Rocoto History, Philosophy, and Future Plans

UFS Workflows Workshop
April 28-30 2020

Christopher Harrop, CIRES/NOAA GSL

Contents

History

How it all began … reliability of RUC forecast model runs was ~70-75%

	Forecast Lead Time													Cumulative Success Rate
Cycle	00	01	02	03	04	05	06	07	08	09	10	11	12	
2002-01-01 00Z														69%
2002-01-01 01Z														85%
2002-01-01 02Z														56%
2002-01-01 03Z														67%
2002-01-01 04Z														72%
2002-01-01 05Z														77%
2002-01-01 06Z														77%
2002-01-01 07Z														75%
2002-01-01 08Z														67%
2002-01-01 09Z														70%
2002-01-01 10Z														73%

History

Why such poor reliability?

- Run scripts assumed every command worked perfectly every time
  - No error checking
  - Cascading failures
  - No option to resubmit failed runs
- HPC system (Jet) had the reliability of … an HPC system
  - Batch system failures
  - File system failures
  - Network failures
- Model code contained bugs
  - Uninitialized variables
  - Index-out-of-bounds errors

History

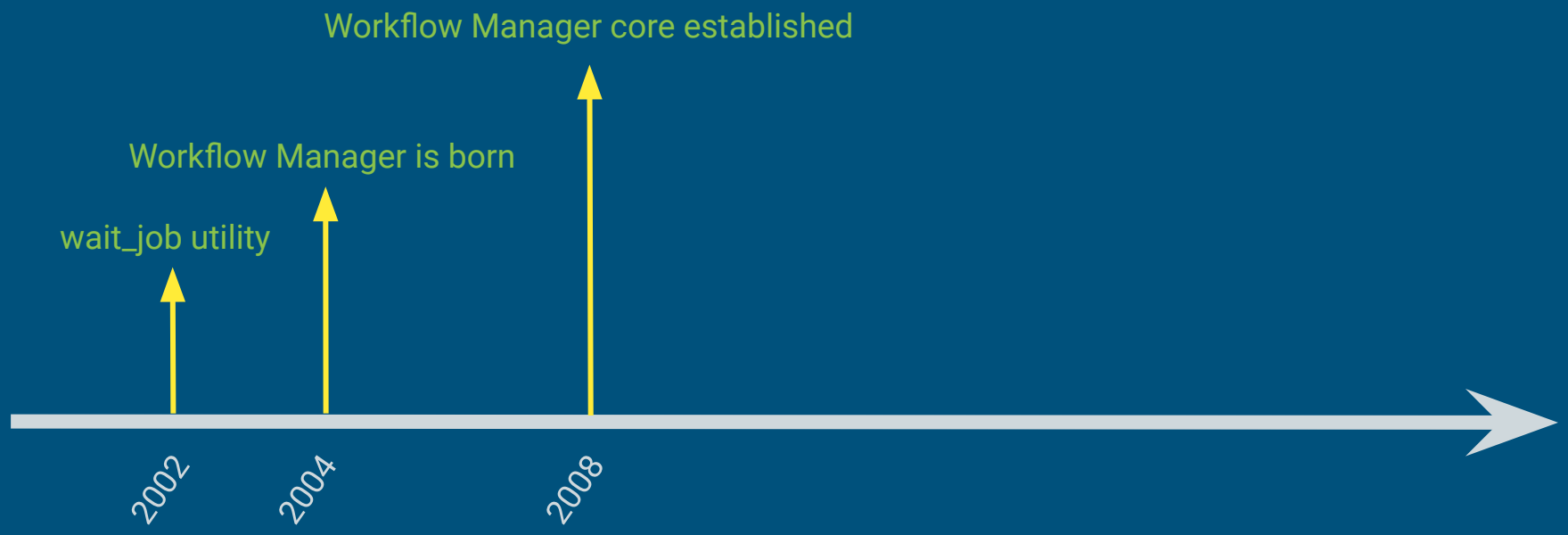
Why such poor reliability?

