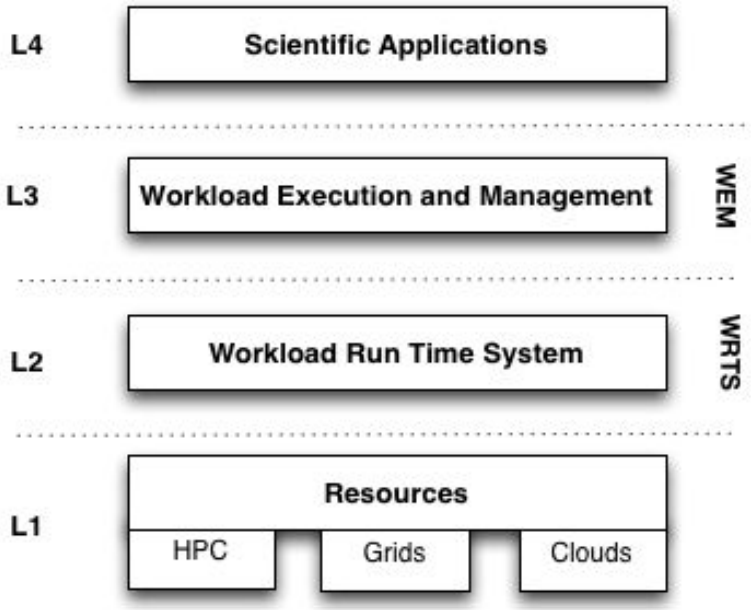
PILOT-JOBS: THEORY and PRACTICE

Shantenu Jha
http://radical.rutgers.edu

Layered View of Distributed Cyberinfrastructure

- L4: Application
  - Development of distributed applications and scientific workflows.
- L3: Workload Exec. & Management (**WEM**)
  - Task distribution and resource selection
- L2: Workload Run-time System (**WRTS**)
  - Managing tasks on a resource.
- L1: Resource Layer.
  - HPC, Grids, Clouds or specialized (data) infrastructure.

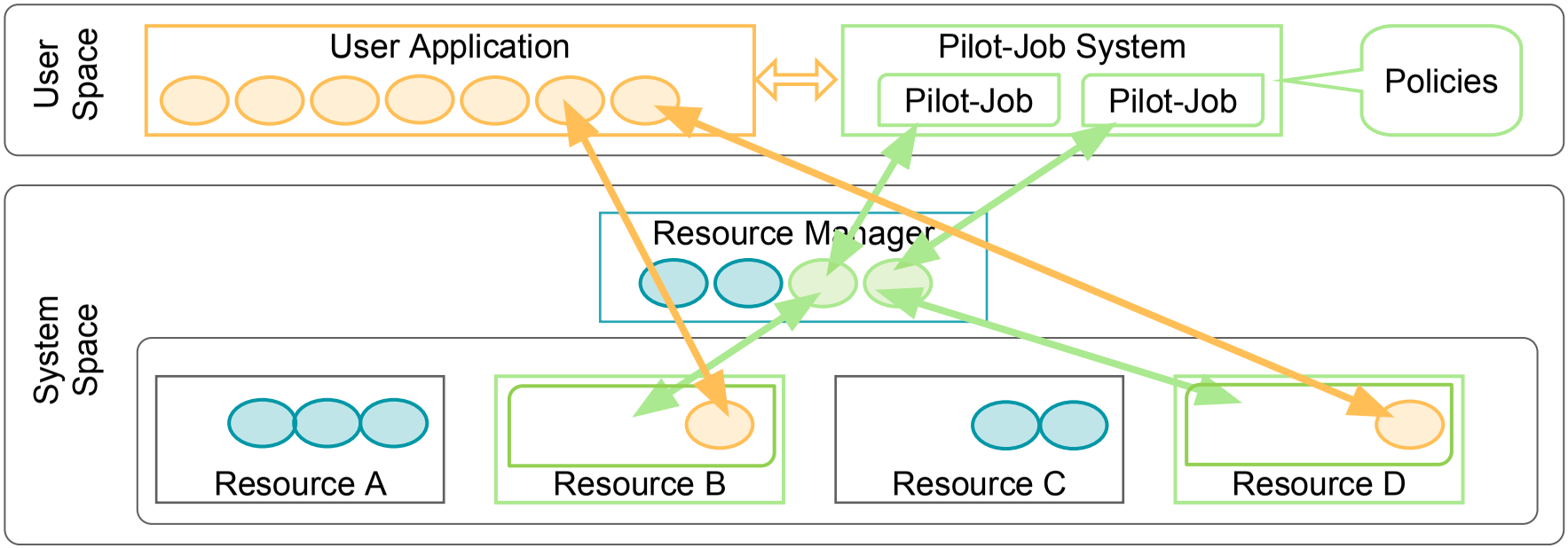


Software and Tools: Challenges

- How to find balance between requirements and distinct solutions?
  - Build upon commonality, yet provide for distinctiveness
- How to balance “specific” “general-purpose” and “functionally extensible”?
  - Reduce software development while meeting performance requirements
- How to balance research ideas with production demands?
  - Software building blocks
- **Can tools adhere to “conceptual models” ?**
  - **Tools and services should identify “what” they are implementing.**

