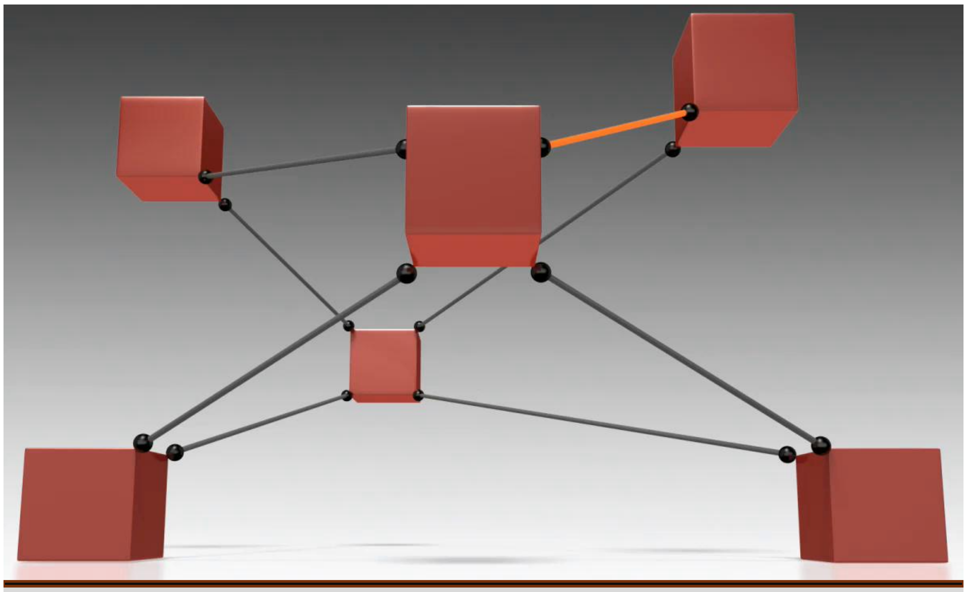
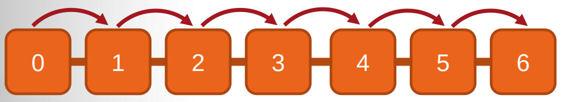


Iterator Design Pattern

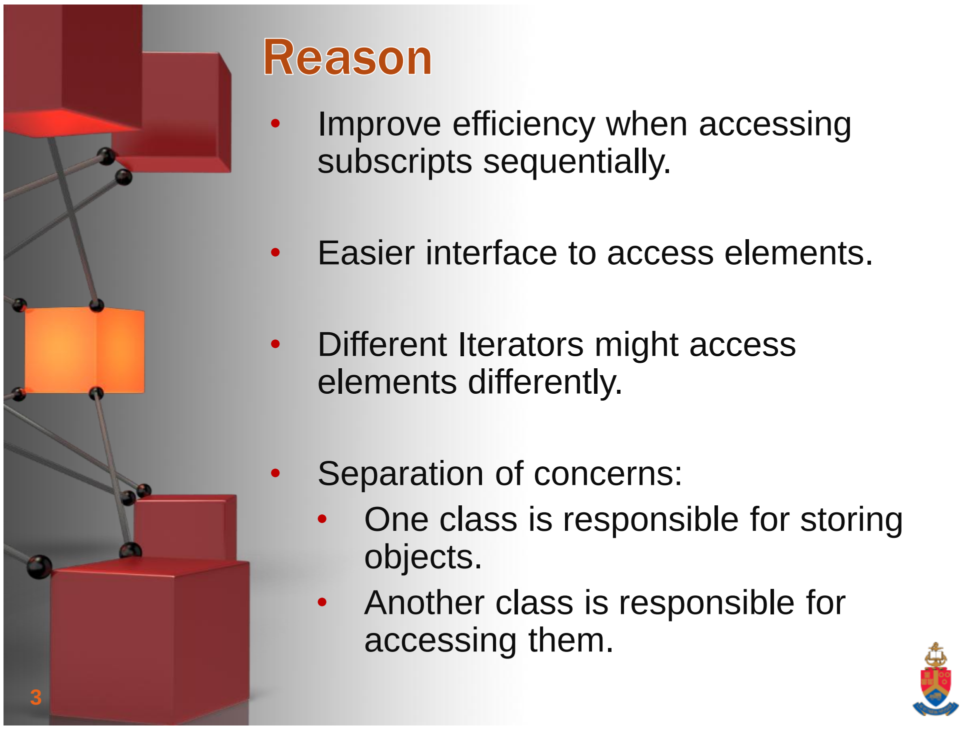
COS 121 – Christoph Stallmann

Introduction

- To iterate means to repeat.
- Implemented as:
  - Recursion
  - Loop structures: *for* and *while* loops
- A class that supports iterations is called an iterator.



2

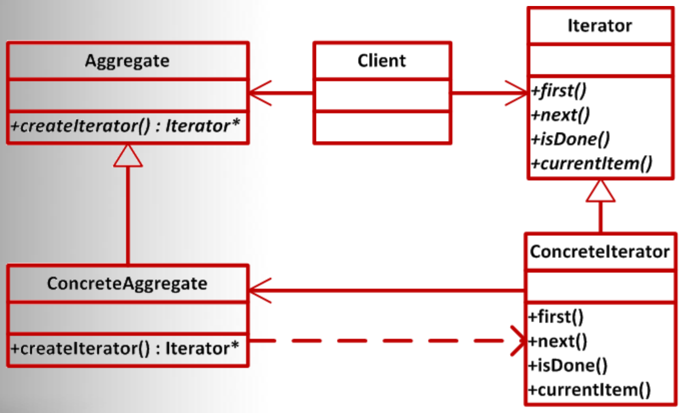


Reason

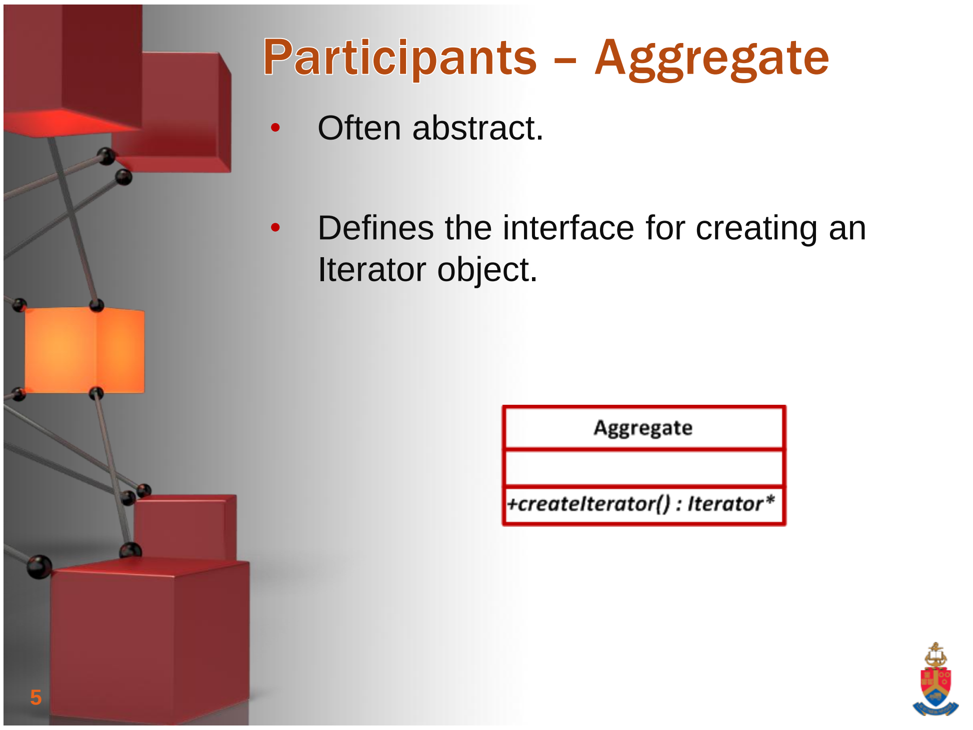
- Improve efficiency when accessing subscripts sequentially.
- Easier interface to access elements.
- Different Iterators might access elements differently.
- Separation of concerns:
  - One class is responsible for storing objects.
  - Another class is responsible for accessing them.

