

AlphaFold Workshop

Lesson 9

Revision 2

Petr Kulhánek

kulhanek@chemi.muni.cz

Laboratory of Computational Chemistry
National Centre for Biomolecular Research
Faculty of Science
Masaryk University
Kamenice 5
CZ-62500 Brno

Infinity

Jobs

Infinity Subsystems

Infinity is composed of two subsystems:

- **AMS** - Advanced Module System for software management
- **ABS** - **Advanced Batch System for job management**

Requested vs Used Resources

Batch system

Job

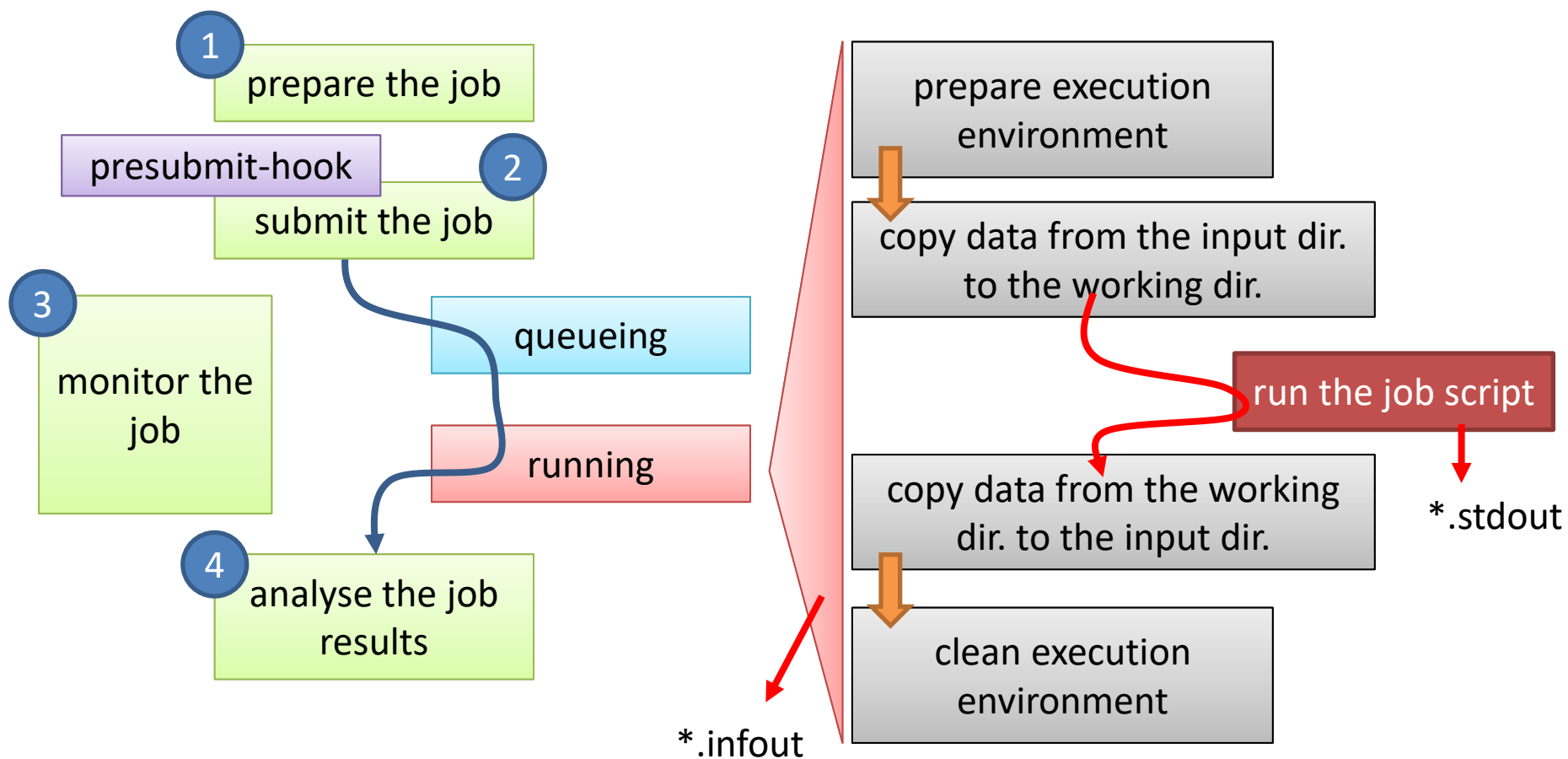
Native batch system (PBSPro)

