



UPPSALA  
UNIVERSITET

# Instructions



UPPSALA  
UNIVERSITET



UPPSALA  
UNIVERSITET

# Design and Development

Paradigms and common  
models



UPPSALA  
UNIVERSITET

## Utilising The formal process of Development

Turns our software process form a unformed chaos with low chance of success into a well ordered shareable software solution.



UPPSALA  
UNIVERSITET

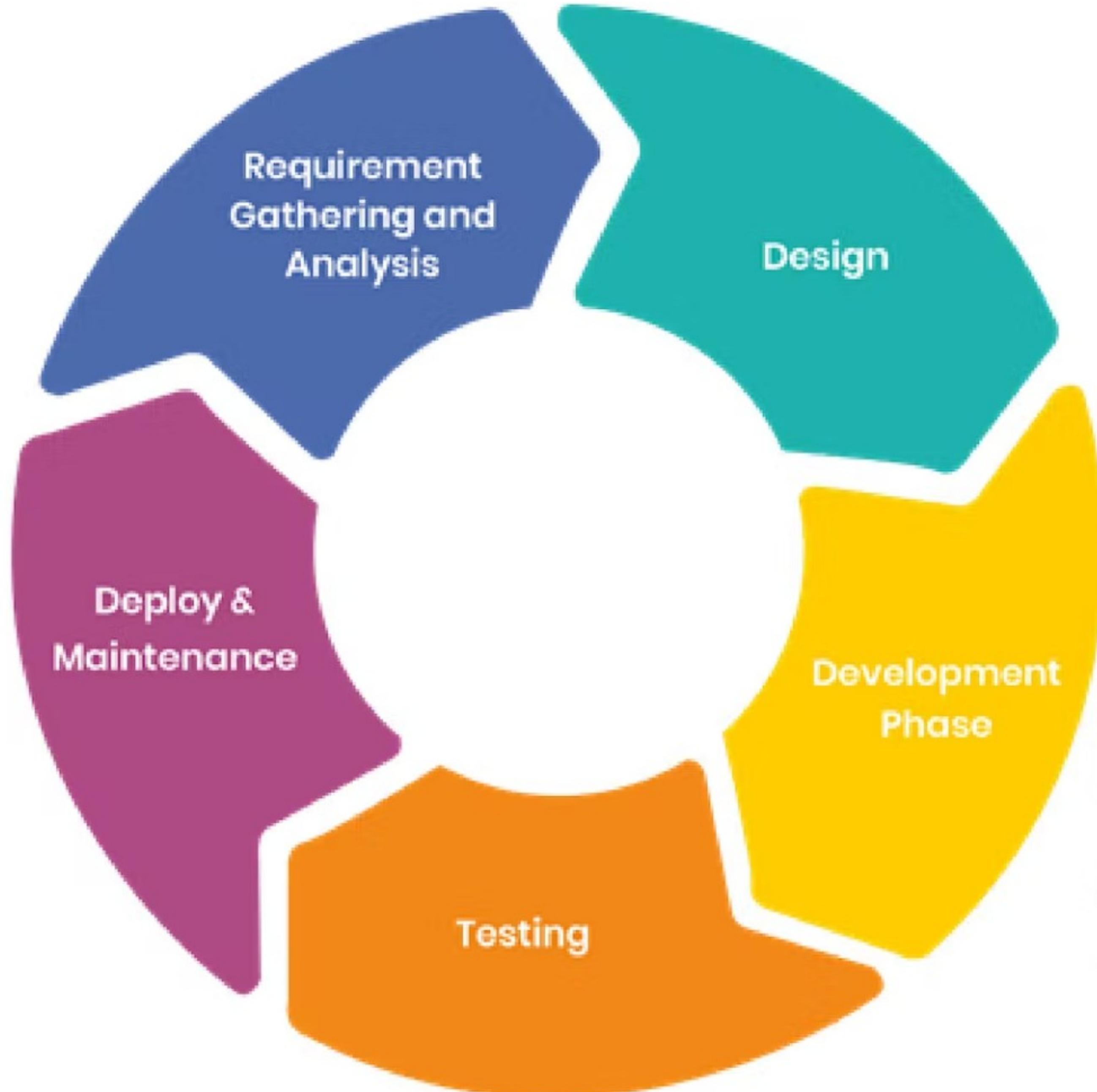




UPPSALA  
UNIVERSITET



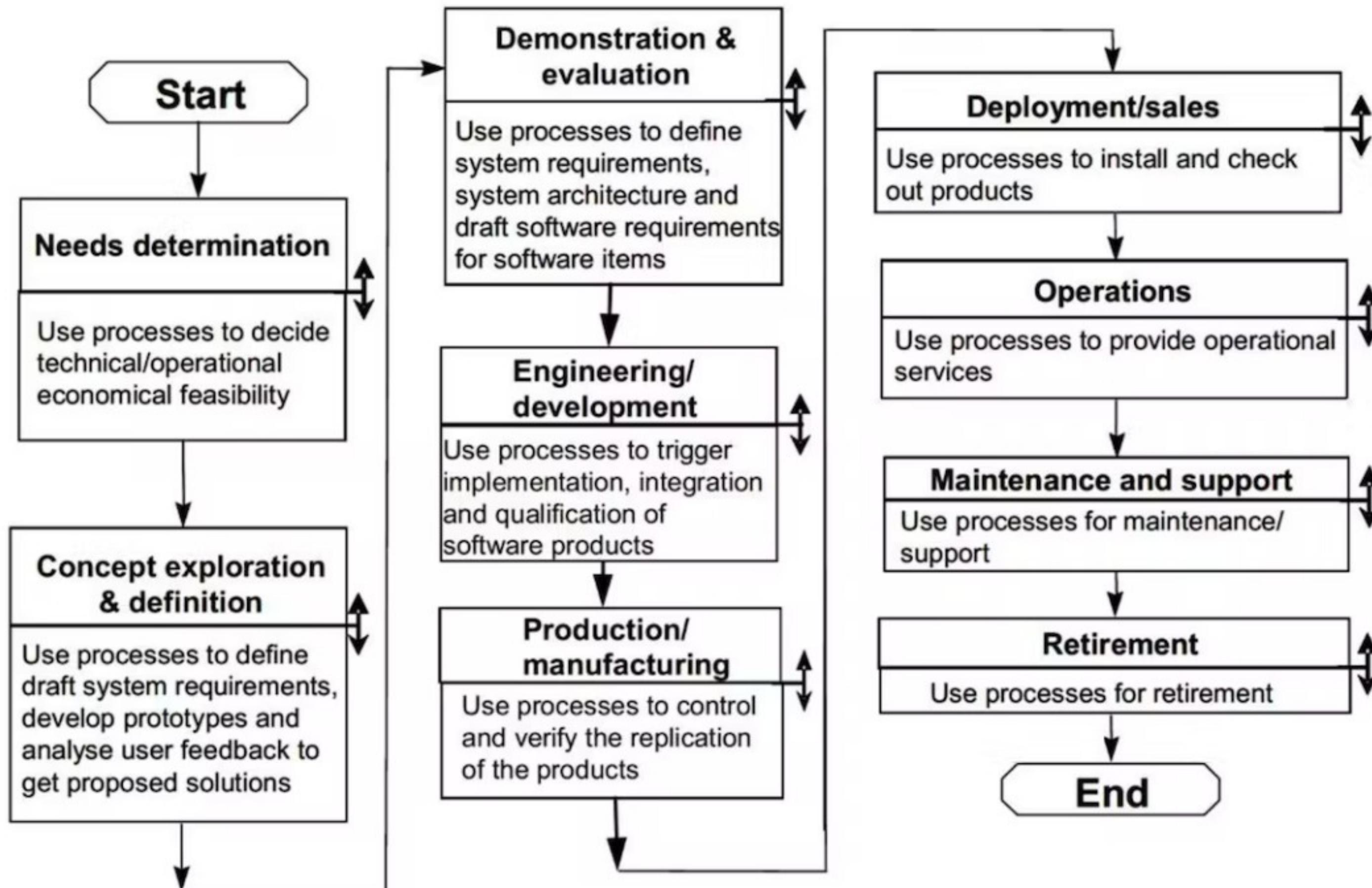
UPPSALA  
UNIVERSITET



# SDLC

Iterations build software over time until the cost of the next phase exceeds the cost we are willing to pay.