3

Structure



4



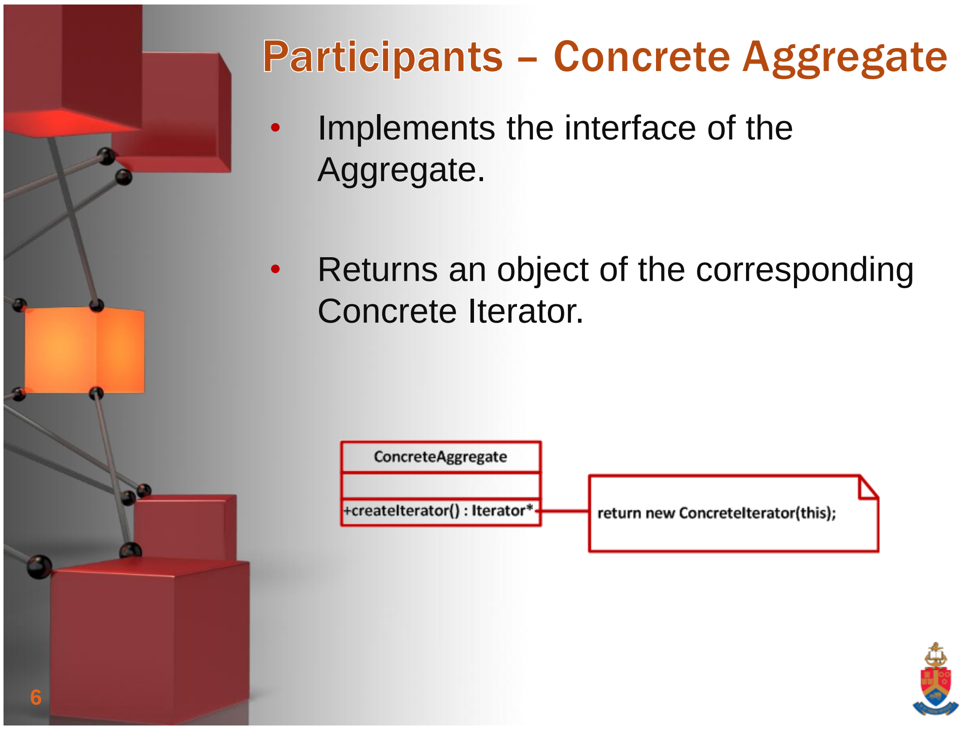
Participants – Aggregate

- Often abstract.
- Defines the interface for creating an Iterator object.

Aggregate

*+createIterator() : Iterator**

5



Participants – Concrete Aggregate

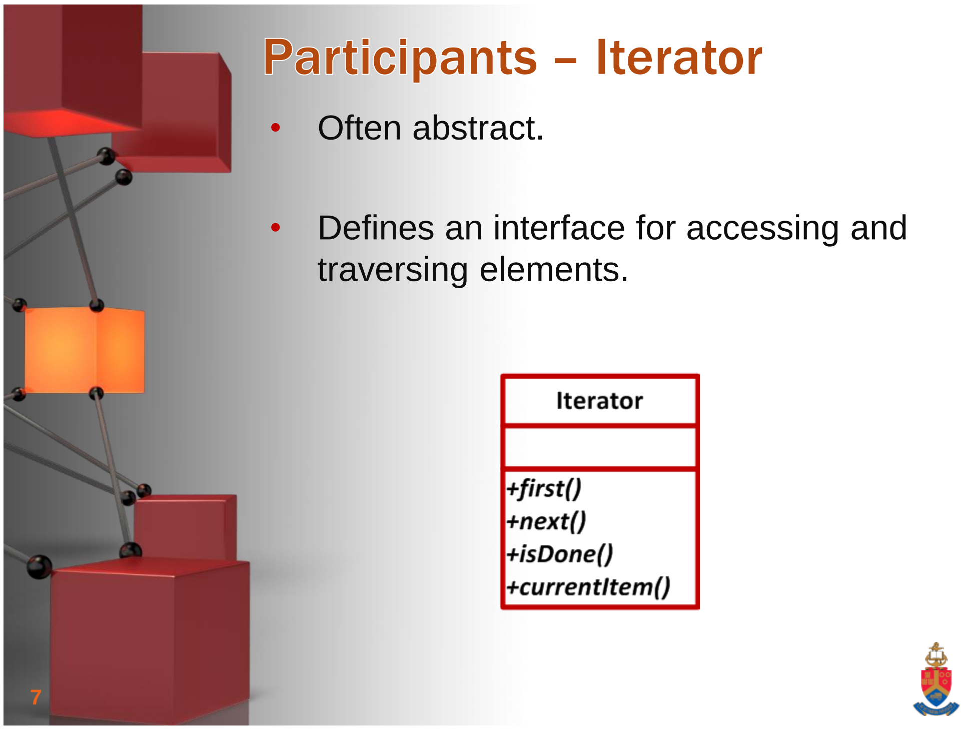
- Implements the interface of the Aggregate.
- Returns an object of the corresponding Concrete Iterator.

ConcreteAggregate

+createIterator() : Iterator*

return new ConcreteIterator(this);

6



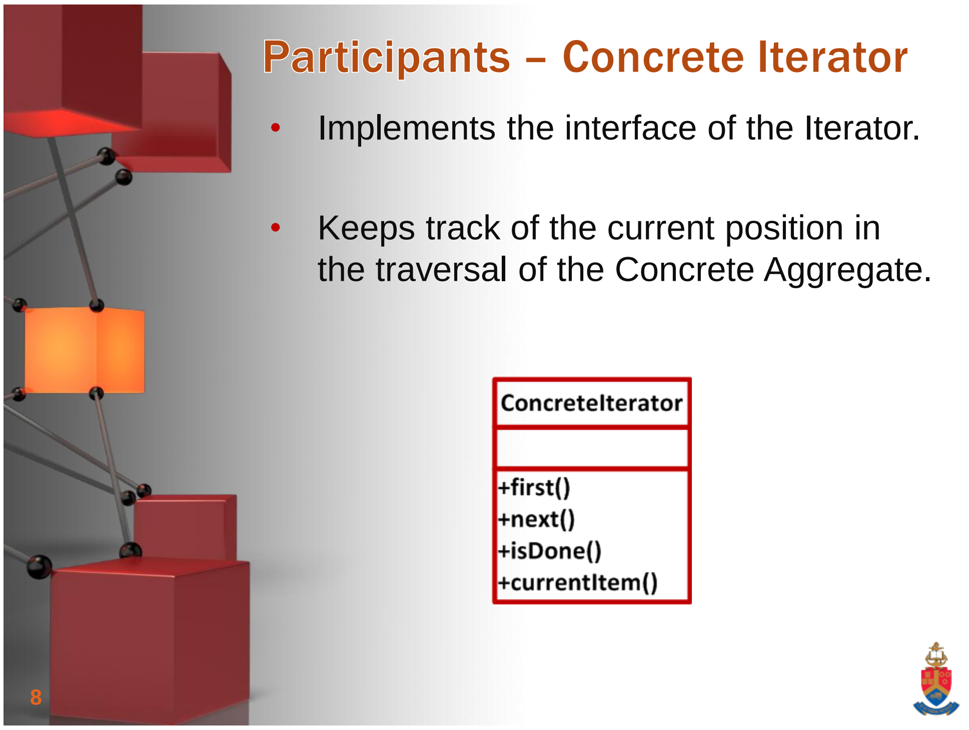
Participants – Iterator

- Often abstract.
- Defines an interface for accessing and traversing elements.

Iterator

+first()
+next()
+isDone()
+currentItem()

7



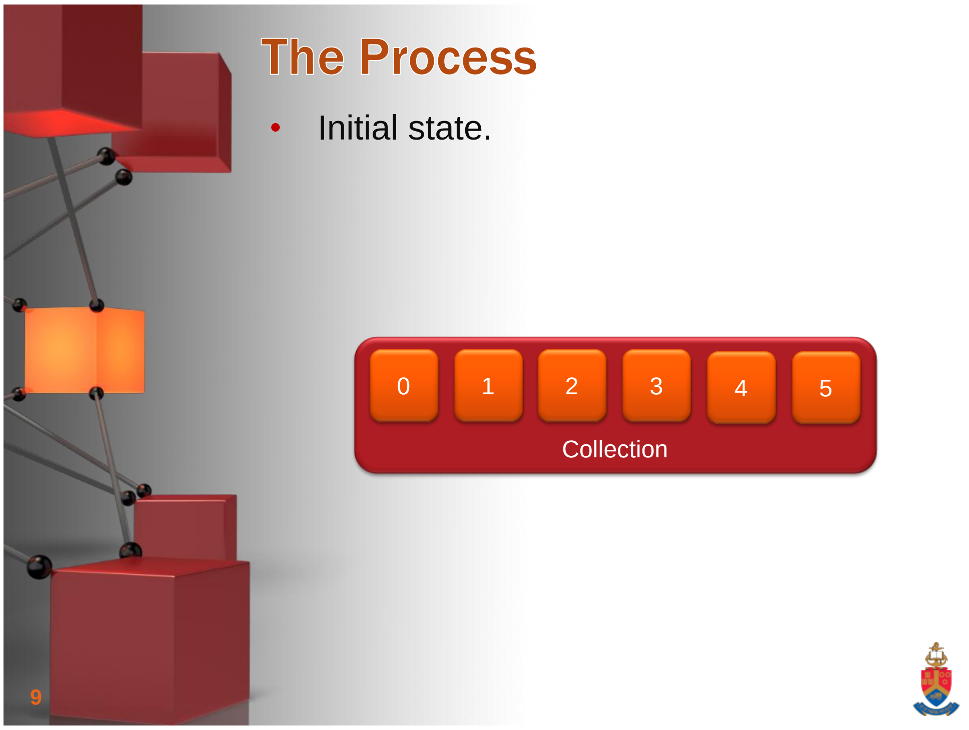
Participants – Concrete Iterator

- Implements the interface of the Iterator.
- Keeps track of the current position in the traversal of the Concrete Aggregate.