- Run scripts assumed every command worked perfectly all the time
  - No error checking
  - Cascading failures
  - No option to resubmit failed runs
- HPC system (Jet) had the reliability of an HPC system
  - Batch system failures
  - File system failures
  - Network failures
- Model code contained bugs
  - Uninitialized variables
  - Index-out-of-bounds errors

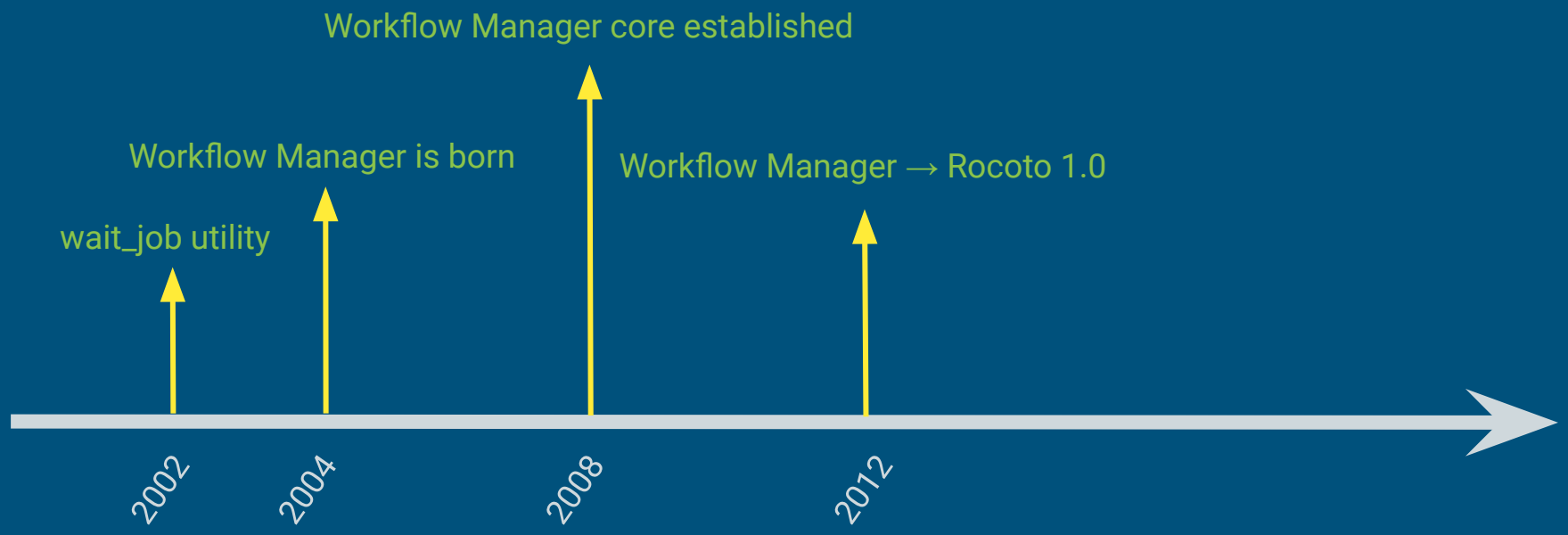
History

History

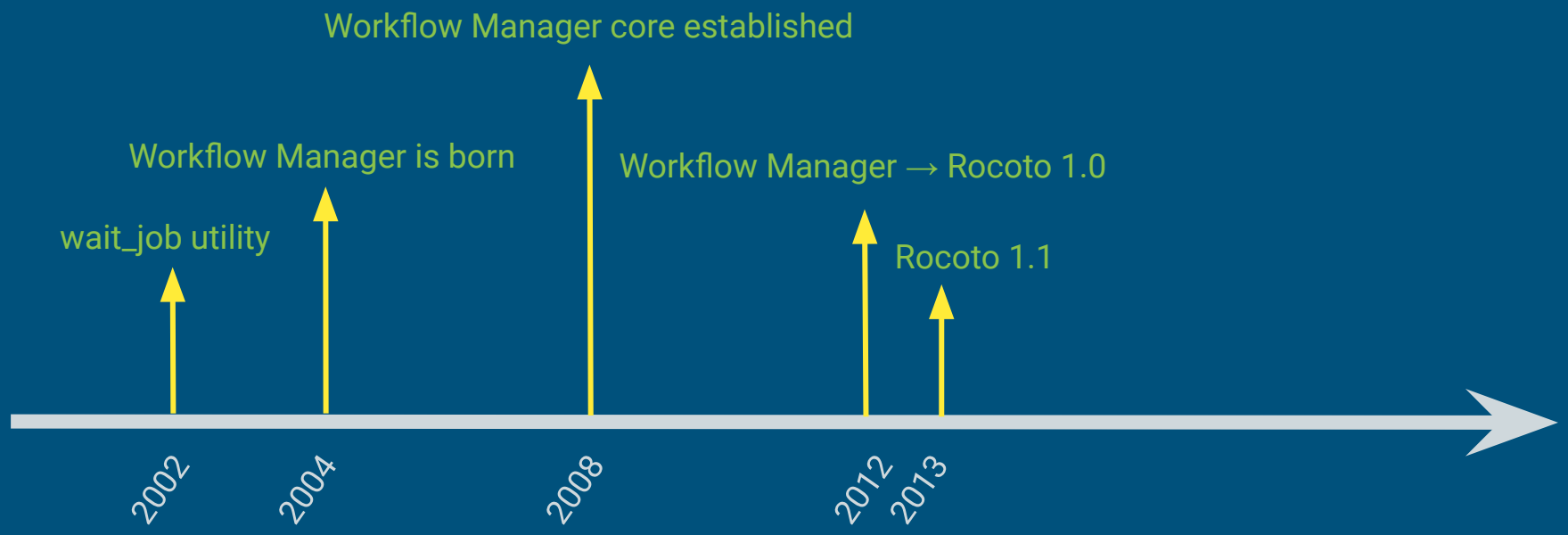
History


Workflow Manager core established
Workflow Manager is born
wait_job utility
2002
2004
2008

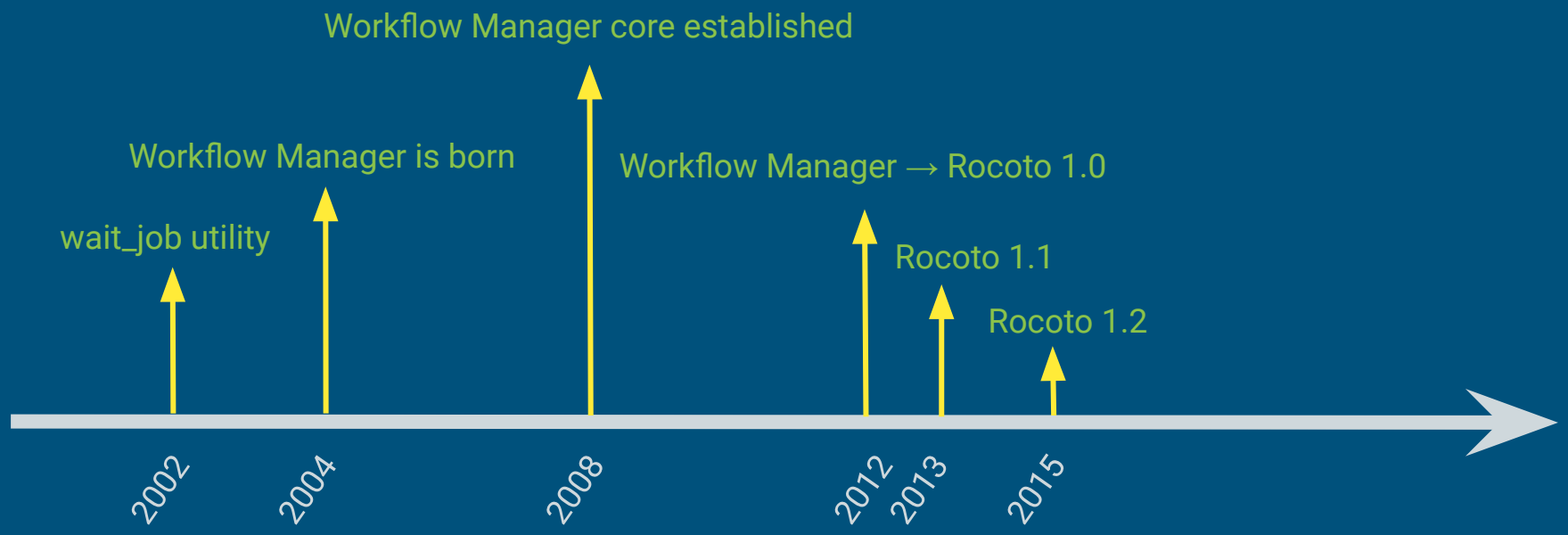
History


Workflow Manager core established
Workflow Manager is born
Workflow Manager → Rocoto 1.0
wait_job utility
2002
2004
2008
2012

