# Using Pilot Systems to Execute Many Task Workloads on Supercomputers

Andre Merzky, Matteo Turilli, Mark Santcroos, Manuel Maldonado, Shantenu Jha

Brookhaven National Laboratory, Rutgers University

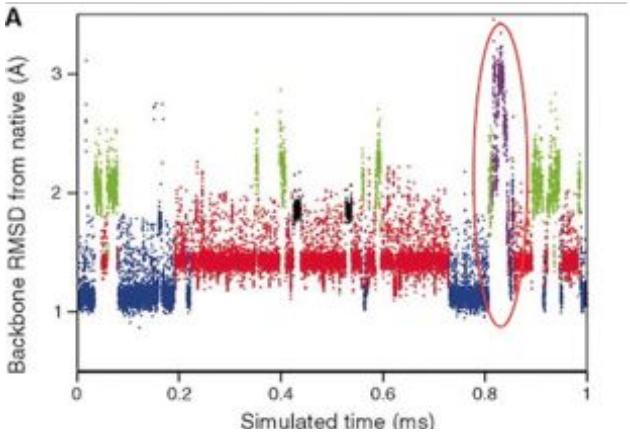
6th Annual MUG Meeting, Columbus, OH

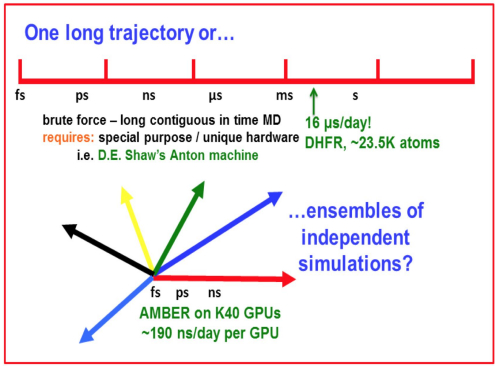
# Outline

- Why Execute Many Task Workloads?
  - Importance of applications with “more than a single task” (workflows)
  - Workflows aren’t what they used to be!
- Building Blocks approach to Workflow Middleware
  - What is a Pilot System?
  - Pilot Abstraction (P* Model)
- RADICAL-Pilot: A Pilot-System for HPC Workflows
  - Programming and Execution Model
  - Implementation on Cray systems
  - Performance characterisation

# Biomolecular (MD) Simulations: Context

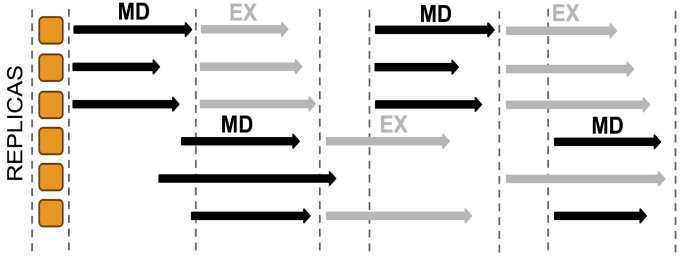
- Larger biological systems
  - Requires **weak** scaling
- Long time scale problem
  - Requires **strong** scaling
  - DE Shaw special purpose computer (Anton)
- Gap between weak and strong scaling will grow
  - Scaling challenges > than either single-partition strong and weak scaling.
  - Ensemble simulations
- **Ensemble-based Adaptive Algorithms:**
  - Intermediate data used to determine next stages
  - Improved simulation efficiency (MSM: $10^3$)

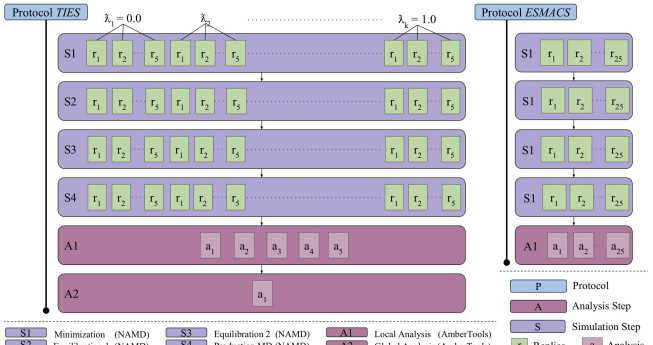


# Biomolecular Adaptive Algorithms

- Many biomolecular **sampling algorithms** formulated as adaptive algorithms/methods:
  - Replica-exchange
  - Expanded Ensemble
  - …
- Types of Adaptivity:
  - Task parameter(s), order, …
  - Task count, iteration count, ….
- **Adaptive logic separate from the MD code**
  - Each task is an independent simulation
  - Task often interact (not a “bag-of-tasks”); degrees and levels of coupling

