# INFO-H420
# Management of Data Science and Business Workflows

## Part II
## Data Pipelines

Dimitris SACHARIDIS

2023-2024

# Part II: Management of Data Science Workflows

- Introduction to Data Science Workflows
- Data Privacy
- Fairness
- Explainability

# Data Science Workflows

# Data Science – Definition

*“**Data science** is the study of the generalizable extraction of knowledge from data.”*

*- Vasant Dhar*

- The term **science** implies knowledge gained through systematic study.
- A data scientist requires an **integrated skill set** spanning mathematics, machine learning, artificial intelligence, statistics, databases, and optimization, along with a deep understanding of the craft of problem formulation to engineer effective solutions.

Data Science and Prediction. Vasant Dhar. Communications of the ACM 56(12), 2013.
https://doi.org/10.1145/2500499

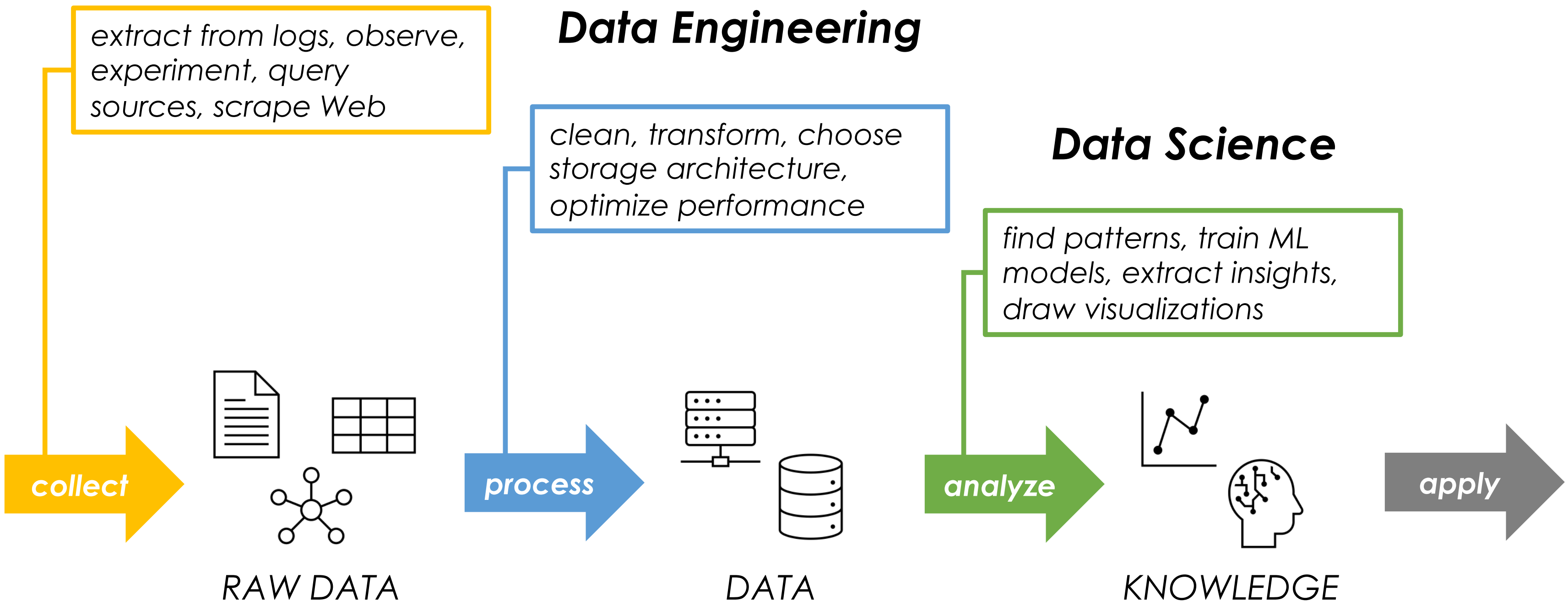
# Data Science – Definition

Data science is an **interdisciplinary field that uses scientific methods**, **processes**, **algorithms and systems to extract** or extrapolate **knowledge** and insights **from** noisy, structured and unstructured **data**, **and apply knowledge from data** across a broad range of application domains.

Data science is related to **data mining**, **machine learning**, **big data**, **computational statistics**, and **analytics**.

- *Wikipedia*

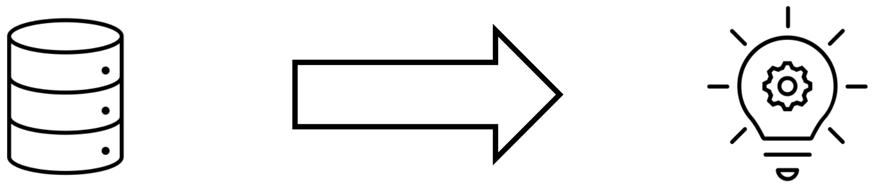
# The Data Lifecycle



# Data Science vs. Data Engineering

*Same Goal*

**convert DATA to KNOWLEDGE**

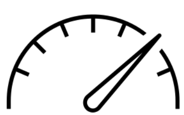


*DATA* *KNOWLEDGE*

*Different Focus*

**Data Engineering**

*is it fast?*



**Data Science**

*is it good?*

★★★

# Data Science Tasks


| Input | Task | Output |
|---|---|---|
| Relational Table | SQL query | Relational Table |
| Unstructured Data | Information Extraction | Structured Data |
| Labelled Data | ML training | ML Model |
| ML Model | Fine Tuning | ML Model |
| Raw Data | Preprocessing | Clean Data |
| Structured Data | Visualization | Chart |

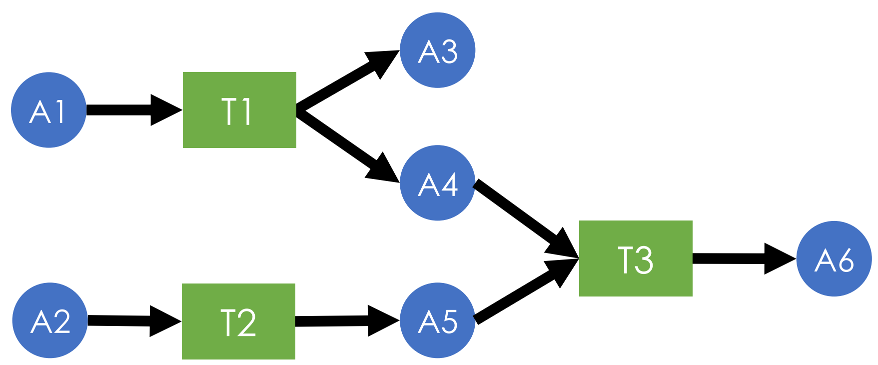
Because the **Input** and **Output** of a Task can have various forms/structures/types (data, chart, text, model), we simply call them **Artifacts**

# Data Science Workflows

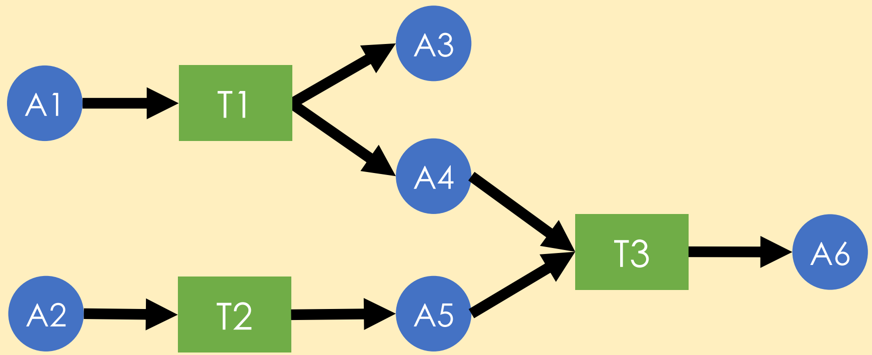
A Data Science **Workflow**, or Data **Pipeline**, is a sequence of

- artifacts
- tasks

Typically portrayed as a **graph**

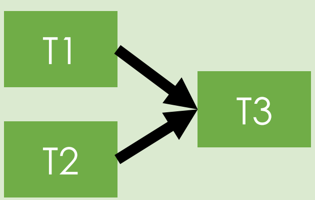


# Data Science Workflows



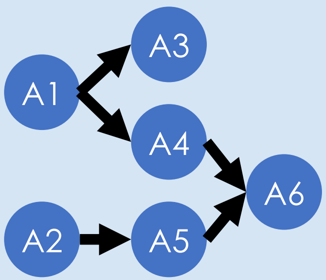

**Task-Artifact Graph**

- shows **flow** among tasks and artifacts
- two types of nodes: **tasks**, **artifacts**
- edges encode **input requirements**, and **outputs**




**Task Graph**

- shows **dependencies** among **tasks**
    - *task dependency graph*
- artifacts are hidden in edges




