Navigating APIs with R: An Introductory Overview

Julia Gustavsen

(julia.gustavsen@agroscope.admin.ch)

Agroscope

Plan for today

- Overview of APIs
- What are REST APIs
- Use cases applicable to data science
- How to use an API with R

A bit about me

👋, PhD microbial ecology

🍁

5 years industry experience in Switzerland– bioinformatics

2 years at Agroscope in IT department

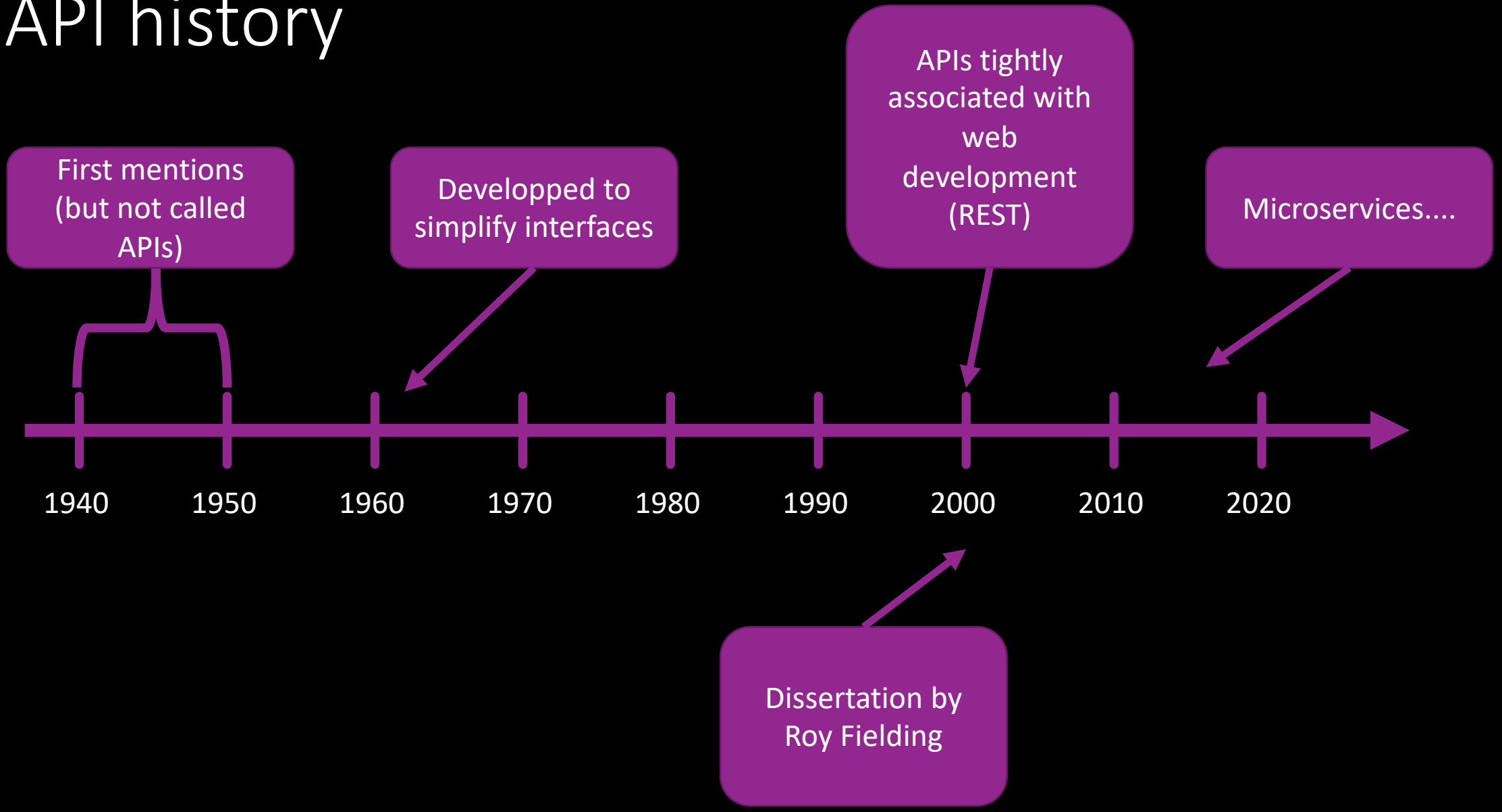
Freetime: hiking, skiing, open source projects such rOpenSci

What is an API?

