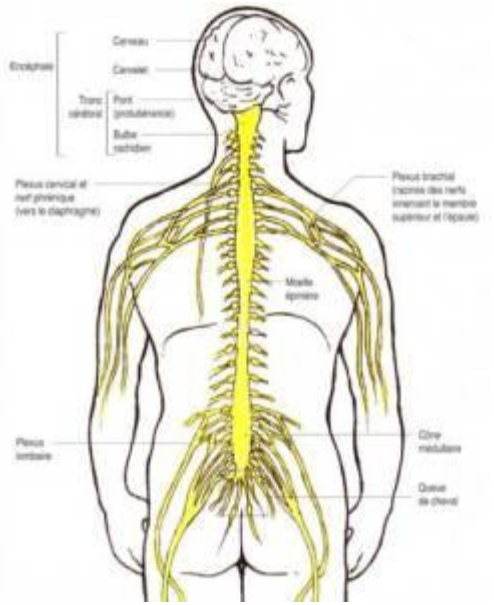
Introduction to Services
Focus on REST* services

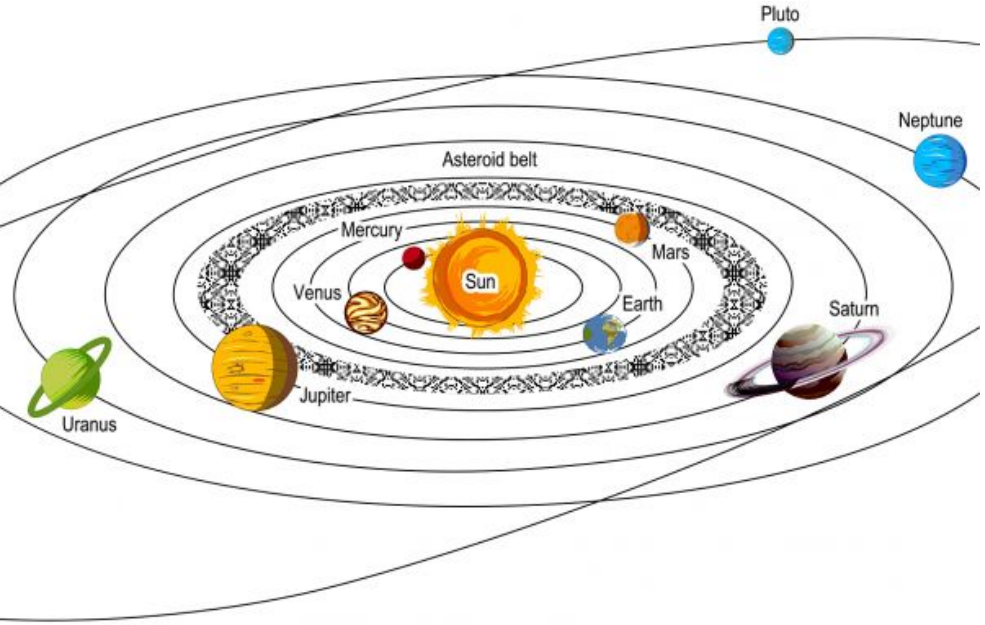
Based on Simon Urli's course

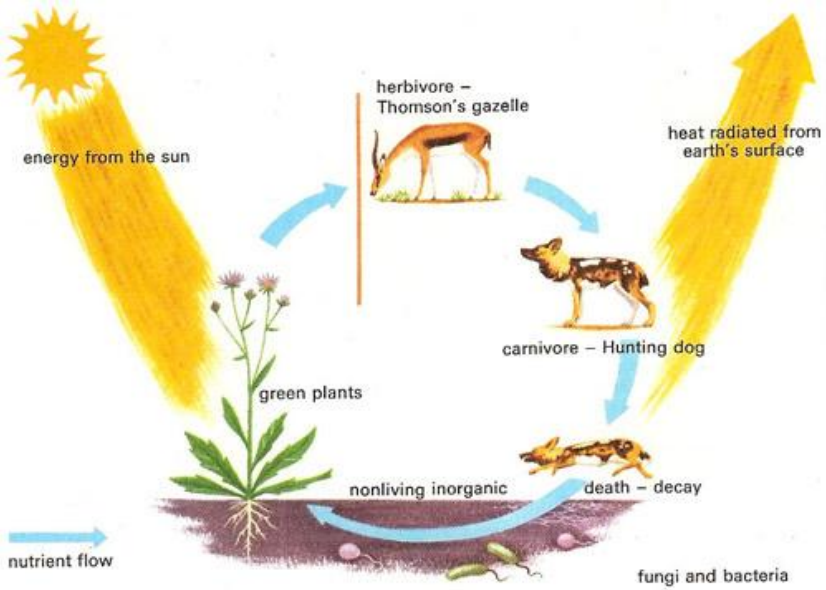
20/09/2016
Cécile Camillieri

iut Nice Côte d'Azur
Université Nice Sophia Antipolis

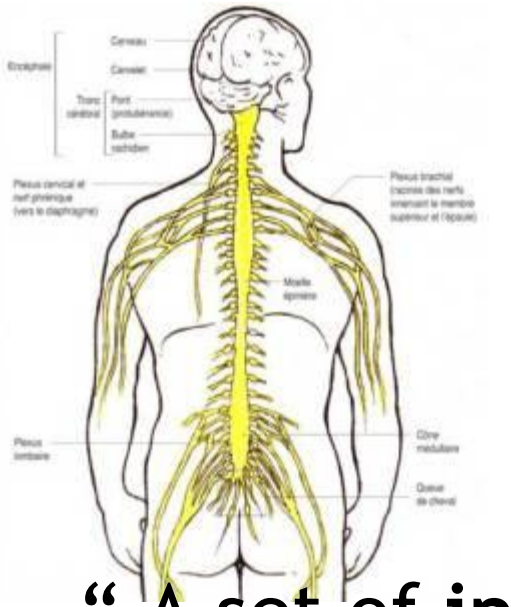


