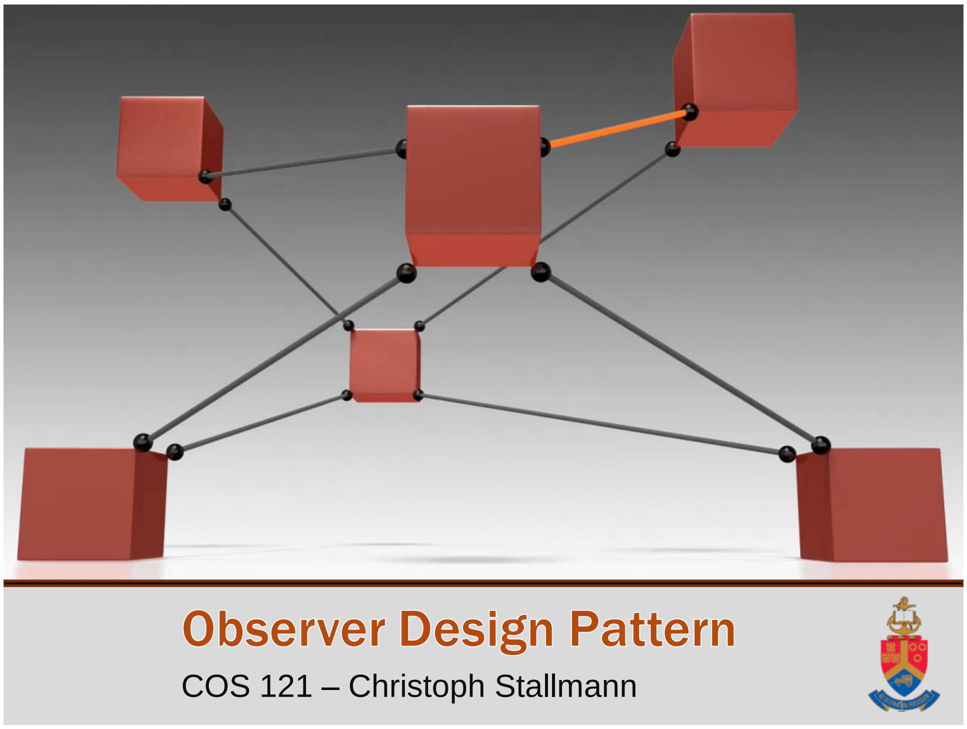
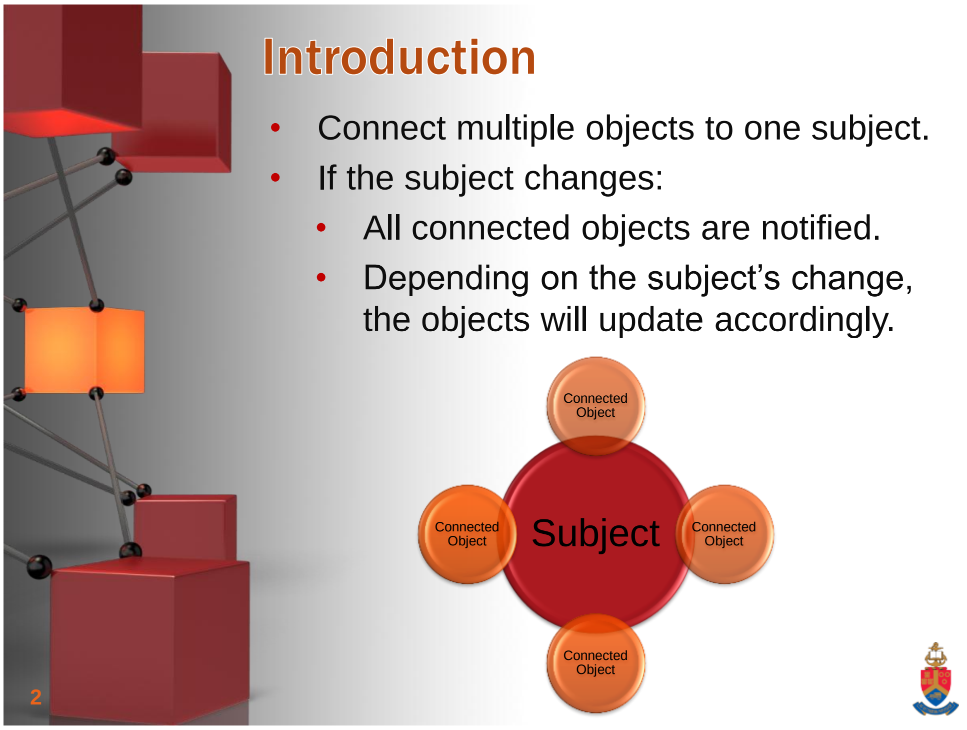


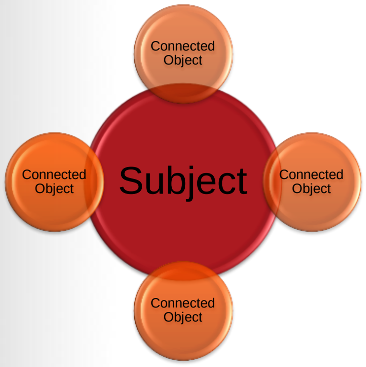
Observer Design Pattern

COS 121 – Christoph Stallmann

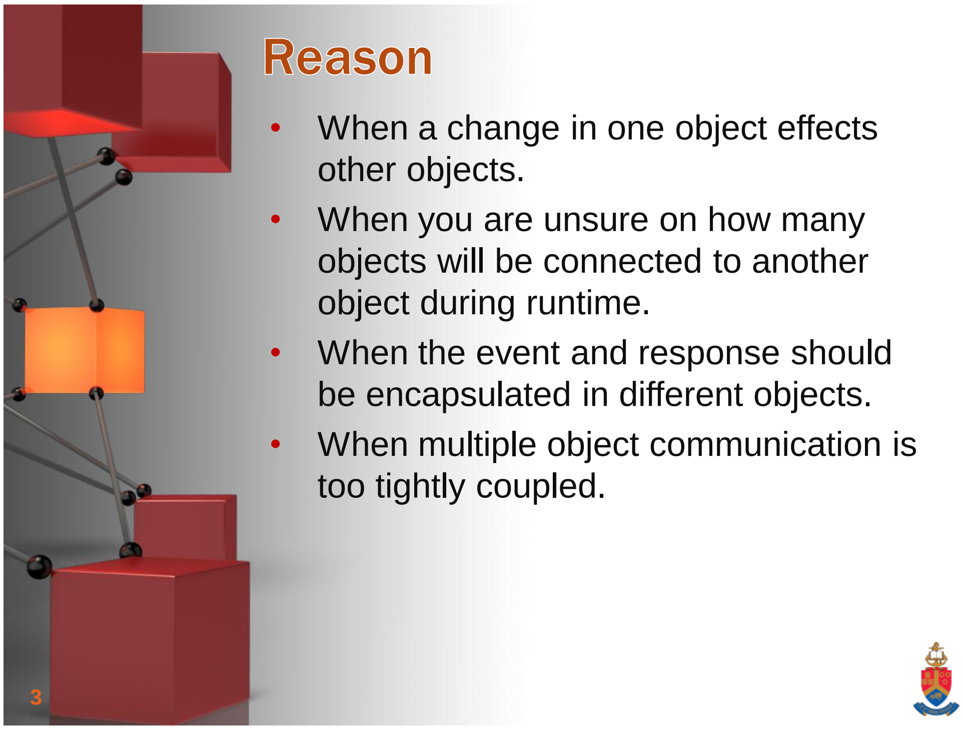


Introduction

- Connect multiple objects to one subject.
- If the subject changes:
  - All connected objects are notified.
  - Depending on the subject's change, the objects will update accordingly.



