ENGR 101 – Chapter 19

More Data Structures

3/20/21

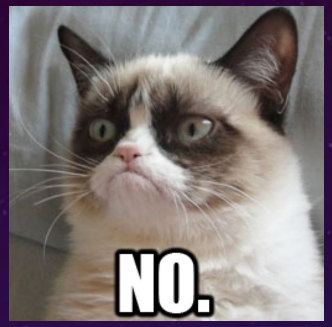
INTRO

Remember Matrices? They were so nice…

data

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35

Is This a Good Approach?



Now we have FIVE variables containing our data instead of one variable. That's code duplication!

vec1	1	2	3	4	5	6	7
vec2	8	9	10	11	12	13	14
vec3	15	16	17	18	19	20	21
vec4	22	23	24	25	26	27	28
vec5	29	30	31	32	33	34	35

WTF IS A VECTOR OF VECTORS?

Review: Declaring the Element Type of a vector

- Declare a vector like this:

```
vector<int> someInts;
```

In addition to the base type of vector, provide the type of elements it will hold.

- A vector can store elements of any type, as long as they match the type with which it is declared.

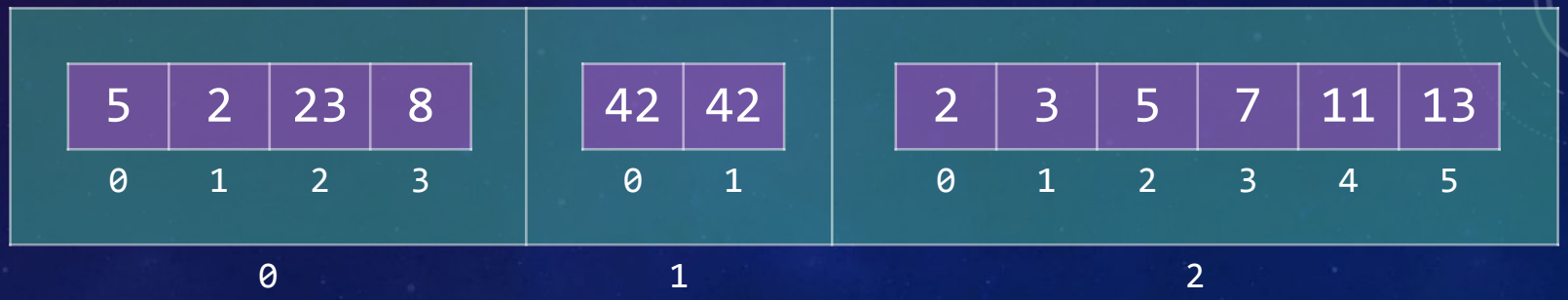
```
vector<double> someDoubles;
vector<bool> someBools;
```

vectors of vectors

- The element type for a `vector` can be anything, even another `vector` type!
- This allows us to create nested `vector`s:

Note: On some compilers, this space is necessary to prevent confusion with the >> operator.

```
vector< vector<int> > someVectorsOfInts;
```



Outer Vector vs. Inner Vector

base type

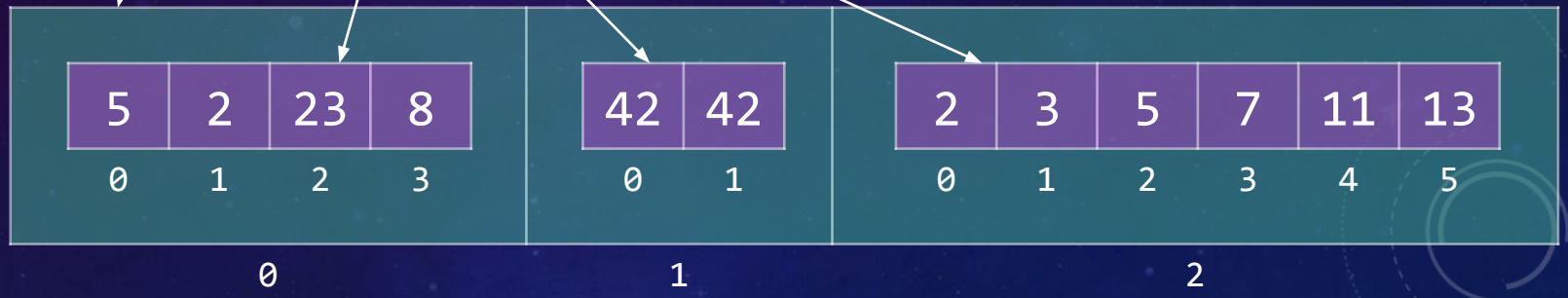
```
vector< vector<int> > someVectorsOfInts;
```