System?


Pluto
Neptune
Asteroid belt
Mercury
Mars
Venus
Sun
Earth
Saturn
Jupiter
Uranus

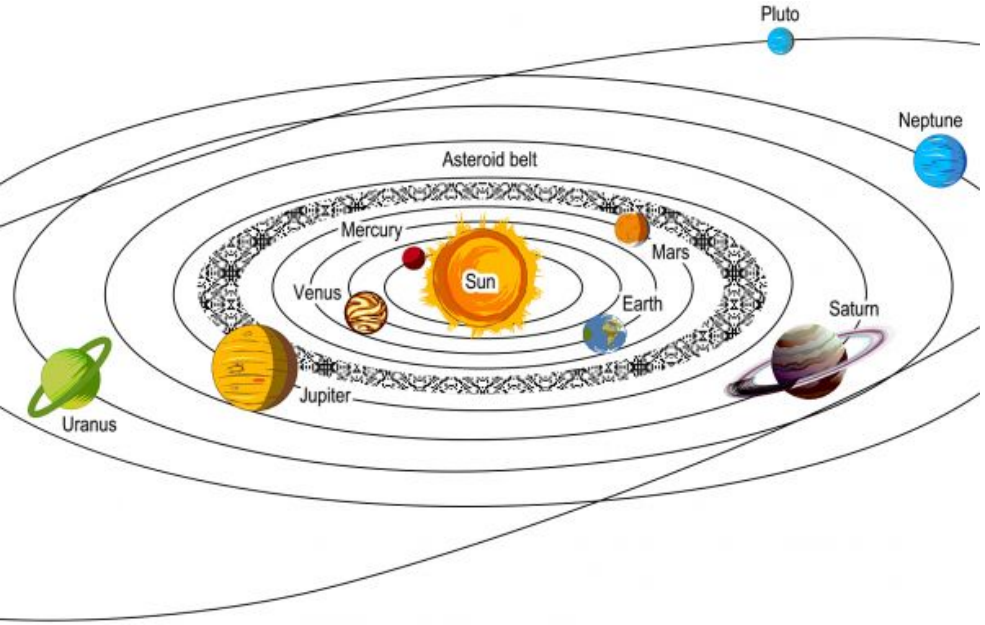

herbivore – Thomson's gazelle
energy from the sun
heat radiated from earth's surface
carnivore – Hunting dog
green plants
nonliving inorganic
death – decay
nutrient flow
fungi and bacteria

