CSCI 2132: Software Development

Arrays in C

Norbert Zeh

Faculty of Computer Science
Dalhousie University

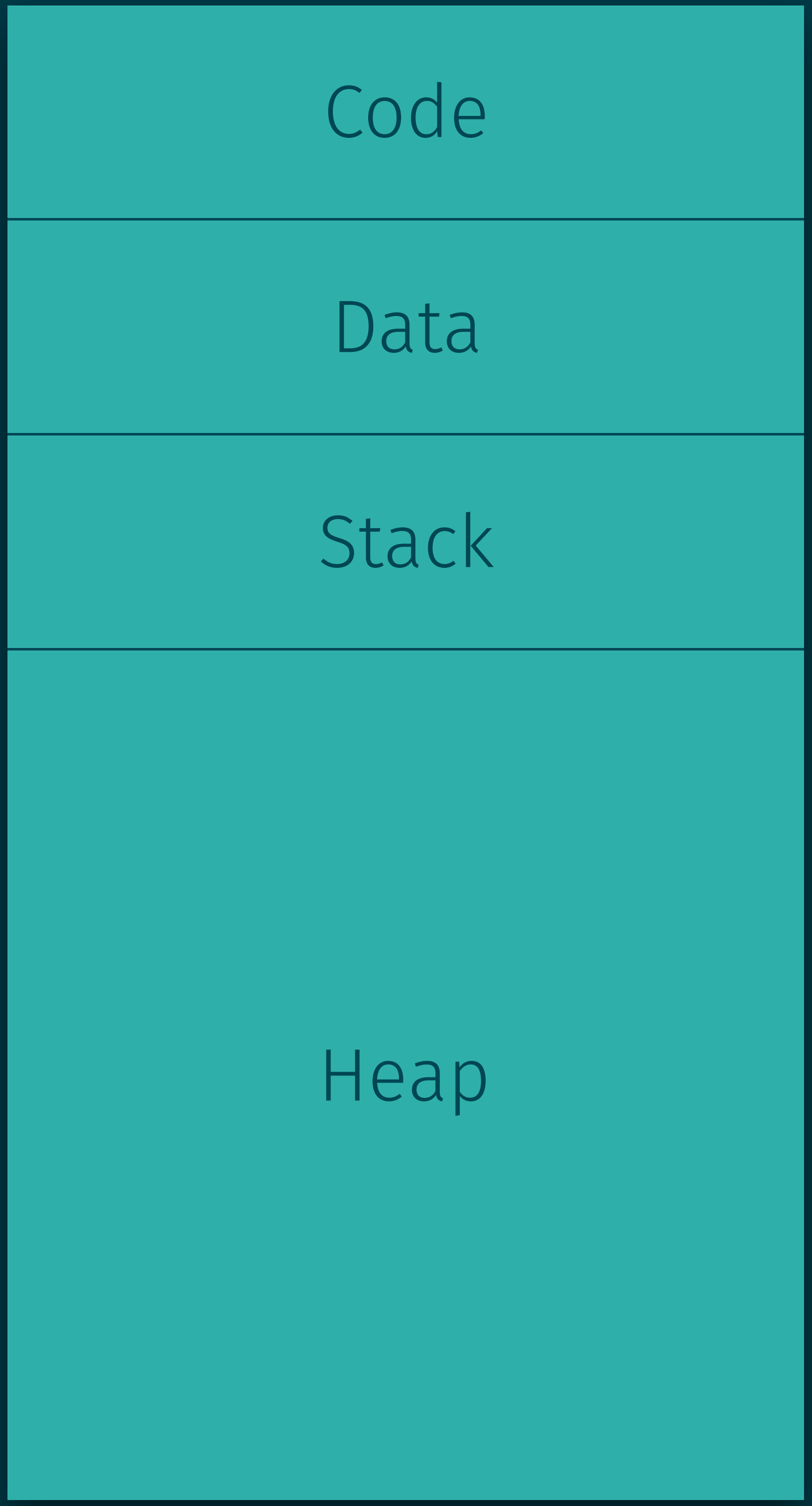
Winter 2019

Arrays vs Scalar Types

- Values of a scalar types (`int`, `float`, `char`, …) are single elements

- Aggregate (also compound or composite) types:
  - Composed of multiple elements
  - In C: arrays and structs

