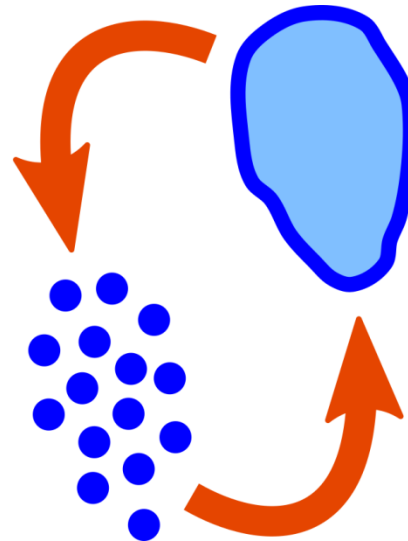


# Hands-on Tutorial on FireWorks

Ivan Kondov

STEINBUCH CENTRE FOR COMPUTING - SCC

**GRK  
2450**

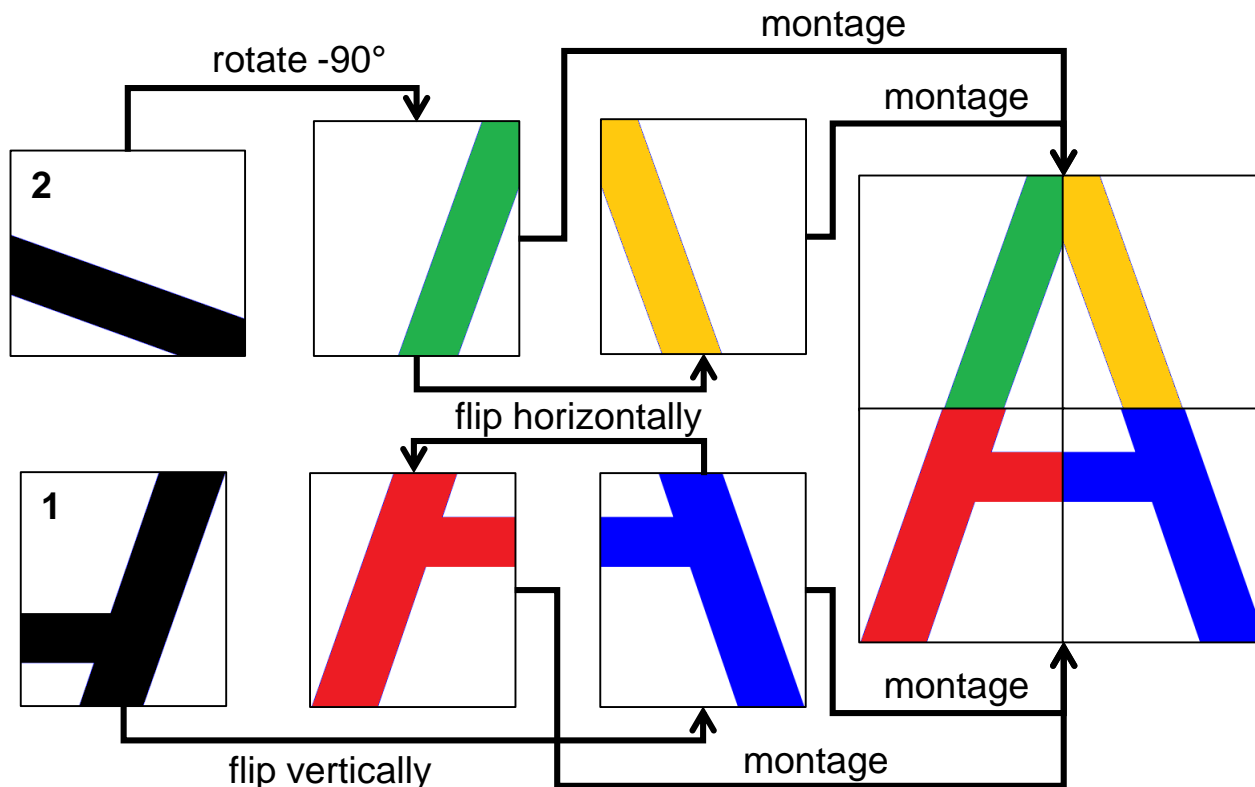


# Agenda

- Introduction to FireWorks
- Tutorial setup
- Basic procedures with short demos: 15 min
- Exercises 1-3: 45 min per exercise
  - Control flow
  - Data flow without files, PyTask
  - Data flow with files, ScriptTask and CommandLineTask
- Advanced usage:
  - Using the batch systems on HPC clusters
  - Detecting lost runs, failures, duplicates
  - Performing queries on LaunchPad
  - Using FilePad
- Totally 20 min for 1-2 breaks

# What is a scientific workflow?

- Coordinated execution of repeatable **actions** accounting for dependencies and concurrency
- **Actions:** computation, pre/post-processing, data management and analysis, visualization
- Other (imprecise) names: protocol, recipe, procedure, job chain, task sequence



Can I complete this twice in the same way with the same result?

Can I perform this thousands of times with different inputs?

Can I trace back **where, when and how** the yellow block has been created?

If I **rerun** a single step what other steps must be rerun too?

From where should I **restart** in case of error?