- **the user specifies** the required computing resources
- **the user must ensure** that the job uses the assigned computing resources

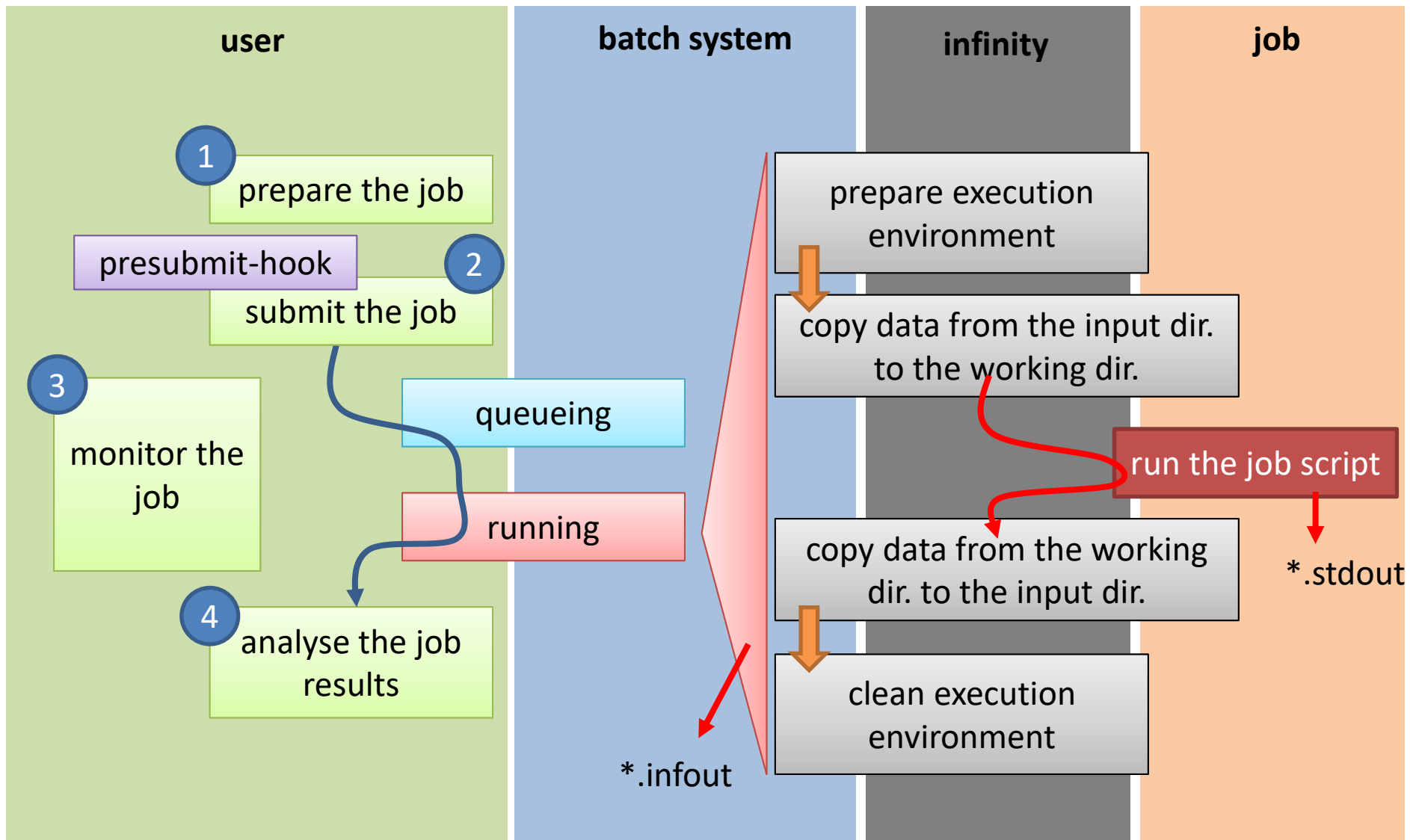
Infinity

- **the user specifies** the required computing resources
- **Infinity environment ensures correct job execution** (selected applications only)
- (other jobs) **the user must ensure** that the job uses the assigned computing resources