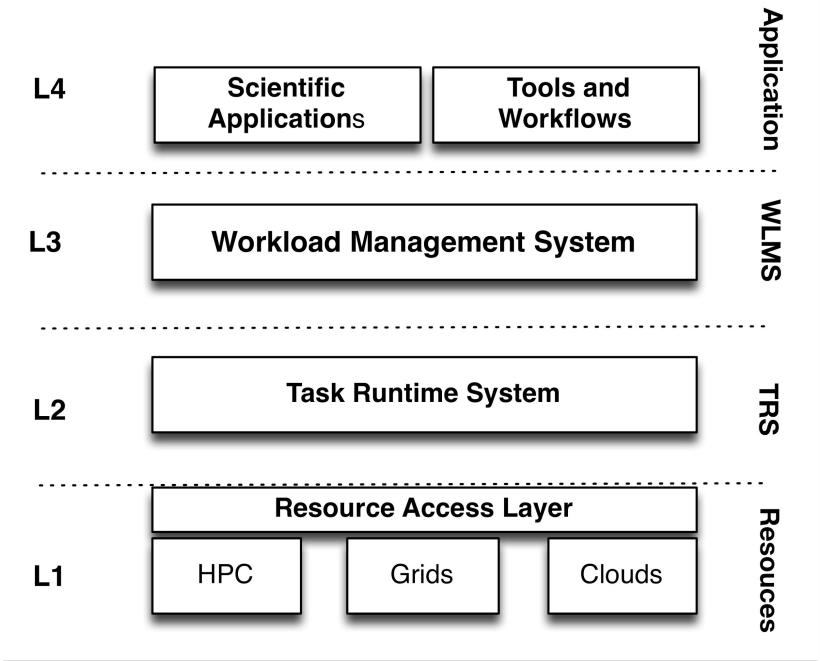


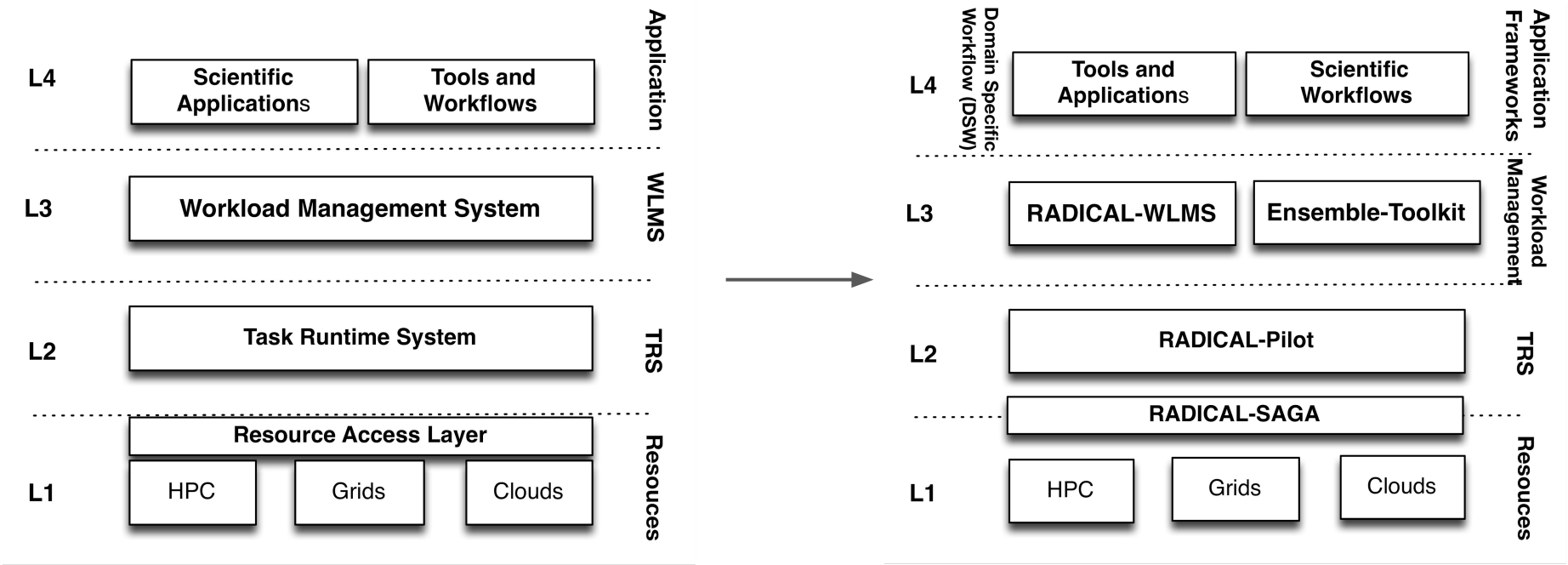
# Why a “Fresh Perspective” to Workflows?

- Initially “Monolithic” Workflow systems with “end-to-end” capabilities
  - Workflow systems were developed to support “big science” projects.
  - Software infrastructure was “fragile”, unreliable, missing services
- Workflows aren’t what they used to be!
  - More pervasive, sophisticated but no longer confined to “big science”
  - Prevent vendor lock-in
  - **Extend traditional focus from end-users to workflow tool developers!**
- Building Blocks (BB)  permit workflow tools and applications can be built
  - Diverse “design points”; unlikely “one size fits all”
  - Last mile distinction → proliferation of workflow systems vs single system

# Developing Workflow Tools Using Building Blocks



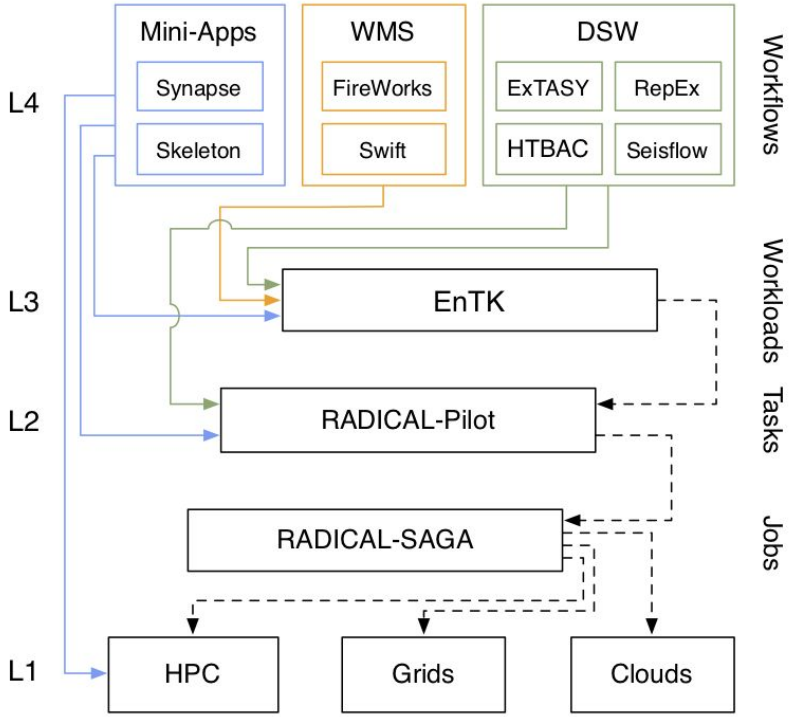
# RADICAL-Cybertools: Building Blocks for Workflows




- BB to support workflows, and the development of workflow tools
- A “laboratory” for testing ideas, support production science
- Stand alone, as well as vertical integration and horizontal extensibility

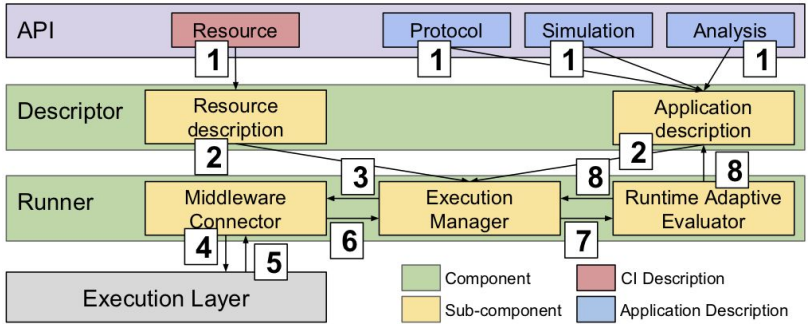
# RADICAL-Cybertools: Building Blocks for Workflows

- A “laboratory” while supporting production grade workflows **and** workflow tools.
  - Consistent with HPC & scale
- **Integrate with existing tools:**
  - Swift, Fireworks, PanDA, Binding Affinity Calculator (BAC)
  - Distinct points of integration, vertical integration and horizontal extensibility
  - Need “faster” start, “scalable” (more tasks) and “better” (resource utilization)
- **Novel tools and libraries:**
  - ExTASY, RepEx, **HTBAC**, Seisflow,..