A Process's Memory Space



Code

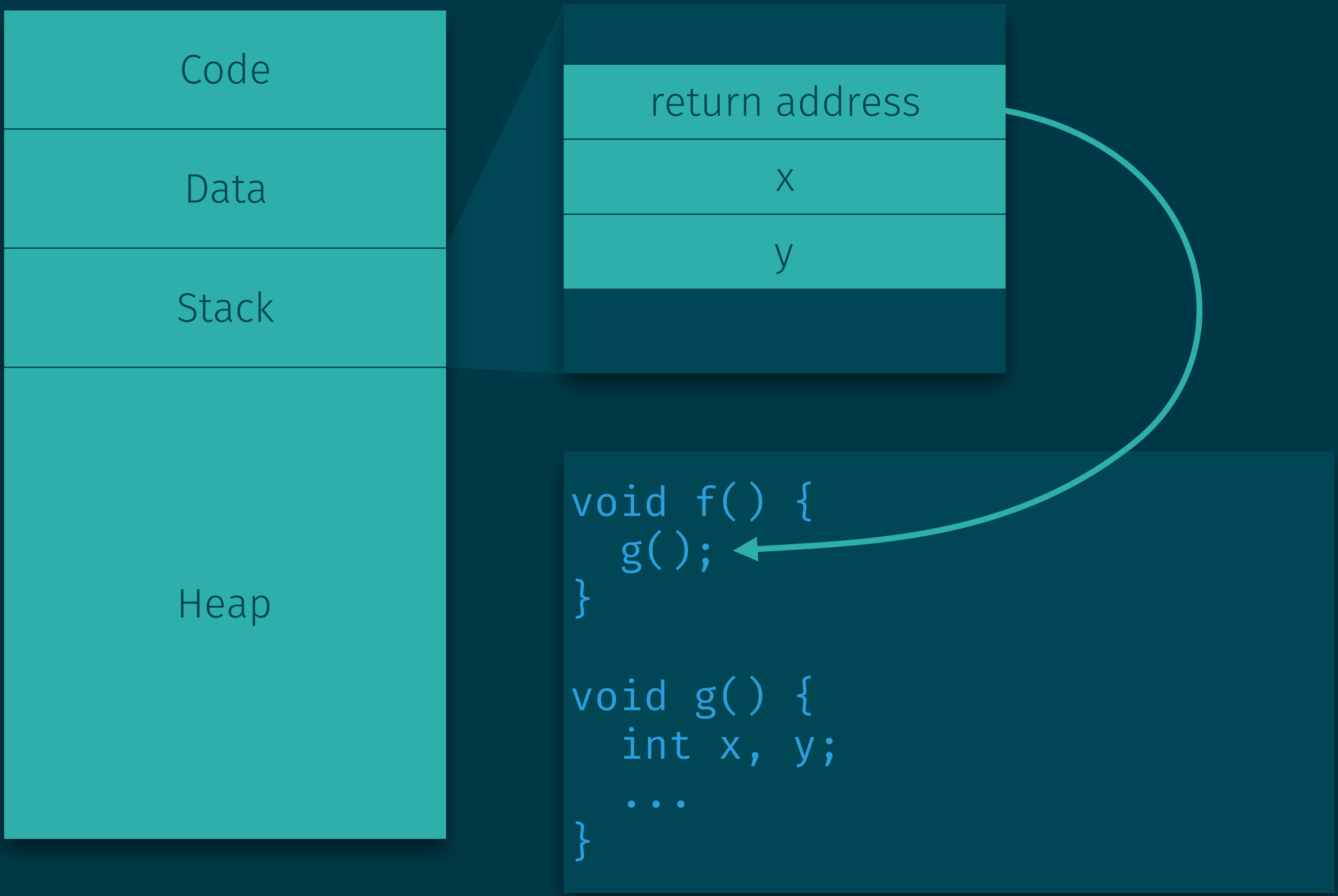
Data

Stack

Heap

- The code to be executed
- Static data (string constants, …)
- Local variables of functions
- Dynamically allocated objects
  - Usually outlive the lifetime of a function call
  - Allocated using `new`, garbage collected in Java
  - Allocated using `malloc()`, freed using `free()` in C

A Process's Memory Space



Code

Data

Stack

Heap

return address

x

y

```
void f() {
   g();
}

void g() {
   int x, y;
   ...
}
```

Allocation of C Arrays

- **C** arrays allocated on stack
- **Java:** Arrays allocated on heap
- Java 6 introduced “escape analysis”
  - Compiler analyzes whether a Java array can be allocated no a stack
  - More efficient if this is possible

Array Length

- Array length is often defined as a macro (constant)

Example:

```
#define N 40
int a[N];
```

  - Array elements accessed as `a[0]`, …, `a[N-1]`

- Why the constant?
  - Size of the array can be changed in one spot

Bounds Checks of Arrays

- Many higher-level languages perform bounds checks:
  Is the provided index in the index range of the array
  (between 0 and n–1)

- C does not do that!

Reason:

Bounds Checks of Arrays

- Many higher-level languages perform bounds checks: Is the provided index in the index range of the array (between 0 and n–1)
- C does not do that!

Reason: Efficiency

Consequences:

Bounds Checks of Arrays

- Many higher-level languages perform bounds checks: Is the provided index in the index range of the array (between 0 and n–1)

- C does not do that!

