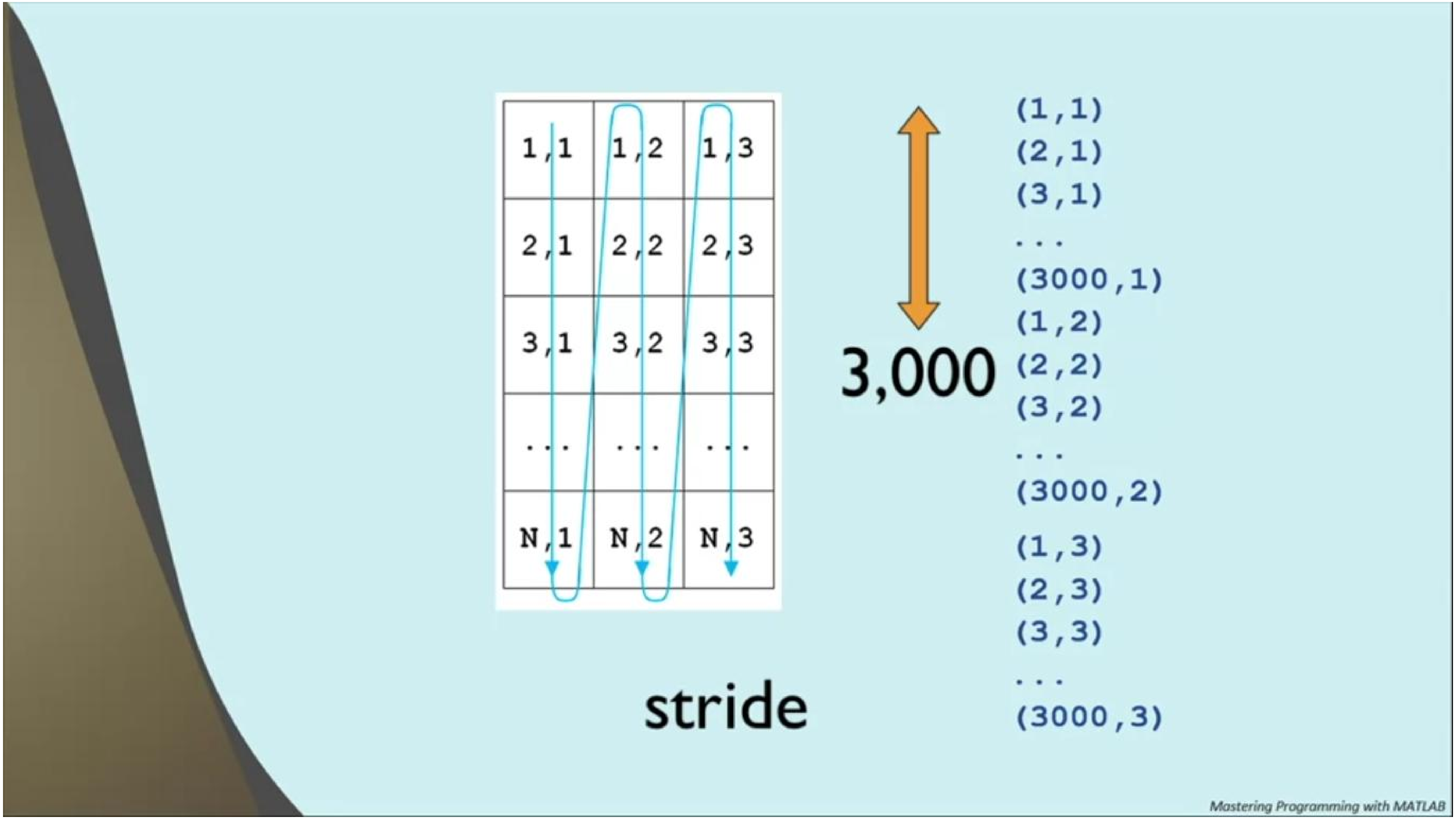
(1,1)	(1,1)
(2,1)	(2,1)
(1,2)	(3,1)
(2,2)	(1,2)
(1,3)	(2,2)
(2,3)	(3,2)
...	(1,3)
(1,3000)	(2,3)
(2,3000)	...
	(1,3000)
	(2,3000)
	(3,3000)



