BEE: Build and Execute Environment
A Workflow Orchestration System

Manage Multi-step Simulations on HPC & Cloud Platforms

Patricia Grubel

Salishan Conference on High Speed Computing

April 28, 2022

LA-UR-22-23300

BEE Team

Tim Randles (PI)
Steven Anaya
Rusty Davis
Patricia Grubel
Qiang Guan
Jake Tronge

Alumni

Paul Bryant
Jieyang Chen
Ragini Gupta
Allen McPherson
Li-Ta Lo
Quincy Wofford

Contact: bee-dev@lanl.gov

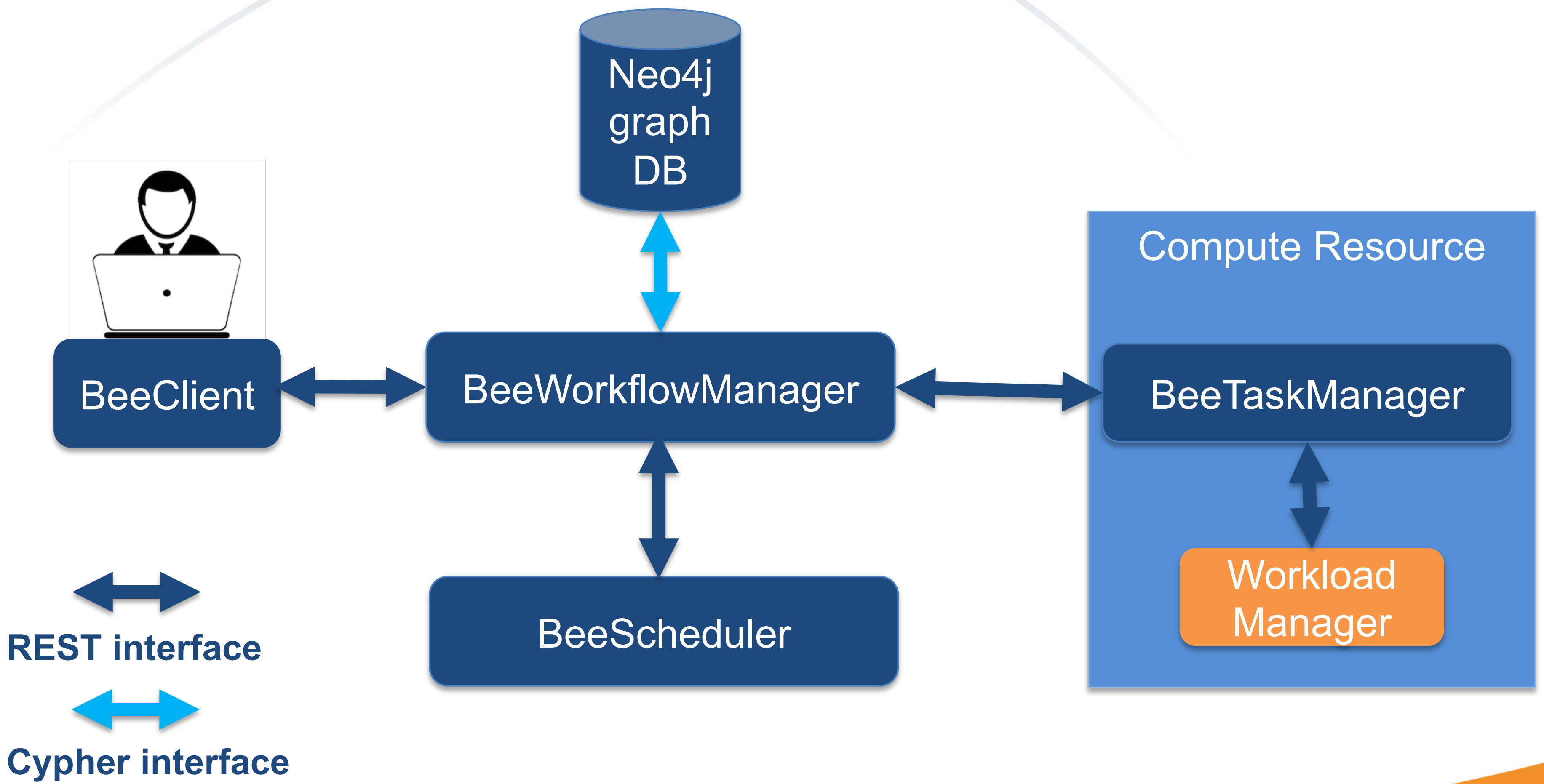
BEE: A Workflow Orchestration System

- Exascale Computing Project (ECP)
- HPC Container Support
  - Configurable support for HPC - Charliecloud and Singularity Container Runtimes
- Supports Multiple Compute Resources
  - HPC and Cloud
  - Enables high resource usability
- Designed for HPC simulations
- Open Standard Workflow Specification: Common Workflow Language (CWL)
- Automation
  - Platform-related setup, configuration and launching
  - Avoid learning arcane technical details of HPC resource managers
- No privileged access required
  - Any user can use on HPC platform of their choice