ConcreteIterator

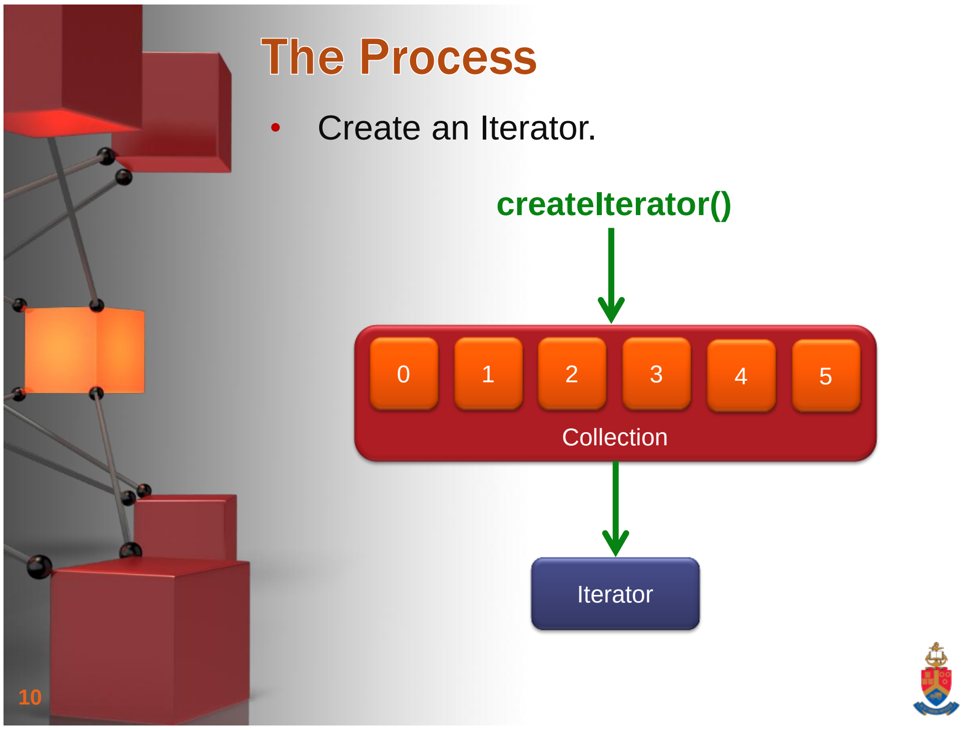
+first()
+next()
+isDone()
+currentItem()

8



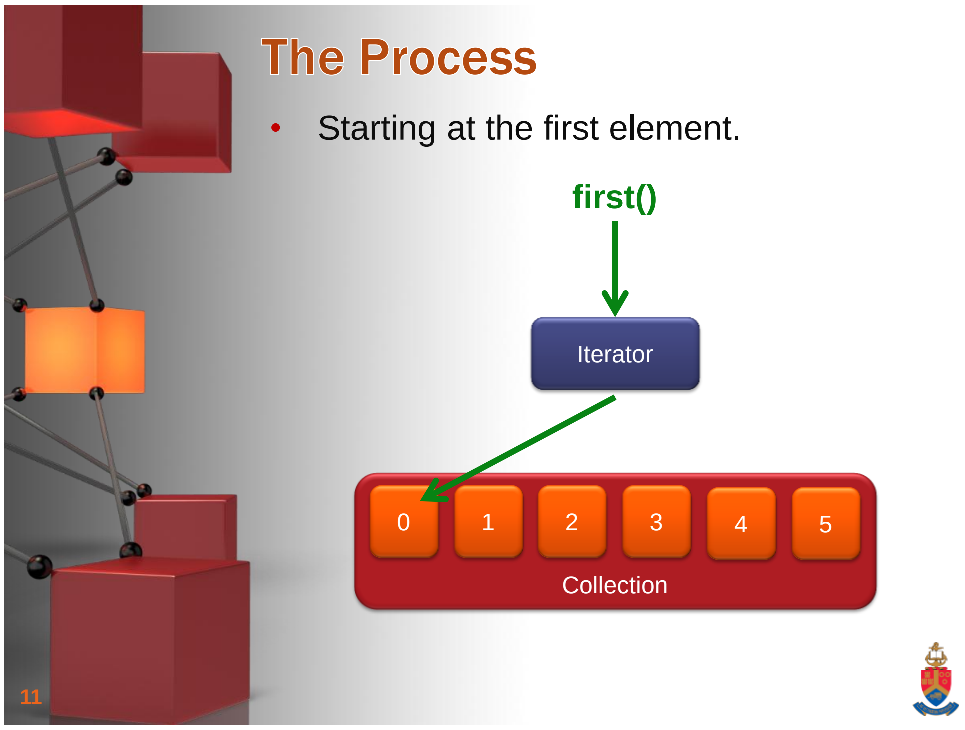
The Process

- Initial state.



The Process

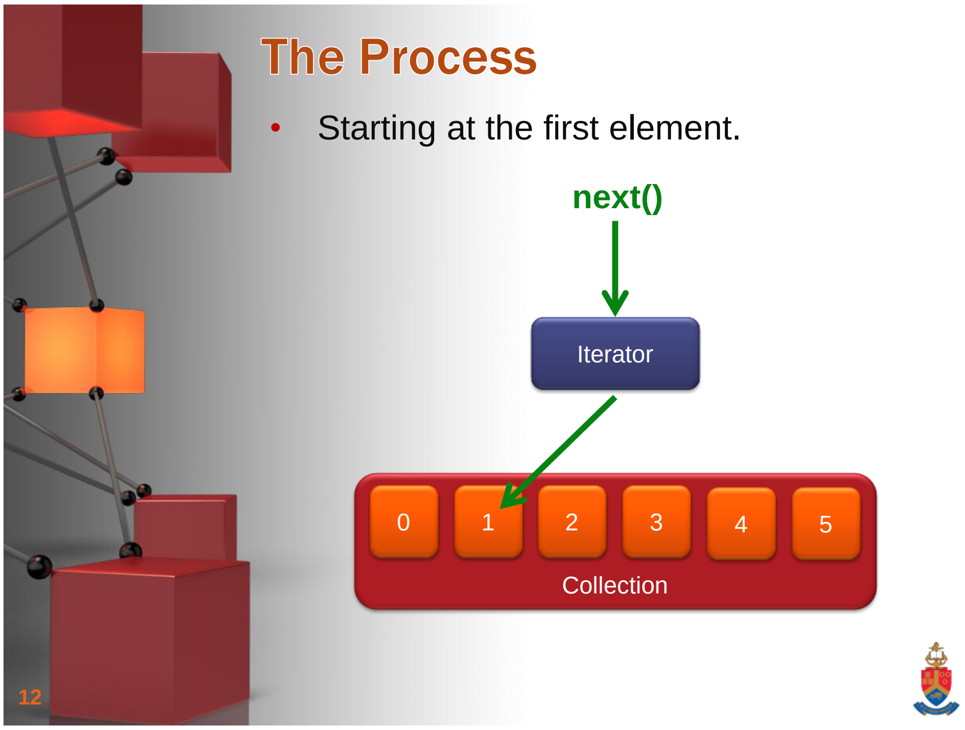
- Create an Iterator.



The Process

- Starting at the first element.

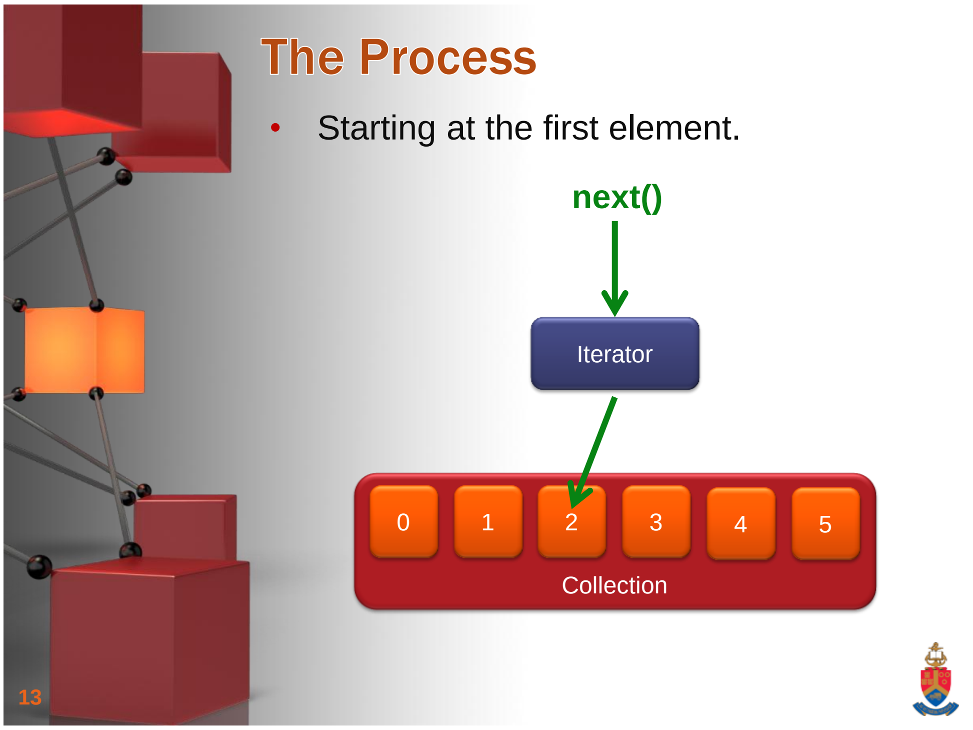
11



The Process

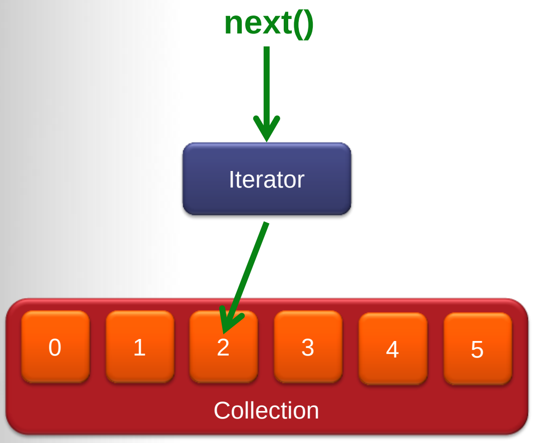
- Starting at the first element.