outer vector

inner vector

inner vector

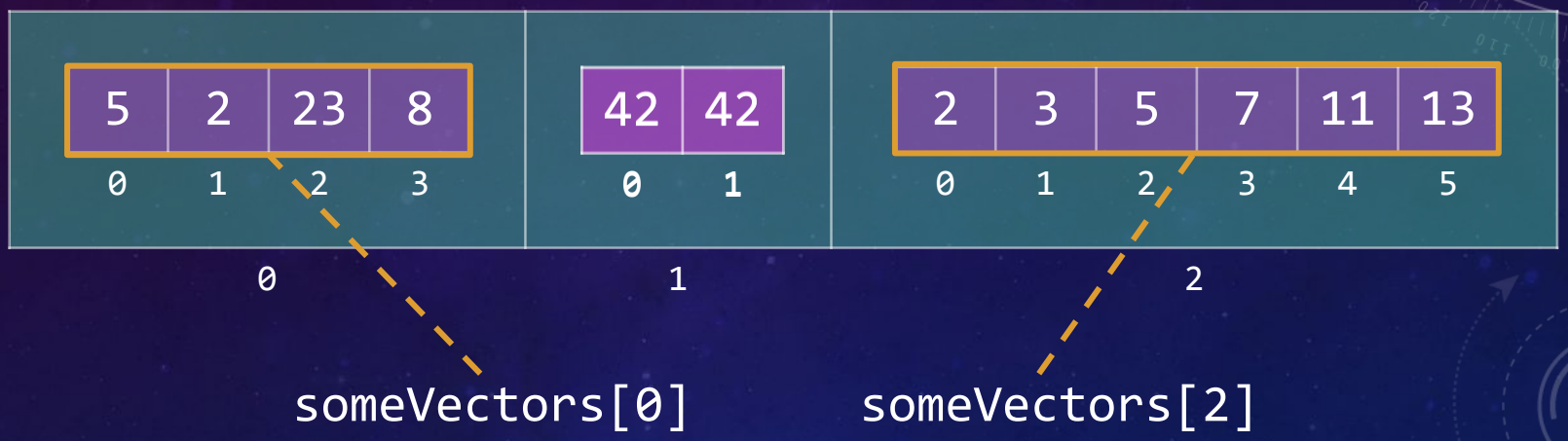
inner vector



INDEXING IN A VECTOR OF VECTORS

Indexing in a vector of vectors

- Indexing in a vector of vectors <u>selects a vector</u>.

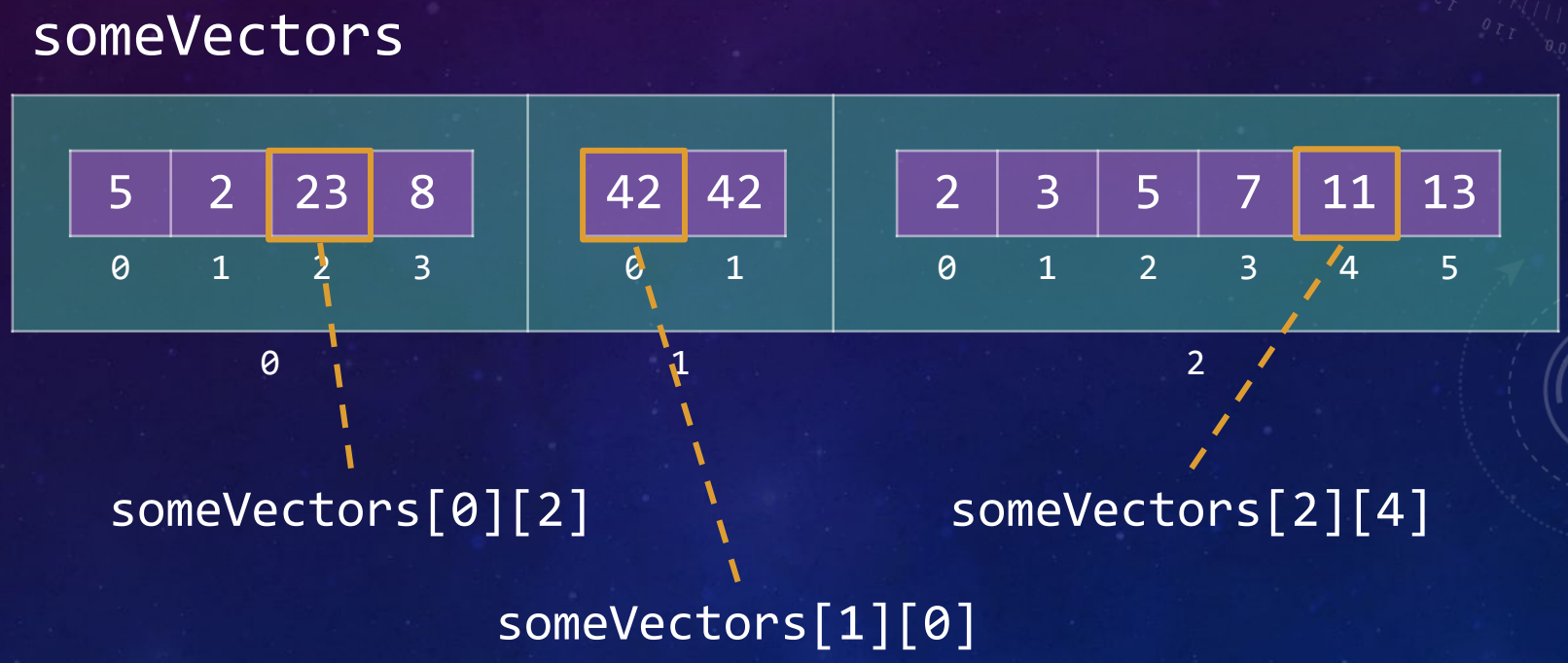



```
someVectors[2] = someVectors[1]
```