# What are common needs for scientific software

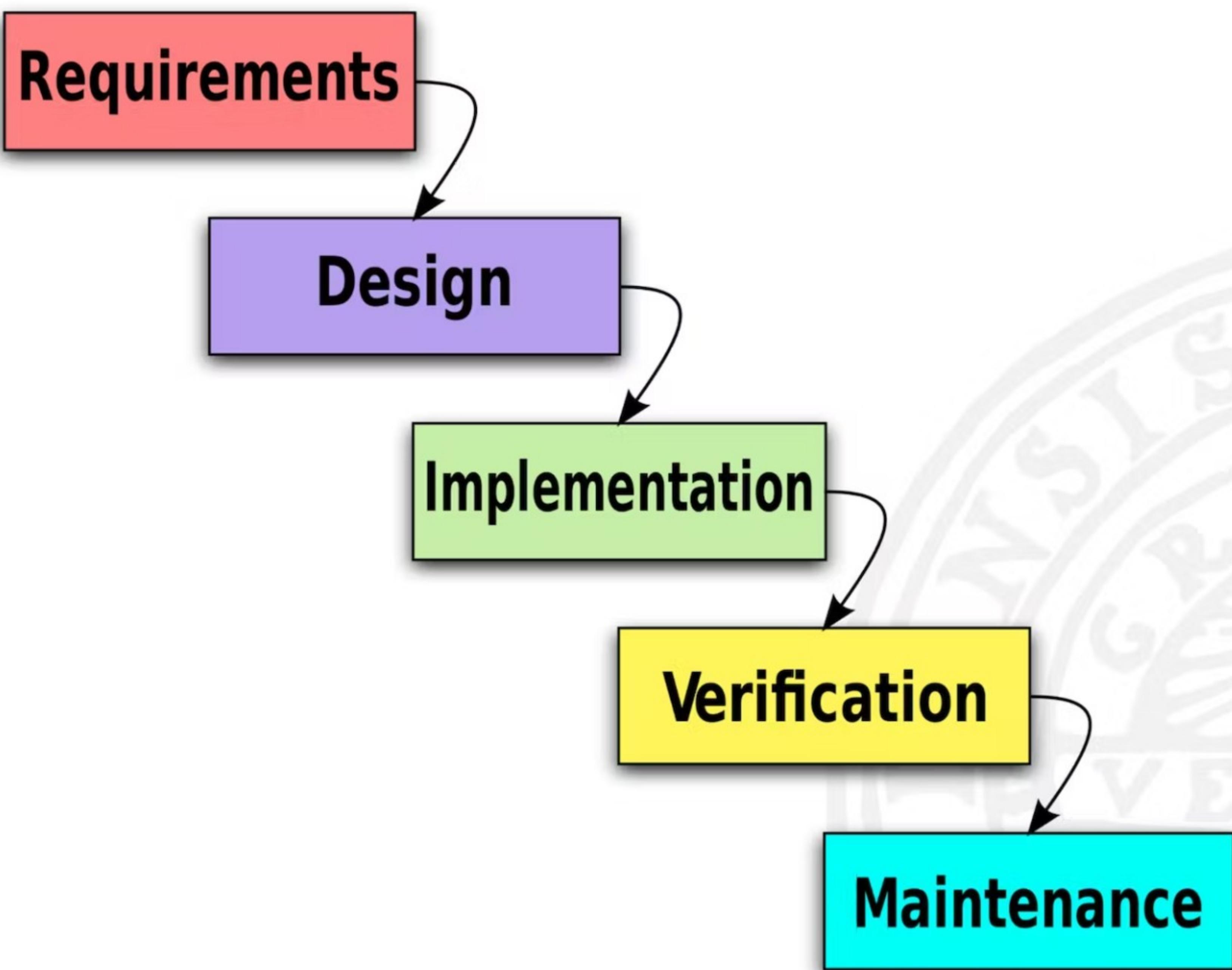




UPPSALA  
UNIVERSITET



UPPSALA  
UNIVERSITET





## 1. Determine objectives

## 2. Identify and resolve risks

Review

## 4. Plan the next iteration

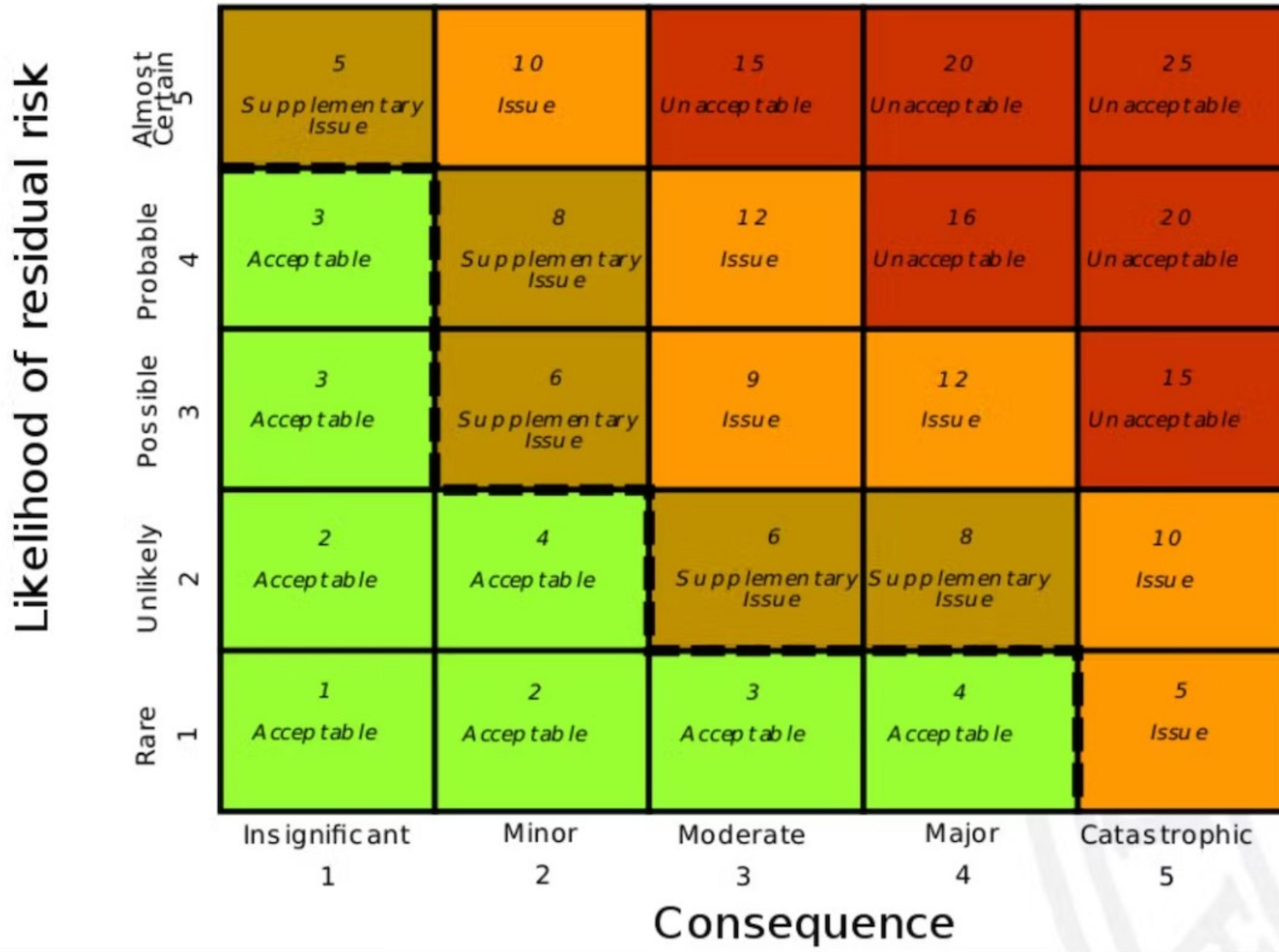
## 3. Development and Test

Cumulative cost  
Progress





# Risk analysis



# Typical Risk when developing Scientific software

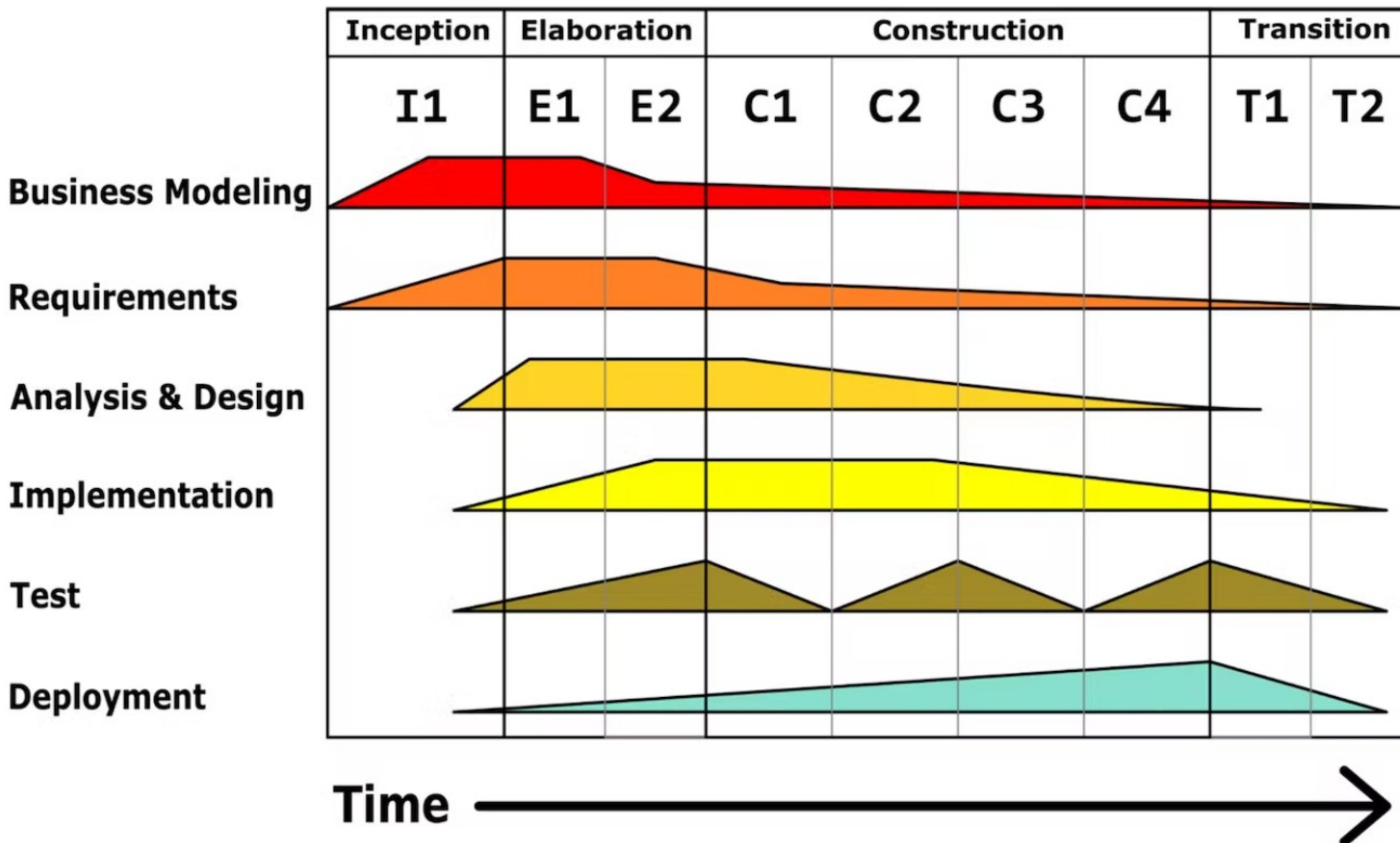
misuse	relevance	output misinterpretation
poorly optimized		not enough competences
	compatible with oss	too little time to develop
		improper testing      unreproducible
	too specific	difficult to maintain
corrupted output		inaccurate results
		version incompatibility
missing code reviews		compatibility issues
		without testing phase
missing documentation		bugs with different input
not enough testing		proprietary dependencies

# Exercise 1

Imagen that you are making a new scientific program like "blast" with a web interface Discuss Risks and Requirements  
(A short Needs Analysis)

# Iterative Development

Business value is delivered incrementally in time-boxed crossdiscipline iterations.