# HTBAC: High-throughput Binding Affinity Calculator

- Python library for defining and executing ensemble-based biosimulation protocols
  - Protocols expressed and implemented using HTBAC's API
  - HTBAC utilizes RADICAL-Cybertools: **EnTK and RP**
- Implemented ESMACS and TIES protocols at scale
- Define additional adaptivity parameters that are passed down to the underlying runtime system.

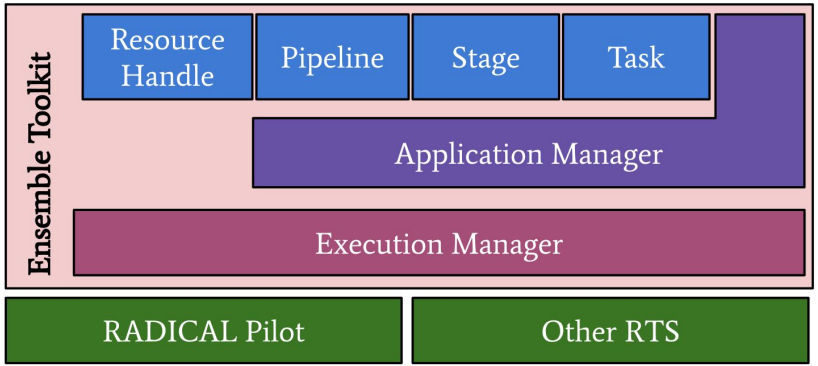



(1) **TIES** (alchemical **protocol**) employs enhanced sampling at each lambda window to yield reproducible, accurate and precise relative binding affinities.

(2) **ESMACS** (endpoint **protocol**) is a computationally cheaper, but less rigorous method, it is used to directly compute the binding strength of a drug to the target protein from MD simulations (as opposed to differences in affinity).

# RADICAL-EnTK: Building Blocks for Workflows

- **Ensemble Toolkit (EnTK):** Toolkit to manage complexity of resource acquisition and task execution for ensemble based applications.
- **Design:**
  - User facing components (blue)
  - Workflow management components (purple) to manage the execution order of the individual tasks of the application
  - Workload management components (red) to manage resources and task execution via a runtime system (green)
- **Integrated with other tools:**
  - HTBAC, Replica-Exchange, ...



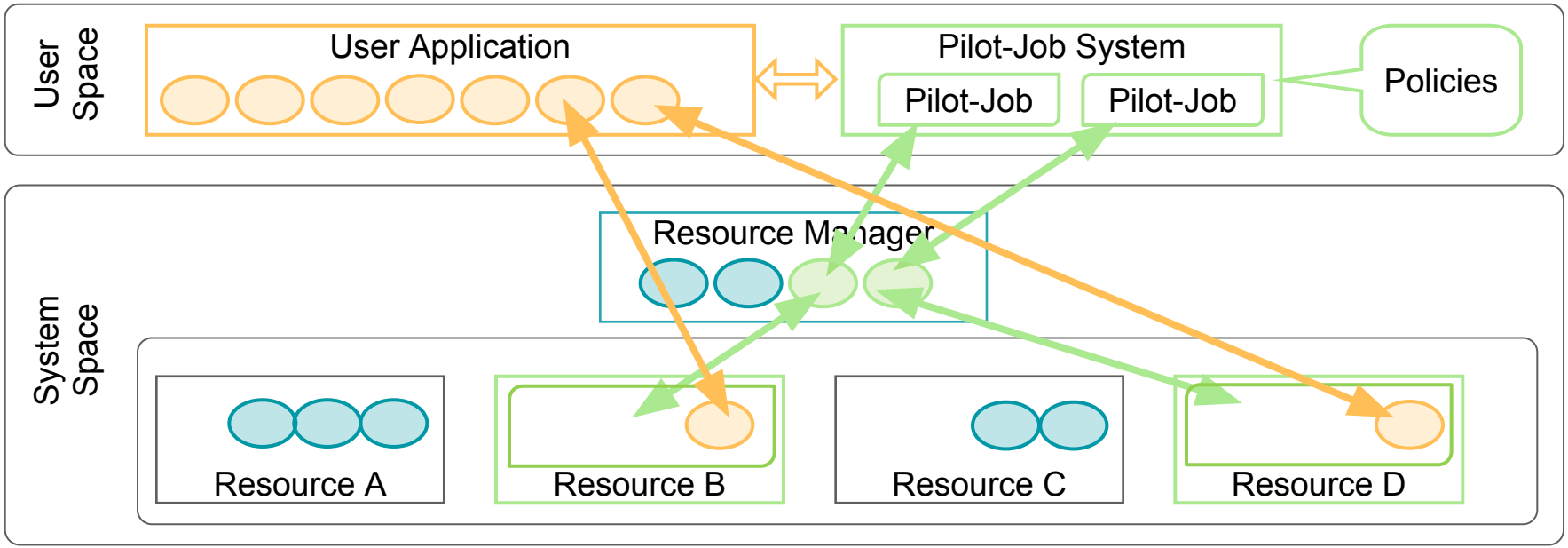

```
# Add post-exec to the Stage
p = Pipeline()
s1 = Stage()
...     # Define tasks and add them
...     # to Stage 's1'
s1.post_exec = {
                    # 'func_condition' operates on
                    # intermediate results to produce a
                    # boolean output
                    'condition': func_condition,

                    # 'func_on_true' is executed when the
                    # condition is True, e.g., adds another
                    # Stage 's2' to Pipeline 'p'
                    'on_true': func_on_true,

                    # 'func_on_false' is executed when the
                    # condition is False, e.g., returns
                    # without any adaptation
                    'on_false': func_on_false
                }    # Add Stage 's1' to Pipeline 'p'
```

Listing 1: Example of the enhanced EnTK API with post-execution properties to describe the adaptation to be performed after the execution of a Stage