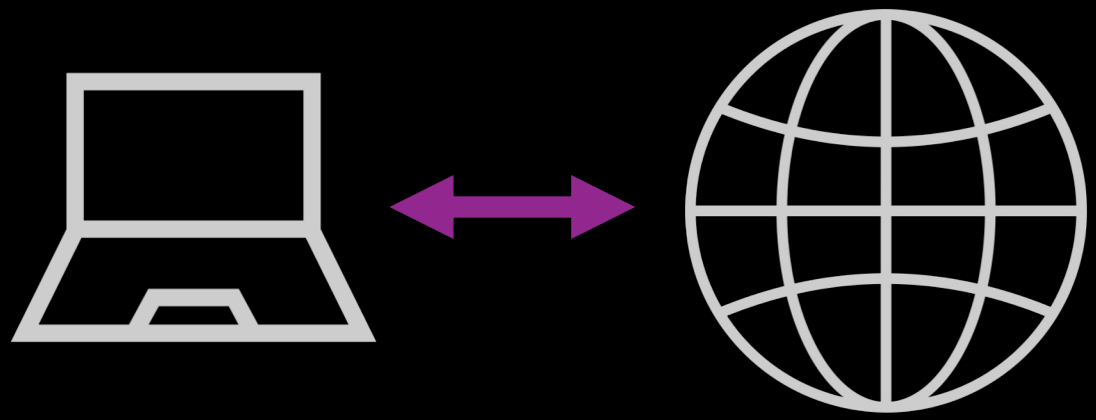
- Application Programmatic Interface.
- What does application programmatic interface mean?
  - Not intended to be directly used by a human, but by a program or software

API makes accessible (often “exposes”) certain parts (e.g. functionality, data, services, etc.) that can be used by others. Idea is that you do not need to understand the internals to be able to use or extract info.

API history

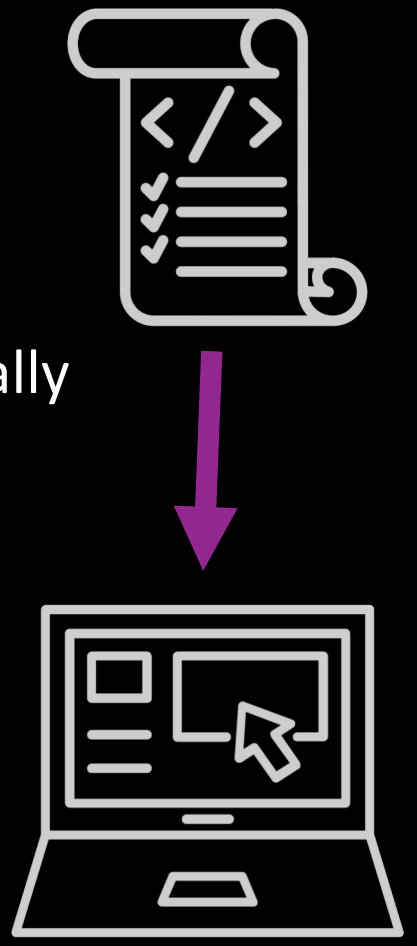

First mentions (but not called APIs)
Developped to simplify interfaces
APIs tightly associated with web development (REST)
Microservices....
1940
1950
1960
1970
1980
1990
2000
2010
2020
Dissertation by Roy Fielding

What can I use an API for?

