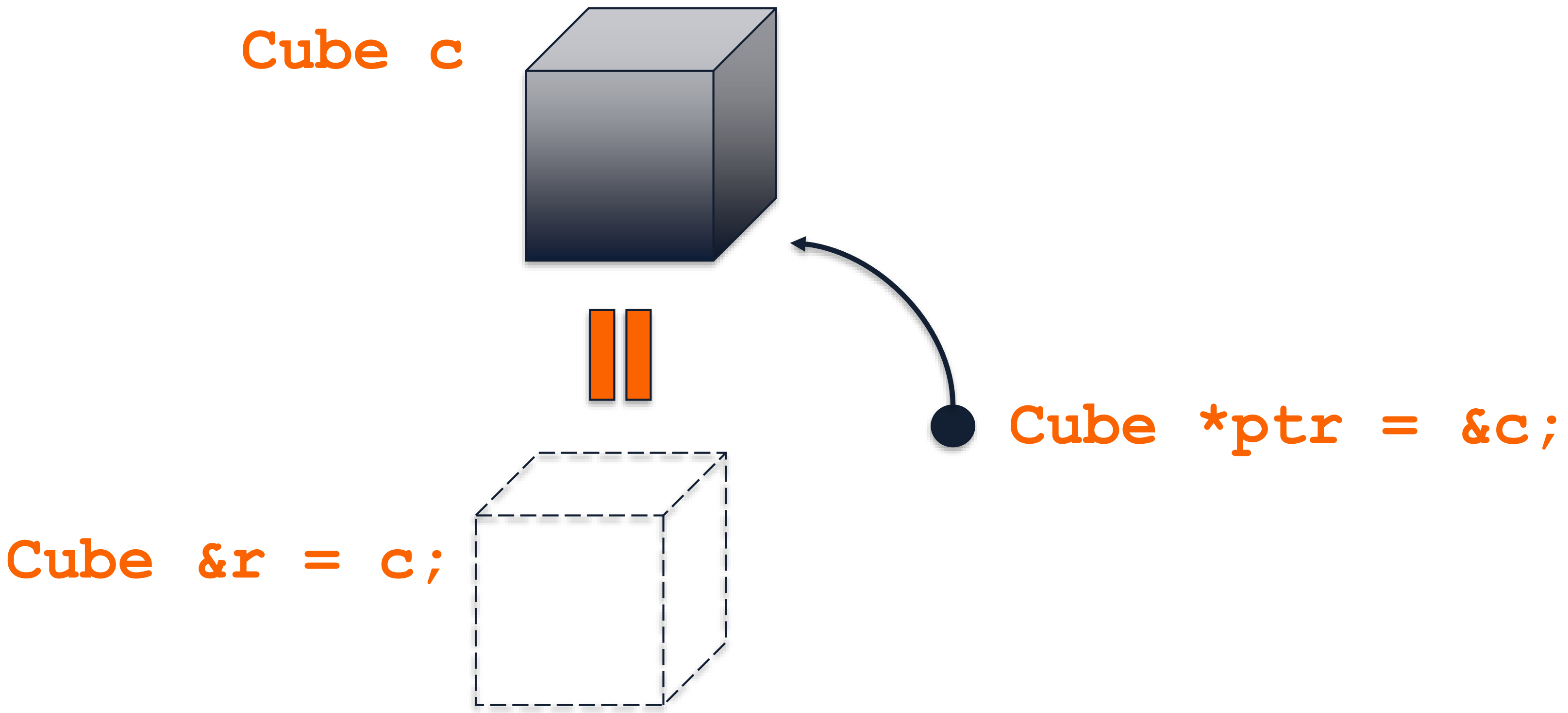
Variable Storage

Prof. Wade Fagen-Ulmschneider

ILLINOIS

In C++, an instance of a variable can be stored directly in memory, accessed by pointer, **or** accessed by reference.



Direct Storage

By default, variables are stored directly in memory.