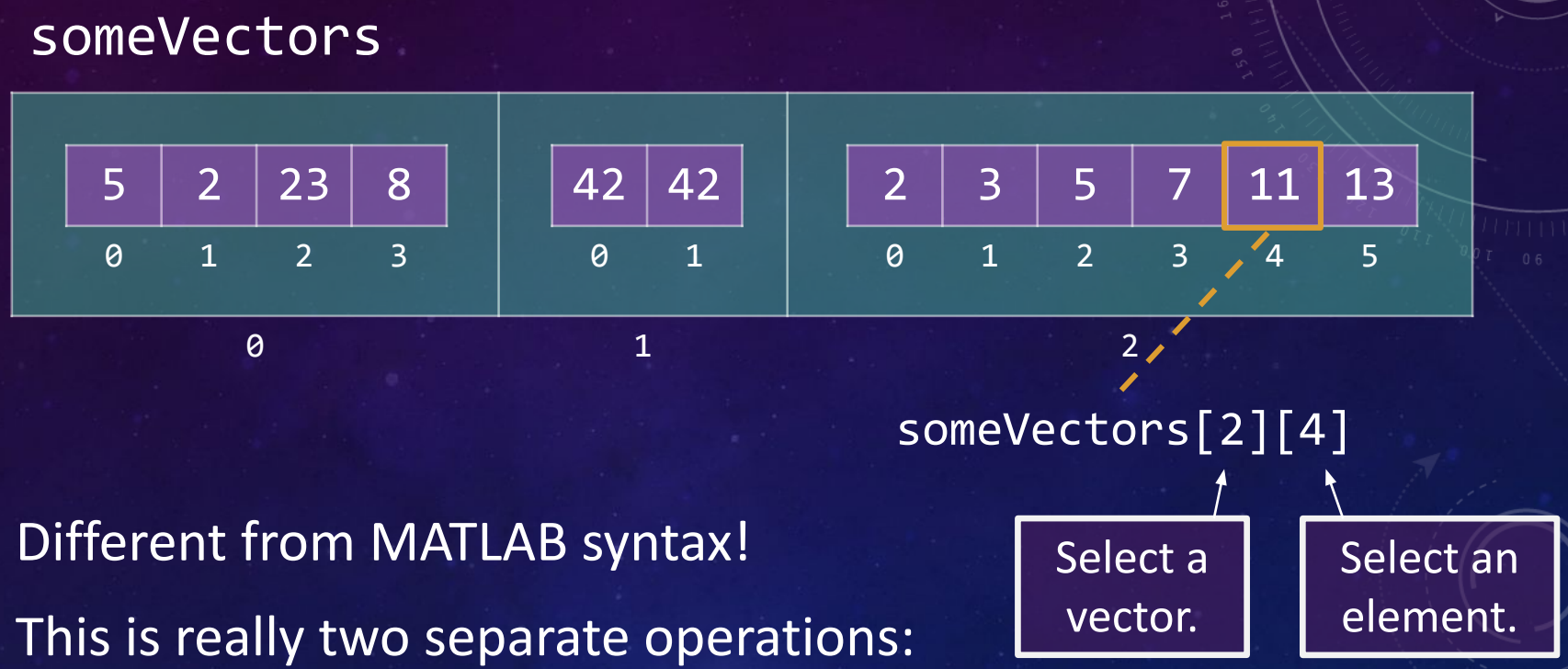
(modifies `someVectors` as above)

Indexing in a vector of vectors

- Indexing in a vector of vectors selects a vector.
- To select an element from that vector, index again.



Order of Indexing



- Different from MATLAB syntax!
- This is really two separate operations:
  - First, select the vector at index 2.
  - Then, from that vector, select the element at index 4.

