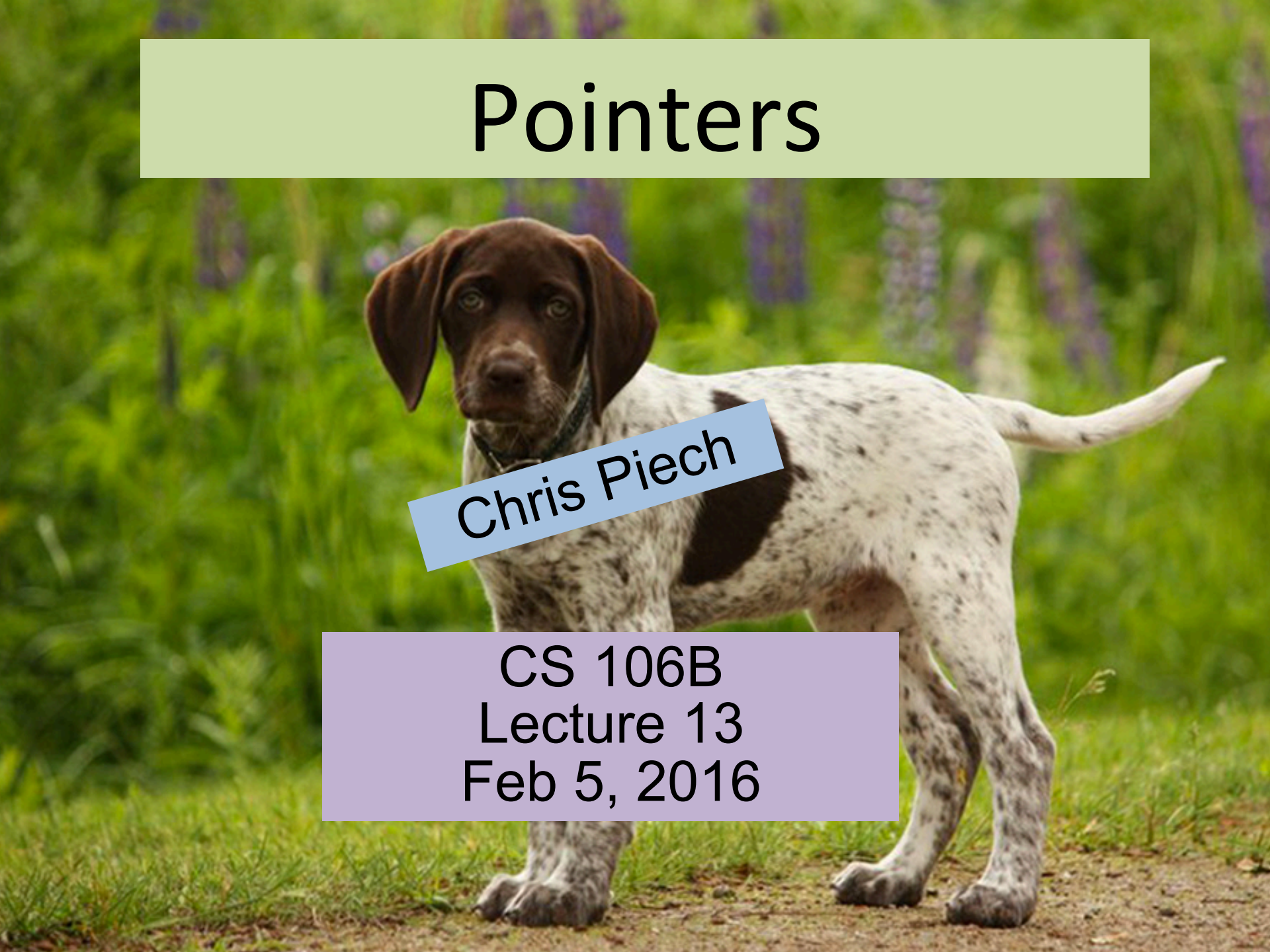
Pointers



Chris Piech

CS 106B
Lecture 13
Feb 5, 2016

Socrative.com

Room: **106BWIN16**

Announcement: Midterm

Last name A-HAN: Hewlett 200
Last name HAP-MC: Hewlett 201
Last name ME-Z: Braun Auditorium

Concepts: Functions, Collections (Stacks, Queues, Vector, Grid, Map, Set), Recursion, Recursive Backtracking

Eg everything up to Monday and in the assignments you have done.

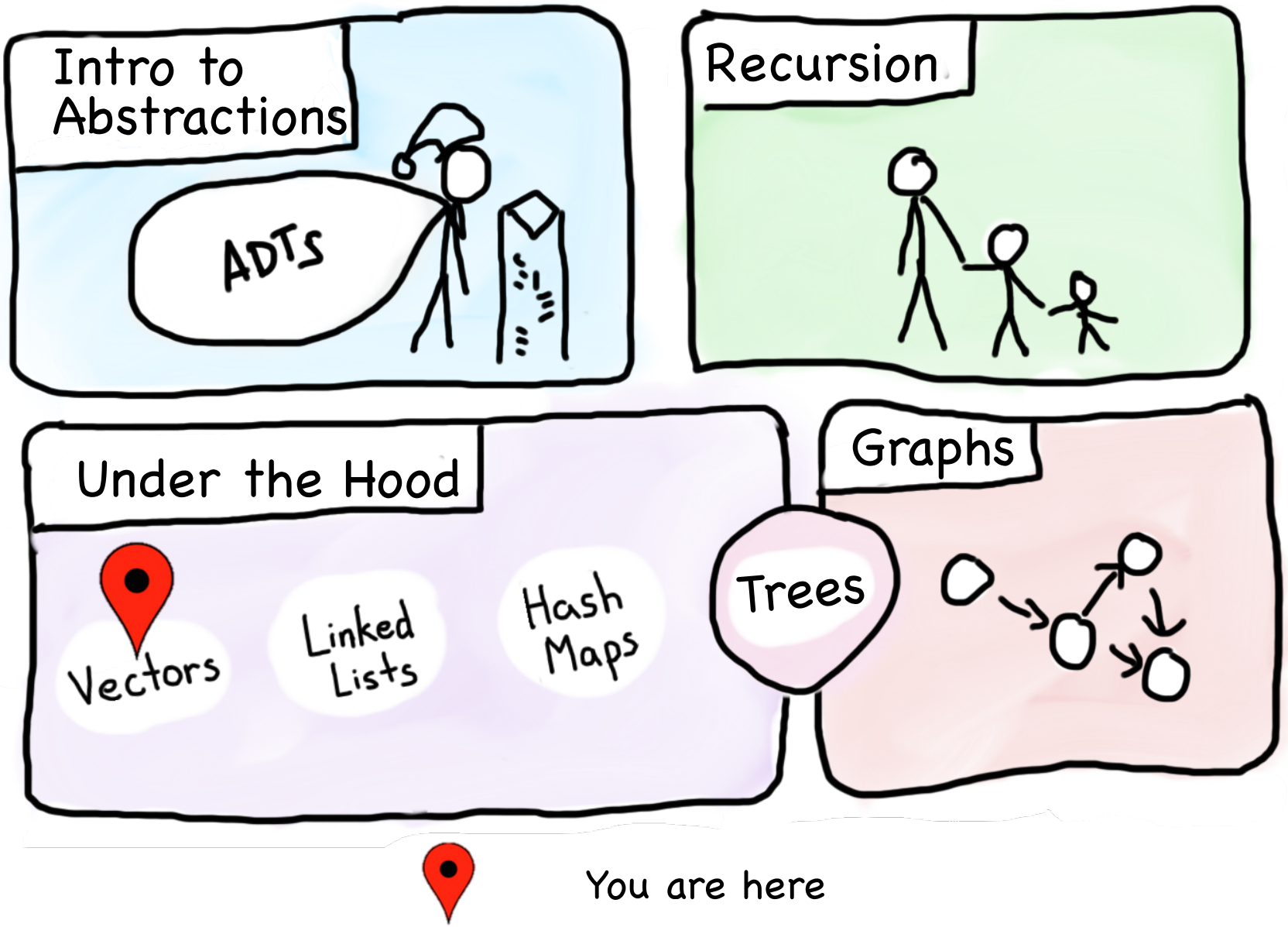
Practice Midterm Sitting

Sunday 10AM-Noon
in Bishop Auditorium

Midterm Documents

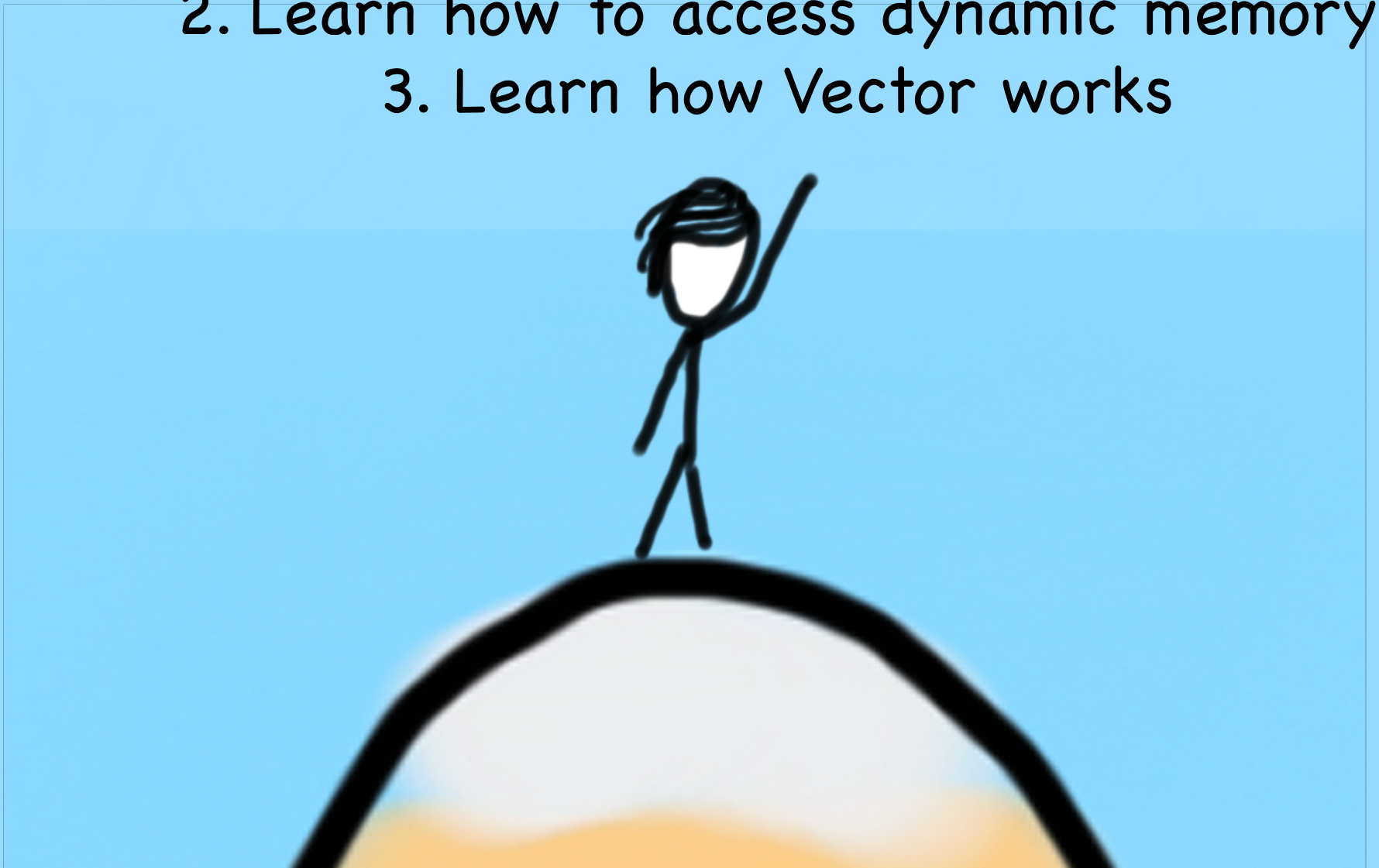
Today: Midterm Strategies
Saturday: Chris Exam #1
Saturday: Concept Handouts
Sunday: Chris Exam #2

