Karlsruhe Institute of Technology

Scientific Workflows: Tutorial

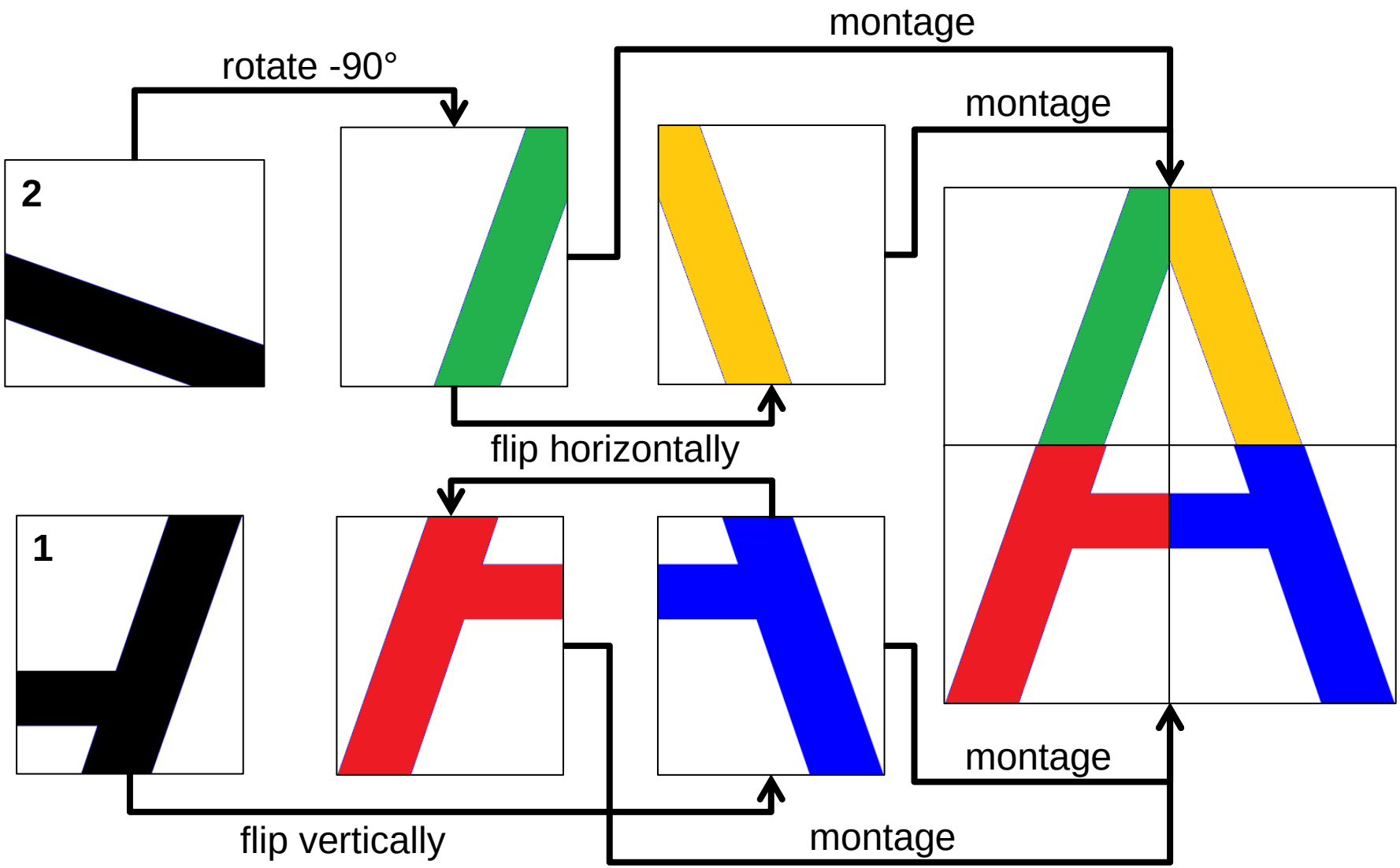
Ivan Kondov, Elnaz Azmi

STEINBUCH CENTRE FOR COMPUTING - SCC

What is a scientific workflow?

- Many different definitions, depending on community
- Coordinated execution of repeatable actions accounting for dependencies and concurrency
- Typical actions
  - Computing, also high performance computing
  - Data management and analysis
  - Pre- and post-processing
  - Visualization
- Other common but imprecise names: “protocol”, “recipe”, “procedure”, “job chain”, “task sequence”, …
- Not to confuse with: business process workflows, pipelines and stream processing

Basic example: image processing



Using a bash script

```
#!/bin/bash -e

# Flip vertically
convert -flip piece-1.png bottom_right.png

# Rotate 90 degrees anti-clockwise
convert -rotate -90 piece-2.png top_left.png

# Flop horizontally top_left.png
convert -flop top_left.png top_right.png

# Flop horizontally bottom_right.png
convert -flop bottom_right.png bottom_left.png

# Put the four pieces together
montage -mode concatenate -tile 2x2 \
top_left.png top_right.png bottom_left.png \
bottom_right.png montaged_image.png
```

- Drawbacks
  - No clear interfaces between steps
  - No reuse of data and code
  - No dependencies
  - No state tracking
  - No concurrency: sequential execution
  - No capability for heterogeneous / distributed resources
- Workarounds
  - Partitioning / Refactoring
  - Checkpointing
  - Chaining with dependencies: sub-blocking, multi-step jobs, multi-jobs or job chains
  - Scheduling

Using a makefile

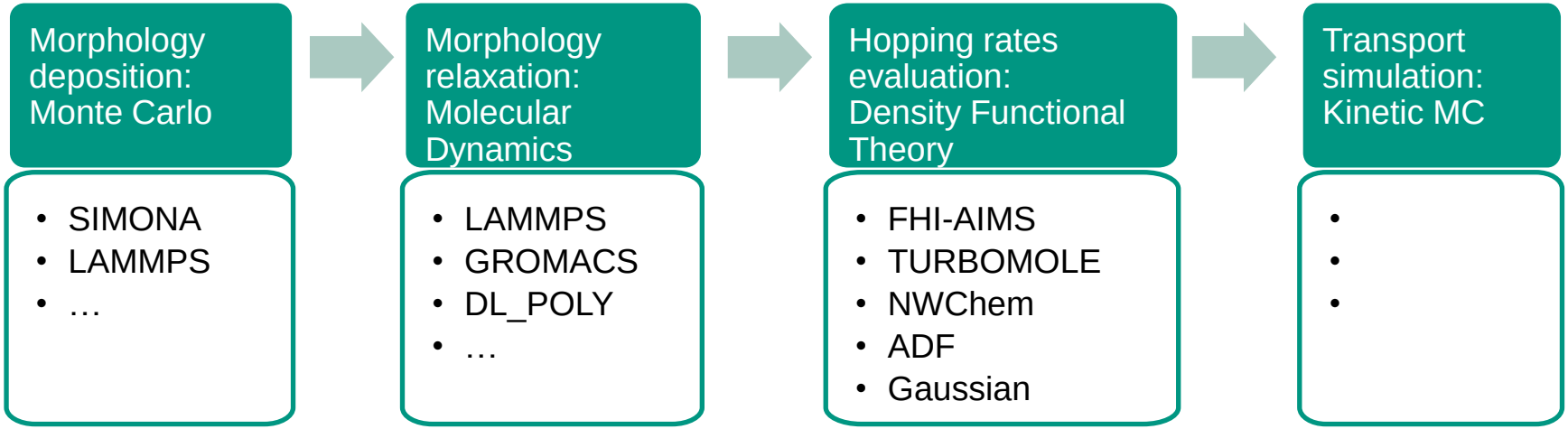
```
default: montaged_image.png

# Flip vertically
bottom_right.png: piece-1.png
    convert -flip $^ $@

# Rotate 90 degrees anti-clockwise
top_left.png: piece-2.png
    convert -rotate -90 $^ $@

# Flop horizontally top_left.png
top_right.png: top_left.png
    convert -flop $^ $@

# Flop horizontally bottom_right.png
bottom_left.png: bottom_right.png
    convert -flop $^ $@

# Put the four pieces together
montaged_image.png: top_left.png top_right.png \
    bottom_left.png bottom_right.png
    montage -mode concatenate -tile 2x2 $^ $@
```

Advantages:

- Steps are defined as ***targets***
- Explicit dependencies defined as ***rules,*** {target, dependencies, commands} triplets
- Data and code reuse
- Implicit state info and checkpoints
  - No state information stored
- Distribute over resources
  - Limited on one host with “-j N”
  - Impossible over nodes and clusters
- Basic idea of the Makeflow system
  http://ccl.cse.nd.edu/software/makeflow

Examples from science and engineering

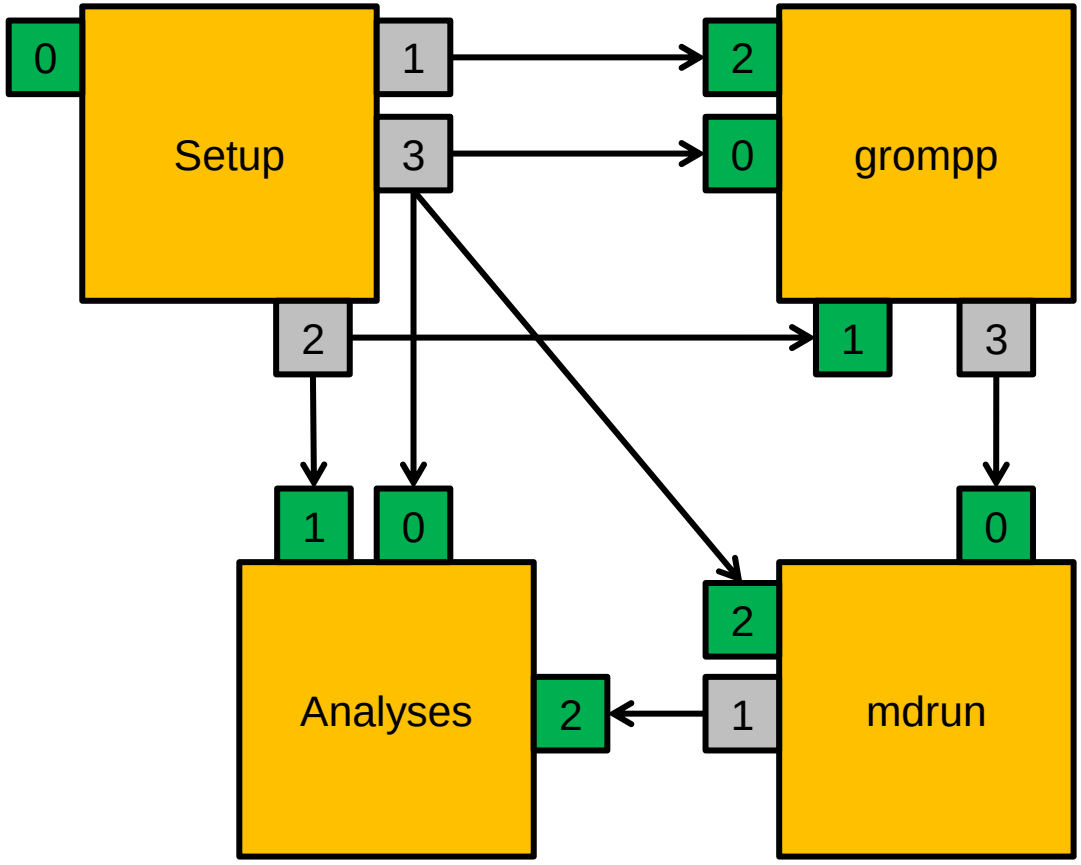
- High throughput computing
  - Life sciences: virtual screening, ligand-protein docking, protein structure prediction, genomics
  - Materials science: virtual materials design
  - High energy physics: event generation
- Multiscale modeling: Loosely coupled multiscale models
  - Climate models: regional – global
  - Materials models: atomistic – coarse-grained – continuum