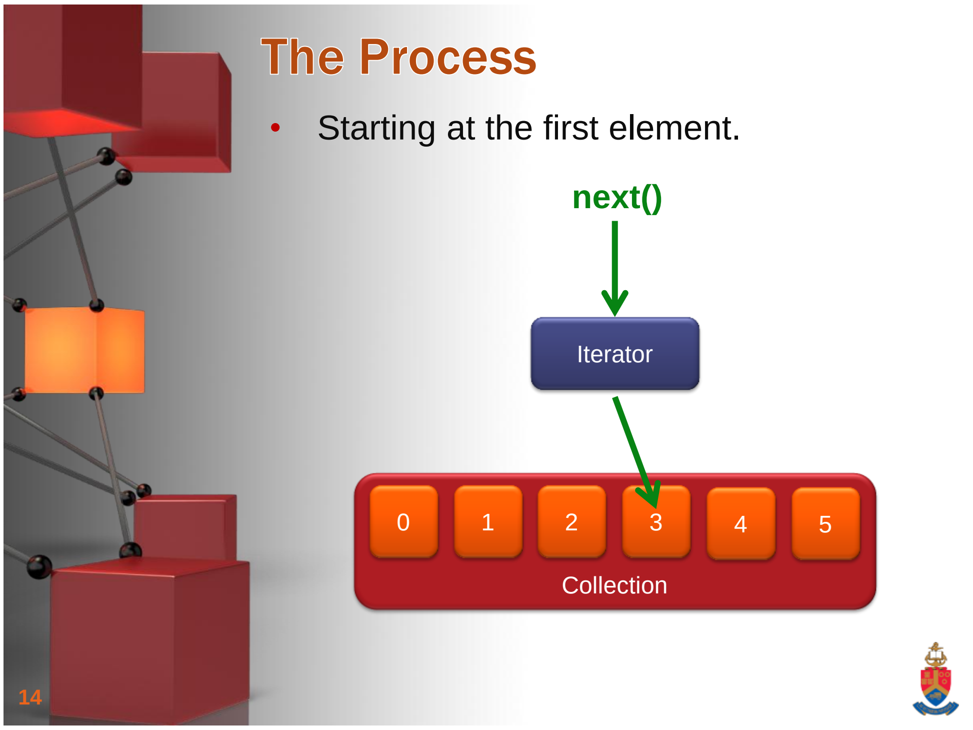
12



The Process

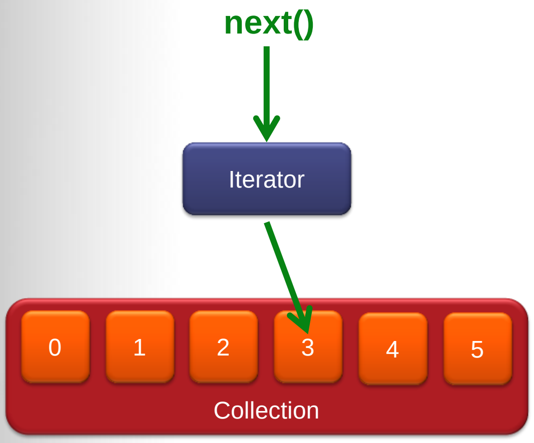
- Starting at the first element.

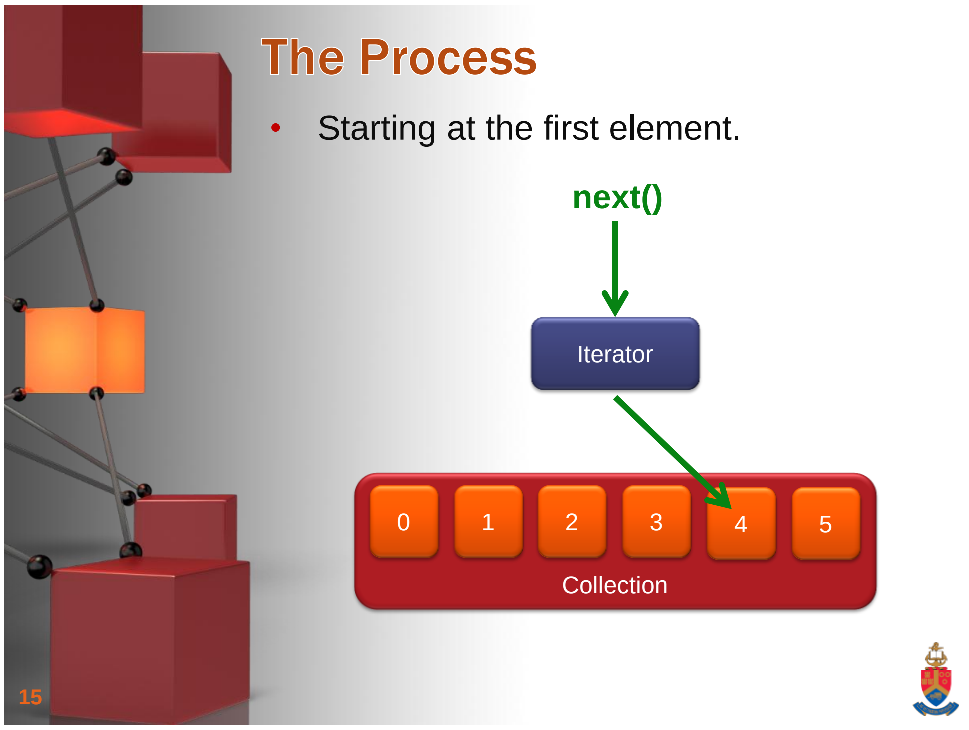




The Process

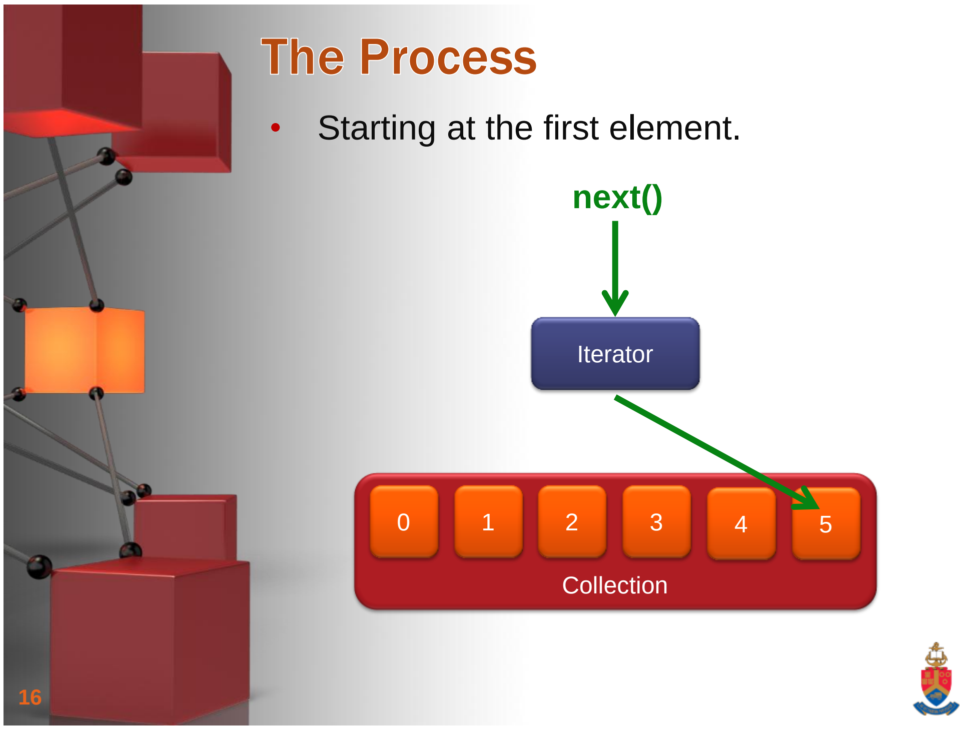
- Starting at the first element.