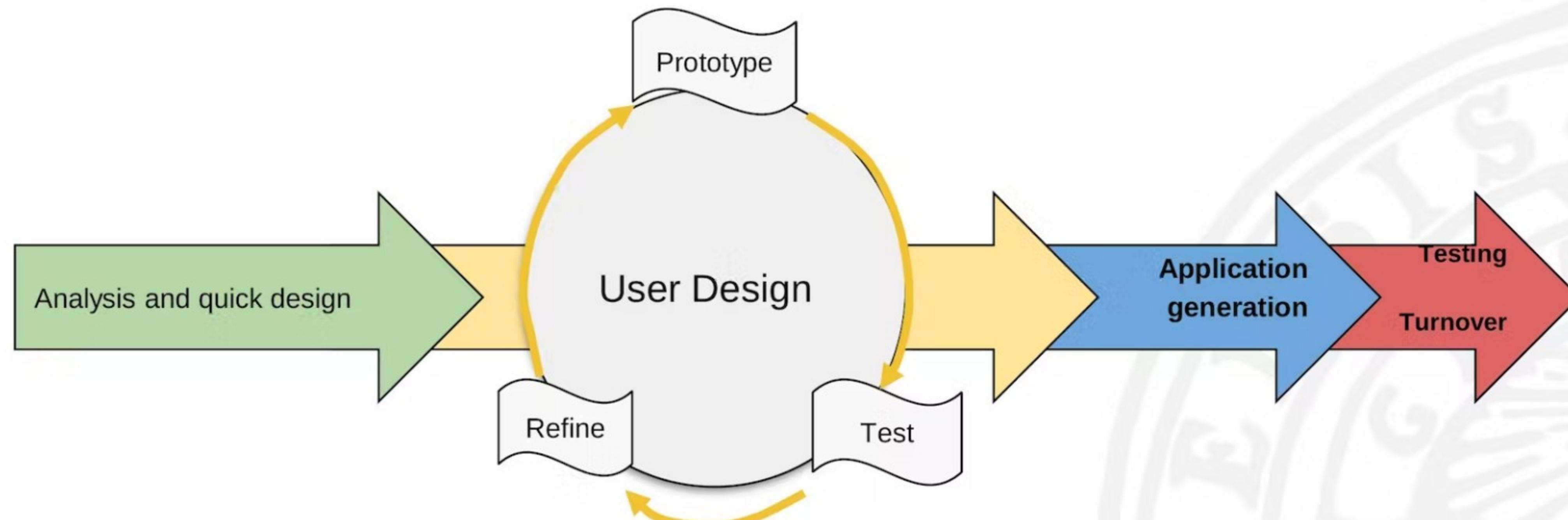


UPPSALA  
UNIVERSITET



UPPSALA  
UNIVERSITET

## Rapid Application Development process



We are uncovering better ways of developing software by doing it and helping others do it.  
Through this work we have come to value:

Individuals and interactions over processes and tools  
Working software over comprehensive documentation  
Customer collaboration over contract negotiation  
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck , James Grenning , Robert C. Martin  
Mike Beedle, Jim Highsmith, Steve Mellor  
Arie van Bennekum, Andrew Hunt, Ken Schwaber  
Alistair Cockburn, Ron Jeffries , Jeff Sutherland  
Ward Cunningham , Jon Kern, Dave Thomas  
Martin Fowler, Brian Marick



# Principles behind the Agile Manifesto

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

# Principles behind the Agile Manifesto

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

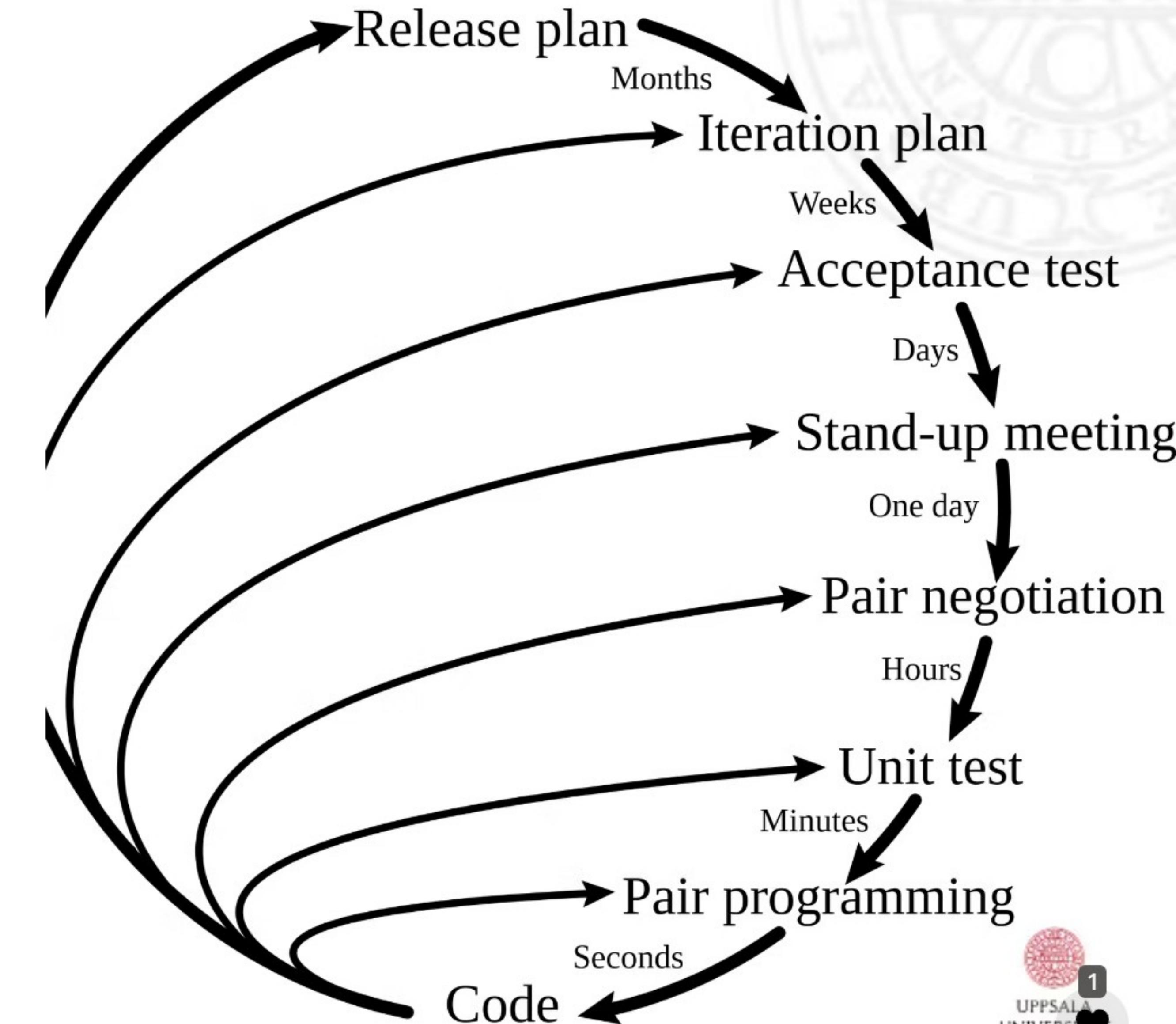
Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