Workflow graph

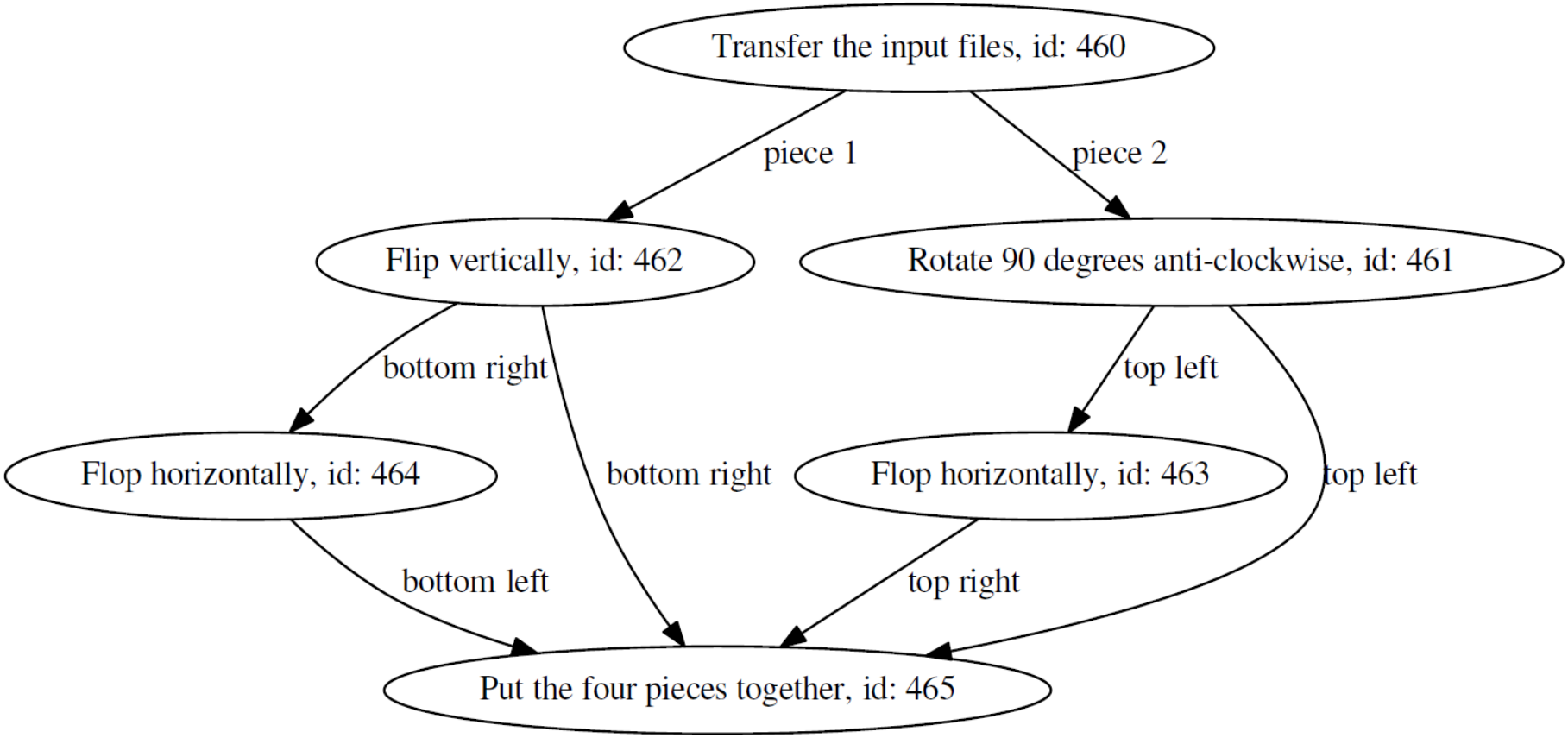
- Directed acyclic graph (DAG)
- Vertices describe the workflow steps - logically separable, reusable components.
- Edges describe the dataflow links.
- The execution sequence is called control flow.
- Control flow satisfies data dependencies but additional rules may also apply.

Workflow for molecular dynamics using Gromacs

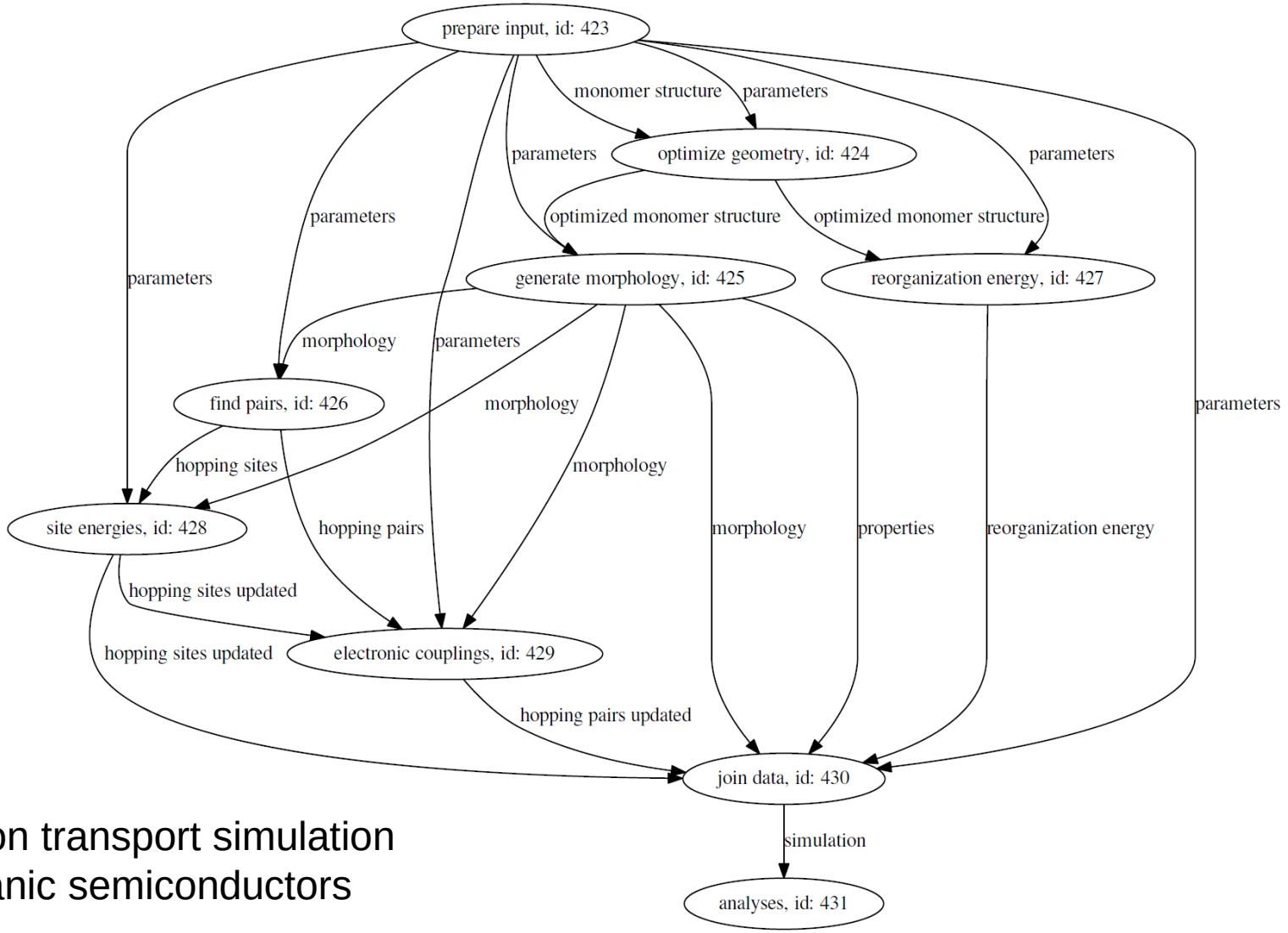


BMC Bioinformatics15, 292 (2014), DOI: 10.1186/1471-2105-15-292
6th International Workshop on Science Gateways (2014), DOI: 10.1109/IWSG.2014.21

The image processing example



Example from materials science



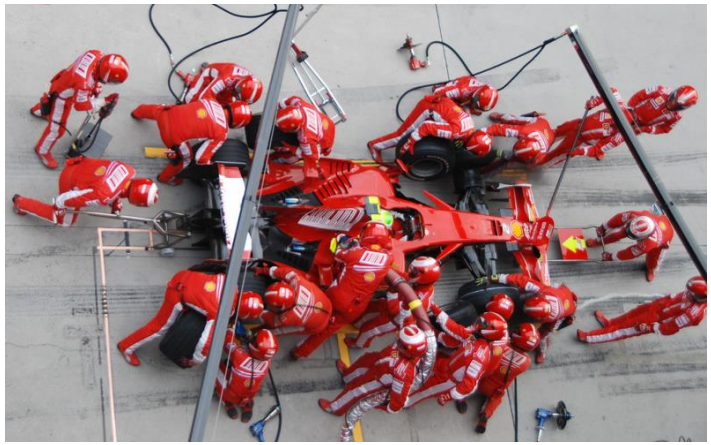
Electron transport simulation
in organic semiconductors

Workflows versus pipelines

Different meanings to different people!