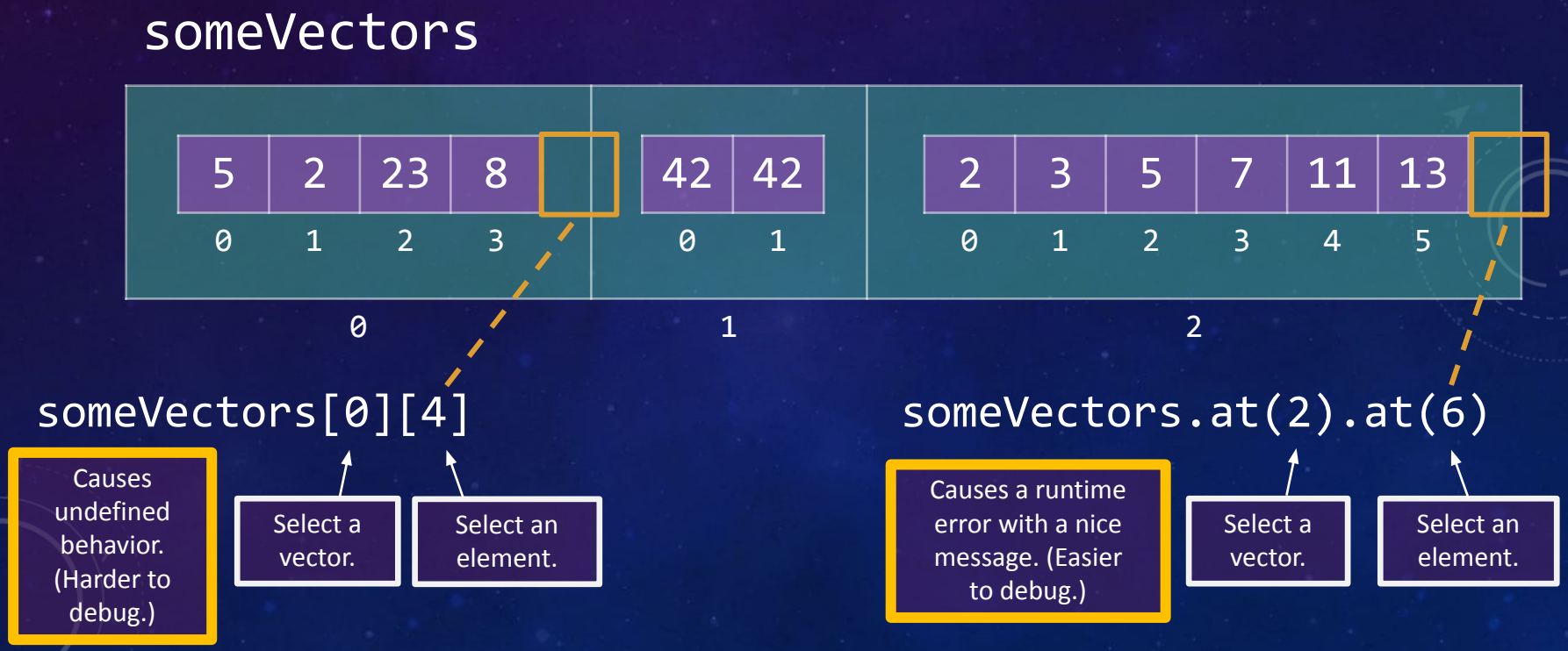
Recall: Indexing Out Of Bounds

- As with `strings`, it's possible to index off the end of a `vector`, which results in undefined behavior at runtime.
- Basically, this goes to whatever memory happens to be next to the `vector`.
  - Maybe you get "lucky" and this memory wasn't important.
  - Maybe you mess up another variable that happens to be there.
  - Maybe your program isn't allowed to use that chunk of memory!
    - This causes a crash called a **segmentation fault** *(aka seg fault)*.
  - Maybe it catches on fire[1]. Who knows!

1 We're just kidding. This won't actually happen.

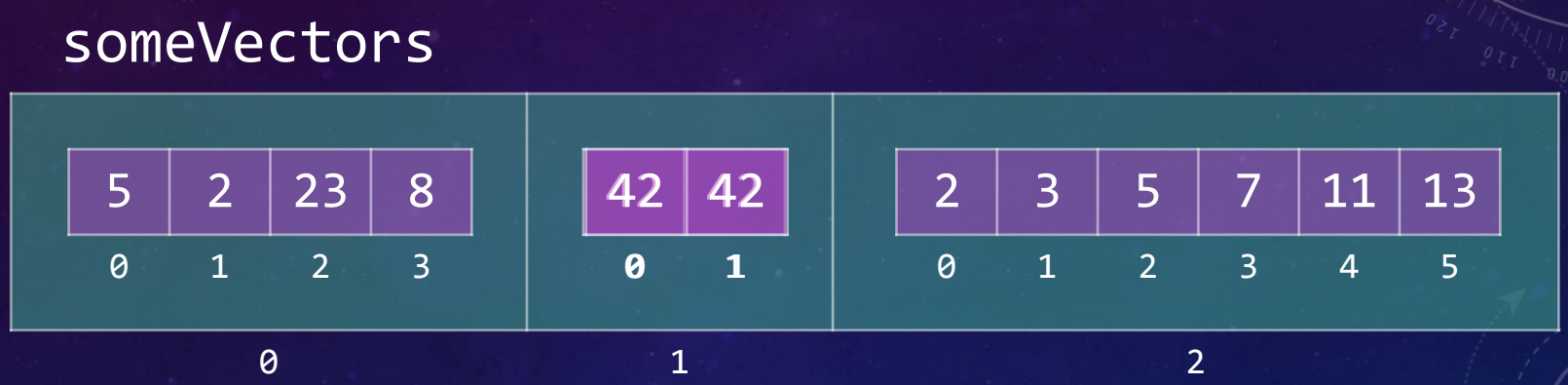
The at Function

- Again, you have the option to use the `.at` function rather than indexing with the square brackets.
  - This contains an implicit check to make sure the index is valid.
- The tradeoff is that `at` is slightly slower than `[]`.