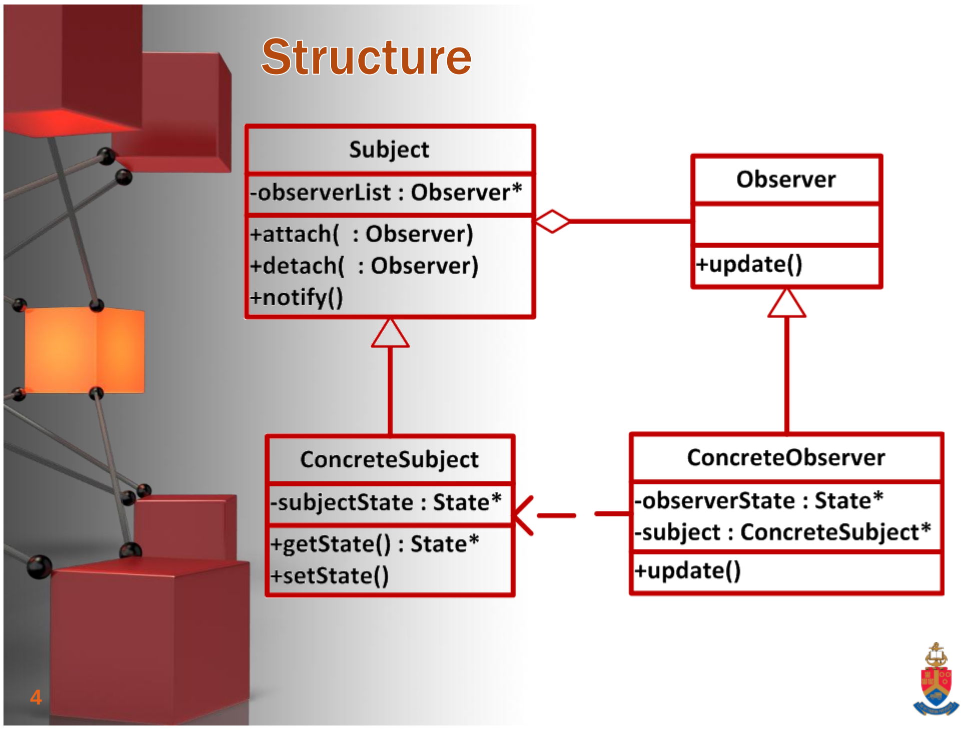
2



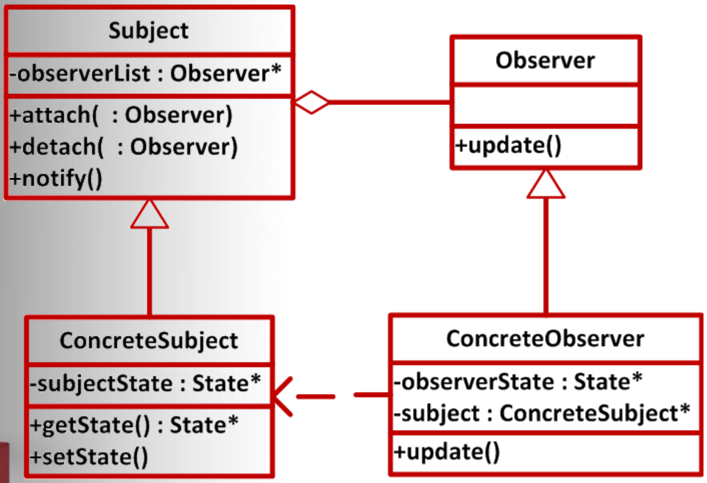
Reason

- When a change in one object effects other objects.
- When you are unsure on how many objects will be connected to another object during runtime.
- When the event and response should be encapsulated in different objects.
- When multiple object communication is too tightly coupled.

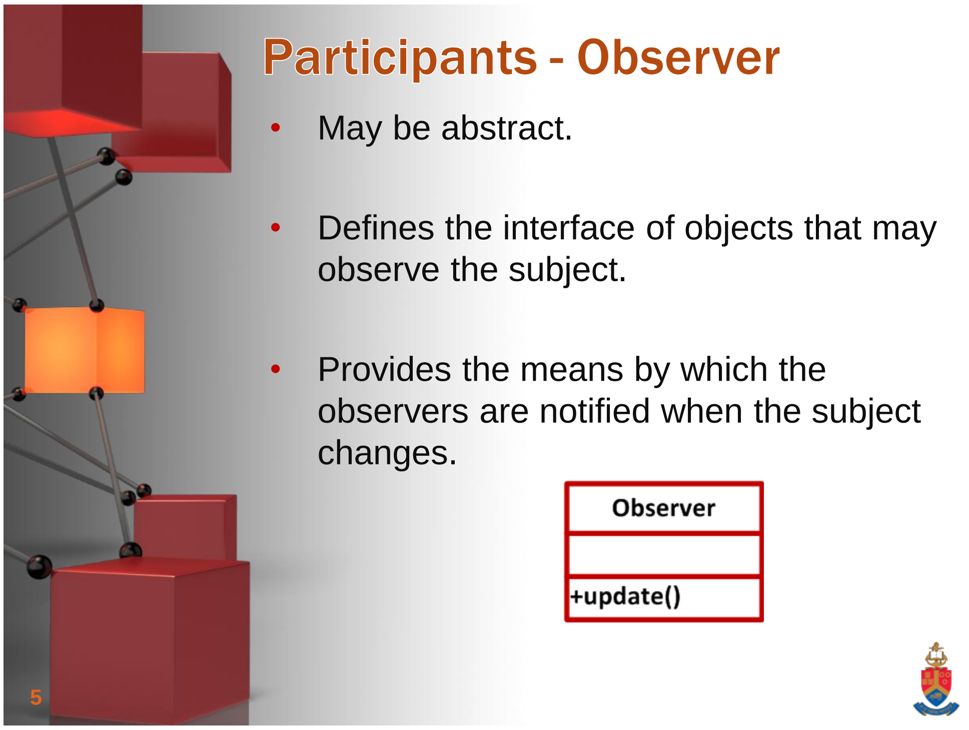
3



Structure



4

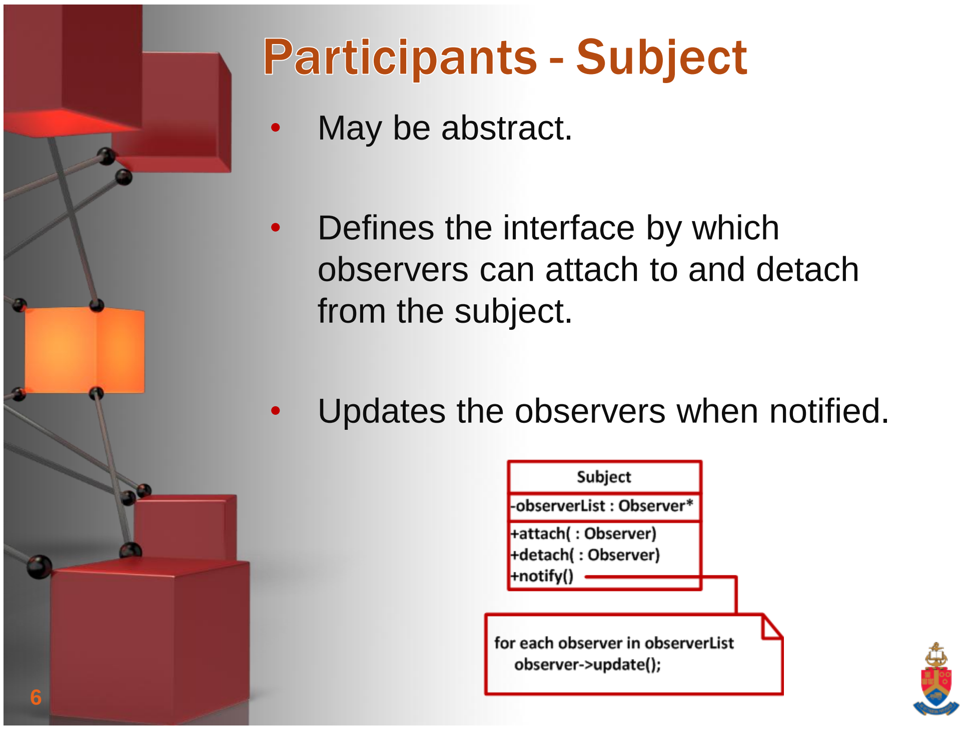


Participants - Observer

- May be abstract.
- Defines the interface of objects that may observe the subject.
- Provides the means by which the observers are notified when the subject changes.

Observer
+update()

5



Participants - Subject

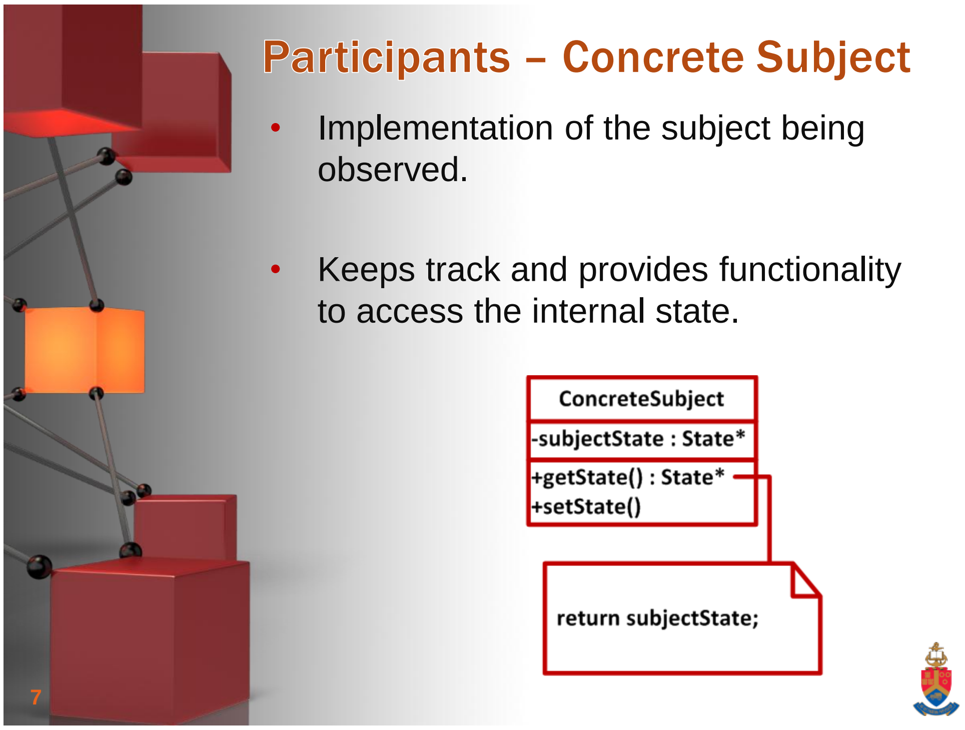
- May be abstract.
- Defines the interface by which observers can attach to and detach from the subject.
- Updates the observers when notified.