Workflow

- Step 2 begins after Step 1 is ready
- Data is passed discretely
- Directed acyclic graph: no loops



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

Pipeline

- All steps (stages) run simultaneously
- Data is passed continuously
- Directed graph, feedback possible

Source: Ford Europe, https://www.flickr.com/photos/fordeu/5709826282

Workflow systems

- Help composing, managing, running workflows
- One size does not fit all: many existing frameworks
  Incomplete list of over 120 different workflow systems:
  https://github.com/common-workflow-language/common-workflow-language/wiki/Existing-Workflow-systems
- Diversity due to
  - Workflow language
  - Workflow editing and validation tools
  - Workflow engine: scheduling, execution
  - Use of distributed resources, e.g. via web services
  - Software license
  - Application domain

Why using a workflow system?

- Automate complex and high-throughput simulations and analyses
- Scalability: exploit concurrency of independent steps
- Distributed computing
  - Parts of the application run on different computing resources
- Data and code reuse
  - Repeat only failed parts of a simulation or a data analysis
- Data and code provenance
  - Reproducibility of research
  - W7
- Validation
- Error tracking and recovery from failure; reliability
- Rapid prototyping, flexible design

What is a workflow system good for?

- You have a ‘spaghetti code’:
  - Heterogeneous control/data flow pattern
  - One or several repeatedly used elements (functions, scripts, executables)
- Coarse-grained parallelism or ‘embarrassing’ parallelism
  - Independent steps run sufficiently long time
- Irregular iteration
  - Run the same job until convergence
- Heterogeneous resource requirements
  - Some stages are parallel, other sequential
  - Some stages require one computer architecture, others another one
  - Different memory requirements at different stages

Workflow system is not good for:

- Regular and/or recursive numerical algorithms
  - Examples: sorting, fast Fourier transform, linear algebra, …
- Regular and explicit stepping
  - Examples: time evolution, solvers
- Synchronous algorithms
  - Parallel steps wait for a certain event to continue.
- Asynchronous algorithms with
  - many tasks
  - short task runtimes
  - large data and/or frequent data exchange
  - Solution: Use a special workflow language (e.g. the Swift language, http://swift-lang.org) or dataflow-driven framework

Some workflow systems

Workflow System	Web Services	Editor	WF Language	Engine	License	Web site
Taverna	WSDL/SOAP & REST	yes	SCUFL2 (ontology)	yes	Apache	taverna.incubator.apache.org
Kepler	yes	yes	OWL ontology	yes	BSD	kepler-project.org
Airavata	WSDL/SOAP & REST	yes	BPEL, SCUFL, Condor DAG	yes	Apache	airavata.apache.org
UNICORE	WSDL/SOAP & REST	URC	Own XML based	yes	BSD	www.unicore.eu
DAGMan	SOAP	no	Condor DAG	yes	Apache	research.cs.wisc.edu/htcondor
FireWorks	no	no	Python, JSON, YAML based	rlaunch	BSD	materialsproject.github.io/fireworks
Pegasus	yes	Wings	OWL/RDF	yes	Apache	pegasus.isi.edu
gUSE	yes	yes	Own XML based	DCI bridge	Apache	guse.hu

FireWorks

- Define, manage and execute workflows
- Open source – a modified BSD license
- Website: https://materialsproject.github.io/fireworks
- Used in the Materials Project http://www.materialsproject.org
- Based on Python and MongoDB

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

Using FireWorks (basic)

Create

- No workflow language
- No workflow editor available
- Write YAML or JSON
- Use Python scripts

Add

- LaunchPad: storage for all FireWorks objects
- Use command line: **`lpad add`**
- Use Python scripts

Launch

- FireWorker: computing resource
- Rocket launcher: **`rlaunch`**
- Execution via a batch system: **`qlaunch`**
- Use Python scripts

Monitor

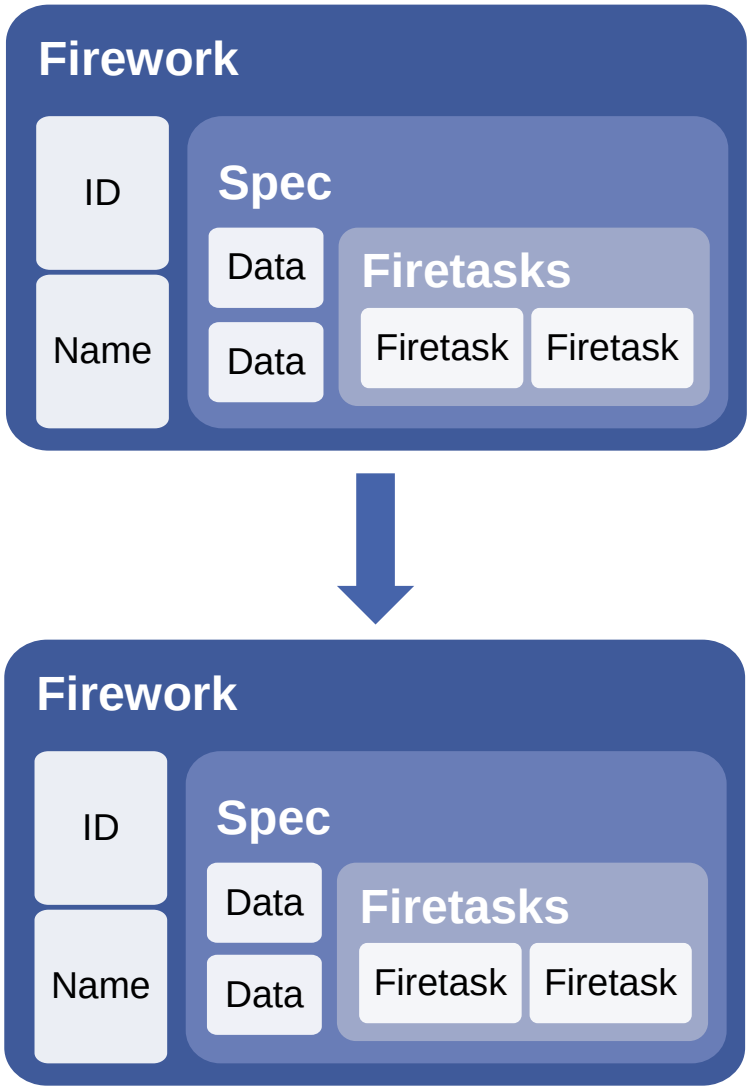
- Command line: **`lpad get_fws`** **`lpad get_wflows`**
- Use tracker: **`lpad track_fws`**
- Use web GUI: **`lpad webgui`**
- Use Python scripts

FireWorks basics

- Firetask: an *atomic* computing job: one code, one script or one Python function

- Firework:
  - Contains one or more Firetasks executed in a sequence
  - Firetasks in one Firework share the same working directory
  - Includes bootstrap information: **spec**

- Workflow
  - A Directed Acyclic Graph (DAG) of Fireworks and links
  - FWAction (optional)



Workflow example in YAML

```
fws:
- fw_id: 1
  name: First act
  spec:
    _tasks:
    - _fw_name: ScriptTask
      script: echo 'To be, or not to be,'
- fw_id: 2
  name: Second act
  spec:
    _tasks:
    - _fw_name: ScriptTask
      script: echo 'that is the question:'
links:
  1:
  - 2
metadata: {}
name: Hamlet workflow
```

- JSON also possible
  - YAML is a superset of JSON
- Attribute *name* is optional
- Submit from the command line:

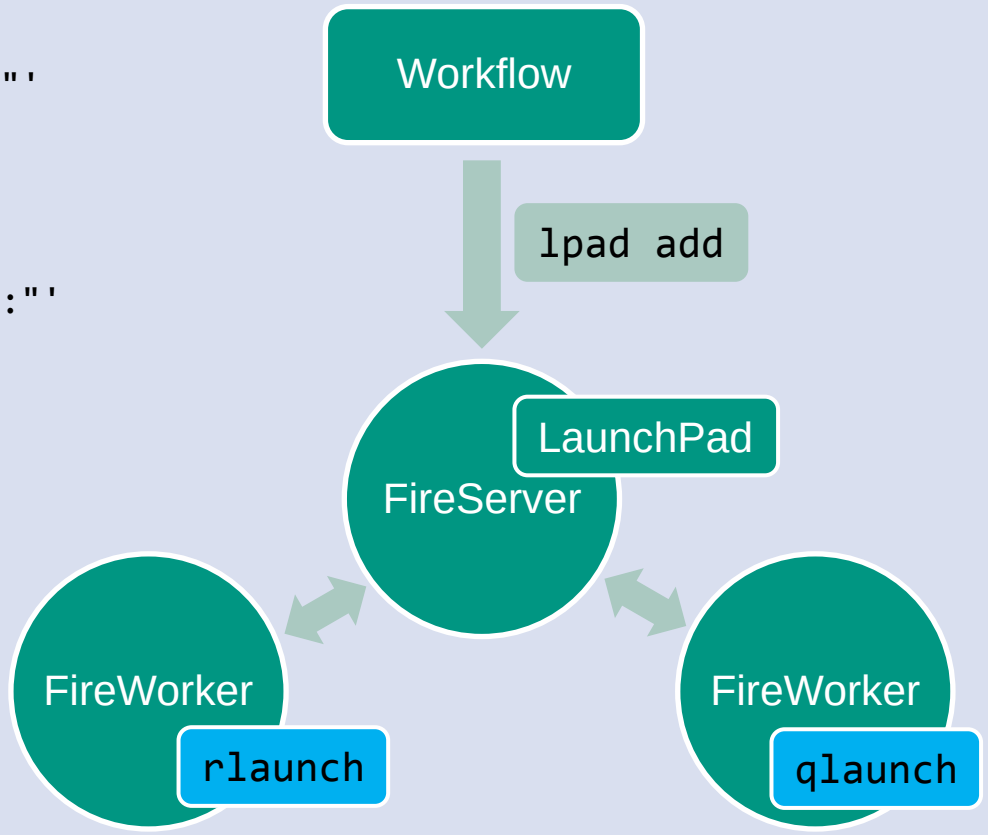
```
lpad add -c hamlet.yaml
```

- Run the workflow:

```
rlaunch rapidfire
```

The same workflow in Python

```
from fireworks import Firework, Workflow, LaunchPad, ScriptTask
from fireworks.core.rocket_launcher import rapidfire

fw1 = Firework(ScriptTask.from_str(
        'echo "To be, or not to be,"'
      )
)
fw2 = Firework(ScriptTask.from_str(
        'echo "that is the question:"'
      )
)
workflow = Workflow(
    [fw1, fw2],
    links_dict={fw1: fw2}
)

launchpad = LaunchPad()
launchpad.add_wf(workflow)
rapidfire(launchpad)
```