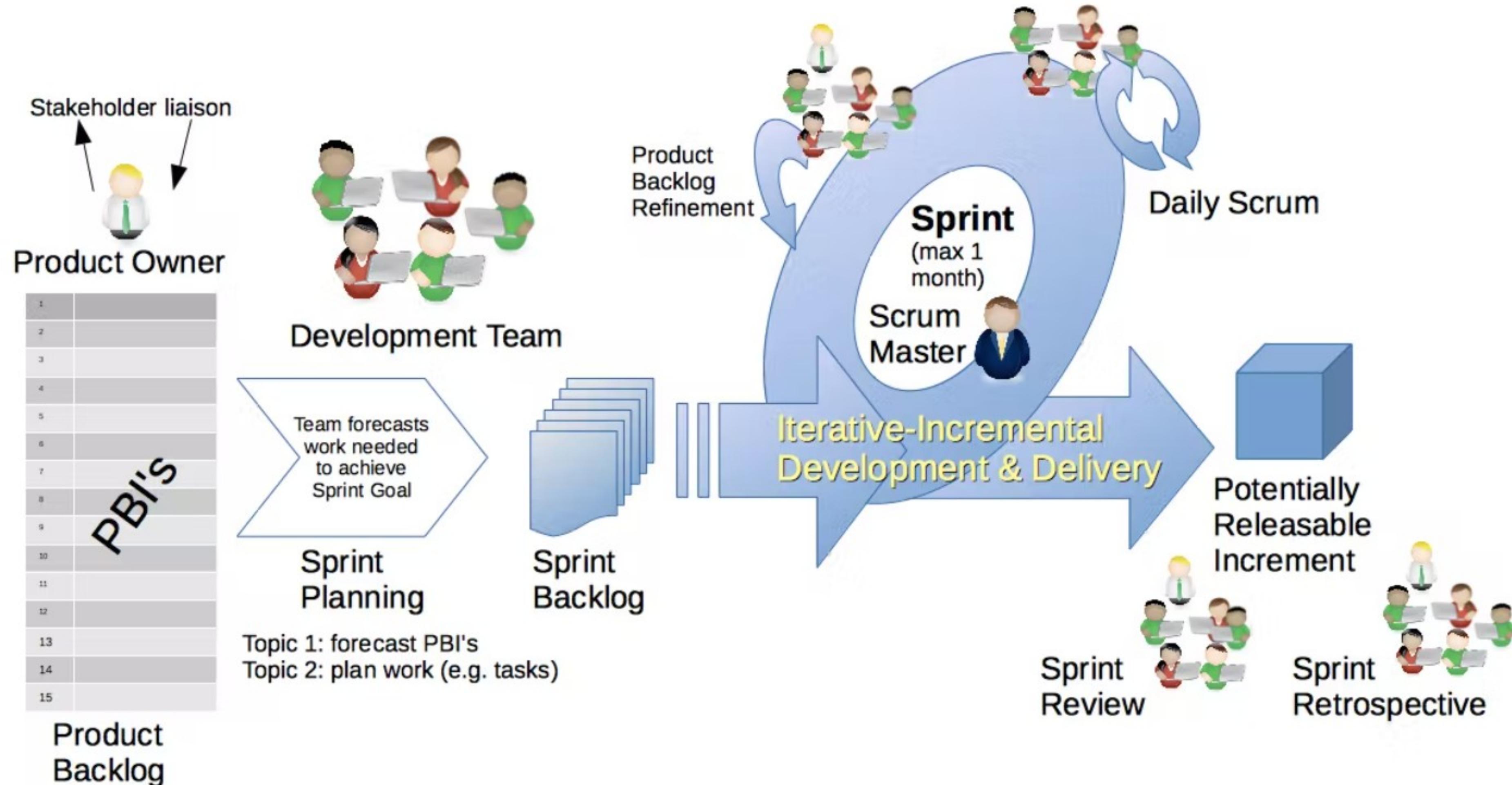
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

# Planning/feedback loops



## eXtreme Programming(XP)

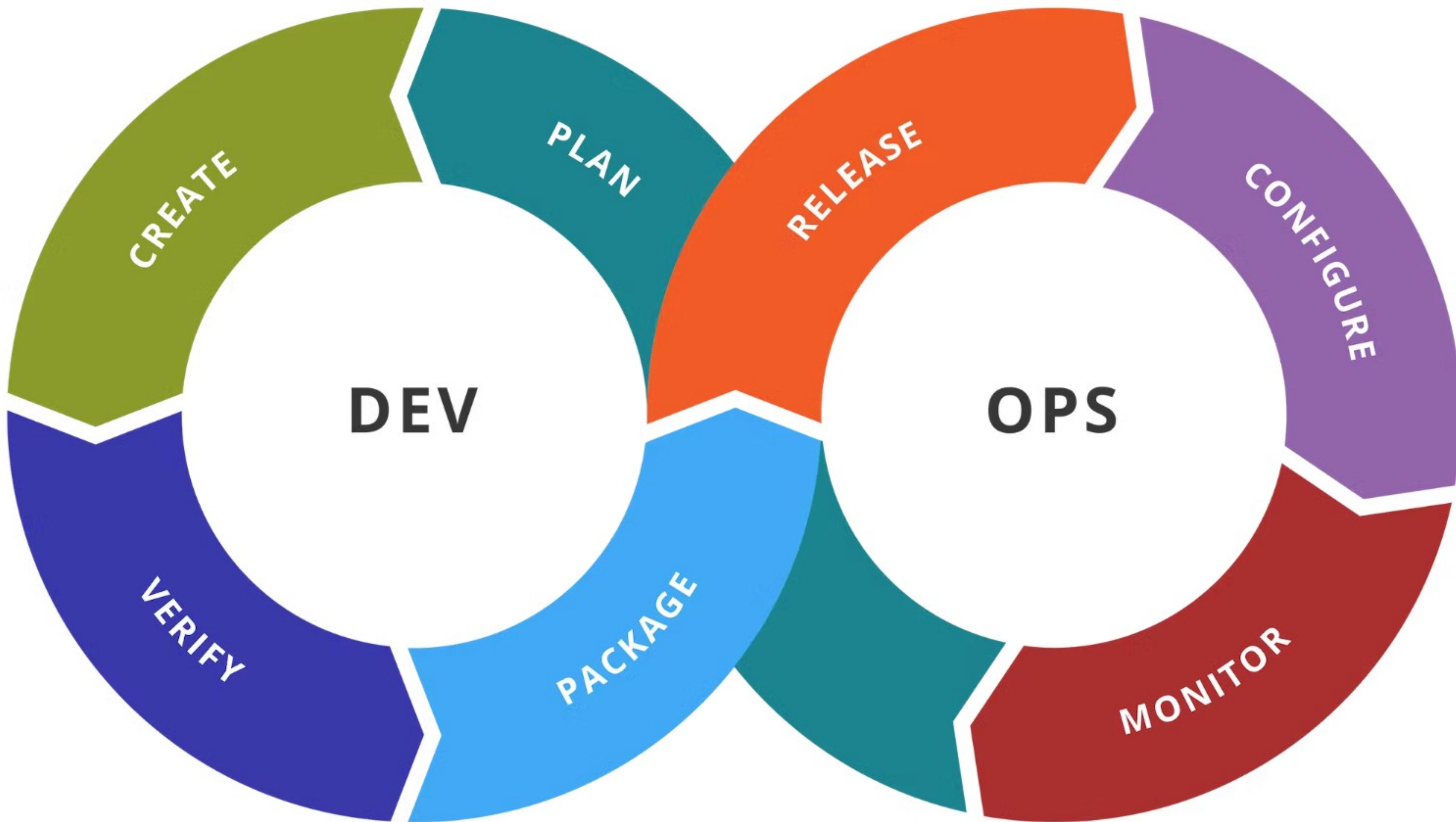
- Key Practice #1 – Pair Programming
- Key Practice #2 – Planning Game, precursor to what many of us know as “Sprint Planning”
- Key Practice #3 – Continuous Process
- Key Practice #4 – Coding Standards
- Key Practice #5 – Sustainable Pace
- Key Practice #6 – Test Driven Development (TDD)



## The Scrum process,

[https://upload.wikimedia.org/wikipedia/commons/d/df/Scrum\\_Framework.png](https://upload.wikimedia.org/wikipedia/commons/d/df/Scrum_Framework.png)