Subject
-observerList : Observer*

+attach(: Observer)
+detach(: Observer)
+notify()

```
for each observer in observerList
    observer->update();
```

6



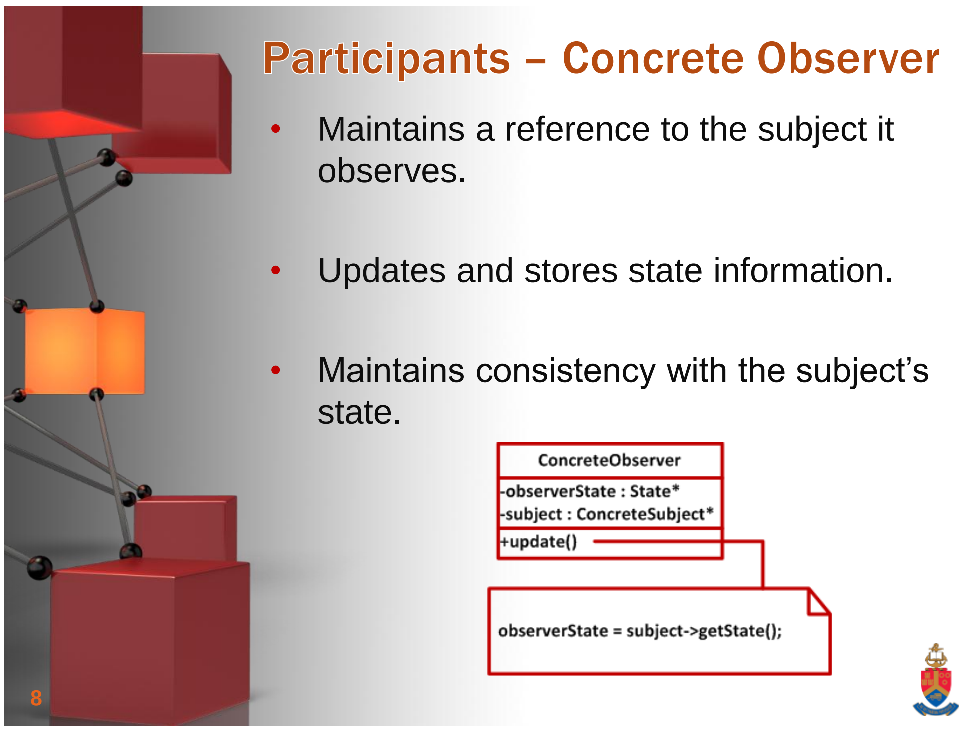
Participants – Concrete Subject

- Implementation of the subject being observed.
- Keeps track and provides functionality to access the internal state.

```
ConcreteSubject
-subjectState : State*
+getState() : State*
+setState()
```

```
return subjectState;
```

7



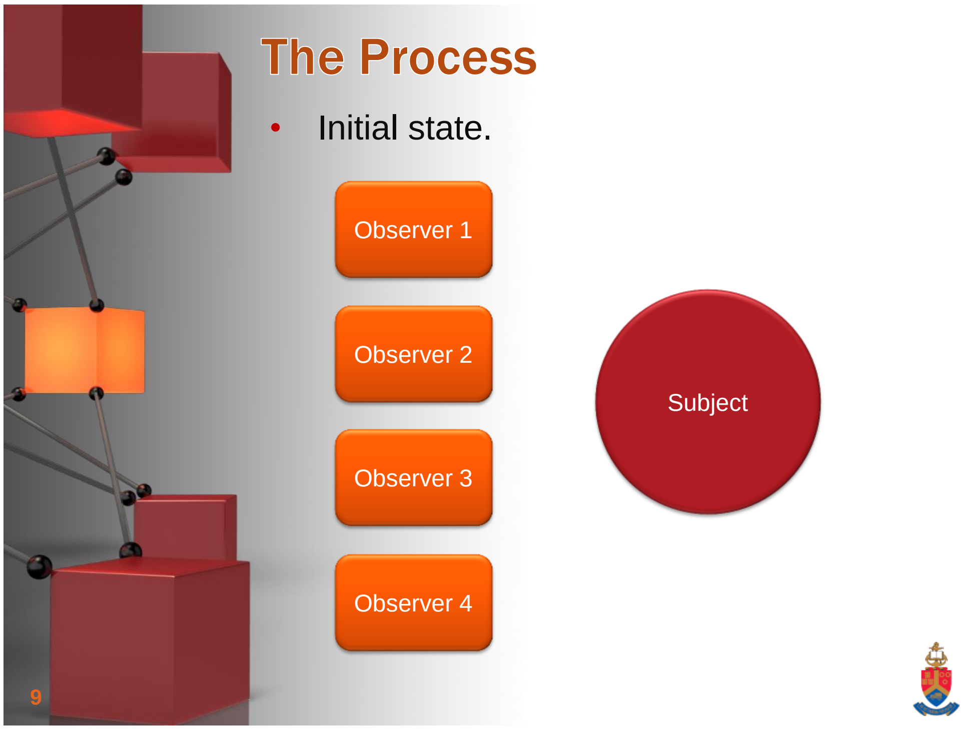
Participants – Concrete Observer

- Maintains a reference to the subject it observes.
- Updates and stores state information.
- Maintains consistency with the subject's state.

```
ConcreteObserver
-observerState : State*
-subject : ConcreteSubject*
+update()
```

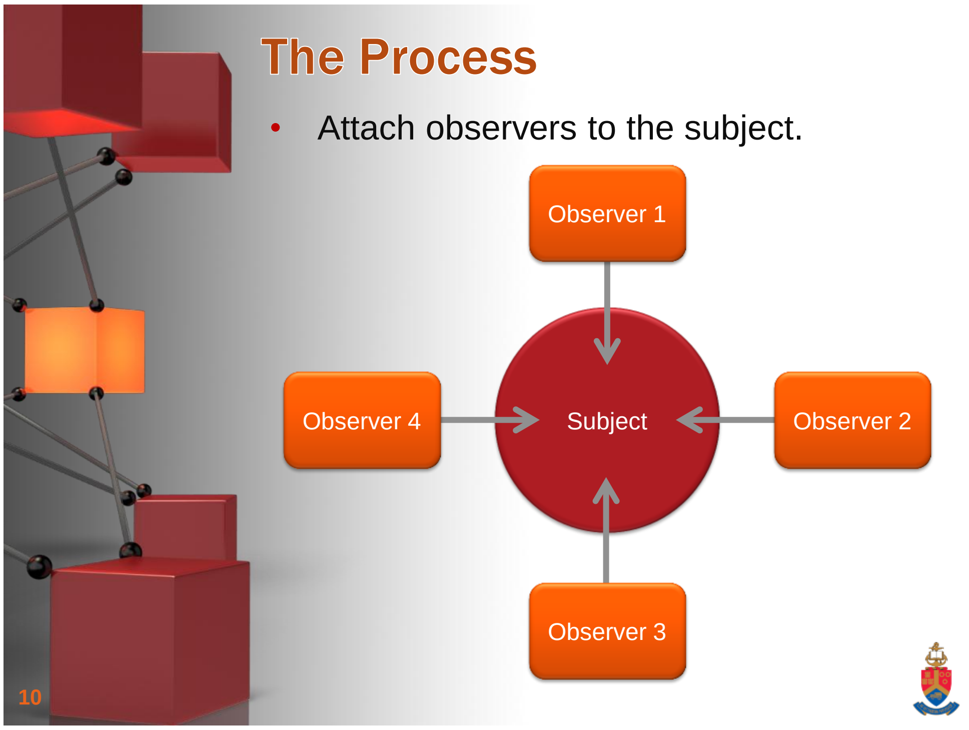
```
observerState = subject->getState();
```

8



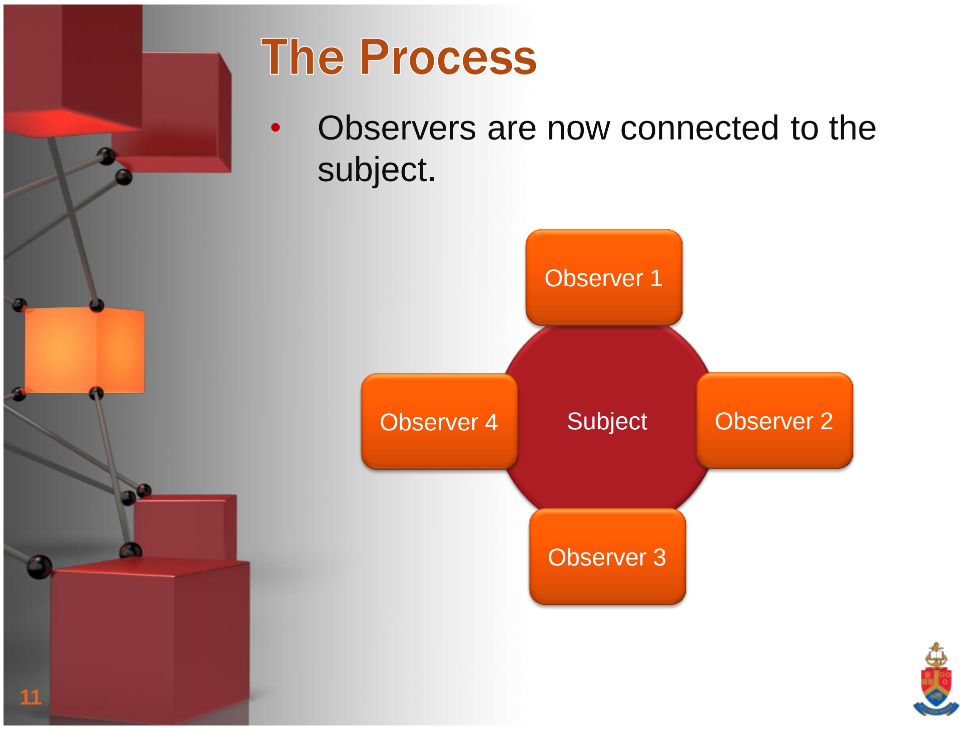
The Process

- Initial state.