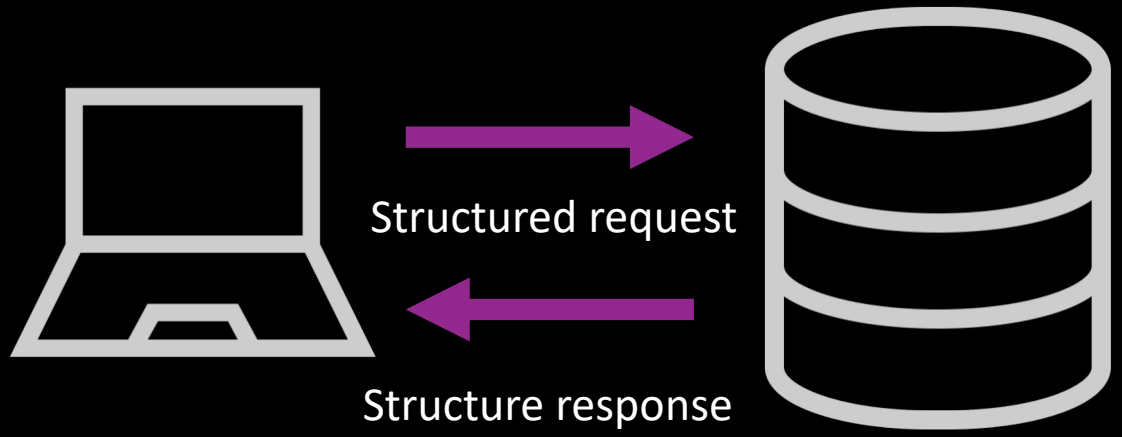


Getting and sending data

Programmatically controllng software

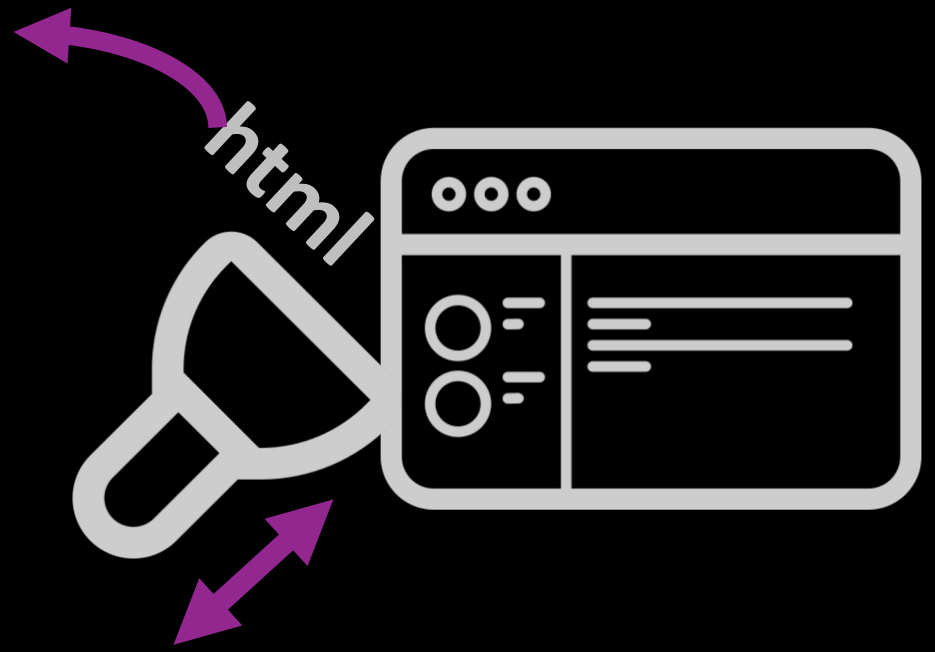


Web API



Structured data exchange

Web scraping



Direct extraction
Html parsing

Photo by Sven Mieke on Unsplash

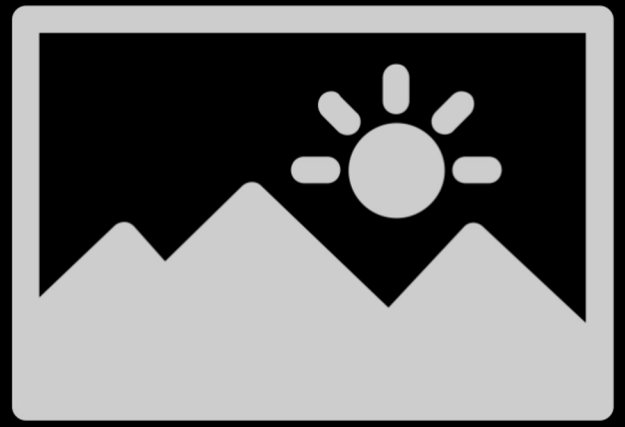
What is a REST API?