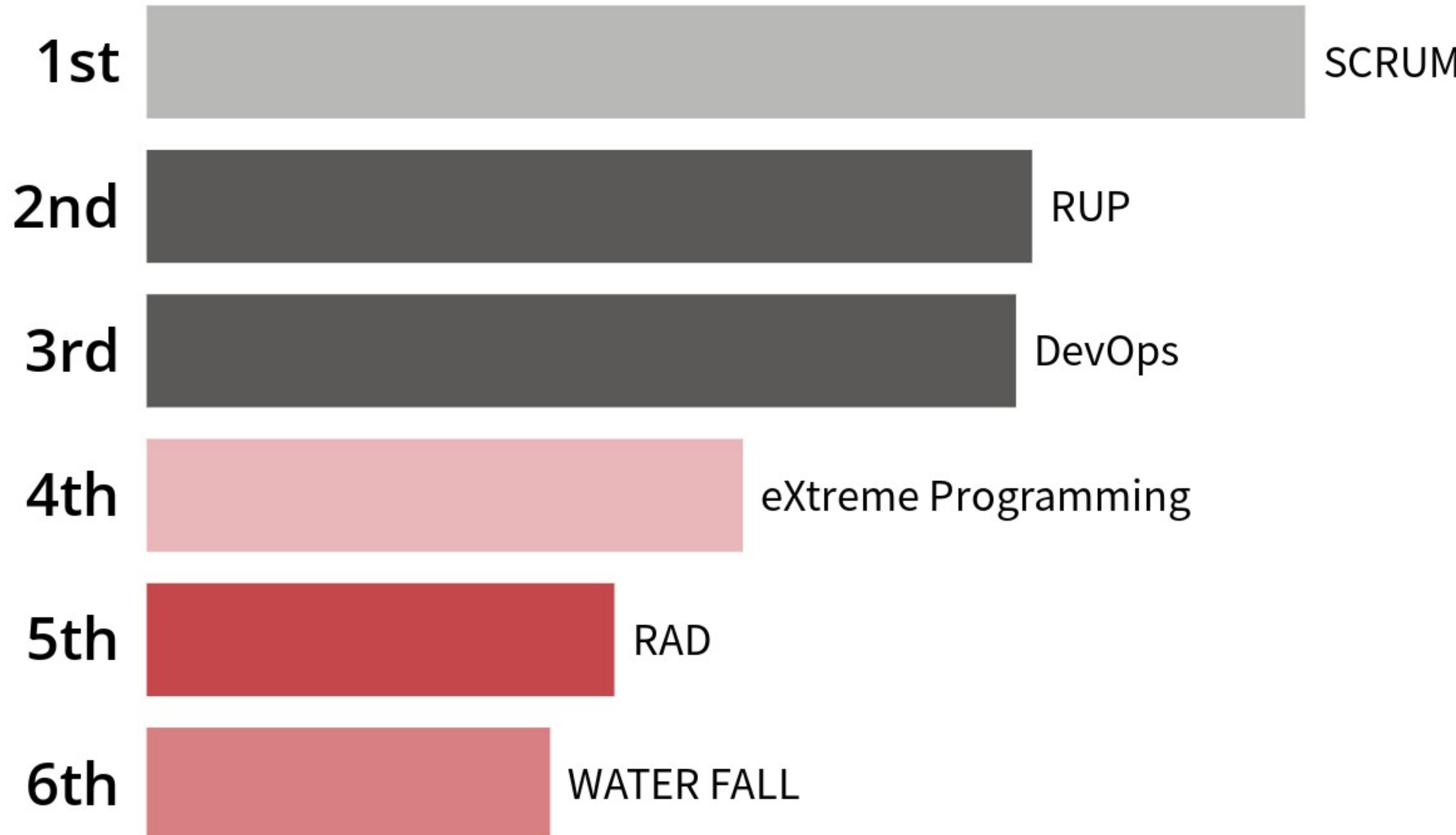




DevOps combines development and operations



# Rank in order of suitable scale of project(largest to smallest)



# Objects And obejectorientation



Dragon



Hero



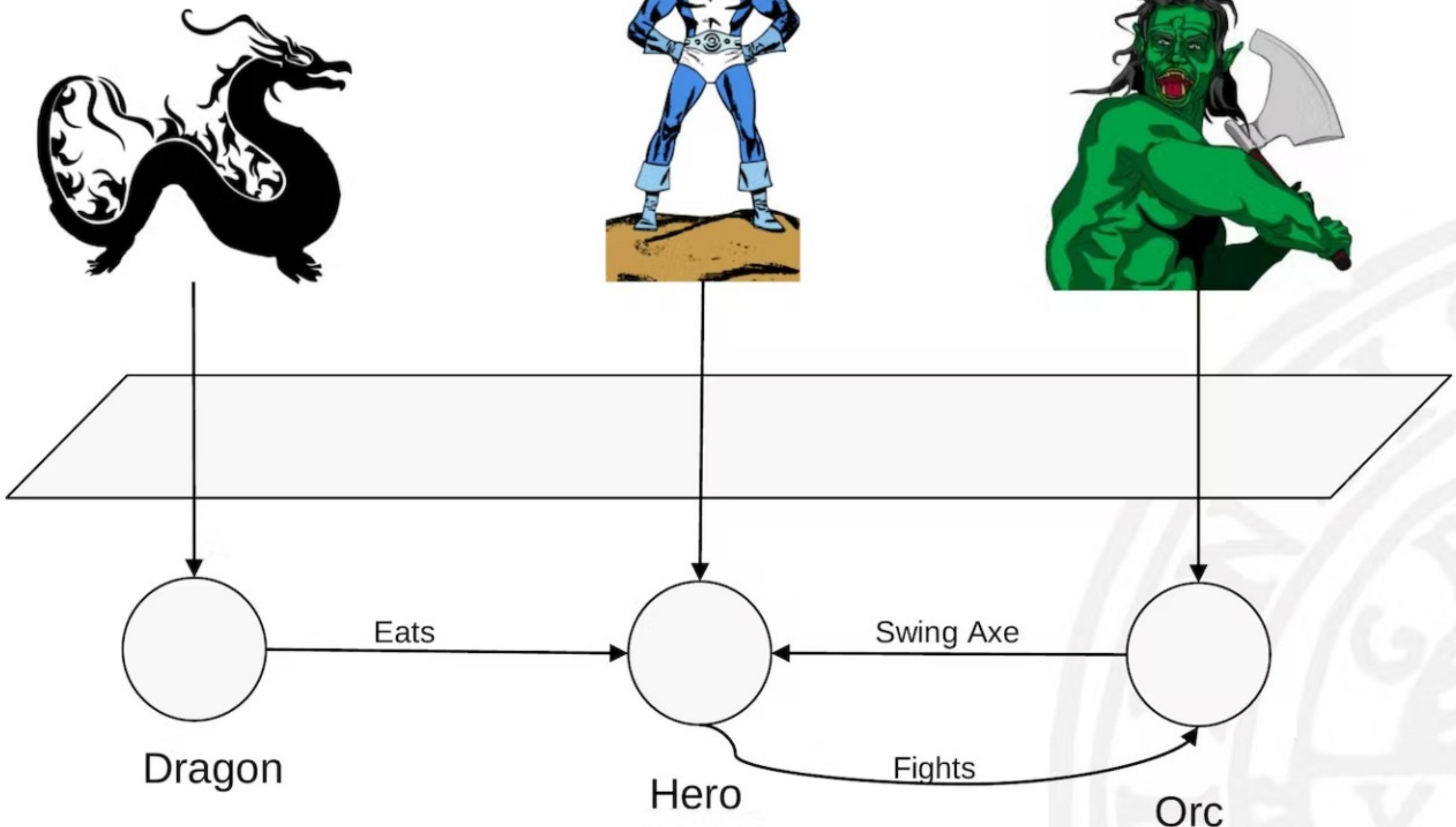
Orc



UPPSALA  
UNIVERSITET



UPPSALA  
UNIVERSITET



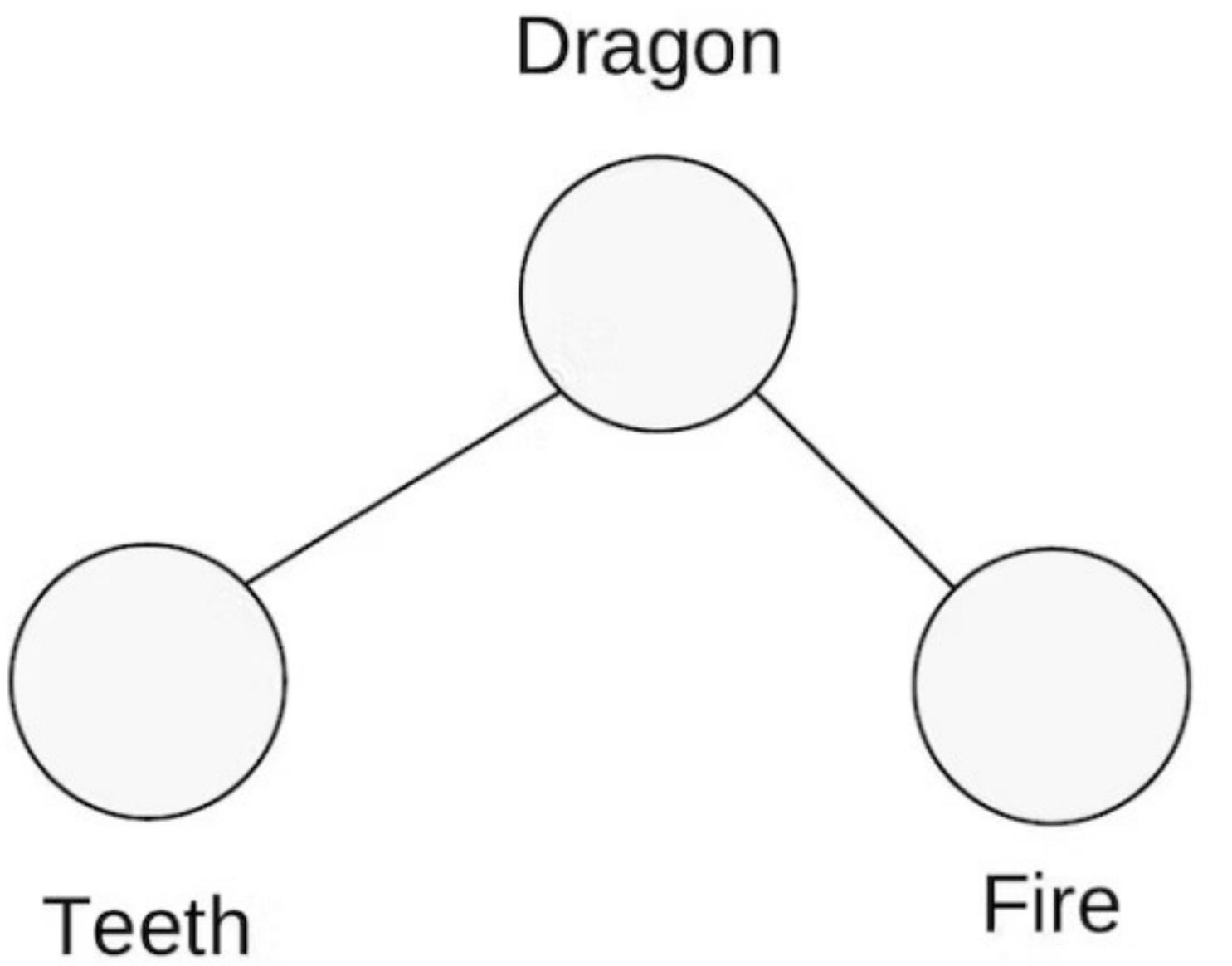


UPPSALA  
UNIVERSITET