Course Syllabus

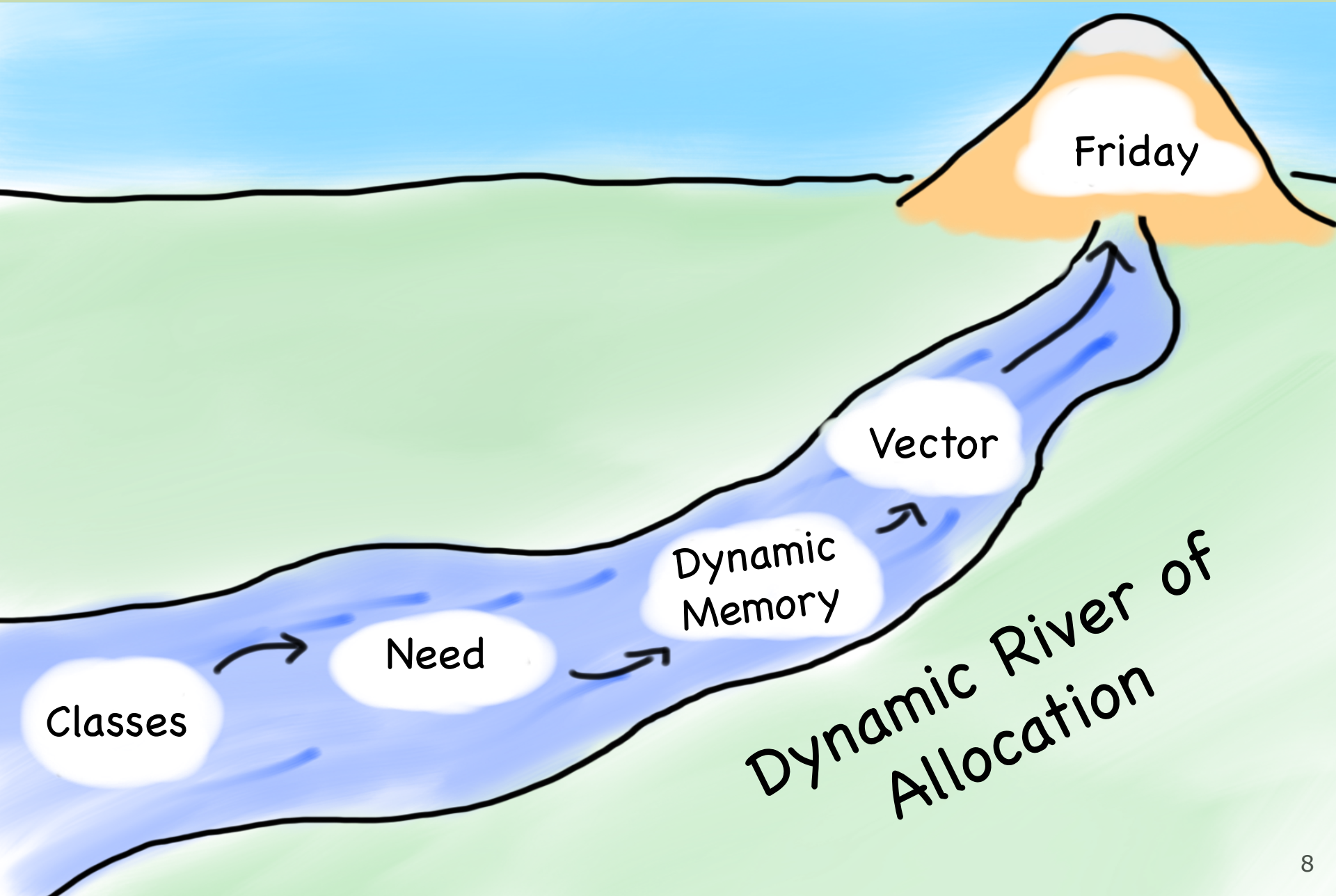


Today's Goals

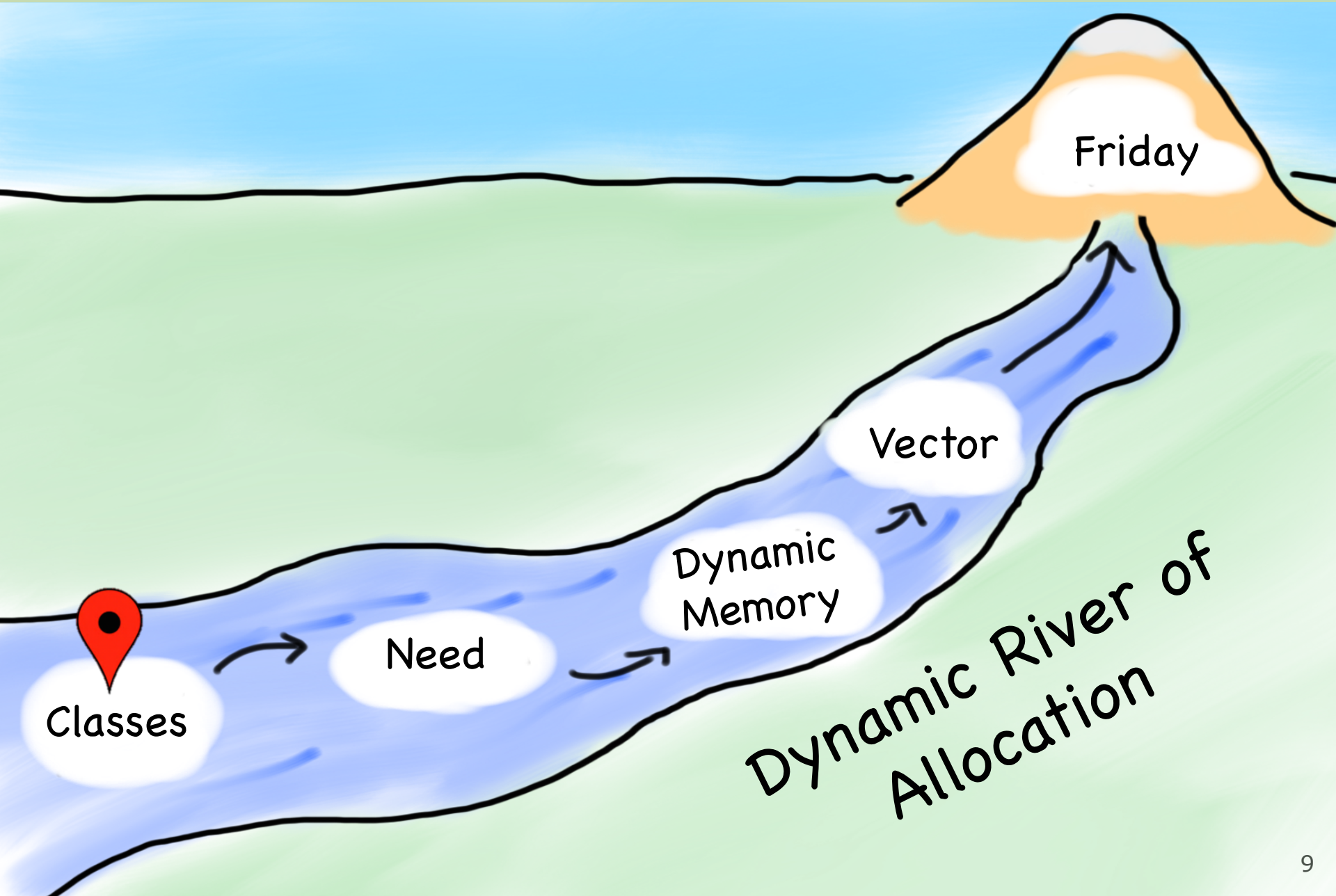
1. Learn how to dynamically create
2. Learn how to access dynamic memory
3. Learn how Vector works



Today's Goals



Today's Goals



Some *large* programs are in C++


WORLD
OF
WARCRAFT

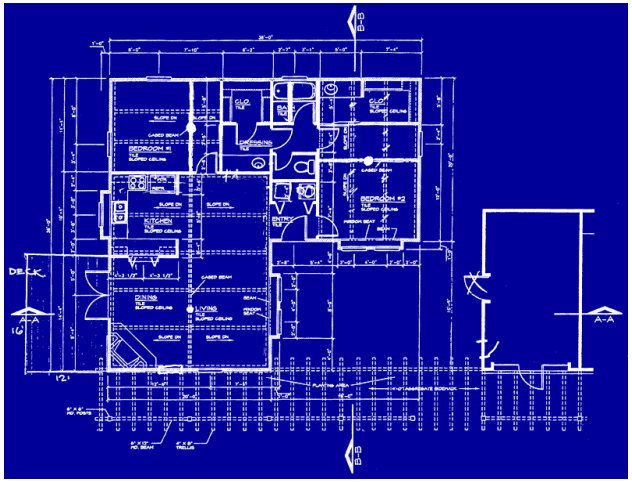

W


Google

Classes

class: A template for a new type of variable.

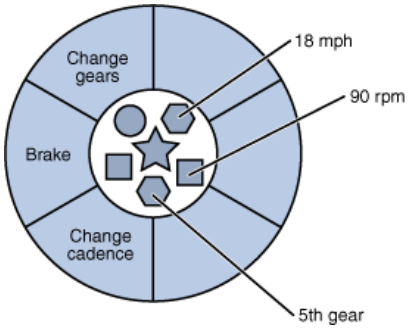
A blueprint is a helpful analogy



Elements of a class

member variables: State inside each object.

- Also called "instance variables" or "fields"
- Declared as `private`
- Each object created has a copy of each field.



member functions: Behavior that executes inside each object.

- Also called "methods"
- Each object created has a copy of each method.
- The method can interact with the data inside that object.

constructor: Initializes new objects as they are created.

- Sets the initial state of each `new` object.
- Often accepts parameters for the initial state of the fields.

Date Class

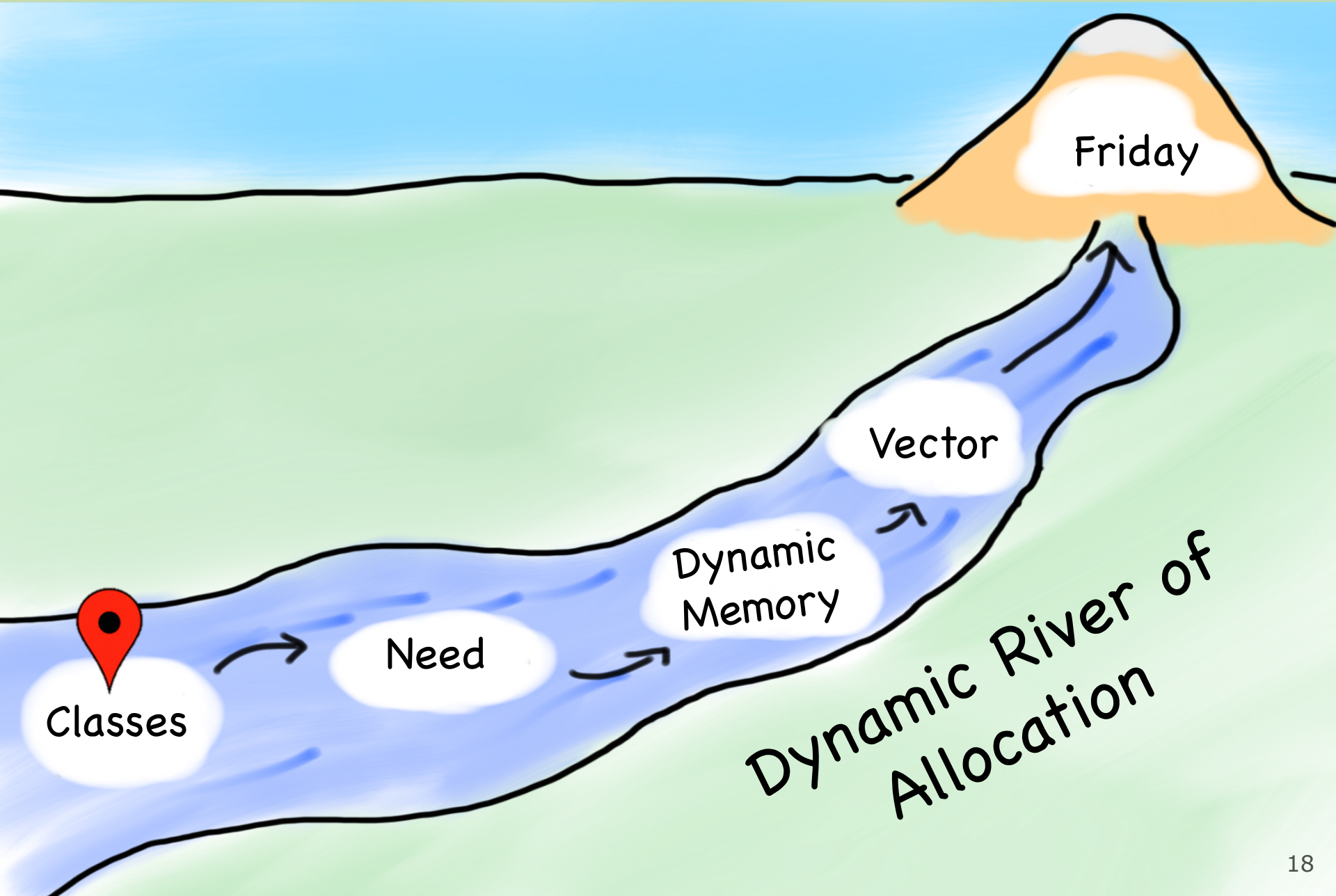
```
int main() {
    Date today(3,2,2016);
    Date springBreak(19,3,2016);

    cout << "spring break: " << springBreak << endl;

    cout << "days until spring break: ";
    cout << today.daysUntil(springBreak) << endl;

    today.incrementDay();

    cout << "days until spring break: ";
    cout << today.daysUntil(springBreak) << endl;

    return 0;
}
```