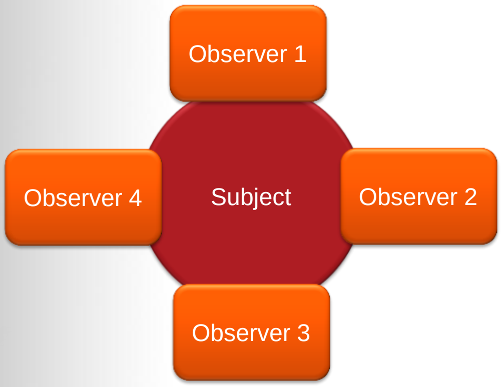
The Process

- Attach observers to the subject.

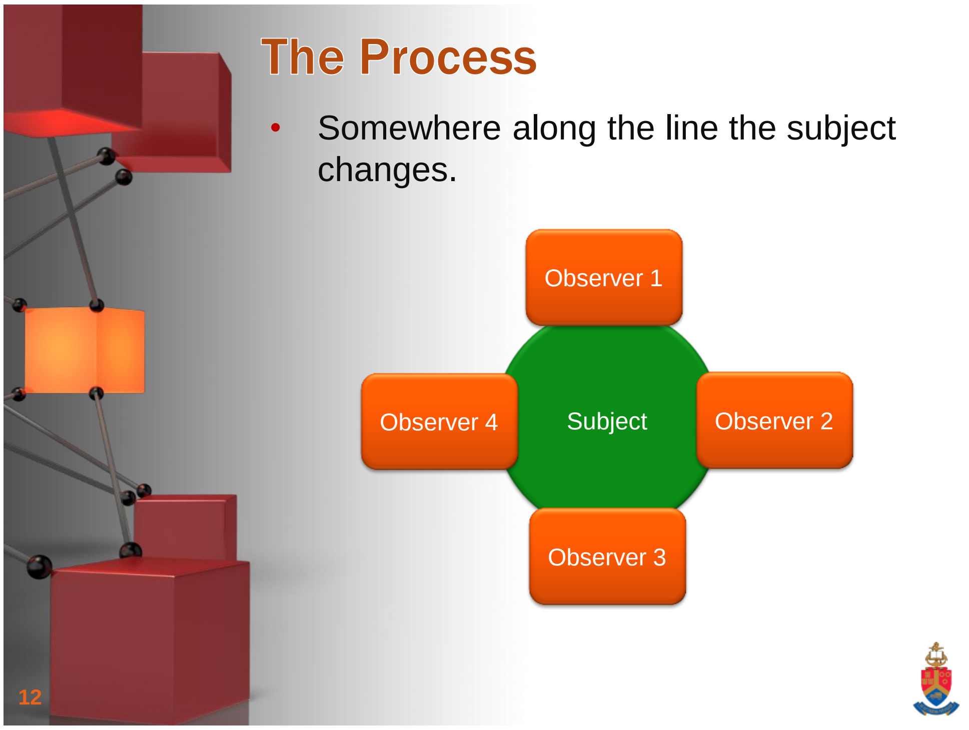


The Process

- Observers are now connected to the subject.

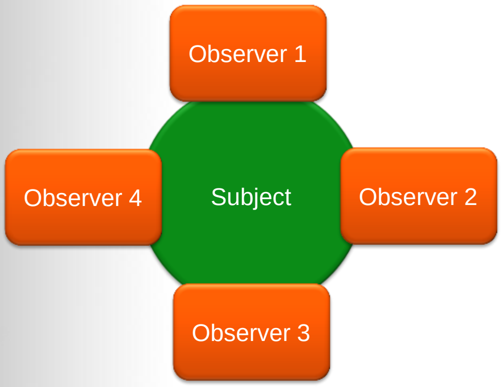


11

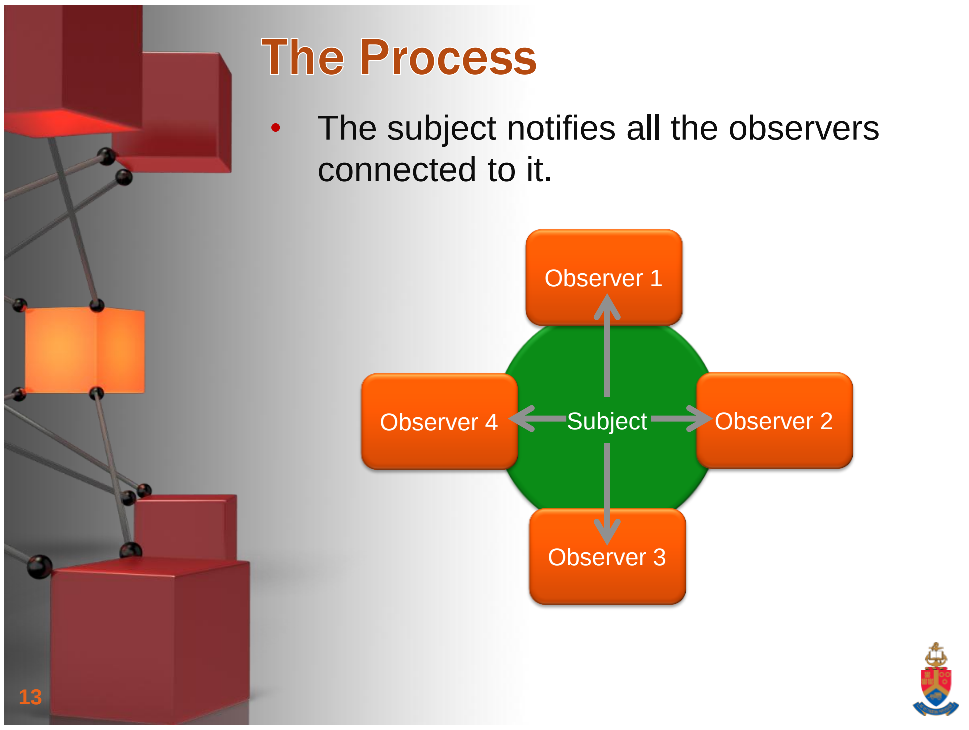


The Process

- Somewhere along the line the subject changes.



12

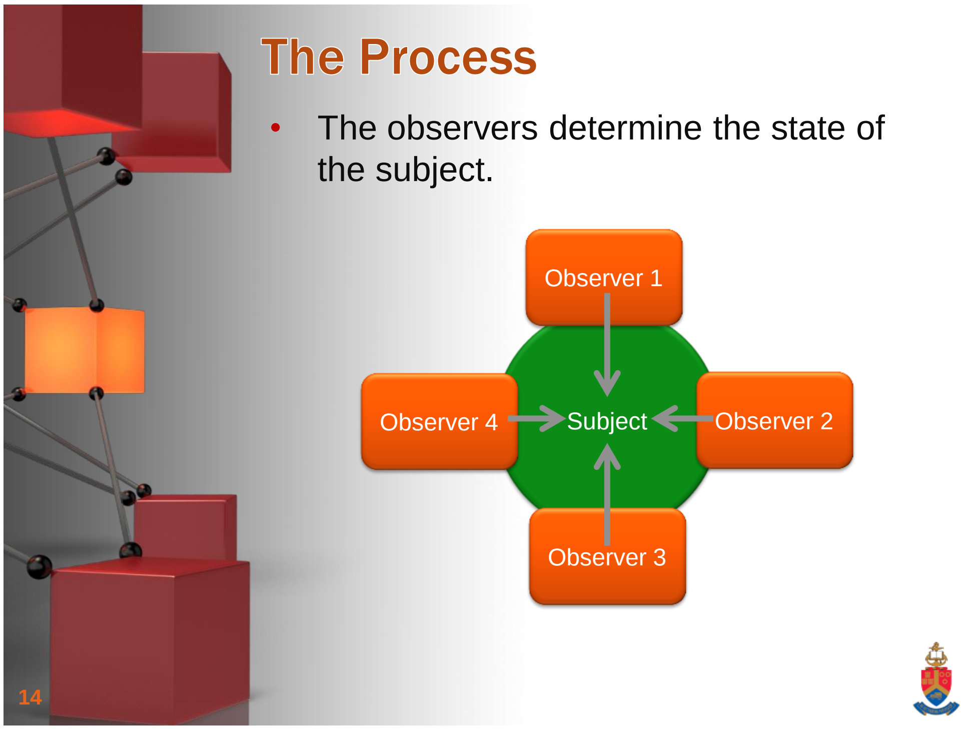


The Process

- The subject notifies all the observers connected to it.