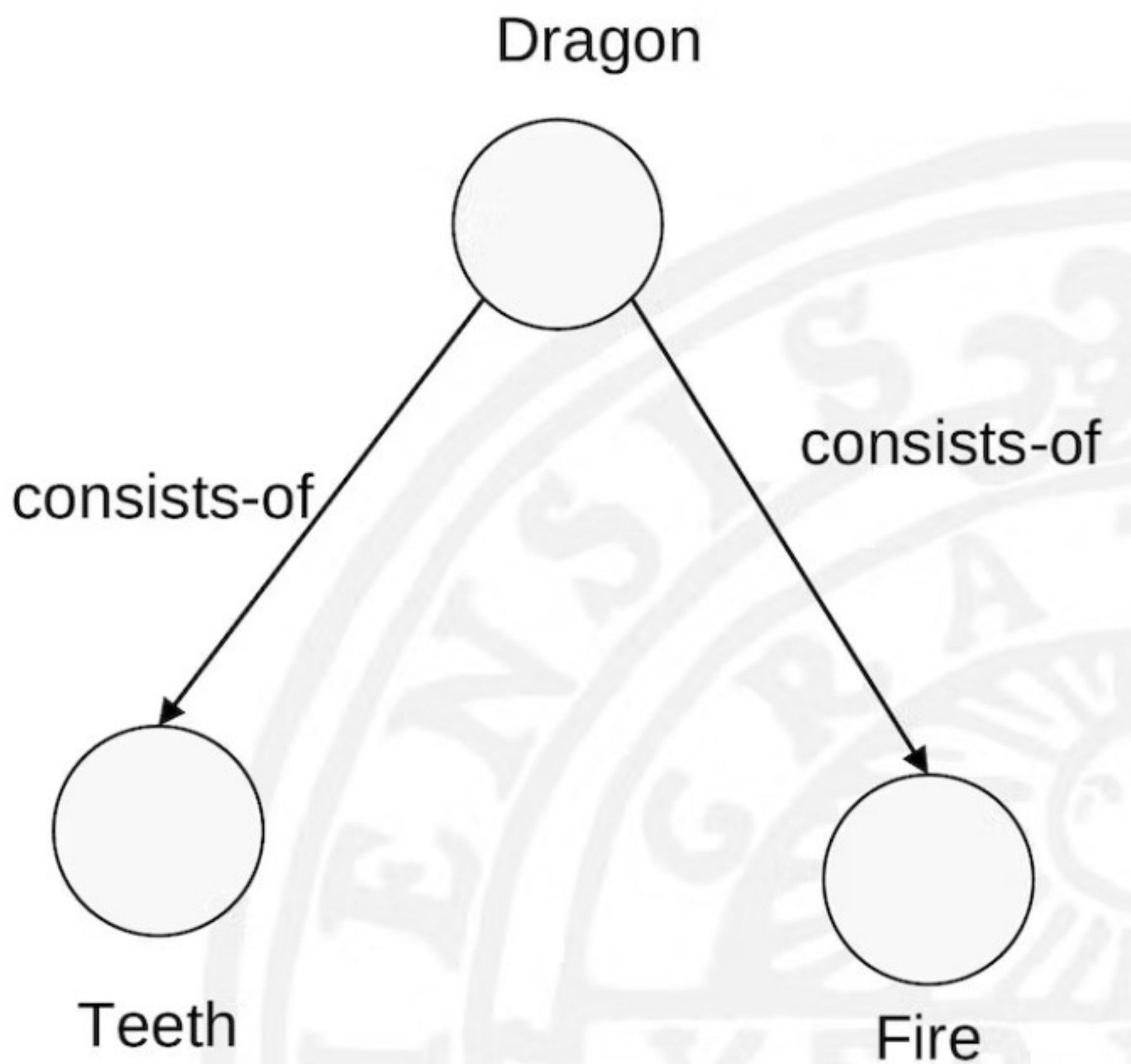


UPPSALA  
UNIVERSITET

## Partition



## Aggregate



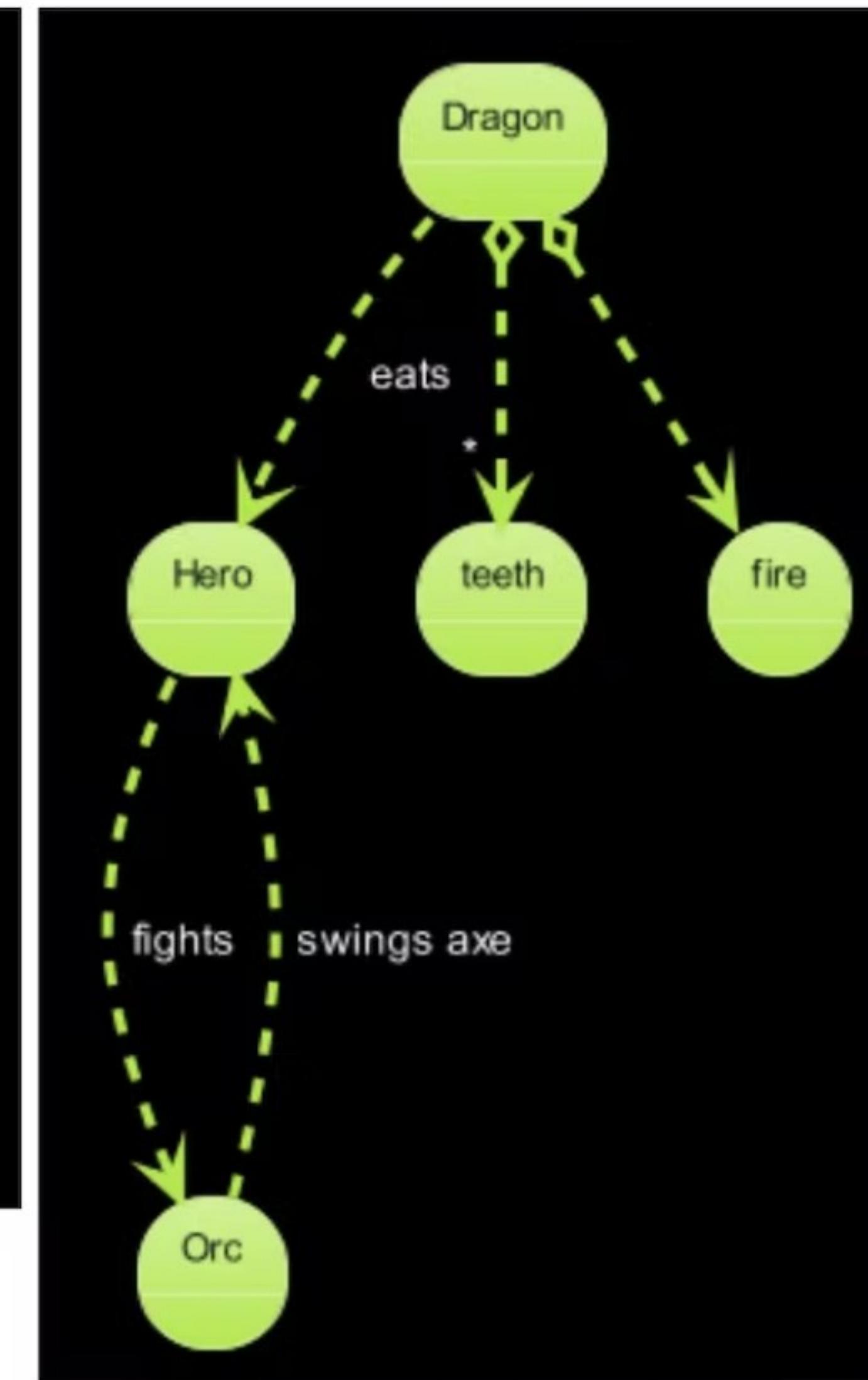


UPPSALA  
UNIVERSITET

```
@startuml
!theme hacker
object Dragon
object Hero
object Orc
object teeth
object fire
```

```
Dragon..>Hero:eats
Hero..>Orc:fights
Orc..>Hero:swings axe
Dragon o..> "*" teeth
Dragon o..> fire
```

```
@enduml
```