But…

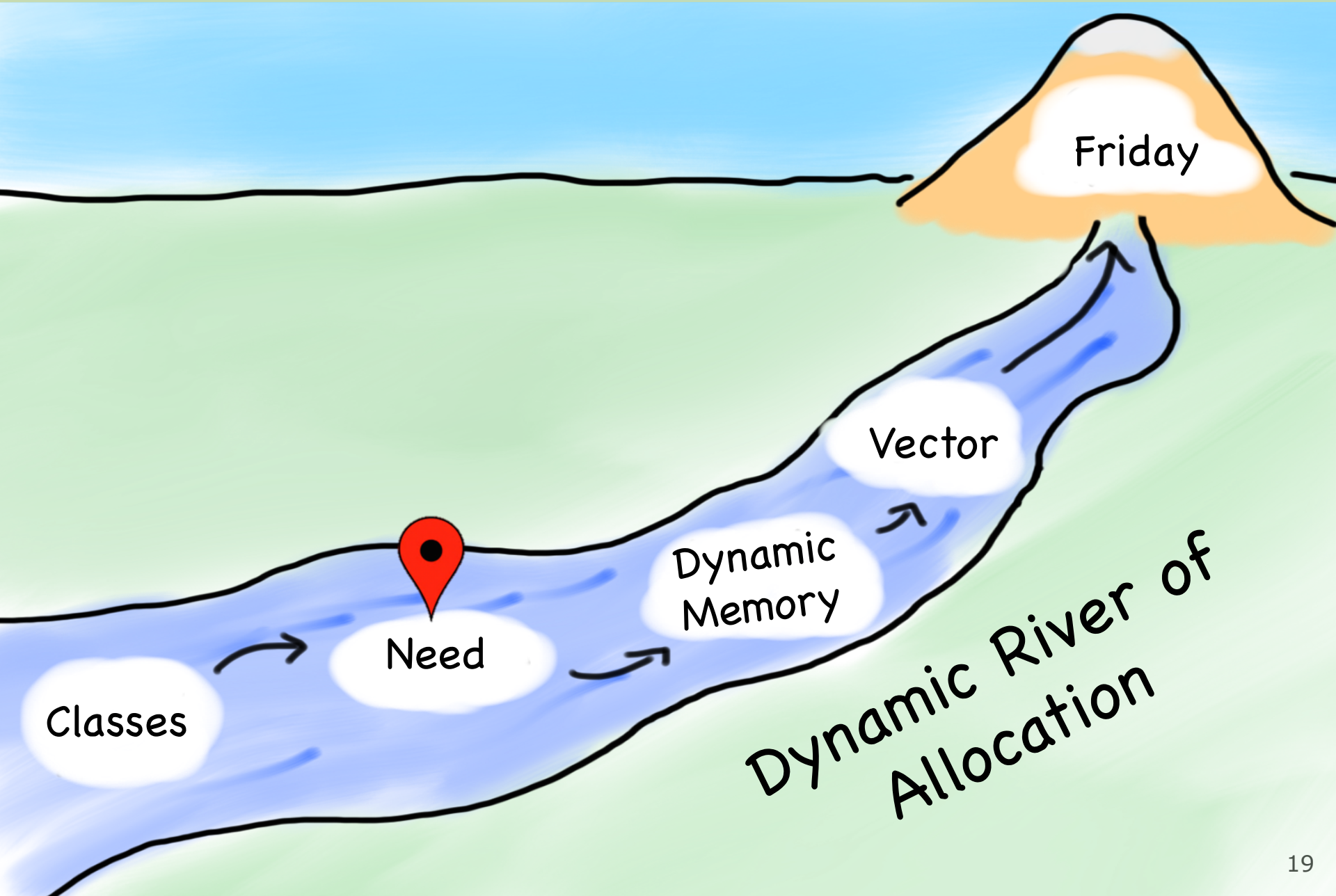
C++ has no Dates ☹

You know what to do

Today's Goals



Today's Goals



Lets Write Vector

A class declaration

```
class ClassName {                       // in ClassName.h
public:
    ClassName(parameters);              // constructor

    returnType name(parameters); // member functions
    returnType name(parameters); // (behavior inside
    returnType name(parameters); //  each object)

private:
    type name;      // member variables
    type name;      // (data inside each object)
};
```

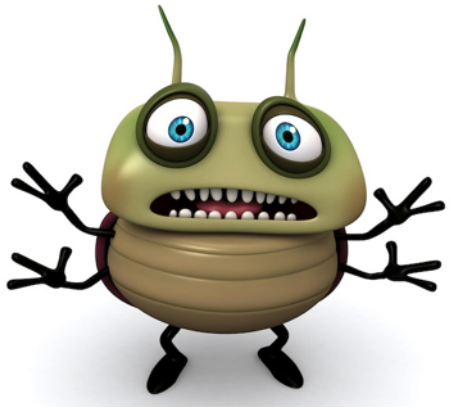
IMPORTANT: *must* put a semicolon at end of class declaration (argh)

VectorInt

```
class VectorInt {          // in VectorInt.h
public:
    VectorInt();           // constructor

    void add(int value); // append a value to the end
    int get(int index);  // return the value at index

private:
    type name;      // member variables
    type name;      // (data inside each object)
};
```

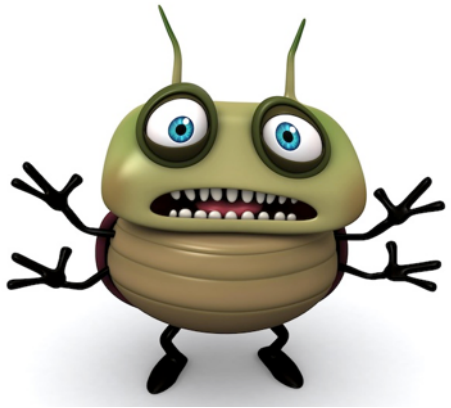
Buggy VectorInt the First

```
class VectorInt {          // in VectorInt.h
public:
    VectorInt();           // constructor

    void add(int value); // append a value to the end
    int get(int index);  // return the value at index

private:
    int value0;      // member variables
    int value1;      // (data inside each object)
};
```



Buggy VectorInt the Second

```
class VectorInt {          // in VectorInt.h
public:
    VectorInt();           // constructor

    void add(int value); // append a value to the end
    int get(int index);  // return the value at index

private:
    Vector<int> values;  // (data inside each object)
};
```



Problems with Stack Variables

Variables have to be known at compile time. Not runtime.

Problems with Stack Variables

Variables have to be known at compile time. Not runtime.

Persistence is out of our control.

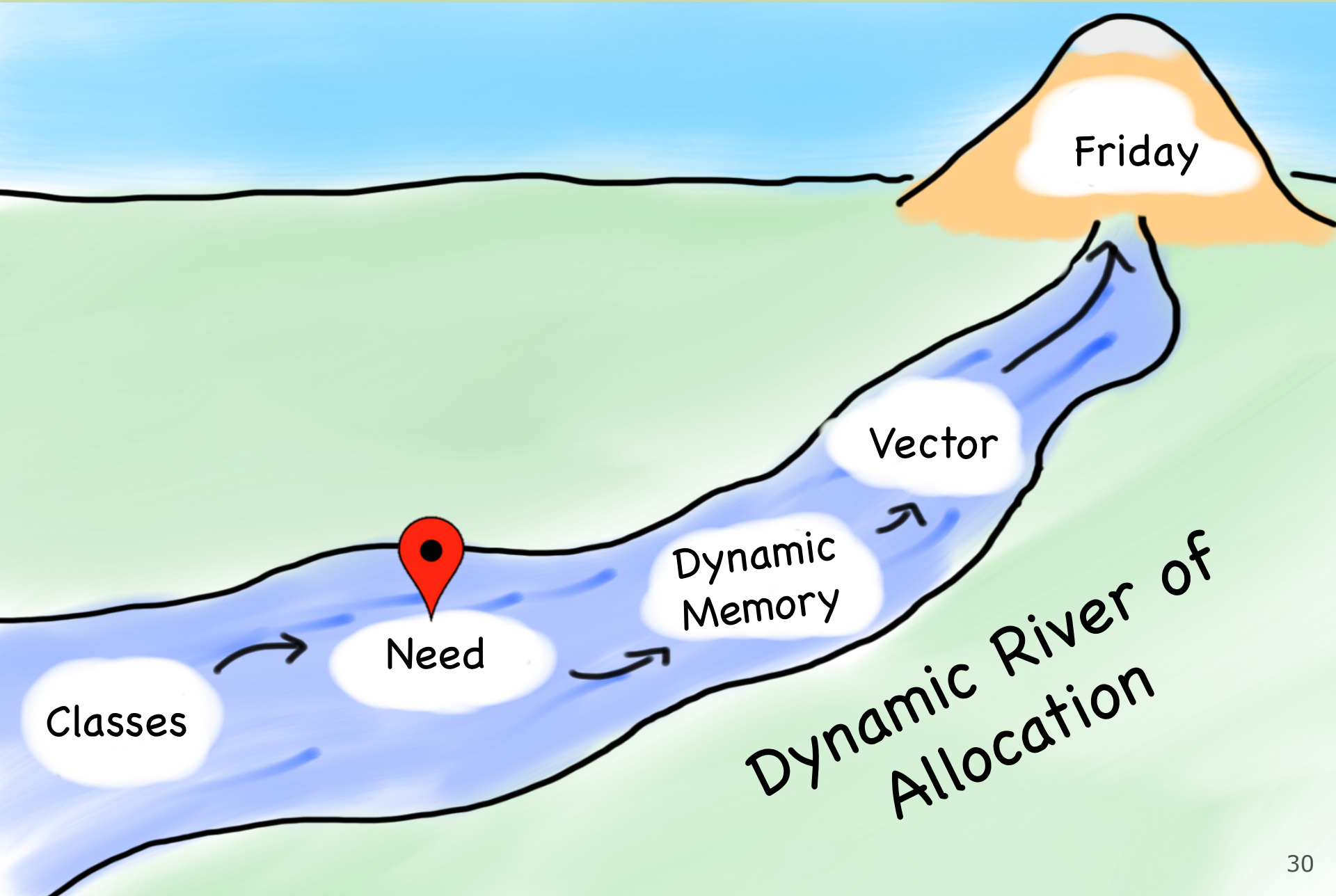
Its hard to share a single large object between classes.

Has Anyone in C++ land thought about this?