Infinity Job Lifetime



Infinity Job Lifetime



Overview of commands

Software management:

- **site** activation of logical computing resources
- **module** activation/deactivation of software

Job management:

Batch system status:

- **pqueues** overview of batch system queues available to the user
- **pnodes** overview of computing nodes available to the user
- **pqstat** overview of all tasks submitted into the batch system
- **pjobs** overview of user tasks submitted into the batch system

Job commands:

- **psubmit** submitting a job into the batch system
- **pinfo** job information
- **pgo** logs the user on to the computing node
- **psync** manual data synchronization

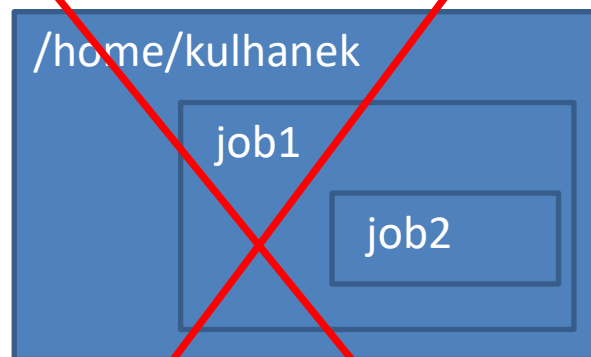
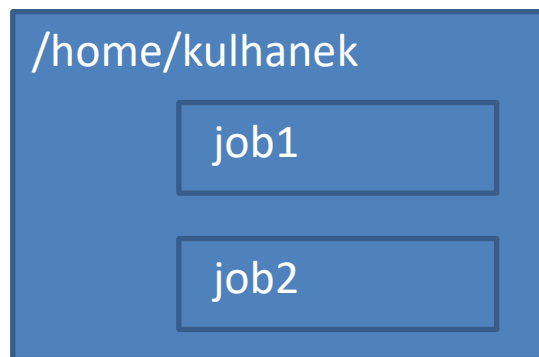
Exercise I

- 1) Examine the configuration of the cluster. Use the following commands:
 - pqueues
 - pnodes

Job

Each job managed by Infinity **must fulfil the** following conditions:

- **job is a directory (!!!!!! NO exception !!!!!)**
- all job input data must be in the job directory
- job directories must not be nested
- job execution is controlled by a bash script
- absolute paths must not be used in the job script; all paths must be provided relative to the job directory (some exceptions are possible)



Job script

The job script is interpreted by the standard bash interpreter or a special **infinity-env** interpreter. This **infinity-env** protects the job from accidental execution, which can lead to possible damage/overwriting/deletion of already calculated data.

```
#!/bin/bash
```

```
# script itself
```

```
#!/usr/bin/env infinity-env
```

```
# script itself
```

NOTE: If you want to preserve a **syntax highlighting** in a text editor, name the file with a **.sh** extension.

Submitting a job

The job is submitted from the **job input directory** by the **psubmit** command:

```
psubmit destination job [resources]
```

destination (where) is either:

- queue name
- alias

job (what) is either:

- job script name
- input file name for automatically detected jobs

resources (how) specify requested resources for the job, if not specified, one CPU is requested

Resource specification (selected)

Source	Description
ncpus	requested number of CPUs
ngpus	requested number of GPUs
nnodes	Requested number of computational nodes (WN)
mem	total amount of requested memory (for CPU), unit mb, gb
gpu_mem	minimum GPU memory per a single GPU accelerator
walltime	maximum job run time
workdir	type of working directory on WN
place	method of occupying computing nodes (scatter, pack, free, excl)
props	required properties of computational nodes

Exercise II

Submit the following job and monitor its progress:

- 1) Create the job input directory

```
$ mkdir -p ~/AlphaFold/L09.infinity
$ cd ~/AlphaFold/L09.infinity
$ pwd
```