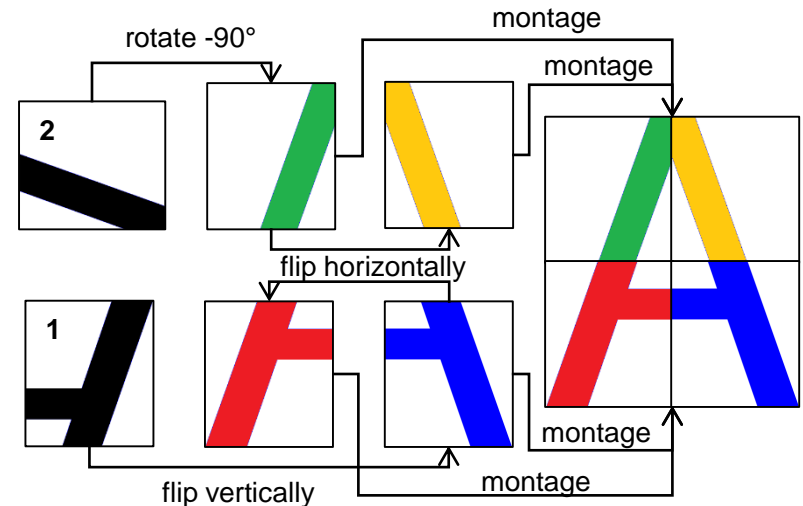
# Tools to realize workflows

Use a script e.g. in Bash

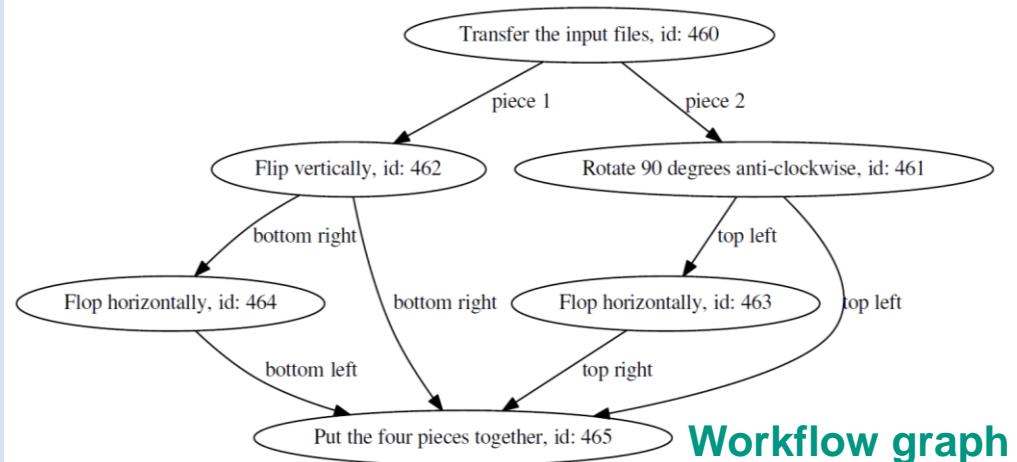
```
#!/bin/bash -eu
convert -flip piece-1.png bottom_right.png
convert -rotate -90 piece-2.png top_left.png
convert -flop top_left.png top_right.png
convert -flop bottom_right.png bottom_left.png
montage -mode concatenate -tile 2x2 \
top_left.png top_right.png bottom_left.png \
bottom_right.png montaged_image.png
```

Better: use a Makefile

```
default: montaged_image.png
bottom_right.png: piece-1.png
    convert -flip $^ $@
top_left.png: piece-2.png
    convert -rotate -90 $^ $@
top_right.png: top_left.png
    convert -flop $^ $@
bottom_left.png: bottom_right.png
    convert -flop $^ $@
montaged_image.png: top_left.png top_right.png \
    bottom_left.png bottom_right.png
    montage -mode concatenate -tile 2x2 $^ $@
```



Even better: use a **workflow management system**



**Workflow graph**

# FireWorks

- Open source, website: <https://materialsproject.github.io/fireworks>
- Developed and used in the Materials Project  
<http://www.materialsproject.org>
- Focus on high-throughput computing
- Some specific features:
  - dynamic workflows
  - duplicates detection
- Two usage modes
  - Python scripts or Jupyter notebooks using the Python API
  - Command line tools (without Python)