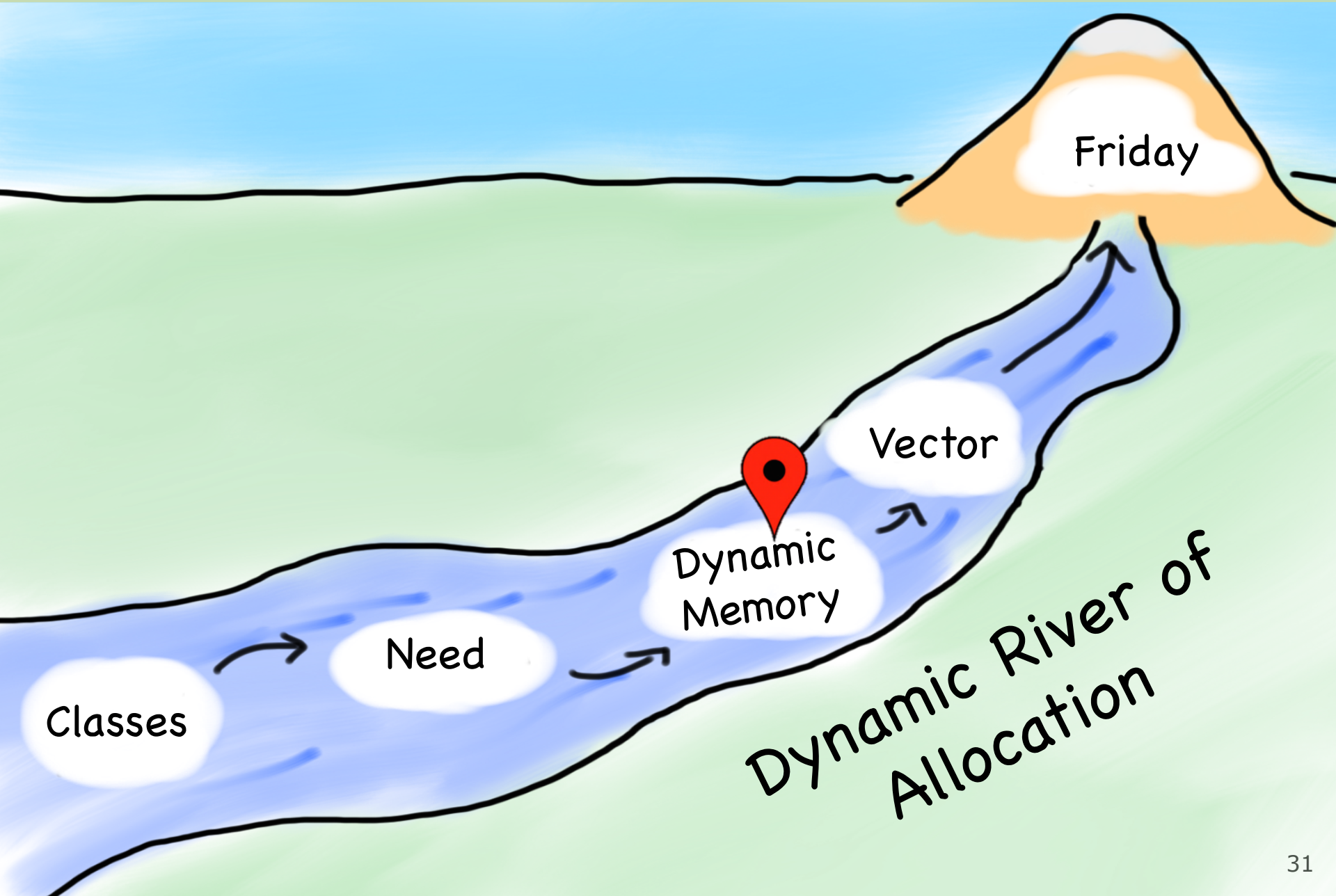
Yes

Dynamic Allocation!

Today's Goals



Today's Goals



Dynamic Allocation

// Makes a new int. Returns the address of the new int

```
new int;
```

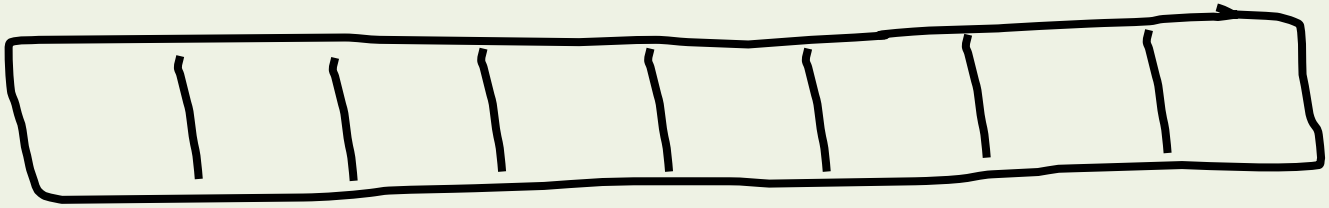
Dynamic Allocation

// Makes a new int. Returns the address of the new int

```
new int;
```

// Makes *n* new ints (in a block)
// returns the address of the block

```
new int[n];
```



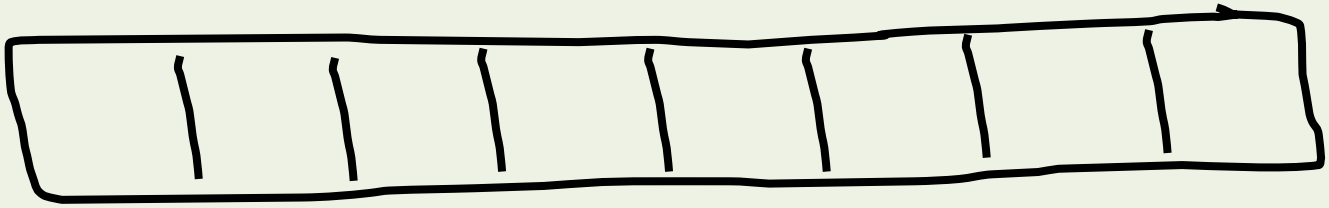
Dynamic Allocation

// Makes a new int. Returns the address of the new int

```
new int;
```

// Makes *n* new ints (in a block)
// returns the address of the block

```
new int[n];
```



* Unlike previous variables, these don't go out of scope!

Pointers

Definition: A pointer is a variable containing the address of another variable

Pointers

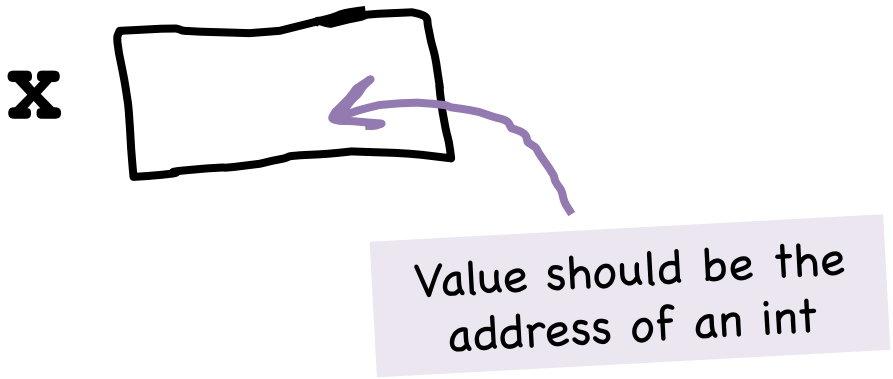
// a variable type that store the address of an int

```
int * x;
```

Pointers

// a variable type that store the address of an int

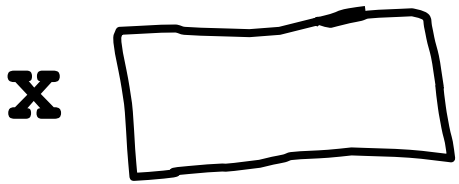
```
int * x;
```



Pointers

// a variable type that store the address of a char

```
char * x;
```

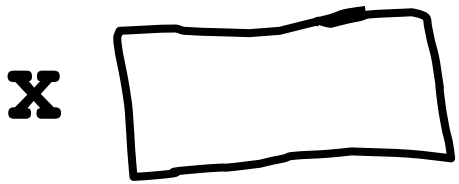


Value should be the address of a char

Pointers

// a variable type that store the address of a GImage

```
GImage * x;
```



Value should be the address of a GImage

Pointers

```
// dynamically request memory for a new GImage.
// calls the constructor.
// store the address of the new GImage.

GImage * image = new GImage("cat.png");
```

Pointers

```
// dynamically request memory for a new GImage.
// calls the constructor.
// store the address of the new GImage.

GImage * image = new GImage("cat.png");
```

Pointers

```
// dynamically request memory for a new GImage.
// calls the constructor.
// store the address of the new GImage.

GImage * image = new GImage("cat.png");
```

124134

Pointers

```
// dynamically request memory for a new GImage.
// calls the constructor.
// store the address of the new GImage.

GImage * image = new GImage("cat.png");
```

124134

124134

Pointers

```
// dynamically request memory for a new GImage.
// calls the constructor.
// store the address of the new GImage.
```

124134

```
GImage * image = new GImage("cat.png");
```

124134

Pointers

```
// dynamically request memory for a new GImage.
// calls the constructor.
// store the address of the new GImage.
```

124134

```
GImage * image = new GImage("cat.png");
```

image 124134

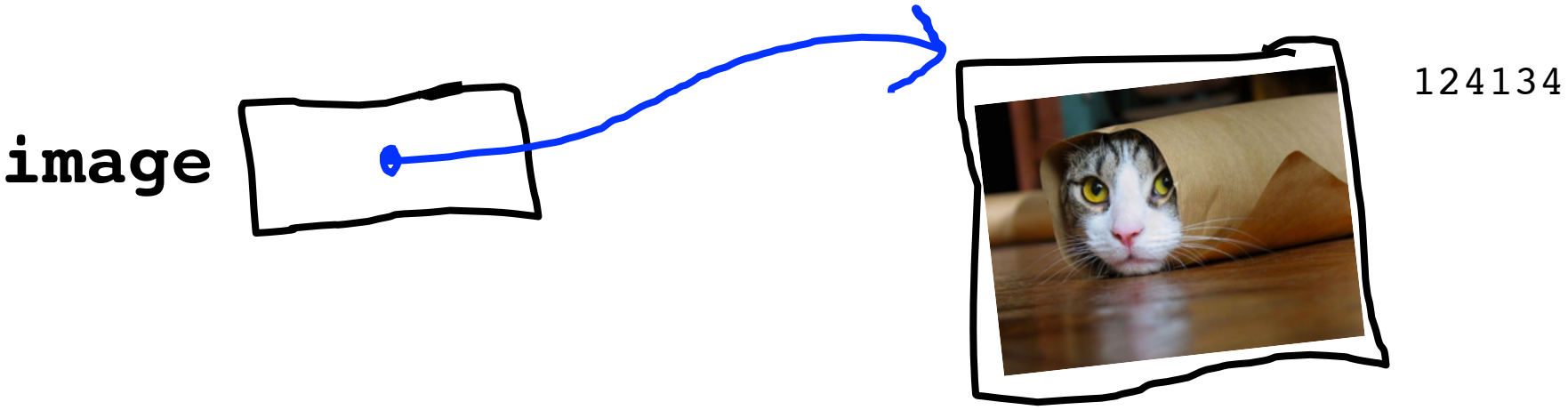
124134

Pointers

```
// dynamically request memory for a new GImage.
// calls the constructor.
// store the address of the new GImage.

GImage * image = new GImage("cat.png");
```