# Pilot Abstraction: Schematic

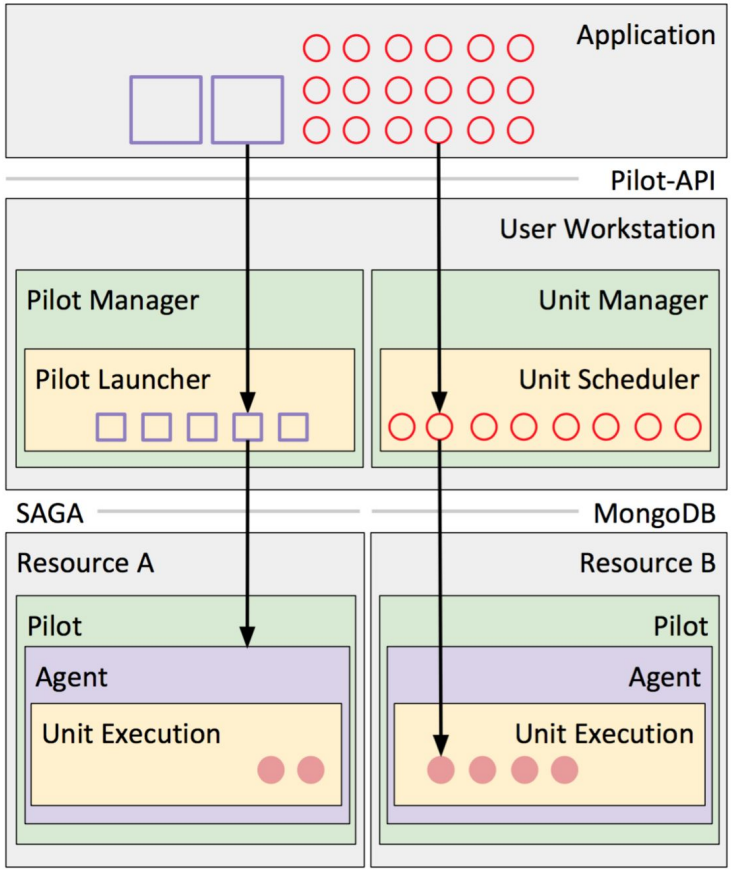
A system that generalizes a placeholder job to allow application-level control of acquired resources via a scheduling overlay.

- Decouples workload from resource management
- Enables the fine-grained “slicing and dicing” of resources
- Build higher-level frameworks without explicit resource management.



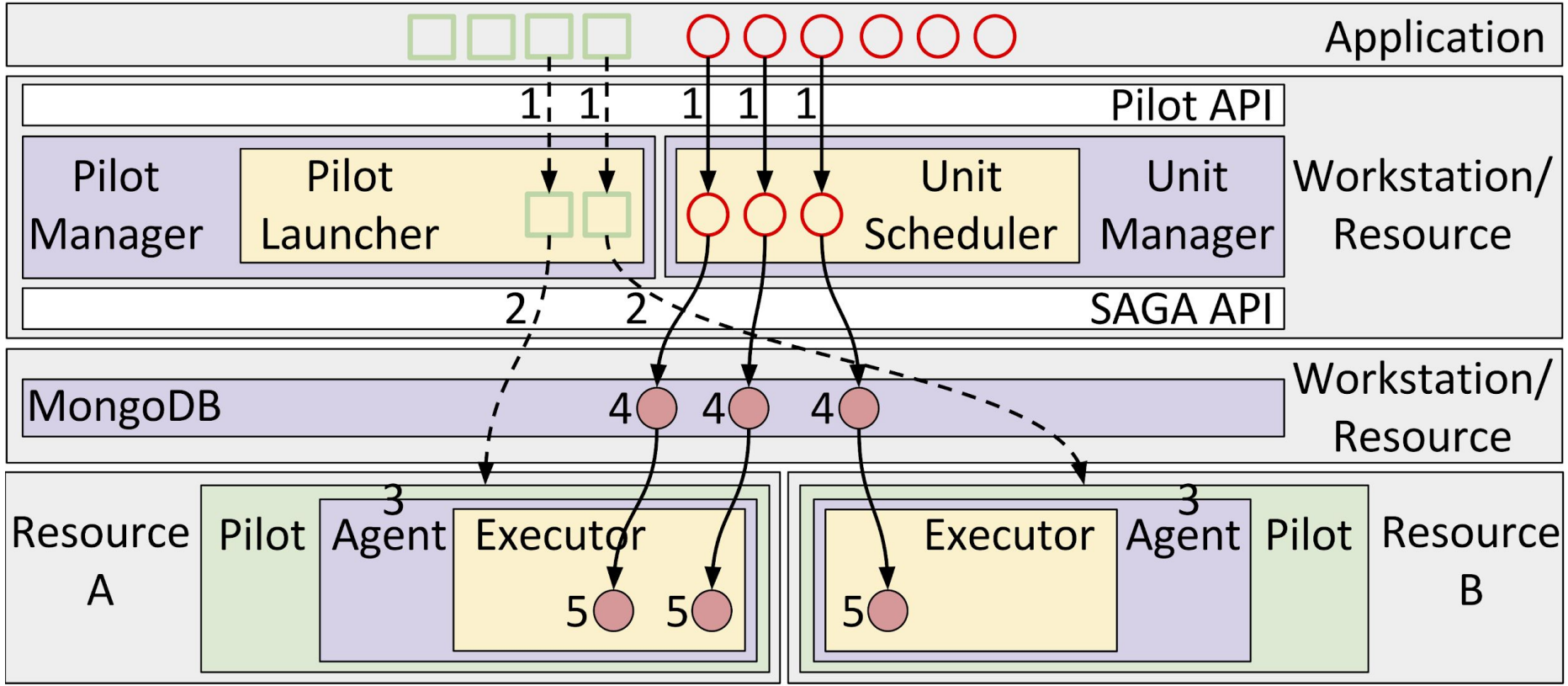
# RADICAL-Pilot: Implementation of Pilot-Abstraction

- *".. a scheduling overlay which generalizes the recurring concept of utilizing a placeholder as a container for compute tasks"*
- Decouples workload from resource management
- Enables the fine-grained spatio-temporal control of resources
- Build higher-level frameworks without explicit resource management
- Provides building block for late-binding of workloads on HPC