Index Re-ordering

```
for ii = 1:M
    for jj = 1:N
        A(ii,jj) = . . .
    end
end
```

```
for ii = 1:N
    for jj = 1:M
        A(jj,ii) = . . .
    end
end
```

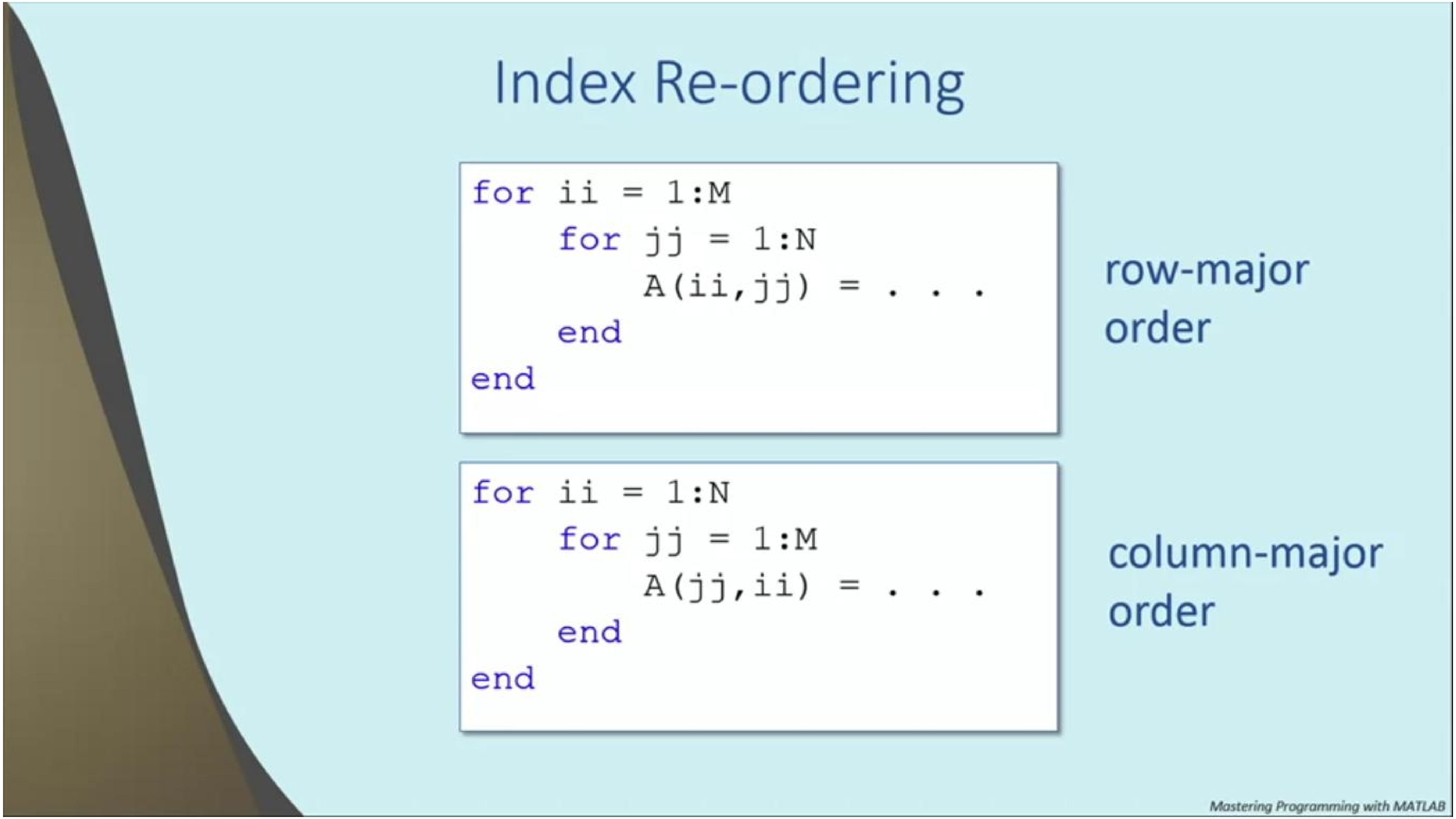


1,1	1,2	1,3
2,1	2,2	2,3
3,1	3,2	3,3
...	...	...
N,1	N,2	N,3

3,000

stride

```
(1,1)
(2,1)
(3,1)
...
(3000,1)
(1,2)
(2,2)
(3,2)
...
(3000,2)
(1,3)
(2,3)
(3,3)
...
(3000,3)
```