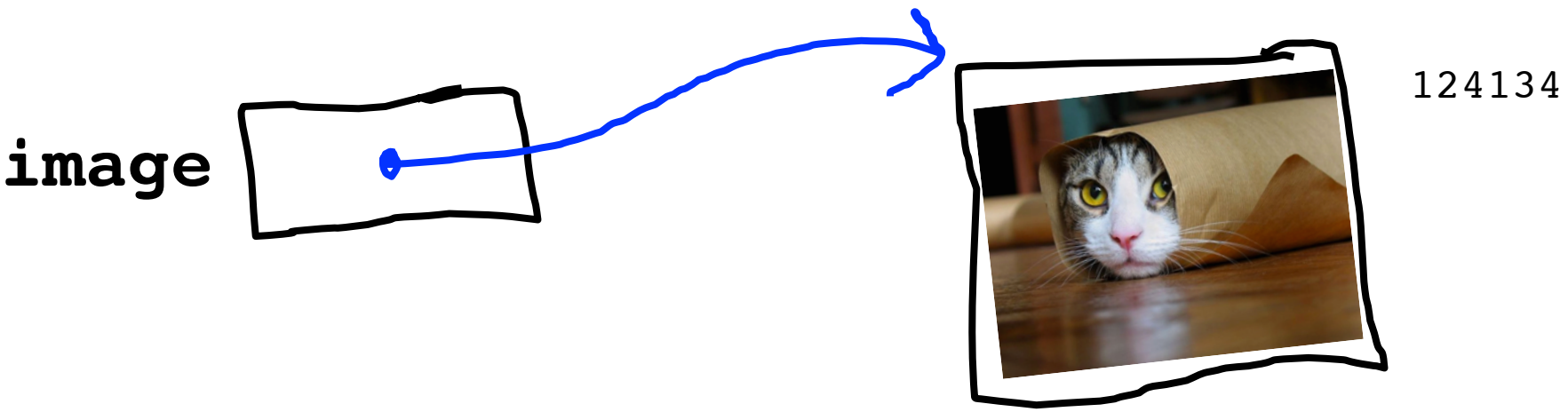
Pointers

```
// dynamically request memory for a new GImage.
// calls the constructor.
// store the address of the new GImage.

GImage * image = new GImage("cat.png");
```



Clean Up

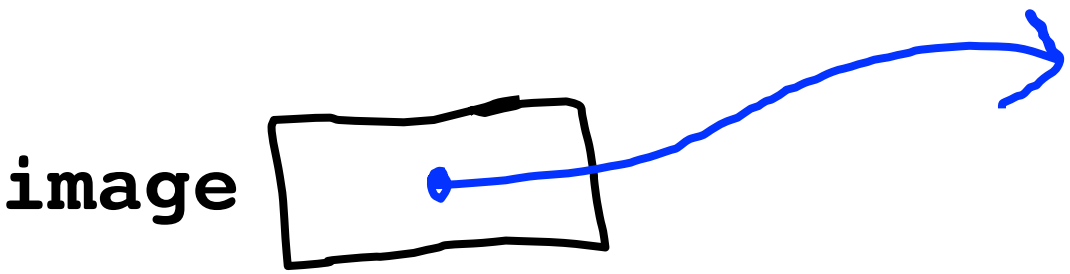
```
// dynamically request memory for a new GImage.
// calls the constructor.
// store the address of the new GImage.

GImage * image = new GImage("cat.png");
```




```
delete image;
```

Clean Up

```
// dynamically request memory for a new GImage.
// calls the constructor.
// store the address of the new GImage.

GImage * image = new GImage("cat.png");
```

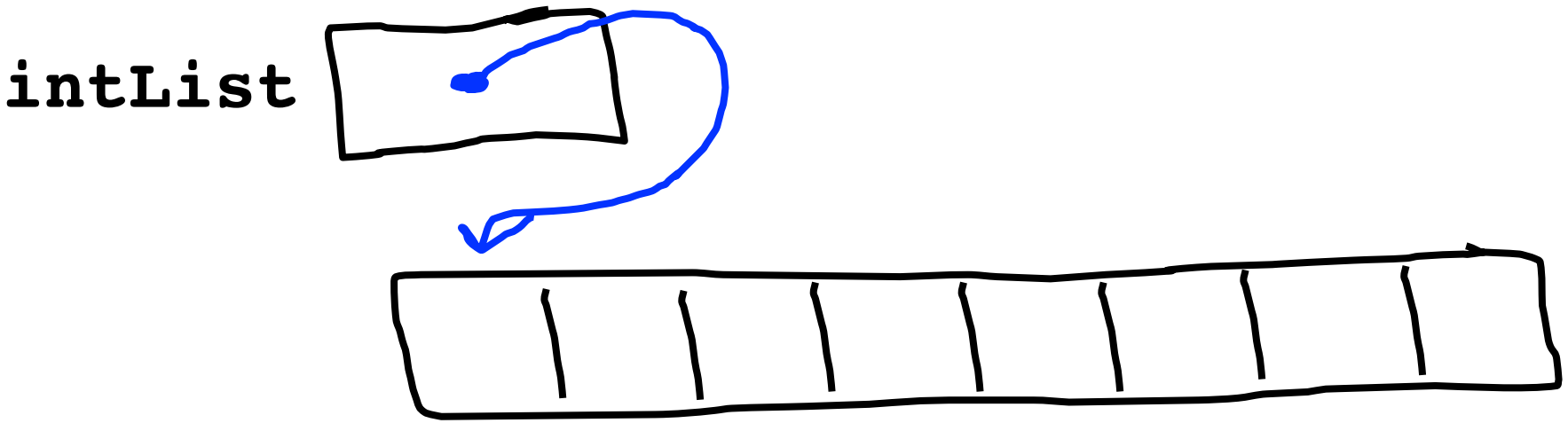



```
delete image;
```

Pointers

```
// dynamically request memory for n integers.
// store the address of the provided ints.
```

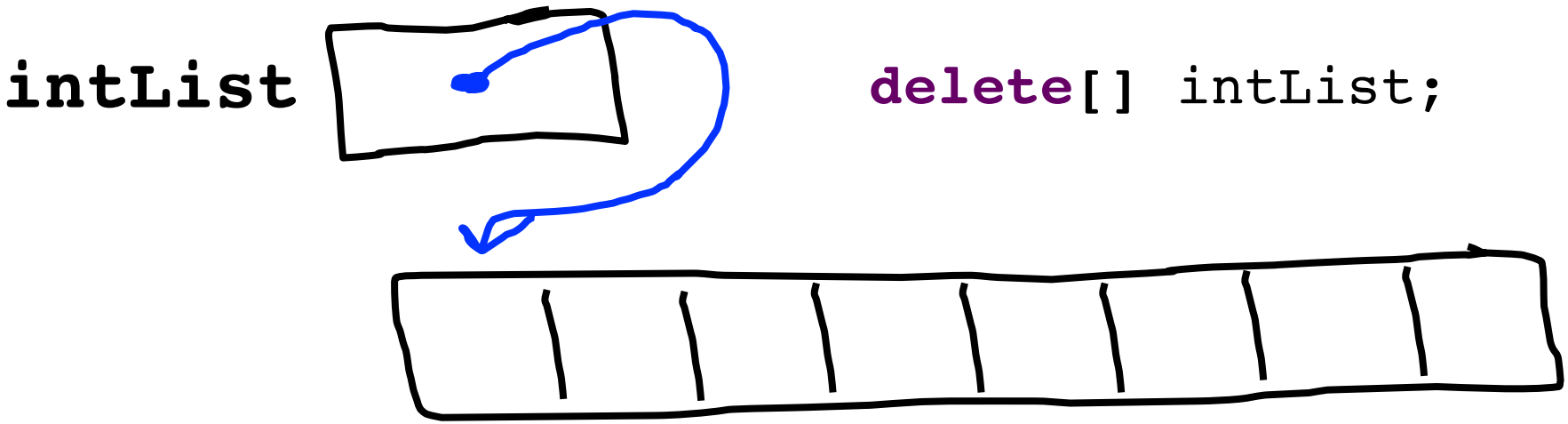
```
int * intList = new int[n];
```



Clean Up

```
// dynamically request memory for n integers.
// store the address of the provided ints.
```

```
int * intList = new int[n];
```

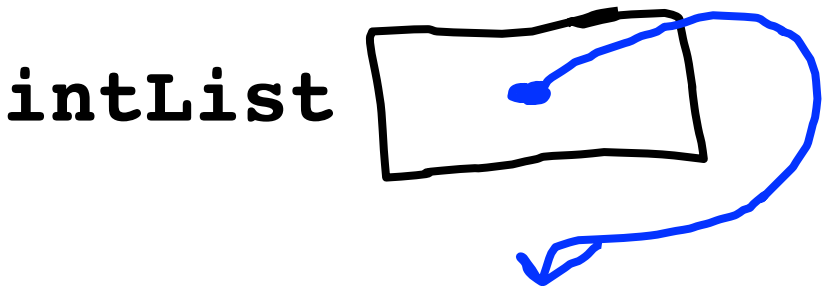



```
delete[] intList;
```

Clean Up

```
// dynamically request memory for n integers.
// store the address of the provided ints.
```

```
int * intList = new int[n];
```




```
delete[] intList;
```

How Does this Help?

Variables have to be known at compile time. Not runtime.

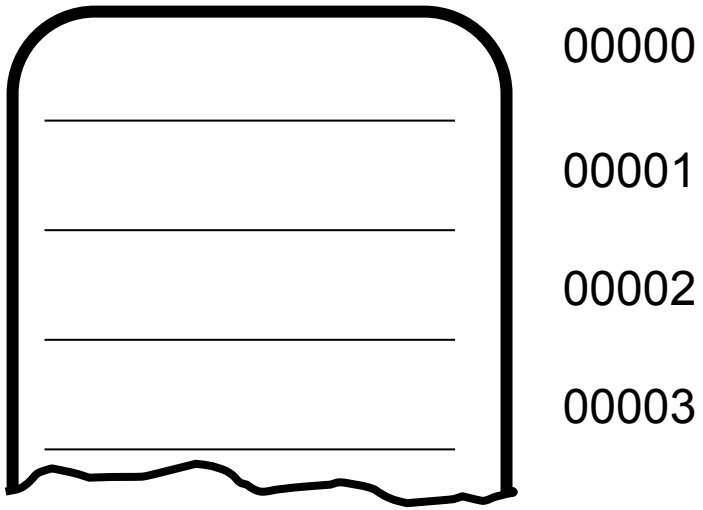
Persistence is out of our control.

Its hard to share a single large object between classes.

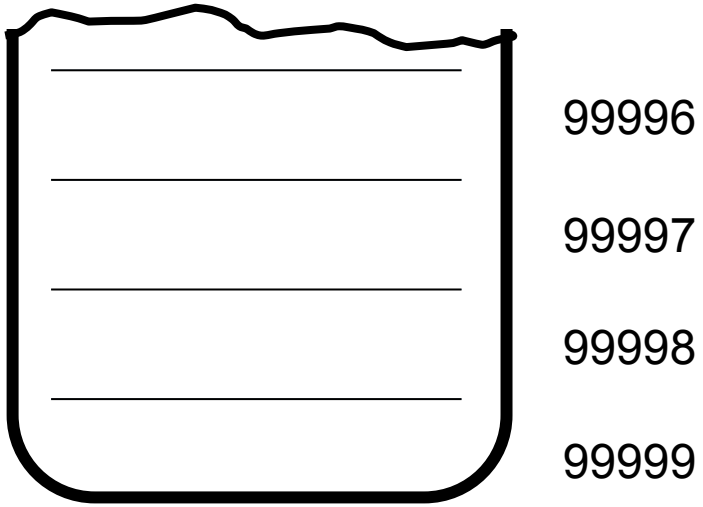
Dig deeper

All Memory Has an Address

RAM not disk



Each program gets it's own



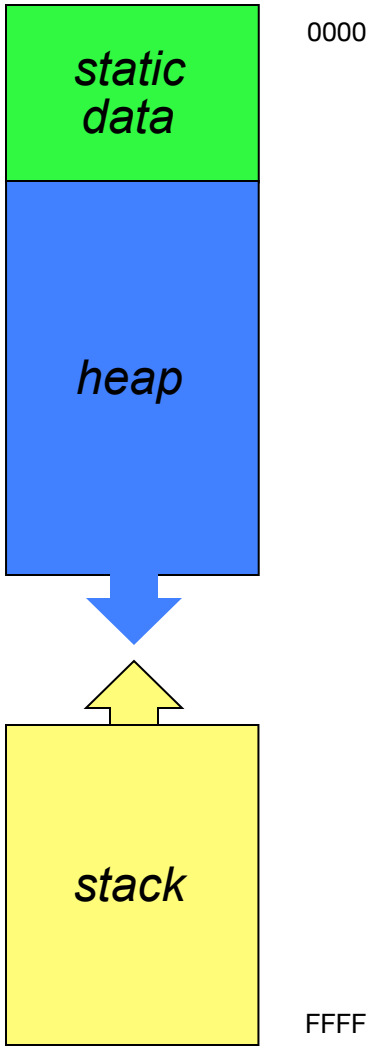
Allocation of Memory

Variables that persist throughout the lifetime of the program, such as constants.