- Reproducible Workflows
  - Complex scientific workflows can be archived, share metadata, re-run

BEE: Internals

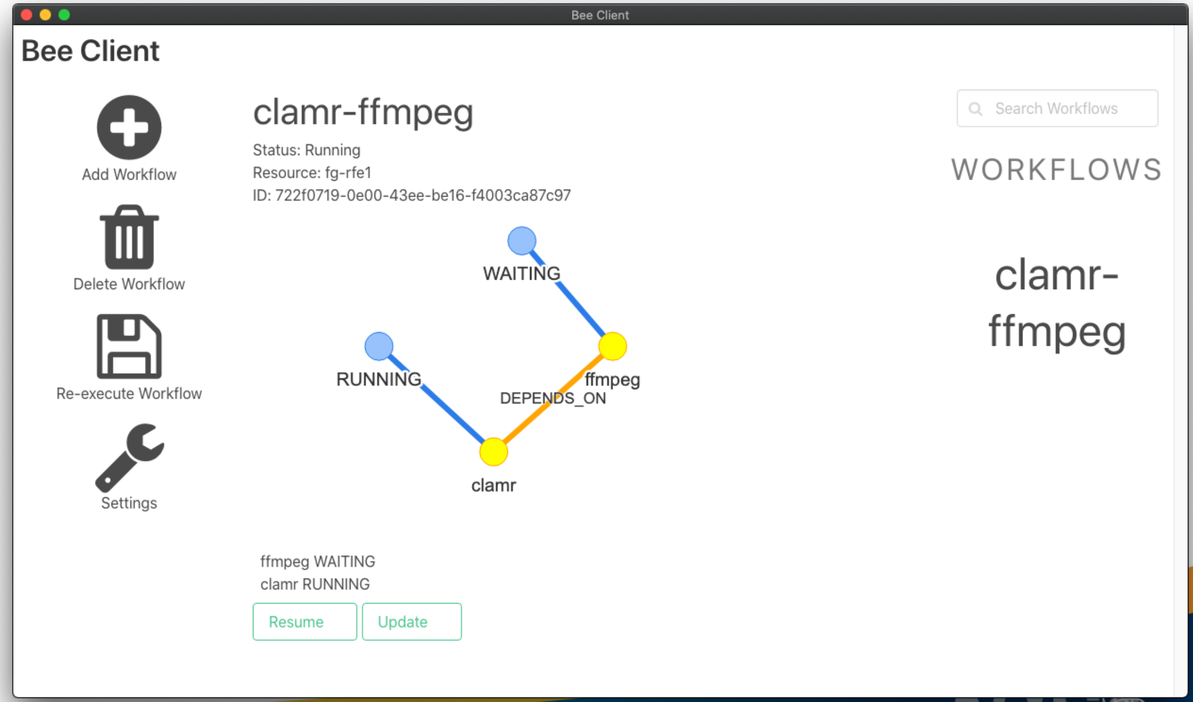
- Python 3
  - Portable across Linux, OS X, Windows
  - Modular Design – Abstract Classes for Major Components
    - Abstract interfaces define each component's API
    - Drivers implement API for specific technology (can easily add new technologies)
    - Support for using different Graph Database (Neo4j)
    - Support for multiple Container Runtimes (Charliecloud, Singularity)
    - Support for Multiple Workload Managers (Slurm, LSF, PBS, Torque/Moab…)
- RESTful API's – Main BEE components expose REST endpoints
  - Easy to enhance and extend new services
- Common Workflow Language (CWL)
  - Open standard, many tools already exist
  - BEE extensions for better HPC support
    - Automatic Setup of HPC Requirements
    - MPI requirement extensions
- Neo4j - Graph database
  - Manage workflow and metadata
  - DAG allows workflow execution
  - Archive workflow & artifacts