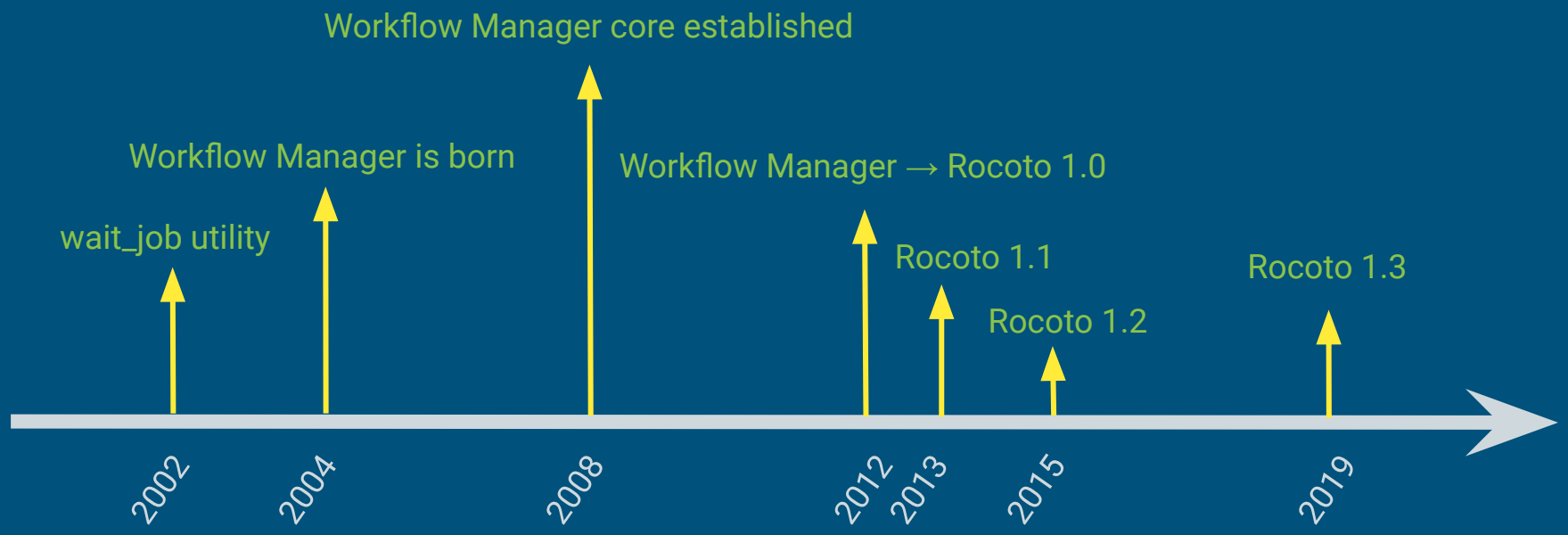
History


Workflow Manager core established
Workflow Manager is born
Workflow Manager → Rocoto 1.0
wait_job utility
Rocoto 1.1
2002
2004
2008
2012
2013

History


Workflow Manager core established
Workflow Manager is born
Workflow Manager → Rocoto 1.0
wait_job utility
Rocoto 1.1
Rocoto 1.2
2002
2004
2008
2012
2013
2015

History


Workflow Manager core established
Workflow Manager is born
Workflow Manager → Rocoto 1.0
wait_job utility
Rocoto 1.1
Rocoto 1.3
Rocoto 1.2
2002
2004
2008
2012
2013
2015
2019

History

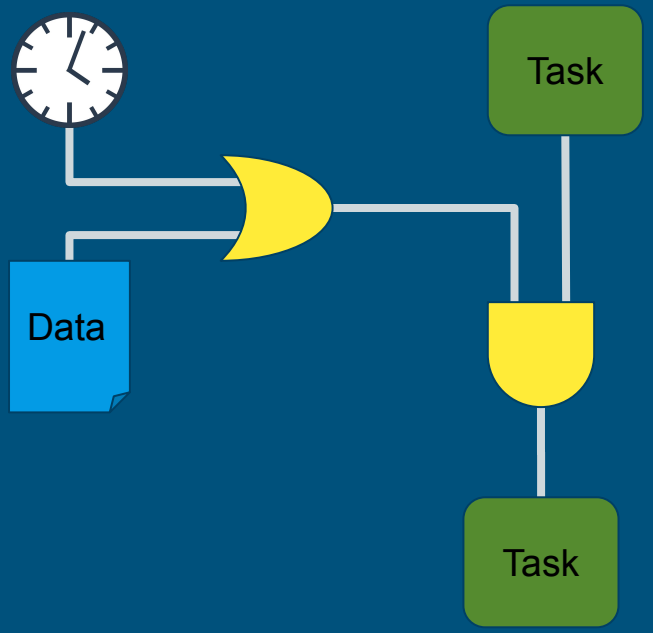
Workflow Automation Achievements

- RUC20 / RUC13 reliability mid 70s% → ~99% (early 2000's)
- Portable to any HPC system (mid 2000's)
- Support for SGE, LSF, Cobalt, MOAB, Torque, PBS, PBSPro, Slurm
- Adoption by FSL / GSD/ GSL modelers for WRF, HRRR, RAP, … (mid 2000's)
- Adoption by the DTC (Development Testbed Center) at NOAA/NCAR (mid 2000's)
- Adoption by EMC, starting with HWRF (early 2010's)

Philosophy

Workflow Automation Challenges

- Unreliable hardware and software
- Scale
  - 100s - 1000s of tasks per instance
  - 100s - 1000s of instances per experiment
- Complexity
  - Dependencies on time, data, execution order
- Realtime delivery constraints
- Retrospective resource management