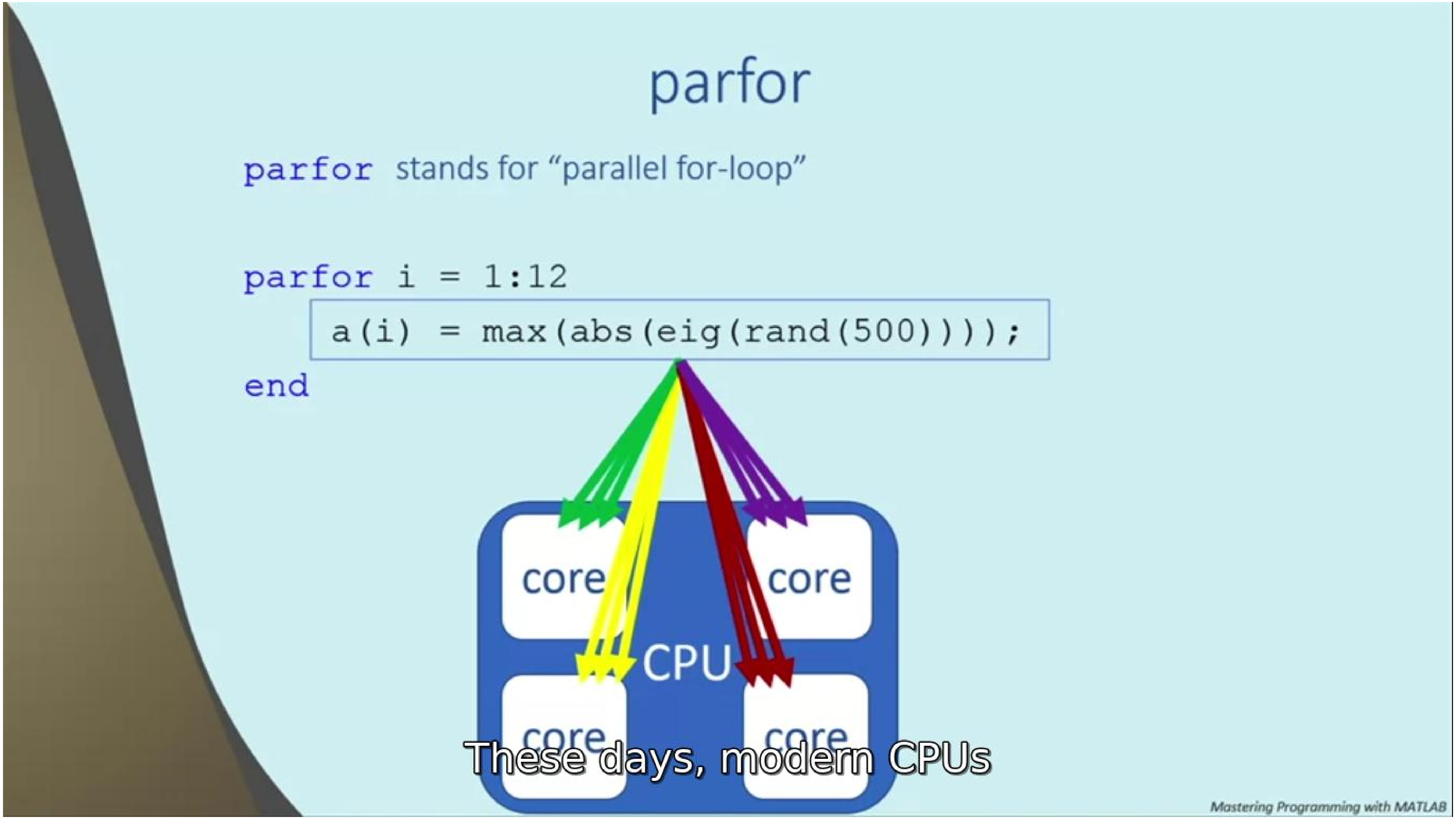
Index Re-ordering

```
for ii = 1:M
    for jj = 1:N
        A(ii,jj) = . . .
    end
end
```