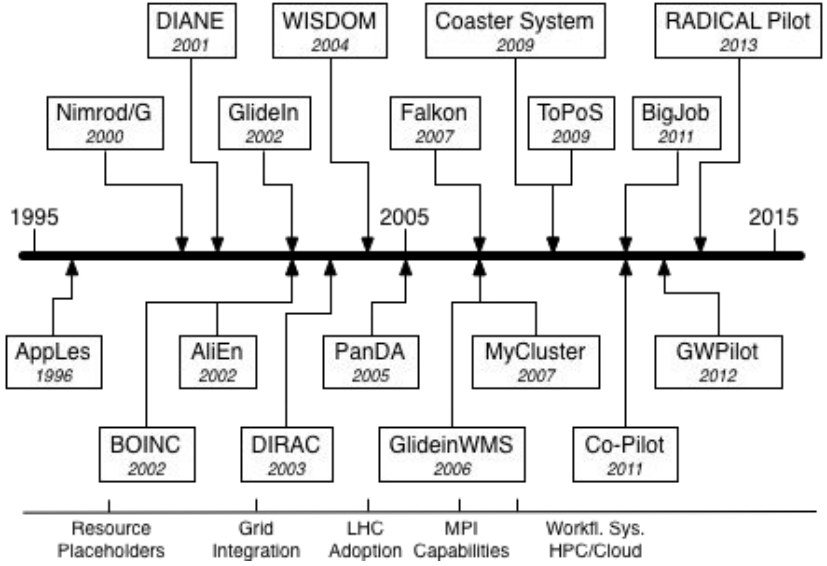


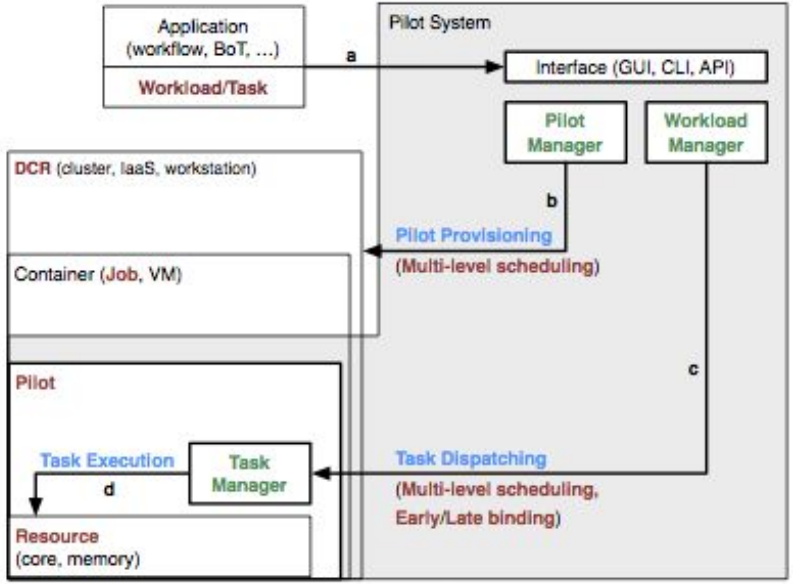

*Comprehensive Perspective on Pilot-Job Systems, ACM Computing Surveys (2018)*

# RADICAL-Pilot: Execution Model



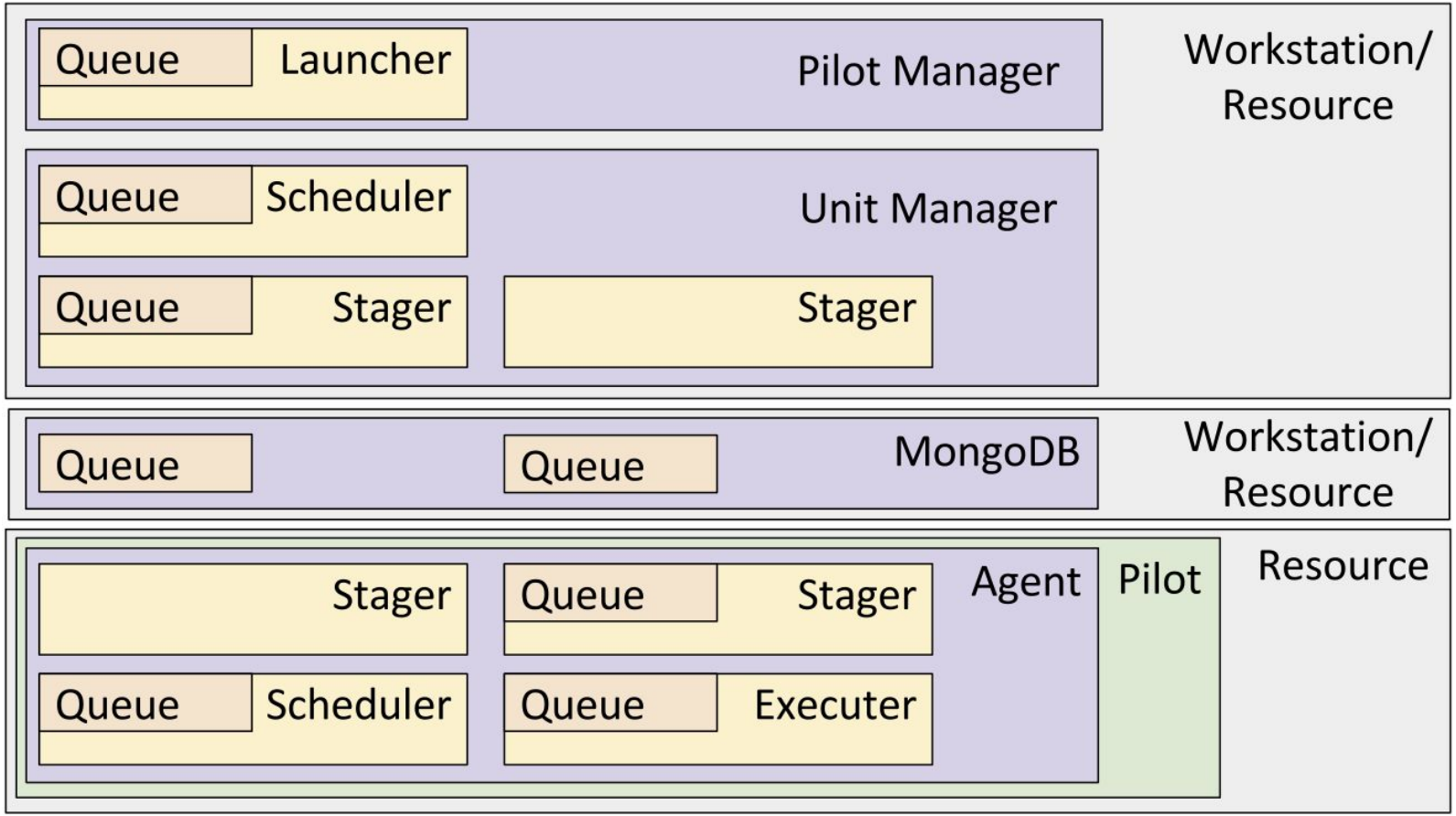
# Pilot Jobs: Many Variations on a Theme