Philosophy

Design Principles and Goals

Philosophy

Design Principles and Goals

- Installable & usable entirely in user space by end users

Philosophy

Design Principles and Goals

- Installable & usable entirely in user space by end users
- Domain-specific, but general purpose

Philosophy

Design Principles and Goals

- Installable & usable entirely in user space by end users
- Domain-specific, but general purpose
- No specific knowledge of models, codes, or HPC systems

Philosophy

Design Principles and Goals

- Installable & usable entirely in user space by end users
- Domain-specific, but general purpose
- No specific knowledge of models, codes, or HPC systems
- Every feature driven by demonstrable need and wide applicability

Philosophy

Design Principles and Goals

- Installable & usable entirely in user space by end users
- Domain-specific, but general purpose
- No specific knowledge of models, codes, or HPC systems
- Every feature driven by demonstrable need and wide applicability
- Portable across HPC systems

Philosophy

Design Principles and Goals

- Installable & usable entirely in user space by end users
- Domain-specific, but general purpose
- No specific knowledge of models, codes, or HPC systems
- Every feature driven by demonstrable need and wide applicability
- Portable across HPC systems
- Portable workflow definitions

Philosophy

Design Principles and Goals

- Installable & usable entirely in user space by end users
- Domain-specific, but general purpose
- No specific knowledge of models, codes, or HPC systems
- Every feature driven by demonstrable need and wide applicability
- Portable across HPC systems
- Portable workflow definitions
- Hands-off resiliency to system problems

Philosophy

Common Workflow Strategies

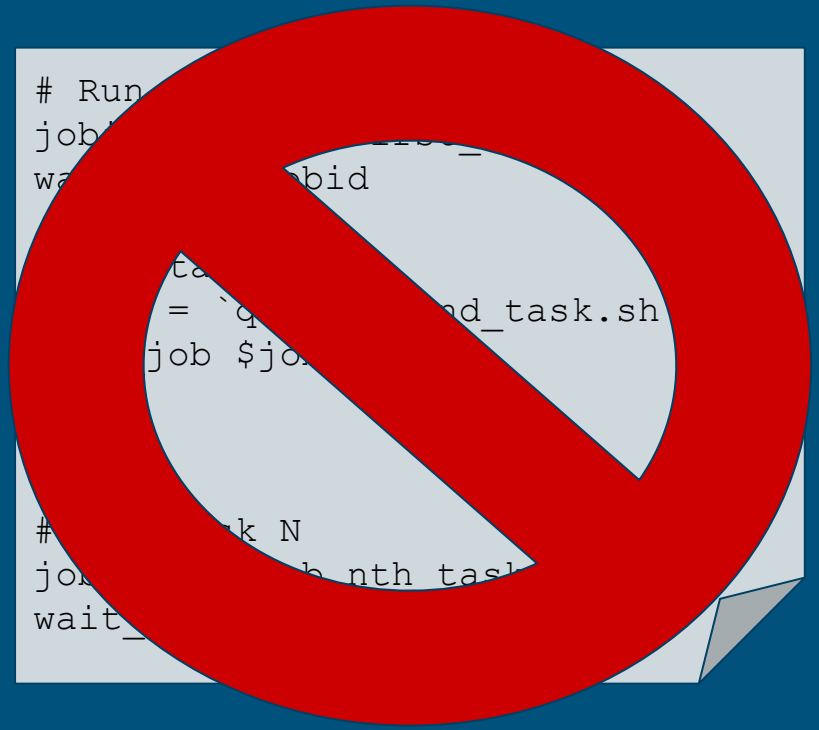
- Monolithic “run” script
- Workflow state embedded in script
- Relies on long up times
- Requires manual restart
- All or nothing reruns
- Fragile

```
# Run task 1
jobid = `qsub first_task.sh`
wait_job $jobid

# Run task 2
jobid = `qsub second_task.sh`
wait_job $jobid
.
.
.
# Run task N
jobid = `qsub nth_task.sh`
wait_job $jobid
```