Reason: Efficiency

Consequences:

- Code crashes (best-case scenario)
- Strange behaviour
- Security issues
- …

Array Initialization

Example:

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

Size can be determined implicitly:

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

If initializer is shorter, the other elements are set to 0:

```
int a[10] = {1, 2, 3};
```

Easy way to set all array elements to 0:

```
int a[10] = {};
```

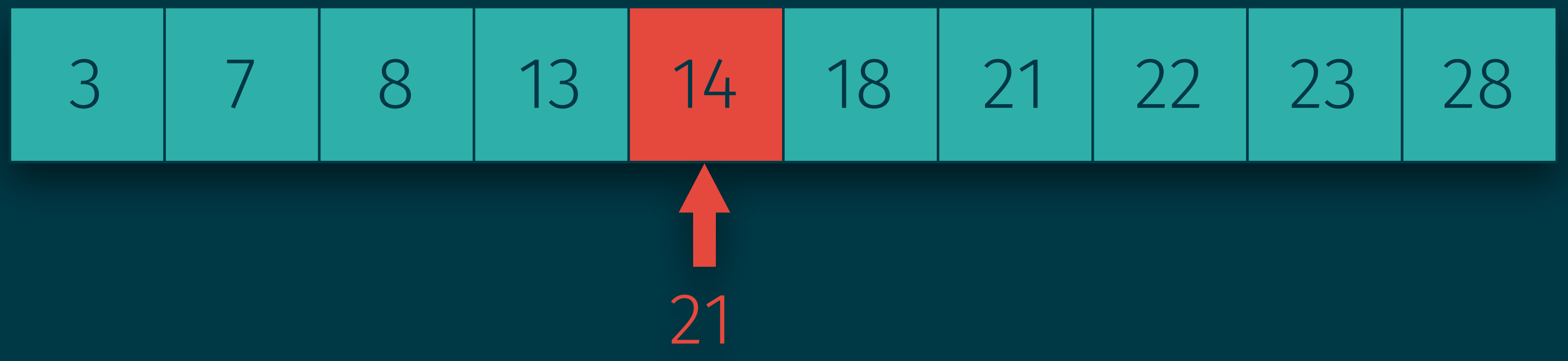
Review: Binary Search

- Method to search for an item x in a sorted array
- In each step, check the middle element y
  - Stop if $x == y$
  - Continue on left half if $x < y$
  - Continue on right half if $x > y$

3	7	8	13	14	18	21	22	23	28

Review: Binary Search

- Method to search for an item x in a sorted array
- In each step, check the middle element y
  - Stop if $x == y$
  - Continue on left half if $x < y$
  - Continue on right half if $x > y$