Pilot Abstraction: Schematic

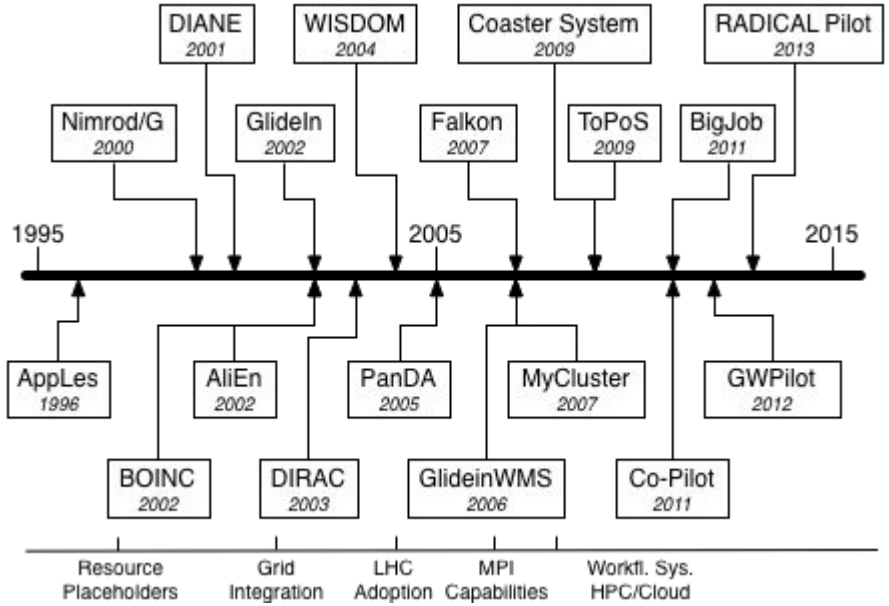
A system that generalizes a placeholder job to allow application-level control of acquired resources via a scheduling overlay.

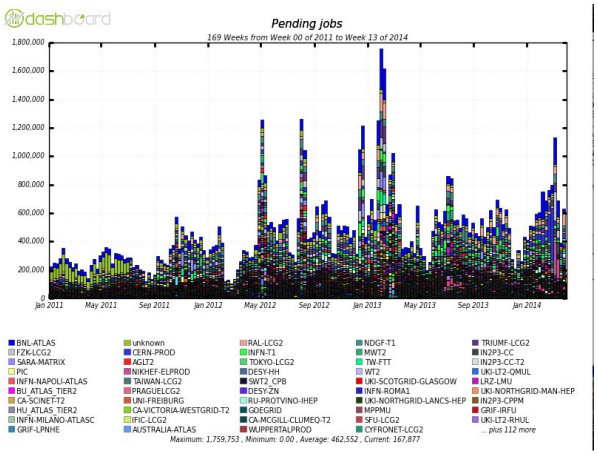


Pilot Abstraction: Overview

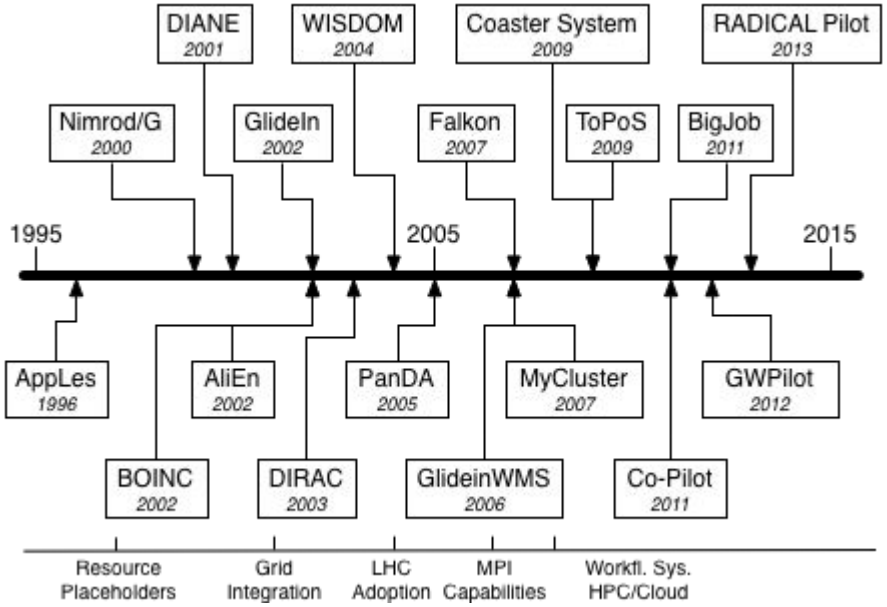
- Working definition:
  - “.. defined as an abstraction that generalizes the reoccurring concept of utilizing a placeholder job as a container for a set of compute tasks”
- Advantages of Pilot-Abstractions:
  - Decouples workload from resource management
  - Flexible Resource Management
    - Enables the fine-grained (ie “slicing and dicing”) of resources
    - Tighter temporal control and other advantages of application-level Scheduling (avoid limitations of system-level only scheduling)
  - Build higher-level frameworks without explicit resource management