BEE: Components for HPC



BEE Client

- Allows user to:
  - Add, start, pause, re-run, or cancel workflows
  - View list of workflows (archived, running or pending)
  - Visualize DAG – shows dependencies & task states
  - Get updates
  - Setup config



Neo4j - Graph Database (GDB)

- Manage Workflow
  - Build Workflow DAG monitors dependencies
  - Visualize DAG
  - Metadata during run – task state, job id …
  - Sends ready tasks to WF Manager
- Archive Workflow
  - Workflow metadata - what cluster, cluster job ids, status, times (submit, start, compute)
  - Provenance - ability to archive the workflow
    - Captures container UUID, input decks, run commands, checkpoint file location
    - Rerun workflows
    - Clone workflows - copy, reset data and go
  - Resiliency – true state of the BEE workflow is in the database
    - Recovery from outages
    - Component restart - components designed to continue using database metadata
  - Checkpoint / Restart metadata

BEE Workflow Manager

- Manages Interactions between all components
- Parses CWL sends WF to GDB
- Receives Ready Tasks
- Sends Ready Tasks to the Task Manager
- Receives job information from TM and relays state to GDB
- Archives Completed Workflows

BEE Task Manager

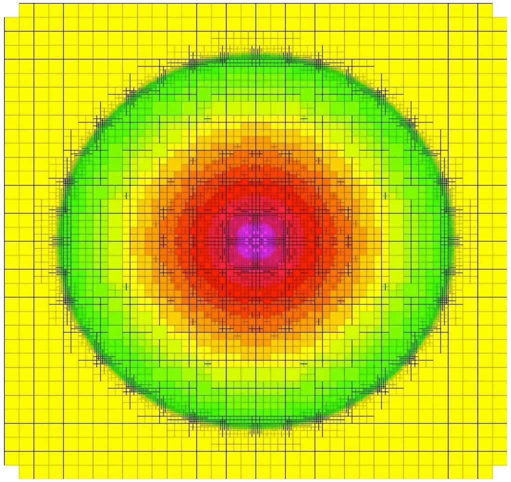
- Receives ready tasks from WF Manager
- Pre-processes Build Requirements
  - Container pulls, builds or copies as defined in task requirements and as needed
- Submits job as defined by each task
  - Uses containerized applications or applications installed on the system as specified by user CWL
- Keeps track of active tasks
- Queries job states
- Sends updates to WF Manager

BEE Scheduler

- Designed to be extensible, easy to configure for user workflows
- To be implemented
  - Allow for users to take advantage of cloud resources or other HPC systems when load is high
  - Choose between multiple Task Managers when available [not implemented yet]

CLAMR Workflow Example

- CLAMR
  - An open source LANL mini-app
  - Simulates shallow water equations
  - Performs hydrodynamic cell-based adaptive mesh refinement (AMR)
  - Intended as a testbed for hybrid algorithm development using MPI and OpenCL