Philosophy

Common Workflow Strategies

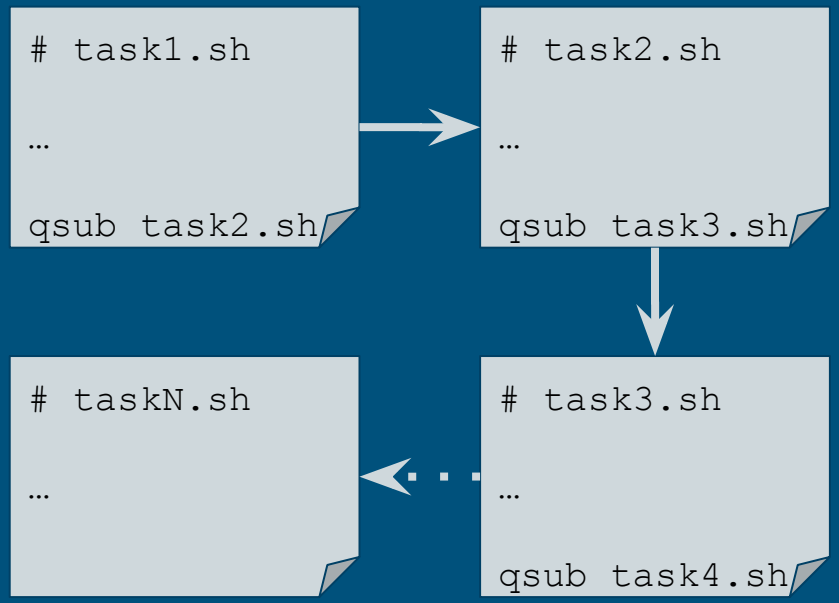
- Monolithic “run” script
- Workflow state embedded in script
- Relies on long up times
- Requires manual restart
- All or nothing reruns
- Fragile



Philosophy

Common Workflow Strategies

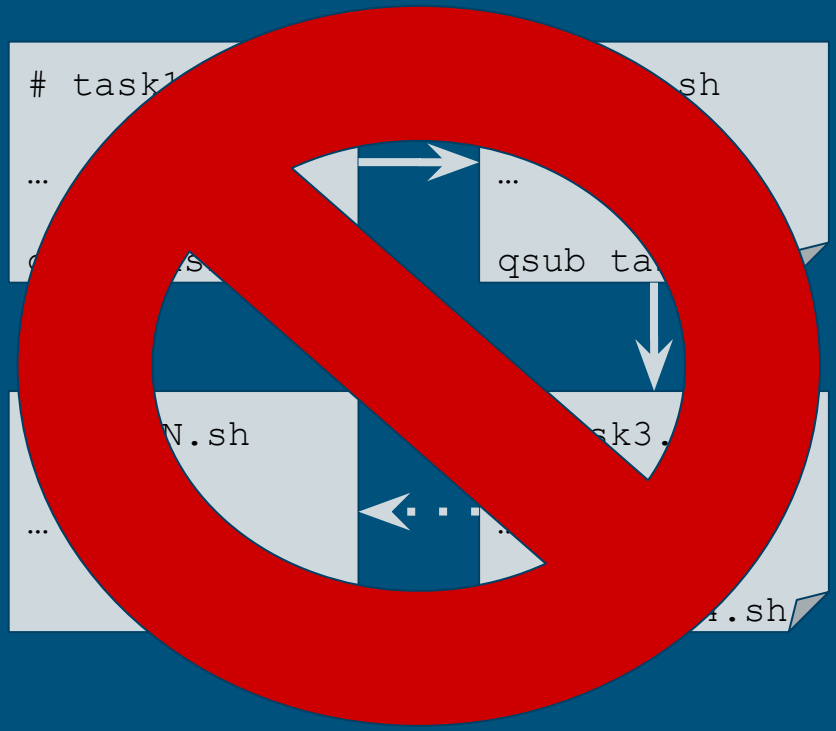
- Job chains
- Workflow state distributed in scripts
- Relies on a perfect batch system
- Failures are difficult to diagnose
- Requires manual restart
- Can restart in middle, but hard
- Fragile



Philosophy

Common Workflow Strategies

- Job chains
- Workflow state distributed in scripts
- Relies on a perfect batch system
- Failures are difficult to diagnose
- Requires manual restart
- Can restart in middle, but hard
- Fragile