A. Jain et al., Concurrency and Computation:  
Practice and Experience 27, 5037 (2015)

# Materials science with FireWorks

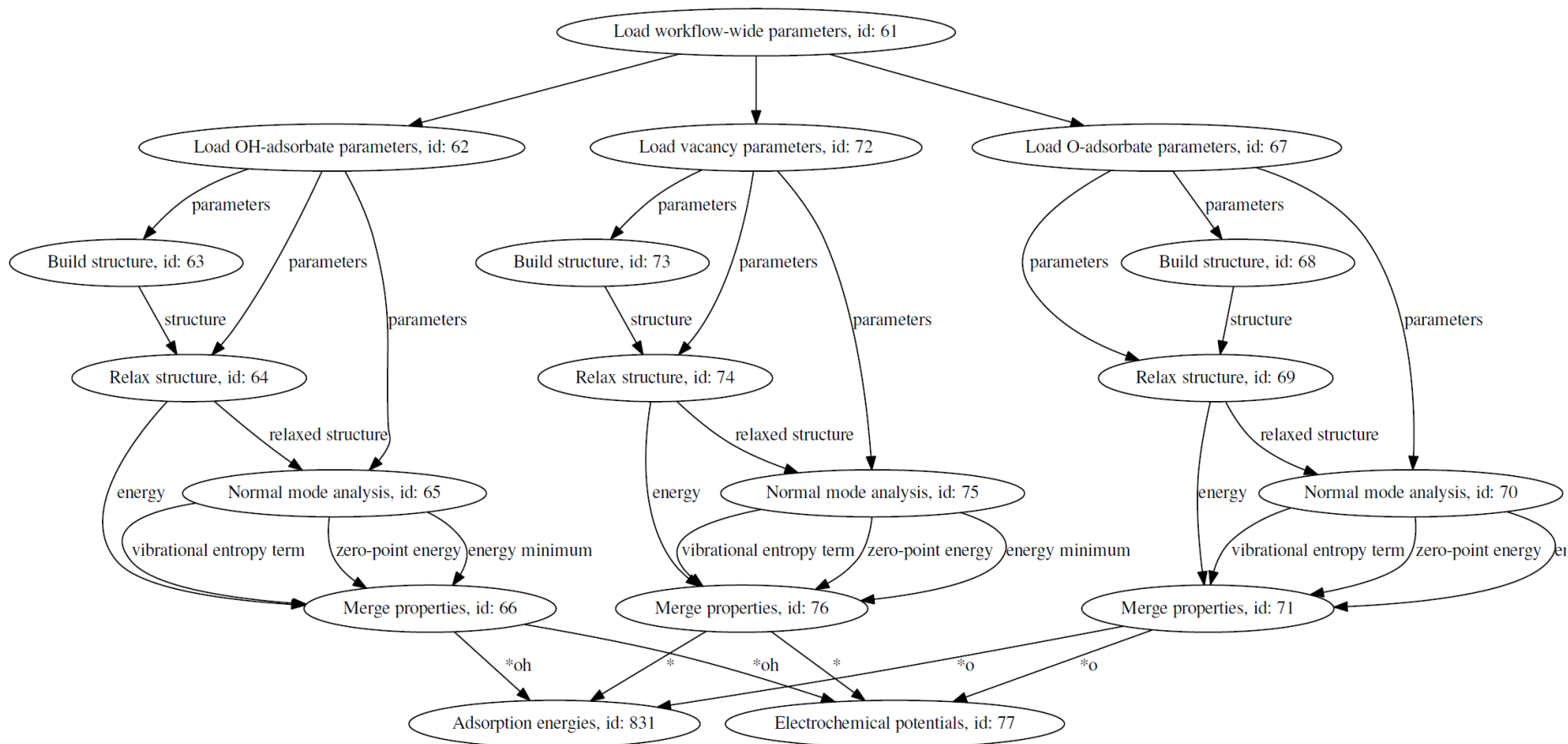
- FireWorks is generic and application agnostic
- ASE: Atomic Simulation Environment
- Atoms object
  - Methods to construct, manipulate, store and visualize atomic structures
- Calculator object
  - Build-in calculators for electronic structure and molecular dynamics codes



A. H. Larsen et al., J. Phys.: Condens. Matter 29, 273002 (2017)  
<https://wiki.fysik.dtu.dk/ase>

# Example using FireWorks with ASE

## High-throughput virtual screening of electrocatalysts



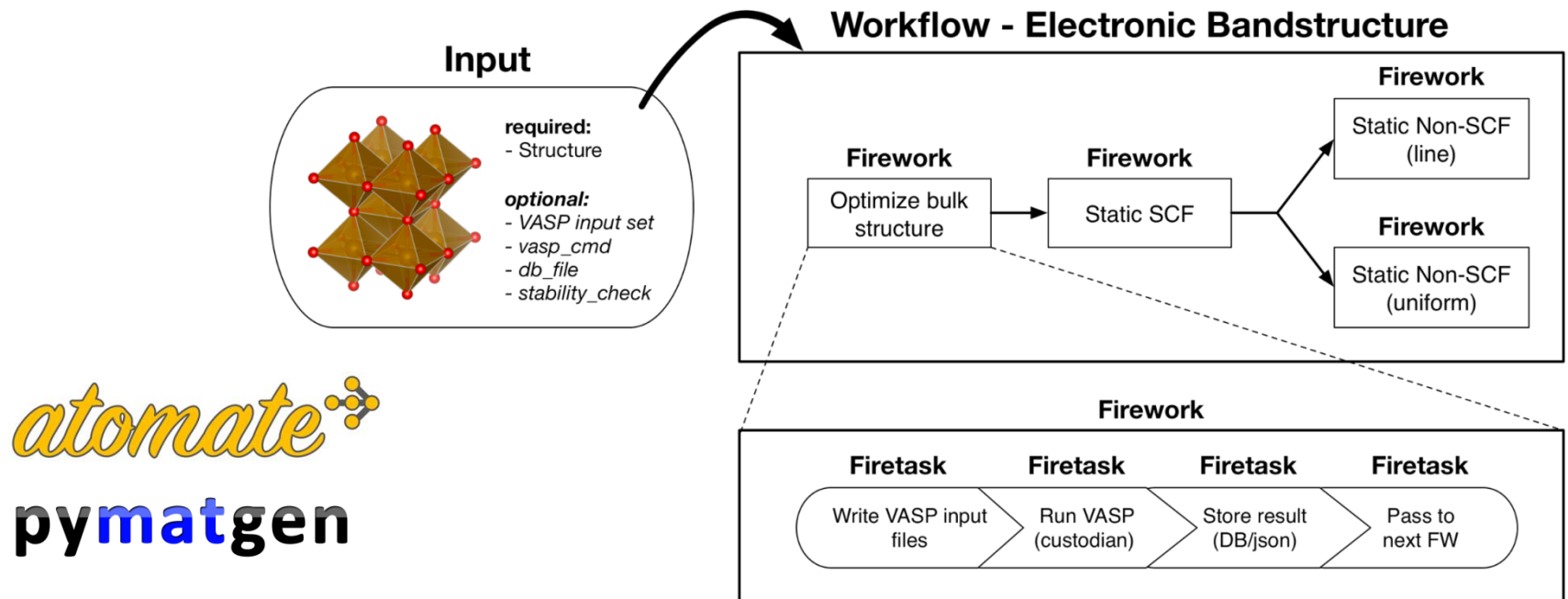
6 DFT runs × 80 cores × 36 hours  
 ≈ 17,000 core hours