CLAMR Visualization

https://github.com/lanl/CLAMR

CLAMR Workflow CWL

clamr_wf.cwl

```
cwlVersion: v1.0

inputs:
  grid_resolution: int
  max_levesl: int
  …
outputs:
  clamr_stdout:
    type: File:
    outputSource: clamr/clamr_stdout
   …

steps:
  clamr:
    run: clamr.cwl
     in:
       grid_res: grid_resolution
       max_levels: max_levels
        …
    out: [clamr_stdout, outdir, time_log]
    hints: DockerRequirement
          copyContainer:: “…/clamr.tar.gz”

  ffmpeg :
    run: ffmpeg.cwl
     in:
       input_format: input_format
       ffmpeg_input: clamr/outdir
     …
  out: [ movie ]
```

clamr.cwl

```
cwlVersion: v1.0

baseCommand: /clamr/CLAMRmaster/clamr_cpuonly
stdout: clamr_stdout.txt
inputs:
  amr_type:
    type: string?
    inputBinding:
      prefix: -A
  grid_res:
    type: int?
    inputBinding:
      prefix: -n
  …
outputs:
  outdir:
    type: Directory
    outputBinding:
      glob: $HOME/graphics_output/graph%05d.png
  …
```

CLAMR Workflow CWL

clamr_wf.cwl

```
cwlVersion: v1.0

inputs:
  grid_resolution: int
  max_levelsl: int
  …
outputs:
  …
  clamr_movie:
    type: File
    outputSource: ffmpeg/movie

steps:
  clamr:
    run: clamr.cwl
     in:
      grid_res: grid_resolution
      max_levels: max_levels
       …
    out: [clamr_stdout, outdir, time_log]
    hints: DockerRequirement
           copyContainer::  "…/clamr.tar.gz"

  ffmpeg :
    run: ffmpeg.cwl
     in:
       input_format: input_format
       ffmpeg_input: clamr/outdir
     …
  out: [ movie ]
```

ffmpeg.cwl

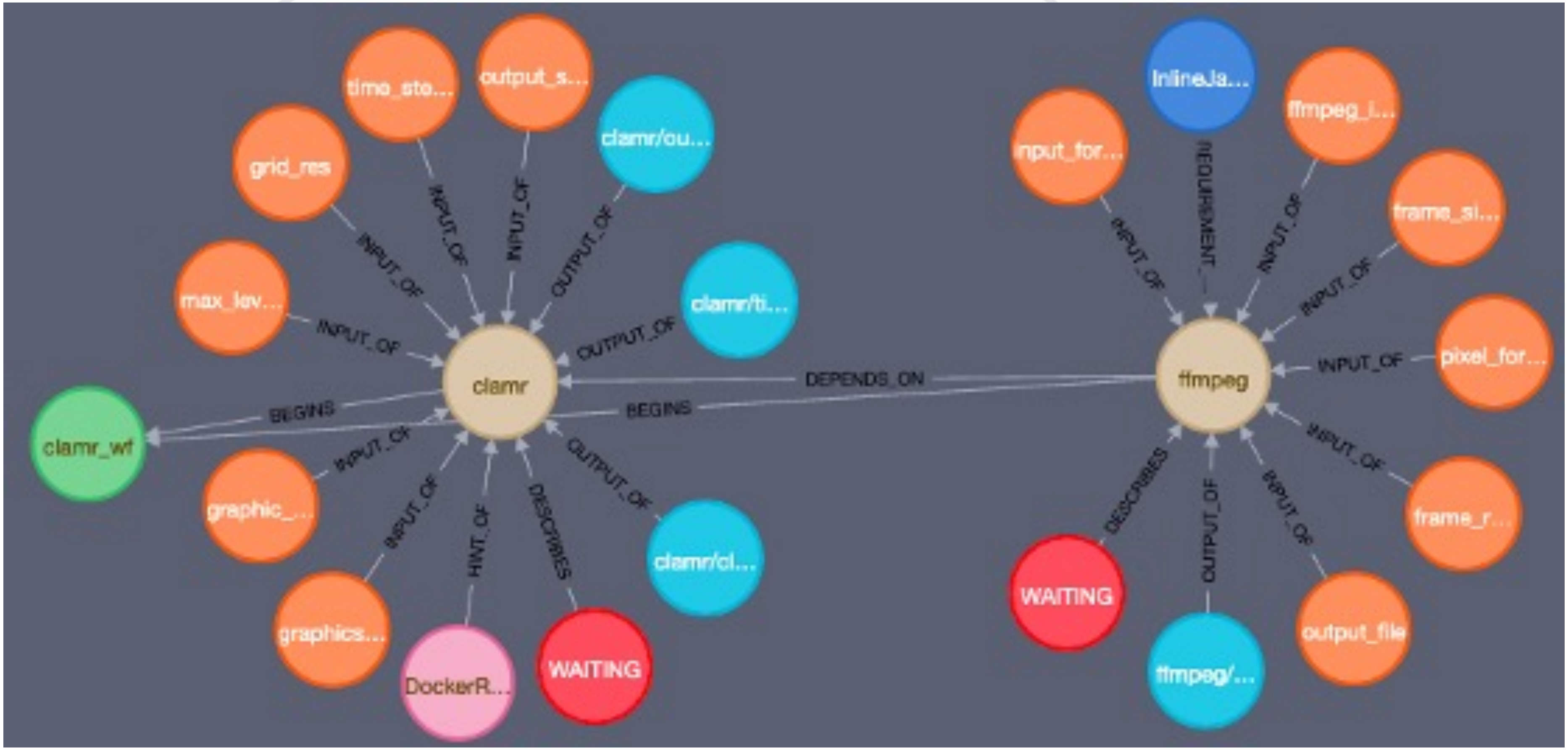
```
cwlVersion: v1.0

baseCommand: ffmpeg -y
inputs:
  input_format:
    type: string?
    inputBinding:
      prefix: -f
      position: 1
  ffmpeg_input:
    type: Directory
    inputBinding:
      prefix: -i
      position:2
      valueFrom: $(self.path + "/graph%5d.png")
  …
outputs:
  movie:
    type: File
    outputBinding:
      glob: $(inputs.output_file)
      # glob: CLAMR_movie.mp4
  …
```