Firetasks

Built-in Firetasks

- ScriptTask
- FileWriteTask, FileDeleteTask, FileTransferTask, CompressDirTask, ArchiveDirTask
- TemplateWriterTask
- PyTask

Custom Firetasks

- PythonFunctionTask
- CommandLineTask
- ForeachTask
- JoinDictTask
- JoinListTask
- … write your own Firetasks → Exercise 5

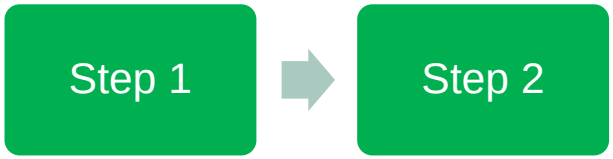
FWAction object

Can be returned by

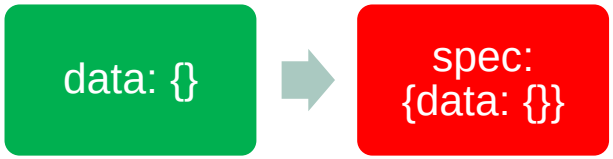
- Firetasks
- Python functions called by **PyTask**

Purpose:

- Pass (meta)data from one firework to another
- Dynamically change workflow



Original workflow

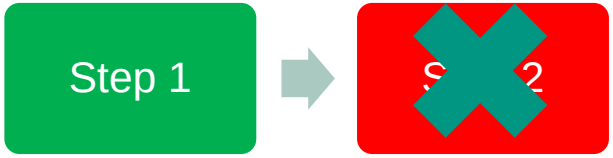


Exercises 2, 3, 5

update_spec, mod_spec

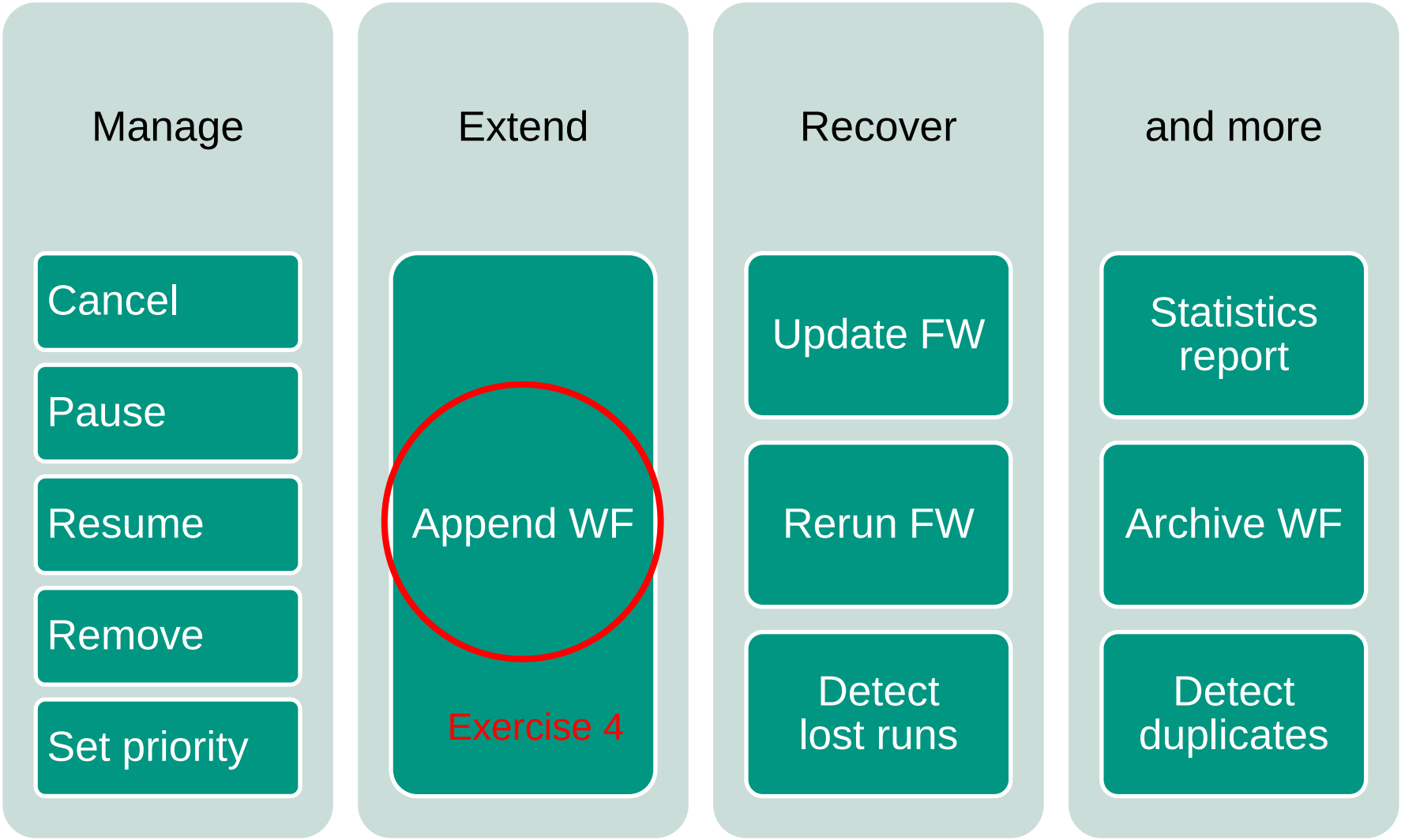
detours

Exercise 5



defuse_children

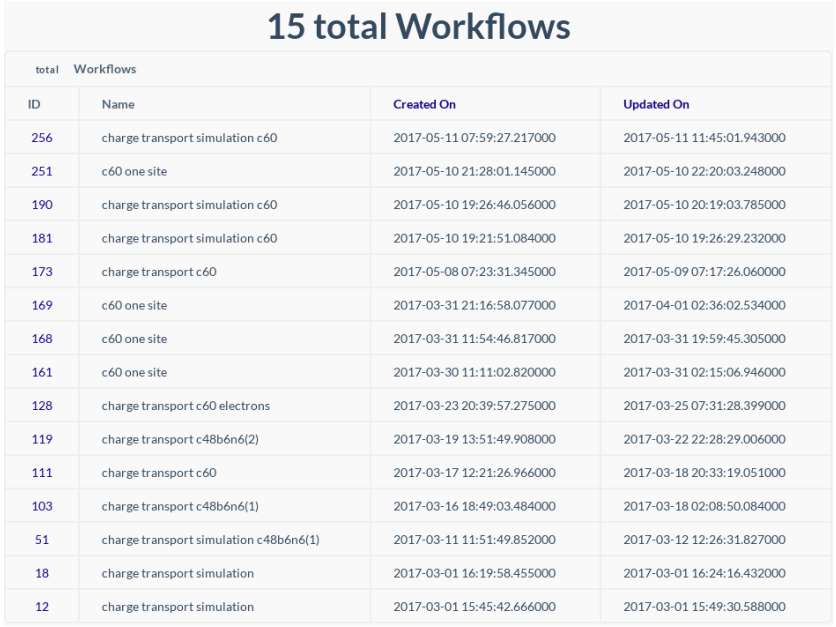
Using FireWorks (advanced)

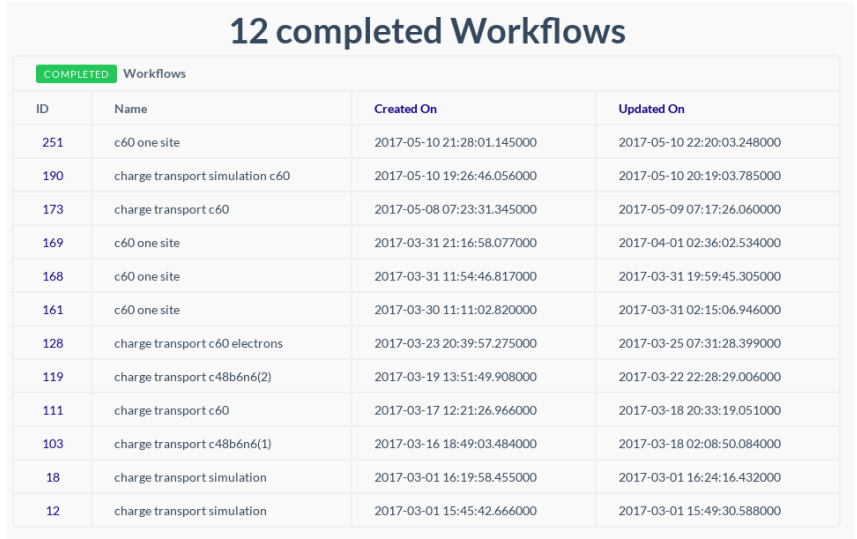


Using FireWorks productively

- Use batch systems: **`qlaunch`**
- Deal with failures and crashes
  - FIZZLED: what to do next?
    - In-place fix (**`lpad update_fws`**) and rerun (**`lpad rerun_fws`**) the Firework – preservation of all COMPLETED guaranteed
    - Error fix and resubmit the whole workflow – better reproducibility
  - RUNNING forever
    - Use the command **`lpad detect_lostruns`**
- Manage duplicates
  - Running workflows with reusing the data from identical Fireworks
  - After fixing an error a new workflow will repeat only the changed Fireworks
- Monitoring (**`lpad webgui`**) and reports (**`lpad report`**)
- Configure security
- Tune performance