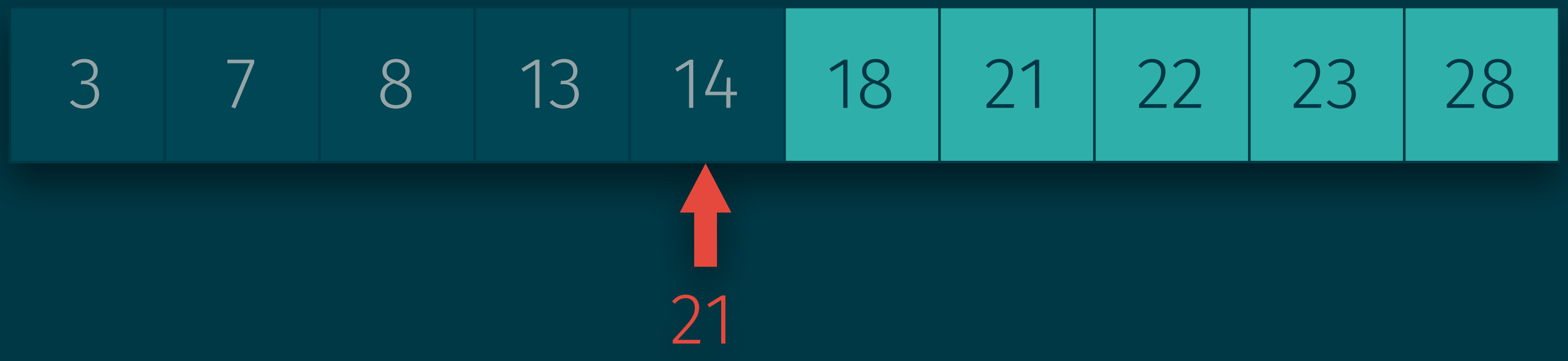


3	7	8	13	14	18	21	22	23	28

21

Review: Binary Search

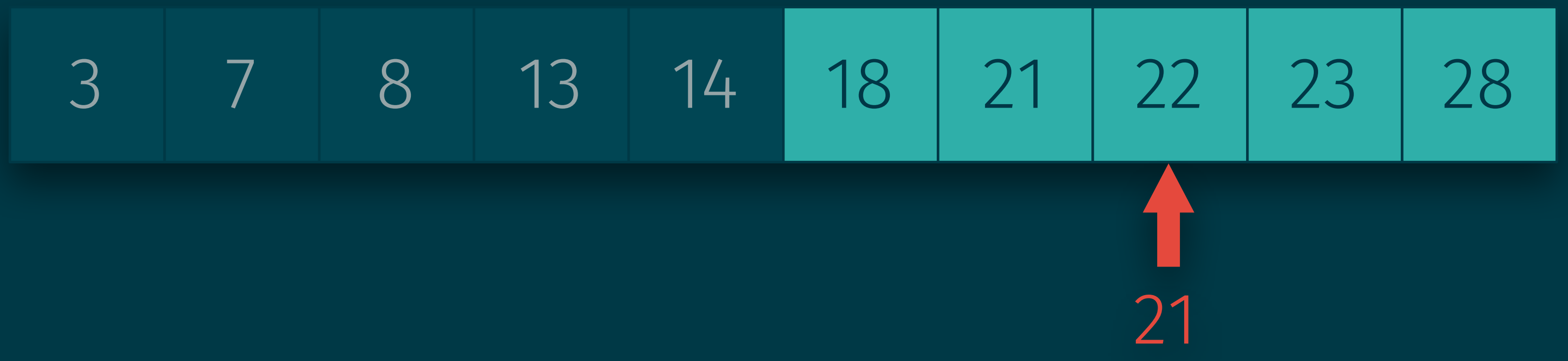
- Method to search for an item x in a sorted array
- In each step, check the middle element y
  - Stop if $x == y$
  - Continue on left half if $x < y$
  - Continue on right half if $x > y$



3	7	8	13	14	18	21	22	23	28

Review: Binary Search

- Method to search for an item x in a sorted array
- In each step, check the middle element y
    - Stop if $x == y$
    - Continue on left half if $x < y$
    - Continue on right half if $x > y$

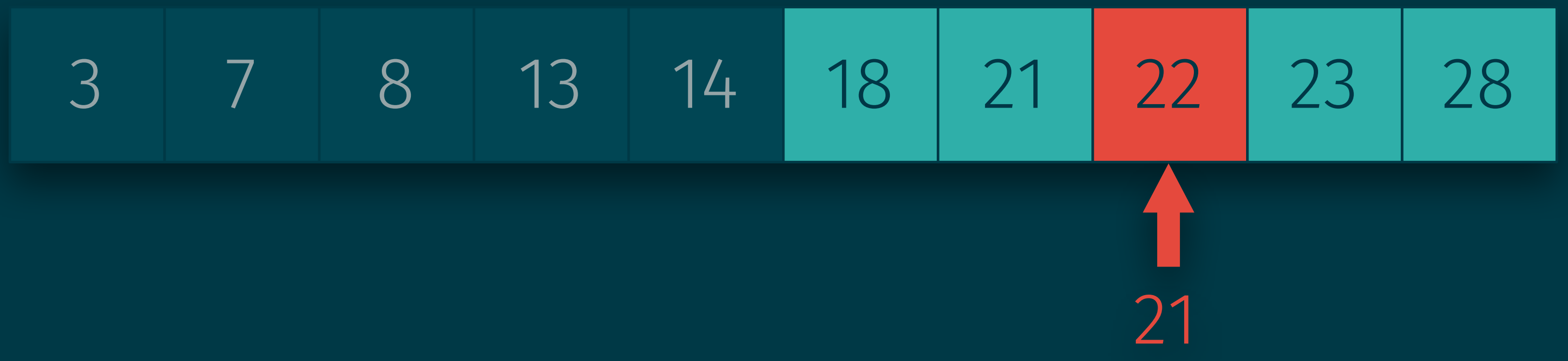


3	7	8	13	14	18	21	22	23	28

21

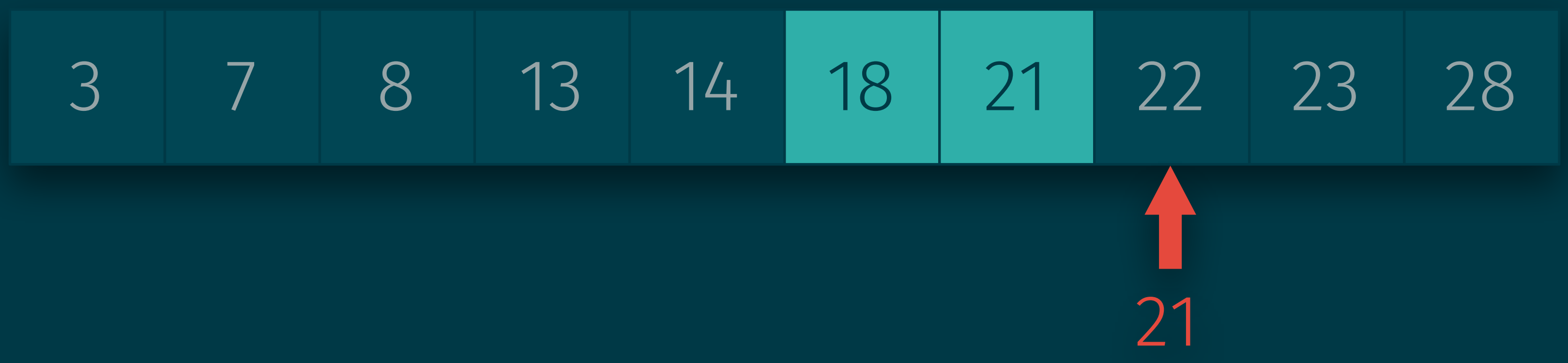
Review: Binary Search

- Method to search for an item x in a sorted array
- In each step, check the middle element y
  - Stop if $x == y$
  - Continue on left half if $x < y$
  - Continue on right half if $x > y$