CLAMR Workflow CWL

clamr_job.yml

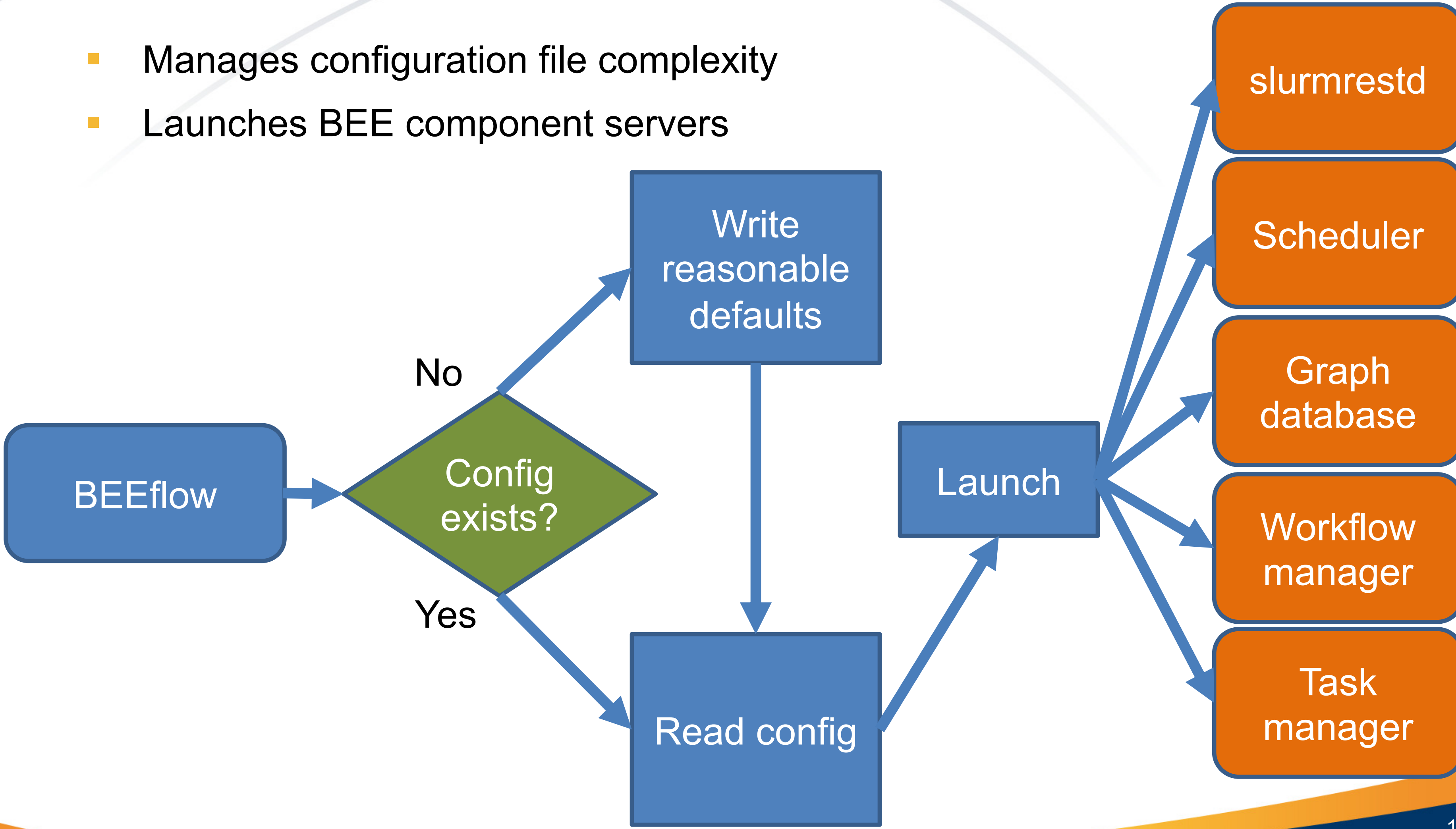
```
# Inputs for CLAMR
# /clamr/CLAMR-master/clamr_cpuonly -n 32 -l 3 -t 5000 -i 10 -g 25 -G png

grid_resolution: 32
max_levels: 3
time_steps: 5000
steps_between_outputs: 10
steps_between_graphics: 25
graphics_type: png

# Inputs for FFMPEG
# ffmpeg -f image2 -r 12 -s 800x800 -pix_fmt yuv420p $HOME/CLAMR_movie.mp4

input_format: image2
frame_rate: 12
frame_size: 800x800
pixel_format: yuv420p
output_filename: $HOME/CLAMR_movie.mp4
```