- **Representational State Transfer** Application Programmatic Interface
- This is generally the common meaning of API nowadays
- Adhere to a structured set of principles

Representational State transfer? Concept

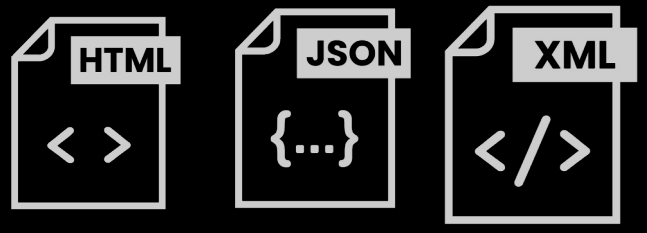
"Client"

Request



Resource

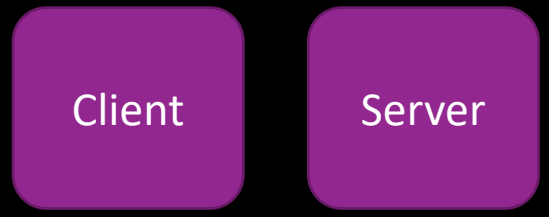
Response



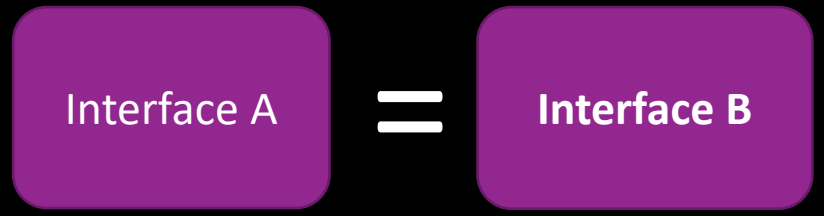
Representation

REST principles