row-major order

```
for ii = 1:N
    for jj = 1:M
        A(jj,ii) = . . .
    end
end
```

column-major order



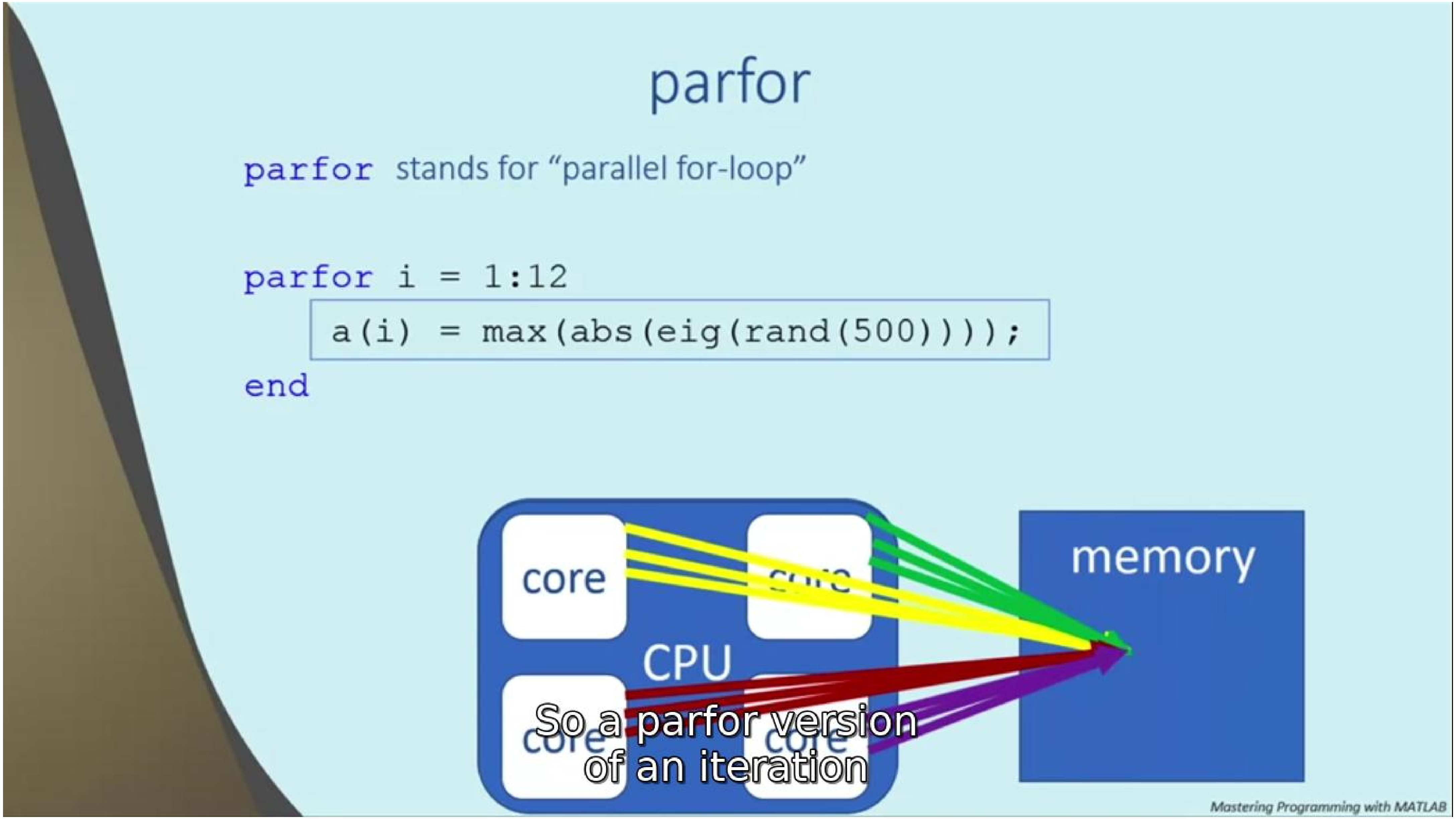
parfor

`parfor` stands for "parallel for-loop"

```
parfor i = 1:12
    a(i) = max(abs(eig(rand(500))));
end
```