**Artifact Graph**

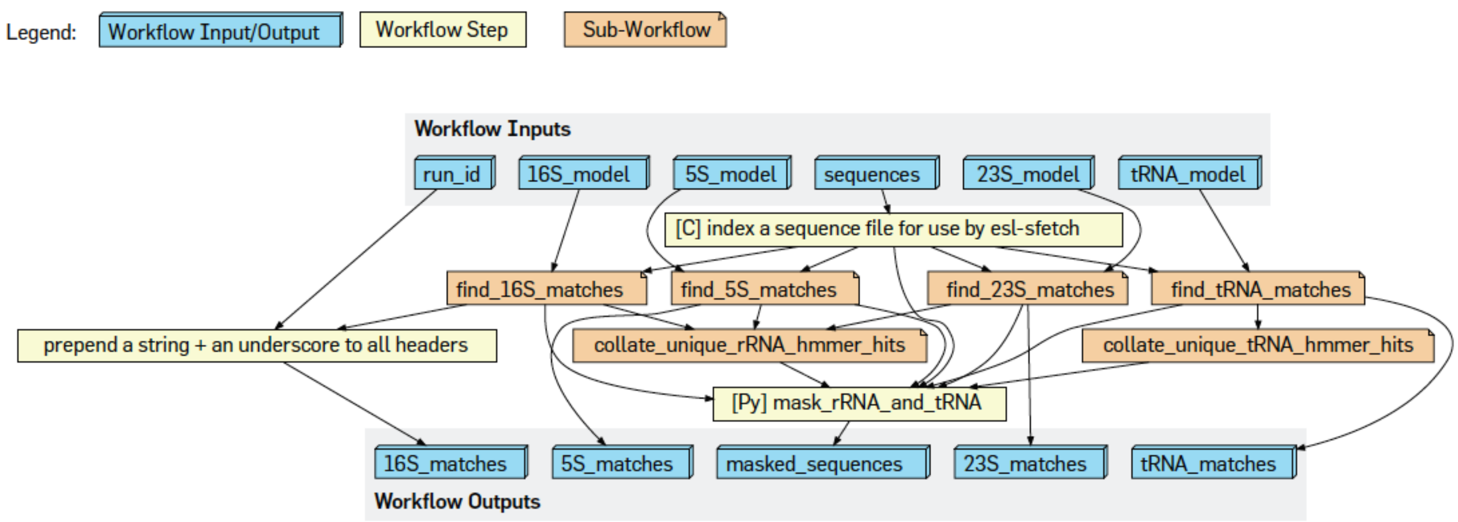
- shows relationships among **artifacts**
- edges encode **tasks**
- tasks are hidden in edges

- all graphs are **directed acyclic graphs** (DAGs)
    - directed edges indicate flow, dependency relationships
    - no (directed) cycles

# Examples of Workflows

Workflows are very popular in Bioinformatics

Example of a **task-artifact** graph for a workflow that matches inputs of genomic sequences to provided sequence models, expressed in the Common Workflow Language (CWL)




Methods included: standardizing computational reuse and portability with the Common Workflow Language. Crusoe et al. Communications of the ACM 65(6), 2022. https://dl.acm.org/doi/10.1145/3486897

# Examples of Workflows

Database queries result in execution plans

Consider the execution of the query:

MODEL = “CIVIC” AND YEAR = 2001 AND
(COLOR = “GREEN” OR COLOR = “WHITE”)

on the following table:

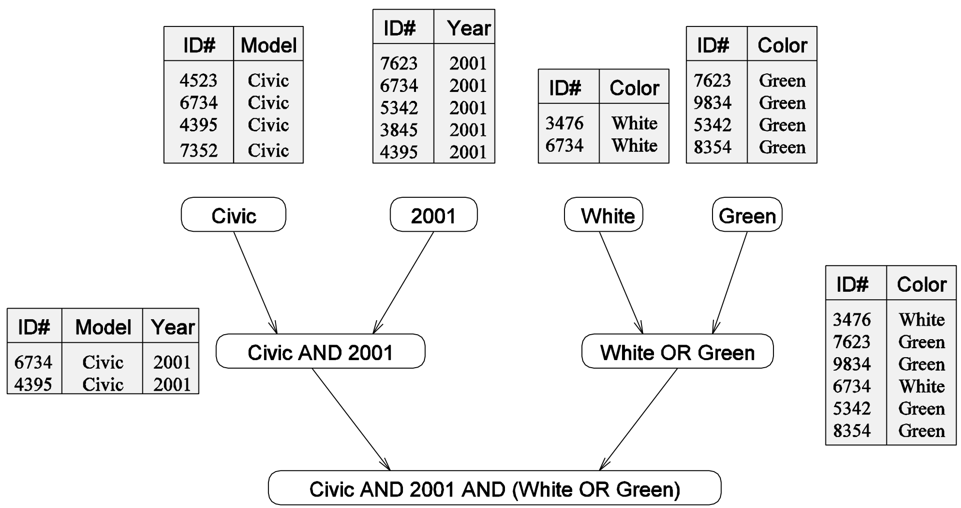
| ID# | Model | Year | Color | Dealer | Price |
|---|---|---|---|---|---|
| 4523 | Civic | 2002 | Blue | MN | $18,000 |
| 3476 | Corolla | 1999 | White | IL | $15,000 |
| 7623 | Camry | 2001 | Green | NY | $21,000 |
| 9834 | Prius | 2001 | Green | CA | $18,000 |
| 6734 | Civic | 2001 | White | OR | $17,000 |
| 5342 | Altima | 2001 | Green | FL | $19,000 |
| 3845 | Maxima | 2001 | Blue | NY | $22,000 |
| 8354 | Accord | 2000 | Green | VT | $18,000 |
| 4395 | Civic | 2001 | Red | CA | $17,000 |
| 7352 | Civic | 2002 | Red | WA | $18,000 |

# Examples of Workflows

Database queries result in execution plans.

Each task can be thought of as generating an intermediate table of entries that satisfy a particular clause.

Here's an **artifact graph**.




| ID# | Model |
|---|---|
| 4523 | Civic |
| 6734 | Civic |
| 4395 | Civic |
| 7352 | Civic |

| ID# | Year |
|---|---|
| 7623 | 2001 |
| 6734 | 2001 |
| 5342 | 2001 |
| 3845 | 2001 |
| 4395 | 2001 |

| ID# | Color |
|---|---|
| 3476 | White |
| 6734 | White |

| ID# | Color |
|---|---|
| 7623 | Green |
| 9834 | Green |
| 5342 | Green |
| 8354 | Green |

| ID# | Model | Year |
|---|---|---|
| 6734 | Civic | 2001 |
| 4395 | Civic | 2001 |

| ID# | Color |
|---|---|
| 3476 | White |
| 7623 | Green |
| 9834 | Green |
| 6734 | White |
| 5342 | Green |
| 8354 | Green |

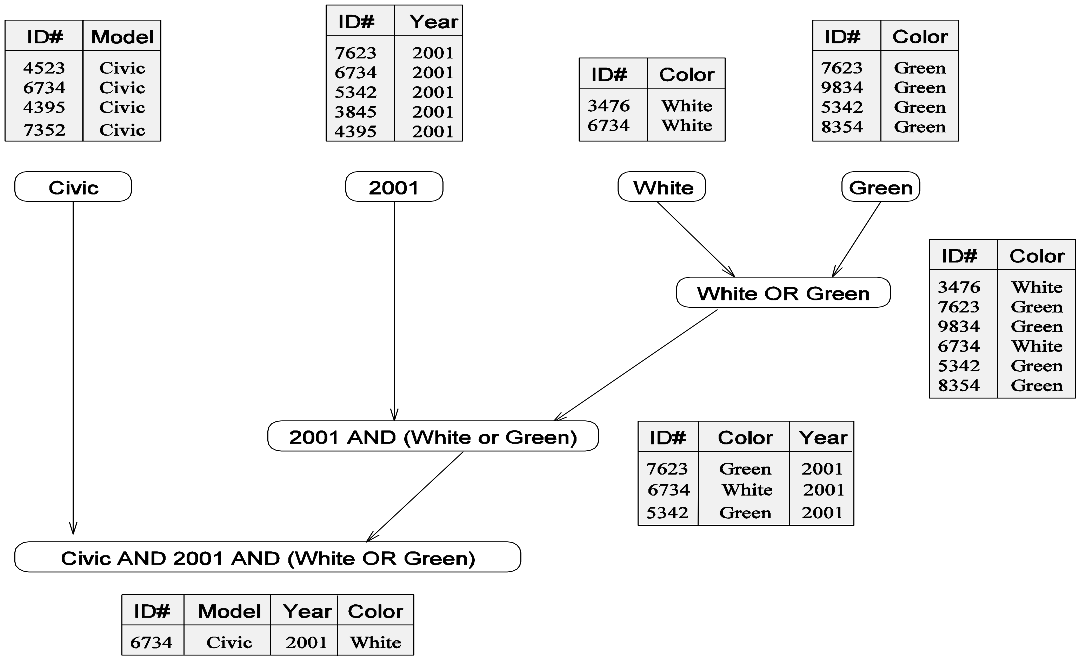
| ID# | Model | Year | Color |
|---|---|---|---|
| 6734 | Civic | 2001 | White |

# Examples of Workflows

Database queries result in execution plans.

An alternate decomposition of the given problem into subtasks, along with their data dependencies.