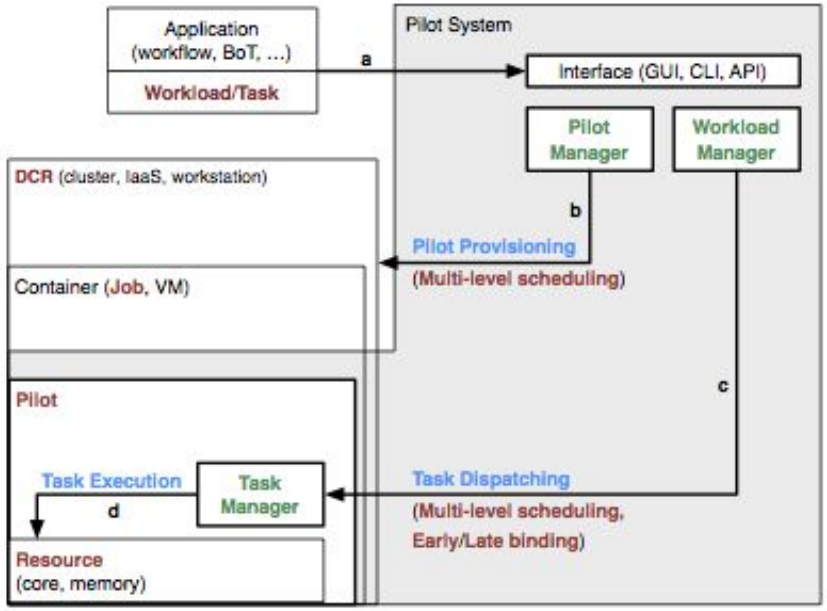
Pilot Jobs: Many Variations on a Theme





P* Model for Pilot Abstraction: Conceptual Model

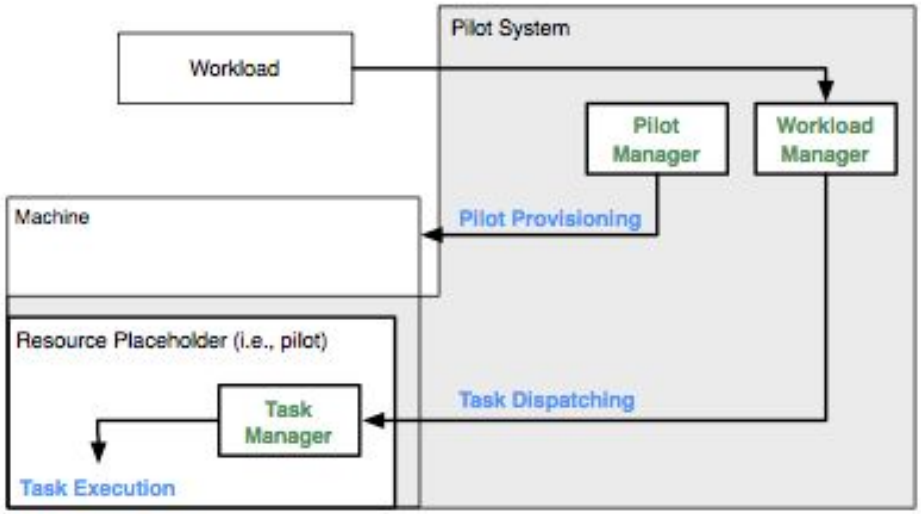




- P* Model of Pilot-Jobs
  - "P*: A Model of Pilot-Abstractions", *8th IEEE International Conference on e-Science 2012*
  - *A Comprehensive Perspective on Pilot-Jobs* http://arxiv.org/abs/1508.04180 *(2015)*

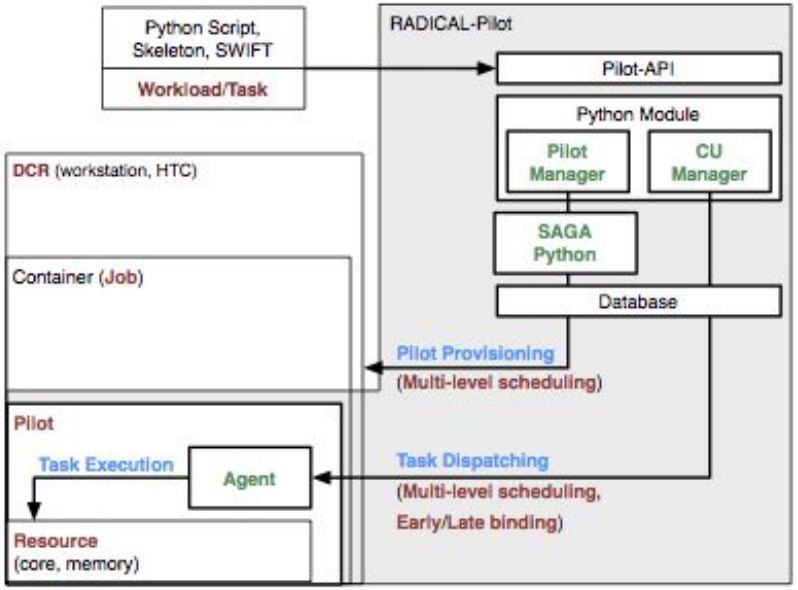
Pilot Abstraction: Logical components and functionality