Philosophy

Common Workflow Strategies

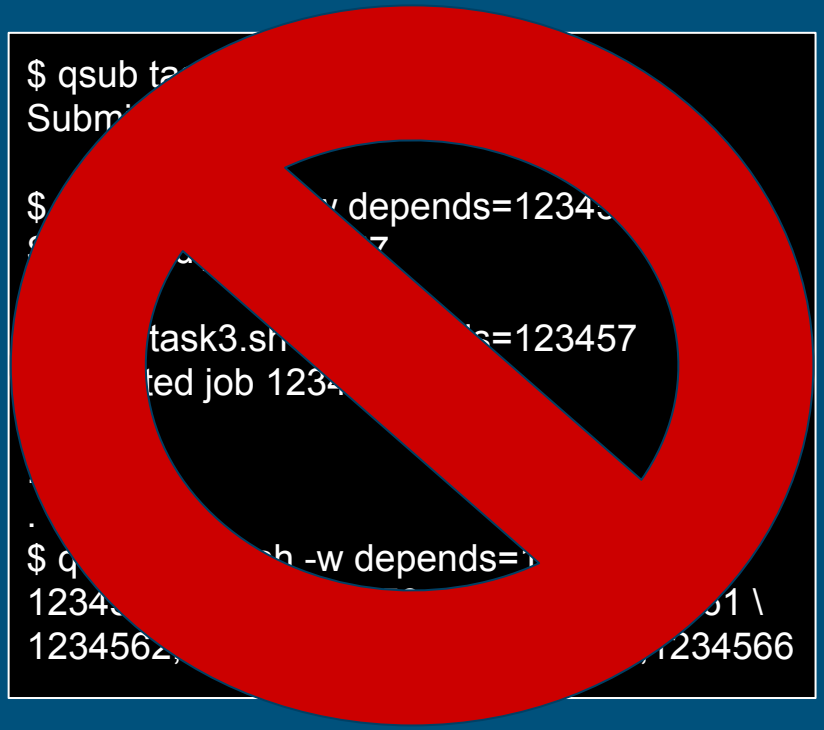
- Batch job dependency shotgun
- Workflow state tracked in bqs
- Relies on a perfect batch system
- Can result in complex train wrecks
- Requires manual restart
- Fragile

```
$ qsub task1.sh
Submitted job 123456

$ qsub task2.sh -W depends=123456
Submitted job 123457

$ qsub task3.sh -W depends=123457
Submitted job 123458
.
.
.
$ qsub taskN.sh -W depends=123456, \
123457,123458,123459,1234560,1234561, \
1234562,1234563,1234564,1234565,...
```

Philosophy

Common Workflow Strategies

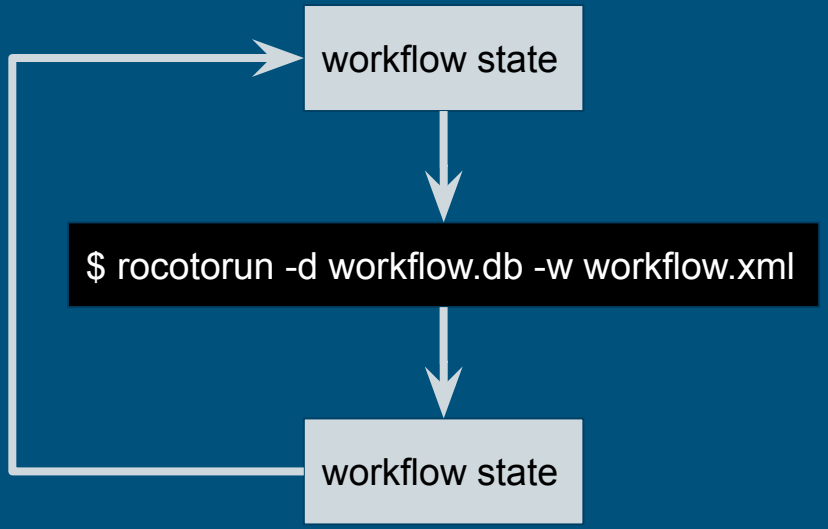
- Batch job dependency shotgun
- Workflow state tracked in bqs
- Relies on a perfect batch system
- Can result in complex train wrecks
- Requires manual restart
- Fragile