These days, modern CPUs

Mastering Programming with MATLAB



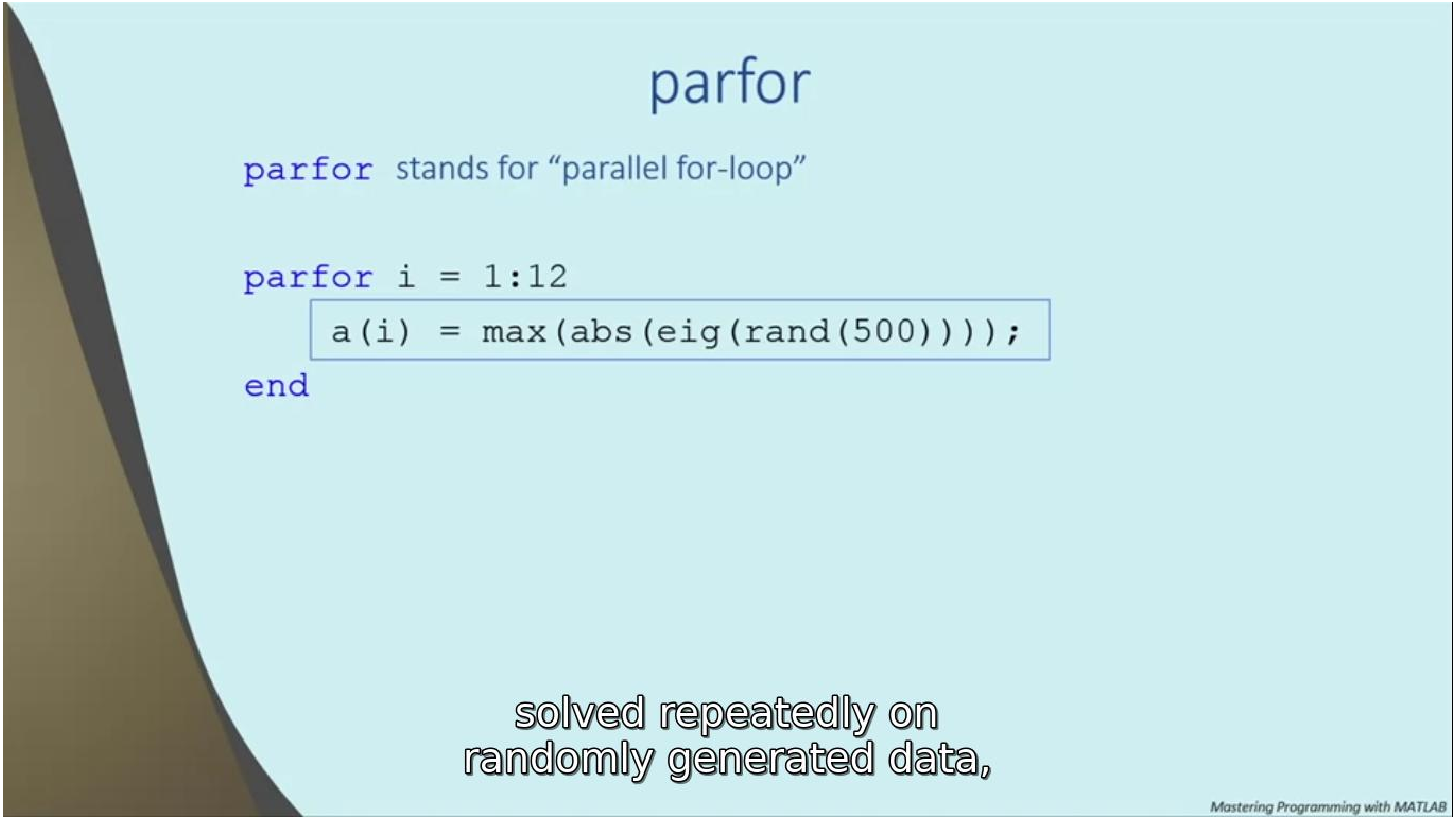
parfor

parfor stands for “parallel for-loop”

```
parfor i = 1:12
    a(i) = max(abs(eig(rand(500))));
end
```