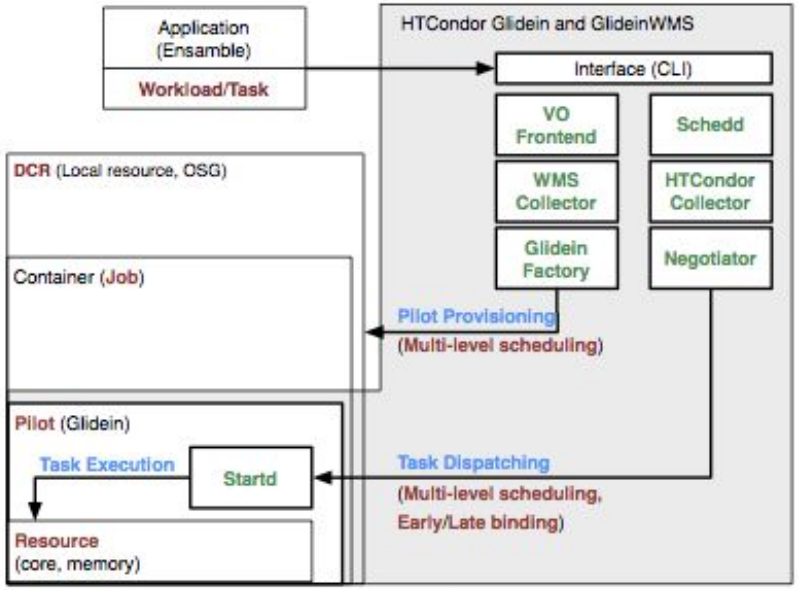
- Necessary Logical components:
  - Pilot Manager
  - Workload Manager
  - Task Executor/Manager
- Necessary functions:
  - Pilot provisioning
  - Task dispatching
  - Task execution

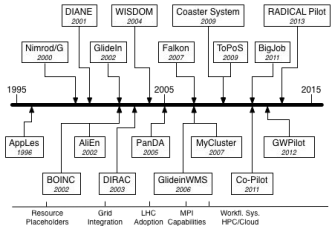


Diagrammatic representation of logical components capabilities.

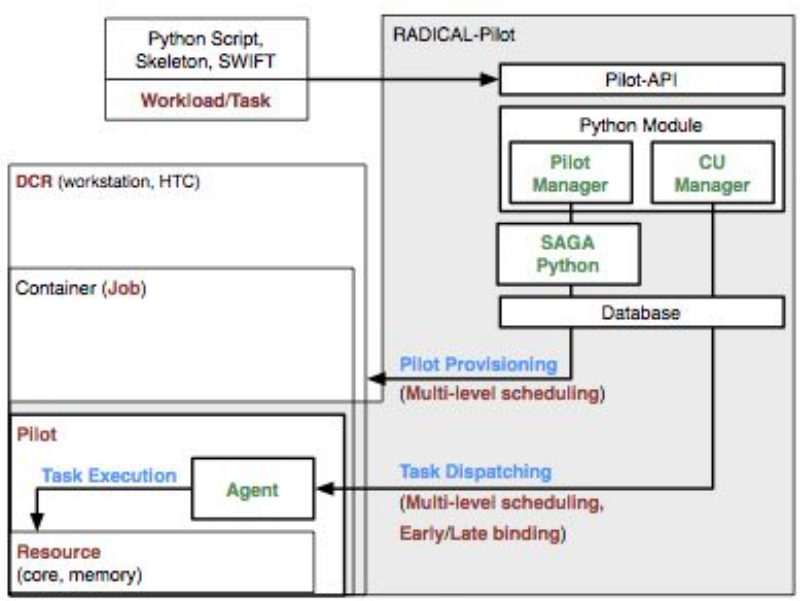
Common Components, Functionality and Vocabulary