Review: Binary Search

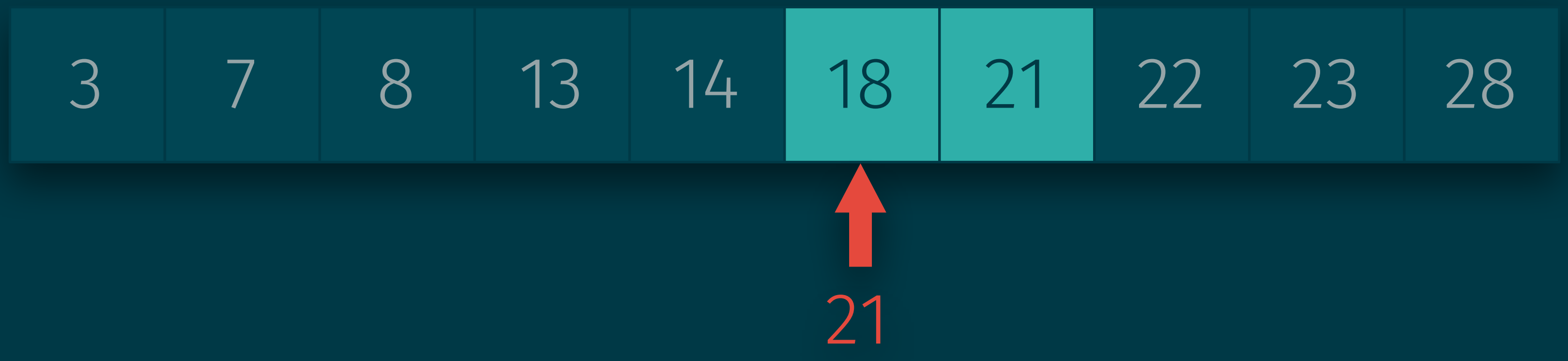
- Method to search for an item x in a sorted array
- In each step, check the middle element y
  - Stop if $x == y$
  - Continue on left half if $x < y$
  - Continue on right half if $x > y$



3	7	8	13	14	18	21	22	23	28

Review: Binary Search

- Method to search for an item x in a sorted array
- In each step, check the middle element y
  - Stop if $x == y$
  - Continue on left half if $x < y$
  - Continue on right half if $x > y$

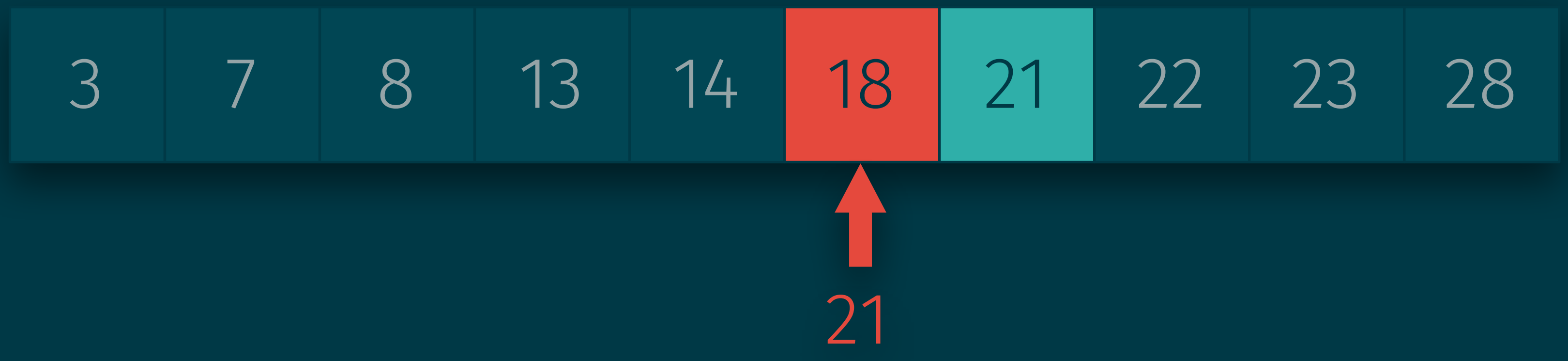


3	7	8	13	14	18	21	22	23	28

21

Review: Binary Search

- Method to search for an item x in a sorted array
- In each step, check the middle element y
  - Stop if $x == y$
  - Continue on left half if $x < y$
  - Continue on right half if $x > y$