$$\begin{tikzcd}
 & \text{Observer 1} & \\
\text{Observer 4} & \text{Subject} \arrow[l] \arrow[r] \arrow[u] \arrow[d] & \text{Observer 2} \\
 & \text{Observer 3} &
\end{tikzcd}$$

13

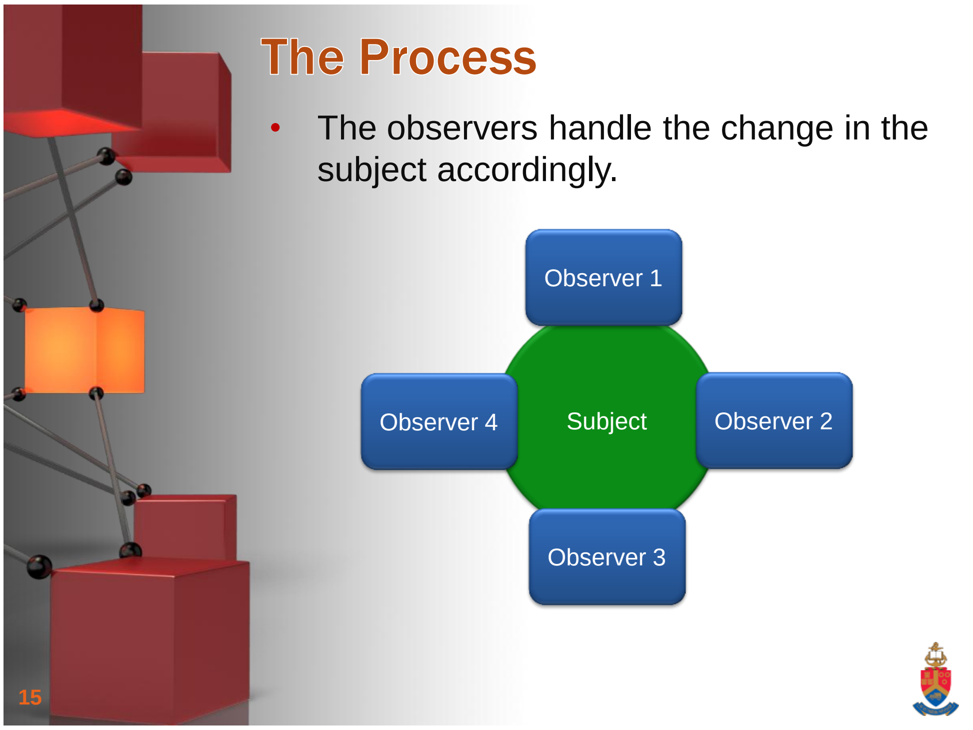


The Process

- The observers determine the state of the subject.

$$\begin{tikzcd}
 & \text{Observer 1} \arrow[d] & \\
\text{Observer 4} \arrow[r] & \text{Subject} & \text{Observer 2} \arrow[l] \\
 & \text{Observer 3} \arrow[u] &
\end{tikzcd}$$

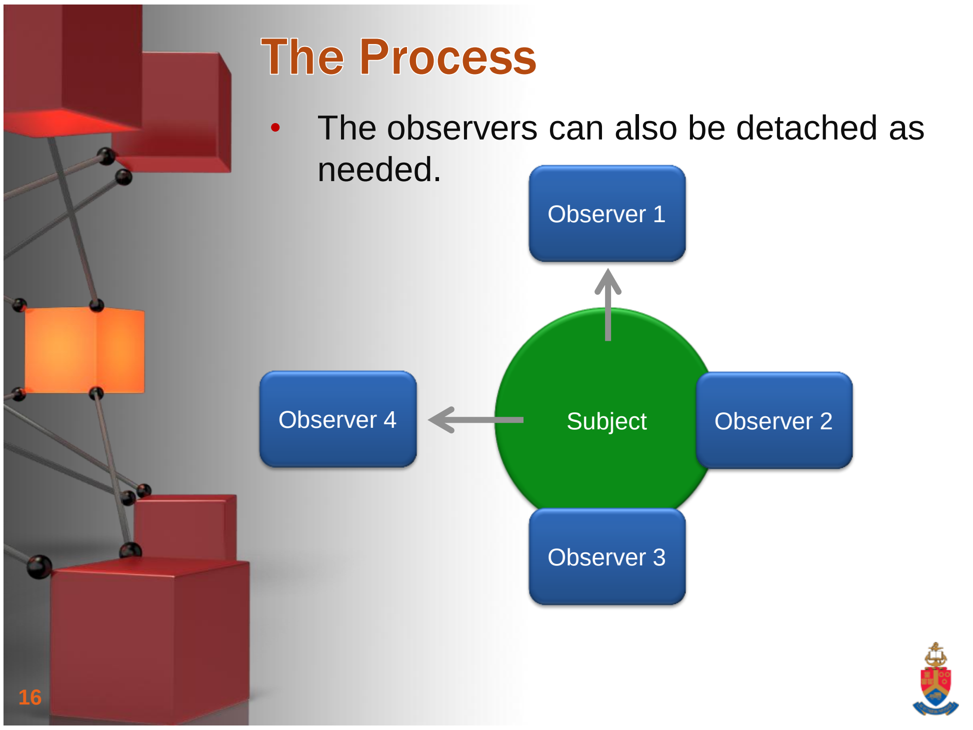
14



The Process

- The observers handle the change in the subject accordingly.

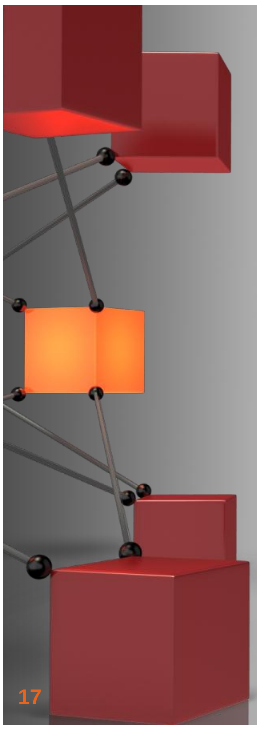
Observer 1
Observer 4
Subject
Observer 2
Observer 3
15



The Process

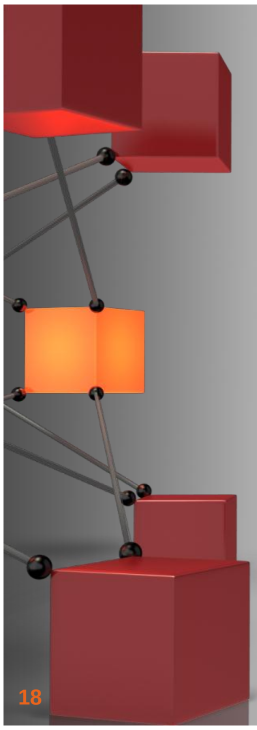
- The observers can also be detached as needed.

Observer 1
Observer 4
Subject
Observer 2
Observer 3
16



The Observer in Qt

- Implemented as Signals and Slots in Qt.
- Multiple Slots can be connected to one Signal.
- If Signal is “emitted”, all Slots are notified.



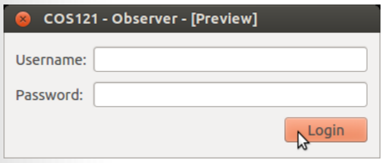
The Observer in Qt

- Attach an observer:

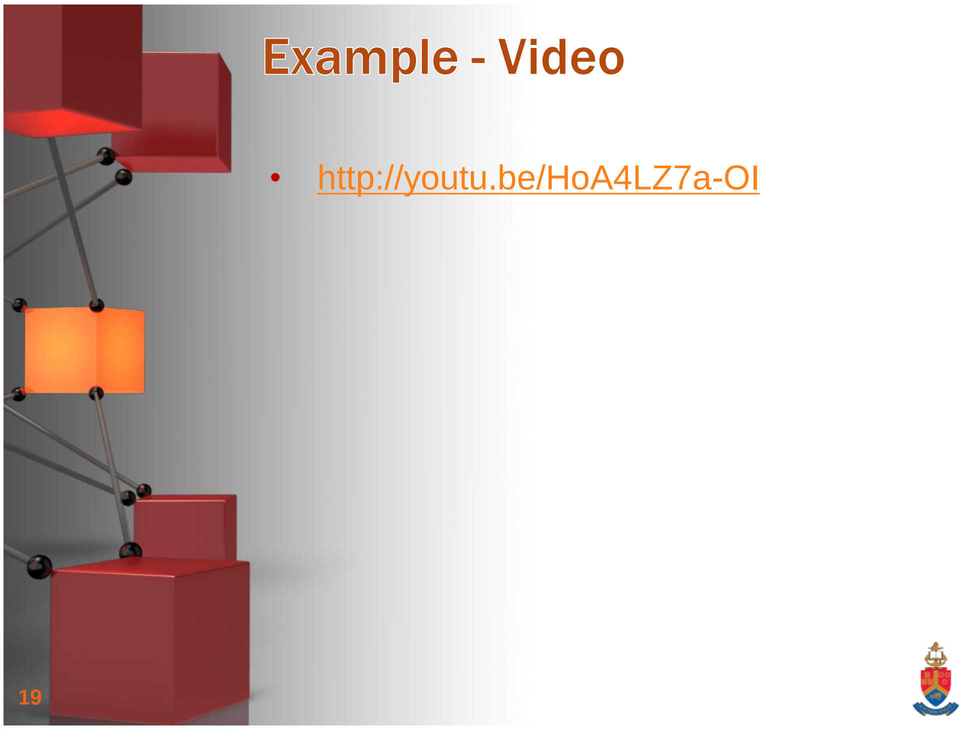
```
QObject::connect(button, SIGNAL(clicked()),
                 this, SLOT(doAction()));
```