So a parfor version
of an iteration

Mastering Programming with MATLAB

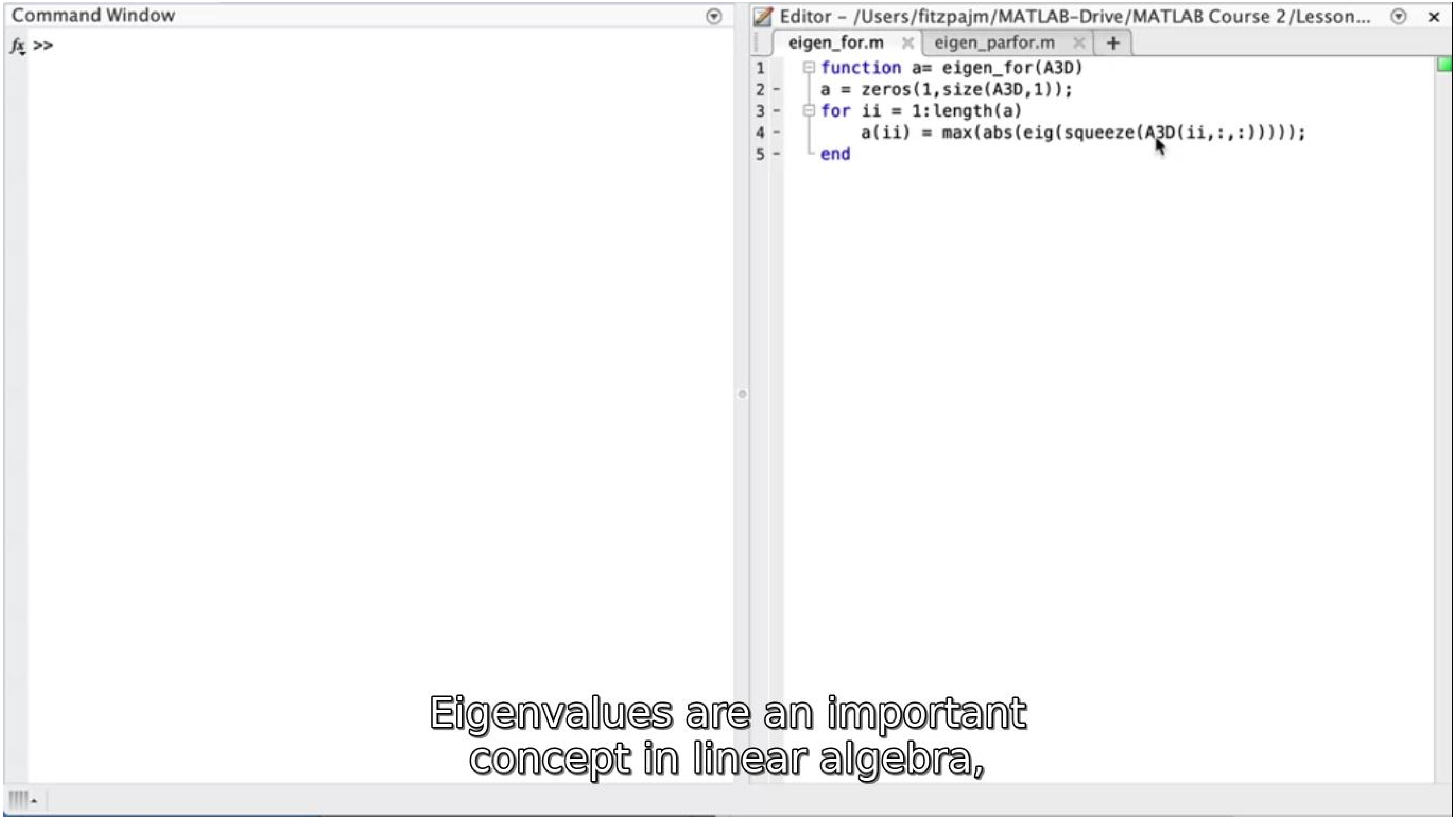


parfor

`parfor` stands for “parallel for-loop”