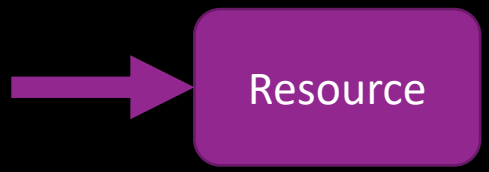
Statelessness



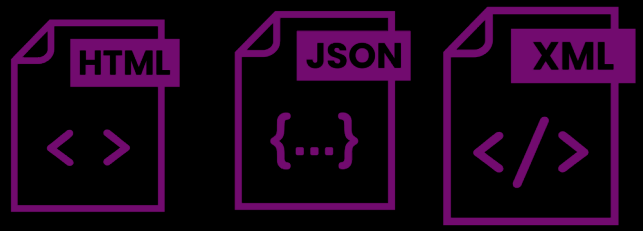
Client-Server Architechture



Uniform interfaces



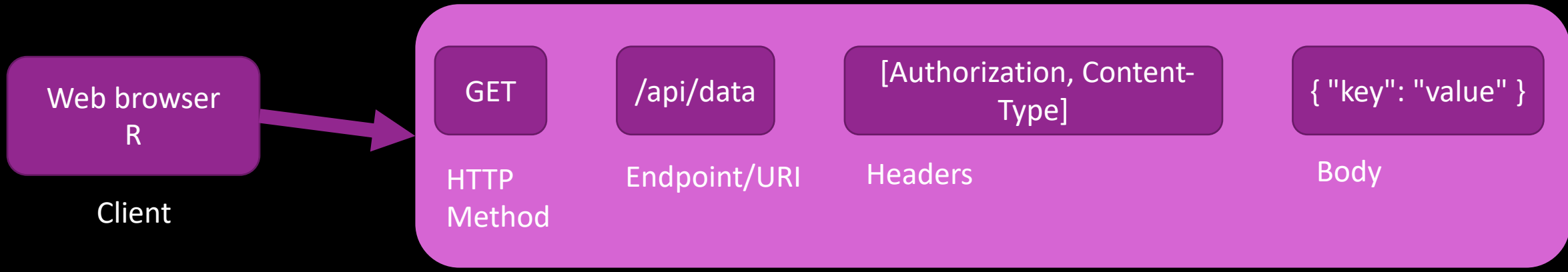
Resource-based



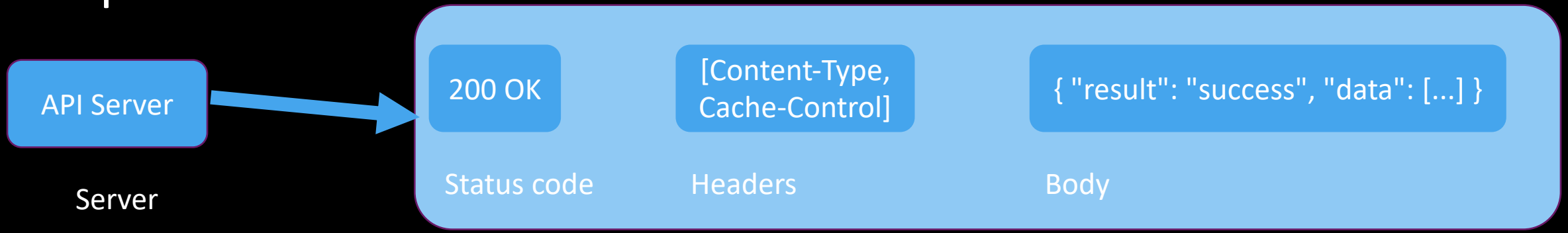
Representations

What is a request / response?

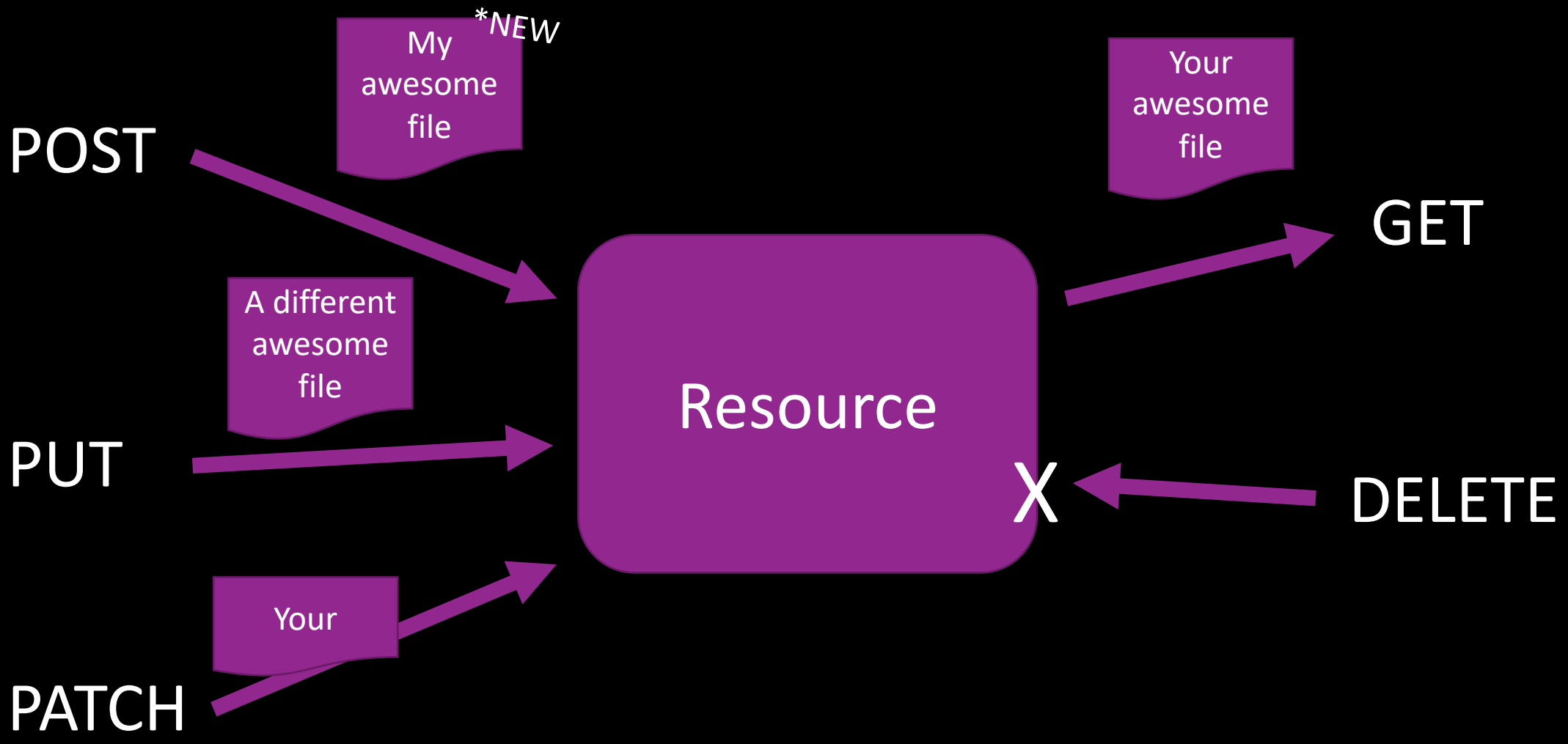
Request



Response



HTTP verbs used in APIs



Why JSON?