- The **type** of a variable has no modifiers.
- The object takes up exactly its size in memory.

```
Cube c;                // Stores a Cube in memory
int i;                 // Stores an integer in memory
uiuc::HSLAPixel p;     // Stores a pixel in memory
```

Storage by Pointer

- The **type** of a variable is modified with an asterisk (*).
- A pointer takes a "memory address width" of memory (ex: 64 bits on a 64-bit system).
- The pointer "points" to the allocated space of the object.

```
Cube *c;              // Pointer to a Cube in memory
int *i;               // Pointer to an integer in memory
uiuc::HSLAPixel *p;  // Pointer to a pixel in memory
```

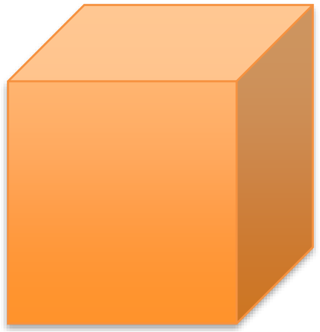
Storage by Reference

- A reference is an **alias** to existing memory and is denoted in the type with an ampersand (`&`).
- A reference does not store memory itself, it is only an alias to another variable.
- The alias must be assigned when the variable is initialized.

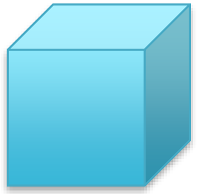
```
Cube &c = cube;       // Alias to the variable `cube`
int &i = count;       // Alias to the variable `i`
uiuc::HSLAPixel &p; // Illegal!  Must alias something
                       when variable is initialized.
```

Example: Cube Currency

Suppose our cubes have a value to them, based on their volume:



10^3 volume
💰1,000

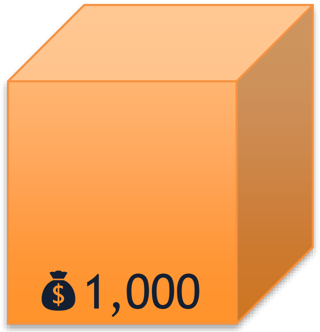


5^3 volume
💰125

1^3 volume
💰1

Example: Cube Currency

When we receive money, we want the cube itself – not a copy of the cube.



cpp-memory2/Cube.cpp

```cpp
Cube::Cube(double length) {
  length_ = length;
  std::cout << "Created $" << getVolume() << std::endl;
}

Cube::Cube(const Cube & obj) {
  length_ = obj.length_;
  std::cout << "Created $" << getVolume() << " via copy" << std::endl;
}

Cube & Cube::operator=(const Cube & obj) {
  std::cout << "Transformed $" << getVolume() << "-> $" <<
                                        obj.getVolume() << std::endl;
  length_ = obj.length_;
  return *this;
}
```

cpp-memory2/ex1/byValue.cpp

```cpp
int main() {
  // Create a 1,000-valued cube
  Cube c(10);

  // Transfer the cube
  Cube myCube = c;

  return 0;
}
```

cpp-memory2/ex1/byRef.cpp

```cpp
int main() {
  // Create a 1,000-valued cube
  Cube c(10);

  // Transfer the cube
  Cube & myCube = c;

  return 0;
}
```

cpp-memory2/ex1/byPointer.cpp

```cpp
int main() {
  // Create a 1,000-valued cube
  Cube c(10);

  // Transfer the cube
  Cube * myCube = &c;

  return 0;
}
```

Pass by ___________

Identical to storage, arguments can be passed to functions in three different ways:

- Pass by **value** (default)
- Pass by **pointer** (modified with `*`)
- Pass by **reference** (modified with `&`, acts as an alias)

cpp-memory2/ex2/byValue.cpp

```cpp
bool sendCube(Cube c) {
  // ... logic to send a Cube somewhere ...
  return true;
}

int main() {
  // Create a 1,000-valued cube
  Cube c(10);

  // Send the cube to someone
  sendCube(c);

  return 0;
}
```

cpp-memory2/ex2/byRef.cpp

```cpp
bool sendCube(Cube & c) {
  // ... logic to send a Cube somewhere ...
  return true;
}

int main() {
  // Create a 1,000-valued cube
  Cube c(10);

  // Send the cube to someone
  sendCube(c);

  return 0;
}
```

cpp-memory2/ex2/byPointer.cpp

```cpp
bool sendCube(Cube * c) {
  // ... logic to send a Cube somewhere ...
  return true;
}

int main() {
  // Create a 1,000-valued cube
  Cube c(10);

  // Send the cube to someone
  sendCube(&c);

  return 0;
}
```

Return by ______________

Similarly, values can be returned all three ways as well:

- Return by **value** (default)
- Return by **pointer** (modified with `*`)
- Return by **reference** (modified with `&`, acts as an alias)
  - *Never return a reference to a stack variable created on the stack of your current function!*