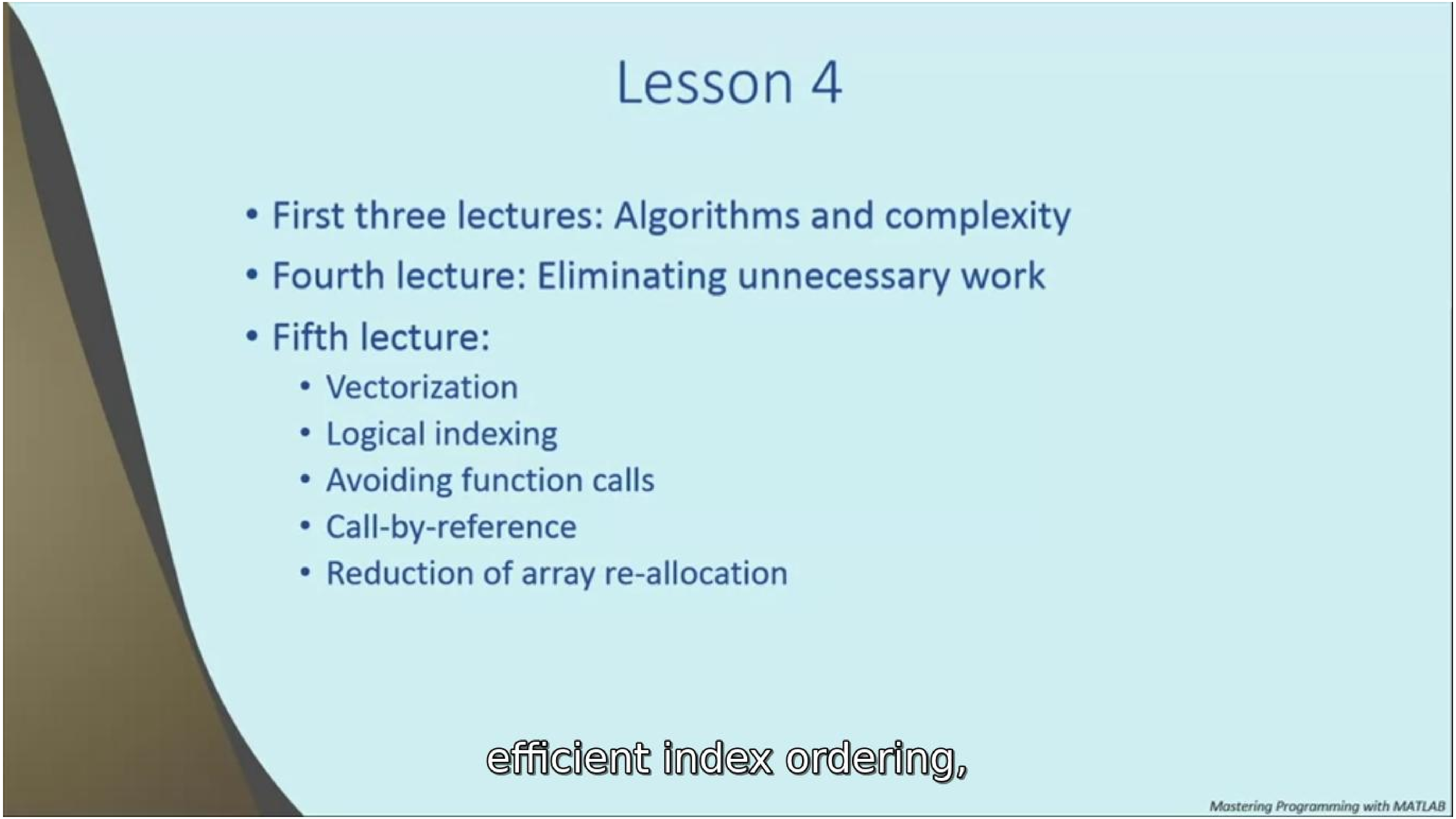
```
parfor i = 1:12
    a(i) = max(abs(eig(rand(500))));
end
```

solved repeatedly on
randomly generated data,


Command Window
fx >>
Editor - /Users/fitzpajm/MATLAB-Drive/MATLAB Course 2/Lesson...
eigen_for.m
eigen_parfor.m
+

```
function a= eigen_for(A3D)
a = zeros(1,size(A3D,1));
for ii = 1:length(a)
    a(ii) = max(abs(eig(squeeze(A3D(ii,:,:)))));
end
```

Eigenvalues are an important
concept in linear algebra,



Lesson 4

- First three lectures: Algorithms and complexity
- Fourth lecture: Eliminating unnecessary work
- Fifth lecture:
  - Vectorization
  - Logical indexing
  - Avoiding function calls
  - Call-by-reference
  - Reduction of array re-allocation

efficient index ordering,