- 2) Create the job script

```
$ gedit my_job &
```

- 3) Submit the job

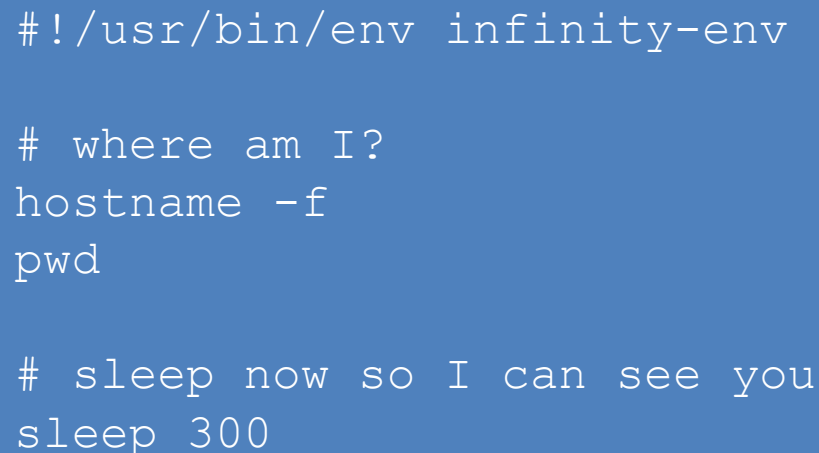
```
$ psubmit ncbw_workshop my_job
```

- 4) Monitor the jobs

- **pjobs, pqstat**
- **ls, pinfo, pgo**

- 5) Analyse the job results

```
$ gedit my_job.stdout &
```



```
#!/usr/bin/env infinity-env
# where am I?
hostname -f
pwd

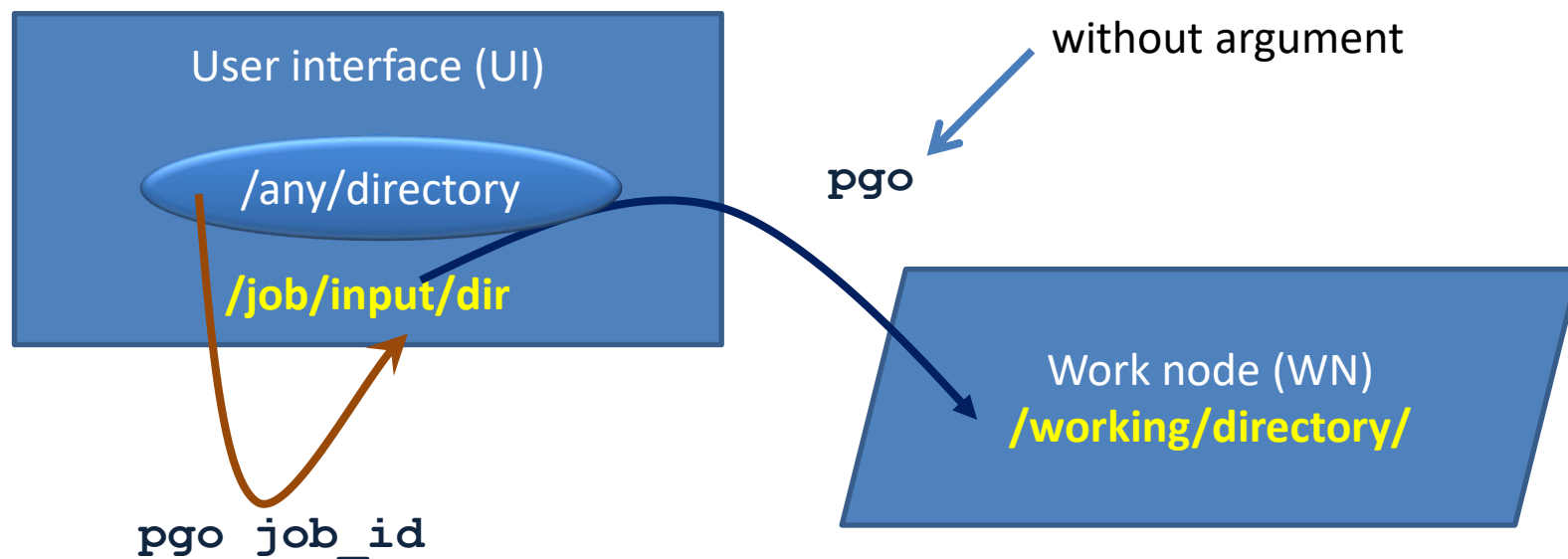
# sleep now so I can see you
sleep 300
```

Monitoring Job Progress

The job progress can be monitored by the **pinfo** command executed in the job input or working directory.

Other options are commands **pjobs** and **pqstat**.

If the job is already running, you can use the **pgo** command, which logs onto the work node and changes the current directory to the job working directory.