- Detach an observer:

```
QObject::disconnect(button, SIGNAL(clicked()),
                    this, SLOT(doAction()));
```



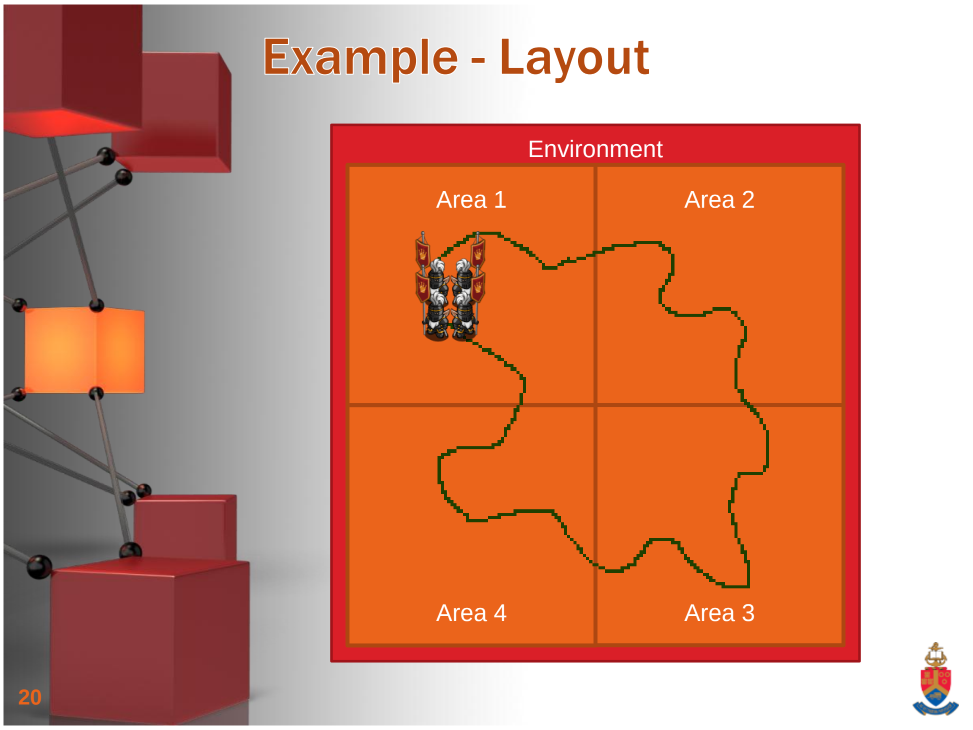
- If the button is clicked, doAction() will be executed.



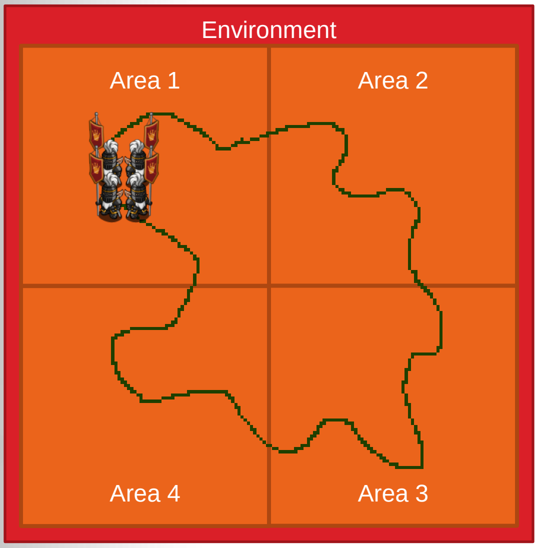
Example - Video

- http://youtu.be/HoA4LZ7a-OI

19

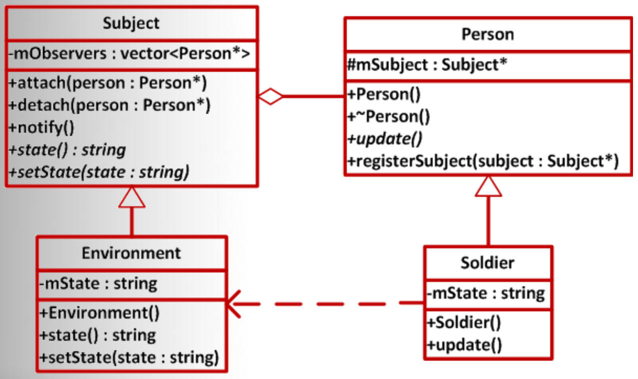


Example - Layout



20

Example - UML

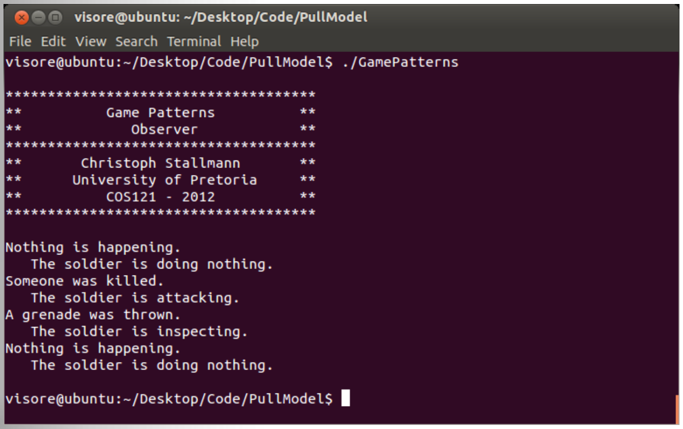


21

Example - Code

22

Example - Output


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

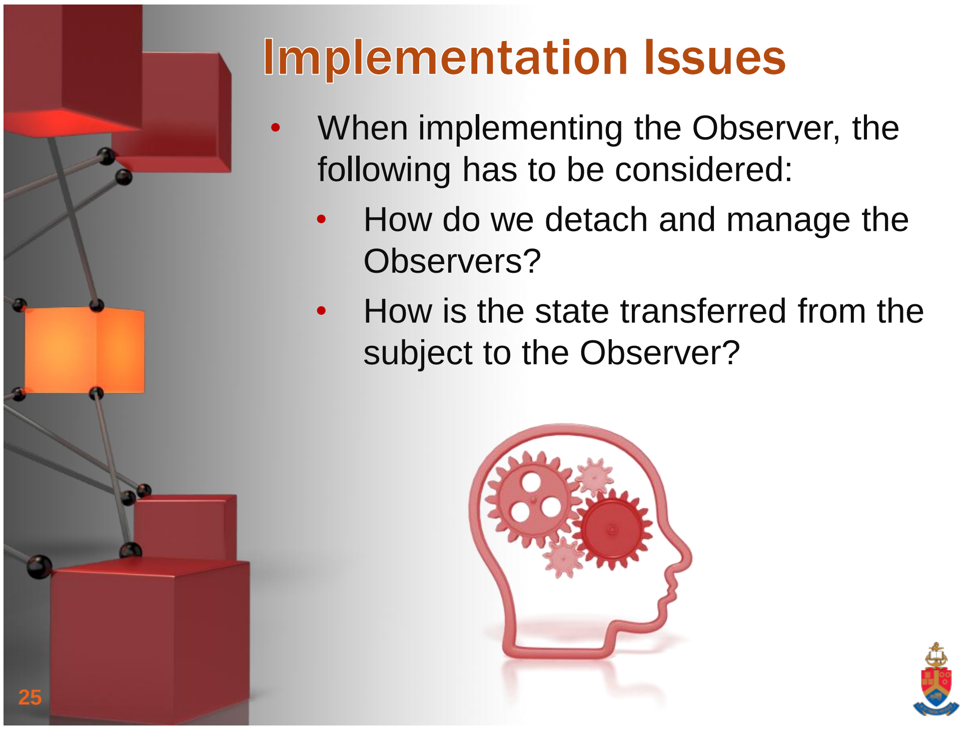
```
visore@ubuntu:~/Desktop/Code/PullModel$ ./GamePatterns

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

Nothing is happening.
   The soldier is doing nothing.
Someone was killed.
   The soldier is attacking.
A grenade was thrown.
   The soldier is inspecting.
Nothing is happening.
   The soldier is doing nothing.

visore@ubuntu:~/Desktop/Code/PullModel$ 
```

23

Improvements Achieved

- Separation of concerns:
  - Observers are not embedded into a subject.
  - Observers can register and deregister as required.
  - The state change and the event action are encapsulated.
- Elimination of “busy wait”:
  - Instead of continuously checking if the state changed, observers are notified.
  - More efficient.