{ "key": "value" }

{ "result": "success", "data": [...] }

```
{
"name": "Daniel Müller”,
"age": 30,
"email": ”daniel.mueller@example.com",
 "address": {
"street": "123 Dorfstrasse",
"city": ”Oberhof",
 "zip": ”5062”
 },
"skills": [”R", "Python", "RESTful APIs"],
 "isStudent": false,
 "grades": null
}
```

- Human readable and Lightweight
- Language Agnostic and Interoperable
- Native support in Javascipt and Web Development

So.....using APIs in R

What are the options for manipulating APIs in R?

- many different options (non exhaustive list) :
  - {curl},{crul}, {httr}, {httr2}, {Rcurl}, using system calls from R, etc.
- Today we will mostly use {httr2} in our demos.

What about security for users while using APIs?

- How to manage secrets – do not store passwords or tokens in scripts/packages.
  - Lots of good advice by Hadley Wickam in a httr vignette entitled Managing secrets , starts with storing in .Renviron file
    - also {secret}, {getPass}, {keyring}, {cyphr}

- Sensitive data – use only HTTPS
- Data privacy, understand where your data are headed....

R Demos

- Use cases for APIs that I am familiar with:
  - Traditional web APIs: examples: Github, DeepL
  - Software with GUI that could be automated: {RCy3} for Cytoscape

Cytoscape

- Open source Software that is used for visualizing networks
- Can use desktop software with graphical user interface
- Also REST API available and Python library and R package that wrap the REST API
  - In R this is {RCy3}

A large amount of development has gone into providing automation possibilities in Cytoscape

Agroscope use cases

- Deepl
  - Note: this is outside of the Confederation's DeepL pro account
  - Use case: IT ticketing system responses, used as part of some prototyping in preparation for input into NLP

- Firecrest
  - API for the Swiss National Supercomputer Center
    - Allows us to transfer data from our research network
      - There is a recommended tool, Globus, but it does not work with the proxy, so this is a great alternative for us.
    - Allows us to launch jobs

Conclusion

- 🚀 **First steps:** Today's talk and demos were a basic introduction to APIs and interacting with APIs using R.
- 🌐 **Endless possibilities:** APIs are the gateway to a vast world of data and services on the web. Many creative possibilities exist...

Questions?

Slides and code available at: https://github.com/jooolia/APIs_with_R_talk_tutorial

APIs is R

There are several R packages that have been widely used for dealing with REST APIs. These packages facilitate making HTTP requests, handling responses, and parsing data in R. Here are some popular ones:

1. **httr:**
   - **Description:** A versatile package that provides functions for working with URLs, HTTP methods, headers, and other aspects of HTTP.
   - **Example:**
     ```R
     library(httr)
     response <- GET("https://api.example.com/data")
     content(response)
     ```

2. **curl:**
   - **Description:** Offers a modern and flexible R interface to libcurl, a widely used library for handling HTTP requests.
   - **Example:**
     ```R
     library(curl)
     handle <- new_handle()
     response <- curl_fetch_memory("https://api.example.com/data", handle = handle)
     ```