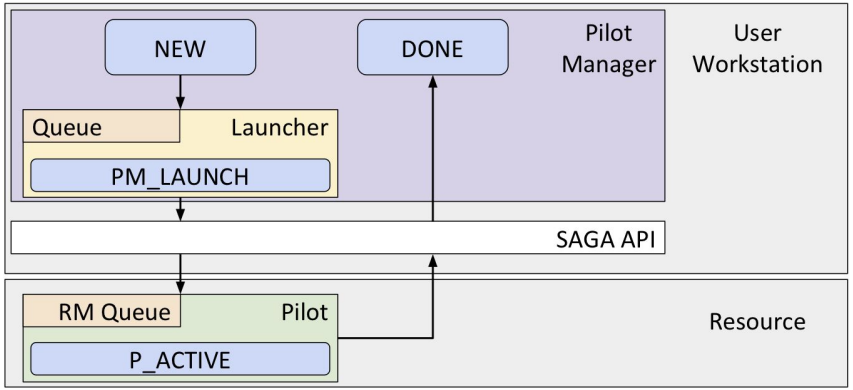



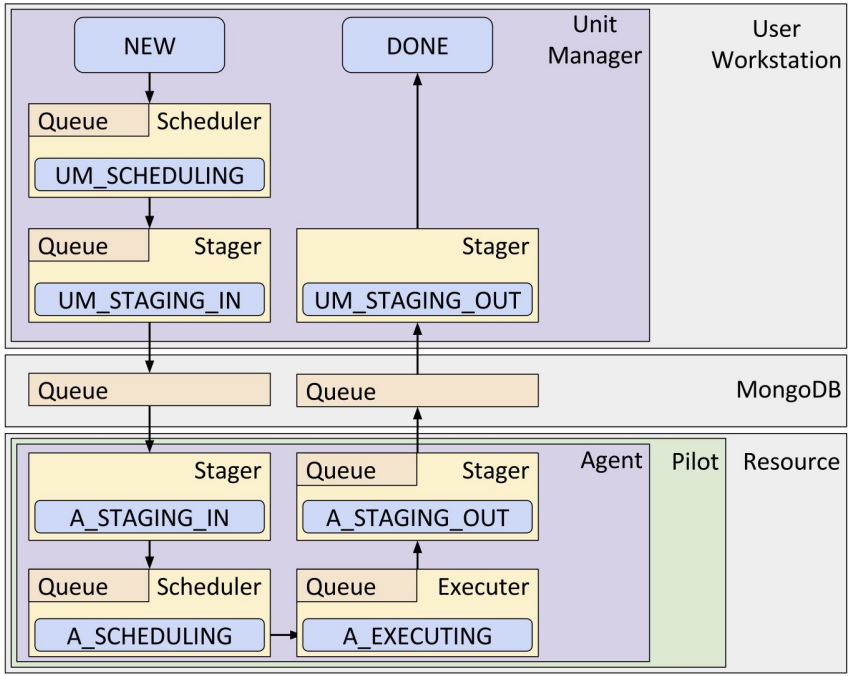

- "P*: A Model of Pilot-Abstractions", *8th IEEE International Conference on e-Science (2012)*
- *A Comprehensive Perspective on Pilot-Jobs* http://arxiv.org/abs/1508.04180 *(ACM Computing Surveys, 2018)*

# RADICAL-Pilot: Overall Architecture



# RADICAL-Pilot: State Model




- Pilot State Model:
  - 4 states, over client & server
- Unit State Model
  - 9 states, spread over 3 components

# RADICAL-Pilot: Programming Model

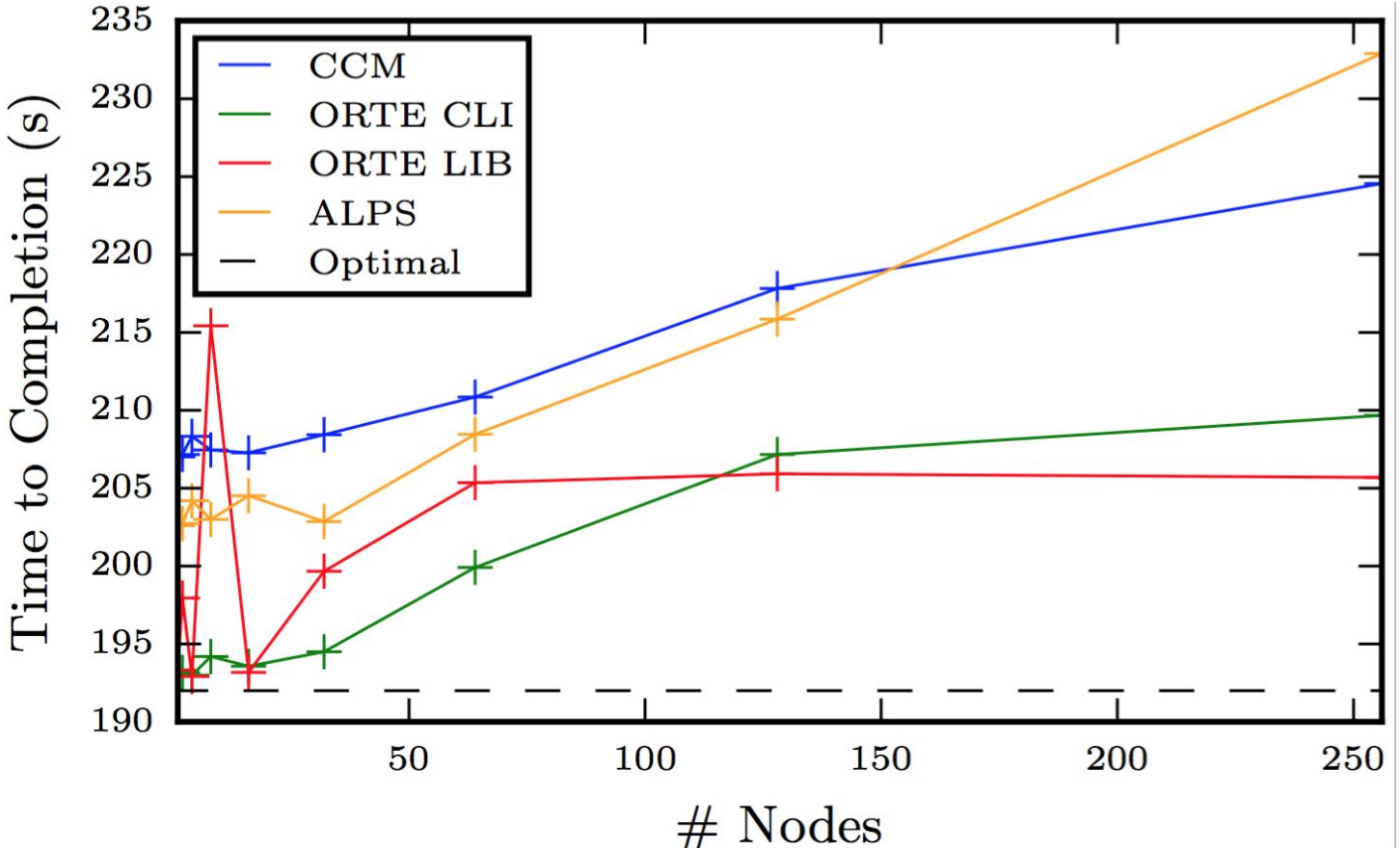
```
# Declare a 64-core pilot that will
# be available for 10 minutes.
pdesc = rp.ComputePilotDescription({
        'resource' : ncsa.bw,
        'cores'    : 64,
        'runtime'  : 10,
        'queue'    : 'debug',
        'project'  : 'gkd'
    })

# Submit the pilot for launching.
pilot = pmgr.submit_pilots(pdesc)

# Make the pilot resources available
# to a unit manager.
umgr.add_pilots(pilot)
```

```
# Number of units to run.
cuds = []
for i in range(0,42):
    # create a new CU description,
    # and fill it.
    cud = rp.ComputeUnitDescription()
    cud.executable = '/bin/date'
    cuds.append(cud)

# Submit units.
umgr.submit_units(cuds)

# Wait for the completion of units.
umgr.wait_units()

# Tear down pilots and managers.
session.close()
```