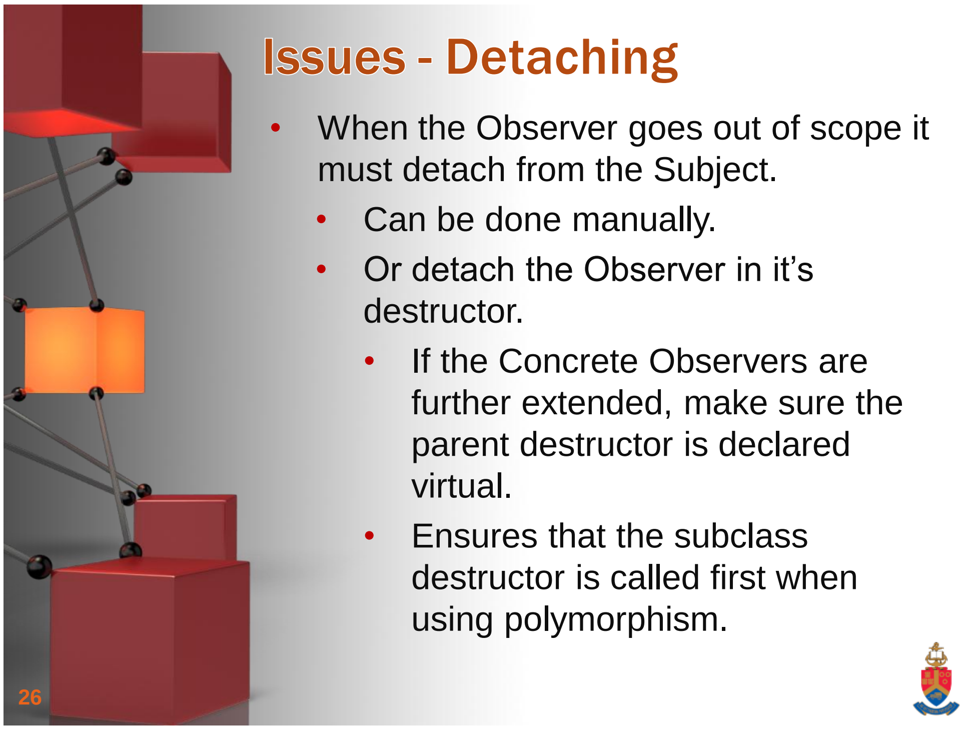
24



Implementation Issues

- When implementing the Observer, the following has to be considered:
  - How do we detach and manage the Observers?
  - How is the state transferred from the subject to the Observer?

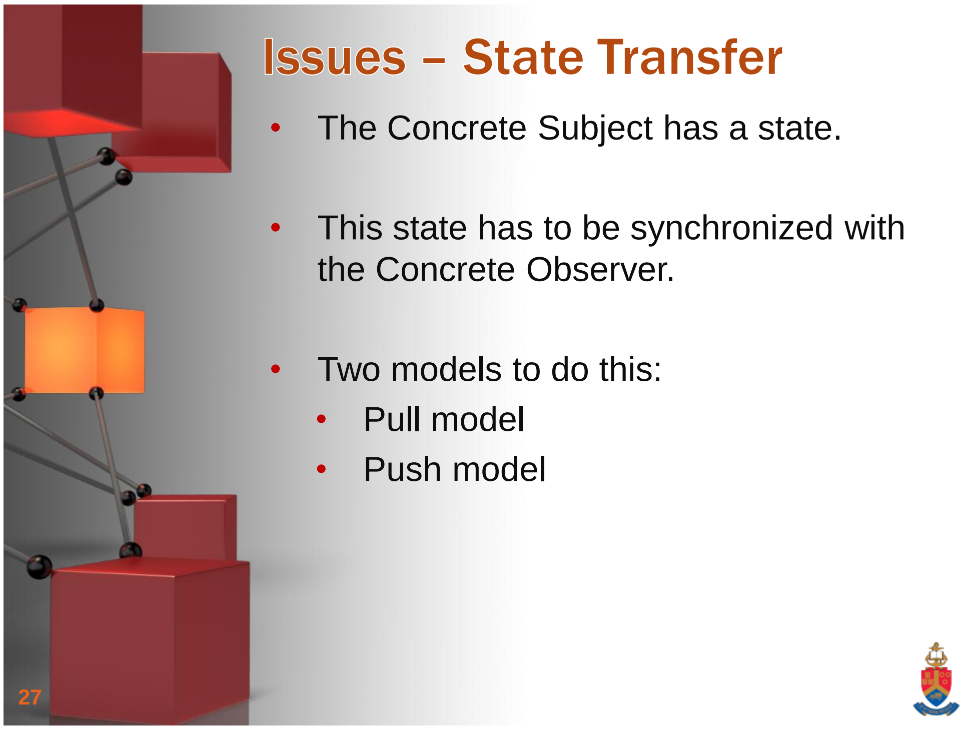
25



Issues - Detaching

- When the Observer goes out of scope it must detach from the Subject.
  - Can be done manually.
  - Or detach the Observer in it's destructor.
    - If the Concrete Observers are further extended, make sure the parent destructor is declared virtual.
    - Ensures that the subclass destructor is called first when using polymorphism.

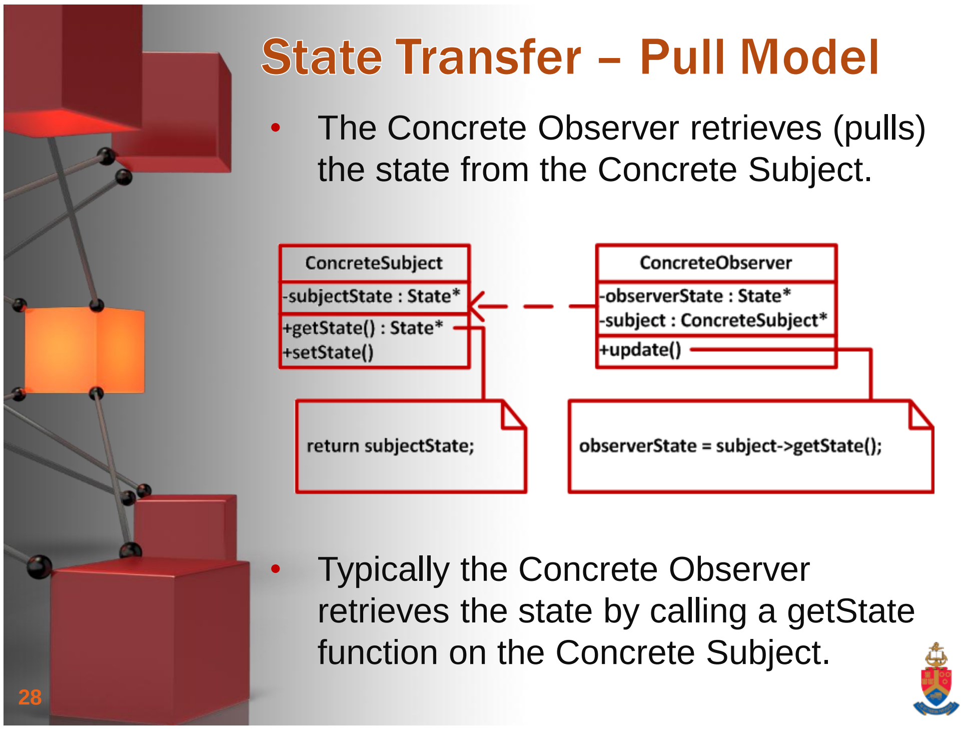
26



Issues – State Transfer

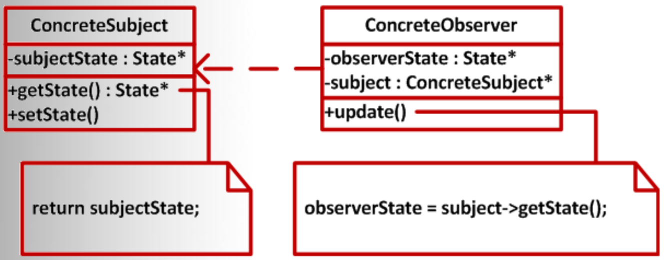
- The Concrete Subject has a state.
- This state has to be synchronized with the Concrete Observer.
- Two models to do this:
  - Pull model
  - Push model

27



State Transfer – Pull Model

- The Concrete Observer retrieves (pulls) the state from the Concrete Subject.

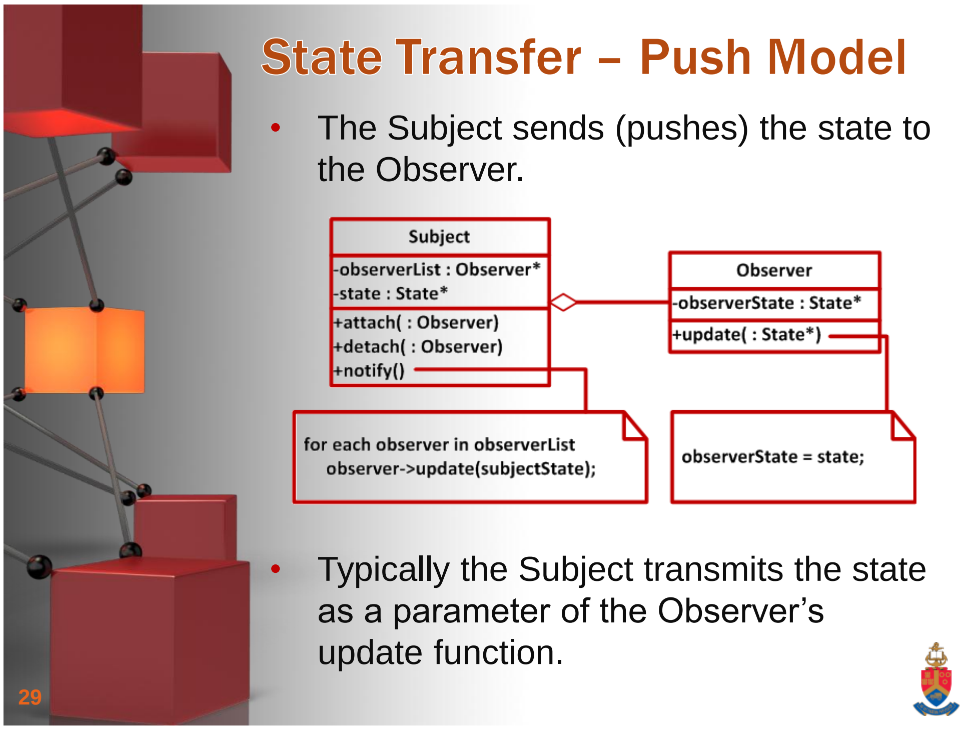


```
return subjectState;
```

```
observerState = subject->getState();
```