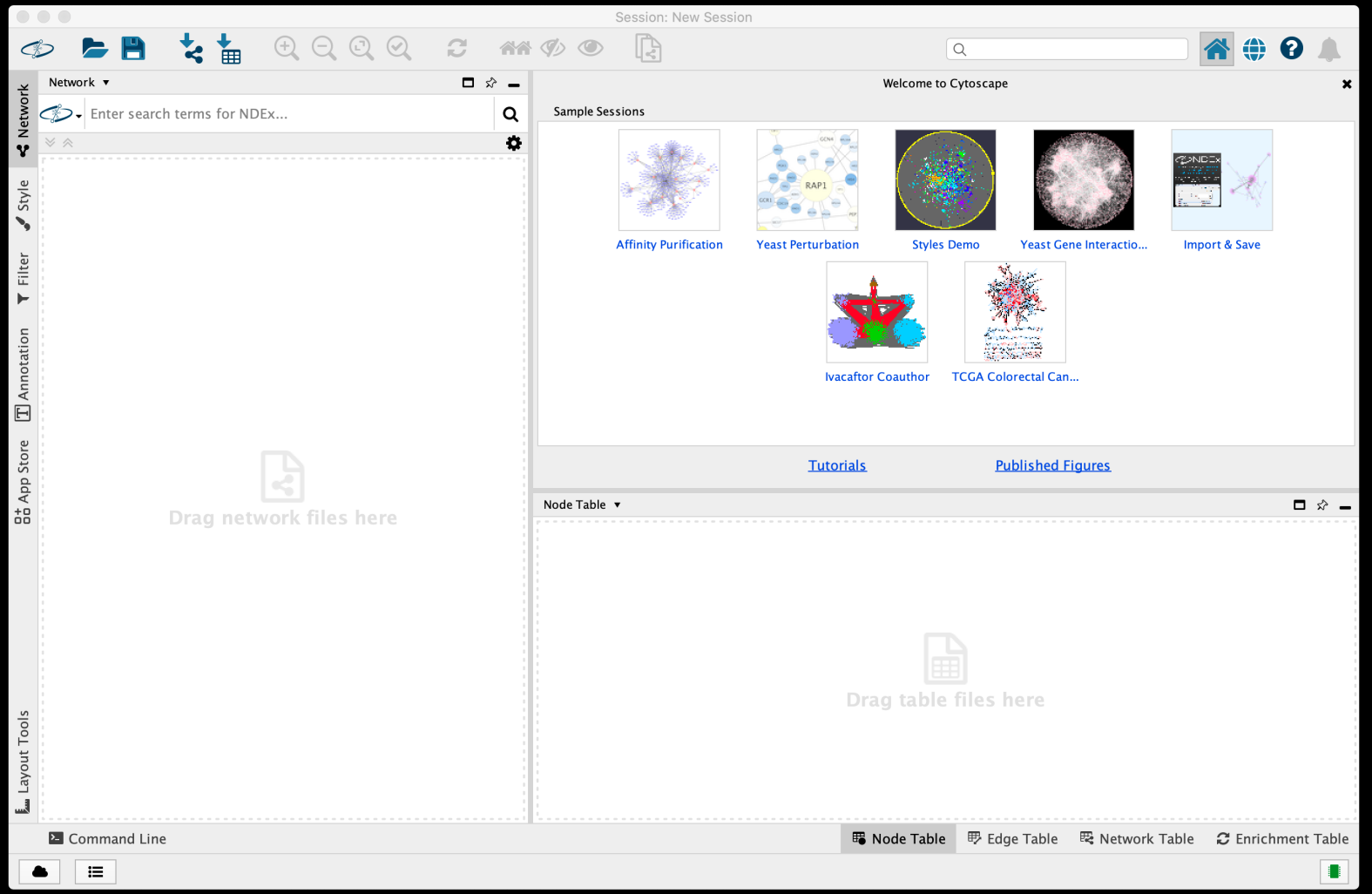
3. **jsonlite:**
   - **Description:** A lightweight package for JSON parsing and generation. Useful for handling JSON data often returned by REST APIs.
   - **Example:**
     ```R
     library(jsonlite)
     data <- fromJSON('{"name": "John", "age": 30}')
     ```

REST vs non-REST API

	REST API	non-REST API
Architectural style	Adheres to principles of REST Stateless communication primordial	Can follow other architechtural designs such as SOAP, GraphQL or others.
Communication protocol	Generally HTTP mehtods Standard HTTP status codes	can use other protocols
Data format	often use JSON or XML for representation Resrouces identified by URIs(uniform resrouce identifiers)	can use XML or JSON or other formats
Documentation standards	Standardized conventions available and tools like Swagger or OpenAPI available	Can have own documenation standards..
Endpoint design	Uses resrouce-based encpoitns and standard HTTP methods	can vary..