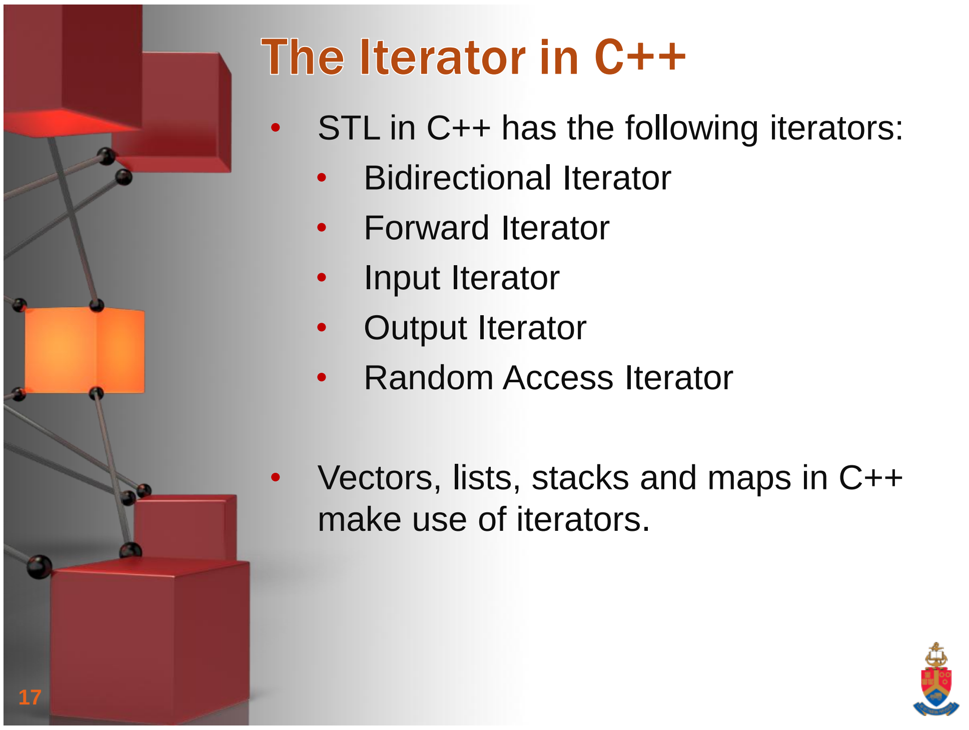
The Process

- Starting at the first element.



The Process

- Starting at the first element.

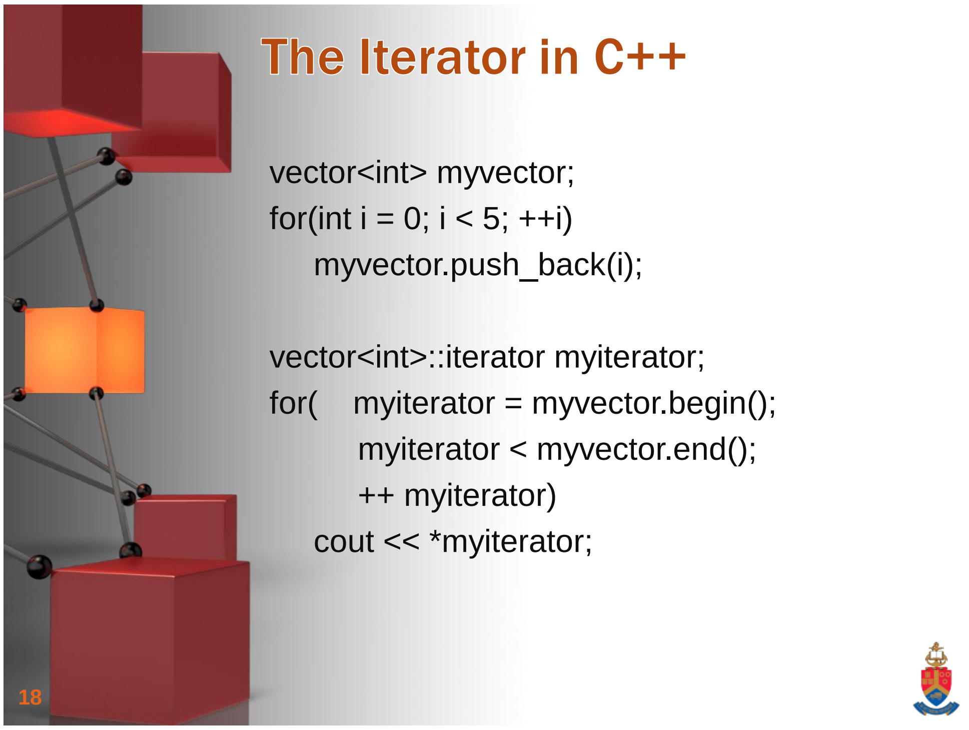


The Iterator in C++

- STL in C++ has the following iterators:
  - Bidirectional Iterator
  - Forward Iterator
  - Input Iterator
  - Output Iterator
  - Random Access Iterator

- Vectors, lists, stacks and maps in C++ make use of iterators.

17

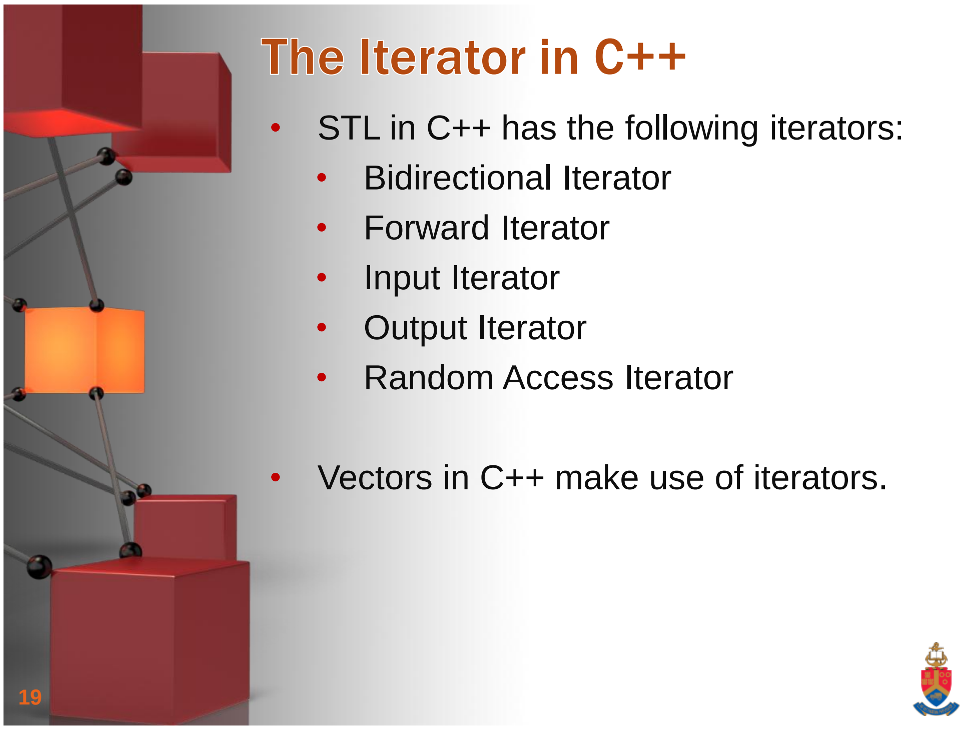


The Iterator in C++

```
vector<int> myvector;
for(int i = 0; i < 5; ++i)
    myvector.push_back(i);

vector<int>::iterator myiterator;
for(    myiterator = myvector.begin();
        myiterator < myvector.end();
        ++ myiterator)
    cout << *myiterator;
```

18

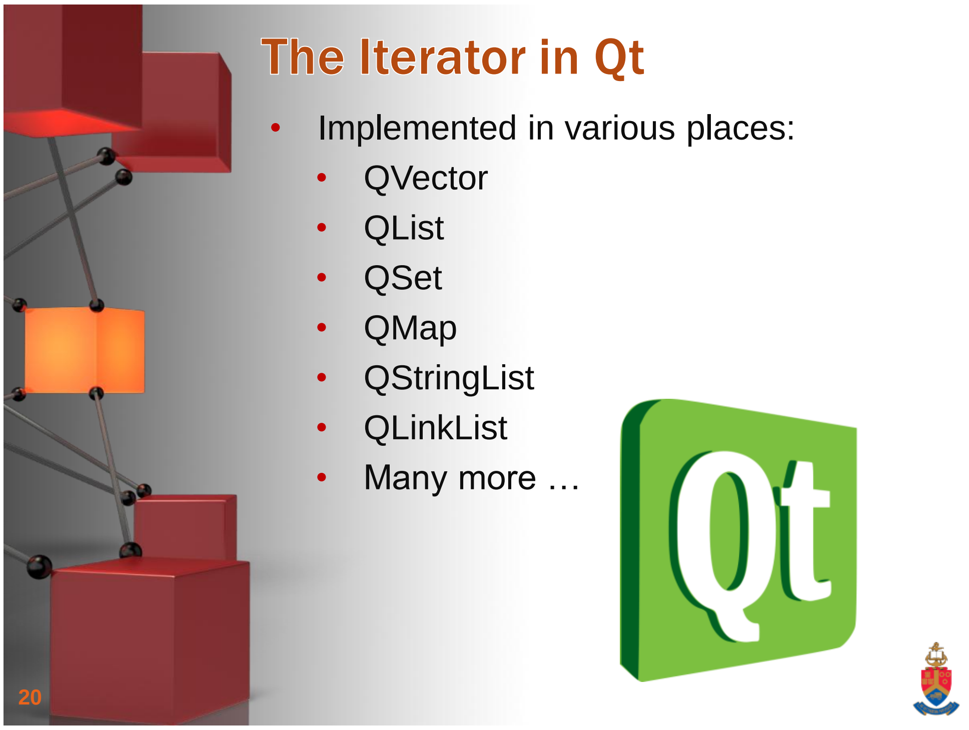


The Iterator in C++

- STL in C++ has the following iterators:
  - Bidirectional Iterator
  - Forward Iterator
  - Input Iterator
  - Output Iterator
  - Random Access Iterator

- Vectors in C++ make use of iterators.