Caution!

- If you use indexing to select something and then assign it into a variable, **you make a copy**!



- Consider this code:

```
vector<int> vec = someVectors[1];
vec[0] = 24;
```

- This doesn't change the original!

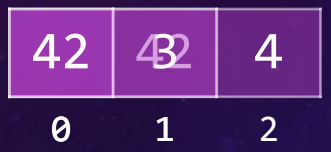
BUILDING AND MODIFYING VECTORS OF VECTORS

Building vectors of vectors

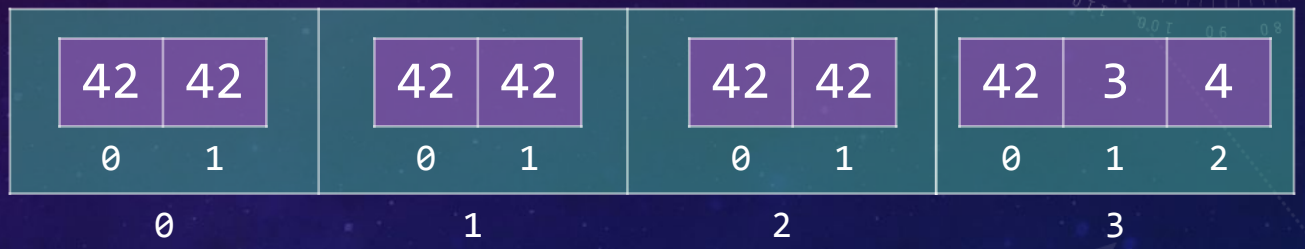
- Use vector constructors:

```
vector<int> someInts(2, 42);
vector< vector<int> > someVectors(3, someInts);
```

someInts



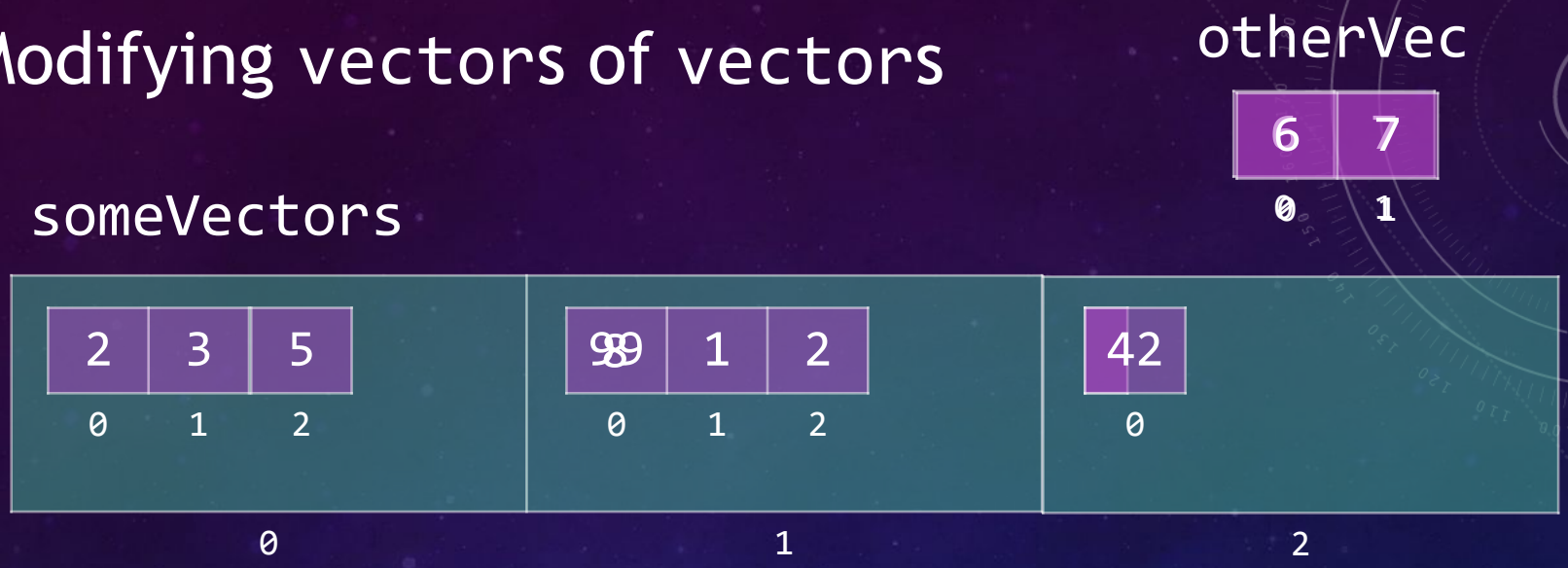
someVectors



- Use push_back/pop_back:

```
someInts.pop_back();
someInts.push_back(3);
someInts.push_back(4);
someVectors.push_back(someInts);
```

Modifying vectors of vectors




```
someVectors[0].pop_back();
someVectors[1][0] = 99;
someVectors[1] = otherVec;
someVectors.push_back(vector<int>());
someVectors[2].push_back(42);
```

Always be mindful of whether an operation is being performed on the "outer" vector or one of the "inner" vectors.