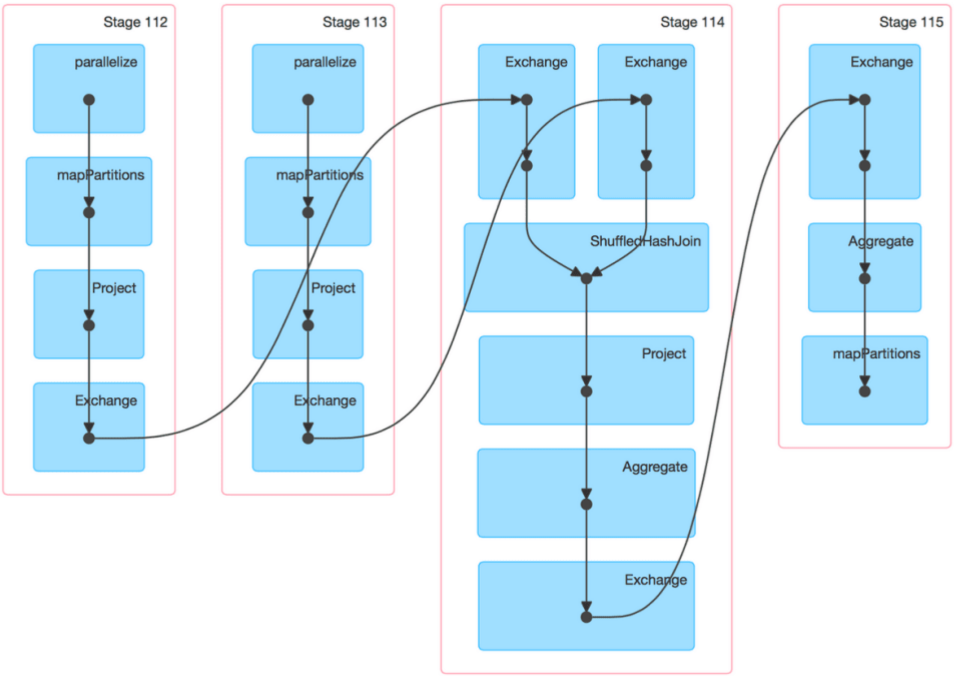
Here's an **artifact graph**.



# Examples of Workflows

Big Data analytics is based on **parallel execution** of task graphs

Example of a **task graph** in Spark



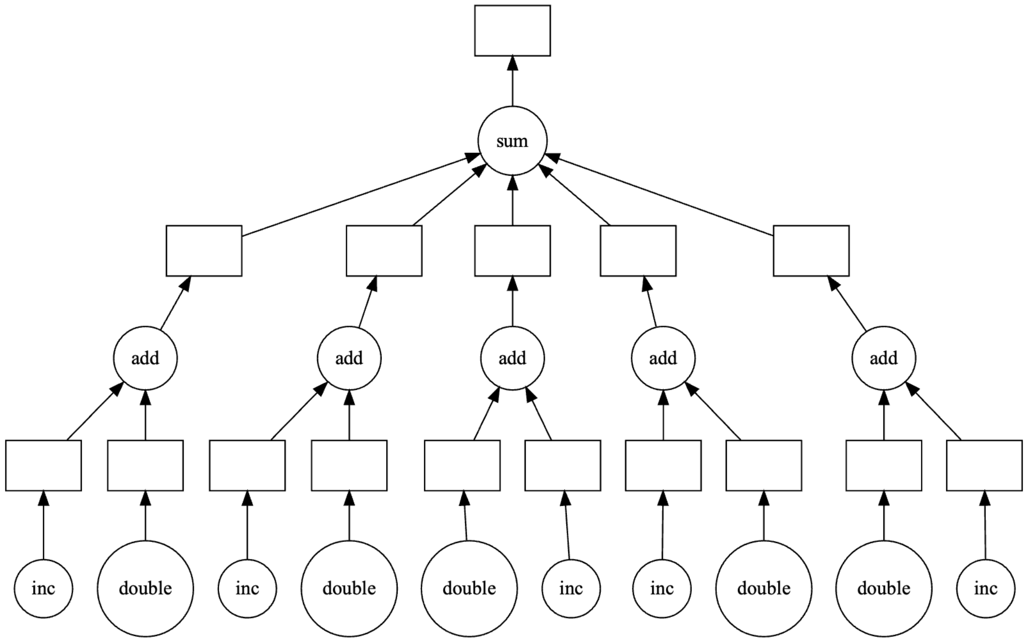

https://www.databricks.com/blog/2015/06/22/understanding-your-spark-application-through-visualization.html

# Examples of Workflows

Big Data analytics is based on **parallel execution** of task graphs

Example of a **task-artifact** graph in Dask

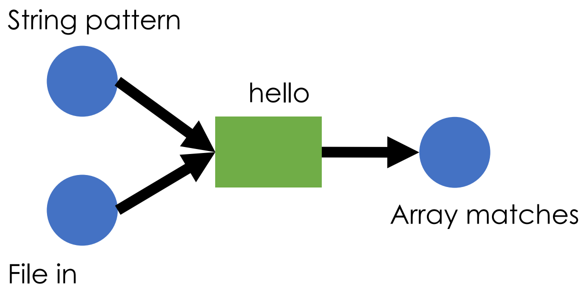
```
import dask

@dask.delayed
def inc(x):
    return x + 1

@dask.delayed
def double(x):
    return x * 2

@dask.delayed
def add(x, y):
    return x + y

data = [1, 2, 3, 4, 5]

output = []
for x in data:
    a = inc(x)
    b = double(x)
    c = add(a, b)
    output.append(c)

total = dask.delayed(sum)(output)
```



# Workflow Languages

- Workflows are typically described in some standardized language
- The **Workflow Description Language** (WDL) aims to have a very human-readable and -writeable syntax
  - A wdl file has syntax similar to Python

```
task hello {
  input {
    String pattern
    File in
  }

  command {
    egrep '${pattern}' '${in}'
  }

  runtime {
    docker: "broadinstitute/my_image"
  }

  output {
    Array[String] matches = read_lines(stdout())
  }
}

workflow wf {
  call hello
}
```




[https://github.com/openwdl/wdl](https://github.com/openwdl/wdl)

# Workflow Languages

- The **Common Workflow Language** (CWL) is written in YAML markup

***a task***

```yaml
cwlVersion: v1.0
class: CommandLineTool

doc: Spoa is a partial order aligment…

inputs:
  readsFA:
    type: File
    format: edam:format_1929
    doc: FASTA file containing a set of sequences…

requirements:
  InlineJavascriptRequirement: {}
hints:
  DockerRequirement:
    dockerPull: "quay.io/biocontainers/spoa:3.4.0--hc9558a2_0"
  ResourceRequirement:
    ramMin: $(15 * 1024)
    outdirMin: $(Math.ceil(inputs.readsFA.size/(1024*1024*1024) + 20))

baseCommand: spoa

arguments: [ $(inputs.readsFA), -G, -g, '-6' ]

stdout: $(inputs.readsFA.nameroot).g6.gfa

outputs:
  spoaGFA:
    type: stdout
    format: edam:format_3976
    doc: result in Graphical Fragment Assembly (GFA) format

$namespaces:
  edam: http://edamontology.org
```

1. Community Maintained File Format Identifier

2. Software Container

3. Dynamic Resource Requirements

C

***a workflow***

```yaml
cwlVersion: v1.0
class: Workflow

inputs:
  pattern: string
  sample_data: File[]

steps:
  find_matches:
    run: grep.cwl
    in:
      pattern: pattern
      files: sample_data
    out: [ text_matches ]

  count_lines:
    run: wc.cwl
    in:
      file: find_matches/text_matches
    out: [ lines ]

outputs:
  number_of_matches:
    type: int
    outputSource: count_lines/lines
```

Methods included: standardizing computational reuse and portability with the Common Workflow Language. Crusoe et al. Communications of the ACM 65(6), 2022. https://dl.acm.org/doi/10.1145/3486897

# Why describe Workflows

- One important reason is to enable Open Science and adhere to the **FAIR** Guiding Principles:
  - Findable, Accessible, Interoperable, Reusable

the three R's: Repeatability, Reproducibility, Reusability

- **repeat** the workflow with same input on same environment and get the same output
- **reproduce** the output of the workflow with the same input on a different environment
- **reuse** workflow, or parts of it, to solve a different problem

# Workflow Registries

- A Workflow Registry is a Platform where one can
  - **register** workflows
    - upload a workflow description, in some supported language
    - receive a unique persistent identifier (pid)
  - **find** and **reuse** workflows
    - explore registered workflows
    - download workflows in supported languages
  - (**execute** workflows)

Dockstore
Create, Share, Use

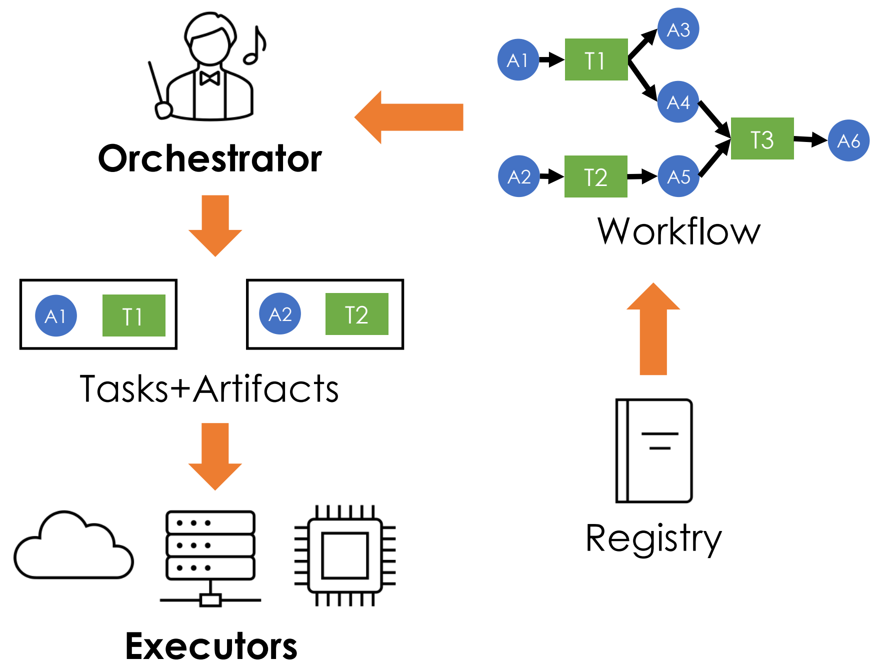
WorkflowHub

# Executing Workflows

# Executing Workflows

- Data Science Workflows are **compute**- and **data**-intensive
- Require heavy computational resources
  - computational clusters, high-performance computing (HPC) machines on premises or in the cloud