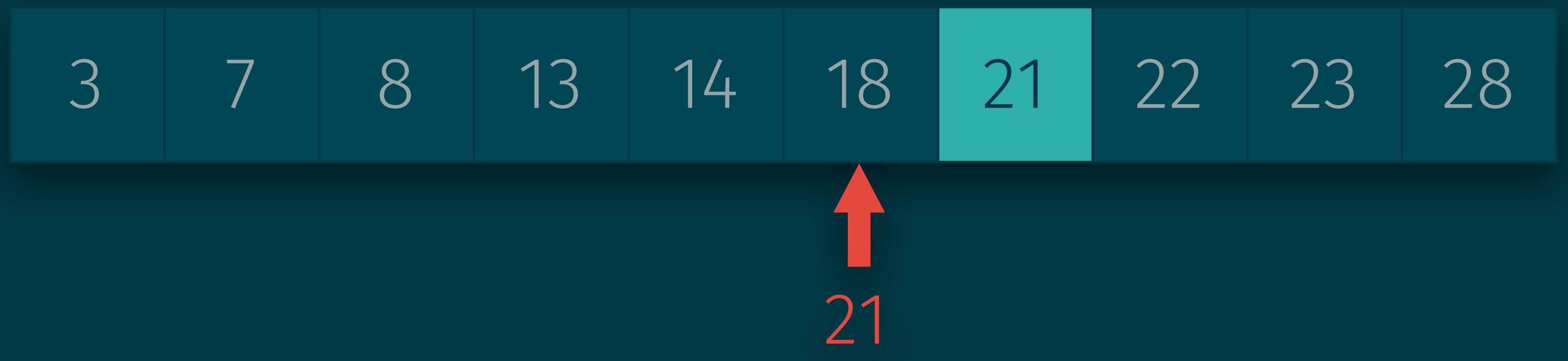


3	7	8	13	14	18	21	22	23	28

21

Review: Binary Search

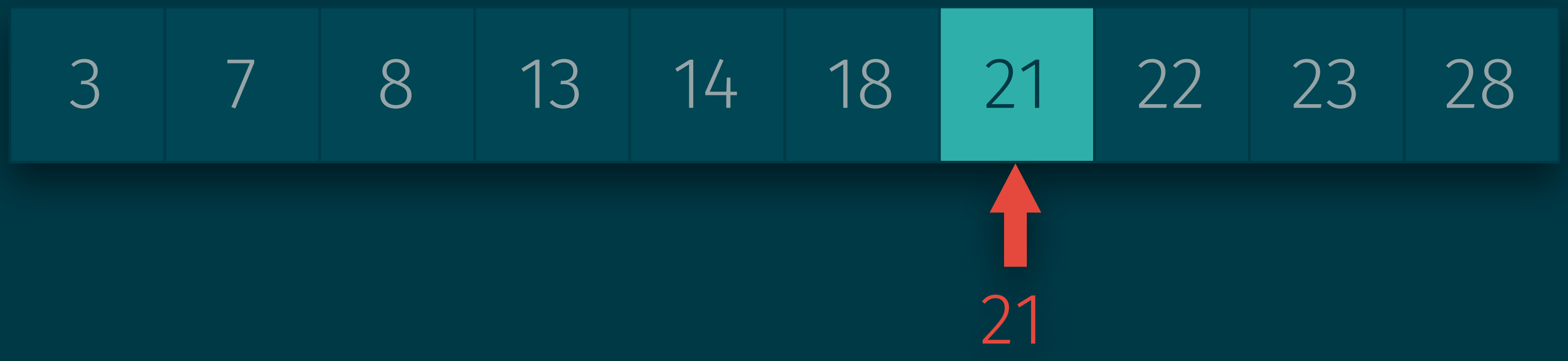
- Method to search for an item x in a sorted array
- In each step, check the middle element y
  - Stop if $x == y$
  - Continue on left half if $x < y$
  - Continue on right half if $x > y$



3	7	8	13	14	18	21	22	23	28

Review: Binary Search

- Method to search for an item x in a sorted array
- In each step, check the middle element y
  - Stop if $x == y$
  - Continue on left half if $x < y$
  - Continue on right half if $x > y$