CLAMR neo4j Workflow



BEEflow

- Manages configuration file complexity
- Launches BEE component servers

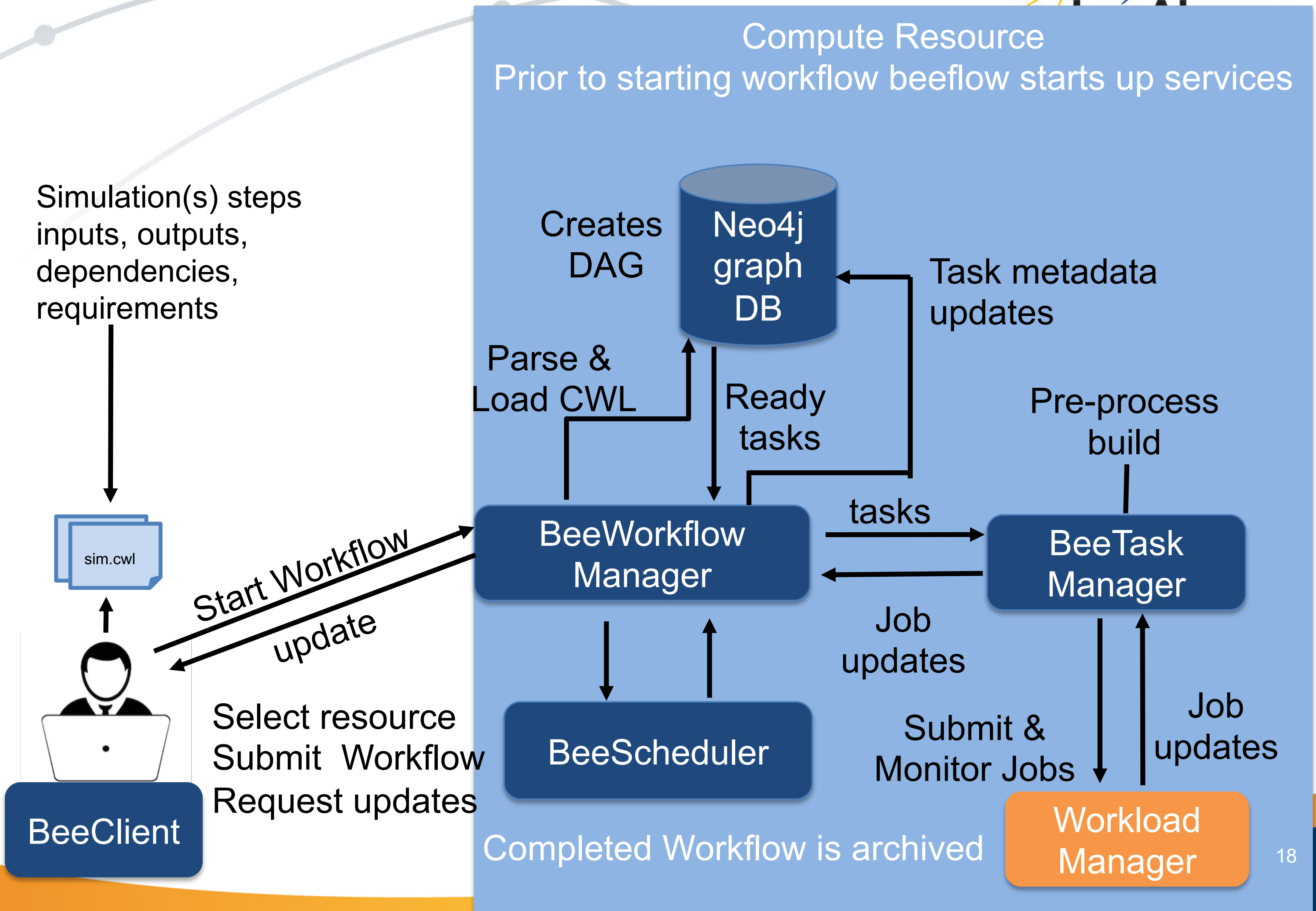


BEE Configuration File

- Orchestrates BEE components

```
# BEE CONFIGURATION FILE #
[DEFAULT]
bee_workdir = /users/<username>/.beeflow
workload_scheduler = Slurm
use_archive = True

[slurmrestd]
slurm_socket = /tmp/slurm_<username>_127.sock

[workflow_manager]
listen_port = 5827

[task_manager]
listen_port = 5877
container_runtime = Charliecloud

[graphdb]
hostname = localhost
dbpass = password
bolt_port = 7718
http_port = 7509
https_port = 7504
gdb_image = /usr/projects/beedev/neo4j-3-5-17-ch.tar.gz
gdb_image_mntdir = /tmp/<username>

[builder]
container_archive = /yellow/users/<username>/.beeflow/container_archive
deployed_image_root = /var/tmp/<username>/beeflow_deployed_containers
```

Current BEE Workflow *



Status and Future Plans

- Current Status
  - Public Release
  - GUI client runs on desktop/laptop – select resource
  - BEE components run on the resource (HPC/cloud)
- Upcoming Milestones
  - Restart/Checkpoint
  - Multiple Workflow Management
- Future
  - Manage workflow to choose platform
  - Based on load and/or availability, hardware, or user-specified constraints

- BeeFlow: A Workflow Management System for In Situ Processing across HPC and Cloud Systems, ICDCS, 2018
- Build and execution environment (BEE): an encapsulated environment enabling HPC applications running everywhere, IEEE BigData, 2018
- BeeSwarm: Enabling Parallel Scaling Performance Measurement in Continuous Integration for HPC Applications, ASE, 2021
- BEE Orchestrator: Running Complex Scientific Workflows on Multiple Systems, HiPC, 2021