- To execute a workflow, you need:
  - An **Orchestrator** to decide how and when to assign tasks to Executors
  - **Executors** that run Tasks



# Cloud Computing

- Cloud computing refers to the delivery of **computing services** such as storage, networking, and computing power **over the internet** (the "cloud")
- Cloud computing allows users to access and use these services on-demand, without having to manage the **underlying infrastructure** themselves
- This can be more cost-effective and scalable than maintaining your own **on-premises** infrastructure
- There are different types of cloud computing
  - **Public cloud** services are available to the general public, typically on a pay-as-you-go basis
  - **Private cloud** services are operated exclusively for a single organization, often on-premises
  - **Hybrid cloud** refers to a combination of public and private cloud

# Cloud Computing

- Common examples of cloud computing services include **Infrastructure-as-a-Service** (IaaS), **Platform-as-a-Service** (PaaS), and **Software-as-a-Service** (SaaS)
- IaaS provides users with access to **fundamental computing resources** such as virtual machines, storage, and networking
- PaaS provides users with a **platform for developing and deploying** their own applications, without having to manage the underlying infrastructure
- SaaS provides users with access to **software applications** that are hosted and managed by the provider

# Cloud Computing Platforms

- Amazon Web Services (AWS) (since 2006)
- Microsoft Azure (since 2010)
- Google Cloud Platform (since 2008)

# Kubernetes

- Kubernetes (abbreviated as k8s) is an open-source platform for **managing** and **orchestrating** containerized applications
  - applications run on-premises, in the cloud, or in a hybrid environment
  - In containers, such as Docker containers
- Kubernetes uses a declarative approach, where users specify the **desired state** of their applications and Kubernetes automatically ensures that the application matches that state
- Kubernetes is **highly scalable**, allowing users to easily add or remove containers and resources as needed

kubernetes

# Kubernetes

- Created by Google, open sourced and donated to the Cloud Native Computing Foundation
  - Comes from the Greek word for helmsman

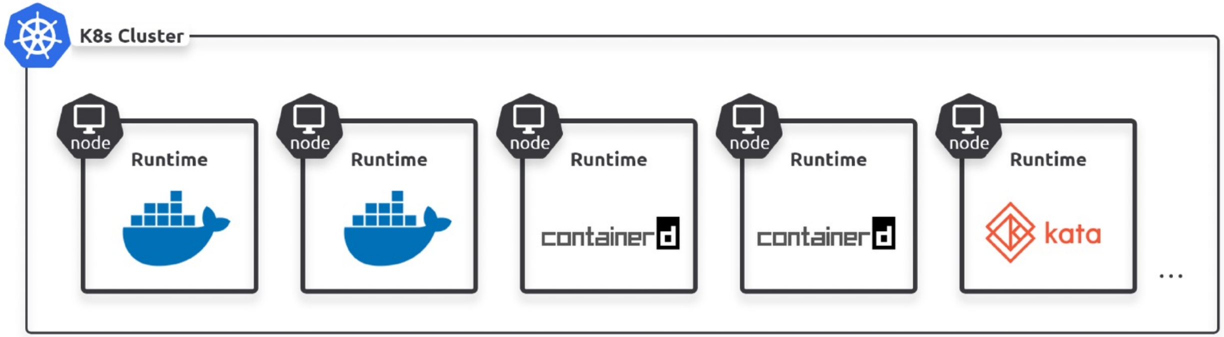
- Kubernetes is considered the **operating system of the cloud**
  - a traditional OS on a server **abstracts server resources** and **schedules applications**
  - Kubernetes on a cloud **abstracts cloud resources** and **schedules containerized applications**

# Containers

- A **containerized application** is an app packaged and run in a container
  - Makes it portable, scalable, and easier to deploy and manage
- A **container** is a lightweight, standalone, and executable **package** of software that includes **everything needed to run the application**, such as the code, libraries, dependencies, and runtime
- Containers are **isolated** from each other and from the host operating system, allowing them to run consistently across different environments
- A popular tool for creating and managing containers is Docker

# Kubernetes Cluster

- Kubernetes runs on a cluster in the cloud, where a group of nodes are used to run containerized applications
  - A node is a physical or virtual machine, provided by the cloud platform, that runs the Kubernetes software and hosts containers
- A cluster typically consists of at least one **master** node and multiple **worker** nodes.
- The master node runs the Kubernetes **control plane**, which is responsible for managing and coordinating the worker nodes.
- The worker nodes run the Kubernetes **kubelet**, which is responsible for running and managing the containers on that node.



# Kubernetes is an Executor

In the context of Data Science workflows

- A **Task** (with its input artifacts) can be delivered as a Containerized Application
- And Kubernetes can act as the **Executor**, running the Tasks on the worker nodes, abstracting the underlying computational infrastructure

- But Kubernetes is not an **Orchestrator** of workflows
  - Only sees/knows of individual Tasks

# Orchestrators of Workflows

# Workflow Orchestrators

- Orchestrators **deploy** Workflows on Executors (the execution infrastructure)
- Provide the features and tools to **monitor** and **manage** workflows

- Examples are Kubeflow and Airflow

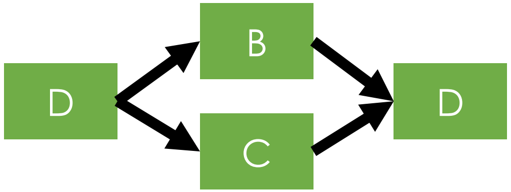
# Kubeflow + Argo Workflows

- Kubeflow is an open-source platform for deploying and managing workflows on Kubernetes
  - Created by Google
- Focused on machine learning (ML) workflows, provides a set of tools and components to make it easier to develop, train, and deploy ML models at scale
- Kubeflow works with Argo workflows, an open-source tool for orchestrating parallel and sequential workflows on Kubernetes
- Argo Workflows uses Kubernetes resources to execute the tasks in a workflow

# Argo Workflows

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: dag-diamond-
spec:
  entrypoint: diamond
  templates:
  - name: echo
    inputs:
      parameters:
      - name: message
    container:
      image: alpine:3.7
      command: [echo, "{{inputs.parameters.message}}"]
  - name: diamond
    dag:
      tasks:
      - name: A
        template: echo
        arguments:
          parameters: [{name: message, value: A}]
      - name: B
        dependencies: [A]
        template: echo
        arguments:
          parameters: [{name: message, value: B}]
      - name: C
        dependencies: [A]
        template: echo
        arguments:
          parameters: [{name: message, value: C}]
      - name: D
        dependencies: [B, C]
        template: echo
        arguments:
          parameters: [{name: message, value: D}]
```

- Argo Workflows uses a YAML-based language
- Here a diamond-shaped **task graph** is defined



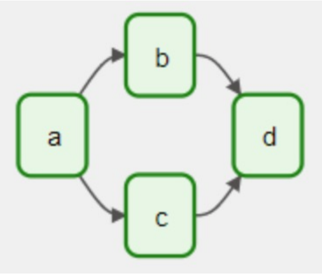
# Airflow

- Apache Airflow is an open-source platform for scheduling and orchestrating workflows
  - Created by Airbnb, now open-source
- Airflow uses a DAG to represent the tasks in a workflow and their dependencies (task-graph)
- Airflow provides a **web-based** user interface for managing and monitoring the workflows, as well as a command-line interface and **Python-based API**
- Available at major Cloud Platforms
- Works with various Executors
  - including Kubernetes

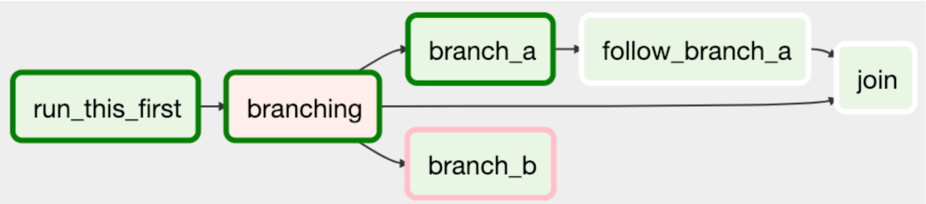
# Airflow

- A DAG can be declared in Python, and viewed in the web-based interface
- Here's the diamond task graph