Dynamically allocated variables

Variables declared locally in functions

The stack and heap grow toward each other to maximize the available space.



URL Metaphore

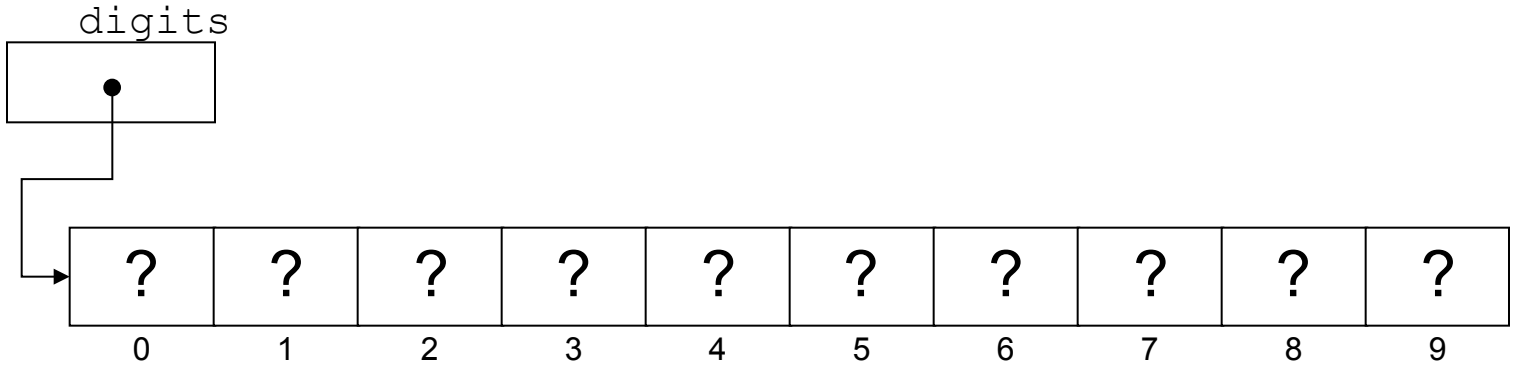
A pointer is like a URL. Not the actual page, the address of the page

Seat Number Metaphor

Dynamic Arrays

```
int size = getInteger("how big?");
int *digits = new int[size];
```

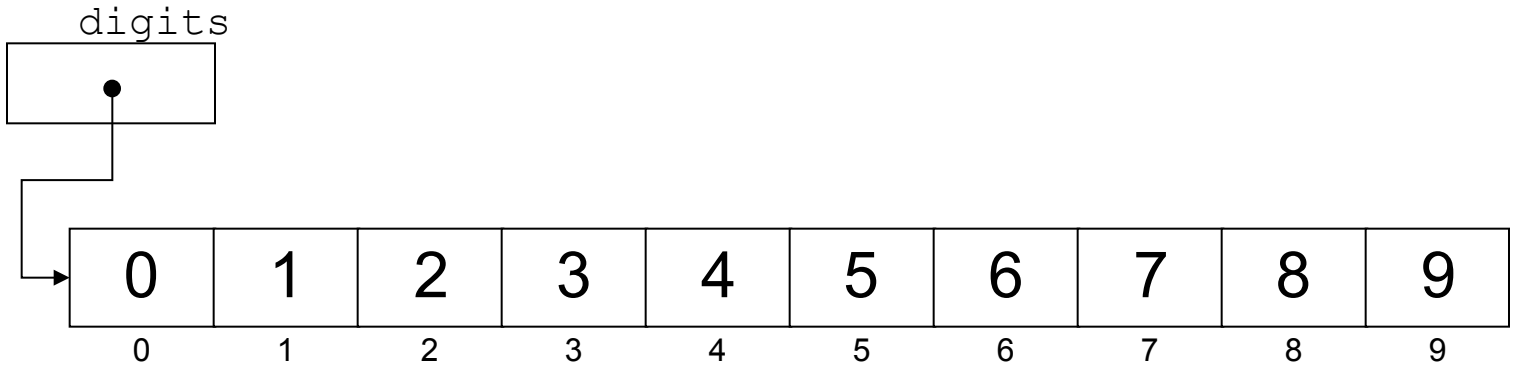
should result in the following configuration:



Dynamic Arrays

```
int size = getInteger("how big?");
int *digits = new int[size];
for(int i = 0; i < size; i++){
   digits[i] = i;
}
```

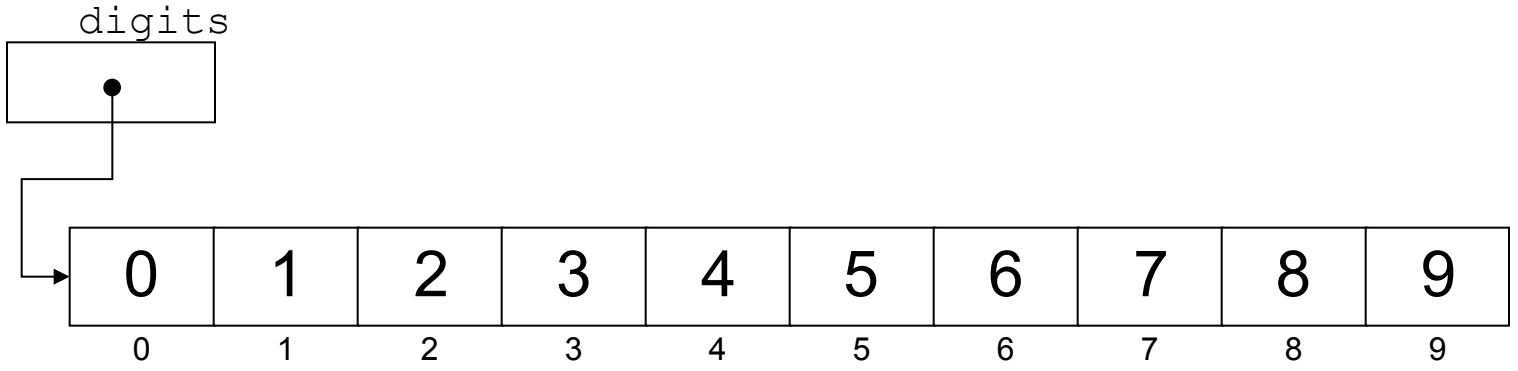
should result in the following configuration:



Dynamic Arrays

```
int size = getInteger("how big?");
int *digits = new int[size];
for(int i = 0; i < size; i++){
   digits[i] = i;
}
```

should result in the following configuration:

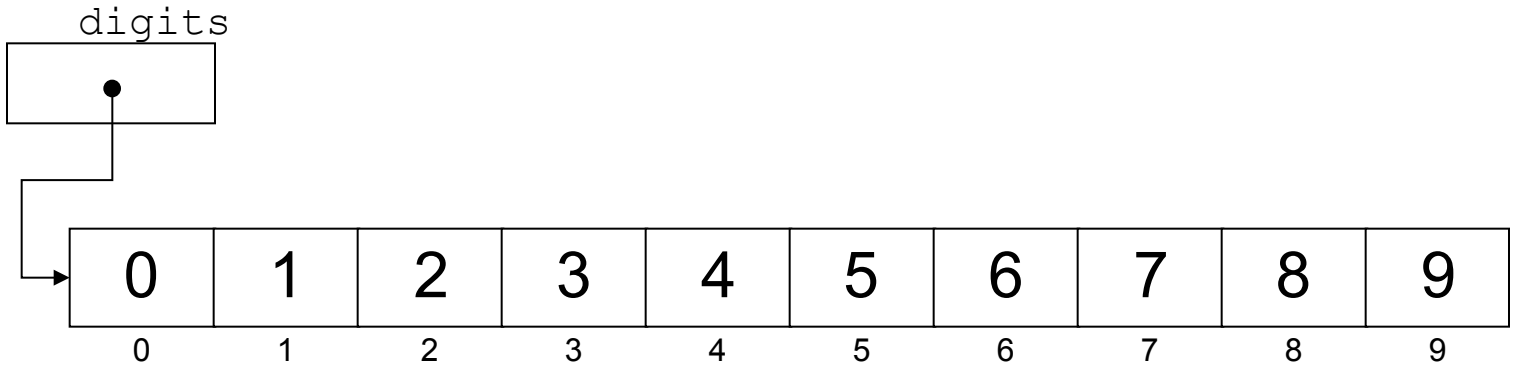


How would you free the memory allocated by this call?

Dynamic Arrays

```
int size = getInteger("how big?");
int *digits = new int[size];
for(int i = 0; i < size; i++){
   digits[i] = i;
}
```

should result in the following configuration:



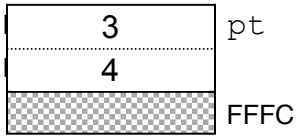
How would you free the memory allocated by this call?

```
delete [] digits;
```

Allocating a Point

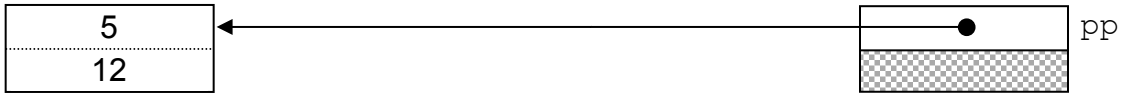
- The usual way to allocate a **`Point`** object is to declare it as a local variable on the stack, as follows:

```
Point pt(3, 4);
```



- It is, however, also possible to allocate a `Point` object on the heap using the following code:

```
Point *pp = new Point(5, 12);
```



The -> Opperator

- How do we access the points x? Or call getX()?

```
pp.getX()
```

because `pp` is not a Point (it's the address of one).

- To call a method (or access a member) given a pointer to an object, you need to write

```
pp->getX()
```

Socrative

```
Point * megan = new Point();
megan->setX(10);
Point * student = megan;
student->setY(7);

cout << student->getX();
cout << " ";
cout << student->getY();
```

a) 10, 7

b) crashes

c) ?, 7

d) ? ?

Socrative

```
Point * megan = new Point();
megan->setX(10);
Point * student = megan;
student->setY(7);

cout << student->getX();
cout << " ";
cout << student->getY();
```

a) 10, 7

b) crashes

c) ?, 7

d) ? ?

Socrative

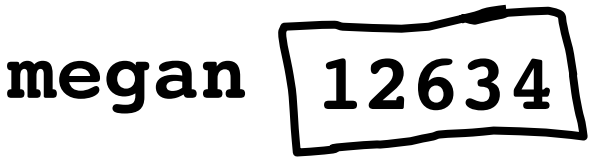
```
Point * megan = new Point();
megan->setX(10);
Point * student = megan;
student->setY(7);

cout << student->getX();
cout << " ";
cout << student->getY();
```

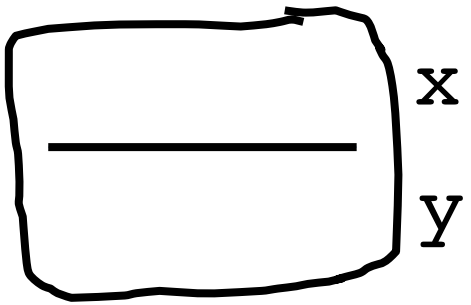
a) 10, 7

b) crashes

c) ?, 7

d) ? ?





12634

Socrative

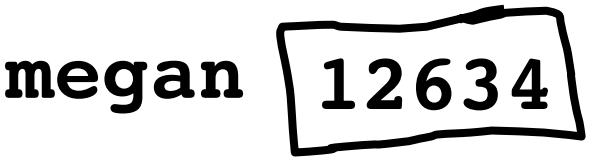
```
Point * megan = new Point();
megan->setX(10);
Point * student = megan;
student->setY(7);

cout << student->getX();
cout << " ";
cout << student->getY();
```

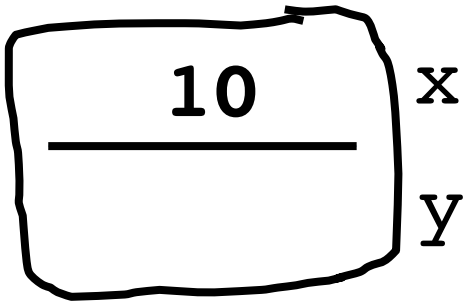
a) 10, 7

b) crashes

c) ?, 7

d) ? ?