# Materials science with FireWorks: Atomate

- Interfaces with Pymatgen
- Library with predefined workflows
- Example: electronic band structure calculation

K. Mathew et al., Comput. Mater. Sci. 139,140–152 (2017)  
<https://atomate.org>



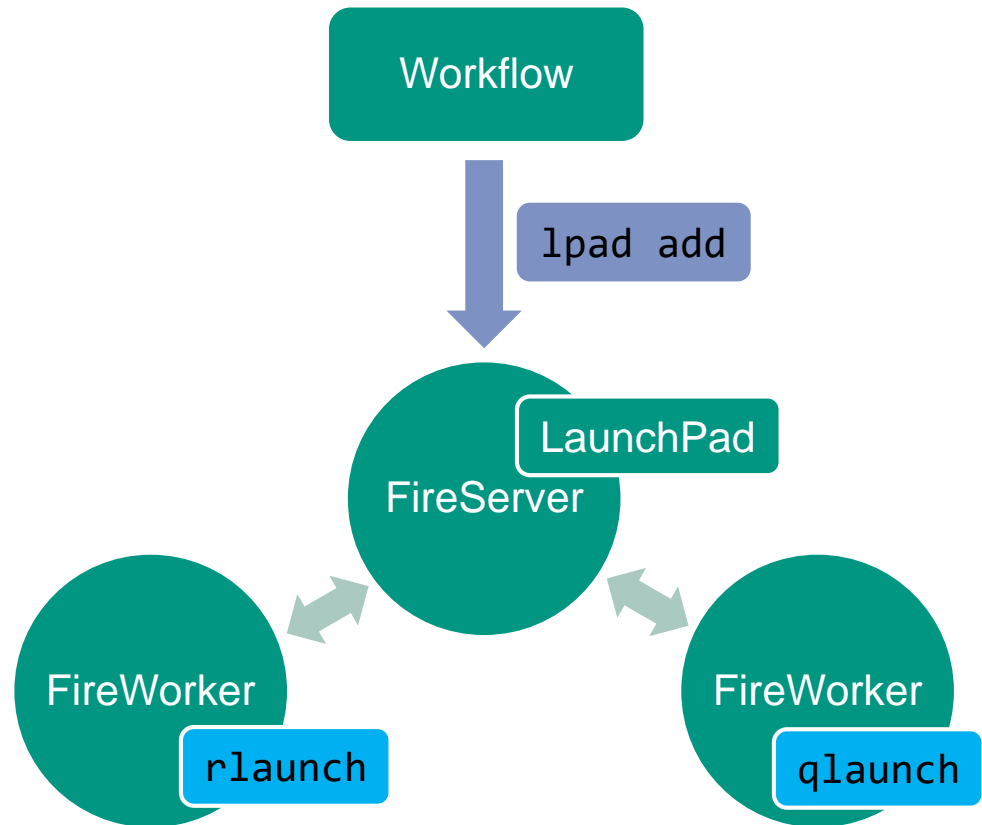
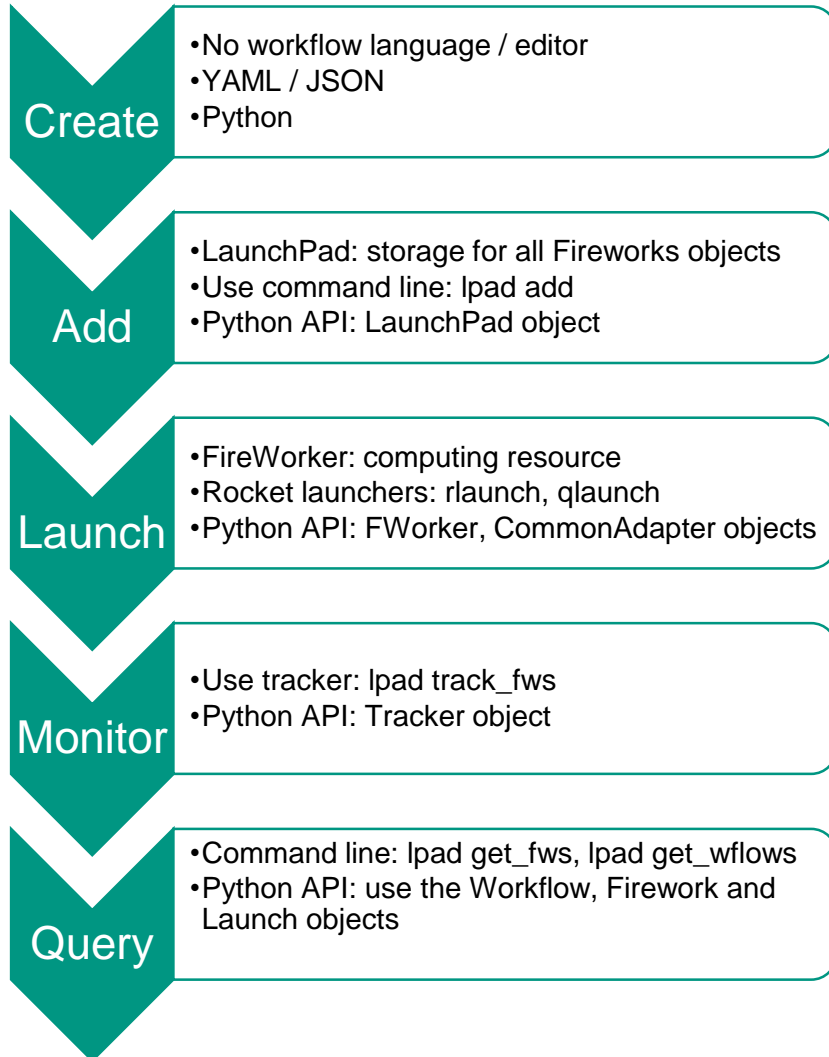
atomate  
pymatgen