19



The Iterator in Qt

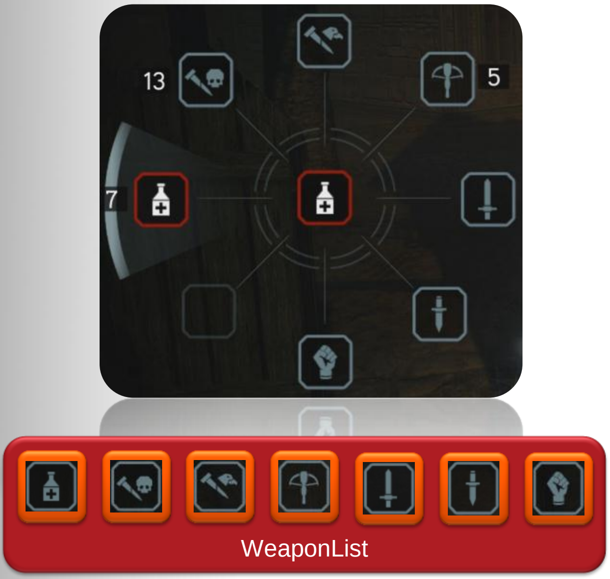
- Implemented in various places:
  - QVector
  - QList
  - QSet
  - QMap
  - QStringList
  - QLinkList
  - Many more …

20

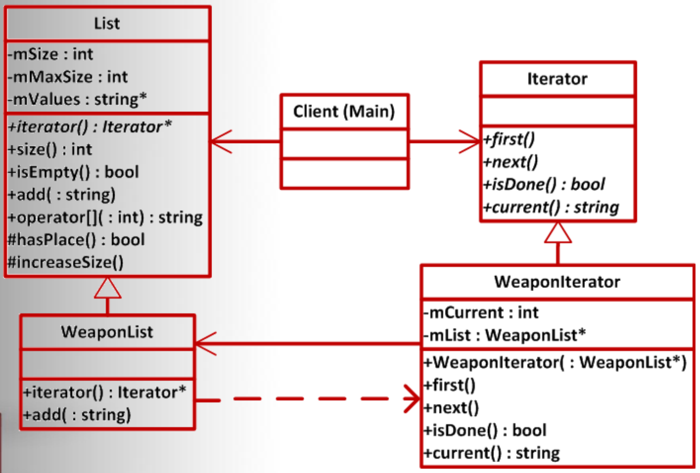
Example – Video

- http://youtu.be/nuS591k75NY

Example – Layout



Example – UML

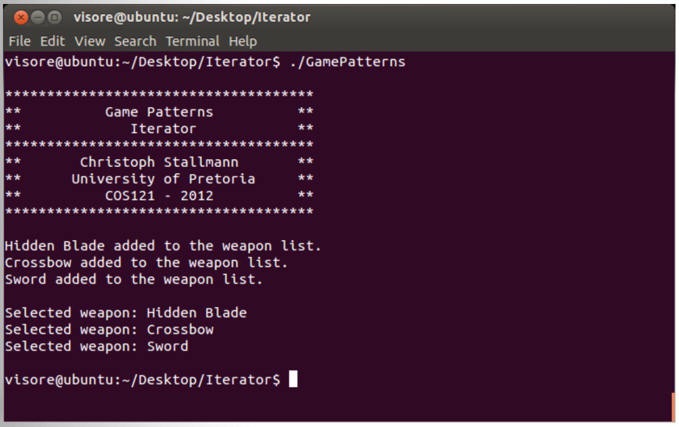


23

Example – Code

24

Example – Output


visore@ubuntu: ~/Desktop/Iterator
File Edit View Search Terminal Help

```
visore@ubuntu:~/Desktop/Iterator$ ./GamePatterns

*************************************
**          Game Patterns          **
**             Iterator            **
*************************************
**       Christoph Stallmann       **
**      University of Pretoria     **
**          COS121 - 2012          **
*************************************

Hidden Blade added to the weapon list.
Crossbow added to the weapon list.
Sword added to the weapon list.

Selected weapon: Hidden Blade
Selected weapon: Crossbow
Selected weapon: Sword

visore@ubuntu:~/Desktop/Iterator$
```

25

Improvements Achieved

- Iterators simplify the aggregate interface.
- Iterators contribute to the flexibility of your code.
  - Easy to change the iterator if the container changes.
- Iterators contribute to the reuse of your code.
  - Same iterator for different containers.
- Easy to iterate differently through the same structure.
- Execute simultaneous yet independent iterations.

26

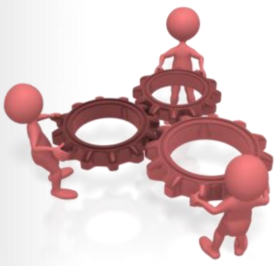
Problems

- Complicated to synchronize an Aggregate with its Iterator.
- Depending on the implementations, iterators might be slower than direct subscript access.

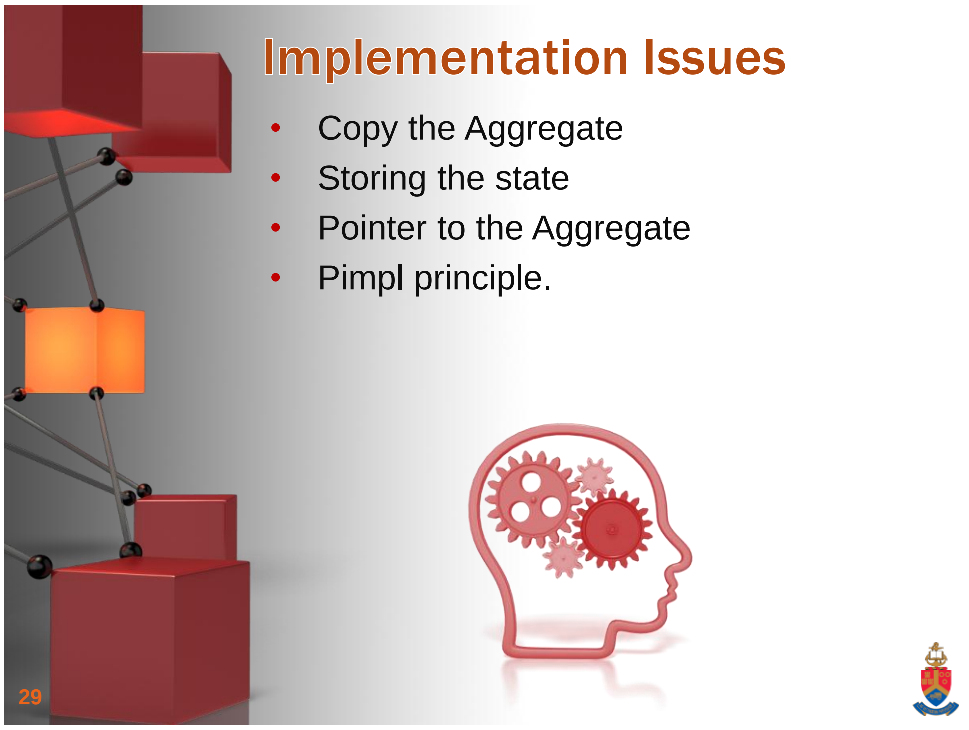
27

Different Implementations

- The Iterator implementation might differ considerably:
  - Some might be optimized for sequential access.
  - Other might be optimized for random access.



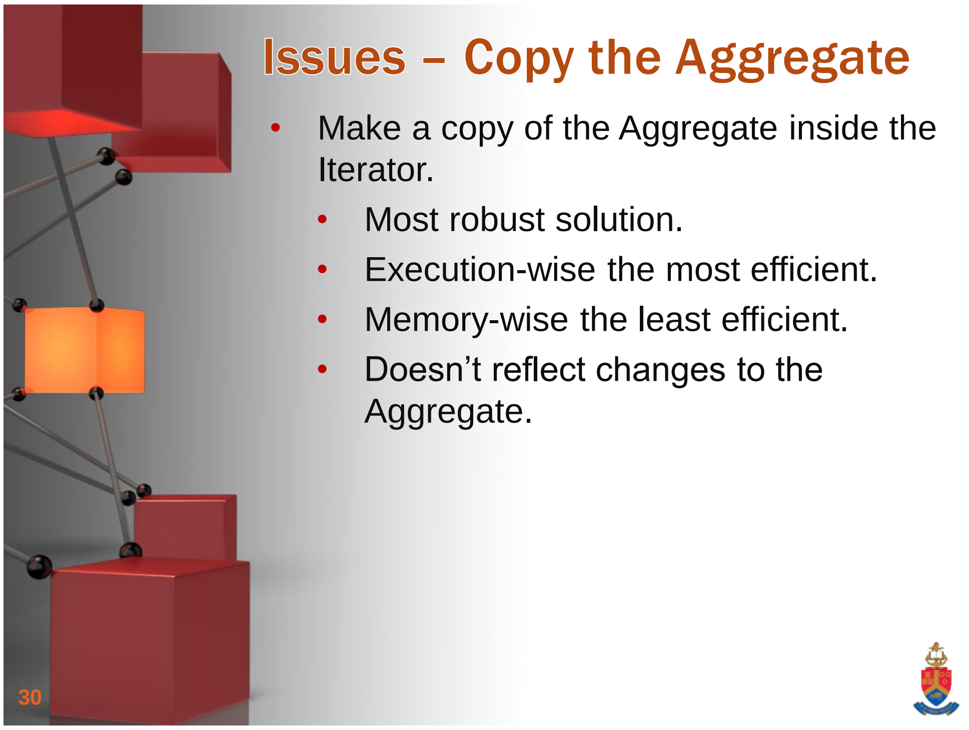
28



Implementation Issues

- Copy the Aggregate
- Storing the state
- Pointer to the Aggregate
- Pimpl principle.