Web GUI

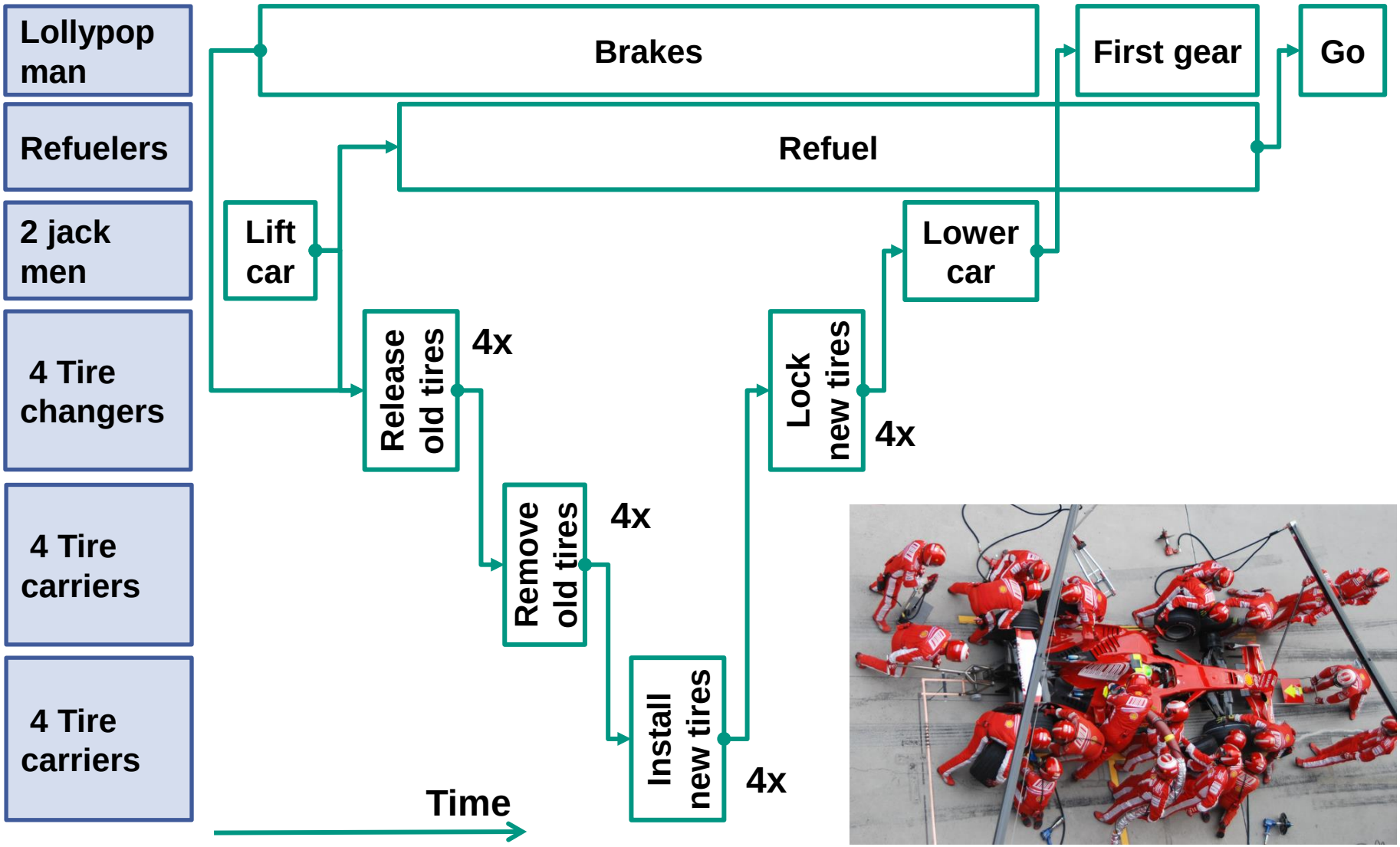




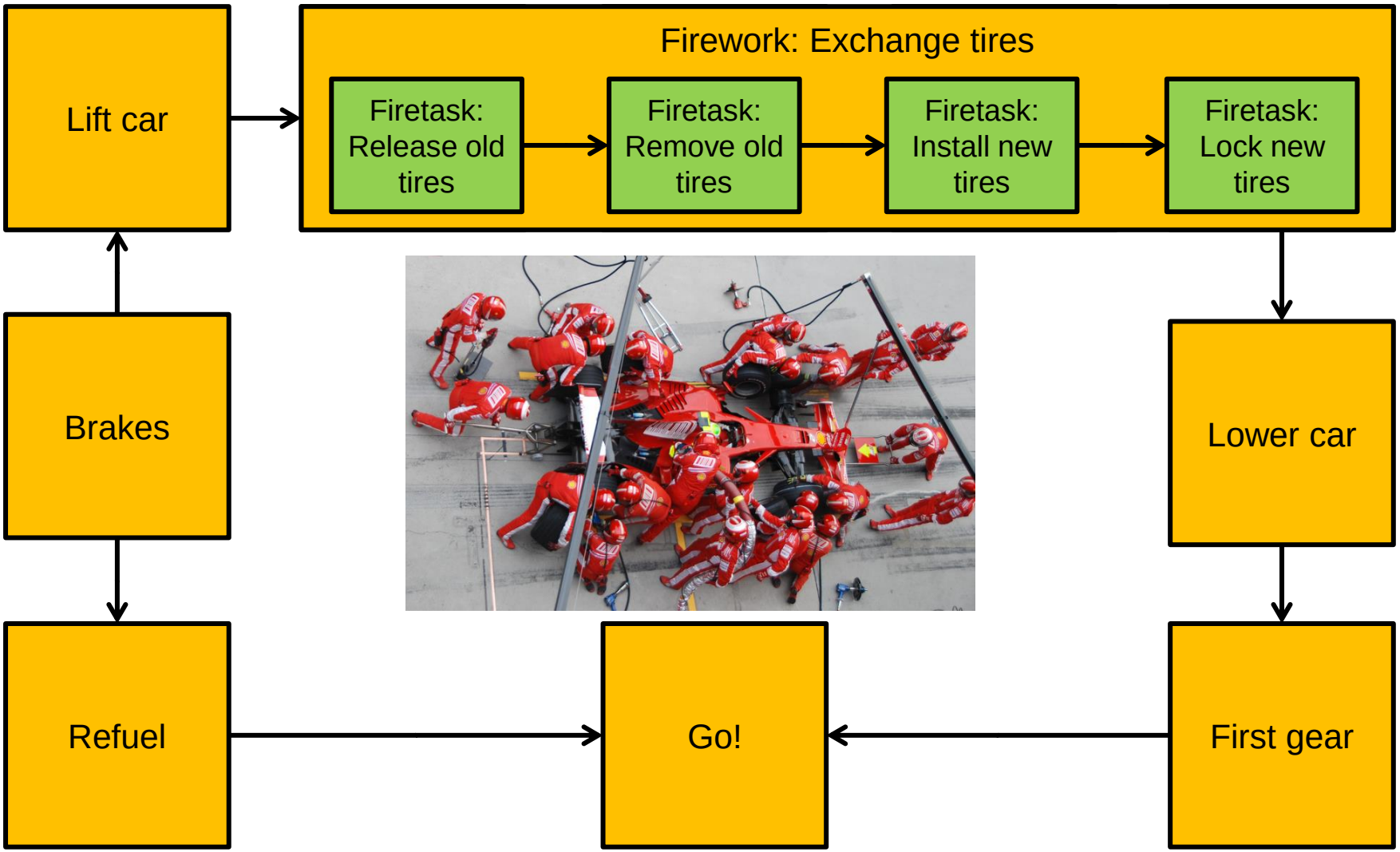
- Use with:

```
lpad webgui
```

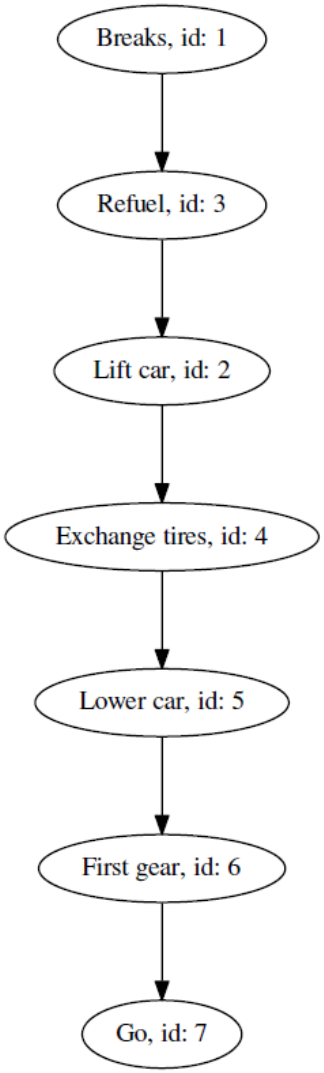
Exercise 1: Managing control flow

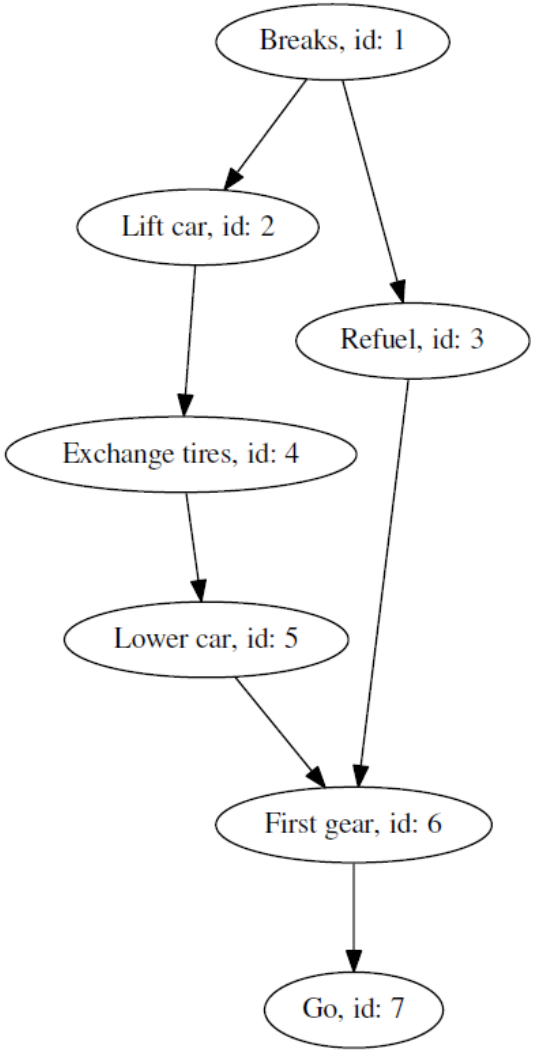


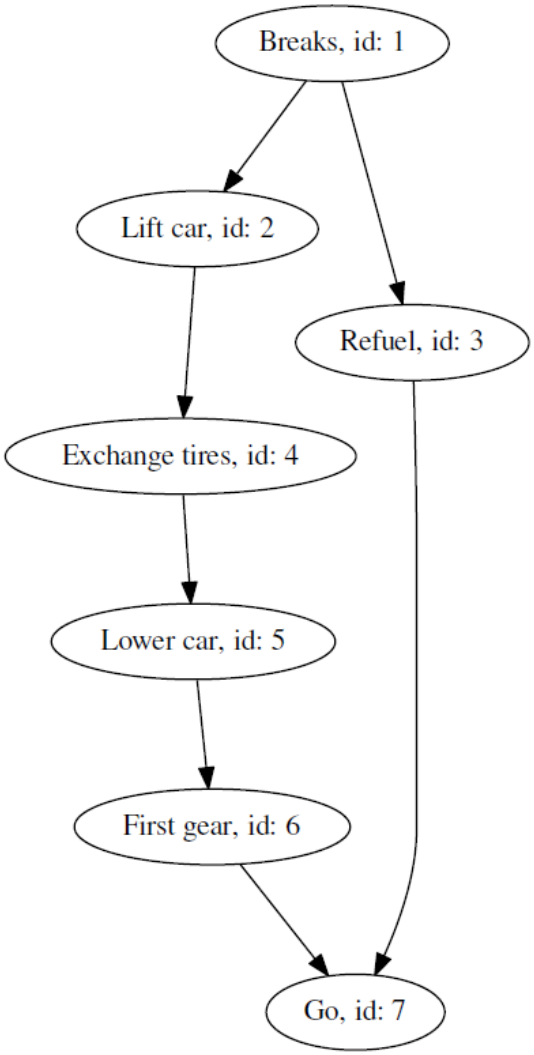
“Sequential” workflow



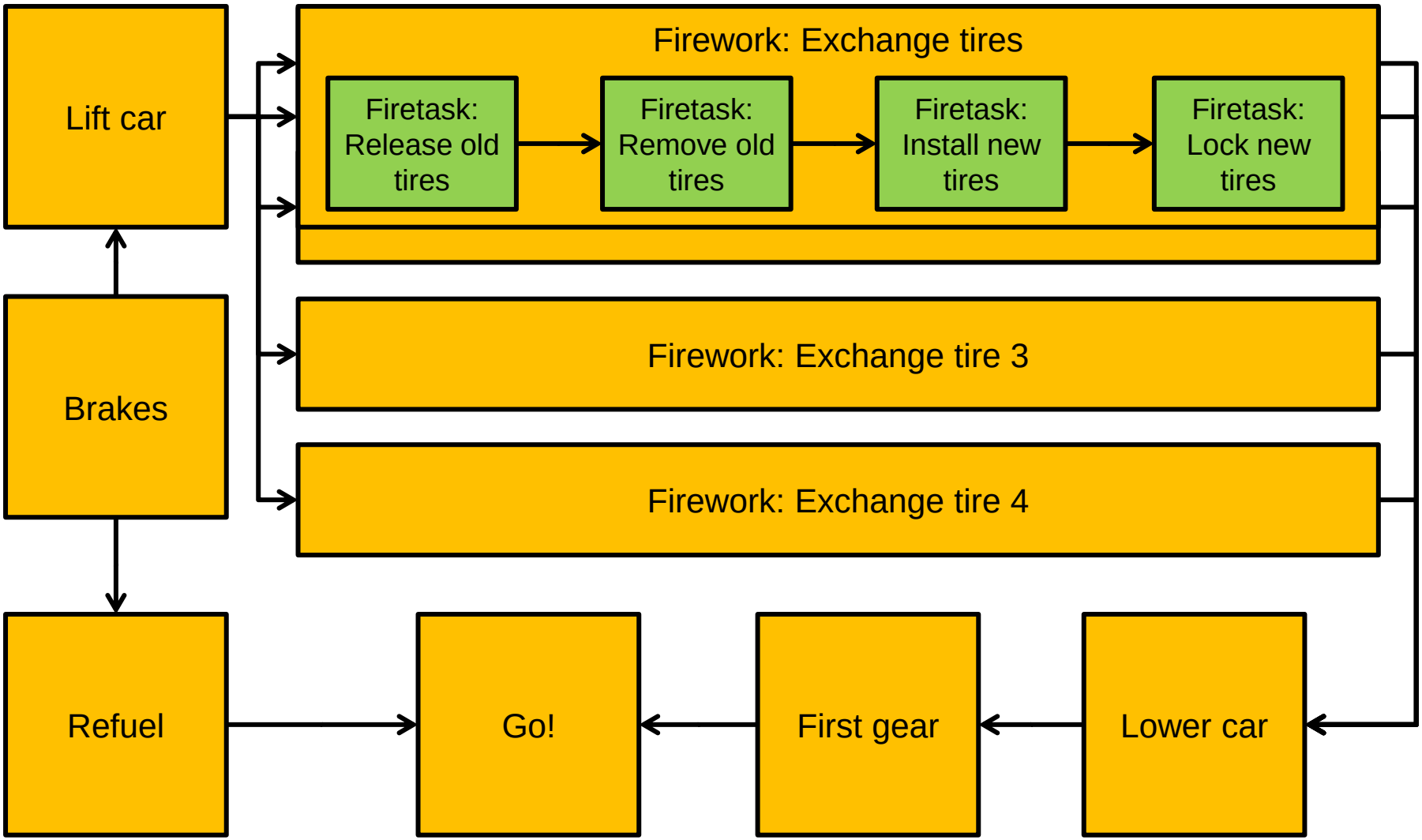
Solution graphs



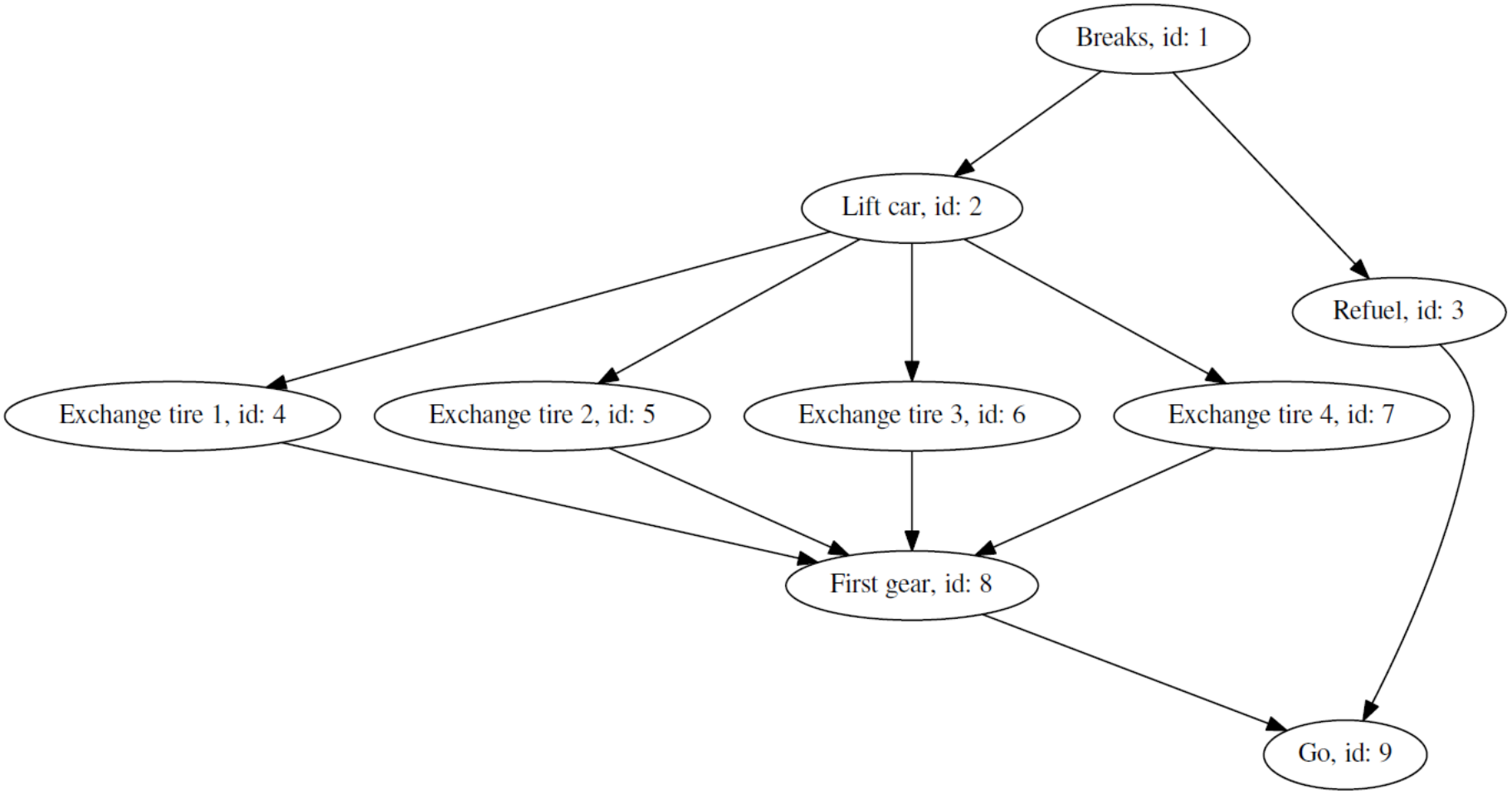




“Parallel” workflow



“Parallel” workflow



Exercise 1: Problems

Problem 1.1

- Switch to working directory and copy problems:

```
cd exercises/work/1_control_flow
cp ../../problems/1_control_flow/*.yaml .
```

- One by one, check the three sequential workflows for errors, and then find and correct the errors. Finally add the workflow to LaunchPad:

```
lpad add -c f1_pitstop_seq_wrong_2.yaml
```

- Query workflow state:

```
lpad get_wflows -d all
```

- Execute with:

```
rlaunch singleshot
```

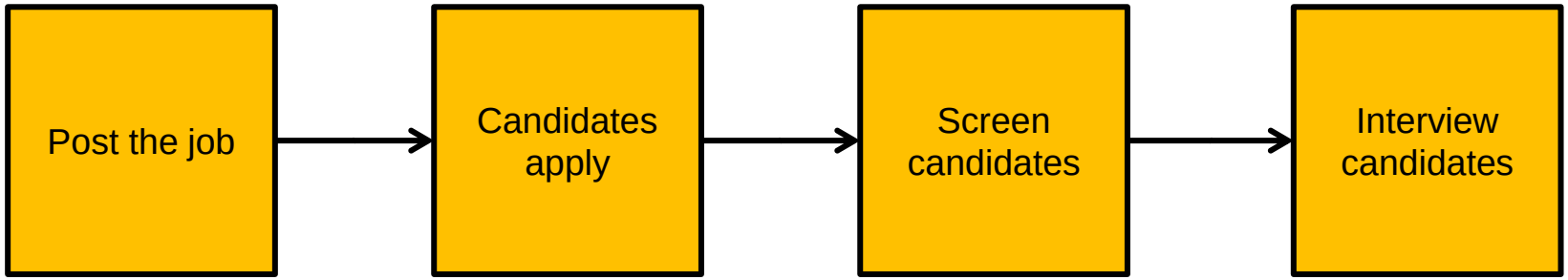
- After running each single Firework, monitor the output and the states of the Fireworks until the workflow is completed.