Monitoring the job in the terminal.

Runtime Files

In the job directory, service files are created when the job is submitted into the batch system, during the life of the job and after its completion.

Their meaning is as follows:

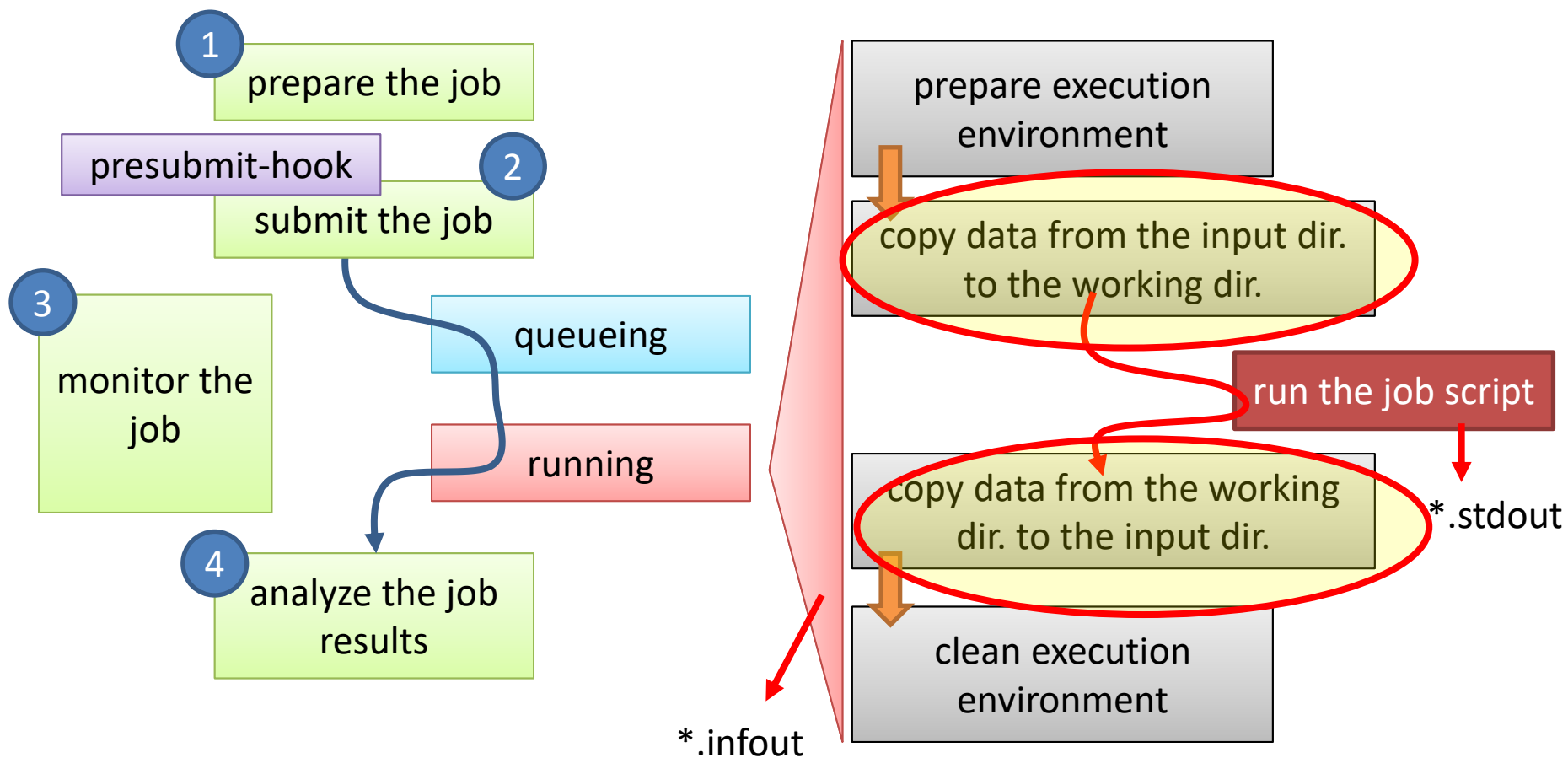
- *.info control file with information about the progress of the task
- *.infex custom script (wrapper), which is run by the batch system
- *.infout standard runtime output of *.infex script, **must be analyzed when the task terminates abnormally**
- *.nodes list of nodes reserved for the job
- *.mpinodes list of nodes reserved for the job in format for OpenMP
- *.hwtopo hardware topology of the work node
- *.infkey unique job identifier
- *.stdout **standard output from running the job script**

Do not delete these files. They provide long-term logging and bookkeeping information about the job.