4 min

Exercise

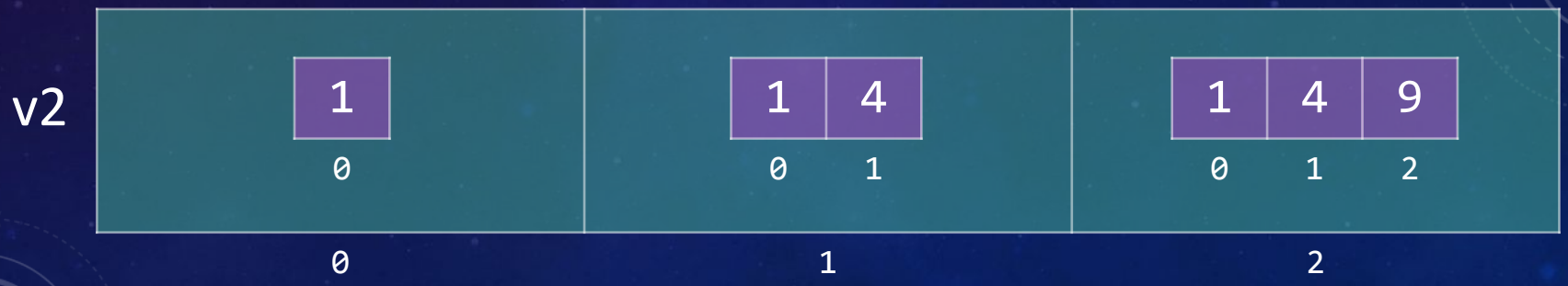
- Draw a diagram showing the contents of v2 after this code:

```
vector< vector<int> > v2; // starts empty
vector<int> v; // starts empty

for (int i = 1; i < 4; ++i) {
  v.push_back(i*i);
  v2.push_back(v);
}
```

Solution

- Draw a diagram showing the contents of v2 after this code:

```
vector< vector<int> > v2; // starts empty
vector<int> v; // starts empty

for (int i = 1; i < 4; ++i) {
  v.push_back(i*i);
  v2.push_back(v);
}
```



Exercise

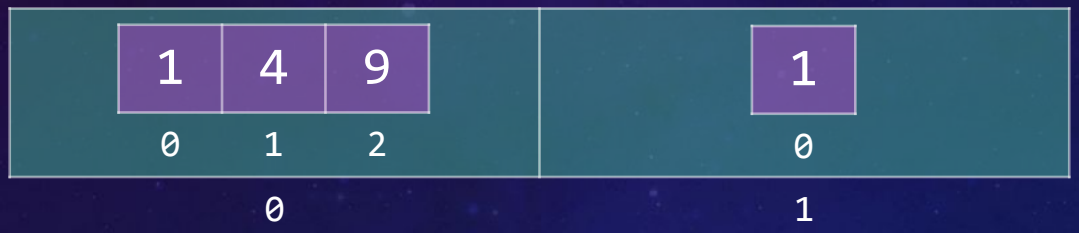
4 min

- Write a short piece of code to change from each state to the next:

v2

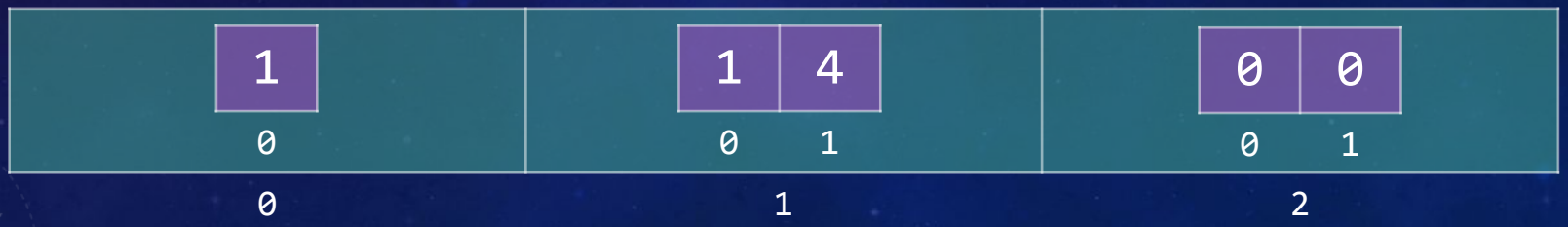

```
// Add some code here
```

v2




```
// Add some code here
```

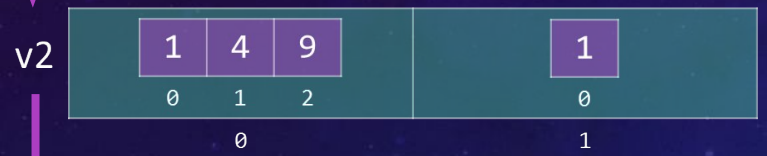
v2



Solution

- Write a short piece of code to change from each state to the next:


```
v2[0] = v2[2];
v2.pop_back();
v2[1].pop_back();
```