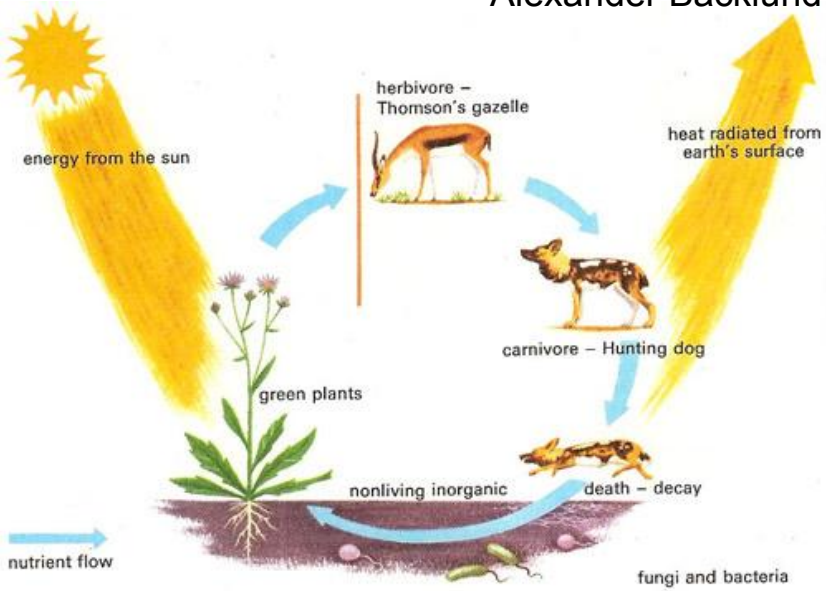
System?



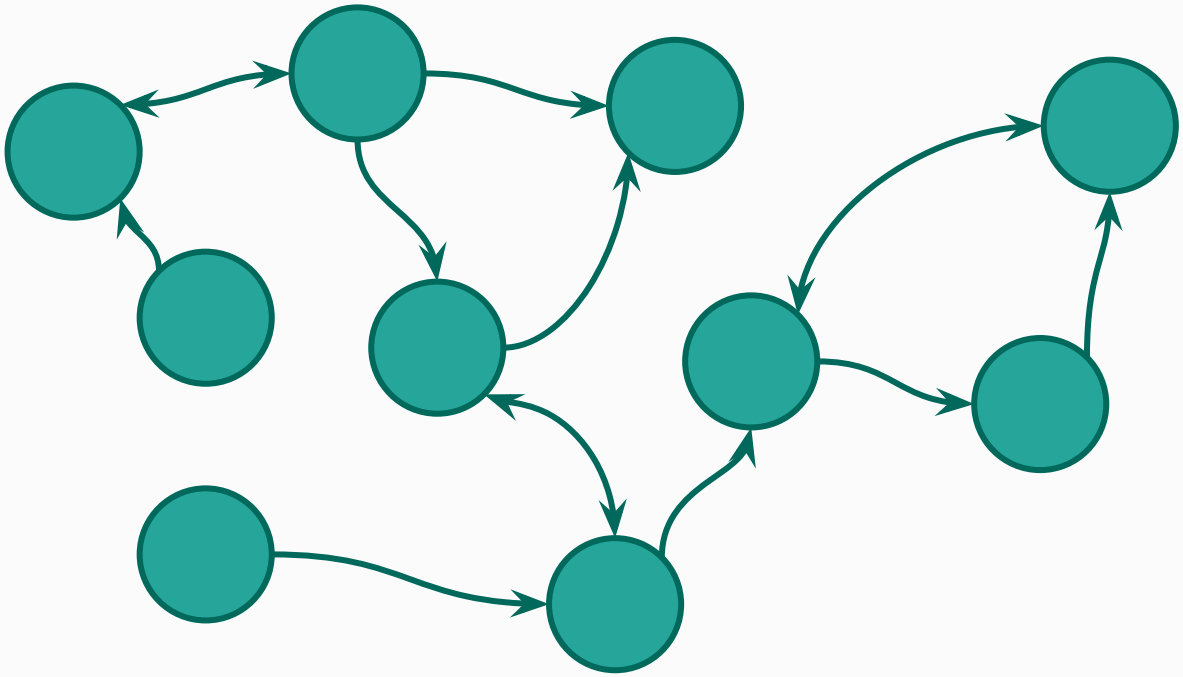
“ A set of **interacting** or **interdependent components** forming a complex whole ”