They can be deleted by: `premovertf`

Infinity Job Lifetime

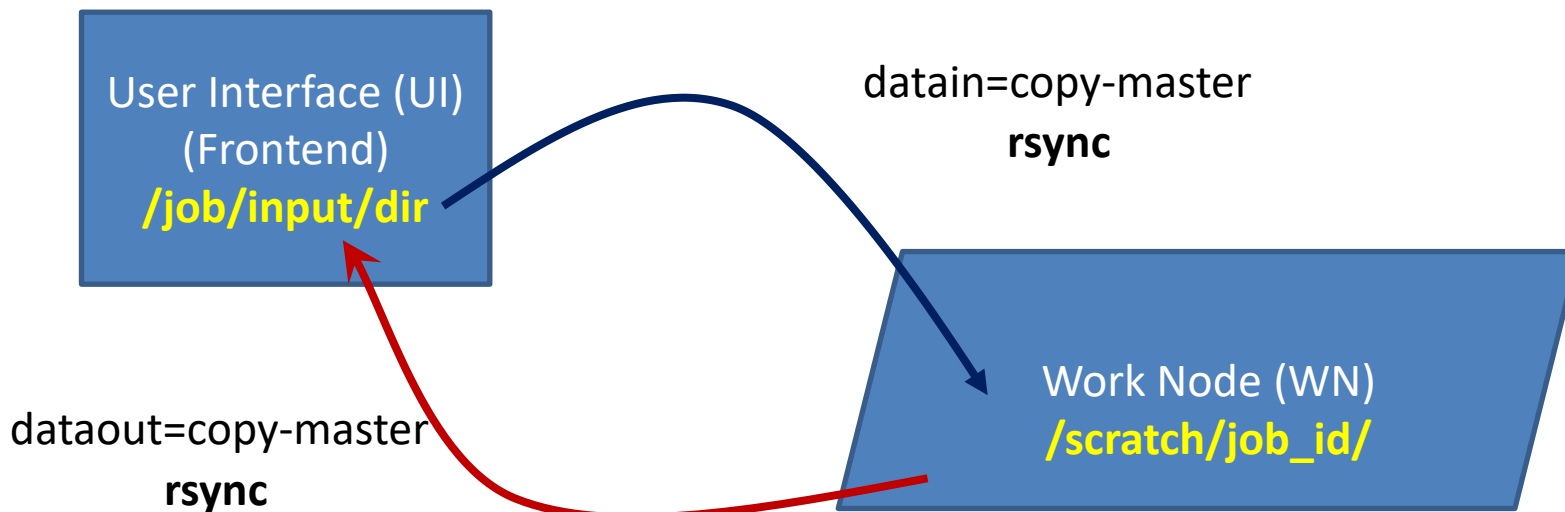
Data management



Data Synchronization, I

Default operating mode

Source	Meaning
<code>workdir=scratch-local</code>	Data is copied from the job input directory to the working directory on the work node. The working directory is created at the beginning of the job by the batch system. When the job is completed, all data from the working directory is copied back to the job input directory. Eventually, the working directory will be deleted if the data transfer was successful.