# Tutorial setup

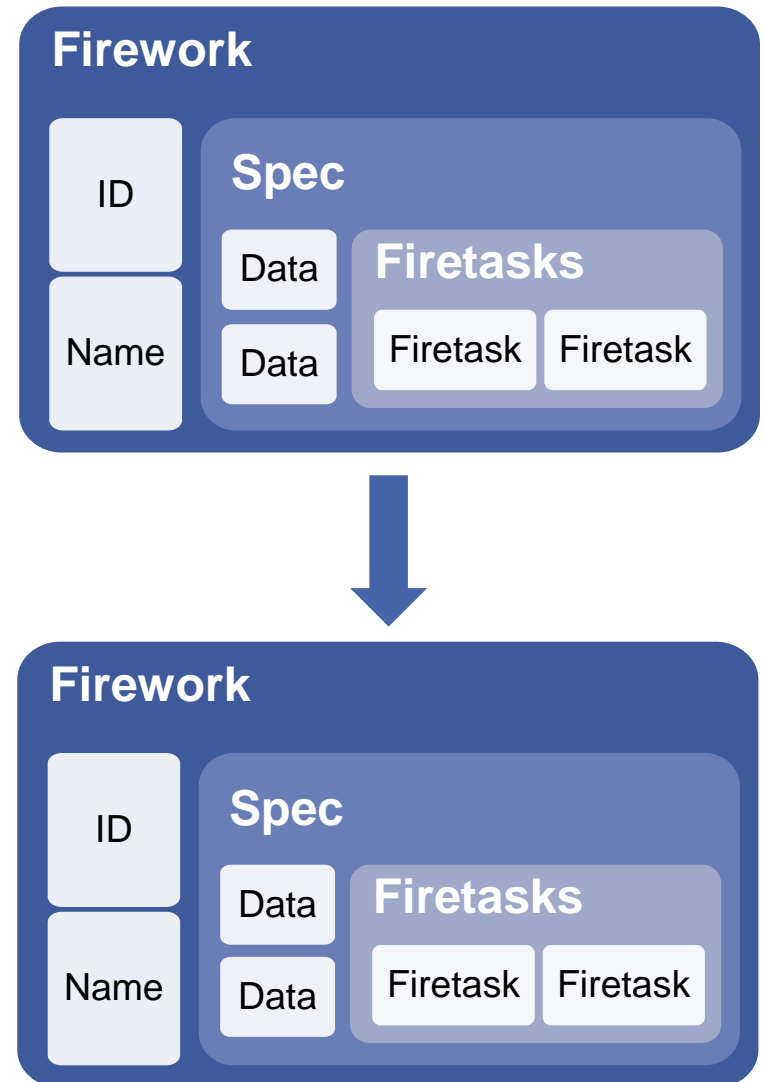
- Tutorial web page:  
<https://git.scc.kit.edu/jk7683/grk2450-fireworks-tutorial>
- docs/setup: Setup instructions, step by step
- docs/basics: Basic commands and procedures
- docs/exercise1-3: Exercises for today

# Basic usage



# Fireworks and workflows

- Firetask: an *atomic* computing job, e.g.:
  - one code / script
  - one Python function
  - one file operation
  
- Firework:
  - Includes bootstrap information: spec
  - Contains one or more Firetasks executed in a sequence
  - Firetasks in one Firework share the same working directory: `_launch_dir`
  - Firetasks may return one or more `FWAction` objects
  
- Workflow
  - A Directed Acyclic Graph (DAG) of Fireworks and links



# Firetasks

## ■ Built-in Firetasks

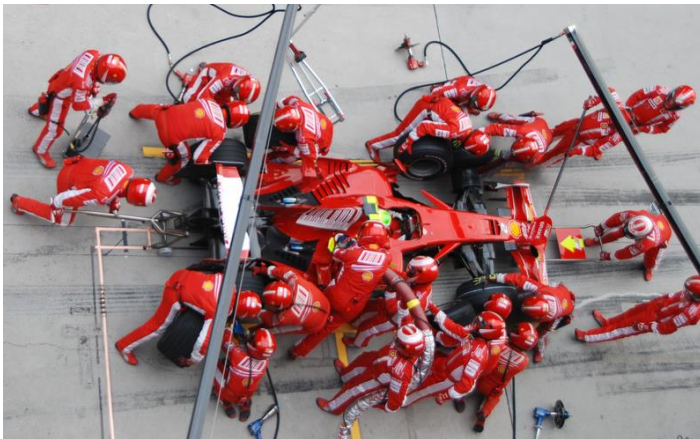
- ScriptTask
- FileWriteTask, FileDeleteTask, FileTransferTask, CompressDirTask, ArchiveDirTask
- TemplateWriterTask (using Jinja templating engine)
- PyTask: if you have a Python function
- CommandLineTask: if you have a console application
- ForeachTask: if you need data parallelism
- JoinDictTask
- JoinListTask