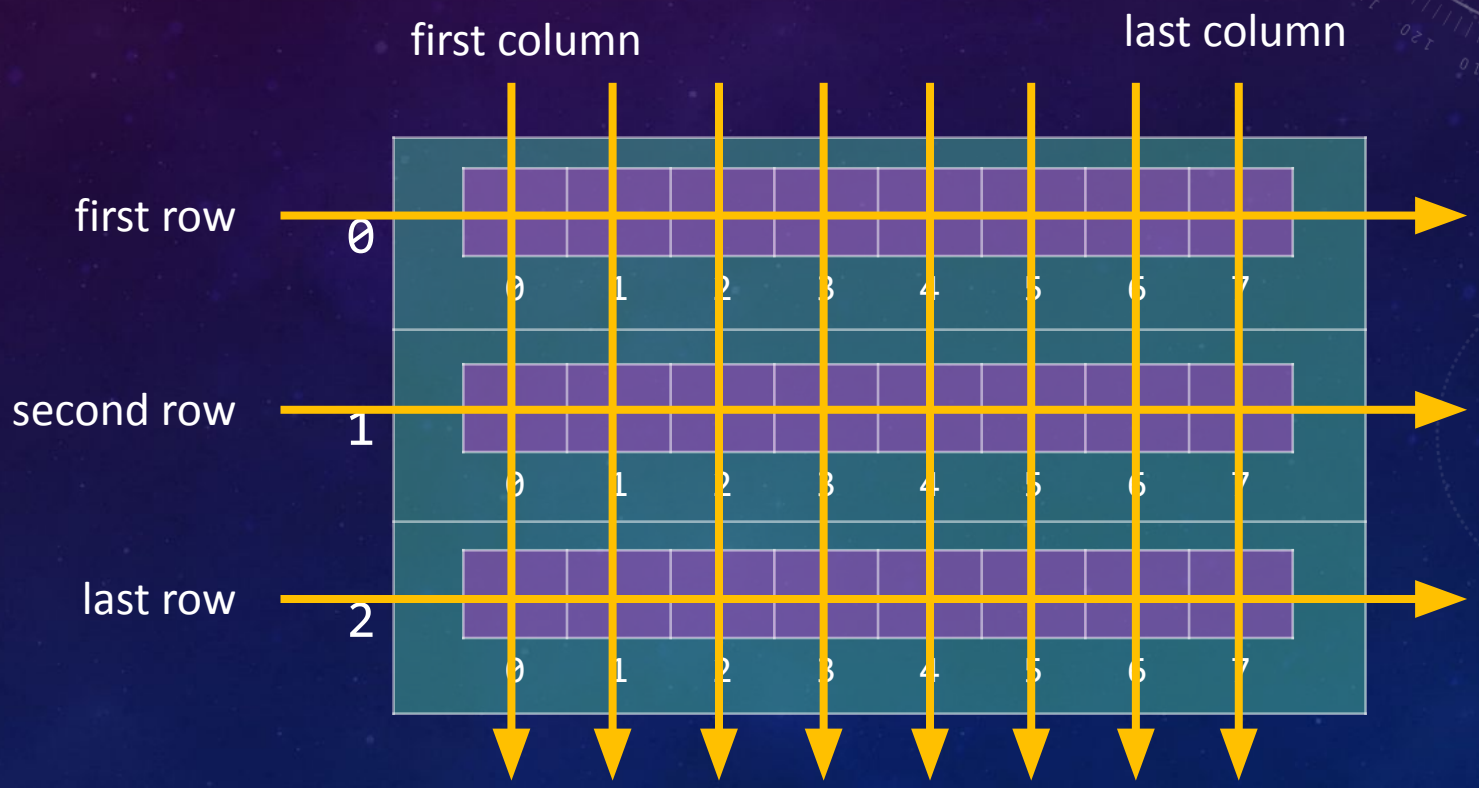



```
v2[0] = v2[1];
v2[1].push_back(4);
vector<int> temp(2,0);
v2.push_back(temp);
```