Data Synchronization, II

Suitable for analysis

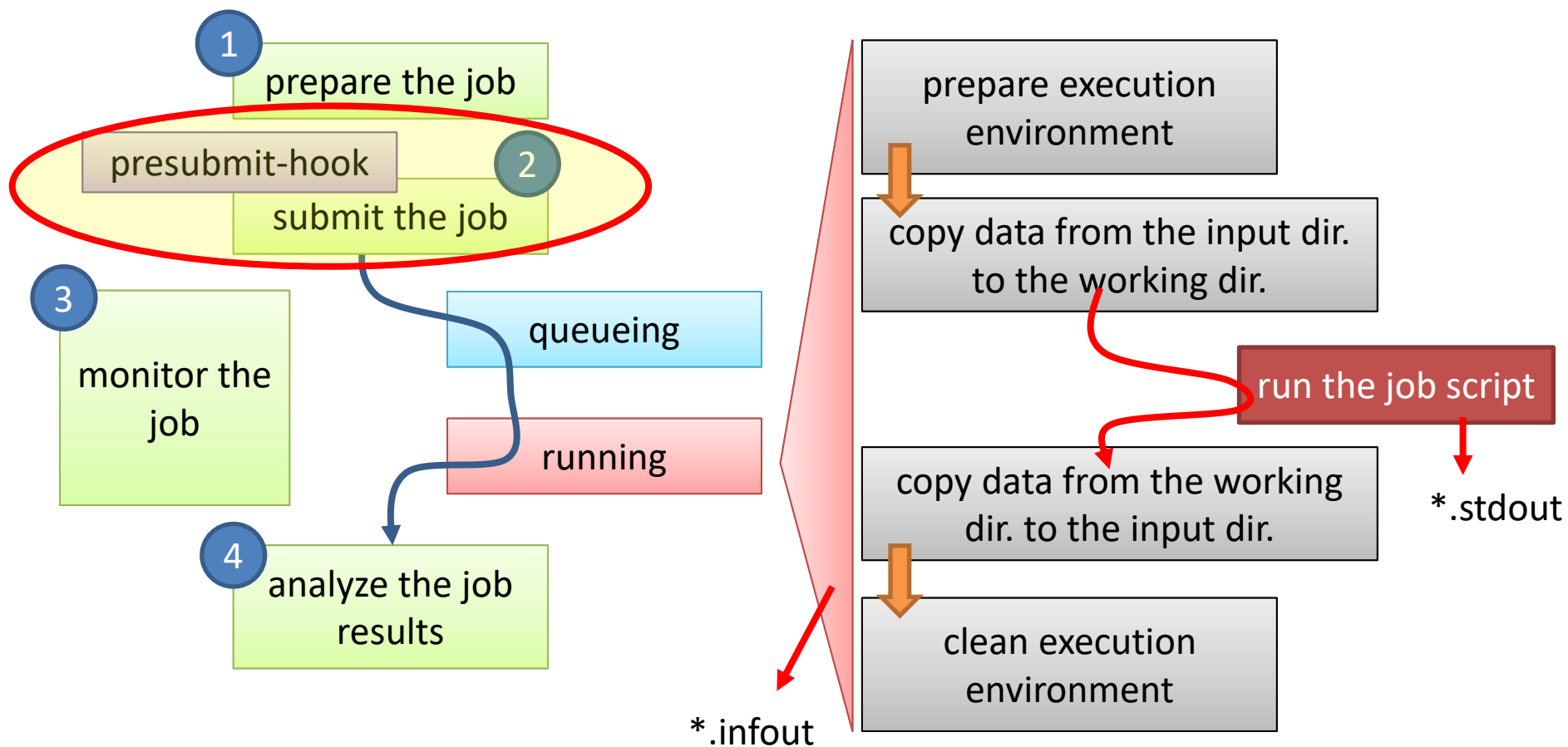
Source	Meaning
<code>workdir=jobdir</code>	Job data is on shared storage.



Suitable for large amount of data, which are sequentially read during the analysis.
Results are stored in small files.

Infinity Job Lifetime

presubmit-hook



presubmit-hook

if the **presubmit-hook** files exists in the input job directory, it is treated as a bash script and executed by the **psubmit** command **before the job is submitted** into a batch system.

The typical usage of **presubmit-hook** is to automatically prepare input data for the job, for example by copying data from the previously executed jobs.

If the exit code from the presubmit-hook script execution is non-zero, the psubmit command will fail too and the job will not be submitted to the batch system. This safety-measure works only if the error states are handled correctly in the script.

Examples:

```
#!/bin/bash
cp ../01.get-dihedrals/dihedrals.list dihedrals.list
cp ../02.opt-HF-3c/last.xyz input.xyz
cp ../02.opt-HF-3c/opt.orca.gbw opt.orca.gbw.prev
```

wrong

if two first cp commands fail but the last will be OK, final exit code will be OK.

```
#!/bin/bash
cp ../01.get-dihedrals/dihedrals.list dihedrals.list || exit 1
cp ../02.opt-HF-3c/last.xyz input.xyz || exit 1
cp ../02.opt-HF-3c/opt.orca.gbw opt.orca.gbw.prev
```

correct

fail at any failure