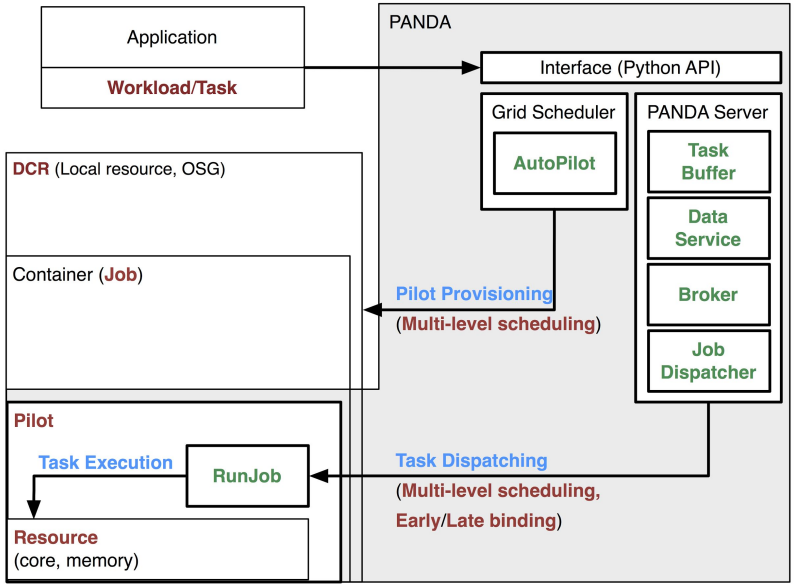


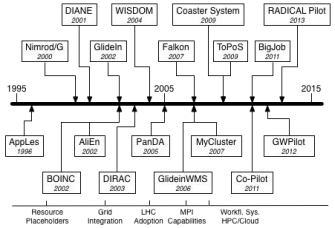




Common Components, Functionality and Vocabulary



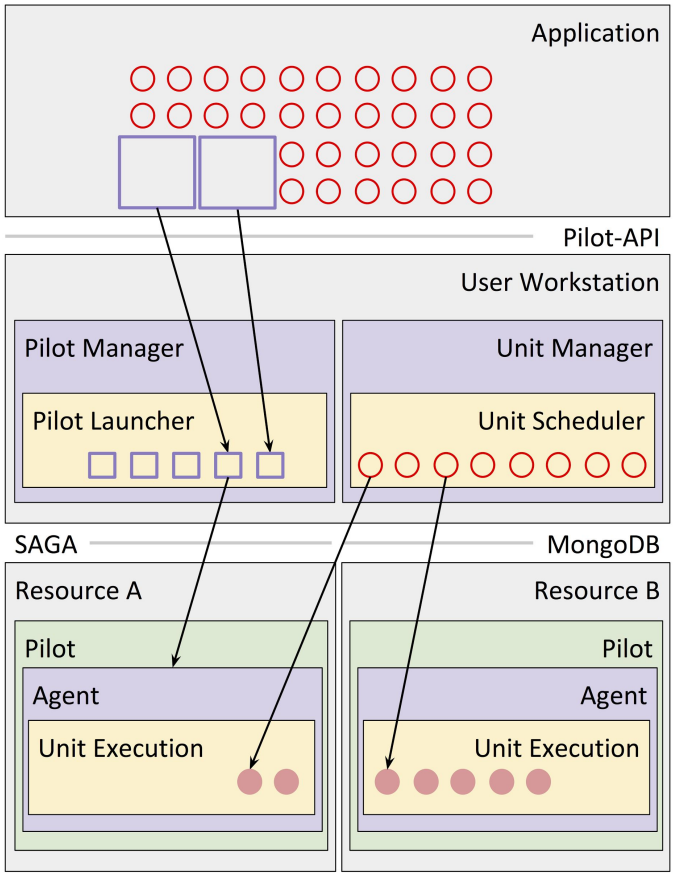




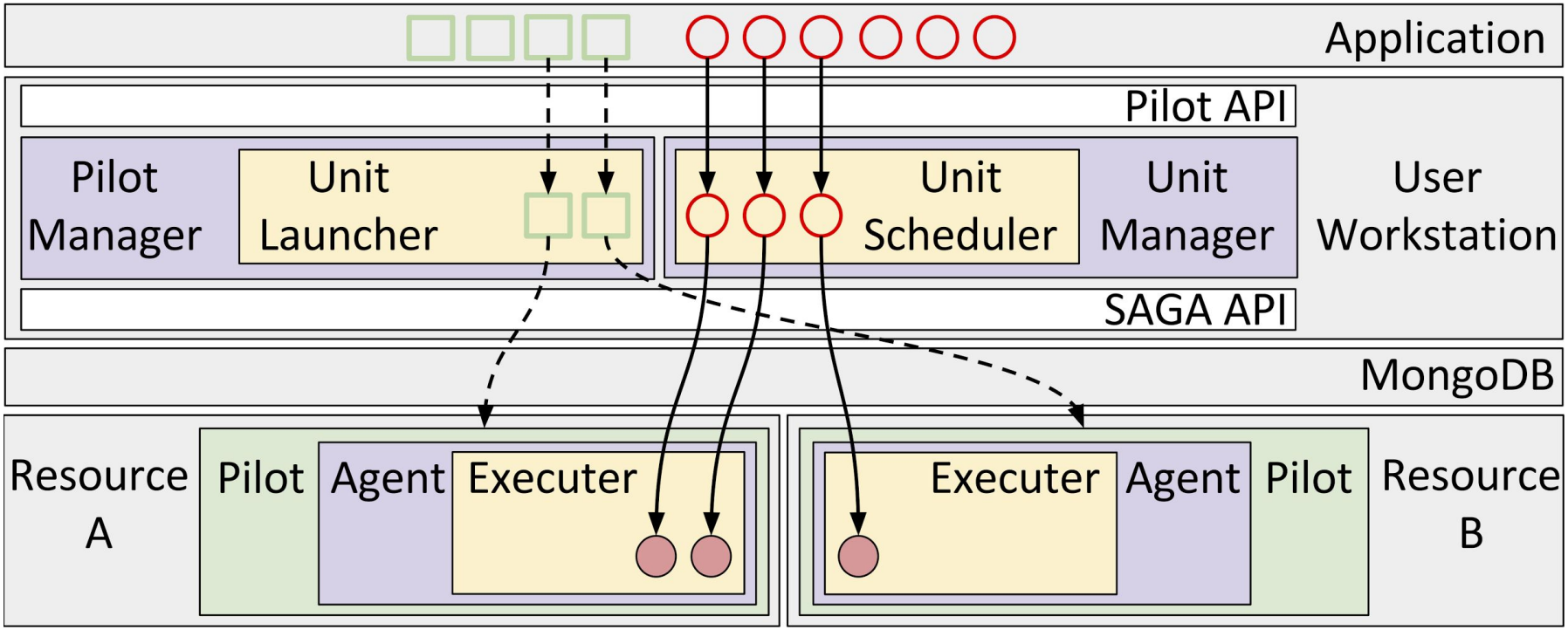
HPC Requirements / Goals

- Workload with *heterogeneous* tasks
  - Varying core count
  - Varying application kernels
  - MPI / non-MPI
- Dynamic workload with workload unknown a priori
  - Dynamic: Tasks (workload) and task relations
- Control over concurrency of tasks
  - Might be loosely coupled (e.g. replica exchange)
- Multiple dimensions of scalability
  - O(100k) concurrent tasks