USES FOR VECTORS OF VECTORS

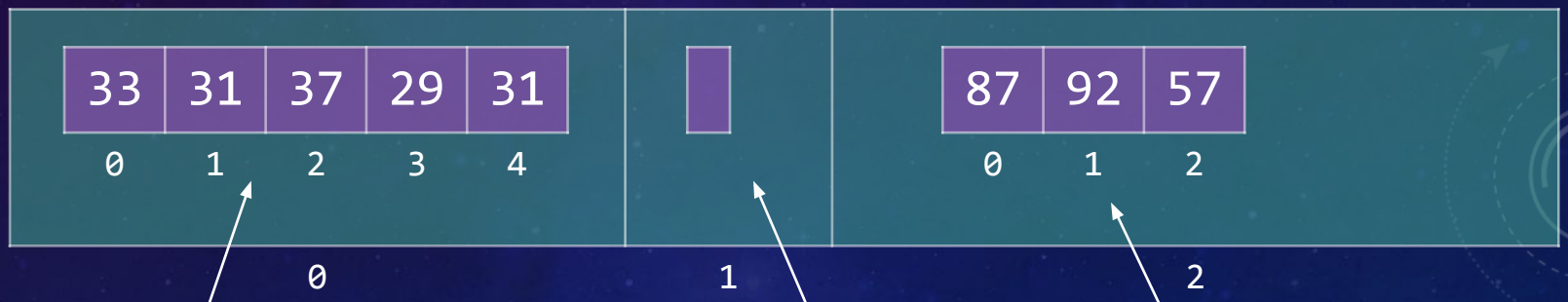
Simulating a Matrix

- Each vector represents a row.
- Elements within represent different columns.



Representing Multiple Sequences of Data

- Vectors of Vectors do not have to be “rectangular” like matrices
- Example: Measuring network latency.
  - We "ping" several different servers by sending 5 packets of information to each. We measure the time taken to receive a response for each packet.



Average the contents of this vector to find the average latency to this server.

This server couldn’t be reached.

This connection was slower and some packets were lost.

ANALYZING DATA IN A “MATRIX”

Write a Program: Analyze Data!

- Let's analyze a set of data in the file `testData.txt`

```
numRows 5
numCols 7

1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31 32 33 34 35
```

- Calculate:
  - sum of each row
  - sum of each column
  - sum of all data points

Analyzing data: Top-Down Design

- High level program design:
  - load the test data
    - open the test data file
    - if it did <u>not</u> open correctly, display error message and end program
    - load in the test data and store in a vector of vectors
  - analyze data
    - calculate sum of each row
    - calculate sum of each column
    - calculate total sum
  - print results

abstract into function `loadData`

abstract into function `sumOfRows`

abstract into function `sumOfCols`

abstract into function `sum`

Recall: Checking for Errors Opening a File

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main() {
  ifstream fin("words.in");

  if( !fin.is_open() ) {
    cout << "Error opening file! ";
    return 1; // Leave main early
  }

  string word; // will hold input
  for(int x = 0; x < 3; ++x) {
    fin >> word;
    cout << "Word " << x << ": " << word << endl;
  }

  fin.close();
}
```

include this in our `main` function

The return value for `main` can be used as the **exit code** for the program. A nonzero value indicates an error.

Pseudocode: main function

```
int main() {
  open the test data file

  if it did not open correctly
    display error message and end program

  make vector of vectors with correct # of empty elements
  loadData()

  make a vector to hold sums of rows
  sumOfRows()

  make a vector to hold sums of columns
  sumOfCols()

  store sum() in a variable

  print out the results

}
```

the `testData.txt` file tells us how many rows and columns there are in the data, so use the "make space then fill" pattern for reading data into vector of vectors

ANALYZING DATA: HELPER FUNCTIONS

Pseudocode: `loadData` helper function

inspiration: the `loadRovers` function from Ch 18

```
void loadData(vector of vectors, input stream from file) {

  get number of rows from vector of vectors
  get number of columns from vector of vectors

  make a temporary variable to store numbers in from >> operator

  loop on outer vector
    loop on inner vector
      read a number from input stream
      store that number in the correct element of the vec of vecs
    end of loop on inner vector
  end of loop on outer vector

}
```

you implement this!

Pseudocode: sumOfRows helper function

inspiration: the “using an accumulator” pattern from Ch 16

```
void sumOfRows(vector of vectors, vector) {
```

this is the “matrix” of data; how should we pass it?

this is the vector that will store the sum of each row; how should we pass it?

you implement this!

```
 traverse the outer vector to go through each row one by one
   traverse the inner vector
     use the "accumulator" pattern to find the sum of each row
  end of loop on inner vector
 end of loop on outer vector

}
```

Pseudocode: sumOfCols helper function

inspiration: the sumOfRows function

```
void sumOfCols(vector of vectors, vector) {

  traverse an inner vector to go through each column one by one
    traverse the outer vector
      use the "accumulator" pattern to find the sum of each column
    end of loop on outer vector
  end of loop on inner vector

}
```

this is the "matrix" of data; how should we pass it?

this is the vector that will store the sum of each column; how should we pass it?

all the inner vectors have the same length for this "matrix" so it doesn't matter which inner vector you use here

you implement this!

Pseudocode: sum helper function

inspiration: the sumOfRows function

```
int sum(vector of vectors) {

  traverse the outer vector to go through each row one by one
    traverse the inner vector
      use the "accumulator" pattern to find the total sum
   end of loop on inner vector
  end of loop on outer vector

  return sum of all elements

}
```

switch to an int return

this is the “matrix” of data; how should we pass it?

you implement this!