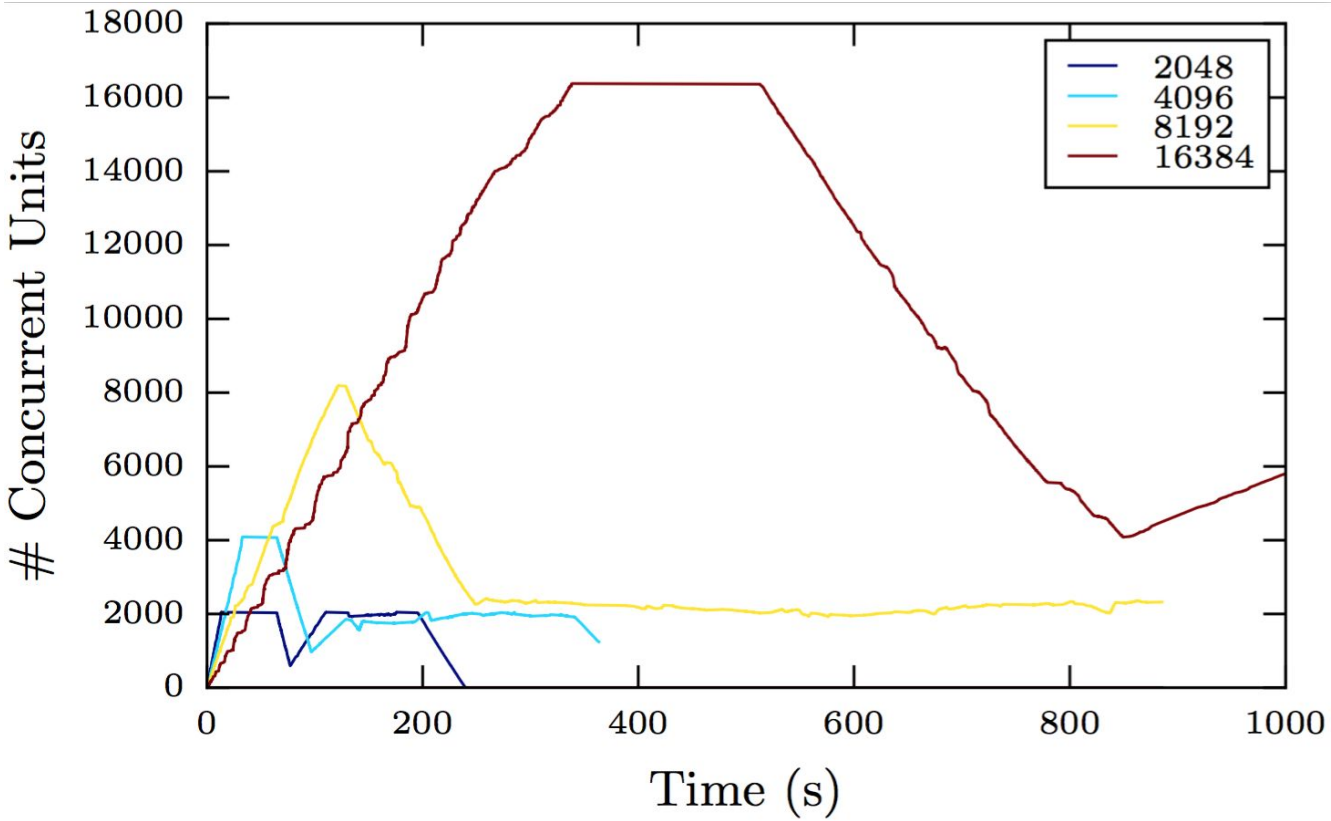
# Execution: (Why not) RADICAL-Pilot + APRUN

- RP Agent runs on MOM node
- Uses aprun to launch tasks onto the worker nodes
- Low throughput (ALPS not designed for short/small tasks)
- Limit on total concurrency (1000 aprun instances)
  - Less than 1000 on other CRAYS
- Maximum of one task per node

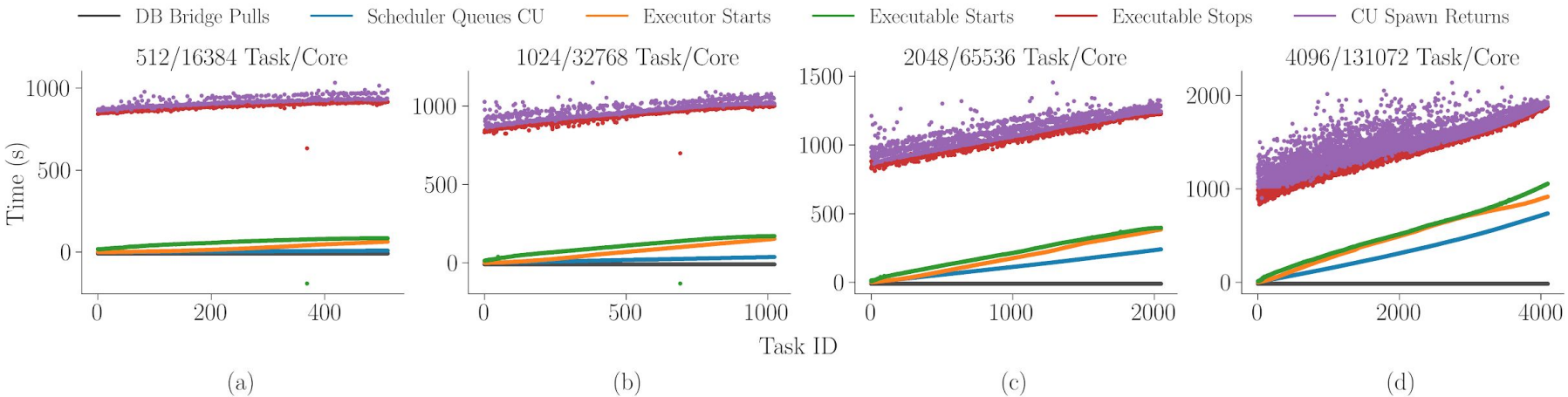
# Agent Performance: Full Node Tasks (3 x 64s)