Alexander Backlund



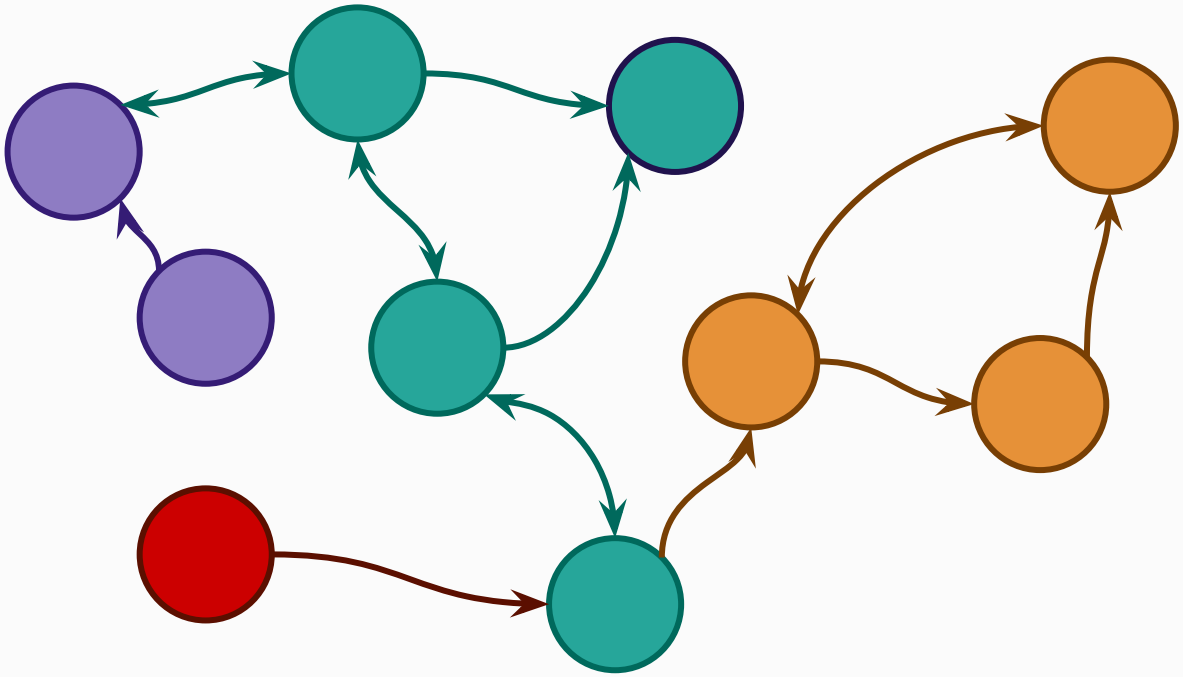


System



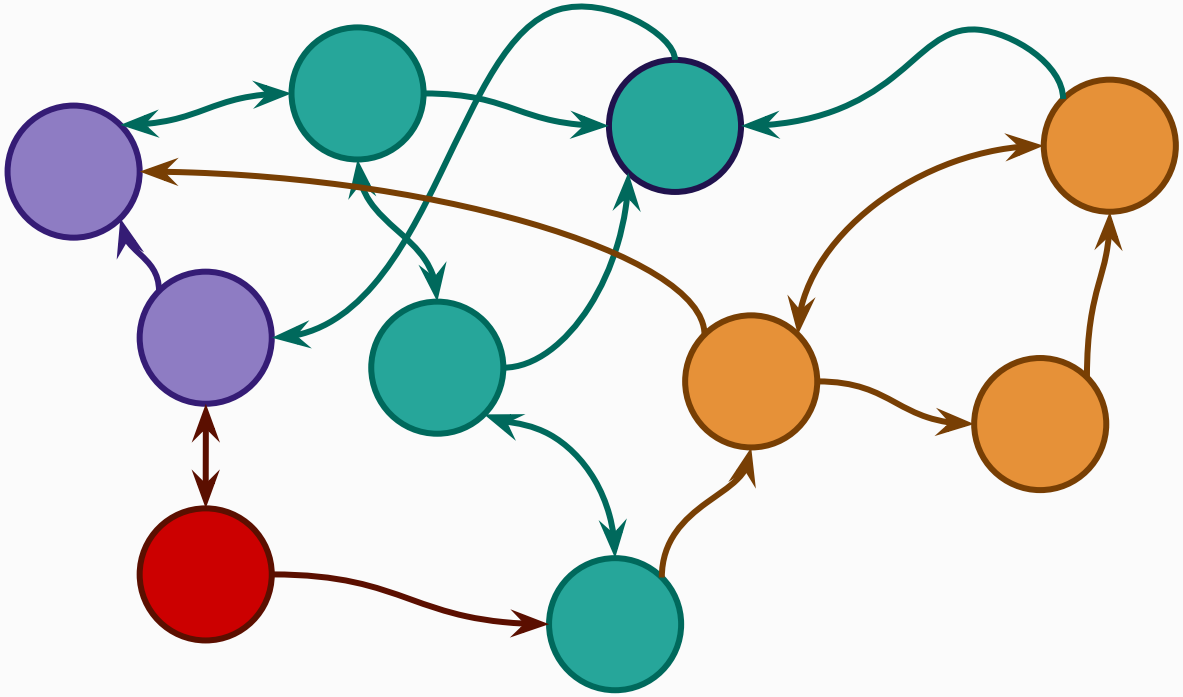
“ A set of **interacting** or **interdependent components** forming a complex whole ”

