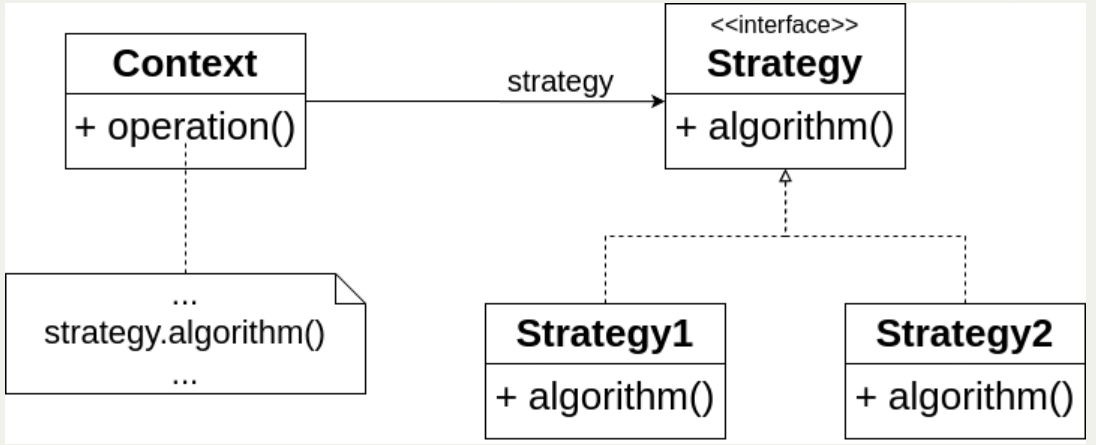
State vs Strategy

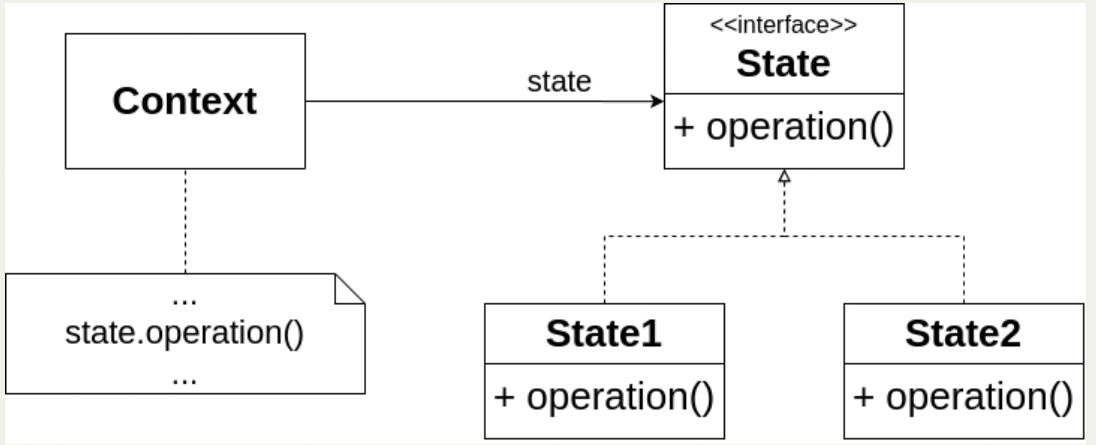
Instructors **Battista Biggio**, **Angelo Sotgiu** and **Leonardo Regano**

M.Sc. in Computer Engineering, Cybersecurity and Artificial Intelligence
University of Cagliari, Italy