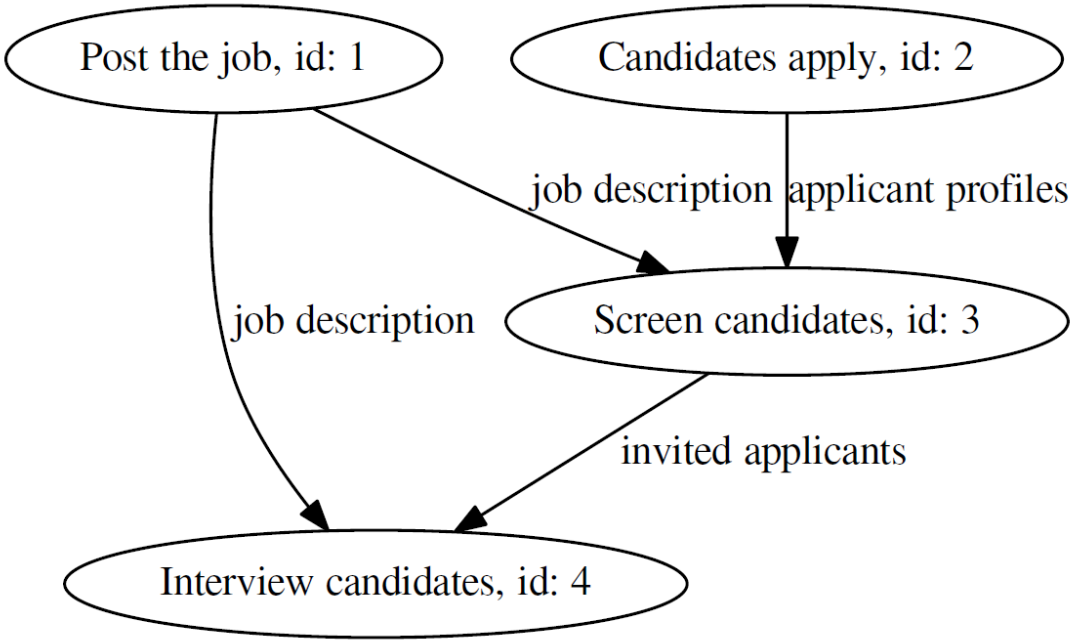
Problem 1.2

- Repeat the steps of **Problem 1.1** for the parallel version of the workflow

f1_pitstop_par_wrong_1.[json|yaml]
f1_pitstop_par_wrong_2.[json|yaml]
f1_pitstop_par_wrong_3.[json|yaml]

Exercise 2: Managing data flow





PythonFunctionTask

- Purpose: passes a list of data entities as positional arguments to a Python function and forwards the returned objects to the next Firework via **FWAction**.

- Mandatory parameters:
  - *function*: the name of the Python function

- Optional parameters:
  - *inputs*: a list of items for input that must be in *spec*
  - *outputs*: a list of returned objects to pass to *spec*

- See and run example under: **exercises/demos/2_data_flow**

```
- fw_id: 1
  name: Grind coffee
  spec:
    _tasks:
    - _fw_name: PythonFunctionTask
      function: auxiliary.print_func
      inputs: [roasted coffee beans]
      outputs: coffee powder
    roasted coffee beans: top selection
- fw_id: 2
  name: Brew coffee
  spec:
    _tasks:
    - _fw_name: PythonFunctionTask
      function: auxiliary.print_func
      inputs: [coffee powder, water]
      outputs: café solo
    water: workflowing water
```

Exercise 2: Problems

Problem 2.1

- Change to folder **exercises/work/2_data_flow**.
- Copy **exercises/inputs/2_data_flow/template.[json|yaml]**
- Complete the workflow to implement **exercises/problems/2_data_flow/recruiting-script.py**.
- Check the workflow, add it to LaunchPad and run it in ***singleshot*** mode.
- Watch the changes in the Fireworks with each rlaunch.

Problem 2.2

- Copy **recruiting-[012].[json|yaml]** from the folder **exercises/problems/2_data_flow**
- Detect and correct the errors and run the workflow in ***rapidfire*** mode.
- Compare the corrected versions to each other and to solution of Problem 2.1.
- Compare the results of two instances of the same workflow. Why do they differ?

```
- fw_id: 1
  name: Post the job
  spec:
    _tasks:
    - _fw_name: PythonFunctionTask
      function: auxiliary.print_func
      inputs: [job description]
      outputs: [job description]
    job description:
      title: chief fiction scientist
      contract conditions: {contract
type: fixed-term, salary: 400, social
insurance: true}
      qualifications: {academic
degree: master, education background:
fiction science, experience: 4,
skills: understanding science fiction}
      work description: work in
fictitious projects
```

Exercise 3: Using command line and files

```
#!/bin/bash -e

# Flip vertically
convert -flip piece-1.png bottom_right.png

# Rotate 90 degrees anti-clockwise
convert -rotate -90 piece-2.png top_left.png

# Flop horizontally top_left.png
convert -flop top_left.png top_right.png

# Flop horizontally bottom_right.png
convert -flop bottom_right.png bottom_left.png

# Put the four pieces together
montage -mode concatenate -tile 2x2 \
top_left.png top_right.png bottom_left.png \
bottom_right.png montaged_image.png
```

Command

Input of type *path*

Output of type *path*

Input type *data,* value -90, prefix *-rotate*, separator ' '

Command in Subprocess style: [montage, -mode, concatenate, -tile, 2x2]

4 inputs of type *path*

1 output of type *path*

Exercise 3: Using the CommandLineTask

- Required parameters
  - *command_spec*: specification of the command

```
command_spec:
  binding:
    prefix: [string, null]
    separator: [string, null]
  source:
    type: [path, data, identifier, stdin, stdout, stderr, null]
    value: [string, number, null]
  target:
    type: [path, data, identifier, stdin, stdout, stderr, null]
    value: string
```

- Optional parameters:
  - *inputs*, *outputs*: the same as for PythonFunctionTask
  - But they are in *command_spec*

```
- fw_id: 1
  name: Image rotation step
  spec:
    _tasks:
    - _fw_name: CommandLineTask
      inputs: [rotation angle, original image]
      outputs: [rotated image]
      command_spec:
        command: [convert]
        original image:
          source: original image
        rotated image:
          target: {type: path, value: /tmp}
        rotation angle:
          binding:
            prefix: -rotate
            separator: ' '
          source: {type: data, value: -90}
    original image:
      type: path
      value: /tmp/piece-2.png
```

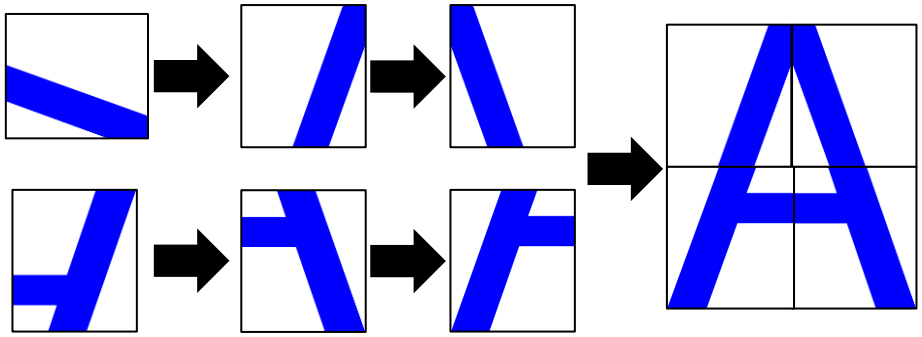
Visible only for this Firetask

Shared with other Firetasks or from parent Fireworks

Exercise 3: Problems

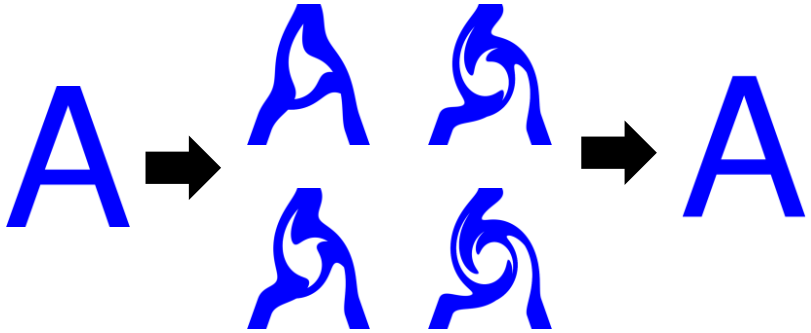
Problem 3.1

- Reconstruct capital letters from one of four pieces: X, O
- Reconstruct capital letters from two of four pieces:
  A, C, D, E, H, M, N, S, T, U, V, W, Y, Z



Problem 3.2

- Swirl the image at different angles: 90°, 180°, 270° and 360°
- Animate the images in the sequence:
  0° → 90° → 180° → 270° → 360° → 270° → 180° → 90° → 0°