Software System



“ A set of **interacting** or **interdependent components** forming a complex whole ”

Software System



“ A set of **interacting** or **interdependent components** forming a complex whole ”

What we want to avoid...

What we’d rather have

But how?

Towards services

→ Objects

→ Components

→ Services

Objects

- Encapsulate methods and attributes in a **single software unit.**
- Needs to be compiled or interpreted
- Made to interact with other Objects

⇒ Assembly at a very low level

Components

- There is not one component definition
  → library, packet, plugin, etc.
- A component is a **black box**
  that is used directly from its **interfaces**
- It is **a part** of the software.

⇒ Assembly at a high level

Services

- A service is a component that is **external** to the system.
- Focuses on a specific functionality.
- A **black box** that is used directly from **interfaces.**
- Need to specify how data is **serialized**.

⇒ Assembly at a high and distributed level

Service Oriented Architecture

Service oriented architecture

→ How to transfer messages?

→ How to find a service and its interfaces?

→ How to handle the obtained data?

Transfer messages

- Services = **distributed** systems
- Communication through **network**

Over **TCP/IP**

- Several protocols

→ RPC, JRMP (for RMI), IIOP (pour CORBA), etc.

Over **HTTP**

- **Web**Services (!)

→ SOAP or REST

Find services and their interfaces

- UDDI: web services discovery

  Centralized directory that can be queried to get information on a service.

- Interface:
  - WDSL for SOAP: XML contract file describing all informations associated to a service.
  - WADL for REST: equivalent to WSDL, not highly used.

Manipulate Data

- WDSL contains all information on the data types
- Serialization in XML or JSON

Web Services

SOAP (Simple Object Access Protocol)

- Standardized by the OMG
- Based on XML
- Can be used with different transfer protocols
- Used in big systems
- Deployment is costly

REST (Representational State Transfer)

- No standard: mostly (good) practices
- Based on the concept of resources
- Uses HTTP for communication
- Used (more or less good) in many open APIs on the web (Twitter, FlickR, Facebook, Instagram, etc…)
- Deployment is easy

WAIT A MINUTE!

One cannot solve everything

Focus on REST

REST and Resources

- Focus on the **data** that is manipulated
- URL is composed from the resources → no verb

  http://myapi.com/**library**
  http://myapi.com/**library**/12/**book**
  http://myapi.com/**library**/12/**book**/42

- Use of HTTP's **CRUD** operations

  POST **C**reate (Update)
  GET **R**ead
  PUT **U**pdate
  DELETE **D**elete

REST and Resources

- Use of HTTP's **CRUD** operations:

POST	**C**reate (Update)
GET	**R**ead
PUT	**U**pdate
DELETE	**D**elete

- Example:

GET	http://myapi.com/**book**	**Retrieve list of books**
POST	http://myapi.com/**book**	**Add a book**
GET	http://myapi.com/**book**/42	**Retrieve book 42**
PUT	http://myapi.com/**book**/42	**Update book 42**
DELETE	http://myapi.com/**book**/42	**Delete book 42**

RESTful APIs

- Client-Server exchange
- Stateless
- Cache the most requested resources
- Resources oriented
- Layers/hierarchy of resources

REST**like** APIs

- Client-Server exchange
- Stateless
- ~~Cache the most requested resources~~
- Resources oriented
- ~~Layers/hierarchy of resources~~

Web APIs

- Client-Server exchange
- ~~Stateless~~
- ~~Cache the most requested resources~~
- Resources oriented
- ~~Layers/hierarchy of resources~~

And for us

Creating a REST* API in Java

- Use the Jersey implementation (https://jersey.com.java)
- Use annotations: @GET, @POST, @Path, @Consumes, etc.
- Generate a **war** through black magic (for now)
- Deploy on an application server (Tomcat, Jetty, etc.)

Demo