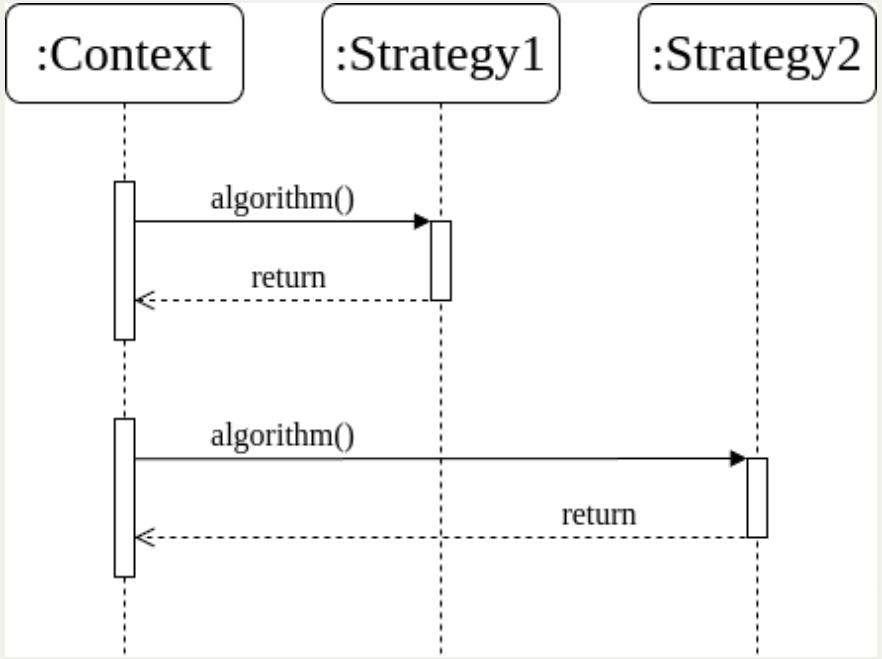
STRATEGY design pattern



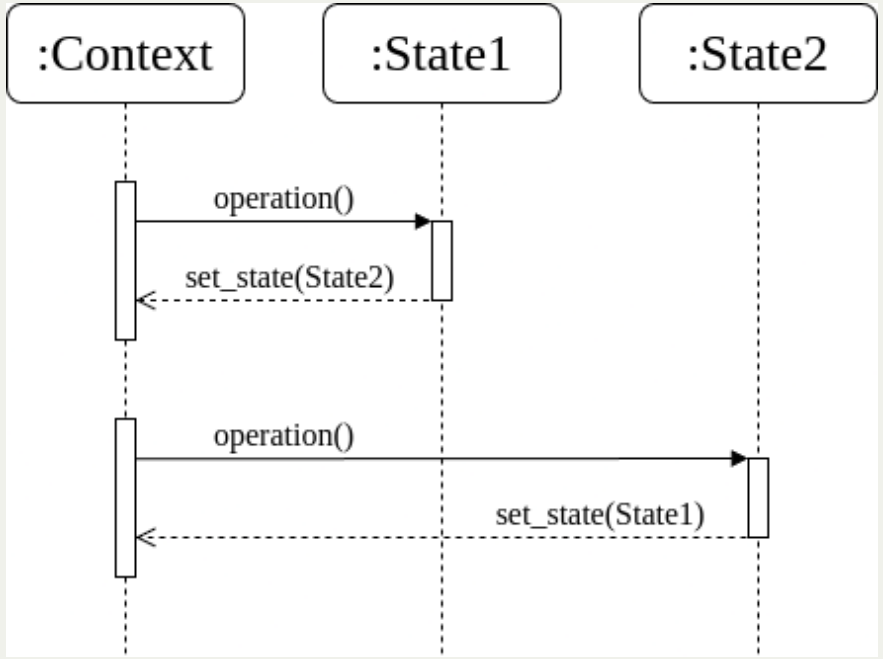
STATE design pattern



STRATEGY design pattern



STATE design pattern



Difference between STATE and STRATEGY

- Both design patterns show different behaviors depending on the context
- The biggest difference is that in STATE there is an **explicit state transition**, and the state transition is explicitly managed by the State object.
- In the STRATEGY pattern the transition between different strategies is driven by an external action.

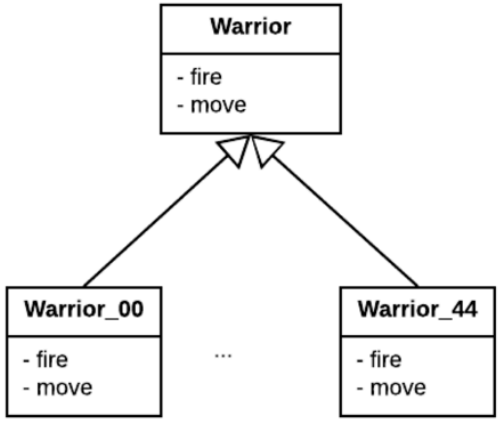
STATE and **STRATEGY** address distinct issues.

- The strategy pattern is employed to dynamically select an algorithm at runtime
- The state pattern facilitates **dynamic switching between different states** as a process evolves.
- In terms of implementation, a key distinction is that the strategy object usually lacks awareness of other strategy objects. In the state pattern, either the state or the context needs awareness of other states to manage transitions.

Composition vs. inheritance

- To attain distinct behaviors, we can employ **inheritance**.
- If there are 5 types of behavior for **firepower** and 5 for **speed**, we have 25 possible behaviors to be implemented in 25 different classes.
- In addition to the large number of classes, we have to consider the fact that objects can change behavior, and therefore they can change class.

SPEED \ FIREPOWER	0	1	2	3	4
0	Class_00	Class_01	Class_02	Class_03	Class_04
1	Class_10	Class_11	Class_12	Class_13	Class_14
2	Class_20	Class_21	Class_22	Class_23	Class_24
3	Class_30	Class_31	Class_32	Class_33	Class_34
4	Class_40	Class_41	Class_42	Class_43	Class_44



Composition vs. inheritance

- When dealing with numerous types of behaviors, opting for composition might prove more convenient. This involves integrating an object (such as a **State** or a **Strategy** object) into the 'main' object, to provide the desired behavior.
- This approach helps to avoid the proliferation of classes.
- Different behaviors can be implemented by **dynamically altering the functions** that define the behavior.