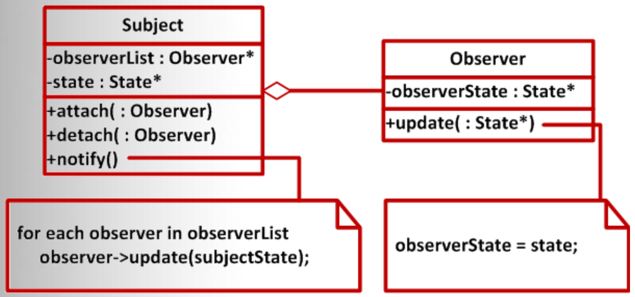
- Typically the Concrete Observer retrieves the state by calling a getState function on the Concrete Subject.

28



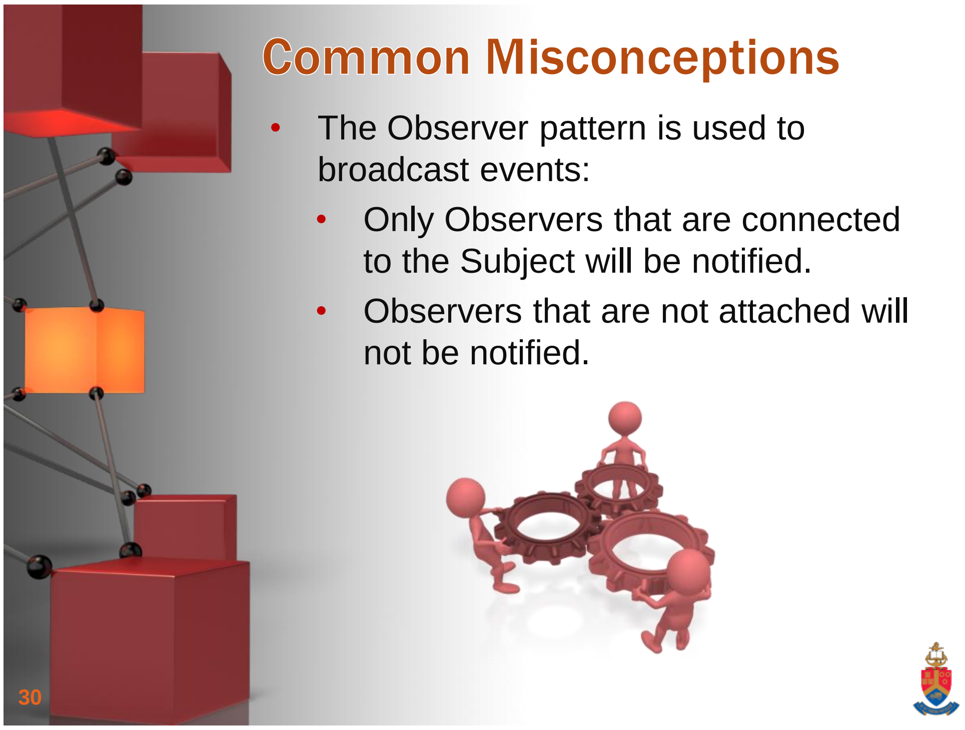
State Transfer – Push Model

- The Subject sends (pushes) the state to the Observer.



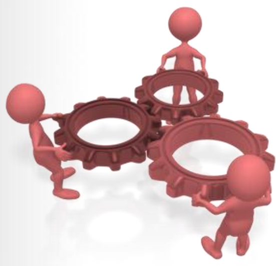
- Typically the Subject transmits the state as a parameter of the Observer's update function.

29

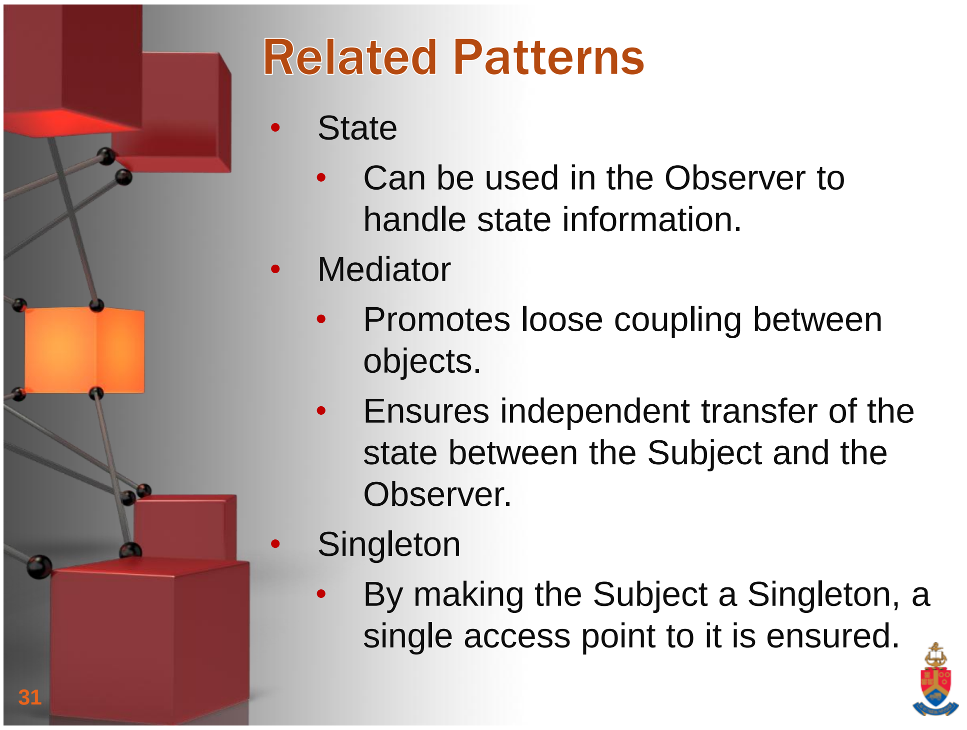


Common Misconceptions

- The Observer pattern is used to broadcast events:
  - Only Observers that are connected to the Subject will be notified.
  - Observers that are not attached will not be notified.



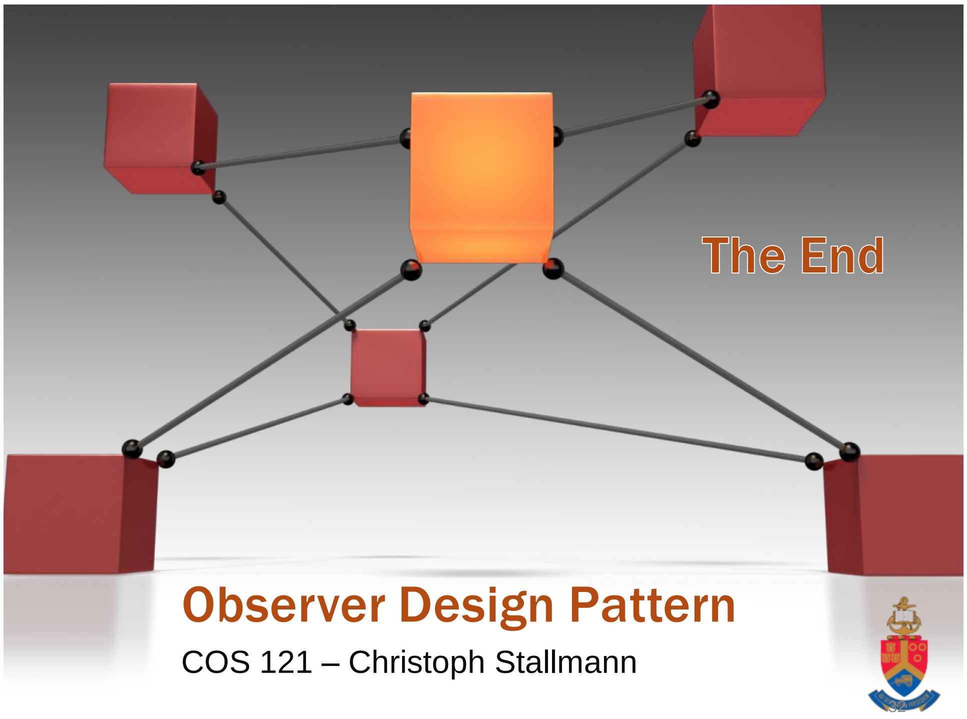
30



Related Patterns

- State
  - Can be used in the Observer to handle state information.
- Mediator
  - Promotes loose coupling between objects.
  - Ensures independent transfer of the state between the Subject and the Observer.
- Singleton
  - By making the Subject a Singleton, a single access point to it is ensured.

31



The End

Observer Design Pattern

COS 121 – Christoph Stallmann