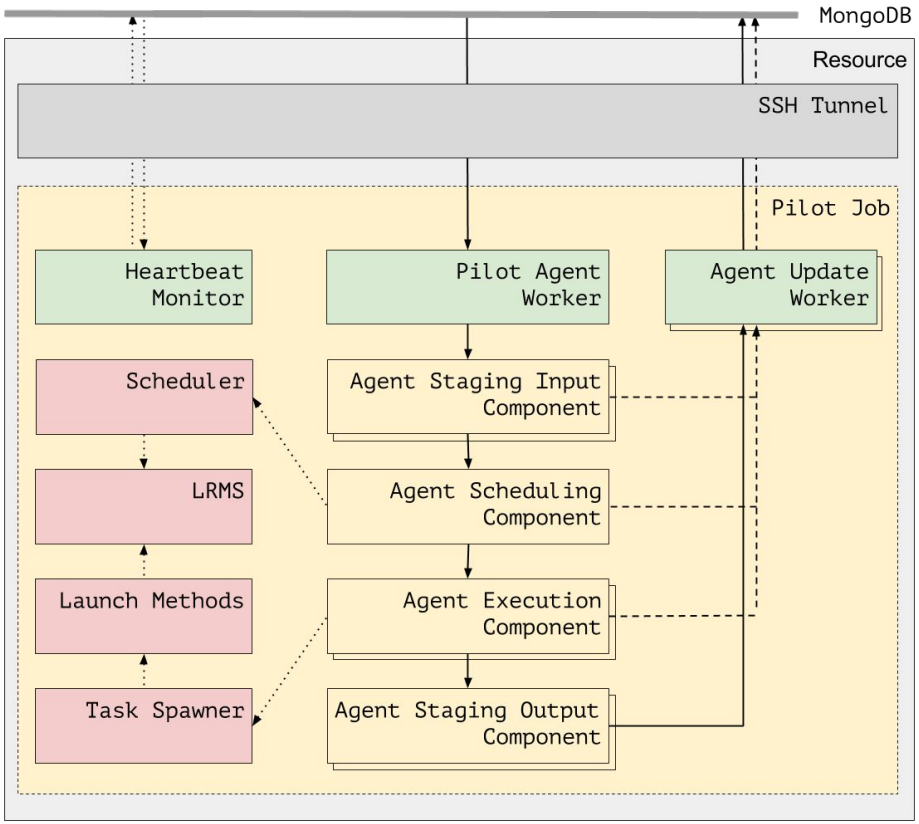
RADICAL-Pilot Architecture



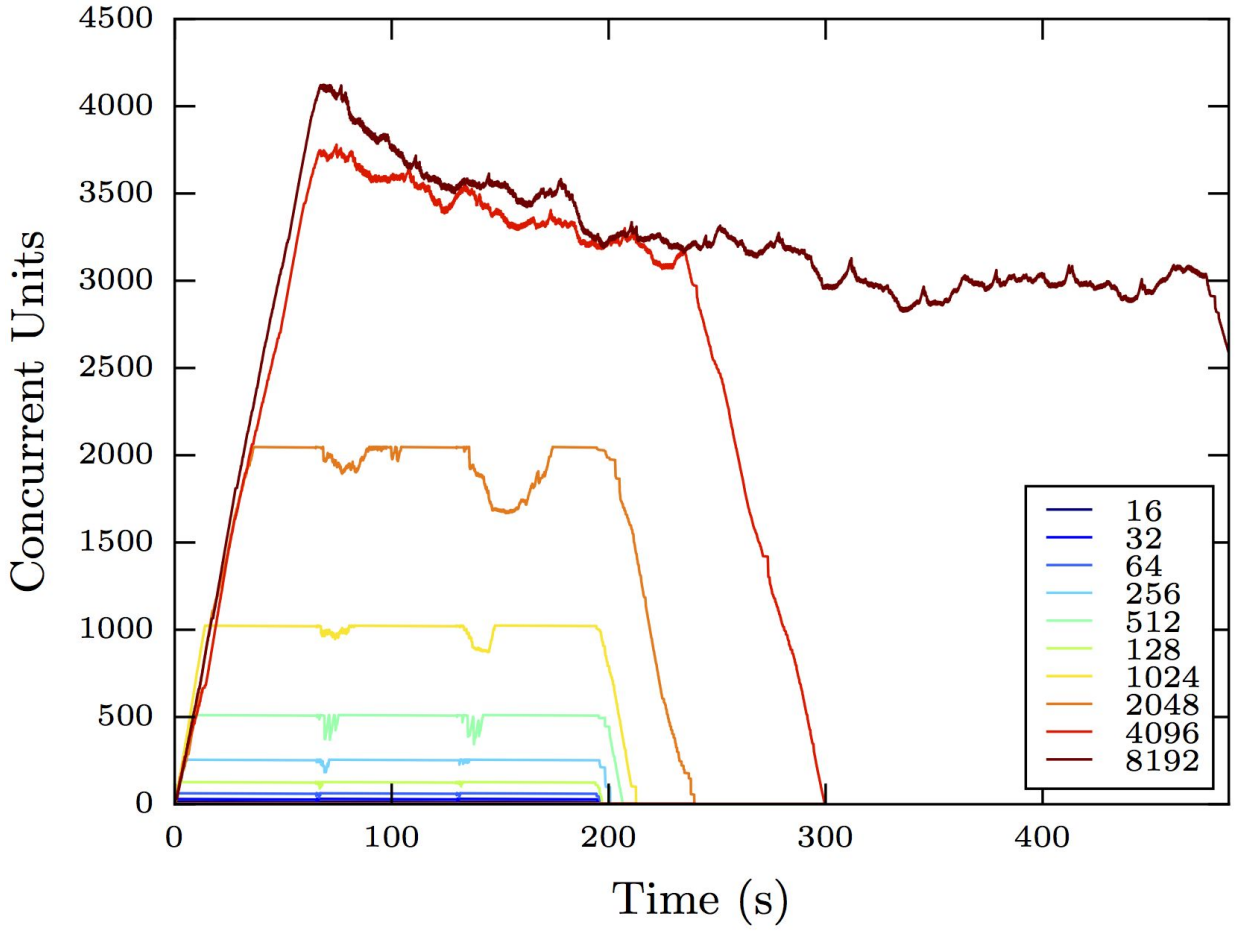
RADICAL-Pilot Architecture (2)