# Agent Performance: Concurrent Units (3x)

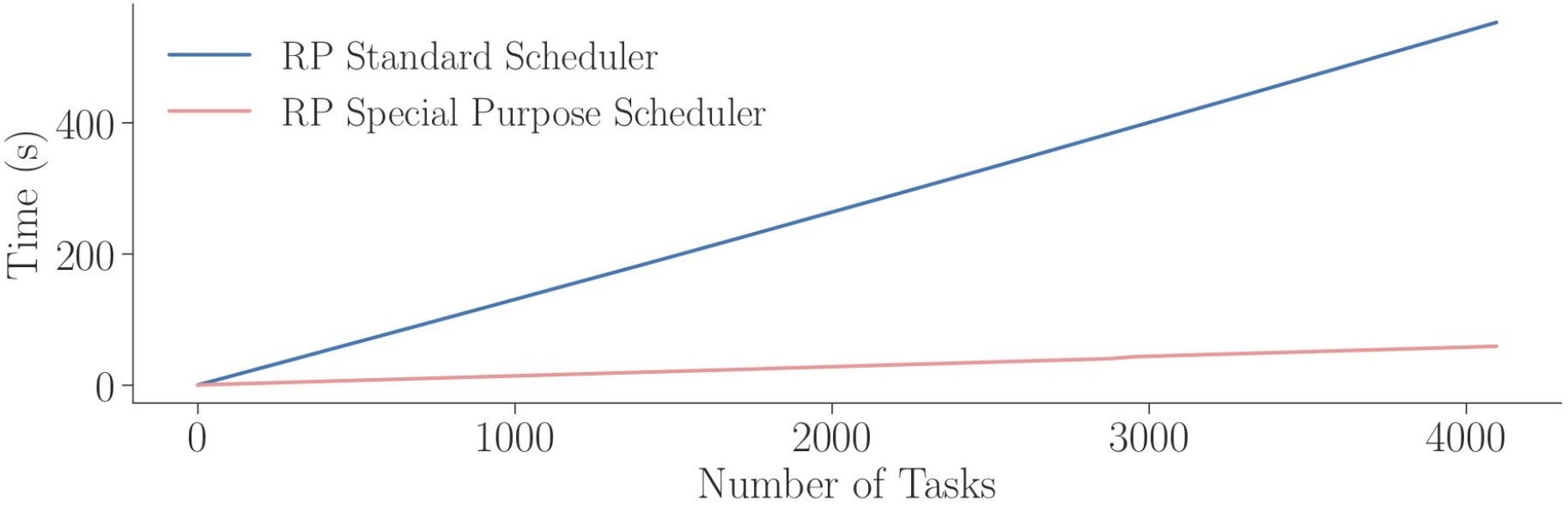


# RADICAL-Pilot: Weak Scaling Performance (Titan)



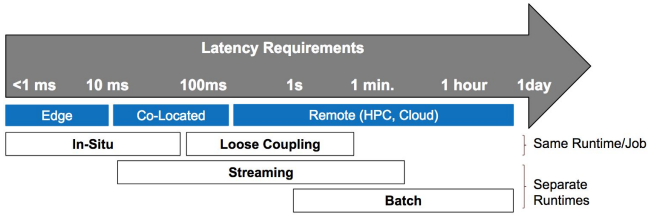
# RADICAL-Pilot: Resource Utilization Performance (Titan)

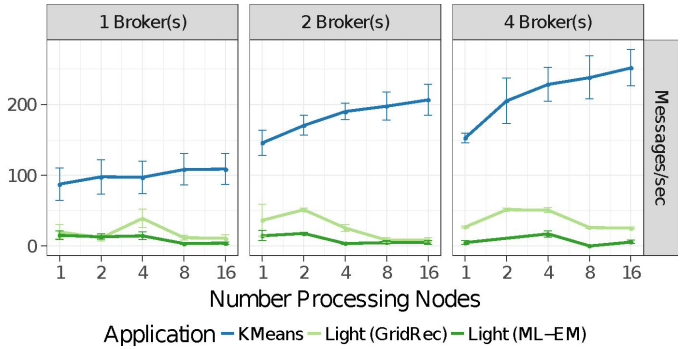
# RADICAL-Pilot: Price of Heterogeneity



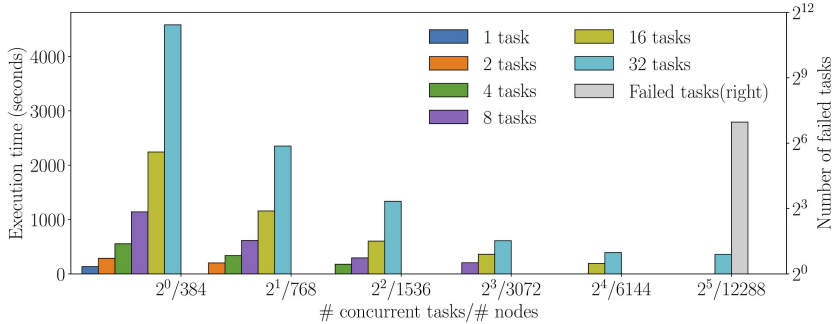
# RCT BB: From Streaming to Seismic Data

- Design HPC stream processing systems
  - Resource contention limits scalability of reconstruction algorithms
  - *Pilot-Streaming: Streaming Processing for HPC https://arxiv.org/pdf/1801.08648.pdf*

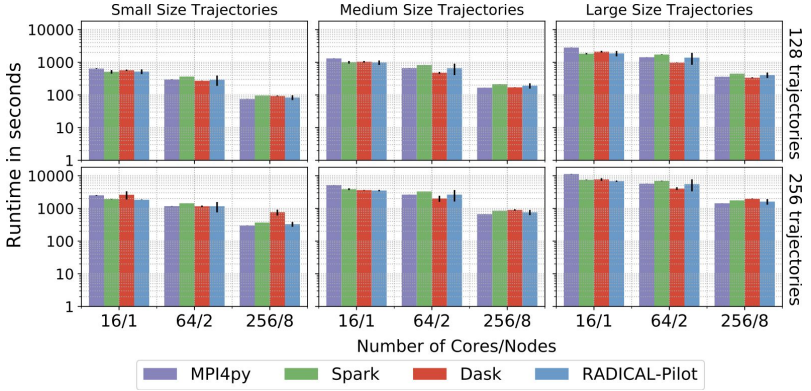



- Supporting Seismic Physics Workflows




- Task Parallel Analysis for Trajectory Data



# Summary

- **Importance and diversity of “workflows” set to increase**
  - Proliferation of middleware systems for “workflows” unsustainable
  - Substitute discussions of software with **abstractions & execution models**
- **Building blocks approach to workflows**
  - Focussed, principled design and development of middleware systems
  - Each building block has well defined performance characterization
- **RADICAL-Pilot: Implementation of Pilot abstraction**
  - Engineered to support heterogeneous workloads and resources
  - Investigated implementation and performance on Cray (Titan and Blue Waters)
- **Results:**
  - (i) ~O(10K) MPI simulations and tasks; (ii) Price for heterogeneity (generality); (iii) Ready for scheduling optimization; (iv) O(100K): When we overcome the scheduling challenges, message subsystem will hit!

# Thank You!