```
first_task >> [second_task, third_task]
third_task << fourth_task
```



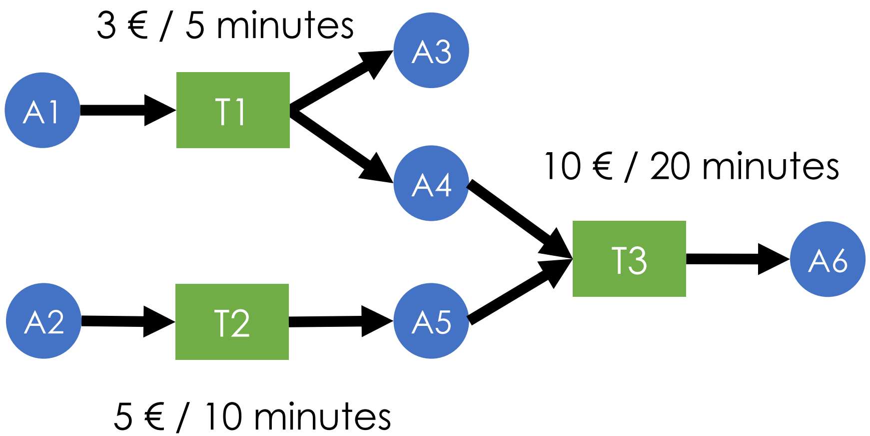

- Airflow also support conditional execution (branching)



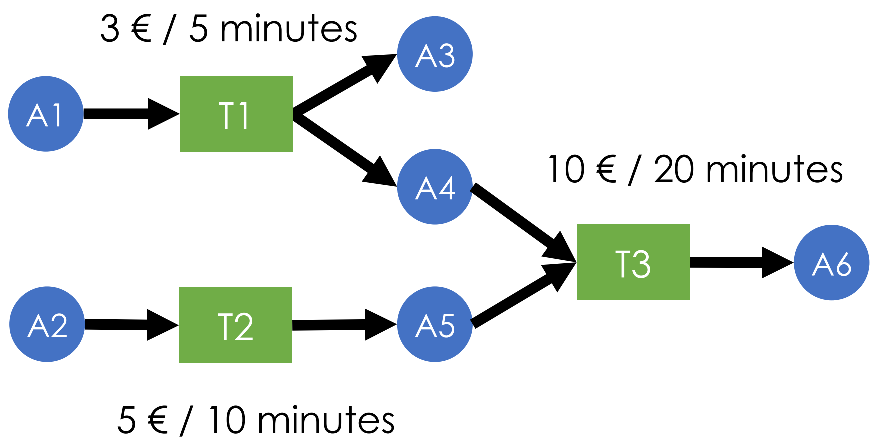
# Workflow Optimization

# Task Cost Measures

- Executing Tasks consumes resources
- Each task has two cost measures
  - the **cost**, e.g., in terms of money,
  - the **time** it takes to execute (response time, duration, cycle time)
- We can indicate these measures on the tasks in a dag



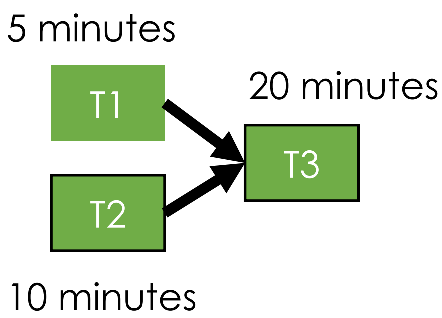
# Workflow Cost Measures



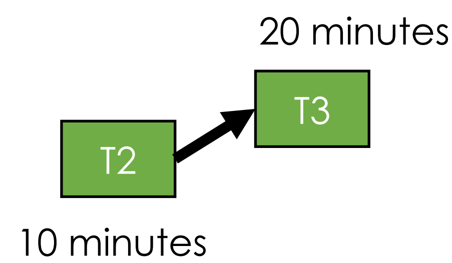

- What is the **total cost** of running this workflow?
  - No surprises, all tasks have to be executed, so 18 €
- What is the **total time** of running this workflow?
  - It depends on how you orchestrate it!
  - What tasks can run in parallel?

# Critical Path in Workflows

- The **critical path** is the sequence of tasks that determines the **minimum time required** to execute the workflow
- It is called the "critical" path because if any of the tasks on this path are delayed, it will delay the entire workflow
- It corresponds to a **longest path** on the task graph
- We define the workflow's **total time** as the time of a critical path
  - On this workflow, the total time is 30 minutes




task graph




a critical path

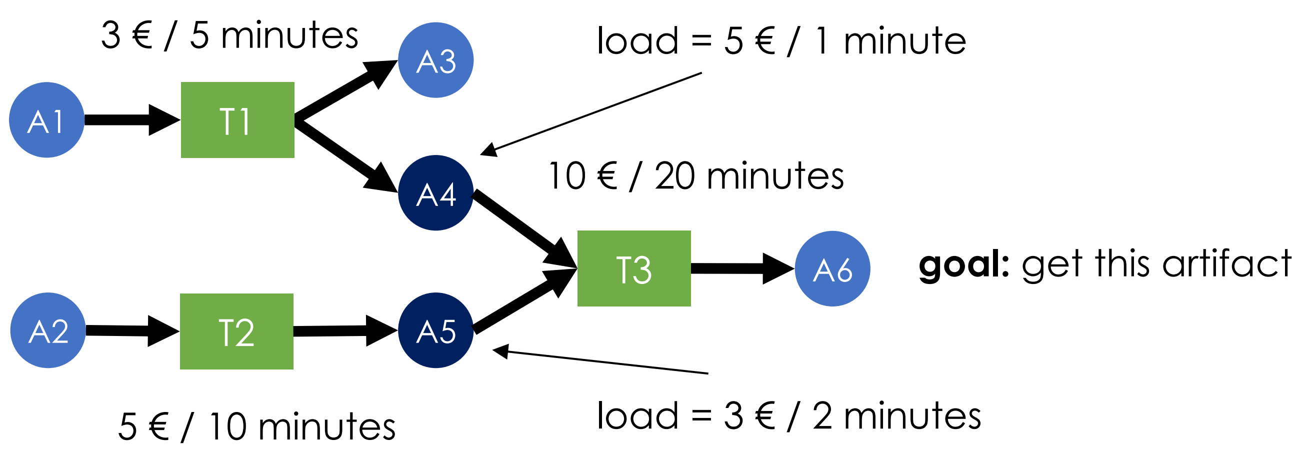
# Reuse – Materialization

- Data Science Workflows are meant to be repeated and reused
  - e.g., in data explorative analysis, ML model selection
- Some of the artifacts may be used across workflows

- Suppose we **materialize**, i.e., store, them for future **reuse**. Can we reduce cost and time?

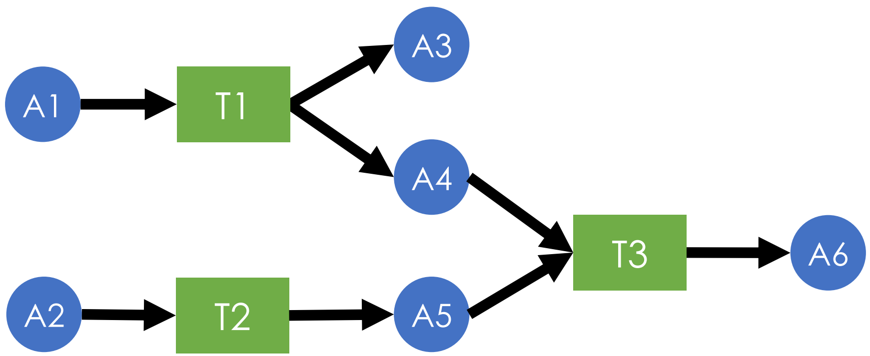
# Reuse Problem

- Given a set of artifacts that are materialized, decide whether to **compute** or **reuse** them when executing a workflow
- What would you decide here?