Philosophy

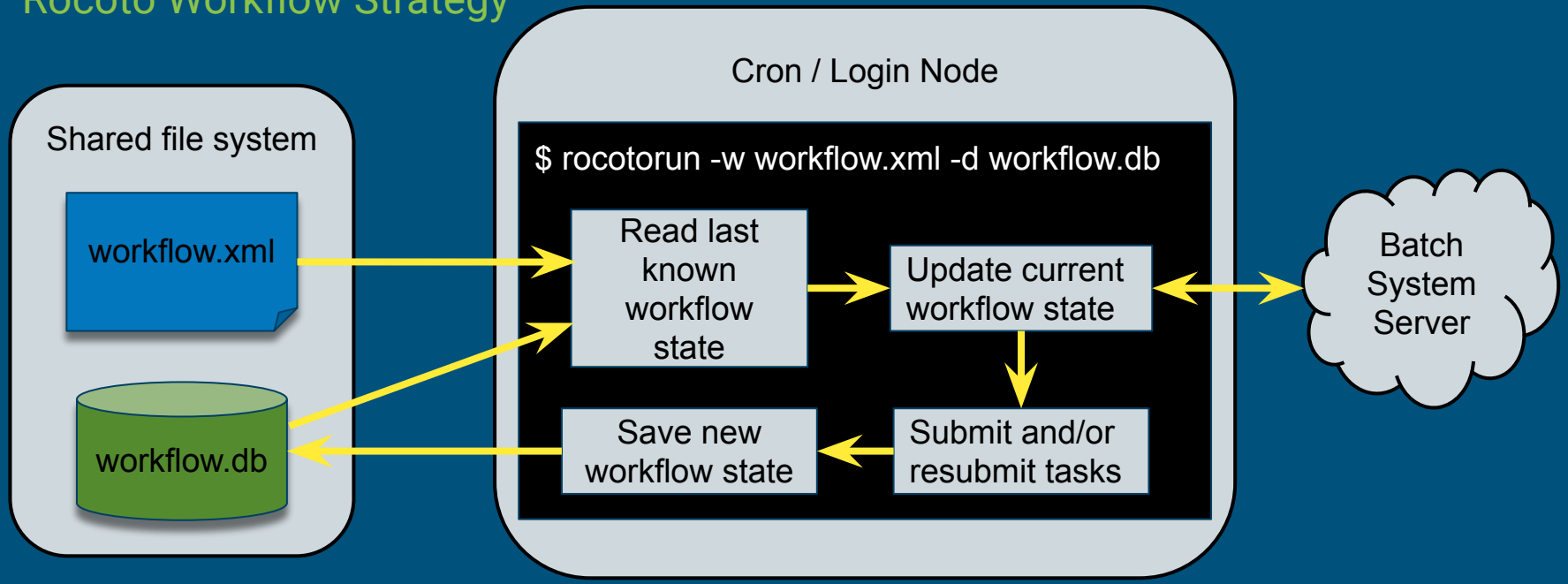
Rocoto Workflow Strategy

- Iterative submission & monitoring
- Workflow state in database
- Recovery from batch system outages
- Task-level failure diagnosis and restart
- Automated hands-off restart
- Resilient to system disruptions



Philosophy

Rocoto Workflow Strategy



Philosophy

Prerequisites for success

- Tasks correctly report their own success or failure
- Tasks do not start or monitor other tasks
- Tasks are idempotent (facilitates at least once execution semantics)
- Prefer small, composable, tasks
- Task granularity that balances flexibility and restart efficiency

Current Capabilities

Design Overview

- DAG (directed acyclic graph) workflow model
- XML workflow descriptions
- Support for all major batch systems
- Workflow instances specified as “cycles” (analysis times)
- Realtime & retrospective modes for “cycle” activation
- Task, Data, and Time dependencies (arbitrary boolean expressions)
- Generic specification of task properties

Current Capabilities

Commands Overview

- Advance workflow state: **rocotorun**
- Retrieve workflow status: **rocotostat**, **rocotocheck**
- Force workflow tasks to run: **rocotoboot**
- Undo completed workflow tasks: **rocotorewind** (from Sam Trahan)
- Force successful task completion: **rocotocomplete** (from Sam Trahan)
- Purge old database entries: **rocotovacuum**

Current Capabilities

Features Overview

- Throttling to manage resources
  - Cycles, cores, tasks (global and per task)
- Cycle life span for real-time mode
- Task retries to increase reliability
- Task deadlines to manage real-time constraints
- Hang dependencies for detection of hung tasks
- Task environment specification
- Customizable membership of tasks to workflow “cycles”

Future Plans

Known Gaps

- XML workflow specification is tedious and error prone
- Lack of support for non-HPC machines without a batch system
- DAG model not expressive enough
  - Evaluation of runtime conditionals is difficult, loops are impossible
  - Artifacts discovered at runtime can't be used in dependencies
- Static workflow definitions
- Long queue wait times due to submission when dependencies are satisfied
- Awkward override control of sub-workflows
- Errors are difficult for end-users to interpret

Future Plans

Planned Enhancements / Refactor

- Improved software development process
- Add Domain-Specific Language (DSL) to generate XML
- Replace DAG model with High-Level Petri Net (HLPN) model
  - Turing complete, allows “if” and “while” constructs
- Add support for laptops and workstations
- Add support for cloud-based workflows

Future Plans

Planned Enhancements / Refactor (cont)

- Explore resource pools to minimize workflow makespans
- Explore cross-site workflow management
- Arbitrary control (play, pause, resume, cancel) of sub-workflows
- Improved mechanism for users to run tasks directly outside workflow system
- Provide hooks for GUI development

Future Plans

Development and Support

- No funding currently dedicated to Rocoto development and support
- Management fully committed to ongoing development and support
- Current development and support team is: me
  - Support has been very easy to manage so far
- Previous contributions and EMC support from Sam Trahan
- Collaborators are welcome

Questions?