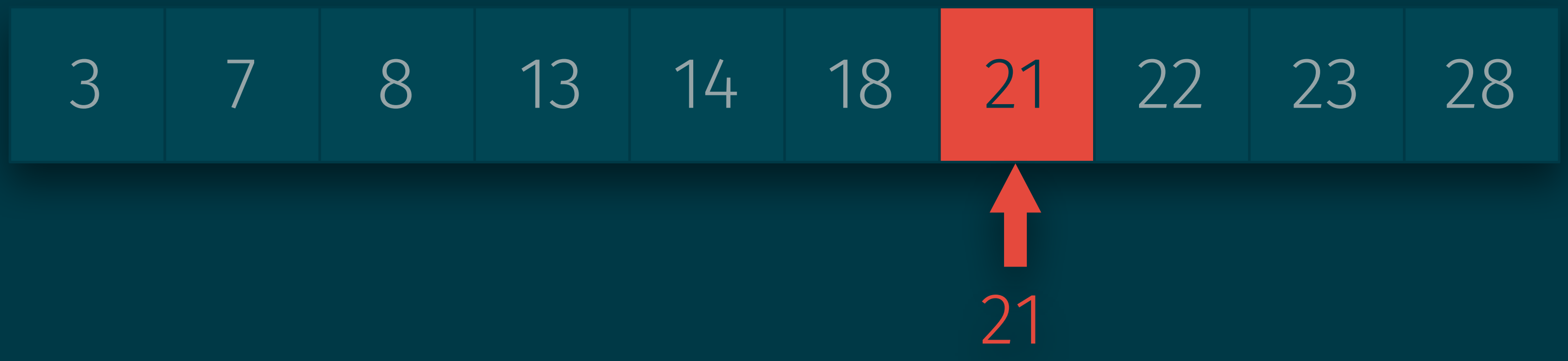


3	7	8	13	14	18	21	22	23	28

21

Review: Binary Search

- Method to search for an item x in a sorted array
- In each step, check the middle element y
  - Stop if $x == y$
  - Continue on left half if $x < y$
  - Continue on right half if $x > y$



3	7	8	13	14	18	21	22	23	28

Implementation of Binary Search

Write a program to:

- Enter 10 numbers in increasing order
- Enter a number to search for
- Report the position where it was found or that the element is not in the array.

Fill In the Blanks

```
#include <stdio.h>

#define LEN 10

int main() {
  int array[LEN], lower, upper, middle, key, i;
  printf(“Enter %d numbers in ascending order:\n”, LEN);
  for (i = 0; i < LEN; ++i)
    scanf(“%d”,          );
  printf(“Enter the number to be searched for: “);
  scanf(“%d”, &key);
```

Fill in the Blanks

```c
#include <stdio.h>

#define LEN 10

int main() {
  int array[LEN], lower, upper, middle, key, i;
  printf(“Enter %d numbers in ascending order:\n”, LEN);
  for (i = 0; i < LEN; ++i)
    scanf(“%d”, &array[i]);
  printf(“Enter the number to be searched for: “);
  scanf(“%d”, &key);
```

Fill in the Blanks

```
#include <stdio.h>

#define LEN 10

int main() {
  int array[LEN], lower, upper, middle, key, i;
  printf(“Enter %d numbers in ascending order:\n”, LEN);
  for (i = 0; i < LEN; ++i)
    scanf(“%d”, array + i);
  printf(“Enter the number to be searched for: “);
  scanf(“%d”, &key);
```

Fill in the Blanks

```
  lower = 0;
  upper = LEN;
  middle = (lower + upper) / 2;
  while (lower < upper) {
    if (key == array[middle]) {
      printf(“%d is the %dth number you entered.\n”,
             );
      return 0;
    } else if (key < array[middle]) {
      upper = ;
    } else {
      lower = ;
    }
    middle = (lower + upper) / 2;
  }
  printf(“Not found.\n”);
  return 0;
}
```

Fill in the Blanks

```
  lower = 0;
  upper = LEN;
  middle = (lower + upper) / 2;
  while (lower < upper) {
    if (key == array[middle]) {
      printf(“%d is the %dth number you entered.\n”,
             key, middle);
      return 0;
    } else if (key < array[middle]) {
      upper =       ;
    } else {
      lower =          ;
    }
    middle = (lower + upper) / 2;
  }
  printf(“Not found.\n”);
  return 0;
}
```

Fill in the Blanks

```
  lower = 0;
  upper = LEN;
  middle = (lower + upper) / 2;
  while (lower < upper) {
    if (key == array[middle]) {
      printf(“%d is the %dth number you entered.\n”,
             key, middle);
      return 0;
    } else if (key < array[middle]) {
      upper = middle;
    } else {
      lower =           ;
    }
    middle = (lower + upper) / 2;
  }
  printf(“Not found.\n”);
  return 0;
}
```

Fill in the Blanks

```
  lower = 0;
  upper = LEN;
  middle = (lower + upper) / 2;
  while (lower < upper) {
    if (key == array[middle]) {
      printf(“%d is the %dth number you entered.\n”,
             key, middle);
      return 0;
    } else if (key < array[middle]) {
      upper = middle;
    } else {
      lower = middle + 1;
    }
    middle = (lower + upper) / 2;
  }
  printf(“Not found.\n”);
  return 0;
}
```

Multidimensional Arrays

- C allows multidimensional arrays:
  - `int m[5][9]` defines a 5 × 9 array
  - Elements are `m[0][0]` to `m[4][8]`
  - Which is the element in row 1 and column 4?