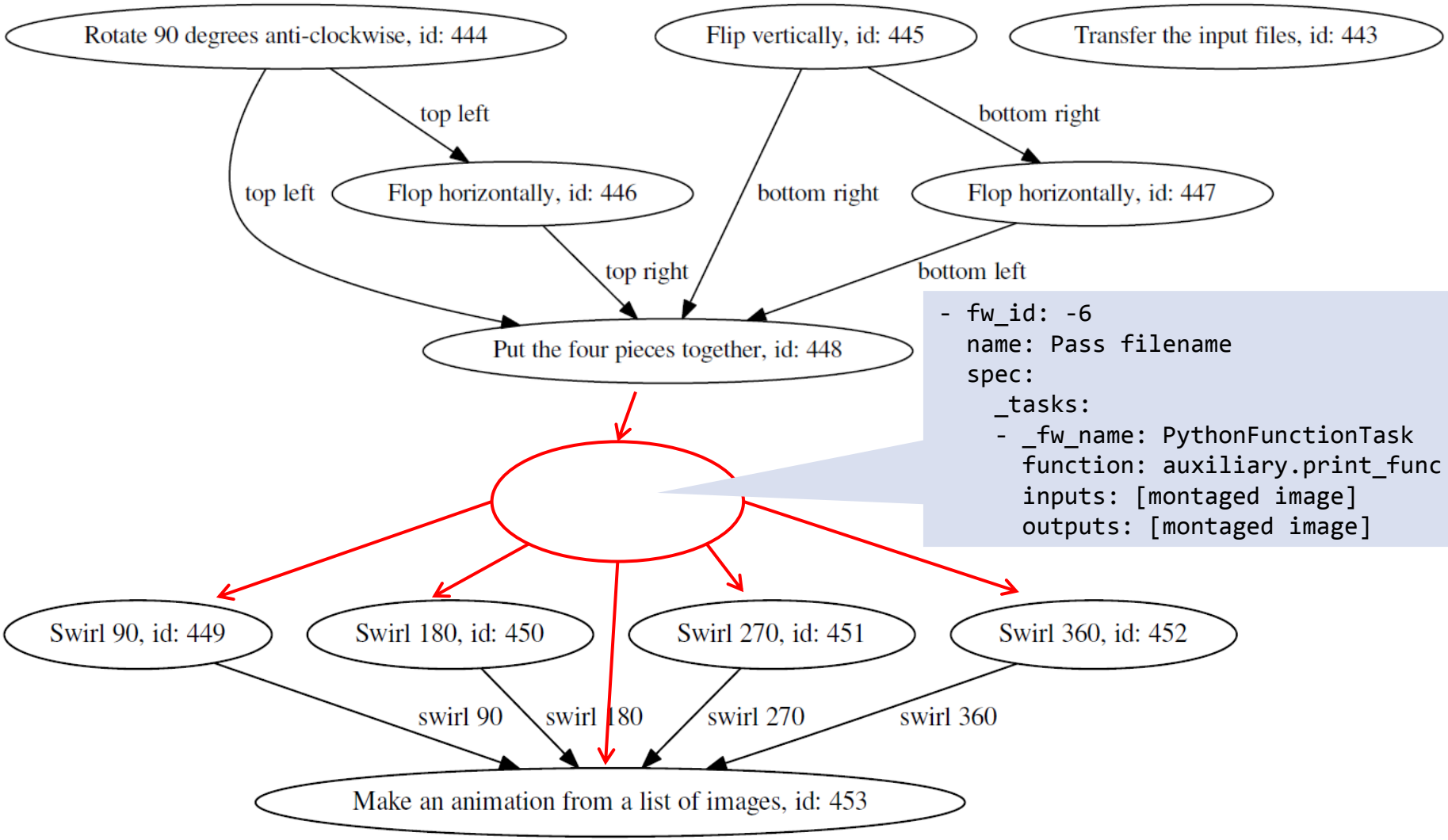
Available operations (in the demos):

- Rotate +90 degrees
- Horizontal flip
- Vertical flip
- Montage

Available operations (in the demos):

- Swirl
- Animate

Exercise 4: Extending a workflow




```
- fw_id: -6
  name: Pass filename
  spec:
    _tasks:
    - _fw_name: PythonFunctionTask
      function: auxiliary.print_func
      inputs: [montaged image]
      outputs: [montaged image]
```

Problem 4.1

```
cp exercises/solutions/3_files_and_commands/image_swirl.json \
image_swirl_montaged.json
```

- Change all Firework IDs to become negative integers
- Add the linking Firework

```
- fw_id: -6
  name: Pass filename
  spec:
    _tasks:
    - _fw_name: PythonFunctionTask
      function: auxiliary.print_func
      inputs: [montaged image]
      outputs: [montaged image]
```

- Add '-6': [-1, -2, -3, -4, -5] to the links
- Set original image: {source: montaged image}

```
lpad get_fws -n "Put the four pieces together"
lpad append_wflow -i <ID> -f image_swirl_montaged.json
lpad get_wflows -s "RUNNING"
rlaunch rapidfire
```

Exercise 5: Writing a Firetask

Firetask

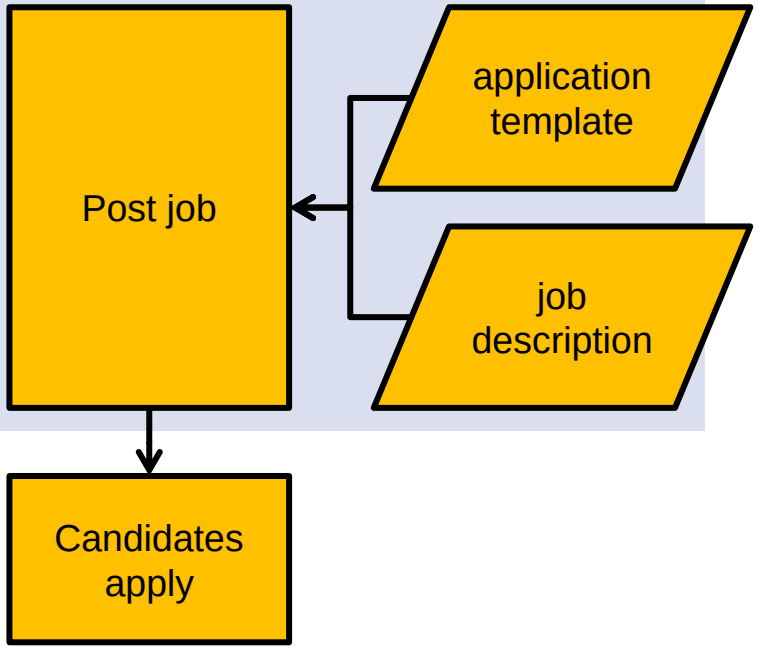
```
from fireworks.core.firework import FiretaskBase, FWAction
from fireworks.utilities.fw_utilities import explicit_serialize
@explicit_serialize
class MyFiretask(FiretaskBase):
    _fw_name = 'MyFiretask'
    required_params = ['par1', 'par2']
    optional_params = ['optional par']
    def run_task(self, fw_spec):
        if self.get('optional par'):
            actions = [
                'update_spec': {
                    'par1': self['par1'],
                    'optional par': self['optional par']
                }
            ]
        else:
            actions = []
        return FWAction(*actions)
```

Firework

```
- fw_id: 1
  spec:
    _tasks:
    - _fw_name: {{custom_tasks.MyFiretask}}
      par1: data.json
      par2: data.yaml
      optional par: {}
```

Problem 5.1: Data Loader Firetask

```
from fireworks.core.firework import FiretaskBase, FWAction
from fireworks.utilities.fw_utilities import explicit_serialize

@explicit_serialize
class ' '(FiretaskBase):
    _fw_name = ' '
    required_params = [' ', ' ']

    def run_task(self, fw_spec):
        json.load(' ')
        return FWAction(update_spec={' ': self[' ']})
```



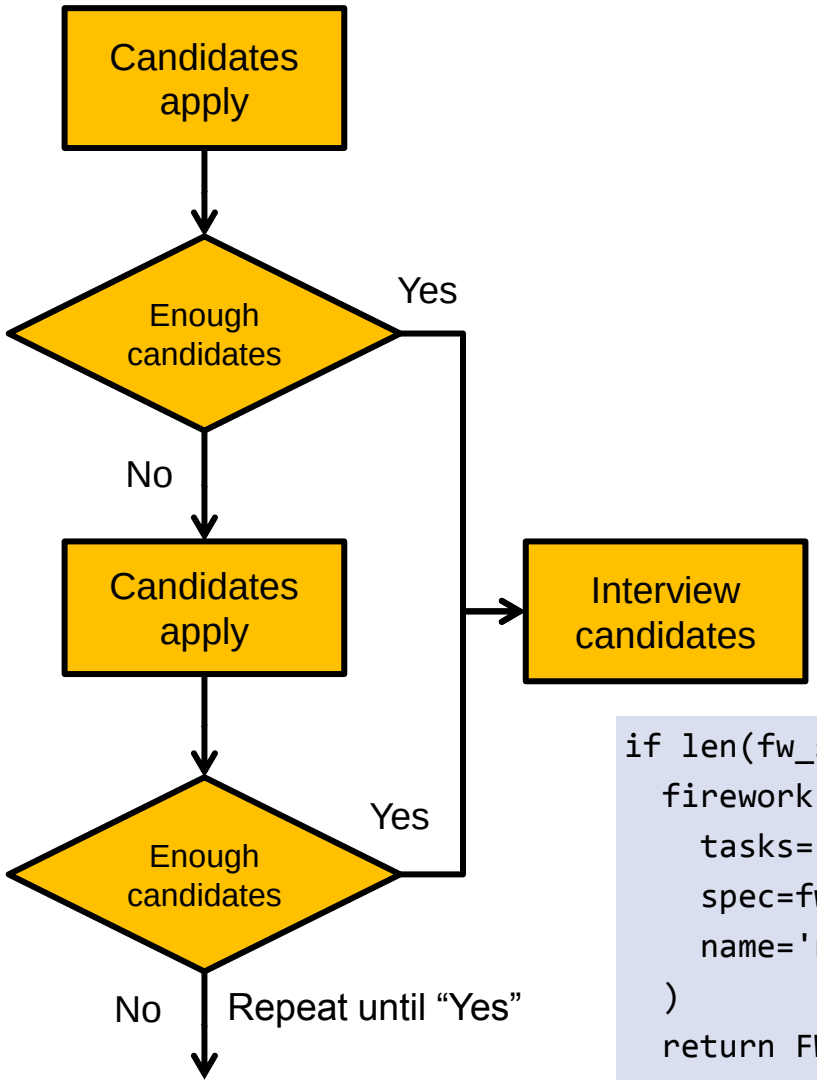
- Use the provided snippet as template
- Use the json.load() method
- Return a FWAction object with update_spec
- Register the Firetask:

```
export PYTHONPATH=`pwd`:$PYTHONPATH
```

- Add the workflow to LaunchPad:

```
lpad add exercises/problems/5_author_firetask/dataloader.json
```

Problem 5.2: Conditional Repeater Firetask



Firework

```
- fw_id: 2
  name: Candidates apply
  spec:
    _tasks:
    - _fw_name: PythonFunctionTask
      function: recruiting.candidates_apply
      inputs:
      - application template
      - maximum applications
      outputs: [applicant profiles]
    - _fw_name: RepeatIfLengthLesser
      measure: applicant profiles
      minimum: number to screen
number to screen: 10
```

Firetask

```
if len(fw_spec[self['measure']]) < fw_spec[self['minimum']]:
  firework = Firework(
    tasks=[load_object(task) for task in fw_spec['_tasks']]
    spec=fw_spec
    name='repeat '+self['measure']
  )
  return FWAction(detours=firework)
```

Legal notice