12634

Socrative

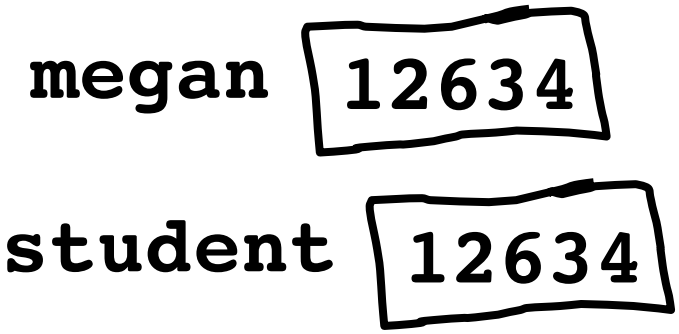
```
Point * megan = new Point();
megan->setX(10);
Point * student = megan;
student->setY(7);

cout << student->getX();
cout << " ";
cout << student->getY();
```

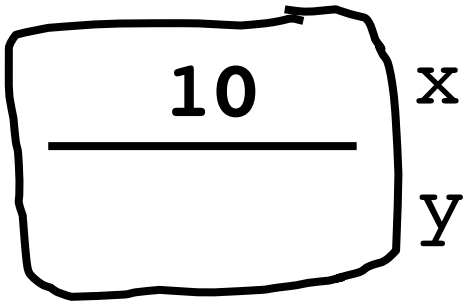
a) 10, 7

b) crashes

c) ?, 7

d) ? ?





12634

Socrative

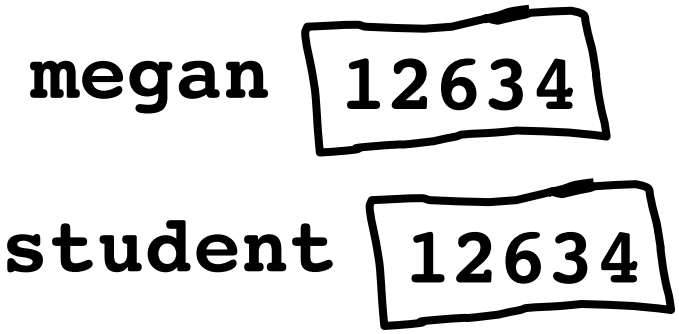
```
Point * megan = new Point();
megan->setX(10);
Point * student = megan;
student->setY(7);

cout << student->getX();
cout << " ";
cout << student->getY();
```

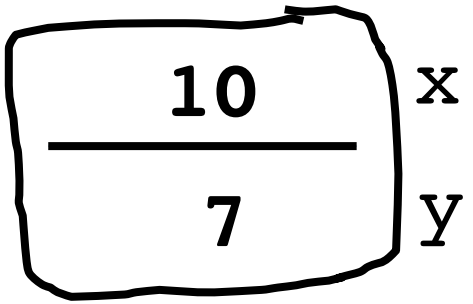
a) 10, 7

b) crashes

c) ?, 7

d) ? ?





12634

Socrative

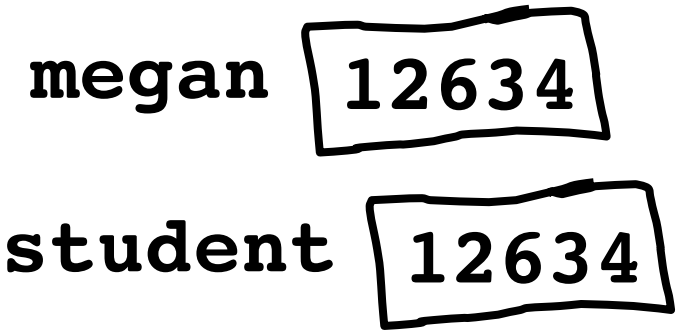
```
Point * megan = new Point();
megan->setX(10);
Point * student = megan;
student->setY(7);

cout << student->getX();
cout << " ";
cout << student->getY();
```

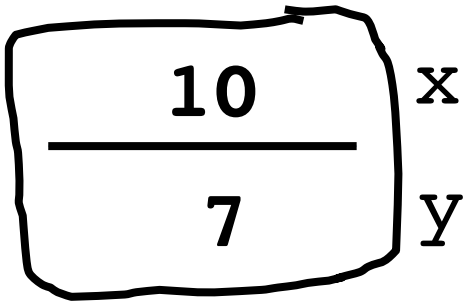
a) 10, 7

b) crashes

c) ?, 7

d) ? ?





12634

Socrative

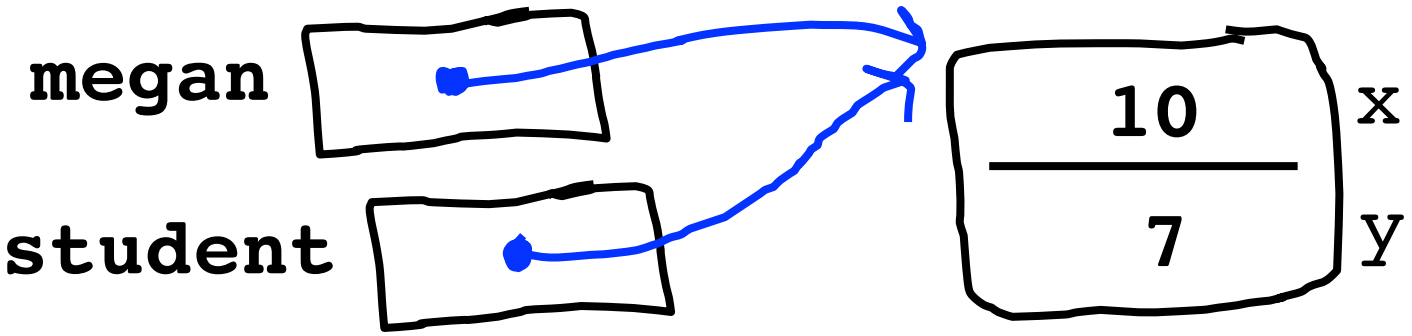
```
Point * megan = new Point();
megan->setX(10);
Point * student = megan;
student->setY(7);

cout << student->getX();
cout << " ";
cout << student->getY();
```

a) 10, 7

b) crashes

c) ?, 7

d) ? ?

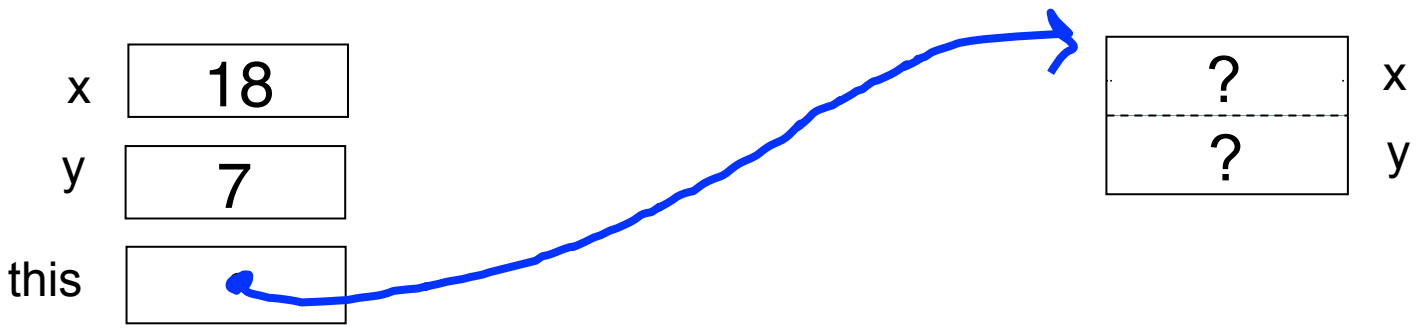


12634

The Keyword This

Pointer to the current instance a method was called on.

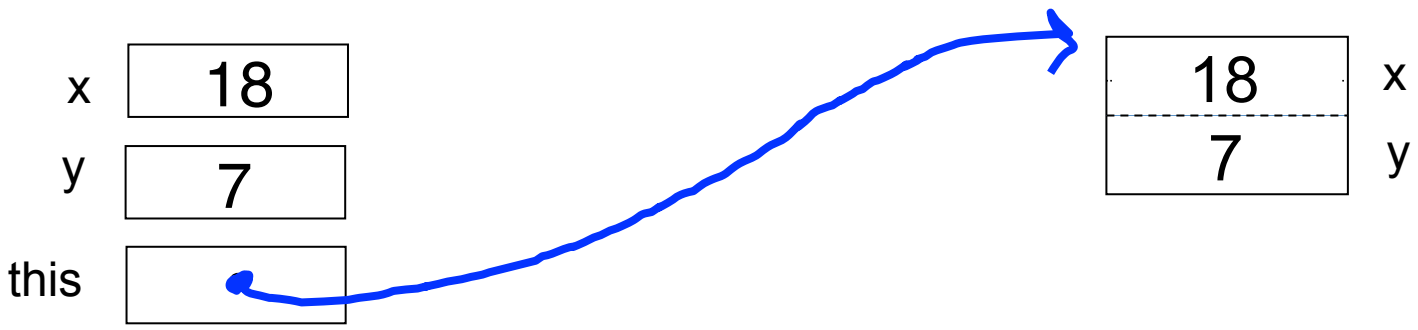
```
Point::Point(int x, int y) {
   this->x = x;
   this->y = y;
}
```



The Keyword This

Pointer to the current instance a method was called on.

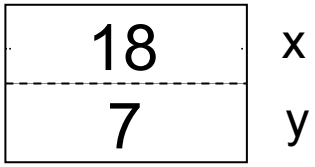
```
Point::Point(int x, int y) {
   this->x = x;
   this->y = y;
}
```



The Keyword This

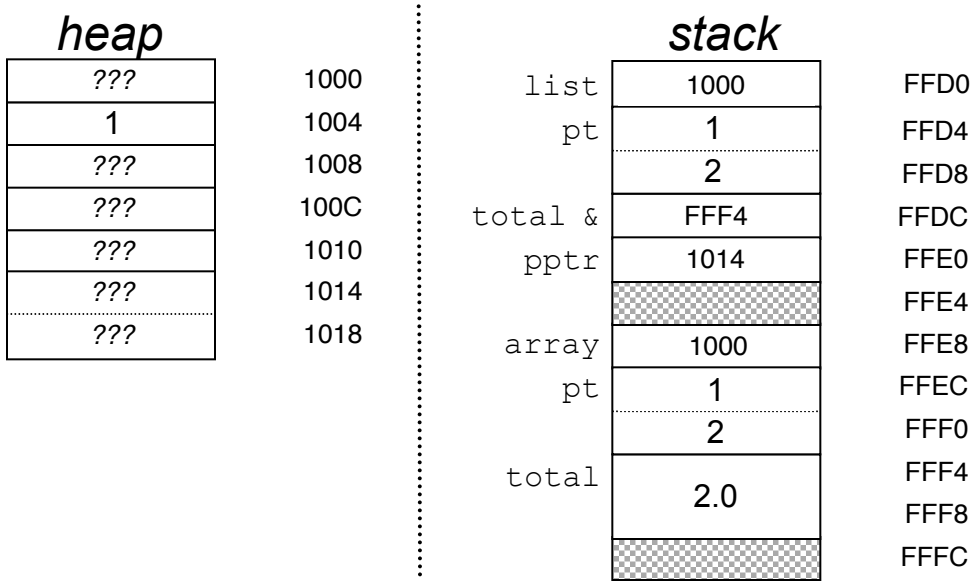
Pointer to the current instance a method was called on.

```
Point::Point(int x, int y) {
   this->x = x;
   this->y = y;
}
```



Heap Stack Diagram

```
int main() {
    void nonsense(int list[], Point pt, double & total) {
        Point *pptr = new Point;
        list[1] = pt.x;
        total += pt.y;
    }
```