  - If I cared about cost, I would load A5.
  - If I cared about time, I would load A4 and A5
- Solving the Reuse Problem is computationally hard
  - Equivalent to Max-Flow problem in graphs, about $O(n^3)$



# Materialization Problem

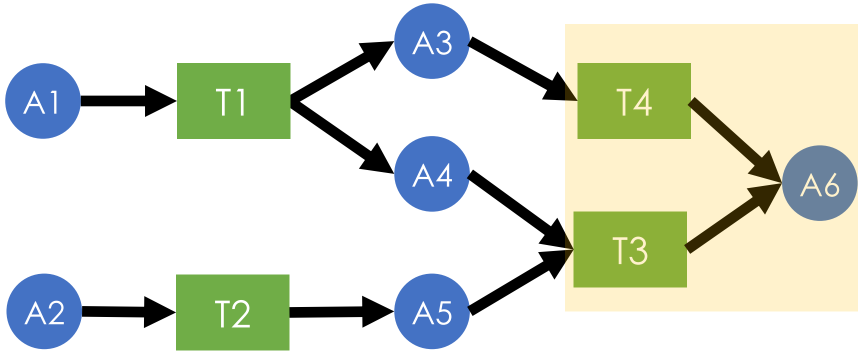
- Given a prediction about future workflows decide if and what artifacts to **materialize**
  - To store or not store A1, A2, …?
- Solving the Materialization Problem is computationally very hard
  - Even assuming complete knowledge of the future
  - NP-hard




**goal:** get this artifact

# Alternatives in Task-Artifact Graphs

- So far in the task-artifact graphs, there is only one way to obtain an artifact
- In the **reuse problem**, there are two ways: **compute** (do the workflow) or **reuse** (load from store)
- More generally, there can be alternate ways to obtain an artifact. Which way to go?
- Should we compute A6 via T4 or via T3?

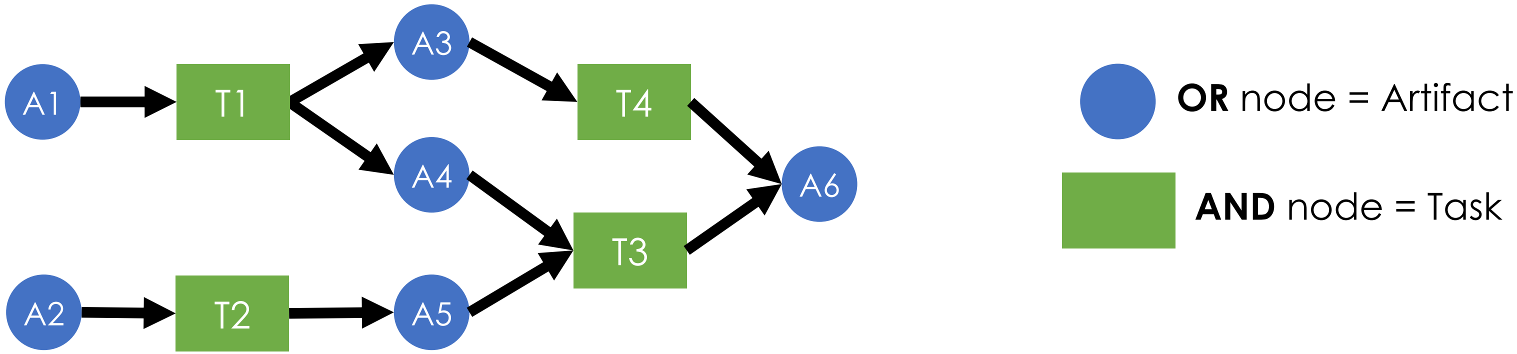



T4 and T3 are equivalent alternatives to get A6

**goal:** get this artifact

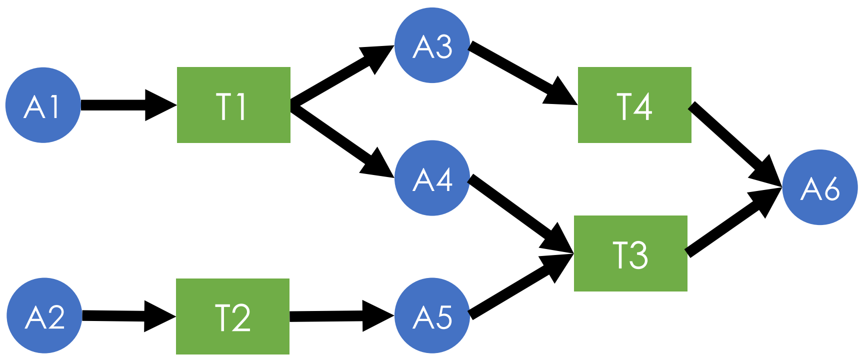
# Task-Artifacts Graphs as AND/OR Graphs

- AND/OR graphs is an abstraction to represent alternatives in workflows
- An OR node means that there are alternative but equivalent ways to reach the node, and only **one in-edge** is necessary
  - All artifacts are OR nodes; most have only one in-edge
- An AND node means that the node requires **all in-edges**
  - All tasks are AND nodes



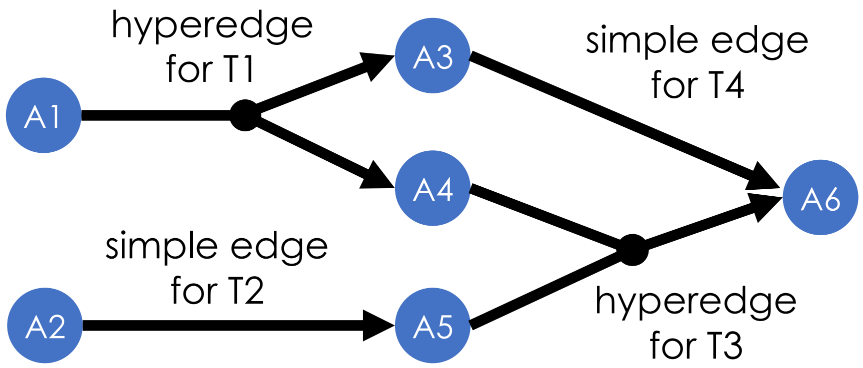
# Artifact Hypergraph

- an AND/OR graph can be represented as a hypergraph
- a hypergraph has **hyperedges**
  - edge is from **one** node to **another** node
  - hyperedge is from a **set** of nodes to another **set** of nodes
- hyperedges represent tasks




AND/OR Graph

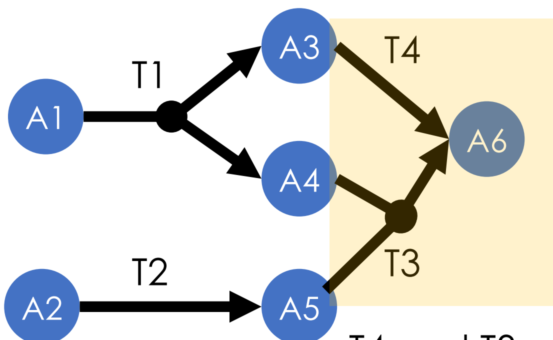
=



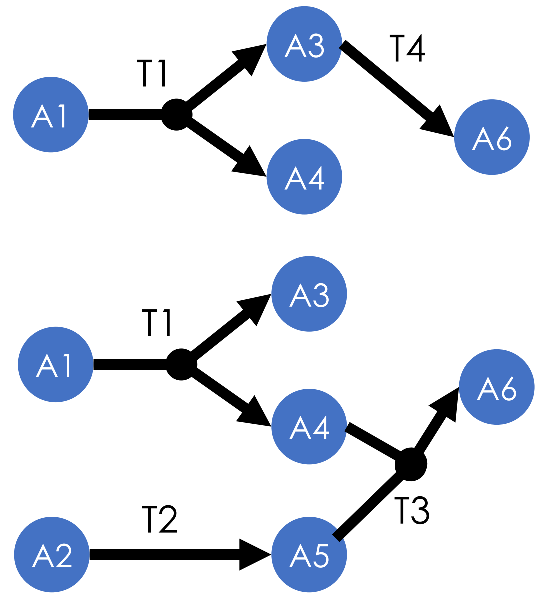

Hypergraph