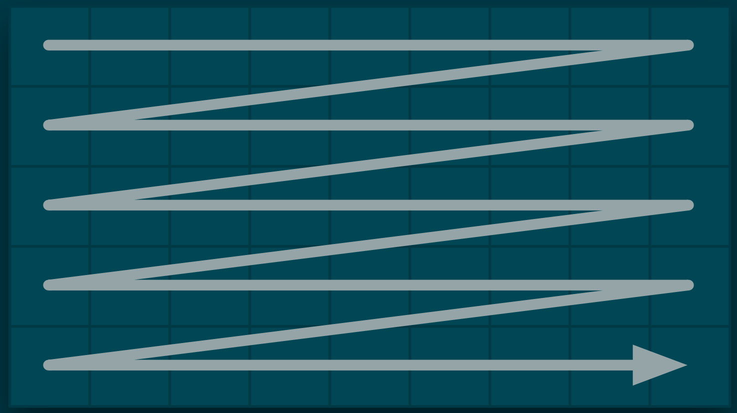
Multidimensional Arrays

- C allows multidimensional arrays:
  - `int m[5][9]` defines a 5 × 9 array
  - Elements are `m[0][0]` to `m[4][8]`
  - Which is the element in row 1 and column 4? `m[1][4]`

Multidimensional Arrays

- C allows multidimensional arrays:
  - `int m[5][9]` defines a 5 × 9 array
  - Elements are `m[0][0]` to `m[4][8]`
  - Which is the element in row 1 and column 4? `m[1][4]`

- Arrays stored in row-major order



Initialization of Multidimensional Arrays

Initializing multi-dimensional arrays:

```
int t[3][3] = {{1, 0, 0},
               {0, 1, 0},
               {0, 0, 1}};
```

Inner parentheses can be omitted:

```
int t[3][3] = {1, 0, 0, 0, 1, 0, 0, 0, 1};
```

Set all entries to 0 (as for one-dimensional arrays):

```
int t[3][3] = {};
```

Variable-Length Arrays (C99–)

C99 introduced variable-length arrays:

- Length not known at compile time
- Length is not dynamic as in Java or C++ vectors

Example:

```
int len, i;
printf(“Enter the number of integers: “);
scanf(“%d”, &len);
int array[len];
printf(“Enter %d numbers: “, len);
for (i = 0; i < len; ++i)
  scanf(“%d”, &array[i]);
```

Variable-Length Arrays (C99–)

Exercise: Rewrite the binary search program using variable-length arrays and ask the user to enter the array length first.

- Variable-length arrays can be multi-dimensional but cannot have initializers.

Exercise: Sudoku (Checker)

Sudoku: Fill a 9 × 9 square with numbers 1..9 so that

- Each row is a permutation of (1, ..., 9)
- Each column is a permutation of (1, ..., 9)
- Each 3 × 3 square is a permutation (1, ..., 9)

4	8	7	3	2	9	6	1	5
2	3	5	6	4	1	7	8	9
1	6	9	5	8	7	4	2	3
6	9	1	7	3	8	2	5	4
5	4	2	9	1	6	8	3	7
8	7	3	2	5	4	1	9	6
9	5	8	4	6	2	3	7	1
7	2	4	1	9	3	5	6	8
3	1	6	8	7	5	9	4	2

Implementation of a Sudoku Checker

(Reading the Input)

```c
#include <stdio.h>

int main() {
  int square[9][9], occurs[9], row, col, block, index;
  printf(“Enter the square:\n”);
  for (row = 0; row < 9; ++row) {
    printf(“Row %d: “, row+1);
    for (col = 0; col < 9; ++col) {
      scanf(“%d”, &square[row][col]);
      if (square[row][col] < 1 || square[row][col] > 9) {
        printf(“Error: element (%d, %d) out of range.\n”,
               row, col);
        return 1;
      }
    }
```

Implementation of a Sudoku Checker

(Checking the Rows)

```
for (row = 0; row < 9; ++row) {
  for (col = 0; col < 9; ++col)
    occurs[col] = 0;
  for (col = 0; col < 9; ++col)
    if (occurs[square[row][col]-1] > 0) {
      printf(“This is not a latin square.\n”);
      return 1;
    } else {
      occurs[square[row][col]-1] = 1;
    }
}
```

Implementation of a Sudoku Checker

(Checking the Columns)

```
for (col = 0; col < 9; ++col) {
  for (row = 0; row < 9; ++row)
    occurs[row] = 0;
  for (row = 0; row < 9; ++row)
    if (occurs[square[row][col]-1] > 0) {
      printf(“This is not a latin square.\n”);
      return 1;
    } else {
      occurs[square[row][col]-1] = 1;
    }
}
```

Implementation of a Sudoku Checker

(Checking the 3 × 3s)

```
 for (block = 0; block < 9; ++block) {
  for (index = 0; index < 9; ++index)
   occurs[index] = 0;
  for (index = 0; index < 9; ++index) {
   row = 3*(block/3) + (index/3);
   col = 3*(block%3) + (index%3);
   if (occurs[square[row][col]-1] > 0) {
    printf(“This is not a latin square.\n”);
    return 1;
   } else {
    occurs[square[row][col]-1] = 1;
   }
  }
 }
 return 0;
}
```