## ■ Writing custom Firetasks

- Requires knowledge of Python

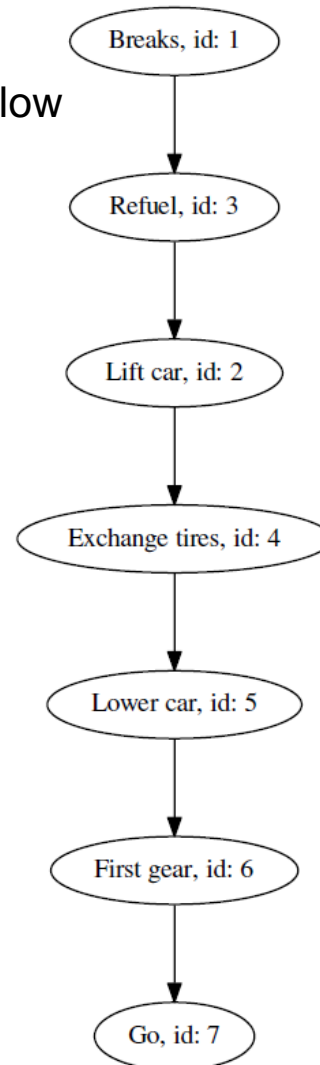
# Exercise 1: Manage control flow

- Use ScriptTask
- Exercise in **docs/exercise3**

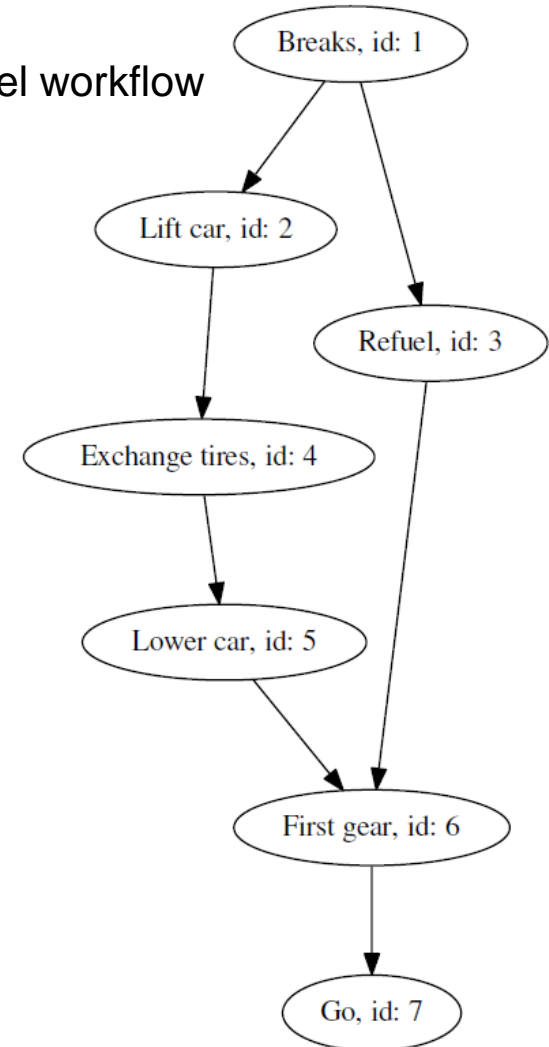


Source: Bert van Dijk, <https://www.flickr.com/photos/zilpho/2964165616>