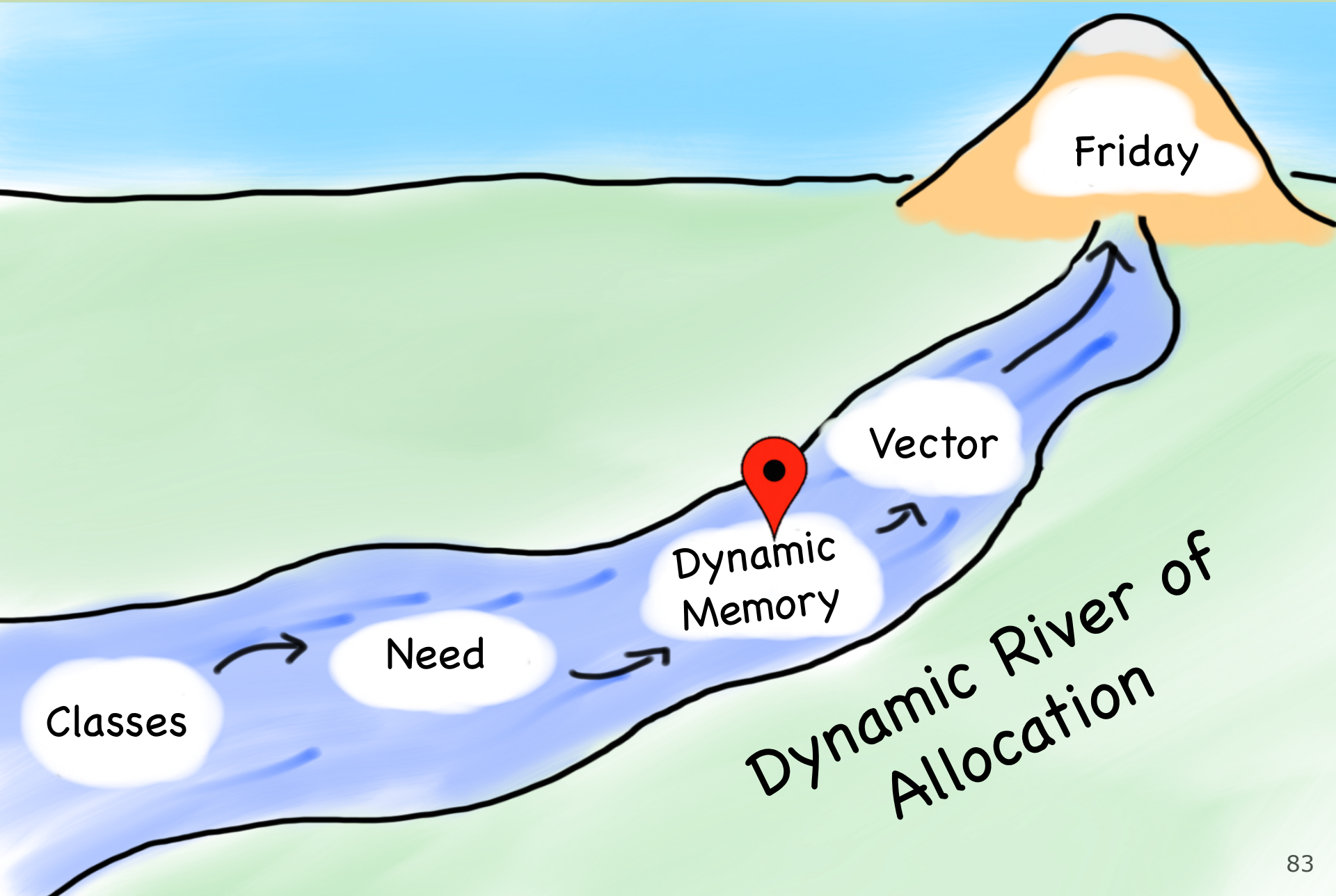
Pointers

Only Access Your Memory

Only a creepy killer would access a hotel room that isn't theirs (either never was, or was but checked out already)

- (Another great film about unusual people who work at hotels)

Today's Goals



Today's Goals

VectorInt

```
class VectorInt {          // in VectorInt.h
public:
    VectorInt();           // constructor

    void add(int value); // append a value to the end
    int get(int index);  // return the value at index

private:
    type name;      // member variables
    type name;      // (data inside each object)
};
```

Actual Vector

```
class VectorInt {          // in VectorInt.h
public:
    VectorInt();           // constructor

    void add(int value); // append a value to the end
    int get(int index);  // return the value at index

private:
    int * data;
};
```

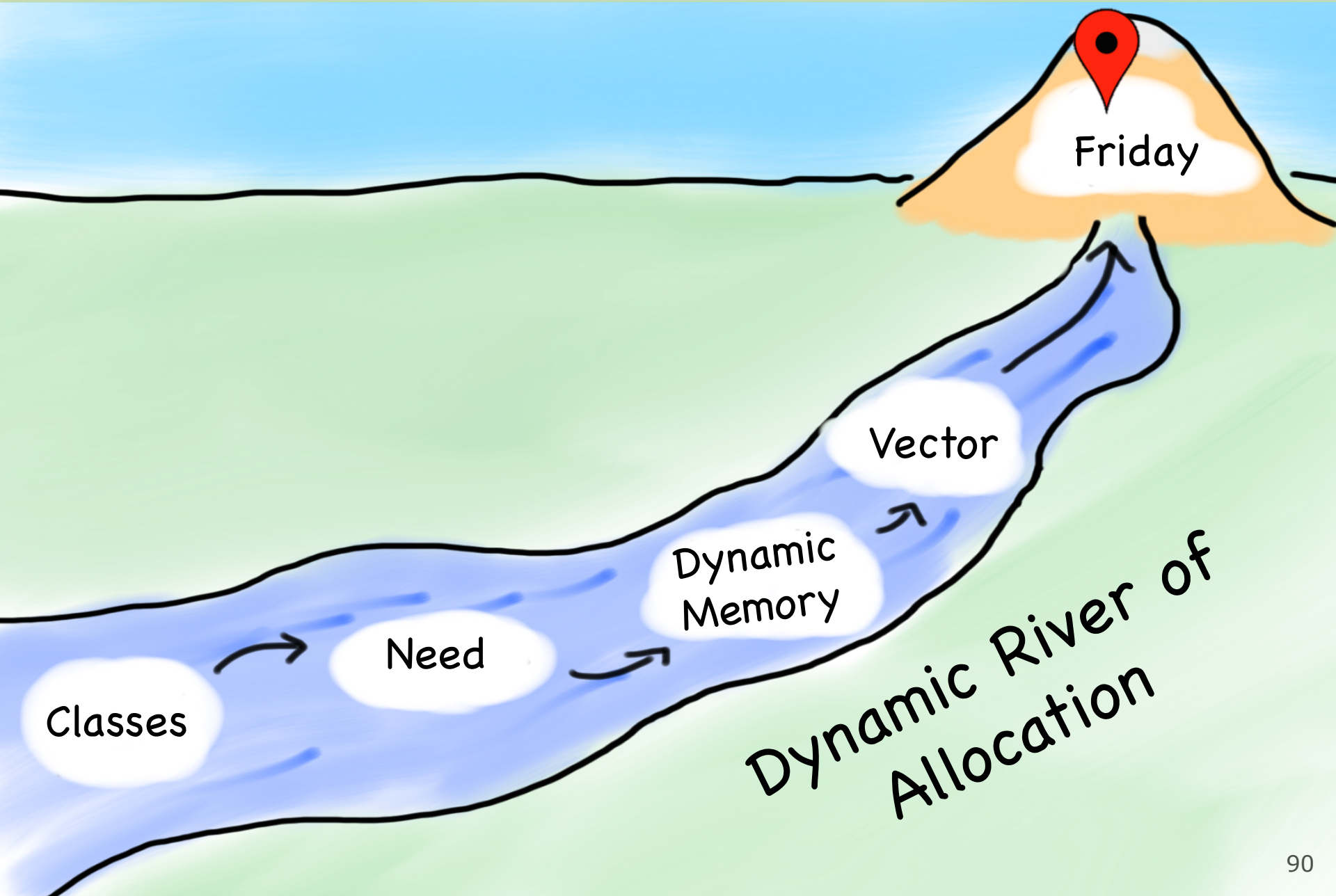
Actual Vector

```
class VectorInt {          // in VectorInt.h
public:
    VectorInt();           // constructor

    void add(int value); // append a value to the end
    int get(int index);  // return the value at index

private:
    int * data;
    int size;
    int allocatedSize;
};
```

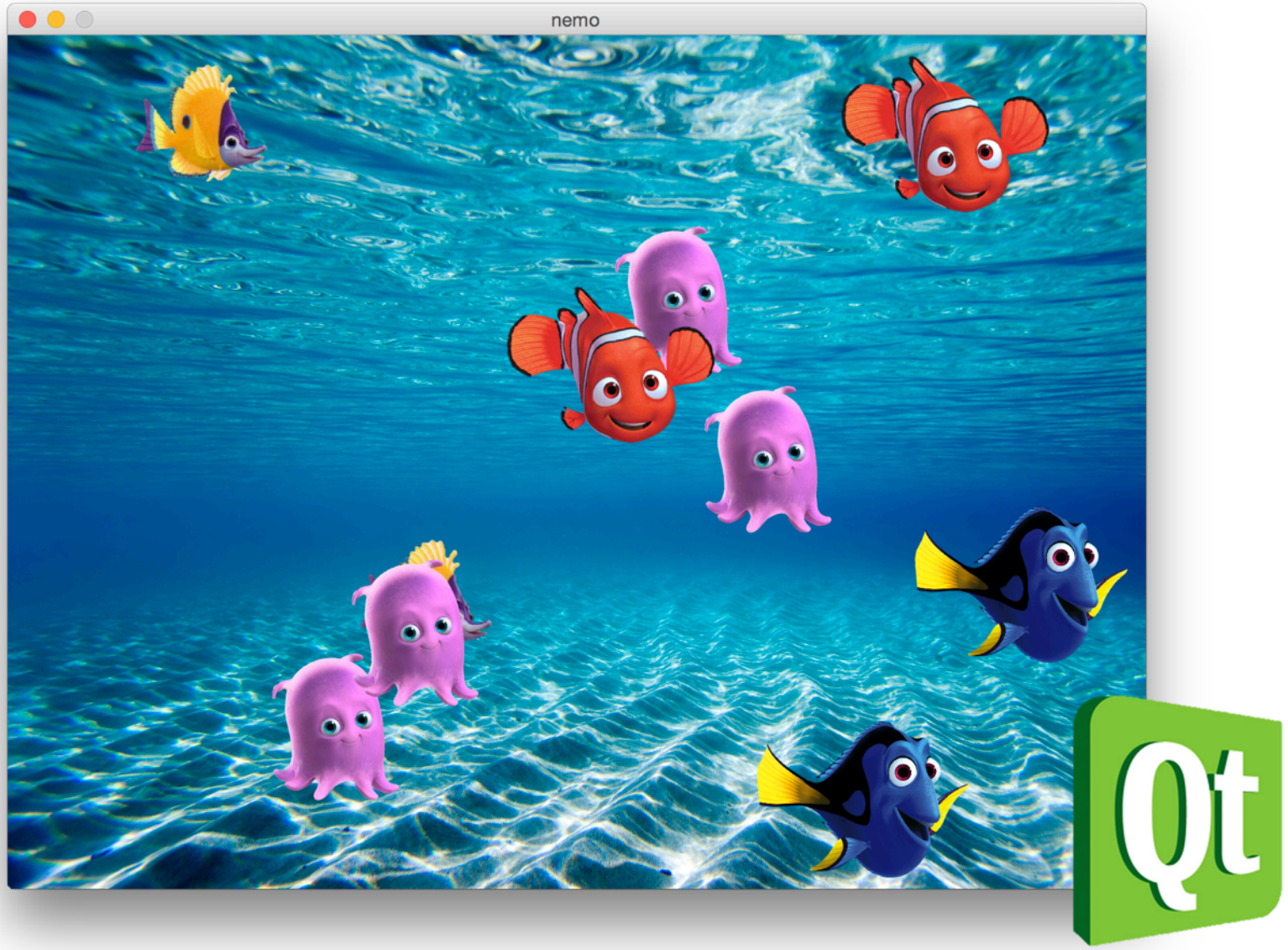

Qt

Today's Goals

Today's Goals



Another Pointer Example



Today's Goals

1. Learn how to dynamically create vars
2. Learn how to access dynamic memory
3. Learn how Vector works