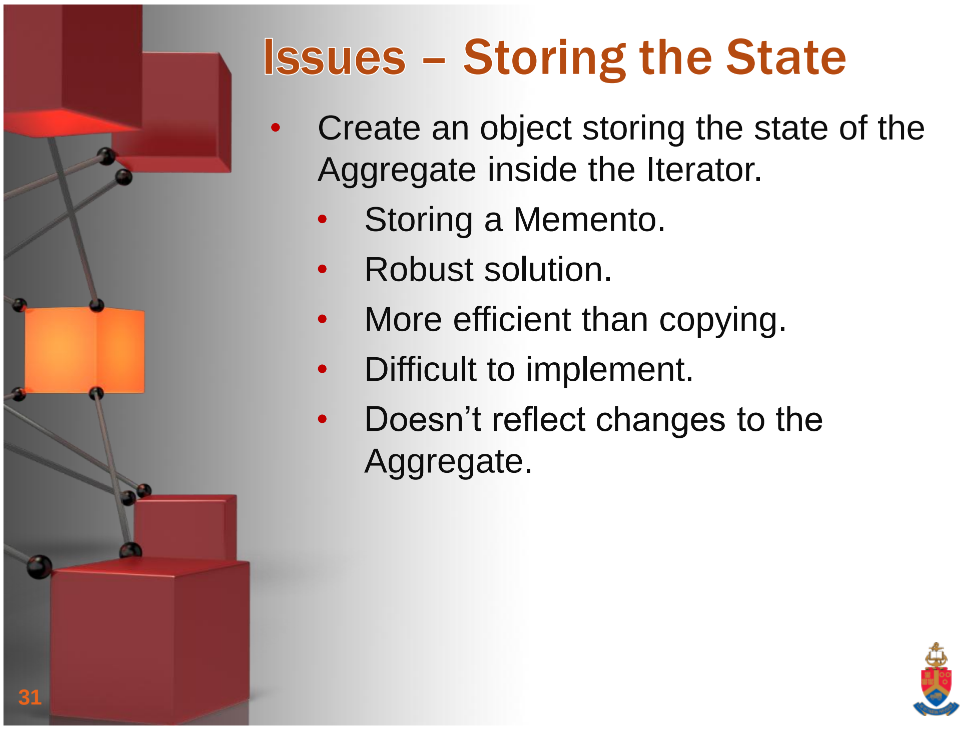
29



Issues – Copy the Aggregate

- Make a copy of the Aggregate inside the Iterator.
  - Most robust solution.
  - Execution-wise the most efficient.
  - Memory-wise the least efficient.
  - Doesn’t reflect changes to the Aggregate.

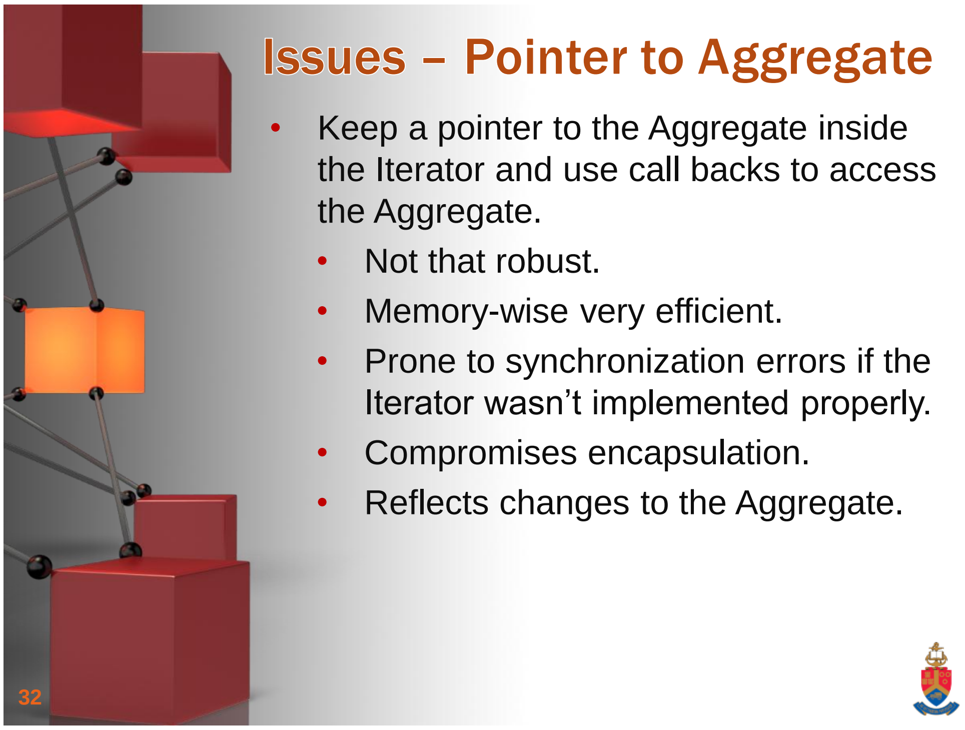
30



Issues – Storing the State

- Create an object storing the state of the Aggregate inside the Iterator.
  - Storing a Memento.
  - Robust solution.
  - More efficient than copying.
  - Difficult to implement.
  - Doesn't reflect changes to the Aggregate.

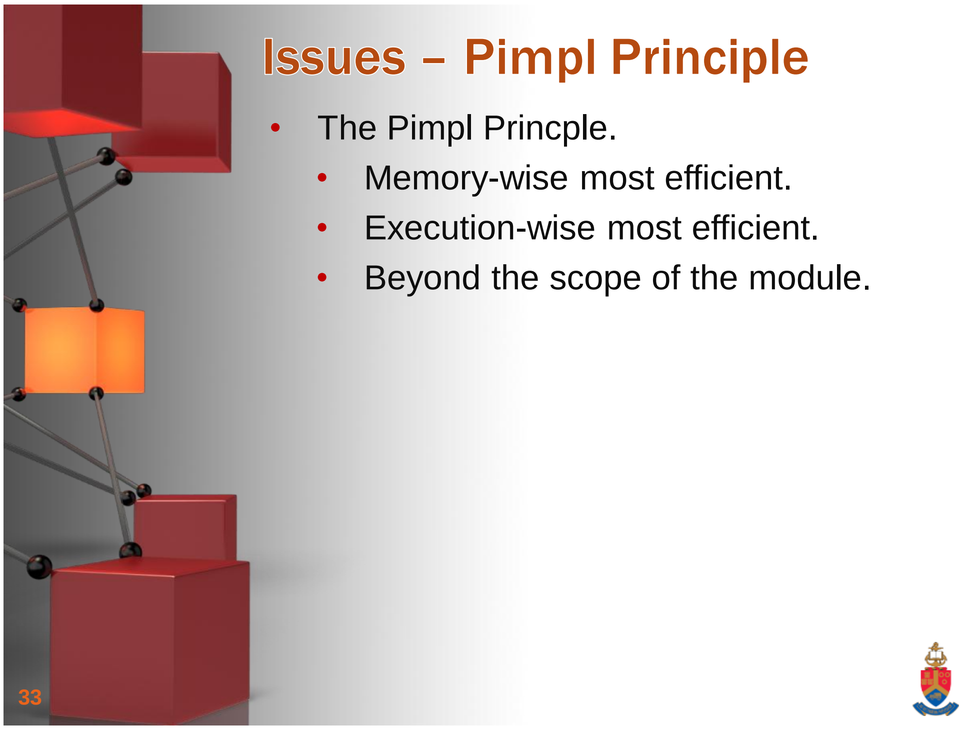
31



Issues – Pointer to Aggregate

- Keep a pointer to the Aggregate inside the Iterator and use call backs to access the Aggregate.
  - Not that robust.
  - Memory-wise very efficient.
  - Prone to synchronization errors if the Iterator wasn't implemented properly.
  - Compromises encapsulation.
  - Reflects changes to the Aggregate.