Sequential workflow



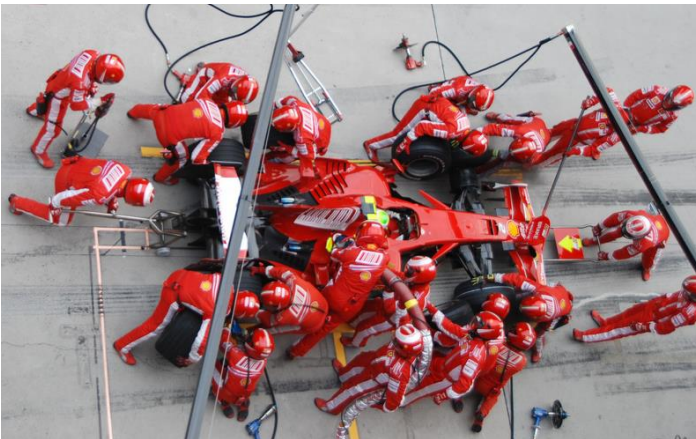
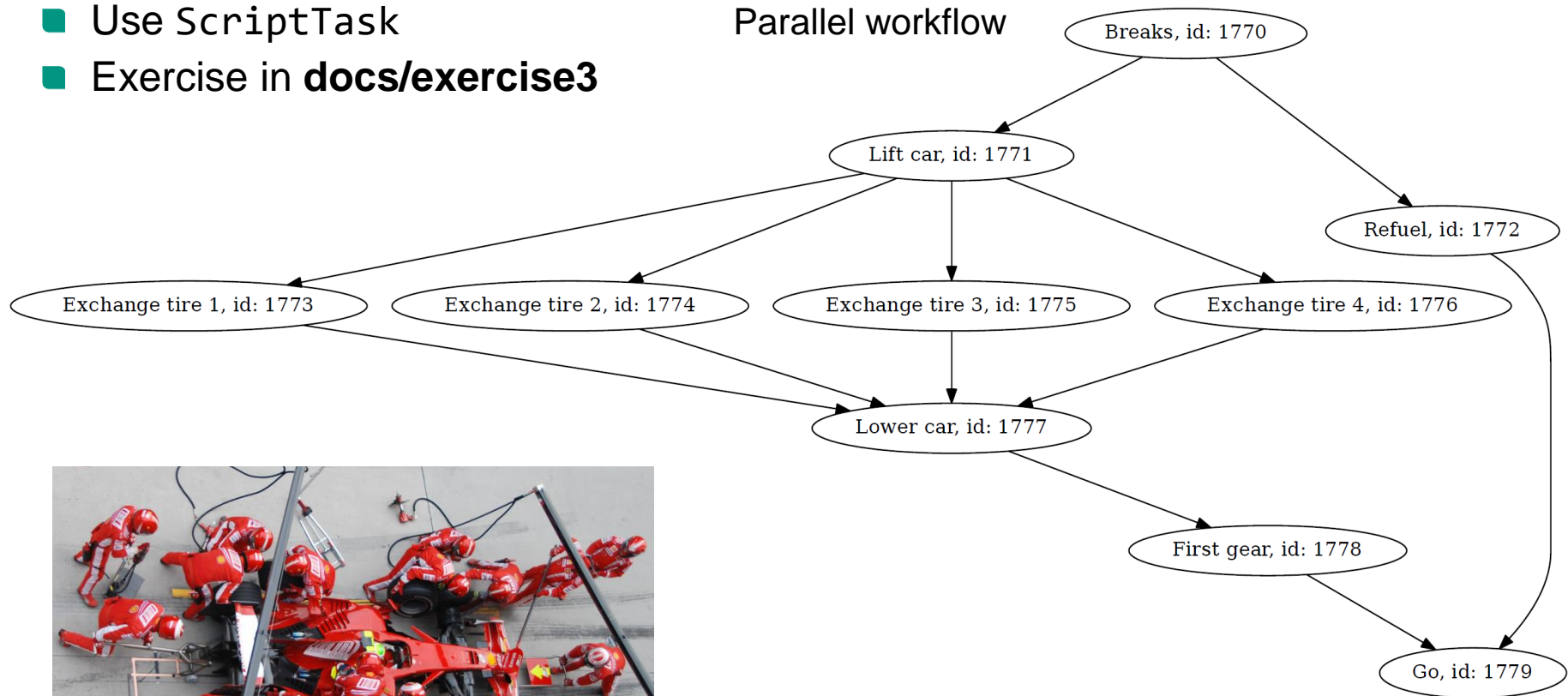
Parallel workflow



# Exercise 1: Manage control flow

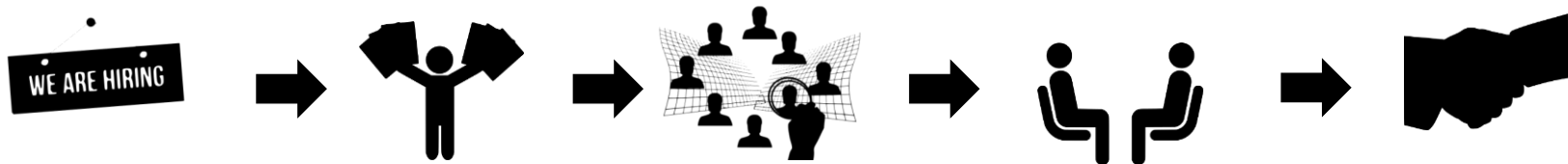
- Use ScriptTask
- Exercise in **docs/exercise3**