# A Plan for an Artifact Hypergraph

- hypergraph encodes alternative ways to retrieve artifacts
  - alternatives exists when an artifact has multiple in-edges
- a **plan** is a sub-hypergraph where each node has **a single in-edge**
- **reuse problem** = create a plan from an artifact hypergraph
  - computationally very hard (NP-hard) to find optimal plan




T4 and T3 are alternatives to get A6



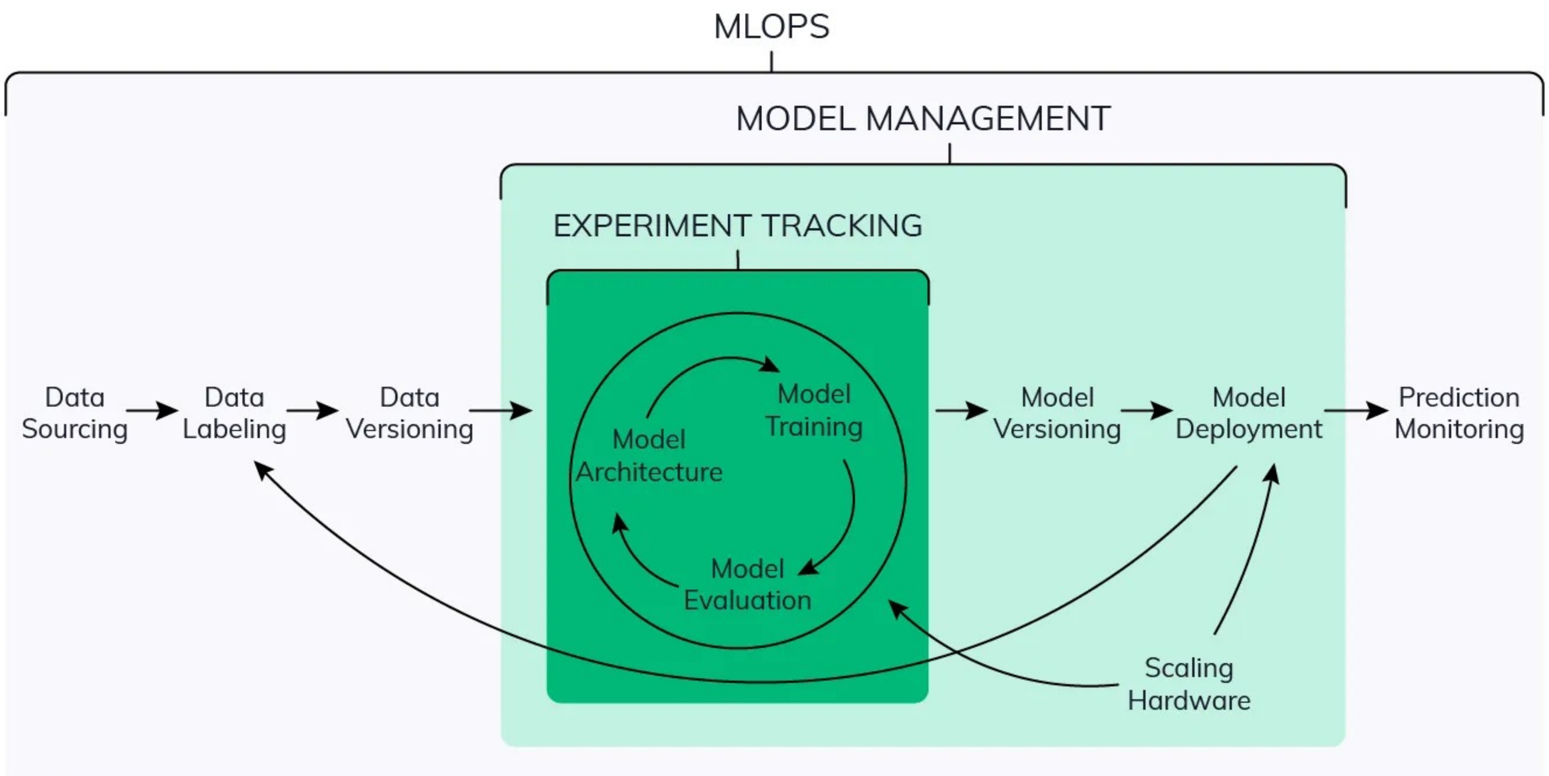

**plan 1**: keep T4

**plan 2**: keep T3

# Data Science Workflows in Practice

# ML Lifecycle

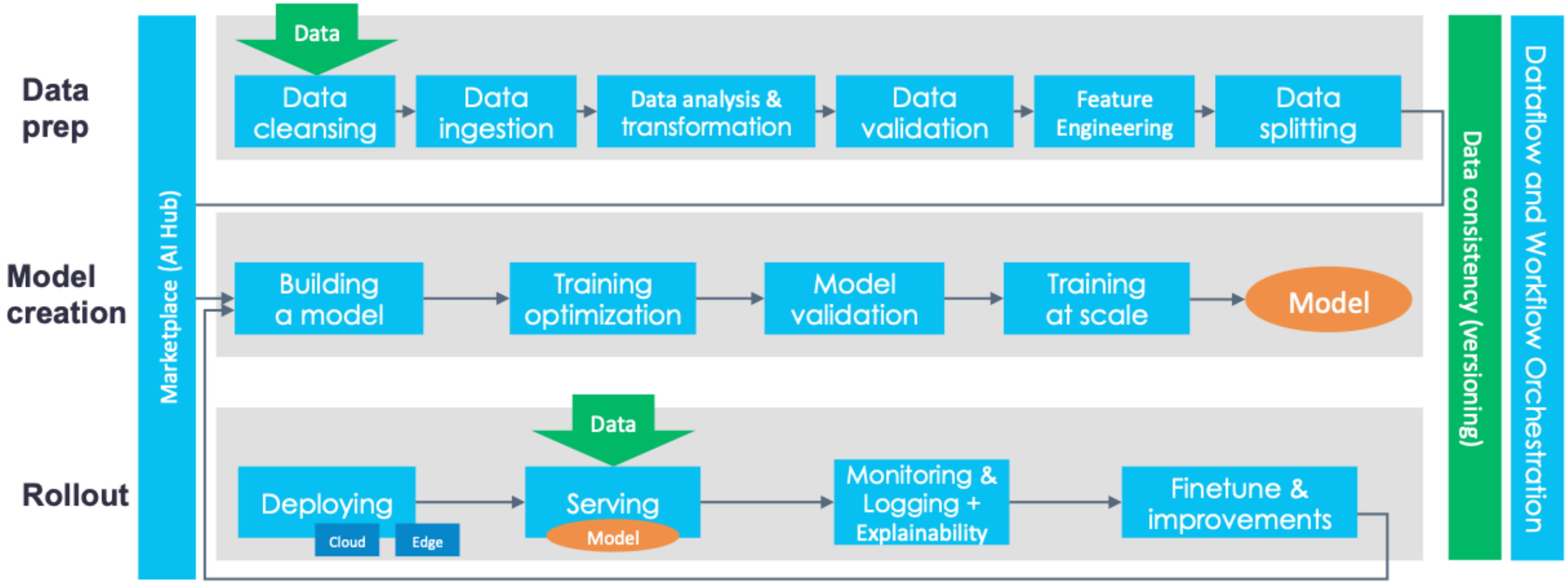
ML Operations (MLOps) = management of the entire ML lifecycle




https://neptune.ai/blog/mlops

# ML Lifecycle

- several workflows within the lifecycle
  - experimentation
  - deployment

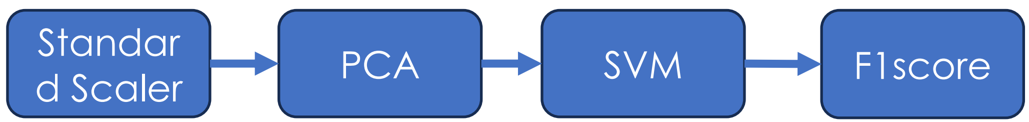



https://www.kdnuggets.com/2020/07/tour-end-to-end-machine-learning-platforms.html

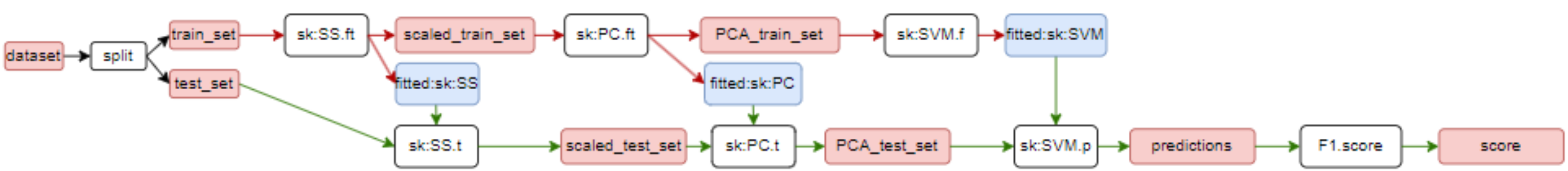
# Abstractions of ML Workflows

Common Machine Learning Strategy


Example Logical Pipeline




Example Physical Pipeline



# Designing a Workflow

at the **physical level** (Implicit): the data scientist provides the code to be executed (e.g., on Jupyter Notebooks)

at the **logical level** (Explicit): the data scientist doesn't have to code

There are two main popular ways of implementing workflows at the logical level:

- Pipeline libraries like (SparkML, Sklearn, ML.Net)
- Data Analytics Platforms (DAPs) that allow the user to graphically construct a Logical pipeline

**Combining both**: Libraries and DAPs allow the user to add custom functionality and implement their own operations

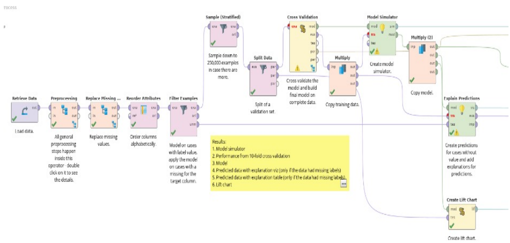
# Data Analytics Platforms

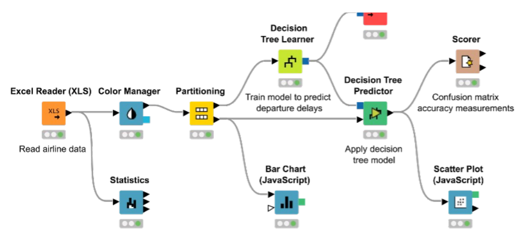
- Offer a graphical UI for describing logical pipelines, by dragging and dropping operators
- The user is not aware of the underneath code

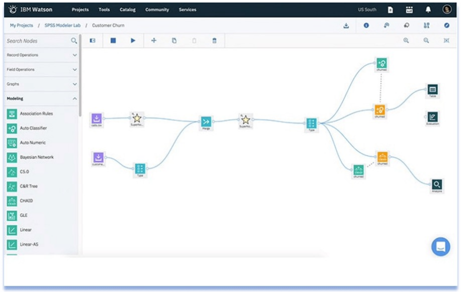
RAPIDMINER

Open for Innovation KNIME