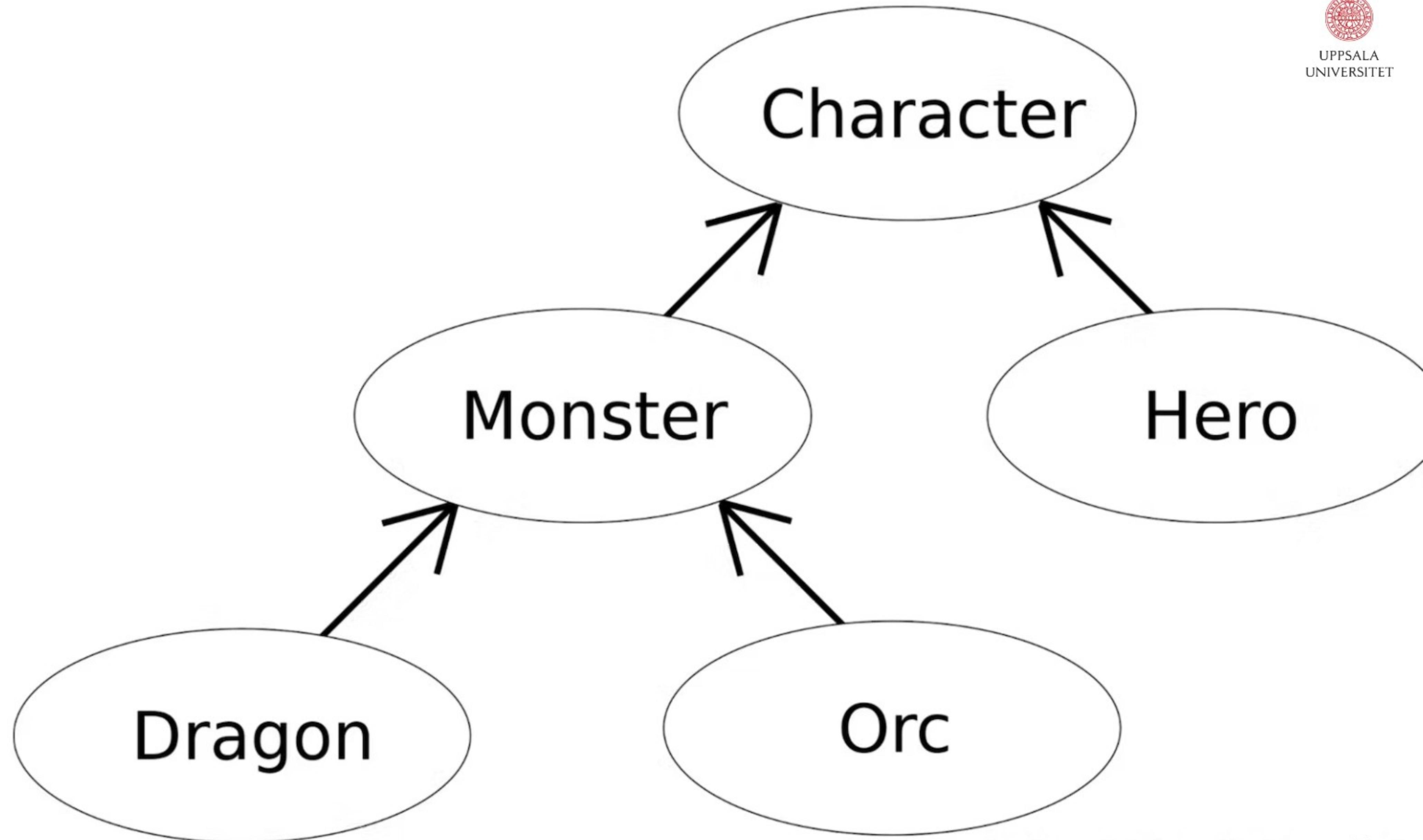




UPPSALA  
UNIVERSITET



UPPSALA  
UNIVERSITET



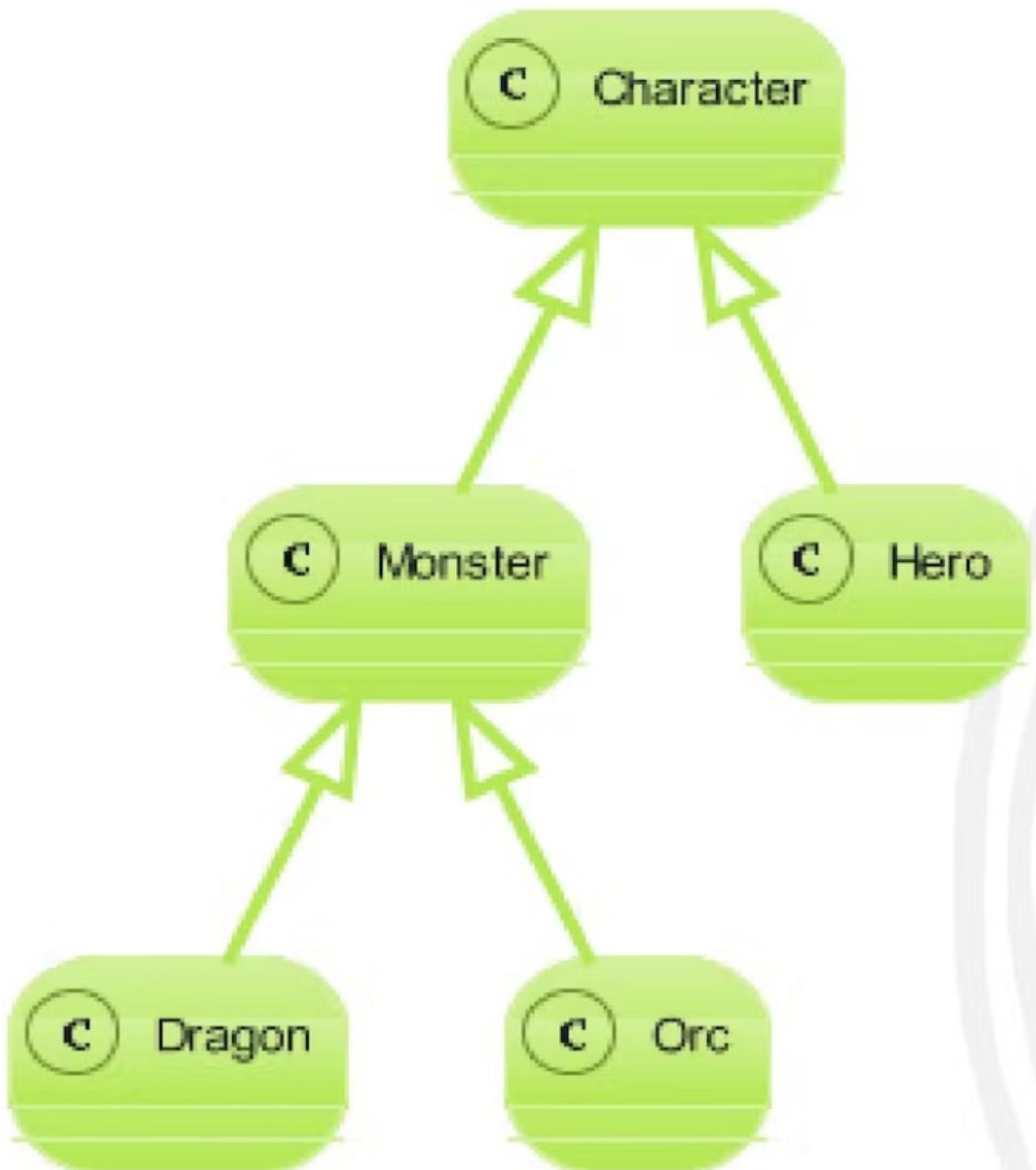


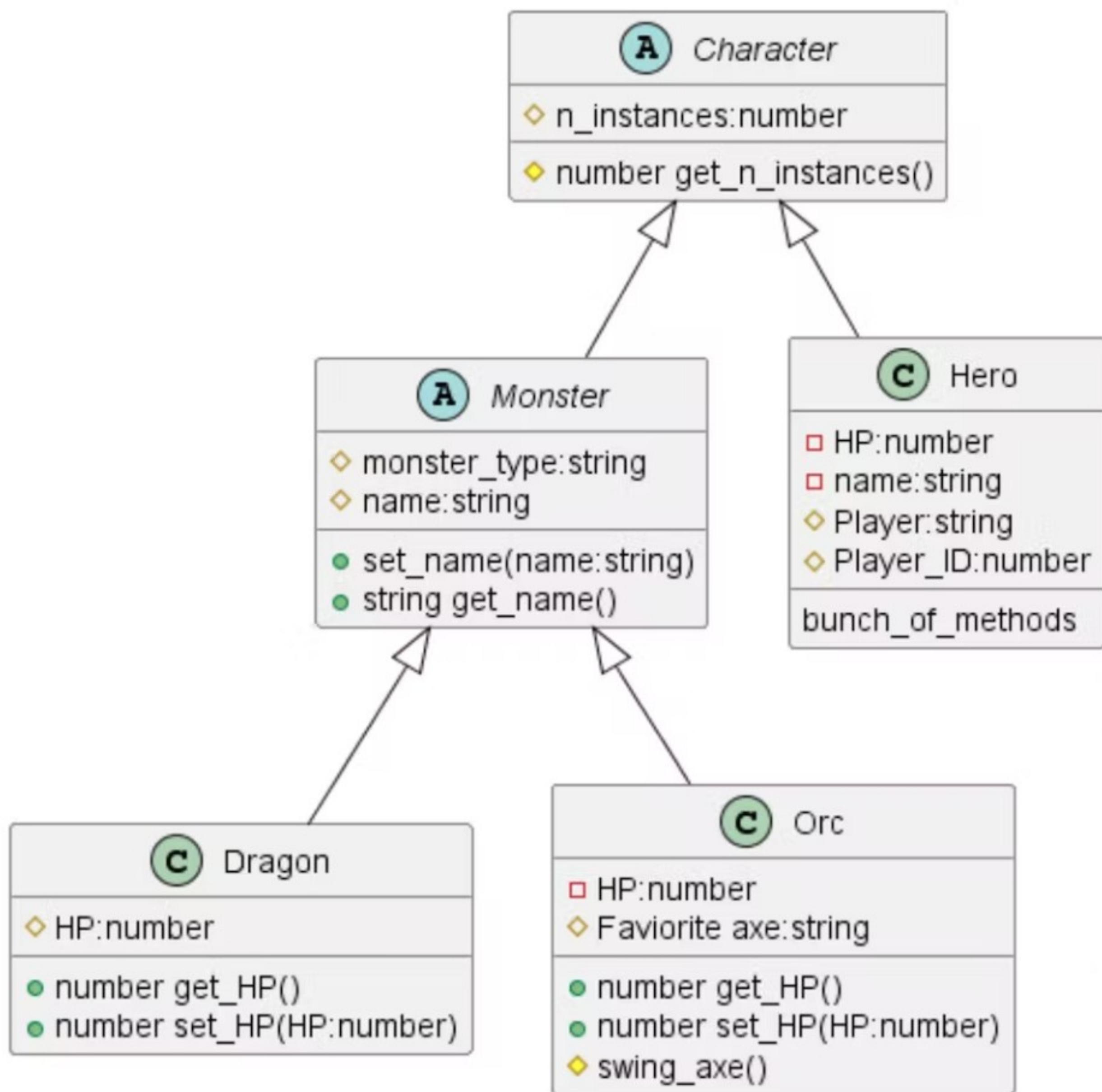
UPPSALA  
UNIVERSITET



UPPSALA  
UNIVERSITET

# Class Diagram







# reference cheats

<https://modeling-languages.com/best-uml-cheatsheets-and-reference-guides/>

# Exercise 2

Design a class hierarchy that solves the requirements from Exercise 1 (Observation does not have to be complete )