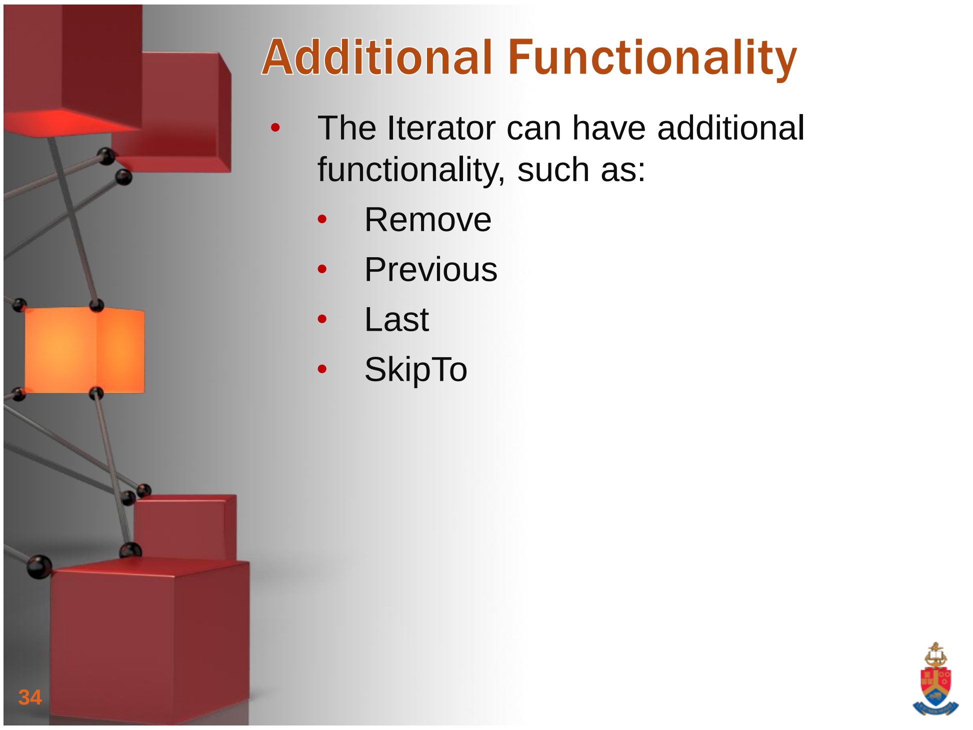
32



Issues – Pimpl Principle

- The Pimpl Princple.
  - Memory-wise most efficient.
  - Execution-wise most efficient.
  - Beyond the scope of the module.

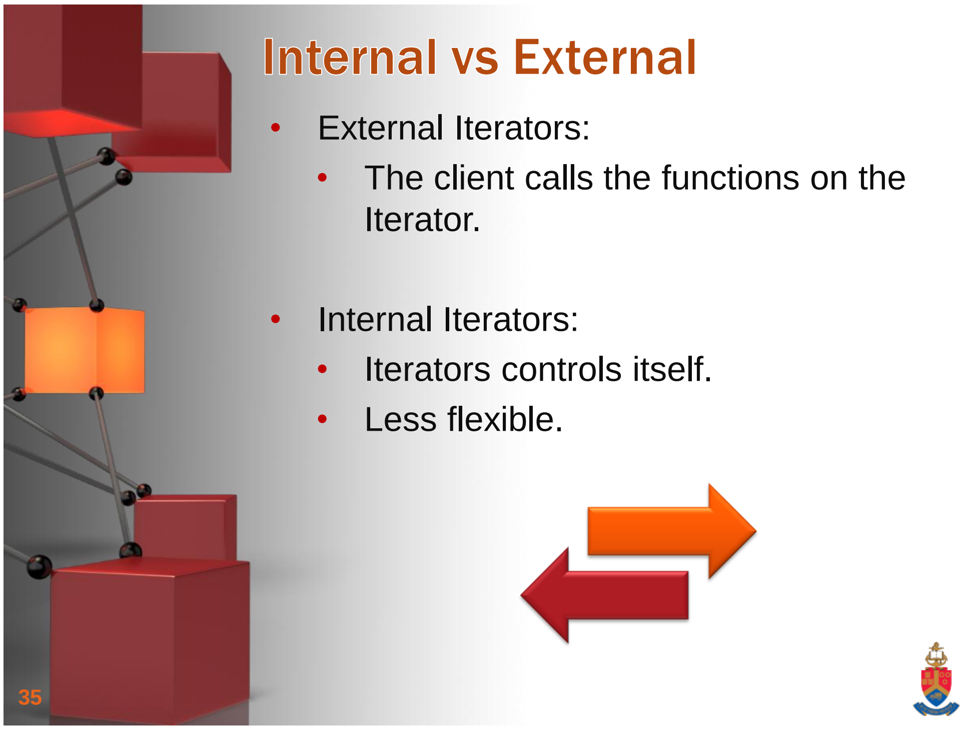
33



Additional Functionality

- The Iterator can have additional functionality, such as:
  - Remove
  - Previous
  - Last
  - SkipTo

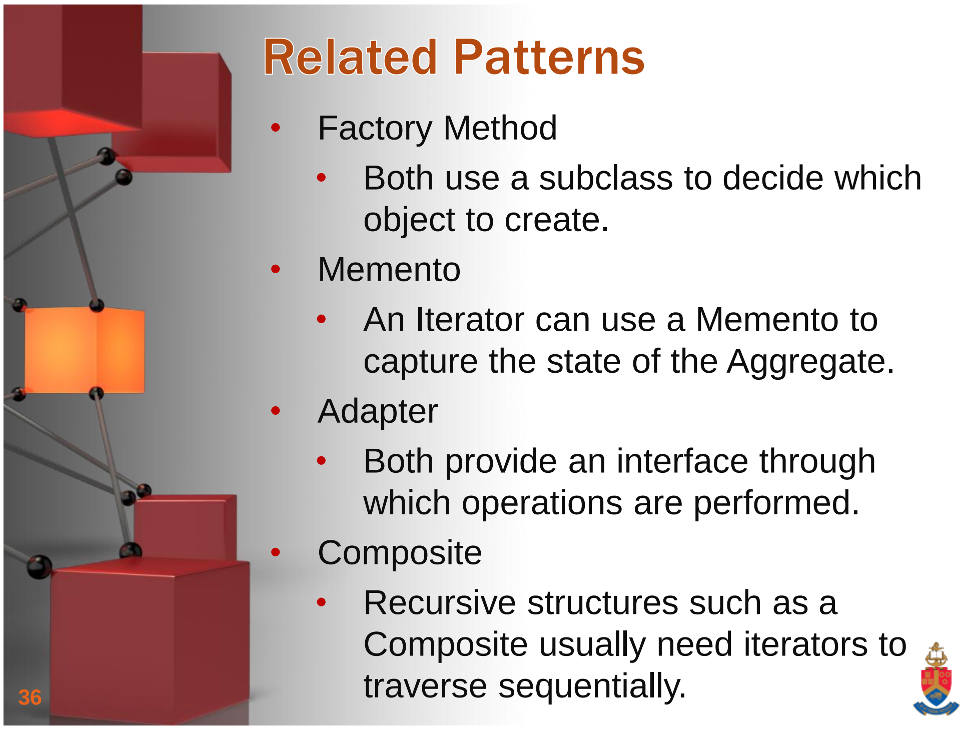
34



Internal vs External

- External Iterators:
  - The client calls the functions on the Iterator.
- Internal Iterators:
  - Iterators controls itself.
  - Less flexible.

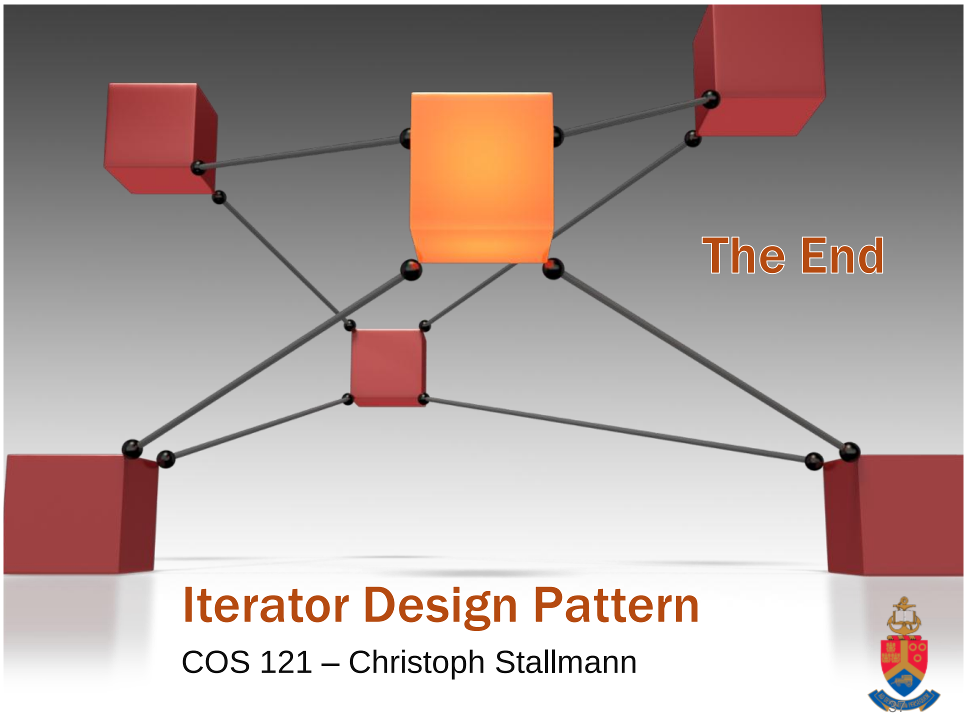
35



Related Patterns

- Factory Method
  - Both use a subclass to decide which object to create.
- Memento
  - An Iterator can use a Memento to capture the state of the Aggregate.
- Adapter
  - Both provide an interface through which operations are performed.
- Composite
  - Recursive structures such as a Composite usually need iterators to traverse sequentially.

36



The End

Iterator Design Pattern

COS 121 – Christoph Stallmann