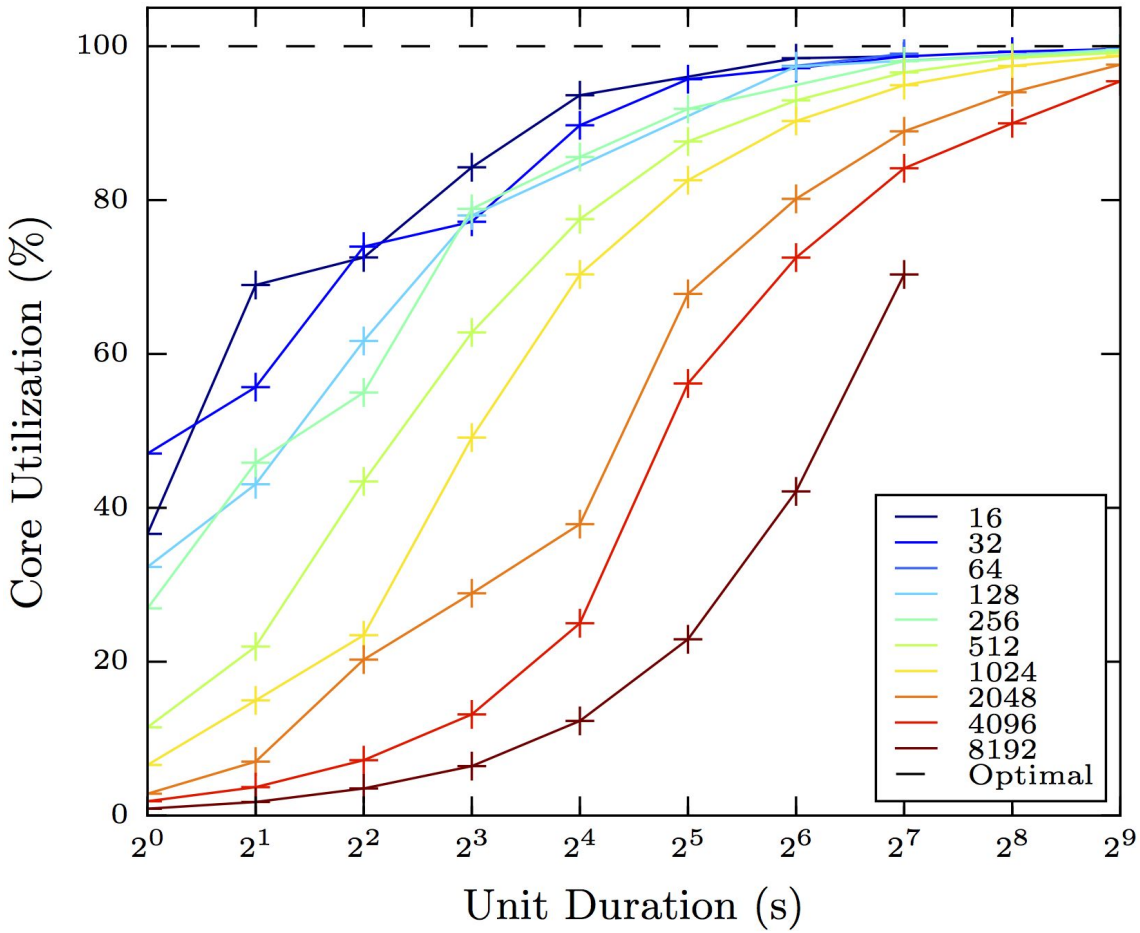
Agent Architecture



Agent Performance: Concurrent Units



Agent Performance: Resource Utilization



Challenges of O(100K) Concurrent Tasks

- Agent communication layer (ZMQ) has limited throughput
  - limit is not yet reached
  - bulk messages (is implemented now)
  - separate message channels
  - code optimization
- Agent scheduler (node placement) does not scale well with number of cores
  - bulk operations (schedule bag of tasks at once)
  - good scheduling algorithms and implementations exist
  - code optimization
- Collecting complete jobs is just as hard as spawning new ones
  - decouple
- Interaction with DB and client side has limited scalability
  - replace with proper messaging protocol (also ZMQ?)

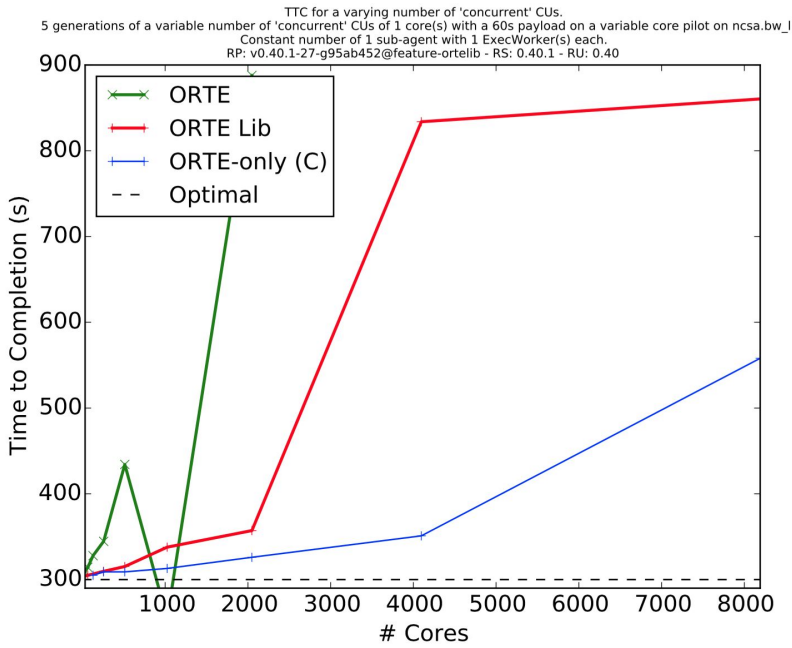
So why is it a challenge...

… to build “Scalable, Extensible and Flexible” tools?

- Every high-performance computing system/supercomputing environment is different, if not unique:
  - **Interoperability** across distinct interfaces and semantics: Batch queue systems, data access/movement etc.
  - **Portability**: Software environments, Simulation Kernel, Python, compilers, libraries, etc.
- Generality is often orthogonal to performance:
  - **Extensibility** + **Flexibility**: You can either make it general-purpose or high-performant.
  - **Reality**: find the sweet spot between general-purpose and performance

TTC Scalability: Variability

BW



!=

Stampede