Parallel workflow

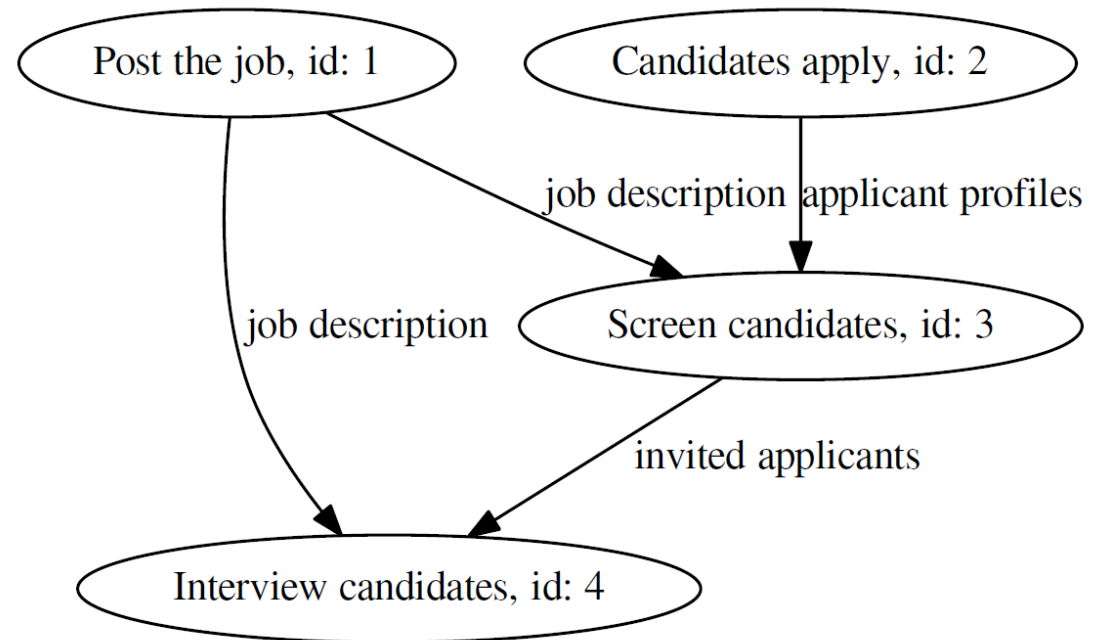


Source: Bert van Dijk, <https://www.flickr.com/photos/zilpho/2964165616>

## Exercise 2: Manage dataflow without files

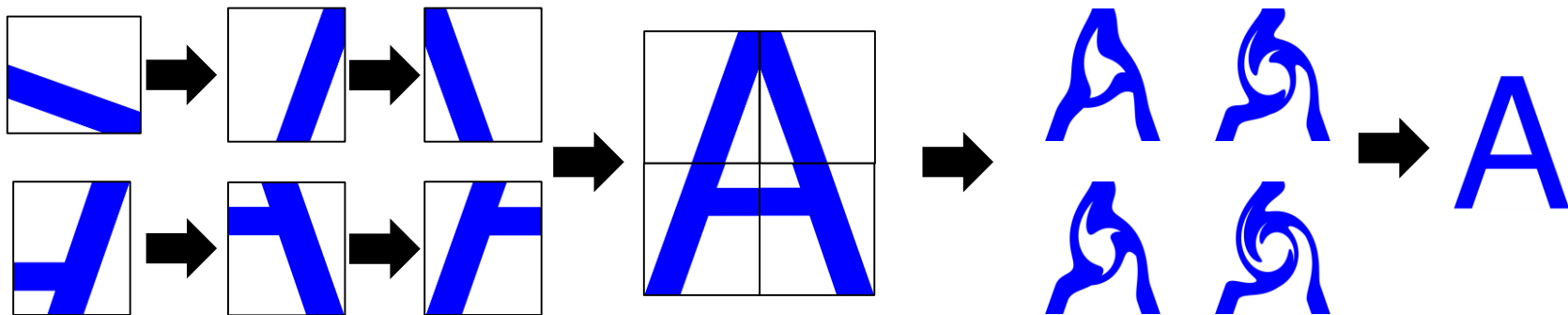
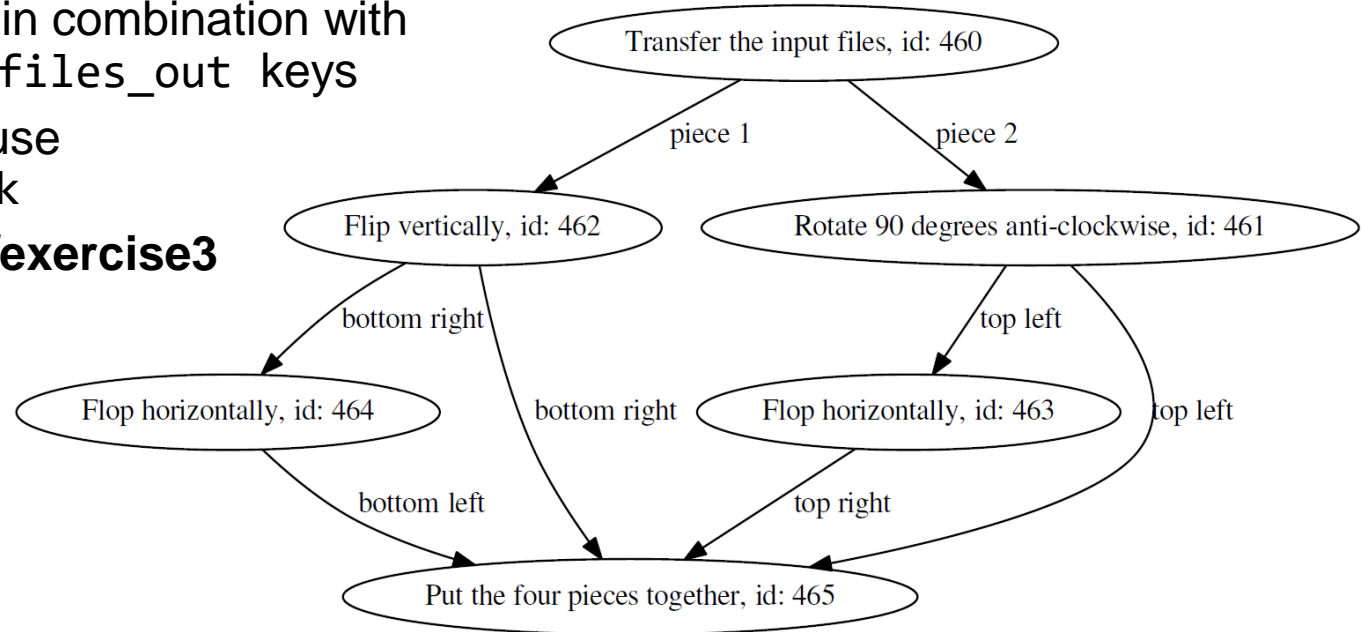


- Exercise in **docs/exercise2**
- Use PyTask



# Exercise 3: Manage dataflow with files

- Use ScriptTask in combination with `_files_in` and `_files_out` keys
- More advanced: use `CommandLineTask`
- Exercise in **docs/exercise3**





# Advanced and productive use

- Tutorial in **docs/advanced**
- Use firework categories and fireworker names
- Submit jobs to the HPC batch system
- Deal with failures and crashes
- Detect lost runs
- Detect duplicates
- Query and analyze data from fireworks and workflows
- Use FilePad to store files

# Legal notice

This work is licensed under an  
Attribution-NonCommercial-NoDerivatives 4.0 International  
Creative Commons License

<http://creativecommons.org/licenses/by-nc-nd/4.0/>

Copyright © 2019 Karlsruhe Institute of Technology (KIT)