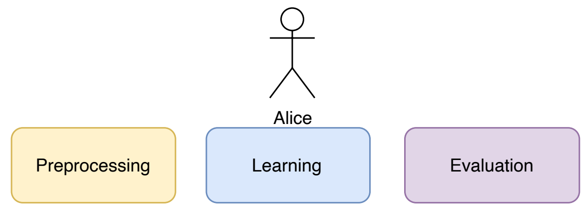
IBM Watson





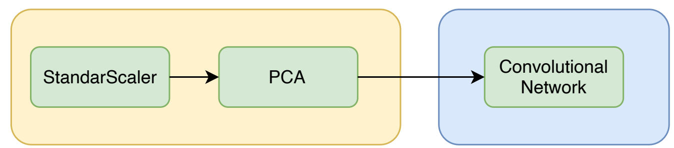



Every Data Scientist can approach the problem with a different strategy




The logic underneath their strategy can be the same or different

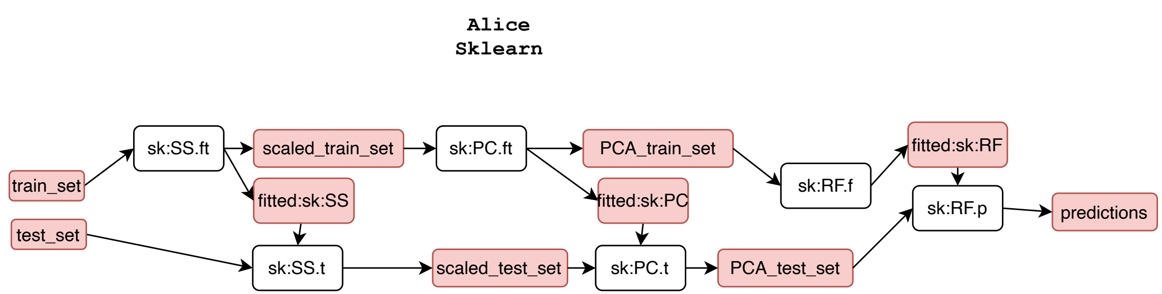
Operator



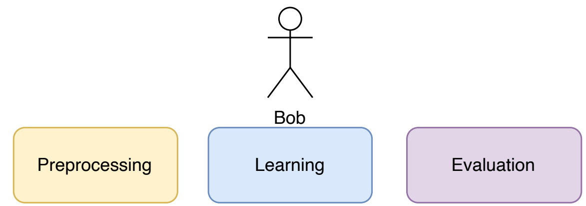

Similar their implementation underneath their logic can be the same or different

Artifacts

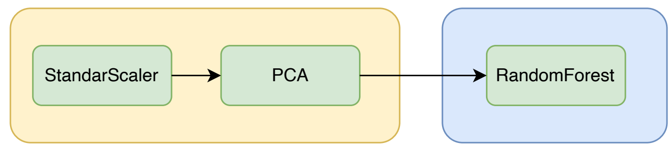
Task



Every Data Scientist can approach the problem with a different strategy

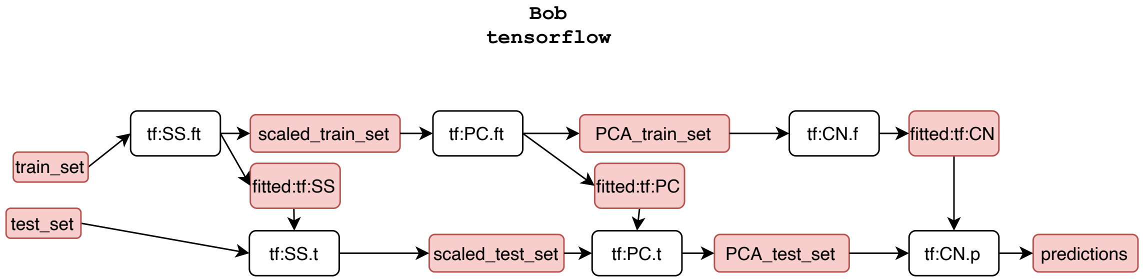



The logic underneath their strategy can be the same or different



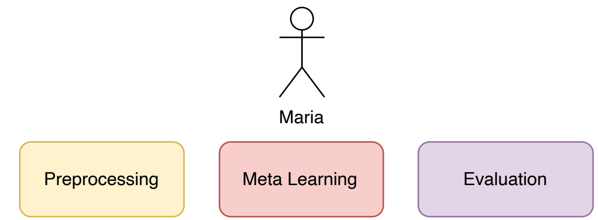

Operator

Similar their implementation underneath their logic can be the same or different



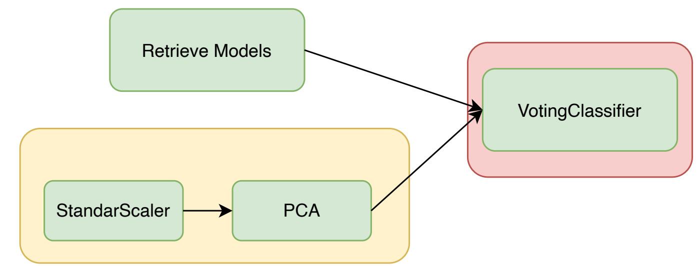

Artifacts Task

Every Data Scientist can approach the problem with a different strategy




The logic underneath their strategy can be the same or different

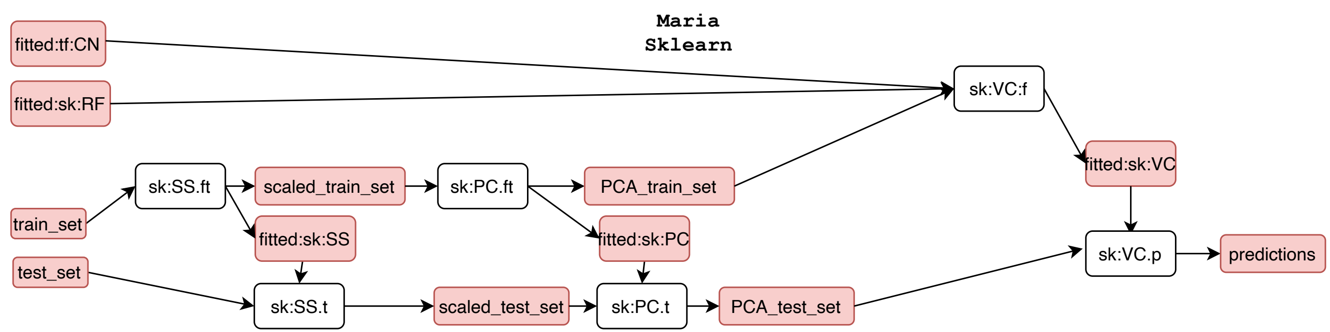
Operator



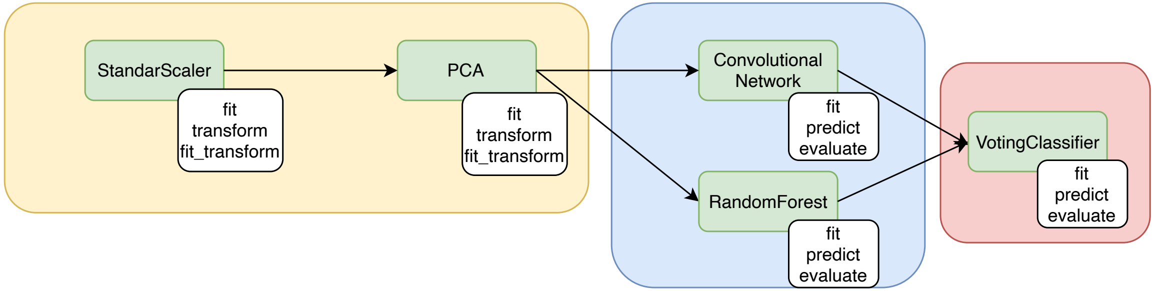

Similar their implementation underneath their logic can be the same or different

Artifacts

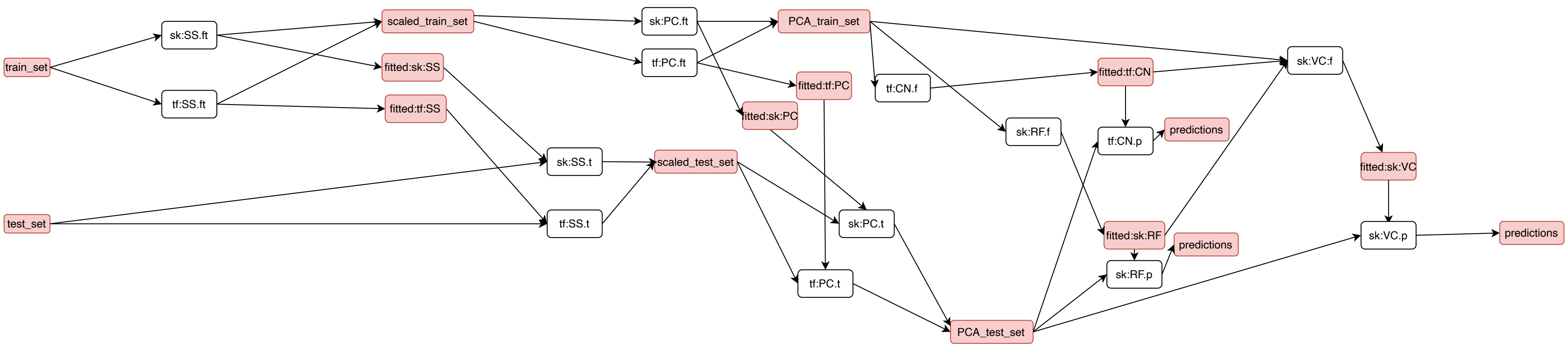
Task



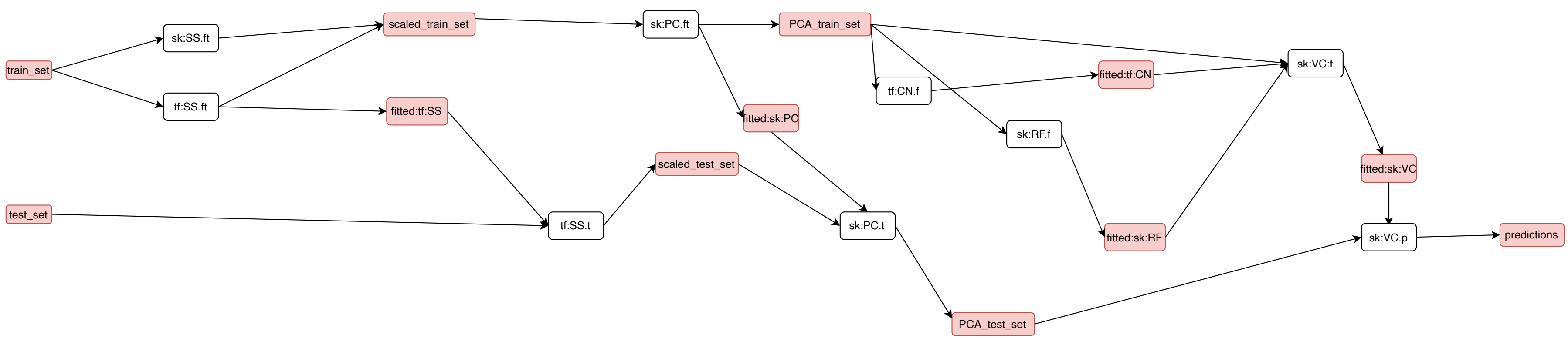
but we an create a unified logical graph



## and a unified task-artifact graph




## and finally a plan



# Optimizing Data Science Workflows

We can leverage:

- **Sharing Computation**: Identifying common subexpression in multiple pipelines so the results are only computed once
- **Materialization**: Storing results so they can be reused in the future
- **Equivalent Verification**: Discovering when the same results can be computed by different pipelines

Creating a Plan: there are trade-offs among when to share, what to materialize and which pipeline to choose.

task-artifact graphs can be huge

creating a plan is not easy