Cytoscape screen shots

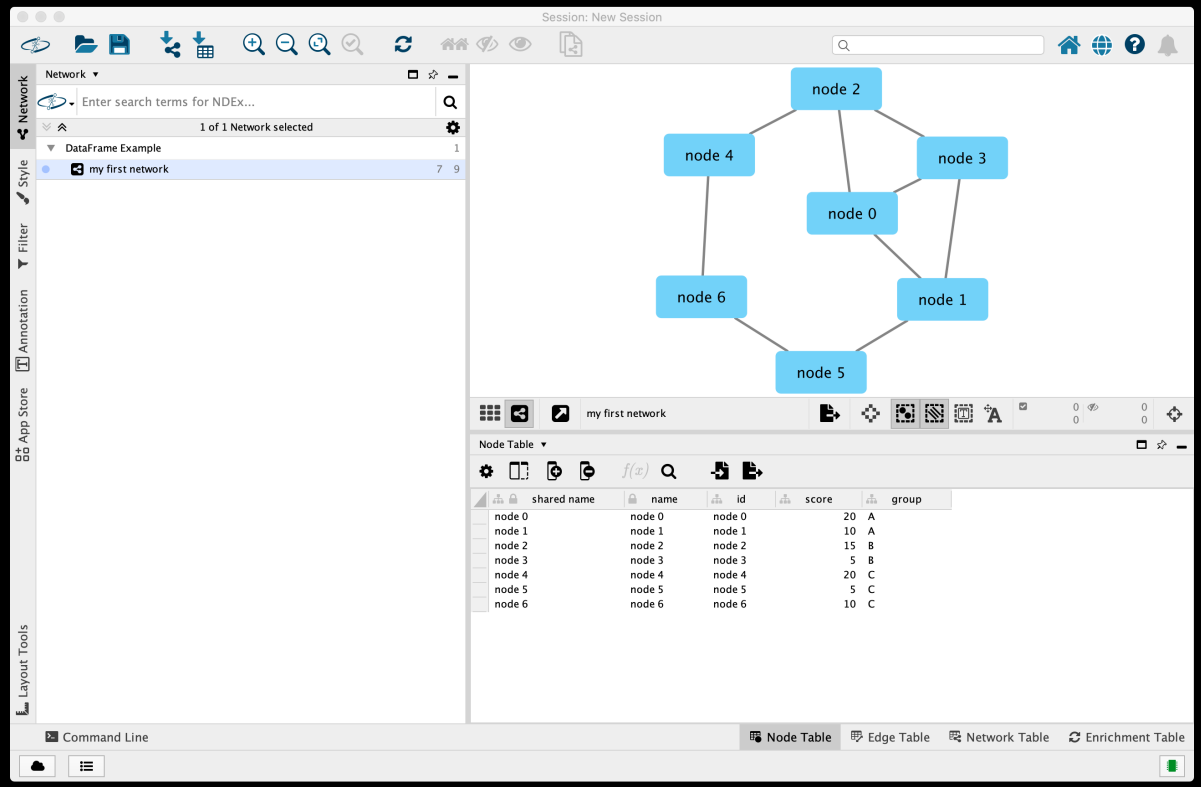
- Open Cytoscape



```r
nodes <- data.frame(id = c("node 0", "node 1", "node 2", "node 3", "node 4", "node 5", "node 6"),
                    group = c("A", "A", "B", "B", "C", "C", "C"), # categorical strings
                    score = as.integer(c(20, 10, 15, 5, 20, 5, 10)), # integers
                    stringsAsFactors = FALSE)
edges <- data.frame(source = c("node 0", "node 0", "node 0", "node 2", "node 2", "node 5", "node 4", "node 5", "node 1"),
                    target = c("node 1", "node 2", "node 3", "node 3", "node 4", "node 6", "node 6", "node 1", "node 3"),
                    weight = c(5.1, 3.0, 5.2, 9.9, 3.5, 4.5, 6, 5.3, 3), # numeric
                    stringsAsFactors = FALSE)


simple_graph_ig <- igraph::graph_from_data_frame(edges, vertices = nodes)
plot(simple_graph_ig)

createNetworkFromDataFrames(nodes, edges,
                            title = "my first network",
                            collection = "DataFrame Example")
```



```
style.name = "myStyle"
defaults <- list(NODE_SHAPE = "diamond",
                 NODE_SIZE = 30,
                 EDGE_TRANSPARENCY = 120)
edgeWidth <- mapVisualProperty('edge width','weight','p')
nodeFill <- mapVisualProperty('node fill
color','score','c',c(5,10,20),c('#99CCFF','#FFFFFF','#FF7777'))
nodeLabel <- mapVisualProperty('node label','group','p')


createVisualStyle(style.name, defaults, list(edgeWidth, nodeFill, nodeLabel))
setVisualStyle(style.name)
```