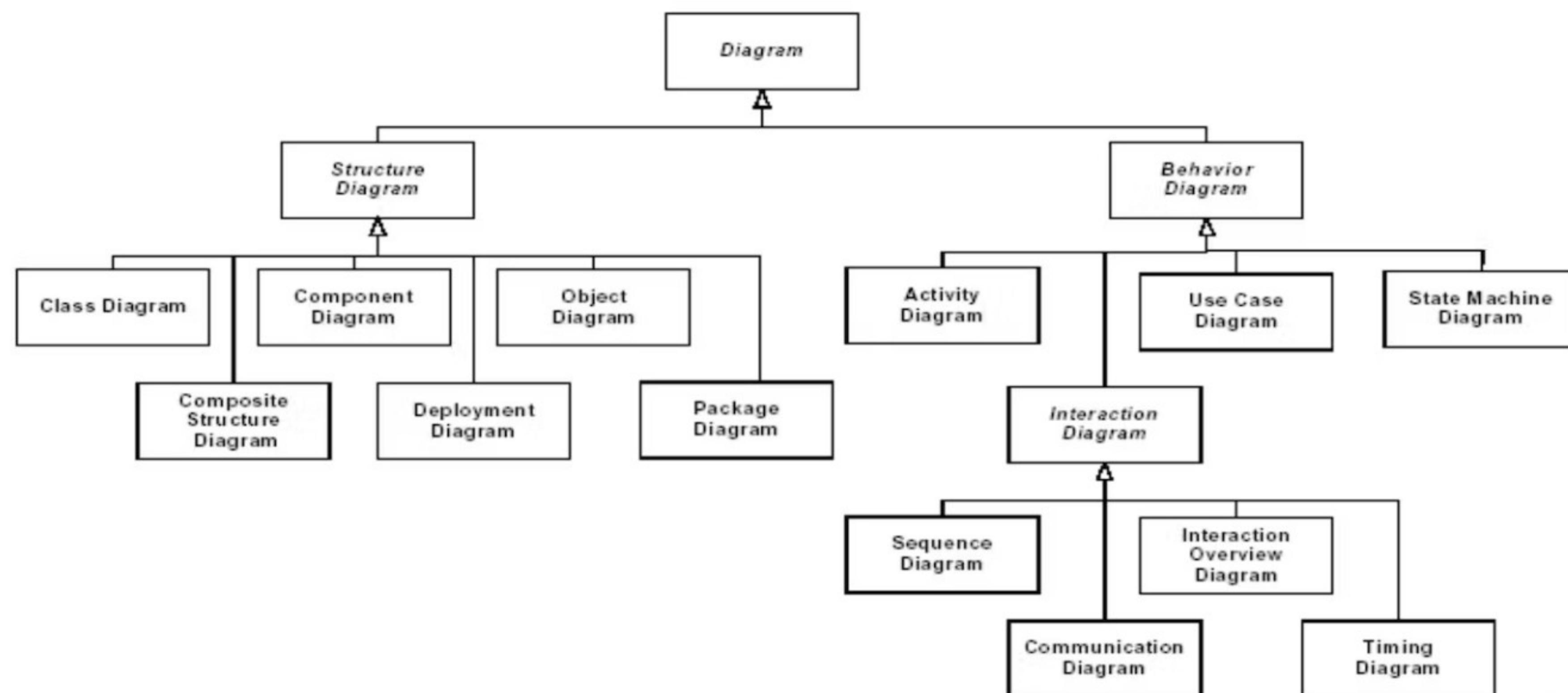
<https://plantuml.com/class-diagram>

[https://www.tutorialspoint.com/uml/uml\\_class\\_diagram.htm](https://www.tutorialspoint.com/uml/uml_class_diagram.htm)





# Schema of UML diagram types





UPPSALA  
UNIVERSITET

# Bubble Sort

```
bubble( array ):  
    for each element in array:  
        for each element i in array:  
            if array[i] > array[i+1]: swap array[i] and array[i+1]
```

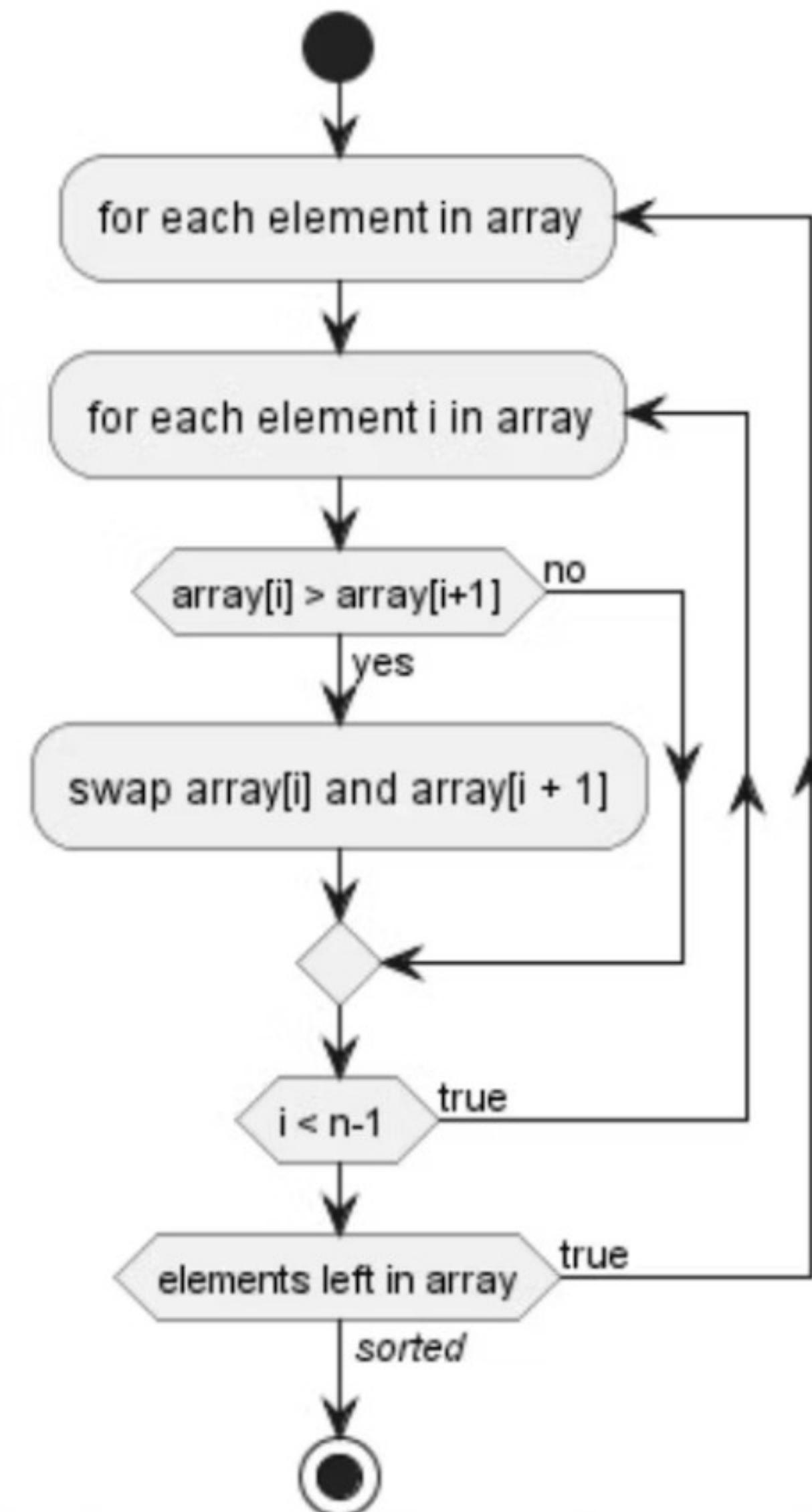




UPPSALA  
UNIVERSITET

```
```plantuml
@startuml
!pragma useVerticalIf on
start
repeat:for each element in array;
    repeat:for each element i in array;
        if (array[i] > array[i+1]) then (yes)
            :swap array[i] and array[i + 1];
        else (no)
        endif
    repeat while (i < n-1 ) is (true)
    repeat while (elements left in array) is (true) not
        //sorted//)
stop
@enduml
```

```



# Exercise 3

Create a Activity diagram that shows the processes of the Needleman-Wunsch algorithm, one diagram for each of the two steps. save to your own course git from day one.

<https://plantuml.com/activity-diagram-beta>





## Needleman-Wunsch pseudocode for calculating F:

```
for i in length(A):
    F(i,0) = p * i
for j in length(B):
    F(0,j) = p * j
for i in length(A):
    for j in length(B):
        match = F(i-1, j-1) + S(A(i), B(j))
        delete = F(i-1, j) + p
        insert = F(i, j-1) + p
        F(i,j) = max(match, delete, insert)
```

## Needleman-Wunsch pseudocode for calculating the alignment:

```
i = length(A)
j = length(B)
while ( i>0 and j>0 ):
    if (i>0 and j>0 and F(i, j) == F(i-1, j-1) + S(A(i), B(j)))
        // match! Or
        alignA = A(i) + alignA
        alignB = B(j) + alignB
        i = i - 1
        j = j - 1
    else if (i > 0 and F(i, j) == F(i-1, j) + d)
        // insertion in A or deletion in B
        alignA ← A(i) + alignA
        alignB ← "-" + alignB
        i = i - 1
    else
        // deletion in A or insertion in B
        alignA ← "-" + alignA
        alignB ← B(j) + alignB
        j = j - 1
    }
```